Architectures des systèmes informatiques

Support de cours LSI 2

D.Chillet

Daniel.Chillet@enssat.fr

http://r2d2.enssat.fr

Plan

- **Introduction**
  - objectifs du cours
  - historique des systèmes informatiques
  - machines algorithmiques : classification
- **Machines algorithmiques**
  - machines dédiées ou programmable
  - modélisation par automate à états finis
  - modèle, matérialisation, implantation d'algorithmes
  - hiérarchisation des machines
  - notion de langage associé à un niveau de la hiérarchie

Plan (suite)

- **Modèle Von Neumann**
  - la machine et le cycle Von Neumann
  - exécution d'instructions
- **Le concept RISC**
  - historique, observations
  - concepts
  - évolution des processeurs RISC
  - processeurs CISC -- RISC
- **Hiérarchie mémoire**
  - mémoire cache

Architecture des systèmes informatiques

- **Fonctionnement Pipeline**
  - principe
  - objectifs
  - problèmes

- **Mécanismes permettant d'augmenter les performances**
  - allongement des pipelines
  - renommage des registres
  - exécution dans le désordre
  - etc etc

- **Panorama de processeurs**
  - Pentium 4
  - Itanium
  - Power PC
  - etc

INTRODUCTION

Introduction

❑ Historique : les bases

➢ le calcul a pour ancêtre : (les machines à calculer)

- ♦ la manipulation de cailloux : *calculus*
- ♦ les cordelettes à nœuds
- ♦ le boulier occidentales (environ 500 av J.C) :
  - pratique, "rapide", mais ne reporte pas la retenue
- ♦ règle à calculer

➢ la cryptographie et le codage ont pour ancêtres :

- ♦ remplacement des lettres par leurs numéros
  - utilisé par la cours d'Angleterre : (1561-1626)
  - les lettres sont codés de la façon suivante :
    - A = aaaaa B = aaaab C = aaaba D = aaabb
    - E = aabaa F = aabab G = aabba H = aabbb
  - exemple : Honni soit qui mal y pense : codage de chaque lettre :
    - Honni = aabbb abbab abbaa abbaa abaaa
  - on prend un texte quelconque, dans lequel on écrit en gras les lettres qui correspondent aux lettres b et en normal les lettres qui correspondent aux a :
    - Il **deviendra un** ser**pe**nt n**o**ir

Introduction

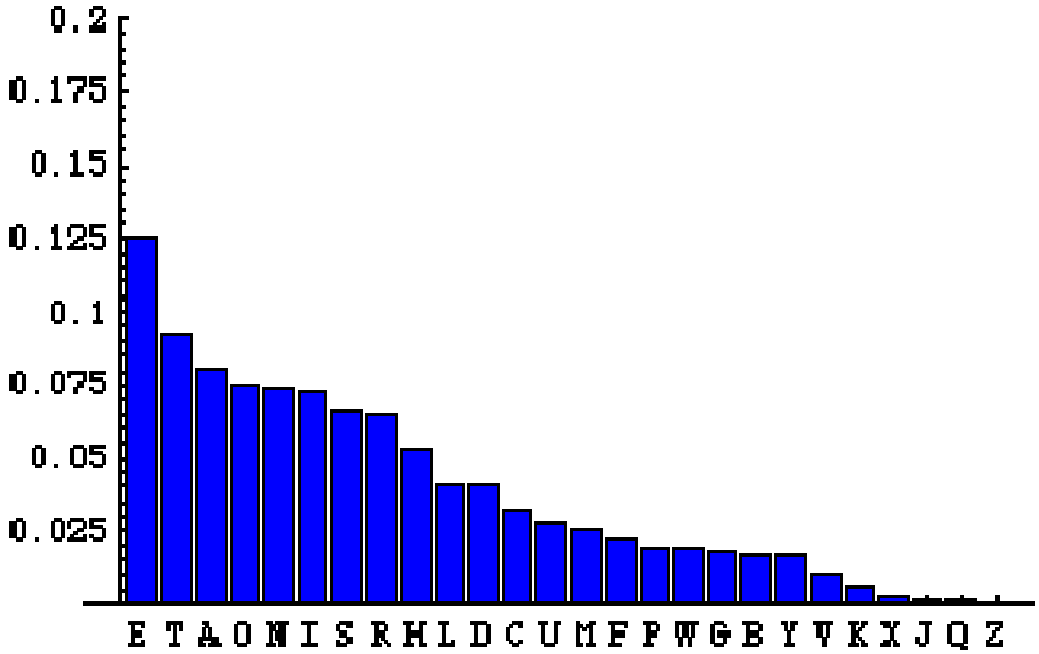
♦ Morse invente le télégraphe et le codage associé :

– les lettres sont transmises par des séries de points et de traits

– représentation binaire de symboles :

- la notation est simple
- les règles opératoires sont simples
- les lettres les plus fréquentes ont un codage le plus petit possible

Fréquence d'apparition des lettres
de l'alphabet dans les textes anglais



A	.-	N	-.	0	-----
B	-...	O	---	1	.----
C	-.-.	P	.--.	2	..---
D	-..	Q	--.-	3	...--
E	.	R	.-.	4	....-
F	..-.	S	...	5	.....
G	--.	T	-	6	-....
H	....	U	..-	7	--...
I	..	V	...-	8	---..
J	.---	W	.--	9	----.
K	-.-	X	-..-	.	.-.-.-
L	.-..	Y	-.--	,	--..--
M	--	Z	--..	?	..--..

Erreur	.....
Debut	-.-.-
Fin	.-.-.

Introduction

➢notion de programme :

♦ enregistrement préalable d'une suite d'opérations :

– orgue barbarie, pianos mécaniques, boites à musique

– portage vers les activités industrielles :

- commande de métier à tisser, utilisation de bandes perforées (fin 18 ième siècle) le motif à tisser est "codé" sur la bande qui tourne en continue. On change le motif en changeant la bande
- Jacquard est le premier à utiliser la notion de programme

➢mot "ordinateur" :

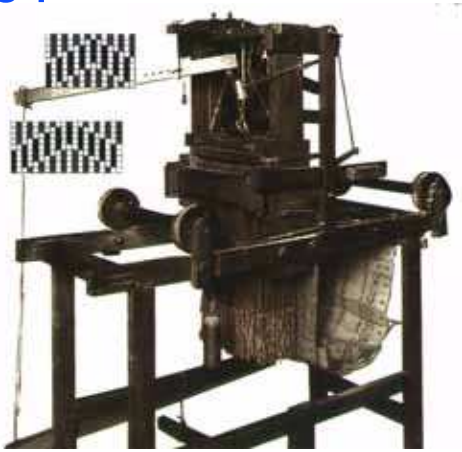
♦ proposition de Jacques Perret à IBM France en 1956

➢mot informatique :

♦ créé par Philippe Dreyfus en 1964 à partir des mots :

– **inform**ation

– autom**atique**

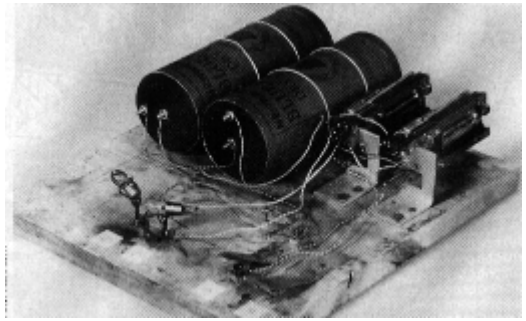


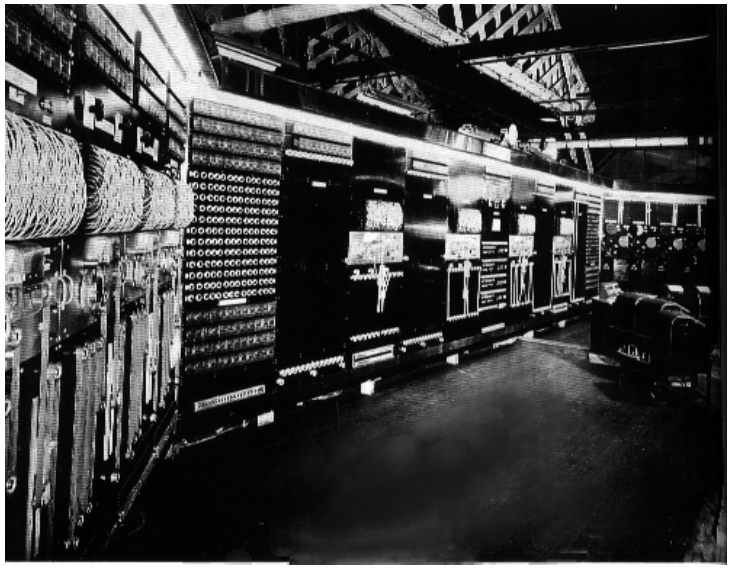
Introduction

- notion de logiciel :
  - naissance à partir de l'apparition des ordinateurs programmables
- Ordinateurs construits :
  - mécanique :
    - Konrad Zuse : machine Z1 (1936), machine entièrement mécanique, utilisation d'un système binaire pour la représentation des nombres, nombres codées en virgule flottante, la machine recevait le programme sur un film 35 mm avec des trous fait à la main
  - électromécanique :
    - Konrad Zuse : machine Z2, Z3 et Z4 : le Z3 (électromécanique) est considéré comme le premier ordinateur avec programme fournit par bande perforée, traitement des nombres en virgule flottante avec 14 bits pour la mantisse et 7 bits pour l'exposant. Machine réalisant 3 ou 4 additions par secondes. La machine ne pouvait pas réaliser de branchements conditionnels

Introduction

– Stibitz (1904 - 1995) : réalisa un additionneur binaire électromécanique à l'aide de relais de téléphone,



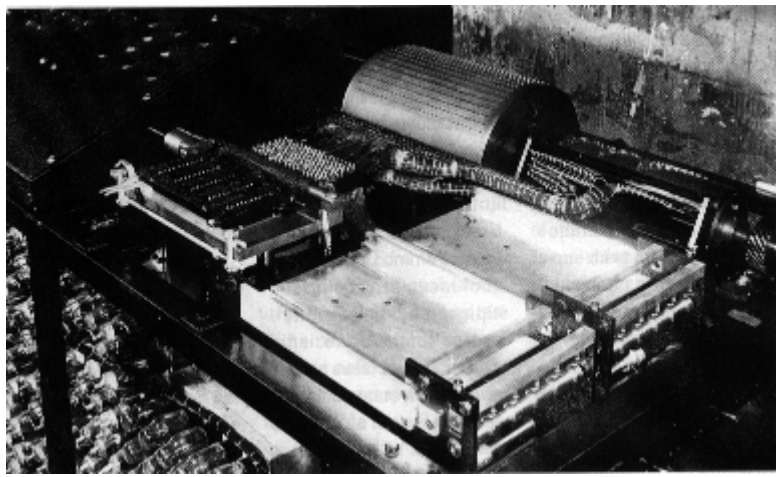


– Les Bell Labs utilisèrent cette idée pour construire une calculatrice de nombres complexes (opérations : addition, soustraction, multiplication, division). Le code binaire était redondant si bien que les machines (peu fiable) étaient capable de détecter des erreurs de calculs

– MARK 1 : machine de Babbage électromécanique, cycle de 6 secondes,

Introduction

- électronique :
  - Stibitz : le Z22 : machine entièrement électronique
  - Atanasoff (1903 1995) : machine ABC (Atanasoff Berry Computer), machine utilisant le code binaire, opérations d'addition et de soustraction, fréquence d'exécution de 60 hertz (60 cycles d'exécution par seconde). Cette machine servit de modèle pour la construction de l'ENIAC.



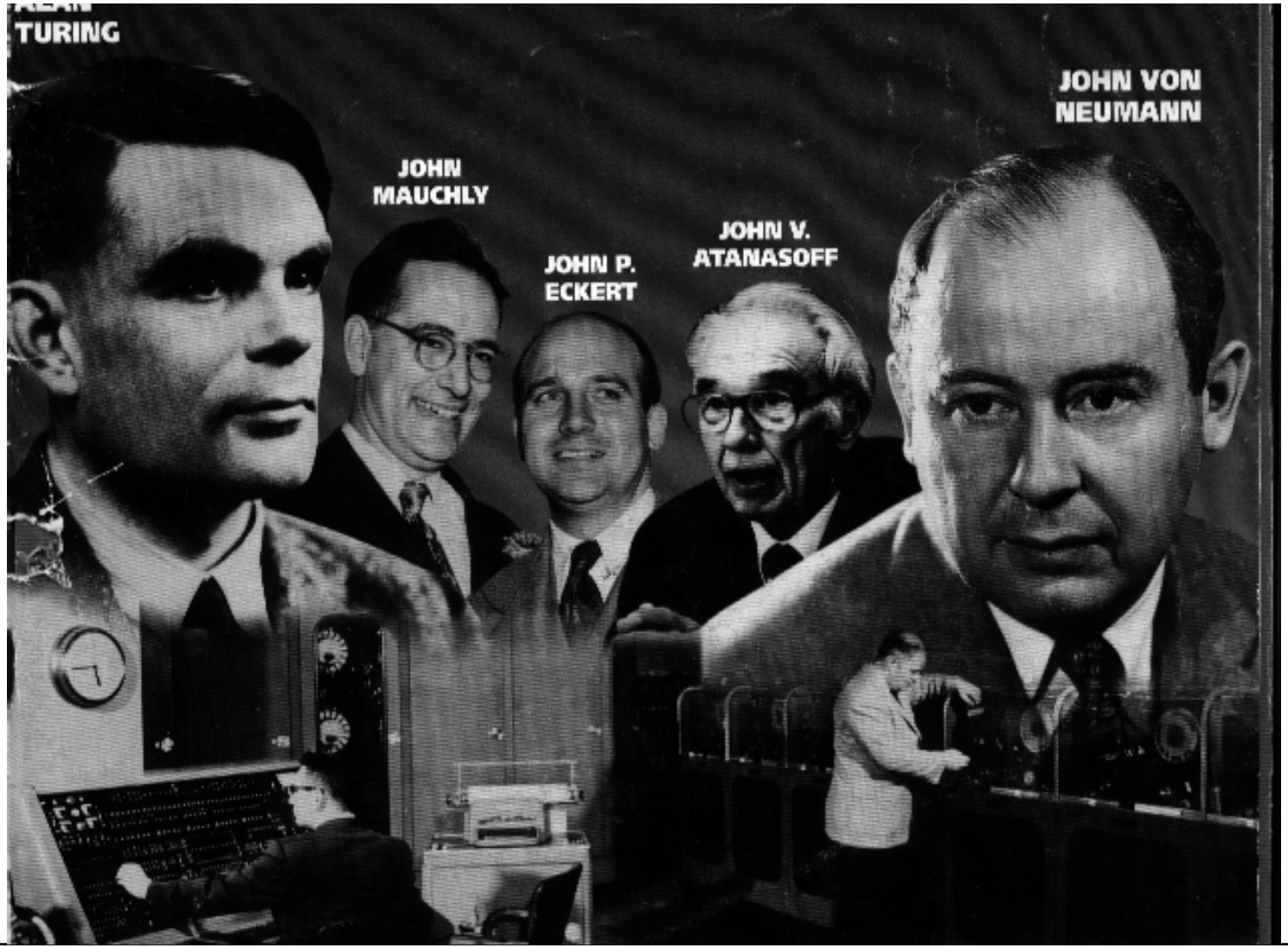
Introduction

– ENIAC :

- 1943 1944, vitesse de travail 200 000 Hz ou étapes par secondes, la machine travaillait avec la base 10 et non la base 2, il était composé d'un multiplieur, d'un diviseur, d'une table de fonction , sa programmation consistait à connecter les différentes unités entres elles, la notion de programmation consistait alors en la réalisation d'un câblage long et non modifiable par la machine elle même. L'ENIAC était sujet à **une panne tous les 3 jours environ**. Il calcul 5000 additions ou 350 multiplications par seconde. Cette machine fut utilisée pour calculer les 70 premières décimale de PI en 70 heures de calcul. La machine fonctionna 10 ans.

Introduction

❑ Quelques grands noms



Introduction

❑ Historique, quelques grands noms :

➢Blaise Pascal : 1623 - 1662 :

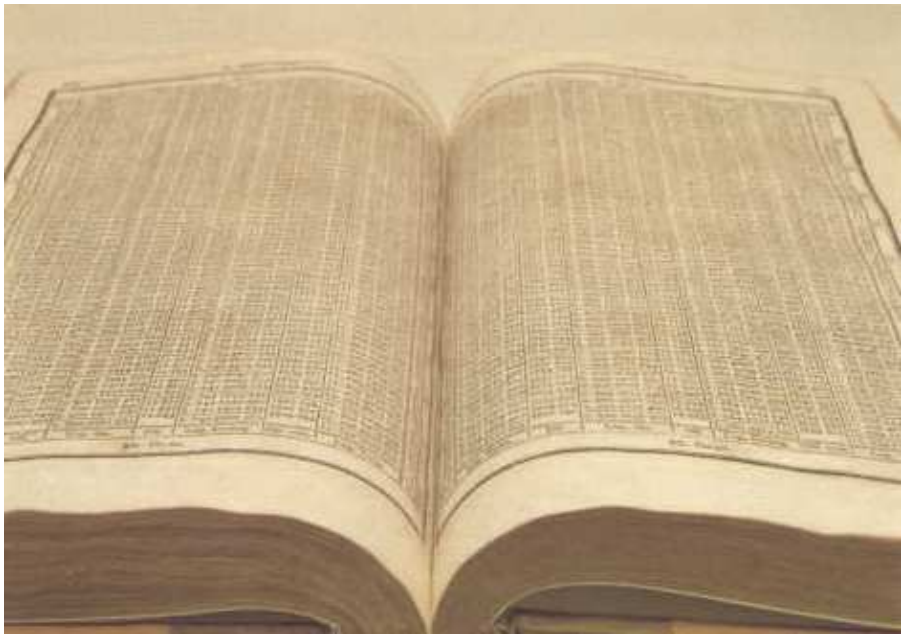
♦ machine Pascaline :

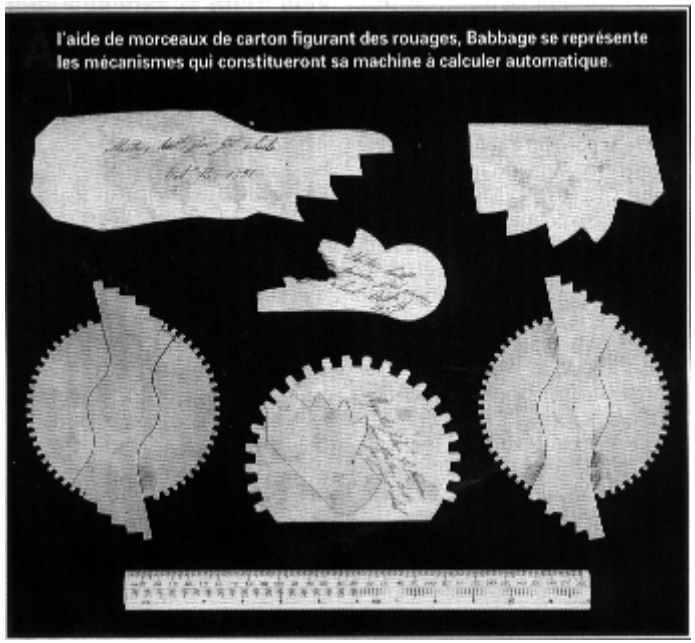
- mise au point à l'âge de 19 ans
- utilise des roues dentées qui permettent de réaliser des additions et soustractions
- les roues comportent 10 positions (de 0 à 9)
- à chaque fois qu'une roue passe de 9 à 0 elle entraîne la roue immédiatement à gauche
- la soustraction est réalisée grâce à la méthode des compléments arithmétiques :
  - s = a - b
  - s = a + ! b

Introduction

➢Babbage : 1792 - 1871

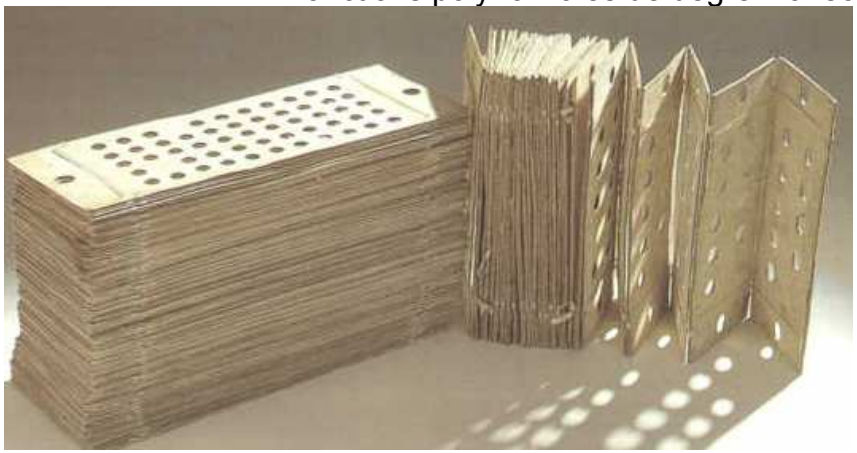
- ♦ prof de mathématique à Cambridge
- ♦ idée de départ :
  - – consistait à mécaniser les calculs des tables utilisées notamment à la navigation

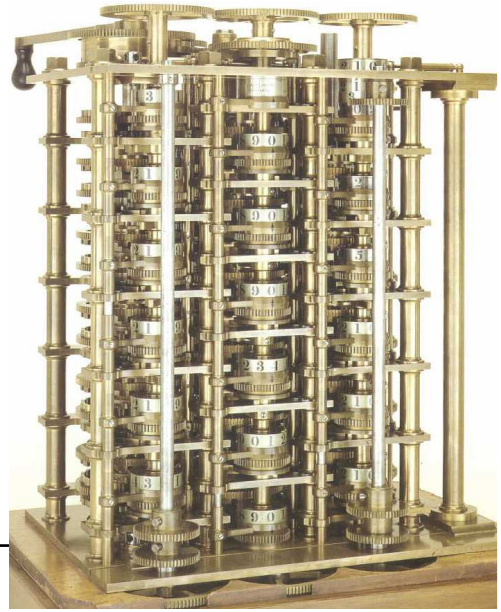




Introduction

- 1820 : machine à différences n°1 : objectifs : produire des tables mathématiques exactes
  - observation : les longs calculs sont souvent des répétitions d'opérations similaires (traitement d'une série d'opérations en ***séquence***)
  - idée de départ : concept développé par Jacquart
  - programme inscrits sur cartes perforées
  - calcul de fonctions polynomiales de degré 6 avec une précision à 16 chiffres
  - cette machine ne fut jamais achevée
  - elle donna toutefois naissance premier calculateur automatique (plus simple) qui mécanisait une règle mathématique
  - ensuite Babbage élabora les plans d'une machine plus perfectionnée : machine à différences n°2
    - fonctions polynomiales de degré 7 avec une précision de 31 chiffres

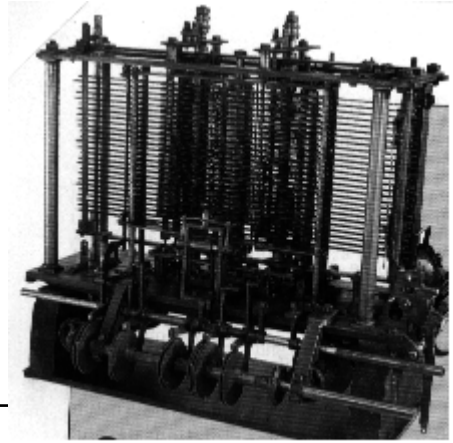




Introduction

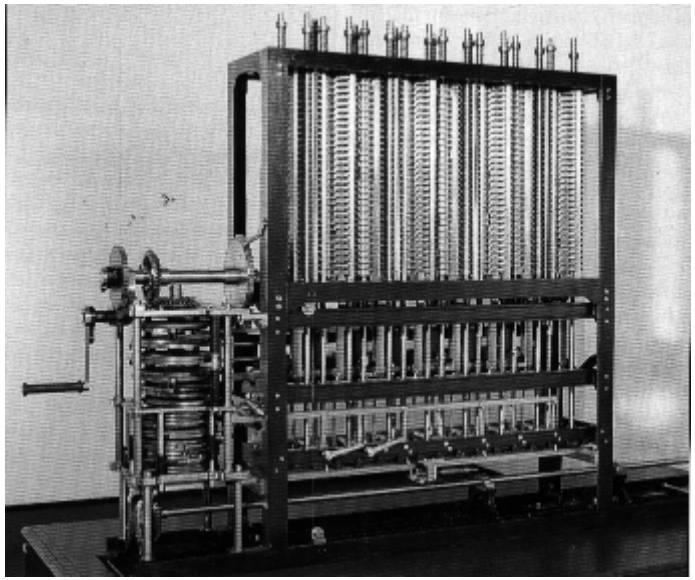
♦ 1833 : machine analytique :

- objectif : trouver la valeur de toute expression mathématique pouvant s'exprimer sous forme algorithmique (déroulement des opérations de façon ***conditionnelle***)
- 4 opérations de base : addition, soustraction, multiplication, division
- itérations et branchements conditionnels permettaient la programmation
- deux éléments principaux :
  - le moulin ====> le processeur
  - le magasin ===> la mémoire
- cette machine ne fut jamais construite
- il fut néanmoins encouragé par Ada Augusta King (1816 1852) qui se passionna pour cette machine et écrivit des programmes à partir des plans fournis par Babbage. Elle est considérée comme la première programmeuse et le langage Ada fut baptisée de son premier prénom.



Introduction

- A partir des plans originaux, la machine à différences n°2 fut construite, elle fonctionne parfaitement :
  - elle est exposée dans un musée à Londres



Introduction

➢ Boole : 1815 - 1864

- ♦ mathématicien
- ♦ Il conçoit l'algèbre qui porte son nom :
  - – algèbre basée sur 2 signes :
    - Oui = 1 ; Non = 0
  - – et sur 3 opérateurs :
    - ET ;
    - OU ;
    - PAS

A	B	ET	OU	PAS B
0	0	0	0	1
0	1	0	1	0
1	0	0	1	1
1	1	1	1	0

  - – algèbre capable d'assurer la totalité des calculs envisageables (toutes fonctions calculables)
- ♦ La logique devient une discipline mathématique

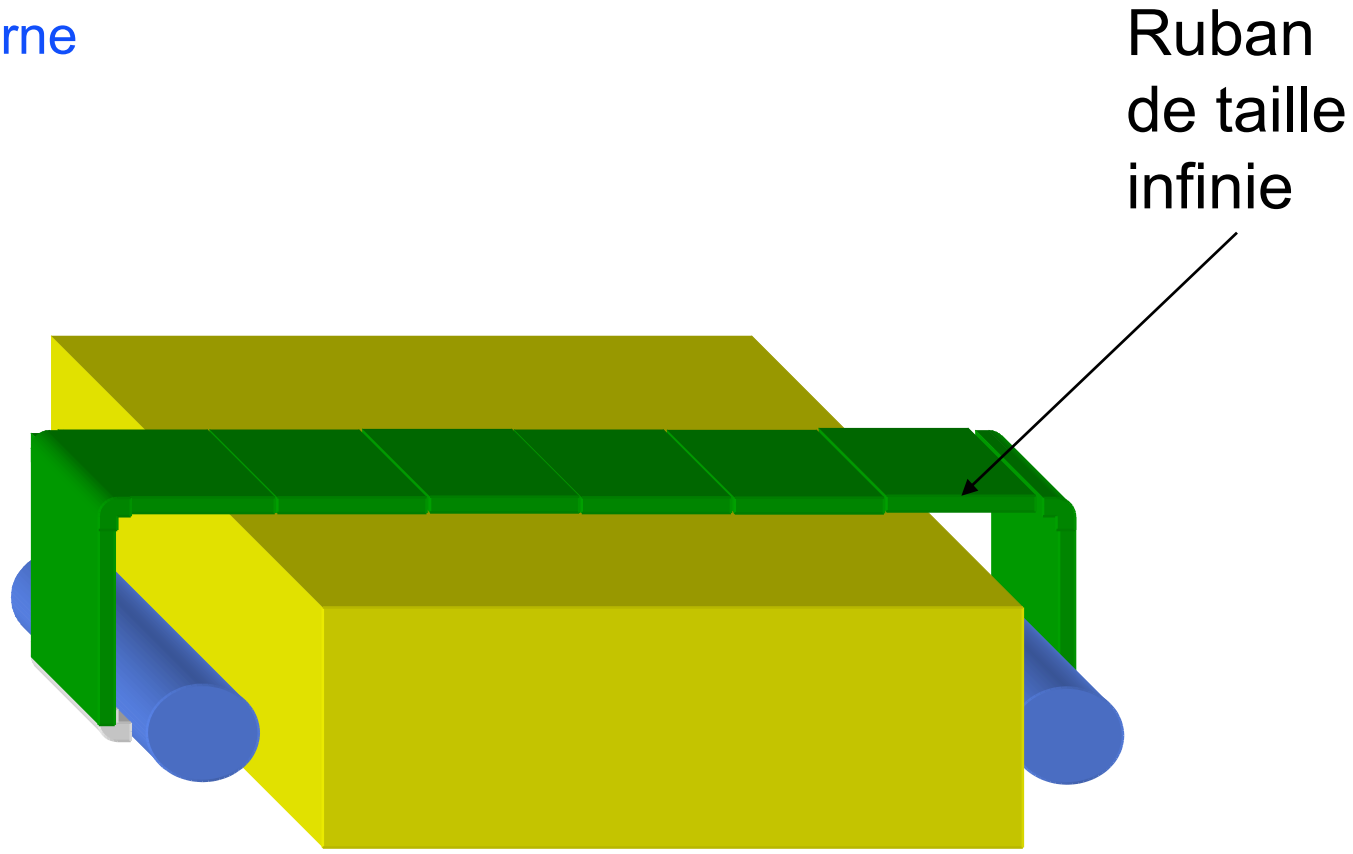
Introduction

➢ Turing : 1912 - 1954

- ♦ il formule le concept de calculabilité :
  - – tout processus logique peut être décomposé en une suite d'opérations élémentaires qui peuvent être exécutées par une machine
- ♦ il définit une machine pouvant travailler de façon automatique :
  - – notion de programmation par table d'instructions : automate
    - à partir du moment ou la configuration est définie par une table d'instructions (machine universelle) la machine exécute son travail
    - 1 calcul particulier ===> une table d'instructions particulière
- ♦ travaillera au déchiffrage des messages allemand, système Enigma
- ♦ il s'intéresse ensuite à la mécanisation des processus de la pensée
  - – réflexions à propos d'une machine capable de jouer aux échecs
- ♦ après la guerre, il travaille sur un projet de machine universelle :
  - – ACE (Automatic Computing Engine)
- ♦ en 1950 il publie un article dans une revue philosophique :
  - – *"L'ordinateur et l'intelligence",* prémisse de l'intelligence artificielle

Introduction

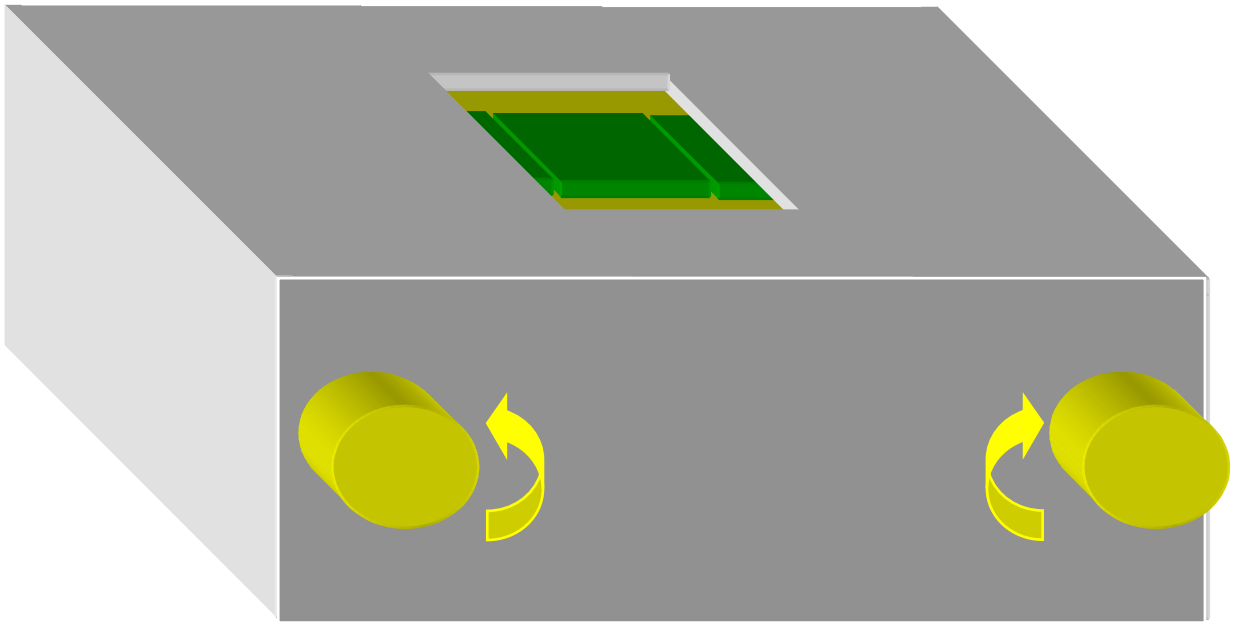
- Machine de Turing :
  - vue interne



Introduction

➢ Machine de Turing :

♦ vue externe



♦ fonctionnement

– la machine est dans un état
  - lecture du ruban
  - écriture sur le ruban
  - déplacement à gauche ou à droite
  - changement d'état

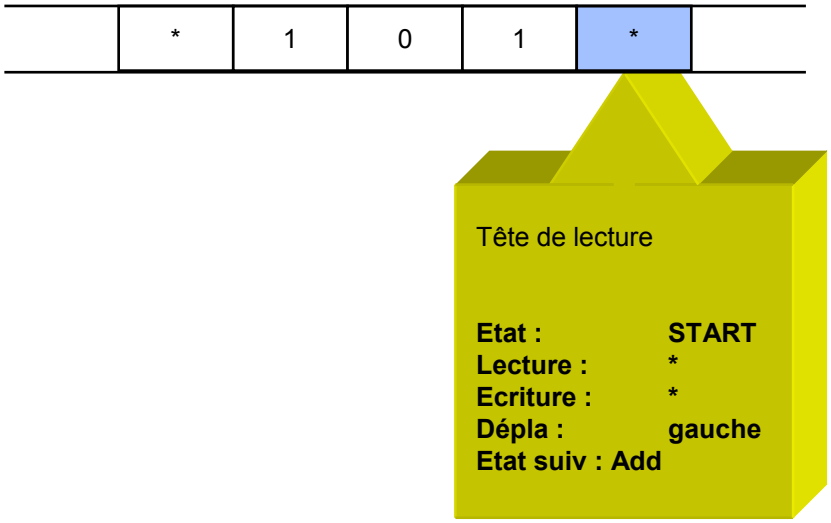
Introduction

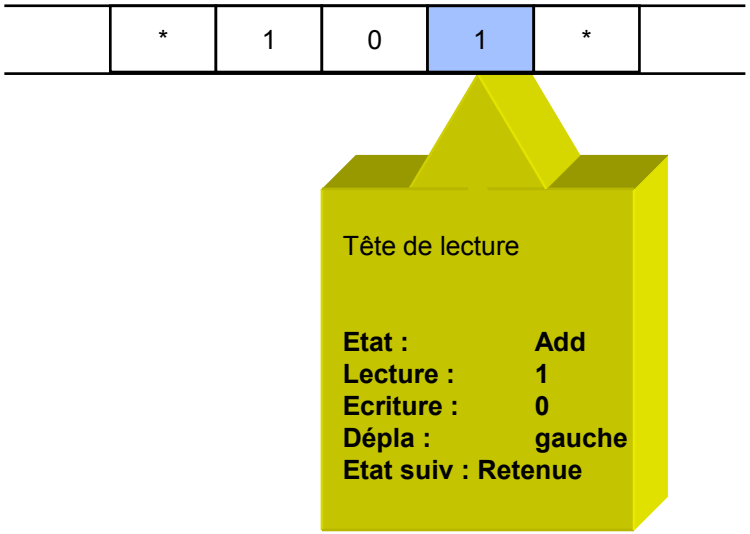
➢ Exemple de machine:

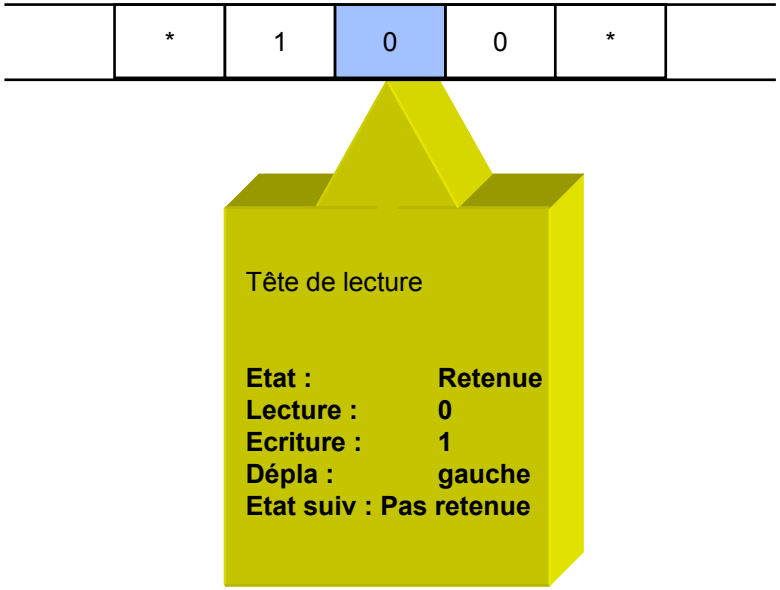
♦ incrémentation d'une valeur

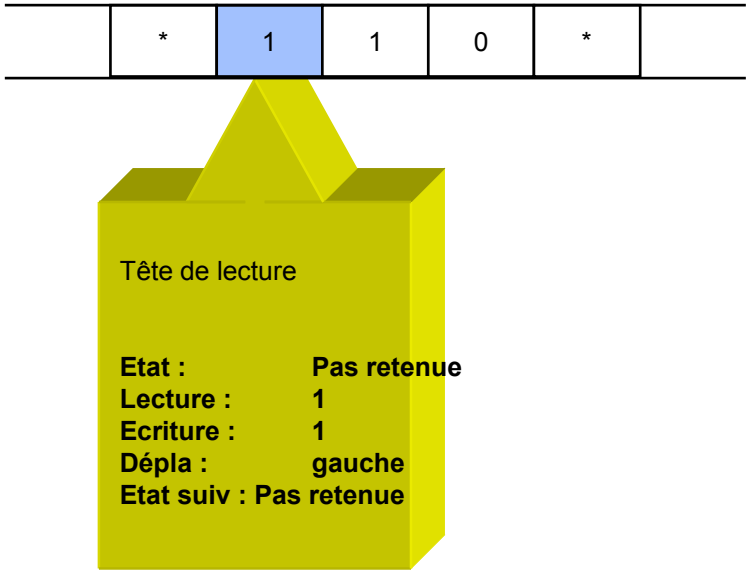
Etat courant	Contenu cellule courante	Contenu à écrire	Déplacement tête de lecture	Nouvel état
START	*	*	gauche	Addition
Addition	0	1	gauche	Pas de retenue
	1	0	gauche	Retenue
	*	*	droite	Retour
Retenue	0	1	gauche	Pas de retenue
	1	0	gauche	Retenue
	*	1	gauche	Dépassement
Pas de retenue	0	0	gauche	Pas de retenue
	1	1	gauche	Pas de retenue
	*	*	droite	Retour
Dépassement	?	*	droite	Retour
Retour	0	0	droite	Retour
	1	1	droite	Retour
	*	*	rien	STOP

Introduction

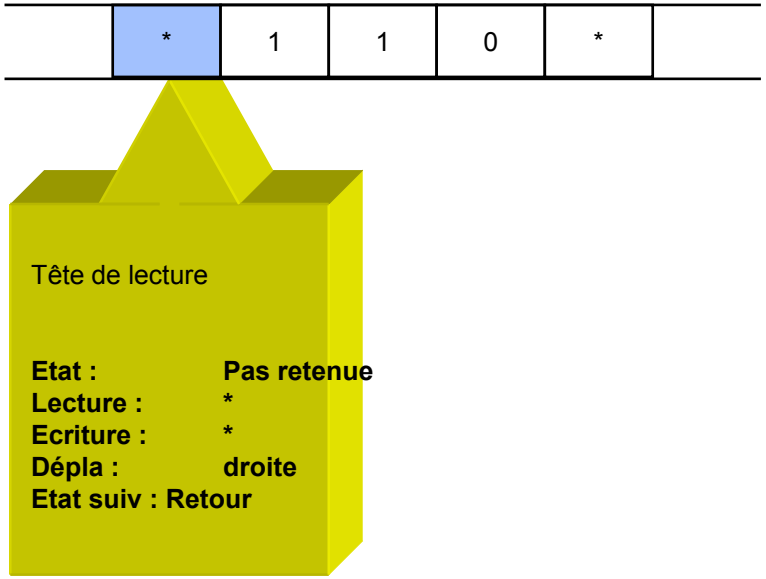


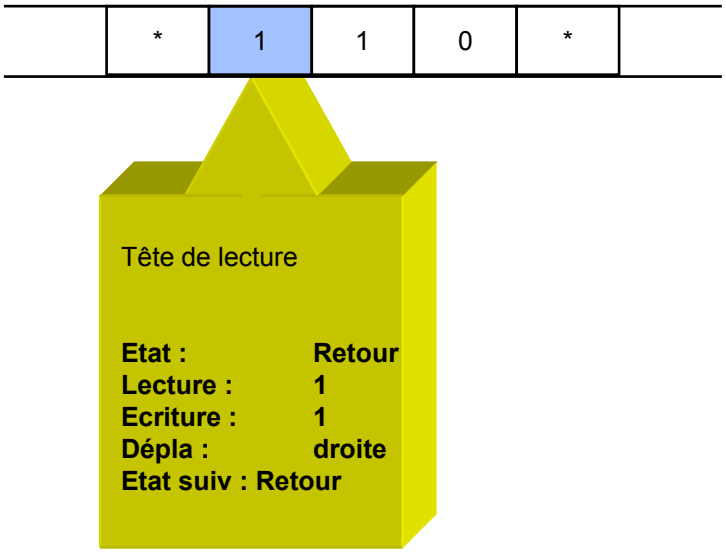


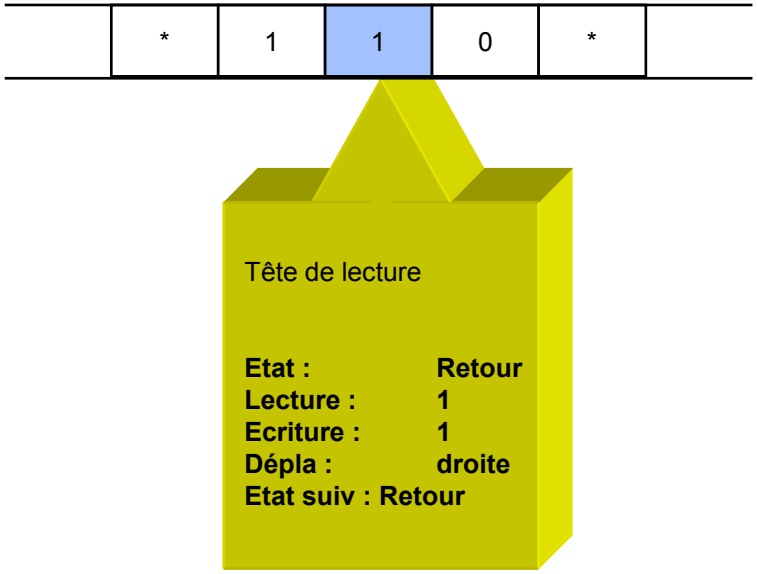


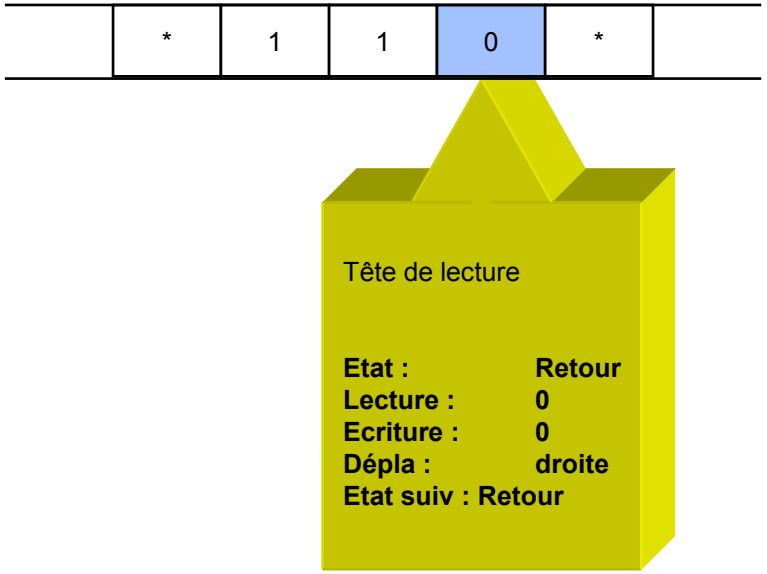


Introduction









Introduction

➢Von Neumann : 1903- 1957

♦ mathématicien Hongrois brillant :

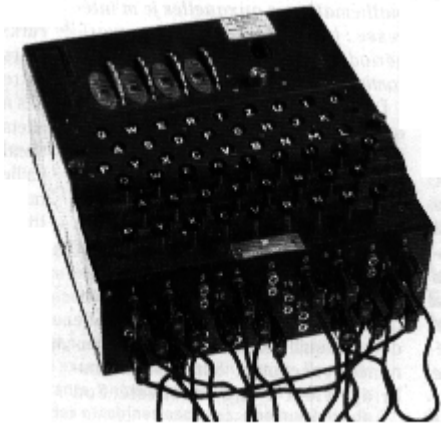
– il changea de nationalité en 1937 pour devenir américain

– il s'est intéressé aux disciplines suivantes :

• mathématiques purs, théories des ensembles, logique, théorie de la mesure, physique, théorie des opérateurs, statistique, analyse numérique, hydrodynamique, balistique, etc etc

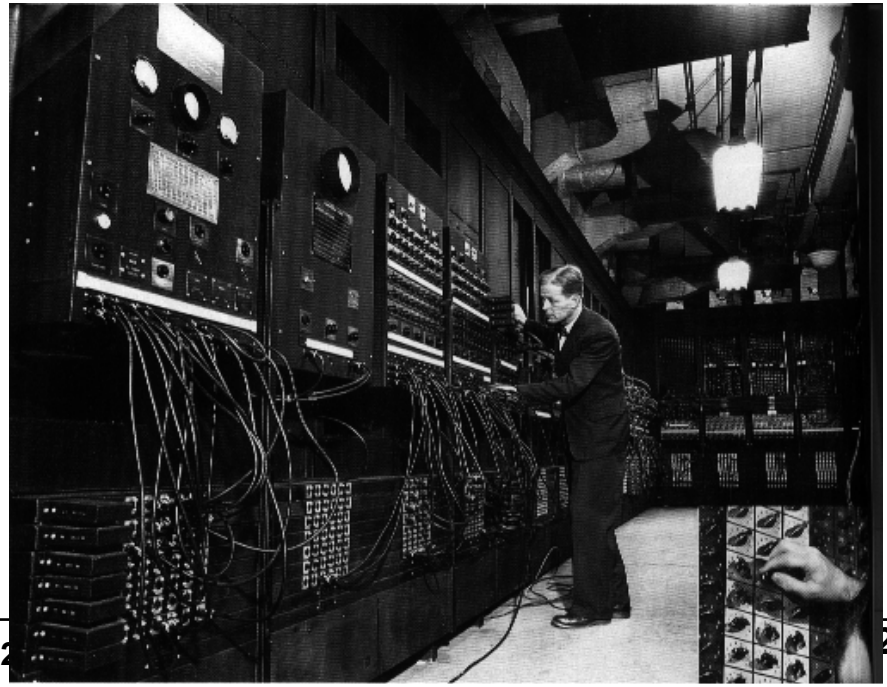
– il participe au développement de la bombe atomique (Los Alamos) :

• pour réaliser les calculs de détonation et d'implosion, il imagine une nouvelle machine composée d'une unité de calcul, d'une mémoire et d'un programme.



Introduction

- en 1944 il rejoint Eckert et Mauchly qui sont en train de mettre au point un calculateur électronique :
  - ENIAC : Electronic Numerical Integrator and Computer : sa première tâche consista à décoder les messages codés des Nazis
  - machine pouvant prendre des décisions en fonction d'un résultat de calcul précédent (résultat positif ou négatif)
- nouveau projet EDVAC : Electronic Discrete Variable Automatic Computer :
  - les instructions sont enregistrées dans une mémoire et le programme peut s'auto modifier, la machine passe du stade de calculateur à celui d'ordinateur
  - il publie un résumé du projet, le rapport (un pré rapport suite aux nombreuses discussions sur les limitations de l'ENIAC avec Eckert et Mauchly) est attribué (maladroitement) à Von Neumann, tous les honneurs lui reviennent (Eckert et Mauchly se sentent exclus du succès auquel ils ont largement participé).



Introduction

– Von Neumann montre qu'en enregistrant le programme dans la mémoire en même temps que les données à traiter, alors on obtient un automate qui a les propriétés de la machine de Turing :

machine algorithmique universelle

– en 1947 un rapport est diffusé par Von Neumann, Burks et Goldstine, il donne les prescriptions pour la réalisation d'un ordinateur :
  - le programme et les données sont enregistrés
  - le programme et les données sont dans une même mémoire découpés en cellules
  - cette mémoire est unique et banalisée
  - les contenus de la mémoire sont accessibles pour la lecture et l'écriture par localisation de leur contenant (c'est ce que l'on appellera plus tard les adresses).
  - la commande de la machine est séquentielle (exécution des opérations instruction après instruction) dans l'ordre des cellules mémoire (sauf en cas de branchements qui peuvent être impératif ou conditionné par des résultats de calcul précédent)
  - l'exécution de chaque instruction est achevée avant que la suivante ne commence
  - l'unité de traitement réalise un jeu d'opérations entre les registres

Un grand nombre de réalisation furent basées sur ce schéma que l'on appelle de "Von Neumann"

Introduction

❑ Les premières machines :

- ➢ défauts des premiers ordinateurs :
  - ♦ fiabilité incertaine :
    - – 1 panne tous les 3 jours pour l'Eniac
  - ♦ besoin de refroidissement :
    - – 5 chevaux pour la Mark 1 (3700 Watt !!)
  - ♦ encombrement certain :
    - – 160 m2 pour la Mark 1
- ➢ défauts des langages associés :
  - ♦ la taille de la mémoire étant très limitée, le code est compacté au maximum :
    - – grande époque du GOTO et de la réutilisation de morceaux de programmes
    - – pas de notion de structuration de programme, d'interface utilisateur et de convivialité

Introduction

❑ Les différentes générations

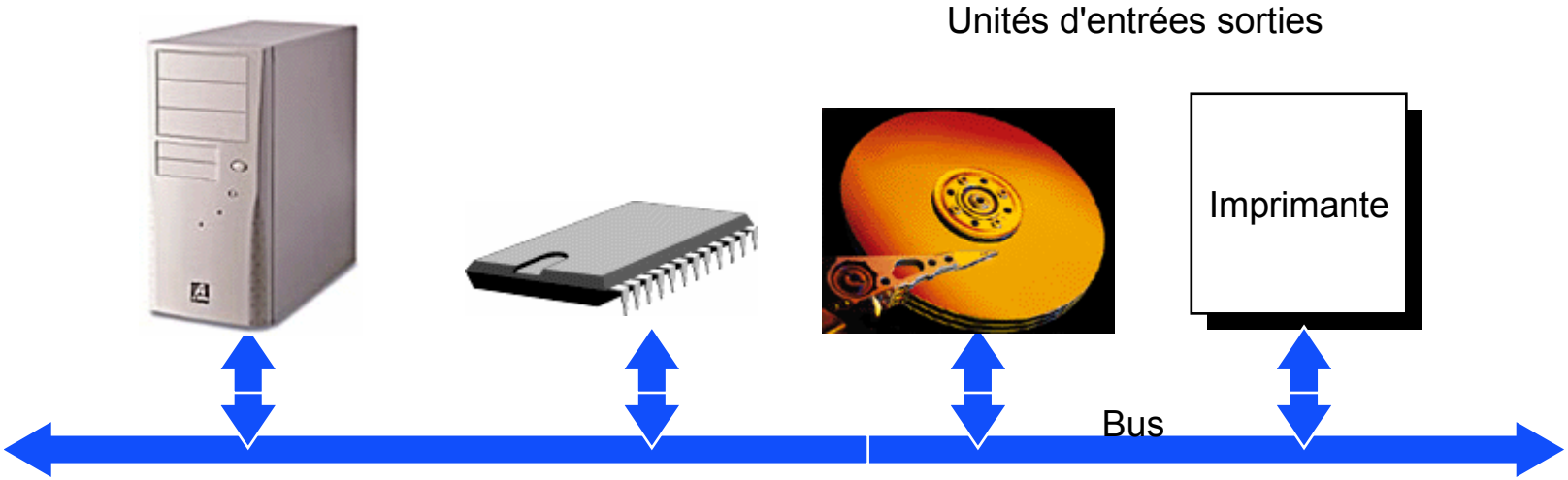
Générations	Dates	Technologies	Vitesses (op/s)
0	1642	Mécanique	quelques op/s
0'		Electromécanique	
0''	1956	Electronique	
1	1946-1957	Tubes à vide	40 000
2	1958-1964	Transistors	200 000
3	1965-1971	Petite et moyenne intégration Small and medium scale integration	1 000 000
4	1972-1977	Forte intégration Large scale integration (LSI)	10 000 000
5	1978-	Très forte intégration Very Large scale integration (VLSI)	100 000 000

Introduction

❑ Qu'est qu'un "ordinateur" ?

➢système composé de :

- ♦ processeur(s) : unité centrale
- ♦ mémoire(s) : stockage
- ♦ dispositif(s) d'entrées sorties : lien avec l'extérieur



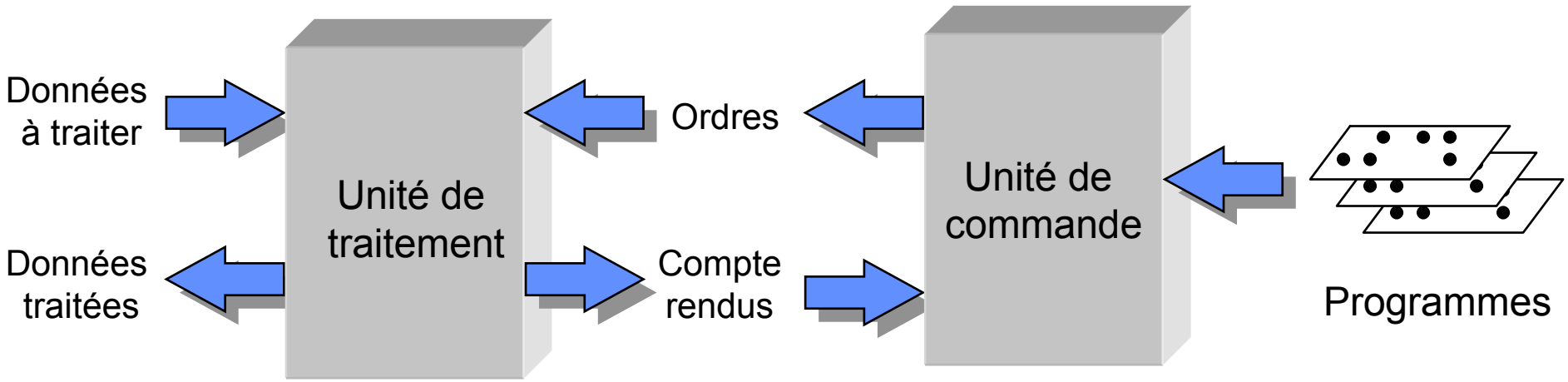
Introduction

➢L'unité centrale :

♦ machine qui exécute des ordres élémentaires de façon ***séquentielle*** sur des ***données*** : on parle de machine algorithmique

– c'est donc une machine qui manipule des données, opérations sur les données :
  - addition, multiplication
  - consommation des données à traiter, et production de données traitées

– notion d'unité de traitement et d'unité de commande
  - à la différence d'une simple calculatrice, l'ordinateur comporte une unité de commande capable de traiter un programme (interne ou externe)



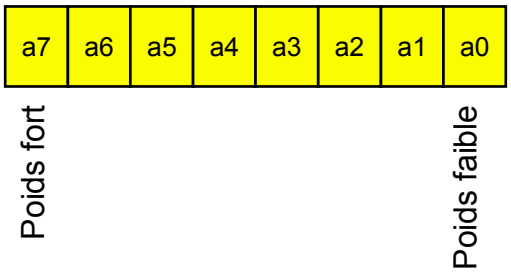
Introduction

- Unité de traitement :
  - elle effectue, sur les données qu'elle reçoit, les traitements commandés par l'unité de commande
  - au cours des traitements, les données temporaires sont stockées dans des mémoires locales internes, REGISTRES

- Unité de commande : on considère 2 types d'unités :
  - elle séquence les opérations sur l'unité de traitement
    - elle récupère les instructions depuis l'extérieur
    - elle décode les instructions
    - elle demande l'exécution de l'opération à l'unité de traitement
    - elle récupère les comptes rendus d'exécution

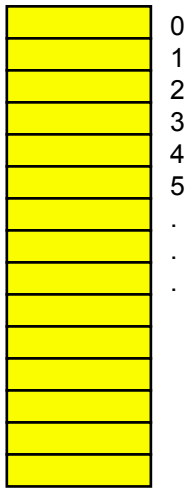
Introduction

- La mémoire :
  - stockage :
    - des données
    - des programmes
  - l'unité de stockage est le bit :
    - valeurs 0 ou 1
  - les données sont stockées dans des cellules mémoires
    - par exemple des cellules de 8 bits

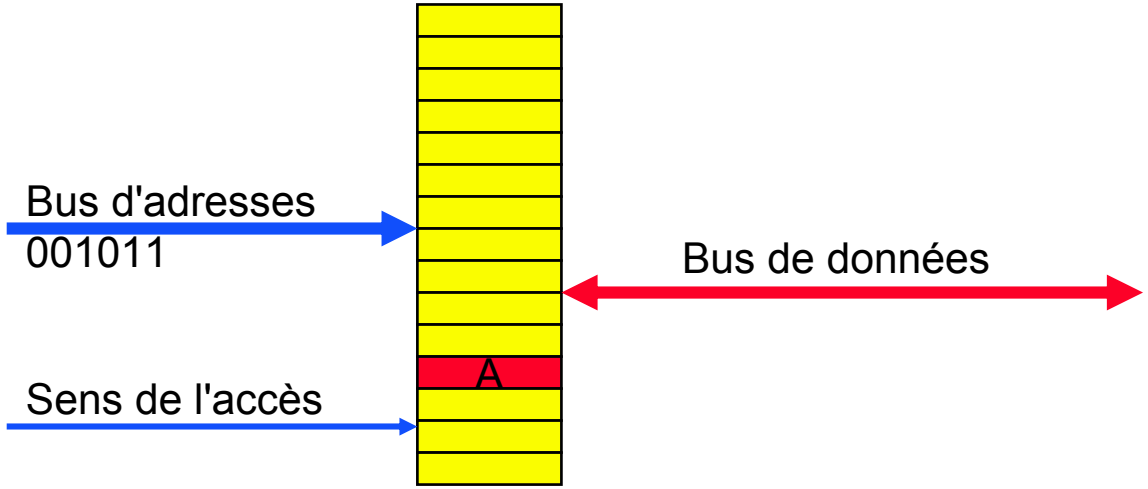


Introduction

♦ chaque cellule possède un numéro : *adresse*



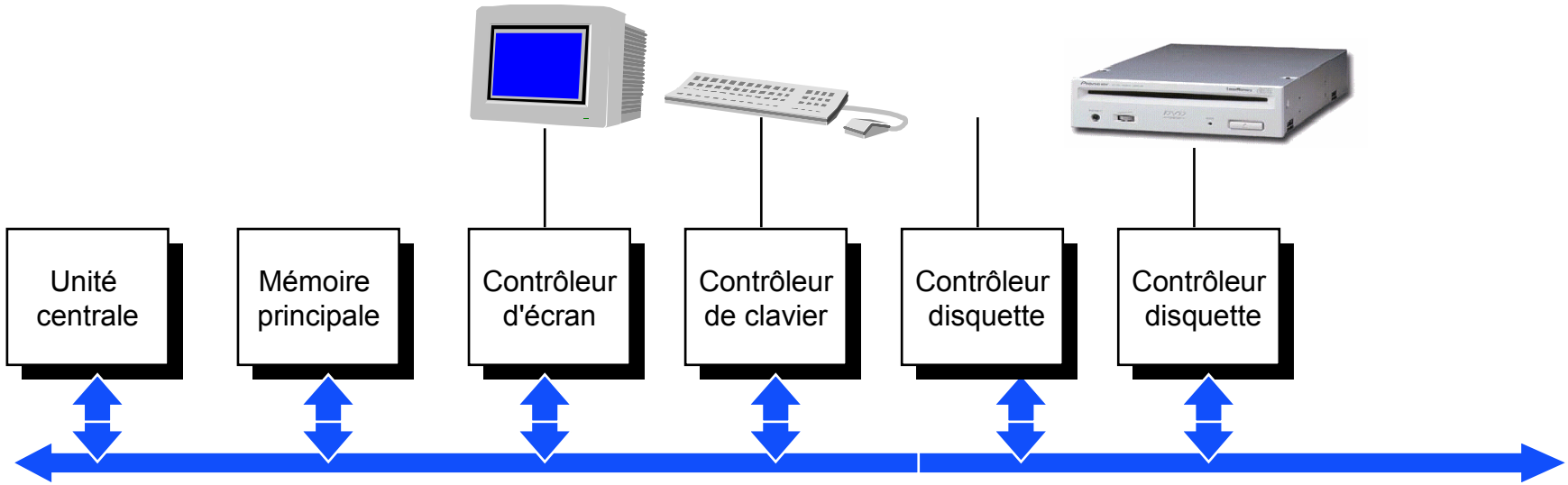
♦ le processeur manipule les données via leurs adresses



Introduction

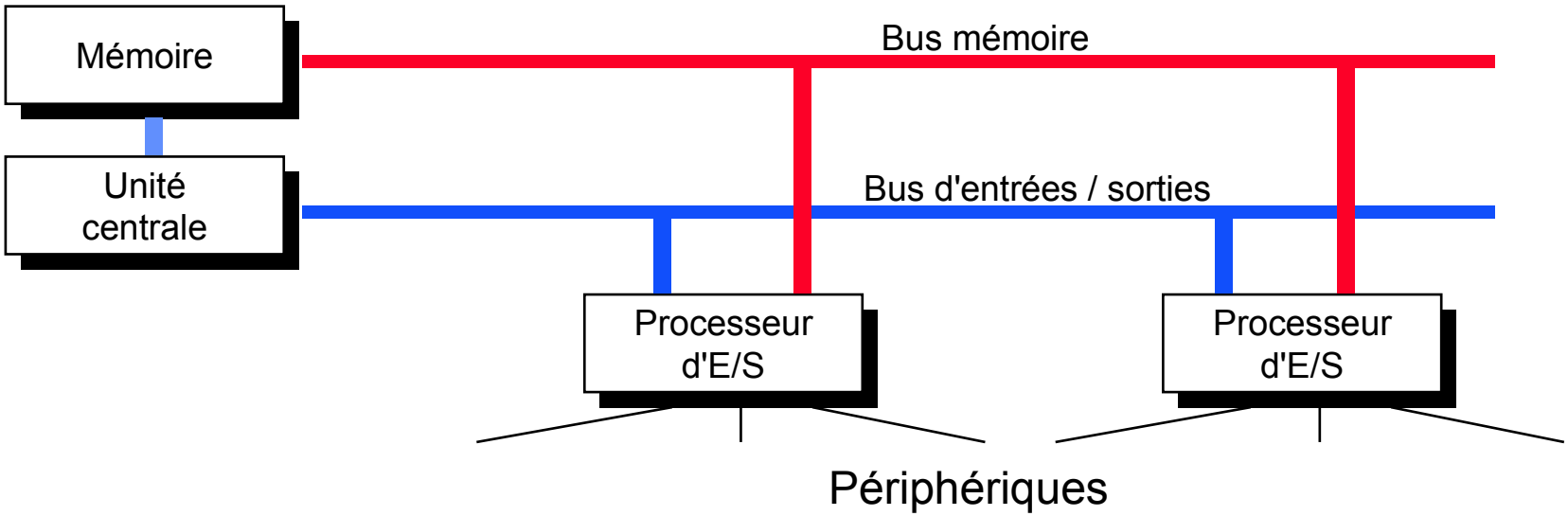
➢Les entrées sorties :

- ♦ échange d'informations entre l'environnement (utilisateur, capteur, appareils de mesure, etc) et la machine
- ♦ 2 modes d'organisation :
  - – pour les petits systèmes :
    - beaucoup plus simple
    - 1 seul bus raccordant le microprocesseur, la mémoire et les entrées/sorties
    - les périphériques sont constitués de 2 parties : le contrôleur et le périphérique à proprement parlé
    - le rôle du contrôleur consiste à piloter le périphérique et à gérer les accès au bus
    - un mécanisme d'arbitrage permet de régler les conflits lorsque le processeur et un contrôleur veulent utiliser simultanément le bus



Introduction

– pour les grands systèmes :
  - mise en place de processeurs d'entrées sorties
  - les périphériques sont connectés sur des canaux gérés par ces processeurs
  - les entrées sorties sont sous traitées aux processeurs spécialisés, soulage l'unité centrale, travail en parallèle
  - bus d'entrées sortie permet au processeur de dialoguer avec les processeurs spécialisés
  - bus mémoire permet aux processeurs d'E/S des accès directe à la mémoire sans passer par le processeur



Introduction

❑ Qu'est ce qu'un système informatique ?

- ➢ système regroupant plusieurs machines :
  - ♦ chaque machine est composée de nœuds:
    - – homogènes
    - – hétérogènes
  - ♦ les nœuds sont soient :
    - – des éléments simples (processeurs simples, petites mémoires)
    - – des éléments complets (ordinateurs complets, processeurs, mémoire, disques, entrées sorties,etc)
  - ♦ les machines dialoguent : (échangent des informations, coopération pour la réalisation d'un travail complexe) :
    - – au moyen de bus communs
    - – d'envoi de messages
  - ♦ notions:
    - – de parallélisme
    - – de partage de ressources
    - – tolérances aux fautes
    - – etc

Machines algorithmiques

Machines dédiées et programmables

❑ Qu'appelle-t-on "machine algorithmique " ?

➢ machine capable :

- ♦ d'exécuter une suite d'instructions de manière automatique
- ♦ de prendre des décisions quant à la suite des traitements à réaliser

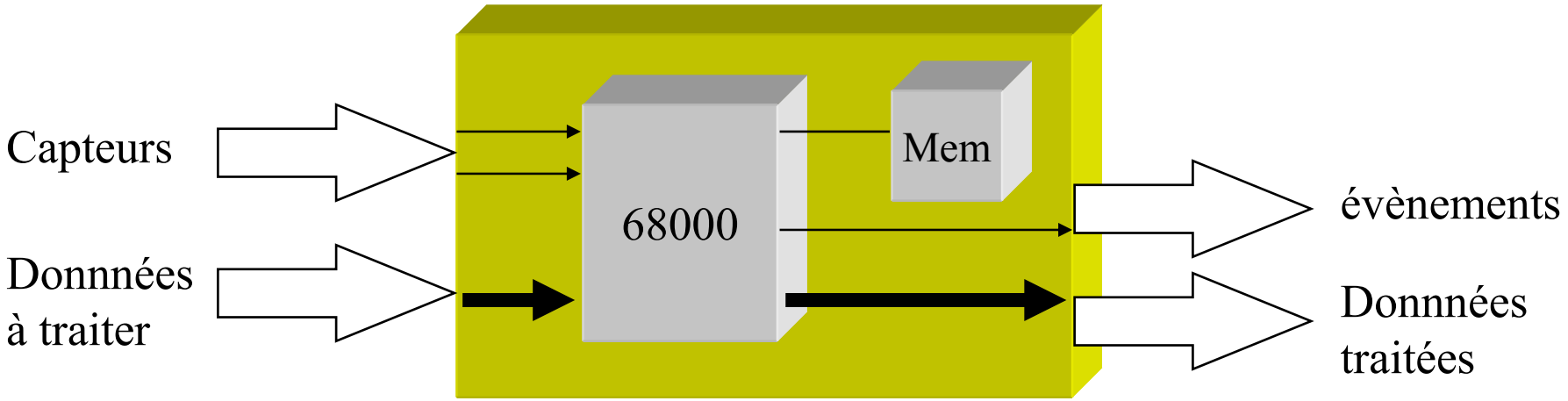
➢ on distingue deux classes :

- ♦ les machines programmables
- ♦ les machines dédiées

Machines dédiées et programmables

➢Les machines programmables :

♦ modèle :



♦ notion de flexibilité :

– existence d'un programme "déroulant" l'application :
  • si l'application change, il "suffit" de modifier le programme

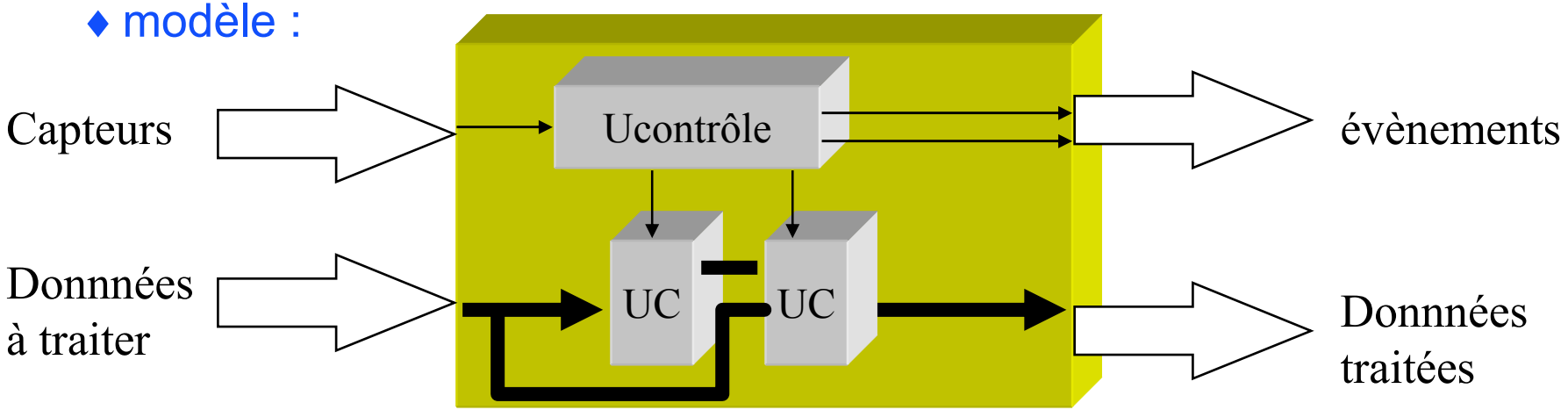
♦ machine d'usage générale :

– utilisation non optimale du processeur
– l'application est "posée" sur le modèle du processeur

Machines dédiées et programmables

➢Les machines dédiées :

♦ modèle :



♦ pas de flexibilité :

- bâtit le processeur à partir des besoins de l'algorithme
- mise en place du bon nombre d'unités de calcul
- chaque unité est "taillée" en fonction des besoins :
  - vitesse (temps de calcul)
  - surface (coût silicium)
  - consommation (puissance)

♦ machine d'usage spécifique :

- utilisation optimale de la machine

Machines dédiées et programmables

➢Machines Dédiées ou Programmables ?

- ♦ programmables :
  - souple pour les modifications éventuelles
  - intéressant pour la réalisation de prototypes

dans certain cas, ne permet pas d'atteindre les performances désirées

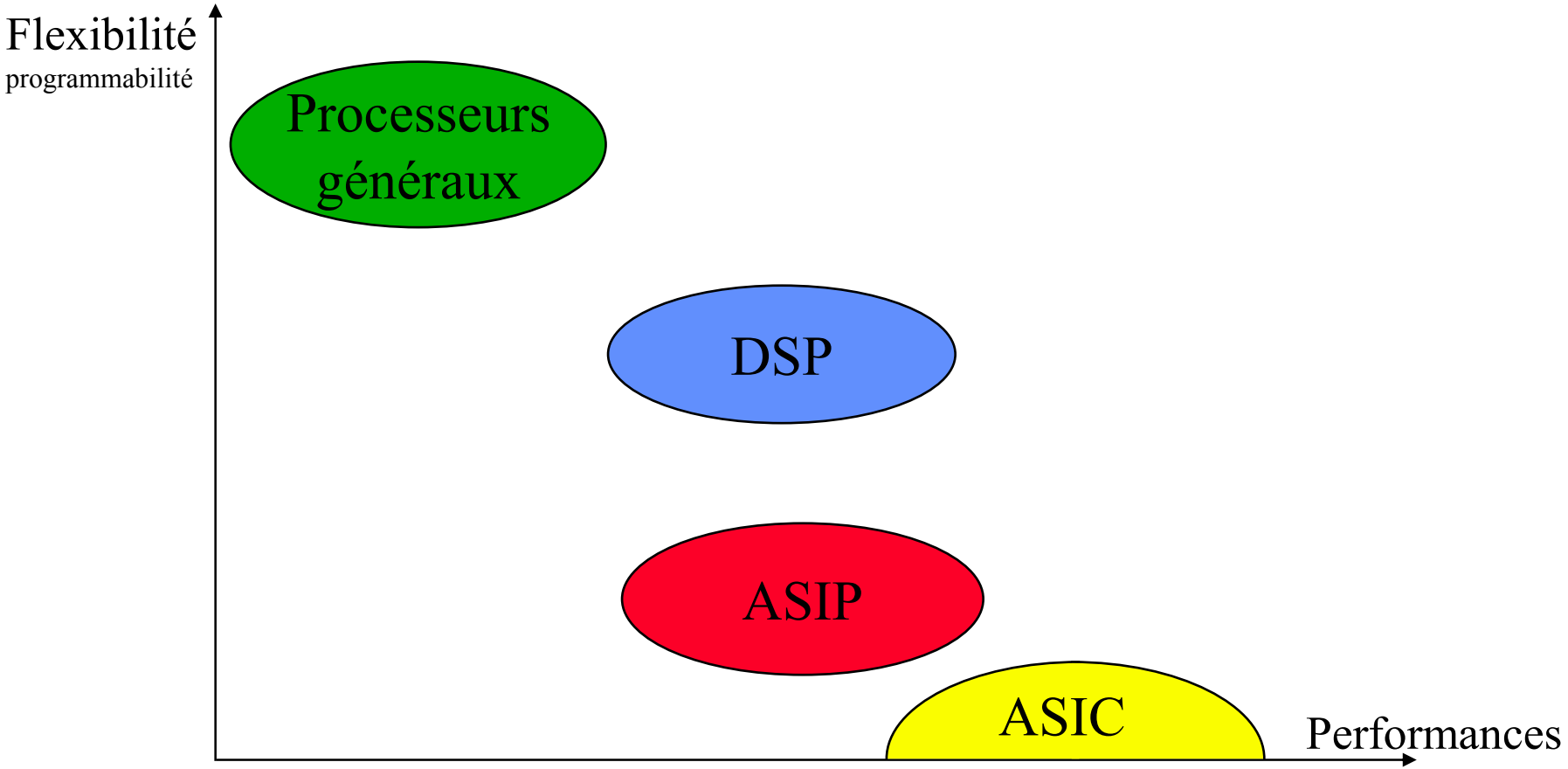
- ♦ dédiées :
  - performantes pour l'application ciblée
  - intéressantes pour les grandes séries

pas évolutives

- ♦ concept d'ASIP :
  - a mi chemin entre programmable et dédié
  - on parle de dédié à une classe d'applications

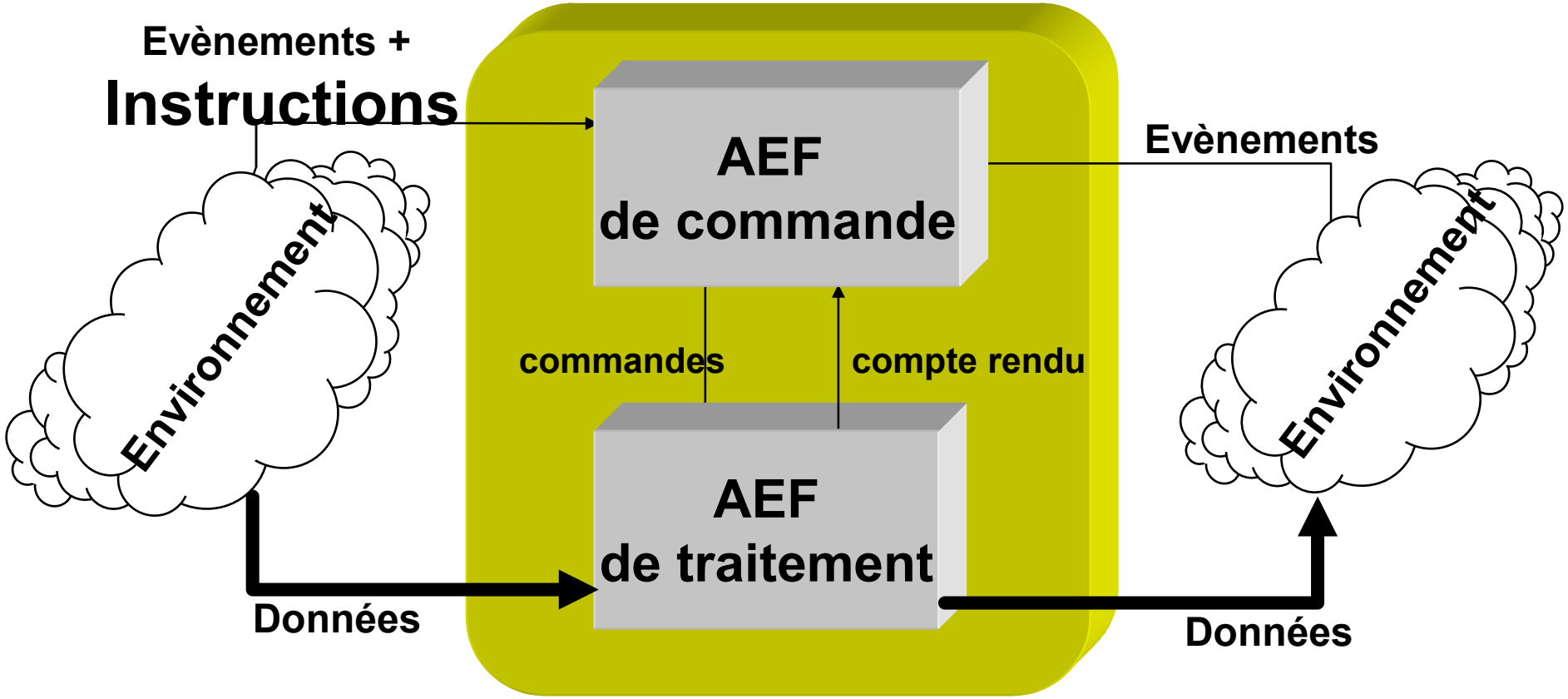
Machines dédiées et programmables

Pour une application donnée



Machines dédiées et programmables

❑ Modélisation d'une machine algorithmique



AEF : Automate à Etats Finis :

➢modélisation mathématique

$$St = g\ (Xt, Et)$$
$$Xt+1 = f\ (Xt, Et)$$

- Et : les variables d'entrée
- St : les variables de sortie
- Xt : le vecteur d'état de l'automate
- g : la fonction qui définit l'état des sorties
- f : la fonction qui définit l'évolution de l'automate

Machines dédiées et programmables

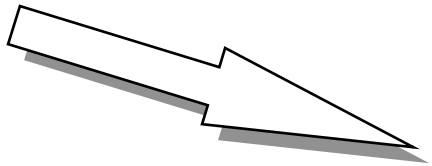
➢ Ils sont plus ou moins compliqués selon les applications :

- machines à faible contrôle : **flot de données**
  - peu ou pas de gestion de branches conditionnelles :
    - if then else
    - switch case
  - beaucoup de calculs
  - exemples : calculs scientifiques, calculs matricielles, etc
- machines ayant peu de calcul : **flot de contrôle**
  - beaucoup de structures conditionnelles
  - peu de calculs
  - exemples : recherche de solutions dans des graphes, parcours de listes

Machines dédiées et programmables

➢L'automate de commande :

- ♦ réagit aux évènements extérieur
- ♦ crée des évènements pour l'environnement
- ♦ génère le flot de commandes
- ♦ traduit l'algorithme à exécuter



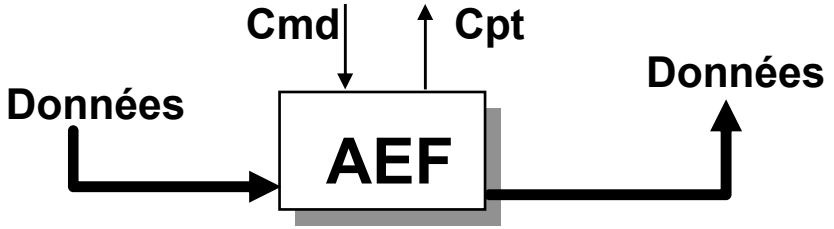
Séquencement

➢ L'automate de traitement :

- – il est piloté par l'AEF de commande
- – réagit aux commandes de celui-ci
- – génère des données pour l'environnement
- – exécute les opérations sur les données reçues
- – produit des comptes rendus de traitement (de calcul)

Machines dédiées et programmables

❑ AEF de traitement :



Et : les variables d'entrée :

commande provenant de l'automate de commande

données provenant de l'environnement (extérieure)

St : les variables de sortie :

compte rendu de calcul

action sur l'environnement

Xt : le vecteur d'état de l'automate :

mémorisation intermédiaire de calcul

$$St = g(Xt, Et)$$
$$Xt+1 = f(Xt, Et)$$

g : la fonction qui définit l'état des sorties :

définit les actions

définit les comptes rendus

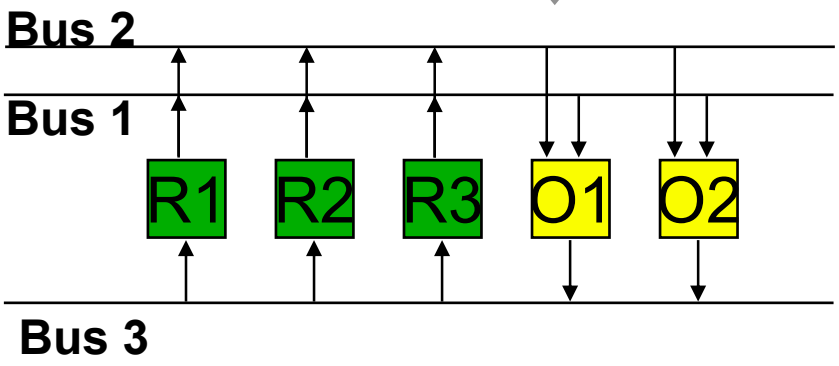
f : la fonction qui définit l'évolution de l'automate :

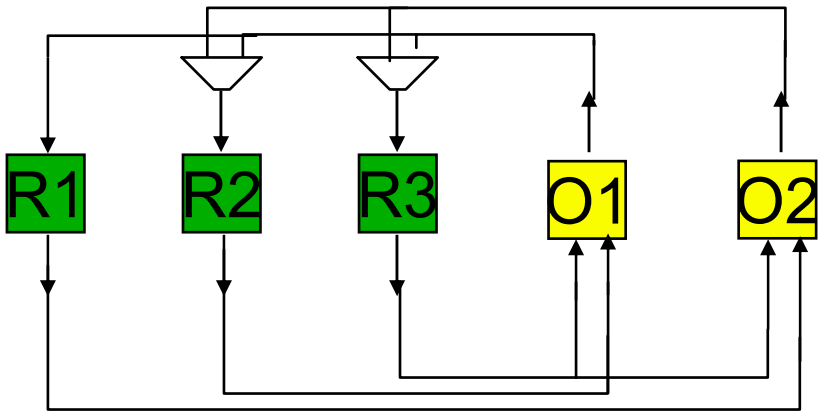
définit les calculs en fonction des commandes

Machines dédiées et programmables

❑ AEF de traitement (suite) :

- ➢2 primitives :
  - ♦ calcul : fonctions combinatoires
  - ♦ mémorisation : registres, file de registres, mémoire
- ➢2 modèles :
  - ♦ chemins point à point
  - ♦ chemins partagés





Machines dédiées et programmables

➢Avantages et inconvénients :

- ♦ bus point à point :
  - parallélisme maximum
  - implantation efficace des algorithmes :
    - extraction du parallélisme de l'application
    - ordonnancement des opérations sur les unités de calculs
  - coûteux :
    - très nombreux bus
    - nombreuses connections
  - intéressant pour les machines à fort taux de calcul : flot de données

- ♦ bus partagés :
  - les bus sont un goulot d'étranglement
  - moins coûteux
  - nécessité de séquencer les opérations indépendantes
  - modèle de tous les processeurs existants

Machines dédiées et programmables

❑ Qu'est ce qui distingue les AEF de traitement ?

