# PLAN

- Généralités systèmes temps réel
  - Problématique temps réel
  - Exécutif temps réel
- Langage Temps Réel
  - Notion de moniteur
  - Ressources
  - Sémaphore
  - Communication entre tâches
- Systèmes à temps critique
  - Bases de l'ordonnancement

# Initiation au Temps-Réel

Objectif du cours

Généralités (bases) et Terminologie

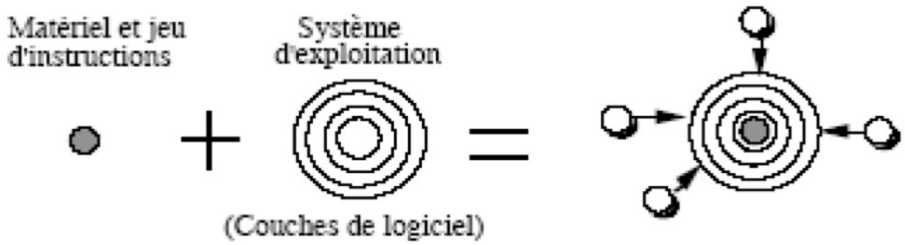
Initiation au TR basée sur le langage LTR
(indépendant de tout OS RT)

Comprendre les problèmes fondamentaux
Apprendre les mécanismes essentiels

# Généralités

**Systèmes d'exploitation … quel rôle et quels objectifs ?**

***Rôle***




Assurer l'indépendance vis à vis des aspects matériels, c'est-à-dire construire une « machine virtuelle » sur la machine physique.

Organiser et optimiser l'utilisation des ressources (matérielles et logicielles) :
- processeurs, mémoire, fichiers, communications, etc.

Procurer une interface utilisateur conviviale.

# Généralités

## *Objectifs*

- Système **transactionnel** : les instants de production des résultats ne sont pas contraints. Cohérence des données, synchronisation des accès, … (parfois classés en systèmes interactifs).

➔ **systèmes transformationnels**

- Système **partagé** : c'est le S.E qui impose sa base de temps aux utilisateurs (utilisation interactive de la machine par un ou plusieurs utilisateurs… partage équitable), les instants de production des résultats respectent des valeurs statistiques.

➔ **systèmes interactifs**

- Système **temps-réel** : c'est l'environnement qui impose sa base de temps au système! Il doit donner le contrôle à la tâche qui est vitale à un instant donné. Sa contrainte principale sera de délivrer les résultats attendus **avant une date limite appelée échéance**, le non-respect de l'échéance pouvant mettre en danger l'installation contrôlée par la machine.

➔ **systèmes réactifs**

# Système réactif

## Système réactif

### *Définitions*

**Définition synthétique …**

*« une application temps réel met en oeuvre des systèmes informatiques ou informatisés coopérant avec l'homme et destinés à la perception, l'observation, l'aide à la décision et la conduite de procédés dynamiques ».*

**Définition du CNRS …**

*« est qualifié de temps réel le comportement d'un système informatique lorsqu'il est assujetti à l'évolution dynamique d'un procédé qui lui est connecté, et qu'il doit piloter ou suivre en "réagissant" à tous ses changements d'état »*

# Système réactif

**Le système réactif …**

- se réfère aux « interactions » entre le procédé et le système informatique : mesures, événements, commandes
- se réfère au « temps » : les interactions entre procédé et système doivent être «instantanées», mais :
  - le fonctionnement d'un système informatique ne permet qu'une observation échantillonnée de son environnement
  - calculer et appliquer une nouvelle commande est une opération du système informatique qui ne peut être instantanée (temps de calcul)

  donc :
  - réagir « dans les temps » implique une observation et une action suffisamment rapides pour que le procédé n'ait pas le temps d'évoluer de façon significative entre une variation d'état et l'action qui en résulte. Le temps réel reflète donc une capacité de réaction à l'échelle de temps du procédé : c'est un **temps relatif**.

la validité de ses réactions ne dépend pas seulement de la justesse des calculs, mais aussi de l'instant de production des résultats. Pour une application temps réel, un résultat juste mais hors délai est un résultat faux

# Architecture & Application

*Architecture / Application*

1- Architecture logicielle (structuration de l'application) et architecture matérielle (entités supportant l'application comme les processeurs par ex) sont étroitement liées. Nous n'évoquerons que l'architecture logicielle.

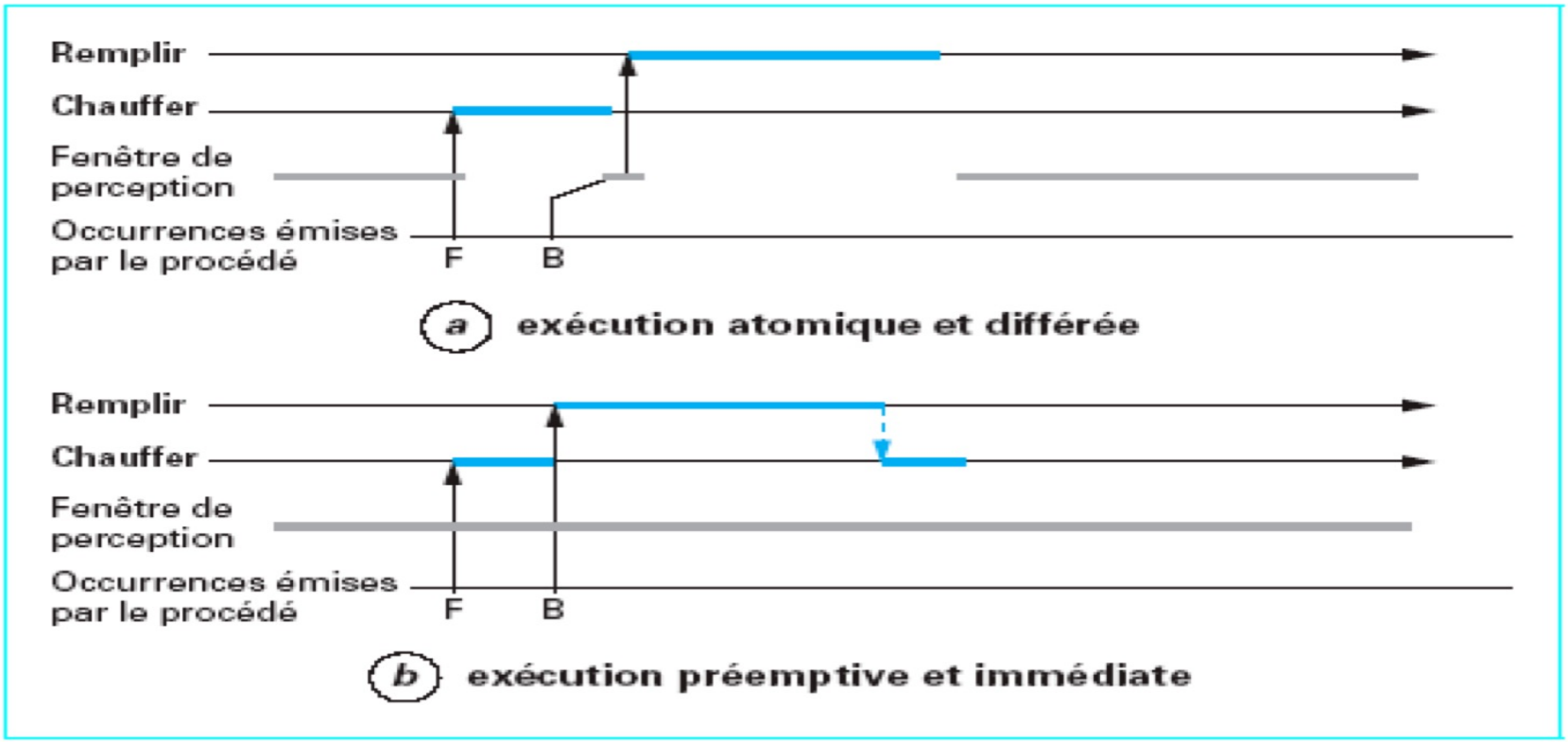
2- Sur une architecture distribuée sur plusieurs processeurs, le réseau devient alors la colonne vertébrale des communications intra-système. Ses performances et capacités (son profil) jouent, à leur tour, un rôle essentiel dans le respect des contraintes de sûreté et de temps.

3- L'architecture opérationnelle comprend à la fois la définition de l'architecture matérielle, la décomposition fonctionnelle de l'application et la projection de ces « fonctions » sur l'architecture matérielle.

# Approches de conception

Techniques de conception/réalisation de l'application

Deux approches de conception différentes tant sur la prise en compte des événements que sur l'exécution des actions résultant de leur occurrence : approches synchrone et asynchrone. La première repose sur la notion d'exécution atomique, l'autre d'exécution préemptive.



# Approches de conception

*Approche synchrone*

L'approche synchrone repose sur l'expression des relations directes entre une tâche et son événement « déclencheur », sans prise en compte du contexte au sens des autres tâches de l'application potentiellement en cours (i.e. en conflit). La démarche consiste à déterminer une exécution séquentielle des tâches, sans avoir recours à la préemption, et prévoir l'utilisation (l'accès) de toute ressource partagée.

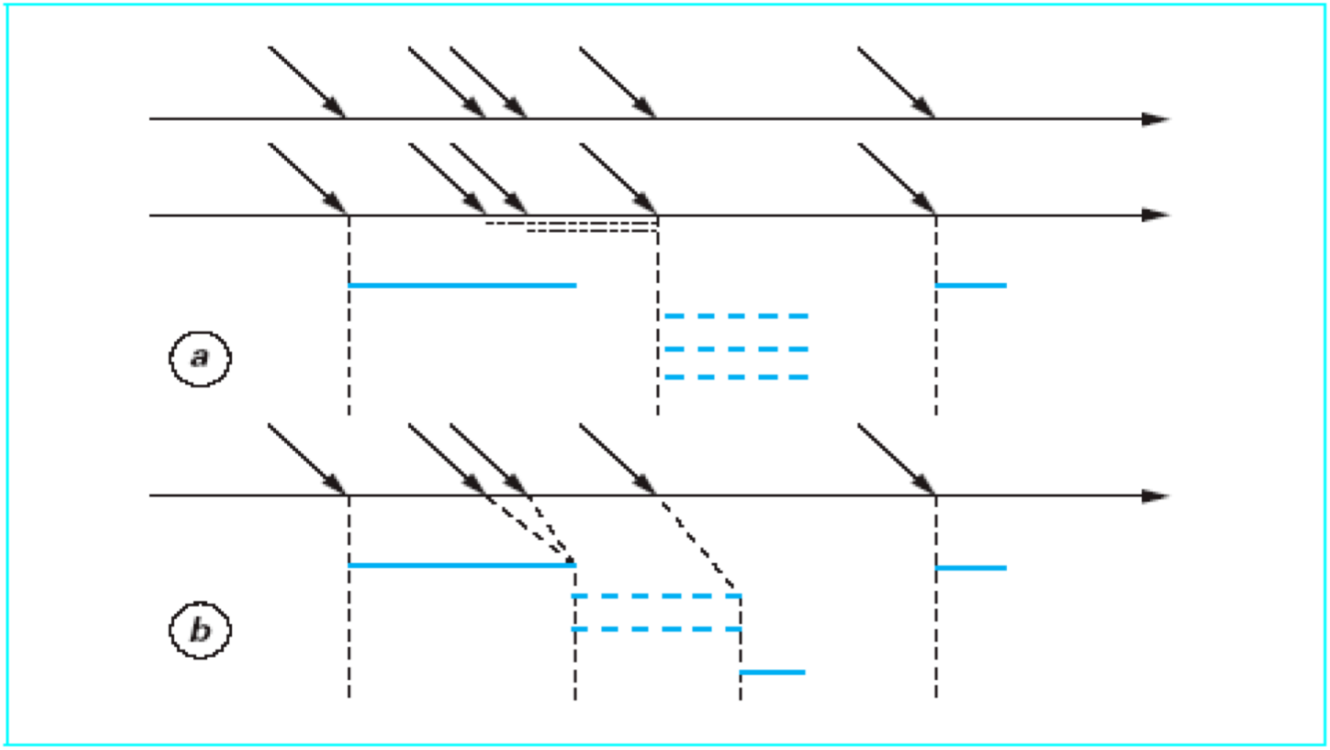
Cependant, la non-préemption pose des problèmes sur le plan temporel :
- la prise en compte des événements est différée du temps d'exécution de la tâche en cours … avec un risque de mauvaise réactivité (a fortiori selon la « gravité » du phénomène) et de cascade d'événements.
-la conception de l'architecture fonctionnelle est importante, notamment la décomposition en tâches … de façon à éviter la monopolisation du processeur par une tâche de faible importance.

Donc les contraintes temporelles de l'application doivent être analysées dès la conception …

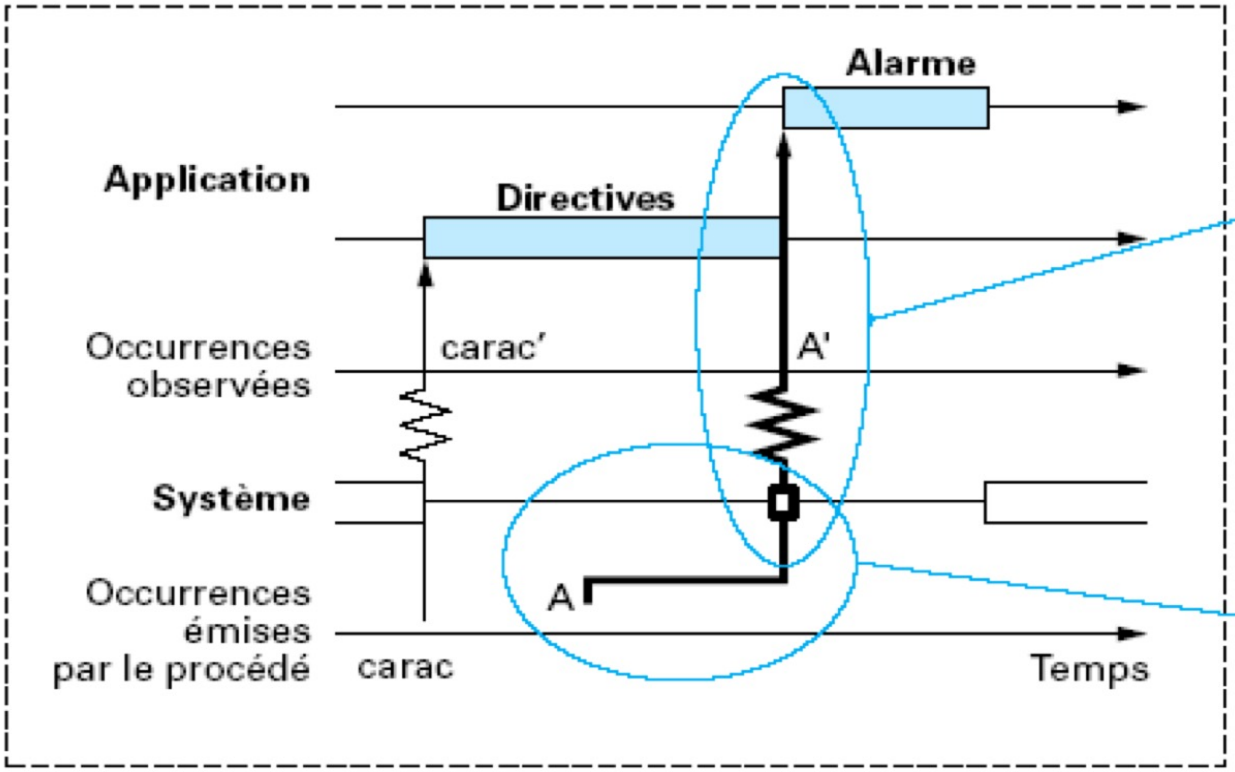
# Approches de conception

La gestion des événements, qui ne sont constatés qu'à l'issue de la tâche en cours (donc avec retard), est délicate : sans connaître leur ordre d'arrivée, comment déterminer l'action à exécuter si l'ordre est primordial ?

Rappel : dans l'approche synchrone, tous ces événements sont considérés comme simultanés (observés au même instant et considérés comme survenus en même temps) ... l'action à entreprendre dépend de la combinaison des événements.



# Approches de conception




En n'observant les occurrences d'événements émis par le procédé qu'à la fin de l'exécution de chaque action, la réaction du système est perçue comme immédiate ... par le système de contrôle.

La prise en compte retardée de certaines occurrences d'évènements est une réaction perçue comme différée ... par le procédé.

*Chronogramme d'une exécution synchrone*

Source : technique de l'ingénieur

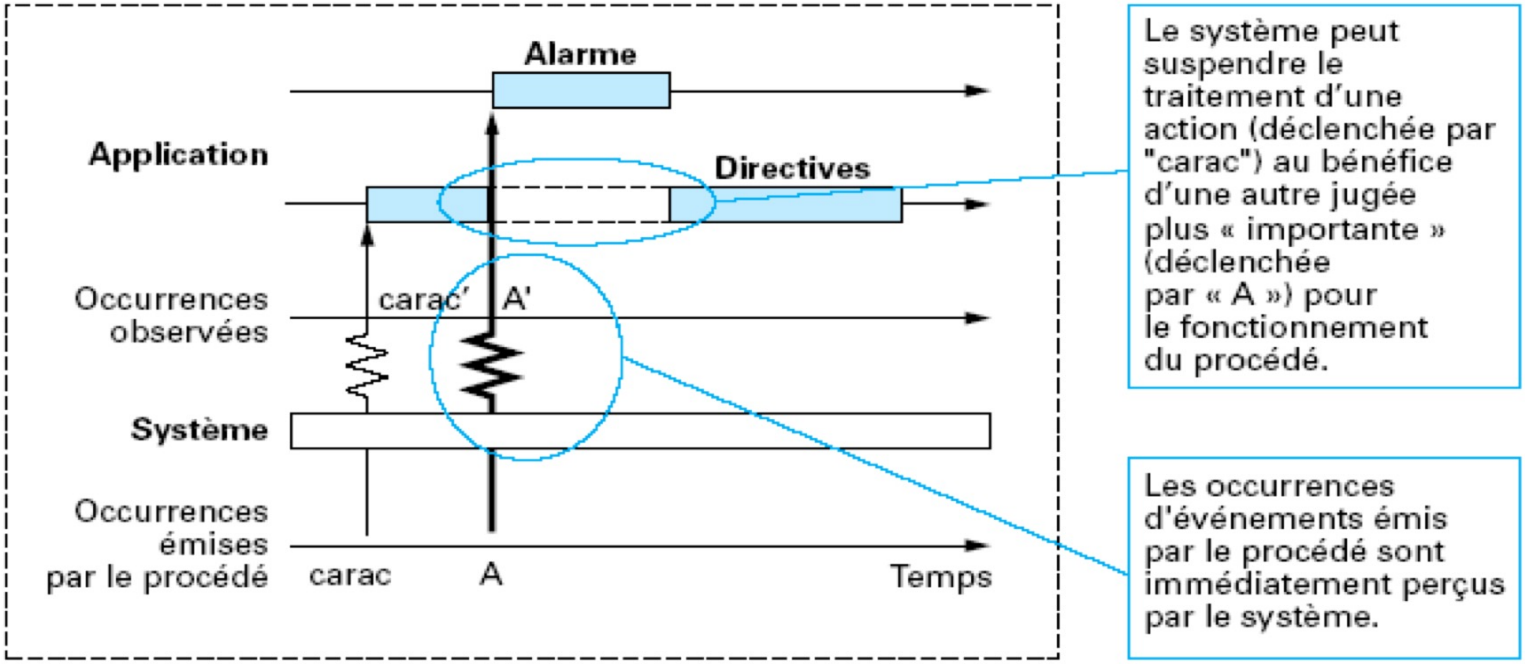
# Approches de conception

*Approche asynchrone*

Contrairement à l'approche synchrone, la technique asynchrone consiste à observer en permanence les occurrences d'événement (même lors de l'exécution des tâches). L'ordre d'occurrence (et les dates) est donc déterminé: le choix de l'action à réaliser est par conséquent non ambiguë …. Un événement nécessitant une réaction « urgente » peut être immédiatement traité.
Cependant la préemption, outre son coût temporel (latences diverses), engendre de nouveaux problèmes :
- Pour s'assurer du respect des contraintes de temps, il faudrait envisager toutes les possibilités de préemption (i.e. tous les scénarii d'occurrence d'événements) … inenvisageable.
- Utiliser une modélisation formelle du comportement de l'application pour pouvoir vérifier cela est délicat car les modèles et outils (en asynchrone) qui permettent une telle preuve sont actuellement très complexes.
- Les applications asynchrones sont souvent qualifiées d'indéterministes, des événements très proches pouvant être considérés comme distincts mais à l'échelle de temps du système contrôlé elles pourraient ne pas devoir être distinguées, donnant lieu à une autre réaction (que celle décidée en les distinguant).

# Approches de conception

- L'affectation du processeur aux tâches à exécuter est un enjeu majeur.
Quelle est la tâche concernée par l'événement mais surtout doit-elle être exécutée immédiatement (i.e. préempter la tâche en cours) ou faut-il l'allouer à une autre tâche ?




*Chronogramme d'une exécution asynchrone*

Source : technique de l'ingénieur

# Approches de conception

Dans un système asynchrone, l'ordonnancement des tâches devient un élément majeur. Le choix de la tâche à exécuter a une influence directe sur le respect des contraintes temporelles. La technique retenue (différents algorithmes) pour effectuer ce choix relève de la **politique d'ordonnancement.**

Enfin, une fois l'ordonnancement déterminé (pour essayer de garantir le respect des contraintes), il faut le mettre en œuvre…..intervient alors l'exécutif (du système d'exploitation) qui lancera l'exécution des tâches, i.e. attribuera le processeur à telle ou telle tâche à chaque instant : l'ordonnanceur. On aura recours à un **exécutif temps-réel.**

Cadre du cours

Approche **asynchrone, Application multi-tâches,**
Calculateur Monoprocesseur, **Exécutif temps réel.**

# Eléments de base d'un exécutif temps-réel

***Rôle***

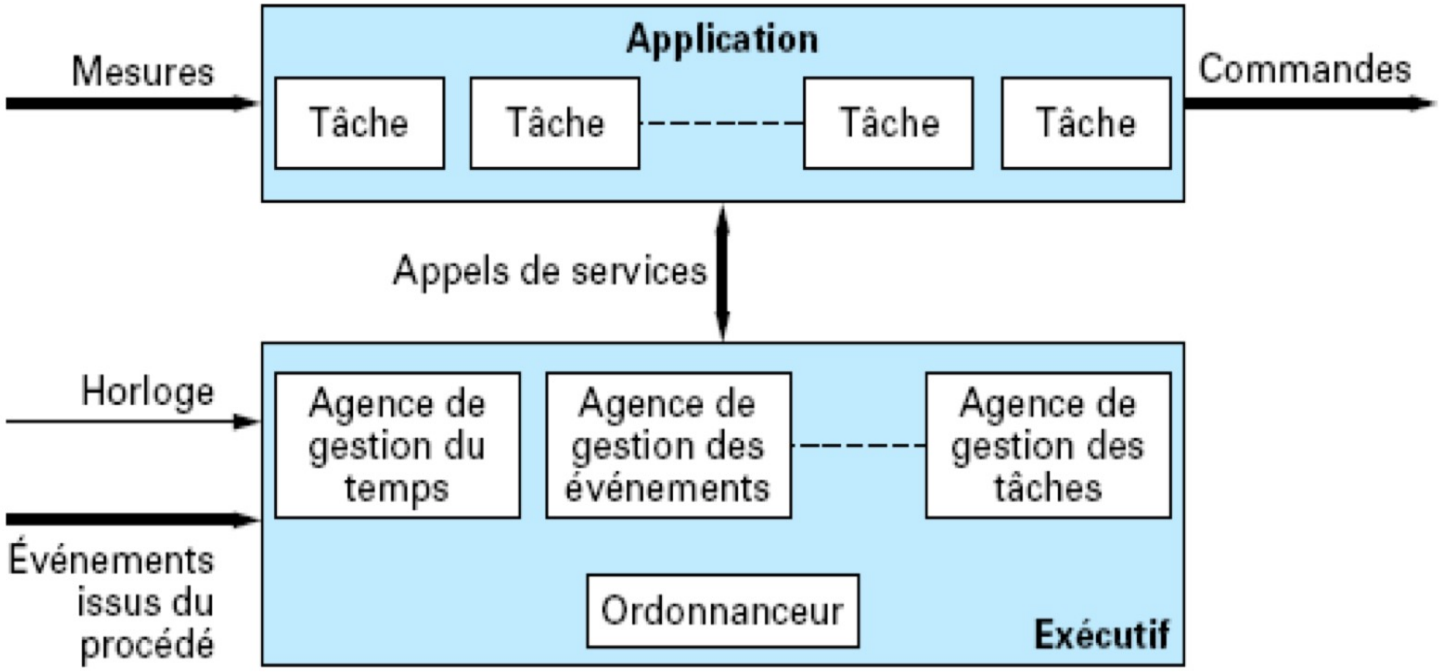
Un exécutif temps réel est souvent « dirigé par les événements ». Ce qualificatif illustre bien son comportement. En effet, les appels à l'exécutif sont la conséquence :

- des occurrences d'événements issus du procédé ; concrètement, l'appel à l'exécutif est effectué par la procédure de réception et de traitement de l'interruption matérielle associée à ces événements ;

- du temps ; concrètement, l'appel à l'exécutif est provoqué par l'interruption régulièrement engendrée par une horloge temps réel équipant le calculateur ;

- des tâches elles-mêmes ; c'est le cas lorsqu'une tâche requiert des services offerts par l'exécutif.

