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

#4 (2018/10/15)

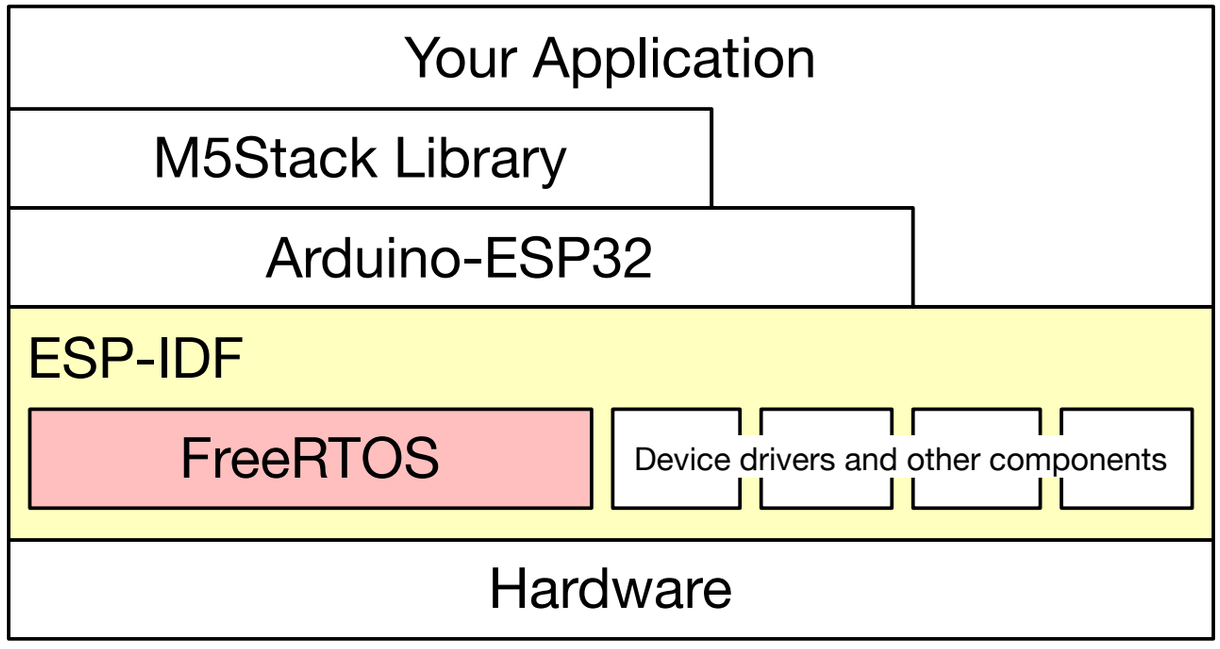
CSC.T431, 2018-3Q
Mon/Thu 9:00-10:30, W832

