

# Exploring OS Variations and I/O Latency: RTOS and No OS

A Comparative Analysis for Autonomous Systems

CIS 573-7101: Operating Systems (2024 Fall: online)

**Philip Carr**

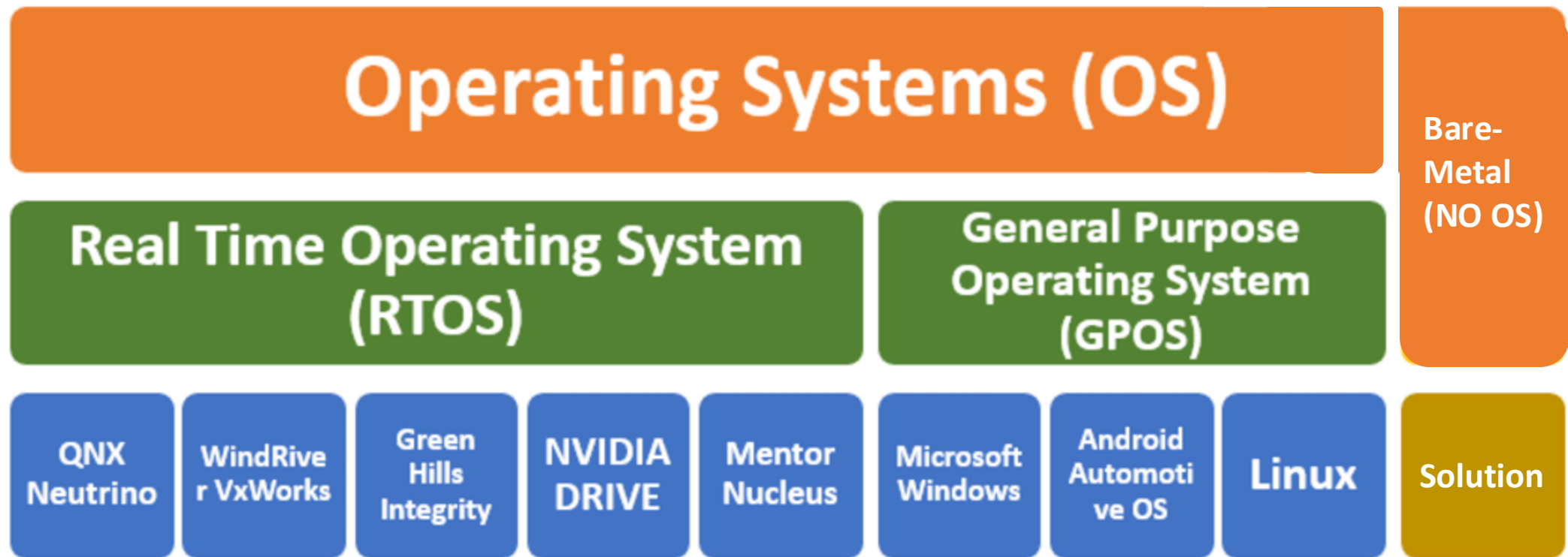
# The Role of OS in Embedded Systems

- An Operating System (OS) manages hardware and software resources, enabling multitasking and efficient resource allocation.
- Embedded systems, such as autonomous vehicles, require precise and predictable behavior, making OS selection a critical design decision.
- **Key Question:** How do different OS choices (RTOS, or no-OS) affect system performance, particularly in latency-critical tasks?

## Example Scenario:

- RTOS ensures real-time responsiveness, crucial for obstacle avoidance or hazard detection.
- No-OS offers minimal overhead but limited flexibility and scalability.

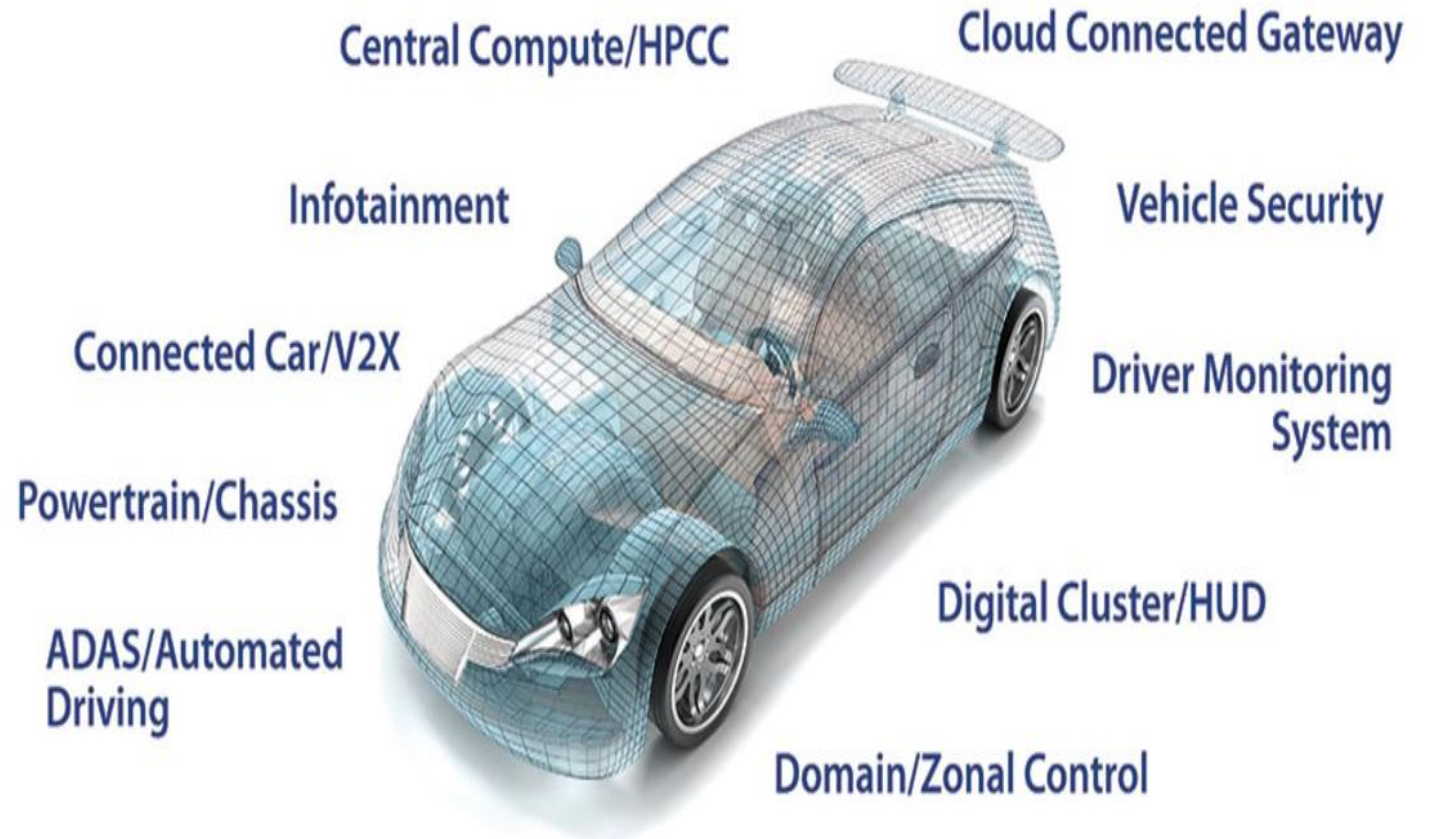
# Comparative Landscape of Operating Systems in Embedded Systems



