

# Introduction aux problématiques architecturales des Systèmes d'Exploitation

Jean-Baptiste.Yunes@univ-paris-diderot.fr

v0.7

Disclaimer: many of the pictures here presented come from the Internet,  
not to overload the slides I assume the fact that no copyright notice is included,  
if any is a copyright infringement please tell me.

# Préambule

Cours inspiré des « cours » de

- Ronan Keryell
  - Systèmes d'exploitation et supports architecturaux  
3A SLR F2B205A 29/10/2008 v1.20
- modifié par François Armand

Ceci **n'est pas** un support de cours...

Dense... beaucoup de notions... pas exhaustif...

Il vous en restera à découvrir par la suite

# Préambule

C'est un cours « informatif »

- Soyez réactifs
  - posez des questions si l'envie vous en prend
- Retenez les concepts, les idées
  - des exemples illustreront certains concepts
- Beaucoup d'exemples sont basés sur le système ouvert de Linux, mais pas que...



# Bibliographie

- Systèmes d'exploitation

Andrew Tanenbaum

- Operating Systems Concepts

Abraham Silberschatz



# Bibliographie

- Unix Internals  
Uresh Vahalia
- The magic garden explained  
Berny Goodheart & James Cox
- Solaris Internals  
Jim Mauro
- Windows Internals  
David Solomon
- Understanding the Linux kernel  
Daniel Bovet & Marco Cesati

# Problèmes

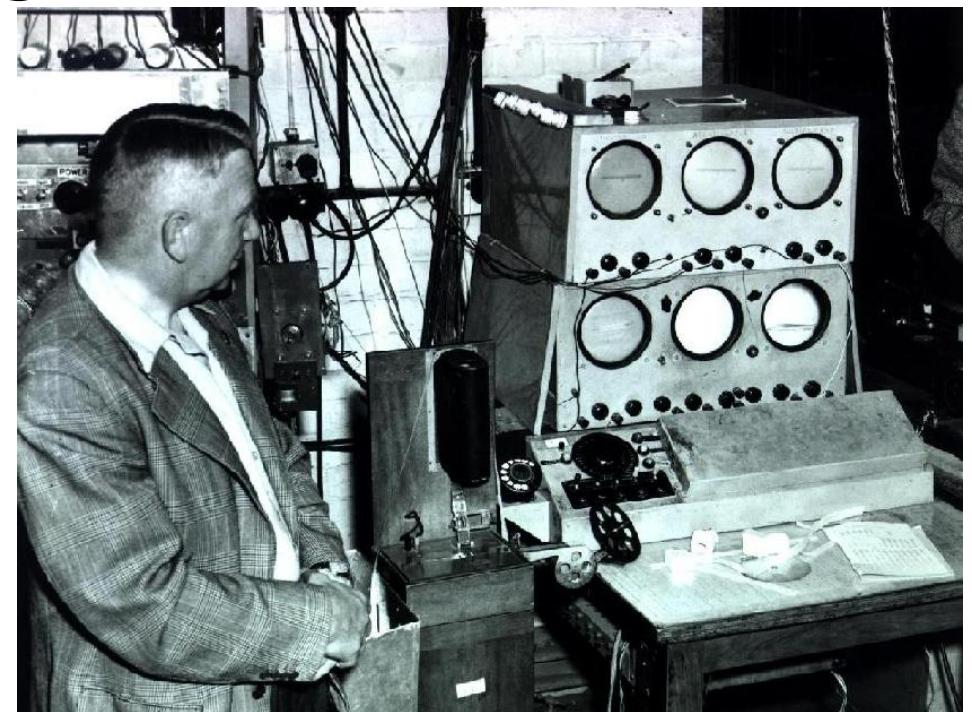
Dès le début

EDSAC, Cambridge University

Electronic Delay Storage Automatic Calculator

Ergonomie de l'usage...

1951



# Besoins

L'informatique est pervasive

Elle envahie tous (beaucoup) les pans de nos activités

- Ordinateurs, téléphones, tablettes, etc
- Voitures, machines à laver, etc
- Applications industrielles...



# Besoins

Système d'exploitation :

- assurer l'ordonnancement de tâches de calcul en garantissant autant que possible un fonctionnement optimal de la machine
- fournir un environnement pratique pour l'exécution de tâches ou le développement d'applications



# Besoins

Les besoins sont variés en pratique :

- Interagir avec le monde extérieur (utilisateurs, autres machines)
- Maintenir le fonctionnement (disponibilité)
- Stocker des données
- Assurer de la sécurité ou la sûreté



# Problèmes

La variété des dispositifs

- processeurs
- architectures matérielles
- périphériques
- ...



# Problèmes

## Modèles de CPU

- Représentation des nombres
- Arithmétique
- Codage des caractères
- Jeu d'instructions



# Problèmes

CPUs (liste non-exhaustive!)

- Intel / AMD families, x86, x86-64
- IBM, Power PC
- Sun, Sparc
- ARM, ARM
- MIPS, MIPS
- Hitachi, SuperH
- IBM/Sony/Toshiba, Cell
- TI, DSP
- ...

# Problèmes

Les périphériques :

- Matériel
- Logiciel

Le cauchemar :

- vous connaissez...l'installation des bons « drivers »
- les mise-à-jours (sécurité, fonctionnalités, etc)



# Problèmes

The major cause of the software crisis is that machines have become several orders more powerful!

To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

In this sense the electronic industry has not solved a single problem, it has only created them, it has created the problem of using its products.

— Edsger Dijkstra, **1972**

# Besoins

Assumer la paresse naturelle

- de l'informaticien ?
- sinon à quoi serviraient les machines ?

Une devise : réutiliser

- le graal des informaticiens
- déclinaisons : portabilité, interopérabilité, pérennité (la durabilité informatique)

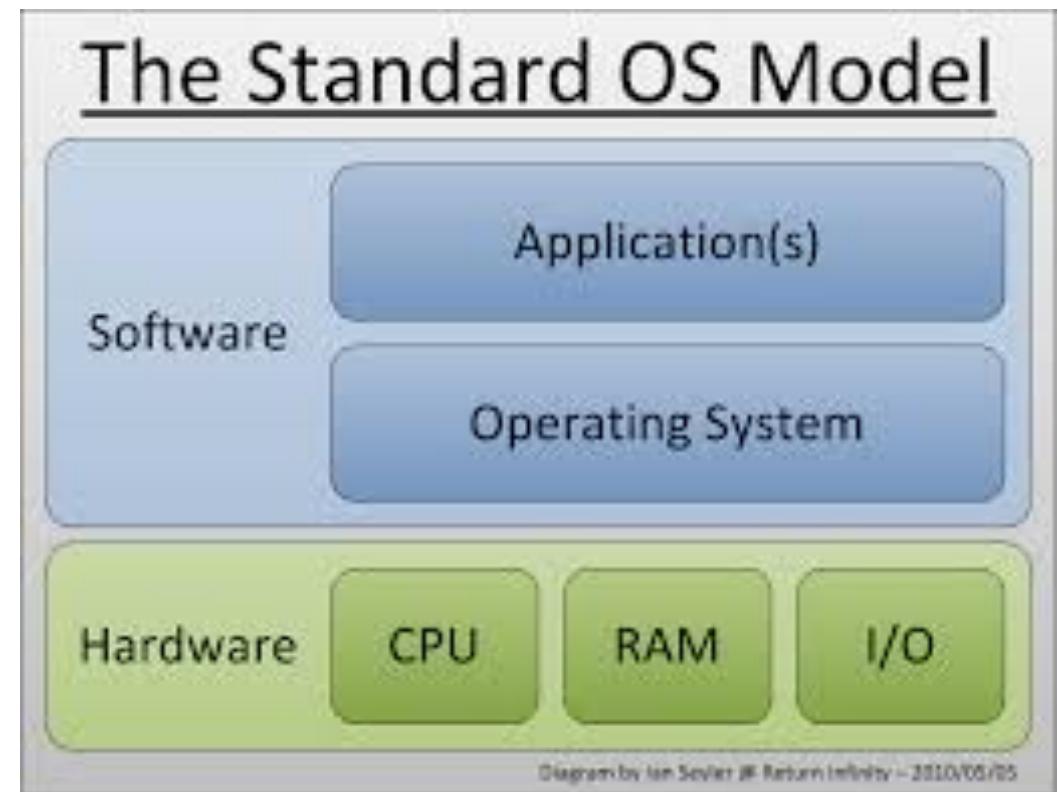
Éviter de réécrire

- gain de temps
- ne pas réintroduire d'erreurs précédemment éliminées
  - en fait, ne réécrire qu'à bon escient

# Architecture système

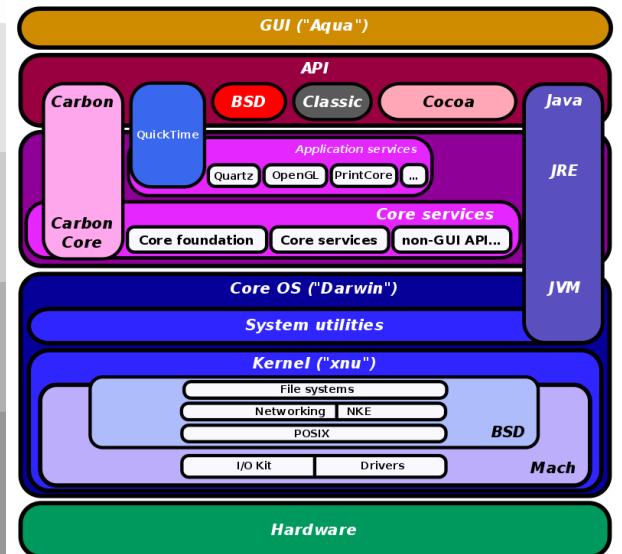
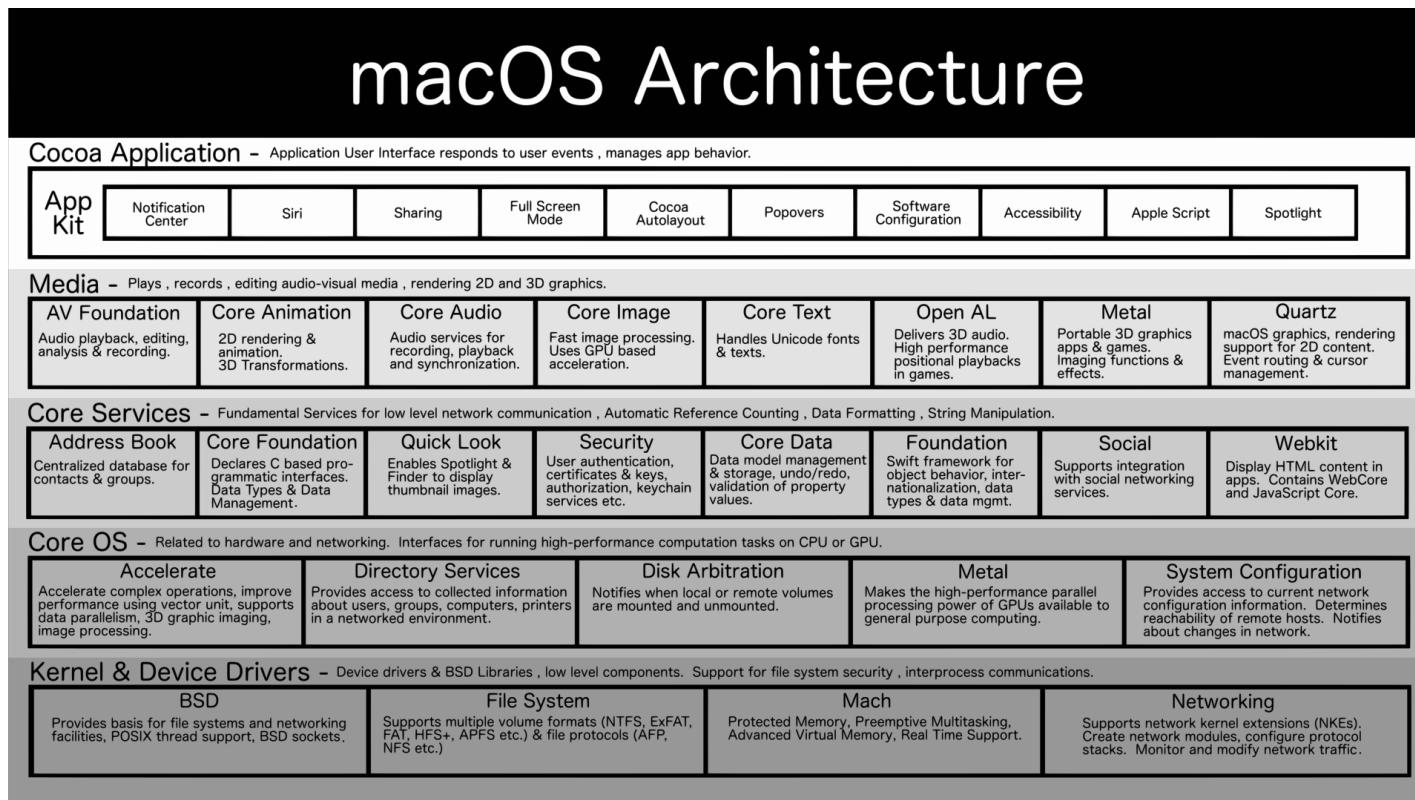
Repose sur l'articulation de trois composants principaux :

- le matériel
- le noyau
- les applications



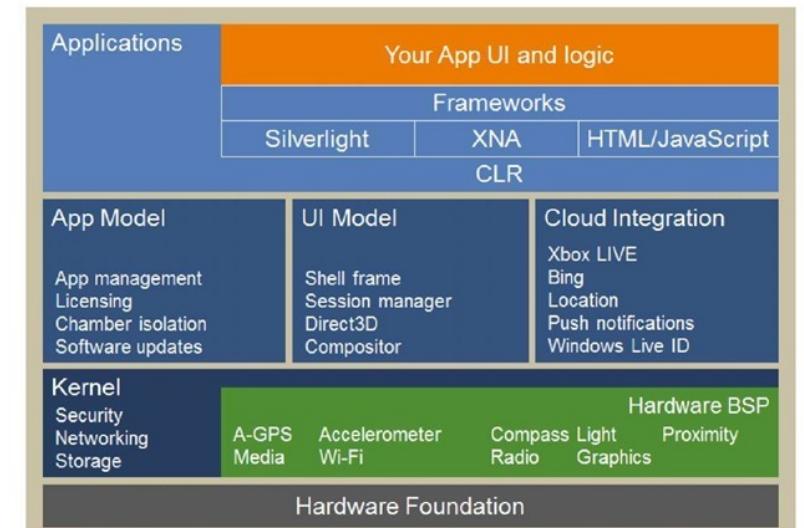
# Architecture système

Ex. : Mac OS

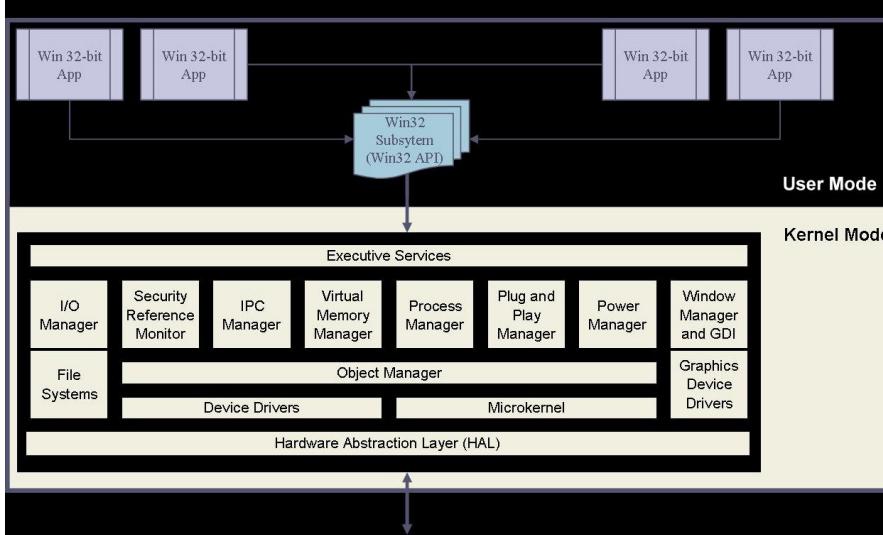


# Architecture système

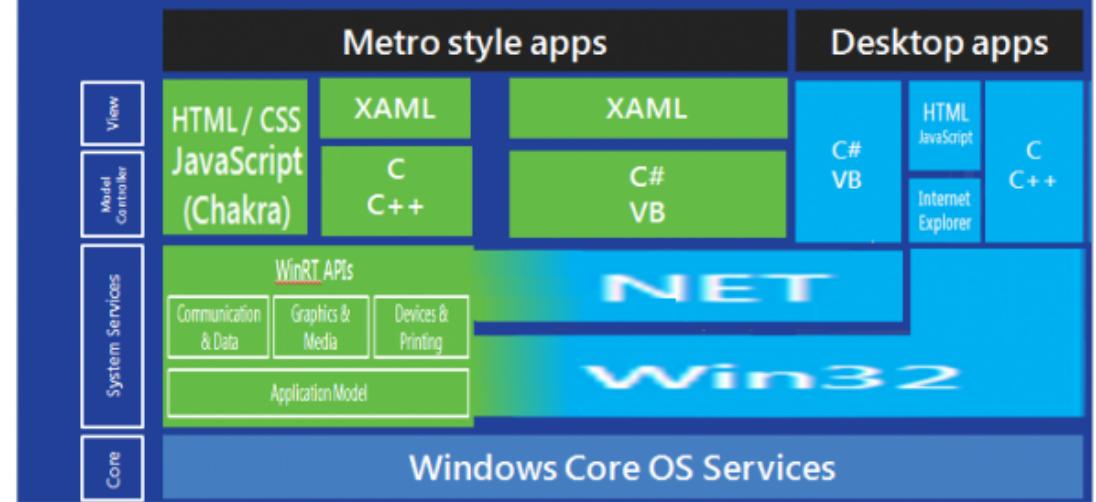
## Windows



Windows XP Architecture diagram

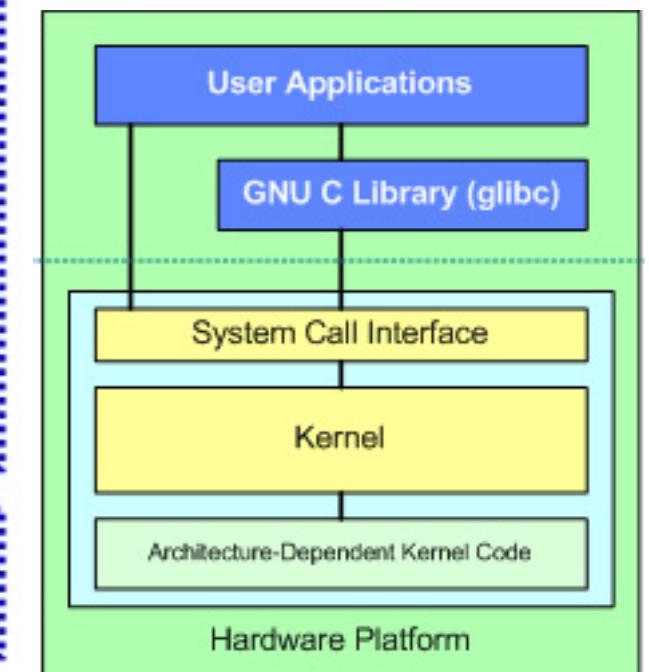
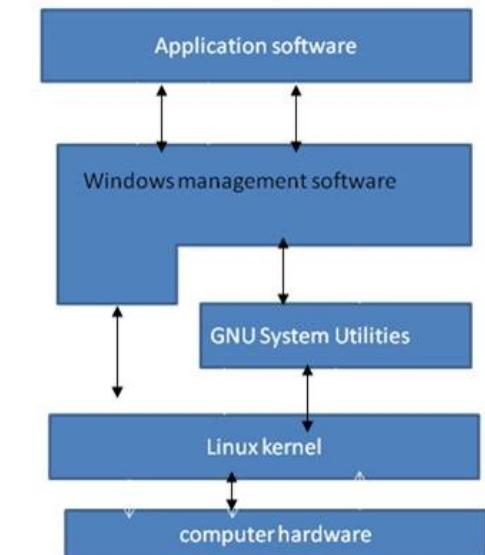
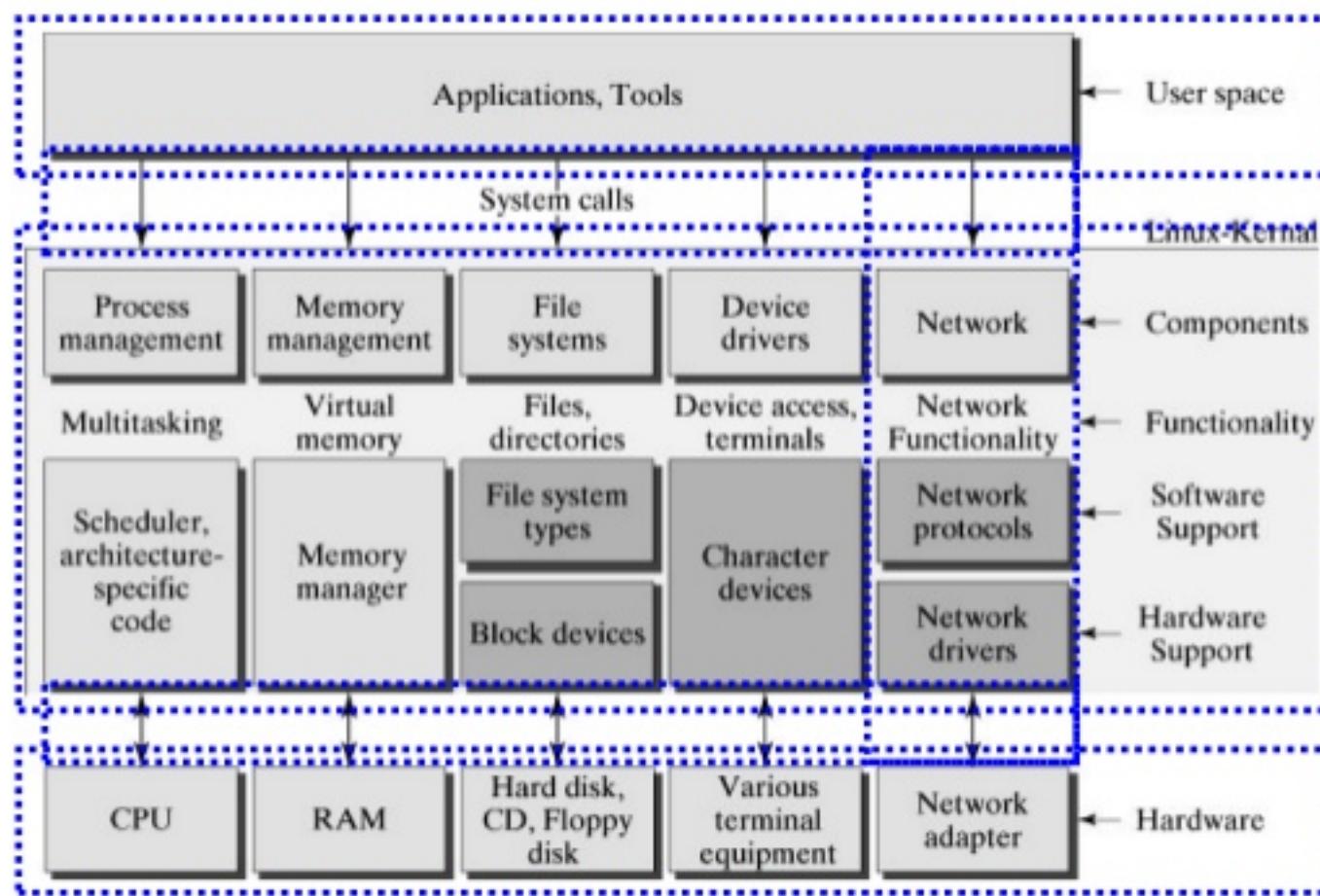


Windows 8



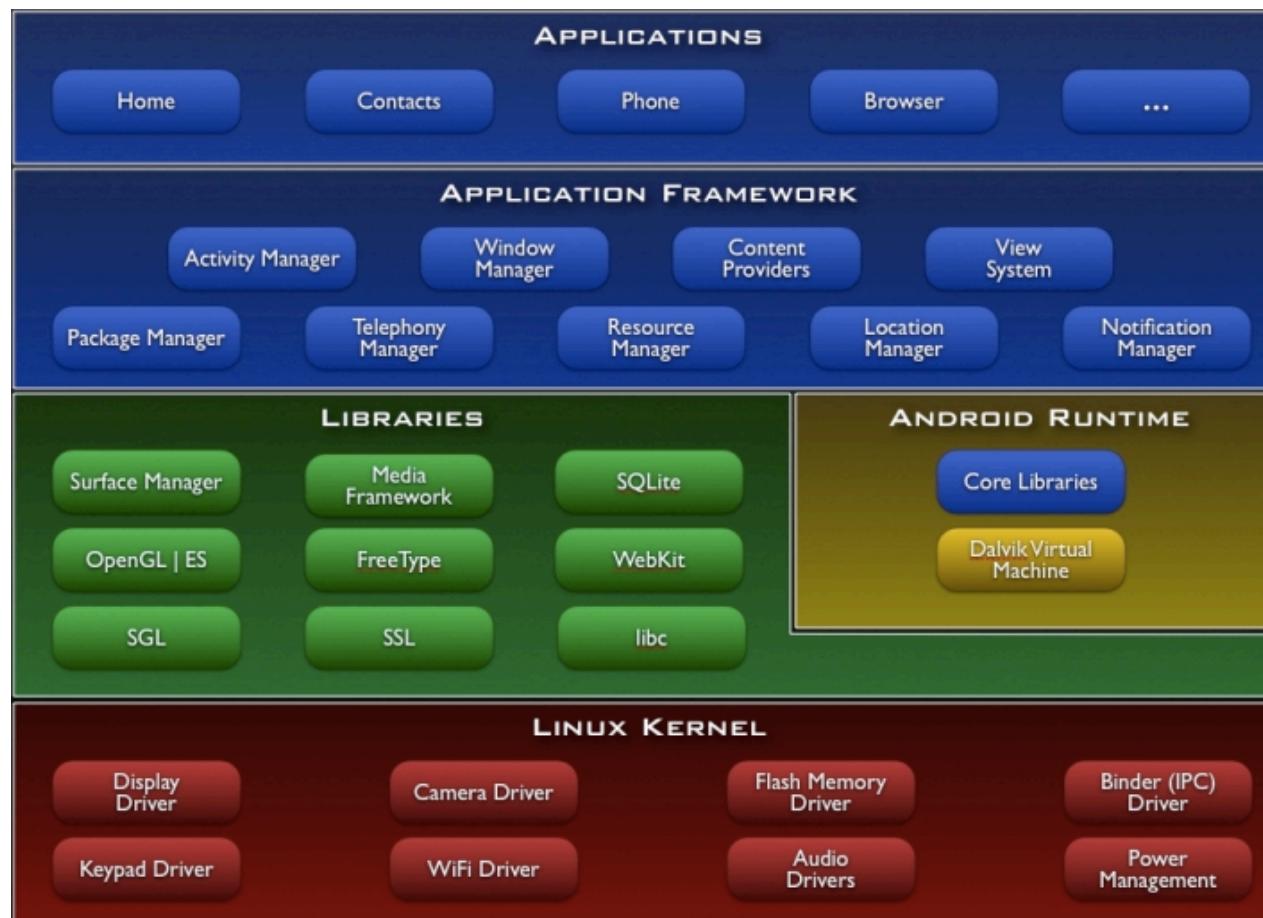
# Architecture système

## GNU/Linux



# Architecture système

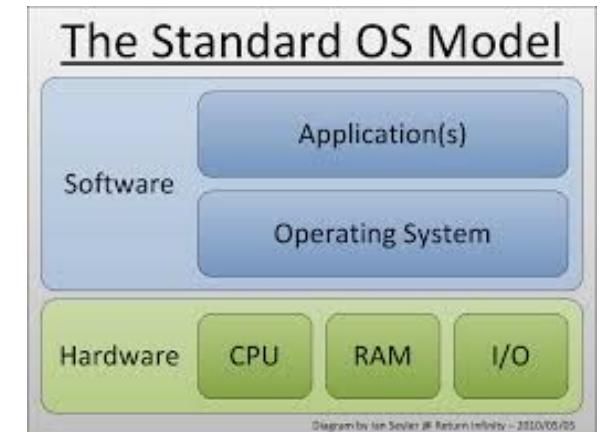
## Android/Linux



# Architecture système

Couplage faible entre les composants

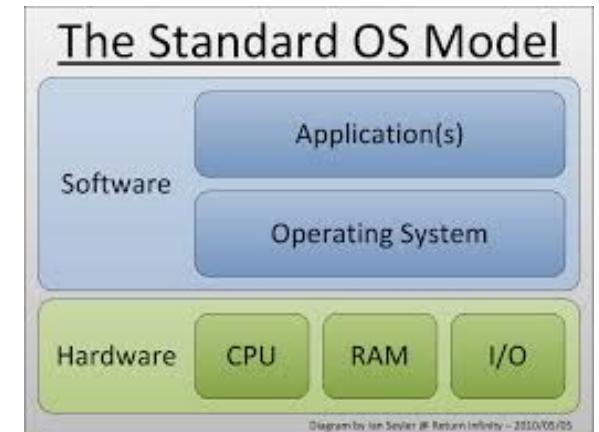
- Modularité
- Relative indépendance de conception
- Portabilité
- Interchangeabilité



# Architecture système

## Quelques problèmes

- L'existent limite/freine l'innovation
  - Optimisation difficile voir impossible
  - Portabilité
  - Interchangeabilité



# Architecture système

Quelques problèmes

L'architecture n'est pas neutre!

Des systèmes de la même famille ne sont pas facilement interchangeables, interopérables, etc

Par exemple la difficile fusion de MacOS et iOS, ou Windows et Windows tablet, Linux/Android

# Familles de systèmes

Unix n'est pas un système, c'est plutôt une famille, idem pour Windows, etc

- dans chaque famille les systèmes sont adaptés à certaines plateformes :  
laptops, desktops, serveurs, tablets, smartphones, game console,  
embedded systems  
processeurs et cœurs (NUMA, SMP, etc)
- adaptés à certains écosystèmes :  
libre, ouvert, propriétaire
- adaptés à certains types d'utilisation :  
mono-utilisateur, multi-utilisateur
- offrent différents portefeuilles d'applications

# Services rendus

Un système est une machine virtuelle :

de très nombreux détails sont cachés

la machine virtuelle est plus « agréable » que la machine brute

- plusieurs programmes co-existent simultanément
- plusieurs utilisateurs
- affranchissement de limitations matérielles (mémoire virtuelle)
- fichiers et répertoires
- interfaces graphiques



# Services rendus

Gestion « optimale » des ressources :

le problème du partage

cpu, mémoire, disque, réseau, etc

Qualité de service :

disponibilité

latence et interactivité

débit

# Fonctions d'un système d'exploitation

Gestion des tâches/threads/processus  
création, destruction, contrôle  
ordonnancement, synchronisation  
concept d'application

Gestion de la mémoire  
allocation, libération, protection d'accès  
pagination, segmentation, swap  
partage

# Fonctions d'un système d'exploitation

Gestion des entrées/sorties

audio, vidéo, capteurs, écrans, claviers, souris,  
trackpad, tactile, USB, wifi, ethernet, etc

pilotes

protocoles

interruptions, DMA, etc

# Fonctions d'un système d'exploitation

Gestion des fichiers

création, destruction, protection

modification

répertoires, disques logiques, disques réseau



# Fonctions d'un système d'exploitation

Gestion de la communication

inter-processus

    signaux, segments partagés, messages, boîte de messages,  
    etc

synchronisation

inter-systèmes

    protocoles réseaux

Gestion des utilisateurs

Gestion du système

    mise au point, maintenance, surveillance, etc

# Fonctions d'un système d'exploitation

Gestion de l'énergie  
devenu essentiel et critique!

cf. offres d'emploi d'énergéticiens/  
informaticiens/physiciens dans les grandes  
compagnies informatiques

## Job Summary

Apple's Platform Architecture team is looking for an exceptional senior software engineer to develop, prototype and evaluate a variety of power management algorithms. The ideal candidate is a self-driven fearless individual that takes

# Aspects non fonctionnels

Débit

calcul, acquisition/restitution

Réactivité

interactivité

Contraintes temps-réel (plus loin)

Sûreté de fonctionnement (plus loin)

Tolérance aux pannes (plus loin)

Gestion de l'énergie

devenu essentiel et critique, cf. offres d'emploi d'énergéticiens/informaticiens/physiciens dans les grandes compagnies informatiques

## Job Summary

Apple's Platform Architecture team is looking for an exceptional senior software engineer to develop, prototype and evaluate a variety of power management algorithms. The ideal candidate is a self-driven fearless individual that takes

# Temps-réel ?

Un système temps-réel doit réagir aux stimuli de son environnement dans des délais impartis propres au domaine

système de freinage, contrôle d'automatisme, vidéo, acquisition données, guidage fusée, etc

Un système temps-réel ne doit pas seulement fournir des données correctes mais en temps et en heure (beaucoup plus difficile)

résultats corrects si fournis en temps, des résultats tardifs sont faux!

# Contraintes en temps-réel

Dures (hard realtime) ou strictes

une contrainte est dure si de son non-respect résulte une catastrophe (Kopetz 1977)

centrale nucléaire, avion, pacemaker, etc

Molles (soft realtime) ou souples ou lâche

les autres...

vidéo, audio, jeu en réseau, etc

en général on considère qu'un rattrapage à court terme suffira à compenser (ex. : lag vidéo...)

# Sûreté de fonctionnement (dependability)

Fiabilité (reliability)

probabilité que le système fonctionne correctement à  $t + \Delta t$ , s'il fonctionnait correctement à  $t$

Maintenabilité (maintainability)

probabilité que le système fonctionne correctement à  $t + \Delta t$ , si une erreur s'est produite à  $t$

Disponibilité (availability)

probabilité que le système soit prêt à assurer un service à  $t$

Sécurité (safety)

absence de conséquence externe catastrophique

Confidentialité (confidentiality)

absence de fuite d'information

Intégrité (integrity)

absence d'altération (cohérence)

# Résistance aux pannes

Super-calculateurs

des milliers, centaines de milliers, millions de processeurs! La probabilité que certains tombent en panne chaque jour est plutôt élevée!

Cray >6.000.000 cores, HPE > 4.500.000 cores, IBM > 4.000.000 cores

Cloud

des milliers, etc de machines, disques, équipements réseau

Google > 1.000.000 serveurs, Facebook > 100.000 serveurs

Processeur

multi-cœur, intégration élevée

Embarquement

spatial, environnements hostiles, militaire, etc

# Processus ?

Une abstraction système de type machine virtuelle  
le programmeur peut penser isolément, la  
machine est pour lui seul

- un processeur (ou plusieurs)
- instructions élémentaires + services
- une mémoire
- indépendante de la réalité (enfin presque)

# Contexte

C'est l'ensemble des informations détenues par le système sur le processus et qui lui permettent de le manipuler à sa guise

- assurer la cohabitation de plusieurs processus
- assurer l'exécution contextuelle de services (droits)
- assurer la bonne communication
- traçage / comptabilité

# Fichier

Conteneur abstrait permettant des opérations de lecture et écriture

- un « fichier » ordinaire
  - texte, audio, vidéo, BdD, code source, etc
- un répertoire
- un programme exécutable
- (Unix) un périphérique
  - disque, clavier, terminal, connexion réseau, etc
  - pseudo périphérique
- (Linux/BSD) un processus!

# Fichier

Tout peut être vu comme un fichier

c'est l'idée de départ d'Unix

l'idée n'a pas été menée jusqu'au bout

la dernière tentative est celle du système  
plan9 (successeur proclamé d'Unix)



# Fichiers spéciaux (Unix)

Disques :

`/dev/sd*` `/dev/disk*`

Terminaux :

`/dev/tty*` `/dev/pty*` `/dev/pts/*`

Souris

`/dev/input/mouse*`

Source aléatoire

`/dev/random` `/dev/urandom`

Puit sans fond

`/dev/null`

Source nulle

`/dev/zero`

# Fichiers spéciaux (Unix)

Mémoire :

`/dev/mem*` `/dev/ram*`

# Fichiers périphériques (Windows)

Vus comme des périphériques :

CON : console

AUX : auxiliaire

LST : imprimante

PRN : imprimante qui numérote les lignes

NUL : puit sans fond

EOF : source infinie de caractère EOF (0x1A)

INP : entrée

OUT : sortie

ne sont plus utilisés depuis longtemps...

# Transparence

C'est transparent à l'utilisateur  $\Leftrightarrow$  Il ne voit rien,  
c'est opaque

Hétérogénéité

fichiers/répertoires : la plupart des utilisateurs  
ne sont en rien conscient des différences...

marque/modèle

etc

# Transparence

C'est transparent à l'utilisateur  $\Leftrightarrow$  Il ne voit rien, c'est opaque

Localisation

L'utilisateur se préoccupe assez peu de l'endroit exact où ses fichiers (données) sont stocké(e)s

local ou non : Clouds

un ami à moi : « c'est là-haut » (dans les nuages)

Mac OS Spotlight, Windows search, Ubuntu dash...

le mail... (Savez-vous où sont vos stockés vos mails ?)

# Transparence

C'est transparent à l'utilisateur  $\Leftrightarrow$  Il ne voit rien,  
c'est opaque

Migration

Cloud, serveurs qui se déplacent sans qu'on le  
sache

Distribution

Les données peuvent être fragmentées et  
distribuées à différents endroits

# Transparence

C'est transparent à l'utilisateur  $\Leftrightarrow$  Il ne voit rien,  
c'est opaque

RéPLICATION (tolérance aux pannes et performances)

Savez-vous quel serveur répond lorsque vous faites une recherche google ?

Cloud

Vision unique

# Transparence

C'est transparent à l'utilisateur  $\Leftrightarrow$  Il ne voit rien,  
c'est opaque

Caches

Données rapatriées au fil de l'eau au plus près  
de celui qui réclame

Transparent ? Vraiment ? Humm...

# Transparence

Un graal difficile à réaliser en pratique

Pas toujours aussi transparent

données perdues lors de crashes

incohérence de vision entre machines/applications

outils différents pour manipuler les objets

cp vs scp vs ftp

débit - latency - asynchronisme

Problème classique : le ping-pong (effet désastreux)

# Le mythe du SSI

Single System Image

le graal du « distribué » : l'utilisateur ne voit qu'une seule machine

ultimement la même machine pour tout le monde

en arrière-plan : l'unification des ordinateurs du monde en une même machine

la matrice ?



# Le mythe du SSI

Les propriétés attendues d'un SSI

- migration (process migration)
- point de contrôle (process checkpointing)
- un seul espace de processus (single process space)
- un seul espace de stockage (single root)
- un seul espace de périphériques (single I/O space)
- un seul espace de communication (single IPC space)

# Tentatives de réalisation SSI

## Amoeba (thx A. Tanenbaum)

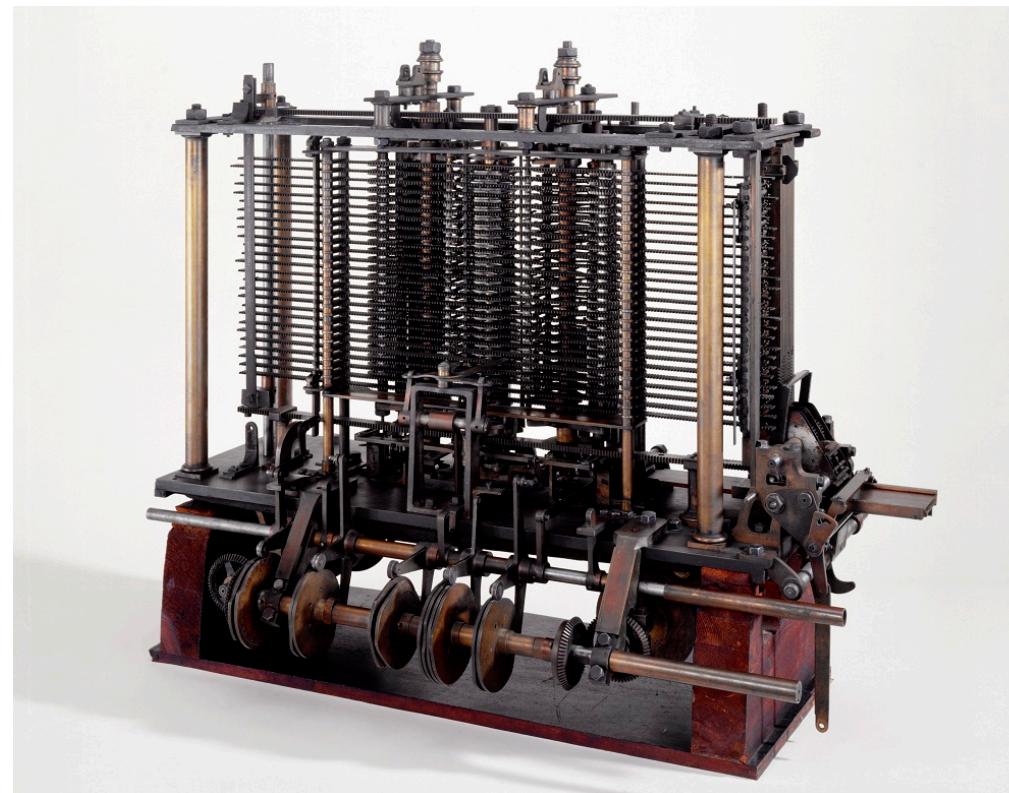
OpenMosix	Name	Process migration	Process checkpoint	Single process space	Single root	Single I/O space	Single IPC space	Cluster IP address <sup>[t 1]</sup>	Source Model	Latest release date <sup>[t 2]</sup>	Supported OS
Amoeba <sup>[t 3]</sup>	Yes	Yes	Yes	Yes	Yes	Unknown	Yes	Unknown	Open	July 30, 1996	Native
AIX TCF	Unknown	Unknown	Unknown	Yes	Unknown	Unknown	Unknown	Unknown	Closed	March 30, 1990 <sup>[7]</sup>	AIX PS/2 1.2
Inferno	No	No	No	Yes	Yes	Yes	Yes	Unknown	Open	March 4, 2015	Native, Windows, Irix, Linux, OS X, FreeBSD, Solaris, Plan 9
Kerrighed	Yes	Yes	Yes	Yes	Yes	Unknown	Yes	Unknown	Open	June 14, 2010	Linux 2.6.30
LinuxPMI <sup>[t 4]</sup>	Yes	Yes	No	Yes	No	No	Unknown	Open	June 18, 2006	Linux 2.6.17	
LOCUS <sup>[t 5]</sup>	Yes	Unknown	Yes	Yes	Yes	Yes <sup>[t 6]</sup>	Yes <sup>[t 6]</sup>	Unknown	Closed	1988	Native
MOSIX	Yes	Yes	No	Yes	No	No	No	Unknown	Closed	May 16, 2016	Linux
<u>openMosix</u> <sup>[t 7]</sup>	Yes	Yes	No	Yes	No	No	No	Unknown	Open	December 10, 2004	Linux 2.4.26
Open-Sharedroot <sup>[t 8]</sup>	No	No	No	Yes	No	No	No	Yes	Open	September 1, 2011 <sup>[8]</sup>	Linux
OpenSSI	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Open	February 18, 2010	Linux 2.6.10 (Debian, Fedora)
Plan 9	No <sup>[9]</sup>	No	No	Yes	Yes	Yes	Yes	Yes	Open	January 9, 2015	Native
Sprite	Yes	Unknown	No	Yes	Yes	No	No	Unknown	Open	1992	Native
TruCluster	No	Unknown	No	Yes	No	No	No	Yes	Closed	October 1, 2010	Tru64
VMScluster	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Closed	September 23, 2016	OpenVMS
z/VM	Yes	No	Yes	No	No	Yes	Yes	Unknown	Closed	November 11, 2016	Native
UnixWare NonStop Clusters <sup>[t 9]</sup>	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Closed	June 2000	UnixWare



# Historique

computerhistory.org (timeline/computers)

histoire.info.online.fr (<2012)

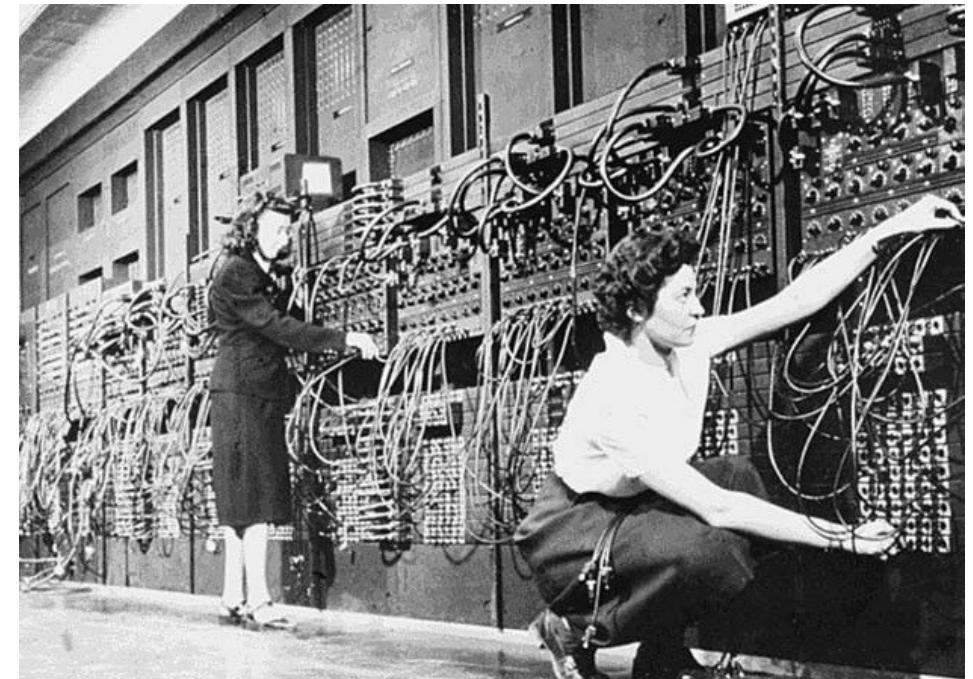


# Historique

1945 — 1955

Tout faire à la main...

