Université IBN TOFAIL – Kénitra
ECOLE NATIONALE DES SCIENCES APPLIQUEES

Cours Systèmes Temps Réel

CH2: Ordonnancement Temps Réel

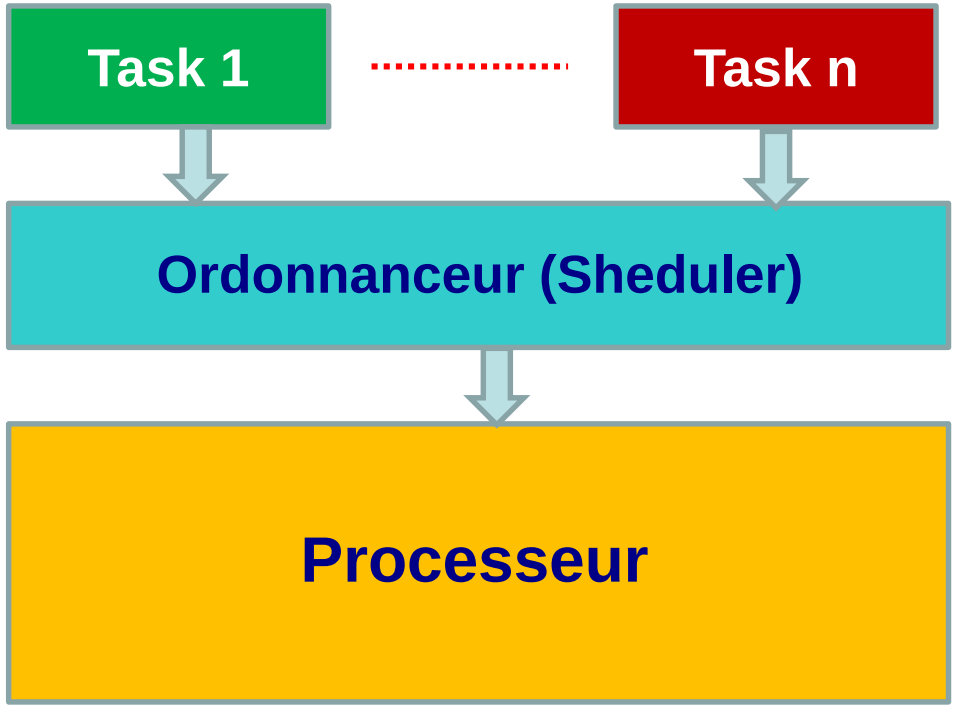
Par:

Hassan EL FADIL

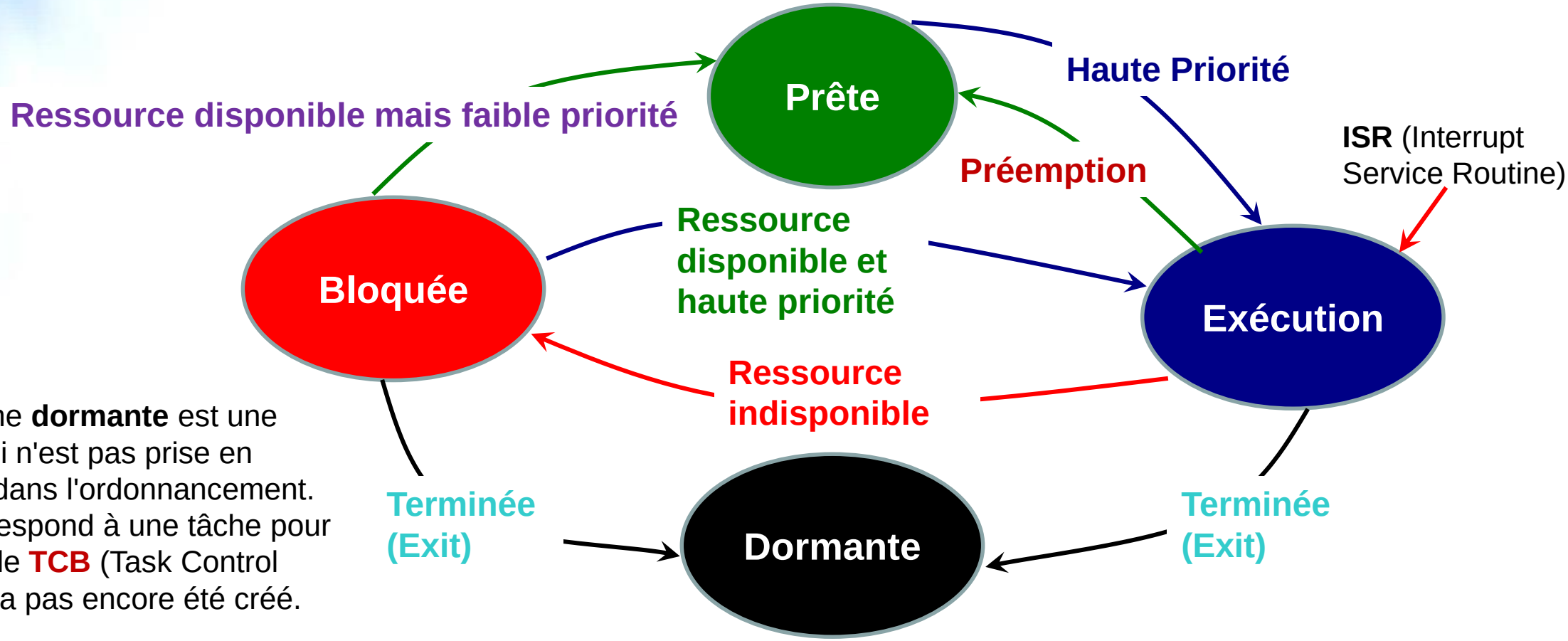
elfadil.h@gmail.com

1. Rôle d'un Ordonnanceur (Gestionnaire de Tâches)

- L'ordonnanceur (**scheduler**) est responsable de la gestion des tâches et de l'allocation des ressources CPU en fonction des priorités et des contraintes temporelles spécifiques des applications temps réel.
- Il vise à optimiser les performances du système en minimisant les temps d'attente, en maximisant l'utilisation du CPU, et en respectant les contraintes temporelles des applications temps réel.

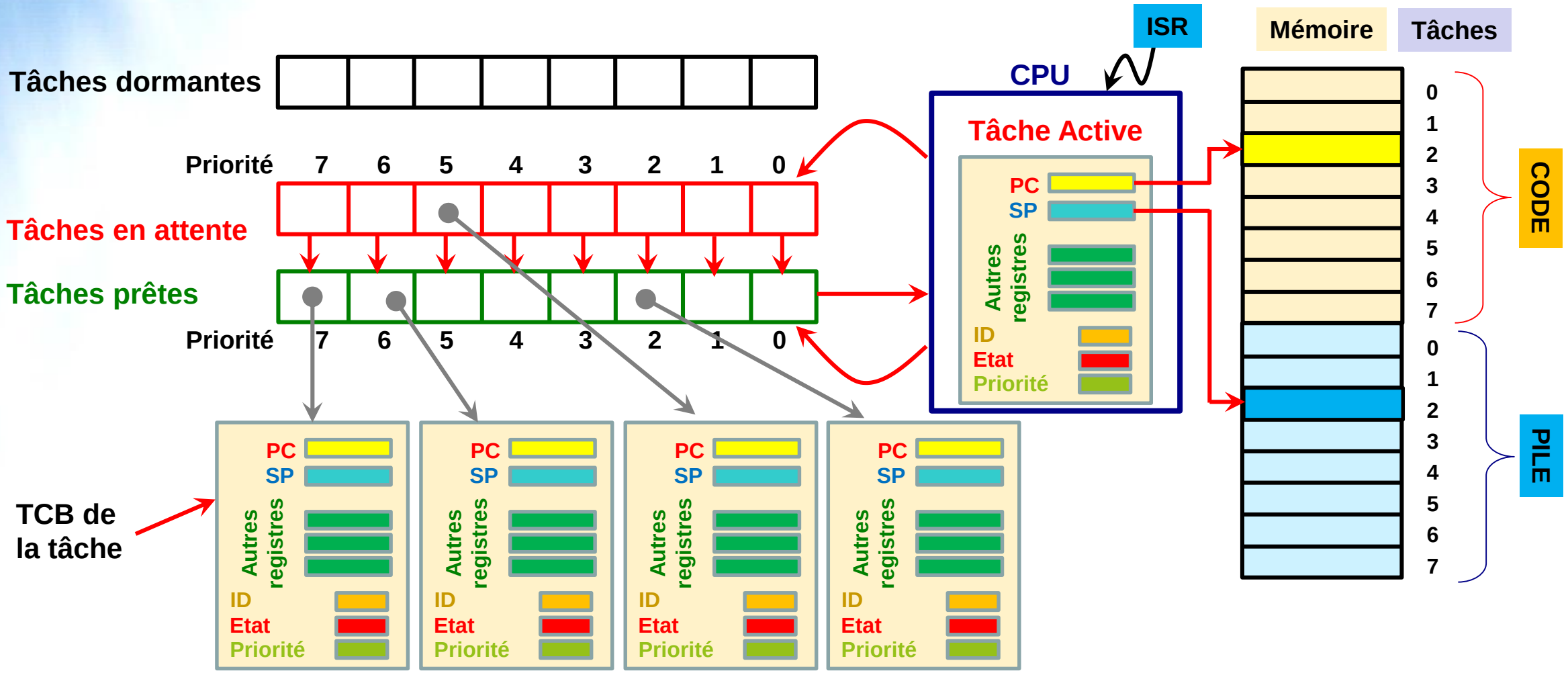


Etats d'une tâche:



Une tâche **dormante** est une tâche qui n'est pas prise en compte dans l'ordonnancement. Elle correspond à une tâche pour laquelle le **TCB** (Task Control Block) n'a pas encore été créé.

