

TP → Créer du dur mettre dedans

Mémoire virtuelle

INF3173 – Principes des systèmes d'exploitation
Automne 2024

Francis Giraldeau
giraldeau.francis@uqam.ca

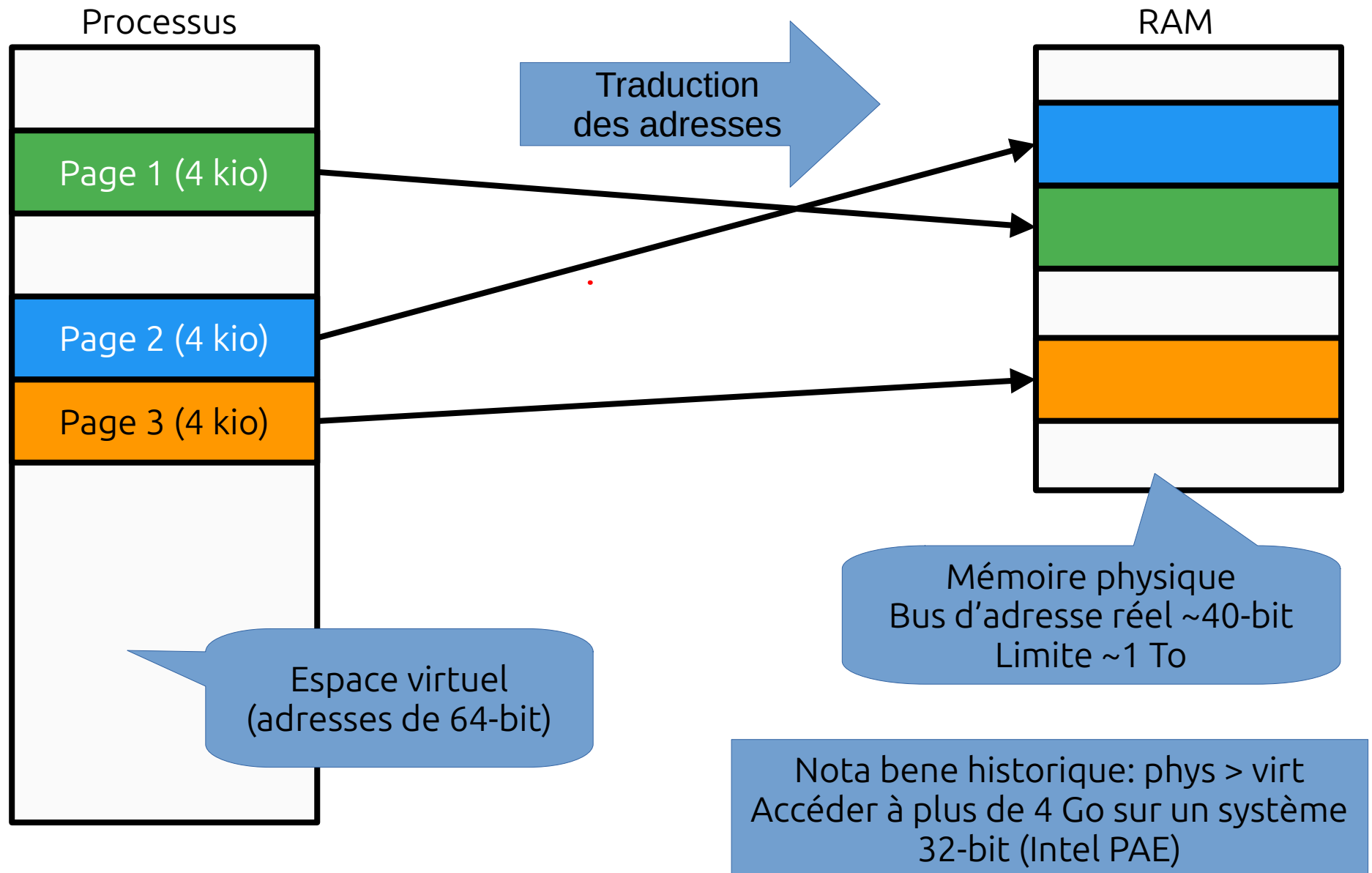
Université du Québec à Montréal



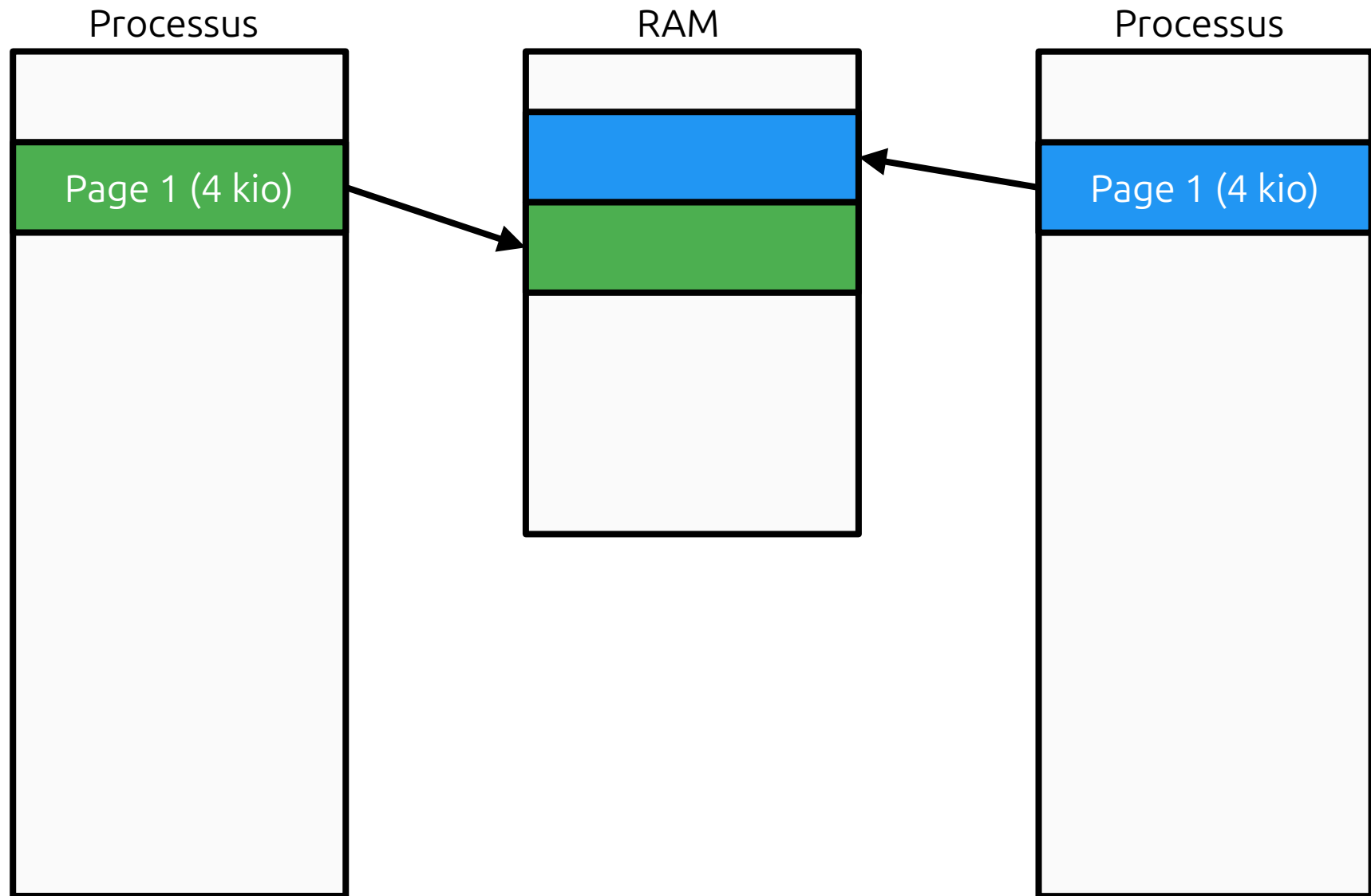
Agenda

- Utilité de la mémoire virtuelle
- Fonctionnement de la mémoire virtuelle
- Carte mémoire d'un processus
- Allocation de mémoire
- Faute de page
- Randomisation des adresses
- Vue directe de fichiers en mémoire (mmap)

