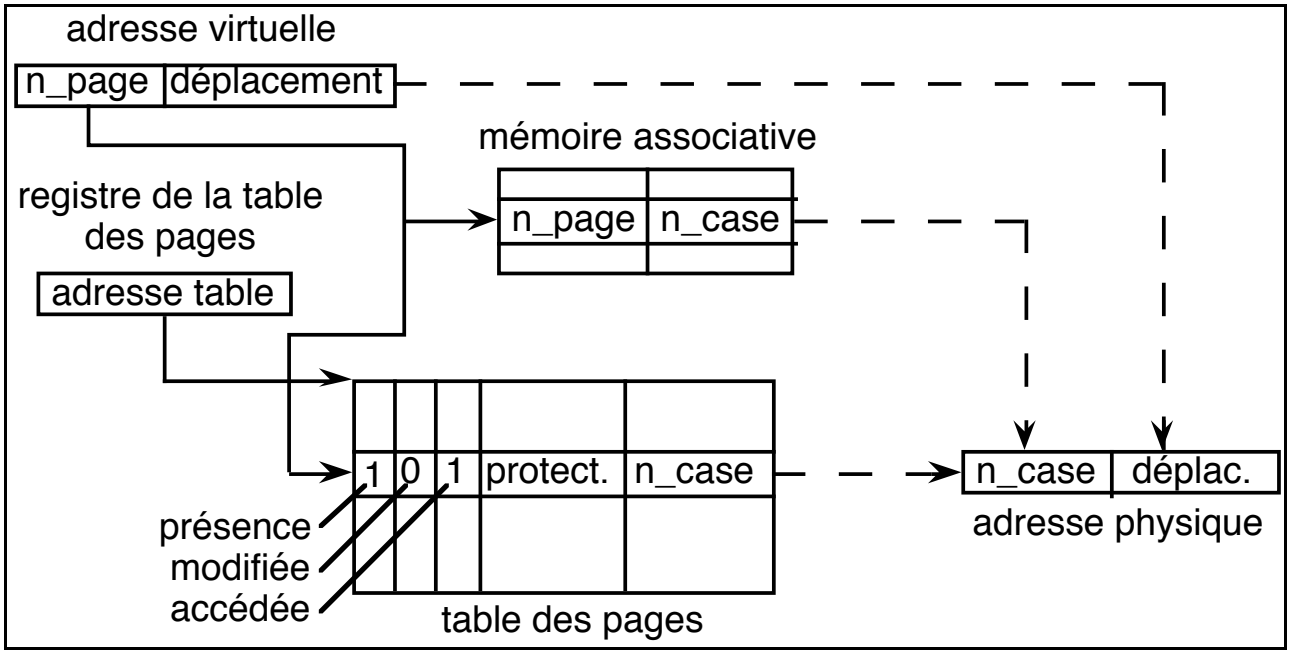


Fonctionnement de la pagination

ACCÈS MÉMOIRE AVEC TRAITEMENT DU DÉFAUT DE PAGE

Lors de l'exécution de chaque instruction du processus élu, Py, tout accès mémoire a la forme:
adresse effective virtuelle = page + déplacement = PVi + Di

- **si la page n'appartient pas à l'espace d'adressage du processus => erreur programme (exception, ou abort)**
- **si la page réside en mémoire centrale (est mappée avec une case Cj) => :**
  conversion d'adresse et envoi sur le bus d'adresse de l'adresse mémoire centrale :
  adresse physique = case + déplacement = Cj + Di
- **sinon**

appel système pour traiter le défaut de page (on passe en mode privilégié et dans le noyau du système)

a) sauvegarde du contexte du processus élu Py. Py devient bloqué;

b1) recherche d'une case libre, ou libération d'une case occupée, soit Ck la case obtenue ;

b2) si la case Ck contient une page qui a été modifiée (case associée à une page PVx) alors
demander son vidage sur disque (son contenu est différent du contenu sur disque) ; finsi ;
demander le chargement dans Ck de l'image disque de PVi

c) en attendant la fin de ces entrées-sorties, le processeur est alloué à un autre processus

b3) à l'arrivée de l'image disque dans Ck, mettre à jour la table des pages

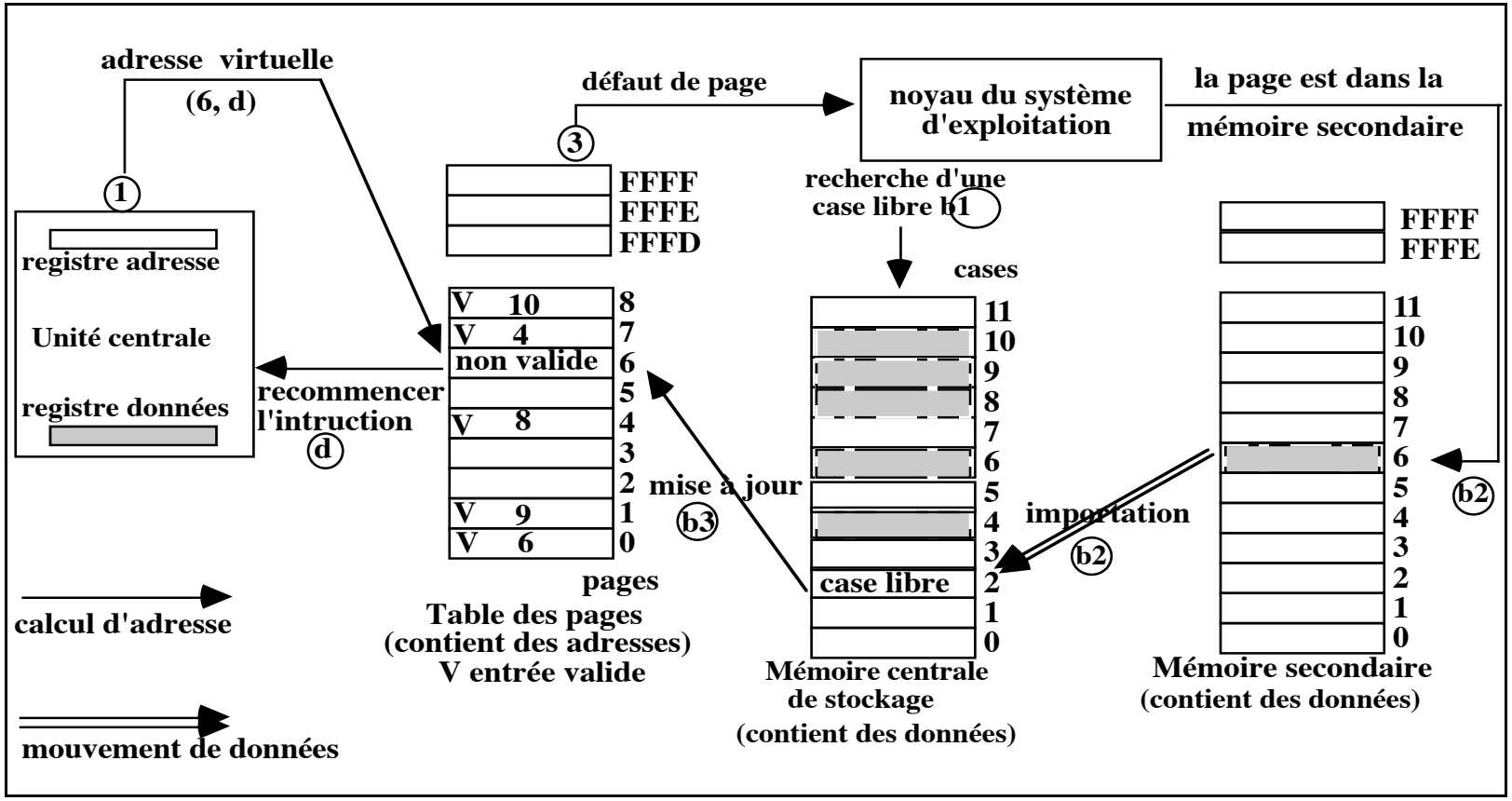
b4) mettre Py à l'état prêt et dans la file d'attente de processeur

c) quand Py est à nouveau élu, recharger son contexte

d) redémarrer l'instruction interrompue par le défaut de page
adresse effective virtuelle = page + déplacement = PVi + Di
devrait être transformée en adresse physique = = Ck + Di
si la case Ck n'a pas, entre temps, été reprise pour un autre processus ;