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

Programming M5Stack using FreeRTOS

- ESP-IDF : Framework for ESP32
- FreeRTOS

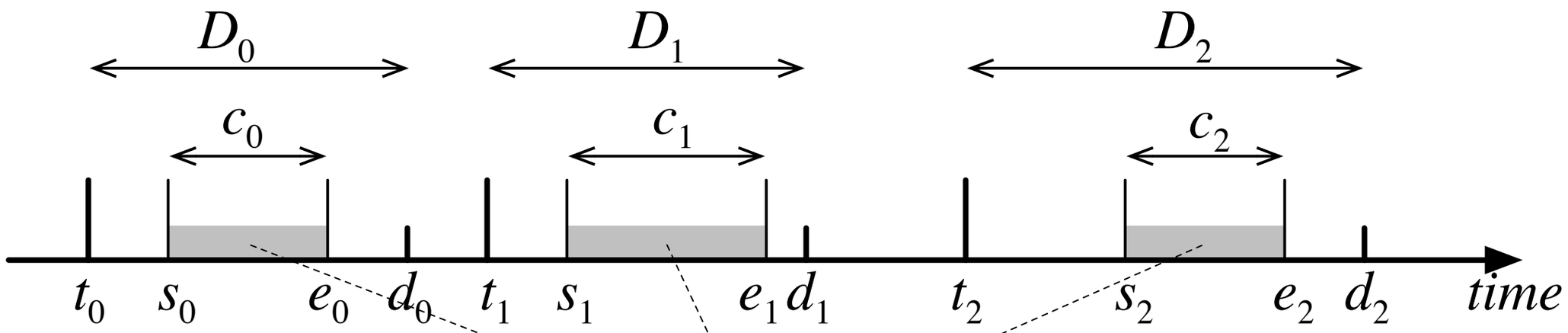


FreeRTOS

- Open-Source RTOS (Real-Time Operating System) kernel for embedded devices
  - https://freertos.org
  - Recently acquired by Amazon
- Simple & Small
  - Basic features: tasks, semaphores, timers
  - Simple memory management capabilities
  - Priority-based preemptive scheduling
    - Unlike common OSs (such as Linux), no advanced memory management systems (such as VM), no file systems, no user accounting systems are provided

Real-Time Task

task is divided into different task instances which have different deadlines



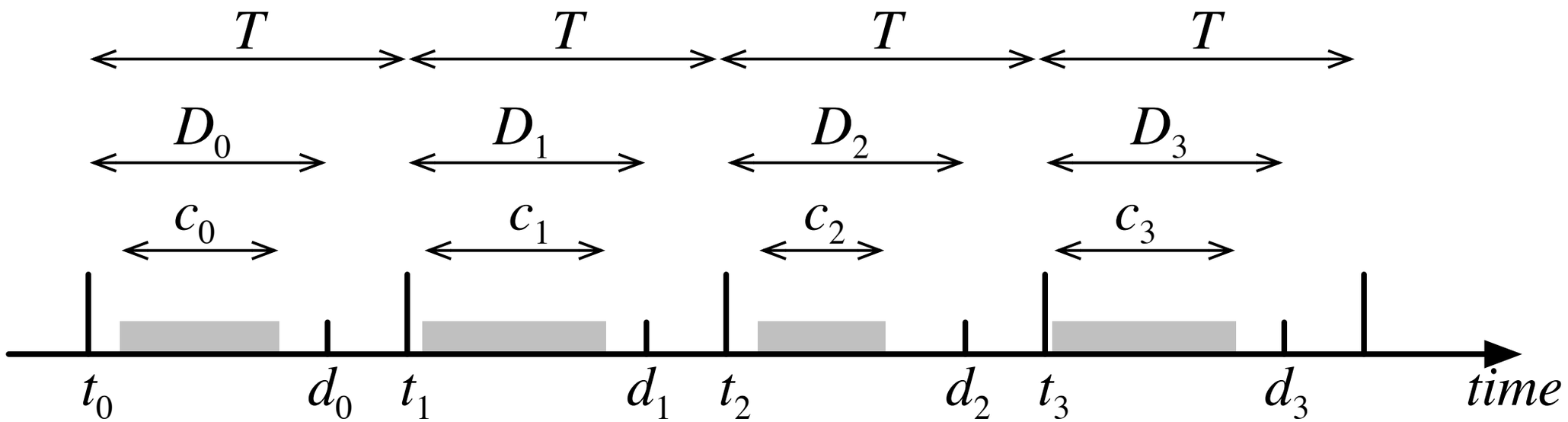
task instances (jobs)

- t_i release (arrival) time of i-th task instance that corresponds to i-th event
- s_i start time of i-th task instance
- e_i end time of i-th task instance
- d_i absolute deadline of i-th task instance
- D_i relative deadline of i-th task instance ($D_i = d_i - t_i$)
- c_i (worst case) execution time of i-th task instance ($c_i \leq D_i$)

Real-time Tasks

- Periodic Task
  - A task consists of a sequence of similar (or identical) jobs that are arrived at a constant rate.
  - e.g. sensor value acquisition, playing videos
- Aperiodic Task
  - A task consists of a sequence of jobs that are arrived at irregular intervals.
  - e.g. user activities click mouse/ type keyboard
- Sporadic Task
  - An aperiodic task characterized by a minimum inter-arrival time between consecutive activities.
  - e.g. network packets

Periodic Task (1)



T period ($T = t_{i+1} - t_i$)

ϕ phase ($= t_0$)

t_i release time of i-th task instance ($t_i = \phi + iT$)