# Eléments de base d'un exécutif temps-réel




*Représentation schématique : application et exécutif*

*(source : technique de l'ingénieur)*

# Eléments de base d'un exécutif temps-réel

Son rôle :

- ordonnancer les exécutions des tâches (priorités, stratégies d'ordonnancement)
- protéger l'accès aux ressources partagées,
- réveiller les tâches en attente d'un délai ou d'une heure de démarrage
- transmettre les signaux de synchronisation, les données entre des tâches asynchrones,
- etc.
- Mais aussi, offrir des services accessibles directement par l'utilisateur :
  - de gestion des tâches (activation, terminaison forcée, etc.)
  - de gestion des événements matériels (interruption) et de synchronisation
  - de communication de messages entre tâches (boîtes aux lettres, appels client-serveur,…)
  - de gestion du temps (réveil différé de tâches)
  - de gestion des ressources partagées (mémoire, …)

**NB** : Dans un système temps réel, les ressources, autres que le processeur, sont souvent attribuées statiquement aux tâches, au moment de leur création. En particulier, on ne peut accepter la perte de temps que cause l'allocation dynamique de la mémoire

# Eléments de base d'un exécutif temps-réel

## *Performances d'un exécutif temps réel*

Il est délicat de définir des critères d'évaluation des exécutifs du marché en raison de la diversité des produits, de la sémantique des objets qu'ils manipulent, de leur champ applicatif, de leur environnement de développement... Dans le cadre de la standardisation, POSIX apporte néanmoins une réponse à ce problème en établissant des métriques applicables aux exécutifs standard. Voici quelques mesures typiques pouvant servir de comparatifs dans des conditions expérimentales précises :

— « **interrupt latency** » est le temps pendant lequel les interruptions sont masquées dans l'exécutif (manipulation de structures critiques), et donc ne peuvent être prises en compte

— « **task response time** » est le temps entre l'occurrence d'une interruption et l'exécution de la tâche réveillée. Il dépend des services invoqués, de la structure de la routine d'interruption (est-elle interruptible, et si oui combien de fois pendant son exécution ?), de la valeur précédente, du temps pris par l'ordonnancement (selon la configuration)...

— « **premptive latency** » est le temps maximal pendant lequel le noyau peut retarder l'ordonnanceur (manipulation de structures critiques). C'est un temps difficile à mesurer, donc rarement fourni. Il est pourtant très important car il est illusoire de garantir un temps de réponse pour une tâche, de par exemple 20 µs, si le noyau régulièrement interdit la préemption pendant 100 µs !

En conclusion, il est certainement difficile d'évaluer et donc de comparer les performances des exécutifs.

# Initiation au Langage Temps-Réel

## INTRODUCTION

Ce cours traite des concepts des "Systèmes Temps Réel", à savoir

Comment écrire et faire dérouler des programmes qui dépendent du temps (par exemple contrôlant un processus).

Qu'est-ce qu'un Moniteur TR ?

Qu'est-ce qu'une "tâche" ?

Quels sont les types de tâches coopérant dans un système TR

Qu'est-ce qu'un événement TR ?

Qu'est-ce qu'une ressource ?

# Initiation au Langage Temps-Réel

LTR a les caractéristiques suivantes:

***- Instructions procédurales classiques:***

| | |
|---|---|
| Variables | Numériques, Booléennes, Chaînes |
| Instructions | Opérations (+, -, *, /, etc.) |
| Contrôle | SI...ALORS...SINON |
| Entrés/Sorties | IN, OUT, PRINT, etc. |

- ***plus: les "concepts TEMPS REEL »***

| | |
|---|---|
| TACHES | (Exécution de programmes) |
| EVENEMENTS | (Attendu par les tâches) |
| RESSOURCES | (gère le partage de programmes) |

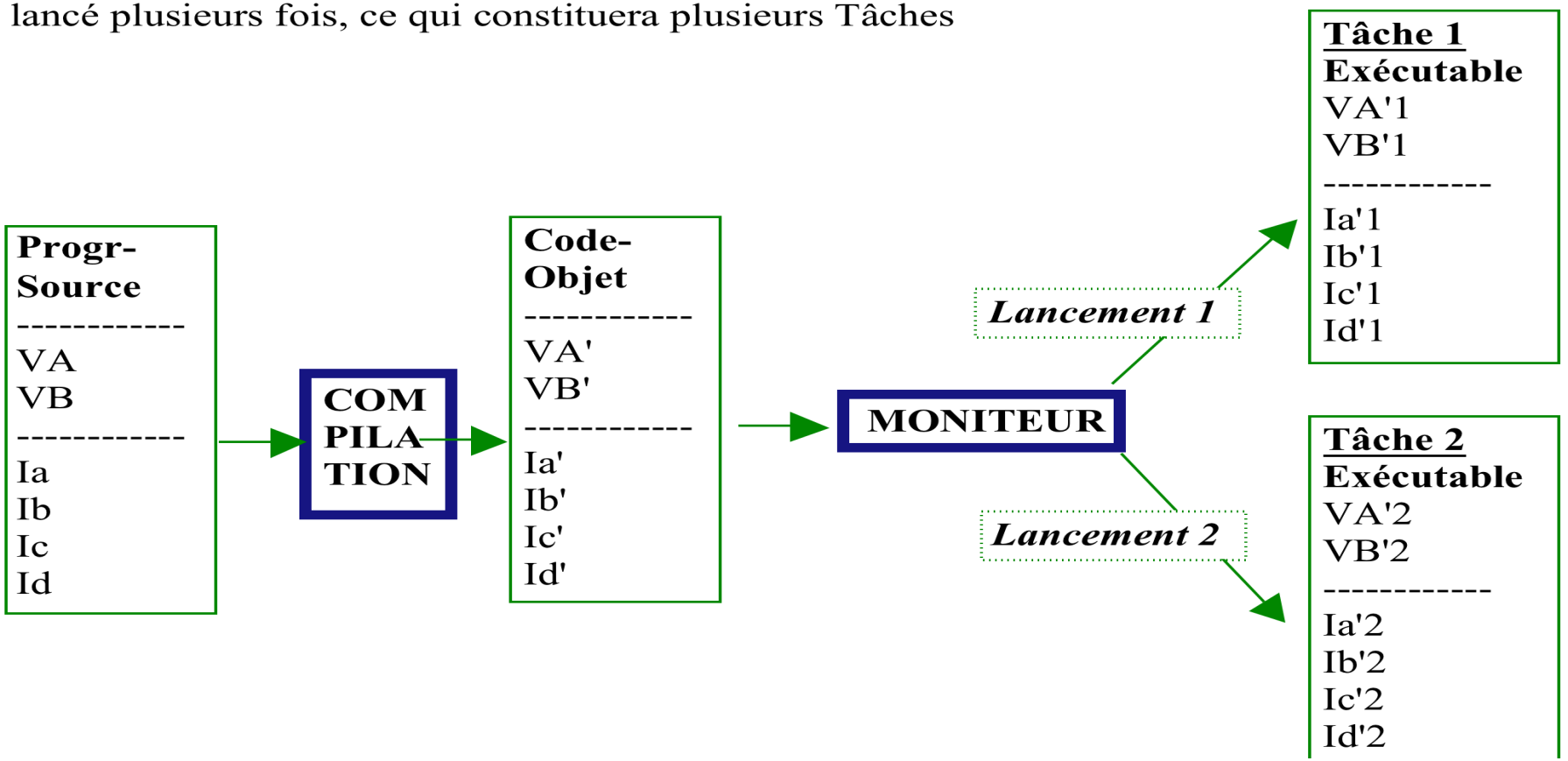
- ***plus: le MONITEUR***
  ***qui "gère" l'ensemble en tenant compte réellement du temps (événements, dates)***

# Initiation au Langage Temps-Réel

## notion de TACHE

Une TACHE est l'exécution d'un Programme à un instant donné

**Différence entre "Tâche" et "Programme"**
Le Programme est une suite d'instructions. Cet ensemble d'instructions peut être lancé plusieurs fois, ce qui constituera plusieurs Tâches



# Initiation au Langage Temps-Réel

**Chaque tâche fonctionne sur un jeu de données qui lui est propre**

**Lors du lancement d'une tâche il y a**
- création d'un "jeu de Données" par tâche
- chargement du code

**Quand la même tâche est lancée plusieurs fois,**

le jeu de données est créé autant de fois que de tâches

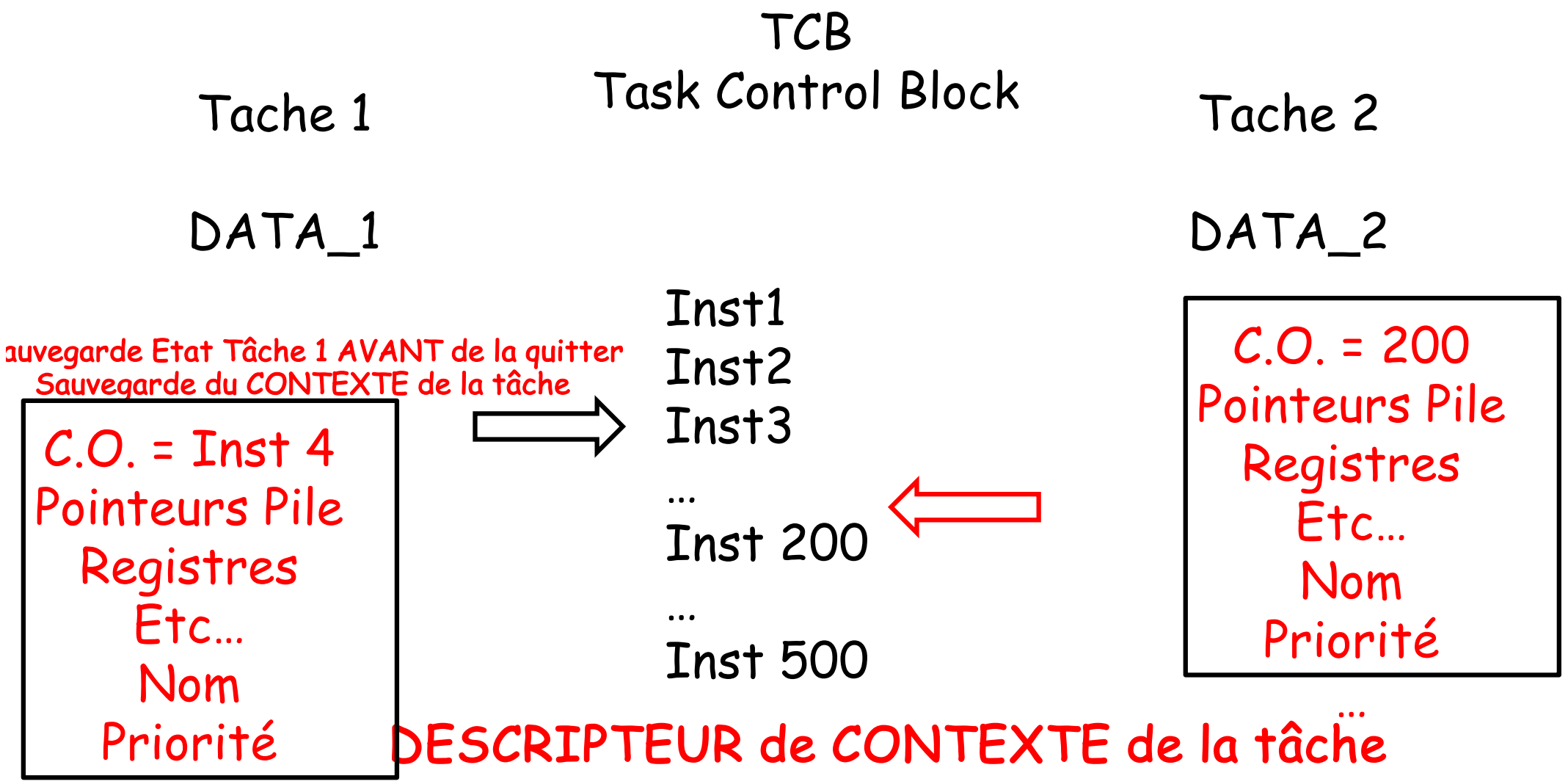
le code est chargé

-soit autant de fois que de tâches,

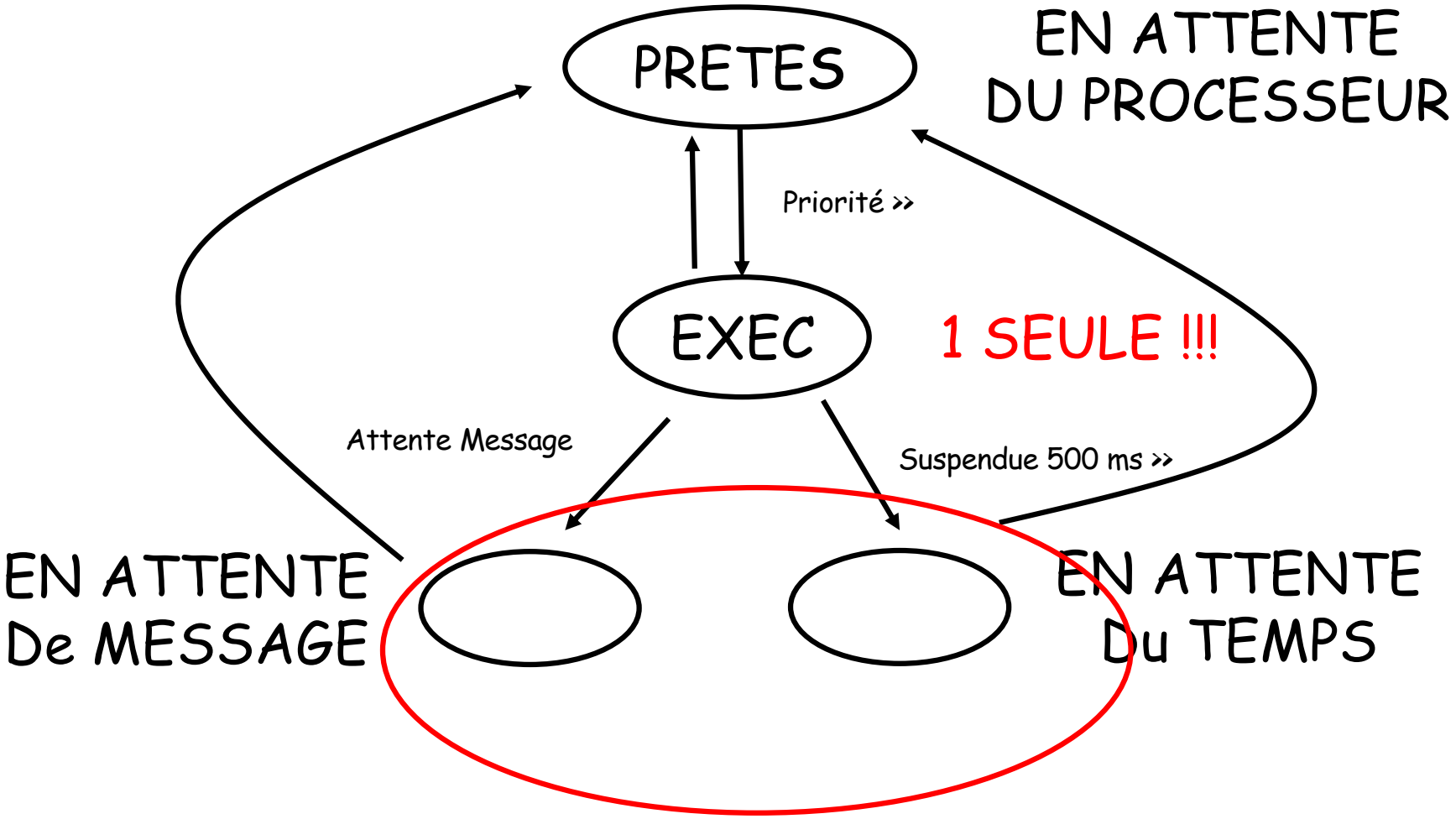
-soit une s[illegible]e code est "partagé" mais pas les données)

**Remarque: l'exécution d'une tâche est aussi parfois nommée "processus"**

# Live ….


TCB
Task Control Block
Tache 1
DATA_1
Tache 2
DATA_2
Inst1
Inst2
Inst3
…
Inst 200
…
Inst 500
auvegarde Etat Tâche 1 AVANT de la quitter
Sauvegarde du CONTEXTE de la tâche
C.O. = Inst 4
Pointeurs Pile
Registres
Etc…
Nom
Priorité
C.O. = 200
Pointeurs Pile
Registres
Etc…
Nom
Priorité
…
DESCRIPTEUR de CONTEXTE de la tâche

# Live ....


PRETES
EN ATTENTE
DU PROCESSEUR
Priorité >>
EXEC
1 SEULE !!!
Attente Message
Suspendue 500 ms >>
EN ATTENTE
De MESSAGE
EN ATTENTE
Du TEMPS

# Initiation au Langage Temps-Réel

## La notion de MONITEUR

Le Moniteur est **l'ARBITRE** qui va gérer l'Unité Centrale.
C'est un ensemble de procédures.

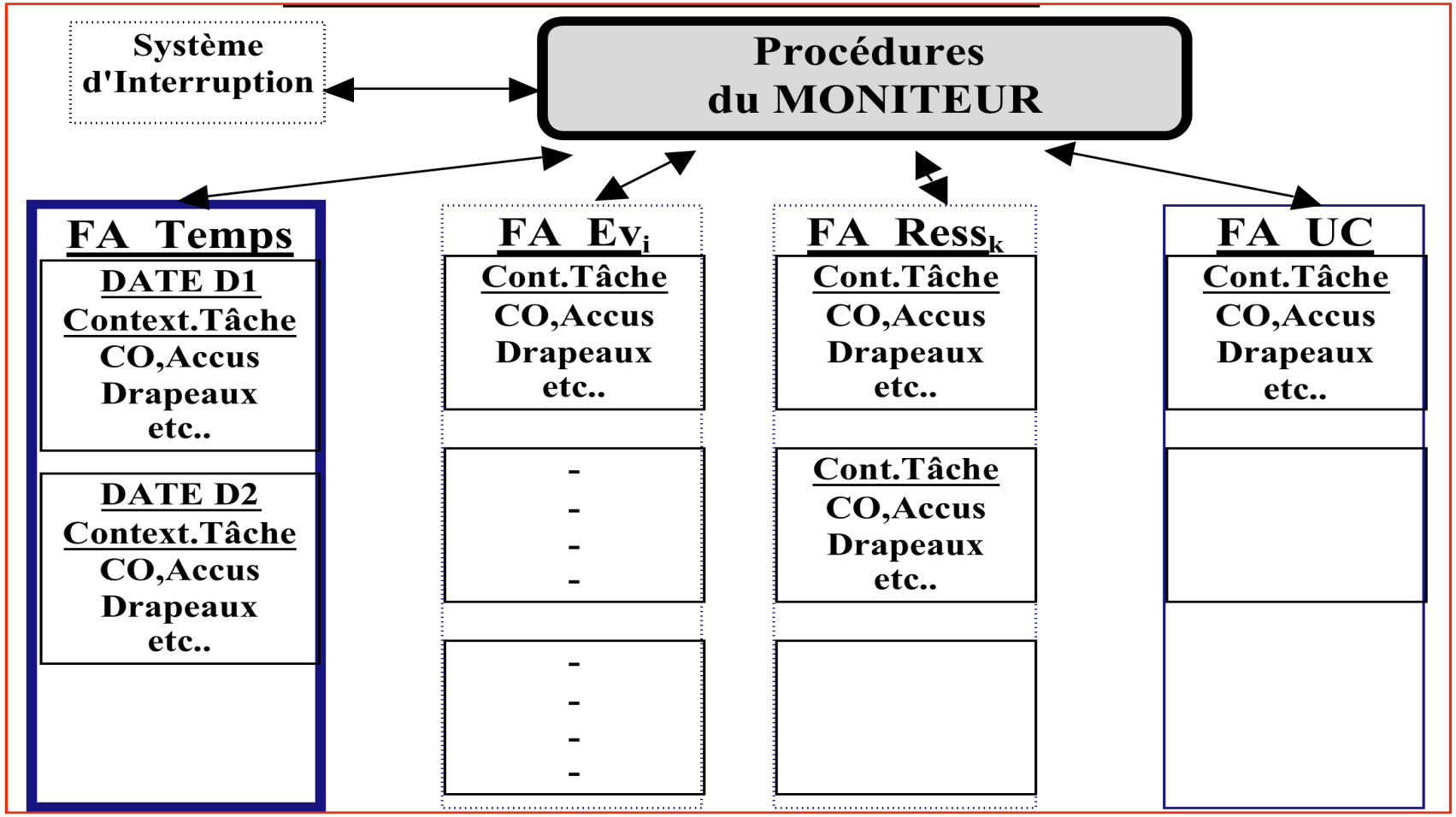
- **il gère les niveaux d'interruptions :** initialisation, forçage d'état
- **il gère le temps:** demande d'attente de dates, compteur de temps
- **il lance et arrête les tâches selon les besoins**
- **il gère des évènements et des ressources** (voir plus loin)

Il fait tout ceci en fonction des demandes de l'utilisateur exprimées sous forme des **Instructions Temps Réel"** contenues dans des tâches:

ex: -Demande de changement d'état des interruptions
-Demande de lancement de tâche

# Initiation au Langage Temps-Réel

***Pour cela il a besoin de Files d'attente***




Les **Files d'Attente** contiennent
les **Contextes des tâches** attendant d'être éxécutées (UC *mono-processeur*)

andreu@lirmm.fr

# Initiation au Langage Temps-Réel

## LES TACHES

Il y a 3 types de tâches dans un système TR:

**initiale - immédiates - différées**

## 1.1 La Tâche INITIALE

### 1.1.1 Définition

La TACHE INITIALE est la première tâche exécutée par le système au moment où il démarre.

La **tâche initiale** permet à l'Utilisateur d'initialiser les systèmes

1 le système physique:

*Forcer des états ou des commandes au départ: fermer vanne, etc.*

2 le système "Logiciel d'Application":

*Forcer les Etats du système d'interruption (Armer, etc.)*

*Démarrer des tâches contrôlant l' Application :*

*les tâches décrivant le contrôle (contr.séq. ou régulation)*

# Initiation au Langage Temps-Réel

**Déclaration**

```
TACHE INTIALE
    Début
        CORPS de la TACHE (Suite des instructions)
        les instructions peuvent être "classiques" ou "TR"
    Fin
```

# Les Tâches IMMEDIATES

## Définition

Les Tâches Immédiates sont les tâches déclenchées par Appel d'interruption.

