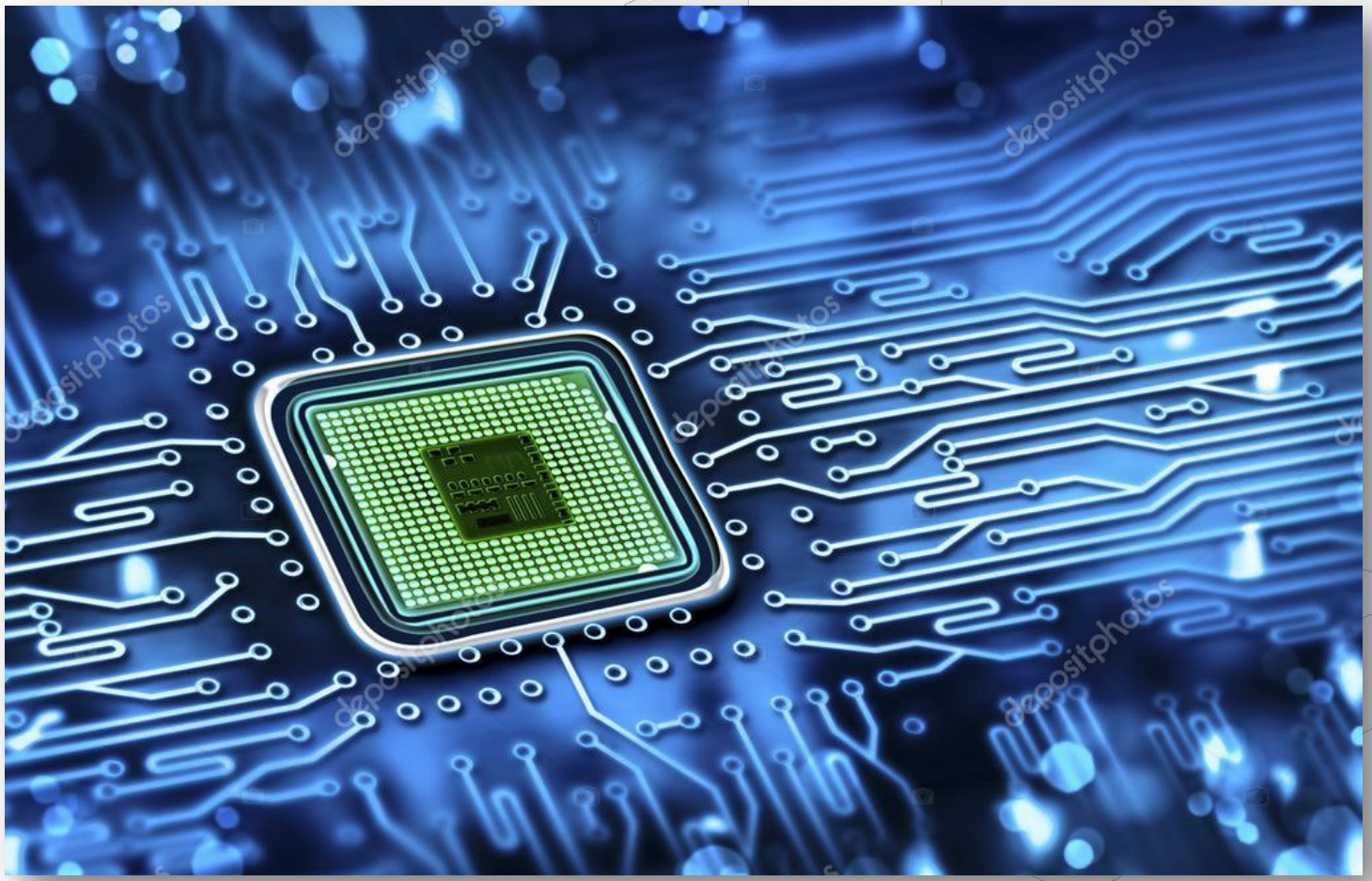
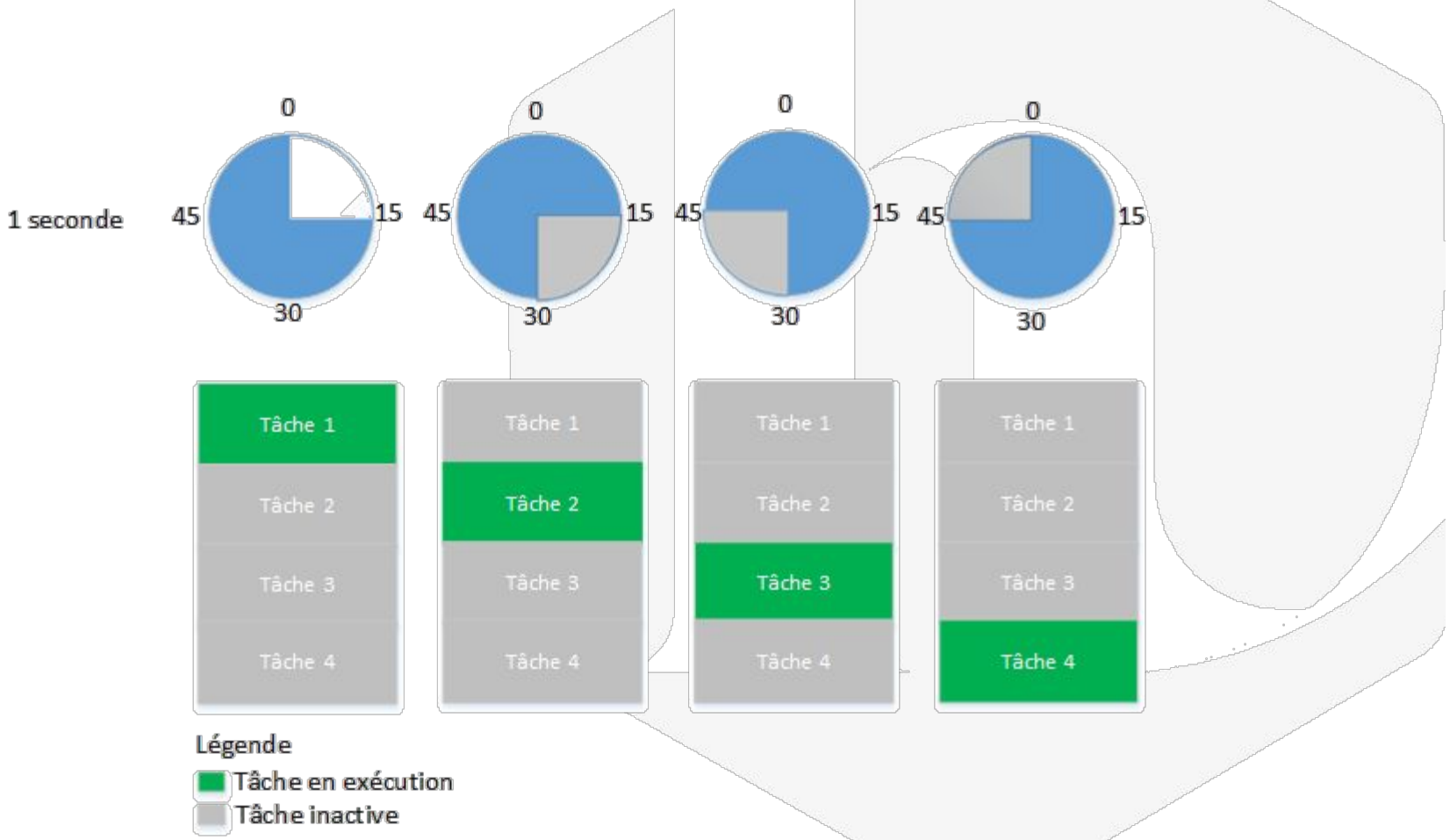
hexagone
École Supérieure d'Informatique

