

# Les Systèmes Temps Réel

## Systèmes d'Exploitation Temps Réel

Linux Temps Réel

RTAI: Real Time Application Interface

H. Demmou

# Première partie

1. Définition et caractéristiques des STR
2. Système d'exploitation temps réel (SETR)
3. Ordonnancement
4. Ordonnancement de tâches dépendantes
  - a. Inversion de priorité
  - b. Algorithmes (héritage de priorité, PCP, IPCP)
5. Ordonnancement temps réel
  - a. Algorithme RMA et DM
  - b. Algorithme EDF
6. Calcul du temps de réponse maximum (WCRT)
7. Calcul du temps d'exécution maximum(WCET)

## Définition des systèmes temps réel

« Un système fonctionne en temps réel s'il est capable d'absorber toutes les informations d'entrée sans qu'elles soient trop vieilles pour l'intérêt qu'elles présentent, et de réagir suffisamment vite pour que cette action ait un sens »

« Le comportement correct d'un système temps réel ne dépend pas seulement des résultats logiques des traitements, mais dépend en plus de la date à laquelle ces résultats sont produits »

*Un système temps réel doit respecter les contraintes temporelles en réagissant suffisamment vite aux variations du monde extérieur et en élaborant la réponse adéquate dans un temps suffisamment court.*

## Définition des systèmes temps réel

IEEE (Institute of Electrical and Electronics Engineers)

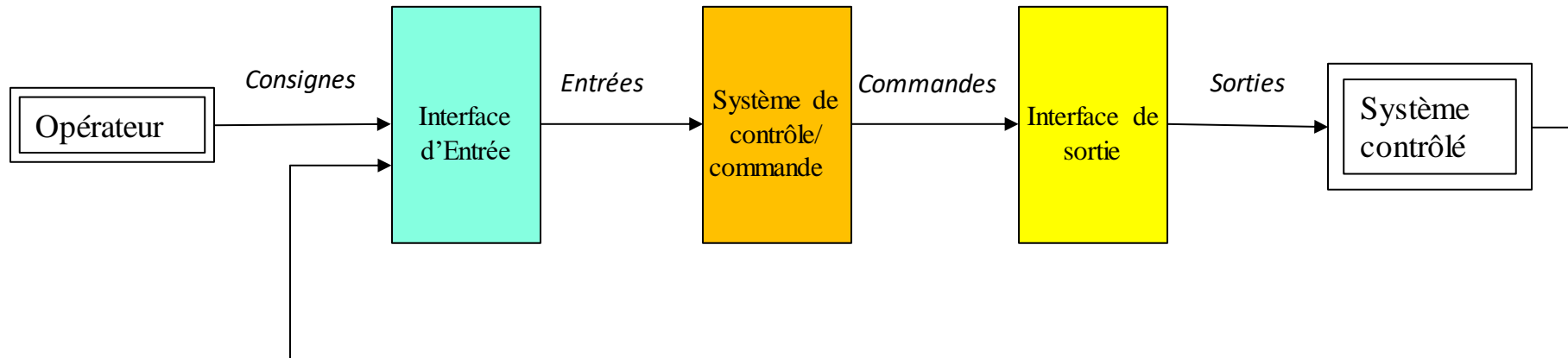
**Embedded System:** is a computing component integrated into a larger system with the purpose of managing its resources and monitoring/controlling its functions using special hardware devices.

**Real-Time System:** is a computing system whose correct behavior depends not only on the value of the computation but also on the time at which outputs are produced.

**Safety critical System:** is a computing system where a failure may cause injury, loss of lives, significant financial loss

**Reactive system :** is a computing system in continuous interaction with the environment (as opposed to information processing)

# systemes temps réel ? Réactif ? Embarqué ?



## Caractéristiques de systèmes Temps Réel

- Contraintes temporelles : respect des échéances temporelles
- Prédicibilité, déterminisme
- Horloge Temps Réel (HTR)
- Réactivité : réaction aux événements internes ou externes, synchrones ou asynchrones
- Sûreté de fonctionnement, Qualité de service

## Caractéristiques des systèmes Temps Réel

- Le temps est un facteur déterminant pour l'ordre d'exécution
- Les résultats du système dépendent du temps (comme variable ou durée d'exécution)
- Les performances sont liées au respect des contraintes temporelles
- les résultats du système dépendent du **temps logique** (ordre d'exécution) et du **temps physique** (date d'exécution)
- Un résultat juste mais **hors délai** est considéré **faux**

## Types de systèmes temps réel

le terme de **temps réel** implique des **contraintes temporelles (CT)**.

Le respect de ces contraintes est un facteur prépondérant pour évaluer la qualité d'un système temps réel.

L'échelle de ces contraintes varie selon le domaine d'application:

- télécommunications ( Microsecondes)
- robots, radars ( Millisecondes)
- procédés industriels ( secondes à qq minutes , qq heures)

Typologie:

Temps réel **souple , soft real time** (les CT sont des préférences, ou des performances liées à la qualité de service)

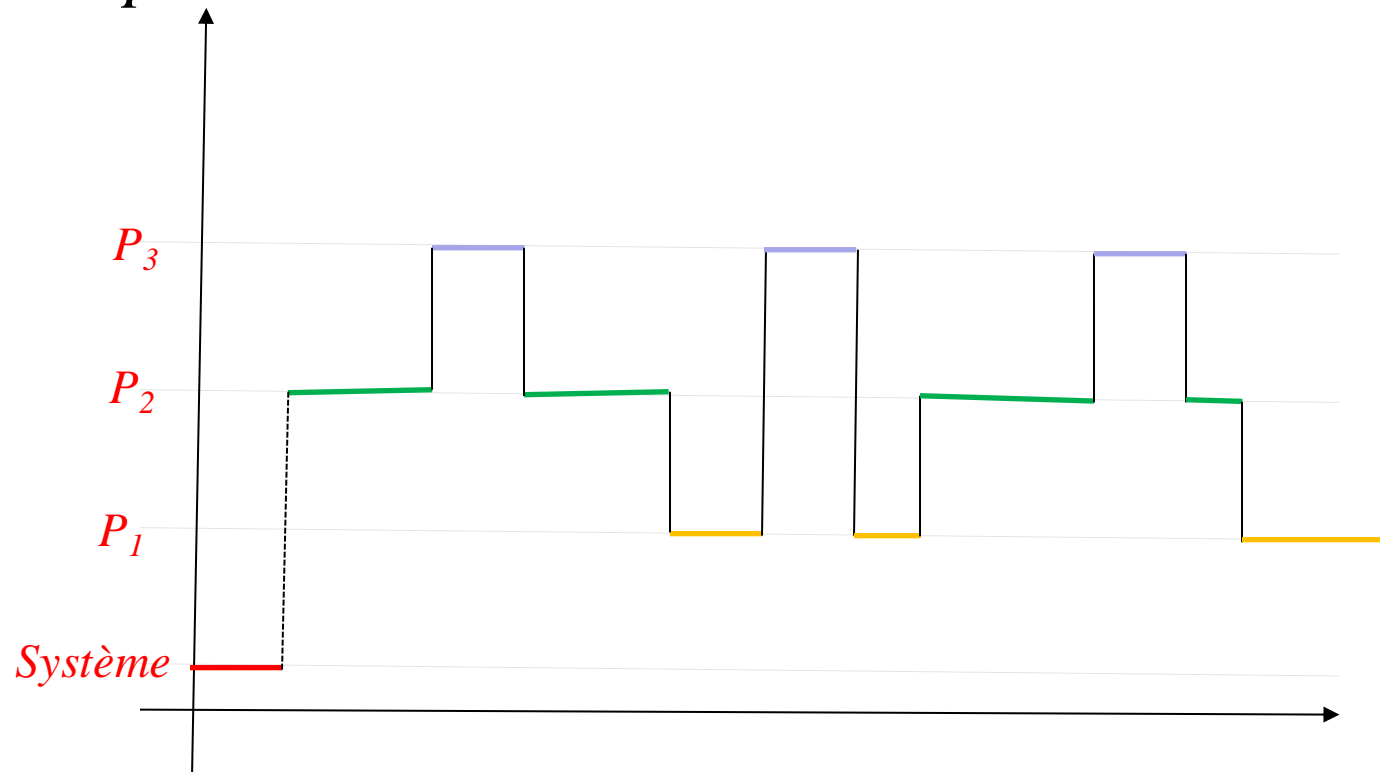
Temps réel **dur (ou strict) hard real time** (les CT expriment des exigences de sûreté de fonctionnement)



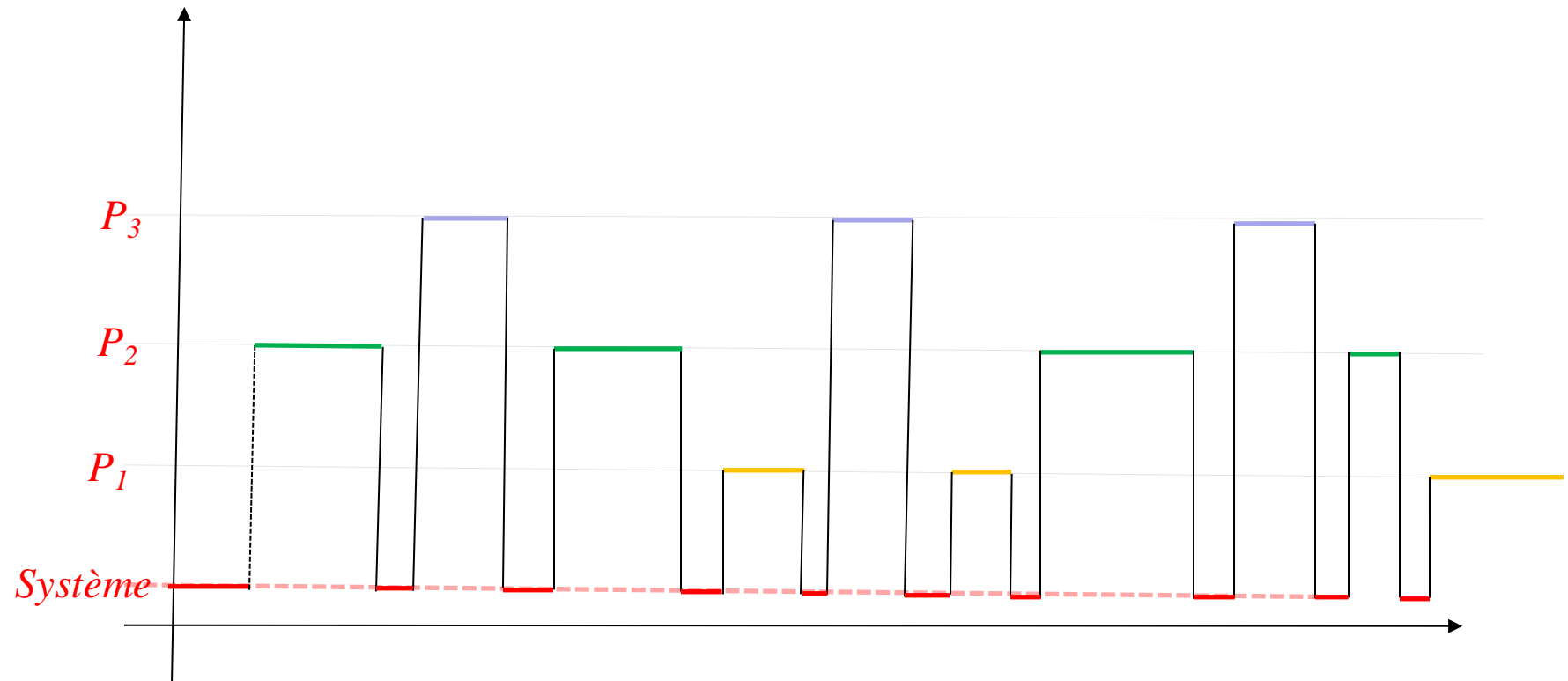
# Systèmes multitâches et systèmes temps réel

*Tâches, processus, thread: entités informatiques autonomes réalisant des activités (actions) de manière séquentielle*

*Multitâches, multiprocessus, multithread : Exécution concurrente ou pseudo parallélisme*



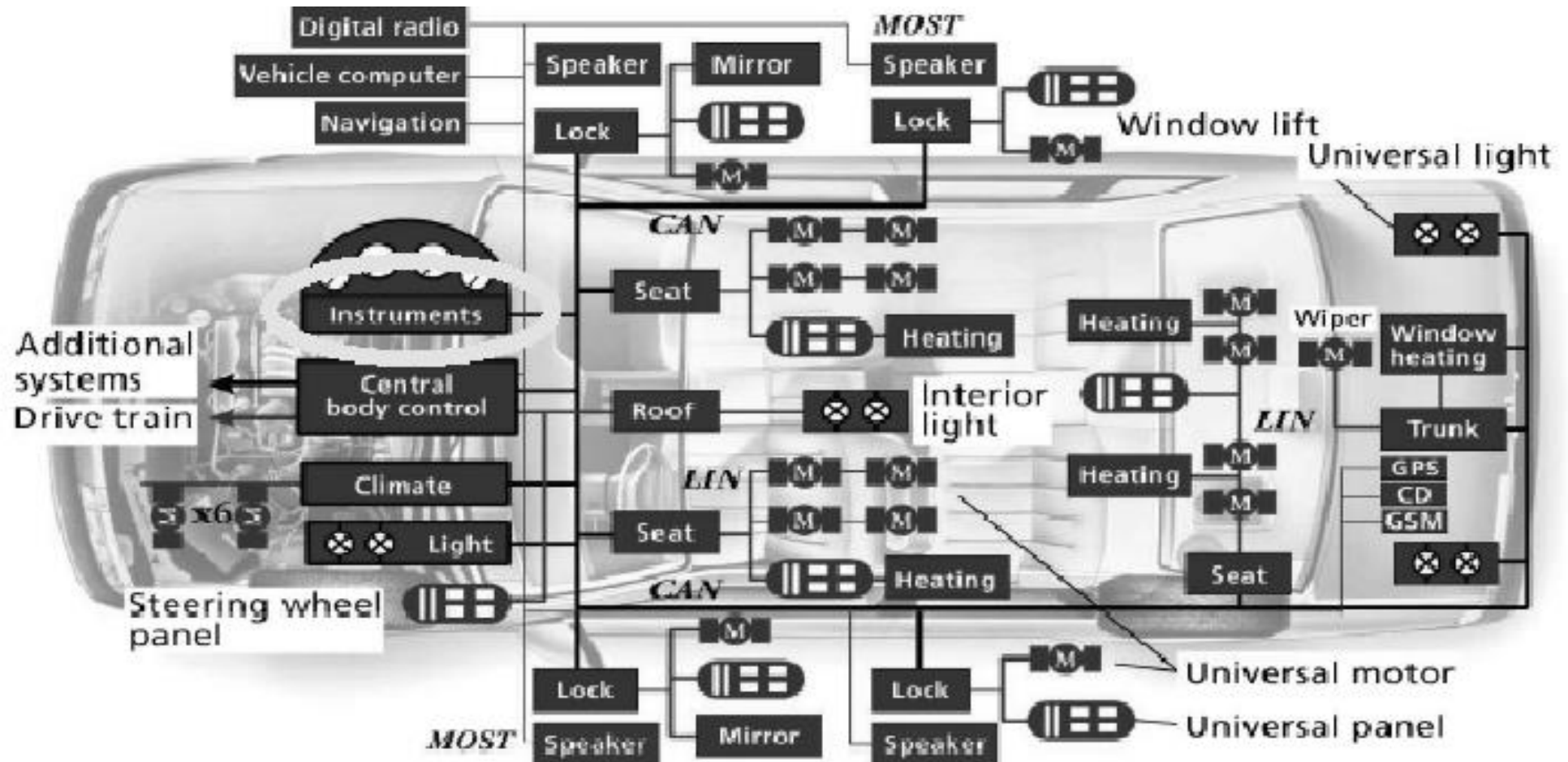
# Systèmes multitâches et systèmes temps réel



## Exemple véhicule automobile

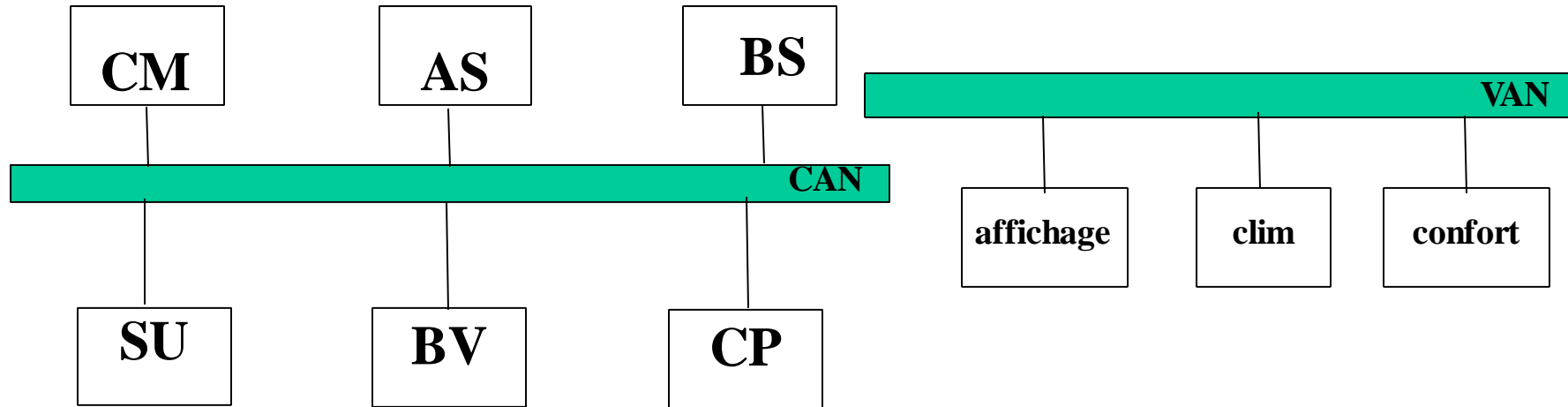


# Exemple véhicule automobile



CAN Controller area network  
 GPS Global Positioning System  
 GSM Global System for Mobile Communications  
 LIN Local interconnect network  
 MOST Media-oriented systems transport

# Exemple véhicule automobile



CM : calculateur du contrôle moteur

AS : calculateur de commande de l'ABS et du contrôle de stabilité

BS : calculateur du boîtier de servitude intelligent, passerelle avec le réseau VAN

SU : calculateur de contrôle de suspension automatique

BV : calculateur de boîte de vitesse automatique

CP : calculateur pour le correcteur de phare selon la mesure d'angle de volant

CAN : réseau local à diffusion avec protocole CSMA/CA ("collision avoidance")

VAN : réseau local pour les services de l'habitacle (affichage, climatisation, confort)

Les tâches et les messages sont tous à échéance sur requête (délai critique = période) et réveil sur période

# Exemple véhicule automobile

message	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
Émetteur	CM1	CP1	CM2	BV1	AS1	AS2	AS3	BS1	SU1	CM3	BV2	AS4
Récepteur	BS3 SU3	BV4 CM5 SU4	AS4	CM4	SU2	CM7 BS5	SU5	CM6 BV3	AS6 BS6 CP2	BS4	BS2	BS7
Période	10	14	20	15	20	40	15	50	20	100	50	100
Durée	8	3	3	2	5	5	4	5	4	7	5	1

	Calculateur du contrôle moteur (CM)							Calculateur boîte de vitesse (BV)			
Tâche	CM1	CM2	CM3	CM4	CM5	CM6	CM7	BV1	BV2	BV3	BV4
Période	10	20	100	15	15	50	40	15	50	50	14
Durée	2	2	2	2	2	2	2	2	2	2	2

## Mise en œuvre Sychrone

Hypothèse de synchronicité = Hypothèse d'instantanéité des actions (durée d'exécution nulle).



