Informatique industrielle A7-19571
Systèmes temps-réel

J.F.Peyre

Partie III : Ordonnancement temps réel

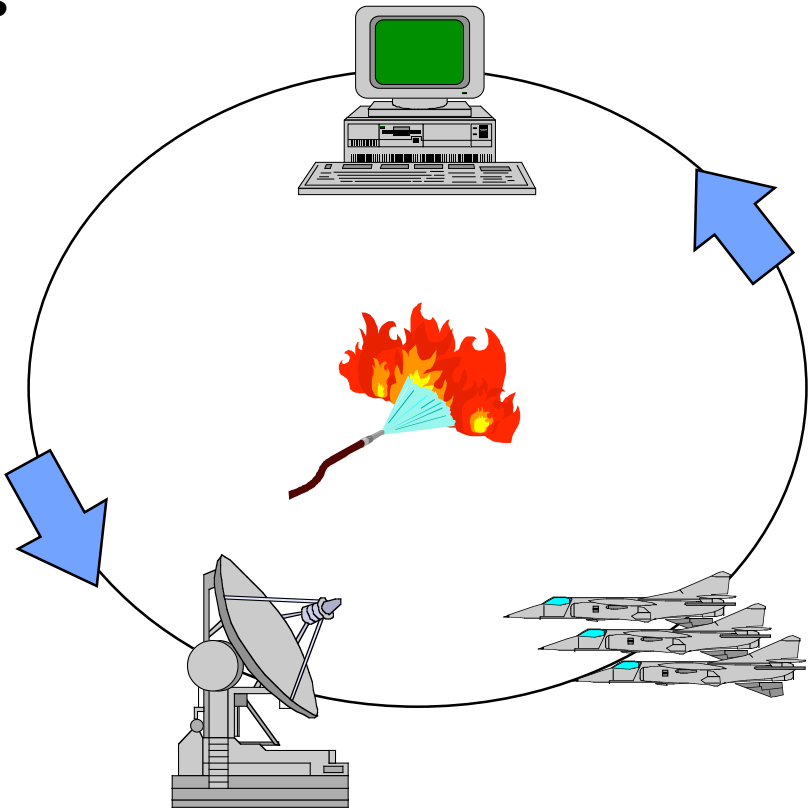
Caractéristiques de l'ordonnancement temps réel

Plan

- Caractéristiques de l'ordonnancement temps-réel
- Ordonnancement des tâches périodiques indépendantes
- Ordonnancement des tâches périodiques dépendantes
- Conclusion

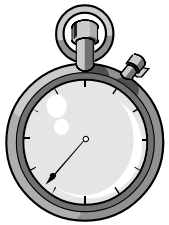
Contexte applicatif

Application de contrôle multitâches



Commandes
Fonction de pilotage

Mesures
Evénements
Fonction de suivi



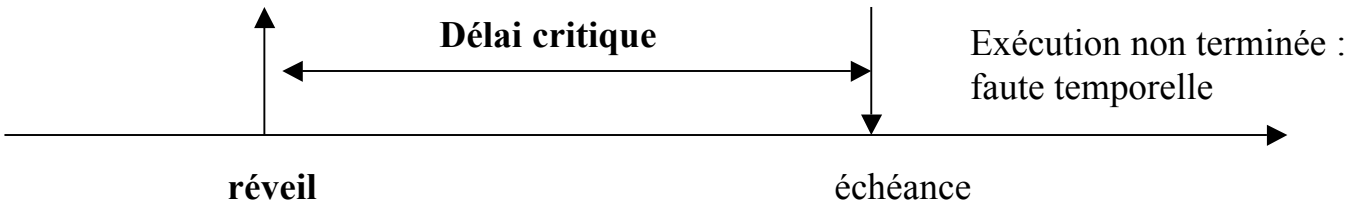
Contraintes de temps

But principal de l'ordonnancement

- **Permettre le respect des contraintes temporelles associées à l'application et aux tâches.**

- **Chaque tâche possède un** délai critique **: temps maximal pour s'exécuter depuis sa date de réveil. La date butoir résultante est appelée** échéance**.**

