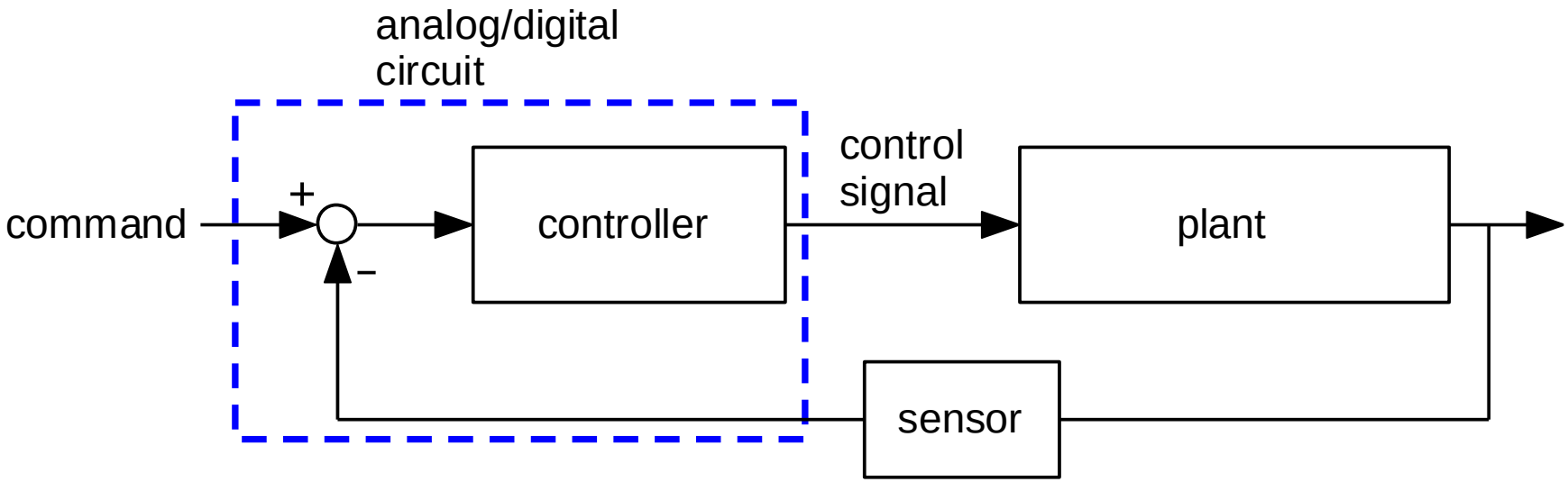
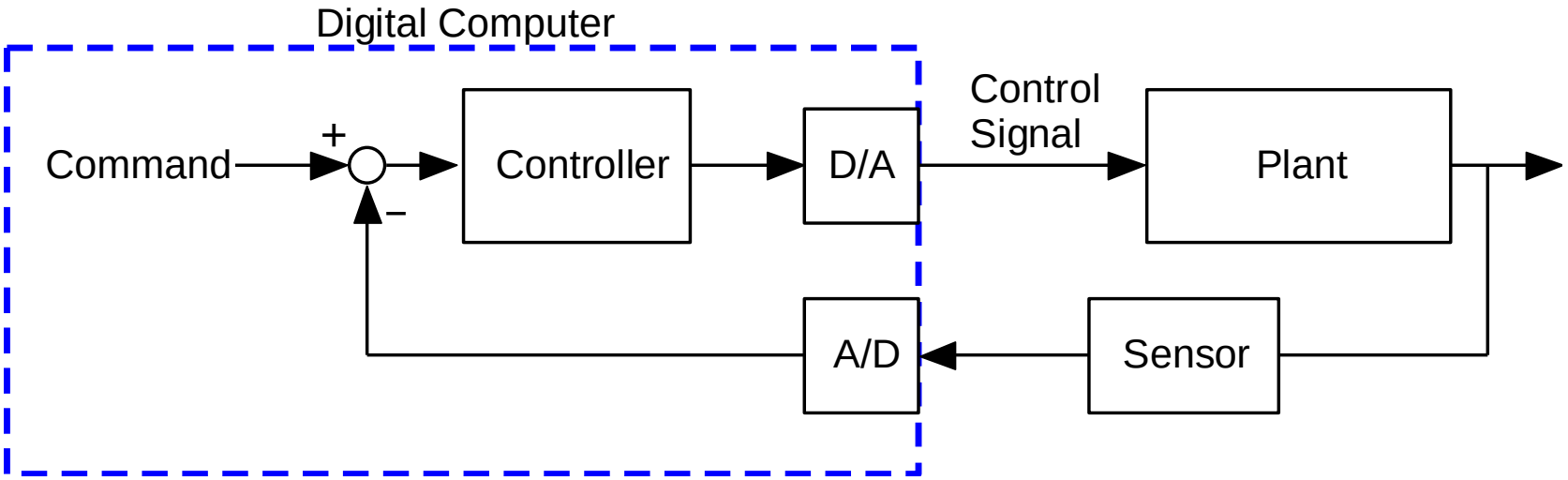
Digital Control & Real-Time