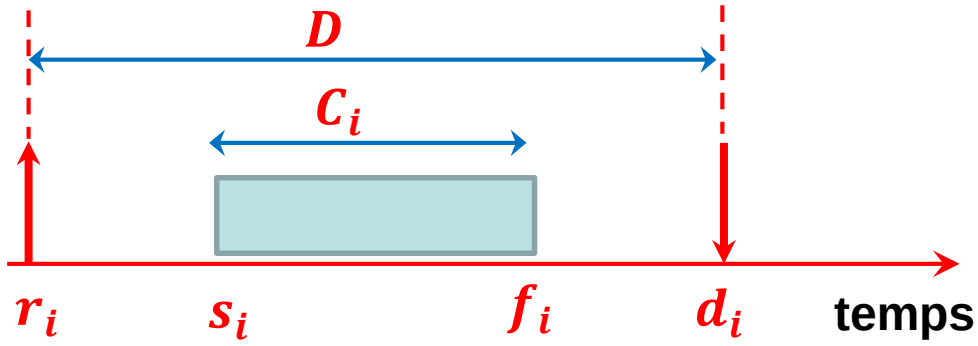
Principe de Fonctionnement de « Scheduler » (Gestionnaire de Tâches):



2. Paramètres de base d'une tâche

2.1. Délai critique d'une tâche:

- Un **délai critique** (ou échéance critique *D*) est le temps maximal autorisé entre l'activation d'une tâche et son achèvement.



- r_i : *date de réveil: moment de déclenchement de la i-ème requête d'exécution de la tâche*
- d_i : *échéance (fin) de la i-ème requête* $d_i = r_i + D$
- s_i : *date de début (Start time)*
- f_i : *date de fin (Finish time)*
- C_i : *temps de calcul (ou d'exécution) maximum sans préemption (Burst time)*

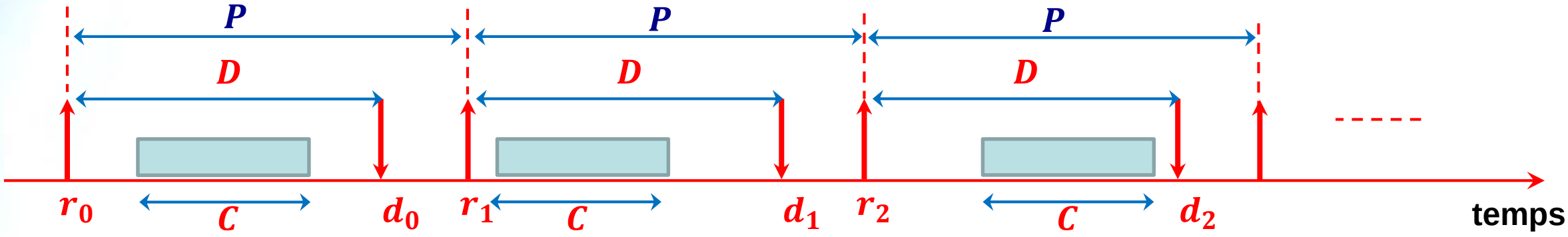
2.2. Périodique / apériodique:

a) Tâche périodique (r_0, C, P, D)

$$0 \leq C \leq D \leq P$$

Les tâches périodiques sont des tâches qui doivent être exécutées à intervalles réguliers ou à des moments spécifiques prédéfinis.

Exemple mesure d'une grandeur par un capteur toutes les 100ms.



C *: Capacité: durée d'exécution au niveau du processeur (burst time)*

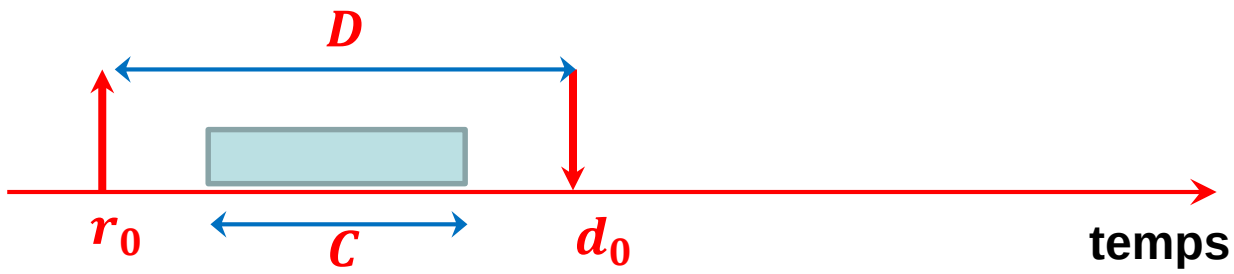
P *: Période*

Une tâche périodique à échéance sur requête ⇒ ***D = P****, représentée par:* ↕

b) Tâche apériodique (r_0, C, D)

Elles sont déclenchées par des événements spécifiques qui peuvent se produire de manière imprévisible ou à des moments variables.

Par exemple, une tâche de gestion des interruptions matérielles.



3. Types des algorithmes d'ordonnancement

- Ordonnanceur **Coopératif (Cooperative)** : tâche par tâche, chaque tâche s'exécute jusqu'à ce qu'elle soit terminée.
- Ordonnanceur **Round-robin** : chaque tâche dispose d'une tranche de temps pour s'exécuter, il n'y a pas de priorité pour l'exécution des tâches.
- Ordonnanceur **non préemptif** (**non-preemptive**): une fois élue, la tâche ne doit plus être interrompue jusqu'à la fin de la requête.
- Ordonnanceur **préemptif** (**preemptive**): une tâche élue peut perdre le processeur au profit d'une tâche plus prioritaire (ou une interruption).

4. Critères d'ordonnancement

- **À maximiser** :
  - *le pourcentage d'utilisation du processeur.*
  - *le débit (nombre moyen de tâches traitées par unité de temps).*

- **À minimiser** :
  - *le temps de réponse : le temps entre une demande et la réponse.*
  - *le temps de rotation : durée moyenne qu'il faut pour qu'une tâche s'exécute.*
  - *le temps d'attente : durée moyenne qu'une tâche passe à attendre.*

5. Performances des algorithmes d'ordonnancement :

- **Temps de rotation ou temps de séjour (Turnaround time) =**
  temps de fin d'exécution - temps d'arrivée
- **Temps d'attente (Waiting time) =**
  temps de rotation - Durée d'exécution
- **Temps moyen d'attente =**
  ∑ (temps d'attente) / nombre de tâches
- **Débit (Throughput) =**
  Nbre de tâches / Temps total

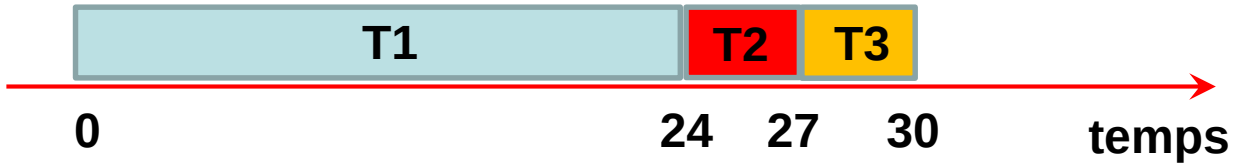
6. Algorithmes d'ordonnancement classiques

6.1. Ordonnancement FCFS (First Come First Served):

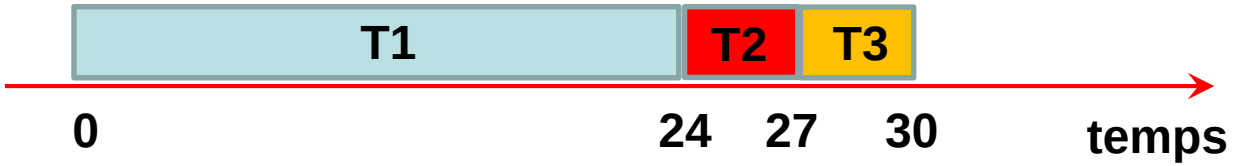
- Connu sous le nom FIFO (First In, First Out), en français (premier arrivé, premier servi);
- les tâches sont rangées dans la file d'attente des tâches prêtes selon leur ordre d'arrivée.
- **Exemple:**

Tâche	Durée d'exécution (en nombre d'unité de temps
T1	24
T2	3
T3	3

Cas où les tâches arrivent au temps 0 dans l'ordre: T1, T2, T3:



- Temps d'attente pour T1= 0, T2= 24, T3= 27
- Temps moyen d'attente: (0 + 24 + 27)/3 = 17
- Débit = Nbre de tâches / temps total = 3/30 = 0,1 (3 processus complétés en 30 unités de temps)
- Temps moyen de rotation: (24+27+30)/3 = 27



Exercice:

Traiter le cas de succession: T2 , T3, T1 ?

- **Avantages de FCFS**:
  - Simple.

- **Inconvénients**:
  - le temps moyen d'attente peut varier grandement,
  - les tâches de faible temps d'exécution sont pénalisées si elles sont précédées dans la file par une tâche de longue durée.

