Advanced System Software
（先端システムソフトウェア）

#5 (2018/10/18)

CSC.T431, 2018-3Q
Mon/Thu 9:00-10:30, W832

Instructor: Takuo Watanabe（渡部卓雄）
Department of Computer Science

e-mail: takuo∅c.titech.ac.jp
http://www.psg.c.titech.ac.jp/~takuo/
ext: 3690, office: W8E-805

Using RTOS Features

- Multitasking
- Synchronization
- Inter-Task Communication
- Memory Management

FreeRTOS on ESP32

- FreeRTOS is an open-source RTOS designed for small-scale embedded systems.
- The ESP32 port of FreeRTOS is included in ESP-IDF as a component.
  - Currently based on FreeRTOS version 8.2
- The runtime system of a typical ESP32 application built with ESP-IDF uses FreeRTOS to manage tasks in the application.
  - An Arduino-based application also uses FreeRTOS because the Arduino-ESP32 depends on ESP-IDF.

FreeRTOS Tasks

- A FreeRTOS task corresponds to a thread in other OS
- FreeRTOS provides cooperative/preemptive multitasking
  - cooperative (aka non-preemptive) multitasking
    - Tasks should voluntarily (and periodically) yield their control using explicit context-switching API calls or other blocking operations.
  - preemptive multitasking
    - Context-switching is realized using timer interrupts and blocking operations. So tasks do not need to invoke explicit context-switching API calls.

Task Creation (in ESP32 FreeRTOS)

- xTaskCreatePinnedToCore
  - Creates and starts a new task with a specified affinity
    - Tasks can be created dynamically within other tasks.

- Task Affinity
  - Notion that specifies the place (core) in which a task runs
  - Core ID (0 or 1 in ESP32)
    - The created task runs on the specified core.
  - tskNO_AFFINITY
    - The created task is not pinned to any specific core.
    - The scheduler decides which core to run it.

Task Creation Example

```
void task1_fun(void *params) {
    for (;;) {
        ...
    }
}

void setup() {
    ...
    xTaskCreatePinnedToCore(
                    task1,     // task function
                    "task1",   // task name (for debugging)
                    4096,      // stack size
                    NULL,      // parameter
                    1,         // priority
                    NULL,      // task handle
                    0);        // task affinity (core number)
    ...
}

void loop() { ... }
```

Task Functions

- The behavior of a task is implemented as a function
  - **void** *ATaskFunction*(**void** **pvParameters*);
- The task function must not return.
- If a task is no longer required, it should be stopped and deleted explicitly from outside of it.

Task Handle (1/2)

- A datum (a pointer) that identifies a task.
- The handle of a task can be obtained as follows.

```
TaskHandle_t task1;
xTaskCreatePinnedToCore(
                task1_fun, // task function
                "task1",   // task name (for debugging)
                4096,      // stack size
                NULL,      // parameter
                1,         // priority
                &task1,    // task handle
                0);        // task affinity (core number)

delay(10000);              // 10 sec.

vTaskDelete(task1);        // stops and deletes task1
```

Task Handle (2/2)

- Examples of task API functions
  - eTaskGetState
  - uxTaskPriorityGet, vTaskPrioritySet
  - vTaskDelete
  - vTaskSuspend, vTaskResume

- In these functions, you may use NULL instead of a task handle to specify the current task.