Les processus

Au cœur du hardware



Une affaire de tâche



Le parallélisme

- plusieurs niveaux auxquels l'exécution en parallèle peut être visible :
  - un cluster réseau coordonnera ses différents éléments au niveau applicatif
  - processeur multi-cœur ou processeurs distincts partageant une même mémoire;
  - le temps CPU fractionné entre les différents programmes fonctionnant en parallèle

- Synchroniser les accès aux ressources pour éviter les incohérences
  - verrouillage d’une zone mémoire commune

Le temps partagé

- Chaque CPU ne permet à un instant donné l'exécution que d'un seul lot d'instructions.

- Vision séquentielle
  - l'interpréteur de commandes attend les instructions au clavier ;
  - il se bloque pour exécuter la commande voulue ;
  - il se replace en écoute une fois la commande terminée.

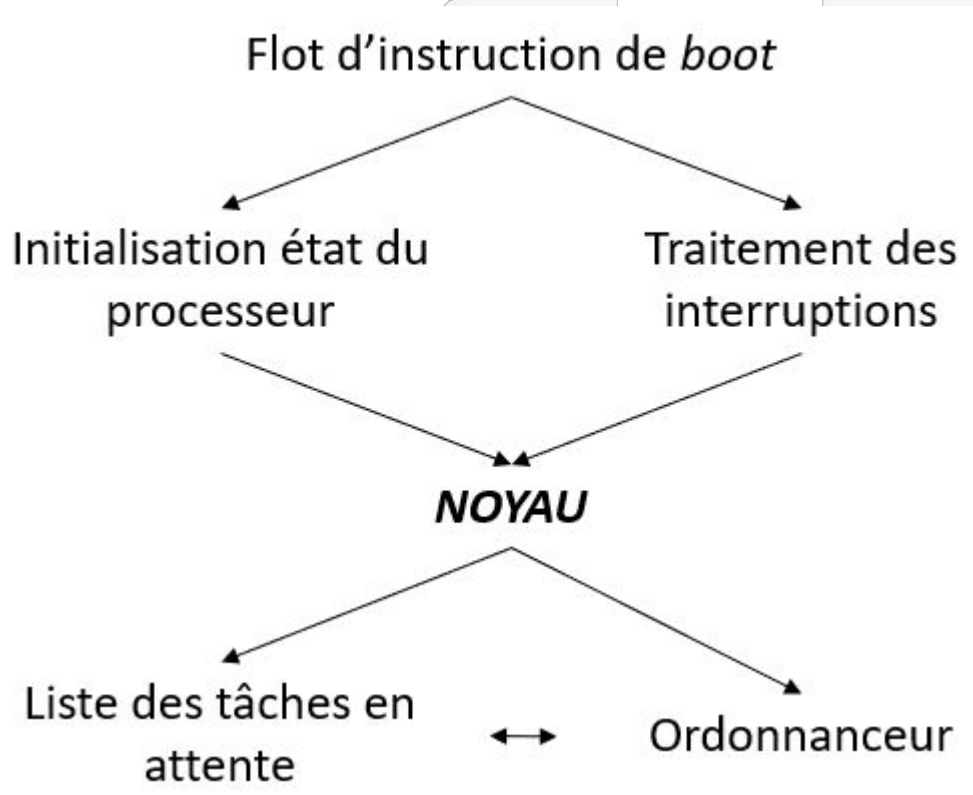
- inacceptable pour au moins deux raisons :
  - les performances (perte de temps durant les attentes) ;
  - le confort d'utilisation (on attend la fin de l’appli 1 pour accéder à l’appli 2).

Le temps partagé

- le système va fractionner le temps CPU entre différents programmes :
  - qui s'exécuteront rapidement à tour de rôle ;
  - illusion d'un parallélisme.

- mécanismes matériels :
  - pagination mémoire : partager la RAM de façon transparente entre tâches ;
  - interruptions : pour interrompre le flot d'instructions en cours de traitement ;
  - Instructions CPU.

