REAL-TIME OPERATING SYSTEMS

Sewmehon E.

CONTENTS

1. Introduction
2. General Purpose Operating System
3. Non-Real-Time Systems
4. What is a RTOS?
5. Types of RTOS
6. Basic Functions of RTOS Kernel
7. Task Management
8. Task States
9. Interrupt Handling
10. Memory Management
11. Exception Handling
12. Task Synchronization
13. Task Scheduling
14. Example of Task Scheduling (RR)
15. Time Management
16. Existing RTOS Categories
17. RT Linux: An Example
18. Conclusion
19. Real-Time Software Designs
20. Real-Time Systems

Introduction(1)

- General Purpose Operating System
- Non-Real-Time systems
- RTOS
- Types of RTOS
- Basic Functions of RTOS kernel
- RTOS Categories

General Purpose Operating System

1. An interface between users and hardware
2. Controlling and allocating memory
3. Controlling input and output devices
4. Managing file systems
5. Facilitating networking

Non-Real-Time Systems

1. Non-Real-Time systems are the operating systems most often used
2. No guaranteed worst case scheduling jitter
3. System load may result in delayed interrupt response
4. System response is strongly load dependent
5. System timing is an unmanaged resource

What is a RTOS?

RTOS is a pre-emptive multitasking operating system intended intended for real-time applications

Able to determine task's completion time

Guarantees task completion at a set deadline.

Predictable OS timing behavior

A system of priority inheritance has to exist

Soft Real-Time System

The soft real-time definition allows for frequently missed deadlines

If the system fails to meet the deadline, possibly more than once, once, the system is not considered to considered to have failed