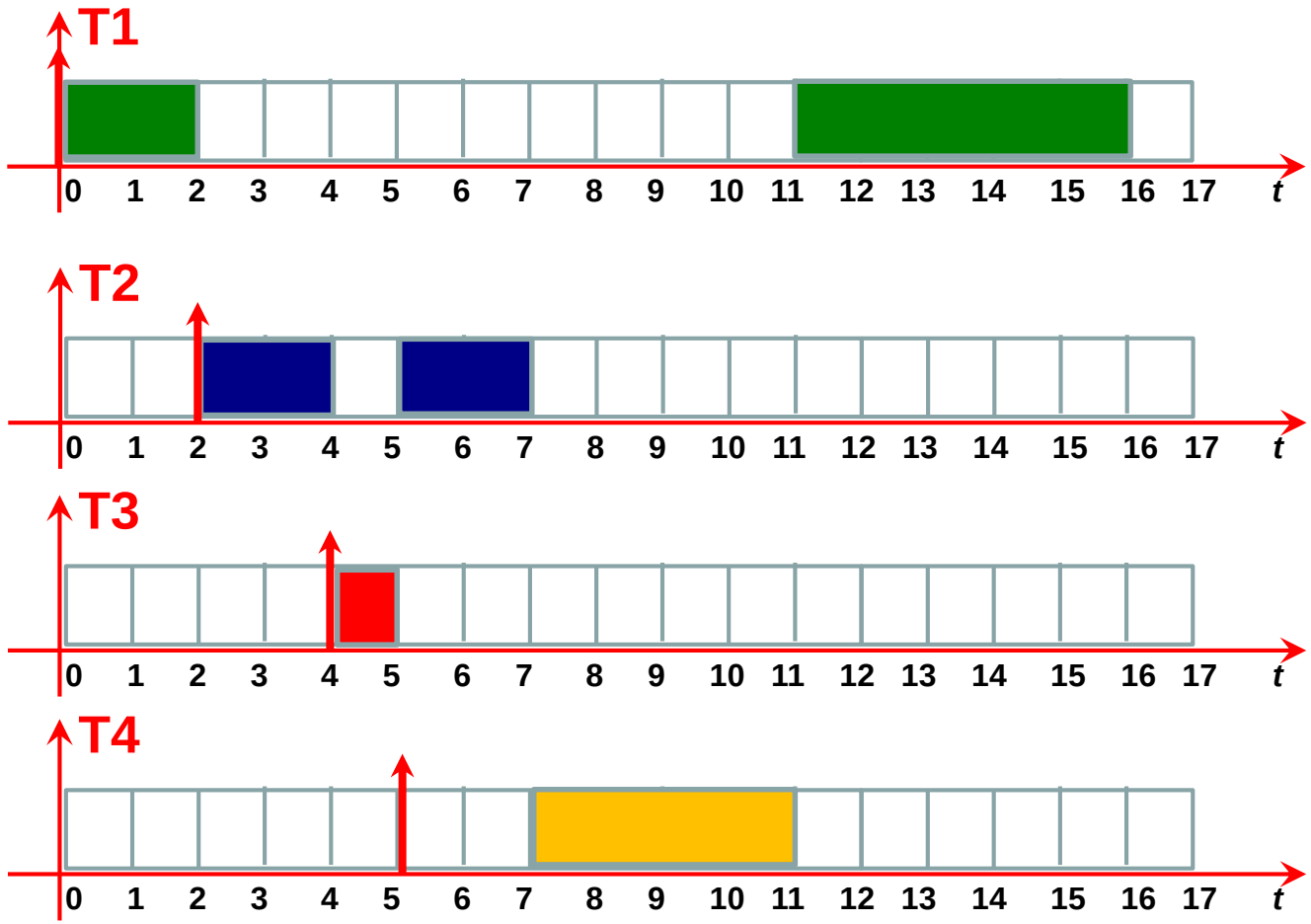
6.2. Ordonnancement SJF (Shortest Job First) (Tâche la Plus Courte d'Abord):

Le processus le plus court part le premier.

a) SJF avec préemption:

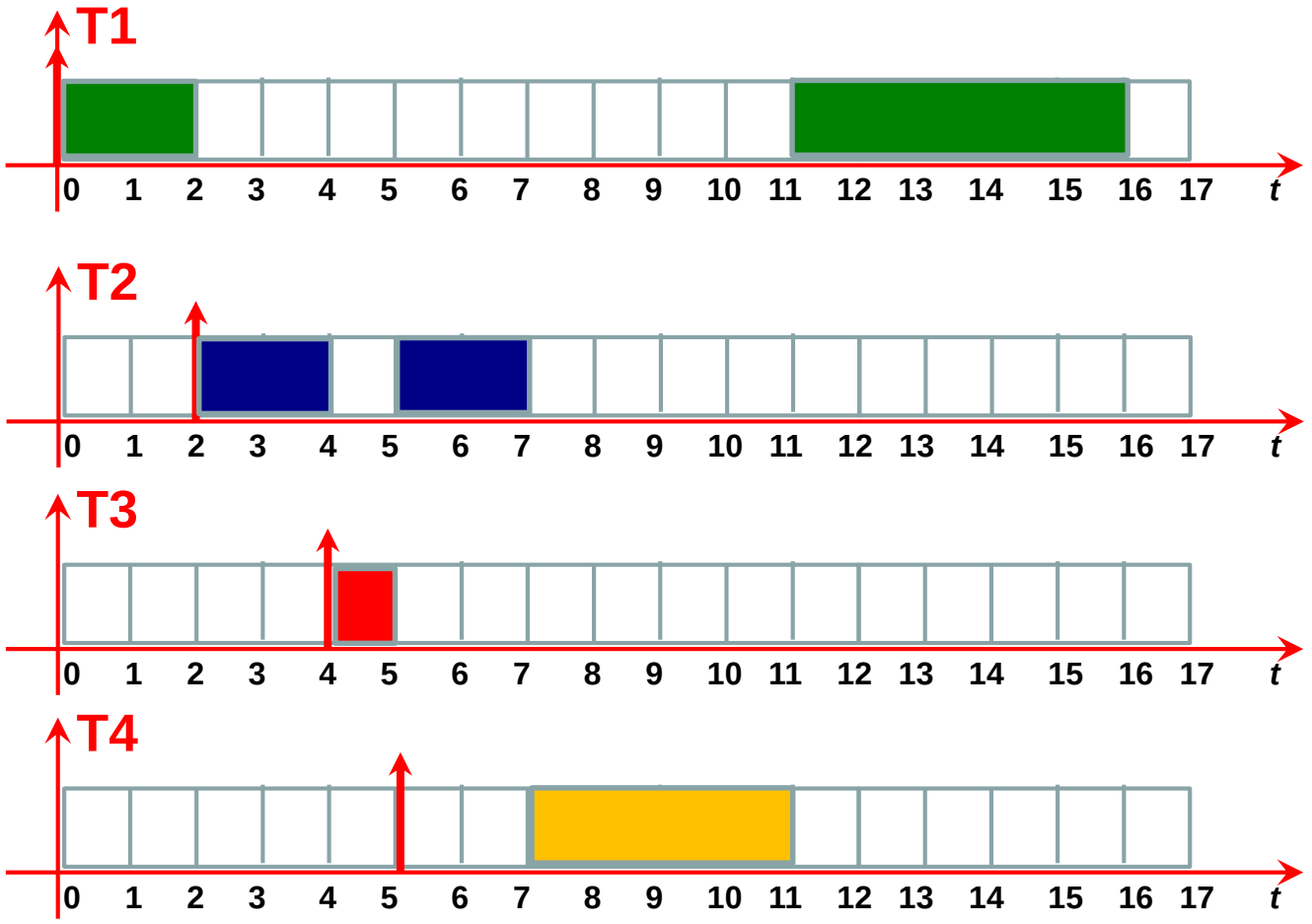
si un processus qui dure moins que le restant du processus courant se présente plus tard, l'UCT est donnée à ce nouveau processus (SRTF: shortest remaining-time first).

Tâche	Temps d'arrivée	Durée d'exécution
T1	0	7
T2	2	4
T3	4	1
T4	5	4



… Algorithmes d'ordonnancement classiques / SJF avec préemption

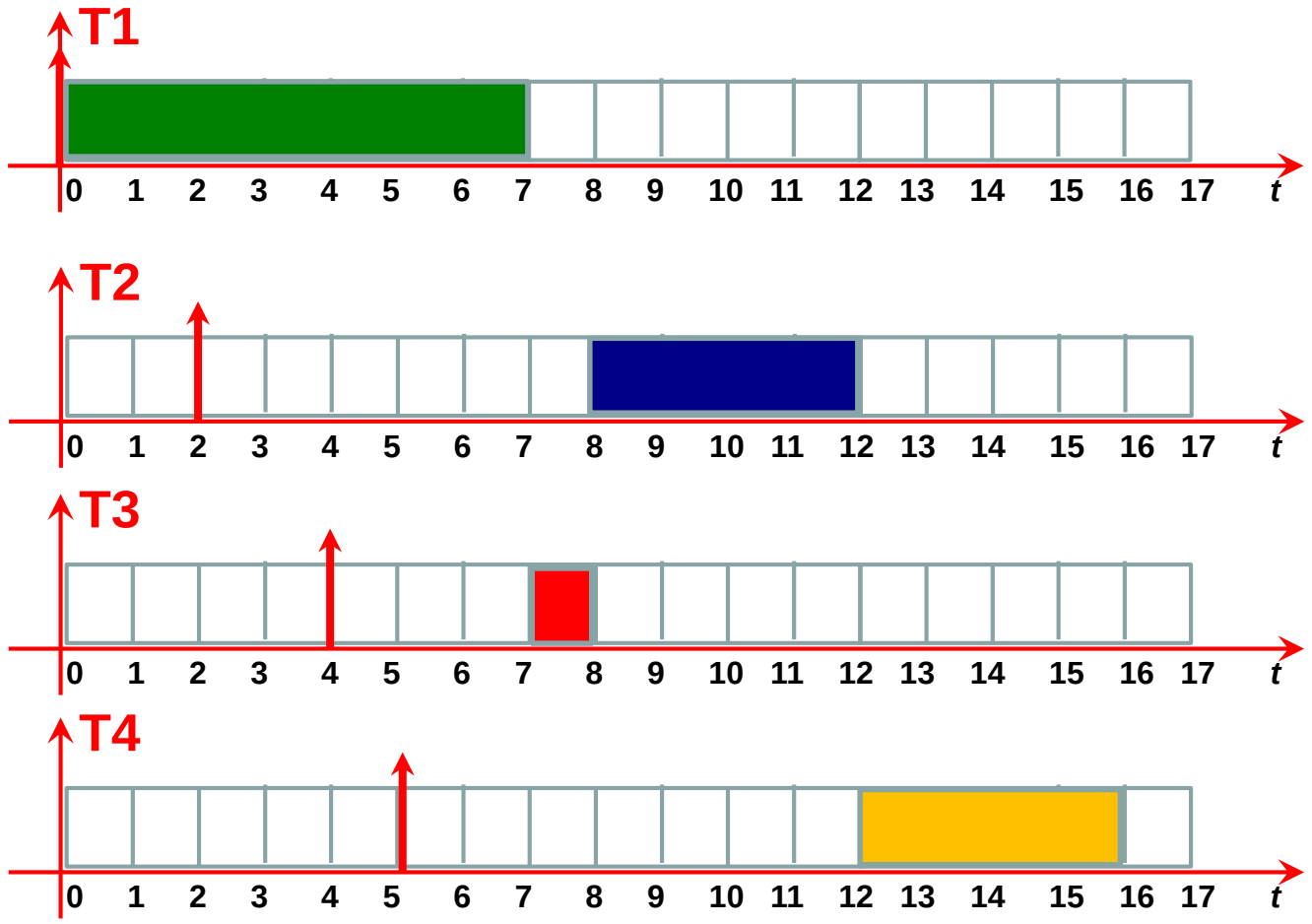
- **Débit** :
  4 tâches/16 (unités de temps) = **1/4**
- **Temps de rotation (turnaround time)**:
  - Pour T1 = 16-0 = **16**
  - Pour T2 = 7-2 = **5**
  - Pour T3 = 5-4 = **1**
  - Pour T4 =11-5 = **6**
- **Temps moyen de rotation**:
  (16+5+1+6)/4 = **7**
- **Temps d'attente**:
  - Pour T1 = 11-2 =16-7 = **9**
  - Pour T2 = 5-4 = **1**
  - Pour T3 = 4-4 = 1-1 = **0**
  - Pour P4 = 7-5 = 6-4 = **2**
- **Temps moyen d'attente**:
  (9+1+0+2)/4 = **3**



a) SJF sans préemption:

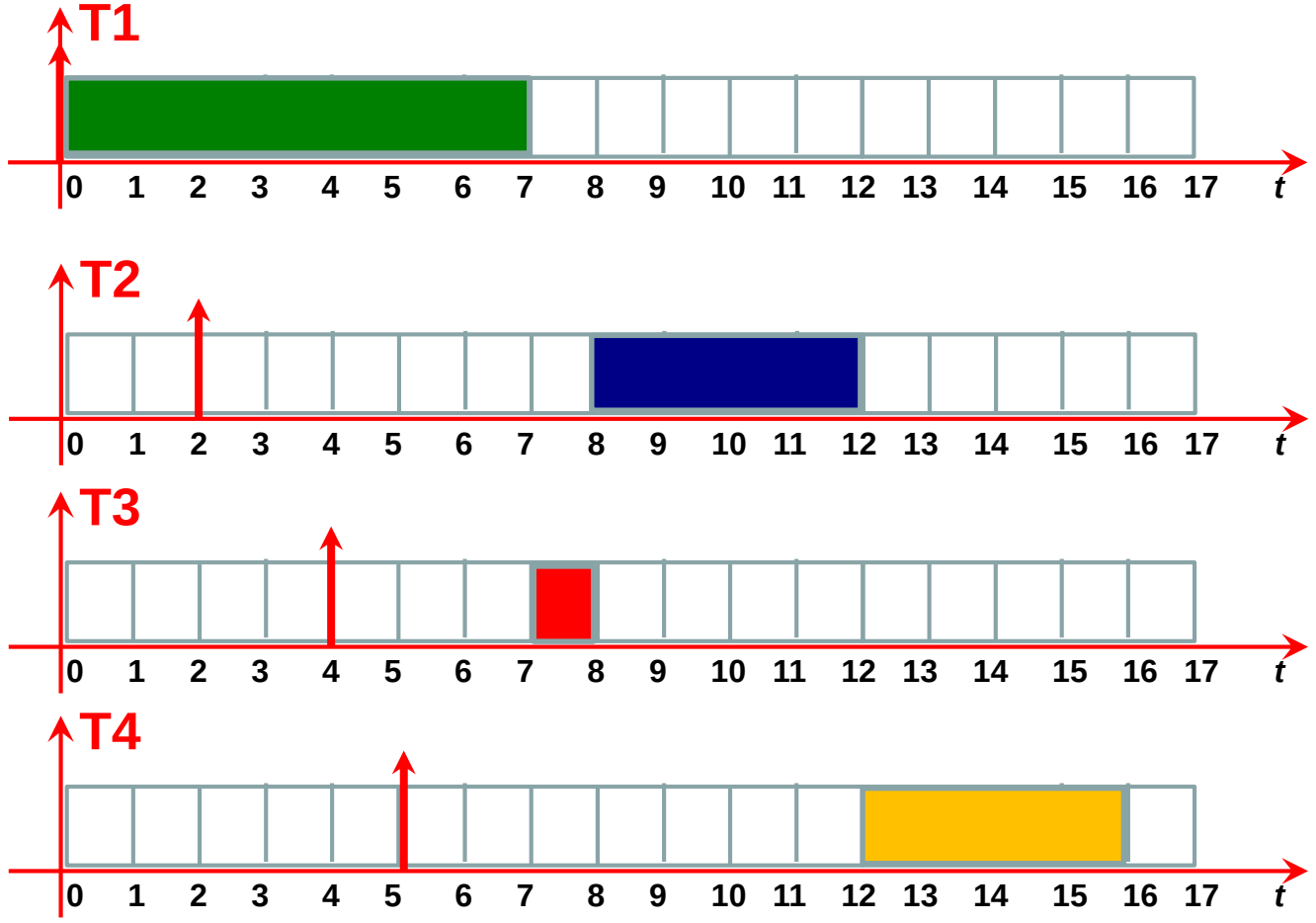
On permet à la tâche courante de terminer son cycle.

Tâche	Temps d'arrivée	Durée d'exécution
T1	0	7
T2	2	4
T3	4	1
T4	5	4



… Algorithmes d'ordonnancement classiques / SJF sans préemption

- **Débit** :
  4 tâches/16 (unités de temps) = **1/4**
- **Temps de rotation (turnaround time):**
  - Pour T1 = 7-0 = **7**
  - Pour T2 = 12-2 = **10**
  - Pour T3 = 8-4 = **4**
  - Pour T4 =16-5 = **11**
- **Temps moyen de rotation**:
  (7+10+4+11)/4 = **8**
- **Temps d'attente**:
  - Pour T1 = **0**
  - Pour T2 = **6**
  - Pour T3 = **3**
  - Pour P4 = **7**
- **Temps moyen d'attente**:
  (0+6+3+7)/4 = **4**



- **Avantages de SJF:**
  - Remédie à l'inconvénient de FCFS,
  - Minimise le temps de réponse moyen.

- **Inconvénients:**
  - Pénalise les travaux longs,
  - Elle impose d'estimer les durées d'exécution des tâches a priori (connues difficilement).

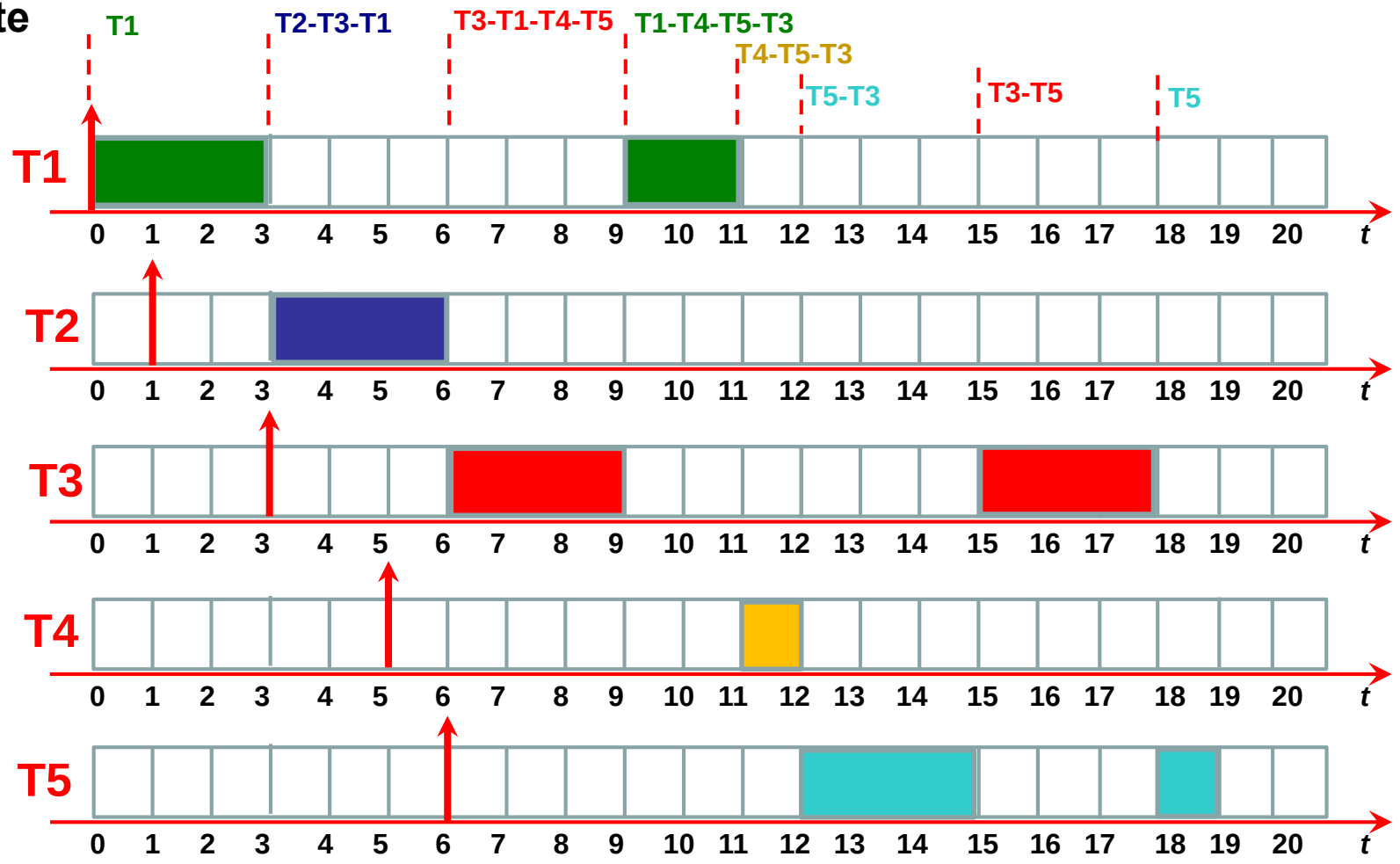
6.3. Ordonnancement par tourniquet (Round Robin) :

- L'ordonnancement Round Robin (RR) est une technique d'ordonnancement utilisée principalement dans les systèmes d'exploitation multitâches pour gérer les processus en attente.