Lancement de l'ordonnancement

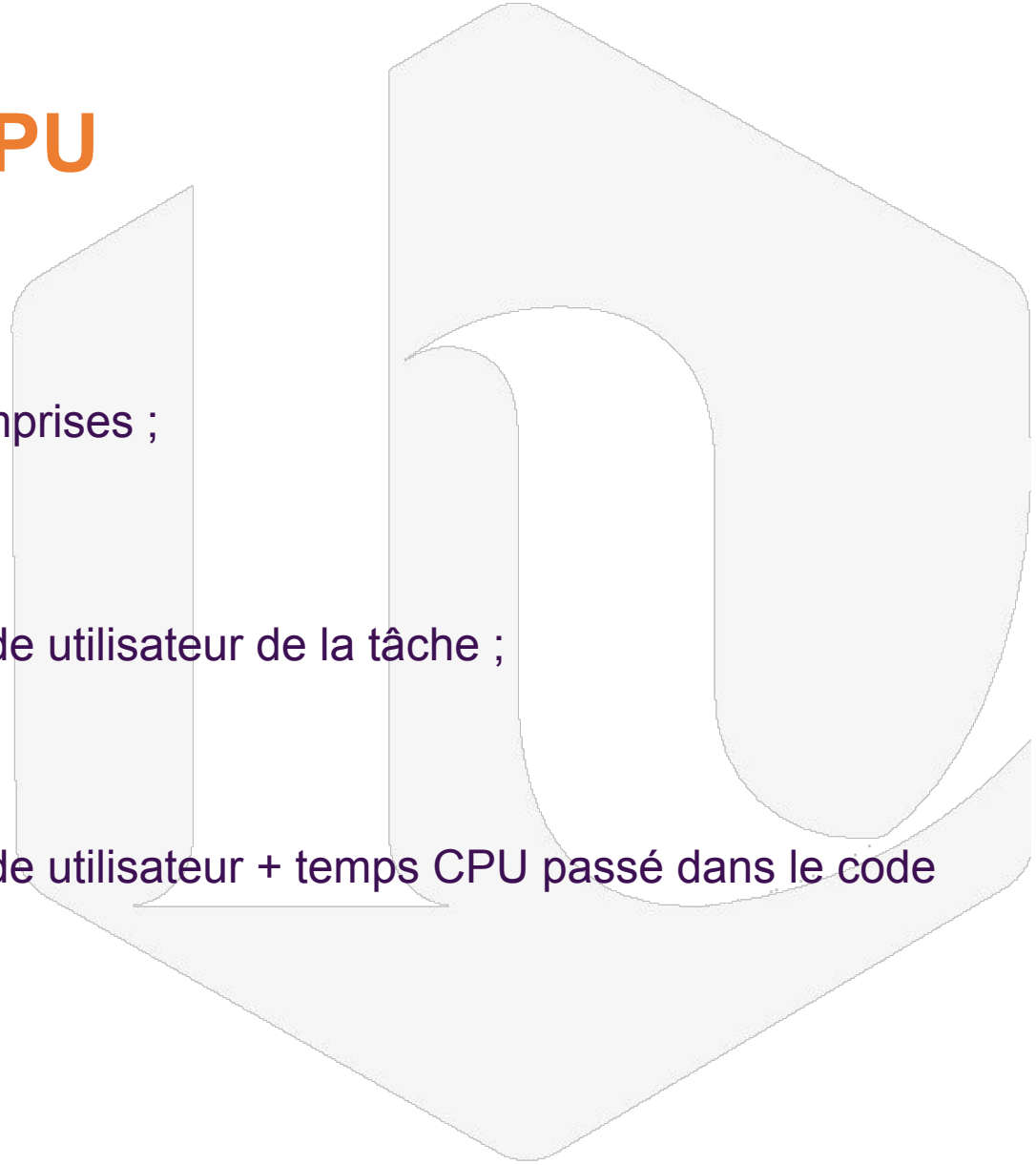


Répartition du temps CPU

- tâches utilisateur (root ou non) :
  - déroulement du code et bibliothèques
  - routines de traitement des signaux reçus
- tâches noyau :
  - traitement des appels système
  - traitement des exceptions processeur générées par l'exécution du code (erreurs applicatives)
  - threads noyau (code dans le noyau et qui participent à son fonctionnement)
- traitement des interruptions externes (horloge, périphériques, etc...).

Différents temps CPU

- temps réel écoulé
  - temps CPU toutes tâches comprises ;
- temps virtuel écoulé
  - temps CPU passé dans le code utilisateur de la tâche ;
- temps d'activité
  - temps CPU passé dans le code utilisateur + temps CPU passé dans le code noyau.



Appel à l’ordonnanceur