- **Le dépassement d'une échéance est appelé** faute temporelle**.**

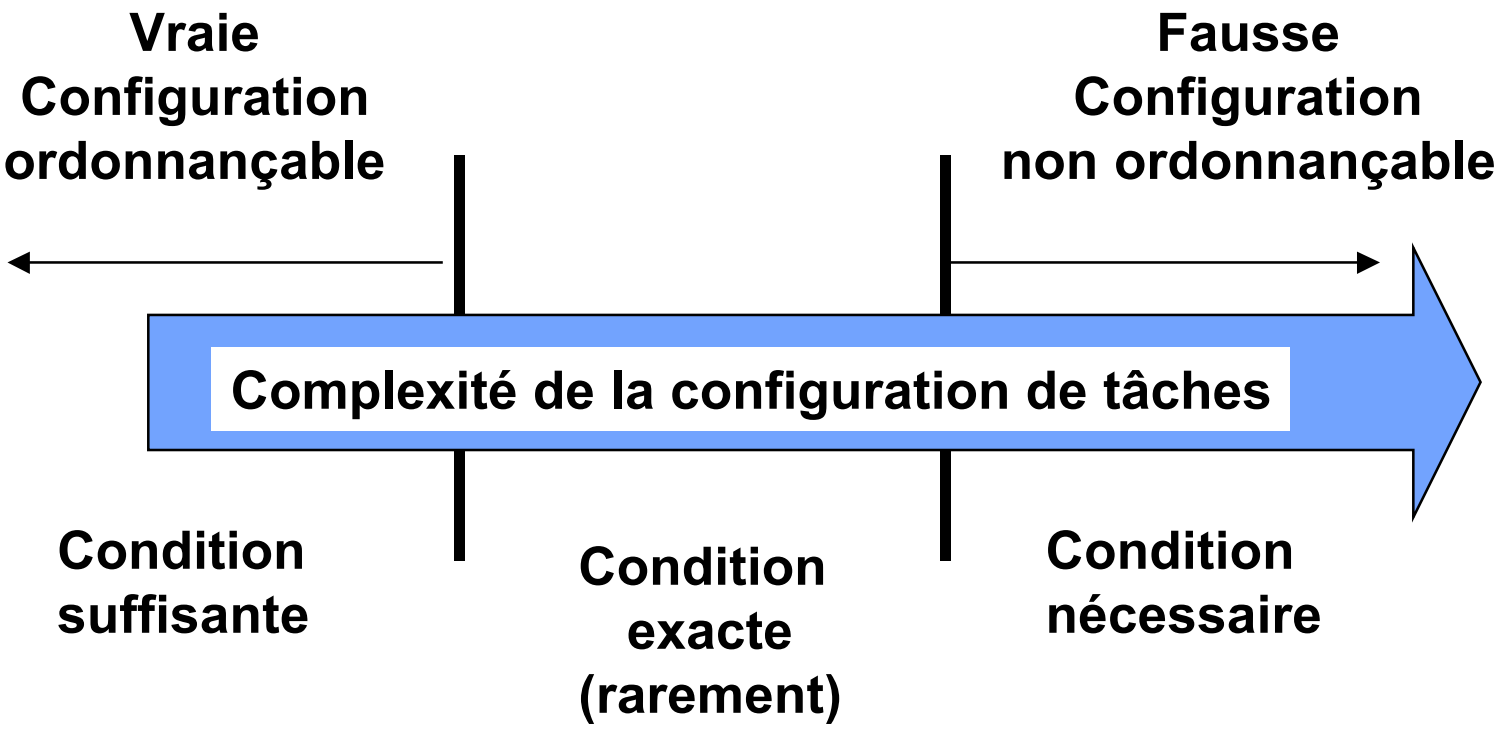


Ordonnançabilité

- Applications embarquées et critiques : **nécessité de certifier l'ordonnancement réalisé, c'est-à-dire de vérifier avant le lancement de l'application (hors ligne) le respect des contraintes temporelles.**

- **Cette certification s'effectue à l'aide de** tests d'acceptabilité **qui prennent en compte les paramètres temporels des tâches (temps d'exécutions Test d'acceptabilité des tâches) .**

Ordonnançabilité (suite)



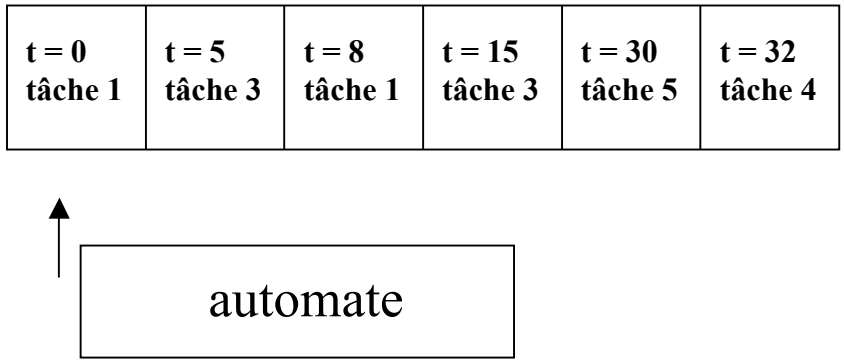
Ordonnançabilité (suite)

- **Tests d'acceptabilité : utilisent les temps d'exécution des tâches.**
  - ☞Il faut pouvoir déterminer et borner ces temps
  - ☞L'exécutif doit être **déterministe**
- Un exécutif *déterministe* **est un exécutif pour lequel les temps de certaines opérations système et matérielles élémentaires peuvent être bornés : temps de commutation, temps de prise en compte des interruptions, etc…**

Ordonnancement hors ligne

- **Un ordonnancement hors ligne établit avant le lancement de l'application une séquence fixe d'exécution des tâches à partir de tous les paramètres de celles-ci.**