Si une instruction contient n adresses, il peut y avoir n + 1 défauts de page (dont 1 pour l'instruction)

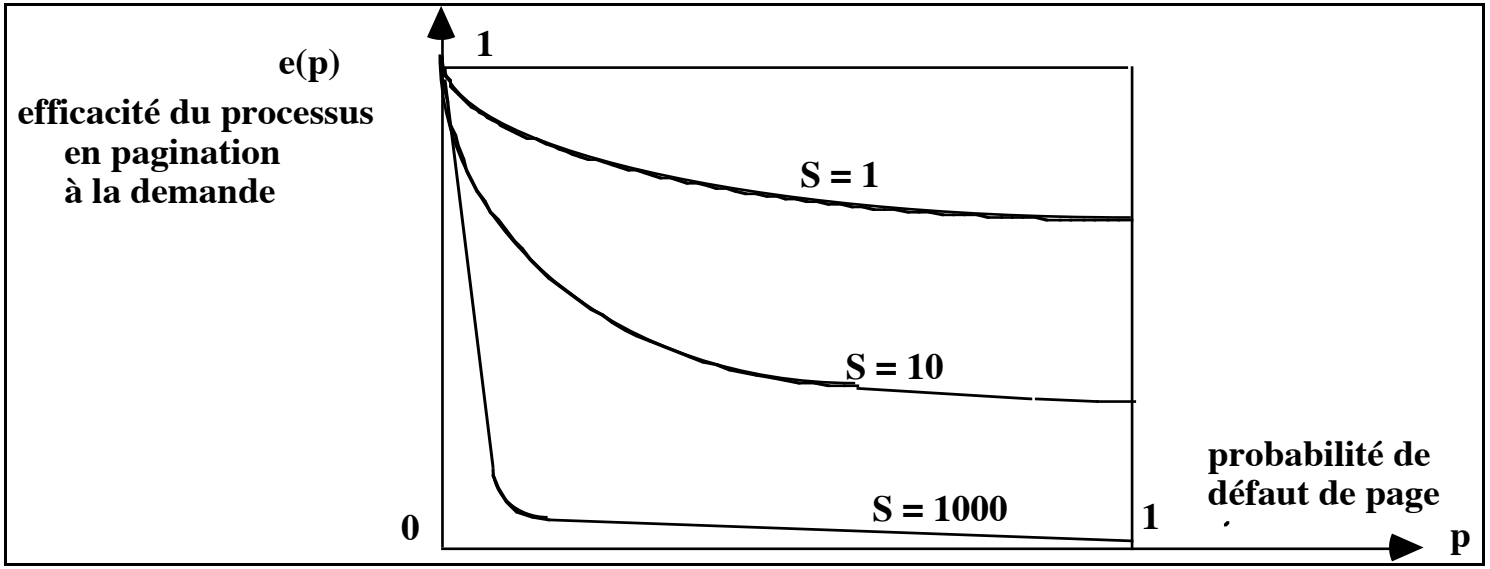
TRAITEMENT DU DÉFAUT DE PAGE



FRÉQUENCE ET COÛT DES DÉFAUTS DE PAGES AVEC LA PAGINATION À LA DEMANDE

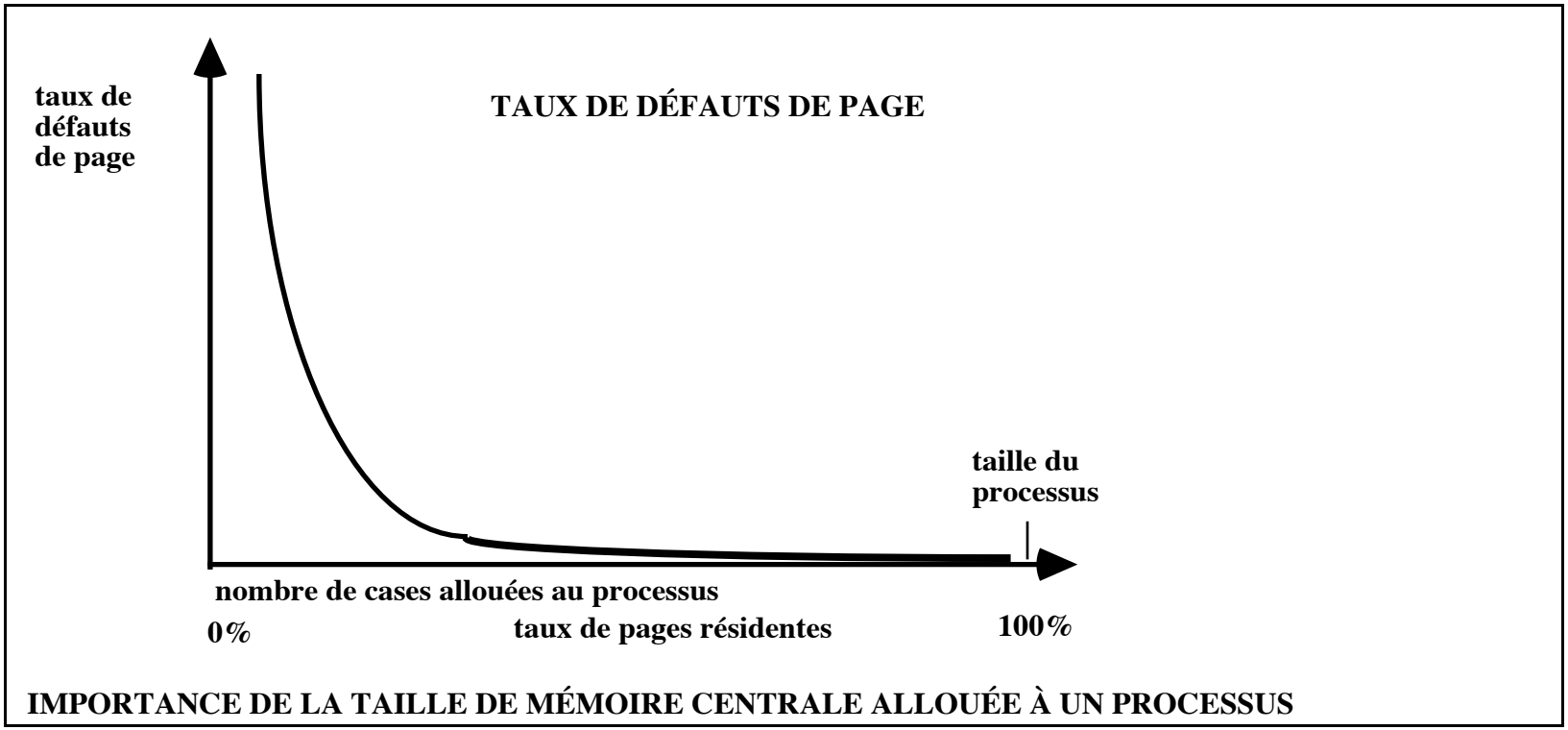
temps d'accès à la mémoire centrale
si page résidente alors t -- de l'ordre de 10 à 200 ns
sinon t + T avec T = 2* t1 + t2 ;
t1 : temps de commutation de processus (50 à 100 instructions, soit 50 à 100 microsecondes)
t2 : temps d'accès disque : déplacement de bras + délai rotationnel + transfert (15 + 8 + 1 = 24 ms)
Soit probabilité p de défaut de page
temps d'accès effectif = p*(T + t) +(1-p)*t = p*T + t
-- par exemple p*25 000 000+ 100 = 25 100 ns quand p = 10^{-3}

Efficacité du processus e (p) = (temps d'accès à la mémoire centrale) / (temps d'accès effectif)
e(p) = 1 / (1 +p*S) avec S = T/ t



L'efficacité dépend de la probabilité de défaut de page et du temps d'accès à la mémoire de va et vient

COÛT DE LA PAGINATION À LA DEMANDE



IMPORTANCE DE LA TAILLE DE MÉMOIRE CENTRALE ALLOUÉE À UN PROCESSUS

REMPLACEMENT DE PAGE

QUAND CHARGER UNE PAGE
A l'élection du processus : • tout le programme pour éviter les défauts de pages
• seulement les pages d'utilisation les plus probables : préchargement
A la demande