Elles obéissent aux lois des niveaux de priorité des interruptions (cf. cours précédent)

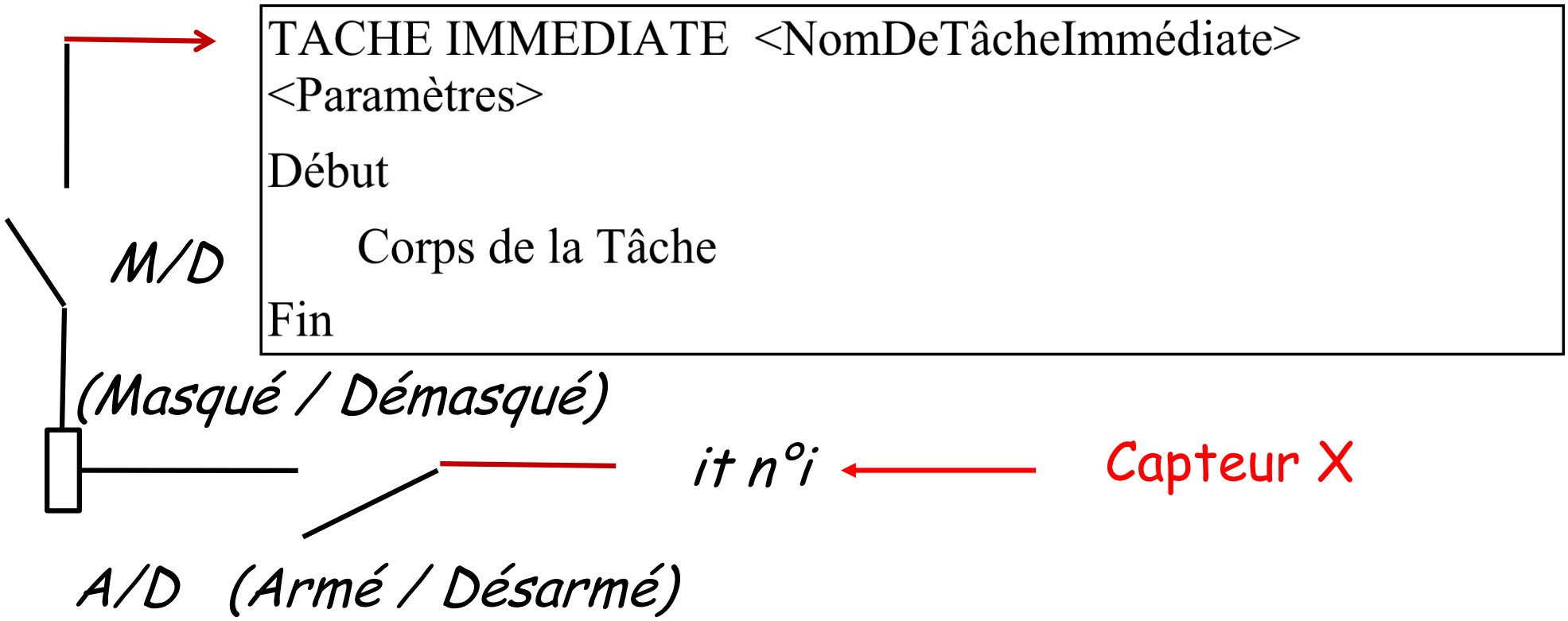
Dans ce qui suit, on supposera l'ordre de priorité décroissant :
le - prioritaire= 0 puis 1, 2, 3, x, N= le + prioritaire
N+1 étant le nombre de niveaux de priorités disponible

A tout instant, la tâche immédiate exécutée est la plus prioritaire de celles qui demandent à être exécutée

**Elles sont donc exécutées immédiatement (d'où leur nom)**
**sauf si elles ne sont pas les plus prioritaires au moment où elles sont appelées**

# Initiation au Langage Temps-Réel

**Déclaration**



## Instructions de Gestion des Tâches Immédiates

ATTACHE <NomDeTâcheImmédiate>,<NuméroDeNiveau>

exemple: **ATTACHE TILITVITESSE , 2**

Au moment où cette instruction est exécutée, le Moniteur associe le niveau d'appel d'interruption (2 de l'exemple) à la Tâche Immédiate (TILITVITESSE de l'exemple).

*Remarque*
*L'utilisateur n'a pas à gérer lui-même les mots de commande (ex: le RC du PIO)*

# Initiation au Langage Temps-Réel

DETACHE <NuméroDeNiveau>

A l'exécution de cette instruction, le Moniteur dissocie le niveau d'appel de toute tâche immédiate.
Cette instruction doit précéder l'association du même niveau à une nouvelle tâche immédiate. (*Mesure de sécurité*)

FORCE <NuméroDeNiveau>  <Etat>

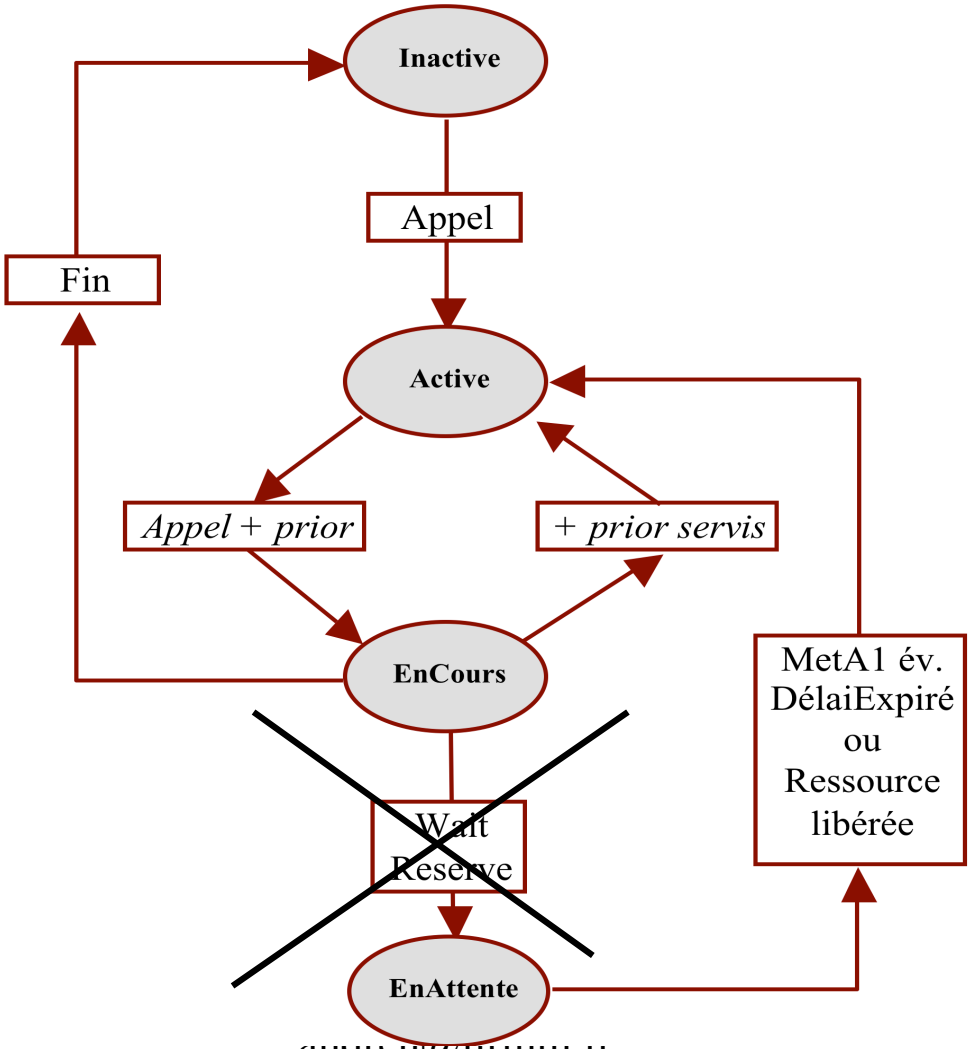
A l'exécution de cette instruction, le Moniteur force le niveau d'appel dans **"état"**

**Etat = {Armé/Désarmé/Masqué/Démasqué}**

ex: FORCE 2 ARME,  FORCE 4 DESA,  FORCE 7 MASQ, FORCE 1 DEMA

# Initiation au Langage Temps-Réel

**Etats d'une tâche Immédiate**



# Initiation au Langage Temps-Réel

**Précaution:**
**Une Tâche immédiate doit être courte car elle mobilise l'UC à son niveau de priorité.**
**Elle ne doit pas comporter de calcul long ou d'attente (ex: lecture ou écriture sur disque)**

Il existe donc une autre catégorie de Tâches qui va être exécutée au plus bas niveau de priorité (niveau 0): les Tâches **DIFFEREE**S.
Ces dernières seront en général plus longues et pourront comporter des attentes.

**Utilisation:**
Les Tâches Immédiates servent à **déclencher des programmes sur des appels** (bouton, signal de synchro périphérique, etc.).

Elles peuvent également "**rendre exécutables**" d'autres tâches moins prioritaires, les Tâches Différées (Cf. sous-chapitre suivant).

**Remarque:**

-Une Tâche immédiate n'est exécutée **immédiatement** que si elle est la plus prioritaire.

Si, au même instant, pendant l'exécution d'une Tâche Immédiate TIn une autre tâche immédiate TIp plus prioritaire demande à être exécutée Tin est interrompue et attend la fin de TIp.

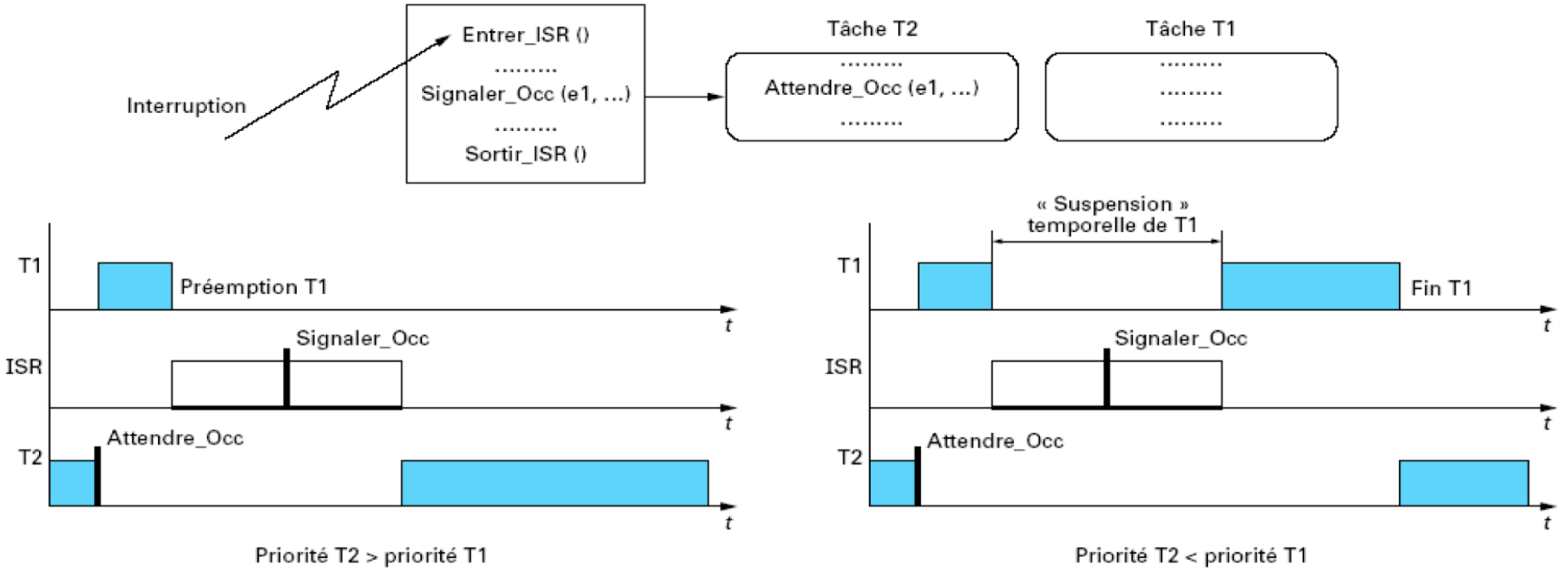
-Si un système comporte beaucoup de Tâches Immédiates et que celles-ci sont appelées en même temps, les conflits de priorité peuvent donner des **attentes des tâches immédiates prohibitives** par rapport aux contraintes devant être respectées.

**On évaluera alors avec précision les parties qui doivent être exécutées vite** et les autres, et **on découpera les tâches de façon à ne faire figurer dans les Tâches Immédiates que les parties les plus urgentes**.

# Initiation au Temps-Réel

Remarque : dans les OS TR les tâches immédiates sont en fait des routines de traitement d'interruption appelées en général : ISR (Interrupt Service Routine).

Cette routine n'a pas le « statut » de tâche car elle n'a pas de contexte reconnu de l'exécutif.




Tout traitant d'interruption (toute tâche immédiate en LTR) est toujours plus prioritaire que toute tâche (tâche différée en LTR).

# Initiation au Langage Temps-Réel

## Les timers

**Timer <NuméroTimer> <Ordre>,<ValeurOrdre>**

Cette instruction permet de se servir de compteurs de temps qui déclenchent des interruptions quand ils passent à 0. On les initialise à une valeur, et on leur donne l'ordre de décompter.

On peut aussi fixer la cadence de l'horloge qui les pilote, comme un multiple d'une base de temps. Cette base de temps peut être l'horloge calculateur ou une horloge externe.

**NuméroTimer**: identifie le Compteur utilisé parmi un certain nombre

**Ordre**: définit l'ordre envoyé au Timer

**ValeurOrdre**: définit le(s) paramètre(s) de l'ordre

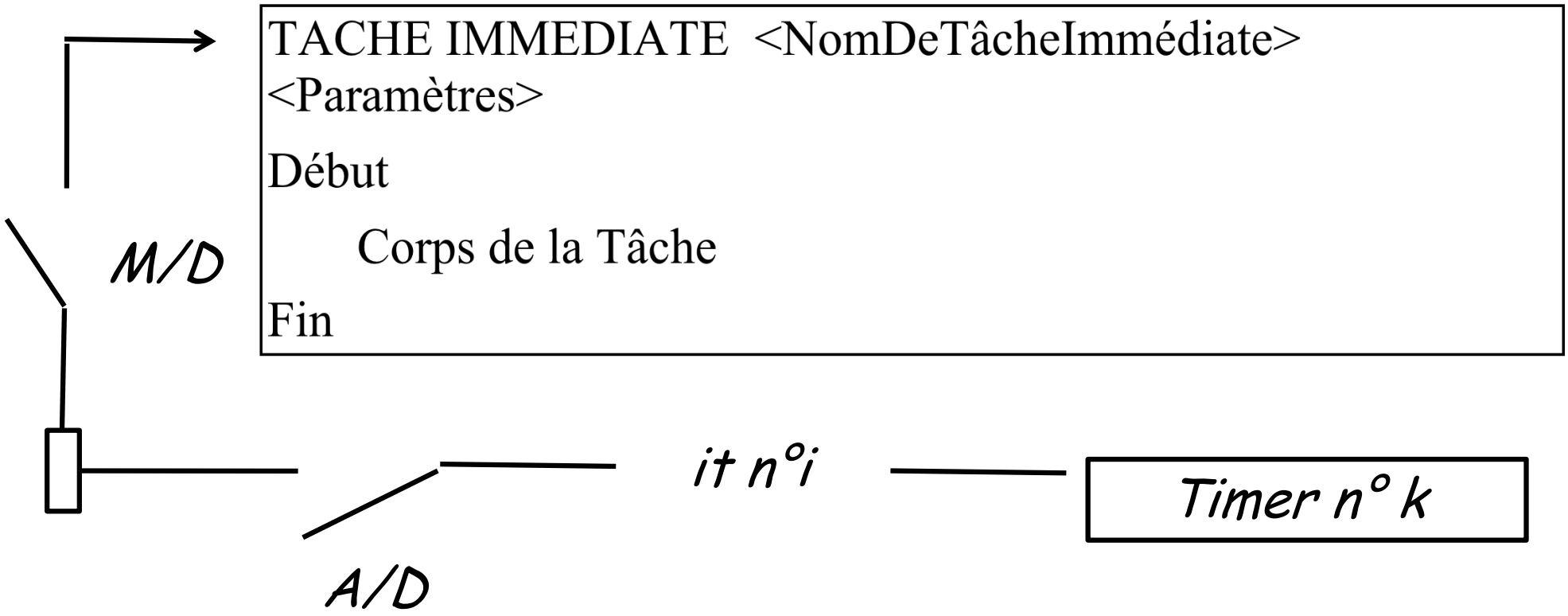
# Initiation au Langage Temps-Réel

## Détail des ordres de Contrôle des Timers

| Ordre,Paramètre(s) | ➔ Effet |
|---|---|
| **Charge , NomDeVariable** | ➔ *met la valeur de la variable dans le Timer et lance le décomptage* |
| **Attache , NuméroNiveau** | ➔ *définit le niveau d'appel déclenché par le Timer* |
| **Force , Etat** | ➔ *force l'état du niveau d'interruption du Timer (Arme, Desa, Masq, Dema).* |
| **Exemples :** | **Timer 1 Charge VALCOMPT**<br>**Timer 1 Charge 500**<br>**Timer3 Attache 5** |

# Initiation au Langage Temps-Réel

**Déclaration**



# Initiation au Langage Temps-Réel

## Les E/S

### Utilitaires d'un Moniteur TR

Le moniteur dispose des procédures suivantes permettant d'entrer ou de sortir des Mots et des Bits, et de contrôler des compteurs de temps.

*Si ces éléments n'existent pas dans un compilateur de système TR ils existent souvent en Macro-assembleur ou en Procédures C que l'on peut LIER (au moyen du LINKER) avec les procédures du Langage TR.*

**Entre <NomDePortEntrée> <NomDeVariable>**

Cette procédure lit le Port d'Entrée **NomDePort** et met la valeur lue dans la variable NomDeVariable. L'adaptation au format (bit ou mot) est automatique dans la procédure.
Ex: ENTRE VE1 NomVar

**Sort <NomDePortSortie> <NomDeVariable>**

Cette procédure affiche la valeur lue dans la variable NomDeVariable sur le Port de sortie **NomDePort** et. L'adaptation au format (bit ou mot) est automatique dans la procédure.
Ex: SORT VS3 NomVar

# Les Tâches DIFFEREES

## Définition

**Les Tâches Différées sont les Tâches que l'UC exécute quand il n'y a plus de Tâche Immédiate à exécuter.**

Leur exécution est demandée avec une priorité logicielle (sans rapport avec les niveaux de priorité d'interruption matérielle).

Leur exécution est différée (d'où leur nom) tant que l'UC exécute des Tâches Immédiates, ou d'autres tâches Différées plus prioritaires.

File d'Attente des TD (FATD)

- Les Tâches Différées en attente exécution sont rangées dans des Files d'Attente. Il y a une File d'Attente par niveau de priorité logicielle.

$$FATD = \{\mathbf{FATD_i}\}$$

# Initiation au Langage Temps-Réel

Principe de fonctionnement

- **Quand aucun niveau d'interruption n'est demandé, le Moniteur lit les files d'attente $FATD_i$,** en commençant par la plus prioritaire.

-**Dés qu'il trouve le contexte d'une tâche éligible** dans une de ces **$FATD_i$,** il la charge en UC, et **elle commence à exécuter** (l'exploration des Files d'Attente s'arrête puisque l'UC est occupée à exécuter la tâche élue).

*Vocabulaire*:

On dira qu'une T.D demandée mais non exécutée est "**éligible**".

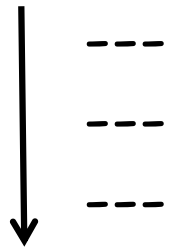
Quand elle aura été choisie par le Moniteur (parce que la plus prioritaire) pour être exécutée on dira qu'elle est "**élue**".

On appelle aussi ces tâches les "**tâches de fond**"

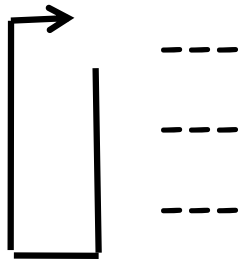
# Initiation au Langage Temps-Réel

## Déclaration

```
TACHE DIFFEREE <NomDeTâcheDifférée>,<ListeDeParamètres>
Début
        Corps de la Tâche
Fin
```



Sans boucle



Avec boucle

## Instructions de Gestion

Exécution d'une Tâche Différée est demandée par une instruction

**REVEILLE <NomDeTâcheDifférée> <Priorité> <ListeDeParamètres>**

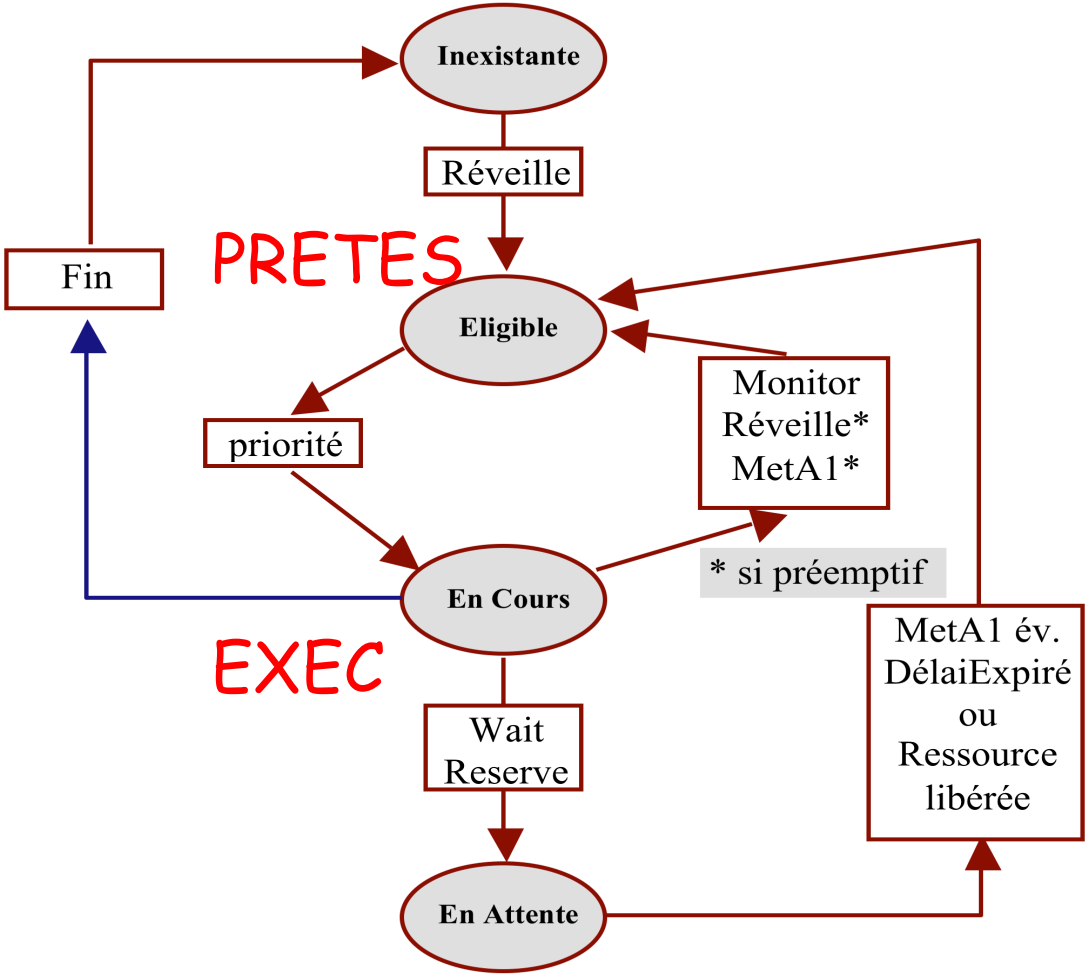
Quand cette instruction est rencontrée dans la tâche courante, le Moniteur:
-1- crée la tâche **NomDeTâcheDifférée** (fabrique son contexte) avec la valeur des paramètres au moment de la création.
-2- rend la tâche éligible en mettant son contexte dans la $\mathbf{FATD_{Priorité}}$

Utilisation:
Cette instruction permet de réveiller une tâche différée à partir d'une autre tâche de nature quelconque.