- interruptions d'horloge, pour éventuellement changer la tâche active ;
- autres événements :
  - interruptions physiques,
  - appels système
  - déblocage des tâches prioritaires.
- en cas de gel, de blocage ou de terminaison de la tâche active
  - sélectionner une nouvelle tâche disponible
- demande de la tâche active, si elle souhaite relacher le CPU.

Politiques d’ordonnancement

- Politique préemptive :
  - tâche en cours d'exécution est peut être interrompue par l'ordonnanceur pour une autre tâche en attente

- La file d’attente
  - FIFO de tâches en attente
  - Une tâche se termine on passe à la suivante
  - Tâche bloquée ou lâche volontairement le CPU ?
    - fin de queue et activation tâche suivante

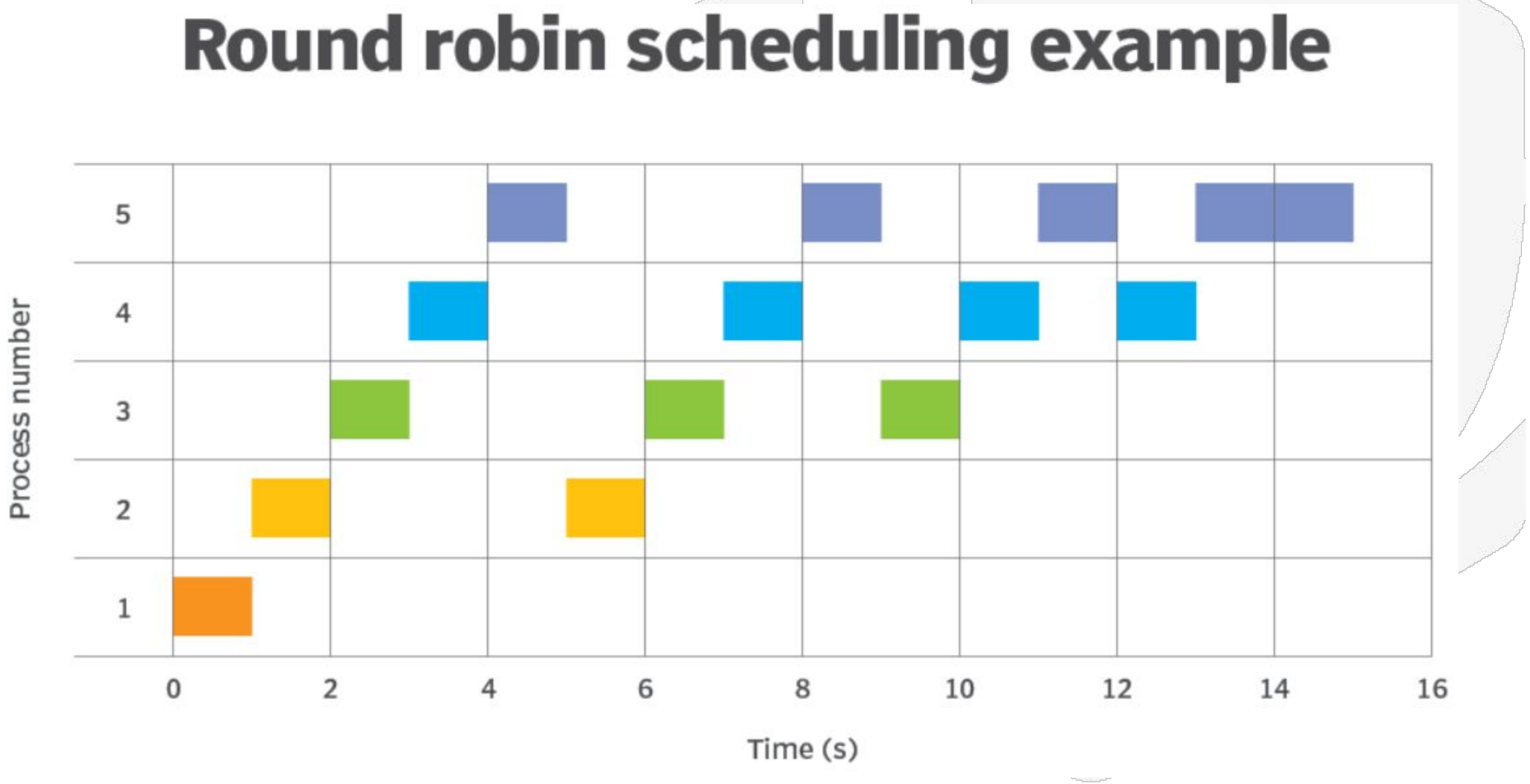
  - Pas optimal
  - Non préemptif
  - tâches courtes peuvent attendre longtemps avant d'être exécutées.

