Informatique Embarquée
M2 / 2017

Ordonnancement, Temps-réel

Introduction

- Ordonnancement:
  - Choisir prochaine tâche qui pourra utiliser le processeur
    - Batch :-)
    - Temps-partagé
    - Temps-réel
- Systèmes embarqués:
  - Mono-processeur, temps-réel
- Voir :
  - http://etr05.loria.fr/slides/jeudi/ETR05-LG.pdf

Attendre un événement

- Solutions
  - Polling / Attente active :
    - `while (flag!=1) ;`
    - `/* flag vaut 1, l'événement attendu s'est produit ! */`
  - Alternative :
    - `while (flag!=1) nanosleep(1) ;`
  - Interruption
    - Quand l'événement est prêt envoi d'un signal matériel au processeur et exécution du code associé.

Interruptions

- Événement matériel signalant un changement d'état d'un périphérique, requérant un traitement de la part du système.
- Exemples:
  - Arrivée d'une trame Ethernet sur un contrôleur
    - Lire la trame, la passer à la pile TCP/IP... et dire au contrôleur que la trame été récupérée, ... permettre au contrôleur de signaler l'arrivée d'une nouvelle trame
  - Une demande d'écriture sur un disque est terminée
    - On n'a plus besoin de garder les données en mémoire, on peut demander un nouveau travail au disque (lecture/écriture d'un bloc)

Interruptions

- Événement déclenché par composant matériel
  - Fin d'entrée/sortie d'un périphérique
  - Échéance de temps terminée
- Signal physique envoyé au processeur de l'extérieur (contrôleur périphérique, horloge..)
  - Par l'intermédiaire d'un PIC
    - Programmable Interrupt Controller
  - Partageable ou non
- Événement asynchrone
  - Indépendant du programme courant
  - Masquable par le système (prise en compte différée jusqu'au démasquage)

Traitement Exceptions/Interruptions (1)

- Processeur suspend l'exécution en cours
- Sauvegarde état courant
  - PC, SP, registre d'état
  - Dans des registres dédiés ou dans une pile
- Sur certains processeurs (ARM), ou utilise des « shadow registers »
  - 2 jeux de registres, un par mode d'exécution

Traitement Exceptions/Interruptions (2)

- Charge nouveau contexte d'exécution :
  - Registre d'état avec mode « superviseur »
  - PC avec adresse mémoire spécifique
- Opérations de sauvegarde et de chargement
  - Non-interruptible
    - Sinon état non cohérent
    - Sauf par NMI, sur PowerPC par exemple
  - Si exception durant opérations (« double-faute »)
    - => arrêt CPU ou traitement spécial (Intel)

Exceptions/Interruptions "Vector Table"

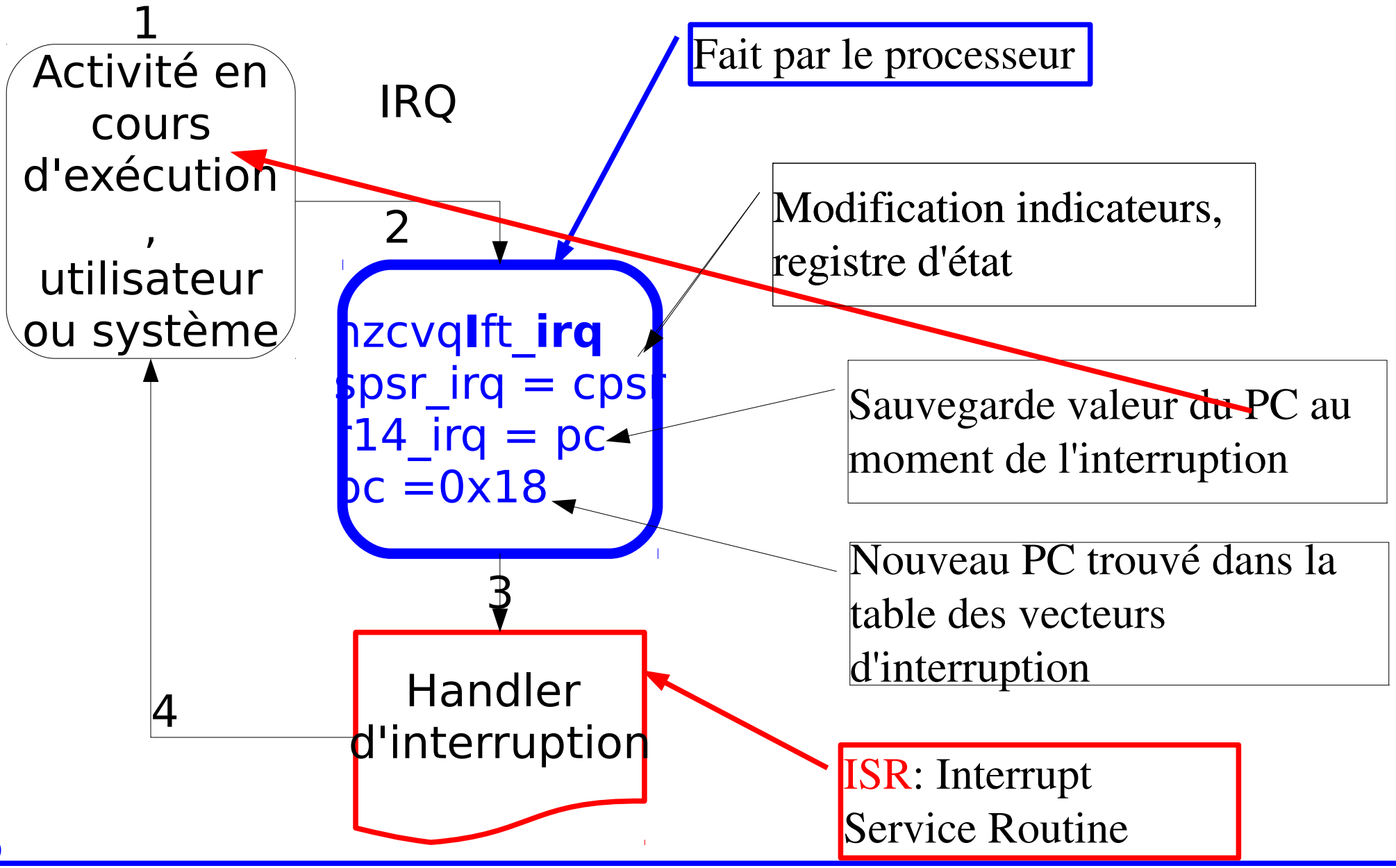
- Adresse de branchement
  - Dépend de l'exception/interruption
- Vers une table de vecteurs localisée
  - ARM: à adresse prédéfinie (0x00000000 ou 0xFFFFFFFF)
  - INTEL: dans un registre (IDTR)
- Contient une instruction de branchement
- Table de vecteurs initialisée par le système

Vector Table (ARM)

- La table de vecteur contient une instruction et non pas une adresse,
- En général: instruction de branchement, sur une fonction appropriée.