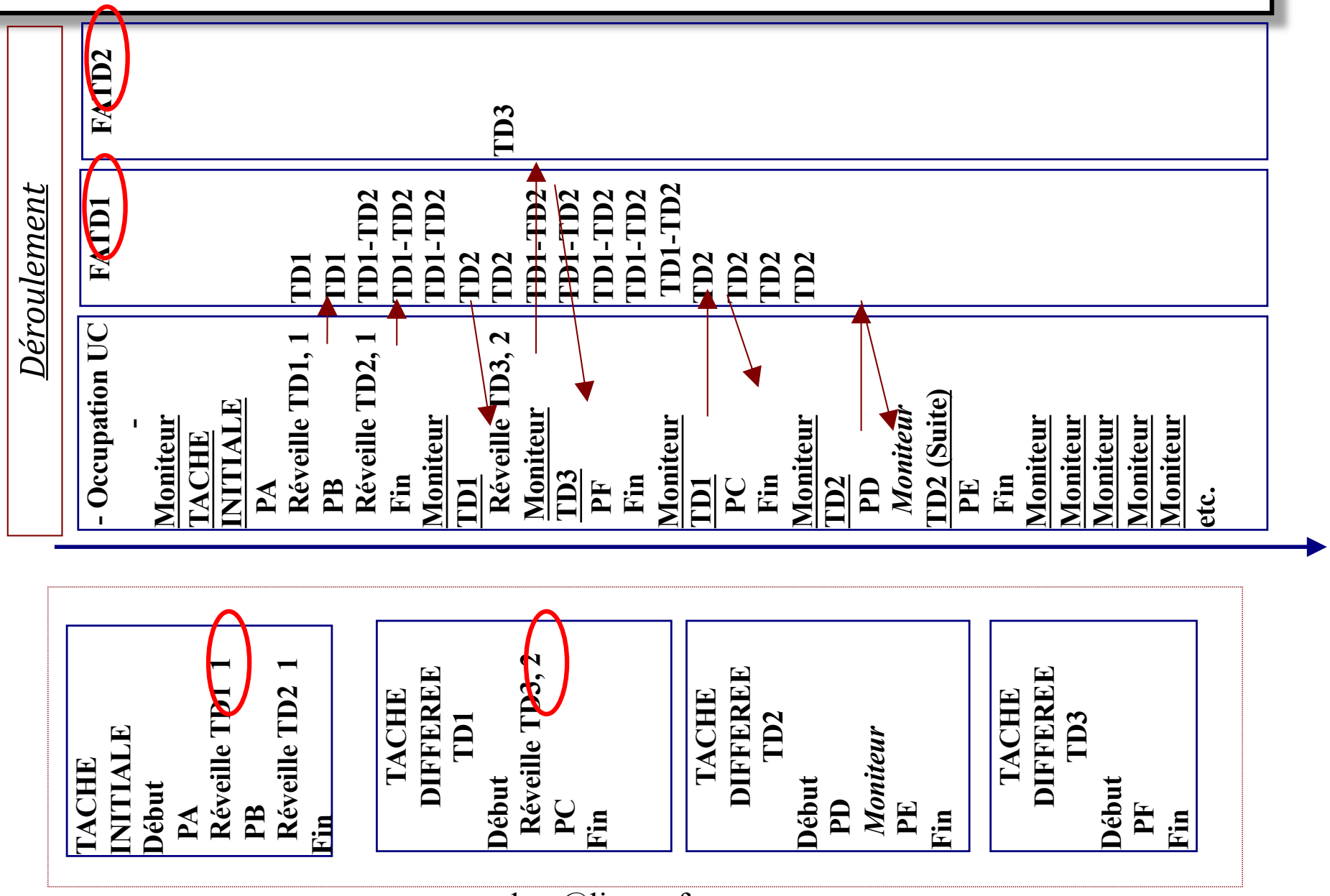
Peut-être créée (REVEILLE) par la tâche initiale…
mais si elle ne doit pas s'exécuter de suite débuter par ex.
par l'attente d'un EVT.

# Initiation au Langage Temps-Réel

**Etats d'une tâche Différée**



# Initiation au Langage Temps-Réel



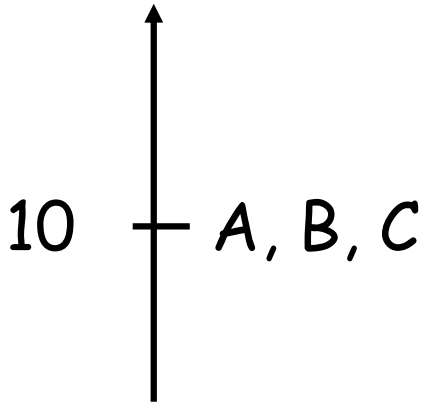
# Initiation au Langage Temps-Réel

Remarque

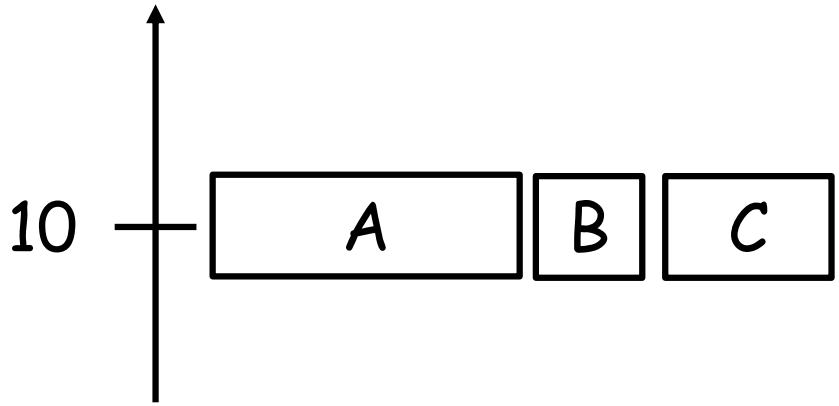
Certains OS n'admettent qu'une seule TD par niveau de priorité
(ex : microC-OS)

Si OS admet plusieurs tâches sur un même niveau de priorité
(ex : QNX), différentes stratégies possibles :

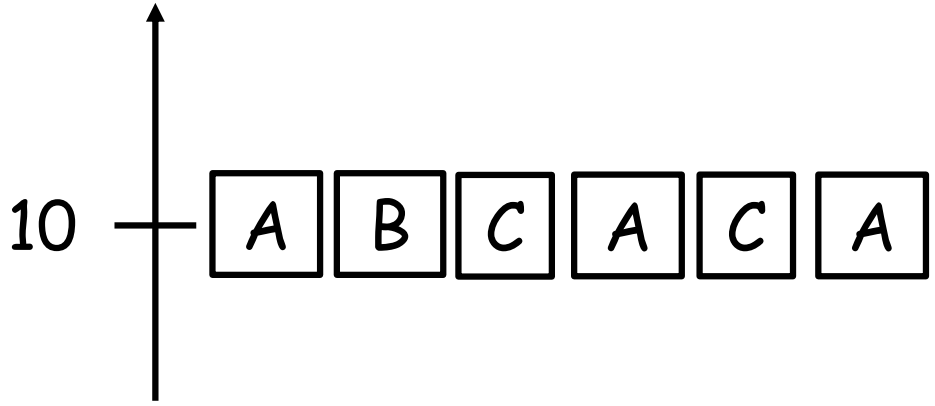
- FIFO
- ROUND-ROBIN (Tourniquet, quantum de temps)




FIFO




RR Round Robin (Tourniquet)
Durée = quantum



# Initiation au Langage Temps-Réel

**Notion de Moniteur "préemptif" ou "non-préemptif".**

**Un moniteur sera dit "non-préemptif", si une tâche différée courante se poursuit tant qu'elle est exécutable**, même si elle, ou des événements extérieurs, réveillent des tâches différées plus prioritaires.

**Un moniteur sera dit préemptif si une tâche différée courante peut être suspendue parce qu'une autre tâche différée de priorité supérieure vient d'être réveillée, et qu'elle est lancée immédiatement.**

Dans un **moniteur préemptif** le fonctionnement de la priorité logicielle rejoint celui de la priorité matérielle:

**En effet, à tout instant c'est la tâche la plus prioritaire qui est exécutée.**

***Dans ce qui suit, on supposera le moniteur LTR Préemptif.***

# Initiation au Langage Temps-Réel

**Identification des tâches en LTR**
Chaque Tâche a un numéro (ex TD1-001).

**C'est ce numéro qui identifie la tâche et non son nom**, puisqu'une tâche (TD1 par exemple) peut être lancée plusieurs fois

Sur un OS, un processus se voit attribué un
TID (Task Identifier) ou PID (Processus Identifier) ou …

Résolution de nom … sur certains OS on peut associer un nom symbolique à une tâche et retrouver son PID à partir de son nom (ex : QNX)

# Initiation au Langage Temps-Réel

**Différence entre "Réveil" et "Lancement" d'une tâche**

Réveiller une tâche c'est la rendre éligible,
c'est à dire mettre son contexte dans une FATD.

Lancer une tâche c'est l'élire ou encore l'exécuter,
c'est à dire charger son contexte en UC.

Une tâche a un **DESCRIPTEUR DE CONTEXTE** contenant :
(parfois appelé TCB, Task Control Block)

- statut, identifiant
- valeur des registres UC (ex: compteur ordinal)
- pointeur sur pile
- données locales
- etc.

Une **commutation de tâches** -> une commutation de contexte (sauvegarde, chargement)
Rem : latence de commutation = critère important de choix d'un OS

# 1 Les Evenements

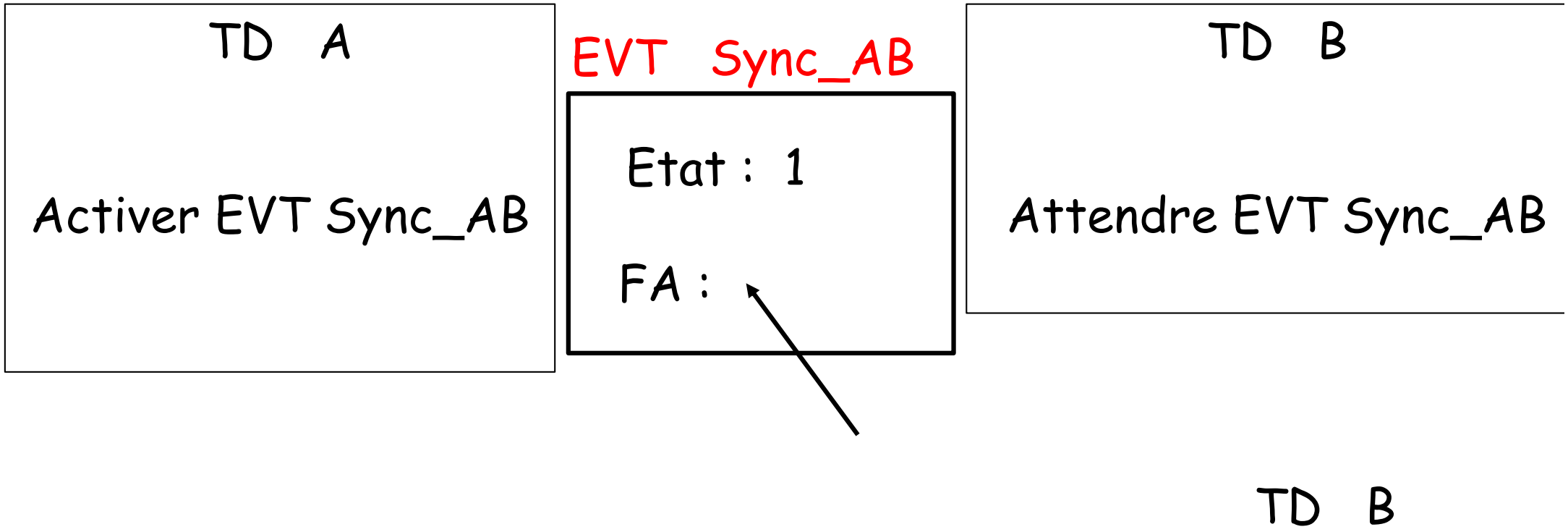
## Définition

Un événement est une entité booléenne (vraie/arrivée ou fausse/attendue) que peuvent attendre ou positionner des tâches.

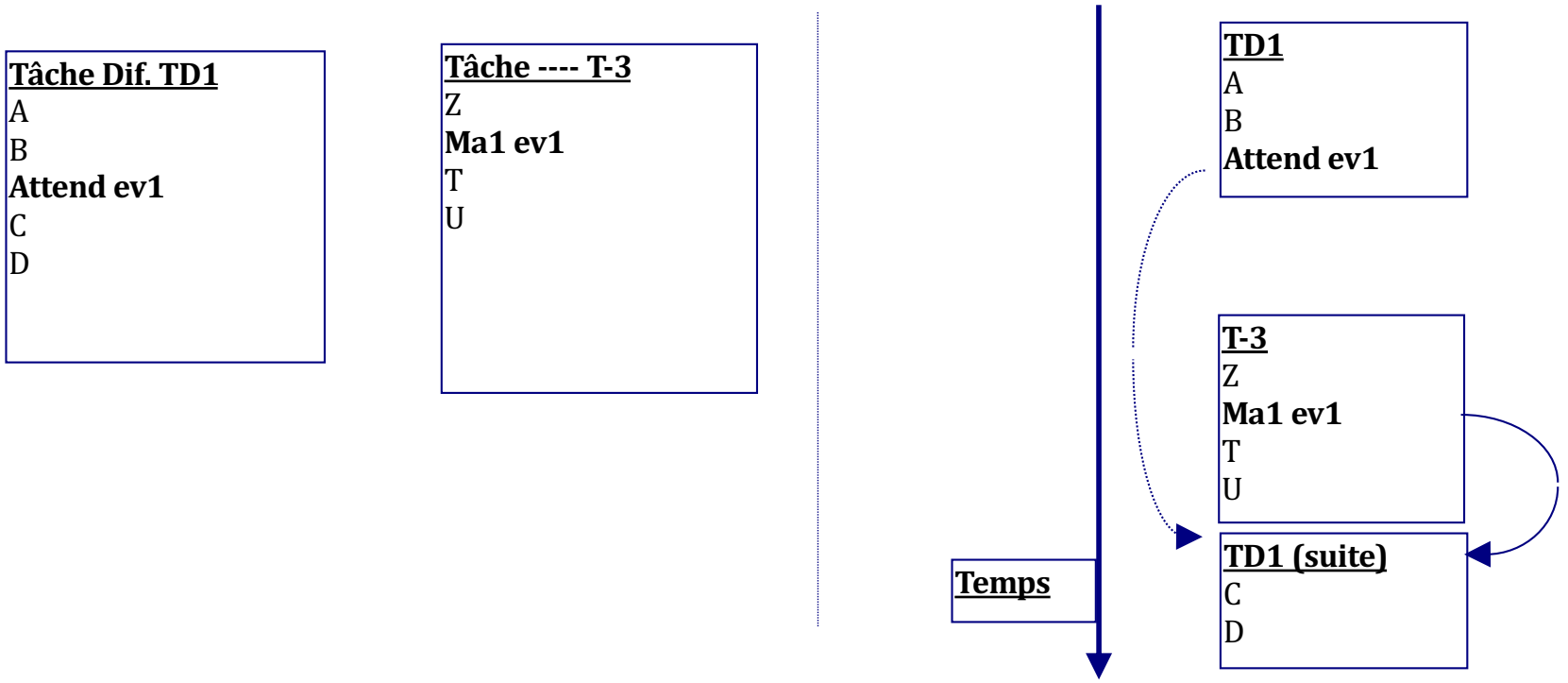
Principe général de fonctionnement

Les événements servent à synchroniser entre elles les tâches.

Une tâche peut attendre un événement positionné par une autre tâche.


TD A
Activer EVT Sync_AB
EVT Sync_AB
Etat : 1
FA :
TD B
Attendre EVT Sync_AB
TD B

# Initiation au Langage Temps-Réel




Puisque certaines tâches attendent un événement, à chaque événement est associée une File d'Attente de événement:

**FAEV(ev1).**

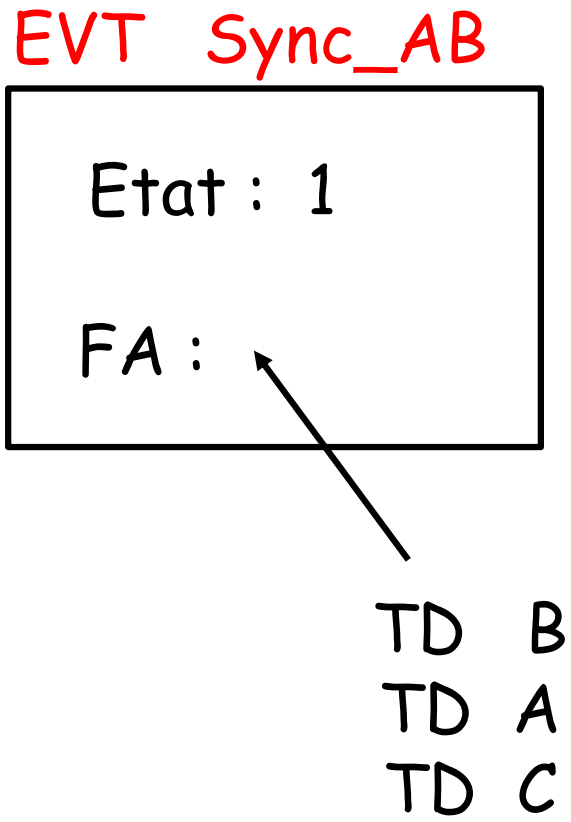
Elle contient le contexte de toutes les tâches qui attendent cet événement, tant que cet événement n'est pas arrivé.

-Quand événement arrive, les tâches de FAEV(ev1) sont rendues éligibles

# Initiation au Langage Temps-Réel

Cela dépend de l'OS :
- Soit toutes les tâches en attente de EVT sont décrochées (LTR)
- Soit seulement la 1ere de la liste

Selon l'OS il est possible de gérer l'ordre de la liste associée à une entité de synchronisation selon :
- Les priorités des tâches,
- Une priorité de classement (normal, prioritaire)
- Une priorité de message (si synchro basée sur basée sur communication, cf. mailbox)


EVT Sync_AB
Etat : 1
FA :
TD B
TD A
TD C

## Déclaration

EVENEMENTS: <Liste d'événements>

Exemple: **EVENEMENTS: VitesseAtteinte, Détection, RetourSynchro**
Notons que des tâches peuvent attendre des dates. Le temps est donc un événement implicite, qui a sa propre file d'attente.

# Instructions de gestion des événements

ATTEND <Evénement>

Quand cette instruction est rencontrée, le moniteur exécute la procédure suivante dépendant de l'état de l'événement :

- si l'événement **est déjà arrivé,**
  - ➔ la tâche courante continue
- si l'événement **n'est pas arrivé**
  - ➔ la tâche courante est suspendue. Elle est mise en FAEV (événement)
  - ➔ le moniteur élit (lance) une autre tâche

# Initiation au Langage Temps-Réel

## MA_0 <Evénement>

Quand cette instruction est rencontrée, le moniteur met l'événement à l'état 0 ou NON arrivé. La tâche courante continue.

## MA_1 <Evénement>

Quand cette instruction est rencontrée, le moniteur met l'événement à l'état 1 ou arrivé. Ce qui se passe ensuite dépend du contenu de la FAEV (Evénement):

- si FAEV (Evénement) est **vide**
  ➔ la tâche courante continue

- si FAEV (Evénement) est **NON vide**
  ➔ le moniteur réveille les tâches de FAEV (événement) et les fait passer dans la FATD de leur priorité au moment de la suspension.

  - *Si le moniteur n'est pas préemptif*, la tâche courante continue
  - *Si le moniteur est préemptif*, et si la tâche courante est moins prioritaire qu'une tâche qui vient d'être réveillée, le moniteur suspend la tâche courante et élit la tâche plus prioritaire. Sinon la tâche courante continue.

# Initiation au Langage Temps-Réel

## Un événement particulier: le temps

### Principe

**-Les Tâches peuvent attendre l'écoulement d'un délai.**
Cela se produit quand une tâche contient l'instruction

| | **ATTENDRE** | **DELAI** |
|---|---|---|
| ex: | ATTENDRE | 245 UT (Unité de temps) |

### Fonctionnement

Soit Dcourante la date à laquelle l'instruction est exécutée
**Quand cette instruction est rencontrée, le moniteur suspend la tâche courante, et enregistre le fait que la tâche doit être réélue à Dcourante+DELAI**

Pour cela, le Moniteur possède sa propre file d'attente du temps FATP.
Dans FATP on trouve des couples **(Date-Contexte de tâche)**

# Initiation au Langage Temps-Réel

Le Moniteur vient périodiquement scruter FATP **sur appel d'un Timer qu'il gère**. De plus, au moment où il range une tâche en attente, il intercale le couple (date-contexte) dans les autres, de façon que les demandes soient rangées dans l'ordre d'exécution.
Exemple

| **FATP** | |
|---|---|
| **87 - TDA** | **x** (n° de la tâche) |
| **340- TDV** | **y** |
| **560-TDRE** | **z** |
| **679- TDRU** | **t** |

Ainsi, chaque fois qu'il vient lire cette file pour savoir s'il y a quelque chose à lancer, il suffit de lire le haut de la File.

UT est donné par le constructeur (et parfois réglable).

# Initiation au Temps-Réel

Attention à la granularité temporelle (pas de temps) selon laquelle
le temps est géré !

Si pas de temps très court … le moniteur passe son temps à gérer la file d'attente
Si pas de temps trop grand … la gestion du temps est imprécise
(et manque de réactivité)

Trouver un compromis …. imposé par l'adéquation à la dynamique
du système contrôlé.

RAPPEL : temps-réel ne veut pas dire rapide … mais adéquat à la dynamique.
(ex : avion et ascenseur … pas la même dynamique)

## Tâche (différée) « périodique »

Remarque : il faut éviter de réaliser une tâche périodique en utilisant un service tel que «Attendre_Délai (durée) » qui suspend une tâche pour une durée donnée. En effet, la période de la tâche est égale au minimum à la valeur du paramètre durée augmentée du temps d'exécution du corps de la tâche, cette dernière valeur étant variable par la préemption possible de la tâche. Il s'ensuit une forte incertitude sur la valeur réelle de la période, qui de surcroît risque d'être variable.

Privilégier une activation par Timer.

Il est est de même pour contrôler des actions de durée précise.

# Initiation au Temps-Réel

Evénement particulier le temps …. le timeout

TimeOut = principe du chien de garde pour demander qu'une opération bloquante ne dure qu'un temps maximal.

Le temps est souvent (voire toujours) associé aux services d'attente offerts par l'exécutif : attente d'une communication, d'une synchronisation, d'une autorisation d'entrée en exclusion mutuelle… il prend alors la forme d'un « chien de garde » (time out) optionnel.

Une attente infinie est possible (paramètre fixé à 0) quand il est impératif que l'événement se soit produit pour poursuivre la tâche.

# Initiation au Temps-Réel

Exercice simple

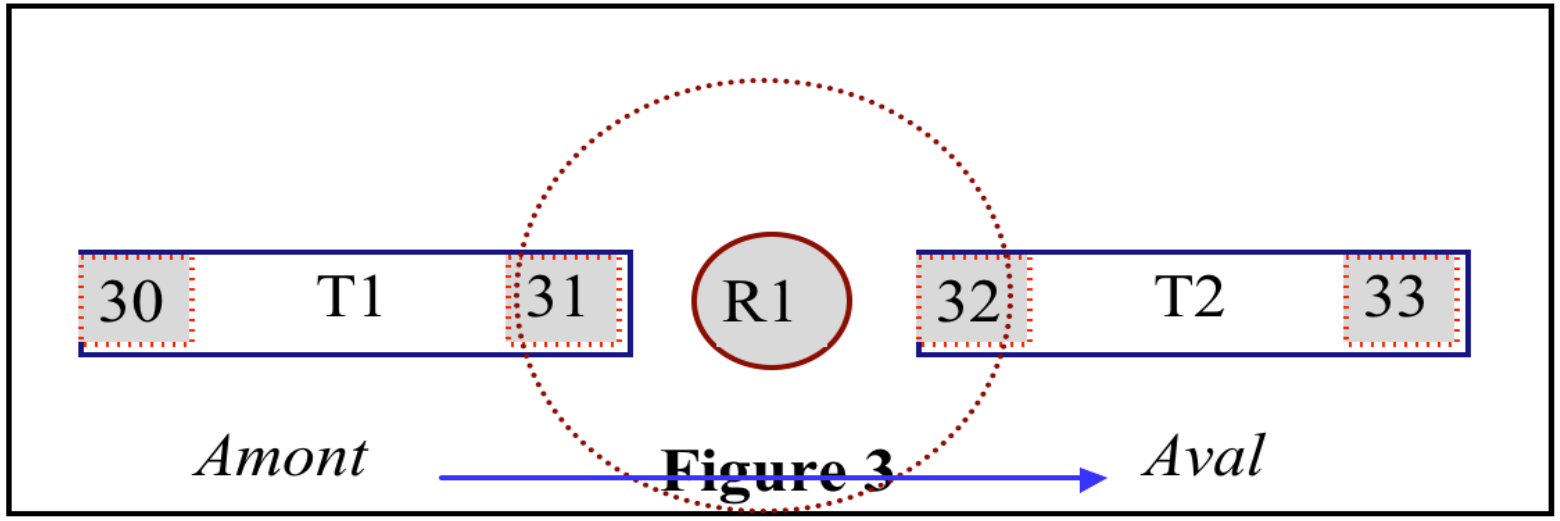


Figure 3

Pièce déposée en 30 -> activation tapis T1 -> Arrêt T1 qd pièce en 31
Pièce en 31 -> transport robot sur position 32 (tapis T2)
Pièce déposée en 32 -> activation tapis T2 -> Arrêt T2 qd pièce en 33
-> retour robot position 31

+

Ne pas réactiver T1 tant que pièce en 31

+

Pas de capteur 33 -> rotation de T2 exactement de 7,65 sec.