- Quantum de temps (TQ), ou quantum (time quantum). C'est la durée fixe pendant laquelle un processus est autorisé à s'exécuter avant d'être suspendu et remplacé par le processus suivant dans l'ordonnancement Round Robin

… Algorithmes d'ordonnancement classiques / RR

File d'attente Prête (Ready Queue):

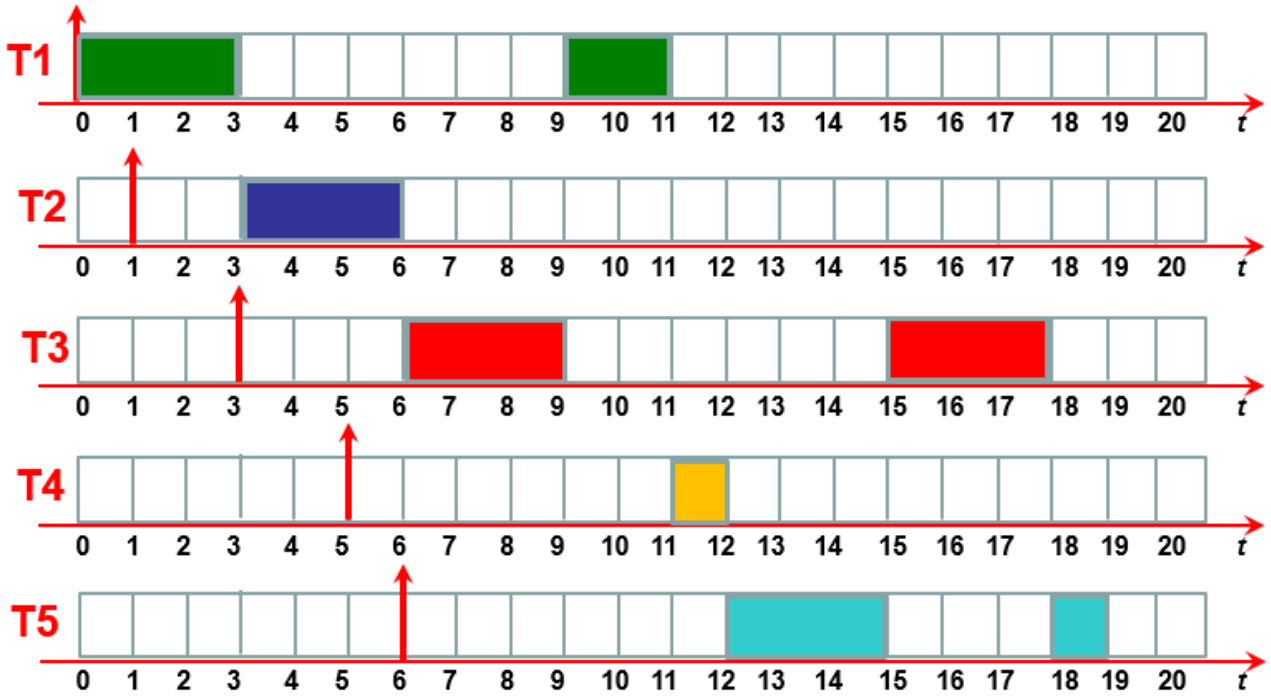


Exemple:

Tâche	Temps d'arrivée	Durée d'exécution
T1	0	5
T2	1	3
T3	3	6
T4	5	1
T5	6	4

TQ=3 (unités de temps)

- **Débit** :
  5 tâches/19 (unités de temps) = **0.263**
- **Temps de rotation (turnaround time):**
  - Pour T1 = 11-0 = **11** Pour T2 = 6-1 = **5**
  - Pour T3 = 18-3 = **15** Pour T4 =12-5 = **7**
  - Pour T5 = 19-6 = **13**
- **Temps moyen de rotation**:
  (11+5+15+7+13)/5 = **6,37**
- **Temps d'attente**:
  - Pour T1 = **6** Pour T2 = **2**
  - Pour T3 = **9** Pour P4 = **6**
  - Pour T5 = **9**
- **Temps moyen d'attente**:
  (6+2+9+6+9)/5 = **6,4**



Avantages:

- *Équité : Chaque tâche obtient une chance égale de s'exécuter,*
- *Simplicité*
- *Prévisibilité : Le comportement de l'algorithme est prévisible, car chaque tâche obtient le même quantum de temps.*

Inconvénients:

- *Surcharge du contexte : Un quantum de temps trop petit entraîne de fréquents changements de contexte, augmentant ainsi la surcharge du système.*
- *Choix du quantum : La performance de l'algorithme dépend fortement du choix de la taille du quantum.*
- *Moins efficace pour les tâches longues :*
- *Pas de priorités*

Exercice:

Un système monoprocesseur ordonnance l'exécution des tâches ci-contre.
Dessiner le digramme de Gantt pour l'algorithme RR. On prend **TQ=4 unités.**

Tâche	Temps d'arrivée	Durée d'exécution
T1	0	3
T2	2	6
T3	4	4
T4	6	5
T5	8	2

6.4. Ordonnancement basé sur les priorités:

a) *Définition*:

- L'ordonnancement basé sur les priorités est une méthode d'ordonnancement dans laquelle chaque tâche se voit attribuer une priorité, et la tâche avec la plus haute priorité est sélectionné pour s'exécuter en premier.
- Les priorités peuvent être fixes (statiques) ou dynamiques:
  - **Priorité statique (fixe)** : Les priorités sont fixées au moment de la création des tâches et ne changent pas.
  - **Priorité dynamique** : Les priorités peuvent changer en fonction du comportement de la tâche ou des politiques du système.

b) *Principe:*

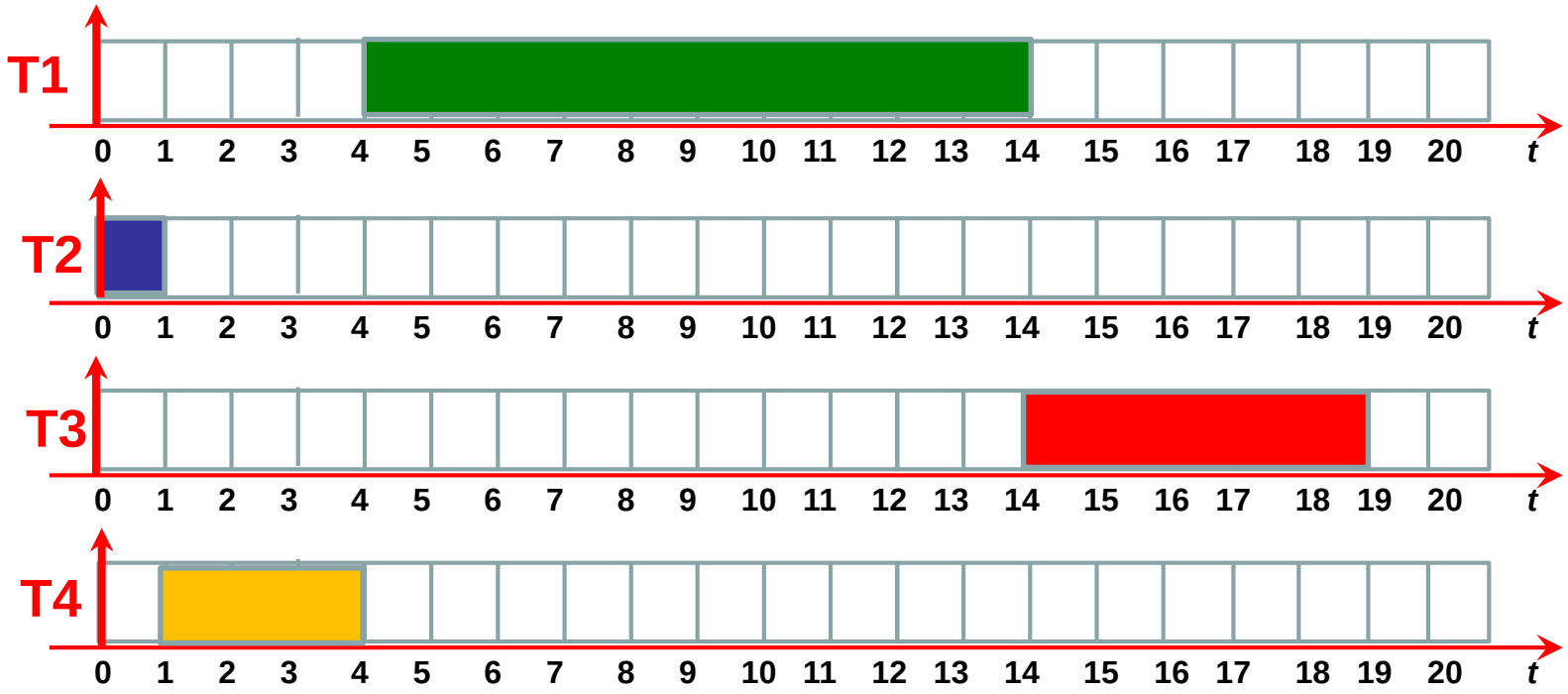
- La tâche avec la priorité la plus élevée est sélectionnée pour s'exécuter.
- En cas de tâches de même priorité, une méthode secondaire comme FCFS (First-Come, First-Served) peut être utilisée.
- Si une tâche avec une priorité plus élevée arrive, la tâche en cours d'exécution peut être interrompue (préemptée) pour permettre à la tâche prioritaire de s'exécuter.

- **Avantages** :
  - Adapté aux systèmes temps réels durs (strictes),
  - Répond efficacement aux interruptions et aux alertes.

- **Inconvénients**:
  - Problème de la famine (Starvation): les processus de basse priorité peuvent être indéfiniment retardés s'il y a toujours des processus de haute priorité prêts à s'exécuter.

Exemple :
Supposons que les 4 tâches arrivent au même temps (instant 0).