Reset	0X00000000
Undefined Instruction	0X00000004
Software Interrupt	0X00000008
Prefetch Abort	0X0000000C
Data Abort	0X00000010
Reserved	0X00000014
Interrupt request	0X00000018
Fast Interrupt Request	0X0000001C

Interruption / ARM

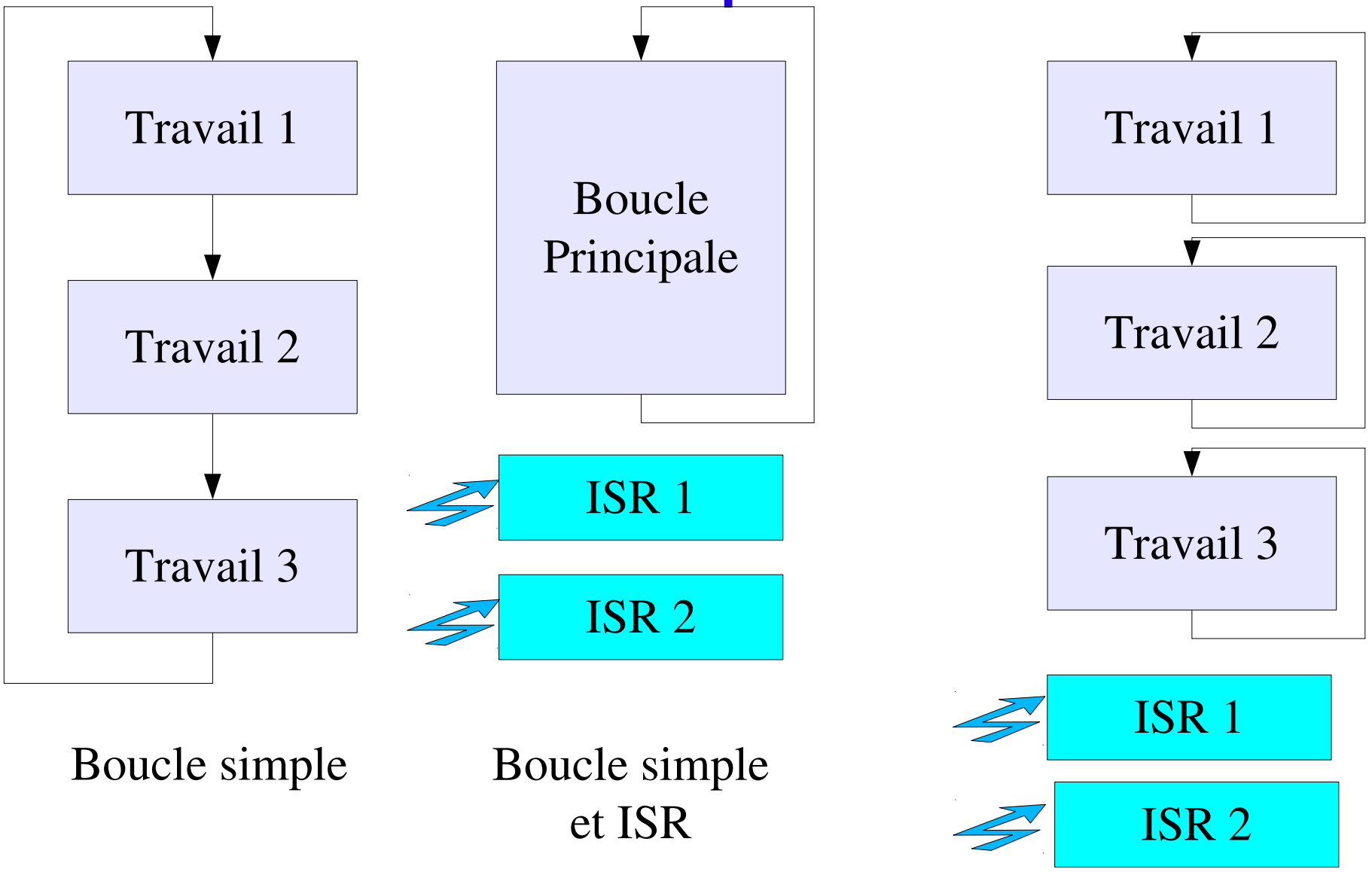


Signaux, en mode utilisateur

```
void wakeup(int signb)
{
  printf("Dring!\n");
  alarm(1);
}
main()
{
  act.sa_handler=wakeup;
  sigaction(SIGALRM, &act, NULL);
  alarm(1); while(1) pause();
}
```

```
#./sig
Dring!
Dring!
Dring!
Dring!
Dring!
```

Application Temps-réel architectures possibles

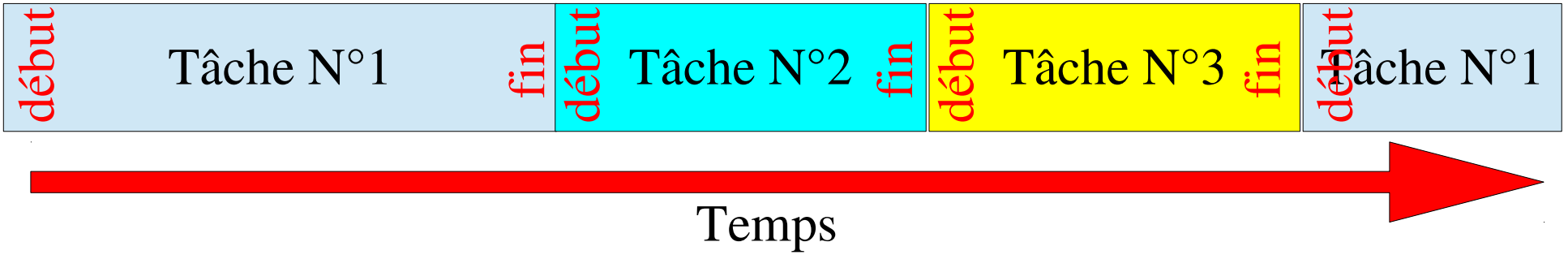


Cf http://www.embedded.com/design/programming-languages-and-tools/4434261/A-multitasking-kernel-in-as-close-as-you-can-get-to-one-line-of-code

