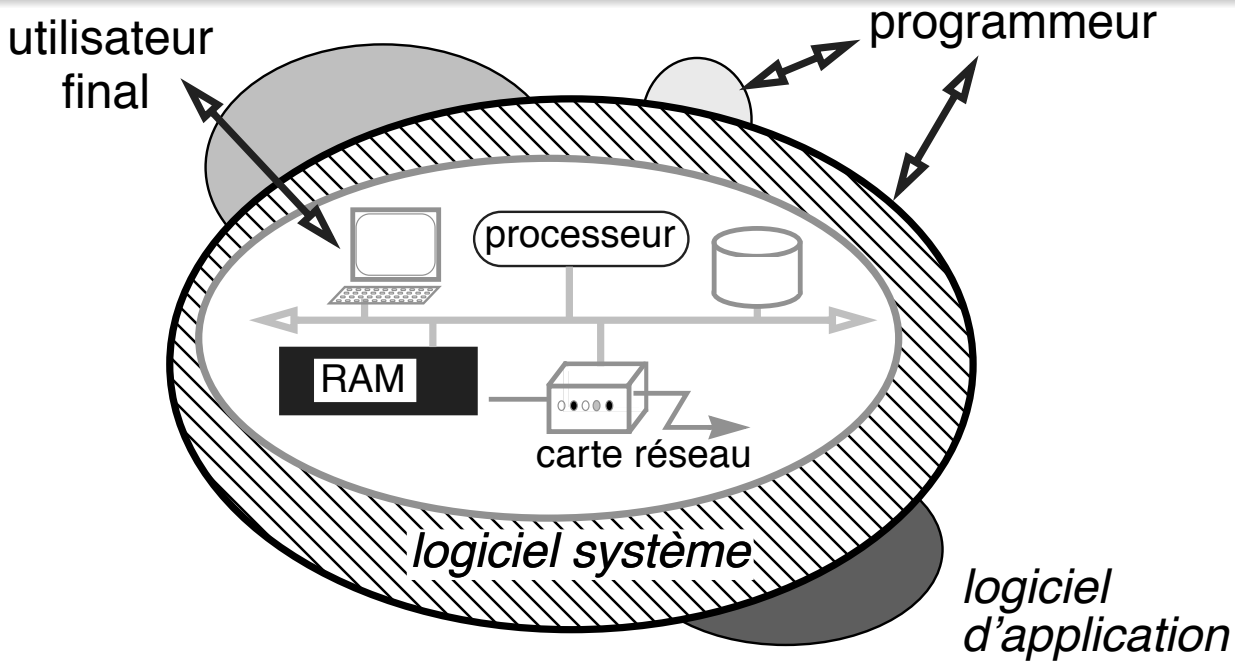
Deuxième partie

Mécanismes de mise en œuvre des SX

Le SX fournit une interface (abstraite) d'accès aux ressources matérielles pour un ensemble de traitements indépendants



$\Rightarrow$

- superviser l'exécution des applications en fonction de la disponibilité des ressources
- gérer le partage
- contrôler l'accès aux ressources
- gérer efficacement les ressources
  - parallélisation (découplage temporel)
  - localité (découplage spatial)

Contenu de cette partie

Mécanismes matériels, protocoles et schémas utilisés par le SX pour

- gérer des traitements concurrents
- protéger/contrôler l'accès aux ressources
  - processeur
  - mémoire
  - périphériques
- utiliser les ressources de manière plus efficace
  - parallélisation
  - localité

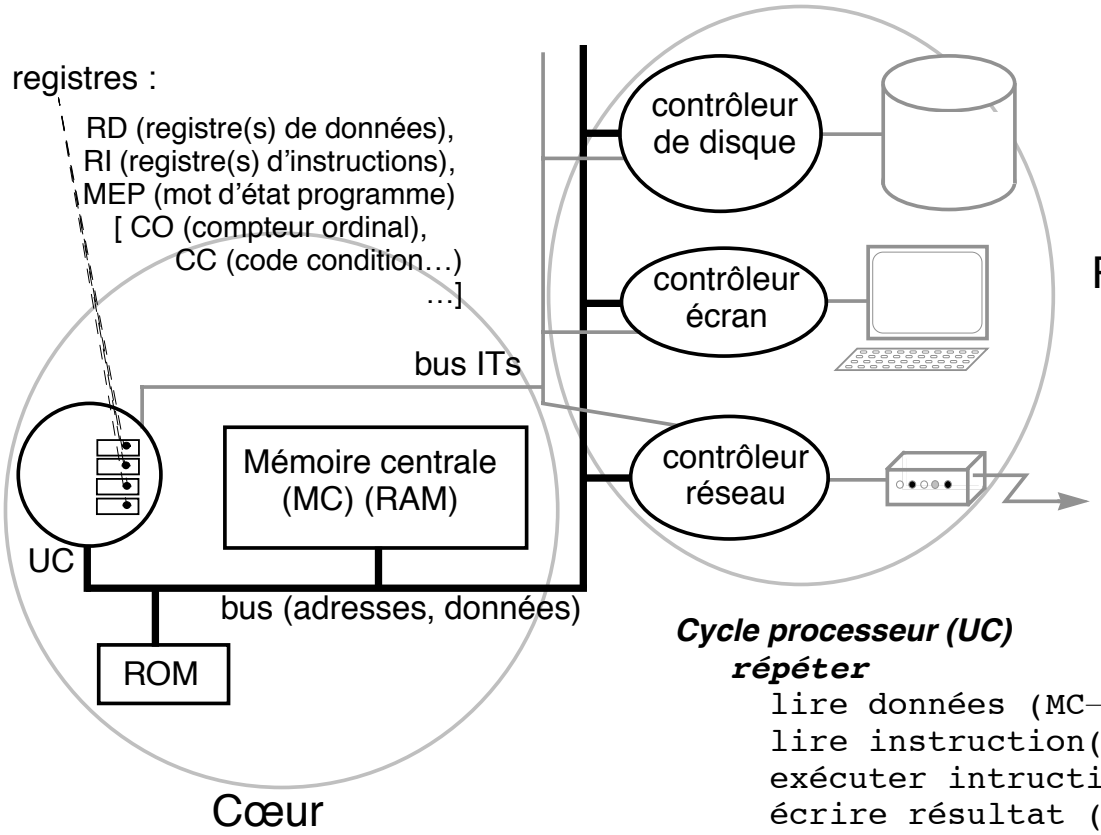
Annexe : mécanismes et services de transfert du contrôle

Contenu des parties suivantes

Algorithmes et politiques utilisés par le SX pour gérer les processus et les différentes ressources

Plan

Architecture d'un ordinateur : modèle (simple), terminologie



Cycle processeur (UC)

```
répéter
  lire données (MC→UC)
  lire instruction(MC→UC)
  exécuter intruction(données)
  écrire résultat (UC→MC)
sans fin
```