Tâche	Priorité	Durée d'exécution
T1	3	10
T2	1	1
T3	4	5
T4	2	3



- Temps moyen d'attente =
  (4 + 0 + 14 + 1)/4 = **4,75**

Exercice:

Un système monoprocesseur ordonnance l'exécution des tâches ci-contre.

1) Dessiner le digramme d'ordonnancement (diagramme de Gantt) pour l'algorithme Priorité.
2) Calculer le temps de séjour (de rotation) de chaque tâche

Tâche	Temps d'arrivée	Durée d'exécution	Priorité
T1	0	5	4
T2	2	4	2
T3	2	2	6
T4	4	4	3

7. Ordonnancement Temps Réel

7.1. Test d'ordonnançabilité (schedulability test):

- Les tests d'ordonnançabilité (ou tests de planifiabilité) sont des méthodes utilisées pour déterminer si un ensemble de *n* tâches peut être correctement ordonnancé (ou planifié) pour respecter leurs contraintes de temps dans un système temps réel.

- Ces tests sont essentiels pour garantir que toutes les tâches critiques respectent leurs délais.

- Il existe trois types de tests:
  - ***Test suffisant****: s'il est vérifié alors les tâches sont ordonnaçables. Sinon, les tâches pourraient être ordonnaçables mais pas nécessairement.*
  - ***Test nécessaire****: s'il est vérifié alors les tâches pourraient être ordonnaçables mais pas nécessairement. Sinon, les tâches ne sont pas définitivement ordonnaçables.*
  - ***Test exacte*** *(nécessaire et suffisant): Les tâches sont ordonnaçables si et seulement si le test est vérifié.*

7.2. Algorithme à Priorités fixes (RM : Rate Monotonic):

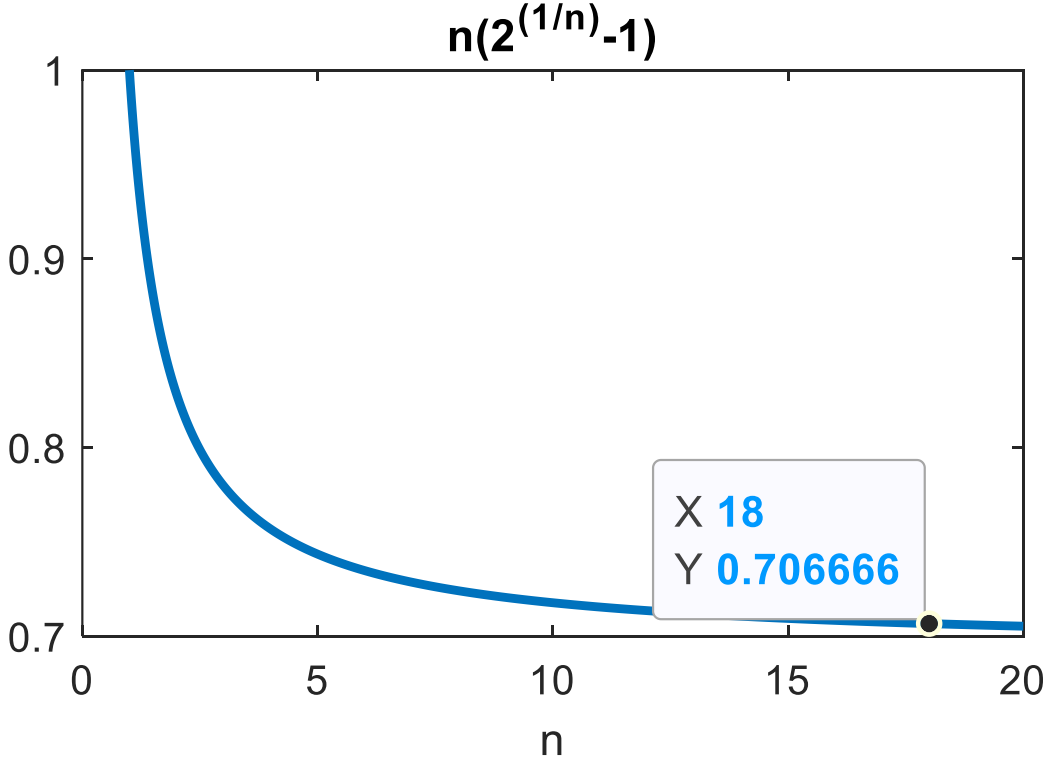
- La tâche de plus petite période est la plus prioritaire (tâches périodiques).
- L'utilisation de cet algorithme est valable uniquement si les tâches sont à échéance sur requête ($D_i = P_i$: Échéance égale à la période).
- **Test d'ordonnaçabilité** (condition suffisante):

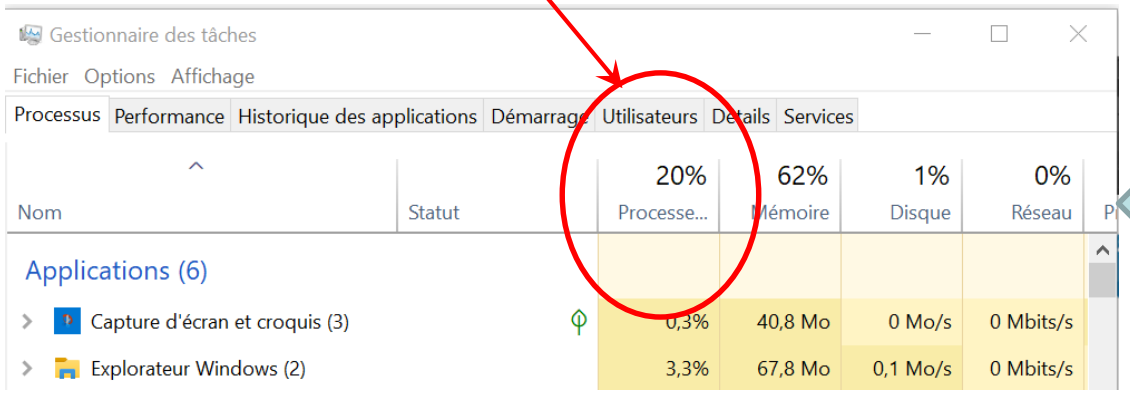
  Un ensemble de ***n*** tâches est ordonnançable si la condition suivante est vérifiée:

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n\left(2^{1/n} - 1\right)$$

Condition de Liu et Layland

- Pour $n \to \infty$, ce résultat tend vers $\boldsymbol{ln2 = 0.69\ (69\%)}$: on ne pourra jamais dépasser 69% de la capacité du processeur
- $U = \sum_{i=1}^{n} \frac{C_i}{P_i}$ représente le taux d'utilisation du processeur





Exemple de gestionnaire de tâches sous Windows

Exemple :

Tâche	Période Pi	Durée d'exécution Ci	Echéance Di
T1	20	3	20
T2	5	2	5
T3	10	2	10

1) Nous avons: $\boldsymbol{D_i = P_i}$
2) Test d'ordonnaçabilité:

a) Taux d'utilisation processeur:

$$\boldsymbol{U = \sum_{i=1}^{n} \frac{C_i}{P_i} = \frac{3}{20} + \frac{2}{5} + \frac{2}{10} = 0.75}$$

a) Limite théorique pour l'ordonnançabilité:

$$\boldsymbol{U_{max} = n\left(2^{1/n} - 1\right) = 3\left(2^{1/3} - 1\right) = 0.7798}$$

Puisque $\boldsymbol{0.75 < 0.7798}$ donc les système constitué par les 3 tâches est ordonnançable.

3) Définition des Priorités:

Tâche qui a la plus petite période est prioritaire:

Prio(T2) > Prio(T3) > Prio(T1)

4) Cycle d'étude: $\boldsymbol{Cycle = [0, PPCM(P_i)]}$

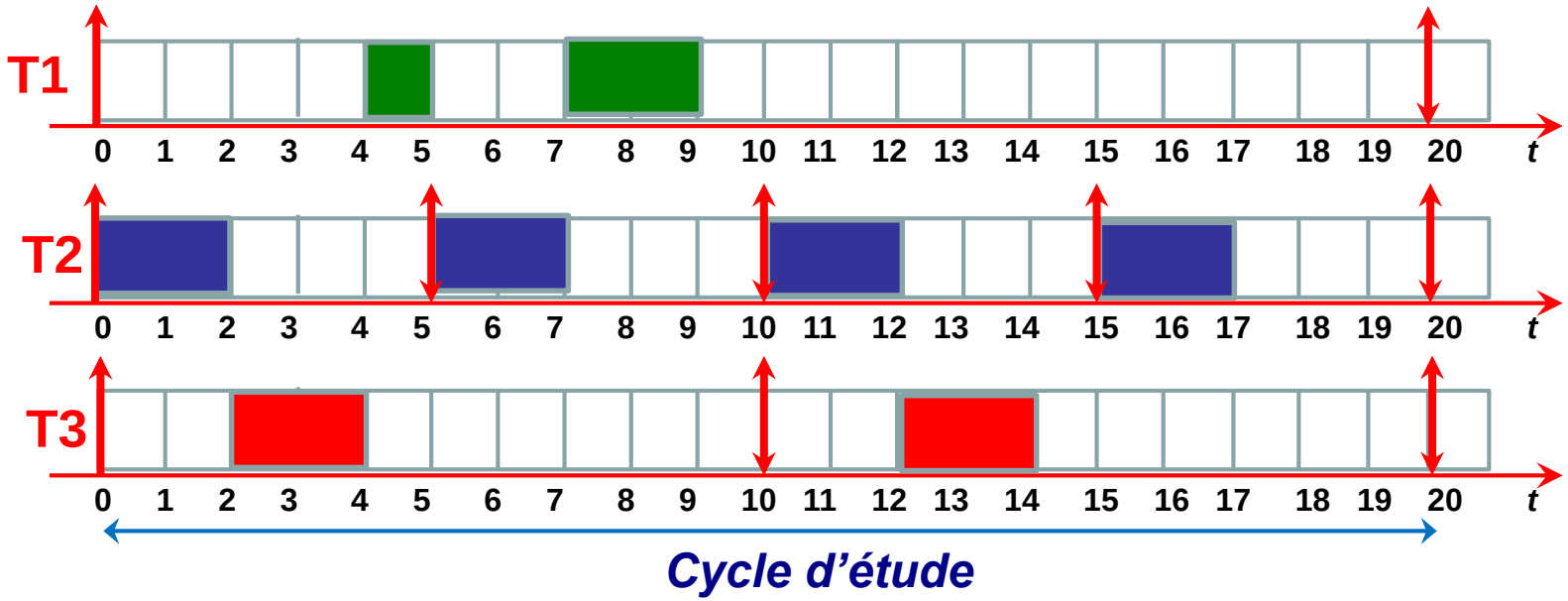
Cycle = Plus Petit Commun Multiple des Périodes (PPCM)

$$\boldsymbol{Cycle = [0, PPCM(20,5,10)] = [0, 20]}$$

Exemple :

Tâche	Période Pi	Durée d'exécution Ci	Echéance Di
T1	20	3	20
T2	5	2	5
T3	10	2	10

Prio(T2) > Prio(T3) > Prio(T1)



Exercice :

Trois tâches présentent la configuration suivante:

- $P_1 = 100$ $P_2 = 150$ $P_3 = 350$
- $C_1 = 20$ $C_2 = 40$ $C_3 = 100$

1) Tracer le diagramme d'activité selon l'algorithme RM.
2) Quel est le taux d'utilisation du CPU?
3) Cette configuration est-elle ordonnançable?