- ➢ largeur des bus :
  - ♦ d'abord en 4 bits, puis 8, ... 64 bits, voir même 128 bits
- ➢ complexité des unités de calcul :`
  - ♦ d'abord entière puis flottante
  - ♦ en 4, puis 8, ... puis 64 bits, (128 bits dans certains cas)
- ➢ nombre de ressources de calcul :
  - ♦ 1 UAL : addition, soustraction, opérations logiques (ou, et, not, etc)
  - ♦ ajout d'une unité flottante, doublement des unités de calcul
- ➢ nombre de registres :
  - ♦ une dizaine dans les premiers processeurs à quelques dizaines dans les processeurs RISC
  - ♦ le rôle des registres :
    - – de calcul d'adresses et de données
    - – ou généraux

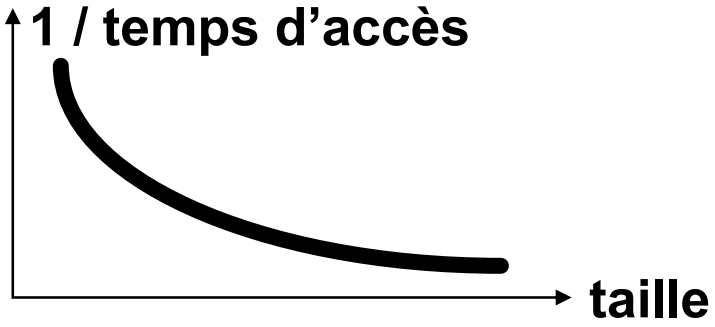
Machines dédiées et programmables

➢la taille de la mémoire adressable :

- ♦ quelques kilo octets à quelques giga octets

➢la présence d'une mémoire cache :

- ♦ recherche d'un compromis entre taille et temps d'accès
- ♦ registres très rapide ⇨ mais coûte cher
- ♦ taille mémoire et temps d'accès évoluent dans des directions opposées



- ♦ mémoires externes aux processeurs ⇨ lente, passage par des buffers et par les pads du circuit
- ♦ l'évolution de la densité a permit d'implanter des mémoires dans le circuit ⇨ bénéficie d'un temps d'accès très rapide

Machines dédiées et programmables

- Caractéristiques de quelques processeurs (1994 - 1995)

	Alpha 21064	Mips R4000	Pentium	Power PC 601	68040	80486
Fréquence	200 Mhz	150 Mhz	66 Mhz	80 Mhz	25 Mhz	50 Mhz
Largeur bus	64 bits	64 bits	32 bits	32 bits	32 bits	32 bits
Cache	2 * 8 ko	2 * 16 ko	2 * 8 ko	32 Ko	2 * 4 ko	8 ko
Unités indép	4	??	3	3	1	1
Registres	2 * 32	2 * 32	2 * 8	2 * 32	16 + 8	2 * 8
Consom	30 w	15 w	16 w	9 w	6 w	5 w
Prix	2,2	4,7	3,8	2,4	1	1,8

Machines dédiées et programmables

- Caractéristiques de quelques processeurs (1997)

	Alpha 21164	Mips R10000	Pentium II	Power PC 604e
Fréquence	600 Mhz	275 Mhz	300 Mhz	350 Mhz
Largeur bus	64 bits	64 bits	32 bits	32 bits
Cache	16 et 8 ko	2 * 32 ko	2 * 16 ko	2 * 32 Ko
Unités indép	4	5	7	6
Registres	2 * 32	2 * 32		
Consom	40 w	30 w	? w	12 w
Prix				

Machines dédiées et programmables

- Caractéristiques de quelques processeurs (2002)

	Pentium 4	Alpha 21364	G4	
Fréquence	2400	1200	1250	
Largeur bus		64		
Cache	12 K trace L1 = 8K L2 = 256 K	L1 = 2*64K L2 = 1,75 M	L1 = 2 * 32K L2 = 256K L3 = 2Mb off	
Unités indép	7	6		
Registres	32 entiers, 80 flottants, 64 MMX	2 * 80 int reg 72 float reg	32 reg de 128 bits	
Consom	55	155	30	
Prix			~ $300	

Machines dédiées et programmables

- Caractéristiques de quelques processeurs (2002)

TABLE I
ITANIUM 2 VERSUS OTHER 64-B 18-μm PROCESSORS

	Itanium 2	USIII	EV7	Power 4 (2 cores)
Frequency (GHz)	1.0	1.05	1.0-1.2	1.3
Pipe stages (mpb)	**8 (6)**	14 (8)	9 (~11)	12 (~11)
Sustainable Int BW	**6 / cycle**	3 / cycle	4/cycle	3/cycle
FP units	2/cycle	2/cycle	2/cycle	2/cycle
On chip cache	**3.3MB** **4 arrays**	96KB 2 arrays	1.8MB 3 arrays	1.7MB 2.5 per core
D Cache read BW	**64GB/s**	16.8	19.2	41.6 / core
Die size (mm^2)	**421**	244	397	400 est.
Core size (no IO, only lowest level caches)	142	206	115	100 est.
Power (Watts)	**130**	75	125	125

Machines dédiées et programmables

❑ AEF de commande :

Et : les variables d'entrée :

compte rendu provenant de l'automate de traitement

évènements provenant de l'environnement (extérieure)

St : les variables de sortie :

évènements à destination de l'environnement

commandes pour le traitement

Xt : le vecteur d'état de l'automate :

état courant de l'automate

$$St = g(Xt, Et)$$
$$Xt+1 = f(Xt, Et)$$

g : la fonction qui définit :

définit les commandes pour l'AEF de traitement

définit les évènements pour l'environnement

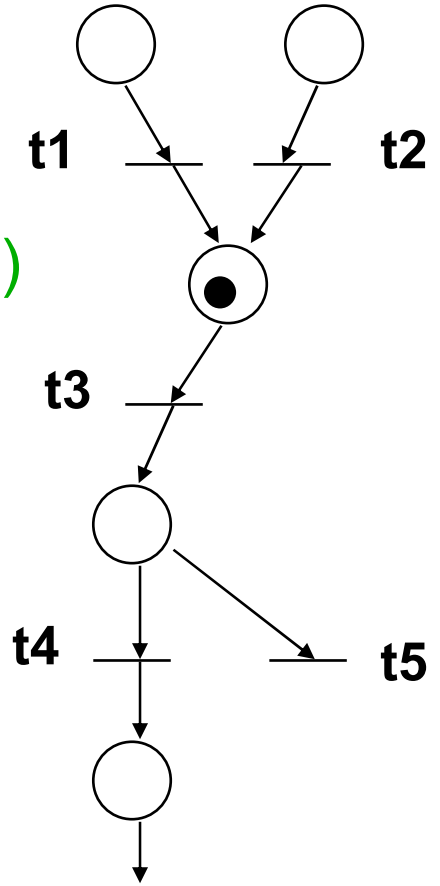
f : la fonction qui définit l'évolution de l'automate :

définit l'état suivant en fonction de l'état courant et des entrées

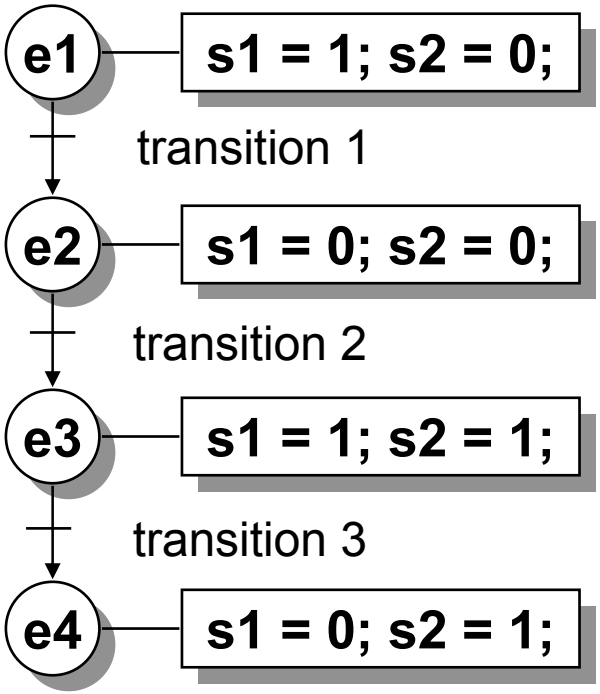
Machines dédiées et programmables

❑ AEF de commande : Modèles utilisés :

➢réseaux de Pétri



➢machine d'états (synchrone ou asynchrone)



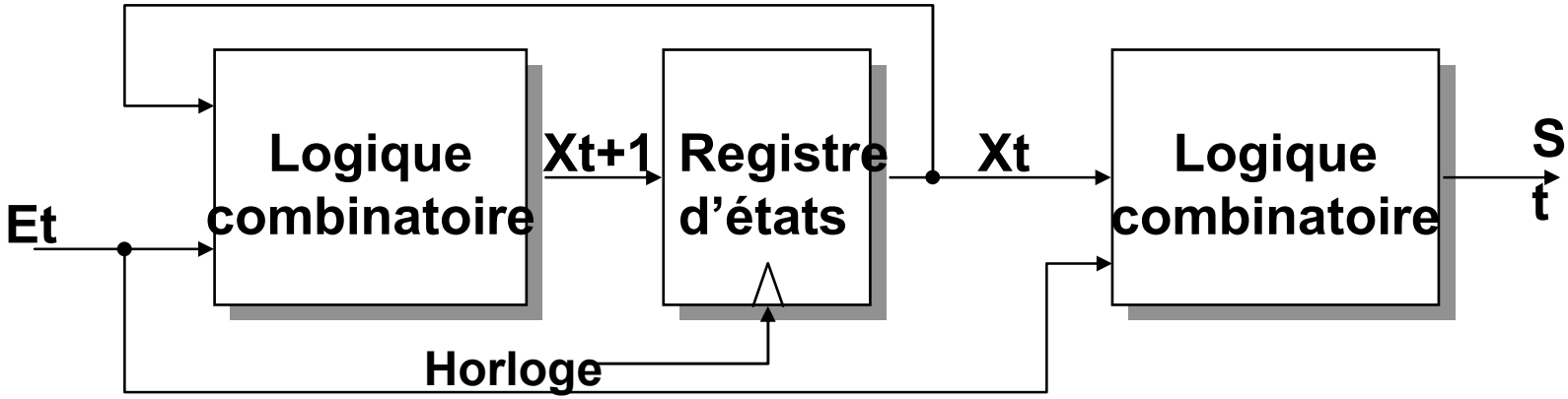
Notion de synchronisation
par rapport à une horloge :

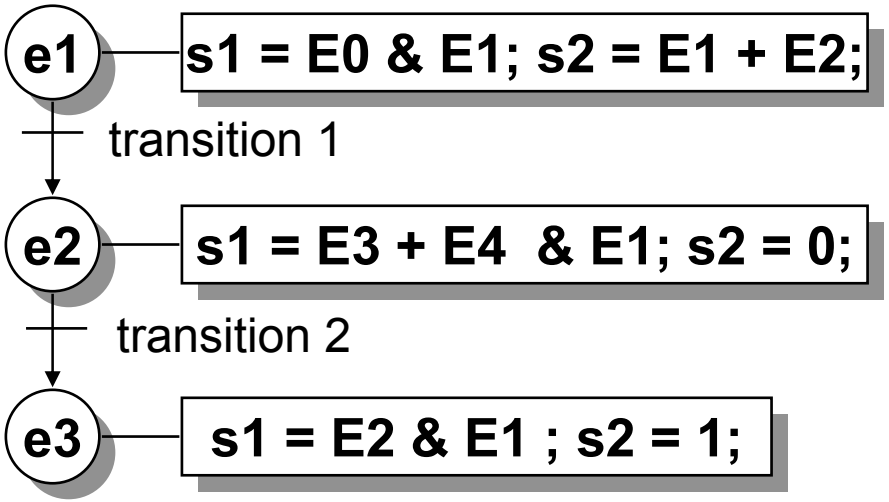
- explicite : exprimée dans la transition
- implicite : n'apparaît pas dans la transition

Machines dédiées et programmables

❑ AEF de commande : matérialisation

➢ machine de Mealy

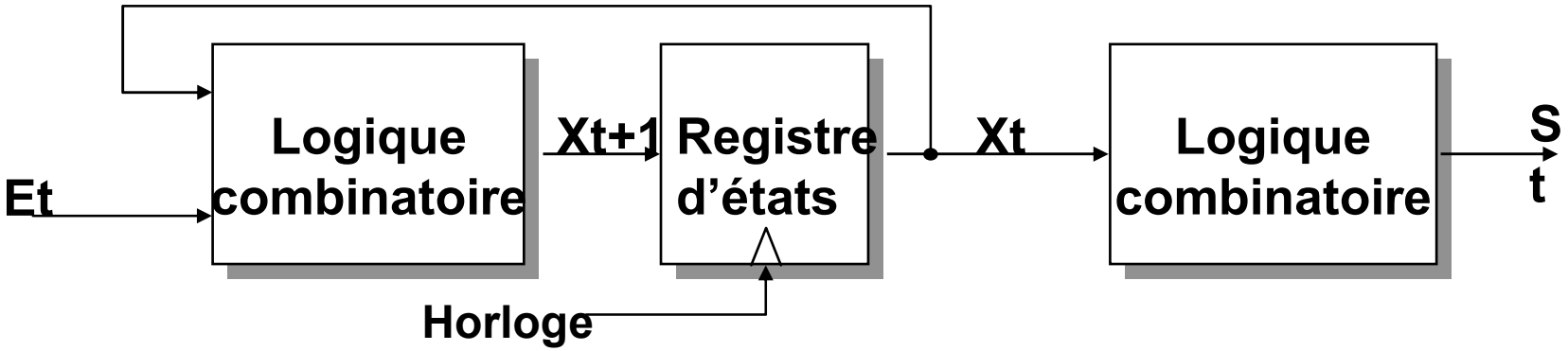


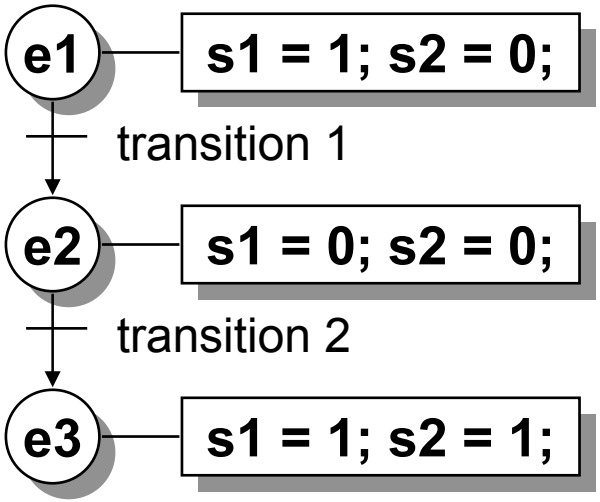


$$St = g(Xt, Et)$$
$$Xt+1 = f(Xt, Et)$$

Machines dédiées et programmables

➢machine de Moore

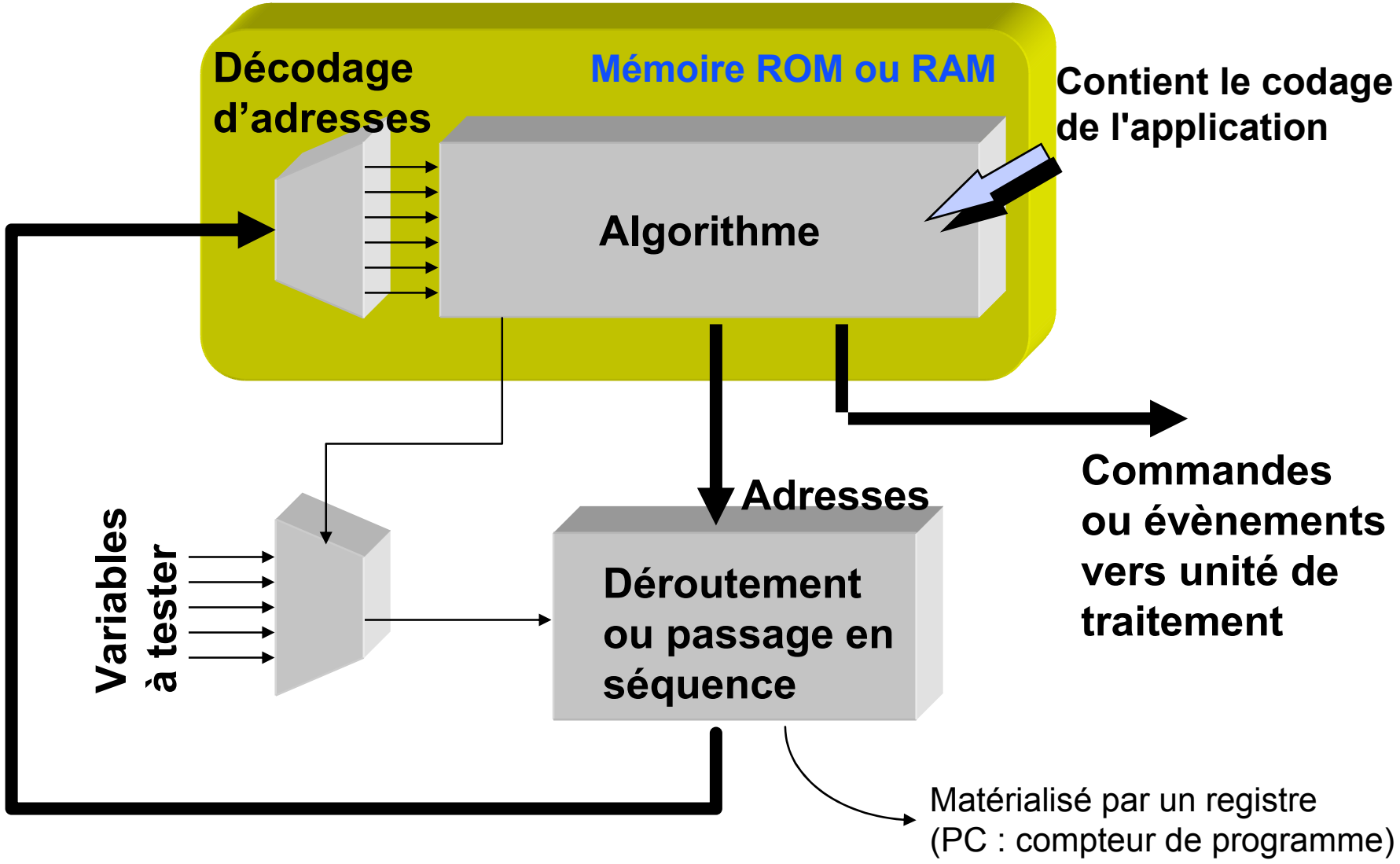




$$St = g(Xt)$$
$$Xt+1 = f(Xt, Et)$$

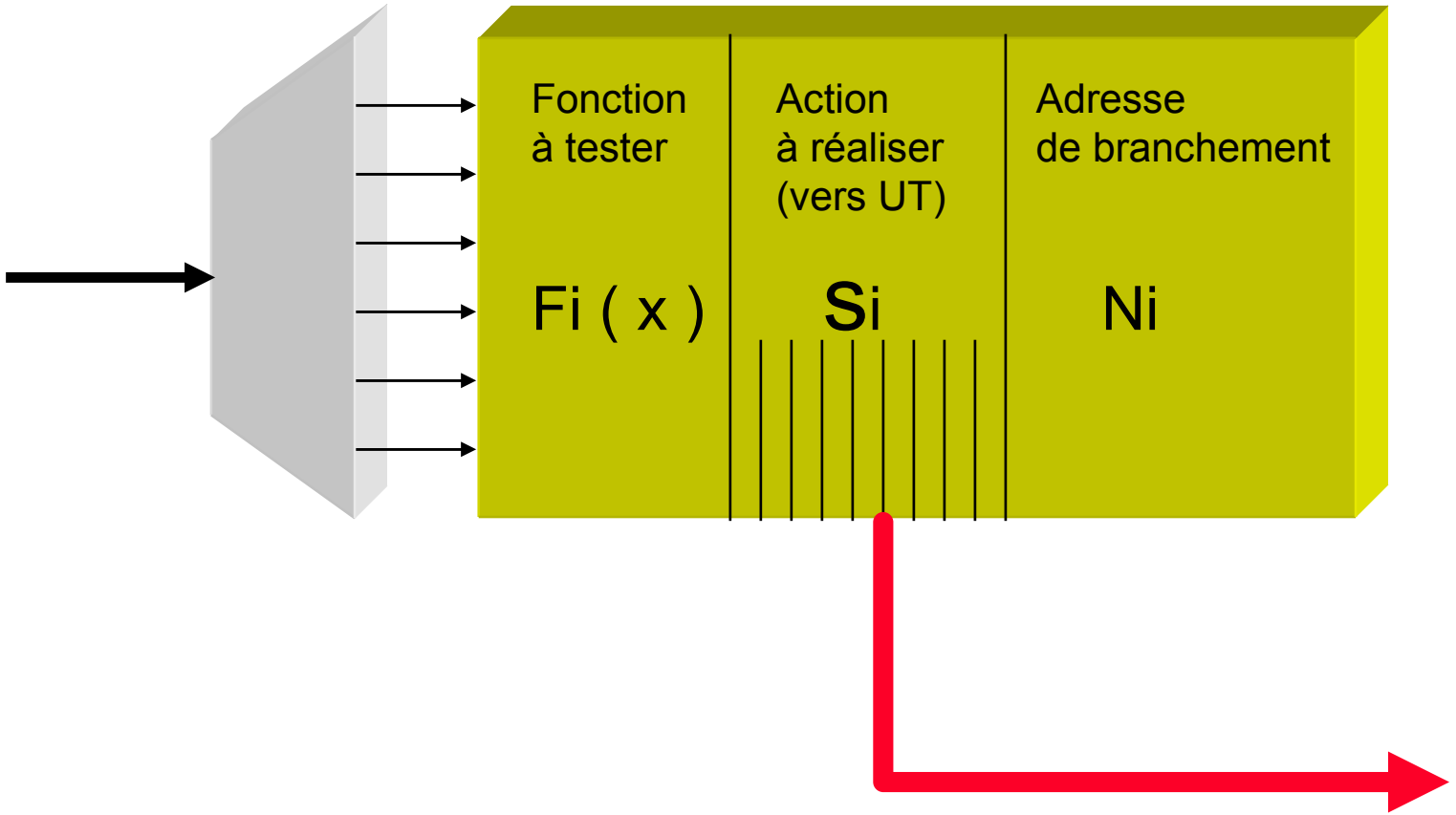
Machines dédiées et programmables

➢stockage dans une mémoire programme :



Machines dédiées et programmables

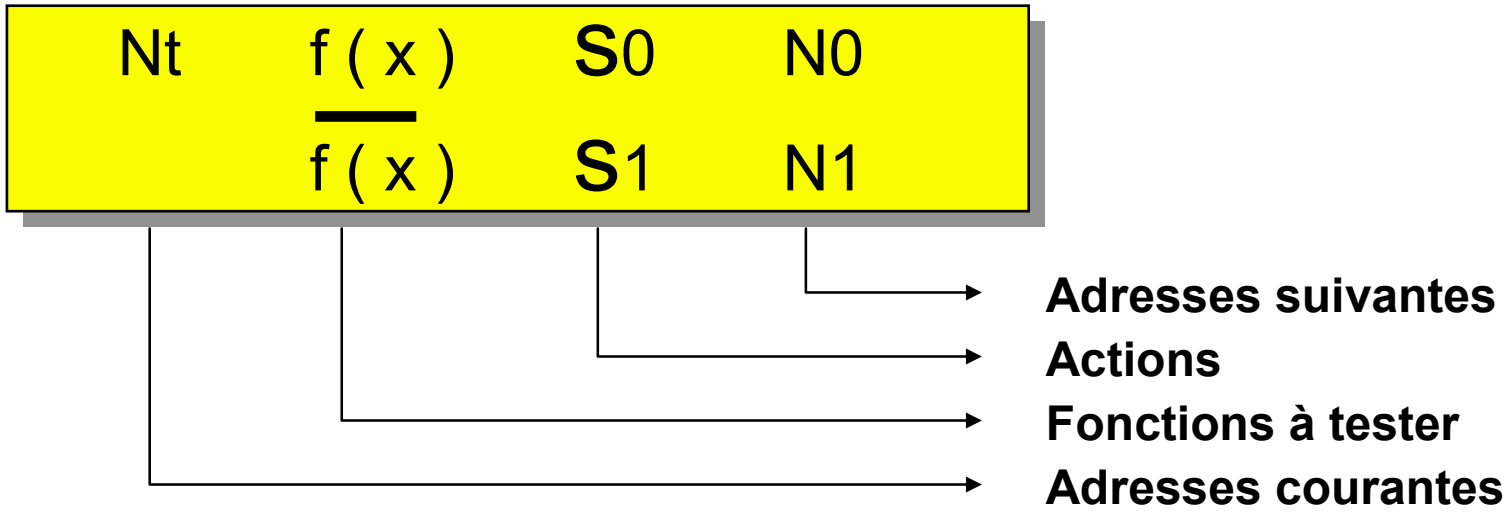
➢Les champs de la mémoire programme :



2.3) Implantation d'un algorithme :

- tout programme peut être implanté dans ce type de machine

- Le modèle général est le suivant :
  - on peut faire un test, une action et un branchement



Machines dédiées et programmables

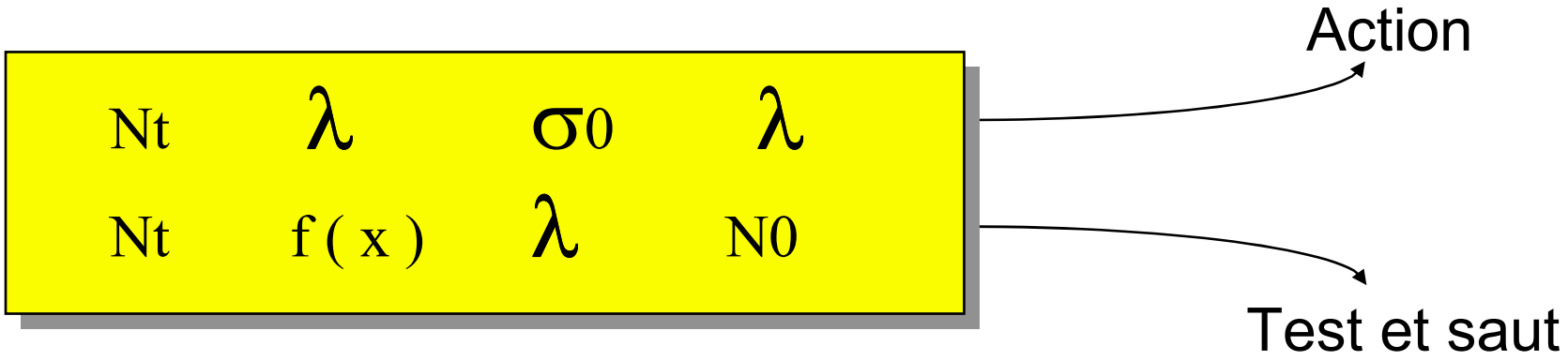
♦ modèle simplifié, on peut faire soit :

– **action** :

- on ne teste rien, on effectue l'action et on poursuit sur l'adresse de programme suivante (Nt+1)

– **test** :

- on n'effectue aucune action, on réalise le test et en fonction du résultat de l'évaluation de la condition on effectue un branchement ou non
- d'un point de vue AEF de traitement, aucune action n'est réalisée
- d'un point de vue AEF de commande, il s'agit d'une action de chargement d'une nouvelle valeur dans le PC de la machine

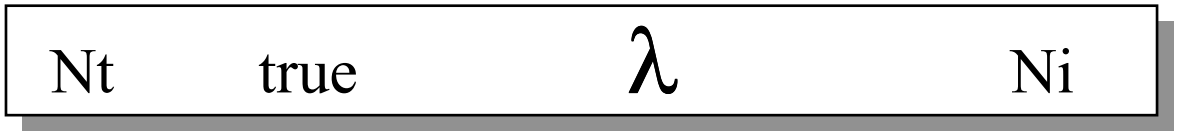


λ : ne rien faire

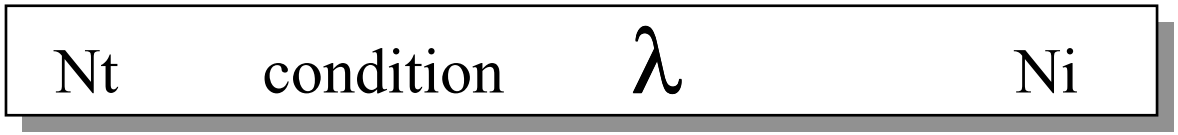
Machines dédiées et programmables

- **Instructions de calcul ou de transfert :**
  - *r1 := r1 + r2 ;*

- **Instructions de saut inconditionnel :**
  - *BRA Ni ;*



- **Instructions de saut conditionnel :**
  - *BCC Ni ;*



Machines dédiées et programmables

Instructions de test

if *condition* then r1 := r1 + r2 ;

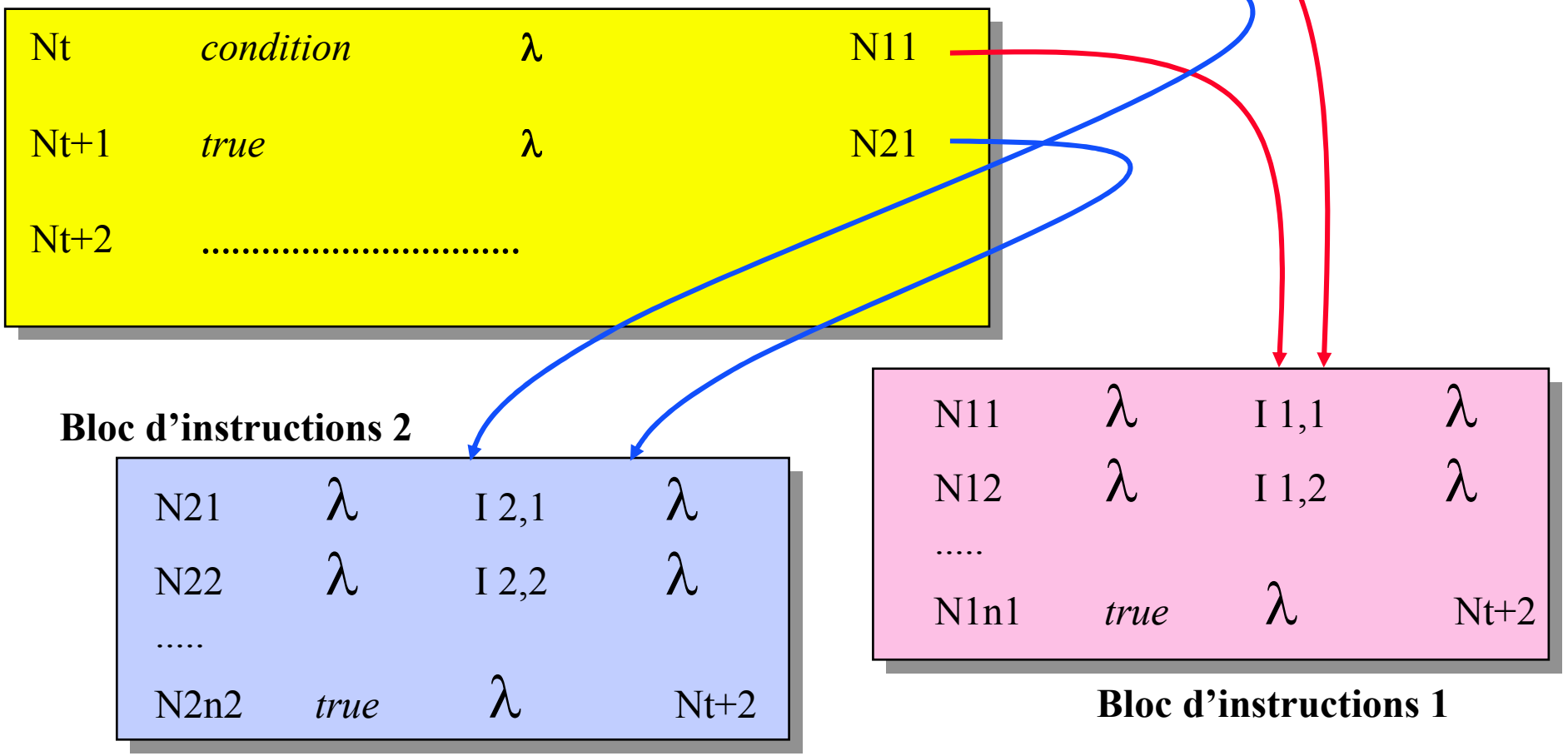
else r1 := r1 - r2 ;

Nt	$\overline{condition}$	λ	Nt+3
Nt+1	λ	Add r1,r1, r2	λ
Nt+2	*true*	λ	Nt+4
Nt+3	λ	Sub r1, r1, r2	λ
Nt+4	.....		

Machines dédiées et programmables

- **Instruction de test :**

  if *condition* then ***bloc instructions 1*** ;
  else ***bloc instructions 2*** ;



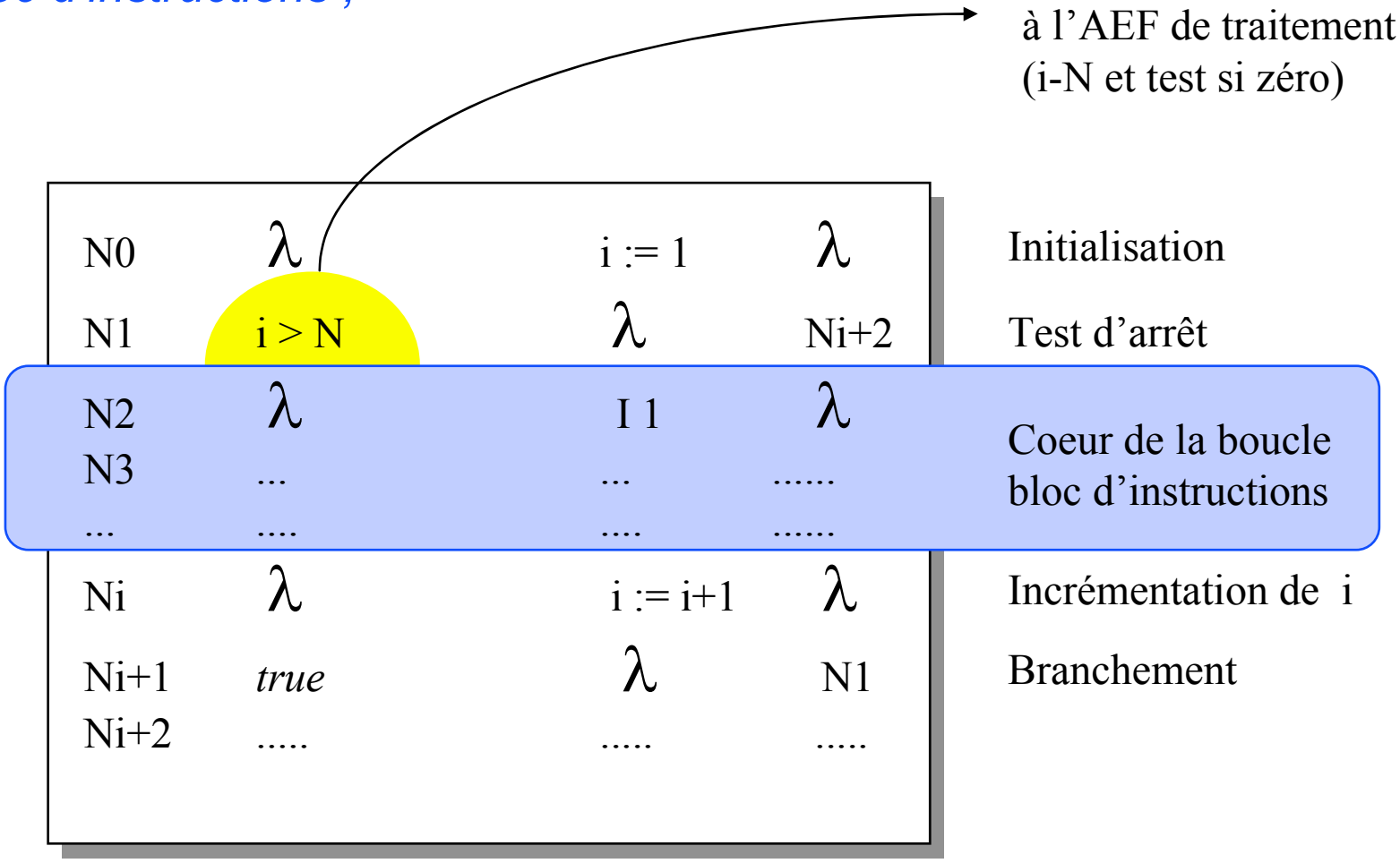
Machines dédiées et programmables

Instruction de boucles

pour i de 1 à N *faire*

bloc d'instructions ;

fait ;



Machines dédiées et programmables

Instruction de boucles

faire

bloc d'instructions ;

tant que condition ;

Branchement

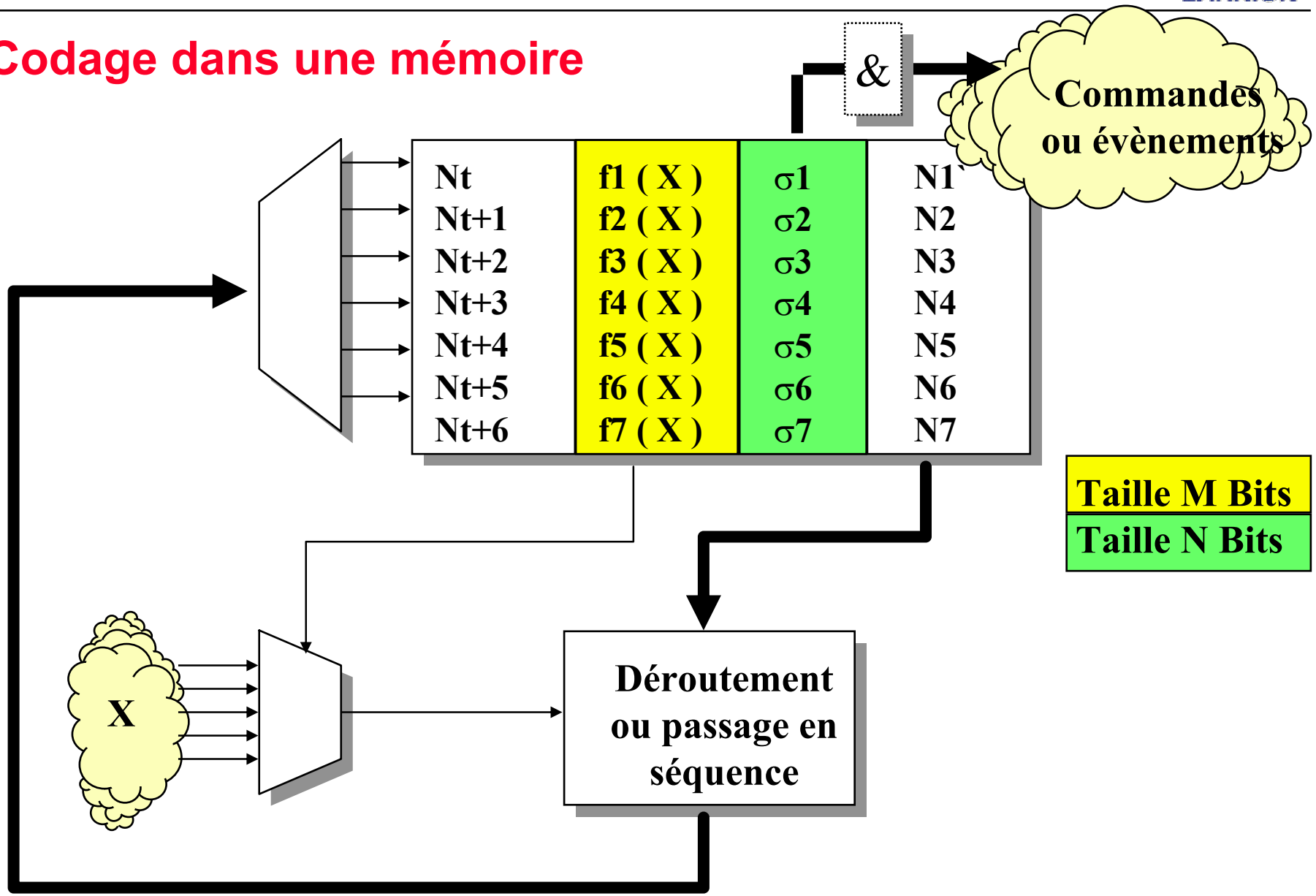
Test d'arrêt

….	…..	…..	…..
N0	λ	I 1	λ
N1	…	I 2	λ
…	….	….	…..
…	….	….	…..
Ni	*condition*	λ	N0
Ni+1	…..	…..	…..

Coeur de la boucle
bloc d'instructions

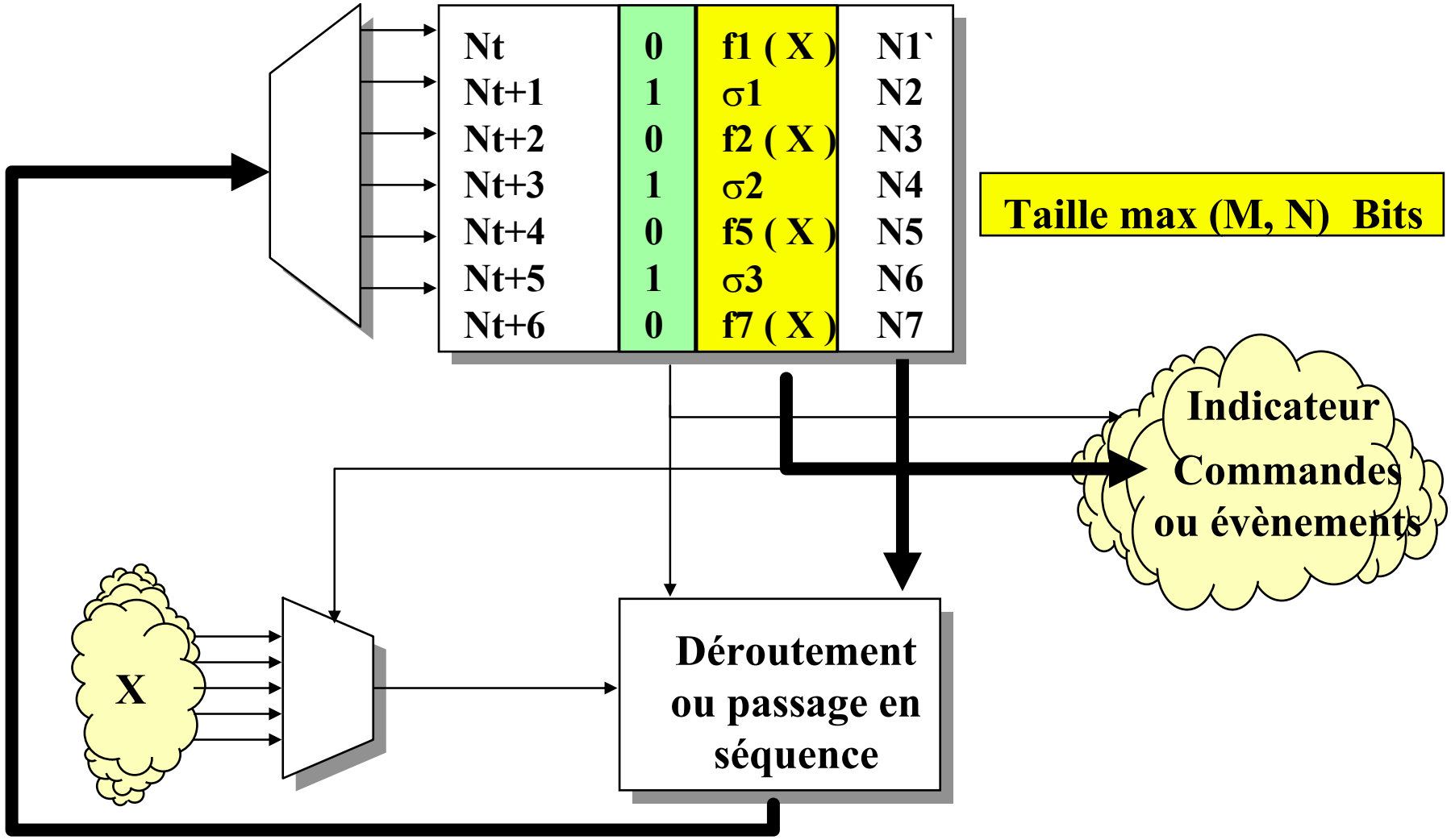
Machines dédiées et programmables

- Codage dans une mémoire



Machines dédiées et programmables

❑ Le modèle dégénéré : diminution du codage

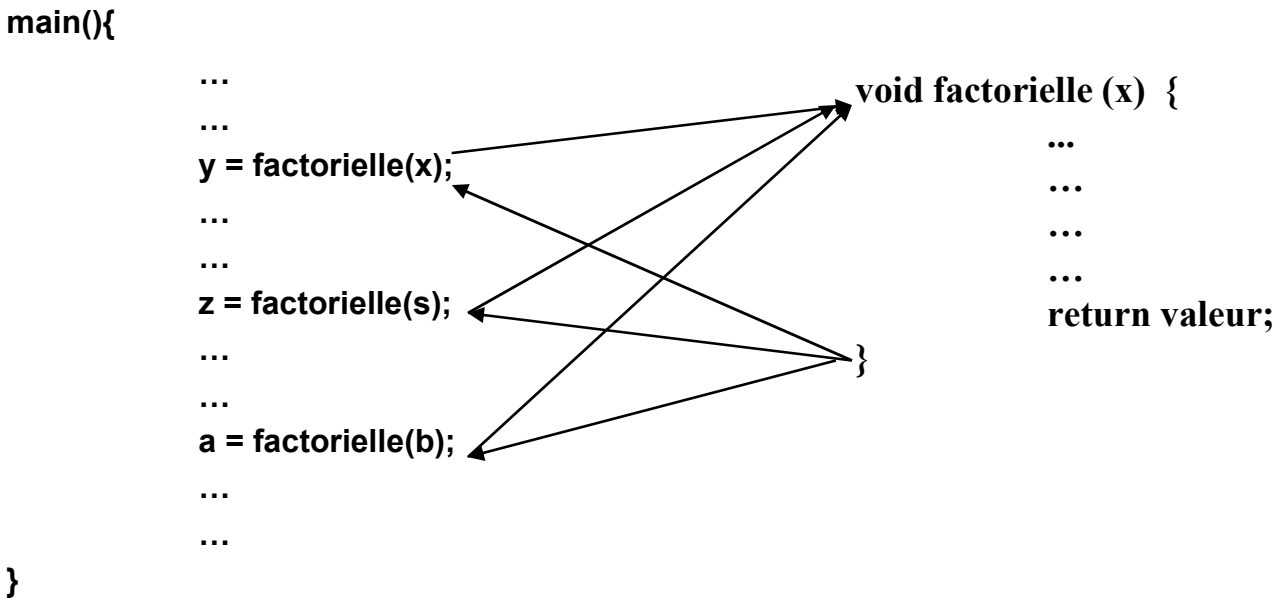


Machines dédiées et programmables

❑ Le déroutement et l'appel de sous programme :

➢ aide à la structuration de programmes :

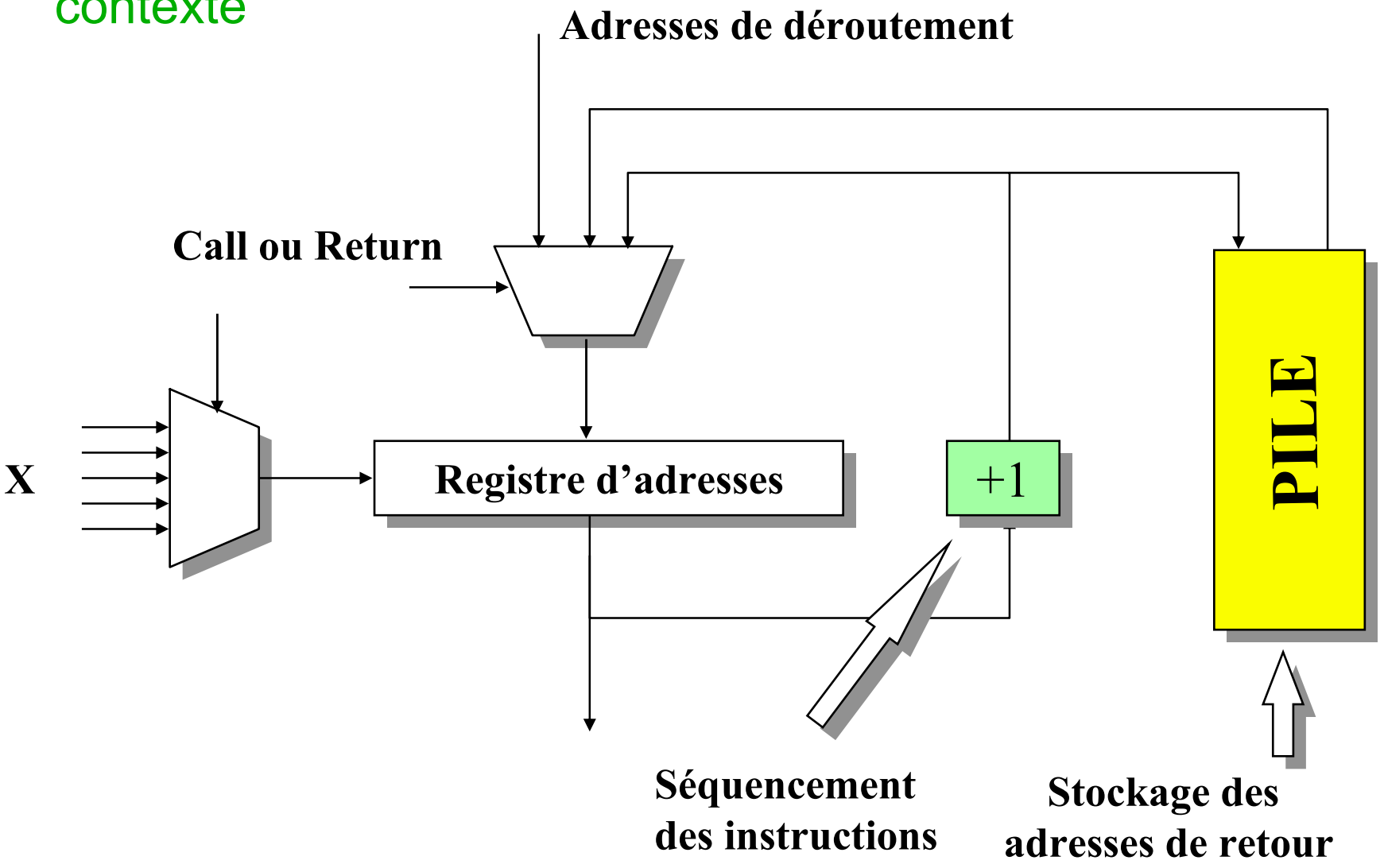
♦ appels de procédures ou fonctions



♦ nécessite la mémorisation de l'adresse de retour

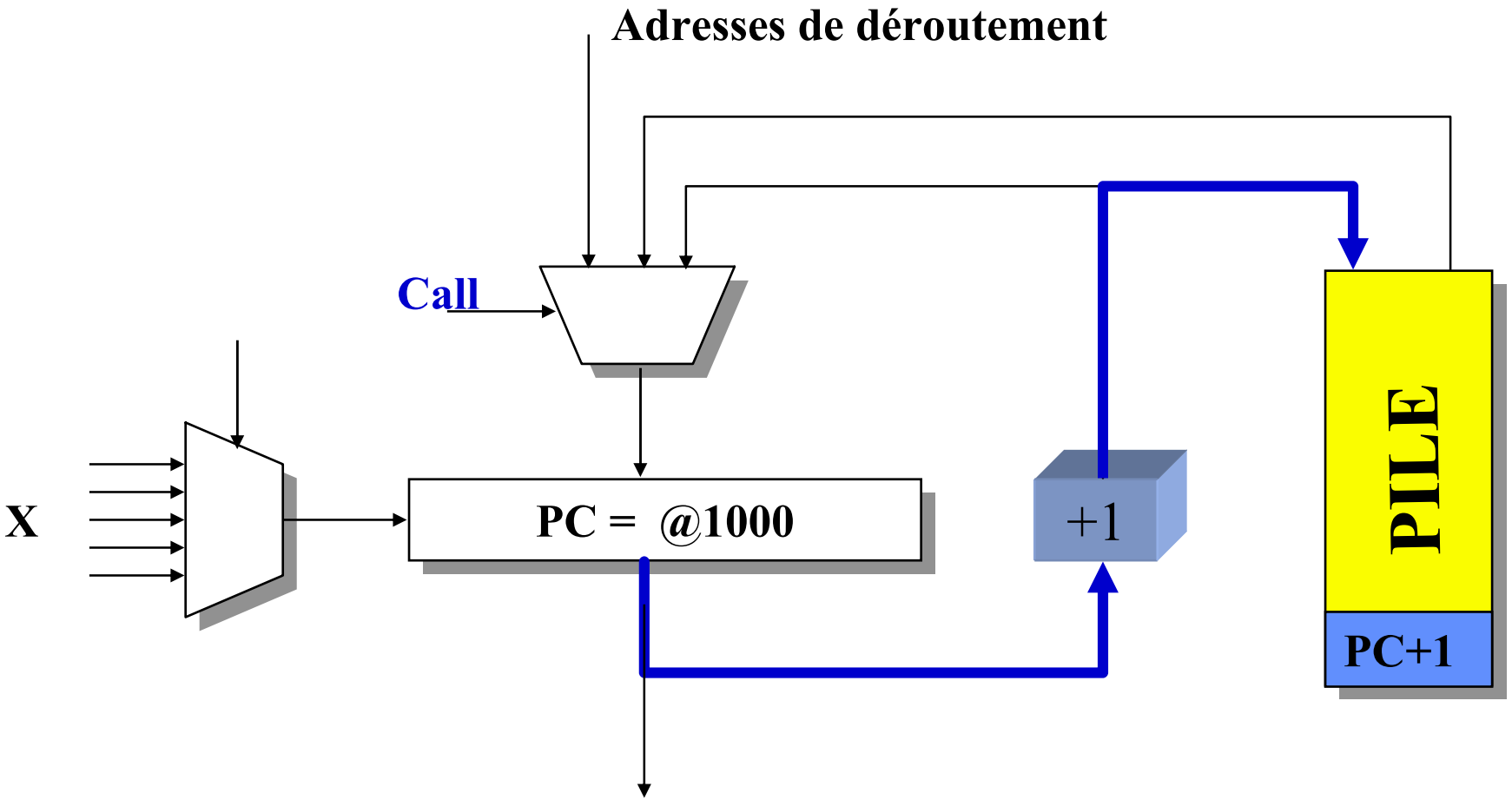
Machines dédiées et programmables

➢ structure pour assurer le déroutement avec sauvegarde de contexte



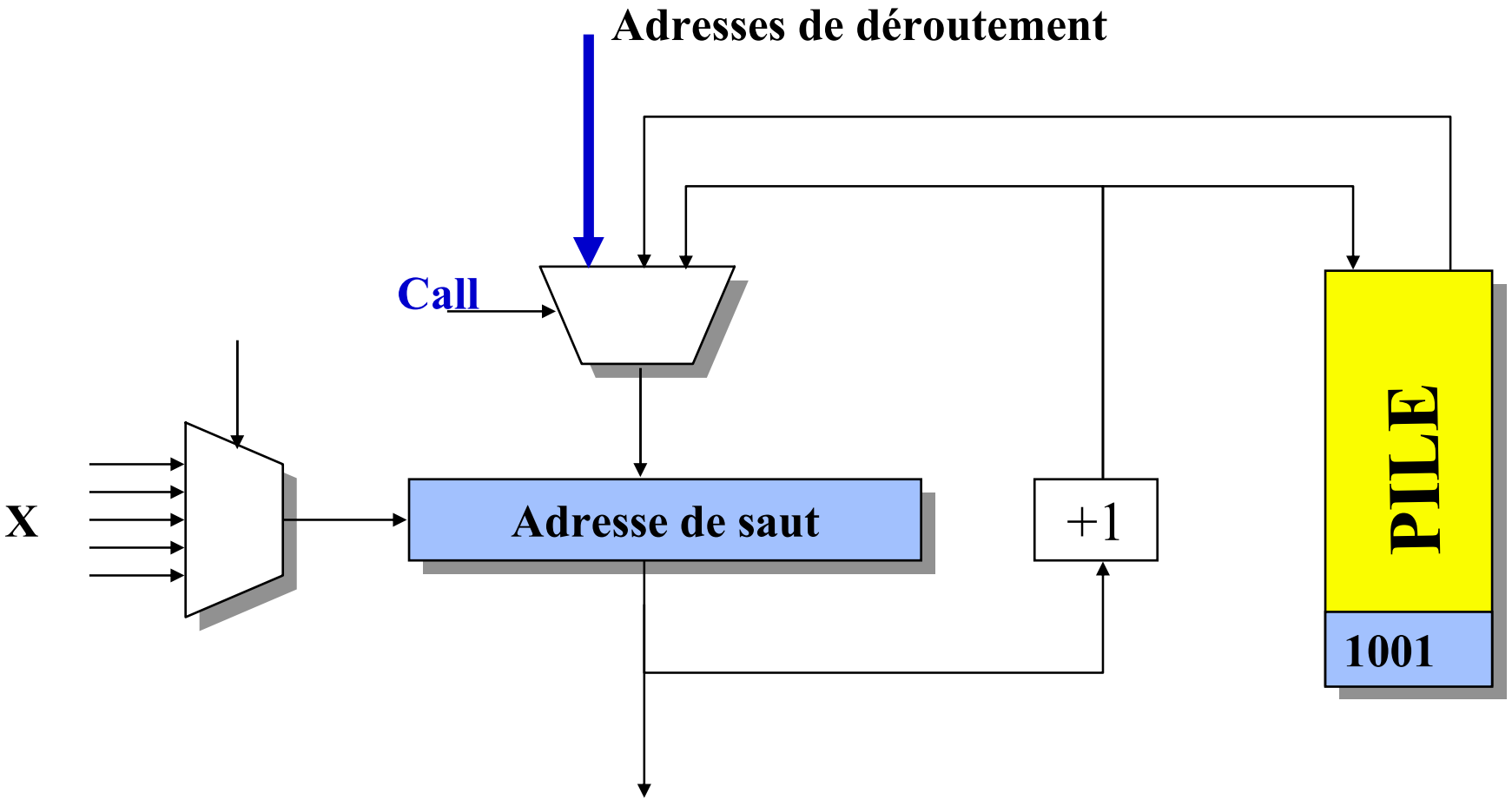
Machines dédiées et programmables

➢ instruction call factorielle



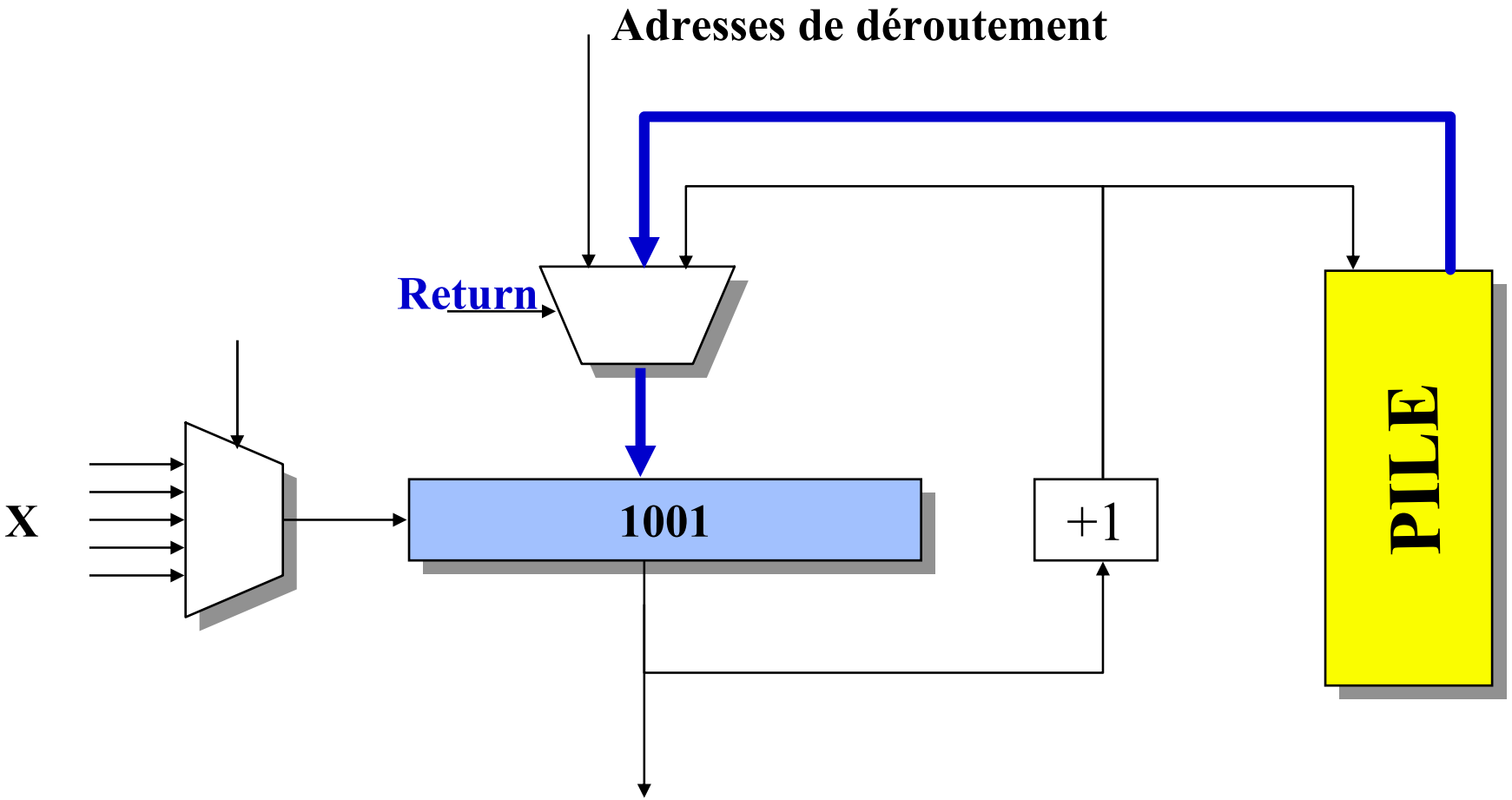
Machines dédiées et programmables

➢ instruction call factorielle suite



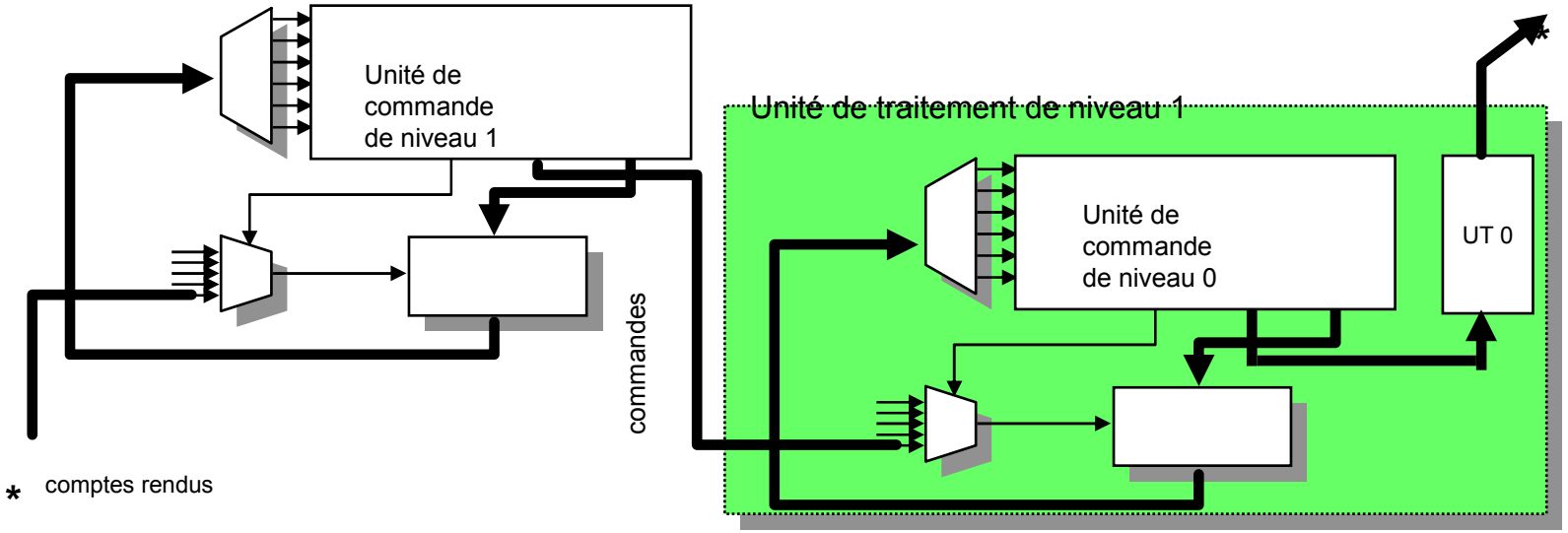
Machines dédiées et programmables

➢ instruction return



Principe de la hiérarchisation

- Une machine M0 :
  - dispose d'un langage L0
  - A l'aide de ce langage, on construit des programmes qui forment le langage L1
- La machine M1 :
  - est alors une machine virtuelle dans laquelle on pourra implanter des algorithmes à l'aide des instructions de L1

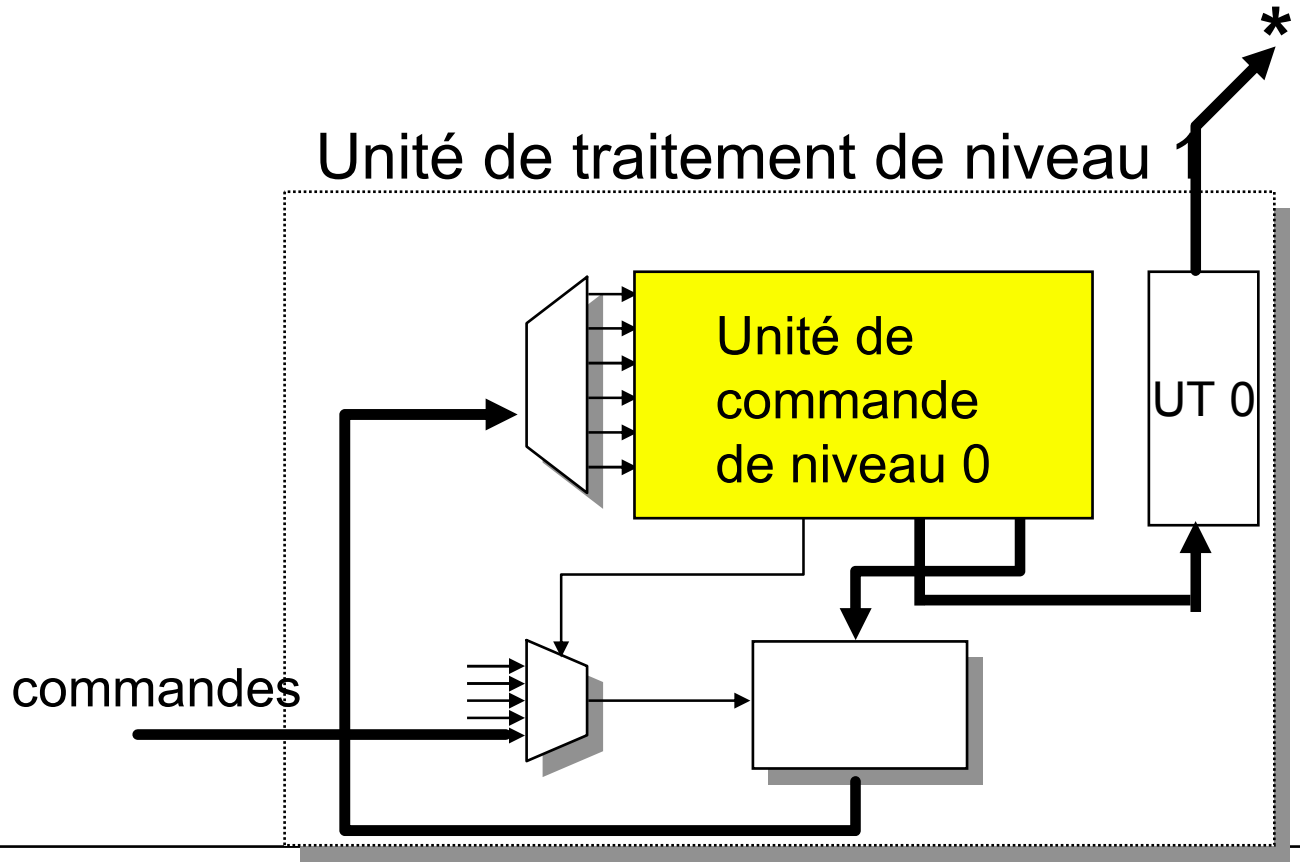


Machines dédiées et programmables

- Il existe plusieurs niveaux de séquencement :
  - il faut les synchroniser
- Le séquenceur de la machine M0 est simple:
  - il est réalisé de façon à travailler le plus rapidement possible
  - il ne sait pas gérer les mécanismes d'appel de sous programme (stockage de l'adresse de retour, passage de paramètres, etc)
- Le séquenceur de la machine M1 est plus complexe :
  - offre des modes de séquencement complexes, des instructions complexes
  - permet de bien structurer le programme

Machines dédiées et programmables

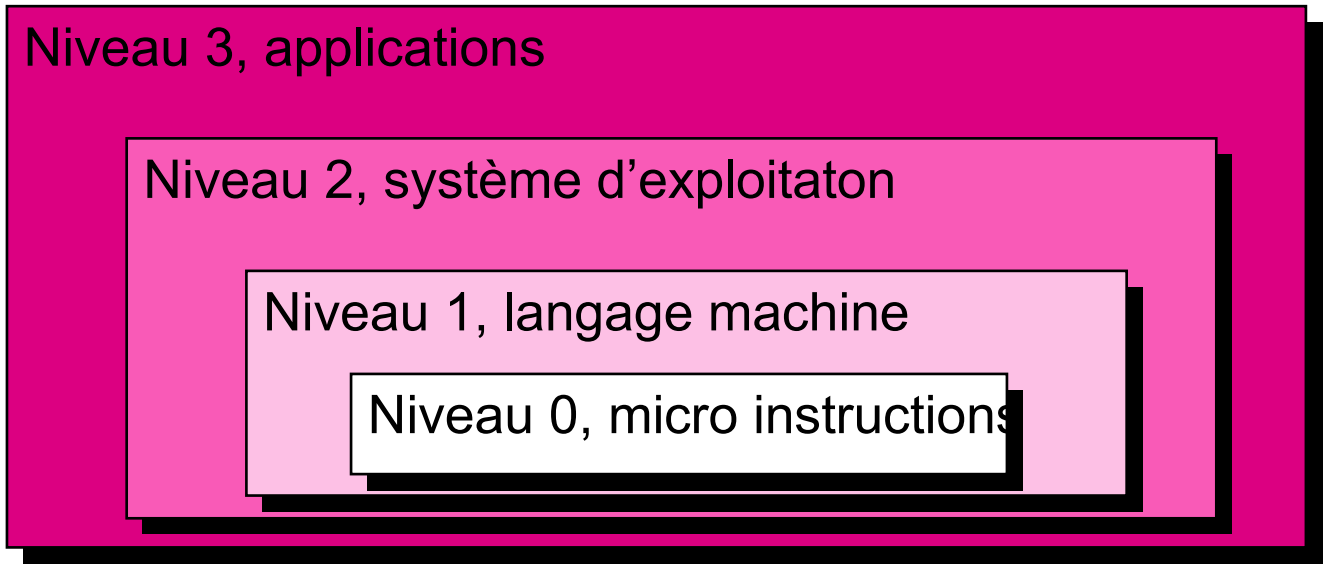
- L'environnement fournit les commandes (instructions) à réaliser
- Le composant mémoire contient l'ensemble des micro programmes permettant l'exécution des instruction du niveau supérieur
- Un micro programme est constitué de micro instructions
- Une micro instruction contient les micro commandes destinées à l'UT0



Machines dédiées et programmables

❑ Extension du principe de hiérarchisation

- ➢ représentation d'un ordinateur sous la forme d'une machine à 4 niveaux hiérarchique
  - ♦ 1 langage pour chaque niveau de la hiérarchie
  - ♦ instructions simples pour le niveau 0
  - ♦ instructions de plus en plus complexes pour les niveaux 1, 2 et 3
  - ♦ machine Mi, composée d'AEF de commande UCi, d'un AEF de traitement UTi, et définissant un langage Li



Machines dédiées et programmables

❑ Pour résumer :

➢ les machines dédiées :

- ♦ conçues par une ***approche descendante***
- ♦ construction de la partie opérative du système
- ♦ répondent strictement aux besoins de l'application
- ♦ performantes et efficaces

figées donc pas évolutives

- ♦ permettent la conception hiérarchique
- ♦ pas intéressant pour la réalisation d'un prototype mais conviennent bien aux grandes séries :
  - – coût (surface, consommation, etc)
  - – protection du système
- ♦ conception d'ASIC
- ♦ temps de conception assez long

Machines dédiées et programmables

- **Pour résumer :**
  - les machines algorithmiques programmables :
    - conçues par une approche ascendante :
      - programmation
    - flexibilité
    - évolutives
    - utilisation non optimale des possibilités de la machine
      - on utilise pas toutes les instructions disponibles
      - on doit programmer les instructions qui n'existent pas
        - dans les premiers processeurs, pas de multiplication
      - on s'arrange pour atteindre la précision souhaitée
        - exemple : addition 16 bits (A+B) sur processeur 8 bits

          AL + BL = SL + Retenue
          AH + BH + Retenue = SH

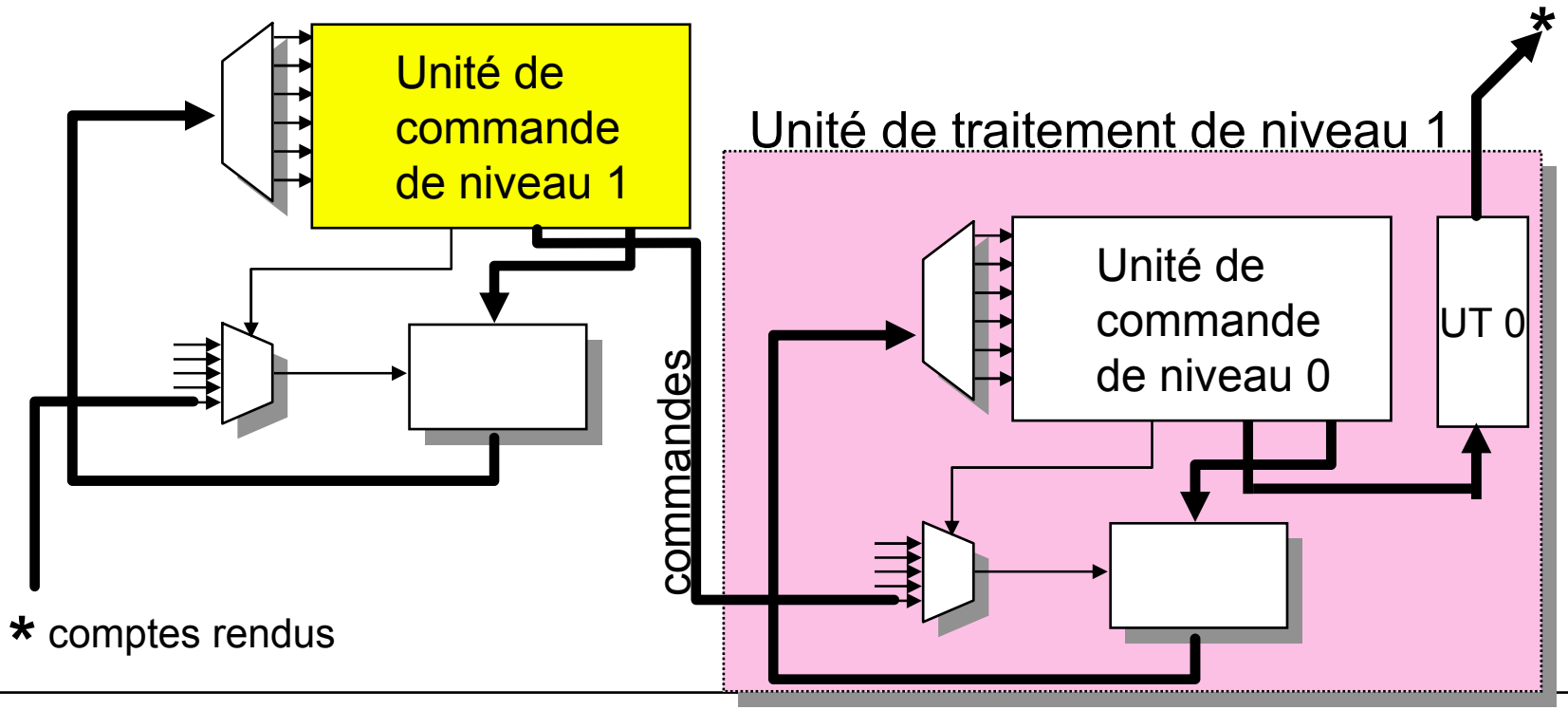
Machine Von Neumann

Machine Von Neumann

❑ Les processeurs Von Neumann :

➢sont des machines à 2 niveaux

- ♦ niveau 0 contient les programmes d'interprétations des instructions du niveau 1
- ♦ niveau 1 permet de constituer le programme application



Machine Von Neumann

- **L'AEF de commande de niveau 1 :**
  - ➢1) recherche le code instruction
  - ➢2) décode l'instruction à réaliser
  - ➢3) exécute l'instruction sur l'AEF de traitement de niveau 0
    - ♦ l'AEF de commande de niveau 0 :
      - – reçoit le code de l'instruction
      - – exécute le micro programme de l'instruction
  - ➢4) stocke les résultats des calculs

CYCLE VON NEUMANN

Machine Von Neumann

- **Recherche du code de l'instruction :**
  - ➢ le processeur accède à la mémoire et charge l'instruction dans un registre d'instructions
  - ➢ l'adresse accédée est l'adresse courante du programme et est stockée dans un registre de compteur de programme PC
  - ➢ ce registre d'adresses évolue de 1 en 1 sauf dans le cas de branchement, où il est chargé avec une autre valeur

Machine Von Neumann

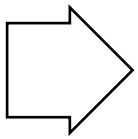
❑ décodage de l'instruction :

➢ il s'agit de savoir quelle instruction doit être réalisée :

➢ une instruction se décompose en :

♦ un code opération

♦ des opérandes



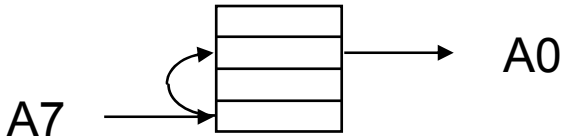
code opératoire
opérande 1
opérande 2

Machine Von Neumann

❑ Exécution de l'instruction :

➢ décomposition de l'instruction en micro instructions par la machine de commande de niveau 0

♦ move 2(A7), A0



➢ envoie des bons signaux de contrôle vers l'UT

Machine Von Neumann

❑ Stockage des résultats de calculs :

- ➢ adressage de la mémoire
- ➢ transferts des données vers la mémoire
- ➢ mise à jour de l'état du processeur :
  - ♦ mémorisation des drapeaux

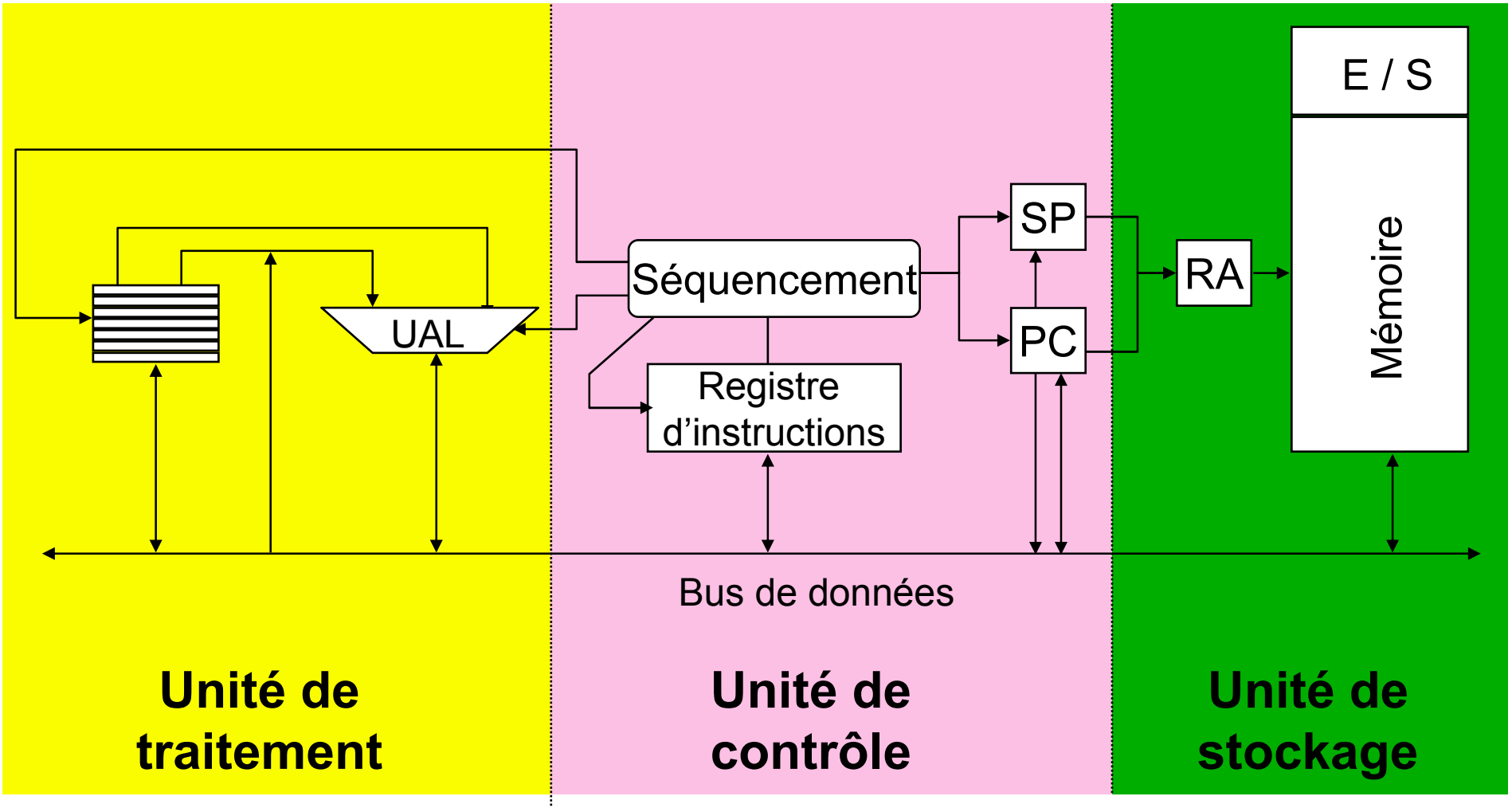
Machine Von Neumann

Exécution d'une instruction plus détaillée :

- 1) chargement de la prochaine instruction depuis la mémoire vers le registre d'instruction
- 2) modification du compteur ordinal (PC) pour qu'il pointe sur l'instruction suivante
  - PC = PC + 1
- 3) décodage de l'instruction
- 4) localisation des éventuelles données nécessaires pour l'instruction
- 5) chargement des données dans les registres internes de l'unité centrale
- 6) exécution de l'instruction
- 7) stockage des résultats dans leurs cases mémoire respectives
- 8) passage à l'instruction suivante, retour en 1

Machine Von Neumann

Les différentes unités

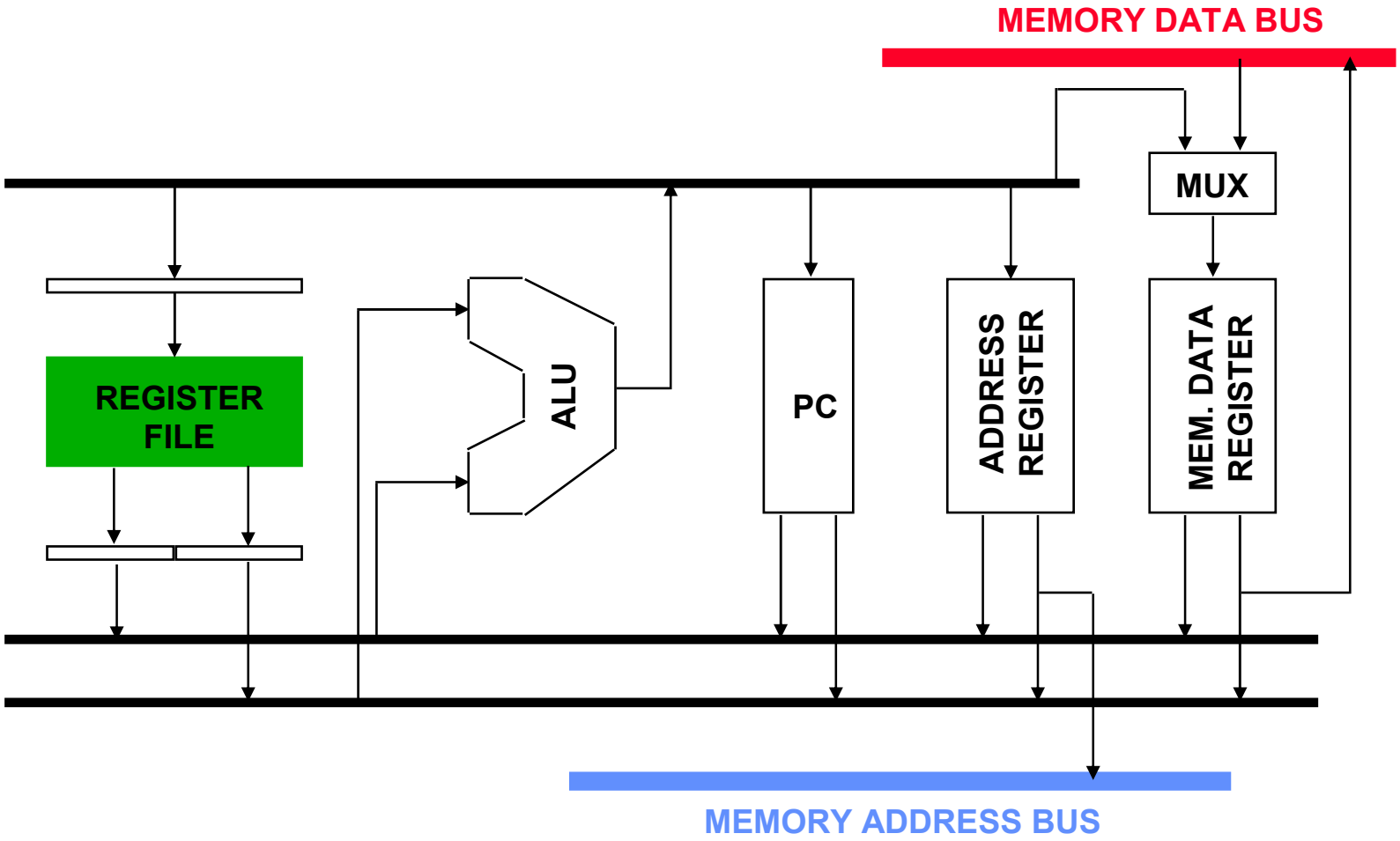


Machine Von Neumann

- **Unité de stockage :**
  - ➢ contient les informations relatives :
    - ♦ au programme, donc à l'application
    - ♦ aux données à traiter
- **Unité de traitement :**
  - ➢ constituée d'UAL et de registres de travail
  - ➢ les registres contiennent les informations en cours de traitement :
    - ♦ registres contenant des données
    - ♦ registres contenant l'état de la machine
- **Unité de contrôle :**
  - ➢ coeur de la machine
  - ➢ réalise le séquencement, déroulement du programme :
    - ♦ décode l'instruction située dans le registre RI
    - ♦ adresse la mémoire par le compteur de programme
    - ♦ le registre d'adresses permet d'accéder aux données

Machine Von Neumann

- Autre représentation de la machine Von Neumann



Machine Von Neumann

- **Remarques :**
  - la taille du bus d'adresses est généralement plus importante que la taille du bus de données
  - la taille du bus d'adresses définit l'étendue de l'espace adressable :
    - bus sur N bits ⇨ 2^N mots adressable
  - pour former une adresse, il faut plusieurs accès à la mémoire

Machine Von Neumann

❑ L'exécution des instructions :

➢ on distingue 3 types d'instructions :

- ♦ instruction de contrôle :
  - – branchement, appel de procédure
- ♦ instruction de traitement :
  - – arithmétique, logique
- ♦ instruction de transfert :
  - – entre registres
  - – entre registres et case mémoire

Machine Von Neumann

❑ Instruction de contrôle :

➢ modification du déroulement du programme en fonction d'évènements :

- ♦ INTERNES au processeur : résultats de calcul, état de la machine
- ♦ EXTERNES au processeur : broches d'entrées / sorties

➢ modification du registre de compteur de programme PC

➢ 3 types de branchement :

- ♦ conditionnel ou inconditionnel
- ♦ explicite ou implicite
- ♦ avec ou sans restitution du contexte

Machine Von Neumann

- **Branchement conditionnel ou inconditionnel :**
  - ➢ conditionnel :
    - ♦ résulte d'un test, par exemple de la valeur d'un drapeaux qui indique l'état de la machine
  - ➢ inconditionnel :
    - ♦ saut quelque soit l'état du processeur

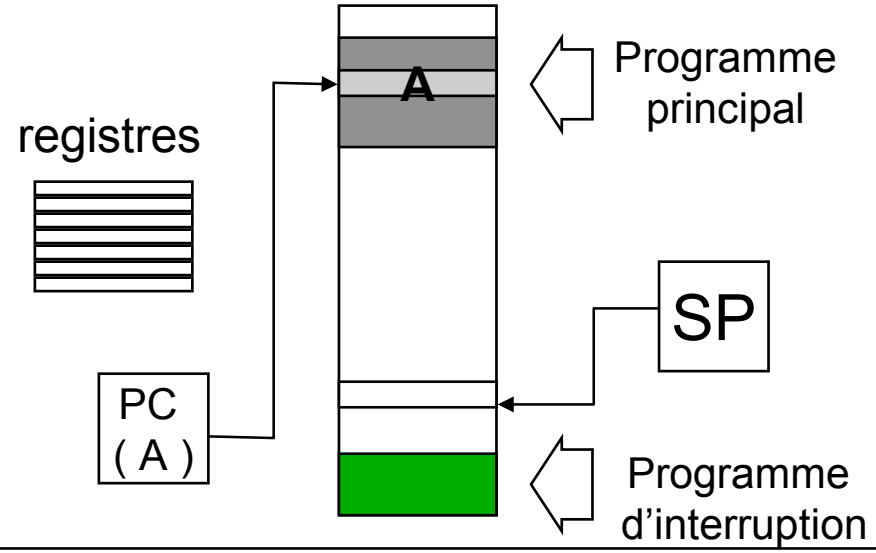
- **Branchement explicite ou implicite :**
  - ➢ explicite :
    - ♦ cas d'un jump, branchement conditionnel
  - ➢ implicite :
    - ♦ cas des interruptions ⇨ signal extérieur
    - ♦ cas des exceptions ⇨ par exemple division par 0

Machine Von Neumann

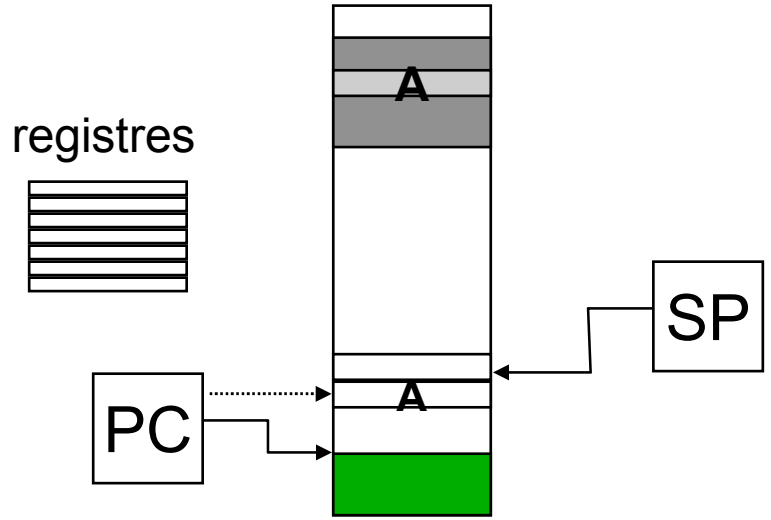
- **Branchement avec sauvegarde de contexte :**
  - lors de l'appel d'une interruption, il faut conserver l'adresse de retour pour assurer le bon déroulement du programme
  - le contexte minimum est constitué de :
    - l'adresse de l'instruction en cours d'exécution lors de l'appel de la procédure
    - charge à l'utilisateur de sauvegarder plus d'informations si cela est nécessaire

Machine Von Neumann

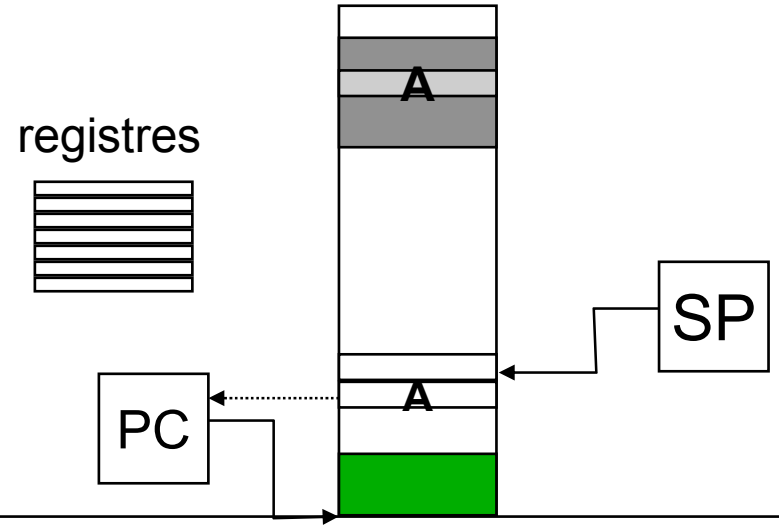
1) Avant l'interruption



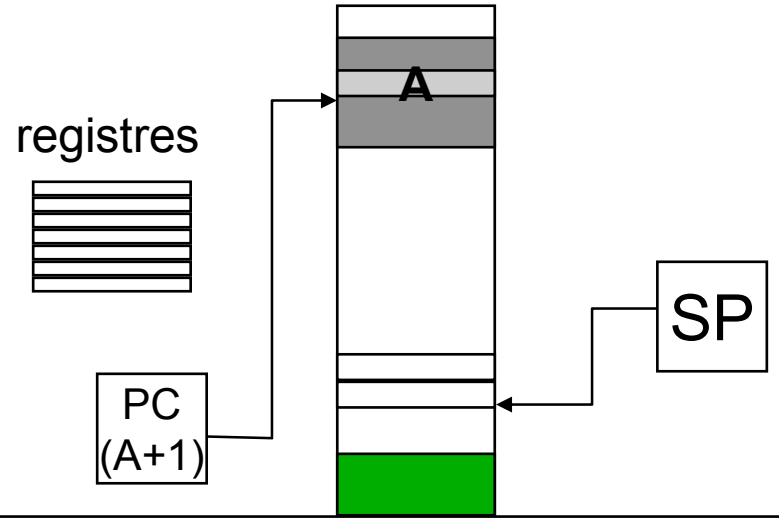
2) Interruption



3) Fin de l'interuption

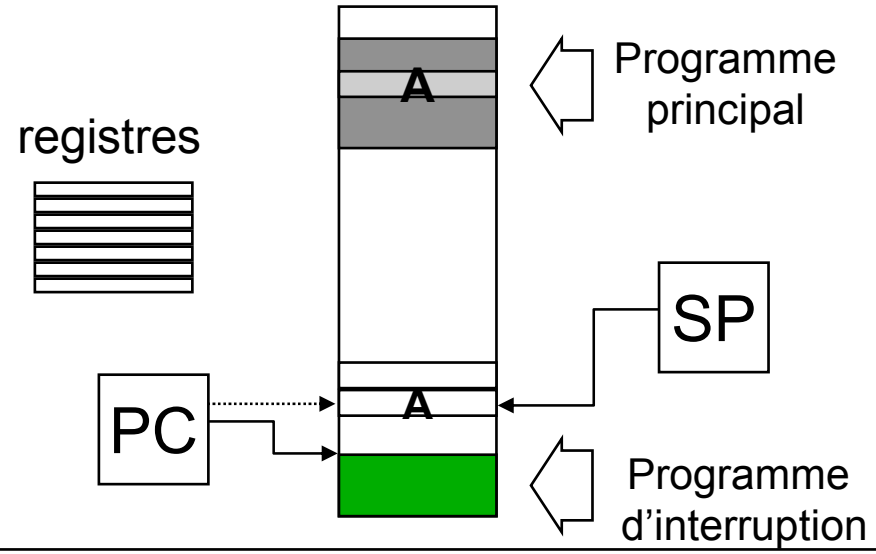


4) Retour au programme principal

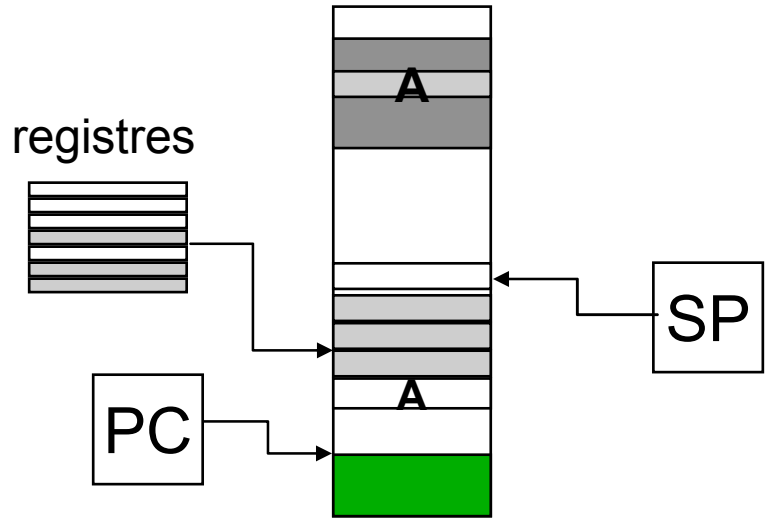


Machine Von Neumann

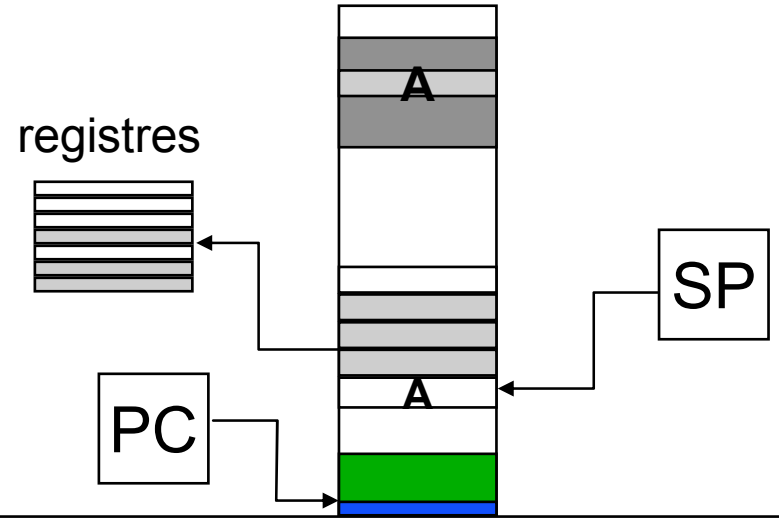
2) Interruption



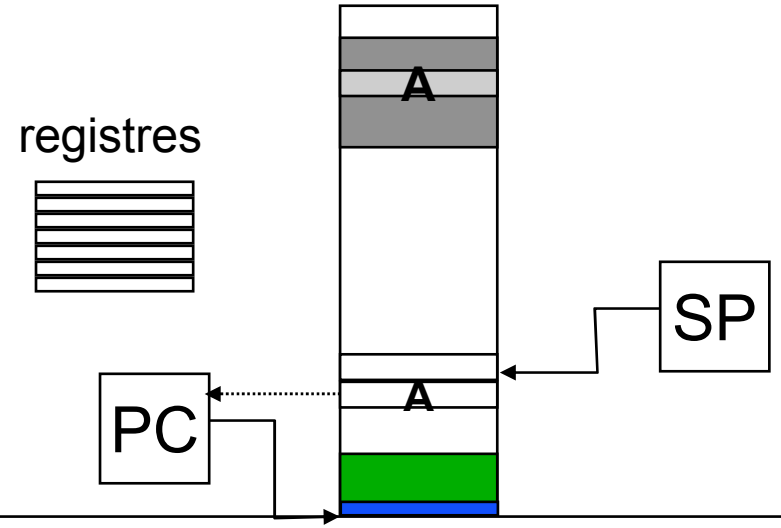
Sauvegarde de contexte plus généra



restitution de contexte plus général



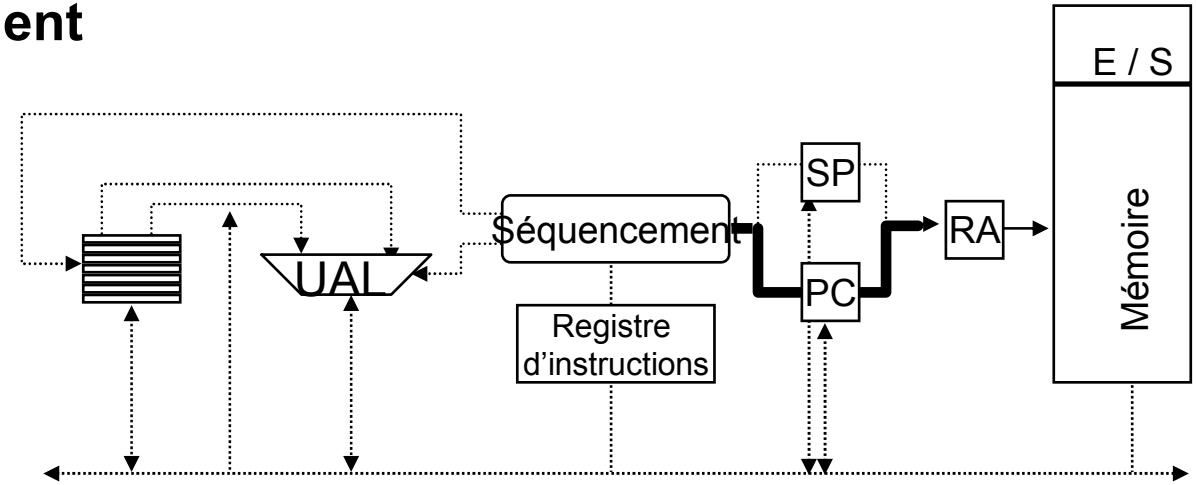
3) Fin de l'interuption



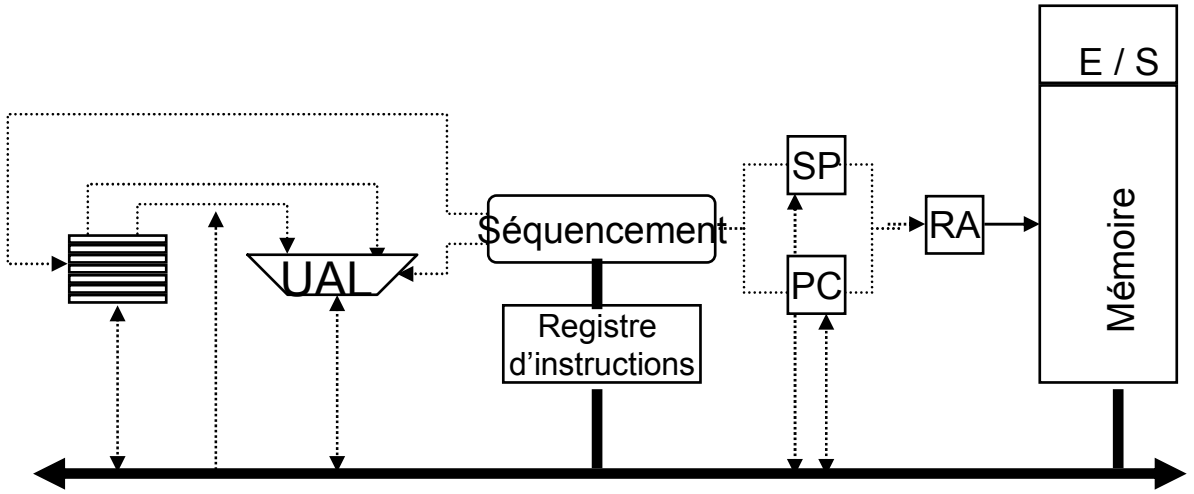
Machine Von Neumann

L'instruction de branchement

a) recherche d'une instruction



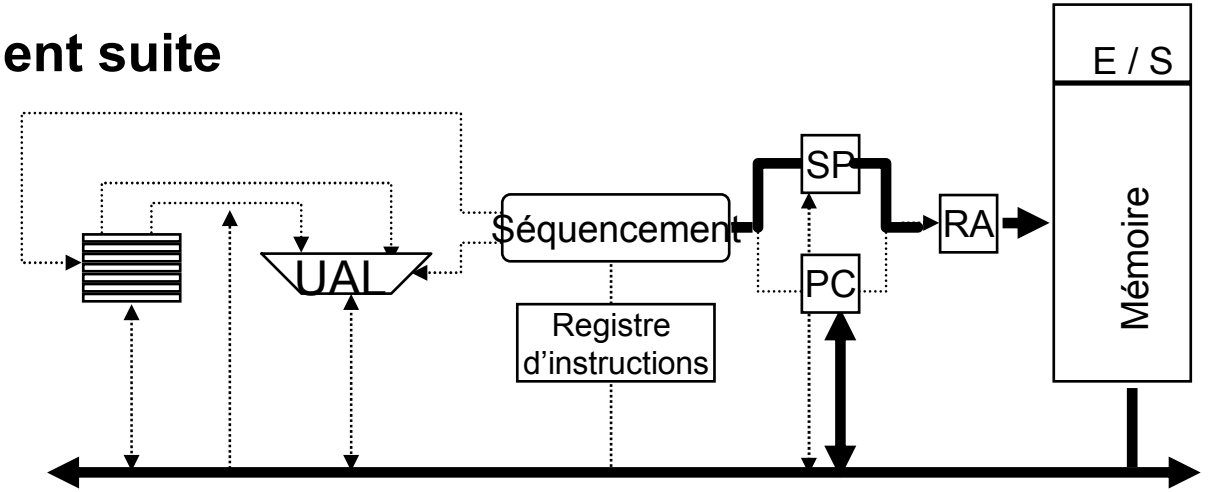
b) décodage de l'instruction



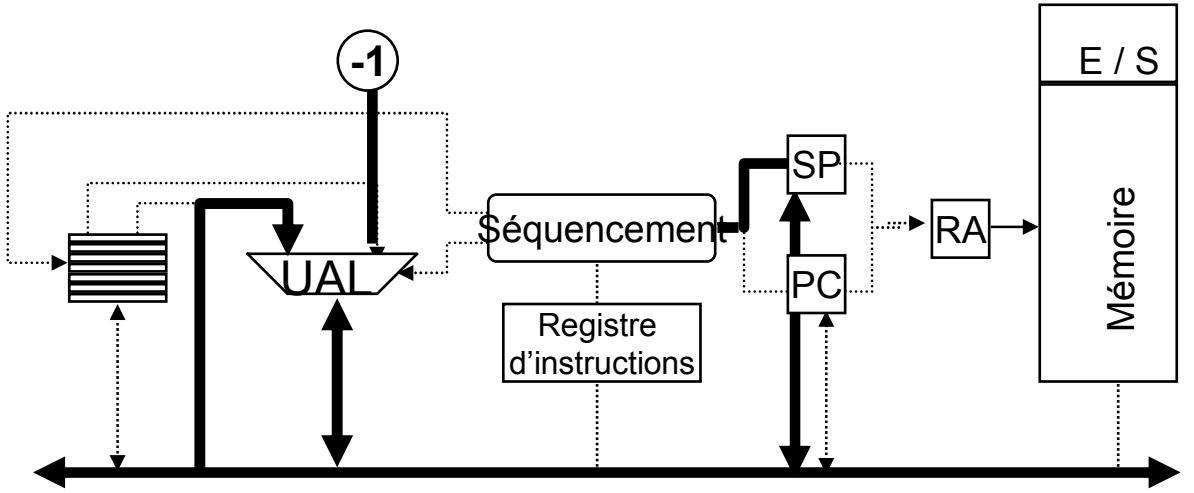
Machine Von Neumann

L'instruction de branchement suite

c) sauvegarde du contexte



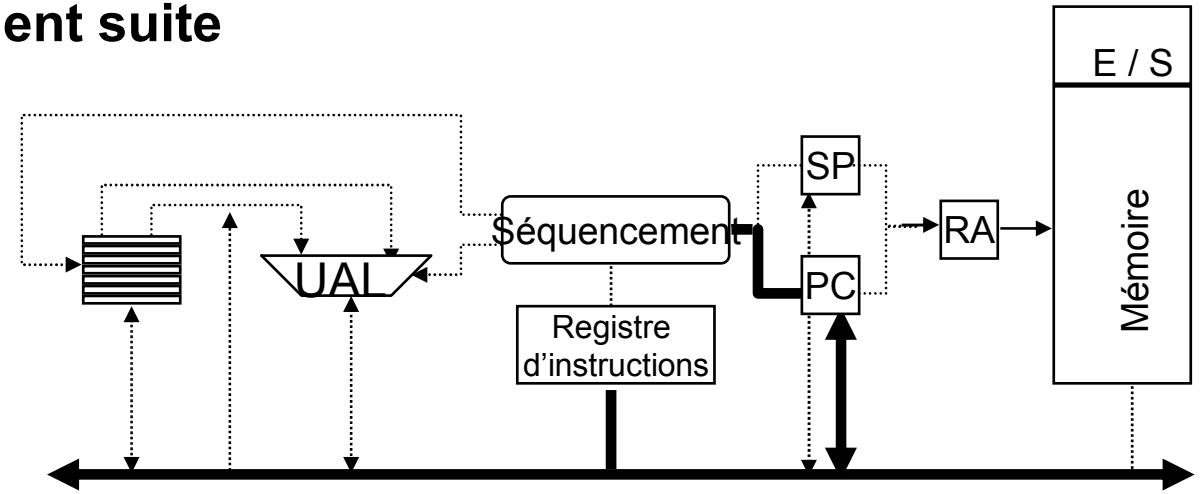
d) Modification du pointeur de pile



Machine Von Neumann

L'instruction de branchement suite

e) recherche l'adresse de débranchement



code	adresse de saut	Mot 1

Machine Von Neumann

❑ Instructions de traitement :