Ordonnancement

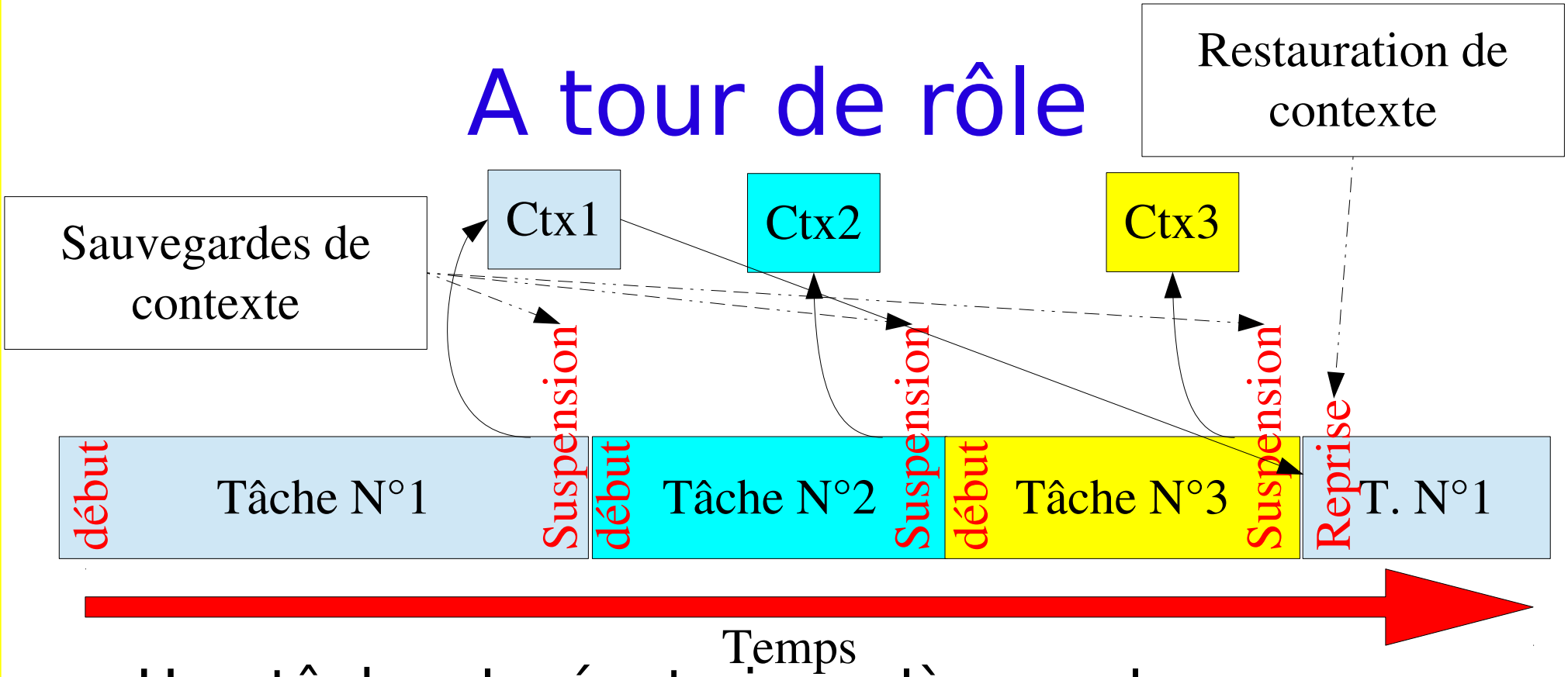
- Il y a beaucoup de possibilités dans les politiques possibles et dans leurs implémentations.
- Il n'est pas forcément nécessaire d'avoir un OS pour procéder à un ordonnancement
  - Une librairie peut suffire
  - On peut même envisager que ce soit fait au niveau matériel.
  - On peut avoir un OS

Run to completion



- On permet aux tâches de s'exécuter jusqu'à leur terminaison
  - Similaire à une grande boucle enchaînant les tâches les unes après les autres
  - Du coup, on réinitialise les tâches à chaque invocation :-(

A tour de rôle



- Une tâche s'exécute jusqu'à ce qu'...
  - Elle fasse un appel bloquant,
  - Elle relâche volontairement le processeur (yield), etc
- Elle reprend son exécution là où elle l'avait laissée

Sauvegarde, restauration de contextes

- De manière naïve :
  - l'ensemble des registres (des valeurs contenues dans) au moment où on veut suspendre l'exécution
    - Sommet de pile, compteur ordinal, etc,...
    - Ex : Task Descriptor processeurs x86
  - Le reste de l'état est contenu dans la mémoire et donc préservé.

Sauvegarde, restauration de contextes

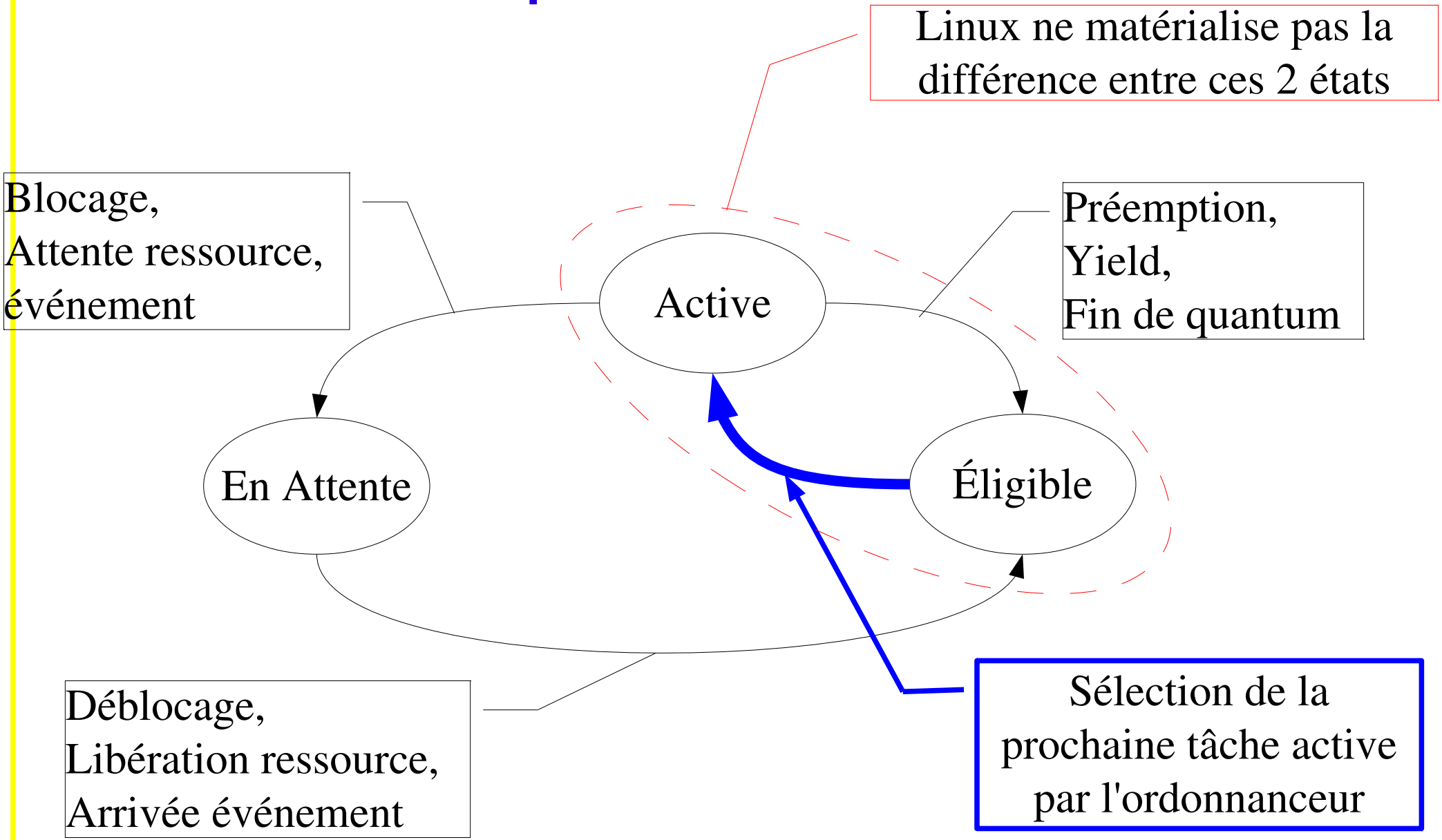
- Dans les faits, on peut alléger la sauvegarde
  - Registres ne contenant rien d'utile, déjà sauvegardés
  - Ou ne sauvegarder qu'au moment où on aura besoin du registre...

