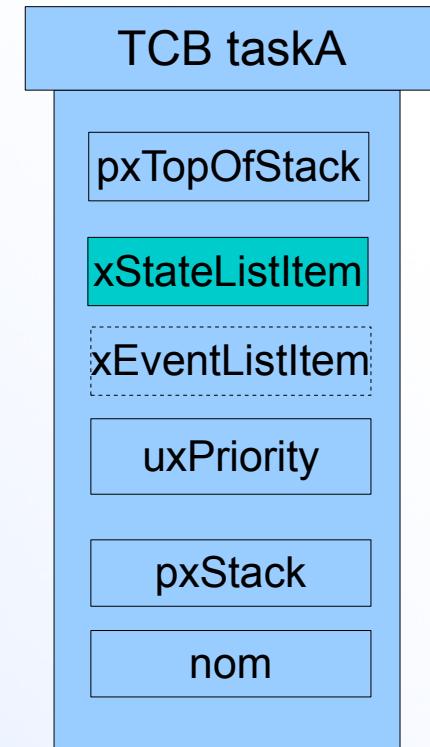
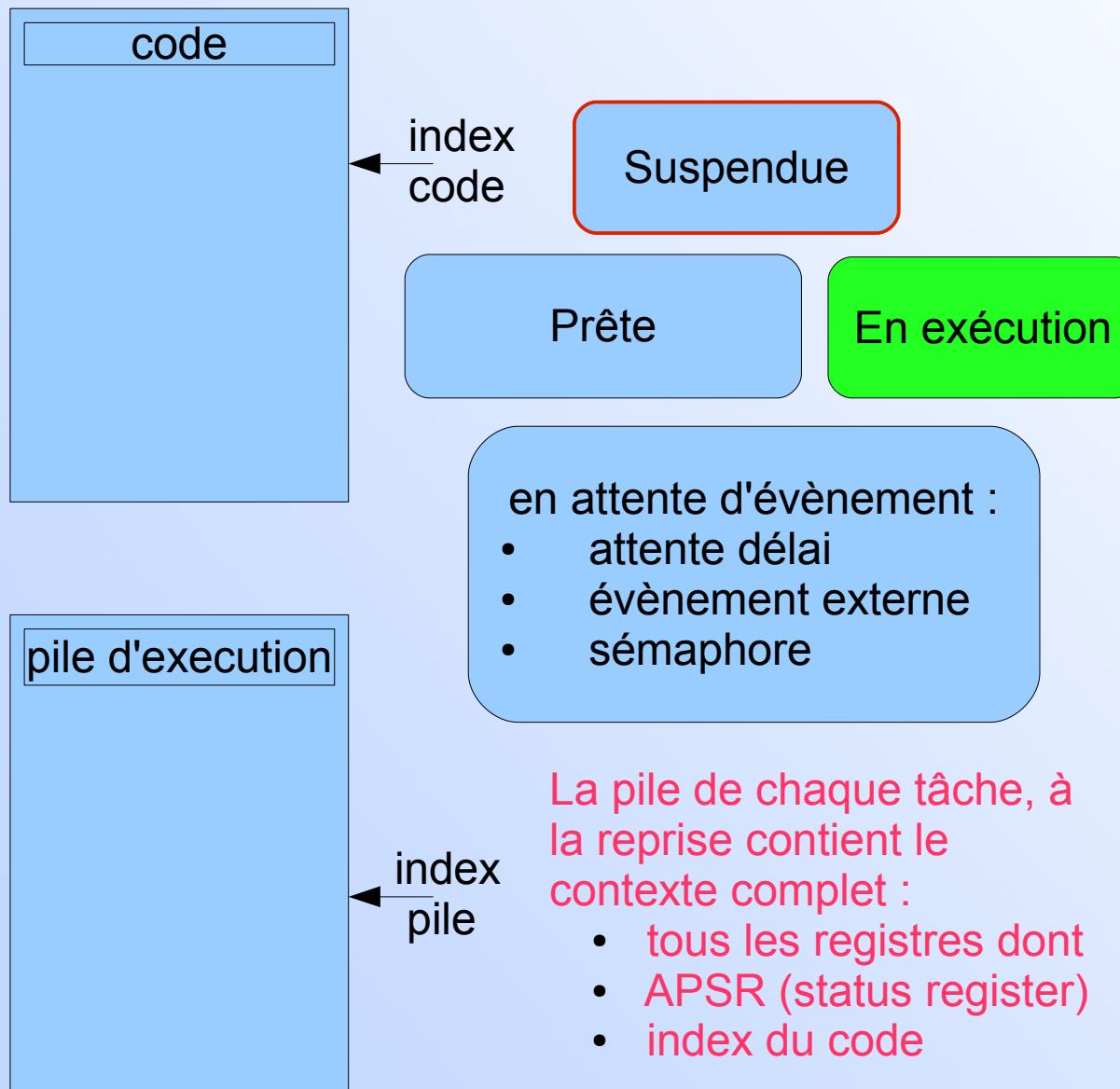


pxCurrentTCB

## Ressources



La pile de chaque tâche, à la reprise contient le contexte complet :

- tous les registres dont
- APSR (status register)
- index du code

## Gestionnaire de liste

uxNumberOfItems
pxIndex
xListEnd
xItemValue
pxNext
pxPrevious

## Connecteur

xItemValue
pxNext
pxPrevious

## Elément de Liste

xItemValue
pxNext
pxPrevious
pvOwner
pvContainer

pxReadyTasksLists[3]

pxReadyTasksLists[2]

pxReadyTasksLists[1]

pxReadyTasksLists[0]

XpendingReadyList

xDelayedTasklist

xOverflowDelayedTaskList

xSuspendedTaskList

xTasksWaitingTermination

Avoir en mémoire l'index de la pile permet la commutation de tâches

Appartenir à une liste définit l'état de la tâche

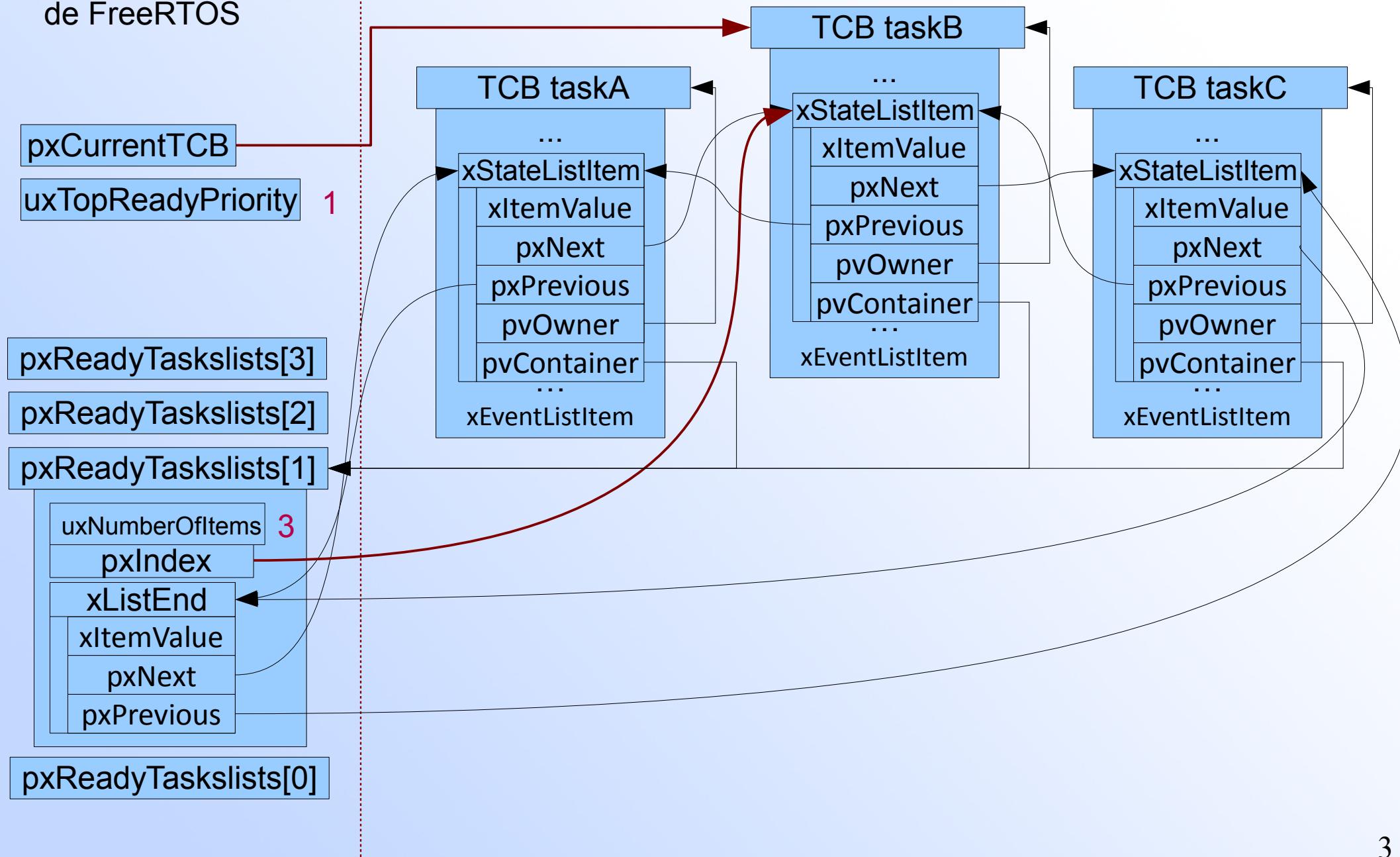
Faire partie d'une liste d'évènement gère les attentes possibles d'évènements

Mémoriser la priorité de la tâche permet de choisir la liste Ready

## TCB taskA

pxTopOfStack
xStateListItem
xEventListItem
uxPriority
pxStack
nom

## Faire partie d'une liste...

Variables d'état  
de FreeRTOS

## États d'une tâche

Variables d'état  
de FreeRTOS

pxCurrentTCB

uxTopReadyPriority

pxReadyTasksLists[3]

pxReadyTasksLists[2]

pxReadyTasksLists[1]

pxReadyTasksLists[0]