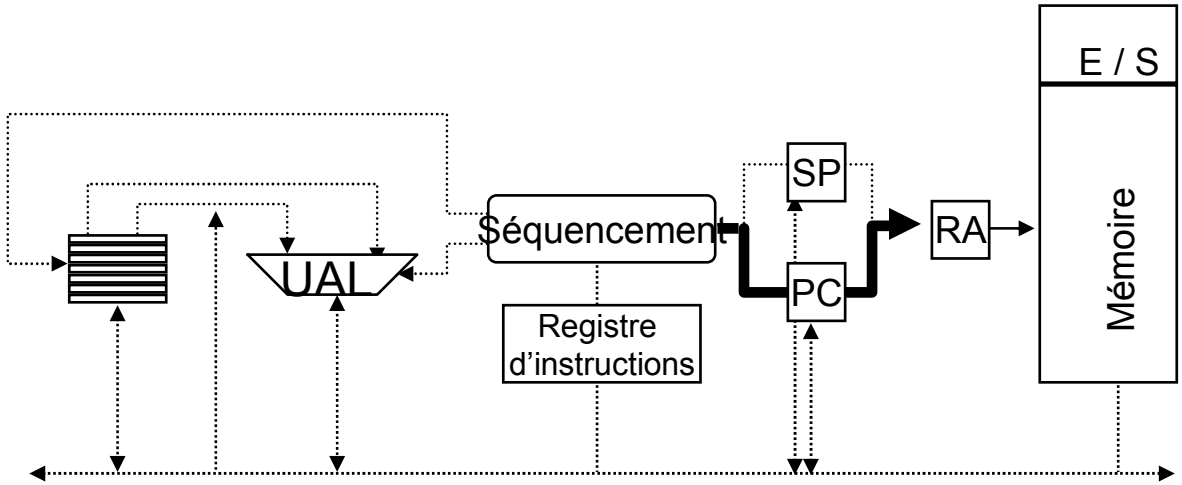
➢ traitement des données :

- ♦ opérations de calculs :
  - arithmétique
  - logique
- ♦ positionnement des drapeaux en fonction des résultats de calcul :
  - nul
  - négatif
  - débordement de calcul, problème de retenue
    - A, B sur 8 bits et opération A+B,
    - théoriquement A+B est un résultat sur 9 bits,
    - si la machine est une machine 8 bits, l'opération A+B va provoquer un débordement,
    - un bit de débordement récupère cette information et il est possible de tester la valeur de ce bit

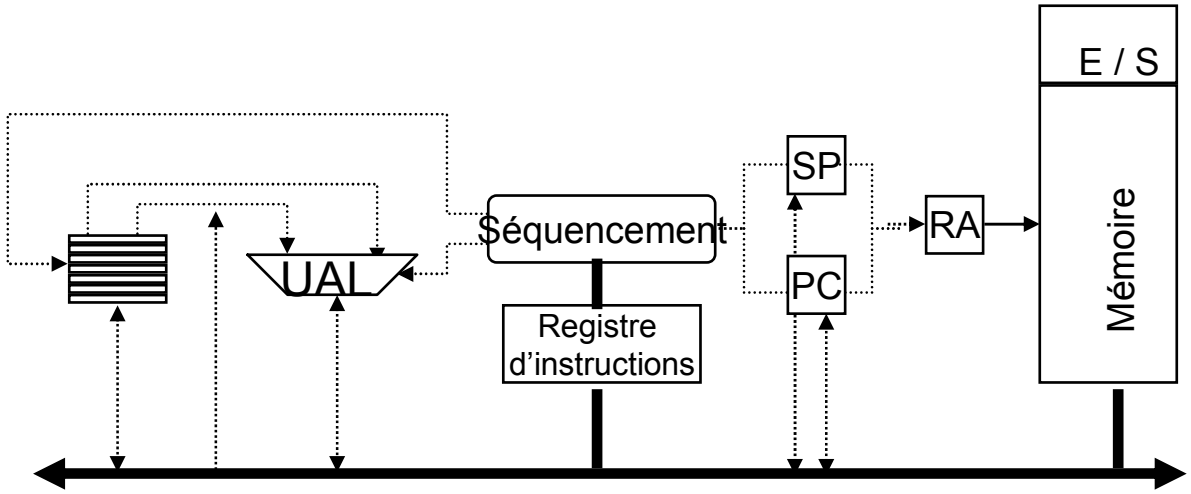
Machine Von Neumann

L'instruction de traitement

a) recherche d'une instruction



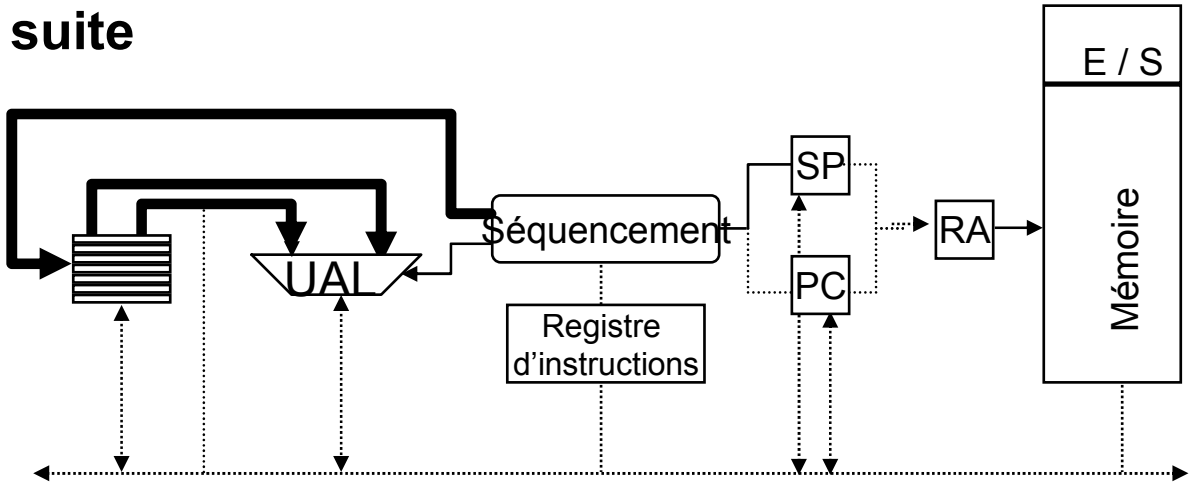
b) décodage de l'instruction



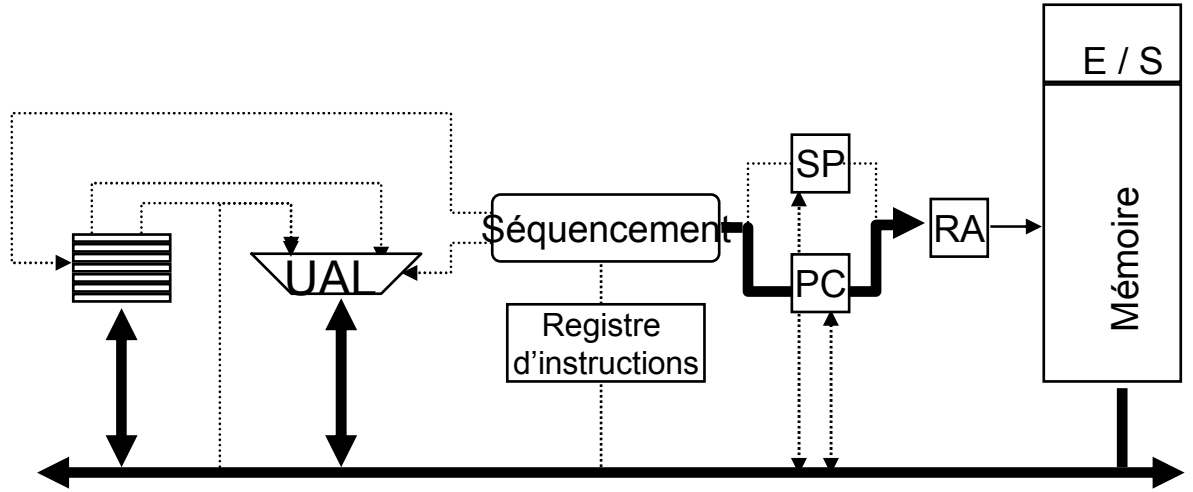
Machine Von Neumann

L'instruction de traitement suite

c) exécution de l'instruction



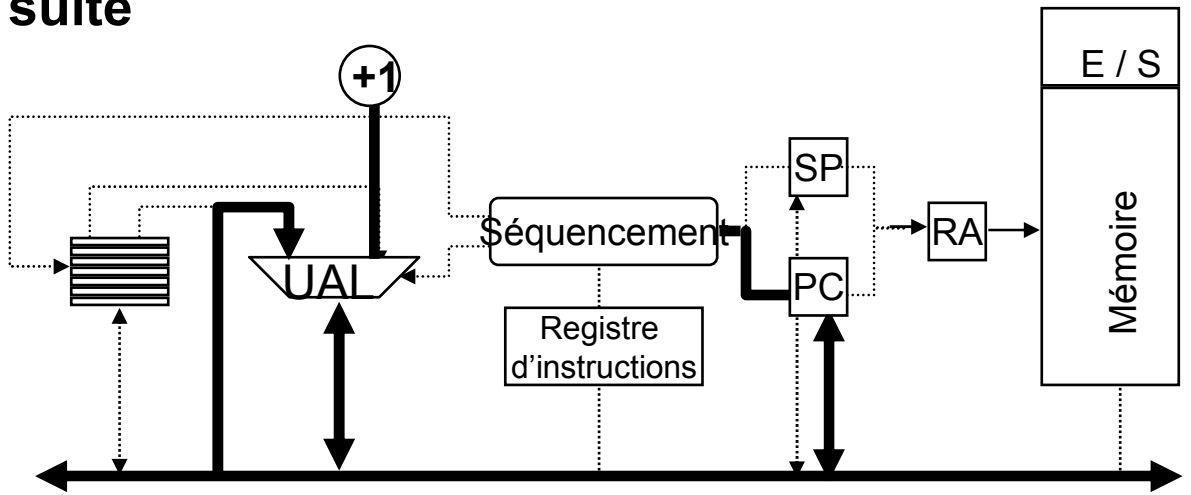
d) Stockage du résultat



Machine Von Neumann

L'instruction de traitement suite

e) passage à l'adresse suivante



Machine Von Neumann

Instructions de transferts :

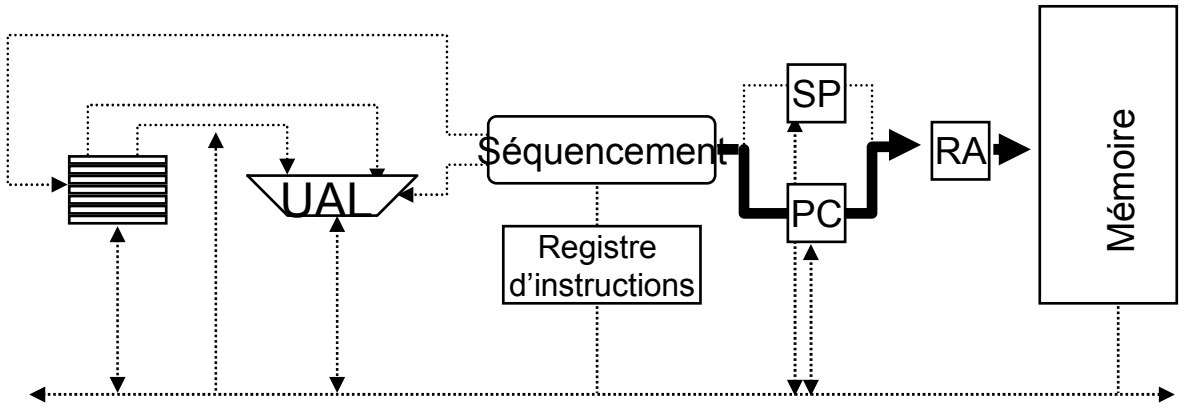
- Il existe au moins 3 types de transferts :
  - entre registres
  - entre mémoire et registre
  - entre deux cases mémoire

- L'adressage d'une donnée peut se faire de manière :
  - adressage immédiat :
  - adressage direct
  - adressage indirect
  - adressage indexé
  - etc

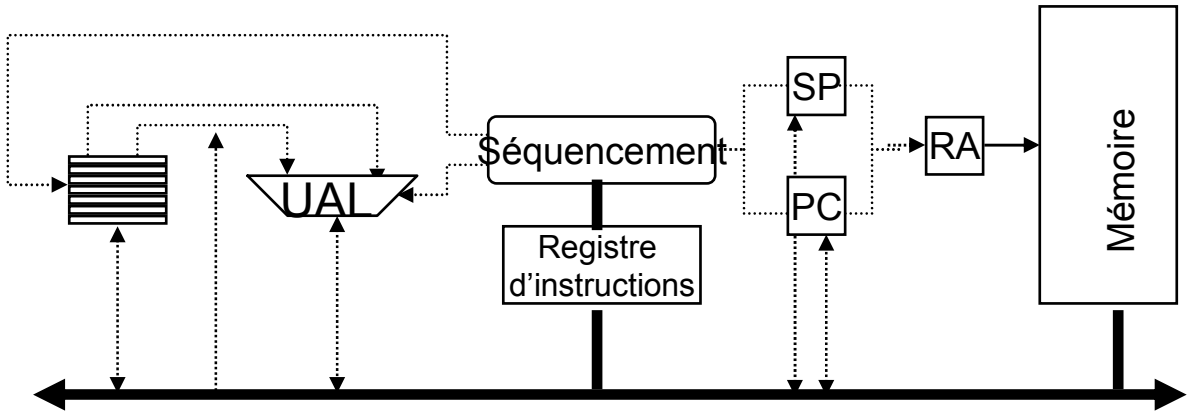
Machine Von Neumann

L'instruction de transfert

A) recherche d'une instruction



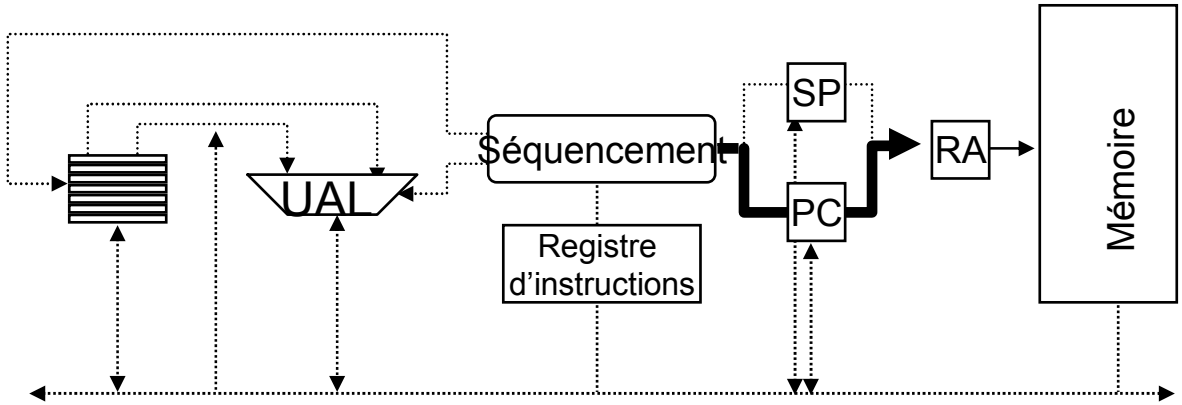
B) décodage de l'instruction



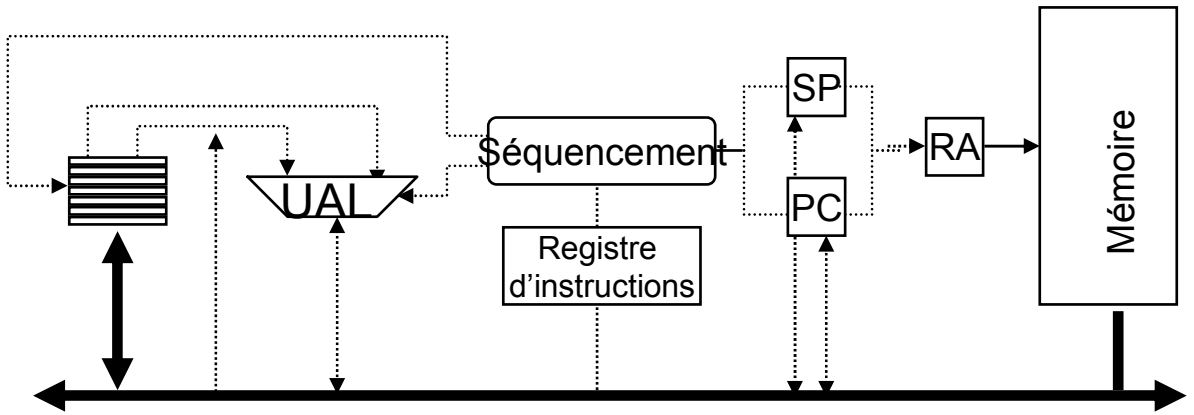
Machine Von Neumann

L'instruction de transfert suite

C) recherche de la donnée



D) transfert effectif



Machine Von Neumann

❑ Transfert registre à registre :

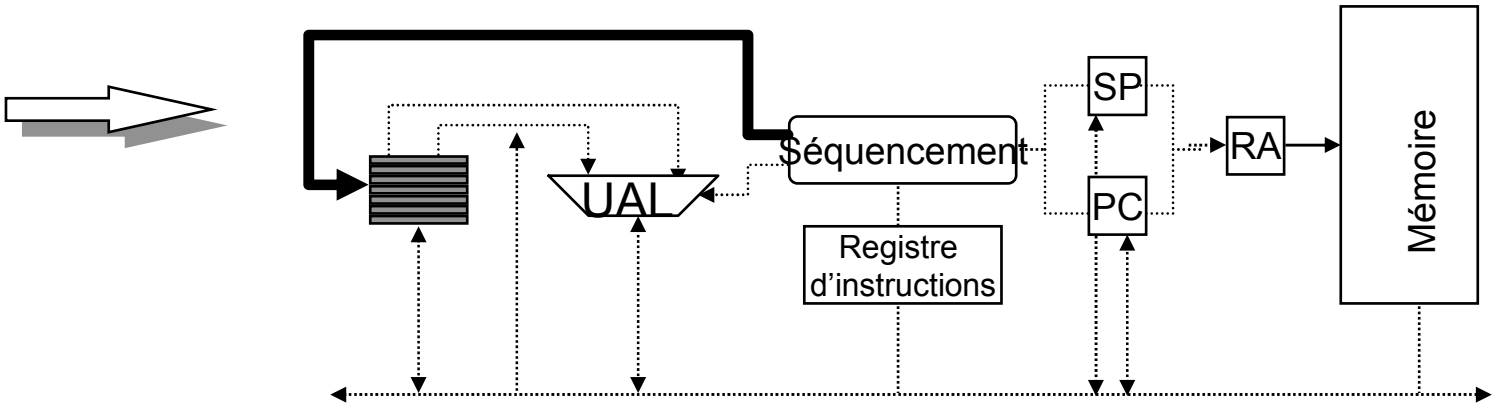
➢codage

code	registre destination	registre source

Mot 1

➢étape de l'instruction :

- ♦ A
- ♦ B
- ♦ D



- ♦ temps d'exécution : 3 unités de temps

Machine Von Neumann

❑ Adressage immédiat :

➢codage

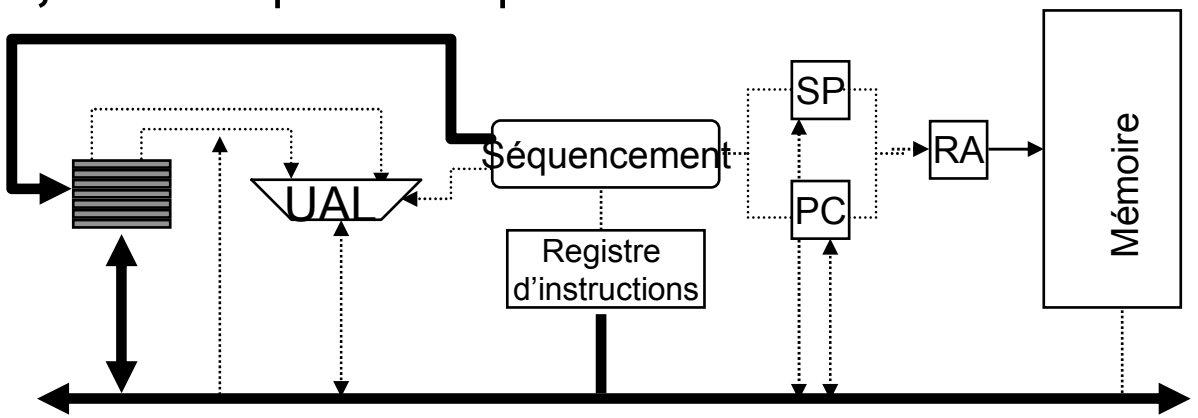
code	registre destination	Mot 1
donnée		Mot 2

➢type de données manipulées : ***constante***

➢étape de l'instruction :

- ♦ A
- ♦ B
- ♦ D

2 fois, une fois pour chaque mot de l'instruction



♦ temps d'exécution : 5 unités de temps

Machine Von Neumann

❑ Adressage direct :

➢codage :

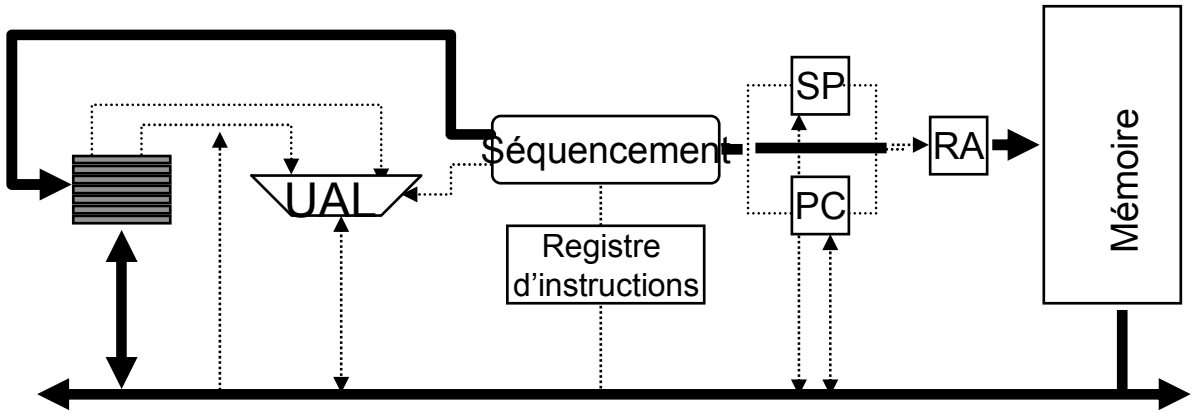
code	registre destination	Mot 1
Adresse partie 1		Mot 2
Adresse partie 2		Mot 3

➢type de données manipulées : ***variable, constante***

➢étape de l'instruction :

- ♦ A
- ♦ B
- ♦ C
- ♦ D

3 fois, une fois pour chaque mot de l'instruction



♦ temps d'exécution : 8 unités de temps

Machine Von Neumann

❑ Adressage indirect :

➢codage : indirect registre

code	registre (adr)	registre destination	Mot 1

➢autre codage : indirect mémoire

code	registre destination	Mot 1
Adresse partie 1		Mot 2
Adresse partie 2		Mot 3

➢type de données manipulées :

variable, tableau, pointeur, constante

➢étape des instructions :

- ♦ A, B, C, D indirect registre 4 UT
- ♦ (A, B) * 3, (C, D) * 3 indirect mémoire 12 UT

Machine Von Neumann

❑ Adressage indirect post incrémenté :

➢codage : indirect registre

code	registre (adr)	registre destination	Mot 1

➢autre codage : indirect mémoire

code	registre destination	Mot 1
Adresse partie 1		Mot 2
Adresse partie 2		Mot 3

➢type de données manipulées : ***tableau, pointeur, variable, constante***

➢étape des instructions :

- ♦ indirect registre : A, B, C, D, incrémentation du registre d'adresses
  5 UT
- ♦ indirect mémoire : (A, B) * 3, (C, D) * 3, incrémentation de l'adresse, stockage de la nouvelle adresse en mémoire
  17 UT

Machine Von Neumann

❑ Adressage indexé:

➢codage : indexé immédiat

code	registre base	registre destination	Mot 1
index			Mot 2

➢autre codage : indexé par rapport à un registre d'index

code	reg base	reg indexé	registre dest	Mot 1

➢type de données manipulées : ***variable***

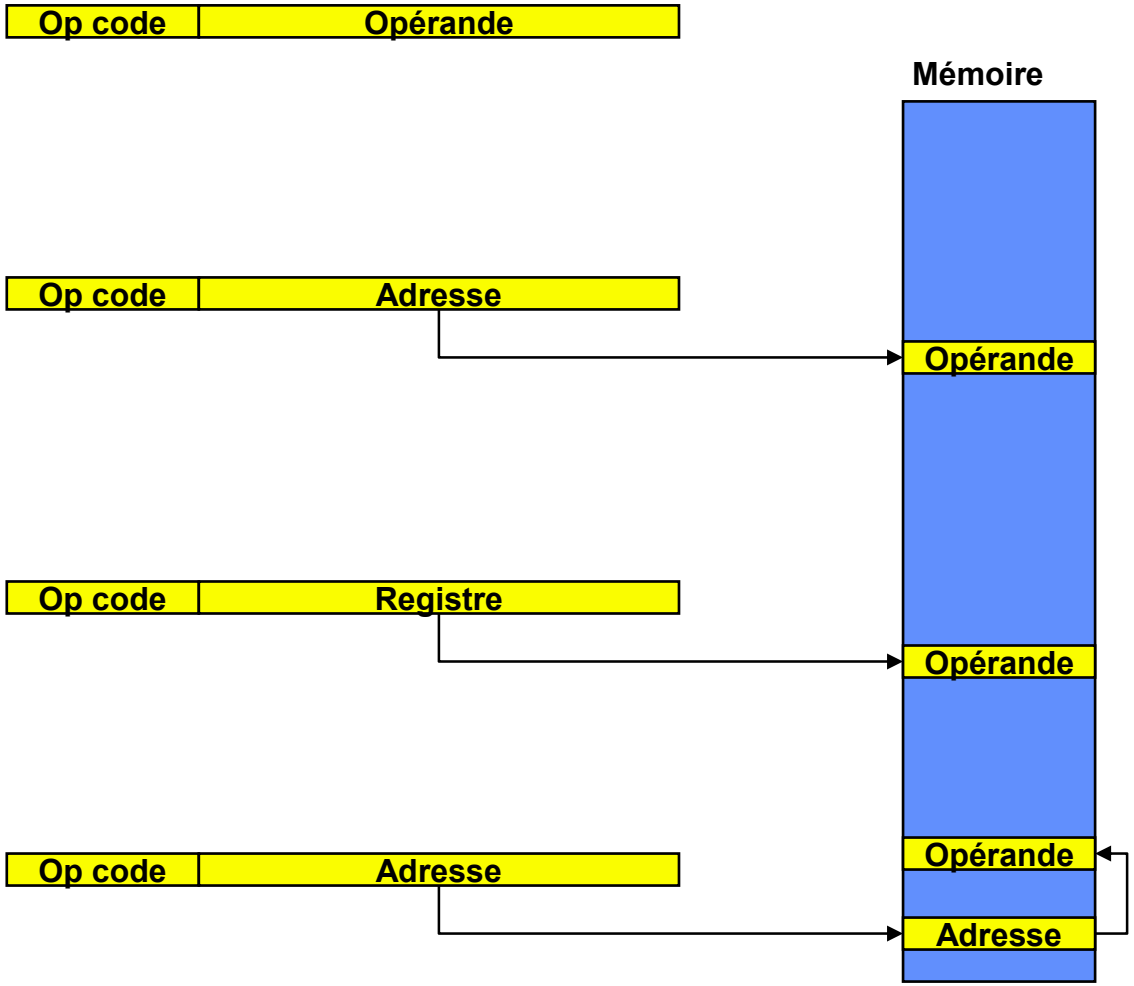
➢étape des instructions :

- ♦ indexé immédiat: (A, B) * 2, C, D, addition de base et index
  7 UT
- ♦ indexé par un registre: A, B, C, D, addition de base et index
  5 UT

Machine Von Neumann

❑ Pour résumer les adressages :

- ➢ Immédiat
- ➢ Direct
- ➢ Registre
- ➢ Indirect mémoire



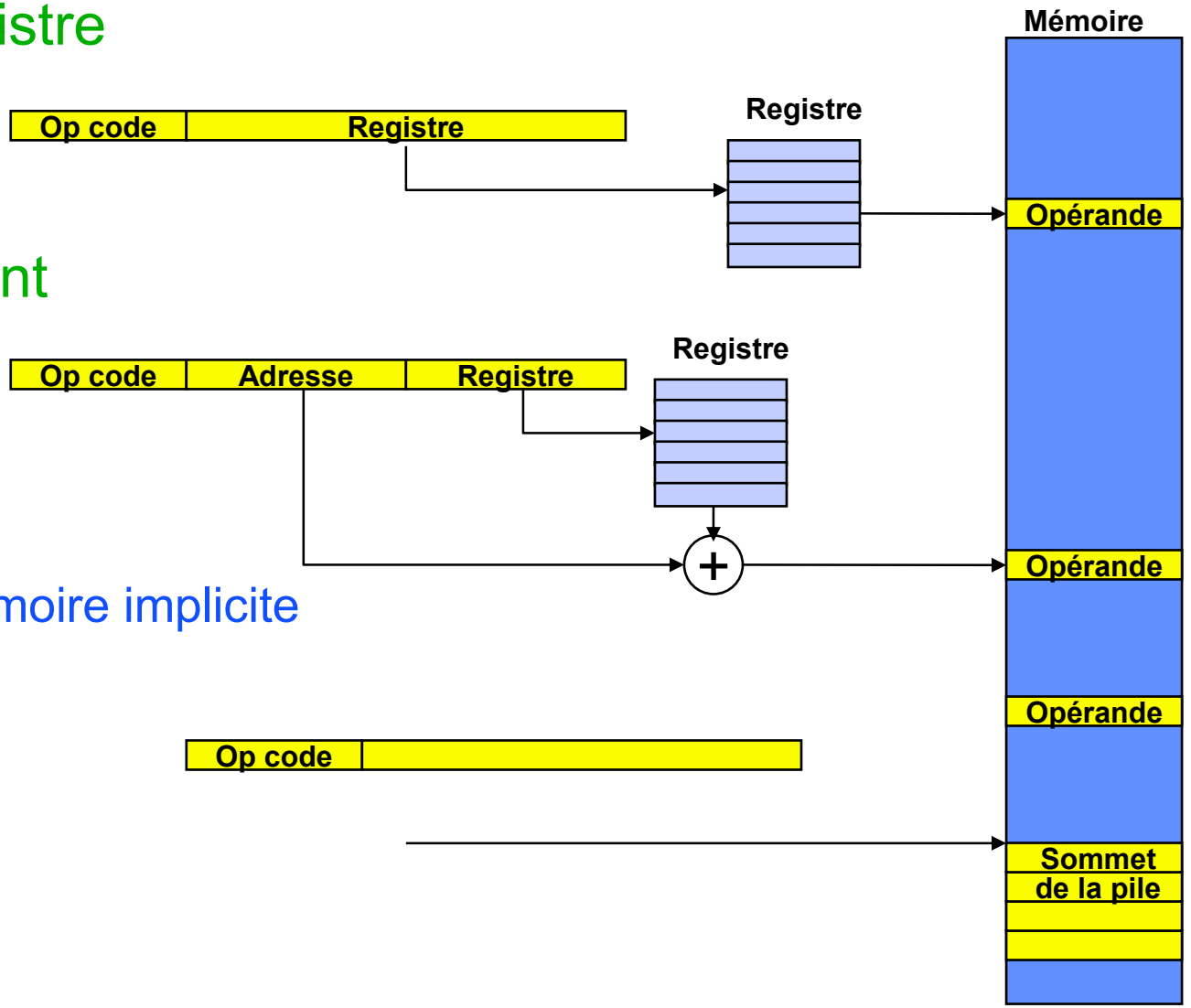
Machine Von Neumann

➢Indirect registre

➢Déplacement

➢Pile :

♦ accès mémoire implicite



Machine Von Neumann

❑ Caractéristiques du jeu d'instructions :

➢ taille du jeu d'instructions :

- ♦ pour les processeurs CISC : le nombre de mots pour coder une instruction est variable
- ♦ pour les RISC on tente de coder les instructions sur une longueur de mot ***fixe***

contrôleur plus simple

Machine Von Neumann

❑ Caractéristiques du jeu d'instructions (suite) :

➢orientation mémoire, registres, pile :

- ♦ mémoire : pas de registres visibles par l'utilisateur, toutes les opérations semblent se faire en mémoire
- ♦ registre : toutes les opérations se font sur des registres de l'unité de calcul
- ♦ pile : les opérations se font sur des données rangées dans la pile que l'on a préalablement chargée

Stack	Accumulator	Register-Memory	Load-store

Push A
Push B
Add
Push A
Mul
Pop C
	Load A
	Add B
	Mul A
	Store C
		Load R1,A
		Add R1,B
		Mul R1,A
		Store C,R1
			Load R1,A
			Load R2,B
			Add R3,R1,R2
			Mul R3,R3,R1
			Store C,R3

6 instr. 4 mem. op.	4 instr. 4 mem. op.	4 instr. 4 mem. op.	5 instr. 3 mem. op.

Machine Von Neumann

❑ Caractéristiques du jeu d'instructions (suite) :

- ➢ jeu d'instructions orthogonal :
  - ♦ indépendance entre les codes opérations et les modes d'adressages
  - ♦ toutes les instructions bénéficient de tous les modes d'adressages
  - ♦ en général, ces processeurs ont un nombre d'instructions plus petit
  - ♦ processeurs plus simples à programmer
  - ♦ développement d'un compilateur plus aisé
- ➢ nombre d'opérandes manipulées :
  - ♦ en général, les instructions sont à :
    - 0 opérande (NOP)
    - 1 opérande
    - 2 opérandes
    - dans certains cas, 3 opérandes

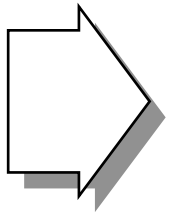
Machines RISC

Machines RISC

❑ Evolution des machines programmables :

➢ premiers processeurs :

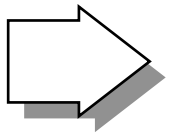
- ♦ instructions simples
- ♦ opérations sur les registres
- ♦ opérations sur les entiers



nb de cycles faible

➢ augmentation du nombre d'instructions

- ♦ instructions plus complexes
- ♦ opérations sur les flottants
- ♦ instructions qui se rapprochent d'un langage haut niveau



nb de cycles important

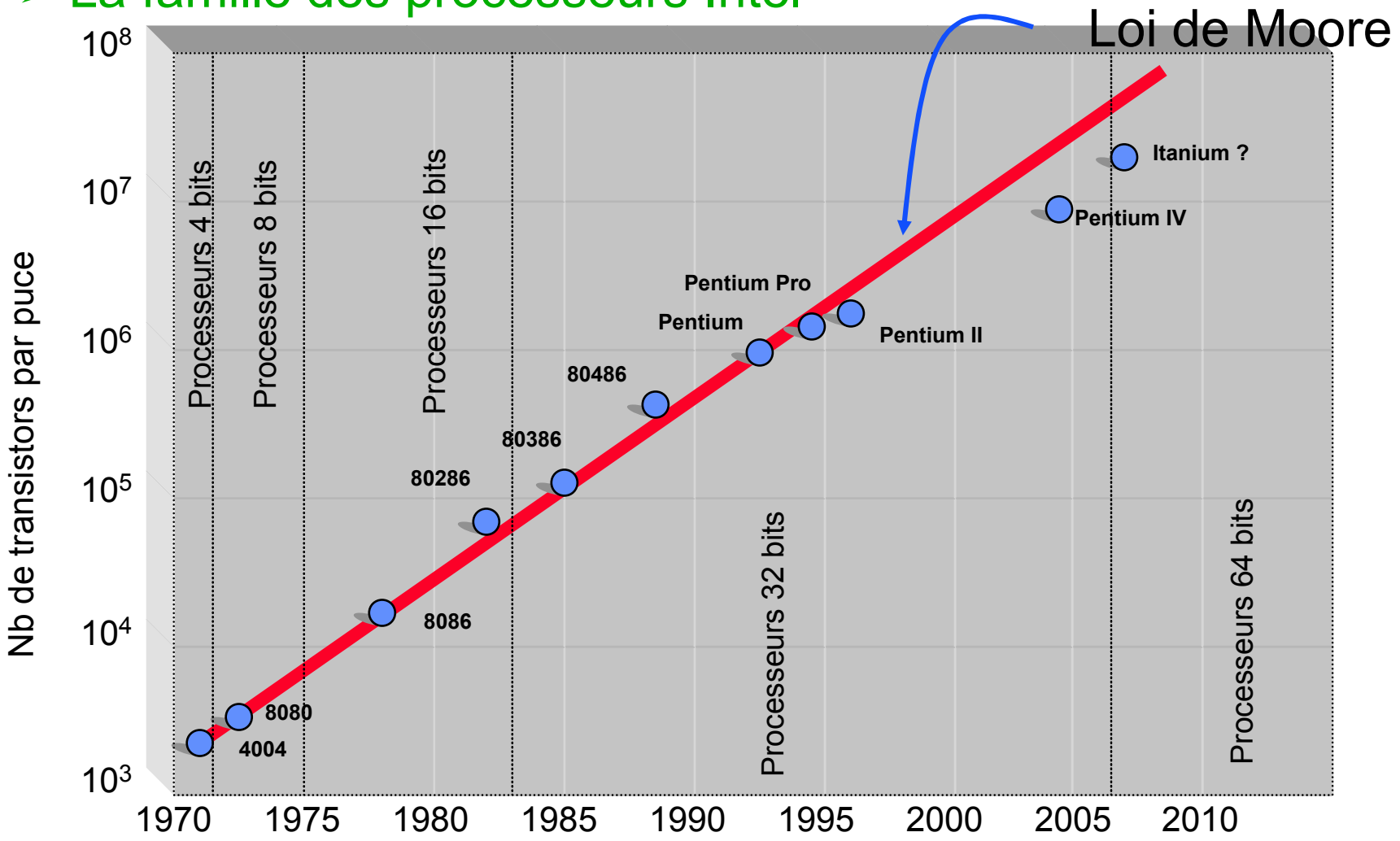
➢ machines très complexes

- ♦ contrôleur très volumineux

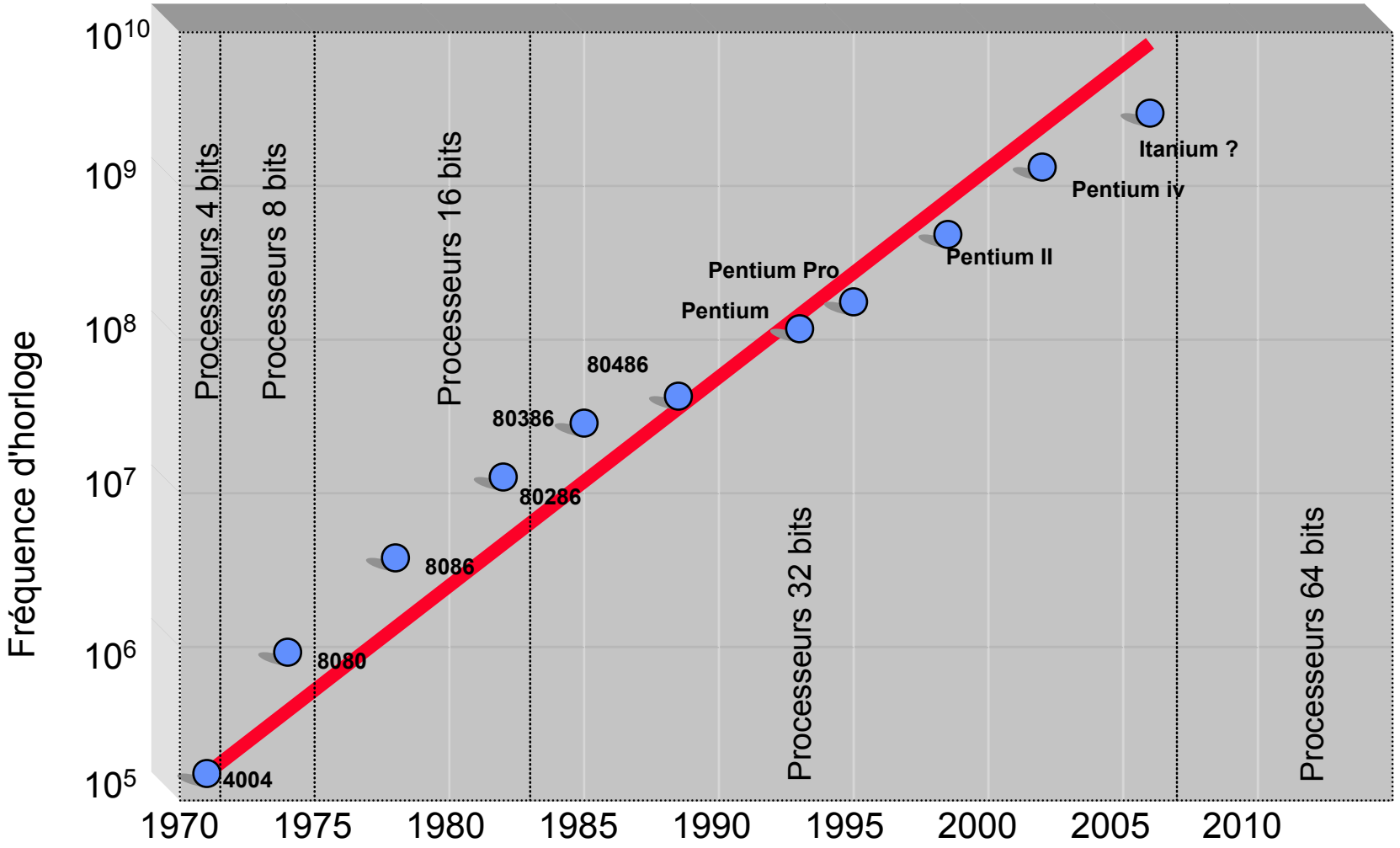
Processeur CISC
Complex Instruction Set Computer

Machines RISC

➢La famille des processeurs Intel



Machines RISC



Machines RISC

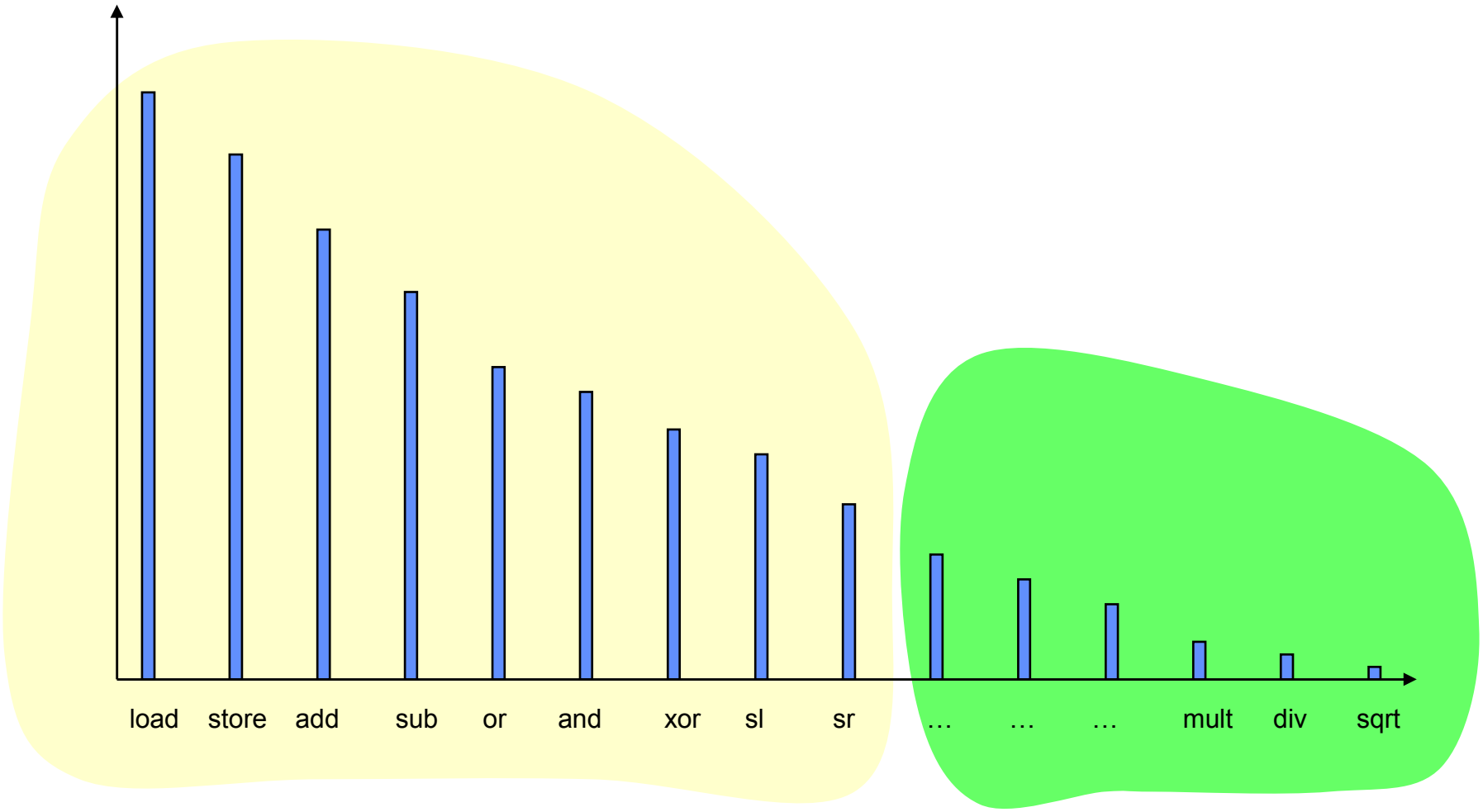
❑ Observation :

➢ les instructions les moins couramment utilisées sont celles qui engendrent le plus de contrôle :

- ♦ adressages complexes (pré-post incrémenté, basé, indexé, ...)
- ♦ tous les calculs peuvent s'exécuter avec des opérandes en mémoire (via tous les adressages)
- ♦ latences des instructions très variables
- ♦ difficulté à pipeliner l'exécution
- ♦ jeux d'instructions non orthogonales
- ♦ jeux d'instructions mal utilisés par les compilateurs

Machines RISC

➢ occurrence d'apparition des instructions



Machines RISC

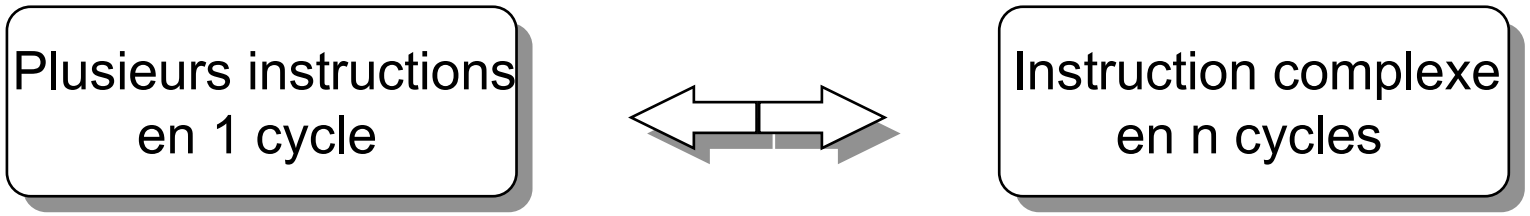
➢ Exemple d'occurrences des instructions pour un 8086

Rank	Instruction	% total execution
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move reg-reg	4%
9	call	1%
10	return	1%
Total		96%

Machines RISC

- **Règle des 90 - 10 :**
  - ➢ par une analyse des exécutions des programmes, on remarque que seules

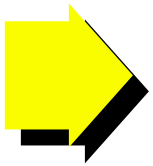
    10 % des instructions sont utilisées 90 % du temps.
  - ➢ les 90 % d'instructions inutilisées coûtent cher en temps et en surface silicium
    - ♦ **Idée** : on limite le nombre d'instructions à celles qui sont le plus utilisées
    - ♦ On réalisera les instructions complexes par programmation
  - ➢ instructions réalisables en 1 cycle ⇒ horloge rapide



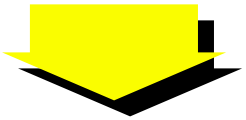
Processeur RISC
Reduce Instruction Set Computer

Machines RISC

- **RISC :**
  - ➢instructions simples
  - ➢instructions rapides



Contrôleur simple



Gain en surface important

➢Que faire de la surface disponible ?

- ♦ augmenter le nombre de registres
- ♦ augmenter le nombre d'UAL  parallélisme
- ♦ placer des unités spécifiques
- ♦ mettre de la mémoire cache dans le processeur
  - – temps d'accès très court
  - – profiter des réutilisations des données

Machines RISC

Historique

- Cahier des charges du premier RISC
  - toutes instructions en 1 cycle
  - accès mémoire par instructions load store
  - opérations addition comparaison entre registres
  - nombre de mode d'adressage réduit
  - même format pour toutes les instructions
  - séquenceur câblé
  - très grand soin apporté à la réalisation du compilateur
- Conséquences
  - le câblage du séquencement redevient possible puisque instructions simples
  - séquenceur rapide et peu coûteux
  - surface silicium libérée, utilisation pour d'autres fonctionnalités
  - gestion matérielle du pipeline plus simple puisque de nombreux problèmes sont reportées sur le logiciel
- Projet RISC I et II de Berkeley
  - RISC I : 1982, 39 codes opérations, abandonnée pour cause d'erreur de conception
  - RISC II : 55 codes opérations, 12 MHz, 41 000 transistors, UAL 32 bits

Machines RISC

❑ Caractéristiques des premiers processeurs RISC

Processeurs	IBM 801	RISC 1	RISC2
Années	1980	1982	1983
Nb instructions	120	39	55
Taille instructions	32		

Machines RISC

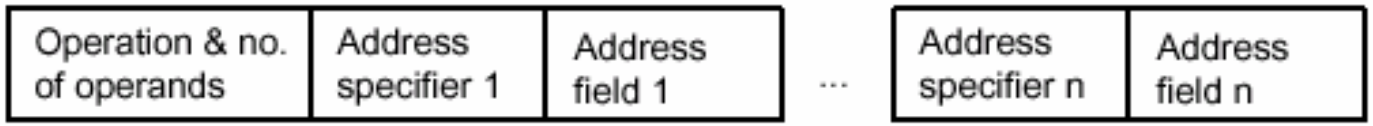
❑ Comparaison CISC - RISC

	CISC			RISC	
Nom	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000
Années	1973	1978	1989	1987	1991
Nb instructions	208	303	235	69	94
Taille instructions	2-6	2-57	1-11	4	4
Nb de mode d'adressage	4	22	11	1	1
Nb registres	16	16	8	40-520	32
Mémoire de contrôle	420	480	246	-	-
Taille cache	64	64	8	32	128

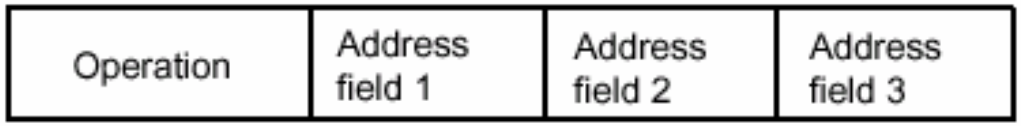
Machines RISC

❑ Exemple : taille des instructions variables / fixes

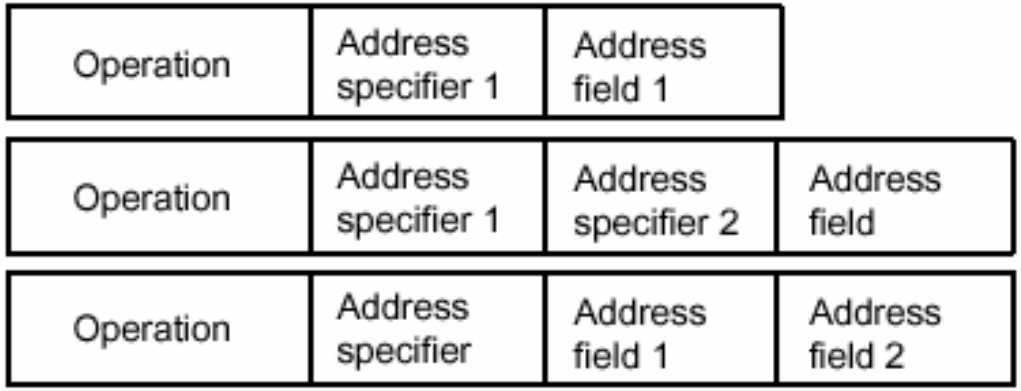
a) Variable (e.g. VAX)



b) Fixed (e.g. DLX, MIPS, PowerPC,...)



c) Hybrid (e.g. IBM 360/70, Intel80x86)



Machines RISC

RISC	CISC
Instructions simples ne prenant qu'un cycle	Instructions complexes prenant plusieurs cycles
Seules les instructions LOAd et STORE font des accès à la mémoire	Toutes les instructions peuvent faire des accès à la mémoire
Traitement pipeline	Peu ou pas de traitement pipeline
Instructions exécutées par le matériel	Instrucitons interprétées par un micro programme
Instructions au format fixe	Instructions en format variable
Peu d'instructions et de modes d'adressage	Beaucoup d'instructions et de modes d'adressage
Toute la complexité est dans le compilateur	Toute la complexité est dans le micro programe
Plusieurs jeux de registres	Un seul jeu de registres

Machines RISC

❑ Les tendances des processeurs RISC et CISC

- ➢ pipelines profonds :
  - ♦ jusqu'à 20 étages pour le Pentium IV
- ➢ superscalaire :
  - ♦ unités fonctionnelles indépendantes
  - ♦ unité de *disptaching* (jusqu'à 6 instructions lancées par cycle pour le R10000)
- ➢ fréquence d'horloge élevée :
  - ♦ bientôt 3 Ghz
- ➢ architectures 32 ou 64 bits
- ➢ hiérarchie mémoire complexe :
  - ♦ bus de données entre cache et buffer large
  - ♦ premier et second niveau de cache intégrés (on chip)
- ➢ jeux d'instructions étendus au multi média
- ➢ multi processeurs on chip

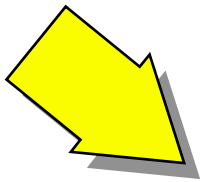
ORGANISATION MEMOIRE

Organisation mémoire

❑ Pourquoi une organisation mémoire ?

➢ Motivations et principe :

- ♦ augmentation de complexité des applications
  - – augmentation du volume de données à mémoriser
  - – augmentation des tailles des mémoires
- ♦ évolution :
  - – en 1980, les ordinateurs n'avaient que quelques kilo octets de mémoires
  - – actuellement, il faut au minimum quelques centaines méga octets
- ♦ les mémoires doivent répondre à 2 contraintes :
  - – taille importante
  - – temps d'accès très court



Contradiction

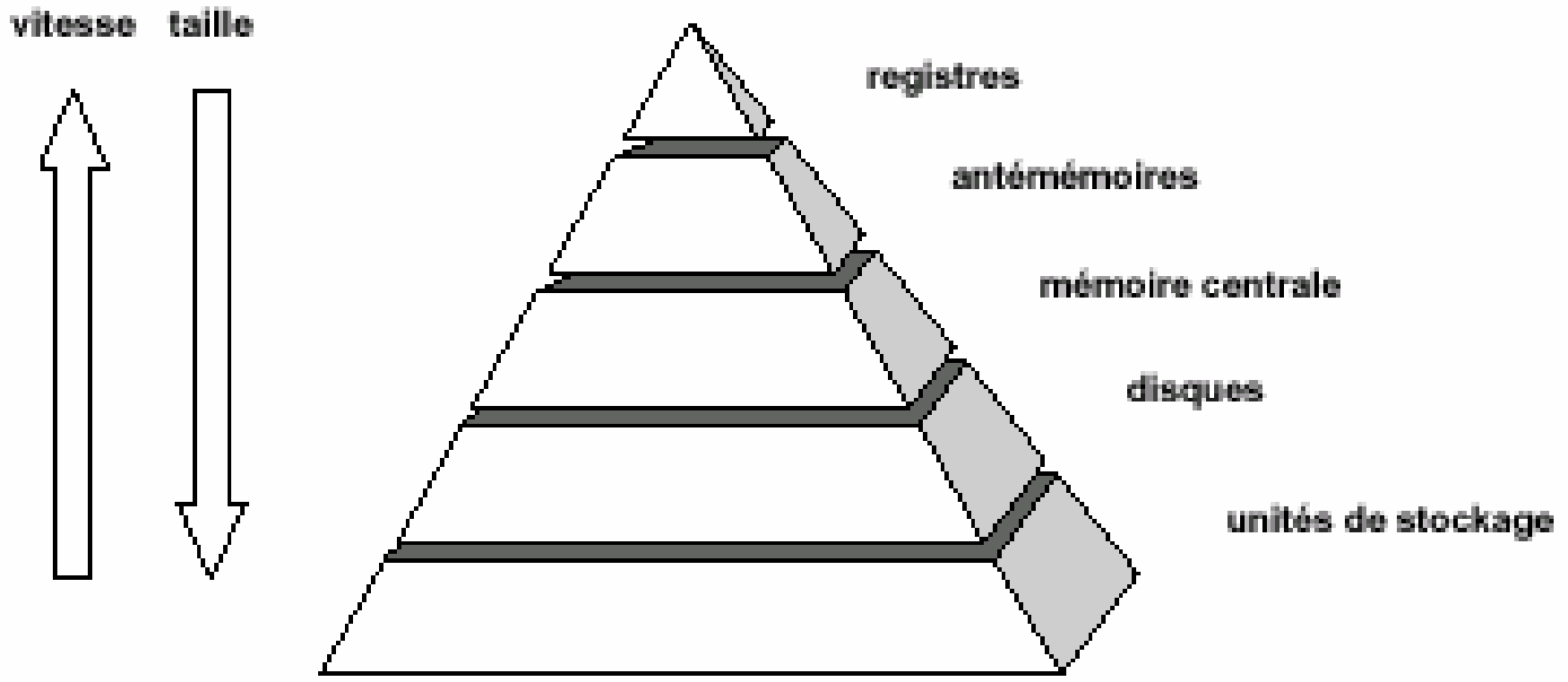
Organisation mémoire

➢Illustration des différents temps d'accès

Technologie	Tps accès	Echelle humaine	Capacités	Indication de prix $ / Mo
Registre	1 à 2 ns	1 s	64 * 64 bits	Fait partie du processeur
Cache intégré	3 ns	3 s	32 ko	Fait partie du processeur
Cache externe	25 ns	25 s	4 Mo	40
Mémoire principale	200 ns	3 min	1 Go	2,5
Disque	12 ms	139 jours	10 Go	0,010
Bande	10 à 20 s	315 ans et plus	50 Go	< 0,05

Organisation mémoire

➢Représentation des niveaux de mémoire



Organisation mémoire

❑ Côté applications

➢Notion de localité dans les accès mémoire :

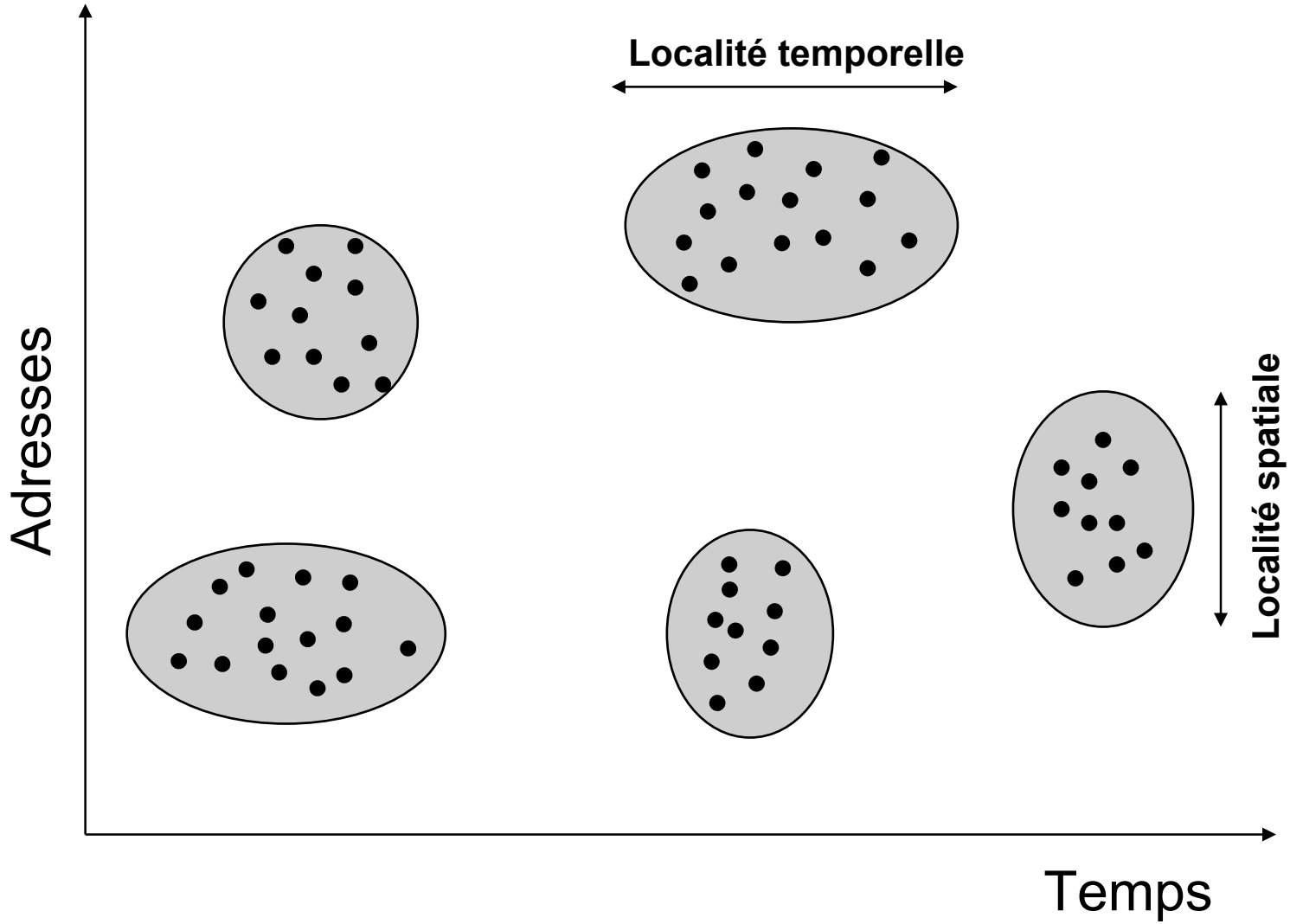
♦ **localité spatiale** :

– *si le processeur exécute l'instruction de l'adresse @i, il y a de forte chance qu'il exécute très prochainement l'instruction de l'adresse @i+1*

♦ **localité temporelle** :

– *si le processeur traite la donnée d à l'instant t, il y a de forte chance qu'il traite à nouveau cette donnée dans une temps très proche*

Organisation mémoire



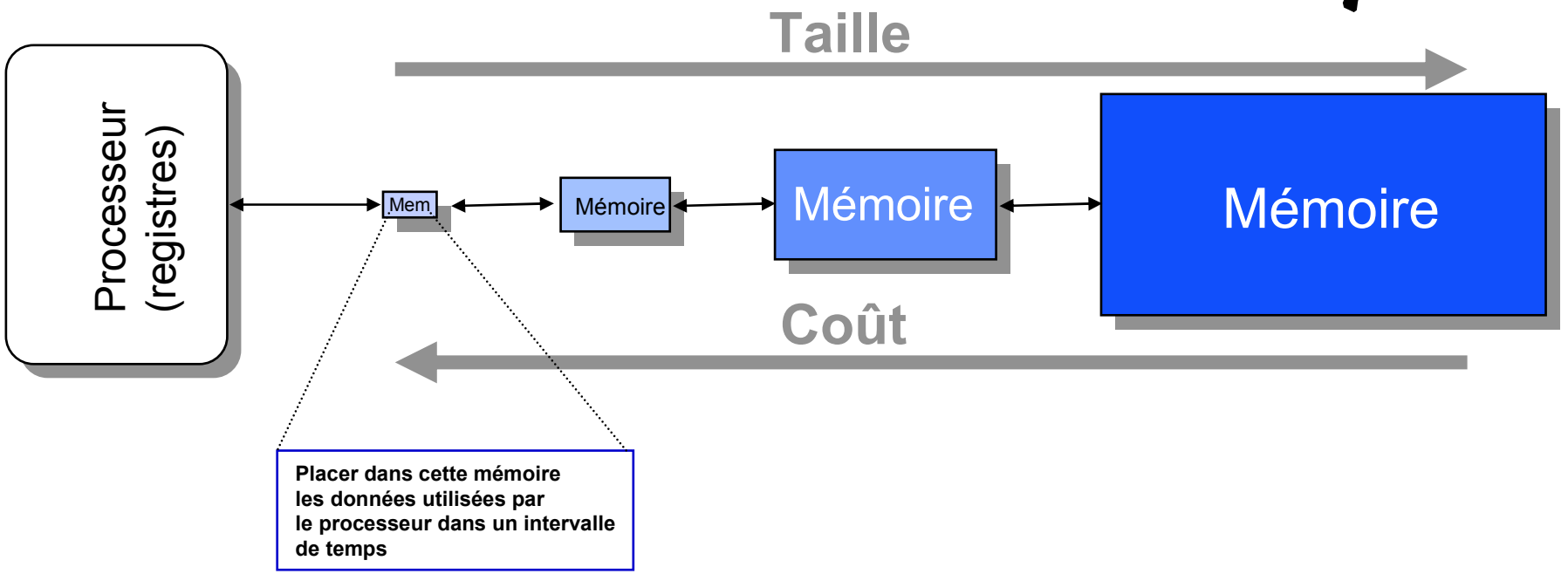
Organisation mémoire

➢ Idées de base :

♦ mise en place d'une hiérarchie mémoire :

– mémoire de petite taille, rapide, proche du processeur

– mémoire de grande taille, lente, éloignée du processeur



Organisation mémoire

❑ Efficacité d'une hiérarchie mémoire :

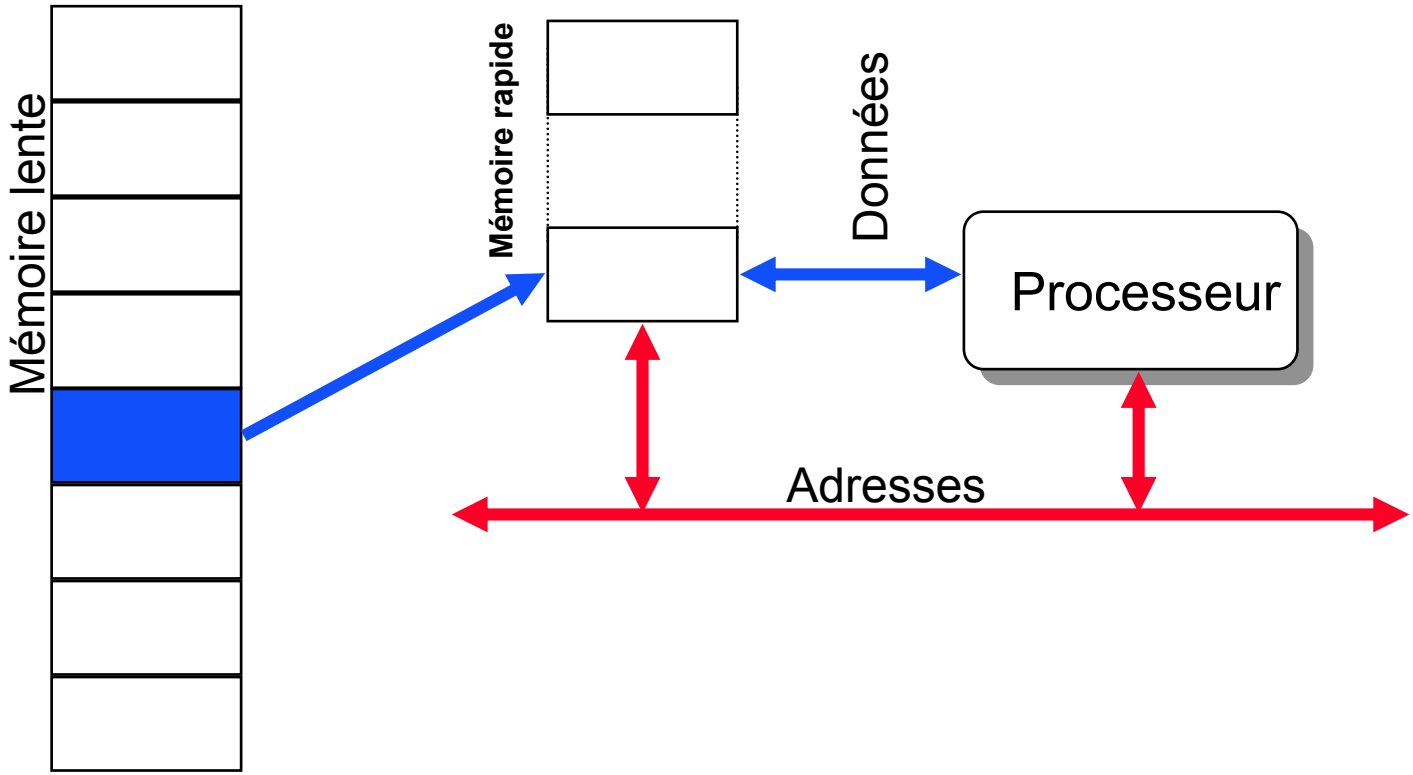
➢ Eff = $\frac{\text{Tps de traitement mémoire rapide}}{\text{Tps de traitement mémoire hiérarchisée}}$

❑ Accélération apportée par la hiérarchisation :

➢ Acc = $\frac{\text{Tps de traitement mémoire lente}}{\text{Tps de traitement mémoire hiérarchisée}}$

Organisation mémoire

➢Fonctionnement



Soit N : la taille du bloc
P : le nombre de données lues et/ou écrites dans un bloc
TpsL : le temps d'accès à la mémoire lente
TpsR : le temps d'accès à la mémoire rapide

Organisation mémoire

➢L'efficacité et l'accélération s'expriment :

$$Eff = \frac{P * TpsR}{2 * N * TpsL + P * TpsR}$$

$$Acc = \frac{P * TpsL}{2 * N * TpsL + P * TpsR}$$

➢Condition à respecter pour que la hiérarchie soit utile

$$2 * N * TpsL + P * TpsR < P * TpsL$$

Organisation mémoire

➢ Intérêt d'une hiérarchie mémoire

$$P > \frac{2 * N * TpsL}{TpsL - TpsR}$$

- ♦ plus la différence entre les temps d'accès est faible, plus les données de la page doivent être réutilisées
- ♦ si P est faible, alors la hiérarchie est peu intéressante

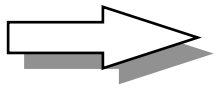
➢ Exemple :

- ♦ Soit N = 256 et TpsL = 2 * TpsR

$$P > \frac{2 * 256 * 2 * TpsR}{2 * TpsR - TpsR}$$

$$P > 2 * 256 * 2$$

$$P > 4 * 256$$



En moyenne, chaque donnée doit être utilisée 4 fois

Organisation mémoire

➢si P $\longrightarrow \infty$ alors :

$$Eff = \frac{P * TpsR}{2 * N * TpsL + P * TpsR}$$

$Eff \longrightarrow 1$

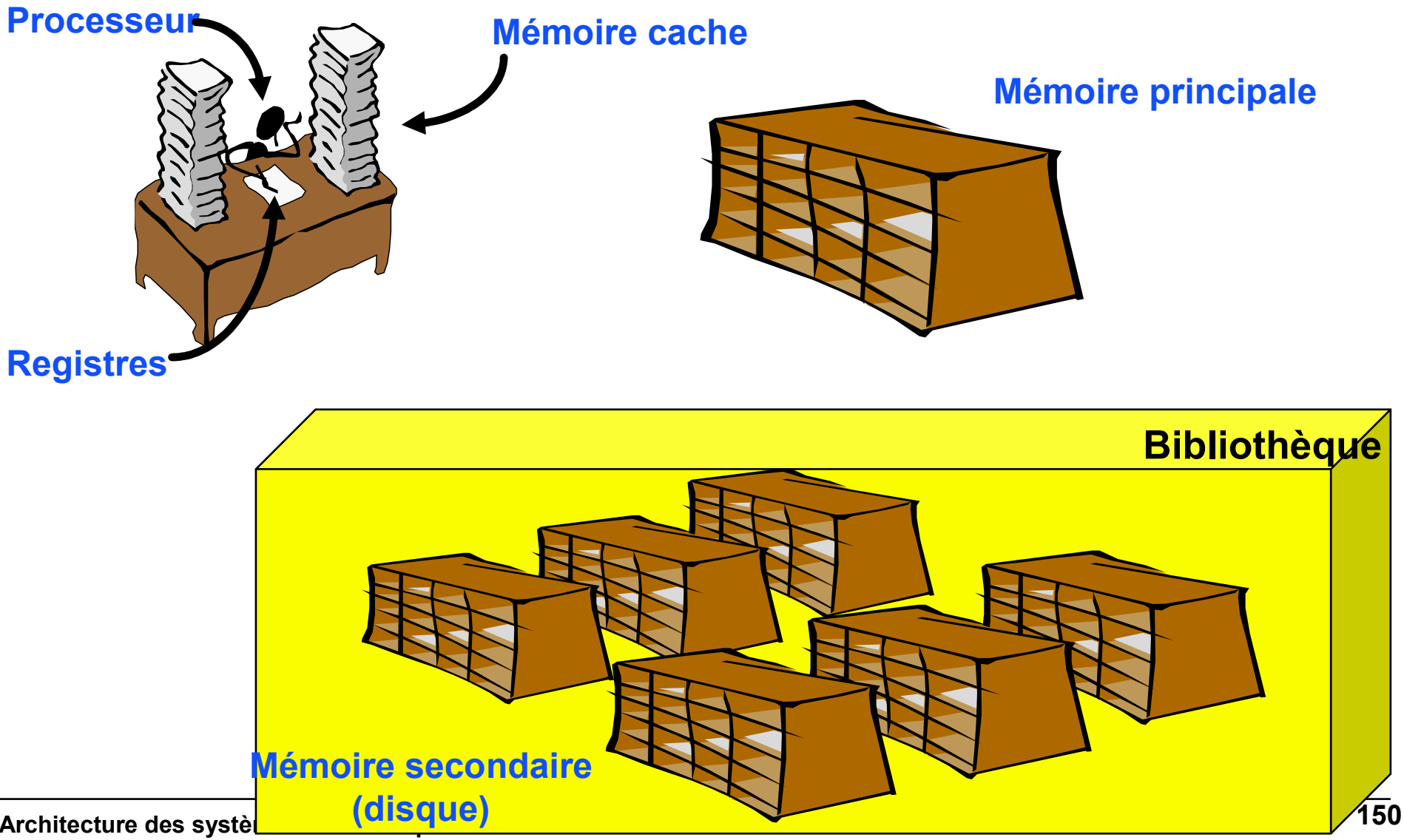
$$Acc = \frac{P * TpsL}{2 * N * TpsL + P * TpsR}$$

$Acc \longrightarrow \frac{TpsL}{TpsR}$

Equivalent
à un système
sans hiérarchie
avec mémoire rapide

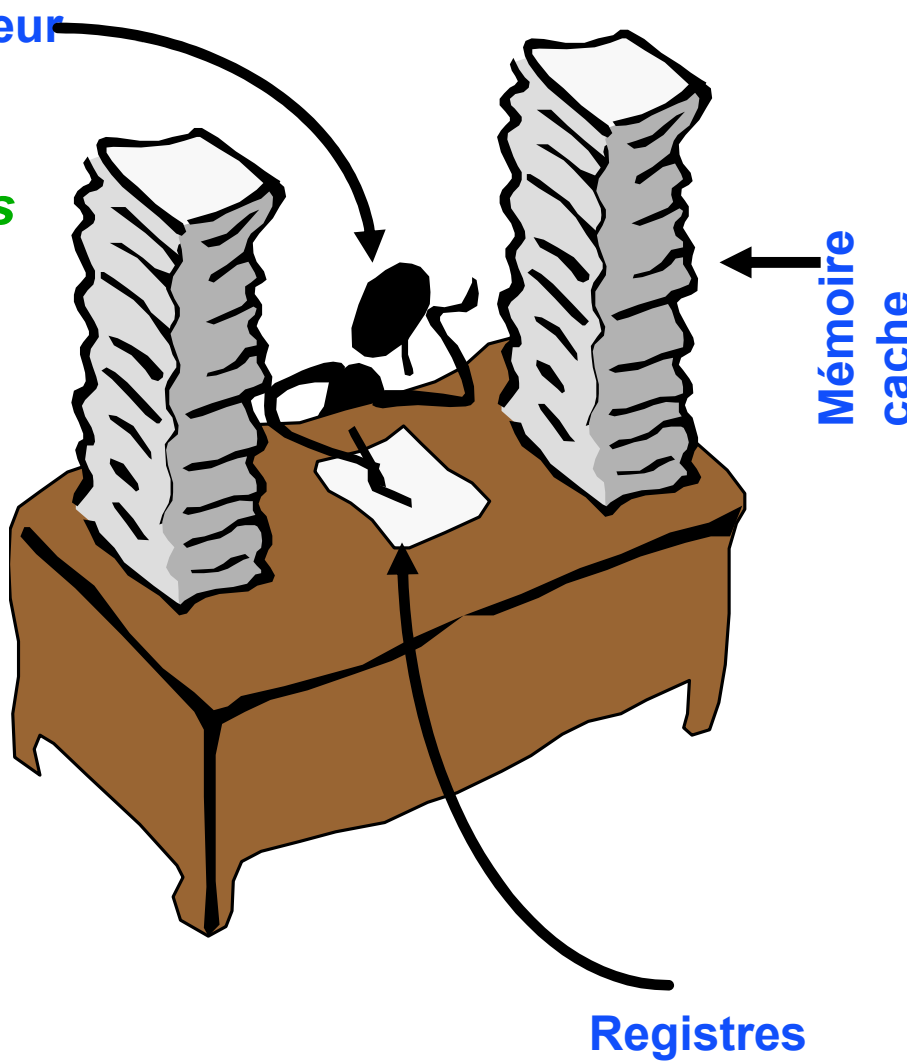
Organisation mémoire

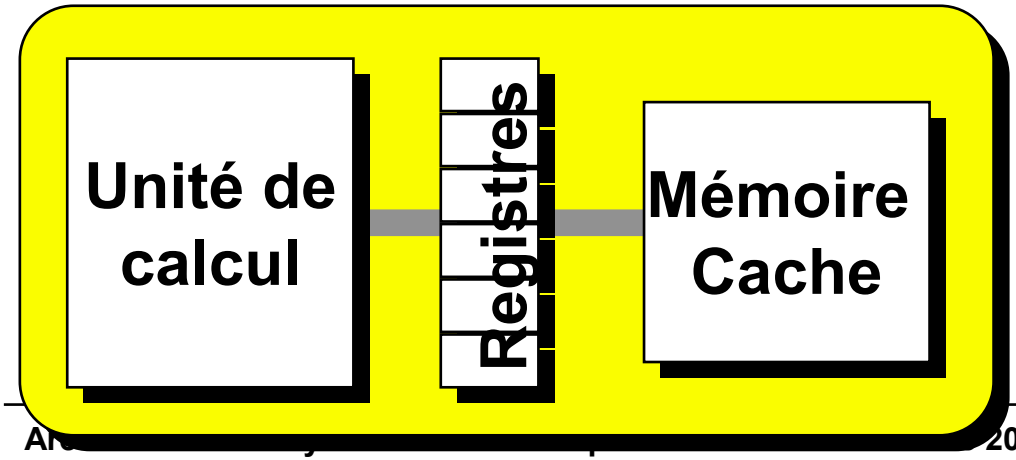
La mémoire cache, comment ça marche ?