Sauvegarde, restauration mode utilisateur !

```
if (sigsetjmp(buf, sigs) == 0 ) {
    /* First time */
    siglongmp(buf, 1)
} else {
    /* Resuming out of a longjmp ! */
}
```

- Voir aussi `getcontext`, `setcontext` et Cie.

États (simplifiés) d'une tâche



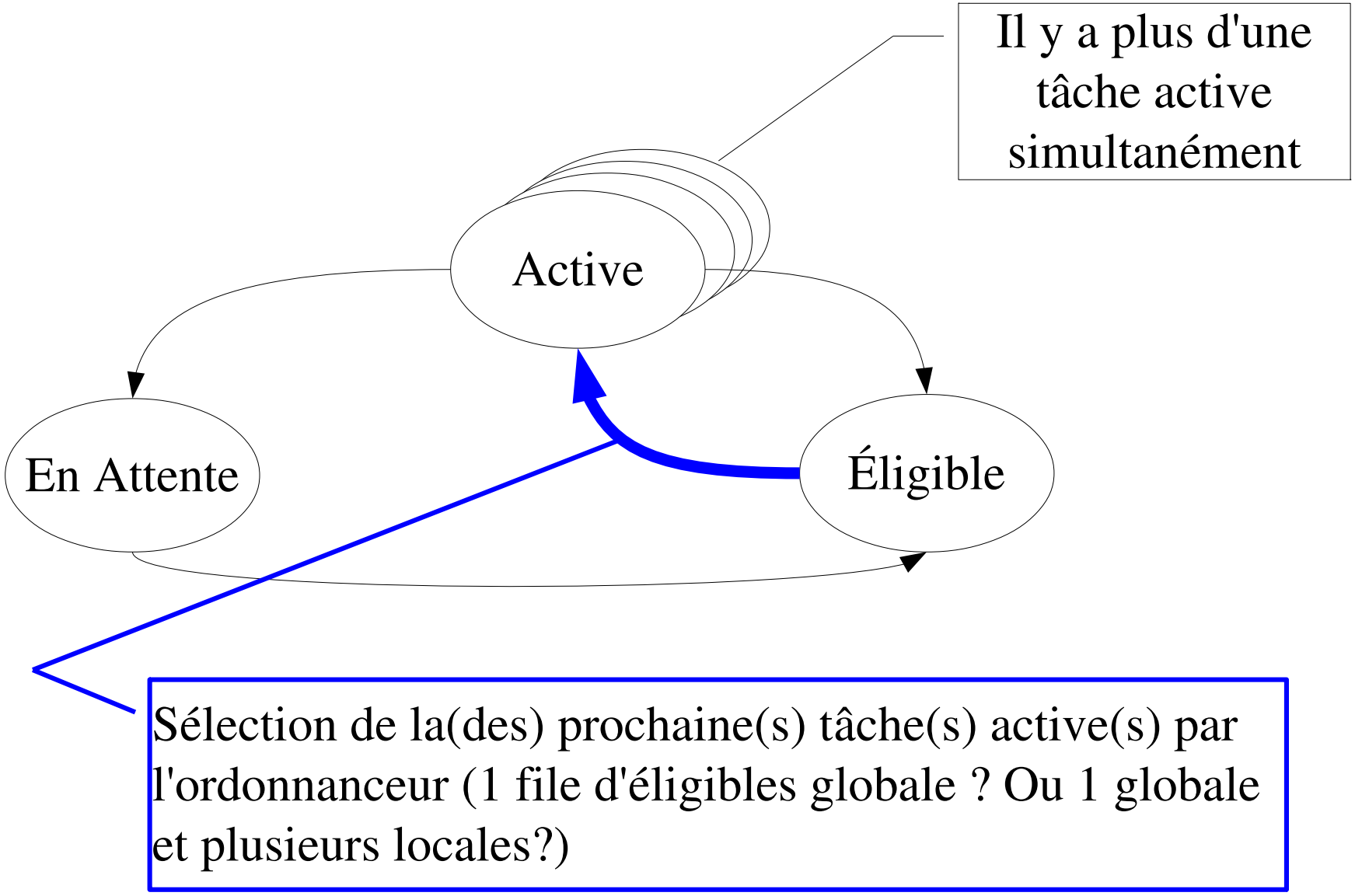
Ordonnancement, dimensions

- Politique (Étape N°1):
  - Quel est la prochaine tâche à obtenir le processeur ?
  - Quels critères pour la choisir ?
- Mécanique (Étape N°2) :
  - Sauvegarder le contexte de la tâche qui perd le processeur
  - Restaurer le contexte de la tâche qui obtient le processeur
- Temporelle :
  - Quand l'ordonnancement peut-il être fait ?

Multiprocesseurs

- Plusieurs processeurs sur une même machine
  - Mémoire partagée entre les processeurs
  - Temps d'accès uniforme à la mémoire ou non (NUMA)
- Plusieurs puces ou plusieurs cœurs sur une même puce, ou plusieurs (hyper) « threads » matérielles sur un même cœur

États (simplifiés) d'une tâche



« Attacher » une tâche à un processeur

- On parle d'affinité (CPU affinity)
- Commande :
  - `taskset -c 3-4 –pid 17915`
- Appel système
  - `int sched_setaffinity(pid_t pid, size_t cpusetsize,cpu_set_t *mask);`
  - `sched_getaffinity(...)`

Préemption

- Arrêter, suspendre momentanément l'exécution d'une tâche au profit d'une autre. Exemples:
  - Répondre au téléphone alors que l'on est en train de faire cours!
  - Répondre à une question au milieu d'une explication
- On reprendra l'exécution de la tâche « préemptée » plus tard.
- Pour qu'une préemption ait lieu, il faut un événement extérieur:
  - Fin d'entrée/sortie, interruption horloge,...