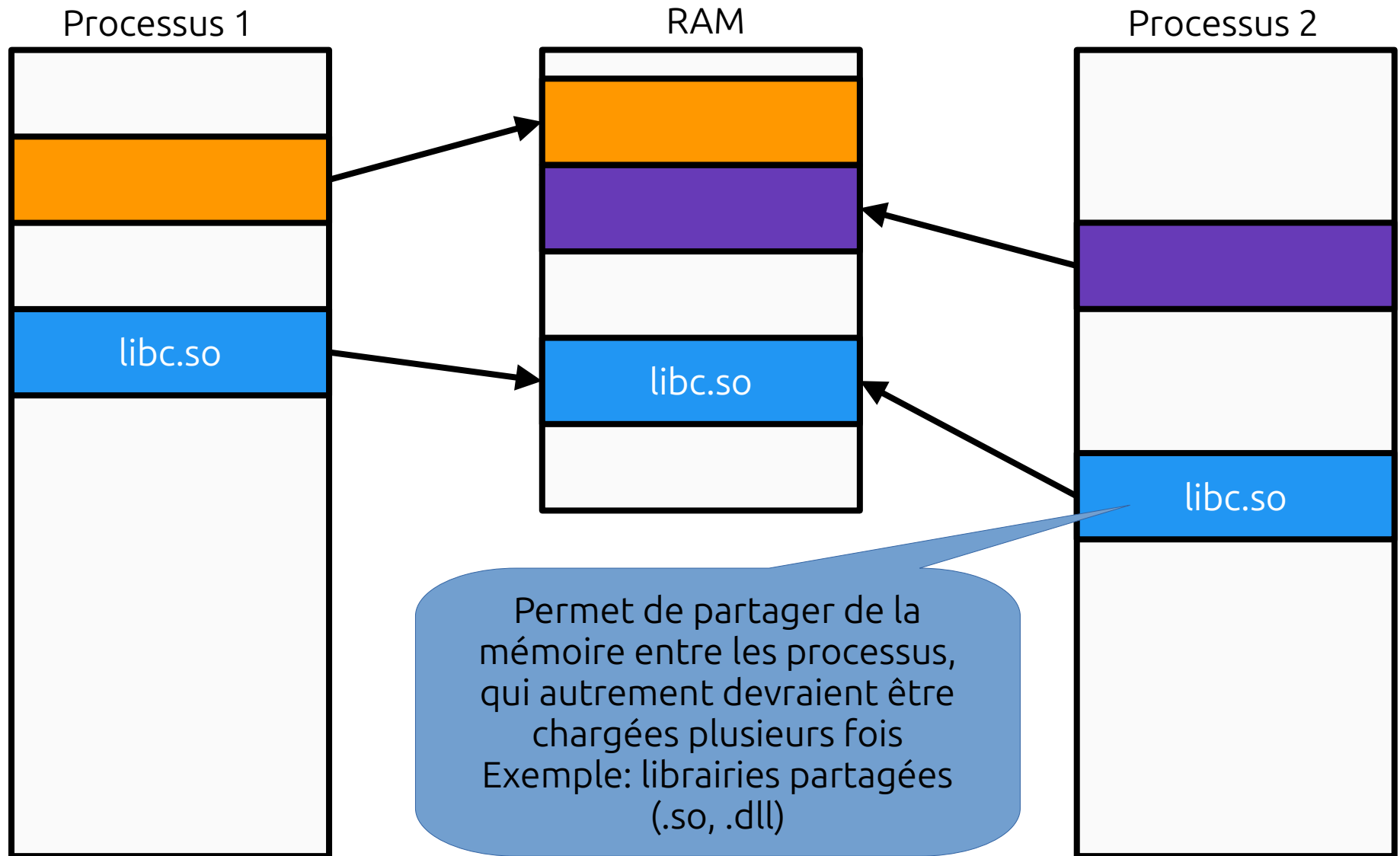
Mémoire virtuelle



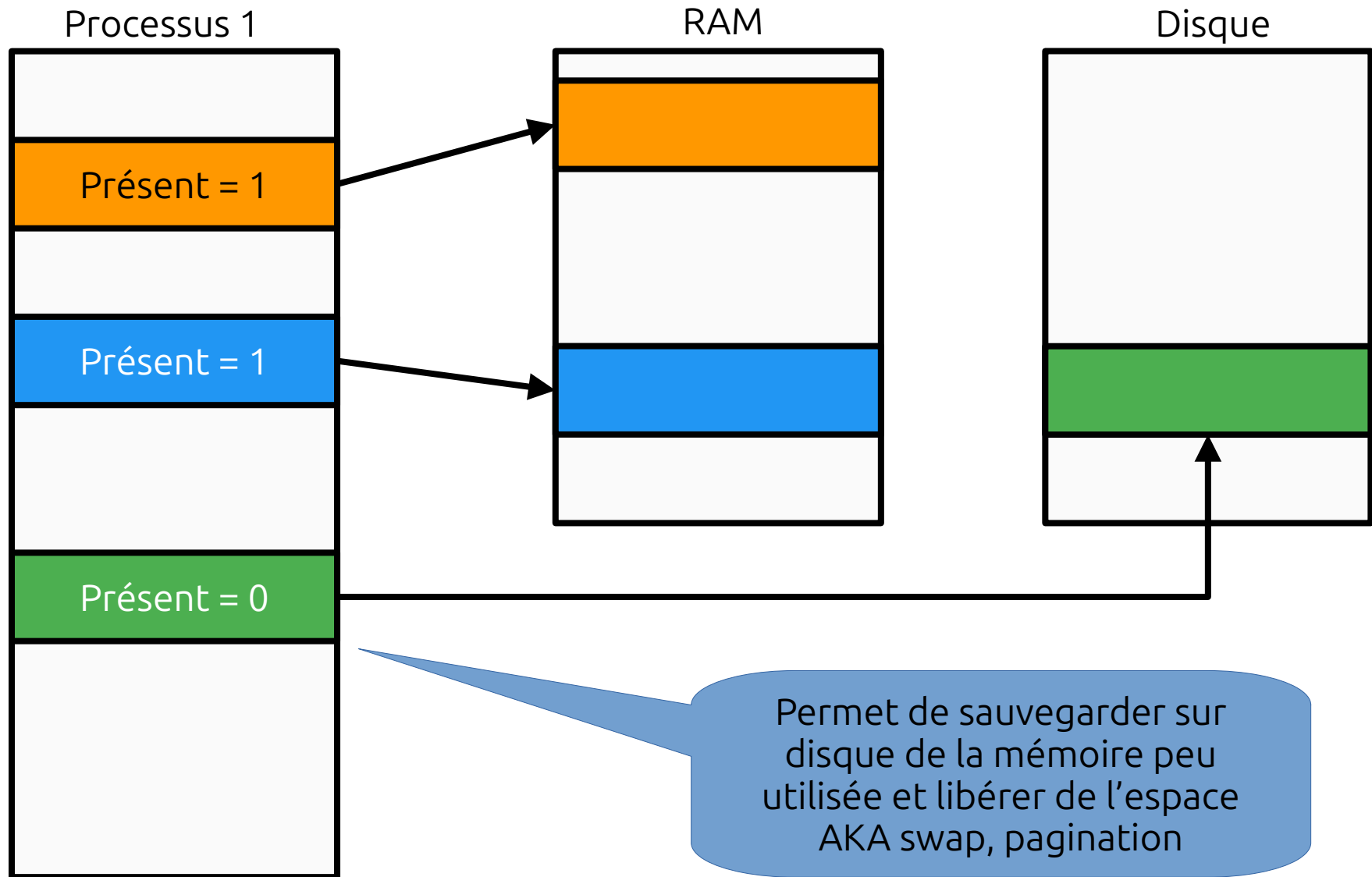
Mémoire virtuelle



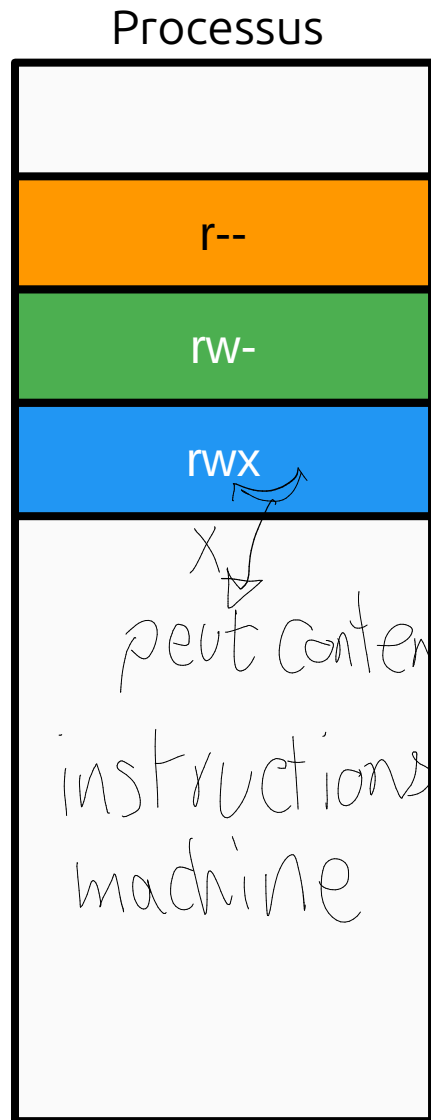
Mémoire virtuelle



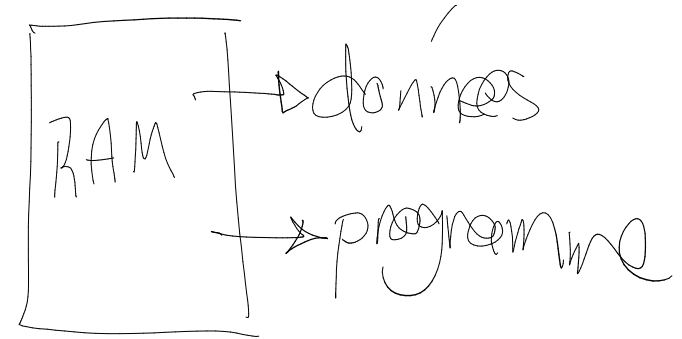
Mémoire virtuelle



Mémoire virtuelle



Permissions:
Lecture (R)
Écriture (W)
Exécution (X)
mprotect(): changer
les permissions



Faute de page: interruption émise par le processeur pour avertir le système d'exploitation dans les situations suivantes:

- Adresse en dehors de l'espace du processus
- Non respect des permissions
- Page non-présente en mémoire

Le système d'exploitation décide quoi faire (charger une page du disque, terminer le programme, etc.)

Défaut de cache v.s. Faute de page

Bloc memoir

- | | |
|--|--|
| <ul style="list-style-type: none"> • Survient quand une donnée n'est pas dans la cache du processeur • Niveau L1 > L2 > L3 > RAM • Événement de la microarchitecture • Résolution sans l'intervention du SÉ | <ul style="list-style-type: none"> • Type d'interruption • Suite à un accès invalide à une adresse mémoire • Le SÉ doit intervenir <ul style="list-style-type: none"> – Ajouter une page – Charger du disque – Terminer l'application • AKA Défaut de page |
|--|--|

Résumé des objectifs

- Donne l'illusion d'un plus grand espace mémoire qu'il n'existe en réalité
 - Espace d'adressage plus grand que le bus physique
 - Sauvegarder des pages peu utilisées sur disque et libérer de la mémoire
- Deux processus peuvent utiliser les mêmes adresses (pas de conflit)
- Charger une seule copie des librairies partagées (économie de mémoire)
- Appliquer des permissions

Quelle taille de page choisir?