Organisation mémoire

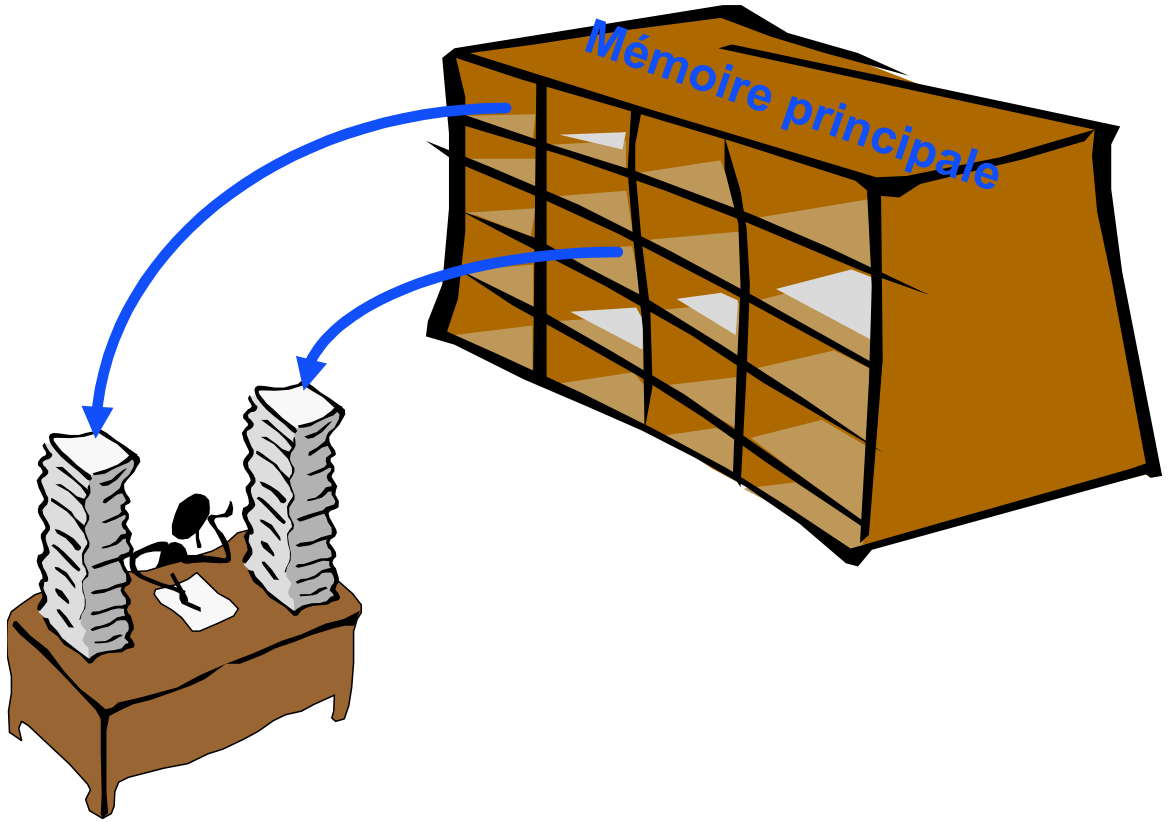
- le *travailleur* :
  - travaille sur les ***feuilles*** placées directement devant lui
  - peut accéder rapidement aux ***dossiers*** qui sont sur son bureau
  - la taille du bureau étant limitée, il y dépose uniquement les dossiers en cours de traitement
  - prend les feuilles ***une à une*** dans les dossiers





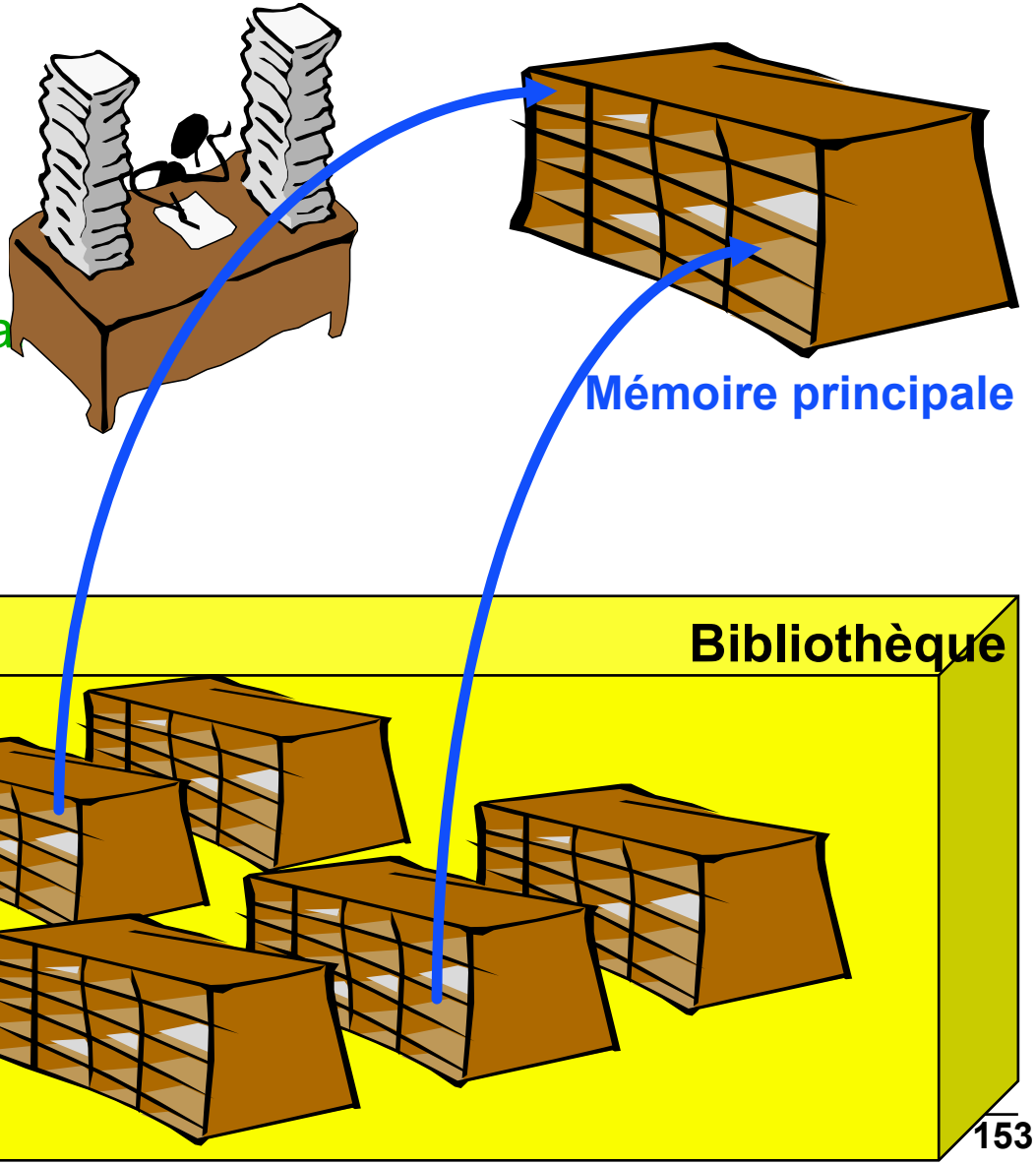
Organisation mémoire

- **L'armoire :**
  - contient les ***dossiers***
  - l'accès est ***plus lent***
  - l'accès se fait ***par dossier*** et non par feuille



Organisation mémoire

- **La bibliothèque :**
  - contient les ***dossiers***
  - l'accès est ***très lent***
  - l'accès se fait ***par N dossiers*** et non dossier par dossier



Organisation mémoire

❑ Fonctionnement d'une mémoire cache

➢mémoire très rapide :

- ♦ temps d'accès proche du temps de cycle du processeur

➢défaut de cache :

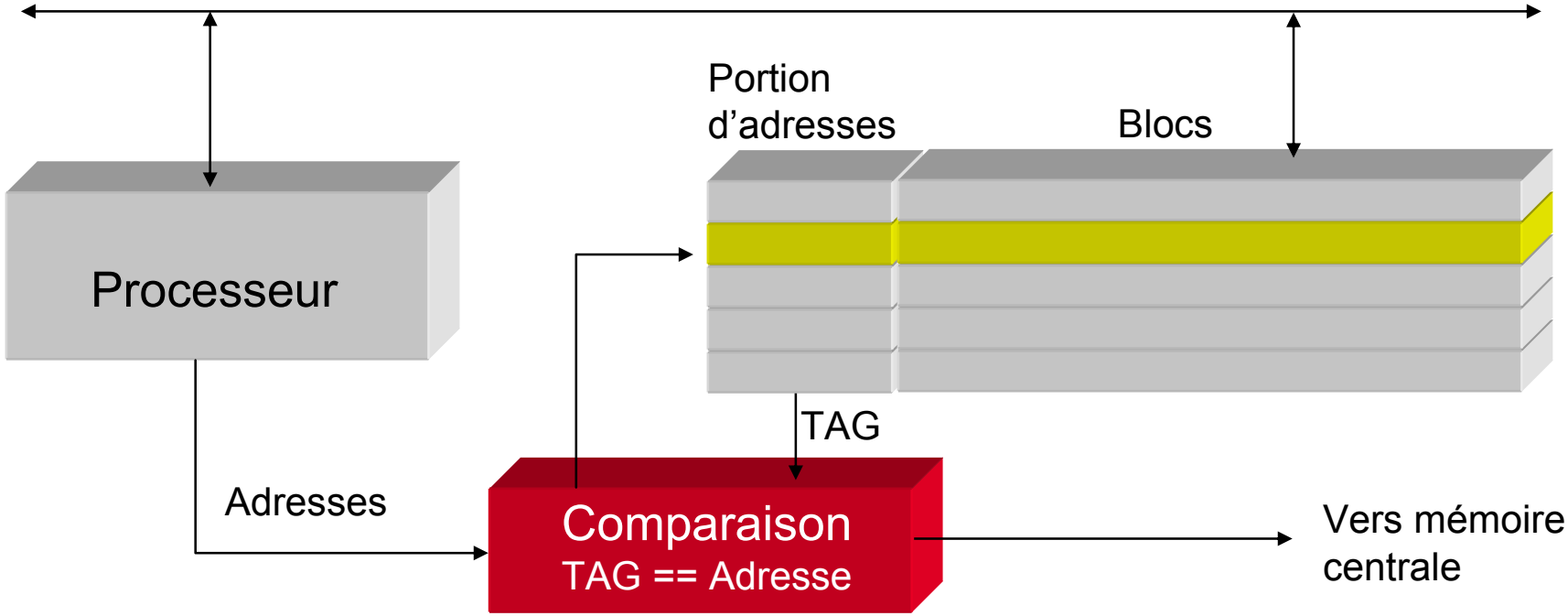
- ♦ *on dit qu'il y a défaut de cache lorsque la donnée réclamée par le processeur ne se trouve pas dans le cache*

➢recopie de petits blocs de la mémoire centrale

Organisation mémoire

➢Nécessité de réaliser une association entre :

- ♦ donnée (ou bloc de données)
- ♦ adresse (portion d'adresse du bloc)

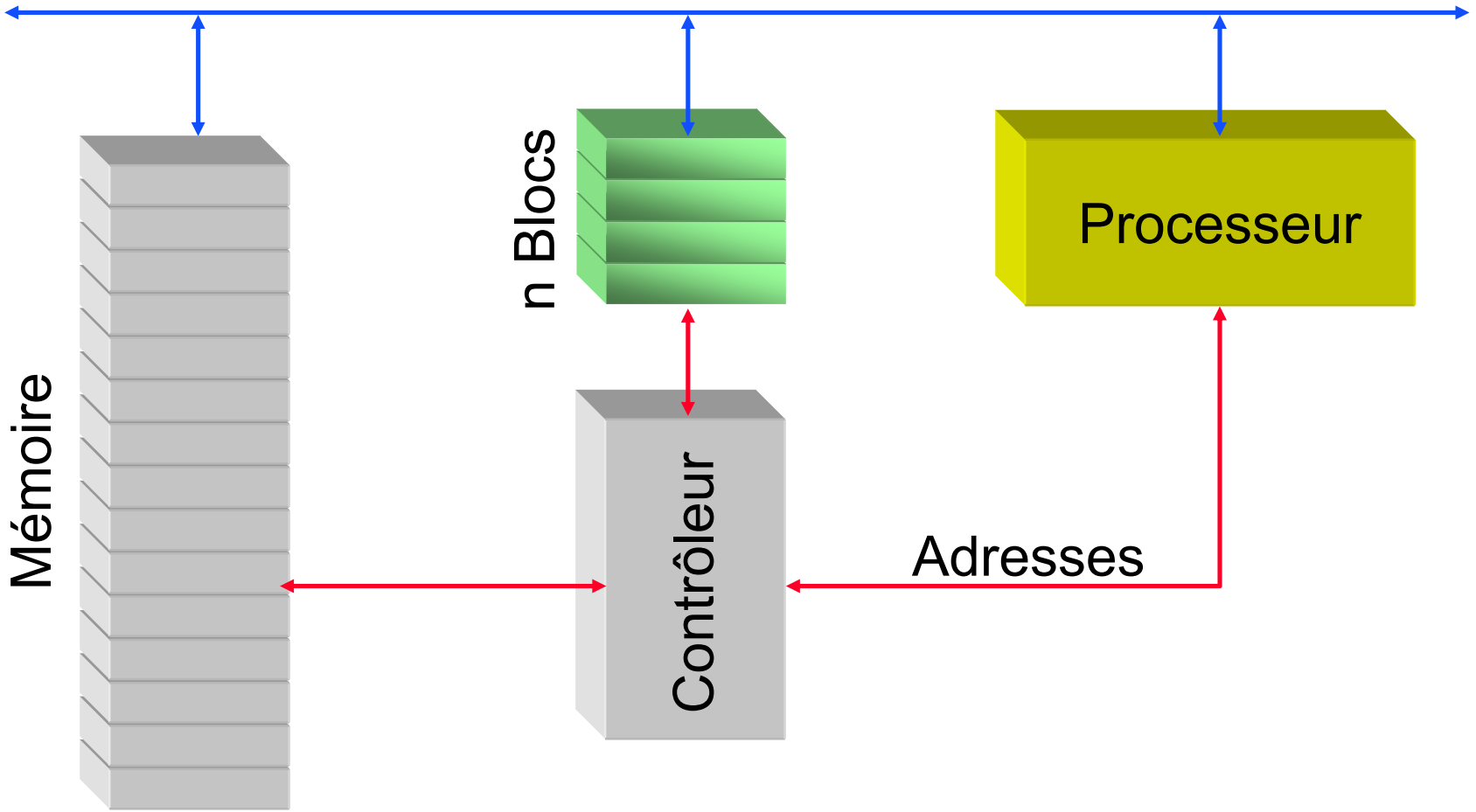


Organisation mémoire

➢On divise : la mémoire cache en n blocs

la mémoire principale en N blocs

➢ Avec N >> n

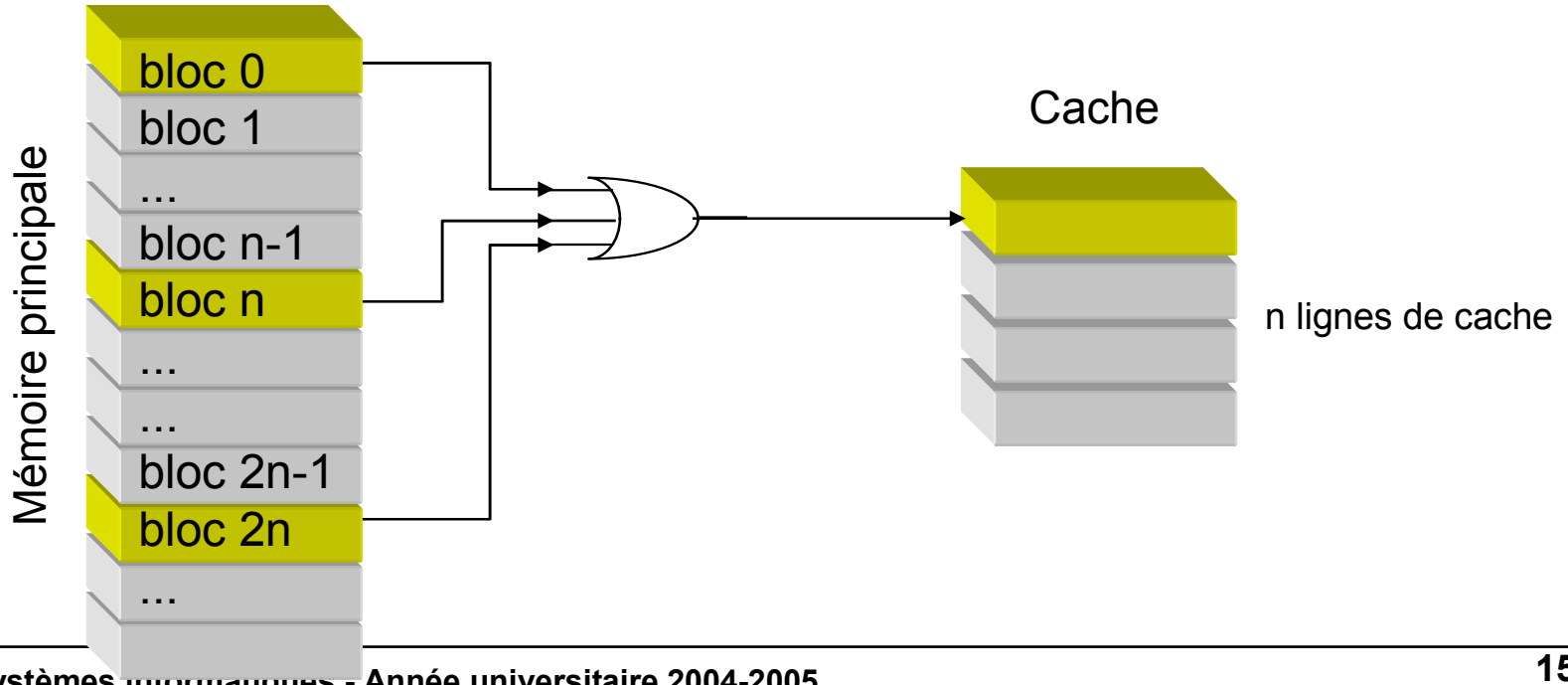


Organisation mémoire

➢5.3.1) Rangement des blocs dans le cache

- ♦ mémoire cache en correspondance direct :
  - – un bloc a une place fixe dans le cache quelque soit l'instant où il est appelé
  - – simplifie la logique à mettre en oeuvre pour la gestion du cache
  - – n'est pas optimale dans la façon de remplacer les blocs
  - – un bloc i de la mémoire principale est stocké dans le cache à l'adresse

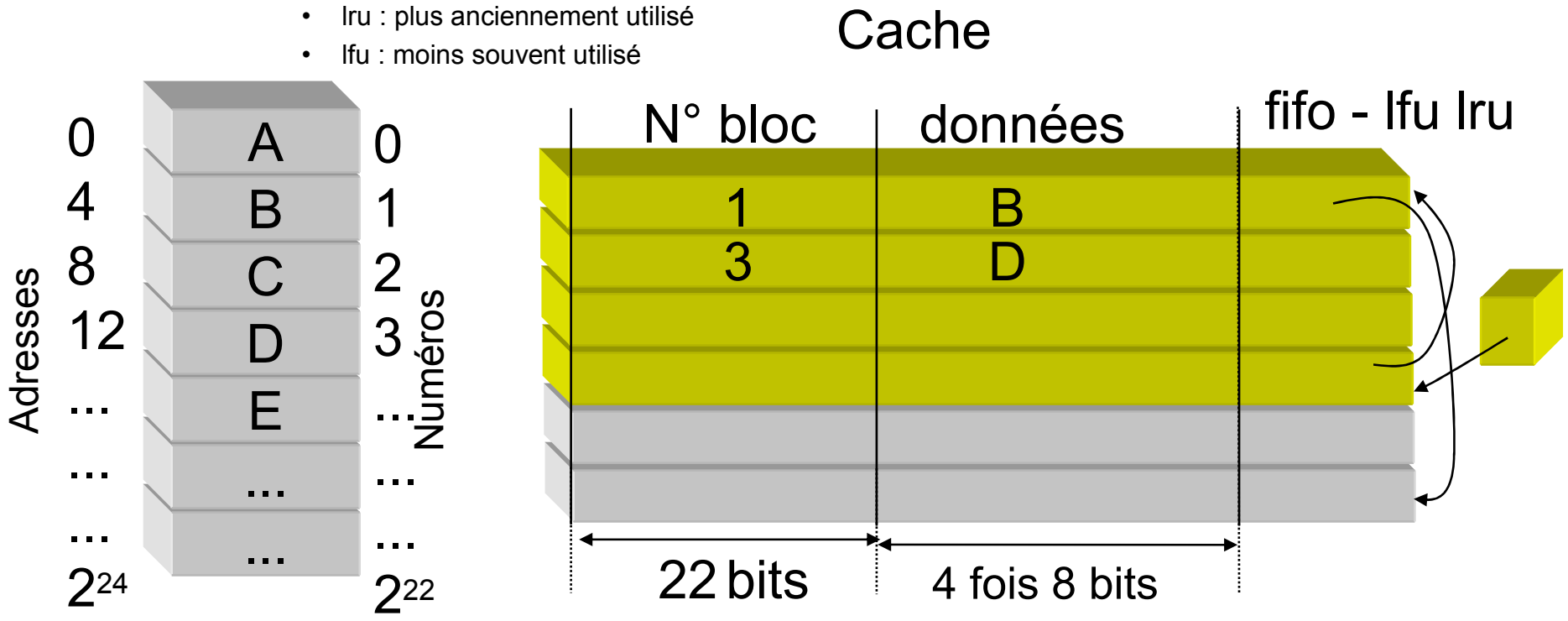
NuméroBlocCache = NuméroBlocMémoire modulo n



Organisation mémoire

♦ mémoire cache associative :

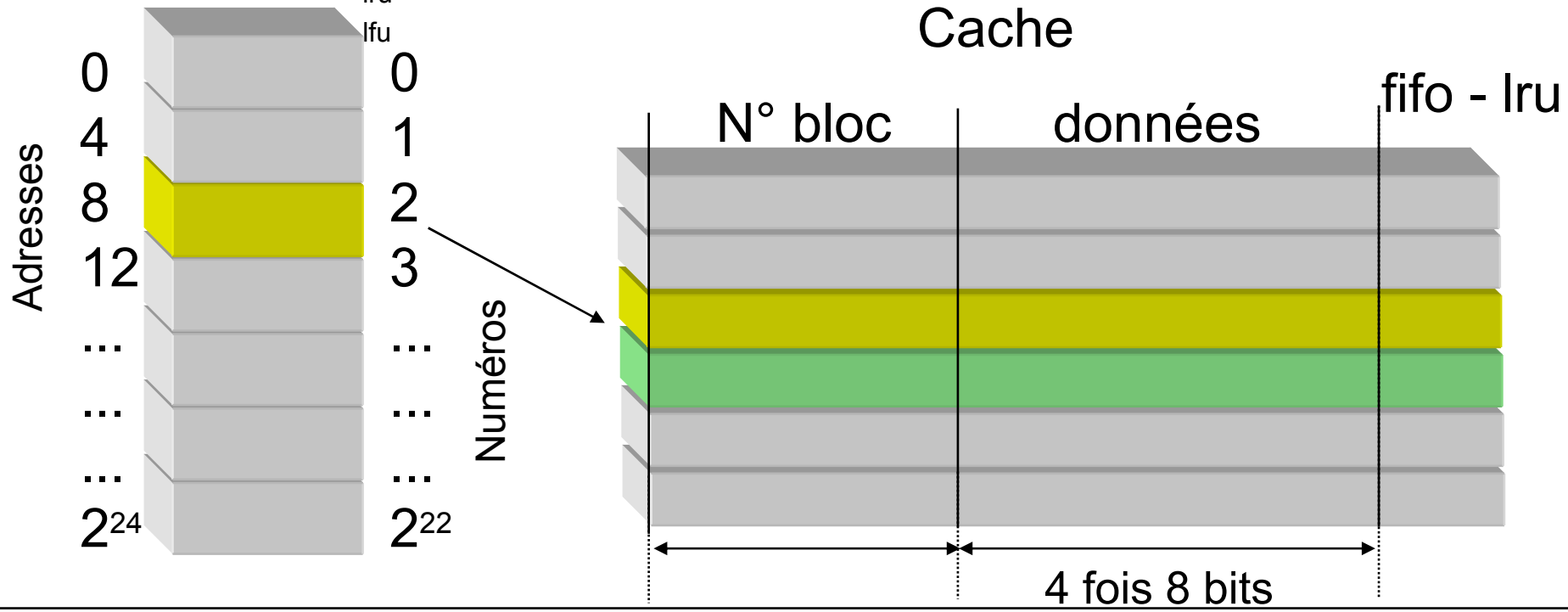
- un bloc à une place différentes dans le cache et ceci en fonction du remplissage du cache avant l'appel du bloc
- logique de gestion plus lourde
- permet de mettre en oeuvre une politique de remplacement des blocs plus judicieuse (par exemple le remplacement du bloc le plus anciennement utilisé) :
  - random : tirage aléatoire
  - fifo : premier entrée, premier sortie
  - lru : plus anciennement utilisé
  - lfu : moins souvent utilisé



Organisation mémoire

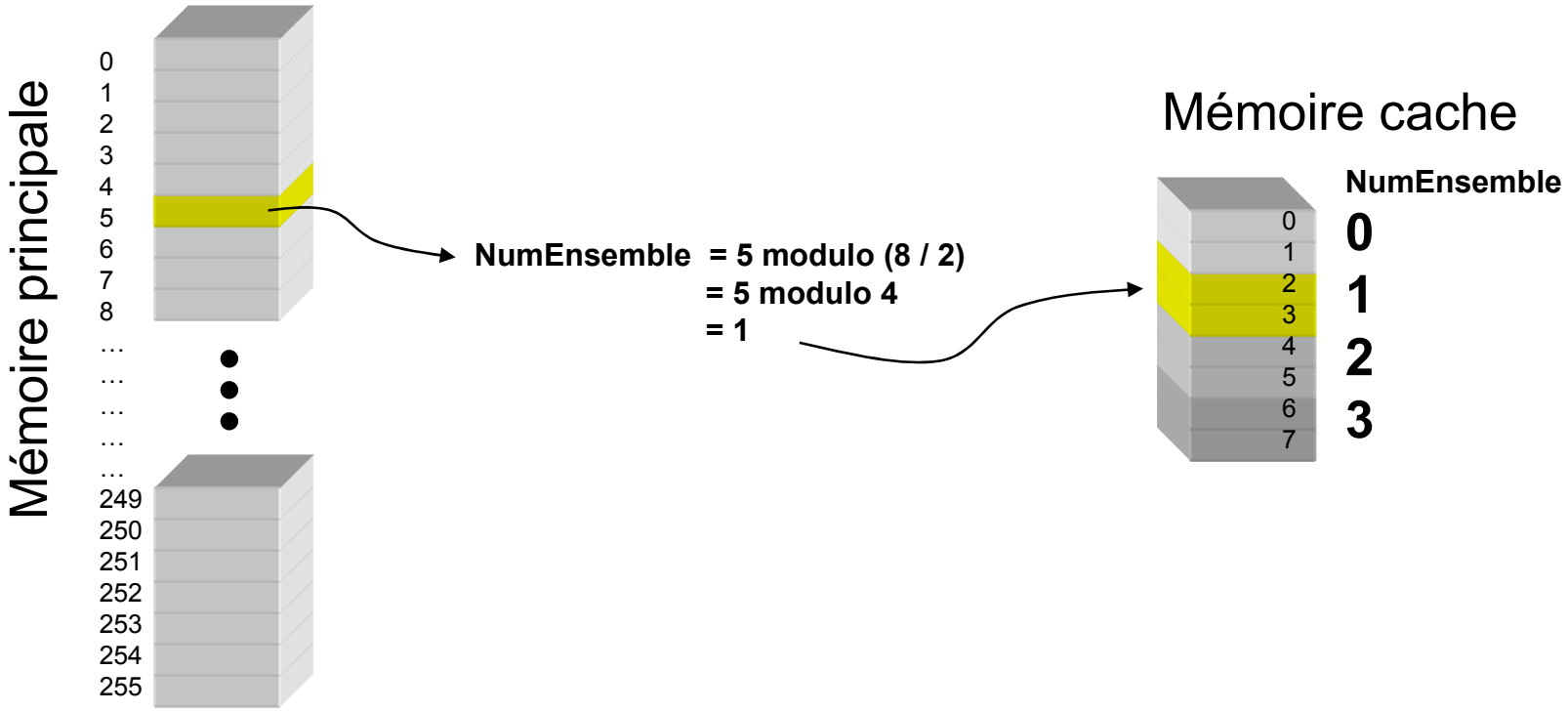
♦ mémoire cache associative par ensemble de N voies :

- un bloc peut prendre place dans N blocs de mémoire cache différents
- simplifie un peu la logique de contrôle
- est généralement suffisant pour assurer de bonnes performances au système mémoire
- politique de remplacement dans les ensembles :
  - random
  - fifo
  - lru
  - lfu



Organisation mémoire

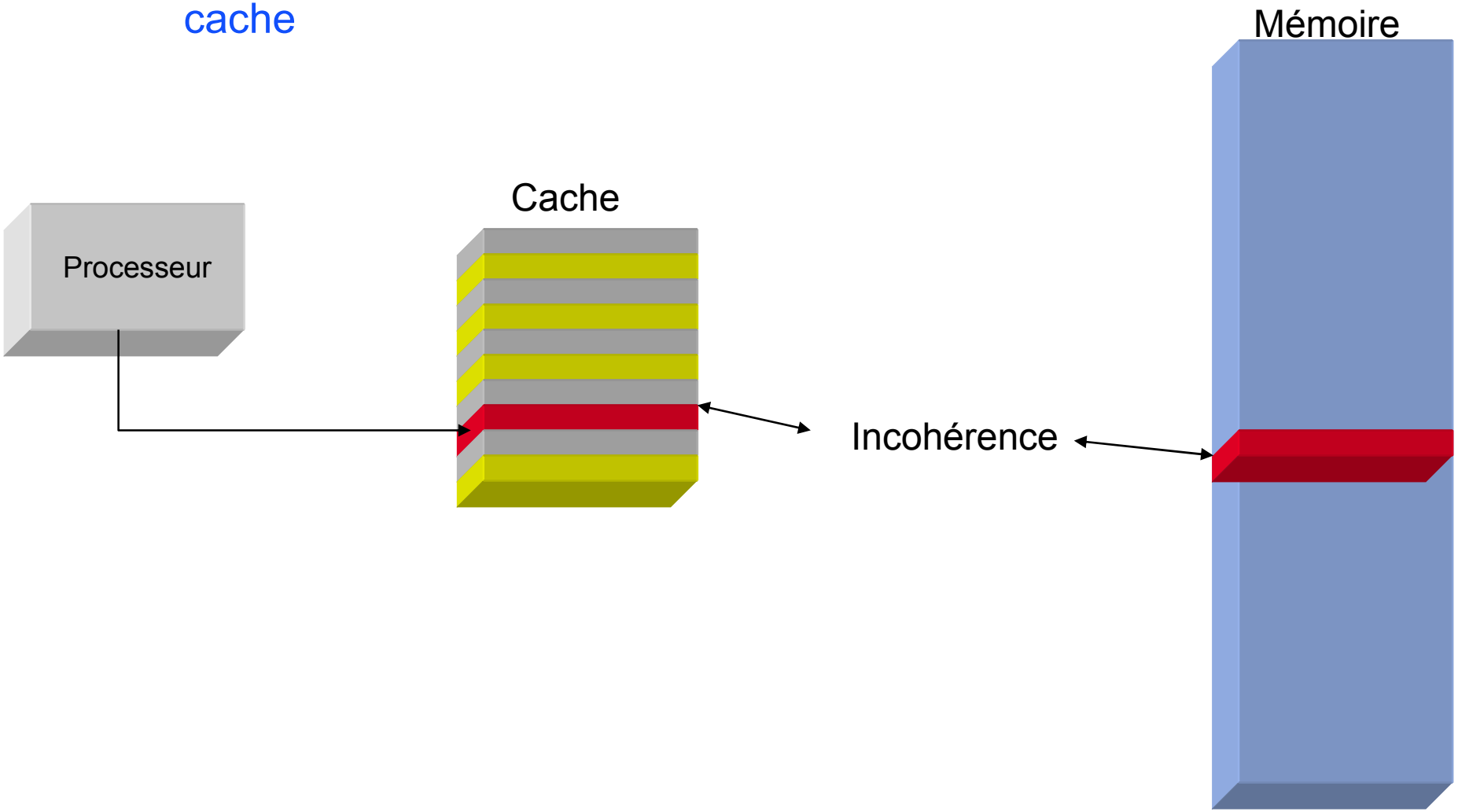
- recherche de l'emplacement dans le cache :
  - si N est le numéro de bloc de la mémoire centrale
  - si L est le nombre de lignes du cache
  - si V est le nombre de voies du cache
  - alors
  - NumEnsemble = N mod (L/V)
  - les numéros de blocs cache concernés par le rapatriement sont alors :
  - NumEnsemble * V <= NumBlocsCache <= (NumEnsemble + 1) * V -1



Organisation mémoire

➢ Politique d'écriture dans la mémoire principale

♦ lorsque le processeur modifie une valeur contenu dans un bloc de cache

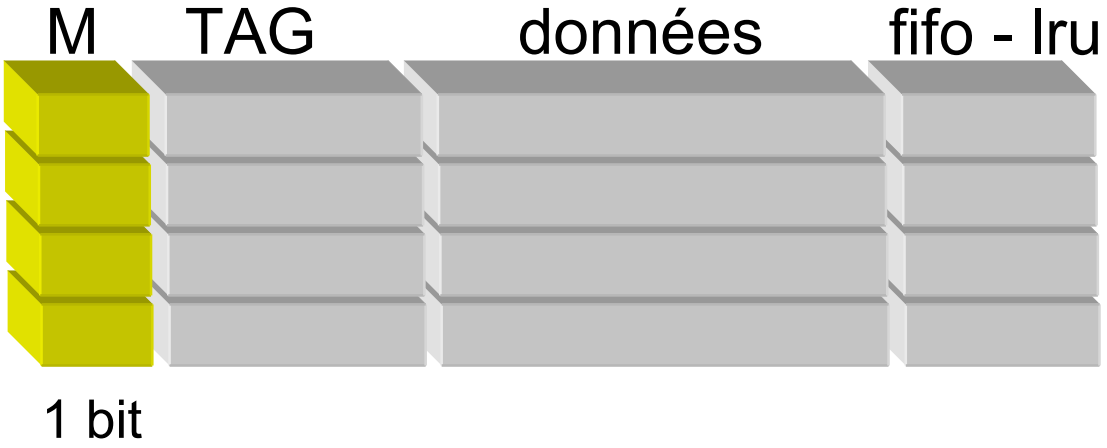


Organisation mémoire

♦ 2 politiques de mise à jour de la mémoire principale :

– write back :

- la mise à jour de la mémoire principale s'effectue lors des remplacements de blocs
- la mémoire et le cache peuvent contenir des blocs incohérents
- un défaut de cache avec rapatriement de bloc sur un bloc modifié coûte cher
- le cache dispose d'un bit indiquant si un bloc a été modifié : bit M



– write through :

- la mise à jour de la mémoire s'effectue à chaque écriture dans le cache
- la mémoire et le cache sont toujours cohérents
- on surcharge le bus d'accès à la mémoire
- pas besoin de bit indiquant l'incohérence entre cache et mémoire centrale

Organisation mémoire

♦ 2 politiques pour l'écriture d'une donnée qui n'est pas dans le cache :

– Miss fetch on write (write allocate) : le bloc est ramené dans le cache, et l'écriture est réalisée dans le cache
  - la pénalité du défaut de cache est identique à un défaut en lecture
  - il faut ramener le bloc complet de la mémoire, puis faire l'écriture

– Miss write around : la donnée est écrite directement dans la mémoire principale
  - méthode plus rapide pour des écritures uniques
  - mais il y aura un autre défaut lorsque le bloc sera réutilisé (en lecture)

Organisation mémoire

Taille optimale d'un cache ?

- plus le bloc est petit, plus l'adresse a lui associer est grande
- exemple : mémoire cache 256 mots de 16 bits, mémoire principale 2^{24} mots de 16 bits (24 lignes d'adresses)

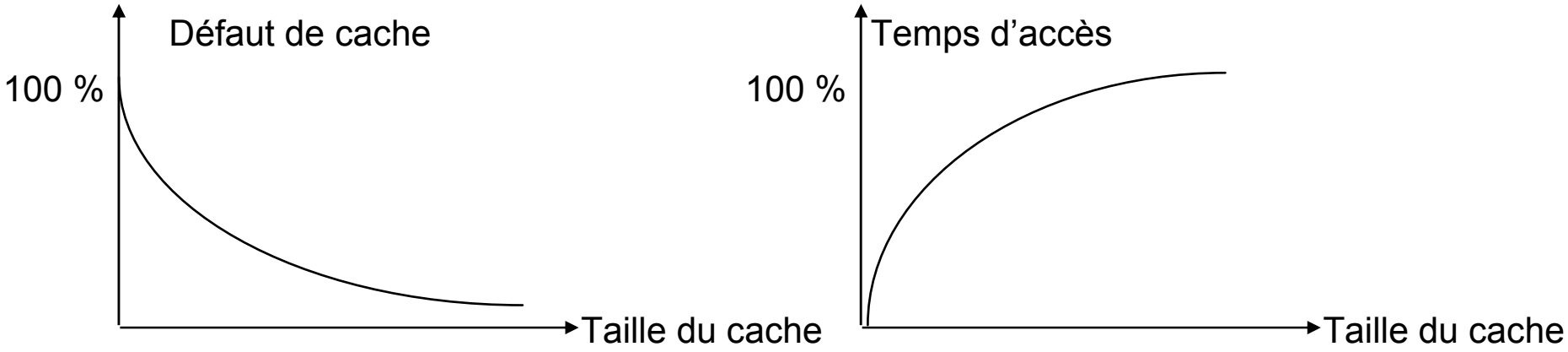
Taille du bloc du cache	Nbr de TAG	Taille du TAG	Mémoire TAG	Mémoire
256	1	16	16	256 * 16 4 Kbits
16	16	20	320	
1	256	24	6144	

- Nbr Tag = Taille Cache / Taille Bloc
- Taille Tag = Nbr lignes Adresses principale - $\log_2$ (Taille Cache)
- Mémoire Tag = Nbr Tag * Taille Tag

Organisation mémoire

➢Comment dimensionner un cache ?

- ♦ il faut analyser l'application
- ♦ relever le nombre de fautes (ou défaut de cache) en fonction de la taille du cache :
- ♦ si h est la probabilité d'accès à une donnée en cache



- ♦ Il faut onc rechercher un compromis entre la taille et le temps d'accès
- ♦ On recherche à minimiser l'expression :

$$Tacces = h \cdot TpsCache + (1 - h)\ TpsMemoire$$

Organisation mémoire

- **Exemple : soit une mémoire ayant un temps d'accès = 100 et une taille égale à 1024**

Tps Cache	--	10	20	30	40	50	60	70
Taille	0	16	32	64	128	256	512	1024
Défaut	100%	60 %	50 %	40 %	30 %	20 %	10 %	0 %
Taccès	100	64	60	58	58	60	64	70

- **Choix d'un cache :**
  - ➢ analyse de l'application, détermination des défauts de cache :
    - ♦ pour les systèmes dédiés, cette analyse est facilité par le déterminisme de l'application
    - ♦ pour un système général, cette analyse s'effectue sur des benchmarks (traitement de texte, compilateur, etc)
  - ➢ détermination :
    - ♦ taille des blocs
    - ♦ taille du cache

Organisation mémoire

❑ Simulation de mémoires caches :

➢Exemple du produit de 2 matrices 32 * 32

- ♦ mémoire cache unifié
- ♦ total des accès mémoire : 269 476
- ♦ mémoire cache de 1024 octets
- ♦ simulations pour des caches :
  - – ayant des blocs de taille :
    - 4
    - 8
    - 16
    - 32
    - 64
    - 128
  - – ayant une politique de remplacement de blocs de type :
    - correspondance directe
    - LRU
    - FIFO
    - Random
  - – ayant des associativités variant de 1 à NbBlocs (toujours en puissance de 2)

Organisation mémoire

♦ Le programme est le suivant :

```
1600        MOVE R1, @a00           -- Pointeur pour la matrice A (Adresse 0)
            MOVE R2, @b00           -- Pointeur pour la matrice B (Adresse N^2)
            MOVE R3, @c00           -- Pointeur pour la matrice C (Adresse 2N^2)
            MOVE R4, N              -- Nombre de lignes
boucle1
            MOVE R5, N              -- Nombre de colonnes
boucle2
            MOVE R6, N              -- Nombre de calcul aik * bkj + cij pour le calcul de cij
            MOVE R7, 0              -- Variable temporaire pour cij
boucle3
            MOVE R8, (R1)++         -- Charge element de la matrice A, prepare acces element suivant
            MOVE R9, (R2)++         -- Charge element de la matrice B, prepare acces element suivant
            MULT R8, R9             -- Effectue le produit aik * bkj
            ADD  R7, R8             -- Effectue l'addition cij + (aik * bkj)
            SUB  R6, 1              -- Calcul de cij termine ?
            BRnz boucle3            -- Non, alors on reboucle et on continue
            MOVE (R3)++, R7         -- Oui, alors on stocke le resultat
            SUB  R1, N              -- On ramene le pointeur de la matrice A en debut de ligne
            SUB  R5, 1              -- Reste-t-il des elements de la ligne i a calculer ?
            BRnz boucle2            -- Oui, alors on reboucle et on continue
            MOVE R2, @b00           -- Non, alors on passe a la ligne suivante
            ADD  R1, N              -- On place le pointeur de la matrice A sur la ligne suivante
            SUB  R4, 1              -- Reste-t-il des lignes a calculer ?
            BRnz boucle1            -- Oui, alors on boucle et on poursuit le caclul
```

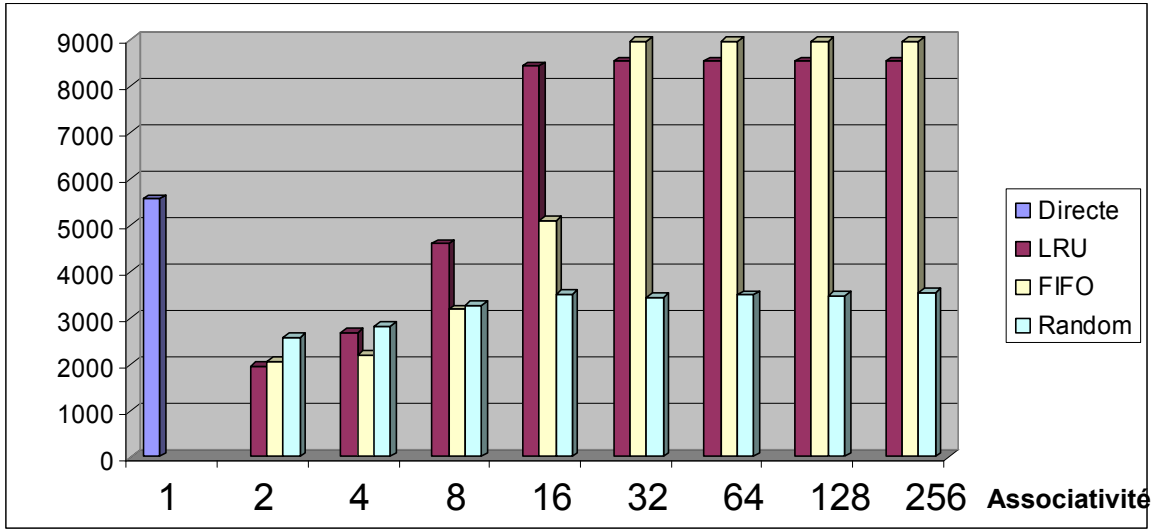
Organisation mémoire

- La liste des accès est la suivante :

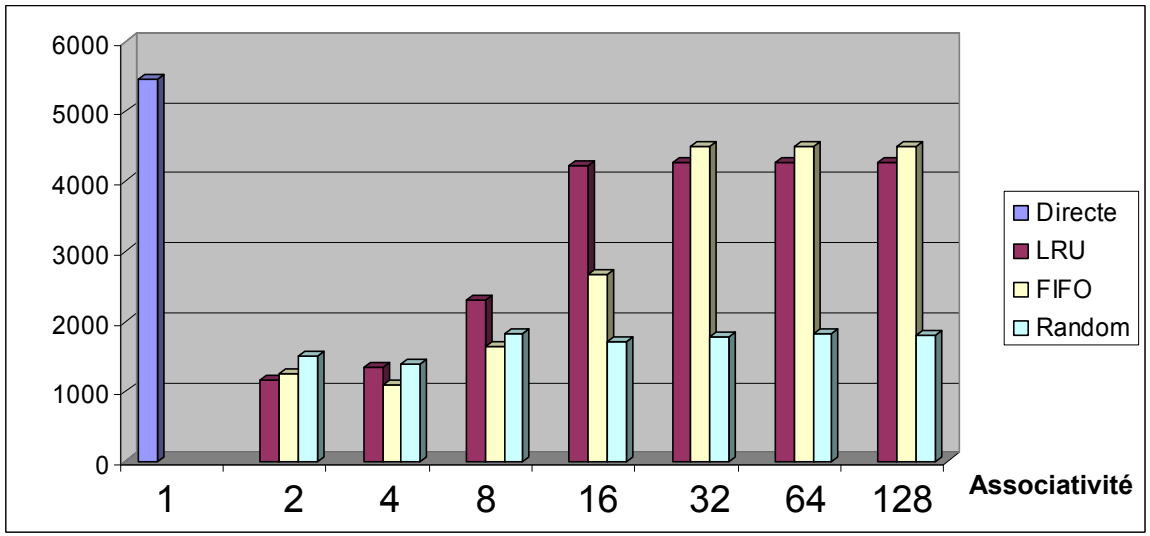
```
2 640 -- Acces a l'instruction MOVE R1,@a00
2 641 -- Acces a l'instruction MOVE R2,@b00
2 642 -- Acces a l'instruction MOVE R3,@c00
2 643 -- Acces a l'instruction MOVE R4,N
2 644 -- Acces a l'instruction MOVE R5,N
2 645 -- Acces a l'instruction MOVE R6,N
2 646 -- Acces a l'instruction MOVE R7,0
2 647 -- Acces a l'instruction MOVE R8,(R1)++
0 0 --- Acces A[0][0]
2 648 -- Acces a l'instruction MOVE R9,(R2)++
0 40 --- Acces B[0][0]
2 649 -- Acces a l'instruction MULT R8,R9
2 64a -- Acces a l'instruction ADD R7,R8
2 64b -- Acces a l'instruction SUB R6,1
2 64c -- Acces a l'instruction BRnz boucle3
2 647 -- Acces a l'instruction MOVE R8,(R1)++
0 1 --- Acces A[0][1]
2 648 -- Acces a l'instruction MOVE R9,(R2)++
0 41 --- Acces B[1][0]
2 649 -- Acces a l'instruction MULT R8,R9
2 64a -- Acces a l'instruction ADD R7,R8
…
…
2 649 -- Acces a l'instruction MULT R8,R9
2 648 -- Acces a l'instruction MOVE R9,(R2)++
0 47 --- Acces B[7][0]
2 649 -- Acces a l'instruction MULT R8,R9
2 64a -- Acces a l'instruction ADD R7,R8
2 64b -- Acces a l'instruction SUB R6,1
2 64c -- Acces a l'instruction BRnz boucle3
2 64d -- Acces a l'instruction MOVE (R3)++,R7
1 80 --- Acces C[0][0]
2 64e -- Acces a l'instruction SUB R1,N
…
…
2 646 -- Acces a l'instruction MOVE R7,0
2 647 -- Acces a l'instruction MOVE R8,(R1)++
0 0 --- Acces A[0][0]
2 648 -- Acces a l'instruction MOVE R9,(R2)++
0 48 --- Acces B[0][1]
2 649 -- Acces a l'instruction MULT R8,R9
```

Organisation mémoire

- ➢ Taille des blocs = 4

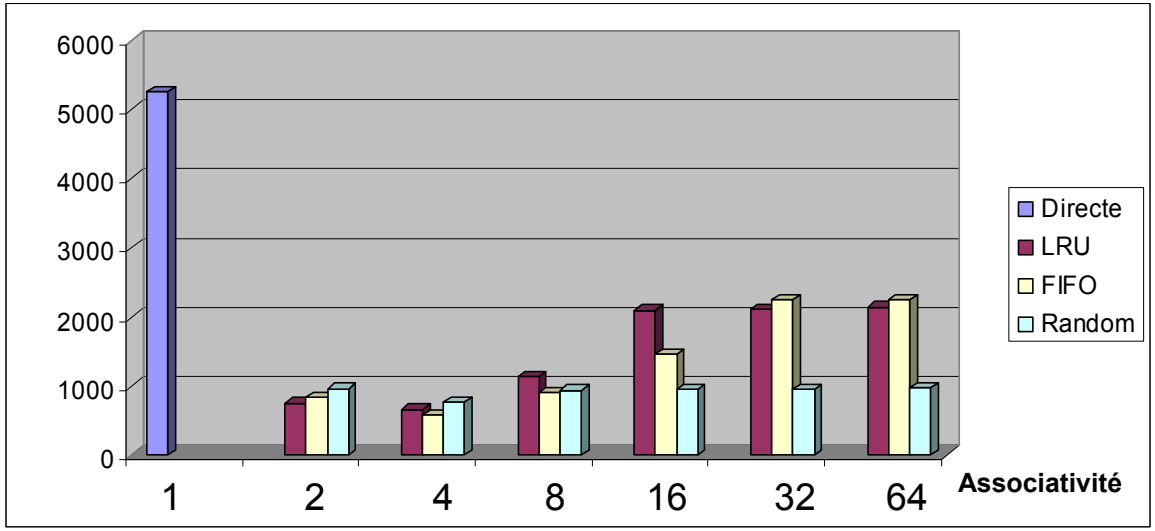


- ➢ Taille des blocs = 8

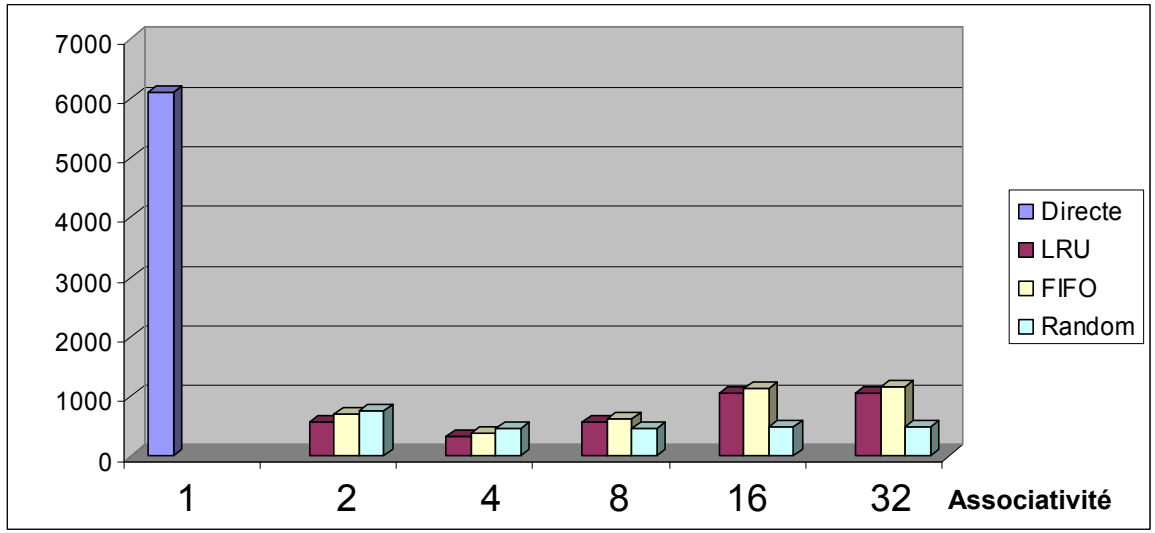


Organisation mémoire

➢ Taille des blocs = 16

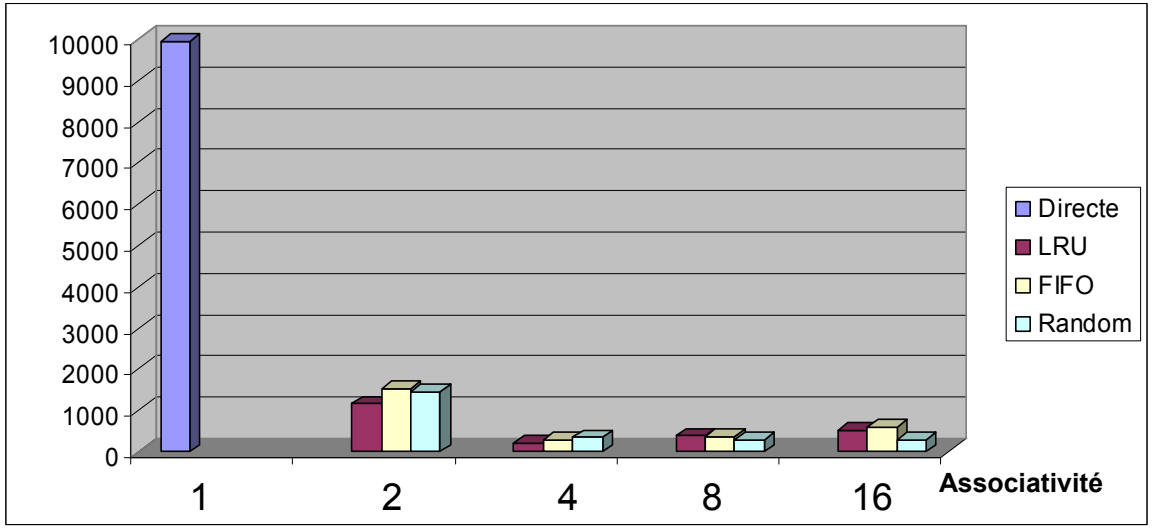


➢ Taille des blocs = 32

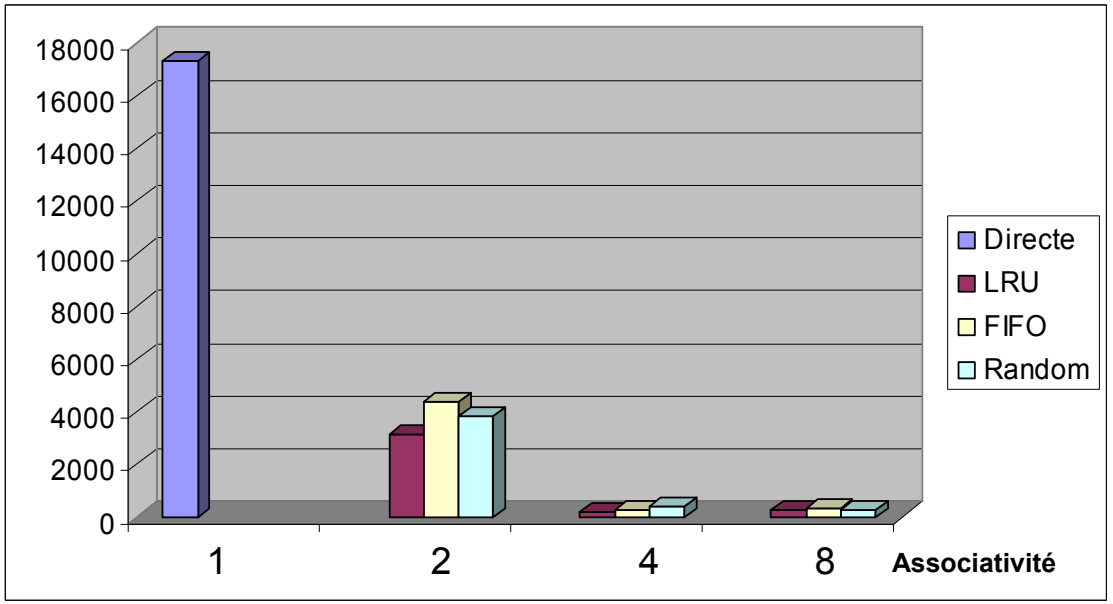


Organisation mémoire

➢ Taille des blocs = 64



➢ Taille des blocs = 128



Organisation mémoire

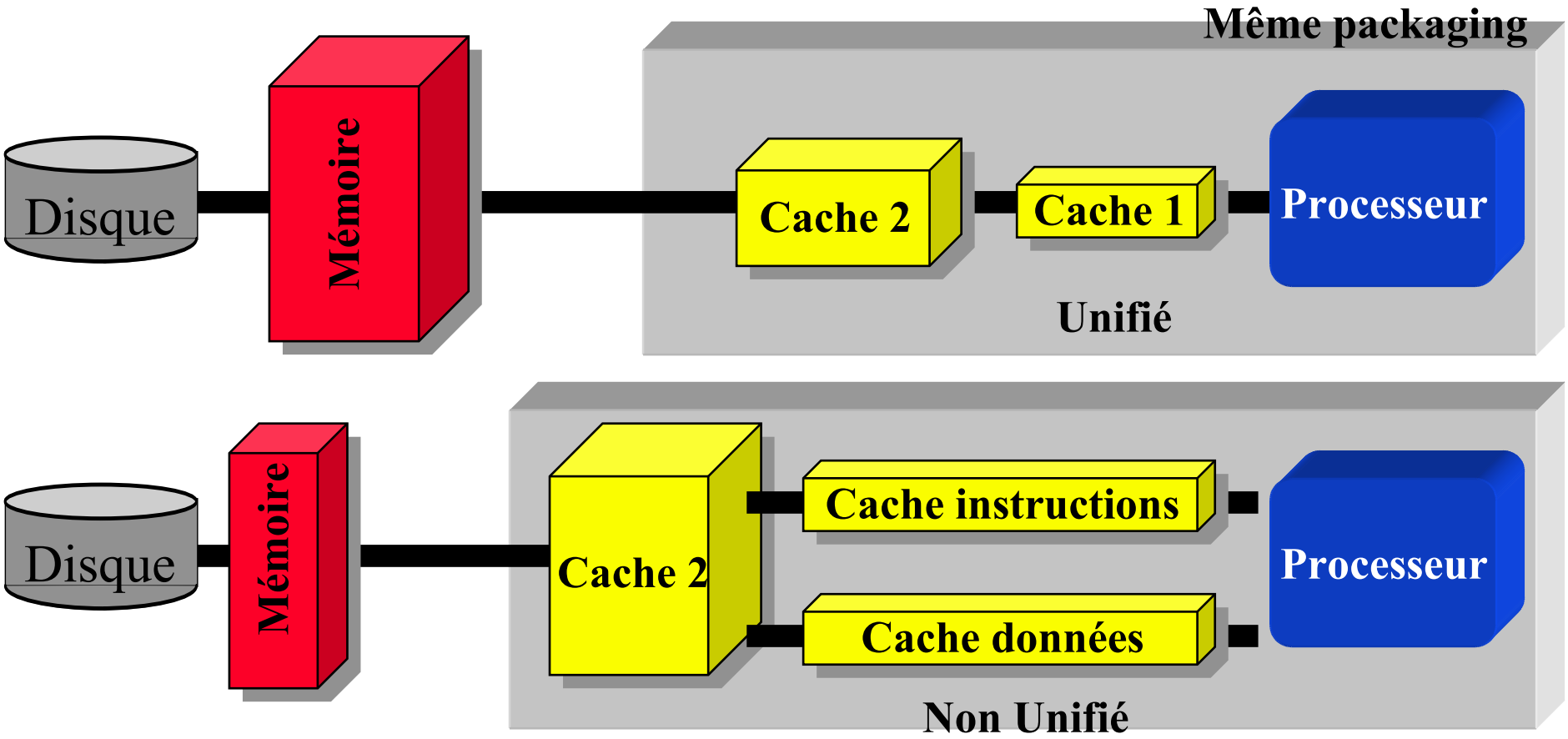
➢Conclusions des simulations :

- une taille de blocs trop petites engendre beaucoup de défauts de cache :
  - bien que la mise en place de blocs de petites tailles apporte une plus grande souplesse dans la gestion des remplacements, il s'avère que le nombre de défauts est lui augmenté
- une associativité trop importante n'apporte pas grand chose :
  - ceci explique pourquoi la plupart des caches que l'on trouve dans les processeurs récents sont à associativité par ensemble de 2, 4 voir 8, mais jamais au delà

Organisation mémoire

- Evolution :
  - les tailles disponibles augmentent
  - les temps de cycle diminuent
  - elles sont de plus en plus souvent intégrées sur le circuit :
    - permet de les faire travailler avec le même temps de cycle que l'unité de calcul
    - permet d'augmenter la taille des bus entre cache et contrôleur (et ceci de façon transparente)
      - 32, 64, 128 bits --- 1, 2, 4 instructions chargées à chaque accès
  - nombre de niveaux augmente
  - nécessite une gestion efficace des remplacement de blocs
  - cache de premier niveau souvent non unifiés :
    - cache instructions et cache données
  - cache de second niveau souvent unifiés

Organisation mémoire



–plus proche de l'architecture Harvard

–permet la réalisation de politiques de gestion des caches différentes

–mise en place de caches non bloquant :

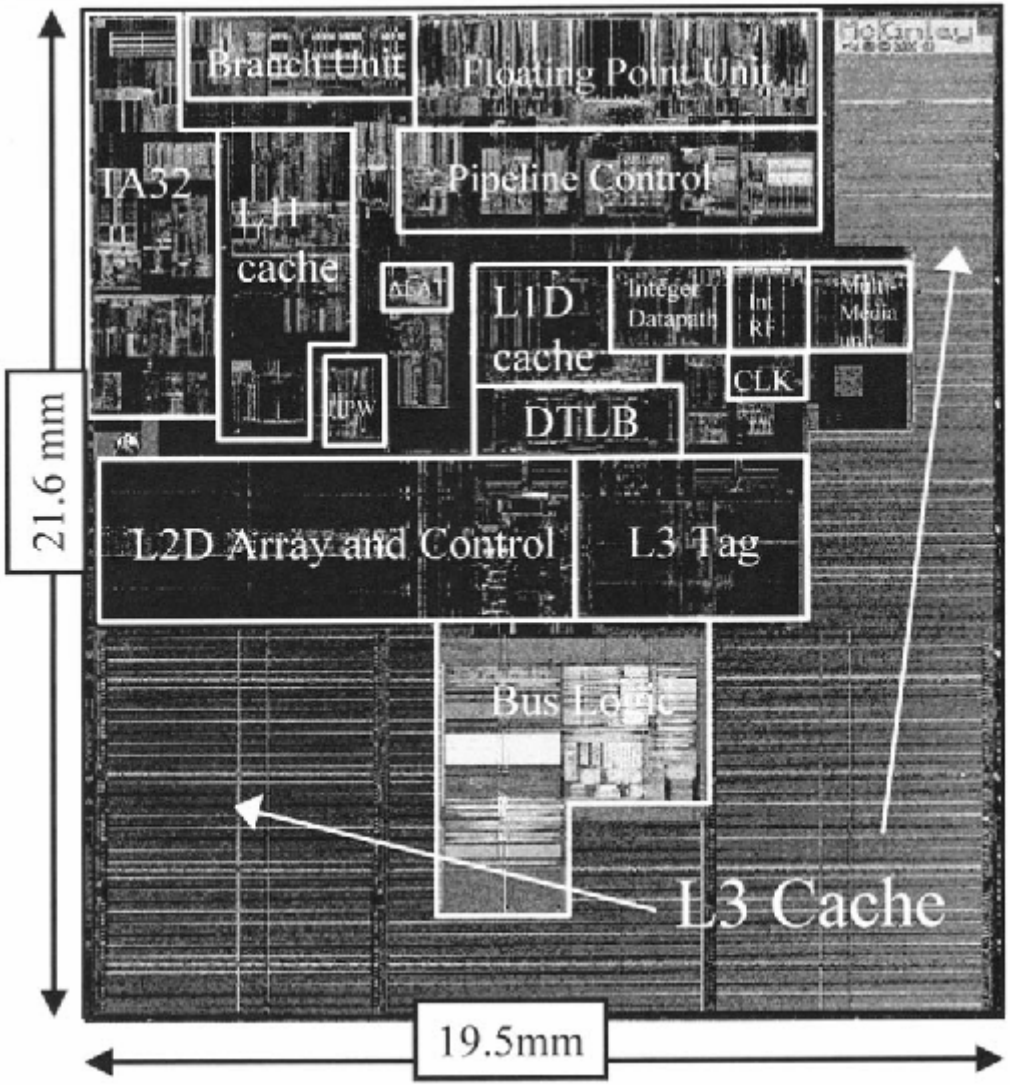
•lors d'un défaut de cache, le processeur peut continuer son exécution (sous entend une exécution dans le désordre)

Organisation mémoire

➢Et pourquoi pas un troisième niveau ?

♦ Solution mise en œuvre dans l'Itanium 2 :

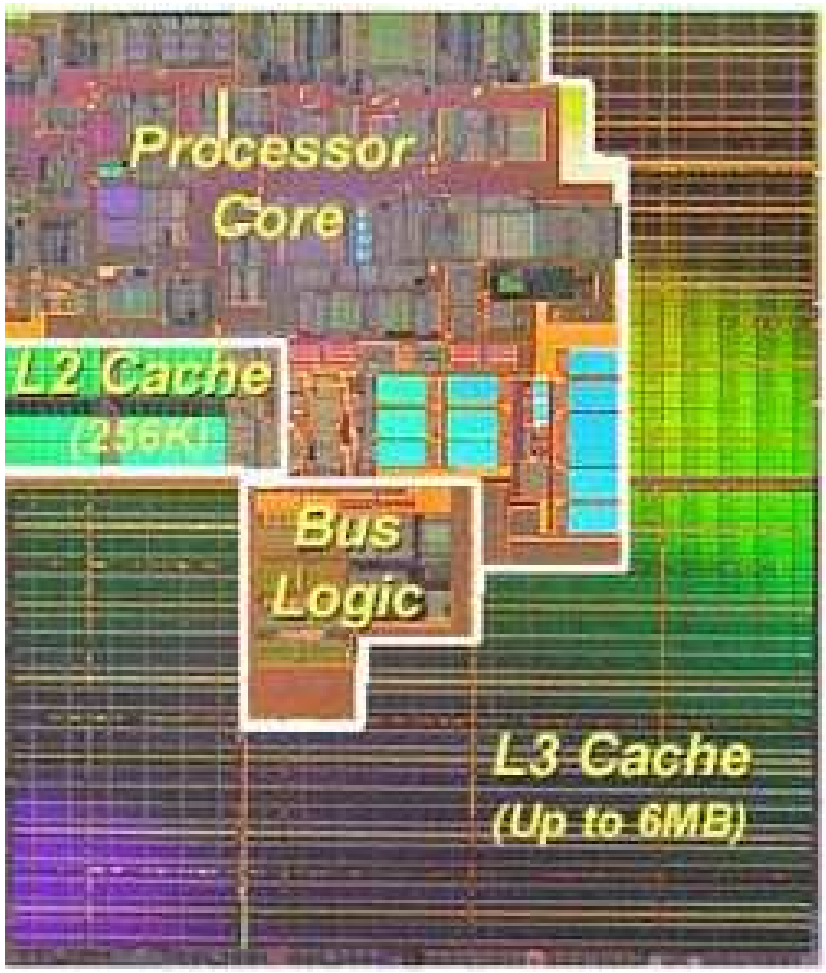
– 3MB pour le niveau 3



Organisation mémoire

➢Itanium 2, cœur Madison :

♦ 410 million de transistors !!!



Organisation mémoire

➢ Amélioration des performances des caches : (1600 articles entre 89 et 95)

♦ 3 axes de recherches important :

– 1) réduction du taux d'échec

– 2) réduction de la pénalité d'échec

– 3) réduction du temps de l'accès réussi au cache

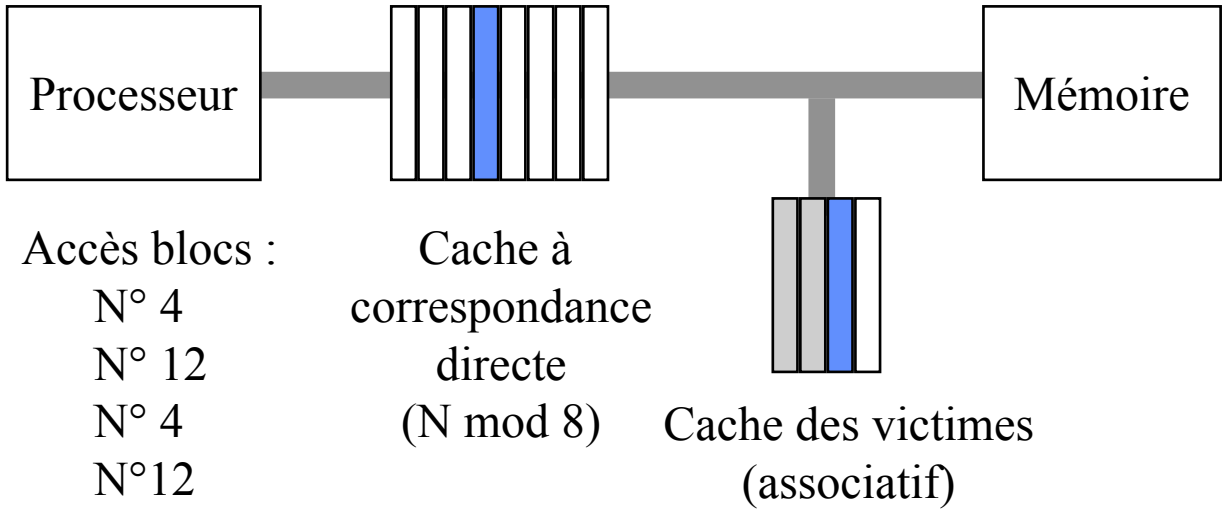
Organisation mémoire

♦ 1) réduction du taux d'échec :

- augmentation de la taille du cache
- augmentation de la taille des blocs :
  - ils tirent profit de la localité spatial, *si le processeur exécute l'instruction de l'adresse @i au temps t, il y a de forte chance qu'il exécute très prochainement l'instruction de l'adresse @i+1*
  - augmente la pénalité d'échec, plus d'information à transférer lorsqu'il y a échec
  - nombre de blocs moins important, donc moins souple
- associativité plus grande :
  - cache plus souple
  - en général, augmente le temps d'accès au cache (plus complexe)
- mise en place d'un cache des victimes :
  - ajout d'un tout petit cache entre le cache et la mémoire centrale
  - ce cache contient seulement les blocs qui sont rejetés du cache à cause d'un échec
  - permet de supprimer entre 20 et 95 % des échecs du au remplacement d'un bloc par un autre et vice versa

Organisation mémoire

– Exemple de cache de victime :



Organisation mémoire

- mise en place de caches pseudo associatifs :
  - c'est un cache à correspondance directe amélioré
  - lorsqu'il y a échec lors d'un accès, et avant d'aller chercher dans la mémoire principale, on va tester un autre bloc : une manière simple de le faire est d'inverser le bit le plus significatif de l'index
- lecture anticipée par matériel :
  - on anticipe la demande du processeur avant qu'il ne réclame (le processeur 21064 lit le bloc demandé (placé dans le cache) et le bloc suivant
- lecture anticipée par le compilateur :
  - instructions de pré chargement disposées par le compilateur dans le code
  - la lecture anticipée n'a de sens que si le processeur peut continuer à travailler pendant que la donnée est en cours de lecture, ce qui conduit à la mise en place de *cache non bloquant*

Organisation mémoire

- optimisation du compilateur :
  - pas de modification du matériel
  - amélioration sur les échecs de lecture d'instructions ou sur les accès aux données
  - le code peut *facilement* être arrangé tout en restant correct (ré ordonnancement des instructions)
  - les données peuvent elles aussi être ré ordonnées dans la mémoire pour améliorer les localités spatiale et temporelle

Organisation mémoire

♦ 2) réduction de la pénalité d'échec :

- priorité aux lectures par rapport aux écritures : une lecture bloque le processeur, pas une écriture
- redémarrage précoce :
  - lors d'un défaut d'accès à la donnée **di**, on rapatrie le bloc de la mémoire vers le cache, dès que la donnée est arrivée dans le cache on la fait passer au processeur qui peut ainsi continuer, en parallèle le bloc continue à être rapatrié
- mot critique en premier :
  - on va chercher en mémoire d'abord le mot manquant et on l'envoie au processeur, ensuite on remplit le bloc de cache en question (parallèlement, le processeur peut continuer)
- ces deux dernières techniques ne sont profitables que pour les tailles de blocs importantes
- cache non bloquant, permet au processeur de poursuivre son exécution en attendant la donnée

Organisation mémoire

- second niveau de cache :
  - répond aux exigences d'augmentation de la taille du cache et de la diminution du temps d'accès
  - premier niveau de cache est assez petit et donc très rapide
  - second niveau beaucoup plus grand et qui limite les accès à la mémoire principale
  - on parle alors d'échec local (T_{LC1} et T_{LC2}) à un cache et d'échec global (T_{LC1} * T_{LC2})

♦ 3) réduction du temps de l'accès réussi :

- caches simples et petits :
  - permet une comparaison de l'adresse avec les étiquettes du cache rapide
  - si le cache est petit il tiendra plus facilement dans le boîtier du processeur (pas de sortie vers l'extérieur qui ralentit les accès

Pipeline

Pipeline des processeurs

- **Comment diminuer les temps de traitement des données ?**
  - ➢ augmentation de la fréquence d'horloge
  - ➢ travaille en parallèle sur plusieurs données indépendante

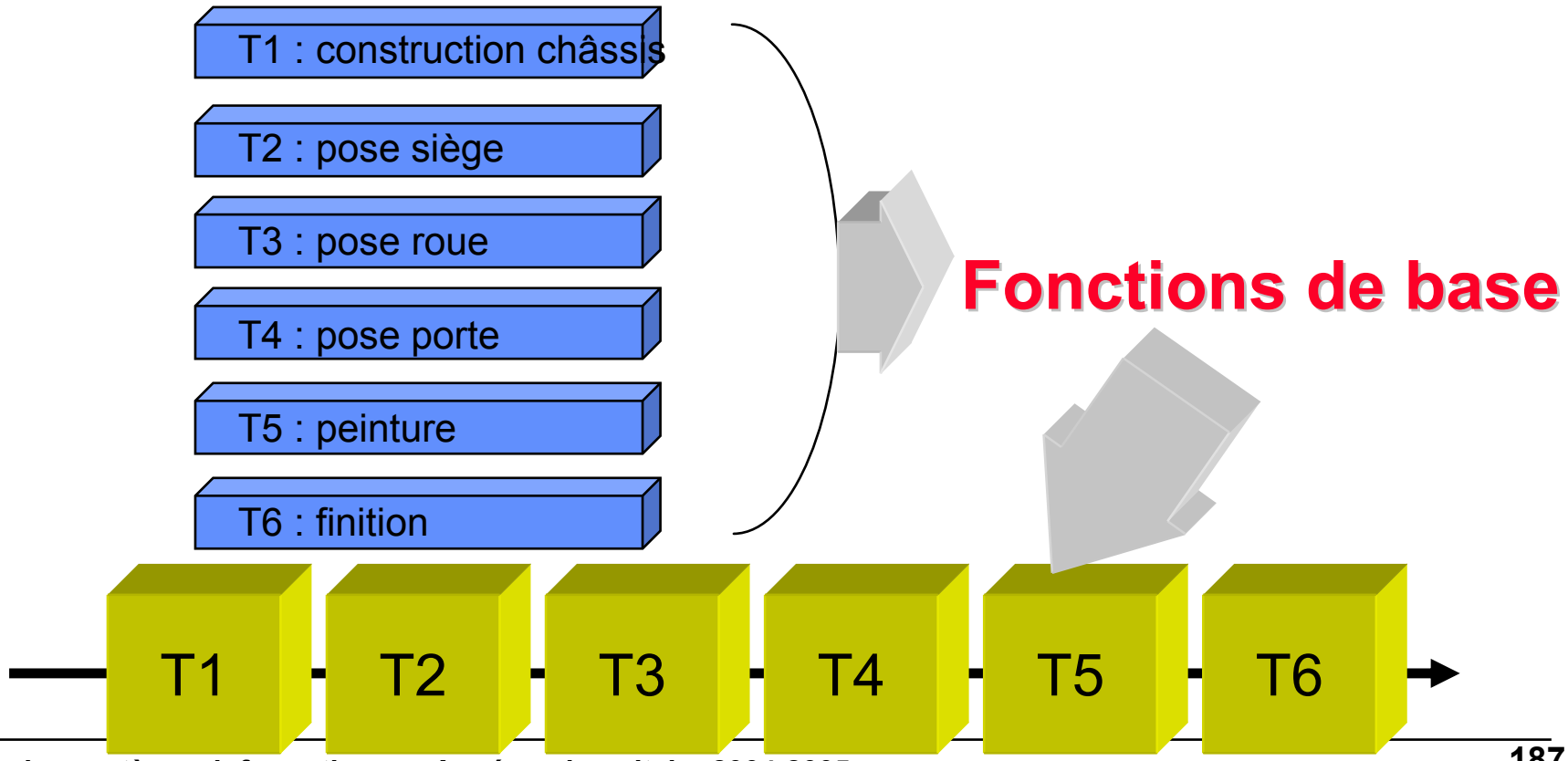
Parallélisme

  - ➢ travaille à la chaine

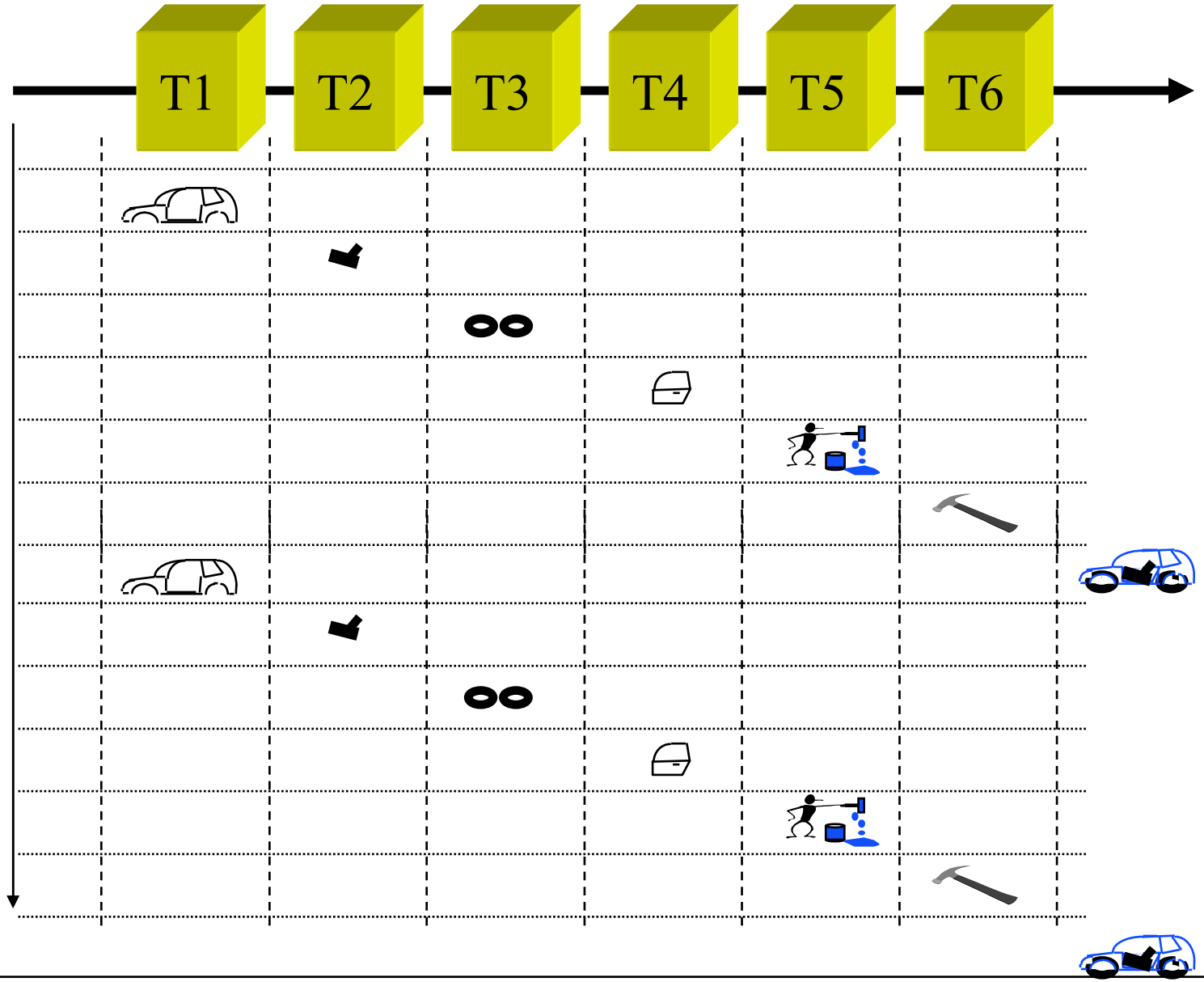
Pipeline

Pipeline des processeurs

- **Le pipeline :**
  - Exemple :
    - Soit la tâche : construction d'une automobile
    - Cette tâche est décomposable en sous tâches :

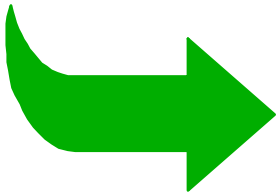


Pipeline des processeurs



Pipeline des processeurs

- **Coût d'une telle production :**
  - ➢ 1 seul ouvrier :
    - ♦ non spécialisé, donc sachant tout faire



mais pas de façon optimale

- **Temps de production d'une pièce :**
  - ➢ 6 unités de temps

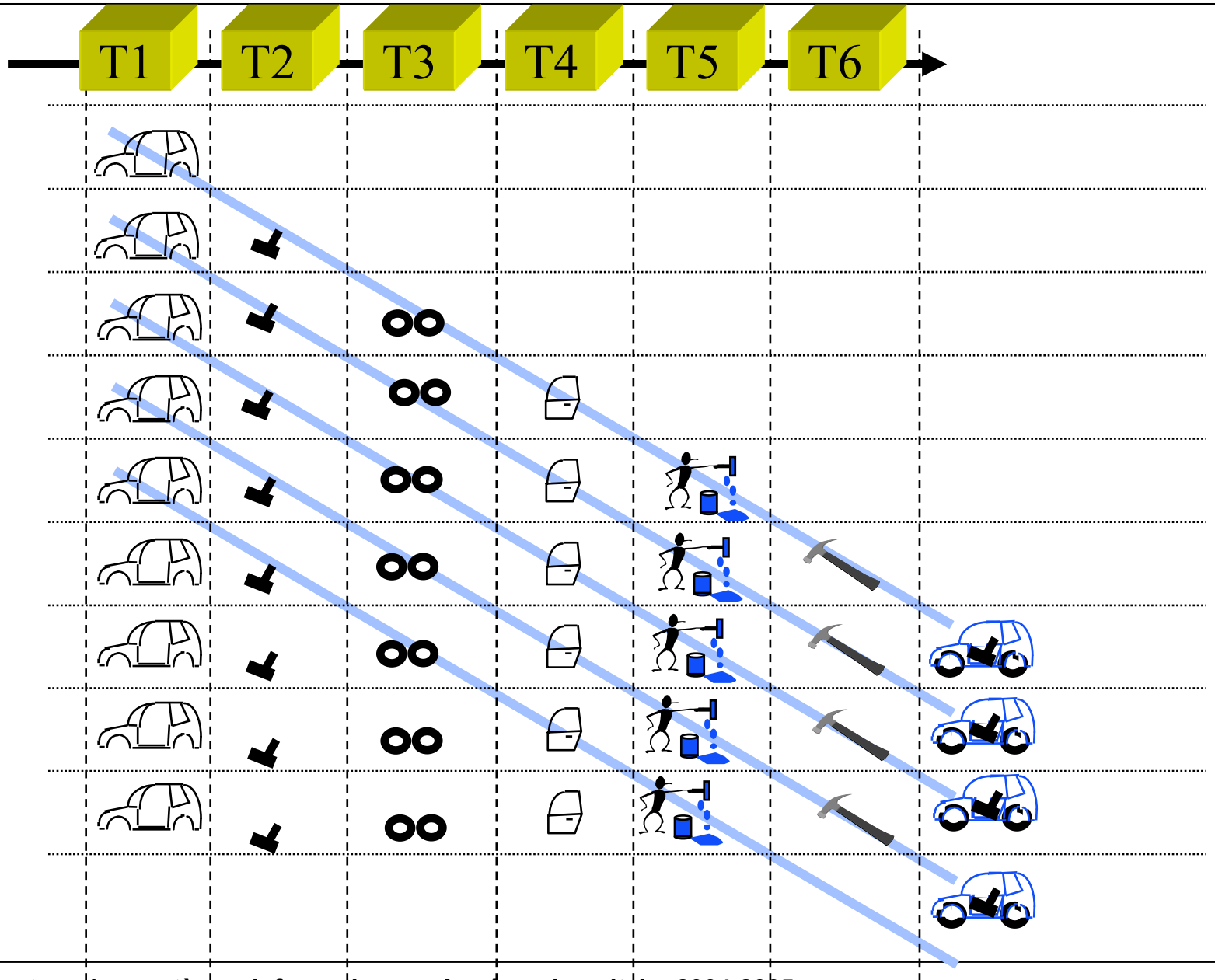
- **Cadence de production :**
  - ➢ 1 nouvelle pièce toutes les 6 unités de temps

Pipeline des processeurs

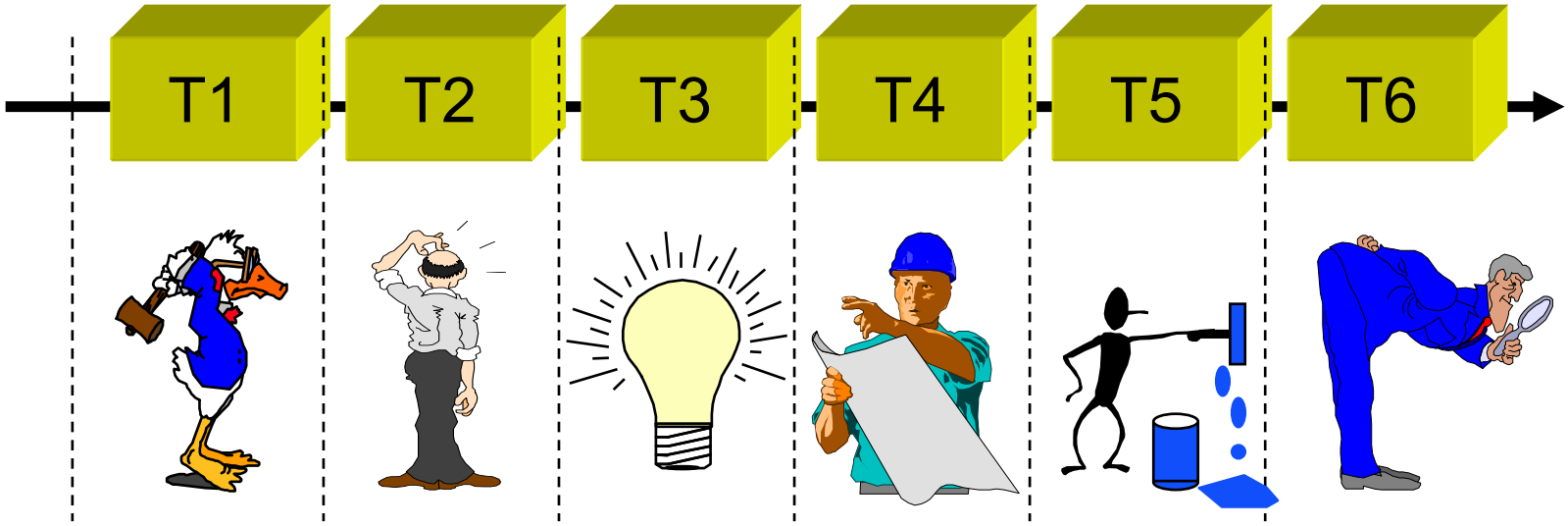
Comment augmenter la production ?

- **Placer un ouvrier par tâche élémentaire :**
  - ouvrier spécialisé :
    - donc ne sachant faire qu'une opération élémentaire
    - traitement optimal

Pipeline des processeurs



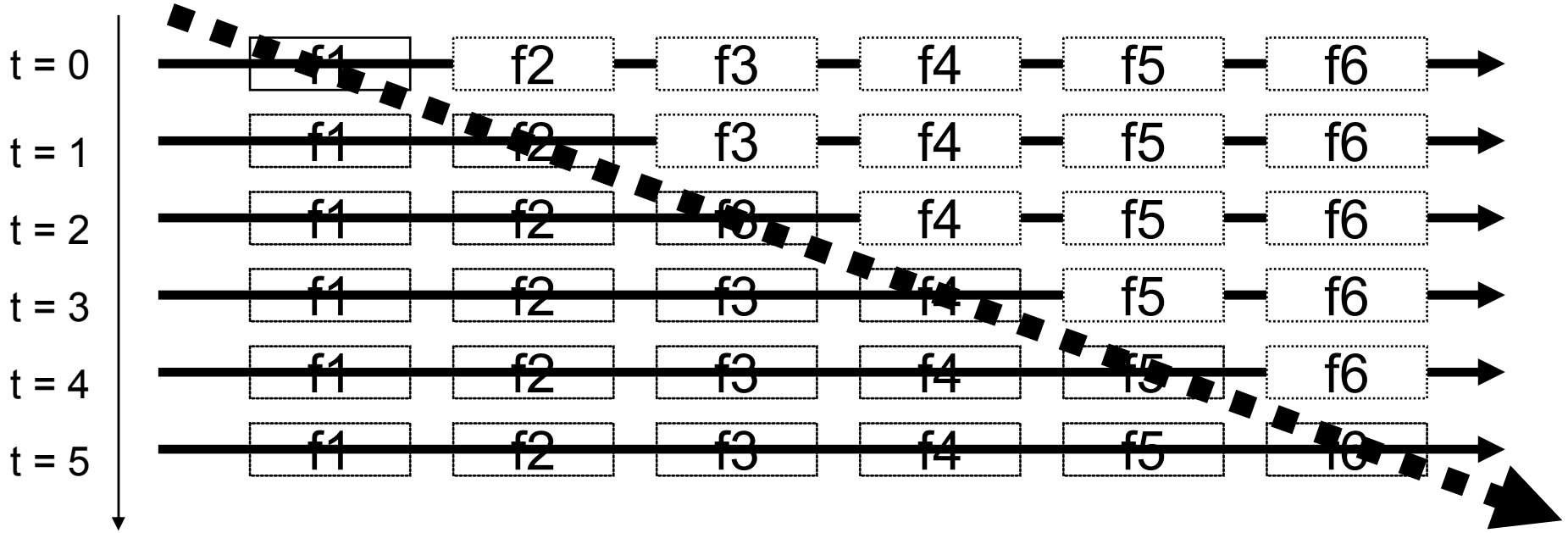
Pipeline des processeurs



- **Coût d'une telle production :**
  - ➢ 6 ouvriers :
- **Temps de production d'une voiture :**
  - ➢ 6 unités de temps
- **Cadence de production :**
  - ➢ 1 nouvelle voiture toutes les unités de temps

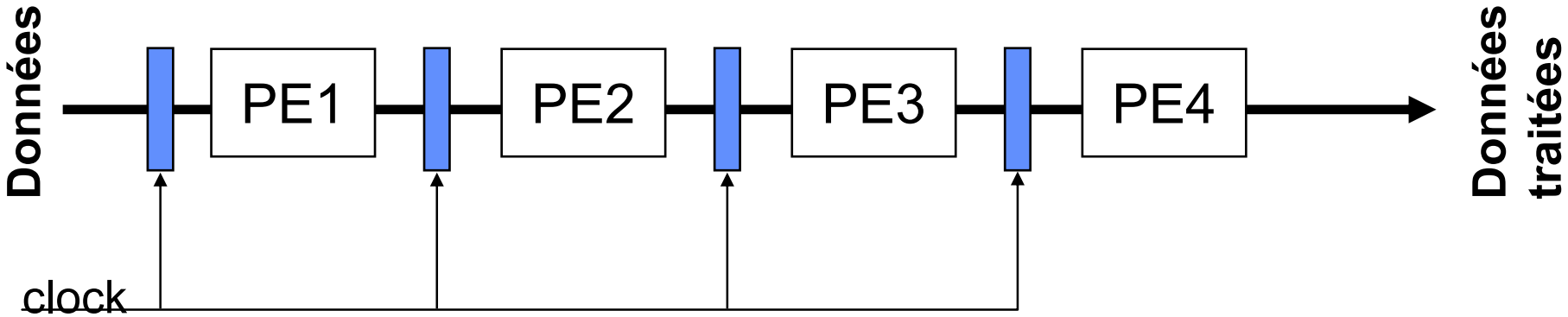
Pipeline des processeurs

- Soit le traitement F à réaliser
- Soit la décomposition F = fn o fn-1 o ..... o f2 o f1
- Soient tn, tn-1, ..., t2 et t1 les temps de traitement de chacune des tâches élémentaires
- Temps de traitement d'une donnée : Tps = $\sum$ ti
- Cadence de traitement : Cadence = max (ti)



Pipeline des processeurs

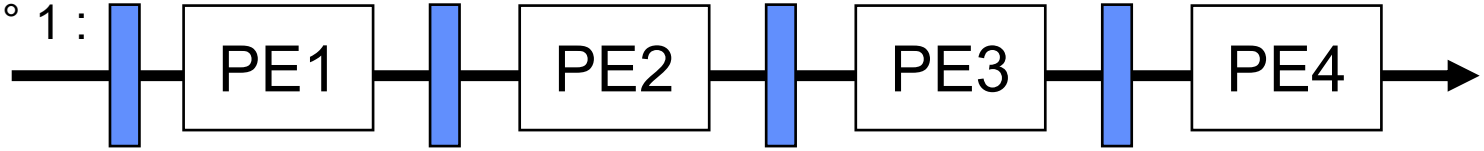
- **Pour le traitement numérique du signal :**
  - ➢fonction élémentaire combinatoire
  - ➢il faut stabiliser les données en entrée de chaque fonction élémentaire
  - ➢on impose une cadence commune : max (ti) + Treg



Pipeline des processeurs

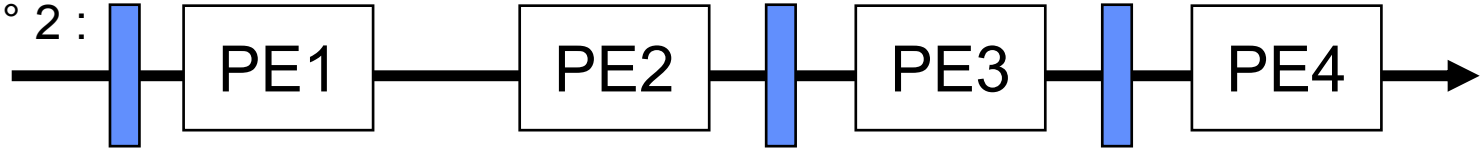
- Soient T(PE1) = 1, T(PE2) = 1, T(PE3) = 2, T(PE4) = 1

Découpage N° 1 :



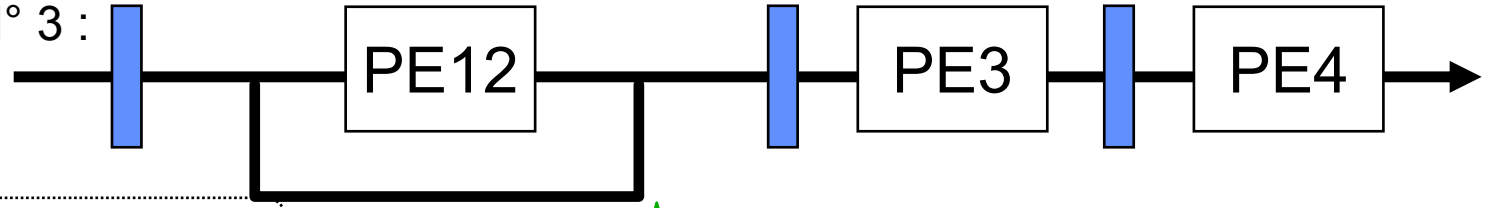
4 processeurs élémentaires
4 registres

Découpage N° 2 :



4 processeurs élémentaires
3 registres

Découpage N° 3 :



3 processeurs élémentaires
3 registres

Ajout d'un certain contrôle pour réutiliser le processeur élémentaire

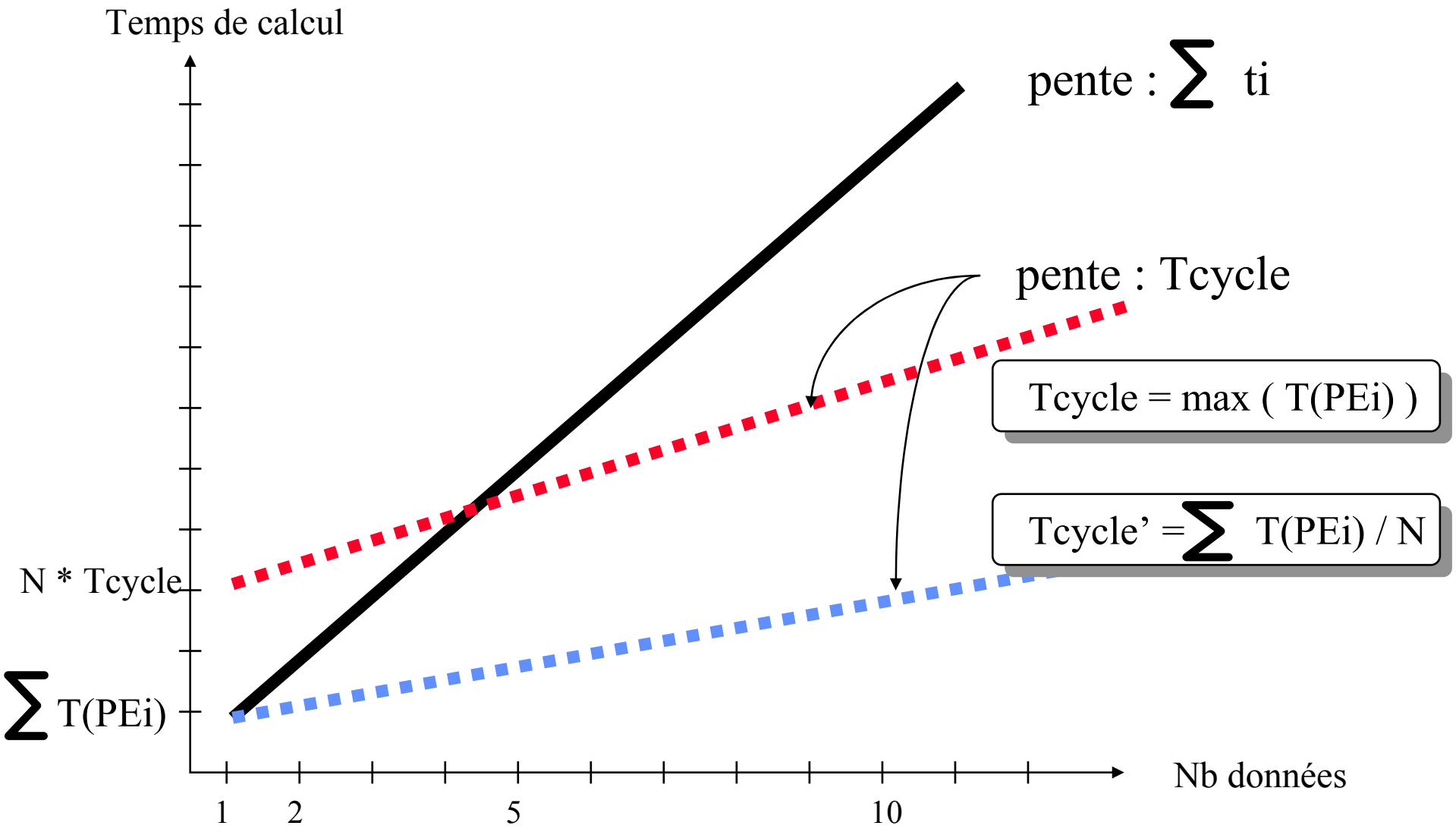
Cadence = 2 unités de temps

Pipeline des processeurs

Généralisation

- soit **TpsCalcul** le temps de calcul de l'ensemble des traitements
- soit **N** le nombre de tranches de pipeline
- soit **Tcycle** le temps de cycle (cadence des traitements) :
  - **Tcycle = TpsCalcul / N + Treg**
- soit **D** le nombre de données à traiter
- soit **T(D)** le temps d e traitement de **D** données :
  - **T(D) = (D + N - 1) * Tcycle**
- le temps de traitement par donnée est donc ramené à :
  - **T(D) / D = (D + N - 1) * (TpsCalcul / N + Treg) / D**
- si **D ->** ∞ et **D >> N** alors le temps de traitement par donnée est
  - **T(D) / D = TpsCalcul / N + Treg**
- si N -> ∞ et D << N alors le temps de traitement par donnée est :
  - **T(D) / D = N / D * Treg**

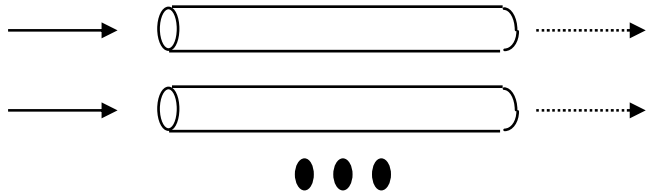
Pipeline des processeurs



Pipeline des processeurs

❑ Problème de l'amorçage :

- ➢ t = 0 d1
- ➢ t = 1 d2
- ➢ t = n-1 dn
- ➢ t = n dn+1 → d1
- ➢ t = n+1 dn+2 → d2



Pipeline des processeurs

❑ Qu'est ce que l'on peut pipeliner ?