Example: Multimedia streaming, Video games

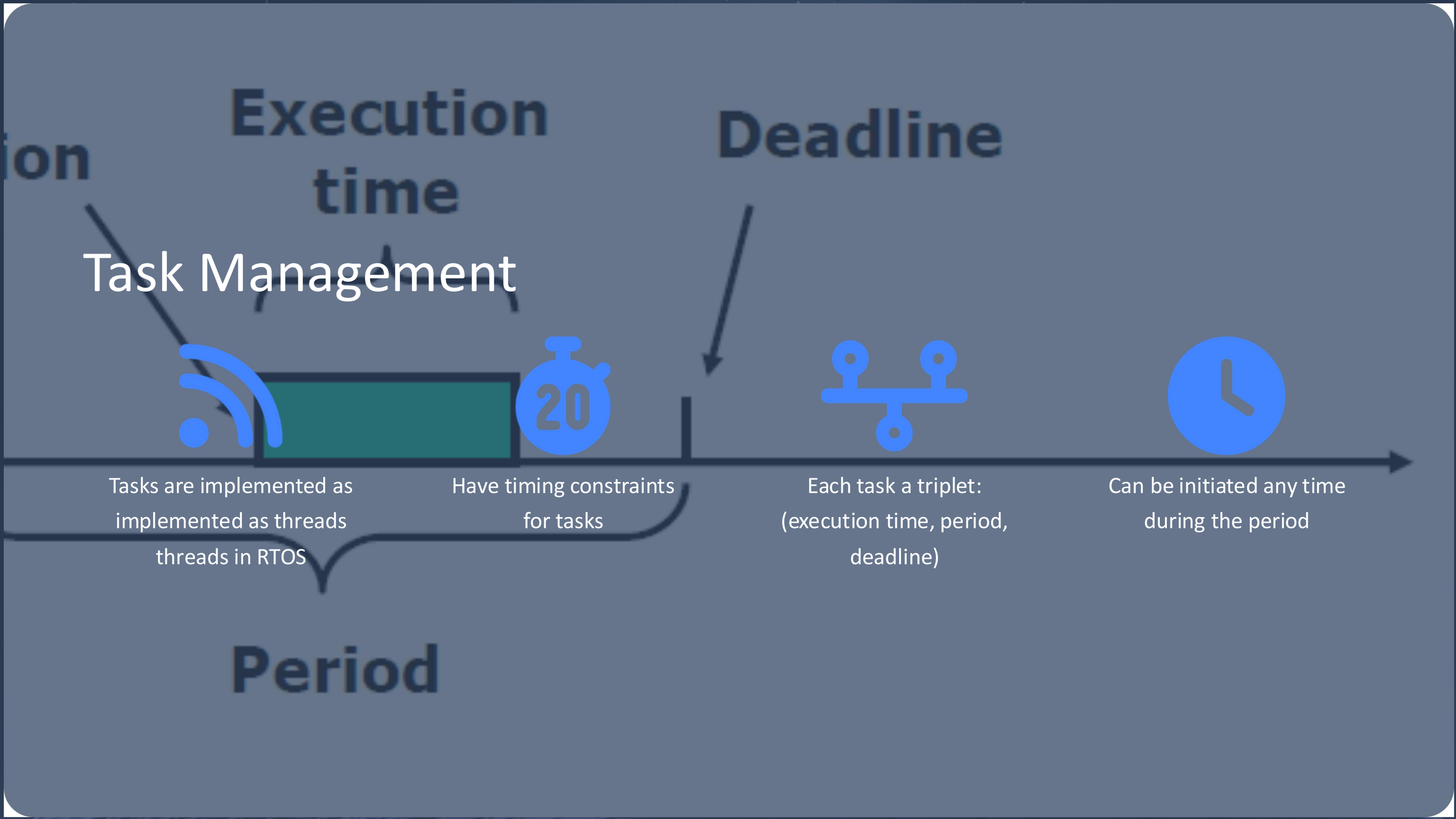
Hard Real-Time System

1. A hard real-time system guarantees that real-time tasks be completed completed within their required deadlines
2. Failure to meet a single deadline may lead to a critical system failure
3. Examples: air traffic control, vehicle subsystems control, medical systems

Basic Functions of RTOS Kernel

1. Task Management
2. Interrupt handling
3. Memory management
4. Exception handling
5. Task synchronization
6. Task scheduling
7. Time management

Task Management



Tasks are implemented as implemented as threads threads in RTOS

Have timing constraints for tasks

Each task a triplet: (execution time, period, deadline)

Can be initiated any time during the period

Task States

Idle: task has no need for computer time

Ready: task is ready to go active, but waiting for processor time

Running: task is executing associated activities

Waiting: task put on temporary hold to allow lower priority task chance to execute

Suspended: task is waiting for resource


Interrupt Handling

Types of Interrupts

Asynchronous or hardware interrupt

Synchronous or software interrupt

Very low Interrupt latency

The ISR of a lower-priority interrupt may be blocked by the ISR of a high-priority

Memory Management

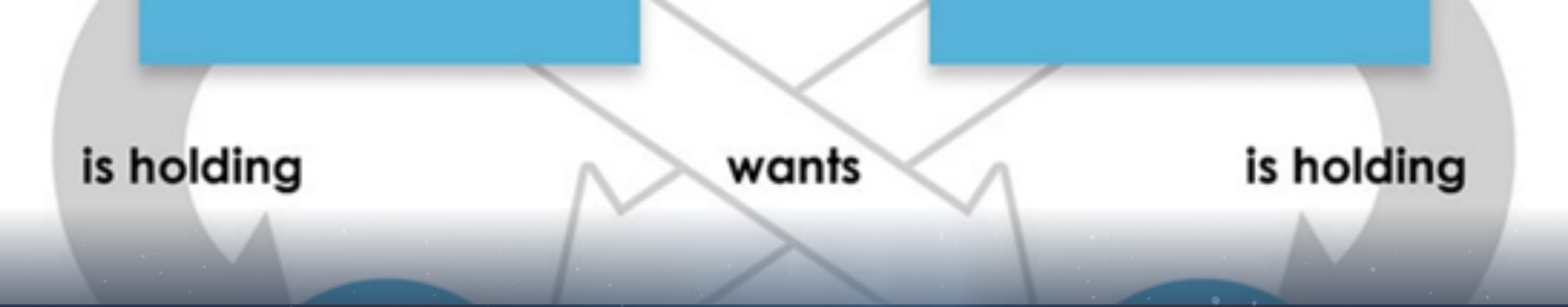
RTOS may disable the support to the dynamic block allocation

