Introduction à la programmation micro-contrôleur

avec un OS temps réel

Julien Deantoni

Contenu du cours

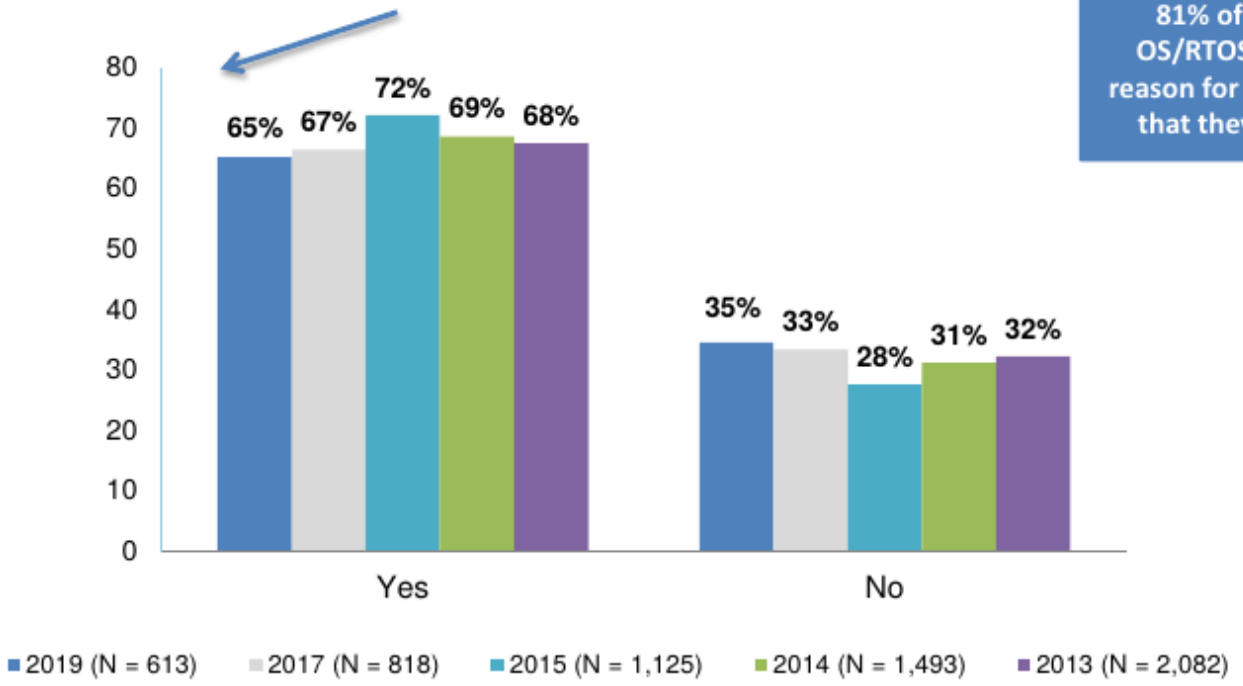
- Généralités
  - Les systèmes considérés
  - Le développement de tels systèmes
- Programmation sans OS
- *Programmation avec un OS temps réel*

Contenu du cours

- Qu'est-ce qu'un OS temps réel (RTOS)
  - RTOS vs GPOS
  - Différents RTOS...
  - Les objets *communs* d'un RTOS
  - Les objets *haut niveau* d'un RTOS
  - Implémentation d'un RTOS
- Programmation avec un OS temps réel
  - Comment utiliser les objets d'un RTOS ?
- Mise en Oeuvre
  - freeRTOS

Market Study

Does your current embedded project use an operating system, RTOS, kernel, software executive, or scheduler of any kind?



81% of those not using OS/RTOSes, said the main reason for NOT using is simply that they are not needed.

RTOS vs GPOS

- **L'essence de la discorde : Prédictabilité[1]**
  - RTOS
    - Multi-tâche / thread depuis toujours
    - Ses actions sont déterministes...
    - **...et interruptibles**
  - GPOS
    - Multi-tâche seulement *récemment*
    - Prends la main sur les tâches en cours...
    - Pour une durée indéterminée et non interruptible

http://embedded-computing.com/pdfs/QNX.Jan05.pdf

1 : Ou Déterminisme...

RTOS vs GPOS

- **L'essence de la discorde : Prédictabilité**[1]
  - RTOS
    - Multi-tâche / thread depuis toujours
    - Ses actions sont déterministes...
    - **...et interruptibles**
  - GPOS
    - Multi-tâche seulement *récemment*
    - Prends la main sur les tâches en cours...
    - Pour une durée indéterminée et non interruptible

Un RTOS n'est pas équitable (*fair*)

http://embedded-computing.com/pdfs/QNX.Jan05.pdf

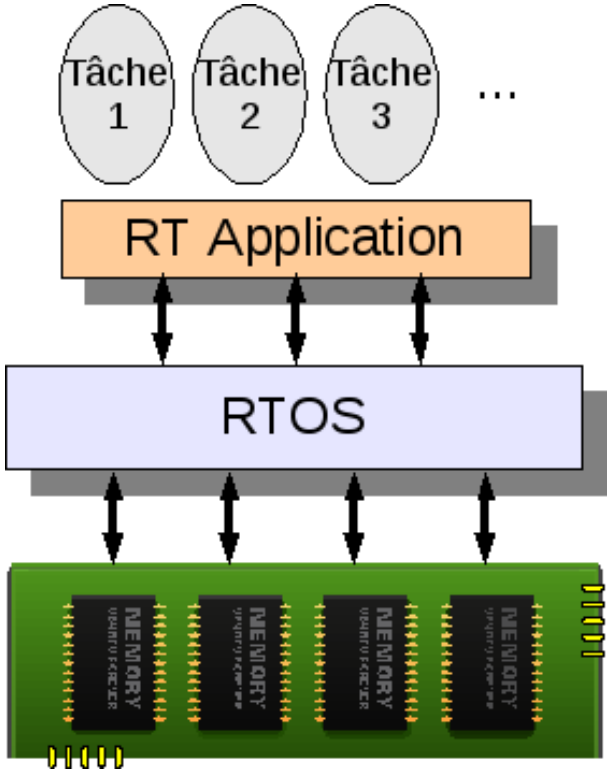
1 : Ou Déterminisme...

Contenu du cours

- **Qu'est-ce qu'un OS temps réel (RTOS)**
  - RTOS vs GPOS
  - **Différents RTOS...**
  - Les objets *communs* d'un RTOS
  - Les objets *haut niveau* d'un RTOS
  - Implémentation d'un RTOS
- Programmation avec un OS temps réel
  - Comment utiliser les objets d'un RTOS ?
- Mise en Oeuvre

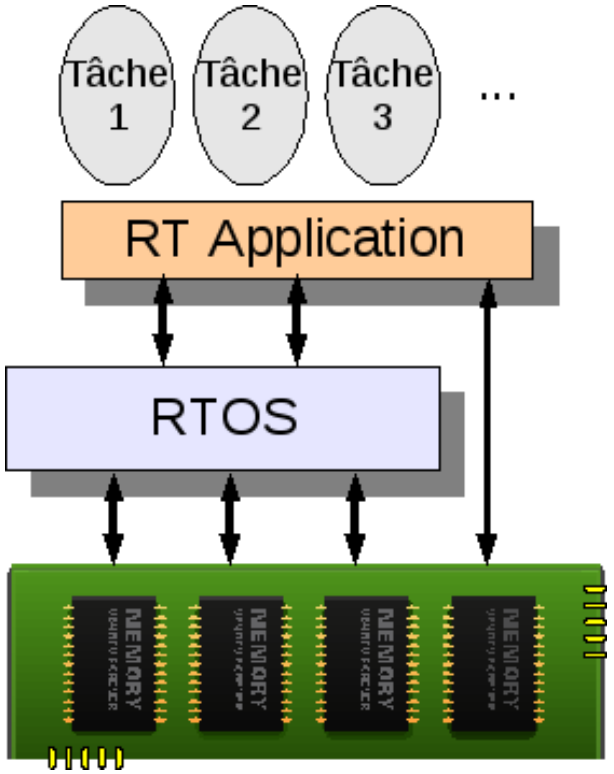
Différents RTOS Le classique (1)

- Complétement développé au dessus du matériel, il fournit les objets standards d'un RTOS
- Utilisation de l'API du RTOS par l'application
- Tout est intercepté par le RTOS (même les interruptions)



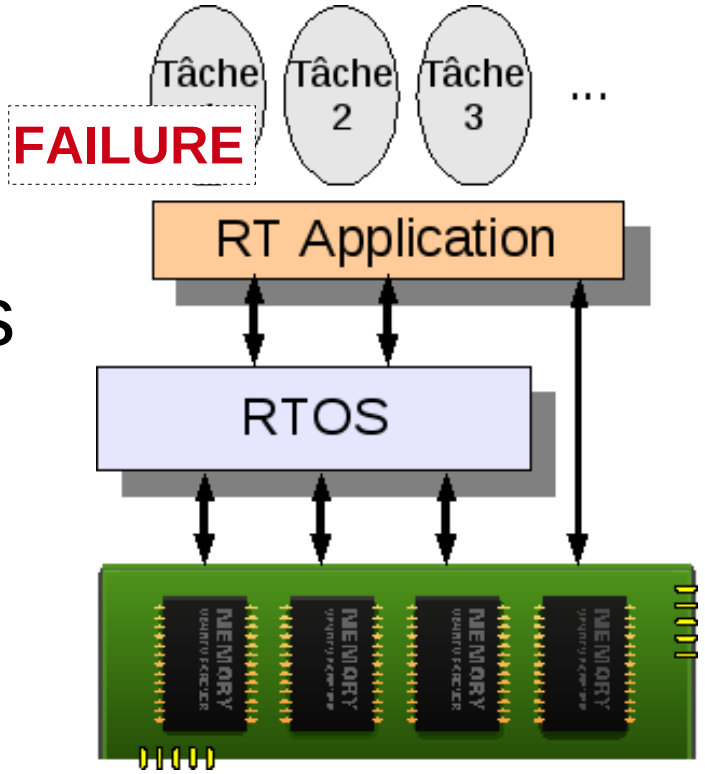
Différents RTOS Le classique (1)

- Complétement développé au dessus du matériel, il fournit les objets standards d'un RTOS
- Utilisation de l'API du RTOS par l'application
- Laisse à l'application certains accès au matériel



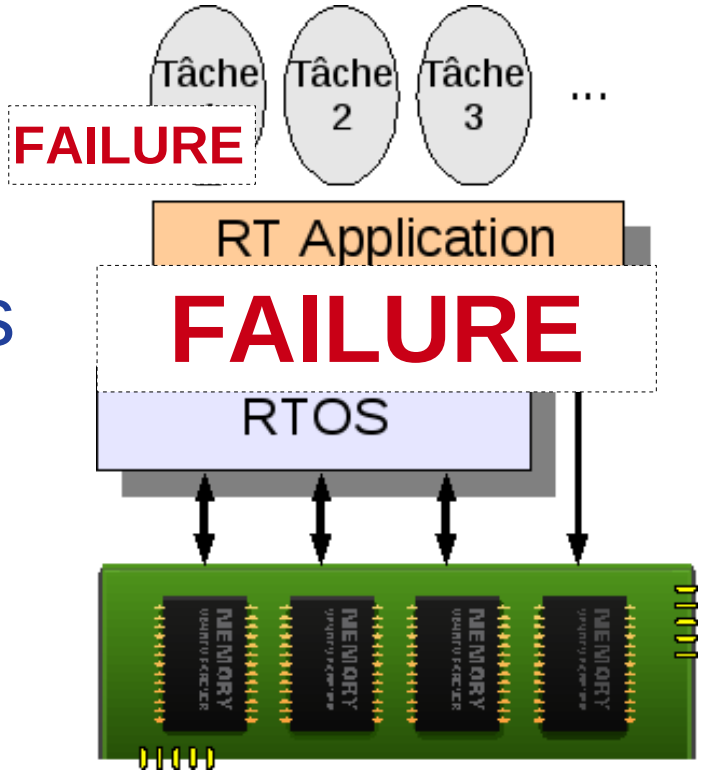
Différents RTOS Le classique (1)

- Complétement développé au dessus du matériel, il fournit les objets standards d'un RTOS
- Utilisation de l'API du RTOS par l'application
- Laisse à l'application certains accès au matériel



Différents RTOS Le classique (1)

- Complétement développé au dessus du matériel, il fournit les objets standards d'un RTOS
- Utilisation de l'API du RTOS par l'application
- Laisse à l'application certains accès au matériel

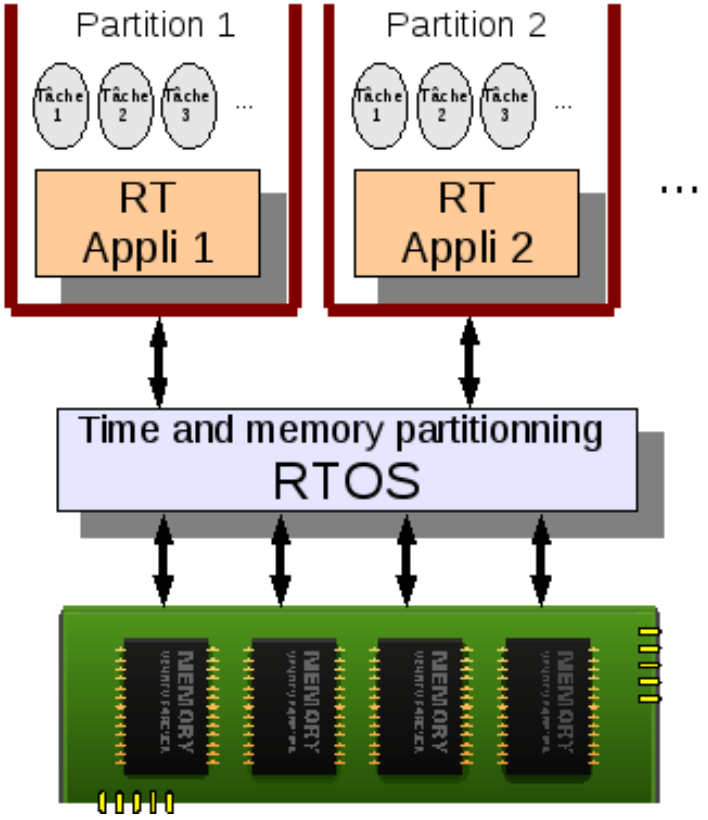


Différents RTOS safety-critical

- Permet d'isoler temporellement et en mémoire des applications différentes

- Utilisation de l'API du RTOS par l'application

- S'assure que le partitionnement est respecté

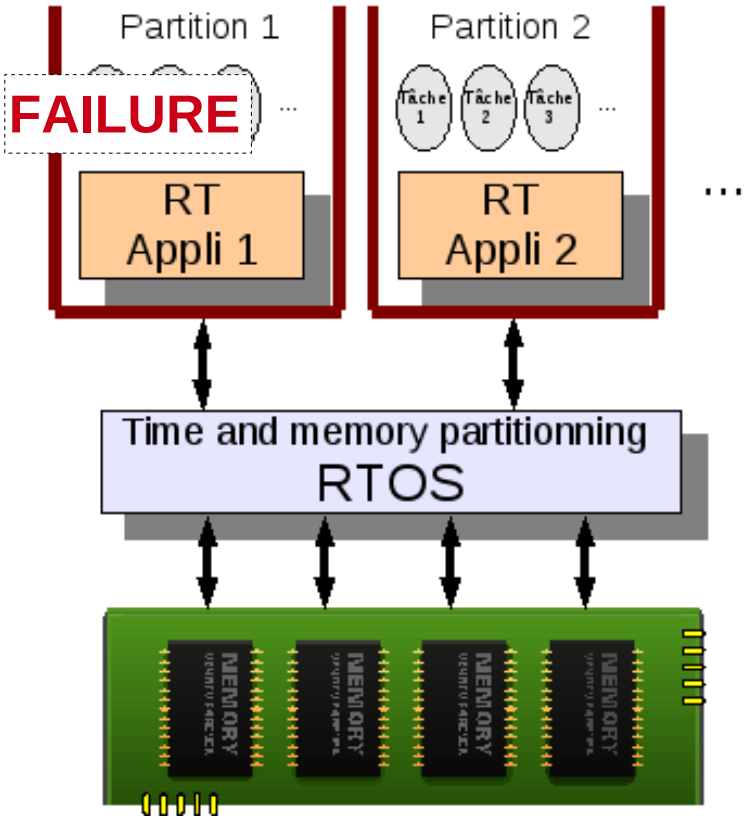


Différents RTOS

safety-critical

- Permet d'isoler temporellement et en mémoire des applications différentes
- Utilisation de l'API du RTOS par l'application
- S'assure que le partitionnement est respecté

