# Understanding Real-Time Systems: Design and Analysis

Based on the book by Phillip A. Laplante & Seppo J. Ovaska

Navarro, Rod Geryk C.

CPE161P-4– C1

# What Are Real-Time Systems?

## Chapter 1: Fundamentals of Real-Time Systems

•Real-time systems process information and provide responses within a specific time limit.

•Examples: Airbag systems, medical devices, autopilot systems in planes.

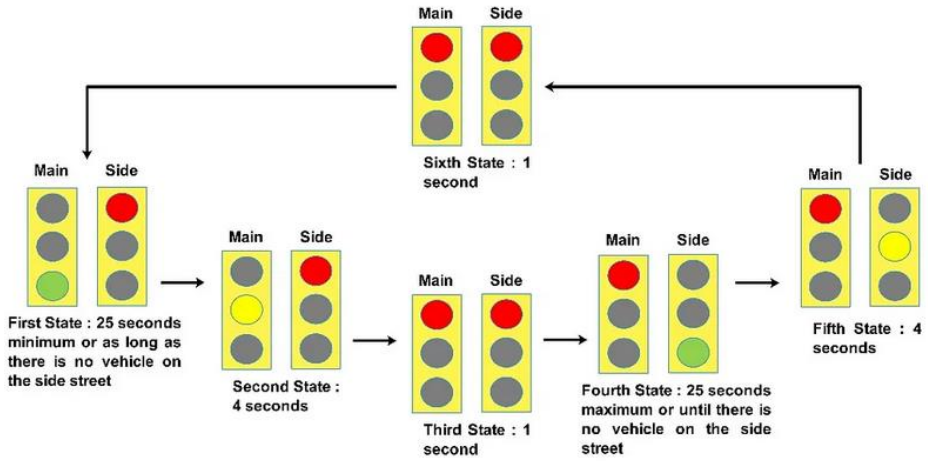


**Figure 1.** A traffic light control system with strict timing requirements.

**Ref:** https://medium.com/dialog-semiconductor/traffic-signal-controller-bf921278809d

# Common Misconceptions

- **Misconception 1**: Real-time means "fast." (Not always; it means predictable timing.)
- **Misconception 2**: All embedded systems are real-time. (Not necessarily.)

| Real Time Requirements | Non Real Time Requirements |
|---|---|
| Big data is required in the entertainment industry for streaming live content, live music, podcast etc. | Non real time big data field servers for purposes which are not time bound. Examples of such requirements are ML training, background data analytics, data analytics over long duration. |
| Extremely powerful systems are needed to handle such requirements with additional resources on standby for fault tolerance. | Moderately powerful systems can be used to match non real time requirements. |
| High speed network connection is required. | Standalone systems can be used. |
| High accuracy is required for sensitive tasks such as rocket launches, money transfers, medical equipment etc. | Correction procedures can be run later on as well. |