- Petite taille (4 kio)
- Réduit les pertes de fragmentation
- Augmente le nombre de pages à gérer
- Grosses pages (4 Mio)
- Augmente les pertes par fragmentation
- Diminue le nombre de pages à gérer

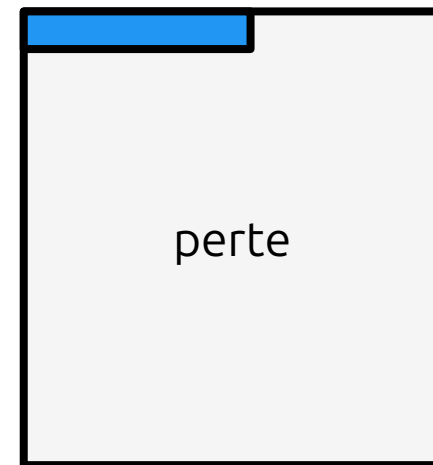
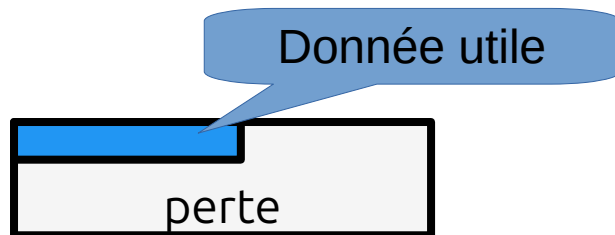
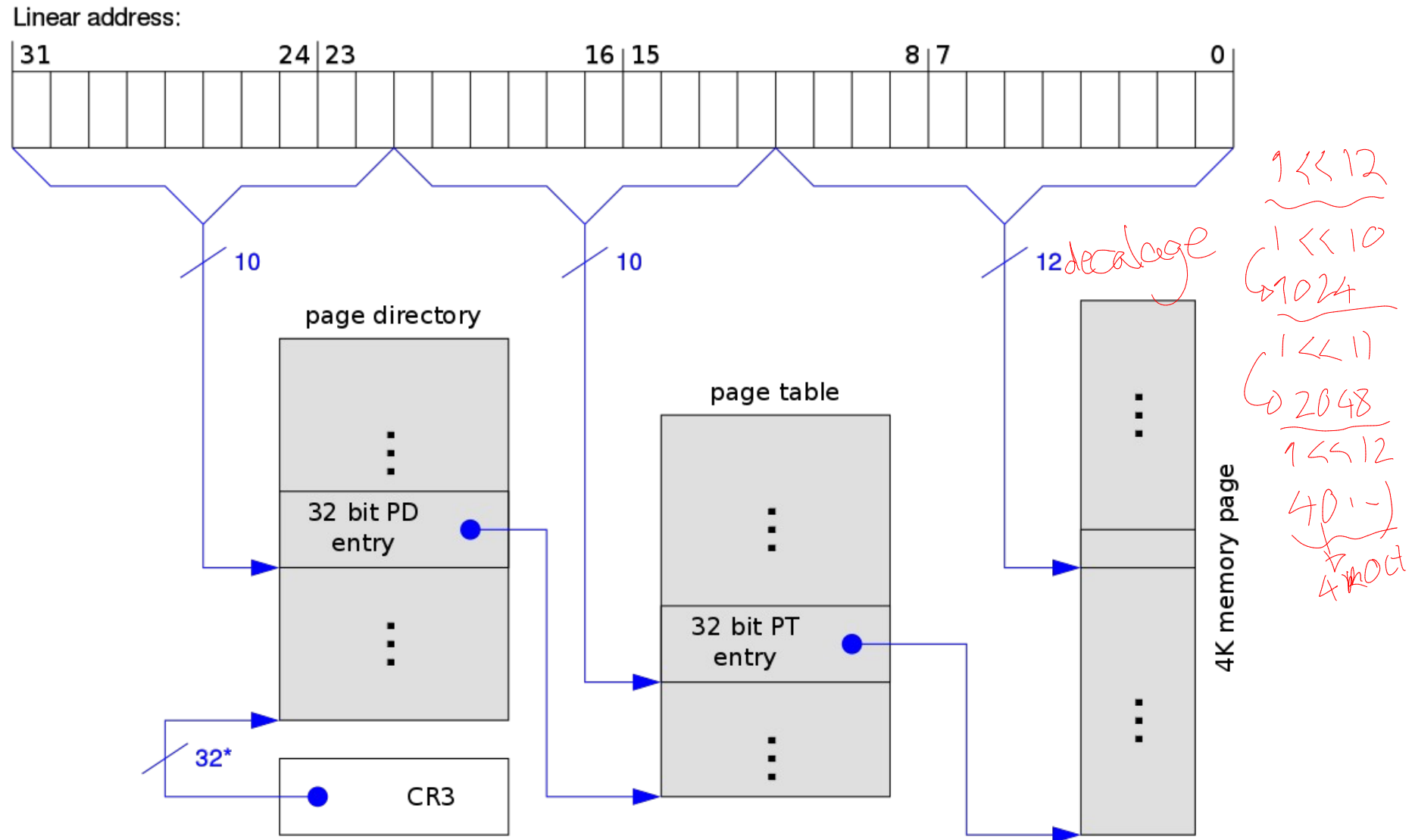


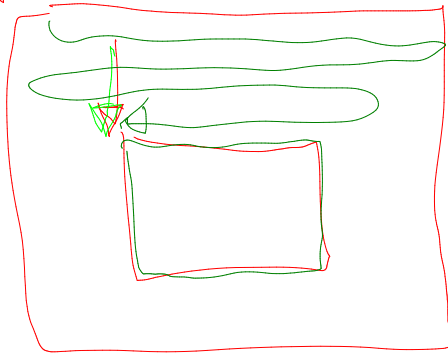
Table de page (4 kio)



*) 32 bits aligned to a 4-KByte boundary

Source de l'image: https://en.wikipedia.org/wiki/Physical_Address_Extension

0
↓
addr debut de page



mod
sous

long addr = (unsigned long) ptr;