COMMENT TROUVER UNE CASE LIBRE
liste de cases libres (écrites ou non) pour remplacement de cases utiles à d'autres moments

CHOIX DE LA PAGE À REMPLACER
remplacement local : dans les pages du processus demandeur
=> nombre fixe de cases par processus , donc nombre fixe de processus
remplacement global : dans l'ensemble des pages résidentes non verrouillées => allocation dynamique
remplacement mixte : lutter contre l'écroulement en limitant le taux de défauts de page

ALGORITHMES DE REMPLACEMENT : choix de la victime

OPTIMAL : remplacer la page dont la prochaine référence est la plus loin dans le futur (ou jamais)
inutilisable, sauf en simulation
"RANDOM" : au hasard
"FIFO" : première entrée, première sortie (“First In First Out”)
"LRU" : la moins récemment utilisée (“Least Recently Used”)
“LFU” : la moins souvent utilisée (“Least frequently Used”)
“Seconde chance” ou “Horloge”
“seconde chance améliorée”

EXEMPLE DE REMPLACEMENTS DE PAGES POUR UNE CHAÎNE DE 18 RÉFÉRENCES

0 2 4 6 8 6 4 2 0 1 3 2 4 7 6 3 2 0

MÉMOIRE DE 5 CASES

FIFO : est victime la page résidente la plus ancienne : 12 défauts de page (5 + 7)

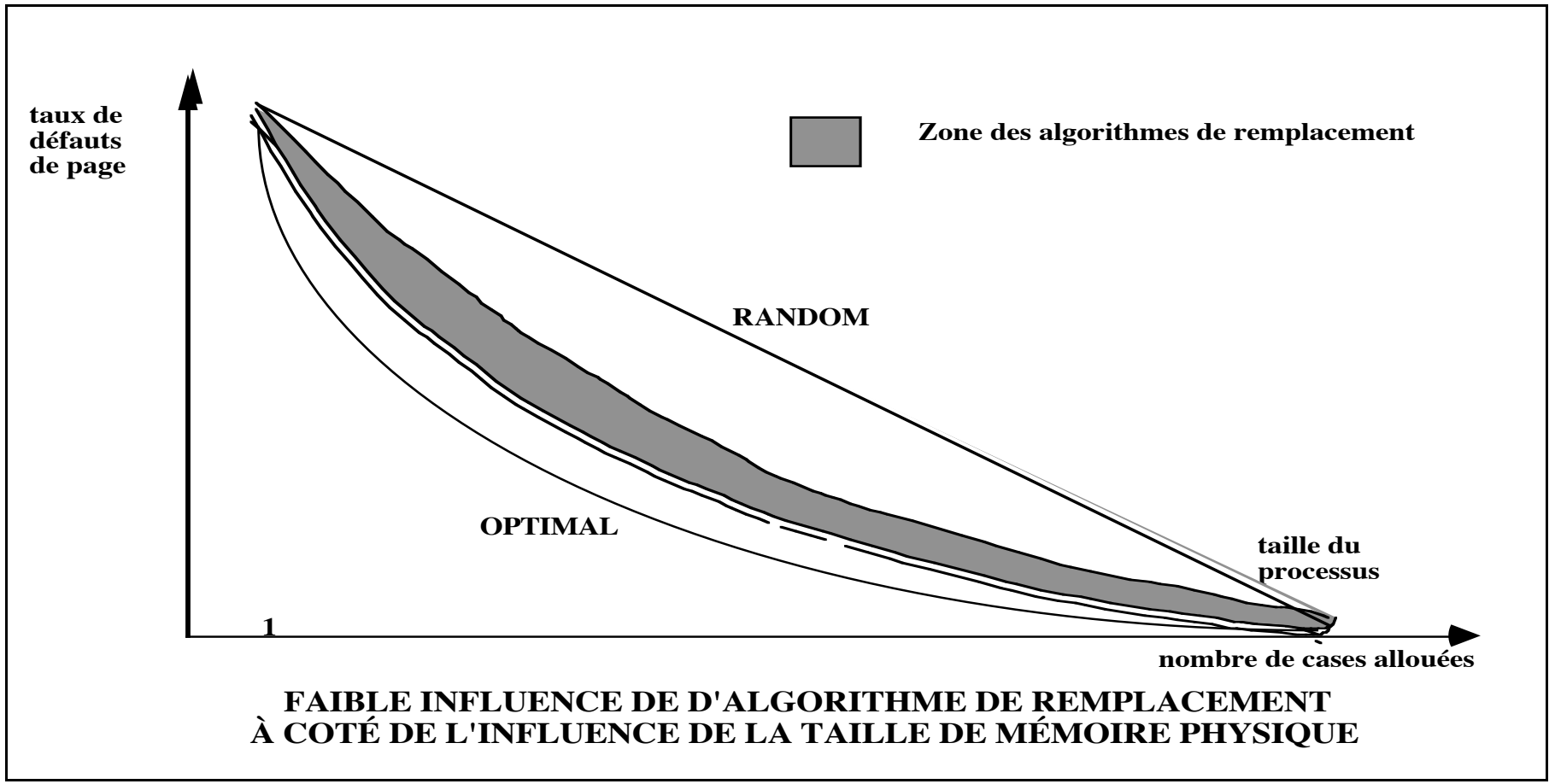
0	**2**	**4**	**6**	**8**	8	8	8	8	**1**	**3**	**2**	**4**	**7**	**6**	6	6	**0**
	0	2	4	6	**6**	6	6	6	8	1	3	2	4	7	7	7	6
		0	2	4	4	**4**	4	4	6	8	1	3	2	4	4	4	7
			0	2	2	2	**2**	2	4	6	8	1	3	2	2	**2**	4
+ ancienne chargée				0	0	0	0	**0**	2	4	6	8	1	3	**3**	3	2
victime									0	2	4	6	8	1			3

LRU : est victime la page résidente dont la référence est la plus ancienne : 10 défauts de page (5 + 5)

0	**2**	**4**	**6**	**8**	**6**	**4**	**2**	**0**	**1**	**3**	**2**	**4**	**7**	**6**	**3**	**2**	**0**
	0	2	4	6	8	6	4	2	0	1	3	2	4	7	6	3	2
		0	2	4	4	8	6	4	2	0	1	3	2	4	7	6	3
			0	2	2	2	8	6	4	2	0	1	3	2	4	7	6
+ anc. référence.				0	0	0	0	8	6	4	4	0	1	3	2	4	7
victime									8	6			0	1			4

OPTIMUM : est victime la page dont la référence future est la plus lointaine : 8 défauts page ((5 + 3)

0	**2**	**4**	**6**	6	4	2	0	2	2	2	4	6	6	3	2	0	**0**
	0	2	4	4	**2**	0	**2**	4	4	4	6	3	3	2	0	**2**	2
		0	2	2	0	**4**	4	6	6	6	3	2	2	0	**3**	3	3
			0	0	**6**	6	6	**0**	0	**3**	**2**	0	0	**6**	6	6	6
+ loin dans futur				**8**	8	8	8	8	**1**	0	0	**4**	**7**	7	7	7	7
victime									8	1			4				



