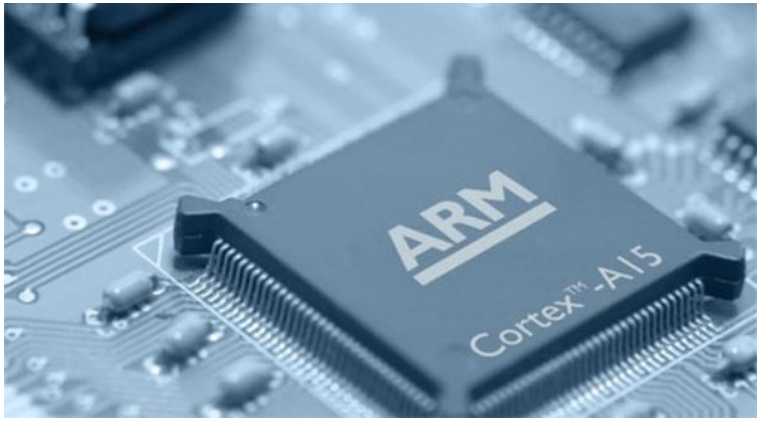


Procesadores ARM - Conmutación de Tareas

Alejandro Furfaro

12 de mayo de 2020

Temario

Temario

Definiciones

Definiciones

Tarea: Unidad de trabajo que un procesador puede despachar, ejecutar, y detener a voluntad, bajo la forma de:

Definiciones

Tarea: Unidad de trabajo que un procesador puede despachar, ejecutar, y detener a voluntad, bajo la forma de:

- La instancia de un programa (o, expresado en términos del lenguaje de teoría de Sistemas Operativos, *proceso*, o *thread*).

Definiciones

Tarea: Unidad de trabajo que un procesador puede despachar, ejecutar, y detener a voluntad, bajo la forma de:

- La instancia de un programa (o, expresado en términos del lenguaje de teoría de Sistemas Operativos, *proceso*, o *thread*).
- Un handler de interrupción, o excepción.

Definiciones

Tarea: Unidad de trabajo que un procesador puede despachar, ejecutar, y detener a voluntad, bajo la forma de:

- La instancia de un programa (o, expresado en términos del lenguaje de teoría de Sistemas Operativos, *proceso*, o *thread*).
- Un handler de interrupción, o excepción.
- Un servicio del S.O. (por ejemplo en Linux, cualquiera de las Syscall).

Definiciones

Definiciones

Espacio de ejecución: Es el conjunto de secciones de código, datos, y pila que componen la tarea.

Definiciones

Espacio de ejecución: Es el conjunto de secciones de código, datos, y pila que componen la tarea.

Contexto de ejecución: Es el conjunto de valores de los registros internos del procesador en cada momento.

Definiciones

Espacio de ejecución: Es el conjunto de secciones de código, datos, y pila que componen la tarea.

Contexto de ejecución: Es el conjunto de valores de los registros internos del procesador en cada momento.

Espacio de Contexto de ejecución: Se compone de un bloque de memoria en el que el S.O. almacenará el contexto completo de ejecución del procesador.

Context Switch

Definición

Salvar y restaurar el estado computacional o contexto de dos procesos o threads cuando se suspende la ejecución del primero (se salva su contexto) para pasar a ejecutar el segundo (se restaura su contexto). Puede incluir, o no, el resguardo y restauración del espacio de memoria (procesos o threads).

Context Switch

Primeras conclusiones

Las tareas en un computador no operan simultáneamente, sino serialmente pero conmutando de una a otra a gran velocidad. Esto significa que se producen cientos o miles de context switches por segundo. Nuestros sentidos no captan la intermitencia de cada tarea, creándose una sensación de simultaneidad.

Entra en juego el Sistema Operativo

Entra en juego el Sistema Operativo

- En particular, el Sistema Operativo tiene un módulo de software llamado ***scheduler***, que trabaja con una lista de tareas a ejecutar.

Entra en juego el Sistema Operativo

- En particular, el Sistema Operativo tiene un módulo de software llamado ***scheduler***, que trabaja con una lista de tareas a ejecutar.
- El ***scheduler*** define un intervalo de tiempo llamado time frame, el que a su vez con ayuda de un temporizador es dividido en intervalos mas pequeños, que se convierten en la unidad de tiempo.

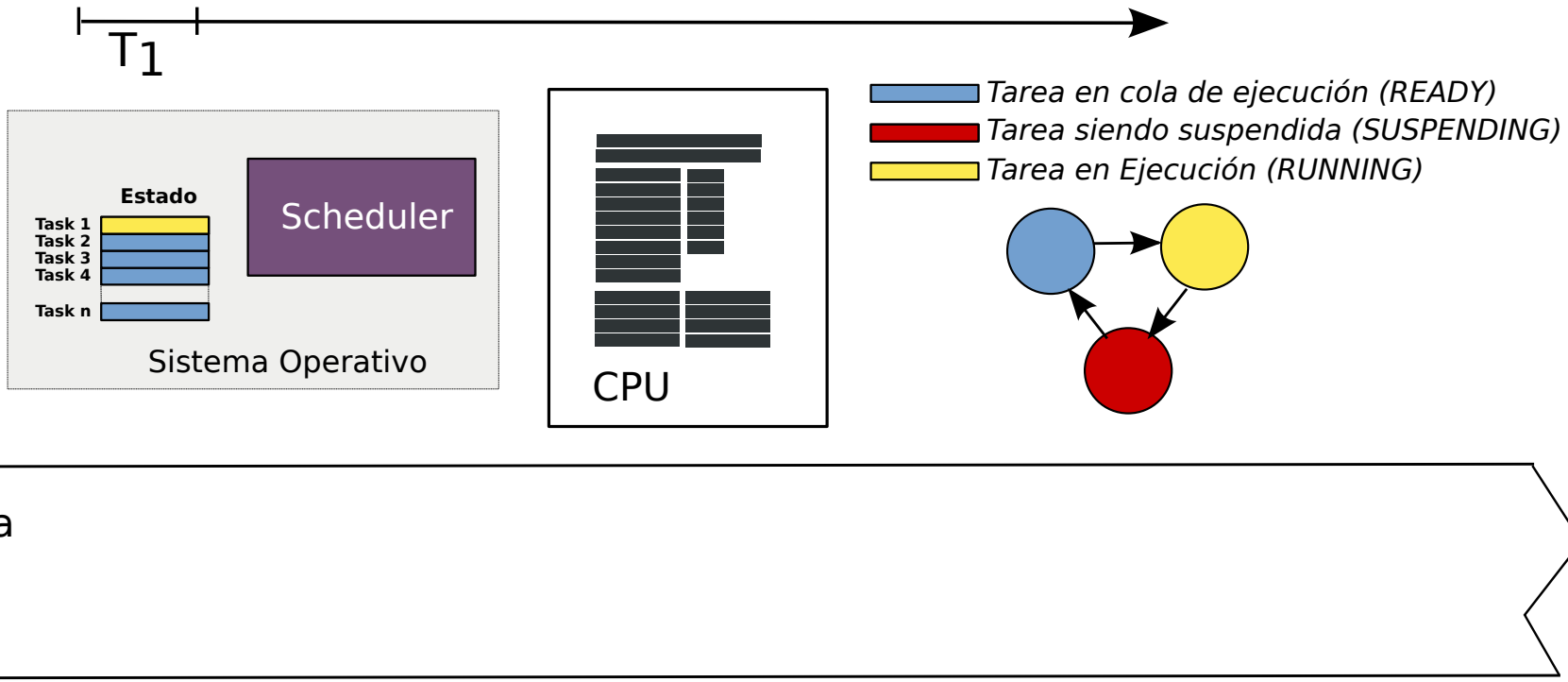
Entra en juego el Sistema Operativo

- En particular, el Sistema Operativo tiene un módulo de software llamado ***scheduler***, que trabaja con una lista de tareas a ejecutar.
- El ***scheduler*** define un intervalo de tiempo llamado time frame, el que a su vez con ayuda de un temporizador es dividido en intervalos mas pequeños, que se convierten en la unidad de tiempo.
- El arte de manejo de prioridades consiste entonces en asignarle a cada tarea un porcentaje del time frame, medido en una cantidad de intervalos unitarios.

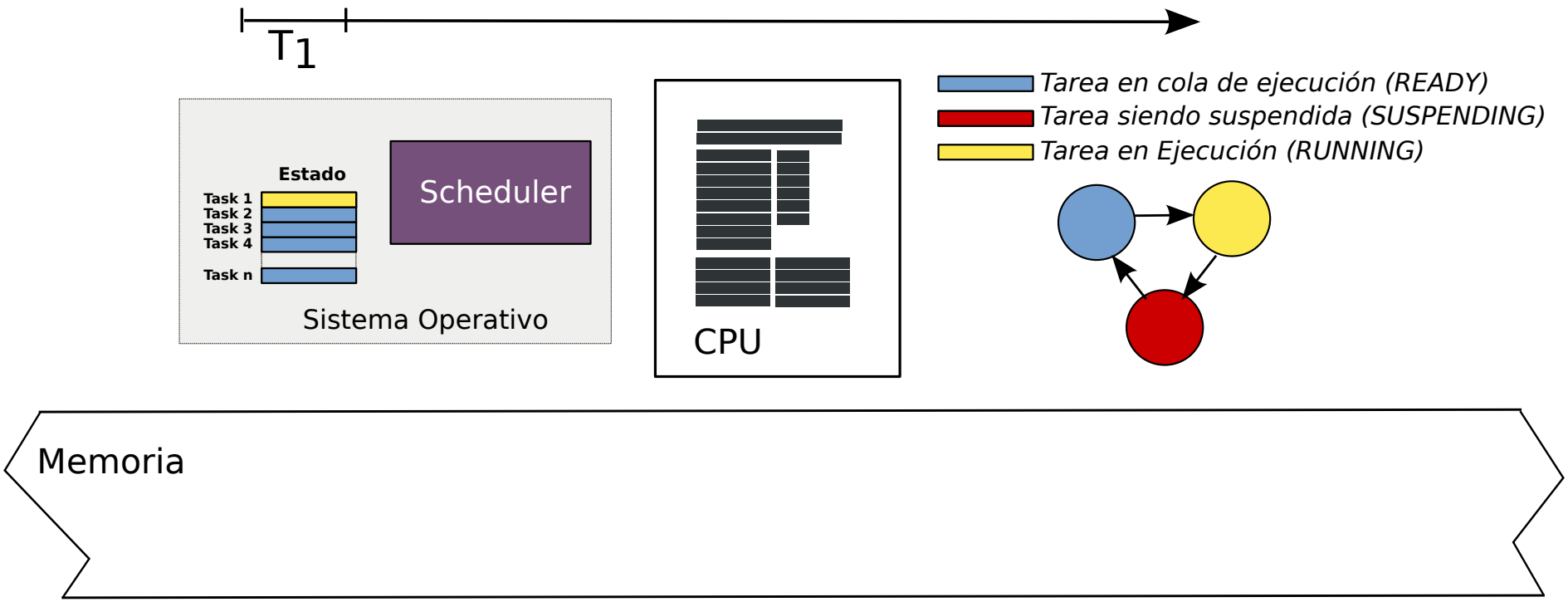
Entra en juego el Sistema Operativo

- En particular, el Sistema Operativo tiene un módulo de software llamado ***scheduler***, que trabaja con una lista de tareas a ejecutar.
- El ***scheduler*** define un intervalo de tiempo llamado time frame, el que a su vez con ayuda de un temporizador es dividido en intervalos mas pequeños, que se convierten en la unidad de tiempo.
- El arte de manejo de prioridades consiste entonces en asignarle a cada tarea un porcentaje del time frame, medido en una cantidad de intervalos unitarios.
- Cada tarea tiene así unos milisegundos para progresar, expirados los cuales suspende una tarea y despacha para su ejecución la siguiente de la lista.

Integrando lo visto hasta aquí

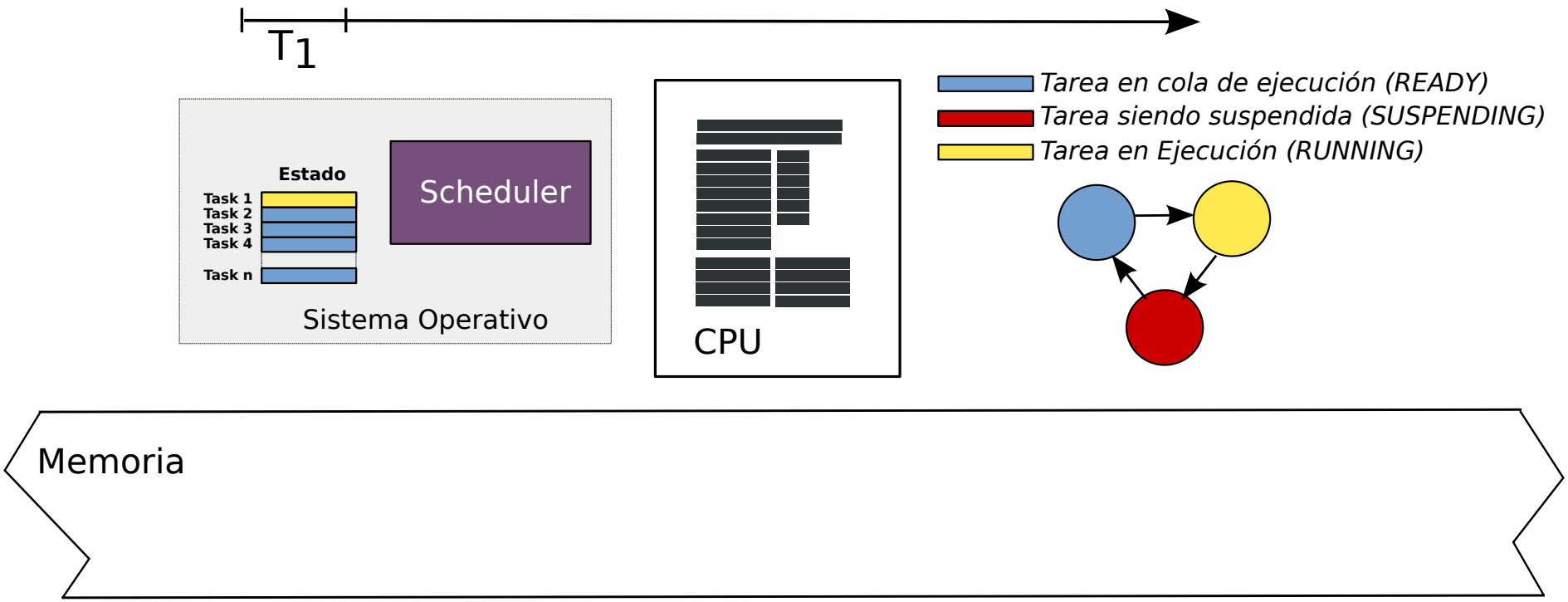


Integrando lo visto hasta aquí



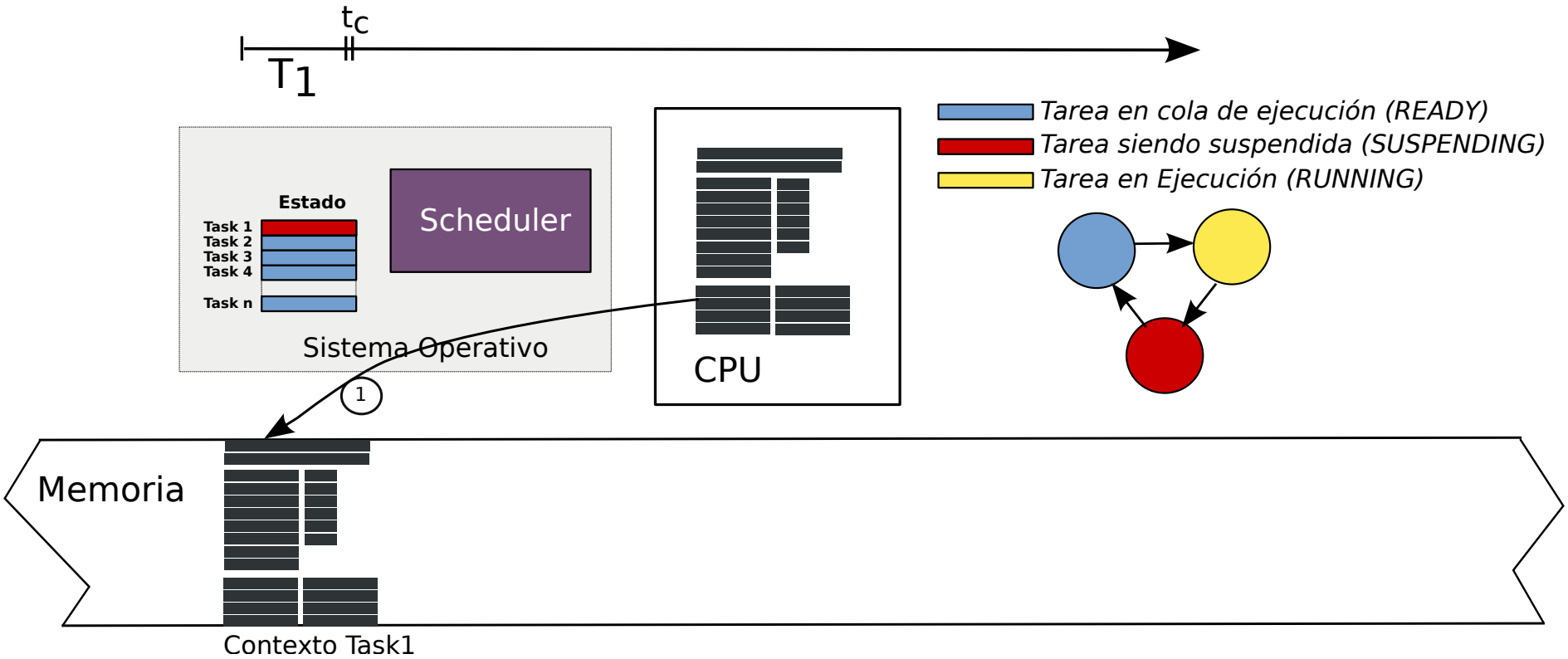
Durante un tiempo T1, el procesador ejecuta la tarea *Task1* de la lista que maneja el ***scheduler***.

Integrando lo visto hasta aquí

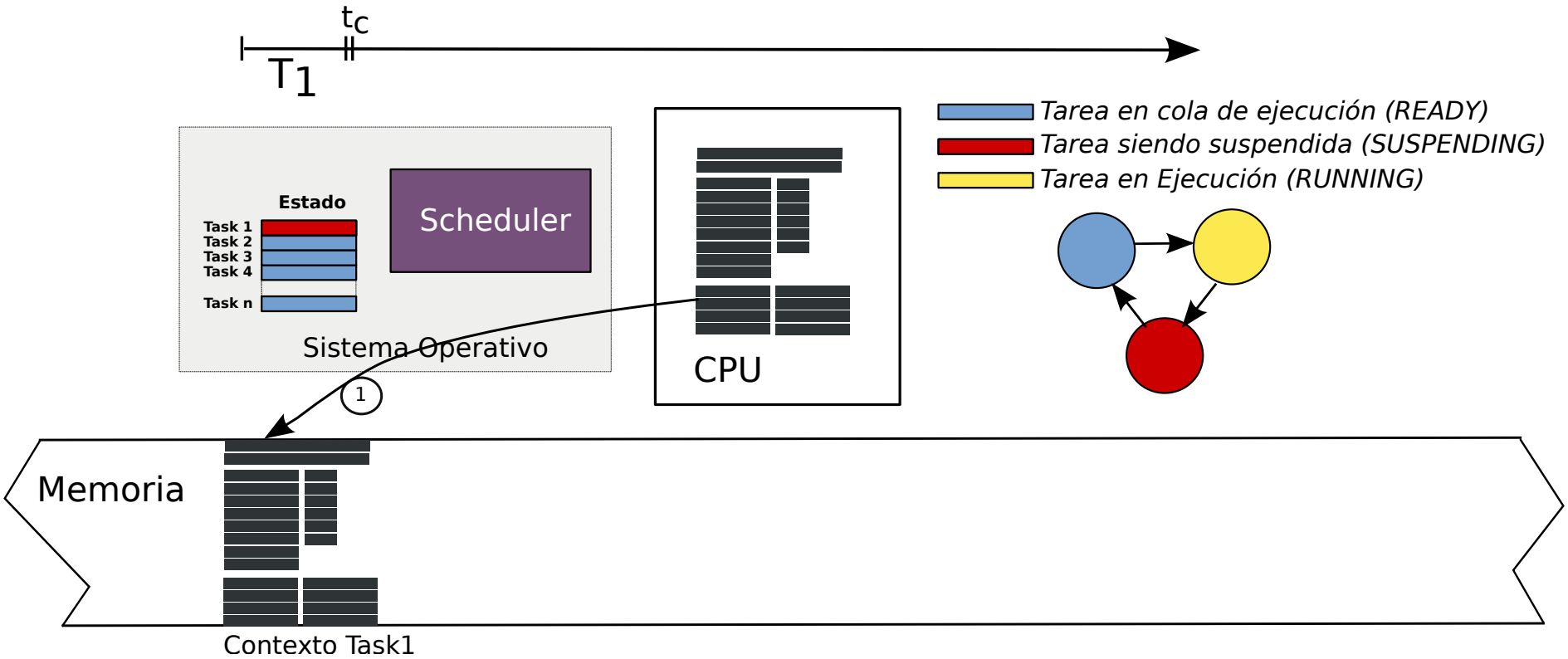


Durante un tiempo T1, el procesador ejecuta la tarea *Task1* de la lista que maneja el ***scheduler***.
El tiempo T1, que se le asignó para ejecución es medida de la prioridad de esta tarea.