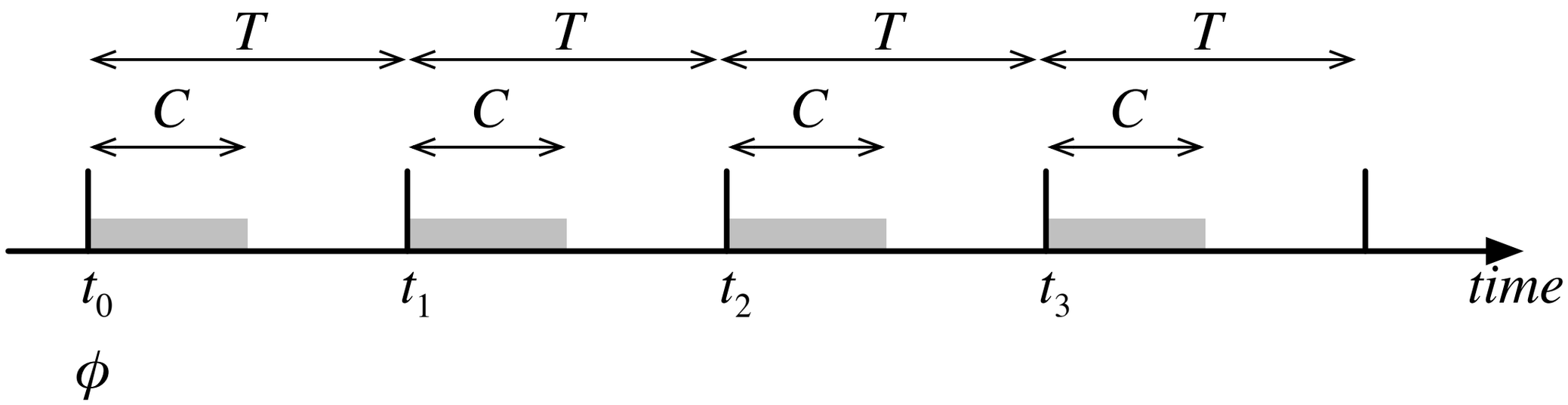
d_i absolute deadline of i-th task instance

D_i relative deadline of i-th task instance ($D_i = d_i - t_i$)

c_i (worst case) execution time of i-th task instance ($c_i \leq D_i$)

Periodic Task (2)

- To make things simpler, we assume that
  - $\forall i \in \boldsymbol{N}.\ D_i = T$,
  - $\forall i \in \boldsymbol{N}.\ s_i = t_i$, and
  - $\forall i \in \boldsymbol{N}.\ c_i = C$.
- Thus we can describe a periodic task τ as a triple (T, C, ϕ).