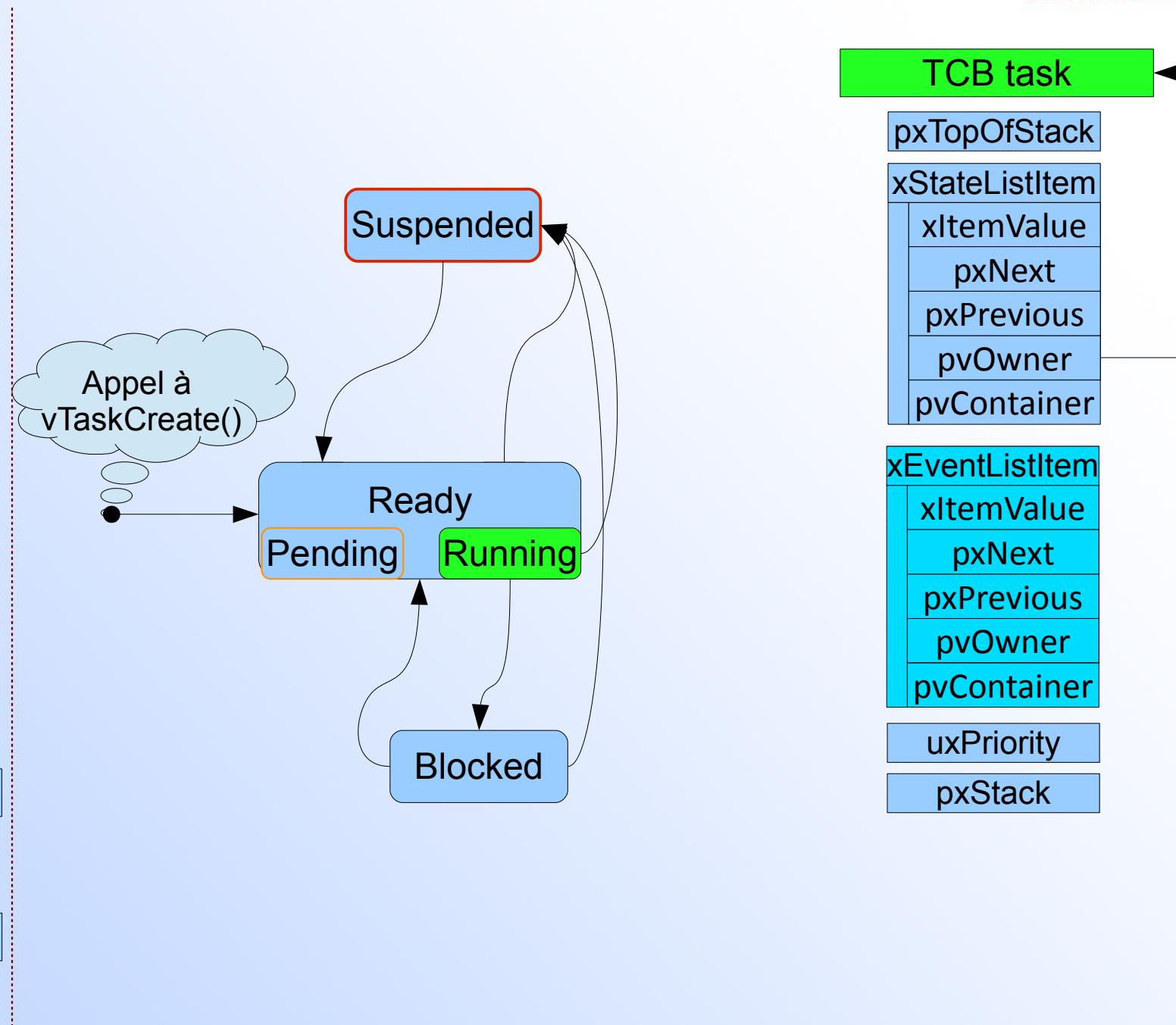
XpendingReadyList

xDelayedTasklist

xOverflowDelayedTaskList

xSuspendedTaskList

xTasksWaitingTermination



Variables d'état  
de FreeRTOS

pxCurrentTCB

uxTopReadyPriority

pxReadyTasksLists[3]

pxReadyTasksLists[2]

pxReadyTasksLists[1]

pxReadyTasksLists[0]

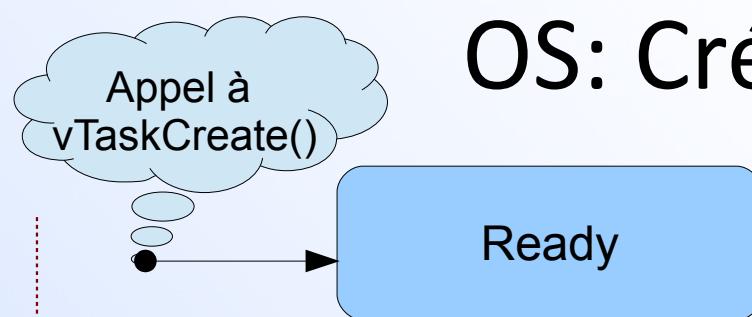
XpendingReadyList

xDelayedTasklist

xOverflowDelayedTaskList

xSuspendedTaskList

xTasksWaitingTermination



Ref manuel page 34

Prérequis :

- Disposer d'un code à exécuter :
- Avoir une liste de paramètres :
- Définir sa priorité :
- Anticiper son besoin mémoire :
  - Cascade d'appels de fonctions
  - Variables locales de ces fonctions
  - sauvegarde du contexte
- Pouvoir agir de manière externe sur la tâche :TaskHandle\_t
  - Pouvoir l'abonner à une autre liste
  - Pouvoir lui envoyer des signalisations directement

Une adresse = nom d'une fonction  
 Une adresse d'une structure par ex  
 Architecture du PROJET  
 taille de PILE

TCB taskA

pxTopOfStack

xStateListItem

xEventListItem

uxPriority

pxStack

nom

Séquence d'actions de VtaskCreate() :

- Réserver la place mémoire pour la pile
- Réserver la place mémoire pour le TCB
- Mettre à jour pxStack sur la fin de pile
- Mettre à jour pxTopOfStack sur le début de la pile
- Stockage du nom (transfert de chaîne de char)
- Mise à jour de uxPriority
- Initialisation de l'élément de liste xStateListItem
- Initialisation de l'élément de liste xEventListItem
- Initialisation de la pile pour préparer la première commutation en tenant compte des paramètres à passer à la tâche
- Inscription sur la bonne Ready Task List

# Cas d'une liste de tâches vide situation à l'initialisation

pxCurrentTCB

uxTopReadyPriority

pxReadyTaskslists[3]

pxReadyTaskslists[2]

pxReadyTaskslists[1]

uxNumberOfItems

pxIndex

xListEnd

xItemValue

pxNext

pxPrevious

pxReadyTaskslists[0]

```
void vListInitialiseItem( ListItem_t * const pxItem )  
{  
};
```

ListItem
xItemValue
pxNext
pxPrevious
pvOwner
pvContainer

```
void vListInitialise( List_t * const pxList )  
{  
};
```

# Cas d'une liste de tâches vide situation à l'initialisation

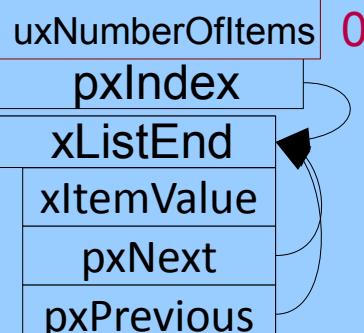
pxCurrentTCB

uxTopReadyPriority

pxReadyTaskslists[3]

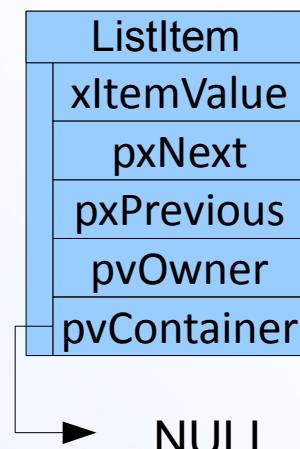
pxReadyTaskslists[2]

pxReadyTaskslists[1]



pxReadyTaskslists[0]

```
void vListInitialiseItem( ListItem_t * const pxItem )  
{  
}
```



```
void vListInitialise( List_t * const pxList )  
{  
}
```

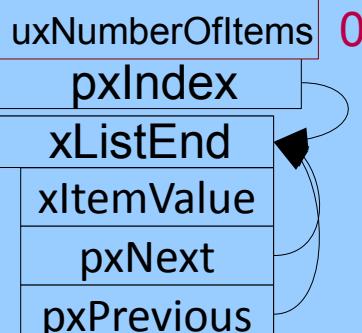
pxCurrentTCB

uxTopReadyPriority

pxReadyTaskslists[3]

pxReadyTaskslists[2]

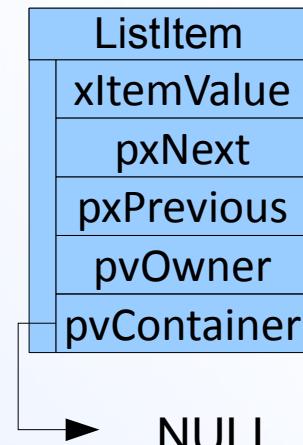
pxReadyTaskslists[1]



pxReadyTaskslists[0]

# Cas d'une liste de tâches vide situation à l'initialisation

```
void vListInitialiseItem( ListItem_t * const pxItem )  
{  
    pxItem->pvContainer = 0;  
}
```



```
void vListInitialise( List_t * const pxList )  
{  
    pxList->pxindex = ;  
    pxList->xListEnd.pxNext = ;  
    pxList->xListEnd.pxPrevious = ;  
    pxList->uxNumberOfItems = ;  
    pxList->xListEnd.xItemValue = ;  
}
```

# Cas d'une liste de tâches vide situation à l'initialisation

Variables d'état  
de FreeRTOS

pxCurrentTCB

uxTopReadyPriority

pxReadyTaskslists[3]

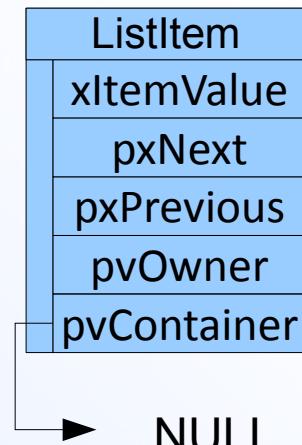
pxReadyTaskslists[2]

pxReadyTaskslists[1]

uxNumberOfItems 0  
pxIndex  
xListEnd  
xItemValue  
pxNext  
pxPrevious

pxReadyTaskslists[0]

```
void vListInitialiseItem( ListItem_t * const pxItem )  
{  
    pxItem->pvContainer = 0;  
}
```



```
void vListInitialise( List_t * const pxList )  
{  
    pxList->pxindex = &(pxList->xListEnd) ;  
    pxList->xListEnd.pxNext = &(pxList->xListEnd) ;  
    pxList->xListEnd.pxPrevious = &(pxList->xListEnd);  
    pxList->uxNumberOfItems = ( UBaseType_t ) 0U ;  
    pxList->xListEnd.xItemValue = ( TickType_t ) 0xffffffffUL ;  
}
```

Variables d'état  
de FreeRTOS

pxCurrentTCB

uxTopReadyPriority

pxReadyTasksLists[3]

pxReadyTasksLists[2]

pxReadyTasksLists[1]

pxReadyTasksLists[0]