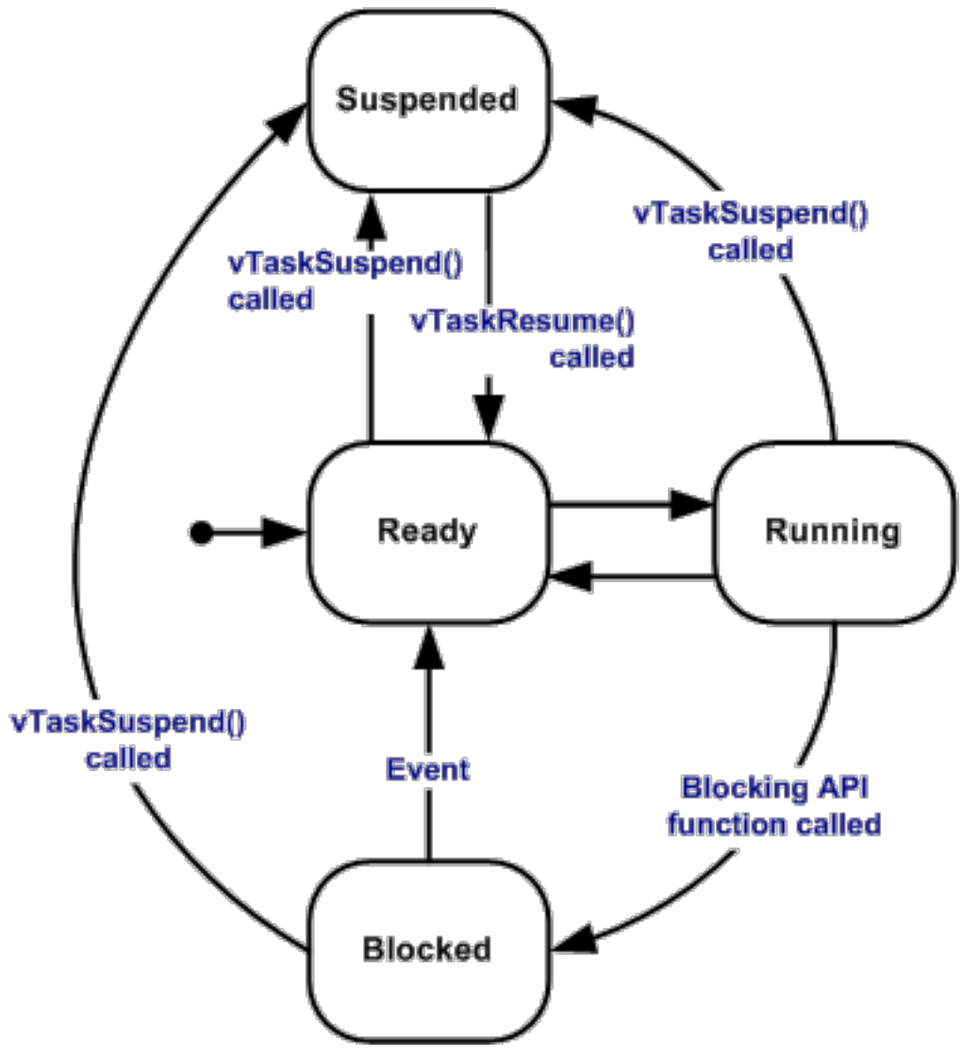
Tasks in an ESP32 Application

- An ESP32 application usually has at least one task to execute its main functionality
  - loop task (in Arduino based application)
  - main task (in Vanilla ESP-IDF application)
- In addition, the runtime system (based on ESP-IDF) has the following tasks:
  - IDLE0, IDLE1
  - Tmr Svc
  - esp_timer
  - ipc0, ipc1

The 'main' function in Arduino-ESP32

```
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "Arduino.h"

...

void loopTask(void *pvParameters) {
    setup();
    for (;;) {
        loop();
    }
}

extern "C" void app_main() {
    initArduino();
    xTaskCreatePinnedToCore(loopTask, "loopTask", 8192, NULL,
                            1, NULL, ARDUINO_RUNNING_CORE);
}
```

Task States



- Running
  - A CPU core is actually running the task
- Ready
  - The task is ready to run. But CPU is not assigned.
- Blocked
  - The task is stopped and is waiting for an event.
    - ex. delay
- Suspended
  - The task is stopped and is waiting to be resumed.

Blocked vs. Suspended

- Blocked
  - A task becomes Blocked state when it executes a blocking function such as vTaskDelay.
  - The blocked task resumes when a specific event occurs. For example, a task blocked with vTaskDelay resumes when the delay period has expired.
- Suspended
  - A task becomes Suspended state when another task (or itself) executes vTaskSuspend.
  - The suspended task cannot resume until another task explicitly invokes vTaskResume for the task.

delay

- Arduino function delay is implemented using FreeRTOS function vDelayTask.

```
void delay(uint32_t ms) {
    vTaskDelay(ms / portTICK_PERIOD_MS);
}
```

- `void vTaskDelay(const TickType_t ticks)`
  - ticks : # of ticks (# of timer interrupts) specifying delay period
    - portTICK_PERIOD_ MS: # of ticks per 1ms
    - portMAX_DELAY : maximum delay period
  - The task becomes Blocked state until the specified delay period expires.

delayMicroseconds

- In Arduino-ESP32 library, delayMicroseconds is implemented as a busy-waiting loop as follows.

```
void delayMicroseconds(uint32_t us) {
    uint32_t m = micros();
    if (us) {
        uint32_t e = m + us;
        if (m > e) //overflow
            while (micros() > e) NOP();
        while (micros() < e) NOP();
    }
}
```

- Unlike delay, delayMicroseconds consumes CPU time. The task does not become Blocked state.

