30-Day FreeRTOS Course for ESP32 Using ESP-IDF

(Day 8)

“Queues: Theory and Practice”

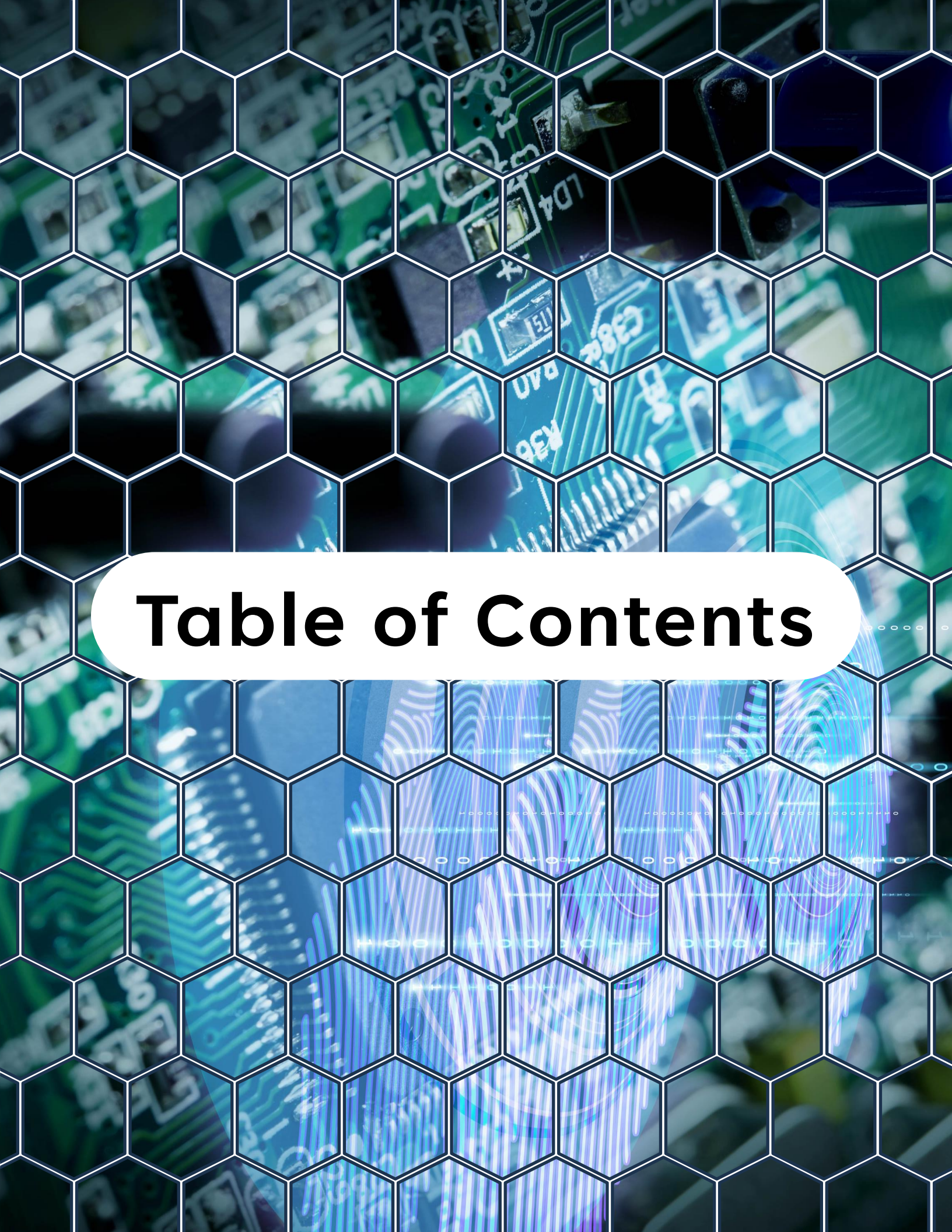
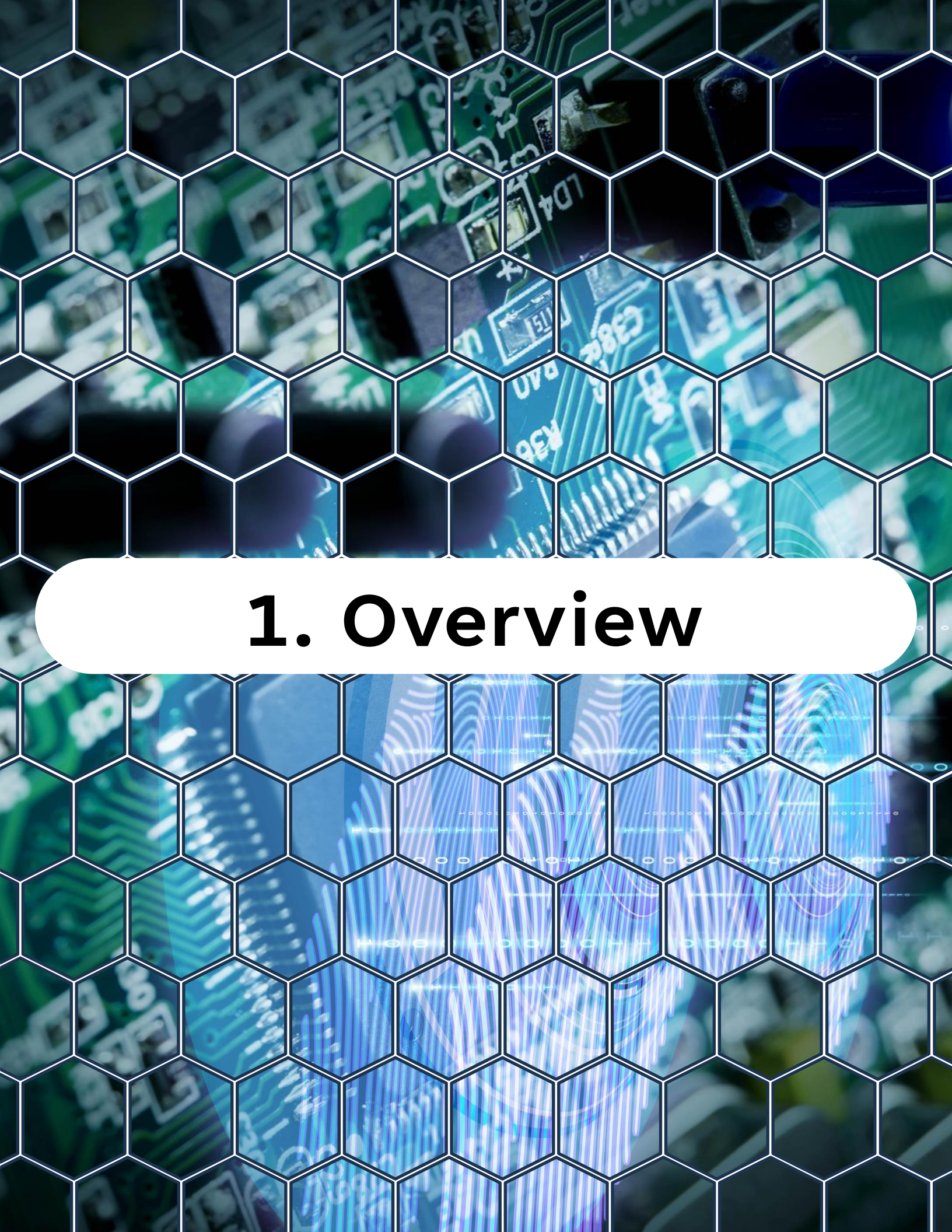