Politiques d’ordonnancement

- Avantage aux tâches rapides
  - nécessaire de connaitre la durée des tâches en avance
  - remplacement de la tâche courante dès l'apparition d'une tâche plus courte

- Le tourniquet (*round-robin*)
  - quantum de temps :
    - tâche a le droit de s'exécuter sans être interrompue par l'ordonnanceur
  - expiration du quantum :
    - tour de la tâche suivante d'obtenir le CPU (nouveau quantum)
  - tâches en attente gérées par FIFO
  - changement de tâche forcé en fin de quantum préemptif

Round-robin


Round robin scheduling example
Process number
5
4
3
2
1
0
2
4
6
8
10
12
14
16
Time (s)

Politiques d'ordonnancement

- temps partagé universel :
  - politique par défaut des systèmes UNIX
  - à chaque tâche une priorité générée par :
    - gentillesse auprès des autres tâches (*nice*) ;
    - temps CPU déjà consommé et/ou passé à attendre le CPU.
  - choix de la tâche dont la priorité dynamique est la plus élevée.
  - tâche de priorité dynamique supérieure à la tâche active ?
    - changement de tâche prématuré.

Les processus sous Linux

- le processus initial **init** (process id 1) est à la racine des processus

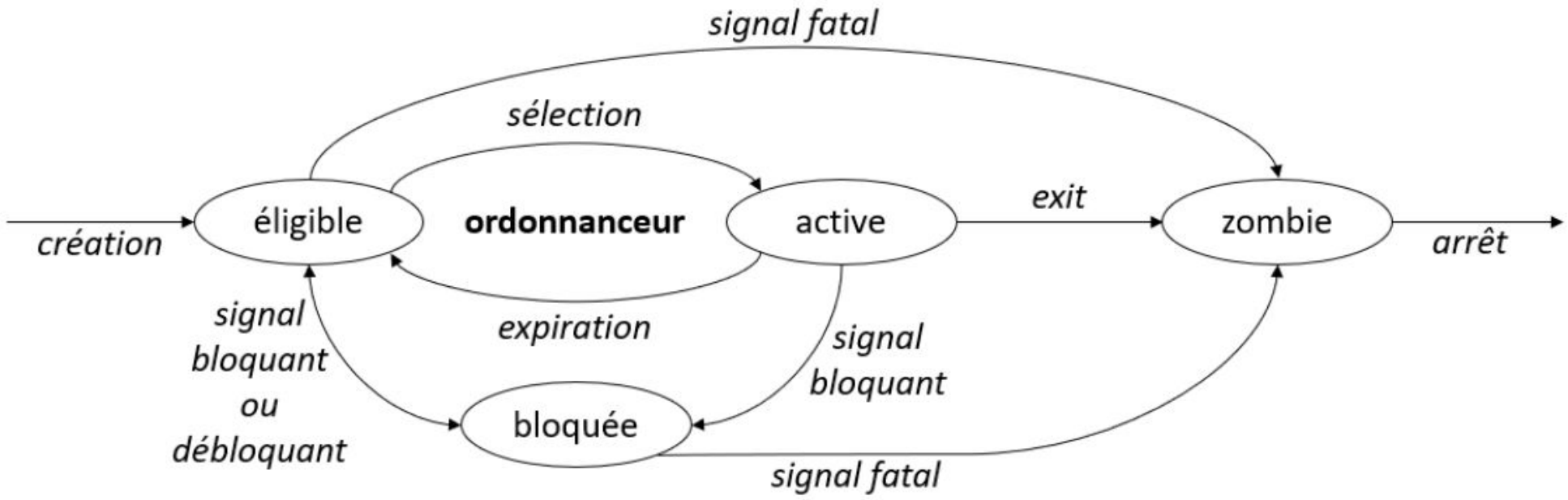
- un processus crée un nouveau processus (fork)
  - ancien processus père
  - nouveau processus fils
  - hiérarchie de processus