→ Évite la propagation des pannes

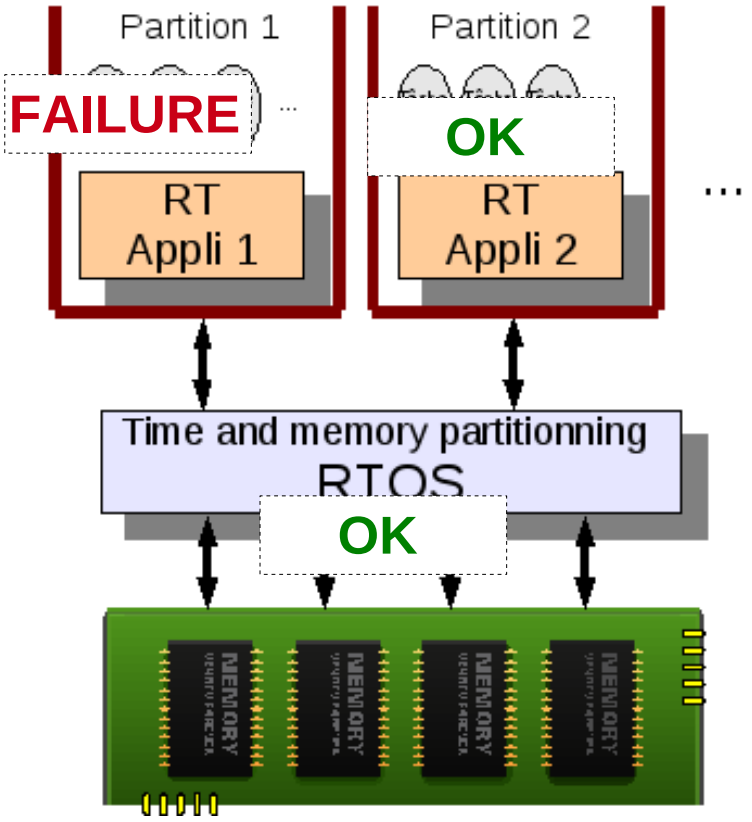


Différents RTOS

safety-critical

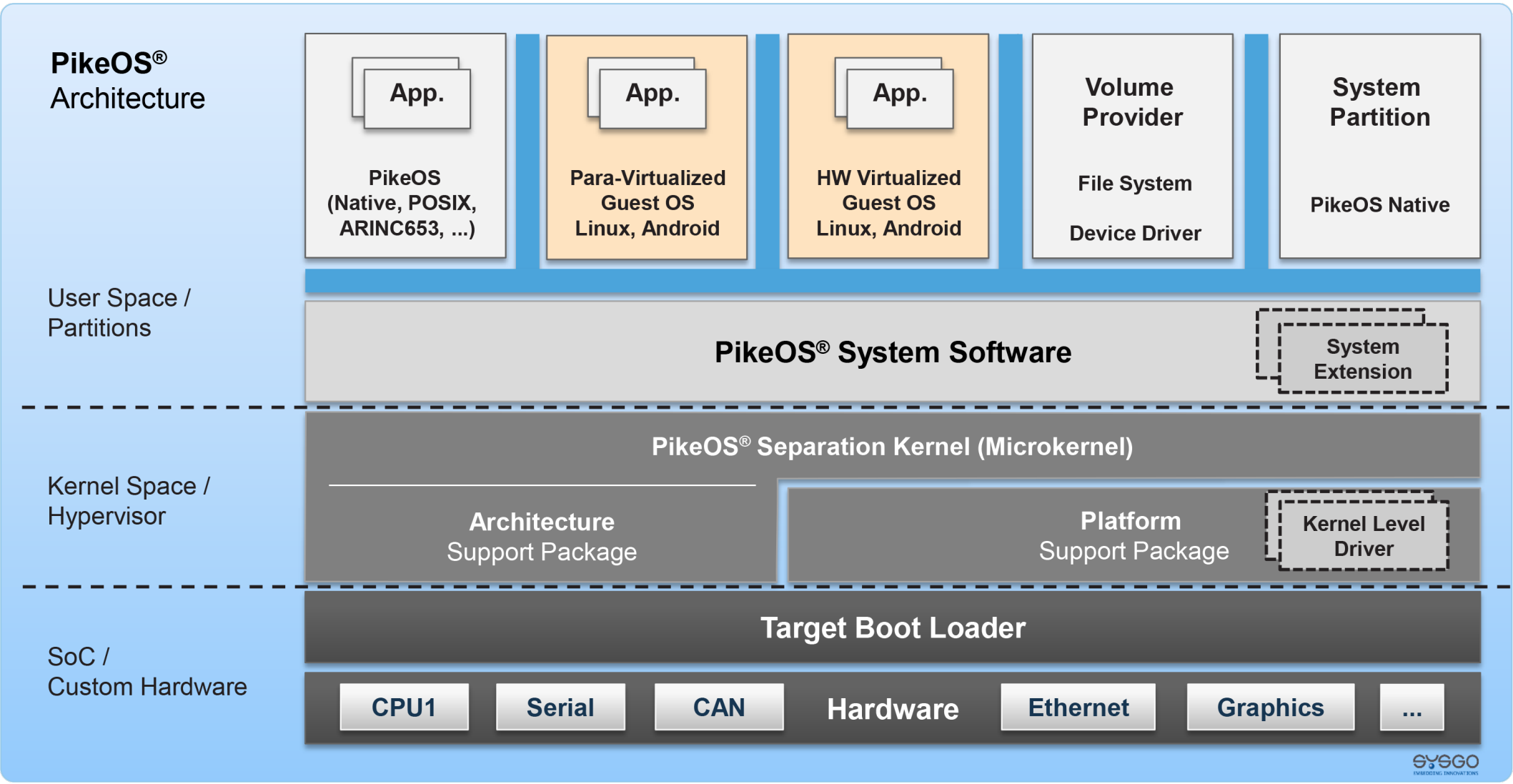
- Permet d'isoler temporellement et en mémoire des applications différentes
- Utilisation de l'API du RTOS par l'application
- S'assure que le partitionnement est respecté

→ Évite la propagation des pannes



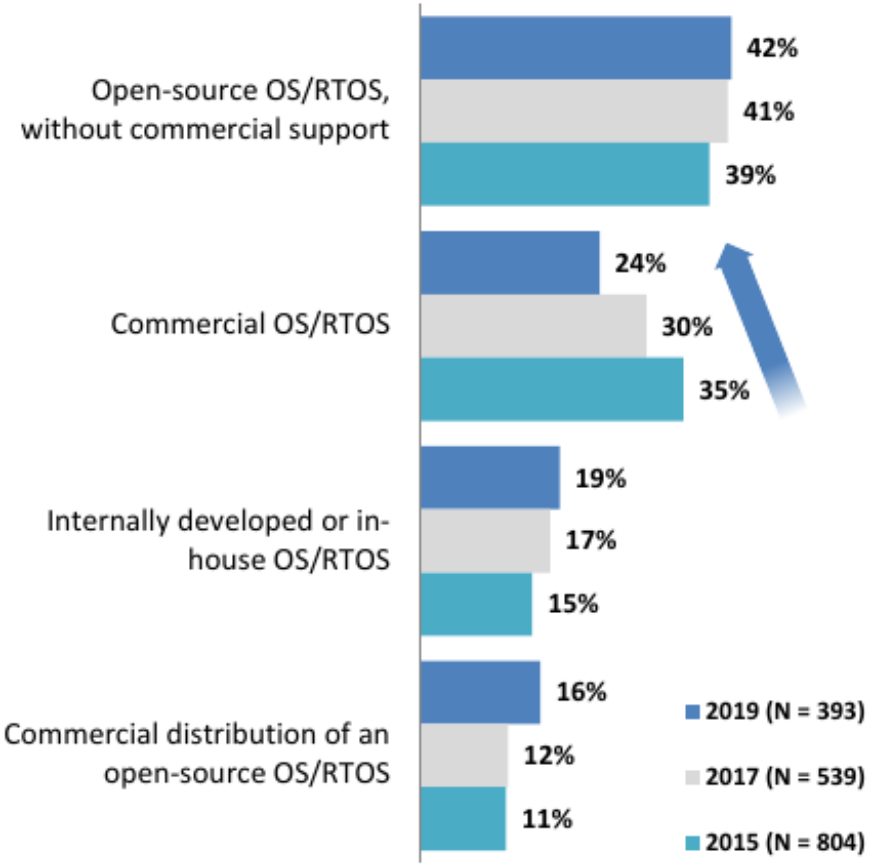
Différents RTOS

safety-critical

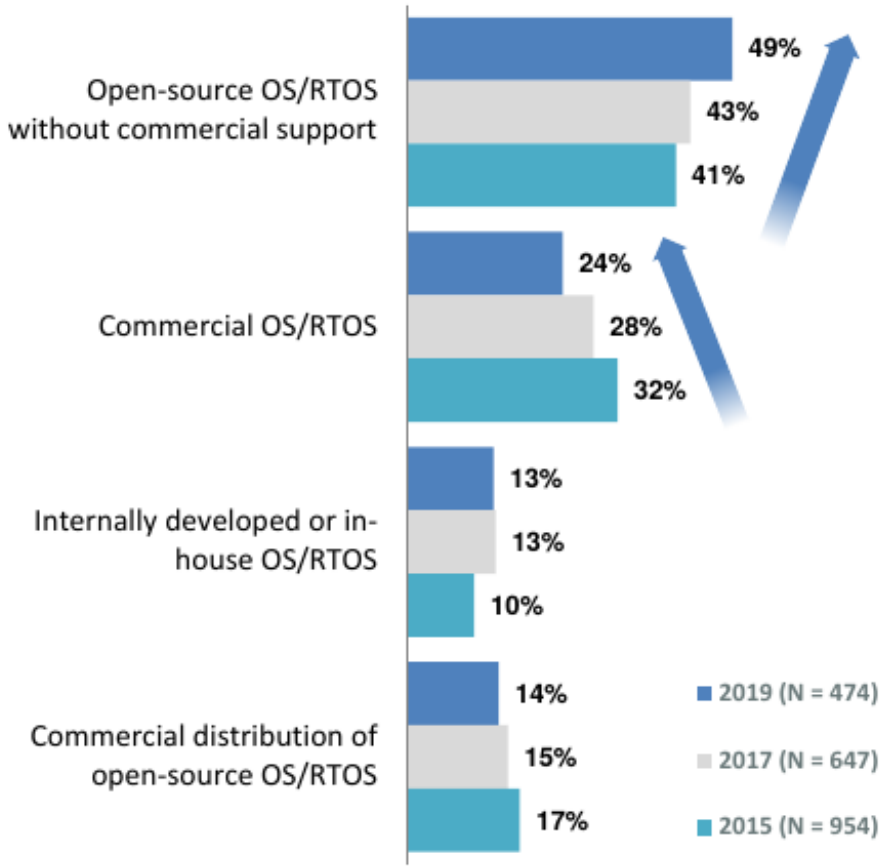


Open or not

My current embedded project uses:



My next embedded project will likely use:



EETimes embedded 2019 Embedded Markets Study

Contenu du cours

- **Qu'est-ce qu'un OS temps réel (RTOS)**
  - RTOS vs GPOS
  - Différents RTOS...
  - **Les objets *communs* d'un RTOS**
    - Tâche
    - Sémaphore
    - Ordonnanceur
    - Les objets *haut niveau*
  - Implémentation d'un RTOS
- Programmation avec un OS temps réel
- Mise en Oeuvre

Les objets du temps réel
les basiques

- Les objets programmables
  - La tâche (ou le thread ou processus léger)
  - Les routines d'interruption, l'alarme
- Objet de communication
  - Le sémaphore
    - Synchronisation
    - Exclusion mutuelle

Les objets du temps réel
la tâche

- Une tâche est un fil d'exécution
- Une tâche est représentée par une structure[1] contenant :
  - Un identifiant
  - Une référence vers son code
  - Une priorité
  - Un état :
    - Prête
    - Bloquée
    - En cours
    - ...
  - Un contexte
    - valeurs des registres
    - compteur de programme
    - pile

1 : Souvent appelée TaskControlBlock

```
typedef struct
{
    void *r_stack;          // Start of stack (top address-1)
  void (*start) (void);         // Entry point of code
  pProcessID pid;               // Pointer to Process ID block
  unsigned char priority;   // Priority of task (0-255)
} FLASH const TaskControlBlock;


typedef struct ProcessID
{                                           //state is encoded in the scheduler
    struct ProcessID * next;
   unsigned char flags;
   unsigned char priority;
   void *ContextPointer;
#ifdef SINGLESTEPSUPPORT
   unsigned char * bp1;
   unsigned char * bp2;
#endif
} * pProcessId;
```

Exemple simplifié d'AvrX

Les objets du temps réel
la tâche

Primitives classiques :