➢ des opérateurs :

- ♦ multiplieur, planche suivante
- ♦ opérations flottantes dans les processeurs

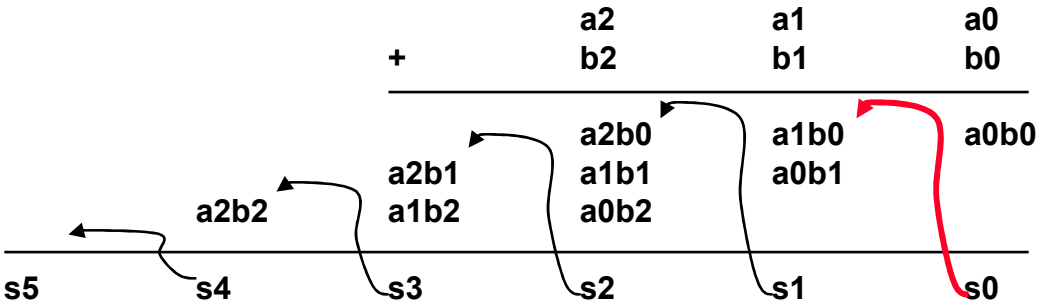
➢ du contrôleur d'un processeur :

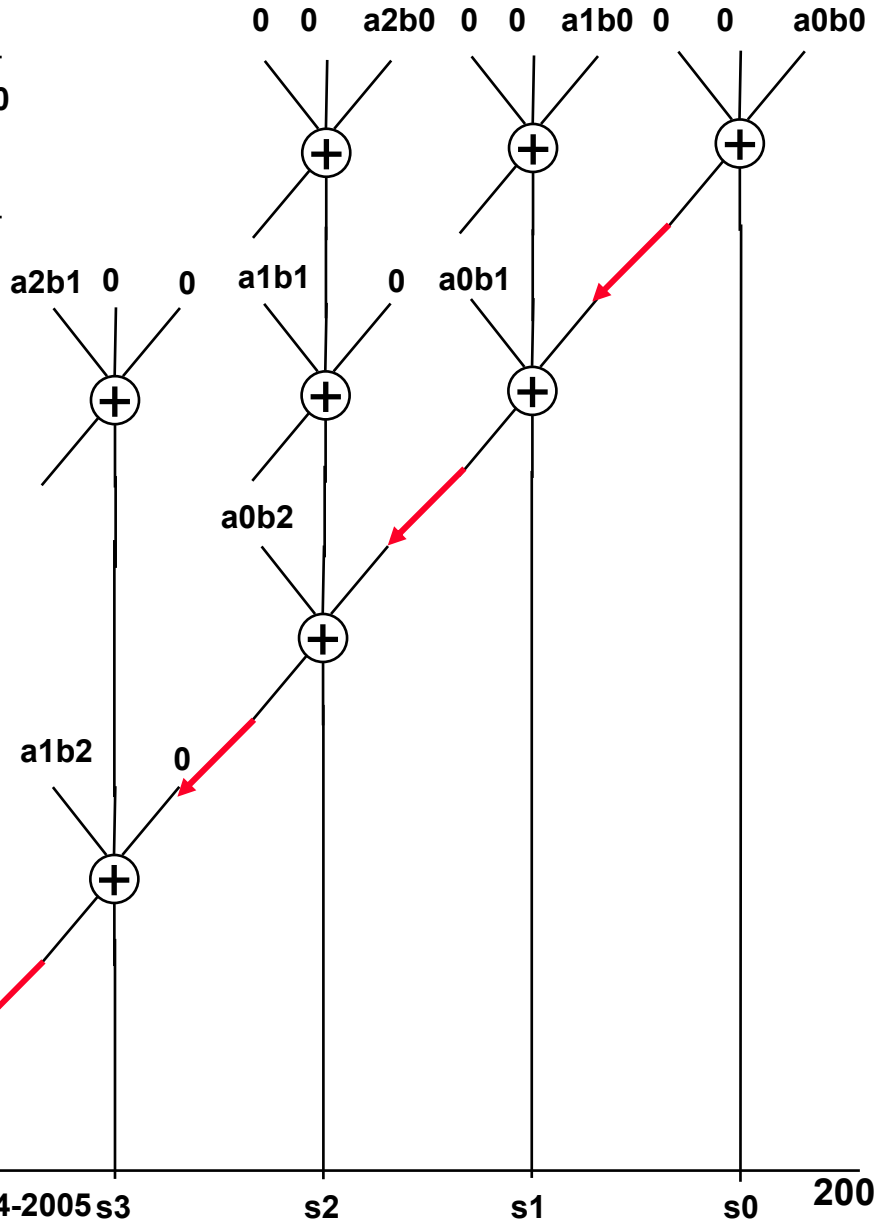
- ♦ étapes LI, DI, EX, WR =====> étages pipeline
- ♦ Processeur RISC

➢ les différentes unités d'un système :

- ♦ mémoires, unité de calcul

Pipeline des processeurs

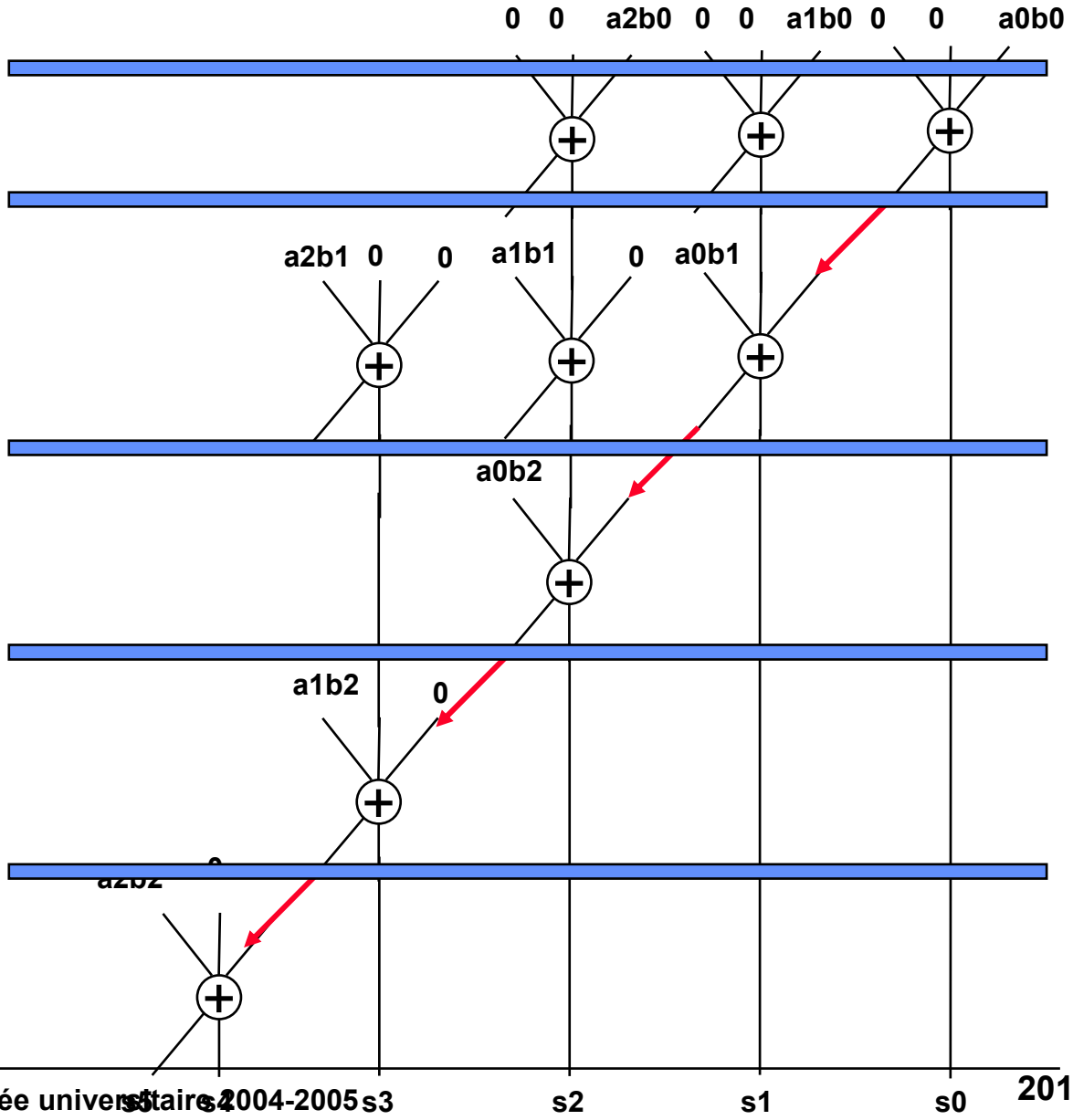




- fonction combinatoire :
  - TpsCalcul = 5 UT
  - 1 UT = TpsAdd

Pipeline des processeurs

- mise en place de registres :
  - 5 étages
  - TpsCalcul * = 5 UT
  - Cadence Calcul = 1 UT
  - 1 calcul toutes les UT

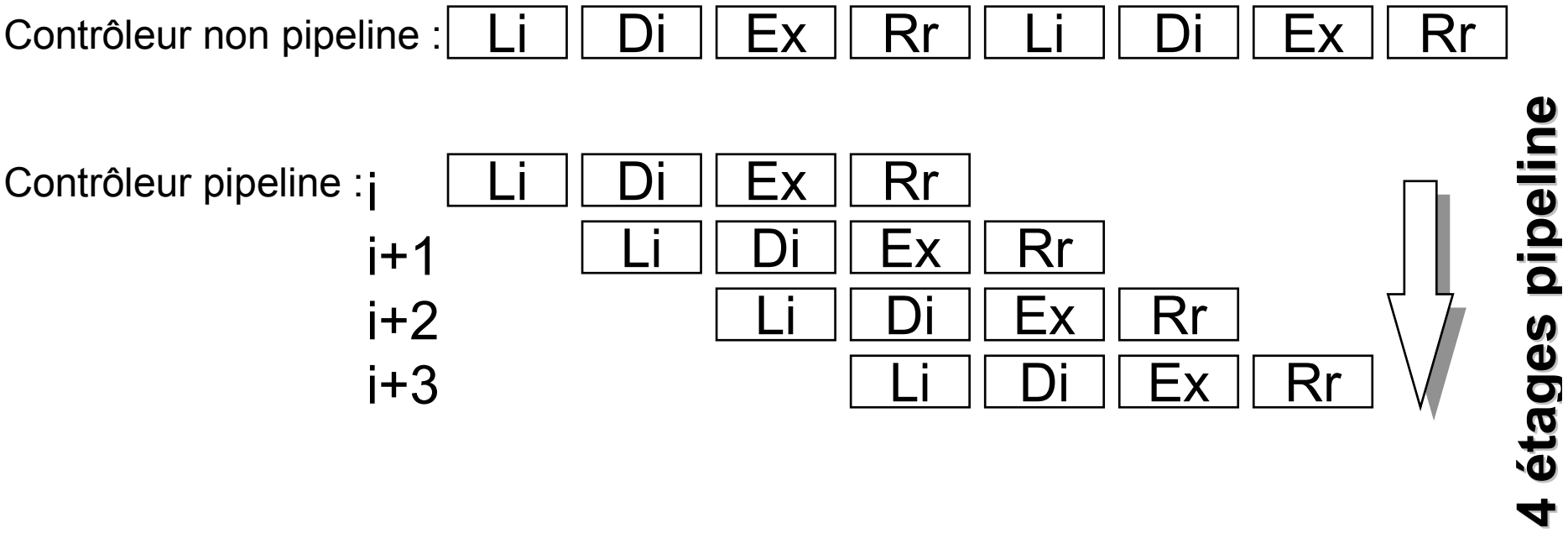


Pipeline des processeurs

❑ Pipelinage d'un contrôleur :

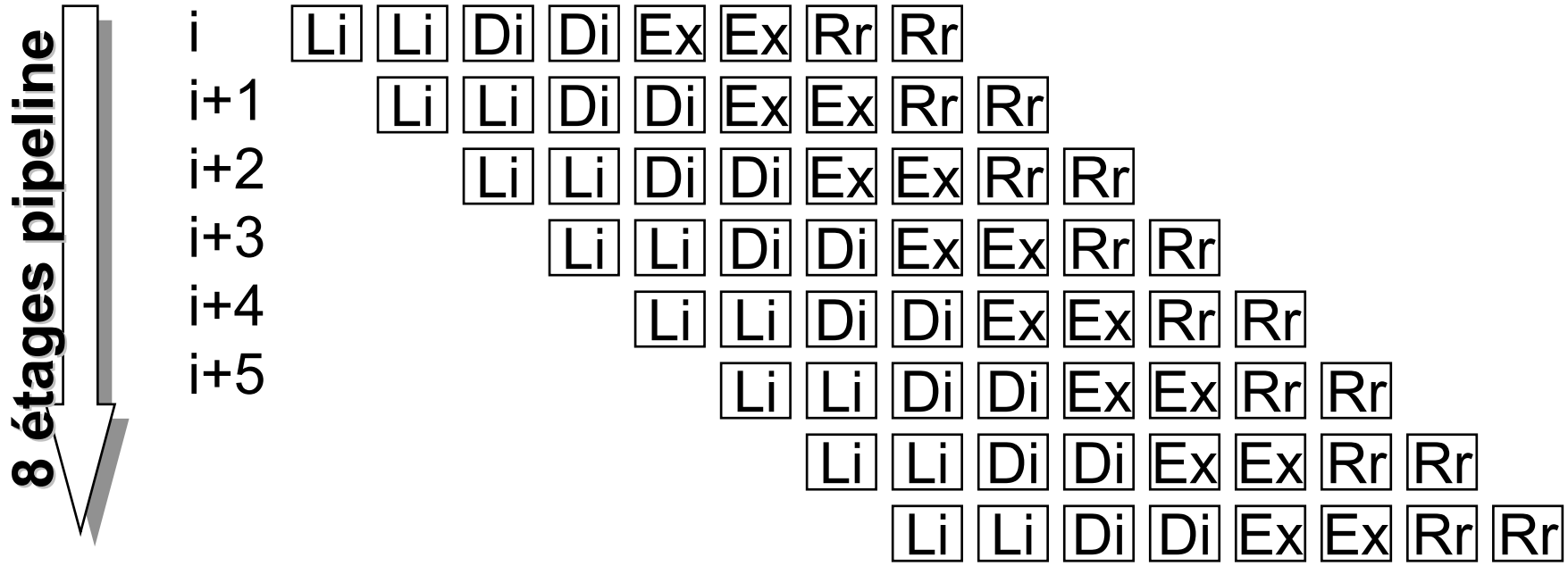
➢ le cycle Von Neumann :

- ♦ Li : lecture instruction
- ♦ Di : décodage instruction
- ♦ Ex : exécution d'instruction
- ♦ Rr : rangement du résultat



Pipeline des processeurs

- On fractionnant encore les tâches en sous tâches,
  1) on diminue le temps de cycle,
  2) on augmente le nombre d'étages de l'architecture

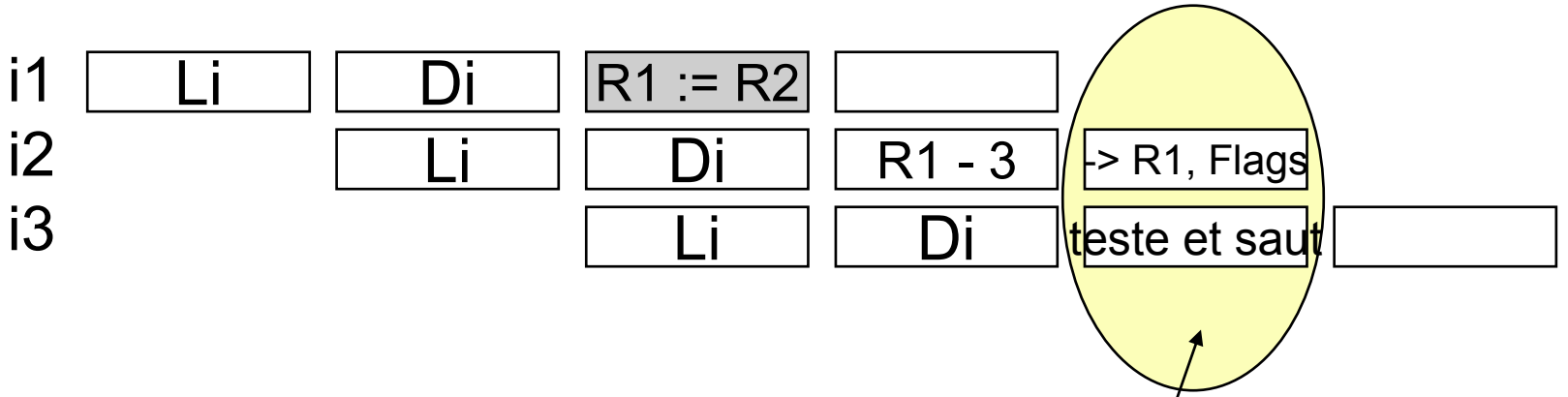


Pipeline des processeurs

❑ Enchaînement du cycle Von Neumann :

➢soient les instructions :

- ♦ i1 : move R1, R2 R1 := R2
- ♦ i2 : sub R1, 3 R1 := R1 -3
- ♦ i3 : beq @adresse si R1 = 0 alors saut



Les flags ne sont pas encore valides lorsqu'on les teste

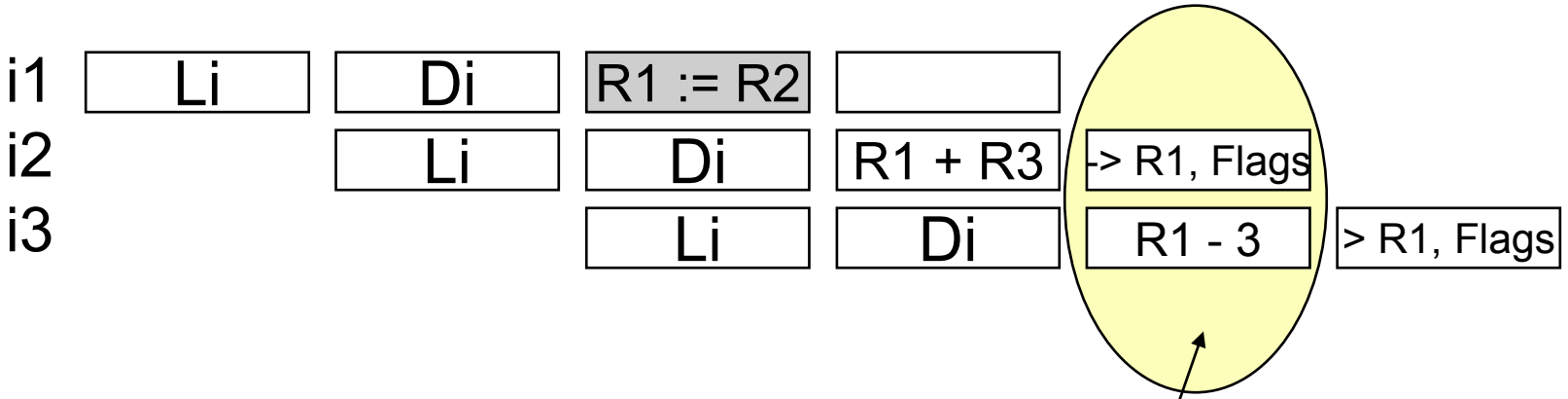
Dépendance entre les instructions

Pipeline des processeurs

❑ Enchaînement du cycle Von Neumann :

➢soient les instructions :

- ♦ i1 : move R1, R2 R1 := R2
- ♦ i2 : add R1,R3 R1 := R1 + R3
- ♦ i3 : sub R1, 3 R1 := R1 -3



Le registre R1 n'est encore chargé

Dépendance entre les données

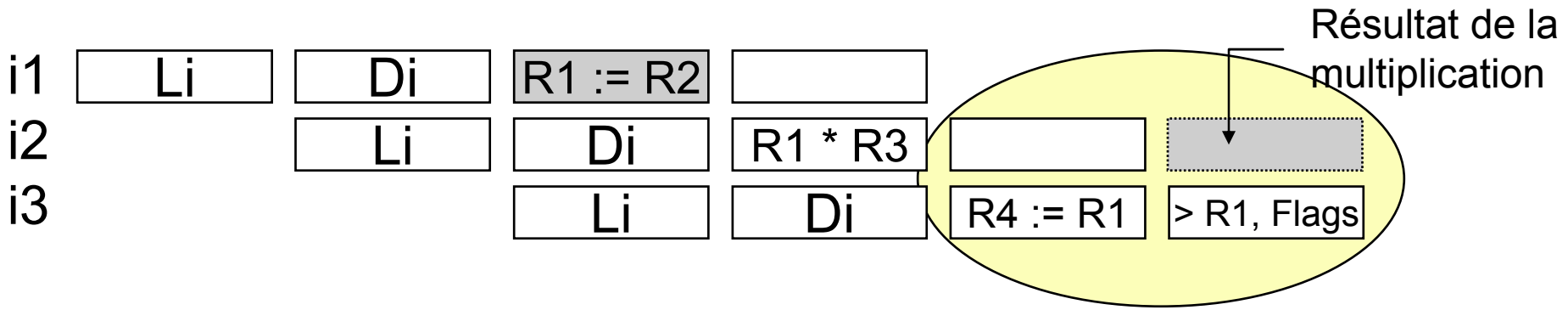
Pipeline des processeurs

❑ Enchaînement du cycle Von Neumann :

➢ soient les instructions :

- ♦ i1 : move R1, R2 R1 := R2
- ♦ i2 : mult R1,R3 R1 := R1 * R3
- ♦ i3 : move R4, R1 R4 := R1

➢ avec l'opération multiplication réalisée sur un opérateur pipeline à 2 étages



Dépendance entre les données

Pipeline des processeurs

- **Le compilateur doit vérifier les dépendances :**
  - ➢entre données
  - ➢entre instructions
  - ➢la différence entre la cadence et le temps de calcul
- **Il ajoutera, si nécessaire des instructions NOP :**

♦ i1 : move	R1, R2	R1 := R2
♦ i2 : sub	R1, 3	R1 := R1 -3
♦ i3 : NOP		
♦ i4 : beq	@adresse	si R1 = 0 alors saut

♦ i1 : move	R1, R2	R1 := R2
♦ i2 : add	R1,R3	R1 := R1 + R3
♦ i3 : NOP		
♦ i4 : sub	R1, 3	R1 := R1 -3

Pipeline ou parallélisme ?

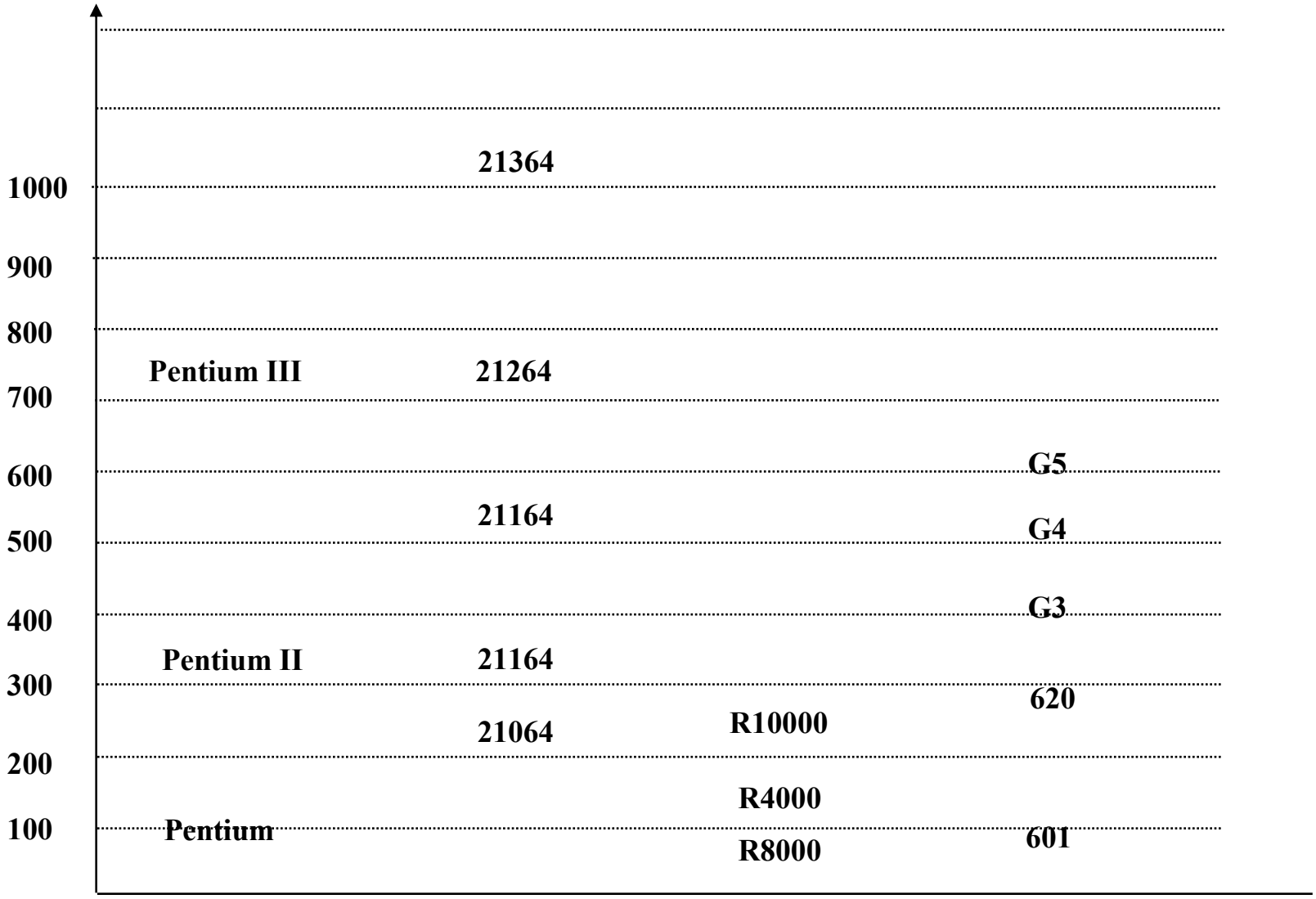
- **Architecture parallèle :**
  - ➢ augmentation de la capacité de traitement par l'exécution concurrente

- **Architecture pipeline :**
  - ➢ augmentation de la capacité de traitement par recouvrement temporel des traitements élémentaires.

AUGMENTATION DE PERFORMANCES

Augmentation de performances

❑ Evolution technologique



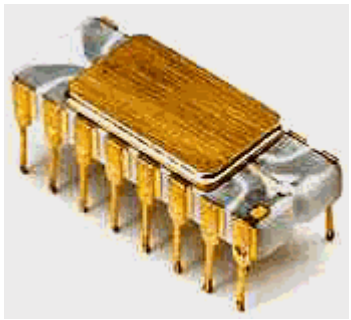
Augmentation de performances

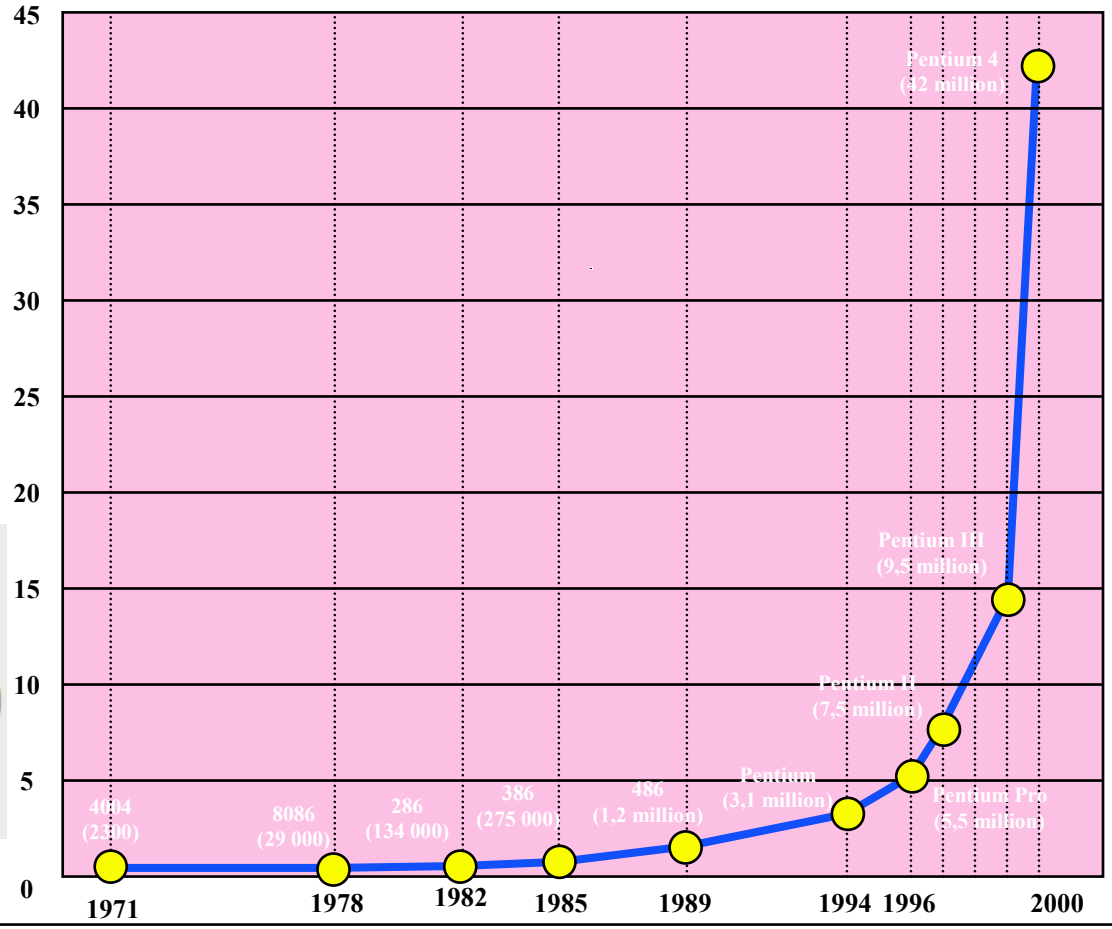
➢obtenue par :

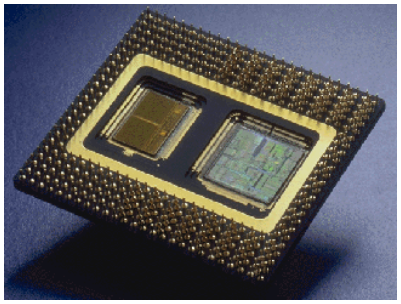
- ♦ l'apport technologique (1 μ m, 0.8 μ m, 0.5 μ m, 0.35 μ m, ...)
- ♦ le pipelinage :
  - – du contrôleur
  - – inter unités fonctionnelles
  - – intra unité fonctionnelle
- ♦ des fréquences élevées peuvent conduire à réduire la complexité du contrôleur :
  - – cas du 21164 dans lequel il n'existe pas d'exécution dans le désordre des instructions
- ♦ nécessite la mise en place de mémoires caches coûteuse :
  - – ayant le même temps de cycle que le processeur et placées dans le même boîtier que le processeur (cache interne)

Augmentation de performances

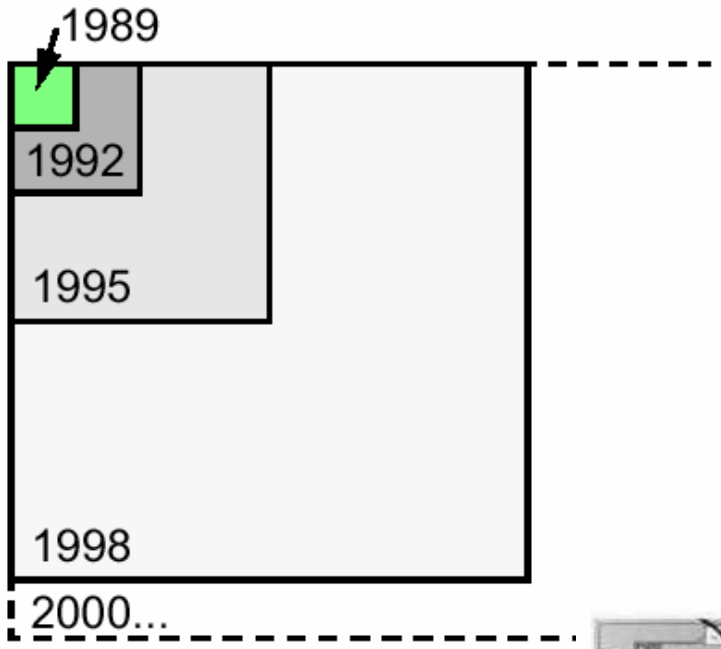
➢ Augmentation du nombre de transistors par puces

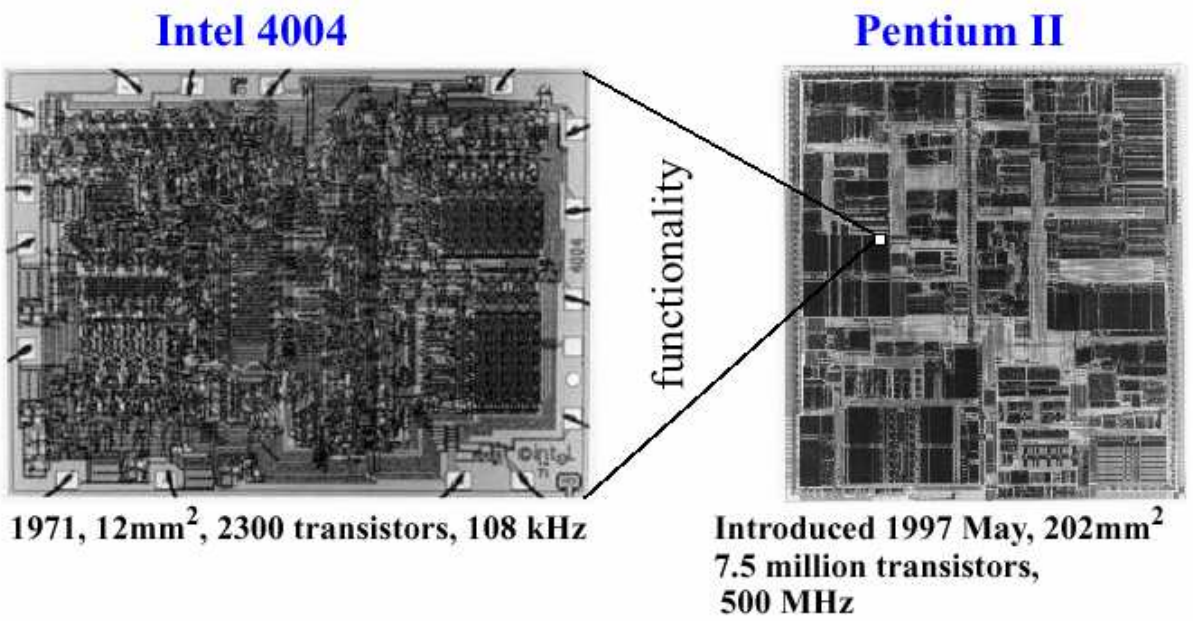






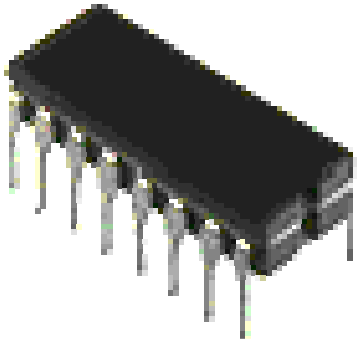
Augmentation de performances

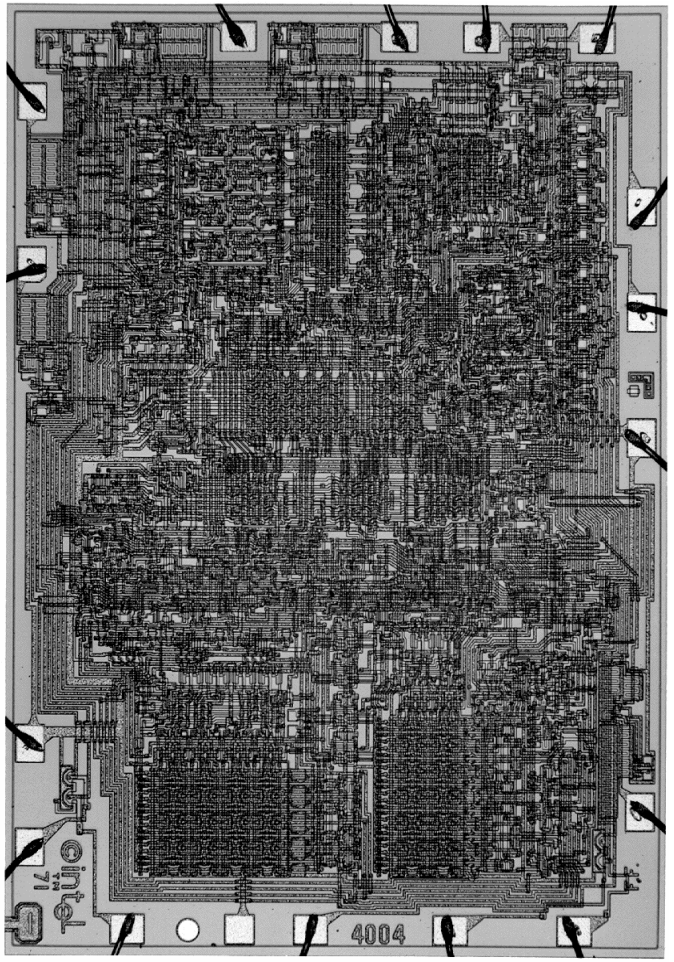




Augmentation de performances

- Intel 4004
  - 1971
  - 108 kHz
  - 4 bits
  - 1200 FF
  - 0,06 MOPS
  - 10 microns
  - 2300 transistors





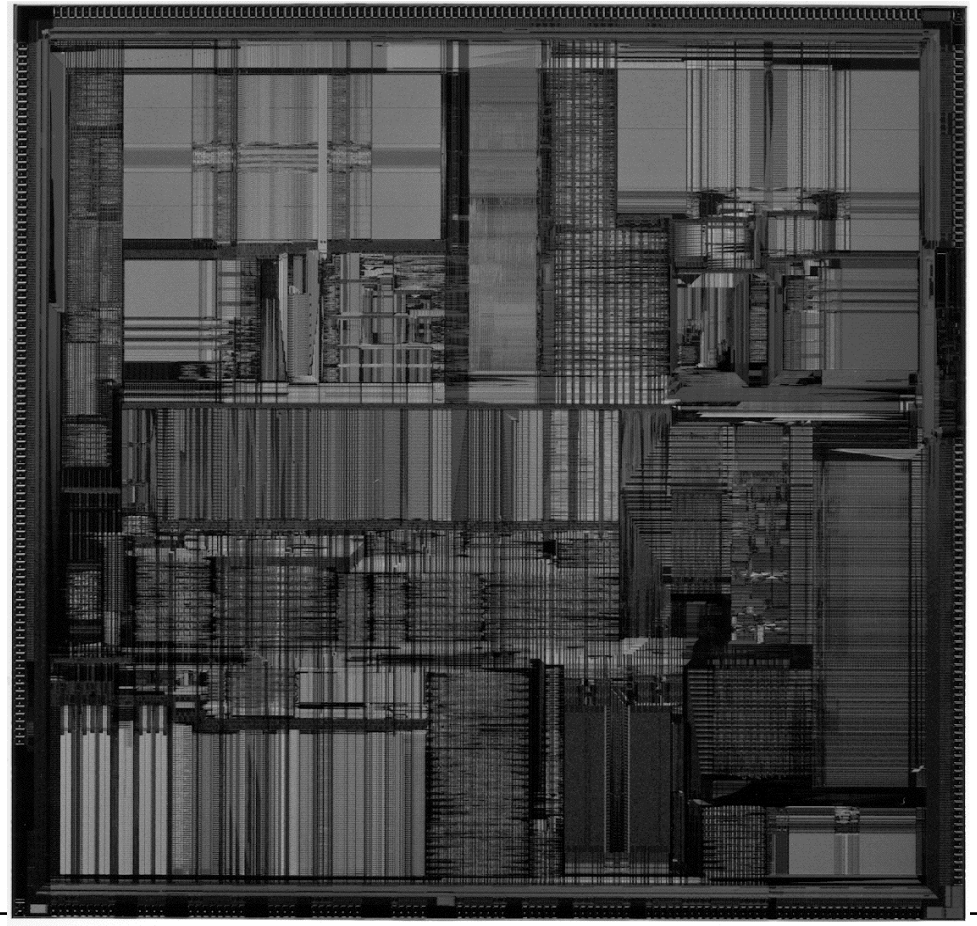
Augmentation de performances

- 1970 Mémoire 4Kbits MOS
- 1972 1er processeur : 4004 (Intel), techno. NMOS
- 1977 16K DRAM et 4K SRAM en production
- 1979 64K DRAM en production
- 1980 Processeur INTEL x86
- 1984 Processeur INTEL 80286 des PC AT
- 1986 1 mégabit DRAM
- 1988 TI/Hitachi 16-megabit DRAM
- 1990 Processeur INTEL 80286 avec fonctions multimédia
- 1990 Wafer de 20cm en production
- 1991 4 mégabit DRAM en production
- 1993 Processeur Pentium
- 1999 Processeur Pentium III
- 2000 Nouvelle architecture Intel, HP : IA 64 (Merced, Itanium)
- 2001 Processeur Pentium IV, 1,7 Ghz
- 2001 Successeur de l 'Itanium, 10 Ghz, 1 milliard de transistors, 100 milliards d 'opérations par seconde

Augmentation de performances

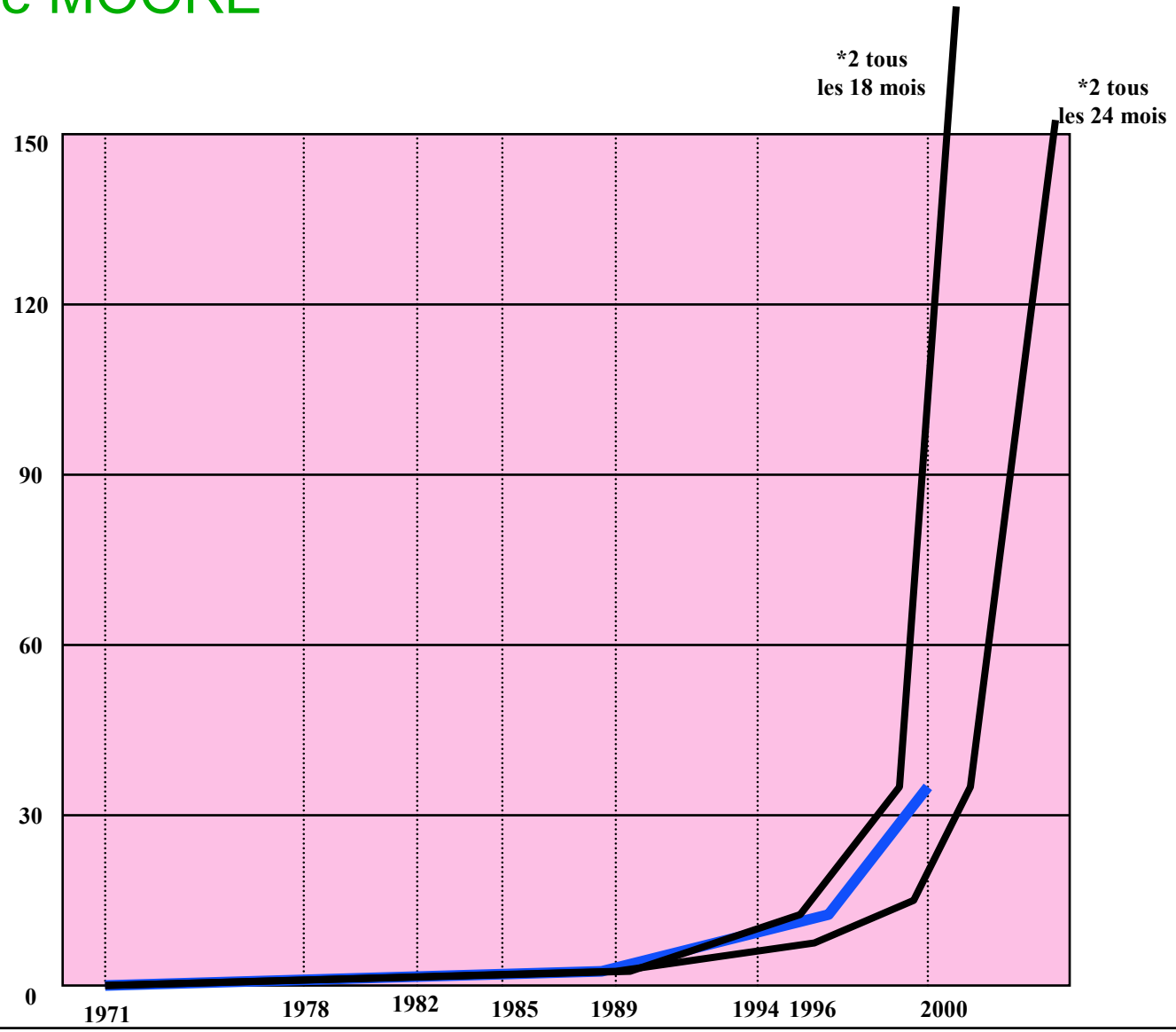
➢INTEL Pentium II :

- 1996
- 5.5 Millions de transistors de 0.35µ
- 200 MHz
- 2 cm^2
- Données 32 bits
- 3.3V, 35W



Augmentation de performances

➢Loi de MOORE



Augmentation de performances

➢Tous les 3 ans une nouvelle technologie

Année d'introduction	1997	1999	2001	2003	2006	2009	2012
Technologie (µm)	0,25	0,18	0,15	0,13	0,1	0,07	0,05
Fréquence (MHz)	750	1250	1500	2100	3500	6000	10000
Mémoire	256M	1G		4G	16G	64G	256G
Nombre de transistors	11M	21M	38M	77M	202M	520M	1350M
Coût par M Tr (Cent.)	500	290	166	97	42	18	8

➢Réduction d'un facteur k

- ♦ vitesse augmente d'un facteur k
- ♦ mémoire augmente d'un facteur k^2

Augmentation de performances

❑ Augmentation du nombre de registres

- ➢ tendances :
  - ♦ les processeurs CISC consistaient à proposer de plus en plus d'instructions travaillant avec la mémoire, donc peu de registres
  - ♦ avec les processeurs RISC, toutes les instructions s'exécutent sur les registres, donc il y a un besoin important en nombre de registres
- ➢ registres généraux ou dédiés :
  - ♦ pentium : CISC
    - – 8 registres généraux (héritage de la famille Intel)
    - – 8 registres dédiés aux opérations flottantes :
      - – implémentés sous forme de PILE
      - – toutes les opérations s'effectuent avec le registre du sommet de pile : goulot d'étranglement sur ce registre
  - ♦ Dec Alpha 21064 : RISC : architecture Load/Store
    - – 32 registres entiers
    - – 32 registres flottants

Augmentation de performances

- ♦ power PC 601: RISC
  - 32 registres entiers
  - 32 registres flottants

➢ la largeur registres augmente :

- ♦ Dec Alpha 21064 : 64 bits
- ♦ Power PC : 32 bits
- ♦ Pentium : 32 bits pour les registres entiers, 80 bits pour les flottants

➢ intégration de registres spécialisés : MMX

- ♦ Pentium : 8 registres 64 bits (MM0, ... MM7)
  - correspondent aux registres flottants, attention aux conflits lors du mixage d'instructions flottantes et MMX

Augmentation de performances

❑ Opérateurs et instructions particuliers

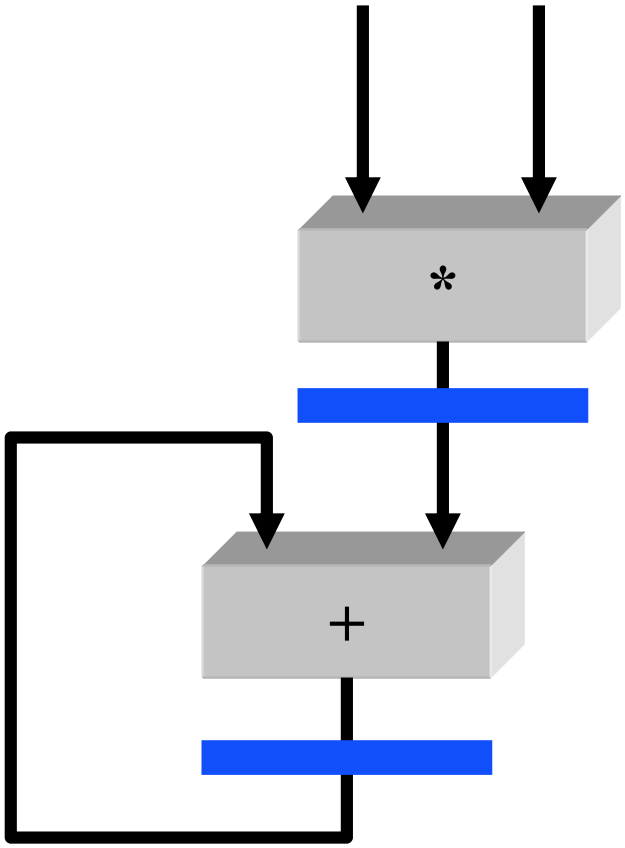
- ➢ architecture Power :
  - ♦ additionneur à 3 entrées
  - ♦ multiplicateur / diviseur
  - ♦ multiplicateur additionneur
- ➢ pentium :
  - ♦ diviseur
  - ♦ racine carrée
  - ♦ instructions transcendantales (trigonométriques, logarithmiques, hyperboliques)
- ➢ instructions multi média :
  - ♦ accélération des traitements du son et de l'image
  - ♦ nouveaux types de données : 8 octets, 4 mots, 2 doubles mots,

Augmentation de performances

➢Processeurs de traitement du signal :

♦ opération et opérateur MAC :

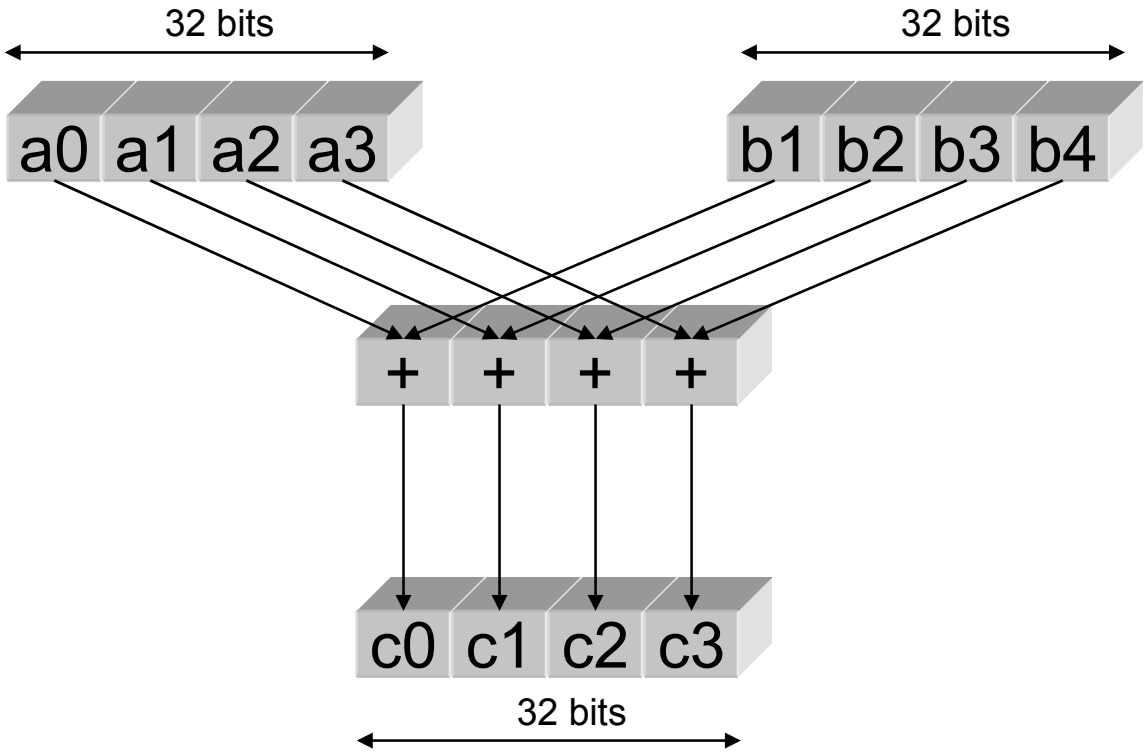
– s = a * b +c en 1 cycle



Augmentation de performances

➢Opérations SIMD ou SWP :

- ♦ SIMD : Single Instruction Multiple Data
- ♦ SWP : Sub Word Parallelism
- ♦ un opérateur * sur N bits peut faire :
  - 1 addition N bits par N bits (32 bits + 32 bits)
  - M additions sur N/M bits (4 * 8 bits + 4 * 8 bits)



Augmentation de performances

- Accélération des traitements vectoriels
- Exploitation du parallélisme de l'application

- Exemple de traitement pouvant bénéficier de ces opérations :
  - traitement d'images de bas niveau : par exemple rehaussement de contraste
  - multiplication de tous les pixels de l'image par un coefficient Beta

  - si on dispose d'un opérateur sur 32 bits pouvant réaliser des opérations sur 8 bits on peut aller 4 fois plus vite

Augmentation de performances

❑ Nombre d'adressages disponibles

➢ tendances :

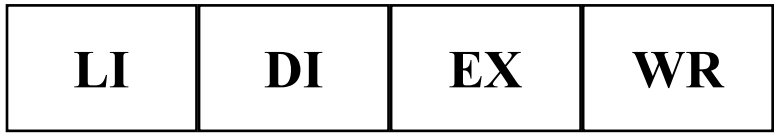
- ♦ processeurs CISC : *augmentation*
  - – beaucoup d'adressages complexes, langage machine proche des langages de programmation évolués
  - – palier le manque de registres
  - – problèmes :
    - – décodage complexe des instructions
    - – temps d'exécution des instructions variables
- ♦ processeurs RISC : *limitation*
  - – diminuer le temps de cycle, donc limiter la complexité des instructions
  - – limitation du nombre d'adressage, et donc facilite le décodage, contrôleur plus simple

Augmentation de performances

❑ Allongement des pipelines de contrôle

➢de Von Neumann sans pipeline : **LI, DI, EX, WR**

➢vers Von Neumann avec 4 étages pipelines :

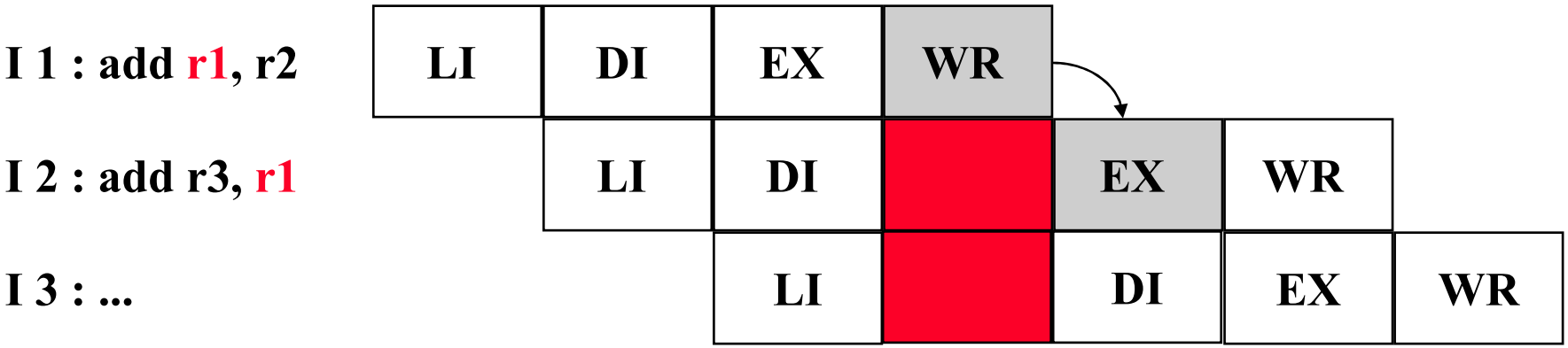


➢découpage en plusieurs sous tâches des phases :

- ♦ décodage :
  - – pré décodage pour les prédictions de branchement
  - – pré décodage instructions entière ou flottante
- ♦ exécution :
  - – les instructions flottantes sont très souvent pipelinées sur plus d'un étage

Augmentation de performances

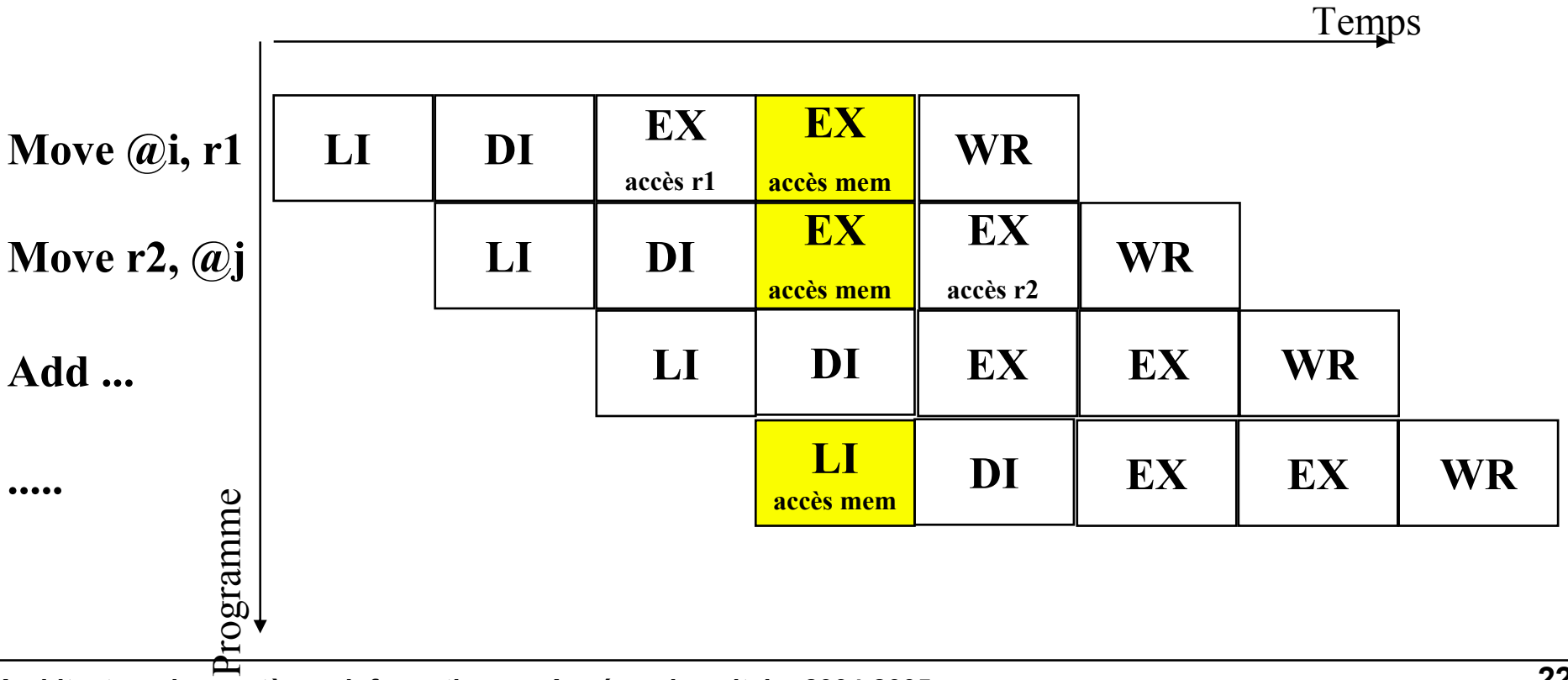
➢problèmes apportés par le pipeline de contrôle :



- ♦ dépendance : d'instructions et de données
- ♦ perte d'efficacité du pipeline pour cause de désamorçage
- ♦ plusieurs accès mémoire par cycle :
  - LI : accès IM en lecture
  - EX : accès DM en lecture ou écriture
  - WR accès DM en écriture

Augmentation de performances

- plusieurs accès mémoire par cycle (exemple) :
  - LI : accès IM en lecture
  - EX : accès DM en lecture ou écriture
  - WR accès DM en écriture



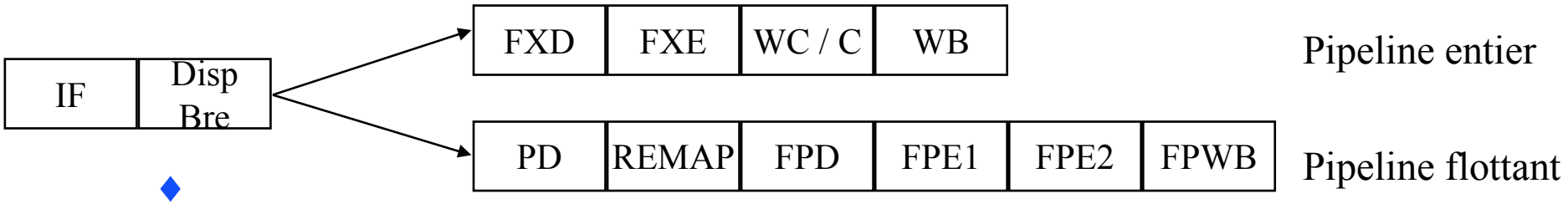
Augmentation de performances

➢ Exemples de pipelines :

♦ le power pc :

– pipeline entier sur 6 étages

– pipeline flottant sur 8 étages

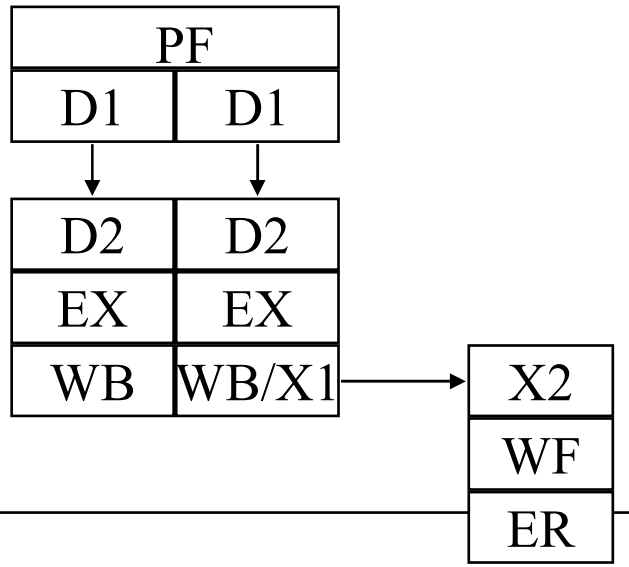


♦

♦ le pentium :

– pipeline entier sur 5 étages

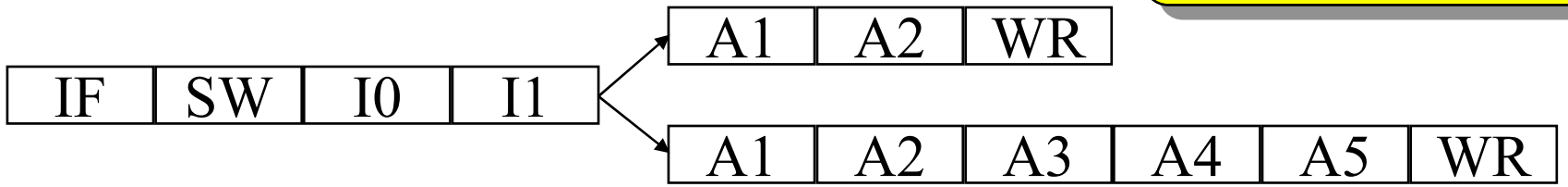
– pipeline flottant sur 7 étages



Augmentation de performances

- le Dec Alpha 21064 :
  - pipeline entier sur 7 étages
  - pipeline flottant jusqu'à 10 étages

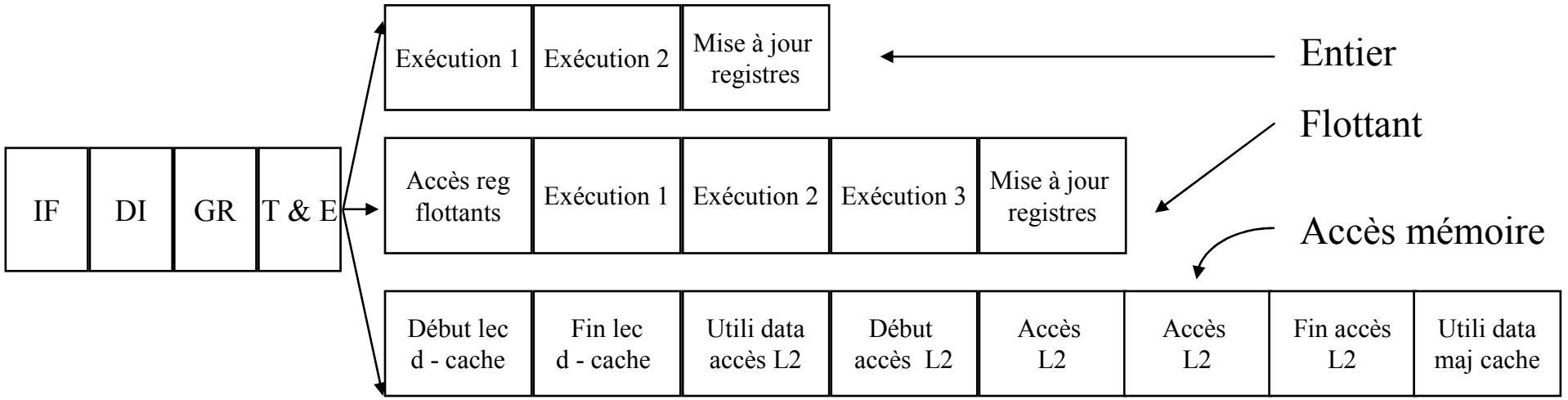
Processeurs super pipelines



- IF : instruction fetch
- SW : ??
- I0 : prédécodage
- I1 : fin décodage
- Ai : exécution
- WR : mise à jour état du processeur

Augmentation de performances

- le Dec Alpha 21164 :
  - pipeline entier sur 7 étages
  - pipeline flottant jusqu'à 9 étages



Augmentation de performances

♦ Le processeur MIPS R10000

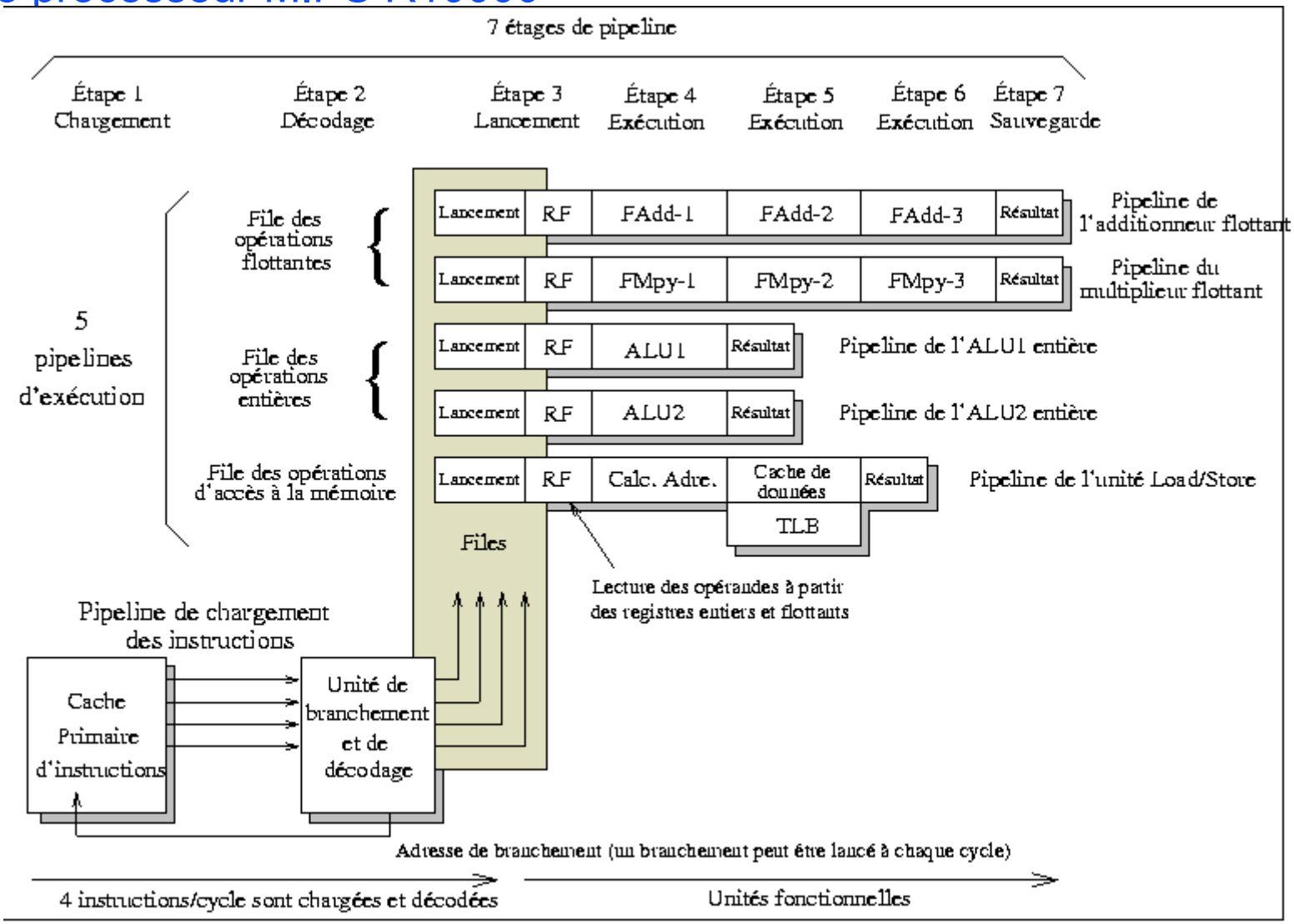


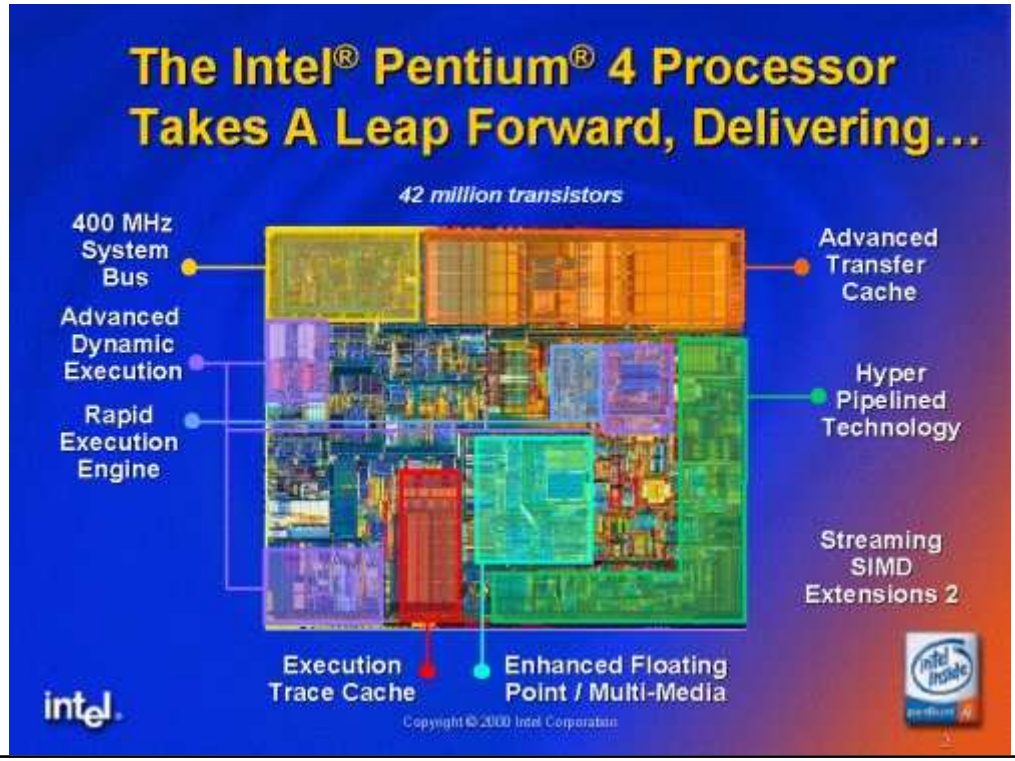
FIG. 3.1 – *Pipeline du MIPS R10000*

Augmentation de performances

♦ Le Pentium 4 :

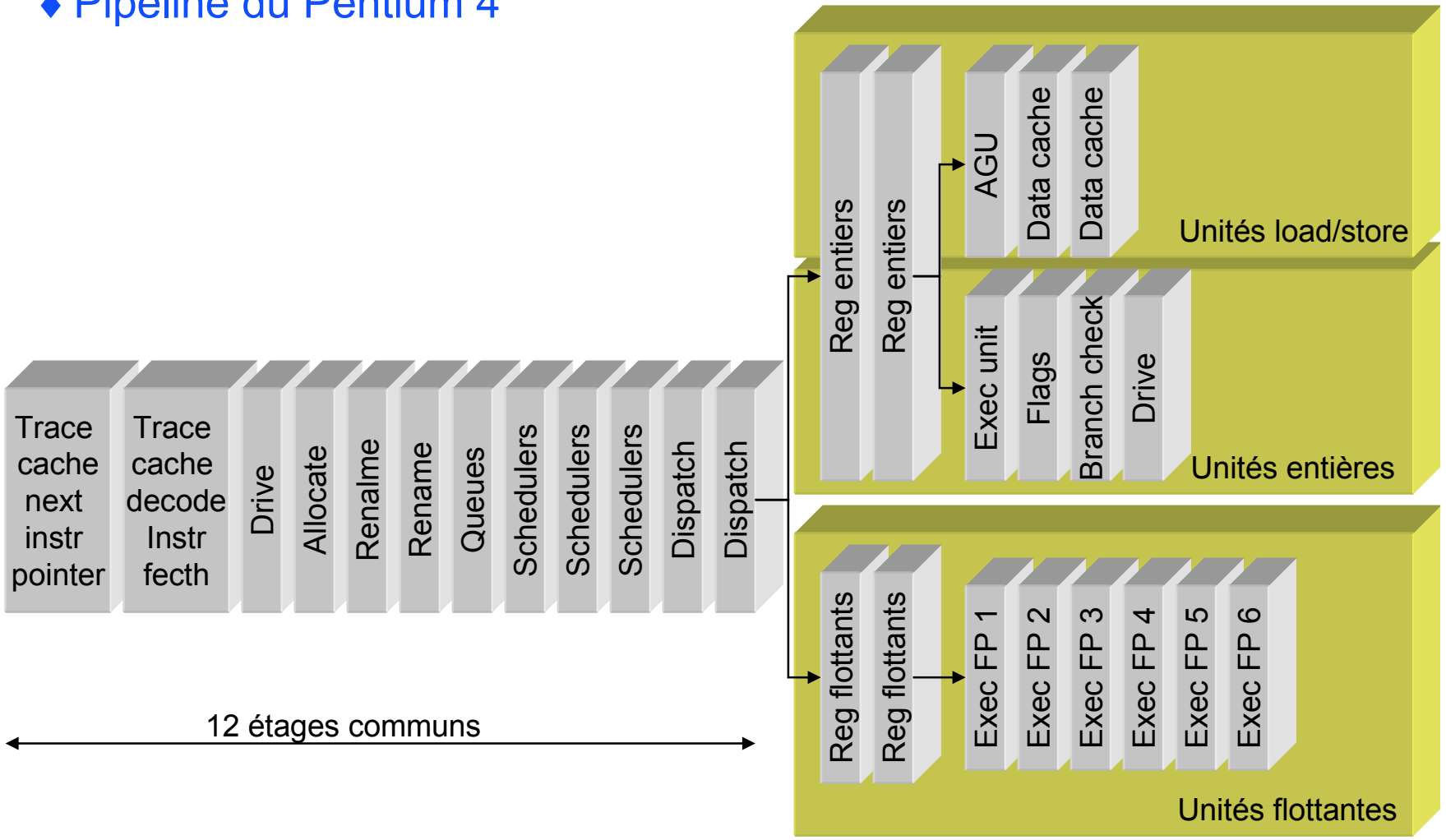
– 20 étages de pipeline :

- microarchitecture Intel NetBurst
- hyperpipeline
- l'augmentation du nombre d'étage permet d'augmenter la fréquence sans changement du procédé de gravage (0,13 micron)
- en contre partie, une mauvaise prédiction de branchement pénalise énormément le processeur : performances de la prédiction de branchement 93 % (90% pour le pentium 3)
- les UAL fonctionnent à vitesse double du processeur



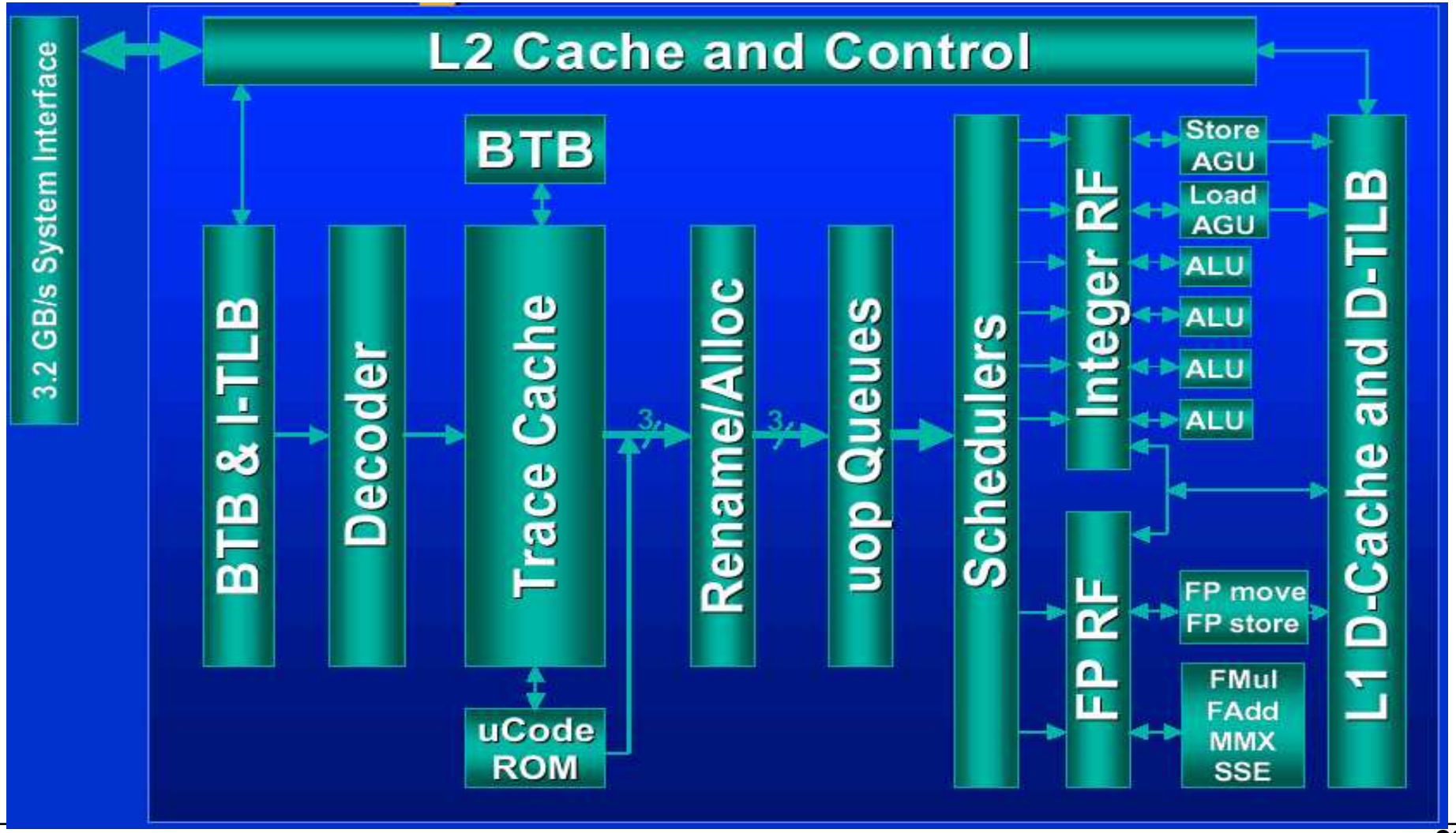
Augmentation de performances

♦ Pipeline du Pentium 4



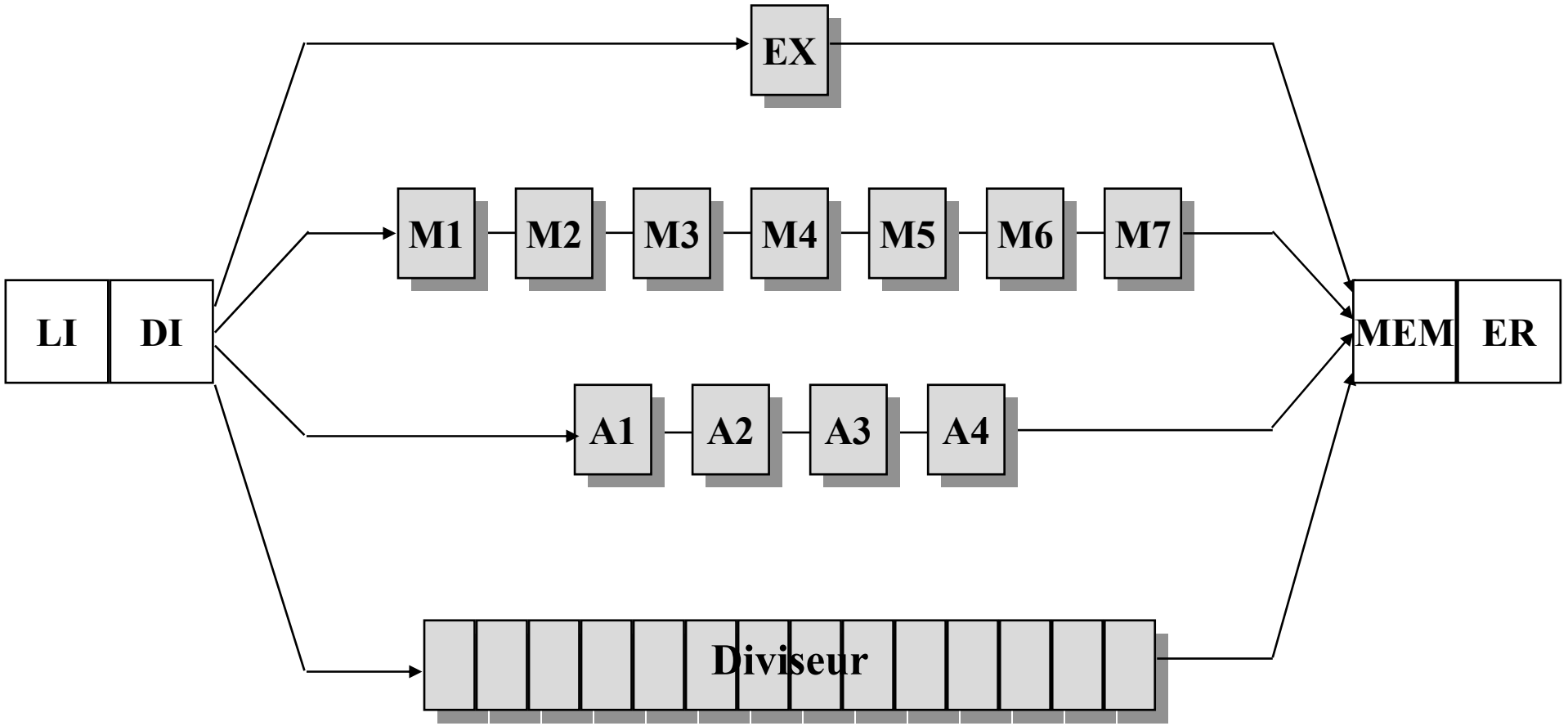
Augmentation de performances

♦ Autre présentation du pipeline du Pentium 4



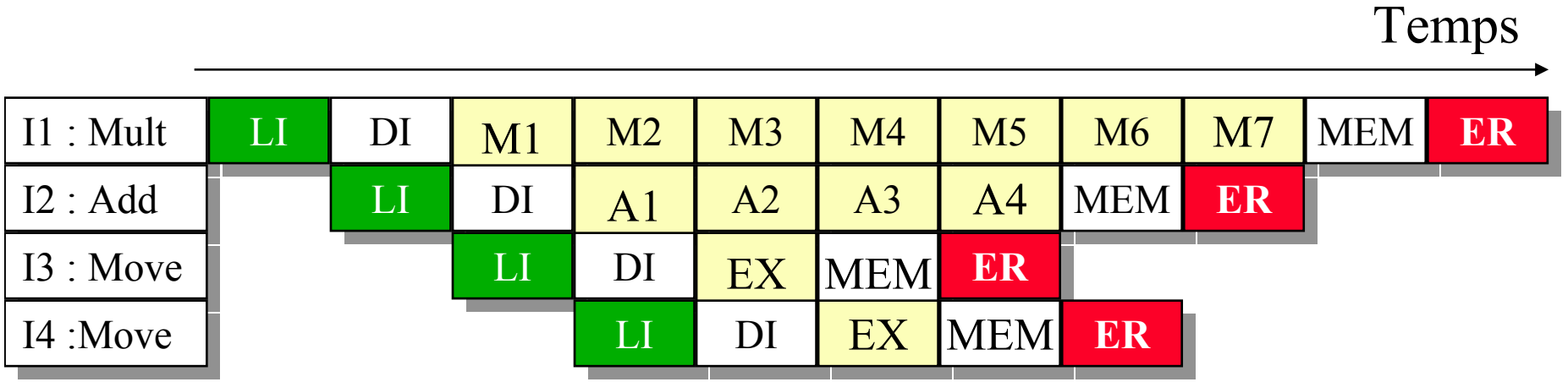
Augmentation de performances

♦ pipeline multi opérations :

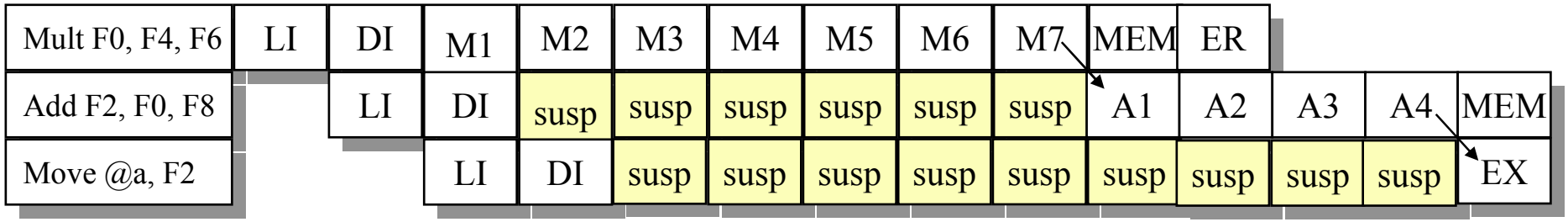


Augmentation de performances

– diagramme d'états de ce pipeline multi opérations :



– les instructions peuvent se terminer dans le désordre par rapport à leur lancement, ceci pose des problèmes avec les exceptions
– plus les pipelines sont longs, plus les pertes de cycles peuvent être importante

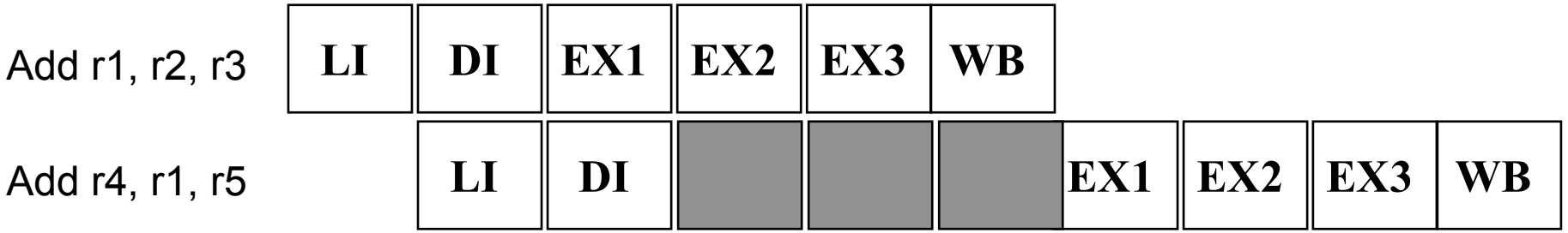


Augmentation de performances

❑ Mécanisme de bypass :

➢ objectif :

- ♦ faire passer une valeur attendue directement à une instruction sans attendre l'écriture en registre
- ♦ exemple de fonctionnement sans ByPass
  - soit le pipeline suivant :
    - lecture opérandes dans l'étage EX1
    - écriture opérande dans l'étage WB



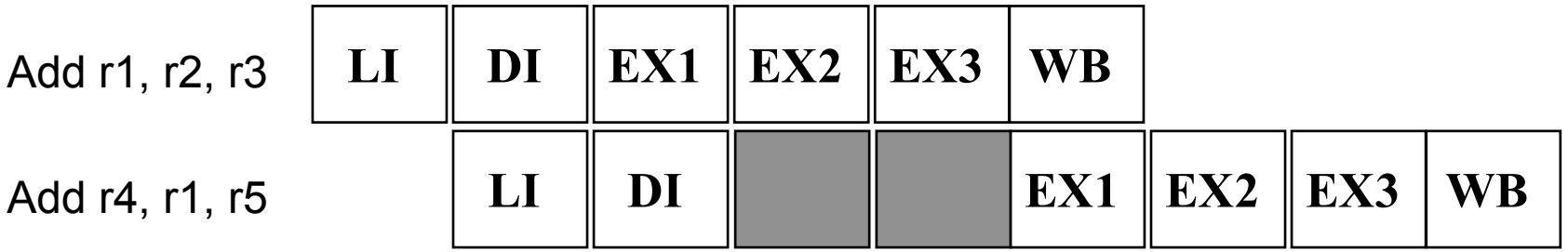
  - on perd 3 cycles

Augmentation de performances

♦ exemple de fonctionnement avec ByPass

– soit le pipeline suivant :

- Lecture opérandes dans l'étage EX1
- Écriture opérande dans l'étage WB

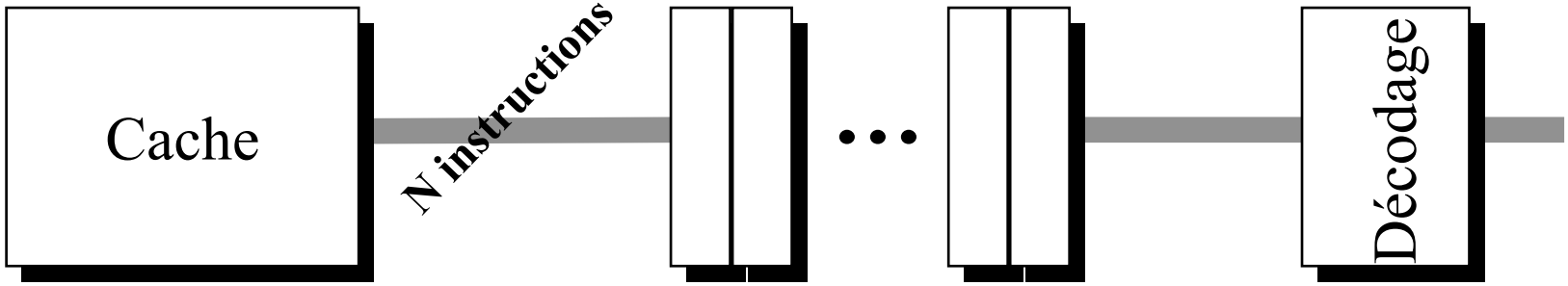


– on ne perd plus que 2 cycles

– le pipeline peut redémarrer plus rapidement

Augmentation de performances

❑ Buffer de préchargement des instructions

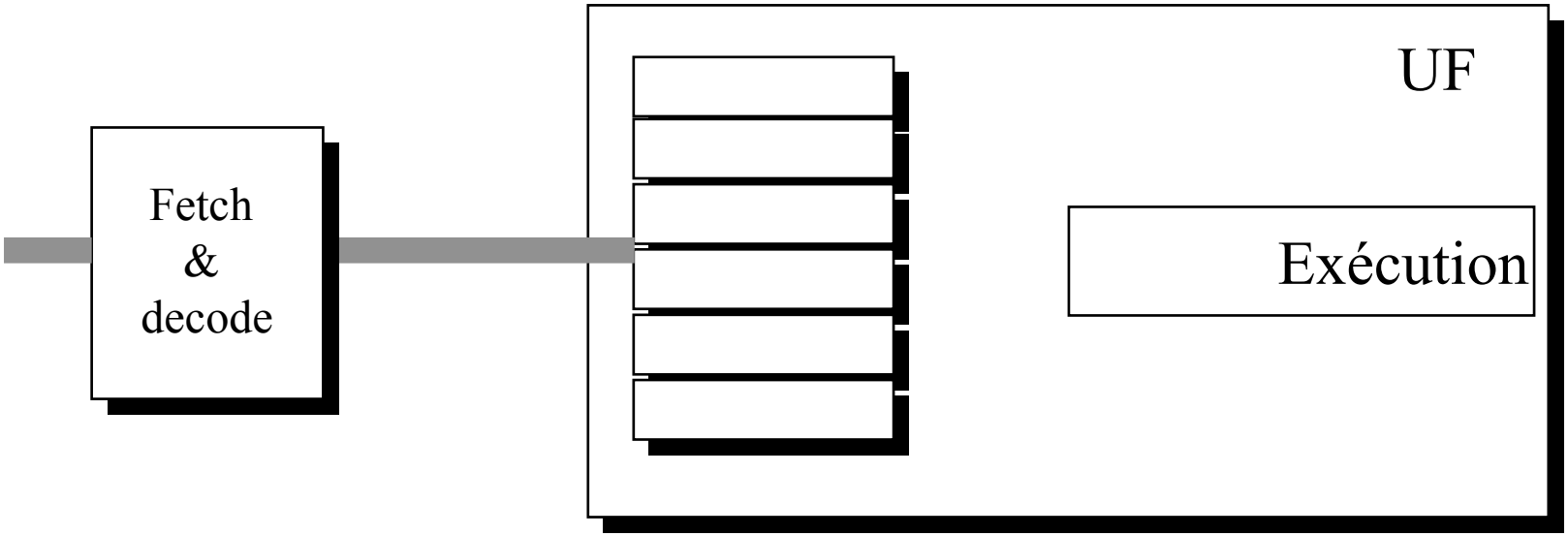


- ➢ permet de stocker un nombre important d'instructions
- ➢ nécessite souvent une augmentation de la largeur du bus entre le cache d'instructions et le buffer
- ➢ favorise le pré décodage des instructions (de branchement notamment)
- ➢ favorise l'exécution dans le désordre des instructions :
  - ♦ le processeur peut piocher dans le buffer d'instructions pour exécuter l'instruction i+1 avant l'instruction i
- ➢ permet d'être toujours en mesure de fournir des instructions au cœur du processeur qui est de plus en plus superscalaire

Augmentation de performances

➢Buffer d'instructions :

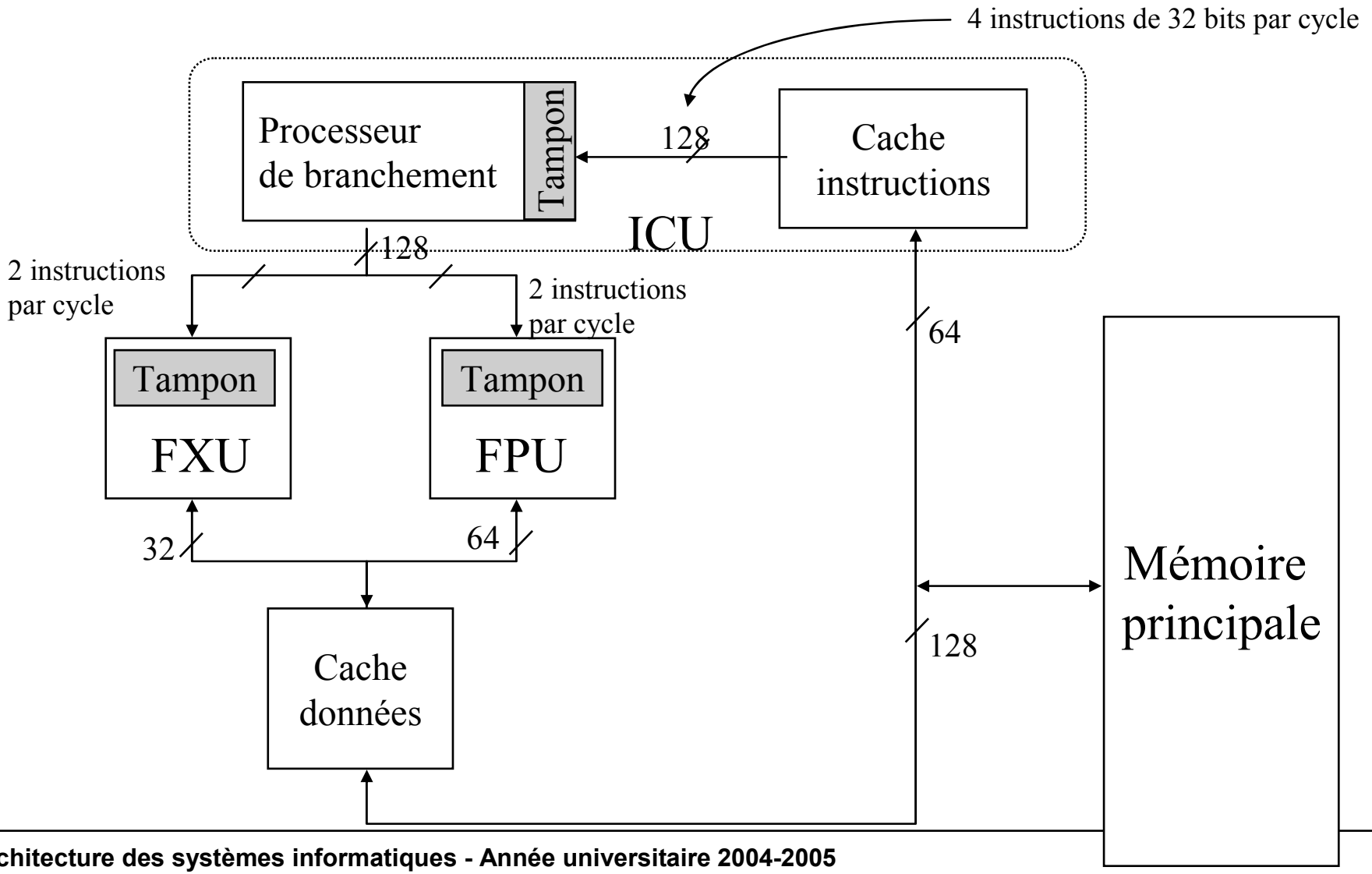
- ♦ découplage entre les étages de lecture / décodage et les étages d'exécution



- ♦ Etage fetch & decode : produit des instructions dans le buffer
- ♦ Etage exécution : consomme des instructions dans le buffer

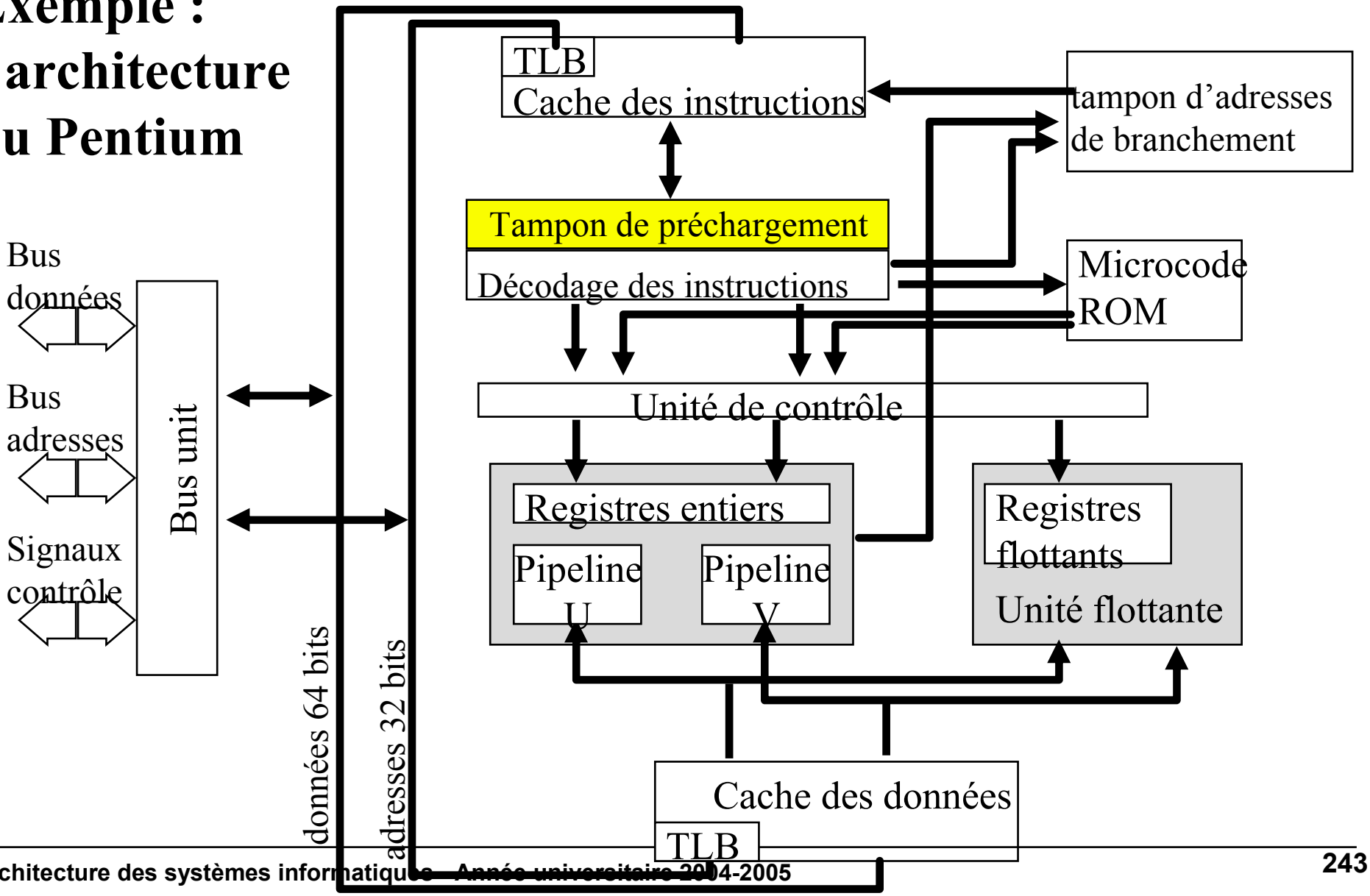
Augmentation de performances

Exemple : architecture Power PC



Augmentation de performances

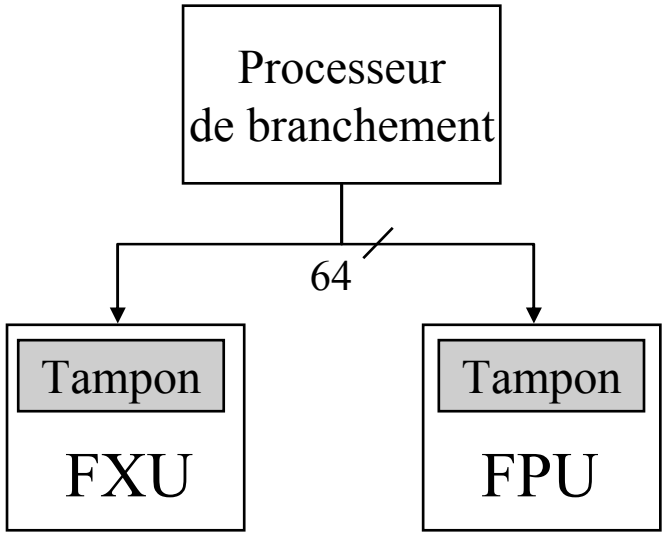
Exemple : l'architecture du Pentium



Augmentation de performances

➢ tamponnage des instructions au niveau de chaque unité :