Integrando lo visto hasta aquí

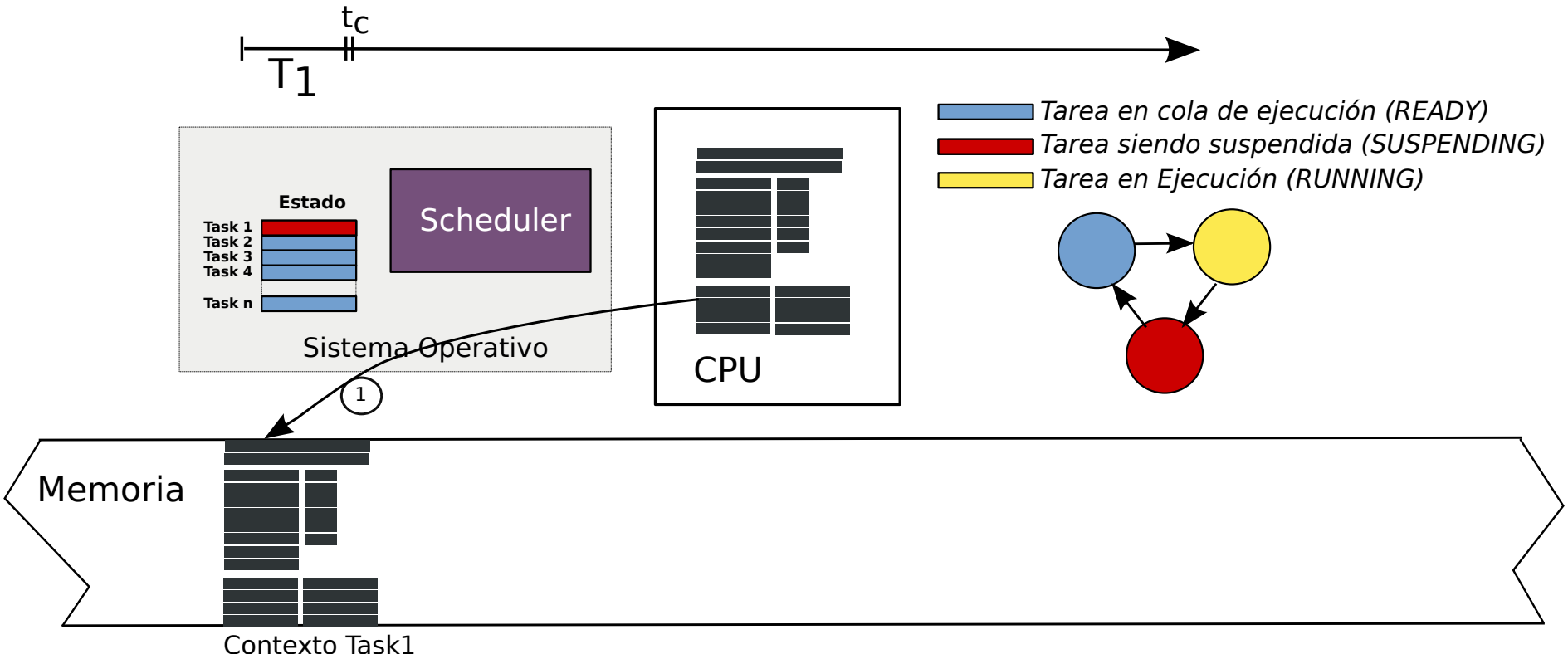


Integrando lo visto hasta aquí



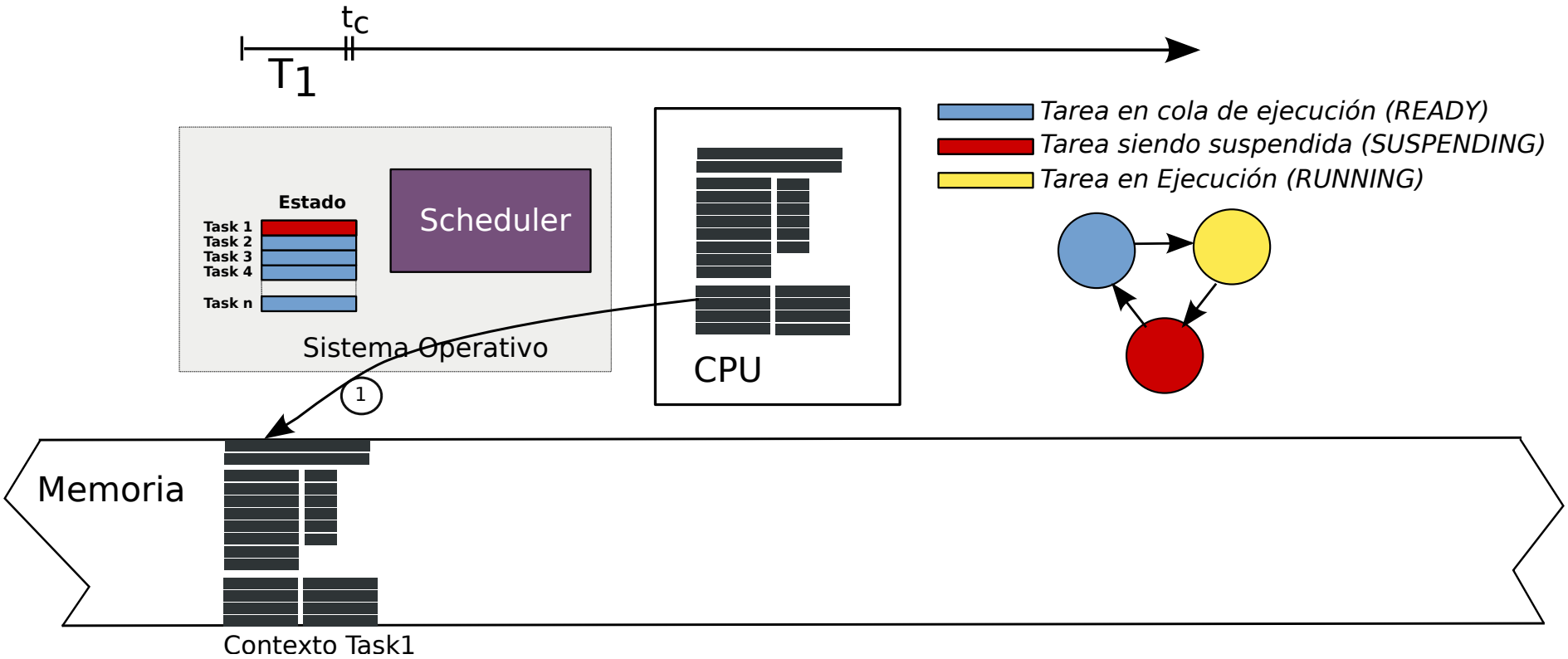
Una vez expirado T1, el ***scheduler*** toma el control del sistema y como la tarea *Task1* no finalizó, la suspende, asignándole estado **SUSPENDING**.

Integrando lo visto hasta aquí



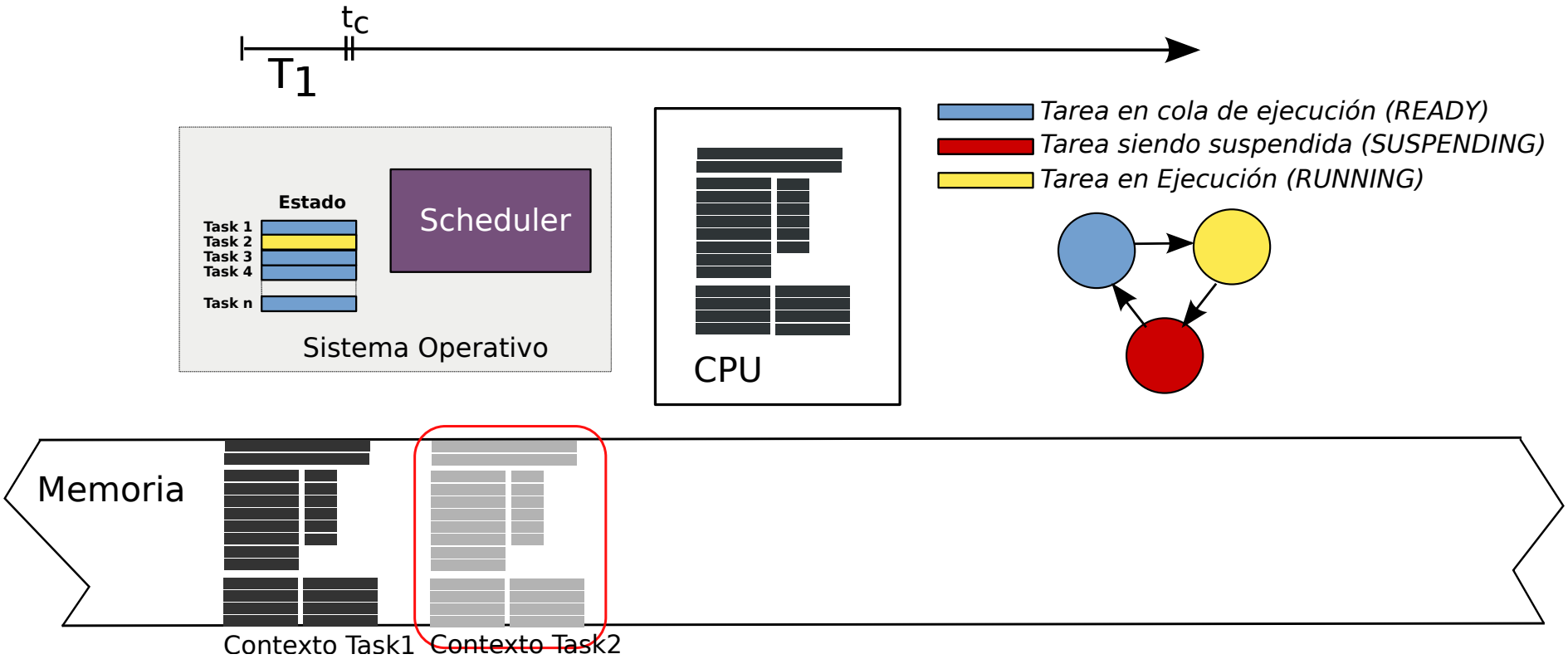
Una vez expirado T1, el ***scheduler*** toma el control del sistema y como la tarea *Task1* no finalizó, la suspende, asignándole estado **SUSPENDING**.
Resguarda el contexto de *Task1* en un área de **memoria del kernel** (Flujo 1).

Integrando lo visto hasta aquí

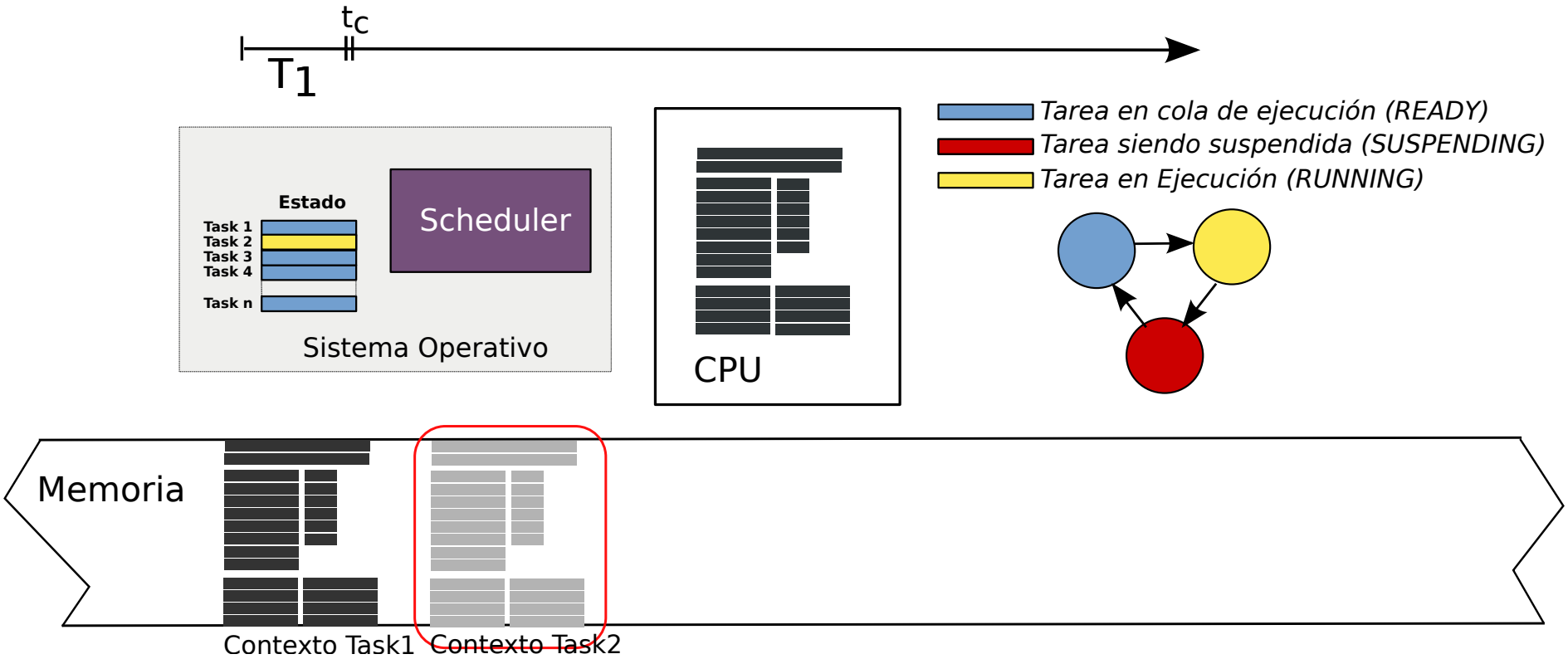


Una vez expirado T1, el ***scheduler*** toma el control del sistema y como la tarea *Task1* no finalizó, la suspende, asignándole estado **SUSPENDING**.
Resguarda el contexto de *Task1* en un área de **memoria del kernel** (Flujo 1).
Contexto: Valor de los registros del procesador al momento de suspender la tarea.

Integrando lo visto hasta aquí

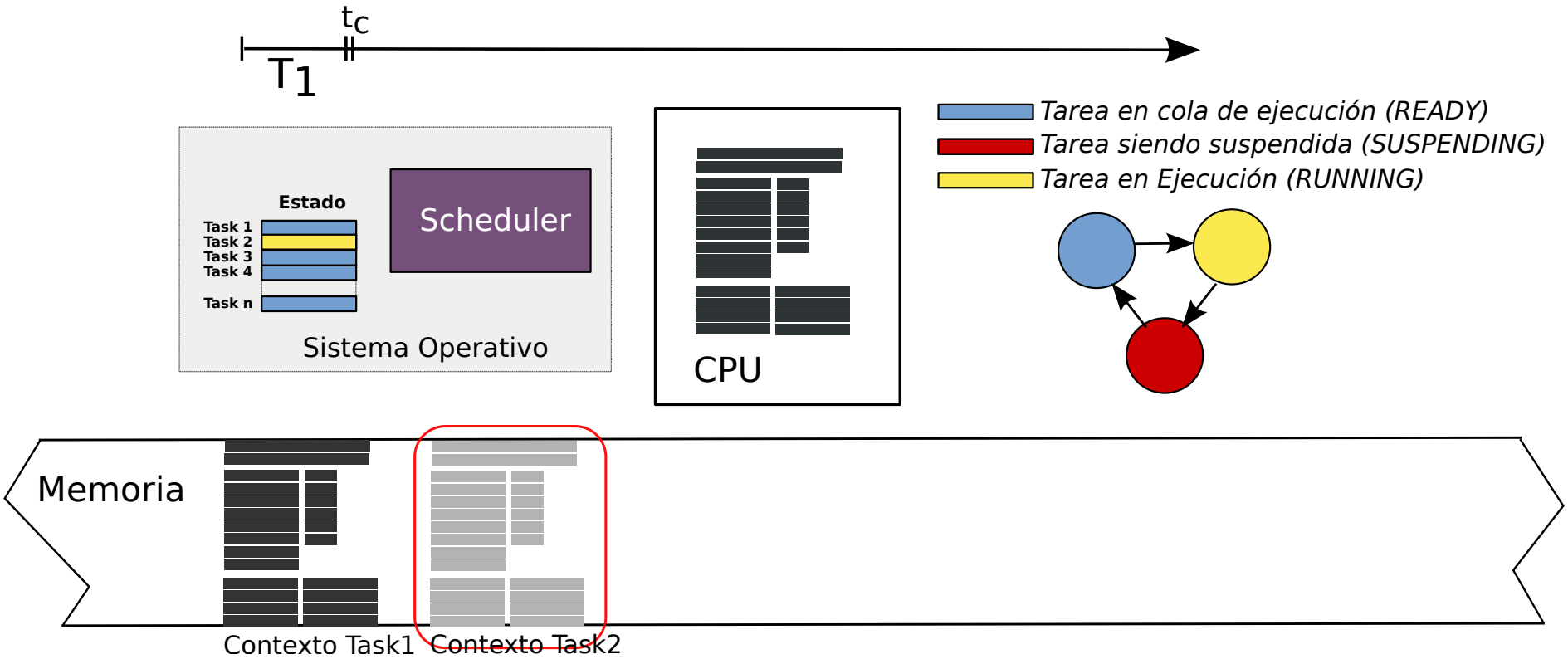


Integrando lo visto hasta aquí



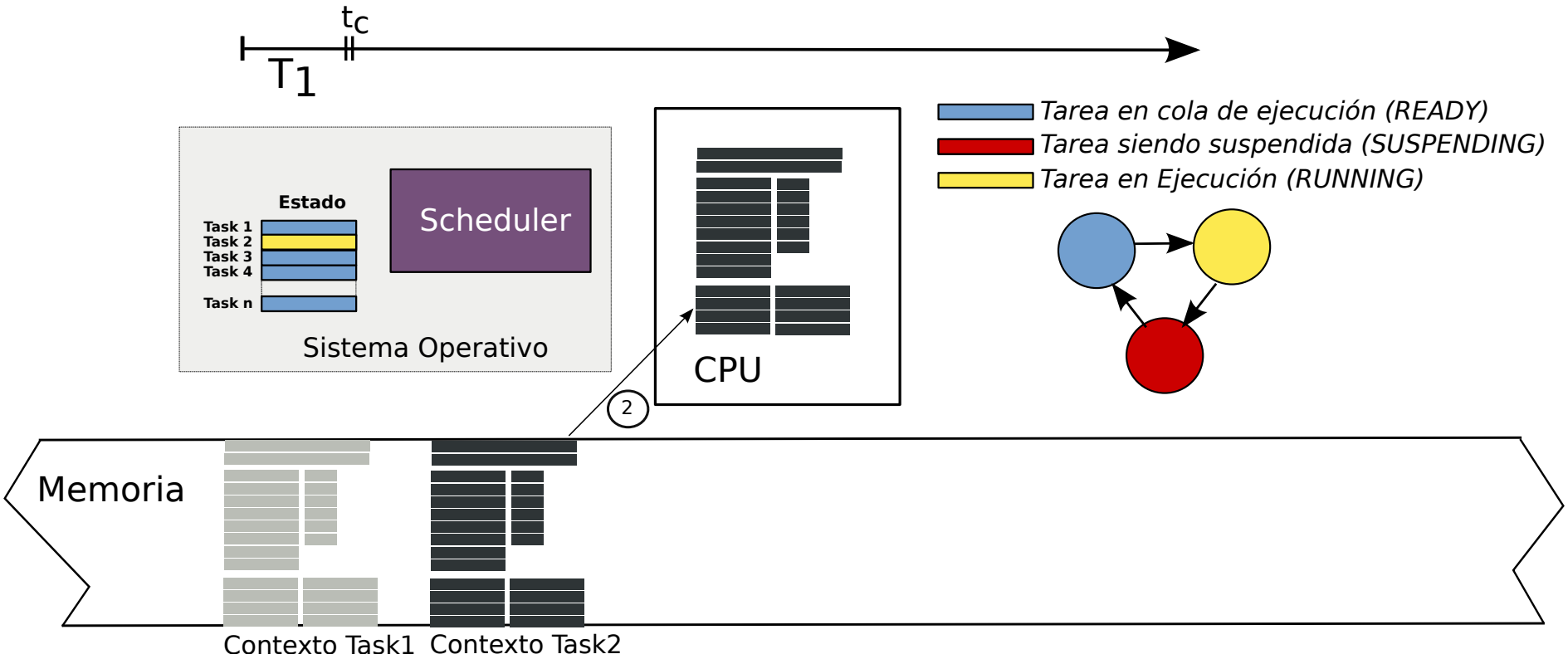
A partir del inicio de la transferencia del contexto de *Task1* a la **memoria del kernel** inicia un intervalo que denominaremos t_c (tiempo de conmutación).