Et si T2 est déjà occupé (i.e. pièce en 32) …. Comment le robot peut-il savoir ?

ressources

# LES RESSOURCES

## Introduction

### Le concept de ressource dans la vie courante

La notion de ressource n'est pas une notion informatique, au départ.
**C'est un concept de la vie courante:**

| | | |
|---|---|---|
| **L'argent** | pour un foyer | est une ressource |
| **Le minerai** | pour un pays | est une ressource |

**le concept de ressource est caractérisé par deux propriétés:**

- **Limitation**: - la quantité de ressource est limitée
  *l'argent d'une famille (le salaire)*
  *les mines*
- **Utilisation partagée** -plusieurs "utilisateurs" veulent utiliser la ressource
  *les membres de la famille*
  *les compagnies minières*

# Exemples de ressource techniques

## Les ressources en Physique (exemple de la Productique)

Dans un atelier de production les ressources sont:

- -les machines-outils
- -les chariots transporteurs
- -les outils
- -les produits bruts

## Les ressources en Informatique

Dans un système informatique les ressources sont:

- -l'Unité Centrale - partagée par les Tâches, en multitâche LTR.
- -la mémoire vive disponible pour un programme
- -la taille disque disponible pour un programme
- -les périphériques de communication (lignes, imprimantes), etc.

**Les instructions de gestion de ressource vont permettre de gérer ce concept**

# Initiation au Langage Temps-Réel

## Les questions fondamentales

Au moment où un programme va utiliser une ressource (par exemple une imprimante) les deux questions fondamentales qui vont se poser sont:

**-est-elle libre ou pas ?**

(en effet plusieurs utilisateurs-programme sont susceptibles de l'utiliser)

**-si elle n'est pas libre, qu'est-ce qu'on fait ?**

Pour gérer ces problèmes, une ressource informatique sera définie par deux éléments:

-la capacité maximale d'utilisateurs

-le sémaphore de protection avec une file d'attente

**Question annexe:**

**- Quel est le lien entre les ressources physiques, et les ressources informatiques ?**

*Les ressources* ***informatiques*** *vont être* ***les concepts de programmation qui permettent d'écrire et d'exécuter les programmes***

*qui gèrent les ressources* ***physiques***

# Initiation au Langage Temps-Réel

## Définition formelle d'une ressource LTR

**Une ressource LTR indicée i (*pour la distinguer parmi plusieurs*) est définie par**

1 **un nombre maximum d'utilisateurs**
**NMAXi** **Constant, défini au départ**
ex: nb d'imprimantes: 2, nb de chariots: 5

**Si NMAX = 1 la ressource est dite "CRITIQUE"**

2 **un nombre d'utilisateurs courant**
**NUCi** **variable dans le temps**
il représente à chaque instant le **nombre d'utilisateurs** en train d'utiliser la ressource

**Si NUCi < NMAXi** **la ressource est dite LIBRE**
**Si NUCi >= NMAXi** **la ressource est dite OCCUPEE**

3 **une File d'attente**
**FARi** **variable dans le temps**
Elle contient le contexte des tâches qui veulent utiliser la ressource mais qui ne le peuvent pas pour le moment parce qu'elle n'est pas libre

# Utilisation des Ressources en LTR

## Introduction

Les Ressources sont utilisées dans les programmes Temps Réel, grâce à

une **directive de déclaration**
et
trois **instructions de gestion**: **"Réserve", "Libère", "Teste"**.

### Déclaration de ressource

Pour utiliser une ressource, il faut d'abord la déclarer par une directive de compilation

**Ressource <NomDeRessource> <NMAX>**

**ex: Ressource Imprimante 1**
**Ressource Chariot 3**

## Instructions de gestion de ressources

### L'instruction "RESERVE"

Format de l'instruction

**RESERVE <NomDeRessource>**

### Principe général

L'instruction **Réserve** sert à savoir si on peut utiliser ou non une ressource.
On la met au début de la zone du programme qui utilise la ressource physique.

**Tâche TD1**
Début
*calculs*
-
**Reserve Impression**
*Instructions d'impression*
-
-
-

**Déroulement**

Cette instruction provoque un **branchement vers le moniteur**, pour gérer la ressource Ri.

**1 - Le moniteur incrémente NUCi** **NUCi + 1**

**2 -Deux cas sont alors possibles:**

**-Si NUCi <= NMAXi Alors c'est que la ressource est Libre**

Le moniteur revient dans la tâche en cours, et elle continue.
Elle rentre donc dans la partie qui utilise la ressource.

**-Si NUCi > NMAXi Alors c'est que la ressource est Occupée**

La Ressource est occupée, elle ne peut donc pas être utilisée par cette tâche.
Le moniteur suspend la Tâche, et met son contexte en File d'Attente FARi.
Il cherche une autre tâche éligible, et la lance.

Plus tard, la Tâche reprendra quand la ressource sera de nouveau libre...

# Initiation au Langage Temps-Réel

### L'instruction "LIBERE"

Format de l'instruction

**LIBERE <NomDeRessource>**

### Principe général

L'instruction Libère sert à indiquer que la tâche a fini d'utiliser la ressource.
On la met à la fin de la zone de programme utilisant la ressource

**Tâche TD1**
Début
*calculs*
-
***Reserve Impression***
*Instructions d'impression*
*Instructions d'impression*
**Libére Impression**
-

# Initiation au Langage Temps-Réel

**Déroulement**

**1 - Le moniteur décrémente NUCi** **NUCi - 1**

**2 -Deux cas sont alors possibles:**

**-Si NUCi < NMAXi** **Alors c'est que la ressource est Libre**

**Le moniteur revient dans la tâche en cours, et elle continue.**

**-Si NUCi >= NMAXi** **Alors c'est que la ressource est Occupée**
**Donc FARi contient au moins une tâche en attente de la ressource**
Or une utilisation vient de se libérer. Donc, **le moniteur prend la plus prioritaire des tâches de FARi (notée ici Tj), et la rend éligible**.
*Dés que Tj deviendra la plus prioritaire, elle sera lancée et rentrera dans la partie utilisant la ressource*
**Le moniteur revient dans la tâche en cours.**

# L'instruction "TESTE"

Format de l'instruction

| TESTE <NomDeRessource> <Paramètre1> <Paramètre2> |
|---|

## 1 .1.1 Principe général

Quand une instruction réserve est rencontrée, la tâche dans laquelle elle est exécutée est bloquée en attente de la ressource. Il existe des cas où ce blocage n'est pas admissible. Cela ne veut pas dire que la ressource doit être prise quand même, mais plutôt que la tâche va exécuter d'autres fonctions sans se bloquer en attente de la ressource (par exemple prévenir l'opérateur que la ressource est occupée).
L'instruction Teste sert à tester l'état d'une ressource sans que la tâche soit bloquée en attente de la ressource, si celle-ci est occupée. Quand elle s'exécute, il y branchement vers le moniteur, test de l'état de la ressource.
**Paramètre1 et Paramètre2** contiennent des informations sur l'état de la ressource:

| Paramétre1 | Signification | Paramétre2 contient |
|---|---|---|

# Initiation au Langage Temps-Réel

**Déroulement**

Quand l'instruction TESTE <NomRessource> est rencontrée, le moniteur teste l'état de la ressource:

**1er cas: la ressource est libre**

*alors*

-**param1 contient "0"**

-**param2 contient le nombre de places encore disponibles** avant que la ressource soit occupée.

*(il peut être testé par n'importe quelle tâche - c'est utile en optimisation)*

-**la tâche continue et la ressource est réservée pour la tâche** (équivaut à RESERVE)

**2éme cas: la ressource est occupée**

*alors*

-**param1 contient "1"**

-**param2 contient le nombre de tâches en file d'attente de la ressource**

*(il peut être testé par n'importe quelle tâche - c'est utile en optimisation)*

-**la tâche continue** mais **la ressource n'est pas réservée.**

ATTENTION: puisque dans les deux cas, la tâche n'est pas bloquée et continue, elle doit contrôler elle-même l'accès ou non à la ressource, en fonction de la valeur de param1

# Initiation au Langage Temps-Réel

Exercice

..... **Exemple**

**Présentation du parking**

On considère un parking de 45 places dont l'entrée est pilotée par ordinateur.
Elle comporte deux barrières BE et BS. La sortie du parking
Quand la voiture se présente devant BEE et BES, s'il y a une place de parking libre, la barrière BEE s'ouvre, sinon BES s'ouvre et la voiture ressort, sans être bloquée devant l'entrée.




DI1, DI2, DI3, DI4 sont des boucles magnétiques qui détectent la présence d'une voiture et lancent un signal d'interruption de niveau respectivement 1, 2, 3, 4.
Les barrières BEE et BES sont ouvertes ou fermées au moyen de deux procédures OUVRE (NomBarr.) et FERME (NomBarr.)
Ces procédures ne sont pas détaillées ici. Elles font intervenir d'autres événements pour détecter que les barrières sont bien fermées ou ouvertes. Elles sont non bloquantes, c'est à dire qu'elles comprennent des attentes pendant lesquelles d'autres tâches peuvent être exécutées.
Les voitures sortent librement du parking, mais leur sortie est détectée par la boucle DI4.

On va programmer le contrôle du parking au moyen de tâches temps-réel.

**Question 1: Y a t il une ressource ?**

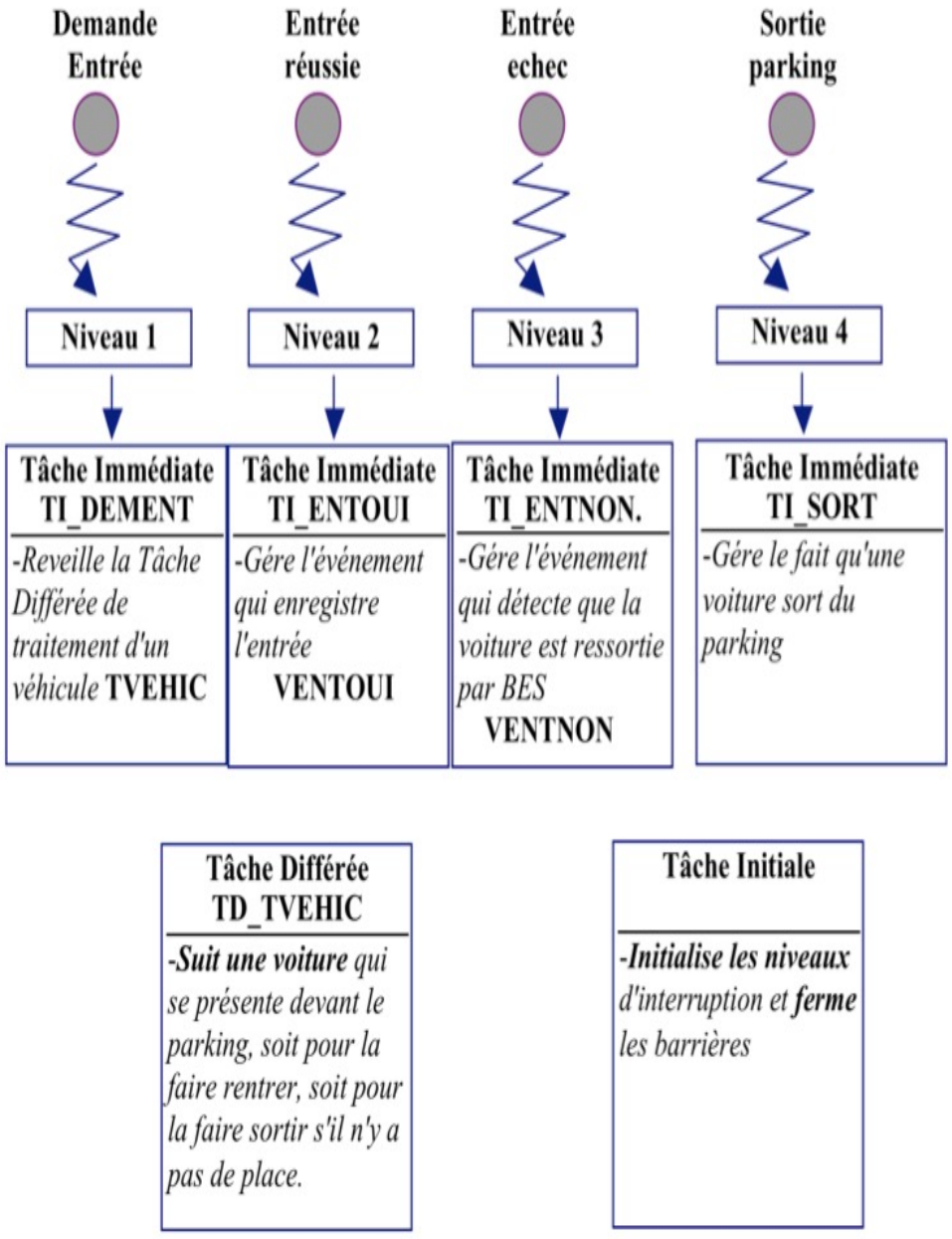
**Question 2: Comment structurer la solution ?**

Y a t il une ressource ?

Y a t il un élément partageable en quantité limitée ?

OUI: le parking - nombre d'utilisateurs: 45

2Comment structurer la solution ?




andreu@lirmm.fr

# Initiation au Langage Temps-Réel

**Déclarations**

Evénement VENTOUI, VENTNON
Ressource PARKING 45

**Tâche Immédiate TI_DEMENT**

début
-Réveille **TD_VEHIC**
fin

**Tâche Immédiate TI_ENTOUI**

début
MA_1 **VENTOUI**
fin

**Tâche Immédiate TI_ENTNON.**

début
MA_1 **VENTNON**
fin

**Tâche Immédiate TI_SORT**

début
LIBERE **PARKING**
fin

**Tâche Initiale**

début
ATTACHE 1 TI_DEMENT
ATTACHE 2 TI_ENTOUI
ATTACHE 3 TI_ENTNON
ATTACHE 4 TI_SORT

FORCE 1 ARME/DEMASQUE
FORCE 2 ARME/DEMASQUE
FORCE 3 ARME/DEMASQUE
FORCE 4 ARME/DEMASQUE

FERME BEE
FERME BES
fin

**Tâche Différée TD_VEHIC**

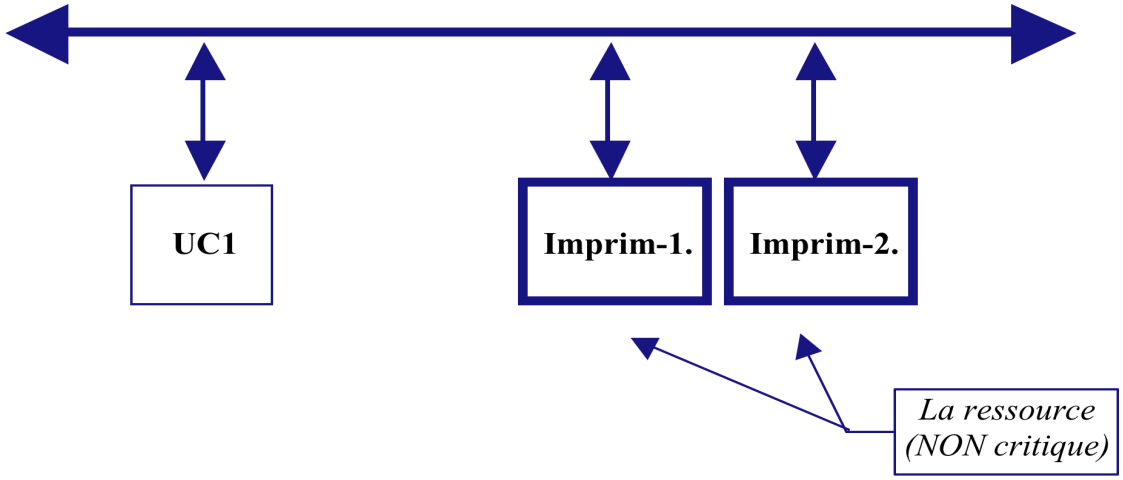
début
-Teste PARKING P1 P2
Si P1 = 0
Alors *ressource libre*
ouvre BEE
attend VENTOUI
MA_0 VENTOUI
ferme BEE
Sinon *ressource occupée*
ouvre BES
attend VENTNON
MA_0 VENTNON
ferme BES
Finsi
fin

## Gestion du Corps de la Ressource

### Les limites des instructions Réserve, Libère

Il a été vu que les instructions RESRVE et LIBERE permettent ou non à une tâche de continuer selon qu'une ressource est libre ou occupée.

**Il peut être nécessaire, quand une ressource peut avoir plusieurs utilisations, que la tâche utilisatrice tienne compte du numéro d'utilisation de l'élément de la ressource.**




Il faut gérer le "**partage du corps de la ressource**".

**Comment gérer le partage du corps de la ressource ?**

Pour gérer ce partage, il faut introduire des paramètres renseignant sur l'état des différentes parties de la ressource.

Le nombre de paramètres à introduire peut varier en fonction de la complexité de la ressource, mais le minimum consiste à introduire les paramètres suffisants pour identifier l'occupation des parties composant la ressource (sur l'exemple, identifier l'état de chaque imprimante)

**Exemple 1 : les imprimantes d'un pool d'impression**

Pour savoir quelle imprimante est libre, on va associer à chaque imprimante un drapeau d'état: **IMP1_OQP et IMP2_OQP**

| | |
|---|---|
| IMP1_OQP à 0 | indique que l'imprimante 1 est libre |
| IMP1_OQP à 1 | indique que l'imprimante 1 est occupée |
| IMP2_OQP à 0 | indique que l'imprimante 2 est libre |
| IMP2_OQP à 1 | indique que l'imprimante 2 est occupée |

Ces drapeaux vont permettre de choisir une imprimante libre quand on utilisera la ressource Impression.

# Initiation au Langage Temps-Réel

**Tâche TD**

début
calculs
**Reserve Impression**
Si **IMP1_OQP = 0**
Alors
**1 -> IMP1_OQP**
**IMPRIME (1, "Message")**
**0 -> IMP1_OQP**
Sinon
**1 -> IMP2_OQP**
**IMPRIME (2, "Message")**
**0 -> IMP2_OQP**
FinSi
**Libère Impression**
Fin

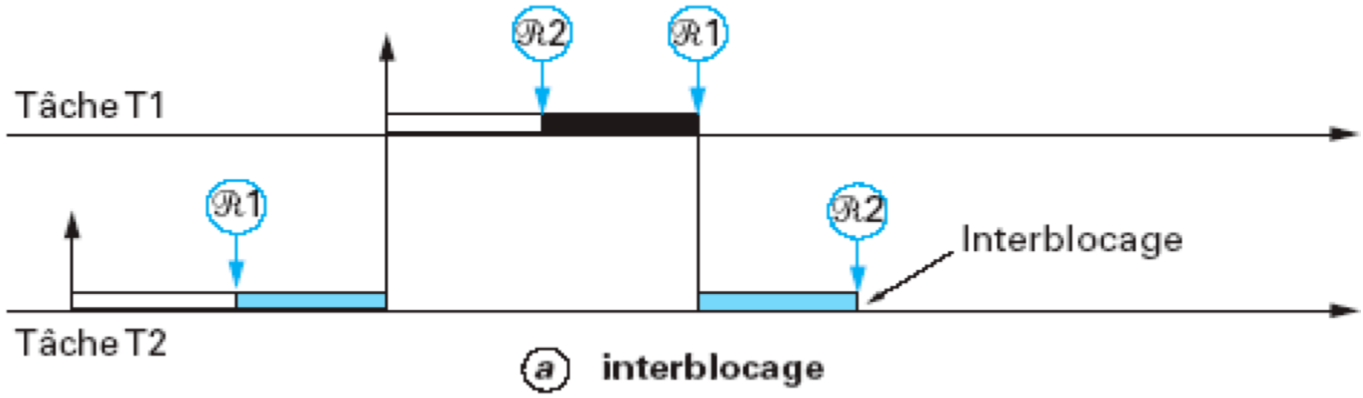
Libère Impression

**ATTENTION au risque de préemption ... Section critique**

# Problème « classique » en TR

## Ressources multiples et interblocage … illustration

Supposons que les deux tâches T1 et T2 ont besoin chacune de deux ressources exclusives R1 et R2, et que ces ressources ne peuvent pas être retirées à une tâche qui les utilise en section critique.



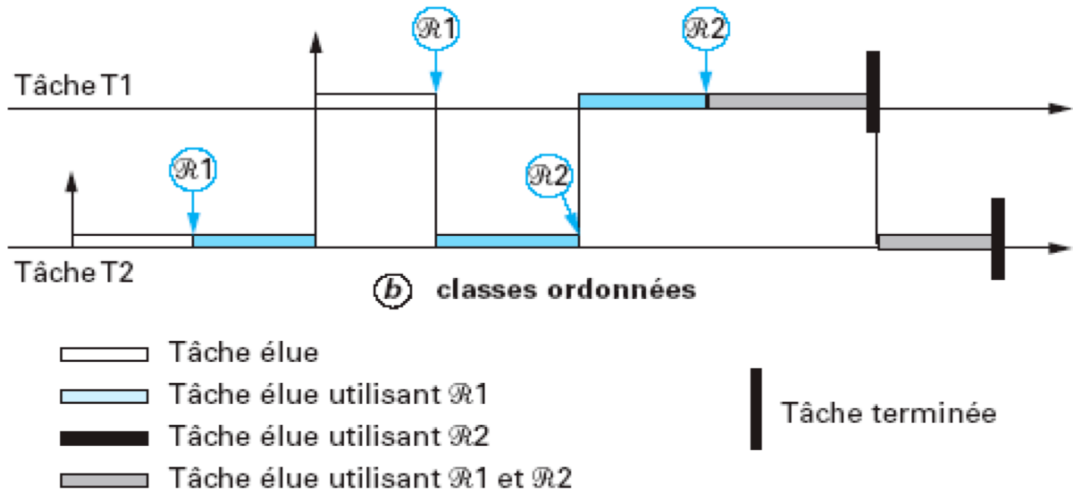

ⓐ interblocage

Si T1 demande successivement R2 et R1 et si T2, au contraire, demande R1 puis R2, les deux tâches peuvent parfois s'attendre mutuellement indéfiniment comme dans le cas décrit sur la figure (a) ci-dessous. C'est une situation d'interblocage, qui peut se généraliser à tous les cas d'attente circulaire entre plusieurs tâches, et dont on ne peut sortir qu'en détruisant une ou plusieurs tâches. Pour une application temps réel, c'est une situation inacceptable.

# Problème « classique » en TR

## Ressources multiples et interblocage … solution générale

Une méthode, appelée la **méthode des classes ordonnées**, permet d'éviter que l'interblocage puisse se produire, **en programmant la demande des ressources selon le même ordre pour toutes les tâches**. Elle n'est applicable que si l'on connaît, dès la programmation, toutes les ressources que les tâches vont appeler. C'est pourquoi on dit que c'est une méthode de **prévention statique**. Dans l'exemple de la figure (b), l'interblocage ne peut jamais se produire si les deux tâches demandent R1 puis R2.



# Problème « classique » en TR

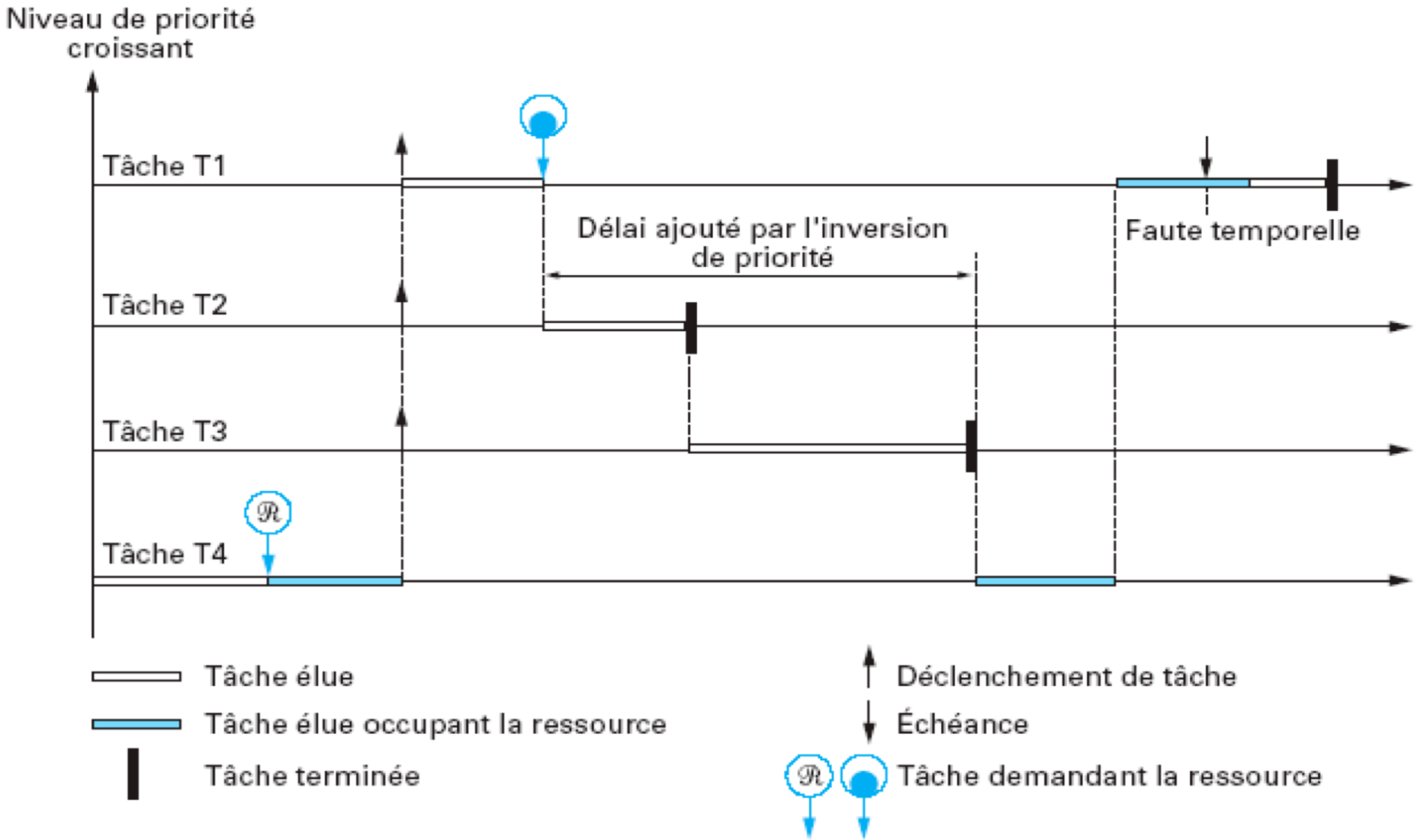
**Impact potentiel de la réservation de ressources … inversion de priorité**

Dans un ordonnancement préemptif, la présence simultanée de priorités fixes et de ressources à accès exclusif peut entraîner un phénomène appelé inversion de priorité.