**Figure 2.** A comparison of a real-time and a non-real-time system.

**Ref:** https://www.naukri.com/code360/library/real-time-and-non-real-time-requirements

# Evolution of Real-Time Systems

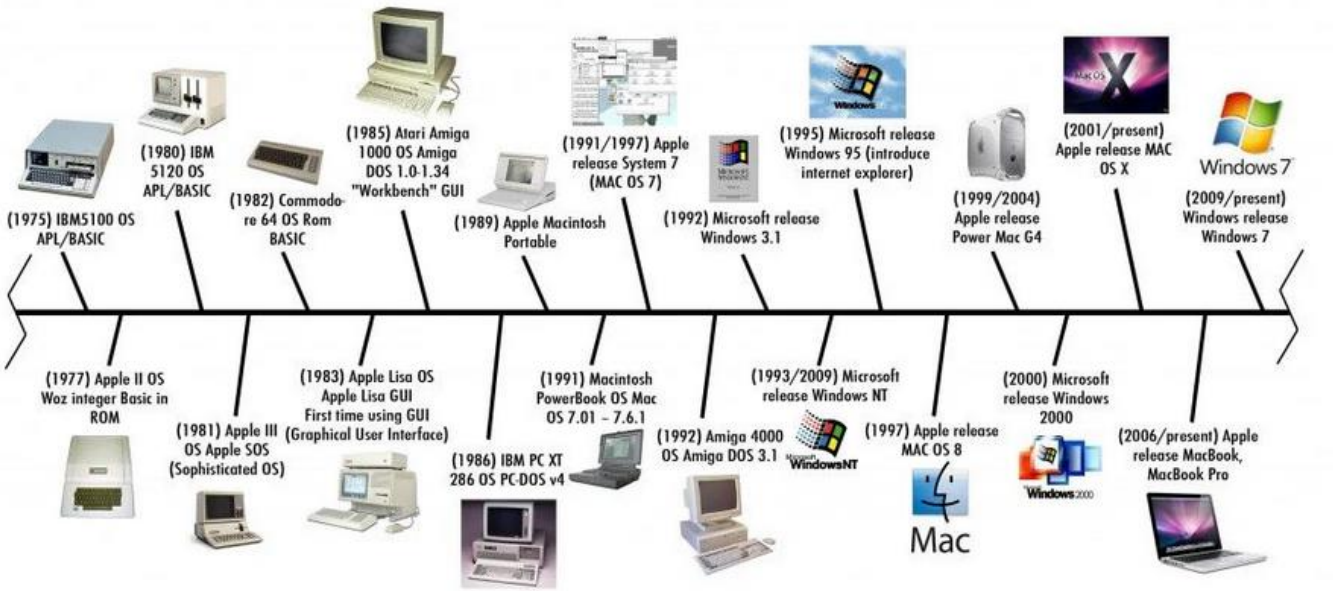


**Figure 3.** A timeline showing major advancements in real-time computing.

**Ref:** https://softwareg.com.au/blogs/computer-hardware/history-of-computer-hardware-timeline

1. From simple mechanical timers to modern AI-driven control systems.

2. Growth in applications: robotics, medical devices, and industrial automation.

# Basics of Processor Architecture

## Chapter 2: Hardware for Real-Time Systems

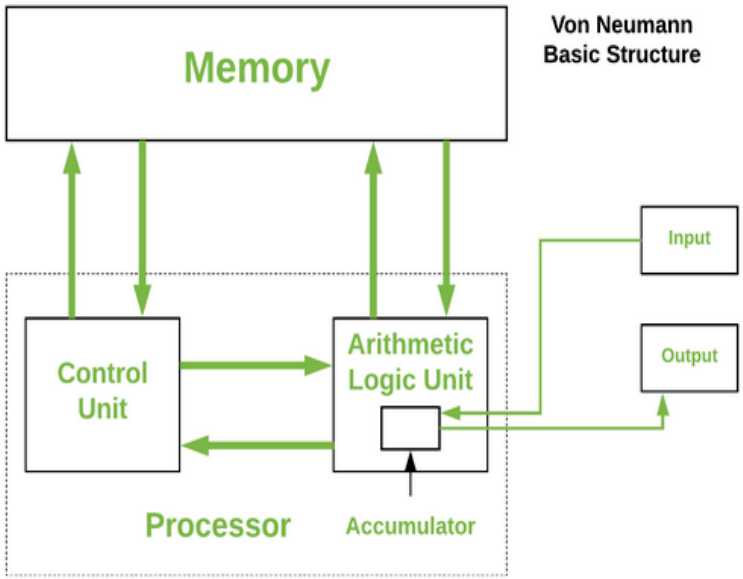


**Figure 4.** Von Neumann Architecture
**Ref:** https://www.geeksforgeeks.org/computer-organization-von-neumann-architecture/

•**Von Neumann Architecture**: Single memory for instructions and data.

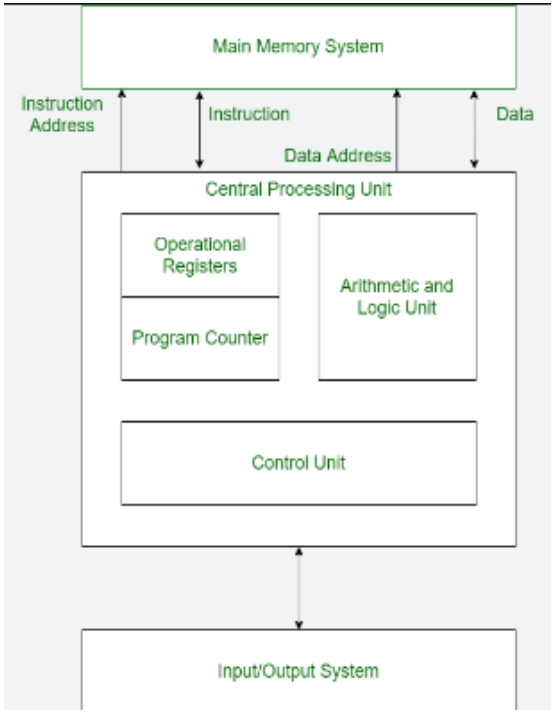


**Figure 5.** Harvard Architecture
**Ref:** https://www.geeksforgeeks.org/harvard-architecture/

•**Harvard Architecture**: Separate memory for instructions and data (faster).

# Instruction Processing & I/O Considerations

CPU processes instructions in steps: Fetch → Decode → Execute.

**Interrupts**: How CPUs handle urgent tasks efficiently.

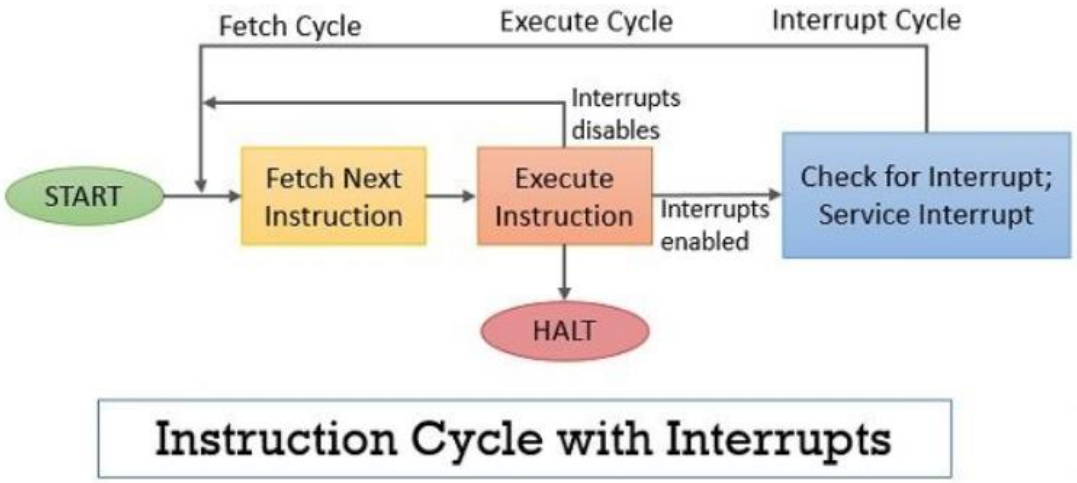


**Figure 6.** Flowchart of instruction execution cycle with Interrupts.

**Ref:** https://binaryterms.com/instruction-cycle.html

# Memory Organization in Real-Time Systems

RAM (volatile) vs. ROM (non-volatile).