Priority

- FreeRTOS provides priority based scheduling
  - The scheduler selects a task with the highest priority from the ready queue.
- Priority:
  - 0 (lowest) - configMAX_PRIORITIES - 1 (highest)
    - configMAX_PRIORITIES = 25 (in ESP32)

Inter-Task Synchronization

- FreeRTOS provides the following inter-task synchronization APIs
  - Semaphore
    - binary semaphore
    - counting semaphore
  - Mutex
  - Task Notification

Binary Semaphore

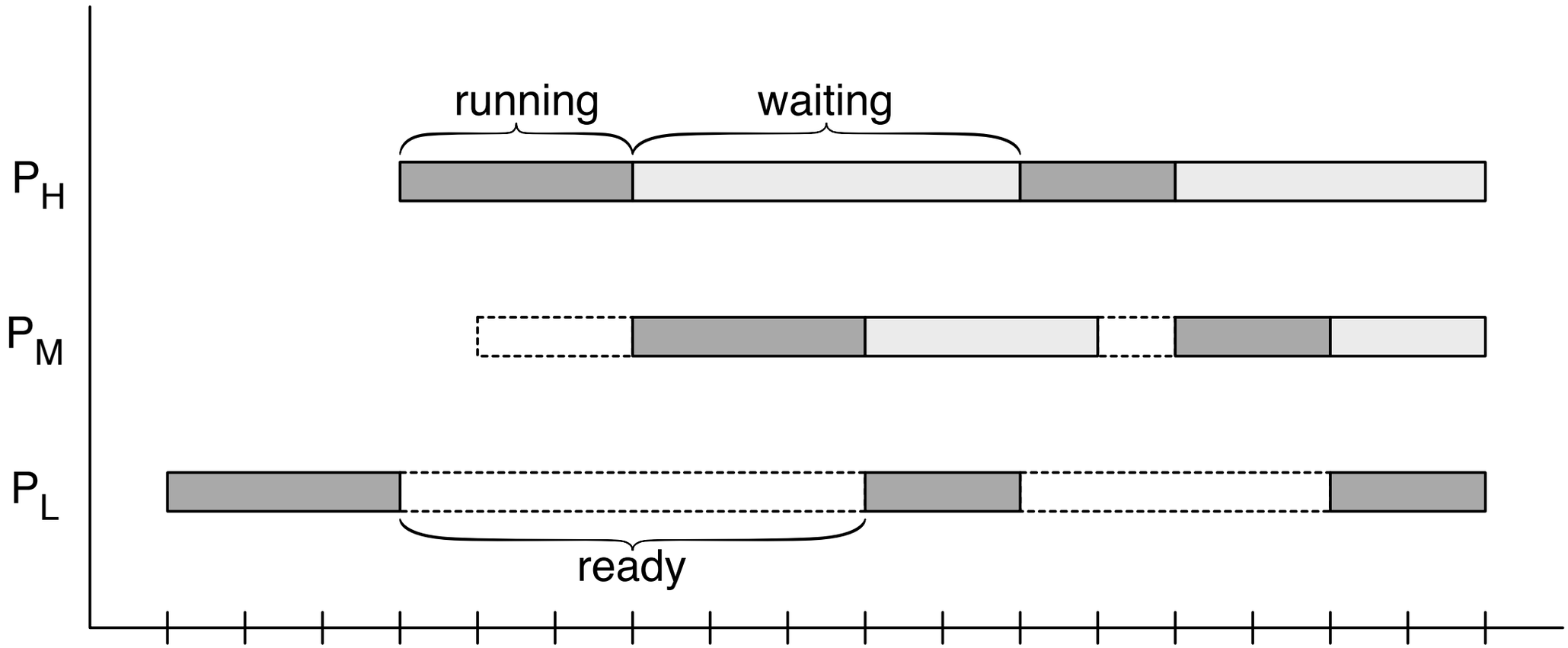
```
SemaphoreHandle_t sem = xSemaphoreCreateBinary();
```

```
TickType_t timeout = 60000;  // 1min. (in ESP32)

if (xSemaphoreTake(sem, timeout)) {
  // The task successfully obtains the semaphore

  // do something with the shared resource

  // Releases the semaphore
  xSemaphoreGive(sem);
}
else {
  // could not obtain the semaphore within timeout
}
```

