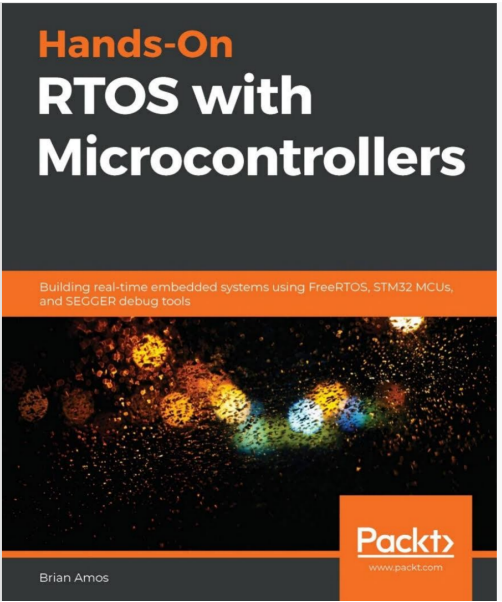
Ecole Nationale
Supérieure
de l'Electronique
et de ses Applications

Noyaux Temps-réel

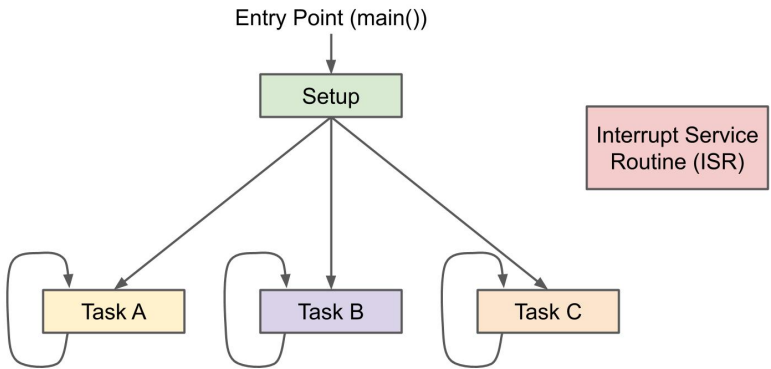
Les tâches dans FreeRTOS

https://www.freertos.org/a00106.html

Laurent Fiack
Bureau D212/D060 – laurent.fiack@ensea.fr

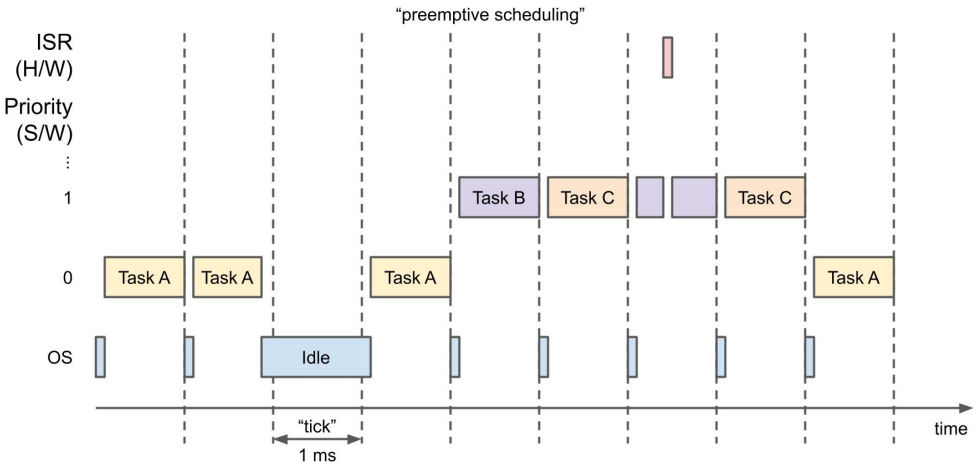

Hands-On
RTOS with
Microcontrollers
Building real-time embedded systems using FreeRTOS, STM32 MCUs, and SEGGER debug tools
Packt
www.packt.com
Brian Amos

What our code looks like

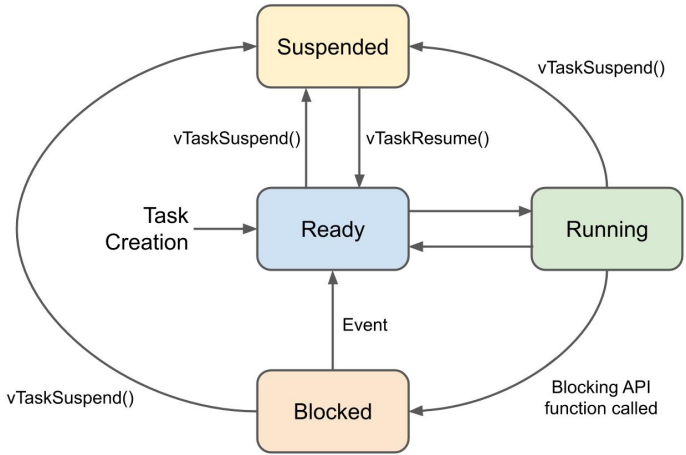


What actually happens*

*assuming single-core processor



Task States


Suspended
vTaskSuspend()
vTaskSuspend()
vTaskResume()
Task
Creation
Ready
Running
Event
vTaskSuspend()
Blocked
Blocking API
function called

Les primitives de gestion des tâches

- Création d'une tâche

```
BaseType_t xTaskCreate (
    TaskFunction_t pvTaskCode,
    const char * const pcName,
    configSTACK_DEPTH_TYPE usStackDepth,
    void *pvParameters,
    UBaseType_t uxPriority,
    TaskHandle_t *pxCreatedTask
);
```

  - Priority de 0 à `configMAX_PRIORITIES - 1` (FreeRTOSConfig.h)
  - TaskHandle permet de retrouver et manipuler la tâche
  - Paramètre de retour :
    - `pdPass` si succes
    - `errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY` sinon

Les primitives de gestion des tâches (suite)

- Destruction d'une tâche

```
void vTaskDelete (TaskHandle_t xTask)
```

- Mise en sommeil d'une tâche

```
void vTaskDelay (const TickType_t xTicksToDelay)
```

- Gestion des priorités

```
void vTaskPrioritySet (TaskHandle_t xTask, UBaseType_t uxNewPriority)
```

```
UBaseType_t xTaskPriorityGet (TaskHandle_t xTask)
```

D'autres primitives

```
void vTaskSuspend (TaskHandle_t)
```

```
void vTaskResume (TaskHandle_t)
```

```
UBaseType_t xTaskResumeFromISR (TaskHandle_t)
```

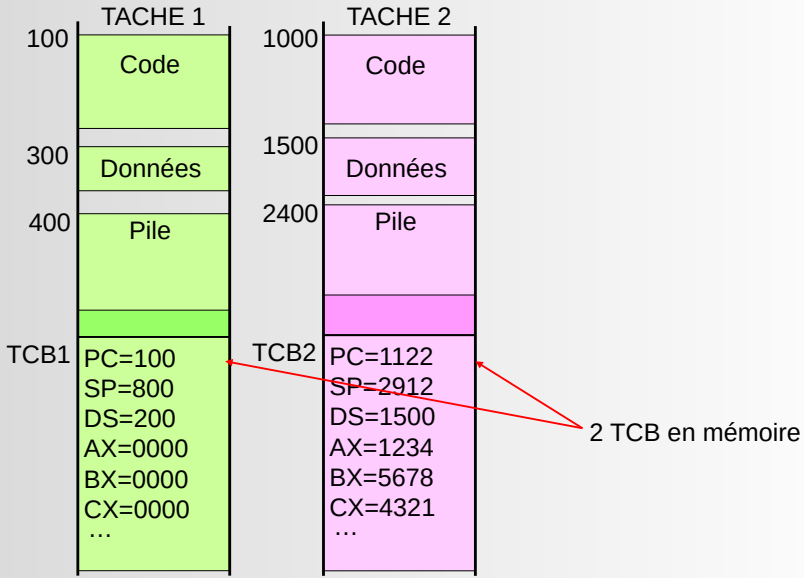
```
UbaseType_t vTaskAbortDelay (TaskHandle_t)
```

```
TaskHandle_t vTaskGetCurrentHandle (void)
```

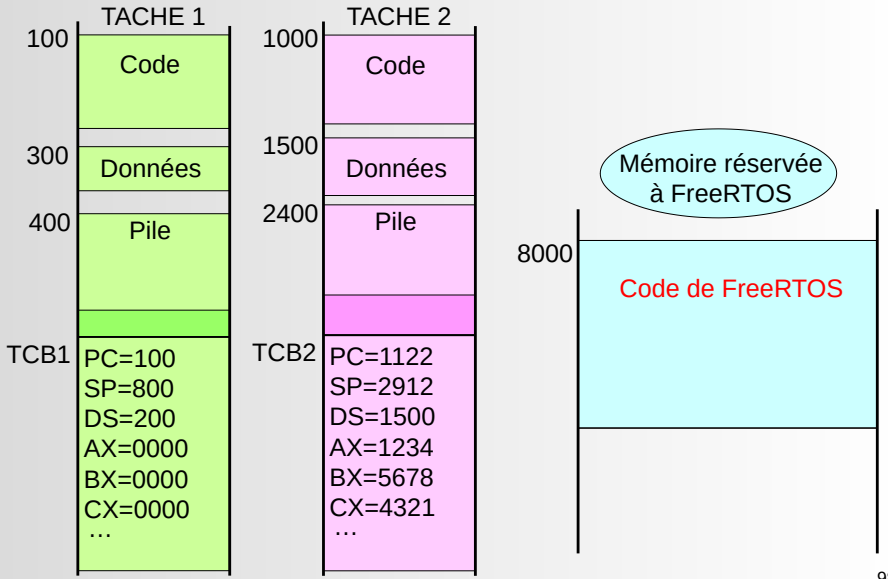
```
TaskHandle_t vTaskGetHandle (const char * pcNameToQuery)
```

- https://www.freertos.org/a00106.html

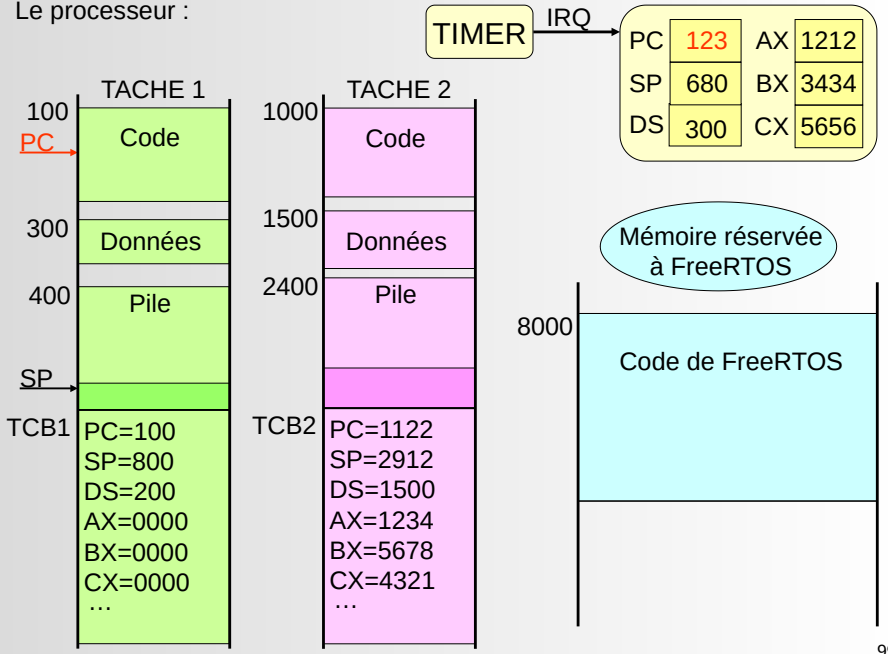
- La commutation de contexte



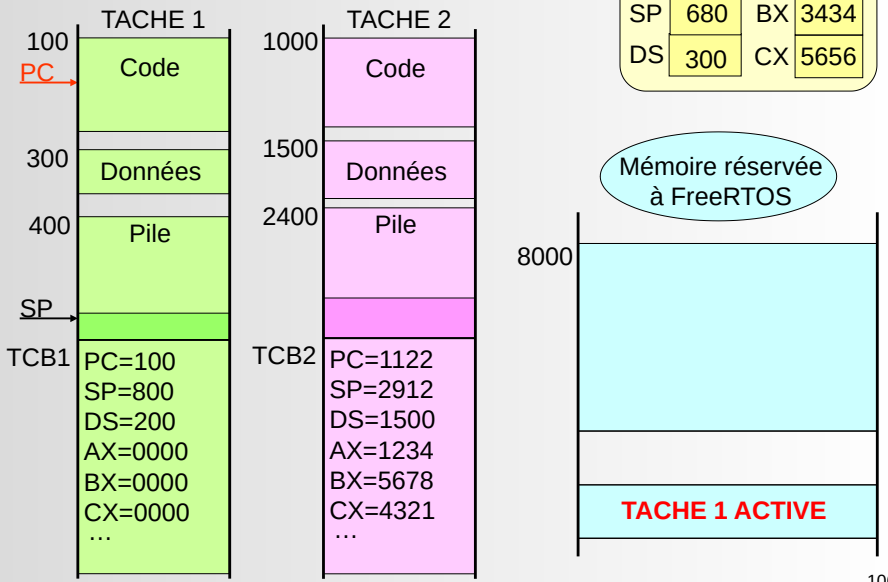
La mémoire réservée à FreeRTOS :



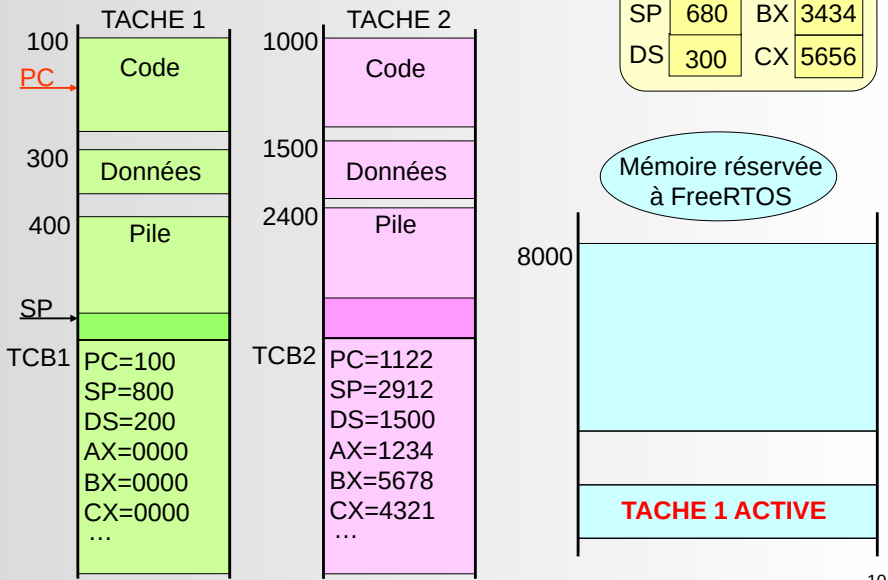
Le processeur :