Le système c'était l'humain avec son expertise  
(inefficace)

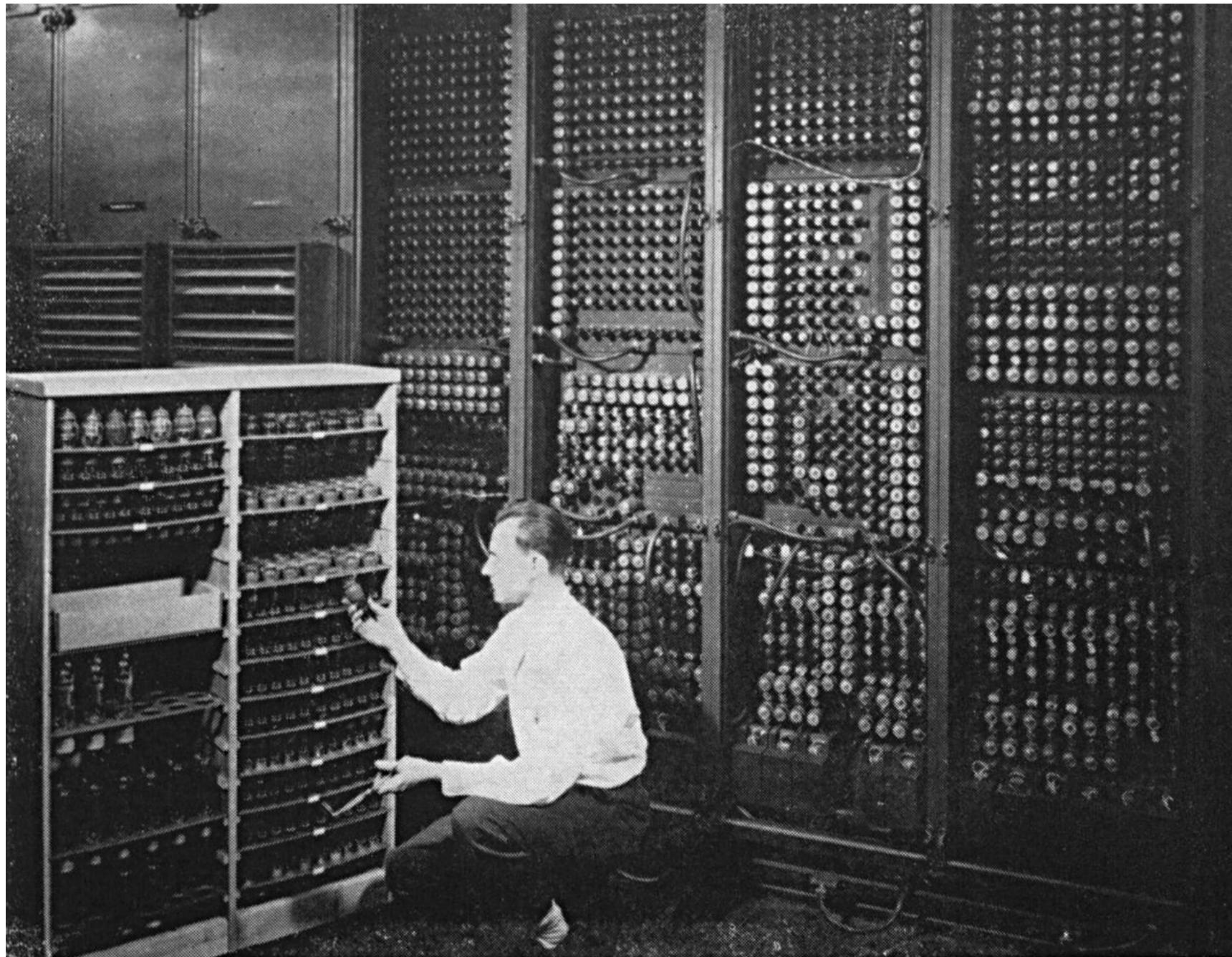


# Histoire — 1945

Principes établis par von Neumann, Eckert, Mauchly (1940)  
la mémoire contient à la fois le programme et ses données  
chemin de données

ENIAC : premier ordinateur entièrement électronique  
un « monstre » de 30 tonnes!  
disponibilité proche de 0...

Attention, de nombreux autres projets ont existé à cette époque (et un peu avant) mais étaient couverts par le secret militaire (Allemagne, Russie, Japon, France, Angleterre)...



**Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.**

# Historique

1955 — 1965

Langages de plus haut-niveau

FORmula TRANslator, COmmon Business Oriented  
Language, ALGOrithmic Language

Générations de machines

Naissance d'une véritable industrie

Atlas : Mémoire Virtuelle

UNIVAC, IBM

BESYS, ShareOS, CTSS, GCOS, TOPS-10

# Histoire — 1955

Toujours un seul utilisateur

Réservation de la machine (tranches horaires)

Traitements par lots (on fait la queue)

Mono-programmation

Premiers langages

écriture des programmes infinitement plus facile et rapide!

Premiers super-calculateurs

# Histoire — 1955

Toujours un seul utilisateur

Réservation de la machine (tranches horaires)

Traitements par lots (on fait la queue)

ordonnancement

Mono-programmation

Premiers langages

écriture des programmes infinitement plus facile et rapide!

Premiers super-calculateurs

Premiers disques durs

système de fichiers à fragmentation

# Histoire — 1960

Parallélisme



Bull Gamma 60, trop avancé (Bull en mourra)

Petits et gros

Spécialisation des tâches pour diminuer les coûts, IBM 1401 (gestion des données) et 7094 (calcul)



# Historique

1965 — 1985

Industrie florissante

Diffusion du temps partagé, de la  
multiprogrammation, etc

MVS, DOS, CP/M, Multics, Unix, VMS, MS-DOS



# Histoire — 1965

L'âge d'or des gros systèmes

multi-programmation

spooling

temps-partagé

disques durs

mémoire

écrans graphiques

modems

La loi de Moore

aujourd'hui on a des problèmes avec les effets quantiques

# Historique

1985 —

Micro informatique

Stations de travail

Réseau (Internet)



# Historique

2000 —

Ordinateurs portables

Smartphones

Informatique embarquée



# Historique

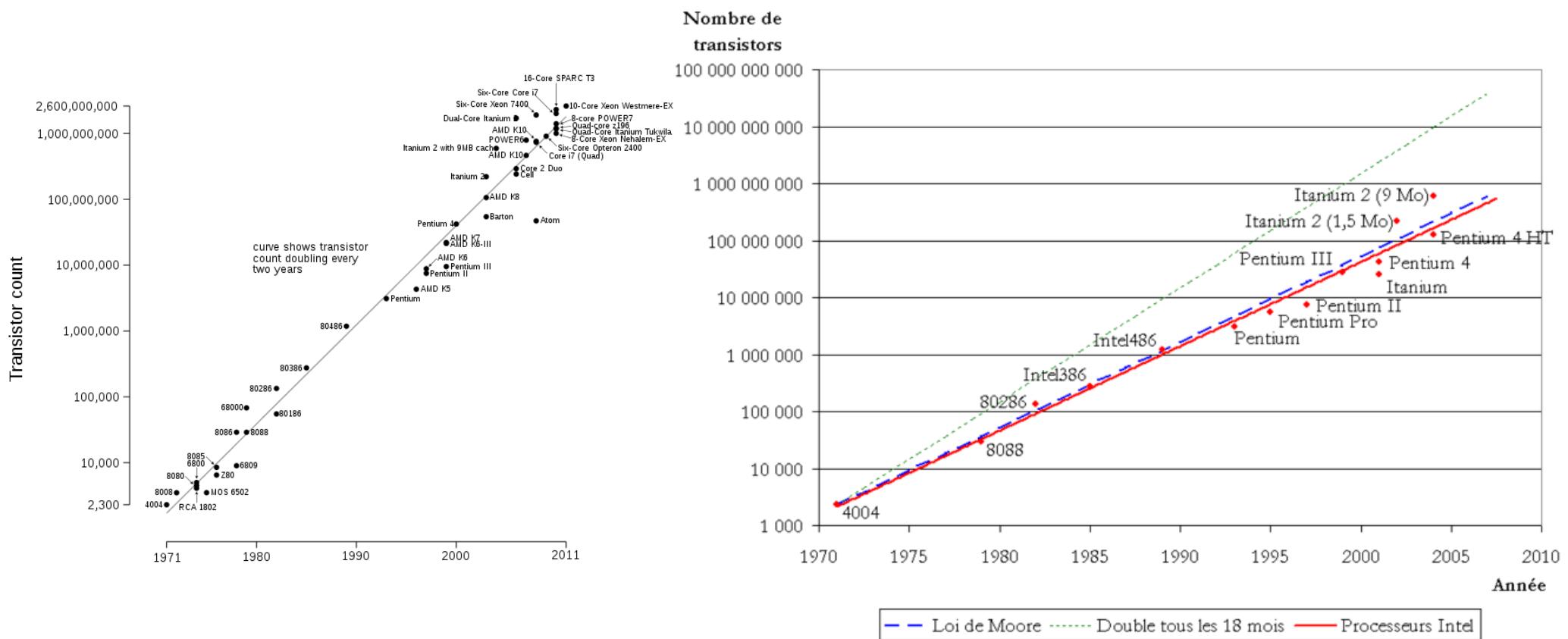
2015 —

Internet des objets (IoT)



# La loi de Moore

Le nombre de transistors sur une puce doublera tous les vingt quatre mois



# La loi de Moore

Elle oblige à modifier les systèmes...

énormément plus de capacités

calcul, mémoire, etc

Les OS ont **énormément** évolué dans les trente dernières années

les dernières avancées ne sont pas nécessairement visibles à l'utilisateur final...

# La loi de Moore

Plus d'augmentation de la densité mais  
augmentation du nombre de cœurs...

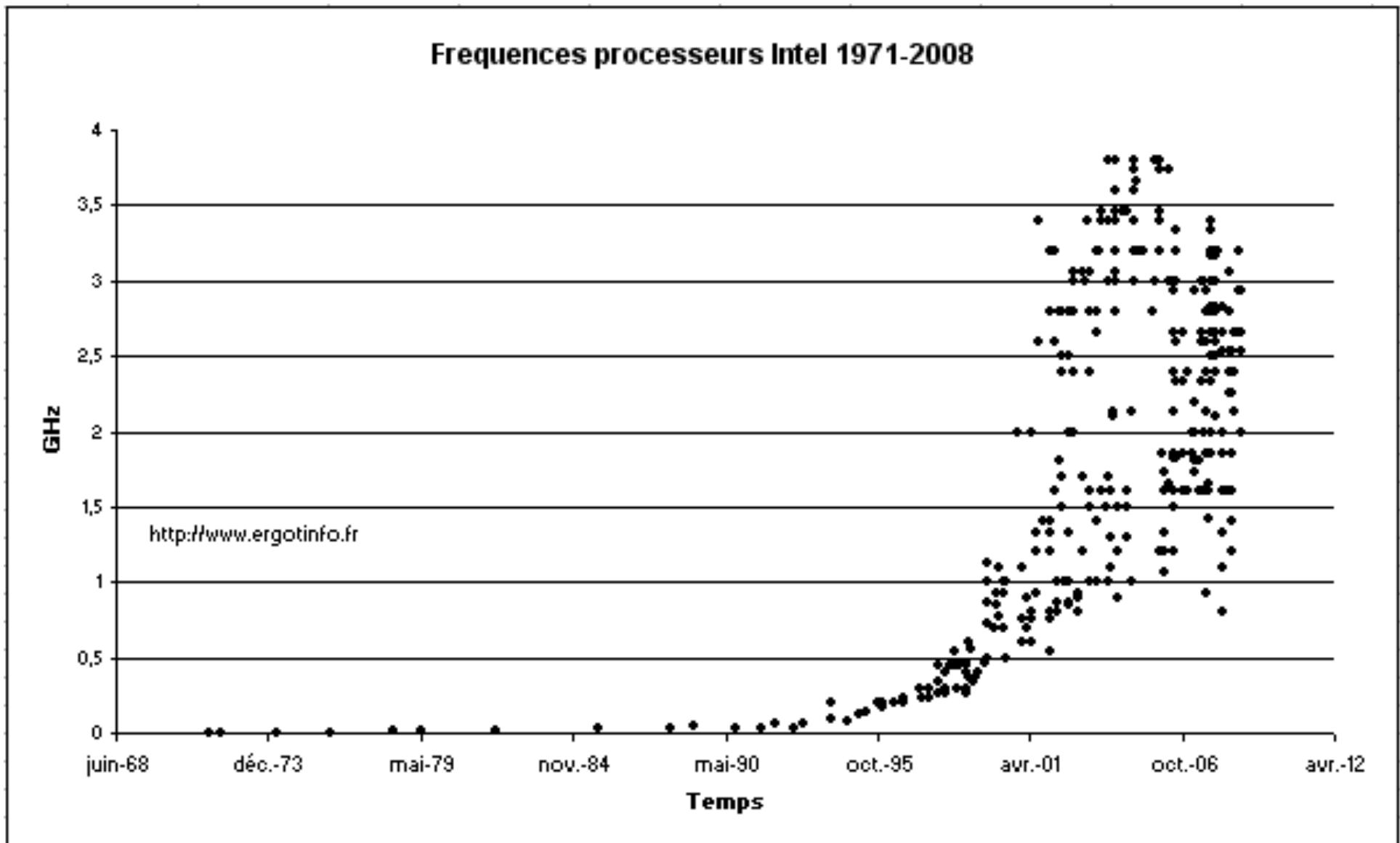
encore de nouveaux OS  
capacité mémoire

pour rappel années 80 quelques Ko/Mo, année 2017,  
en Go (facteur  $\sim 10^3$ )

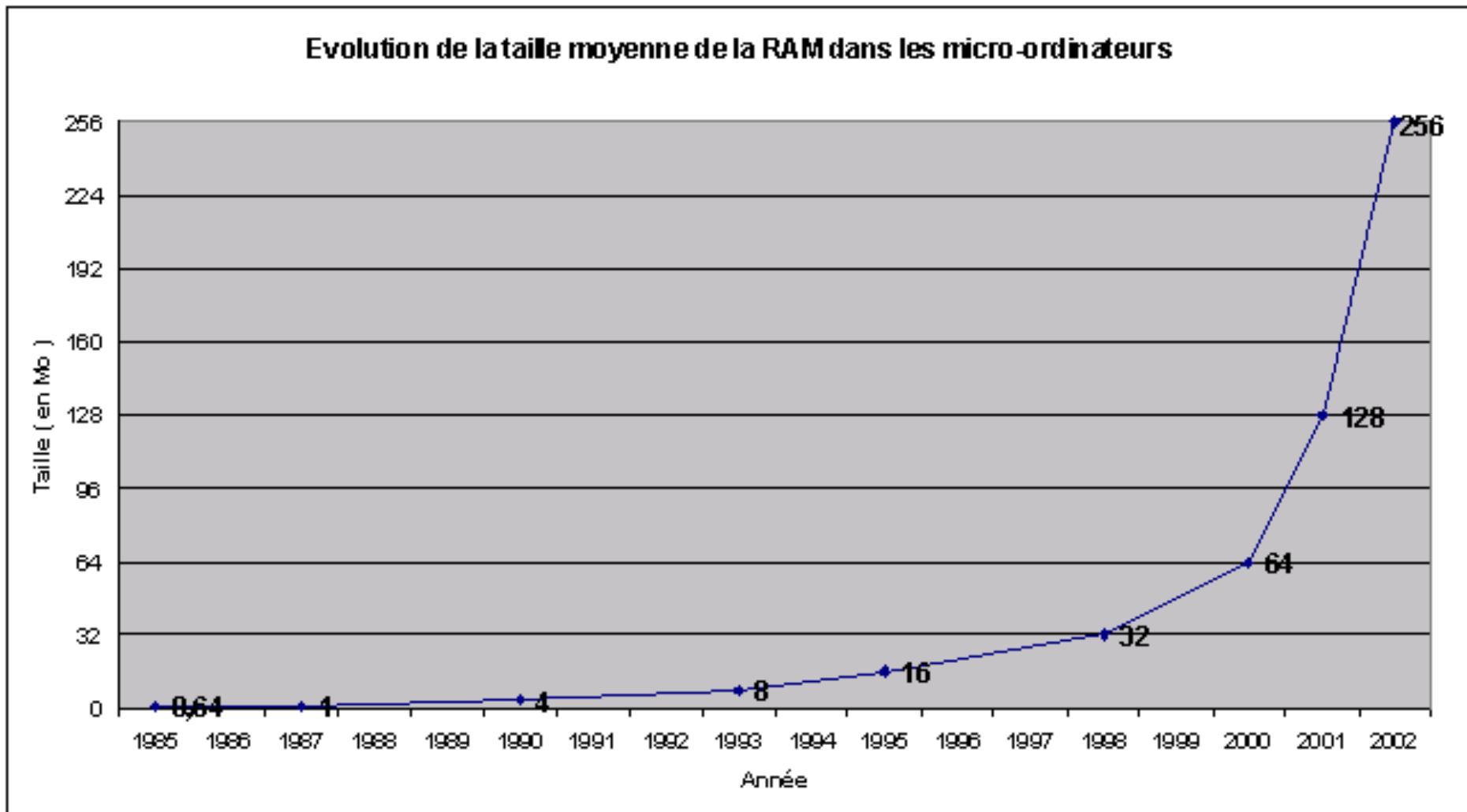
capacité stockage permanent

pour rappel quelques 100Mo en 80, année 2017, en  
To (facteur  $\sim 10^4$ )

# Évolution

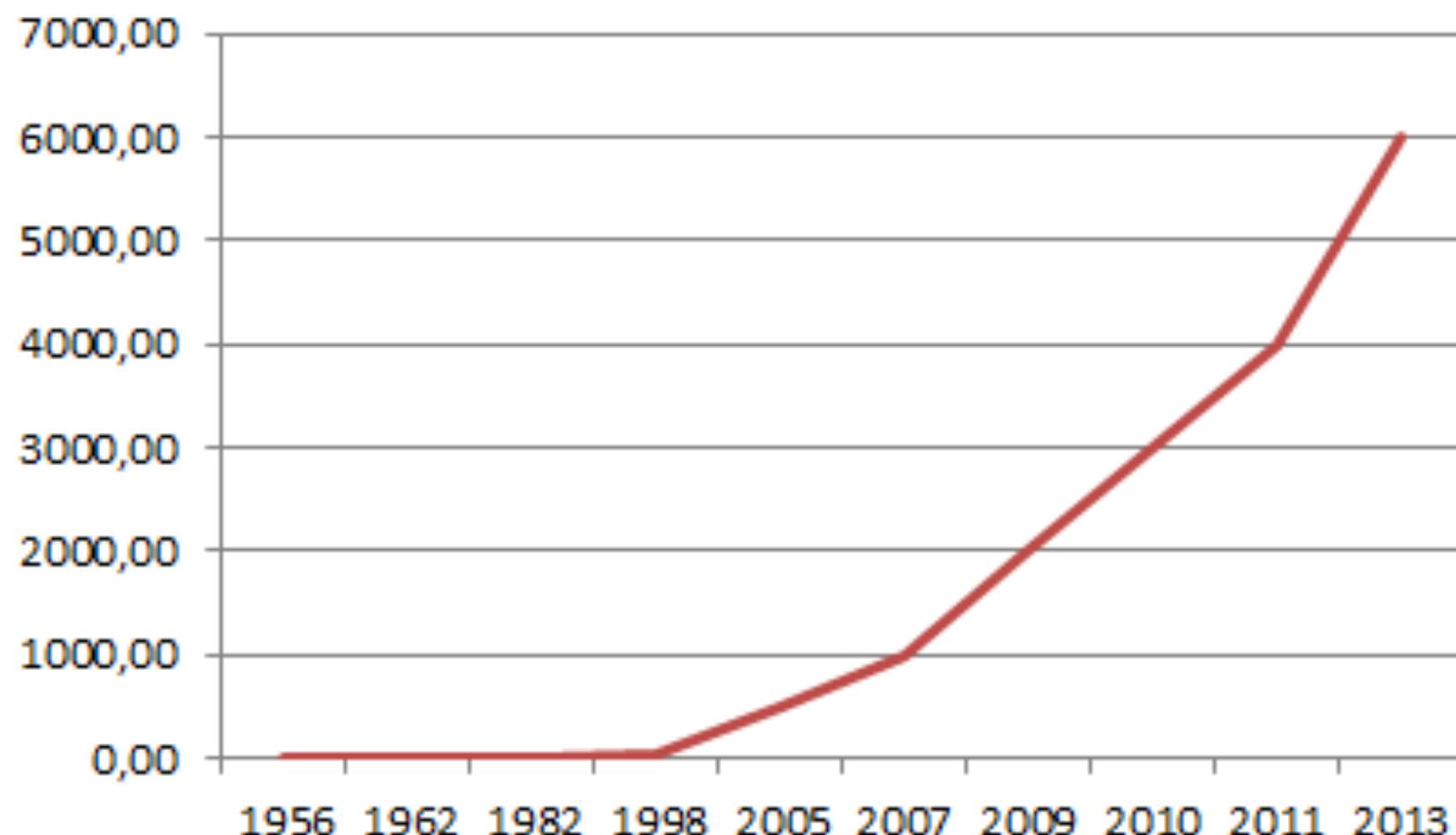


# Évolution



# Évolution

## Évolution de la capacité de stockage



# Une pierre angulaire — MULTICS

MIT + Bell (AT&T) + GE

écrit en langage de haut-niveau PL/1

temps partagé

shell

traitement de texte

terminaux distants

idée : offrir un accès à tous les habitants de Boston

Fichtre : un minitel américain!

computer grid (tiens, tiens...)

# Une pierre angulaire — MULTICS

Technologies

à peu près le cœur de tous les OS d'aujourd'hui

- temps partagé
- intégrité / sécurité
- mémoire virtuelle, segmentation
- système de fichiers hiérarchique (un must)
- multi-tâche préemptif

# Une pierre angulaire — MULTICS

A fini par disparaître  
trop difficile à mettre en œuvre  
en pratique assez lourd à l'usage  
mais...une très grande influence sur les OS  
futurs...

Unix est arrivé pour le manger tout cru  
la révolution

# Multics2Unix

En attendant, K. Thompson travailla sur l'écriture d'un jeu pour se faire, il a créé un environnement de développement et un OS

- systèmes de fichiers (s5fs)
- gestion de processus
- périphériques comme fichiers
- shell

Système auto-suffisant nommé Unics / Unix

B a évolué en C et l'on en mesure le succès encore aujourd'hui

# Unix

La décennie 70 a permis d'obtenir le cœur d'Unix tel qu'on le connaît aujourd'hui

la portabilité...

la version 7

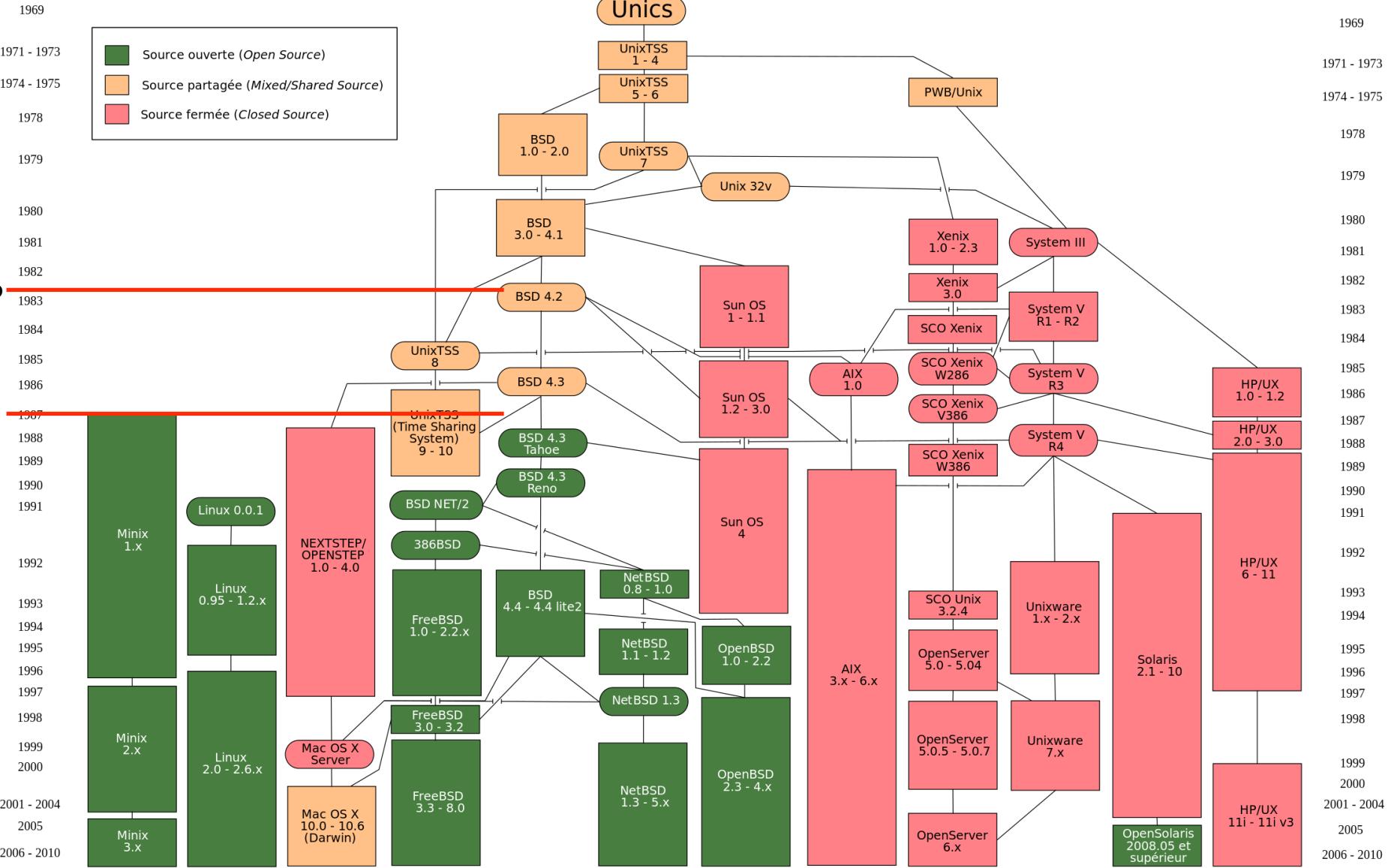
contribution des « utilisateurs »

comme BSD et Linux aujourd'hui favorisé par l'absence de « commerce » dans le projet

# Unix

TCP/IP

Libre



# Xenix

Microsoft et Santa Cruz Operation  
collaborent pour un portage sur Intel x86

Xenix

# La famille BSD

Après un retentissant procès anti-trust l'université de Berkeley  
récupère une licence gratuite  
premier portage sur machine 32bits en 79

csh

tcp/ip

sockets

ffs

Robustesse...

BSDI créée pour commercialiser BSD en 1994

# La famille System V

Bell Labs commercialise Unix

System III

System V

SVR2

SVR3 (1987)

IPC, Streams, RFS, shared libs...

# Sun OS

Un système Unix très important

1982 Bill Joy quitte Berkeley pour fonder Sun Microsystems

NFS

Yellow pages

vfs

automount



# GNU/Hurd

Richard Stallman (Torvalds avant l'heure)  
souhaite créer un système complètement  
affranchi du secteur privé

de très nombreuses applications et briques  
diverses sont écrites dans cet esprit

l'écriture du noyau Hurd est entreprise mais trop  
tard, Linux emportera le morceau...

sans doute à la grande fureur de Stalman

# Petit retour en arrière : 1981

Apparition du premier PC (IBM)

un retour en arrière pour tous les spécialistes

MS/DOS...

mono-\* (programmation, utilisateur, ...)

mais a lancé l'informatique pour tous...

une démocratisation qui a changé le monde

# L'arrivée massive des $\mu$ -processseurs

Milieu des années 80

massification de l'équipement en ordinateurs

domestiques en particulier

apparition d'écrans graphiques

EGA, CGA, VGA

GUI et multi-fenêtrage

# L'arrivée massive des $\mu$ -processseurs

Gros ordinateurs parallèles  
agrégation de nombreux  $\mu$ -processseurs  
gestion de ces ressources de calcul



# Réseau

Le développement des réseaux  
apparition des systèmes distribués  
fichiers partagés



# Le futur en 1985

De nombreux projets tentent de s'affranchir d'Unix (déjà)

Chorus

MACH

Minix

Steve Jobs créé NeXTstep

on en mesure encore aujourd'hui la portée

MacOS c'est NeXTstep! iOS c'est MacOS...

MacOS : MACH+BSD/propriétaire

Objective-C, Display PostScript, des innovations à la pelle!

# Années 1990 : un tournant

Tim Berners-Lee crée le premier navigateur web  
et au-delà le web

Linus Torvalds transforme Minix en Linux

Linux est pris en main par la grande communauté  
des informaticiens/internautes

une victoire écrasante qui a proclamé la  
disparition de nombreux systèmes

...et figé la recherche sur les OS

# La victoire de Linux

Linux est disponible sur presque toutes les plateformes  
y compris les plus exotiques...



# Windows

Durant ces années, Microsoft a développé son très célèbre système malgré les controverses parfois mauvaise qualité sur beaucoup de plans mais la loi du marché est forte...

Dernier avatar

Windows 10

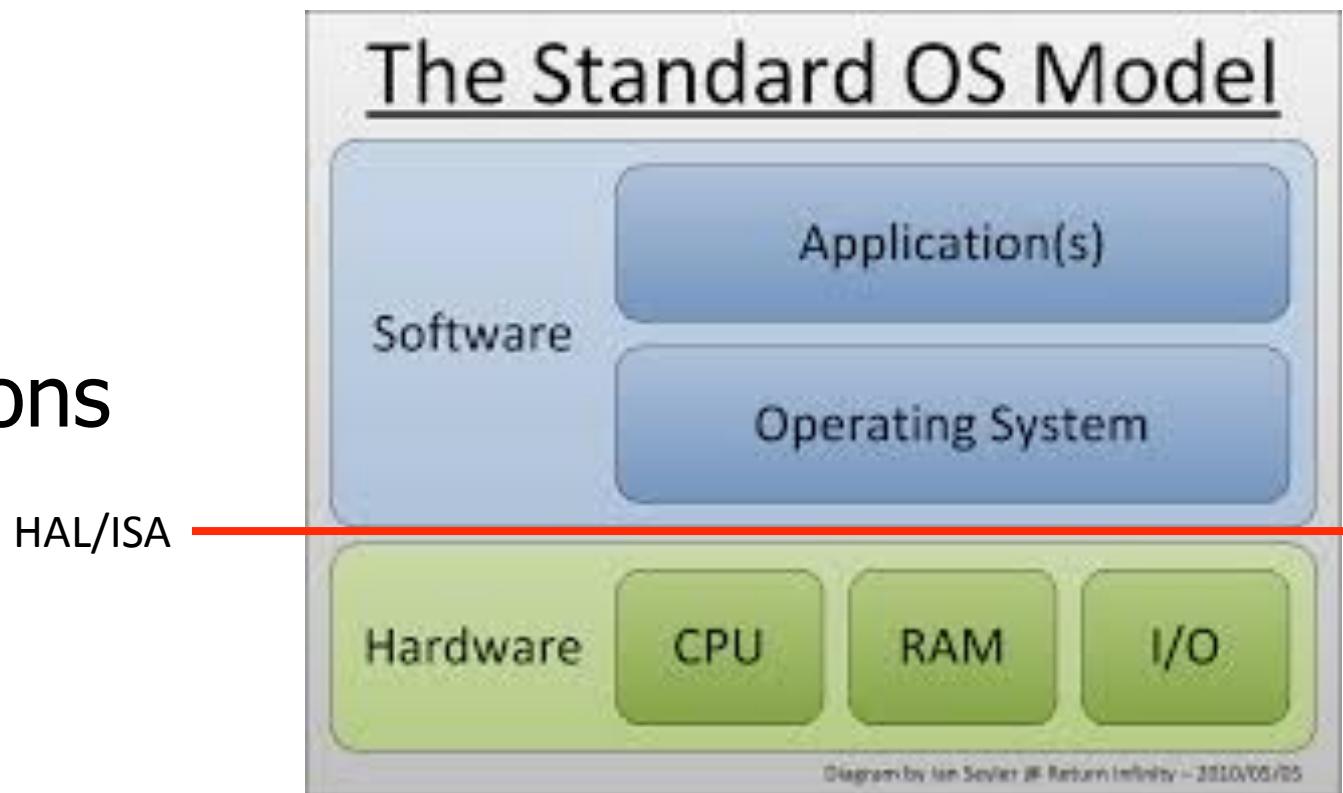


# Rappels et Concepts de base

# Architecture système

Repose sur l'articulation de trois composants principaux :

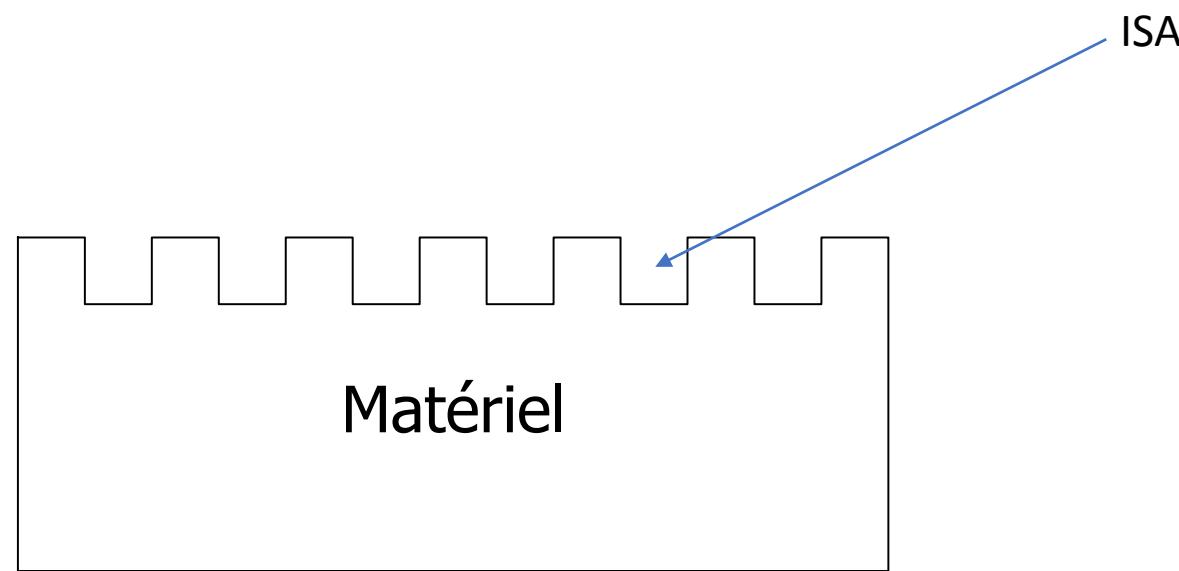
- le matériel
- le noyau
- les applications



# ISA

ISA : Instruction Set Architecture

« API » : hardware/software, une architecture abstraite



# ISA

Différents modes d'exécution  
en général au moins deux (x86 : 4 rings)

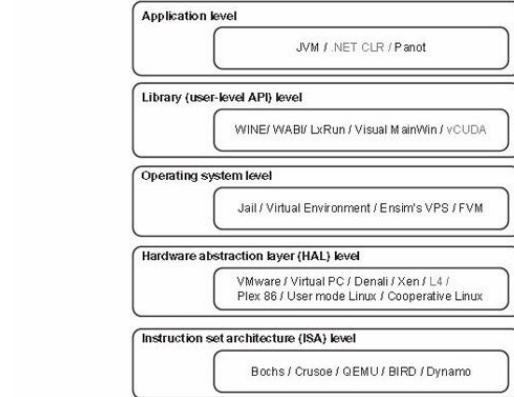
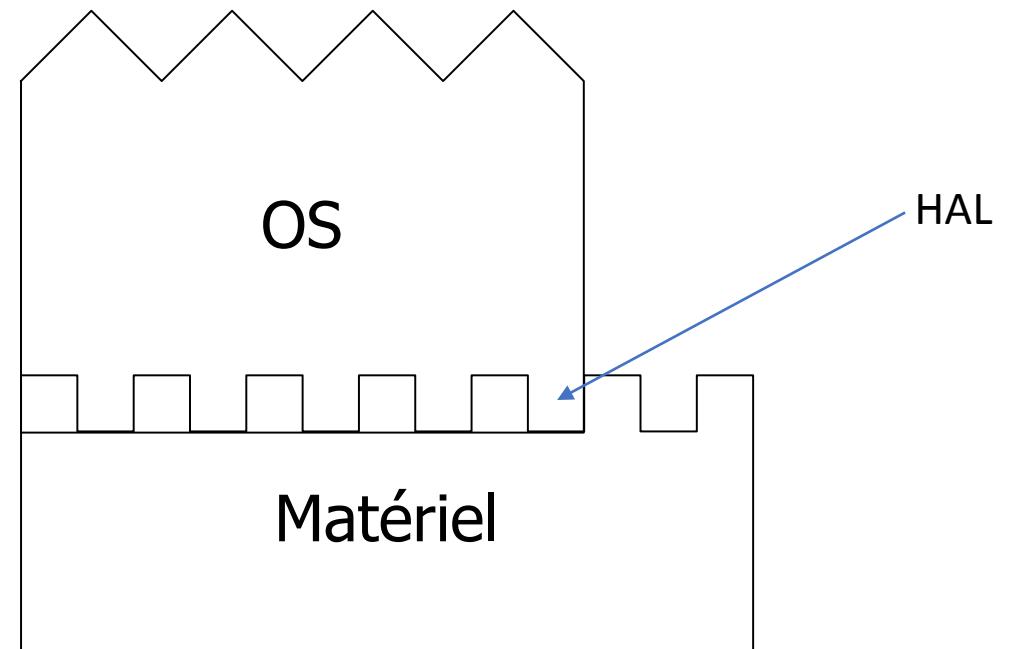
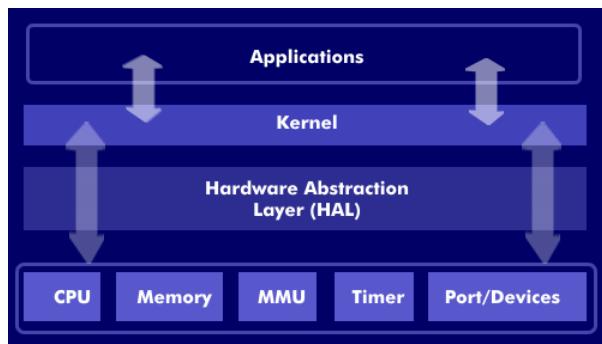
- un pour le kernel
- un pour les applications utilisateurs

Support pour la virtualisation  
Interface avec le matériel

# HAL

## HAL : Hardware Abstract Layer

une couche logicielle permettant de s'affranchir  
de détails hardware...



# Portabilité, compatibilité, interopérabilité

Compatibilité

« temporelle », ou synonyme de portabilité

Portabilité

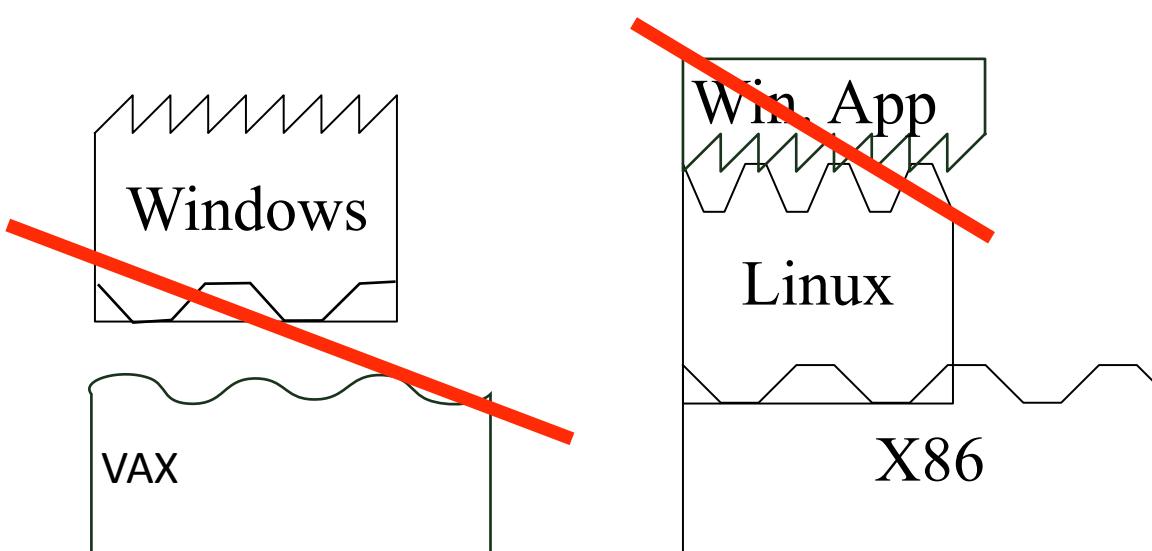
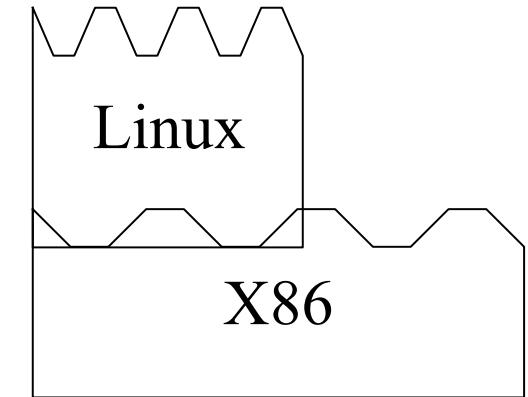
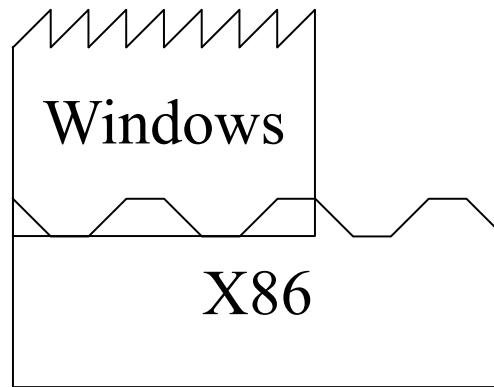
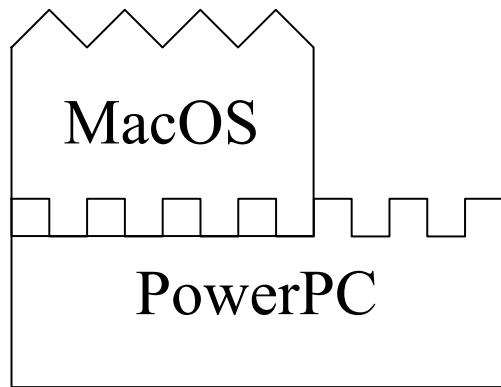
« verticale »

Interopérabilité

« horizontale »



# Portabilité, Compatibilité



# Portabilité des OS

Avec processeurs différents (OS α sur Intel et Motorola)  
au niveau du source

HAL : OSX sur PowerPC et Intel

au niveau du binaire

simulation/émulation

Avec architecture différente (PC x86, Carte x86) :  
BSP : Board Support Package, abstraction de  
spécificités

# Portabilité des applications

Source ?

une question d'API

Binaire ?