Événement activateur

pas de préemption

exécution séquentielle des tâches

les événements sont mémorisés pendant l'exécution

prise en compte différée des événements

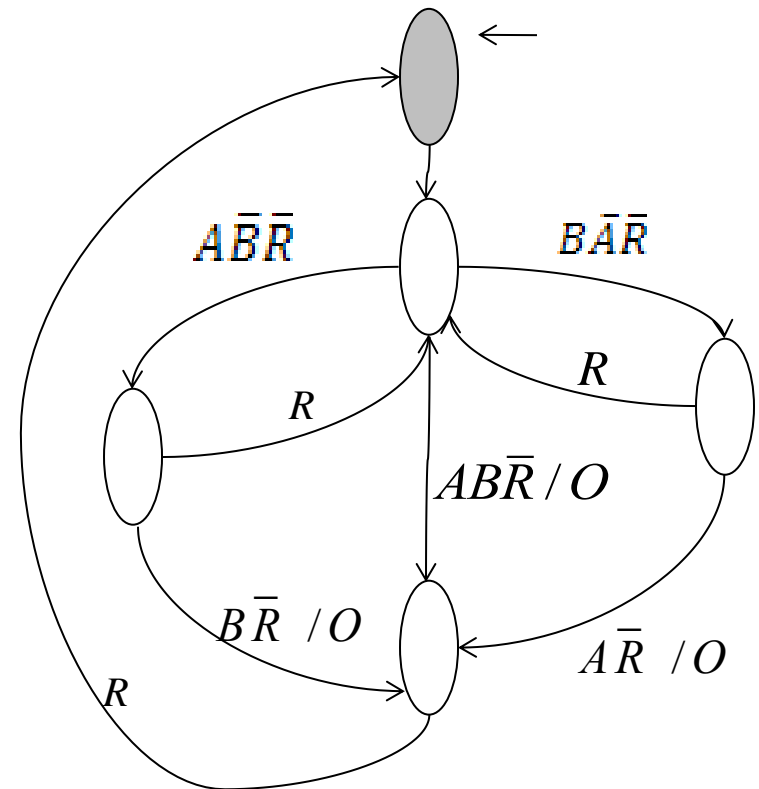
Résolution des problèmes liés au parallélisme et au temps réel doit être fait à la conception (ou compilation)

Langage : Lustre , Esterel, Signal

## Mise en œuvre Synchronise

Exemple avec le langage Esterel : Emettre le signal de sortie O dès l'occurrence des 2 signaux d'entrée A et B. Recommencer à chaque occurrence du signal d'entrée R

```
module ABRO:  
  input A,B,R;  
  output O;  
  loop  
    [await A||await B];  
    emit O;  
    await R  
  end loop  
end module
```



exemple de programme réactif



# Mise en œuvre Asynchrone

Les événements sont pris en compte au moment de leur occurrence.

Ordre et date d'arrivée des événements inconnus  
préemption autorisée  
politique d'ordonnancement des tâches

## Exécutif temps réel

gérer l'exécution des tâches en respectant des contraintes temps réel  
en fonction des événements

- externes (interruptions matérielles) générés par le procédé ou l'HTR
- internes , générés par les tâches en exécutant
  - des primitives du noyau (communication, synchronisation,...)
  - des instructions qui déclenchent des exceptions

# Les Systèmes d'Exploitation Temps Réel (RTOS)

## Fonctions principales

gérer la mémoire principale  
exécuter les commandes d'entrée-sortie  
gérer la mémoire secondaire  
gérer l'exécution des processus (en multiprogrammation)  
interpréter et exécuter les commandes de l'utilisateur

## Temps réel

- avoir un **temps de réponse** compatible avec l'évolution de l'environnement
- assurer le **déterminisme** temporel
- disposer de mécanismes d'interruption efficaces
- disposer de primitives de gestion des interactions entre les processus respectant des contraintes temporelles.

## RTOS commerciaux

- LynxOS <http://www.linuxworks.com>
- pSOS+ <http://www.windriver.com> (rachat)
- Qnx <http://www.qnx.com>
- RTLinux <http://www.fmslab.com> (IP acquise par VxWorks)
- VxWorks <http://www.windriver.com>

## RTOS open source

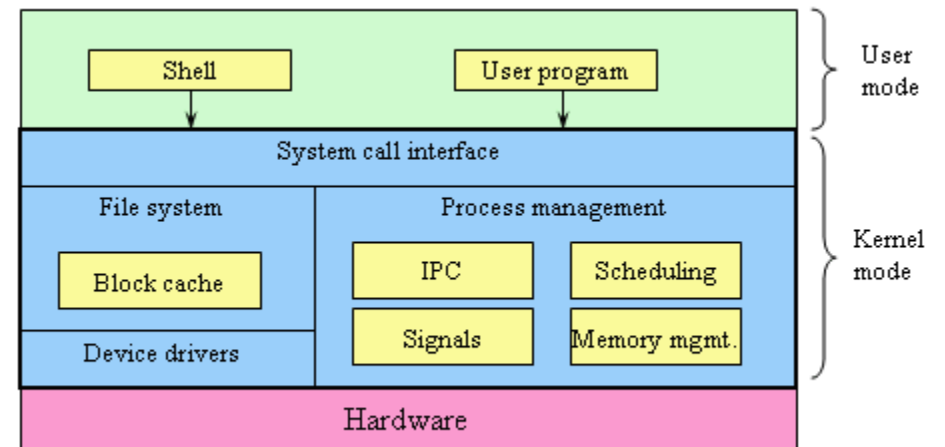
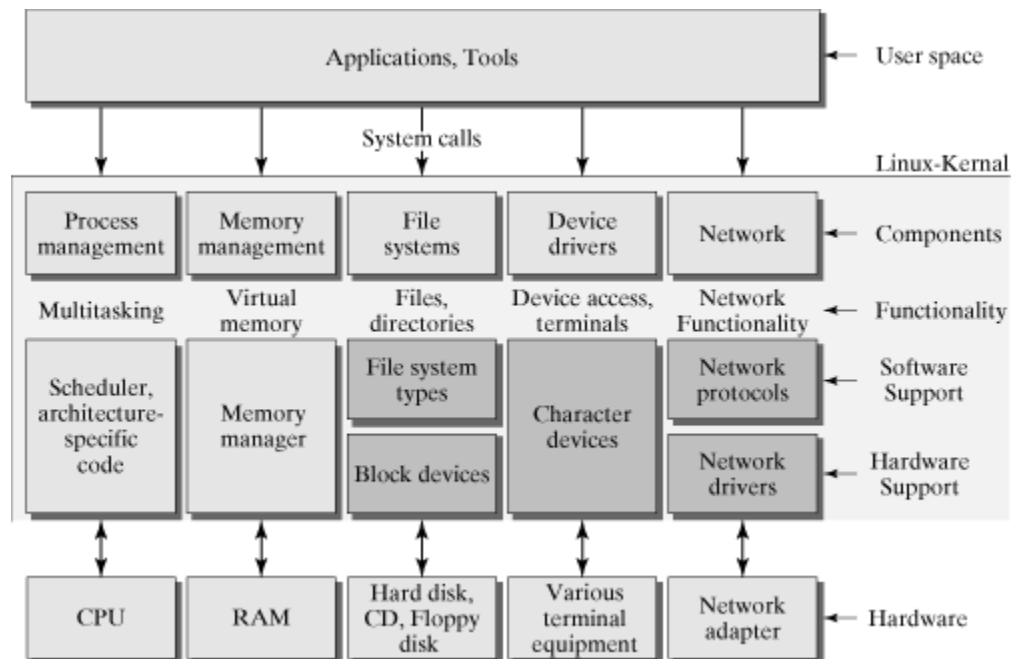
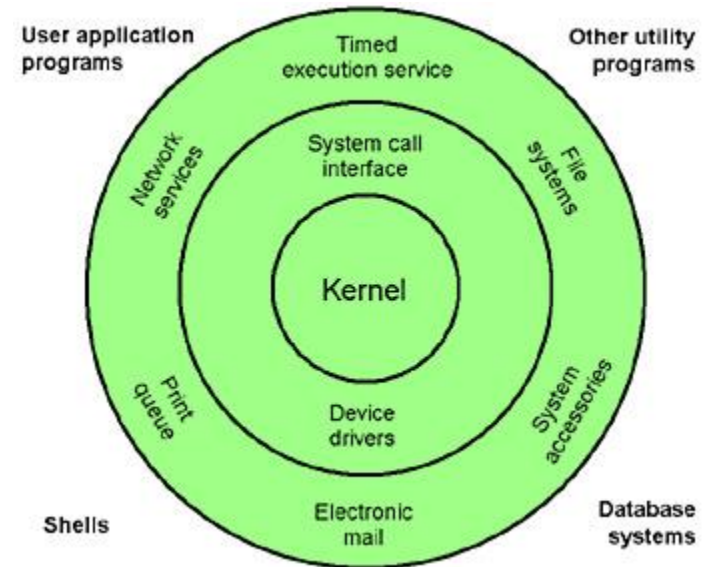
- eCos <http://ecos.sourceware.org>
- RTAI <http://www.rtai.org>
- uclinux <http://www.uclinux.org>
- XENOMAI <http://www.xenomai.org>
- FreeRTOS

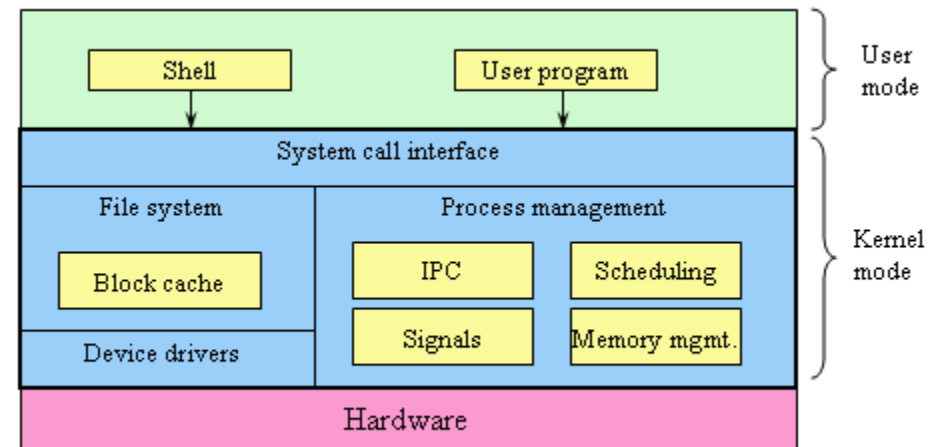
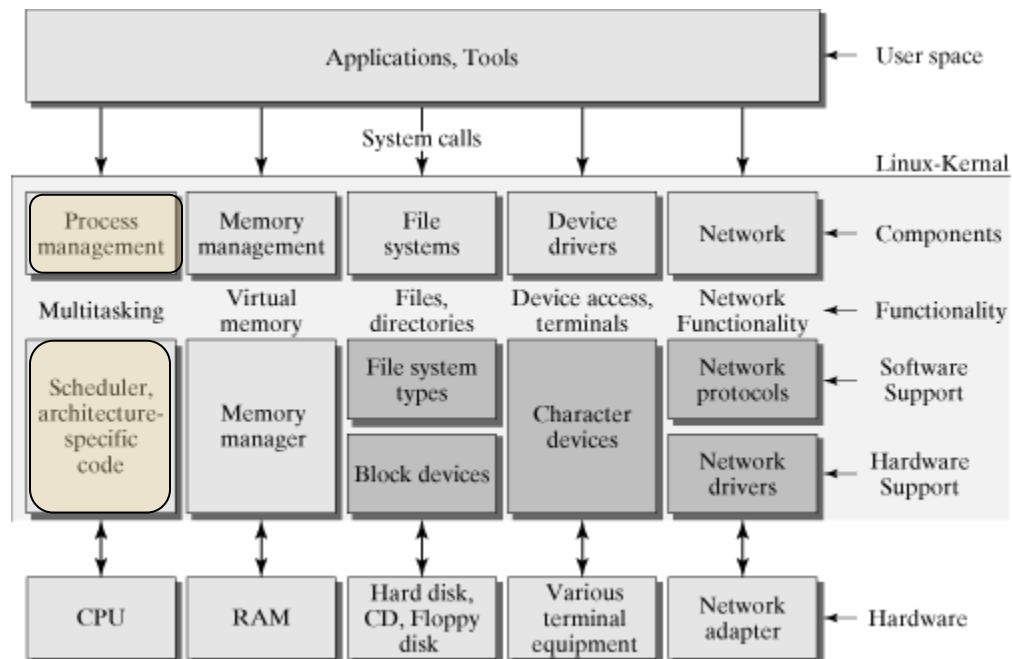
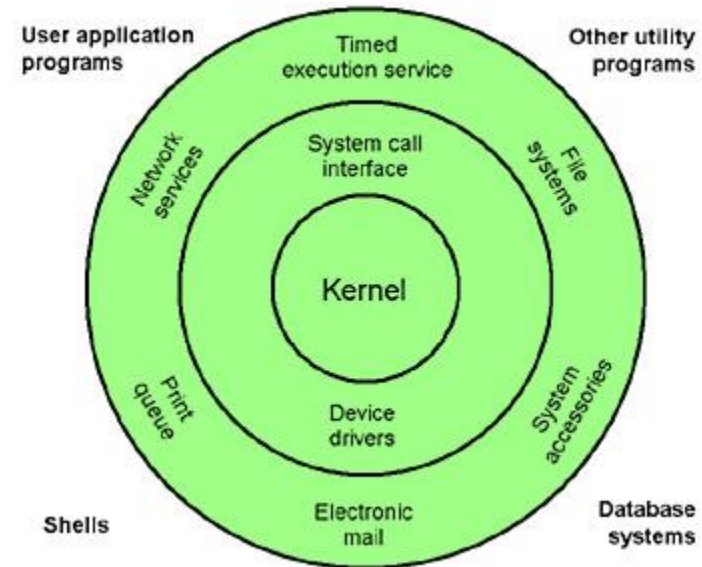
# Système d'exploitation, exécutif et Noyau temps réel

Un *noyau multitâche temps réel* fournit les primitives minimums pour la gestion de processus (tâches) en tenant compte des contraintes de temps réel, ainsi que la prise en compte des interruptions.

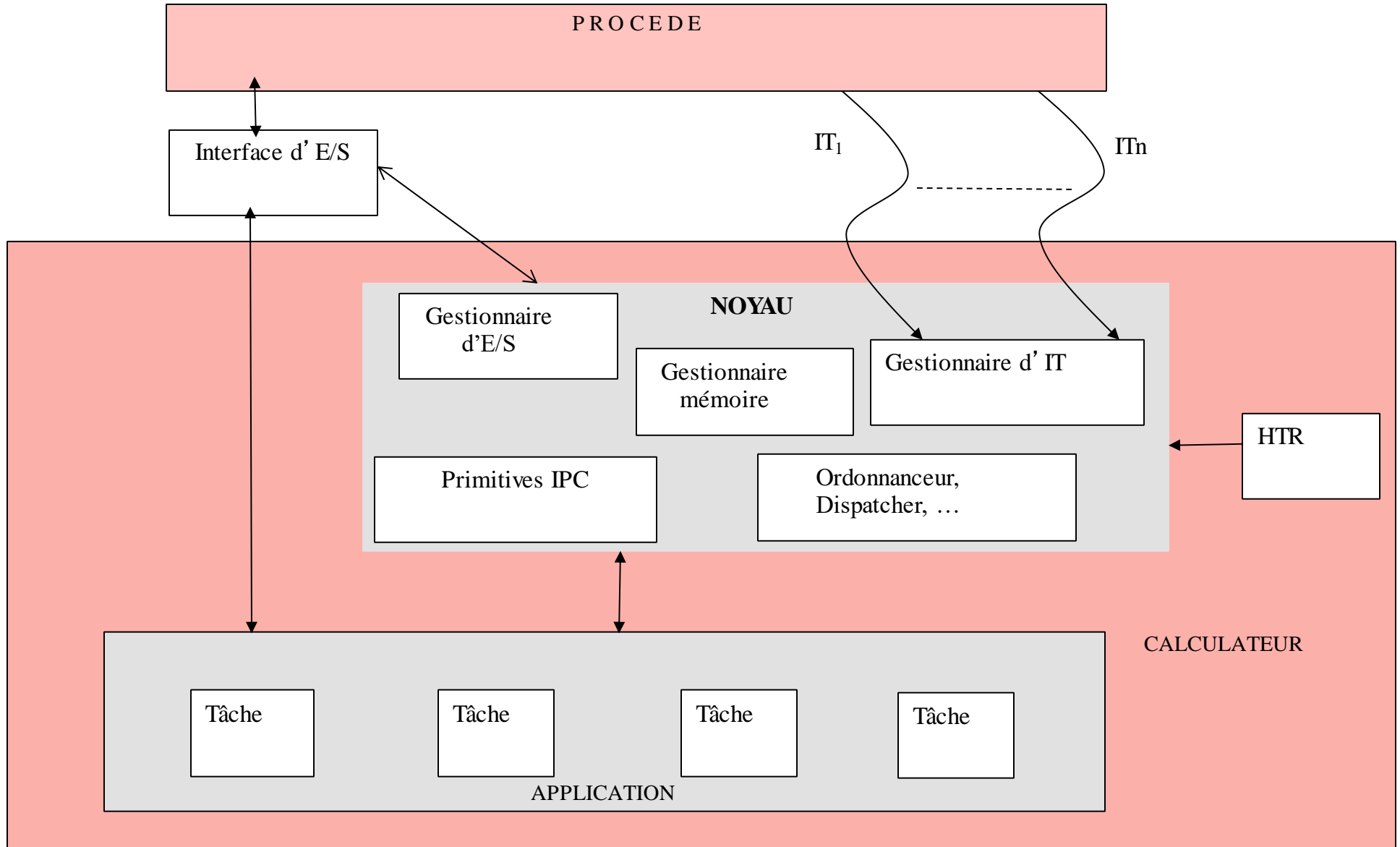
Un *exécutif temps réel* est constitué du noyau, augmenté d'un certain nombre de fonctionnalités (ou agences) (telle que gestion de la mémoire, gestion de fichiers, gestion d'entrées/sorties).

Un *système d'exploitation* fournit un support logiciel complet.( Utilitaires, environnement de développement, gestion réseau, Interface Utilisateur, etc.)





# Calculateur pour la commande



## Noyau Temps Réel

- **Interface d'entrée** : reçoit les requêtes, réalise la sauvegarde et aiguille les requêtes vers le point d'entrée correspondant (en général dans le gestionnaire d'objets).
- **Gestionnaire des objets**, gère l'état des différents objets en maintenant leur cohérence, en agissant sur des structures de données de type tableaux ou liste. La gestion de ces listes d'objets dépend du temps et des événements internes ou externes auxquels l'exécutif est réceptif.
- **L'Ordonnanceur**, chargé de gérer l'exécution des tâches en affectant le processeur de manière dynamique et réactive. Différents types d'algorithmes sont utilisés pour réaliser cette fonctionnalité.
- **Dispatcher (Lanceur)**, chargé d'activer la tâche active en restaurant son contexte d'exécution.
- **Gestionnaire d'interruptions**, qui intercepte les interruptions, les acquitte et effectue le traitement des interruptions (ce qui consiste à générer un événement en entrée de l'exécutif), avant de terminer la prise en compte.



# Les Objets d'un noyau

**La tâche:** C'est l'objet qui peut être actif dans l'application. C'est une séquence d'activités élémentaires (instructions en langage de programmation).

Le descripteur de tâches (Task ou Process Control Block) contient:

- Nom ou Numéro (identificateur)
- État
- Le pointeur de programme pour la tâche
- Les registres du CPU
- Informations sur la mémoire (limites, pages, segments,...)
- délai d'attente, durée d'utilisation processeur
- Priorité, Pointeurs sur les files d'ordonnancement
- États des entrées/sorties

**Les Messages, Boîtes aux lettres, Pipes, Sémaphores:** Ce sont des objets utilisés pour la synchronisation, la communication et le partage de ressources entre les éléments actifs de l'application.

Les objets liés au temps : **Timer, Horloge Temps Réel**

## Les états d'une tâche

**Actif** = la tâche possède toutes les ressources qui lui sont nécessaires pour s'exécuter

**Prêt** = la tâche attend la ressource processeur

**Suspendu** = la tâche est en attente d'un événement (interne , externe, horloge)

**Dormant** = existe mais n'a pas de requête pour l'exécutif (elle n'est donc pas gérée)

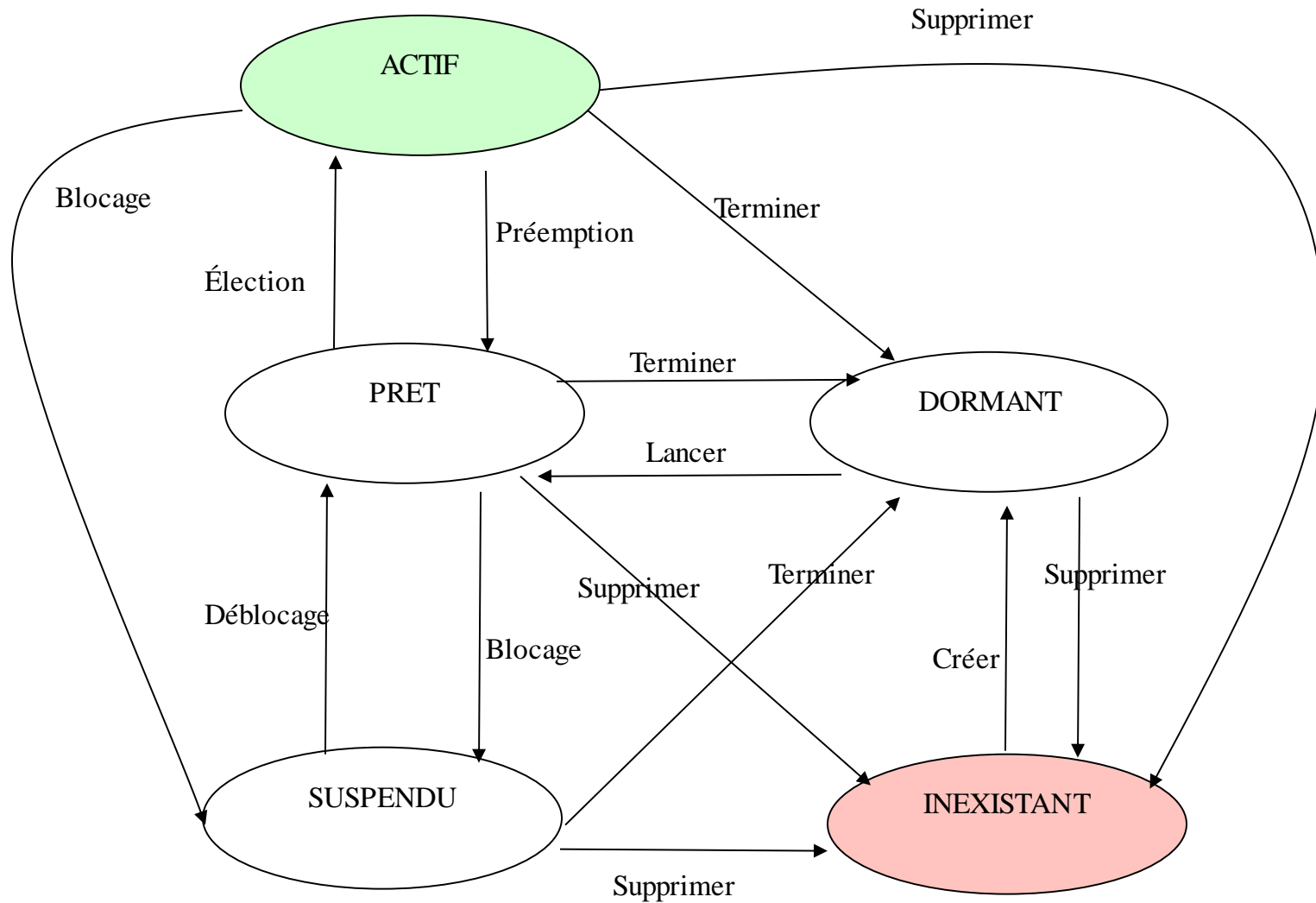
**Inexistant** = la tâche n'est pas créée

Le changement d'état d'une tâche est soit la conséquence d'un événement externe (IT ou HTR) soit celle d'une requête interne (appel système), soit celle de l'ordonnancement.