Table of Contents

Table of Contents

1. Overview
2. What Are **Queues** in FreeRTOS?
3. Queue Functions
4. Behavior of Queues
5. Example: Two Tasks Communicating with a Queue
6. Real-World Use Cases
7. Best Practices
8. Summary
9. Challenge for Today



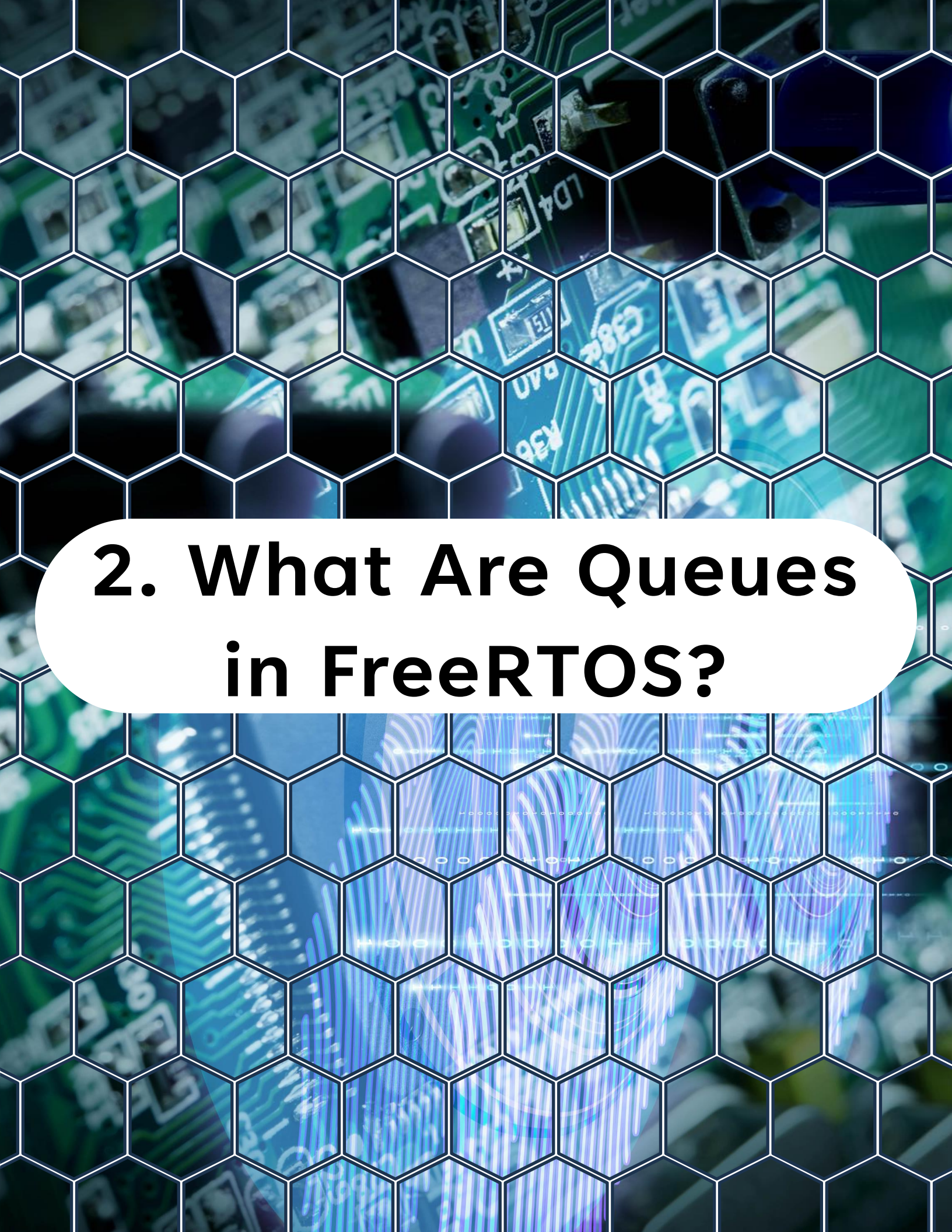
1. Overview

1. Overview

On **Day 8**, you'll learn:

- What **queues** are in FreeRTOS and why they matter
- How tasks communicate safely using queues
- How to create, send, and receive data in queues
- Queue behavior when full/empty
- Practical ESP32 code example with multiple tasks

By the end of this lesson, you'll be able to use queues to share data between tasks **without race conditions**.


