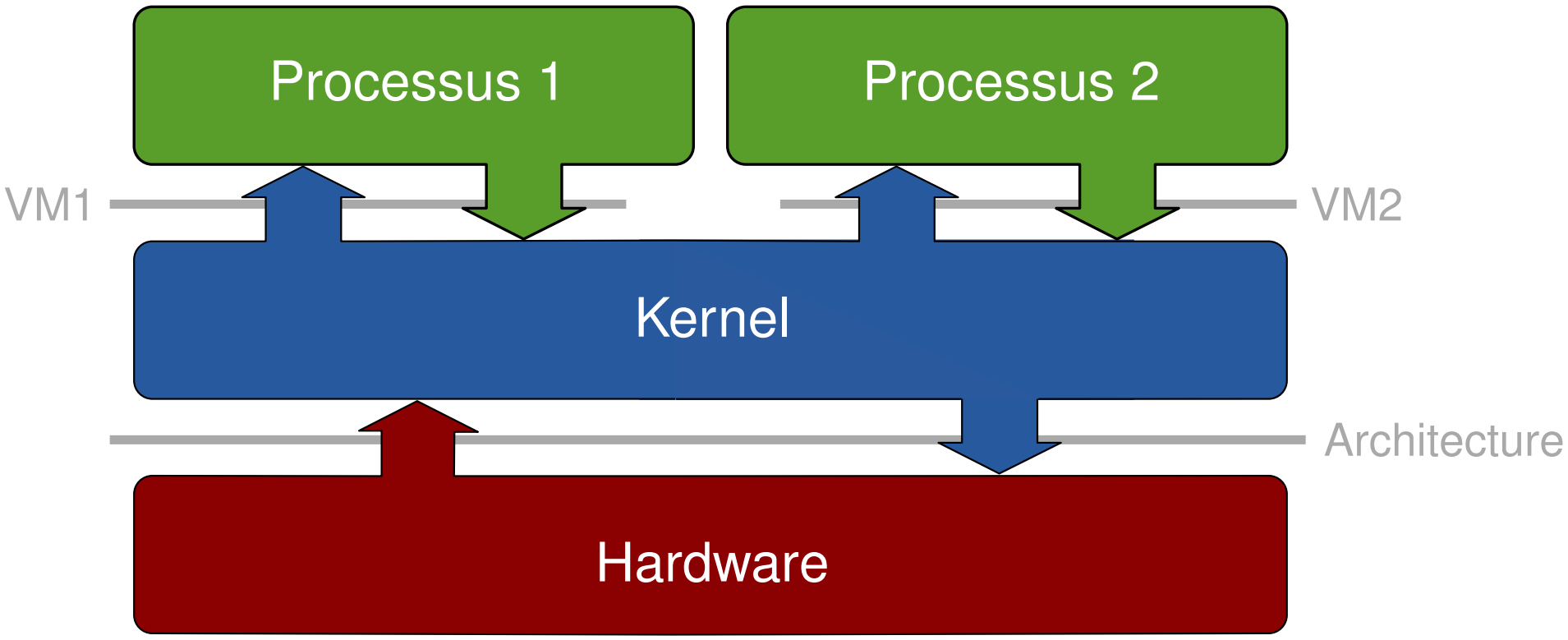
Partage du temps et ordonnancement

Guillaume Salagnac

Insa de Lyon – Informatique

2020–2021

Résumé des épisodes précédents: noyau vs userland



Le processus vu comme une «machine virtuelle»

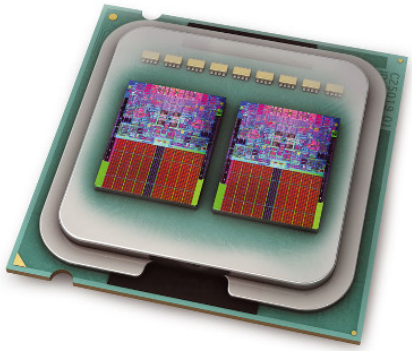
- un processeur pour moi tout seul: «CPU virtuel»
- une mémoire pour moi tout seul: «mémoire virtuelle»

Plan

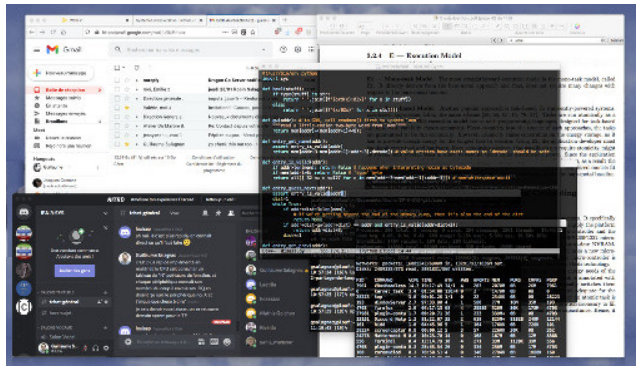
Définitions: Multitasking vs Multiprocessing

Multitraitement, en VO *Multiprocessing*, *multi-core computing*

Utilisation simultanée de plusieurs CPU dans un même système



VS



Multiprogrammation = multitâche

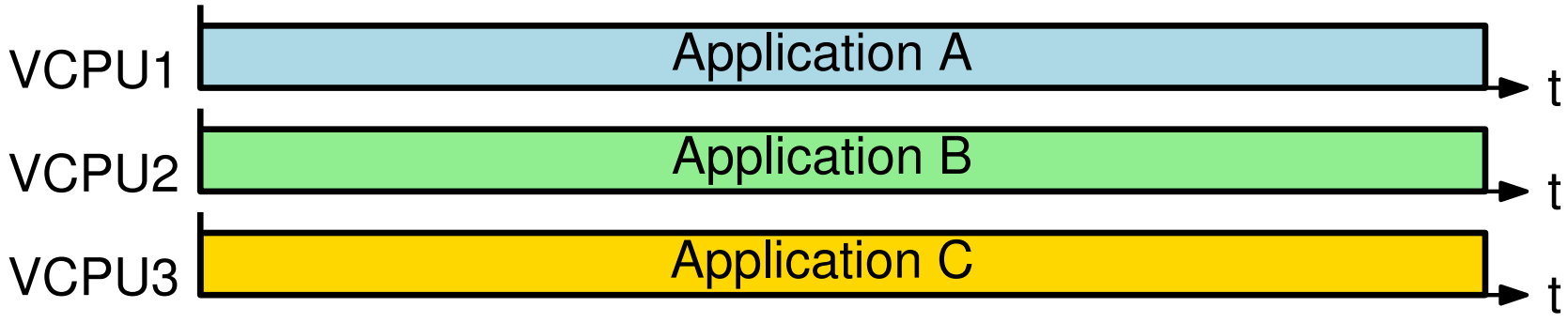
ou en VO multiprogramming = multitasking

Faculté d'exécuter plusieurs programmes «à la fois»

- en général: nombre de CPU $<<$ nombre de processus

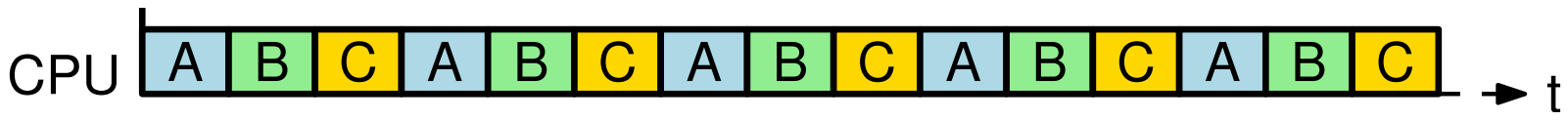
Pseudo-parallélisme et entrelacement

Abstraction = 1 VCPU / application



VS

Implementation = CPU time-sharing



Remarque: L'entrelacement doit être assez fin pour ne pas se voir

Le «Degré de multiprogrammation»

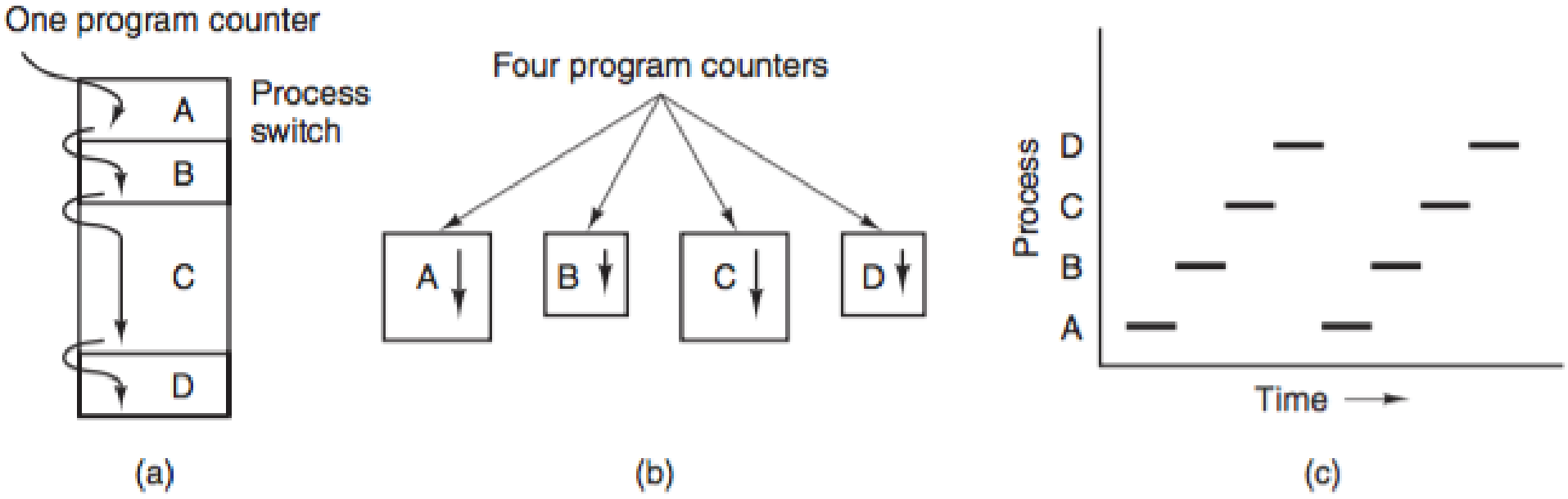


Figure 2-1. (a) Multiprogramming four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at once.