- **Cette séquence est rangée dans une table et exécutée en ligne par un automate**

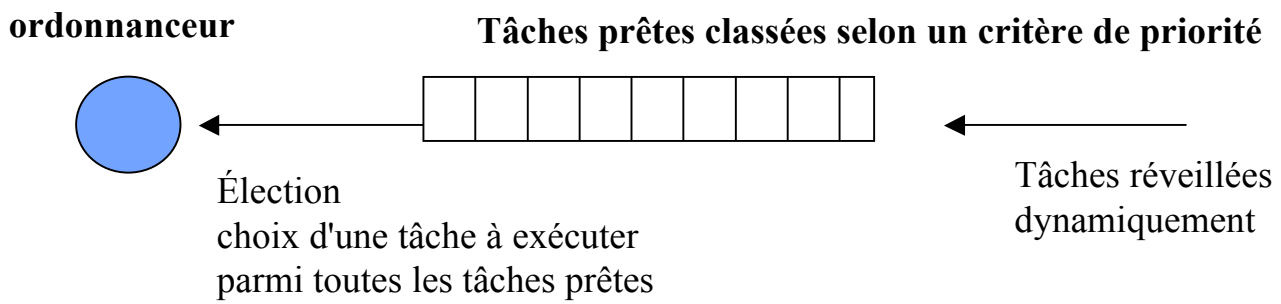


Séquence construite hors ligne

Ordonnancement en ligne

- **La séquence d'exécution des tâches est établie dynamiquement par l'ordonnanceur au cours de la vie de l'application en fonction des événements qui surviennent.**

- **L'ordonnanceur choisit le prochaine tâche à élire en fonction d'un critère de priorité.**



Modélisation des tâches périodiques

- **Les tâches périodiques correspondent aux mesures sur le procédé ; elles se réveillent régulièrement (toutes les P unités de temps)**

  ☞ périodiques strictes : contraintes temporelles dures à respecter absolument

  ☞périodiques relatives : contraintes temporelles molles qui peuvent être non respectées de temps à autre (sans échéance)

  ☞périodiques à échéance sur requête (délai critique = période)

Modélisation des tâches périodiques strictes

Tp (r_0, C, R, P) $0 \leq C \leq R \leq P$

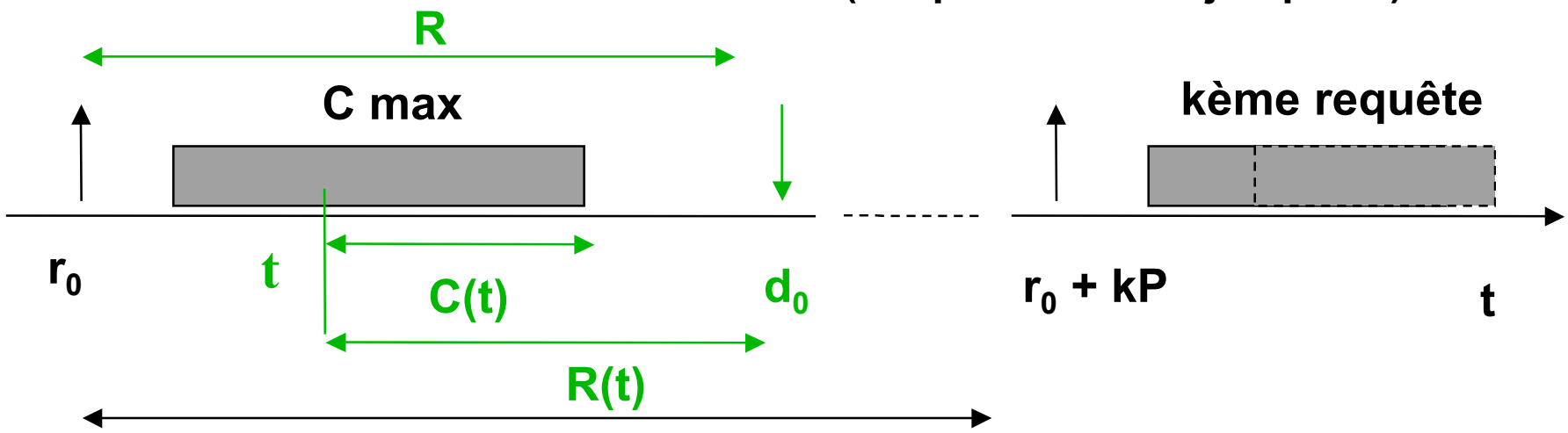
Tp (t, C(t), R(t))

R = P, à échéance sur requête

$d_k = r_{k+1}$

- r_0, date de premier réveil
- P, période
- r_k, date de réveil de la kème requête

  $r_k = r_0 + kP$
- C, temps d'exécution
- R, délai critique
- d_k, échéance = r_k + R
- C(t) : temps d'exécution restant à t
- R(t) : délai critique dynamique (temps restant à t jusqu'à d)