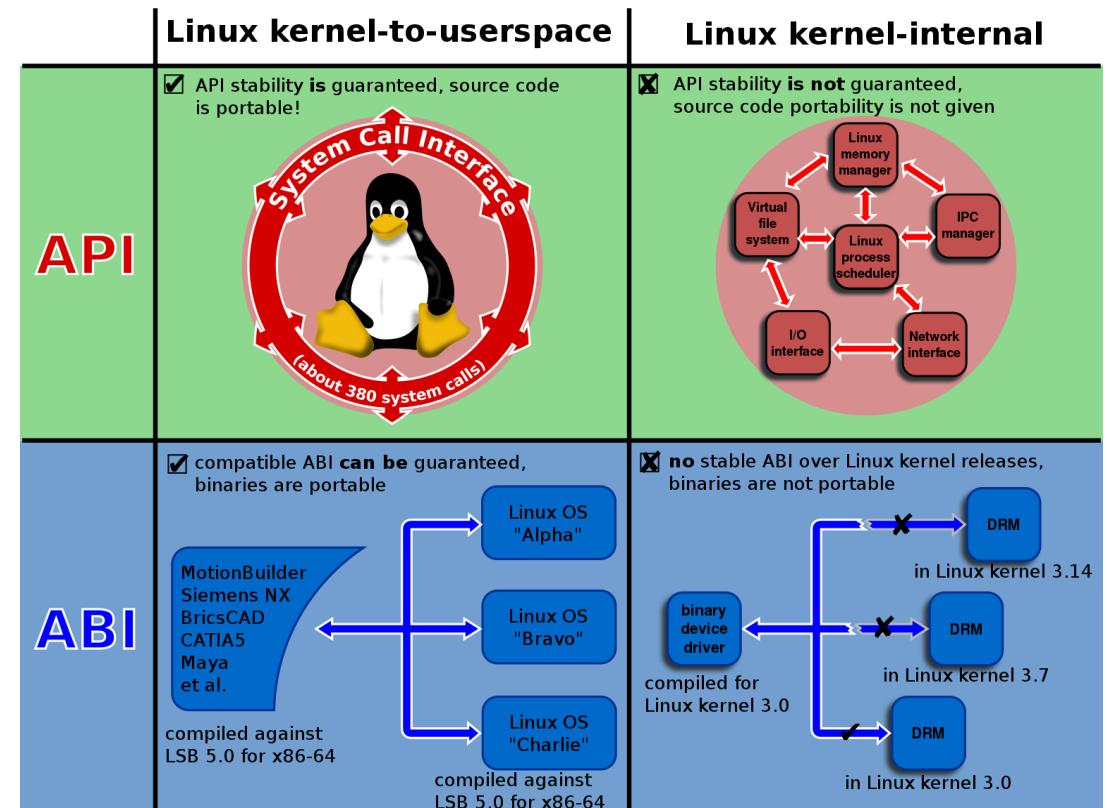
une question d'ABI

Application

Binary

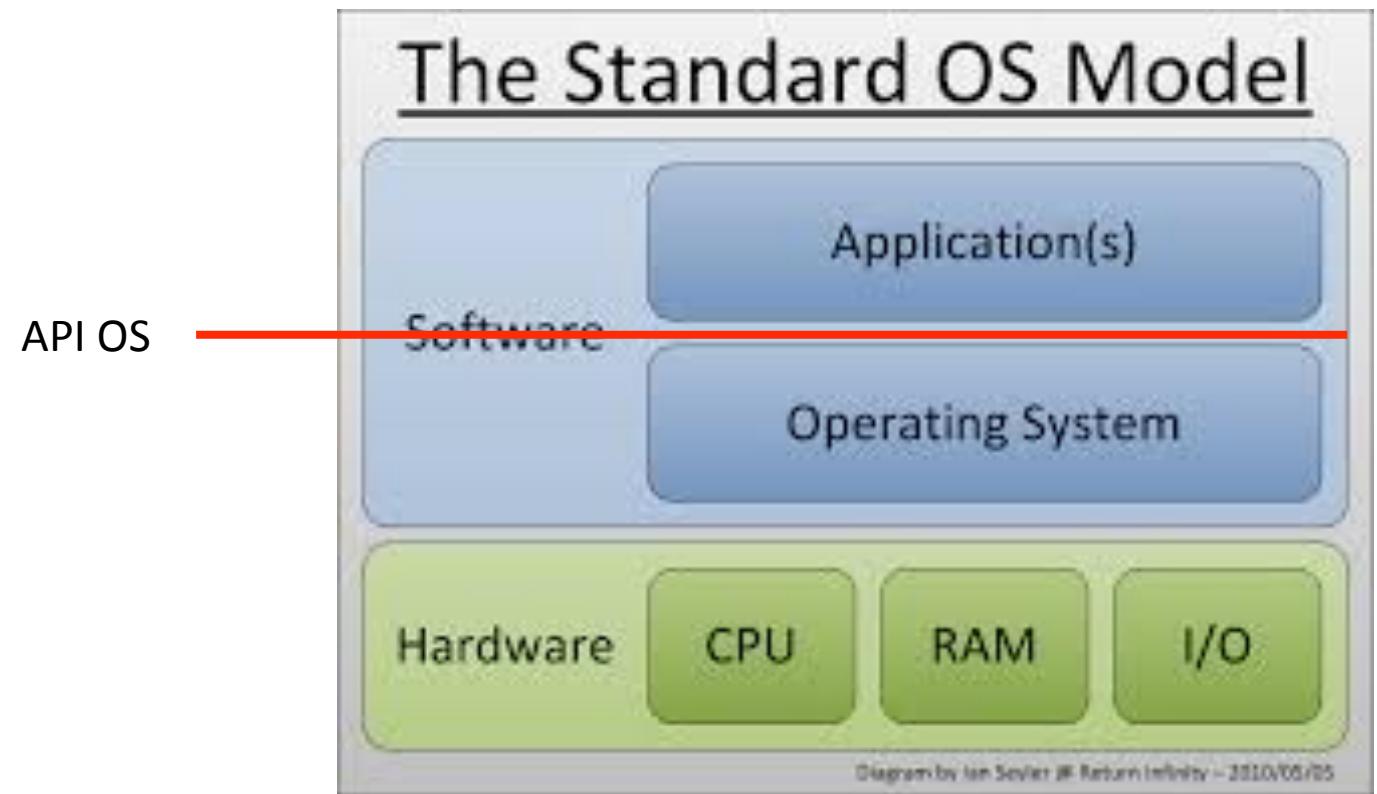
Interface

Ex.: flinux?



# API

API : Application Programming Interface



# API

C'est la définition des services offerts par le système vus comme des fonctions pouvant être appelées dans un langage

des manuels de références

par exemple man 2 et 3 d'Unix

des prototypes de fonctions, bibliothèques de code binaire

Ex. : POSIX

Portable Operating System Interface (for uniX)

Permet un portage au niveau du code source

# ABI

Portabilité de binaires sur OS différents mais offrant la même API sur des plateformes compatibles (même ISA)

# ABI

Comment passe-t'on des paramètres ?

les compilateurs génèrent du code pour une ABI

Comment fait-on appel à un service système ?

numérotation des appels systèmes

Comment invoquer une bibliothèque dynamique ?

Comment invoquer un handler de signal ?

Quelle est la structure d'un exécutable ?

ELF, mach-O, COFF, a.out, PE

etc.

# Autres problèmes...

La structure et l'organisation des fichiers ?

où créer des fichiers temporaires ?

où sont les commandes externes ?

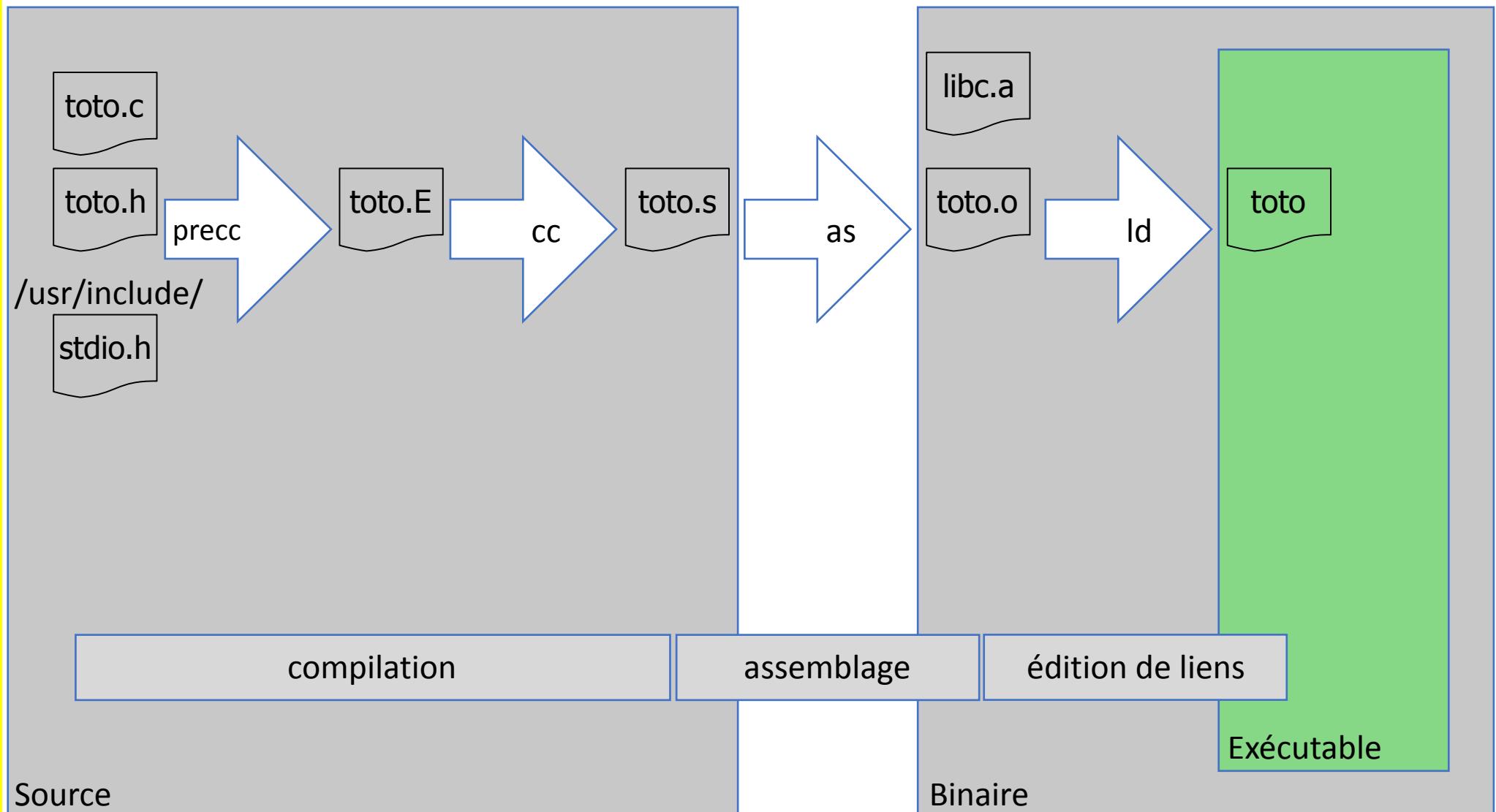
où sont les bibliothèques ?

Quel est l'espace mémoire attribué à une exécution ?

L'administration d'un système ?

packaging: .rpm RedHat, .deb Debian, .pkg MacOS, .msi  
Windows

# Compilation (native)



# Exemple (source)

```
#include <stdio.h>
int main() {
    printf("Hello\n");
}
```

# Exemple (assembleur)

Intel  
MacOS  
llvm-gcc

```
.section    __TEXT,__text,regular,pure_instructions
.macosx_version_min 10, 13
.globl     _main
.p2align   4, 0x90
_main:      ## @main
            .cfi_startproc
## BB#0:
Lcfi0:      pushq    %rbp
Lcfi1:      .cfi_def_cfa_offset 16
Lcfi2:      .cfi_offset %rbp, -16
            movq    %rsp, %rbp
            .cfi_def_cfa_register %rbp
            subq    $16, %rsp
            leaq    L_.str(%rip), %rdi
            movb    $0, %al
            callq   _printf
            xorl    %ecx, %ecx
            movl    %eax, -4(%rbp)      ## 4-byte Spill
            movl    %ecx, %eax
            addq    $16, %rsp
            popq    %rbp
            retq
            .cfi_endproc

L_.str:     .section    __TEXT,__cstring,cstring_literals
            ## @.str
            .asciz    "Hello\n"

.subsections_via_symbols
```

# Exemple (assembleur)

Intel  
Ubuntu  
gcc

```
.file      "toto.c"
.section   .rodata
.LC0:
.string    "Hello"
.text
.globl    main
.type     main, @function

main:
.LFB0:
.cfi_startproc
pushq    %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq    %rsp, %rbp
.cfi_def_cfa_register 6
movl    $.LC0, %edi
call    puts
popq    %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc

.LFE0:
.size     main, .-main
.ident   "GCC: (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4"
.section  .note.GNU-stack,"",@progbits
```

# Exemple (assembleur)

Intel  
Windows  
gcc

```
.file      "toto.c"
.def       __main;
.scl       2;
.type      32;
.undef
.section .rdata,"dr"

LC0:
.ascii  "Hello\0"
.text
.globl   __main
.def     __main;
.scl    2;
.type   32;
.undef

_main:
pushl    %ebp
movl    %esp, %ebp
andl    $-16, %esp
subl    $16, %esp
call    __main
movl    $LC0, (%esp)
call    __puts
leave
ret
.ident  "GCC: (tdm-1) 4.9.2"
.def     __puts;
.scl    2;
.type   32;
.undef
```

# Exemple (exécutable)

dans les trois cas un exécutable différent  
ce n'est même pas le même système et donc pas la  
même ABI ni API

## MacOS

```
Mach-O 64-bit executable x86_64
```

## Ubuntu

```
ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared  
libs), for GNU/Linux 2.6.24, BuildID[sha1]=7023dfd9e4ac3387919290ac20ee8a23e66fa386, not  
stripped
```

## Windows 7

```
PE32 executable (console) Intel 80386, for MS Windows
```

# Statique vs Dynamique

Le code source ne contient pas la définition de la fonction `printf`

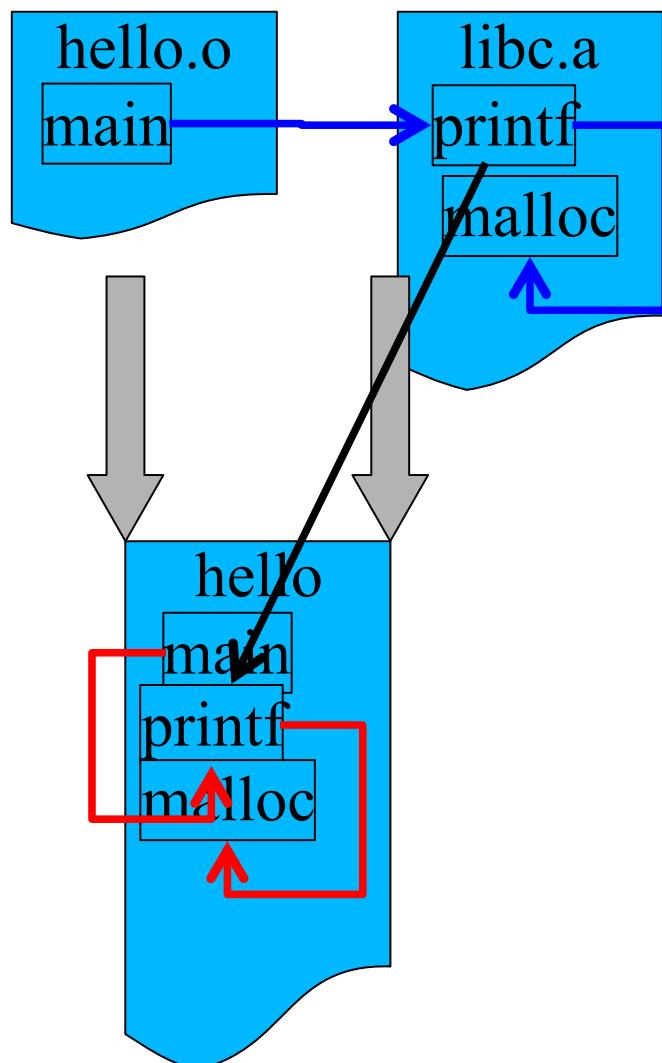
en dynamique les exécutables non plus (par défaut, aujourd'hui)

en statique les exécutables sont clos (rarement utilisé aujourd'hui)

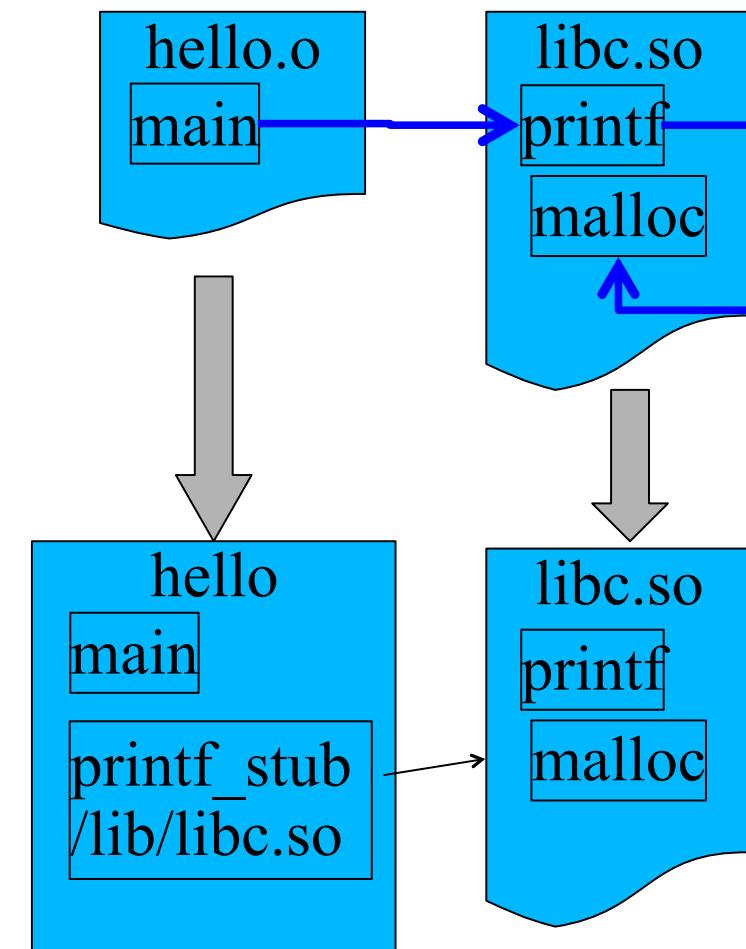


# Librairies

Statiques



Dynamiques



# Bibliothèques dynamiques

Lien avec la bibliothèque libc.so (entre autres...)

```
$ cc -o toto toto.c
$ file toto
toto: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.24,
BuildID[sha1]=7023dfd9e4ac3387919290ac20ee8a23e66fa386, not
stripped
$ size toto
      text        data         bss      dec      hex filename
    1192          560           8    1760    6e0  toto
```

# Bibliothèques statiques

Lien avec la bibliothèque libc.a (entre autres...)

```
$ cc -o toto toto.c -static #ou -Wl,-static
$ file toto
toto: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 2.6.24, BuildID[sha1]=bb6748732f6d7af5f915522eb01ad61106e158f2, not stripped
$ size toto
      text      data          bss      dec      hex filename
 783139      7532        9600  800271    c360f      toto
```

# Bibliothèques

## Statiques

duplique le code dans l'exécutable (sur disque et en mémoire)

avantages : chargement et exécution simple, pas de dépendances

inconvénients : taille plus importante (assez notable), ne bénéficient pas des mises-à-jour des bibliothèques...

## Dynamiques

pas de duplication des codes des fonctions (ni sur disque, ni en mémoire)

avantages : taille réduite, bénéfice des mises-à-jour

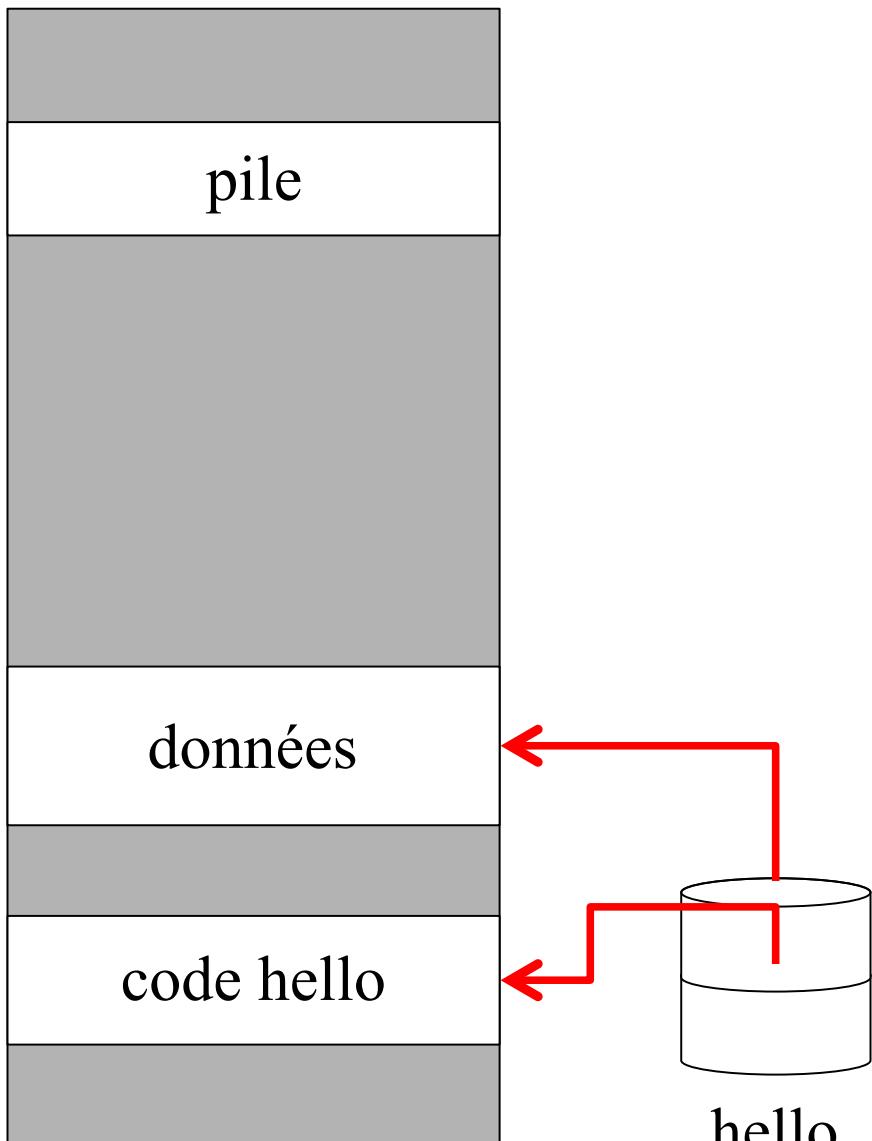
inconvénients : nécessiter d'avoir les bibliothèques disponibles, dépendance entre code et bibliothèques, chargement plus complexe

nécessite d'avoir un code de nature particulière : PIC (Position Independent Code)

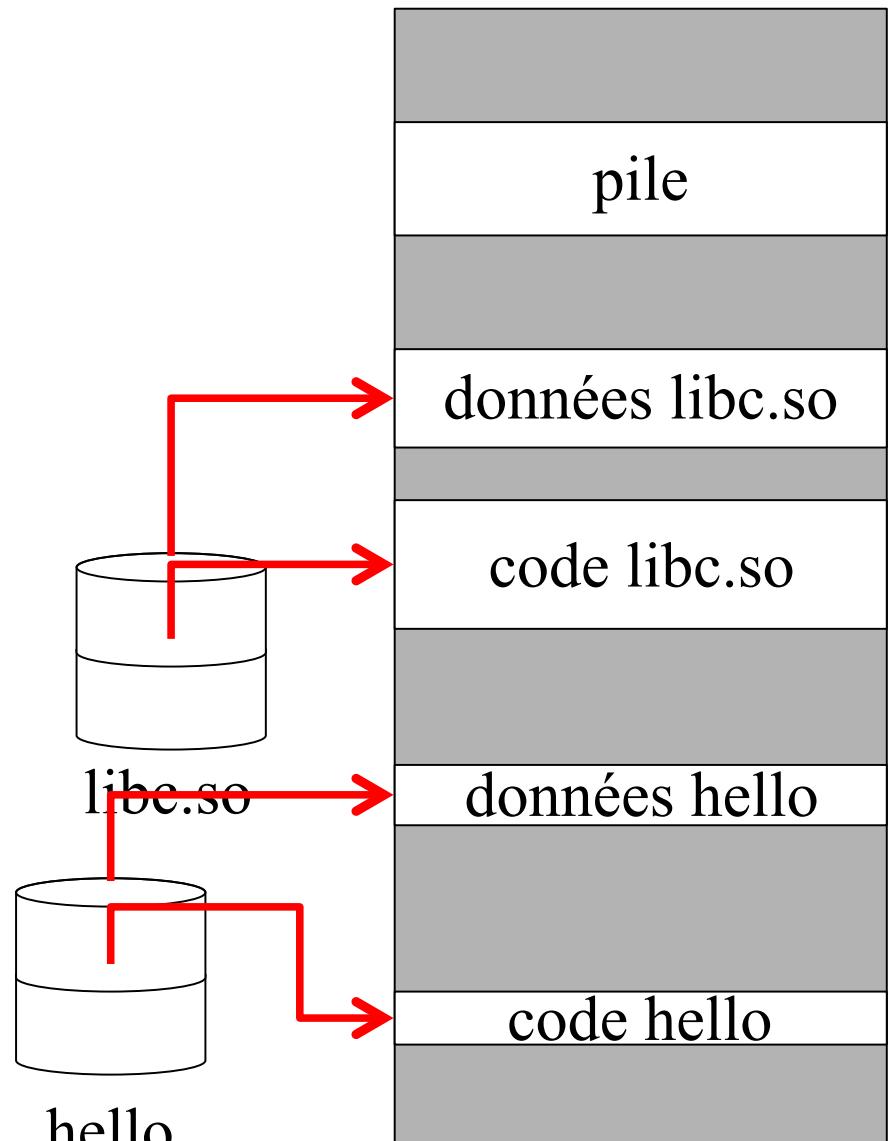
option -fpic de gcc

# Librairies

Statiques



Dynamiques



# Grandes fonctions d'un OS

Gestion Mémoire

Gestion des exécutions

Gestion du matériel

Gestion des fichiers



# Gestion mémoire

Allocation/Déallocation

Partage

Protection



# Services mémoire (\*nix)

Explicites (sur requête du code)

- Le tas : malloc/new
- La mémoire partagée : mmap
- La protection : mprotect
- Le comportement système : mlock
- exec

Implicites

- chargement de page, défauts de page
- gestion de la pile
- violation des droits

# Régions / Segments

Un segment ou une région :

espace mémoire contigu uniforme du point de vue  
de sa gestion (droits,...)

brk/sbrk

exec

fork

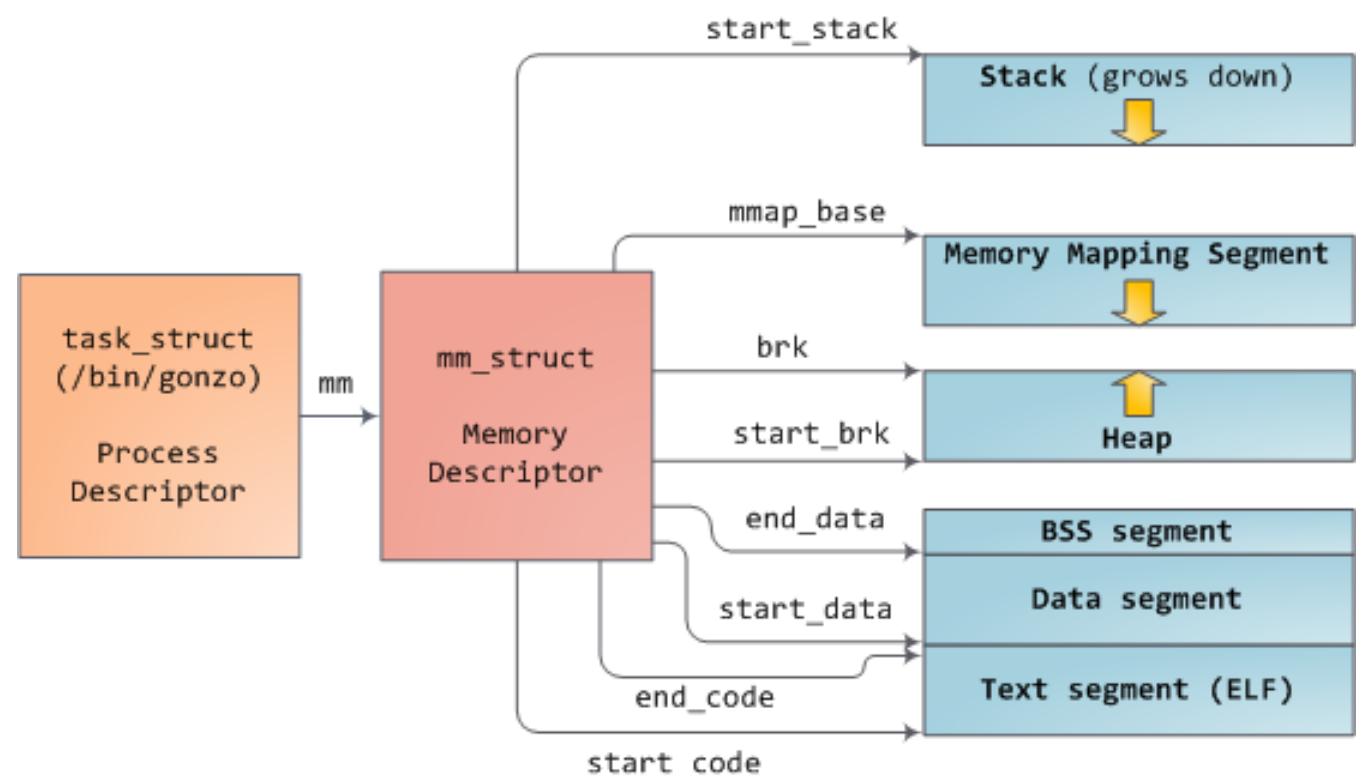
mmap

shmat

# Régions (Linux)

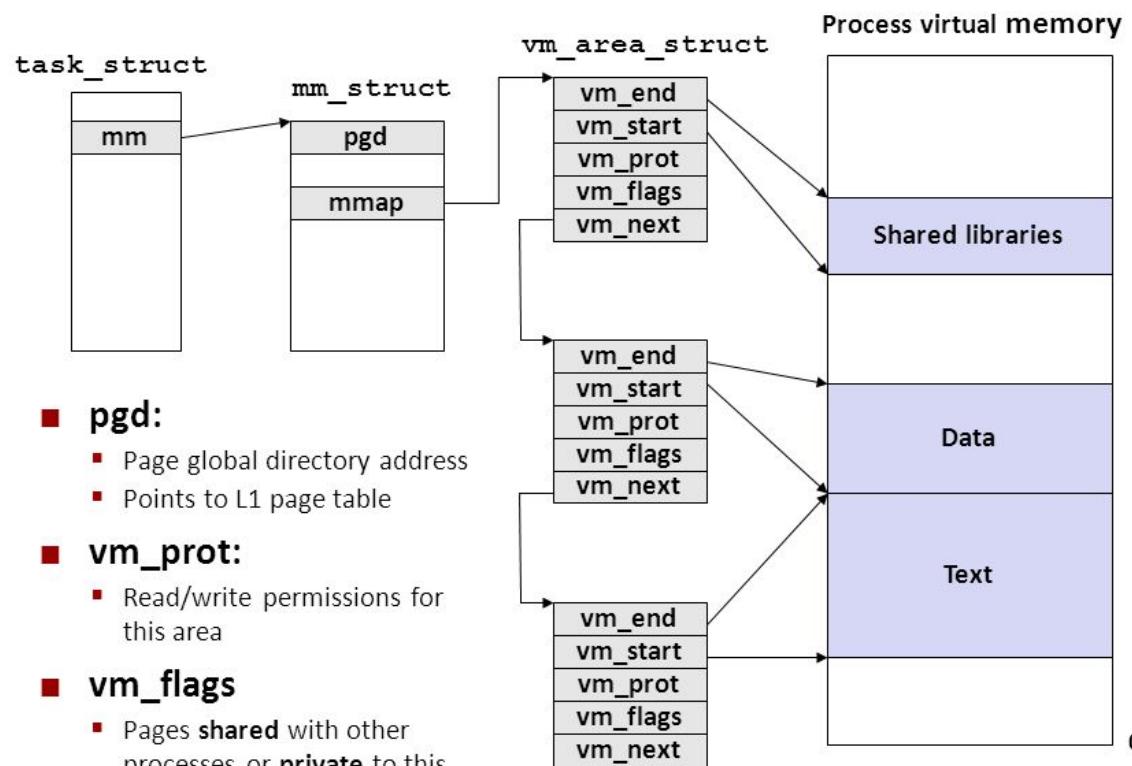
À chaque processus (`task_struct`) est associé la description de son espace d'adressage

`mm_struct`



# Régions (Linux)

Chaque segment de son espace d'adressage est décrit par une structure `vm_area_struct`



# Questions à se poser

Intervalles d'adresses qui peuvent être atteintes ?

Adresses valides ?

Que faire si tentative d'accès à une adresse  
« interdite » ?

Qu'est ce qui est autorisé avec une adresse valide ?

Besoins en termes de gestion des adresses (point  
de vue processus ou système) ?

De quelle mémoire physique dispose-t'on ?

# Évolution de la gestion mémoire

Les premiers temps

pas d'OS donc pas de gestion mémoire ? C'est sûr ? La pile ?  
Les overlays ?

Un OS avec une seule application

pas de vrai besoin sauf celui de l'isolation entre les deux parties...

Un OS et plusieurs applications

les besoins sont nettement plus compliqués

chargement/déchargement des apps, isolation inter-proc,  
proc/OS, gestion dynamique...

# Besoins complémentaires...

Charger des applications même si l'espace mémoire physique est entièrement occupé  
swap / pagination

Permettre à une application d'utiliser un espace qui dépasse la taille de la mémoire physique

Exploiter une machine qui aurait plus d'espace de mémoire physique qu'un processus n'a d'espace mémoire possible

# Besoins complémentaires...

Support des machines multi-processeurs

SMP (Symmetric Multi-Processor)

tous les CPU accèdent à la même mémoire, de la même manière

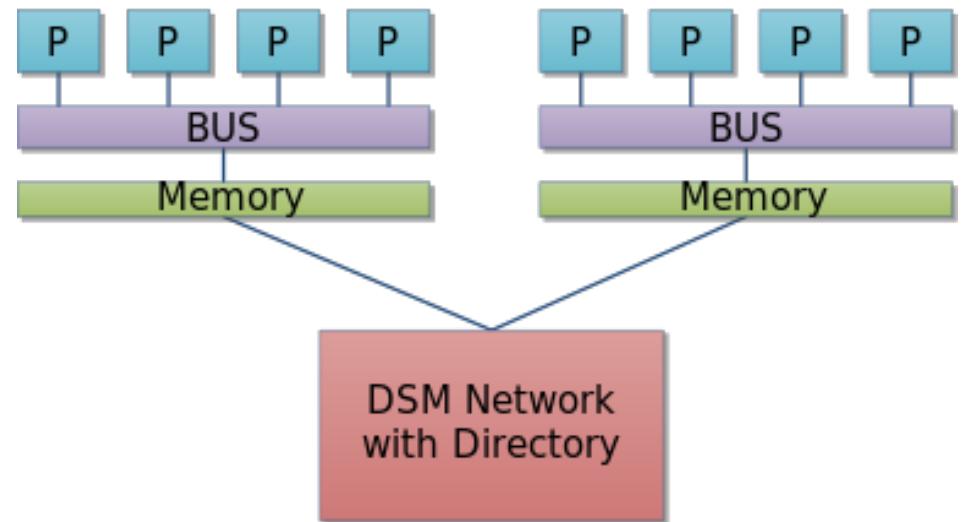
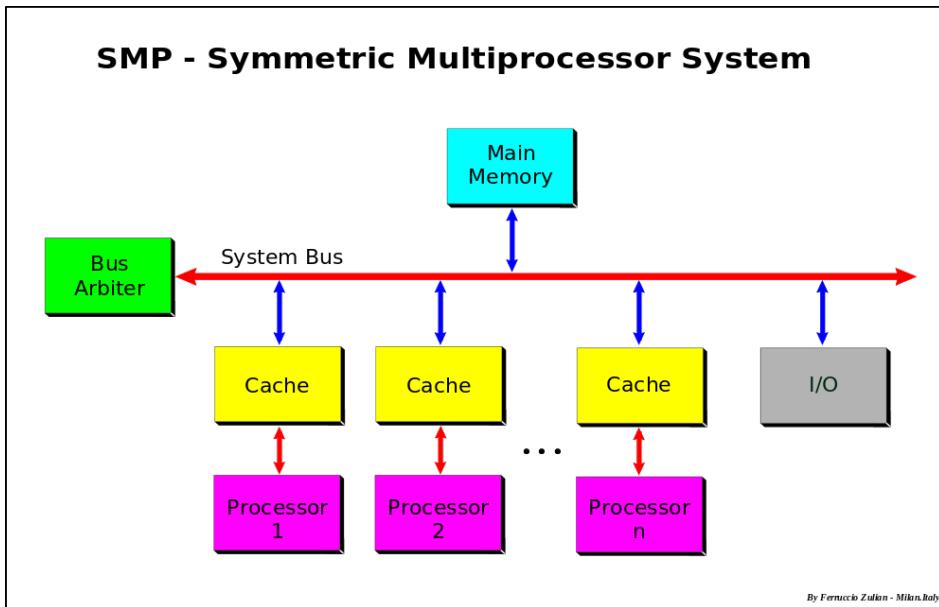
attention aux caches...

NUMA (Non-Uniform Memory Access)

même mémoire mais l'usage n'est pas uniforme  
(accès dépendant des couples bancs/CPU)

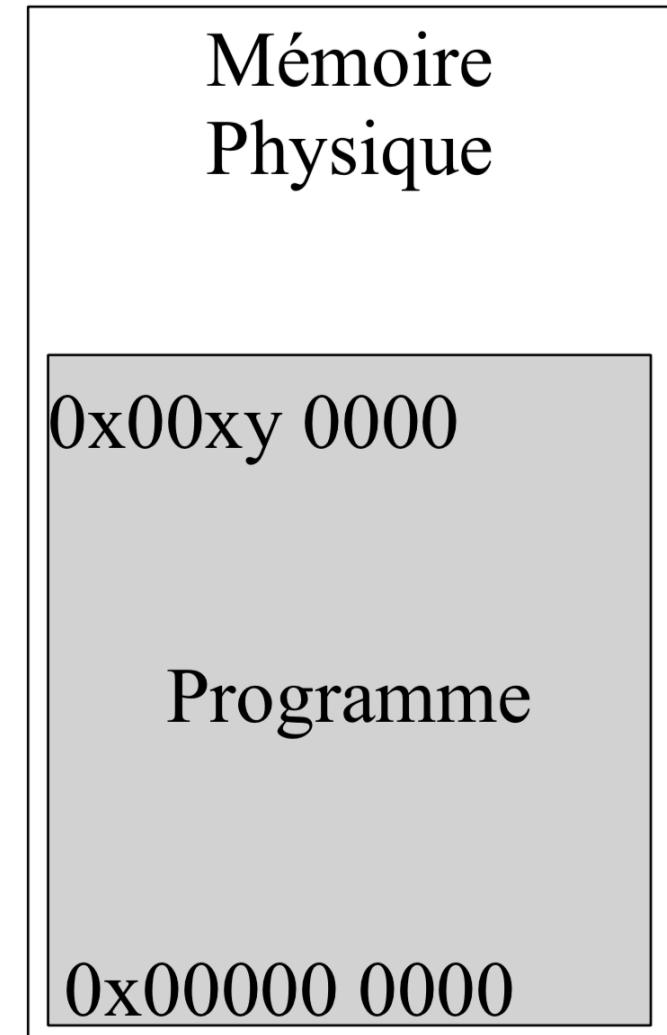
attention au placement des données, etc

# SMP/NUMA



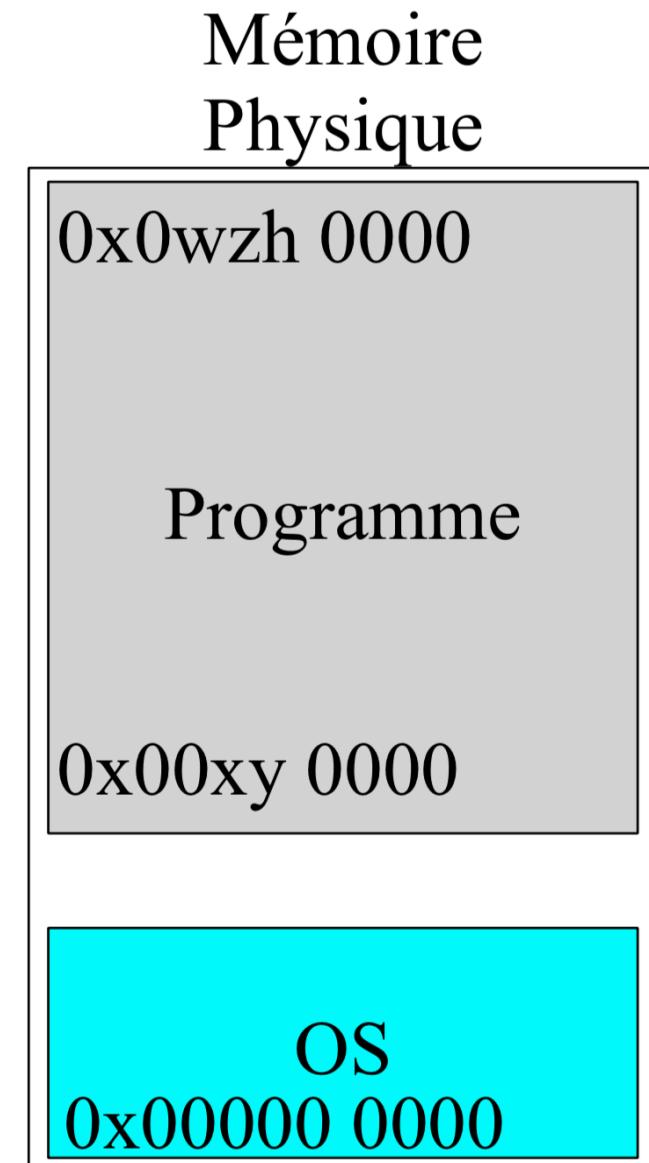
# Cas Mono-tâche sans OS

- Le programme est chargé en mémoire (par ex en 0)
- Son édition de liens a eu lieu en connaissance de cause
- Le programme n'a pas accès à des services de gestion mémoire, mais il peut en implanter si besoin.



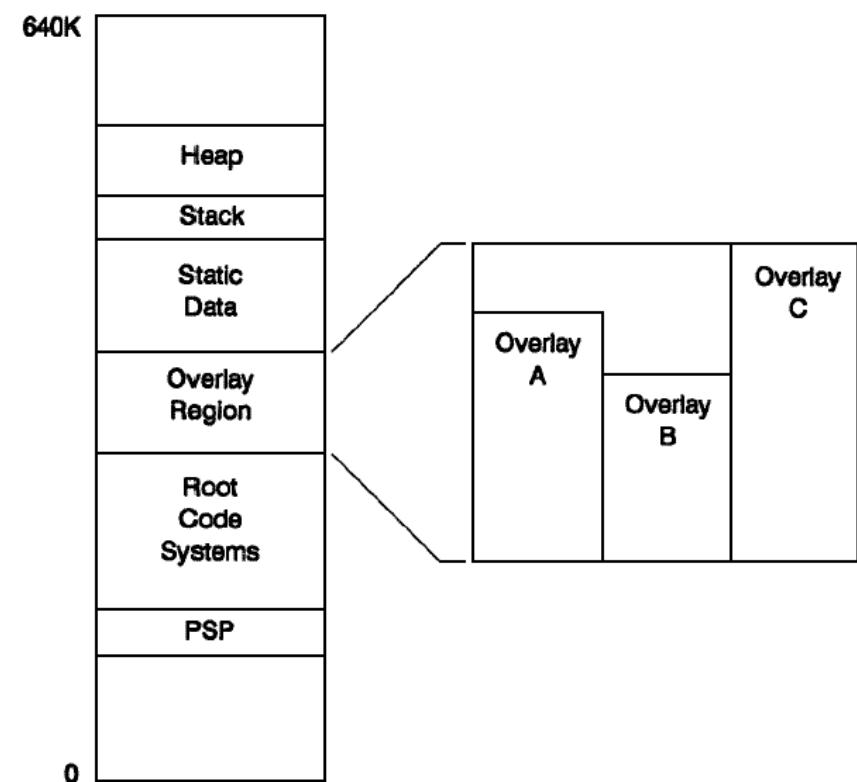
# Cas Mono-tâche avec OS

- Le programme est chargé en mémoire à une adresse tenant compte de la présence de l'OS
- Son édition de liens a eu lieu en connaissance de cause, ou l'OS a effectué un calcul final de "relocation" (relogement)
- En général: accès à de la mémoire physique, et pas d'isolation entre appli et OS.