Définition: degré de multiprogrammation

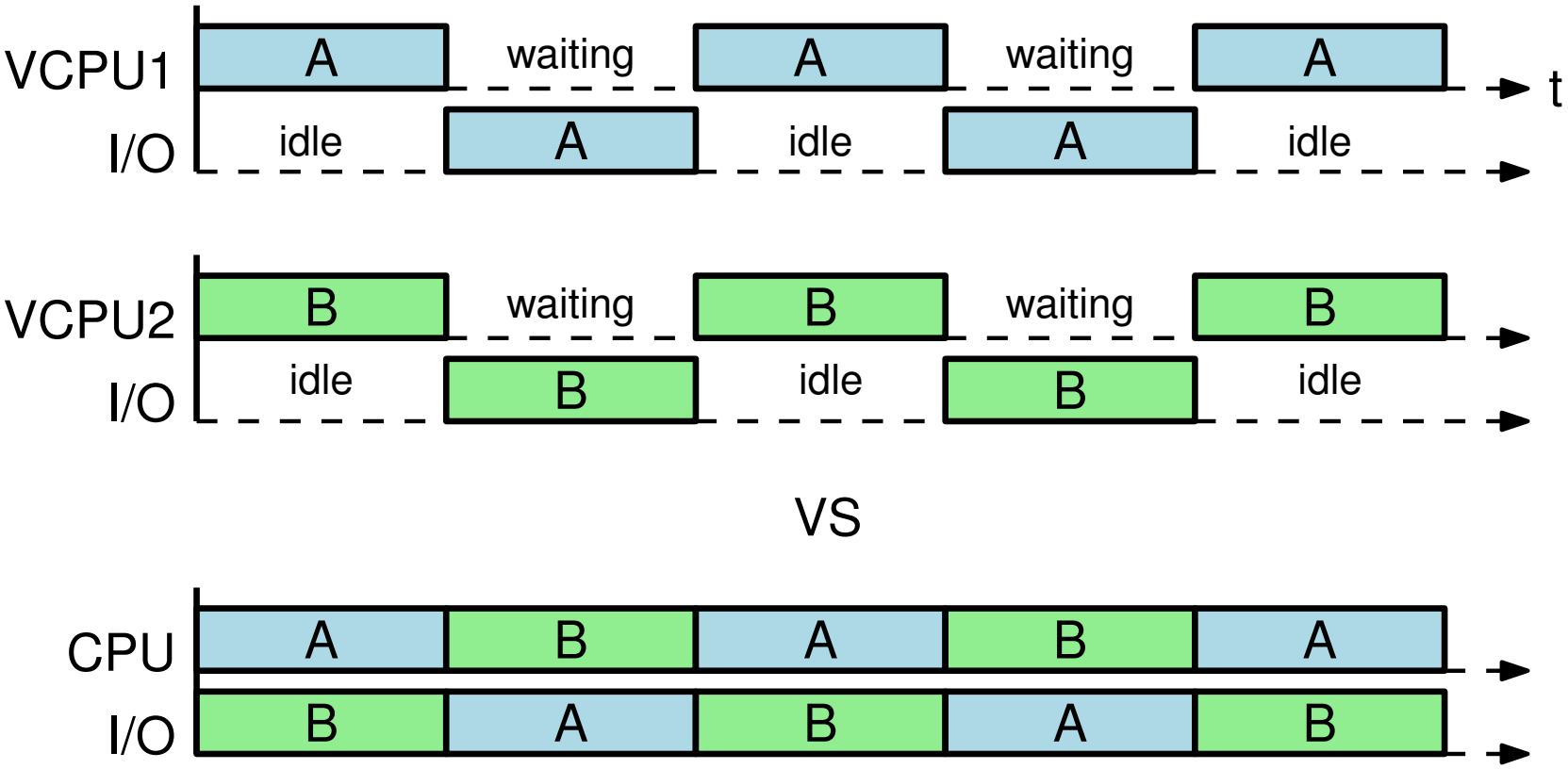
nombre de processus actuellement chargés en mémoire

source: Tanenbaum. *Modern Operating Systems (4th ed, 2014)*. page 87

Pourquoi la multiprogrammation ?

Loi empirique

L'exécution d'un programme alterne entre des phases de calcul (en VO CPU burst) et des phases d'entrées-sorties (I/O burst)



La multiprogrammation: remarques

Vous avez dit «des phases d'entrées-sorties» ?

- latence d'accès au matériel: disque, réseau...
- lenteur de l'utilisateur d'un programme interactif
- synchronisation avec d'autres programmes

Mauvaise solution: attente active (polling)

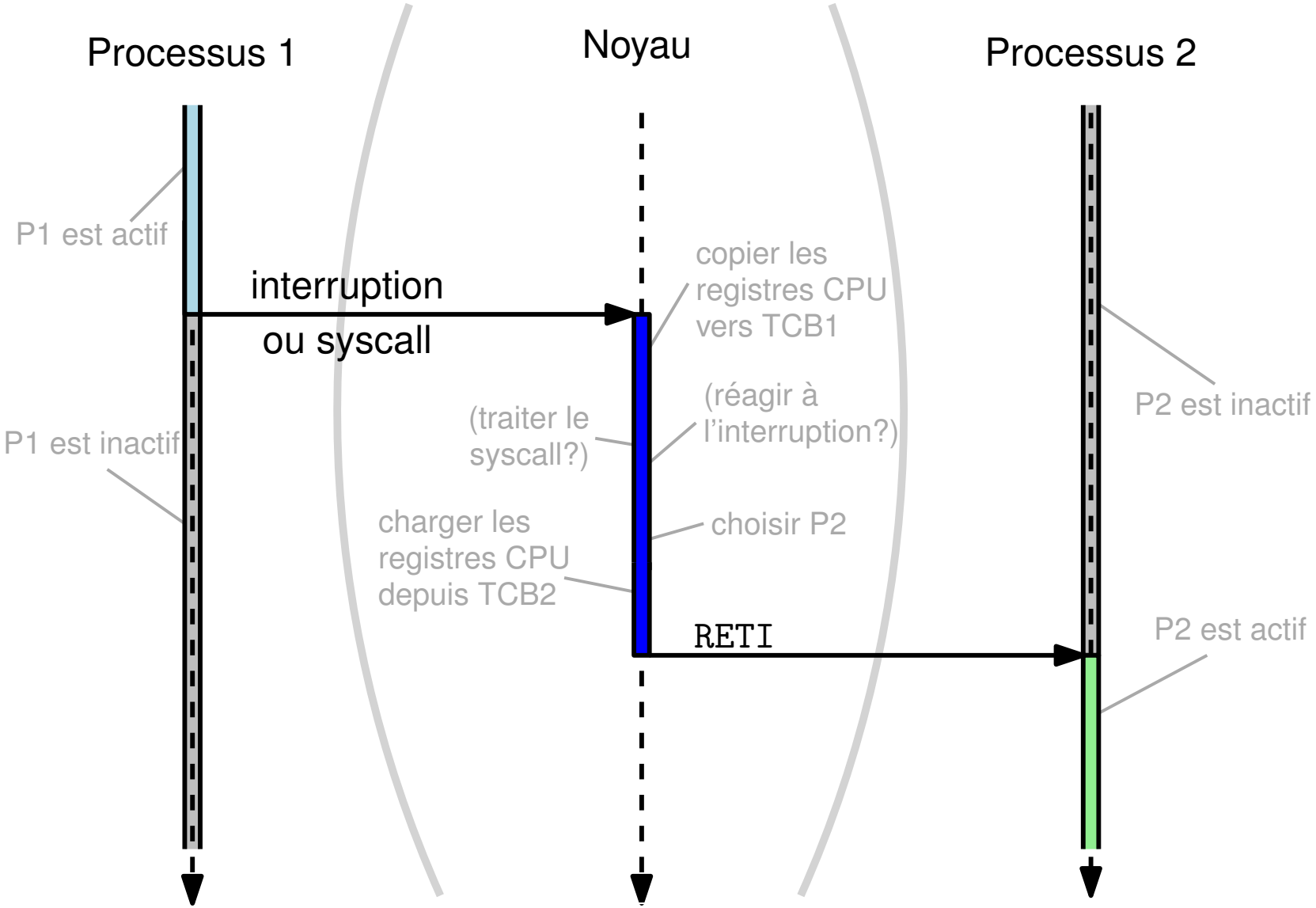
- difficile pour le programmeur d'application
- temps processeur gâché à attendre

Bonne solution: attente passive

- plus facile pour le développeur
- meilleur taux d'utilisation du CPU
  - masquage (= recouvrement) des latences

▶ besoin d'un mécanisme pour se partager le processeur

Commutation de contexte, ou en VO context switch



Context switch: remarques

- dispatcher = implémentation du context switch
  - exécuté très souvent ▶ doit être bref (dispatch latency)
- scheduler = choix du processus à qui rendre la main
  - possible que P2 $=$ P1, par exemple `gettimeofday()`...
  - possible que P2 $\neq$ P1, par ex. `read()` ▶ appel bloquant

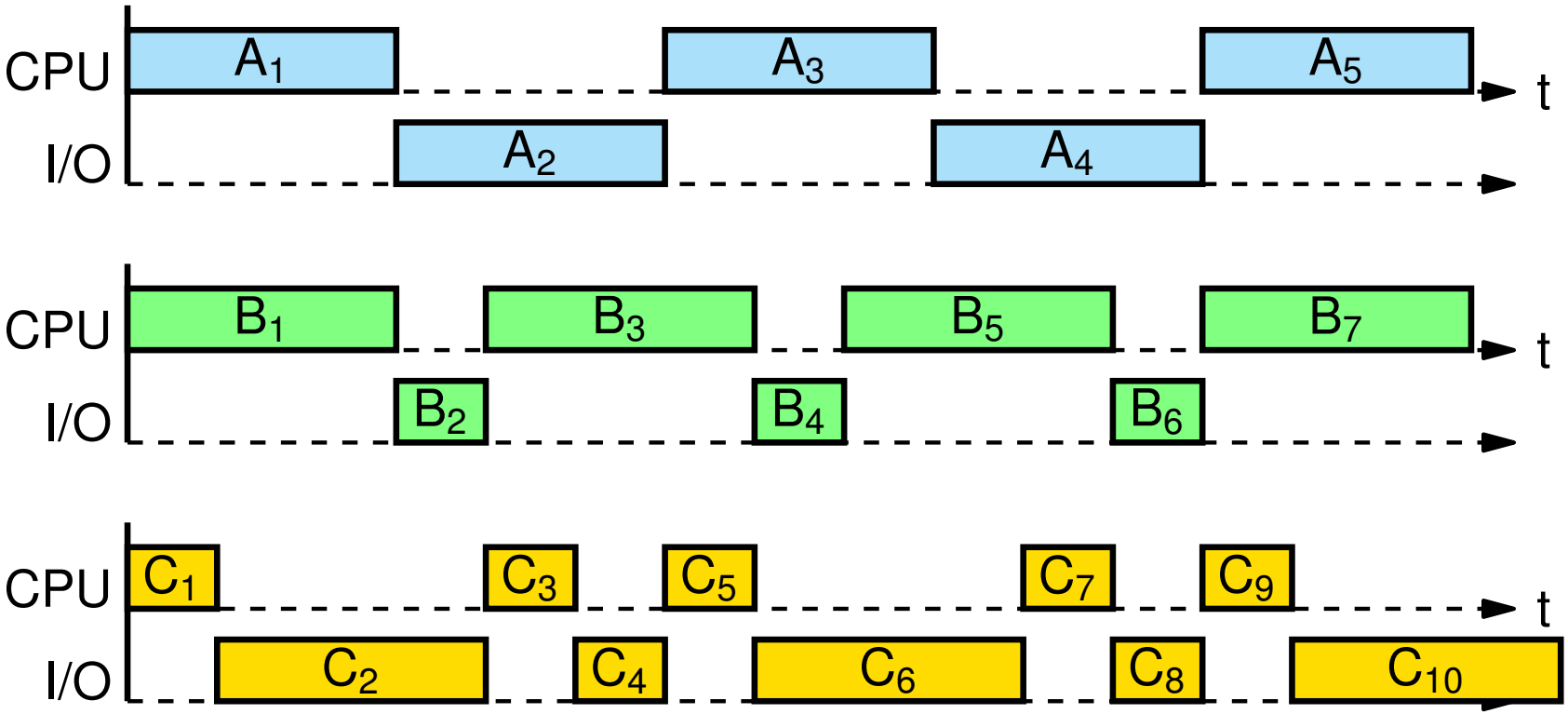
Structures de données du noyau:

- Process Control Block = PCB
  - représente un processus et son état d'exécution (données...)
  - contient PID, fichier exécutable, permissions, etc (priorité...)
  - contient un TCB
- Thread Control Block = TCB
  - représente un VCPU, aussi appelé contexte d'exécution
  - contient une copie du contenu du CPU: registres, PC, SR...

vocabulaire: (dans ce chapitre) «processus» == «thread»

Plan

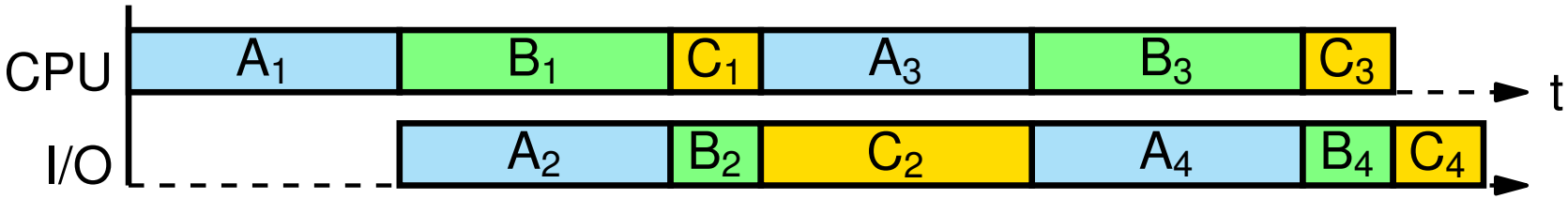
Quel processus exécuter après un context switch?