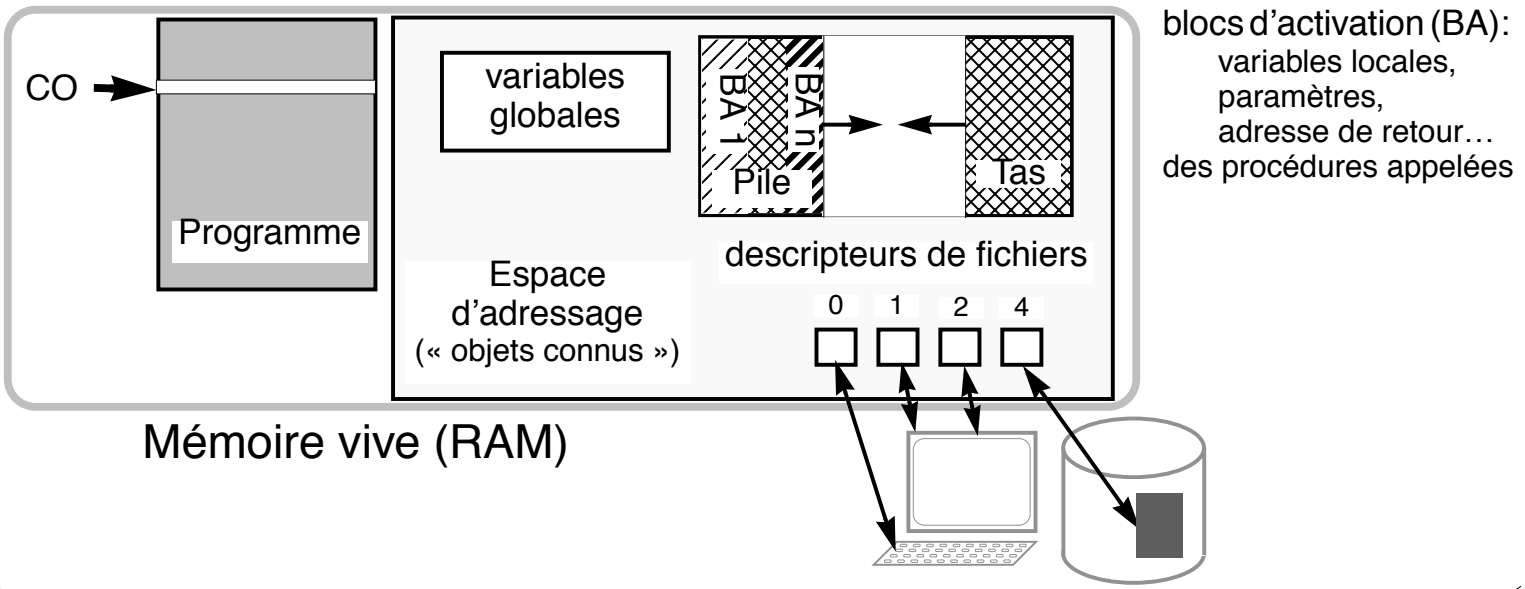
Plan

Exécution d'un programme : notion de processus

processus (*activité*) $\triangleq$ *exécution* d'un programme par un processeur

- Analogie :
  - *livre* $\sim$ programme (statique) ;
  - (activité de) *lecture* d'un livre $\sim$ processus (dynamique)
- Un programme peut être exécuté par plusieurs processus en même temps, chaque processus travaillant sur ses propres données
  Exemple : traitement de texte

Point de vue du programmeur



Plan

Mise en œuvre du partage de ressources entre activités concurrentes

Objectif

Répartir à tout instant les ressources entre les différents processus, de sorte à

- permettre l'exécution simultanée de plusieurs processus
- fournir à chaque processus les ressources nécessaires à son exécution

Structures de données

→ gérer l'état d'allocation des ressources aux différents processus

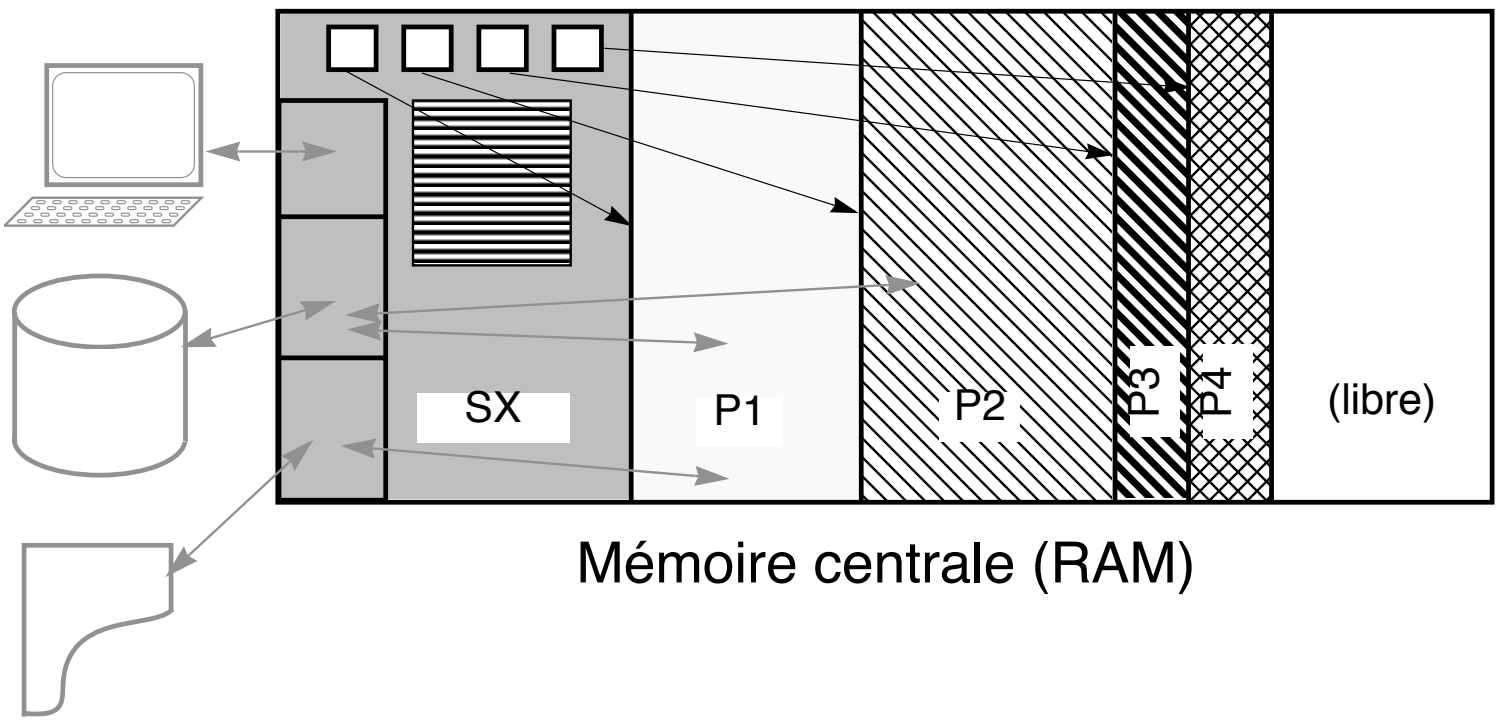
- par processus : ressources { attendues / obtenues } *(descripteurs de processus)*
- par ressource : processus utilisateurs (élus), processus en attente