**FAIBLE INFLUENCE DE D'ALGORITHME DE REMPLACEMENT
À COTÉ DE L'INFLUENCE DE LA TAILLE DE MÉMOIRE PHYSIQUE**

REMPLACEMENT DE PAGE : APPROXIMATIONS DE LRU

FIFO

SECONDE CHANCE voir figure ci-après

SECONDE CHANCE AMÉLIORÉE

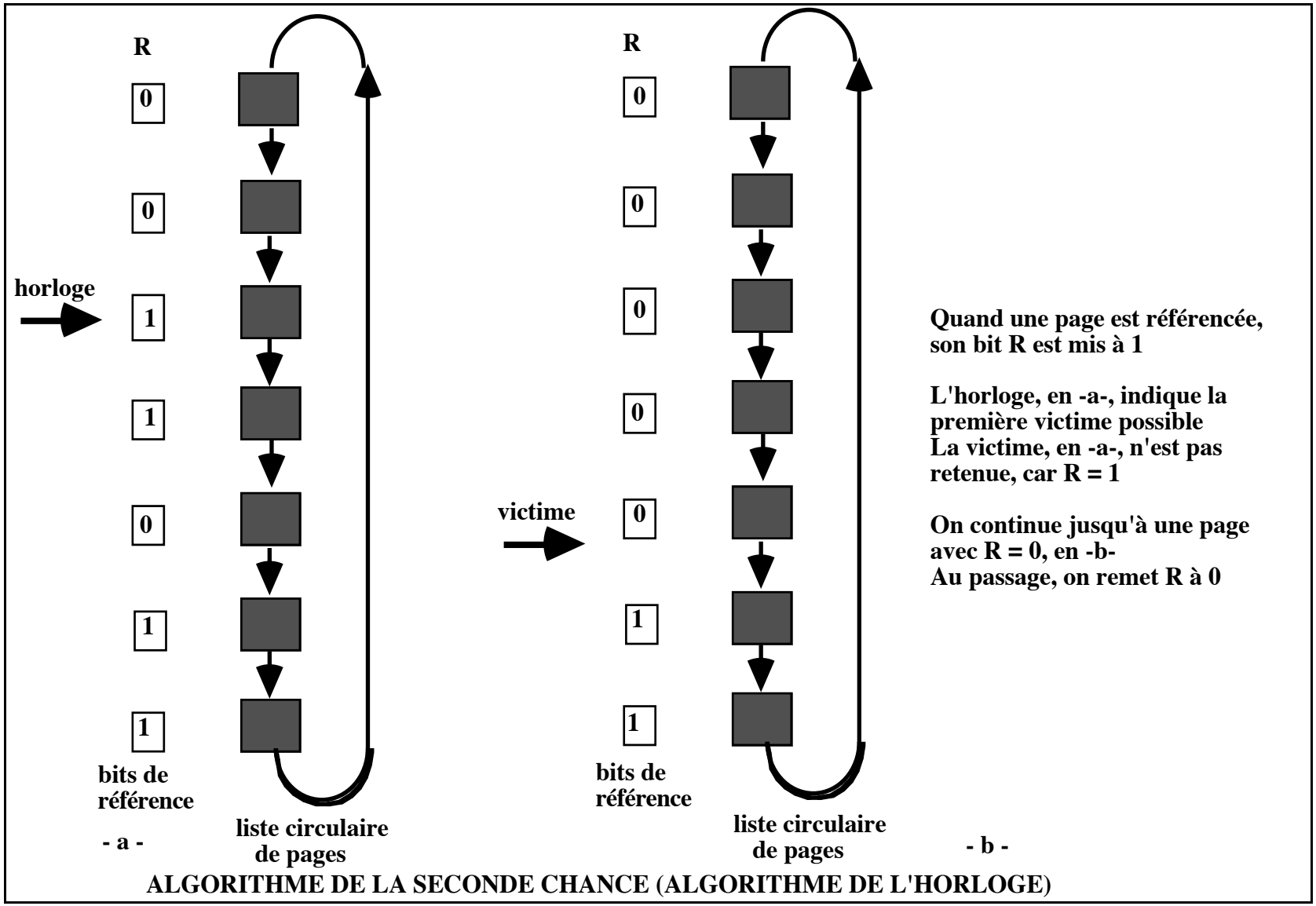
on utilise le couple de bits R (témoin de référence) et M (témoin de modification) pour classer les pages

- **(0, 0) : page ni récemment utilisée, ni modifiée : la meilleure victime**
- **(0, 1) : page pas récemment utilisée, mais modifiée**
- **(1, 0) : page récemment utilisée, mais pas modifiée**
- **(1, 1) : page récemment utilisée et modifiée : la plus mauvaise victime**

On parcourt la liste des pages jusqu'à trouver une victime :

- **d'abord en cherchant dans les pages de la classe (0, 0), ni récemment utilisées, ni modifiées,**
- **puis en cherchant dans les pages de la classe (0, 1), pas récemment utilisées, mais modifiées,**
- **puis en cherchant dans les pages de la classe (0, 0), récemment utilisées, mais pas modifiées,**
- **puis en cherchant dans les pages de la classe (0, 1), récemment utilisées et modifiées.**

On est certain de trouver une victime pendant l'un des 4 parcours



ALGORITHME DE LA SECONDE CHANCE (ALGORITHME DE L'HORLOGE)

POLITIQUES D'ALLOCATION DES PAGES

PLACEMENT COMPLET : PAS DE PAGINATION À LA DEMANDE

le processus doit être entièrement chargé en mémoire centrale avant d'être lancé (en particulier cas du temps réel)
le processus est entièrement résident
avantage : pas d'allocation contigüe, pas de fragmentation externe, les cases utilisables sont quelconques,
simplicité de gestion
et surtout temps d'accès uniforme (pas de délais d'accès variables et inconnus dus à la pagination à la demande)

PAGINATION À LA DEMANDE

ALLOUER UN NOMBRE FIXE DE CASES À CHAQUE PROCESSUS (multiprogrammation constante)

allocation égalitaire : même quantité à chaque processus
allocation proportionnelle à la taille du programme
valeurs fixes pour taux de multiprogrammation donnée
algorithme de remplacement local : la victime est une page du processus

ALLOUER UN NOMBRE VARIABLE DE CASES À CHAQUE PROCESSUS (multiprogrammation variable)

algorithme de remplacement global : la victime est une page d'un processus quelconque
• allocation sauvage : victime vraiment quelconque
• allocation contrôlée : (il existe une politique de contrôle pour éviter l'écroulement) :
• on garde toujours un stock de victimes, hors allocation aux processus
un processus cyclique, démon de pagination, vide périodiquement les pages victimes écrites
• deux classes de processus : les allocataires et les ajournés
- un processus allocataire participe à la multiprogrammation, peut demander des cases et le processeur
- un processus ajourné ne le peut plus et ses pages constituent un premier choix de victimes

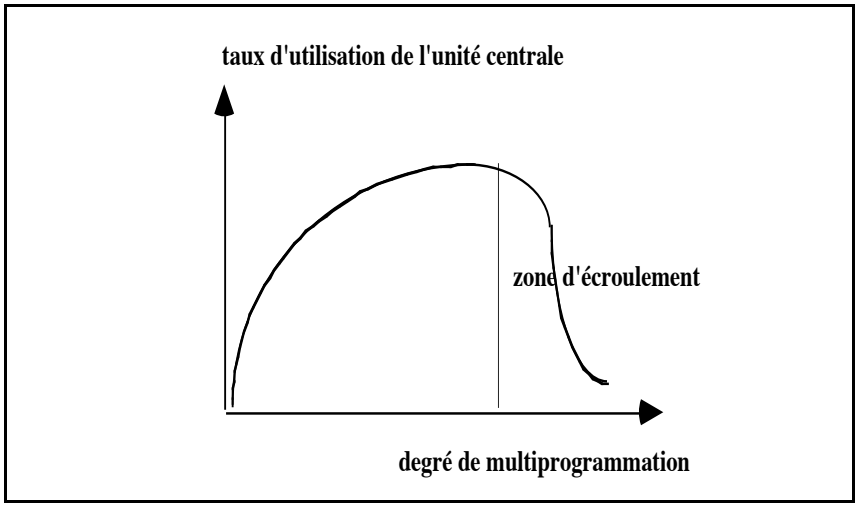
ÉCROULEMENT DU SYSTÈME

Des objectifs mis en oeuvre indépendamment pour :

- **améliorer le rendement du processeur**
  (augmenter la multiprogrammation quand le taux d'E/S croit)
- **améliorer le temps de réponse pour les utilisateurs interactifs en augmentant le partage de la mémoire**
  (grâce à la pagination à la demande et au remplacement)

peuvent conduire à l'écroulement du système :

- **le temps de réponse est augmenté considérablement et les performances sont fortement dégradées**



OBSERVATION DE L'ÉCROULEMENT DU SYSTÈME

PHÉNOMÈNE OBSERVÉ

a) faible utilisation de l'UC -> l'allocateur introduit un processus de plus
b) élection de ce nouveau processus qui n'a aucune page résidente
c) défauts de page avec remplacement et vol de page à un processus prêt -> trafic disque va et vient
d) toujours faible utilisation de l'UC -> l'allocateur introduit un processus de plus
b) se reproduit avant le déblocage des processus en attente de page