Control System



Digital Control System



Real-Time

- Real-Time is the essential part in digital control

- “A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred.”

- Late correct answer is wrong answer!

Real-Time

- “Real-time in operating systems: The ability of the operating system to provide a required level of service in a bounded response time.”

- POSIX Standard 1003.1

Hard vs Soft Real-Time

- Hard Real-Time
  - Absolute deadlines that must be met
  - Example: Braking system controller
- Soft Real-Time
  - Time tolerance within which an event can occur
  - Example: Multimedia streaming

Real-Time OS

- Multi-threaded and pre-emptible
- Thread priority has to exist
- Must support predictable thread synchronization mechanisms
- A system of priority inheritance must exist

Commercial Real-Time OS

- Wind River Systems
  - VxWorks
  - pSOS
- QNX Software Systems
  - QNX
- Green Hills Software
  - Integrity
- Mentor Graphics
  - VRTX

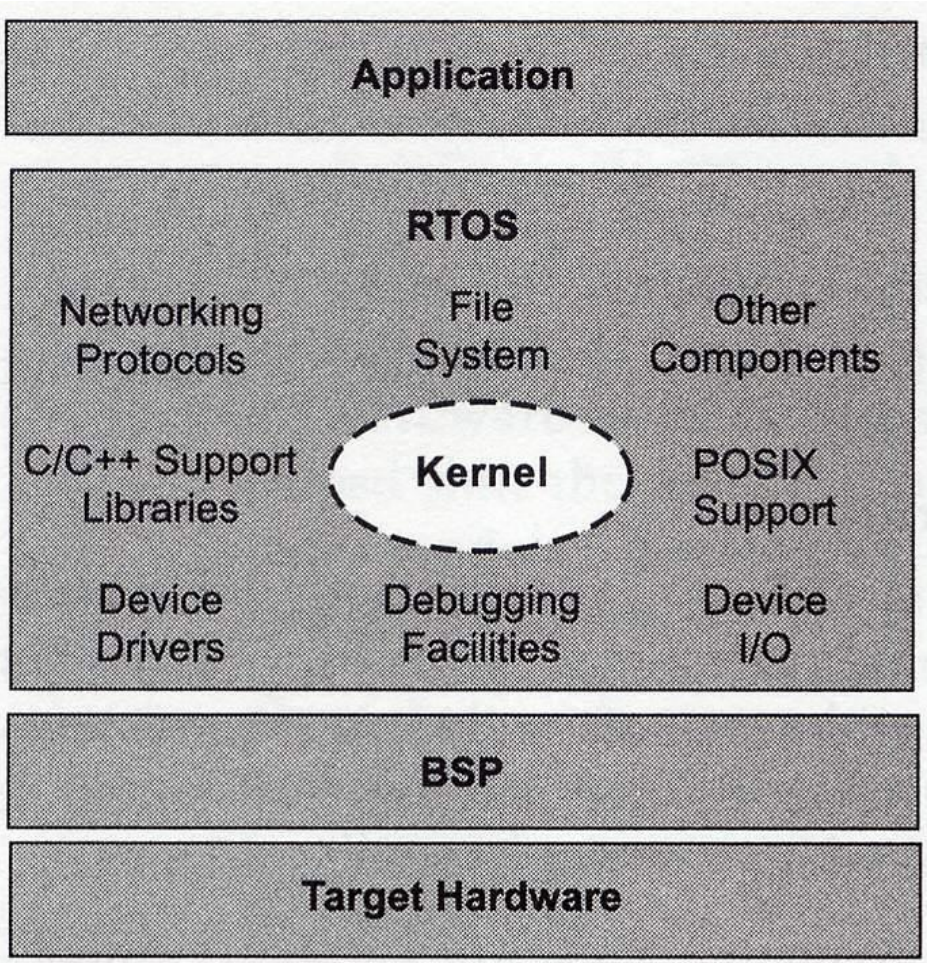


Figure 4.1 High-level view of an RTOS, its kernel, and other components found in embedded systems.

