30-Day FreeRTOS Course for ESP32 Using ESP-IDF

(Day 5)

“Task States and Priorities”

Table of Contents

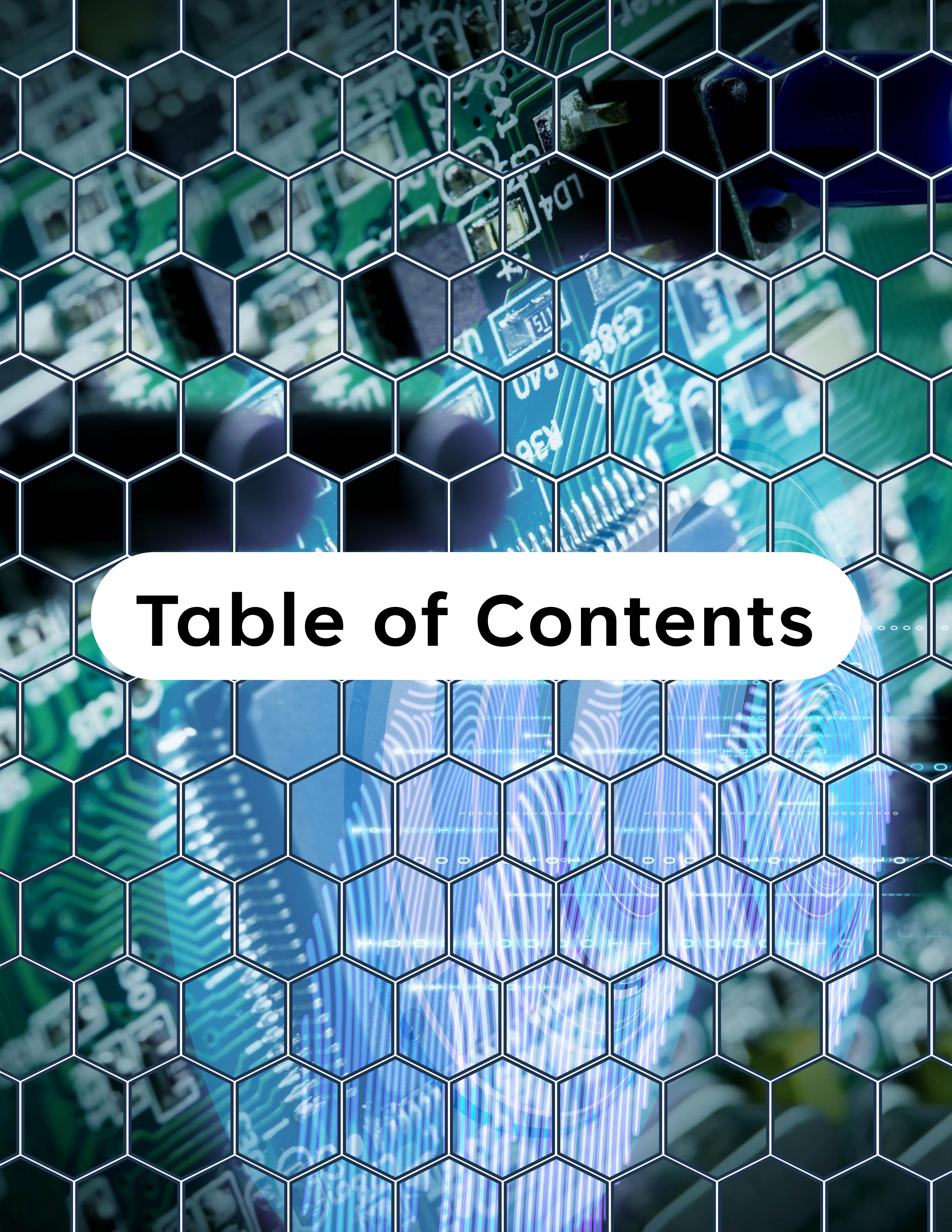
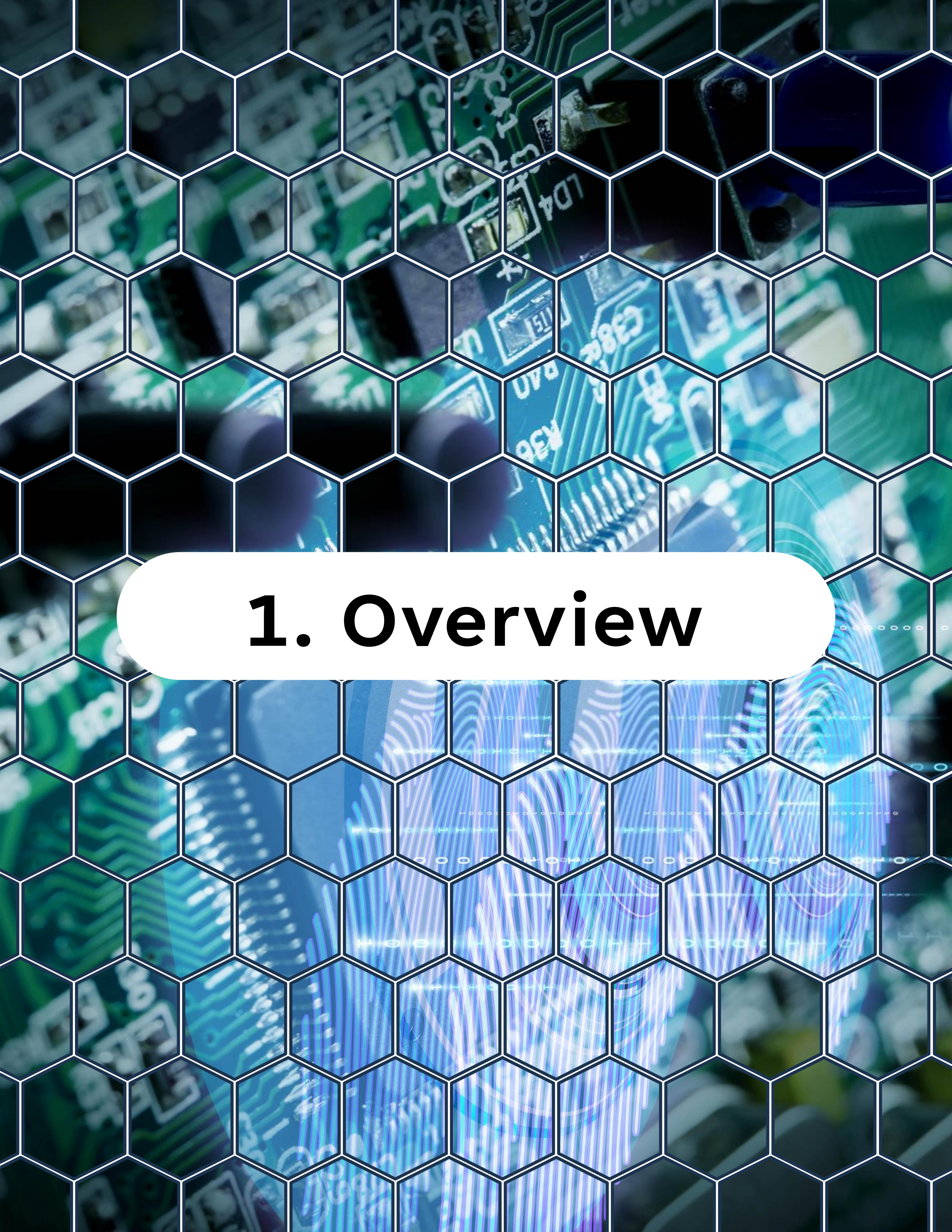