Question: étant donnés ces trois processus et un seul CPU, comment les organiser dans le temps «au mieux» ?

Ordonnancement naïf

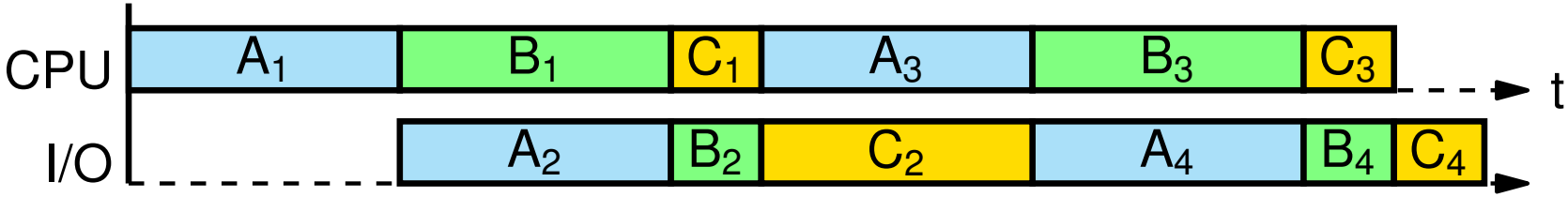
Première idée: exécuter A, B et C à tour de rôle



- plutôt bien pour A, moins bien pour B, et médiocre pour C

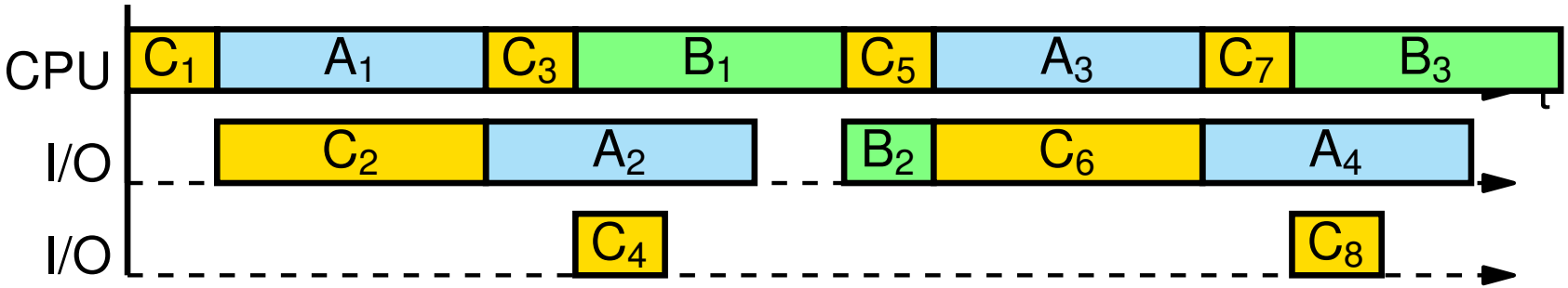
Ordonnancement naïf

Première idée: exécuter A, B et C à tour de rôle



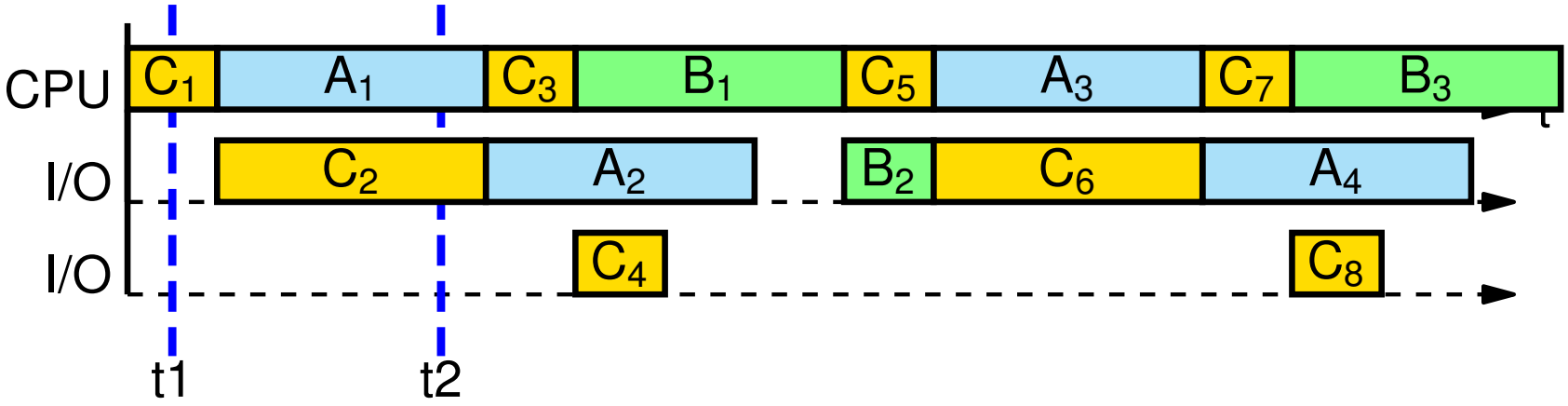
- plutôt bien pour A, moins bien pour B, et médiocre pour C

Deuxième idée: exécuter C aussi souvent que possible



- beaucoup mieux pour C, et presque aussi bien pour A et B

Tous les processus ne sont pas en permanence candidats à l'exécution



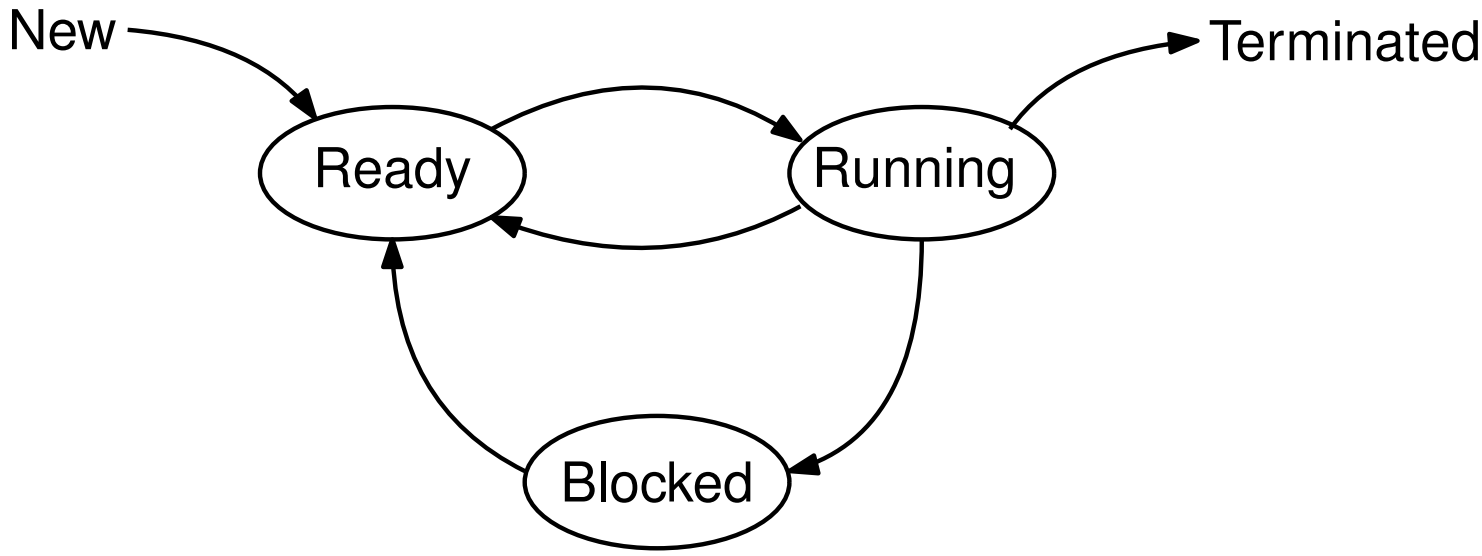
à l'instant t1:

- C est en train de s'exécuter
- A et B attendent de pouvoir s'exécuter

à l'instant t2:

- A est en train de s'exécuter
- C attend que son opération d'entrée-sortie se termine
- B attend de pouvoir s'exécuter

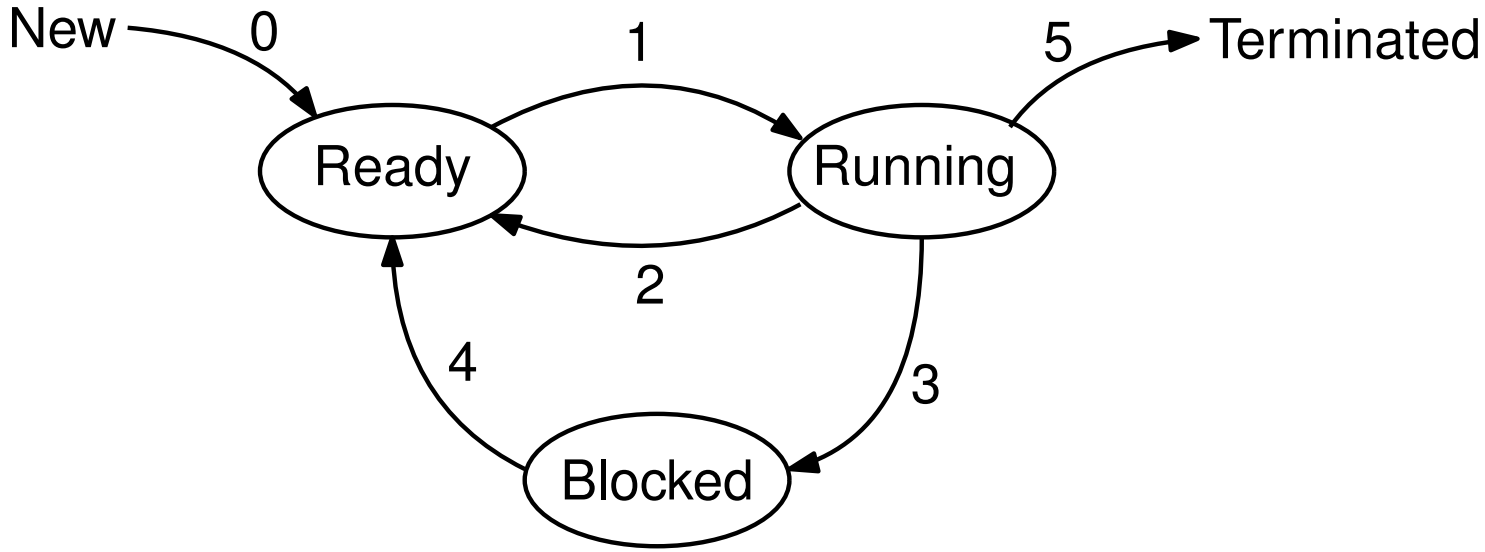
Diagramme état-transitions (1)



État du processus (mémorisé dans son Process Control Block)

- New: en cours de création par le noyau
- Running = actif: actuellement exécuté par le processeur
- Ready = activable: attend qu'on lui donne la main
- Blocked = endormi = suspendu: en attente d'un évènement
- Terminated = fini: en cours de nettoyage par le noyau

Diagramme état-transitions (2)



Transitions:

0 le noyau a fini de créer le processus

1 le dispatcher charge le processus sur le CPU

2 une IRQ / un syscall interrompt l'exécution du programme

3 le programme fait un syscall bloquant
- par ex: entrée-sortie `read()`, délai passif `sleep()`, etc...

