Bring ideas to life
VIA University College

FreeRTOS

Introduction

Todays Agenda

Understand the basic concepts and requirements of real-time systems
Understand the basic concepts of a Real-Time Operating System (RTOS)
Be able to program a simple real-time system using FreeRTOS

What is a Real-time system?

A real-time system is any information processing system which must respond within a **finite** and **specified period.**

- The correctness depends not only on the logical result but also the time it was delivered
- Failure to respond in time is as bad as the wrong response!

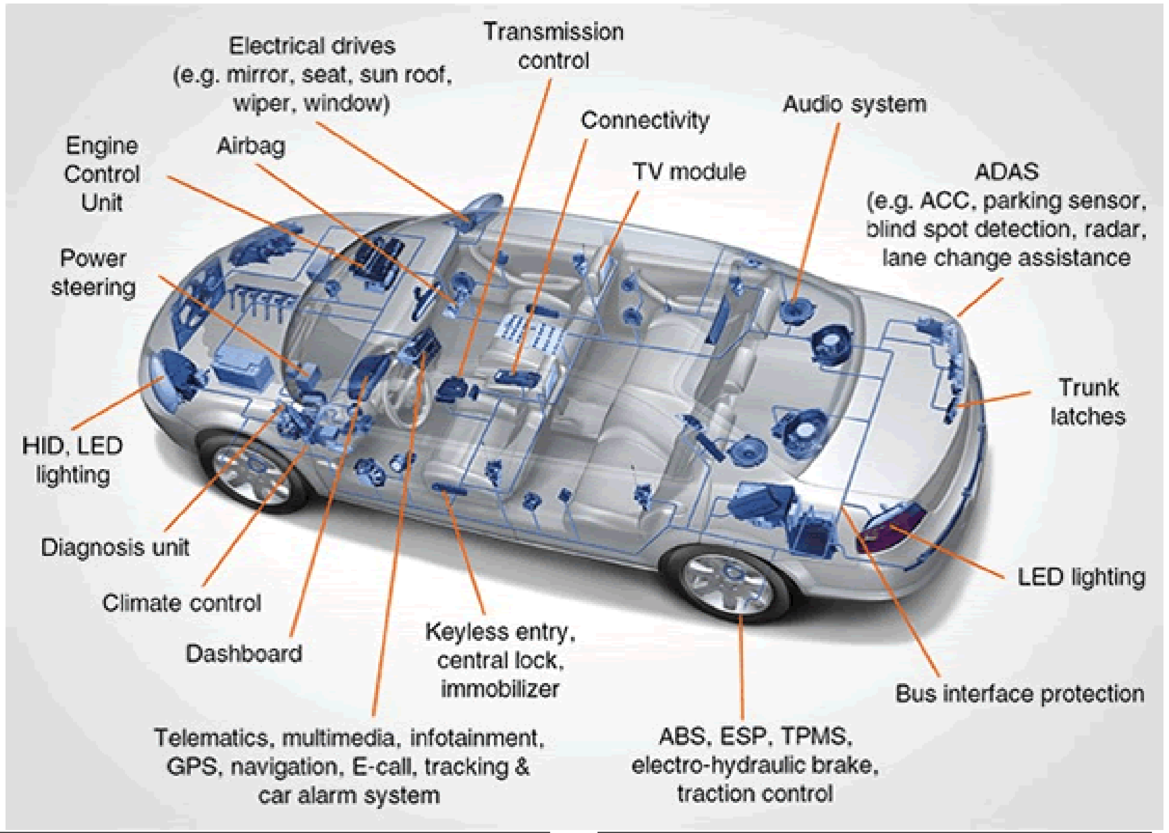
A real-time system is not necessarily fast but
it will respond within a guarantied time limit

Speed is not the issue - reliability on deadlines is!

Embedded Systems

- The computer in an embedded system is a component in a larger engineering system
- The computer is normally interfaced directly to some physical equipment
- Is dedicated to monitor and/or control the physical equipment

Typical known as **Embedded-Systems**


Electrical drives (e.g. mirror, seat, sun roof, wiper, window)
Transmission control
Connectivity
Audio system
Engine Control Unit
Airbag
TV module
ADAS (e.g. ACC, parking sensor, blind spot detection, radar, lane change assistance
Power steering
Trunk latches
HID, LED lighting
Diagnosis unit
LED lighting
Climate control
Dashboard
Keyless entry, central lock, immobilizer
Bus interface protection
Telematics, multimedia, infotainment, GPS, navigation, E-call, tracking & car alarm system
ABS, ESP, TPMS, electro-hydraulic brake, traction control

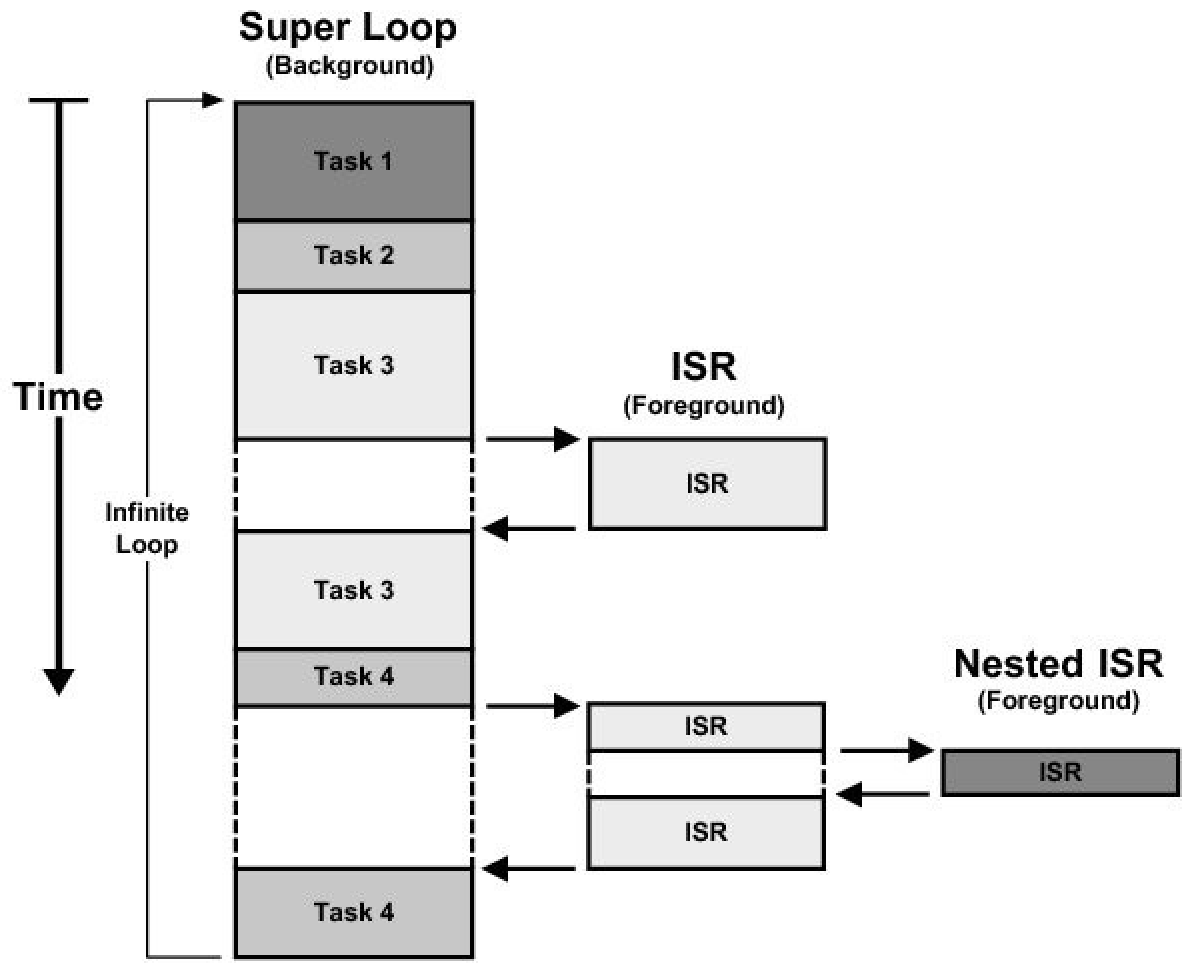
Embedded Systems

- The user don’t care (or know) that he or she is interacting with a computer (or many computers)