- ♦ permet d'assurer une continuité dans l'alimentation du pipeline d'une unité même en l'absence d'instruction pour cette unité
- ♦ dans le Power PC : les instructions s'exécutent dans l'ordre

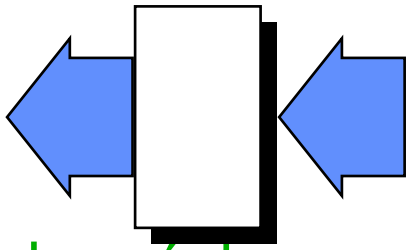


Augmentation de performances

❑ Module de dispatching

➢gestion de la cohérence entre les différentes unités et les différents types d'instructions :

- ♦ unités entières
- ♦ unité flottante
- ♦ unité de branchement



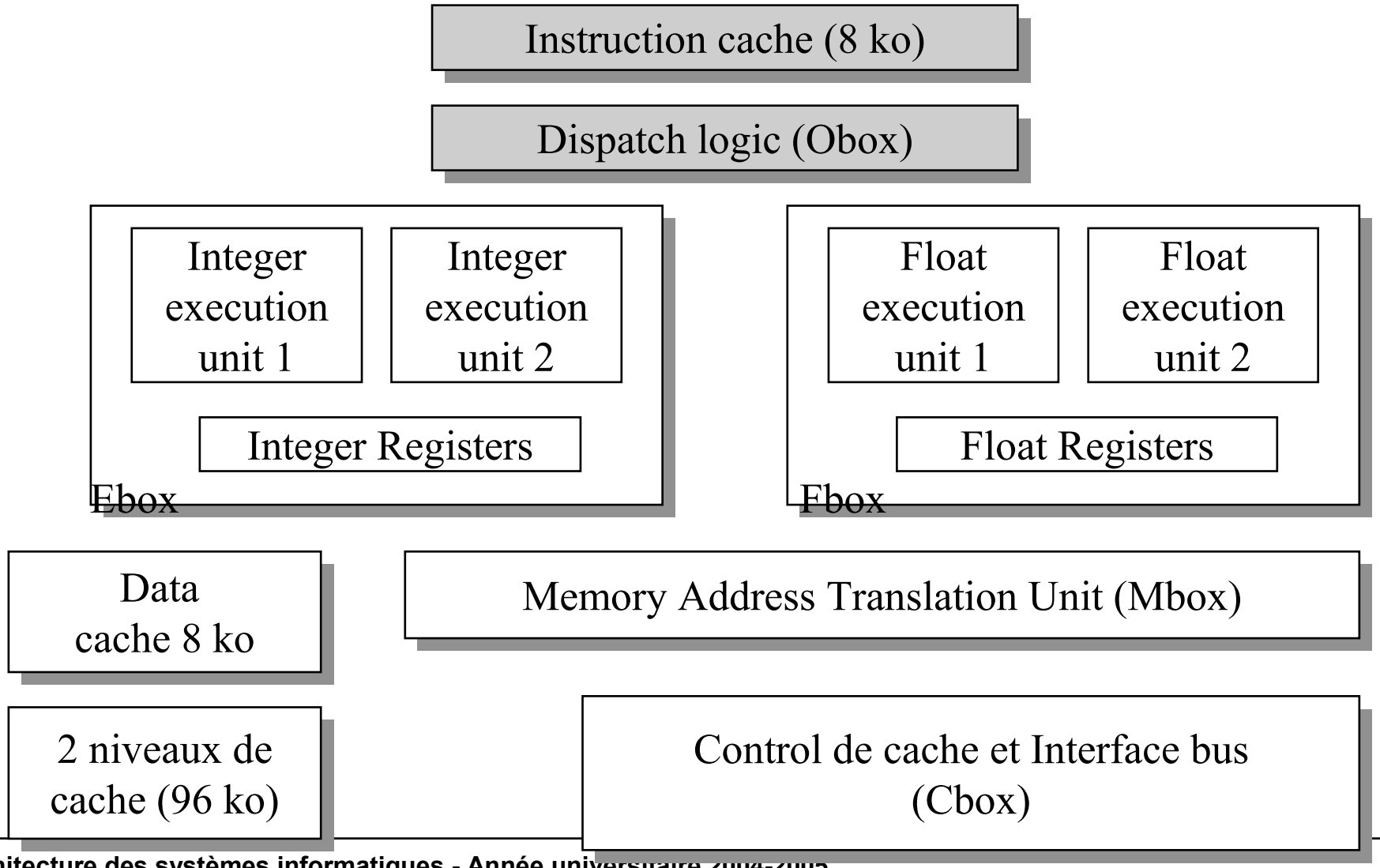
- instructions de branchement
- instructions flottantes
- instructions entières

➢puise dans le buffer de préchargement, pré décodage de l'instruction, et envoie vers le bon type d'unité

➢exemple : le power pc :

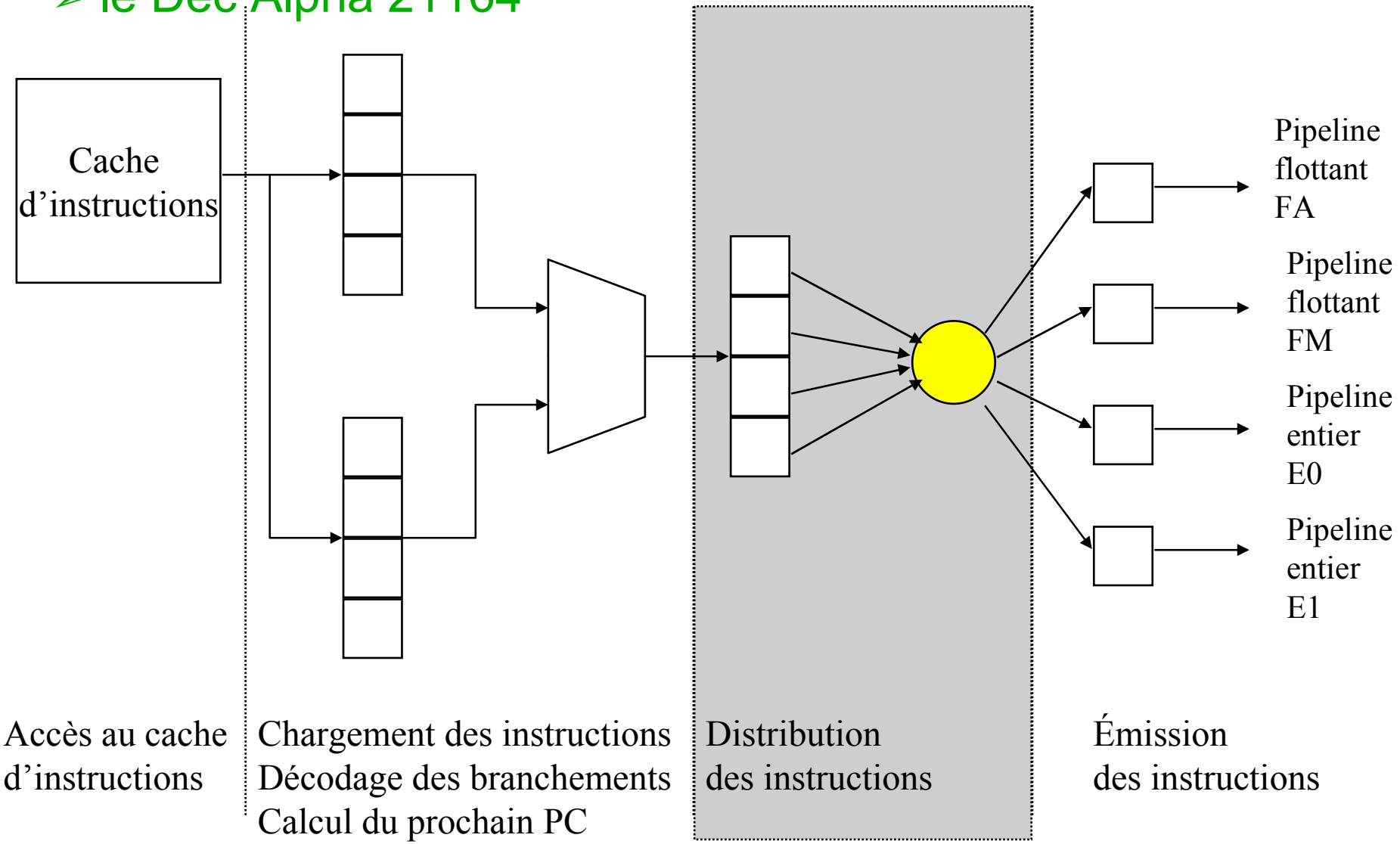
Augmentation de performances

➢exemple : le Dec Alpha 21164:



Augmentation de performances

➢le Dec Alpha 21164



Augmentation de performances

❑ Predicated instructions :

- ♦ instructions conditionnées à la valeur d'un registre prédicat
- ♦ permet de supprimer des branchements conditionnels
- ♦ instructions classiques + :
  - – un registre valant vraie ou faux
  - – si le registre contient vraie alors l'instruction est exécutée
  - – sinon l'instruction se comporte comme un NOP

```
If (x==0) {
        a = b +c;
        d = e - f;
}
g = h *i;
```

```
(Pred = (x == 0))
if (Pred) then a = b + c;
if (Pred) then d = e - f;
g = h * i;
```

- ♦ complique l'exécution des instructions
- ♦ nécessite l'ajout de ports sur la file de registres
- ♦ les instructions qui se comporteront comme des NOP consomment des ressources processeur
- ♦ intéressant pour les petites structures conditionnelles if then else

Augmentation de performances

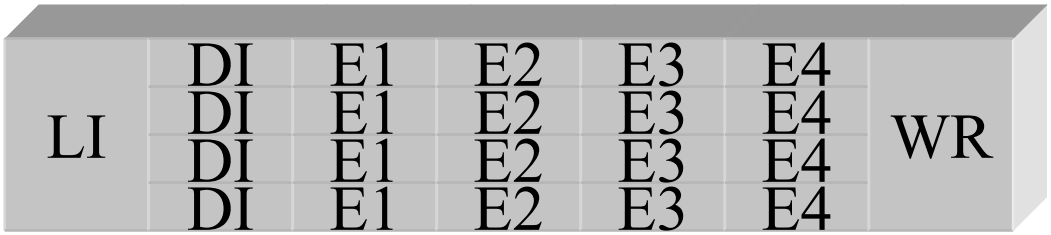
❑ Parallélisme dans les processeurs :

- ➢ plusieurs modèles :
  - ♦ pipeline :
    - – il s'agit d'un traitement à la chaîne
    - – plusieurs instructions sont exécutées en même temps mais elles sont à des niveaux de traitement différents
  - ♦ VLIW :
    - – plusieurs instructions sont exécutées en parallèle sur des unités différentes
    - – l'ordonnancement des instructions est déterminés à la compilation, statique
  - ♦ superscalaire :
    - – plusieurs instructions sont exécutées en parallèle sur des unités différentes
    - – l'ordonnancement est déterminé à l'exécution, dynamique

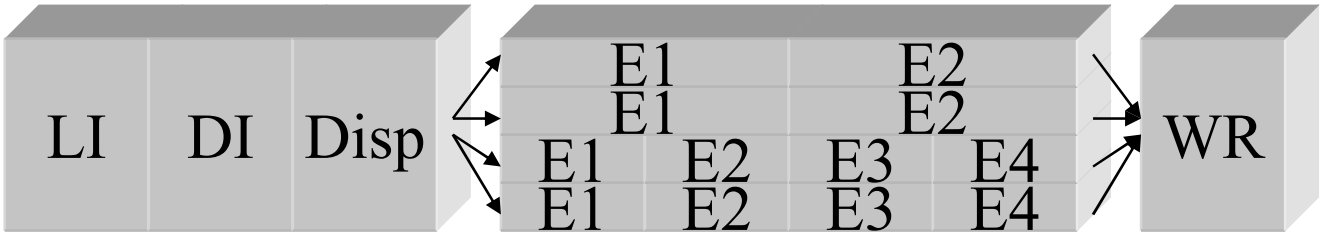
Augmentation de performances

➢pipeline

➢VLIW



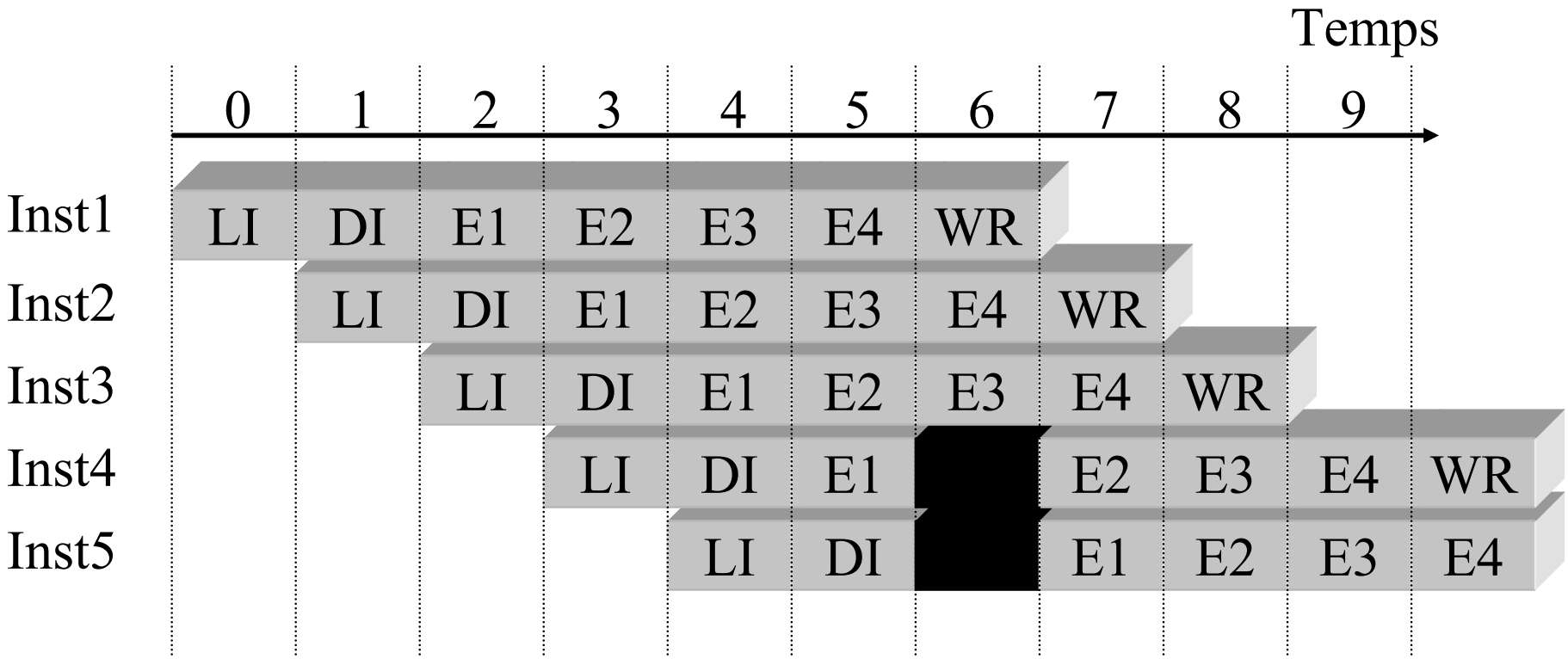
➢superscalaire



Augmentation de performances

➢Fonctionnement :

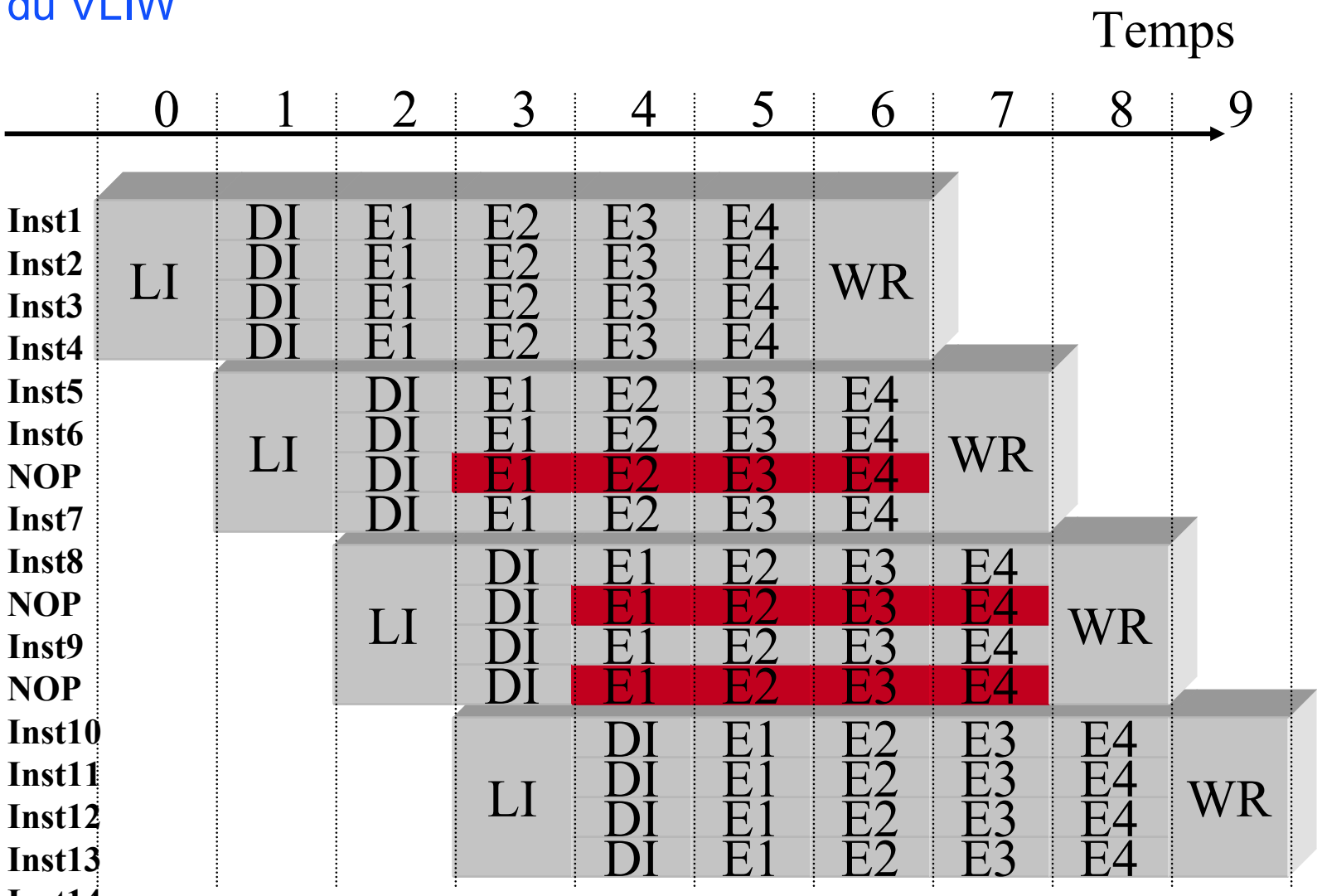
♦ du pipeline :



Augmentation de performances

➢Fonctionnement :

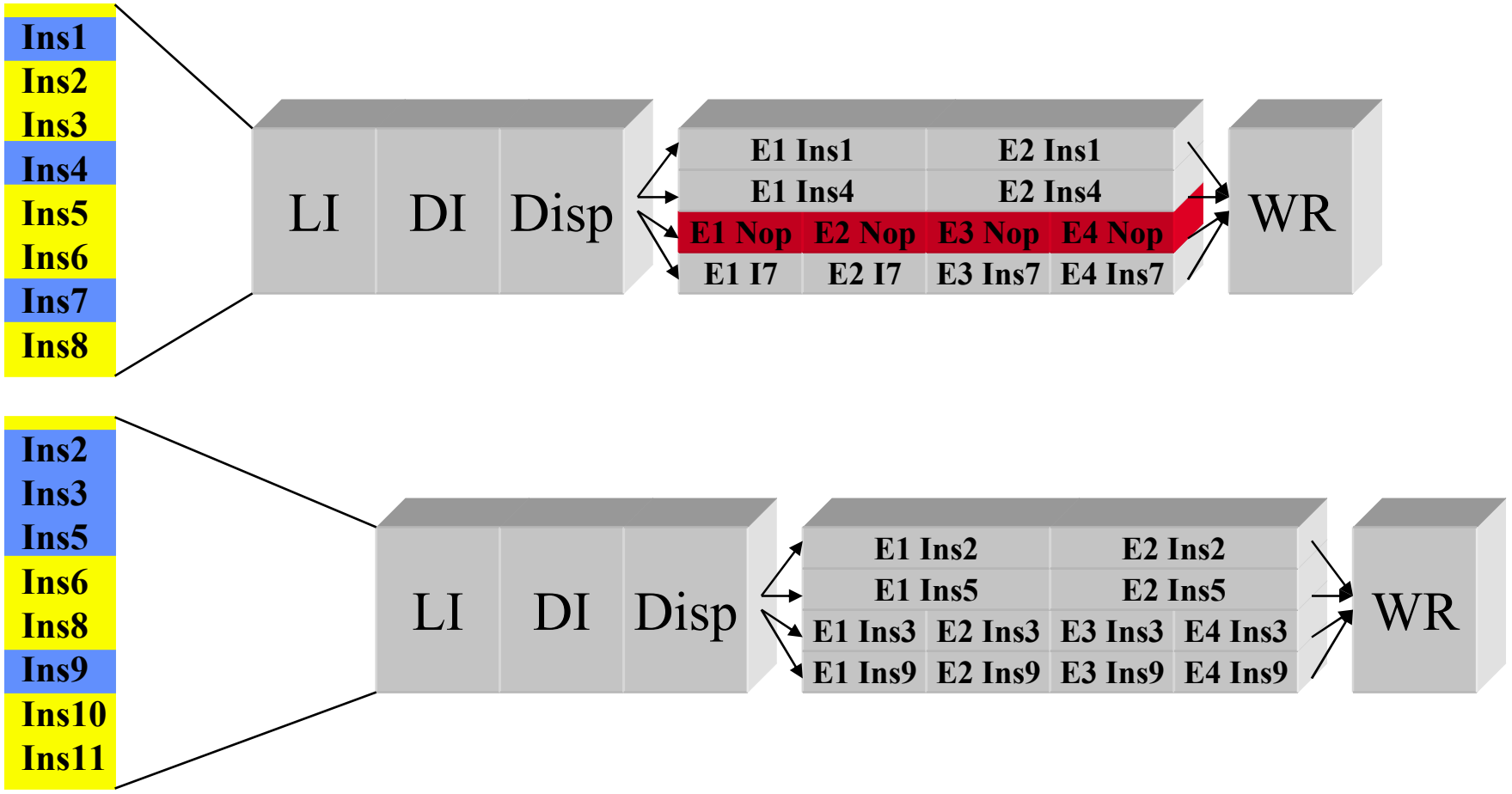
♦ du VLIW



Augmentation de performances

➢Fonctionnement :

♦ du superscalaire



Augmentation de performances

❑ Processeurs superscalaire

➢mise en place d'unités fonctionnelles indépendantes :

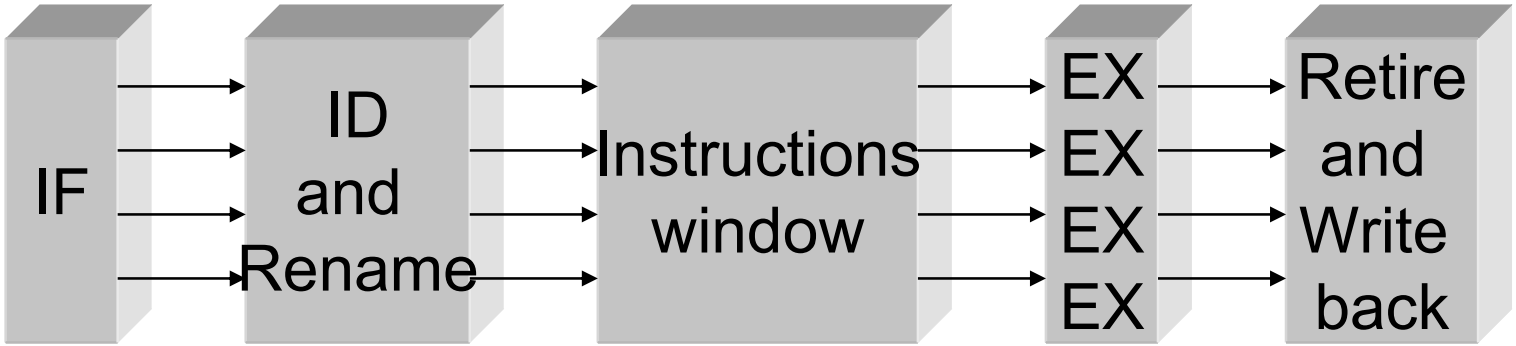
- ♦ fonctionnement parallèle des unités :
  - donc les instructions s 'exécutant en parallèles ne doivent pas avoir de dépendance (données, instructions)
- ♦ unités équivalentes ou non :
  - Pentium : 2 pipelines U et V, ne sont pas équivalentes, c'est le pipeline U qui réalise les opérations flottantes (en plus des instructions entières)
  - Dec Alpha 21164 :
    - 2 unités entières équivalentes
    - 2 unités flottantes équivalentes
- ♦ unités fortement pipelinées :
  - Pentium : 7 étages
  - Power pc : 8 étages
  - Dec alpha : 10 étages

Augmentation de performances

- Associé à l'exécution dans le désordre
  - pour une utilisation optimale des pipelines l'ordonnancement des instructions est déterminé dynamiquement
  - logique de contrôle lourde
  - compilation simplifiée, toute (ou partie de) la complexité est rejetée sur le matériel
  - nécessite une gestion rigoureuse des ressources :
    - quelles instructions sont actives ?
    - quels registres vont être modifies par quelles instructions ?
    - quelles instructions peut être lancée à un temps de cycle précis ?
    - (fonctionnement détaillé par le tableau des marques, voir un peu plus loin)

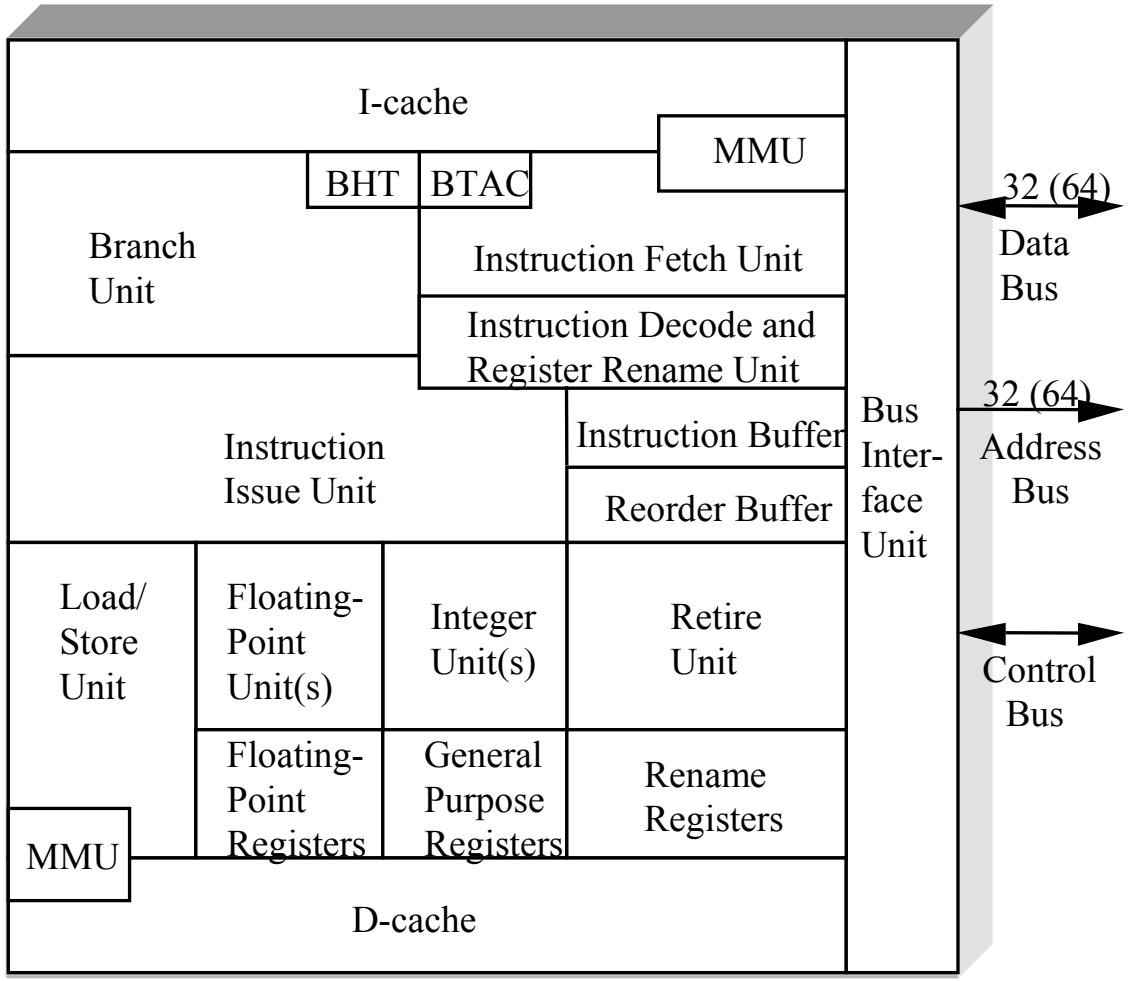
Augmentation de performances

➢Pipeline classique pour un superscalaire



Augmentation de performances

➢Architecture générale d'un processeur superscalaire



Augmentation de performances

➢Exemple du Pentium 4

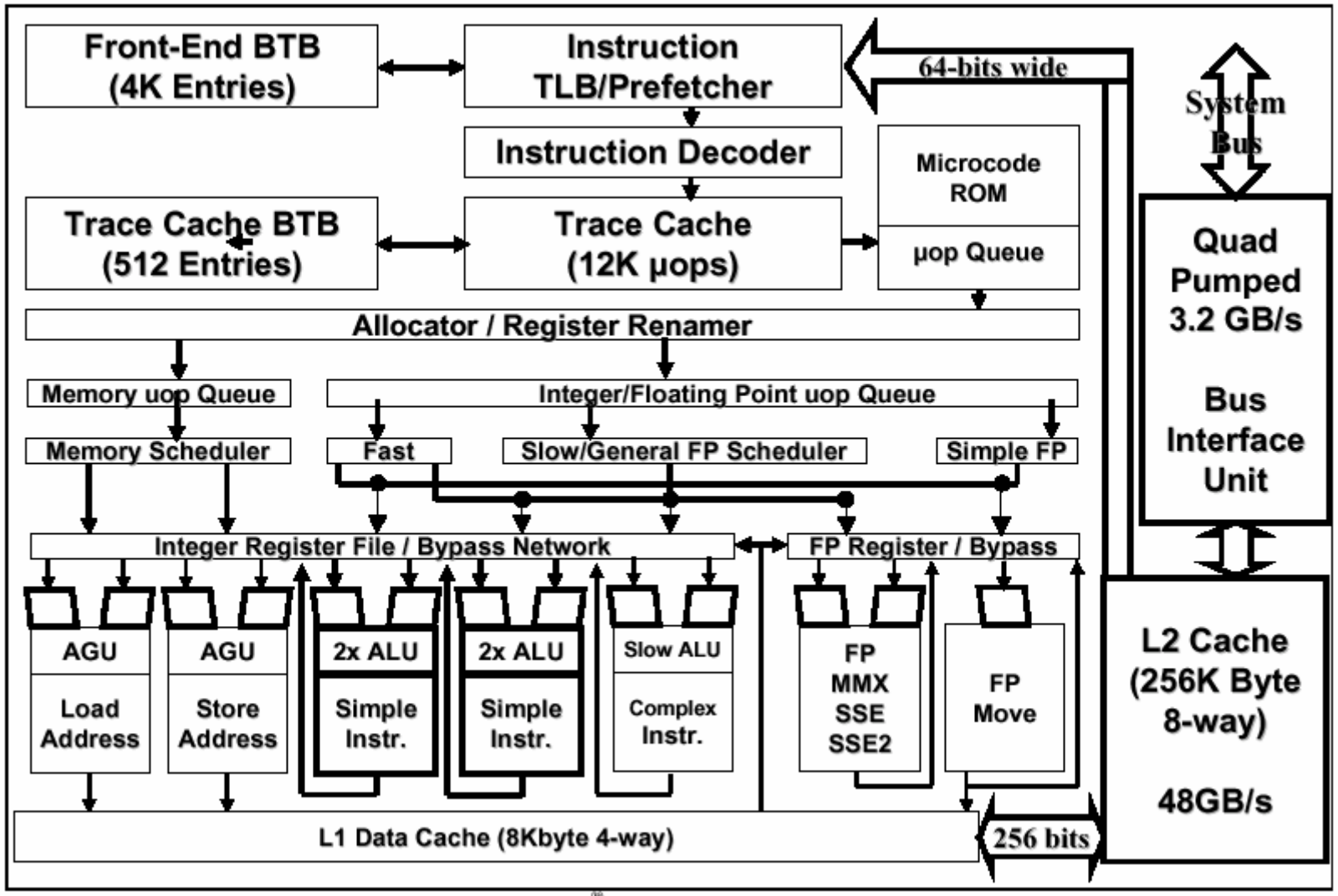


Figure 4: Pentium® 4 processor microarchitecture

Augmentation de performances

➢ Exemple du Pentium 4, suite :

♦ 7 unités indépendantes

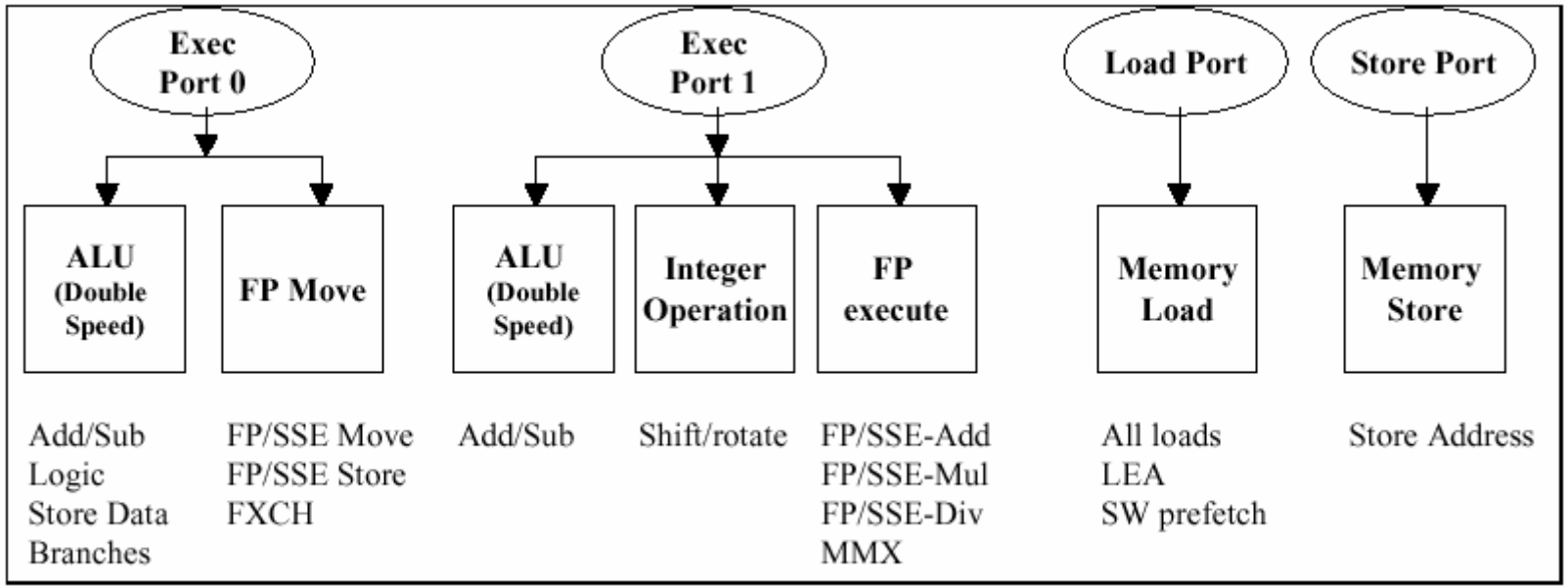


Figure 6: Dispatch ports in the Pentium® 4 processor

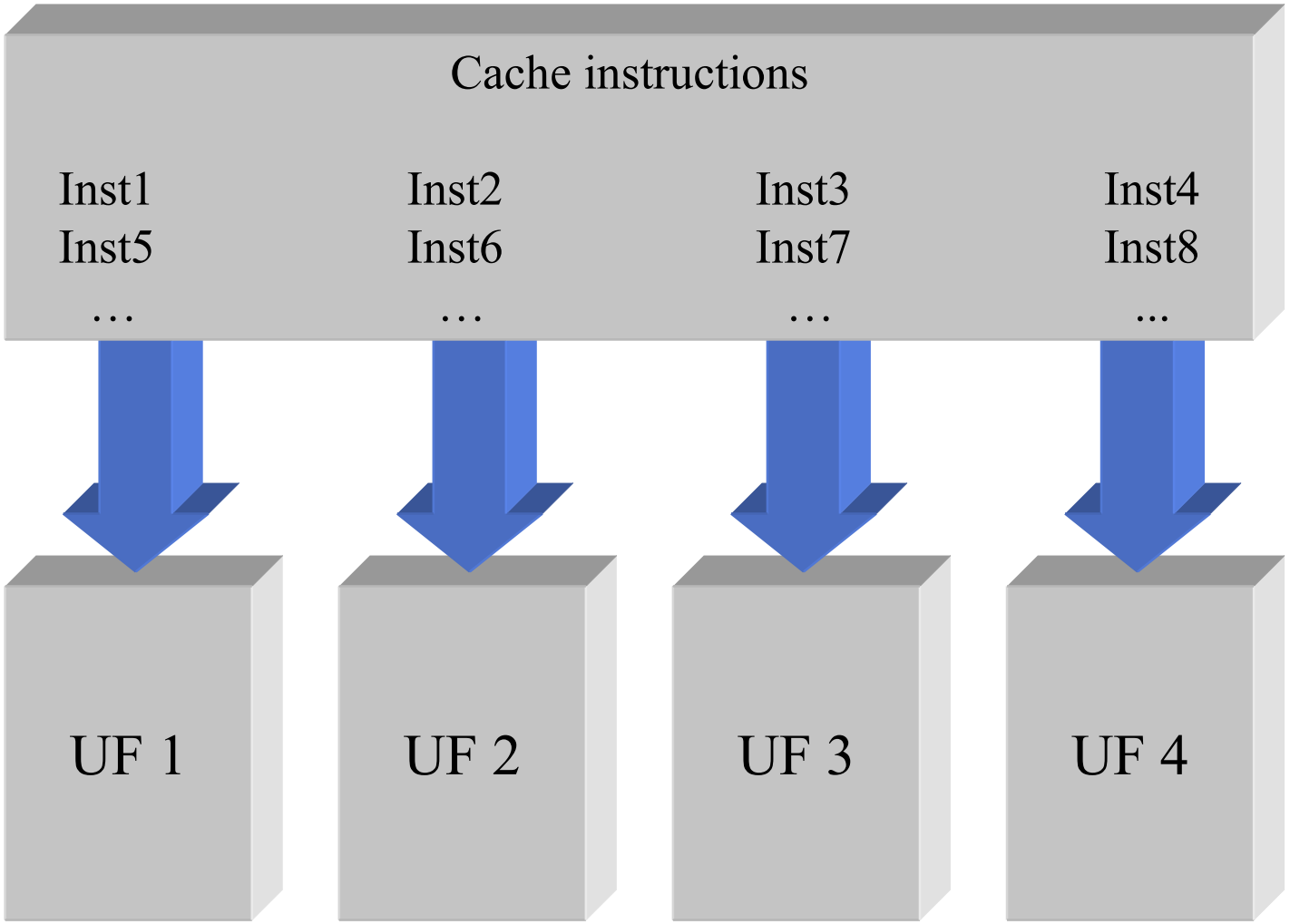
Augmentation de performances

❑ Processeurs VLIW

- ➢ pas d'exécution dans le désordre
- ➢ ordre des instructions déterminé à la compilation
- ➢ toute la complexité est rejetée sur le compilateur :
  - ♦ extraction du parallélisme de l'application
  - ♦ exploitation de celui-ci sur l'architecture
- ➢ a chaque cycle :
  - ♦ lecture d'une seule instruction longue :
    - – Very Long Instruction Word (exemple une instruction de 128 bits)
  - ♦ une instruction longue est composée de :
    - – N instructions élémentaires (exemple 4 instructions de 32 bits)
  - ♦ la position des instructions élémentaires définies l'unité fonctionnelle qui exécutera l'instruction

Augmentation de performances

➢Architecture VLIW typique

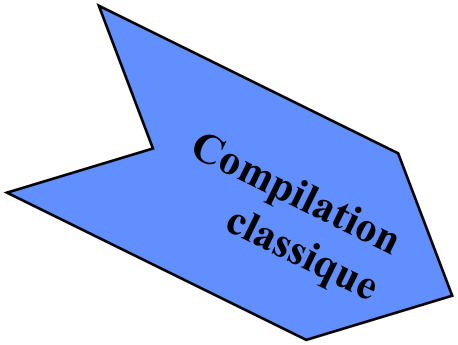


Augmentation de performances

➢ Exemple de code compilé :

♦ soit le programme réalisant le calcul du produit scalaire entre 2 vecteurs U et V :

```
P = 0.0;
for (i=0 ; i < Taille ; i++) {
        P = P + U[i] * V[i];
}
```

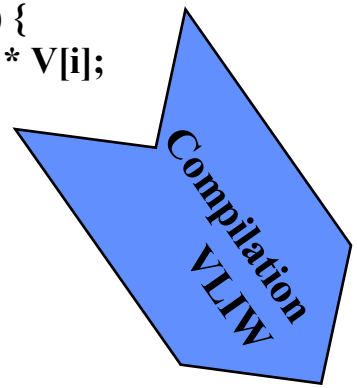



```
Debut
        Move    R0, @U
        Move    R1, @V
        Move    R2, @P
        Move    R3, Taille
        Move    F0, 0
Boucle
        Move    F1, (R0)
        Move    F2, (R1)
        Multf   F1, F2
        Addf    F0, F1
        Add     R0, 1
        Add     R1, 1
        Sub     R3, 1
        Bnez    Boucle
        Move    (R2), F0
```

Augmentation de performances

➢Exemple de code compilé (suite) :

```
P = 0.0;
for (i=0 ; i < Taille ; i++) {
        P = P + U[i] * V[i];
}
```



Debut	Move R0, @U	Move R1, @V	Move F0, 0	NOP
	Move R2, @P	Move R3, Taille	NOP	NOP
Boucle	NOP	NOP	Move F1, (R0)	Move F2, (R1)
	Add R0, 1	AddR1, 1	Multf F1, F2	NOP
	Sub R3, 1	NOP	Addf F0, F1	NOP
	Bnez Boucle	NOP	NOP	NOP
	NOP	NOP	Move (R2), F0	NOP

Augmentation de performances

➢ Exemple de code compilé (suite) :

- ♦ taille du code :
  - – compilation classique sur processeur RISC :
    - 14 instructions de format 32 bits = 448 bits
  - – compilation VLIW :
    - 7 instructions longues de format 4 * 32 bits = 896 bits

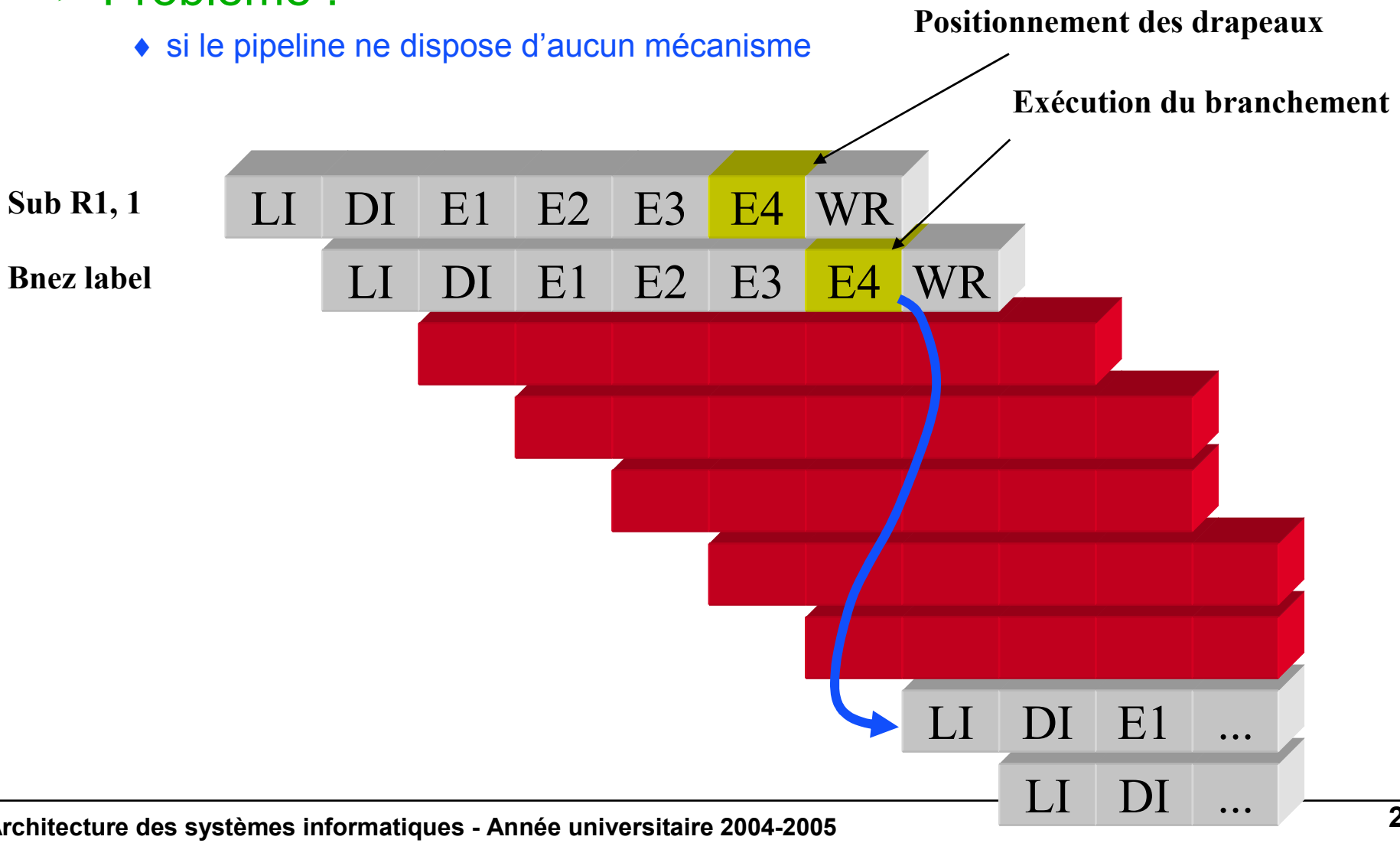
- ♦ pour cet exemple le code VLIW est 2 fois plus encombrant que le code RISC classique !!!
  - – problématique notamment pour les systèmes embarqués :
    - l 'espace mémoire peut être compté
    - la consommation doit être réduite
  - – solution :
    - taille de l'instruction longue variable, notion de bundle :
    - solution mise en œuvre dans l 'IA 64 (Itanium), Processeur Lx ST

Augmentation de performances

❑ Prédiction de branchement

➢Problème :

♦ si le pipeline ne dispose d'aucun mécanisme



Augmentation de performances

➢Que faire ?

♦ limiter le nombre de cycles perdus dans les branchements, 3 solutions :

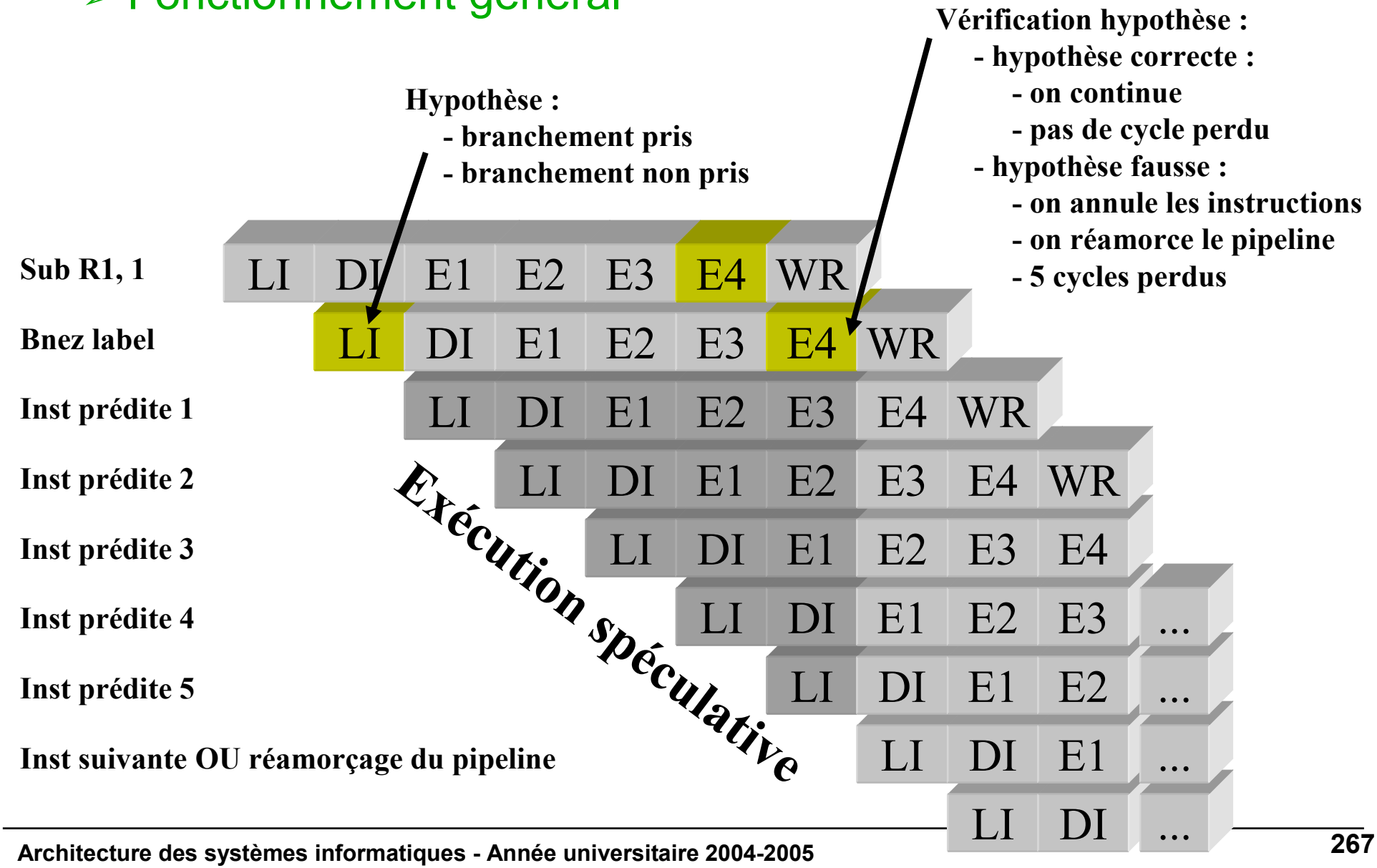
– limiter la longueur des pipelines :
  - c'est en contradiction avec l'augmentation de performance liée au travail à la chaîne

– limiter le nombre d'instructions de branchement :
  - certains processeurs proposent des instructions du type CMOVE

– tenter de prédire le comportement du branchement :
  - si la prédiction est correcte alors pas de perte de cycles

Augmentation de performances

➢Fonctionnement général



Augmentation de performances

♦ dans le cas ou l'hypothèse est fausse :

– les 5 instructions qui suivent le branchement doivent être annulées :

- on indique à l ’étage WR de ne pas réaliser les écritures (en registres ou en mémoire) et ce pendant 5 cycles
- les 5 instructions ont donc un comportement équivalent à des NOP
- on a perdu 5 cycles
- le processeur réamorce le pipeline avec l ’instruction située à l ’adresse label

♦ dans le cas ou l'hypothèse est correcte :

– les 5 instructions qui suivent le branchement poursuivent leur exécution

– pas de cycle perdu

Augmentation de performances

➢Coût d'une mauvaise prédiction :

- sur un Alpha 21264 :
  - 4 à 9 cycles
- sur un Pentium II
  - 11 cycles

Augmentation de performances

➢ Technique pour établir la prédiction (hypothèse) :

- ♦ doit se tromper le moins souvent possible
- ♦ n'influence pas le nombre de cycles perdus lors du mauvaise prédiction
- ♦ type de prédictions :
  - – prédiction statique :
    - prédiction globale au processeur quelque soit le branchement :
    - par exemple : branchements toujours prédit pris, ou branchements toujours prédits non pris
  - – prédiction statique :
    - parfois le compilateur a la possibilité de positionner un bit dans l'instruction pour indiquer au processeur comment le branchement doit être prédit
  - – prédiction statique dépendant du sens du branchement :
    - prédiction globale au processeur
    - si le branchement est arrière alors prédiction statique 1 (par exemple : prédit pris)
    - si le branchement est avant alors prédiction statique 2 (par exemple : prédit non pris)
  - – prédiction dynamique :
    - mémorisation du comportement de chaque branchement (historique)
    - décision par rapport aux comportements précédents

Augmentation de performances

➢prédiction statique simple

- utilise la technologie des compilateurs
- examen des exécutions précédentes du programme :
  - on observe les branchements : (voir figure), on en conclue que la plupart des branchements sont pris, la manière la plus simple de prédire le branchement est alors de dire que tout branchement sera pris

➢prédiction statique tenant compte du sens du branchement

♦ le comportement des branchements dépend souvent de leur sens :

- on appelle branchement arrière un branchement du type :
  - @i BR condition, @j avec @ j < @i
- on appelle branchement avant un branchement du type :
  - @i BR condition, @j avec @ j > @i

Augmentation de performances

♦ Les branchements arrières :

– sont souvent issues de la compilation d 'une boucle :

```
For (i = 0 ; i < N ; i++) {
        cœur de la boucle ;
}
```

```
        Move R1, N
Boucle
        Bloc d 'instructions
        Sub R1, 1
        Brnez Boucle
        …
```

– comportement d 'une boucle facilement prédictible :

- le branchement est pris N fois
- et non pris 1 fois (à la sortie de la boucle)

♦ Prédiction de branchement du processeur :

– les branchements **arrières** seront toujours **prédits pris**

Augmentation de performances

♦ Les branchements avants :

– ils sont issues (entre autres) de structures conditionnelles

```
If condition then
        Bloc instructions 1 ;
else
        Bloc instructions 2 ;
end if;
```

```
        Br condition, else
        Bloc instructions 1
        JUMP suite
else
        Bloc instructions 2
suite
```

– comportement plus délicat à prédire

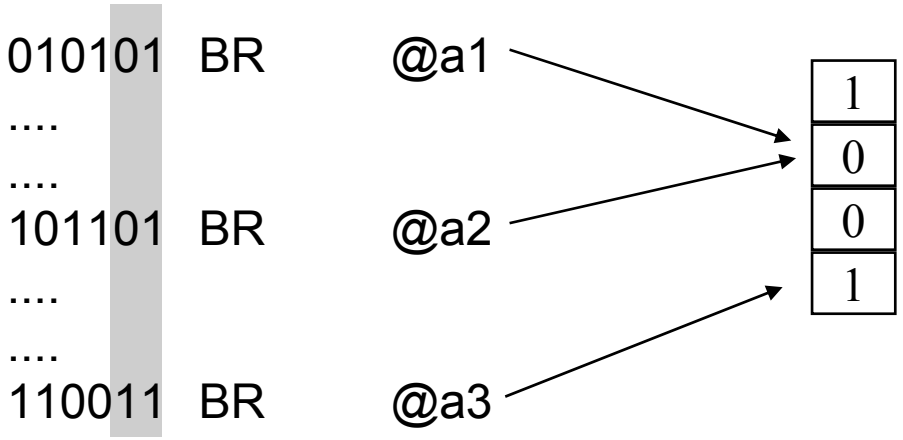
♦ Prédiction de branchement du processeur :

– les branchements **avants** seront toujours **prédits non pris**

Augmentation de performances

➢prédiction de branchement dynamique à 1 bit

- dépend du comportement des branchements au cours de l'exécution
- on conserve un historique du comportement des branchements
- la prédiction change si le comportement du branchement change
- gestion par une table d'historique des branchements :



Augmentation de performances

– efficacité de cette prédiction, soit le code suivant :

```
@a1     i = 0
@a2     ...
...     ...
@an     i = i + 1
@an+1   si i < N alors BR @a2
```

– au premier passage dans la boucle, 2 cas peuvent se produire :
  – le branchement est prédit pris, et il est effectivement pris
  – le branchement est prédit non pris, et il est pris

RUPTURE PIPELINE

– à la dernière itération de la boucle, le bit de prédiction est à 1 (indique que le branchement doit être pris), le branchement est donc prédit pris, mais il ne sera pas pris

RUPTURE PIPELINE

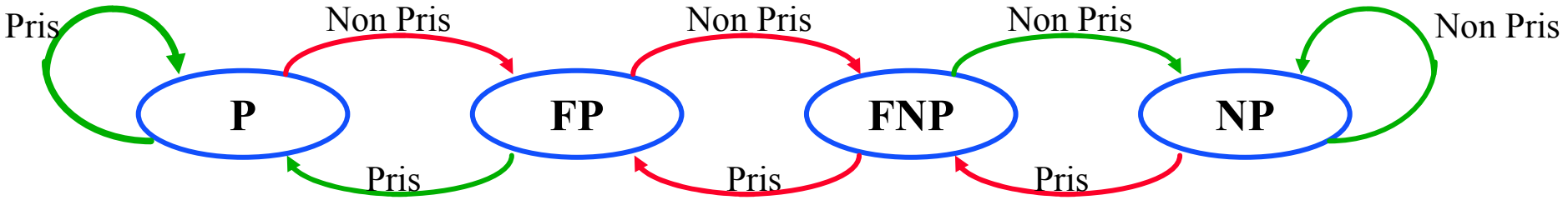
– au retour dans la boucle, lors de la première itération, le bit de prédiction est à 0 (dernier branchement non pris), donc le branchement est prédit non pris, mais il sera pris

RUPTURE PIPELINE

Augmentation de performances

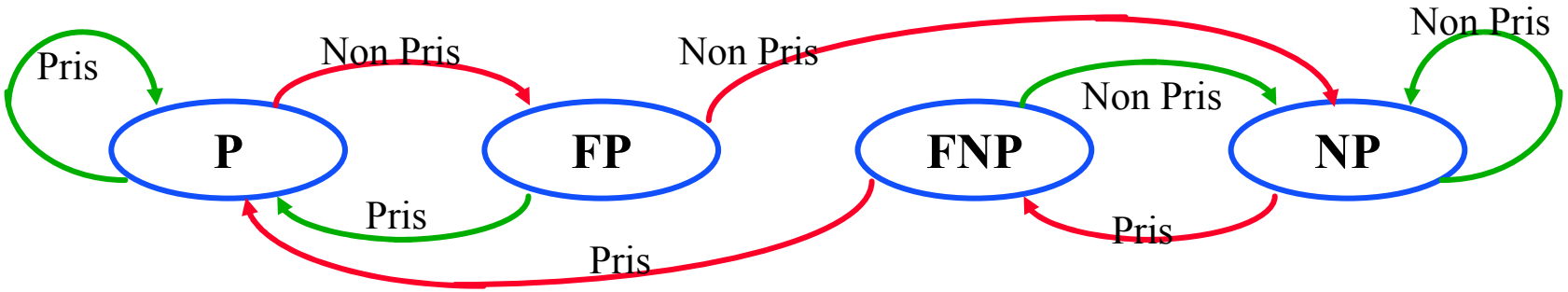
➢prédiction de branchement dynamique à 2 bits

- il s'agit d'un cas particulier d'un schéma plus général à base de compteur n bits à saturation
- un compteur n bits peut prendre ces valeurs entre 0 et $2^n - 1$
- si la valeur du compteur est supérieure à 2^{n-1} (la moitié de la valeur max) alors le branchement est prédit pris, sinon il est prédit non pris
- le compteur est incrémenté si le branchement est pris et décrémenté si le branchement est non pris
- des études ont montré que des prédicteurs à 2 bits sont presque aussi efficace que des prédicteurs n bits, et ils sont moins coûteux, donc la plupart des processeurs se limite à une prédiction à 2 bits
- ***Schéma 1 :***

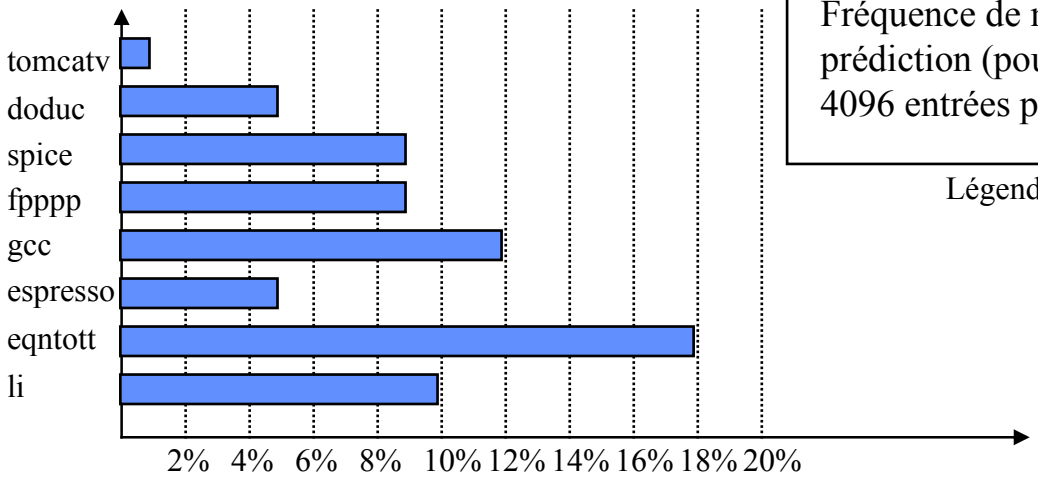


Augmentation de performances

– ***Schéma 2:***



– comportement vis à vis des boucles, très bonne prédiction :
  – une erreur de prédiction lors de la dernière itération de la boucle
  – pas d'erreur de prédiction au retour dans la boucle



Fréquence de mauvaise prédiction (pour un tampon de 2 bits à 4096 entrées pour les Spec89)

Légende :

tomcatv : prg de génération de grille (vectorisable)
doduc : simulation monte carlo d'un réacteur nucléaire
spice : logiciel de simulation de circuit
fpppp : prg de chimie quantique calculant les dérivées intégrale de 2 électons
gcc : compilateur C GNU
espresso : minimisation de fonctions booleennes
eqntott : traduit une équation booléenne en table de vérité
li : interpréteur lisp résolvant le problème des 9 reines

Augmentation de performances

➢Prédiction de branchement à corrélateur ou à deux niveaux

- dynamique à 2 bits qui tient compte des autres branchements
- soit le code :

```
if ( a == 2 )
        a = 0 ;
if ( b == 2 )
        b = 0;
if ( a != b )
        ....
```

Fragment de code du programme eqntott

- un mécanisme de prédiction tenant compte des branchements précédents sera plus efficace
  - exemple : branchement 1 non pris, branchement 2 non pris alors le branchement 3 sera forcement pris

Augmentation de performances

➢ Autre solution : déroulage de boucles lors de la compilation

- pour qu'un pipeline reste toujours plein, il faut exhiber le parallélisme
- dans un code de boucle, on peut dérouler (partiellement ou complètement) la boucle:

```
for (i=1 ; i<20 ; i=i+1) {
        x[i] = v1[i] + v2[i] ;
}
```

```
x[0] = v1[0] + v2[0] ;
x[1] = v1[1] + v2[1] ;
....
x[18] = v1[18] + v2[18] ;
x[19] = v1[19] + v2[19] ;
```

→ plus de branchement conditionnel, donc plus de perte de cycles

Augmentation de performances

➢ Autres solutions : branchement différé :

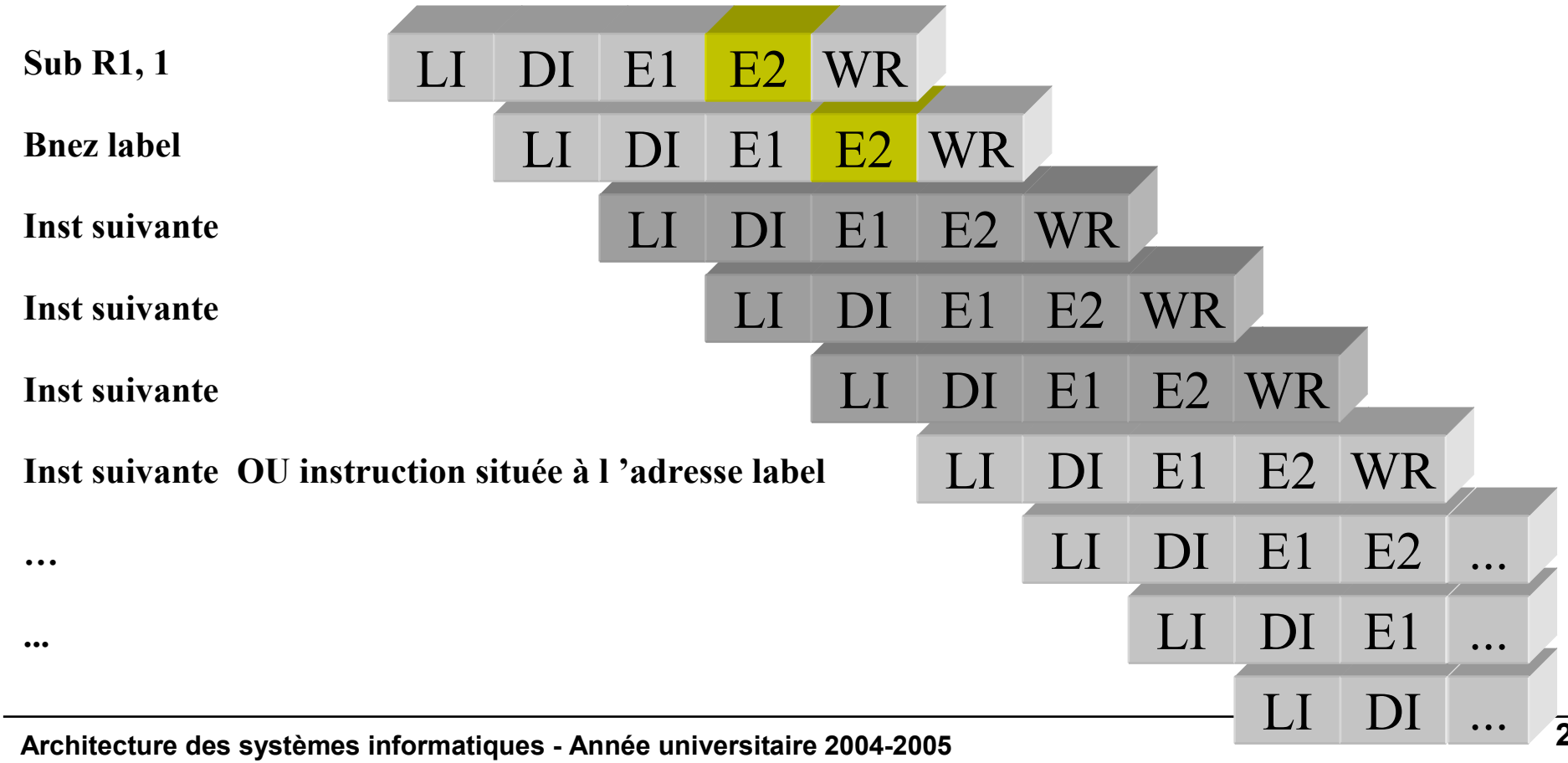
- ♦ notion de délai de branchement : n

  instruction de branchement
  successeur 1
  successeur 2
  ...
  successeur n

- ♦ les instructions successeurs sont exécutées que le branchement soit pris ou non
- ♦ en pratique toutes les machines à branchement différé ont un seul délai

Augmentation de performances

♦ Branchement différé, fonctionnement



Augmentation de performances

♦ Pour un branchement différé efficace :

– le délai de branchement doit être remplie efficacement :

• éviter les NOP

```
BI 1;
If condition then
            BI 2 ;
else
            BI 3 ;
end if;
BI 4;
```

Compilation non efficace

```
            BI 1
            Br condition, else
            NOP
            NOP
            NOP
            BI 2
            JUMP suite
else
            BI 3
suite
            BI 4
```

Compilation efficace

```
            BI 1
            Br condition, else
            Inst du BI 1
            Inst du BI 1
            Inst du BI 4
            BI 2
            JUMP suite
else
            BI 3
suite
            BI 4
```

Augmentation de performances

- Avantages des branchements différés :
  - mise en œuvre simple
  - ordonnancement par le compilateur assez simple à réaliser
  - convenait assez bien au premier processeur RISC, puisqu'il y avait des pipelines assez courts et des délais de branchement égaux à 1
- Inconvénients des branchements différés :
  - repose sur l'aspect implémentation (qui définit le délai de branchement)
  - l'allongement des pipelines ne permet plus d'ordonnancer simplement les opérations
  - perte de cycles importante

Augmentation de performances

➢Taille du tampon de prédiction :

- ♦ le taux d'efficacité de la prédiction dépend largement de la taille de ce tampon
- ♦ ce tampon fonctionne de la même manière qu'un cache :
  - – une adresse de branchement **@a1** qui n'a pas été utilisée depuis un certain temps est écrasée par une autre plus récente **@a2** (on perd donc la prédiction de branchement pour le branchement de l'adresse **@a1**)
- ♦ un tampon de faible taille augmente fortement le taux d'échec de la prédiction
- ♦ pour les programmes Spec89, avec un tampon de 4096 entrées, on obtient des taux d'échec variant entre 1 et 18 %.

Augmentation de performances

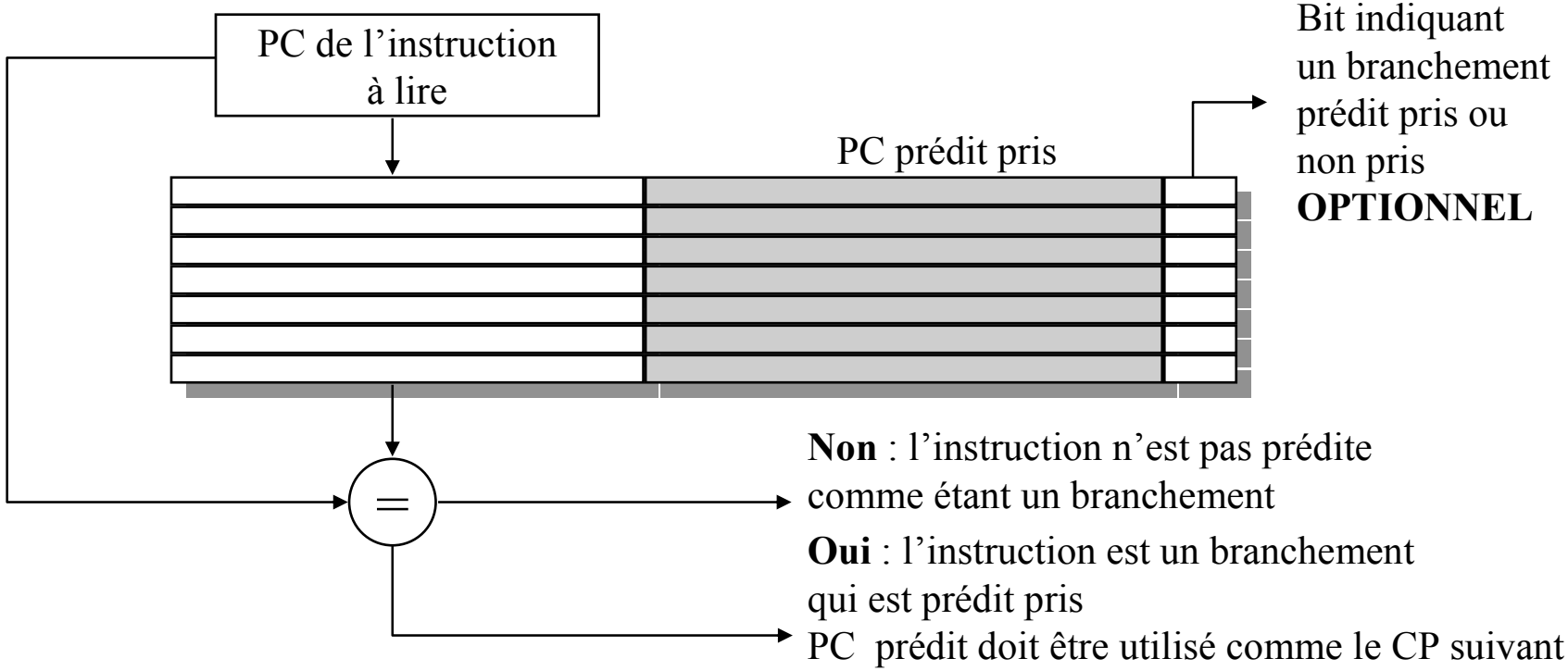
Exemples d'implémentation :

- pas de prediction — Intel 8086
- prédiction statique
  - jamais pris — Intel i486
  - toujours pris — Sun SuperSPARC
  - arrière pris, avant non pris — HP PA-7x00
  - semi statique, par profiling — PowerPCs
- prédiction dynamique :
  - 1-bit — DEC Alpha 21064, AMD K5
  - 2-bit — PowerPC 604, MIPS R10000, Cyrix 6x86 and M2, NexGen 586
  - two-level adaptive — PentiumPro, Pentium II, AMD K6
- prédiction hybride — DEC Alpha 21264
- Predication — Intel/HP Itanium et les DSP ARM processors, TI TMS320C6201
- Eager execution (limited) — IBM mainframes: IBM 360/91, IBM 3090

Augmentation de performances

❑ Tampon d'adresses de branchement

- ➢ l'intérêt est de connaître au plus tôt l'adresses de branchement
- ➢ fonctionnement similaire à celui d'un cache
- ➢ ce cache est indexé par l'adresse de l'instruction en cours de lecture (LI)
- ➢ on connaît donc l'adresse de l'instruction prédite dès la fin du cycle LI



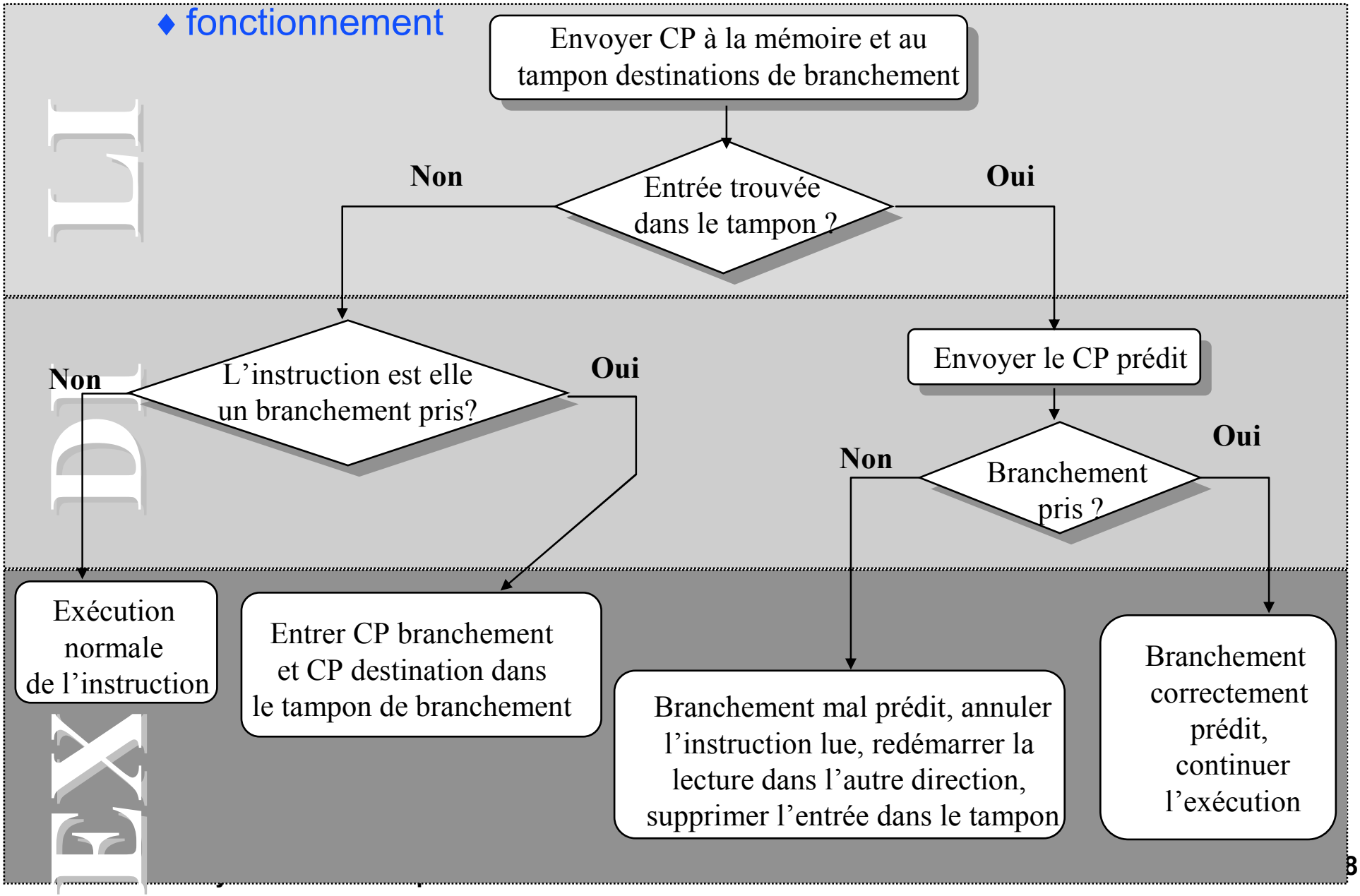
Augmentation de performances

➢ Fonctionnement :

- ♦ lorsque le processeur recherche l'instruction située à l'adresse @a, l'adresse @a est comparée à l'ensemble des adresses contenues dans le tampon.
- ♦ 2 cas peuvent alors se produire :
  - – l'adresse est présente dans le tampon, alors l'instruction que l'on est en train de lire est un branchement conditionnel qui a été pris lors de l'exécution précédent. Le branchement est alors prédit pris et l'instruction prédite suivante peut alors être chargée immédiatement.
  - – l'adresse n'est pas présente dans le tampon, l'instruction que l'on est en train de lire est une instruction classique ou une instruction de branchement conditionnel qui n'a pas été pris lors de l'exécution précédente.

Augmentation de performances

♦ fonctionnement



Augmentation de performances

❑ Trace Cache :

➢allongement des pipelines :

♦ conséquence : probabilité d'avoir plusieurs instructions de branchement dans le pipeline importante :

– plusieurs prédictions de branchement en cours

➢objectif du cache de trace :

♦ capturer la séquence d'instructions exécutée :

– de façon dynamique

• suite de blocs exécutés par le processeur

♦ différence avec un cache d'instructions :

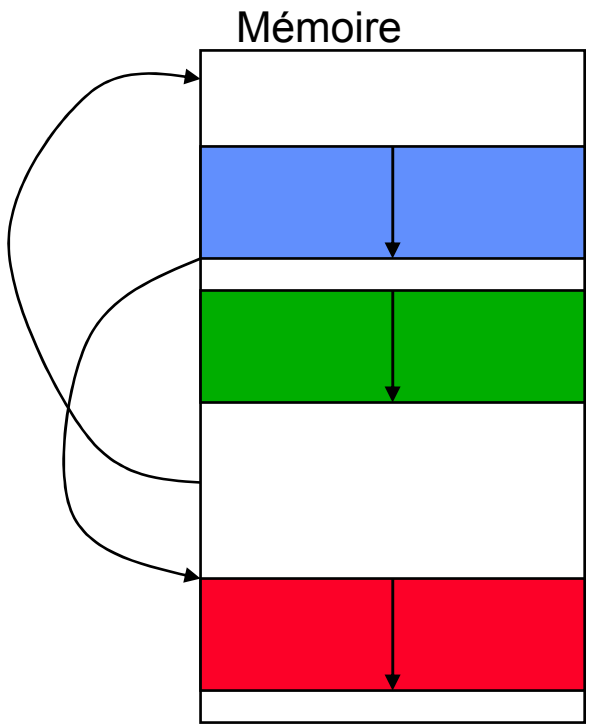
– le cache d'instructions mémorise une séquence statique d'instructions :

• suite de blocs consécutifs dans la mémoire

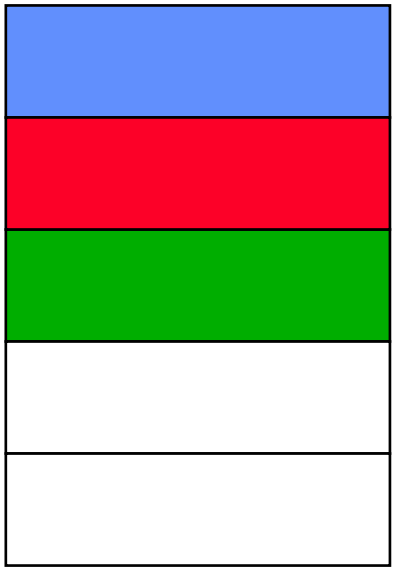
Augmentation de performances

➢Fonctionnement :

- ♦ soit le programme suivant en mémoire
- ♦ et soit l'exécution :
  - – bloc instructions 1
  - – appel à une fonction : bloc instructions 2
  - – bloc instructions 3



Cache d'instructions classique



Augmentation de performances

- la ligne du cache contient la séquence des instructions exécutées :
  - y compris les différents branchements conditionnels ou inconditionnels

Cache de trace

- si le processeur recommence la même séquence :
  - alors le trace de cache fournit la séquence d'instructions précédemment exécutée
  - pas de prédiction de branchement
  - si le processeur ne répète pas la même séquence, alors il faut stocker une nouvelle trace d'exécution

Augmentation de performances

❑ Exécution spéculative :

➢ idée générale :

- ♦ les instructions génèrent beaucoup de valeurs prédictibles
- ♦ le processeur est parfois bloqué par l'indisponibilité d'une opérande

➢ exemple :

```
add     R1, R2, R3      // R1 = R2 + R3
mult    R4, R1, R6      // R4 = R1 * R6
```

- ♦ la dépendance sur R1 bloque l'exécution de la multiplication

➢ solution :

- ♦ si l'addition a déjà été calculée :
  - – alors on peut poser comme hypothèse que le résultat de l'addition sera le même
  - – donc on peut exécuter spéculativement la multiplication
  - – lorsque l'addition sera terminée, deux possibilités :
    - le résultat est égal à celui spéculé ===> rien de particulier à faire
    - le résultat est différent de celui spéculé ===> il faut refaire le calcul

Augmentation de performances

❑ Renommage de registres

➢ Objectifs : casser les dépendances entre différentes données afin d'utiliser au mieux le pipeline

➢ Classification des aléas de données :

♦ LAE : lecture après écriture :

```
add R1, R2, R3        // R1 := R2 + R3
add R5, R1, R8        // R5 := R1 + R8
```

– la deuxième instruction lit l'ancienne valeur de R1
– c'est le type d'aléas le plus courant

♦ EAE : écriture après écriture :

```
mult  R1, R2, R3      // R1 :=R2 * R3
add   R1, R5, R8      // R1 := R5 +  R8
```

– existe uniquement si le pipeline du ***mult*** est plus long que le pipeline de l'***add***

Augmentation de performances

– ordre des écritures inversé, la valeur écrite dans R1 est le résultat du ***mult*** plutôt que le résultat du ***add***

MOVE R1, 0(R2)	LI	DI	EX 1	EX 2	EX 3	MEM	ER
ADD R1, R5, R8		LI	DI	EX	MEM	ER	

♦ EAL : écriture après lecture :

– très rare car la plupart des pipelines (sauf pour les processeurs à exécution dans le désordre)
  – lisent très tôt les données à traiter
  – écrivent très tard les données traitées

```
add   R2, R1, R3      // R2 := R1 + R3
add   R1, R4, R5      // R1 := R4 + R5
```

♦ LAL : lecture après lecture, ce n'est pas un aléas de fonctionnement

```
add   R2, R1, R3      // R2 := R1 + R3
add   R4, R1, R5      // R4 := R1 + R5
```

Augmentation de performances

➢Solutions :

- LAE : pas de solution en effectuant du renommage de registres, la solution consiste alors à intercaler des instructions indépendantes là où c'est nécessaire
- EAE : on peut renommer le registre dans lequel est effectué la dernière écriture

```
mult   R1, R2, R3      // R1 :=R2 * R3
add    R'1, R5, R8     // R'1 := R5 +  R8
```

- EAL : idem précédent

```
add   R2, R1, R3       // R2 :=  R1 + R3
add   R'1, R4, R5      //  R'1 := R4 + R5
```

- Solution idéale : disposer d'un nombre infini de registres virtuels
- Effets du nombre fini de registres : limite le nombre d'instructions qu'il est possible de lancer simultanément

Augmentation de performances

❑ Exécution dans le désordre :

- ➢ motivations :
  - ♦ assurer au mieux le remplissage des pipelines de toutes les unités disponibles
  - ♦ permettre le maximum de parallélisme
- ➢ Pour un processeur superscalaire de degré N, les dépendances de données ou d'instructions provoque des blocages des pipelines
- ➢ Il s'agit de réaliser un ***ordonnancement dynamique*** des opérations en fonction de la disponibilité des ressources
- ➢ Pour un processeur superscalaire de degré N, c'est donc la possibilité d'exécuter ***toujours*** N instructions par cycle
- ➢ 2 façons de réaliser l'exécution spéculative :
  - ♦ statique :
    - – très souvent réaliser par le logiciel, le compilateur :
      - – difficile à réaliser lorsqu'il y a manipulation de pointeurs
    - – pas optimale, surtout lorsque le processeur supporte un système d'exploitation multitâches (chaque tâche est ordonnancée au mieux, mais l'ensemble !!!)