4 l'évènement attendu se produit
- par ex: donnée disponible, délai écoulé, etc...

5 le programme se termine (volontairement ou non)

Problème d'ordonnancement: formulation

Ordonnancement CPU, ou en VO process scheduling

- étant donné K processus ready = prêts à s'exécuter,
  - et connaissant «leurs caractéristiques»
- étant donné $N \geqslant 1$ processeurs disponibles,

décider quel processus exécuter sur chaque processeur

Remarque: à quels instants cette question se pose-t-elle ?

- lors d'une transition running $\rightarrow$ blocked (3), par ex. `sleep()`
- lors de la terminaison (5) d'un processus
- lors d'une transition blocked $\rightarrow$ ready (4)
- lors d'une transition running $\rightarrow$ ready (2)
  - par ex. sur réception d'une interruption du system timer
- lors de la création (0) d'un nouveau processus

Ordonnancement préemptif vs coopératif

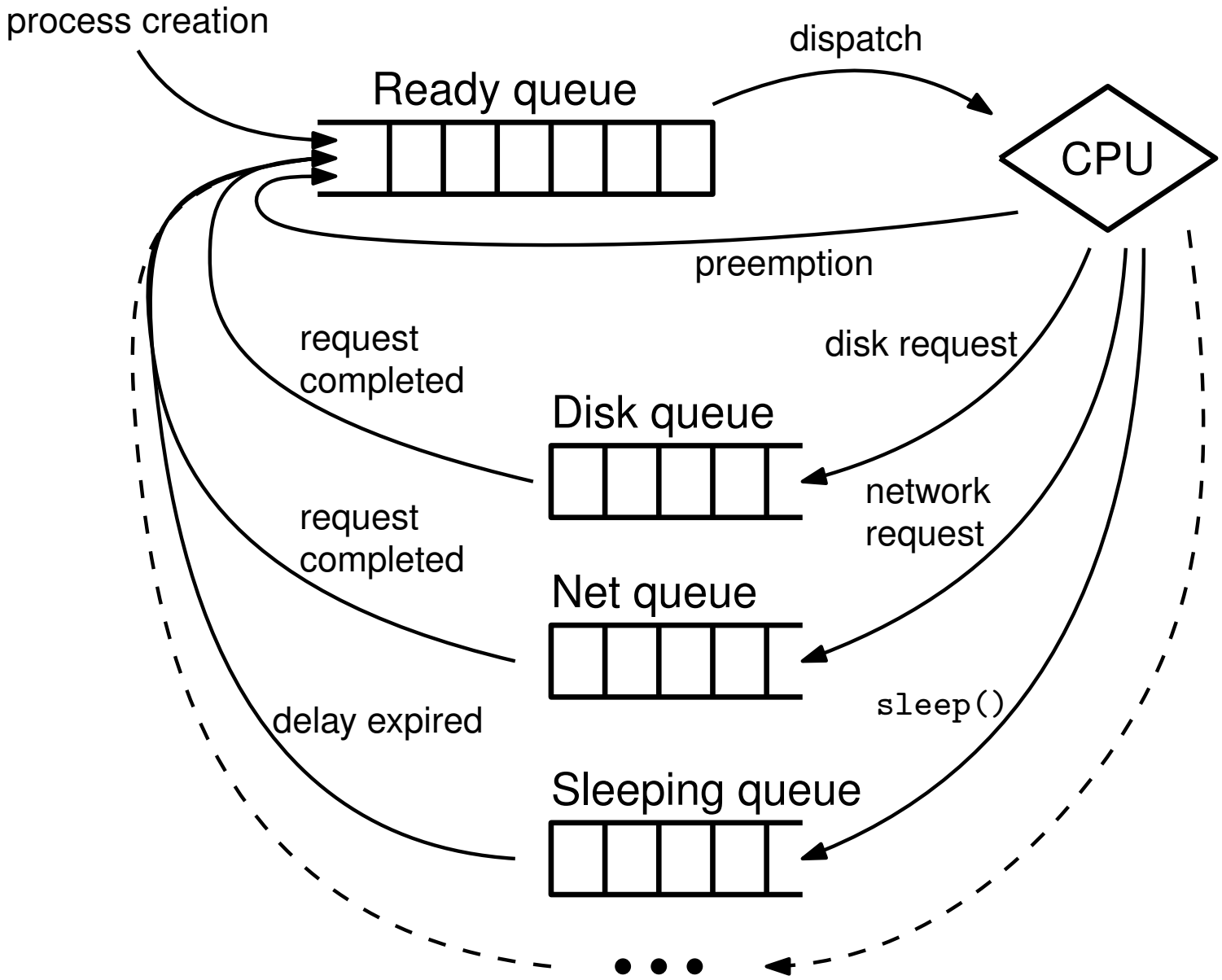
Ordonnanceur non-préemptif: appelé seulement sur (3) et (5)

- les applications rendent explicitement la main au noyau
  - appels système bloquants
  - + un appel système `yield()` pour lâcher le CPU
- plus efficace mais suppose de faire confiance aux applications

Ordonnanceur préemptif: sur (3), (5) et aussi (0), (2) et (4)

- permet au noyau de garder le contrôle de la machine
  - interruptions de timer régulières pour assurer la préemption
- coûteux mais tous les OS grand public font ça

Organisation des PCB en files d'attente



Organisation des PCB en files: remarques

Les PCB des processus prêts forment la Ready Queue

- aussi appelée Run Queue

Rôle du scheduler: choisir un PCB parmi la Ready Queue

Processus bloqués ▶ transférés dans une autre file d'attente

- une Device Queue par périphérique d'entrées-sorties
- une file d'attente pour les processus endormis
- ... une file pour chaque autre raison d'être Blocked

Plan

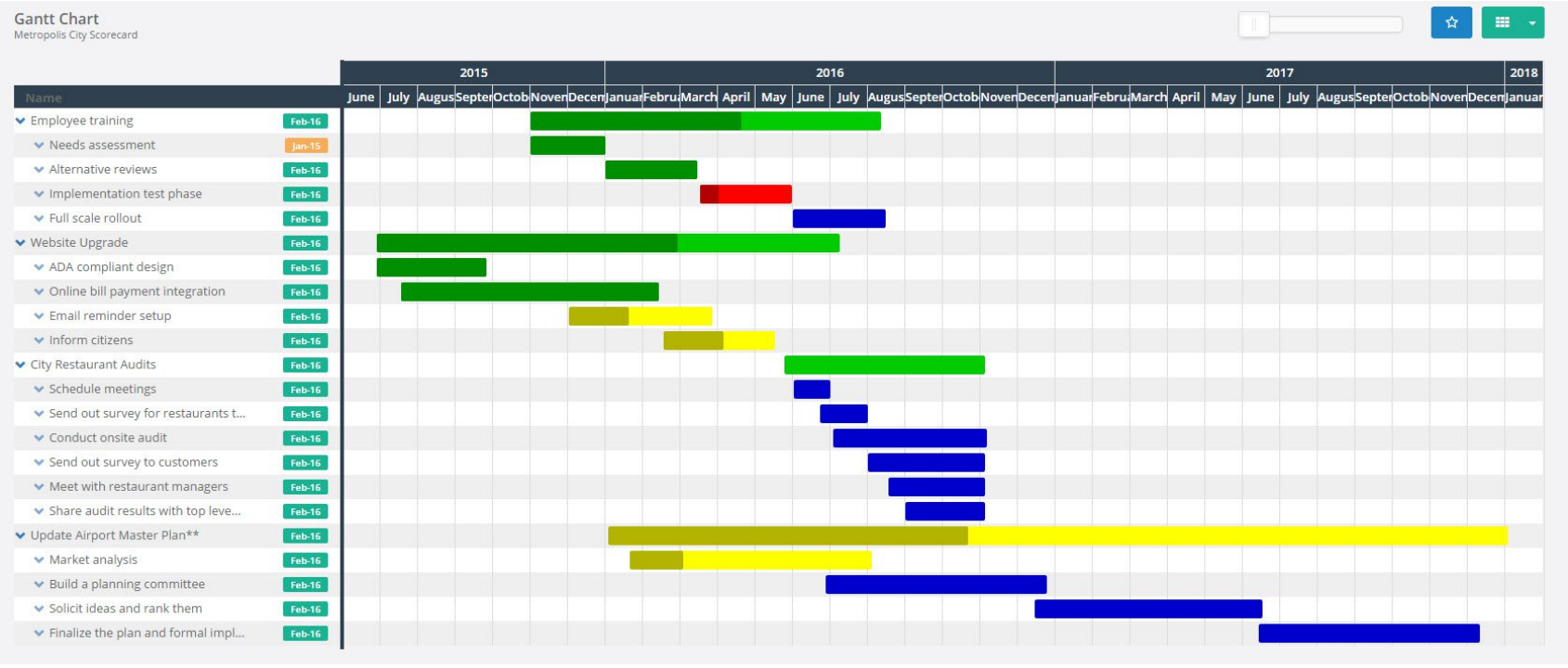
1. Introduction: rappels sur la notion de processus
2. Multitâche par commutation de contextes d'exécution
3. Ordonnancement: formulation du problème
4. Ordonnancement: les stratégies classiques
5. Évaluation de politiques d'ordonnancement

Ordonnancement = planification de tâches

Ordonnancement a priori: projets, usine, etc

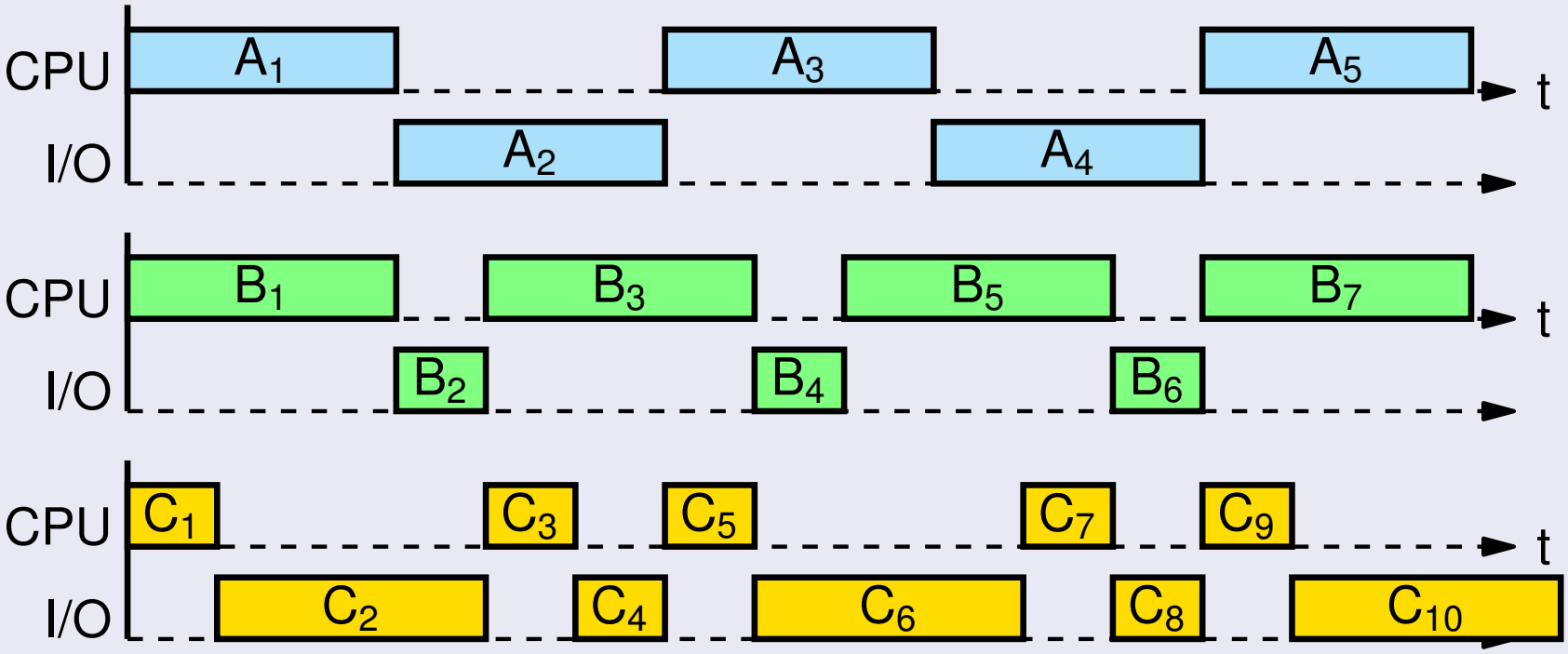
Entrées: ensemble de «tâches» avec durées et dépendances
(+ ensemble de «ressources» disponibles)

Sorties: une date de début pour chaque tâche
(+ affectation des ressources)



Ordonnancement a priori vs processus infinis

Vision omnisciente



VS

Vision de l'ordonnanceur à t=0

La Ready Queue contient A, B et C. Le CPU est libre.