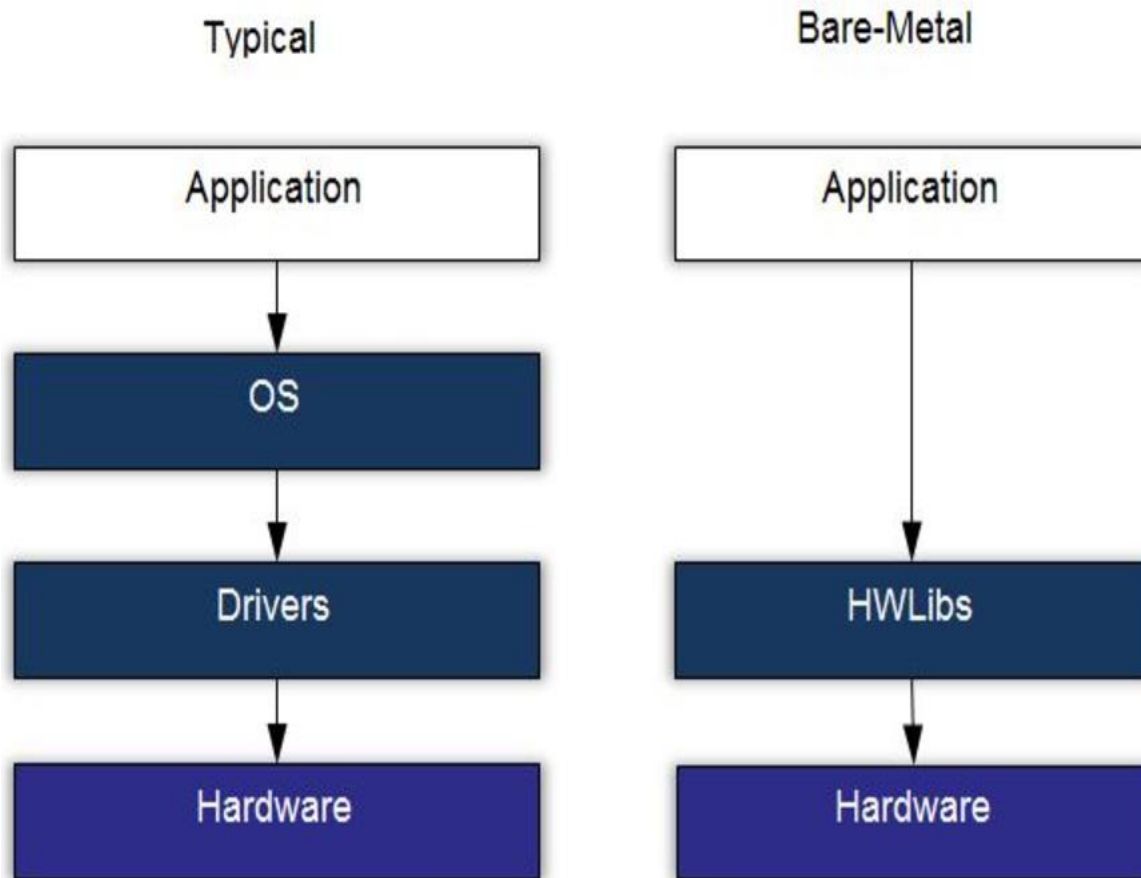
# RTOS in the Real World - Automotive Applications

## Green Hills' Leadership in Safety and Security

- Green Hills provides industry-leading RTOS, tools, and hypervisor services tailored for **automotive safety and security**.
- Achieves **ISO 26262 ASIL D** certification, ensuring the highest safety standards for functional safety.
- Adopts **ISO/SAE 21434 CAL4** and **UNECE WP.29 CSMS** for cutting-edge automotive cybersecurity compliance.
- A trusted partner for supporting advanced **vehicle electronics development** with the latest global safety and security standards.



# NO OS (Bare-Metal) in the Real World - Applications



- **Application Layer:** Directly interacts with hardware without an OS intermediary.
- **HWLibs (Hardware Libraries):** Provides low-level access to hardware functionality (e.g., GPIO, I2C, UART).
- **Hardware:** The physical components controlled directly by the application.

# Objectives of the Project

- **Analyze OS Types:**
  - Investigate the trade-offs between GPOS, RTOS, and no-OS in terms of latency, scalability, and complexity.
- **Evaluate I/O Latency:**
  - Measure and compare I/O latency under different OS setups.
- **Design a System Model:**
  - Develop an autonomous vehicle architecture to test OS performance in a real-world scenario.
- **Provide Practical Insights:**
  - Recommend the best OS setup based on performance and cost-efficiency for latency-critical systems.
- **Deliverables:**
  - A comparative analysis of OS variations.
  - A working system prototype demonstrating the effects of OS types.
  - A presentation summarizing findings and recommendations.

# Anticipation of Results

- **Hypothesis:**

- **No-OS:** Will perform well in simple scenarios but may struggle with multitasking or complex decision-making.
- **RTOS:** Will excel in low-latency scenarios due to deterministic task scheduling.
- **GPOS:** Will provide flexibility but introduce significant delays in real-time operations.