Augmentation de performances

- dynamique :
  - par le matériel, logique de contrôle très importante gérant les émission d'instructions vers les différentes unités
  - beaucoup plus souple

➢ D'une façon générale l'exécution dans le désordre est très liée à :

- la prédiction de branchement
  - très bons résultats si la prédiction matérielle de branchement est bonne
- la détection des dépendances :
  - renommage des registres, afin de casser les dépendances de données
- la taille des buffers (*locaux et globaux)* de préchargement d'instructions :
  - plus le buffer est important, plus le processeur a de possibilité pour lancer des instructions aux différents temps

Augmentation de performances

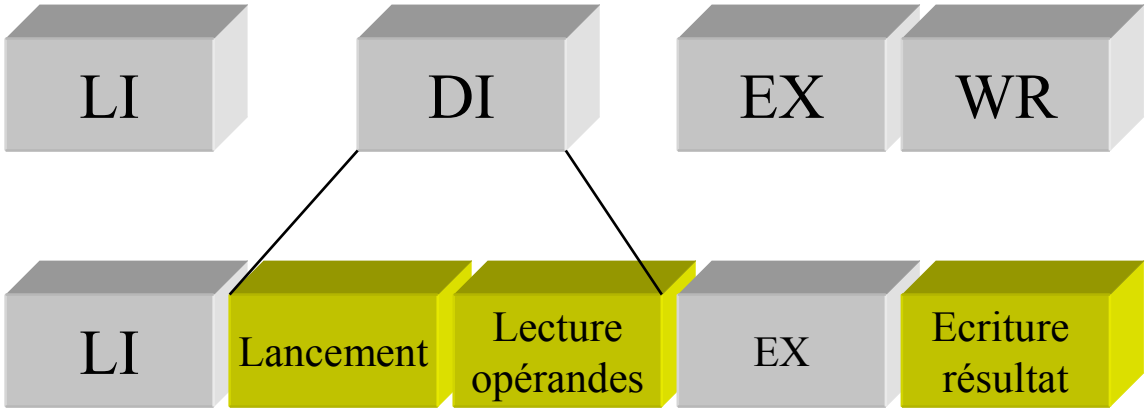
➢Tous les étages ne sont pas exécutés dans le désordre :

- dans l'ordre :
  - Lecture d'instruction
  - Décodage d'instruction
  - Renommage de registres
- dans le désordre :
  - exécution d'instruction
- dans l'ordre :
  - mise à jour état processeur
  - écriture dans les registres destination et/ou mémoire

Augmentation de performances

➢Le tableau des marques :

- ♦ il s'agit d'une structure de données permettant :
  - – d'assurer l'exécution non ordonnée des instructions
  - – de détecter les aléas
- ♦ il détermine quand l'instruction :
  - – peut lire ses opérandes sources et commencer son exécution
  - – peut écrire son résultat dans le registre destination



Pipeline classique

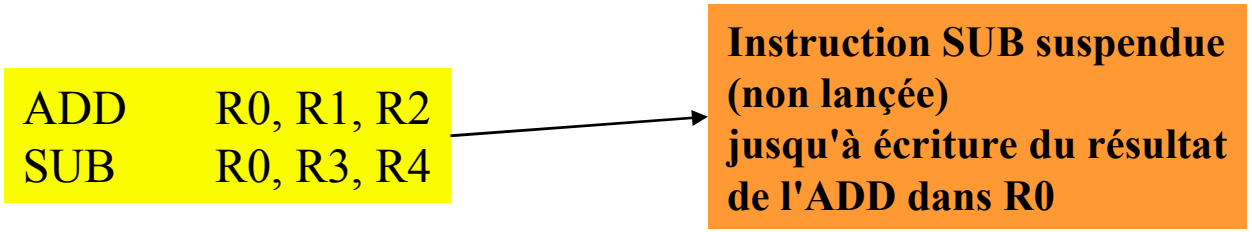
Pipeline avec tableau des marques

Augmentation de performances

♦ Les étapes du pipeline :

– **Lancement :**

- **Si** une unité fonctionnelle est libre pour exécuter l'instruction
- **et si** il n'y a pas d'autre instruction active qui a le même registre destination (aléas de type EAE) :

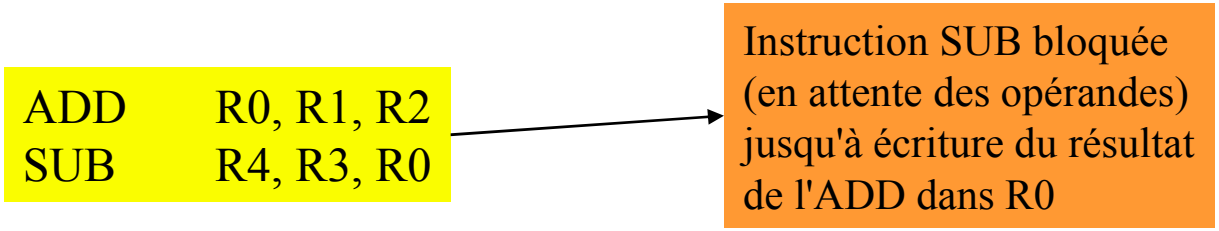


- **alors** le tableau des marques lance l'instruction
- **et** effectue une mise à jour de sa structure de données
- **Sinon**, l'instruction est suspendue jusqu'à disparition de la dépendance

Augmentation de performances

– **Lecture opérandes :**

- un opérande source est disponible si aucune autre instruction active lancée auparavant ne va le modifier (détection des dépendances de type LAE)

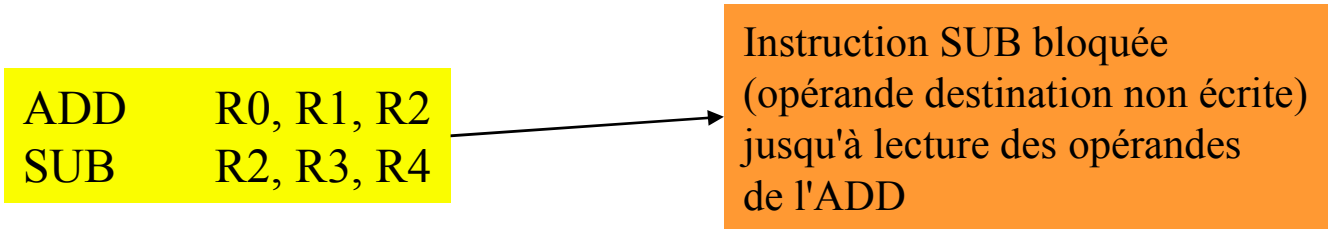


- **lorsque tous les opérandes sont disponibles**, l'instruction peut passer dans l'étage d'exécution

Augmentation de performances

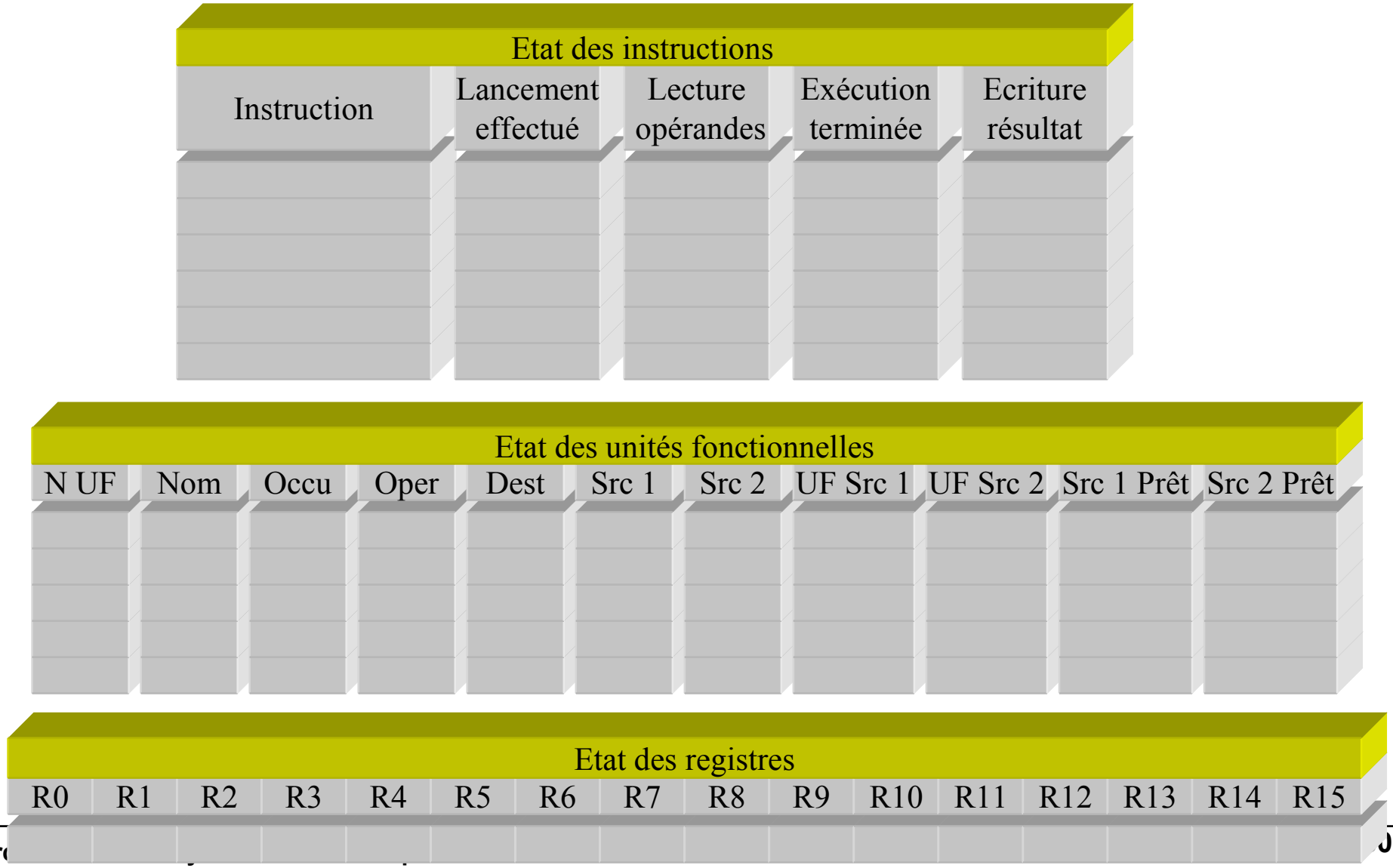
– **Exécution instruction :**
  - lorsque l'unité fonctionnelle a terminé, elle l'indique au tableau des marques
  - l'unité fonctionnelle **n'effectue pas** l'écriture dans l'opérande destination

– **Ecriture résultat :**
  - le tableau des marques **vérifie** qu'il n'existe pas une instruction lancée auparavant qui n'aurait pas encore lue ses opérandes (dépendances de type EAL)



Augmentation de performances

➢Représentation du tableau des marques



Augmentation de performances

➢ Exemples :

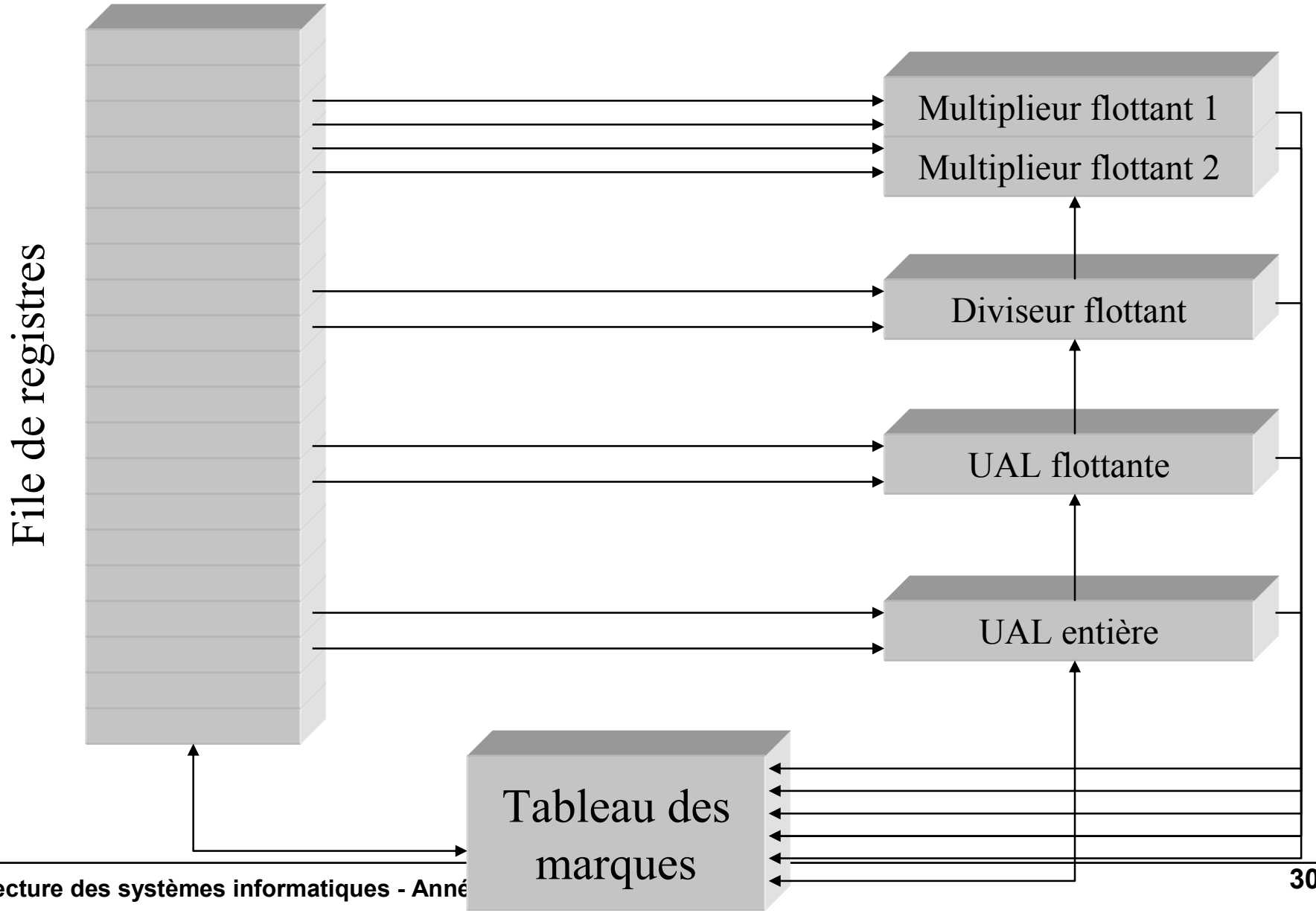
♦ on admettra que le processeur :

– analyse une fenêtre d'instruction de taille 6

– possède :

- 1 unité entière UAL (assurant aussi les accès mémoire)
- 1 unité flottante UAL
- 2 unités de multiplication flottante
- 1 unité de division flottante

Augmentation de performances



Augmentation de performances

➢ Exemple 1 : code avec dépendances

♦ soit le code suivant :

- LOAD **R2**, (R3)
- LOAD **R6**, (R2)
- MULTF **R0**, **R2**, R4
- SUBF R8, **R6**, **R2**
- DIVF R10, **R0**, **R6**
- ADDF **R6**, R8, R2

LAE sur :
- R6
- R2
- R0

EAL sur :
- R6

Augmentation de performances

♦ Temps de cycle 1 : analyse des 6 instructions

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R2, (R3)**	#			
LOAD	**R6, (R2)**				
MULTF	**R0, R2, R4**	#			
SUBF	**R8, R6, R2**	#			
DIVF	**R10, R0, R6**	#			
ADDF	**R6, R8, R2**				

> **Unité UAL libre**
> **Plus d'UAL dispo**
> **Unité Mult libre**
> **Unité UALF libre**
> **Unité Div libre**
> **Plus d'UALF dispo**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R2	R3				Oui	
2	Mult1	#	multf	R0	R2	R4	UAL		Non	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2		UAL	Non	Non
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Non

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1		UAL				UAL		UALF		Div					

Augmentation de performances

♦ Temps de cycle 2

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R2, (R3)**	#	#		
LOAD	**R6, (R2)**				
MULTF	**R0, R2, R4**	#			
SUBF	**R8, R6, R2**	#			
DIVF	**R10, R0, R6**	#			
ADDF	**R6, R8, R2**				

> Lecture opérandes
> Plus d'UAL dispo
> Opérandes non dispo
> Opérandes non dispo
> Opérandes non dispo
> Plus d'UALF dispo

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R2	R3				*Oui*	
2	Mult1	#	multf	R0	R2	R4			Non	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Non	Non
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Non

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1		UAL				UAL		UALF		Div					

Augmentation de performances

♦ Temps de cycle 3

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat	
LOAD	**R2, (R3)**	#	#	#		**> Exécution terminée**
LOAD	**R6, (R2)**					**> Plus d'UAL dispo**
MULTF	**R0, R2, R4**	#				**> Opérandes non dispo**
SUBF	**R8, R6, R2**	#				**> Opérandes non dispo**
DIVF	**R10, R0, R6**	#				**> Opérandes non dispo**
ADDF	**R6, R8, R2**					**> Plus d'UALF dispo**

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R2	R3				Oui	
2	Mult1	#	multf	R0	R2	R4			Non	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Non	Non
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Non

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1		UAL						UALF		Div					

Augmentation de performances

♦ Temps de cycle 4

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R2, (R3)**	#	#	#	#
LOAD	**R6, (R2)**				
MULTF	**R0, R2, R4**	#			
SUBF	**R8, R6, R2**	#			
DIVF	**R10, R0, R6**	#			
ADDF	**R6, R8, R2**				

> Ecriture résultat
> Plus d'UAL dispo
> Opérandes non dispo
> Opérandes non dispo
> Opérandes non dispo
> Plus d'UALF dispo

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R2	R3				Oui	
2	Mult1	#	multf	R0	R2	R4			***Oui***	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Non	***Oui***
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Non

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1		***UAL***				UAL		UALF		Div					

Augmentation de performances

♦ Temps de cycle 5

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R2, (R3)**	#	#	#	#
LOAD	**R6, (R2)**	#			
MULTF	**R0, R2, R4**	#	#		
SUBF	**R8, R6, R2**	#			
DIVF	**R10, R0, R6**	#			
ADDF	**R6, R8, R2**				

> Instruction terminée
> UAL dispo
> Opérandes disponibles
> Opérandes non dispo
> Opérandes non dispo
> Plus d'UALF dispo

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R6	R2				Oui	
2	Mult1	#	multf	R0	R2	R4			Oui	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Non	Oui
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Non

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1						UAL		UALF		Div					

Augmentation de performances

♦ Temps de cycle 6

Etat des instructions

Instruction		Lancement effectué	Lecture opérandes	Exécution terminée	Ecriture résultat
LOAD	**R2, (R3)**	#	#	#	#
LOAD	**R6, (R2)**	#	#		
MULTF	**R0, R2, R4**	#	#		
SUBF	**R8, R6, R2**	#			
DIVF	**R10, R0, R6**	#			
ADDF	**R6, R8, R2**				

> Instruction terminée
> Opérande dispo
> Cycle 1 (3 cycles)
> Opérandes non dispo
> Opérandes non dispo
> Plus d'UALF dispo

Etat des unités fonctionnelles

N UF	Nom	Occu	Oper	Dest	Src 1	Src 2	UF Src 1	UF Src 2	Src 1 Prêt	Src 2 Prêt
1	UAL	#	load	R6	R2				Oui	
2	Mult1	#	multf	R0	R2	R4			Oui	Oui
3	Mult2									
4	UALF	#	subf	R8	R6	R2	UAL		Non	Oui
5	Div	#	divf	R10	R0	R6	Mult1	UAL	Non	Non

Etat des registres

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Mult1						UAL		UALF		Div					

Augmentation de performances

- Stations de réservation :
  - les instructions attendent leurs opérandes dans les stations de réservation
  - les stations peuvent être :
    - globales aux unités fonctionnelles
    - locales aux unités fonctionnelles

Augmentation de performances

❑ Modèle SMT ou Hyperthreading

- ➢ Simultaneous MultiThreading
- ➢ hyperthreading (terme utilisé par Intel)
- ➢« Exécution simultanée de plusieurs processus léger »
- ➢Motivations :
  - ♦ un processeur superscalaire de degré N, peut exécuter, en théorie, jusqu ’à N instructions par cycle, or on observe que ce type de machine n ’exécute guère plus de 2 ou 3 instructions par cycle (parallélisme limité) :
    - – granularité de parallélisme fine
  - ♦ une machine multiprocesseur exécute plusieurs processus sur différents processeurs, mais globalement le parallélisme inter-processus reste limité :
    - – granularité de parallélisme importante

Augmentation de performances

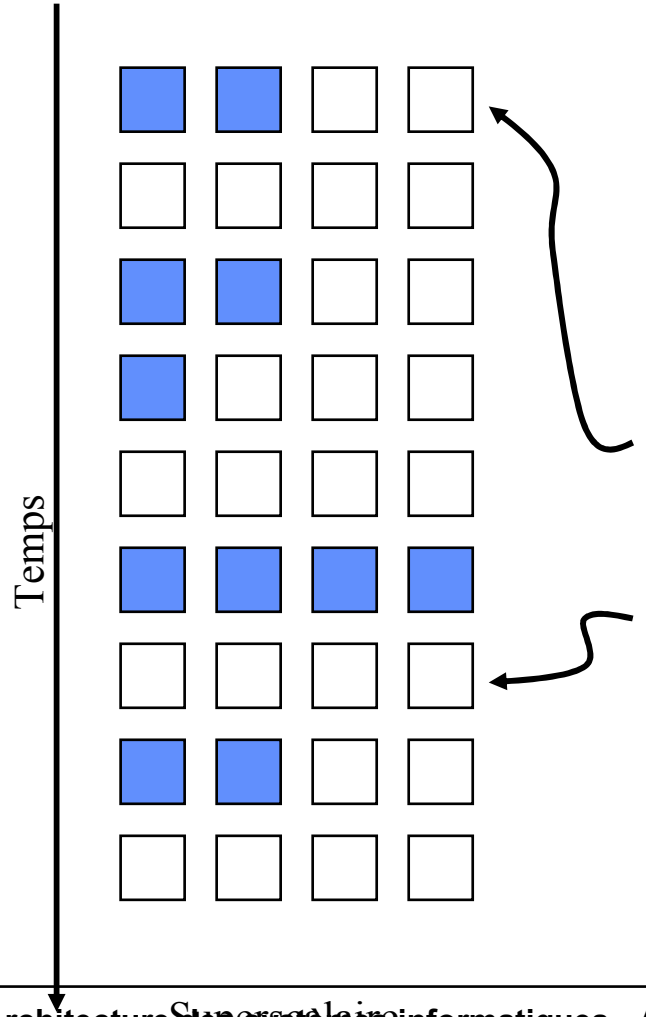
➢Objectif des SMT :

- ♦ exploiter tous les parallélismes disponibles dans les applications :
  - – grains fin ou gros grains
- ♦ utilisation efficace des ressources de la machine

Augmentation de performances

❑ Le SMT : comment ça marche ?

➢Rappel : fonctionnement d'une machine superscalaire



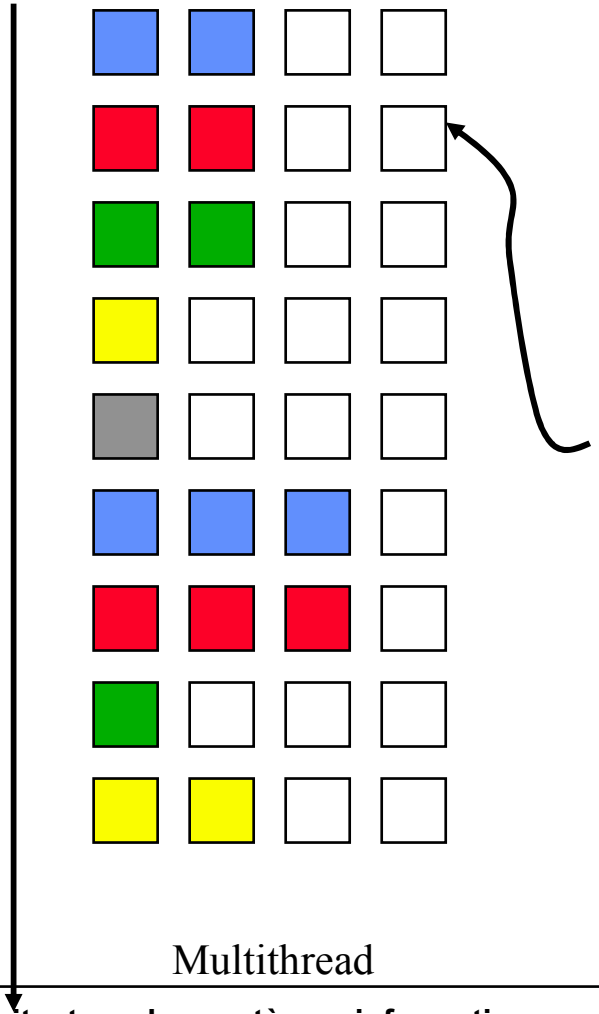
A chaque temps de cycle, le processeur tente de lancer un maximum d'instructions d'un processus.

Le faible parallélisme des applications provoque des trous dans les lignes horizontales.

De plus, certaines instructions demandent plus d'un cycle pour être exécutées, cela provoque des trous dans les lignes verticales

Augmentation de performances

➢Rappel : fonctionnement d'une machine multithread



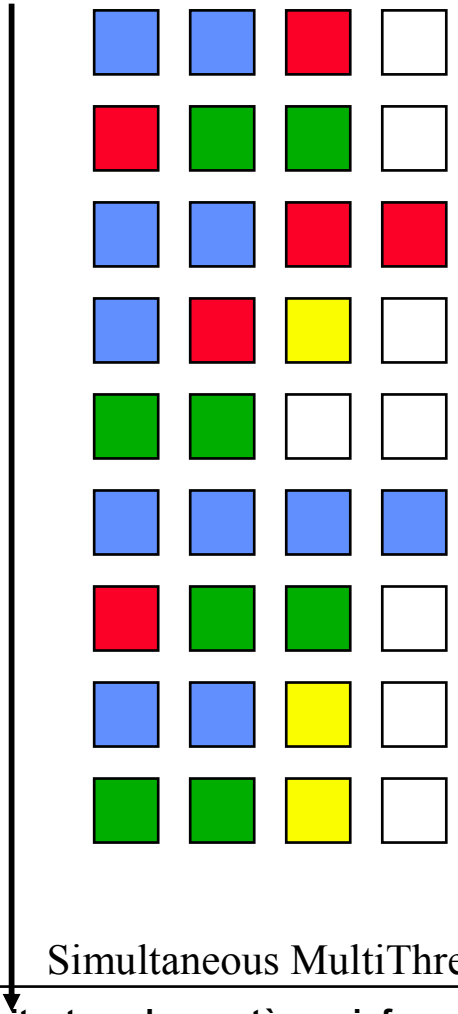
A chaque temps de cycle, le processeur tente de lancer un maximum d'instructions d'un processus.
Au temps de cycle suivant, il fait de même avec un autre processus.

Le faible parallélisme intra-processus provoque des trous dans les lignes horizontales.

Les trous dans les lignes verticales sont supprimés.

Augmentation de performances

➢Fonctionnement d'une machine multithread



Simultaneous MultiThread

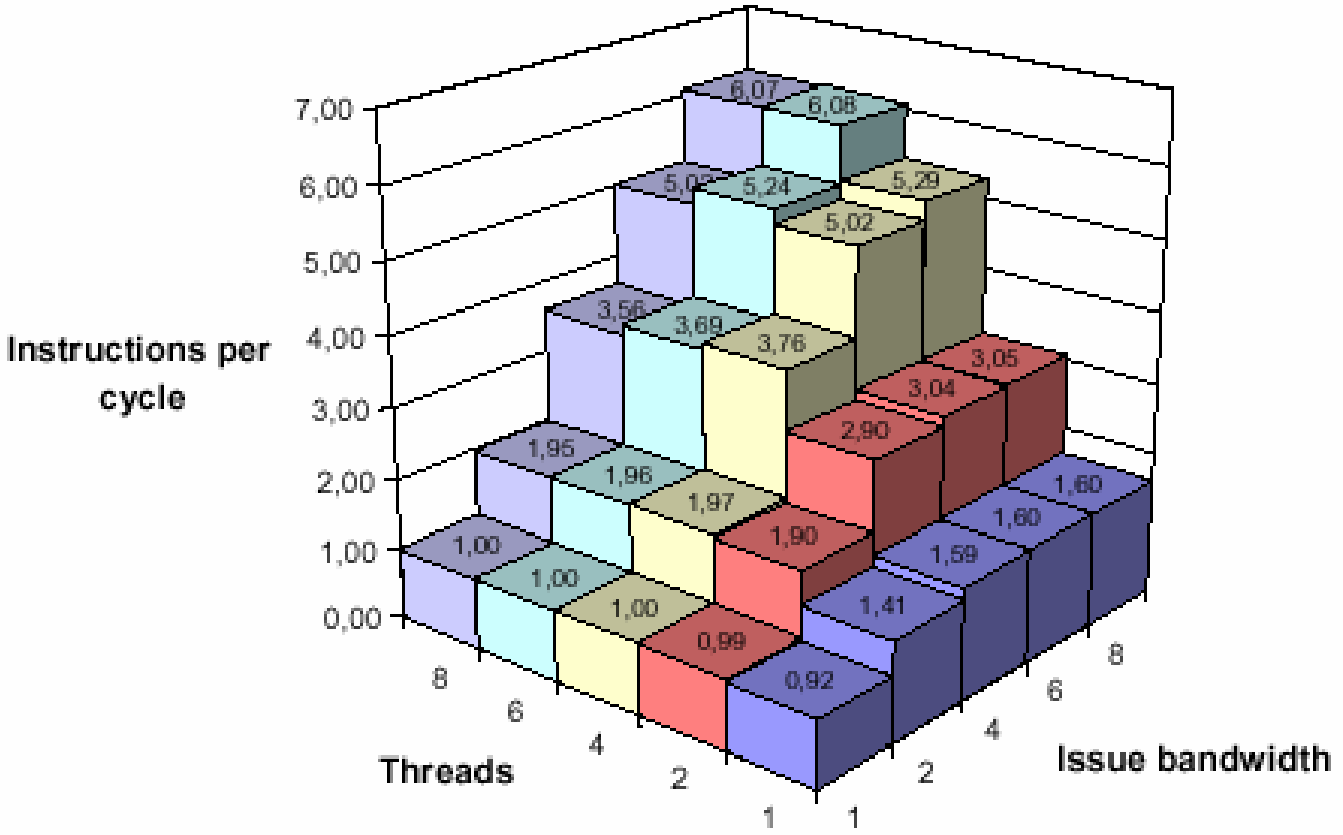
A chaque temps de cycle, le processeur tente d'occuper un maximum de ses ressources.
Il lance un maximum d'instructions parmi toutes les instructions de tous les processus de l'application

Les tous dans les lignes horizontales sont moins nombreux.

Les trous dans les lignes verticales sont supprimés.

Augmentation de performances

- **Performances :**
  - ➢ le modèle SMT offre de bonne performances pour les applications qui ne peuvent être parallélisée
  - ➢ meilleure utilisation des ressources du processeur



Augmentation de performances

- **Processeurs implémentant ce concept:**
  - Power PC G3
  - l'ex futur processeur Alpha 21464, prévu pour 2003
  - le Pentium 4 :
    - environ 5% de surface supplémentaire pour une augmentation de performance de 27 %)

Augmentation de performances

- **Coût matériel :**
  - ➢nécessite la mise en place de plusieurs contexte physique
- **Processeurs classiques :**
  - ➢1 contexte physique
  - ➢plusieurs contextes d'exécution
- **Processeurs SMT :**
  - ➢plusieurs contextes physiques
  - ➢plusieurs contextes d'exécution

Augmentation de performances

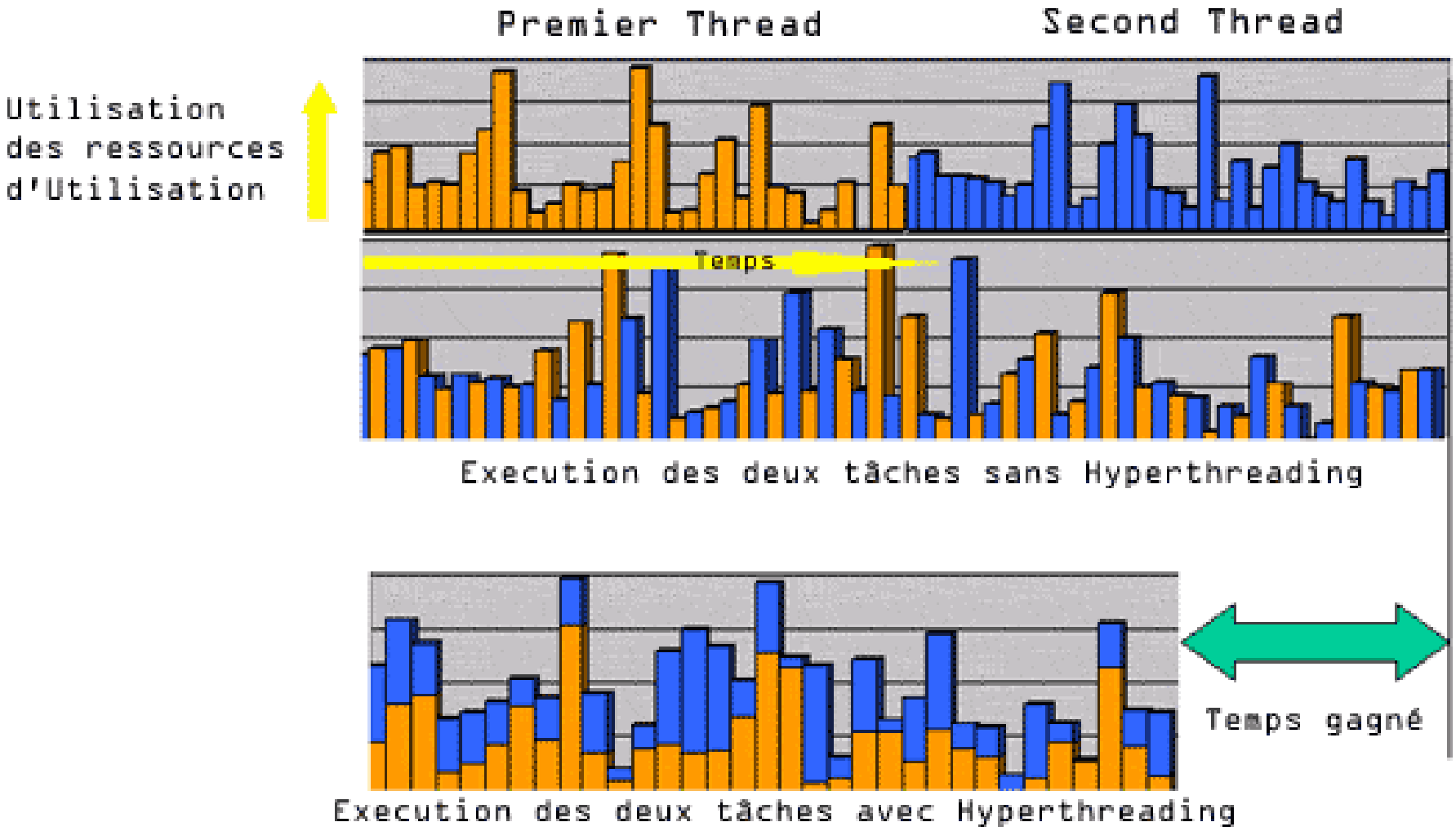
L'hyperthreading du Pentium 4

- à un cycle donné le processeur exécute 1 thread
- un seul thread ne parvient pas à occuper les 20 étages pipelines d'un Pentium 4

- objectif : permettre l'exécution de plusieurs threads
  - mise en place de plusieurs contextes physiques d'exécution

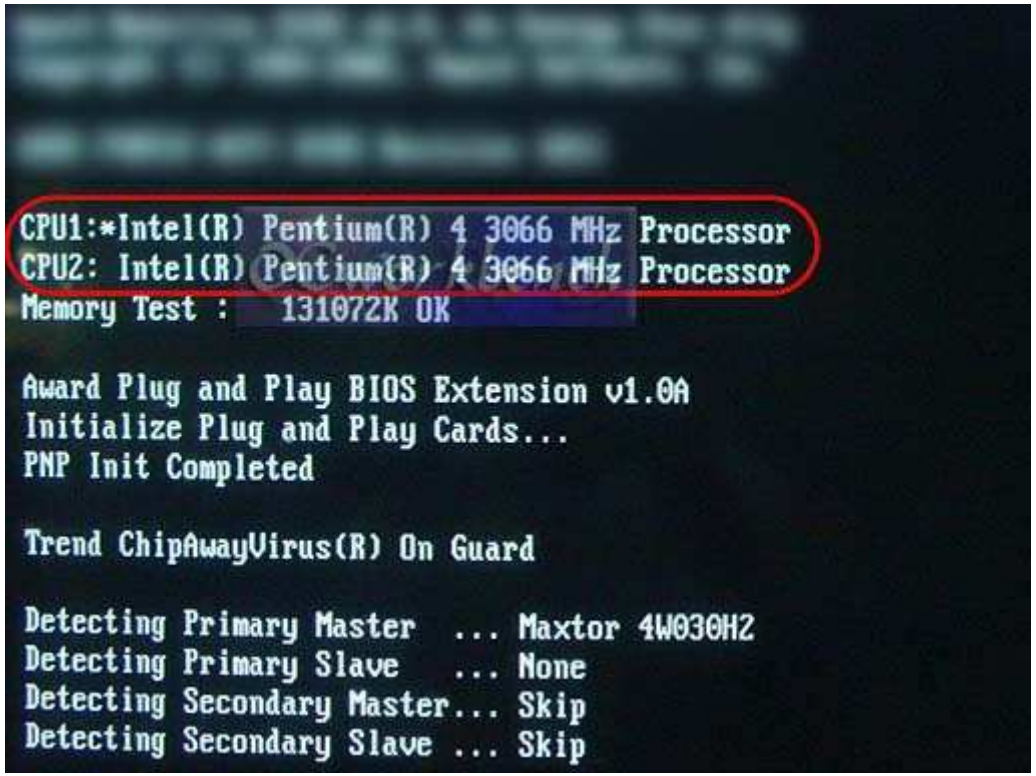
Augmentation de performances

- **L'hyper threading suite :**



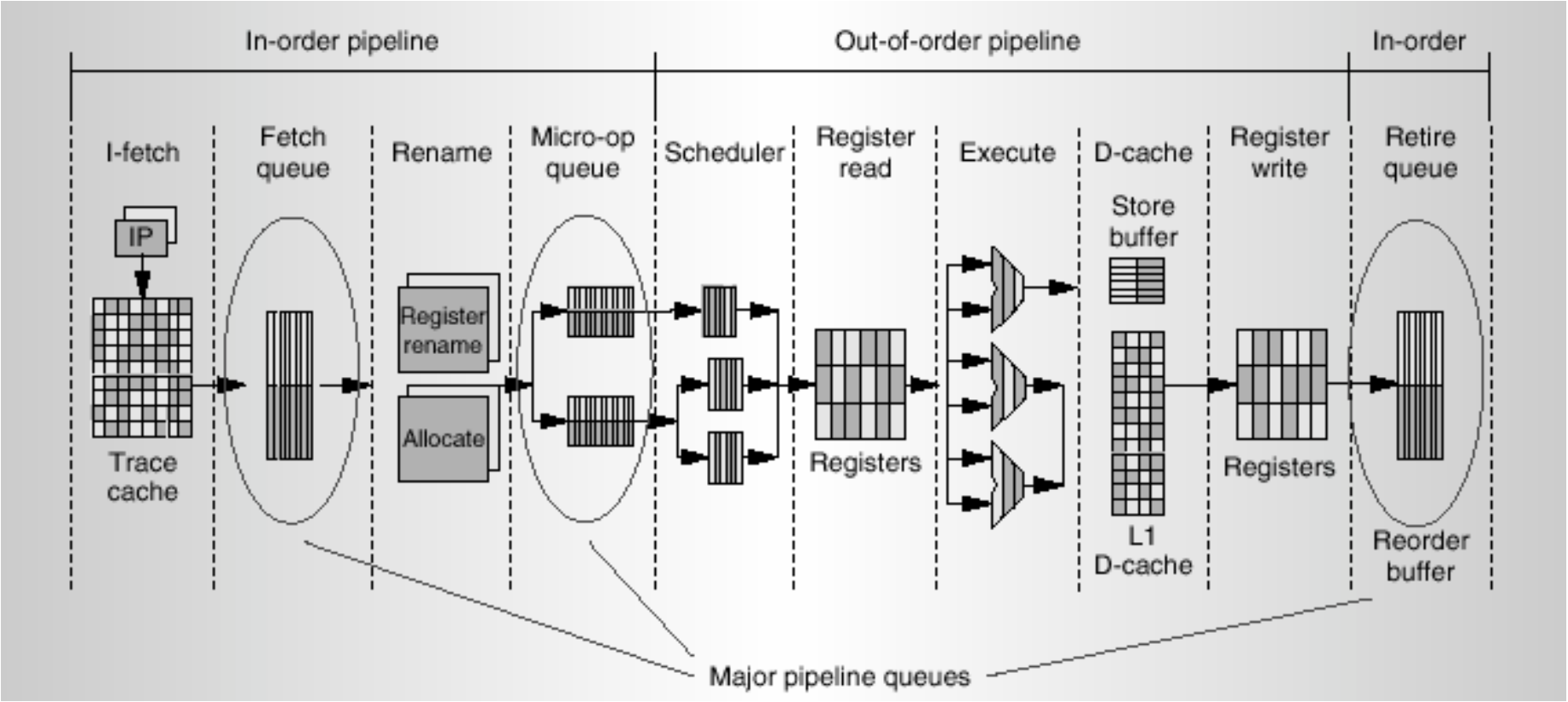
Augmentation de performances

- **L'hyper threading suite :**
  - ➢ le système d'exploitation voit 2 processeurs



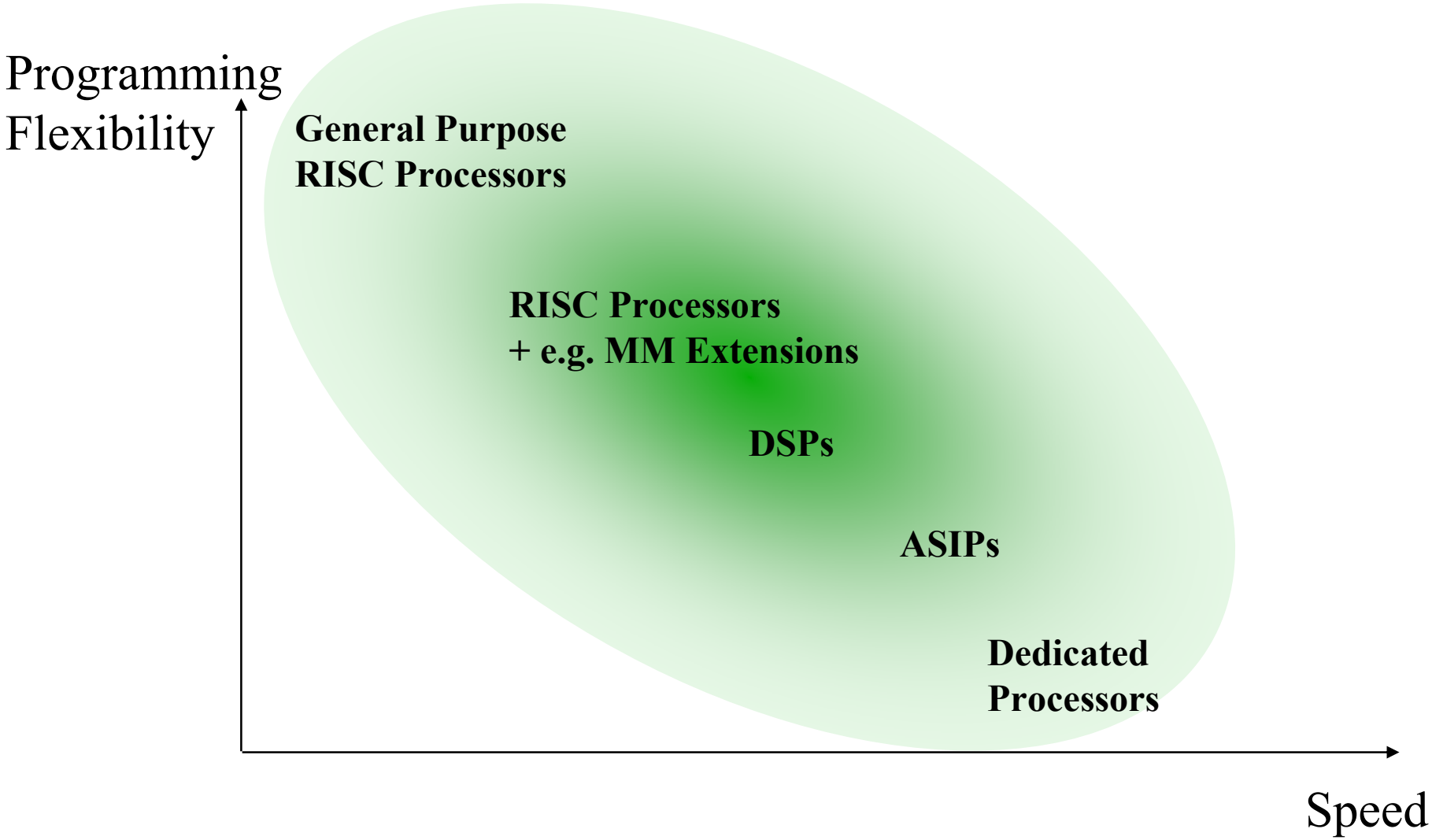
Augmentation de performances

➢ les principales parties du pipeline du Pentium 4



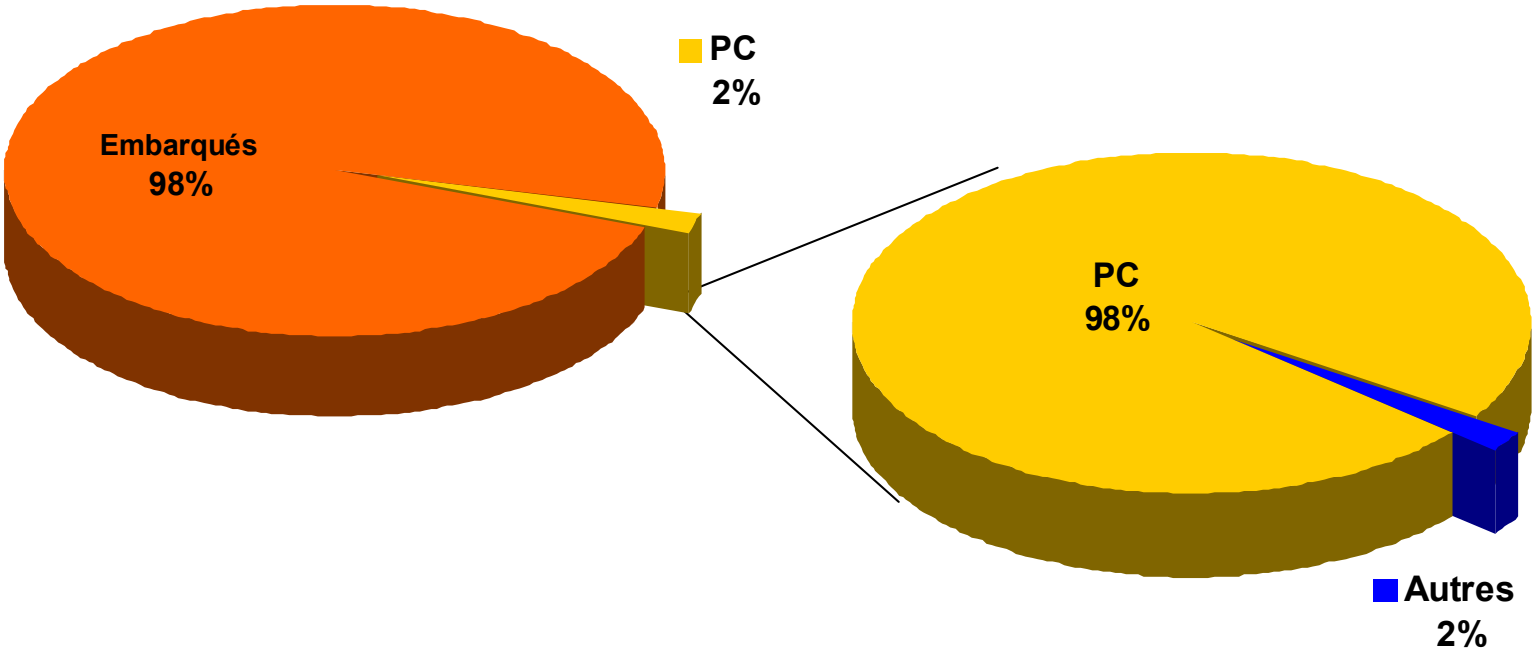
PANORAMA

Panorama de processeurs



Panorama de processeurs

- **Volume des ventes :**
  - ➢50 millions de microprocesseurs pour PC
  - ➢4 milliards de microprocesseurs
  - ➥ Le marché économique est dans les µP *embarqués*



Les processeurs généraux

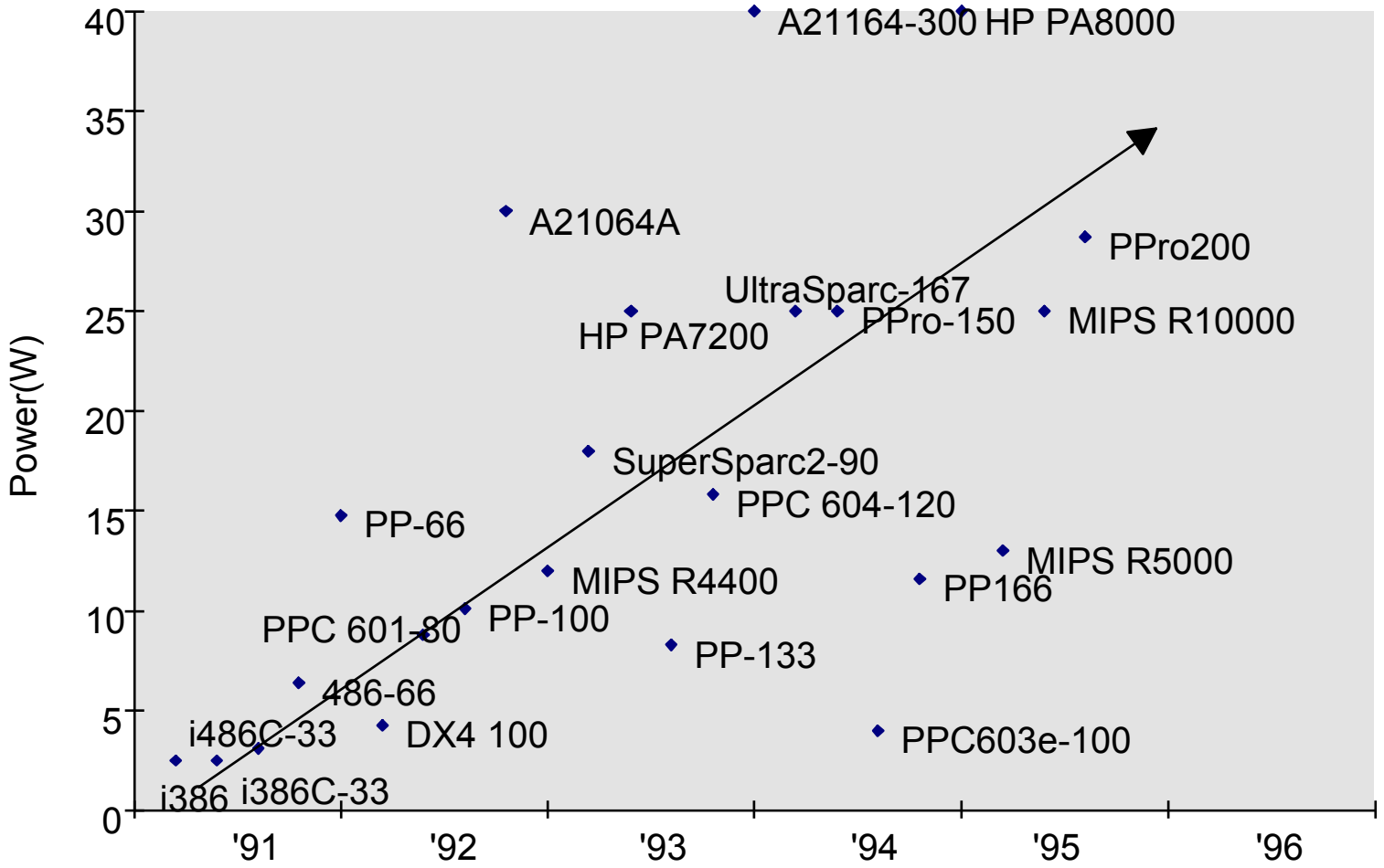
- Voir tableaux d'informations générales sur les différents constructeurs :
  - **http://infopad.EECS.Berkeley.EDU/CIC/summary/local/**
- Remarques générales :
  - augmentation de la consommation des processeurs :
    - c'est une conséquence de l'augmentation
      - de la fréquence d'horloge
      - du nombre de transistors
      - de la taille totale du circuit
    - des études de refroidissement ont été menées :
      - nouvelle technique de refroidissement (caloduque)
      - abaissement de température (jusqu'à - 40 °C)
  - les tensions d'alimentation diminuent :
    - pour limiter l'augmentation de consommation : (Pentium Pro = 30 Watts ====> 8 Watts)

$$P = \alpha \cdot C \cdot Vdd^2 \cdot f$$

Panorama de processeurs

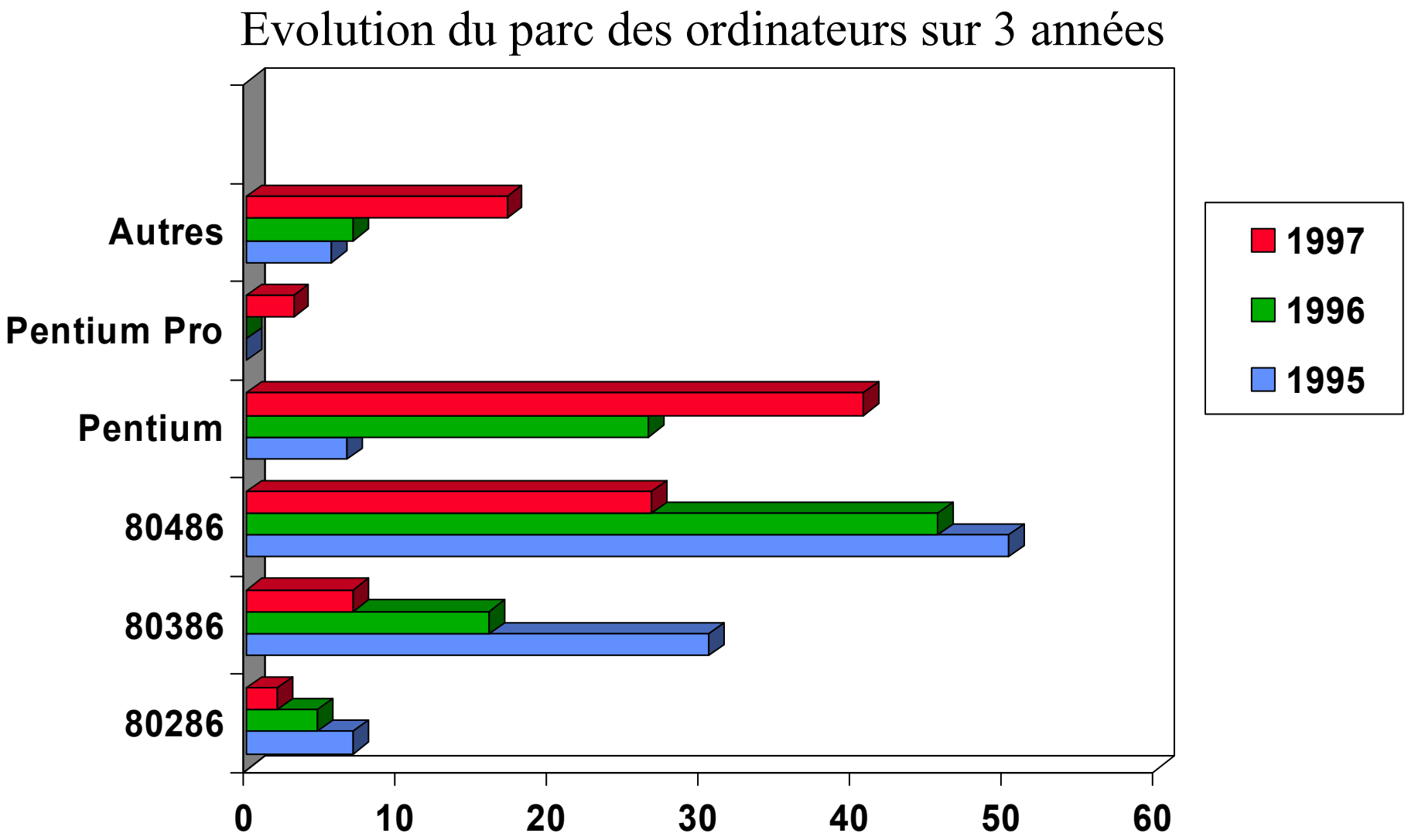
♦ Evolution de la consommation des microprocesseurs

– x4 tous les 3 ans



[Source: Microprocessor Report]

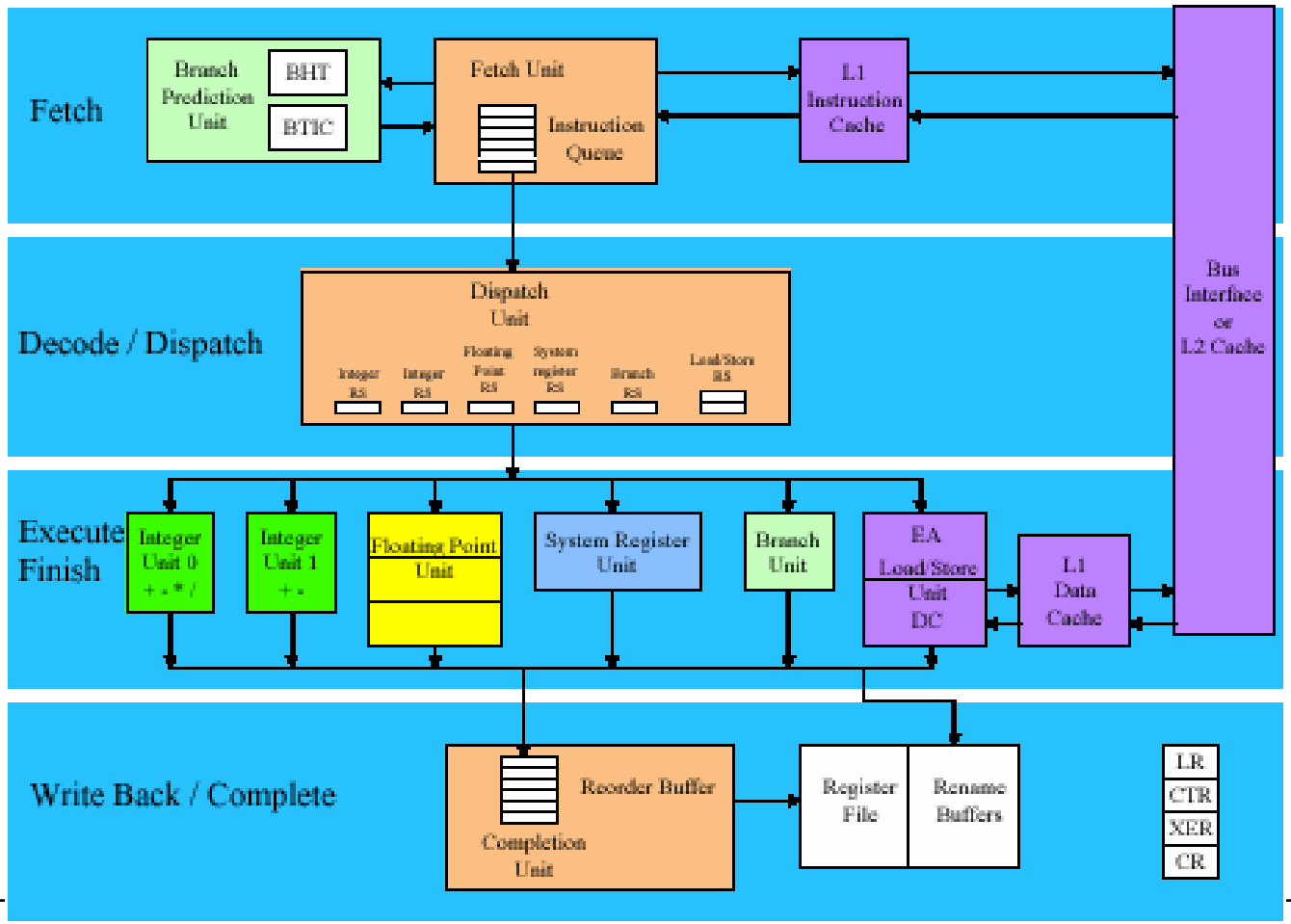
Panorama de processeurs



Panorama de processeurs

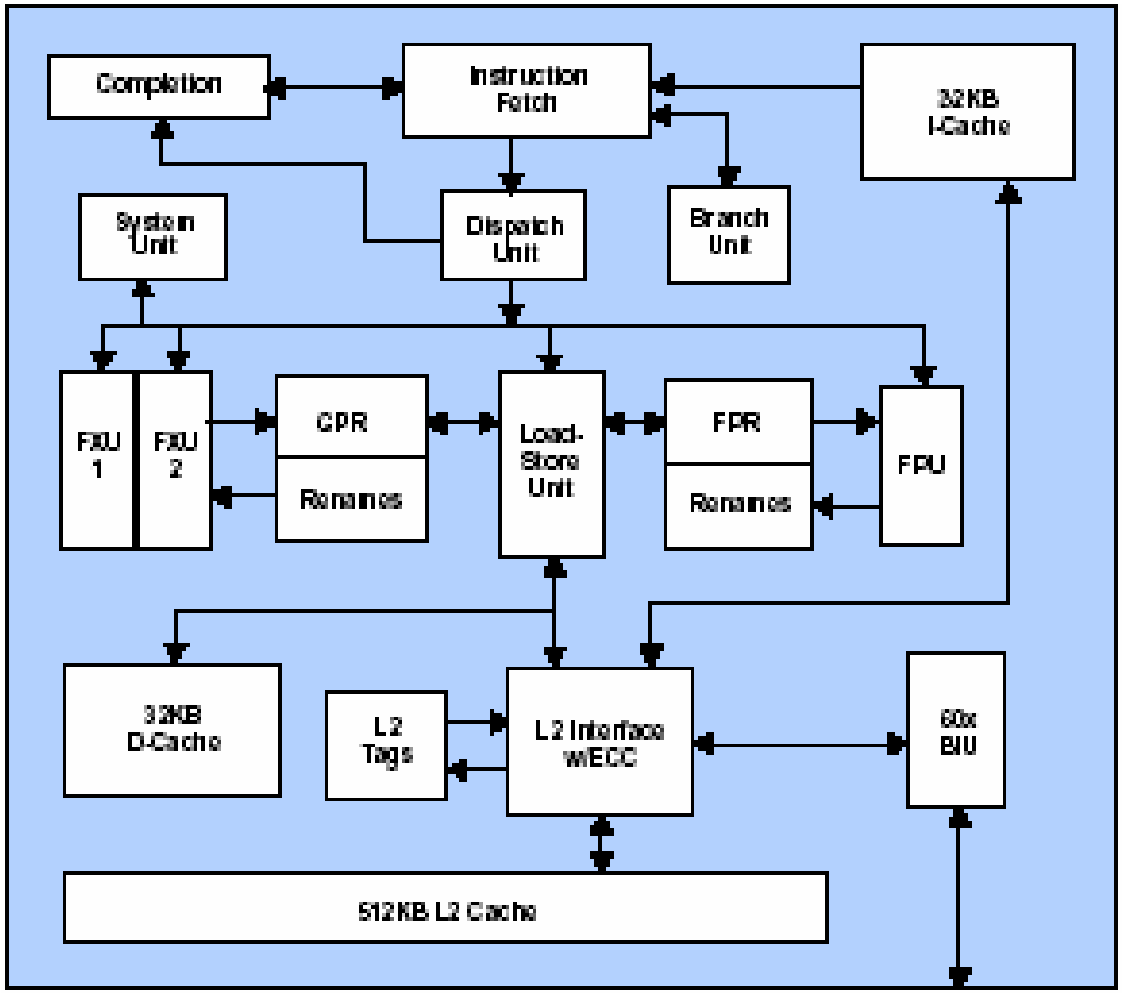
❑ Exemple de processeur à architecture power pc

➢le power pc 750



Panorama de processeurs

❑ Le power PC 750 FX



Panorama de processeurs

- Power PC 750 FX
  - fréquence : 600 à 1000 Mhz
  - cache L1 de 2 fois 32 Ko
  - cache L2 de 512 Ko interne
  - technologie : 0,13 micron
  - mémoire adressable : 64 Go (36 bits d'adresses)
  - tensions d'alimentation :
    - 1,4 V pour le cœur
    - 3,3 V pour les I/O
  - dissipation : 3,6 W (800 Mhz - 1,4 V)

Panorama de processeurs

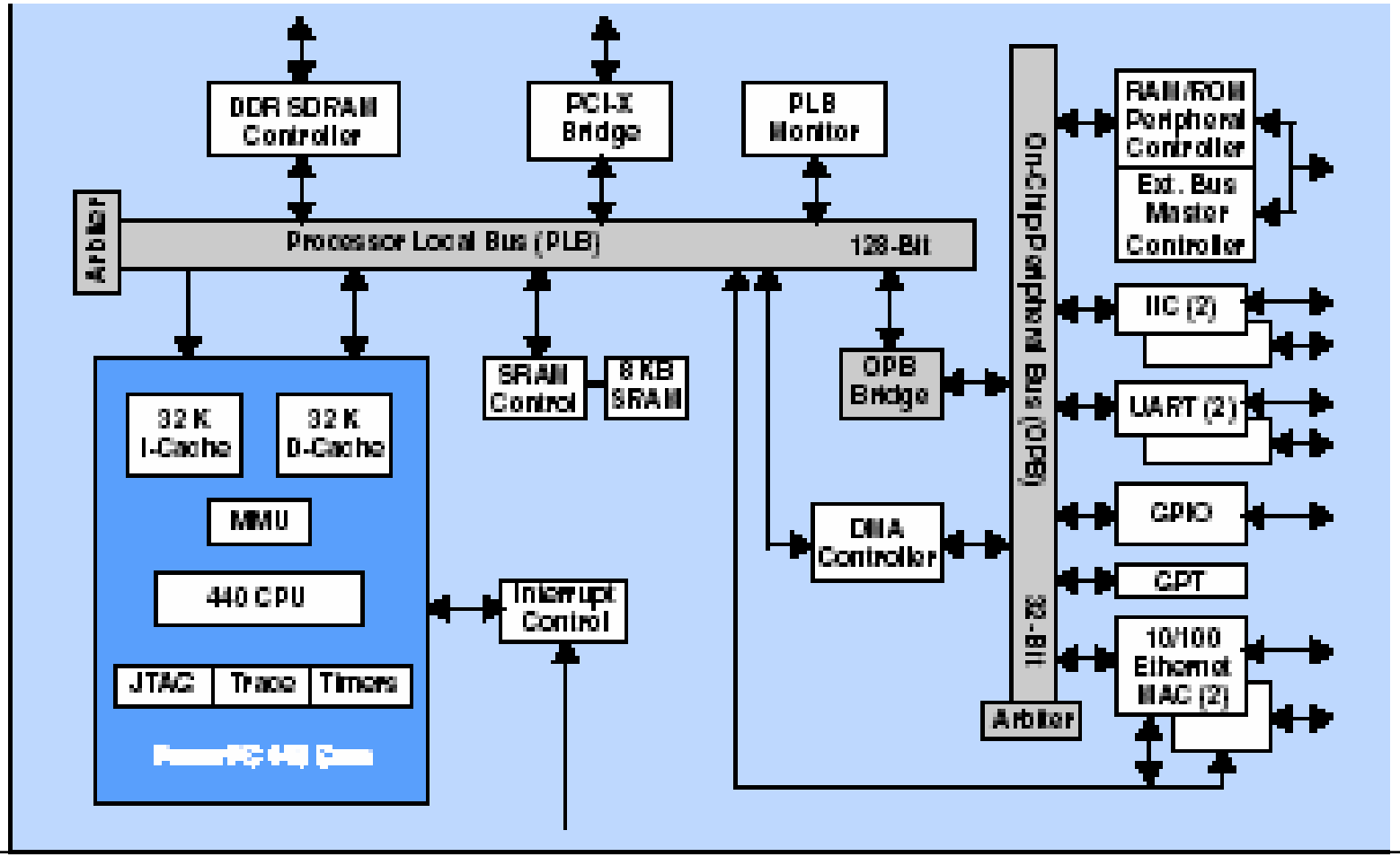
❑ Processeur Gekko : GameCube

➢Gekko Chip:

- IBM PowerPC Architecture
- Customized 405 MHz processor
- 0.18 micron copper wire technology
- 32-bit integer
- 64-bit floating point performance

Panorama de processeurs

- **System on chip processeur :**
  - ➢Le power pc 440 GP embedded

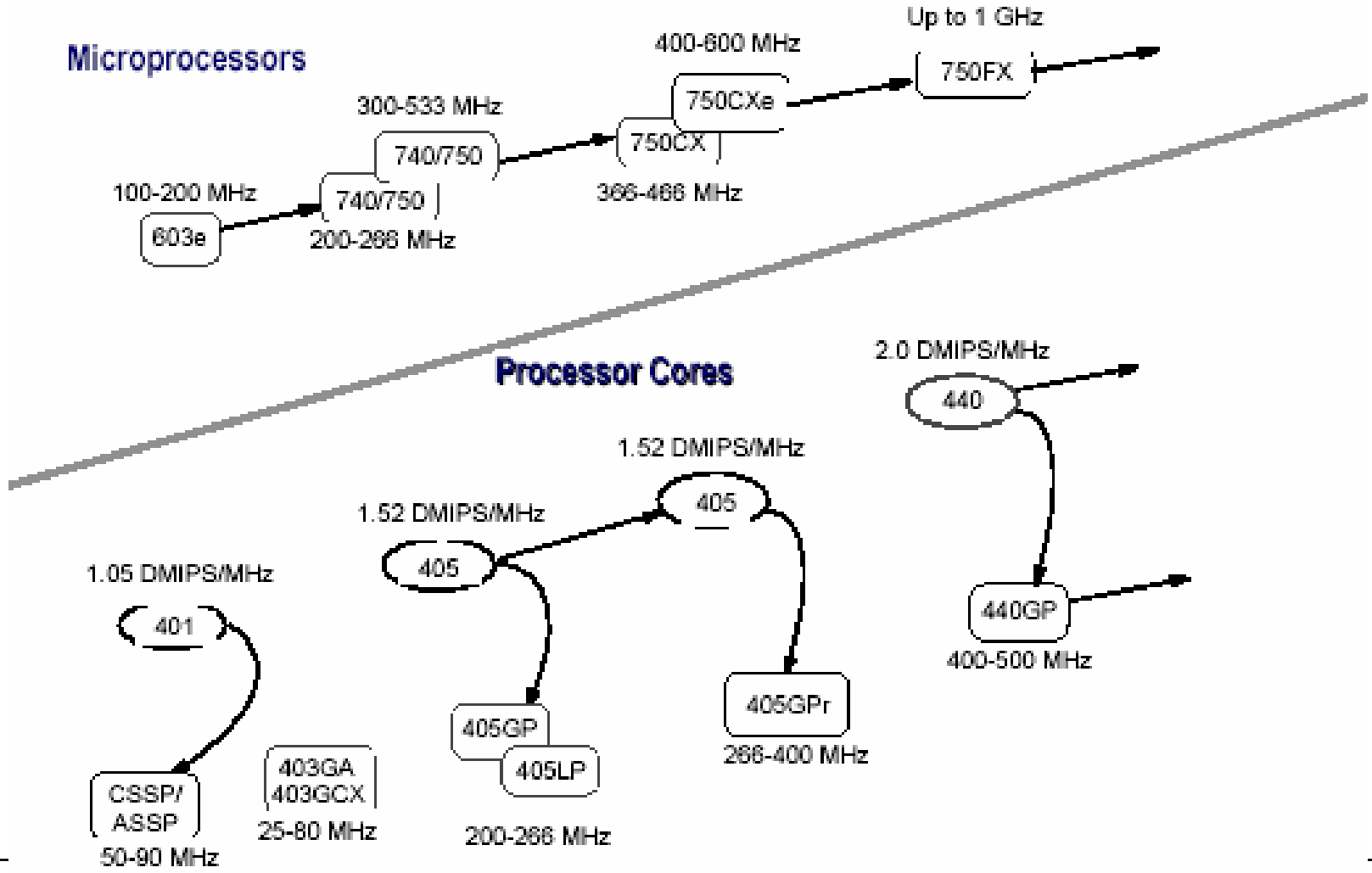


Panorama de processeurs

- Power PC 440 GP Embbeded
  - fréquence : 400 à 500 Mhz
  - 7 étages pipeline
  - Superscalaire de degré 2
  - cache instructions de 32 Ko, cache données de 32Ko
  - technologie : 0,18 micron
  - prédiction de branchement dynamique
  - 24 instructions de type DSP
  - mémoire adressable : 64 Go (36 bits d'adresses)
  - tensions d'alimentation :
    - 1,8 V pour la logique
    - 2,5 V pour la mémoire SDRAM
    - 3,3 V pour les I/O

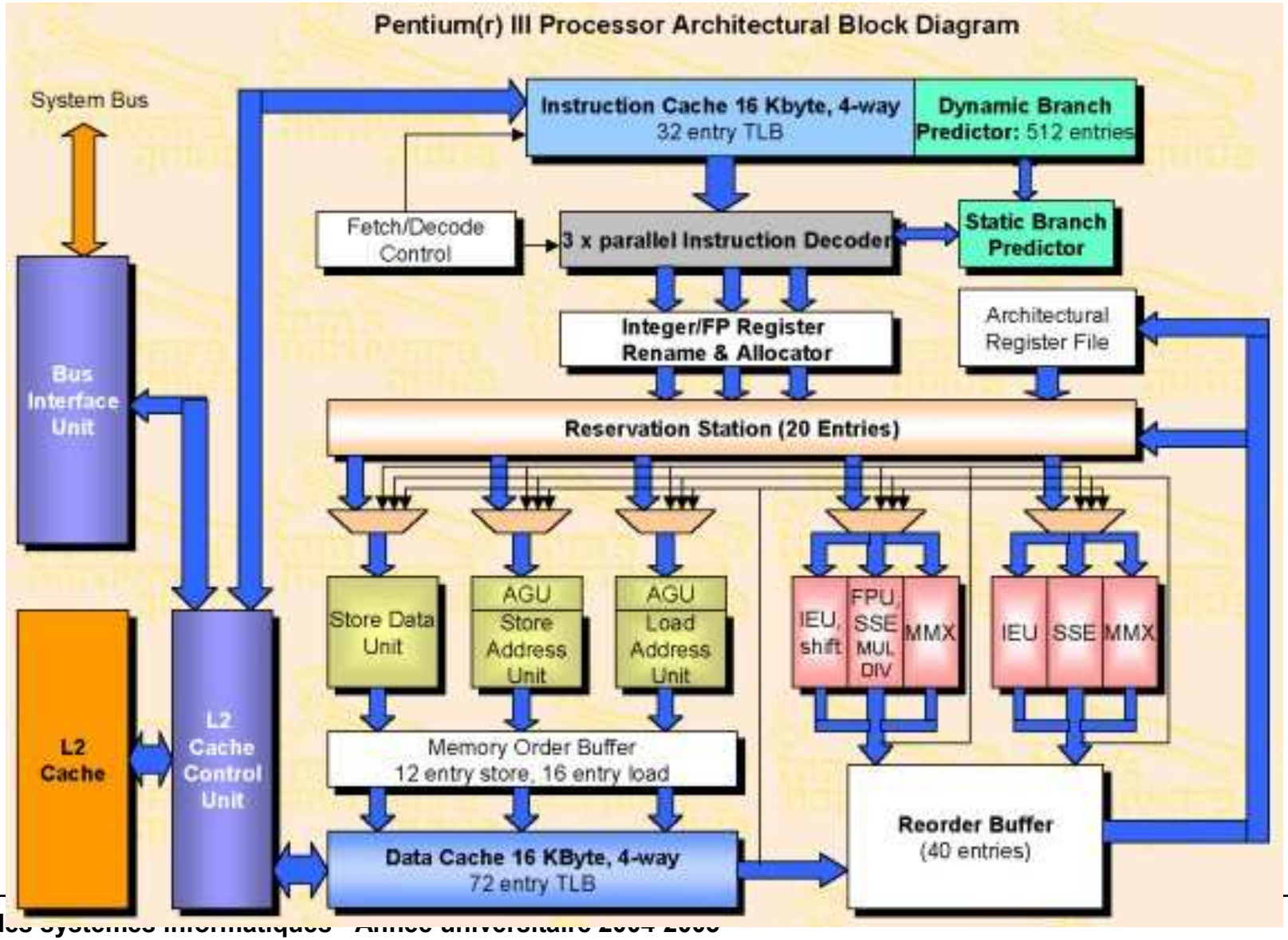
Panorama de processeurs

❑ Power Pc Road Map



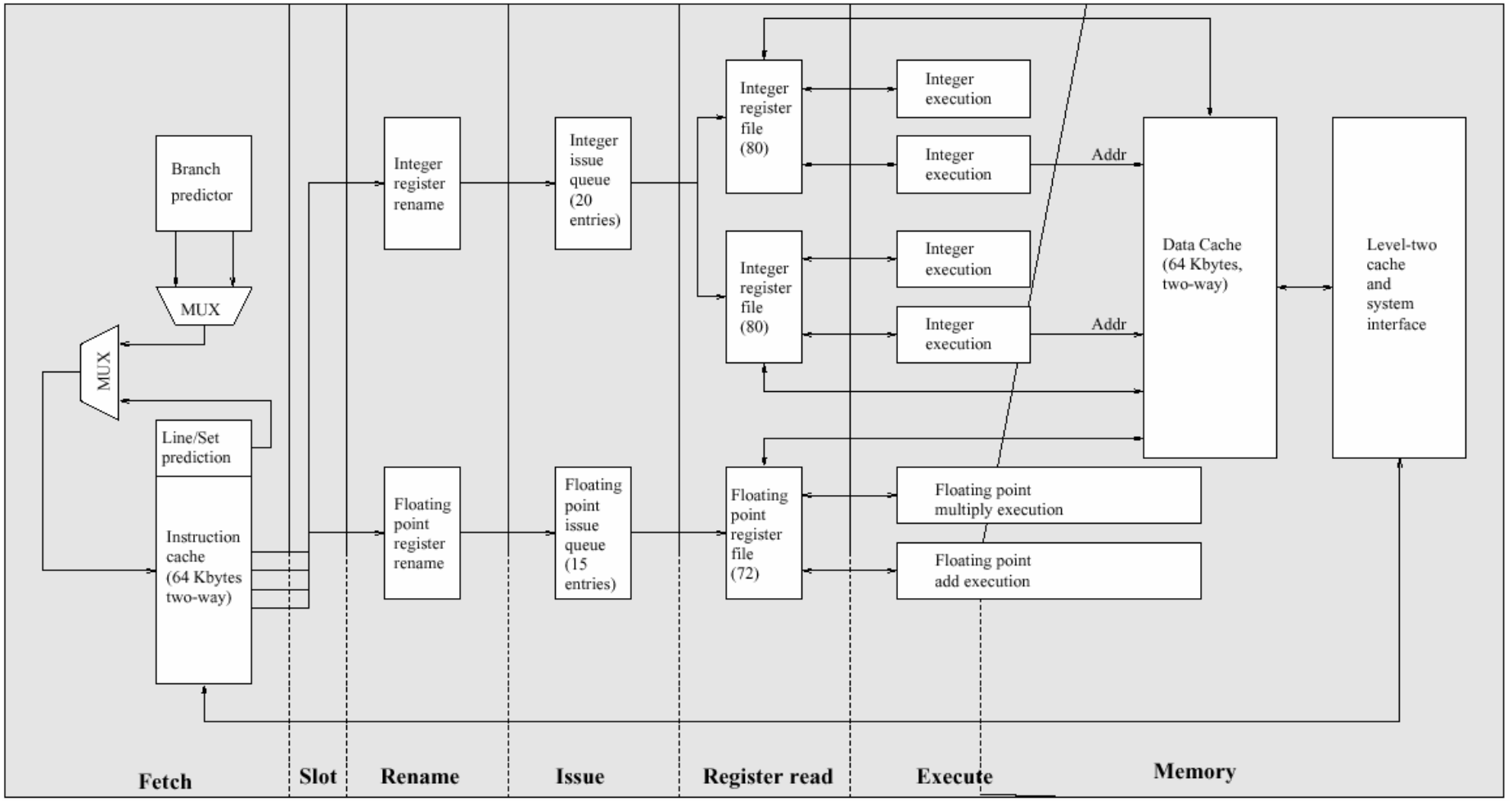
Panorama de processeurs

❑ Pentium III : architecture



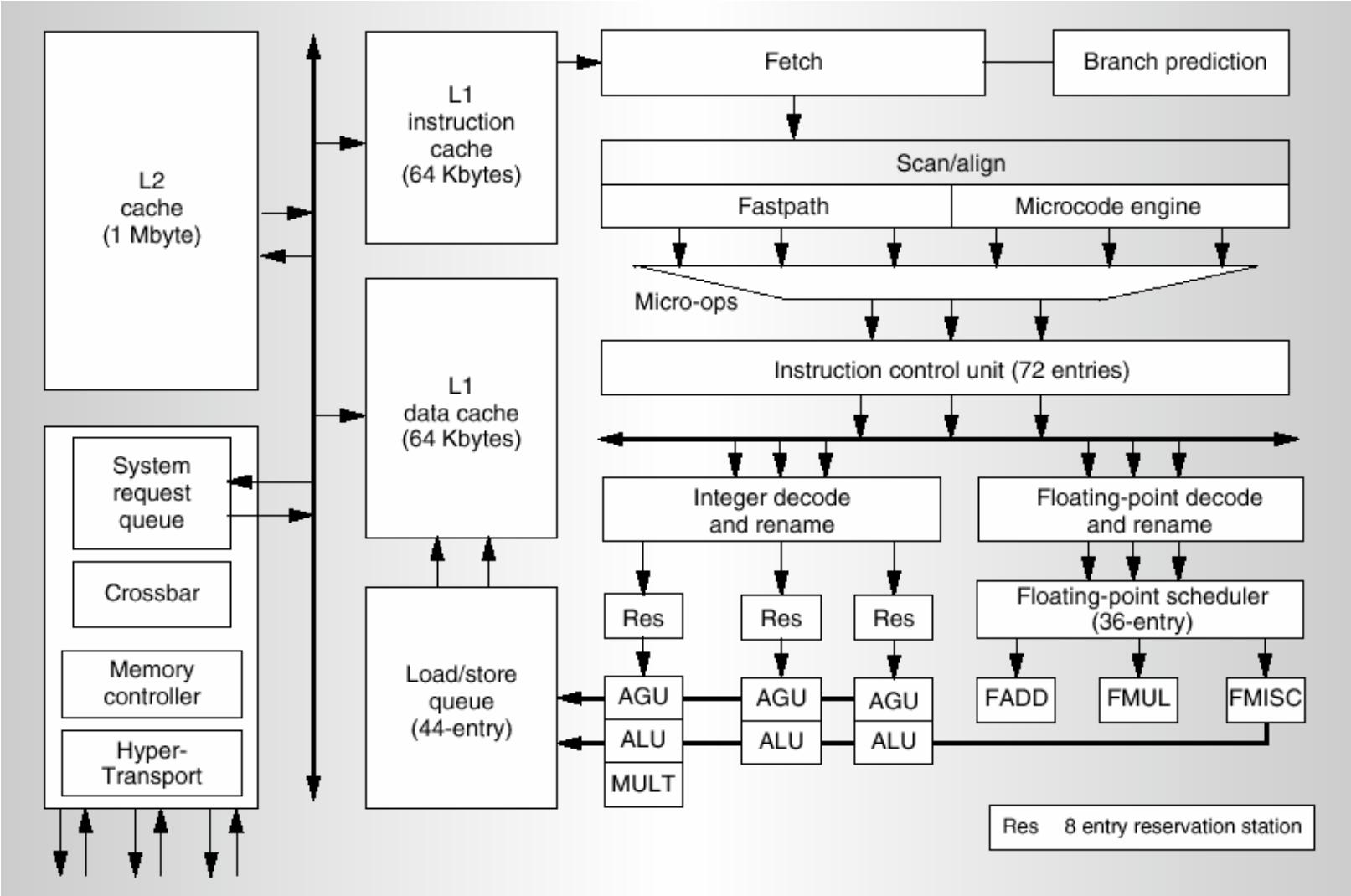
Panorama de processeurs

❑ Dec Alpha 21264 : architecture

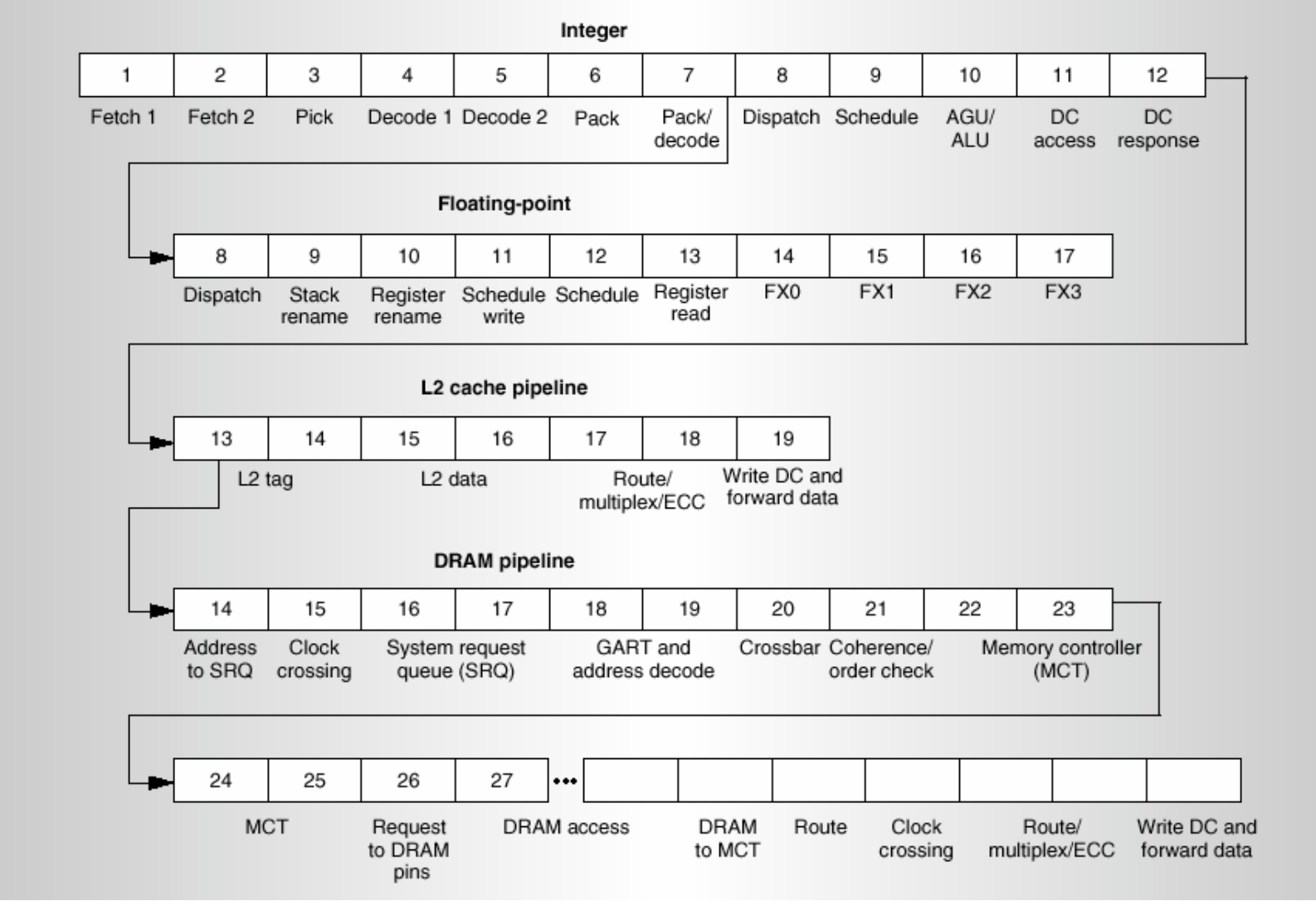


Panorama de processeurs

❑ L'Opteron de ADM

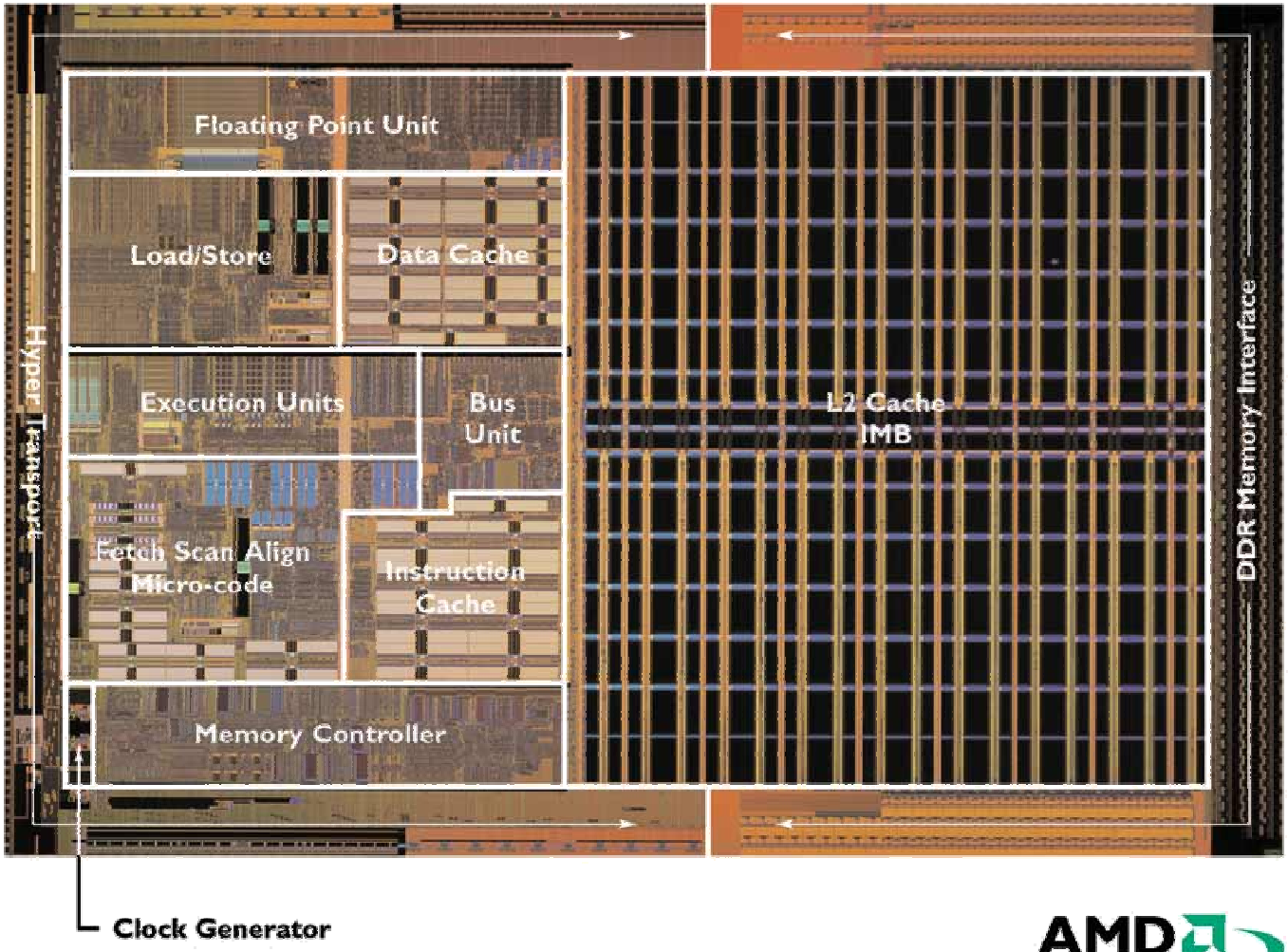


❑ Le pipeline de l'Opteron de ADM



Panorama de processeurs

Le Layout de l'Opteron



Panorama de processeurs

L'architecture IA 64 : Processeur Itanium :

- architecture développée conjointement entre Intel et Hewlett Packard
- c'est une machine RISC 64 bits
- la base de cette architecture est le processeur PA-RISC de chez HP
- capable de tourner du code IA 32, compatibilité oblige

Panorama de processeurs

➢ Problèmes avec le Pentium II :

- ♦ instructions de longueurs très variables
- ♦ énormément de formats différents
- ♦ les instructions peuvent référencer la mémoire

Conduit à un contrôleur très complexe

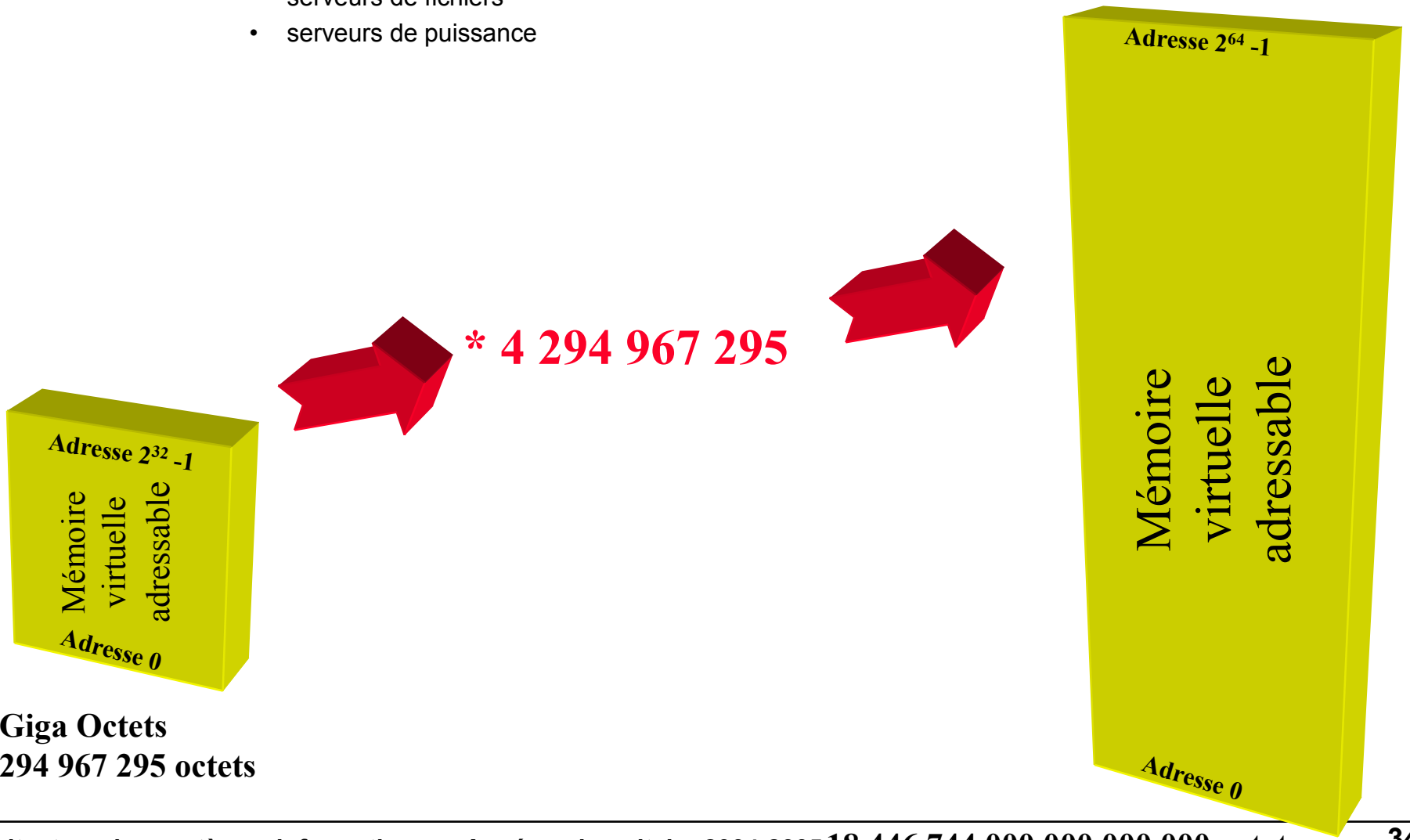
Grand nombre de transistors pour le décodage, le contrôle du pipeline

Alors qu'on pourrait utiliser la place pour d'autres choses, par exemple un cache L1 plus grand

- ♦ Pour arriver à tout faire rapidement :
  - nécessite la mise en place d'un pipeline long :
    - pipeline sur 12 étages
    - dans ce cas il faut une très bonne prédiction de branchement faute de quoi un grand nombre de cycles sont perdus lors des mauvaises prédictions

Panorama de processeurs

- l'architecture IA 32, ne peut adresser "que" 4 Goctets de mémoire :
  - ça commence à faire juste
    - serveurs de fichiers
    - serveurs de puissance



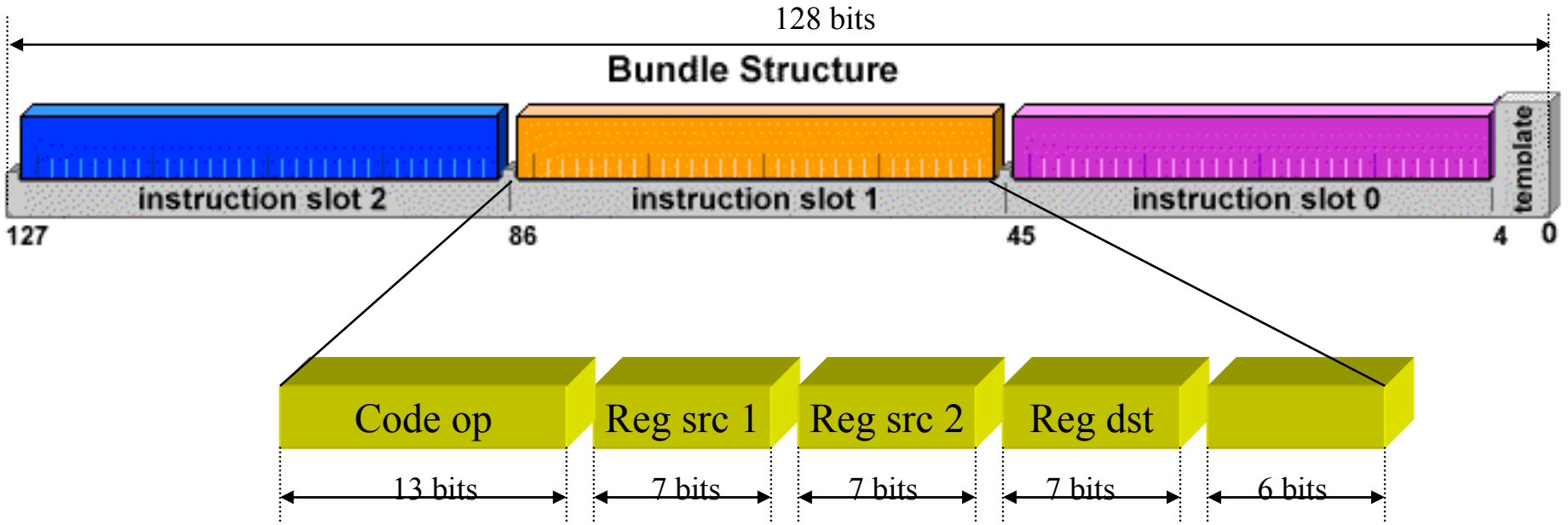
4 Giga Octets
4 294 967 295 octets

18 446 744 000 000 000 000 octets

Panorama de processeurs

➢Description de l'architecture :

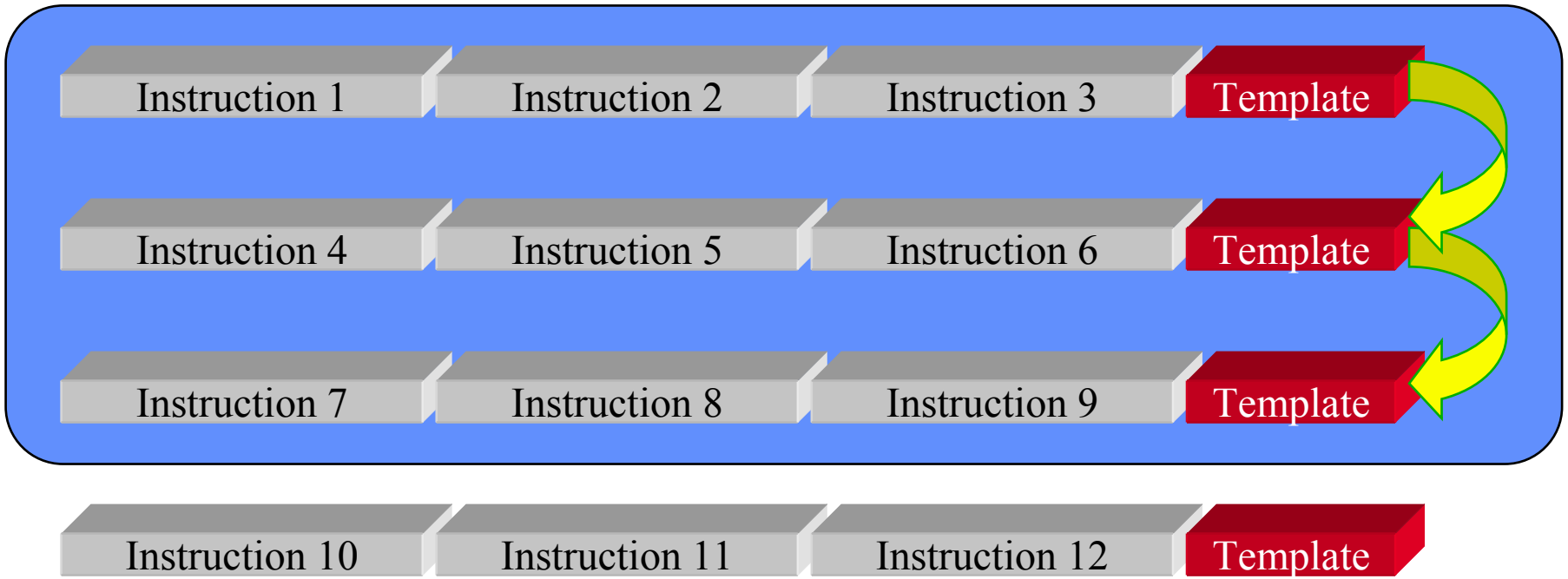
- ♦ 256 registres généraux 64 bits (128 entiers, 128 flottants)
- ♦ les instructions sont regroupées par 3, c'est un petit VLIW (LIV)



- ♦ les instructions peuvent comporter 3 opérandes :
  - – 2 opérandes sources, 1 opérande destination

Panorama de processeurs

- L'architecture définit des bundles :
  - ce sont des ensembles d'instructions traitées si possible ensemble
  - le champ template définit le parallélisme dans le groupe de 3 instructions
  - un bit du template indique si les bundles sont chaînés