Integrando lo visto hasta aquí

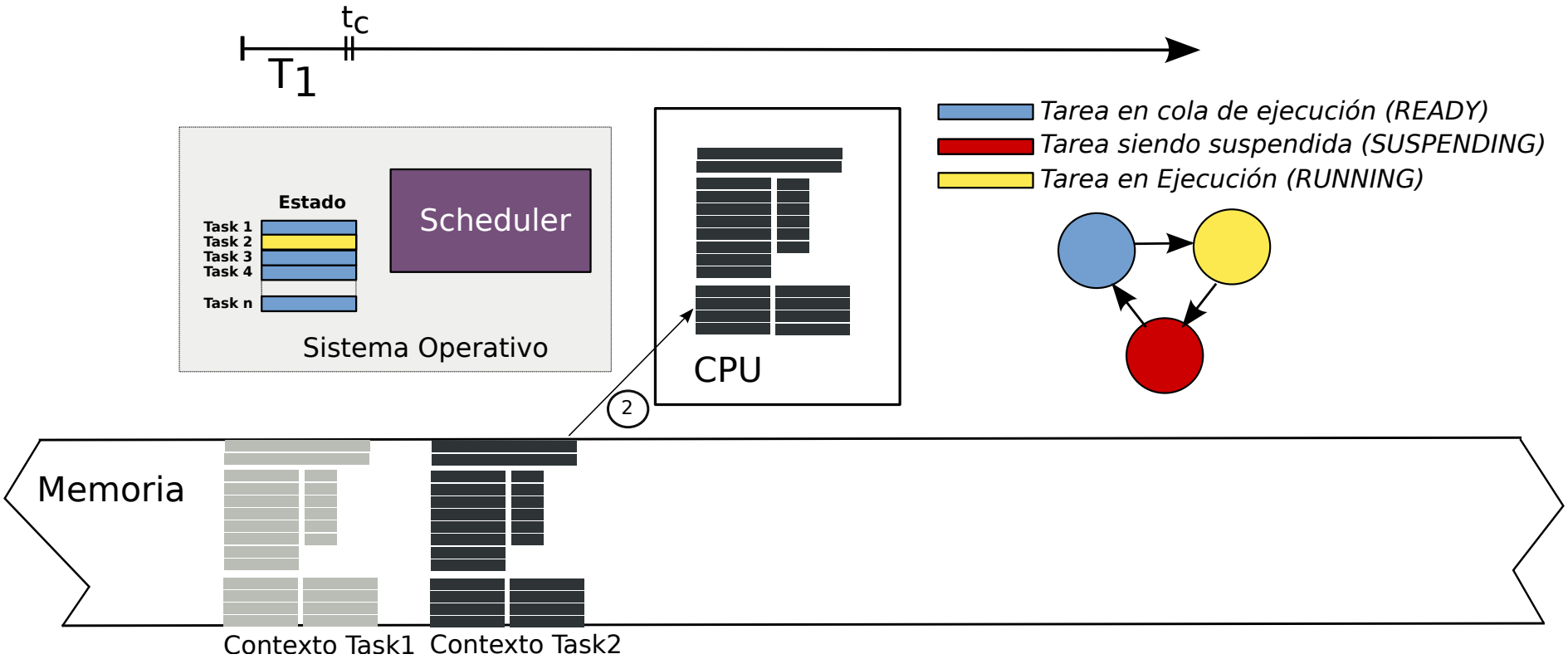


A partir del inicio de la transferencia del contexto de *Task1* a la **memoria del kernel** inicia un intervalo que denominaremos t_c (tiempo de conmutación). Seguidamente el ***scheduler*** busca la siguiente tarea en la lista de tareas en ejecución (en el gráfico *Task2*), identifica la dirección de memoria de kernel en donde está almacenado su contexto, y marca a la tarea **RUNNING**.

Integrando lo visto hasta aquí

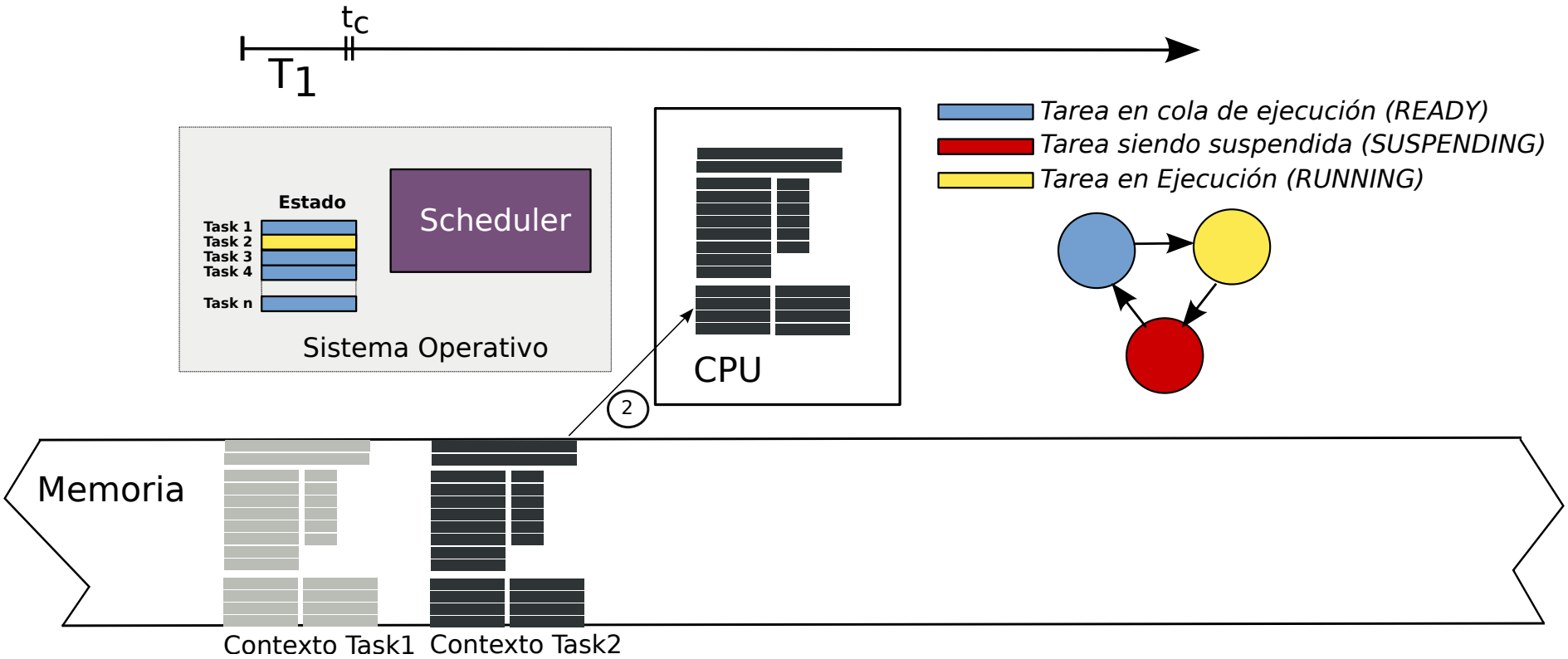


Integrando lo visto hasta aquí



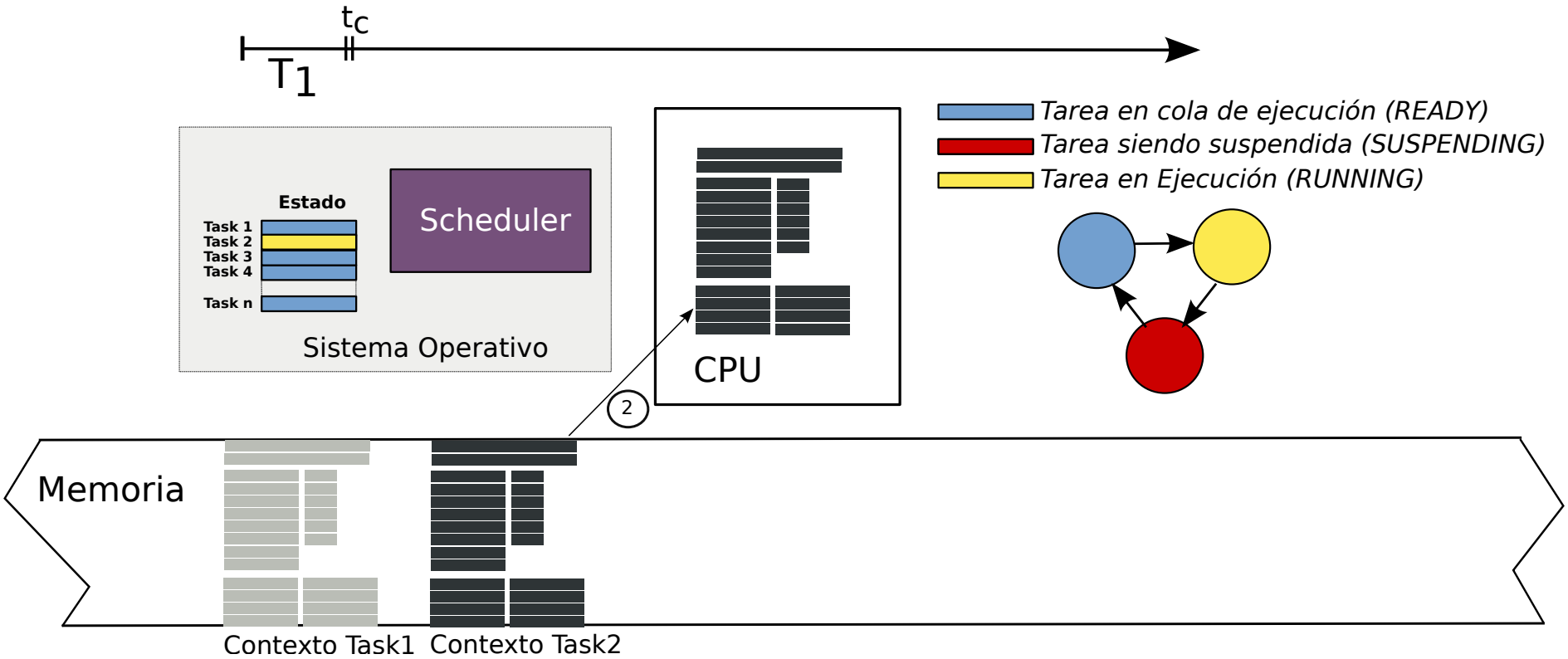
Se carga en el procesador el contexto de la tarea *Task2* (Flujo 2).

Integrando lo visto hasta aquí



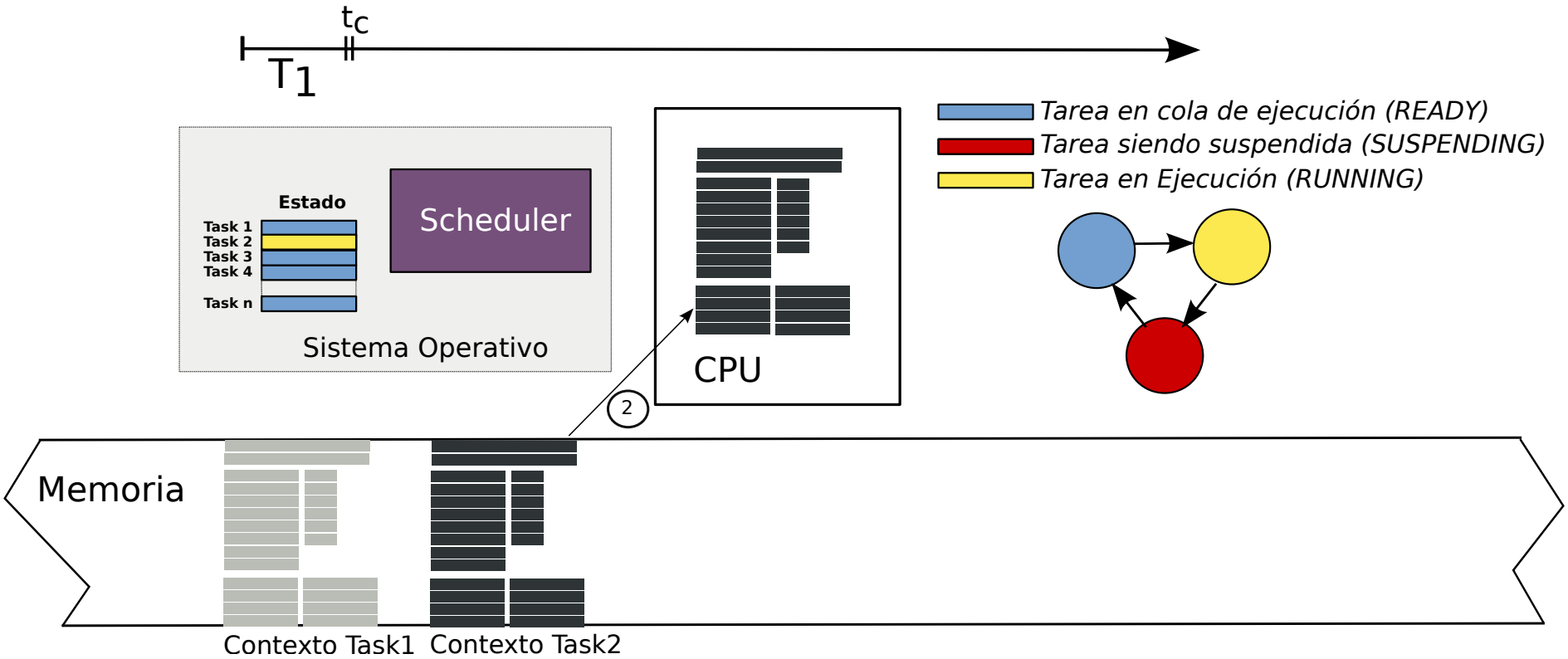
Se carga en el procesador el contexto de la tarea *Task2* (Flujo 2).
En el siguiente ciclo de clock, se reasume la ejecución de *Task2*, en la instrucción siguiente a la última ejecutada antes de ser suspendida.

Integrando lo visto hasta aquí



Se carga en el procesador el contexto de la tarea *Task2* (Flujo 2).
En el siguiente ciclo de clock, se reasume la ejecución de *Task2*, en la instrucción siguiente a la última ejecutada antes de ser suspendida.
Si *Task2* fue recién insertada en la lista de ejecución, entonces el procesador ejecutará su primer instrucción.

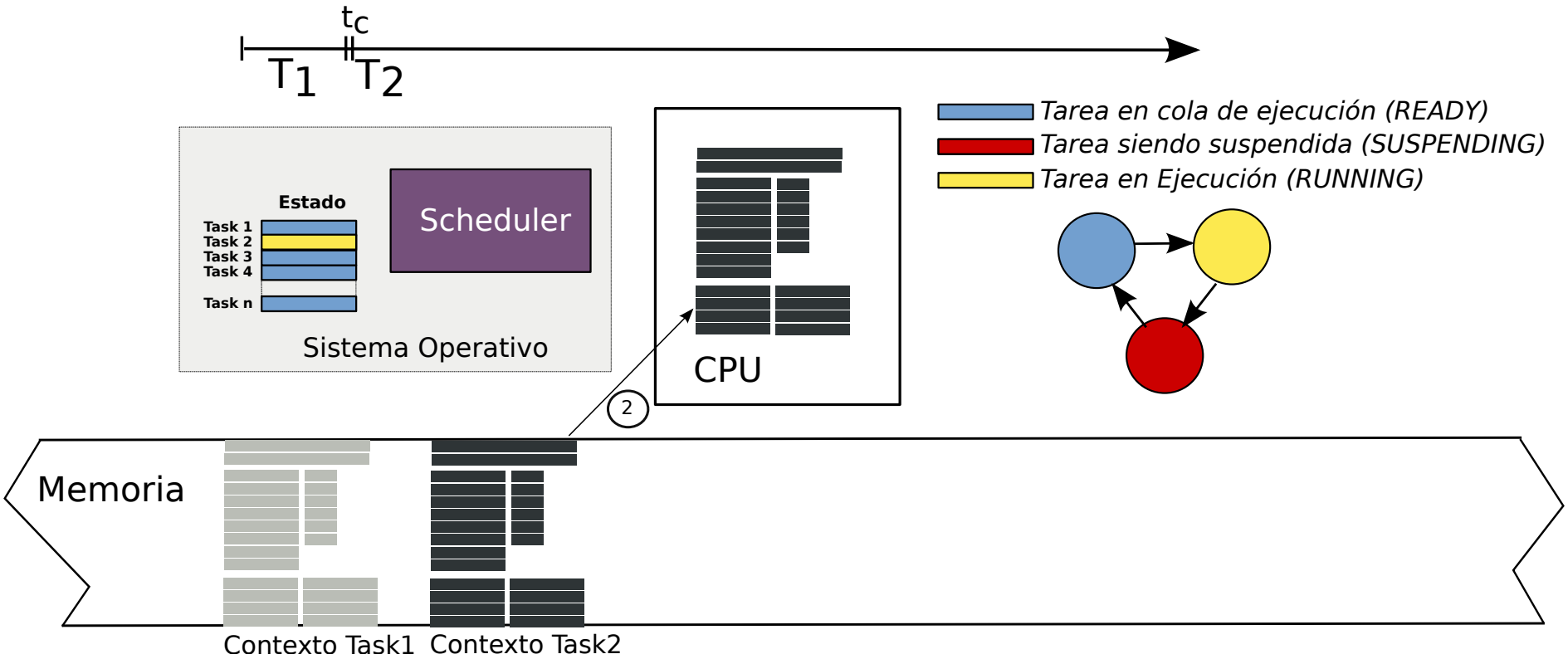
Integrando lo visto hasta aquí



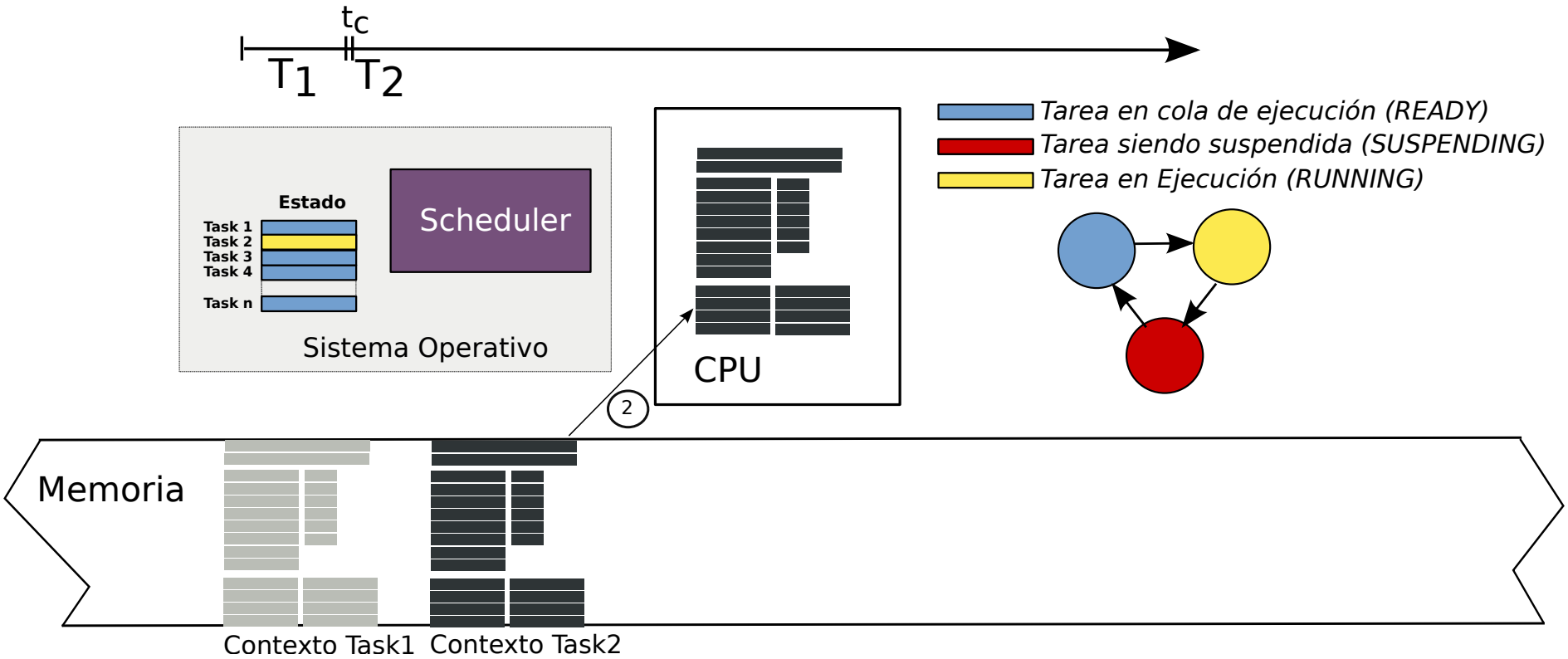
Se carga en el procesador el contexto de la tarea *Task2* (Flujo 2).
En el siguiente ciclo de clock, se reasume la ejecución de *Task2*, en la instrucción siguiente a la última ejecutada antes de ser suspendida.
Si *Task2* fue recién insertada en la lista de ejecución, entonces el procesador ejecutará su primer instrucción.

Llegamos al final del intervalo que denominamos t_c.