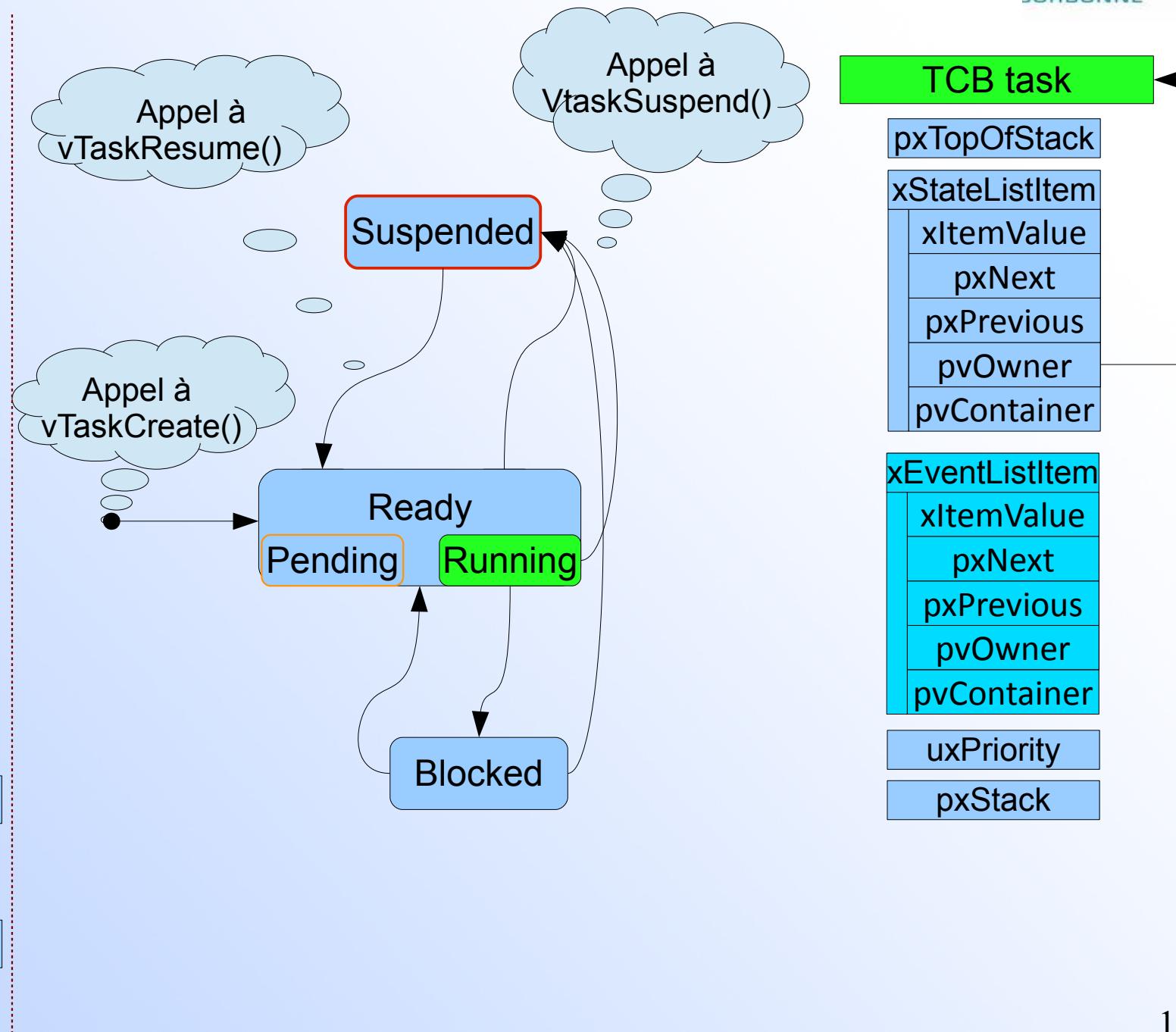
XpendingReadyList

xDelayedTasklist

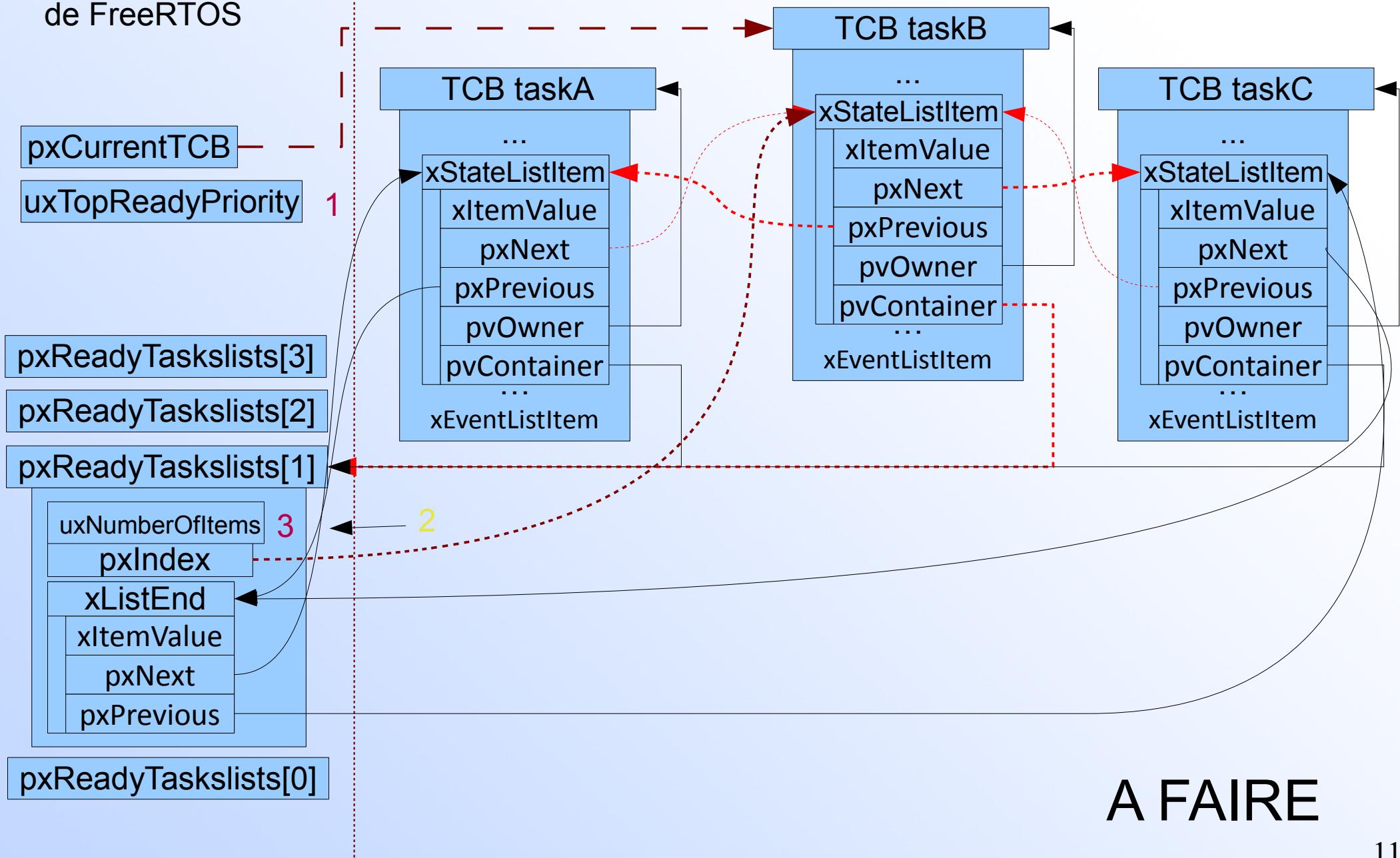
xOverflowDelayedTaskList

xSuspendedTaskList

xTasksWaitingTermination

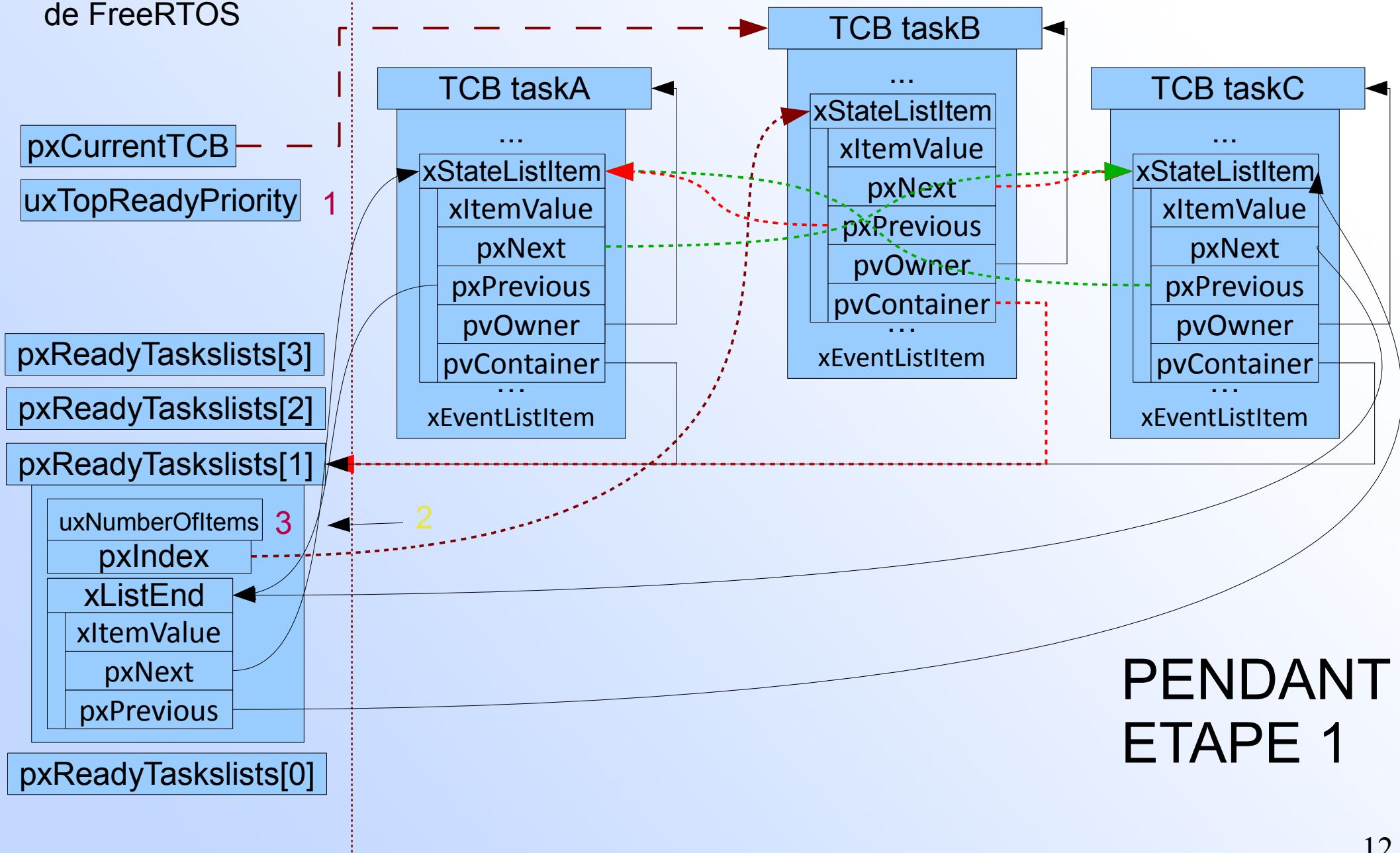


## Supprimer une tâche

Variables d'état  