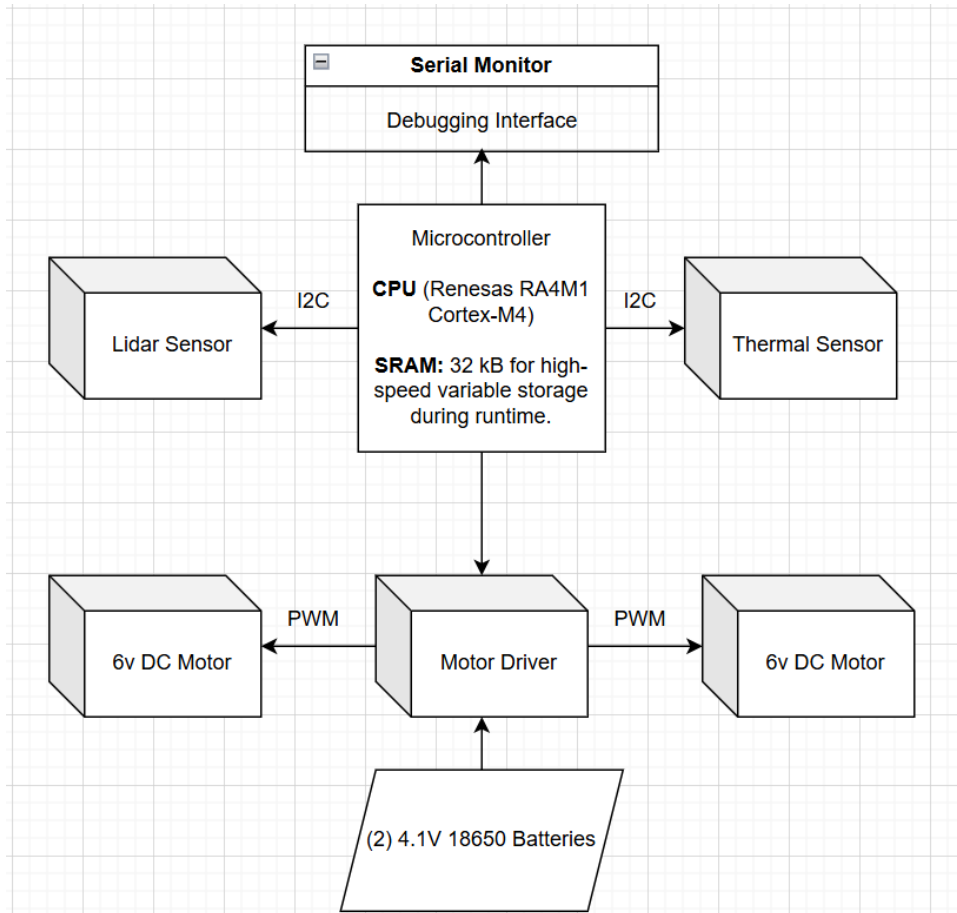
- **Expected Insights:**

- Quantify the latency differences between OS types.
- Determine the trade-offs between scalability real-time performance (RTOS), and simplicity (no-OS).

- **Questions to Explore:**

- Can RTOS consistently meet real-time requirements in complex systems?
- Is no-OS sufficient for small-scale applications?

# Importance of OS Selection in Embedded Systems



## Impact on System Reliability:

- In an autonomous vehicle, quick response times are essential for tasks like detecting and avoiding obstacles or responding to fires. OS selection defines the system's ability to handle simultaneous inputs, process data, and make timely decisions.

## RTOS:

Guarantees deterministic behavior and low-latency task scheduling. Ideal for applications requiring real-time responses.

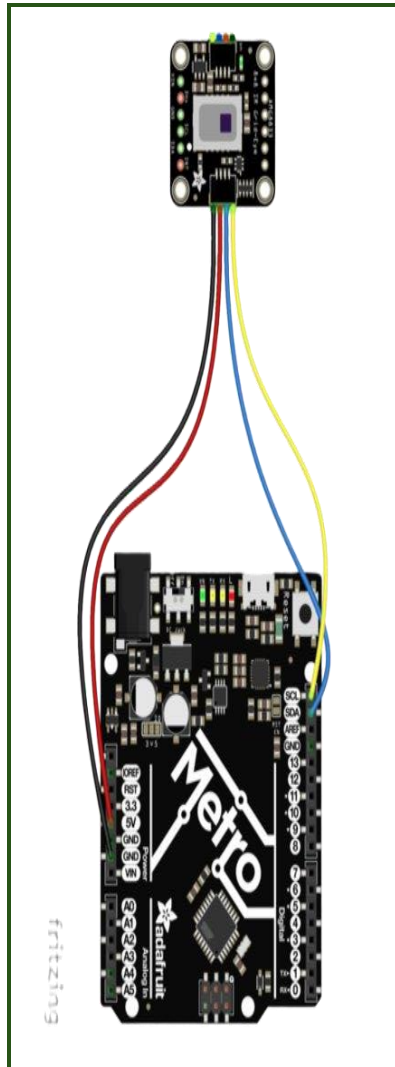
- Example:** FreeRTOS can ensure consistent motor control and sensor data processing in milliseconds.

## No-OS:

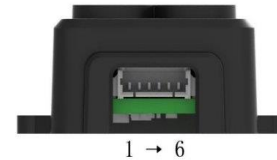
Simple and efficient but lacks multitasking capability. Best for straightforward tasks with minimal complexity.



# Overview of Autonomous Vehicle Application



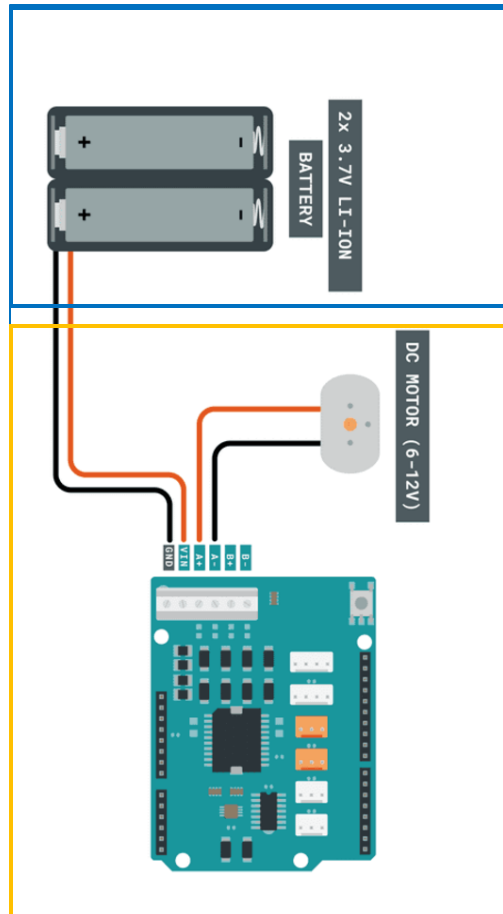
No.	Function	Description
1	+5V	Power supply
2	RXD/SDA	Receiving/Data
3	TXD/SCL	Transmitting/Clock
4	GND	Ground
5	Configuration Input	Ground: I2C mode /3.3V: Serial port Communications mode
6	Multiplexing output	Default: on/off mode output I2C mode: Data availability signal on but not switching value mode