Modélisation des tâches apériodiques

- **Les tâches apériodiques correspondent aux événements**

- **Elles se réveillent de manière aléatoire**

  ☞ apériodiques strictes : contraintes temporelles dures à respecter absolument

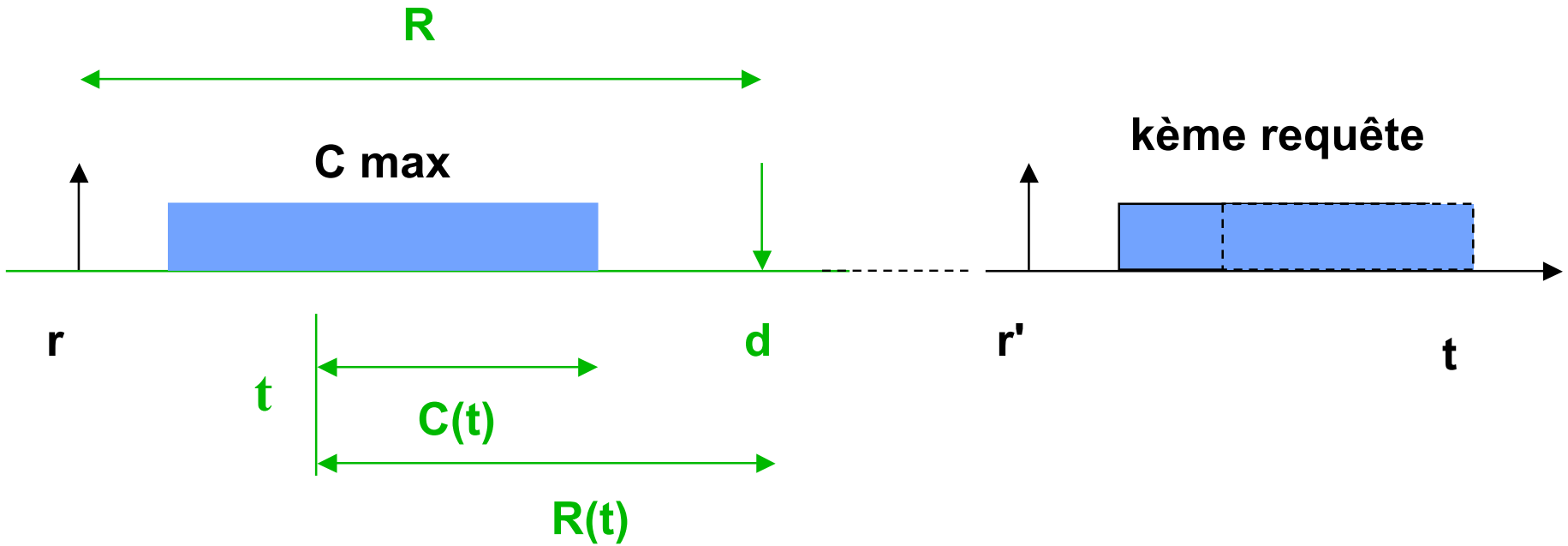
  ☞apériodiques relatives : contraintes temporelles molles qui peuvent être non respectées de temps à autre (sans échéance)

Modèlisation des tâches apériodiques strictes

Tap (r, C, R)

Tap (t, C(t), R(t))

- r, date aléatoire de réveil
- C, temps d'exécution
- R, délai critique
- d_k, échéance = r_k + R
- C(t) : temps d'exécution restant à t
- R(t) : délai critique dynamique (temps restant à t jusqu'à d)



Ordonnancement des tâches périodiques indépendantes

Principes

- **Algorithmes en ligne préemptifs + test d'acceptabilité évaluable hors ligne**
- **Les priorités affectées aux tâches sont soit constantes (évaluées hors ligne et fixes par la suite), soit dynamiques (elles changent dans la vie de la tâche)**
- **L'ordonnancement d'un ensemble de tâches périodiques est cyclique et la séquence se répète de manière similaire sur ce que l'on appelle la période d'étude.**
- **Pour un ensemble de tâches à départ simultanée (t = 0), la période d'étude est l'intervalle [0, PPCM(P_i)]**

Ordonnancement des tâches périodiques indépendantes (suite)

- **Trois algorithmes possibles**
  - Rate Monotonic (priorité constante)
  - Inverse deadline (priorité constante)
  - Earliest deadline (priorité dynamique)