de FreeRTOS

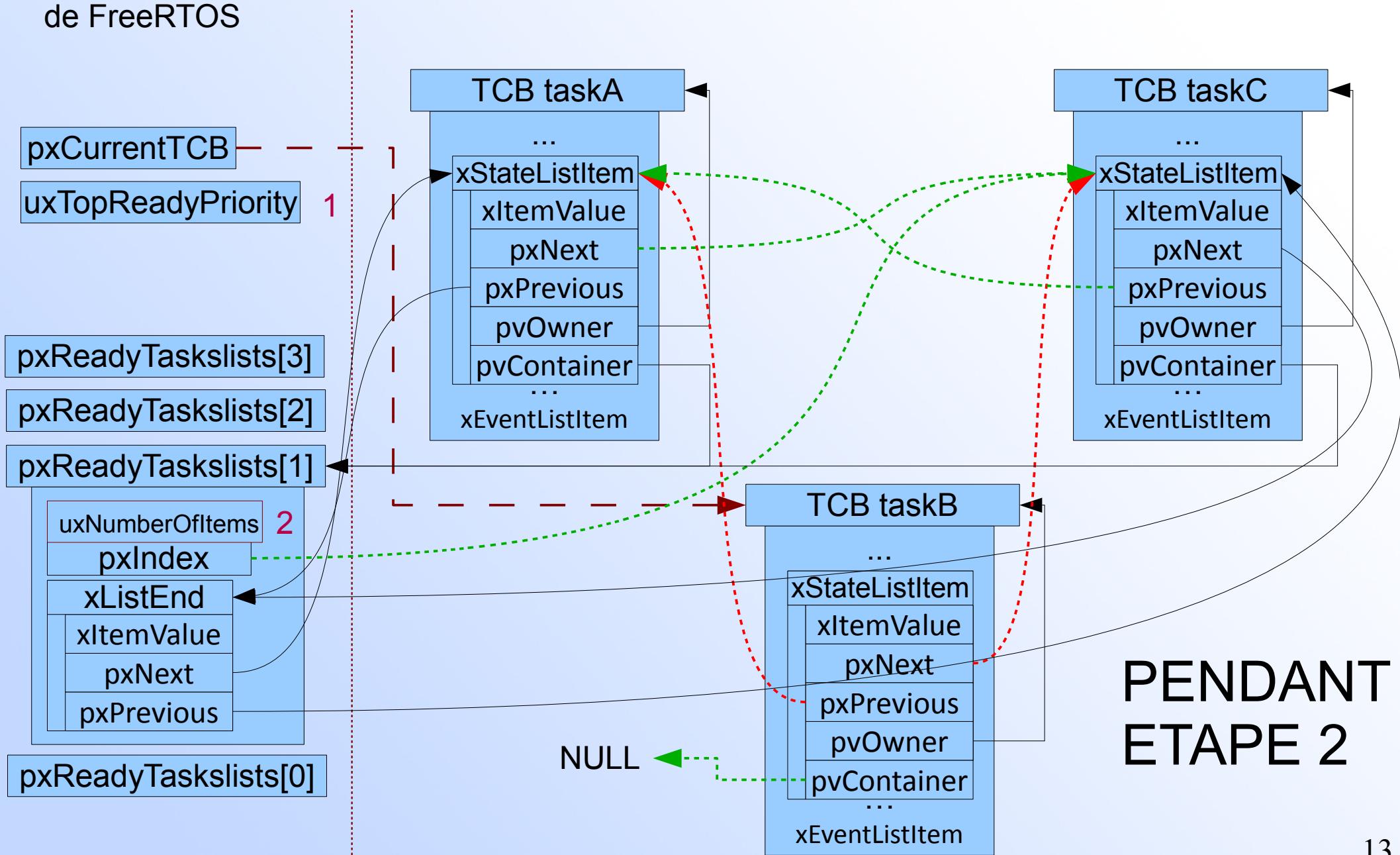
# Supprimer une tâche d'une liste de tâches chaînées

Variables d'état  
de FreeRTOS



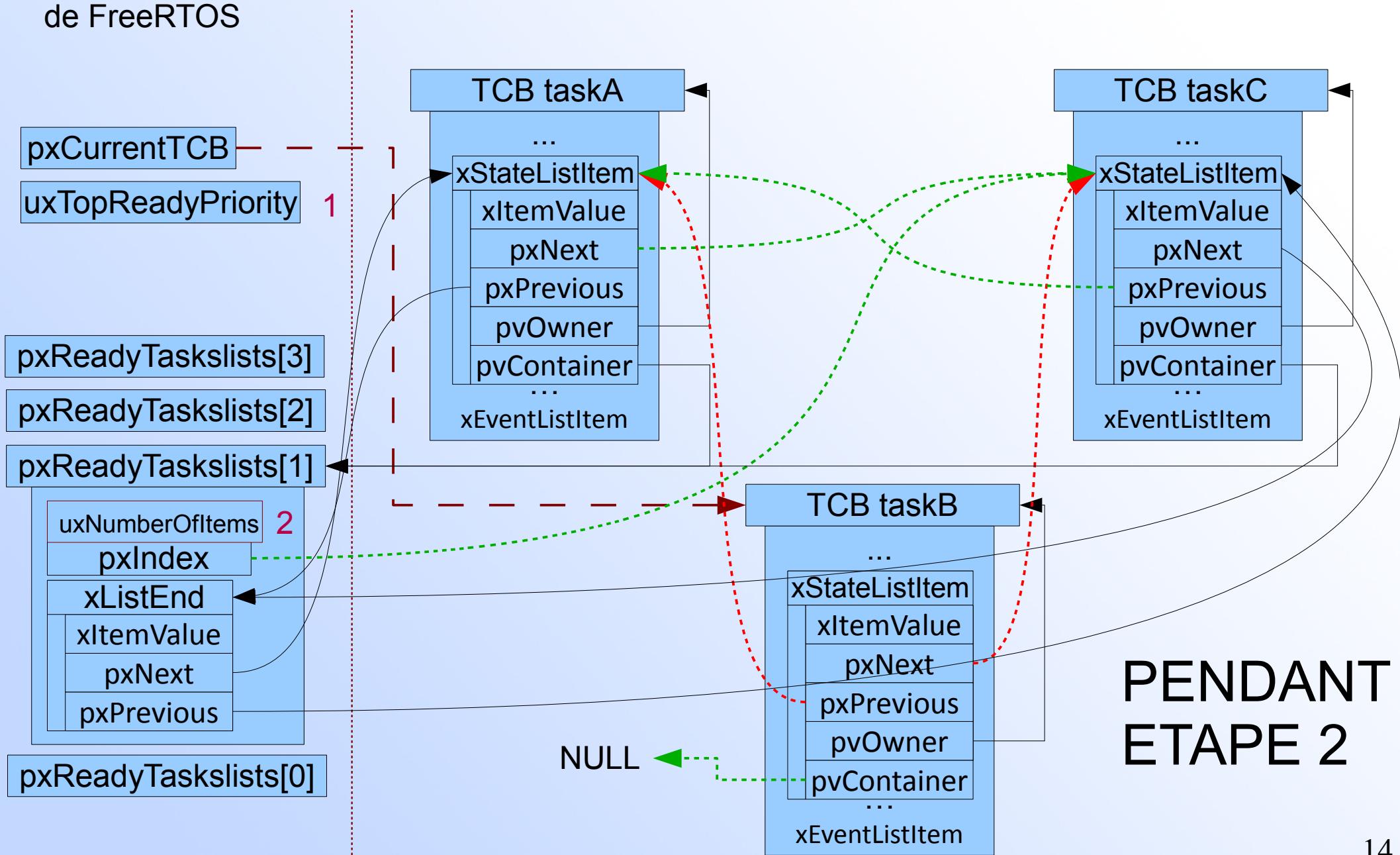
# Supprimer une tâche d'une liste de tâches chaînées