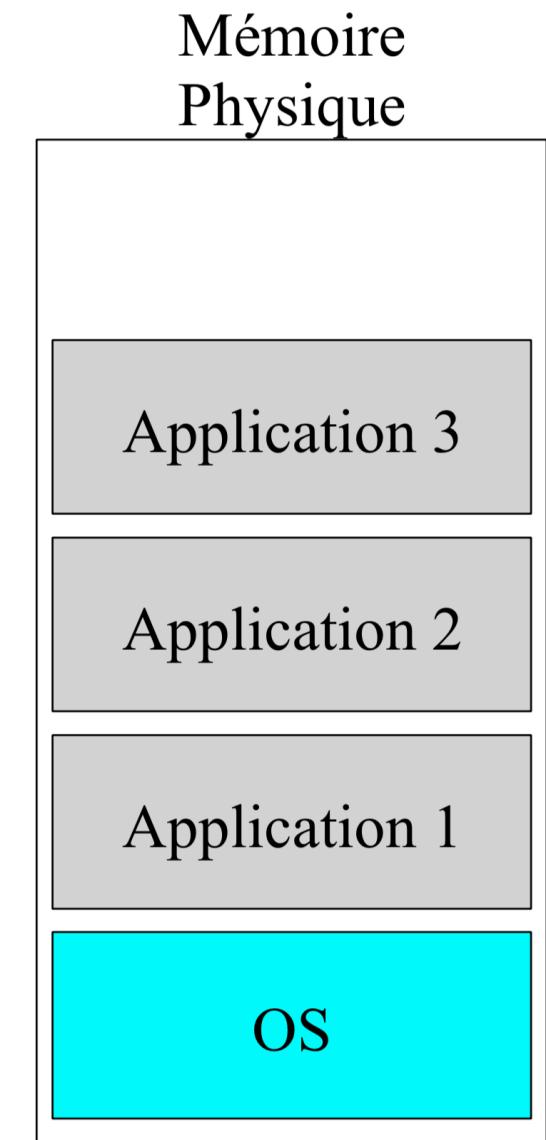
# Overlays

Très difficile d'obtenir l'exécution d'un programme dont la taille dépasse celle de la mémoire physique  
overlays (MS/DOS)



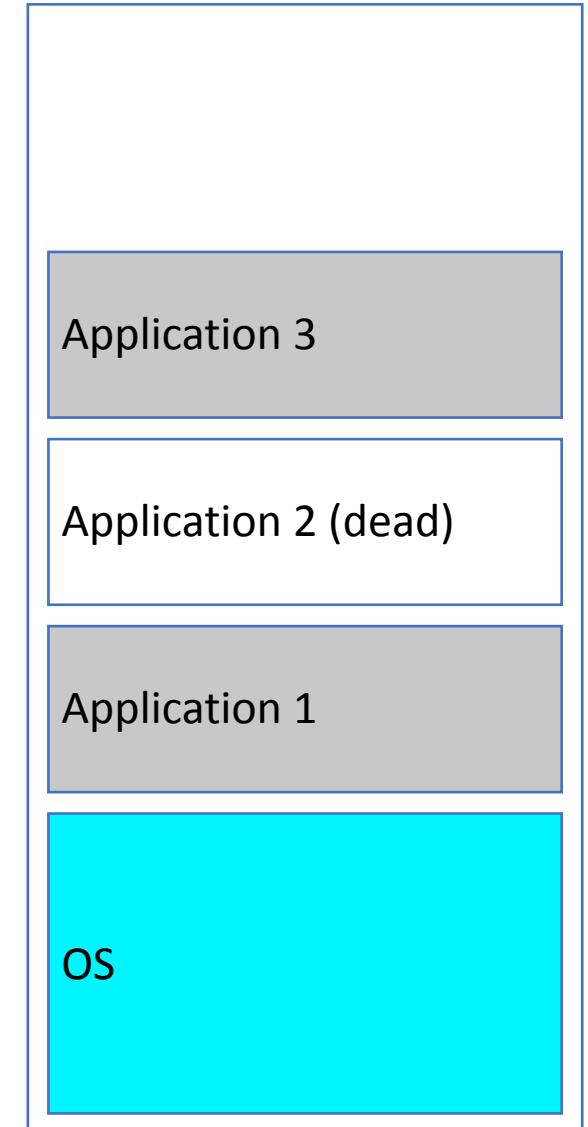
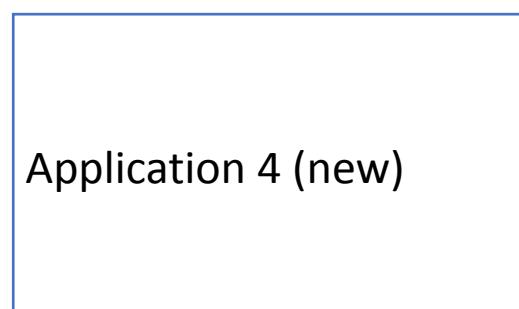
# Cas Multi-tâches avec OS

- L'adresse en mémoire physique est déterminée dynamiquement par l'OS
- Les programmes sont chargés de manière continue (code, données)
- Il peut y avoir isolation
- On peut alors rencontrer des problèmes de fragmentation
- Ex: GCOS6 (versions initiales)



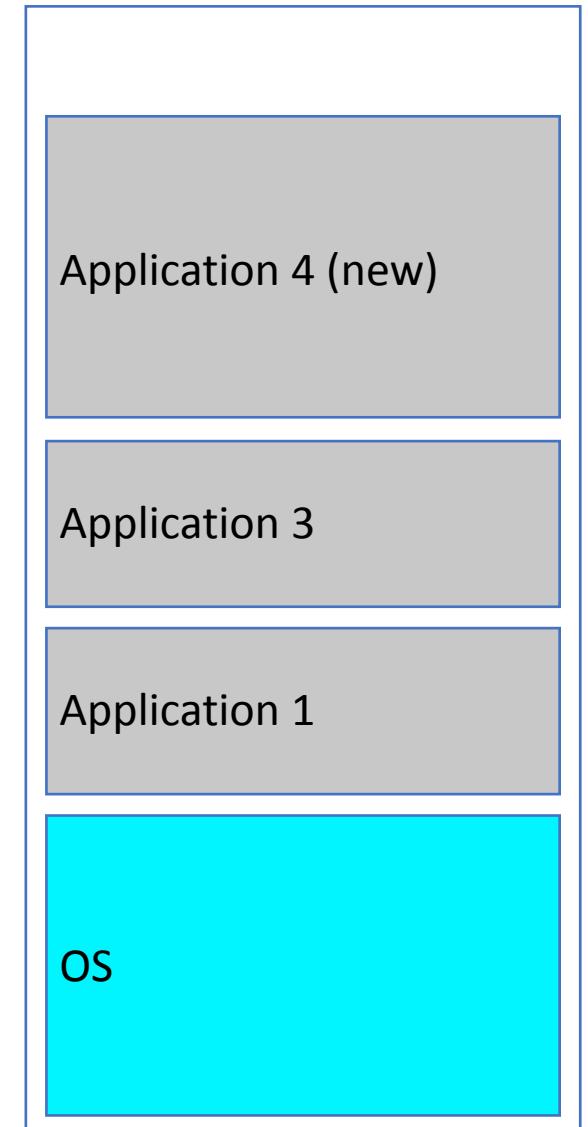
# Problème de Fragmentation

- L'application n°2 de la page précédente s'est terminée
- Même si l'espace total disponible est suffisant on ne peut pas charger l'application n°4 dans une zone continue!



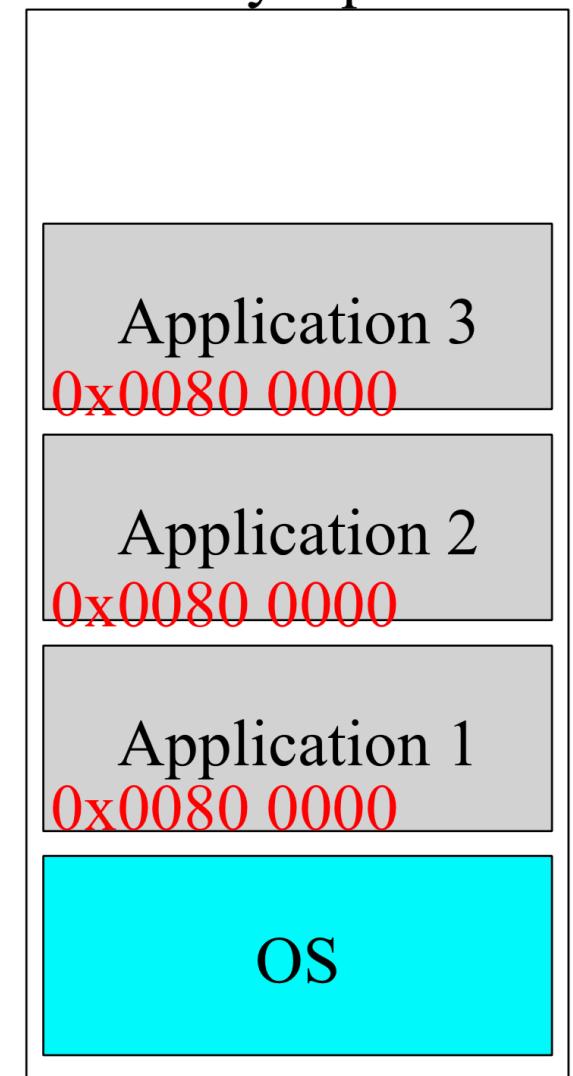
# Problème de Fragmentation

- Arrêter 3, relancer 3 et 4.
- ou figer 3, déplacer 3, réchauffer 3 et lancer 4.



# Cas Multi-tâches avec OS, Mémoire Virtuelle

- L'adresse en mémoire physique est déterminée dynamiquement par l'OS
- Les programmes sont chargés de manière continue (code, données possiblement disjoints)
- Il y a isolation
- Mais chaque programme tourne dans un espace d'adressage virtuel.
- Résout le problème de fragmentation précédent, par swap d'un processus entier!
- Ex: Unix V7



# Support matériel pour la mémoire virtuelle (simple)

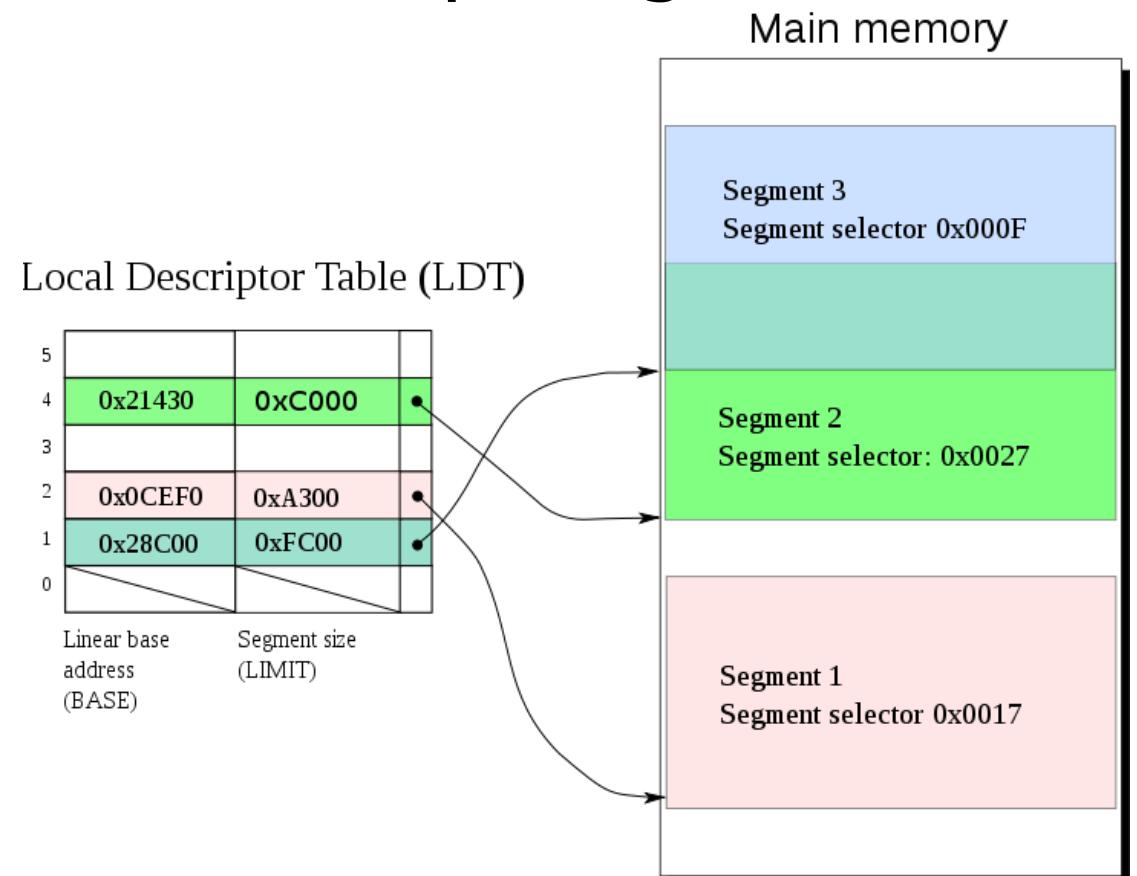
Mécanisme de conversion à la volée des adresses virtuelles utilisées par le processus en adresses physiques

dans notre cas simple

adresse de base pour la région de code, pile, données...

# Les segments Intel

Ce concept est implanté dans la famille des processeurs Intel x86 en mode protégé



# Pagination

Segmentation poussée à l'extrême...

découpage de l'espace virtuel en pages

page : bloc de mémoire, souvent 4Ko

traduction de l'adresse de début de page et utilisation d'un offset

1 traduction virtuelle / physique pour 4k adresses

heureusement la plupart des adresses sont invalides...

pas besoin de continuité physique (sauf cas rares)

besoin de ne charger que les pages utiles

# Gestion mémoire

MPU : Memory Protection Unit

en charge du contrôle des accès à la mémoire

MMU : Memory Management Unit

en charge de la gestion de l'espace d'adressage

# Memory Protection Unit

Découpage physique en régions

Pas de translation d'adresse (ou très élémentaire)

Restreint l'accès aux régions physiques par le processus courant

# Memory Protection Unit

Définit des régions matérielles

début, taille

droits : lecture, écriture, exécution, cachable...

Fonctionne en accord avec le mode de fonctionnement du processeur (rings...)

accès invalides...exception!

# Région MPU

Une région est associée à une partie de l'espace d'un processus  
instructions (exécutable, read-only)  
données (non exécutable, read/write)

Granularité

Région de longueur  $2^n$ , 4Ko, 8Ko

dans certains systèmes embarqués cela peut être tout petit  
(quelques octets)

Adresse

multiple de sa taille

# MPU 8 régions

~~Données  
Bibliothèque 1~~

~~Données  
Programme 1~~

~~Code  
Programme 1~~

Données  
Bibliothèque 2

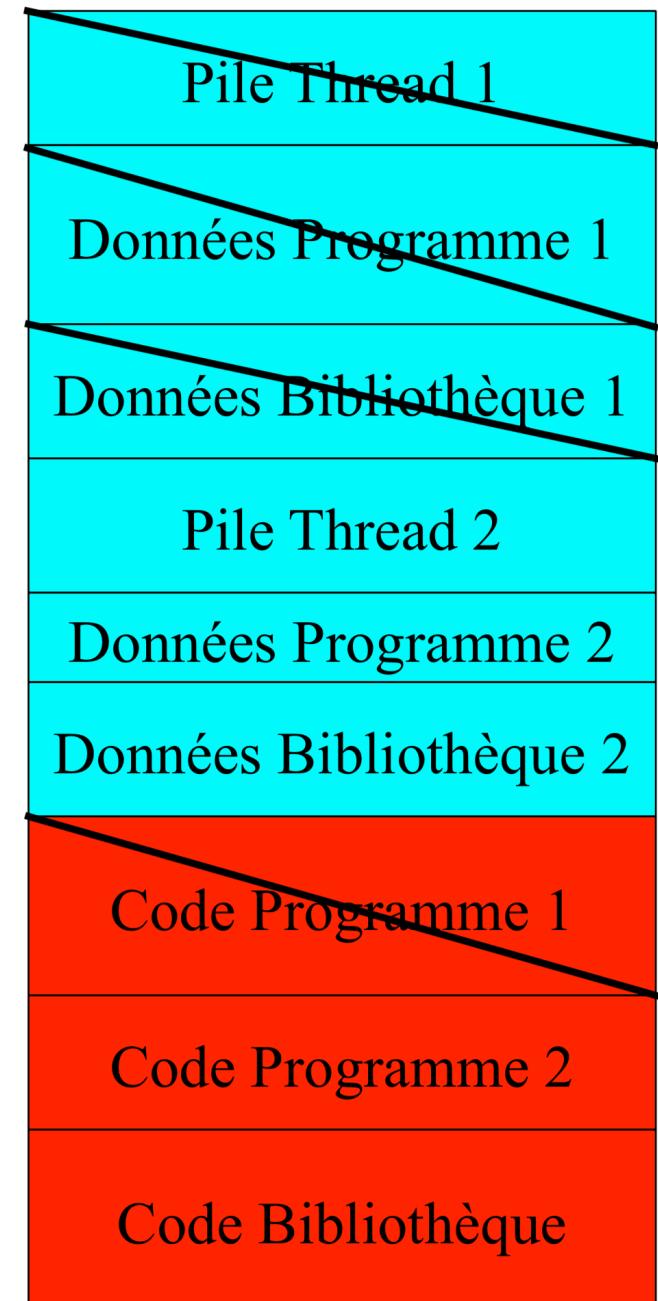
Données  
Programme 2

Code  
Programme 2

Pile Thread  
Courante

Code  
Bibliothèque

- R/W
  - R/X
  - 
  -
- Région invalide

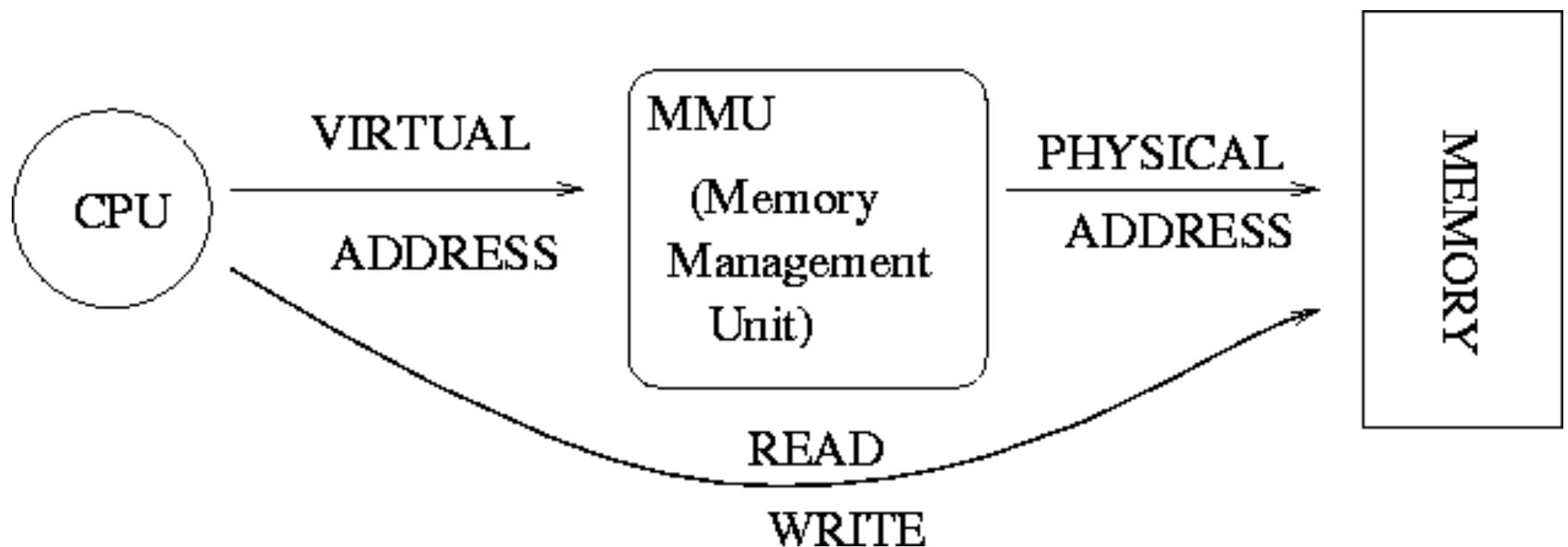


# Memory Management Unit

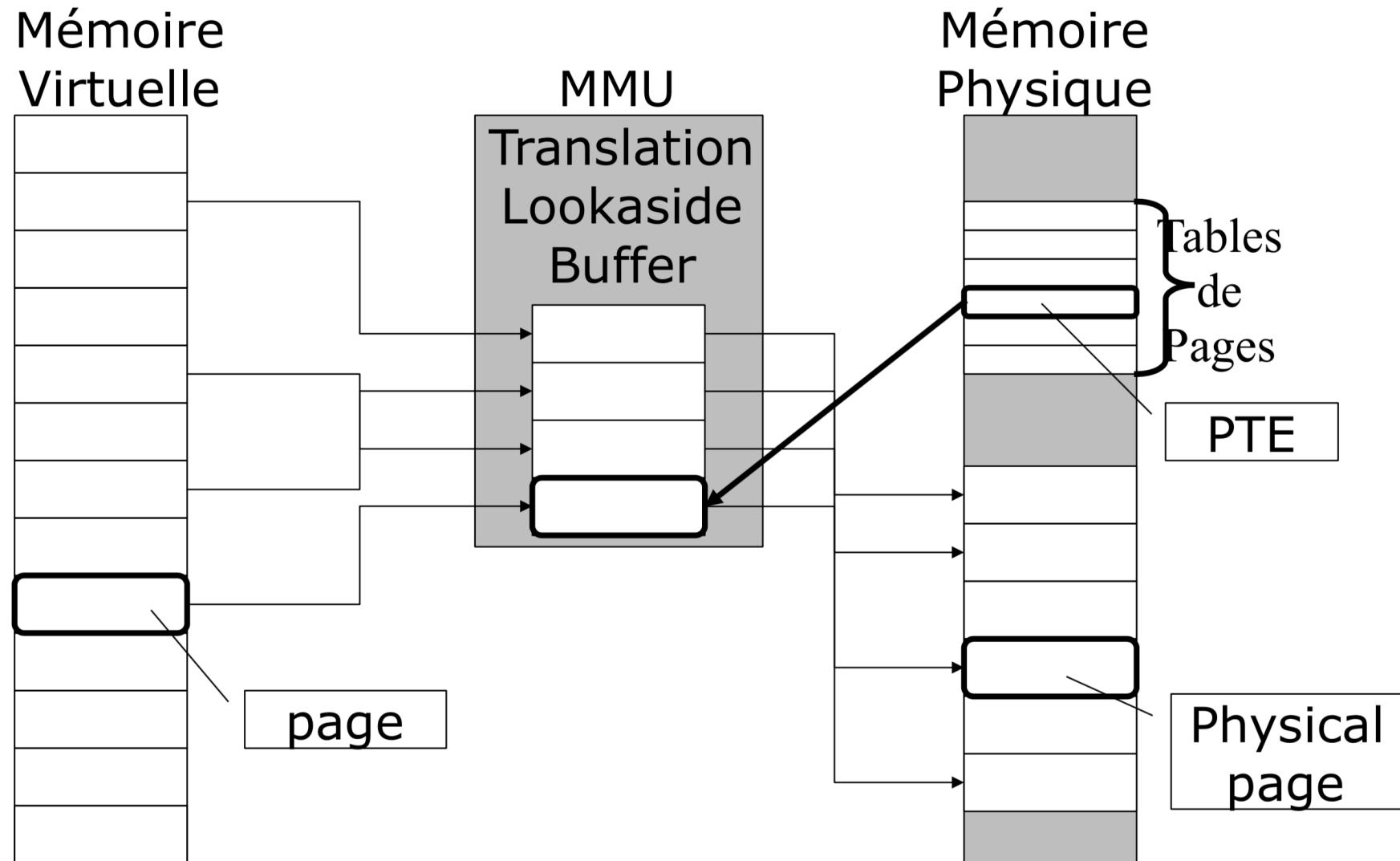
- Notion d'espace d'adressage virtuel
  - Permet de « mapper » des fichiers dans l'espace d'adressage d'un processus
  - Contient souvent grandes zones non définies
- Décomposition de la mémoire en pages (en général de même taille 4KB ou 8KB)
- Gestion dynamique fine
- Conversion adresse virtuelle => adresse physique
  - utilisation structure "arborescente" multi niveaux
  - Intel : 2 niveaux (1 Directory + Page Tables)

# MMU

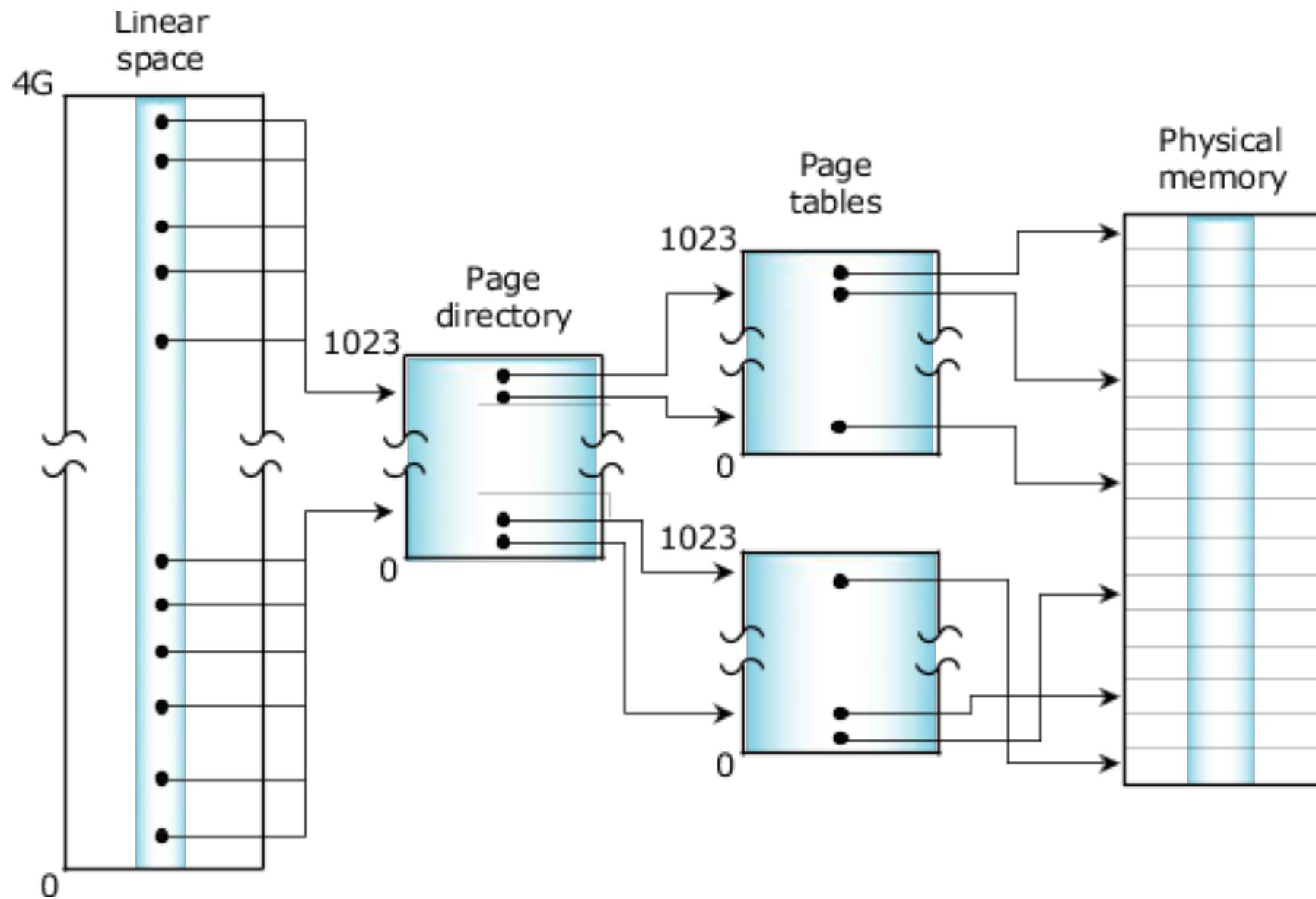
$$\Phi_{\text{address}} = \text{MMU}(\text{virtual}_{\text{address}})$$



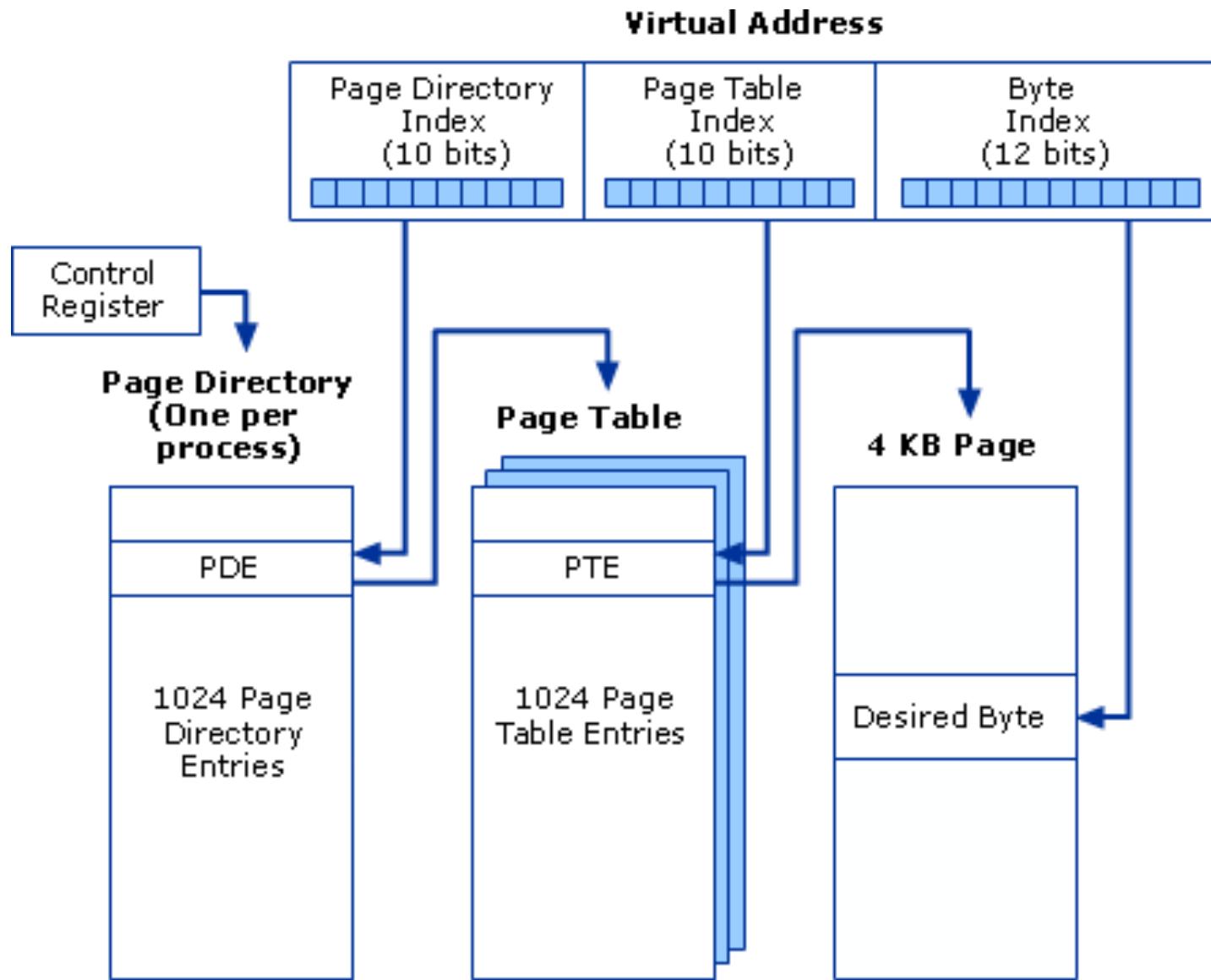
# Memory Management Unit (MMU)



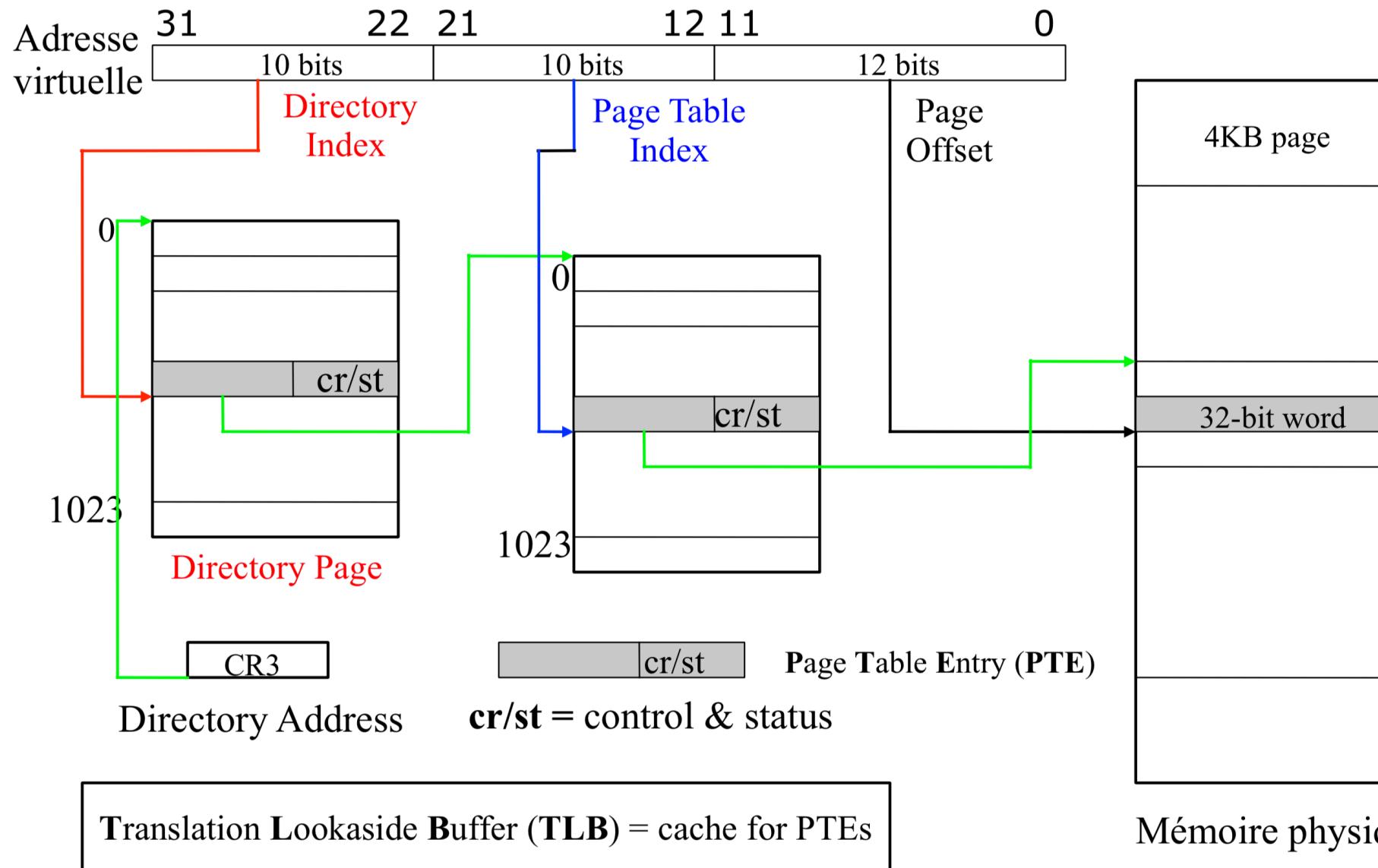
# MMU (Playbook OS, BlackBerry)



# MMU (Intel)



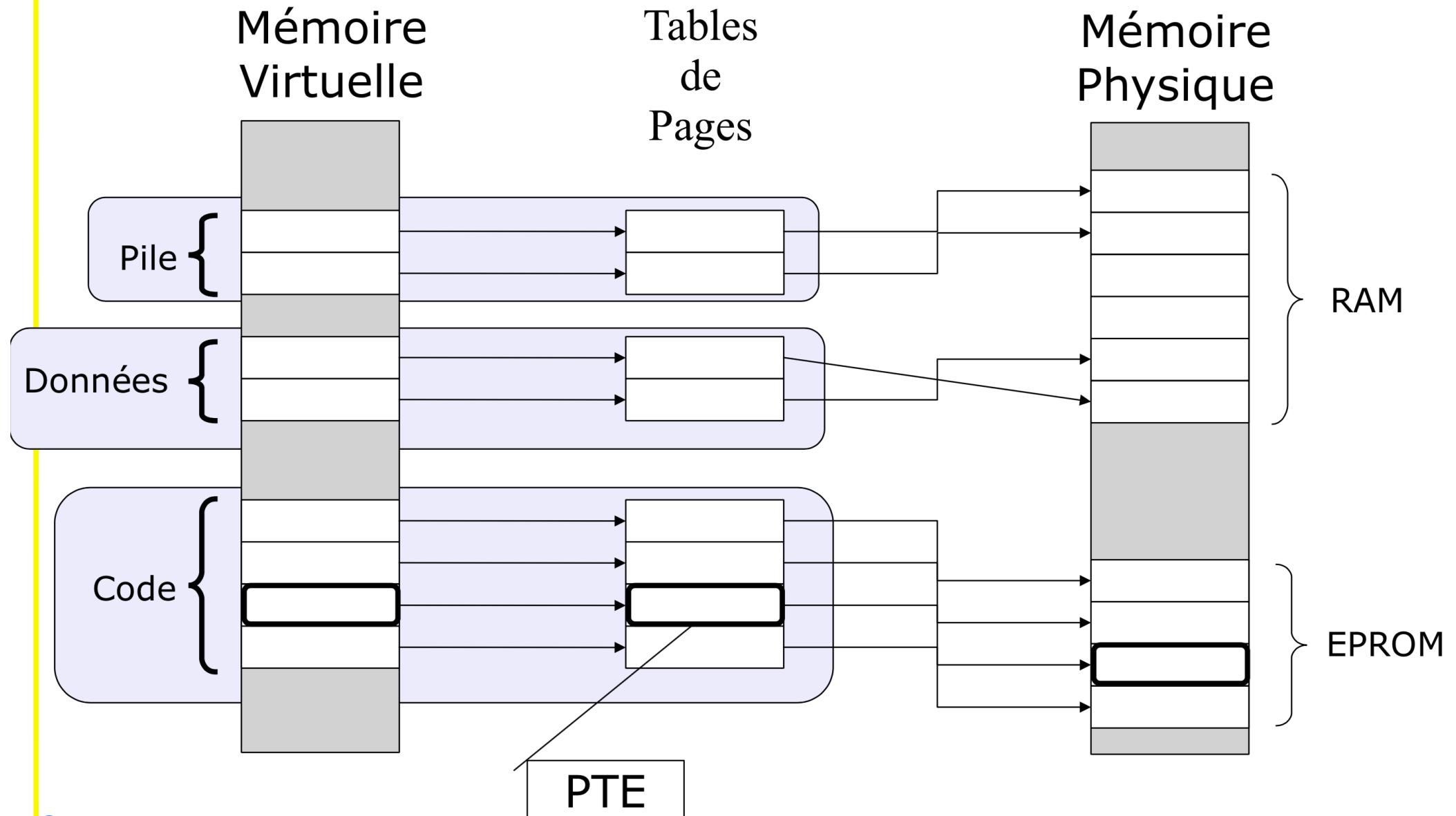
# MMU Intel



# Page Table Entry

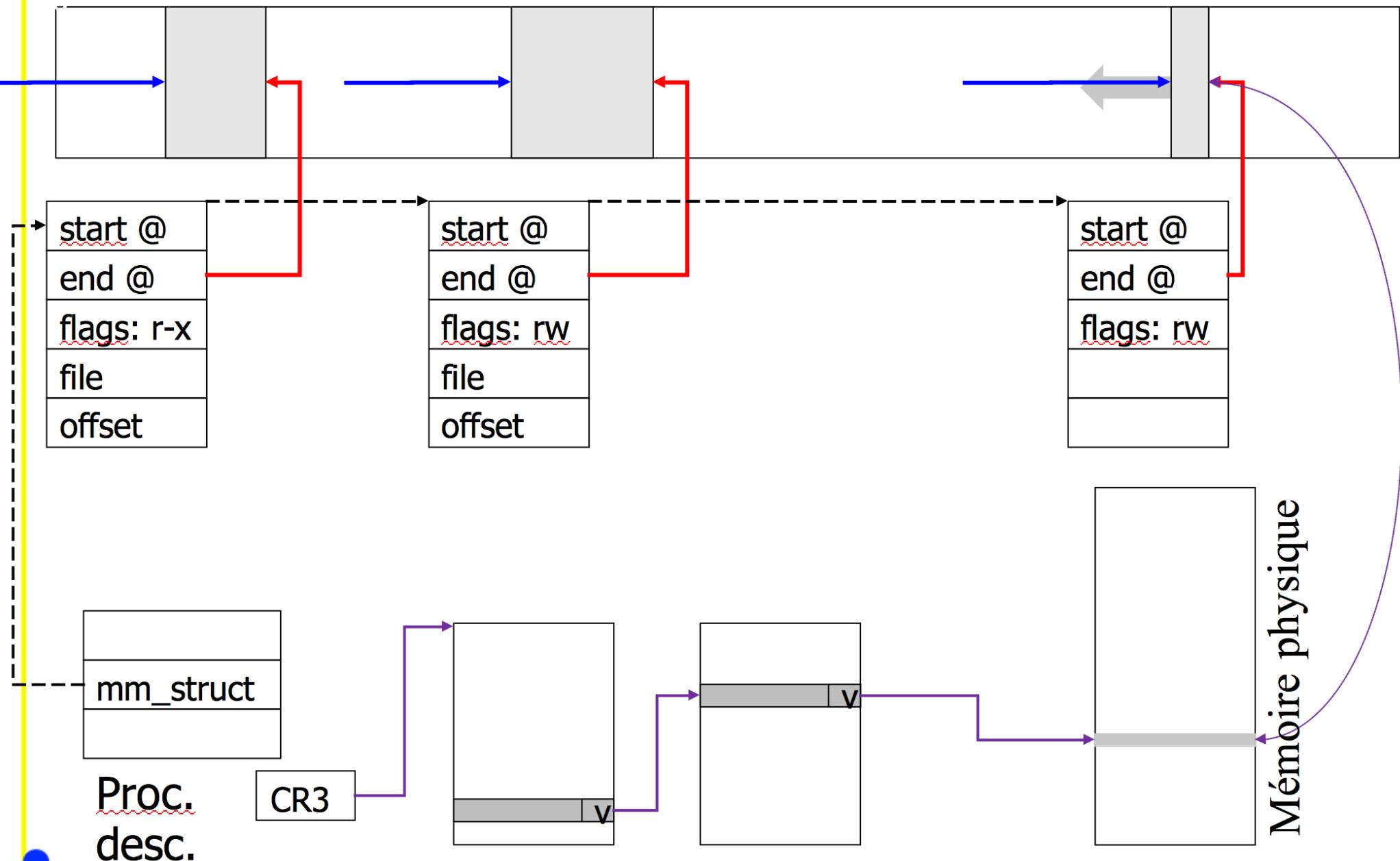
- Déscrit une page d'un espace virtuel
- Adresse de la page physique associée, si valide
- Bits de contrôle
- Page physique associée valide
- Droits d'accès (lecture, écriture, exécution)
- Bits de status
- Page accédée
- Page modifiée