When a task is created the RTOS simply returns an already initialized memory location

When a task dies, the RTOS returns the memory location to the pool

No virtual memory for hard RT tasks

Exception Handling



Exception Handling

Exceptions are triggered triggered by the CPU in in case of an error

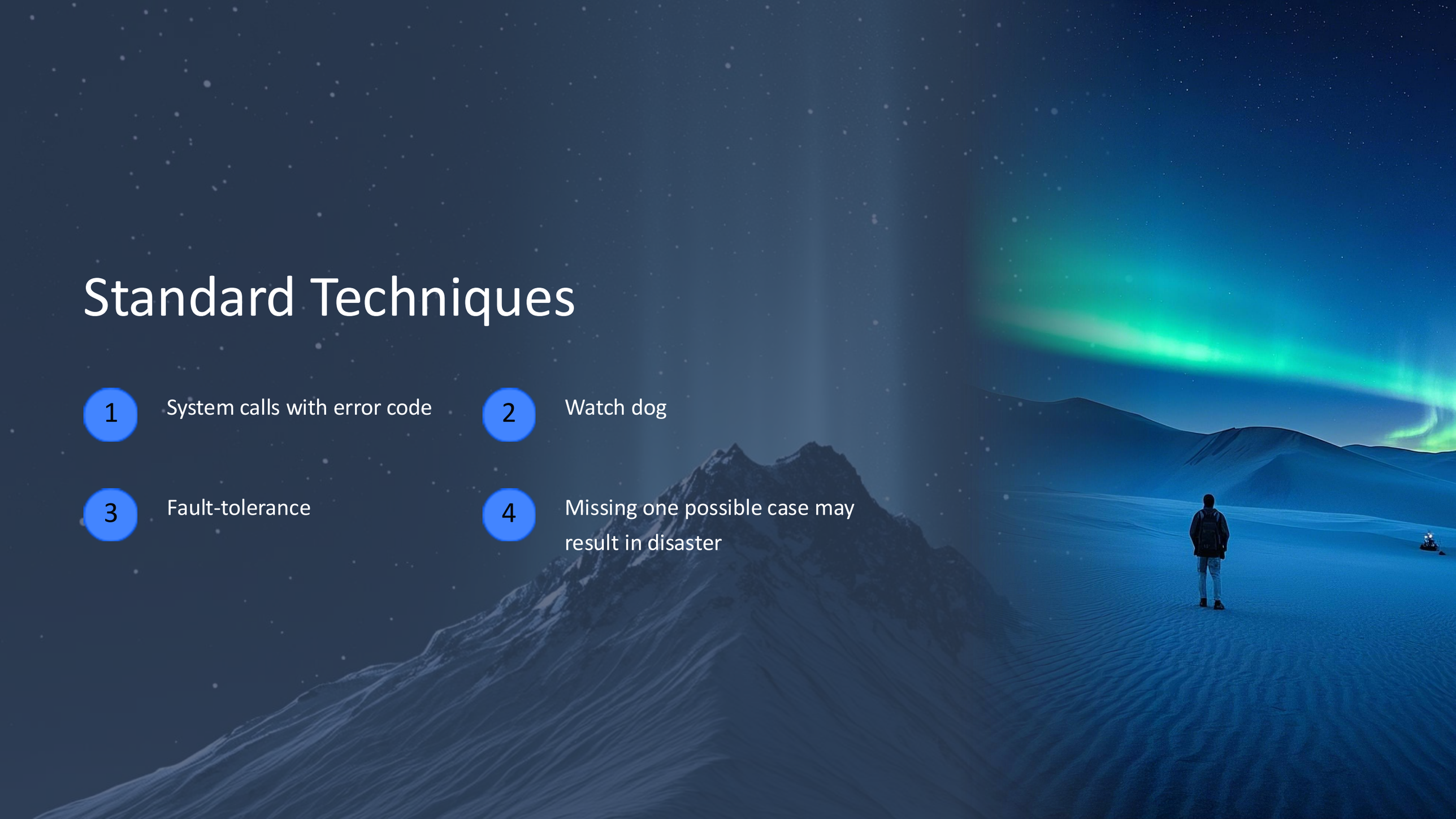
E.g.: Missing deadline, running out of memory, timeouts, deadlocks, divide by zero, etc.