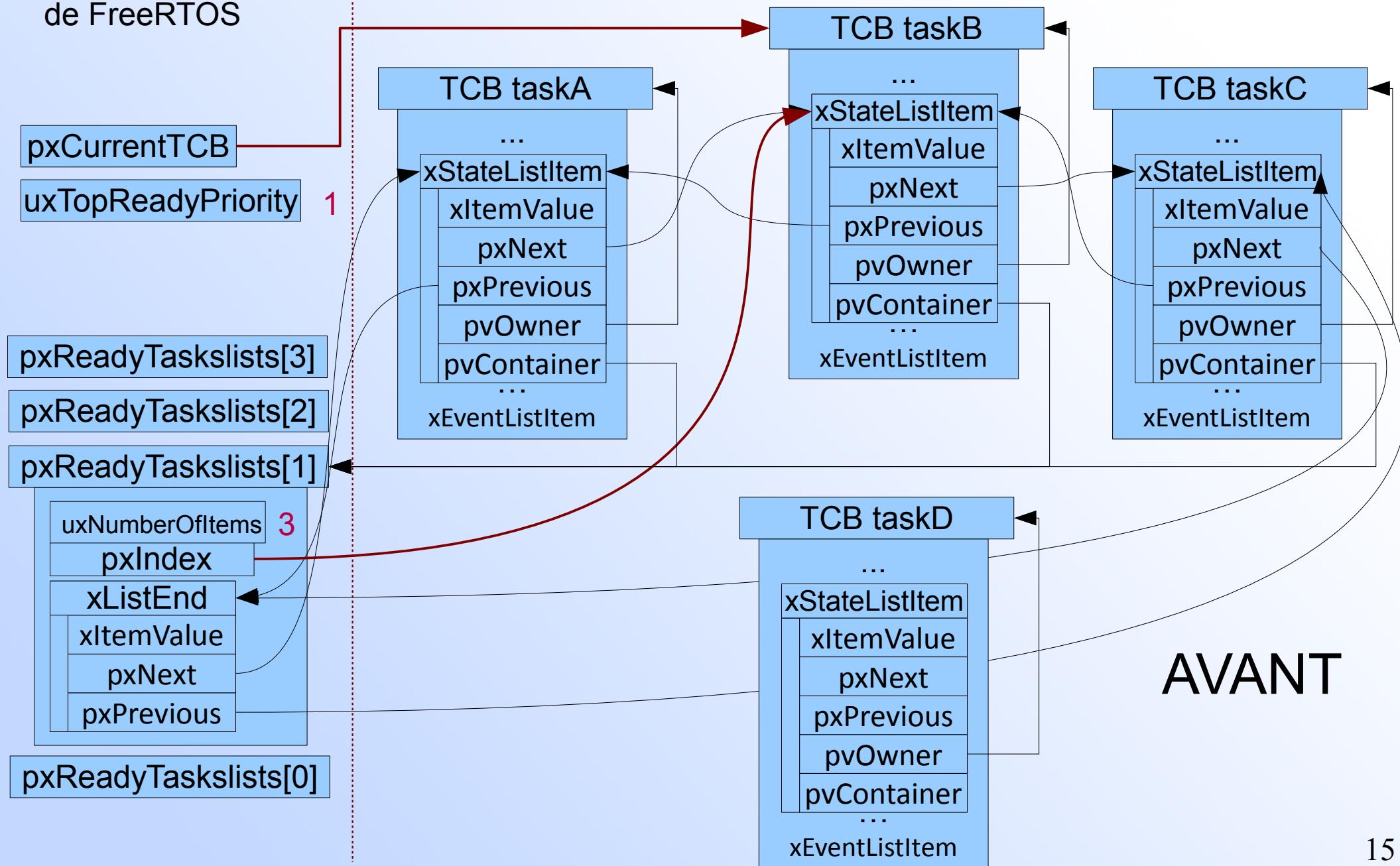
Variables d'état  
de FreeRTOS



# Supprimer une tâche d'une liste de tâches chaînées

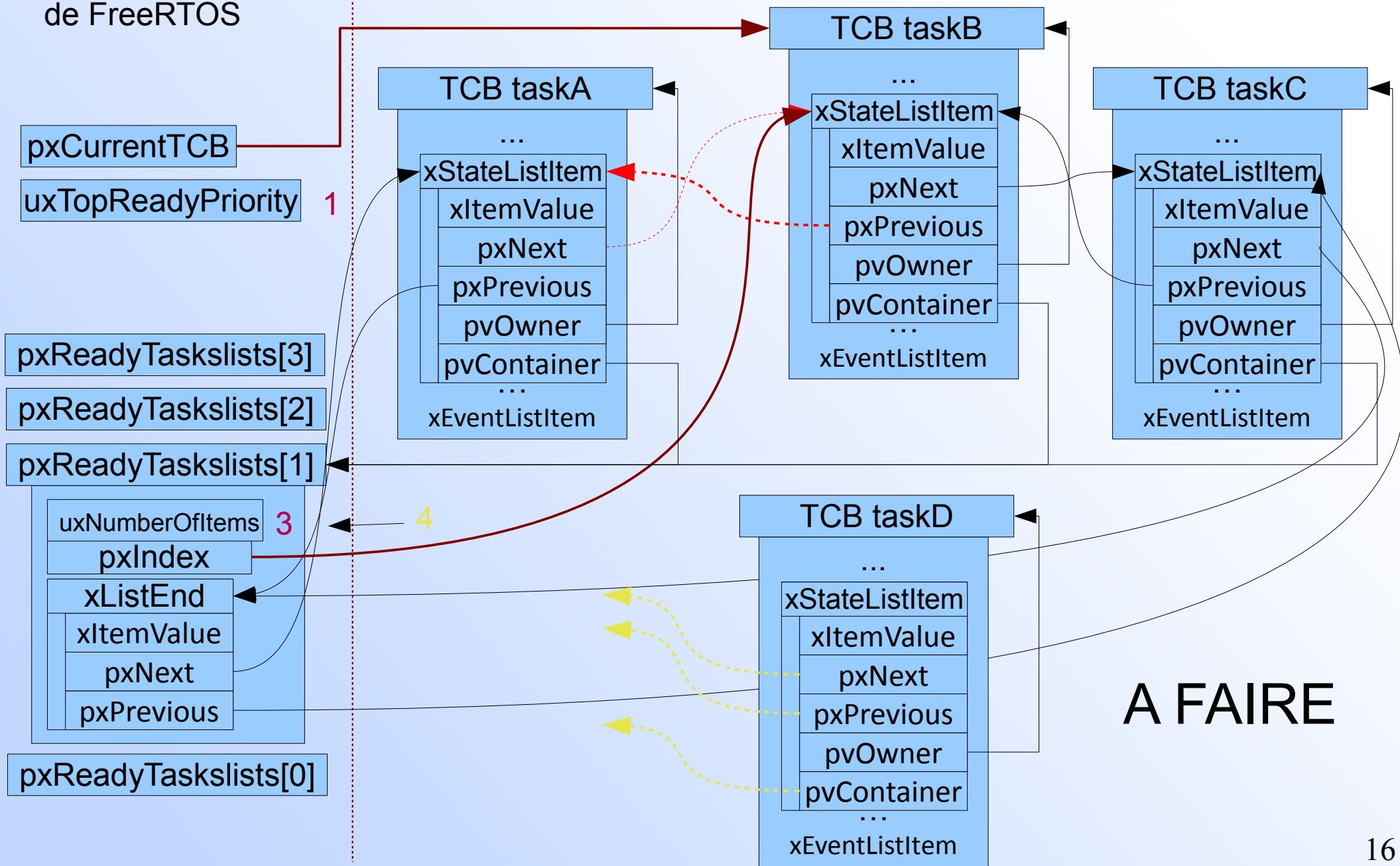
Variables d'état  
de FreeRTOS



Insérer une tâche  
dans une liste de tâches chaînéesVariables d'état  
de FreeRTOS

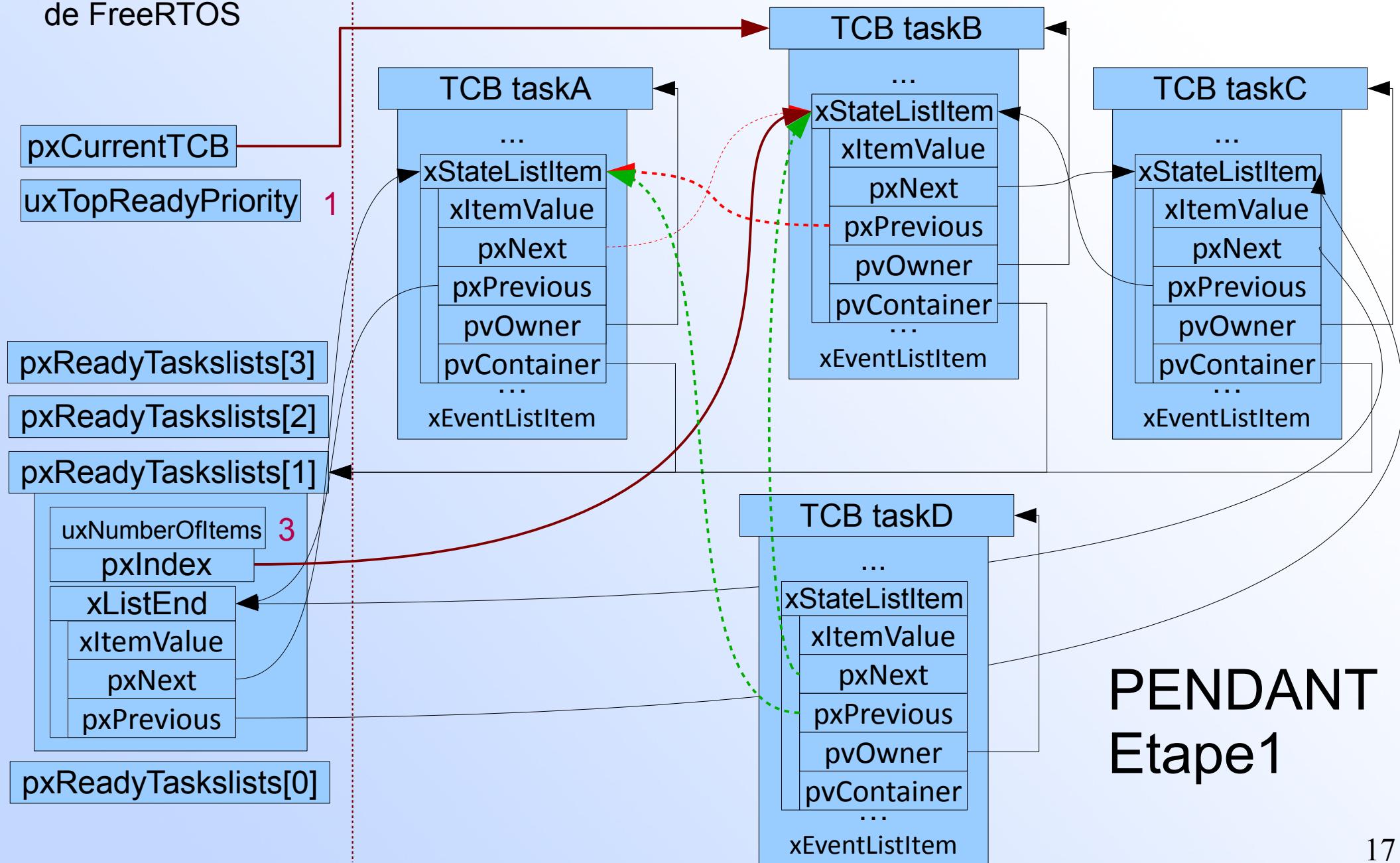
## Insérer une tâche

## dans une liste de tâches chaînées

Variables d'état  
de FreeRTOS

Variables d'état  
de FreeRTOS

## dans une liste de tâches chaînées



Variables d'état  
de FreeRTOS

pxCurrentTCB

uxTopReadyPriority

pxReadyTasksLists[3]

pxReadyTasksLists[2]

pxReadyTasksLists[1]

pxReadyTasksLists[0]

XpendingReadyList

xDelayedTasklist

xOverflowDelayedTaskList

xSuspendedTaskList

xTasksWaitingTermination

# États d'une tâche attente d'un délai ou d'un évènement



TCB task
pxTopOfStack
xStateListItem
xItemValue
pxNext
pxPrevious
pvOwner
pvContainer
xEventListItem
xItemValue
pxNext
pxPrevious
pvOwner
pvContainer
uxPriority
pxStack

## Différer une tâche : liste de délai

Tâche prévue dans 1900 Ticks : Rendez vous au Tick 5100

Variables d'état  
de FreeRTOS

pxCurrentTCB

uxTopReadyPriority 1

pxReadyTaskslists[3]

pxReadyTaskslists[2]

pxReadyTaskslists[1]

pxReadyTaskslists[0]

xDelayedTasklist1

uxNumberOfItems 3

pxIndex

xListEnd

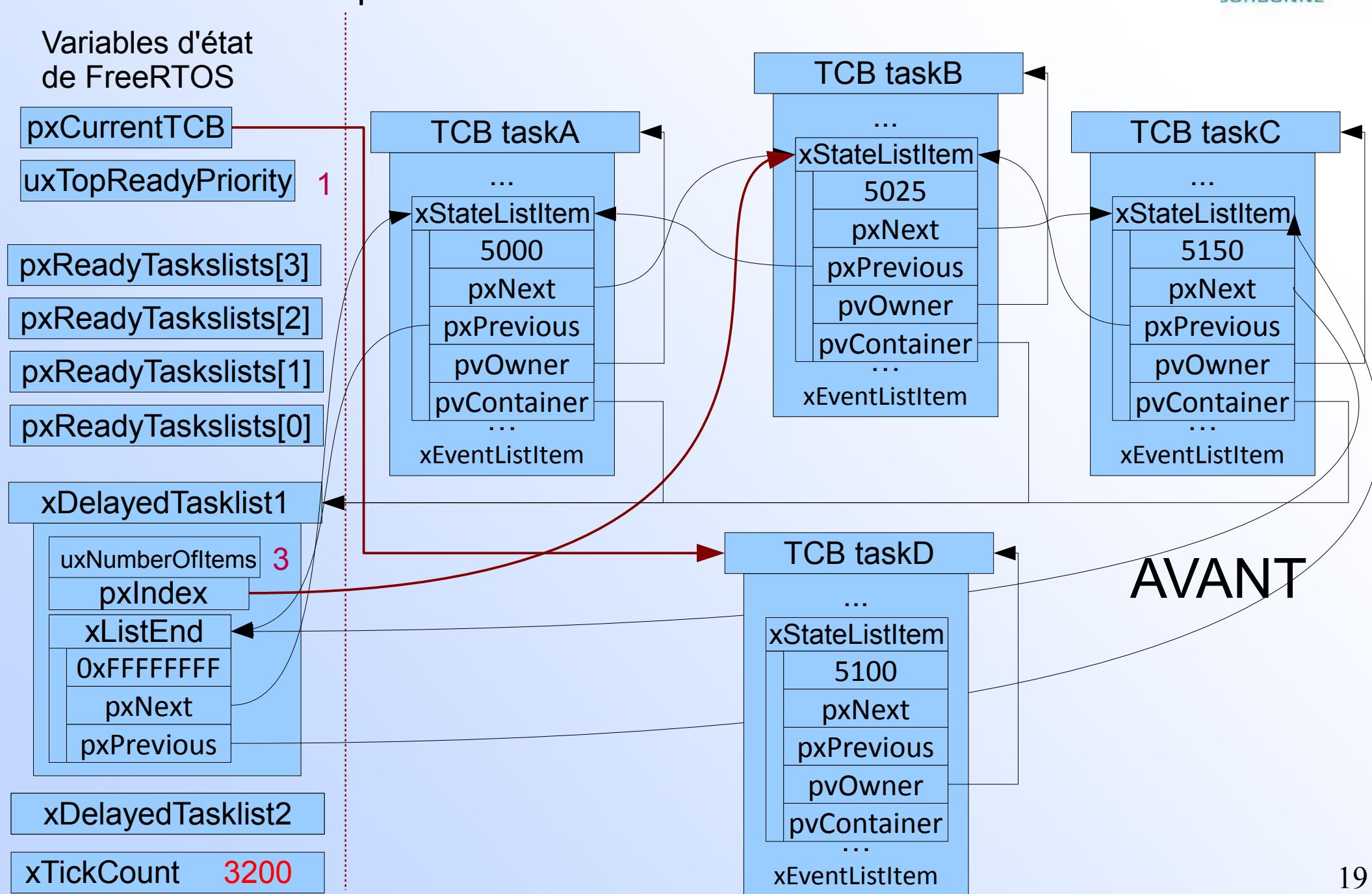
0xFFFFFFFF

pxNext

pxPrevious

xDelayedTasklist2

xTickCount 3200



# EI2I4 RTOS 20/21 Différer une tâche : liste de délai :

Le calcul déborde : Rendez vous au Tick 120 < xTickCount

Variables d'état  
de FreeRTOS

pxCurrentTCB

uxTopReadyPriority 1

pxReadyTaskslists[3]