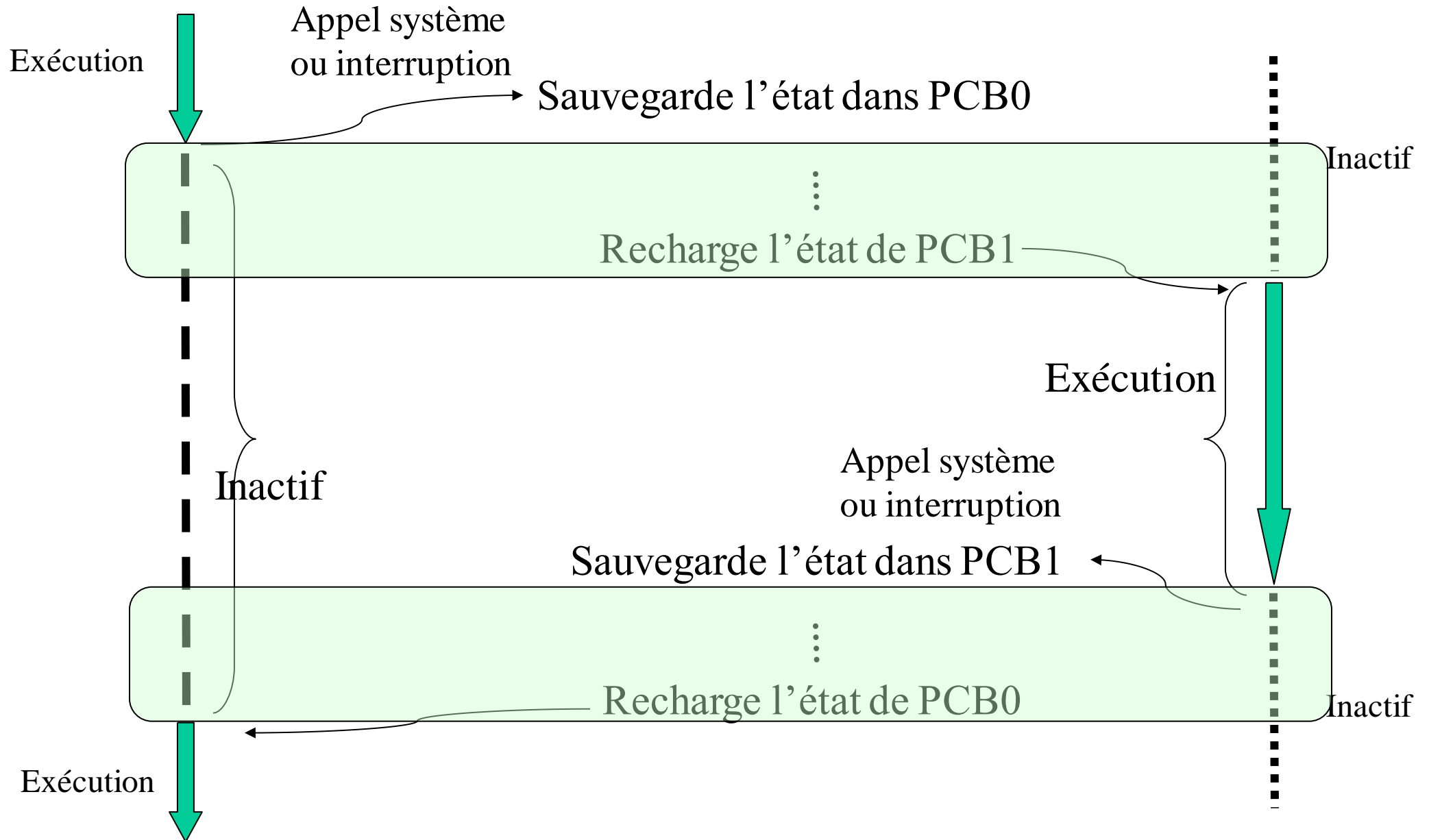
Réquisition du processeur

- suite à l'ordonnancement
- suite au réveil d'une tâche sur réception de message ou de signal
- suite au réveil d'une tâche en attente de délai
- suite au réveil d'une tâche endormie

# Les états d'une tâche



# La commutation de contexte



### Latence du système

C'est le délai global entre l'occurrence d'un évènement et la réaction attendue en sortie du système (exemple: lancement d'une tâche).

C'est un délai composite qui prend en compte:

- les délais de scrutation du système
- les délais liés au système d'exploitation
- les délais de traitement
- les délais de transmission de message

## Délai de Latence

$$Latence = t_{cpu} + t_{wait}$$

$$t_{cpu} = \alpha t_{exec} + \alpha t_{spinlocks}$$

$$t_{wait} = \alpha t_{I/O} + \alpha t_{sem} + \alpha t_{int} + \alpha t_{preempt}$$

$T_{spinlocks}$  : scrutation ,  $t_{sem}$  : exclusion mutuelle

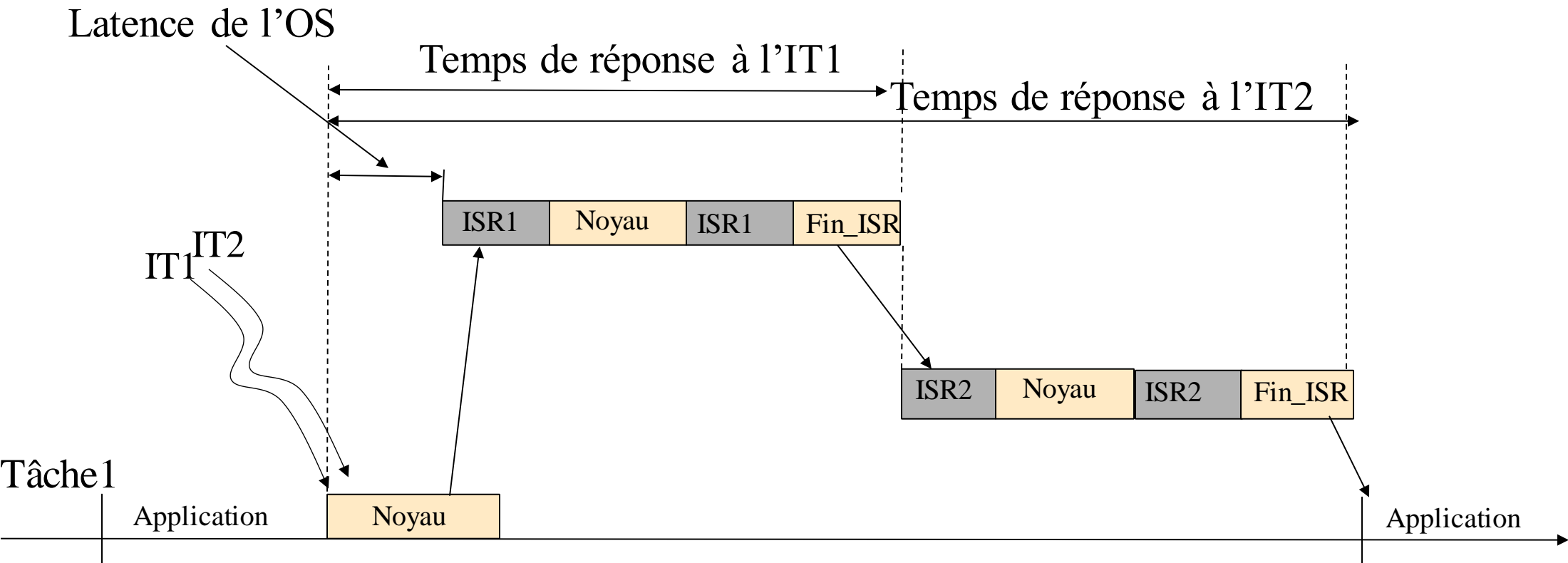
$t_{I/O}$  : traitement des demandes d'E/S

$t_{int}$  : traitement des interruptions

$t_{preempt}$  : commutation de contexte

# Les Interruptions

cas de deux interruptions quasi simultanées



# Les Ordonnancements Temps Réel

L'ordonnancement est le mécanisme par lequel le noyau choisit la tâche à activer. avec, dans le cas temps réel, 2 objectifs cruciaux:

- en fonctionnement nominal, assurer le respect des contraintes temporelles de toutes les tâches
- en fonctionnement anormal, assurer l'exécution des tâches les plus critiques nécessaires à la sécurité du système

Avec, en plus des objectifs de performance tels que

- minimiser le temps de réponse
- réduire la gigue de certaines tâches
- équilibrer la charge des sites (cas réparti )

Propriété des algorithmes

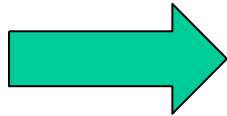
- **séquence valide** : *ordonnancement respectant les contraintes de toutes les tâches*
- **ordonnançabilité** : *il existe au moins un algorithme capable de fournir une séquence valide*
- **optimal** : *capable de fournir une séquence valide pour toute configuration de tâches ordonnançable*



# Les Ordonnancements Temps Réel

Catégories d'ordonnancement:

Hors ligne



- Liste préétablie  $\Rightarrow$  ordre d'exécution prédéfini
- Mise en œuvre simple, temps de réponse minimum
- Pas de flexibilité

En ligne



- Choix de la tâche à activer fait en cours d'exécution
- Prise en compte des événements et de l'état du système pour faire ce choix
- Algorithme d'ordonnancement dépend du temps
- Temps de réponse plus important, algorithme plus complexe
- Flexibilité

# Les Ordonnancements Temps Réel

Catégories d'ordonnancement:

Non Préemptif



L'exécution d'une tâche ne peut pas être interrompue par L'Ordonnanceur

Préemptif



L'Ordonnanceur peut réquisitionner le processeur en faveur d'une autre tâche que celle en cours d'exécution

Statique



Les priorités des tâches sont fixées à priori et ne changent pas en cours d'exécution

Dynamique



Les priorités des tâches dépendent de paramètres qui évoluent dans le temps.

# Ordonnancement de tâches partageant de ressources

## Le problème de l'inversion de priorité

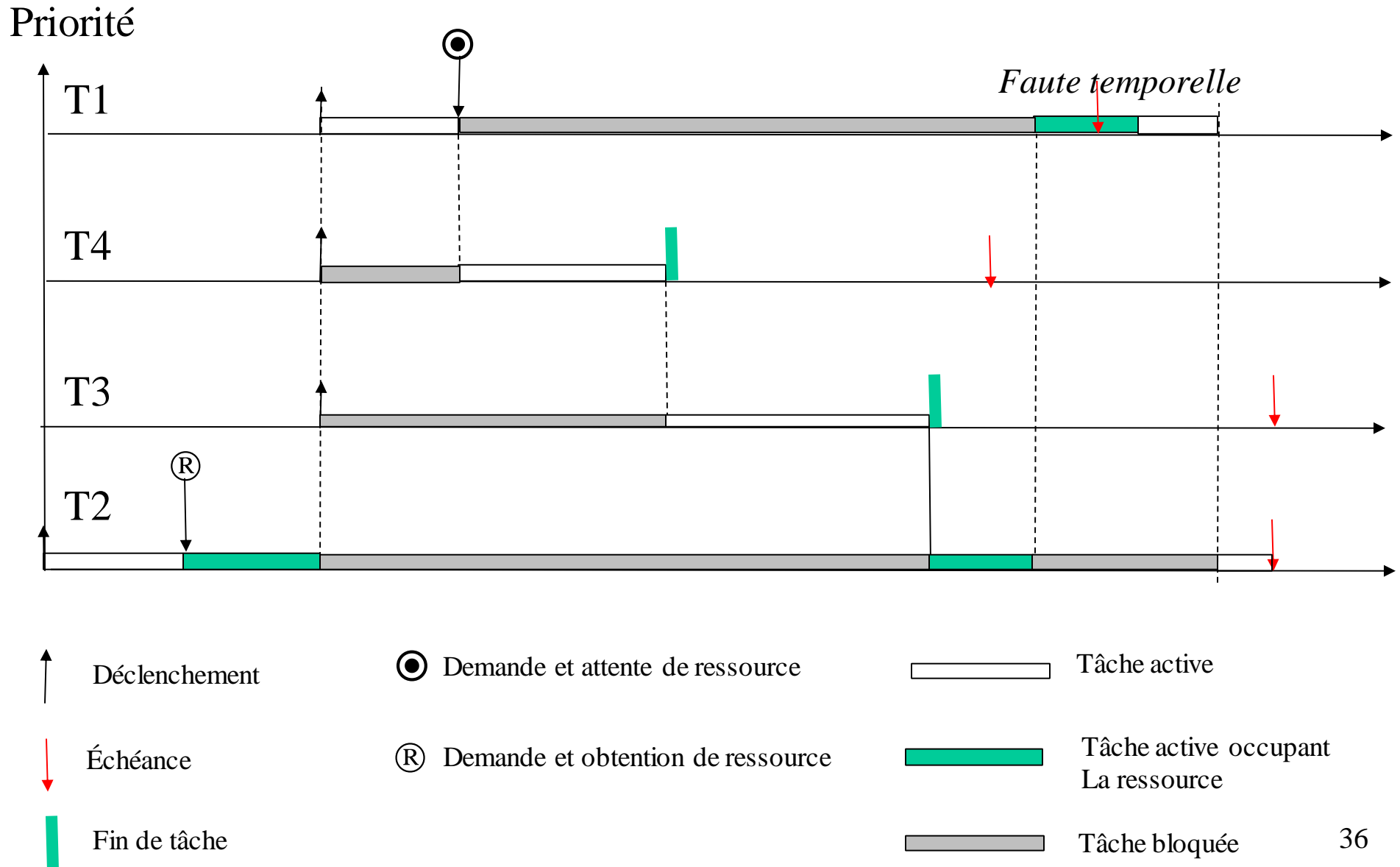
1. Une tâche T2 de faible priorité (20) demande et prend un sémaphore S
2. Une tâche T1 de priorité plus élevée (2) demande le sémaphore S
3. La tâche T1 est mise en attente sur S
4. T2 reprend son exécution
5. Une tâche T3 de priorité ( 10) plus élevée que T2 se réveille
6. T3 s'exécute alors que T1 reste toujours en attente

Toute tâche de priorité supérieure à celle de T2 pourra s'exécuter mais pas T1

C'est l'inversion de priorité

# Ordonnancement de tâches partageant de ressources

Le problème de l'inversion de priorité:  $P_{T1} > P_{T4} > P_{T3} > P_{T2}$



# inversion de priorité : un exemple

L'incident en 1997 de Pathfinder, le robot envoyé sur Mars:  
resets intempestifs répétés

3 tâches dans ce scénario:

1. gestion du bus de communication (haute priorité)
2. Communication (priorité moyenne),
3. collecte de données météo (faible priorité,)

Scénario d'exécution:

Pathfinder commence à récolter les données météo,  
le robot s'est mis à exécuter en boucle des reset du système,  
déclenché par un Watchdog qui a détecté des dépassements  
d'échéance sur la tâche 1.

les tâches 1 et 3 partagent la ressource Bus

la tâche 3 prend le bus, la tâche 1 le demande et reste  
bloquée, la tâche 2 prend la main pour une durée longue, il y  
a dépassement pour la tâche 1



# Comment le diagnostic a été effectué?

- La NASA a reproduit sur terre l'exécution sur le même matériel (IBM RS6000) et le même OS temps réel (VxWorks).
- des heures et des heures d'ingénieurs ont été nécessaires pour reproduire la trace d'exécution.
- **l'inversion de priorité** est finalement identifiée.
- VxWorks implémente **l'héritage de priorité** mais la valeur par défaut est « False ».
- Par chance il a été possible de télécharger la correction sur le robot déployée.
- les resets intempestifs ont cessé.



# Ordonnancement de tâches partageant de ressources

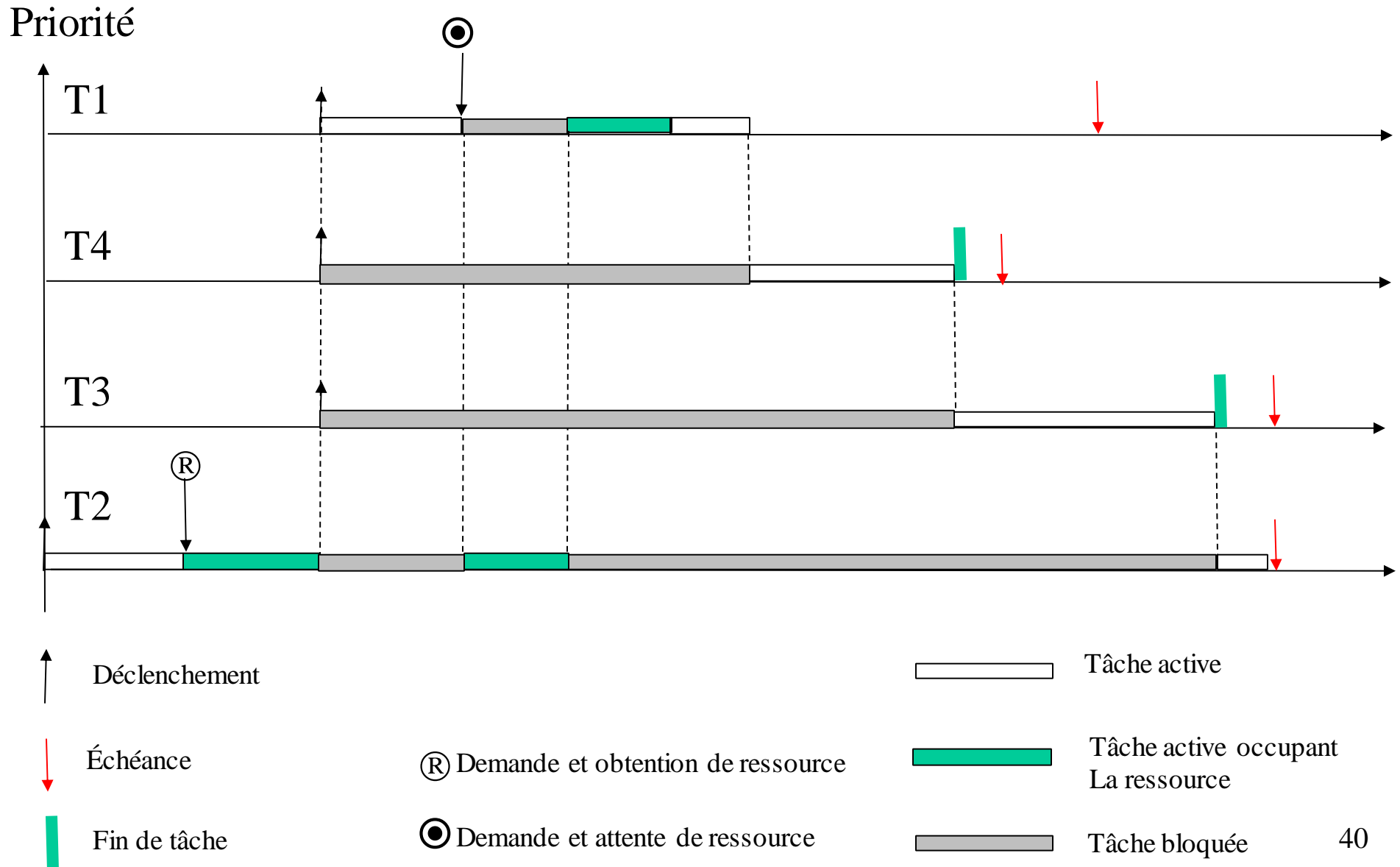
## L'héritage de priorité

Pour éviter le problème de l'inversion de priorité, introduction du mécanisme d'héritage: mutex à héritage