- **Appel effectué depuis une autre tâche**
  - création (prête ou bloquée)
  - Lancement
  - Réveil
- **Appel effectué depuis une autre tâche ou dans la tâche elle-même**
  - Destruction
  - Suspension
  - Demande du niveau de priorité
  - Changement de priorité
- **Appel dans la tâche elle même**
  - mise en sommeil
  - appels de fonctions

```
INTERFACE void AvrXRunTask(TaskControlBlock *);
INTERFACE unsigned char AvrXInitTask(TaskControlBlock *);

INTERFACE void AvrXResume(pProcessID);
INTERFACE void AvrXSuspend(pProcessID);
INTERFACE void AvrXBreakPoint(pProcessID);
INTERFACE unsigned char AvrXSingleStep(pProcessID);
INTERFACE unsigned char AvrXSingleStepNext(pProcessID);

INTERFACE void AvrXTerminate(pProcessID);
INTERFACE void AvrXTaskExit(void);
INTERFACE void AvrXHalt(void);     // Halt Processor (error only)

INTERFACE void AvrXWaitTask(pProcessID);
INTERFACE Mutex AvrXTestPid(pProcessID);

INTERFACE unsigned char AvrXPriority(pProcessID);
INTERFACE unsigned char AvrXChangePriority(pProcessID, unsigned char);
INTERFACE pProcessID AvrXSelf(void);
```

Exemple d'AvrX

Les objets du temps réel
la tâche

Exemple de primitives

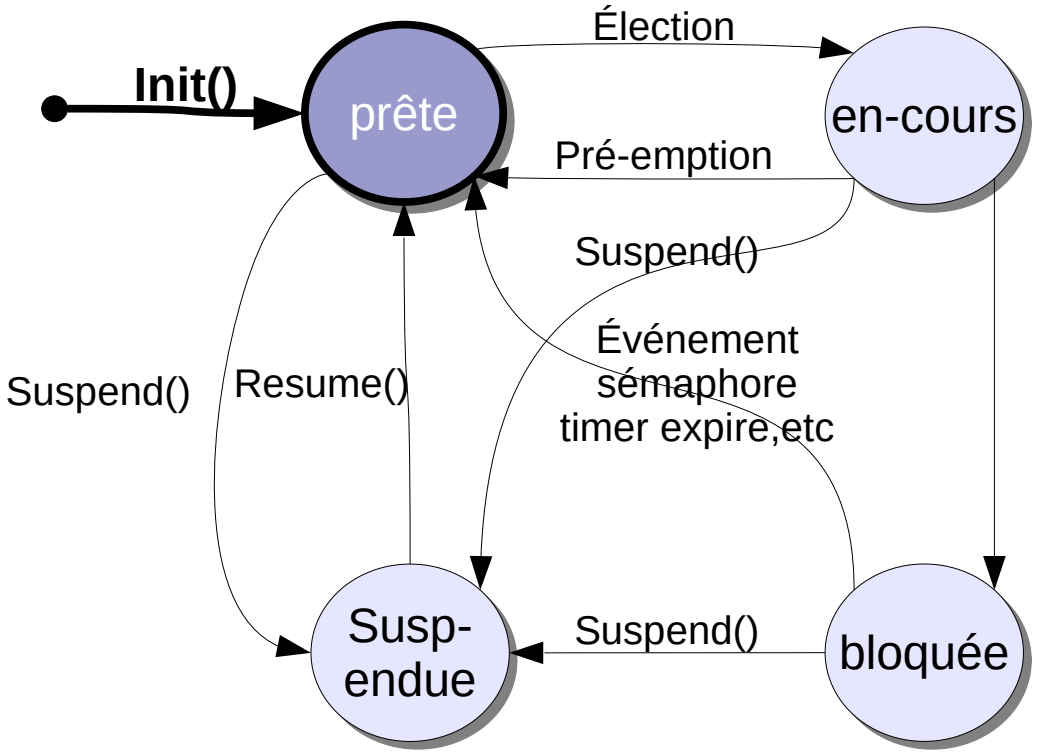
- **VxWorks**
  - LOCAL int taskSpawn(char* "taskName",int priority,0,10000, functionName,0,0,0,0,0, \\ 0,0,0,0,0)
  - LOCAL int taskInit(…)
  - taskSafe ()
- **IRMX**
  - static TOKEN rq_create_task(priority, (void far *) taskId, ...)
- **Win CE**
  - CreateProcess(NULL|appliName, "MonProcessFils", …)
  - HANDLE CreateThread(lpThreadAttributes, dwStackSize, lpStartAddress, lpParameter, \\ dwCreationFlags, CREATE_SUSPENDED, lpThreadId)
- **RT-Linux**
  - int init_module(void)
  - int pthread_create(pthread_t, pthread_attr_t , void *(*start_routine)(void*), void *arg)
- **TIM micro-kernel**
  - int install_task (char * taskName, int stackSize, void * functionAd);
- **RTX_166 Tiny Real-Time**
  - os_create_task (int tasd_id);
  - os_delete_task (int tasd_id);

Les objets du temps réel

la tâche

État d'une tâche :

- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée

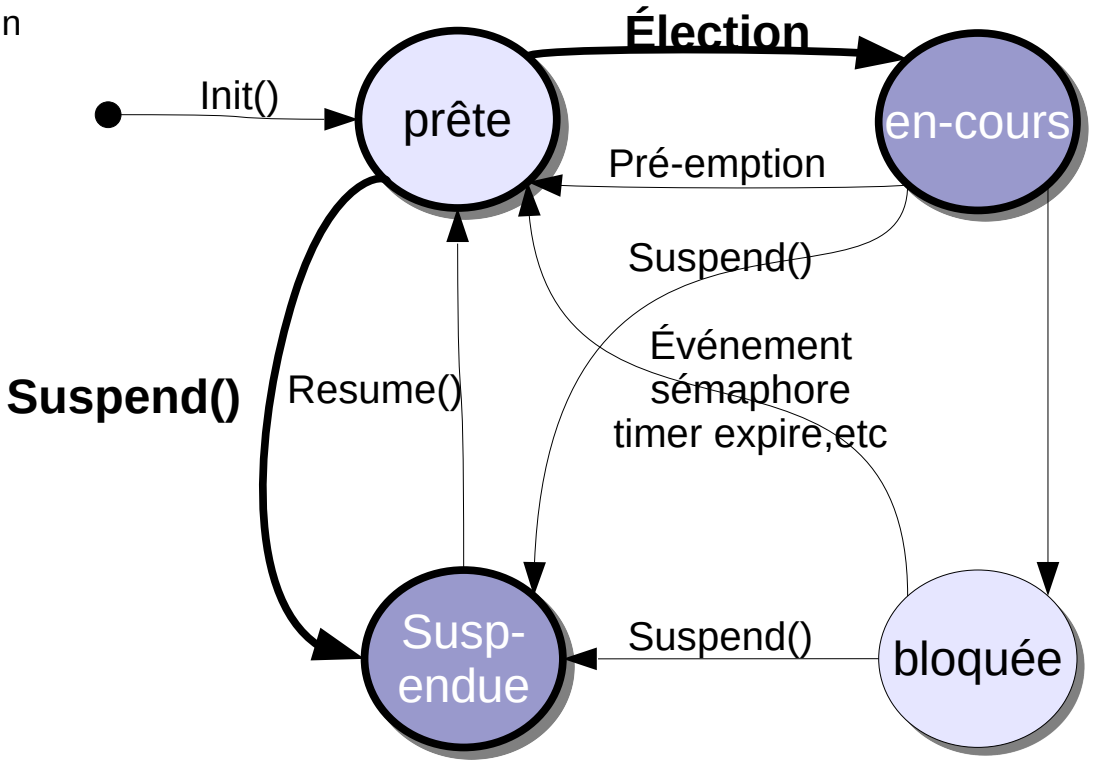


Les objets du temps réel

la tâche

État d'une tâche :

- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée

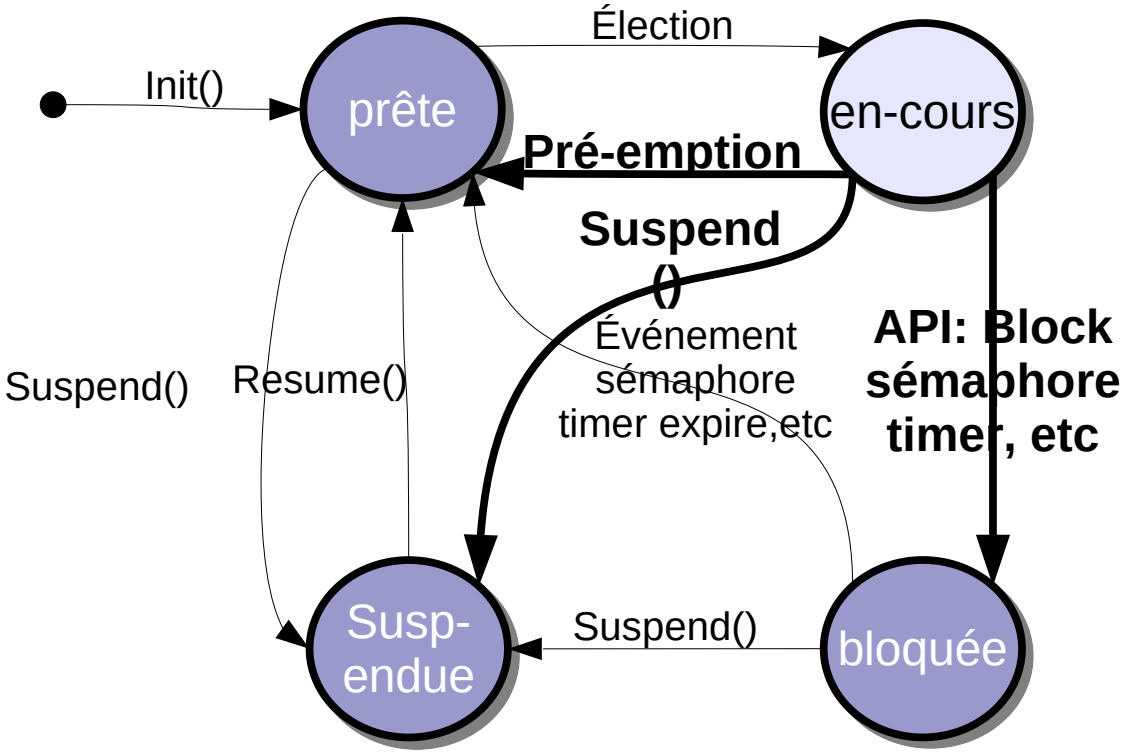


Les objets du temps réel

la tâche

État d'une tâche :

- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée

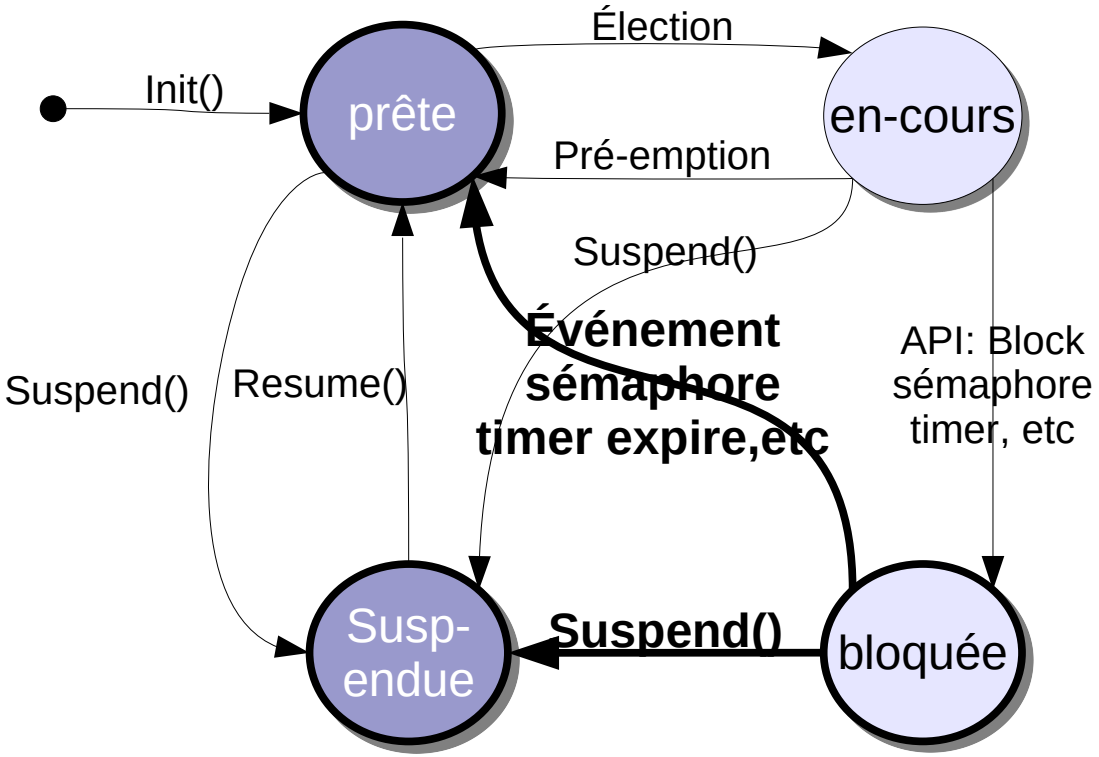


Les objets du temps réel

la tâche

État d'une tâche :

- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée

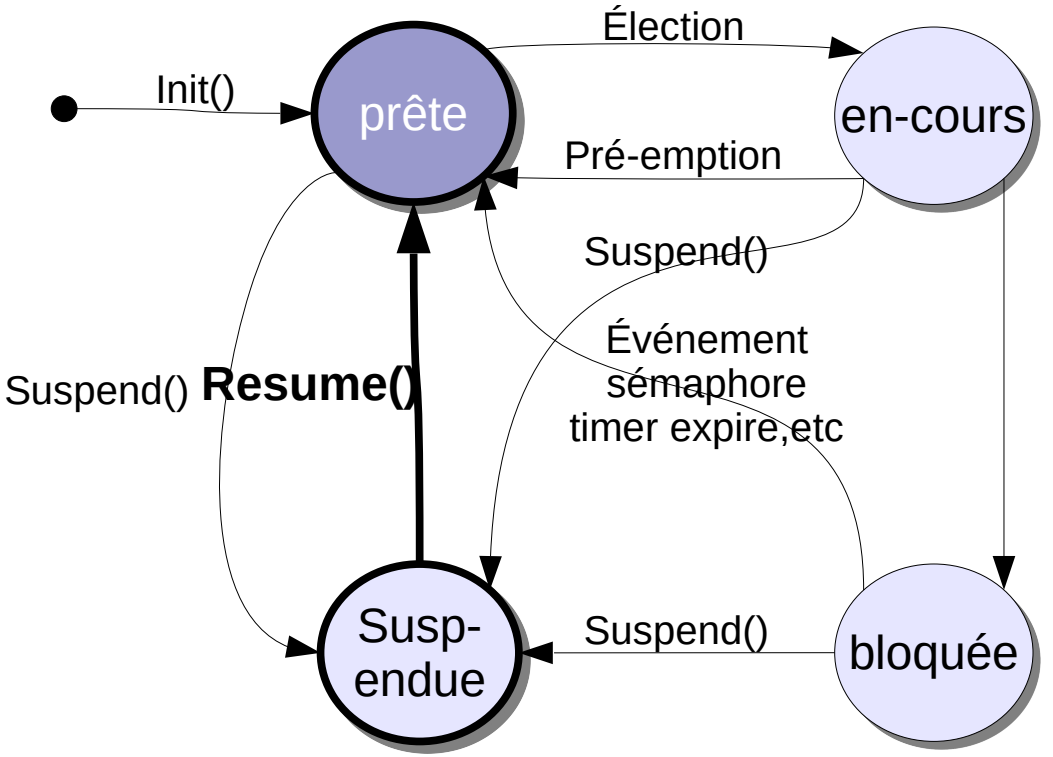


Les objets du temps réel

la tâche

État d'une tâche :

- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée

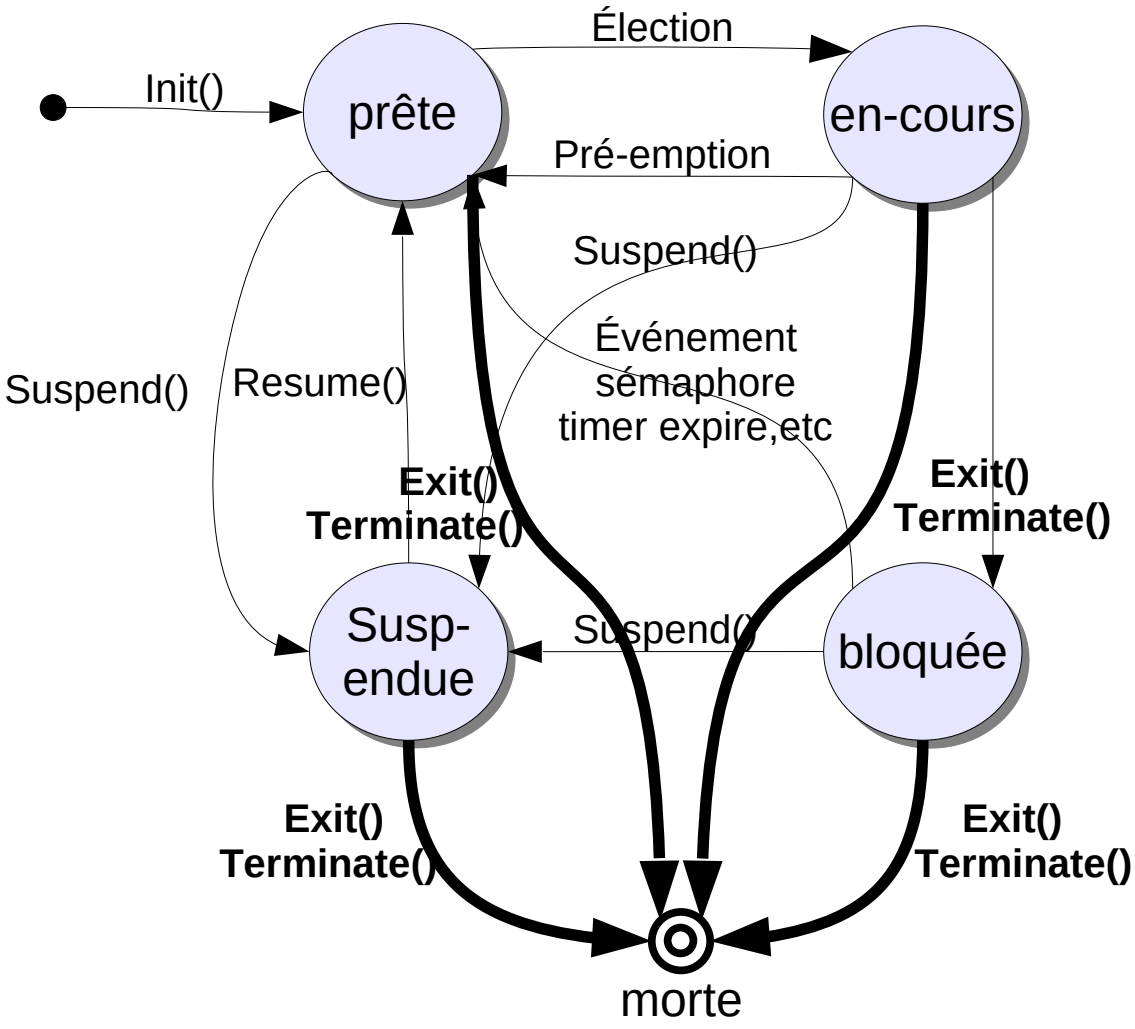


Les objets du temps réel

la tâche

État d'une tâche :

- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée



Les objets du temps réel
la tâche

Tâche classique :

- Initialisation (variables, etc)
- Boucle infinie
  - Faire quelques choses
  - Attendre
    - Ressource
    - Timer
    - Sémaphore

```
AVRX_GCC_TASKDEF(task1, 8, 3)
{
unsigned char valueD=0;
PORTD = valueD;
 while (1)
 {
  AvrXStartTimer(&timer1, 20); // 20 ms delay
  AvrXWaitTimer(&timer1);
  PORTD=++valueD;  // Modify PORTD
  }
}
```

Exemple AvrX

Les objets du temps réel
la tâche

Tâche classique :

- Initialisation (variables, etc)
- Boucle infinie
  - Faire quelques choses
  - Attendre
    - Ressource
    - Timer
    - Sémaphore

```
AVRX_GCC_TASKDEF(task1, 8, 3)
{
unsigned char valueD=0;
PORTD = valueD;
 while (1)
 {
  AvrXStartTimer(&timer1, 20); // 20 ms delay
  PORTD=++valueD;  // Modify PORTD
  AvrXWaitTimer(&timer1);
  }
}
```

Exemple AvrX

Les objets du temps réel
routine d'interruption

Interruption classique :

- Démasque des ITs (ou non)
- Faire le minimum de chose
  - MAJ d'une variable
  - Modification d'un sémaphore
- Fin

```
AVRX_SIGINT(TIMER0_OVF_vect)
{
    IntProlog();    // Save interrupted context, switch stacks
    TCNT0 = TCNT0_INIT;      // Reload the timer counter
    AvrXTimerHandler();      // Process Timer queue
    Epilog();       // Restore context of next running task
}
```

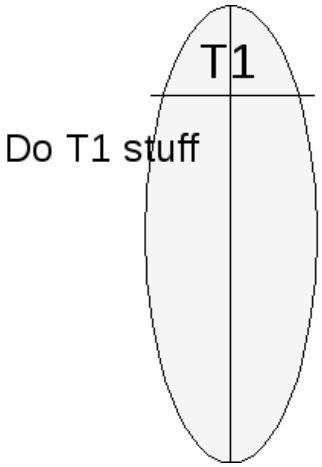
Exemple AvrX

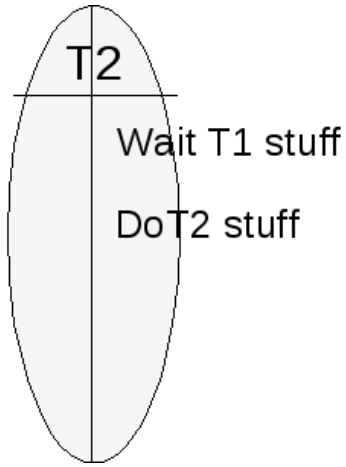
Les objets du temps réel
le sémaphore

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ses primitives sont atomiques !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - exclusion mutuelle
  - Interruption logicielle

Les objets du temps réel
le sémaphore

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ses primitives sont atomiques !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - **synchronisation entre tâches**
  - exclusion mutuelle
  - Interruption logicielle

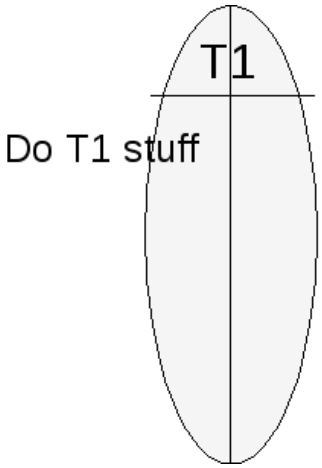


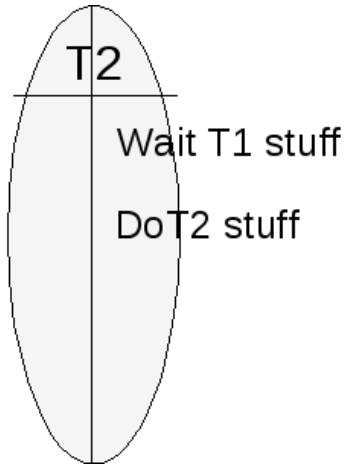


Les objets du temps réel
le sémaphore

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - **synchronisation entre tâches**
  - exclusion mutuelle
  - Interruption logicielle

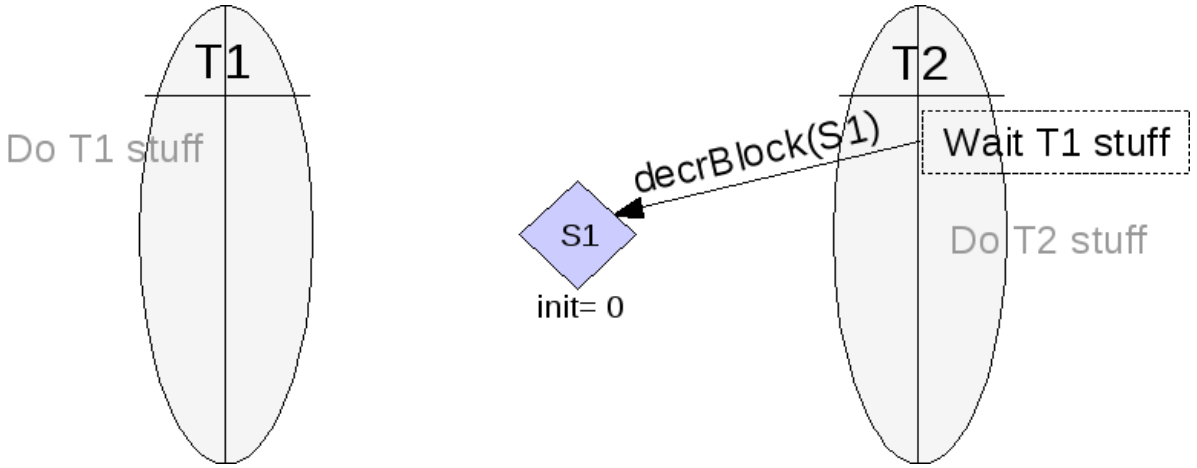
Lacatre: langage d'aide à la conception d'application temps réel
www.lisyc.univ-brest.fr/pages_perso/babau/cours/multitache.pdf





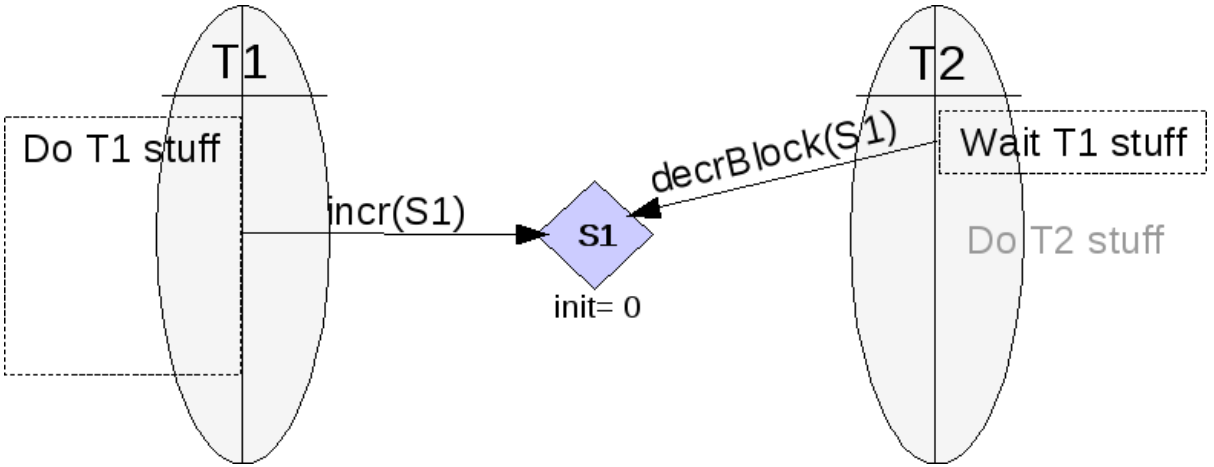
Les objets du temps réel
le sémaphore

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - **synchronisation entre tâches**
  - exclusion mutuelle
  - Interruption logicielle



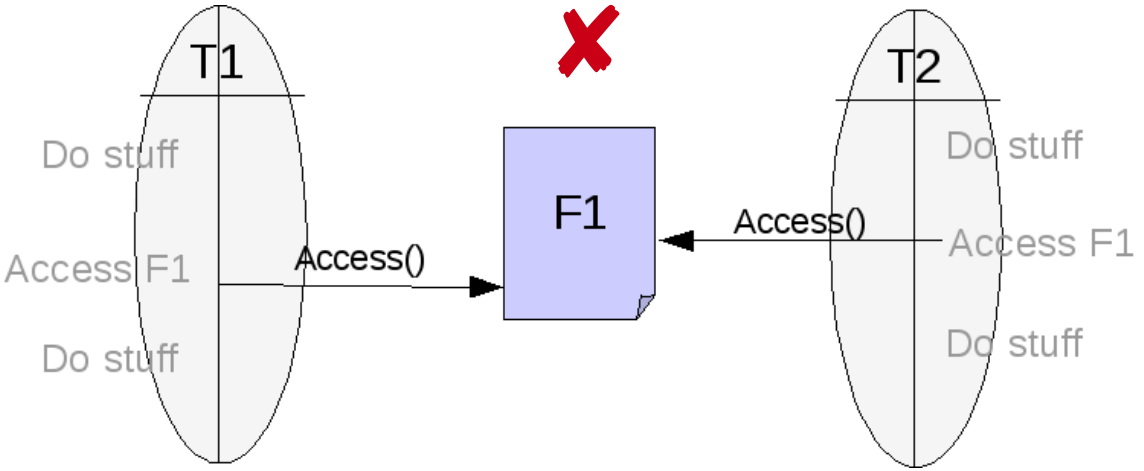
Les objets du temps réel
le sémaphore

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - **synchronisation entre tâches**
  - exclusion mutuelle
  - Interruption logicielle