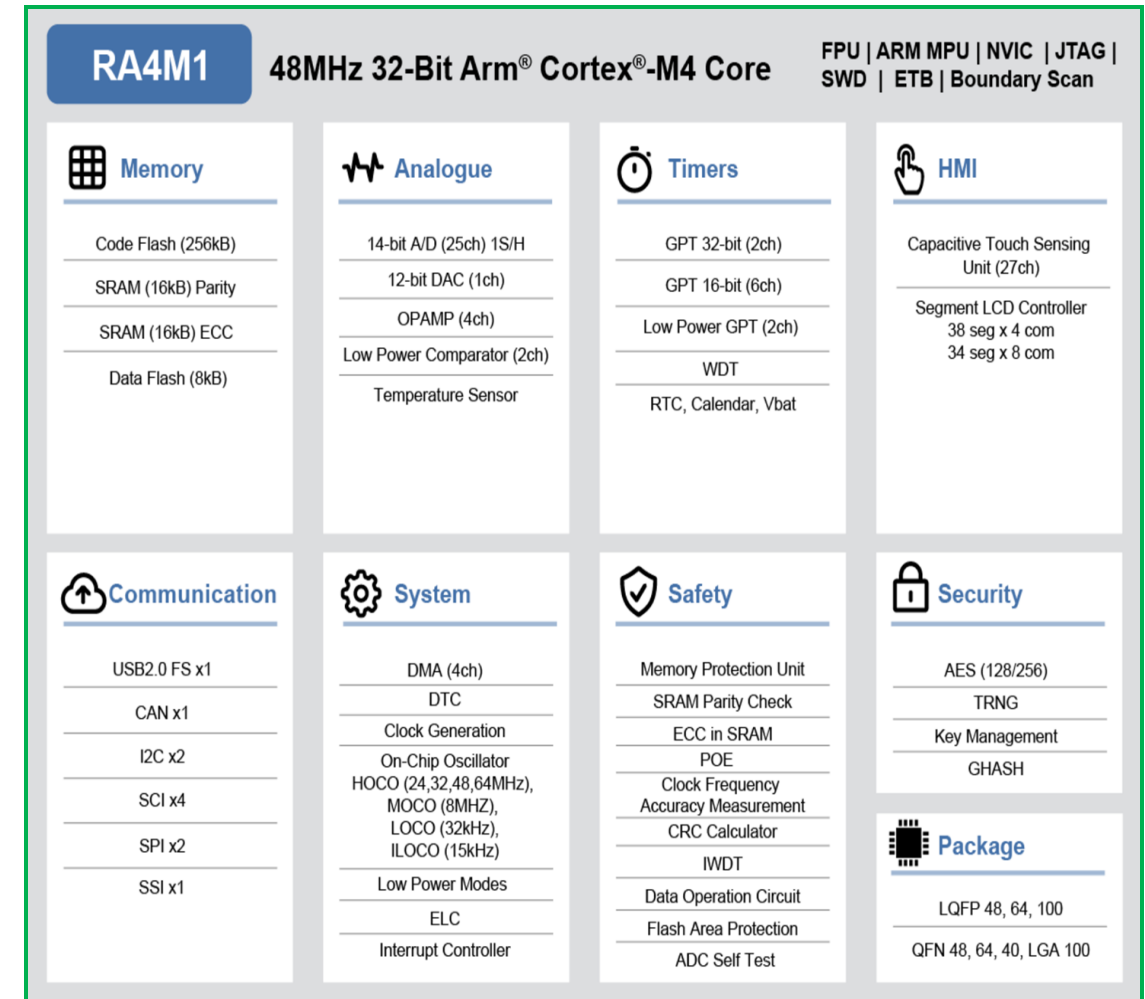
- **Lidar Sensor:** The Lidar detects an obstacle within a critical range, triggering a motor stop under all OS types.
- **Thermal Sensor:** The thermal sensor detects high temperatures, triggering a motor reverse operation.
  - **Vin → 3.3V**
  - **GND → GND**
  - **SDA → A4**
  - **SCL → A5**

# Overview of Autonomous Vehicle Application

## Pt. 2






- **Motor Driver and Motors:** Enable movement and direction control.
- **CPU (Renesas RA4M1 Cortex-M4):** Processes sensor data and controls the vehicle.
  - **I/O Latency Testing and Task Scheduling:** The robust system and interrupt management support real-time testing scenarios for comparing RTOS, GPOS, and No OS environments.
  - **Integration with Sensors and Actuators:** Multiple ADC/DAC and communication interfaces simplify connecting thermal and LIDAR sensors, as well as motor drivers.
  - **Power Optimization:** The low-power modes and precise clock generation allow efficient power usage for an autonomous vehicle system.
- **Power Supply:** Two 4.1V 18650 batteries.



# Initial Struggles with RTOS on Arduino

- **Kernel Limitations with Arduino R3:**
- The **Arduino Uno R3** has a very limited **kernel environment**, making it unsuitable for running robust RTOS implementations.
  - The R3's **ATmega328P microcontroller** lacks the capability to handle **kernel-level task scheduling** efficiently due to its limited processing power and memory (2KB SRAM).
  - This caused significant difficulties when trying to implement **task prioritization** and **timing-based operations** required by FreeRTOS.

Comparison of Arduino UNO R3 and R4			
Product Name	Arduino UNO Rev3	Arduino UNO R4 Minima	Arduino UNO R4 WiFi
			
Microcontroller	ATmega328P Microchip (8-bit AVR RISC)	Renesas Electronics RA4M1 (32-bit ARM Cortex-M4)	Renesas Electronics RA4M1 (32-bit ARM Cortex-M4)
Operating Voltage	5V	5V	5V
Input Voltage	6-20V	6-20V	6-20V
Digital IO Pin	14	14	14
PWM digital IO pin	6	6	6
Analog Input Pin	6	6	6
DC current for each I/O pin	20mA	8mA	8mA
Clock Speed	16MHz	48MHz	48MHz
Flash Memory	32kb	256kb	256kb
SRAM	2kb	32kb	32kb
USB	USB-B	USB-C	USB-C
DAC(12位)	1	1	1
SPI	1	2	2
I2C	/	2	2
CAN	/	1	1
Operational Amplifier	/	1	1
SWD	/	1	1
RTC	/	/	1
QWIIC 12C CONNECTOR	/	/	1
Light Emitting Diode Matrix	/	/	12*8(96 red indicators)

# FreeRTOS on the Microcontroller

## Tasks (Task 1, Task 2, Task 3, Task 4):

- These represent independent units of execution in FreeRTOS.
- Each task is prioritized, allowing critical tasks to preempt non-critical ones.