Integrando lo visto hasta aquí

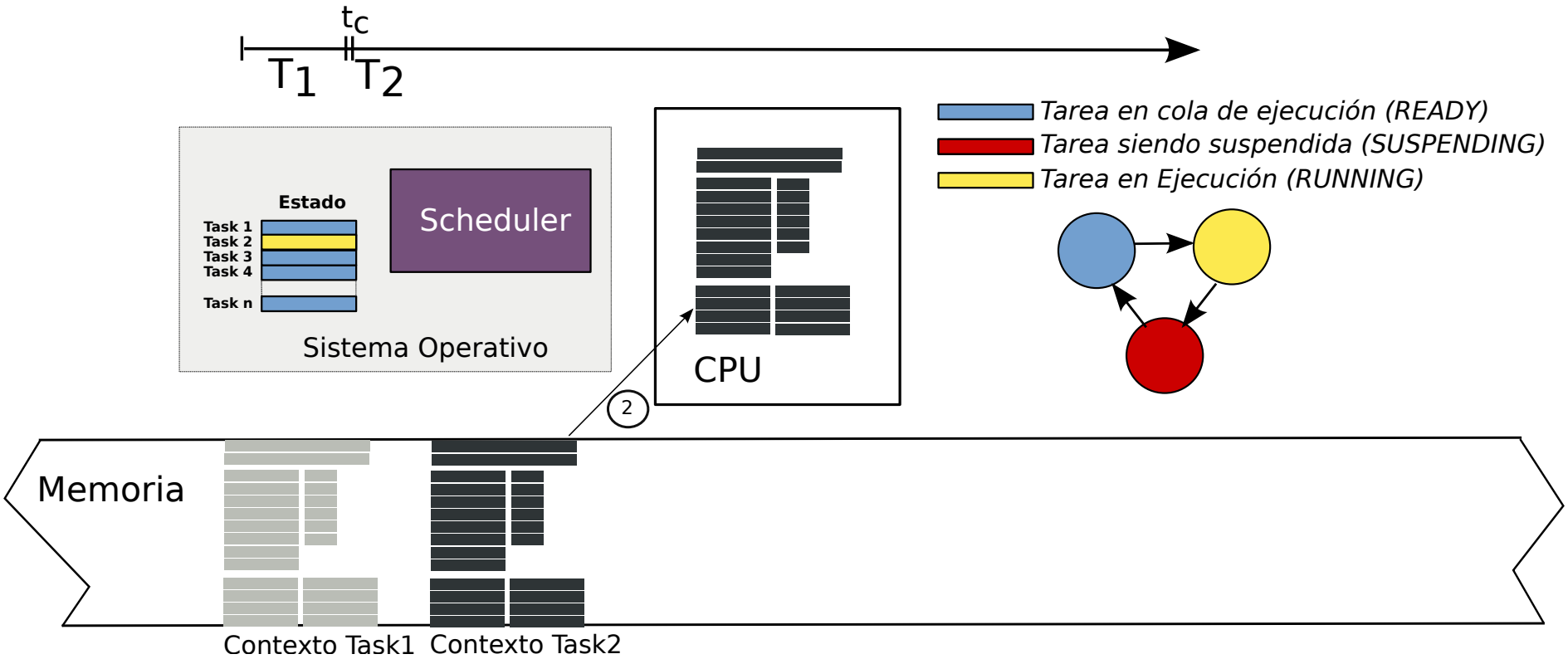


Integrando lo visto hasta aquí



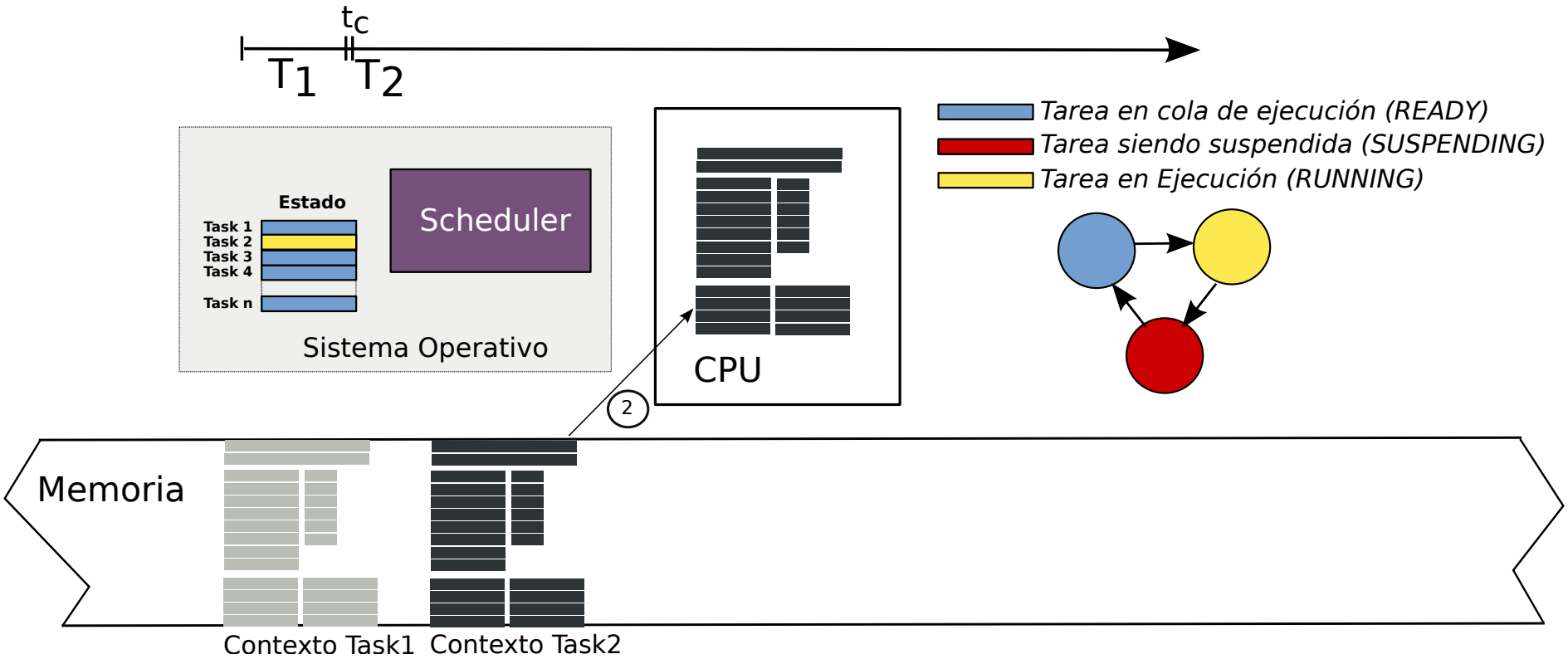
El procesador ejecuta *Task2* durante el tiempo T2.

Integrando lo visto hasta aquí



El procesador ejecuta *Task2* durante el tiempo T2.
Notar que T2 != T1 ya que el Sistema Operativo asigna prioridades diferentes a cada tarea.

Integrando lo visto hasta aquí

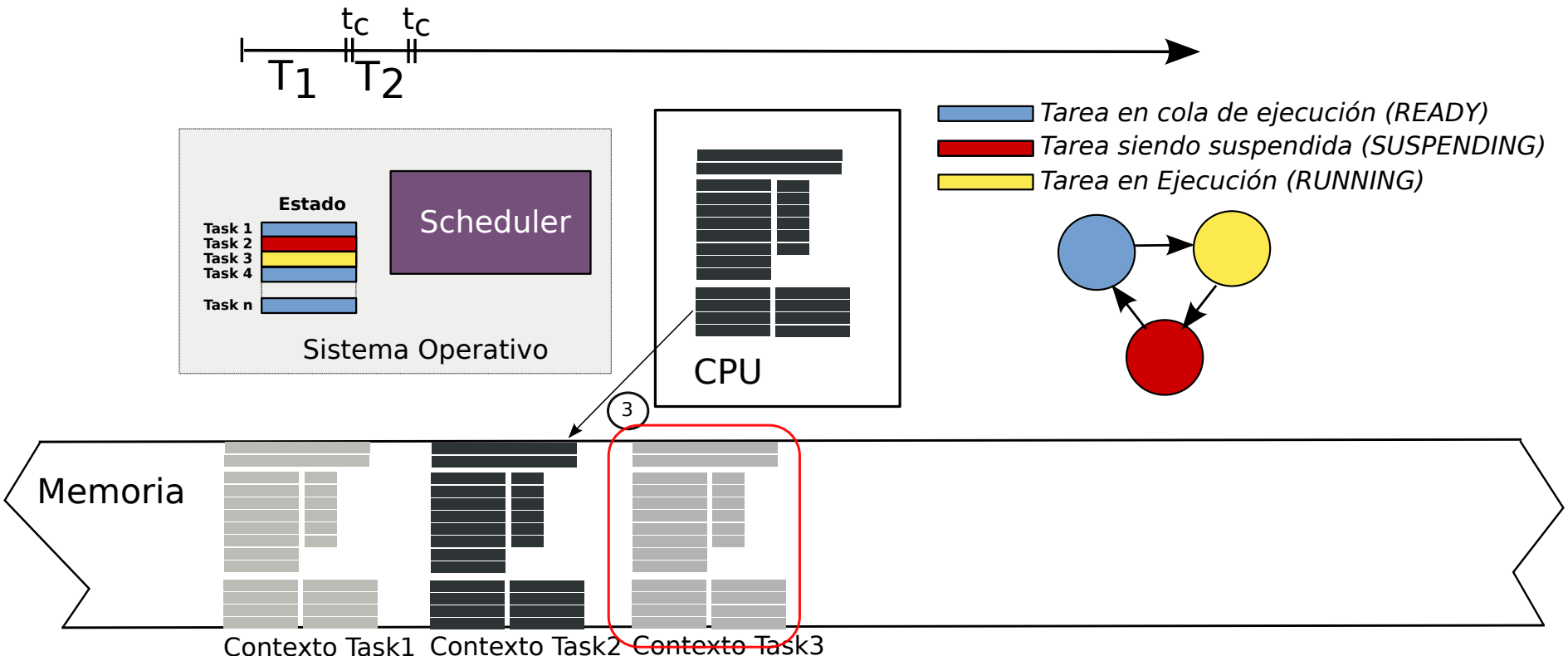


El procesador ejecuta *Task2* durante el tiempo T2.

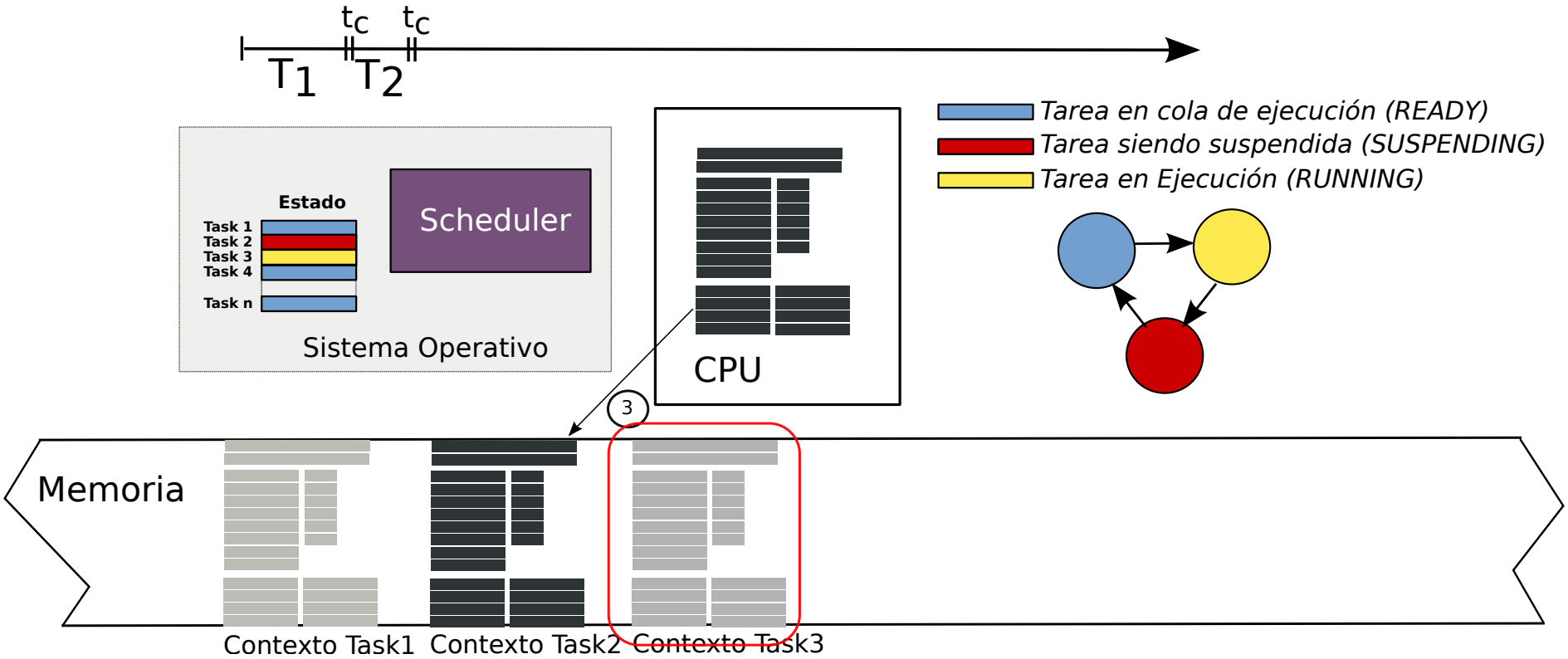
Notar que T2 != T1 ya que el Sistema Operativo asigna prioridades diferentes a cada tarea.

Nuevamente el ***scheduler*** tomará el control cuando expire el tiempo T2.

Integrando lo visto hasta aquí

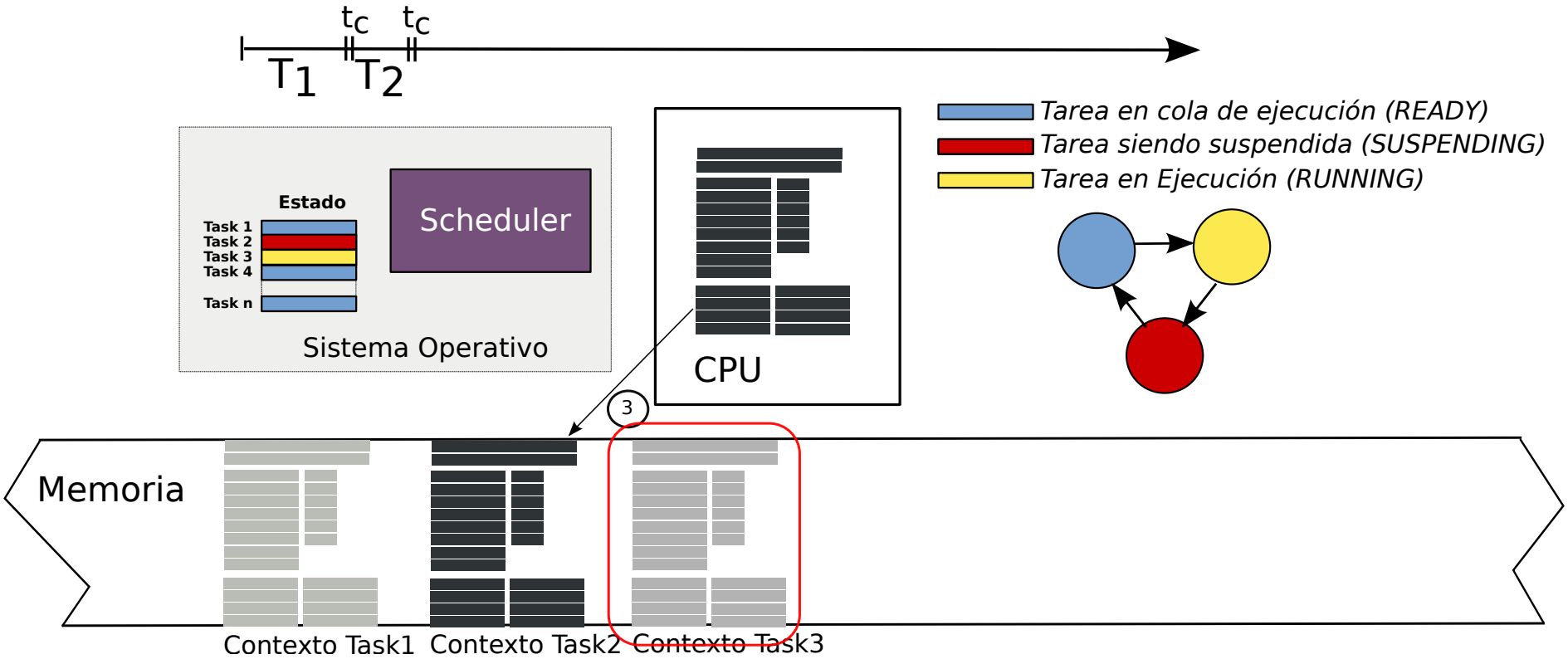


Integrando lo visto hasta aquí



El Sistema Operativo pone *Task2* **SUSPENDING** ya que su tiempo de ejecución expiró.

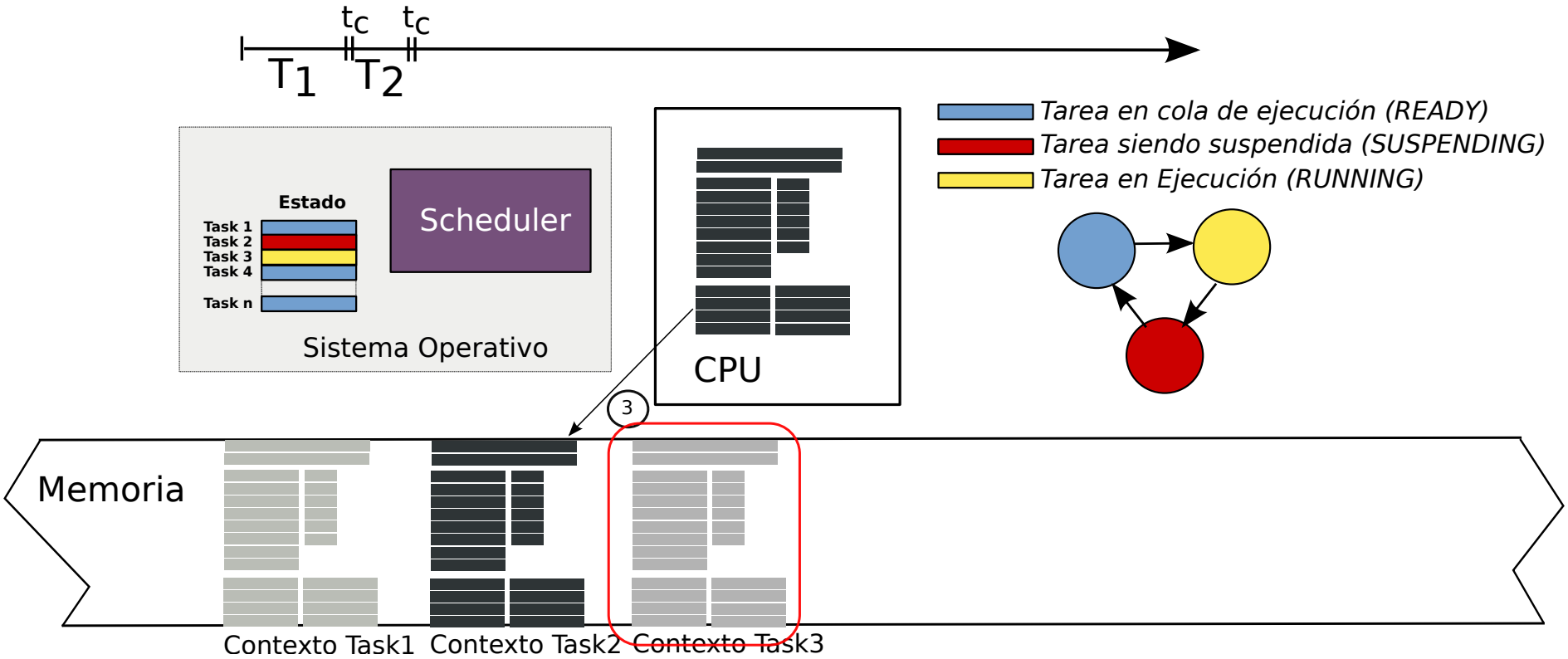
Integrando lo visto hasta aquí



El Sistema Operativo pone *Task2* **SUSPENDING** ya que su tiempo de ejecución expiró.

En el flujo 3, se ve el resguardo del contexto. Por otra parte se busca el identificador de la tarea siguiente en la lista.