CAUSES DE L'ÉCROULEMENT DU SYSTÈME

Le schéma de Von Neumann nécessite une utilisation conjointe de l'unité centrale et de la mémoire avec un espace de travail qui correspond à un minimum d'objets accessibles (code et données)
Si, pour chaque processus, l'espace de travail devient insuffisant, le taux de défaut de page augmente :

- **baisse de la durée d'exécution entre deux défauts de page**
- **augmentation du nombre de transferts de page**
- **retrait de pages à un processus lui-même en attente de page ou au processus qui va être élu et référencer la page qui vient de lui être retirée**

REMÈDES : MESURES DE COMPORTEMENT DU SYSTÈME ET STRATÉGIES GLOBALES

Régulation de charge (ou contrôle de charge) mettant en oeuvre une stratégie globale

mesures de paramètres de régulation du degré de multiprogrammation (régulation de charge)

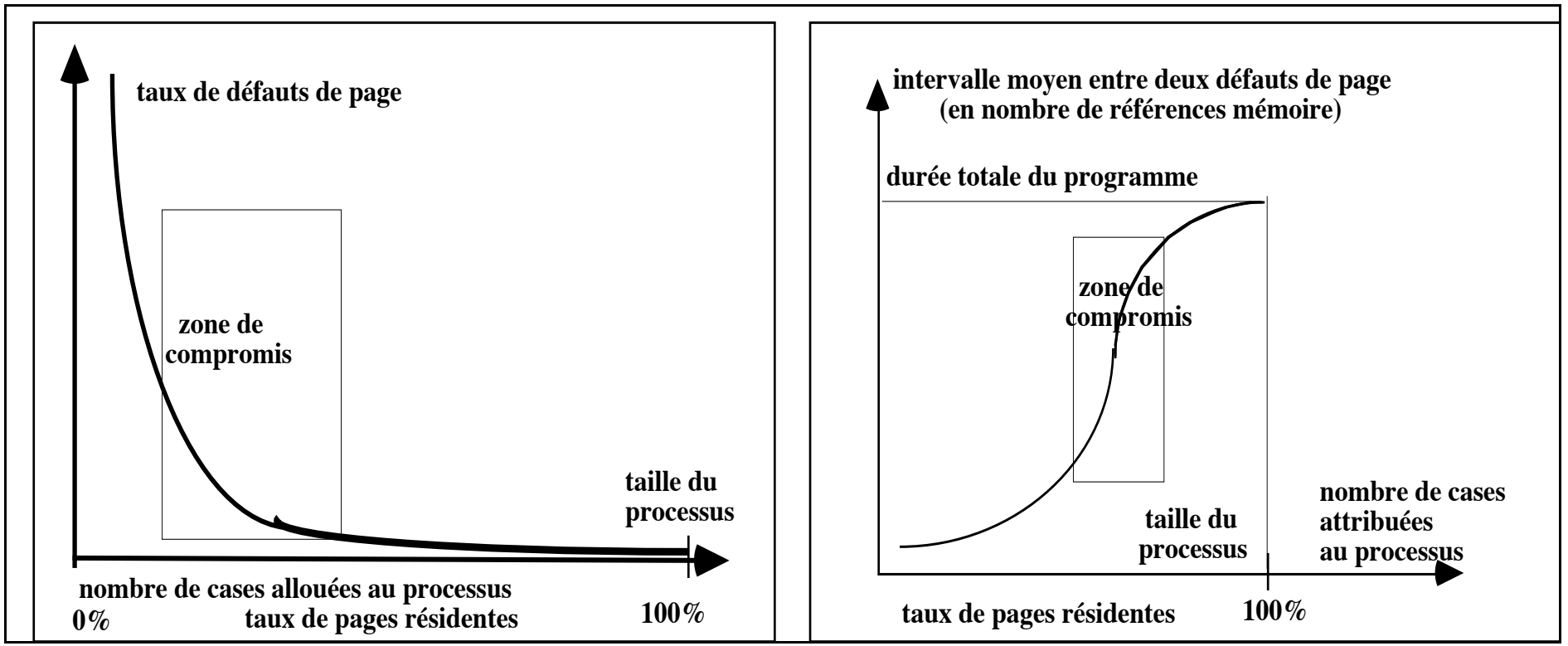
types de mesures en ligne

- **taux d'activité du processeur et trafic de migration**
- **trafic de migration**
- **espace de travail minimum**

A ne pas négliger : reprogrammation pour diminuer la taille de l'espace de travail

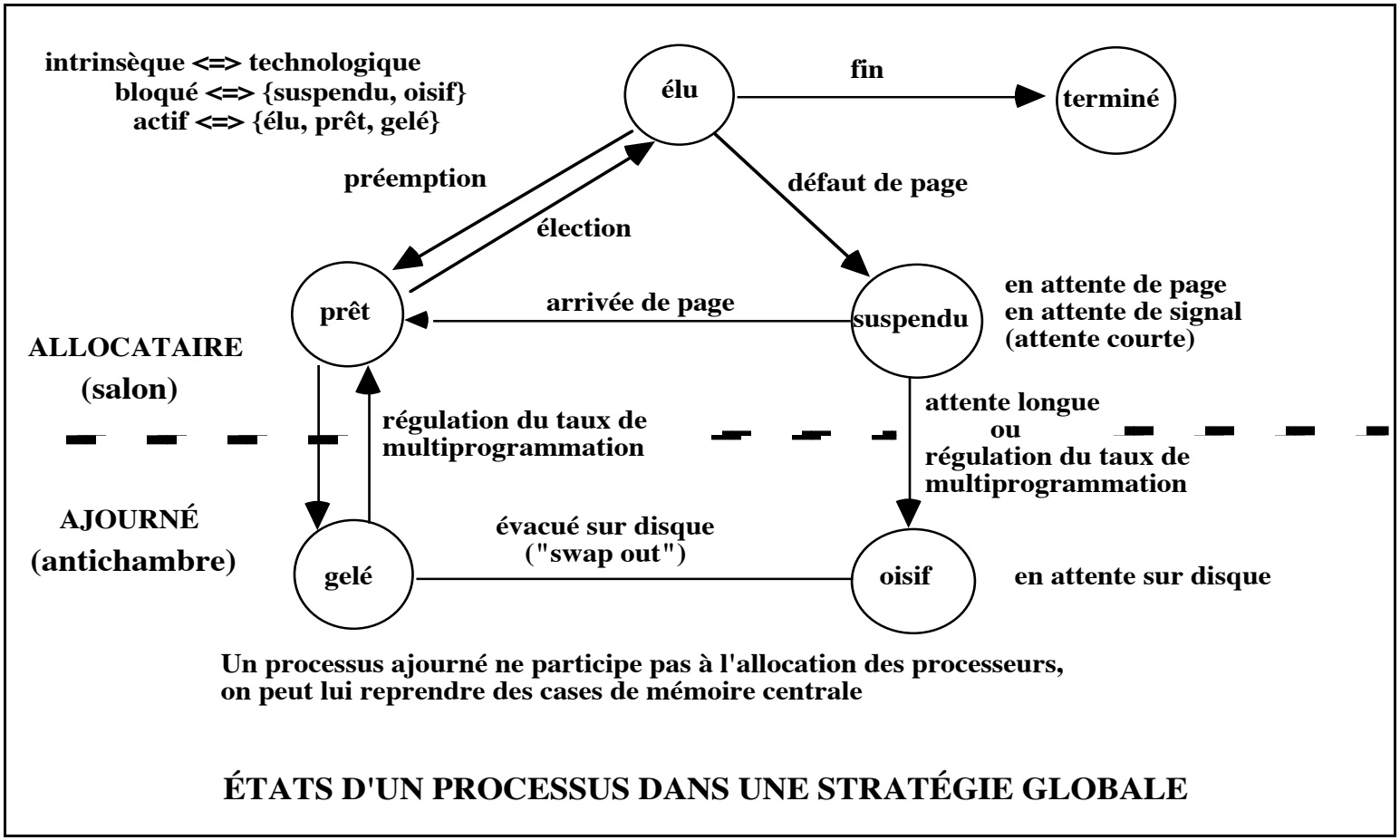
COMPORTEMENT D'UN PROCESSUS EN MÉMOIRE RESTREINTE