Error at system level, e.g. deadlock

Error at task level, e.g. timeout

Standard Techniques

1. System calls with error code
2. Watch dog
3. Fault-tolerance
4. Missing one possible case may result in disaster



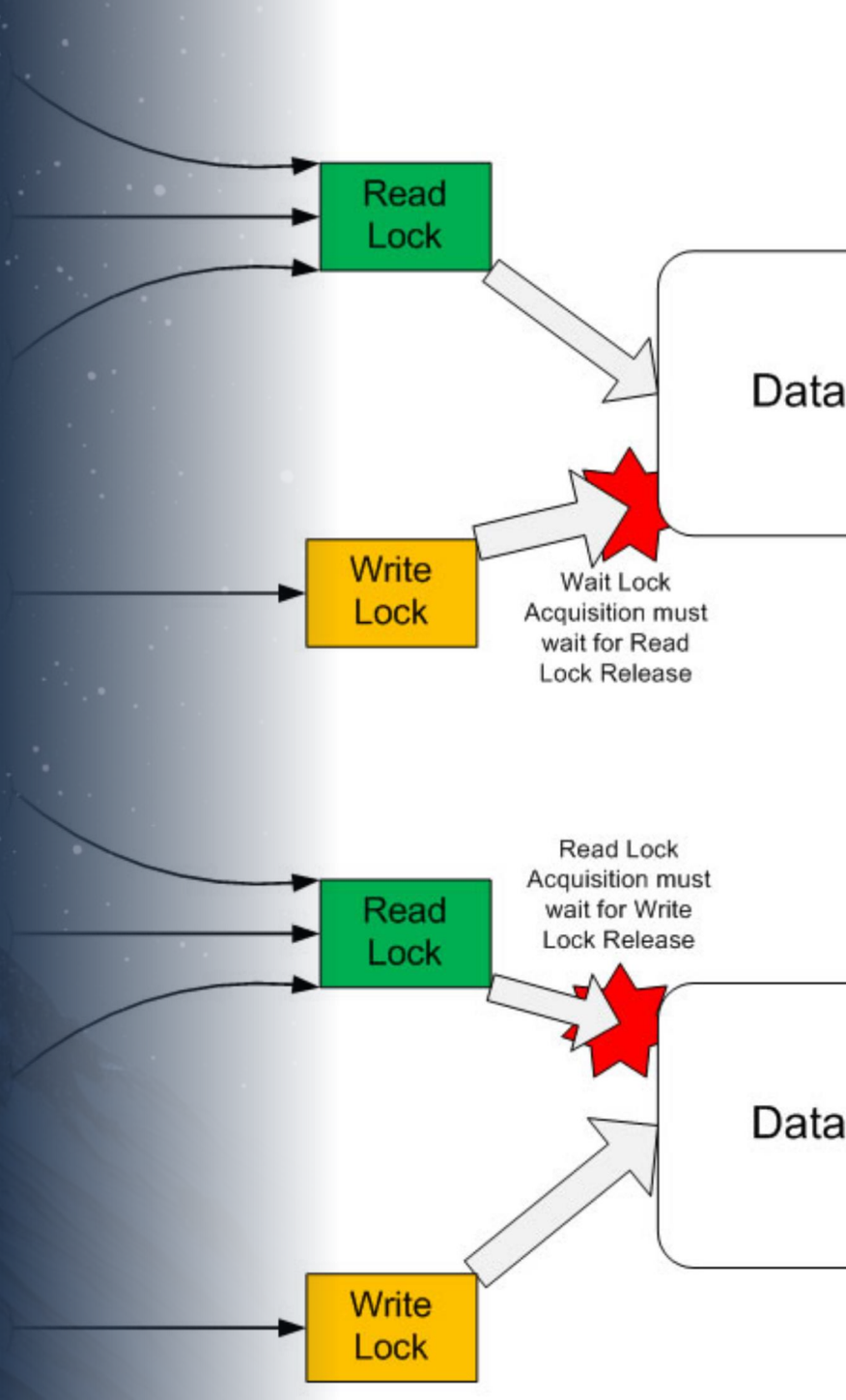
Task Synchronization

Semaphore

Mutex

Spinlock

Read/write locks



Task Scheduling

Scheduler is responsible for time-sharing of CPU among tasks

Priority-based Preemptive Scheduling

Rate Monotonic Scheduling

Earliest Deadline First Scheduling

Round robin scheduling

Priority-Based Preemptive Scheduling

Assign each process a priority

At any time, scheduler runs highest priority process ready to run

Rate Monotonic Scheduling

1. A priority is assigned based on the the inverse of its period
2. Shorter execution periods = higher priority
3. Longer execution periods = lower priority

Earliest Deadline First Scheduling Scheduling

Priorities are assigned according to deadlines

The earlier the deadline, the higher the priority