**Cache Memory**: Faster access for commonly used data.
**Hierarchical memory**: Helps balance speed and cost.

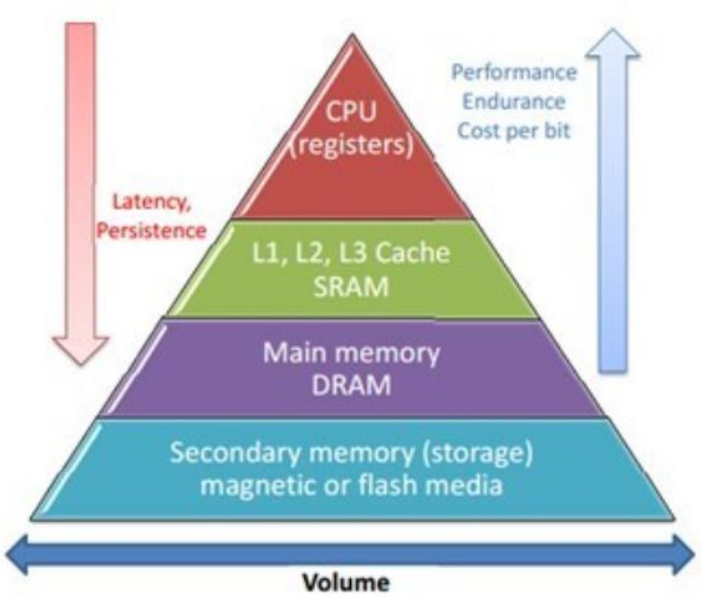


**Figure 7.** A pyramid showing different memory types.
**Ref:** https://www.researchgate.net/figure/The-memory-hierarchy-pyramid_fig1_319529366

# Multi-Core Processors & Pipelining

Multi-core CPUs process tasks in parallel, increasing efficiency.

**Pipelining**: Breaking tasks into smaller steps to work on them simultaneously.

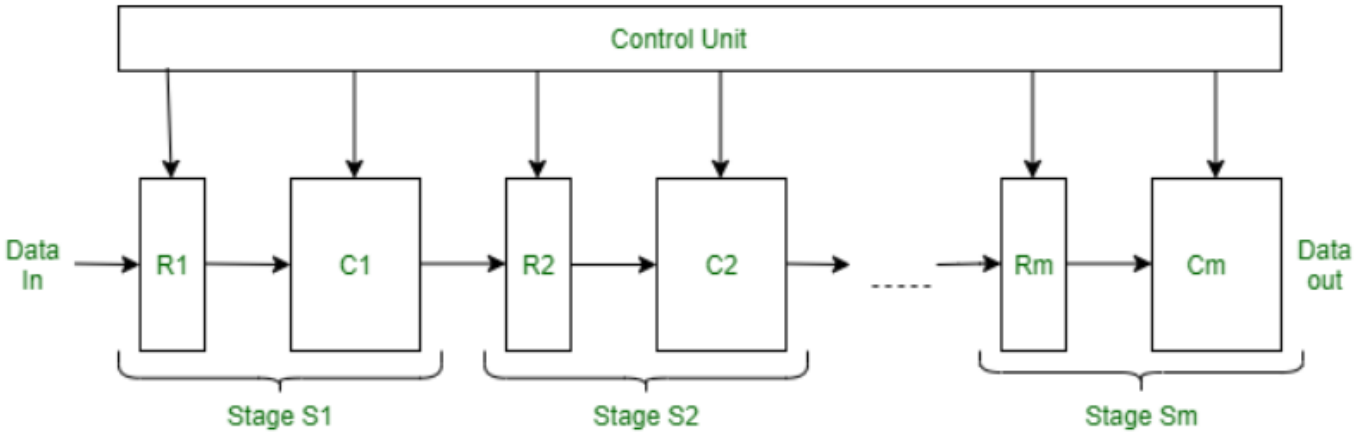


Figure - Structure of a Pipeline Processor

**Figure 8.** A pipeline processing diagram.
**Ref:** https://www.geeksforgeeks.org/pipelined-architecture-with-its-diagram/

# Microprocessors vs. Microcontrollers

- **Microprocessor**: General-purpose CPU (used in computers).

- **Microcontroller**: Includes CPU, memory, and I/O (used in embedded systems)

# What is an RTOS?

## Chapter 3: Real-Time Operating Systems (RTOS)

•Special software that manages tasks with strict timing rules.

•**Examples**: FreeRTOS, VxWorks, QNX.

| Real time operating system | General purpose operation system |
|---|---|
| The time response of the RTOS is deterministic. | The time response of the GPOS is not deterministic. |
| The real time operating system optimizes memory resources. | The GPOS does not optimize the memory resource. |
| The RTOS mainly used in the embedded system. | GPOS mainly used in PC, server, tablets, and mobile phones. |
| The real time operating system has task deadline. | The general-purpose operating system has no task deadline. |
| It doesn't have large memory. | It has large memory. |
| RTOS is designed and developed for a single-user environment. | GPOS is designed for multiuser environment. |
| Example: Free-RTOS, Contiki source code, etc. | Example: Linux, Windows, IOS, etc. |

**Figure 9.** Comparison of general OS vs. RTOS task scheduling.
**Ref:** https://www.investopedia.com/terms/g/gantt-chart.asp

**Figure 9.** Comparison of general OS vs. RTOS task scheduling.
**Ref:** https://medium.com/@aditya.bonte20/difference-between-rtos-and-gpos-11d1990044ec