1. Une tâche T2 de faible priorité (20) accède et verrouille le mutex à héritage Mx
2. Une tâche T1 de priorité plus élevée (2) demande l'accès au mutex à héritage Mx
3. La tâche T1 est mise en attente sur Mx
4. T2 reprend son exécution avec la priorité (2) de T1 jusqu'au déverrouillage de Mx
5. Une tâche T3 de priorité (10) plus élevée que celle de T2 se réveille
6. T3 ne peut pas s'exécuter car T2 s'exécute avec la priorité de T1
7. T2 déverrouille l'accès à Mx
8. T2 est préemptée au profit de T1 qui reprend l'exécution et verrouille l'accès à Mx

# Ordonnancement de tâches partageant de ressources

L'héritage de priorité:  $P_{T1} > P_{T4} > P_{T3} > P_{T2}$ , T1 et T2 partagent la ressource





# Ordonnancement de tâches partageant des ressources

## Priority Ceiling Protocol : PCP (priorité plafond)

L'héritage de priorité:

- donne une borne max qui peut être inacceptable dans certains cas
- chaîne de blocage possible

D'où le PCP:

Ressource partagée  priorité plafond = Max (Prio) des tâches qui la partagent

Une tâche  $T_i$  ne peut accéder à une ressource que si elle est libre et si

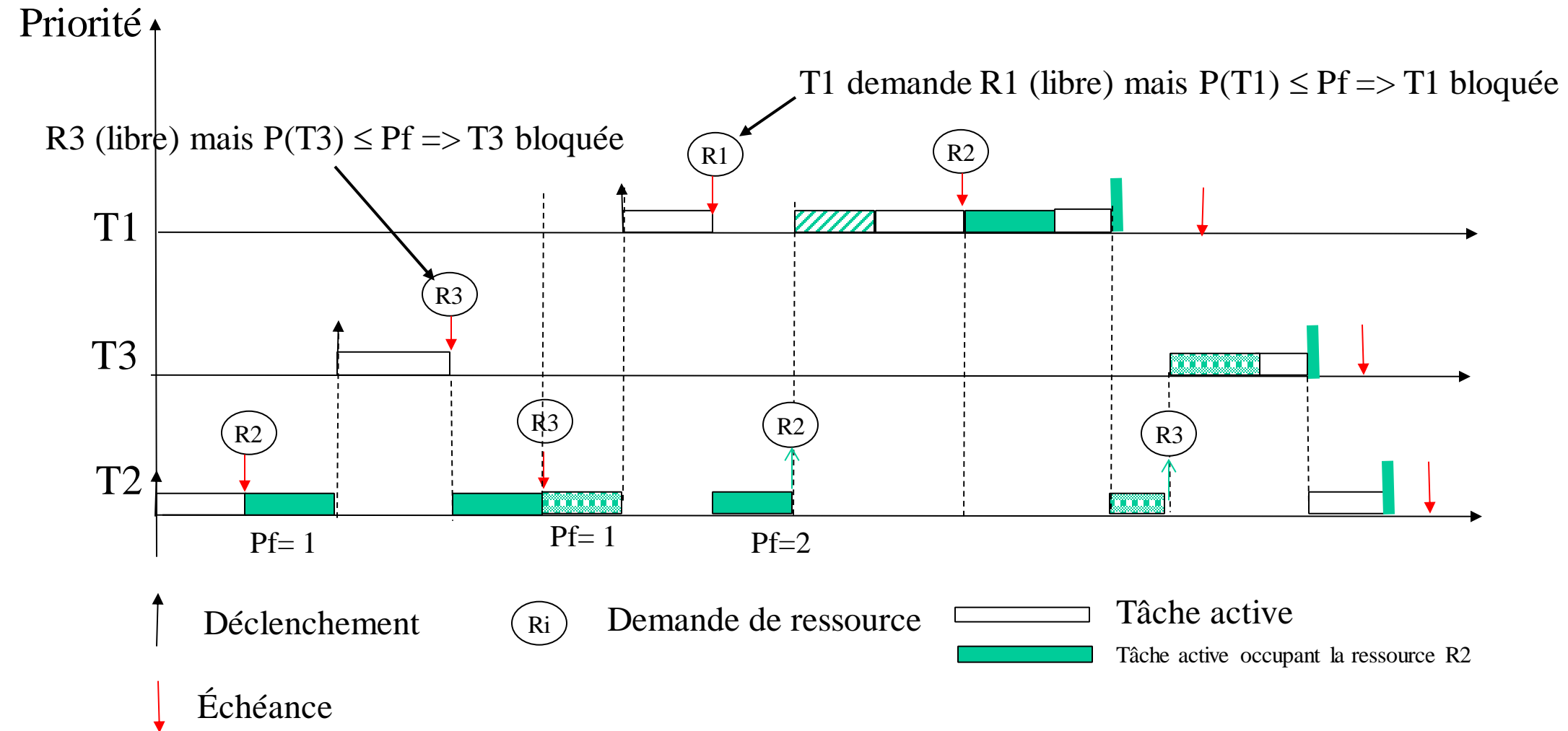
$Prio(T_i) > Max(plafond \text{ des ressources occupées par les autres tâches})$

Si  $T_i$  bloque plusieurs tâches  $T_j$  plus prioritaires alors  $Prio(T_i) = Max(Prio(T_j))$

# Ordonnancement de tâches partageant des ressources

Priority Ceiling Protocol : PCP (priorité plafond):

$$P_{T1} = 1 > P_{T3} = 2 > P_{T2} = 3,$$
$$Pf_{R1} = 1, Pf_{R2} = 1, Pf_{R3} = 2$$



# Ordonnancement de tâches partageant des ressources

## Immediate Priority Ceiling Protocol : IPCP

PCP + Priorité dynamique de tâches

$$P_{\text{dynamique}}(T_i) = \text{Max} ( P(T_i) , P_f(\text{ressources occupées par } T_i))$$

Exemple:

2 tâches T1 et T2 accèdent à 2 ressources R1 et R2

$$P(T_1) = 1, P(T_2) = 3$$

$$P_{f_{R1}} = 1, P_{f_{R2}} = 1$$

à t=0, T<sub>2</sub> est activée , demande R1 et l'obtient , P(T<sub>2</sub>) = ?

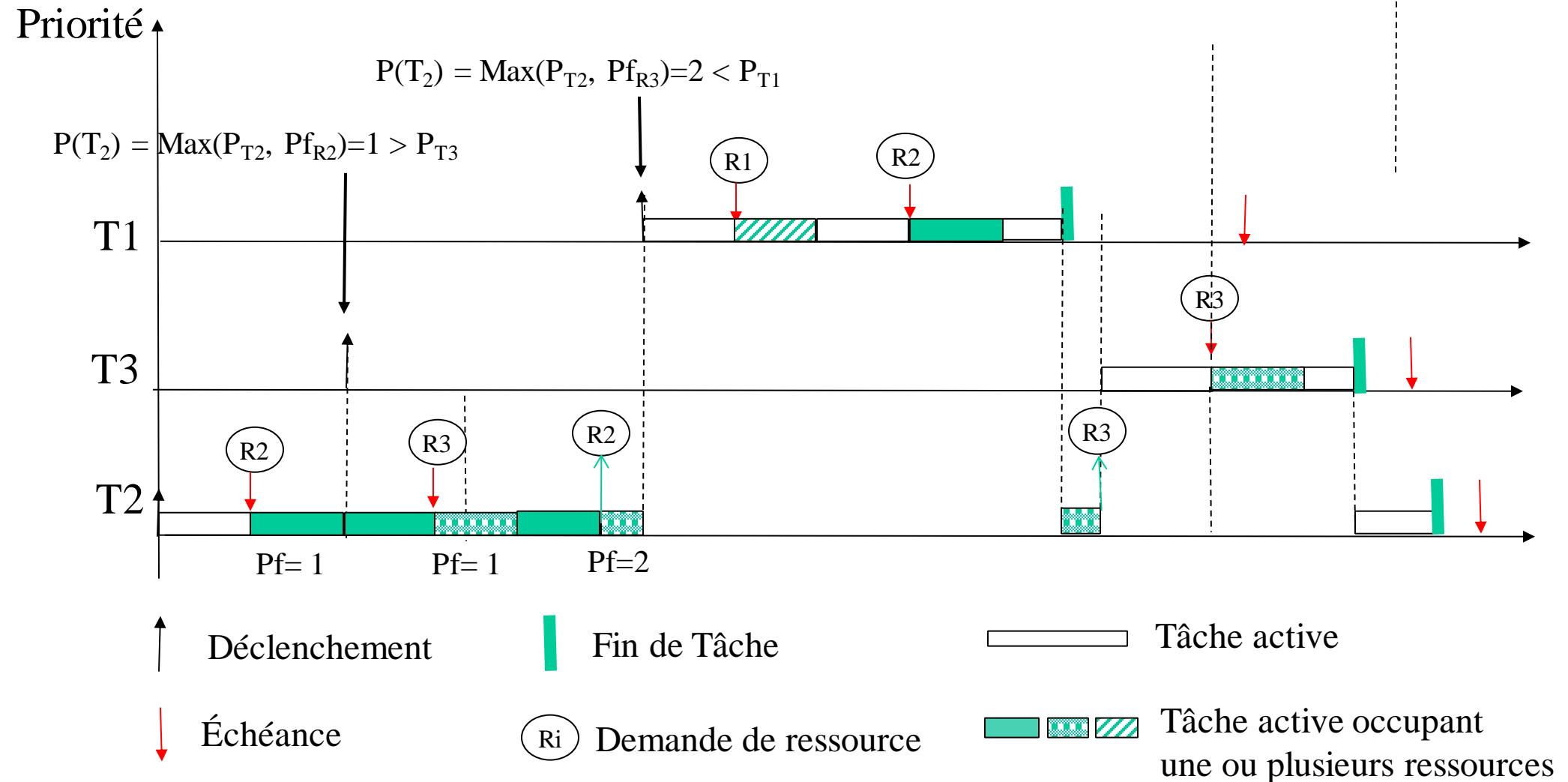
à t=1 T1 est activée, et demande à s'exécuter , que se passe-t-il?

# Ordonnancement de tâches partageant des ressources

Immediate Priority Ceiling Protocol : IPCP (priorité plafond +  $P_{\text{dynamique}}$ ):

$$P_{T1} = 1 > P_{T3} = 2 > P_{T2} = 3,$$

$$Pf_{R1} = 1, Pf_{R2} = 1, Pf_{R3} = 2$$



# Les Ordonnancements Temps Réel

## Caractéristiques temporelles d'une tâche $\tau_i$

$r_i$  : date d'activation

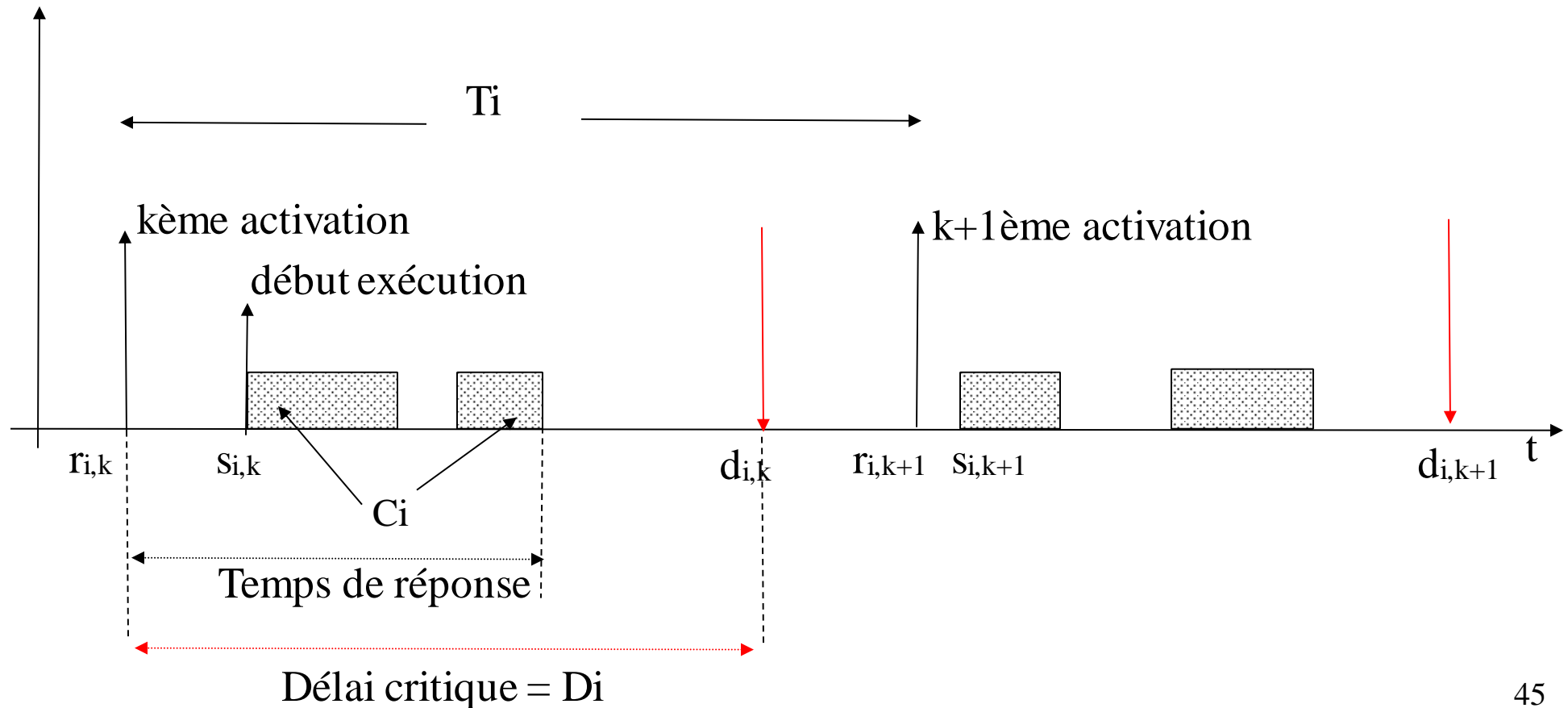
$s_i$  : date début d'exécution  $C_i$  : durée d'exécution

$D_i$  (délai critique)

$d_i$  : échéance relative à la tâche =  $r_i + D_i$

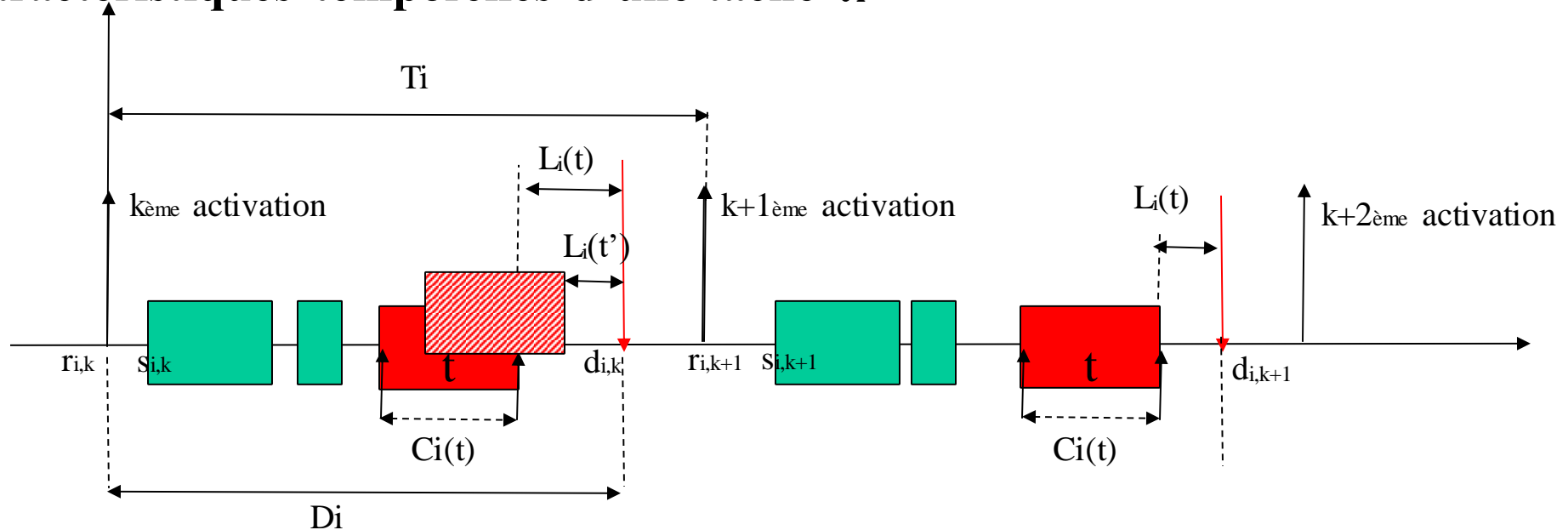
$T_i$  : période de la tâche

$TR_i$  : temps de réponse de la tâche  $i$



# Les Ordonnancements Temps Réel

## Caractéristiques temporelles d'une tâche $\tau_i$



$C_i(t)$  = durée d'exécution restante à l'instant  $t$

$D_i(t)$  = délai critique résiduel à l'instant  $t$

La laxité de  $T_i$  correspond à la marge de manœuvre pour  $T_i$  à l'instant  $t$

$$L_i(t) = \text{Laxité}_i(t) = D_i(t) - C_i(t) = D_i + r_i - t - C_i(t)$$

La gigue temporelle d'une tâche est donnée par

$$\text{Gigue}_i = \max \{ |TR_{i,k} - TR_{i,k+1}| / D_i \} \text{ (différence des temps de réponse)}$$

$$\text{Ou bien } \text{Gigue}_i = \max \{ |s_{i,k} - s_{i,k+1}| / T_i \}$$

# Les Ordonnancements Temps Réel

**Configuration de tâches** : ensemble de  $n$  tâches partageant des ressources pour leur exécution

Début simultané ou échelonné

Facteur d'utilisation du processeur pour une tâche donnée :

$$U_i = C_i / T_i$$

Facteur d'utilisation du processeur pour une configuration de  $n$  tâches périodiques

$$U = \sum_{i=1}^n \frac{C_i}{T_i}$$

Facteur de charge du processeur pour une configuration de  $n$  tâches périodiques avec  $T_i \neq D_i$

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

## Tâches non périodiques

*Tâches sporadiques* : on ne leur connaît pas de période mais on connaît l'intervalle minimum entre deux requêtes successives

Caractéristiques :  $C_{\text{spor}}$  = temps processeur nécessaire à son exécution

$D_{\text{spor}}$  = le délai critique

$P_{\text{spor}}$  = intervalle minimum entre deux requêtes  
d'exécution

*Tâches apériodiques* : pas de période connue et pas de délai minimum entre deux requêtes

Afin de faciliter leur étude on ramène les tâches non périodiques à des tâches périodiques pour l'applicabilité des algorithmes temps réel.