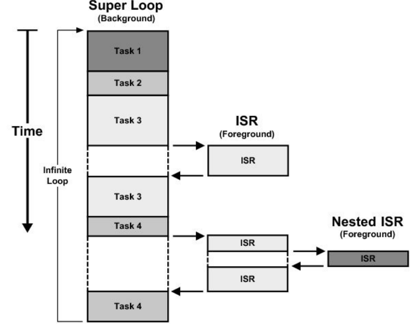
- 99% of **all produced processors** are used **in embedded systems** (and they are mainly running C programs)

- Many embedded systems are **NOT** real time systems or use an OS/RTOS
  - These are often simple **Foreground/Background Systems**
    - *More on next slide*

Foreground/Background Systems



Foreground/Background Systems



The **problem** with foreground/background systems is how to deal with timing issues

- If the interrupt handlers deal with functionality the interrupt latency becomes large
- If the interrupt handlers only set flags the current function executing cannot be pre-empted

Conclusion

- For even modest size systems foreground/background systems are complex to program and maintain
- For real time systems it is very difficult to determine if deadlines can be meet in a foreground/background system

What is an Operating system?

In computing, an **operating system (OS)** is an **interface between hardware and user**, which is responsible for the management and coordination of activities and the sharing of the resources of a computer, that acts as a host for computing applications run on the machine

As a host, one of the purposes of an OS is to handle the **details of the operation of the hardware.** This relieves application programs from having to manage these details and makes it easier to write applications. Almost all computers use an OS of some type

One of the purposes of an operating system is to handle the **resource allocation and access protection** of the hardware

This relieves the application programmers from having to manage these details

An Operating System is **just a program** that **supports the use of threads/tasks** and **controls the access to resources**

Source: https://openbookproject.net/courses/intro2ict/system/os_intro.html

What is a Real-Time System?

Any system in which ***the time at which output is produced is significant****. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness.*

- Oxford dictionary of computing

A real-time system is a system that is required to ***react to stimuli*** *from the environment (including the passage of physical time)* ***within time intervals dictated by the environment****.*

- Randell et al.

Categories of Real-Time Systems

Soft Real-Time

- Response times are important
- System still will work if deadlines sometimes are missed

- E.g. Windows, Linux, MacOS, Infotainment systems etc.

Soft-Firm Real-Time

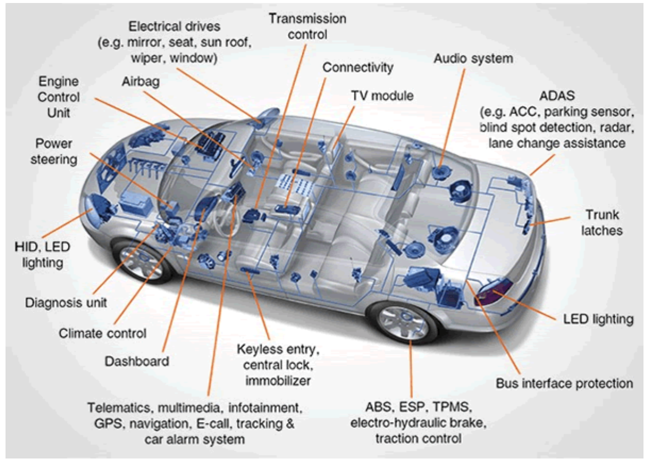
- Systems which are soft real-time but in which there is no benefit from late delivery of service

- E.g. Financial forecast systems etc.

Hard Real-Time

- Fatal if deadlines are missed
- Correctness depends not only on the logical result but also the time it was delivered
- Failure to respond is as bad as the wrong response!

- E.g. Flight Control Systems, ABS-Brakes, Airbags, Engine Control Systems etc.


Electrical drives (e.g. mirror, seat, sun roof, wiper, window)
Transmission control
Connectivity
Audio system
Engine Control Unit
Airbag
TV module
ADAS (e.g. ACC, parking sensor, blind spot detection, radar, lane change assistance
Power steering
Trunk latches
HID, LED lighting
Diagnosis unit
LED lighting
Climate control
Dashboard
Keyless entry, central lock, immobilizer
Bus interface protection
Telematics, multimedia, infotainment, GPS, navigation, E-call, tracking & car alarm system
ABS, ESP, TPMS, electro-hydraulic brake, traction control

A car includes all three categories

Hard Real-Time Embedded Systems Characteristics

- **Guaranteed response times**
  - We need to be able to predict with **confidence** the worst case response times for systems (more about this in the RTP1 elective course)
  - **Efficiency is important** but **<u>predictability is essential</u>**
- **Concurrent control of separate system components**
  - Devices operate in parallel in the real-world
  - Better to model this parallelism by concurrent entities/task in the program
- **Facilities to interact with special purpose hardware**
  - Need to be able to program devices in a reliable and abstract way

Hard Real-Time Embedded Systems Characteristics

- **Support for numerical computation**
  - Be able to support the discrete/continuous computation necessary for control system feed-back and feed-forward algorithms

- **Large and complex**
  - Vary from a few hundred lines of assembler or C to 20 million lines of code, also variety as well as size is an issue

- **Extreme reliability and safety**
  - Embedded systems typically control the environment in which they operate
  - **Failure to control can result in loss of life, damage to environment or economic loss**

What is an RTOS?

Real-**T**ime **O**perating **S**ystem (RTOS)

Essential features:

- Can handle Soft- and/or Hard Real-Time or both
- Reliability on deadlines and predictability
- Simulates that tasks running in parallel (Concurrency)

Nice to have features:

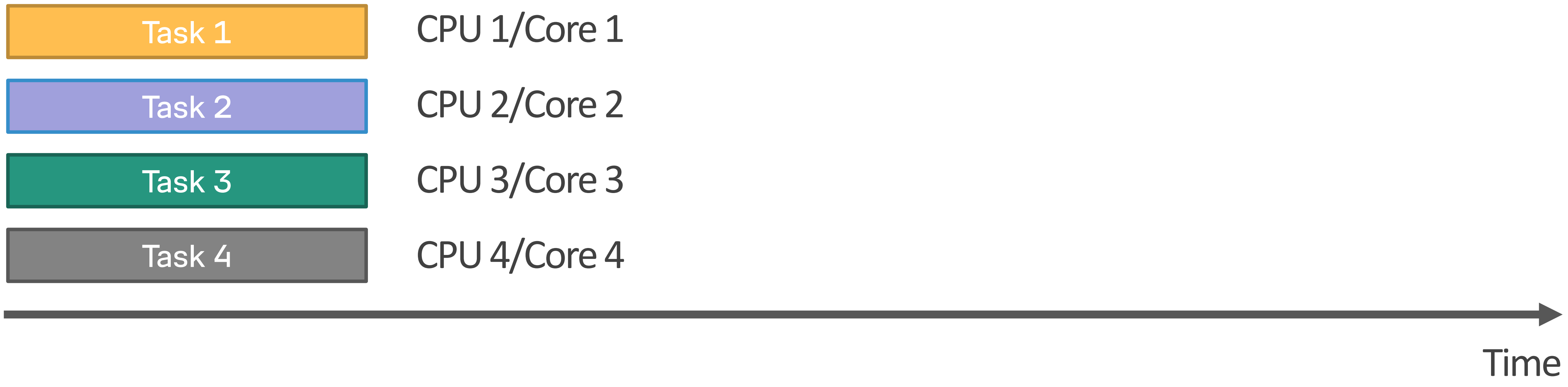
- Portability – FreeRTOS is nearly there
- Safe and easy coding