- comment décider qui exécuter ? rien pour les départager

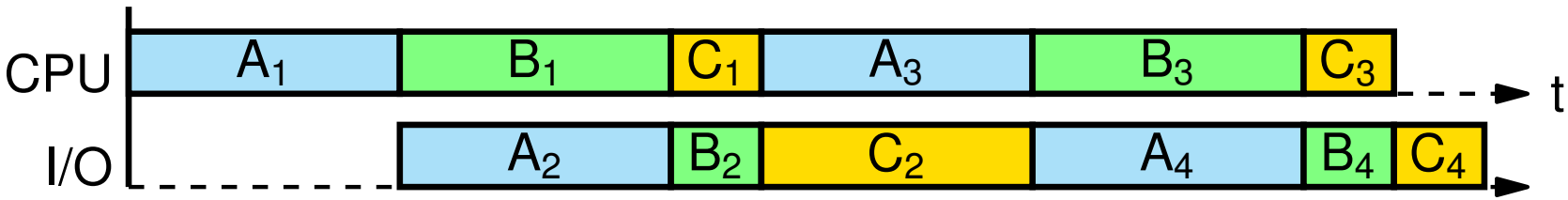
Ordonnancement FCFS: First Come First Served

aussi appelé FIFO (First In First Out)

Politique d'ordonnancement FCFS: principe

choisir la tâche qui est arrivée le plus tôt dans la ready queue

Dans notre exemple:



Remarques:

- inspiré par les files d'attente FIFO de la vie réelle
- ordonnancement non préemptif
- *convoy effect*, en VF effet de convoi, effet d'accumulation
  - «petites» tâches risquent d'être désavantagées, par ex. C
- plutôt équitable ; aucun risque de famine

Le risque de famine, en VO starvation

Définition: famine = privation

Situation dans laquelle une tâche, bien que prête, se retrouve à attendre indéfiniment avant de pouvoir s'exécuter

Remarques:

- ordonnanceur non-préemptif + 1 tâche infinie = famine
  - hypothèse: pas de tâches de durée infinie
- ordonnanceur préemptif + malchance = famine
  - malchance, ou malveillance (par ex. déni de service)

Risque de famine vs absence garantie de famine

en VO: non-starvation, bounded waiting

Découpage de l'exécution en «bursts» (en VF: rafales)

Hypothèse de travail:

- pour chaque processus dans la ready queue, le noyau a un moyen pour connaître la durée de sa prochaine CPU-burst

Remarque: en pratique, les «tâches» orchestrées par un scheduler d'OS sont ces CPU-bursts, et non pas les processus

Vision de l'ordonnanceur à t=0

La Ready Queue contient A_1, B_1 et C_1.

- à quelle tâche doit-on donner le processeur en premier ?

Différents goulots d'étranglement

Un processus est qualifié de *«compute-bound»* (VF «limité par le calcul») si sa performance dépend beaucoup de la vitesse du CPU

- activité = surtout des CPU-bursts

Un processus est qualifié de *«I/O-bound»* (VF «limité par les E-S») si sa performance dépend beaucoup de la vitesse des E-S

- activité = surtout des I/O-bursts

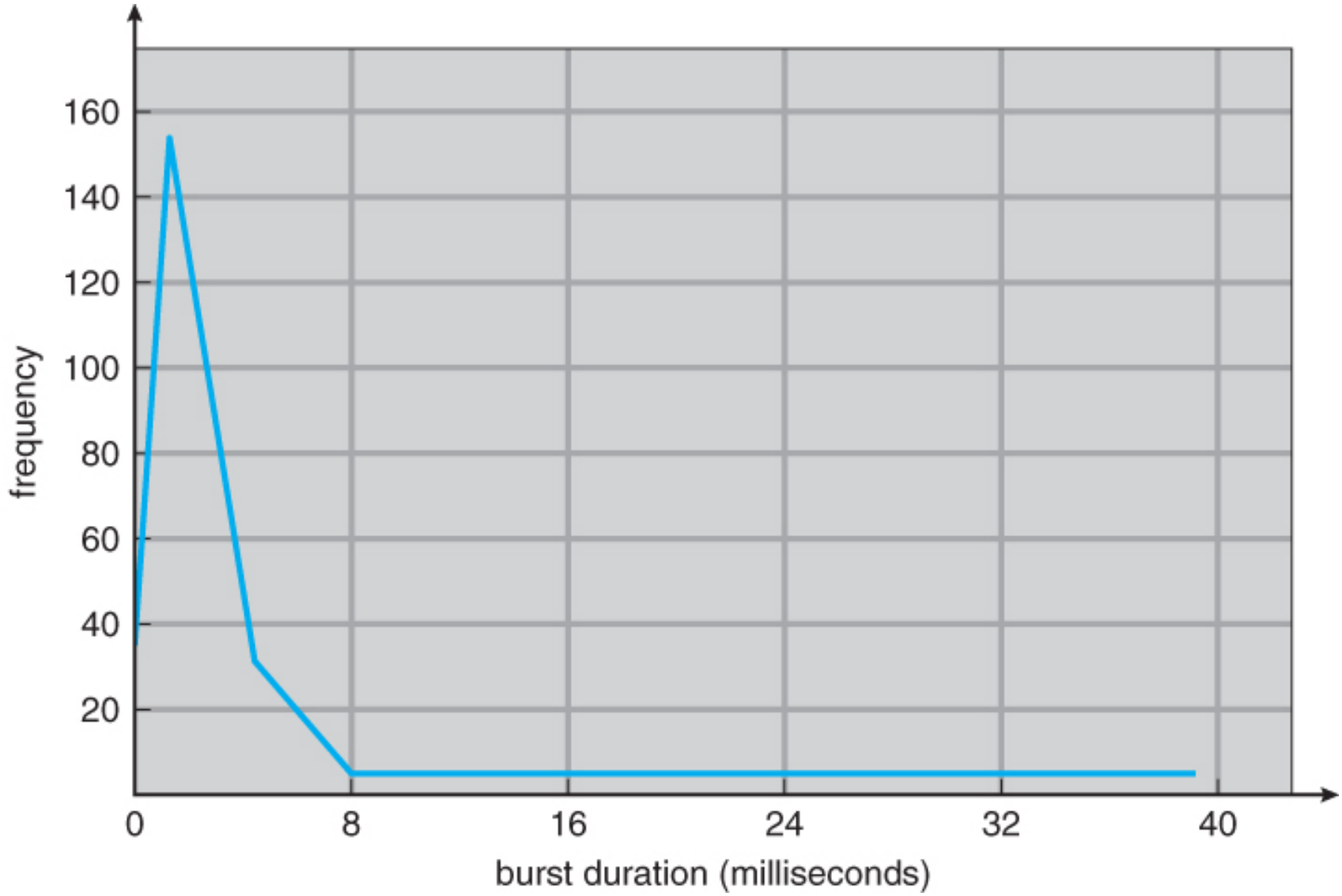
Loi empirique

En pratique, tout processus peut être considéré *soit* comme étant plutôt compute-bound *soit* comme étant plutôt I/O-bound.

Remarques:

- processus A dans notre exemple: pas réaliste
- certains processus changent de comportent à certains moments

Beaucoup de bursts courts, peu de bursts longs



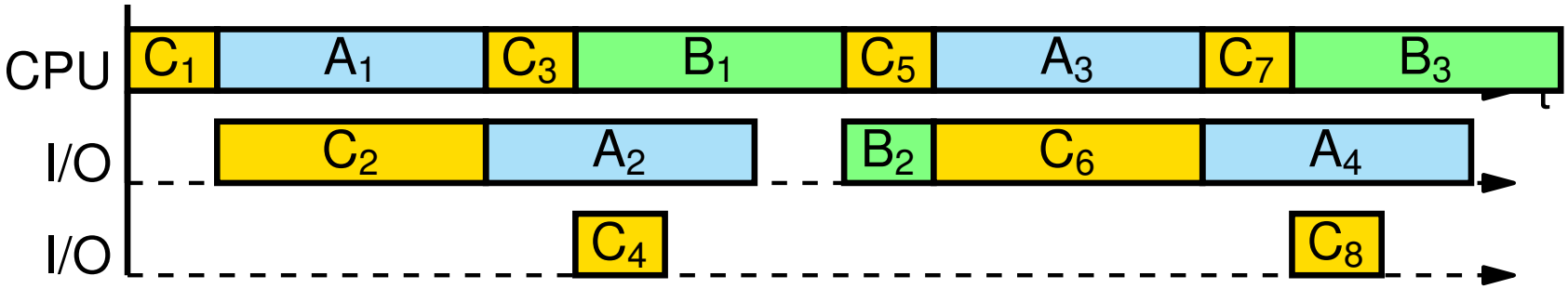
source: Silberschatz *Operating Systems Concepts Essentials (2011)*. p 177

Ordonnancement SJF: Shortest Job First

Politique d'ordonnancement SJF: principe

choisir la tâche la plus courte dans la ready queue

Dans notre exemple:



Remarques

- avantageux pour les processus IO-bound...
- ...sans être réellement pénalisant pour les CPU-bound
- mais: risque de famine

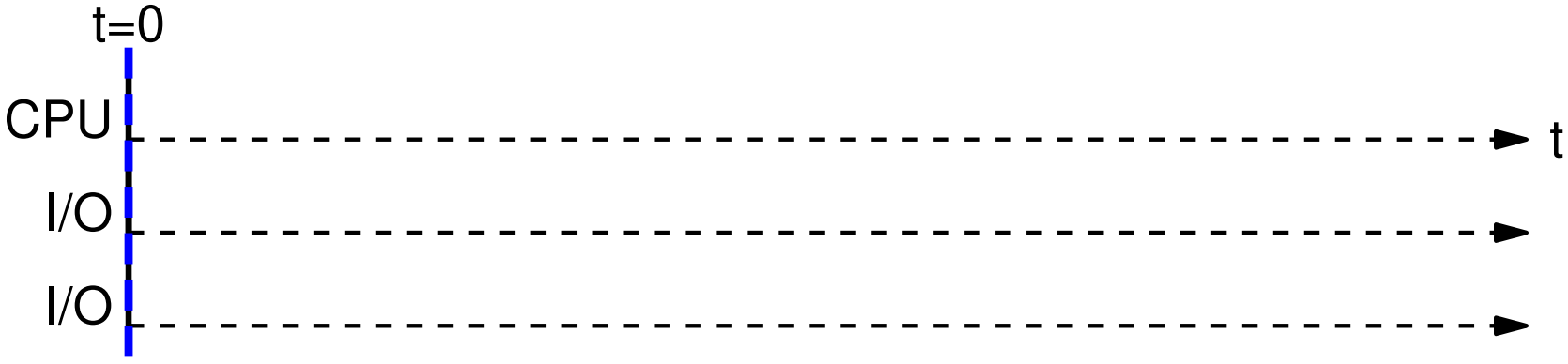
Ordo. SRTF: Shortest Remaining Time First