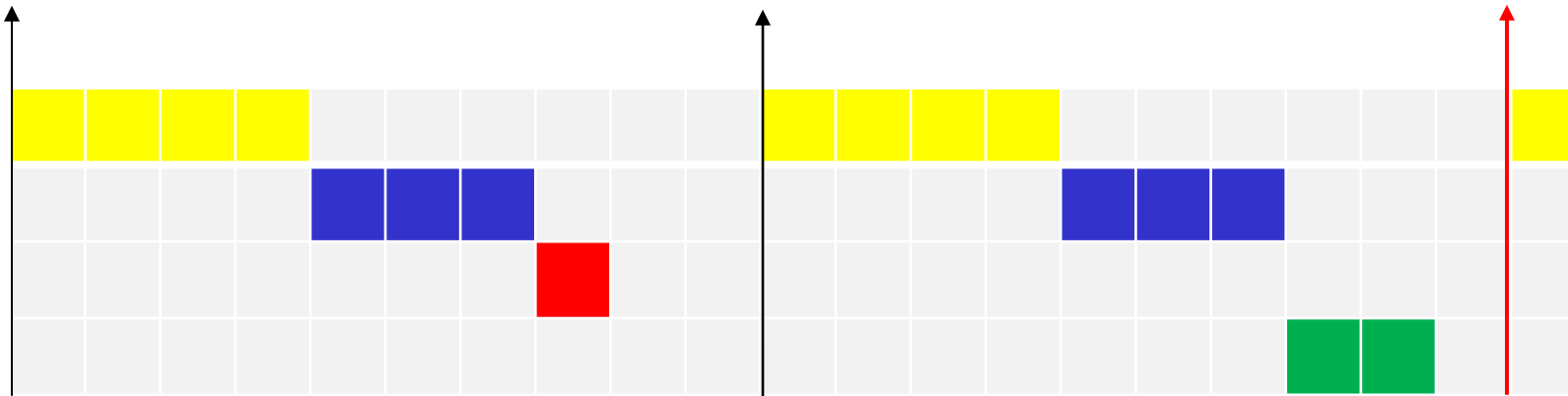
# Les Ordonnancements Temps Réel

Un modèle de base:

- *Nombre de tâches fixe*
- *Tâches périodiques et indépendantes (pas de partage de ressources, synchronisation, communication)*
- *Temps de commutation négligeable*

Ordonnancement cyclique : A,B,C,D

Tâches	Période $T_i$	Durée $C_i$
A	10	4
B	10	3
C	20	1
D	20	2

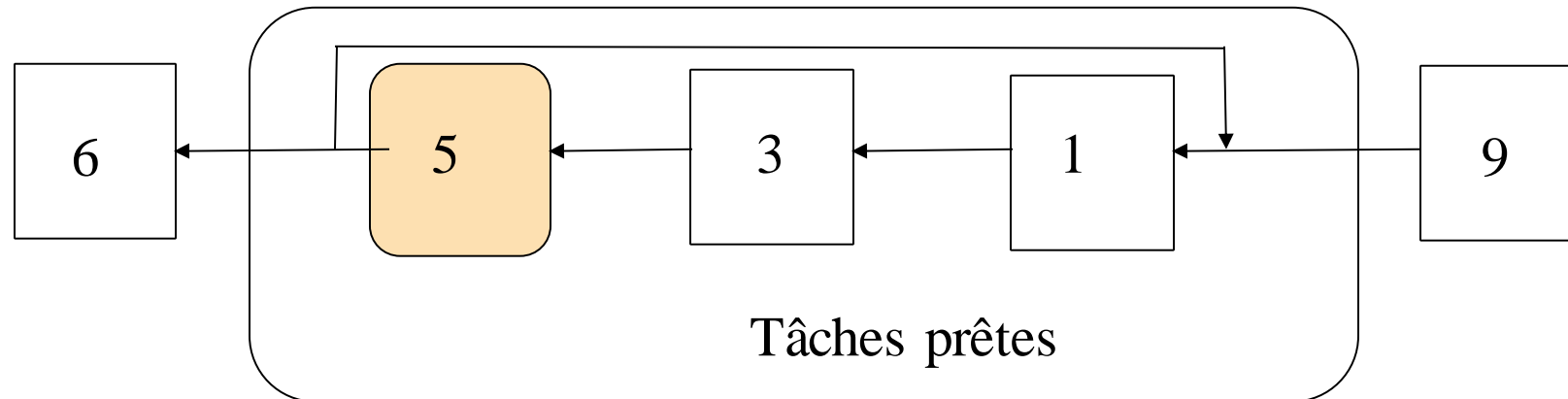


```
while(true) {  
  A();  
  B();  
  C();  
  attendre interruption;  
  A();  
  B();  
  D();  
  Attendre interruption;  
}
```

# Les Ordonnancements Temps Réel

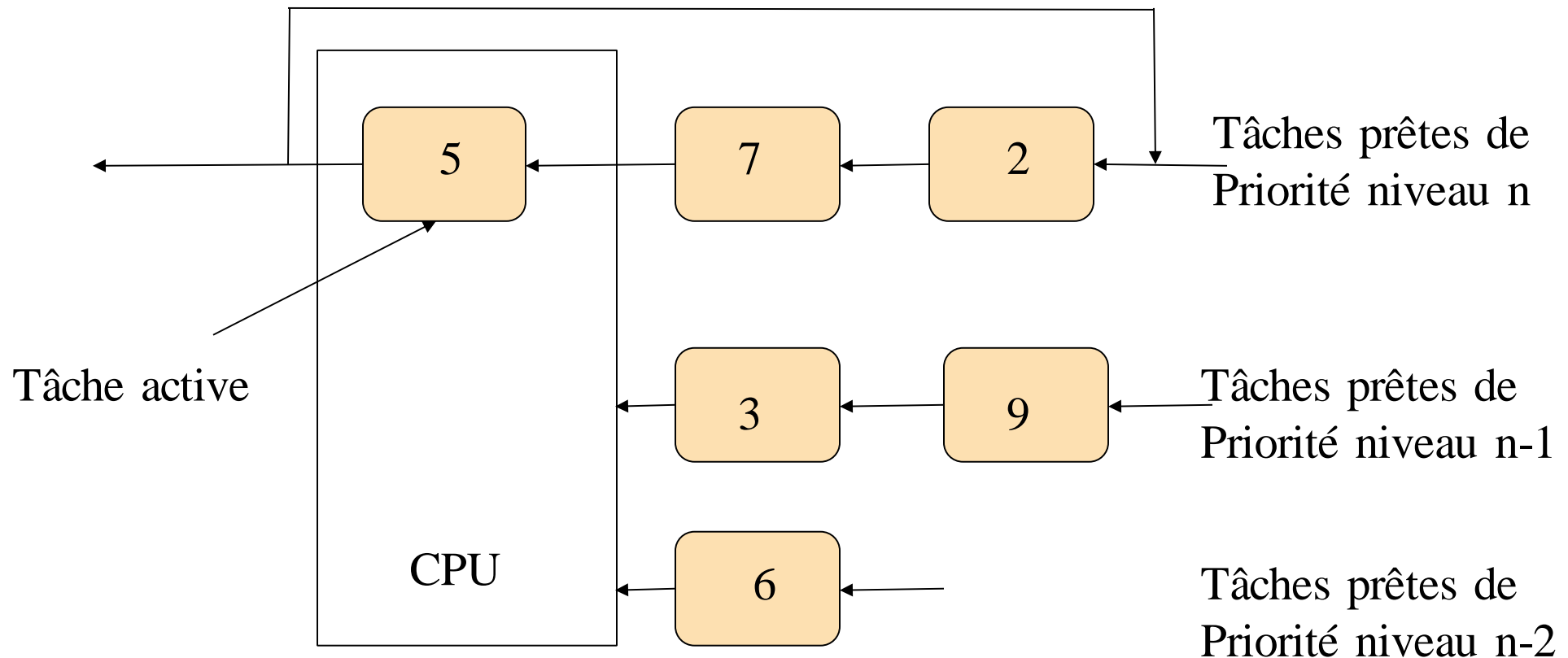
Quelques algorithmes classiques:

- premier arrivé premier servi (FIFO)
- plus courte durée en premier (SJF)
- circulaire (round robin ou tourniquet) : chaque tâche prête dispose à tour de rôle d'un quantum de temps pour s'exécuter



# Les Ordonnancements Temps Réel

Ordonnancement par priorité à files multiples, ou aussi FIFO à files multiples

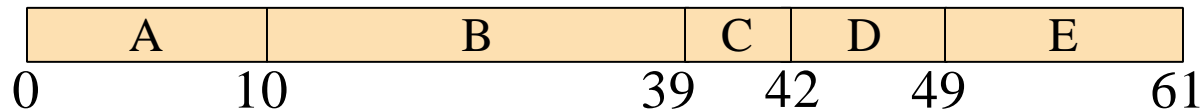


# Les Ordonnancements Temps Réel

Exemple d'évaluation d'algorithmes / temps moyen d'attente

Tâches :                                      A                      B                      C                      D                      E,      activation  $t=0$   
 temps d'exécution (us):            10                      29                      3                      7                      12

## FIFO (A,B,C,D,E)



Temps moyen d'attente :

Temps de réponse moyen :

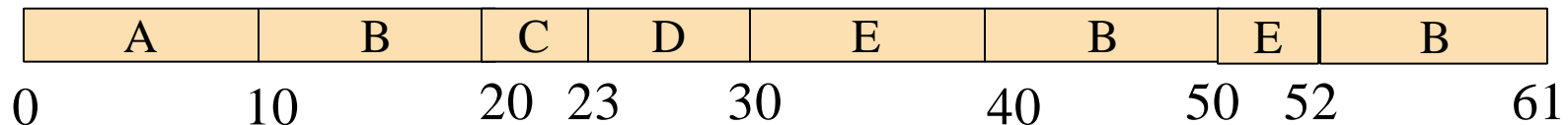
## SJF (Short Job First)



Temps moyen d'attente :

Temps de réponse moyen:

## Tourniquet avec $\tau = 10$ us



Temps moyen d'attente :

Temps de réponse moyen:

## Ordonnancement selon la période : **Rate Monotonic Algorithm** (RMA ou RMS)

- Algorithme en ligne à priorité constante
- Optimal pour une configuration de tâches périodiques à échéances sur requêtes
- Tâches indépendantes
- Priorité inversement proportionnelle à la période

Condition *suffisante* d'ordonnançabilité :

$$U = \sum_{i=1}^n \frac{C_i}{T_i} \leq n(2^{1/n} - 1) \xrightarrow{n \rightarrow \infty} 0.69$$

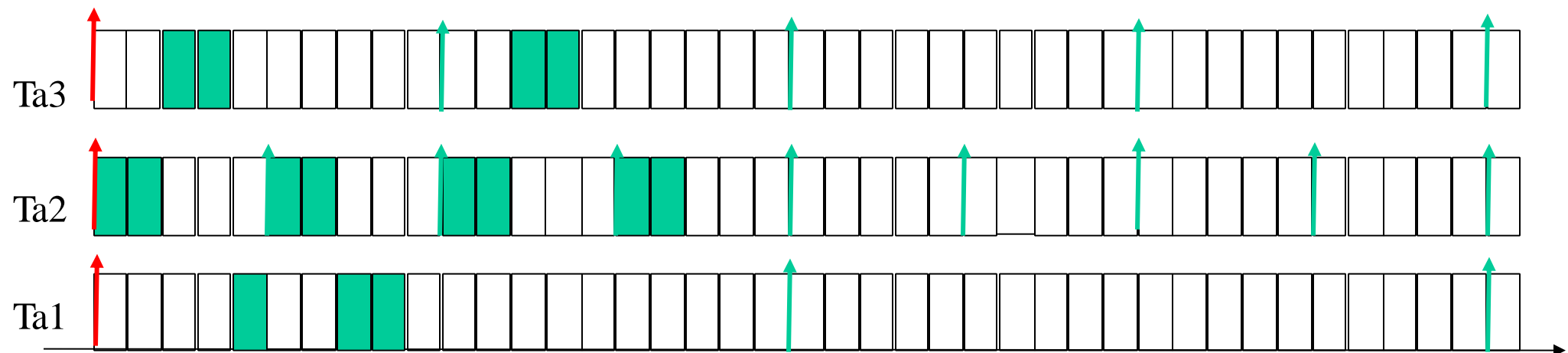
# Les Ordonnements Temps Réel

## Rate Monotonic Algorithm

Exemple 1:

Tâches	$r_0$	$C_i$	$T_i$	$Pr_i$
Ta3	0	2	10	2
Ta2	0	2	5	1
Ta1	0	3	20	3

$$U = \frac{3}{20} + \frac{2}{5} + \frac{2}{10} = 0.75 < 0.779$$

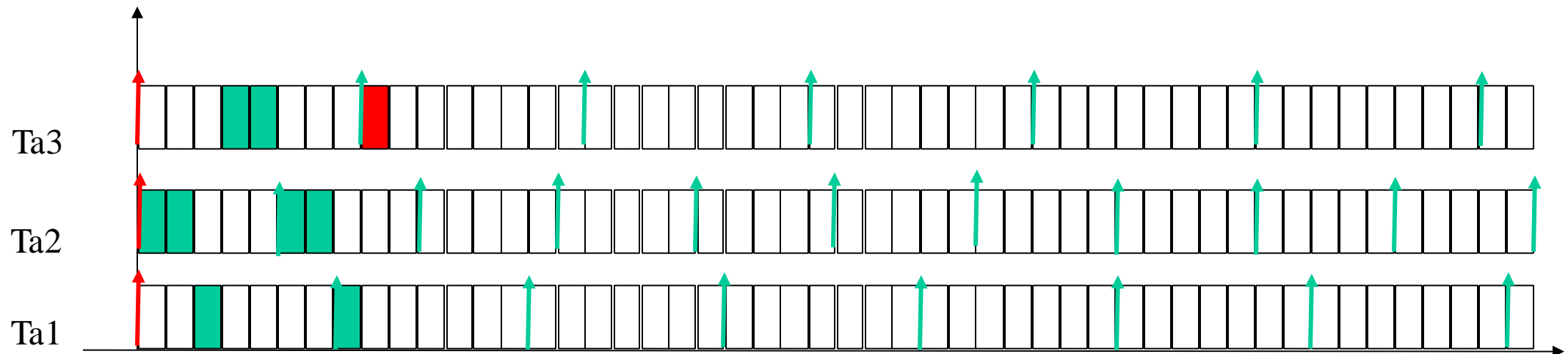


# Les Ordonnancements Temps Réel

Exemple 2:

Tâches	$r_i$	$C_i$	$d_i$	$T_i$	$U_i$	$Pr_i$
Ta3	0	3	8	8	0.375	3
Ta2	0	2	5	5	0.4	1
Ta1	0	1	7	7	0.143	2

$$U=0.918 \geq 3(2^{1/3}-1) = 0.779$$



Dans le cas d'incertitude où la condition suffisante d'ordonnançabilité n'est pas vérifiée mais la condition nécessaire l'est alors on peut appliquer un test défini comme suit:

*Si un ensemble de  $n$  tâches indépendantes avec priorités fixes affectées par RMA respecte la 1<sup>ère</sup> échéance de chaque tâche quand toutes les tâches sont démarrées en même temps (à  $T_0$ ), alors l'ensemble est toujours ordonnançable.*



# Les Ordonnancements Temps Réel

**Ordonnancement selon le délai critique :**

**Inverse Deadline ou Deadline Monotonic)**

La tâche la plus prioritaire est celle de plus petit délai critique

- Performances
- équivalente à RMA pour tâches à échéances sur requête
  - meilleure dans le cas de configurations arbitraires
  - optimal pour une configuration de tâches à échéances < requêtes

Condition suffisante d'ordonnançabilité  $U = \sum_{i=1}^n \frac{C_i}{D_i} \leq n(2^{1/n} - 1)$

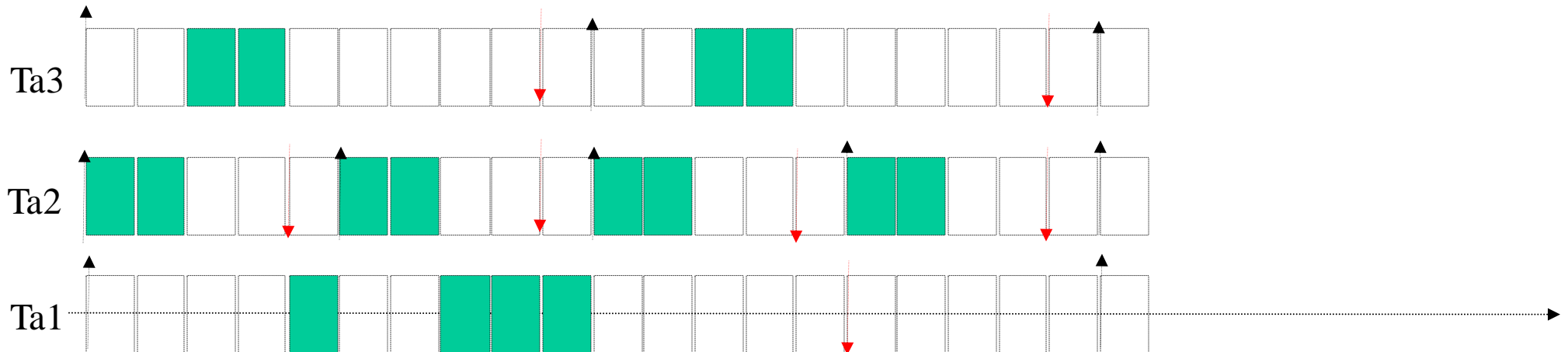
# Les Ordonnancements Temps Réel

**Exemple : Inverse Deadline ou Deadline Monotonic . Priorité fixe**

Tâches	$r_0$	$C_i$	$T_i$	$D_i$	$Pr_i$
Ta3	0	2	10	9	3 (2)
Ta2	0	2	5	4	1
Ta1	0	4	20	8 (15)	2 (3)

Condition suffisante d'ordonnançabilité  $U = \sum_{i=1}^n \frac{C_i}{D_i} = 1.22 > n(2^{1/n}-1)$   $U = 0.92$

Condition nécessaire d'ordonnançabilité  $U = \sum_{i=1}^n \frac{C_i}{T_i} \leq 1 = 0,8$



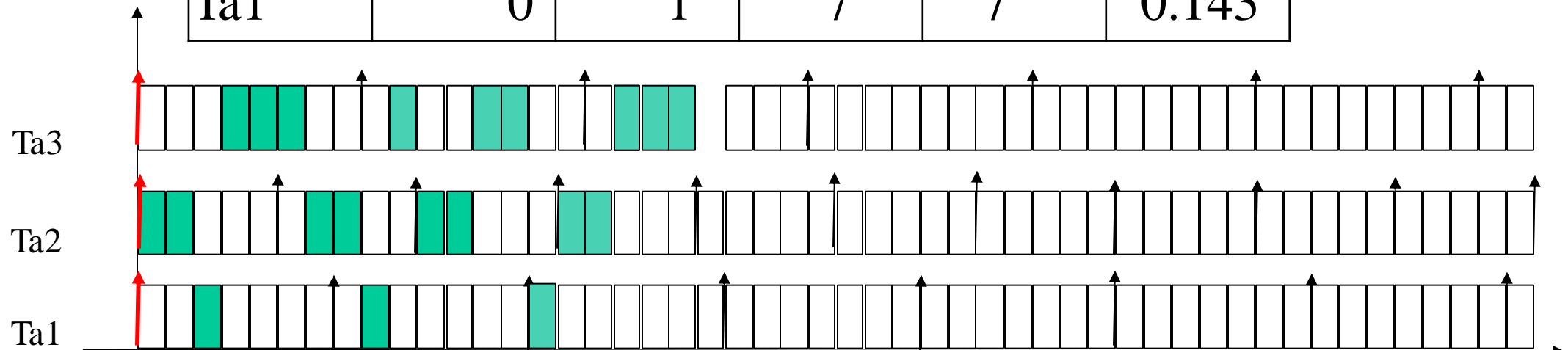
# Les Ordonnancements Temps Réel

## Ordonnancement selon l'échéance dynamique : Earliest Deadline First (EDF)

Condition *suffisante* d'ordonnançabilité : 
$$U = \bigvee_{i=1}^n \frac{C_i}{D_i} \leq 1$$

Condition *nécessaire et suffisante* (si  $T_i = D_i$ ) : 
$$U = \bigvee_{i=1}^n \frac{C_i}{T_i} \leq 1$$

Tâches	$r_i$	$C_i$	$d_i$	$T_i$	$U_i$
Ta3	0	3	8	8	0.375
Ta2	0	2	5	5	0.4
Ta1	0	1	7	7	0.143



**Calcul du temps de réponse maximum  
WCRT (Worst Case Response Time)  
Tâche à Priorité fixe et indépendantes**

$$Tr_i = C_i + \sum_{j \in hp(i)} Attente\_Tâche_j$$

$$Tr_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{Tr_i^n}{T_j} \right\rceil \times C_j$$

Avec  $hp(i)$  = ensemble des tâches de priorité supérieure à celle de la tâche  $i$

$$WCRT_i = Tr_i^k, \quad k = \text{itération de convergence}$$

## Calcul du temps de réponse maximum WCRT (Worst Case Response Time)

Tâches	$r_0$	$C_i$	$T_i$	$D_i$	$Pr_i$
Ta3	0	2	10	9	2
Ta2	0	2	5	4	1
Ta1	0	4	20	10	3

$$WCRT_2 = C_2 = 2$$

$$Tr_3^0 = C_3 = 2; \dots \rightarrow WCRT_3 = 4;$$

$$Tr_1^0 = C_1 = 4; \dots \rightarrow WCRT_1 = 10$$

## Calcul du temps de réponse maximum WCRT (Worst Case Response Time)

tâches partageant des ressources

Durée de blocage avec protocole héritage de priorité

$$B_i = \sum_{k=1}^K \text{usage}(k, i) * C(k)$$

Durée de blocage avec OPCP ou IPCP

$$B_i = \sum_{k=1}^K \text{Max usage}(k, i) * C(k)$$

# Calcul du temps de réponse maximum WCRT (Worst Case Response Time)

tâches partageant des ressources

Calcul de  $B_i$

$K$ : nombre de ressources

$$usage(k,i) = \begin{cases} 1 & \text{si } \exists T_j \text{ tq } T_j \text{ utilise } k \text{ et } Prio(T_j) < Prio(T_i) \text{ ET} \\ & \exists T_n \text{ tq } T_n \text{ utilise } k \text{ et } Prio(T_n) \geq Prio(T_i), \text{ incluant } T_i \\ 0 & \text{sinon} \end{cases}$$