Multitasking

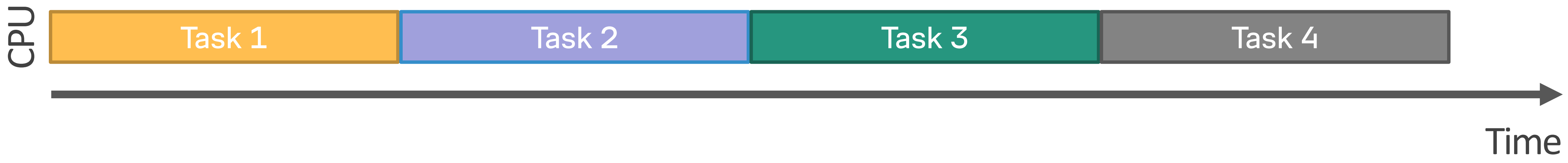
The job of the scheduler of an Operating System (OS) is to switch between the tasks

- Tasks shares the CPU power
- Tasks logically executes concurrently
- How much CPU each task is given depends on the scheduling policy of the OS

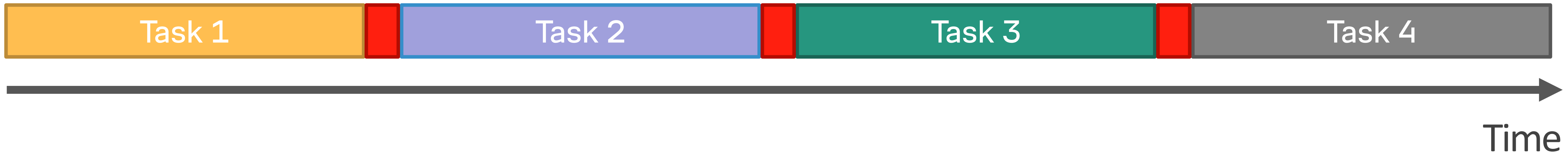
Multitasking (Multiple CPUs/Cores)



Multitasking (One CPU) Ideal



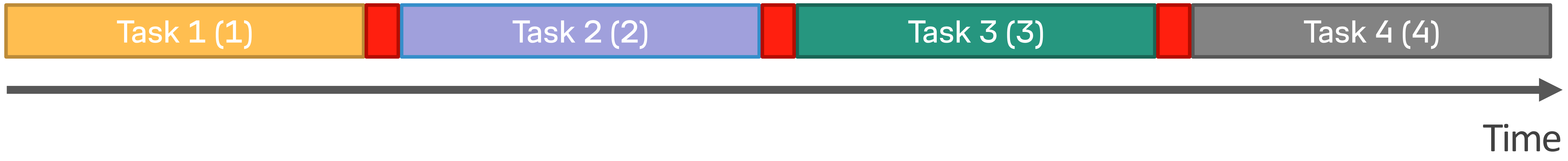
Multitasking (One CPU) With OS



🟥 OS Overhead (Task switching) (2-5 %)

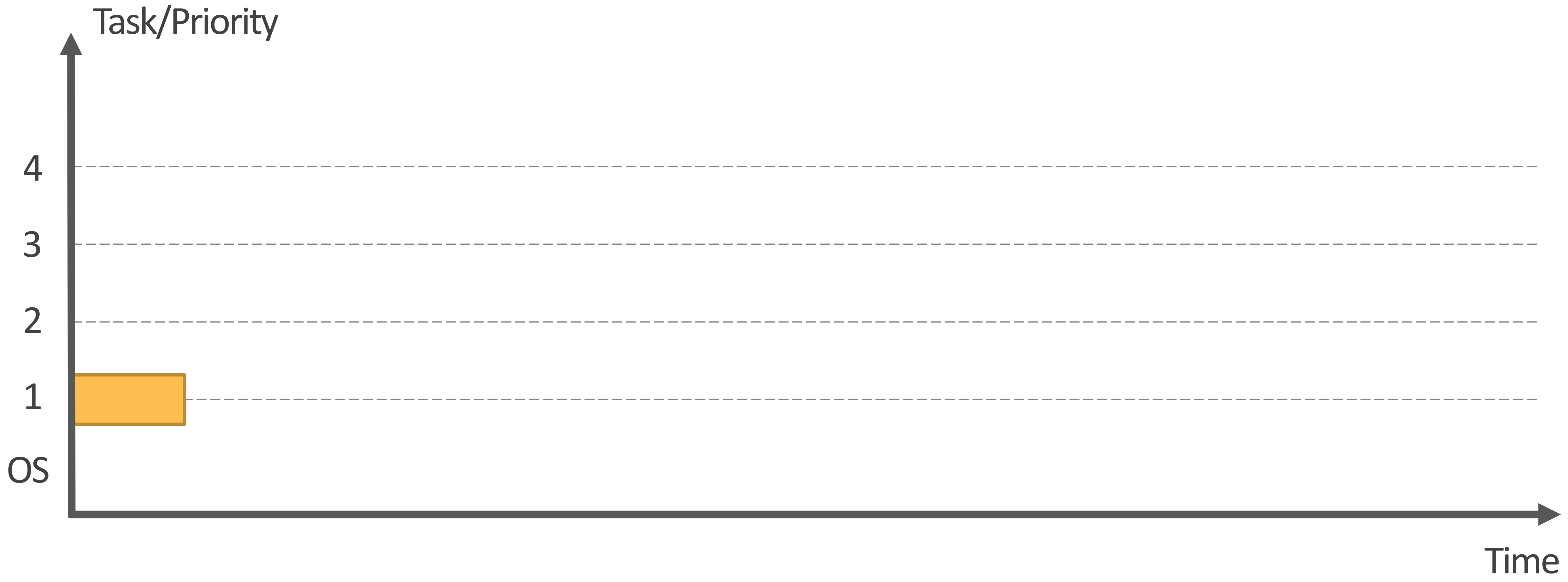
Closer look at our Tasks

Some task has often higher priority than others – priorities in (parentheses)



🟥 OS Overhead (Task switching)