Priorities are dynamically chosen

Round Robin Scheduling

Designed for time-sharing systems systems

Ready queue treated as a circular buffer

Jobs get the CPU for a fixed time

Process may use less than a full time time slice

Time Management

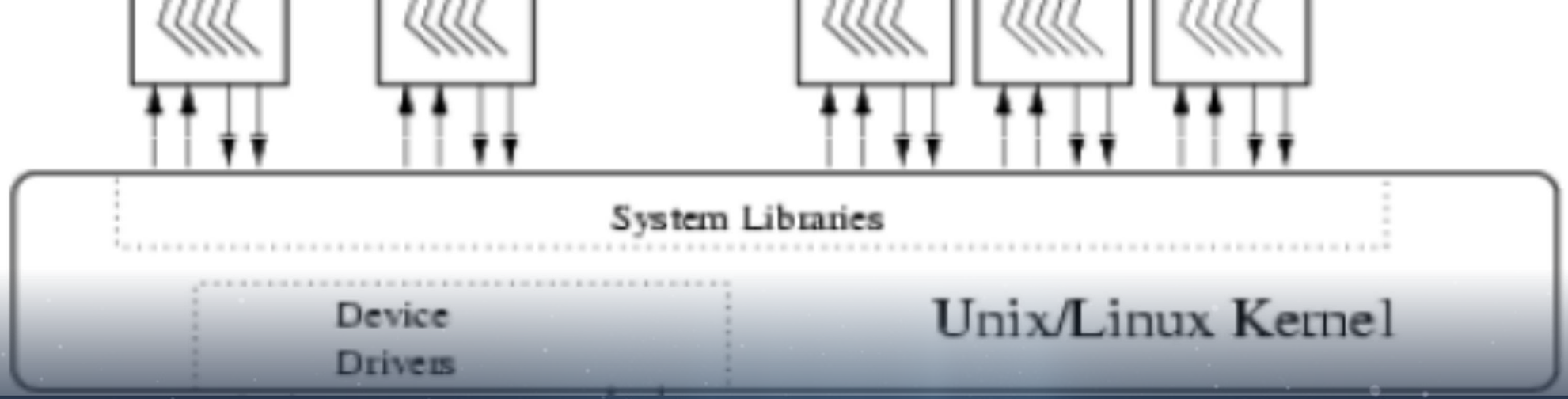
Time interrupt: A high resolution hardware timer is programmed to to interrupt the processor at fixed rate.

Each time interrupt is called a system tick.

The tick may be chosen according to the given task parameters.

Existing RTOS Categories

1. Priority based kernel for embedded applications
2. VxWorks, OSE, QNX
3. Real-Time Extensions of existing time-sharing OS
4. Real-time Linux, Real-time NT
5. Research RT Kernels
6. MARS, Spring



RT Linux: An Example

RT-Linux is an operating system, in which a small real-time kernel co-exists with standard Linux kernel

Conclusion

RTOS is an OS for response time controlled and event controlled processes. The processes have predictable latencies and execute by pre-emptive scheduling.