$C(k)$  : durée d'utilisation maximum de la ressource  $k$  par une tâche  $T_j$  de priorité inférieure à celle de  $T_i$  ( $Prio(T_j) < Prio(T_i)$ )

Temps de réponse maximum

$$Tr_i^{n+1} = B_i + C_i + \sum_{j \in hp(i)} C_j \times \left\lceil \frac{Tr_i^n}{T_j} \right\rceil$$

# 5 Tâches: t1, t2, t3, t4, t5 et 3 ressources R1, R2, R3

## Utilisation des ressources

	R1	R2	R3
t1 (1)		20	
t2 (2)	5 ( C(R1) )		10
t3 (3)		5	5
t4 (4)			5
t5 (5)	10 (C(R1))	3	

*usage(R1,1)=?*  
*usage(R2,1)=?*  
*usage(R3,1)=?*  
*B1=?*

$Pf(R1)=2$  ;  $Pf(R2)=1$ ;  $Pf(R3)=2$

Calcul de Bi

Avec l'héritage de priorité  $B1=5$ ,  $B2=20$ ,  $B3=18$ ;  $B4=13$ ;  $B5=?$

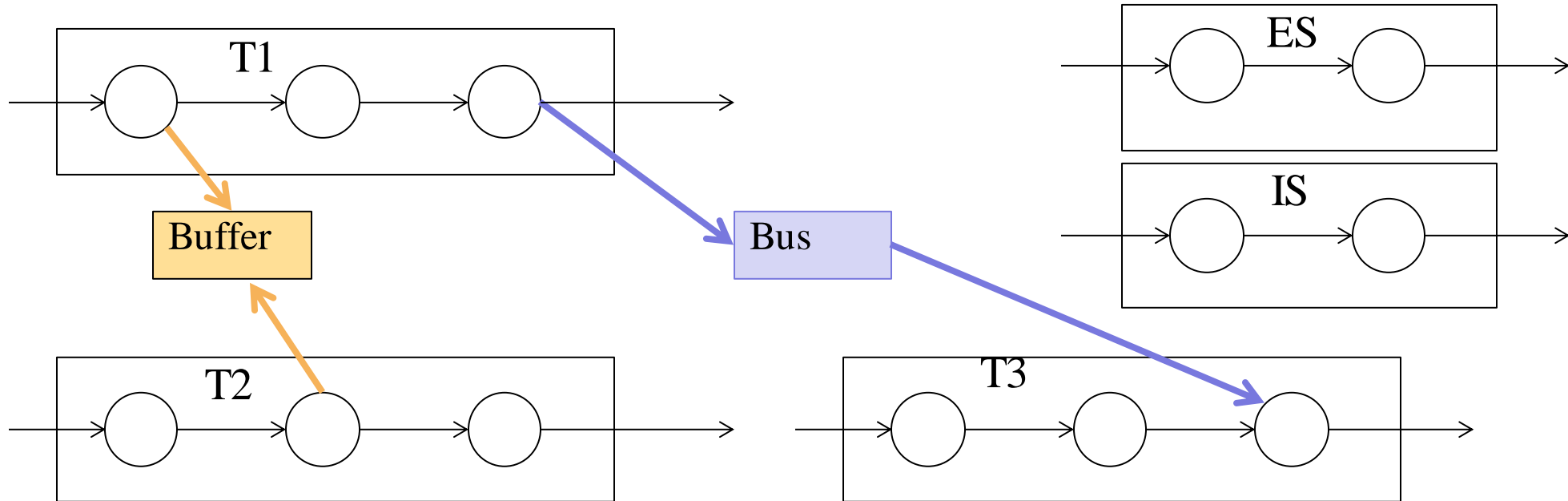
Avec IPCP  $B1=5$ ,  $B2=10$ ,  $B3=10$ ;  $B4=10$ ;  $B5=?$

$$usage(k,i) = \begin{cases} 1 & \text{si } \exists T_j \text{ tq } T_j \text{ utilise } k \text{ et } Prio(T_j) < Prio(T_i) \text{ ET} \\ & \exists T_n \text{ tq } T_n \text{ utilise } k \text{ et } Prio(T_n) \geq Prio(T_i), \text{ incluant } T_i \\ 0 & \text{sinon} \end{cases}$$



Exemple: 5 tâches, 2 ressources Buffer et Bus

T1 utilise le Buffer (2ms) puis le Bus (10ms) , T2 utilise le Buffer (20ms) , T3 utilise le Bus (10ms)



Tâches	Ci	Période	Priorité	Bi (Héritage)	Bi (Plafond)	D	BUS	BUFFER
ES	5	50	1			6	0	0
IS	10	100	2			100	0	0
T1	20	100	3			100	10	2
T2	40	150	4			130	0	20
T3	100	350	5			350	10	0

# Exercices sur les ordonnancements

Tâche	Période	Durée Ci	Priorité	$r_{i,0}$	Utilisation ressources
A	10	4	1	7	EVQE
B	20	3	2	5	EQE
C	20	3	3	5	EEE
D	40	6	4	2	EQQQQE
E	40	4	5	0	EVVE

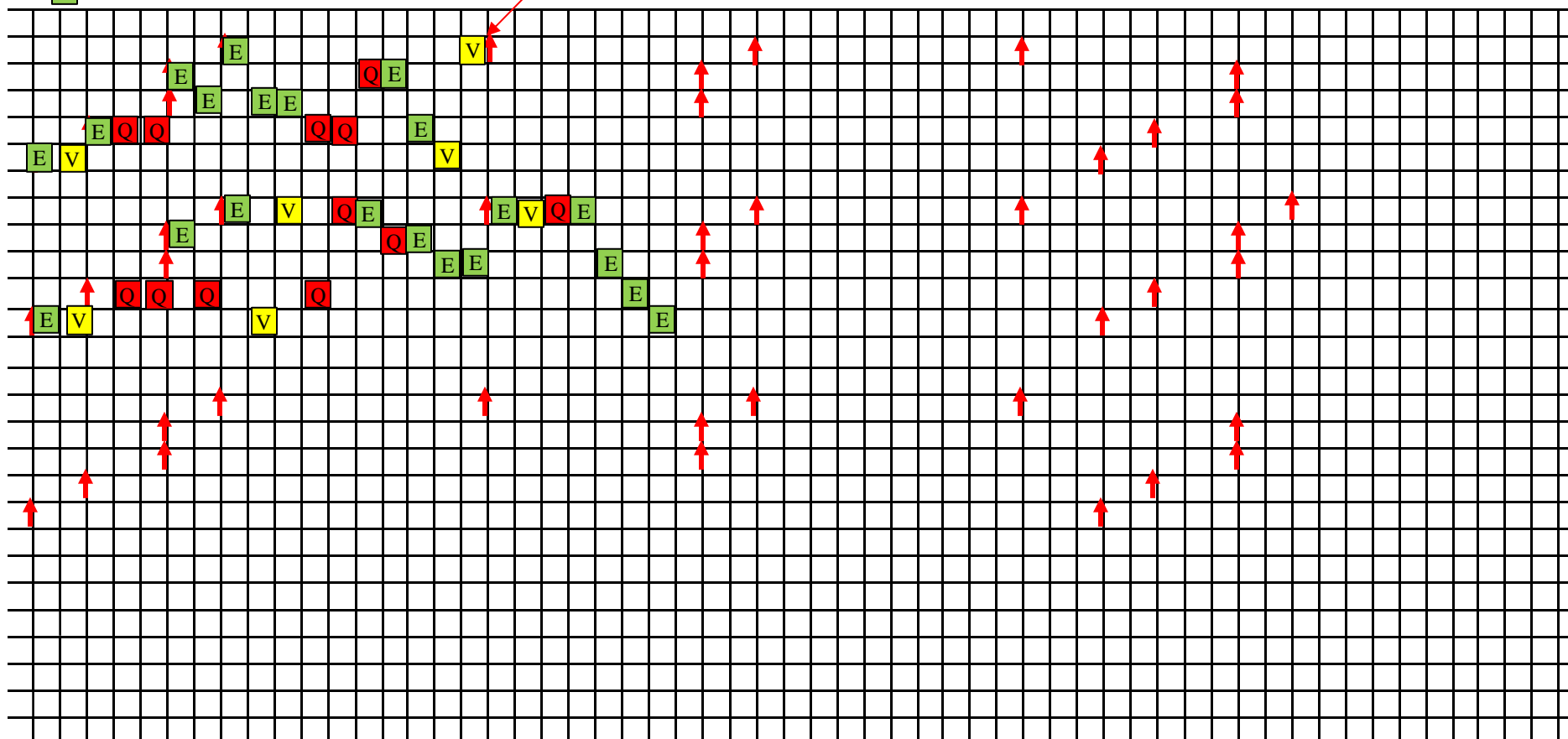
1. ordonnançabilité de cette configuration (sans les ressources) avec RMA?
2. Chronogramme RMA avec ressources?
3. Calcul des WCRT sans prendre en compte les blocages liés aux ressources?
4. Chronogramme avec héritage de priorité ?
5. Chronogramme avec PCP ?
6. Chronogramme avec IPCP ?
7. Calculer les temps de blocage pour chaque tâche avec PCP.
8. Calcul des WCRT avec les temps de blocage?

Tâche	Période	Durée Ci	Priorité	$r_{i,0}$	Utilisation ressources
A	10	4	1	7	EVQE
B	20	3	2	5	EQE
C	20	3	3	5	EEE
D	40	6	4	2	EQQQQE
E	40	4	5	0	EVVE

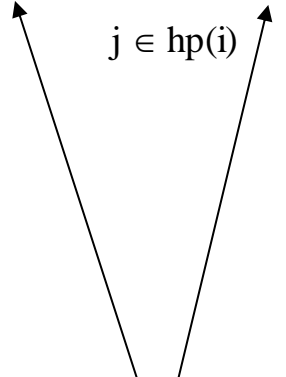
V

Q

E

E  
5

# Worst Case Execution Time: WCET

$$\text{WCRT}_i^{n+1} = B_i + C_i + \sum_{j \in \text{hp}(i)} C_j * \left\lceil \frac{\text{Tr}_i^n}{T_j} \right\rceil$$


**WCET**

## Comment évaluer WCET?

*A partir du code de la tâche*

### *Questions ?*

*Que fait le code de la tâche?*

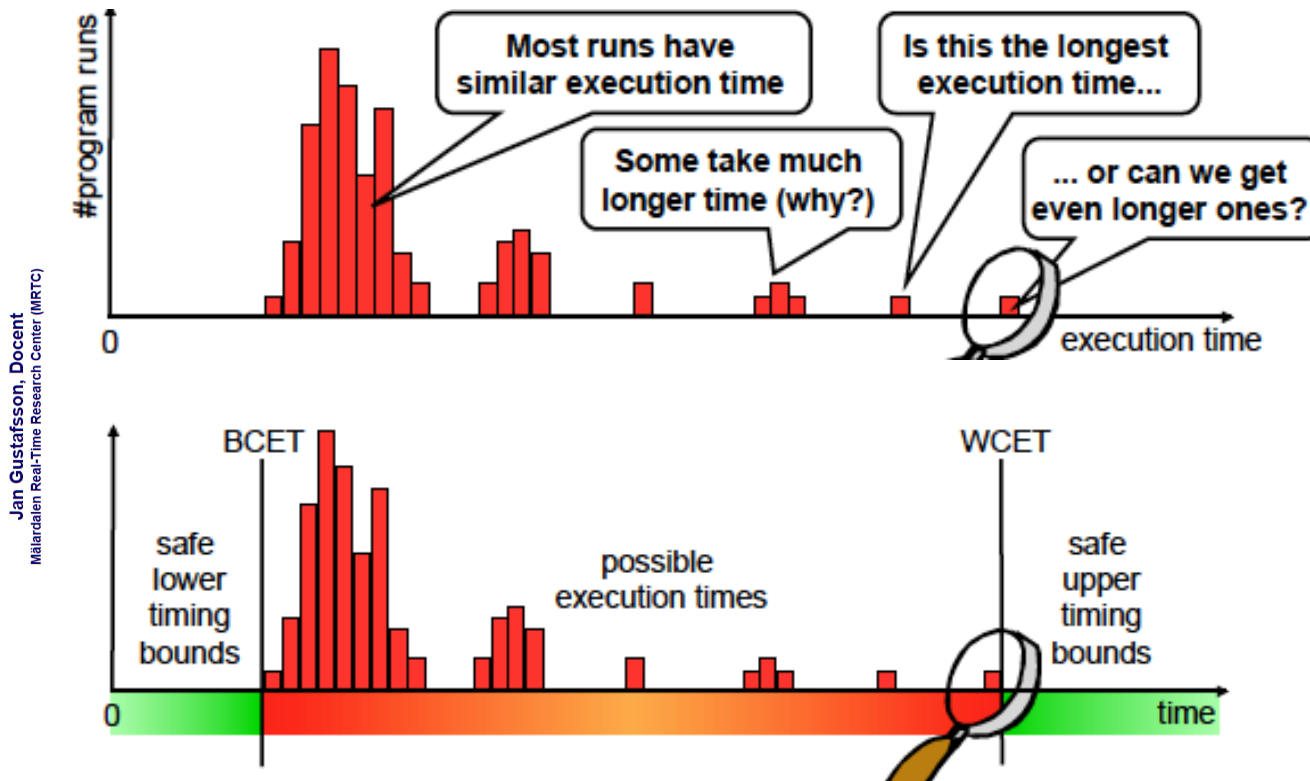
*Le code fait-il toujours la même chose?*

*Combien le résultat est-il important?*

*Quel est le temps d'exécution de ce code?*

*Le code s'exécute-t-il toujours avec le même temps d'exécution?*

# Worst Case Execution Time: WCET



*Comment obtenir une borne du WCET ?*

1. *Mesure du temps d'exécution*
2. *Analyse statique du code:*
  - *À base de flot*
  - *À base d'arbres*
  - *À base d'énumération*

# Worst Case Execution Time: WCET

Méthodologie basée sur des jeux de données des entrées:

- **Déterminer la combinaison des entrées** la plus pessimiste puis exécuter le code et mesurer le temps d'exécution: *problème difficile*
- **Exécuter le code** avec le maximum de combinaison d'entrées possibles et **mesurer le temps d'exécution** : *explosion combinatoire, par exemple 5 variables de 32 bits donnent  $4294967296^5$  combinaisons*
- **Ajouter une marge de sécurité**
- Limite : difficulté combinatoire , durée de tests trop long
- Alternative: Déterminer un ensemble suffisamment représentatif des entrées et lancer des exécutions sur cet ensemble. Pb: garantie du résultat

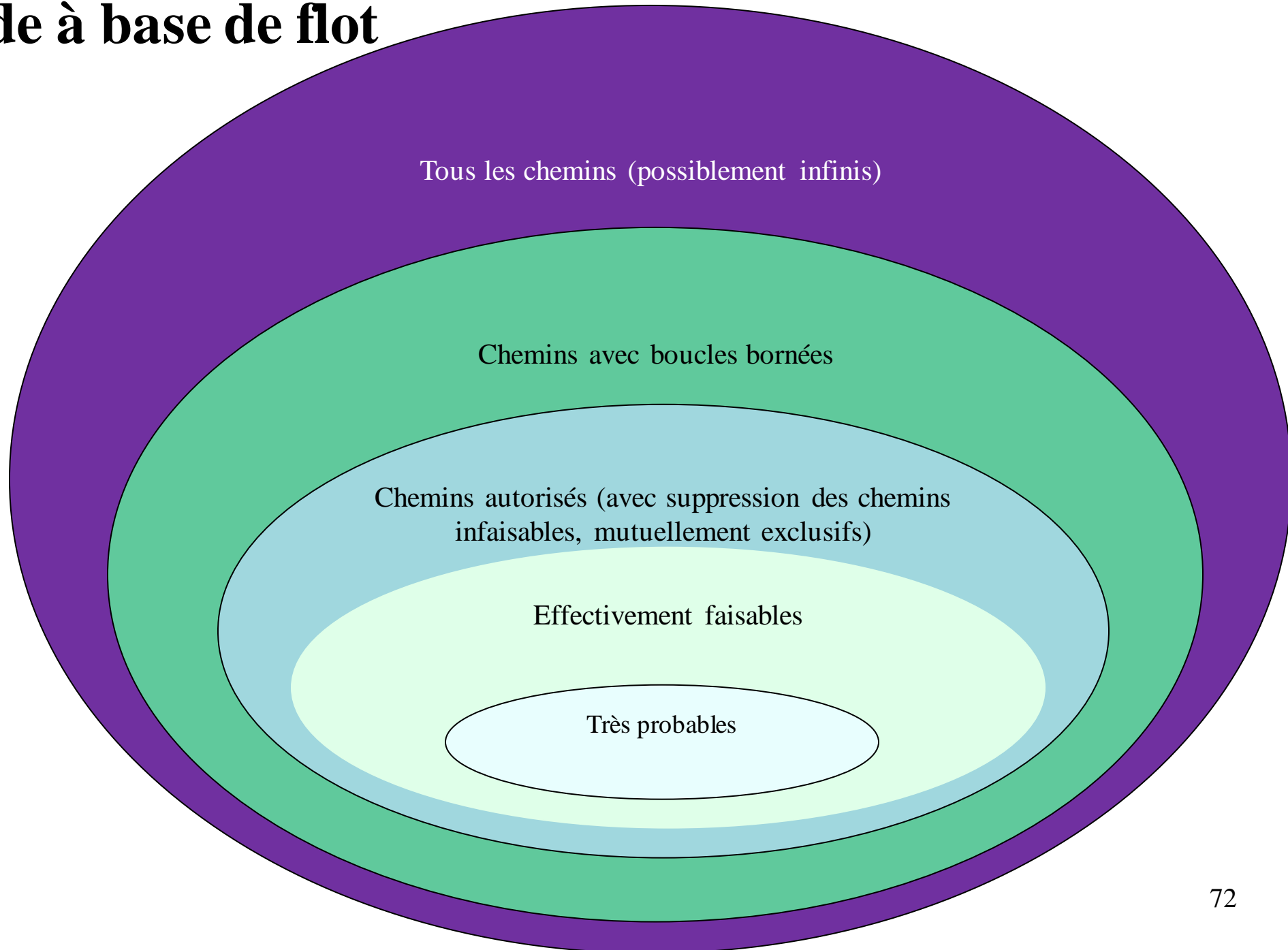
Comment faire la mesure?

- Logiciel : en utilisant l'horloge du système, simulation de CPU,
- Matériel + logiciel : en utilisant des instruments de mesures externes (oscilloscope,...) durant l'exécution du code.