pxReadyTaskslists[2]

pxReadyTaskslists[1]

pxReadyTaskslists[0]

xDelayedTasklist1

uxNumberOfItems 3

pxIndex

xListEnd

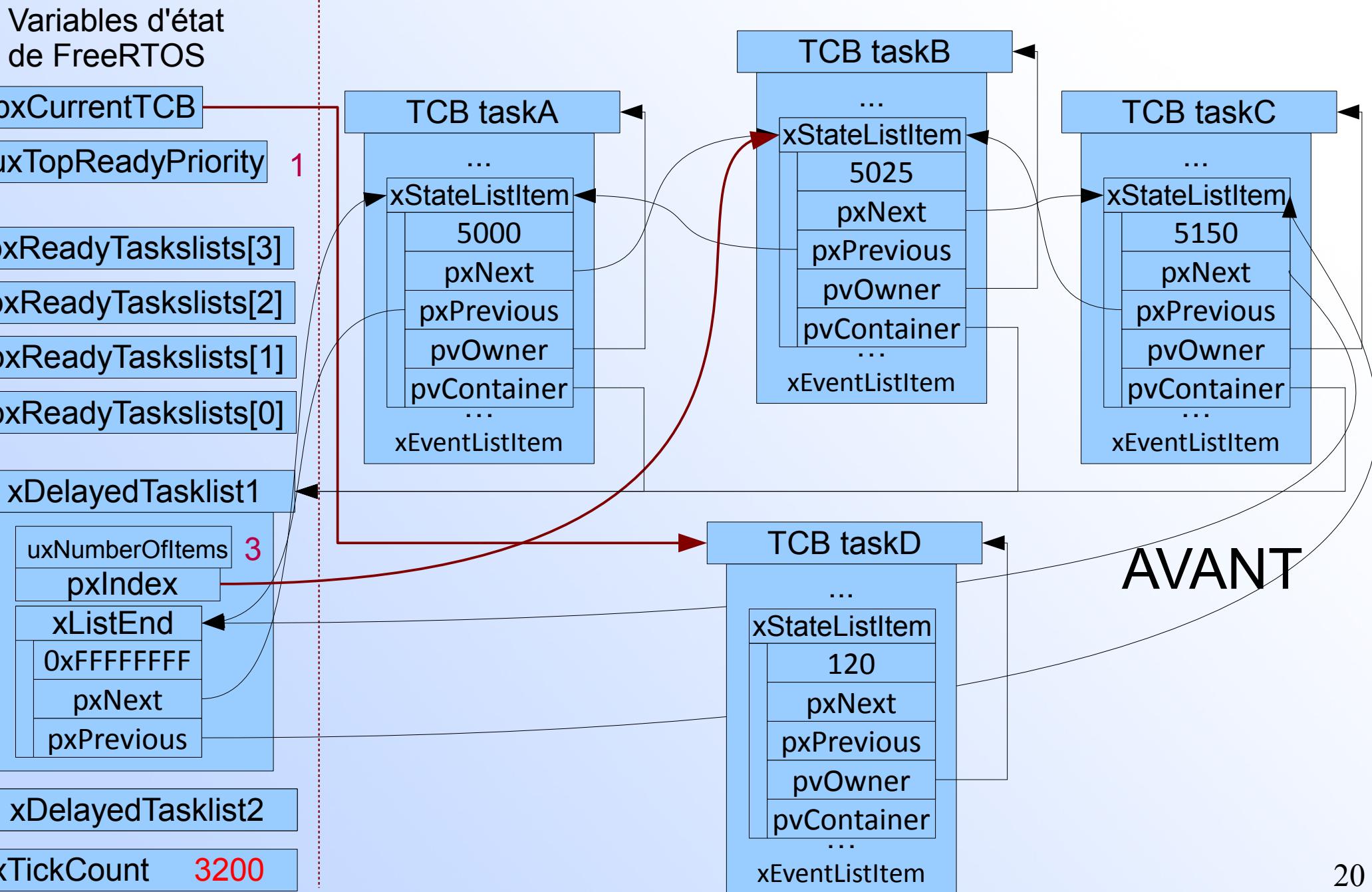
0xFFFFFFFF

pxNext

pxPrevious

xDelayedTasklist2

xTickCount 3200



# Différer une tâche :

## Listes de délai en ping pong

Variables d'état  
de FreeRTOS

pxCurrentTCB

uxTopReadyPriority 1

pxReadyTaskslists[3]

pxReadyTaskslists[2]

pxReadyTaskslists[1]

pxReadyTaskslists[0]

xDelayedTasklist1

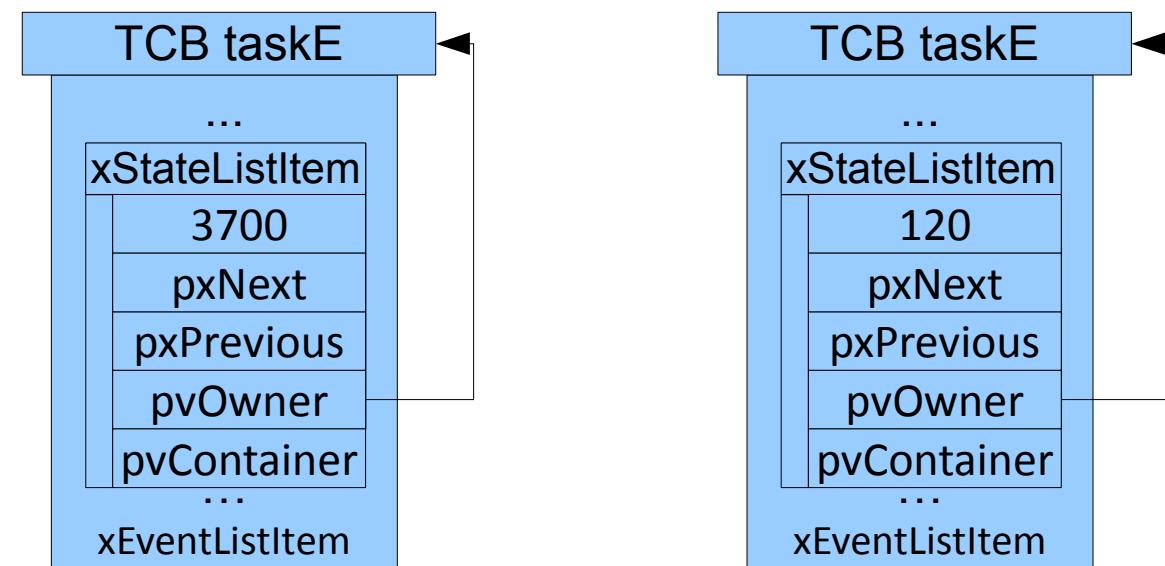
xDelayedTasklist2

pxDelayedTasklist

pxOverflowDelayedTasklist

xTickCount 3200

On range une tâche à retarder :  
dans la liste "delayed"  
dans la liste "Overflowdelayed"



Quand xTickCount déborde :

La liste pointée par pxDelayedTasklist sera vide

On échange les deux pointeurs !!!!

Variables d'état  
de FreeRTOS

pxCurrentTCB

uxTopReadyPriority

pxReadyTasksLists[3]

pxReadyTasksLists[2]

pxReadyTasksLists[1]

pxReadyTasksLists[0]