## FreeRTOS Kernel:

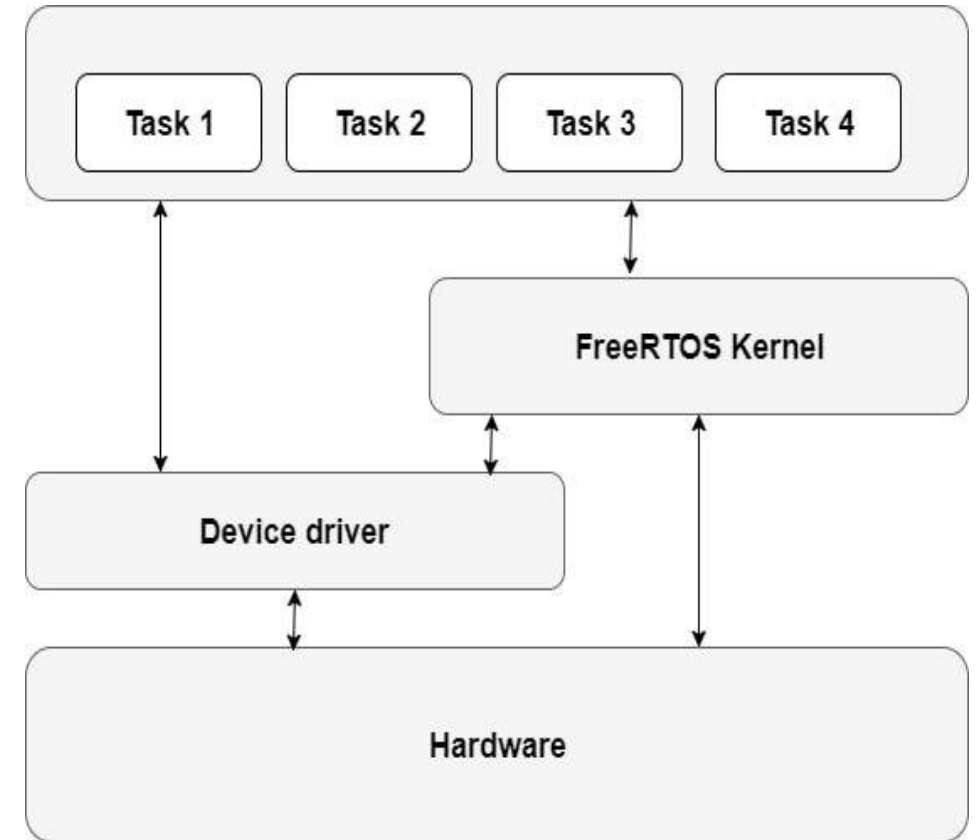
- Acts as the central manager for all tasks.
- In our project, the kernel prioritized tasks like obstacle detection and motor stopping to ensure safety.

## Device Driver:

- Connects tasks to the underlying hardware.
- In our project:
  - Facilitates communication between tasks and hardware components like the I2C bus for sensors or PWM outputs for motor control.

## Hardware:

- Refers to the microcontroller and connected components:



# RTOS Variants for Safety Applications:

## Hard RTOS:

Guarantees fixed, short response times for critical tasks. Essential for **braking systems** and **airbag deployment** in automotive safety, where timing failures can cause catastrophic results. Ensures immediate motor control or emergency braking when critical distances or hazards are detected.

## Soft RTOS:

Prioritizes response times but tolerates occasional delays. Suitable for less critical systems like **media streaming** or **navigation maps** in non-autonomous scenarios. Non-critical tasks like periodic sensor data logging could use Soft RTOS to free up resources for critical operations.

## Real RTOS:

Offers even shorter response times than Hard RTOS for ultra-critical tasks. Ideal for **advanced self-driving vehicles**, where microsecond-level decisions are required to avoid collisions. Potential future need for real-time obstacle avoidance in autonomous systems.

## Firm RTOS:

Similar to Soft RTOS but tolerates occasional timing lapses without significant consequences. Useful for features like **climate control** or **infotainment**, where delays are acceptable. May handle background tasks like reporting sensor statuses without interrupting safety-critical functions.

# How No OS is Implemented in Arduino

## Task Management

- **Single Threaded:**

- Arduino executes instructions sequentially within the `loop()` function.
- Concurrent tasks are simulated using state machines or timer-based logic.

```
void loop() {  
    readSensor();    // Task 1  
    controlMotor();  // Task 2  
    updateLED();     // Task 3  
}
```

Tasks are executed one after another, relying on efficient code to ensure real-time responsiveness.

- **Key Functions:**

- `setup()` for initialization.
- `loop()` for task execution.

# Initial Struggles with No OS

## **Task Management:**

- Struggle: Simultaneously running multiple tasks (e.g., motor control, sensor reading, communication).
- Impact: Required manual time-sharing and precise timing.

## **Lack of Multitasking:**

- Struggle: No native task scheduler for handling concurrent processes.
- Impact: Increased complexity in coding; required manual state machines.

## **Hardware Interrupts:**

- Struggle: Managing interrupts from LIDAR and thermal sensors while maintaining motor control.
- Impact: Interrupt conflicts caused unstable system behavior.

# System Architecture Diagram

## System Components:

- **I2C Bus:** Transfers data from the Lidar and thermal sensors to the CPU, enabling real-time input for processing.
- **PWM Outputs:** Modulates motor speed and direction by transmitting precise control signals to the motor driver.
- **CPU (Renesas RA4M1):** Serves as the system's decision-making unit, processing sensor data and executing commands for motor control.
- **Power Source:** Provides a stable energy supply, ensuring seamless operation of all system components.

## Operating System Impact on Architecture:

- **RTOS (Real-Time OS):** Ensures deterministic scheduling of sensor input and motor control tasks, ideal for time-sensitive applications.
- **No OS (Bare-Metal):** Executes tasks sequentially, offering simplicity at the expense of advanced functionality and multitasking.