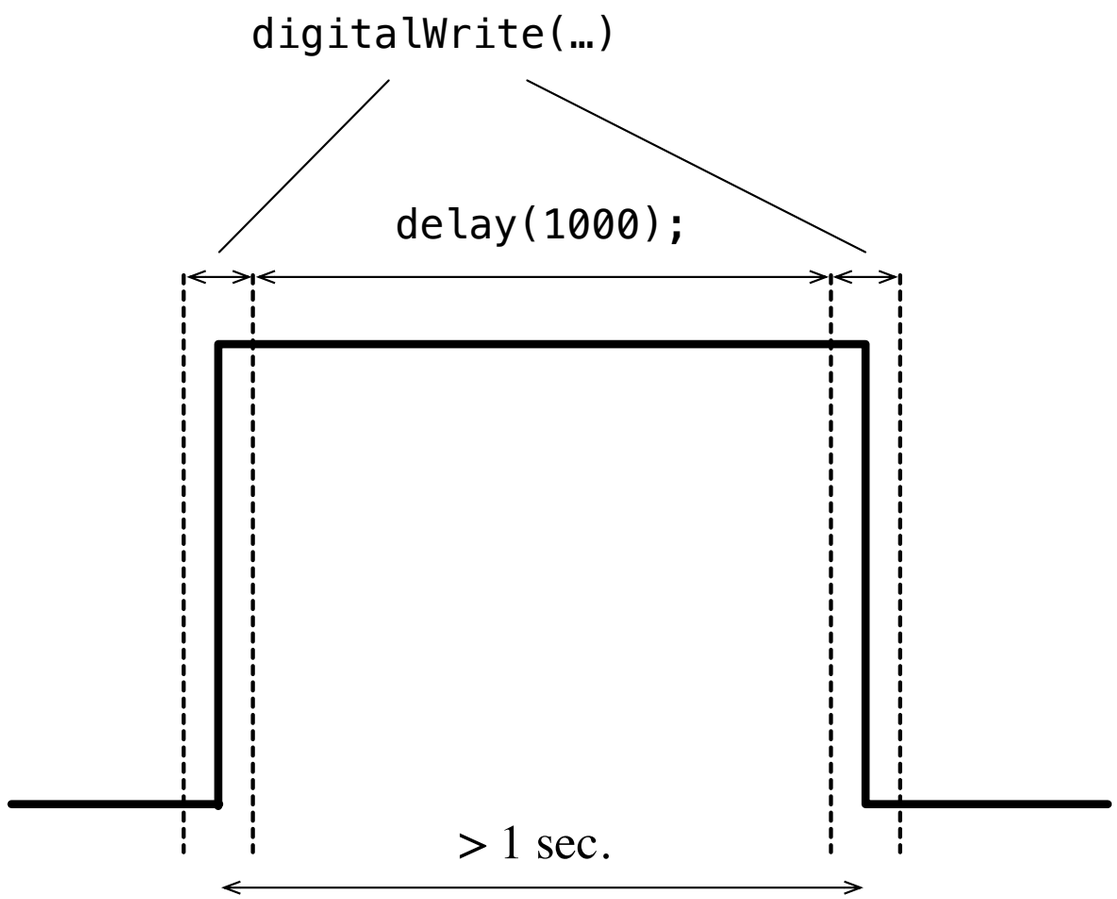
Programming Periodic Tasks

- T: Toggle an LED at a 1 second interval.
- S: Its easy.

```
#define LED_PIN 21

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_PIN, !digitalRead(LED_PIN));
  delay(1000);
}
```

Programming Periodic Tasks

- T: Is it really a 1 second interval?
- S: Well ... almost?



Programming Periodic Tasks

- T: Yes. The function digitalRead/Write are simple and sufficiently fast. But if you insert some work like the following, the resulting interval is longer than 1 second.

```
void loop() {
  digitalWrite(LED_PIN, !digitalRead(LED_PIN));
  work_in_less_than_1s();
  delay(1000);
}
```

Programming Periodic Tasks

- T: Toggle an LED at an exactly 1 second interval, but with *`work_in_less_than_1s`*`()`.
- S: Hmm... Here it is.

```
unsigned long target;

void setup() {
    pinMode(LED_PIN, OUTPUT);
    target = millis();
}

void loop() {
    if (millis() > target) {
        target = millis() + 1000;
        digitalWrite(LED_PIN, !digitalRead(LED_PIN));
        work_in_less_than_1s();
    }
}
```

A well known Arduino function is delay() which pauses the program for an amount of milliseconds specified as parameter.

millis(), on the other hand, is a function that returns the amount of milliseconds that have passed since program start.

https://www.norwegiancreations.com/2017/09/arduino-tutorial-using-millis-instead-of-delay/

Programming Periodic Tasks

- T: OK. Can you identify any drawbacks of your last solution?
- S: The CPU is too busy while idle running. It is possible to put a *pause time* like the following. But this may lower the accuracy of the interval.

```
void loop() {
    if (millis() > target) {
        target = millis() + 1000;
        digitalWrite(LED_PIN, !digitalRead(LED_PIN));
        work_in_less_than_1s();
    }
    delay(10);
}
```

Programming Periodic Tasks

- T: Do you think you can show another solution?
- S: (Google, Google, ...) OK. It works!

```
TickType_t lastWakeTime;

void setup() {
    pinMode(LED_PIN, OUTPUT);
    lastWakeTime = xTaskGetTickCount();
}

void loop() {
    vTaskDelayUntil(&lastWakeTime, 1000 / portTICK_PERIOD_MS);
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    work_in_less_than_1s();
}
```

returns tick number since the programme starts

wake up at this time

tick numbers that finish the delays

delay until these two args add up

Programming Periodic Tasks

- T: (Hmm... How did S find FreeRTOS API?) OK. Now, can you program two periodic tasks with different intervals in a single loop?
  - ex) Toggle an LED at a 1 second interval and output "Hello" to the USB-serial port at 3 second interval.
- S: Well .... I think I could (in the next slide)

```ino
#include <Arduino.h>
#define LED_PIN 21

unsigned long last_a;
unsigned long last_b;

void setup() {
    pinMode(LED_PIN, OUTPUT);
    Serial.begin(115200);
    last_a = last_b = millis();
}

void loop() {
    unsigned long curr = millis();
    if (curr > last_a + 1000) {
        last_a = curr;
        digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    }
    if (curr > last_b + 3000) {
        last_b = curr;
        Serial.println("Hello");
    }
}
```