2. What Are Queues
in FreeRTOS?

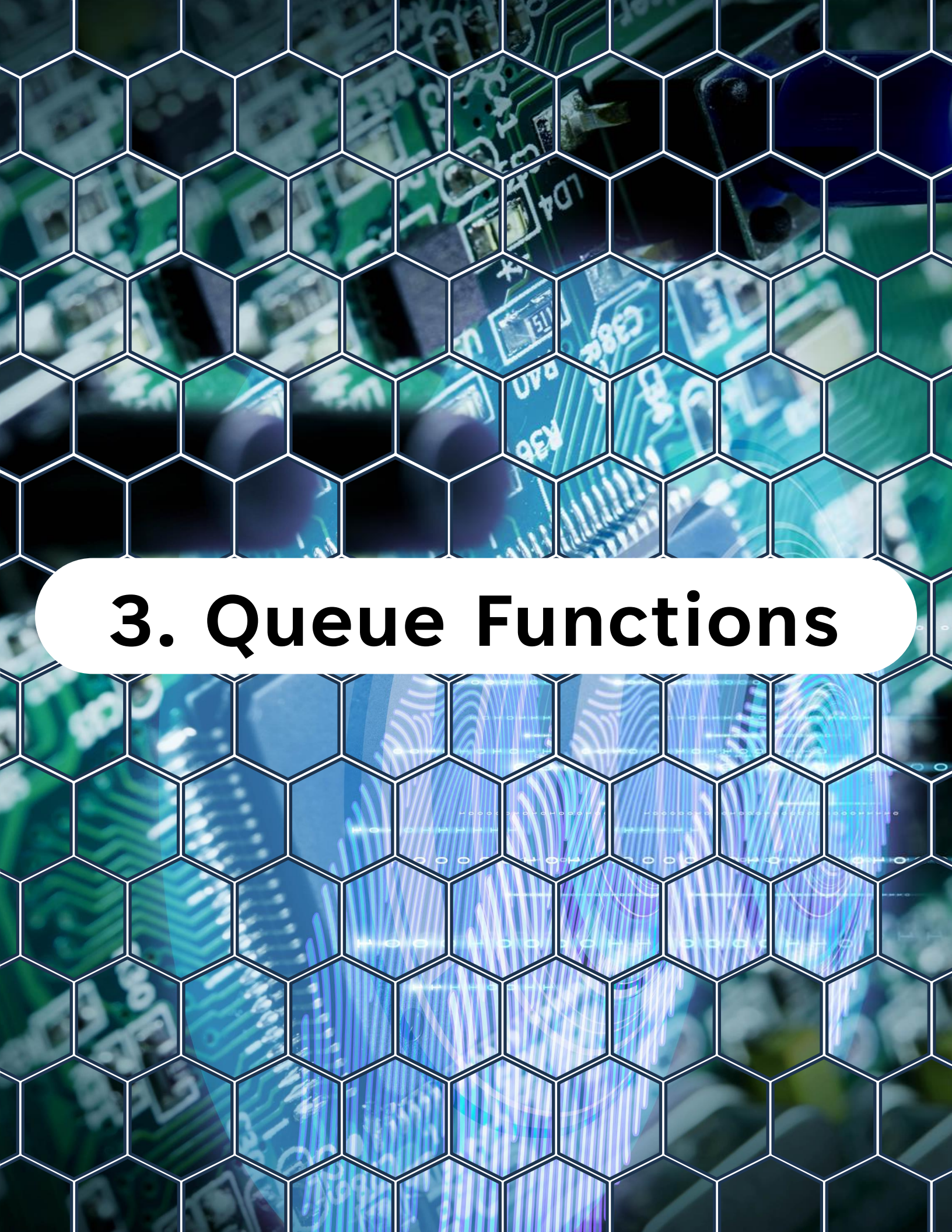
2. What Are Queues in FreeRTOS?