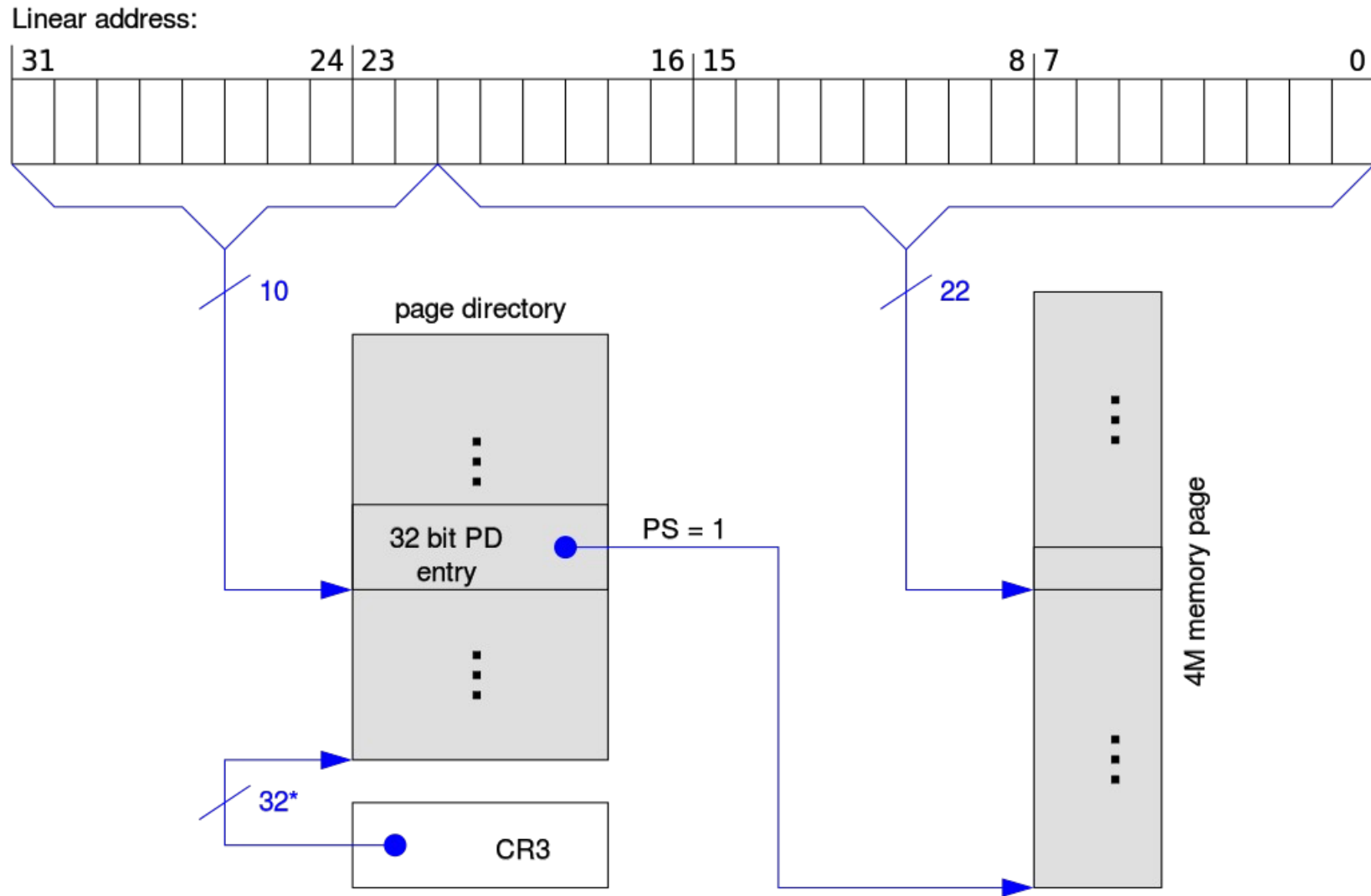
long reste = addr - reste;

void debutpage_ptr = (void*) debut_page;

ghex "name-executable" → pour voir
bin
hexdump -C "file-name"

4095

Table de page (4 Mio)



*) 32 bits aligned to a 4-KByte boundy

Source de l'image: https://en.wikipedia.org/wiki/Physical_Address_Extension

Entrée dans la table de page (Intel)

31	12	11	9	8	7	6	5	4	3	2	1	0	
Physical page number	AVL	G	P A T	D	A	P C D	P W T	U / S	R / W	P	(a)		

63	62	52	51	12	11	9	8	7	6	5	4	3	2	1	0	
N X	AVL		Physical page number			AVL	G	P A T	D	A	P C D	P W T	U / S	R / W	P	(b)

NX – No eXecute

AVL – AVaiLable to the OS

G – Global page

PAT – Page Attribute Table

D – Dirty (modified)

A – Accessed (referenced)

PCD – Page Cache Disable

PWT – Page Write-Through

U/S – User/Supervisor

R/W – Read/Write access

P – Present (valid)

Cache de traduction: TLB

- Traduire les adresses virtuelles est complexe et très fréquent (chaque instruction!)
- Solution: utiliser une cache de la traduction
- TLB: Translation Lookaside Buffer
- Doit être invalidée (au moins partiellement) après un changement de processus (ordonnancement)

TLB: Impact de performance

- Parcours de la table de page: 100ns
- Accès TLB: 20ns
- Taux de succès en cache TLB $\Rightarrow s$
- Temps d'accès moyen = $s * 20 + (1 - s) * 100$

Faute de page mineur v.s. Majeur

- | | |
|--|--|
| <ul style="list-style-type: none">• Mineur (rapide)• Gestion faite sur-le-champs• Ajout d'une page zéro dans une plage valide• Copie d'une page• Délai de l'ordre microseconde | <ul style="list-style-type: none">• Majeur (lent)• La page demandée requiert un accès disque• Processus mis à l'état bloqué, ordonnanceur invoqué• Délai de l'ordre de la milliseconde (dépend du périphérique) |
|--|--|

Carte mémoire d'un processus

- Fichier /proc/<PID>/maps
- Fichier éphémère (disparaît quand le processus se termine)