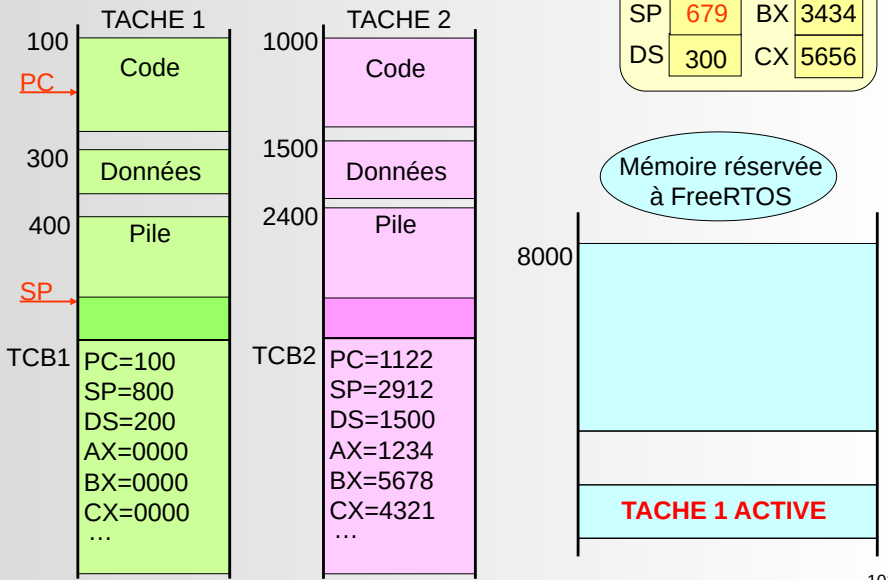
La tâche 1 est active



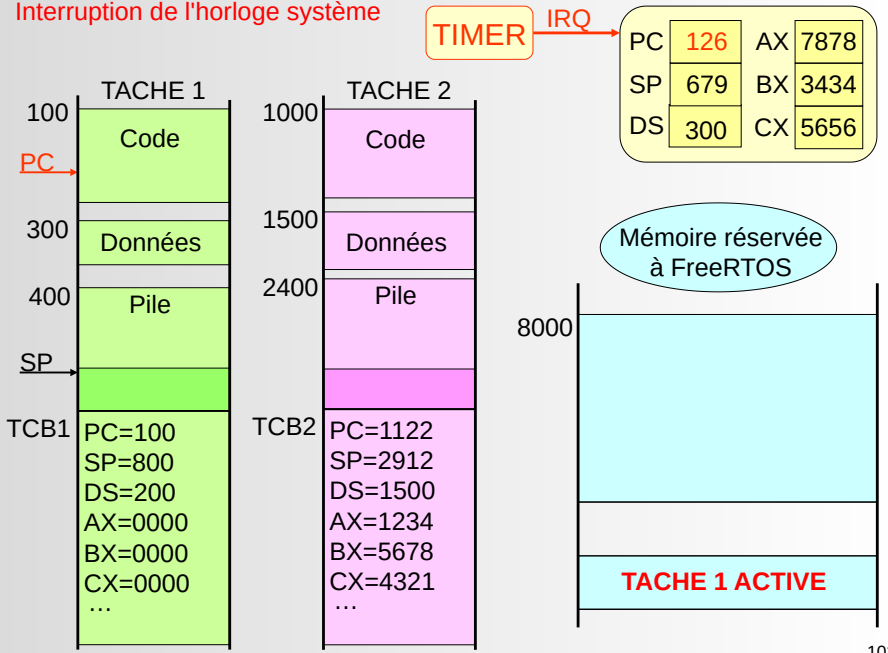
La tâche 1 est active



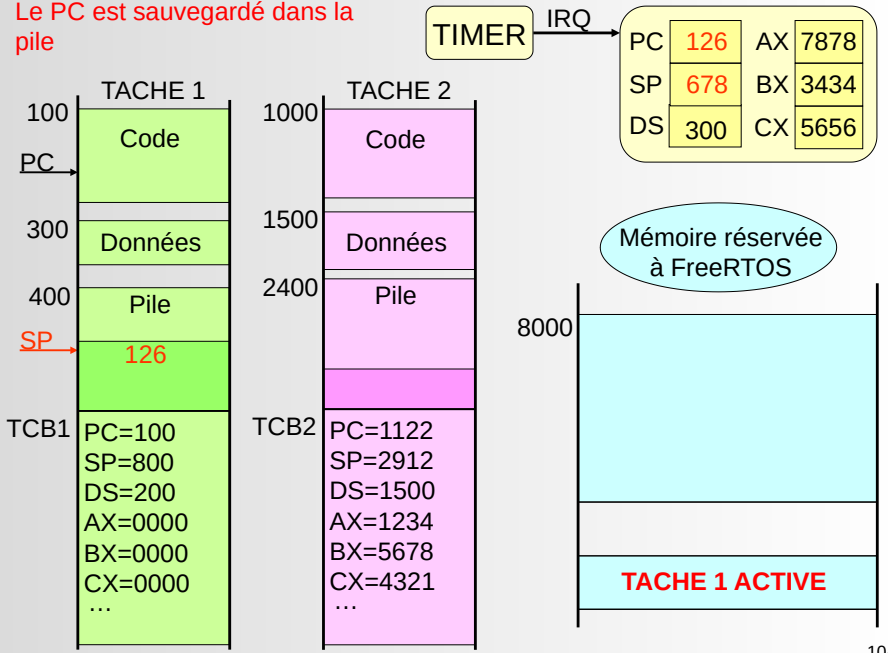
La tâche 1 est active



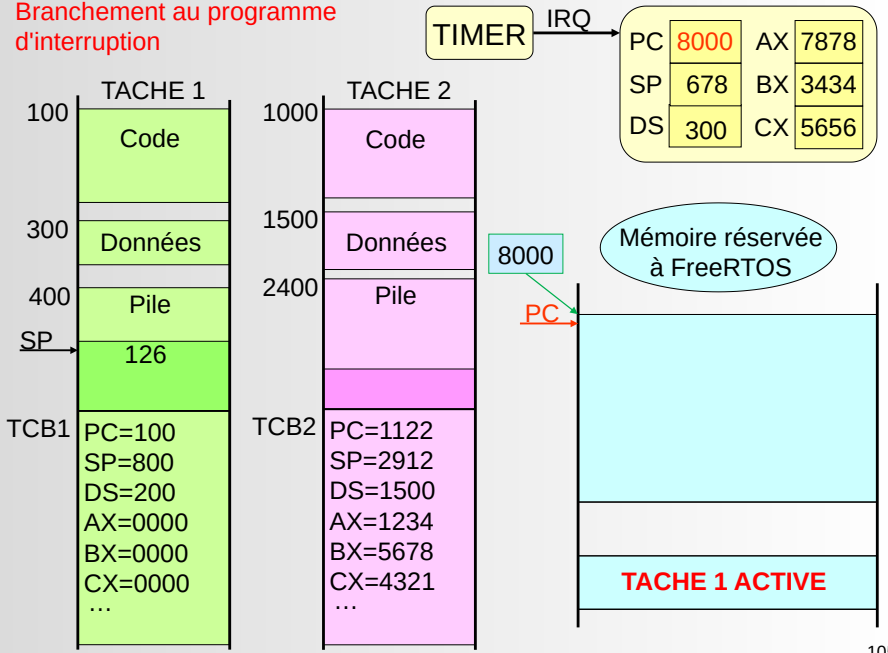
Interruption de l'horloge système



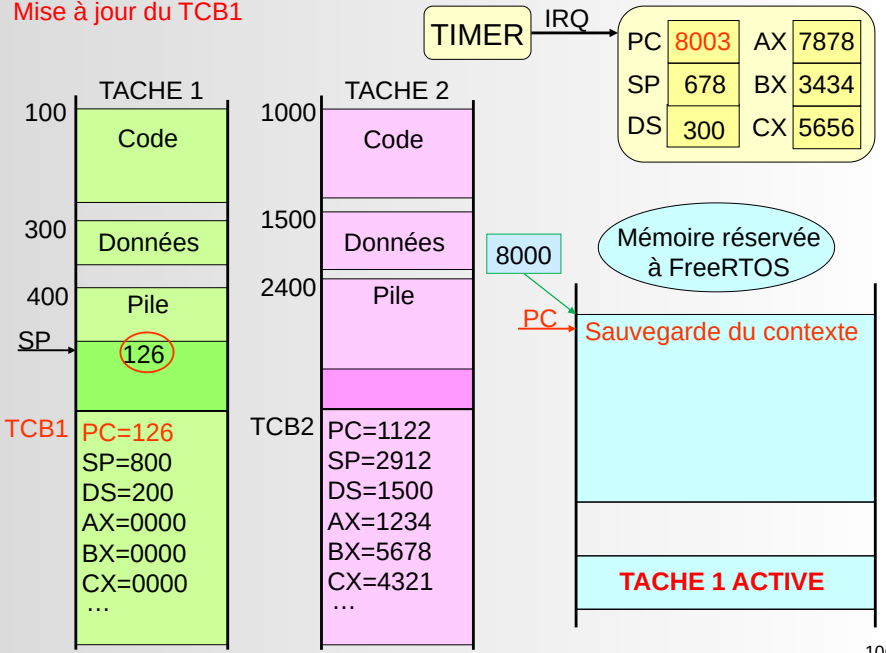
Le PC est sauvegardé dans la pile



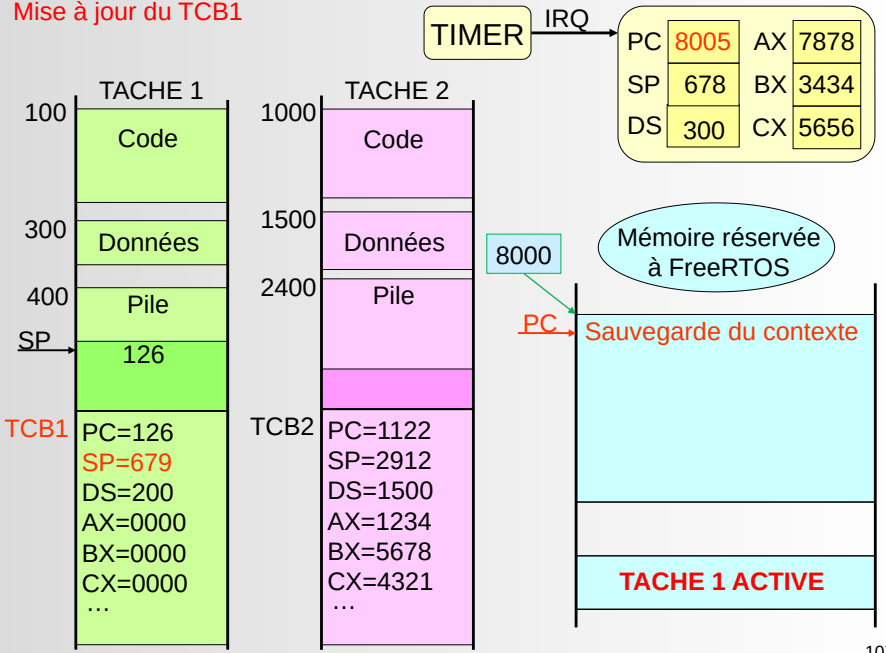
Branchement au programme d'interruption



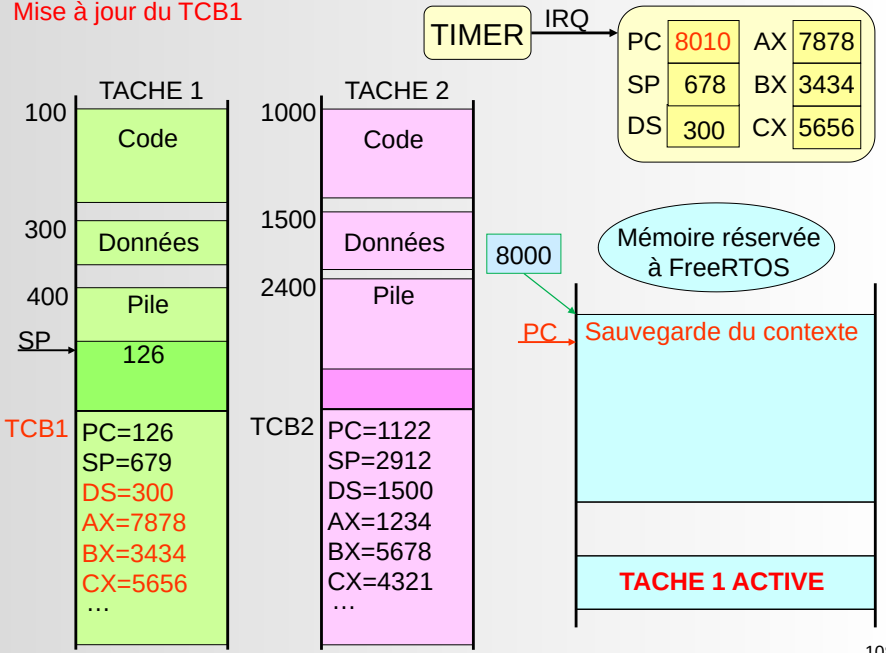
Mise à jour du TCB1



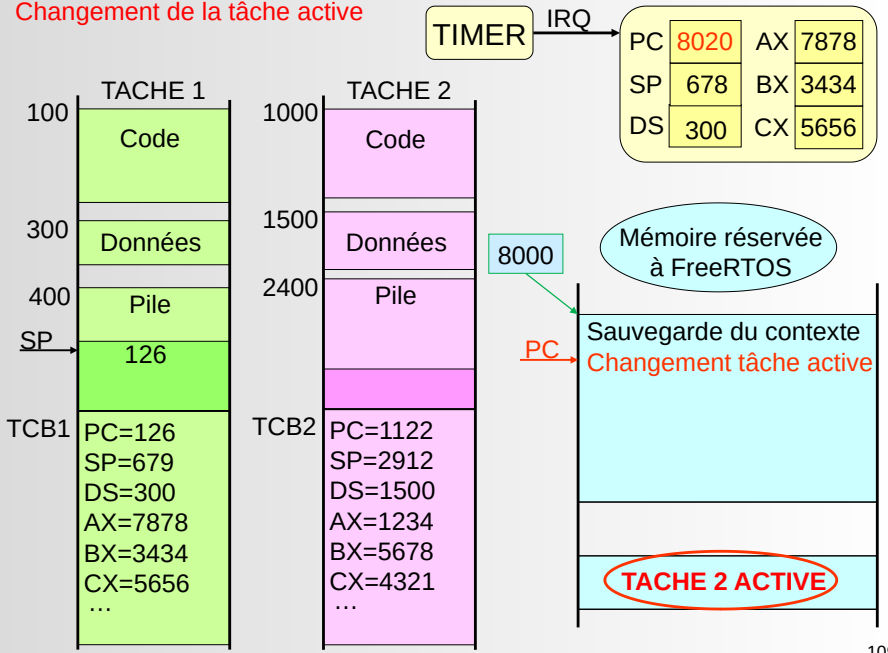
Mise à jour du TCB1



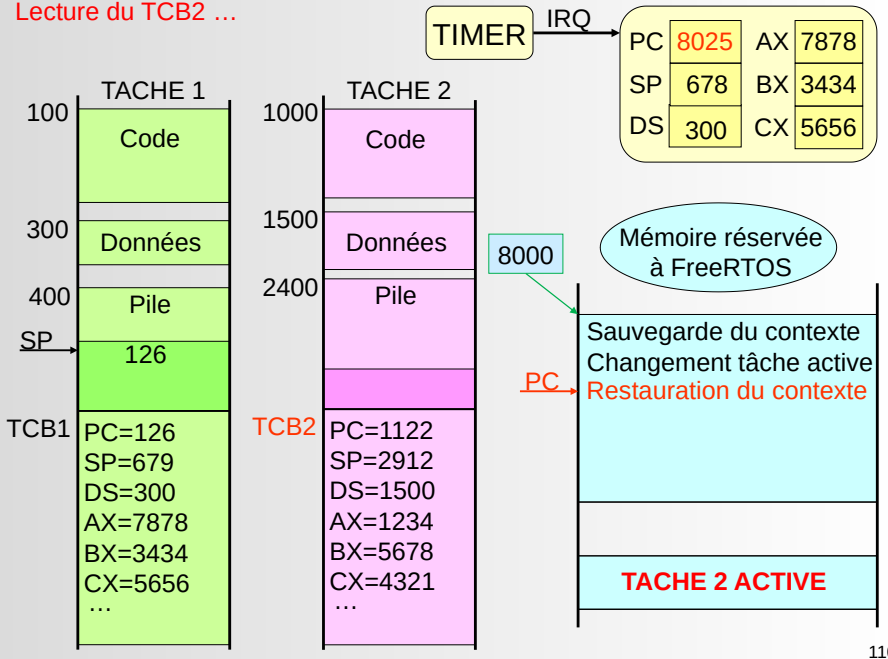
Mise à jour du TCB1



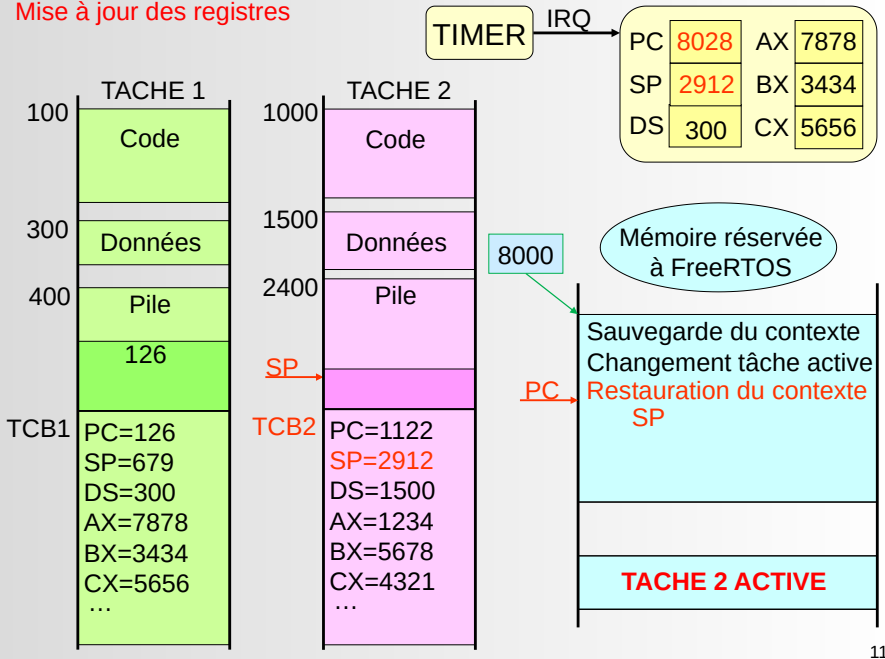
Changement de la tâche active



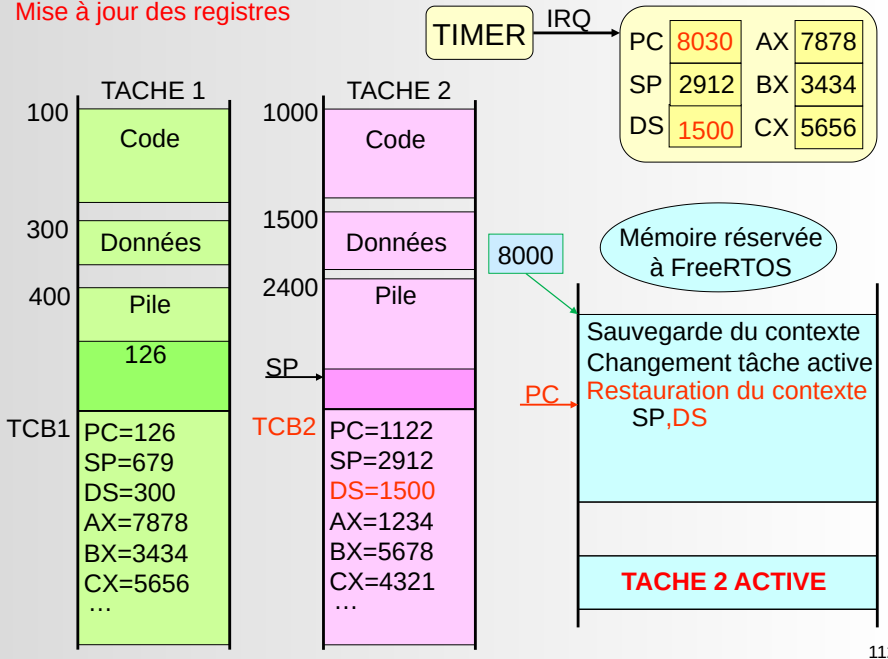
Lecture du TCB2 …



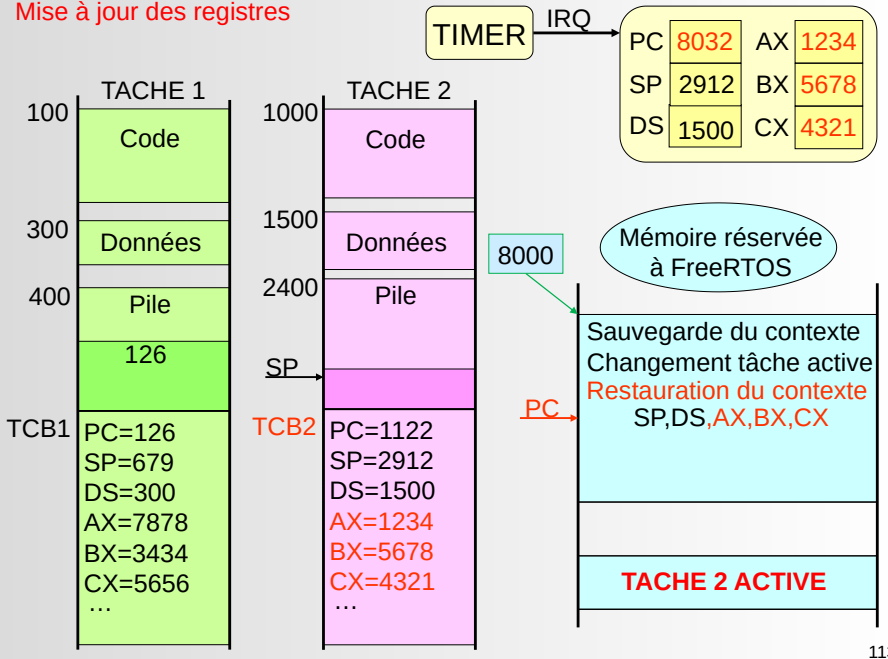
Mise à jour des registres



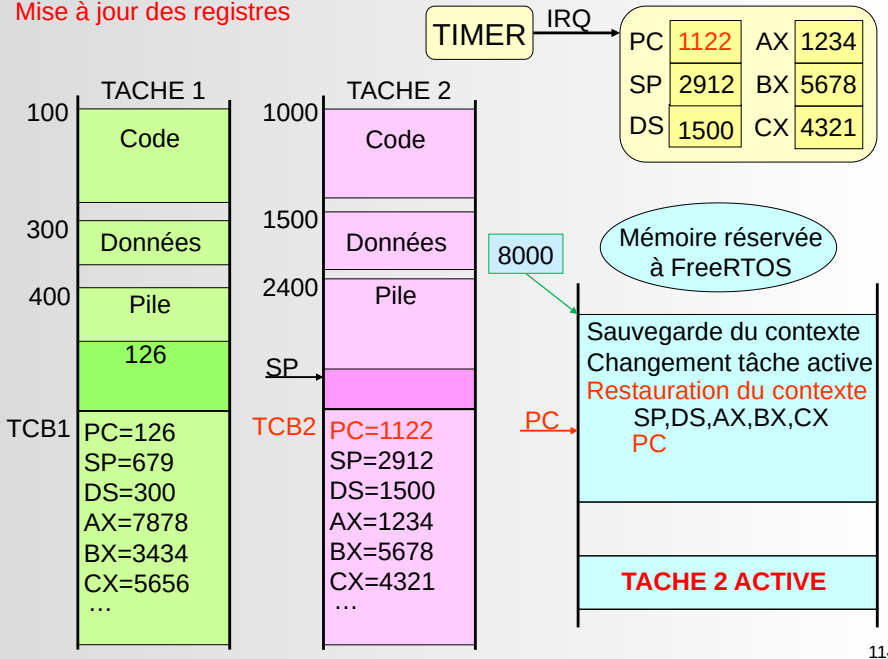
Mise à jour des registres



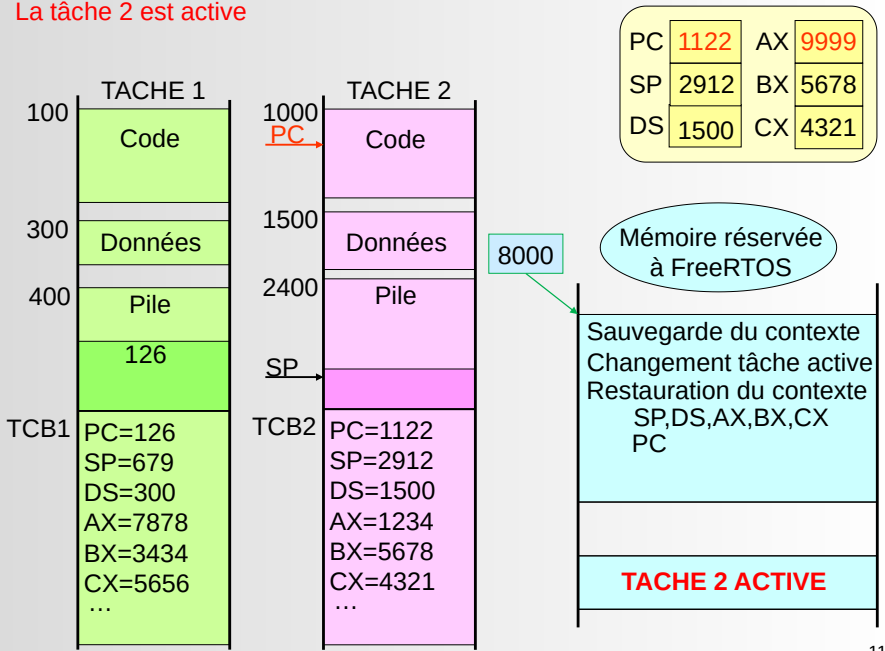
Mise à jour des registres



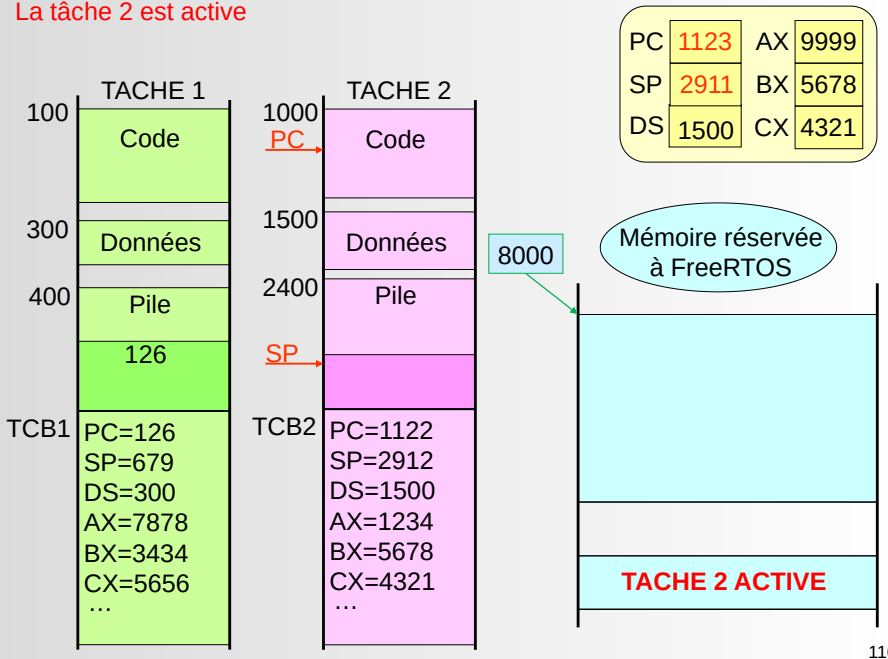
Mise à jour des registres