Table of Contents

1. Overview



1. Overview

On Day 5, you'll learn:

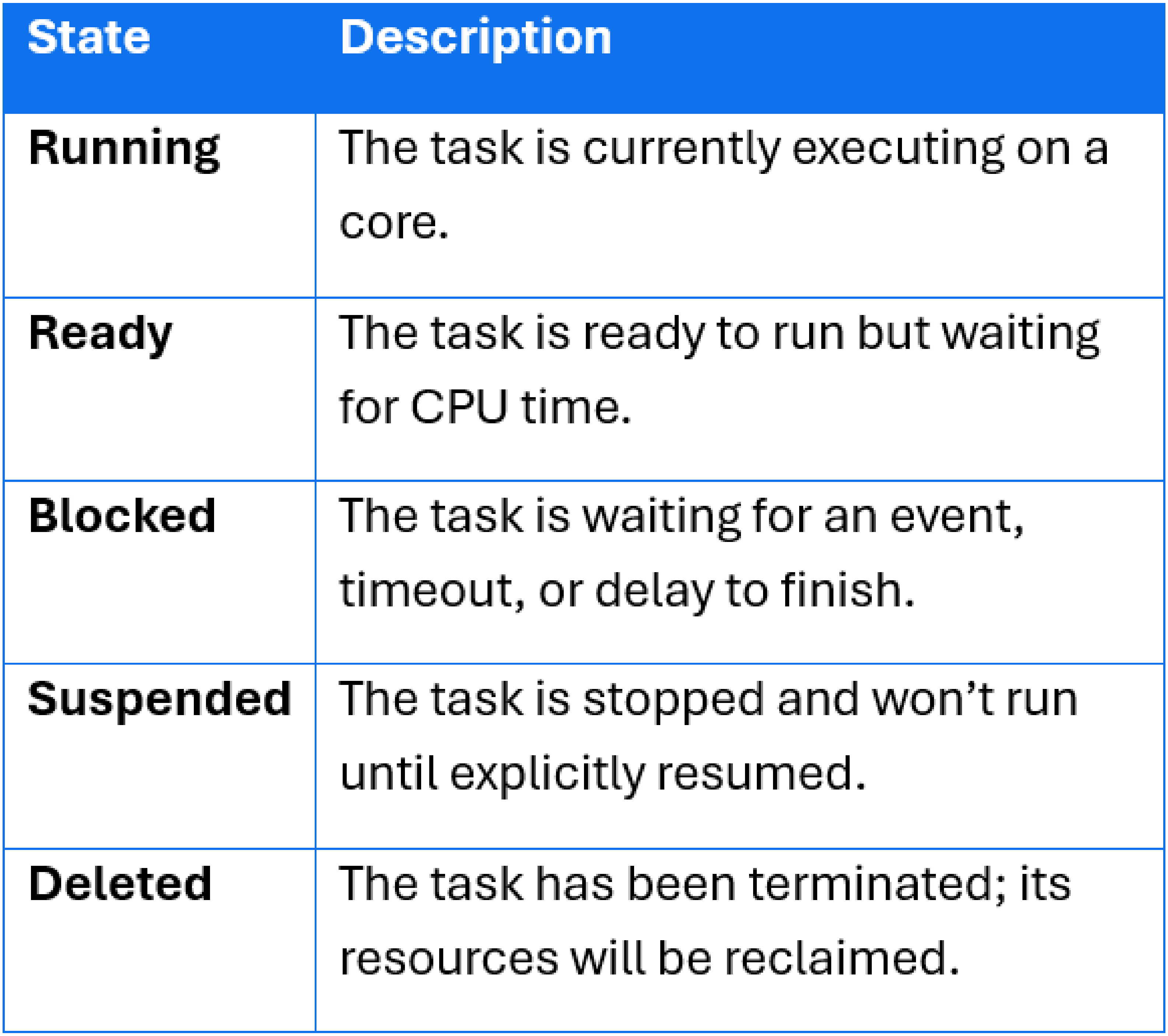
- The **lifecycle** of a FreeRTOS task
- The different **task states** and what they mean
- How priorities determine **which task runs next**
- How to change task priority dynamically
- Practical example of priority management