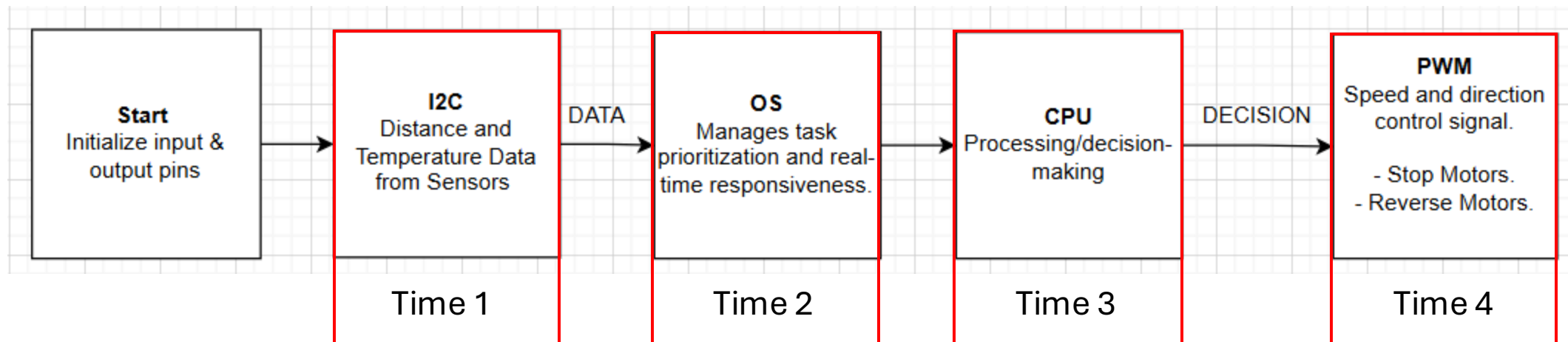
# Understanding Latency in Embedded Systems

## Types of Latency in Embedded Systems:

- 1.Sensor Read Latency:** Time taken to read data from sensors (e.g., distance or temperature).
- 2.Task Scheduling Latency:** Delay between task assignment and execution by the OS.
- 3.Response Latency:** Time required for the system to execute a decision (e.g., stopping a motor).
- 4.Total System Latency:** Cumulative time from input (e.g., sensor activation) to output (e.g., motor response).

## Timestamps (Timing References):

- Sensor Read Latency ( $T_2 - T_0$ ): Between "I2C" and "OS".
- Task Scheduling Latency ( $T_3 - T_2$ ): Between "OS" and "CPU".
- Motor Response Latency ( $T_4 - T_3$ ): Between "CPU" and "PWM".
- Total Response Time ( $T_4 - T_0$ ): From "Start" to "PWM".

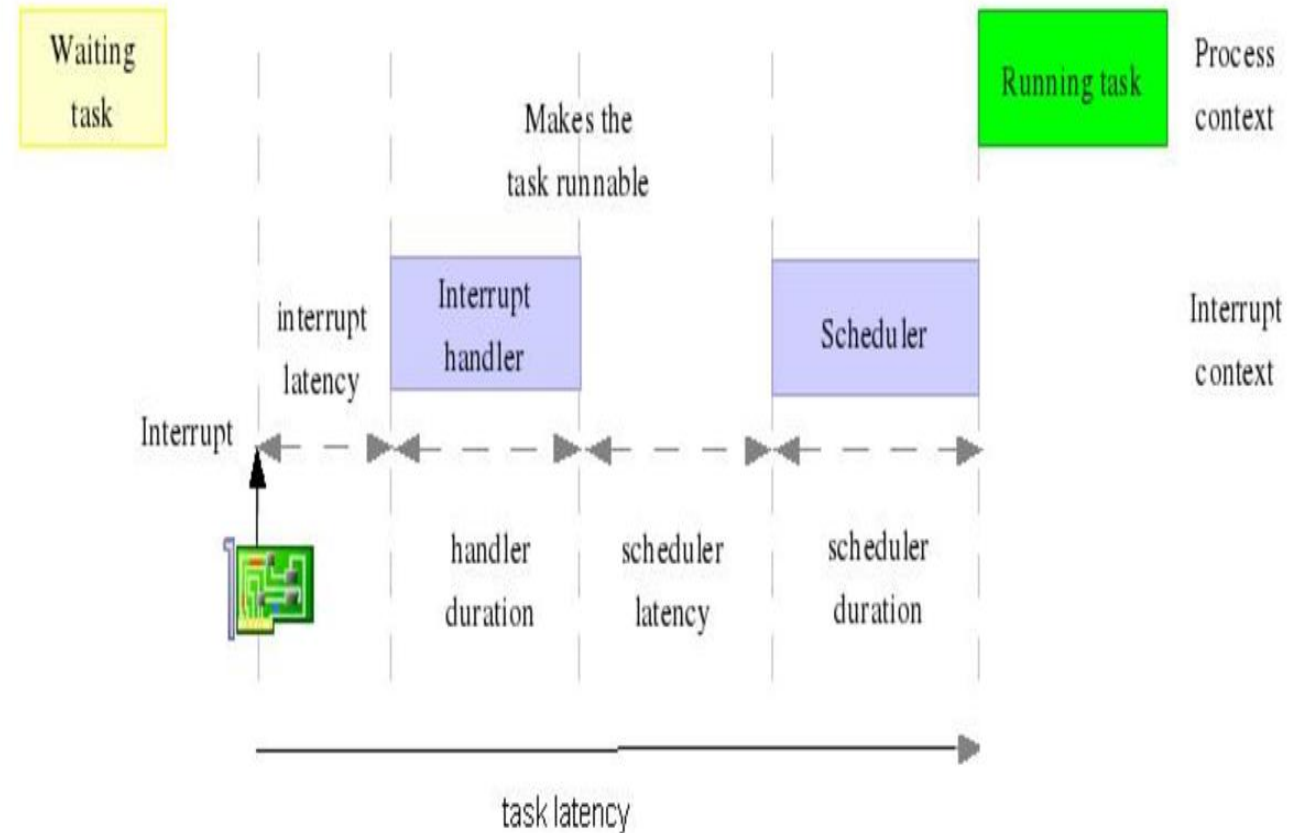


# Concept of Task Latency

- The time taken by a task to execute from start to finish (e.g., reading sensor data).
- Includes:
  - Task execution time.
  - Any delays or blocking caused by sequential execution (in No OS).