# Task Scheduling in RTOS

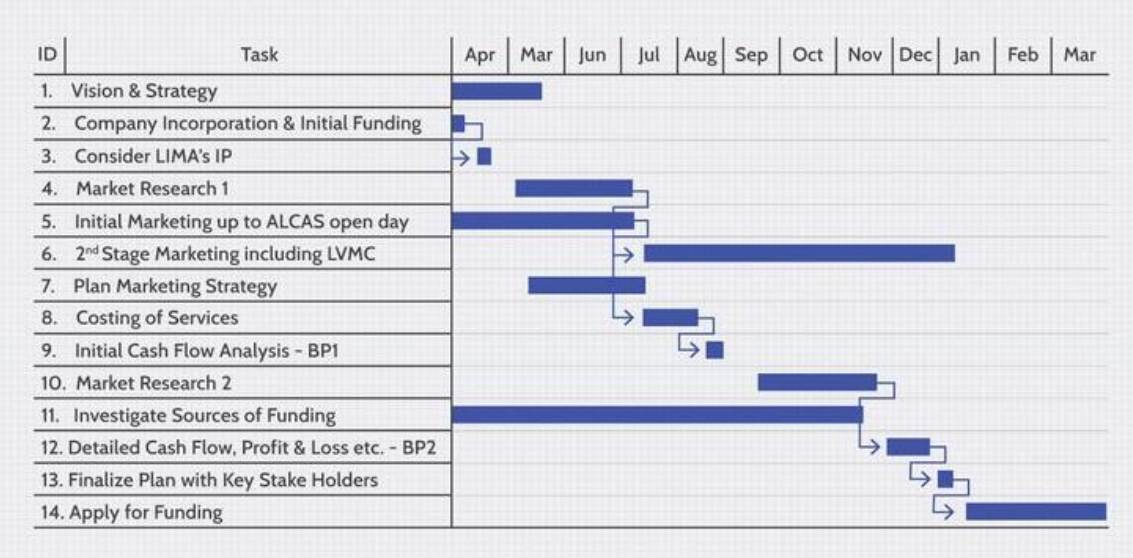


**Figure 10.** Comparison of general OS vs. RTOS task scheduling.
**Ref:** https://www.investopedia.com/terms/g/gantt-chart.asp

**Round-robin scheduling**: Equal time for each task.

**Priority scheduling**: Higher-priority tasks execute first.

**Image**: Gantt chart showing different scheduling strategies.

# Synchronization and DeadlocksSemaphores: Prevent two tasks from using the same

1. **Semaphores**: Prevent two tasks from using the same resource at the same time.

2. **Deadlock**: Two tasks waiting for each other to finish (causing a system freeze).

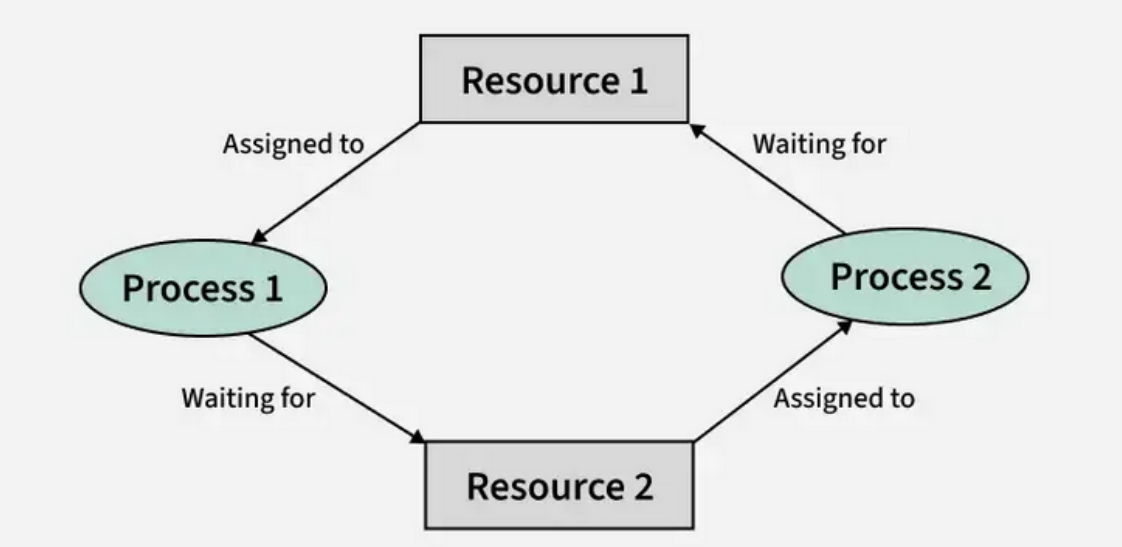


**Figure 12.** A diagram of two processes stuck in a deadlock.
**Ref:** https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/

**Ada**: Designed specifically for real-time safety-critical applications.

**C and C++**: Most used for embedded systems.

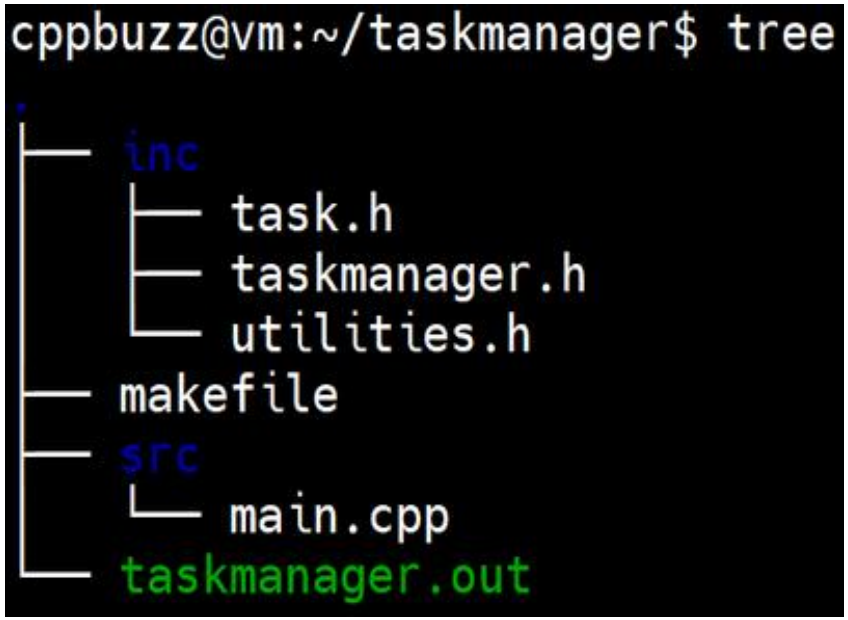


**Figure 13.** Code snippet showing real-time task management in C
**Ref:** https://www.cppbuzz.com/c++/projects/c++-mini-project-on-task-manager-with-source-code