OBSERVATIONS MACROSCOPIQUES SUR UN PROCESSUS



CHOIX DU BON TAUX DE MULTIPROGRAMMATION

Il doit résulter d'un compromis entre :
- **le nombre de pages résidentes, déterminant le nombre de références mémoire entre 2 défauts de page**
- **les vitesses relatives du processeur et de l'accès au disque de va et vient.**



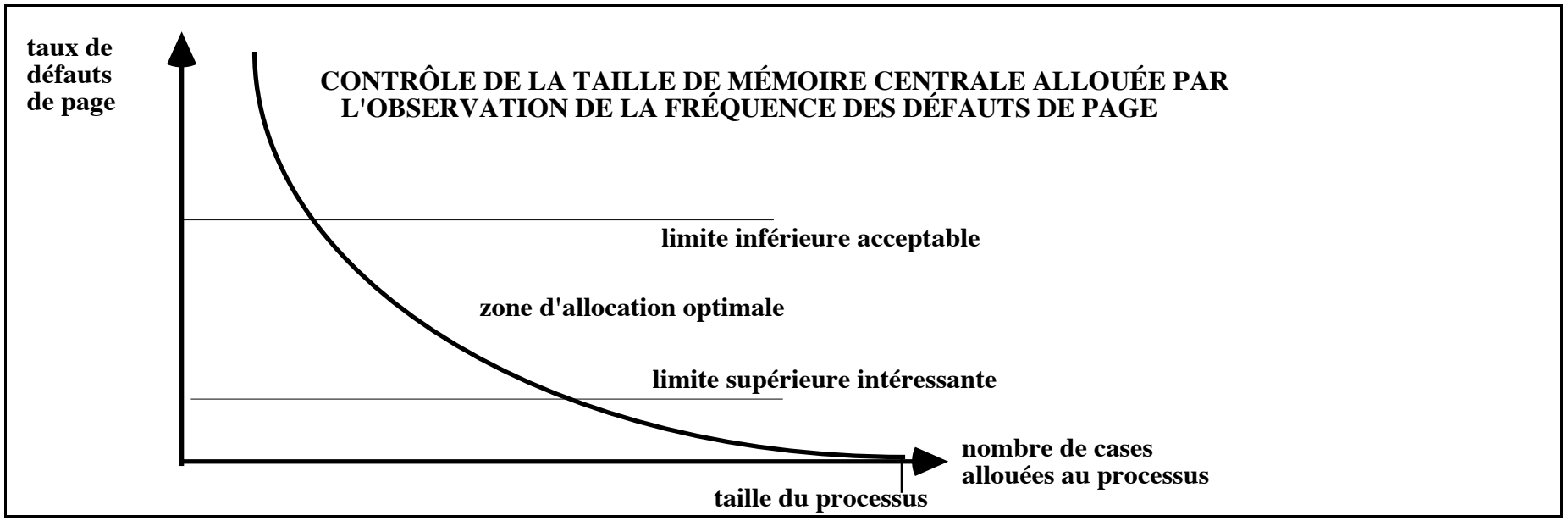
ÉTATS D'UN PROCESSUS DANS UNE STRATÉGIE GLOBALE

STRATÉGIE GLOBALE AVEC RÉGULATION DE LA CHARGE

PREMIER TYPE DE RÉGULATION (OU CONTRÔLE) : PFF

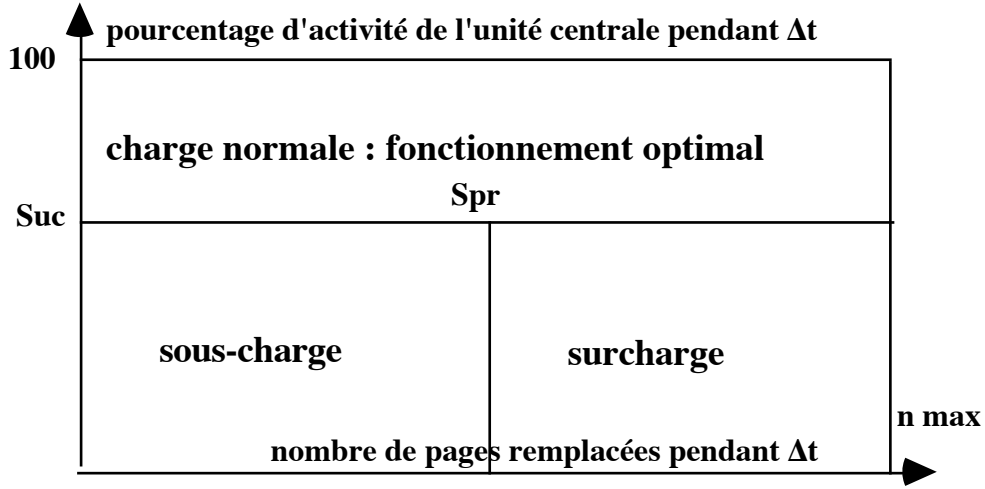
PFF ("Page Fault Frequency" dans Unix ou Linux))

Charge estimée par l'observation de la fréquence des défaux de page (trafic de pagination)

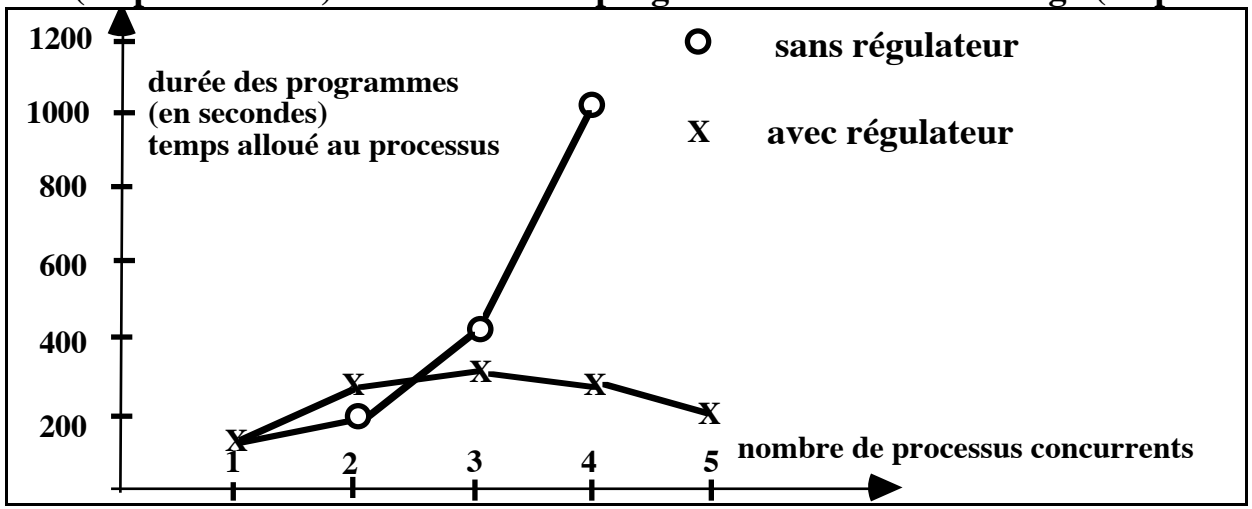


DEUXIÈME TYPE DE RÉGULATION (OU CONTRÔLE) : IBM 370 VM

Charge estimée par deux paramètres : activité de l'unité centrale et nombre de pages remplacées pendant Δt (Shils 1968 avec Δt = 10 s)



régulation : augmenter (resp. diminuer) le taux de multiprogrammation si sous-charge (resp.surcharge)