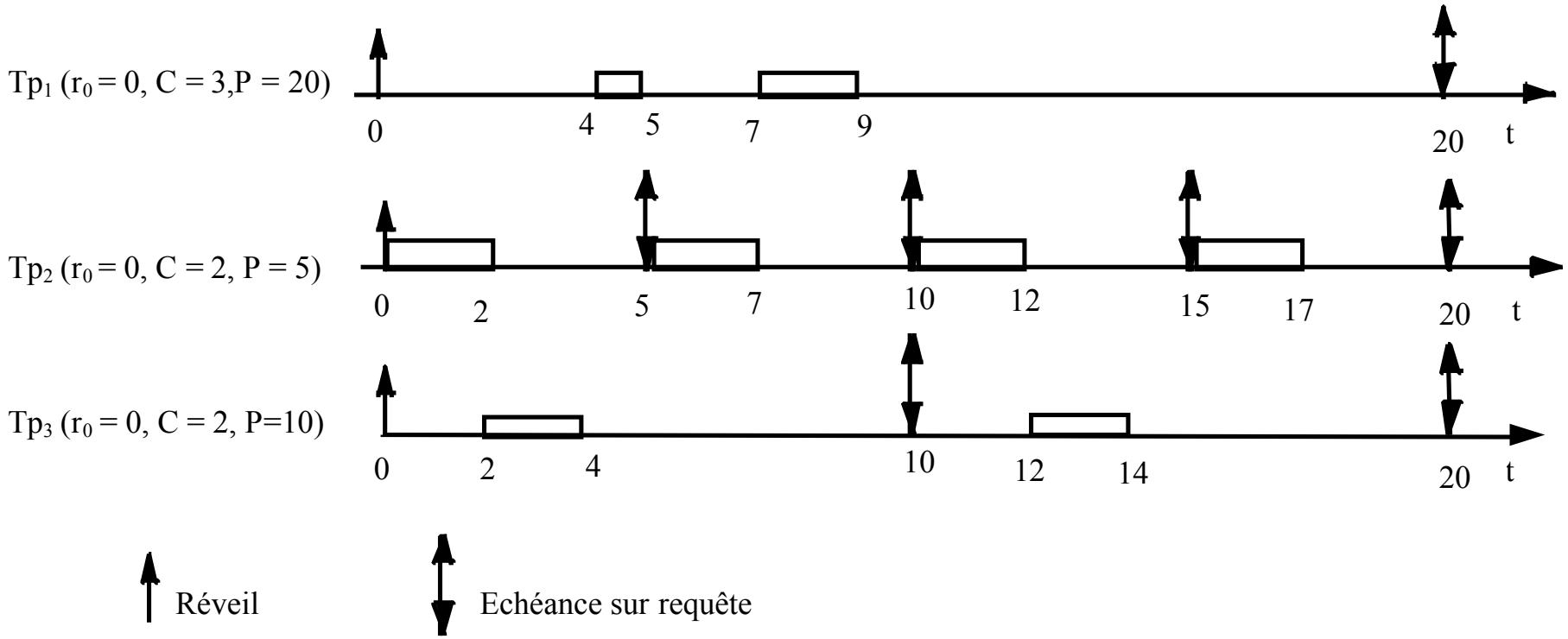
Algorithme Rate Monotonic

- Priorité de la tâche fonction de sa période. Priorité constante
- La tâche de plus petite période est la tâche la plus prioritaire
- Pour un ensemble de n tâches périodiques à échéance sur requête Tp_i (r_0, C_i, P_i), un test d'acceptabilité est la condition suffisante :

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

Algorithme Rate Monotonic (suite)