Préemption

- Il y a des moments où le fait d'être préempté pourrait conduire à des incohérences
  - Mise à jour / Lecture de variables globales accédées par la tâche (le code) exécuté pendant la préemption.
  - Cas usuel: mise à jour d'une liste par une tâche préemptée, et utilisation de cette liste par le code exécuté au cours de la préemption.
- Il faut donc pouvoir se protéger contre la préemption si nécessaire.
- Introduction d'une latence (d'un délai) dans la prise en compte de l’événement

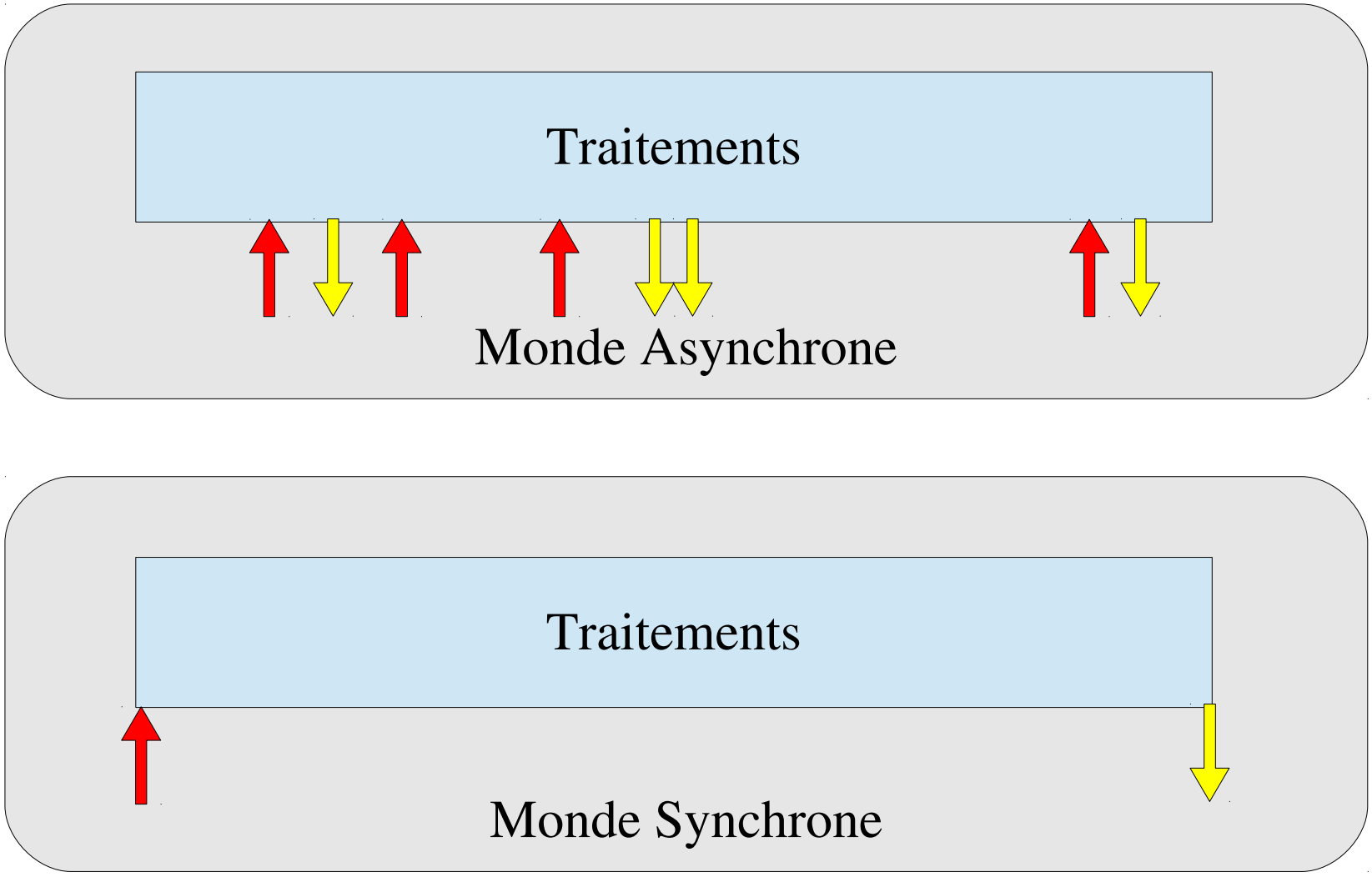
Ordonnancement Temps-Réel

- Tâches connues à l'avance (système fermé)
  - « dimensionnement » du système
  - Séquençage manuel
- Approche « pire cas » (worst case)
  - Pas basé sur une moyenne
  - Pas basé sur tests ou estimation
  - Identifier les cas pires et configurer le système pour qu'il fonctionne dans ces cas
    - Il fonctionnera donc dans n'importe quel autre cas

Synchrone / asynchrone

- Deux types de systèmes:
  - Synchrone:
    - Existence d'une base de temps commune,
    - Les événements n'arrivent pas n'importe quand,
  - Asynchrone:
    - Pas d'hypothèse sur les instants où les événements peuvent se produire

Synchrone / Asynchrone



Déterminisme

- Pouvoir garantir que le système respectera ses spécifications, notamment temporelles, pendant sa durée de vie
  - Ré-exécution donne des résultats identiques
- Méthodologie
  - Spécifications => problèmes à résoudre
  - Déterminer les cas pires
  - Conditions de faisabilité (CF)
  - Déterminer valeurs numériques CF
  - Vérifier

Caractéristiques

- Modèles:
  - de tâches
  - de contraintes temporelles
  - d'ordonnancement

Tâches

- Tâches concrètes:
  - on impose un scénario d’activation particulier des tâches
- Tâches non concrètes:
  - on ne connaît pas à priori les instants d’activation (ou de première activation) des tâches.

Tâches

- Durée (pire cas) d'une tâche τ_i: C_i
  - Tâche seule sans interruption OS
  - Par analyse ou par mesure
- (Pire) temps de réponse d'une tâche τ_i: R_i
  - Temps entre demande activation et réponse
  - Prend en compte: délai dans exécution induit par les autres tâches
- $R_i \geq C_i$

Tâches

- Gigue
  - Délai entre arrivée d'un événement et sa prise en compte
  - Gigue de τ_i: G_i

Tâches

- Périodiques
  - Demande activation tous les T_i (la période)
- Sporadiques
  - Demande activation variable ≥ T_i
- Apériodiques
  - 1 activation pendant la vie du système
- Fenêtrées
  - Au plus n_i sur une durée « glissante » W_i

Tâches

- Dépendances entre tâches
  - Tâches indépendantes
  - Tâche simple: code séquentiel
  - Arbre
  - Client / Serveur
  - Graphe

Contraintes Temporelles

- Échéance de terminaison au plus tard
  - Tâche τ_i activée à instant t_i doit être terminée au plus tard à instant $t_i + D_i$
  - D_i : échéance relative
  - $t_i + D_i$: échéance absolue
- Échéance de démarrage au plus tard

Contraintes Temporelles

- Déterminer les Conditions de Faisabilité
- Pire temps de réponse:
  - $\forall$ i = 1,.. n $R_i \leq D_i$
    - Temps de réponse ≤ échéance
  - Charge du processeur $U = \Sigma_{i=1,n}(C_i/T_i) \leq 1$
  - En préemptif
    - $U \leq n(2^{1/n} - 1)$
    - Difficile à utiliser dans la réalité
- Il existe d'autres approches