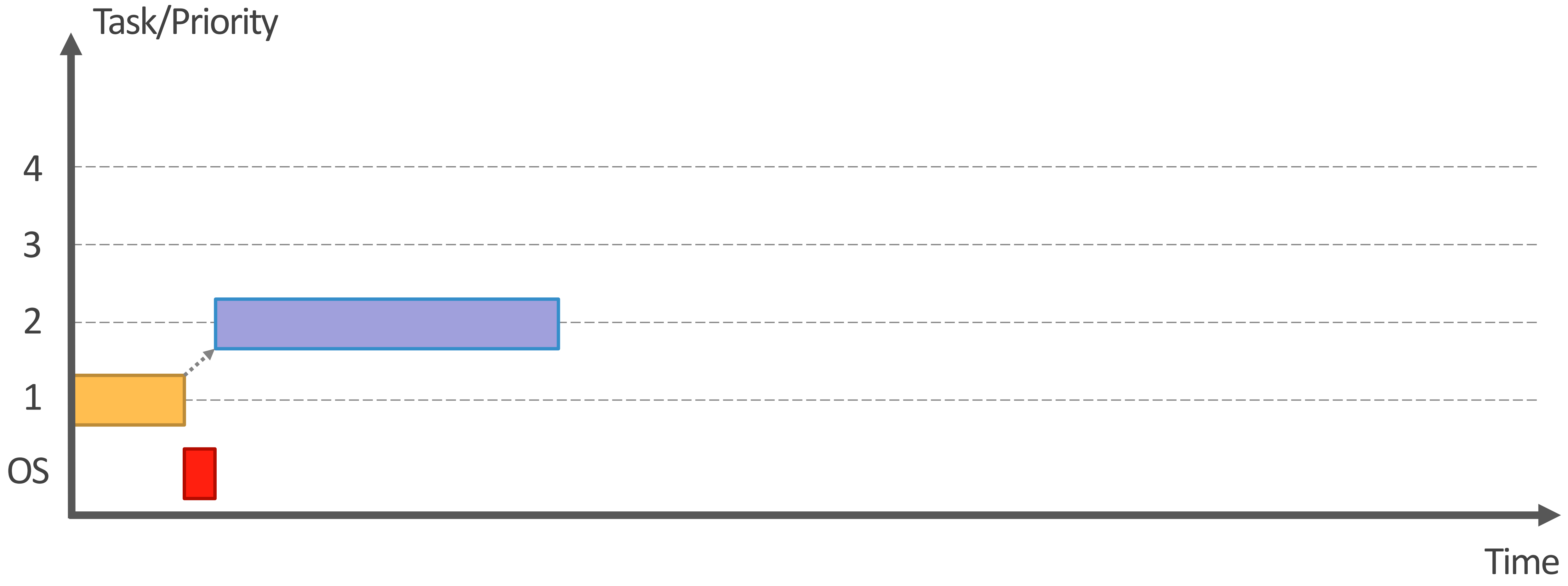
Closer look at our Tasks
Pre-emptive Scheduling



🟥 OS Overhead (Task switching)

Closer look at our Tasks

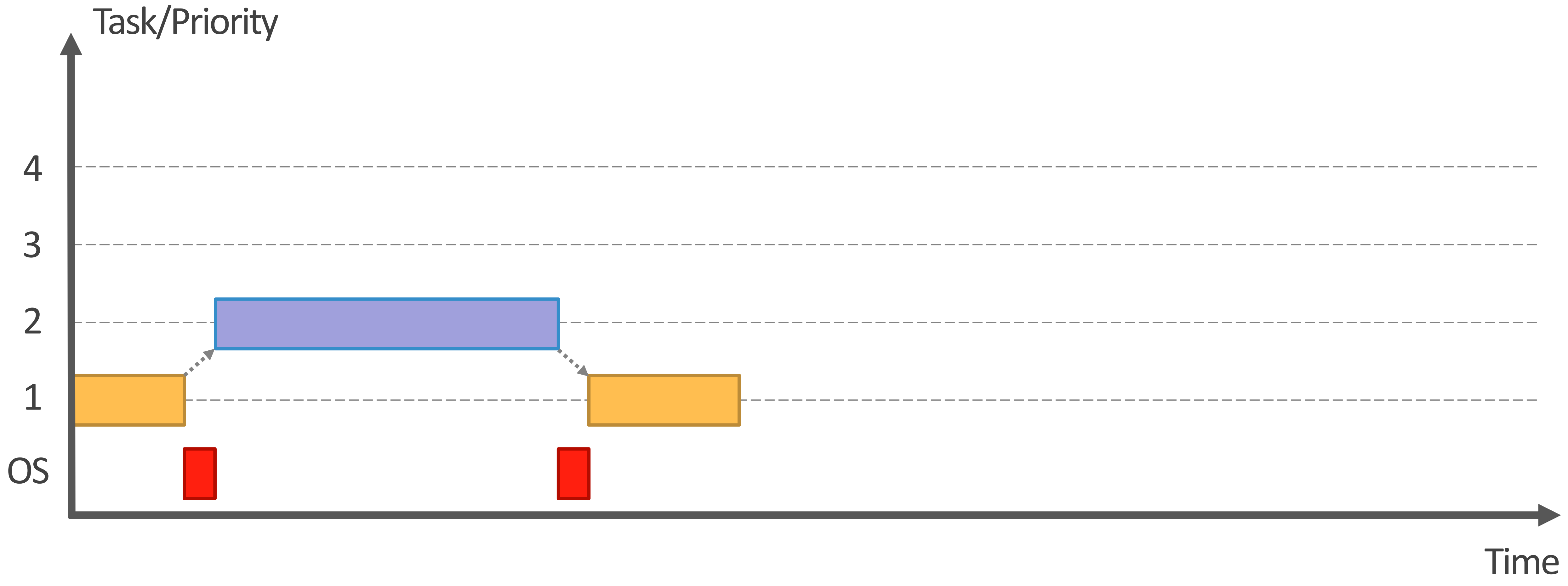
Pre-emptive Scheduling



🟥 OS Overhead (Task switching)

Closer look at our Tasks

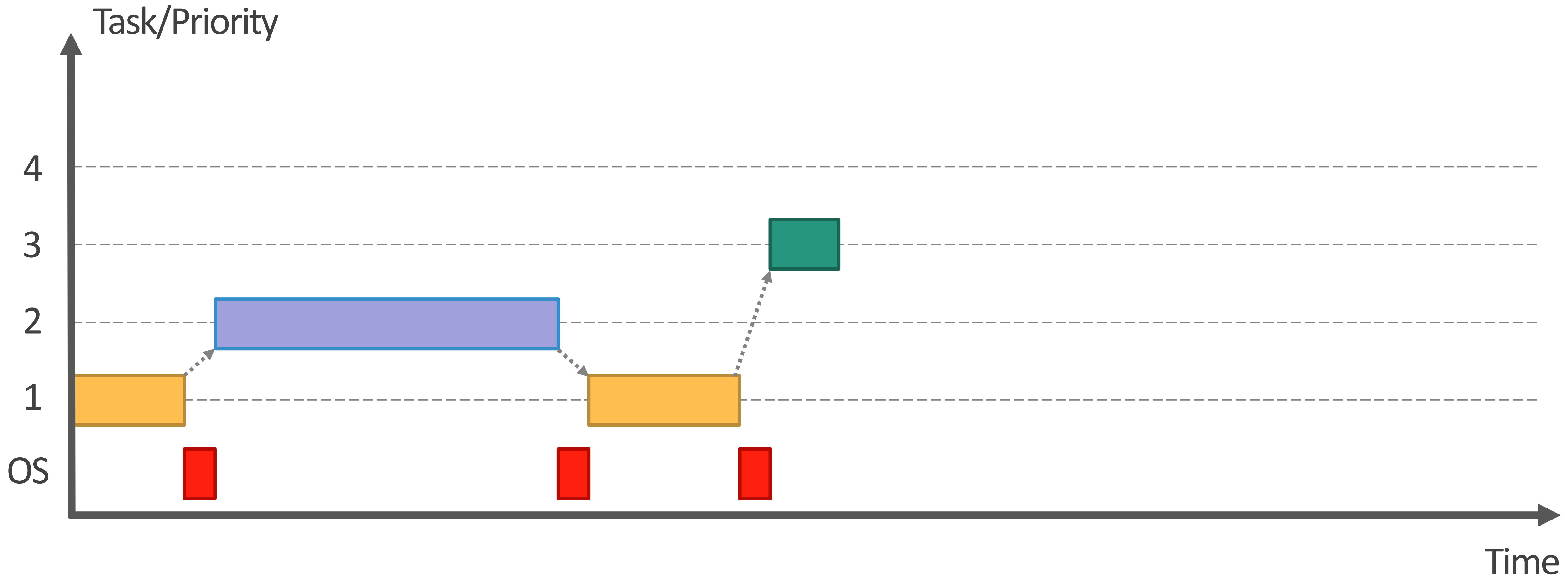
Pre-emptive Scheduling



🟥 OS Overhead (Task switching)