Exemple d'ordonnancement



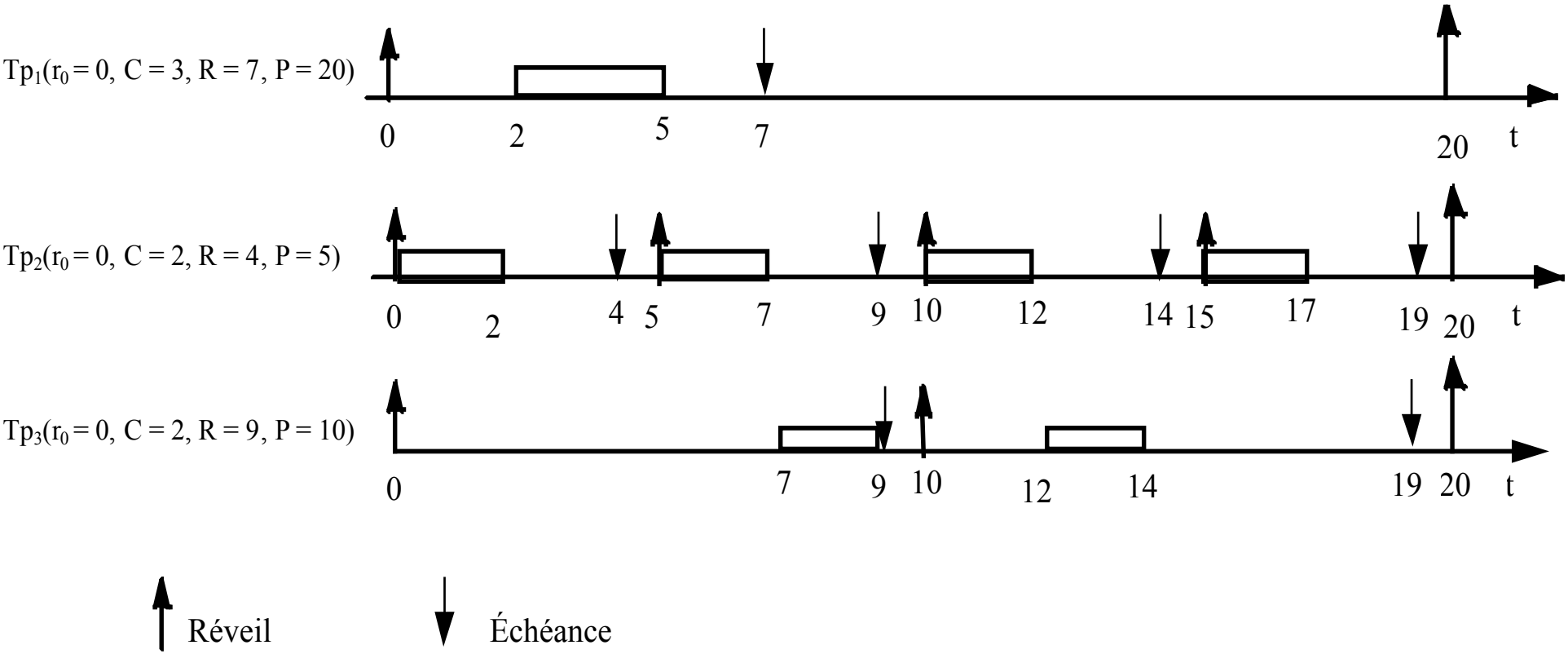
Algorithme Inverse Deadline

- Priorité de la tâche fonction de son délai critique. Priorité constante
- La tâche de plus petit délai critique est la tâche la plus prioritaire
- Pour un ensemble de n tâches périodiques à échéance sur requête Tp_i (r_0, C_i, R_i, P_i), un test d'acceptabilité est la condition suffisante :

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

Algorithme Inverse Deadline (suite)

Exemple d'ordonnancement



Algorithme Earliest Deadline

- **Priorité de la tâche fonction de son délai critique dynamique. Priorité dynamique**
- **A t, la tâche de plus petit délai critique dynamique (de plus proche échéance) est la tâche la plus prioritaire**

CNS (tâches ER)

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

CS (tâches quelconques)

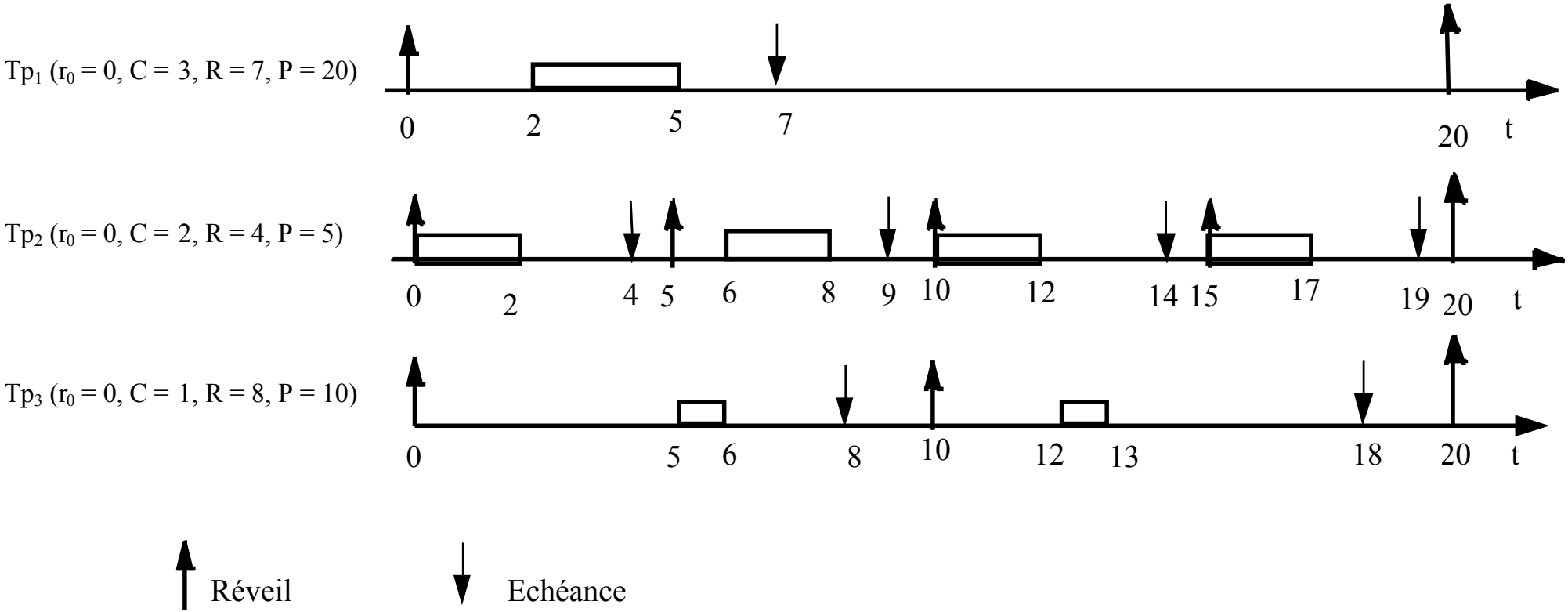
$$\sum_{i=1}^{n} \frac{C_i}{R_i} \leq 1$$