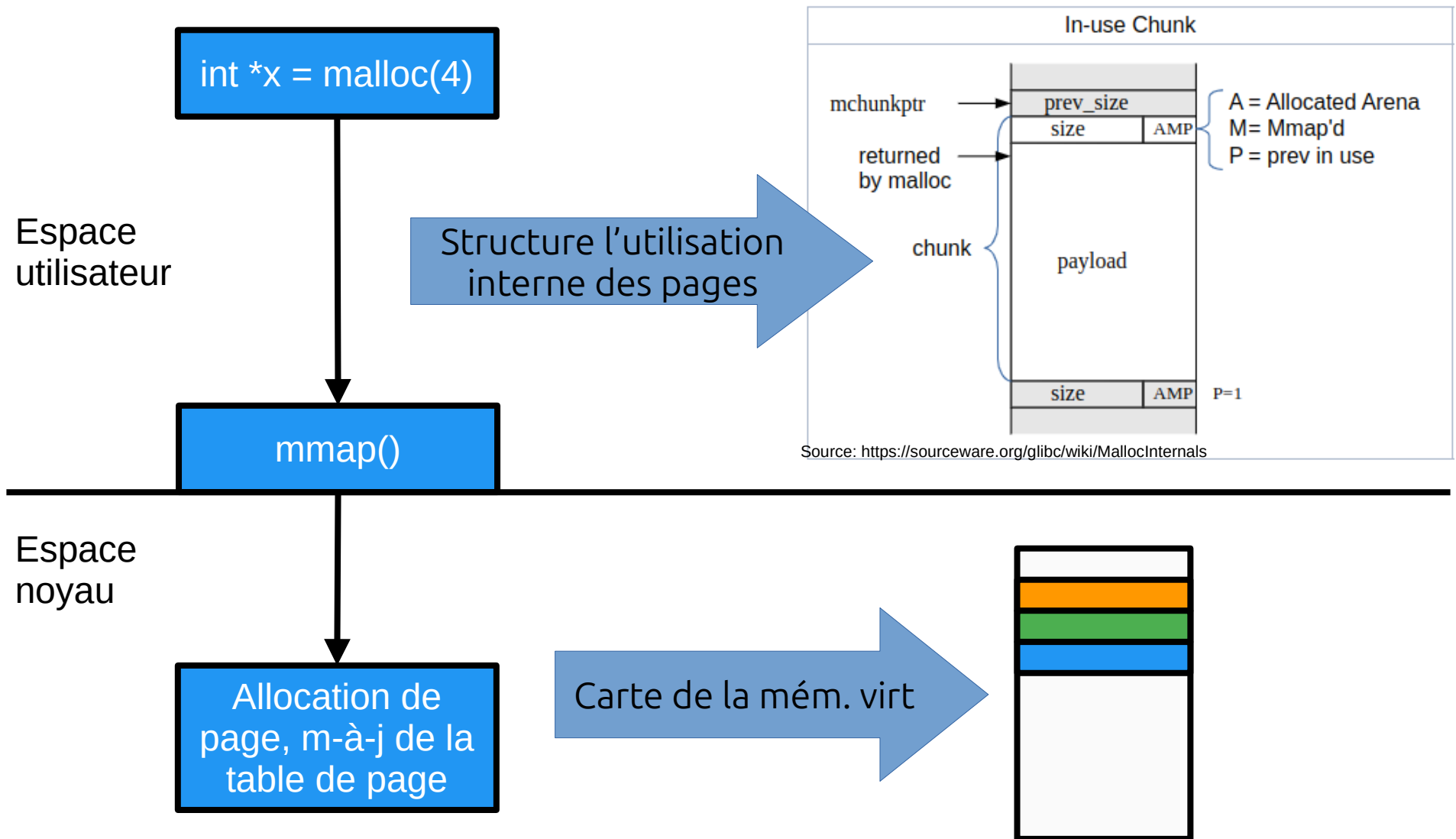
Intervalle d'adresses, permissions, décalage, périphérique, inode, chemin

→ tous le proc a un file *→ avec 'mprotection', on peut changer la permission*

```
francis@berta:~$ cat /proc/415494/maps
563514412000-563514414000 r--p 00000000 00:1c 9088 /usr/bin/sleep
563514414000-563514418000 r-xp 00002000 00:1c 9088 /usr/bin/sleep
563514418000-563514419000 r--p 00006000 00:1c 9088 /usr/bin/sleep
56351441a000-56351441b000 r--p 00007000 00:1c 9088 /usr/bin/sleep
56351441b000-56351441c000 rw-p 00008000 00:1c 9088 /usr/bin/sleep
56351508d000-5635150ae000 rw-p 00000000 00:00 0 [heap]
7f8ee961c000-7f8ee961d000 r--p 00000000 00:1c 27760 /usr/lib/locale/locale-archive
7f8ee961d000-7f8ee961e000 rw-p 00000000 00:00 0
7f8ee961e000-7f8ee961f000 r--p 00000000 00:1c 59450 /usr/lib/x86_64-linux-gnu/libc.so.6
7f8ee961f000-7f8ee9620000 r-xp 00028000 00:1c 59450 /usr/lib/x86_64-linux-gnu/libc.so.6
7f8ee9620000-7f8ee9621000 r--p 001bd000 00:1c 59450 /usr/lib/x86_64-linux-gnu/libc.so.6
7f8ee9621000-7f8ee9622000 r--p 00214000 00:1c 59450 /usr/lib/x86_64-linux-gnu/libc.so.6
7f8ee9622000-7f8ee9623000 rw-p 00218000 00:1c 59450 /usr/lib/x86_64-linux-gnu/libc.so.6
7f8ee9623000-7f8ee9624000 rw-p 00000000 00:00 0
7f8ee9624000-7f8ee9625000 rw-p 00000000 00:00 0
7f8ee9625000-7f8ee9626000 r--p 00000000 00:1c 58828 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f8ee9626000-7f8ee9627000 r-xp 00002000 00:1c 58828 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f8ee9627000-7f8ee9628000 r--p 0002c000 00:1c 58828 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f8ee9628000-7f8ee9629000 r--p 00037000 00:1c 58828 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f8ee9629000-7f8ee962a000 rw-p 00039000 00:1c 58828 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffcf1d20000-7ffcf1d21000 rw-p 00000000 00:00 0 [stack]
7ffcf1d21000-7ffcf1d22000 r--p 00000000 00:00 0 [vvar]
7ffcf1d22000-7ffcf1d23000 r-xp 00000000 00:00 0 [vdso]
7ffcf1d23000-7ffcf1d24000 r-xp 00000000 00:00 0 [vsyscall]
```

par ex un variable
un instruction
→ variable global
165-21
2 5 167-23
61849d1367 000
61849d1278 000
0 f 000
978944 oct
1140% = 239 pages
(bogue de code)
Debut
fin
fin - Debut → taille zone (octet)
nbr page = taille zone / taille page → #page

Allocation de mémoire



Appel système mmap()

Si on fait write → il faut pas sync tout suite

- ANONYMOUS v.s. fichier

- Anonyme: obtenir un espace mémoire (i.e. malloc())
- Fichier: projeter un fichier sur disque en mémoire (sans read/write!)

Sync → est global il faut pour tous

- PRIVATE v.s. SHARED

- Privé: Modifications non-visibles aux autres processus (COW)
- Partagé: Modifications visibles entre processus + modifie fichier

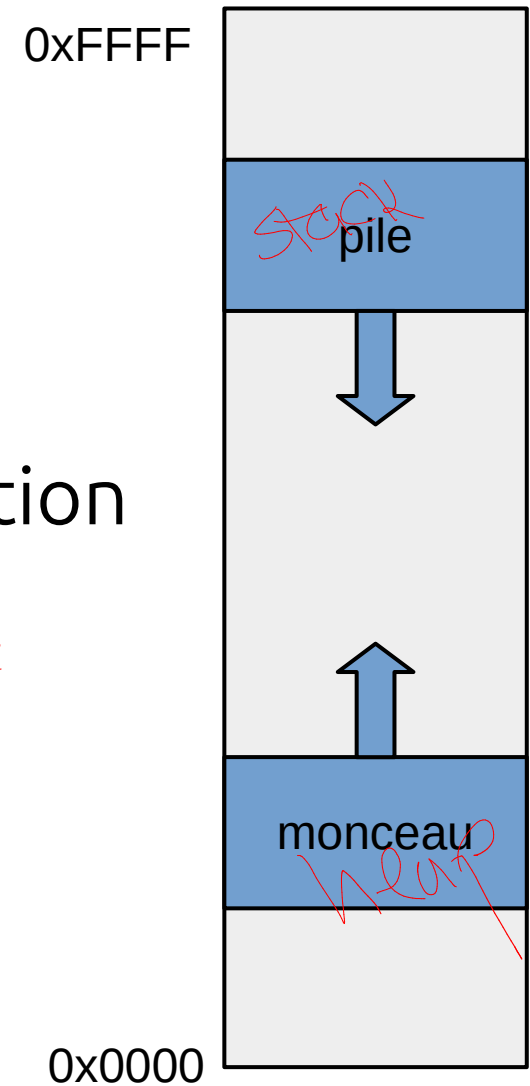
- Par défaut, la page accédée est chargée au premier accès