Politique d'ordonnancement SRTF: principe

chaque fois qu'une nouvelle tâche arrive dans la ready queue, lancer la tâche la plus courte de la ready queue

- variante *préemptive* de SJF

Dans notre exemple:



Ready Queue: A_1, B_1, C_1

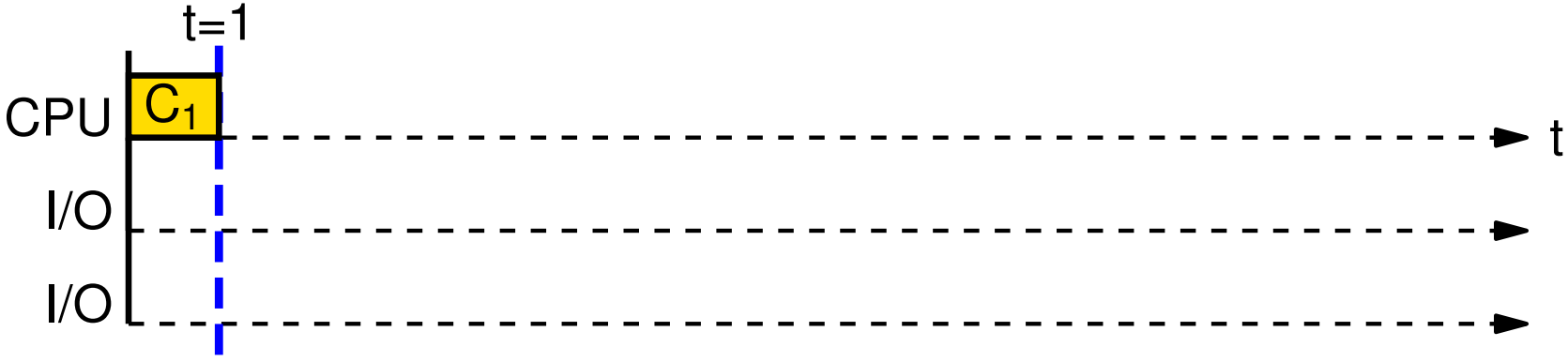
Ordo. SRTF: Shortest Remaining Time First

Politique d'ordonnancement SRTF: principe

chaque fois qu'une nouvelle tâche arrive dans la ready queue, lancer la tâche la plus courte de la ready queue

- variante *préemptive* de SJF

Dans notre exemple:



Ready Queue: A_1, B_1

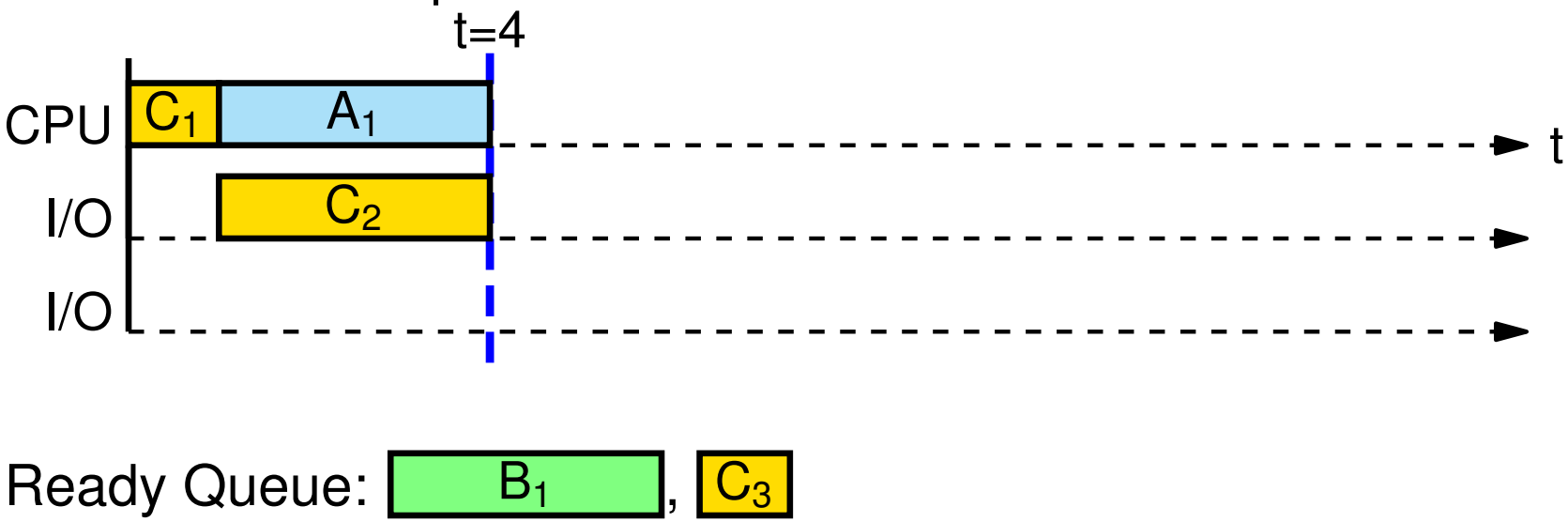
Ordo. SRTF: Shortest Remaining Time First

Politique d'ordonnancement SRTF: principe

chaque fois qu'une nouvelle tâche arrive dans la ready queue, lancer la tâche la plus courte de la ready queue

- variante *préemptive* de SJF

Dans notre exemple:



Ready Queue: B_1, C_3

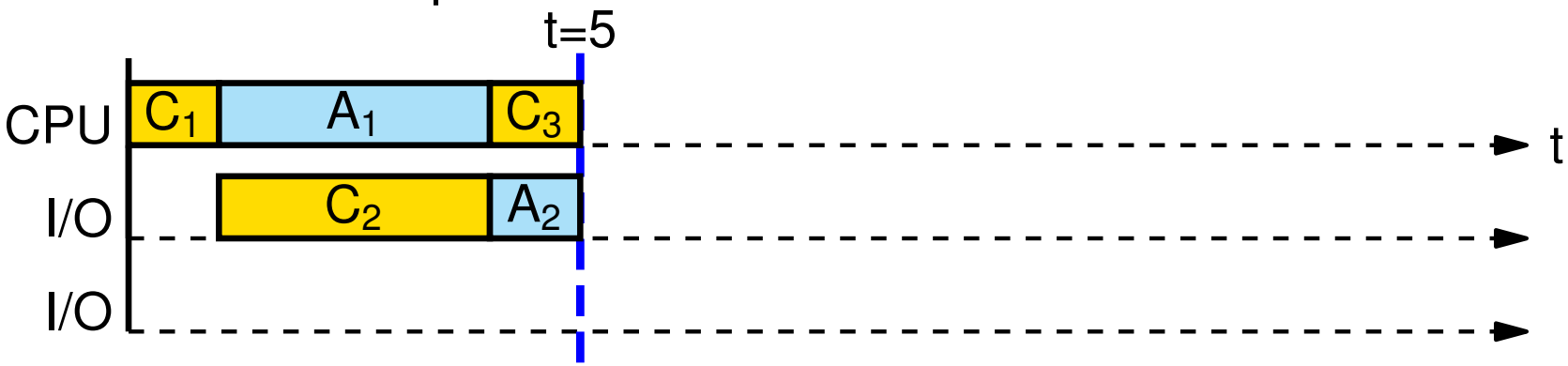
Ordo. SRTF: Shortest Remaining Time First

Politique d'ordonnancement SRTF: principe

chaque fois qu'une nouvelle tâche arrive dans la ready queue, lancer la tâche la plus courte de la ready queue

- variante *préemptive* de SJF

Dans notre exemple:



Ready Queue: B_1

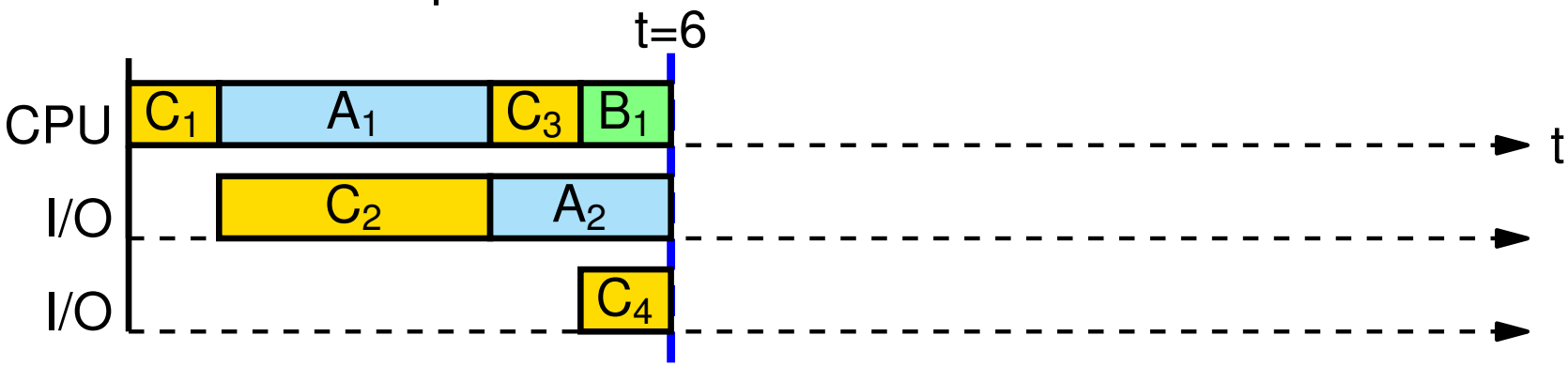
Ordo. SRTF: Shortest Remaining Time First

Politique d'ordonnancement SRTF: principe

chaque fois qu'une nouvelle tâche arrive dans la ready queue, lancer la tâche la plus courte de la ready queue

- variante *préemptive* de SJF

Dans notre exemple:



Ready Queue: B_1, C_5

- Préemption de la tâche active (B_1) au profit de la tâche C_5

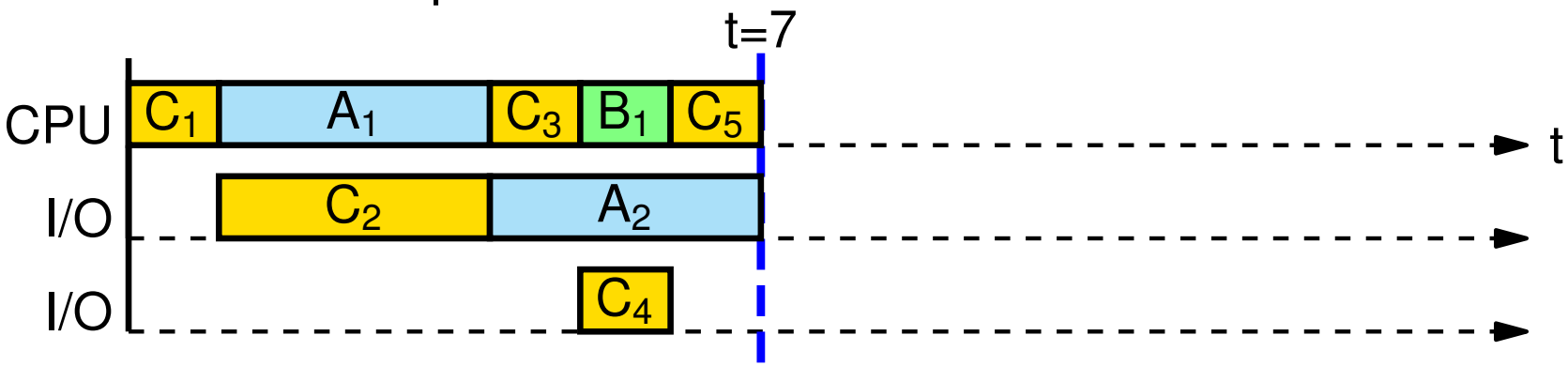
Ordo. SRTF: Shortest Remaining Time First