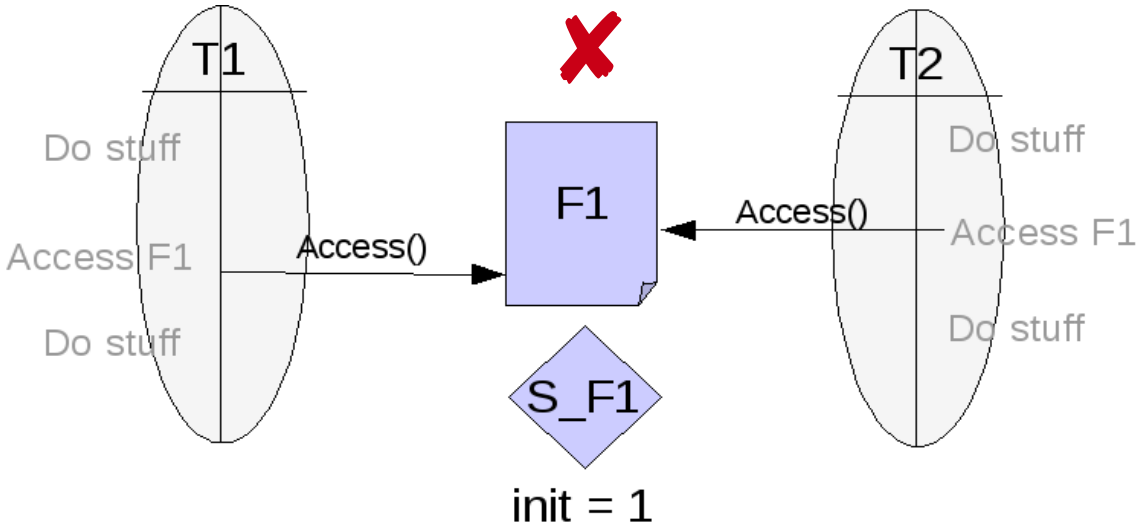
Les objets du temps réel
le sémaphore

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - **exclusion mutuelle**
  - Interruption logicielle



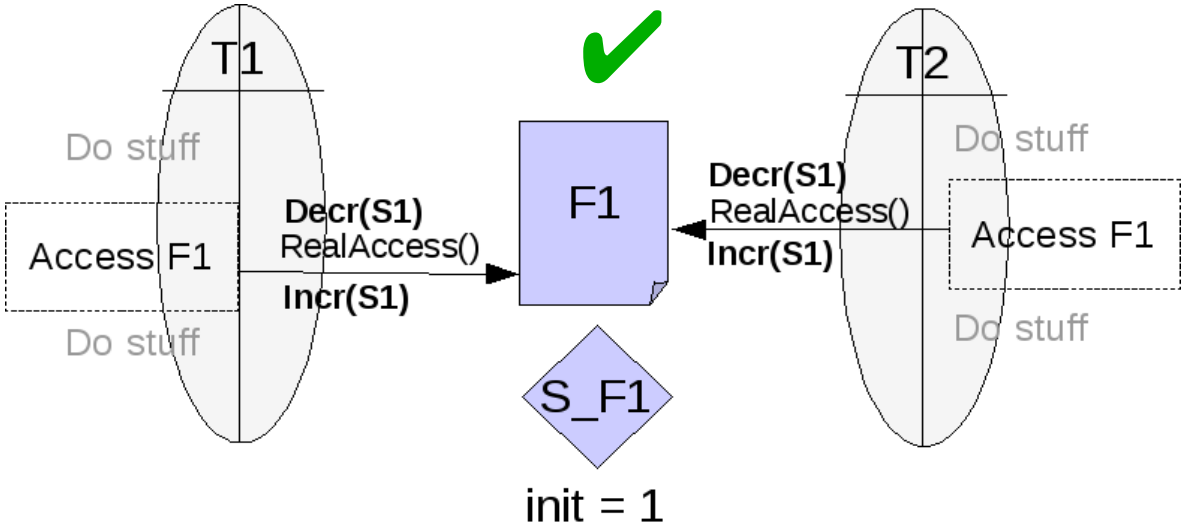
Les objets du temps réel
le sémaphore

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - **exclusion mutuelle**
  - Interruption logicielle



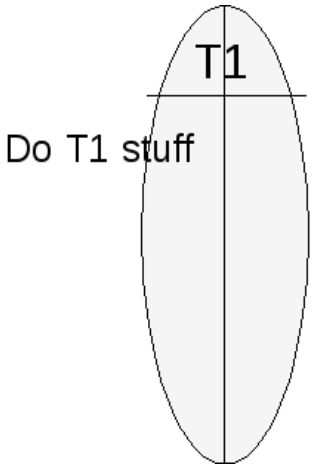
Les objets du temps réel
le sémaphore (2)

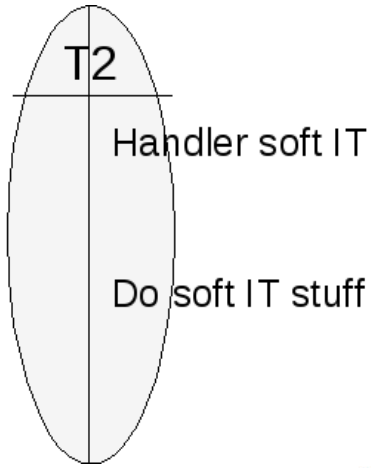
- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - **exclusion mutuelle**
  - Interruption logicielle



Les objets du temps réel
le sémaphore (3 ?)

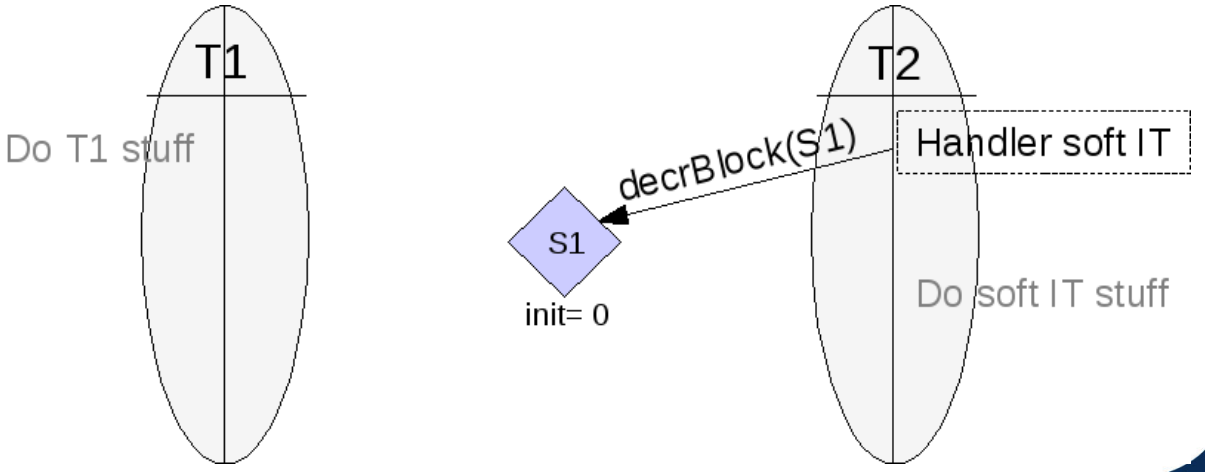
- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - exclusion mutuelle
  - **Interruption logicielle**





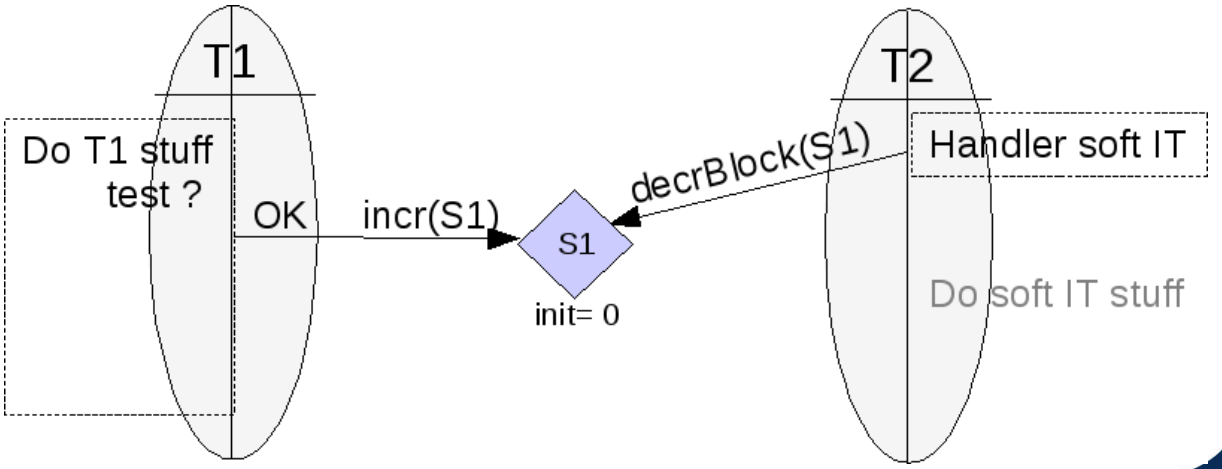
Les objets du temps réel
le sémaphore

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - exclusion mutuelle
  - **Interruption logicielle**



Les objets du temps réel
le sémaphore

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - exclusion mutuelle
  - **Interruption logicielle**



Les objets du temps réel
le sémaphore

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - exclusion mutuelle
  - Interruption logicielle
- **Accès par file d'attente**
  - **FIFO**
  - **PRIORITY**

Les objets du temps réel
le sémaphore

- Primitives classiques
  - Création
    - Initialisation
  - Destruction
    - impact sur l ’application
  - Dépôt (incrémentation)
    - nombre d’unité
  - Retrait (décrémentation)
    - nombre d’unité
    - temps d’attente
      - Infini
      - fini

```
void AvrXSetSemaphore(pMutex);       //blocking
void AvrXIntSetSemaphore(pMutex);    //non blocking
void AvrXWaitSemaphore(pMutex);
Mutex AvrXTestSemaphore(pMutex);     //blocking
Mutex AvrXIntTestSemaphore(pMutex); //non blocking
void AvrXResetSemaphore(pMutex);
void AvrXResetObjectSemaphore(pMutex);
```

Exemple d'AvrX

Les objets du temps réel
le sémaphore

Exemples de primitives

- **VxWorks**

```
semId = semBCreate(SEM_Q_FIFO | SEM_Q_PRIORITY, SEM_FULL | SEM_EMPTY) ;
semId = semCCreate(SEM_Q_FIFO | SEM_Q_PRIORITY, initCount) ;
semId = semMCreate(SEM_Q_FIFO | SEM_Q_PRIORITY |SEM_DELETE_SAFE| SEM_INVERSION_SAFE)
status = semGive(semId)
status = semFlush(semId) ; /* déblocage de toutes les tâches en attente
status = semTake(semId, temps | WAIT_FOREVER | NO_WAIT) ;
```

- **iRMX**

```
semId_tk = rq_create_semaphore(valInit,valMax,flags,&status) ;
rq_send_units(semId,nbUnite,&status)
reste = rq_receive_units(semId_tk,nbUnite,temps,&status)
```

- **Win32**

```
HANDLE CreateSemaphore( LPSECURITY_ATTRIBUTES, InitialCount, MaximumCount, lpName);
ReleaseSemaphore ( semhandle , unitNumber , 0 )
WaitForSingleObject ( sem , INFINITE | Time-out ) // timeout en ms
```

- **TIM micro-kernel** : pas de « sémaphore »

Les objets du temps réel
l'ordonnanceur

- Mode préemptif
  - Ordonnanceur appelé après chaque changement d'état
    - Appel des primitives de communication
    - Après une routine (si déblocage possible d'une tâche)

- Mode non préemptif
  - Ordonnanceur appelé à la fin de tâche / boucle
    - task_suspend(), task_exit()

- Ordonnancement classique
  - Round Robin
  - Round Robin par priorités

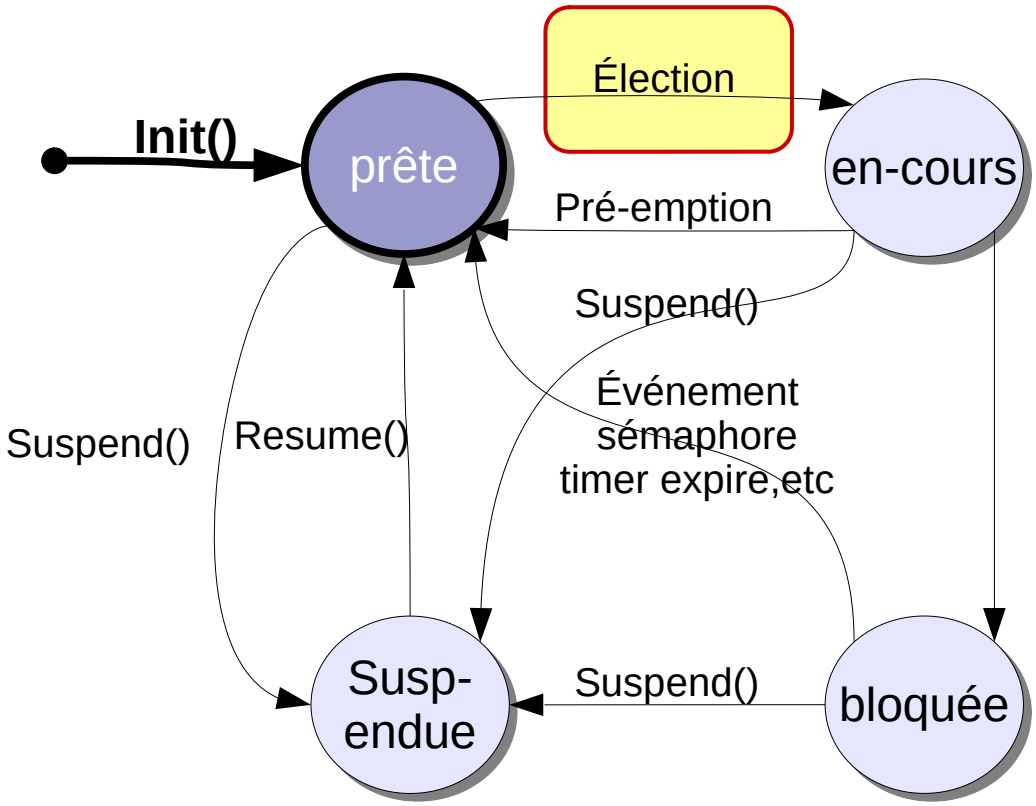
- Politique d'ordonnacement (choix de prio)
  - EDF (earliest deadline first)
  - RMA (rate monotonic analysis)

Les objets du temps réel

la tâche (4)

État d'une tâche :

- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée



Les objets du temps réel
l'ordonnanceur (1)

- C'est le cœur d'un RTOS, c'est le chef qui élicite les tâches prêtes…
- Ça peut être compliqué de comprendre
  - Comment donner les priorités aux différentes tâches ?
  - Est-ce que le système est ordonnançable ?
  - comment s'ordonnancent les différentes tâches ?

Comment donner les priorités aux différentes tâches ?

- L'***algorithme d'ordonnancement*** permet d'affecter les priorités aux tâches.
- Selon les algos ils déterminent des priorités fixes, dynamiques ou un mélange entre les deux.
- Les priorités fixes sont affectées lors de la conception et ne varient pas lors de l'exécution
- Les priorités dynamiques sont affectées à l'exécution en fonction des paramètres des tâches tels que l'échéance.

Comment donner les priorités aux différentes tâches ?

- L'*algorithme d'ordonnancement* permet d'affecter les priorités aux tâches.
- RMA (Rate Monotonic Analysis) propose de trouver les **priorités fixes** des tâches. Il met la priorité la plus haute aux traitements *les plus fréquents*

Comment donner les priorités aux différentes tâches ?