Latency ( $T3 - T2$ ) is the time elapsed between:

- **T2:** When a task is initiated (e.g., a sensor read request is sent).
- **T3:** When the task is completed (e.g., the sensor value is returned).



# Latency Metrics in Our OS Implementations

## Tasks Measured:

- **Distance Reading:** Time taken to retrieve LIDAR sensor data.
- **Temperature Reading:** Time taken to scan the thermal sensor.
- **Motor Control:** Time taken to decide and execute motor operations.

## Latency Metrics:

- **Distance Task Latency:** From start to end of distance measurement.
- **Temperature Task Latency:** From start to end of temperature reading.
- **Motor Control Task Latency:** From start to motor state change.
- **Total Task Latency:** Sum of all three.

## Tools Used:

- `millis()` function to measure time at each stage.

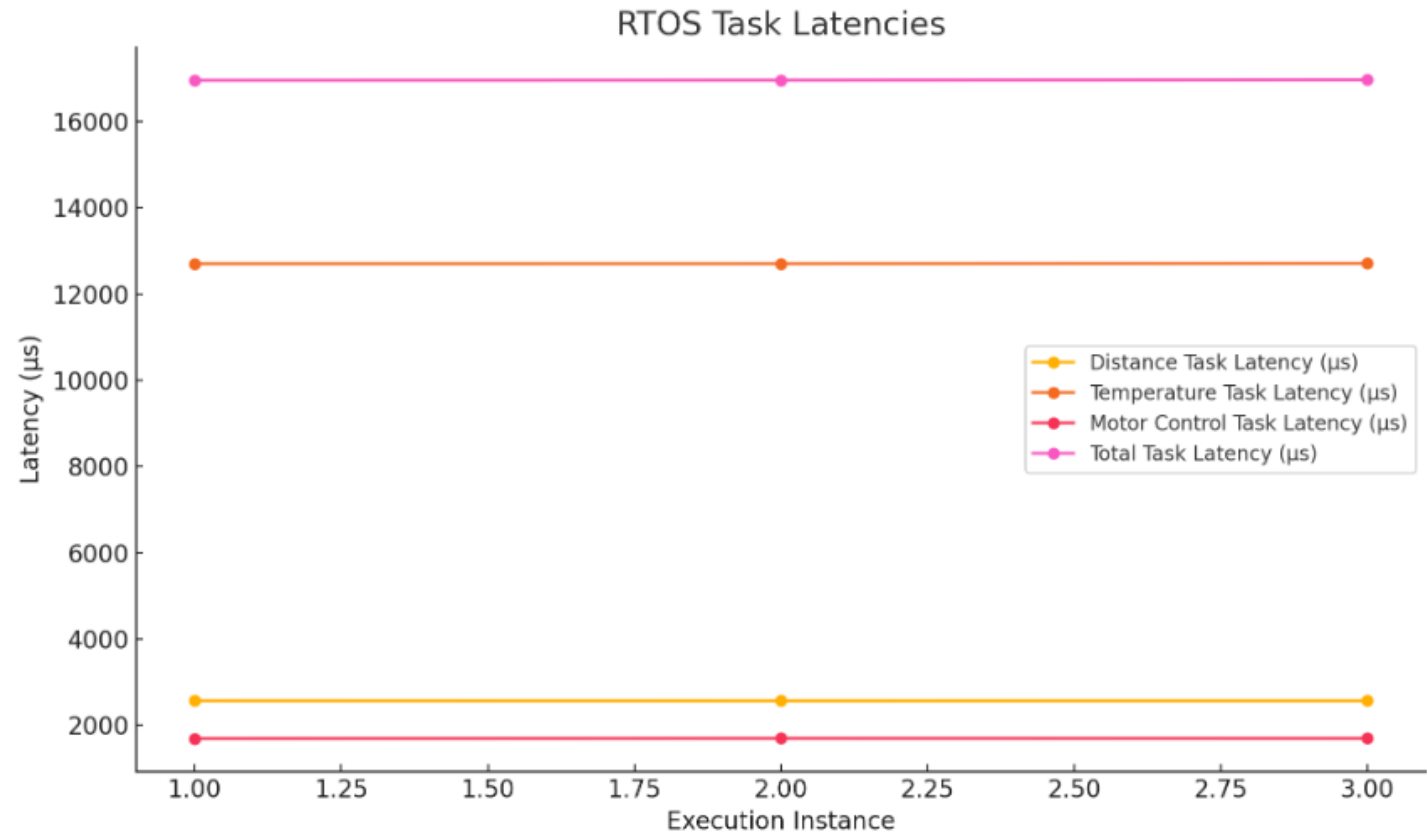
# RTOS Task Latency Metrics

## Consistent Task Latency:

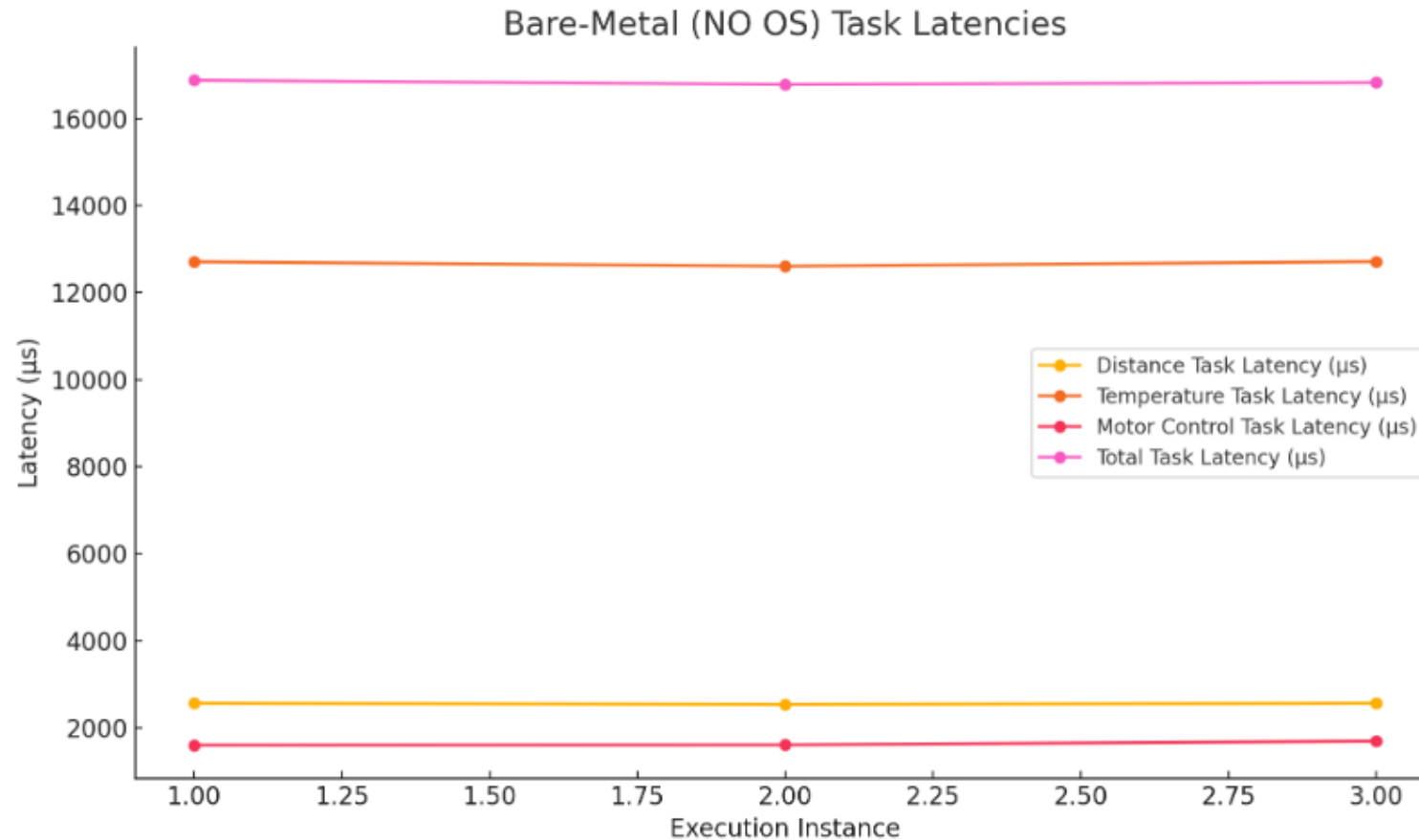
- The **Distance Task Latency** remains stable at around **2565  $\mu\text{s}$**  across all three execution instances.
- The **Temperature Task Latency** is consistent at approximately **12703  $\mu\text{s}$** .
- The **Motor Control Task Latency** remains constant at around **1690  $\mu\text{s}$** .

## Total Task Latency:

- The **Total Task Latency** adds up to approximately **16959  $\mu\text{s}$** , which is consistent across all execution instances.
- This is a sum of all three task latencies, reflecting RTOS's deterministic behavior.



# NO OS Task Latency Metrics



## Consistent Task Latency:

- **Distance Task Latency** remains stable at approximately **2566 µs** for all three execution instances.
- **Temperature Task Latency** is consistent around **12708 µs**.
- **Motor Control Task Latency** fluctuates slightly but stays near **1690 µs**, showing reliability.

## Total Task Latency:

- The **Total Task Latency** averages approximately **16879 µs**, showcasing consistency even without a dedicated operating system.

# Task Latency Comparison

## Distance Task Latency

- **Observation:**

- RTOS has consistent latency around 2565–2566  $\mu\text{s}$ .
- NO OS shows slight variations, ranging from 2539–2566  $\mu\text{s}$ .

- **Explanation:**

- The consistent performance in RTOS is due to its deterministic task scheduling.
- NO OS processes tasks sequentially, which introduces minor timing variations, likely due to hardware or execution timing differences.

# Task Latency Comparison Pt.2

## Temperature Task Latency

- **Observation:**

- RTOS consistently records latencies between 12703–12711  $\mu$ s.
- NO OS records slightly lower latencies, ranging from 12591–12713  $\mu$ s.