***Exemple :*** soit quatre tâches de priorité décroissantes, T1, T2, T3 et T4. À un instant donné seule T4 est déclenchée et a obtenu une ressource critique. Puis T1, T2 et T3 sont réveillées par un événement externe. L'ordonnanceur préemptif à priorités élit T1. Supposons que T1 réclame la ressource critique déjà allouée à T4. Comme cette ressource ne peut être réquisitionnée pour T1, cette tâche doit attendre que T4 la libère. L'ordonnanceur à priorité élit donc T2, puis T3, qui s'exécutent. Ce n'est qu'après leur exécution que l'ordonnanceur peut élire T4 qui peut alors finir d'utiliser la ressource critique et, enfin, la libérer. Et seulement alors, T1, la tâche la plus prioritaire, peut s'exécuter. Le temps de réponse de T1 s'est rallongé du temps d'exécution de T2 et T3, moins prioritaires qu'elle. Cela peut parfois entraîner un dépassement d'échéance.

# Problème « classique » en TR

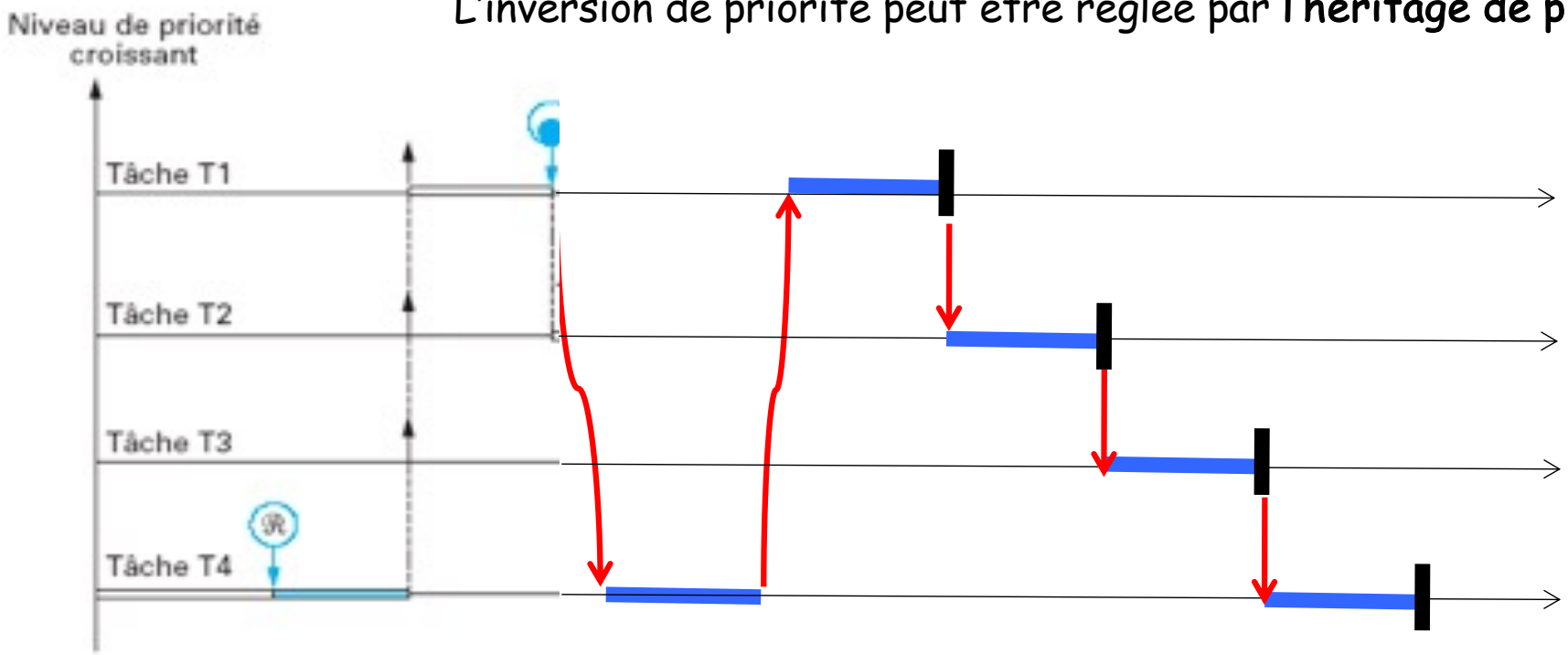
**Impact potentiel de la réservation de ressources … inversion de priorité**



# Problème « classique » en TR

**Impact potentiel de la réservation de ressources … inversion de priorité**

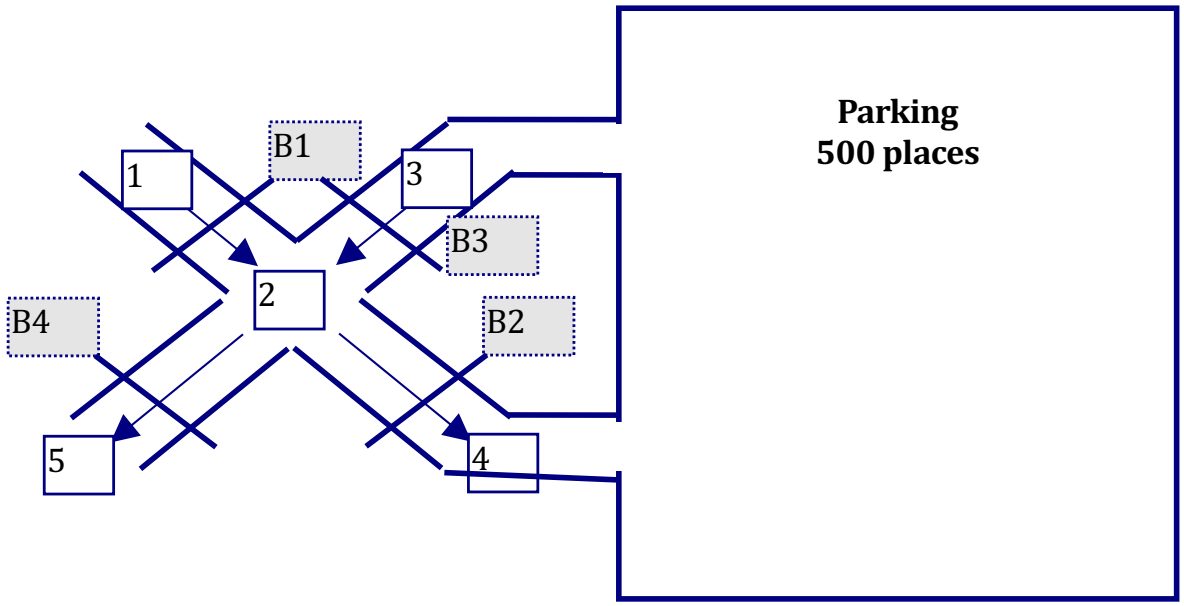
L'inversion de priorité peut être réglée par **l'héritage de priorité**




Dans le cas présenté, la tâche T4 hériterait **momentanément**
de la priorité de la tâche T1 (jusqu'à la libération du sémaphore concerné)
afin de limiter le retard induit par l'inversion de priorité.

# Initiation au Langage Temps-Réel

Exemples de gestion de ressources … **ressources multiples**



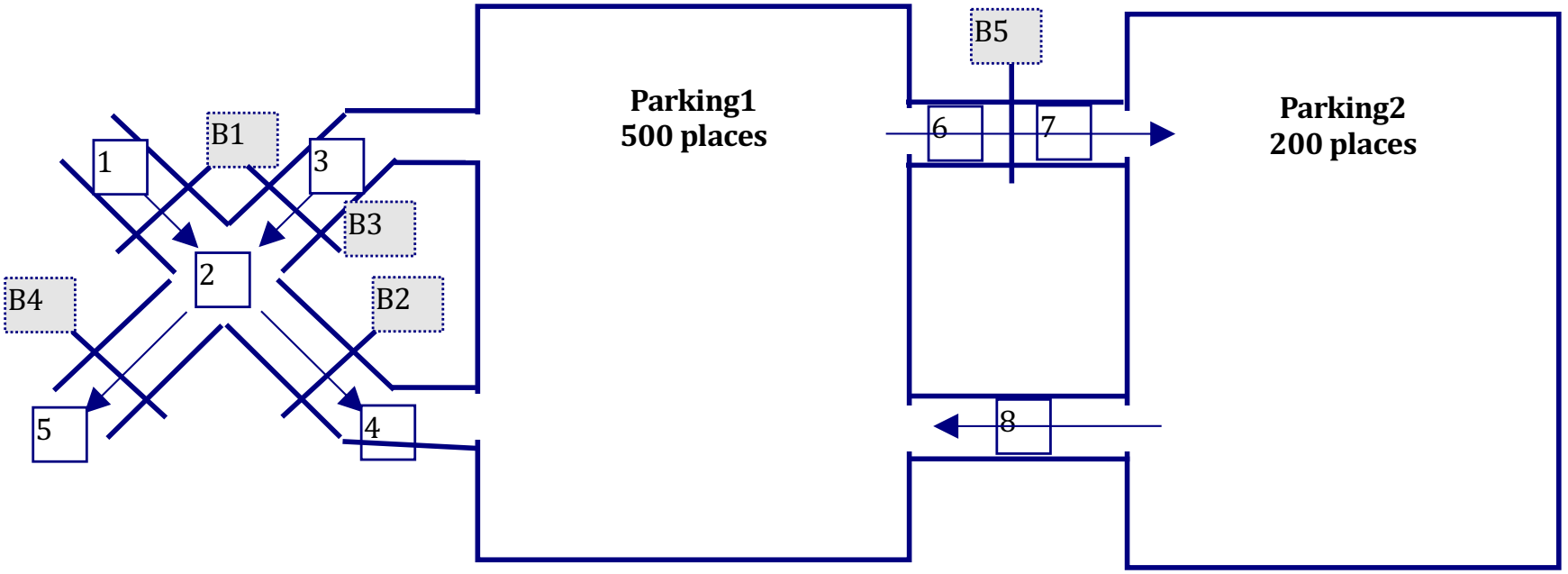

Attention à
l'ordre de réservation

Les ressources intrinsèques du système sont:

- **le Parking (500 places)**
- **le croisement en 2 (1 place)**

# Initiation au Langage Temps-Réel

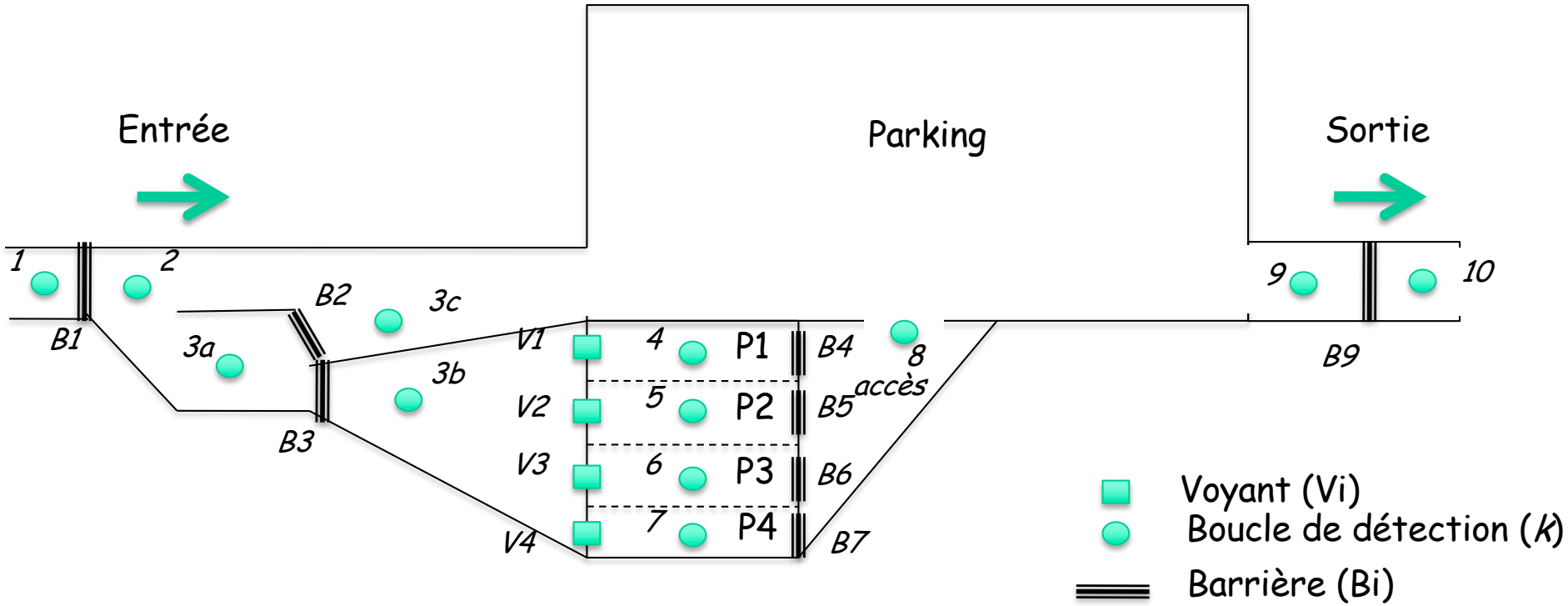
Exemples de gestion de ressources … **ressources agrégées**




**les ressources** **PARKINGTOTAL 700**
**PARKING2 200**

# Initiation au Langage Temps-Réel

Exemples de gestion de ressources … **ressources à corps multiple**



# SEMAPHORES ET COMMUNICATIONS ENTRE TACHES

## Introduction

Les sémaphores constituent un ensemble de mécanismes de base permettant à des processus informatiques asynchrones de communiquer et de se synchroniser. Ils sont à la base des procédures de gestion des événements et des ressources.

Nous allons en voir les définitions et les mécanismes.

Certains langages "Temps Réel" les utilisent tels quels.

## Sémaphores (Dijkstra-67/68)

### Définition

Un sémaphore "s" est constitué

- d'une variable entière e(s)
- d'une file d'attente f(s)

# Initiation au Langage Temps-Réel

## Primitives d'utilisation

Le sémaphore "**s**" est utilisé à travers deux primitives **p(s)** et **v(s)** qui sont insérées dans des **processus** (suite temporelle d'exécution d'instructions, appelée précédemment "Tâche")

**1 .1.1 P(s)**

**p(s):** *(**P** est le processus dans lequel p(s) est exécuté)*

**e(s) ← e(s) - 1**

**Si e(s) < 0 alors Blocage du processus P**

**P ⇒ f(s)** *(P va en file d'attente)*

**1. 1.2 v(s)**

**v(s):** *(**Q** est le processus dans lequel v(s) est exécuté)*

**e(s) ← e(s) +1**

**Si e(s) ≤ 0 alors Lire f(s) → P' actif**

*(Débloquer un processus P' contenu dans f(s - Critère de choix dans f(s) quelconque (FIFO, Prior, etc.).*

# Initiation au Langage Temps-Réel

## Application

Protection d'une ressource.

| | |
|---|---|
| **RESERVE = p(s).** | **LIBERE = v(s)** |

Le nombre d'utilisateurs possibles est $\mathbf{e_0(s)}$ **: NMAX**

**Note importante: p(s) et v(s) sont ININTERRUPTIBLES**

⇩

**Attention à l'impact de la préemption**

Illustration sur l'accès à des données en mémoire partagée
Notion de ré-entrance

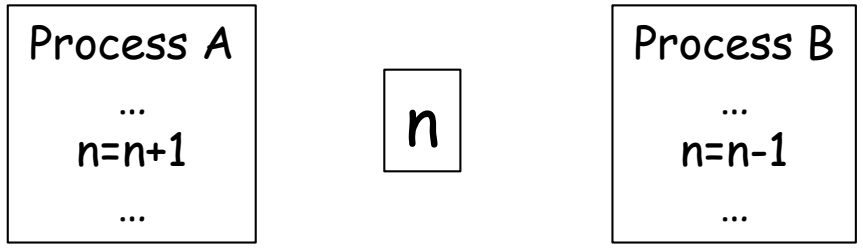
# Problème « classique » en TR

Impact de la préemption
Illustration sur l'accès à des données en mémoire partagée

**Variables partagées (shared variables)**
-Possible entre threads ou dès lors que l'espace d'adressage est commun aux tâches (cas de microC-OS par exemple)

**Mémoire partagée (shared memory)**
- Possible entre processus, mapping de zones mémoires.

**Problème de l'accès multiple … nécessité de synchronisation**
Soient 2 threads (analogie possible avec 2 process) manipulant une même variable n. Soit n =5, et le process A lancé avant le process B.

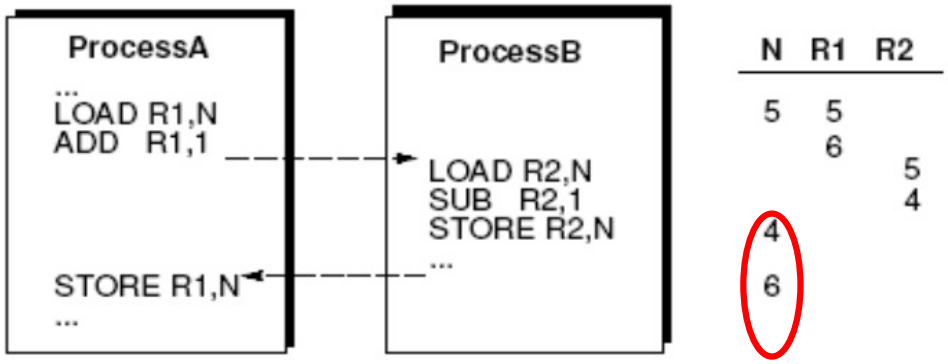


# Problème « classique » en TR

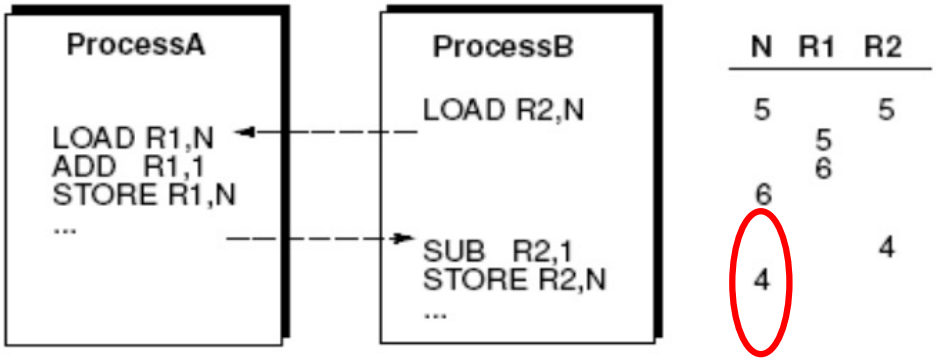
Les instructions haut-niveau ne sont pas **atomiques** …

| Process A | n | Process B |
|---|---|---|
| … | | … |
| LOAD R1,n | | LOAD R2,n |
| ADD R1, 1 | | SUB R2, 1 |
| STORE R1, n | | STORE R2, n |
| … | | … |

Conséquence, si B préempte A avant la fin de l'incrémentation … ?




Conséquence, si A préempte B avant la fin de l'incrémentation … ?

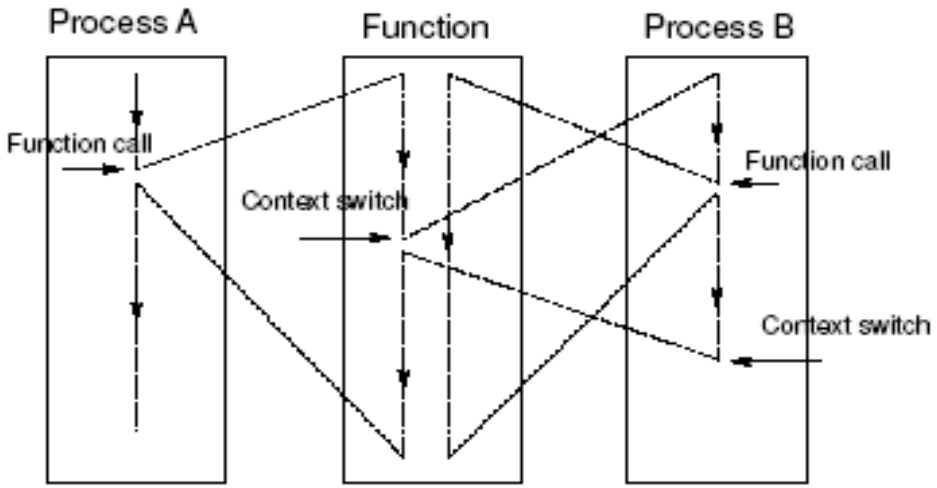



**La valeur de n est celle issue du process A ou du process B … indéterminé !**

# Problème « classique » en TR

Le problème est identique dans le cas de code partagé **non-réentrant** (i.e. utilisant des variables « globales »).

Remarques :
- **réentrance garantie pour les primitives offertes par un système d'exploitation certifié temps-réel**.
- le problème de code non-réentrant se pose souvent dans les librairies.