- L'*algorithme d'ordonnacement* permet d'affecter les priorités aux tâches.
- RMA (Rate Monotonic Analysis) propose de trouver les priorités fixes des tâches. Il met la priorité la plus haute aux traitements ***les plus fréquents***

Comment donner les priorités aux différentes tâches ?

- L'*algorithme d'ordonnacement* permet d'affecter les priorités aux tâches.
- RMA (Rate Monotonic Analysis) propose de trouver les priorités fixes des tâches. Il met la priorité la plus haute aux traitements *les plus fréquents*
- EDF (Earliest Deadline First) propose de trouver les **priorités dynamiques** des tâches. Il met la priorité la plus haute à la tâche la plus proche de son échéance.

Comment donner les priorités aux différentes tâches ?

- L'*algorithme d'ordonnacement* permet d'affecter les priorités aux tâches.
- RMA (Rate Monotonic Analysis) propose de trouver les priorités fixes des tâches. Il met la priorité la plus haute aux traitements *les plus fréquents*
- EDF (Earliest Deadline First) propose de trouver les priorités dynamiques des tâches. Il met la priorité la plus haute à la tâche la plus proche de **son échéance**.

Est-ce que le système est ordonnançable ?

- Avant de répondre il faut comprendre le ”modèle” de tâche utilisé habituellement
  - Task = <P, D, ET, prio>
    - P est la période
    - D est l'échéance (deadline)
    - ET est le temps d'exécution (souvent le pire (WCET)
    - Prio est la priorité

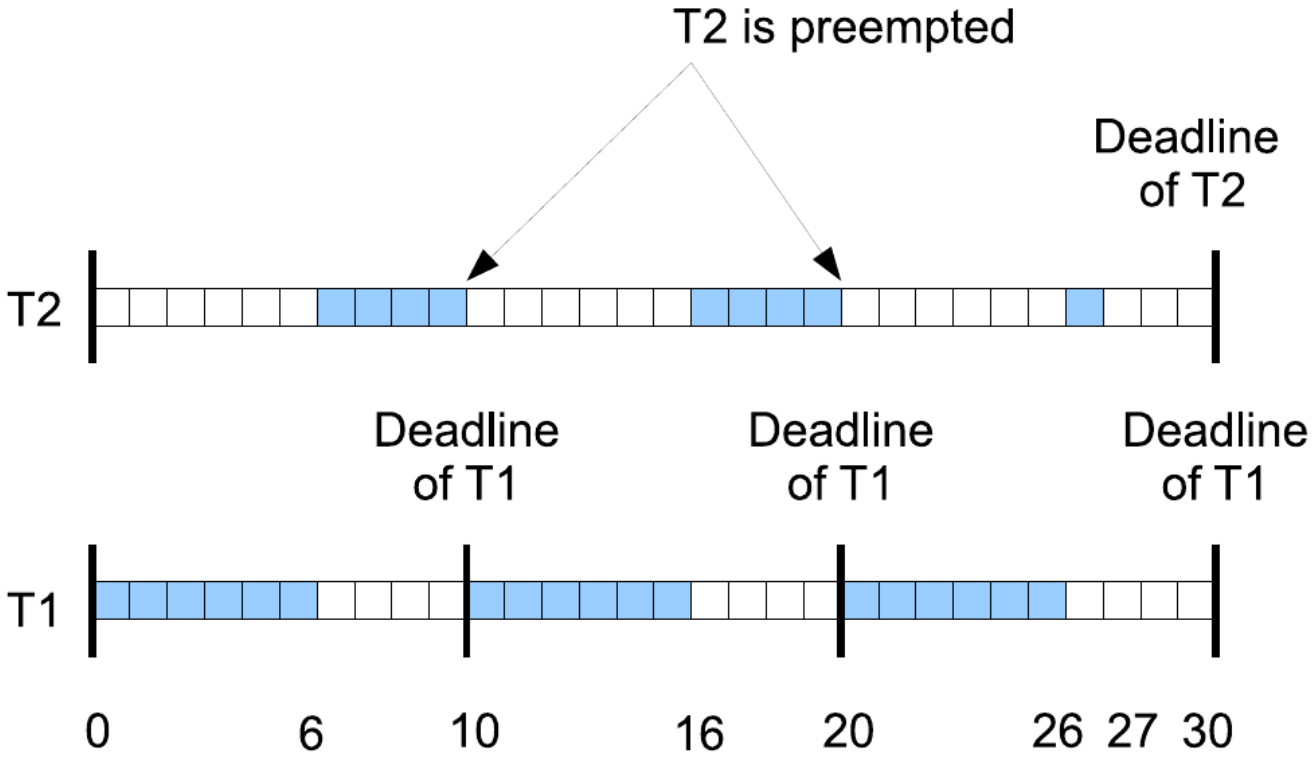
Est-ce que le système est ordonnançable ?

- Avant de répondre il faut comprendre le ”modèle” de tâche utilisé habituellement
  - Task = <P, D, ET, prio>
    - P est la période
    - D est l'échéance (deadline)
    - ET est le temps d'exécution (souvent le pire (WCET)
    - Prio est la priorité

Un système est ordonnançable si on trouve un
moyen d'exécuter toutes les taches avant leur échéance

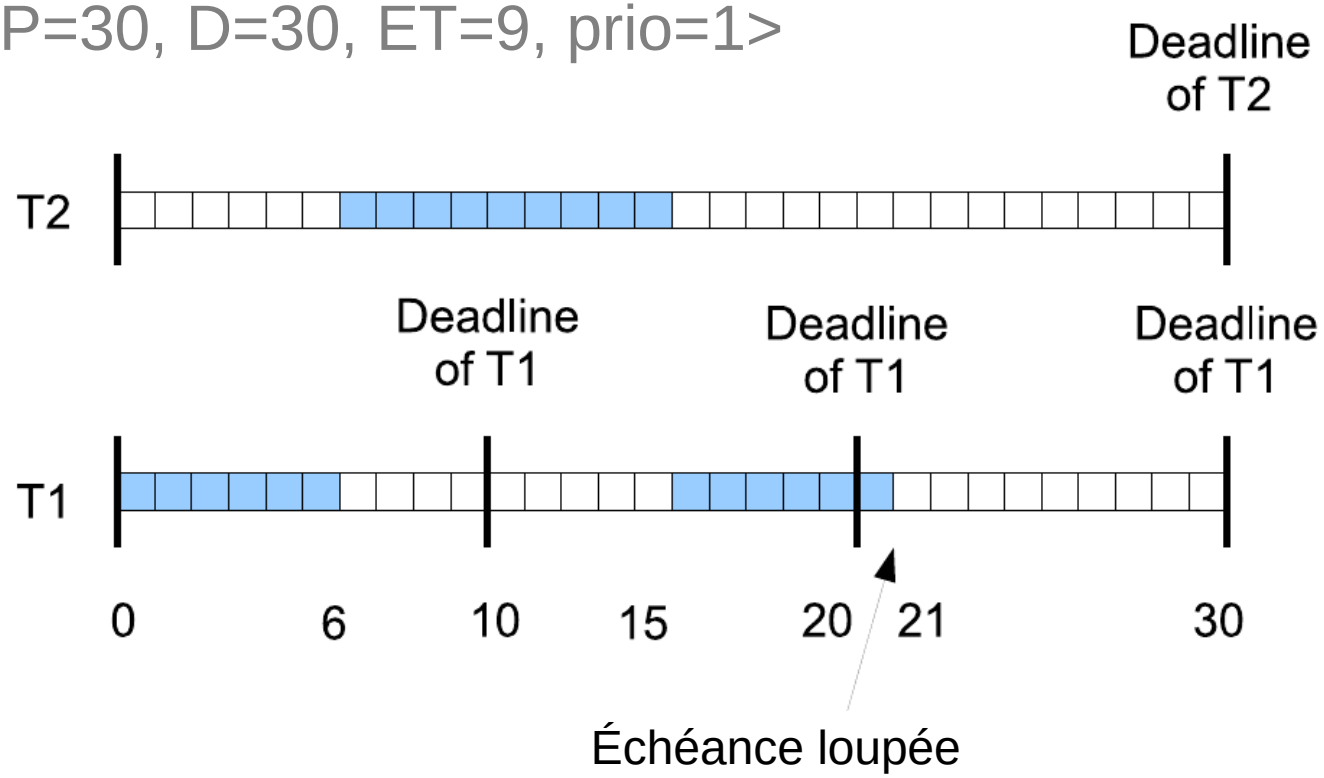
Est-ce que le système est ordonnançable ?

- Avant de répondre il faut comprendre le ”modèle” de tâche utilisé habituellement
  - T1= <P=10, D=10, ET=6, prio=0>
  - T2 : <P=30, D=30, ET=9, prio=1>



Est-ce que le système est ordonnançable ?

- Avant de répondre il faut comprendre le ”modèle” de tâche utilisé habituellement
  - T1= <P=10, D=10, ET=6, prio=0>
  - T2 : <P=30, D=30, ET=9, prio=1>



Worst Case Execution Time ?

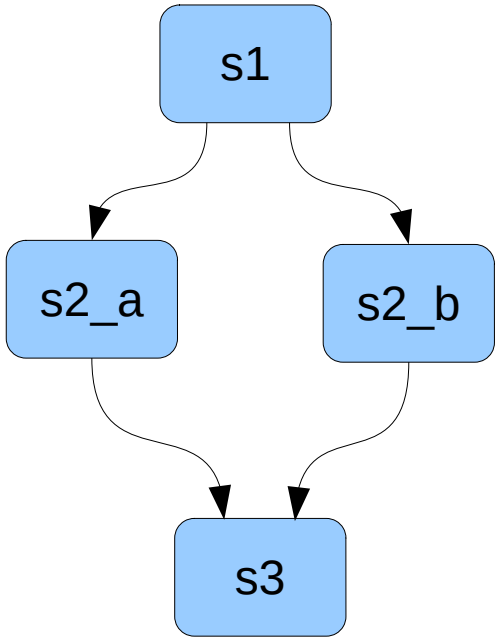
- Usually done by static analyis of the object code

http://www.mrtc.mdh.se/projects/wcet/

- Usually consider limited hardware capabilities (no caches, no branch prediction)
- The global idea:
  - Extract the control flow of the program,
  - Extract sequential, unbranched part of the program (segments)
  - Compute execution time of each segment
  - Sum the max of all segments according to the control flow

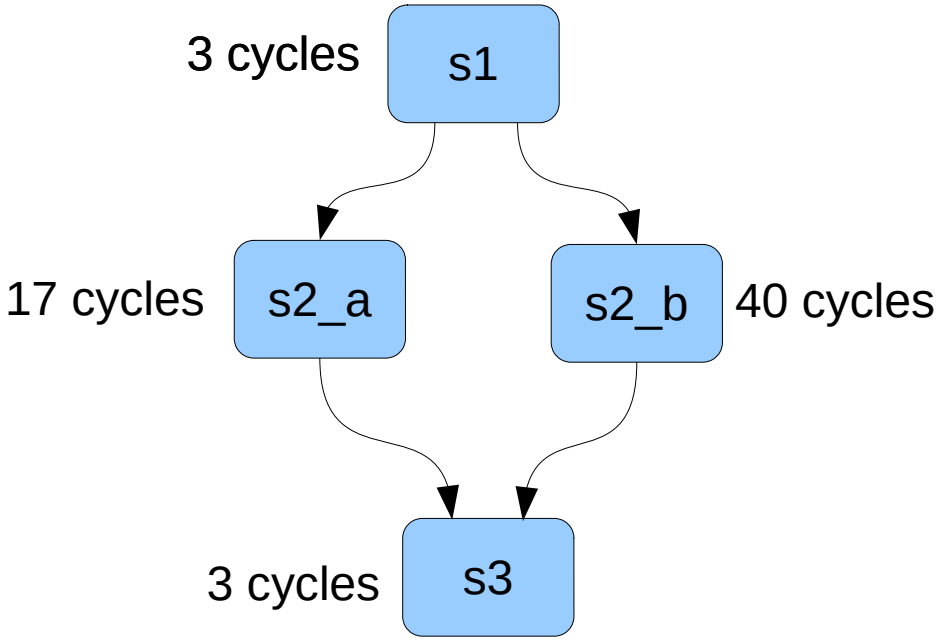
Worst Case Execution Time ?

```
f(int i) {
  PORTA=0x55
  if (i > 0){
    PORTB = PORTB << (i%2)
  }else{
    PCMSK0 = 0x01
    PCIRC0 = i/someVar
    sei()
  }
  PORTA=0xAA
}
```



Worst Case Execution Time ?

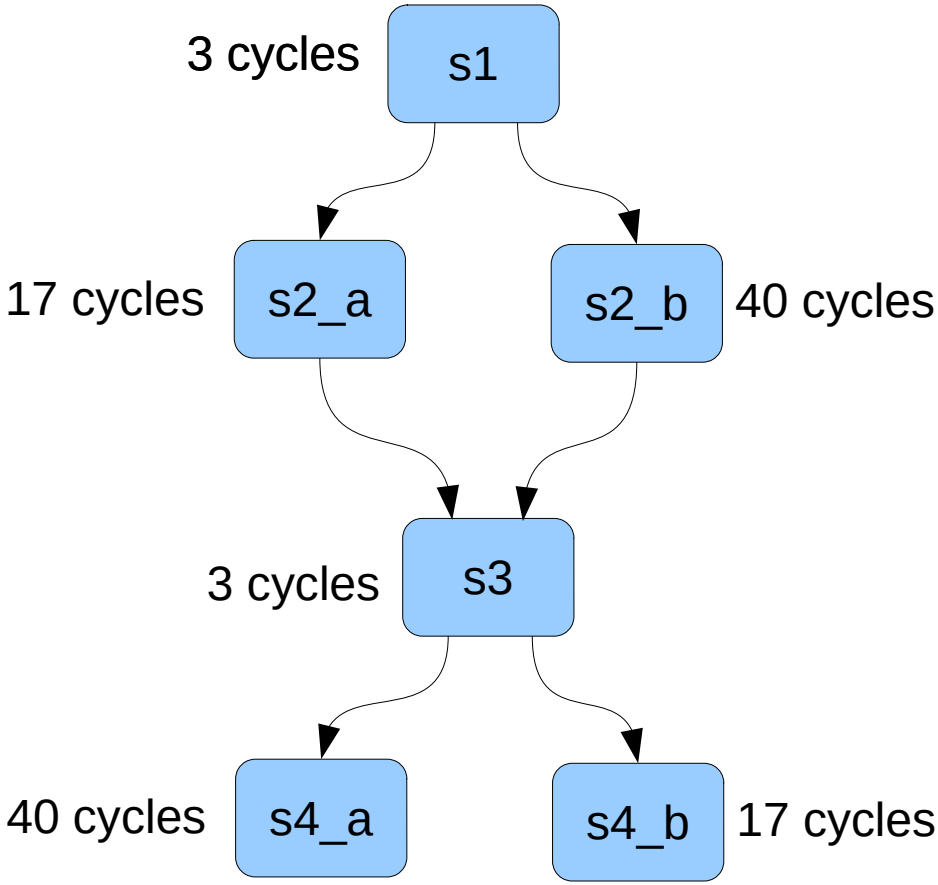
```
f(int i) {
  PORTA=0x55
  if (i > 0){
    PORTB = PORTB << (i%2)
  }else{
    PCMSK0 = 0x01
    PCIRC0 = i/someVar
    sei()
  }
  PORTA=0xAA
}
```



WCET
=
46 cycles

Worst Case Execution Time ?