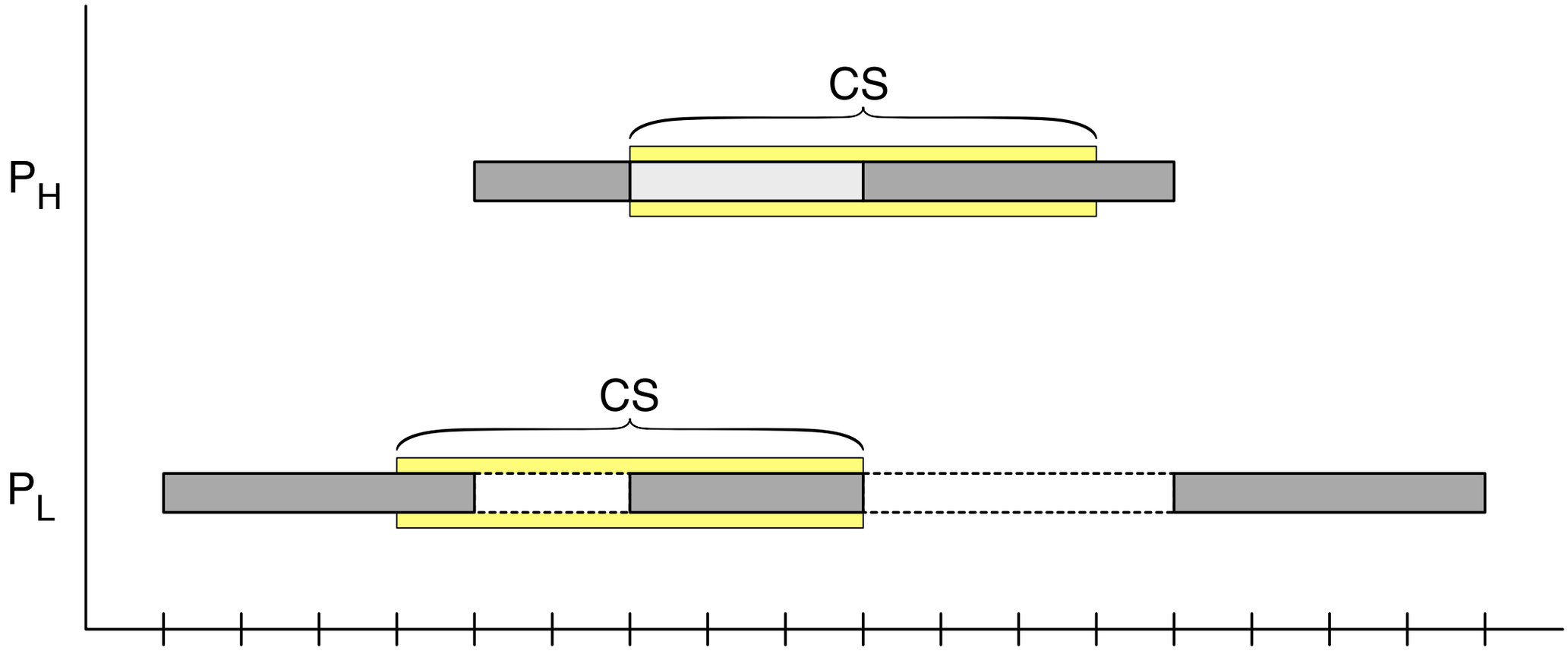
Mutex

- xSemaphoreCreateMutex
  - You may create a mutex instead of a binary semaphore
- Mutex vs. Binary Semaphore
  - Mutexes include a priority inheritance mechanism, but binary semaphores do not.
  - Usage
    - binary semaphores: inter-task synchronization
    - mutexes: mutual exclusion