**Mettre en place une exclusion mutuelle**
- protéger en accès multiple par sémaphore d'exclusion mutuelle « mutex »
- ou **section critique**
(masquage des it, opérations indivisibles, ie atomiques).

```
   Process A
...
disable interrupts;
access critical section;
enable interrupts;
...
```

# Sémaphores d'exclusion mutuelle

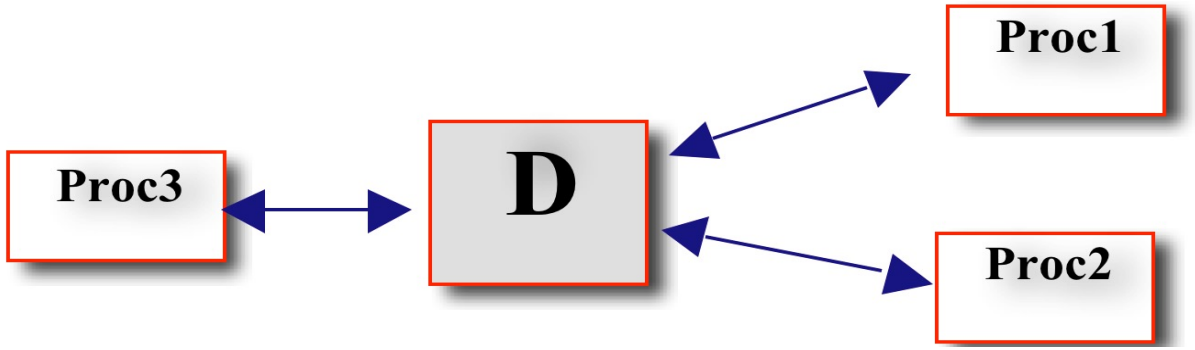
## Définition

C'est un sémaphore "mutex"dont la valeur initiale est **$e_0$(mutex)=1**

**On appelle alors la ressource une "ressource critique"**

## Application

Protection contre l'accès multiple à une donnée commune D (Tableau-Base de Données)



# 1 Synchronisation entre Tâches par sémaphores

## Principe

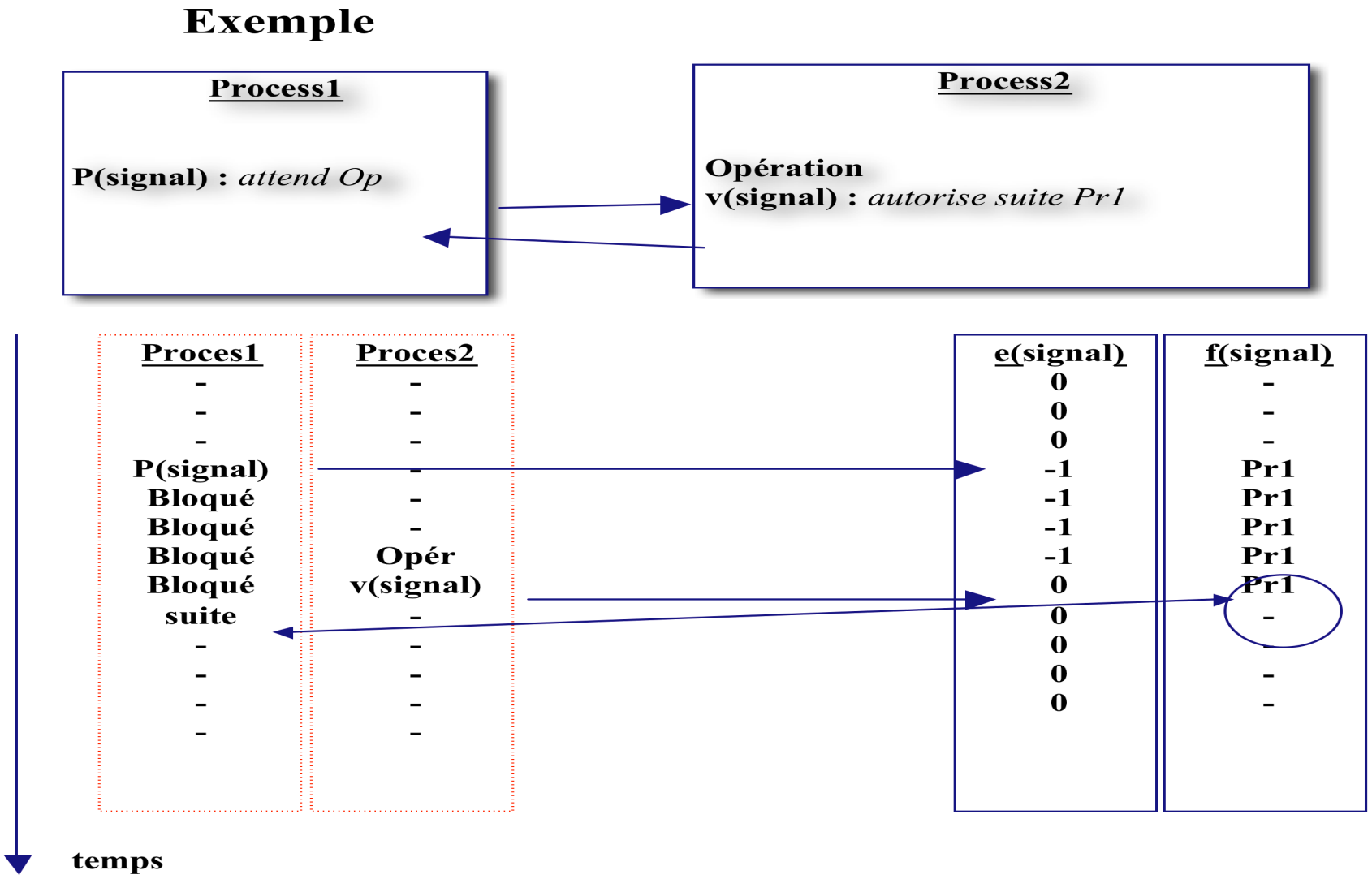
Un sémaphore peut également servir à faire attendre une Tâche.
Une tâche **Process1** attend qu'une tâche **Process2** soit passée en une opération **Opér**, pour continuer.
Le sémaphore (ici "signal") joue alors le rôle d'un événement "**event**".
Il est mis à 0 à l'instant initial.

## Application

| | |
|---|---|
| **Wait Event = p(signal);** | **MetA1 Event = v(signal)** |

# Initiation au Temps-Réel

**Exemple**



# Initiation au Temps-Réel

## Communication par message, avec sémaphore

## Ex : Modèle Producteur-Consommateur

### Définition

Une Tâche nommée "**Producteur**" envoie dans un **Tampon** des messages à traiter. La vitesse de Producteur est inconnue.
Une autre Tâche nommée "**Consommateur**" lit **Tampon** et traite les messages un par un. Sa vitesse est aussi inconnue.

## Principe de la solution

On utilise deux sémaphores:

**nvide**: représente **"le nombre d'emplacements libres du Tampon"**
**- $e_0$(nvide) = n**

**nplein**: représente **"le nombre de messages à traiter"**
**- $e_0$(nplein) = 0**

**Producteur**

Début
ProductionMessage
**p(nvide)**
Mise Message en T
**v(nplein)**
Retour Début

**Consommateur**

Début
**p(nplein)**
Lecture Message de T
**v(nvide)**
TraiteMessage
Retour Début

# Initiation au Temps-Réel

## Sémaphore à Message

### Définition

Il permet en même temps de synchroniser et d'envoyer des messages entre processus

C'est un sémaphore qui comporte
- l'entier **e(s)**
- une file d'attente de processus **$f_p(s)$**
- une file d'attente de messages **$f_m(s)$**

### Primitives

les primitives p et v gèrent aussi les messages
-Pm(s) gère e(s) et lit un message de **$f_m(s)$**
-Vm(s) gère e(s) et écrit un message dans **$f_m(s)$**

**Pm(s,mr)**

```
e(s) ← e(s) - 1
Si e(s) < 0
        Etat(Proc) = bloqué
        proc ->  fp(s)
Sinon
        Lire M dans fm(s)
        mr = M
Finsi
```

**Vm(s,mt)**

```
e(s) ← e(s) + 1
Si e(s) ≤ 0
        Choisir Proc' dans fp(s)
        Lire fm(s) -> mt
        Activer Proc'
Sinon
        mt ->  fm(s)
Finsi
```

# Initiation au Temps-Réel

## Bilan Sémaphores

***Sémaphore binaire***
variable de type booléen, pour régler les conflits en exclusion mutuelle (souvent initialisé à 1).

***Sémaphore à compteur***
variable de type entier (compteur), pour régler les conflits en exclusion mutuelle ou bien pour des ressources partageables.

***Sémaphore comme objet de synchronisation entre tâches (ex : EVT, RDV)***
objet de synchronisation, pour signaler un evt, possible signalisation multipoints (en vidant la file d'attente d'un sémaphore), (souvent initialisé à 0).

***Sémaphore d'exclusion mutuelle***
Le sémaphore d'exclusion mutuelle apporte trois raffinements supplémentaires :
- la prise en compte de l'inversion de priorité par un algorithme d'héritage de priorité, - la protection contre la suppression pour une tâche en section critique (suppression différée);
- la gestion des accès récursifs sur un même sémaphore (ensemble de routines qui ont besoin d'exclusion mutuelle et qui s'appellent l'une l'autre).

# Initiation au Temps-Réel

## Communication par message, avec boîte aux lettres

La communication de base est **normalement** conçue autour de la communication **asynchrone** par boîte aux lettres, de taille fixe ou variable. Cet objet est toujours un objet public sur lequel on peut mettre en œuvre un modèle de coopération producteurs/consommateurs et cette technique est qualifiée d'asynchrone car il y a un modèle d'exécution asynchrone entre les partenaires producteurs et consommateurs.

L'objet Boîte aux lettres se compose de deux files : l'une est destinée à recevoir les messages en attente de consommation, l'autre est destinée à recevoir les tâches en attente de message. Si une file contient un élément, alors l'autre file est forcément vide. La gestion de la file des messages s'effectue en mode FIFO, sauf si l'exécutif a un service utilisant la notion de « message urgent », ce dernier étant doté d'une priorité purement applicative, sans rapport avec la priorité de la tâche productrice. La file des tâches peut être gérée en FIFO ou par la priorité (des tâches).

# Initiation au Temps-Réel

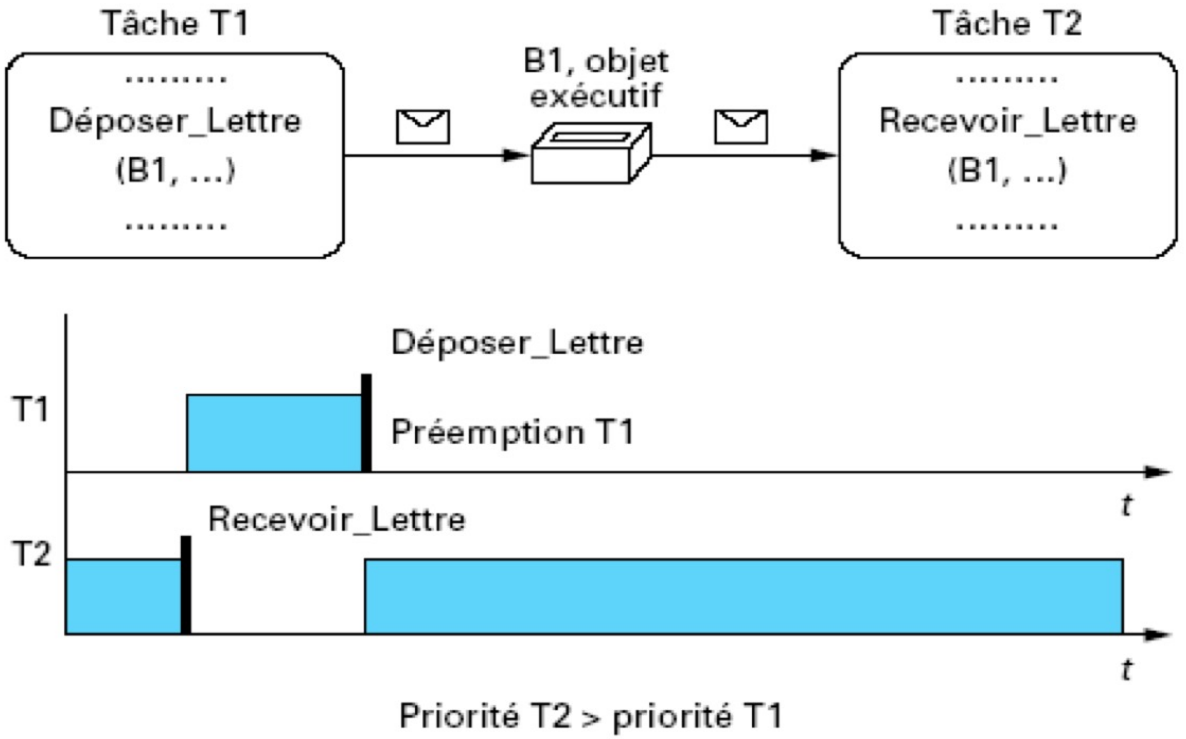
## Communication par message, avec boîte aux lettres

Le contenu et la taille d'un message peuvent être conditionnés par l'implémentation, mais on peut envisager des services permettant :

- la transmission par valeur de messages de taille fixe ou variable. Dans ce cas, il est souhaitable que l'exécutif recopie les messages, libérant ainsi l'espace mémoire utilisé par la tâche productrice pour déposer le message. Il s'ensuit que le dépôt d'un message peut être retardé si l'exécutif ne dispose plus de tampon mémoire ;
- la transmission de messages par référence dans la mesure où les espaces d'adressage des tâches sont compatibles pour que le consommateur puisse faire des références dans l'espace mémoire de la tâche productrice. Dans ce cas, il peut être nécessaire de mettre en oeuvre un protocole applicatif supplémentaire pour protéger les données produites tant que le consommateur ne les a pas utilisées.

# Initiation au Temps-Réel

Communication par message, avec boîte aux lettres

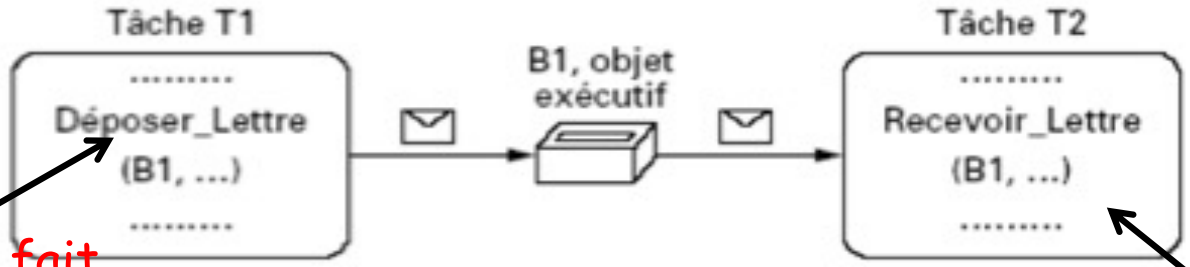



Attention au comportement réel des primitives de l'OS

# Initiation au Temps-Réel

Communication par message, avec boîte aux lettres

Exemple de comportement de primitives non asynchrone … cas QNX




Contient en fait
Attente ACK

Nécessaire donc
de signaler ACK !

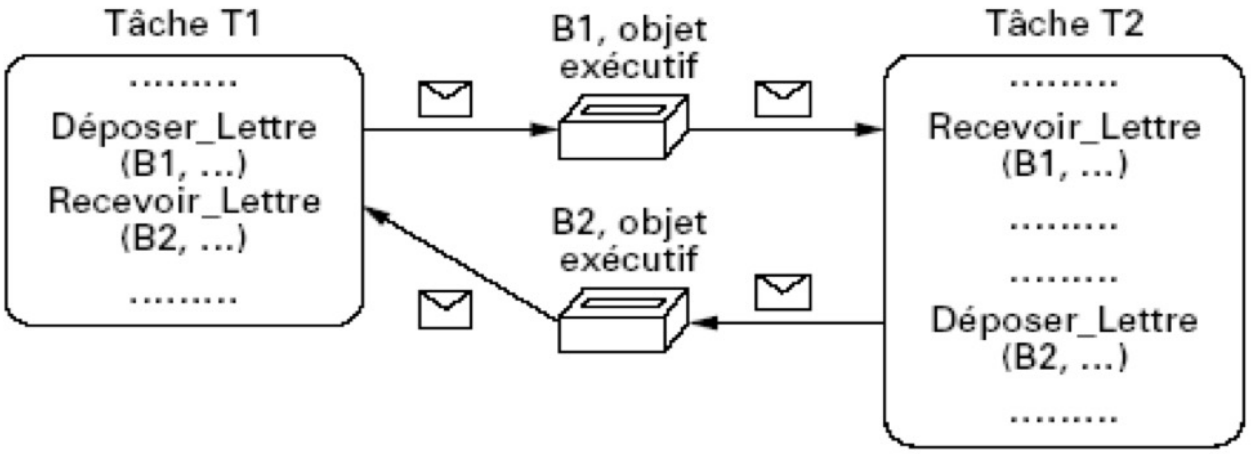
C'est un modèle de communication **synchrone** car les intervenants doivent être présents au point de rendez-vous pour que l'échange puisse avoir lieu (ici, l'émetteur attend ACK du récepteur). Le principal avantage de cette technique est qu'elle ne nécessite pas de stockage intermédiaire des données échangées, ce qui la rend très déterministe d'un point de vue comportemental.

C'est une communication par rendez-vous

# Initiation au Temps-Réel

## Communication par message, RDV via boîte aux lettres

**Remarque** : il est toujours possible de faire une communication synchrone entre deux tâches en utilisant le mécanisme asynchrone des boîtes aux lettres. Il suffit de mettre en place un échange «croisé» dans chaque tâche, comme le montre la figure ci-dessous.



# Initiation au Temps-Réel

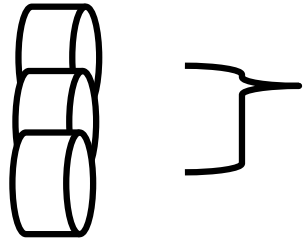
## Communication par message, avec « pipes »

On rencontre parfois dans des exécutifs, la mise en oeuvre des « tuyaux » (pipe) pour la communication entre tâches sur un même processeur. Ces objets sont alors gérés en utilisant les fonctions standard d'un système d'entrée/sortie (open, close, read, write, ctrl).

### Intérêt

Même si la sémantique de ces objets est la même que celle des boîtes aux lettres, on a toutefois une possibilité très intéressante sur cet objet de communication qui est de pouvoir faire une **attente sélective sur plusieurs tuyaux** (**select**), ce que ne permettent pas les boîtes aux lettres traditionnelles.

fd1
fd2
fd3



```
fd-liste = fd1 & fd2 & fd3
select (fd-liste)
si source = fd1 alors …….
si source = fd2 alors …….
……
```

attente sélective
sur plusieurs pipes

# Concepts temps-réel et modélisation par RdP

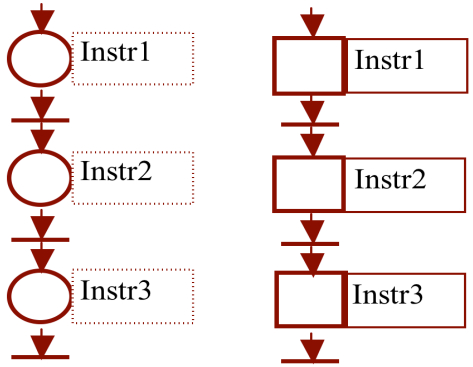
**Concepts & mécanismes de base**

**Tâche, processus**

**LANGAGE Temps Réel**

**Tâche TA**
début
**Instr1**
**Instr2**
**.....**
**Instrn**
fin

**Modèle RdP/Grafcet**




**NOYAU Temps Réel**

**Process TA**
begin
**Instr1**
**Instr2**
**.....**
**Instrn**
end

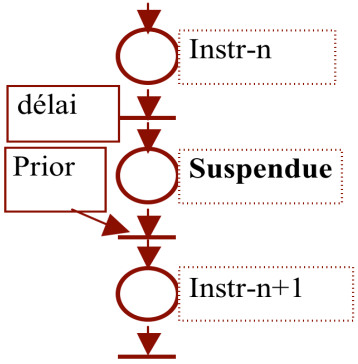
**Concepts**

**Attente**

**LANGAGE Temps Réel**

**Instr-n**
**ATTEND 3s**
**Instr-n+1**

**Modèle RdP**




**NOYAU Temps Réel**

***Primitive de base***
**Wait 3s**

***Ou Ecriture directe en Sémaphore***

**-Tâche qui attend**
**lance Timer 3s**
**P(s)**

**-Tâche Timer**
**V(s)**

# Concepts temps-réel et modélisation par RdP

**Concepts et Mécanisme de base**

**Synchro entre Tâches sémaphore s(t0)=0**

**ATTEND = p(s)**

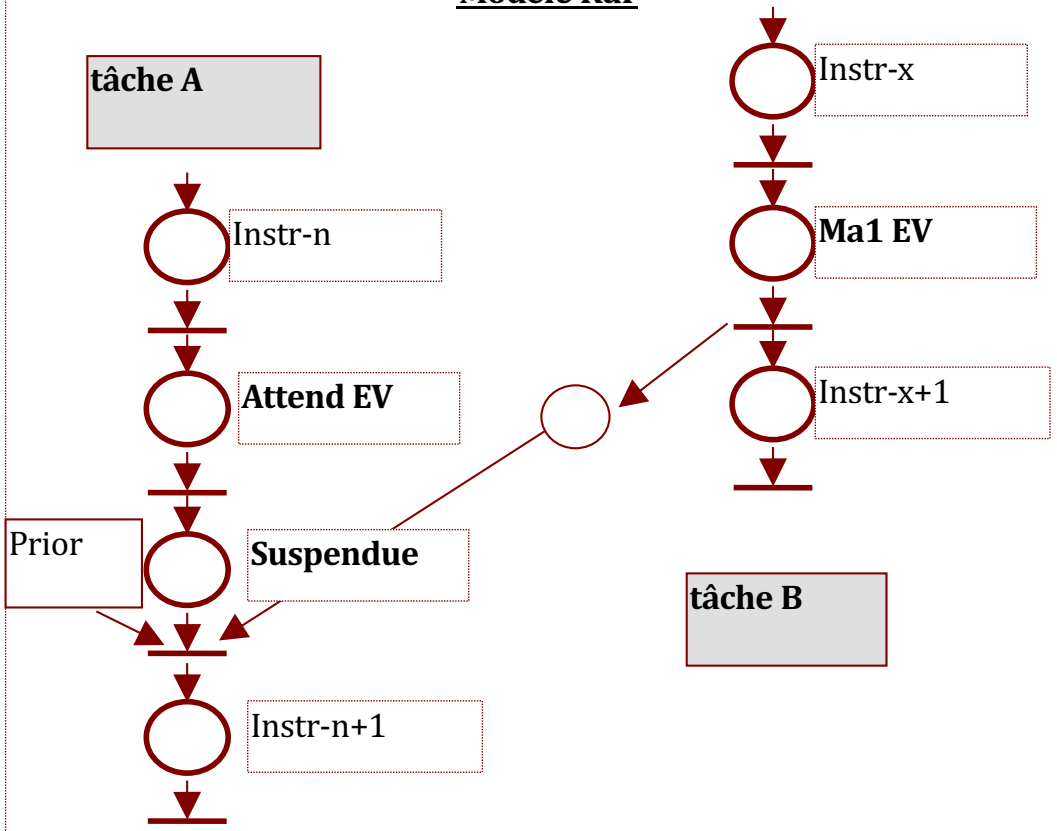
**MA1 = v(s)**

**LANGAGE Temps Réel**

**Tâche TA**
**Instr-n**
**ATTEND EV**
**Instr-n+1**

**Tâche TB**
**Instr-x**
**MA1 EV**
**Instr-x+1**

**Modèle RdP**



# Concepts temps-réel et modélisation par RdP

**Concepts**

**Synchro avec Messages**

**1 sémaph.**
**S(t0)=0**
**1 File de M**

**ATTEND**
**=**
**p(s)+litM**

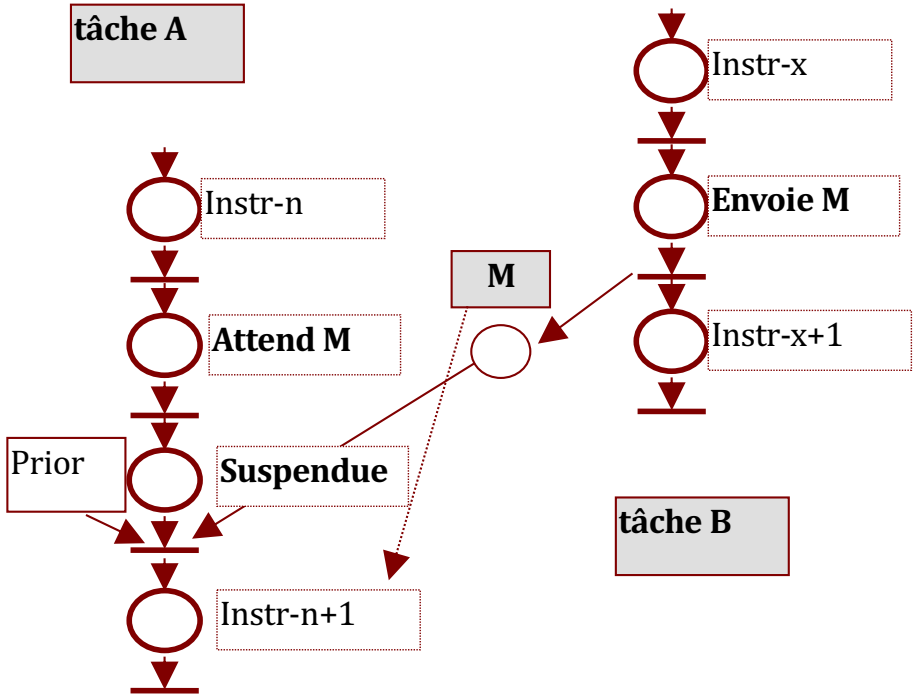
**ENVOIE**
**=**
**v(s)+metM**

**LANGAGE Temps Réel**

**Tâche TA**
**Instr-n**
**ENVOIE Message**
**Instr-n+1**

**Tâche TB**
**Instr-x**
**ATTEND Message**
**Instr-x+1**

**Modèle RdP**



# Concepts temps-réel et modélisation par RdP

**Concepts**

**Synchro avec Messages Bloquant**

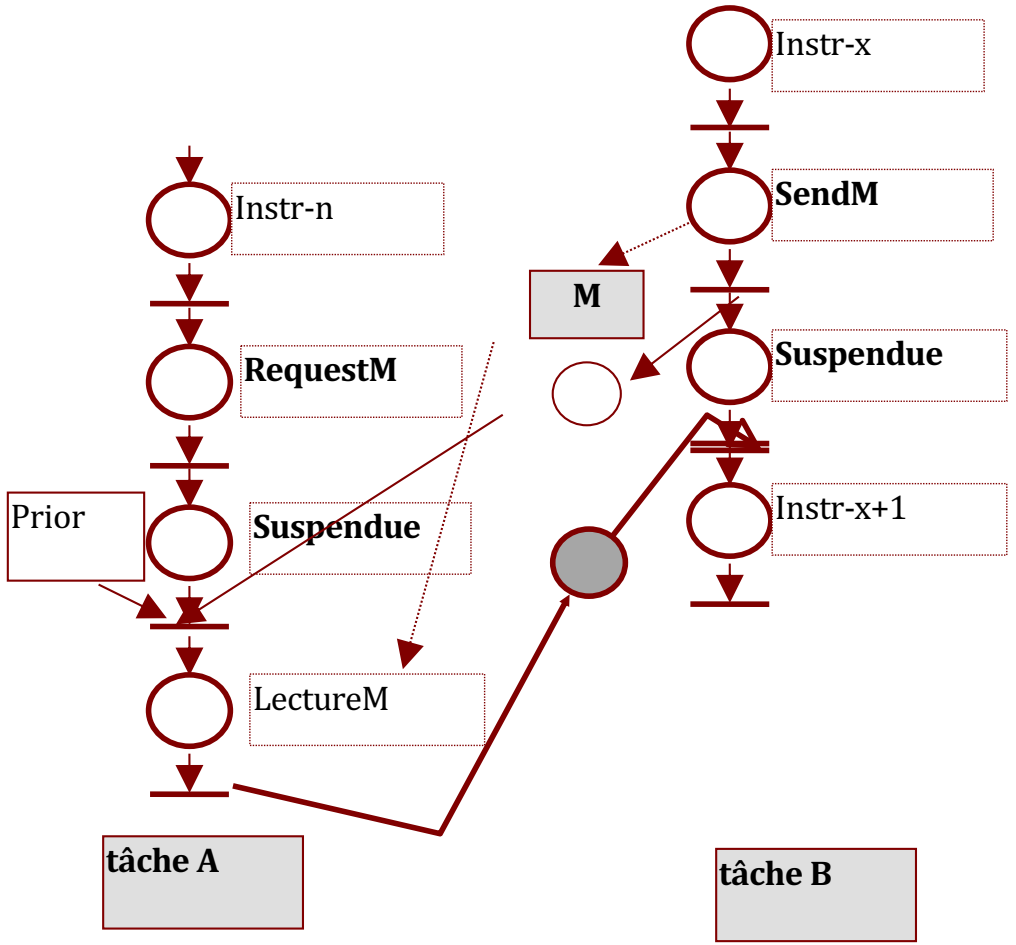
**2 sémaph.**
**1 File de M**

**Request**
**=**
**p(s1)+litM**
**+ v(s2)**

**Send**
**=**
**v(s1)+metM+**
**p(s2)**

**LANGAGE Temps Réel**

**Modèle RdP**



# Systèmes à temps critique

Ce sont des systèmes dont les contraintes de date d'exécution sont strictes :