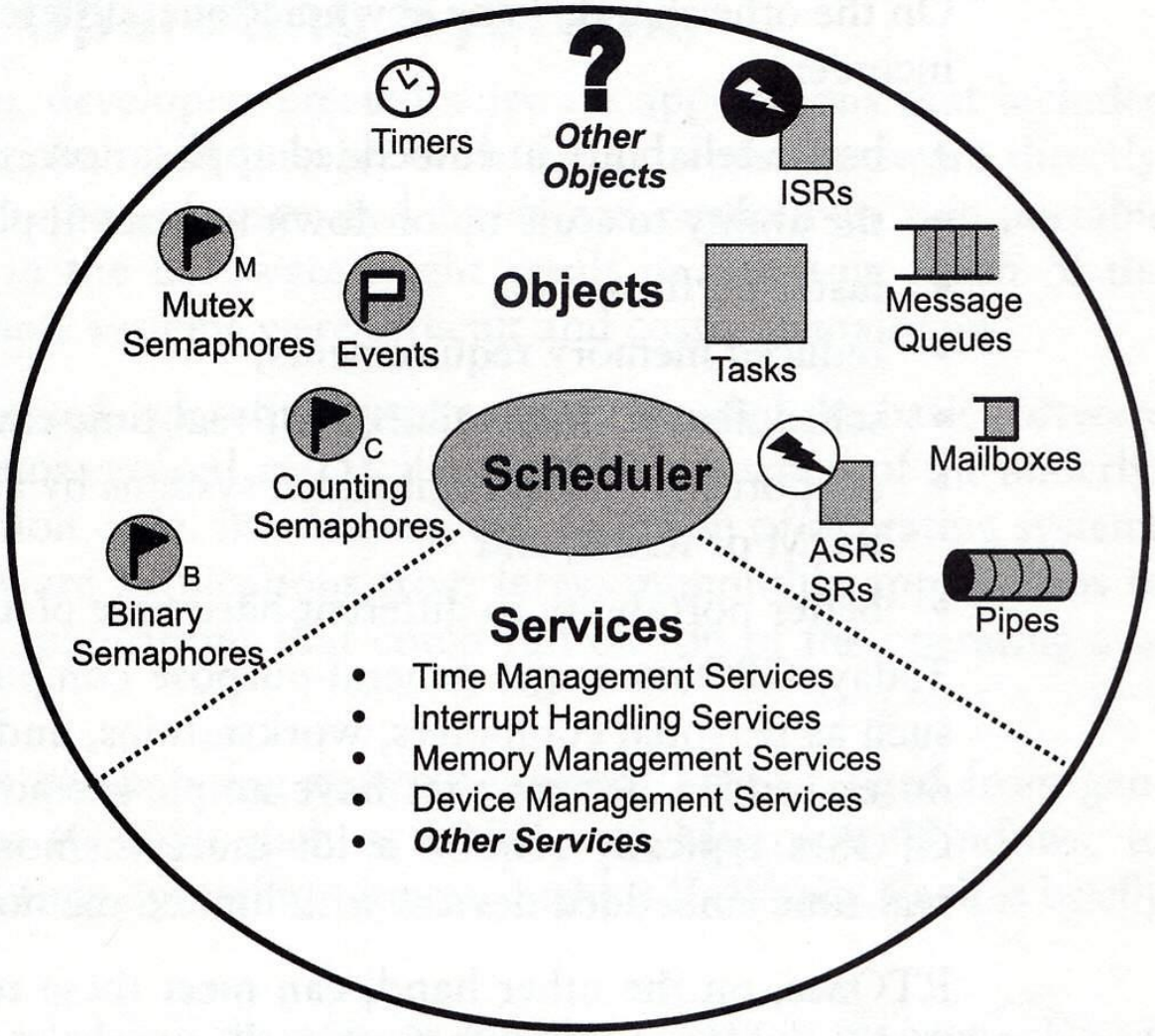


Figure 4.2 Common components in an RTOS kernel that including objects, the scheduler, and some services.

Scheduler

- Determine which task executes when
- Schedulable entities-a kernel object that can compete for execution on a system-> process, task
- Multitasking: many thread of execution appear to be running concurrently

Scheduler

- Context: the state of CPU registers
- Context switch
- When a new task is created, TCB(task control block) is also created
- TCB: system data structure