Ordonnanceurs

- Time Driven:
  - Ordonnanceur détermine quand il est invoqué
    - En général: invocation périodique fréquence: T_{tick}
- Event Driven: (celui que l'on va considérer)
  - Ordonnanceur invoqué sur réception d'un événement

Ordonnanceurs

- Ordonnancement par priorités:
- Fixes (FP)
  - La priorité d'une tâche est identique pour toutes ses activations (Rate Monotonic, Deadline Monotonic)
- Dynamiques (DP)
  - La priorité d'une tâche est calculée pour chacune de ses activations (Earliest Deadline First)
- Mixtes

Ordonnanceurs

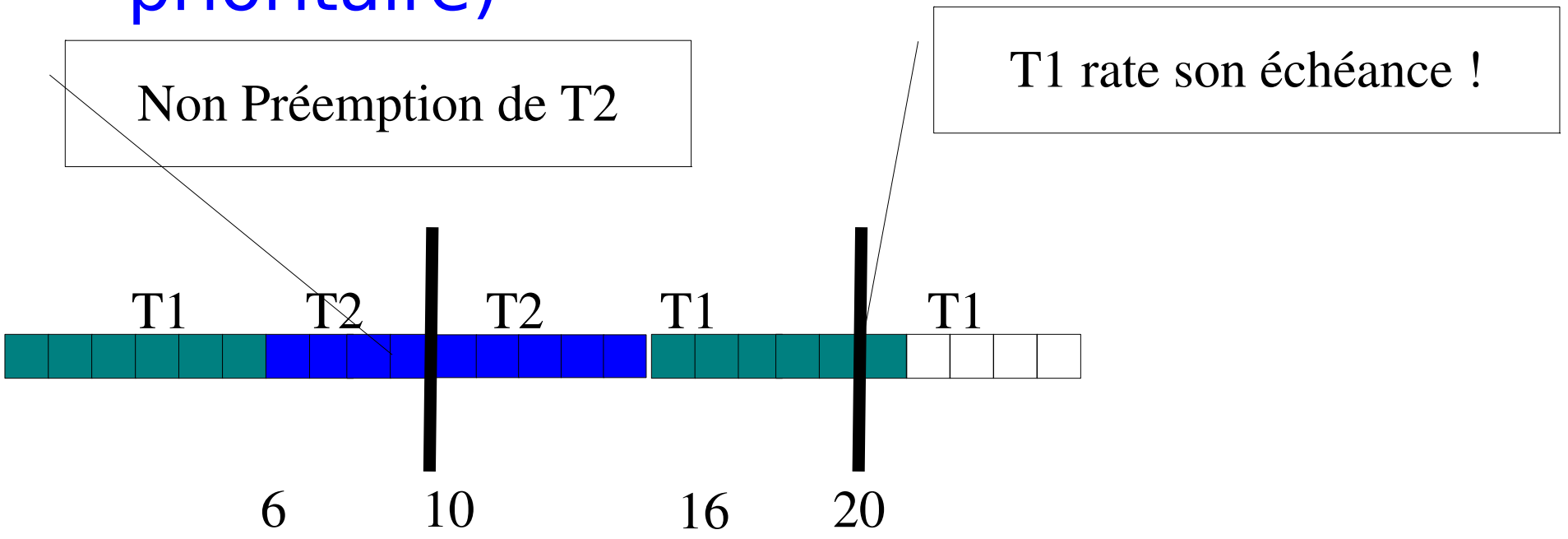
- Un algorithme d’ordonnancement est optimal si il trouve toujours une solution pour respecter les contraintes temporelles des tâches lorsqu'il en existe une.
- Si un algorithme optimal ne trouve pas de solution alors il n'en existe pas.
- On peut trouver une solution d'ordonnancement faisable avec un algorithme DP alors qu'il n'existe pas de solution avec FP.

Rate Monotonic

- Priorité déterminée en fonction de la période
- Plus la période est petite, plus la priorité est élevée
- Optimal
  - pour tâches périodiques ou sporadiques
  - Avec ordonnancement préemptif

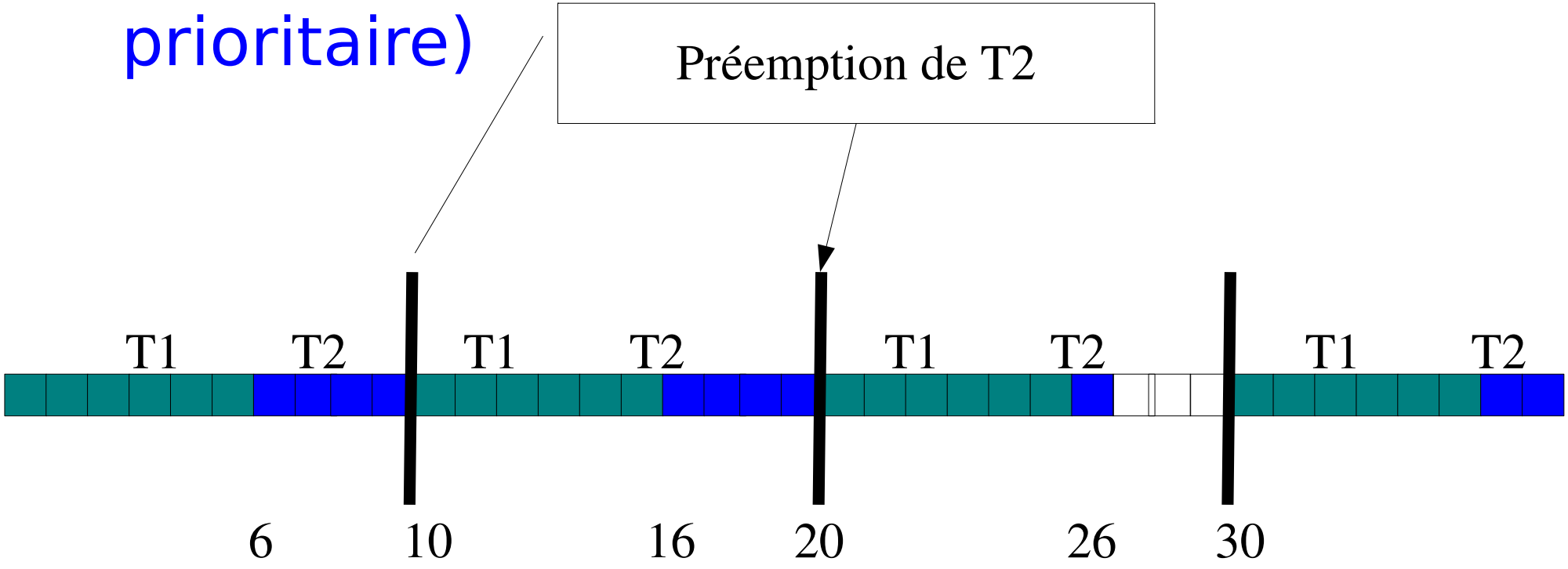
Exemple Non-Préemptif

- T1 C1=6, P1=10, Prio1=0 (la plus prioritaire)
- T2 C2=9, P2=30, Prio2=1 (moins prioritaire)



Exemple Préemptif

- T1 C1=6, P1=10, Prio1=0 (la plus prioritaire)
- T2 C2=9, P2=30, Prio2=1 (moins prioritaire)



RM Exemple

- Supposons 3 tâches
  - Sur un système où la priorité la plus élevée est 0
  - Affichage d'un indicateur toute les 100 ms
    - Période 100, Charge 20, Priorité 0
  - Lecture d'un capteur toutes les 250 ms
    - Période 250, Charge 50, Priorité 1
  - Surveillance toutes les 500 ms
    - Période 500, Charge 150, Priorité 2

RM Exemple

- U = 20/100 + 50/250 + 150/500 = 0.7
- Limite = $3(2^{1/3} - 1) = 0.779$
- U ≤ Limite
  - OK !