- Systèmes rapides qui font des tâches non cycliques,
  avec des temps maximum d'exécution stricts,
  ou des tâches cycliques lourdes.
- Système confronté à des événements imprévisibles,
  et qui doit répondre rapidement

ex: Spatial, Avionique, Transport, Robotique, etc.

Notions d'ordonnancement

# Systèmes à temps critique

## Notions d'ordonnancement

### Notion de condition d'admission

Si on donne trop de choses à faire au système en un temps strict, il peut se révéler impossible de le faire.

Ex: -à une date d1 on réveille T1 durée 300ms - T2 durée 250ms - T3 700ms

-on veut qu'elles soient toutes terminées à d1 + 900ms

dT1+dT2+dT3 = 1250 ms - Impossible à faire en 900ms

*Voir aussi page suivante, l'exemple*

Au moment d'admettre une nouvelle tâche dans un système, on cherchera si elle vérifie des conditions d'admission (Que l'ensemble des durées des tâches à faire ne dépasse pas le temps restant).

Si elle ne peut être admise on remplacera l'une des tâches (soit déjà existante, soit celle que l'on veut admettre) par une autre tâche, **moins précise, mais plus courte** de façon à vérifier les conditions d'admission

Ces conditions sont calculées au fil de l'eau, au fur et à mesure que les tâches se terminent et commencent.

# Initiation au Langage Temps-Réel

Illustration d'un système surchargé

**Les dates**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**Les tâches à exécuter avec les dates de fin d'exécution**

| A | A | A | A | A | A | | | | B | B | B | B | | | | C | C | C | C | C | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 10 | 10 | 10 | 10 | 10 | | | | 7 | 7 | 7 | 7 | | | | 9 | 9 | 9 | 9 | 9 | | | | | | | |

**Les solutions (toutes les combinaisons possibles)**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | A | A | A | A | B | B | B | B | | | | | | | | | | | SOLUTION MAUVAISE dès la première tache |
| 10 | 10 | 10 | 10 | 10 | 10 | 7 | 7 | 7 | 7 | | | | | | | | | | | |
| A | A | A | A | A | A | C | C | C | C | C | | | | | | | | | | SOLUTION MAUVAISE idem |
| 10 | 10 | 10 | 10 | 10 | 10 | 9 | 9 | 9 | 9 | 9 | | | | | | | | | | |
| B | B | B | B | A | A | A | A | A | A | C | C | C | C | C | | | | | | SOLUTION MAUVAISE seulement pour la dernière |
| 7 | 7 | 7 | 7 | 10 | 10 | 10 | 10 | 10 | 10 | 9 | 9 | 9 | 9 | 9 | | | | | | |
| B | B | B | B | C | C | C | C | C | A | A | A | A | A | A | | | | | | SOLUTION MAUVAISE seulement pour la dernière |
| 7 | 7 | 7 | 7 | 9 | 9 | 9 | 9 | 9 | 10 | 10 | 10 | 10 | 10 | 10 | | | | | | |
| C | C | C | C | C | B | B | B | B | | | | | | | | | | | | SOLUTION MAUVAISE dès la seconde |
| 9 | 9 | 9 | 9 | 9 | 7 | 7 | 7 | 7 | | | | | | | | | | | | |
| C | C | C | C | C | A | A | A | A | A | A | | | | | | | | | | SOLUTION MAUVAISE dès la seconde |
| 9 | 9 | 9 | 9 | 9 | 10 | 10 | 10 | 10 | 10 | 10 | | | | | | | | | | |

# Classification des systèmes

## Selon la nature de l'ordonnancement

### Ordonnancement statique

Il est fait à priori. Il est en général Off-line donc il peut exiger des calculs complexes d'optimisation. Le résultat peut être performant.
Il n'est possible que si on connaît exactement les tâches à faire et leur durée.
Donc pas d'imprévu !

### Ordonnancement dynamique

Il est fait au fur et à mesure du changement d'état du système, fil de l'eau.
Il nécessite donc une connaissance à jour de l'état du système (Tâches en exécution, en attente, durée de ces tâches, dates de fin obligatoires)
Le calcul de l'ordonnancement est lui-même une tâche qui s'ajoute à la charge du système. Il doit donc être simple, sous peine de surcharger le système.

## Selon la précision

Quand toutes les tâches ne rentrent pas on peut également décomposer le problème en classant les tâches en deux catégories:

Les tâches à contraintes strictes:
La date de fin DOIT être respectée

Les tâches à contraintes relatives
La date de fin PEUT être dépassée

## Conclusion

Ce sont des moniteurs spécialisés dans des domaines d'application, et même parfois sur mesure pour une application donnée.

# Ordonnancement des tâches périodiques

## Paramètres des Tâches périodiques

Les paramètres $\tau_i$ d'une tâche i sont:

$$\tau_i = \{ r_i, C_i, R_i, T_i \}$$

$r_i$ est la date de première disponibilité de la tâche $\tau_i$

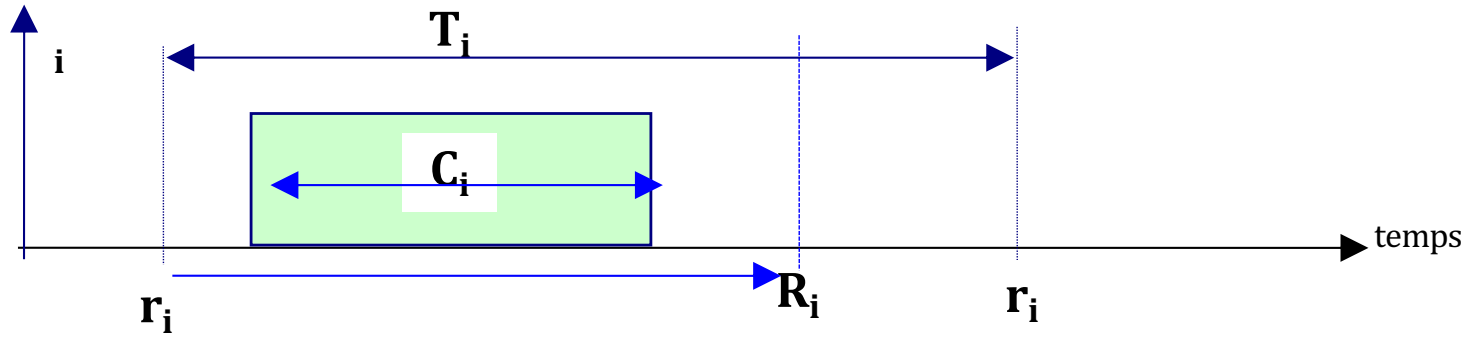
$C_i$ est la durée exécution de $\tau_i$ sans préemption (sans qu'elle soit coupée)

*NB : elle peut l'être, mais ça ne change pas la charge globale de l'UC sur la durée de vie de la tâche.*

$R_i$ est le délai critique au bout duquel la tâche $\tau_i$ doit être finie après $r_i$

$T_i$ est la période de $\tau_i$ c'est à dire l'intervalle de temps entre deux événements déclenchant $\tau_i$

# Initiation au Langage Temps-Réel




**Quelques définitions et conditions**

-*I est l'indice de la tâche*

- **Conditions d'existence:** $0 \leq C_i < R_i - r_i \leq T_i$
- **Facteur de charge local:** $ChL_i = C_i / (R_i - r_i)$ **(% du temps pris par cette tâche) entre $r_i$ et $R_i$**
- **Facteur de charge global:** $ChG_i = C_i / T_i$ **(% du temps pris par cette tâche global)**

- **Condition d'admission:** $\sum_{i=1}^{i=n} Chx_i < 100\ \%$ **sur tout intervalle**

## Ordonnancement préemptif des tâches périodiques

L'objectif de l'ordonnancement consiste à définir les priorités relatives des tâches, pour conditionner leur élection.

### Intelligent Fixed Priority

Le processus (exécution d'une tâche) choisi comme le plus prioritaire est celui qui dispose du plus petit temps de réponse **Ri**

*Existe en deux Variantes très voisines: Deadline Driven, Earliest Deadline First*

**Choix par Date de fin: (**Earliest Deadline First EDF**)**
On choisit le processus (la tâche) dont la date dont la fin est la plus proche

**Choix par Date de début au plus tard: (**Deadline Driven**)**
On choisit le processus (la tâche) dont la date de début au plus tard (=date de fin - durée) est la plus proche

On constate qu'il y a une différence, mais que les deux donnent une solution correcte.

# Initiation au Langage Temps-Réel

## Exemples d'algorithmes

### Les taches à éxecuter, au moment ou elles sont éligibles

| | | | | | | | | | | Date de début au plus tard |
|---|---|---|---|---|---|---|---|---|---|---|
| A | A | A | A | A | | | | | | |
| 9 | 9 | 9 | 9 | 9 | | | | | | **4** |
| B | B | B | B | B | B | B | | | | |
| 25 | 25 | 25 | 25 | 25 | 25 | 25 | | | | **18** |
| C | C | C | C | | | | | | | |
| 12 | 12 | 12 | 12 | | | | | | | **8** |
| | | | D | D | D | D | D | | | |
| | | | 14 | 14 | 14 | 14 | 14 | | | **9** |
| | | | | | | | E | E | | |
| | | | | | | | 24 | 24 | | **22** |

## SOLUTIONS

### Choix par date de fin

| **date** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tache | A | A | A | A | A | C | C | C | C | D | D | D | D | D | E | E | B | B | B | B | B | B | B | | **OK** |
| date max fin | 9 | 9 | 9 | 9 | 9 | 12 | 12 | 12 | 12 | 14 | 14 | 14 | 14 | 14 | 24 | 24 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | | |
| | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | X | |

### Choix par date de début au plus tard

| **date** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tache | A | A | A | A | A | C | C | C | C | D | D | D | D | D | B | B | B | B | B | B | B | E | E | | **OK** |
| Date max début | 9 | 9 | 9 | 9 | 9 | 12 | 12 | 12 | 12 | 14 | 14 | 14 | 14 | 14 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 24 | 24 | | |
| | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | X | |

# Initiation au Langage Temps-Réel

Exercice : établir l'ordonnancement des tâches suivantes,
selon les deux méthodes évoquées (EDF, DD)

| Tâche | A | B | C | D | E |
|---|---|---|---|---|---|
| Durée (Ci) | 5 | 2 | 7 | 1 | 6 |
| Délai critique (Ri) | 9 | 7 | 15 | 11 | 22 |

NB (simplifications) :

1- on considèrera que les tâches sont indépendantes (i.e. leur date de première disponibilité est $\mathbf{r_i} = 0$).

2- on ne cherchera à ordonnancer que sur la première période

## Ordonnancement préemptif des tâches Apériodiques

Les algorithmes s'inspirent de I.F.P, mais en plus, comme des tâches aléatoires (apériodiques) apparaissent, à tout instant on doit comptabiliser les charges restant à faire pour les tâches en cours.
On calcule s'il reste de la charge UC disponible pour la nouvelle tâche dans son espace

$$[\ r_i\ ,\ R_i\ ]$$

pour l'accepter ou non.

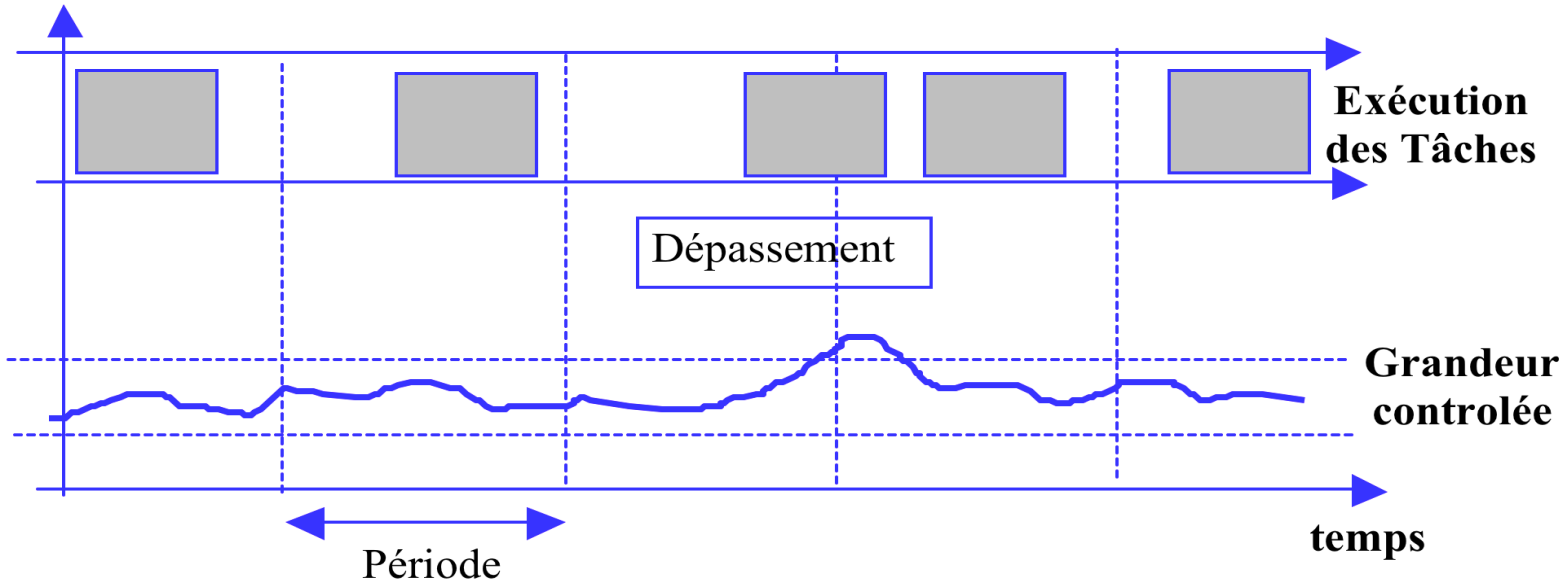
## Incidence des ressources

**Sur les Algorithmes statiques:**
Si des ressources interviennent, il n'y a d'autre solution que de calculer toutes les combinaisons possibles. Le problème est NP-Complet
**Sur les algorithmes dynamiques :**
Une tâche en section critique reçoit une priorité spéciale

## Conclusion

-Systèmes sur mesure
- Coûteux
- Nécessité par des applications critiques (spatial, nucléaire)

Industriellement, pour des applications courantes on utilisera des LTR en prenant des machines sous chargées et en admettant que les temps soient parfois dépassés...pourvu que ce soit rare !



# Complément du cours : Notions sur la norme POSIX

La norme POSIX ( Portable Operating System Interface for Computer Environments) :
- complète UNIX de fonctionnalités temps-réel,
- interface qui permet le portage des applications sur des supports d'exécution différents, tout en maintenant la même interface de programmation POSIX,
- définit plusieurs « extensions » ou standards sont les suivants, avec l'accent mis sur ceux liés au temps réel.

P1003.1 (ex-POSIX.1) définit une interface de programmation standard pour un système d'exploitation UNIX incluant la gestion des processus (fork, exec, kill,...), la gestion de l'environnement (getpid, getiud,...), la gestion des fichiers, etc. C'est le système de base que la plupart des systèmes d'exploitation UNIX implémentent.
- L'additif P1003.1a contient des extensions mineures nécessaires à P1003.1 ;
- P1003.1b (ex-POSIX.4) définit les extensions temps réel du système de base. Cela concerne l'ordonnancement, les sémaphores, les queues de messages, les signaux temps réel, les horloges à haute résolution, les entrées/sorties synchrones et asynchrones... ;
- P1003.1c (ex-POSIX.4a) étend le modèle traditionnel du processus UNIX, qui est initialement un simple « thread » d'exécution dans un espace d'adressage isolé, vers la possibilité d'exécuter de multiples threads au sein d'un processus (même espace d'adressage). On trouve également dans ce standard les « mutex » (sémaphore avec héritage de priorité) et les variables de condition ;
- P1003.1d (ex-POSIX.4b) apporte encore des extensions à P1003.1b notamment pour la gestion des interruptions et des traitants associés ;

# Complément du cours : Notions sur la norme POSIX

Remarques :
- Les exécutifs industriels qui implémentent les standards POSIX ne sont en général pas «nés» avec POSIX mais lui sont antérieurs. Ils proposent généralement, outre la fourniture de l'interface POSIX, la possibilité d'utiliser des mécanismes propriétaires. Cela ne va pas dans le sens de la standardisation et ne facilite bien sûr pas le portage des applications.

- Les threads constituent l'unité de base d'exécution. En cela, ils sont tout à fait comparables aux tâches de l'exécutif généraliste. Le thread appartient à une structure d'accueil : le processus, qui lui permet de partager son espace d'adressage avec d'autres threads. Le processus peut ainsi être vu comme une coquille, une capsule qui protège ses éléments internes que sont les threads. L'ordonnancement des threads est un ordonnancement préemptif à priorité.

# Complément du cours : Notions sur les threads

***Pourquoi les threads ?***

Inconvénients du processus classique:

- Changement de contexte long (notamment pour les applications du type "temps réel" ou "multi média"),
- Pas de partage de mémoire « direct » (sinon shared memory, comm,… )
- Manque d'outils de synchronisation
- Interface rudimentaire (fork, exec,exit,wait)

Pourquoi le changement de contexte est-il "long »:

90% de ce temps est consacré à la gestion de la mémoire,

On introduit une nouvelle forme de processus : ceux-ci **partagent la mémoire, ainsi on résout aussi celui du changement de contexte «long»**

Ce nouveau type d'activité s'appelle un thread

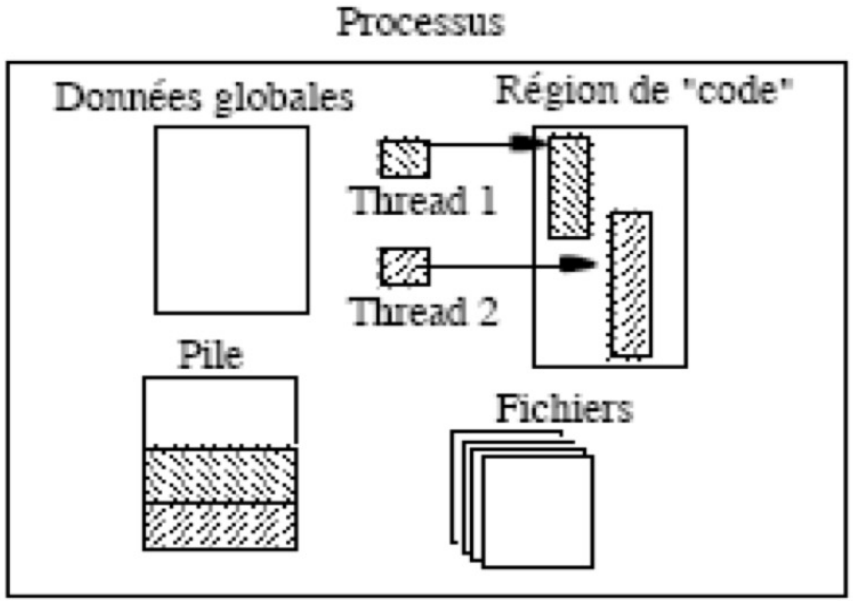
Les threads ne rendent pas obsolètes les processus classiques en particulier dans le cas où la séparation des espaces d'adressage entre applications s'impose, pour des raisons de sécurité, par exemple.

On peut donc envisager les threads comme des processus qui partagent toutes leurs ressources, sauf la pile et quelques registres (compteur ordinal, pointeur de pile).

# Complément du cours : Notions sur les threads

*Thread et processus*

Un **thread exécute une fonction. Donc, un thread :**
-ne "voit" qu'une partie de la région de code du processus qui l'héberge,
- dispose de sa propre pile pour implanter les variables locales,
- partage les données globales avec les autres threads.




Les ressources propres à un thread sont :

- un identificateur (le thread identifier, ou tid, équivalent du pid),
- une priorité,
- une configuration de registres, une pile
- un masque de signaux,
- d'éventuelles données privées.

Le nombre et l'identité des threads d'un processus sont invisibles depuis un autre processus

## *Implantation d'un thread*

Un thread peut être implanté :

– **au niveau du NOYAU, il est alors ordonnancé indépendamment du processus dans lequel il a été créé,**

– au niveau du PROCESSUS qui l'accueille, il accède alors au processeur dans les quanta alloués à ce processus

Dans le premier cas, le thread est l'unité d'ordonnancement. Quand un processus est lancé, on exécute en fait un thread associé à main.

La seconde méthode sollicite moins le noyau (pas d'appel à celui-ci pour les changements de contexte), mais :

- problème de gestion des E/S qui vont bloquer tout le processus (les appels systèmes bloquants doivent être redéfinis)
- pas de réel contrôle sur l'ordonnancement !

# Quelques références

- Systèmes d'exploitation temps réels
  - Y. Trinquet & J. P. Elloy / Techniques de l'Ingénieur
- Prototypage des applications temps réel embarquées
  - N. Du Lac / Techniques de l'Ingénieur
- Ordonnancement temps réel : Ordonnancement centralisé
  - F. Cottet et al. / Techniques de l'Ingénieur
- Introduction aux systèmes temps réel
  - C. Bonnet & I. Demeure : Editions Hermes
- Cours Langage Temps Réel
  - F. Prunet & D. Andreu