By the end of this lesson, you'll understand **how the scheduler decides which task runs**, and how to use priorities effectively for responsive systems.

2. Task Lifecycle in FreeRTOS

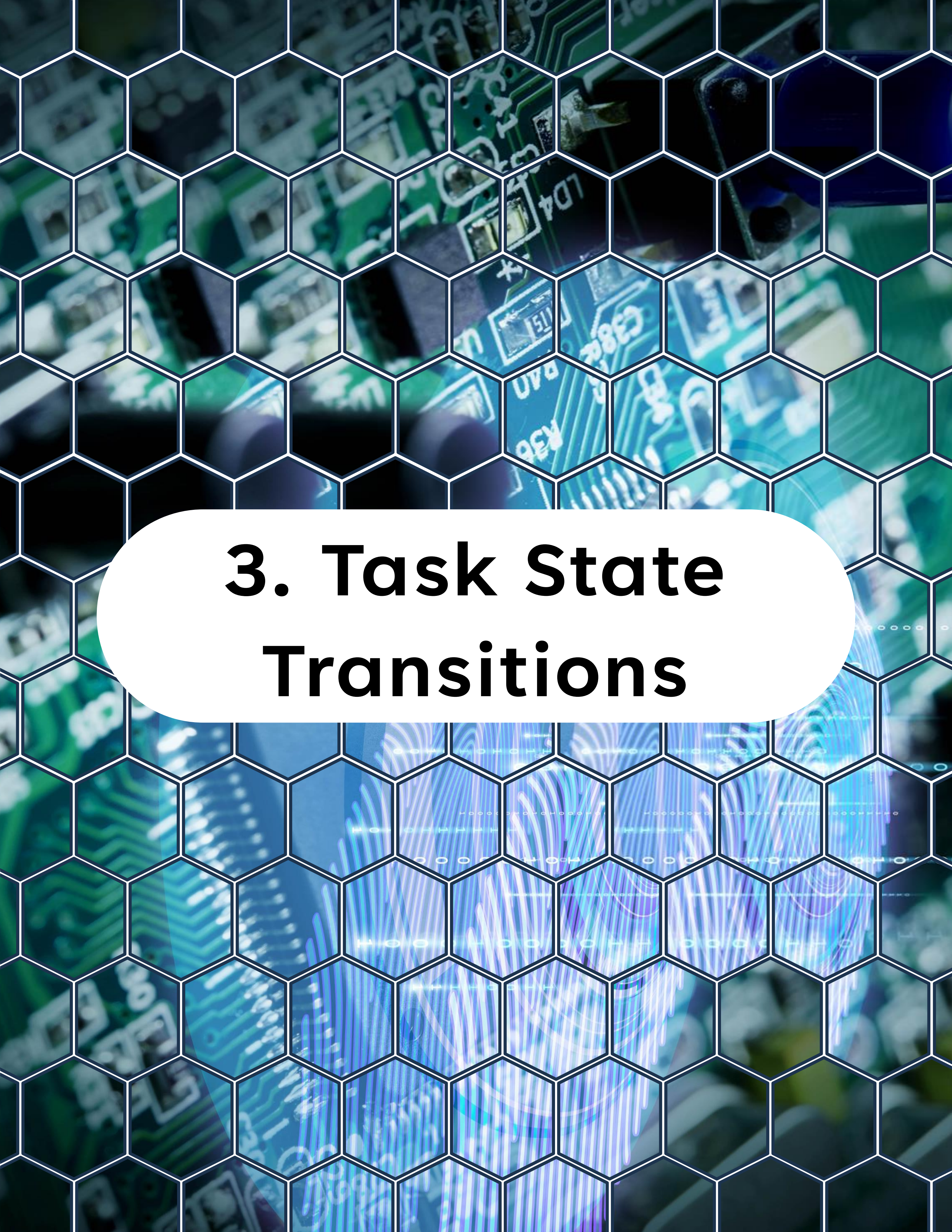
2. Task Lifecycle in FreeRTOS

In FreeRTOS, tasks can be in one of **five main states:**



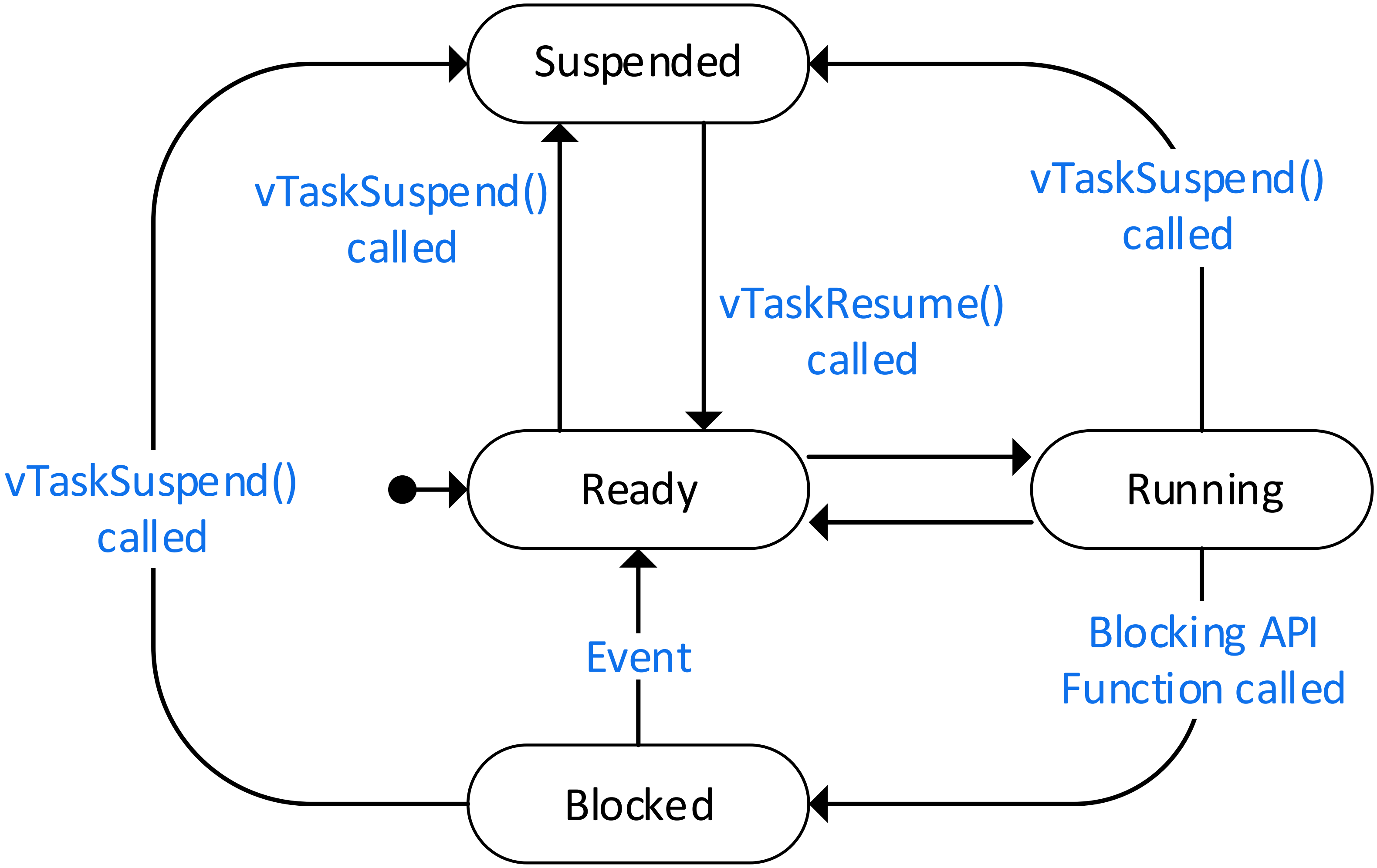
State	Description
Running	The task is currently executing on a core.
Ready	The task is ready to run but waiting for CPU time.
Blocked	The task is waiting for an event, timeout, or delay to finish.
Suspended	The task is stopped and won’t run until explicitly resumed.
Deleted	The task has been terminated; its resources will be reclaimed.