Integrando lo visto hasta aquí

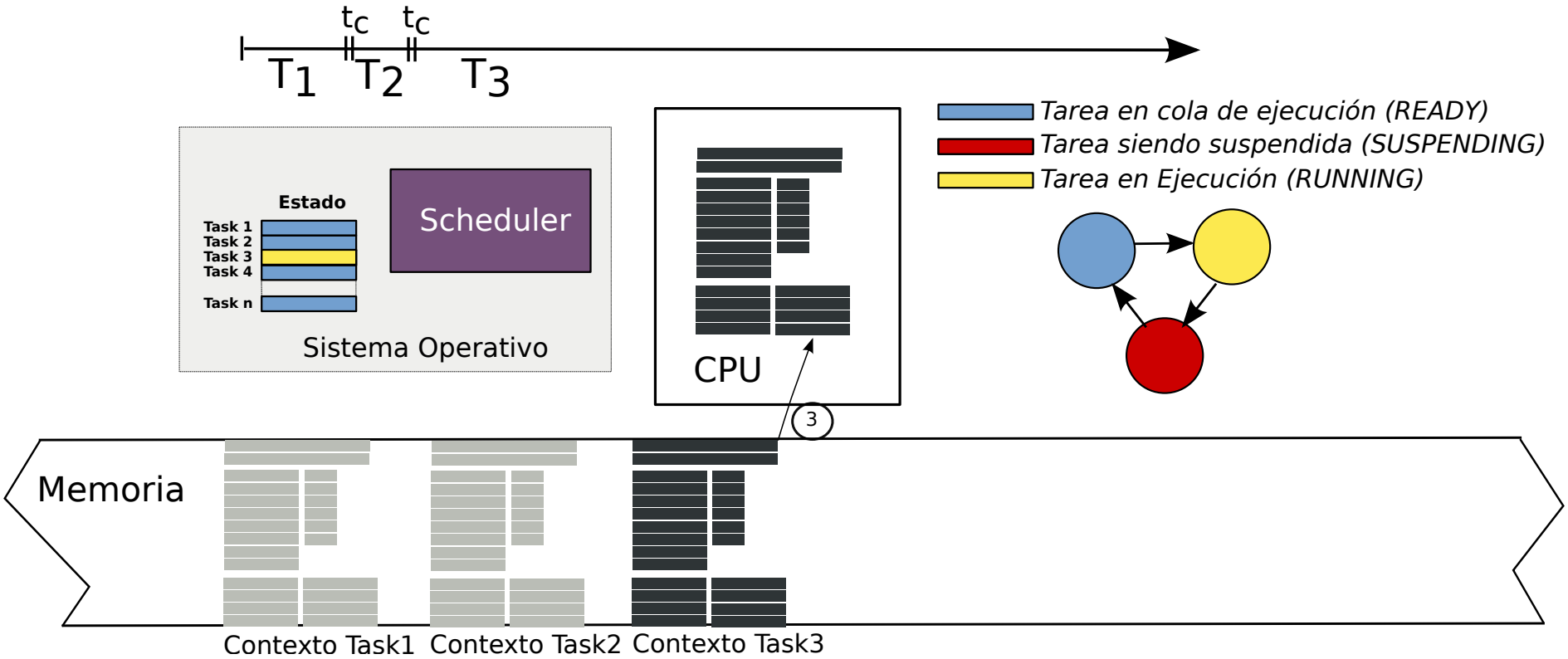


El Sistema Operativo pone *Task2* **SUSPENDING** ya que su tiempo de ejecución expiró.

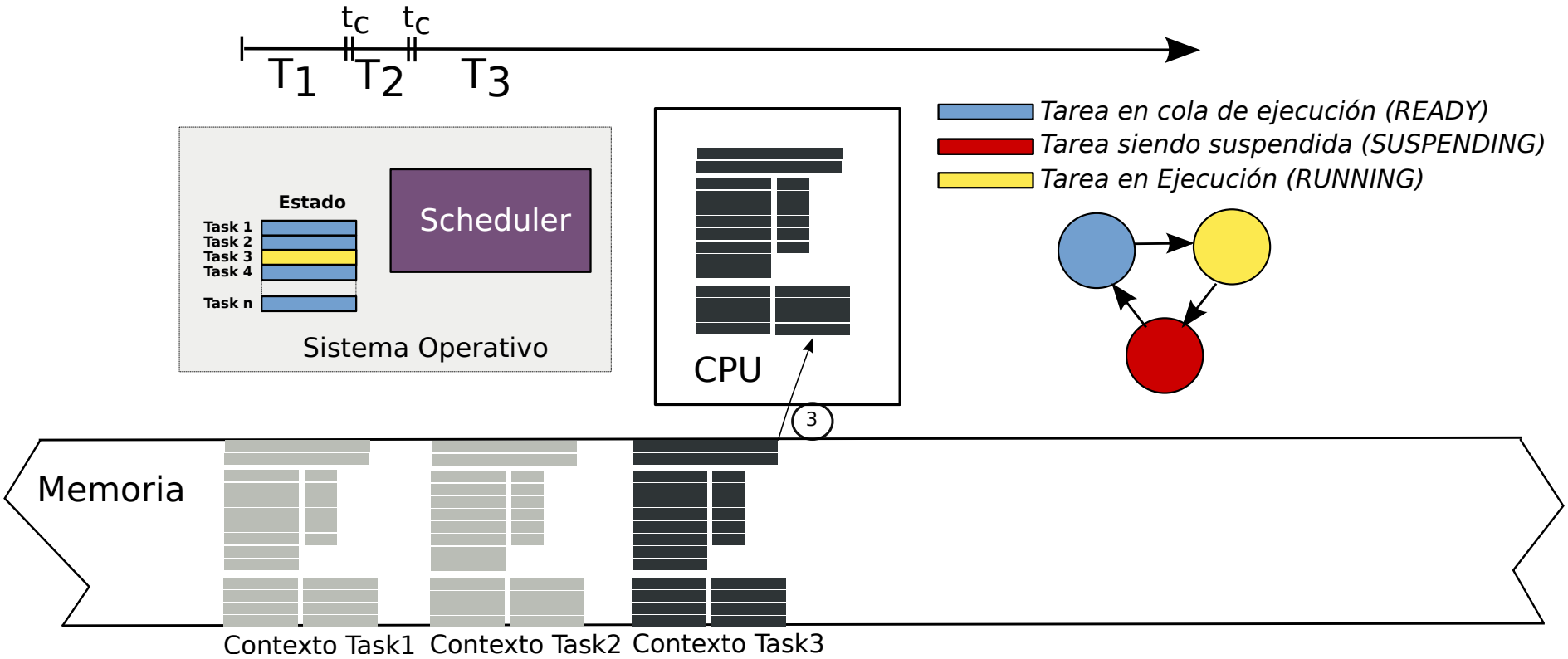
En el flujo 3, se ve el resguardo del contexto. Por otra parte se busca el identificador de la tarea siguiente en la lista.

Una vez ubicado, se identifica el puntero a la dirección de memoria en donde comienza el contexto de la siguiente tarea.

Integrando lo visto hasta aquí

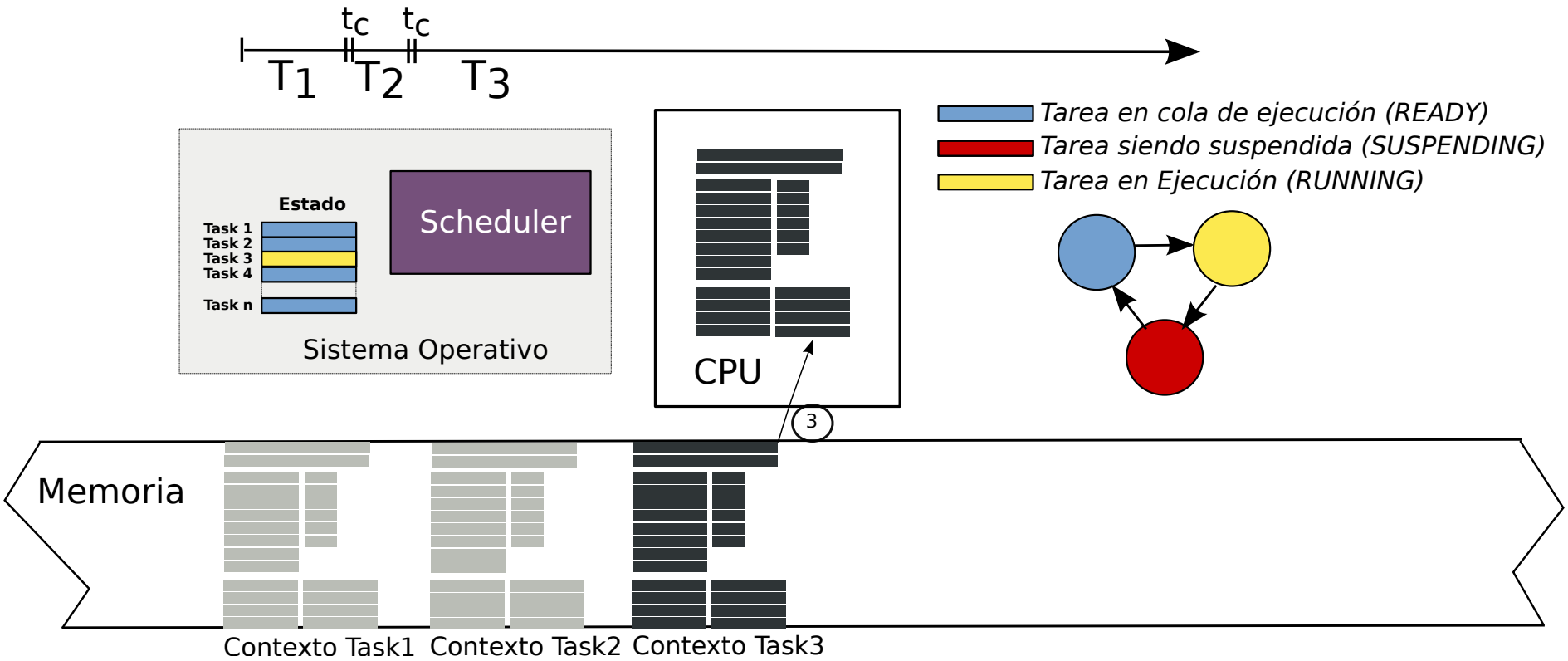


Integrando lo visto hasta aquí



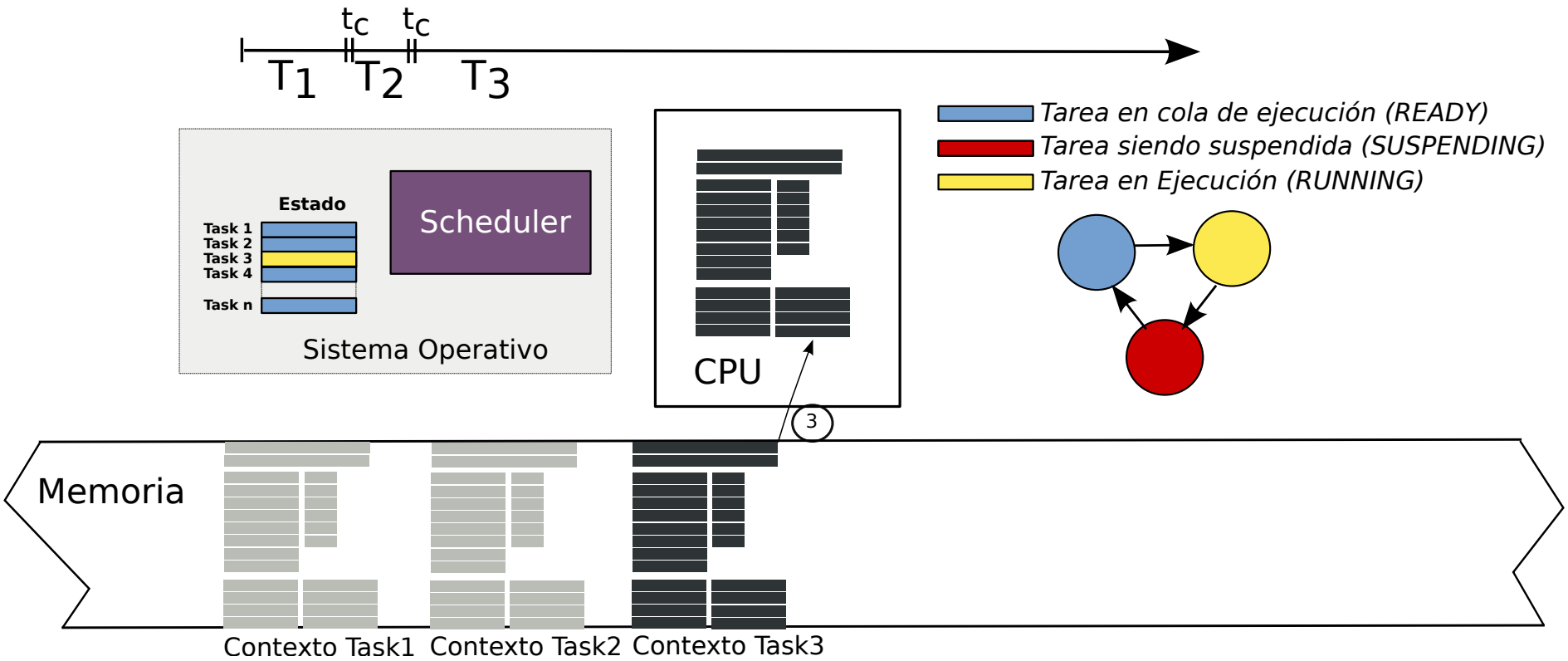
Identificada la tarea siguiente, se carga su contexto en el procesador.

Integrando lo visto hasta aquí



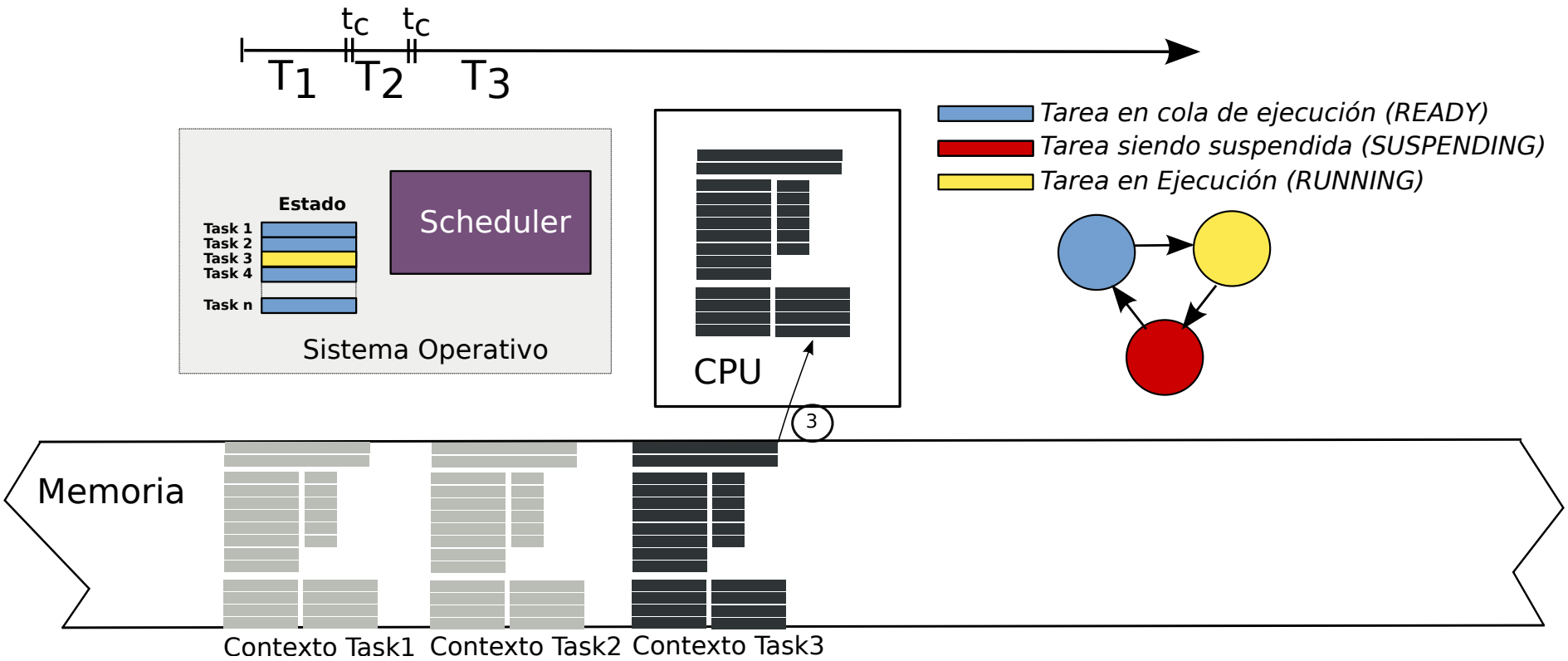
Identificada la tarea siguiente, se carga su contexto en el procesador.
Observar que el tiempo t_c es siempre el mismo independientemente de las tareas.

Integrando lo visto hasta aquí



Identificada la tarea siguiente, se carga su contexto en el procesador.
Observar que el tiempo t_c es siempre el mismo independientemente de las tareas.
Finalizado este proceso en el ciclo de clock siguiente el procesador estará ejecutando la tarea *Task3*.

Integrando lo visto hasta aquí



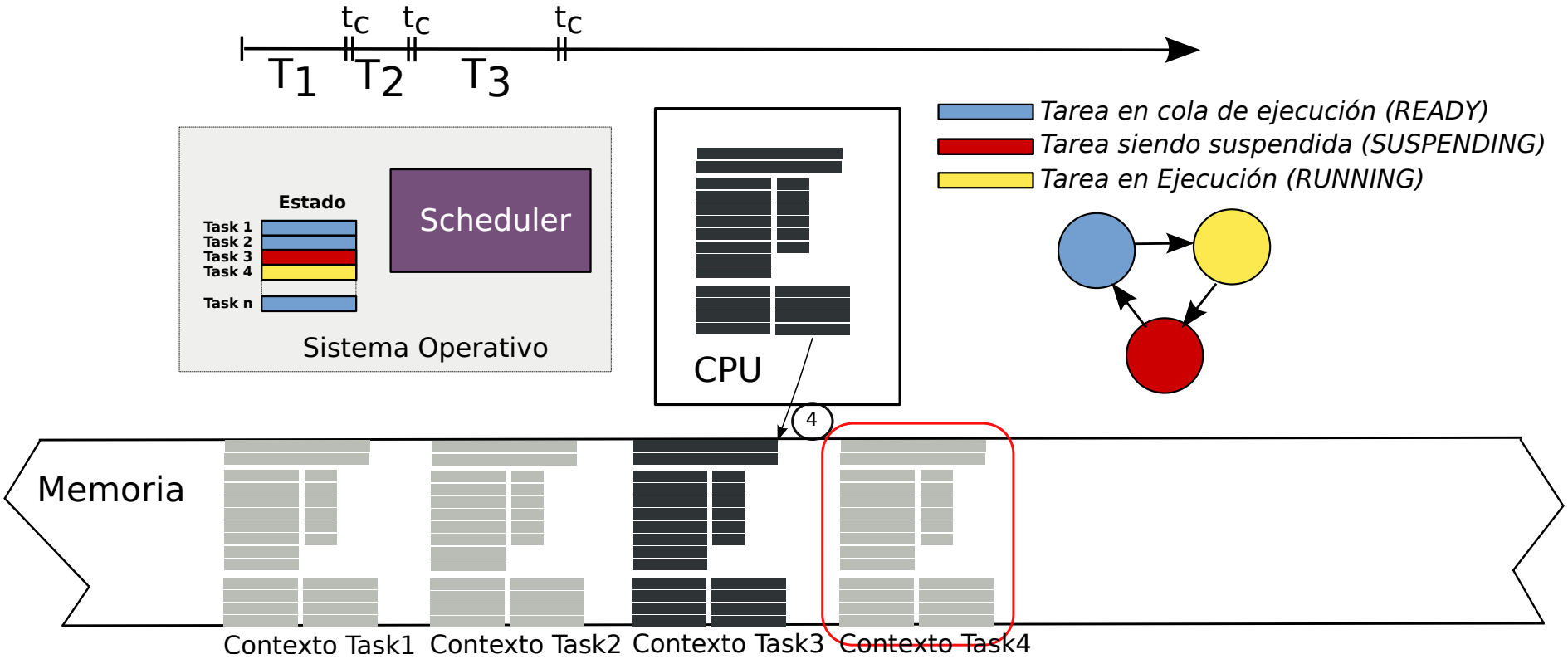
Identificada la tarea siguiente, se carga su contexto en el procesador.

Observar que el tiempo t_c es siempre el mismo independientemente de las tareas.

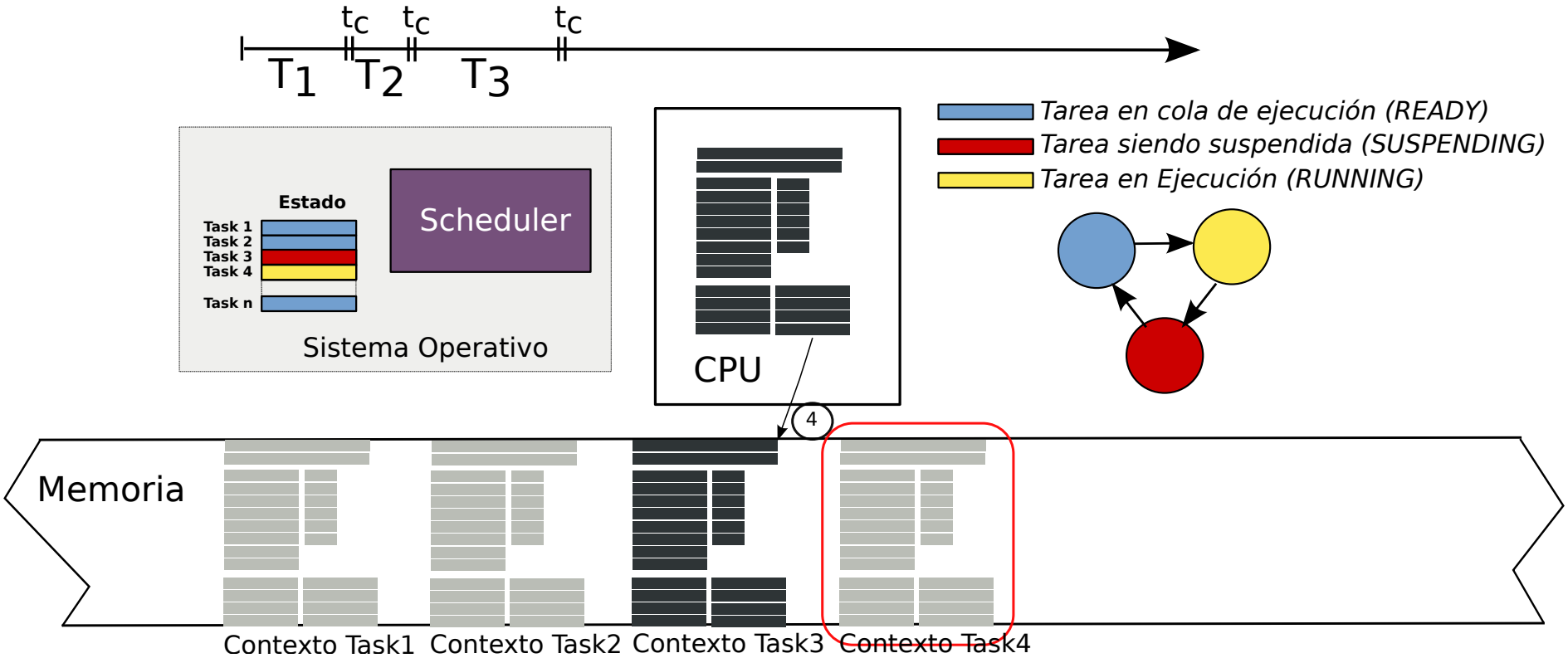
Finalizado este proceso en el ciclo de clock siguiente el procesador estará ejecutando la tarea *Task3*.