# Exemple de "Mapping"



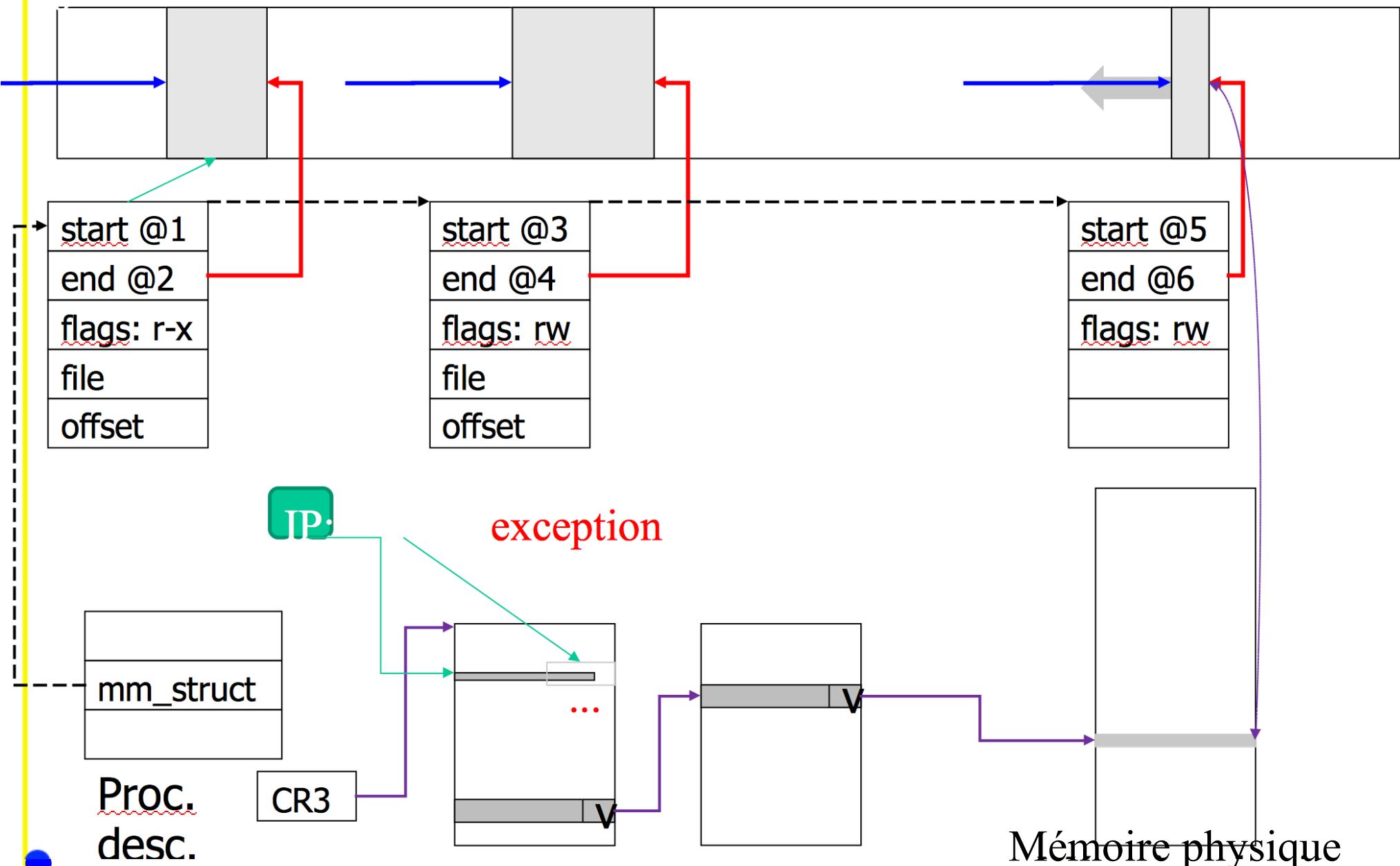
# Gestion Mémoire

Espace adressage du processus après exec



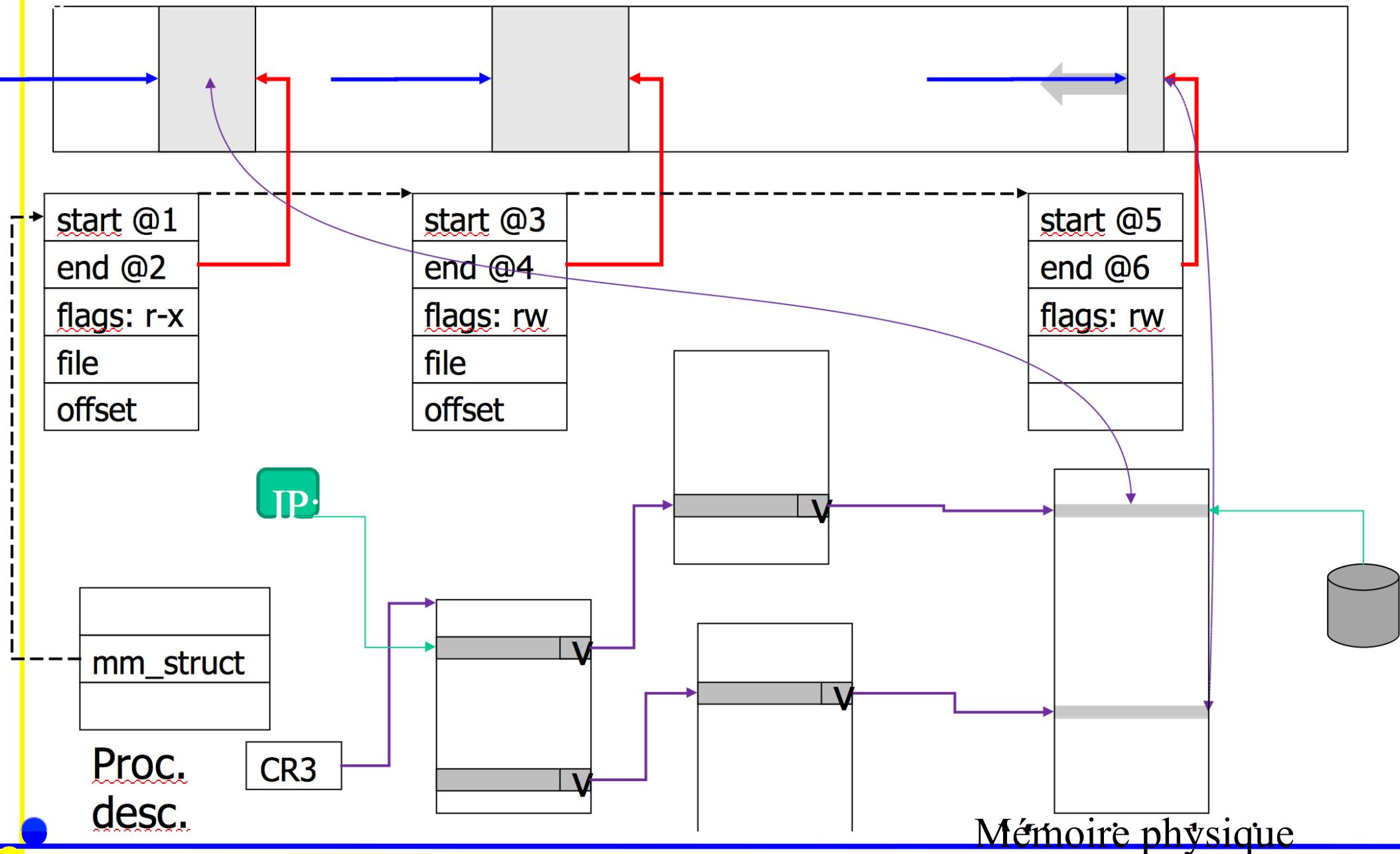
# Gestion Mémoire

Accès 1ere instruction!



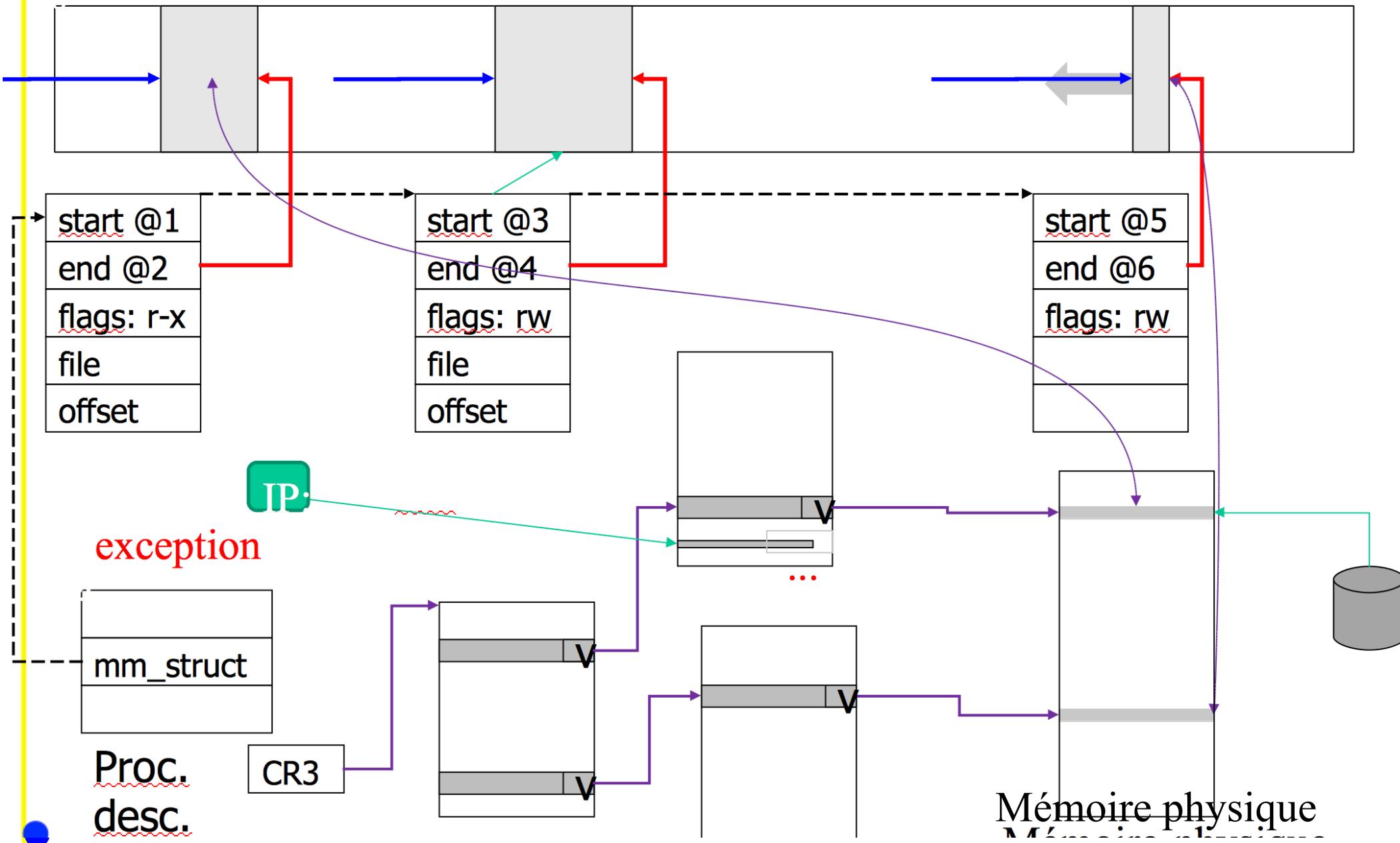
# Gestion Mémoire

Accès 1ere instruction! Chargement 1ere page de code!



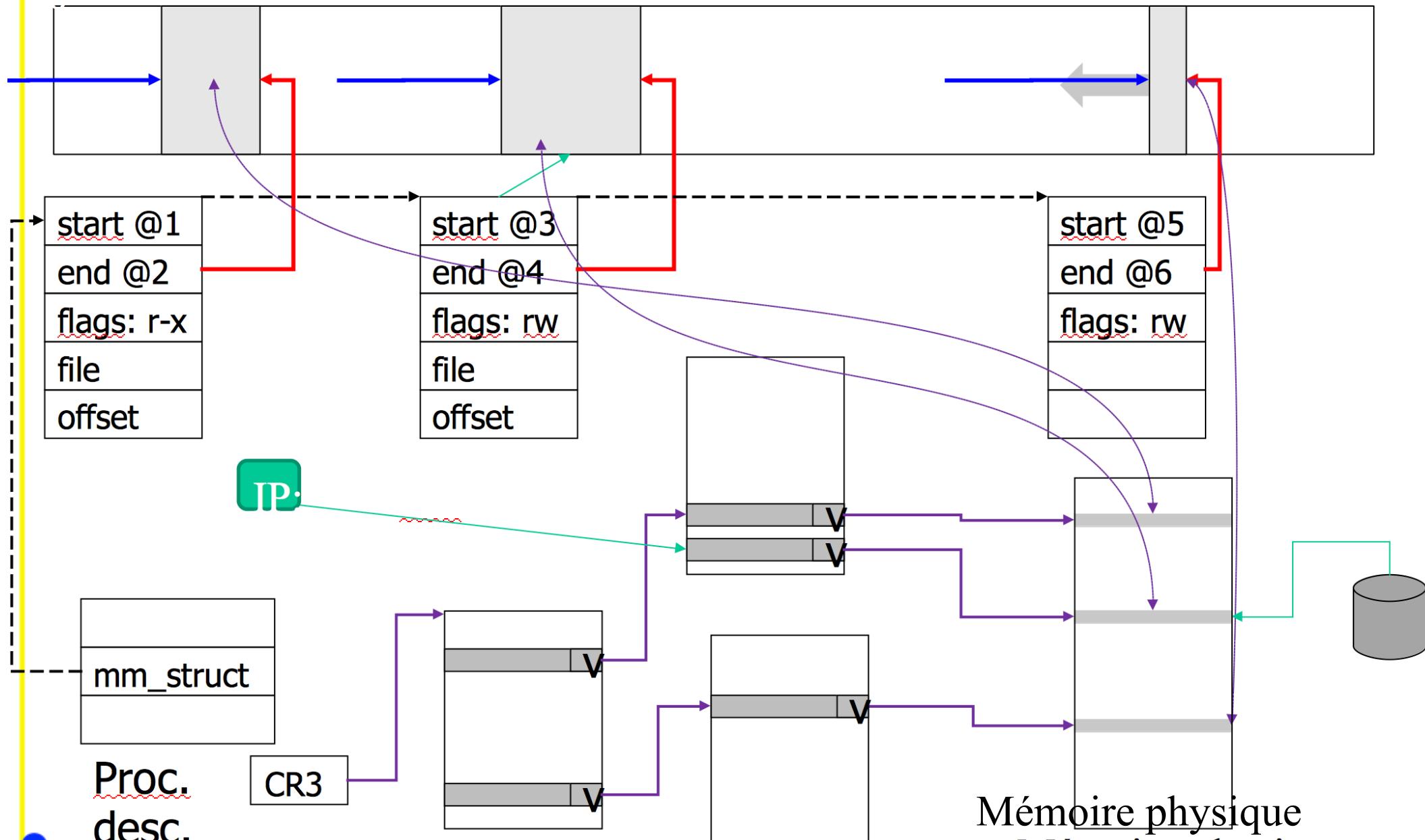
# Gestion Mémoire

## Exécution 1ère instruction!



# Gestion Mémoire

Exécution 1ere instruction! Chargement 1ere page de données



# Caches Mémoire

But : accélérer les accès mémoire

Basé sur :

- le principe de **localité temporelle**

la probabilité d'utiliser au temps  $t+\Delta t$  la donnée utilisée au temps  $t$  est d'autant plus élevée que  $\Delta t$  est petit

- le principe de **localité spatiale**

la probabilité d'utiliser au temps  $t+1$  d'une donnée voisine de celle utilisée au temps  $t$  est d'autant plus élevée que la distance entre les deux est faible

# Caches Mémoire

Attention, il s'agit de principes, ce ne sont donc pas des lois universelles...

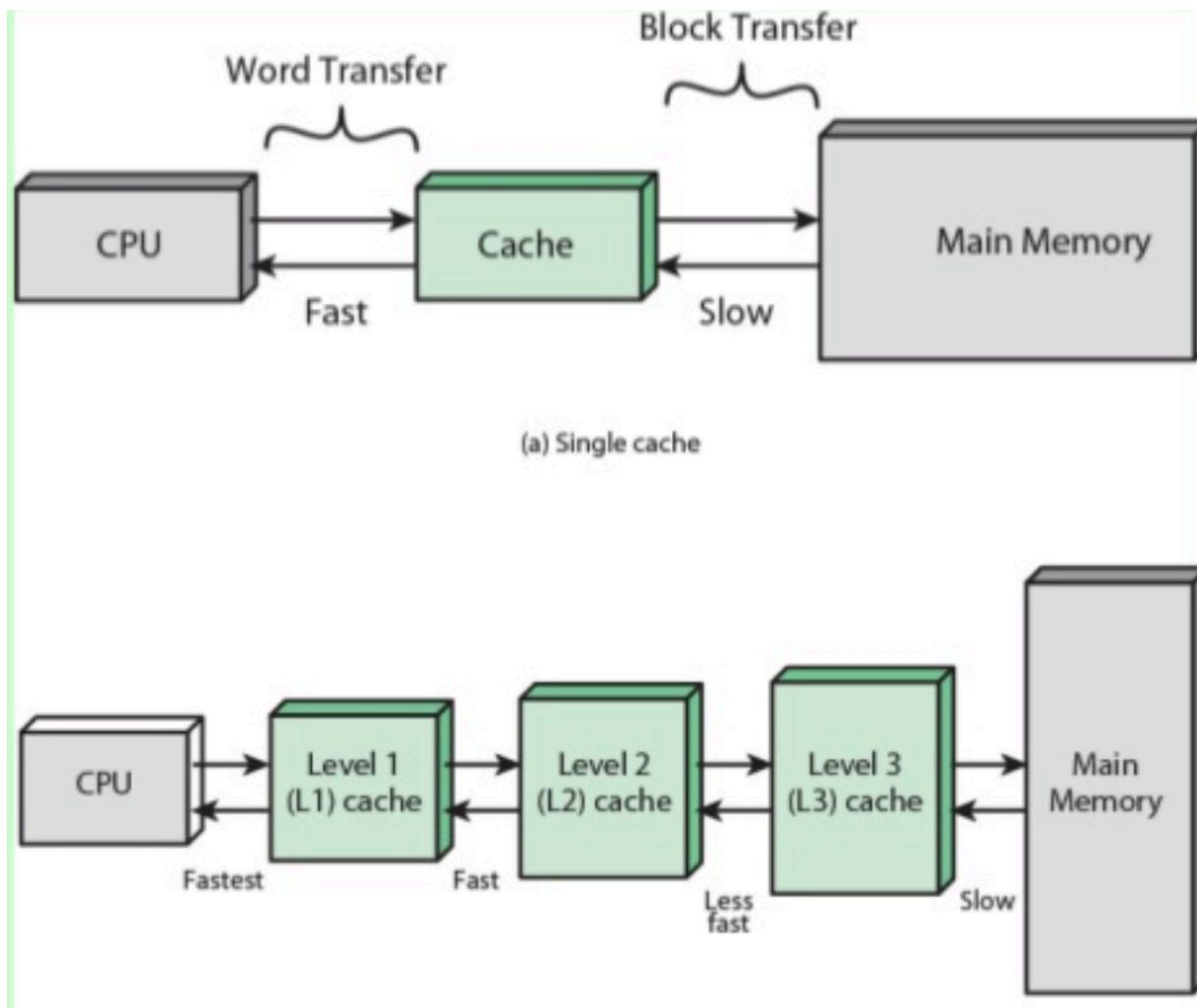
Cela dépend fortement de la façon d'utiliser les données

En général, ces principes sont respectés « naturellement » ou « instinctivement »

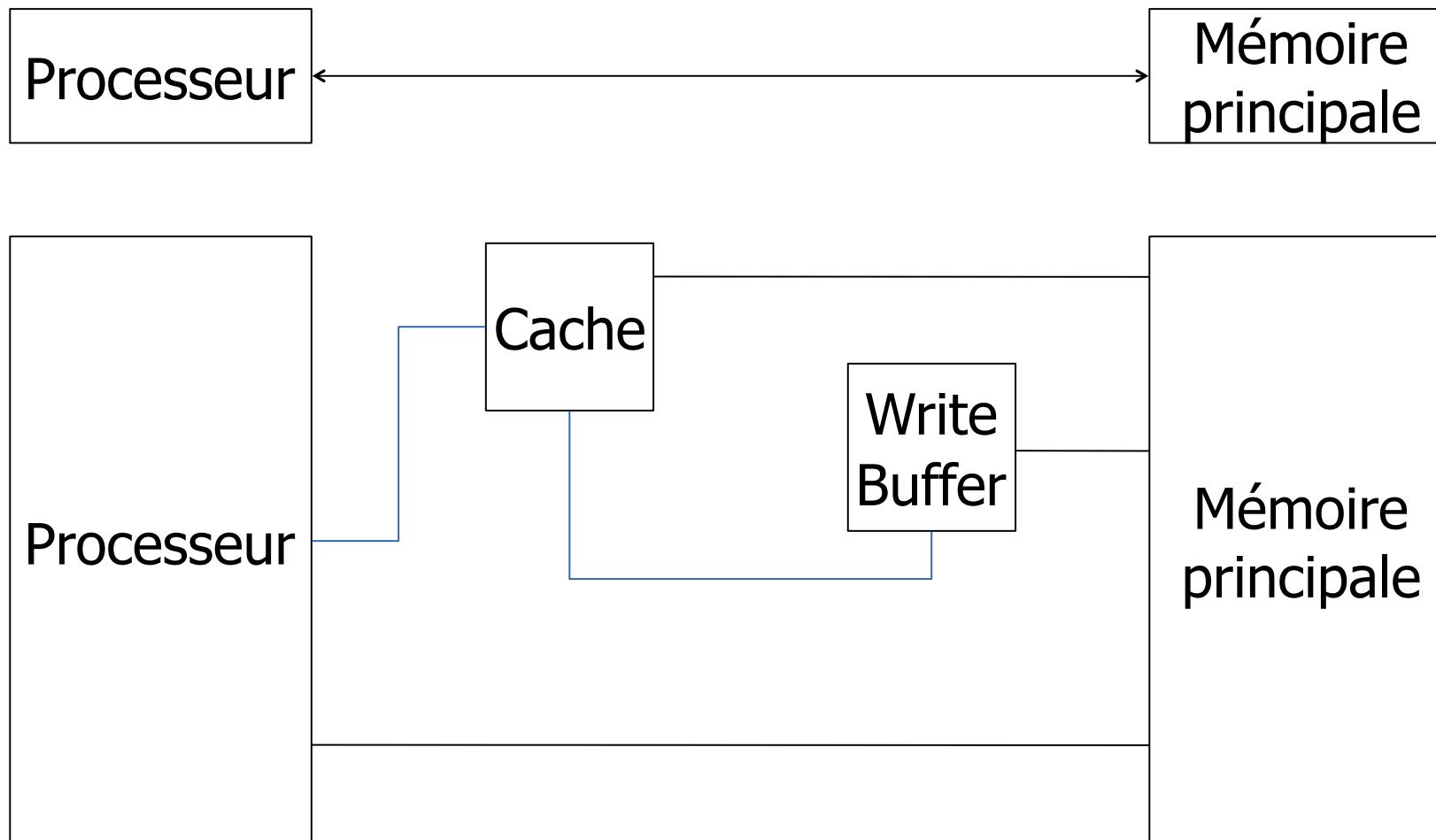
mais rien d'obligatoire

on peut violer ces principes...

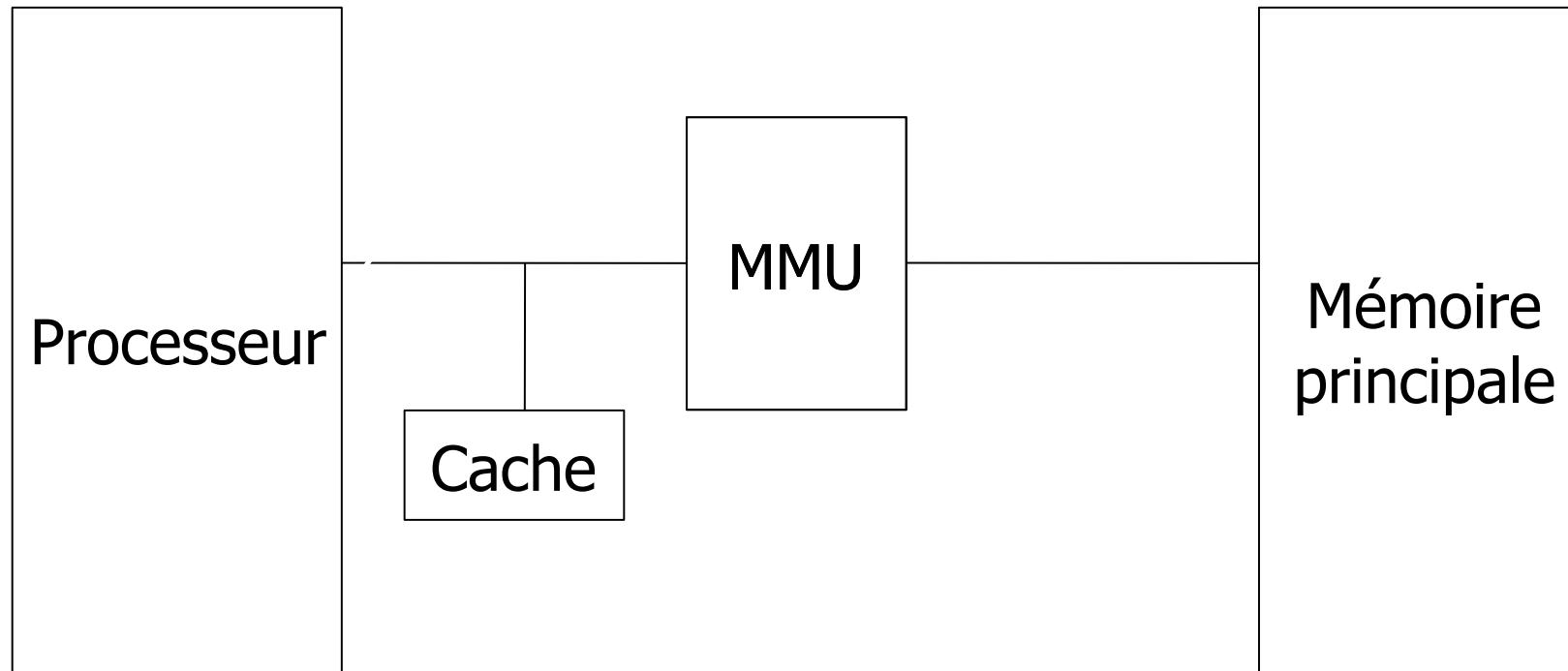
# Caches Mémoire



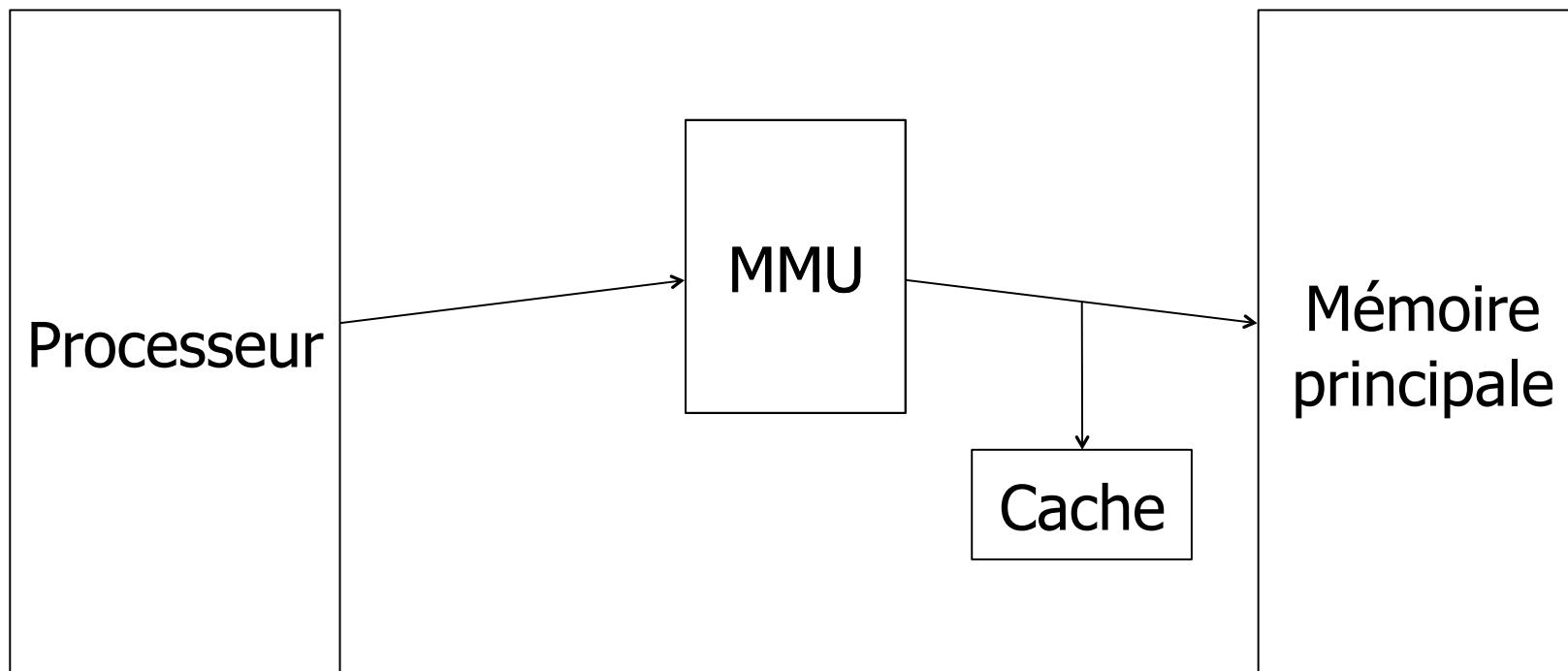
# Caches Mémoire



# Cache Logique



# Cache Physique



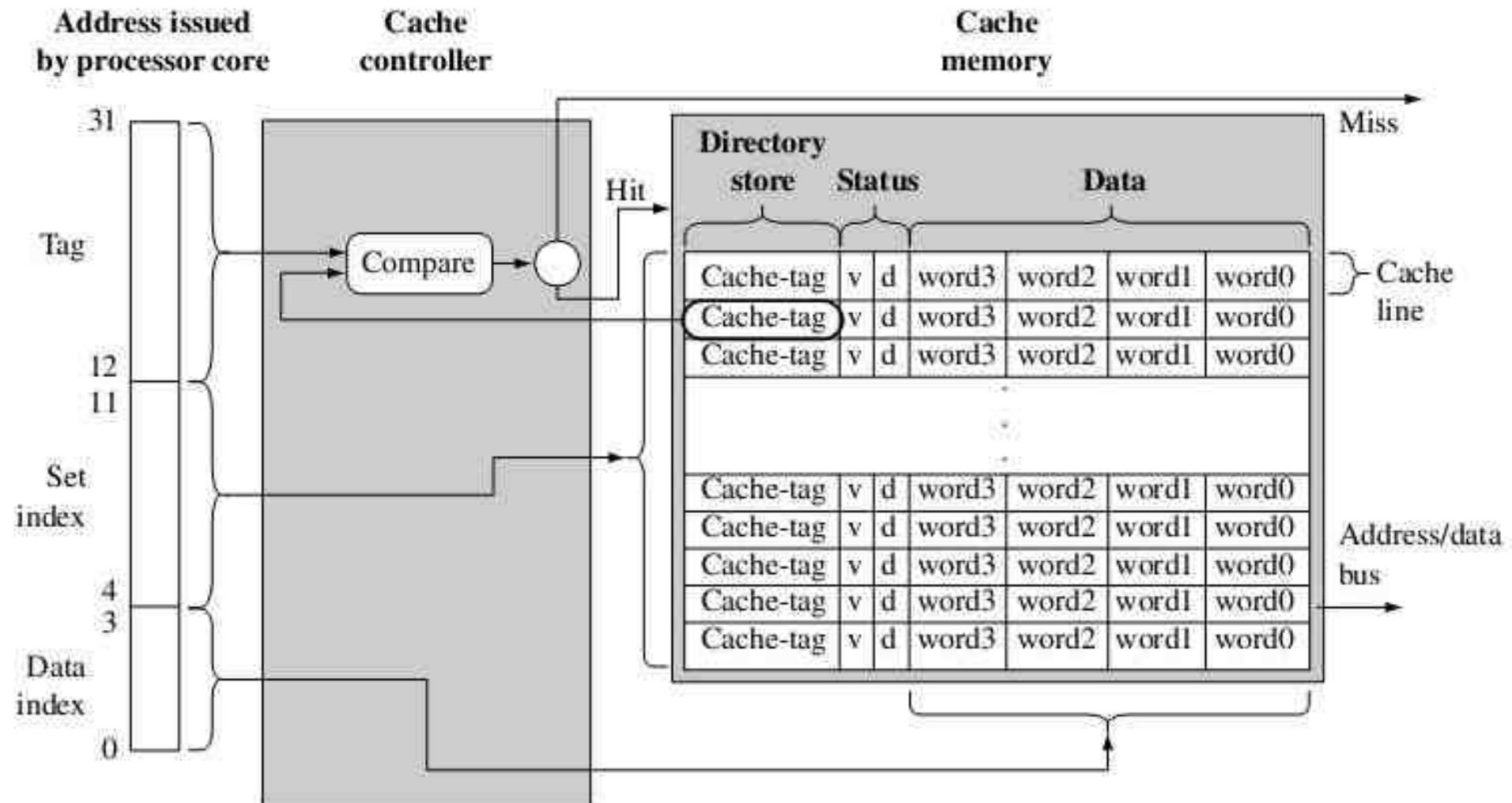
# Composition d'un Cache

- Cache est décomposé en « lignes de cache »
- Ligne chargée avec des "blocs"
  - Zones contiguës de mots mémoire
  - Lignes identifiées à partir de l'adresse
- États
  - Ligne valide ou non
  - Ligne modifiée ou non

# Caches - Politique de gestion

- Writethrough
  - Toute modification est immédiatement répercutée en mémoire centrale => lent
- Writeback (utilisation du dirty bit)
  - Écrit dans une ligne de cache valide
  - Ligne de cache contient des données plus récentes
  - Ligne écrite quand nécessité de remplacer la ligne
- Politiques de remplacement
  - Round-robin ou pseudo aléatoire

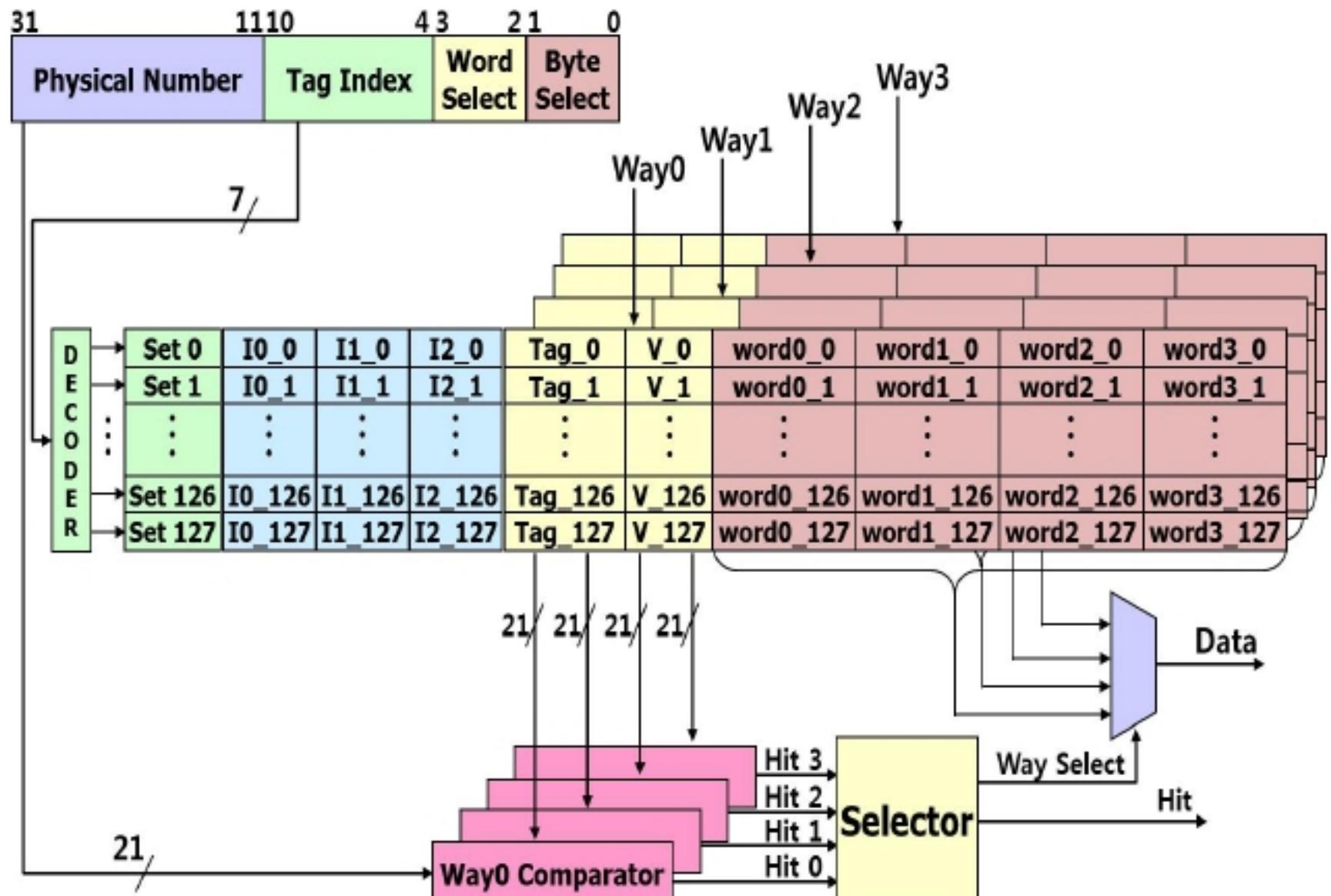
# Architecture d'un cache



V = valid

D = dirty (modifié)

# Caches Associatifs



# Caches - contrôle

- Flush: vider tout le contenu
- Clean: écrire toutes les données modifiées en mémoire
- Possibilité d'effectuer ces opérations par portion (granularité: la ligne)
- Opérations pour Instruction caches et Data caches
- Possibilité de verrouiller du code et des données en cache

# Quid si plusieurs processeurs

- Avec chacun leur cache !!!
- Problèmes de cohérence en vue !
  - Lecture seule... OK
  - Écriture..
    - Il faut s'assurer que la donnée modifiée n'est pas (plus) présente dans les autres caches...
    - Cache snooping implémenté par matériel (coopération entre les caches). Protocole de cohérence de cache

# Intel PAE

- Machines 32 bits
  - Normalement limitées à  $2^{32}$  octets (4 Giga)
- Intel PAE: Physical Address Extension
  - Permet  $2^{36}$  (64 Giga) octets de mémoire physique!
  - Adresses physiques => 36 bits!
  - Adresses virtuelles toujours 32 bits!
- Une application est toujours limitée à 4 Giga
- Mais la capacité physique est accrue, moins de swap.

# Intel PAE

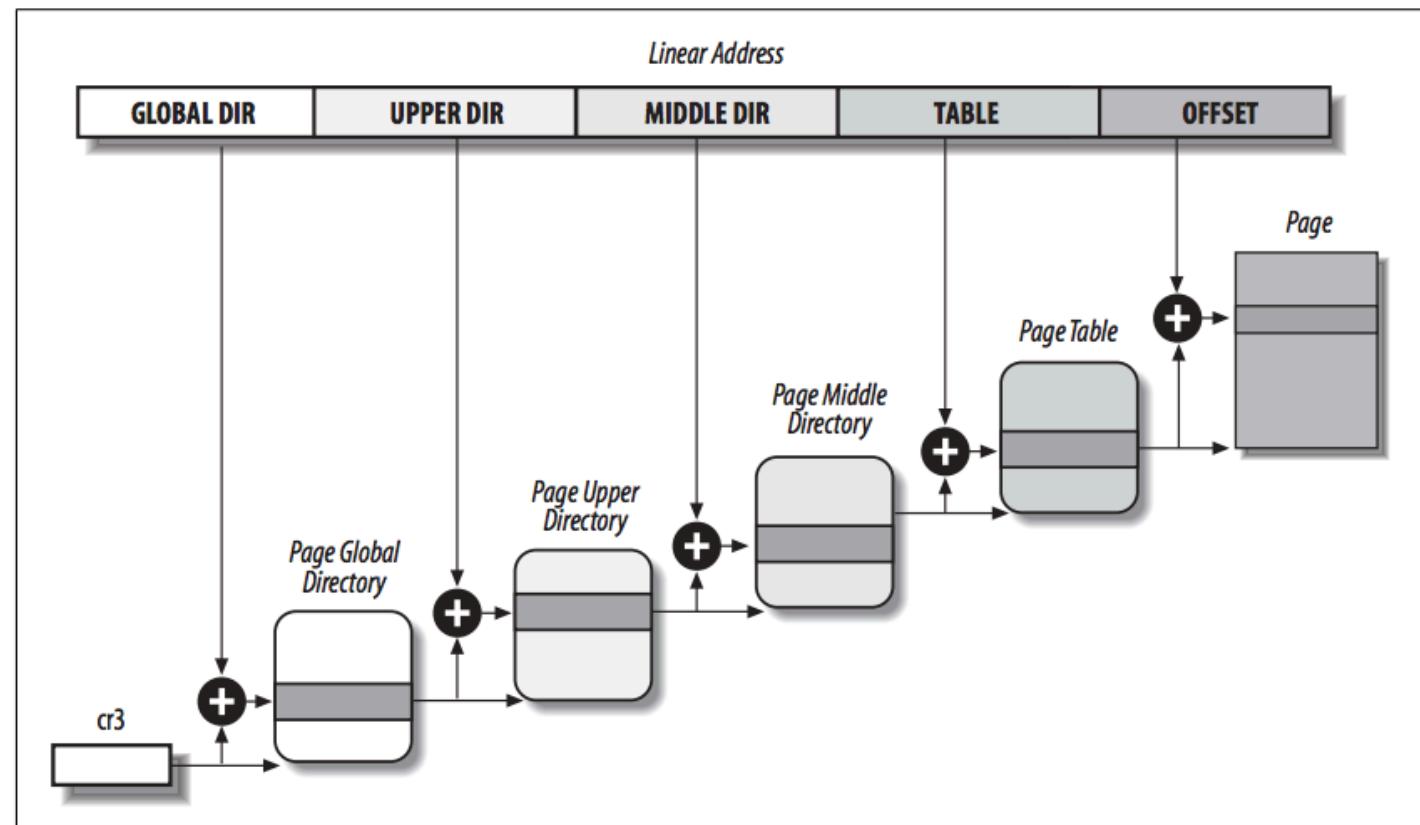
- Conversion adresse 32 bits en 36 bits:
  - Modification des PTE
    - Offset dans une page de 4K: 12 bits
    - Adresse d'une page:  $36 - 12 \Rightarrow 24$  bits utiles
    - Les PTE prennent maintenant 64 bits (en PAE) au lieu de 32 bits.
  - Moitié moins de PTE dans les tables de niveau 1 et 2
  - Espace d'adressage virtuel réduit à 1/4!
- CR3 pointe alors sur 4 tables de niveau 1 au lieu de 1
- Activation par bit 5 (PAE) de CR4

# Machines 64 bits

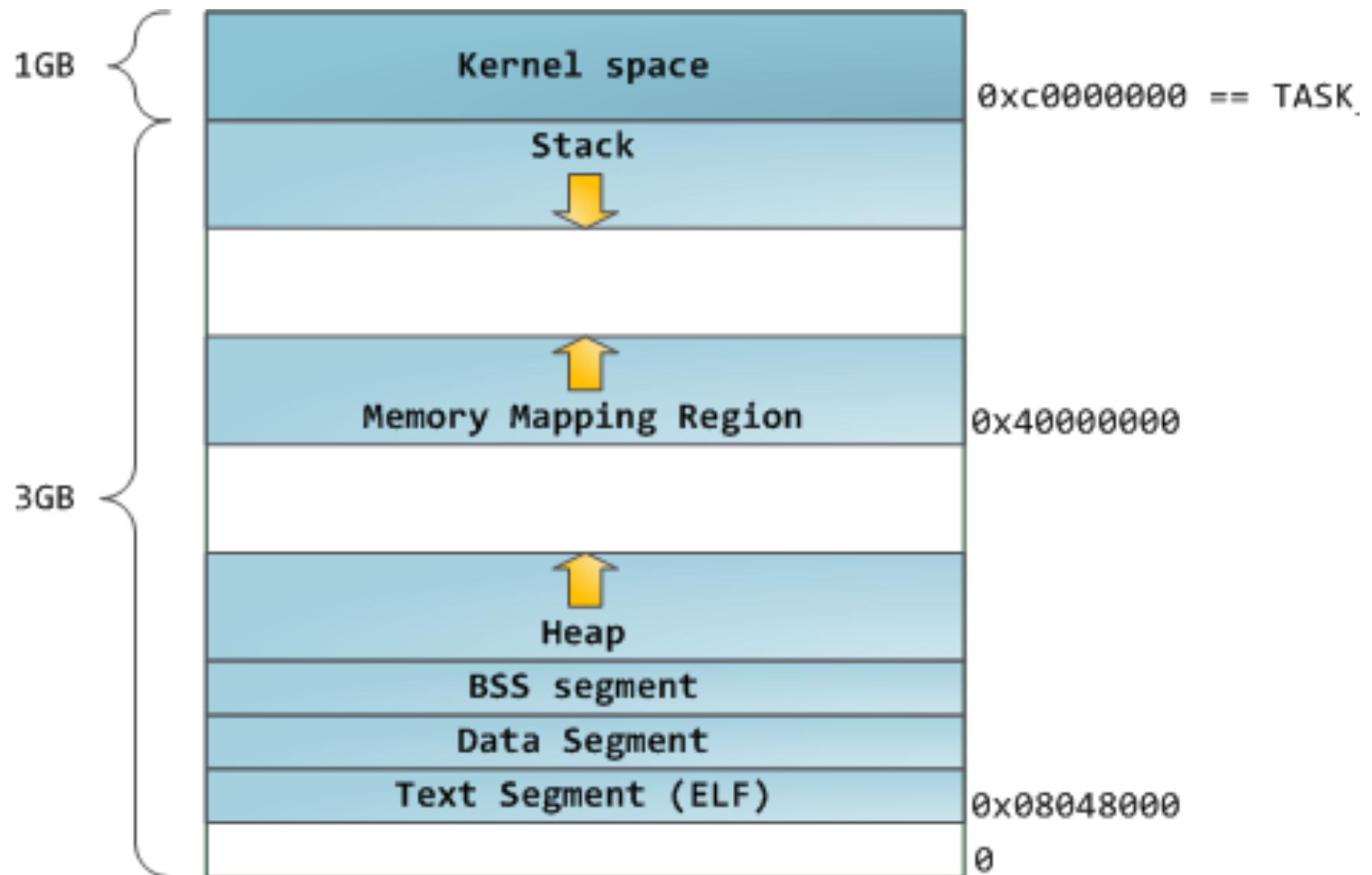
- N'utilisent pas (pas encore?) la totalité de l'espace d'adressage possible
  - 1TB => adresses physiques sur 40 bits
  - 256 TB => adresses physiques sur 48 bits
- Nécessité d'augmenter le nombre tables d'indirection

# Linux

- Linux prévoit jusqu'à 4 niveaux d'indirections
- Sur x86/32 bits les niveaux Upper/Middle sont « inexistant »



# Espace d'adressage Linux 32-bits



# Mémoire Virtuelle en pratique

- Nouveau programme « chargé » pour exécution :
  - Configuration de la MMU, mais pas de page chargée en mémoire physique
- Quand le processus peut s'exécuter
  - Instruction Pointer
    - Le processeur cherche à lire l'instruction correspondante pour l'exécuter
    - Ouille ! Il n'y a pas de page encore en mémoire
    - Donc la MMU va générer une exception..