- **Avantages :**
  - Simplicité de mise en œuvre
  - Optimal pour les ordonnancements à priorité statique
  - Répandu dans les exécutifs classiques
  - Bon comportement en cas de surcharge
- **Inconvénients**
  - Surdimensionnement possible du système

7.3. Algorithme Inverse Deadline (ID) ou Deadline Monotonic (DM):

- La priorité d'un e tâche dépend de son délai critique (deadline).
- La tâche la plus prioritaire est celle de plus petit délai critique.
- **Propriétés**:
  - Optimal pour les tâches périodiques indépendantes où ($D_i \leq P_i$: Échéance inférieure ou égale à la période).
  - **Priorité fixe** (déterminée au début de l'algorithme).

Test d'ordonnançabilité (Schedulability) :

- Pour ***n*** tâches périodiques, le système est ordonnançable par l'algorithme DM si:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n\left(2^{1/n} - 1\right)$$

- Cette condition est suffisante mais pas nécessaire.

Exemple :

Tâche	Période Pi	Durée d'exécution Ci	Echéance Di
T1	20	3	14
T2	5	2	5
T3	15	2	15

1) Nous avons: $\boldsymbol{D_i \leq P_i}$

2) Test d'ordonnaçabilité:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} = \frac{3}{14} + \frac{2}{5} + \frac{2}{15} = 0.7476$$

Limite théorique pour l'ordonnançabilité:

$$U_{max} = n\left(2^{1/n} - 1\right) = 3\left(2^{1/3} - 1\right) = 0.7798$$

Puisque $0.7476 < 0.7798$ donc les système constitué par les 3 tâches est ordonnançable.

3) Définition des Priorités:

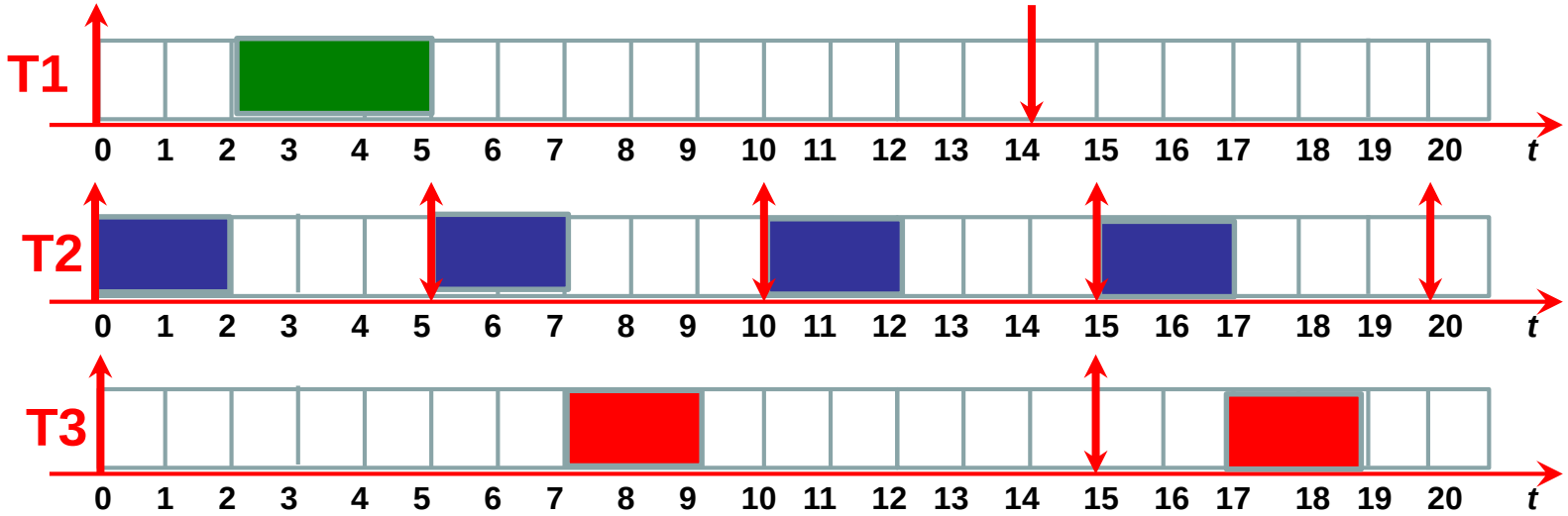
Tâche qui a la plus petite échéance est prioritaire:

Prio(T2) > Prio(T1) > Prio(T3)

Exemple :

Tâche	Période Pi	Durée d'exécution Ci	Echéance Di
T1	20	3	14
T2	5	2	5
T3	15	2	15

Prio(T2) > Prio(T1) > Prio(T3)



- **Avantages** :
  - Voir Algorithme RM
  - Rate Monotonic Scheduling pénalise les tâches peu fréquentes (longue période) mais urgentes (faible échéance).
  - Deadline Monotonic Scheduling s'avère meilleur dans ce cas.
- **Inconvénients** :
  - Voir RM

7.4. Algorithme à priorité variable ou dynamique (EDF : Earliest Deadline First) :

- **Propriétés**:
  - A chaque instant (i.e. à chaque réveil de tâche), la priorité maximale est donnée à la tâche dont l'échéance est la plus proche.
  - Ordonnancement dynamique préemptif basé sur les priorités.
  - Les priorités sont mises à jour à chaque réveil d'une tâche.
  - Optimal aussi bien pour les tâches périodiques qu'apériodiques.
  - Cycle d'étude = [0 ,$PPCM(Pi)$] (départ simultané des tâches).

- **Test d'ordonnançabilité (EDF)**:
  - Pour *n* tâches synchrones à échéance sur requête: $\boldsymbol{D_i = P_i}$

Liu et Layland, 1973 $$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Condition exacte
(nécessaire + suffisante)

  - Pour les autres tâche avec: $\boldsymbol{D_i \leq P_i}$

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$$

Condition suffisante

Exemple :

Tâche	Période Pi	Durée d'exécution Ci	Echéance Di
T1	20	1	8
T2	5	2	4
T3	10	4	10

1) Nous avons: $D_i \leq P_i$

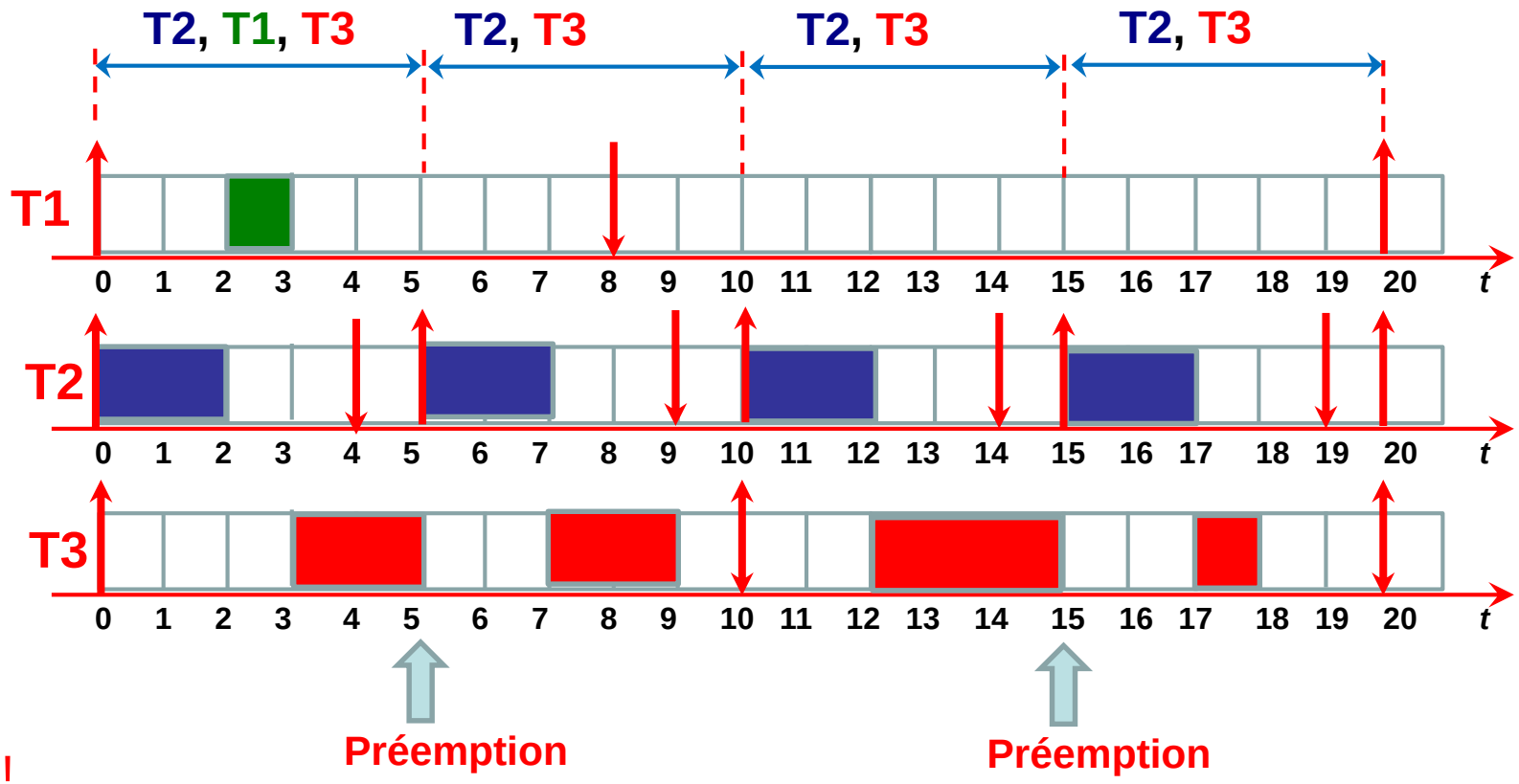
2) Test d'ordonnaçabilité:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} = \frac{1}{8} + \frac{2}{4} + \frac{4}{10} = 1.025 > 1$$

⇨ Peut être ordonnançable ou pas !