FIFO

- *Pour un niveau de priorité donné,* les tâches traitées dans leur ordre de demande d'activation
- Minimise le temps de réponse de l'ensemble des tâches (overhead de l'ordonnanceur minimum)
- Temps de réponse maximum unique commun
- Pas optimal pour des tâches ayant des échéances différentes

Round Robin (Tourniquet)

- *Pour un niveau de priorité donné,* le processeur est attribué à tour de rôle aux tâches actives
- Équitable
- Pas optimal pour tâches temps-réel
  - Mais RR peut trouver une solution faisable sur certains problèmes impossibles en FP

Earliest Deadline First

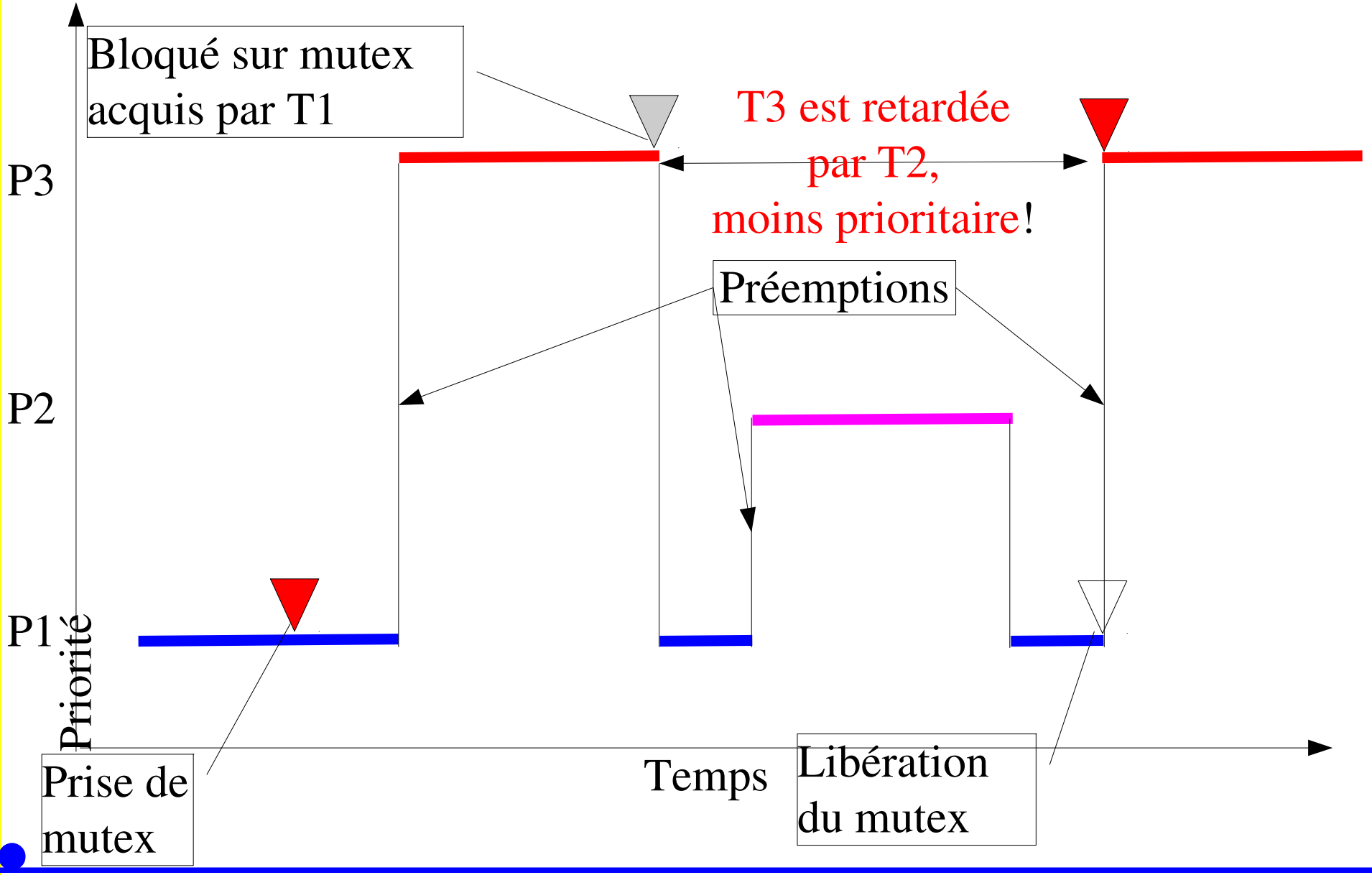
- Priorité tâche $\equiv_i$ activée à t_i : $t_i + D_i$
- Optimal en préemptif et en non préemptif non concret
- Très performant
- Instable en cas de surcharge: (effet avalanche)
  - Basculer vers autre ordonnancement,
  - Éliminer des tâches
  - Dégrader le service

Linux et EDF

```
struct sched_param {
    ….
    u64 sched_runtime;
    u64 sched_deadline;
    u64 sched_period;
} ;
```

runtime ≤ deadlline ≤ period

Inversion de Priorité



Inversion de Priorité

- Pour éviter ce genre de situation, changer momentanément la priorité de la tâche T1
  - Au moment où elle acquiert le mutex, lui affecter une priorité associée à la ressource protégée. T1 retrouvera sa priorité usuelle à la libération du mutex. « Priority ceiling protocol »
  - Au moment où T3 se bloque sur le mutex verrouillé par T1, on fait hériter T1 de la priorité de T3, jusqu'à la libération du mutex par T1... « Priortiy inheritance protocol ». Complexifie l'ordonnanceur. Transitivité!

Linux et le temps-réel

- Linux supporte 3 types de processus / threads
  - SCHED_NORMAL : Conventionnels (temps-partagés)
  - Temps-réel:
    - SCHED_FIFO: Priorité fixe (First-In First-Out)
    - SCHED_RR: Priorité fixe avec quantum de temps à priorité égale (Round-Robin)
- Les threads FIFO et RR sont plus prioritaires que les autres, et partagent la même plage de priorités (0..99). pas affectées par nice.
- Les threads temps partagé ont une priorité 100...139, affectée par nice. Le noyau utilise en fait une priorité dynamique (utilisation cpu, temps d'attente..)