CN (tâches quelconques)

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

Algorithme Earliest Deadline (suite)

Exemple d'ordonnancement



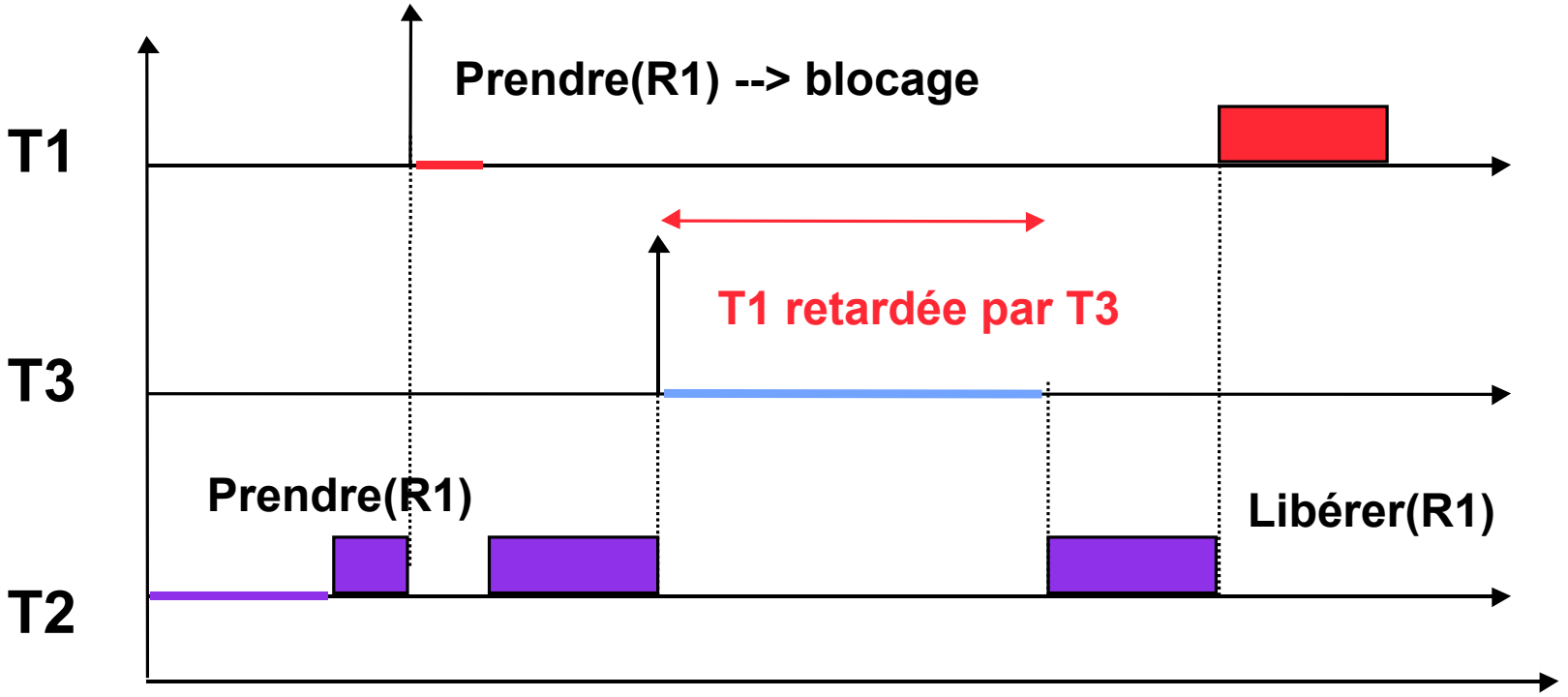
Ordonnancement des tâches périodiques avec des dépendances

Problème de l'inversion de priorité

Prio (T1) > Prio (T3) > Prio (T2), T1 et T2 utilisent R1, mais pas T3

R1 est une ressource non partageable

T1 est retardée par toutes les tâches de priorité intermédiaire



Problème de l'inversion de priorité (suite)

- **L'inversion de priorité est la situation pour laquelle une tâche de priorité intermédiaire (T3) s'exécute à la place d'une tâche de forte priorité (T1) parce que la tâche de forte priorité (T1) est en attente d'une ressource acquise par une tâche de plus faible priorité (T2).**

- **A priori, on ne peut pas borner le temps d'attente de la tâche de haute priorité qui risque ainsi de dépasser son échéance car l'exécution de T3 n'était pas prévisible.**

Problème de l'inversion de priorité (suite)

- **Solutions mises en œuvre :**
  - Borner le temps d'attente des tâches sur l'accès aux ressources.
  - Calcul de bornes B, qui peuvent ensuite être ajoutées au temps d'exécution des tâches et ainsi être intégrées dans les tests d'acceptation des configurations.