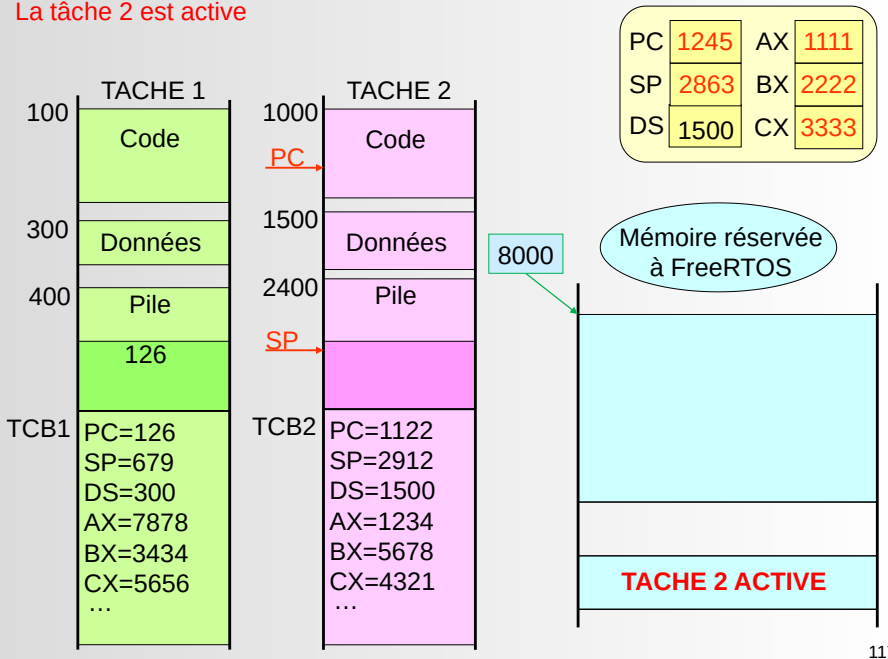
La tâche 2 est active



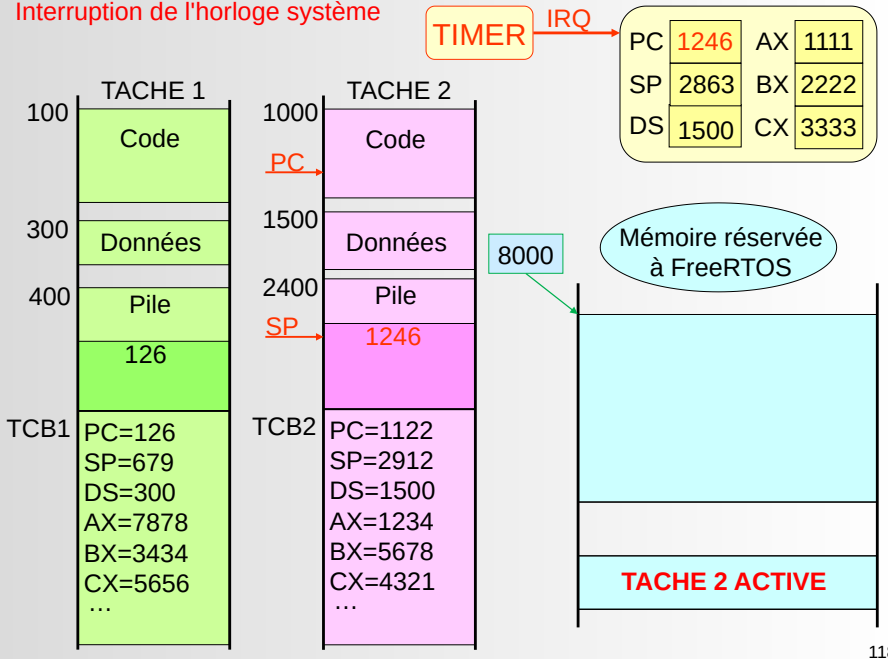
La tâche 2 est active



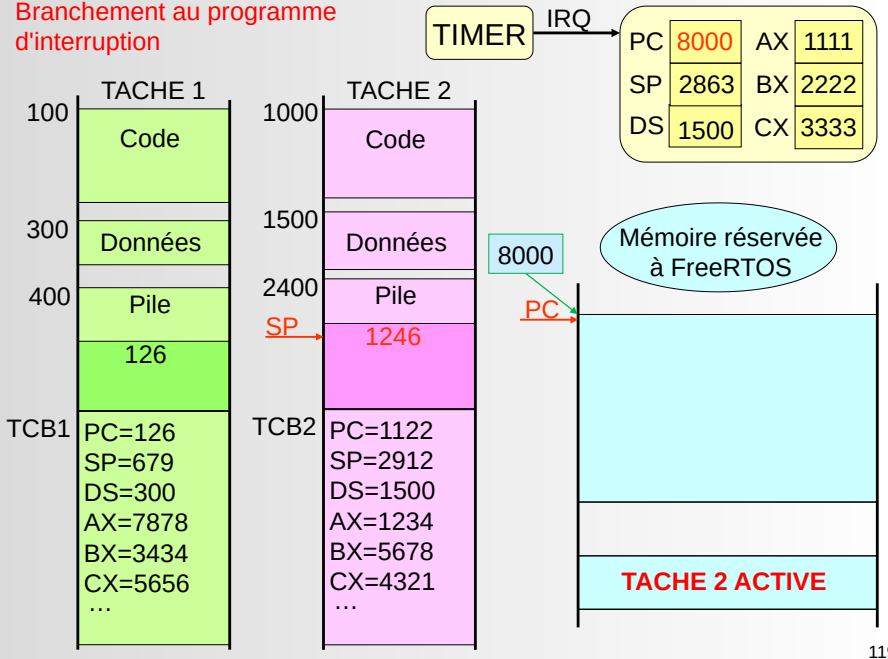
La tâche 2 est active



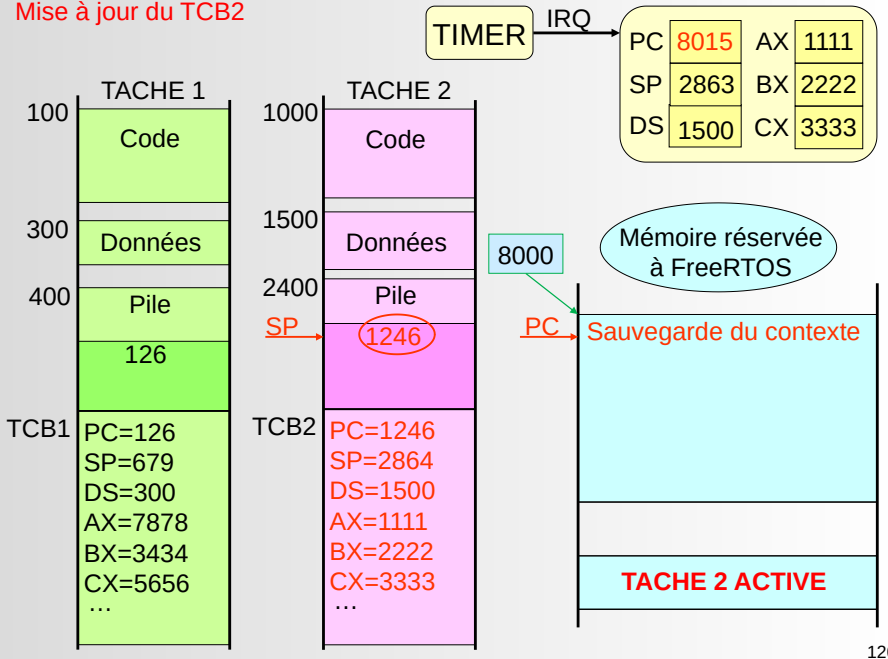
Interruption de l'horloge système



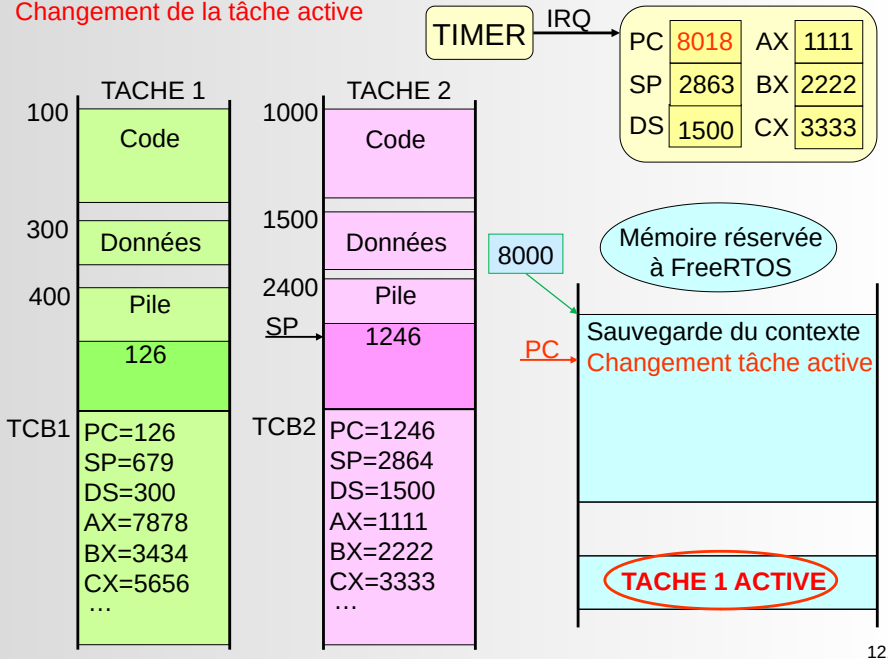
Branchement au programme d'interruption



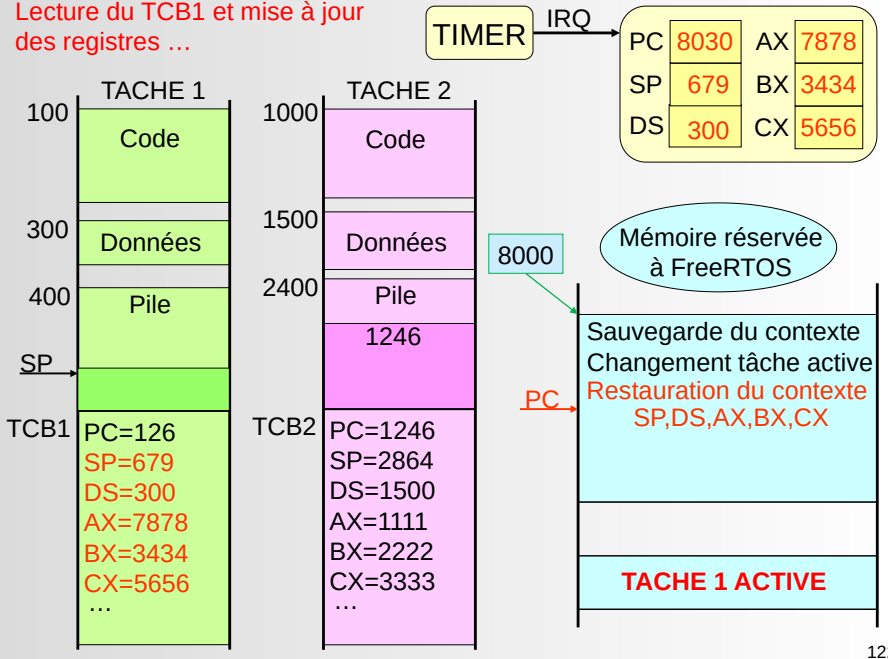
Mise à jour du TCB2



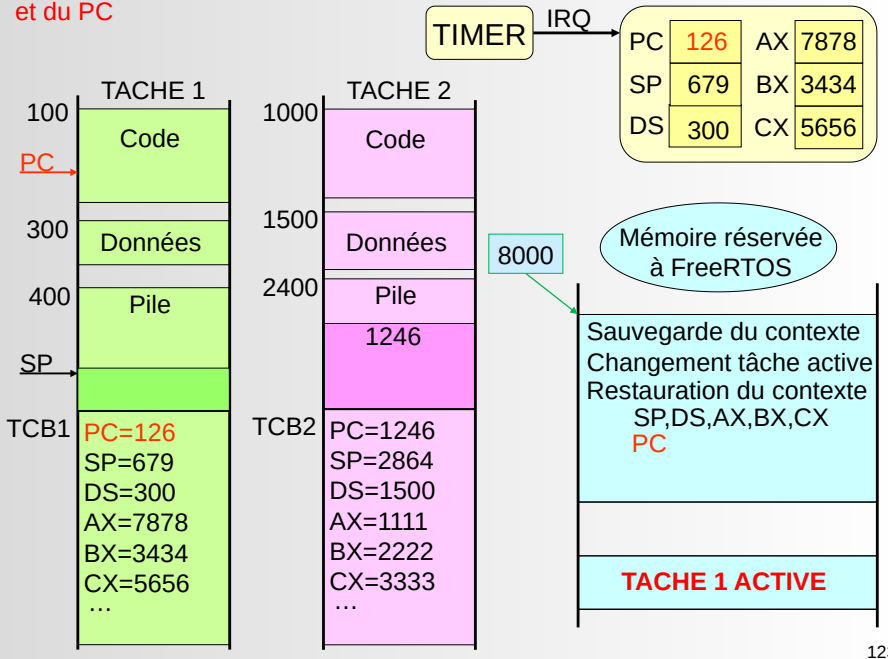
Changement de la tâche active



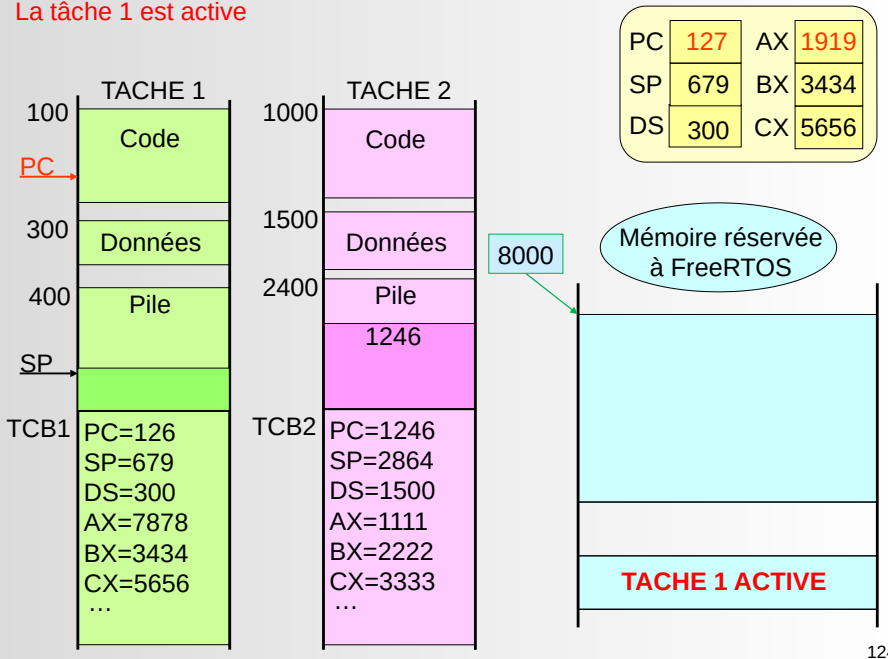
Lecture du TCB1 et mise à jour des registres …



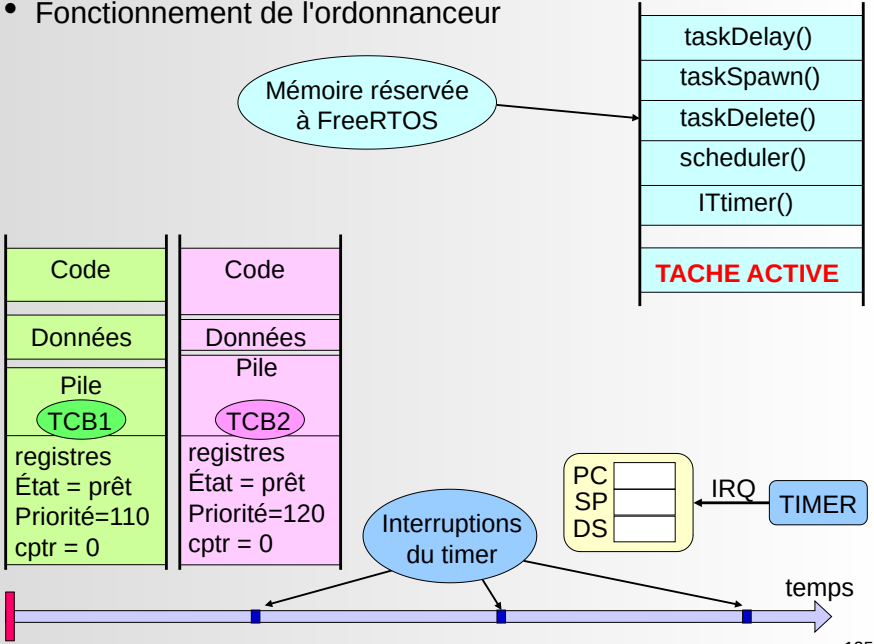
et du PC


TIMER
IRQ
PC 126
SP 679
DS 300
AX 7878
BX 3434
CX 5656
TACHE 1
100
Code
PC
300
Données
400
Pile
SP
TCB1
PC=126
SP=679
DS=300
AX=7878
BX=3434
CX=5656
…
TACHE 2
1000
Code
1500
Données
2400
Pile
1246
TCB2
PC=1246
SP=2864
DS=1500
AX=1111
BX=2222
CX=3333
…
8000
Mémoire réservée
à FreeRTOS
Sauvegarde du contexte
Changement tâche active
Restauration du contexte
SP,DS,AX,BX,CX
PC
TACHE 1 ACTIVE