Programming Periodic Tasks

- T: Hmm. Good.
- T: Can you do the same thing using vTaskDelayUntil ?
- S: ...
- T: Now, Write a program that can start several periodic tasks at runtime. Can you add some aperiodic tasks to your code? And, can you ...
- S: ....

MultiTask (in M5Stack Example)

```
void task1(void *pvParameters) {
    for (;;) {
        Serial.print("task1 Uptime (ms): ");
        Serial.println(millis());
        delay(100);
    }
}

void task2(void *pvParameters) {
    for (;;) {
        Serial.print("task2 Uptime (ms): ");
        Serial.println(millis());
        delay(200);
    }
}
void task3(void *pvParameters) {
    for (;;) {
        Serial.print("task3 Uptime (ms): ");
        Serial.println(millis());
        delay(1000);
    }
}
```

MultiTask (in M5Stack Example)

```
void setup() {
    // Task 1
    xTaskCreatePinnedToCore(
                    task1,     /* Function to implement the task */
                    "task1",   /* Name of the task */
                    4096,      /* Stack size in words */
                    NULL,      /* Task input parameter */
                    1,         /* Priority of the task */
                    NULL,      /* Task handle. */
                    0);        /* Core where the task should run */

    // Task 2
    xTaskCreatePinnedToCore( ... );

    // Task 3
    xTaskCreatePinnedToCore( ... );
}

void loop() { }
```

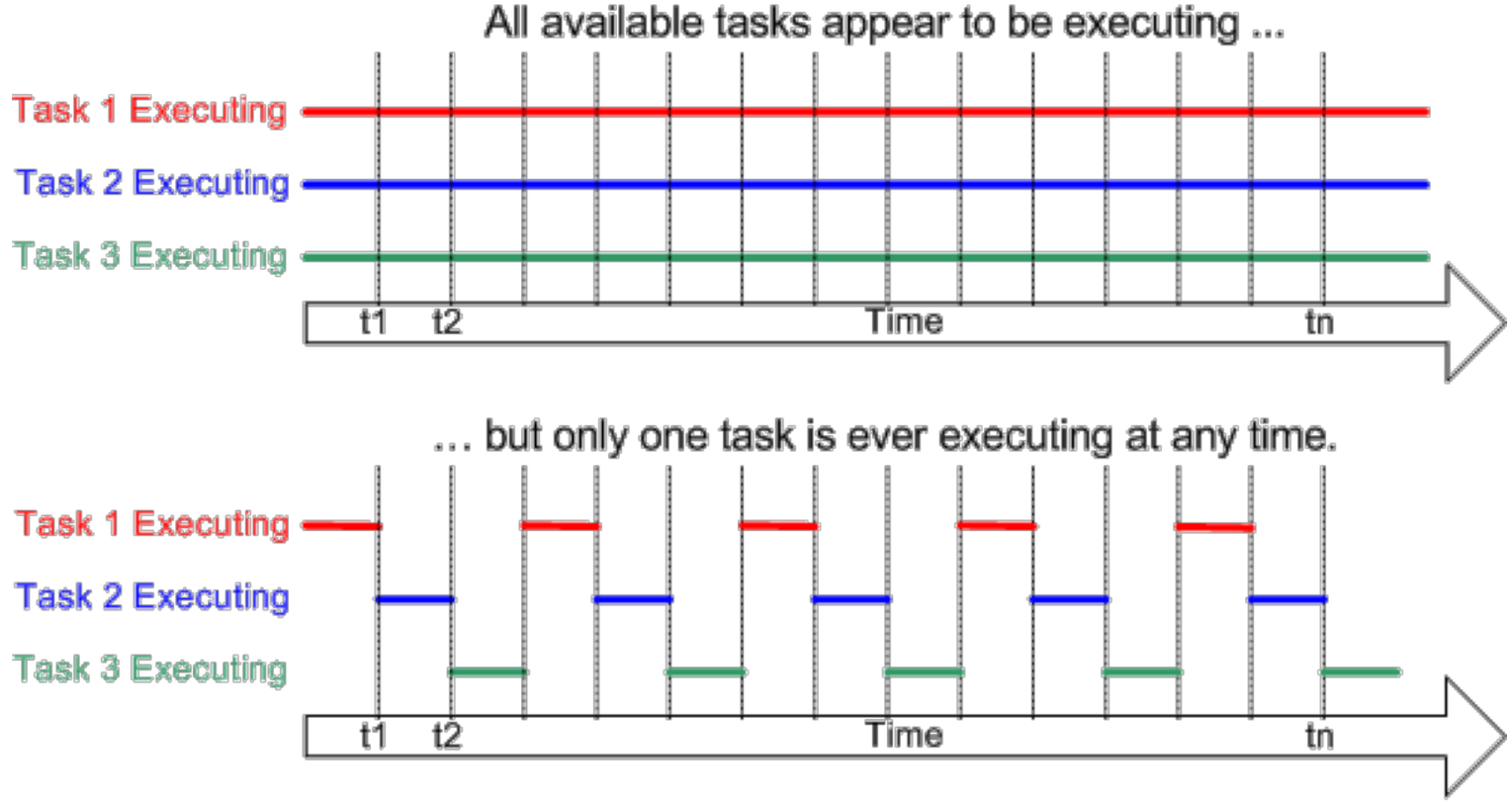
The 'main' function in Arduino-ESP32

```
#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);
}
```