- processus avec enfants se termine
  - Rattachement au processus **init** (pas d'orphelin)

Les types de tâche

- tâche active
  - en cours d’éxecution.
- éligible
  - en cours d'exécution et en attente de ressources ;
  - classées en fonction de critères de priorité.
- Bloquée
  - tâche ayant exécuté un appel système bloquant (demande d'une ressource) ;
  - état interruptible ;
  - Réception d’un signal.
- Zombies
  - tâches terminées dont l'arrêt et le code de terminaison n’ont pas été acquittés.

Etats des processus



Commandes Shell

- ps
  - Afficher des informations sur tout ou partie de la liste des processus du système
  - *ps auxw*
  - *ps afx*
  - *ps axl*
- pstree
  - arbre des liens de parenté entre processus
- top
  - visualiser les informations des processus
  - classer les processus

Propriétés des processus

- contexte CPU :
  - État instantané du CPU au cours de l'exécution du programme
  - numéro d'identifiant (PID)
  - pointeur sur le contexte du père
  - identités : *uid* et *gid* réels, effectifs
  - état vis-à-vis de l'ordonnanceur
  - …
- contexte utilisateur :
  - mémoire occupée par le programme (code, données, tas, bibliothèques, pile).
- contexte noyau
  - géré par le noyau de l'OS.

Gestion des identités

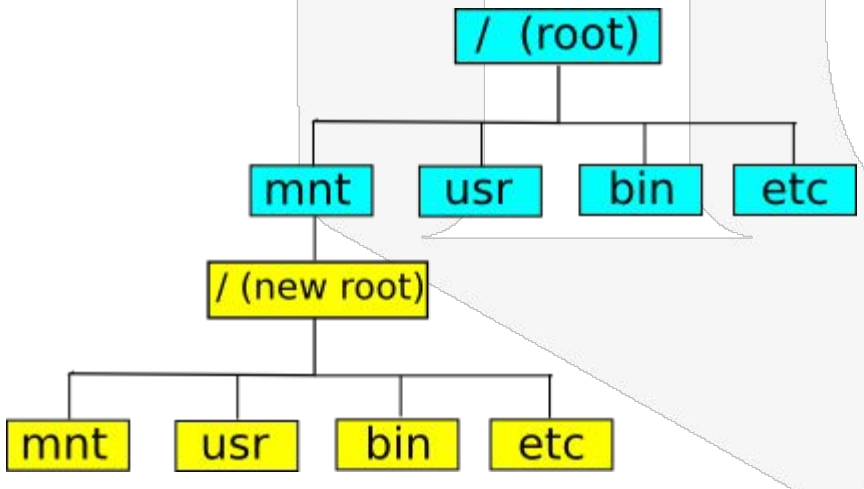
- chaque processus dispose de trois identités :
  - uid réel (uid)
    - identité de l'utilisateur qui a lancé l'application
  - uid effectif (euid)
    - privilèges effectifs du processus
  - uid sauvegardé (suid)
    - privilèges effectifs qui ont été obtenus puis abandonnés provisoirement.

- descente de privilèges
  - opération pour un processus à perdre une partie de ses privilèges.
  - processus lancé en tant que root peut changer d'identité
  - irréversible
  - *Setuid*

- *gestion des groupes (gid, egid, sgid)*
  - *Même méthode !*

Chroot

- fonction de sécurité intéressante ;
- permet d'enfermer un processus dans une cage minimaliste pour le système de fichiers.