3. Task State Transitions



3. Task State Transitions

Here's how tasks move between states:



4. How Priorities Work

4. How Priorities Work

FreeRTOS uses **preemptive scheduling**:

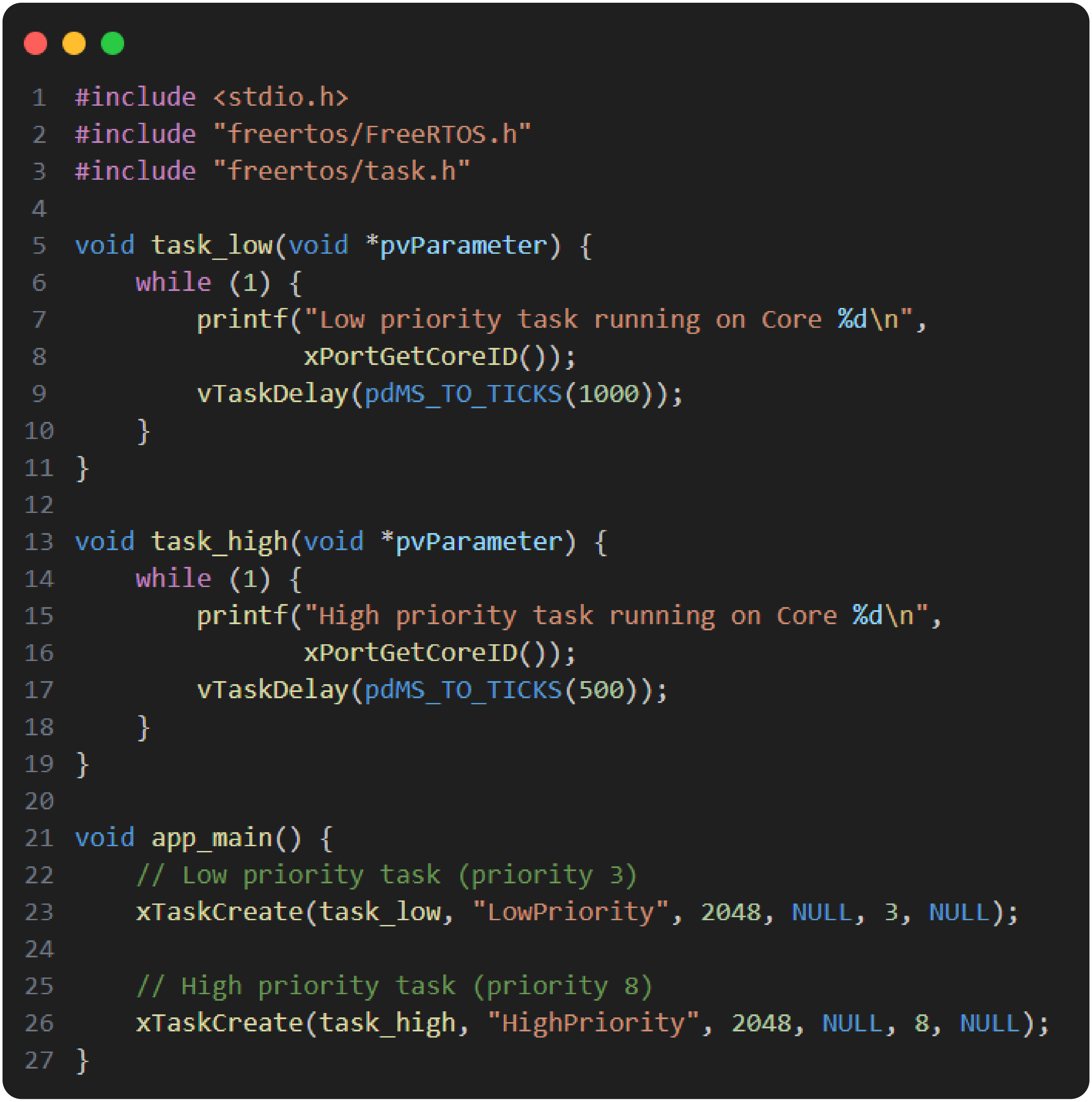
- **Higher priority tasks** always run before lower priority tasks.
- Tasks with **equal priority** share CPU time via **time slicing** (if enabled).
- Priority range in ESP-IDF defaults to:
  `0 (lowest) → configMAX_PRIORITIES - 1 (highest)`
  In ESP-IDF, `configMAX_PRIORITIES` is **25** by default.

📌 **Important:** A low-priority task may never run if a high-priority task never blocks or yields.