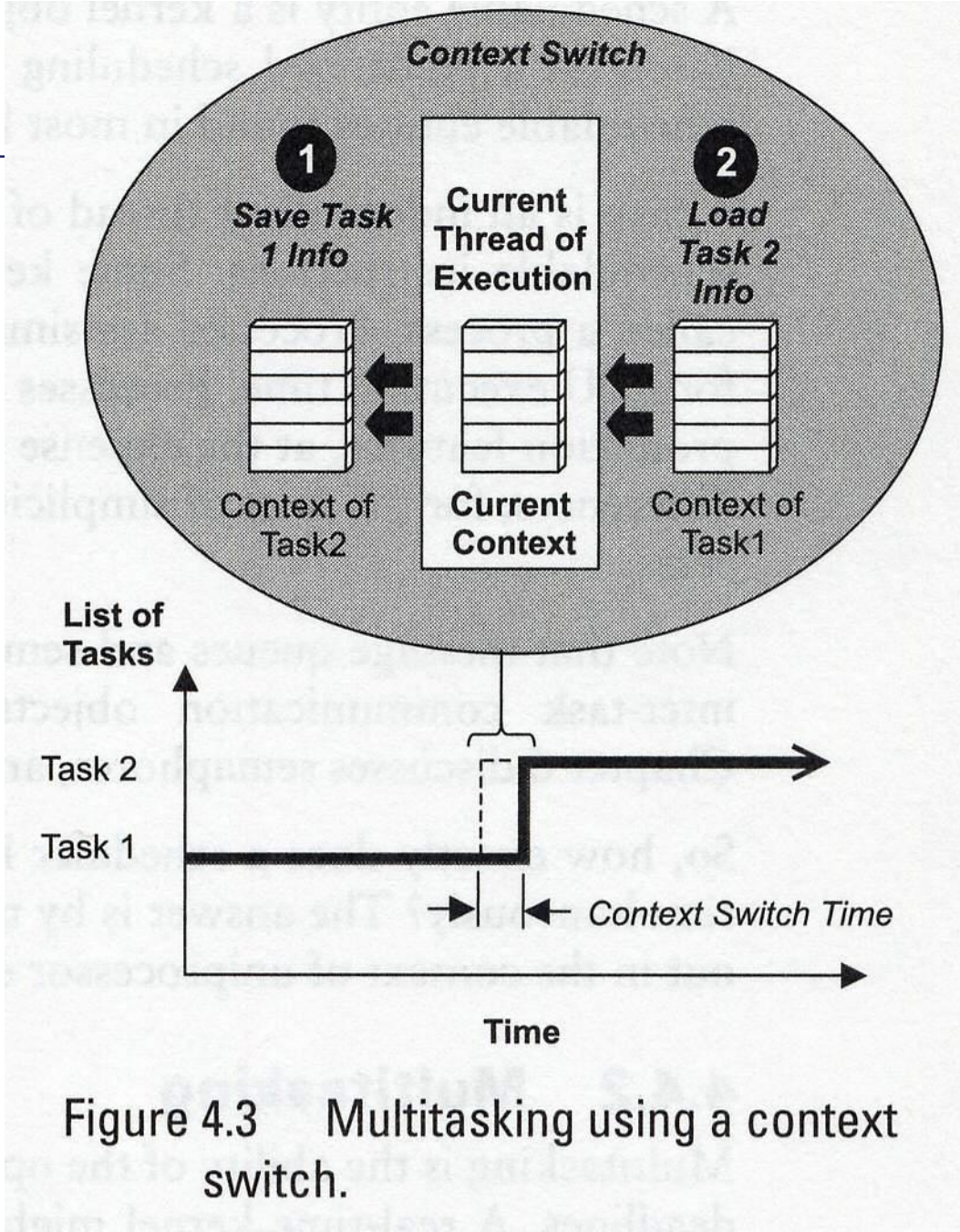


Figure 4.3 Multitasking using a context switch.

Scheduling Algorithms

- Preemptive priority-based scheduling