Closer look at our Tasks

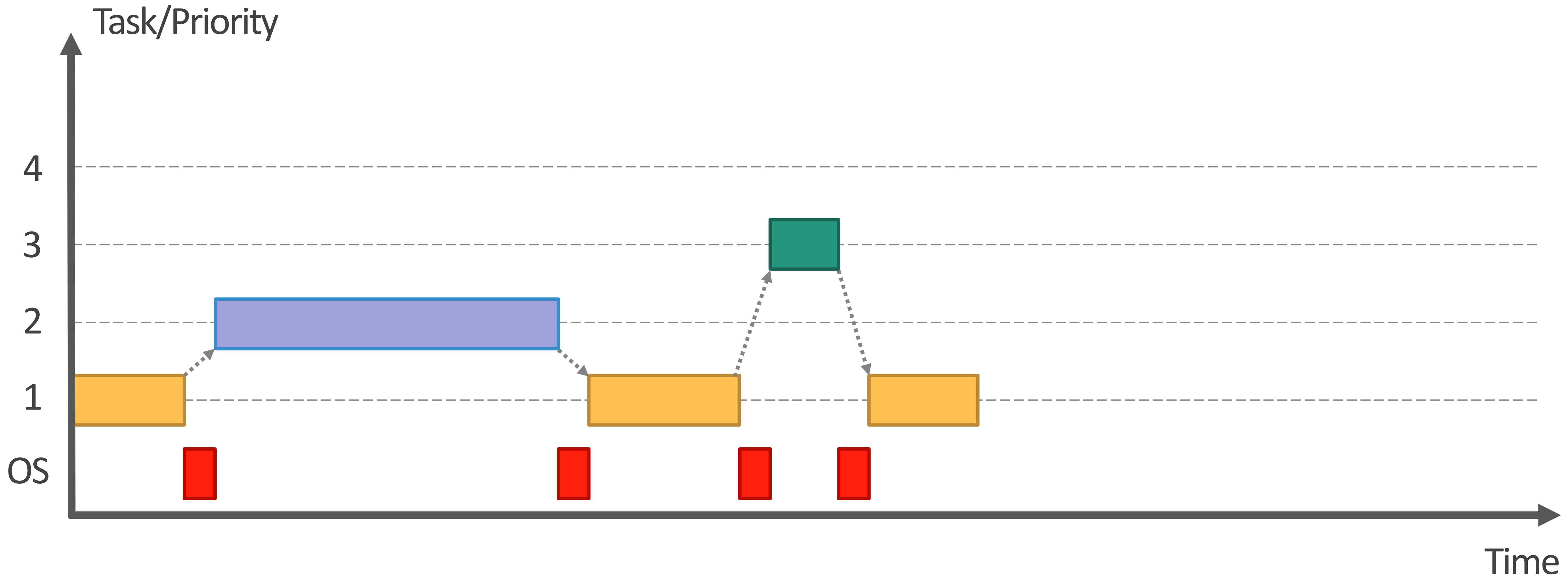
Pre-emptive Scheduling



🟥 OS Overhead (Task switching)