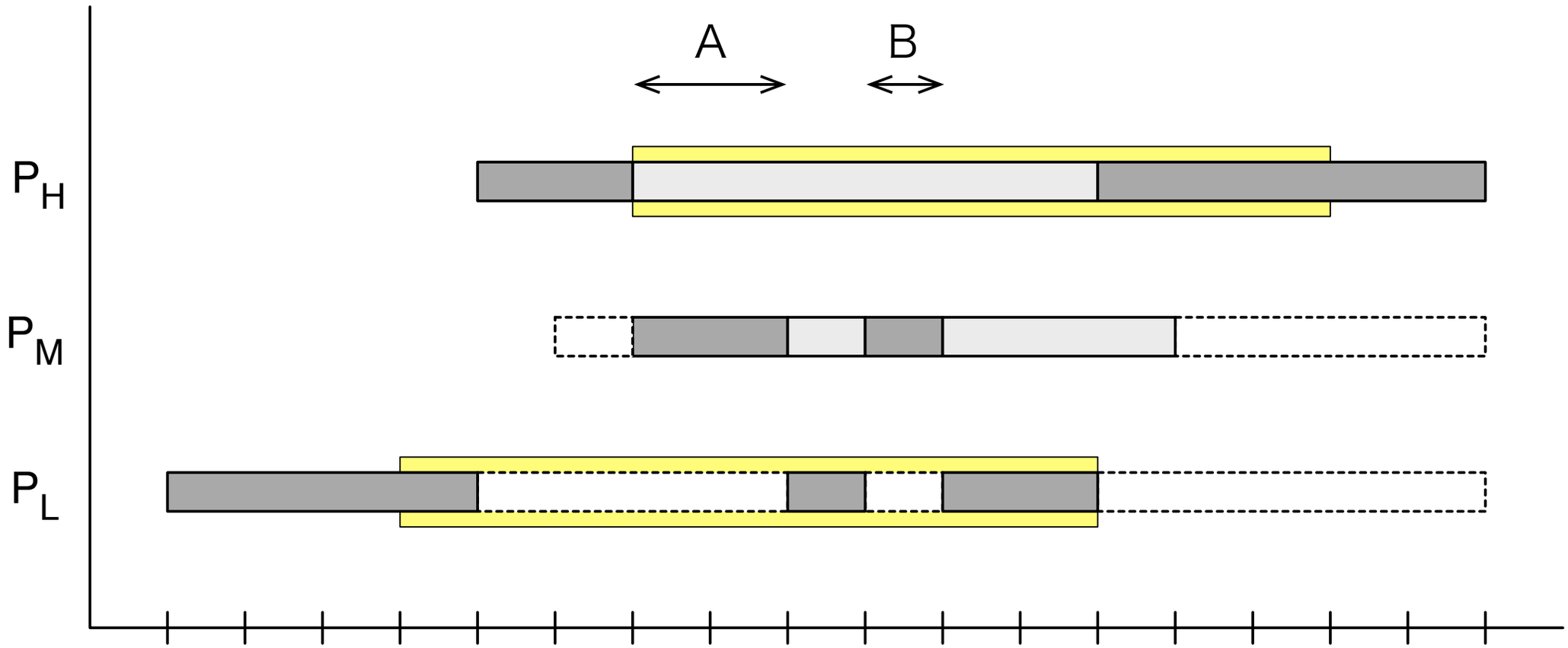
Priority-Based Scheduling



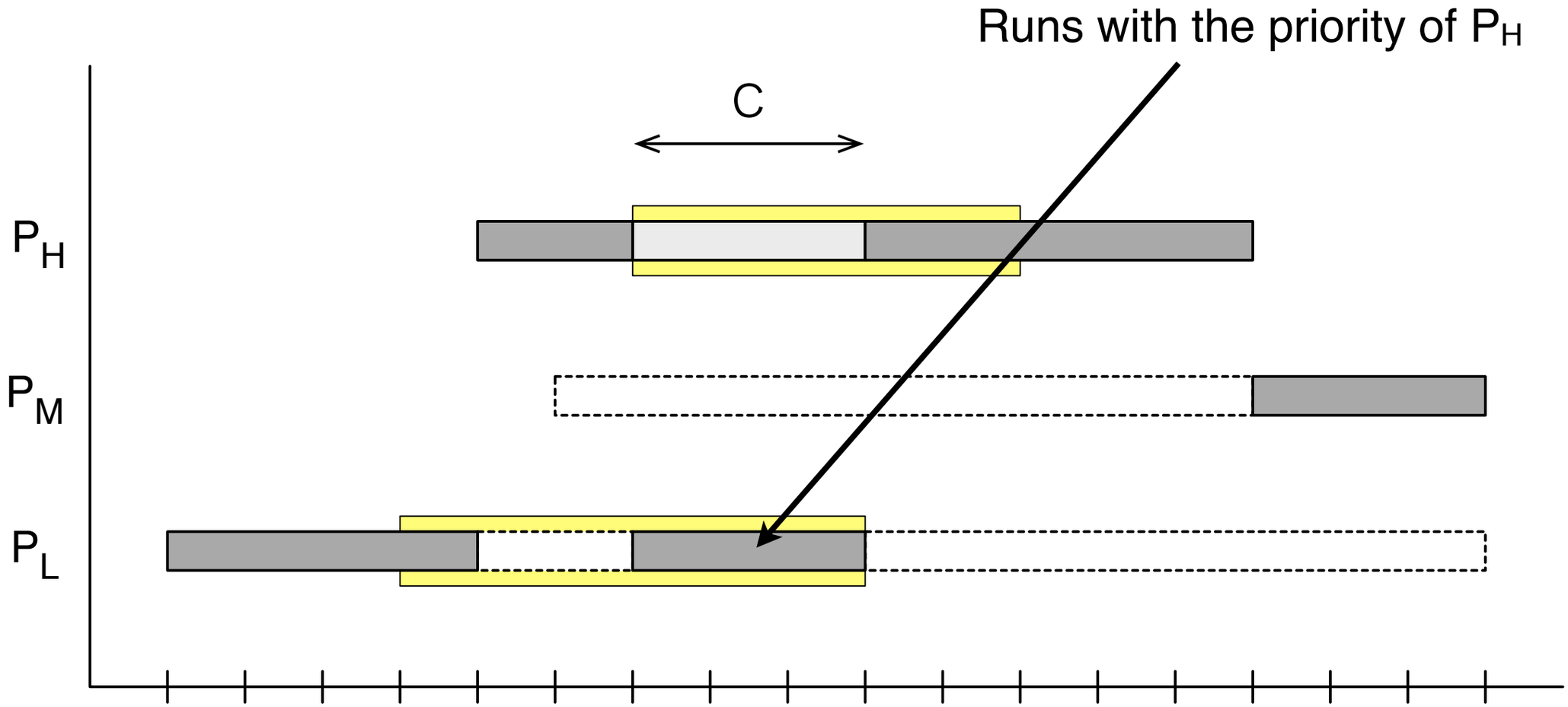
Critical Section



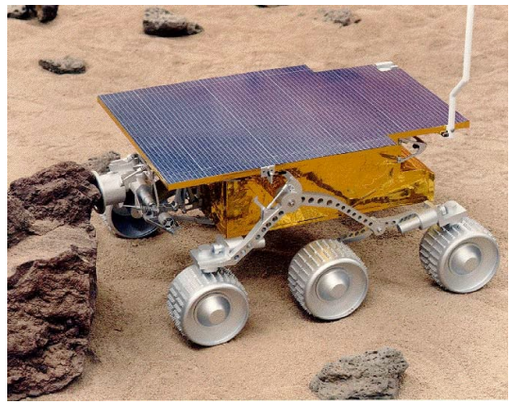
Priority Inversion



Priority Inheritance



The Sojourner Rover (Mars Pathfinder) Incident (Jul., 1997)



- The mission of the rover landed on Mars was to gather meteorological data.
- But it experiences repeated resets after starting the gathering task.
- The resets were issued by the watchdog timer.
- Timing overruns caused by priority inversion.

- See "What really happened on Mars?"
  - http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/Mars_Pathfinder.html

Task Notification

- FreeRTOS provides a mechanism for task notification
  - Available from FreeRTOS 8.2
- More efficient than binary semaphore/mutex

```ino
#include <Arduino.h>

void task1_fun(void *param) {
    uint32_t nv;
    for (;;) {
        if (xTaskNotifyWait(0, 0, &nv, 10000 / portTICK_PERIOD_MS))
            Serial.printf("task1: received %d\n", nv);
        else
            Serial.printf("task1: timeout\n");
    }
}

TaskHandle_t task1;

void setup() {
    Serial.begin(115200);
    xTaskCreatePinnedToCore(task1_fun, "task1", 4096, NULL, 1, &task1, 0);
}

void loop() {
    static uint32_t count = 0;
    xTaskNotify(task1, count++, eSetValueWithoutOverwrite);
    delay(1000);
}
```