Création d'un nouveau processus

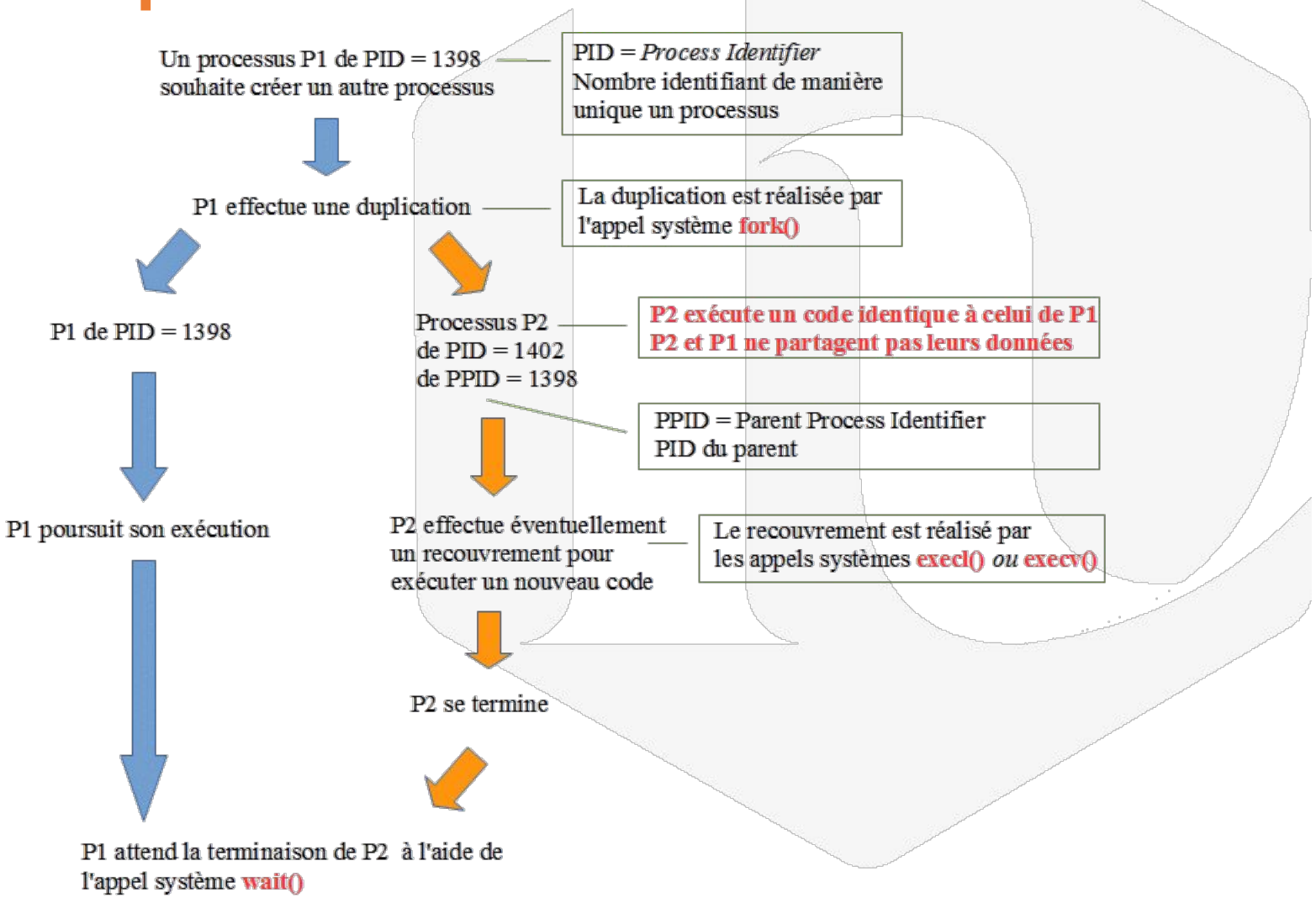
- lorsqu'un processus utilisateur se duplique via un appel à la fonction fork
  - Lancement d'un nouvel exécutable
    - 1 – vérification des permissions X
      - en fonction du type d'exécutable
    - 2 – changement les identités effectives du processus appelant
    - 3 – initialisation des espaces mémoire (code, données, tas, bibliothèques, pile) et threads
    - 4 – conservation des paramètres (pid et ppid,…)
    - 5 – conservation des table des fichiers ouverts et verrous

- à l'initiative du noyau
  - lancer le premier processus init
  - gérer le branchement chaud d'un périphérique sous Linux
  - …