Esta continuará durante un tiempo T3.

Integrando lo visto hasta aquí

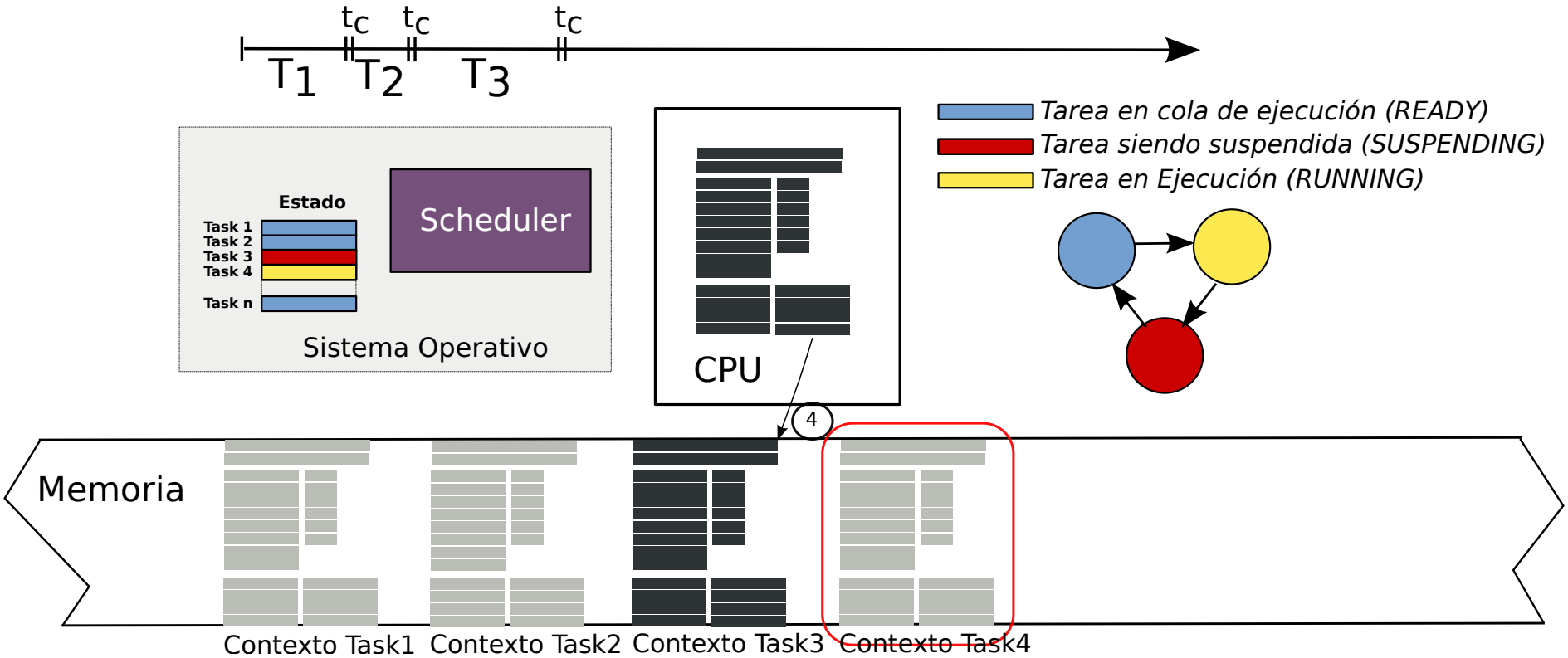


Integrando lo visto hasta aquí



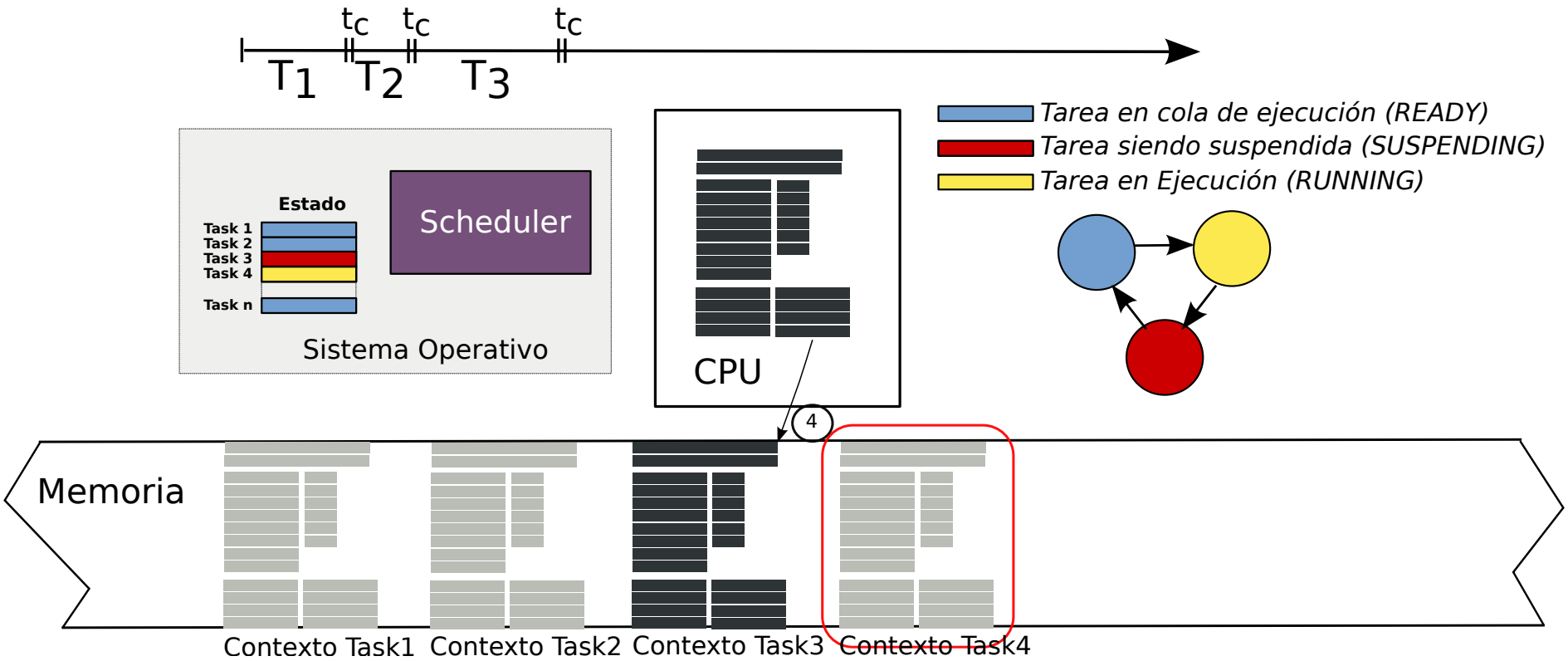
Una vez expirado el tiempo de ejecución de *Task3*, se repite el procedimiento con la siguiente tarea en la lista del ***scheduler***.

Integrando lo visto hasta aquí



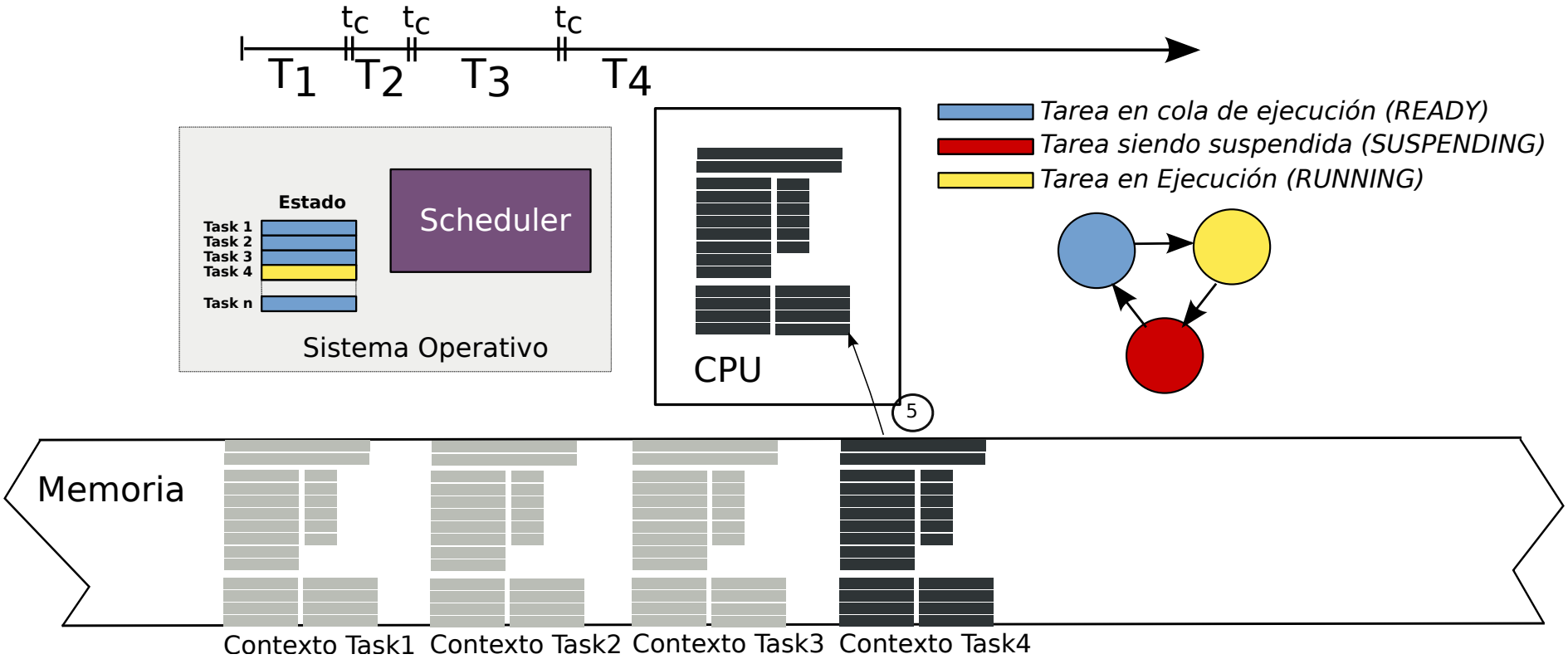
Una vez expirado el tiempo de ejecución de *Task3*, se repite el procedimiento con la siguiente tarea en la lista del ***scheduler***.
En este caso *Task4*.

Integrando lo visto hasta aquí

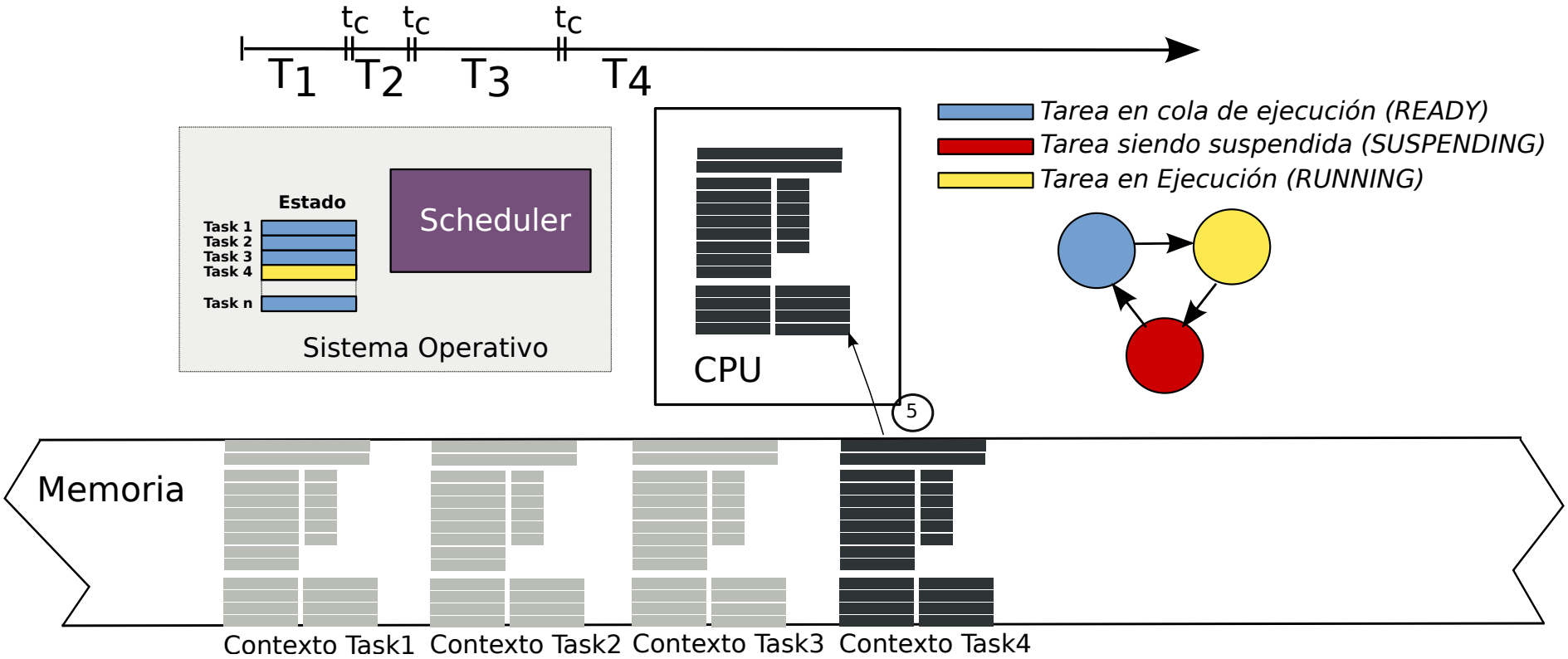


Una vez expirado el tiempo de ejecución de *Task3*, se repite el procedimiento con la siguiente tarea en la lista del ***scheduler***.
En este caso *Task4*.
El flujo 4 muestra el resguardo del contexto de *Task3*.

Integrando lo visto hasta aquí

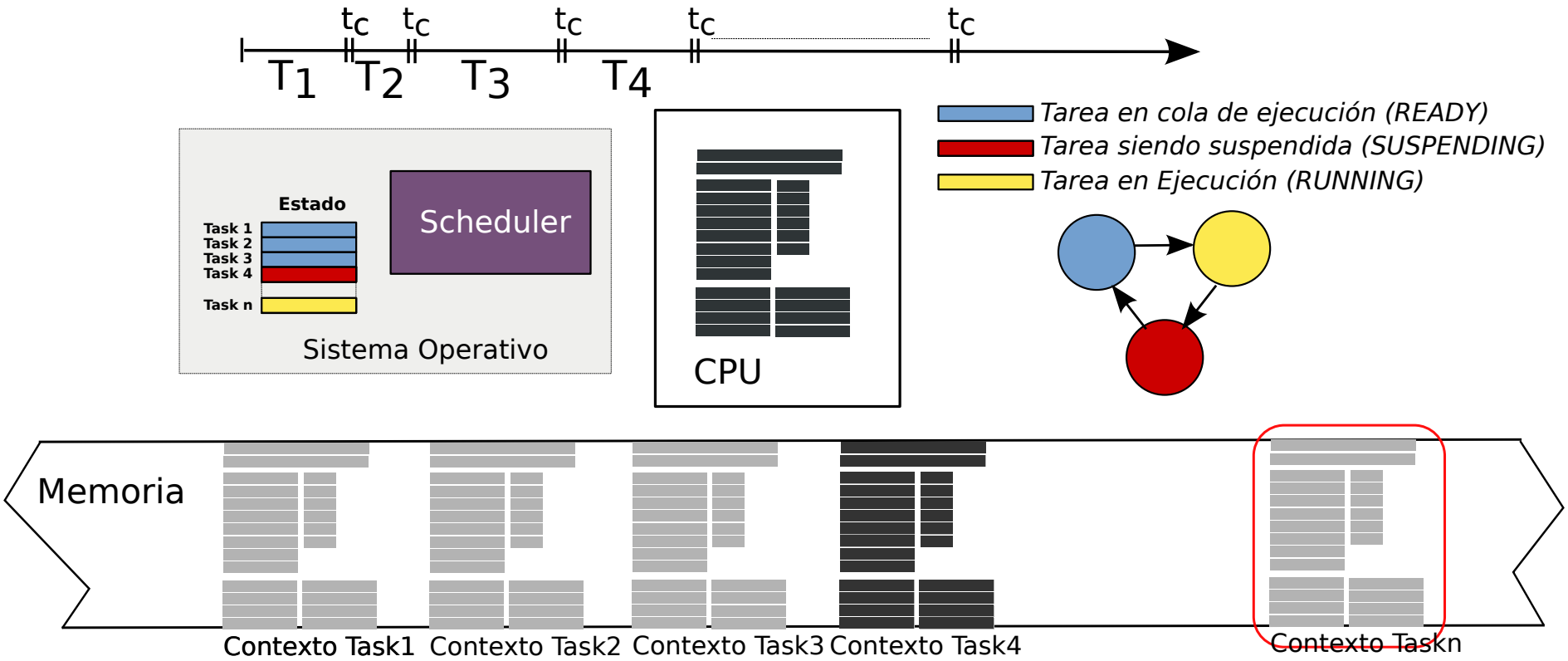


Integrando lo visto hasta aquí

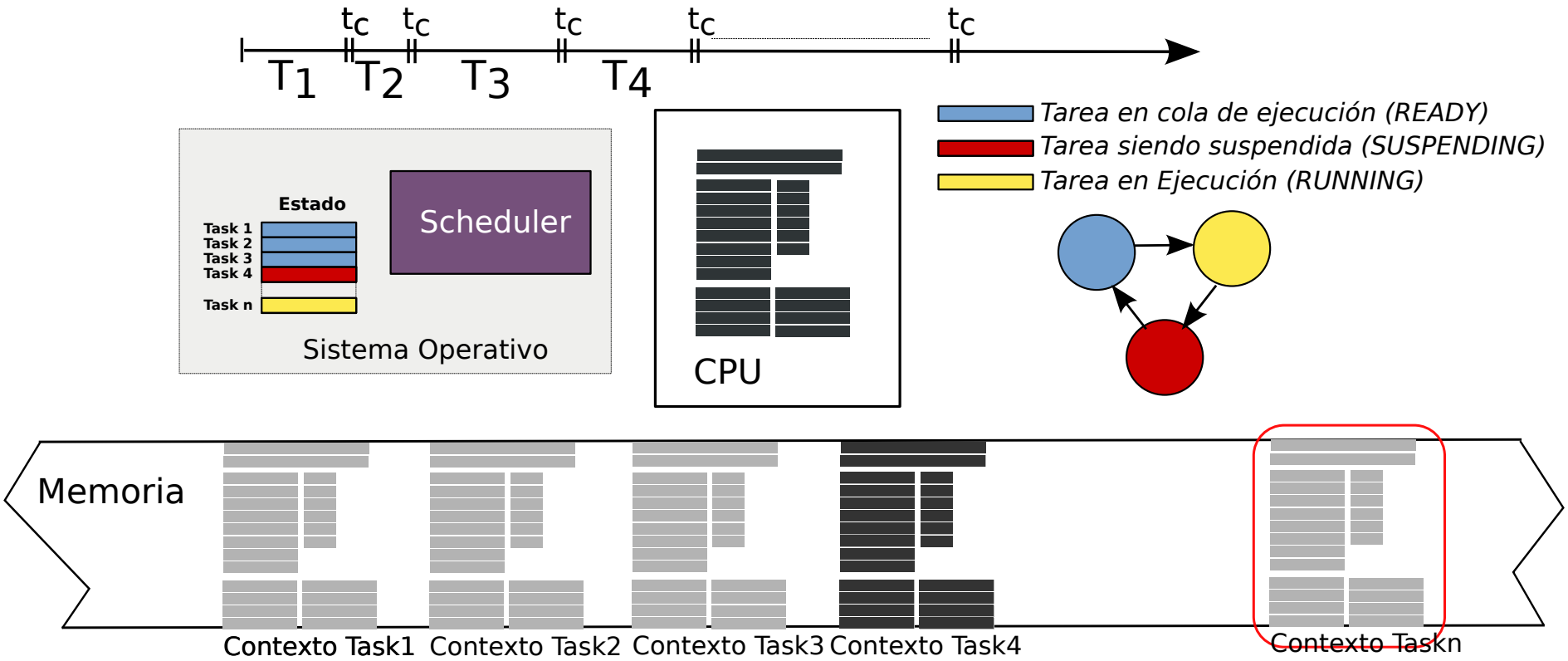


El Flujo 5 muestra como se carga en el procesador el contexto de *Task4*, que ejecutará durante el tiempo T4.