A queue is a FIFO (First-In, First-Out) buffer provided by FreeRTOS. It allows tasks (or ISRs) to **send and receive data safely** without corrupting memory.

Queues are widely used for:

- **Task-to-task communication**
- **ISR-to-task signaling**
- **Producer/consumer systems** (e.g., sensor → logger)

📌 Queues handle **synchronization** and **copying data**, so tasks don't fight over shared variables.



3. Queue Functions

3. Queue Functions

Creating a Queue:

```
QueueHandle_t xQueueCreate(UBaseType_t uxQueueLength,
                           UBaseType_t uxItemSize);
```

- `uxQueueLength`: number of items the queue can hold
- `uxItemSize`: size of each item in bytes

Sending to a Queue:

```
xQueueSend(QueueHandle_t xQueue,
           const void * pvItemToQueue,
           TickType_t xTicksToWait);
```

- `xTicksToWait`: how long to wait if the queue is full

Sending to a Queue from ISR:

```
xQueueSendFromISR(...);
```

3. Queue Functions

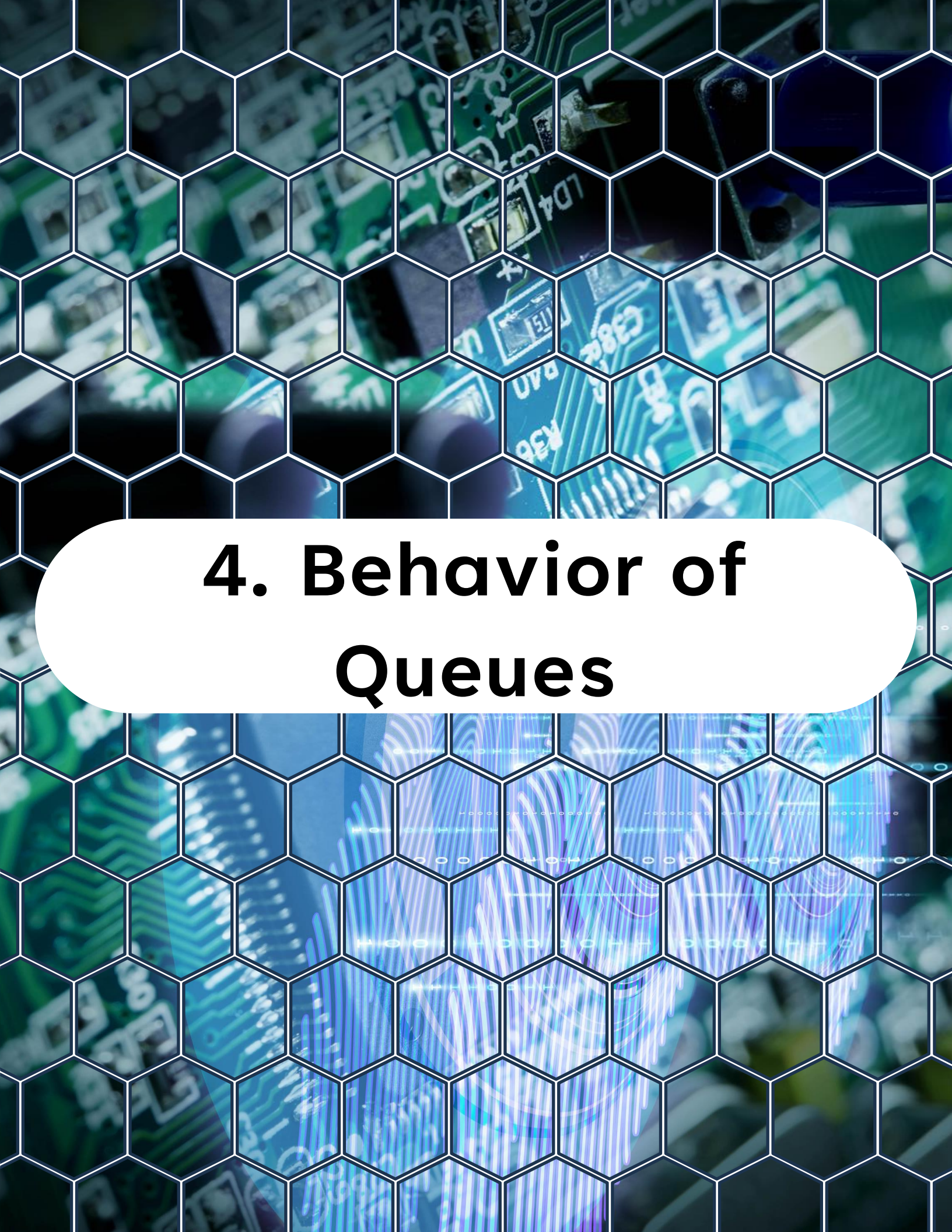
Receiving from a Queue:

```
xQueueReceive(QueueHandle_t xQueue,
              void * pvBuffer,
              TickType_t xTicksToWait);
```