- **Explanation:**

- RTOS's multitasking adds context-switching overhead, which increases the latency slightly.
- NO OS benefits from direct sequential processing, reducing the overhead and resulting in slightly lower latencies.

# Task Latency Comparison Pt.3

## Motor Control Task Latency

- **Observation:**

- RTOS latency is around 1690–1694  $\mu\text{s}$ .
- NO OS latency ranges from 1605–1696  $\mu\text{s}$ , with slightly lower minimum values.

- **Explanation:**

- RTOS ensures consistent motor task performance due to its scheduler but adds overhead.
- NO OS allows direct access to hardware, resulting in faster execution, but at the cost of flexibility and task management.



# Task Latency Comparison Pt.4

## Total Task Latency

- **Observation:**

- RTOS total latency ranges from 16959–16969  $\mu\text{s}$ .
- NO OS total latency ranges from 16781–16971  $\mu\text{s}$ , with slightly lower minimum values.

- **Explanation:**

- RTOS's additional latency is caused by context-switching and scheduling overhead.
- NO OS minimizes latency by eliminating overhead but sacrifices task isolation and prioritization.

# Comparative Analysis of OS Variations

## RTOS (Real-Time Operating System):

- **Strengths:** Precise task scheduling, deterministic behavior, and multitasking capabilities.
- **Ideal For:** Complex systems like autonomous vehicles requiring real-time responsiveness.
- **Latency Observation:** Tasks executed with slight overhead due to scheduling, resulting in total latency averaging **16,959  $\mu$ s**.

## Bare-Metal (NO OS):

- **Strengths:** Minimal latency, direct hardware access, and simplicity.
- **Ideal For:** Simple systems with fixed functionality and fewer tasks, such as basic motor and sensor control.
- **Latency Observation:** Slightly lower total latency averaging **16,879  $\mu$ s** due to the absence of a scheduler.

# References

- Green Hills Software. (n.d.). *Automotive solutions*. Retrieved from [https://www.ghs.com/products/auto\\_solutions.html](https://www.ghs.com/products/auto_solutions.html)
- Blackberry QNX. (n.d.). *Software-defined vehicle: Automotive RTOS*. Retrieved from <https://blackberry.qnx.com/en/ultimate-guides/software-defined-vehicle/automotive-rtos>
- Android Source. (2024). *Automotive car framework core: GAPB 2024*. Retrieved from <https://source.android.com/static/docs/automotive/car-framework-core/gapb-2024-aaos-101-day3-carframework-core.pdf>
- International Journal of Science and Technology. (2015). *Automotive software design*. Retrieved from <https://sciresol.s3.us-east-2.amazonaws.com/IJST/Articles/2015/Issue-19/Article54.pdf>