Integrando lo visto hasta aquí

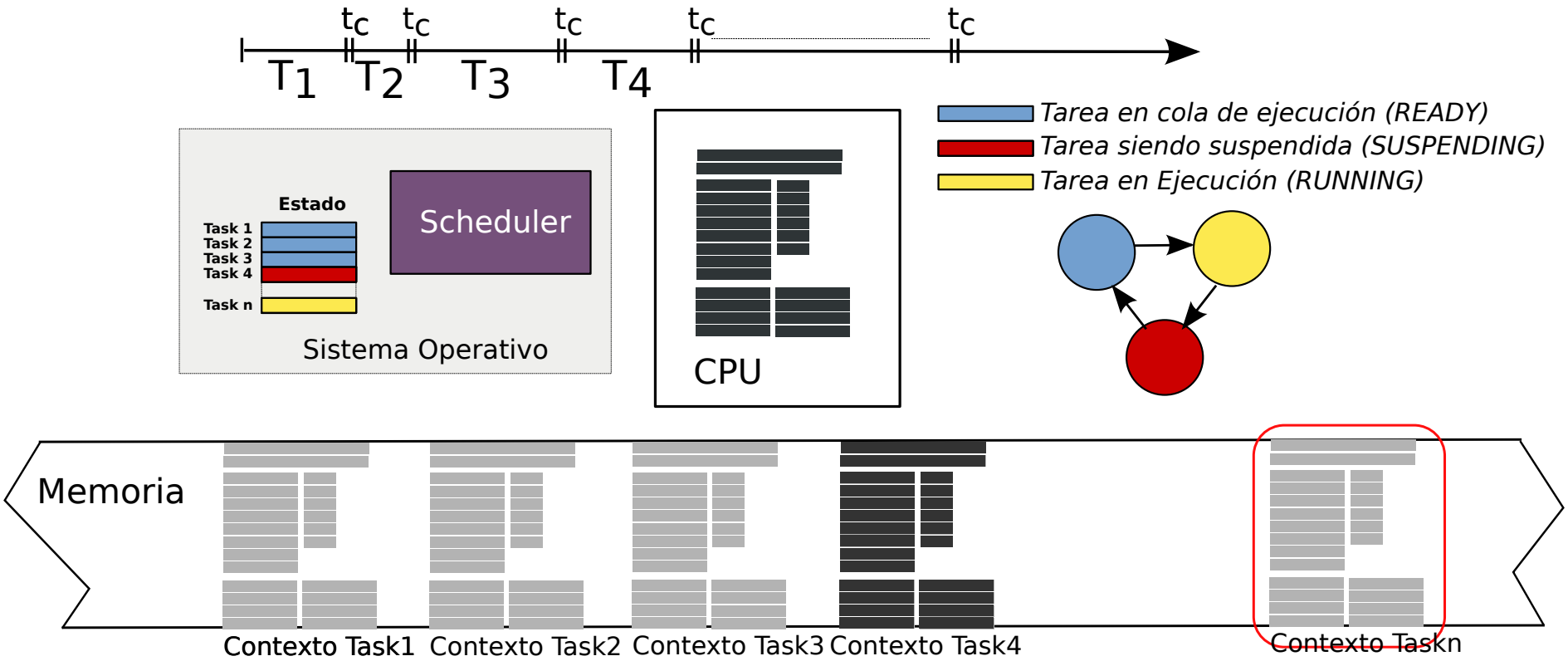


Integrando lo visto hasta aquí



Finalmente llega el turno de la última tarea en la lista.

Integrando lo visto hasta aquí



Finalmente llega el turno de la última tarea en la lista.

Una vez completada esta el ***scheduler*** volverá al inicio de la lista y reasumirá la tarea *Task1*.

Temario

Modelo de Programación de Sistemas

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones
  - ✓ Programación en C, C++, o Assembler en entornos Linux, o Windows.

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones
  - ✓ Programación en C, C++, o Assembler en entornos Linux, o Windows.
  - ✓ Desarrollo de aplicaciones orientadas a objetos en vistosos entornos gráficos.

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones
  - ✓ Programación en C, C++, o Assembler en entornos Linux, o Windows.
  - ✓ Desarrollo de aplicaciones orientadas a objetos en vistosos entornos gráficos.
  - ✓ Desarrollo de aplicaciones en microcontroladores.

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones
  - ✓ Programación en C, C++, o Assembler en entornos Linux, o Windows.
  - ✓ Desarrollo de aplicaciones orientadas a objetos en vistosos entornos gráficos.
  - ✓ Desarrollo de aplicaciones en microcontroladores.
- Trabajamos con un conjunto de recursos que se conocen como “Modelo de programación de Aplicaciones”.

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones
  - ✓ Programación en C, C++, o Assembler en entornos Linux, o Windows.
  - ✓ Desarrollo de aplicaciones orientadas a objetos en vistosos entornos gráficos.
  - ✓ Desarrollo de aplicaciones en microcontroladores.
- Trabajamos con un conjunto de recursos que se conocen como “Modelo de programación de Aplicaciones”.
- Este modelo describe los registros de propósito general, las instrucciones, los tipo de datos, los modos de direccionamiento. . . tal parece que estaría todo dicho.

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones
  - ✓ Programación en C, C++, o Assembler en entornos Linux, o Windows.
  - ✓ Desarrollo de aplicaciones orientadas a objetos en vistosos entornos gráficos.
  - ✓ Desarrollo de aplicaciones en microcontroladores.
- Trabajamos con un conjunto de recursos que se conocen como “Modelo de programación de Aplicaciones”.
- Este modelo describe los registros de propósito general, las instrucciones, los tipo de datos, los modos de direccionamiento. . . tal parece que estaría todo dicho.
- Esto no es todo

¿Acaso hay mas recursos de desarrollo?

Un paso mas hacia el hardware

Si nuestro requerimiento es diseñar un sistema que permita almacenar simultáneamente en memoria varias tareas, administrar su ejecución, evitar que se interfieran, y regular su acceso a los recursos de hardware disponibles, evidentemente estamos entrando en un mundo en el que la visión de Programador de Aplicaciones claramente no es suficiente.

¿Acaso hay mas recursos de desarrollo?

¿Acaso hay mas recursos de desarrollo?

- Entonces se trata de desarrollar un conjunto de tareas independientes entre si que contribuyan a implementar el sistema requerido, y un software que las administre.

¿Acaso hay mas recursos de desarrollo?

- Entonces se trata de desarrollar un conjunto de tareas independientes entre si que contribuyan a implementar el sistema requerido, y un software que las administre.
- Necesitamos contar con un SoC cuya CPU tenga capacidades para realizar el propósito planteado.

¿Acaso hay mas recursos de desarrollo?

- Entonces se trata de desarrollar un conjunto de tareas independientes entre si que contribuyan a implementar el sistema requerido, y un software que las administre.
- Necesitamos contar con un SoC cuya CPU tenga capacidades para realizar el propósito planteado.
- Si nos las tiene, se puede desarrollar un sistema como el que nos estamos planteando, de todos modos. Pero no esperemos milagros en cuanto a rendimiento, ni le exijamos demasiado al modelo implementado.

¿Acaso hay mas recursos de desarrollo?

- Entonces se trata de desarrollar un conjunto de tareas independientes entre si que contribuyan a implementar el sistema requerido, y un software que las administre.
- Necesitamos contar con un SoC cuya CPU tenga capacidades para realizar el propósito planteado.
- Si nos las tiene, se puede desarrollar un sistema como el que nos estamos planteando, de todos modos. Pero no esperemos milagros en cuanto a rendimiento, ni le exijamos demasiado al modelo implementado.
- Primer paso: tener claro que CPU necesitamos para el requerimiento de nuestro sistema

Modelo de Programación de Sistemas

Modelo de Programación de Sistemas

- El modelo de programación de Aplicaciones de una CPU tiene una visión restringida del sistema en su conjunto.

Modelo de Programación de Sistemas

- El modelo de programación de Aplicaciones de una CPU tiene una visión restringida del sistema en su conjunto.
- El modelo de programación de Sistemas, incluye un conjunto de recursos de hardware que amplía la capacidad de la CPU de resolver funciones que le permitan a un Sistema Operativo proveer los servicios y el entorno de programación que “ven ”las aplicaciones, de manera mucho mas veloz y eficiente.

Modelo de Programación de Sistemas

- El modelo de programación de Aplicaciones de una CPU tiene una visión restringida del sistema en su conjunto.
- El modelo de programación de Sistemas, incluye un conjunto de recursos de hardware que amplía la capacidad de la CPU de resolver funciones que le permitan a un Sistema Operativo proveer los servicios y el entorno de programación que “ven ”las aplicaciones, de manera mucho mas veloz y eficiente.
- De hecho, el manejo de las excepciones y de las interrupciones de hardware que hemos visto anteriormente es claramente el terreno del Sistema Operativo.

Modelo de Programación de Sistemas

- El modelo de programación de Aplicaciones de una CPU tiene una visión restringida del sistema en su conjunto.
- El modelo de programación de Sistemas, incluye un conjunto de recursos de hardware que amplía la capacidad de la CPU de resolver funciones que le permitan a un Sistema Operativo proveer los servicios y el entorno de programación que “ven ”las aplicaciones, de manera mucho mas veloz y eficiente.
- De hecho, el manejo de las excepciones y de las interrupciones de hardware que hemos visto anteriormente es claramente el terreno del Sistema Operativo.
- Y hay muchas mas funciones para proveer a un Sistema Operativo desde el hardware que no son necesarias para las aplicaciones.

Scheduling de Tareas

Scheduling de Tareas

- Cualquier Sistema Operativo posee un módulo llamado **scheduler**, que le permite administrar la ejecución de tareas / procesos.

Scheduling de Tareas

- Cualquier Sistema Operativo posee un módulo llamado **scheduler**, que le permite administrar la ejecución de tareas / procesos.
- Podemos considerar al **scheduler** dividido en dos capas:

Scheduling de Tareas

- Cualquier Sistema Operativo posee un módulo llamado **scheduler**, que le permite administrar la ejecución de tareas / procesos.
- Podemos considerar al **scheduler** dividido en dos capas:
  - ✓ Una de alto nivel en la que se implementan una o mas *políticas* para manejar el cambio de tareas, y que se traducen en diferentes *algoritmos de scheduling*.

Scheduling de Tareas

- Cualquier Sistema Operativo posee un módulo llamado **scheduler**, que le permite administrar la ejecución de tareas / procesos.
- Podemos considerar al **scheduler** dividido en dos capas:
  - ✓ Una de alto nivel en la que se implementan una o mas *políticas* para manejar el cambio de tareas, y que se traducen en diferentes *algoritmos de scheduling*.
  - ✓ Una de bajo nivel en la que se realiza el *context switch*.

Scheduling de Tareas

- Cualquier Sistema Operativo posee un módulo llamado **scheduler**, que le permite administrar la ejecución de tareas / procesos.
- Podemos considerar al **scheduler** dividido en dos capas:
  - ✓ Una de alto nivel en la que se implementan una o mas *políticas* para manejar el cambio de tareas, y que se traducen en diferentes *algoritmos de scheduling*.
  - ✓ Una de bajo nivel en la que se realiza el *context switch*.
- En este punto nos vamos a concentrar en el módulo de *context switch*, dejando para mas adelante los algoritmos de scheduling.

Scheduling de Tareas

- Cualquier Sistema Operativo posee un módulo llamado **scheduler**, que le permite administrar la ejecución de tareas / procesos.
- Podemos considerar al **scheduler** dividido en dos capas:
  - ✓ Una de alto nivel en la que se implementan una o mas *políticas* para manejar el cambio de tareas, y que se traducen en diferentes *algoritmos de scheduling*.
  - ✓ Una de bajo nivel en la que se realiza el *context switch*.
- En este punto nos vamos a concentrar en el módulo de *context switch*, dejando para mas adelante los algoritmos de scheduling.
- El *context switch* como veremos es físicamente dependiente del procesador y se escribe en assembler.

Capa de bajo nivel: Context Switch

Capa de bajo nivel: Context Switch

- Un Context Switch genérico debe llevar a cabo las siguientes acciones:

Capa de bajo nivel: Context Switch

- Un Context Switch genérico debe llevar a cabo las siguientes acciones:

1. Resguardar los registros core y cualquier otro del modelo de programador de aplicaciones en un área de memoria accesible solo para el kernel.

Capa de bajo nivel: Context Switch

- Un Context Switch genérico debe llevar a cabo las siguientes acciones:

1. Resguardar los registros core y cualquier otro del modelo de programador de aplicaciones en un área de memoria accesible solo para el kernel.
2. En general el contexto forma parte de una estructura mas amplia llamada genéricamente **TCB** (por **T**ask **C**ontrol **B**lock) o en la jerga de los sistemas operativos de propósito general PCB (**P**rocess en lugar de **T**ask).

Capa de bajo nivel: Context Switch

- Un Context Switch genérico debe llevar a cabo las siguientes acciones:

1. Resguardar los registros core y cualquier otro del modelo de programador de aplicaciones en un área de memoria accesible solo para el kernel.
2. En general el contexto forma parte de una estructura mas amplia llamada genéricamente **TCB** (por **T**ask **C**ontrol **B**lock) o en la jerga de los sistemas operativos de propósito general PCB (**P**rocess en lugar de **T**ask).
3. Determinar la ubicación del TCB de la siguiente tarea en la lista de tareas en ejecución.

Capa de bajo nivel: Context Switch

- Un Context Switch genérico debe llevar a cabo las siguientes acciones:

1. Resguardar los registros core y cualquier otro del modelo de programador de aplicaciones en un área de memoria accesible solo para el kernel.
2. En general el contexto forma parte de una estructura mas amplia llamada genéricamente **TCB** (por **T**ask **C**ontrol **B**lock) o en la jerga de los sistemas operativos de propósito general PCB (**P**rocess en lugar de **T**ask).
3. Determinar la ubicación del TCB de la siguiente tarea en la lista de tareas en ejecución.
4. Extraer el contexto de la nueva tarea y copiar registro a registro en la CPU.

Capa de alto nivel: Políticas de Scheduling

Capa de alto nivel: Políticas de Scheduling

- La capa de alto nivel llama al context switch cuando necesita conmutar tareas, en base a criterios definidos en su algoritmo.

Capa de alto nivel: Políticas de Scheduling

- La capa de alto nivel llama al context switch cuando necesita conmutar tareas, en base a criterios definidos en su algoritmo.
- Antes de invocar al context switch evalúa una serie de variables:

Capa de alto nivel: Políticas de Scheduling

- La capa de alto nivel llama al context switch cuando necesita conmutar tareas, en base a criterios definidos en su algoritmo.
- Antes de invocar al context switch evalúa una serie de variables:
  1. prioridad,

Capa de alto nivel: Políticas de Scheduling

- La capa de alto nivel llama al context switch cuando necesita conmutar tareas, en base a criterios definidos en su algoritmo.
- Antes de invocar al context switch evalúa una serie de variables:
  1. prioridad,
  2. Preemption,

Capa de alto nivel: Políticas de Scheduling

- La capa de alto nivel llama al context switch cuando necesita conmutar tareas, en base a criterios definidos en su algoritmo.
- Antes de invocar al context switch evalúa una serie de variables:
  1. prioridad,
  2. Preemption,
  3. Atomicidad de operaciones,

Capa de alto nivel: Políticas de Scheduling

- La capa de alto nivel llama al context switch cuando necesita conmutar tareas, en base a criterios definidos en su algoritmo.
- Antes de invocar al context switch evalúa una serie de variables:
  1. prioridad,
  2. Preemption,
  3. Atomicidad de operaciones,
  4. concurrencia,

Capa de alto nivel: Políticas de Scheduling

- La capa de alto nivel llama al context switch cuando necesita conmutar tareas, en base a criterios definidos en su algoritmo.
- Antes de invocar al context switch evalúa una serie de variables:
  1. prioridad,
  2. Preemption,
  3. Atomicidad de operaciones,
  4. concurrencia,
  5. limitaciones en tiempo (latency),

Capa de alto nivel: Políticas de Scheduling

- La capa de alto nivel llama al context switch cuando necesita conmutar tareas, en base a criterios definidos en su algoritmo.
- Antes de invocar al context switch evalúa una serie de variables:
  1. prioridad,
  2. Preemption,
  3. Atomicidad de operaciones,
  4. concurrencia,
  5. limitaciones en tiempo (latency),
  6. paralelismo (en caso de que el sistema cuente con mas de una CPU), entre otros.

Capa de alto nivel: Políticas de Scheduling

- La capa de alto nivel llama al context switch cuando necesita conmutar tareas, en base a criterios definidos en su algoritmo.
- Antes de invocar al context switch evalúa una serie de variables:
  1. prioridad,
  2. Preemption,
  3. Atomicidad de operaciones,
  4. concurrencia,
  5. limitaciones en tiempo (latency),
  6. paralelismo (en caso de que el sistema cuente con mas de una CPU), entre otros.
- Por lo tanto un **scheduler** resulta un código que puede oscilar entre algo muy sencillo, casi trivial, hasta algoritmos muy complejos que puedan considerar los aspectos señalados en el ítem anterior.

Conclusiones

Approach bootom-up

Intentaremos abordar el diseño de un scheduler mas o menos sencillo comenzando por la parte basal: El Context Switch.
Esta es la parte dependiente del hardware. La capa superior interfaz consistente mediante puede reutilizarse, para lo cual es conveniente escribirla en C, para segurar su portabilidad.

Temario

Como se resuelve un context switch por Hardware

Como se resuelve un context switch por Hardware

- En la enorme mayoría de los microprocesadores (y mas aun en los micro controladores) no hay recursos específicos para ayudar a realizar un context switch (también llamado task switch)

Como se resuelve un context switch por Hardware

- En la enorme mayoría de los microprocesadores (y mas aun en los micro controladores) no hay recursos específicos para ayudar a realizar un context switch (también llamado task switch)
- Es decir, que no hay instrucciones específicas ni unidades de hardware que nos ayuden en el context switch.

Como se resuelve un context switch por Hardware

- En la enorme mayoría de los microprocesadores (y mas aun en los micro controladores) no hay recursos específicos para ayudar a realizar un context switch (también llamado task switch)
- Es decir, que no hay instrucciones específicas ni unidades de hardware que nos ayuden en el context switch.
- Resumiendo. ¿Que nos queda?:

Como se resuelve un context switch por Hardware

- En la enorme mayoría de los microprocesadores (y mas aun en los micro controladores) no hay recursos específicos para ayudar a realizar un context switch (también llamado task switch)
- Es decir, que no hay instrucciones específicas ni unidades de hardware que nos ayuden en el context switch.
- Resumiendo. ¿Que nos queda?:

Como

Como

- La primer pregunta a responder es ¿Como activamos el context switch?

Como

- La primer pregunta a responder es ¿Como activamos el context switch?
- En principio sabemos que activar el context switch es una decisión del **scheduler** en función de una serie de métricas, y criterios establecidos como políticas en su algoritmo.

Como

- La primer pregunta a responder es ¿Como activamos el context switch?
- En principio sabemos que activar el context switch es una decisión del **scheduler** en función de una serie de métricas, y criterios establecidos como políticas en su algoritmo.
- El tema es que estas métricas como por ejemplo el tiempo de ejecución de una tarea van cambiando en el tiempo, por lo tanto deben ser monitoreadas de manera regular

Como

- La primer pregunta a responder es ¿Como activamos el context switch?
- En principio sabemos que activar el context switch es una decisión del **scheduler** en función de una serie de métricas, y criterios establecidos como políticas en su algoritmo.
- El tema es que estas métricas como por ejemplo el tiempo de ejecución de una tarea van cambiando en el tiempo, por lo tanto deben ser monitoreadas de manera regular
- Por tal motivo debe implementarse un método para que se active de manera periódica para evaluar las métricas que se hayan definido.

Como

- La primer pregunta a responder es ¿Como activamos el context switch?
- En principio sabemos que activar el context switch es una decisión del **scheduler** en función de una serie de métricas, y criterios establecidos como políticas en su algoritmo.
- El tema es que estas métricas como por ejemplo el tiempo de ejecución de una tarea van cambiando en el tiempo, por lo tanto deben ser monitoreadas de manera regular
- Por tal motivo debe implementarse un método para que se active de manera periódica para evaluar las métricas que se hayan definido.
- ¿Como?: Respuesta: Usando un timer.

Donde

Donde

- La siguiente pregunta es ¿Donde ubicamos el código del **scheduler** entonces?

Donde

- La siguiente pregunta es ¿Donde ubicamos el código del **scheduler** entonces?
- La respuesta es trivial: En el handler de Interrupción del Timer seleccionado.