Algorithmique

- Mécanismes matériels utiles à la réalisation du partage proprement dit
  - mémoire → partage physique
  - processeur → partage temporel
  - périphériques « autonomes » → gestion des interactions (échange des requêtes et des résultats)
- Politiques d'allocation : choix des processus élus, selon les besoins utilisateur (priorités, équité...), la nature du périphérique...
  → chapitres suivants

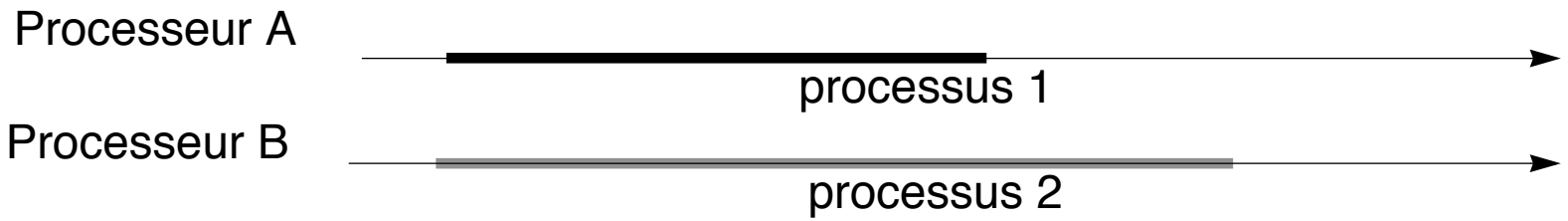
Mémoire centrale

- Partage physique
- Une partie de la mémoire espace système est attribuée au SX. Cet espace contient notamment les données utiles à la gestion
  - des processus (descripteurs de processus),
  - des ressources (requêtes en attente)
  - des échanges avec les périphériques (tampons d'E/S...)



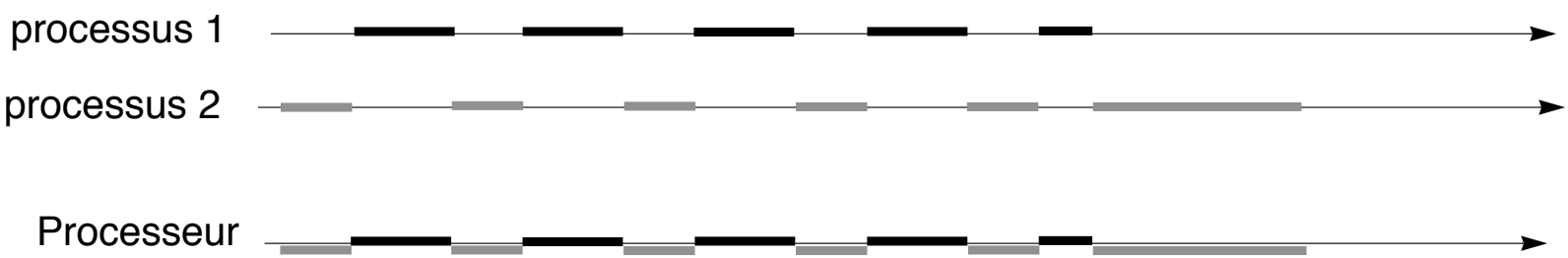
Processeur

Partage physique (ex : multicœur) : *vrai parallélisme*



Partage temporel (*pseudoparallélisme*) (cas le plus fréquent)

Le processeur est alloué à tour de rôle à chacun des processus.



$\Rightarrow$ mécanisme pour

- interrompre le processus en cours et sauvegarder son état
- restaurer l'état du processus suivant, puis reprendre son exécution

La notion de processus permet d'abstraire la gestion physique des processeurs

Mécanisme pour le partage temporel du processeur : commutation de contexte

Contexte $\triangleq$ ensemble des informations à sauvegarder
pour pouvoir poursuivre un traitement interrompu entre 2 instructions

Etat d'un processus (en général)
- valeur des données utilisées
- code exécuté
- état (contexte) du processeur

Données et instructions restent souvent inchangées
$\rightarrow$ il suffit de sauver l'*état du processeur* :
- registres généraux (données et instructions)
- mot d'état programme (MEP, ou PSW) :
  compteur ordinal (CO), code condition, masque d'IT, contexte mémoire accessible, mode (programme/superviseur)...

Commutation de contexte

sauvegarde du contexte courant
restauration du prochain contexte
} par une instruction indivisible

Interaction entre le SX et les périphériques

Problème

Informer le SX dès qu'un événement nouveau (fin de traitement, incident...) se produit sur l'un des périphériques.

Solutions

- Attente active (scrutation) :
  le SX teste en permanence l'état des différents périphériques
  - → simple et fiable
  - → coûteux et inadapté aux applications temps-réel
    - → E/S sur systèmes simples/anciens
- Interruptions (mécanisme matériel)
  - le SX n'attend pas, mais poursuit son activité
  - un périphérique signale au SX le moment où l'événement survient
  - à la réception du signal, le SX interrompt le traitement en cours, pour traiter l'événement signalé
  - l'événement traité, le SX reprend le traitement interrompu
  - → réponse « rapide » au « signal » dePi
  - → plus complexe ; requiert un support matériel (ITs + commutation)

Prise en compte des interruptions : cycle de l'UC avec ITs

RI : registre instruction
RD : registres données
as: adresse sauvegarde CO
ai : adresse 1ère instruction à exécuter sur interruption
IP : booléen vrai ssi interruption présente
IA : booléen vrai ssi traitement interruption autorisée

```
répéter
    RI := Mem[CO];
    RD := charger(Mem[CO]);
    CO :=CO + 1;
    exécuter (RI);
     si (IP et IA) alors
     début -- commuter(traitement en cours,traitant IT)
          IP := IA := faux;
          Mem[as] :=CO;
          CO := ai;
      fin si;
fin répéter;
```

```
ai :  < traiter l'interruption >;
       -- commuter(traitant IT, traitement interrompu)
       IA:=vrai;
       CO:=Mem[as]
```

Les ITs permettent au SX de réagir aux périphériques à tout moment

Exemples

- calcul et E/S simultanés (systèmes de 3ème génération)
- intervention externe : utilisateur, chaînes de mesures
- délai de garde

Terminologie

- Niveaux d'interruption : causes d'interruption possibles, identifiées
  - Chaque niveau peut être traité séparément, par une routine (procédure) de traitement (traitant, handler)
  - Les adresses de ces routines sont souvent conservées dans une zone fixée de l'espace système : vecteur d'interuption
  - Priorités possibles entre niveaux
- Masque d'interruption : ensemble des niveaux pouvant provoquer une IT
- Niveau désarmé (vs armé) $\rightarrow$ l'IT est ignorée
- Niveau masqué (vs démasqué) $\rightarrow$ le traitement de l'IT est retardé

Que se passe t-il lorsqu'une IT est reçue
alors qu'une autre IT est déjà en cours de traitement ?