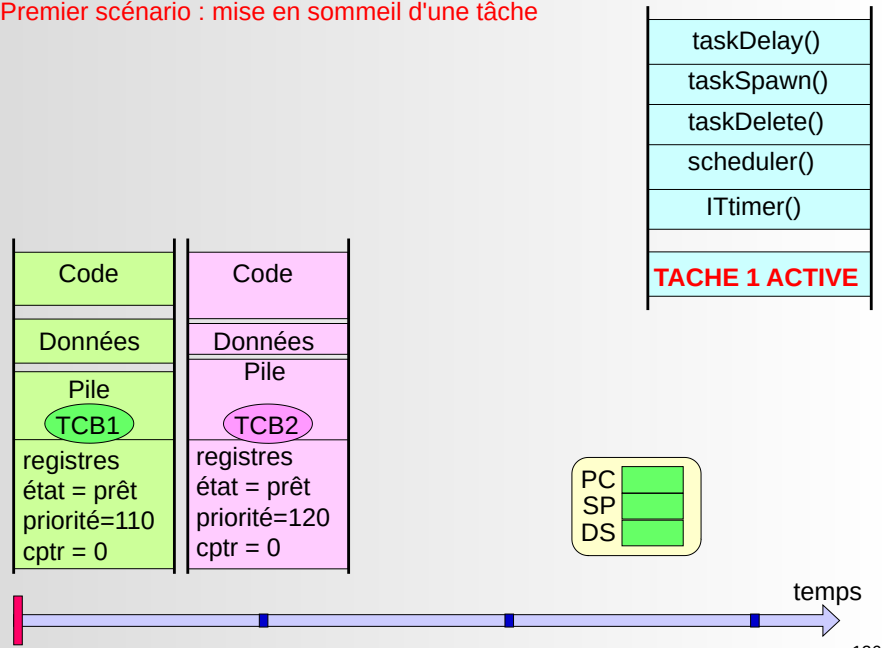
La tâche 1 est active



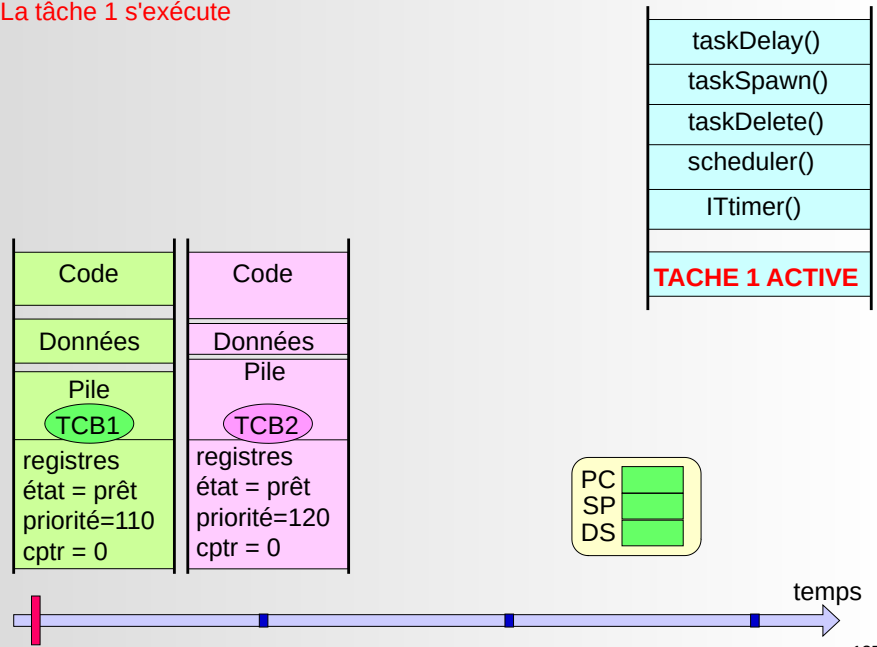
- Fonctionnement de l'ordonnanceur



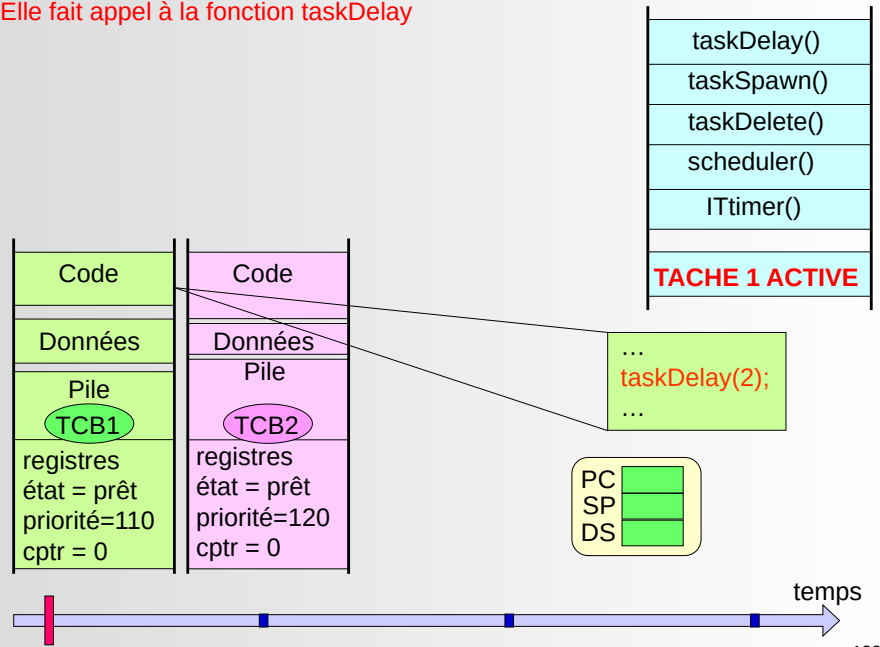
Premier scénario : mise en sommeil d'une tâche



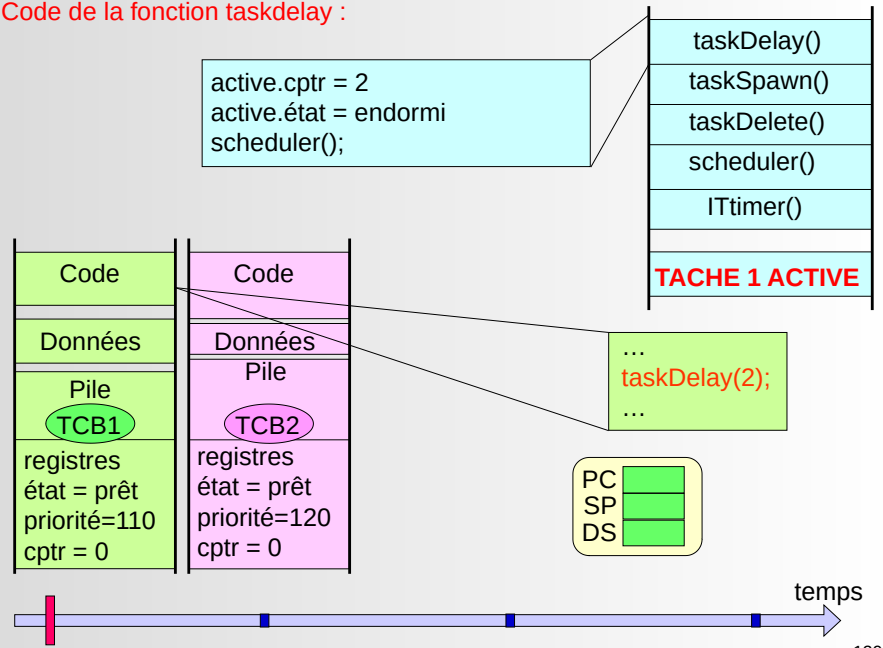
La tâche 1 s'exécute



Elle fait appel à la fonction taskDelay

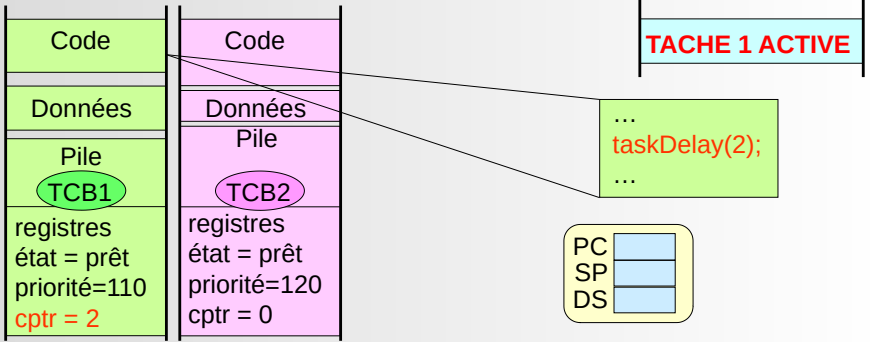


Code de la fonction taskdelay :

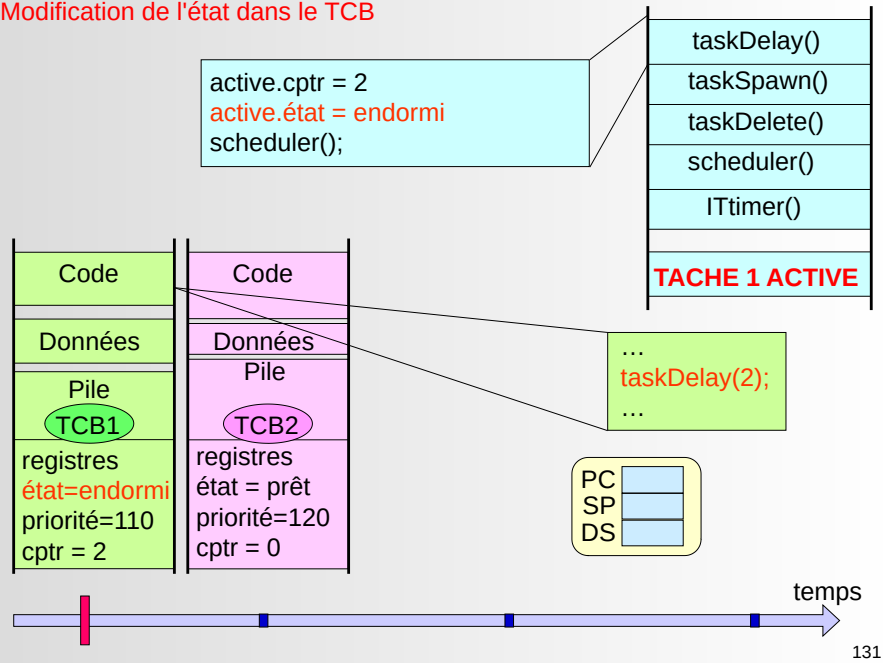


Initialisation du compteur de sommeil de la tâche active dans le TCB

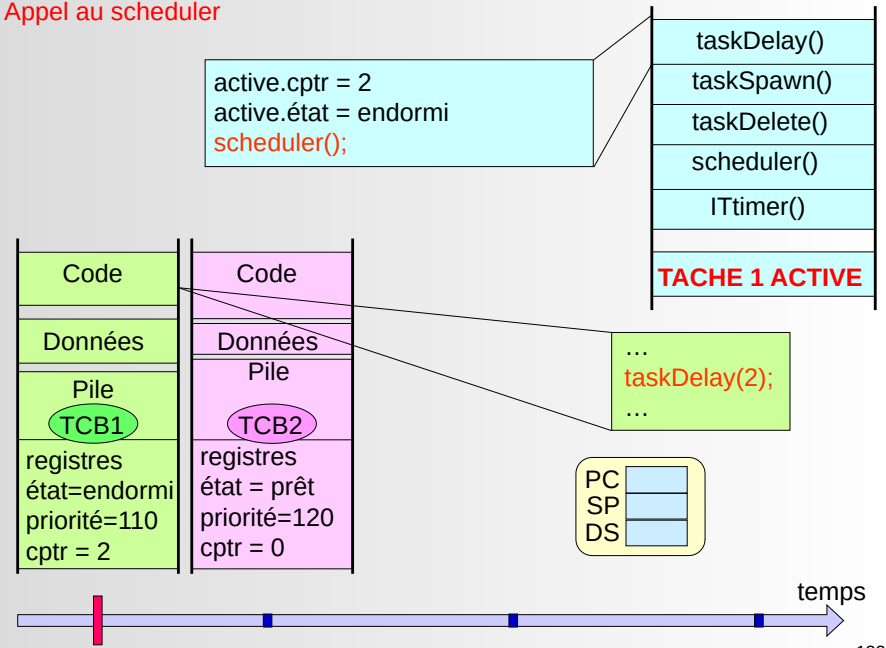
```
active.cptr = 2
active.état = endormi
scheduler();
```



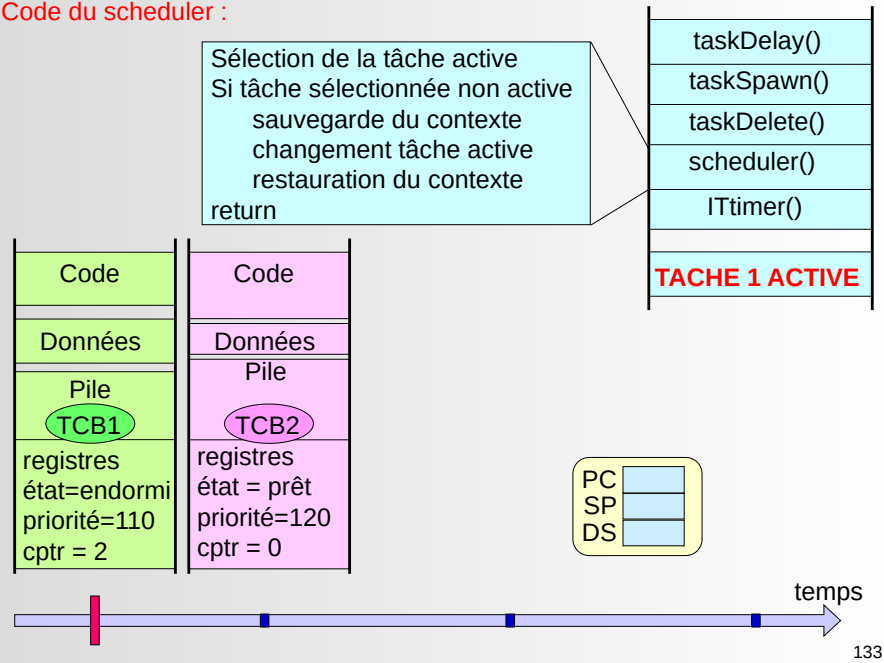
Modification de l'état dans le TCB



Appel au scheduler



Code du scheduler :



Sélection de la tâche qui deviendra active

Sélection de la tâche active
Si tâche sélectionnée non active
sauvegarde du contexte
changement tâche active
restauration du contexte
return

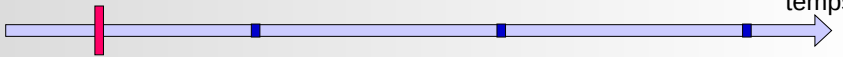
taskDelay()
taskSpawn()
taskDelete()
scheduler()
ITtimer()

TACHE 1 ACTIVE

Code	Code
Données	Données
Pile	Pile
TCB1	TCB2
registres état=endormi priorité=110 cptr = 2	registres état = prêt priorité=120 cptr = 0

TACHE 2

PC
SP
DS



Si la tâche sélectionnée n'est pas celle qui était active

Sélection de la tâche active
Si tâche sélectionnée non active
sauvegarde du contexte
changement tâche active
restauration du contexte
return

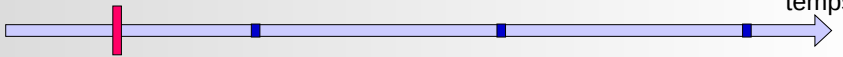
taskDelay()
taskSpawn()
taskDelete()
scheduler()
ITtimer()

TACHE 1 ACTIVE

TACHE 2

Code	Code
Données	Données
Pile	Pile
TCB1	TCB2
registres état=endormi priorité=110 cptr = 2	registres état = prêt priorité=120 cptr = 0

PC
SP
DS



Sauvegarde du contexte dans le TCB

Sélection de la tâche active
Si tâche sélectionnée non active
    sauvegarde du contexte
    changement tâche active
    restauration du contexte
return

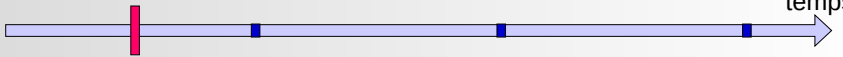
taskDelay()
taskSpawn()
taskDelete()
scheduler()
ITtimer()

TACHE 1 ACTIVE

TACHE 2

Code	Code
Données	Données
Pile	Pile
TCB1	TCB2
registres état=endormi priorité=110 cptr = 2	registres état = prêt priorité=120 cptr = 0

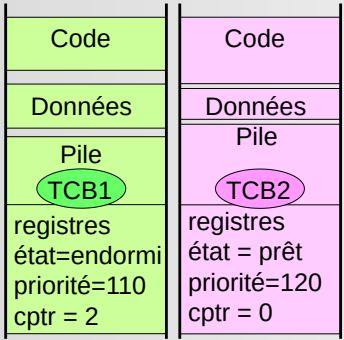
PC
SP
DS

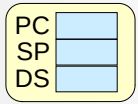


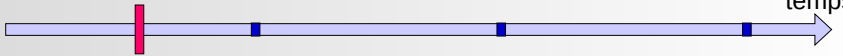
Mise à jour de la tâche active

Sélection de la tâche active
Si tâche sélectionnée non active
    sauvegarde du contexte
    changement tâche active
    restauration du contexte
return

taskDelay()
taskSpawn()
taskDelete()
scheduler()
ITtimer()
TACHE 2 ACTIVE







Les registres sont remis à jour à partir du TCB de la tâche active

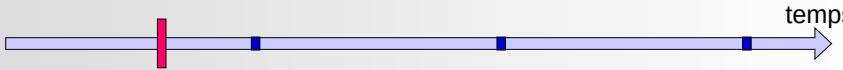
Sélection de la tâche active
Si tâche sélectionnée non active
    sauvegarde du contexte
    changement tâche active
    restauration du contexte
return

taskDelay()
taskSpawn()
taskDelete()
scheduler()
ITtimer()

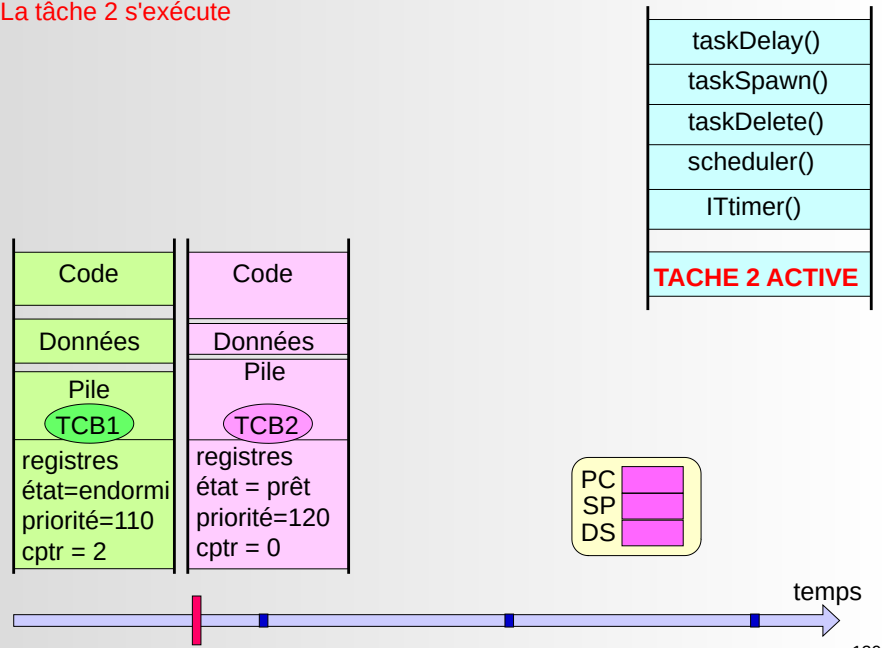
TACHE 2 ACTIVE

Code	Code
Données	Données
Pile	Pile
TCB1	TCB2
registres état=endormi priorité=110 cptr = 2	registres état = prêt priorité=120 cptr = 0