Politique d'ordonnancement SRTF: principe

chaque fois qu'une nouvelle tâche arrive dans la ready queue, lancer la tâche la plus courte de la ready queue

- variante *préemptive* de SJF

Dans notre exemple:



Ready Queue: A_3, B_1

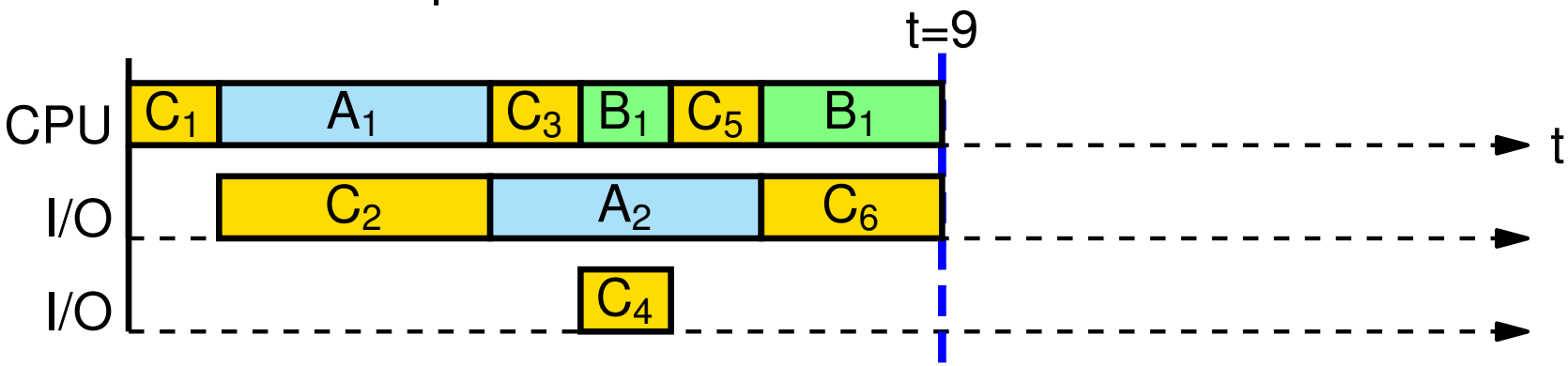
Ordo. SRTF: Shortest Remaining Time First

Politique d'ordonnancement SRTF: principe

chaque fois qu'une nouvelle tâche arrive dans la ready queue, lancer la tâche la plus courte de la ready queue

- variante *préemptive* de SJF

Dans notre exemple:



Ready Queue: A_3

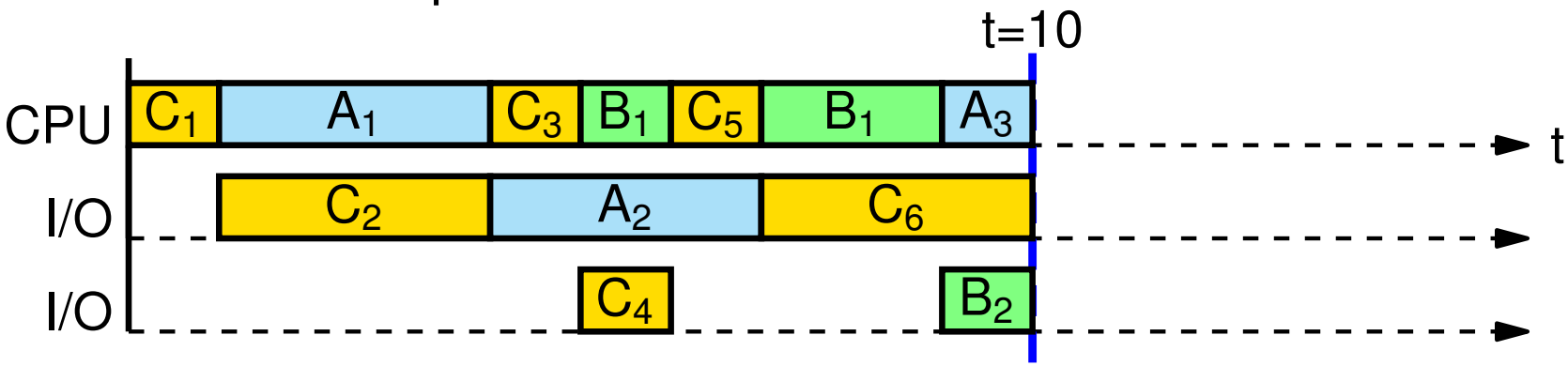
Ordo. SRTF: Shortest Remaining Time First

Politique d'ordonnancement SRTF: principe

chaque fois qu'une nouvelle tâche arrive dans la ready queue, lancer la tâche la plus courte de la ready queue

- variante *préemptive* de SJF

Dans notre exemple:



Ready Queue: A_3, B_3, C_7

- Préemption de la tâche active (A_3) au profit de la tâche C_7

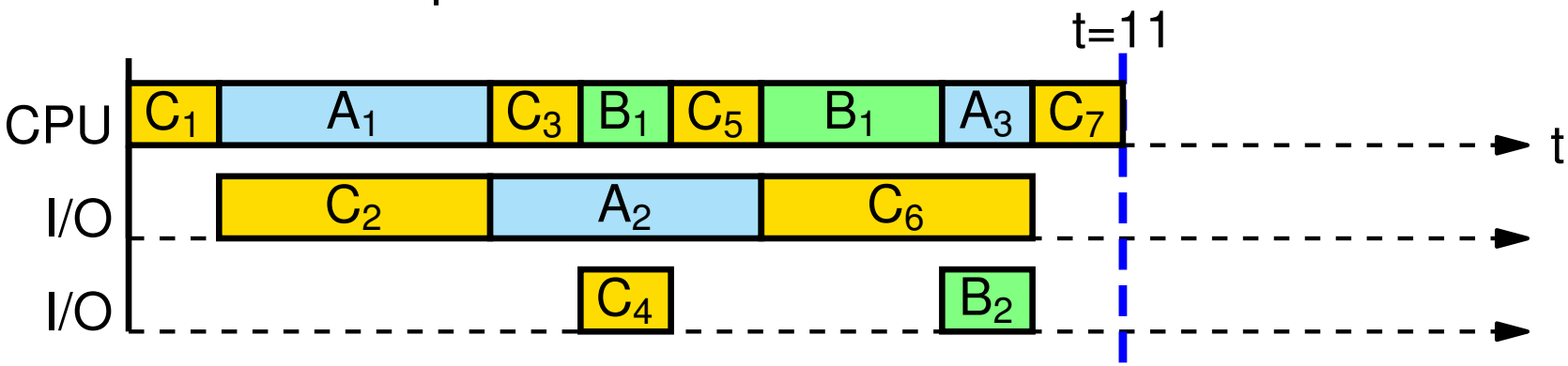
Ordo. SRTF: Shortest Remaining Time First

Politique d'ordonnancement SRTF: principe

chaque fois qu'une nouvelle tâche arrive dans la ready queue, lancer la tâche la plus courte de la ready queue

- variante *préemptive* de SJF

Dans notre exemple:



Ready Queue: A_3, B_3

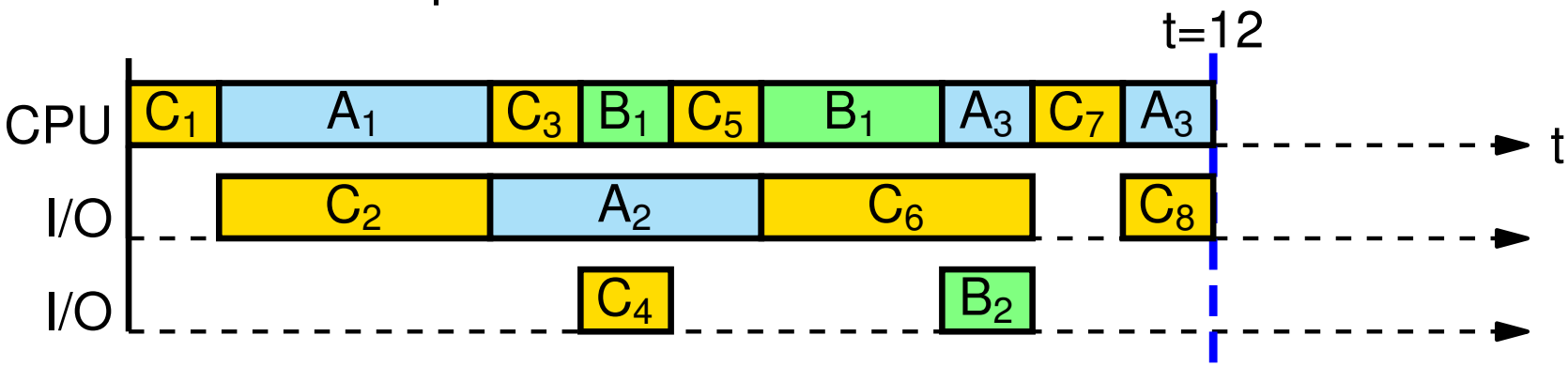
Ordo. SRTF: Shortest Remaining Time First

Politique d'ordonnancement SRTF: principe

chaque fois qu'une nouvelle tâche arrive dans la ready queue, lancer la tâche la plus courte de la ready queue

- variante *préemptive* de SJF

Dans notre exemple:



Ready Queue: A_3, B_3, C_9

- C est prêt mais A garde le CPU

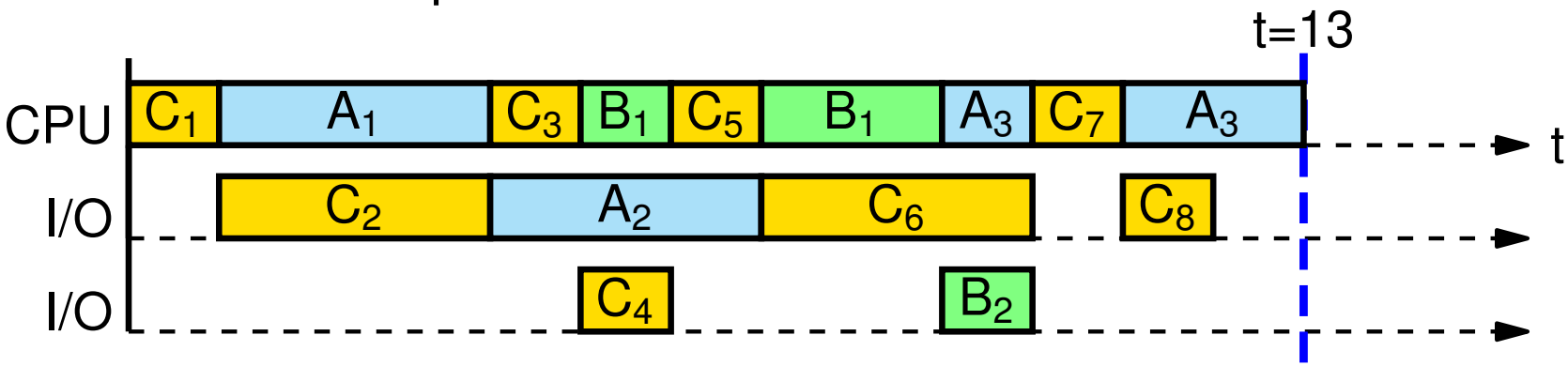
Ordo. SRTF: Shortest Remaining Time First