# Object-Oriented vs. Procedural Programming

**Procedural**: Direct and simple (C, Assembly).

**Object-Oriented**: Reusable and modular (C++, Java).

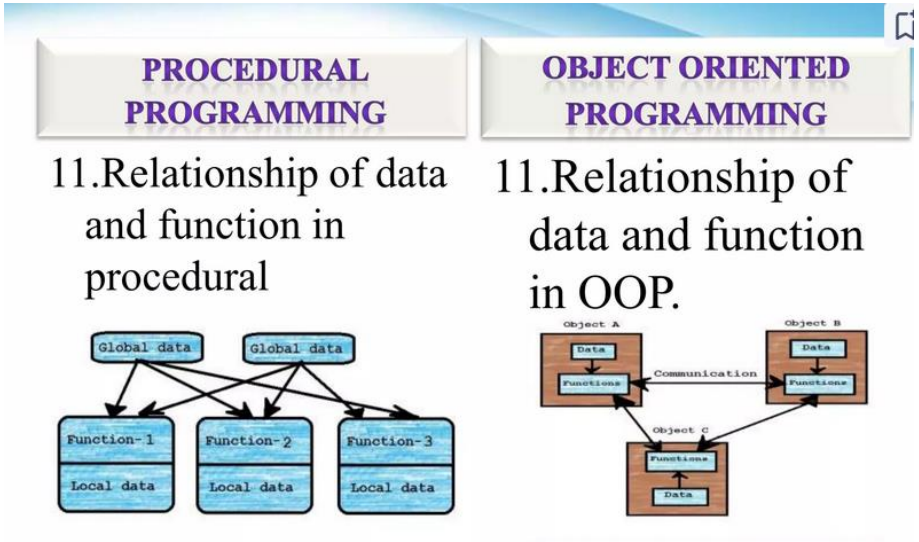


**Figure 14.** A side-by-side comparison of an OOP class vs. procedural function.

**Ref:** https://www.slideshare.net/abhishekkmr_92/ak-procedural-vs-oop#8

- **Clearly Define System Requirements**:
  - Articulate what the system must do to meet user needs.
  - Ensure requirements are specific, measurable, attainable, relevant, and time-bound (SMART).
- **Provide Specific Examples**:
  - Example: "The system must respond within 2ms."
  - Use clear, concise language to avoid ambiguity.
- **Use Structured Requirement Document Format**:
  - Implement a standardized format for requirement documentation.
  - Ensure consistency and clarity across all system requirements.
- **Incorporate Visuals for Clarity**:
  - Example: Include images of a structured requirement document format.
  - Use diagrams and flowcharts to enhance understanding.
- **Review and Validate Requirements**:
  - Conduct reviews with stakeholders to ensure requirements are accurate and complete.
  - Validate requirements through prototyping or simulation if possible.
- **Ensure Traceability**:
  - Maintain traceability of requirements throughout the project lifecycle.
  - Link requirements to their corresponding design, implementation, and testing stages.

# Formal and Semiformal Specification Methods

•**Statecharts**: Graphical representation of system states.

•**Petri Nets**: Used for modeling concurrent processes.

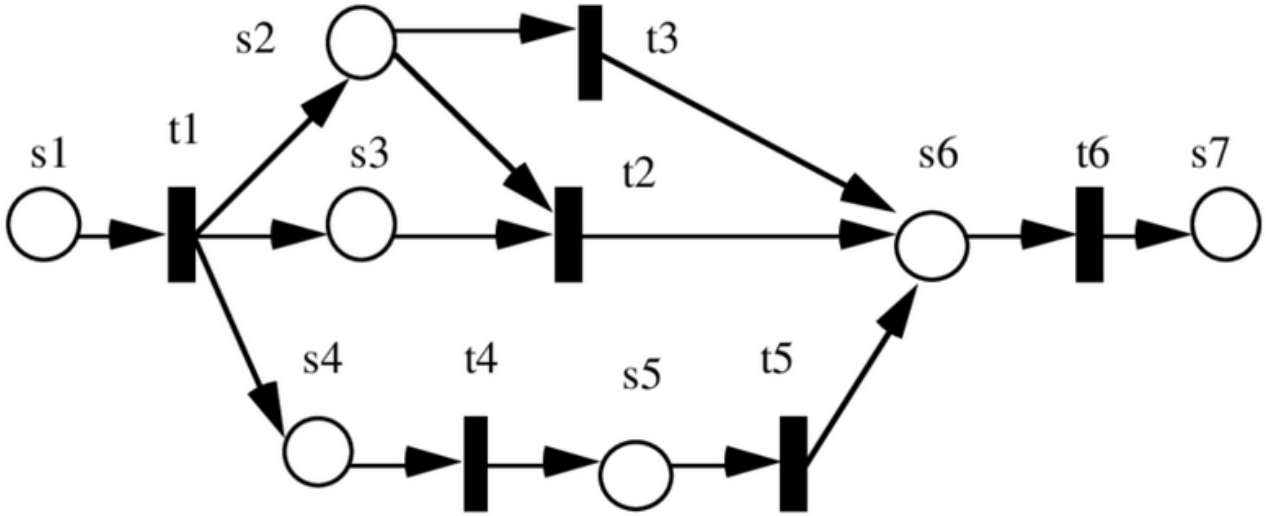


**Figure 15.** A sample Petri Net diagram.
**Ref:** https://www.researchgate.net/figure/An-Example-of-Petri-Nets-Graph_fig4_228713375

•Benefits: Modularity, reusability, and scalability.