An RTOS is an OS for the systems having the hard or soft real timing constraints and deadline on the tasks.


Real-Time Software Designs

Real-Time Systems

1. Systems which monitor and control their environment
2. Inevitably associated with hardware devices
3. Sensors: Collect data from the system environment
4. Actuators: Change (in some way) the system's environment
5. Time is critical. Real-time systems MUST respond within specified times.

Stimulus/Response Systems

Given a stimulus, the system must produce a response within a specified time

1

1. Periodic stimuli: Stimuli which occur at predictable time intervals

   For example, a temperature sensor may be polled 10 times 10 times per second

2. Aperiodic stimuli: Stimuli which occur at unpredictable times

   For example, a system power failure may trigger an interrupt which must be processed by the system.

Architectural Considerations

Because of the need to respond to timing demands made by different stimuli/responses, the system architecture must allow for fast switching between stimulus handlers

Timing demands of different stimuli are different so a simple sequential loop is sequential loop is not usually adequate

Real-time systems are usually designed as cooperating processes with a real-time executive controlling these processes

23. A Real-Time System Model

System Elements

Sensors control processes	Collect information from sensors. May buffer information collected in response to a sensor stimulus.
Data processor	Carries out processing of collected information and computes the system response.
Actuator control	Generates control signals for the actuator.

27. Hardware and Software Design

R-T Systems Design Process

Identify the stimuli to be processed and the required responses to these stimuli.

Aggregate the stimulus and response processing into concurrent processes.

For each stimulus and response, identify the timing constraints.

A process may be associated with each class of stimulus and response.

R-T Systems Design Process (cont)

Design algorithms to process each class of stimulus and response. These must meet the given timing requirements.

Design a scheduling system which will which will ensure that processes are are started in time to meet their deadlines.

Integrate using a real-time executive or operating system.

Timing Constraints

May require extensive simulation and experiment to ensure that these are these are met by the system.

May mean that certain design strategies such as object-oriented design cannot be used because of the additional overhead involved.

May mean that low-level programming language features have to be used for performance reasons.

State Machine Modelling

The effect of a stimulus in a real-time system may trigger a transition from one state to another.

However, FSM models lack structure. Even simple systems can have a complex model.

Finite state machines can be used for modelling real-time time systems.

Real-Time Programming

Hard-real time systems may have to be programmed in assembly language to ensure that deadlines are met.

Languages such as C allow efficient programs to be written but do not have constructs to support concurrency or shared resource management.

Ada as a language designed to support real-time systems design includes a general-purpose concurrency mechanism.

Java as a Real-Time Language

1. Java supports lightweight concurrency (threads and synchronized methods) and can be used for some soft real-time systems.
2. Java 2.0 is not suitable for hard RT programming or programming where precise control of timing is required.
3. Not possible to specify thread execution time.
4. Uncontrollable garbage collection.
5. Not possible to discover queue sizes for shared resources.
6. Variable virtual machine implementation.
7. Not possible to do space or timing analysis.

Real-Time Executives

1. Real-time executives are specialised operating systems which manage the processes in the RTS.
2. Responsible for process management and resource (processor and memory) allocation.
3. May be based on a standard RTE kernel which is used used unchanged or modified for a particular application. application.
4. Does not include facilities such as file management.

Executive Components

Real-time clock: Provides information for process scheduling.

Interrupt handler: Manages aperiodic requests for service.

Scheduler: Chooses the next process to be run.

Resource manager: Allocates memory and processor resources.

Despatcher: Starts process execution.

Non-Stop System Components

1. Configuration manager: Responsible for the dynamic reconfiguration of the system software and hardware. Hardware modules may be replaced and software upgraded without stopping the systems.

2. Fault manager: Responsible for detecting software and hardware faults and taking appropriate actions (e.g. switching to backup disks) to ensure that the system continues in operation.

Process Priority

The processing of some types of stimuli must sometimes take priority.

Interrupt level priority: Highest priority which is allocated to processes requiring a very fast response.

Clock level priority: Allocated to periodic processes.

Within these, further levels of priority may be assigned.


Thank You