FreeRTOS Tasks

- A task corresponds to a thread of control.
- FreeRTOS provides multitasking.

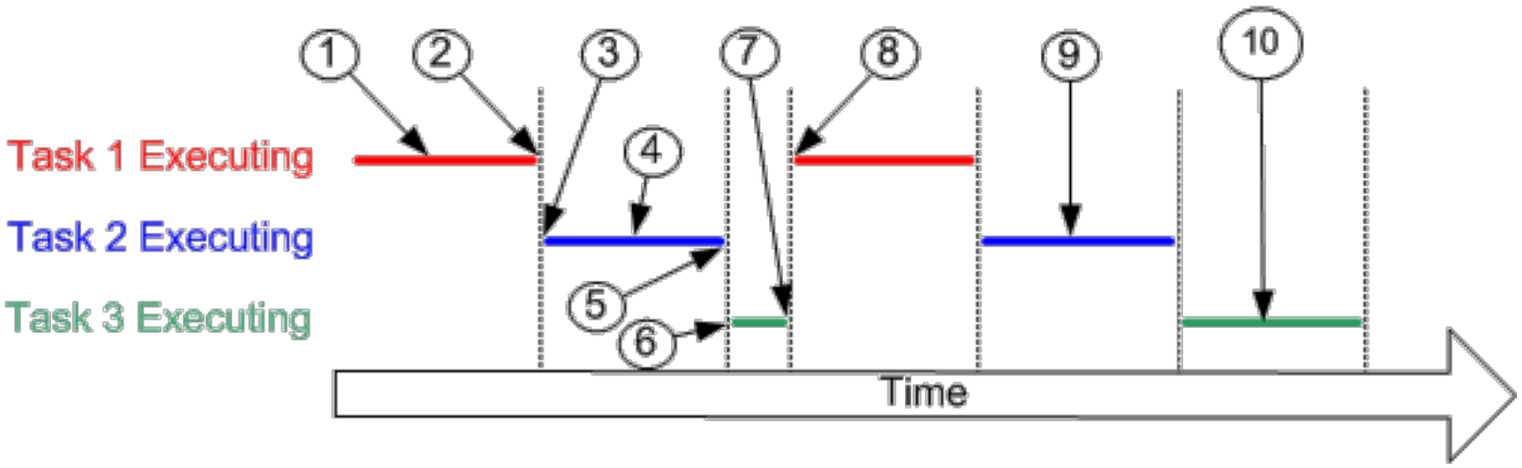
at any time task suspended will finally resume