Selon le matériel, ou le niveau, la nouvelle IT peut

- être ignorée (IT désarmée)
- provoquer l'abandon du traitement d'IT en cours (IT perdue)
- être traitée selon une politique de niveaux de priorité
  - les niveaux inférieurs ou égaux à l'IT en cours sont masqués
  - l'arrivée d'une IT de niveau supérieur interrompt le traitement en cours qui sera repris ensuite : traitements en cascade (empilés)

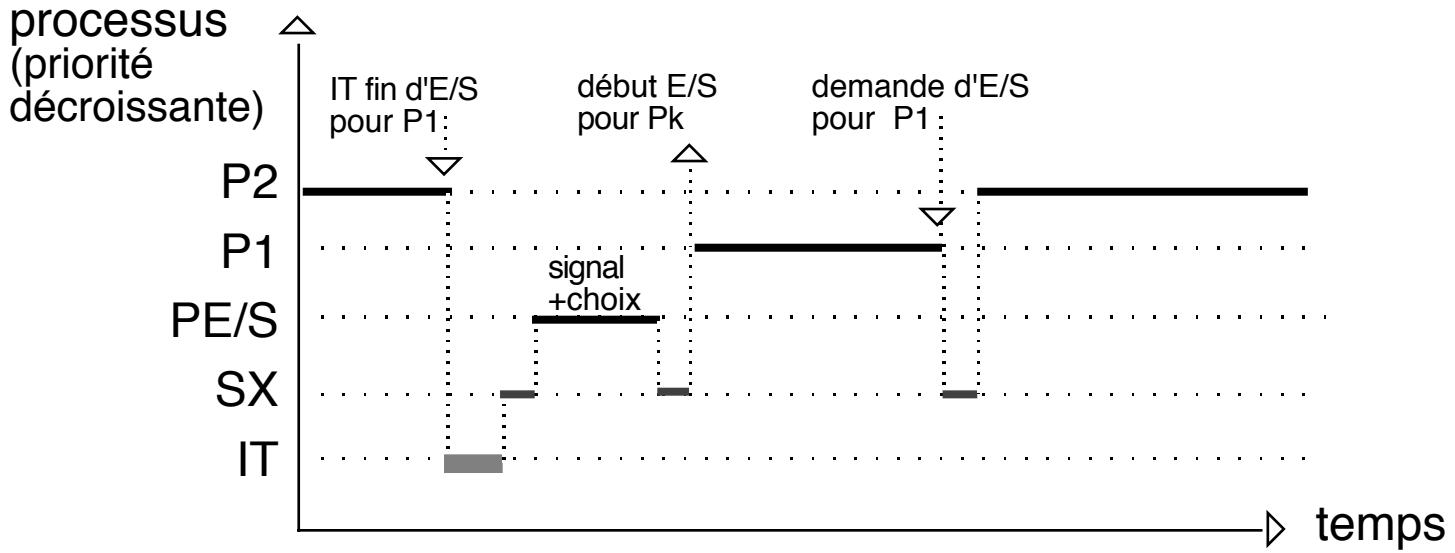
Plan

Contrôle du processeur : interruptions

Problème

Le SX doit pouvoir reprendre le processeur aux traitements en cours

- utilisation des interruptions d'E/S (2e génération)



- horloge (3e génération)
  périphérique générant des IT à des instants programmables
  → le SX peut reprendre la main à intervalles réguliers

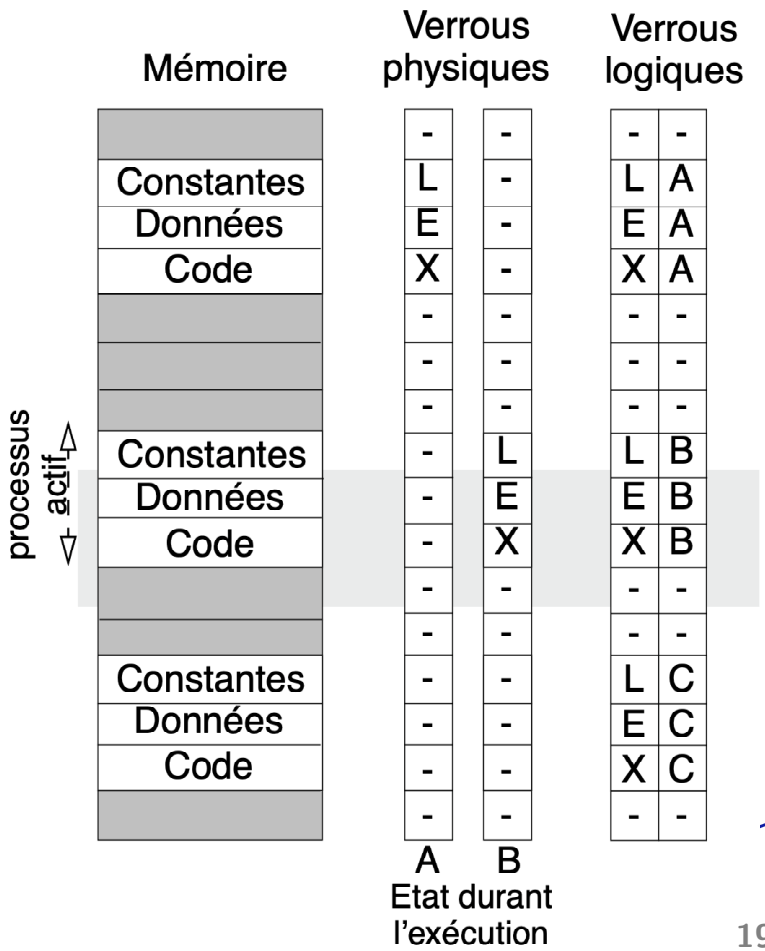
Protection de la mémoire (1/2)

Problème

fixer les opérations permises (lire, écrire...) { pour chaque traitement / sur chaque mot mémoire

Manière directe (données)