### *Pourquoi ne pas simplement mesurer le WCET ?*

- *Mesurer toutes les différentes combinaisons d'exécution est **intraitable** ( $10^{40}$  pour une tâche de taille moyenne)*
- *La sélection des données de test peut ne pas contenir la trace d'exécution la plus longue*
- *La sélection des données de test peut ne pas contenir la trace de scénarios rares (interruption, ...)*
- *L'état interne du processeur peut n'être jamais passé par le pire cas*
- *Conclusion : les mesures peuvent convenir pour une estimation du WCET*

## Méthode à base de flot





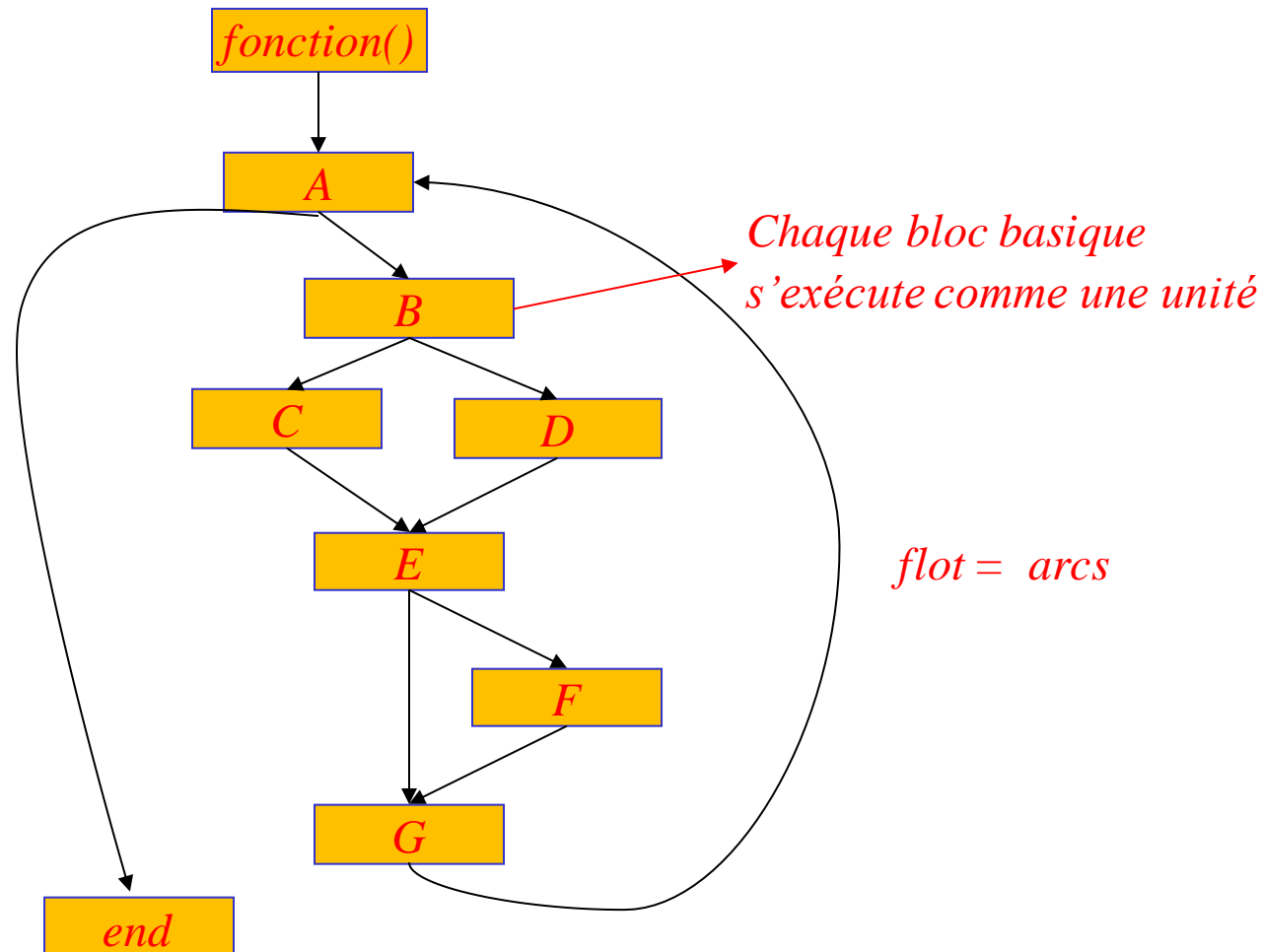
# WCET: méthodes d'analyse statique

## Méthode à base de flot :

*permet de déterminer une borne du nombre de fois que les différentes parties d'un programme sont exécutées (nombre d'itérations de boucle, profondeur d'une récursion, séquence infaisable).*

*fonction(x, i)*

```
A:   while (i < 100)
B:   if (x > 5) then
C:   x = x * 2;
      else
D:   x = x + 2;
      end
E:   if (x < 0) then
F:   b[i] = a[i];
      end
G:   i = i + 1;
      end
```



*Graphe de contrôle de flot*

## Méthode à base de flot :

*fonction(x, i)*

```
A:   while (i < 100)
B:   if (x > 5) then
C:   x = x * 2;
      else
D:   x = x + 2;
      end
E:   if (x < 0) then
F:   b[i] = a[i];
      end
G:   i = i + 1;
      end
```

*Borne de boucle : 100*

*Chemin infaisable : ABCEFG car C et F s'exclut mutuellement*

*Double boucle*

*triangle(a, b)*

```
for i := 1 to N do
for j := 1 to i do
A[i,j] = ....
end
```

*2 boucles bornées à N*

*Borne estimée à  $N * N$  (10 000 si  $N=100$ )*

*En réalité :  $(N + 1) * N / 2$  (5050 si  $N=100$ )*

## WCET: Influences du matériel

*Modéliser le comportement interne des processeurs?*

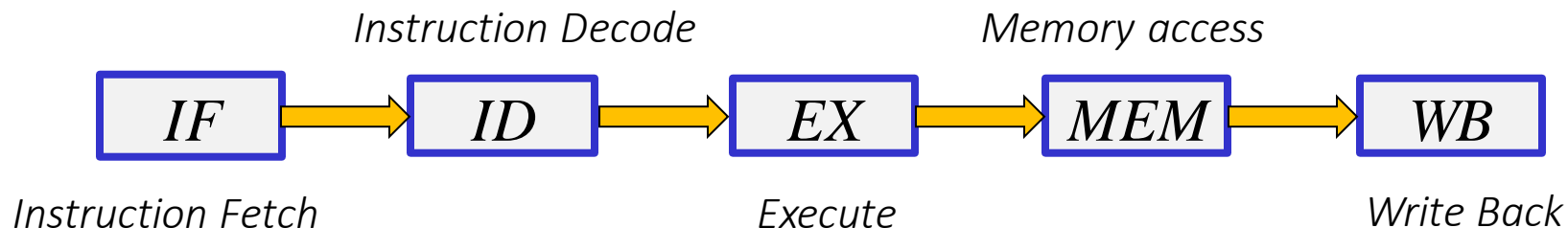
*Prendre en compte les modèles d'accès à la mémoire (cache,...) ?*

*Prendre en compte l'architecture des processeurs ? 8-16-32 ou 64 bits ?*

*Pipelines, accès caches (hit , miss)*

*Pipelines = parallélisme d'exécution au niveau hardware du processeur*

*Exemple processeur RISC classique: pipeline 5-niveaux*



*Idée : superposer les 5 niveaux d'activité du processeur pour créer du parallélisme d'exécution et donc réduire le temps d'exécution*

# WCET: Influences du matériel

	$i_{1,1}$	$i_{1,2}$	$i_{1,3}$	$i_{1,4}$	$i_{1,5}$	$i_{2,1}$	$i_{2,2}$	$i_{2,3}$	$i_{2,4}$	$i_{2,5}$
IF										
ID										
EX										
MEM										
WB										

*Pas de pipelining: l'instruction suivante, ne démarre qu'après la fin des 5 étapes de l'instruction actuelle.*

$$T = 10$$

IF										
ID										
EX										
MEM										
WB										

*Fonctionnement en pipeline: l'instruction suivante, démarre après la fin de la première étape (IF) de l'instruction actuelle.*

$$T = 6$$

*Accélération = longueur du pipeline*

*Problème de dépendance :*

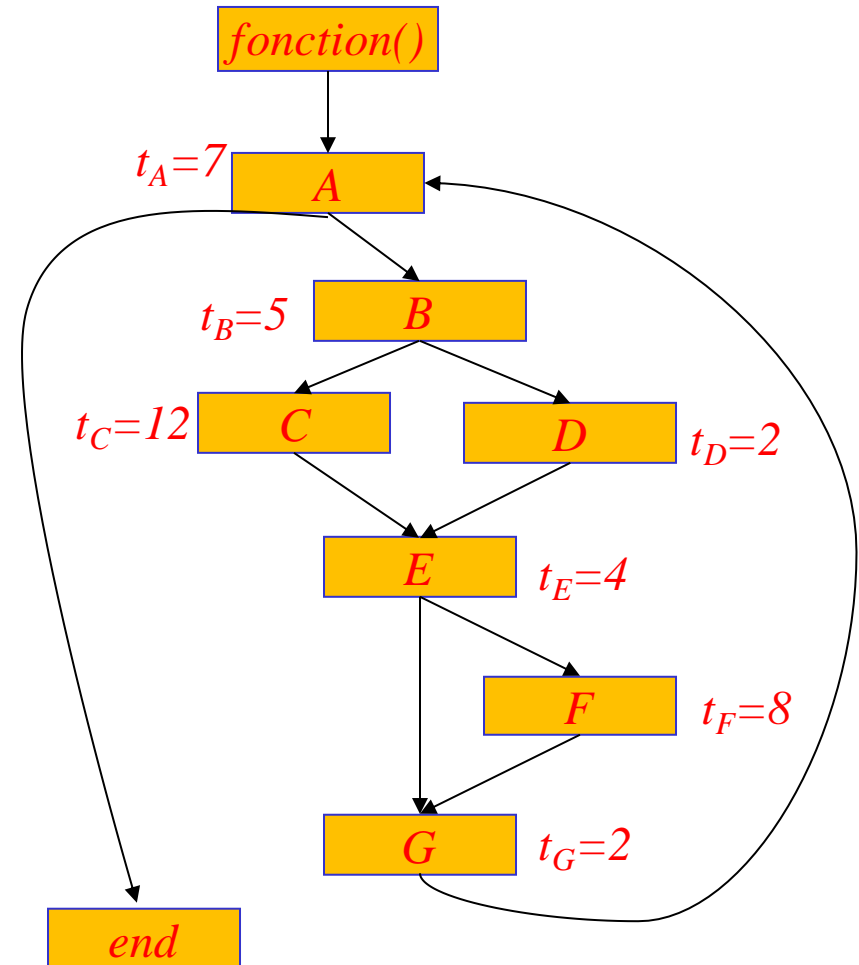
$i1$  : add  $\$r0$ ,  $\$r1$ ,  $\$r2$

$i2$  : add  $\$r3$ ,  $\$r0$ ,  $\$r4$

# WCET: Influences du matériel

*fonction(x, i)*

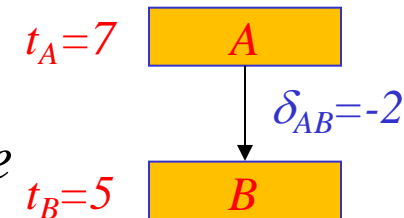
A:	<i>while (i &lt; 100)</i>	<i>7 cycles</i>
B:	<i>if(x &gt; 5) then</i>	<i>5</i>
C:	<i>x = x * 2;</i>	<i>12</i>
	<i>else</i>	
D:	<i>x = x + 2;</i>	<i>2</i>
	<i>end</i>	
E:	<i>if(x &lt; 0) then</i>	<i>4</i>
F:	<i>b[i] = a[i];</i>	<i>8</i>
	<i>end</i>	
G:	<i>i = i + 1;</i>	<i>2</i>
	<i>end</i>	



$t_i$  = durée pour un nœud

$\delta_{ij}$  = écart de durée entre nœud  $i$  et  $j$

Représente le gain de temps possible en pipeline



# WCET: Influences du matériel

$$t_A = 7$$

IF							
EX							
MEM							
F							

$$t_B = 5$$

IF							
EX							
MEM							
F							

$$t_{AB} = 10$$

IF									
EX									
MEM									
F									

$$\delta_{AB} = 10 - (7 + 5) = -2$$

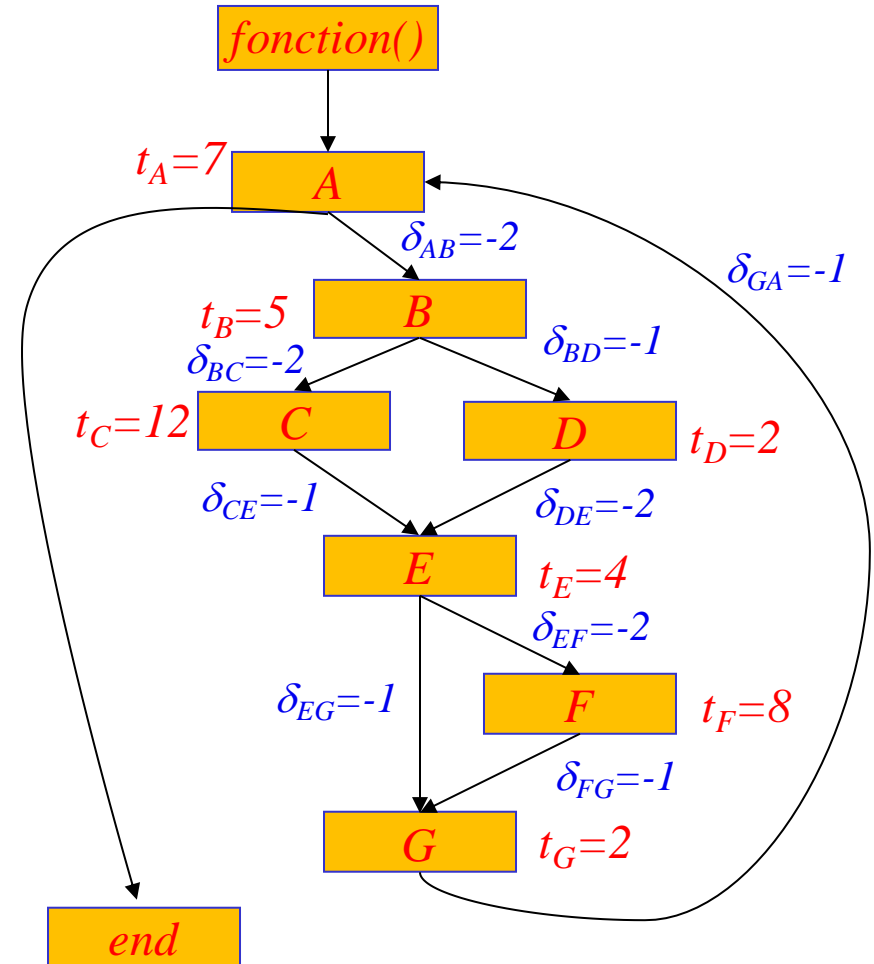
Calcul de  $\delta$  pour un séquence  $AB..YZ$  :  $\delta_{AB..YZ} = t_{AB..YZ} - t_{AB..Y} - t_{B..YZ} + t_{B..Y}$

Séquence  $AB$  :  $\delta_{AB} = t_{AB} - t_A + t_B = 10 - 7 - 5 = -2$

# WCET: Influences du matériel

*fonction(x, i)*

A:	<i>while (i &lt; 100)</i>	<i>7 cycles</i>
B:	<i>if(x &gt; 5) then</i>	<i>5</i>
C:	<i>x = x * 2;</i>	<i>12</i>
	<i>else</i>	
D:	<i>x = x + 2;</i>	<i>2</i>
	<i>end</i>	
E:	<i>if(x &lt; 0) then</i>	<i>4</i>
F:	<i>b[i] = a[i];</i>	<i>8</i>
	<i>end</i>	
G:	<i>i = i + 1;</i>	<i>2</i>
	<i>end</i>	



*Durée d'exécution pour une séquence :  $T(A..Z) = \sum t + \sum \delta$*

Exemple:

$$T(ABCEG) = 7 + 5 + 12 + 2 + 4 - (2 + 2 + 1 + 1) = 24$$

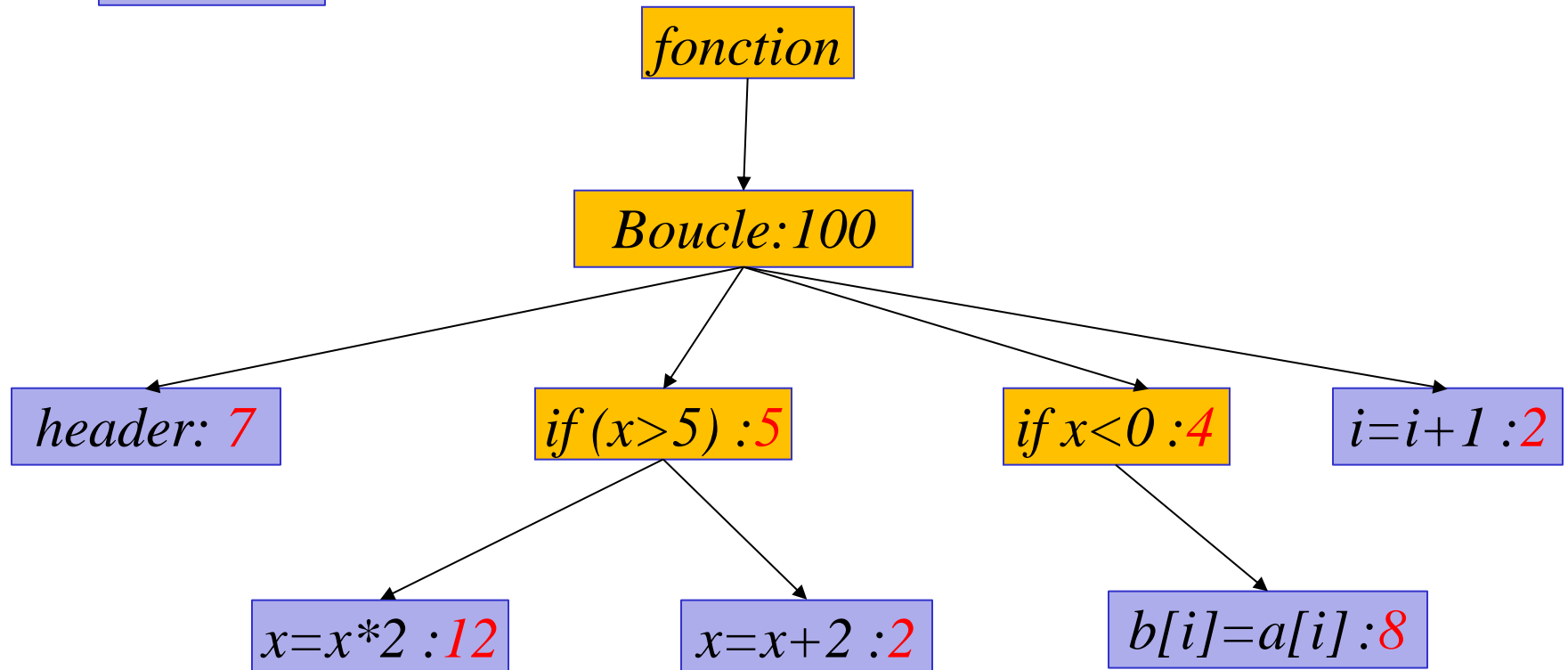
# WCET: Méthodes à base d'arbres

## Structure d'arbres :

- Arbre syntaxique du programme
- Blocs de base

## Principe:

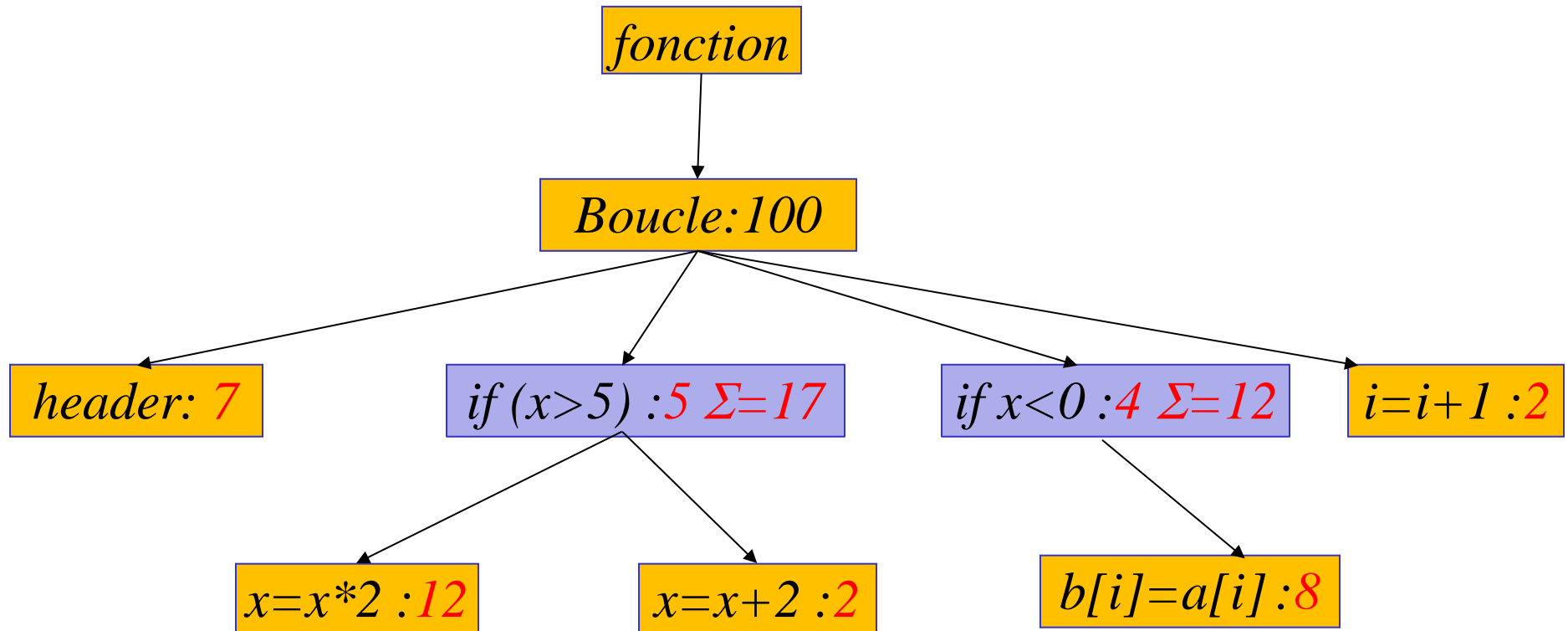
- Détermination du temps d'exécution de chaque bloc de base
- Calcul du WCET par analyse de l'arbre de bas en haut : identifier les feuilles terminales  $?: ?$  avec leur durée