# « Défaut de page »

- Le SE va « traiter » l'exception
  - Examen de l'adresse (virtuelle) ayant causé l'exception
  - Si adresse invalide => on « tue » le processus
  - Si adresse valide,
    - Trouver quelle partie (page) du fichier (programme) contient l'instruction correspondante...
    - Trouver une page de mémoire physique « libre »
    - Copier la page sur disque dans la page mémoire
    - Mettre à jour les tables de pages appropriées (et/ou TLB)
    - Ré-exécuter l'instruction ayant causé l'exception

# Défaut de page, suite

- L'instruction va probablement référencer une donnée du programme
  - Scénario identique à ci-dessus
  - Protection différente associée à la page :
    - Lecture / écriture vs lecture / exécution
- And so on...
- On va donc « remplir » petit à petit la mémoire physique avec des pages :
  - Provenant de programmes binaires, librairies
  - De pile et allouées dynamiquement (tas)

# Gestion utilisation mémoire physique

- On peut « reprendre » une page attribuée à un processus :
  - En cas de nouvel accès, le mécanisme de « récupération » des défauts de page, ramènera la page en mémoire
  - Mais, risque de « thrashing » :
    - Recycler une page physique (page X du fichier A) en l'attribuant à la page Y du fichier B
    - Résoudre un défaut de page pour ramener X de A
    - Et recommencer => Inefficace
  - Ne pas toucher au « Working Set »

# Gestion utilisation mémoire physique

- Réutiliser des pages inutilisées et à défaut, pas accédées récemment
- Pages de processus terminés
- Ou LRU : « Least Recently Used »
  - Problème le matériel n'enregistre pas d'estampille temporelle associée au PTE...
  - Heuristique « Not Recently Used » en s'appuyant sur l'indicateur « accédée »
- Anticiper plutôt qu'agir dans l'urgence d'une demande
- Maintenir (autant que possible) un ensemble de pages libres, par exemple quand il n'y a pas d'applications en cours d'exécution

# Cas « extrême »

- Il peut être nécessaire de recycler des pages physiques contenant des modifications...
  - Ne pas perdre les modifications !
  - 2 solutions
- La page est issue d'un fichier : (fichier mappé)
  - Écrire le contenu de la page dans le fichier
- C'est une page « anonyme » (pile, tas, données)
  - Il n'y a pas (pas encore) de stockage associé
  - Utiliser une zone de « Swap »

# Autres contraintes

- Ne pas éjecter les pages partagées
  - Il y a plus de chance qu'elles appartiennent à un WS
- Ne pas éjecter les pages sur lesquelles une Entrée-Sortie est en cours...
- Ne pas éjecter les pages verrouillées en mémoire par les processus

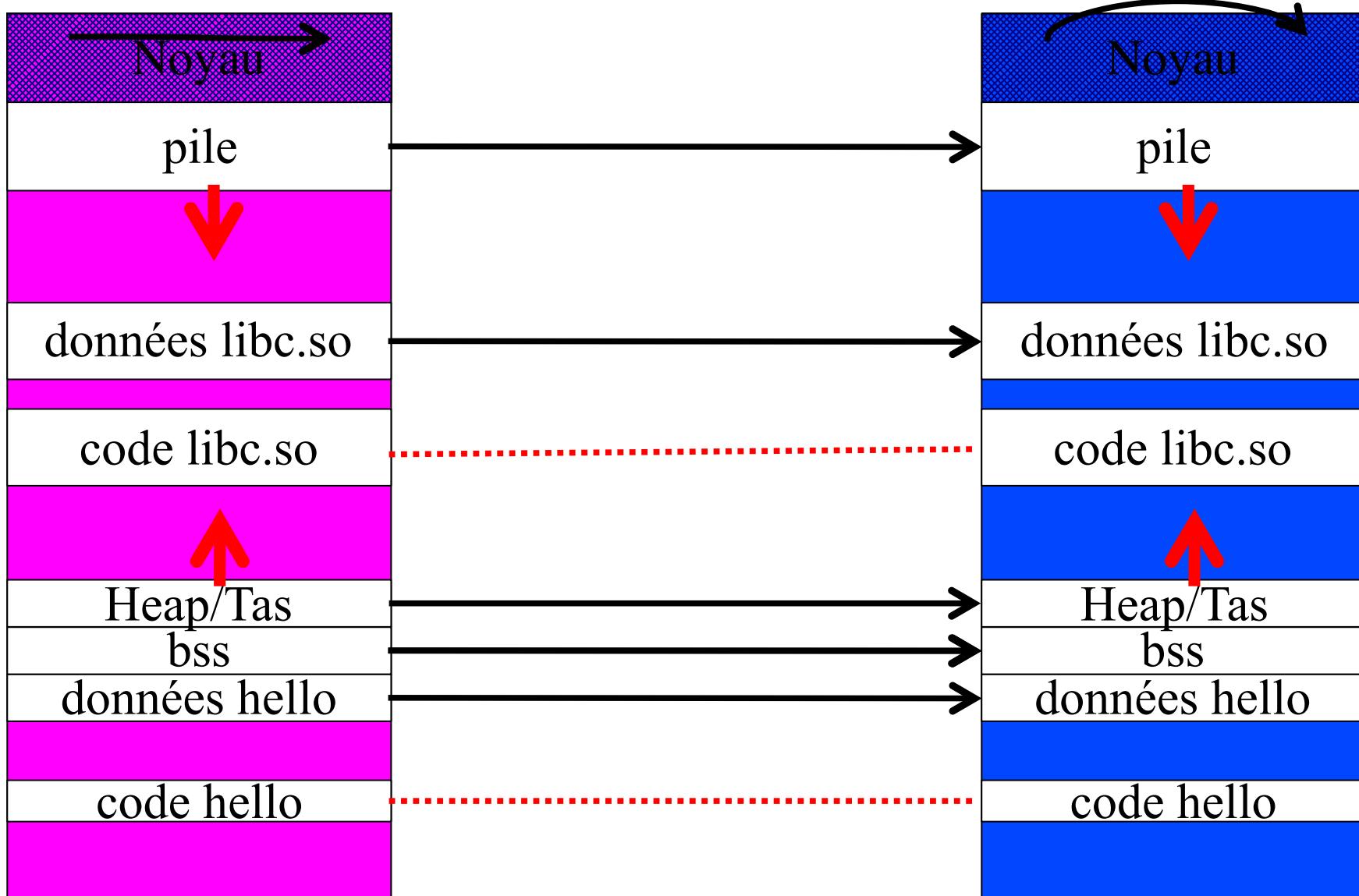
# Création d'un processus

Unix / Linux :

- Par duplication d'un processus existant
- `fork() // 1 appel, 2 retours!`

```
if ((pid = fork()) == 0) {  
    // le nouveau processus: fils  
    execlp(file,....); // nouveau prog  
} else {  
    // le processus pré-existant: père  
    waitpid(pid,...);  
}
```

# Fork



# Fork et espace d'adressage

- Le nouveau processus reçoit une copie
  - Mêmes adresses pour code, données, tas, pile...
- Donc....
- Pas possible si pas d'espace d'adressage virtuel...
- Pas possible si la machine ne dispose pas de MMU.
  - Dans ces deux cas, il n'y a pas d'indirections entre les adresses vues par le processus et les adresses en mémoire physique.
- Sur les machines sans MMU, il faut pouvoir créer les processus autrement que par « clonage »
  - Par exemple: posix\_spawn

# Processus « trop lourds »

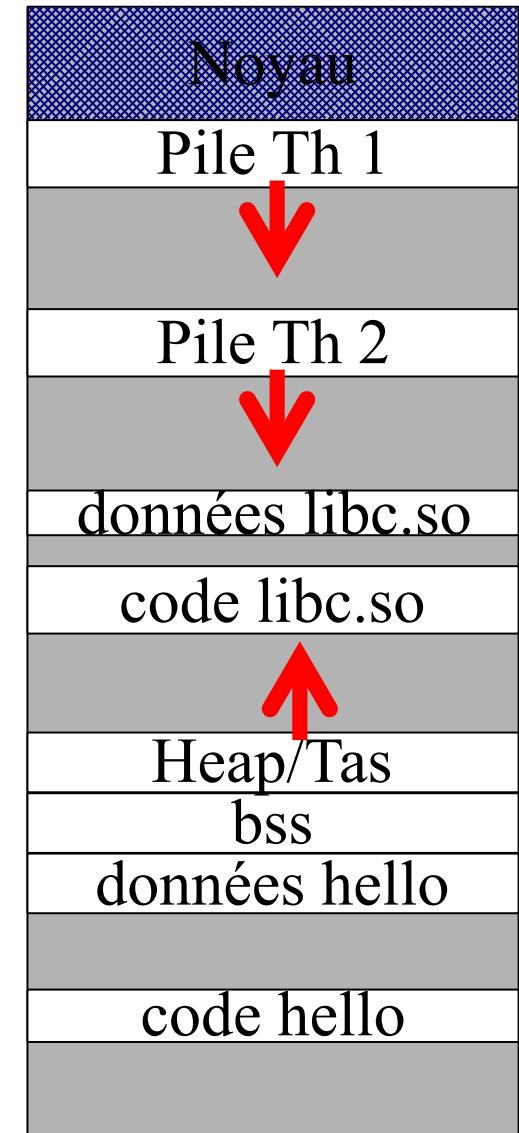
- Il faut d'autres moyens de créer des « activités » permettant de traiter de requêtes :
  - Un processus par requête, trop couteux, trop gourmand en ressources
    - A la création
    - Lors de l'exécution : changement de contexte
  - Trouver un moyen d'avoir plusieurs exécutions en parallèle au sein du même processus.

# Manipuler les threads

- `pthread_create(thout, attrin, fnin, argin)`
  - `pthread_attr_init`: Scheduling policy, parameters...
- `pthread_exit(void *valin)`
  - Attention, `exit()` termine le processus (toutes les threads)
- `pthread_join(thin, void **valout)`
  - Attendre la fin d'une thread en particulier (~ `wait`)
- `pthread_self()`
  - Qui suis-je? Similaire à `getpid()`

# pthread\_create

- Espace adressage partagé
- Les piles ne sont pas protégées les unes des autres
- Handler de signaux peuvent être exécutés par n'importe quelle thread
  - Masquer les signaux dans les threads qui ne veulent pas être perturbées



# Copy On Write

- Quand on crée un processus par fork
  - La plupart du temps, on va complètement modifier son espace d'adressage suite à l'appel exec...
  - Travail inutile et coûteux : duplication espace d'adressage
- Donc les pages qui pourraient être dupliquées sont marquées comme étant en lecture seule
  - Mais le système sait que c'est un mensonge
  - Quand un des 2 processus cherche à écrire dans cette page, cela déclenche une exception (violation de droits)

# Copy on Write (COW)

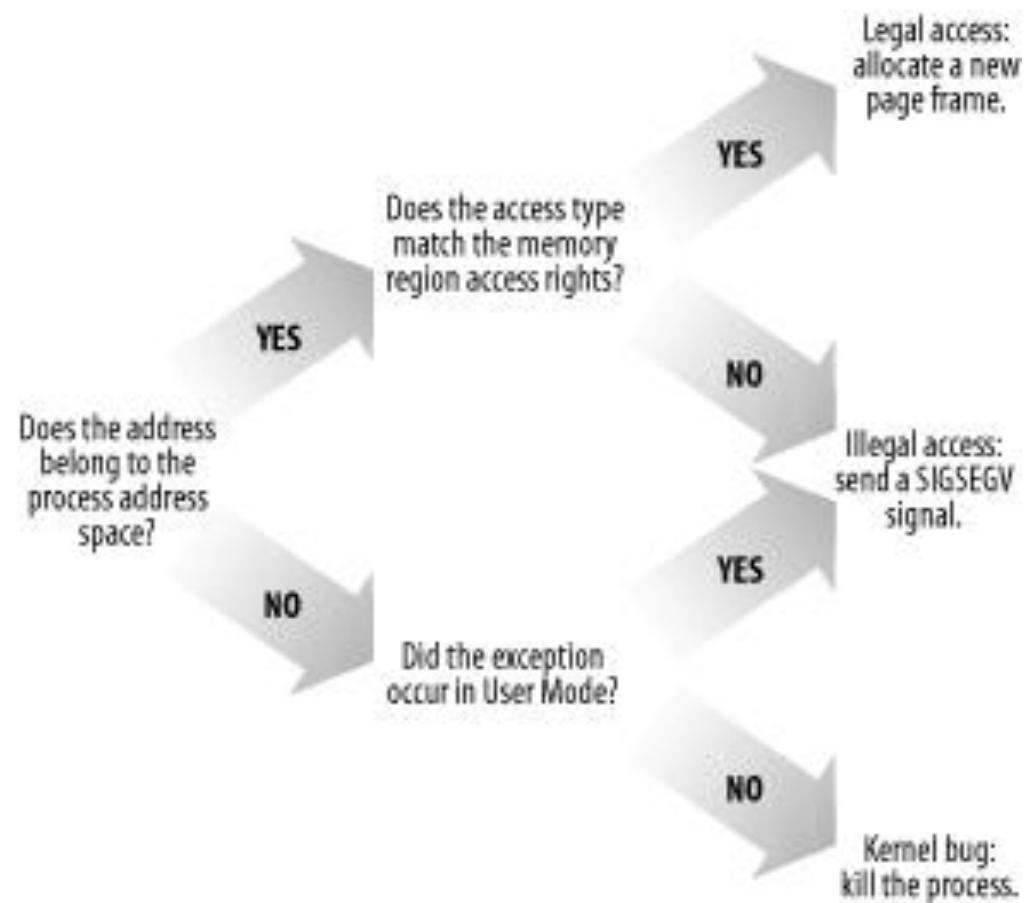
- Le système s'aperçoit de sa supercherie
  - Allocation d'une page physique libre
  - Copie de la page dans la page libre allouée
  - Marque les 2 pages comme pouvant être modifiées
- Permet de ne faire que le travail réellement indispensable
  - Technique classique « lazy »
  - Ou de procrastination !

# Traitement des exceptions

- L'adresse fautive est-elle dans l'espace d'adressage du processus?
  - Oui
    - Est-ce un problème de protection?
      - Oui => Envoyer un signal (SIGSEGV)
      - Non => Accès valide, allouer une page physique
    - Non
      - Accès depuis mode utilisateur?
        - Oui => Envoyer un signal (SIGSEGV)
        - Non => Bug noyau :-(

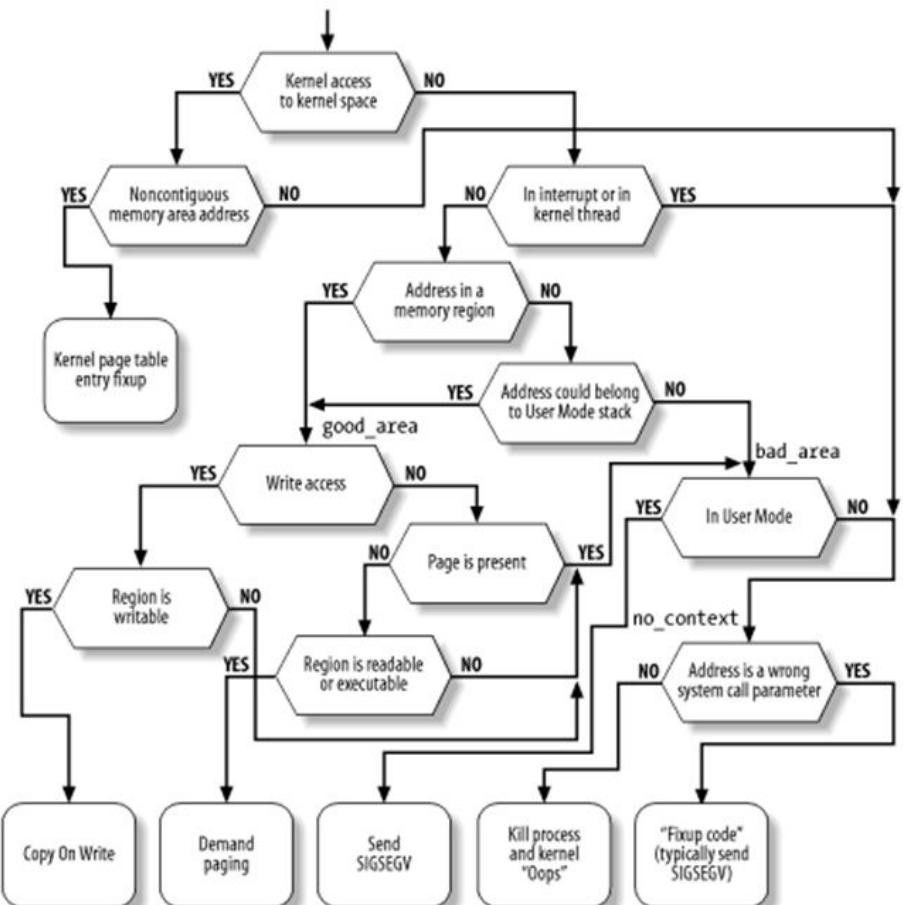
# Traitement des exceptions

## Workflow d'un page fault Linux



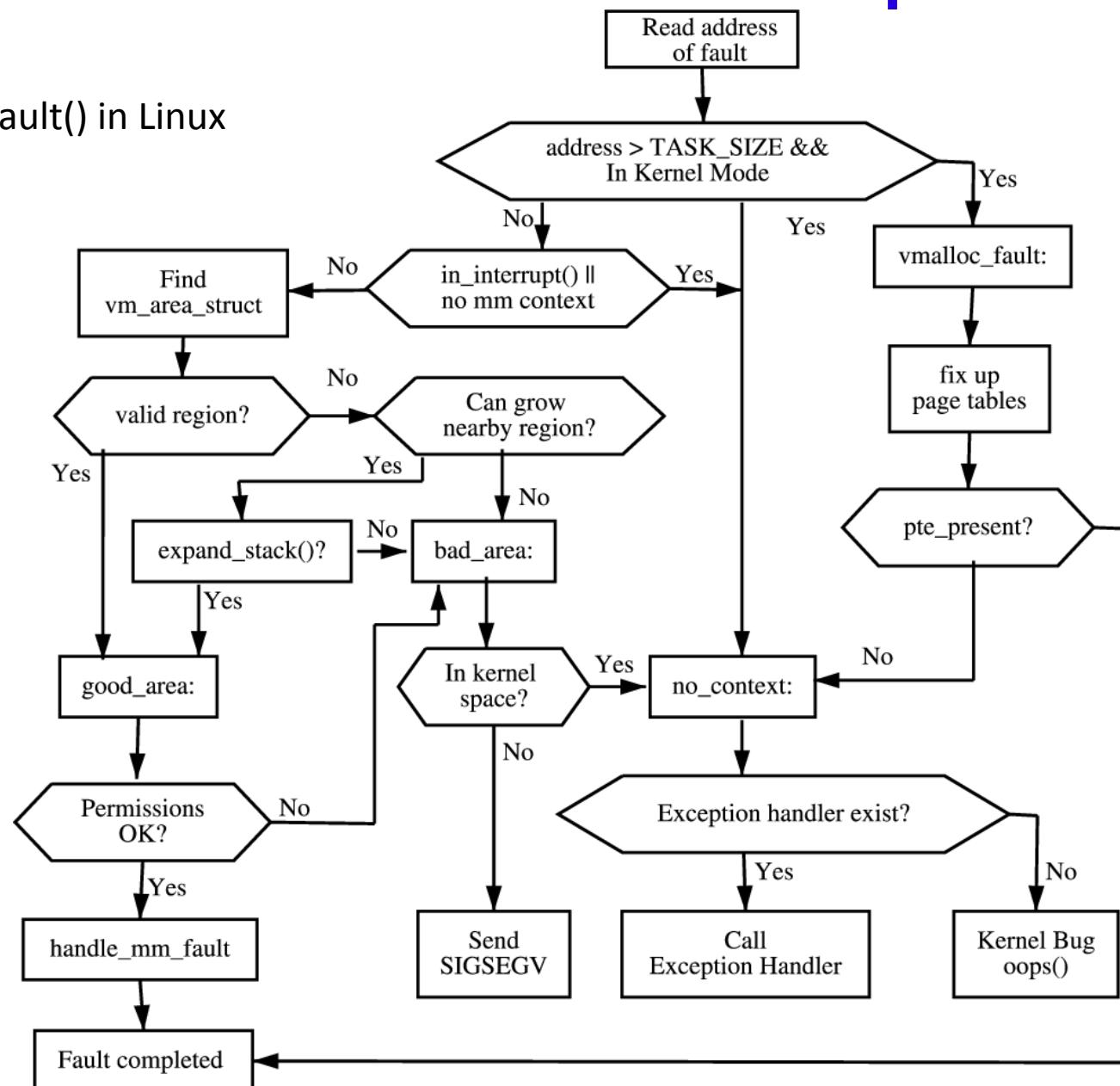
# Traitement des exceptions

## Workflow détails



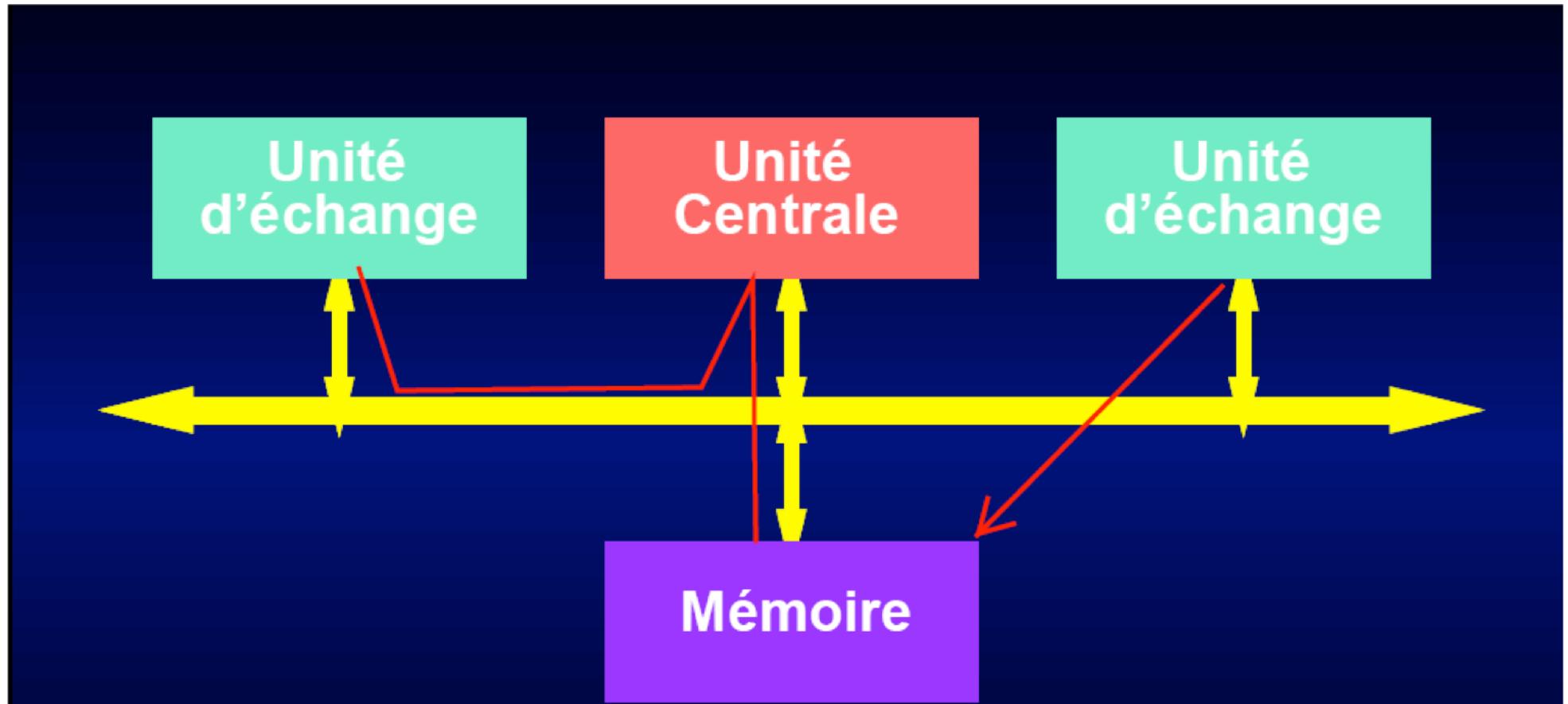
# Traitement des exceptions

`do_page_fault()` in Linux



# Concurrence, Parallélisme et Ordonnancement

# Structure Monoprocesseur



- Historiquement E/S pilotées depuis UC
- Puis déléguées à des contrôleurs de périphériques

# Direct Memory Access

- Permet de configurer un transfert de données entre la mémoire et un (contrôleur de) périphérique sans que les données transitent par le CPU
- Décharge le CPU => disponible pour d'autres tâches
- En général : on donne des adresses physiques au DMA
- Empêcher la Gestion Mémoire de changer le contenu de la page tant que le DMA est en cours (verrouillage)

# Déterminer un état

Exemple : le bloc disque N dont j'ai demandé la lecture est-il effectivement en mémoire ?

Polling :

Je pose continuellement la question jusqu'à ce que la réponse soit oui, ou qu'il y ait une erreur

N'ai-je pas mieux à faire ?

Interruption :

Le contrôleur de disque envoie un « signal » matériel au processeur quand son travail (lecture du bloc N) est terminé.

Le processeur déclenche l'interruption en exécutant une fonction configurée par le SE.

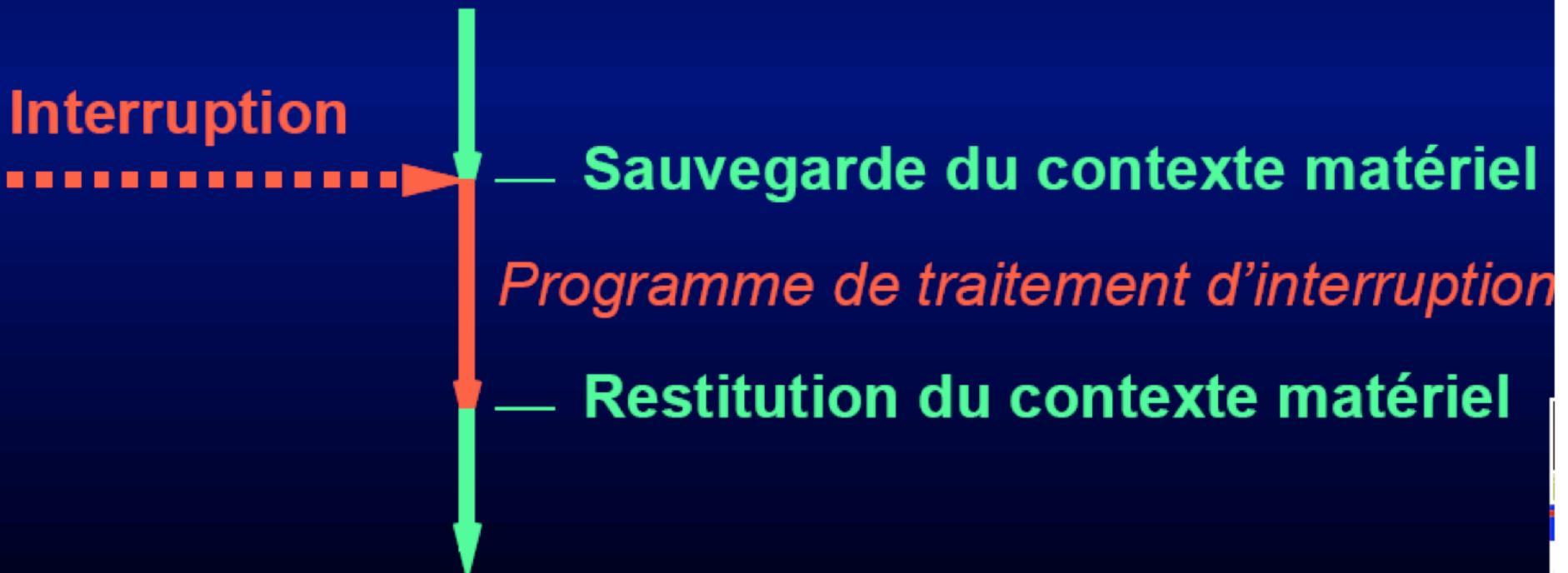
Entre temps, le SE peut permettre l'exécution de processus / threads..

# Interruptions matérielles

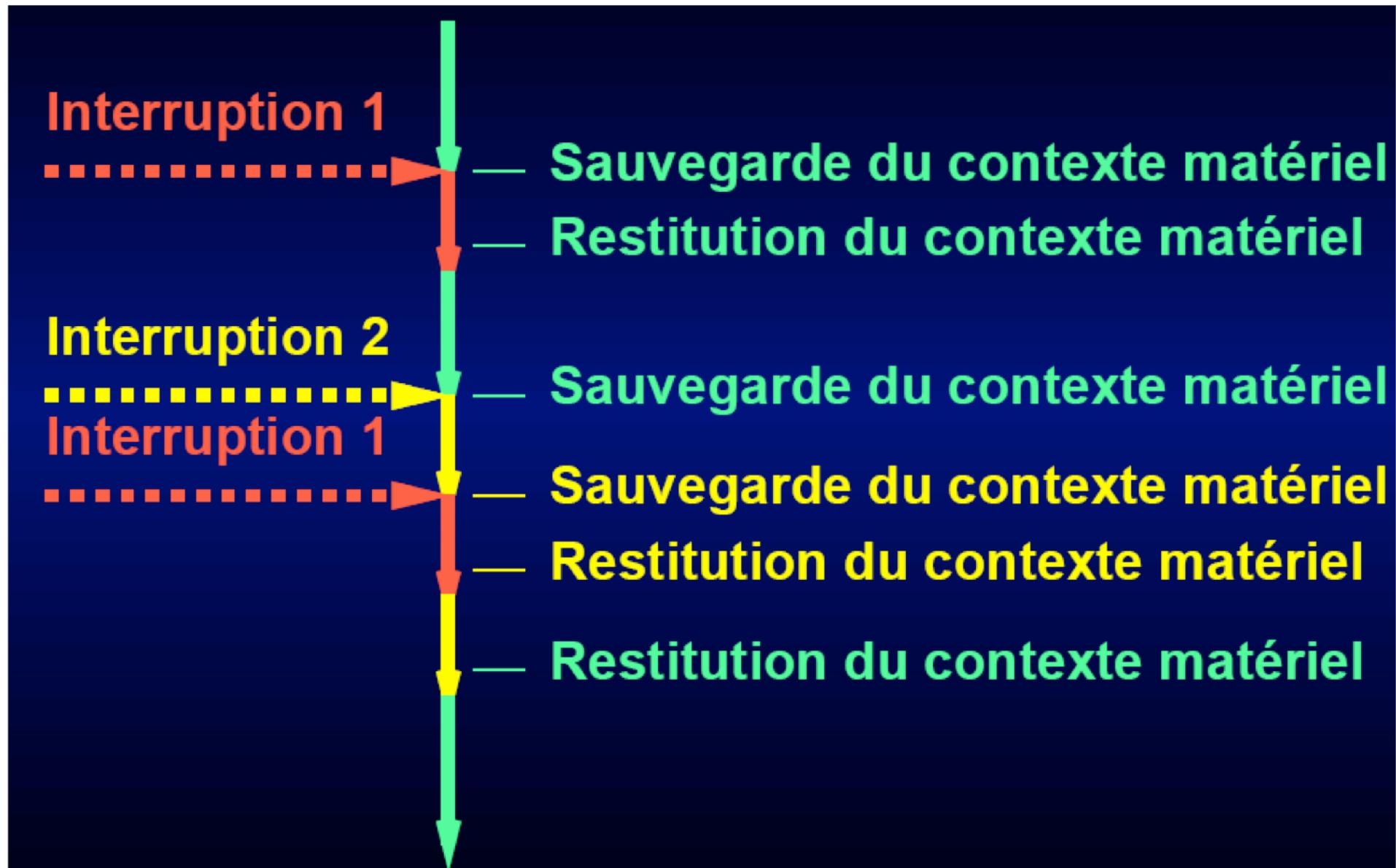
Besoin de surveiller et réagir

Comment éviter de passer son temps à surveiller les E/S ?

Interruption  $\equiv$  événement prioritaire qui interrompt déroulement normal d'un programme en cours d'exécution sur une unité centrale



# Interruptions imbriquées



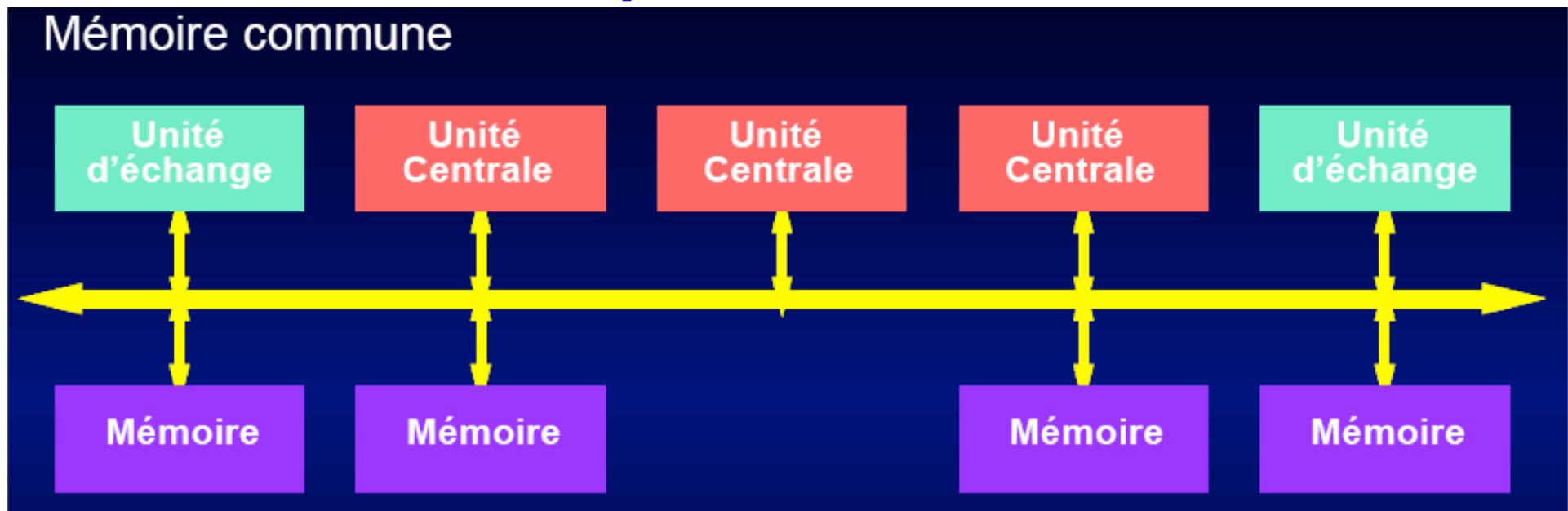
# Interruptions

- Types d interruptions :
  - Externes : matérielles déclenchées par du matériel externe : horloge, clavier, réseau, entrée sortie, fin de ligne vidéo, panne de courant,...
  - Internes : interruptions logicielles, déroutement Instruction de passage dans le noyau, division par 0, instruction inexiste, instruction non autorisée, accès mémoire invalide,...
- Le concept équivalent d'interruption existe au niveau logiciel : signaux Unix = fonction (handler) exécutée sur certaines conditions (erreur, communication possible,...),...

# Gestion des interruptions

- Contexte matériel : registres à sauvegarder quand survient une interruption
  - Compteur ordinal
  - Sommet de pile de programme
  - Mot d'état du programme
  - État du pipeline interne
- Dépend du processeur
- Possibilité de masquer les interruptions
- Éventuellement auto-masquage pour éviter récursivité infinie
- Niveaux (priorités) d'interruptions : Parer au plus urgent
- Structure de pile pour accepter un nombre arbitraire d'interruptions en cours

# Multiprocesseurs



- Plusieurs unités centrales pour augmenter la puissance: interprètent instructions des programmes applicatifs et du système d'exploitation
- Mémoire : plusieurs bancs pour augmenter taille et débit

# Noyau / SE

- « Première » couche de logiciel sur la machine
- Fournit des services génériques aux processus (tâches)
- Gestionnaire des ressources
- Permet d'exécuter plusieurs programmes / processus simultanément
  - Fournit le support pour la concurrence
- Le noyau n'a personne (sauf le matériel) pour l'aider

# Concurrence : le problème !

# Concurrence (problème : exemple)

```
#define MAXLOOP 100000000
static volatile int value;

void *my_func(void *arg) {
    for (int i=0; i < MAXLOOP; i++) {
        value++;
    }
    return NULL;
}
```

# Concurrence (problème : exemple)

```
int main() {  
    my_func(NULL);  
    my_func(NULL);  
    printf("Value: %d\n", value);  
}
```

```
ask> gcc -O0 -o test test.c  
ask> ./test  
Value: 200000000  
ask>
```

# Concurrence (problème : exemple)

```
int main() {  
    pthread_t th1, th2;  
    int res;  
    res = pthread_create(&th1, NULL, my_func, NULL);  
    res = pthread_create(&th2, NULL, my_func, NULL);  
    res = pthread_join(th1, NULL);  
    res = pthread_join(th2, NULL);  
    printf("Value: %d\n", value);  
}
```

```
ask> gcc -O0 -o test test.c  
ask> ./test  
Value: 101594476  
ask> ./test  
Value: 102256442  
Ask>
```

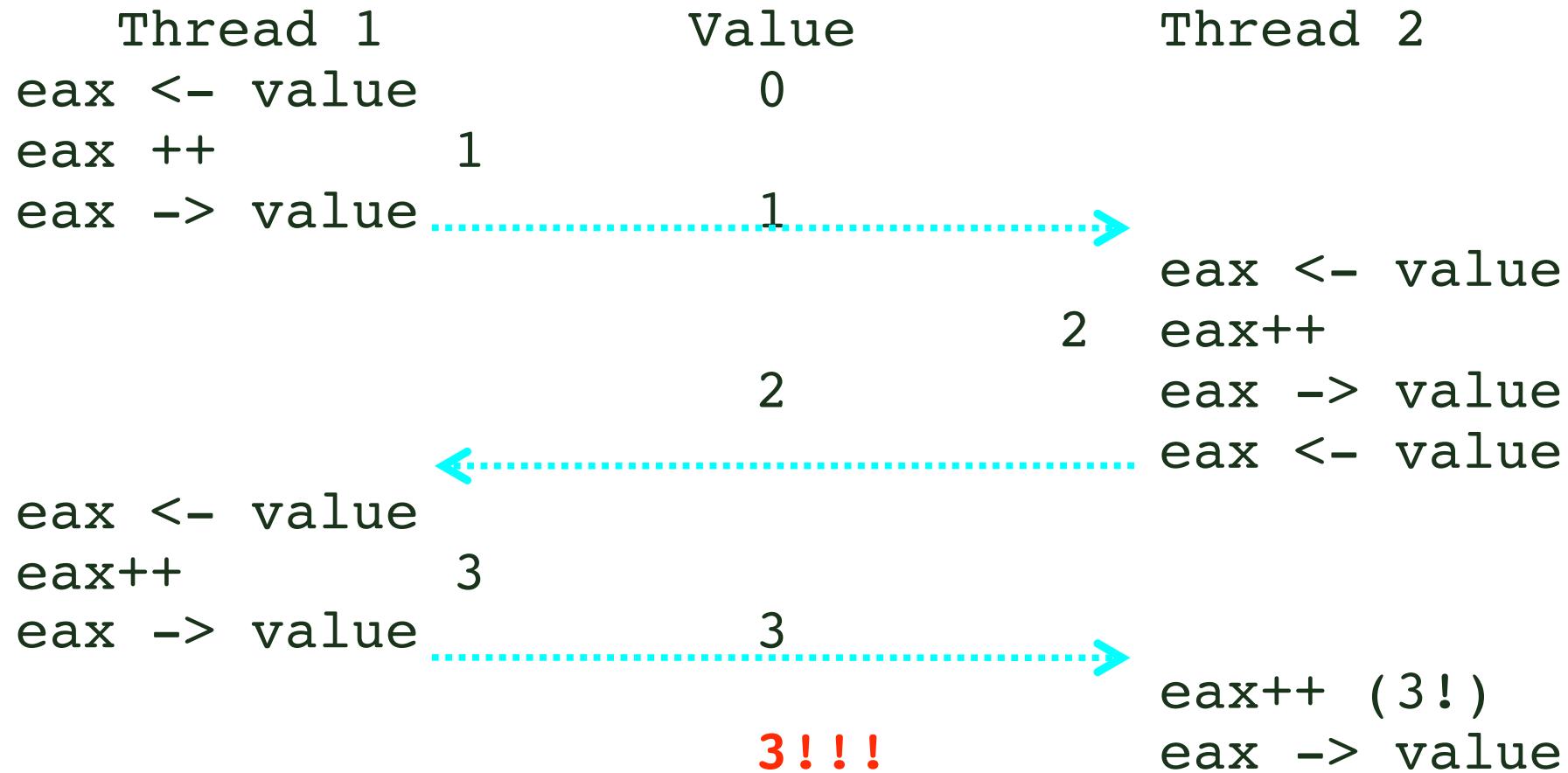
# Concurrence (problème : exemple)

## Traduction assembleur (sans optimisation!)

```
LBB0_1:                                ## =>This Inner Loop Header: Depth=1
    cmpl    $10000000, -12(%rbp) ## imm = 0x5F5E100
    jge     LBB0_4
## BB#2:
    movl    _value(%rip), %eax
    addl    $1, %eax
    movl    %eax, _value(%rip)
## BB#3:
    movl    -12(%rbp), %eax
    addl    $1, %eax
    movl    %eax, -12(%rbp)
    jmp     LBB0_1
LBB0_4:
    xorl    %eax, %eax
                                                ## kill: %RAX<def> %EAX<kill>
    popq    %rbp
    retq
```

## in Loop: Header=BB0\_1 Depth=1  
eax = value  
eax++  
value = eax  
## in Loop: Header=BB0\_1 Depth=1

# Dépendance sur l'ordonnancement



# Concurrence (problème : exemple)

Pire : on peut « revenir » en arrière!

```
ask> for i in 1 2 3 4 5 6 7 8 9
do
    ./test
done
Value: 104218758
Value: 100816077
Value: 96476592
Value: 99058844
Value: 102678228
Value: 101099097
Value: 94603564
Value: 104799101
Value: 89938246
Value: 95373723
Ask>
```

# Dépendance sur l'ordonnancement

Thread 1	Value	Thread 2
	0	
i=0 eax <- value 0		0 i=0 eax <- value
		1 eax++
		eax -> value
eax++ 1	1	
eax -> value	1	1 i=1 eax <- value
i=1 eax <- value 1		2 eax++
eax++ 2		
eax -> value	2	
...	...	
i=997 eax <- value 997	997	
eax++ 998		
eax -> value	998	eax -> value
i=998 eax <- value 2	2	
		2 i=2 eax <- value
		3 eax++
		eax -> value
	3	...
	999	
		999 i=999 eax <- value
		1000 eax++
		eax -> value
eax++ 3	1000	
eax -> value 3		
i=998 eax <- value 4	3	
eax++ 5		
eax -> value 5	5	
i=999 eax <- value 5		
eax++ 6	5	
eax -> value 6	6	

# Concurrence

On ne peut pas ne pas s'en préoccuper

Pour les applications :

un processus mono-thread pas de problème

Euh... délivrance des signaux ?

plusieurs threads, il faut agir

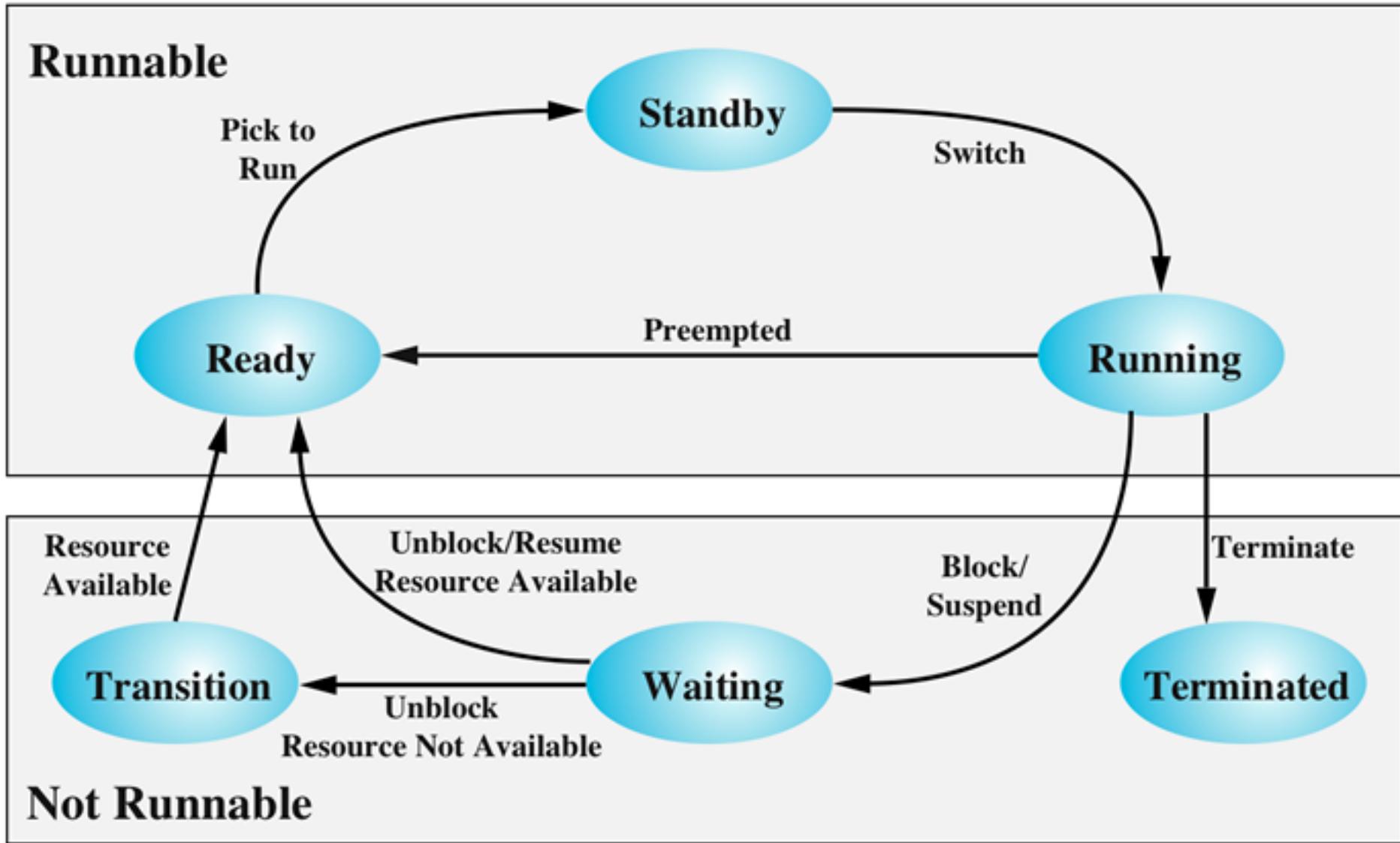
Pour le noyau

la concurrence est rude! les processus, les threads,  
les interruptions, les tâches internes...