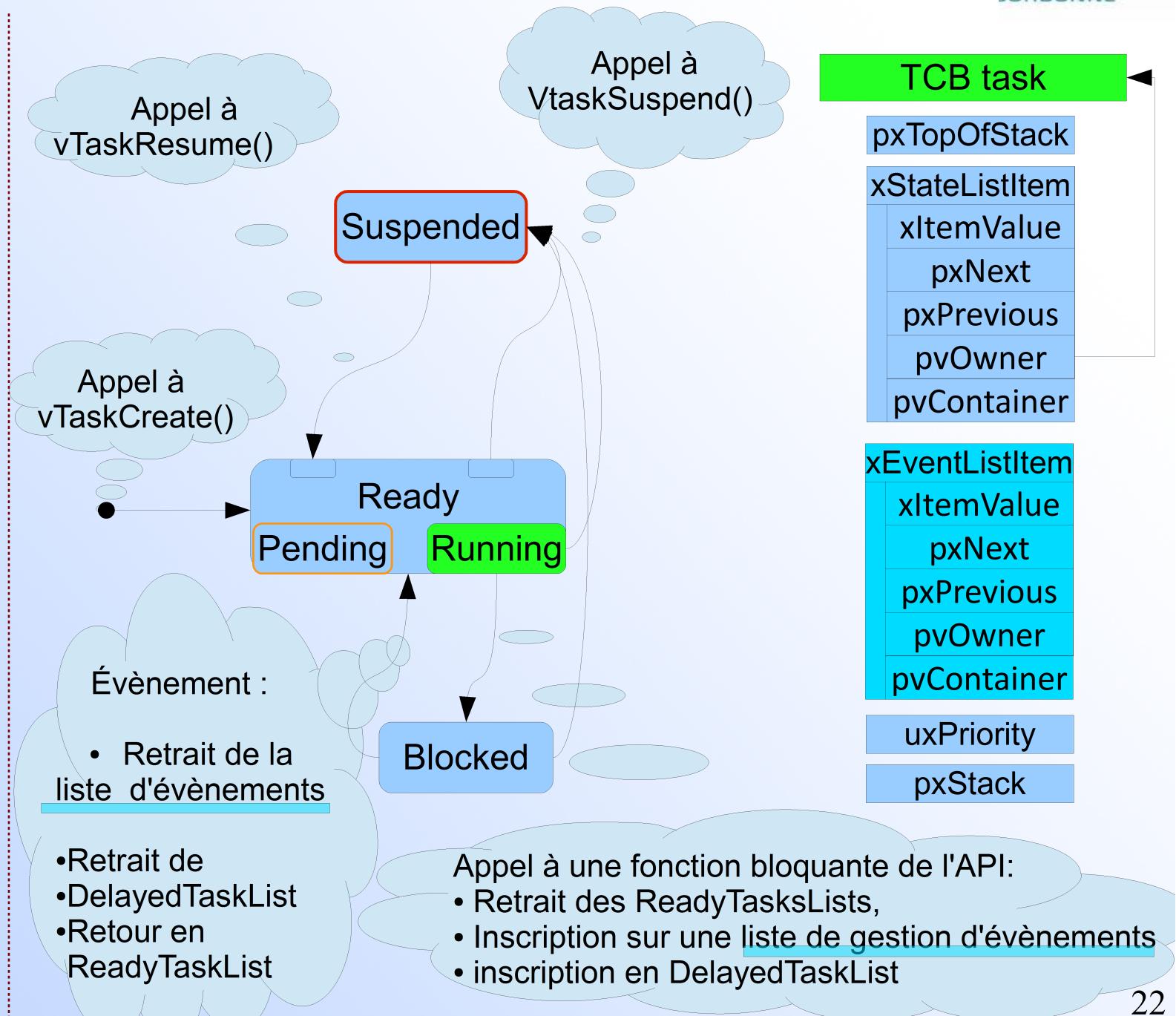
XpendingReadyList

xDelayedTasklist

xOverflowDelayedTaskList

xSuspendedTaskList

xTasksWaitingTermination



# EI2I4 2023 xPortSysTickHandler : réduction des délais RTOS

## appel à xTaskIncrementTick()

```
BaseType_t xTaskIncrementTick( void )
{
    TCB_t *pxTCB; TickType_t xItemValue; BaseType_t xSwitchRequired = pdFALSE;

    if( uxSchedulerSuspended == ( UBaseType_t ) pdFALSE )
    {
        if( ++xConstTickCount == ( TickType_t ) 0U ) { taskSWITCH_DELAYED_LISTS(); }

        if( xConstTickCount >= xNextTaskUnblockTime )
        {
            for( ; )
            {
                if( listLIST_IS_EMPTY( pxDelayedTaskList ) != pdFALSE )
                {
                    xNextTaskUnblockTime = portMAX_DELAY; break;
                }
                else
                {
                    pxTCB = listGET_OWNER_OF_HEAD_ENTRY( pxDelayedTaskList );
                    xItemValue = listGET_LIST_ITEM_VALUE( &( pxTCB->xStateListItem ) );
                    if( xConstTickCount < xItemValue ){ xNextTaskUnblockTime = xItemValue; break; }

                    ( void ) uxListRemove( &( pxTCB->xStateListItem ) );
                    if( listLIST_ITEM_CONTAINER( &( pxTCB->xEventListItem ) ) != NULL )
                    {
                        ( void ) uxListRemove( &( pxTCB->xEventListItem ) );
                        prvAddTaskToReadyList( pxTCB );
                    }
                }
            }
        }
    }
    /* A task being unblocked cannot cause an immediate context switch if preemption is turned off. */
    #if ( configUSE_PREEMPTION == 1 )
    {if( pxTCB->uxPriority >= pxCurrentTCB->uxPriority )
        {xSwitchRequired = pdTRUE;}
    }
    #endif /* configUSE_PREEMPTION */
}
```

xTaskIncrementTick() : suite...  
partage du temps CPU si

```
/* Tasks of equal priority to the currently running task will share
processing time (time slice) if preemption is on, and the application
writer has not explicitly turned time slicing off. */
#if ( ( configUSE_PREEMPTION == 1 ) && ( configUSE_TIME_SLICING == 1 ) )
{if( listCURRENT_LIST_LENGTH( &( pxReadyTasksLists[ pxCurrentTCB->uxPriority ] ) ) > \
( UBaseType_t ) 1 )
{ xSwitchRequired = pdTRUE;}
}
#endif /* ( ( configUSE_PREEMPTION == 1 ) && ( configUSE_TIME_SLICING == 1 ) ) */

}
else /* uxSchedulerSuspended == ( UBaseType_t ) pdTRUE*/
{ ++uxPendedTicks; }

#if ( configUSE_PREEMPTION == 1 )
{ if( xYieldPending != pdFALSE ) { xSwitchRequired = pdTRUE; }
}
#endif /* configUSE_PREEMPTION */

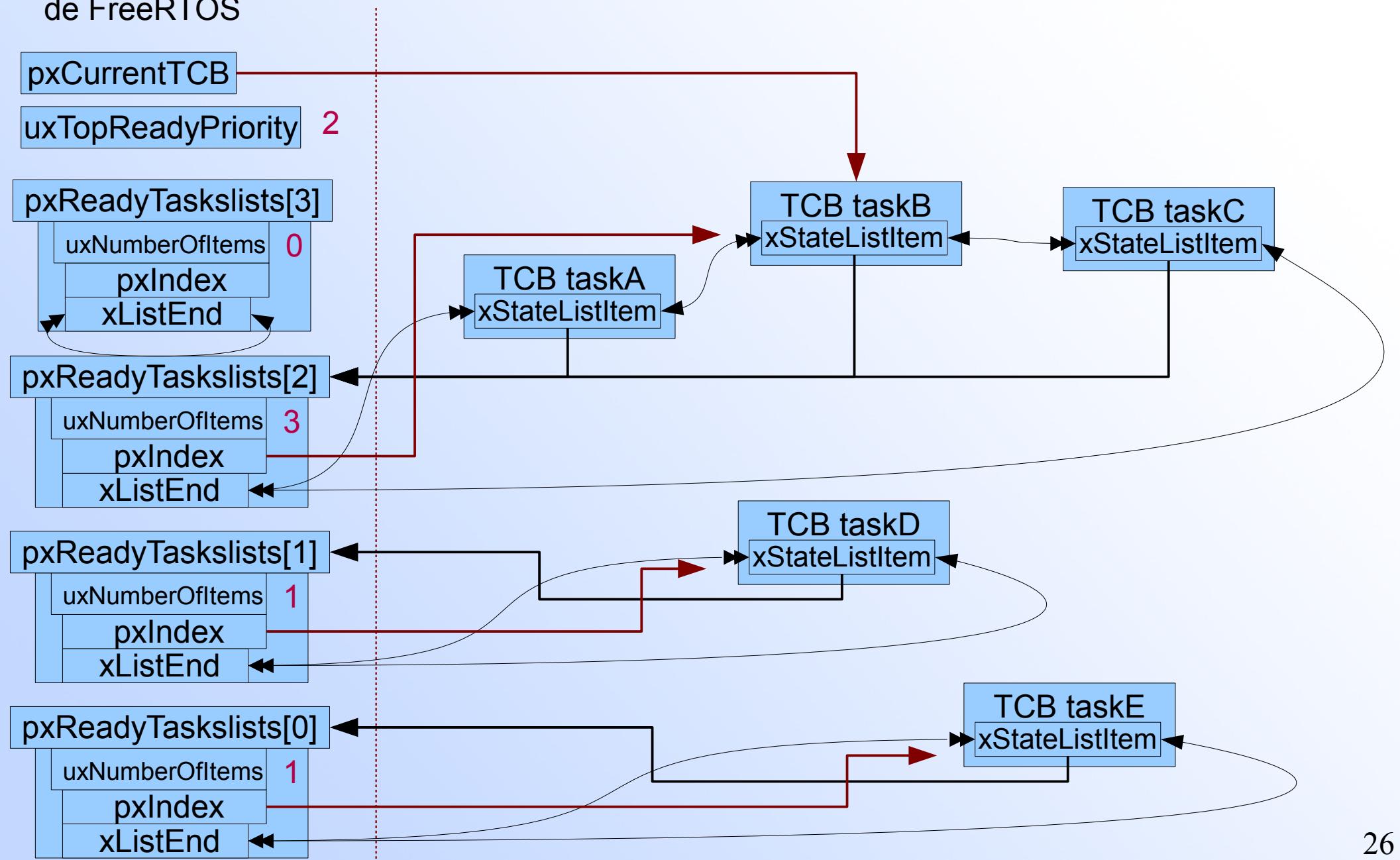
return xSwitchRequired;
}
```

xPortSysTickHandler :  
décision de commuter ou pas après le Tick

```
void xPortSysTickHandler( void )
{
    /* The SysTick runs at the lowest interrupt priority, so when this interrupt
     executes all interrupts must be unmasked. There is therefore no need to
     save and then restore the interrupt mask value as its value is already
     known - therefore the slightly faster vPortRaiseBASEPRI() function is used
     in place of portSET_INTERRUPT_MASK_FROM_ISR(). */
    vPortRaiseBASEPRI();
    {
        /* Increment the RTOS tick. */
        if( xTaskIncrementTick() != pdFALSE )
        {
            /* A context switch is required. Context switching is performed in
             the PendSV interrupt. Pend the PendSV interrupt. */
            portNVIC_INT_CTRL_REG = portNVIC_PENDSVSET_BIT;
        }
    }
    vPortClearBASEPRIFromISR();
}
```

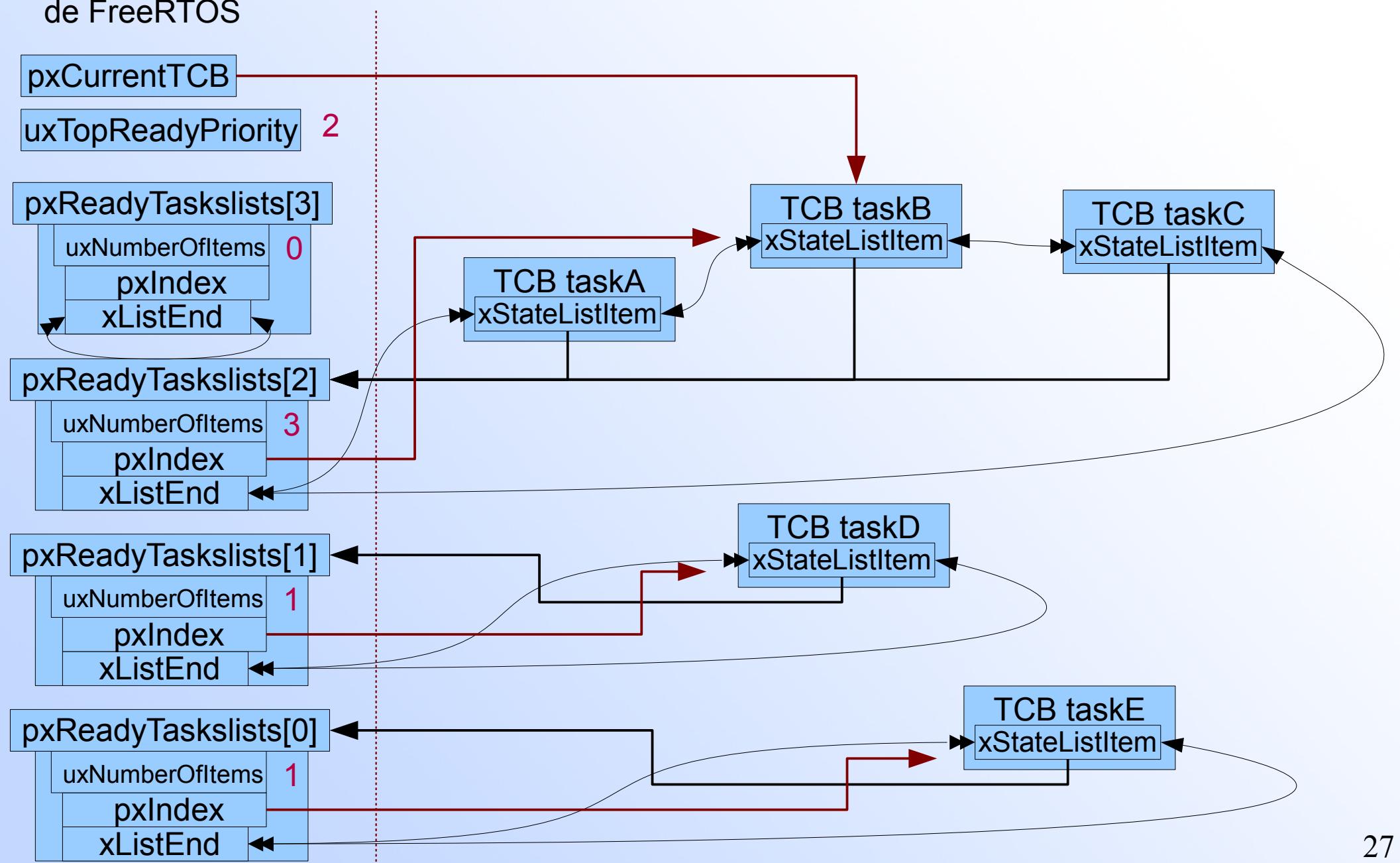
# Chaque niveau de priorité est une liste de tâches chaînées