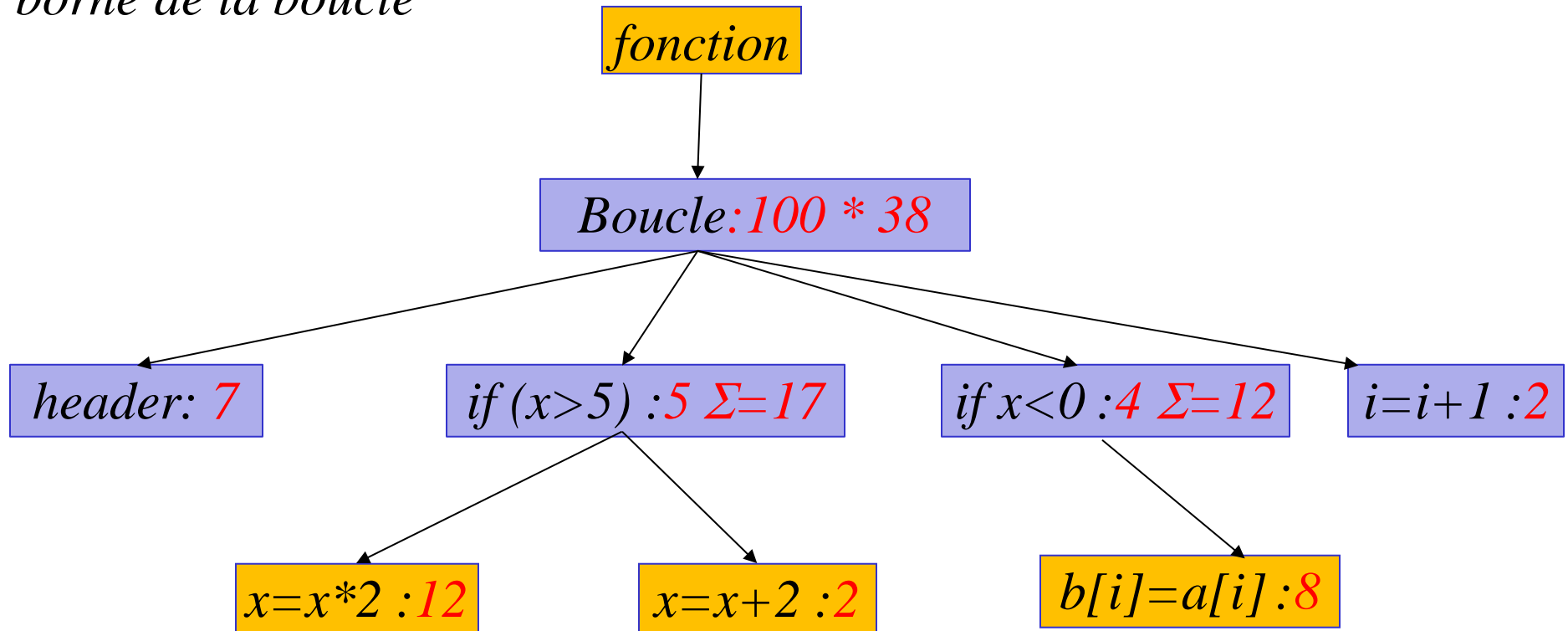
## WCET: Méthodes à base d'arbres

***Branches if** : prendre la valeur max des feuilles descendantes et faire la somme avec le bloc if*



## WCET: Méthodes à base d'arbres

**Boucles** faire la somme des feuilles descendantes puis multiplier par la borne de la boucle



*Conclusion : le temps d'exécution de fonction est estimé à 3800 cycles.*

# WCET: Méthodes à base de chemins

*Rechercher le chemin le plus long (une boucle à la fois)*

*Préparer la boucle*

- *enlever les arcs de boucle*
- *rediriger vers des feuilles 'continuer'*

*Chemin le plus long: ABCCEFG*

$$T = 7 + 5 + 12 + 4 + 8 + 2 = 38$$

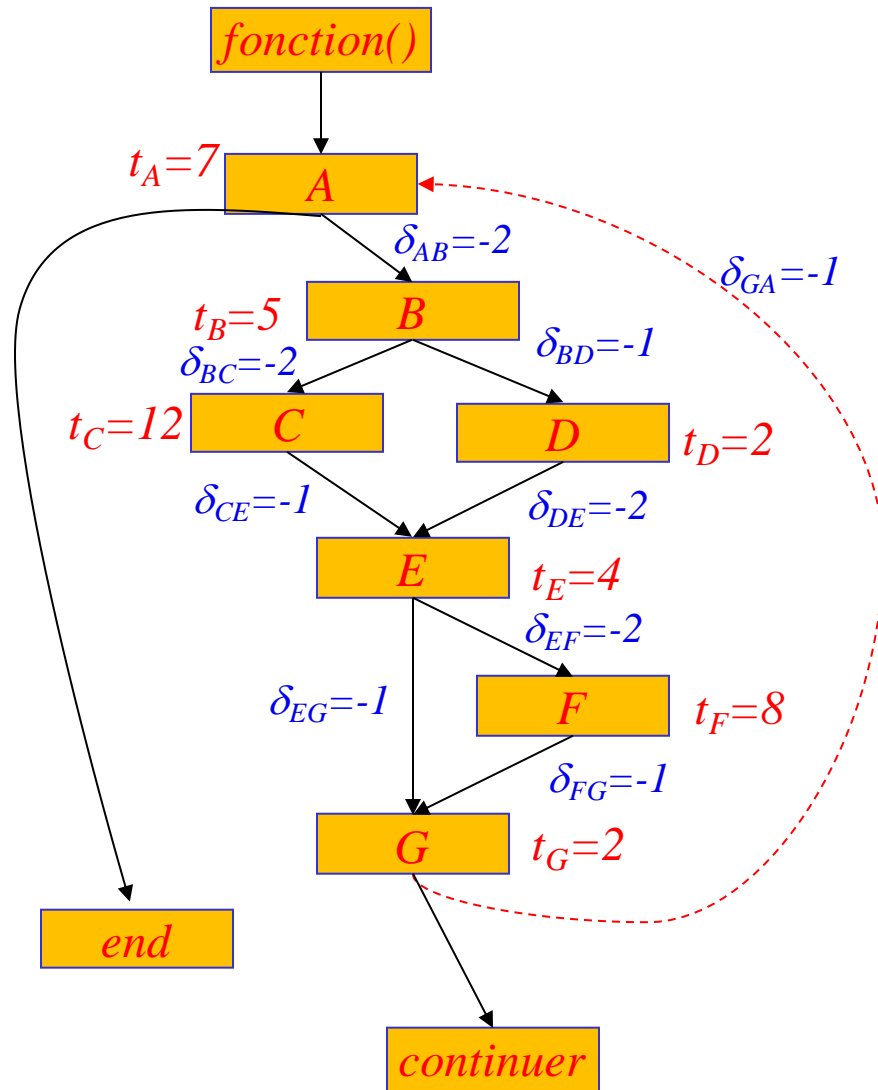
$$\text{Temps total} = 100 * 38 = 3800 \text{ cycles}$$

*Mais ABCCEFG est infaisable*

*nouveau chemin le plus long: ABCEG*

$$T = 7 + 5 + 12 + 4 + 2 = 30$$

$$\text{Temps total} = 100 * 30 = 3000 \text{ cycles}$$



# WCET: méthodes d'analyse statique

## IPET (Implicit Path Enumeration Technique)

*Le programme est représenté par un CFG (Control Flow Graph)*

*Utiliser ensuite des méthodes telles que : PLNE (Programmation linéaire en nombre entier) ou Programmation par contraintes pour résoudre le problème de maximisation (WCET)*

*On définit :  $X_i$  : nombre d'exécutions du bloc de base  $i$*

*$t_i$  : WCET du bloc de base  $B_i$*

$$WCET = \text{Max} \sum_0^n X_i \cdot t_i$$

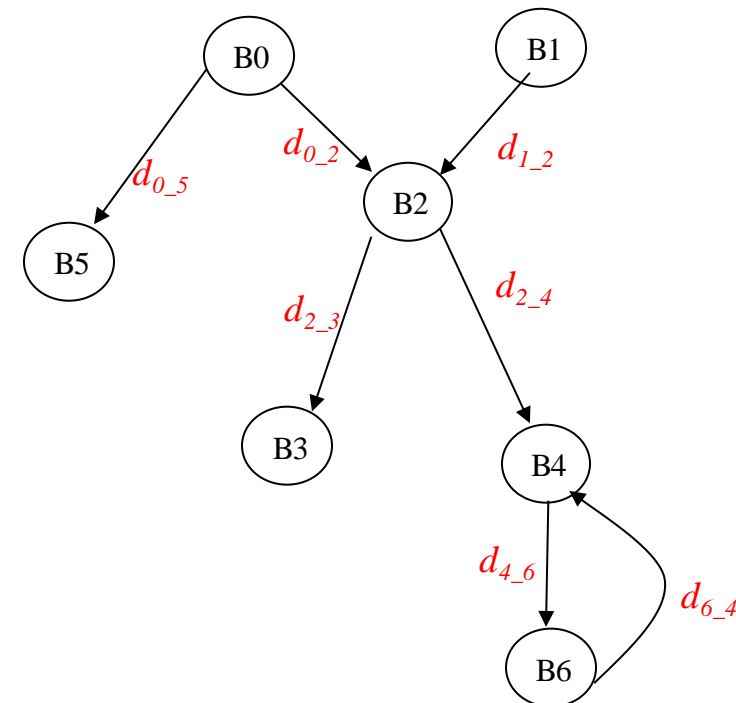
*Et  $d_{i\_j}$  = flot de contrôle d'exécution entre  $i$  et  $j$*

*Avec les contraintes suivantes (préservation de flots):*

$$X_i = \sum_{Bj \rightarrow Bi} d_{j\_i} = \sum_{Bi \rightarrow Bk} d_{i\_k}$$

$$\left. \begin{array}{l} X_2 = d_{0\_2} + d_{1\_2} \\ X_2 = d_{2\_3} + d_{2\_4} \end{array} \right\} d_{0\_2} + d_{1\_2} = d_{2\_3} + d_{2\_4}$$

*Contraintes de boucle ( $n$  itérations)  $d_{4\_6} \leq n \cdot d_{2\_4}$*



Maximiser :  $7*X1+5*X2+12*X3+2*X4+4*X5+8*X6+2*X7$

En respectant les contraintes:

$$X0 = 1$$

$$X8=1$$

$$X0=d_{0\_1}$$

$$X1=d_{0\_1}+d_{7\_1}=d_{1\_2}+d_{1\_8}$$

$$X2=d_{1\_2}+d_{1\_8}=d_{2\_3}+d_{2\_4}$$

$$X3=d_{2\_3}=d_{3\_5}$$

$$X4=d_{2\_4}=d_{4\_5}$$

$$X5=d_{3\_5}+d_{4\_5}=d_{5\_6}+d_{5\_7}$$

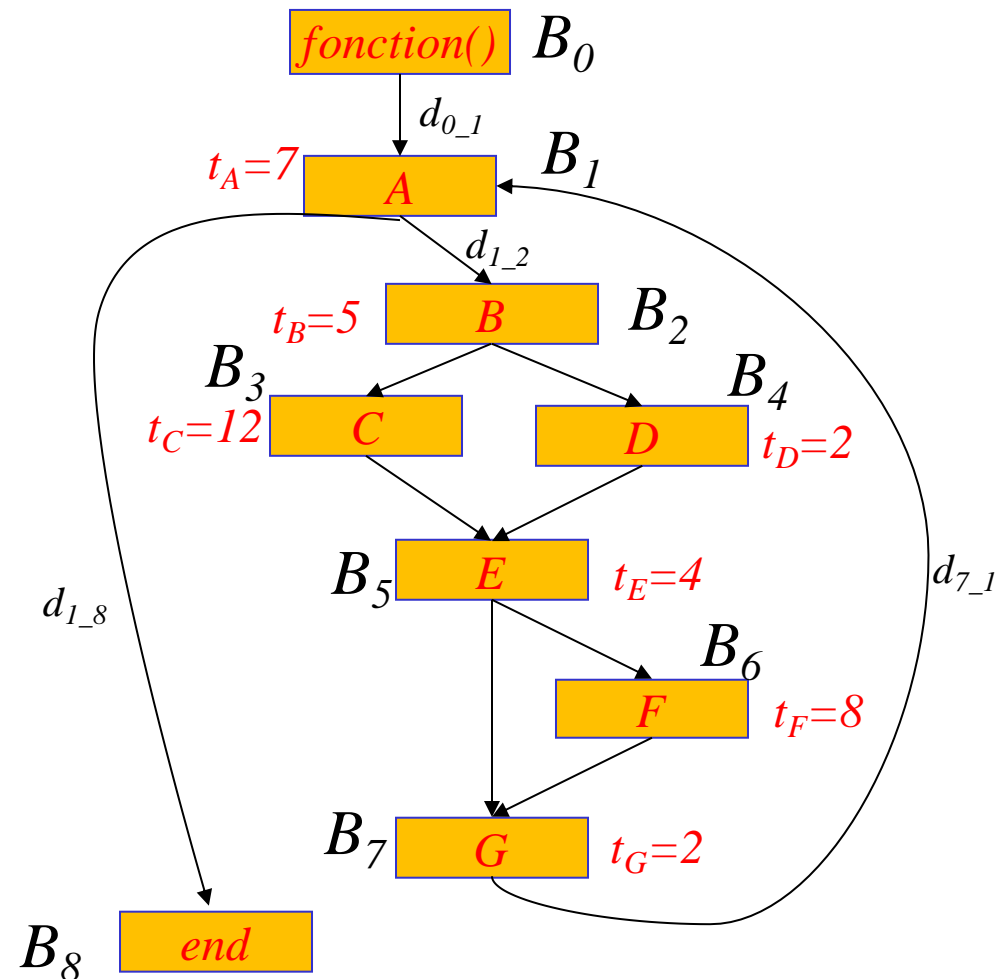
$$X6=d_{5\_6}=d_{6\_7}$$

$$X7=d_{5\_7}+d_{6\_7}=d_{7\_1}$$

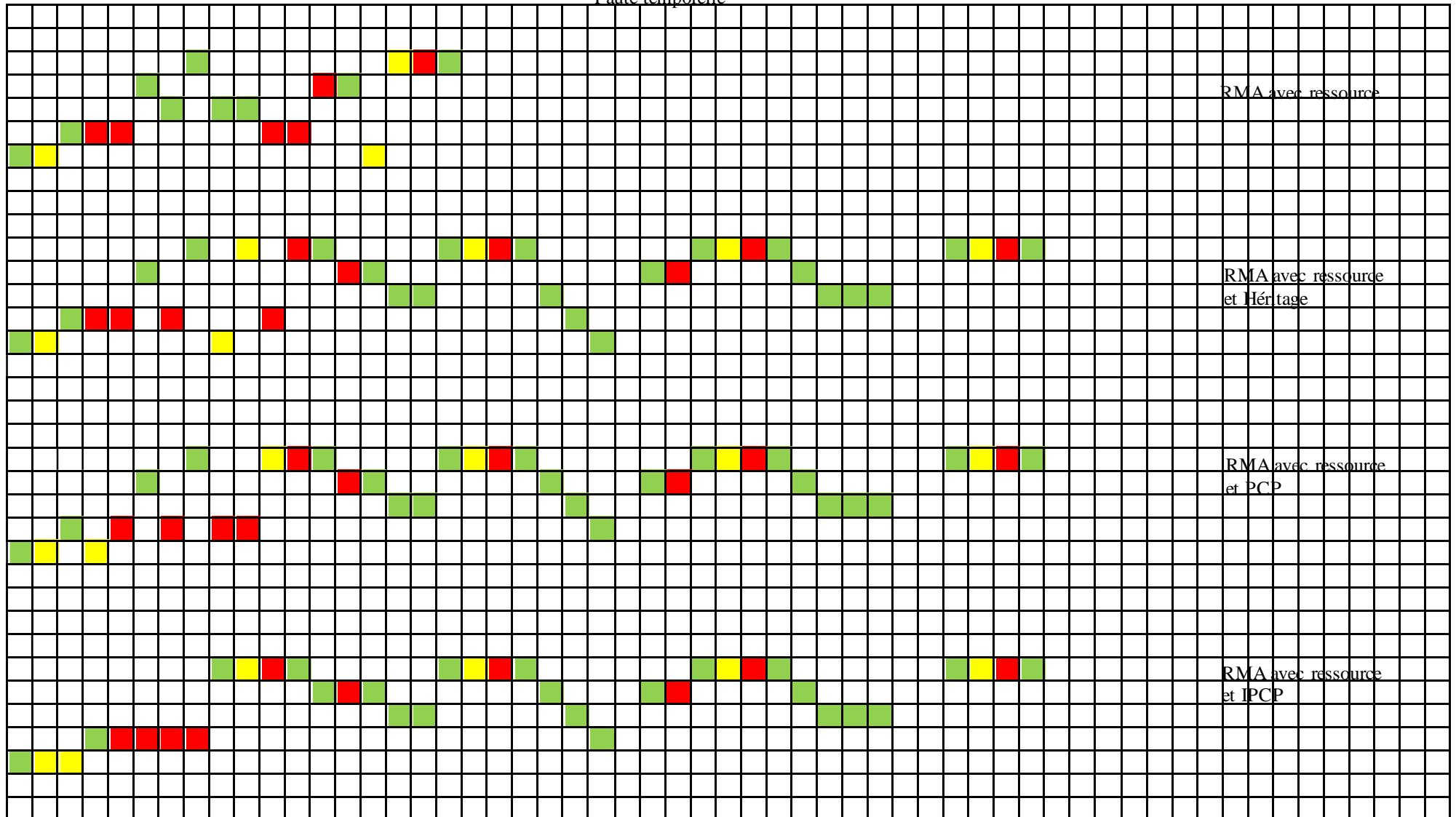
$$X8=d_{1\_8}$$

$$X2 \leq 100*X1$$

$$X3+X6=100 \text{ ?}$$



Faute temporelle



# Exercices sur les ordonnancements

Tâche	Période	Durée Ci	Priorité	$r_{i,0}$	Utilisation ressources
A	10	4	1	7	EVQE
B	20	3	2	5	EQE
C	20	3	3	5	EEE
D	40	6	4	2	EQQQQE
E	40	4	5	0	EVVE

1. ordonnançabilité de cette configuration (sans les ressources) avec RMA?
2. Chronogramme RMA avec ressources?
3. Calcul des WCRT sans prendre en compte les blocages liés aux ressources?
4. Chronogramme avec héritage de priorité ?
5. Chronogramme avec PCP ?
6. Chronogramme avec IPCP ?
7. Calculer les temps de blocage pour chaque tâche avec PCP.
8. Calcul des WCRT avec les temps de blocage?

$$WCRT_A = 4;$$

$$WCRTB^0 = 3; WCRTB^1 = 3 + \left\lfloor \frac{3}{10} \right\rfloor \times 4 = 7; WCRTB^2 = 3 + \left\lfloor \frac{7}{20} \right\rfloor \times 4 = 7; WCRTB = 7$$

$$WCRTC^0 = 3; WCRTC^1 = 3 + \left\lfloor \frac{10}{10} \right\rfloor \times 4 + \left\lfloor \frac{3}{20} \right\rfloor \times 3 = 10; WCRTC^2 = 3 + \left\lfloor \frac{10}{10} \right\rfloor \times 4 + \left\lfloor \frac{10}{20} \right\rfloor \times 3 = 10;$$

$$WCRTC = 10.$$

$$WCRTD^0 = 6; WCRTC^1 = 6 + \left\lfloor \frac{6}{10} \right\rfloor \times 4 + \left\lfloor \frac{6}{20} \right\rfloor \times 3 + \left\lfloor \frac{6}{20} \right\rfloor \times 3 = 16;$$

$$WCRTC^2 = 6 + \left\lfloor \frac{16}{10} \right\rfloor \times 4 + \left\lfloor \frac{16}{20} \right\rfloor \times 3 + \left\lfloor \frac{16}{20} \right\rfloor \times 3 = 20; WCRTC^3 = 6 + \left\lfloor \frac{20}{10} \right\rfloor \times 4 + \left\lfloor \frac{20}{20} \right\rfloor \times 3 + \left\lfloor \frac{20}{20} \right\rfloor \times 3 = 20;$$

$$WCRTD = 20.$$

$$WC RTE^1 = 4 + \left\lfloor \frac{4}{10} \right\rfloor \times 4 + \left\lfloor \frac{4}{20} \right\rfloor \times 3 + \left\lfloor \frac{4}{20} \right\rfloor \times 3 + \left\lfloor \frac{4}{20} \right\rfloor \times 6 = 20; WC RTE^2 = 4 + \left\lfloor \frac{20}{10} \right\rfloor \times 4 + \left\lfloor \frac{20}{20} \right\rfloor \times 3 + \left\lfloor \frac{20}{20} \right\rfloor \times 3 + \left\lfloor \frac{20}{20} \right\rfloor \times 6 = 24;$$

$$WCRTD^3 = 6 + \left\lfloor \frac{24}{10} \right\rfloor \times 4 + \left\lfloor \frac{24}{20} \right\rfloor \times 3 + \left\lfloor \frac{24}{20} \right\rfloor \times 3 + \left\lfloor \frac{24}{20} \right\rfloor \times 6 = 36;$$

$$WC RTE^3 = 6 + \left\lfloor \frac{36}{10} \right\rfloor \times 4 + \left\lfloor \frac{36}{20} \right\rfloor \times 3 + \left\lfloor \frac{36}{20} \right\rfloor \times 3 + \left\lfloor \frac{36}{20} \right\rfloor \times 6 = 40; WC RTE = 40.$$