- Les modifications en mémoire sont éventuellement écrites sur disques

- Écrire dans la page la marque comme "dirty"
- Périodiquement, les pages "dirty" sont écrites sur disque
- Forcer à écrire sur disque avec msync()

Allocation mémoire avec brk/sbrk

- Utilisé pour le monceau du programme
- Agrandir et rétrécir une plage mémoire du processus
- Différence avec la pile: l'allocation persiste après le retour d'une fonction
- Agrandir vers des adresses plus élevées *→ prag break → zone mémoire*
- brk(addr): adresse absolue
- sbrk(décalage): agrandissement ou rétrécissement



malloc → gros quantité → mmap
sbrk → petit quantité

brk - TRESHOLD → 128KB

sup fichier contient les pages inutile

\$ dd if = /dev/zero of = bidon bs = 1M
count = 1000

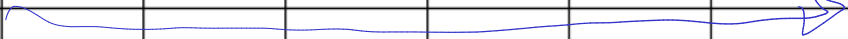
create fichier

Pagination

- Mécanisme pour donner l'illusion d'avoir plus de mémoire qu'en réalité (surengagement)
- Sauvegarder une page en mémoire sur disque et réutiliser la page pour une donnée utile (implique une faute de page)
- Sur quelle base décider de libérer une page?
- Si on libère une page et que l'on y accède l'instant suivant, alors on a une forte pénalité.
- La meilleure décision: prédire le futur
- Basée sur l'âge (la plus ancienne est libérée) *→ ordre d'insertion*
- Basé sur l'usage (page inutilisée) *→ Bit d'accès*

Algorithme FIFO

Tableau accès avec 4 pages disponibles

Temps	0	1	2	3	4	5	6	7	8	9	10	11
Accès	10	20	20	10	30	40	10	30	40	10	20	30
PAGE 1	10	10	10	10	10	10						
PAGE 2		20	20	20	20	20						
PAGE 3					30	30						
PAGE 4						40						
Faute	✓	✓	X	X	✓	✓						

↳ quand on change un page à

On remplace la page qui est la plus ancienne en priorité

Procédure: maintenir une liste des pages chargées. Si la page n'est pas dans la liste, ajouter à la fin, si la liste est pleine, évincer celle du début

Algorithme FIFO

Tableau accès avec 4 pages disponibles

Temps	0	1	2	3	4	5	6	7	8	9	10	11
Accès	10	20	20	10	30	40	10	30	40	10	20	30
PAGE 1	10	10	10	10	30	30						
PAGE 2		20	20	20	20	40						
PAGE 3												
PAGE 4												
Faute	V	V	X	X								

On remplace la page qui est la plus ancienne en priorité

Procédure: maintenir une liste des pages chargées. Si la page n'est pas dans la liste, ajouter à la fin, si la liste est pleine, évincer celle du début

Algorithme Least Recently Used (LRU)

Tableau accès avec 4 pages disponibles

Temps	0	1	2	3	4	5	6	7	8	9	10	11
Accès	10	20	20	10	30	40	10	30	40	10	20	30
PAGE 1	10	10	10	10	10	40	40	30				
PAGE 2		20	20	20	30	30	10	10				
PAGE 3												
PAGE 4												
Faute	✓	✓	X	X	✓							

→ getr usage - → il donne nbr faute de page

On remplace la page qui est la plus ancienne en priorité

Procédure: maintenir une liste des pages chargées. Si la page est dans la liste, enlever de la liste et remettre au début, si la liste est pleine, évincer celle à la fin de la liste

Choix des pages à évincer en réalité

- Algorithme “Pas utilisé récemment” pour une question de performance
- Approximation de LRU basé sur le bit d'accès de la page
- Le système d'exploitation met à zéro le bit d'accès des pages périodiquement
- Le processeur remet à 1 le bit lors d'un accès à la page (le système d'exploitation n'intervient pas)
- Si le bit d'accès à une page reste à zéro, alors il s'agit d'une page inutilisée récemment
- Le noyau maintient une liste de pages froides sujette à être évincée sur disque
- Ensemble de travail (Working Set Size)

Copy-On-Write (COW)

- Après `fork()`, les pages `MAP_PRIVATE` sont protégées en écriture
- L'accès en écriture cause une faute de page
- Une nouvelle page est allouée et le contenu de la page d'origine est copié
- L'écriture s'effectue dans la nouvelle page
- Évite de copier tout l'espace d'emblée
 - Souvent, `fork()` est suivi de `exec()` qui réinitialise tout l'espace mémoire de toute façon!

Écroulement (*trashing*)

- Quand la mémoire disponible est très faible, alors le SÉ sauvegarde des pages sur disque
- Le système continue de fonctionner, mais cause un ralentissement important
- Stratégie pour atténuer le problème: mettre sur disque des pages AVANT de manquer de mémoire
- Paramètre noyau “vm.swappiness”
 - Valeur de 0: utilise la pagination en dernier recours
 - Valeur 100: utilise la pagination agressivement
 - Valeur par défaut sur Ubuntu: 60
 - `cat /proc/sys/vm/swappiness` → il attend de 60% de remplir RAM, après il commence

Métriques sur la mémoire

- VIRT: Somme de toute la mémoire virtuelle
 - Surestime l'utilisation réelle de la mémoire
 - Certaines pages sont sur disque ou partagées
- RES: Resident (RAM occupée)
- SHR: mémoire du processus + proportion des pages partagées

Tasks: 380 total, 1 running, 379 sleeping, 0 stopped,
%Cpu(s): 1.9 us, 0.5 sy, 0.0 ni, 97.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 68.9/11853.4 [|||||
MiB Swap: 0.6/2048.0 [|||

Affichage de top

PID	USER	PR	N	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5293	francis	20	0	13.6g	342728	33884	S	3.8	2.8	24:09.08	teams
3770	francis	20	0	1366352	248084	176260	S	1.9	2.0	56:51.71	Xorg
4048	francis	20	0	4812572	267896	39320	S	1.9	2.2	55:42.22	gnome-shell
5136	francis	20	0	1034684	52872	12808	S	1.9	0.4	17:45.53	teams