- Blocks the task until data arrives or timeout expires

Receiving from a Queue from ISR:

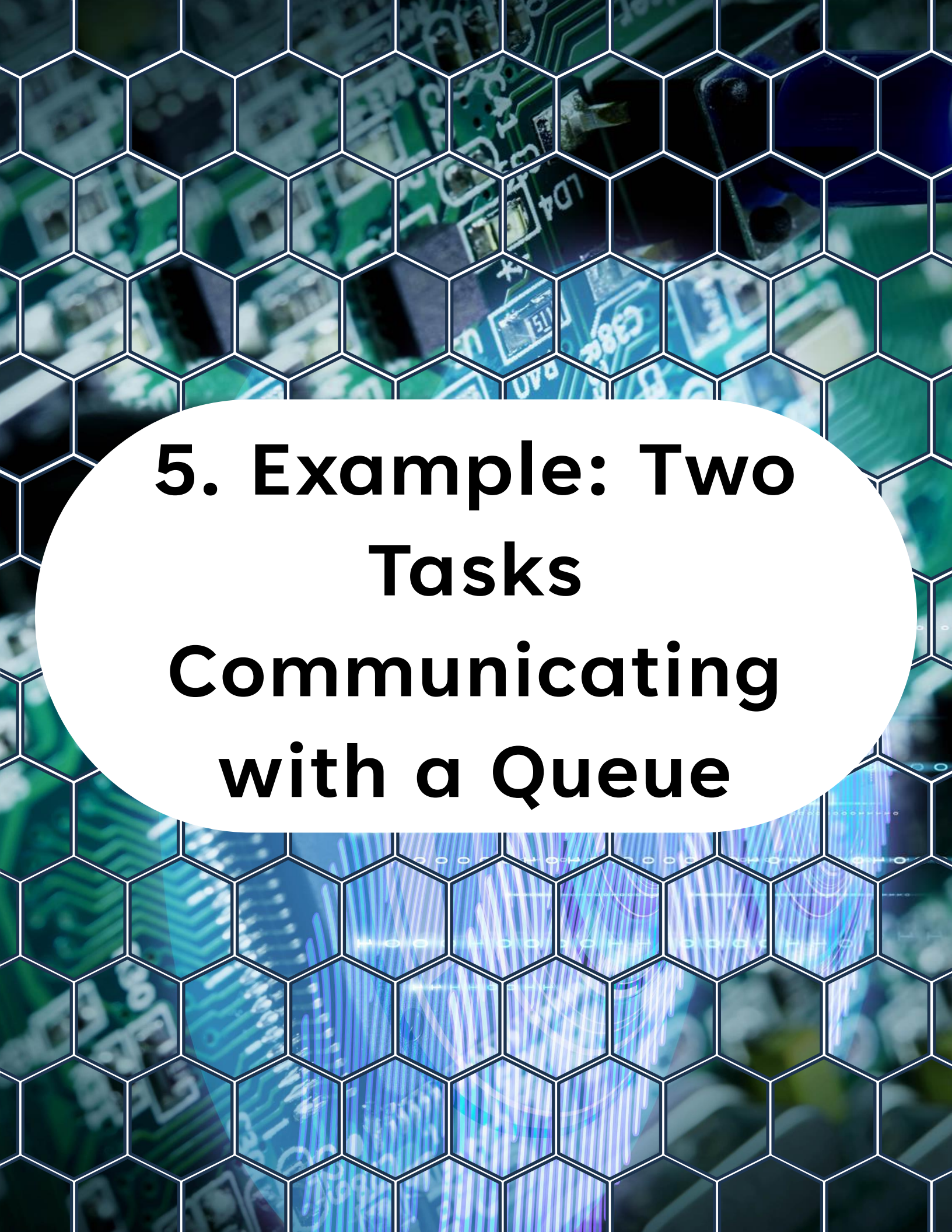
```
xQueueReceiveFromISR(...);
```



4. Behavior of Queues

4. Behavior of Queues

- If the **queue is full**, a task trying to send can **block** until space is available.
- If the **queue is empty**, a task trying to receive can **block** until data arrives.
- Queues can store **any type of data** — integers, structs, or custom messages.



5. Example: Two Tasks Communicating with a Queue

5. Example: Two Tasks Communicating with a Queue

Code Example

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

#define QUEUE_LENGTH 5

QueueHandle_t queue;

void producer_task(void *pvParameters) {
    int count = 0;
    while (1) {
        if (xQueueSend(queue, &count, pdMS_TO_TICKS(100)) == pdPASS) {
            printf("Producer sent: %d\n", count);
            count++;
        } else {
            printf("Producer: Queue full!\n");
        }
        vTaskDelay(pdMS_TO_TICKS(500));
    }
}

void consumer_task(void *pvParameters) {
    int value;
    while (1) {
        if (xQueueReceive(queue, &value, pdMS_TO_TICKS(1000)) == pdPASS) {
            printf("Consumer received: %d\n", value);
        } else {
            printf("Consumer: Queue empty!\n");
        }
    }
}

void app_main(void) {
```

5. Example: Two Tasks Communicating with a Queue

```
34 void app_main(void) {
35     queue = xQueueCreate(QUEUE_LENGTH, sizeof(int));
36     if (queue == NULL) {
37         printf("Failed to create queue\n");
38         return;
39     }
40 
41     xTaskCreate(producer_task, "Producer", 2048, NULL, 5, NULL);