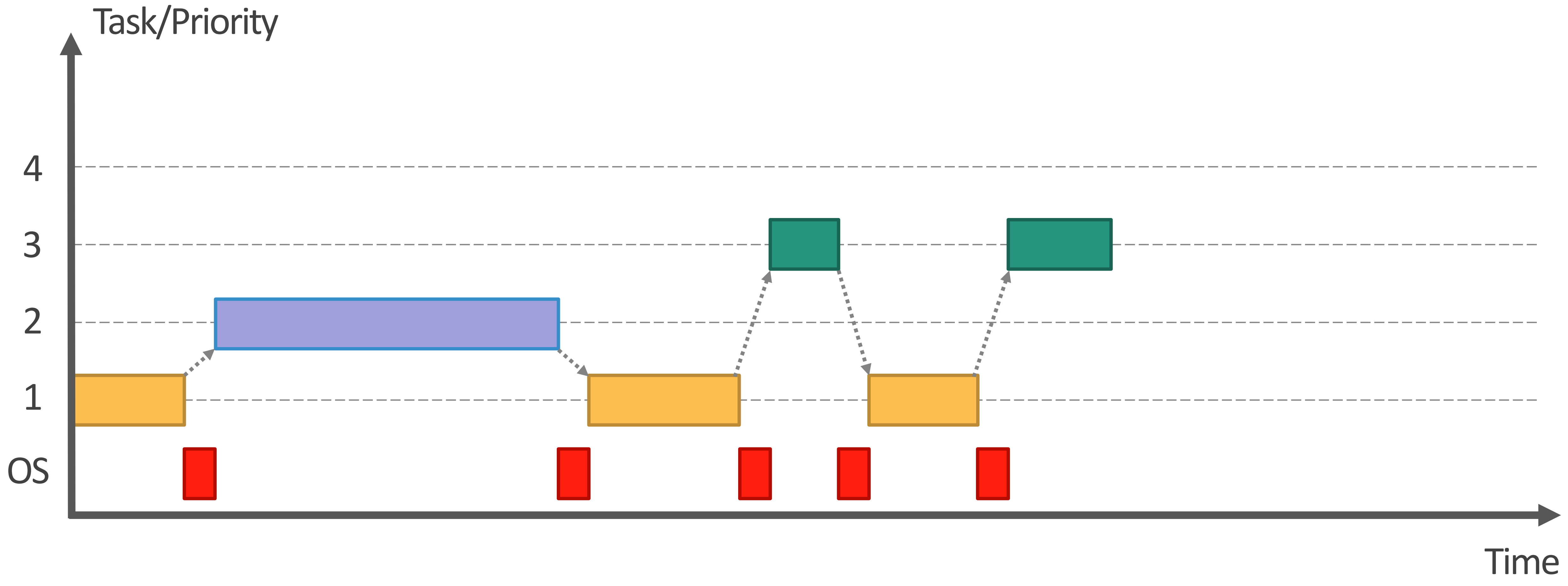
Closer look at our Tasks

Pre-emptive Scheduling



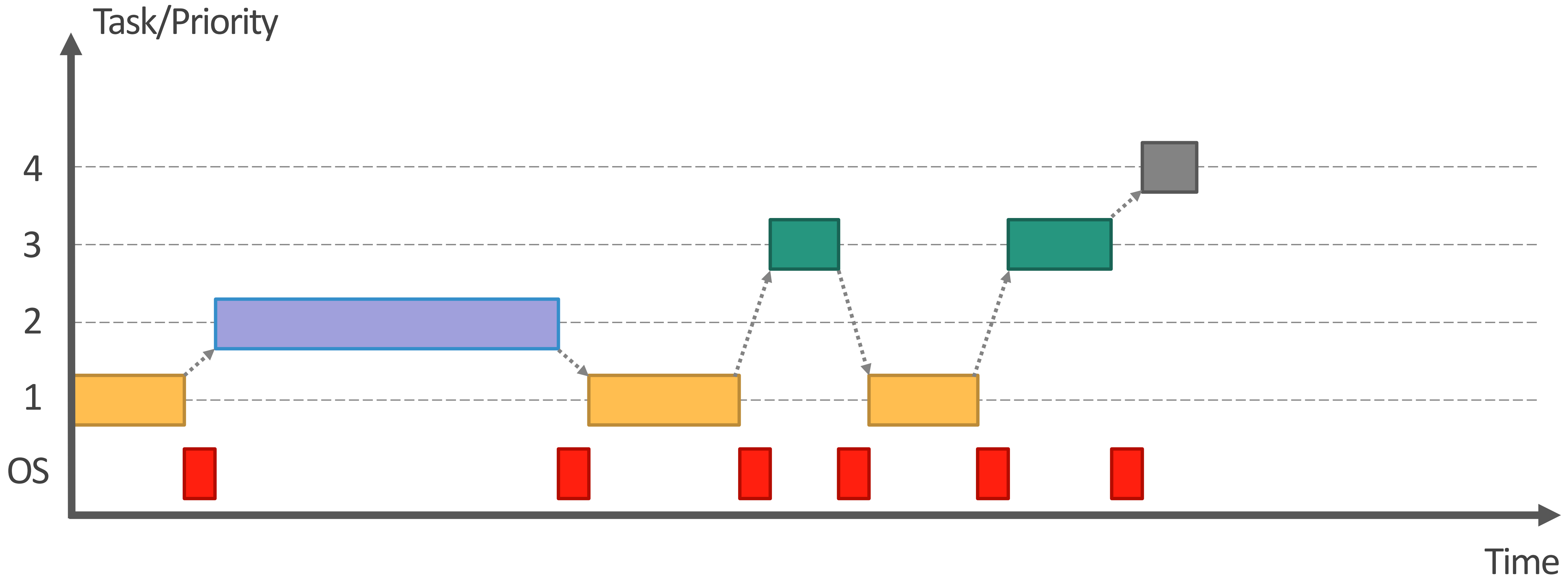
🟥 OS Overhead (Task switching)

Closer look at our Tasks
Pre-emptive Scheduling



🟥 OS Overhead (Task switching)

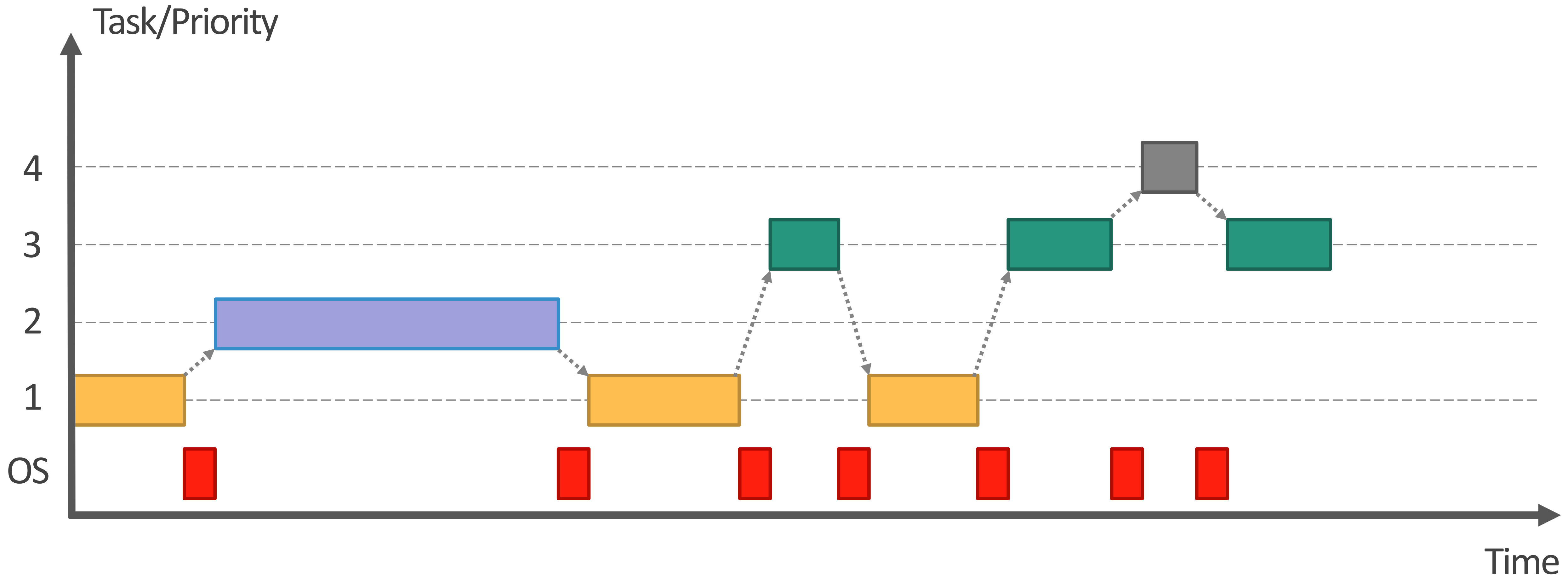
Closer look at our Tasks
Pre-emptive Scheduling



🟥 OS Overhead (Task switching)