Cache des pages

→ si tu liras une file
pour 2em fois est plus
rapide

- Lorsqu'une page est chargée du disque, mais n'est plus requise, elle est gardée en cache
- Si elle est requise, elle est déjà en mémoire et on évite l'accès disque

```
francis@berta:~$ free -mh
```

	total	used	free	shared	buff/cache	available
Mem:	11Gi	6.9Gi	3.1Gi	904Mi	1.6Gi	3.4Gi
Swap:	2.0Gi	11Mi	2.0Gi			

```
$ echo 3 | sudo tee /proc/sys/vm/drop_caches
```

Libère toutes les
pages en cache

```
francis@berta:~$ free -mh
```

	total	used	free	shared	buff/cache	available
Mem:	11Gi	5.4Gi	4.7Gi	932Mi	1.4Gi	4.8Gi
Swap:	2.0Gi	11Mi	2.0Gi			

Oublie tout qui a fait avant

Libération de 1.5 Gi...
Mais il faudra recharger les pages depuis le disque

Résumé appels systèmes

- Mmap : miroir en mémoire
- Sbrk : allocation monceau
- Mincore : déterminer si une page est présente
- Sync : synchroniser sur disque

$f_{write} \xrightarrow{apple} write$

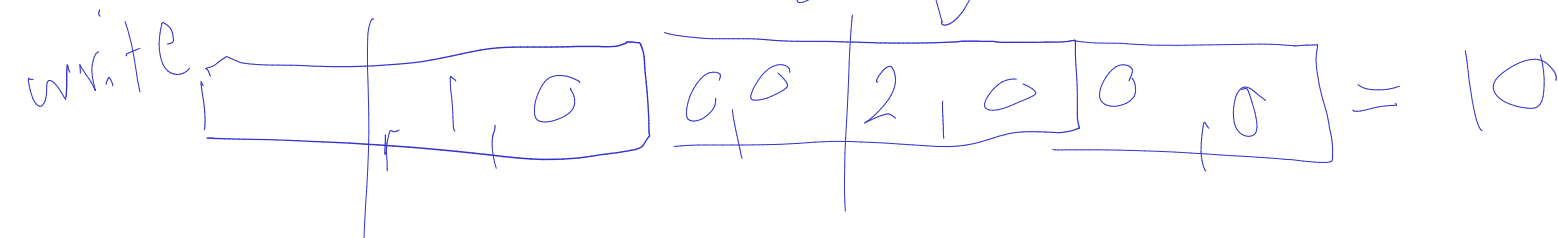
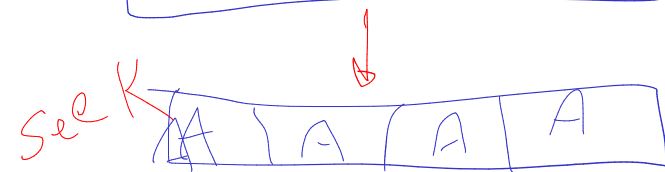
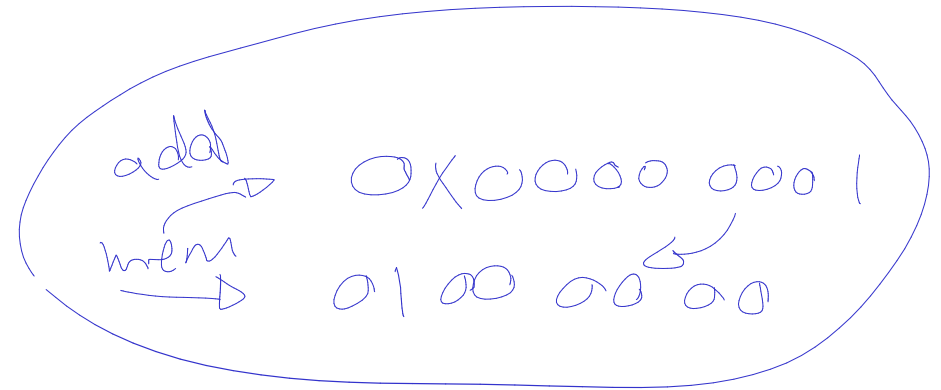
③ data $\{^4, ^4\} = 8$
magic $= 4$

write

$\leftarrow 4$

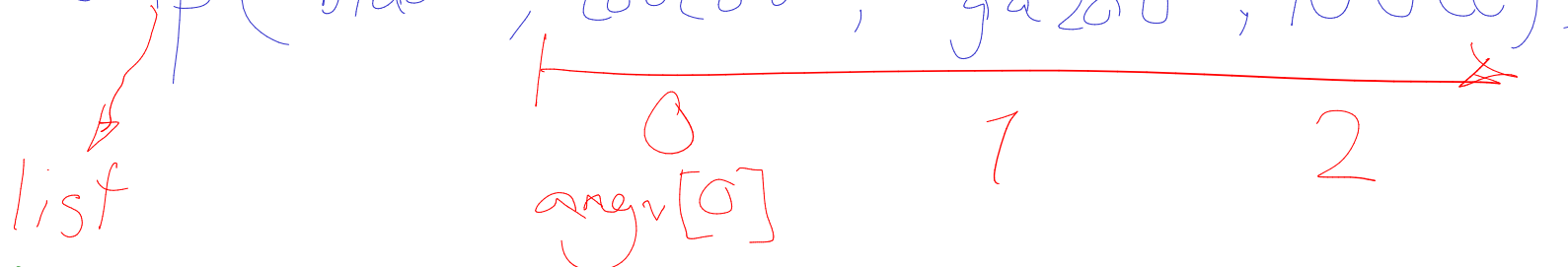
$\leftarrow 2$

$\leftarrow 10$



plus table
à gauche

petit
Boutist

5) `exclp("bidon", "coucou", "gazon", NULL);`

list
0
1
2
argv[0]

6) wait \rightarrow parent attend de child.

8) pthread_create & fork() \rightarrow done

9) goto

10) Critic $\rightarrow 10^{-12}$ picoseconde

instruction \rightarrow nanoseconde

microsec
microsecond \rightarrow si c'est pas disc mecanique

time mystere

real 2.8s

user 2.7s \rightarrow tmp CPU passé en mode utilisateur (vert)

sys 0.1 \rightarrow CPU mode sys / noyau \rightarrow exec (Blue)

Calcul

User + Sys \rightarrow tmp CPU total

$0.1 + 2.7 = 2.8 = \text{egal tmps écoulé}$

1	{ \rightarrow bloqué		user 2.49		sys 0.05	{ \rightarrow preempté
0,002						
0						

14

1. fichier exist pas
2. permissions insuffisantes.
3. max fichiers ouvert _{attents}
4. disque plein

13

pos	pos < file size
0	✓
4	✓
8	✓
12	✓

apres read return 0 pour ca
et il rest 12 qui est
egale à

4
byte write(fd, &data, sizeof(data))

byte	byte < 4	Cas err	
0	✓	0	✓
4	✓	-1	✓
		-1	✓
		⋮	