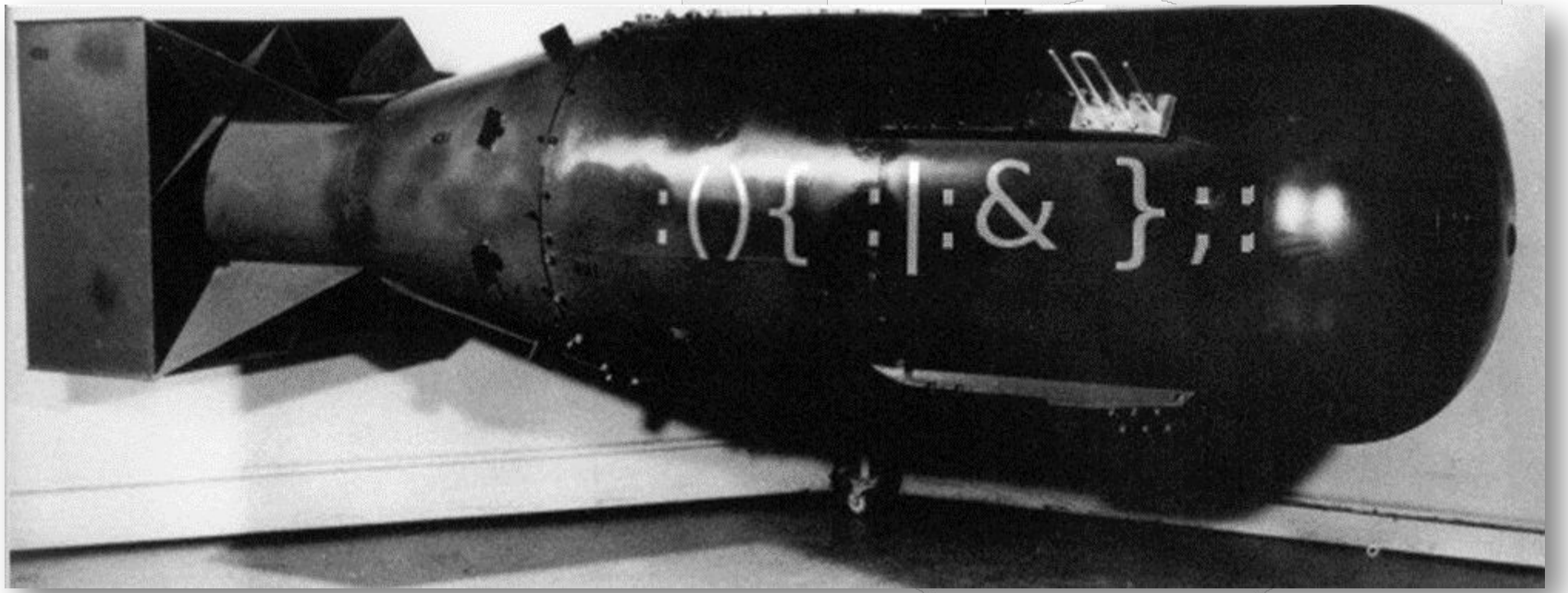
Fin d'un processus

- appel **exit**
  - demander au noyau de le terminer en précisant un code de retour
- père récupère le code de retour via un appel à une fonction **wait**.

Cycle d'un processus



La fork bomb

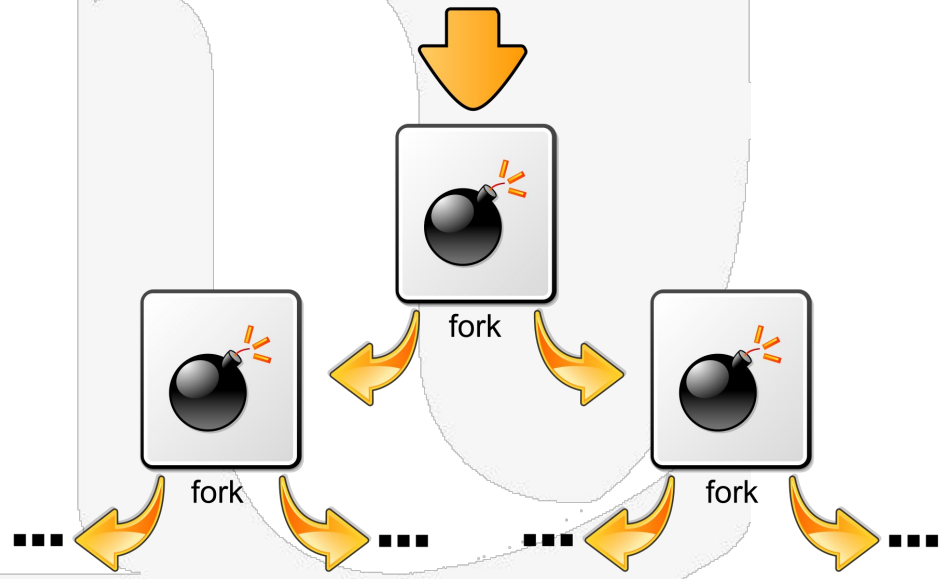


La fork bomb

- créer un grand nombre de processus très rapidement
  - fonction fork
  - saturer l'espace disponible dans la liste des processus
  - aucun nouveau programme ne peut démarrer

- :(){ :|:& };:

  - :() définit une fonction nommée :
  - { :|:& } est le corps de la fonction
    - la fonction s'appelle elle-même (:),
    - redirige la sortie à l'aide d'un pipe (|)
    - cache le nouveau processus en fond avec &
    - La fonction : s'appelle récursivement à l'infini.



- while($true) {
  Start-Process powershell.exe -ArgumentList "-NoExit", "Get-ChildItem -Recurse C:";
  Invoke-Expression -Command 'while($true) {Start-Process powershell.exe -ArgumentList « -NoExit", "Get-ChildItem -Recurse C:"}';
}