# Rappels

- On n'exécute pas les processus (ou les threads) les unes après les autres (le temps du batch est révolu).
- De toute manière on a plein (trop?) de processeurs à disposition.
- En pratique (mono processeur pour rester simple) :
  - Un composant du noyau (l'ordonnanceur) attribue le processeur à chacun pour un temps donné limité (quantum ou tranche -slice- de temps).
  - Le choix est un choix « politique », il y a plusieurs politiques possibles (cf temps-réel)

# États d'un processus (Windows)



# États d'un processus (Linux)

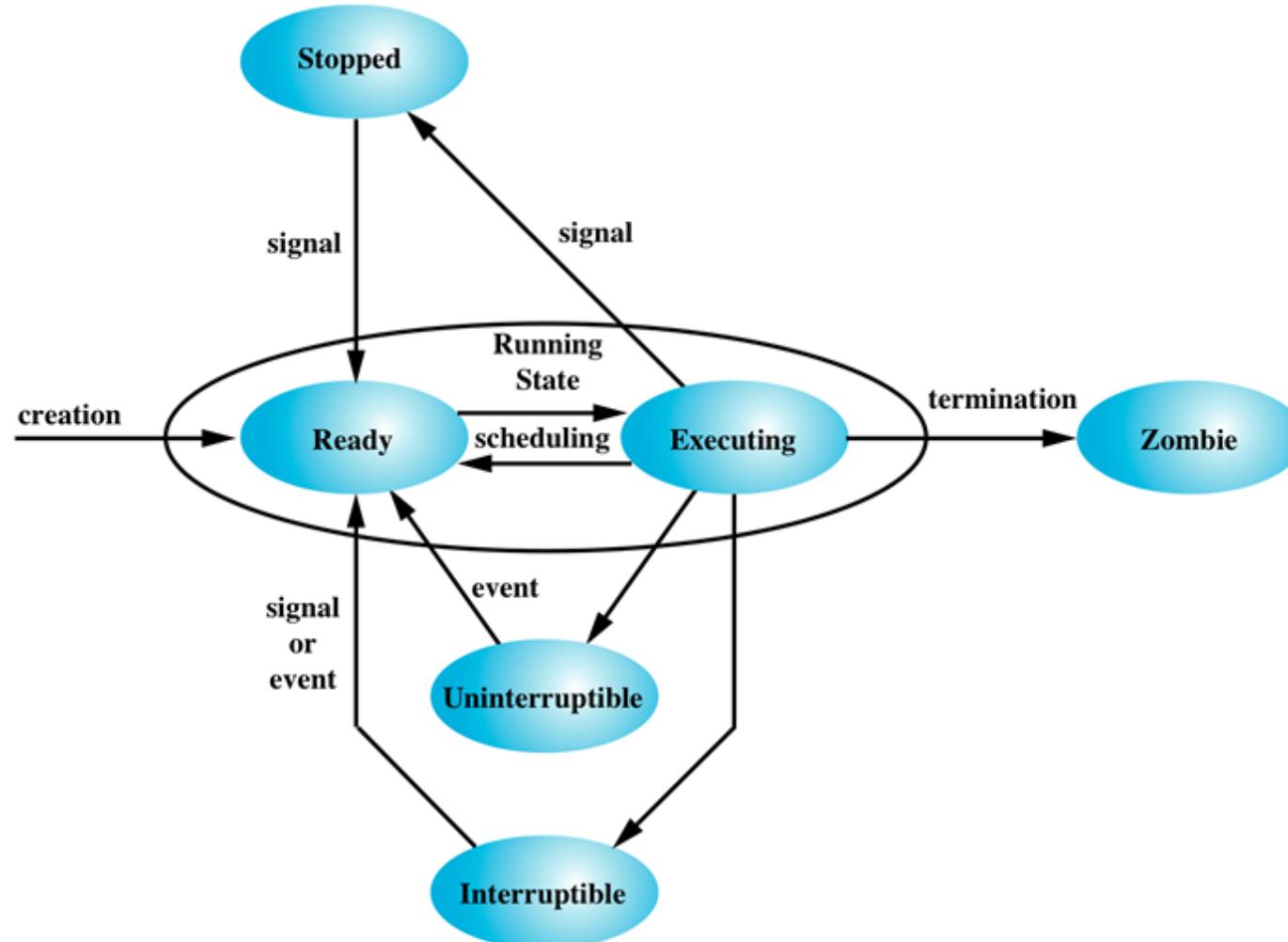


Figure 4.18 Linux Process/Thread Model

# États d'un processus (Unix)

zOMBIE

sLEEPING

rUNNING uSER

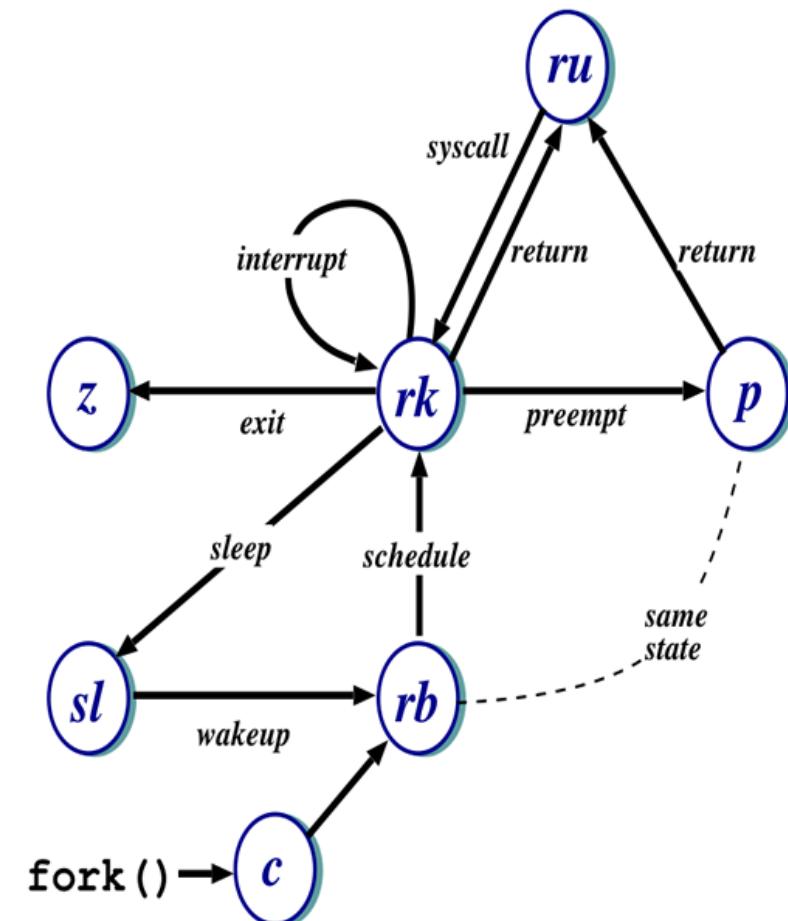
cREATED

rUNNING kERNEL

pREEMPTED

rUNNAbLE

Unix Process States



# Solutions au problème de concurrence

- ➊ Les outils de synchronisation
  - ➋ Mutex (exclusion mutuelle)
  - ➋ Spin-Locks
  - ➋ Sémaphores
  - ➋ Transactions (en général pas dans les OS)
  - ➋ Mémoire transactionnelle
  - ➋ HW / SW / Mixte
  - ➋ Algorithmes « lock-free »

# Synchronisation : mutex

## Services:

- `pthread_mutex_init`, `pthread_mutex_lock`
- `pthread_mutex_unlock`, `pthread_mutex_try_lock`

## Mutex:

- Ressource (structure mémoire) dans l'espace d'adressage du processus,
- En mémoire partagée entre 2 processus
- Ne pas confondre
- Avec les sémaphores (voir plus loin)
- Avec les sémaphores System V (`semget`)

# Concurrence (problème : exemple)

```
#define MAXLOOP 100000000
static volatile int value;

void *my_func(void *arg) {
    pthread_mutex_init(...);
    for (int i=0; i < MAXLOOP; i++) {
        pthread_mutex_lock(...);
        value++;
        pthread_mutex_unlock(...);
    }
    return NULL;
}
```

# Synchronisation : mutex

- Propriétés diverses:
  - Intra / Inter processus
  - Récursif, Sur, Détection de dépendance circulaire
  - Héritage de priorité...
  - Problèmes d'inversion de priorité
- Plus il y a de propriétés, plus l'acquisition coûte cher!
  - On essaye de faire que l'acquisition d'un mutex libre n'entraîne pas d'appel au noyau...
- Utilisation de « Test and Set » ou supports similaires

# Synchronisation : sémaphores

- Le mutex protège une section critique contre des exécutions concurrentes simultanées
  - Permet une sérialisation des traitements.
- Quid si on veut synchroniser des jobs d'impression vers 2 imprimantes?
  - Utilisation de sémaphore

# Sémaphore

- Un sémaphore est associé à une valeur
  - Lors de son initialisation
  - qui représente le nombre d'exemplaires de la ressources
- Pour pouvoir imprimer, il faut faire une opération **P()**, prendre, puis-je..., sur le sémaphore...
- Quand on a fini d'imprimer, on fait une opération **V()**, vendre, vas-y..., sur le sémaphore

# P() V()

Deux opérations atomiques...

```
V() :  
    cpt = cpt+1  
    Si ∃p ∈ attente reveiller(p)
```

```
P() :  
    Si cpt<0 attendre  
    cpt = cpt-1
```

# Concurrence au sein du noyau

- Entre processus / threads invoquant le noyau de manière concurrente
  - Multi-processeurs
  - Mono processeur :
    - 1 thread en attente (Ex : lecture de bloc disque)
    - 1 autre thread peut s'exécuter et faire un appel système.
- Entre threads et routines d'interruption !
  - Demande d'entrée sortie sur un disque
  - Et fin d'une entrée sortie disque précédente

# Concurrence au sein du noyau

- Linux 2.4
  - Big Kernel Lock : équivalent à un mutex unique
    - Acquisition lors de l'appel système
    - Relâché au retour
    - Lâché / Attente événement / Acquisition
  - Pas efficace !
    - Spécialement en multi-processeurs (Loi d'Amdahl)
    - Spécialement pour les applications temps-réel
- Linux plus récents : des verrous pour chaque objet
  - Descripteurs de processus, de fichiers, ...
  - Travail long et mise au point difficile

# Préemption

Propriété d'un système autorisant l'interruption temporaire (ou définitive) d'une tâche en cours afin de réaliser une tâche de priorité plus élevée

- dans la vie courante : répondre au téléphone, répondre à une question au milieu d'un cours...
- la tâche préemptée reprendra plus tard

La préemption n'intervient qu'à l'aide d'un événement extérieur

- fin d'E/S, horloge,...

# Préemption

La préemption (sans soin particulier) peut conduire à des incohérences

- modification de variables qui sont utilisées par le code préempté...
- modification de structures de données utilisées par le code préempté...

Il faut donc parfois se prémunir contre certaines préemptions

- introduction de délai dans la prise en compte des événements

# Préemption et concurrence au sein du noyau

se protéger contre les interruptions

- masquer/démasquer les interruptions

à manipuler avec précaution et sur des sections de code les plus courtes possibles

sinon (et parallèlement)

- exécuter le moins de code possible sous interruption  
si possible différer ce qui peut l'être dans des threads
- sans ces précautions : interrupt storm, live locking

# Ordonnanceurs

Time driven

L'ordonnanceur détermine quand il est invoqué  
généralement à intervalles constants  
overhead de l'ordonnancement

Event driven

déclenchement sur des événements

# FIFO

aka FCFS : First Come First Serve

Basique :

un nouveau job est placé en fin de file

le job à exécuter est retiré en début de file

# FIFO

J1 0-50

J2 50-60

J3 60-67

Job	Durée	Arrivée
1	50	0
2	10	0
3	7	0

Métrique temps d'attente moyen :

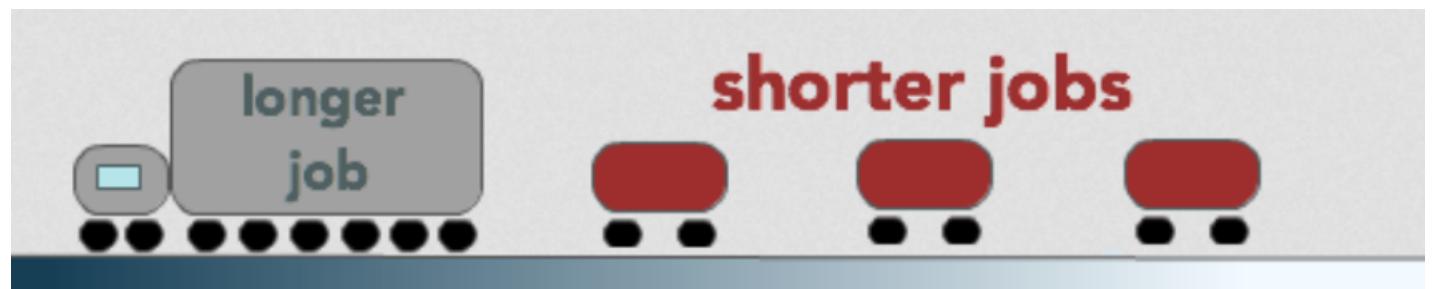
$$(0+50+60)/3 = 36,6\dots$$

Pas terrible...

# FIFO

Trop d'inconvénients :

- mauvais temps d'attente moyen
- mauvaise gestion des ressources
- effet « convoi »



# SJF

Shortest Job First

on fait passer les tâches les plus courtes avant les autres

des résultats optimaux (sous certaines conditions)

# SJF

J1 : 17-67

J2 : 7-17

J3 : 0-7

Temps d'attente moyen

$$(0+7+17)/3 = 8$$

Pas mal!

Job	Durée	Arrivée
1	50	0
2	10	0
3	7	0

# SJF

Inconvénient :

connaître le temps de calcul à l'avance...

ce n'est pas irréaliste mais pas toujours possible

Il en existe des variantes : SRT Shortest Remaining Time qui nécessite de la préemption

# Temps partagé

Le temps est vu comme une succession d'intervalles d'exécution (quantum) qui sont distribués aux différents processus au cours du temps



# Tourniquet (Round Robin)

C'est une FIFO avec préemption :

lorsque le quantum est épuisé le processus est placé en fin de file et le suivant est pris en tête de la file d'attente

équitable mais pas optimal

problèmes avec le temps-réel

# Tourniquet (Round Robin)

Quantum = 1

J1, J2, J3, J1, J2, J3, J1, J2, J2

Quantum = 2

J1, J1, J2, J2, J3, J3, J1, J2, J2

Job	Exécution	Début
1	3	0
2	4	0
3	2	0

# Tourniquet (Round Robin)

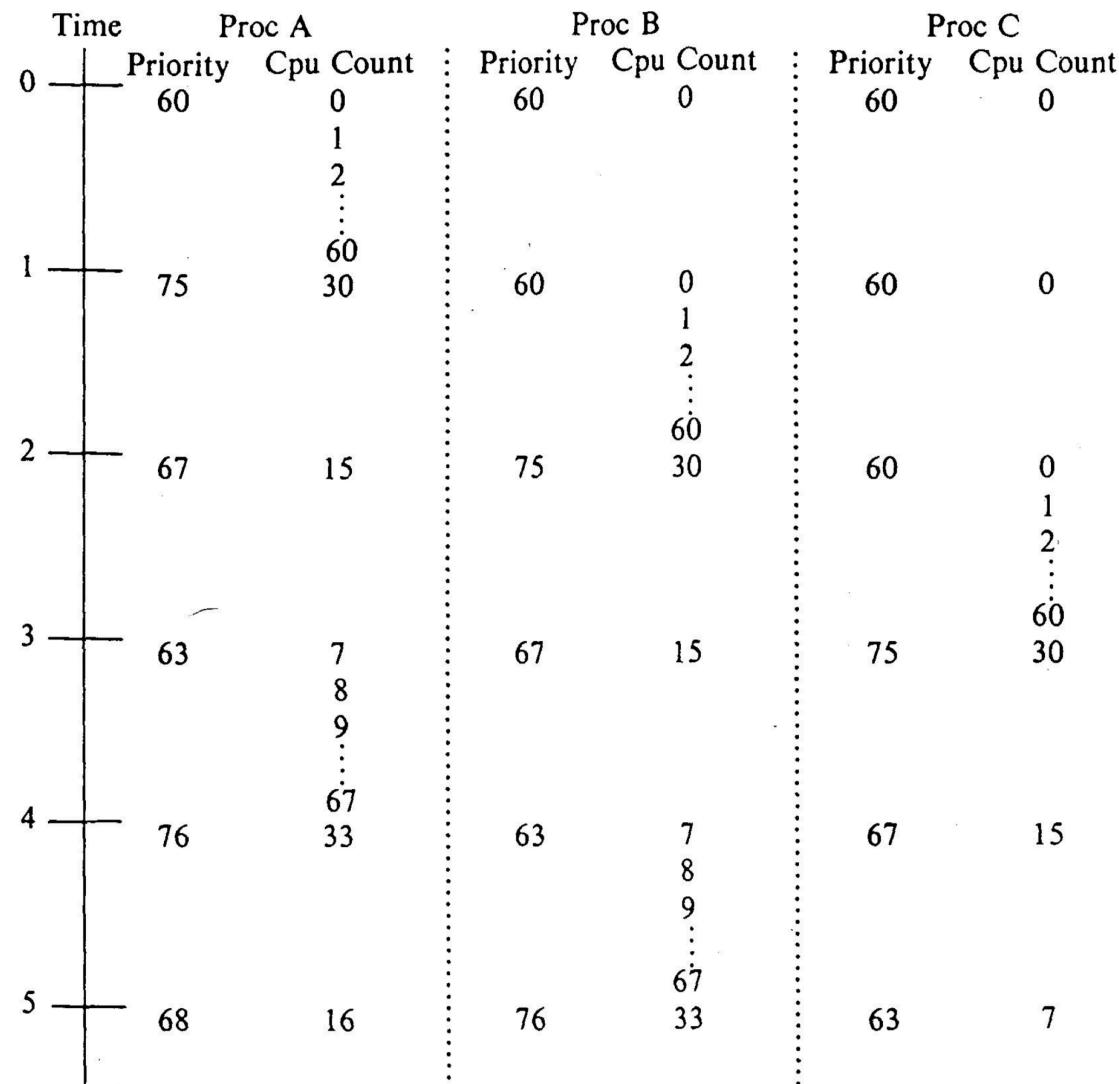
Partage du temps équitable entre processus de même priorité

partage avec priorité entre processus de priorité différentes...

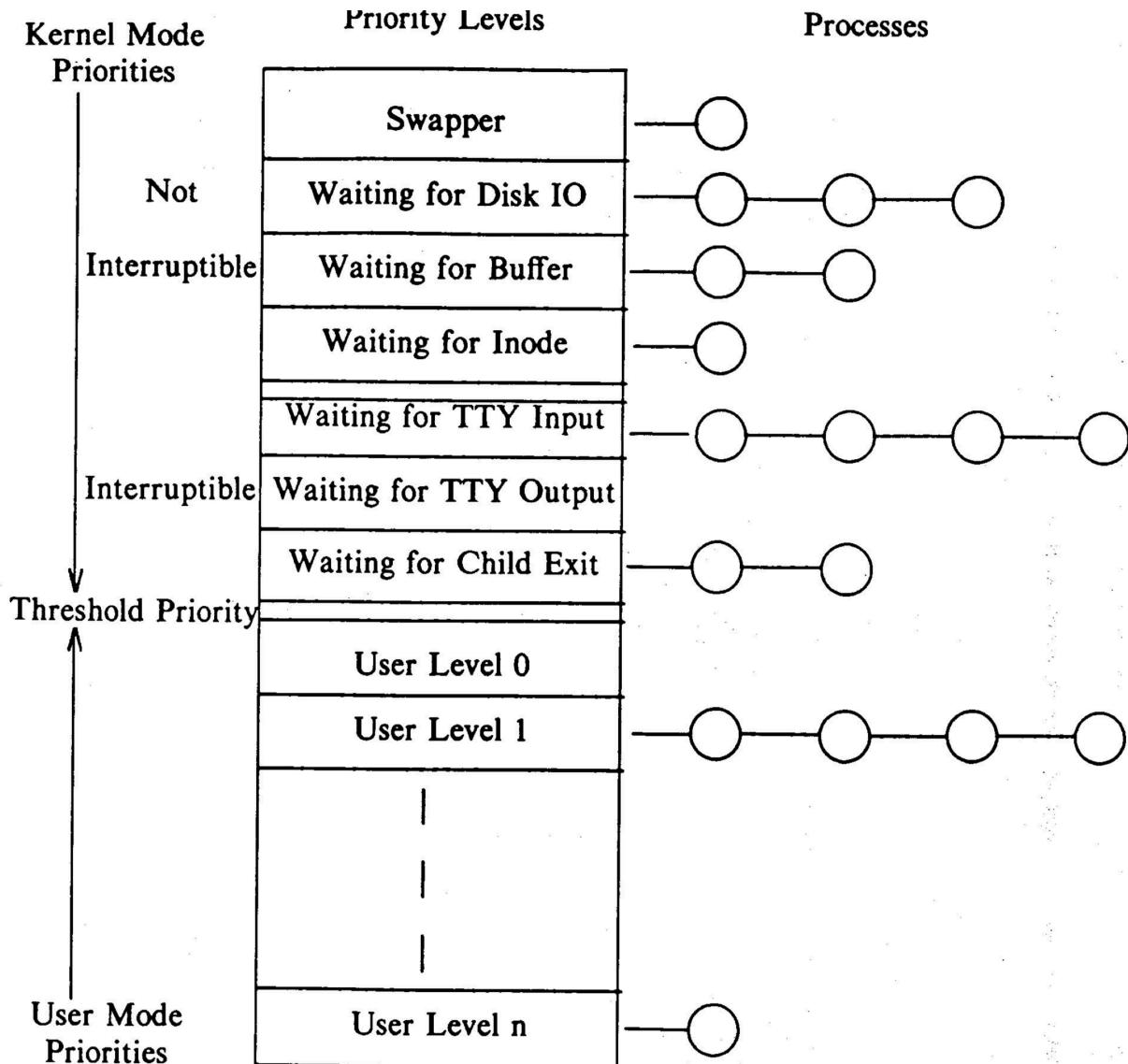
pas optimal pour le temps-réel



# Ordonnanceur Unix



# Priorités



# Earliest Deadline First

Quantum = 1

J1, J1, J1, J1, J2, J2, J2, J2, J3x10, J1x6

Quand J2 arrive il reste 29 pour J1, J2 préempte

Quand J3 arrive il reste 28 et 27 pour J1, J2 on continue avec J2...

Quand J2 termine il reste 25 pour J1 et 24 pour J3...

Job	Arrivée	Exécution	Contrainte
1	0	10	33
2	4	3	28
3	5	10	27

# Earliest Deadline First

Intéressant car optimal mais...

en cas de surcharge, effet avalanche : tout le monde rate sa deadline!

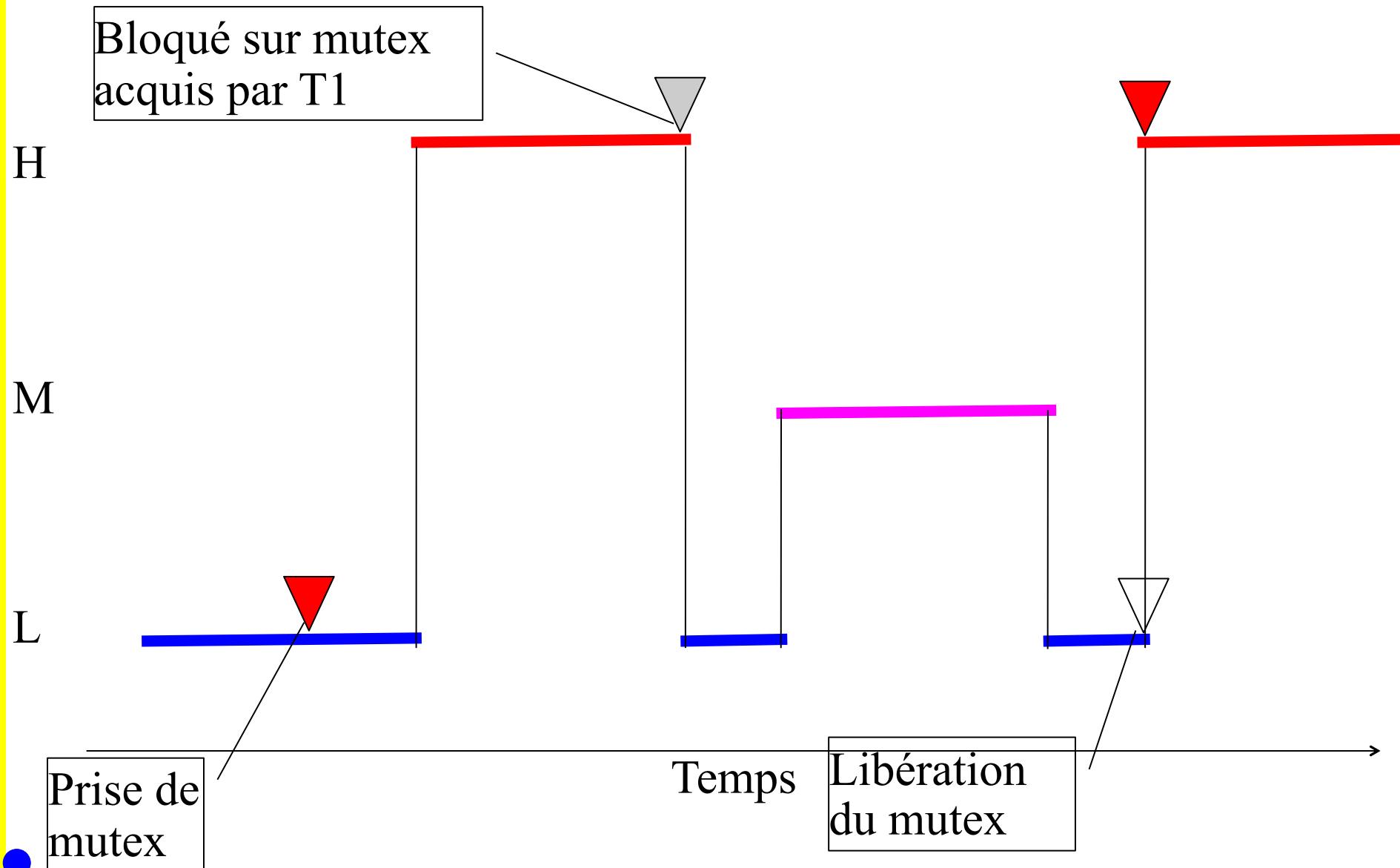
il faut donc

basculer vers un autre ordonnanceur

éliminer des tâches

dégrader le service

# Inversion de Priorité



# Inversion de Priorité

- Pour éviter ce genre de situation, changer momentanément la priorité de la tâche T1
  - Au moment où elle acquiert le mutex, lui affecter une priorité associée à la ressource protégée. T1 retrouvera sa priorité usuelle à la libération du mutex. « Priority ceiling protocol »
  - Au moment où T3 se bloque sur le mutex verrouillé par T1, on fait hériter T1 de la priorité de T3, jusqu'à la libération du mutex par T1... « Priority inheritance protocol ». Complexifie l'ordonnanceur.

# Linux et le temps-réel

- Linux supporte 3 types de processus / threads
  - SCHED\_NORMAL : Conventionnels (temps-partagé)
  - Temps-réel:
    - SCHED\_FIFO: Priorité fixe (First-In First-Out)
    - SCHED\_RR: Priorité fixe avec quantum de temps à priorité égale (Round-Robin)
- Les threads FIFO et RR sont plus prioritaires que les autres, et partagent la même plage de priorités (0..99). pas affectées par nice.
- Les threads temps partagé ont une priorité 100...139, affectée par nice. Le noyau utilise en fait une priorité dynamique (utilisation cpu, temps d'attente..)

# Linux et le temps réel

SCHED\_NORMAL un thread est suspendu si

- accès à ressource (I/O, mutex, etc)
- rend la main de lui-même (yield, sleep)
- son quantum a expiré

SCHED\_FIFO

- accès ressource
- rend la main de lui-même
- préempté par thread de priorité plus élevée

SCHED\_RR

accès ressource

- rend la main lui-même
- préempté par thread de priorité plus élevée
- quantum expiré

# Linux et le temps-réel

- Pour « créer » un processus temps-réel, il faut être super-utilisateur.
  - Pourquoi?
- Utilisation des appels POSIX pthread
  - pthread\_attribute au moment de la création de la thread
  - Changement dynamique après la création de la thread.

# Linux et le Temps-réel

- Linux est-il temps-réel?
  - Comprendre:
    - Linux est-il adapté pour permettre à des applications ayant des contraintes temps-réel de s'exécuter sans problème.
  - Caractéristiques attendues
    - Déterminisme
    - Faible latence, « préemptabilité »
    - Ordonnanceur approprié
    - Différents services (timers, synchronisation, communication...)

# Linux, Préemption

- Possibilité de permettre à une tâche prioritaire de démarrer immédiatement son exécution
- A l'opposé, imposer d'attendre que l'exécution en cours atteigne un point où l'on peut procéder à l'ordonnancement
- Linux 2.4:
  - Peu de points de préemption... Exécution en mode système ressemble à une « grosse section critique », ordonnancement lors du retour vers le mode utilisateur

# Linux 2.6

- Incorporation de certaines des modifications contenue par le patch PREEMPT-RT
  - Ordonnanceur O(1)
  - Amélioration des points d'ordonnancement internes
  - High-Precision Timers
- Reste entre autres:
  - Traitement des interruptions

# Améliorer la Préemption

- Minimiser:
  - La durée des sections critiques (pendant lesquelles les interruption sont masquées par le code de base)
    - Réduit la latence de prise en compte
  - Le traitement effectué sous interruption
    - Pendant que l'on traite une interruption, soit toutes les interruptions sont masquées, soit l'interruption que l'on traite est masquée.
    - Ex: si le traitement d'un interruption horloge prend plus d'un « tick », le système ne sera pas à l'heure et inutilisable.
- Permettre un ordonnancement en retour d'interruption

# Linux, PREEMPT-RT

- Permettre à des threads définies par l'utilisateur de s'exécuter à des priorités plus élevées que le traitement des interruptions.
- Crée un thread « noyau » par niveau d'interruption.
- L'administrateur système peut ajuster les priorités.
- Augmente la charge du système: passage obligatoire par des threads, coût d'ordonnancement.

# Linux, PREEMPT-RT

- Attention! Un marteau est un très bon outil, à condition de savoir s'en servir et de ne pas se taper sur les doigts!
- Le noyau 2.4 n'est pas préemptif,
- Pas possible de tourner de véritable apps TR dures
- Certaines des modifications ont été incorporées au noyau 2.6
- Reste toujours la nécessité d'un patch suivant les besoins applicatifs

# Linux 2.6

- Ordonnanceur en O(1)
- Remplacé par CFS depuis 2.6.23
- Offre 3 modes de préemption (config)
  - Défaut (pas de préemption)
  - Préemption Volontaire
  - Préemption « involontaire »
  - Encore insuffisant (y compris pour audio)
- 2.6.18 : héritage de priorité
- 2.26.21: High-resolution timers (Posix) ~1µ sec.

# Patch PREEMPT-RT

- Introduit un niveau de préemption « complète »
- Permet des latences encore plus courtes  
~100µsec
- Impact global sur performances
- On passe plus temps en changement de contexte
- Problèmes:
  - Pas dans l'arbre référence (résistance!)
  - Pilotes de périphériques à adapter (quid si binaire?)

# Linux 3.14

Temp-réel

Ordonnanceur EDF avec SCHED\_DEADLINE

Toutefois le manuel indique

0. WARNING

=====

Fiddling with these settings can result in an unpredictable or even unstable system behavior. As for -rt (group) scheduling, it is assumed that root users know what they're doing.

# Ordonnancement et multiprocesseurs !

- Équilibrer (répartir) la charge entre les processeurs...
- Une thread peut s'exécuter sur le proc 1 puis sur le proc 2...
- Oui, mais les données en cache ne suivent pas...
- Défaut de cache => exécution ralentie !
- Notion d'affinité
- Demander à lier l'exécution sur un (ou plusieurs) processeurs.

# Ordonnancement et multiprocesseurs !

- Le mieux peut être l'ennemi du bien
- Une file unique et commune de processus à exécuter ?
- La file doit être manipulée avec précaution :
  - Section critique, goulot d'étranglement !
- En pratique
- Une file par processeur : pas de contention
- Et une file commune : contention moins fréquente

# Gestion des fichiers

- Types de fichiers:
  - Régulier, répertoire, liens symboliques, device bloc, caractère, tubes nommés ? Autres?
- Propriétés:
  - Droits d'accès, propriétaire, liens, taille, dates
- Ouvrir, lire/écrire, fermer, créer, supprimer, renommer... "mapper" en mémoire (mmap)

# Besoins

- Pleins de "petits fichiers" (production de logiciel)
- Manipulation de gros fichiers (images, video, audio)
- Très gros fichiers (bases de données)
- Accès séquentiel, direct, aléatoire...
- Taille du fichier et/ou des disques:
  - $> 4G => > 2^{32}$  octets

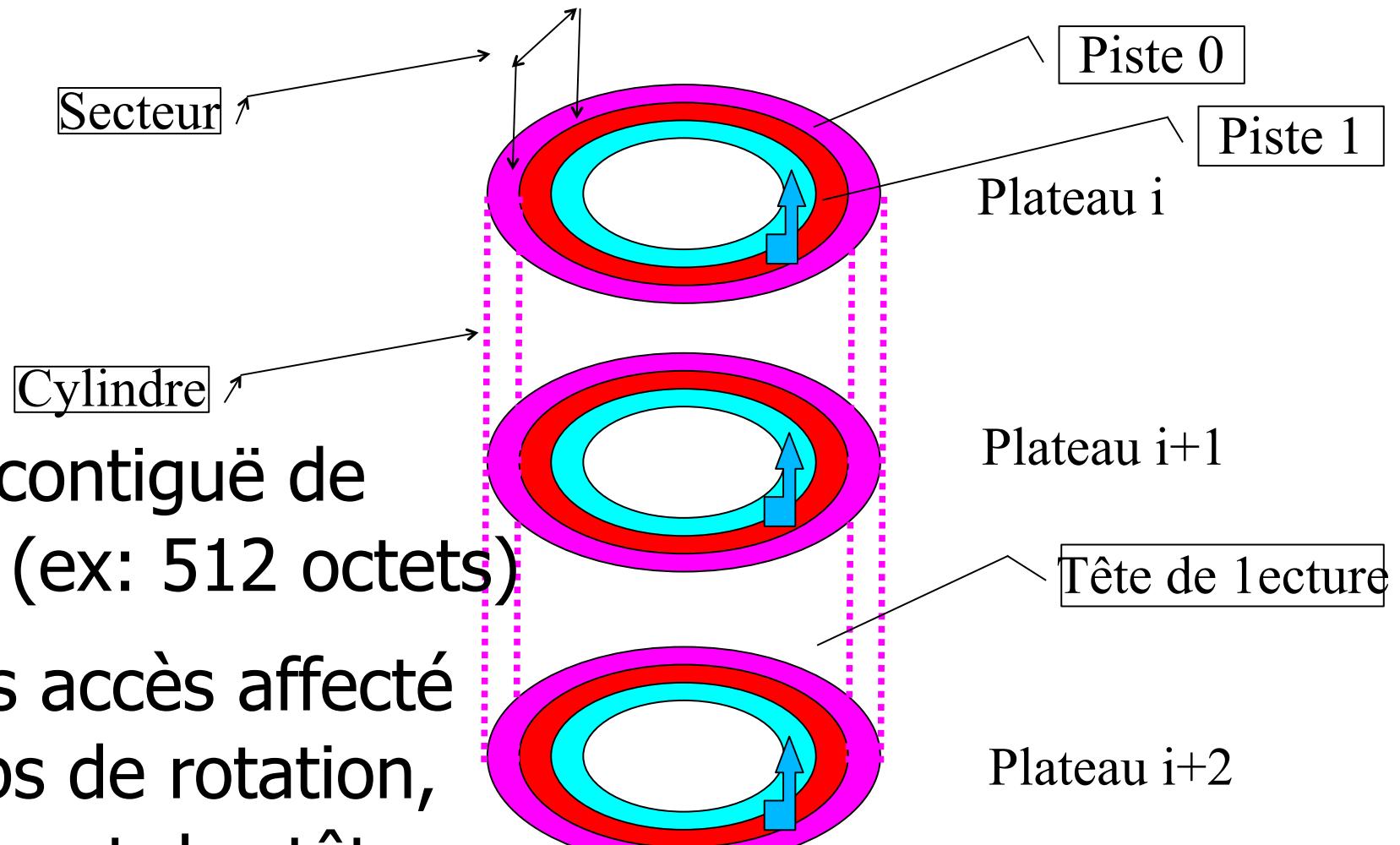
# Besoins

- Stocker plusieurs systèmes de fichiers sur un seul disque, un système de fichiers sur plusieurs disques
- Avoir des tailles de systèmes de fichiers adaptables en fonction des besoins:
  - Augmenter / diminuer la taille (dynamiquement) des disques :-) et des systèmes de fichiers
- Résister aux pannes / erreurs des disques
- Faire des sauvegardes / restaurations

# Systèmes de Fichiers

- Systèmes de fichiers "historiques": FAT, vFAT, UFS, S5, ext2
- Systèmes plus récents: ext3, extf4, reisferfs, xfs, Btrfs, ZFS, NTFS, APFS...
- Réseau: NFS v2, v3, v4, SambaFS, Coda, Lustre...
- Flash:JFFS, JFF2, YAFFS...
- Autres: Procfs, sysfs, ....
- Voir [http://en.wikipedia.org/wiki/List\\_of\\_file\\_systems](http://en.wikipedia.org/wiki/List_of_file_systems)
- [http://en.wikipedia.org/wiki/Comparison\\_of\\_file\\_systems](http://en.wikipedia.org/wiki/Comparison_of_file_systems)

# Disques Magnétiques

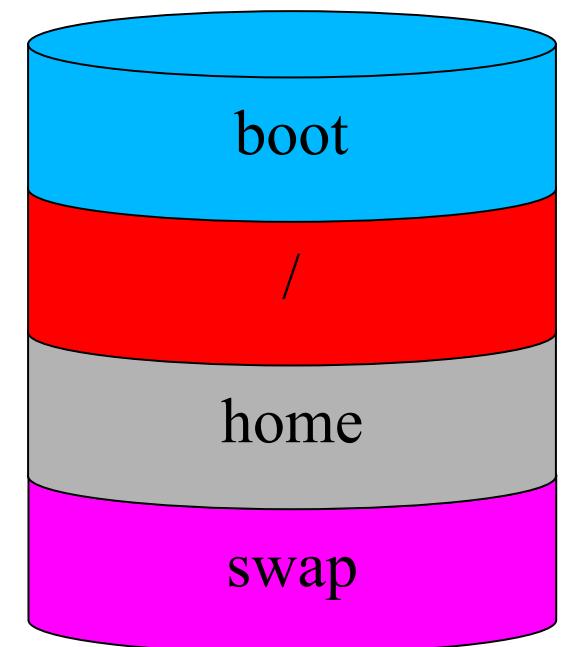


- Suite contiguë de secteurs (ex: 512 octets)

- Temps accès affecté par temps de rotation, déplacement des têtes d'un cylindre à l'autre

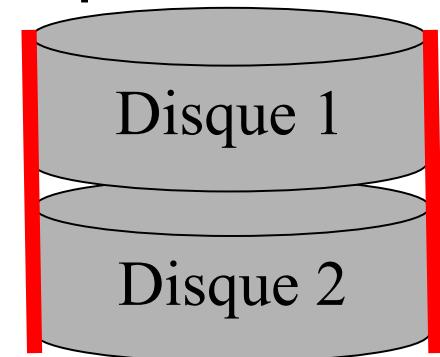
# Utilisation des disques

- Partitionnement: découpage d'un disque en sous - ensemble (partitions)
  - Flexibilité de gestion, sauvegarde
  - Flexibilité en cas d'erreur
  - Pb de redimensionnement
- Plusieurs types de partitionnement
  - Monde Microsoft, BSD, Solaris...
  - Microsoft:
    - 4 partitions physiques
    - 3 physiques + 1 contenant partitions "étendues"



# Utilisation des disques

- Agrégation: inverse de la partition
  - Considérer 2 disques comme n'en étant qu'un seul.
  - Les disques agrégés peuvent être
    - Des disques physiques
    - Des partitions
  - On pourrait ensuite partitionner un tel disque agrégé!
- Linux: LVM (Logical Volume Manager), Veritas VxVM, ...