•**Design patterns**: Common solutions for recurring problems.

**Benefits of Design Patterns**

•**Flexibility**:

- Design patterns allow systems to adapt to changes with minimal impact.
- This flexibility supports the evolution of systems as new requirements emerge.

•**Cost-Effectiveness**:

- By reusing established patterns, development and maintenance costs are reduced.
- Efficient solutions lower the need for extensive rework, saving time and resources.

# Life Cycle Models for Real-Time Software

**Agile Model**: Iterative, allows for flexibility.

**Waterfall Model**: Step-by-step, rigid process.

| Waterfall | Agile (Scrum) |
|---|---|
| Feasibility evaluation takes a long phase and is done in advance to avoid reworking in the next project phases. | Feasibility test takes a shorter while considerably. Clients are engaged in the early project phase to get the buy-in and refine the needs in the long run. |
| Project planning is done at the beginning of the project and is not open to any changes later on. | The plan is not given the foremost priority and is done during sprint planning. Modifications are welcome except during an active sprint. |
| Project progress gets monitored according to the project plan. | The development gets tallied in each sprint. |
| Only the project managers communicate and carry out progress review meetings weekly/ monthly. | Communication is frequent, face-to-face, and clients also participate throughout the project. |
| Roles are not interchangeable once distributed among project team members. | You can switch roles quickly, and the team can work in cycles. |
| Documentation gets a lot of emphases and that is pretty comprehensive. | There's a need to file requirements, build designs, and write test plans to promote working software delivery. |

**Figure 16.** Waterfall vs. Agile model comparison.
**Ref:** https://www.edrawsoft.com/agile-vs-waterfall.html

**Networking Technologies in Real-Time Systems**

- **Wired Networks**:
  - Examples: Ethernet, CAN bus.
  - Provide reliable and high-speed data transmission.
  - Commonly used in environments where stability and low latency are critical.
- **Wireless Networks**:
  - Examples: Wi-Fi, Zigbee.
  - Offer flexibility and ease of installation.
  - Suitable for applications requiring mobility and remote access.

**Industrial Communication Protocols**

- **Fieldbus Networks**:
  - Used for industrial automation communication.
  - Facilitate real-time data exchange between control systems and devices.
- **Real-Time Network Protocols**:
  - Designed to meet the stringent timing requirements of real-time systems.
  - Ensure timely and deterministic communication for critical applications.

# Time-Triggered Communication

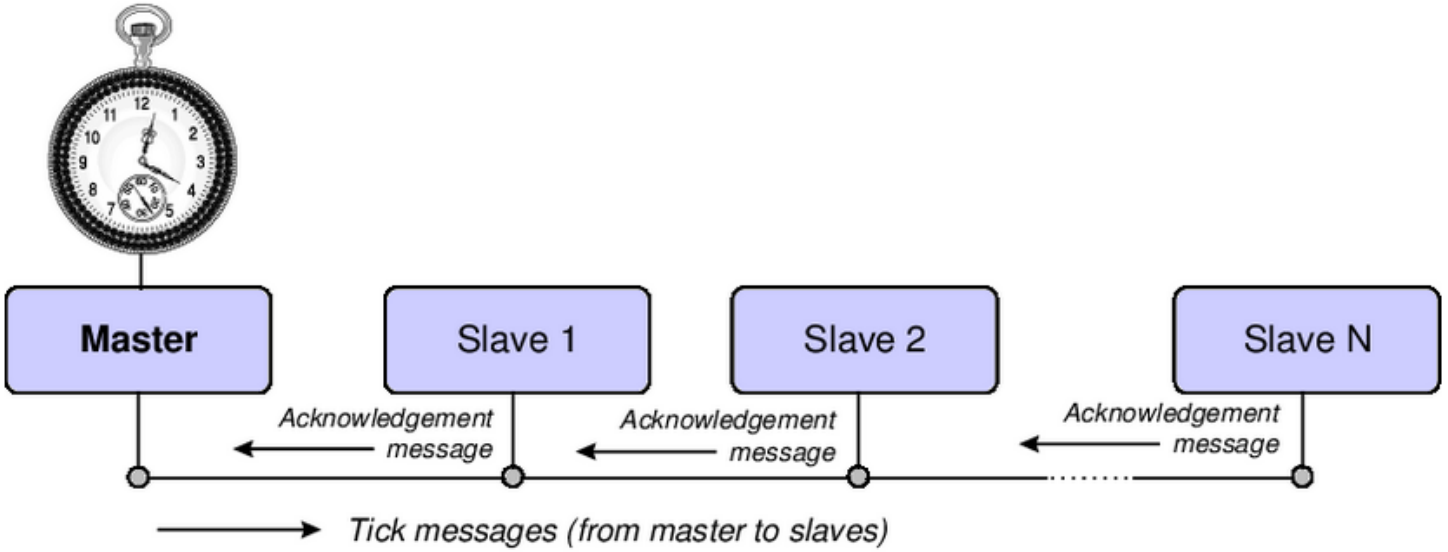


**Figure 17.** A time-triggered message scheduling chart.
**Ref:** https://www.researchgate.net/figure/Simple-architecture-of-time-triggered-shared-clock-scheduler_fig1_308611848

- Ensures messages are sent at fixed intervals.
- Used in **automotive and aerospace systems.**

# Measuring Real-Time Performance

## Chapter 8: Performance Analysis and Optimization

**Key Performance Metrics in Real-Time Systems**

- **Response Time**:
  - Measures how quickly a system can respond to an input or event.
  - Critical for applications where delays can lead to failure or safety risks.
- **Jitter**:
  - Represents the variability in response time.
  - Minimizing jitter is essential to maintain consistent performance in real-time systems.
- **Deadline Miss Rate**:
  - Indicates the frequency of missed deadlines.
  - A lower deadline miss rate is crucial for ensuring the reliability of time-critical applications.