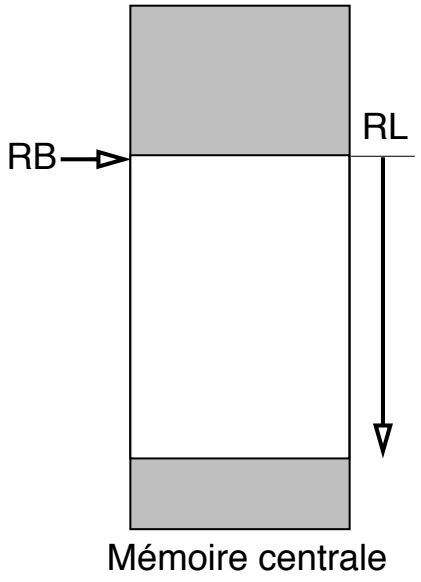
- la mémoire est divisée en blocs
- à chaque bloc sont associés
  - un verrou physique fixant (et permettant de contrôler) les droits d'accès pour l'exécution en cours :
    - accès interdit (-),
    - accès en exécution (X),
    - accès en lecture (L),
    - accès en lecture et écriture (E)
  - un verrou logique, définissant un propriétaire pour le bloc, et ses droits d'accès

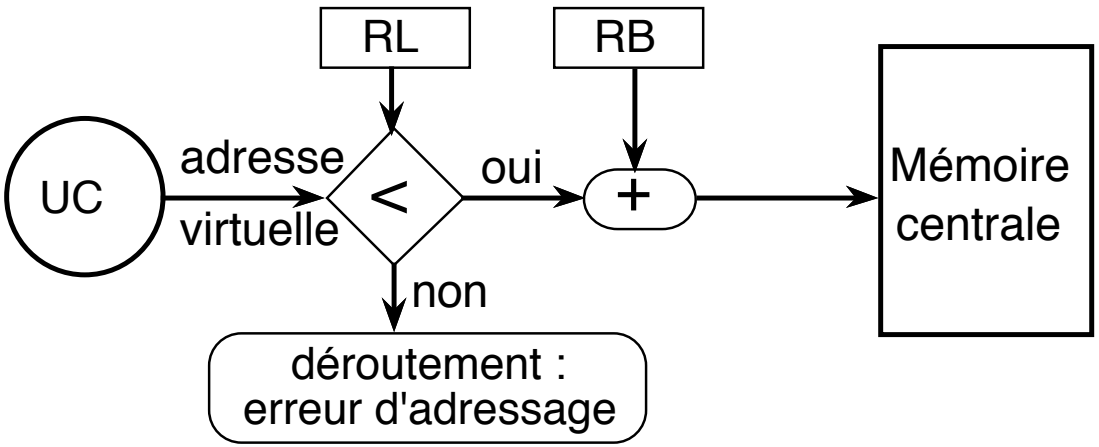


Protection de la mémoire (2/2)

Manière détournée (encapsulation) : via le mécanisme d'adressage

- les programmes sont implantés dans des zones contiguës et utilisent des adresses relatives (relogeables) commençant à 0
- utilisation de 2 registres

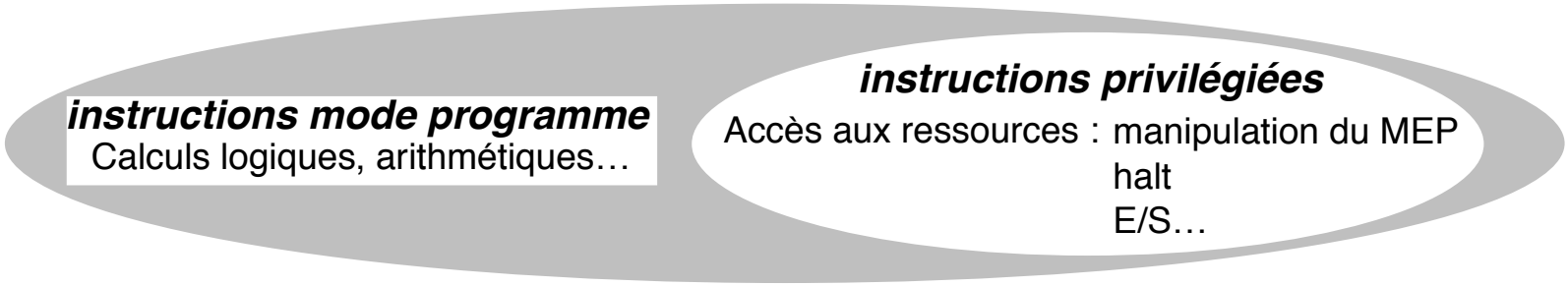




- base (RB) : registre de translation (utilisé par le SX)
  → évaluation de l'adresse réelle
- limite (RL) : taille de l'espace du processus lors de l'accès

Encapsulation des ressources (1/2)

Mode superviseur et instructions privilégiées



Protocole d'accès aux ressources contrôlées par le SX

- accès aux ressources obligatoirement en mode superviseur
- changement de mode : programmme → superviseur
  possible seulement via une instruction d'appel superviseur :
  `trap(n)` (ou `SVC(n)...`)
  - commutation de contexte
    (avec la routine système identifiée par le paramètre (n))
    → passage en mode superviseur
  - exécution de la routine système, qui se termine par une
  - restauration (commutation) du contexte de l'appelant
    → retour au mode utilisateur

→ l'accès aux ressources passe nécessairement par les procédures système

Encapsulation des ressources (2/2)

Installer le SX dès le démarrage : amorce

Un (petit) programme permet d'installer en mémoire le SX, avant tout autre programme : chargeur initial (ou amorce, ou bootstrap).

- IBM 360 : une seule instruction (IPL (E/S)) + pupitre
- sur minis vers 1965 : saisir un petit programme aux clés
- sur ordinateurs actuels : programme en ROM

Chargement téléscopique

Le chargeur initial peut en charger un autre plus complet, etc...

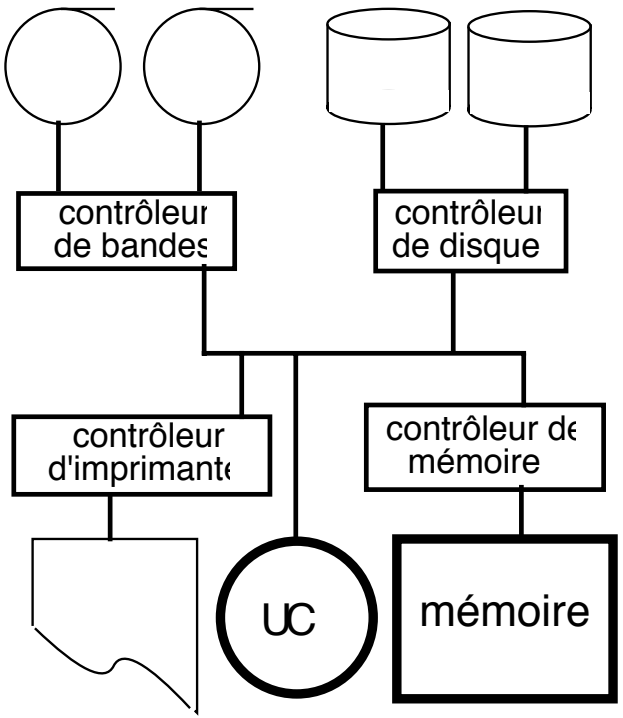
$$\text{ROM} \rightarrow \text{bootblock} \rightarrow \text{fichier boot} \rightarrow \text{UNIX}$$