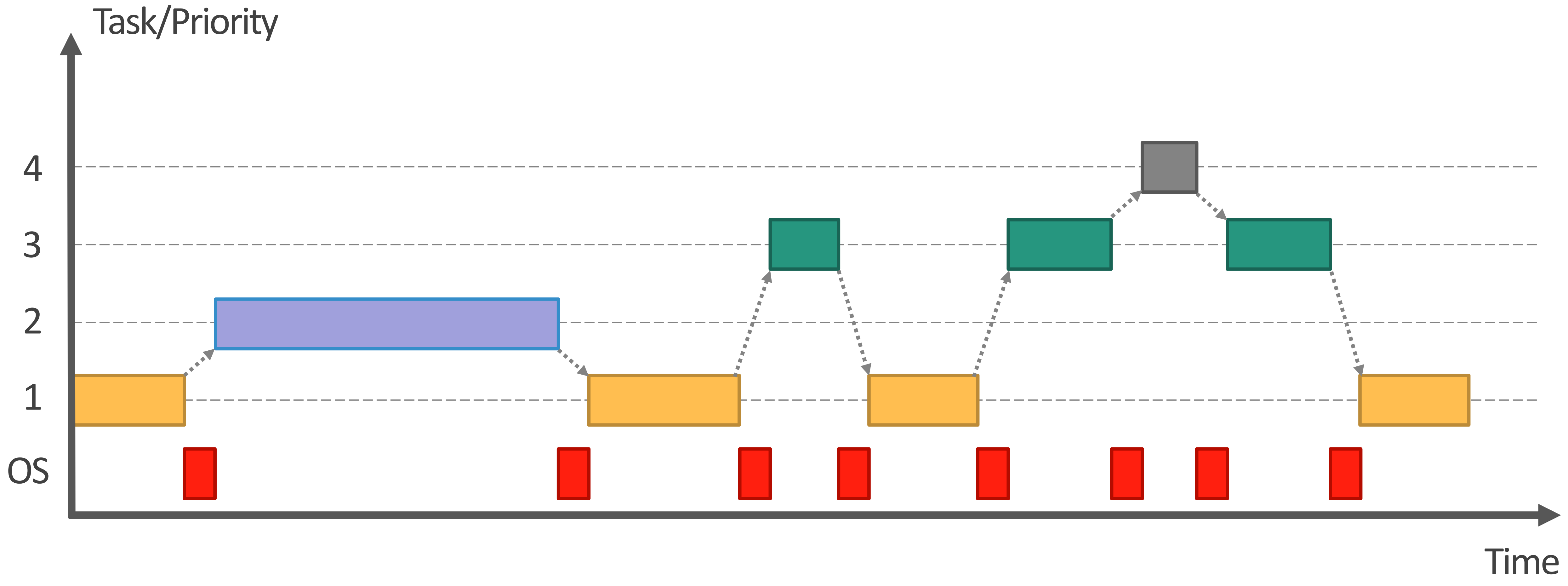
Closer look at our Tasks

Pre-emptive Scheduling



🟥 OS Overhead (Task switching)

Closer look at our Tasks
Pre-emptive Scheduling



🟥 OS Overhead (Task switching)

Small wrap up Video's

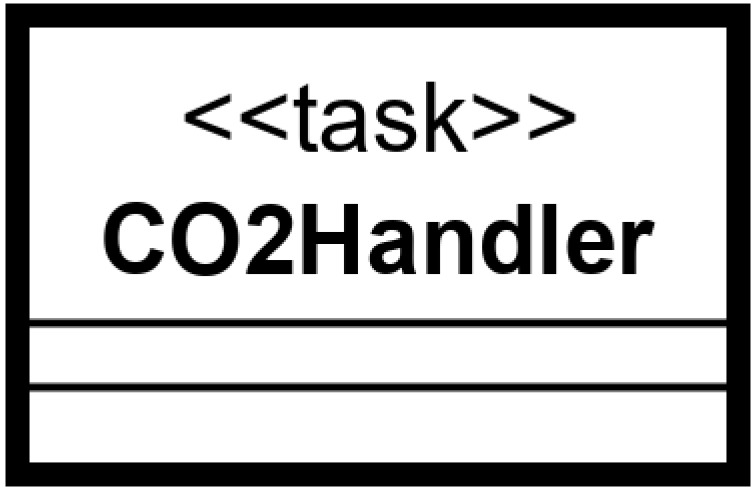
Foreground-background system vs. kernel based approach

Intro to Kernels: Getting Started with Micrium OS #1:
https://www.youtube.com/watch?v=A86nuYOhmqQ&list=PL-awFRrdECXu9I7ybAI5tEgwn7BQF6N56&index=2

Note: We are not going to use Micrium OS but FreeRTOS

Introduction to RTOS Part 1 - What is a Real-Time Operating System (RTOS)?
https://www.youtube.com/watch?v=F321087yYy4&list=PLXyB2ILBXW5FLc7j2hLcX6sAGbmH0JxX8

What is a Task?

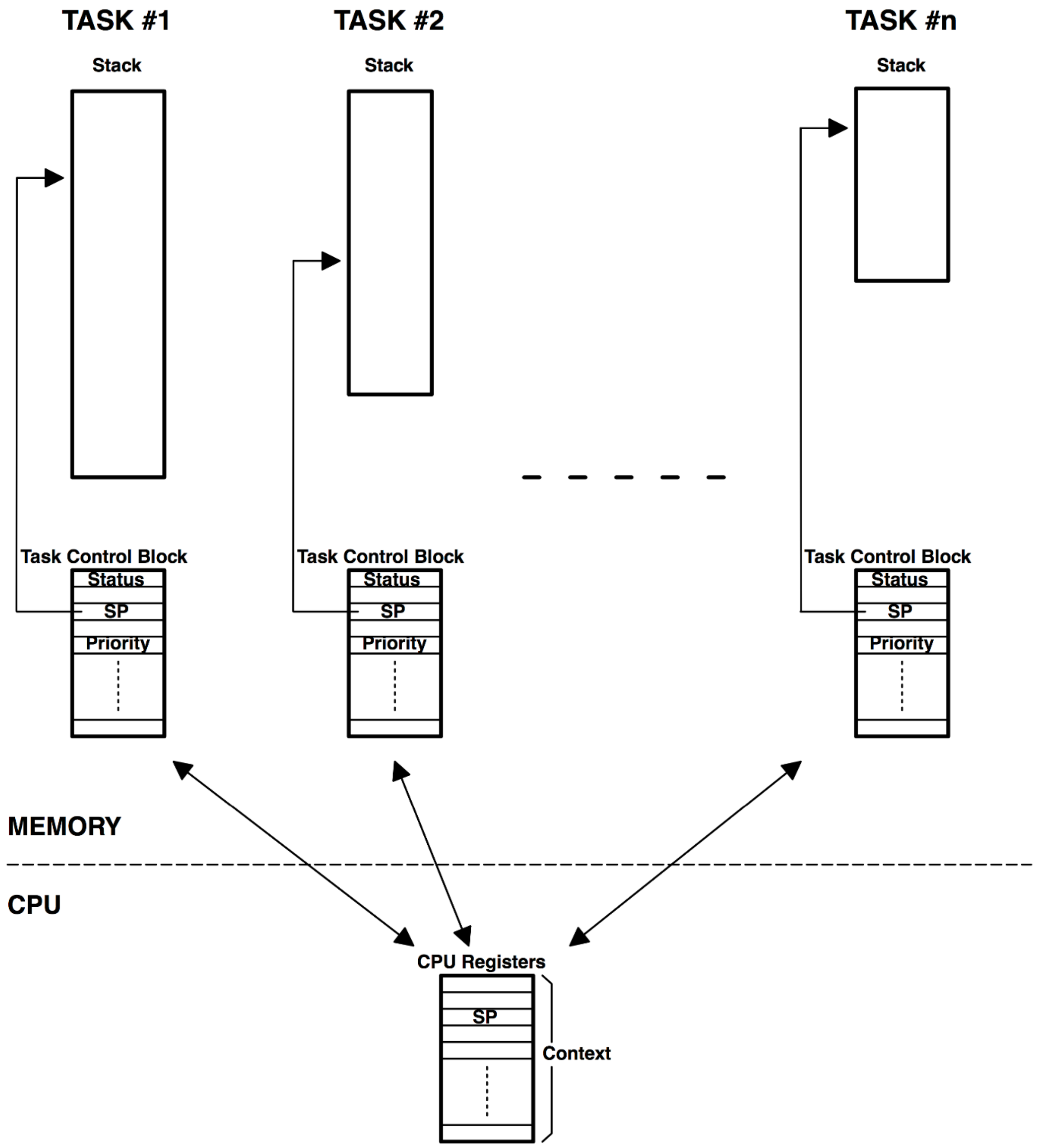


Many names: thread, process etc.
An active/executable entity in a system
From the OS perspective:
- A task has a priority
- Has its own stack area
- And some housekeeping data (The Task Control Block/TCB)

From the schedulers perspective:
- A series of consecutive jobs

Seen from us:
- It boils down to a simple function with an endless loop

Scheduler


TASK #1
Stack
TASK #2
Stack
TASK #n
Stack
Task Control Block
Status
SP
Priority
Task Control Block
Status
SP
Priority
Task Control Block
Status
SP
Priority
MEMORY
CPU
CPU Registers
SP
Context