42     xTaskCreate(consumer_task, "Consumer", 2048, NULL, 5, NULL);
43 }
44 
```

Expected Output:

Producer sent: 1

Consumer received: 1

Producer sent: 2

Consumer received: 2

- If the consumer falls behind, the queue fills, and the producer will print "**Queue full!**".
- If the producer stops sending, the consumer eventually prints "**Queue empty!**".

6. Real-World Use Cases

6. Real-World Use Cases

- Sensor sampling task → Queue → Data logging task
- ISR for UART reception → Queue → Parser task
- Button ISR → Queue → Debouncer task

Queues let you decouple **data producers** from **data consumers**, improving modularity.

7. Best Practices

7. Best Practices

- Use **blocking timeouts** (pdMS_TO_TICKS(x)) instead of busy loops.
- Keep queue items **small** — use pointers to large data blocks instead of copying them.
- For **fast ISR-to-task communication**, consider **Task Notifications** (Day 16).
- Monitor system performance with `uxQueueMessagesWaiting()`.

8. Summary

8. Summary

✅ Queues are **FIFO buffers** for safe inter-task and ISR communication.

✅ Use `xQueueCreate()`, `xQueueSend()`, `xQueueReceive()` for task-to-task transfers.

✅ Handle **full** and **empty** queues gracefully with timeouts.

✅ Perfect for **producer/consumer patterns** in embedded systems.

9. Challenge for Today

9. Challenge for Today

Build a system with:

1. A **producer task** that generates a random number every 200 ms and sends it to a queue.
2. A **consumer task** that prints the number and checks if it’s even or odd.
3. Modify the producer to **suspend itself** if the queue is full, and let the consumer resume it when space is available.