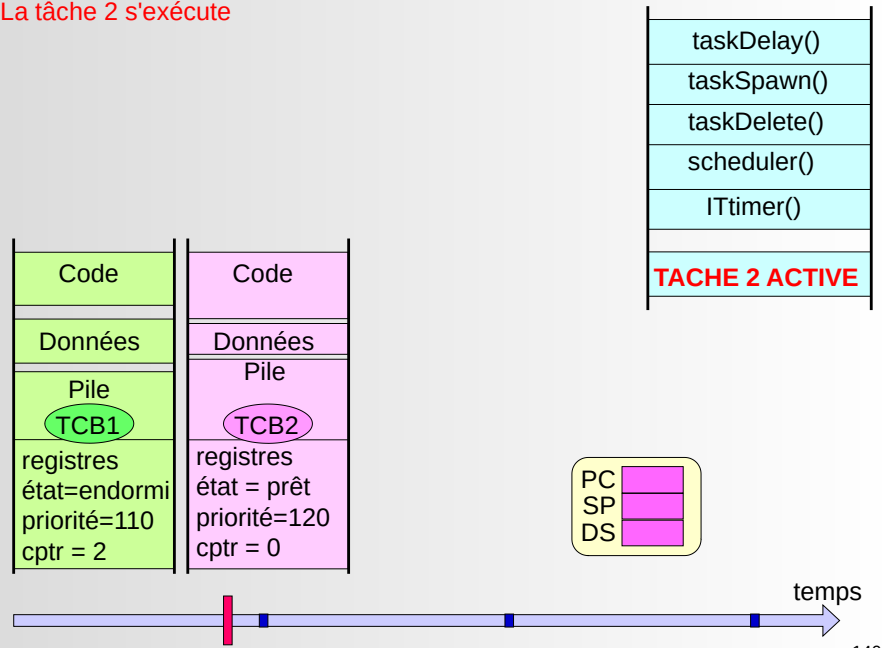
PC
SP
DS



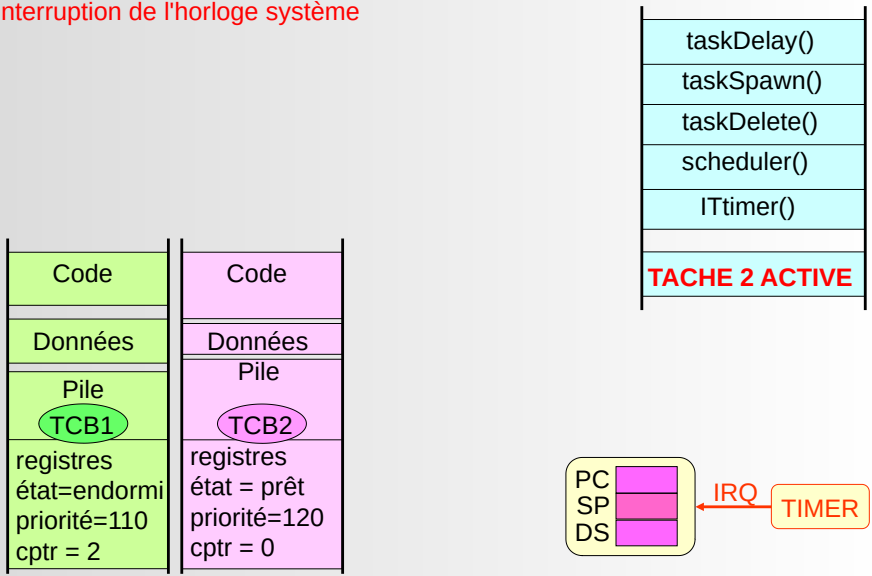
La tâche 2 s'exécute



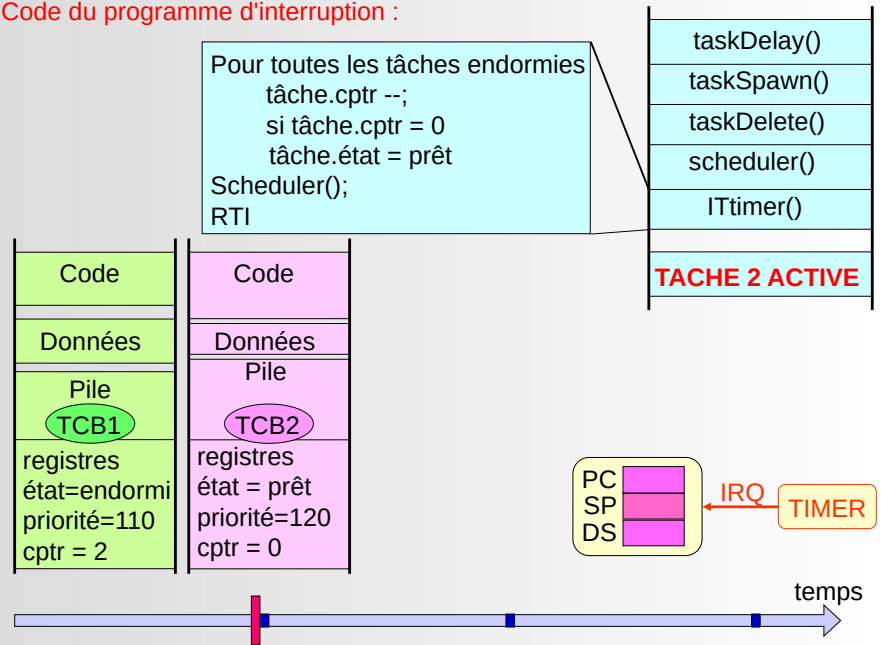
La tâche 2 s'exécute



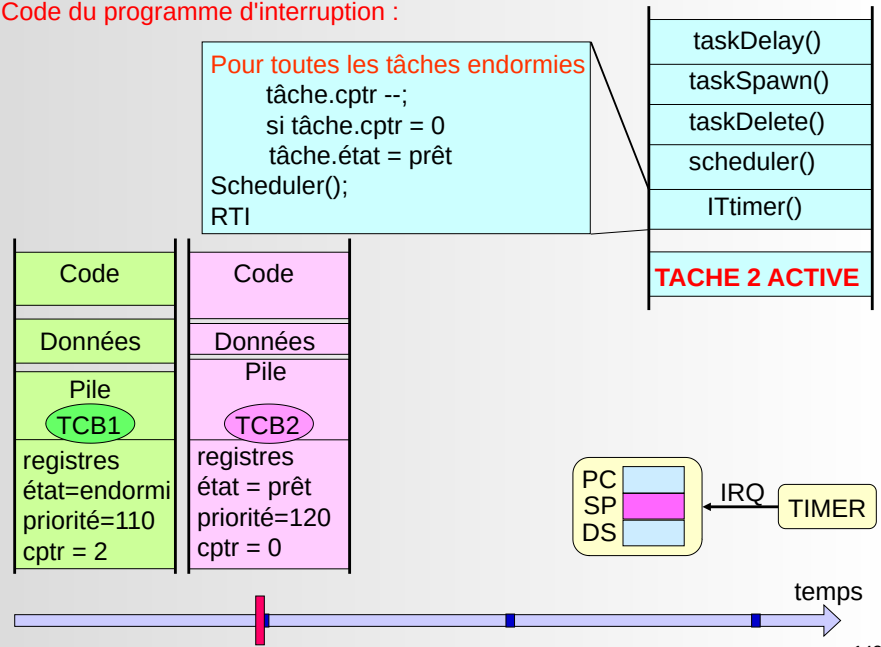
Interruption de l'horloge système



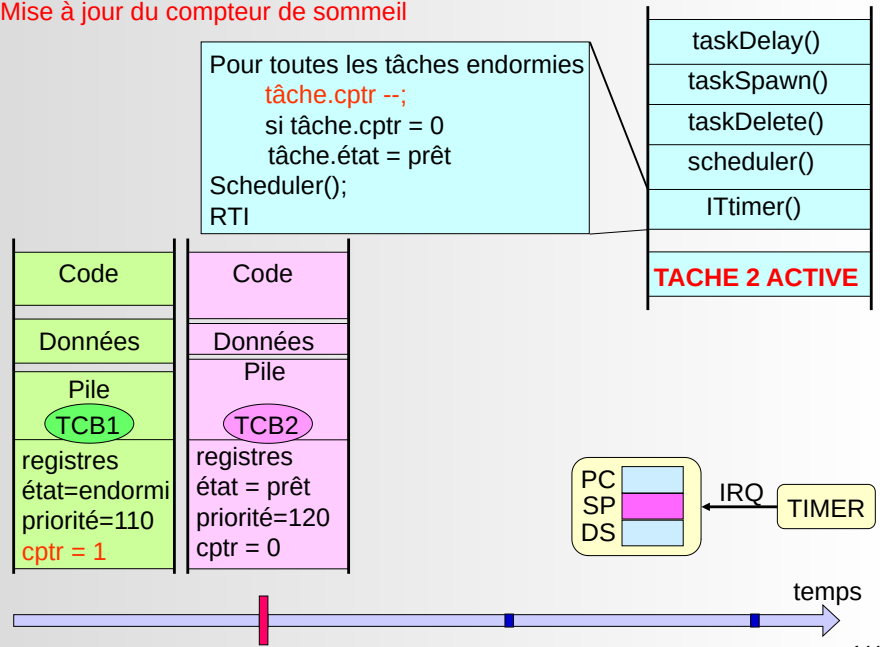
Code du programme d'interruption :



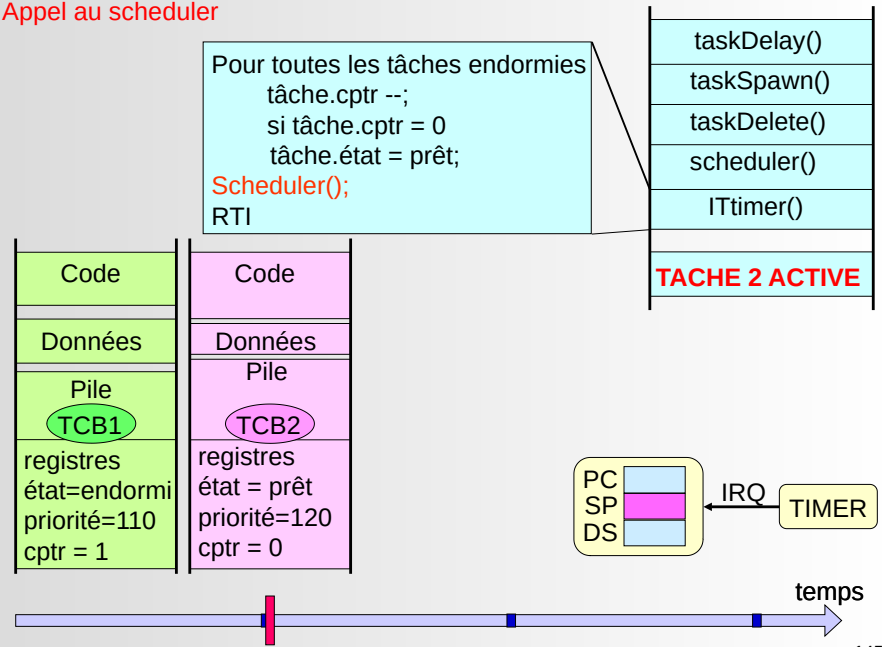
Code du programme d'interruption :



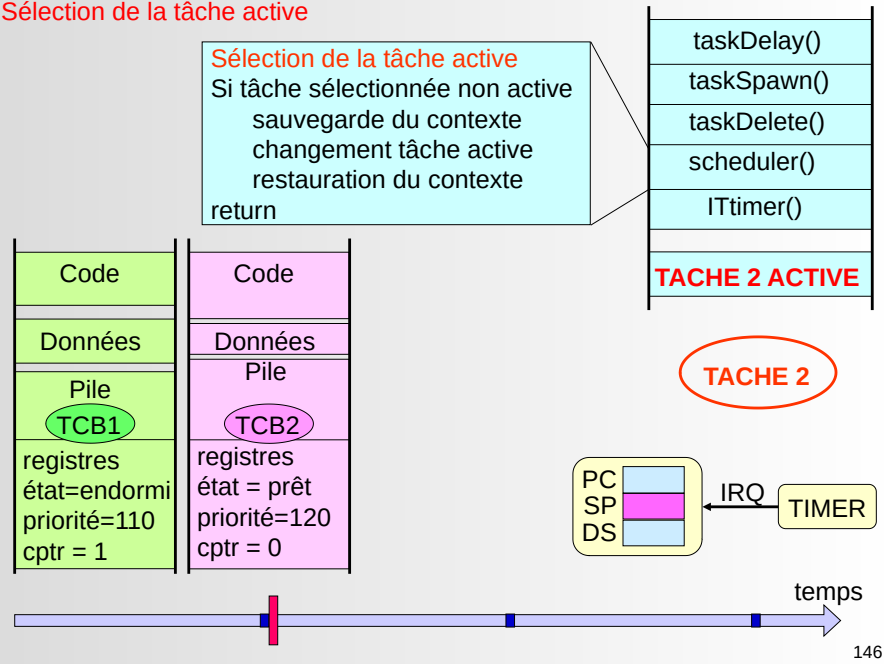
Mise à jour du compteur de sommeil



Appel au scheduler



Sélection de la tâche active



Retour au programme d'interruption

Sélection de la tâche active
Si tâche sélectionnée non active
    sauvegarde du contexte
    changement tâche active
    restauration du contexte
return

taskDelay()
taskSpawn()
taskDelete()
scheduler()
ITtimer()

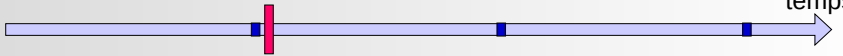
TACHE 2 ACTIVE

Code	Code
Données	Données
Pile	Pile
TCB1	TCB2
registres état=endormi priorité=110 cptr = 1	registres état = prêt priorité=120 cptr = 0