5. Example: Tasks with Different Priorities

5. Example: Tasks with Different Priorities

Code Example

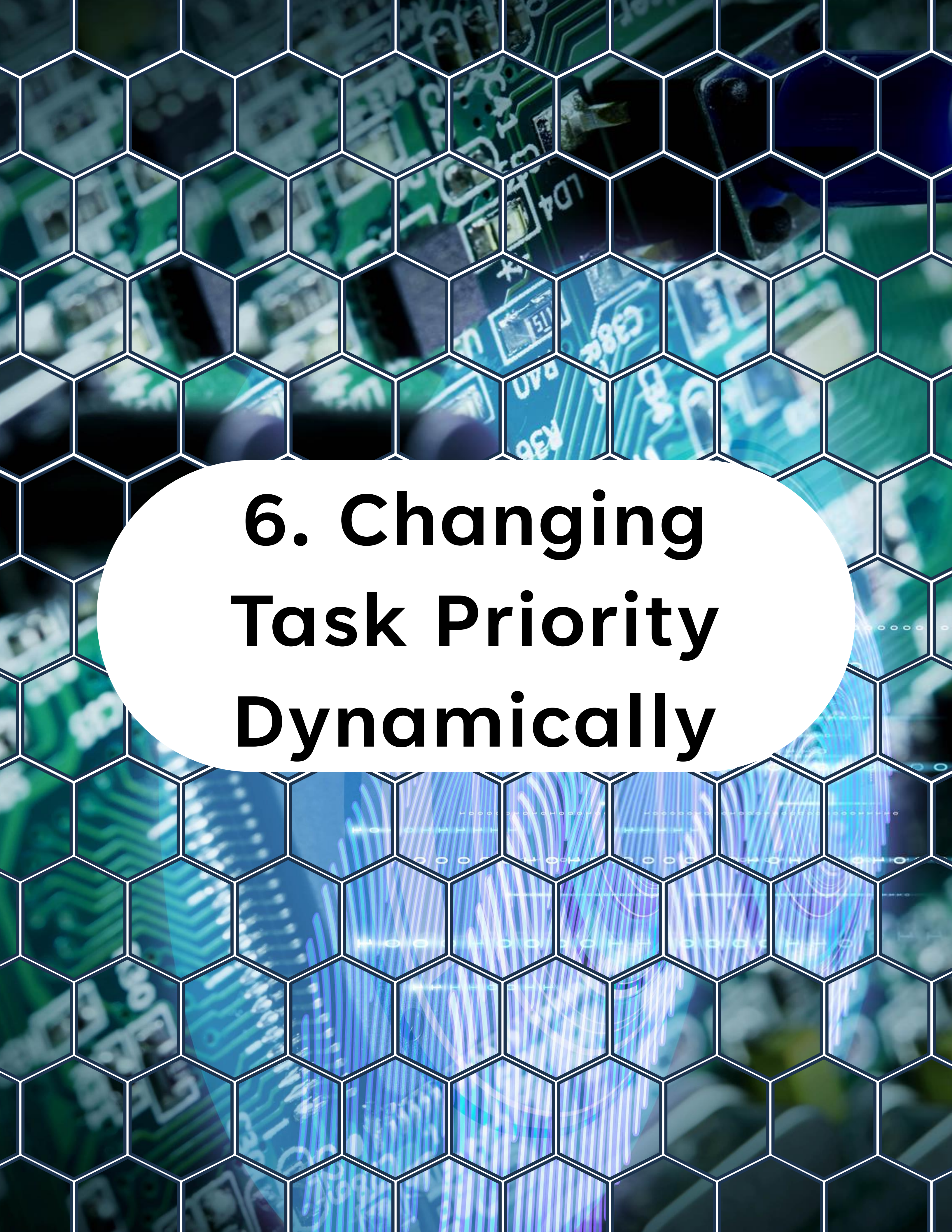


```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

void task_low(void *pvParameter) {
    while (1) {
        printf("Low priority task running on Core %d\n",
               xPortGetCoreID());
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

void task_high(void *pvParameter) {
    while (1) {
        printf("High priority task running on Core %d\n",
               xPortGetCoreID());
        vTaskDelay(pdMS_TO_TICKS(500));
    }
}

void app_main() {
    // Low priority task (priority 3)
    xTaskCreate(task_low, "LowPriority", 2048, NULL, 3, NULL);

    // High priority task (priority 8)
    xTaskCreate(task_high, "HighPriority", 2048, NULL, 8, NULL);
}
```

5. Example: Tasks with Different Priorities

Expected Behavior

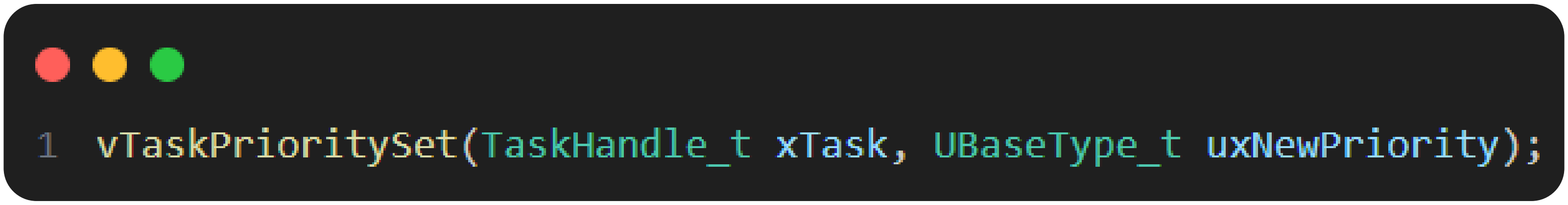
- The **high-priority task** will run more often because the scheduler always picks the highest-priority ready task.
- The low-priority task will run only when the high-priority task is **delayed or blocked**.