3) Cycle d'étude = [0 ,$PPCM(Pi)$] = $[0, 20]$

Priorités:



- **Avantages** :
  - Utilisation possible de 100% du processeur
  - Meilleur que RM pour les tâches de courte échéance
  - Optimal pour les ordonnancements à priorité dynamique si les échéances sont inférieures aux périodes
- **Inconvénients** :
  - Légère complexité de la mise en œuvre
  - Moins répandu dans les exécutifs que RM
  - Mauvais comportement en cas de surcharge

7.5. Algorithme à priorité variable (LLF : Least Laxity First):

- **Propriétés**:
  - Ordonnancement dynamique préemptif basé sur les priorités.
  - Les priorités sont mises à jour à chaque instant: il y a plus de changements de contexte que dans le cas EDF.
  - Optimal aussi bien pour les tâches périodiques qu'apériodiques.
  - La tâche à qui il reste le moins de marge s'exécute d'abord.
  - Autrement dit, la priorité maximale est donnée à la tâche qui a la plus petite laxité résiduelle *L*(t) : celle qui présente une urgence à s'exécuter.

- **Laxité dynamique résiduelle**:

$$L(t) = D - t - RT$$

RT: Remaining execution time
t : temps actuel
D: Echéance

C'est le retard maximum pour reprendre l'exécution d'une tâche.

- **Test d'ordonnançabilité (LLF)** :
  - Pour *n* tâches synchrones à échéance sur requête: $\boldsymbol{D_i = P_i}$

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Condition exacte
(nécessaire + suffisante)

  - Pour les autres tâche avec: $\boldsymbol{D_i \leq P_i}$

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$$

Condition suffisante

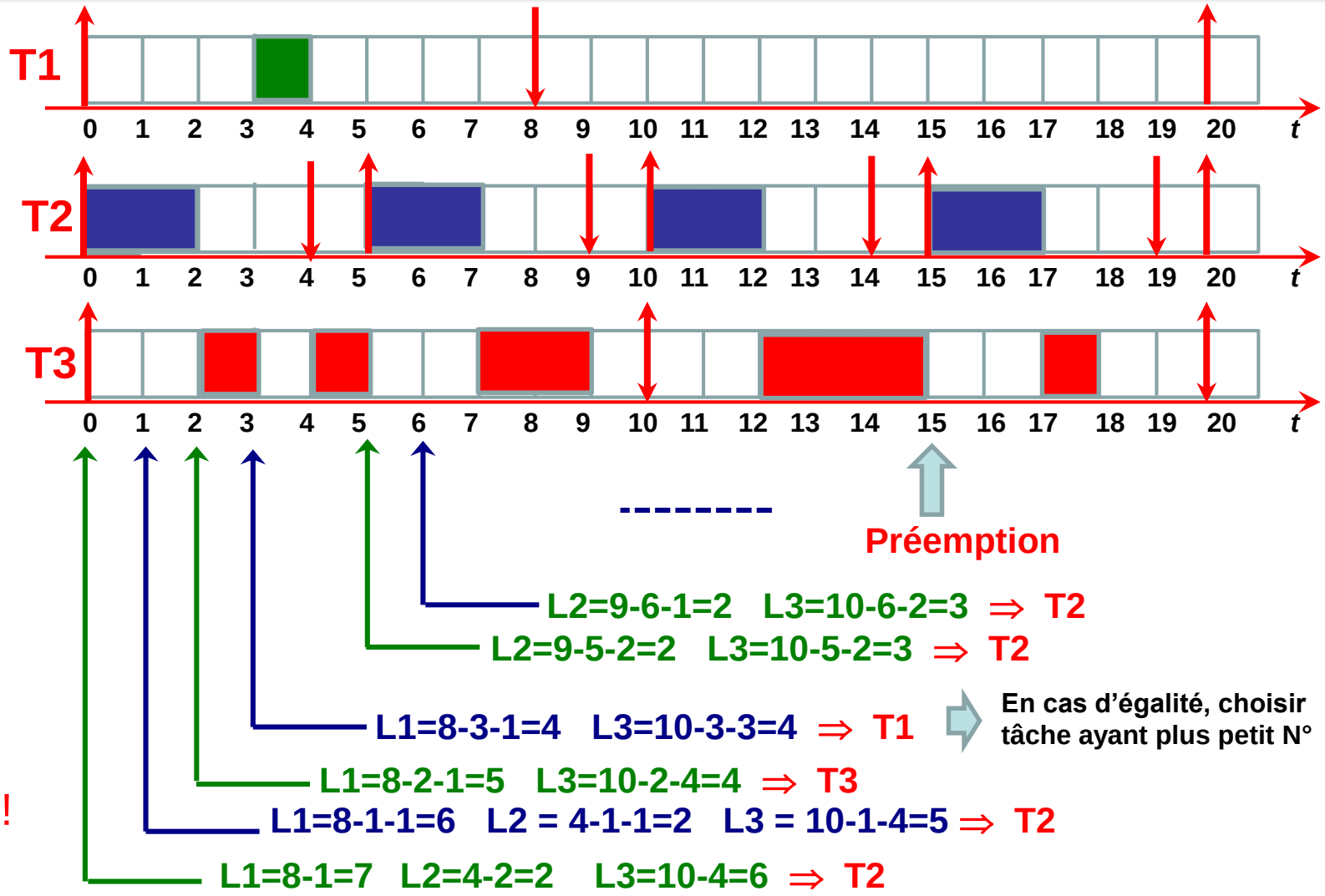
Exemple :

Tâche	Période Pi	Durée d'exécution Ci	Echéance Di
T1	20	1	8
T2	5	2	4
T3	10	4	10

1) Nous avons: $D_i \leq P_i$
2) Test d'ordonnaçabilité:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} = \frac{1}{8} + \frac{2}{4} + \frac{4}{10} = 1.025 > 1$$

⇨ Peut être ordonnançable ou pas !



- **Avantages** :
  - Meilleur qu'EDF dans le cas de multi-processeur
- **Inconvénients** :
  - Forte complexité de la mise en œuvre
  - Difficulté du calcul du temps de calcul restant
  - Mauvais comportement en cas de surcharge
  - Nombre de préemptions engendrées supérieur

Exercice :

Tracer le diagramme d'ordonnancement des tâches ci-contre pour un algoritme LLF.

Tâche	Temps d'arrivée	Durée d'exécution Ci	Echéance Di
T1	0	10	33
T2	4	3	28
T3	5	10	29

Exercice :

Soit les trois tâches indépendantes et préemptibles, toutes prêtes à la date t=0 :

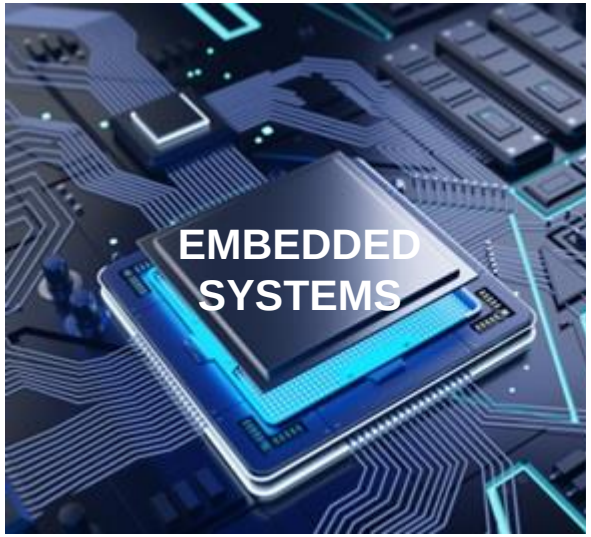
- T1 (C=1, D=P=3),
- T2 (C=1, D=P=4),
- T3 (C=2, D=P=6).

Pour chacune des politiques d'ordonnancement RM, EDF et LLF :

1. Calculer U, le facteur d'utilisation du processeur, que peut-on en conclure ?
2. Donner un schéma du séquencement des tâches.

8. Tableau récapitulatif des différents algorithmes

Caractéristique	FCFS (First-Come, First-Served)	SJF (Shortest Job First)	RR (Round Robin)	RM (Rate Monotonic)	DM (Deadline Monotonic)	EDF (Earliest Deadline First)	LLF (Least Laxity First)
Type	Statique	Statique	Statique	Statique	Statique	Dynamique	Dynamique
Priorité	Fixe	Fixe	Fixe	Fixe	Fixe	Dynamique	Dynamique
Critère de Priorité	Ordre d'arrivée	Durée de tâche	Tourniquet	Période	Échéance	Échéance	Laxité
Préemption	Non	Non	Oui	Oui	Oui	Oui	Oui
Complexité	Très simple	Moyenne	Simple	Moyenne	Moyenne	Moyenne à complexe	Complexe
Utilisation Optimale des Ressources	Faible	Haute	Moyenne à faible	Haute	Haute	Optimale	Très haute



FIN !
Questions ?