- **Deux protocoles principaux :**
  - le protocole de **l'héritage de priorité**
  - le protocole de' la **priorité plafonnée**

Protocole de l'héritage de priorité

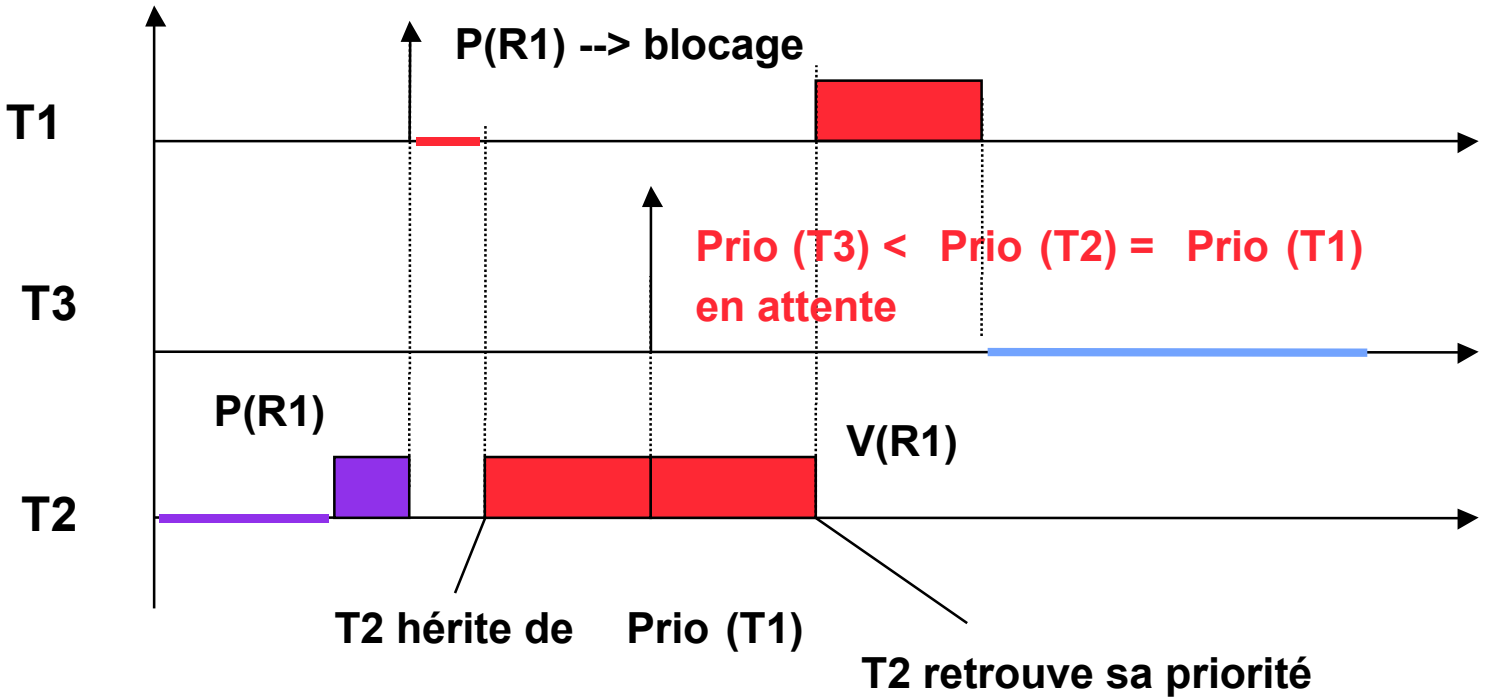
- **Principe : une tâche T détentrice d'une ressource R hérite de la priorité de la tâche la plus prioritaire mise en attente sur la ressource R; après avoir libéré R, T retrouve sa priorité initiale**

- **Ainsi T est ordonnancée au plus vite pour libérer le plus rapidement possible la ressource R.**

- **Ce protocole ne prévient pas l'interblocage**

Protocole de l'héritage de priorité (suite)

➳ Une tâche en section critique hérite de la priorité de la plus haute tâche en attente sur la section critique



Protocole de la priorité plafonnée

- **Principe :**
  - Chaque ressource R possède une priorité qui est celle de la tâche de plus haute priorité pouvant demander son accès.
  - Une tâche T détentrice d'une ressource R hérite de la priorité des tâches T' plus prioritaires mises en attente sur cette ressource R, jusqu'à ce qu'elle libère la ressource R.
  - **Cependant, la tâche T ne peut obtenir la ressource R que si la priorité de R est strictement supérieure à celles de toutes les ressources déjà possédées par les autres tâches T".** Par ce biais, on prévient l'interblocage.

Protocole de la priorité plafonnée (suite)