6. Changing Task Priority Dynamically

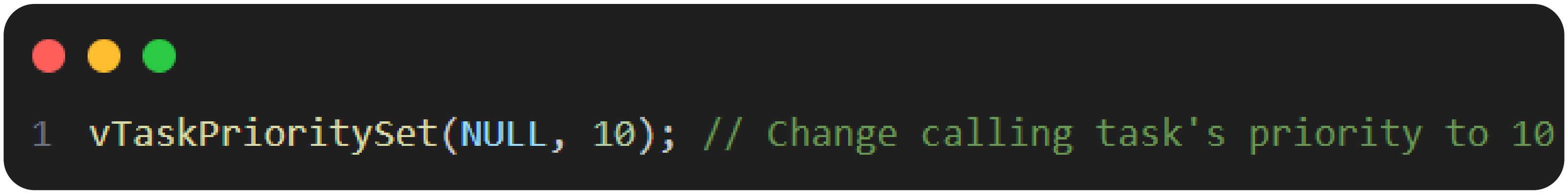
6. Changing Task Priority Dynamically

You can adjust a task’s priority during runtime using:



```c
vTaskPrioritySet(TaskHandle_t xTask, UBaseType_t uxNewPriority);
```

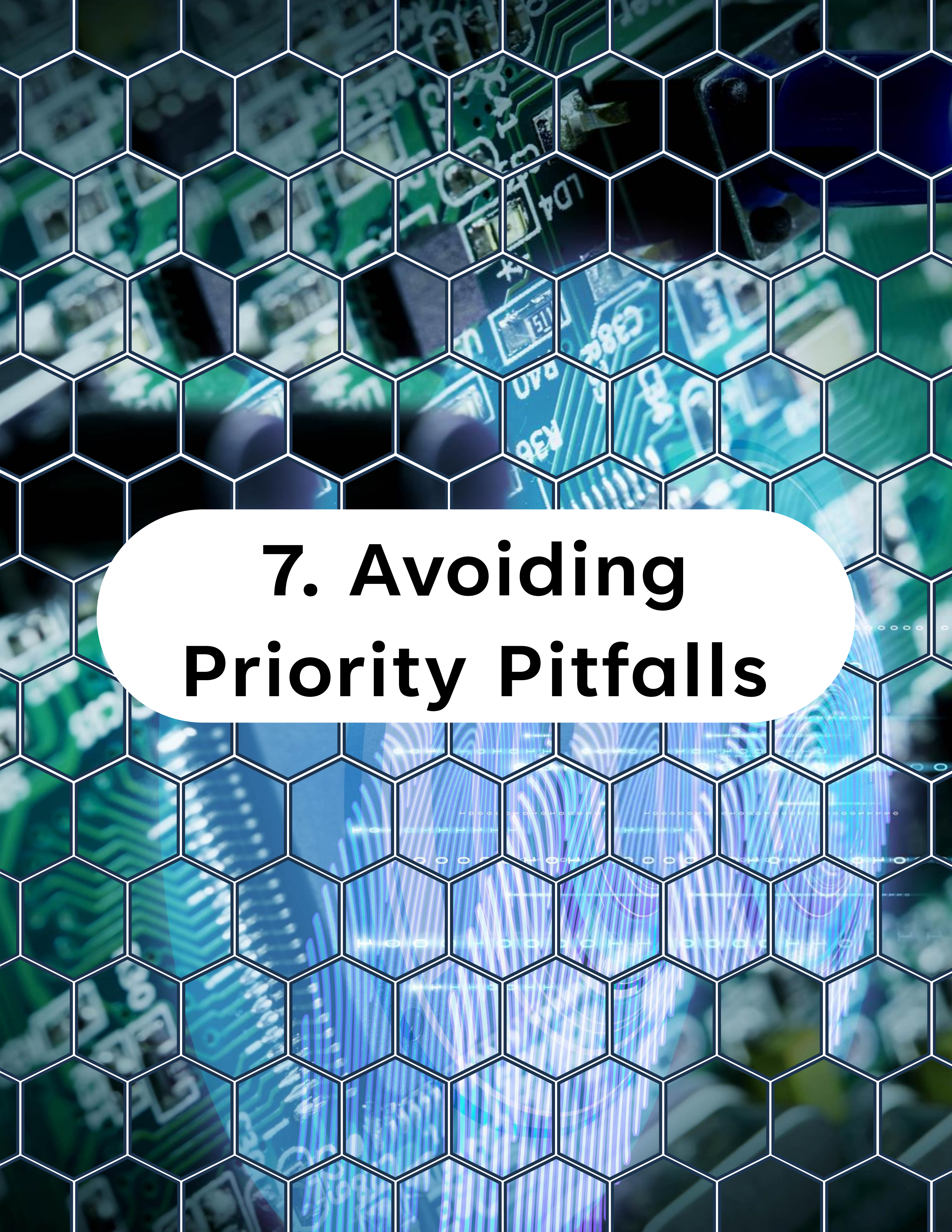
Example:



```c
vTaskPrioritySet(NULL, 10); // Change calling task's priority to 10
```