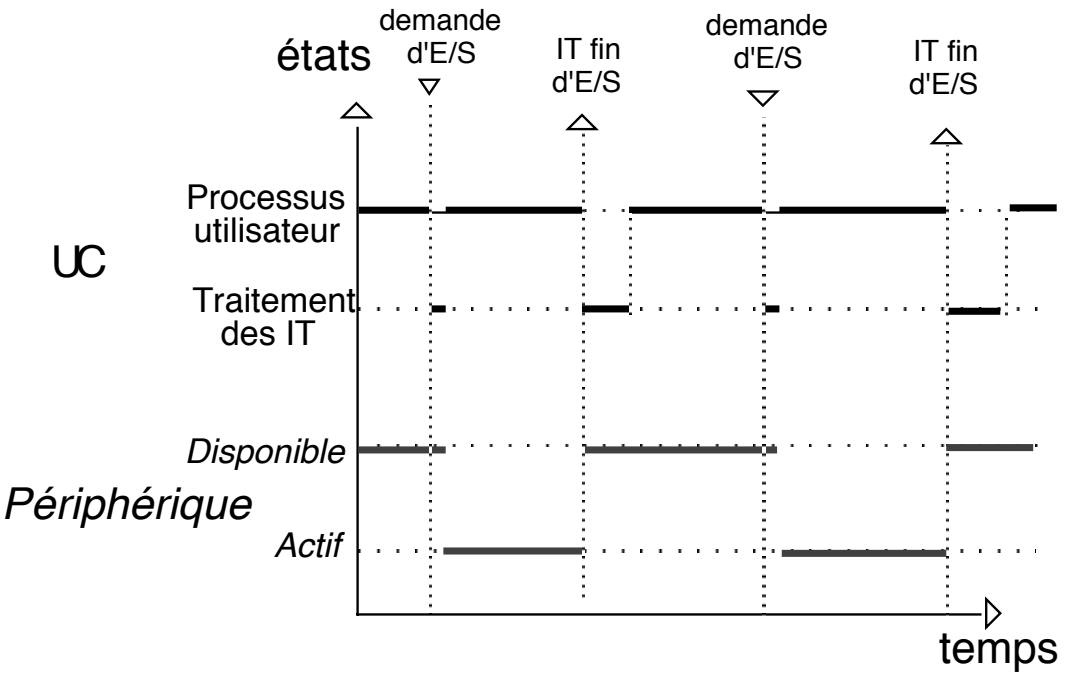
- ROM : tests matériels, puis chargement du bloc d'amorce (bootblock) (ex : MBR)
- bloc d'amorce ($\approx$ 1Ko) : programme (chargement) + table (adresse fichier d'amorce)
- fichier d'amorce ($\approx$ 100 Ko) (exemples (Linux) : LILO, GRUB) : choix et chargement du noyau
- les tables sont modifiables (instruction *installboot*)