Variables d'état  
de FreeRTOS



# Comment décider quelle est la tâche suivante ?

Variables d'état  
de FreeRTOS



# RTOS décider quelle est la tâche suivante

```
void vTaskSwitchContext( void ) //tout est déjà prêt pour un choix rapide de la bonne tâche
{
    if( uxSchedulerSuspended != (0) ) {xYieldPending = ( 1 );}
    else      { xYieldPending = ( ( BaseType_t ) 0 ) ;

// on va calculer le niveau de priorité, l'index du tableau de pointeurs pxReadyTasklists[]
// pour savoir dans quelle liste il faut chercher la future tâche.

    UBaseType_t    uxTopPriority = ( 31UL - ( uint32_t ) __clz( ( uxTopReadyPriority ) ) );

//__clz donne le nombre de zéro à gauche du premier bit non nul de uxTopReadyPriority

    List_t * const pxConstList = ( &( pxReadyTasksLists[ uxTopPriority ] ) ); //liste identifiée

    ( pxConstList )->pxIndex = ( pxConstList )->pxIndex->pxNext; //on change de TCB

    if( ( void * ) ( pxConstList )->pxIndex == ( void * ) &( ( pxConstList )->xListEnd ) )
        { ( pxConstList )->pxIndex = ( pxConstList )->pxIndex->pxNext; }

// si la tâche sélectionnée est le connecteur de liste, on passe à la tâche suivante
//pointeur pxCurrentTCB mis à jour

    ( pxCurrentTCB ) = ( pxConstList )->pxIndex->pvOwner;

}
```

# États d'attentes d'une tâche

Variables d'état  
de FreeRTOS

**pxCurrentTCB**

**uxTopReadyPriority**

**pxReadyTasksLists[3]**

**pxReadyTasksLists[2]**

**pxReadyTasksLists[1]**

**pxReadyTasksLists[0]**

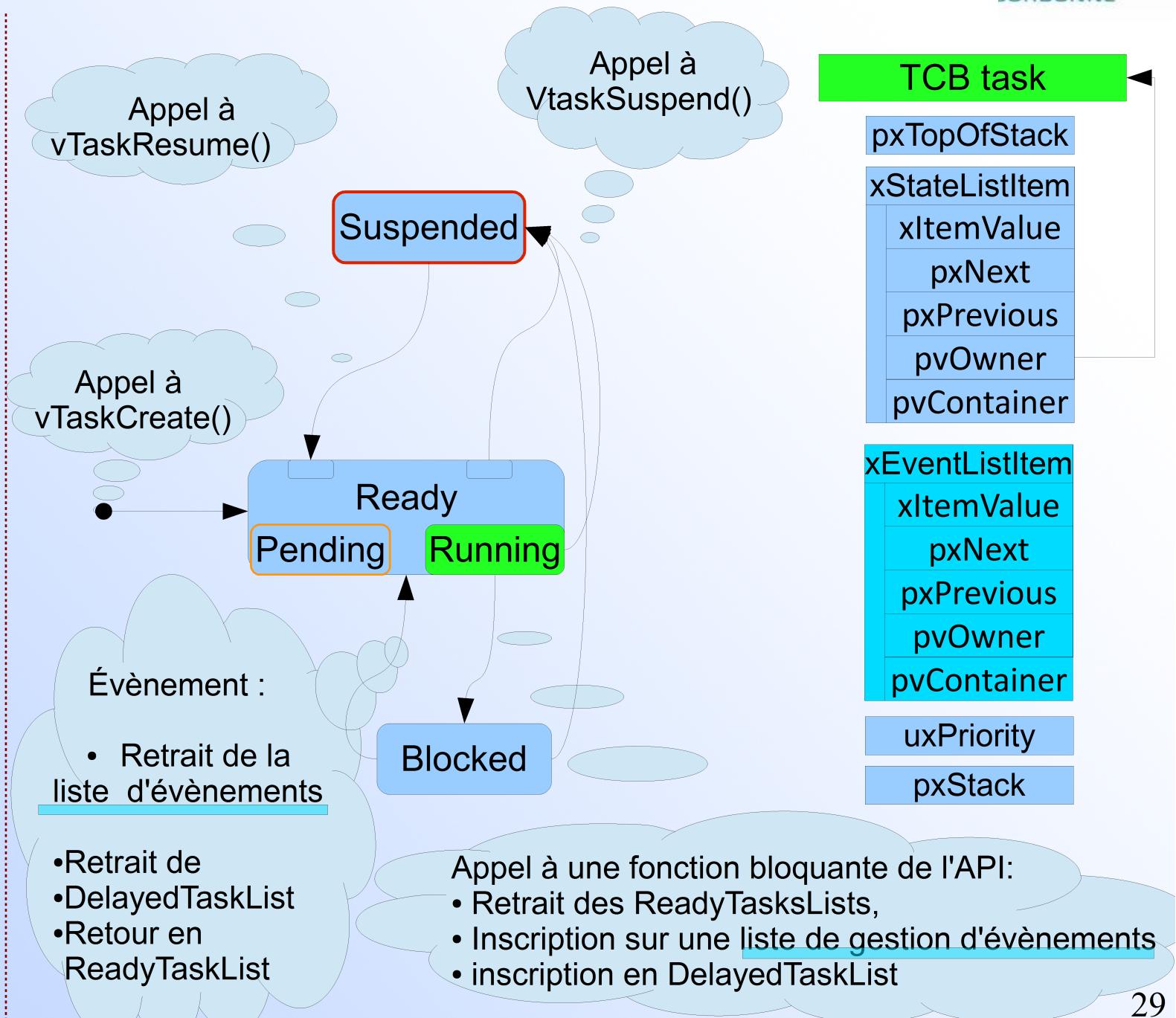
**XpendingReadyList**

**xDelayedTasklist**

**xOverflowDelayedTaskList**

**xSuspendedTaskList**

**xTasksWaitingTermination**



# Concept d'attente

Variables d'état  
de FreeRTOS

pxCurrentTCB

uxTopReadyPriority

pxReadyTasksLists[3]

pxReadyTasksLists[2]

pxReadyTasksLists[1]

pxReadyTasksLists[0]

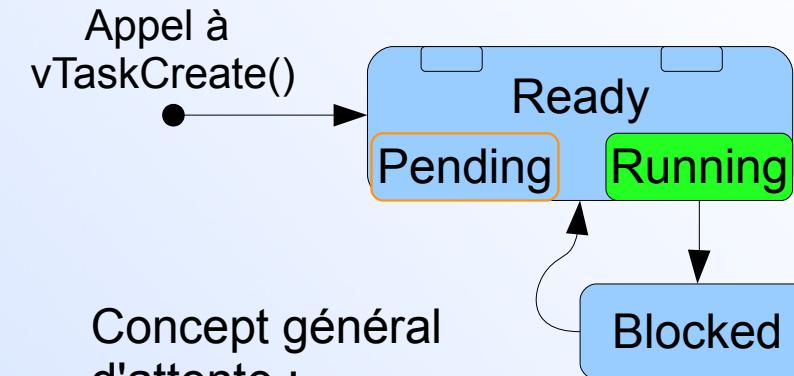
XpendingReadyList

xDelayedTasklist

xOverflowDelayedTaskList

xSuspendedTaskList

xTasksWaitingTermination



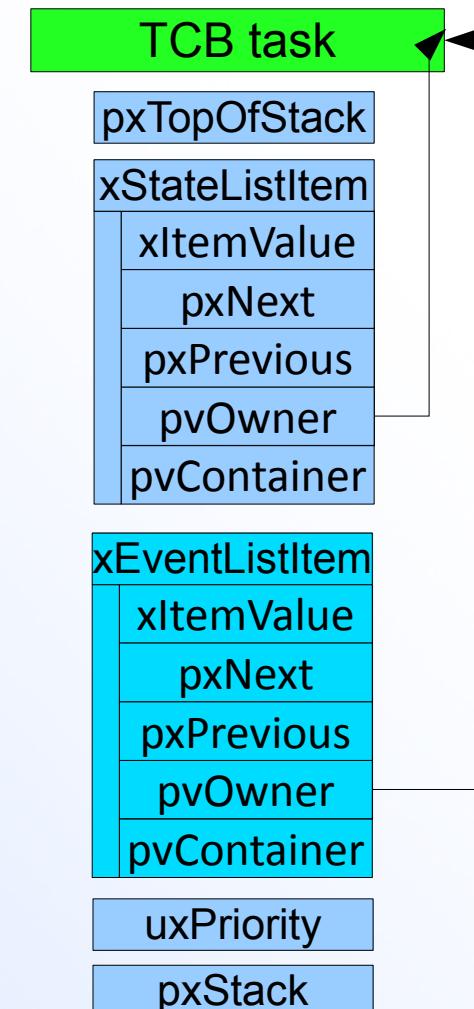
Concept général  
d'attente :

- attente d'un délai
- attente de la disponibilité d'une info stockée dans une queue
- attente d'une mise à jour d'un flux
- attente d'un sémaphore
- attente d'une synchronisation
- attente d'une autorisation d'utilisation d'une ressource partagée

Technique mise en place par l'OS:

Appel à une fonction de l'API:

- Retrait des ReadyTasksLists,
- Inscription sur une liste de gestion d'évènements
- Gestion éventuelle du time\_out
- Commutation de tâche



Conséquence : la tâche ne continue pas son execution et ne bloque pas le CPU à attendre, L'OS va commuter de tâche.

Variables d'état  
de FreeRTOS

pxCurrentTCB

uxTopReadyPriority

pxReadyTasksLists[3]

pxReadyTasksLists[2]

pxReadyTasksLists[1]

pxReadyTasksLists[0]

XpendingReadyList

xDelayedTasklist

xOverflowDelayedTaskList

xSuspendedTaskList

xTasksWaitingTermination

# liste des états d'une tâche