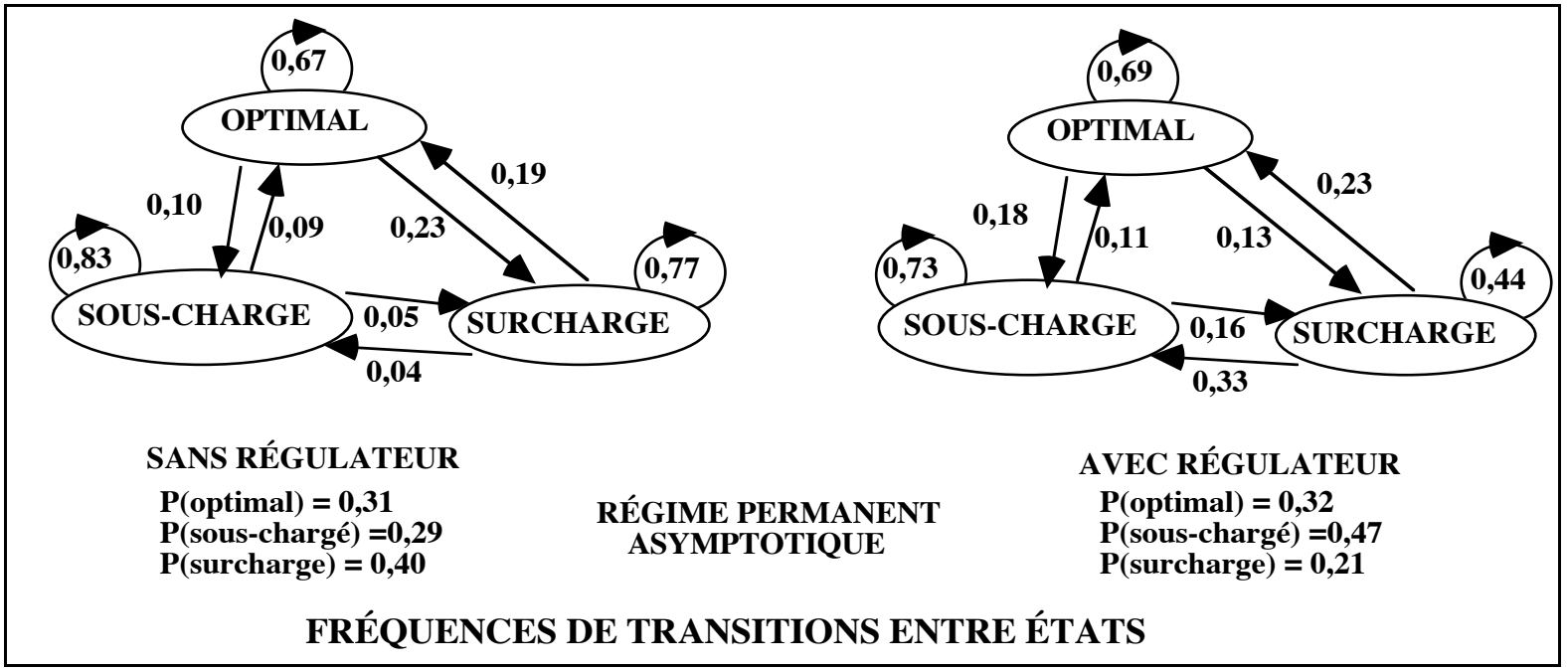
RÉGULATION DANS IBM 370 VM

Cette régulation limite la surcharge, mais peut introduire une oscillation (au profit de la sous-charge)

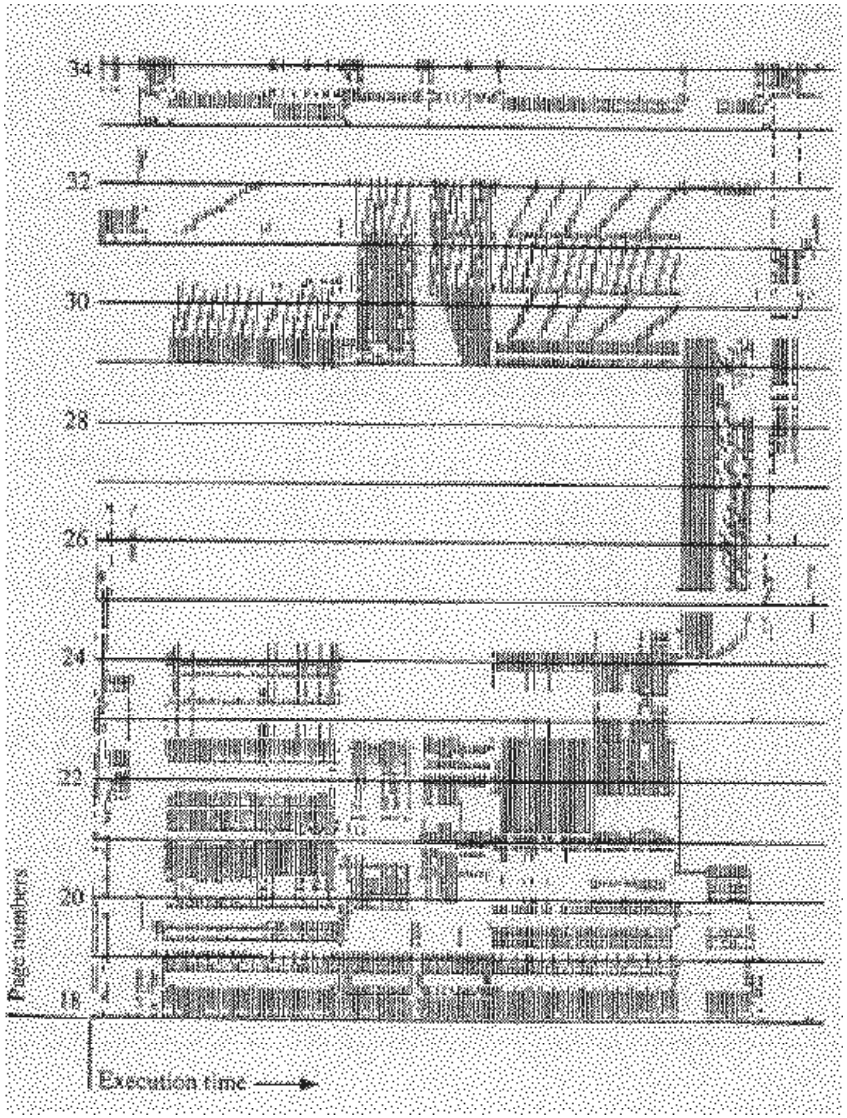
transitions sans régulateur			
après / avant	sous-charge	normal	surcharge
sous-charge	0,83	0,09	0,05
normal	0,10	0,67	0,23
surcharge	0,04	0,19	0,77

transitions avec régulateur			
après / avant	sous-charge	normal	surcharge
sous-charge	0,73	0,11	0,16
normal	0,19	0,69	0,13
surcharge	0,33	0,23	0,44

Oscillation classique en asservissement sans amortissement ou hystérésis



TROISIÈME TYPE DE RÉGULATION PAR ANALYSE DU COMPORTEMENT DES PROGRAMMES



iSuite des références dans l'espace d'adressage d'un programme (Hatfield 1972)

CHARGE PAR ESTIMATION APPROCHÉE DE L'ENSEMBLE DE TRAVAIL (VAX/VMS, IBM/VM, 4.3 BSD UNIX)

L'ENSEMBLE DE TRAVAIL : ET(T, Δ)

CHAÎNE DE RÉFÉRENCES DE PAGES PAR UN PROCESSUS

Exemple

... 2 6 1 5 7 7 7 5 6 2 1 6 2 1 3 4 1 2 3 4 4 4 4 3 4 3 4 4 3 3 1 3 2 3 2 1 1 3 4 3 4 5 4 3 ...
<---------- Δ ----------> t1 ----> t2 <----------Δ ---------> t3

chaîne à t1 : (2 6 1 5 7 7 7 5 6 2) ; chaîne à t2 : (7 7 7 5 6 2 1 6 2 1) ; chaîne à t3 : (4 4 4 3 4 3 4 4 3 3)
Avec Δ = 10, on obtient ET(t1, Δ) = ET(t2, Δ) = {1, 2, 5, 6, 7,} ET(t3, Δ) = {3, 4}
taille de l'ensemble de travail : TET (t1,Δ) = 5 alors que TET(t3,Δ) = 2
taille de l'espace d'adressage (mémoire virtuelle) du programme : 7