Linux et le temps-réel

- Pour « créer » un processus temps-réel, il faut être super-utilisateur.
  - Pourquoi?
- Utilisation des appels POSIX pthread
  - pthread_attribute au moment de la création de la thread
  - Changement dynamique après la création de la thread.

Appels POSIX

```
struct sched_param param = {.sched_priority = 52 };
pthread_attr_t attr;


pthread_attr_init(&attr);
pthread_attr_setinheritsched(&attr,
PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&attr, SCHED_FIFO);
pthread_attr_setschedparam(&attr, &param);


pthread_create(&th, &attr, &fn, NULL);
```

Linux et le Temps-réel

- Linux est-il temps -réel?
  - Comprendre:
  - Linux est-il adapté pour permettre à des applications ayant des contraintes temps-réel de s'exécuter sans problème.
- Caractéristiques attendues
  - Déterminisme
  - Faible latence, « préempatbilité »
  - Ordonnanceur approprié
  - Différents services (timers, synchronisation, communication...)

Linux et le Temps-réel

- Sources d'informations
  - Building Embedded Linux Systems
    - K. Yaghmour, J. Masters, G. Ben-Yossef, P. Gerum
  - http://elinux.org/Real_Time
  - http://rt.wiki.kernel.org/

Linux, Préemption

- Possibilité de permettre à une tâche prioritaire de démarrer immédiatement son exécution
- A l'opposé, imposer d'attendre que l'exécution en cours atteigne un point où l'on peut procéder à in ordonnancement
- Linux 2.4:
  - Peu de points de préemption... Exécution en mode système ressemble à une « grosse section critique », ordonnancement lors du retour vers le mode utilisateur

Problèmes à résoudre

- Points d'ordonnancement internes au système
- Coût de l'ordonnancement
  - Fixe ou proportionnel au nombre de processus/threads éligibles
- Traitement des interruptions

Linux 2.6

- Incorporation de certaines des modifications contenue par le patch PREEMPT-RT
  - Ordonnanceur O(1)
  - Amélioration des points d'ordonnancement internes
  - High-Precision Timers
- Reste entre autres:
  - Traitement des interruptions

Améliorer la Préemption

- Minimiser:
  - La durée des sections critiques (pendant lesquelles les interruption sont masquées par le code de base)
    - Réduit la latence de prise en compte
  - Le traitement effectué sous interruption
    - Pendant que l'on traite une interruption, soit toutes les interruptions sont masquées, soit l'interruption que l'on traite est masquée.
    - Ex: si le traitement d'un interruption horloge prend plus d'un « tick », le système ne sera pas à l'heure et inutilisable.
- Permettre un ordonnancement en retour d'interruption

Linux interruptions

- SoftIRQ:
  - Faire le minimum dans l'ISR
  - Faire le travail réel et « long » après avoir acquitté l'interruption, avant de reprendre le travail interrompu
  - Possiblement faire faire ce travail par une thread dédiée `ksoftirqd`, haute priorité
- Tasklet
  - On peut avoir plusieurs softIRQ simultanément (1/CPU)
  - On a une seule tasklet (pour un IRQ donnée), CPU variable

Linux, PREEMPT-RT

- Permettre à des threads définies par l'utilisateur de s'exécuter à des priorités plus élevées que le traitement des interruptions.
- Crée un thread « noyau » par niveau d'interruption.
- L'administrateur système peut ajuster les priorités.
- Augmente la charge du système: passage obligatoire par des threads, coût d'ordonnancement.

Linux, PREEMPT-RT

- Attention! Un marteau est un très bon outil, à condition de savoir s'en servir et de ne pas se taper sur les doigts!
- Le noyau 2.4 n'est pas préemptif,
  - « Pas possible » de tourner des applications TR
  - Modifications appropriées maintenues ailleurs
- Certaines des modifications ont été incorporées au noyau 2.6
  - Reste toujours la nécessité d'un patch suivant les besoins applicatifs

Linux 2.6

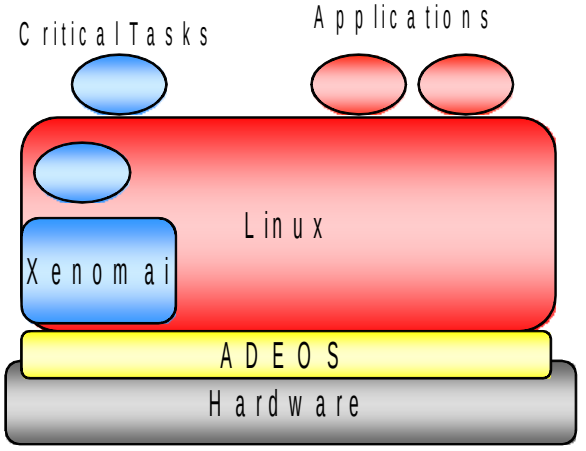
- Ordonnanceur en O(1)
  - Remplacé par CFS depuis 2.6.23
- Offre 3 modes de préemption (config)
  - Défaut (pas de préemption)
  - Préemption Volontaire
  - Préemption « involontaire »
    - Encore insuffisant (y compris pour audio)
- 2.6.18 : héritage de priorité
- 2.6.21: High-resolution timers (Posix) ~1µ sec.

Patch PREEMPT-RT

- Introduit un niveau de preemption « complète »
- Permet des latences encore plus courtes ~100µsec
- Impact global sur performances
  - On passe plus temps en changement de contexte
- Problèmes:
  - Pas dans l'arbre référence (résistance!)
  - Pilotes de périphériques à adapter (quid si binaire?)

Adeos/Xenomai

- Solution au problème du temps-réel dur:
  - Offrir un environnement RT « à côté » de Linux
- Exemples
  - RTLinux (FSMLabs, WinRiver)
  - RTAI
  - Adeos /Xenomai
  - Solutions de virtualisation



Adeos /Xenomai

- ADEOS / XENOMAI
  - Module noyau Linux chargé après démarrage du noyau Linux
- ADEOS
  - « Prend le contrôle » du processeur
  - Intercepte exceptions / interruptions
  - Distribue les interruptions à des « domaines » par ordre de priorité. (virtualisation des interruptions)
  - Linux est un domaine avec une priorité faible

Adeos / Xenomai

- Temps de latence
  - Déterminé par ADEOS lui-même
  - Temps de « préempter » Linux
    - Adeos préempte Linux # Linux préempte Linux
- Domaine prioritaire : Xenomai
  - Exécutif TR
    - Tâches en mode utilisateur / superviseur
    - Différentes interfaces (skins) disponibles pour favoriser migration depuis des OS TR existants

Sauvegarde, restauration de contextes

- De manière naïve :
  - l'ensemble des registres (des valeurs contenues dans) au moment où on veut suspendre l'exécution
    - Sommet de pile, compteur ordinal, etc,...
    - Ex : Task Descriptor processeurs x86
  - Le reste de l'état est contenu dans la mémoire et donc préservé.
- Dans les faits, on peut alléger la sauvegarde
  - Registres ne contenant rien d'utile, déjà sauvegardés
  - Ou ne sauvegarder qu'au moment où on aura besoin du registre...