# Résister aux erreurs, Performances

- Pour résister aux pannes (crevaisons) une voiture a une roue de secours (en principe)
- Pour résister aux pannes / erreurs disques, il nous faut des disques supplémentaires
- Diverses possibilités d'agencement / coût

# Système RAID

Redundancy of Inexpensive Array Disk

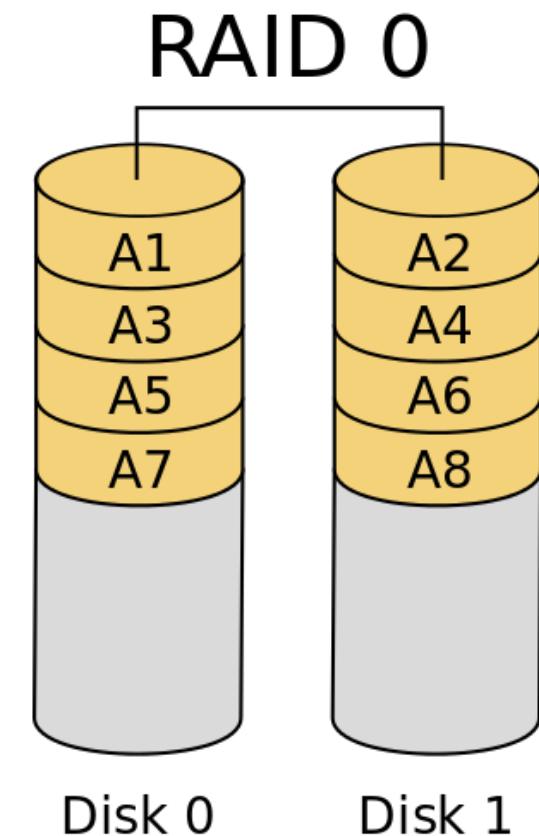
l'idée originale est d'agréger des disques afin  
d'obtenir des avantages divers

pas forcément la tolérance aux pannes...

# RAID 0

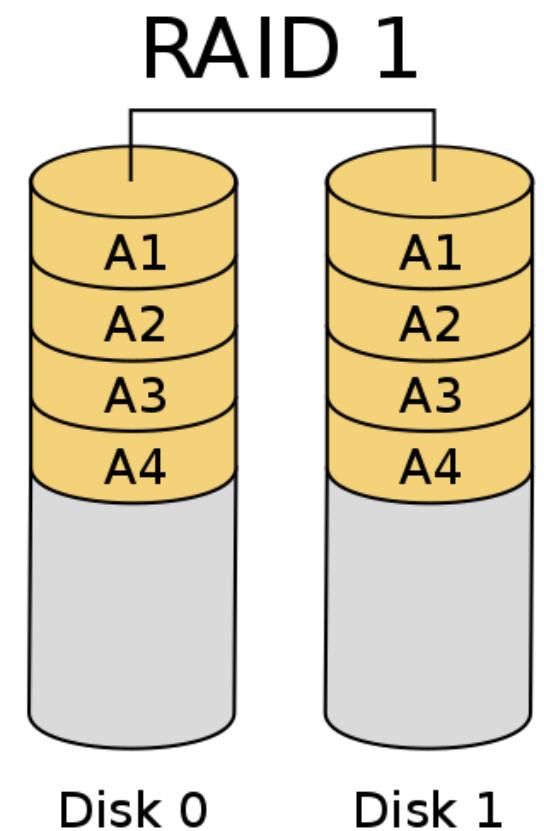
Ce niveau n'offre pas de tolérance aux pannes, il permet d'obtenir des gains intéressants de performances en I/O

parallélisme...



# RAID 1

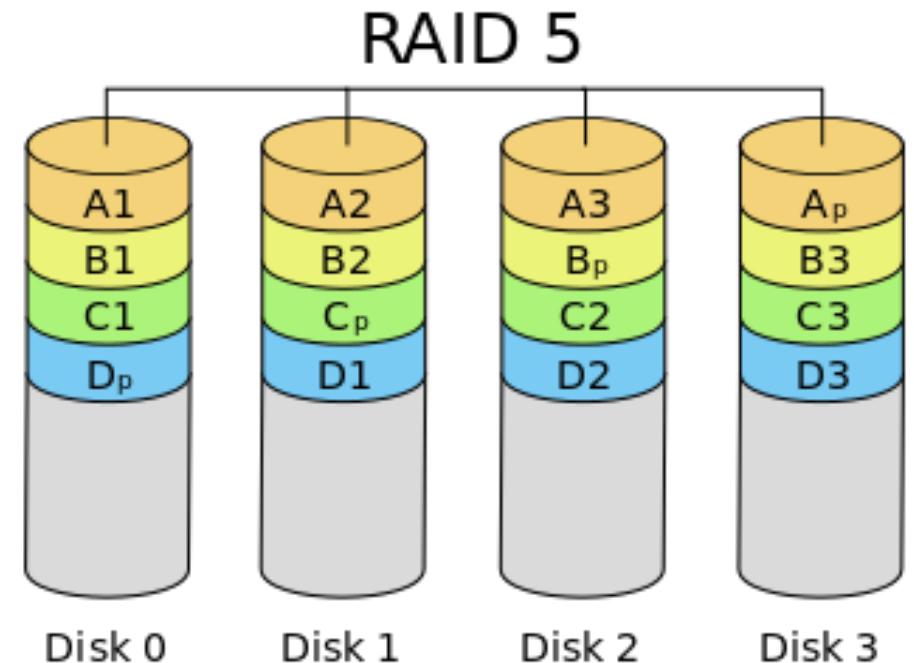
La redondance de secours...  
aucun gain d'espace!



# RAID 5

L'agrégation répartie du RAID 0 avec le contrôle d'erreur

$$X_p = X_1 + X_2 + X_3 + \dots + X_n$$



# Stocker les informations

- L'utilisateur manipule :
  - Fichiers (données et meta-données)
  - Une arborescence (nom de fichiers, chemins d'accès)
- Principe général:
  - Un Super-bloc: "racine" de la description de l'arborescence
  - Des Inodes: descripteurs de fichiers (méta-données et accès aux données)
  - Des répertoires associant noms symboliques et inodes

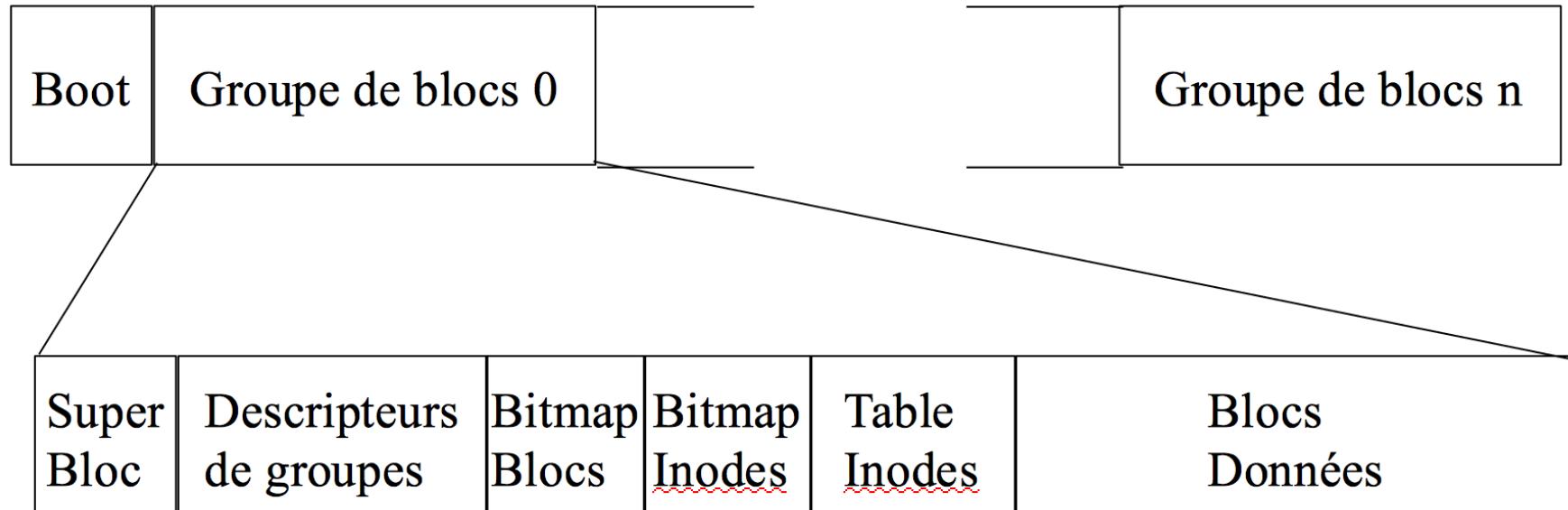
# Rôles du SGF

- Organiser la représentation physique des données sur le disque (spécifique du SGF)
- Accéder aux données sur le disque (communs à tous les SGF)
- Fournir les services invoqués par l'utilisateur: open, read, write
  - Partiellement commun à tous les SGF
  - Partiellement spécifique à chaque SGF
- Compromis latence vs débit (as usual!)

# Ext2 fs

- Choix de la taille du bloc (1K, 2K, 4K) au mkfs
  - 1K : peu de « perte » mais beaucoup d'accès disque
  - 4K : accès plus rapide mais plus de « perte »
  - « fragmentation »
- Nombre d'inodes
  - Par défaut ou choisi à la création
- Pré-allocation (non persistante)
- « Fast » Symbolic links (< 60 caractères)

# Ext2 : Structures sur disque



- Bitmaps (blocs et inodes) sur 1 bloc
- 1K => 8192 blocs (de 1 K)
- 4K => 32768 blocs (de 4K)

# Ext2 : Super Bloc, qq champs

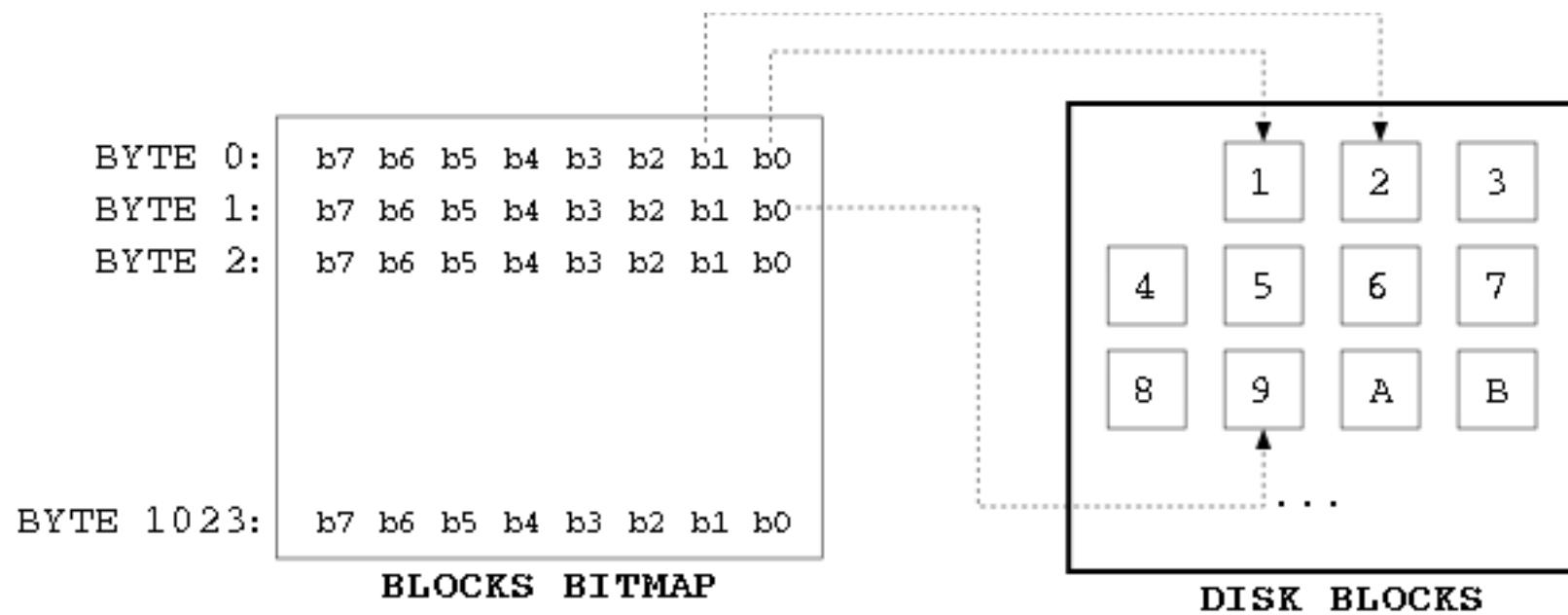
- inodes\_count, blocks\_count
- free\_blocks\_count, free\_inodes\_count
- block\_size, frag\_size
- blocks\_per\_group, inodes\_per\_group
- mount\_count, max\_mount\_count
- check\_interval, last\_checked
- state, magic
- inode\_size

# Ext2 : Descripteur de groupe

- N° bitmap bloc, N° bitmap inode
- N° 1er bloc inodes
- Blocs libres (groupe), inodes libres (groupe)
- Nombre de directories dans le groupe

# Ext2fs : bitmaps

Même mécanisme blocs/inodes



# Ext2 : Inode

- Mode (type et permission)
- Uid, Gid
- Size, nombre de blocks
- Atime, mtime, ctime, dtime
- Nombre de liens ???
- Blocs de données [15 ] !?!

# Ext2 : types de fichiers

- 0 inconnu
- 1 ordinaire (regular)
- 2 répertoire (directory)
- 3 character device
- 4 bloc device
- 5 tube nommé (named pipe)
- 6 socket
- 7 lien symbolic (soft link)

# Répertoires

Un répertoire n'est finalement qu'un fichier comme un autre

ses accès sont contrôlés différemment par l'API pour éviter les problèmes de cohérence

la structure de ses données est exploitée par l'OS

Ils constituent une partie critique et fragile des SGF

c'est l'espace de nommage des fichiers!

# Ext2 : répertoire

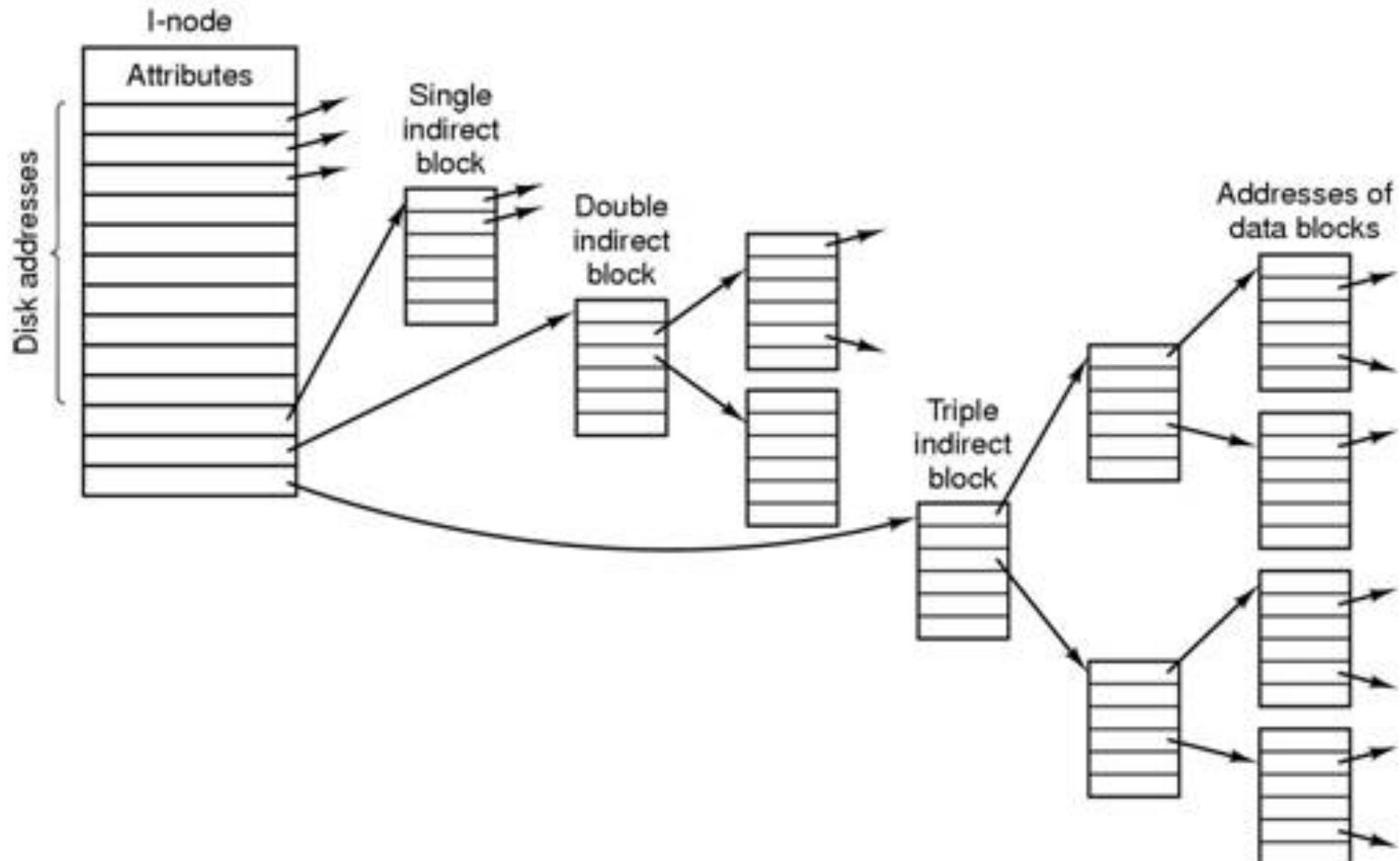
entry length				
4o	2o	1o	1o	name length
inode n°	entry length	name length	type	name

- Le nom est complété sur multiple de 4 octets avec des \0
- Longueur une entrée : 1octet => 255 car. max
- Une entrée détruite dans le répertoire est marquée avec un n° d'inode à 0

# Ext2 : entrée de répertoire

	inode	rec_len	name	file_type	name_len
0	13	12	1 2 . \0 \0 \0		
12	2	12	2 2 . . \0 \0		
24	18	16	m u s i c \0 \0 \0		music/
40	15	16	t e s t . t x t		test.txt
56	19	12	b i n \0		bin/

# Ext2 : blocs de données



# Ext2 : Allocation des inœuds

Pour les répertoires

Si le répertoire père est /

on le place dans un groupe relativement libre

Sinon dans le groupe de son père, sinon où l'on peut

Pour les fichiers

Groupe du père tant que possible

Sinon où l'on peut

# Ext2 : Allocation des blocs

On essaie d'allouer des blocs consécutifs

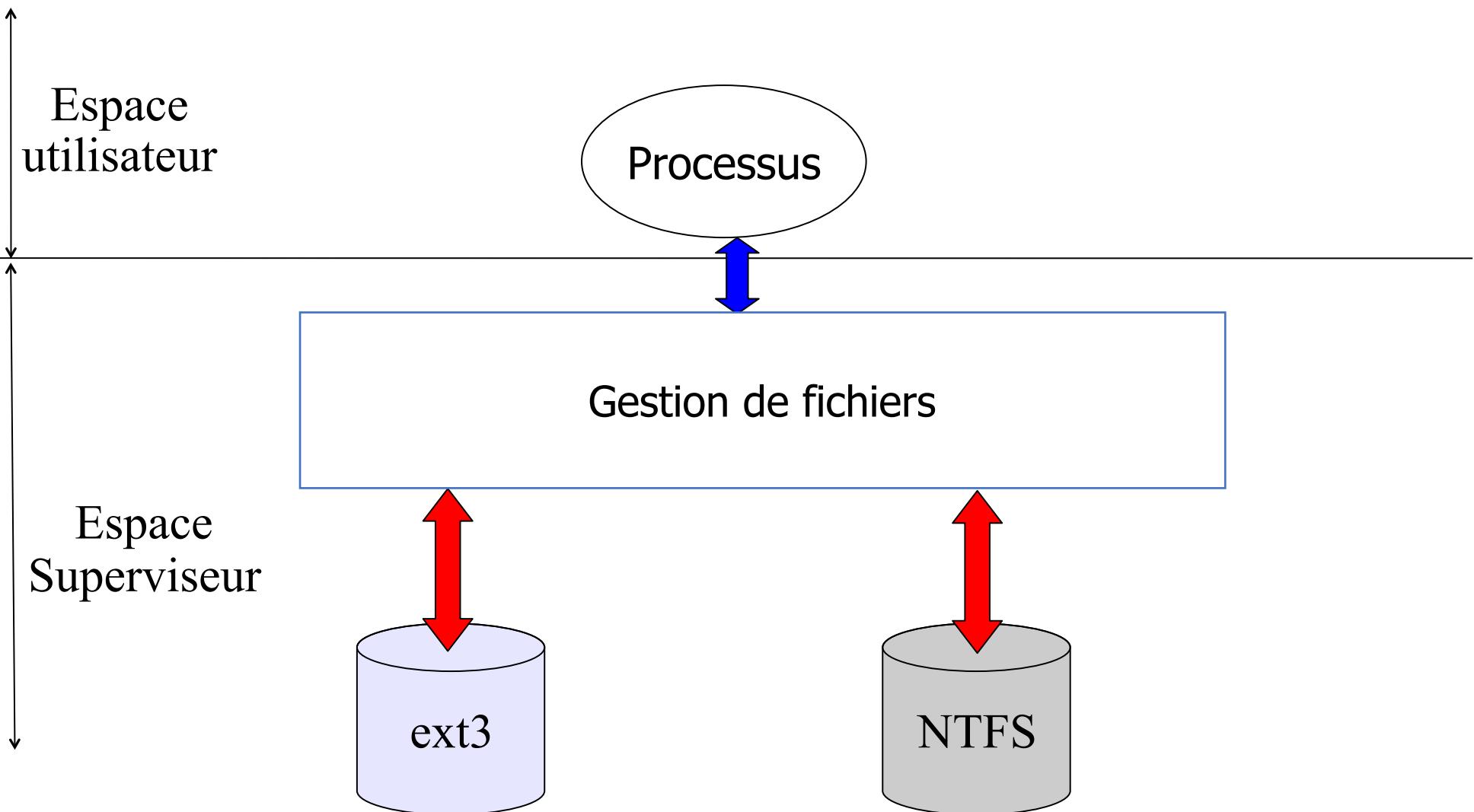
On essaie de rester dans le même groupe

À défaut on essaie d'allouer là où il y a au moins 8 blocs libres consécutifs

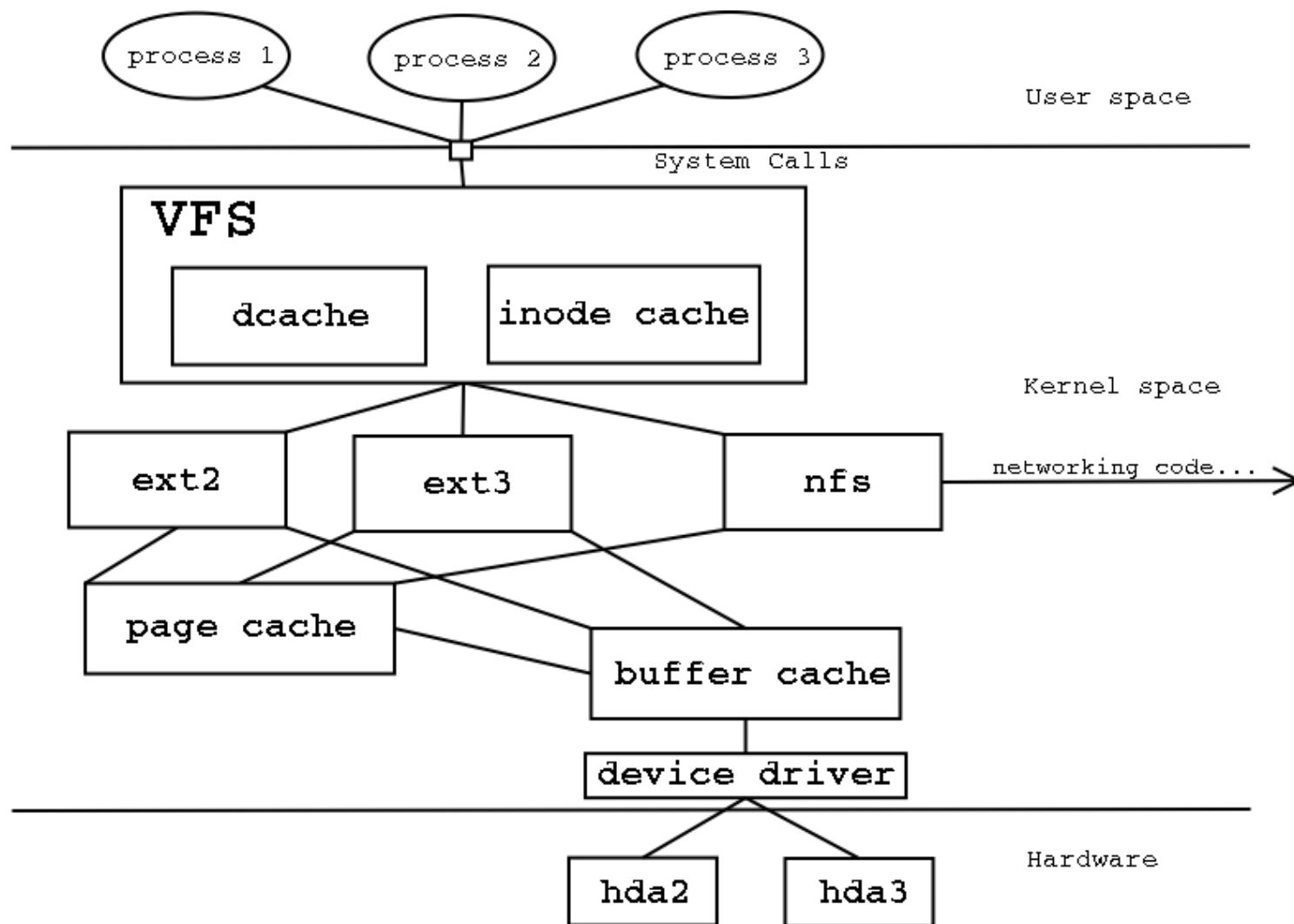
On réalise de la préallocation

relâchée à la fermeture effective...

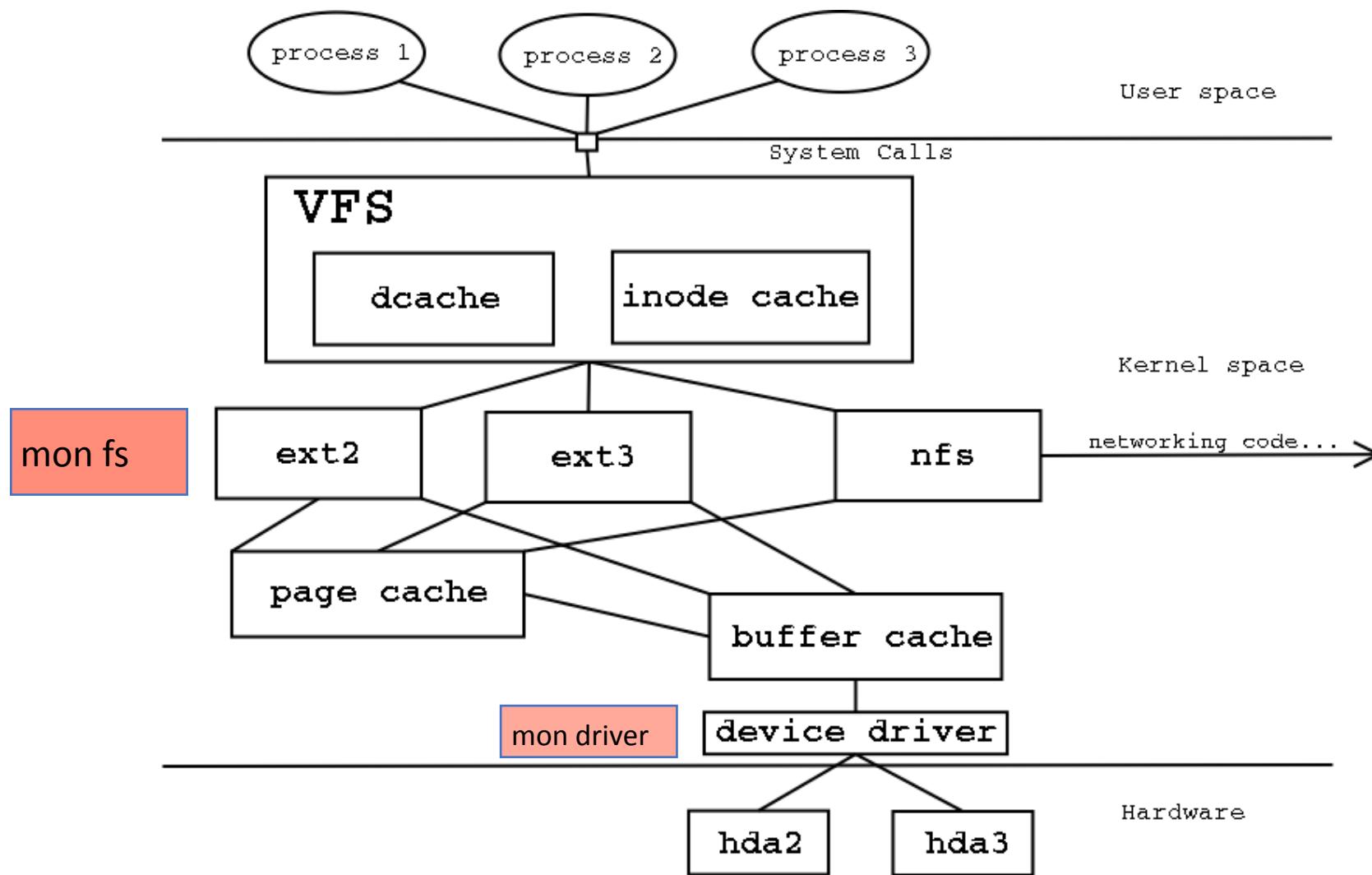
# Structure Générale



# Structure plus détaillée



# Structure plus détaillée



# VFS

La couche VFS Virtual File System (apparue avec SunOS 2.0) permet d'abstraire les particularités sous-jacentes  
le système de fichiers concret doivent simplement fournir des données exploitables pour VFS

inœuds

entrées de répertoires

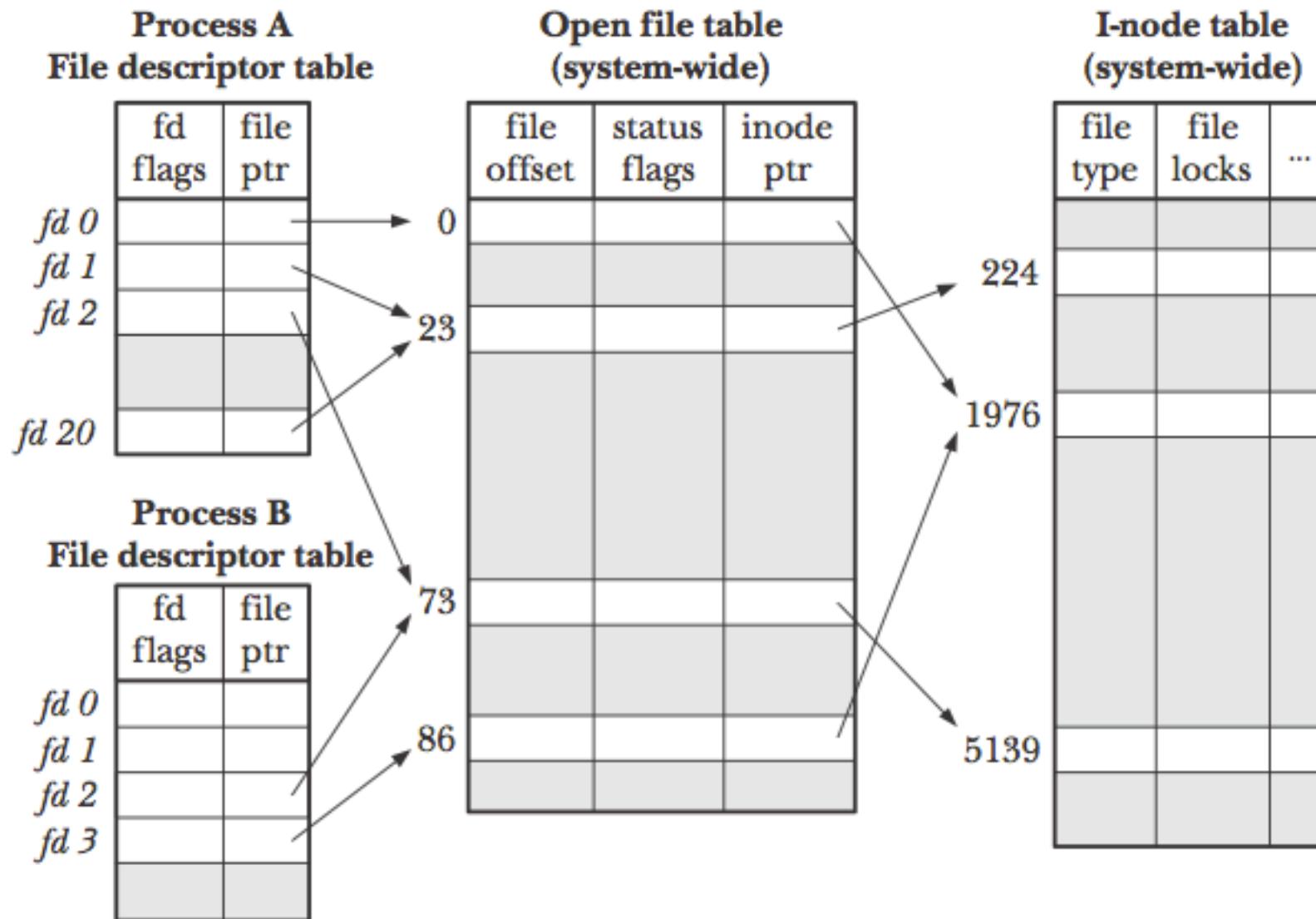
si certaines données ou services sont impossibles à fournir, on trouve des alternatives (valeurs par défaut, échec des appels)

C'est une vision objet, mais en C...

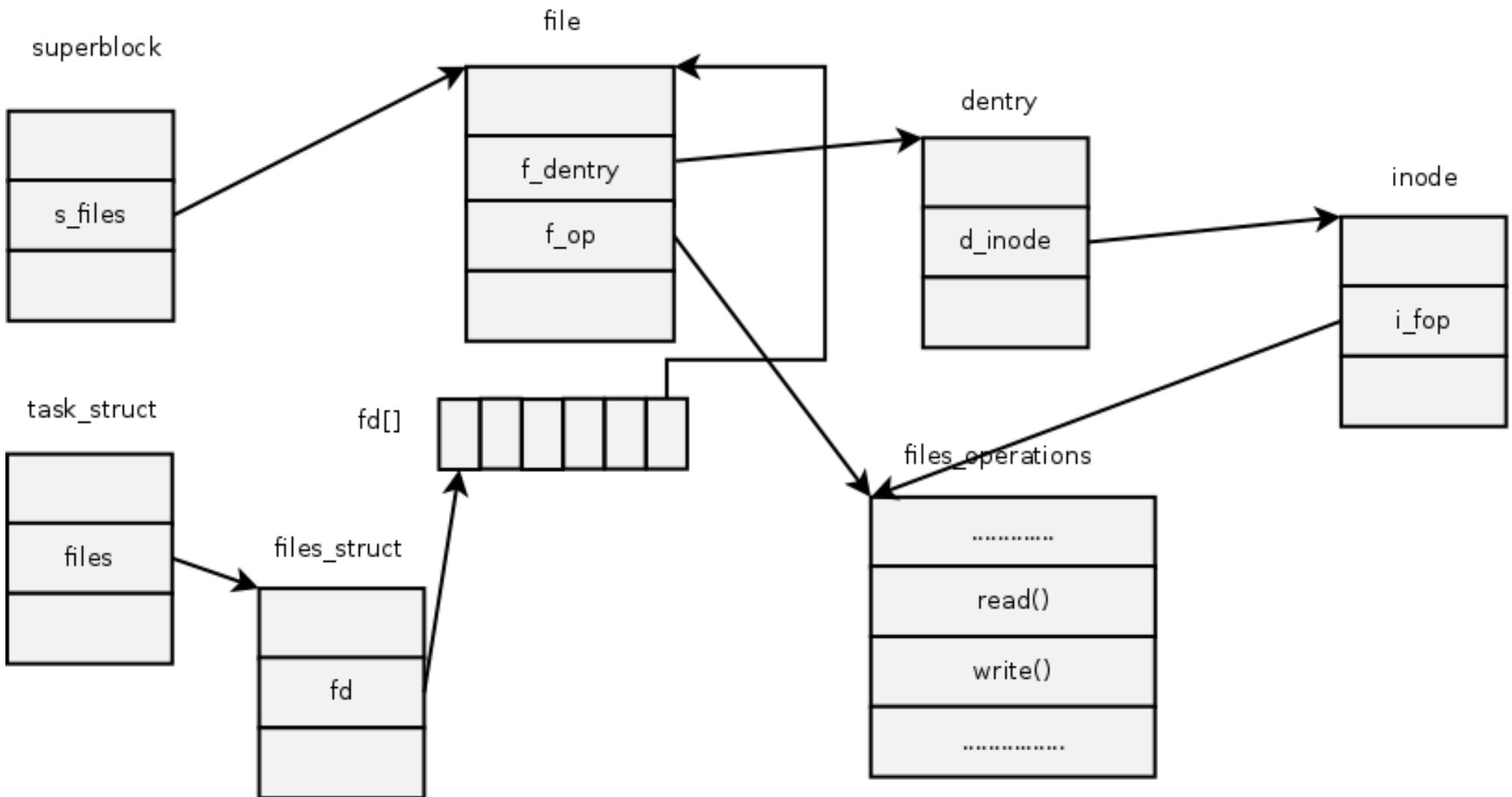
# Contexte fichier Processus

- Namespace (cf Linux, processus)
- Répertoire courant et racine
- Fichiers Ouverts (+ indicateur "close on exec")
- Limites (ulimit)
- Identité(s) utilisateur

# Fichiers ouverts



# Linux : fichiers ouverts



# APFS

APple File System

clonage (technique de copy on write)

chiffrement

fichiers creux

....

# NFS

Network File System (inventé par Sun)

Concurrents : AFS, RFS

Idée : délocaliser les fichiers de façon transparente



# SGF distribué

Propriétés attendues :

transparence de l'accès

transparence de la localisation

nommage indépendant de la localisation

tolérance aux pannes

mobilité des fichiers



# SGF distribué

Espace de nommage uniforme ?

si oui, transparence de la localisation

Serveur avec ou sans état ?

des requêtes SGD nécessitent de conserver un état, qui le fait :

client ? serveur plus simple

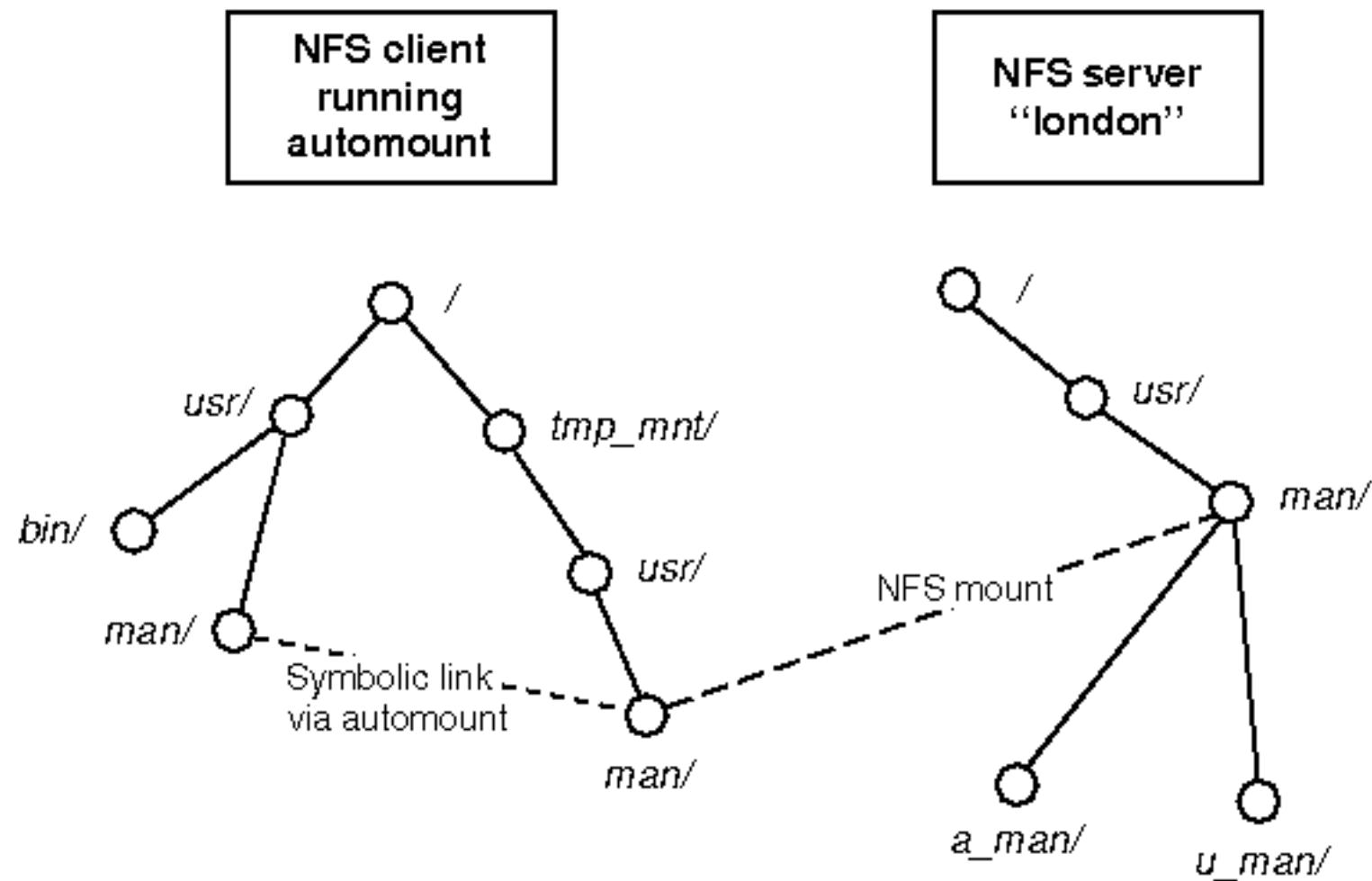
serveur ? plus complexe et en cas de plantage plus problématique...

Sémantique des écritures

à la Unix (tout est tout de suite visible ?) coûteux

« session » à la fermeture ou de temps en temps...

# NFS montage



# NFS conception

Non restreint à Unix

Protocole indépendant du matériel

Mécanisme simple de récupération après plantage

Accès transparent pour les utilisateurs

Sémantique Unix

Performances comparables à des disques locaux

Indépendance de la couche transport

# Containers

Docker

Virtualisation (para

Cloud (openstack