PROPRIÉTÉS DE L'ENSEMBLE DE TRAVAIL

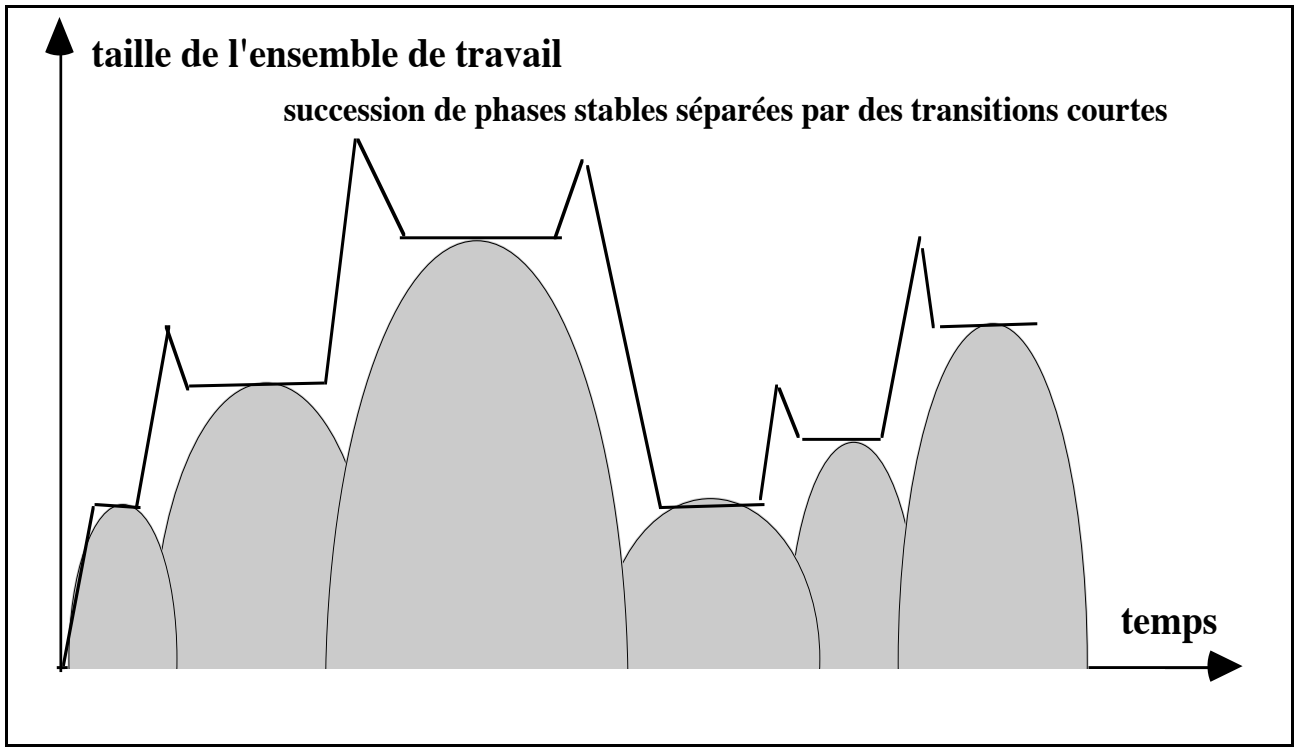
non uniformité : répartition non uniforme. ex. : 75% des références s'adressent à 20% des pages

localité : sur un intervalle t .. t + T, stabilité de l'ensemble de travail ET(t, Δ) et de sa taille TET(t, Δ)

succession de phases stables séparées par des transitions courtes (quelques % du temps d'exécution)

TAILLE DE L'ENSEMBLE DE TRAVAIL : TET(t, Δ) est nettement inférieur à la taille du programme

TAILLE DE L'ENSEMBLE DE TRAVAIL : TET(t, Δ)



ESTIMATION DE L'ENSEMBLE DE TRAVAIL :

en phase stable, ET(t,Δ) est un bon estimateur pour ET(t + Δ, Δ)

TET (t, Δ) est un très bon estimateur de TET(t +Δ, Δ)

ex : 80% à 90% des références dans ET(t, Δ) se retrouvent dans ET(t + Δ, Δ)

RÈGLE D'UTILISATION DE LA MÉMOIRE CENTRALE DE TAILLE TMC

N : TAUX DE MULTIPROGRAMMATION VARIABLE CALCULÉ À CHAQUE VA ET VIENT

Taille mémoire allouable = TMC - taille ensemble de pages victimables - système résident
TETi : taille minimum fixée a priori, taille max variable
EPV : ensemble de pages victimables : taille min fixée a priori, rôle de cache (+ démon de pagination)
Nombre N de processus allocataires : Pi, i $\in$ 1..N, est tel que : $\sum$ TETi(t,Δ) $\leq$ taille mémoire allouable

LORS D'UN DÉFAUT DE PAGE DÛ AU PROCESSUS Pi, on fait nb(i) := nb(i) +1

tester d'abord si la page est encore dans EPV (cela s'appelle récupération de page)
si nb(i) $\leq$ TETi, , faire du remplacement avec une page de EPV (la plus ancienne par exemple)
si nb(i) > TETi, s'il y a encore des pages disponibles dans EPV, alors en prendre une,
sinon remplacer la page la plus ancienne de ETi

EN FIN DE QUANTUM, OU DEMANDE D'ENTRÉE- SORTIE LONGUE, à t

réévaluation de TETi(t,Δ) en examinant l'utilisation des pages pendant t et t - Δ
Les pages qui sont conservées dans ETI(t, Δ) déterminent TETi(t,Δ), les autres sont envoyées dans EPV
Si TETi augmente, on vérifie que le taux de multiprogrammation respecte la contrainte des $\sum$ TETi.

CONTRÔLE DE CHARGE ET RÉGULATION DU TAUX DE MULTIPROGRAMMATION,

Périodiquement à un multiple de quantum (entre 20 et 100 quanta), au départ, au réveil d'un processus,..., on reconstitue la liste des processus allocataires.
On calcule un ordre de priorité entre processus actifs. Cette priorité tient compte du type d'activité du processus, du temps passé comme allocataire et du temps d'attente comme ajourné,...
Deviennent allocataires les N premiers de la liste tels que la contrainte soit respectée (cela détermine N)
Les pages des processus allocataires qui deviennent ajournés sont données à EPV.
En parallèle, le démon de pagination recopie sur disque les pages écrites, sans les effacer.

ACTIONS DE L'UTILISATEUR INTELLIGENT

INFLUENCE DE LA STRUCTURATION DES PROGRAMMES AVEC UNE BONNE LOCALITÉ

Supposons des pages de 2048 octets, soit 512 mots, et un tableau A stocké ligne par ligne

```
A : array(1.. 512, 1.. 512) of Integer;
-- A est rangé séquentiellement : A(1, 1), A(1, 2),..., A(1, 512), A(2, 1), A(2, 2),..., A(512, 512)
```

```
for j in 1..512 loop
    for i in 1.. 512 loop
        A(i, j) := 0;
    end loop;
end loop;
```

```
for i in 1..512 loop
    for j in 1..512 loop
    A(i, j) := 0;
    end loop;
end loop;
```

Parcours de l'espace d'adressage :

AI(1, 1), A(2, 1),...
Un mot par page et passe à la page suivante

taille mémoire allouée de 511 pages pour A =>
512 * 512 défauts de page

A(1, 1), A(1, 2), ...
512 mots par page = une ligne i

taille mémoire allouée de 1 page pour A =>
512 défauts de page

PERMUTATION DE L'ORDRE DE PLACEMENT DES MODULES EN MÉMOIRE VIRTUELLE

• regrouper dans un ensemble de pages minimal des modules s'appelant les uns les autres réduit la taille de l'espace de travail

=> diminution jusqu'à 50% du taux de défaut de page

AUTRES EXEMPLES DE POLITIQUES GLOBALES

Problème à résoudre : éviter les trop longues listes d'attente, cause ou témoin d'inéfficacité
Découper en étapes : antichambre -> salon ; politique -> mécanismes

ex.: prévention d'interblocage (voir chapitre 6):
but : limiter le nombre de processus en attente et immobilisant des ressources déjà allouées
on n'alloue de ressources qu'à un nombre nmax de processus, les autres attendent dans une antichambre
=> meilleure utilisation des ressources, meilleur temps de réponse moyen

ex : ordonnancement des travaux en traitement par lots :
l'ordonnanceur du long terme fait le choix des travaux à multiprogrammer (tient compte des taux d'E/S,...), l'ordonnanceur court terme alloue le processeur aux processus prêts.

ex. : Gestion de tolérance aux fautes temporelles des applications temps réel
(équipe Cratère du laboratoire Cedric/CNAM) :
a) régisseur pour garder les travaux ordonnançables les plus importants,
b) ordonnanceur “earliest deadline first” pour allouer le processeur aux travaux retenus