Plan

Parallélisation de l'utilisation des ressources : délégation des E/S

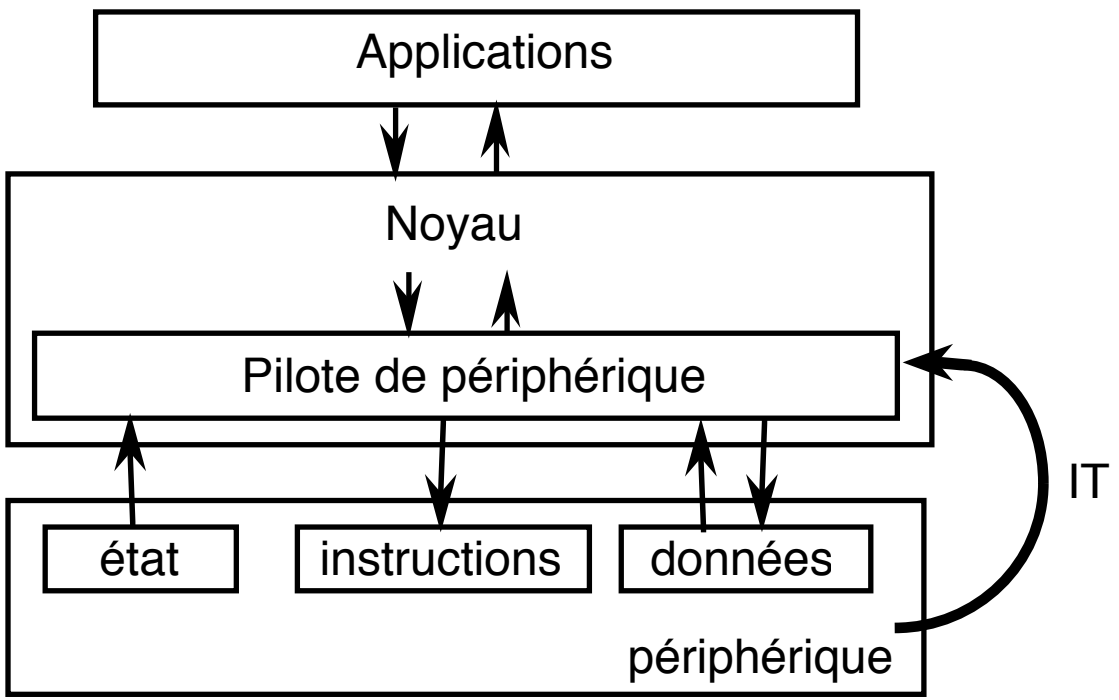
Moyen : périphériques autonomes + IT





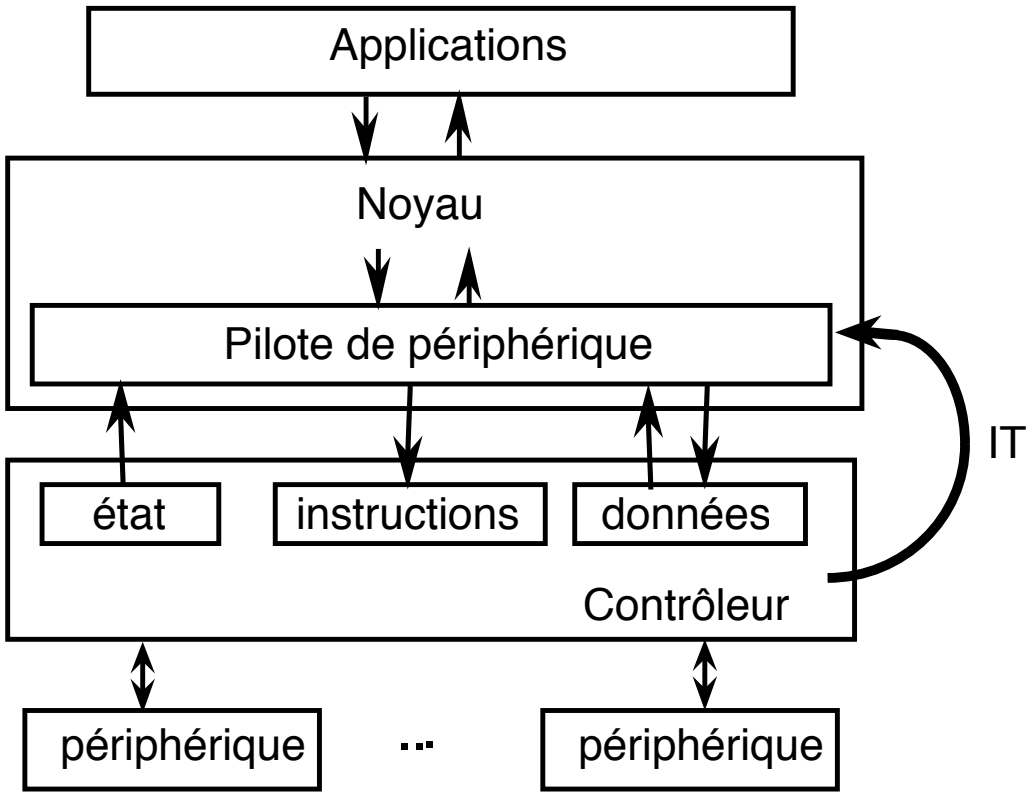
Délégation des E/S : principe

- les interactions avec chaque périphérique sont gérées par un composant du SX (pilote), spécifique à chaque périphérique
- l'interface d'un périphérique est un ensemble de registres accessibles
  - par un jeu d'instructions spécifique (peu flexible/portable)
  - ou comme des mots mémoires «ordinaires» (Memory mapped I/O)



Réduire le nombre des IT

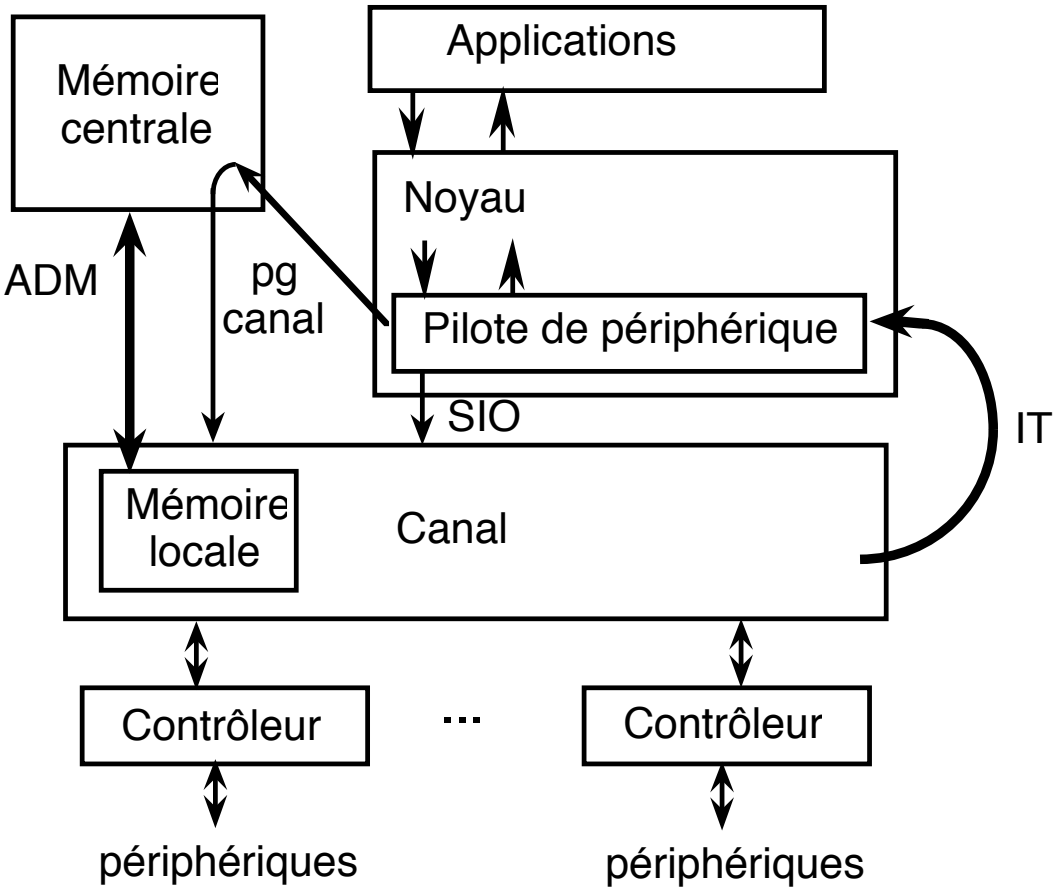
→ ajout de contrôleurs intermédiaires chargés de « regrouper » les IT



Par rapport au schéma précédent, les contrôleurs gèrent :

- des opérations plus complexes
- vérification des données
- plusieurs périphériques similaires (mais un seul transfert à la fois)

Réduire encore le nb d'IT : périphériques à ADM et processeurs canaux



- Le SX prépare et place en mémoire centrale le programme canal, puis lance le canal
- Accès direct à la mémoire centrale (ADM) :
  les commandes canal échangent directement les données par blocs avec la mémoire centrale
- Le canal signale par une IT la fin de l'exécution de son programme

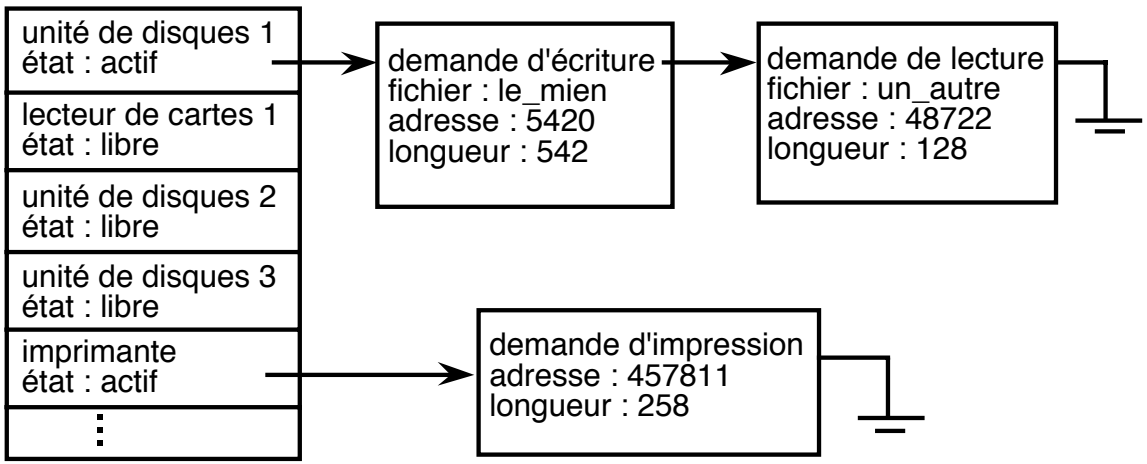
Canal : processeur pouvant exécuter des séquences d'E/S