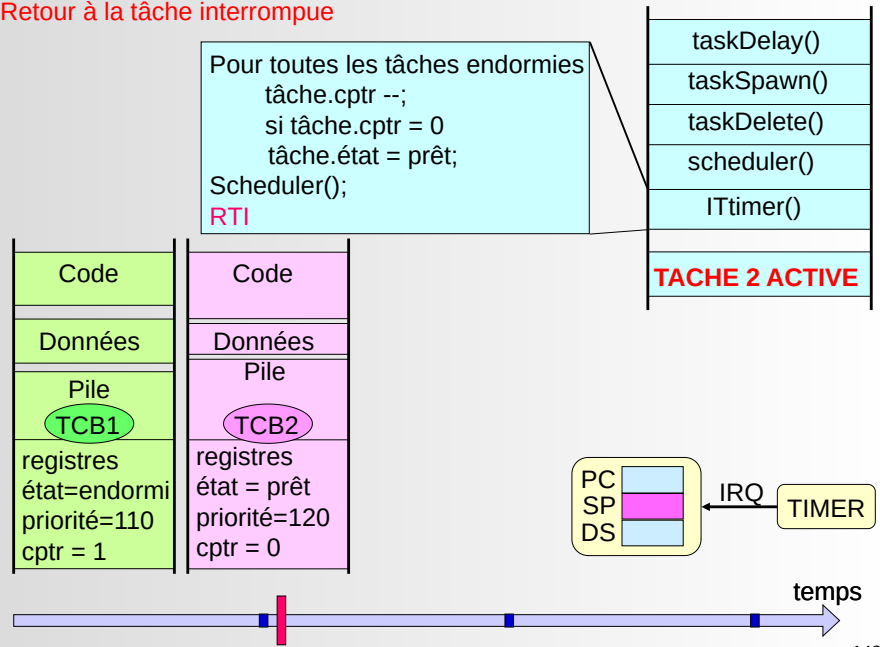
PC
SP
DS

IRQ

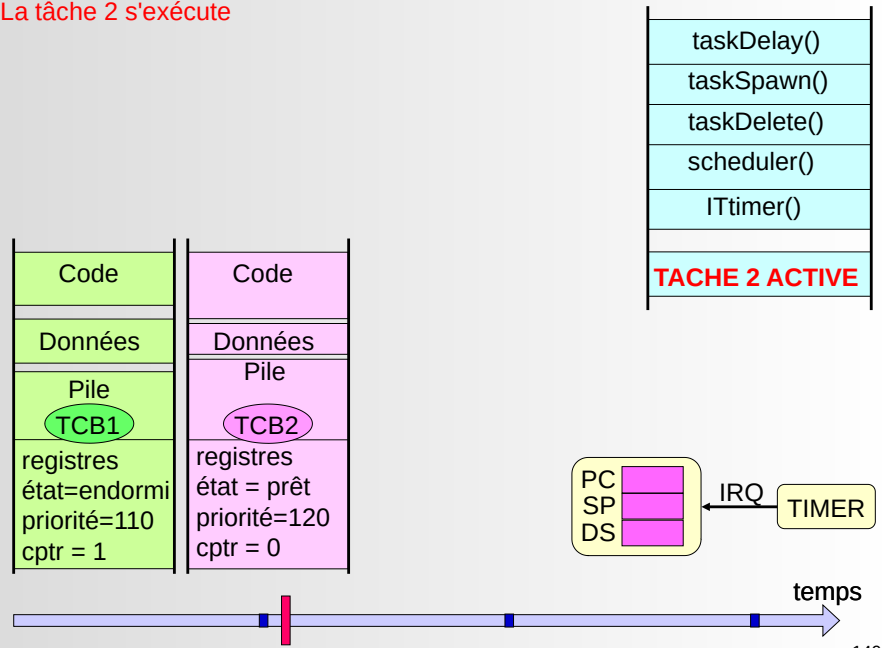
TIMER



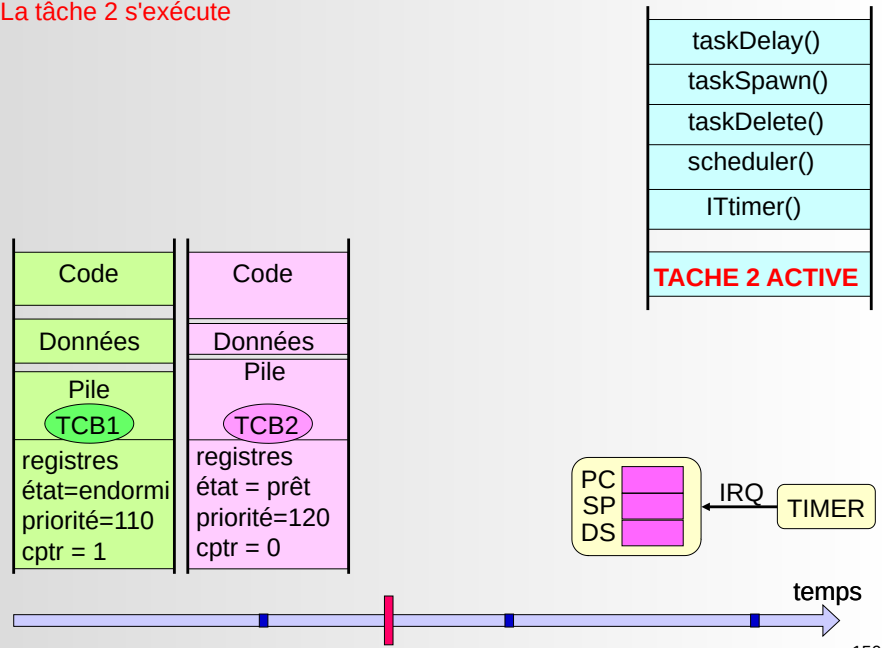
Retour à la tâche interrompue



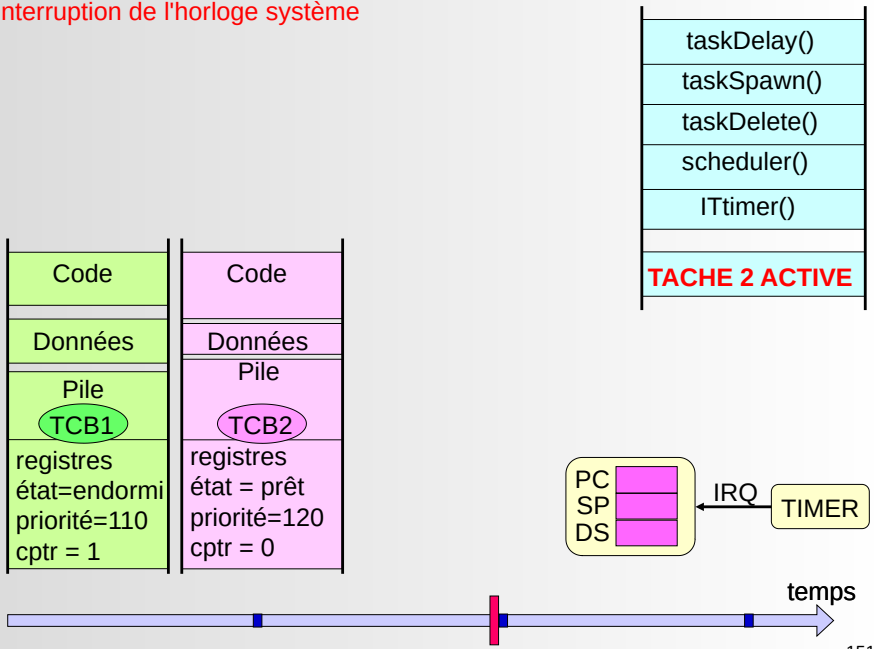
La tâche 2 s'exécute



La tâche 2 s'exécute

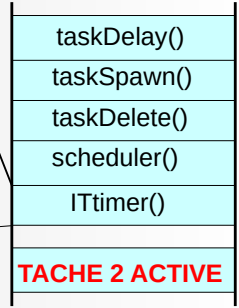


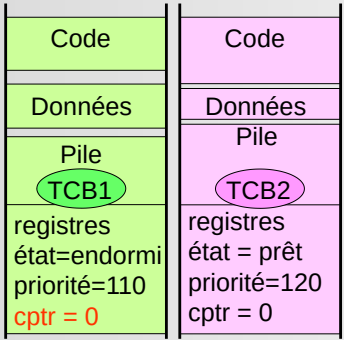
Interruption de l'horloge système

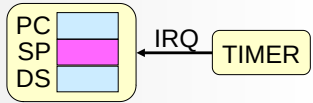


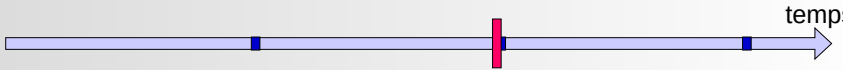
Mise à jour du compteur de sommeil

Pour toutes les tâches endormies
    tâche.cptr --;
    si tâche.cptr = 0
    tâche.état = prêt;
Scheduler();
RTI

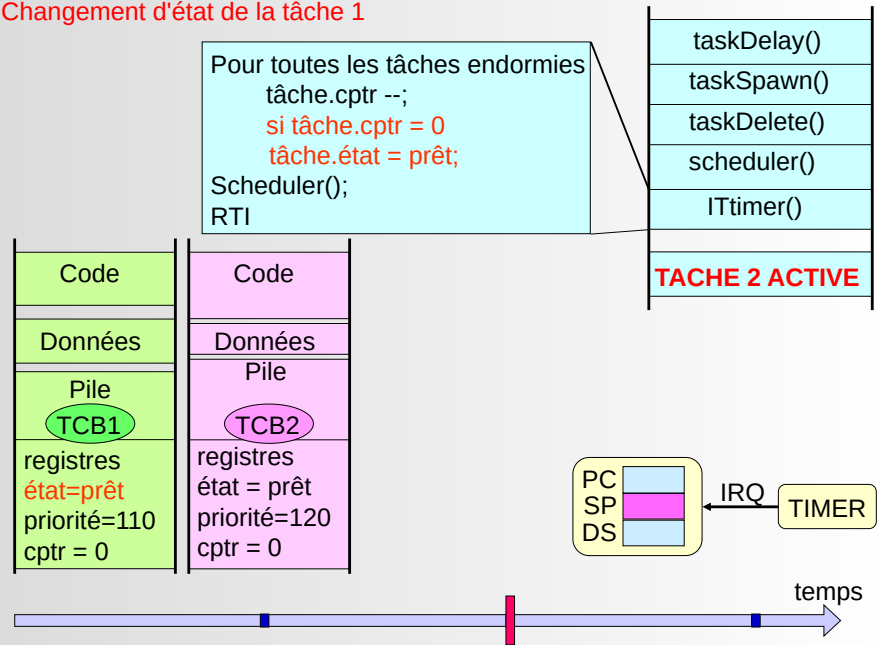




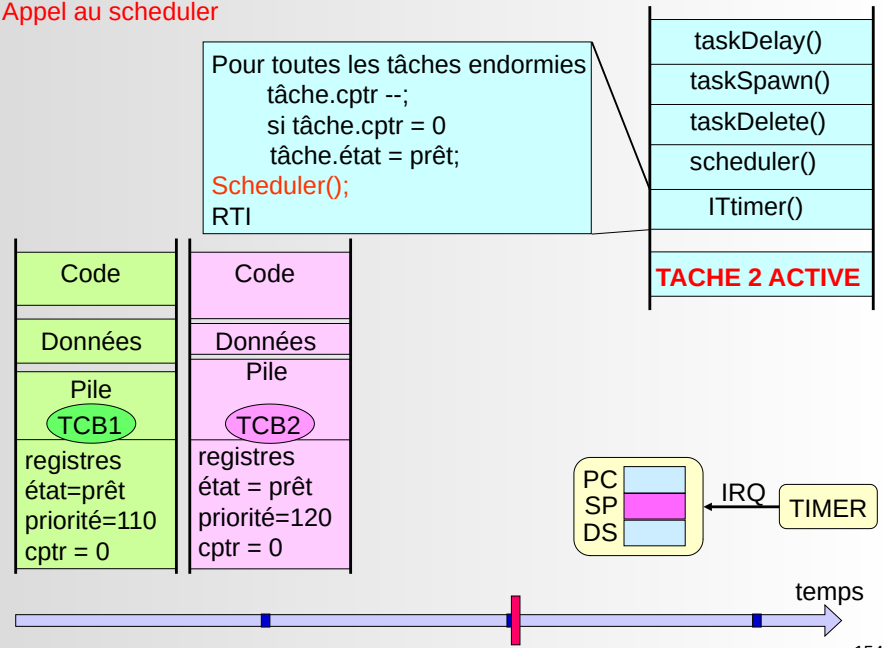




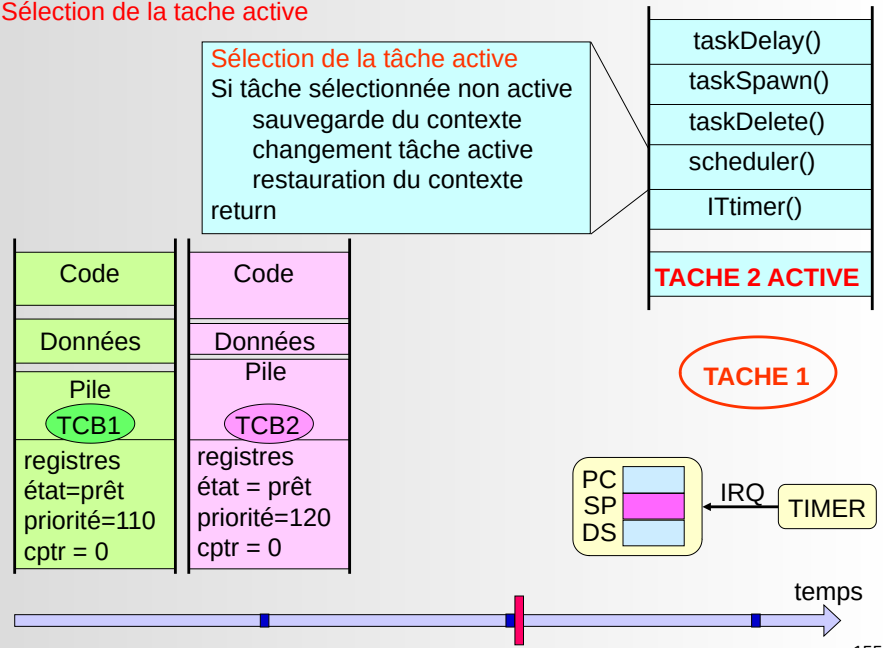
Changement d'état de la tâche 1



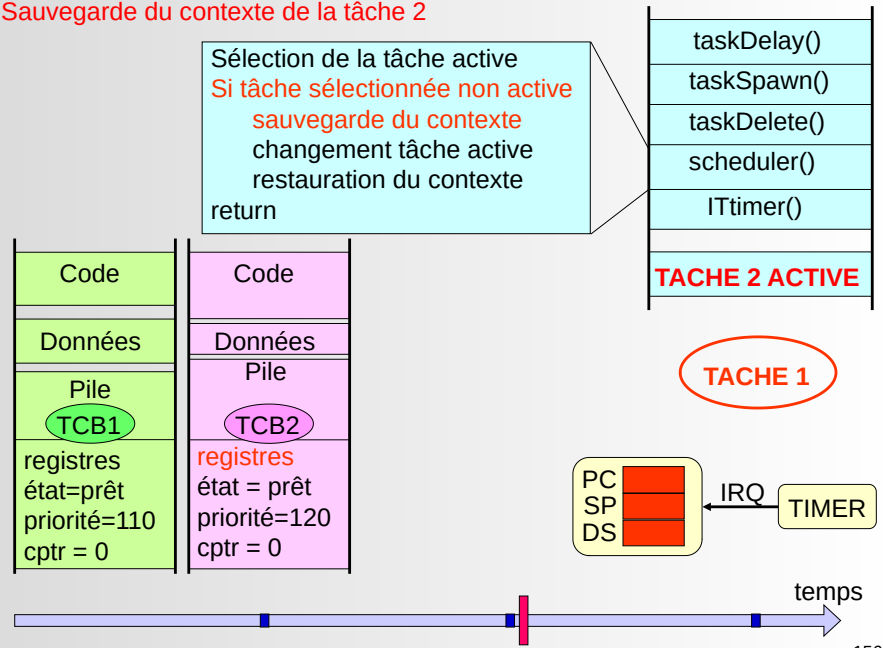
Appel au scheduler



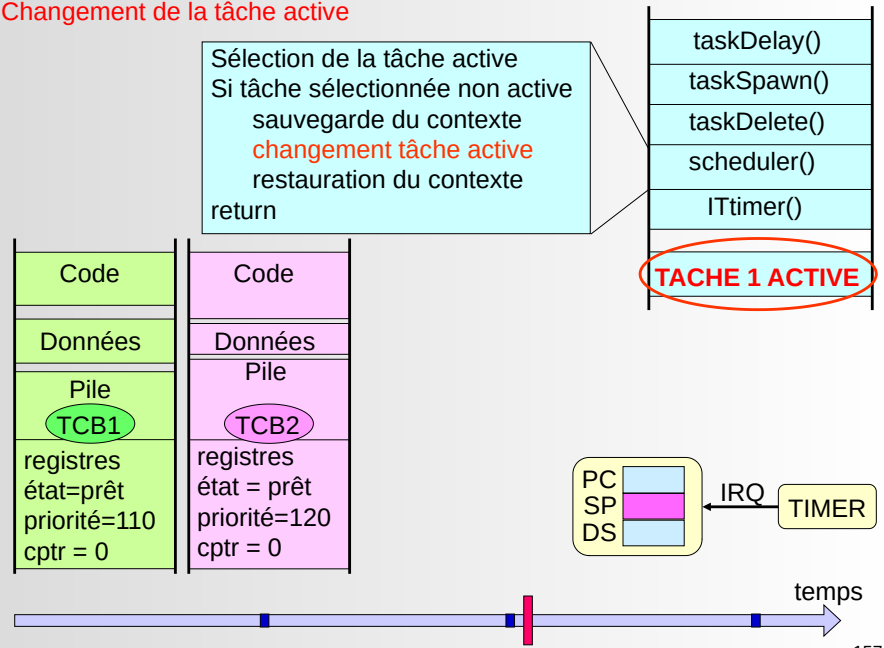
Sélection de la tache active



Sauvegarde du contexte de la tâche 2

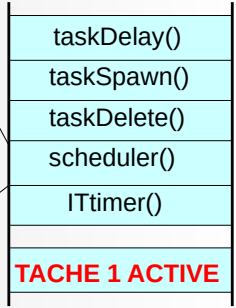


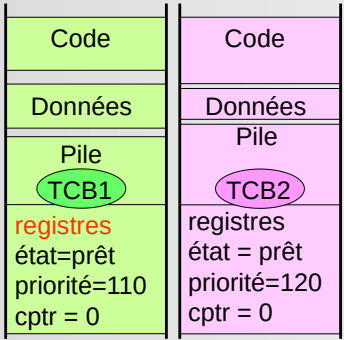
Changement de la tâche active

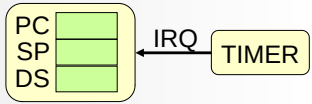


Restauration du contexte de la tâche 1

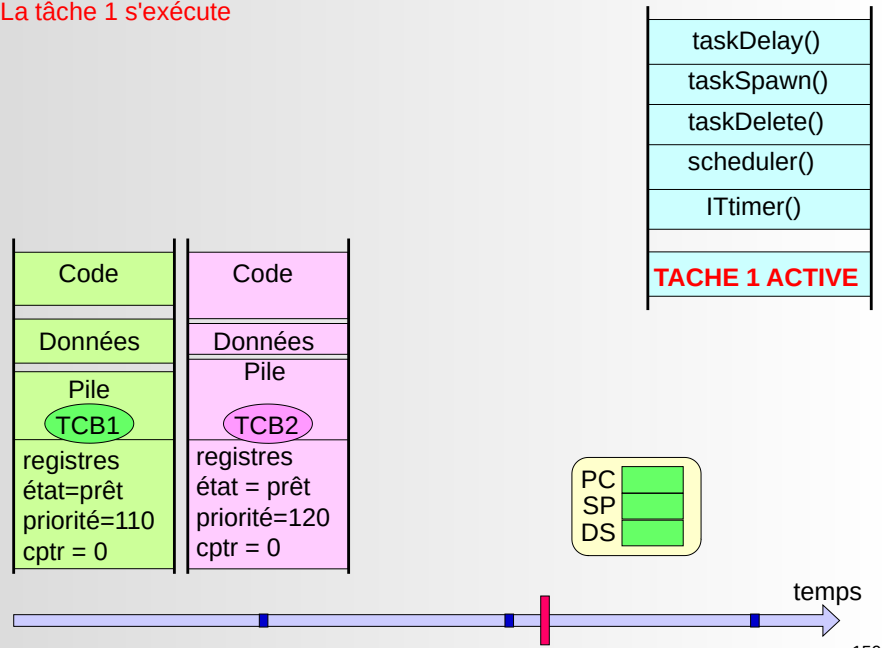
Sélection de la tâche active
Si tâche sélectionnée non active
    sauvegarde du contexte
    changement tâche active
    restauration du contexte
return



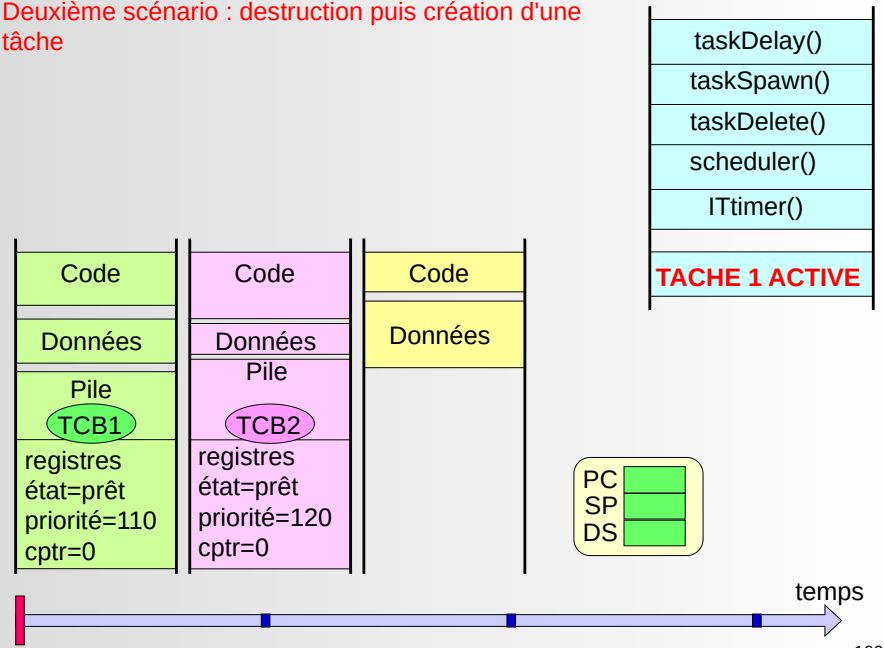




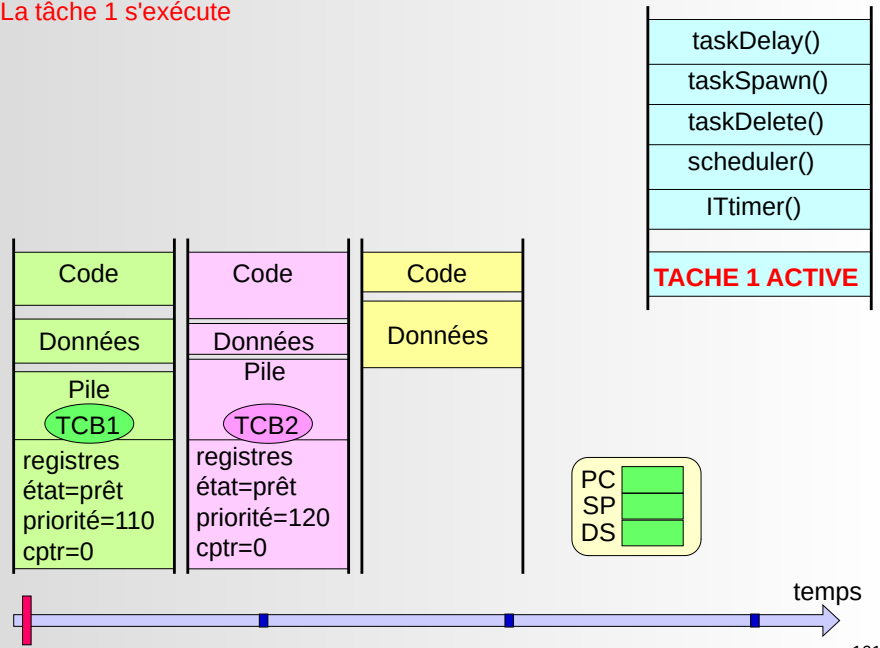
La tâche 1 s'exécute



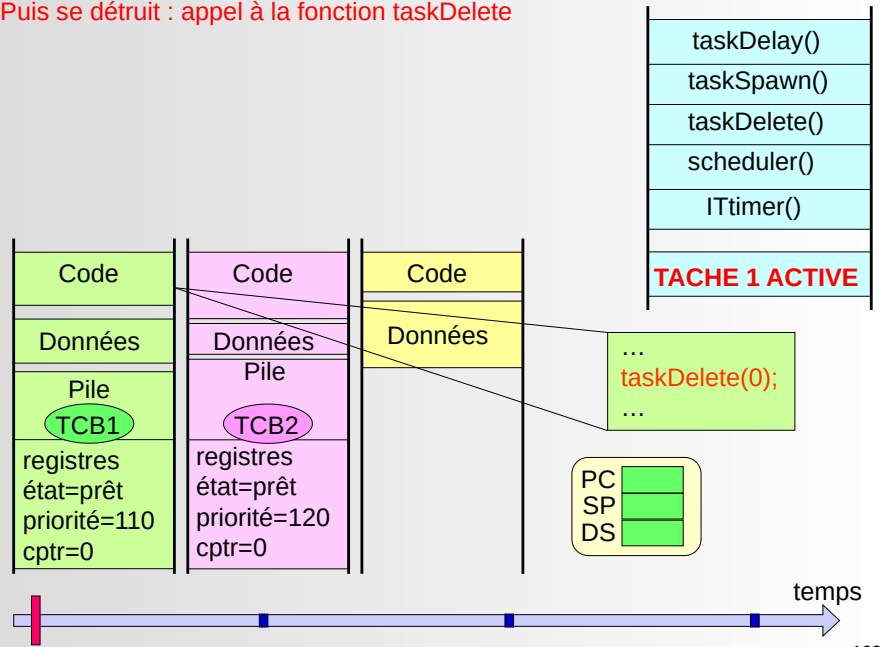
Deuxième scénario : destruction puis création d'une tâche