**Example Application**

- **Airbag Deployment System**:
  - Requires extremely low response time to deploy airbags in milliseconds during a collision.
  - Response time analysis ensures the system meets stringent safety standards.
  - Minimized jitter and low deadline miss rate are vital to guarantee timely airbag deployment in every scenario.

# Optimization Strategies

**Code optimization**: Reducing execution time.

**Hardware acceleration**: Using GPUs or FPGAs.

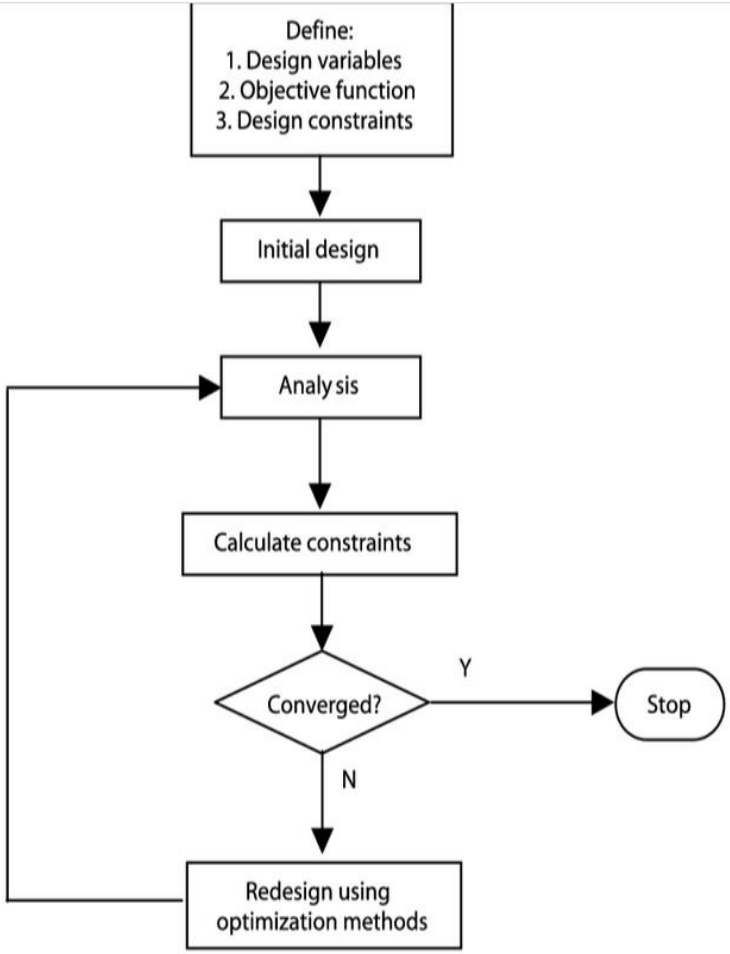


**Figure 18.** Flowchart of an optimized system process.

**Ref:** https://www.researchgate.net/figure/Flow-chart-of-the-optimization-procedure_fig1_246044375

**Examples of Critical Real-Time Systems**

- **Medical Devices**:
  - Examples: Pacemakers, insulin pumps, and ventilators.
  - Require precise timing and reliability to ensure patient safety.
- **Nuclear Reactors**:
  - Control and safety systems monitor and manage reactor operations.
  - Timely responses are crucial to prevent accidents and ensure safe operation.

**Fail-Safe Mechanisms**

- **Redundant Backup Systems**:
  - Implement multiple layers of redundancy to handle failures.
  - Ensure that a backup system can take over seamlessly if the primary system fails.
- **Watchdog Timers**:
  - Continuously monitor system performance.
  - Trigger corrective actions if the system becomes unresponsive or fails.

# Cybersecurity in Real-Time Systems

**Security in Real-Time Systems**

- **Threats**:
  - **Hacking**:
    - Unauthorized access to systems, potentially leading to data breaches and control overrides.
  - **Malware**:
    - Malicious software designed to disrupt, damage, or gain unauthorized access to systems.
  - **Data Corruption**:
    - Unintended alterations to data that can compromise system integrity and functionality.
- **Solutions**:
  - **Encryption**:
    - Protects data by converting it into a coded format, making it inaccessible to unauthorized users.
  - **Firewalls**:
    - Act as barriers to prevent unauthorized access to or from private networks.
  - **Access Controls**:
    - Restrict user access to systems and data based on predefined permissions and roles.
  - **Regular Updates and Patches**:
    - Keep systems up to date with the latest security measures and fixes for known vulnerabilities.

# Future of Real-Time Systems

**Advanced Technologies in Real-Time Systems**

- **AI-Powered Real-Time Decision-Making**:
  - Utilizes machine learning algorithms to analyze data and make decisions instantly.
  - Enhances efficiency and accuracy in applications like autonomous vehicles and predictive maintenance.

- **Quantum Computing for Real-Time Processing**:
  - Explores the use of quantum mechanics to perform computations at unprecedented speeds.
  - Potentially revolutionizes real-time processing capabilities, enabling solutions to complex problems beyond the reach of classical computing.