Découplage des activités (1/2)

But : éviter que le SX ou le périphérique restent en attente d'un résultat ou d'une nouvelle requête

Découpler les requêtes

→ associer à chaque périphérique une file de demandes en attente

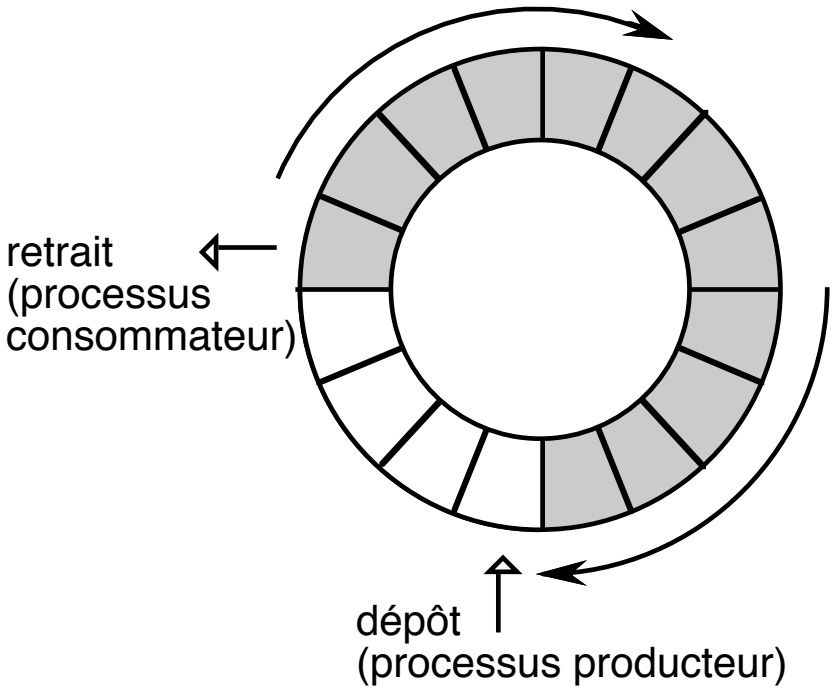


- nouvelle demande d'E/S → la demande est déposée dans la file (si le périphérique est déjà actif), le demandeur est mis en attente, et un nouveau processus actif est choisi.
- IT de fin d'E/S → le demandeur est réveillé, et une nouvelle E/S est choisie dans la file, puis lancée

Découplage des activités (2/2)

Découpler l'accès aux résultats/données

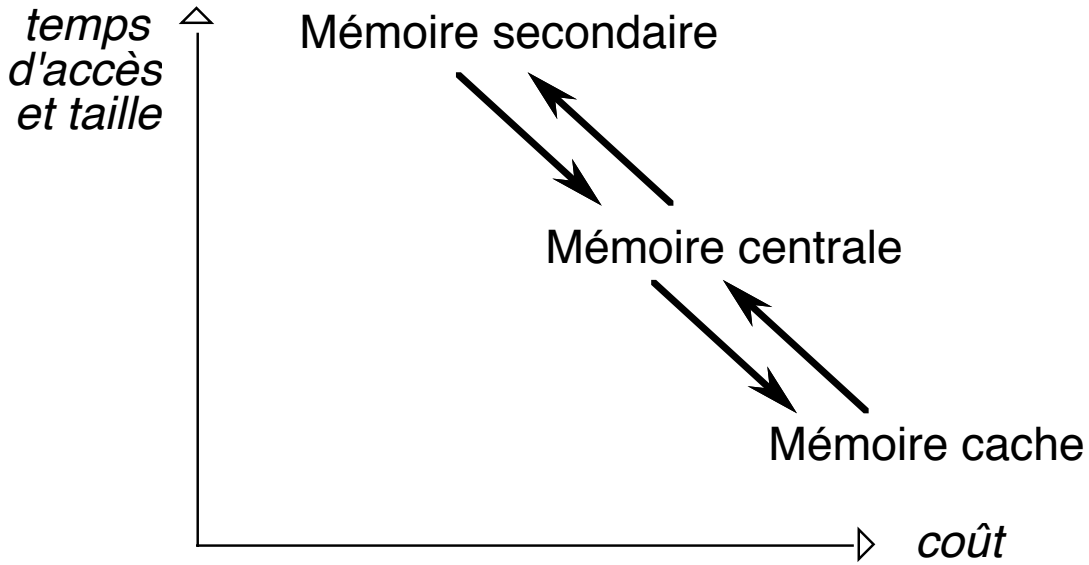
Même schéma (interaction producteurs-consommateurs) que pour les requêtes : les données en attente de traitement sont conservées dans des tampons FIFO



→ variations de vitesse (temporaires) entre « processeurs » absorbées : si leurs vitesses moyennes sont égales, le producteur et le consommateur pourront progresser indépendamment.

Découplage des données : hiérarchie de mémoires

Niveaux de mémoire



Idée : un processus n'utilise que peu de données à la fois
$\rightarrow$ ranger les données prochainement utilisées en mémoire rapide

Heuristique : principe de localité

ce qui a été accédé récemment le sera prochainement
(le passé récent est une bonne image du futur proche)

Mécanismes matériels de transfert du contrôle

Transfert de contrôle $\triangleq$

- arrêt du traitement en cours C
- commutation de contexte entre C et un autre traitement A
- exécution/reprise de A

Variante structurée éventuelle (et fréquente) :
A se termine par un transfert de contrôle vers C $\rightarrow$ pile d'exécution

Transfert asynchrone indépendant du flot de contrôle défini par le programmeur (programmation événementielle)

- *Interruptions* (matérielles)

Transfert synchrone intégré (attente) au flot de contrôle du traitement

- *Appel superviseur*
- *Déroutement* : commutation (et traitement) causés par l'exécution du processus en cours (division par zéro, protection mémoire...)

Services logiciels de transfert du contrôle

- Commutation de contexte
  - API système : sauvegarde/restauration de contexte (exemple UNIX : setjmp, longjmp)
  - structures/bibliothèques des langages de programmation : coroutines (Modula 2), méthodes (périmées !) suspend/resume de la classe Java Thread
- Transfert asynchrone (interaction « asynchrone »)
  - API système : « interruptions » logicielles (ex : signaux UNIX)
  - langages de programmation : support à la programmation par événements (schéma publier/s'abonner) (Java : Swing, Beans)
- Transfert synchrone
  - appels systèmes, appel procédural
  - exceptions (Ada, Java, Modula 3 . . .)