Donde

- La siguiente pregunta es ¿Donde ubicamos el código del **scheduler** entonces?
- La respuesta es trivial: En el handler de Interrupción del Timer seleccionado.
- Casos prácticos:

Donde

- La siguiente pregunta es ¿Donde ubicamos el código del **scheduler** entonces?
- La respuesta es trivial: En el handler de Interrupción del Timer seleccionado.
- Casos prácticos:
  1. SYSTICK en los Cortex-M (Recordar que el SYSTICK forma parte de un core CORTEX-M3 en adelante)

Donde

- La siguiente pregunta es ¿Donde ubicamos el código del **scheduler** entonces?
- La respuesta es trivial: En el handler de Interrupción del Timer seleccionado.
- Casos prácticos:
  1. SYSTICK en los Cortex-M (Recordar que el SYSTICK forma parte de un core CORTEX-M3 en adelante)
  2. El Timer T1 en un SITARA 533x. En la Beagle Bone se utiliza el T1

Donde

- La siguiente pregunta es ¿Donde ubicamos el código del **scheduler** entonces?
- La respuesta es trivial: En el handler de Interrupción del Timer seleccionado.
- Casos prácticos:
  1. SYSTICK en los Cortex-M (Recordar que el SYSTICK forma parte de un core CORTEX-M3 en adelante)
  2. El Timer T1 en un SITARA 533x. En la Beagle Bone se utiliza el T1
  3. El Timer T1 del PIT (Programable Interrupt Timer) en cualquier PC IBM compatible, conocido como timer tick

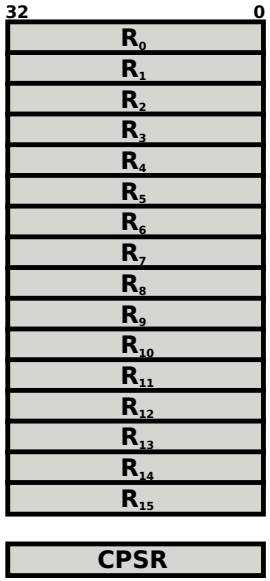
Que

Que

- Siempre se necesita resguardar los registros de propósito general R0-R15, y el CPSR. Pero si están habilitados los coprocesadores...

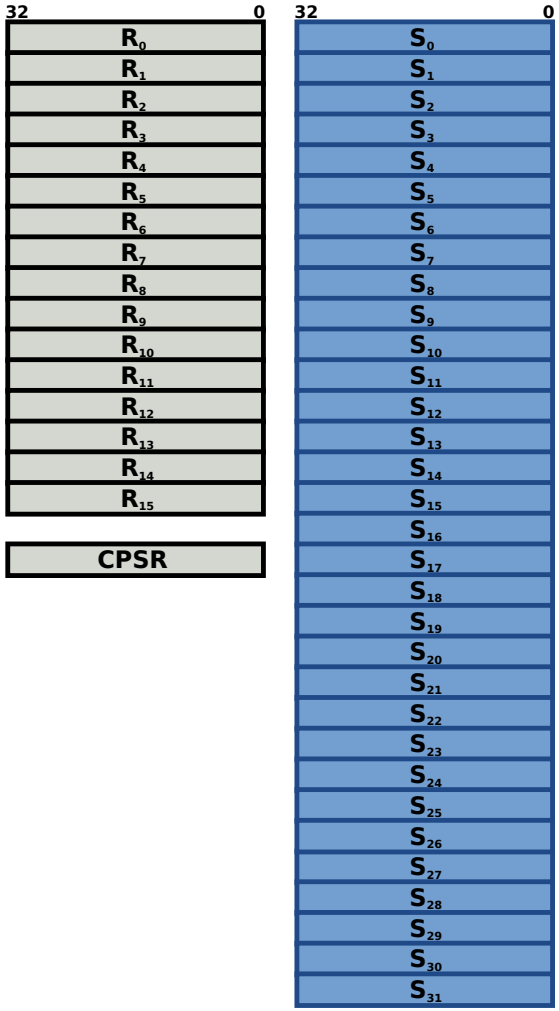
Que

- Siempre se necesita resguardar los registros de propósito general R0-R15, y el CPSR. Pero si están habilitados los coprocesadores...



Que

- Siempre se necesita resguardar los registros de propósito general R0-R15, y el CPSR. Pero si están habilitados los coprocesadores...

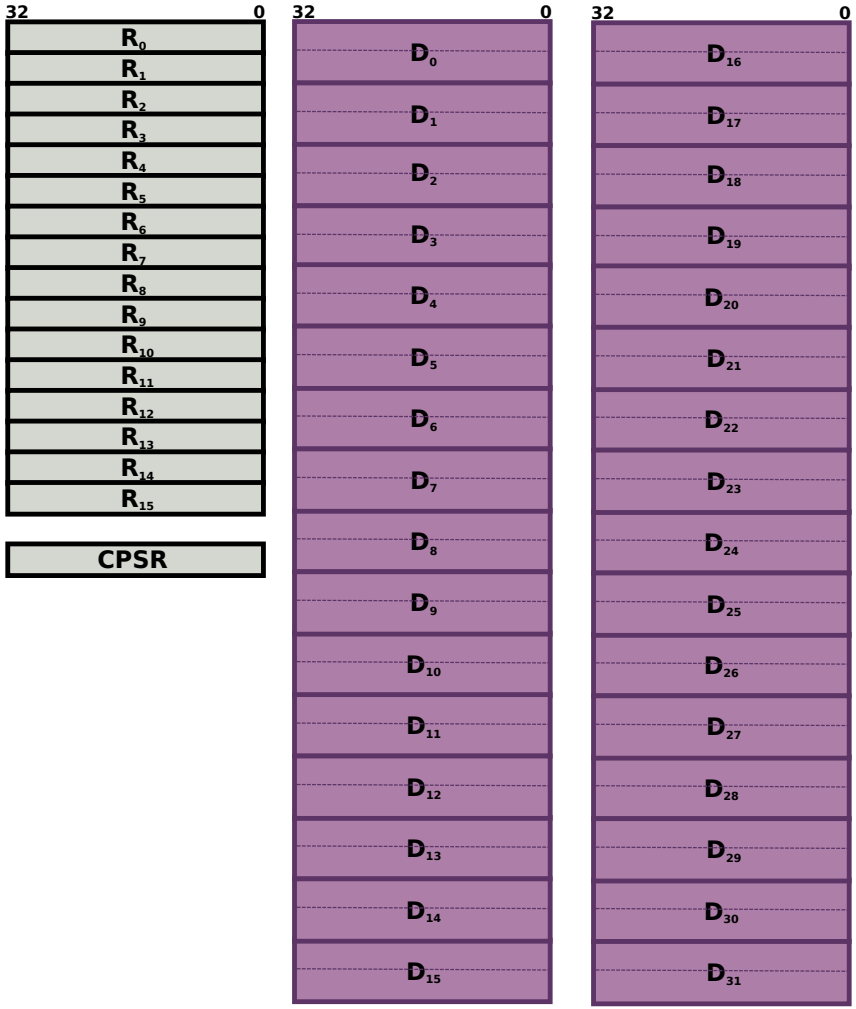


Que

- Siempre se necesita resguardar los registros de propósito general R0-R15, y el CPSR. Pero si están habilitados los coprocesadores...

Que

- Siempre se necesita resguardar los registros de propósito general R0-R15, y el CPSR. Pero si están habilitados los coprocesadores...



Guardando los registros del core

Guardando los registros del core

- Si el scheduler se escribe como handler de una Interrupción, entonces el procesador pasa a ejecutar de modo User a Modo IRQ (no se recomiendo Fast IRQ.

Guardando los registros del core

- Si el scheduler se escribe como handler de una Interrupción, entonces el procesador pasa a ejecutar de modo User a Modo IRQ (no se recomiendo Fast IRQ.
- LR recibe el PC+4 (efecto pipeline)

Guardando los registros del core

- Si el scheduler se escribe como handler de una Interrupción, entonces el procesador pasa a ejecutar de modo User a Modo IRQ (no se recomiendo Fast IRQ.
- LR recibe el PC+4 (efecto pipeline)
- El resto queda tal cual.

Guardando los registros del core

- Si el scheduler se escribe como handler de una Interrupción, entonces el procesador pasa a ejecutar de modo User a Modo IRQ (no se recomiendo Fast IRQ.
- LR recibe el PC+4 (efecto pipeline)
- El resto queda tal cual.
- EL contexto se lleva a la pila y de allí se mueve al **TCB**.

Guardando los registros del core

- Si el scheduler se escribe como handler de una Interrupción, entonces el procesador pasa a ejecutar de modo User a Modo IRQ (no se recomiendo Fast IRQ.
- LR recibe el PC+4 (efecto pipeline)
- El resto queda tal cual.
- EL contexto se lleva a la pila y de allí se mueve al **TCB**.

```
1  sub   lr,lr,#4
2  push  {lr}
3  push  {r0-r13}
4  /*Obtener TCD actual*/
5  /*Pasar contexto a TCB actual*/
6  /*Obtener el prox. TCB*/
7  pop   {r0-r13}
8  pop   {pc}
```

Context switch con FPU y SIMD

Context switch con FPU y SIMD

- Como hemos visto solo se debe salvar este contexto para aquellas tareas que lo utilicen, y solo si éste se ha modificado.

Context switch con FPU y SIMD

- Como hemos visto solo se debe salvar este contexto para aquellas tareas que lo utilicen, y solo si éste se ha modificado.
- De otro modo, si se opta por un approach mas conservador para simplificar el algoritmo de scheduling se está transfiriendo cada vez un bloque de tamaño 5x el de los Registros core.

Context switch con FPU y SIMD

- Como hemos visto solo se debe salvar este contexto para aquellas tareas que lo utilicen, y solo si éste se ha modificado.
- De otro modo, si se opta por un approach mas conservador para simplificar el algoritmo de scheduling se está transfiriendo cada vez un bloque de tamaño 5x el de los Registros core.
- Nuevamente estamos ante una disyuntiva: ¿Como podemos comprobar si este contexto cambió?.

Context switch con FPU y SIMD

- Como hemos visto solo se debe salvar este contexto para aquellas tareas que lo utilicen, y solo si éste se ha modificado.
- De otro modo, si se opta por un approach mas conservador para simplificar el algoritmo de scheduling se está transfiriendo cada vez un bloque de tamaño 5x el de los Registros core.
- Nuevamente estamos ante una disyuntiva: ¿Como podemos comprobar si este contexto cambió?.
- Respuesta: No podemos.

Context switch con FPU y SIMD

- Como hemos visto solo se debe salvar este contexto para aquellas tareas que lo utilicen, y solo si éste se ha modificado.
- De otro modo, si se opta por un approach mas conservador para simplificar el algoritmo de scheduling se está transfiriendo cada vez un bloque de tamaño 5x el de los Registros core.
- Nuevamente estamos ante una disyuntiva: ¿Como podemos comprobar si este contexto cambió?.
- Respuesta: No podemos.
- ¿Entonces? ¿Que hacemos?

Leer el manual! (nunca falla)

Del manual de arquitectura Parte B System Level Architecture:

B1.11.3. Context switching with the Advanced SIMD and Floating-point Extensions

In an implementation that includes one or both of the Advanced SIMD and Floating-point Extensions, if the Floating-point registers are used by only a subset of processes, the operating system might implement lazy context switching of the extension registers and extension system registers. In the simplest lazy context switch implementation, the primary context switch software disables the Advanced SIMD and Floating-point Extensions, by disabling access to coprocessors CP10 and CP11 in the Coprocessor Access Control Register, see Enabling Advanced SIMD and floating-point support on page B1-1228. Subsequently, when a process or thread attempts to use an Advanced SIMD or Floating-point nstruction, it triggers an Undefined Instruction exception. The operating system responds by saving and restoring the extension registers and extension system registers. Typically, it then re-executes the Advanced SIMD or Floating-point instruction that generated the Undefined Instruction exception.

Bienvenidos al modelo de Programación de Sistemas

Bienvenidos al modelo de Programación de Sistemas

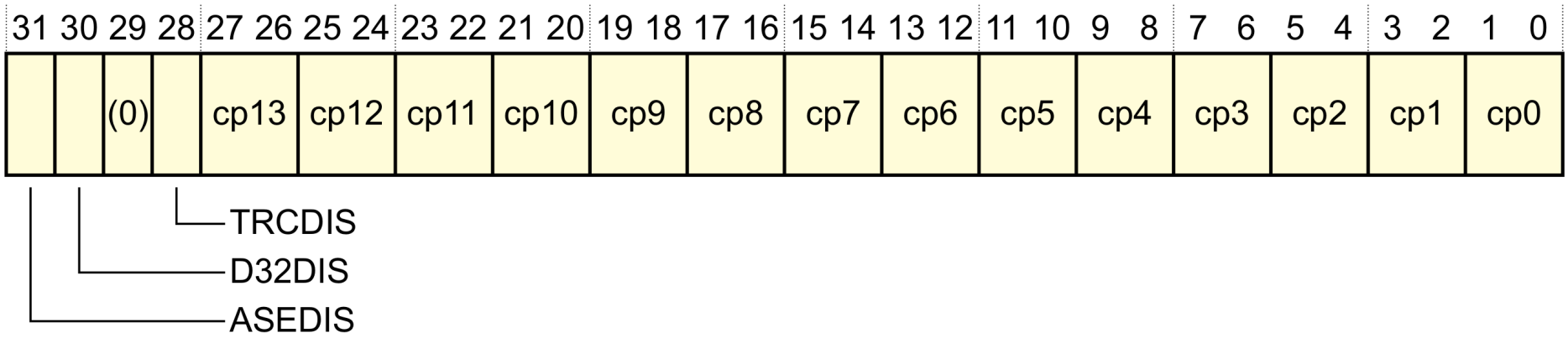
CPACR, Coprocessor Access Control Register, VMSA

En este registro solo accesible en modo privilegiado mediante las instrucciones de acceso a este tipo de registros se puede activar / desactivar los Coprocesadores cp10 (VFP) y cp11 (SIMD)

Bienvenidos al modelo de Programación de Sistemas

CPACR, Coprocessor Access Control Register, VMSA

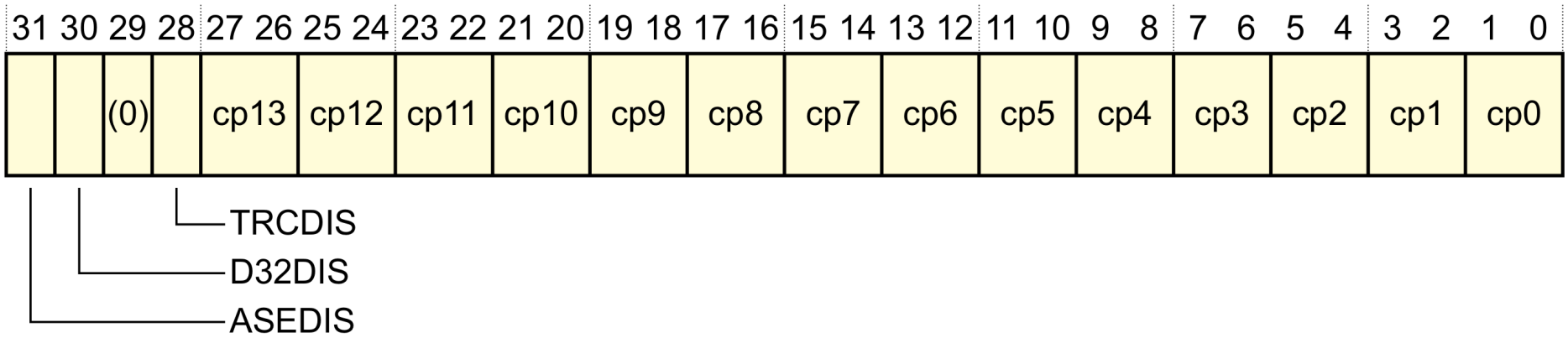
En este registro solo accesible en modo privilegiado mediante las instrucciones de acceso a este tipo de registros se puede activar / desactivar los Coprocesadores cp10 (VFP) y cp11 (SIMD)



Bienvenidos al modelo de Programación de Sistemas

CPACR, Coprocessor Access Control Register, VMSA

En este registro solo accesible en modo privilegiado mediante las instrucciones de acceso a este tipo de registros se puede activar / desactivar los Coprocesadores cp10 (VFP) y cp11 (SIMD)



valor	Acción
00	Acceso Denegado. Genera Undefined Instruction Exception.
01	Acceso en Modo Prilegiado solamente
10	Reservado
11	Acceso en Modo Privilegiado y User

Acceso al CPACR

```
1 MRC p15, 0, <Rt>, c1, c0, 2 ; Lee CPACR en el reg. core Rt
2 MCR p15, 0, <Rt>, c1, c0, 2 ; Escribe Rt en CPACR
```

Acceso al CPACR

```
1 MRC p15, 0, <Rt>, c1, c0, 2 ; Lee CPACR en el reg. core Rt
2 MCR p15, 0, <Rt>, c1, c0, 2 ; Escribe Rt en CPACR
```

- El registro CAPCR se lee desde el registro de control del Coprocesador 15.

Acceso al CPACR

```
1 MRC p15, 0, <Rt>, c1, c0, 2 ; Lee CPACR en el reg. core Rt
2 MCR p15, 0, <Rt>, c1, c0, 2 ; Escribe Rt en CPACR
```

- El registro CAPCR se lee desde el registro de control del Coprocesador 15.
- En este Copropcesador se habilitan las funciones mas importantes del sistema, incluido el acceso a los Coprocesadores CP0 a CP13.

Acceso al CPACR

```
1 MRC p15, 0, <Rt>, c1, c0, 2 ; Lee CPACR en el reg. core Rt
2 MCR p15, 0, <Rt>, c1, c0, 2 ; Escribe Rt en CPACR
```

- El registro CAPCR se lee desde el registro de control del Coprocesador 15.
- En este Copropcesador se habilitan las funciones mas importantes del sistema, incluido el acceso a los Coprocesadores CP0 a CP13.
- Con este par de instrucciones se habilita o deshabilitan los Corpocesadores CP10 y CP11 que en este caso nos interesa.

Lazy FPU Context Switch

Lazy FPU Context Switch

- Se implementa en el Handler de Undefined Instruction Exception.

Lazy FPU Context Switch

- Se implementa en el Handler de Undefined Instruction Exception.
- El truco consiste en tener CP10 y CP11 deshabilitados

Lazy FPU Context Switch

- Se implementa en el Handler de Undefined Instruction Exception.
- El truco consiste en tener CP10 y CP11 deshabilitados
- Cuando el procesador fetchea una instrucción y al decodificarla la identifica como SIMD o como una operación de la VFP, intenta derivarla a esas unidades.

Lazy FPU Context Switch

- Se implementa en el Handler de Undefined Instruction Exception.
- El truco consiste en tener CP10 y CP11 deshabilitados
- Cuando el procesador fetchea una instrucción y al decodificarla la identifica como SIMD o como una operación de la VFP, intenta derivarla a esas unidades.
- Si éstos coprocesadores no están habilitados generará una Excepción Undefined Instruction.

Lazy FPU Context Switch

- Se implementa en el Handler de Undefined Instruction Exception.
- El truco consiste en tener CP10 y CP11 deshabilitados
- Cuando el procesador fetchea una instrucción y al decodificarla la identifica como SIMD o como una operación de la VFP, intenta derivarla a esas unidades.
- Si éstos coprocesadores no están habilitados generará una Excepción Undefined Instruction.
- El handler analizará la causa de la falta, y si corresponde a estas instrucciones, habilitará los coprocesadores, busque el tcb de la tarea actual en el cual deberá haber un puntero no nulo a una estructura especial que contendrá todos los registros de punto flotante y SIMD y los cargará en los registros del procesador.

Lazy FPU Context Switch

- Se implementa en el Handler de Undefined Instruction Exception.
- El truco consiste en tener CP10 y CP11 deshabilitados
- Cuando el procesador fetchea una instrucción y al decodificarla la identifica como SIMD o como una operación de la VFP, intenta derivarla a esas unidades.
- Si éstos coprocesadores no están habilitados generará una Excepción Undefined Instruction.
- El handler analizará la causa de la falta, y si corresponde a estas instrucciones, habilitará los coprocesadores, busque el tcb de la tarea actual en el cual deberá haber un puntero no nulo a una estructura especial que contendrá todos los registros de punto flotante y SIMD y los cargará en los registros del procesador.
- Eventualmente mantendrá un estado (un flag global por ejemplo) para que el scheduler al suspender la tarea guarde el contexto VFP / SIMD e inactive ambos coprocesadores antes de switchear a la siguiente tarea

Conclusiones

Conclusiones

- No estamos desarrollando una aplicación, sino un sistema.

Conclusiones

- No estamos desarrollando una aplicación, sino un sistema.
- Cada funcionalidad relevante, o evento significativo, lleva asociada una tarea.

Conclusiones

- No estamos desarrollando una aplicación, sino un sistema.
- Cada funcionalidad relevante, o evento significativo, lleva asociada una tarea.
- Se necesita un sistema capaz de administrar la ejecución de estas tareas y su acceso a los recursos

Conclusiones

- No estamos desarrollando una aplicación, sino un sistema.
- Cada funcionalidad relevante, o evento significativo, lleva asociada una tarea.
- Se necesita un sistema capaz de administrar la ejecución de estas tareas y su acceso a los recursos
- Nos hemos basado mayormente en Cortex-A.

Conclusiones

- No estamos desarrollando una aplicación, sino un sistema.
- Cada funcionalidad relevante, o evento significativo, lleva asociada una tarea.
- Se necesita un sistema capaz de administrar la ejecución de estas tareas y su acceso a los recursos
- Nos hemos basado mayormente en Cortex-A.
- No obstante, Cortex-M entra en el mundo de la FPU a partir de M4. Con modelos mas reducidos pero conceptualmente similares.