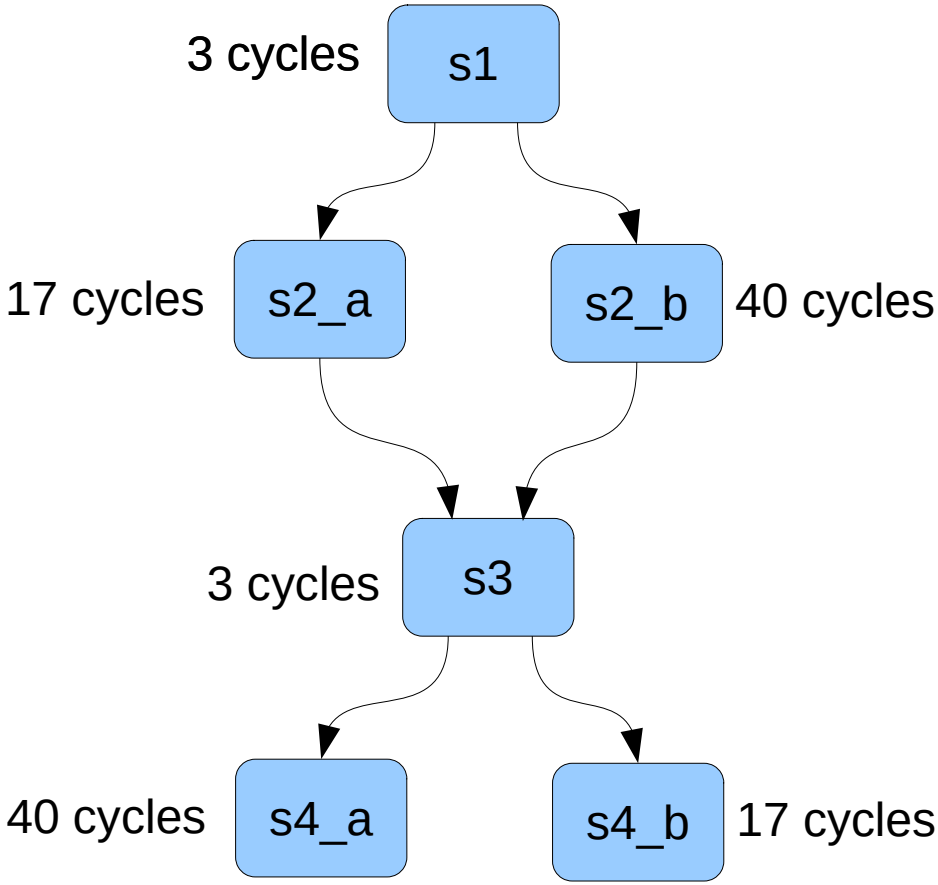
```
f(int i) {
  PORTA=0x55
  if (i > 0){
    PORTB = PORTB << (i%2)
  }else{
    PCMSK0 = 0x01
    PCIRC0 = i/someVar
    sei()
  }
  PORTA=0xAA
  if (i <= 0){
   PORTB = PORTB << (i%2)
  }else{
    PCMSK0 = 0x01
    PCIRC0 = i/someVar
    sei()
  }
}
```



Worst Case Execution Time ?

```
f(int i) {
  PORTA=0x55
  if (i > 0){
    PORTB = PORTB << (i%2)
  }else{
    PCMSK0 = 0x01
    PCIRC0 = i/someVar
    sei()
  }
  PORTA=0xAA
  if (i <= 0){
   PORTB = PORTB << (i%2)
  }else{
    PCMSK0 = 0x01
    PCIRC0 = i/someVar
    sei()
  }
}
```

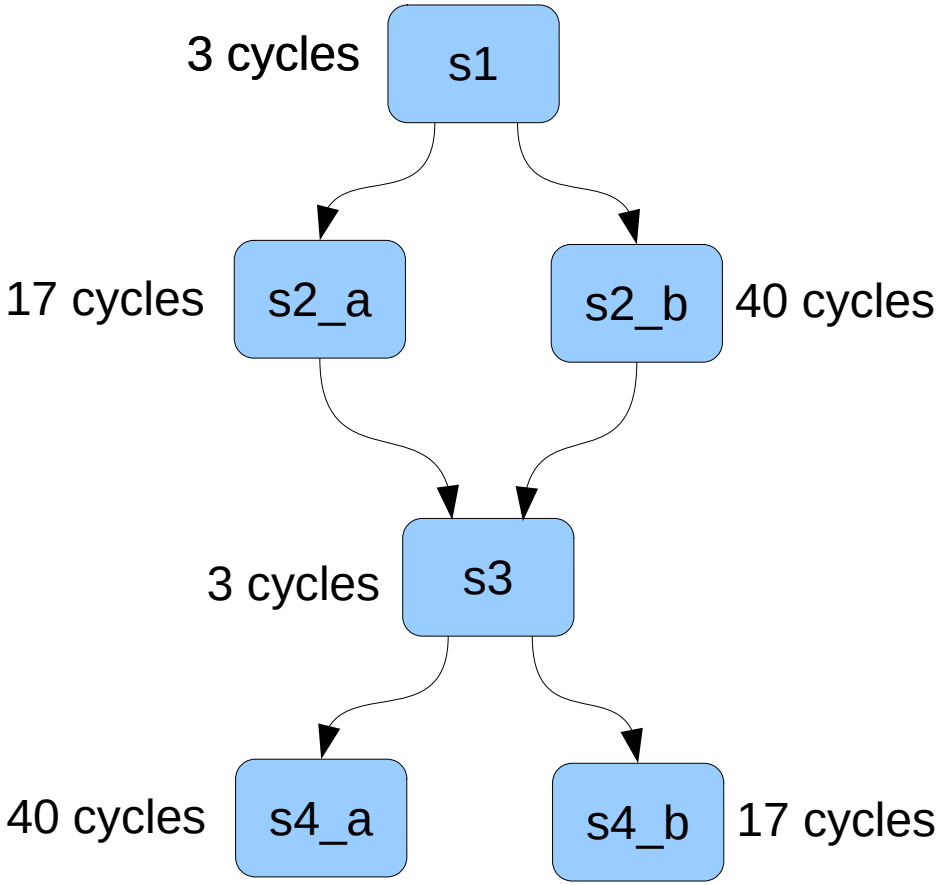
WCET
=
??? cycles



Worst Case Execution Time ?

```
f(int i) {
  PORTA=0x55
  if (i > 0){
    PORTB = PORTB << (i%2)
  }else{
    PCMSK0 = 0x01
    PCIRC0 = i/someVar
    sei()
  }
  PORTA=0xAA
  if (i <= 0){
   PORTB = PORTB << (i%2)
  }else{
    PCMSK0 = 0x01
    PCIRC0 = i/someVar
    sei()
  }
}
```

WCET
=
86 cycles

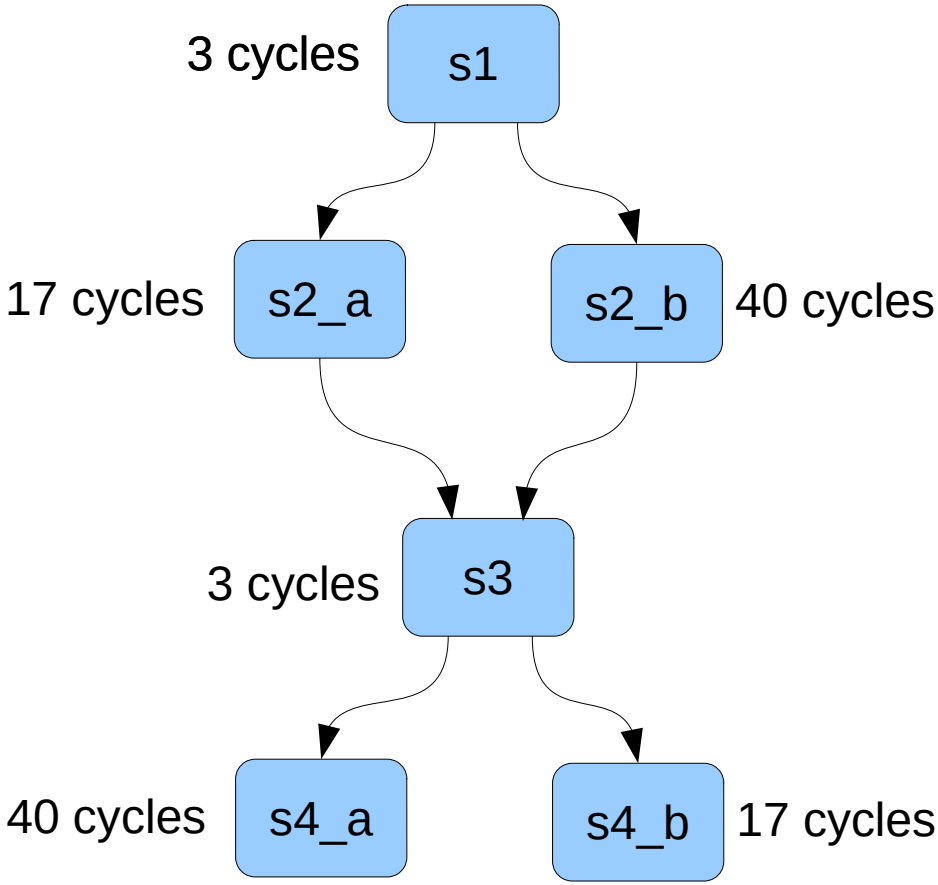


Pessimistic result….

Worst Case Execution Time ?

```
f(int i) {
  PORTA=0x55
  if (i > 0){
    PORTB = PORTB << (i%2)
  }else{
    PCMSK0 = 0x01
    PCIRC0 = i/someVar
    sei()
  }
  PORTA=0xAA
  if (i < 0){
   PORTB = PORTB << (i%2)
  }else{
    PCMSK0 = 0x01
    PCIRC0 = i/someVar
    sei()
  }
}
```

WCET
=
63 cycles



Les objets du temps réel
timer logiciel (1)

- Pas de nombre limité (théoriquement)
- attente non active (pour le micro-contrôleur)
- Différentes stratégies :
  - 1 tâche spécifique qui modifie plusieurs sémaphores lors de l'expiration d'un timer (pas forcément les même à chaque fois) (période ≅ timer)
  - Appel d'un delai dans la tâche concernée (période dépendante du temps d'exécution des calculs)
  - Démarrage et arrêt d’un timer dans la tâche concerné

Les objets du temps réel
objets *haut-niveau*

- **Événement**
  - synchronisation entre tâches
  - rendez-vous
- **Boîte aux lettres (FIFO)**
  - échange d'information synchronisées entre tâches
  - objet de communication entre lecteurs/écrivains
- **Pipe**
  - échange d'information synchronisées entre tâches de processus distinct
  - objet de communication entre lecteurs/écrivains

Contenu du cours

- **Qu'est-ce qu'un OS temps réel (RTOS)**
  - RTOS vs GPOS
  - Différents RTOS...
  - Les objets *communs* d'un RTOS
  - **Implémentation d'un RTOS**
- Programmation avec un OS temps réel
- Mise en Oeuvre

Implémentation d'un RTOS gestion du temps

- Base de temps unique pour le système
  - ➔ Mise en place d'une interruption périodique : compteur de ticks
    - Activation d'un timer
    - intervalle de X ms (10, 200, ...)
    - peut souvent être modifiée

  - L'OS prend la main
  - L'OS incrémente le compteur de tick
  - Vérifie si des timers logiciels ont expiré
    - Modifie des sémaphores et/ou l'état des tâches selon
    - Ré-ordonnance si des changement ont eu lieu
  - L'OS rend la main

Implémentation d'un RTOS
L'ordonnanceur

- Sans priorité, non pré-emptif
  - Tient à jour une liste (un tableau) des tâches dans l'état ”prête”
  - Lance l'exécution de la prochaine tâche dans la liste dès que la tâche en cours rend la main

- Sans priorité, pré-emptif
  - Tient à jour une liste (un tableau) des tâches dans l'état ”prête”
  - Lance l'exécution de la prochaine tâche dans la liste dès que la tâche en cours passe dans l'état 'bloquée', 'suspendue' ou 'morte'

Implémentation d'un RTOS
L'ordonnanceur (2)

- Avec priorité, pré-emptif
  - Tient à jour plusieurs listes des tâches dans l'état ”prête” : une par priorité
  - Lance l'exécution de la prochaine tâche dans la liste de plus haute priorité non vide dès qu'une tâche de plus haute priorité est ”prête”. Dans ce cas la tâche en cours passe dans l'état 'bloquée'. (ou rarement 'suspendue' ou 'morte')

Implémentation d'un RTOS
L'ordonnanceur (2)

- Avec priorité, pré-emptif
  - Tient à jour plusieurs listes des tâches dans l'état ”prête” : une par priorité
  - Lance l'exécution de la prochaine tâche dans la liste de plus haute priorité non vide (dès qu'une tâche de plus haute priorité est ”prête”) et la tâche en cours passe dans l'état 'bloquée', 'suspendue' ou 'morte')
- Inversion de priorité ?
  - Si une tâche de haute priorité est en attente d'une ressource bloquée par une basse priorité, la basse priorité devient prioritaire pour que la haute priorité puisse s'exécuter

Contenu du cours

- **Qu'est-ce qu'un OS temps réel (RTOS)**
  - RTOS vs GPOS
  - Différents RTOS...
  - Les objets *communs* d'un RTOS
  - Implémentation d'un RTOS
- **Programmation avec un OS temps réel**
- Mise en Oeuvre

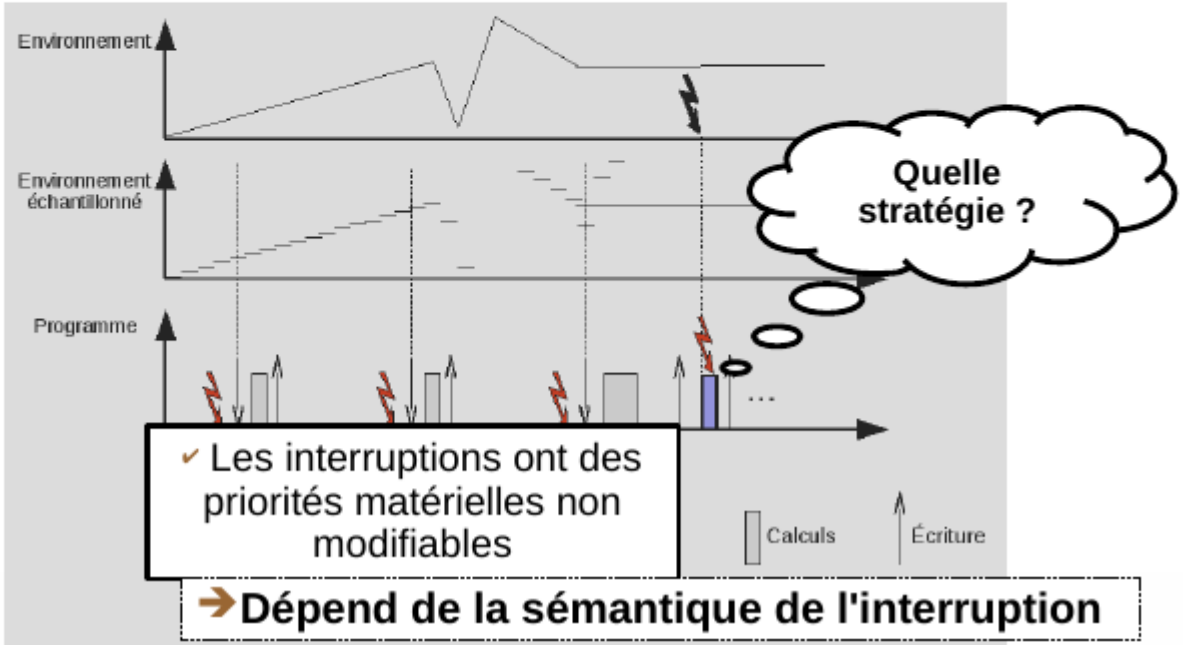
Programmation avec un RTOS les tâches (1)

Rappel :

Micro-contrôleur sans OS : avec IT

2) Avec interruption
   - Inconvénient :
     - Demande un peu de technique

Limite du sans OS ?