# Conclusion & Final Thoughts

Real-time systems have become indispensable in the landscape of modern technology, underpinning a wide range of applications that demand precise timing and immediate responsiveness. These systems are designed to perform tasks within strict time constraints, ensuring that operations are executed in a timely manner to achieve desired outcomes. From managing critical functions in healthcare devices, such as pacemakers and ventilators, to controlling industrial automation processes in manufacturing plants, real-time systems play a pivotal role in maintaining the safety, efficiency, and reliability of various technological ecosystems.

# Before you go…..

# Let’s test your knowledge

**Fundamentals of Real-Time Systems**

1. What is the defining characteristic of a real-time system?
A) High processing power
B) Execution of tasks in any order
C) Guaranteed response within a specific time
D) Use of advanced programming languages

2.Which of the following is an example of a **hard real-time system**?
A) Video streaming application
B) Online shopping website
C) Anti-lock braking system (ABS) in cars
D) Cloud-based database

3.Which of the following is **NOT** a real-time system scheduling approach?
A) Rate-Monotonic Scheduling (RMS)
B) Earliest Deadline First (EDF)
C) Round-robin scheduling
D) Pipelining

4.n real-time systems, **latency** refers to:
A) The time required to execute all tasks
B) The delay between task initiation and completion
C) The total system execution time
D) The number of completed tasks per second

**Hardware for Real-Time Systems**

5.What is the main advantage of the **Harvard architecture** over the **Von Neumann architecture**?
A) Uses separate memory for instructions and data
B) Requires less hardware
C) Has faster data processing in general-purpose computing
D) Reduces power consumption significantly

6.Which of the following memory types provides the **fastest** access speed?
A) RAM
B) Hard disk
C) Cache memory
D) Flash storage

7.Which type of processor is typically used in **real-time embedded systems**?
A) General-purpose CPU
B) Digital Signal Processor (DSP)
C) Quantum Processor
D) Graphics Processing Unit (GPU)

**Real-Time Operating Systems (RTOS)**

8.What is the main role of an **RTOS (Real-Time Operating System)**?
A) To optimize memory usage
B) To manage hardware components efficiently
C) To schedule tasks with strict timing constraints
D) To improve system security

9.What happens in a **priority inversion** scenario?
A) A lower-priority task prevents a higher-priority task from executing
B) The system reboots unexpectedly
C) The highest-priority task executes first
D) The CPU enters an idle state

10.Which of the following real-time scheduling algorithms **dynamically assigns priorities based on task deadlines**?
A) Rate-Monotonic Scheduling (RMS)
B) First-Come-First-Serve (FCFS)
C) Earliest Deadline First (EDF)
D) Round-robin

11.Which of these programming languages is **commonly used** in real-time systems?
A) Python
B) Ada
C) JavaScript
D) SQL

12.Why is **garbage collection** generally **avoided** in real-time systems?
A) It increases CPU efficiency
B) It can cause unpredictable delays
C) It improves memory utilization
D) It reduces power consumption

**Real-Time System Design and Engineering**

13.In real-time system development, **Worst-Case Execution Time (WCET)** refers to:
A) The average execution time of a task
B) The shortest execution time of a task
C) The maximum time required for a task to complete
D) The total time for all tasks in the system

14.Which method is commonly used to **model real-time systems** and their state transitions?
A) Entity-Relationship Diagrams (ERD)
B) Unified Modeling Language (UML)
C) Finite State Machines (FSM)
D) Relational Databases

15.What is the primary goal of **real-time system performance analysis**?
A) Reducing power consumption
B) Ensuring that all tasks meet their deadlines
C) Increasing execution speed
D) Eliminating system logs

**Real-Time Scheduling and Synchronization**

16.What is the **main disadvantage** of **Rate-Monotonic Scheduling (RMS)**?

A) It does not support preemptive multitasking
B) It is only effective for dynamic task priorities
C) It cannot guarantee schedulability if CPU utilization exceeds a certain limit
D) It can only schedule one task at a time

17.What is the purpose of a **semaphore** in real-time systems?
A) Prevent deadlocks
B) Synchronize concurrent tasks
C) Allocate memory efficiently
D) Handle interrupts

18.What is the key advantage of **Earliest Deadline First (EDF)** scheduling?
A) Simple to implement
B) Maximizes CPU utilization by prioritizing tasks with the nearest deadline
C) Does not require task priority levels
D) Ensures 100% predictability of execution

**Real-Time System Case Studies**

19.A **pacemaker** that regulates heartbeats is an example of a:
A) Hard real-time system
B) Soft real-time system
C) Best-effort system
D) Cloud-based system

20. In an **automated car braking system**, what happens if a real-time deadline is missed?
A) The system continues operating normally
B) The car will gradually slow down
C) The system fails, causing potential accidents
D) The system restarts automatically

# Thank you

**Answer Key:**

1. C) Guaranteed response within a specific time
2. C) Anti-lock braking system (ABS) in cars
3. D) Pipelining
4. B) The delay between task initiation and completion
5. A) Uses separate memory for instructions and data
6. C) Cache memory
7. B) Digital Signal Processor (DSP)
8. C) To schedule tasks with strict timing constraints
9. A) A lower-priority task prevents a higher-priority task from executing
10. C) Earliest Deadline First (EDF)
11. B) Ada
12. B) It can cause unpredictable delays
13. C) The maximum time required for a task to complete
14. C) Finite State Machines (FSM)
15. B) Ensuring that all tasks meet their deadlines
16. C) It cannot guarantee schedulability if CPU utilization exceeds a certain limit
17. B) Synchronize concurrent tasks
18. B) Maximizes CPU utilization by prioritizing tasks with the nearest deadline
19. A) Hard real-time system
20. C) The system fails, causing potential accidents