Tasks - Context switches

Stopping and saving one tasks state and letting another task run

What happens in a context switch?
- Save condition/state of task (to TCB)
- Choose next task
- Load condition/state of task (from TCB)
- Restores condition register/program status word (from TCB)
- Set Program counter to new task (from TCB)
- Set Stack pointer to new task (from TCB)

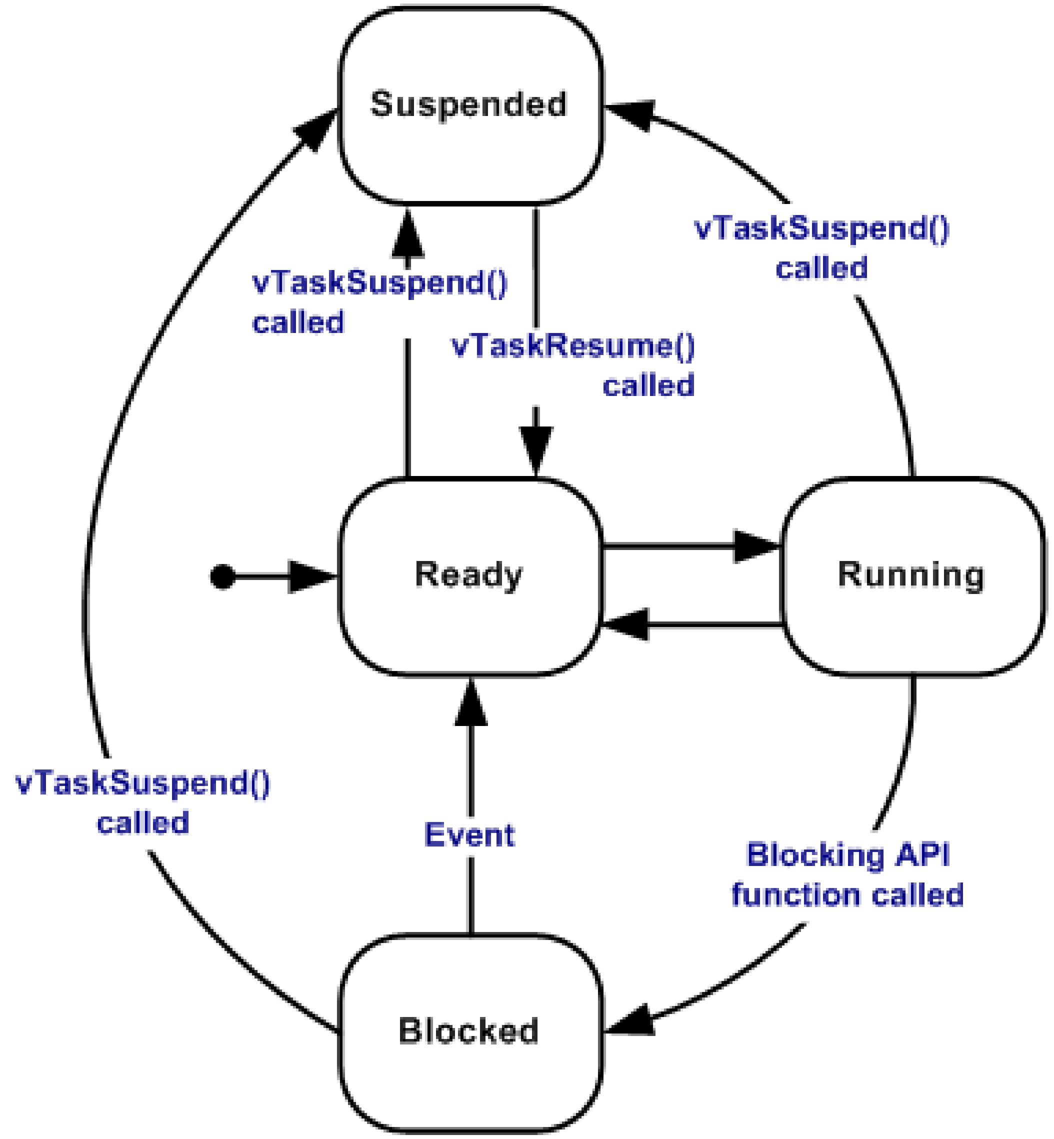
FreeRTOS

The FreeRTOS project was founded by Richard Barry. Richard graduated with 1st Class Honours in Computing for Real Time Systems. He's been directly involved in the start up of several companies, primarily working in the industrial automation and aerospace and simulation markets. Richard is currently a Principal Engineer at Amazon Web Services, owners and maintainers of the FreeRTOS project.

Developed in partnership with the world's leading chip companies over an 18-year period, and now downloaded every 170 seconds, FreeRTOS is a market-leading real-time operating system (RTOS) for microcontrollers and small microprocessors. Distributed freely under the MIT open source license, FreeRTOS includes a kernel and a growing set of IoT libraries suitable for use across all industry sectors. FreeRTOS is built with an emphasis on reliability and ease of use.

FreeRTOS has become the de facto standard RTOS for microcontrollers by removing common objections to using free software, and in so doing, providing a truly compelling free software model.

FreeRTOS – Task States



Running
- When a task is actually executing it is said to be in the Running state
- Has the CPU

Ready
- Ready tasks are those that are able to but are not currently executing because a different task of equal or higher priority is already in the Running state

Blocked
- A task is said to be in the Blocked state if it is currently waiting for either a temporal or external event
- Tasks in the Blocked state do not use any processing time and cannot be selected to enter the Running state

Suspended
- Tasks in the Suspended state only enter or exit the Suspended state when explicitly commanded to

Create Tasks in FreeRTOS

Signature:

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

Create Tasks in FreeRTOS

Globally declared:

```
#define task1_TASK_PRIORITY ( tskIDLE_PRIORITY + 5 )
TaskHandle_t x1Handle = NULL;
```

In main (or in some other task):

```
xTaskCreate(vTask1, "Task 1", configMINIMAL_STACK_SIZE, NULL,
            task1_TASK_PRIORITY, &x1Handle);
```

Start FreeRTOS

When tasks are created the scheduler must be started

In main create the tasks:

```
xTaskCreate(vTask1, "Task 1", configMINIMAL_STACK_SIZE,
NULL, task1_TASK_PRIORITY, &x1Handle);

// Create other tasks here

vTaskStartScheduler(); // Give control to the RTOS

while(1){;} // We should never come here !!!
```

A Task in FreeRTOS

```
void vTask1(void *pvParameters) {
     // Remove compiler warnings if pvParameters not used
     (void)pvParameters;

     while(1) {
          PORTB = 0xFE;
          vTaskDelay(500);  // 500 time ticks
     }
}
```

Let's Try it

1. Follow the instructions in the guide: *Setting up a simple FreeRTOS project in VS2022.pdf*
2. See if you can compile and run it

3. Take a look at the main file and try to understand what is going on
   1. Prepare questions if you find any

FreeRTOS documentation can be found here:
https://www.freertos.org/FreeRTOS-quick-start-guide.html

Exercise

Take a look at the FreeRTOS functions

- *vTaskDelay*
- *vTaskDelayUntil*

API: https://www.freertos.org/a00112.html

1. Try to incorporate these functions in the demo program
2. Implement two extra task in the program

Be sure that you understand the parameters for *xTaskCreate*

API: https://www.freertos.org/a00125.html