Programmation avec un RTOS
les tâches

Principe :

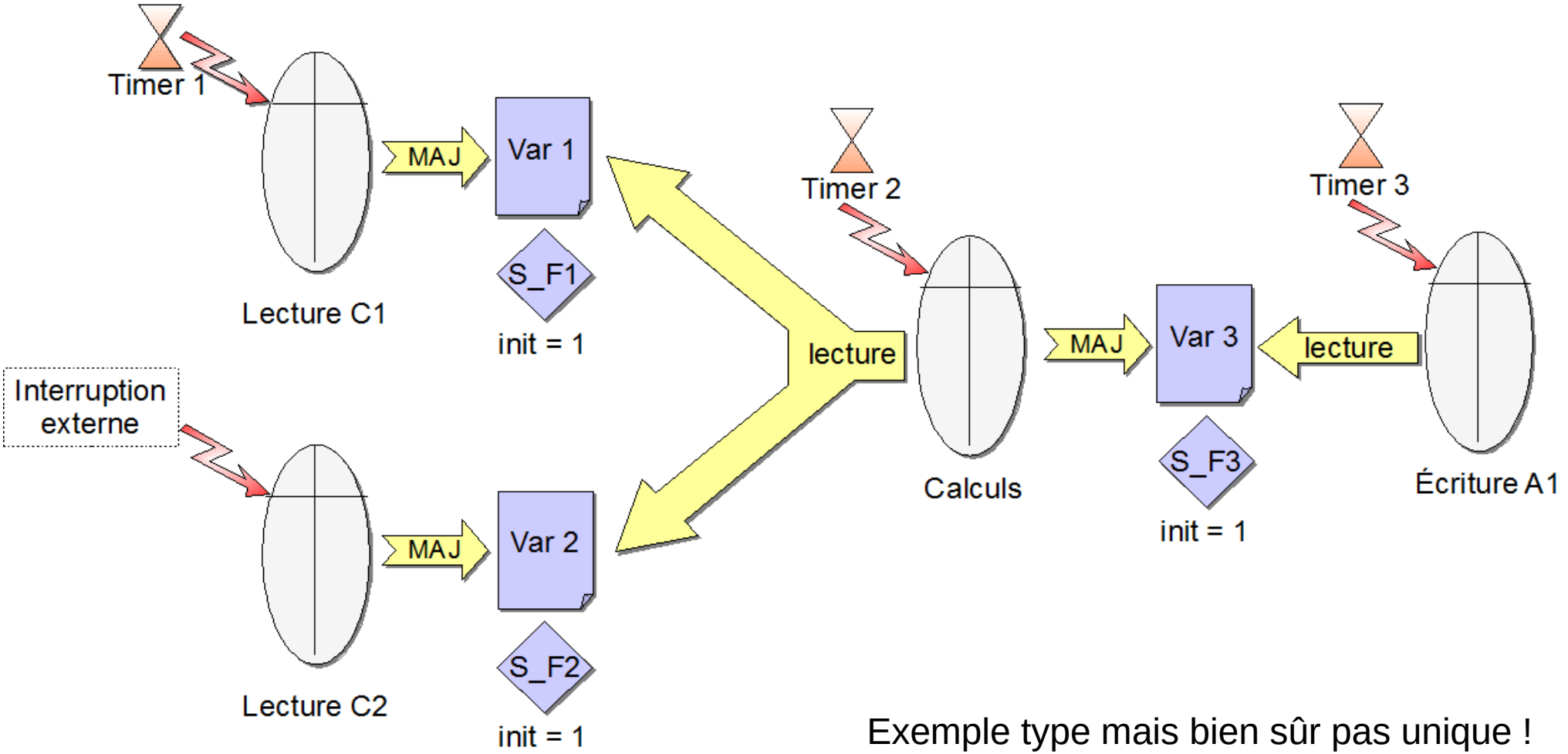
- Une tâche par préoccupation, exemple :
  - Lecture des capteurs : 1 (ou plusieurs) tâche
  - Calcul : 1 (ou plusieurs) tâches
  - Écriture sur actionneurs : 1 (ou plusieurs) tâche

- Chaque tâche a son propre rythme
  - Périodique (déclenchée par timer)
  - Déclenchée par des handlers d'interruptions externes

- **Éviter la création de tâche dynamique**

Programmation avec un RTOS
les tâches (2)

Principe :



Exemple type mais bien sûr pas unique !
Chaque cas est différent

Programmation avec un RTOS
les ressources

- Rôle
  - entité logique ou physique
  - peut être partagée par plusieurs tâches
- Objet non présent en tant que tel dans un OS
- Type de ressource
  - structure de données / zone mémoire
  - fichier
  - dispositif physique (*e.g.*, périphérique d'un micro contrôleur)
  - réseau

Programmation avec un RTOS les ressources

- Sémaphore d'exclusion mutuelle
- Passage en mode non préemptif ou super-priorité
- Masquage / démasquage des interruptions
  - Pas trop longtemps !
- Opération atomique
  - Difficile à assurer si on réutilise un OS existant

Choix d'un RTOS
critères

- Normes
  - SCEPTRE (1982)
  - POSIX (Unix)
  - OSEK/VDX (Automobile)
  - Profil MARTE de l'OMG ?
  - Autosar
- Domaine
  - Ferroviaire : l'OS préconisé est QNX
  - Avionique: Arinc
- Supports
  - Matériel : processeurs supportés, cartes, mémoire (4 Go sous CE)
  - Drivers (série, LCD, CAN), piles de protocole
  - Fournisseurs et suivi des versions

Choix d'un RTOS
critères

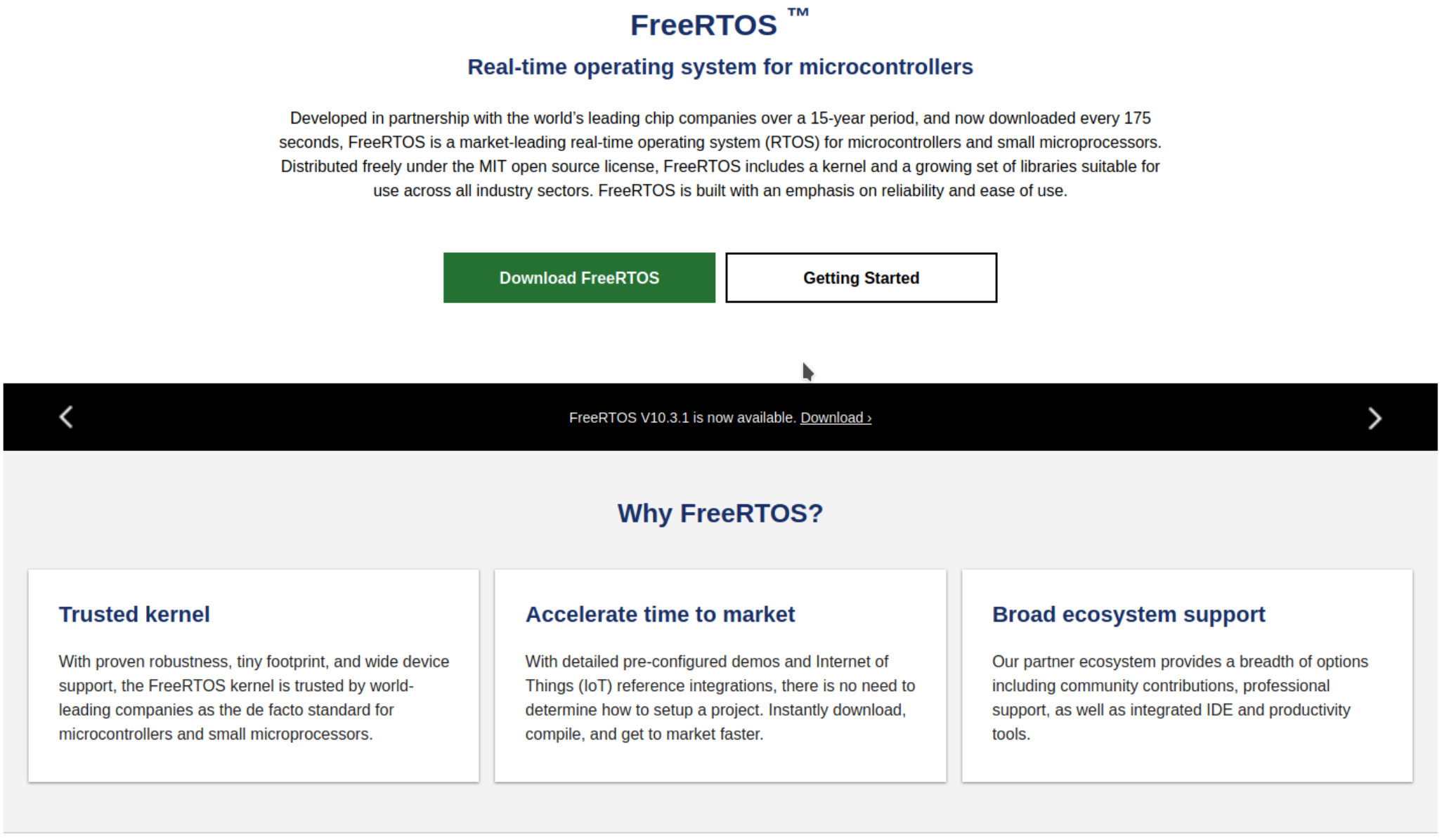
- Optimisation des ressources
  - Critères de taille et/ou de performance
  - Le comportement de l'applicatif est connu / estimé
  - Le comportement du support est configurable par l'application
- Réutilisation dans plusieurs contextes applicatifs
  - Services variés, services de haut niveau
  - Utilisation de librairies
- Adaptabilité pour les systèmes ouverts
  - Gestion dynamique de composants ou de services
    - Installation, ajout/retrait, …

- **Maintenabilité**
  - Accès au code
  - traçabilité des appels
- **Portabilité**
  - Respect de *normes*

Contenu du cours

- **Qu'est-ce qu'un OS temps réel (RTOS)**
  - RTOS vs GPOS
  - Différents RTOS...
  - Les objets *communs* d'un RTOS
  - Implémentation d'un RTOS
- Programmation avec un OS temps réel
- **Mise en Oeuvre**
  - **FreeRtos: http://freertos.org**

Mise en oeuvre



FreeRTOS ™

Real-time operating system for microcontrollers

Developed in partnership with the world's leading chip companies over a 15-year period, and now downloaded every 175 seconds, FreeRTOS is a market-leading real-time operating system (RTOS) for microcontrollers and small microprocessors. Distributed freely under the MIT open source license, FreeRTOS includes a kernel and a growing set of libraries suitable for use across all industry sectors. FreeRTOS is built with an emphasis on reliability and ease of use.

Download FreeRTOS | Getting Started

FreeRTOS V10.3.1 is now available. Download ›

Why FreeRTOS?

Trusted kernel

With proven robustness, tiny footprint, and wide device support, the FreeRTOS kernel is trusted by world-leading companies as the de facto standard for microcontrollers and small microprocessors.

Accelerate time to market

With detailed pre-configured demos and Internet of Things (IoT) reference integrations, there is no need to determine how to setup a project. Instantly download, compile, and get to market faster.

Broad ecosystem support

Our partner ecosystem provides a breadth of options including community contributions, professional support, as well as integrated IDE and productivity tools.

Mise en oeuvre

FreeRTOS

Features

Tiny, power-saving kernel

Scalable size, with usable program memory footprint as low as 9KB. Some architectures include a tick-less power saving mode

Support for 40+ architectures

One code base for 40+ MCU architectures and 15+ toolchains, including the latest RISC-V and ARMv8-M (Arm Cortex-M33) microcontrollers

Modular libraries

A growing number of add-on libraries used across all industries sectors, including secure local or cloud connectivity

IoT Reference Integrations

Take advantage of tested examples that include all the libraries essential to securely connect to the cloud

MIT licensed, with options

FreeRTOS can be used for any purpose under its MIT license. Our strategic partner also provides commercial licenses, and safety certification.

Mise en oeuvre

FreeRTOS

Features

Tiny, power-saving kernel

Scalable size, with usable program memory footprint as low as 9KB. Some architectures include a tick-less power saving mode

Support for 40+ architectures

One code base for 40+ MCU architectures and 15+ toolchains, including the latest RISC-V and ARMv8-M (Arm Cortex-M33) microcontrollers

Modular libraries

A growing number of add-on libraries used across all industries sectors, including secure local or cloud connectivity

IoT Reference Integrations

Take advantage of tested examples that include all the libraries essential to securely connect to the cloud

MIT licensed, with options

FreeRTOS can be used for any purpose under its MIT license. Our strategic partner also provides commercial licenses, and safety certification.

→ **Mais nous ne verrons que les choses très basiques**

Conclusion

micro-contrôleur avec un RTOS

- Des concepts génériques...
  - Tâches, sémaphores, etc
- … mais mises en œuvre spécifiques

➔ **Bien lire les spécifications**

- Implémentation
  - Liée aux spécificités du matériel (mémoire, IT)
    - ➔ Une couche de portage spécifique par cible
  - Simple et prédictible en temps
    - ➔ Allocations statiques

➔ **Maitrise du HW, prédiction a priori**

- Choix selon l'utilisation
  - Critères : coût (environnement et royalties, formation des équipes de développement), accès au code, développements spécifiques, taille, prédictibilité, durée de vie du produit, qualité des fournisseurs

➔ **Phase à ne pas négliger**