This can be useful for:

- Temporarily boosting a task’s responsiveness
- Reducing a task’s priority after a critical section is complete



7. Avoiding Priority Pitfalls

7. Avoiding Priority Pitfalls

❌ Priority Inversion

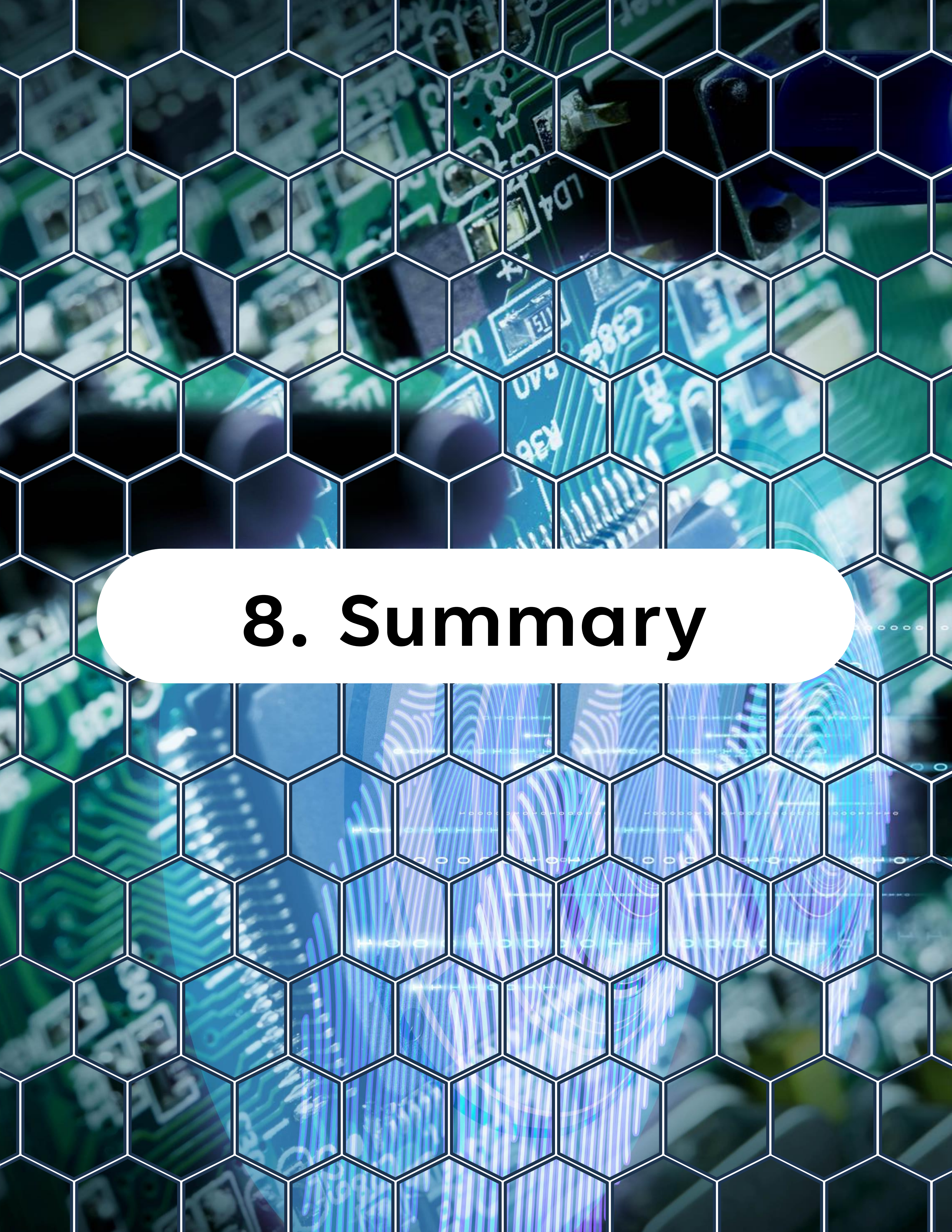
Occurs when a low-priority task holds a resource needed by a high-priority task.

Solution: Use mutexes with **priority inheritance**.

❌ Starvation

A low-priority task never runs because high-priority tasks keep CPU busy.

Solution: Ensure high-priority tasks block or yield regularly.



8. Summary

8. Summary

✅ Tasks have **five main states** in FreeRTOS.

✅ Priorities control **who runs next** — higher priority wins.

✅ Equal priorities share CPU time if time-slicing is enabled.

✅ Change priorities dynamically with `vTaskPrioritySet()` when needed.

✅ Avoid **priority inversion** and **starvation** by proper task design.

9. Challenge for Today

9. Challenge for Today

Create three tasks:

1. **Low-priority task** → Prints every 1s
2. **Medium-priority task** → Prints every 500ms
3. **High-priority task** → Runs at startup, changes its own priority to lowest after 5 iterations

Observe how the scheduling changes when priorities are adjusted at runtime.