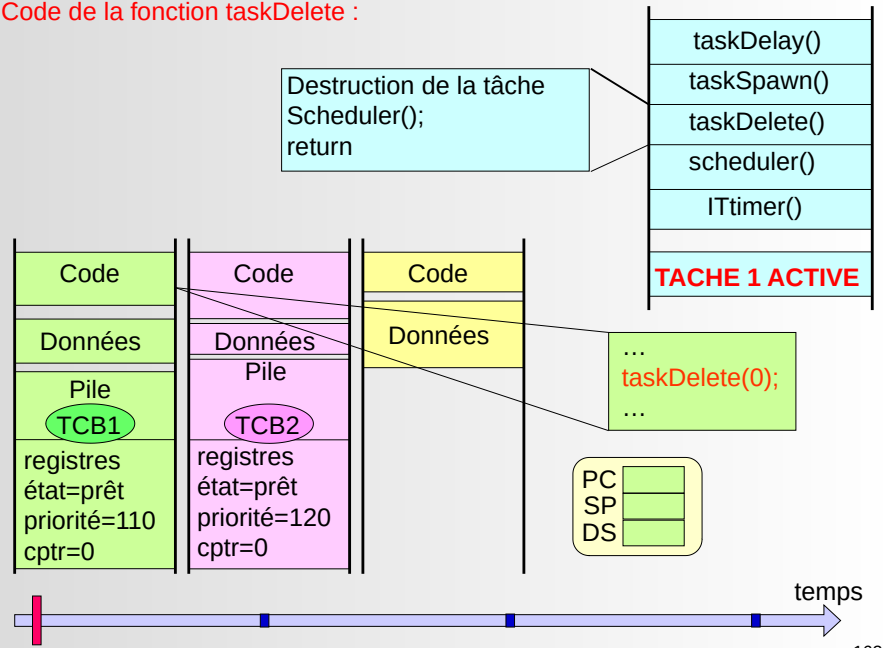
La tâche 1 s'exécute



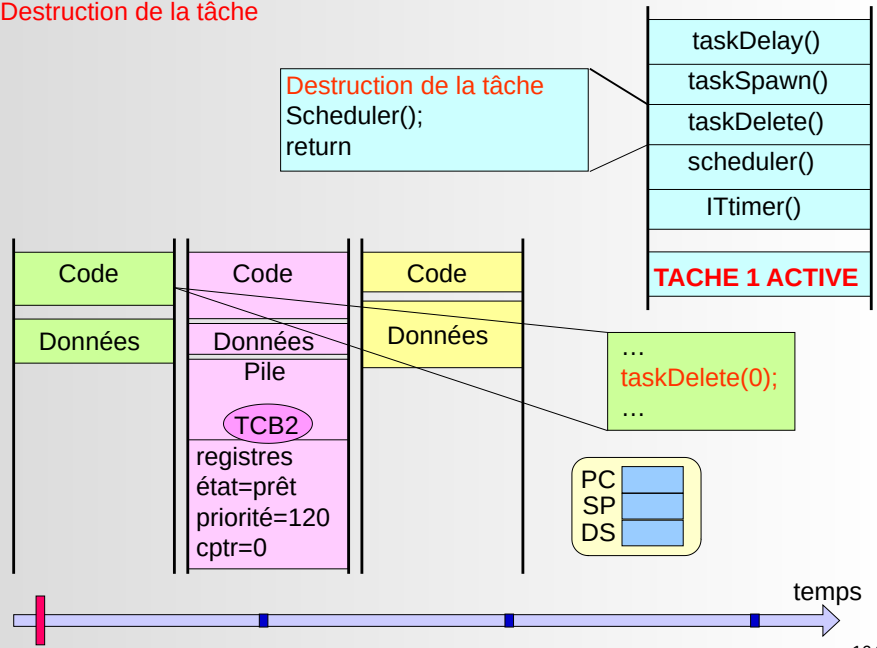
Puis se détruit : appel à la fonction taskDelete



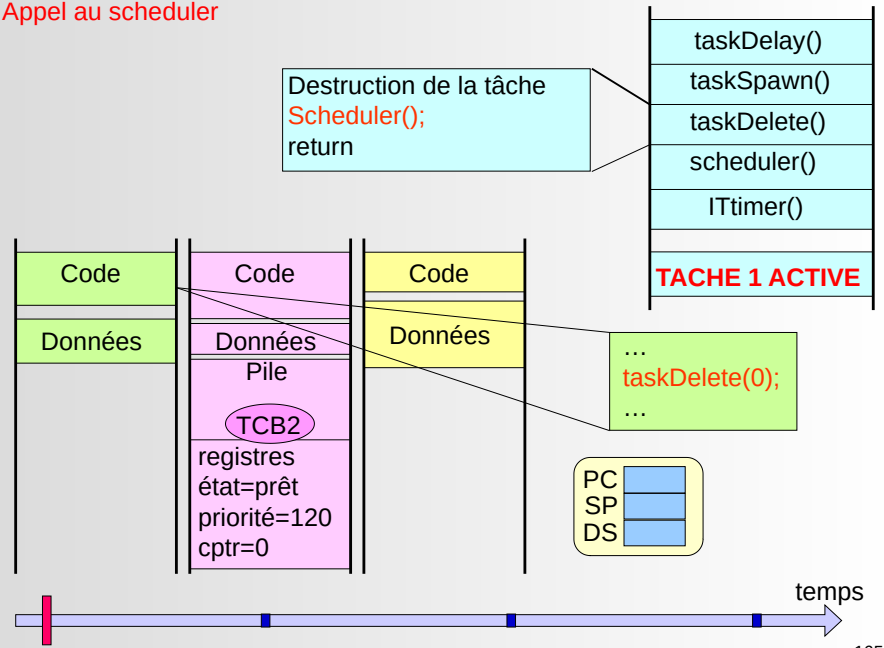
Code de la fonction taskDelete :



Destruction de la tâche



Appel au scheduler

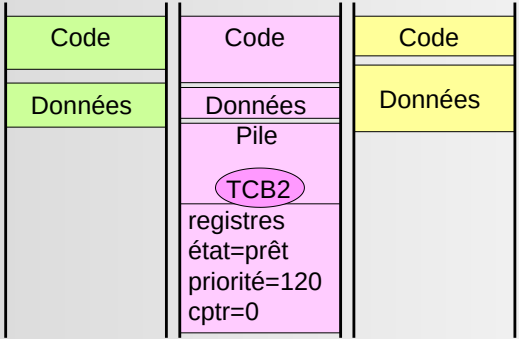


La tâche sélectionnée est la tâche 2

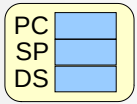
Sélection de la tâche active
Si tâche sélectionnée non active
sauvegarde du contexte
changement tâche active
restauration du contexte
return

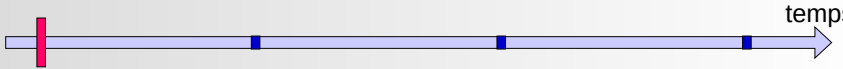
taskDelay()
taskSpawn()
taskDelete()
scheduler()
ITtimer()

TACHE 1 ACTIVE



TACHE 2



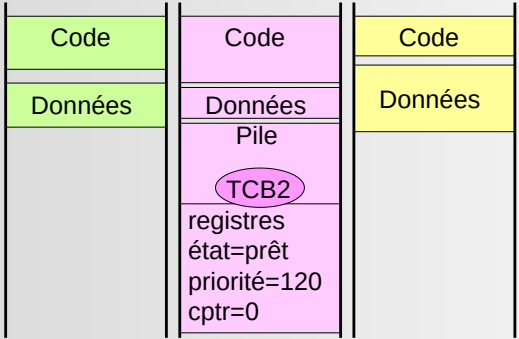


Il n'y a pas de sauvegarde de contexte à effectuer !

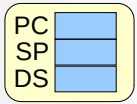
Sélection de la tâche active
Si tâche sélectionnée non active
    sauvegarde du contexte
    changement tâche active
    restauration du contexte
return

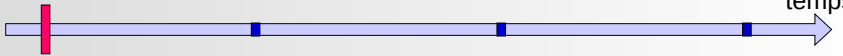
taskDelay()
taskSpawn()
taskDelete()
scheduler()
ITtimer()

TACHE 1 ACTIVE

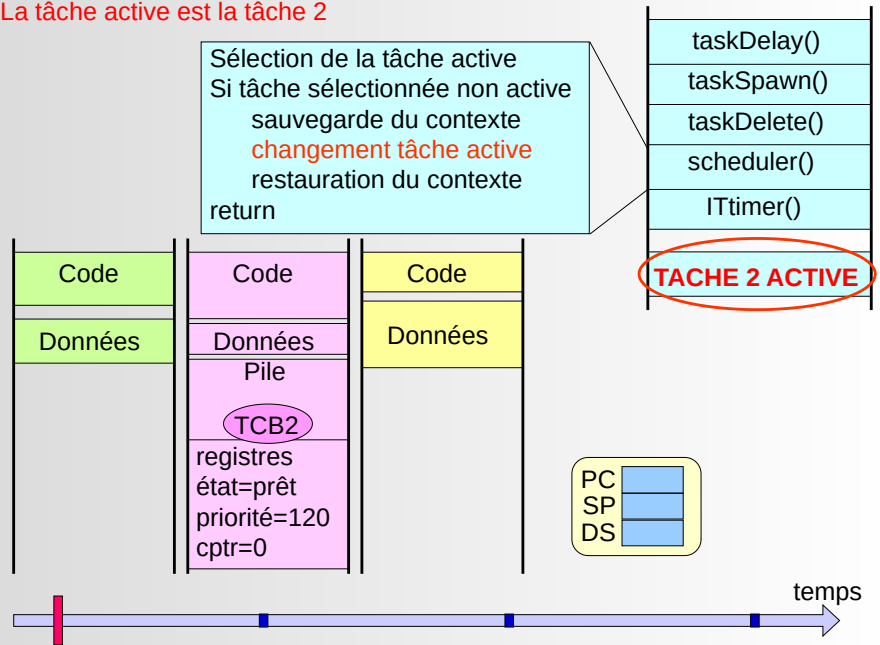


TACHE 2

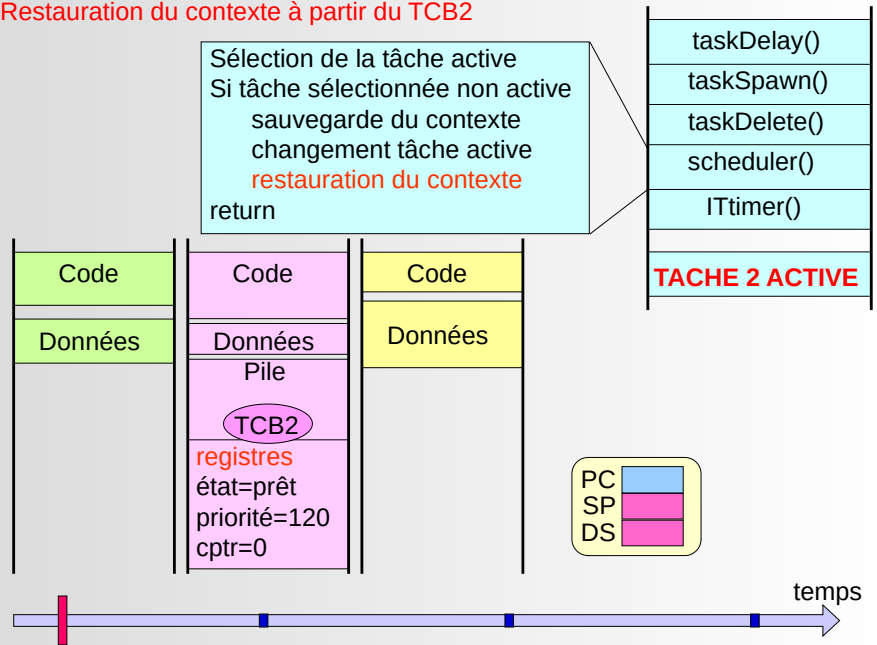




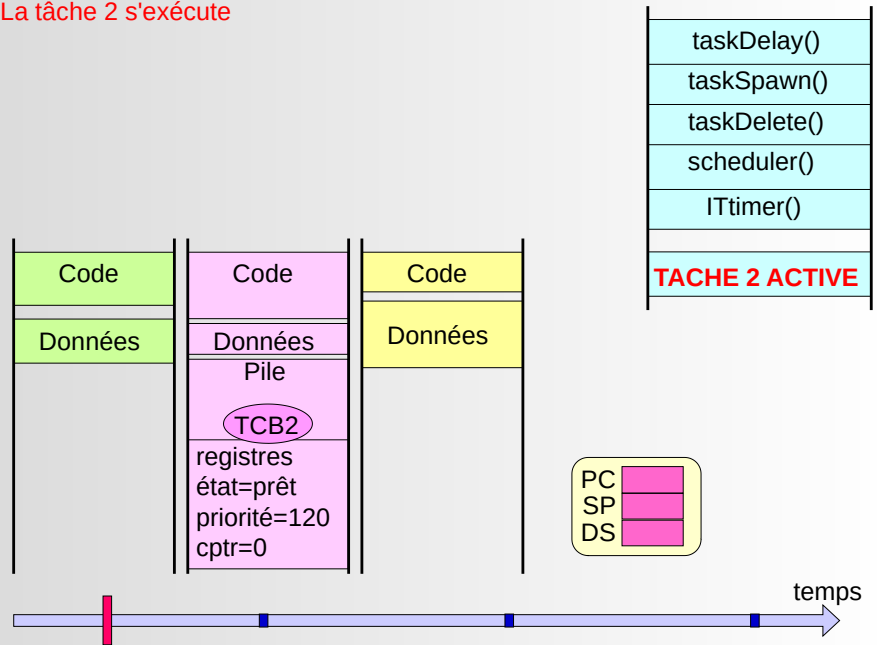
La tâche active est la tâche 2



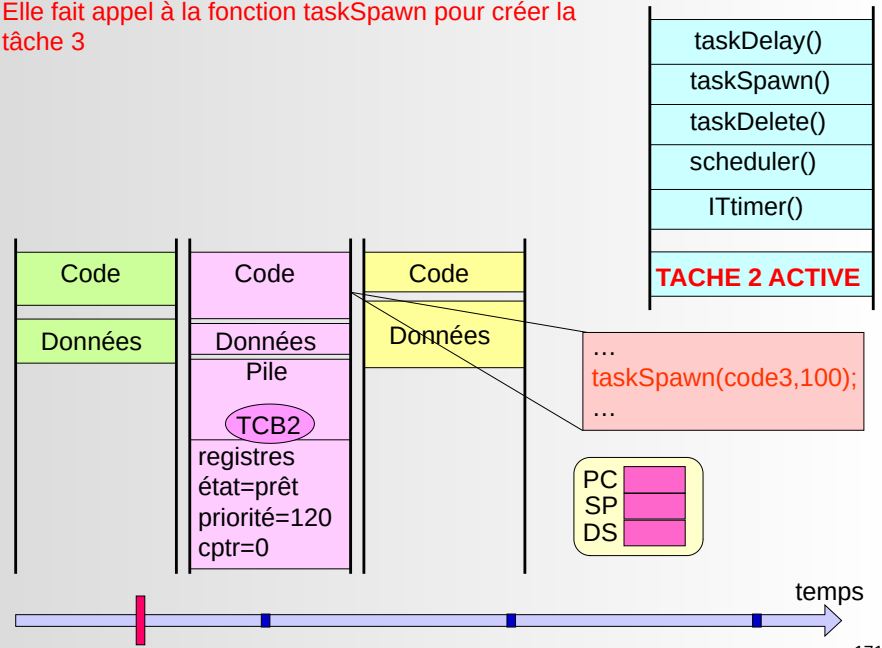
Restauration du contexte à partir du TCB2



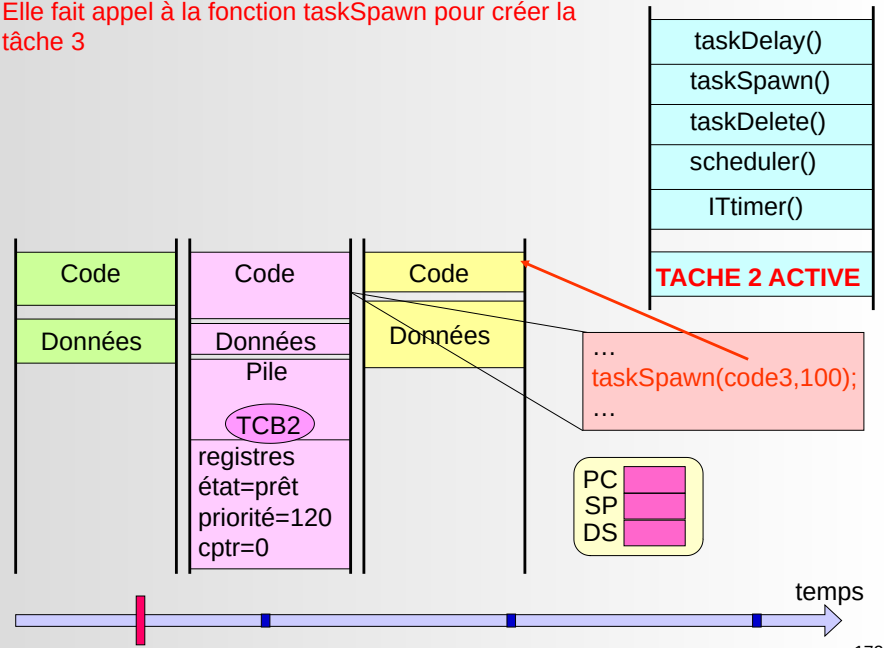
La tâche 2 s'exécute



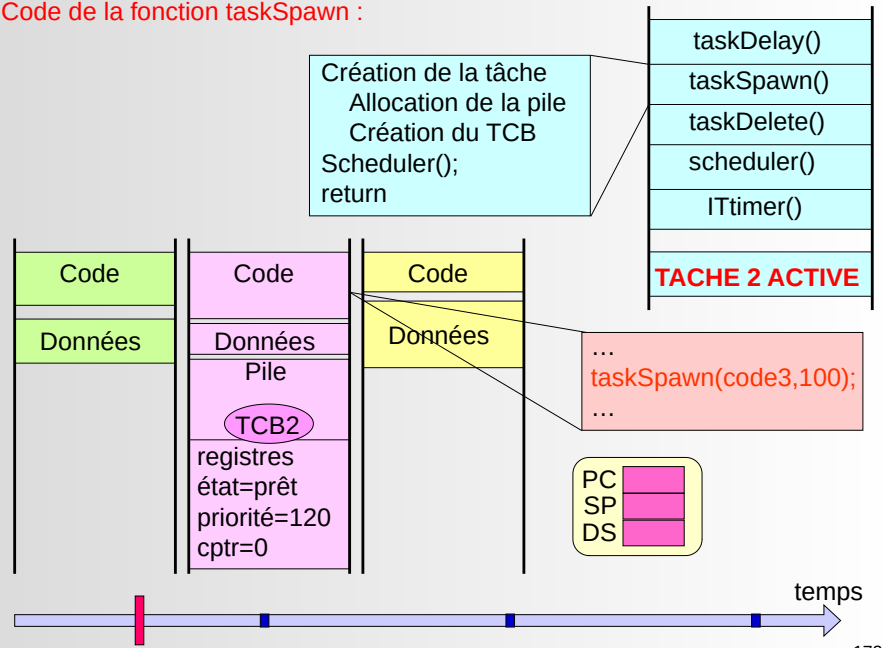
Elle fait appel à la fonction taskSpawn pour créer la tâche 3