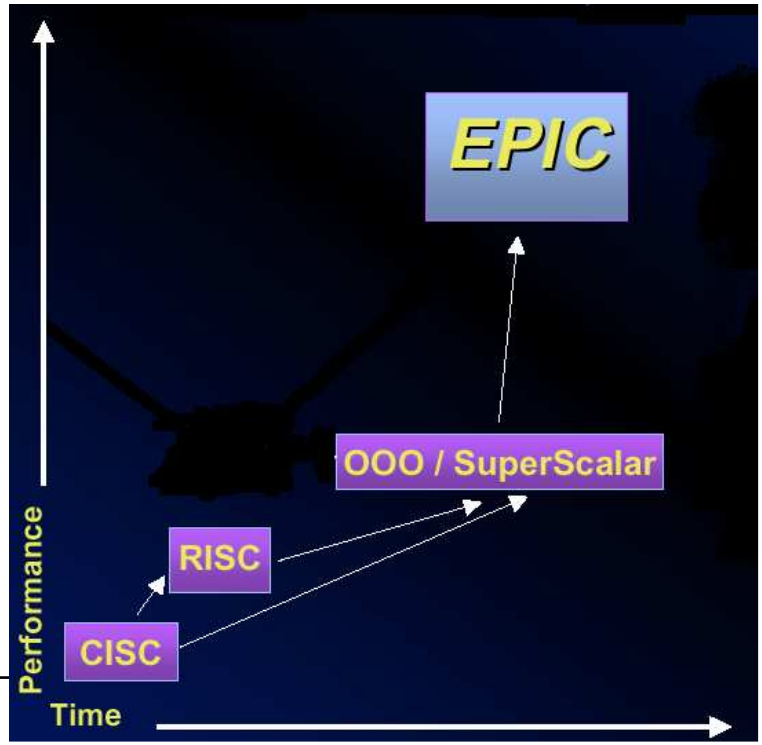


Panorama de processeurs

♦ Rôle du compilateur :

- regroupement des instructions de façon à exploiter le maximum de parallélisme
- toute la complexité est renvoyée vers le logiciel, ce qui décharge d'autant le matériel :
  - la surface peut alors être consacrée à autre chose
- modèle EPIC : Explicitly Parallel Instruction Computing

- actuellement une grosse partie de la complexité du processeur (superscalaire) est utilisée pour l'exécution dans le désordre



Panorama de processeurs

- 2 types de processeurs :
  - low-end CPU :
    - le processeur peut traiter 1 seul bundle par cycle
    - on ne passe au bundle suivant que lorsque le bundle courant est terminé
  - high-end CPU :
    - le processeur peut traiter plusieurs bundles par cycle :
      - prise en compte du chaînage des bundles
    - le processeur peut lancer un nouveau bundle alors que les précédents ne sont pas terminés :
      - type d'exécution superscalaire

Panorama de processeurs

♦ Parallélisme adapté à la machine :

– le champ template définit quelles opérations peuvent s'exécuter en parallèle indépendamment de la machine :

• exemple : le template indique que les instructions I1, I2, I3, I4 peuvent s'exécuter en parallèle

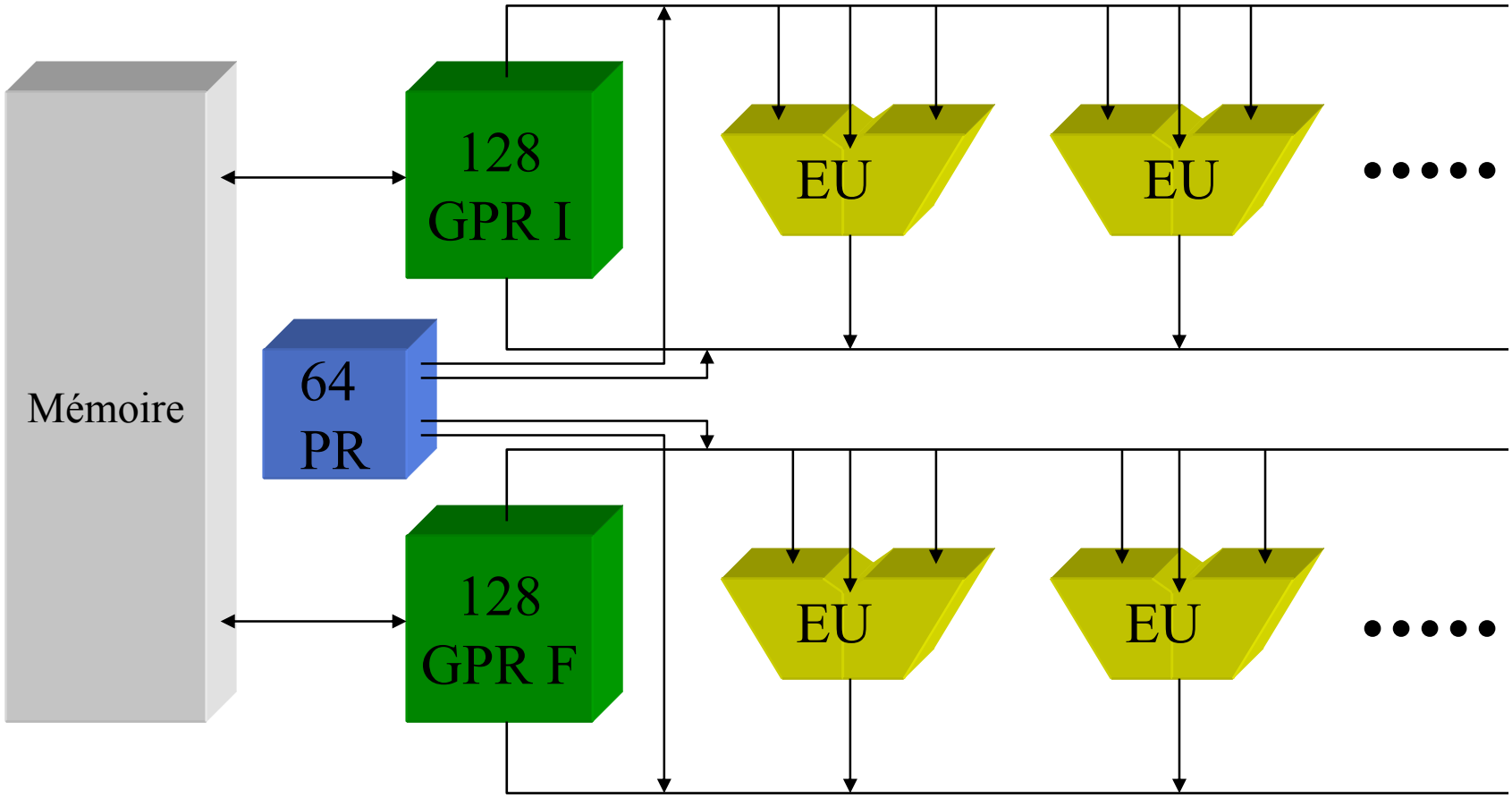
– en fonction du nombre d'unités fonctionnelles disponibles la machine lancera 1, 2, 3 ou 4 instructions en parallèle :

• exemple : si la machine possède 2 unités fonctionnelles, alors 2 instructions seront lancées

• si un nouveau Itanium est développé avec 4 unités fonctionnelles, alors le processeur lancera 4 instructions en parallèle

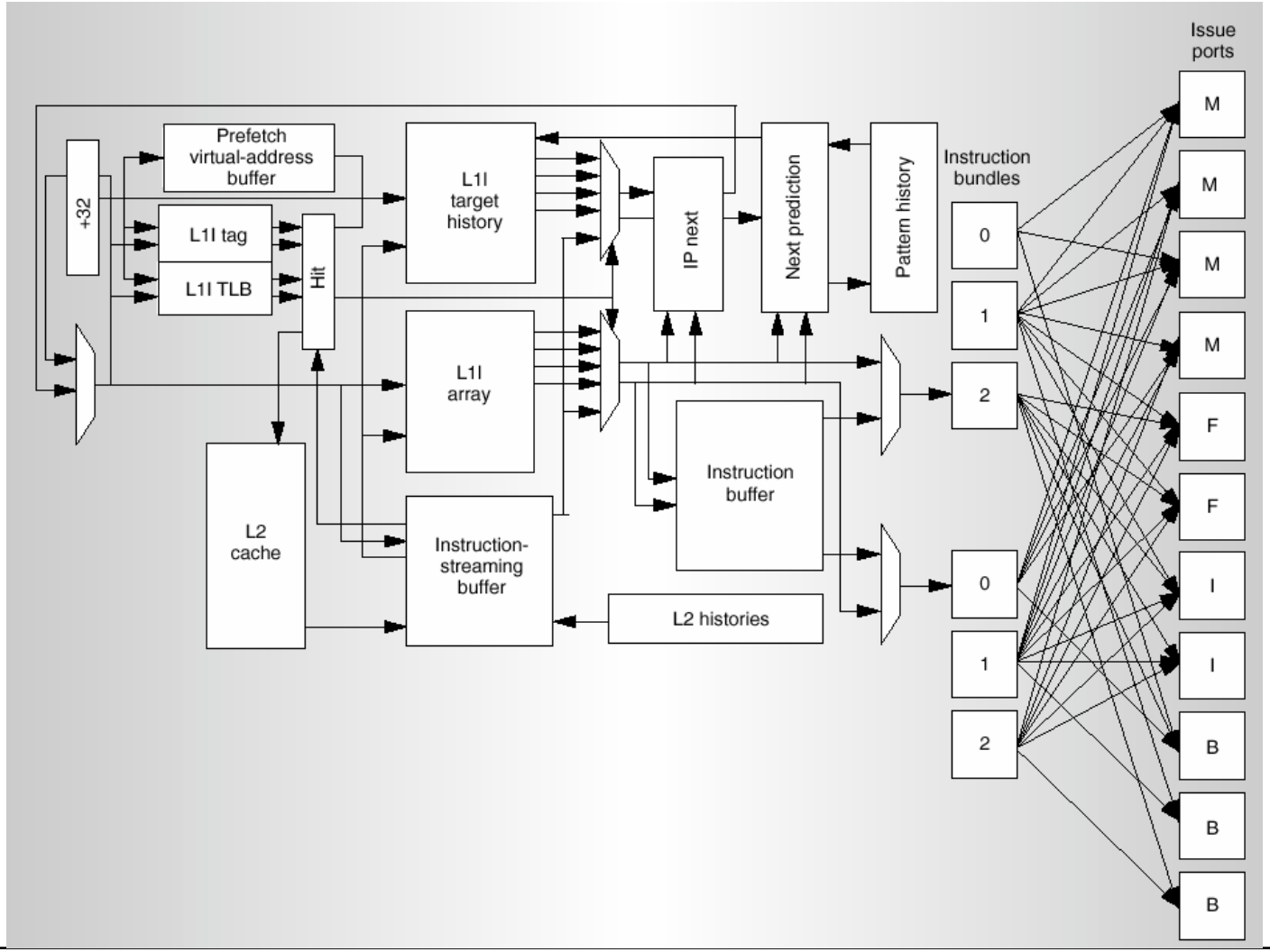
Panorama de processeurs

➢Exemple d'une version de processeur Itanium



Panorama de processeurs

➢Front end (Itanium 2) permettant de dispatcher les instructions

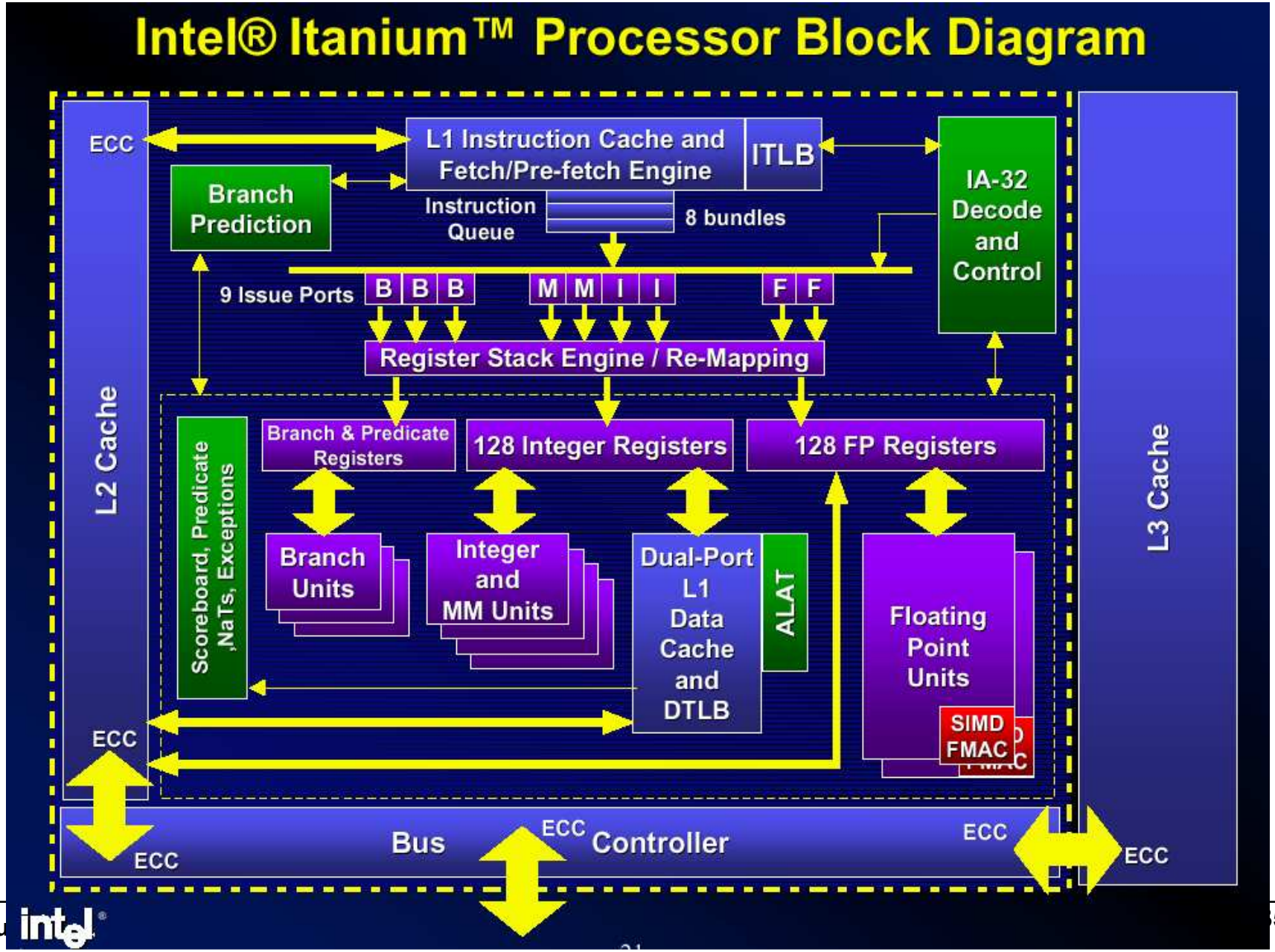


Panorama de processeurs

➢ Le pipeline du processeur Itanium :

♦ 10 étages

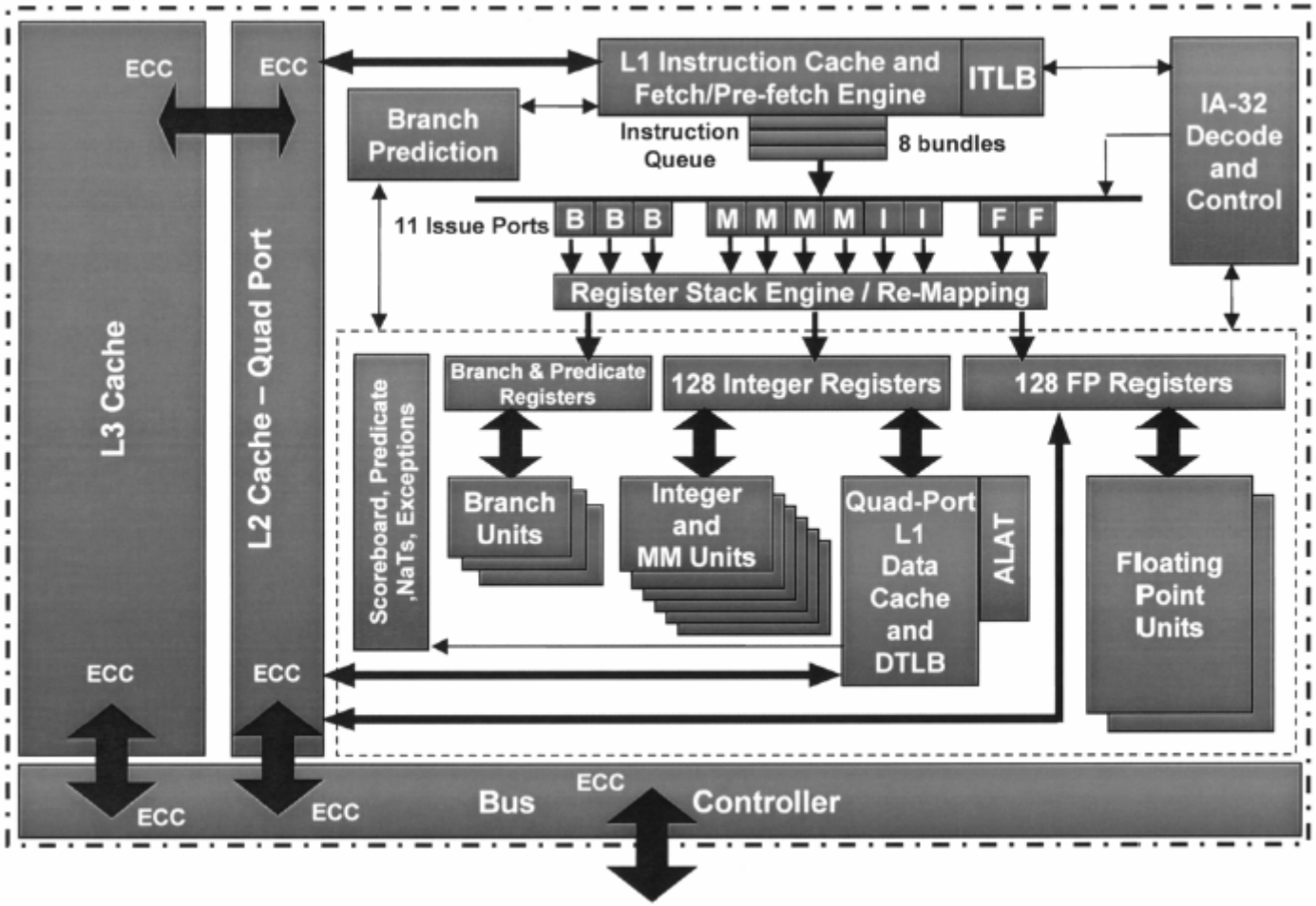
Panorama de processeurs



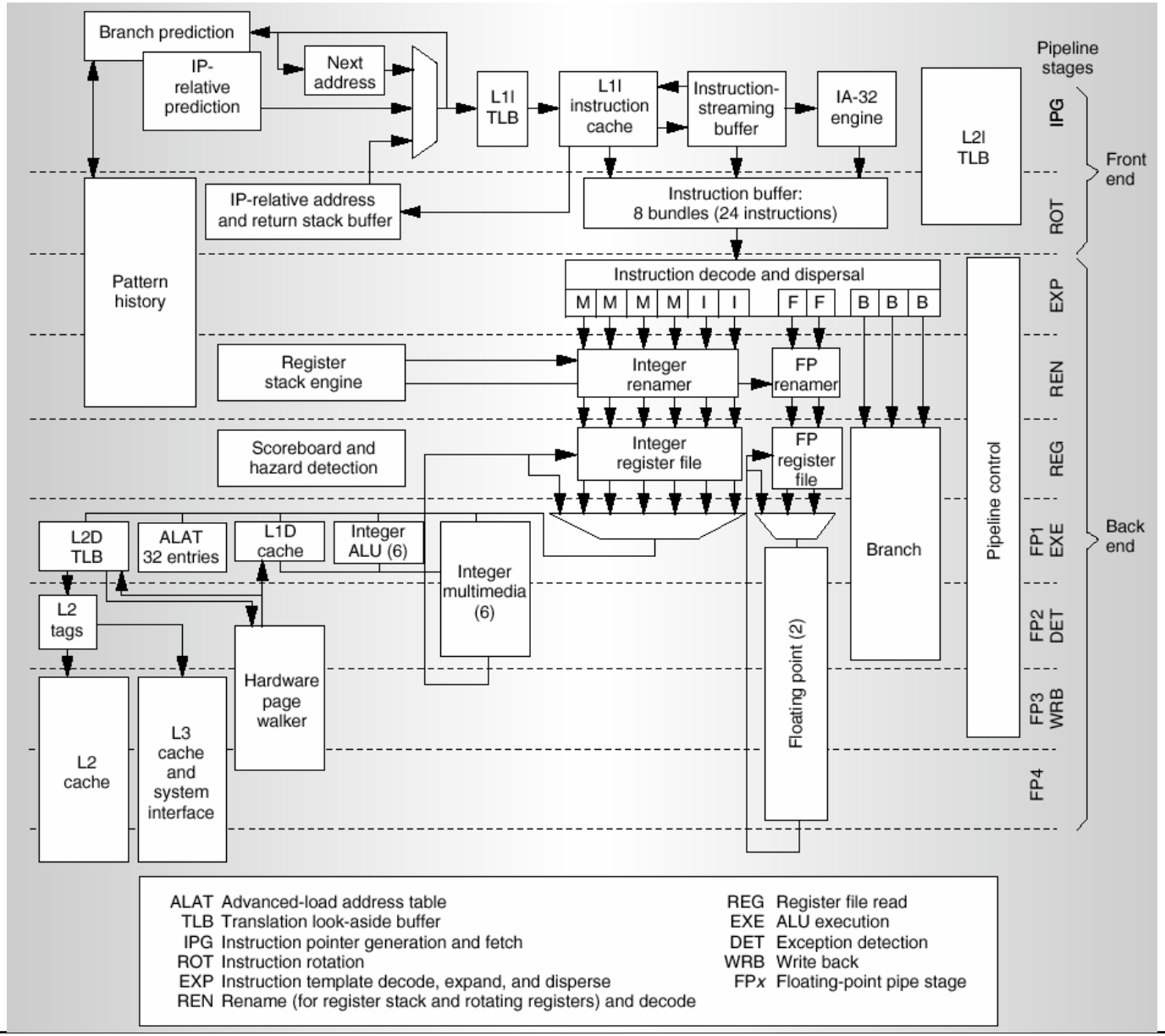
Panorama de processeurs

➢Exemple d'une version de processeur Itanium :

♦ Itanium 2



Panorama de processeurs



Panorama de processeurs

➢ Limitation des pertes de cycles :

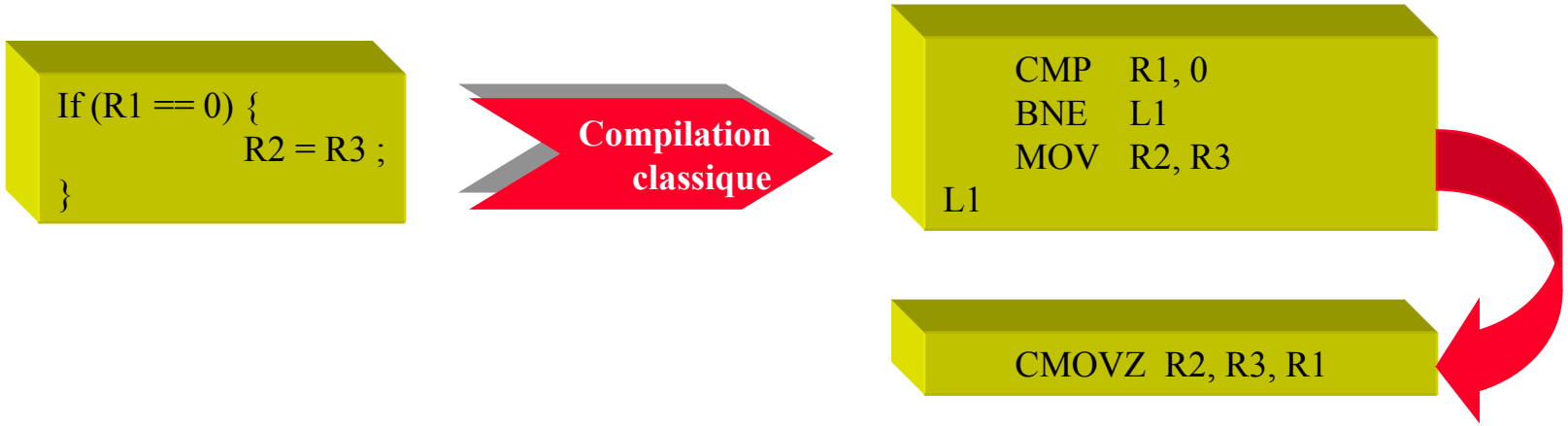
♦ la longueur du pipeline impose :

– soit une très bonne prédiction de branchement (intel estime que 20 à 30 % de la puissance de calcul des processeurs est consommée par les mauvaises prédictions)

– une autre solution consiste à supprimer certains branchements

– Exemple :

• soit le code C suivant :



Panorama de processeurs

– Autre exemples :

```
If (R1 == 0) {
        R2 = R3 ;
        R4 = R5 ;
} else {
        R6 = R7 ;
        R8 = R9 ;
}
```

Compilation classique

```
        CMP    R1, 0
        BNE    L1
        MOV    R2, R3
        MOV    R4, R5
        BR     L2
L1
        MOV R6, R7
        MOV R8, R9
L2
```

```
        CMOVZ   R2, R3, R1
        CMOVZ   R4, R5, R1
        CMOVN   R6, R7, R1
        CMOVN   R8, R9, R1
```

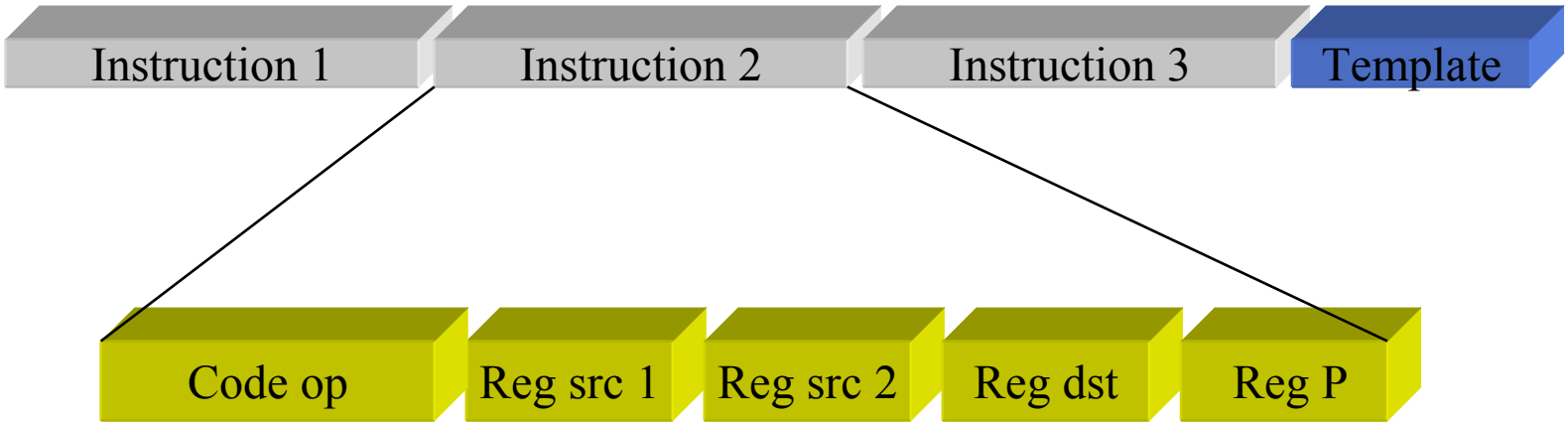
– Avantages :
  - plus d'instructions de branchements conditionnels
  - pas de réamorçage de pipeline
  - code plus compact

– Inconvénients :
  - deux instructions vont se comporter comme des NOP
  - pas utilisable dans tous les cas, mais quand même

– Remarque : les deux branches de la structure conditionnelles sont exécutées, mais une seule est retenue

Panorama de processeurs

- Le registre de "test" :
  - predicate register
  - est codé dans les 6 dernier bits



Panorama de processeurs

- Dans le processeur Itanium, toutes les instructions sont conditionnelles :
  - les instructions de comparaison positionnent les registres Predicate
  - exemple, soit le code :

P1, P2 = cmp (A == 0)

P1 = (A == 0)
P2 = (A != 0)

A	P1	P2
0	1	0
autre	0	1

Panorama de processeurs

♦ Utilisation de ces instructions :

```
If (a && b)
                j = j +1 ;
else
                if ( c )
                                k = k + 1 ;
                else
                                k = k -1 ;
i = i + 1 ;
```

Compilation classique

```
        beq     a, 0, L1
        beq     b, 0, L1
        add     j, j, 1
        jump    L3
L1
        beq     c, 0, L2
        add     k, k, 1
        jump    L3
L2
        sub     k, k, 1
L3
        add     i, i, 1
```

```
                P1, P2  = cmp   (a == 0)
<P2>            P1, P3  = cmp   (b == 0)
<P3>                    add     j, j, 1
<P1>            P4, P5  = cmp   (c != 0)
<P4>                    add     k, k, 1
<P5>                    sub     k, k, 1
                        add     i, i, 1
```

– Remarque : les deux branches de la structure conditionnelles sont exécutées, mais une seule est retenue

Panorama de processeurs

➢Format général des instructions :

1)	<Pi>			instruction
2)		Pj, Pk	= cmp	(relation)
3)	<Pi>	Pj, Pk	= cmp	(relation)

- ♦ format 1 :
  - instruction conditionnelle, l'instruction sera exécutée si et seulement si Pi est vraie :
    - en faite l'instruction s'exécute mais l'étage d'écriture de résultat est inhibé si la condition est fausse
- ♦ format 2 :
  - la comparaison positionne les registres *predicate* :
    - Pj = vraie si la relation est vraie, à faux sinon
    - Pk = faux si la relation est vraie, à vraie sinon
- ♦ format 3 :
  - la comparaison positionne les registres predicate si et seulement si Pi est vraie

Panorama de processeurs

- Les chargements spéculatifs :
  - ils ont pour rôle de demander la donnée à la mémoire avant que celle-ci ne soit éventuellement utiliser
- Un Load spéculatif provoque :
  - une fetch mémoire : le fetch peut avoir deux conséquences :
    - si le load échoue
- L'instruction Check vérifie la disponibilité de la donnée :
  - si la donnée est disponible alors le check se comporte comme une instruction nop
  - si la donnée n'est pas disponible, alors l'exception est exécutée

Panorama de processeurs

♦ Exemple :

```
If (b[j] == true) && (a[i+j] == true) && (c[i-j+7] == true)) then
                .....
else
                .....
end
```

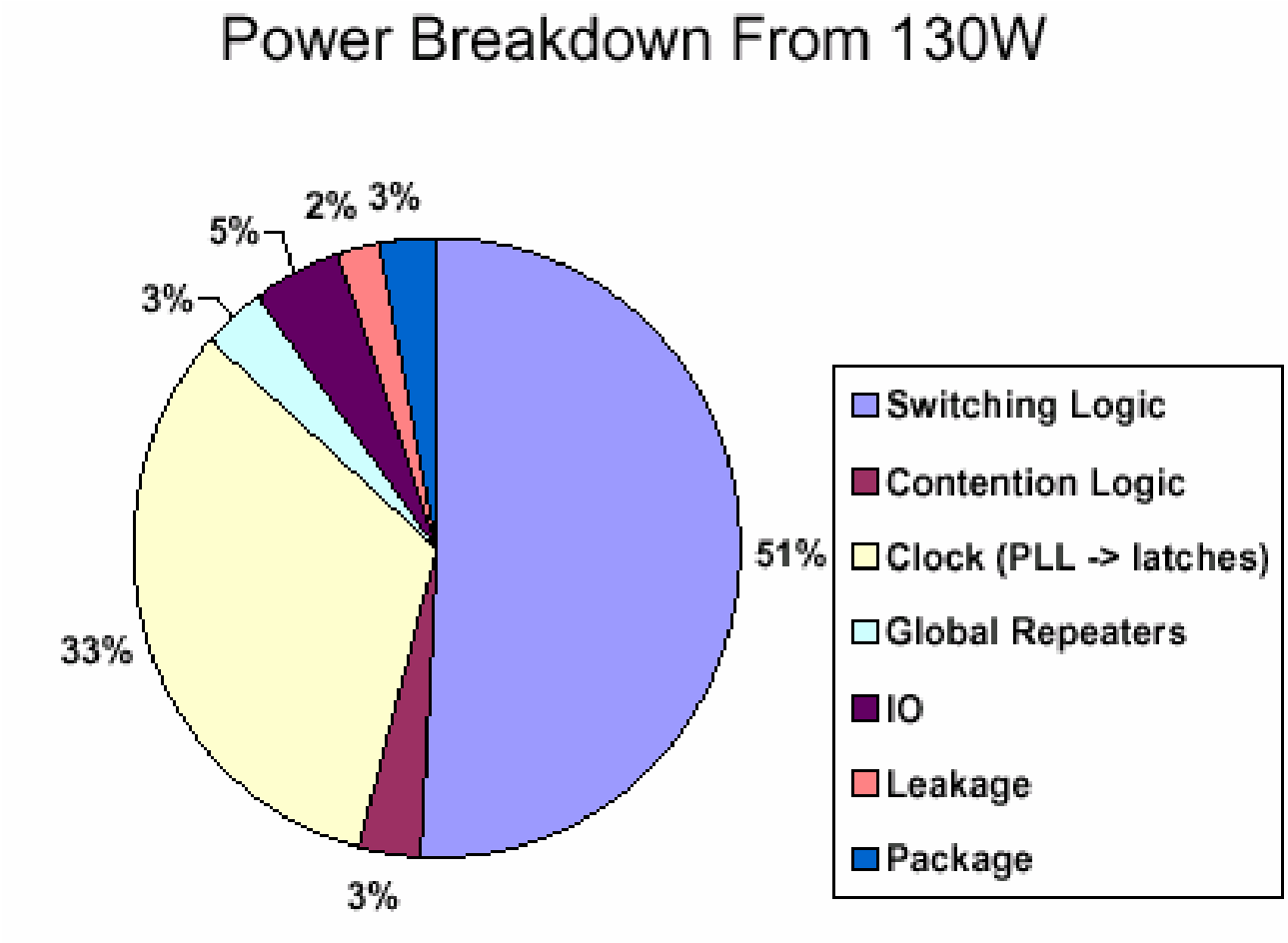
Compilation classique

```
        R1 = &b[j]
        ld  R2, (R1)
        bne R2, 1, L2
        R3 = &a[i+j]
        ld R4, (R3)
        bne R4, 1, L2
        R5 = &c[i-j+7]
        ld R6, (R5)
        bne R6, 1, L2
L1      .....
        ....
L2
        .....
        .....
```

```
                        R1 = &b[j]
                        R3 = &a[i + j]
                        R5 = &c[i - j + 7]
                        ld      R2, (R1)
                        ld.s    R4, (R3)
                        ld.s    R6, (R5)
                P1, P3  = cmp   (R2 == 1)
        <P1>            chk.s R4
        <P1>    P3, P4  = cmp   (R4 == 1)
        <P3>            chk.s R6
        <P3>    P5, P6  = cmp   (R5 == 1)
        <P6>            br L2
L1                      .......
L2                      .......
```

Panorama de processeurs

♦ Remarque : la consommation dans le processeur …



Roadmap INTEL

Panorama de processeurs

Les processeurs multi média

- Les objectifs : inclure l'ensemble des besoins multi média dans un seul boîtiers :
  - puissance de calcul : GSM, codage MPEG
  - haute résolution graphique (graphique 3D)
  - téléphonie, Fax, vidéo phone
  - courrier électronique
  - portable, donc faible consommation (3,3 volts)
- Applications concernées :
  - communication spécialisées transmission :
    - capture d'images et de sons : graphique, animation, photo numérique, caméra VT, TV
    - transmission : hertzien terrestre, satellite, câble, ATM
    - restitution : écran, projecteur, haute définition

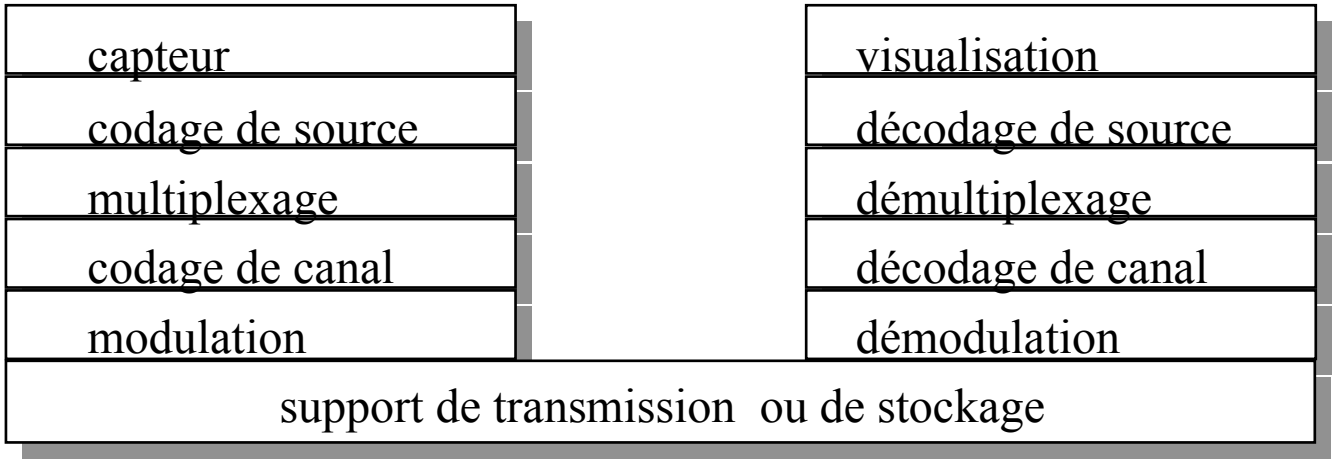
Panorama de processeurs

- stockage : RAM, disque dur, disque magnéto optique, CD rom, DAT

Convergence :

–audiovisuel	- communication
–informatique	- multi média globale,
–télécommunication	- intelligente, interactive

– Exemple :



Panorama de processeurs

Processeurs MMX

- pentium MMX : MultiMédia eXtensions
  - ajout d'instructions spécifiques au multi média (57 instructions supplémentaires)
  - augmentation de performances
  - ce n'est pas un nouveau processeur MAIS simplement une extension
  - les applications doivent être développées spécifiquement pour ce processeur (pour tirer le meilleur partit de ces capacités)
- DEC Alpha :
  - 13 instructions supplémentaires
  - augmentation de 0,6 % de la surface
  - fréquence d'horloge augmentée à 550 Mhz
  - codage MPEG 2 temps réel (complexité =
  - alimentation réduite (2.8 volts)
  - diminution de la consommation

Panorama de processeurs

- Ultra Sparc II :
  - 30 instructions spécifiques (Visual Instruction Set)
  - décodage MPEG2 temps réel
  - augmentation de la surface de 3 % du circuit
  - fréquence augmentée à 250 Mhz
  - 2,5 Gops en pic de traitement
  - tension d'alimentation à 2,6 volts
- MIPS V :
  - extension MMX
  - jusqu'à 8 multiplications 8 bits en parallèle
  - accumulateur sur 192 bits

Panorama de processeurs

Les *vrais* multi média

- Mpact :
  - puissance de calcul : 3 Bops (Gops)
  - fréquence de 125 MHz
  - graphique 2D et 3D
  - 5 ports d'entrées sorties
  - vidéo (VGA : million de couleurs, pixels codés sur 24 bits)
  - haute qualité audio (48 Khz), effets audio puissant (réverbération, atténuation du bruit, écho, etc), précision du contrôle
  - 8 voies simultanées d'enregistrement et d'écoute avec différents formats et différentes fréquences d'échantillonnages
  - interface MIDI (Musical Instrument Digital Interface)
  - Fax, Modem : supporte beaucoup de formats et de vitesse de transmission (jusqu'à 33600 bits par seconde)
  - téléphonie, vidéo phone : estimation de mouvements

Panorama de processeurs

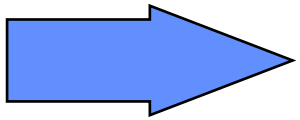
- MPEG temps réel : 30 images par seconde, codage 18 bits par pixel 352*240 (décodage MPEG 1 et 2, encodage MPEG 1) supporte windows 95
- faible tension (3,3 volts)
- accélération graphique :
  - dessins en fil de fer
  - génération de polygone
  - tracer de lignes
- instructions :
  - papillon FFT
  - mult, Add, Mad : algorithme de Booth, arbre de Wallace
- opérateurs :
  - 4 UALs
  - un bloc d'estimation de mouvements (pour le MPEG) :
    - 400 éléments arithmétique (pointe à 20 BOPS)

Panorama de processeurs

- Le Trimédia :
  - TM 1000 Philips
  - 4 milliard d'opérations par seconde (4 BOPS)
  - architecture VLIW :
    - 100 Mhz
    - jusqu'à 5 instructions par cycle
    - adressage sur 32 bits
    - 128 registres généraux sur 32 bits
    - 27 unités fonctionnelles
  - caches :
    - 32 Ko pour les instructions
    - 16 Ko pour les données
  - périphériques :
    - audio
    - vidéo
    - graphique

Panorama de processeurs

- Cœur du processeur : CPU temps réel (RTOS : real time operating system)
- développement en C ANSI ou en C++
- 8 canaux de sortie audio (100 Khz)
- 1 canal d'entrée audio 100 Khz
- 1 cana vidéo
- Coprocesseur MPEG 1 et 2
- supporte 37 opérations :
  - multimédia
  - DSP

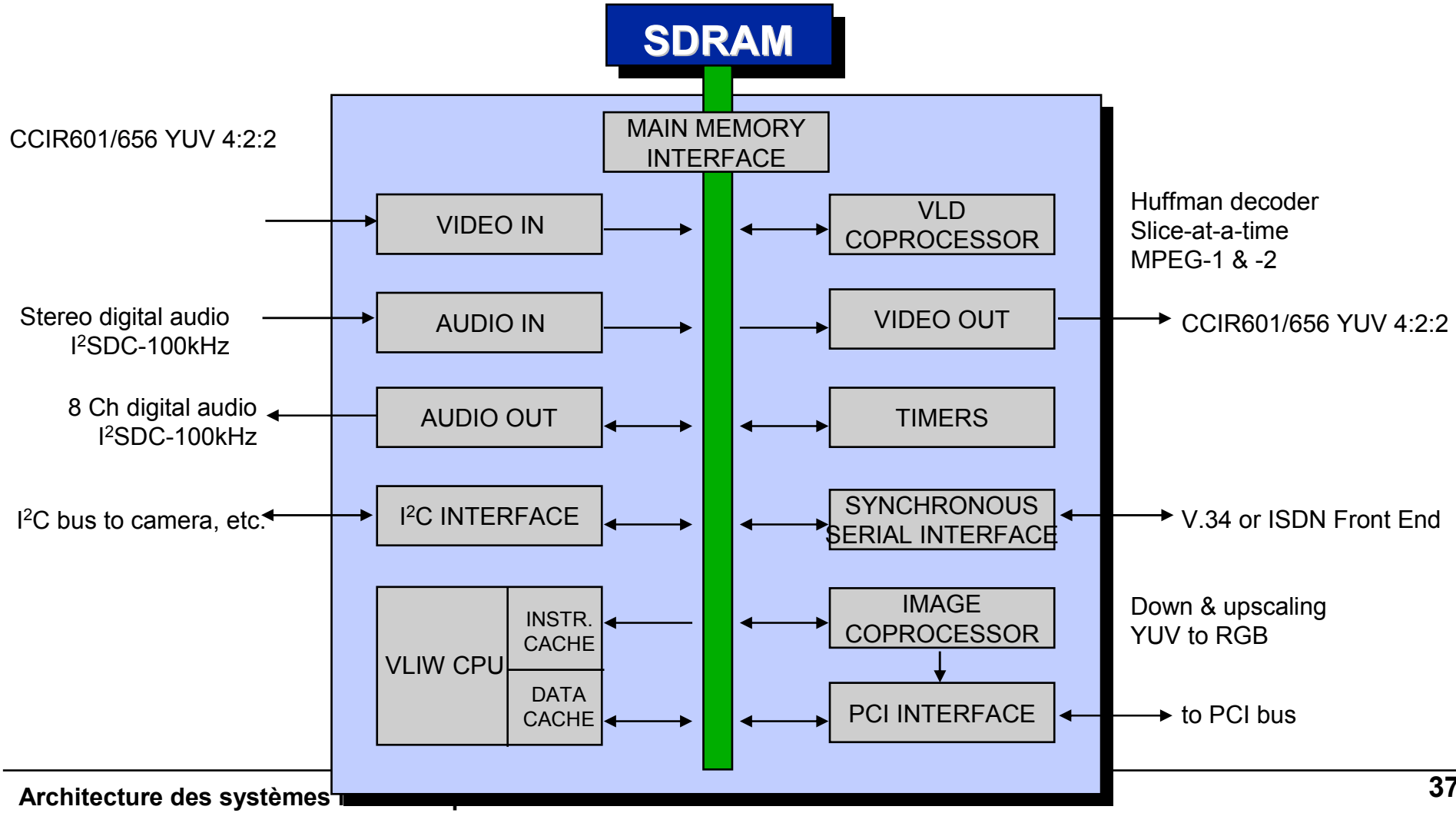


- estimation du mouvement
- valeur absolue
- addition
- multiplication
- moyenne
- etc

- gestion du parallélisme :
  - lors de la compilation : détection du parallélisme durant la compilation, ordonnancement des opérations durant la compilation
  - simplifie énormément la logique de gestion par rapport à un processeur qui évalue le parallélisme durant l'exécution

Panorama de processeurs

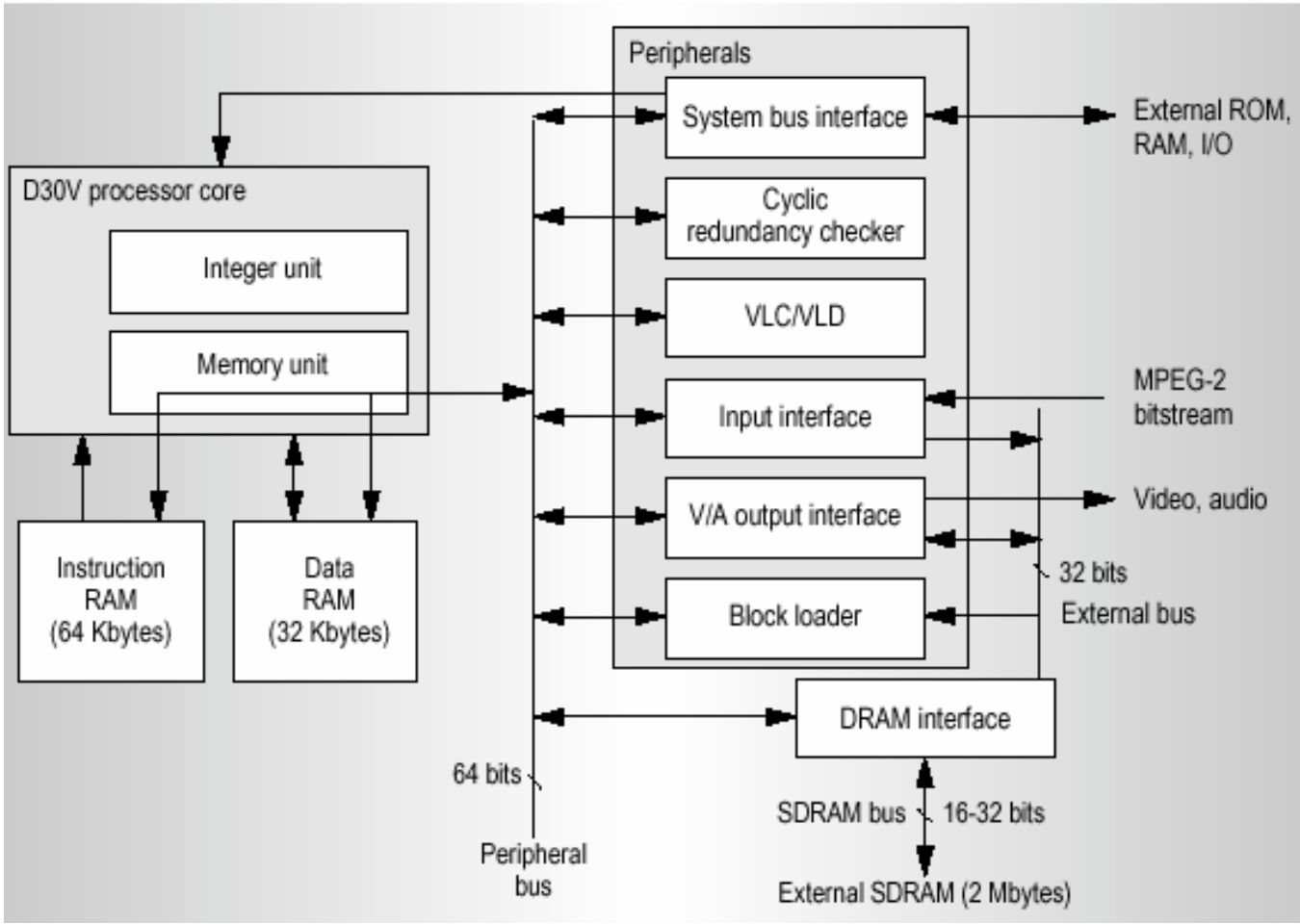
♦ Architecture du processeur Trimedia



Panorama de processeurs

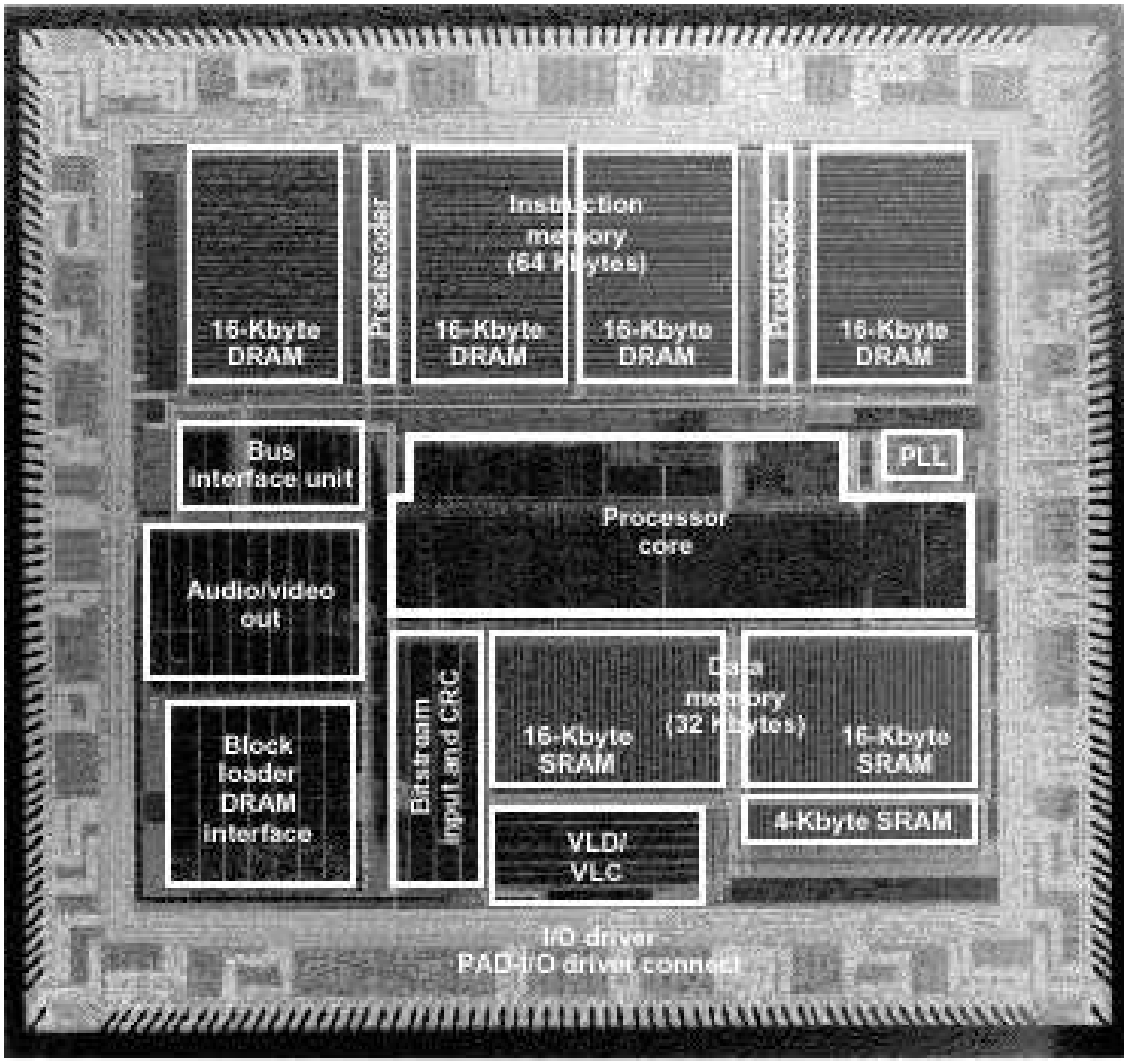
Le D30V/MPEG

Schéma bloc du processeur



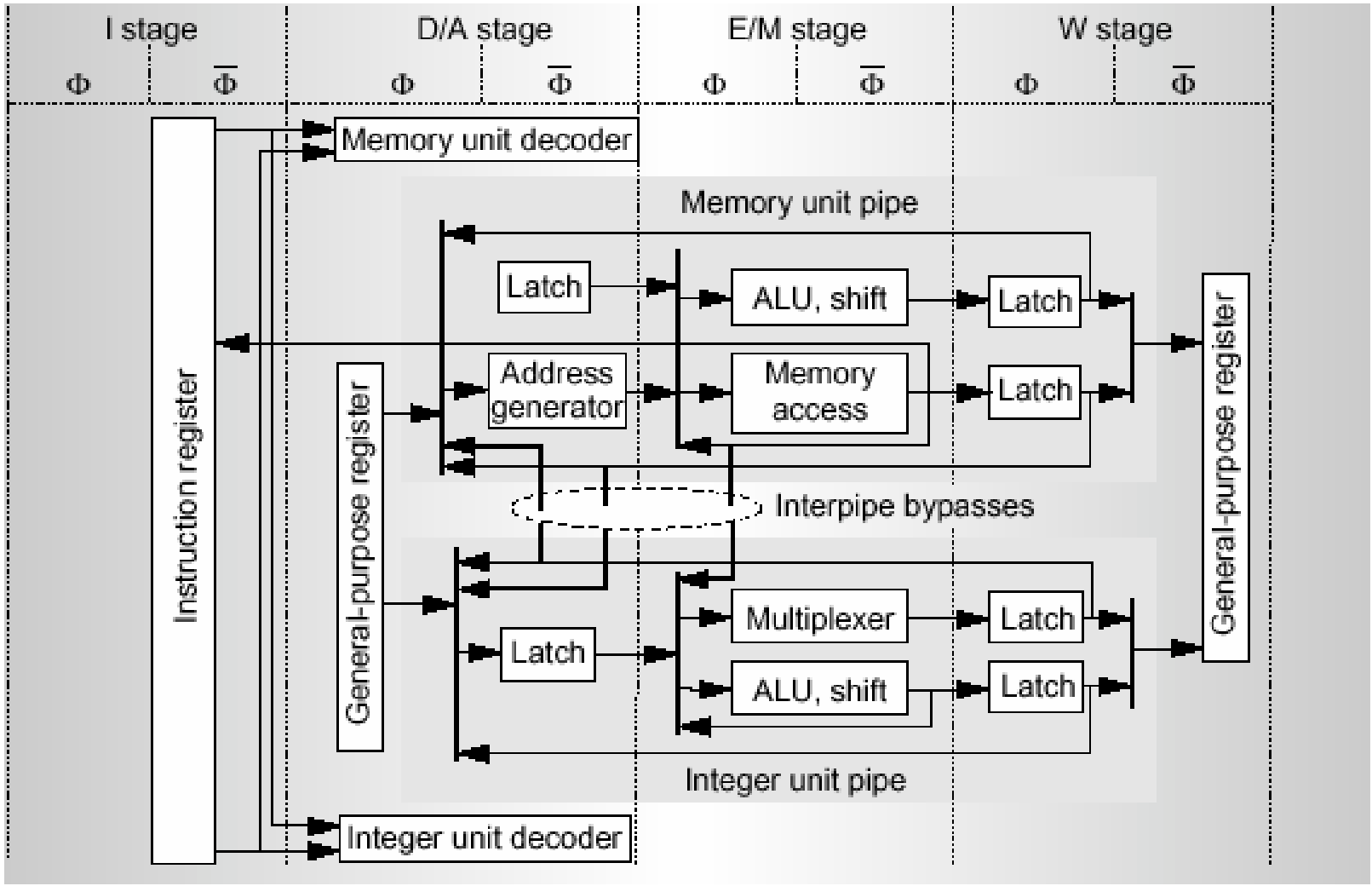
Panorama de processeurs

- **Le layout du D30V/MPEG**



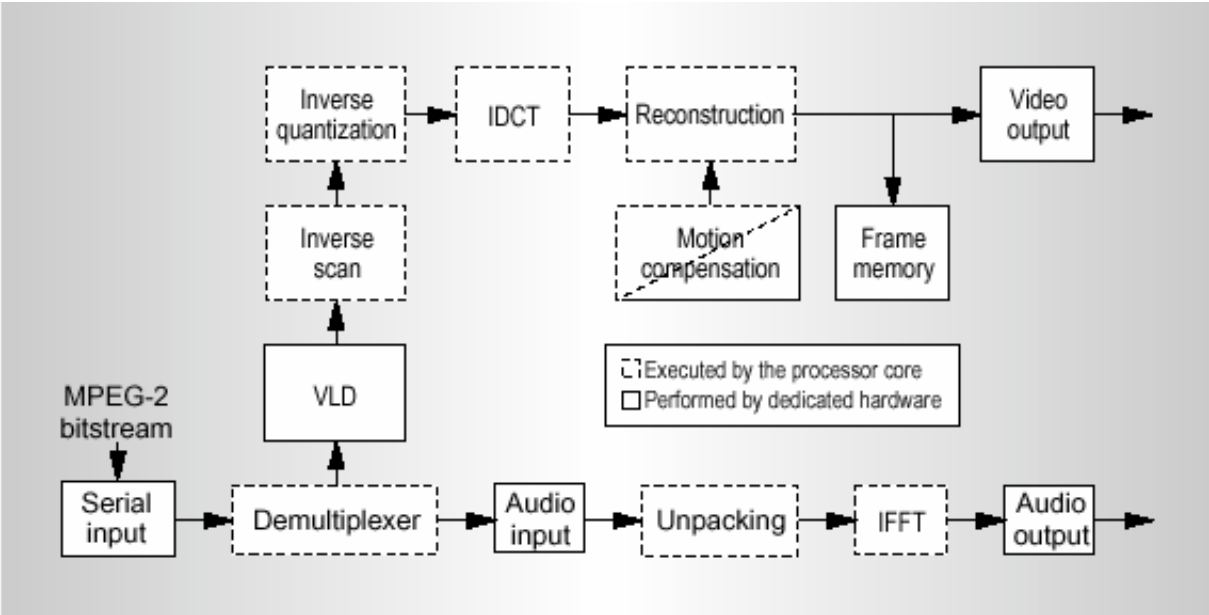
Panorama de processeurs

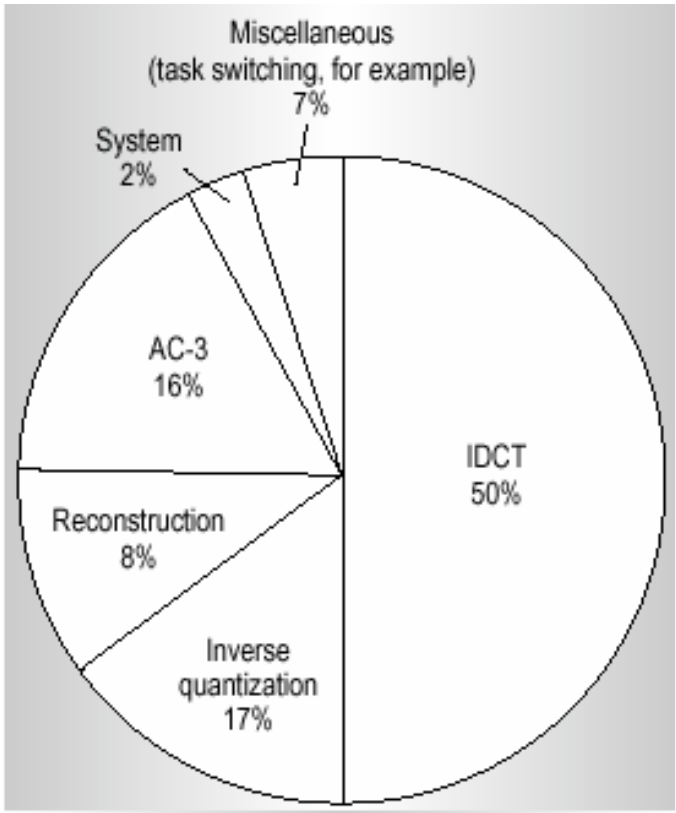
Le pipeline D30V/MPEG



Panorama de processeurs

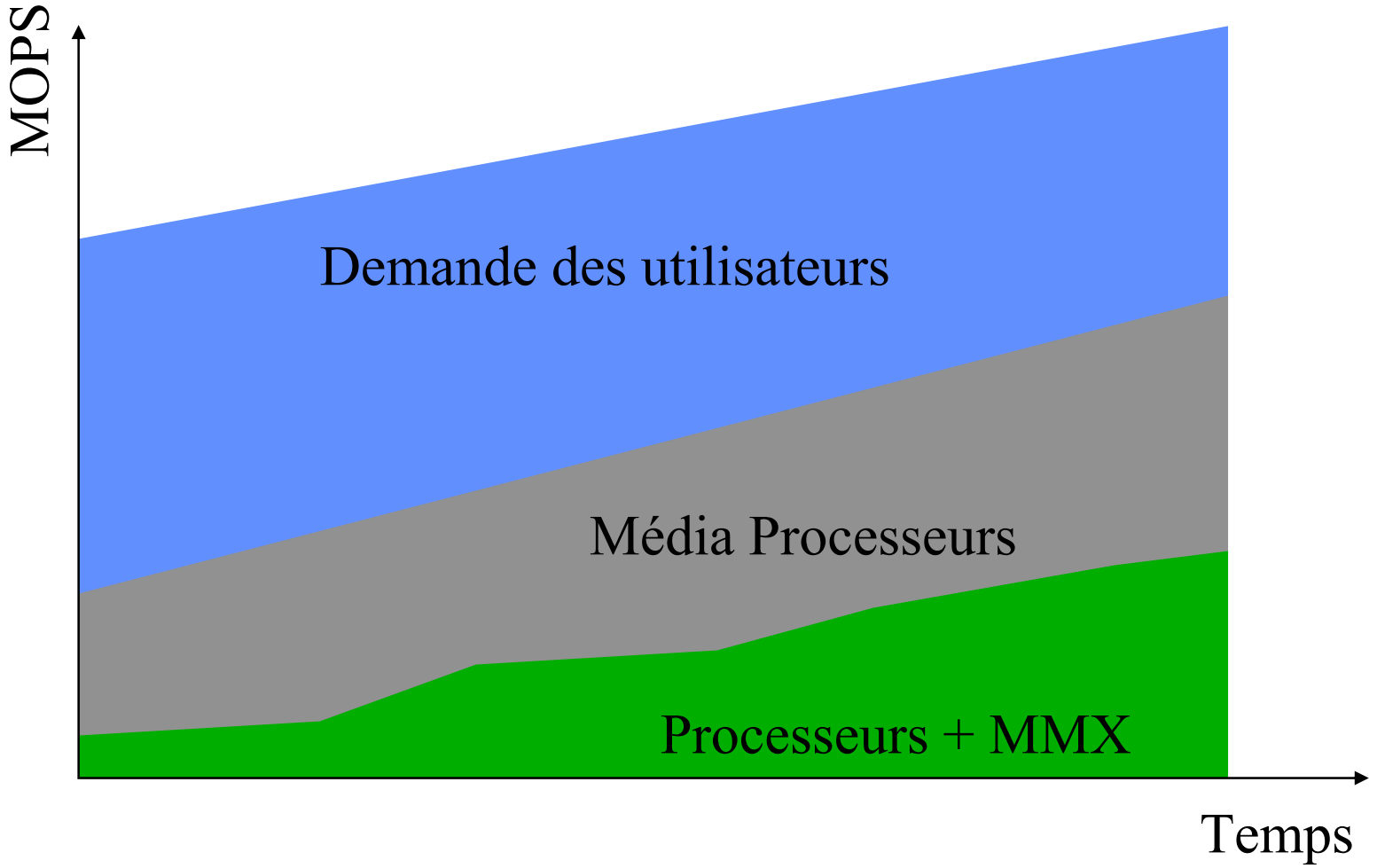
➢ Mpeg : schéma bloc et répartition en terme de puissance de calcul





Panorama de processeurs

MMX ou Média Processeurs



Panorama de processeurs

❑ Cœurs de processeurs :

➢La société Zoran développe des cœurs de *processeurs* de traitement du signal (audio et vidéo) :

- ♦ circuits disponibles sous forme logicielle (Vérilog ou VHDL)
- ♦ indépendant de la technologie
- ♦ fournis sous forme de modèles synthétisables
- ♦ interface générique permettant de les associer à d'autres fonctions
- ♦ prédiction de Zoran pour l'horizon 2001 : 20 % des circuits spécifiques seront développés à partir de composants virtuels
- ♦ la difficulté actuelle est liée aux problèmes de propriétés industrielles de ces circuits (*pouvoir les diffuser sans pour autant se les faire pirater*)

Panorama de processeurs

- Xilinx Alliance core :
  - propose un cœur de processeur "LavaCore" :
    - configurable Java Processor Core
    - exécution directe (hardware) du byte code Java
    - cœur implémentable dans les circuits de la famille :
      - Virtex-II
      - Virtex-E

Panorama de processeurs

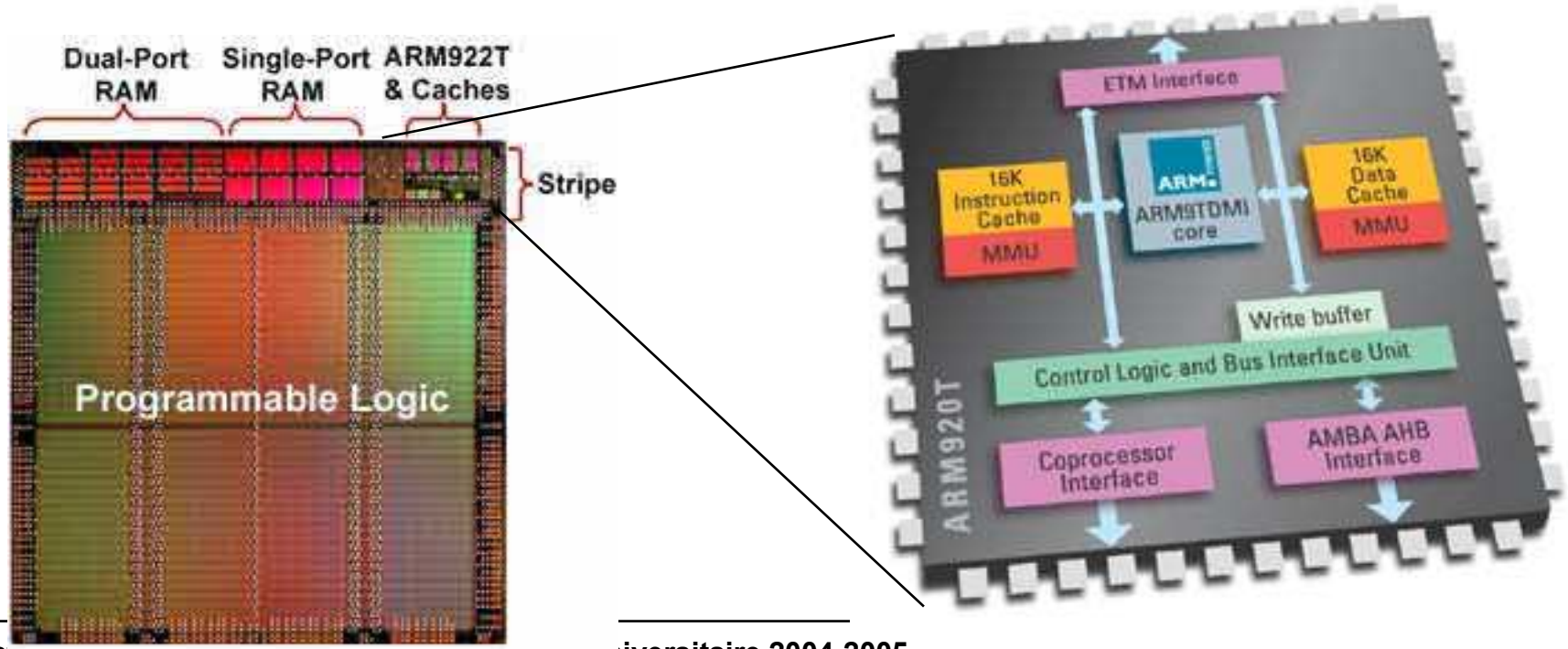
➢ Altera :

♦ propose des cœurs de processeur :

– Nios :

• processeur RISC

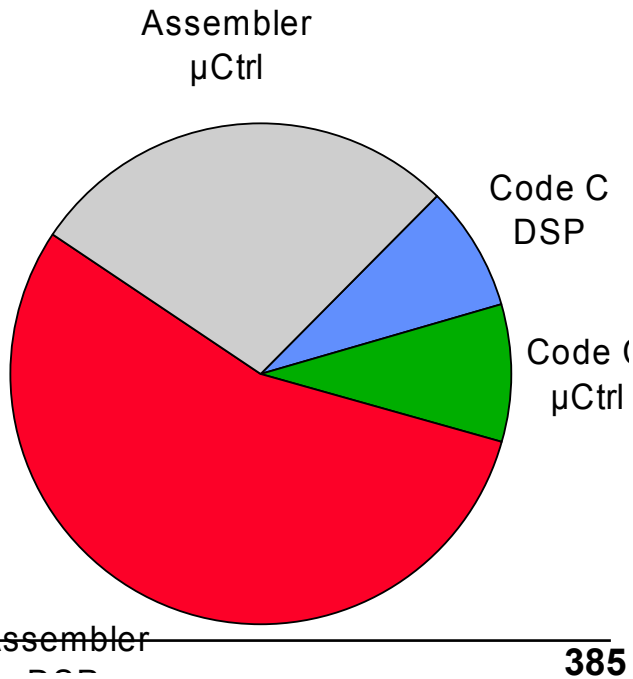
– ARM : Excalibur EPXA10 Device



Panorama de processeurs

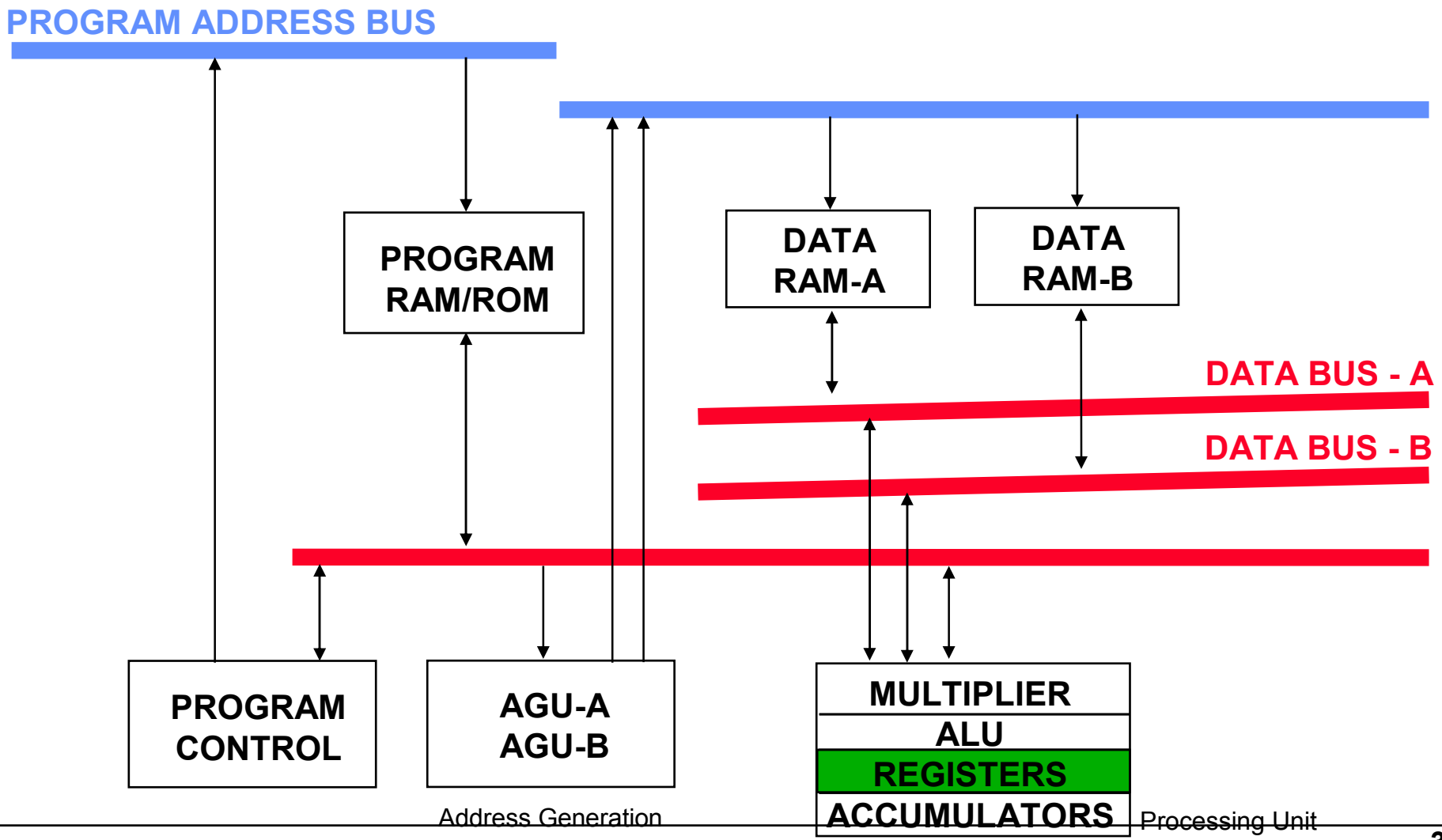
Les processeurs de traitement du signal

- Ce sont les premiers processeurs a avoir adopté en masse l'architecture Harvard :
  - bus de données et bus d'adresses distincts
- Contrôleur RISC
- Ils possèdent des opérateurs spécifiques très performants pour les applications de traitement du signal :
  - multiplication - addition
  - manipulation de bits (très utile pour la FFT)
- Mise en place de plusieurs bancs mémoire parallèles : accès simultané à plusieurs données
- Traitements en 32 bits flottants ou 16 bits virgule fixe
- Traitements parallèles
- Nécessité de faible consommation :
  - applications portables : téléphonie, systèmes embarqués



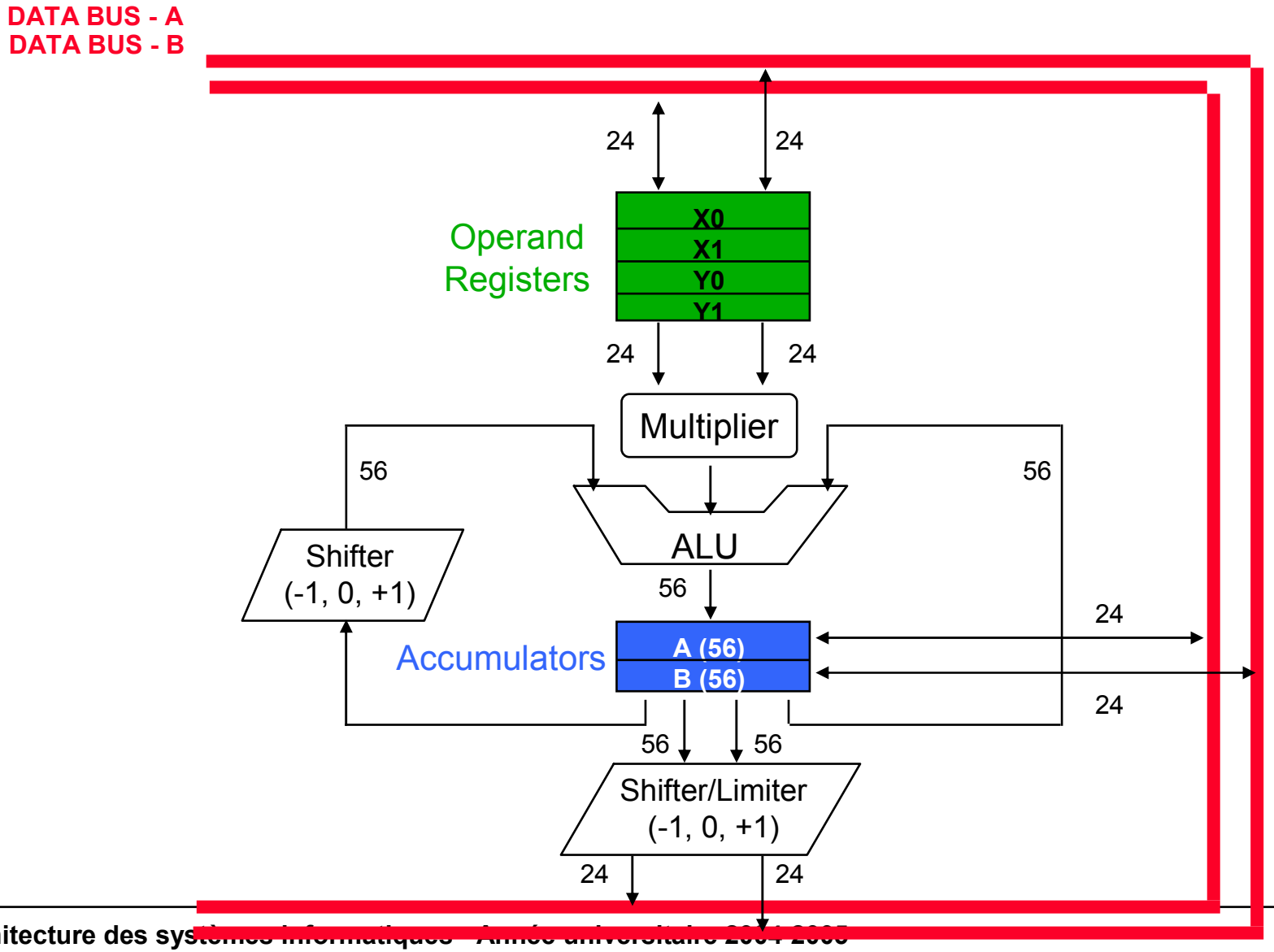
Panorama de processeurs

♦ Structure générale d'un DSP



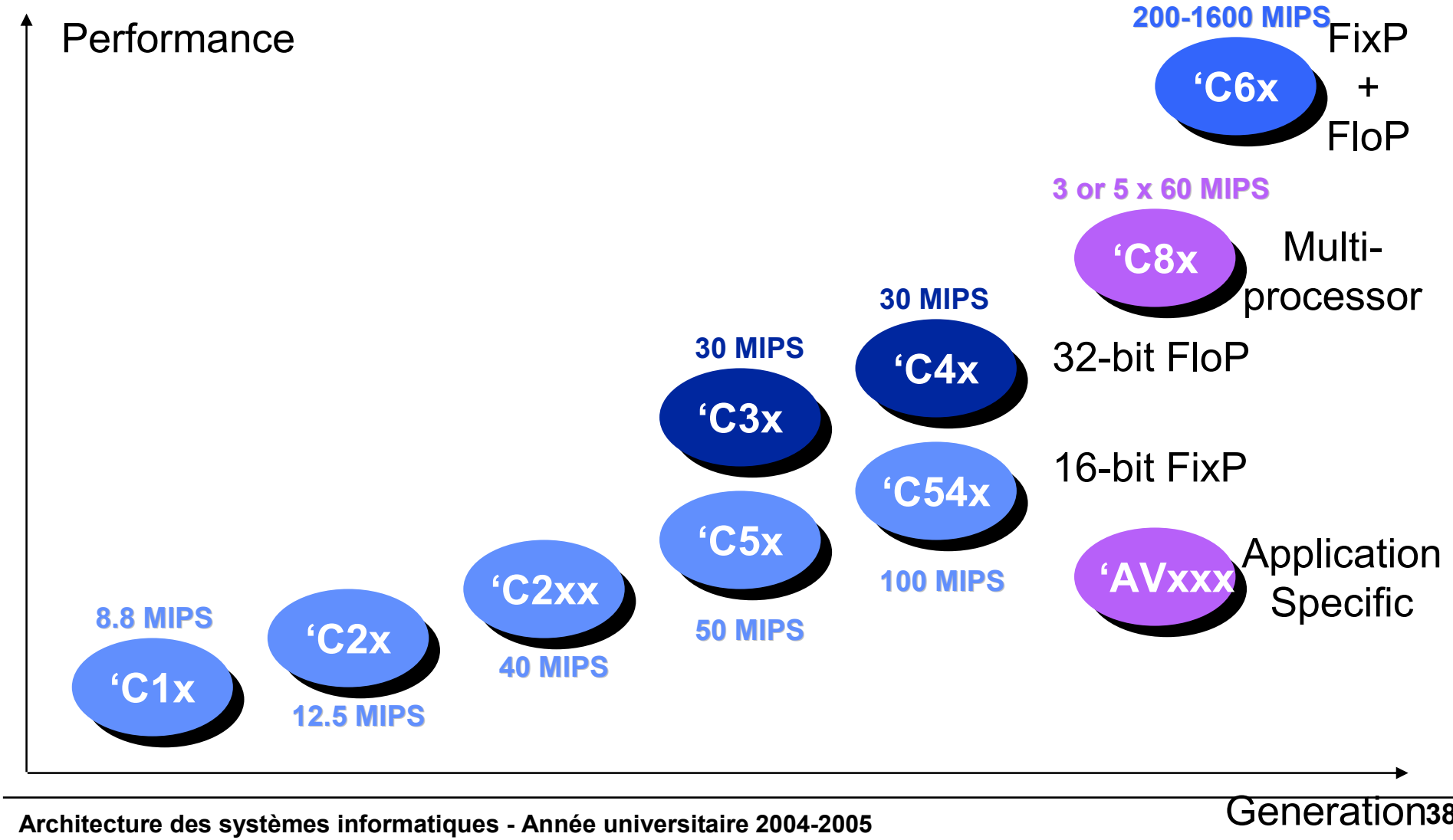
Panorama de processeurs

♦ Structure générale d'une unité de calcul de DSP



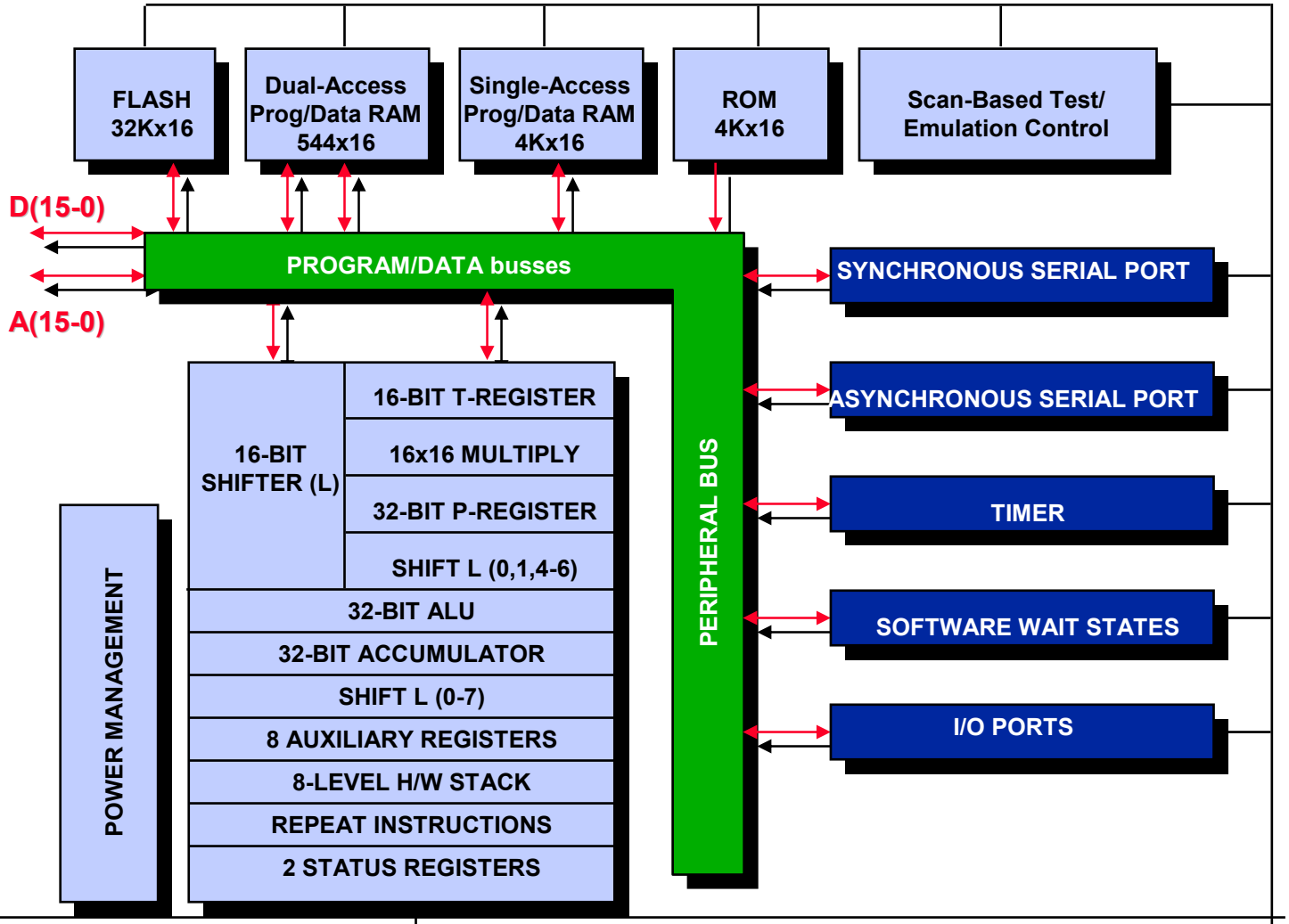
Panorama de processeurs

❑ La famille des processeurs TMS 320 C XX



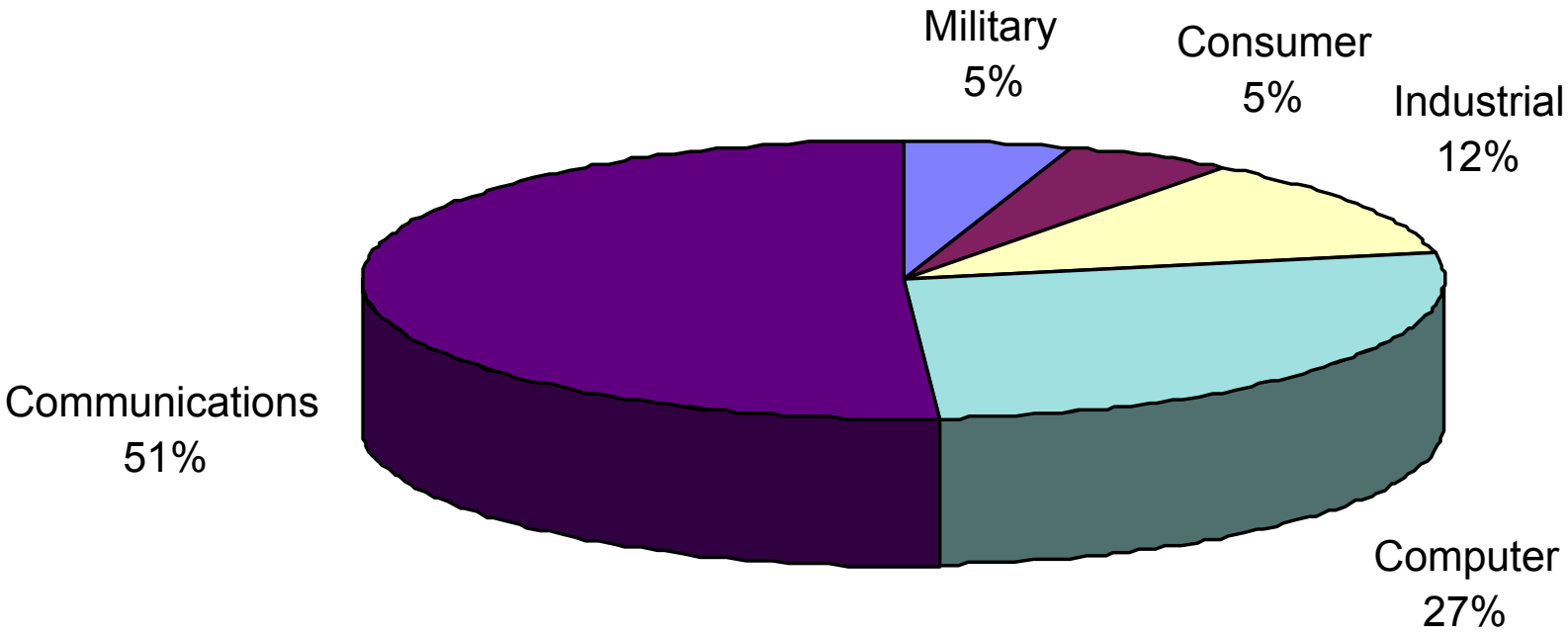
Panorama de processeurs

♦ Architecture des processeurs TMS 320 C XX



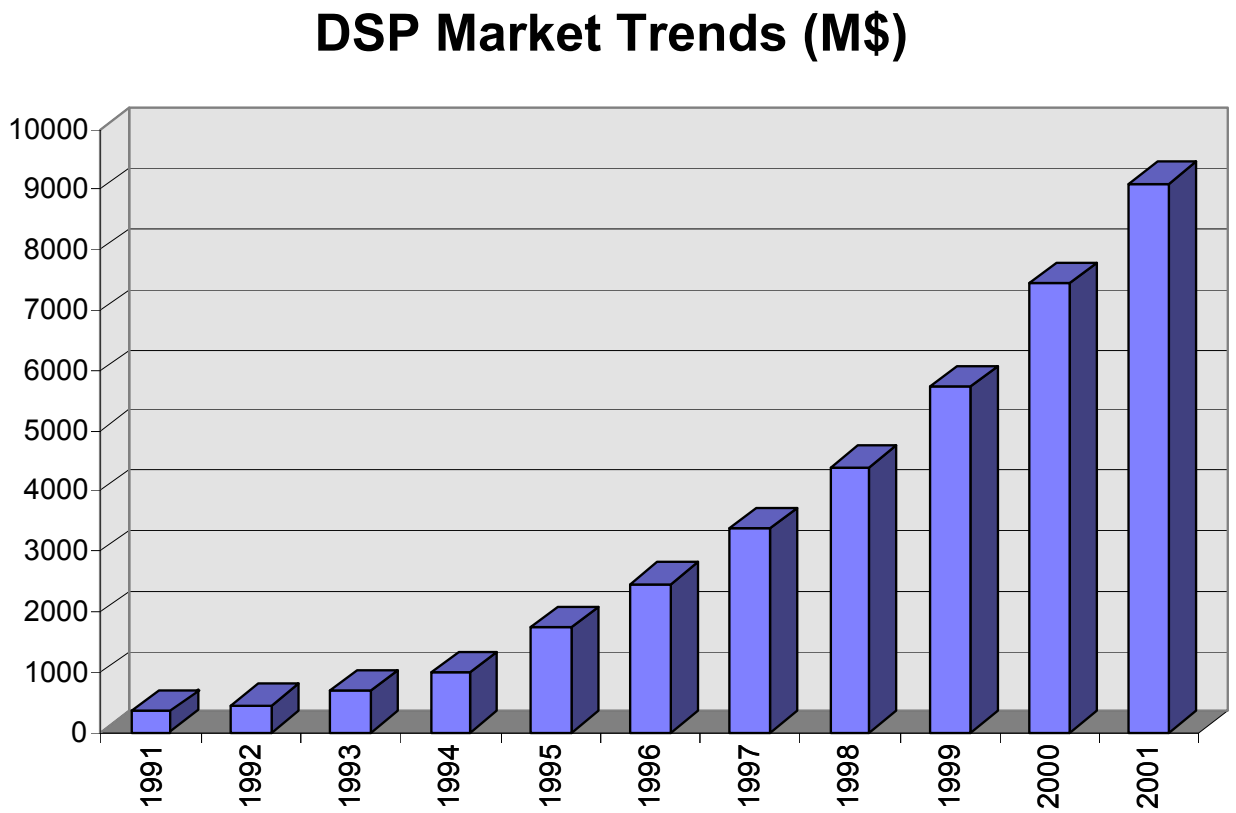
Panorama de processeurs

➢Utilisation des DSP (REE 10 / 97)



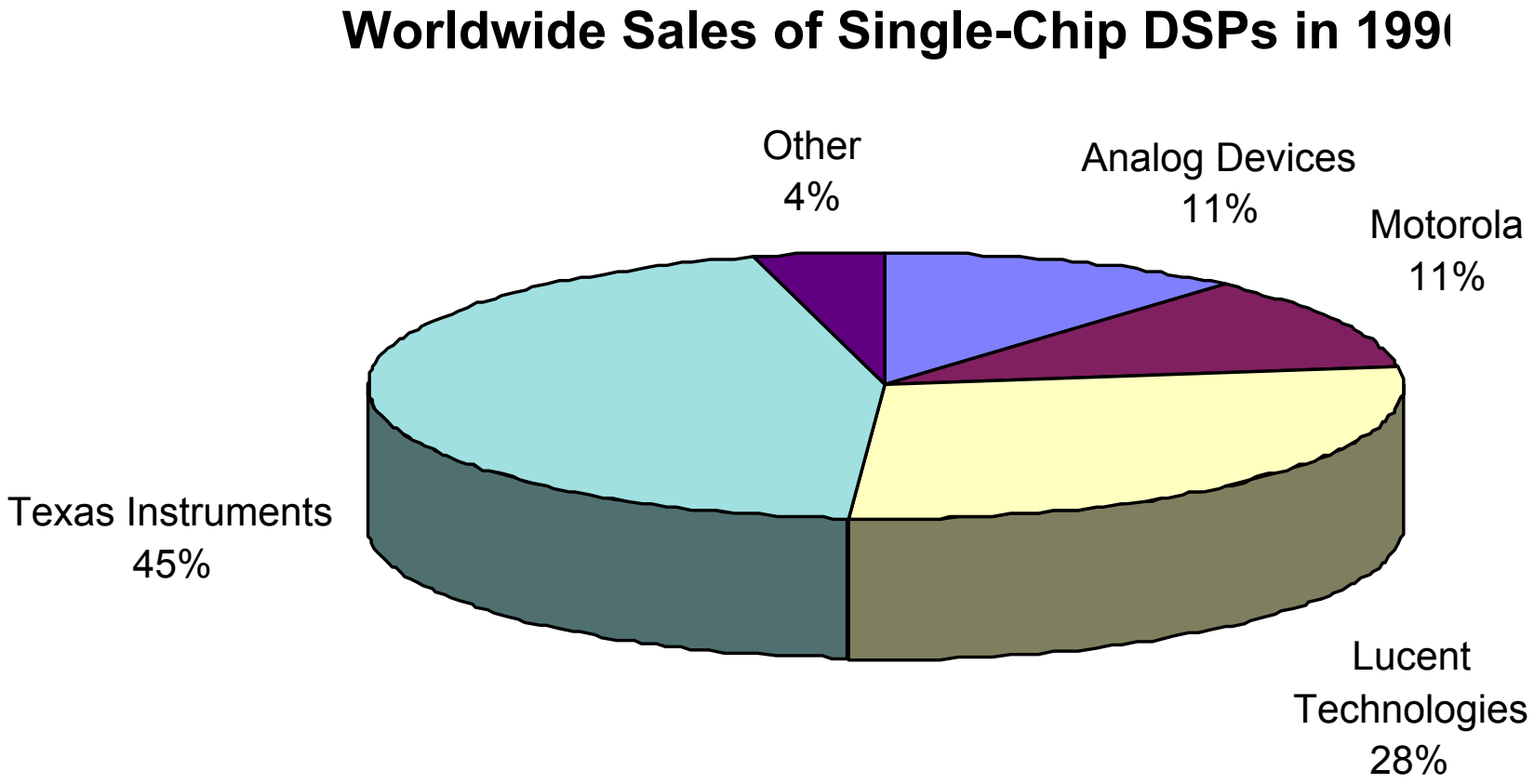
Panorama de processeurs

➢Evolution du marché des DSP



Panorama de processeurs

➢Les vendeurs de DSP



Marché de $395M