https://www.freertos.org/implementation/a00004.html

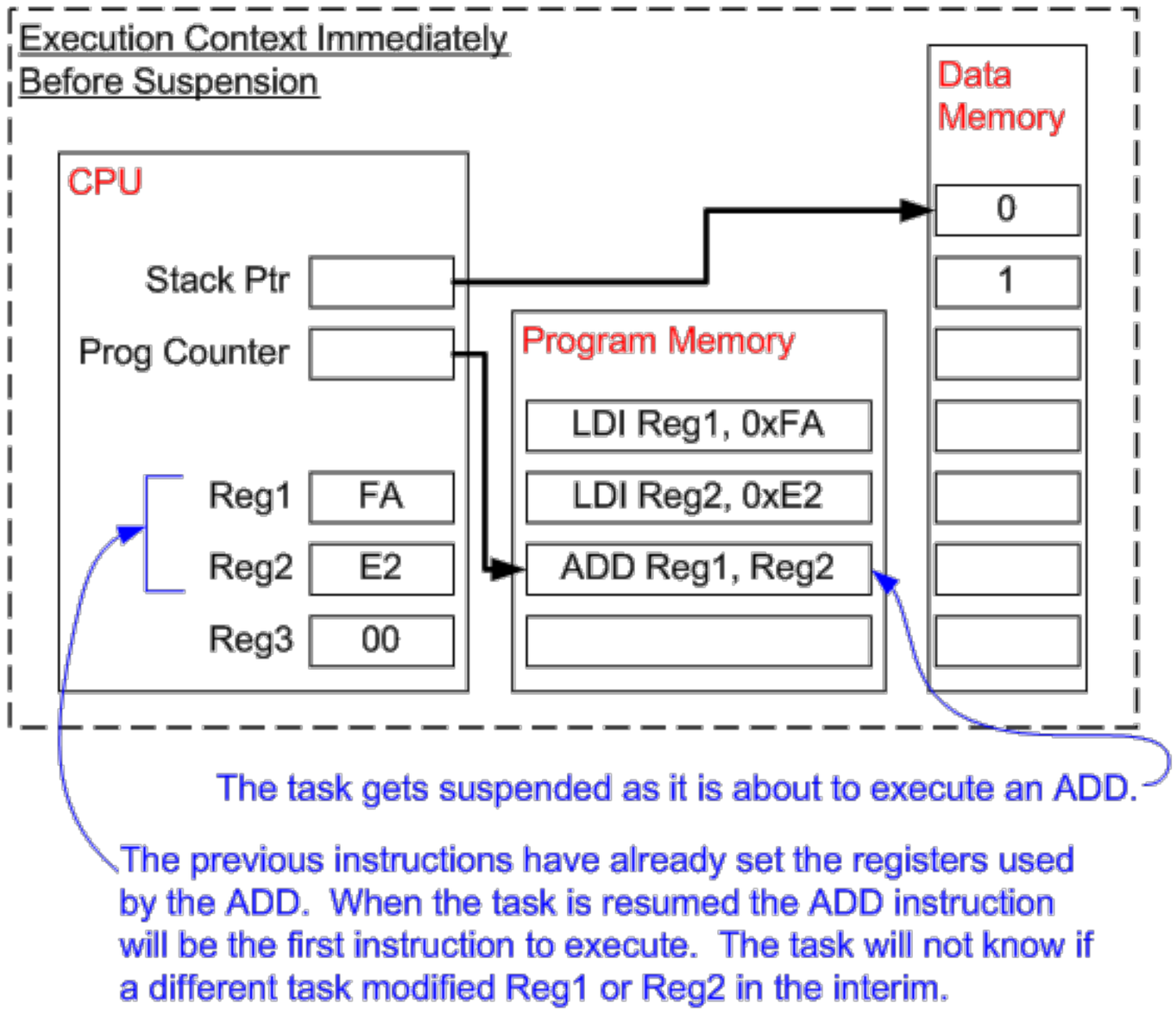
Scheduling

- The scheduler decides which task should be executing at any particular time.
- Scheduling policy: prioritized, fair



https://www.freertos.org/implementation/a00005.html

Context Switching



https://www.freertos.org/implementation/a00006.html

Ticks

- A tick corresponds to a timer event (an interruption event issued by the interval timer)