Politique d'ordonnancement SRTF: principe

chaque fois qu'une nouvelle tâche arrive dans la ready queue, lancer la tâche la plus courte de la ready queue

- variante *préemptive* de SJF

Dans notre exemple:



Ready Queue: B_3, C_9

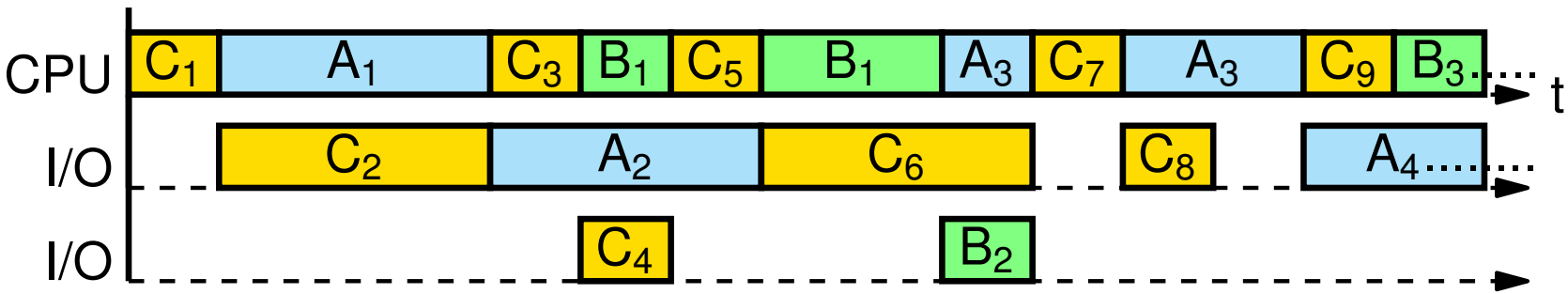
Ordo. SRTF: Shortest Remaining Time First

Politique d'ordonnancement SRTF: principe

chaque fois qu'une nouvelle tâche arrive dans la ready queue, lancer la tâche la plus courte de la ready queue

- variante *préemptive* de SJF

Dans notre exemple:



Remarques:

- ordonnancement préemptif: plus réactif que SJF
- mais famine toujours possible

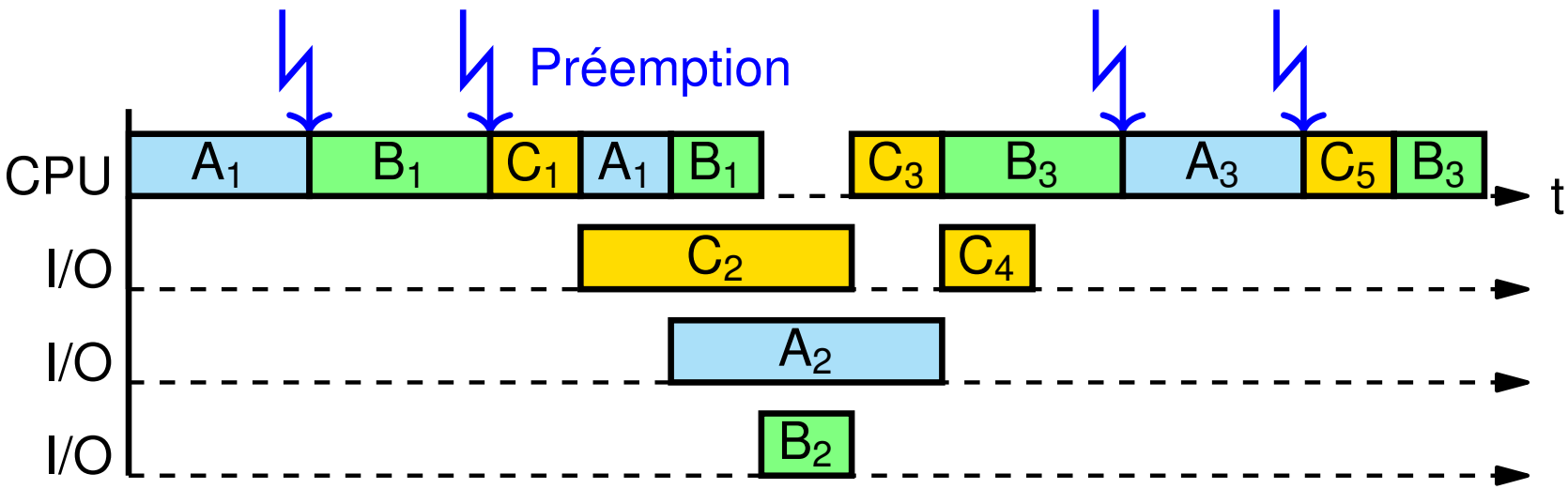
Ordonnancement RR: Round Robin (en VF: tourniquet)

Politique d'ordonnancement Round Robin: principe

- donner le CPU à tour de rôle à chaque tâche...
- ... pour une durée maximale q (en VO time quantum)
- préemption lorsqu'une tâche dépasse son quantum

▶ variante préemptive de FCFS

Dans notre exemple, ordonnancé en RR avec $q = 2$



Ordonnancement Round Robin: remarques

- Variante préemptive de FCFS
  - nouvelles tâches prêtes toujours ajoutées en fin de file
  - ready queue traitée comme une file d'attente FIFO
  - ▶ famine impossible

- Préemption grâce à IRQ du system timer
  - si IRQ à fréquence constante ▶ notion de kernel tick
    - quantum exprimé en nombre de ticks
  - si IRQ flexibles = *tickless kernel*
    - quantum exprimé en millisecondes

- Compromis entre durée du quantum et latence du dispatcher
  - si quantum trop long ▶ manque de réactivité
  - si quantum trop court ▶ surcoût en performance

La vraie vie: ordonnancement à priorités

Ordonnancement à priorités: principe

- maintenir plusieurs ready queues distinctes
- les examiner par ordre de priorité décroissante
- chaque file applique une politique différente: RR, SRTF...

Variantes:

- priorités fixes ▶ ordonnancement temps-réel
- priorités variables ▶ temps partagé (AKA *best-effort*)
  - exemple: Multi-Level Feedback Queues Scheduling (MLFQ)
  - on se donne des critères pour changer un processus de file

Exemple de MLFQ:

- priorité haute: RR q=5ms ▶ processus «interactifs»
- priorité moyenne: RR q=50ms ▶ processus I/O-bound
- priorité basse: SRTF ▶ CPU-bound en tâche de fond

Plan

1. Introduction: rappels sur la notion de processus
2. Multitâche par commutation de contextes d'exécution
3. Ordonnancement: formulation du problème
4. Ordonnancement: les stratégies classiques
5. Évaluation de politiques d'ordonnancement

Évaluation de politiques d'ordonnancement

Méthodologies d'évaluation

- simulation déterministe: sur un scénario donné
  - dérouler les algorithmes, à la main ou sur machine
- modélisation stochastique
  - théorie des files d'attente, chaînes de markov...
- instrumentation de système réel: benchmarking
  - interférences de performances, choix de la workload

Nous = simulation à la main

- 1 «tâche» = 1 CPU-burst
- paramètres = durée d'exécution et date d'arrivée

Exemples:

tâche	durée
T1	6
T2	8
T3	3

tâche	arrivée	durée
T1	0	8
T2	1	4
T3	2	9
T4	3	5

Critères d'évaluation

CPU utilization rate: fraction du temps où le CPU est productif

- i.e. occupé à exécuter du code applicatif (vs noyau, ou *CPU idle*)

Throughput = débit: nombre de tâches terminées par unité de temps

- attention: n'a de sens que si les «tâches» peuvent «terminer»

Non-Starvation: absence garantie de risque de famine

- propriété «qualitative» en VF: non-privation, équité

Turnaround time (tps de séjour): durée entre arrivée et terminaison

- n'a de sens que si les «tâches» peuvent «terminer»

Waiting time (temps d'attente): durée passée dans la ready queue

- aussi: «missed time» en VF: temps d'attente

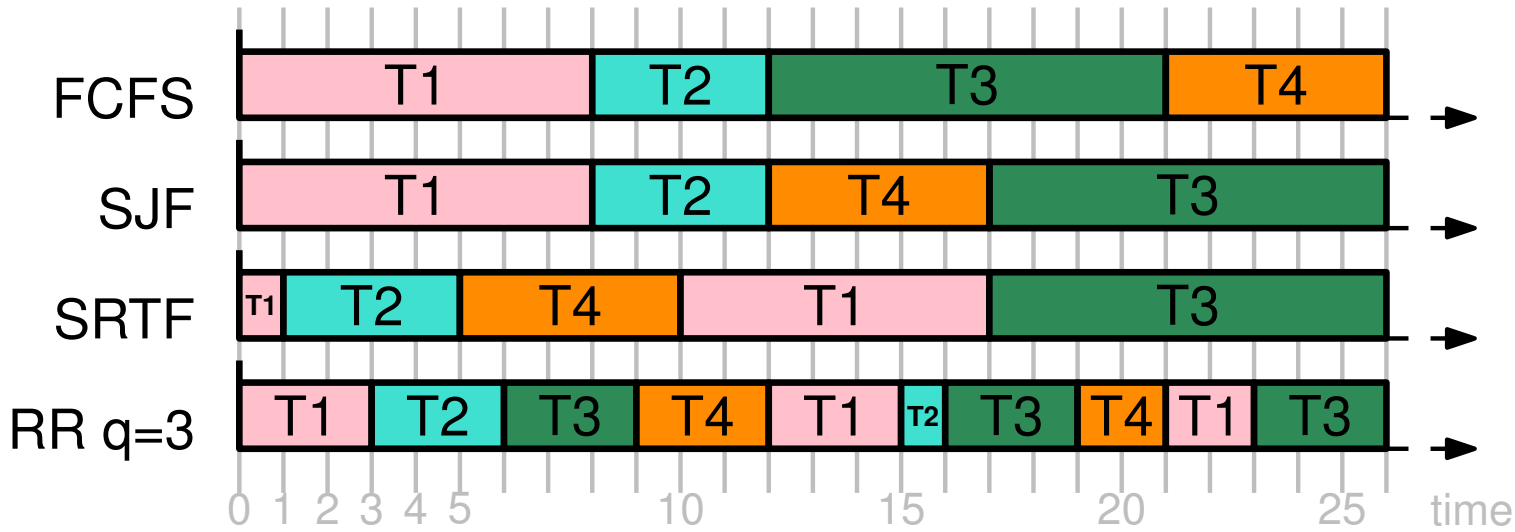
Response time: durée entre arrivée et «réponse»

- attention: il faut définir ce qu'on appelle «réponse»

Exemple

Soit le scénario suivant:

tâche	arrivée	durée
T1	0	8
T2	1	4
T3	2	9
T4	3	5



$$\text{Pour SRTF: TT} = \frac{(17-0)+(5-1)+(26-2)+(10-3)}{4} = 13$$

$$\text{WT} = \frac{(10-1)+0+(17-2)+(5-3)}{4} = 6.5$$

Plan

1. Introduction: rappels sur la notion de processus
2. Multitâche par commutation de contextes d'exécution
3. Ordonnancement: formulation du problème
4. Ordonnancement: les stratégies classiques
5. Évaluation de politiques d'ordonnancement

À retenir

Policy vs Mechanism

- Multitasking vs Multiprocessing
- Virtual CPU vs context switch + scheduling

Concepts importants:

- Dispatcher, Scheduler, Process Control Block, Préemption, CPU-burst / IO-burst, États d'un processus, Ready Queue...

Stratégies d'ordonnancement (en VO scheduling policies)

- First Come First Served
- Shortest Job First et Shortest Remaining Time First
- Round Robin avec un certain quantum
- Ordonnancement à priorités fixes ou dynamiques
  - dont Multi-Level Feedback Queue

Évaluation: Turnaround Time, Waiting Time...