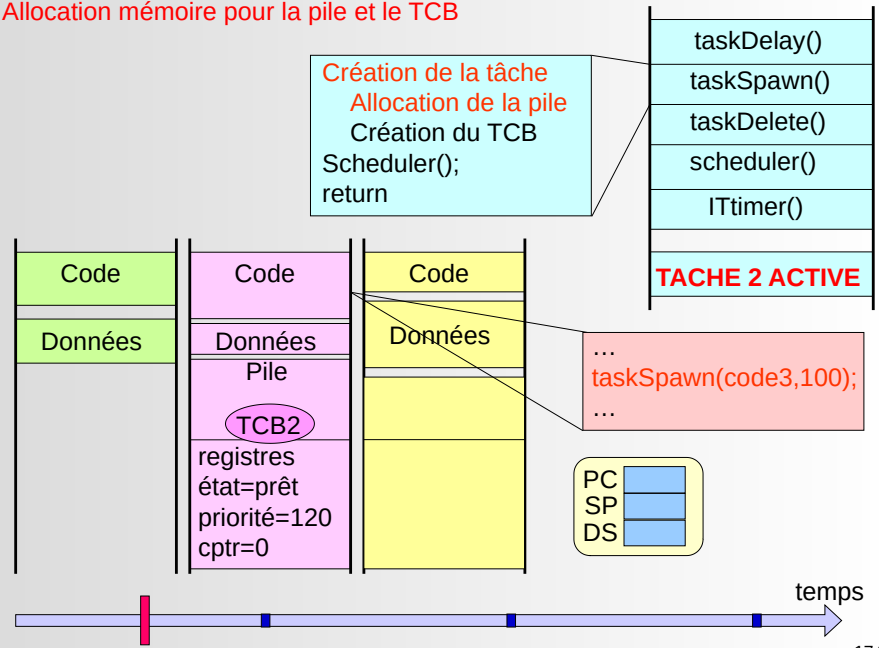
Elle fait appel à la fonction taskSpawn pour créer la tâche 3



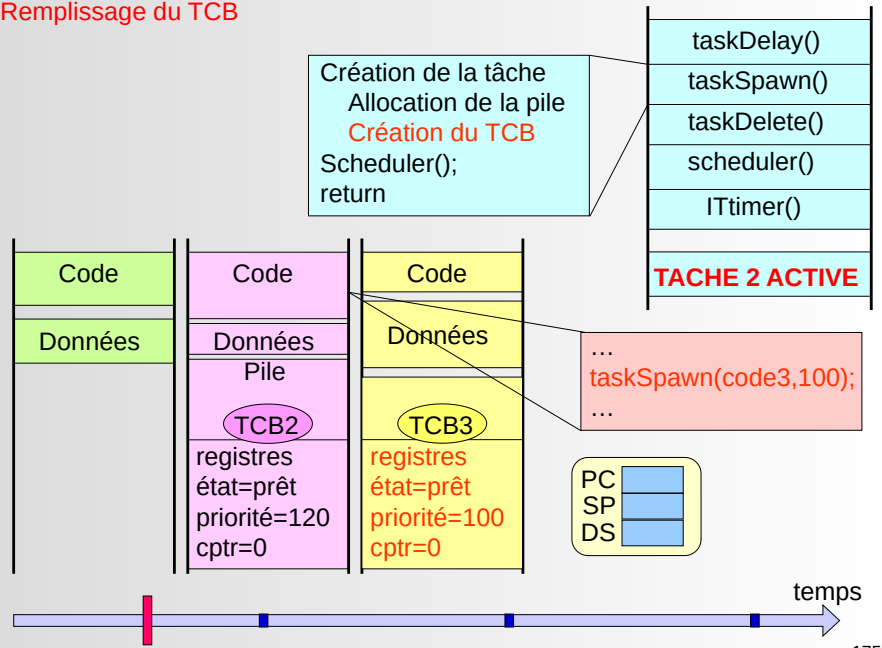
Code de la fonction taskSpawn :



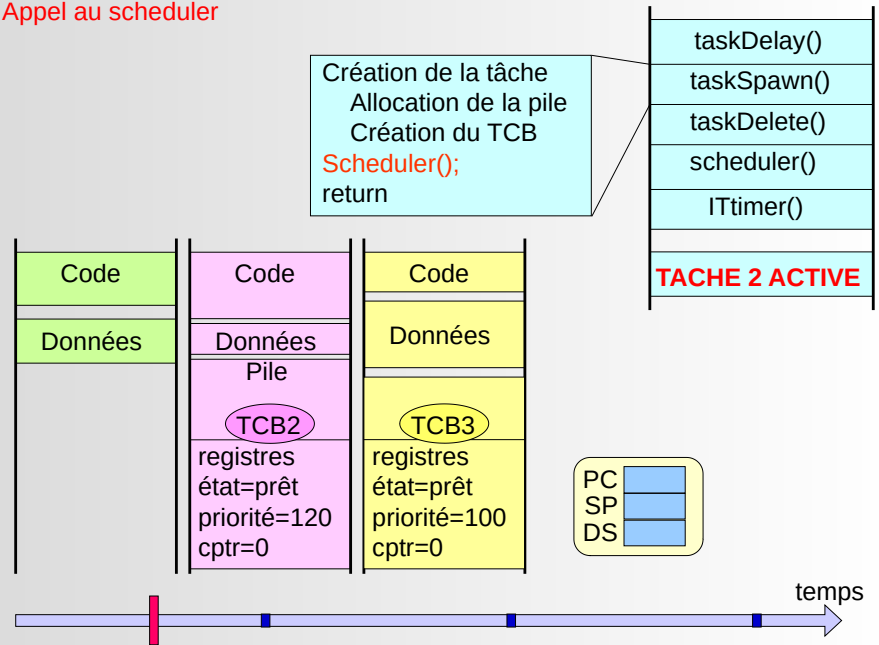
Allocation mémoire pour la pile et le TCB



Remplissage du TCB



Appel au scheduler

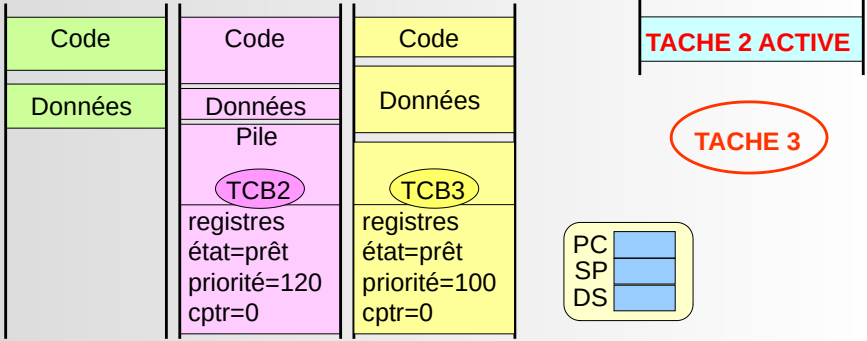


La tâche sélectionnée est la tâche 3

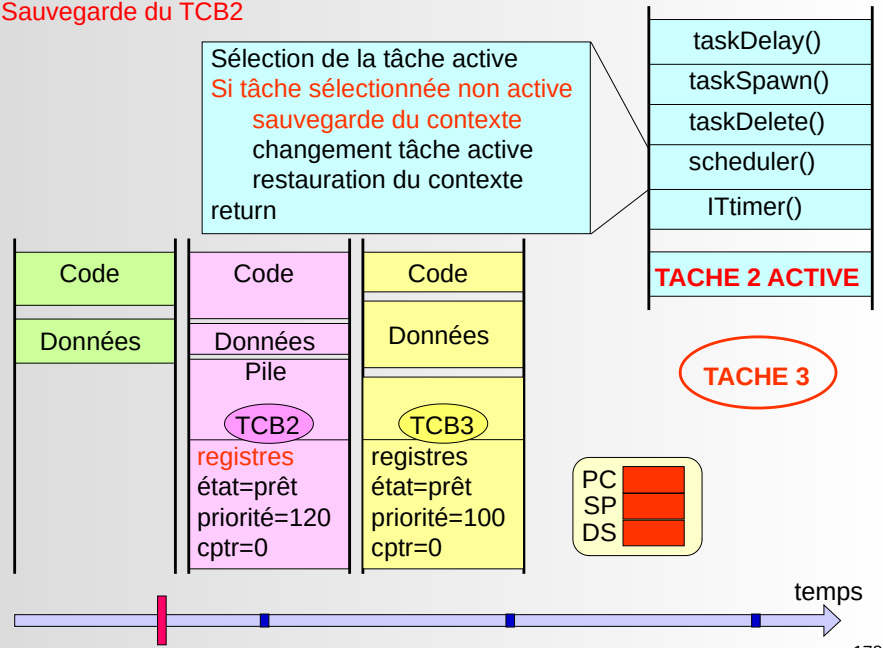
Sélection de la tâche active
Si tâche sélectionnée non active
sauvegarde du contexte
changement tâche active
restauration du contexte
return

taskDelay()
taskSpawn()
taskDelete()
scheduler()
ITtimer()

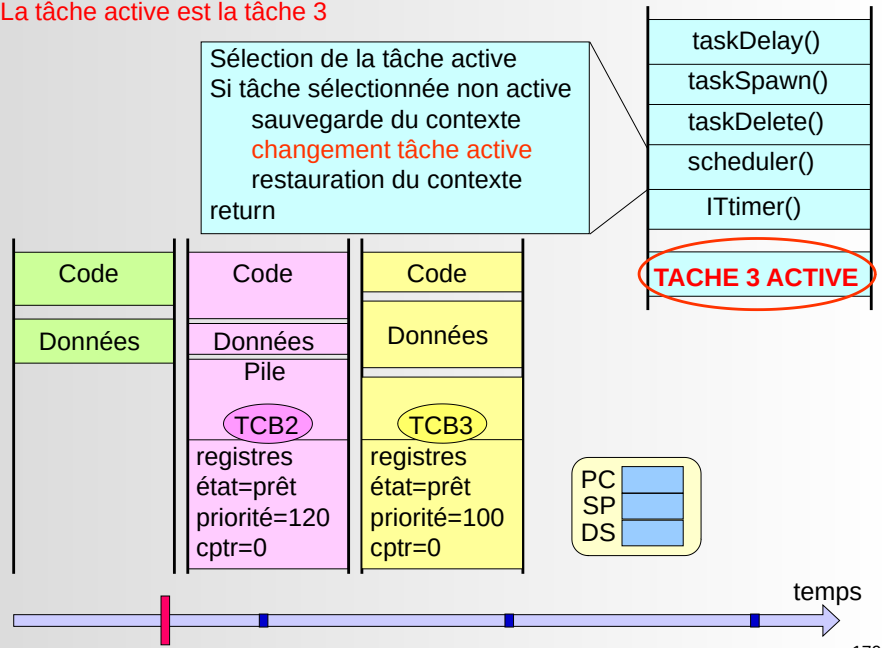
TACHE 2 ACTIVE



Sauvegarde du TCB2

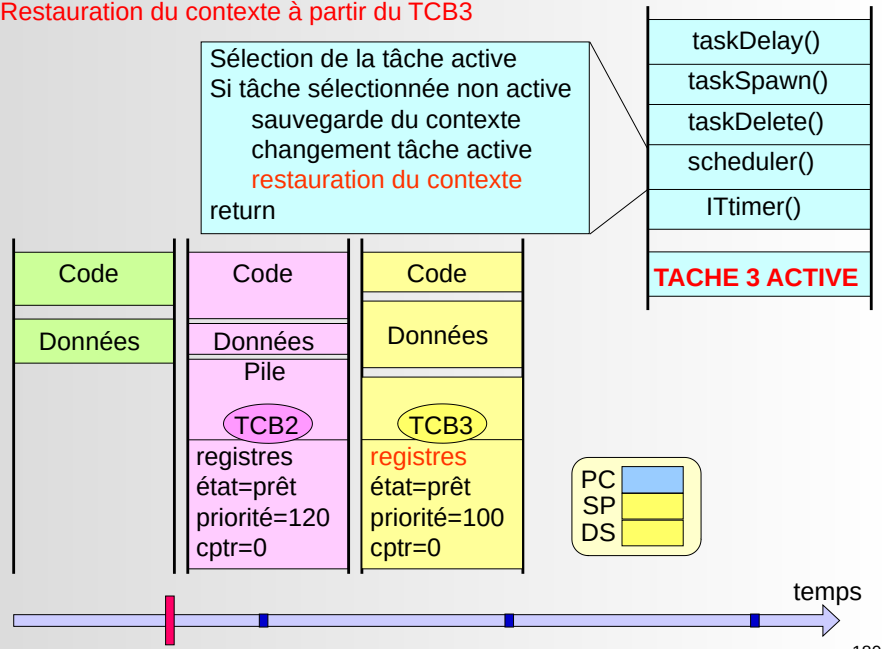


La tâche active est la tâche 3

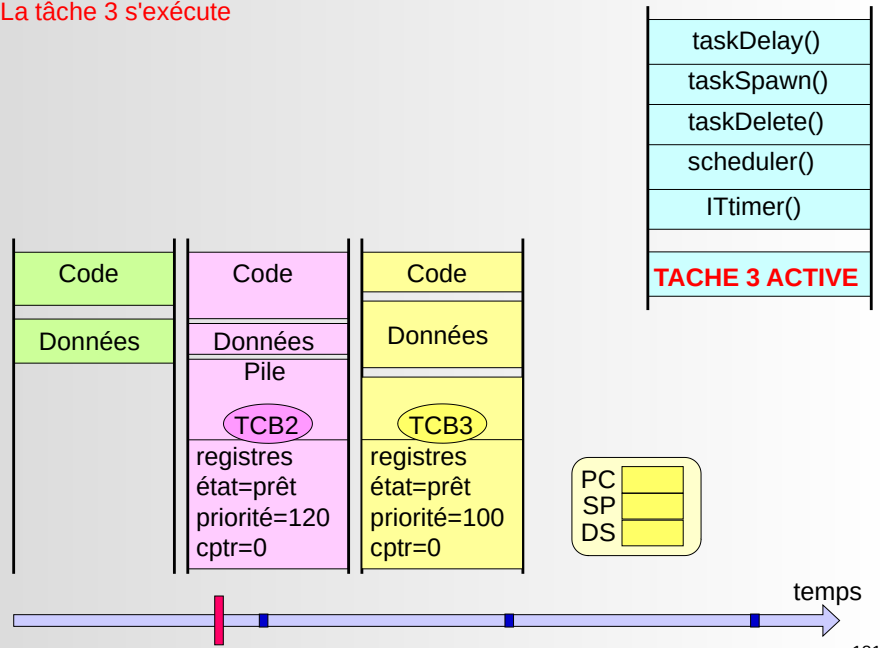


Restauration du contexte à partir du TCB3

Sélection de la tâche active
Si tâche sélectionnée non active
    sauvegarde du contexte
    changement tâche active
    restauration du contexte
return



La tâche 3 s'exécute



```c
#include <stdio.h>
#include <FreeRTOS.h>

#define STACK_SIZE 1000
#define DELAY_1 4
#define DELAY_2 4

void codeTache (void * pvParameters);

int main(void){
    BaseType_t xReturned;
    TaskHandle_t xHandle1 = NULL;
    TaskHandle_t xHandle2 = NULL;

    /* Create the task, storing the handle. */
    xReturned = xTaskCreate(
        codeTache,       /* Function that implements the task. */
        "Task1",         /* Text name for the task. */
        STACK_SIZE,      /* Stack size in words, not bytes. */
        (void *) DELAY_1,  /* Parameter passed into the task. */
        tskIDLE_PRIORITY,/* Priority at which the task is created. */
        &xHandle1 );     /* Used to pass out the created task's handle. */

    xReturned = xTaskCreate(
        codeTache,       /* Function that implements the task. */
        "Task2",         /* Text name for the task. */
        STACK_SIZE,      /* Stack size in words, not bytes. */
        (void *) DELAY_2,  /* Parameter passed into the task. */
        tskIDLE_PRIORITY,/* Priority at which the task is created. */
        &xHandle2 );     /* Used to pass out the created task's handle. */

    vTaskStartScheduler();
}
```

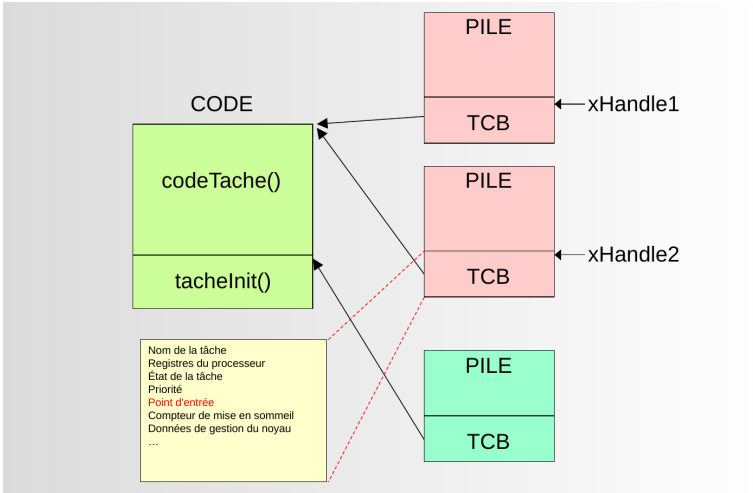
```c
void codeTache (void * pvParameters) {
    int compteur = 0;
    int duree = (int) pvParameters;
    char* s = pcTaskGetName(xTaskGetCurrentTaskHandle());

    while (1) {
        printf("Je suis la tache %s et je m'endors pour %d periodes\n", s, duree);
        vTaskDelay(duree);
        compteur++;
    }
}
```

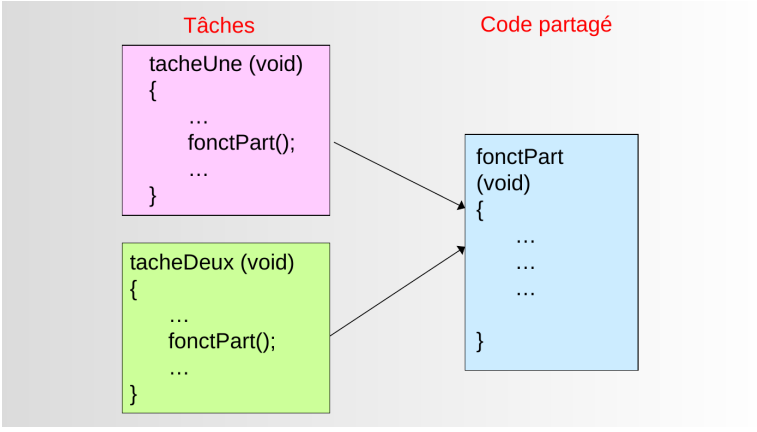
```c
/* Pour permettre l'utilisation de printf, copiez ces lignes dans le fichier main.c */
int __io_putchar(int ch) {
    HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
    return ch;
}
```

- Que faut-il ajouter pour que le printf fonctionne ?

Les segments en mémoire



Code partagé et réentrance



- Une fonction est réentrante si une seule copie du code peut être appelée par plusieurs tâches

Exemple de réentrance

Code non-réentrant

```
int tmp;

void swap(int* x, int* y) {
    tmp = *x;
    *x = *y;
    /* Hardware interrupt invoke isr() here. */
    *y = tmp;
}

void isr() {
    int x = 1, y = 2;
    swap(&x, &y);
}
```

Code réentrant

```
void swap(int* x, int* y) {
    int tmp;

    tmp = *x;
    *x = *y;
    /* Hardware interrupt invoke isr() here. */
    *y = tmp;
}

void isr() {
    int x = 1, y = 2;
    swap(&x, &y);
}
```

Code partagé et variable dynamique en pile