- Round-robin scheduling

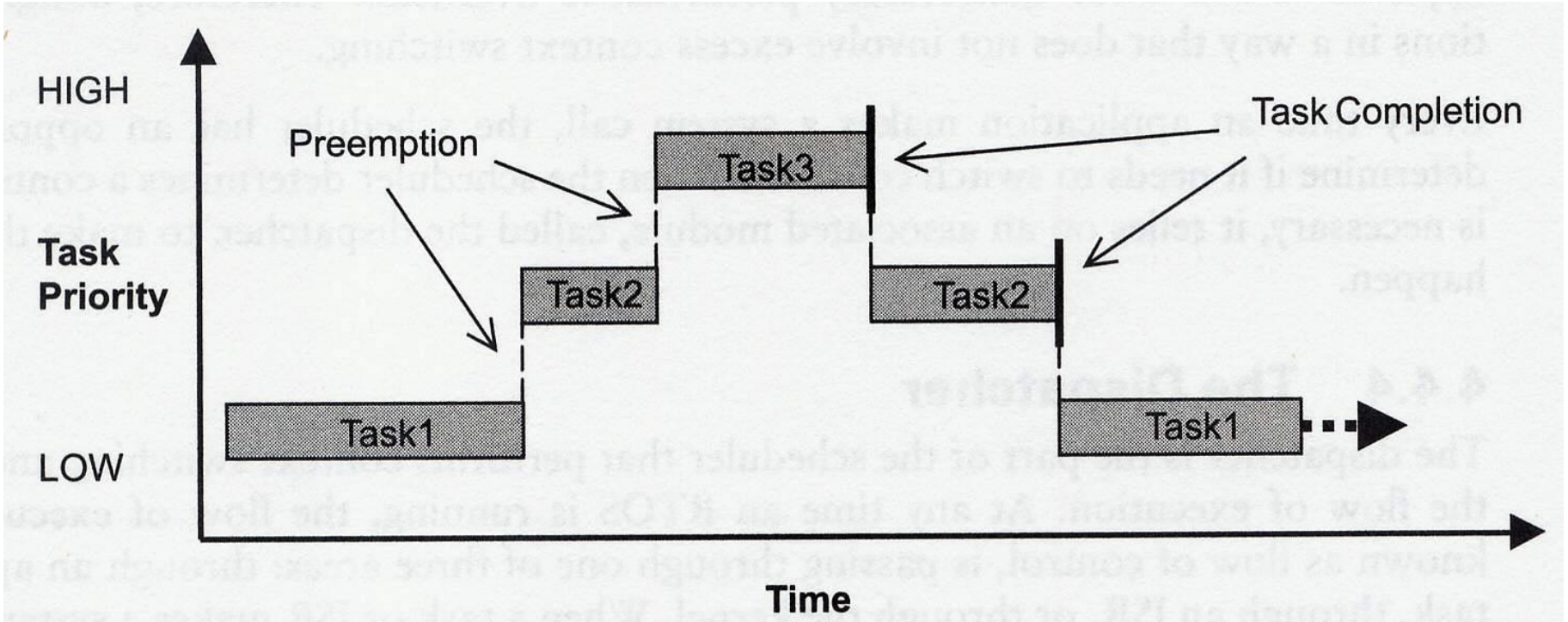


Figure 4.4 Preemptive priority-based scheduling.

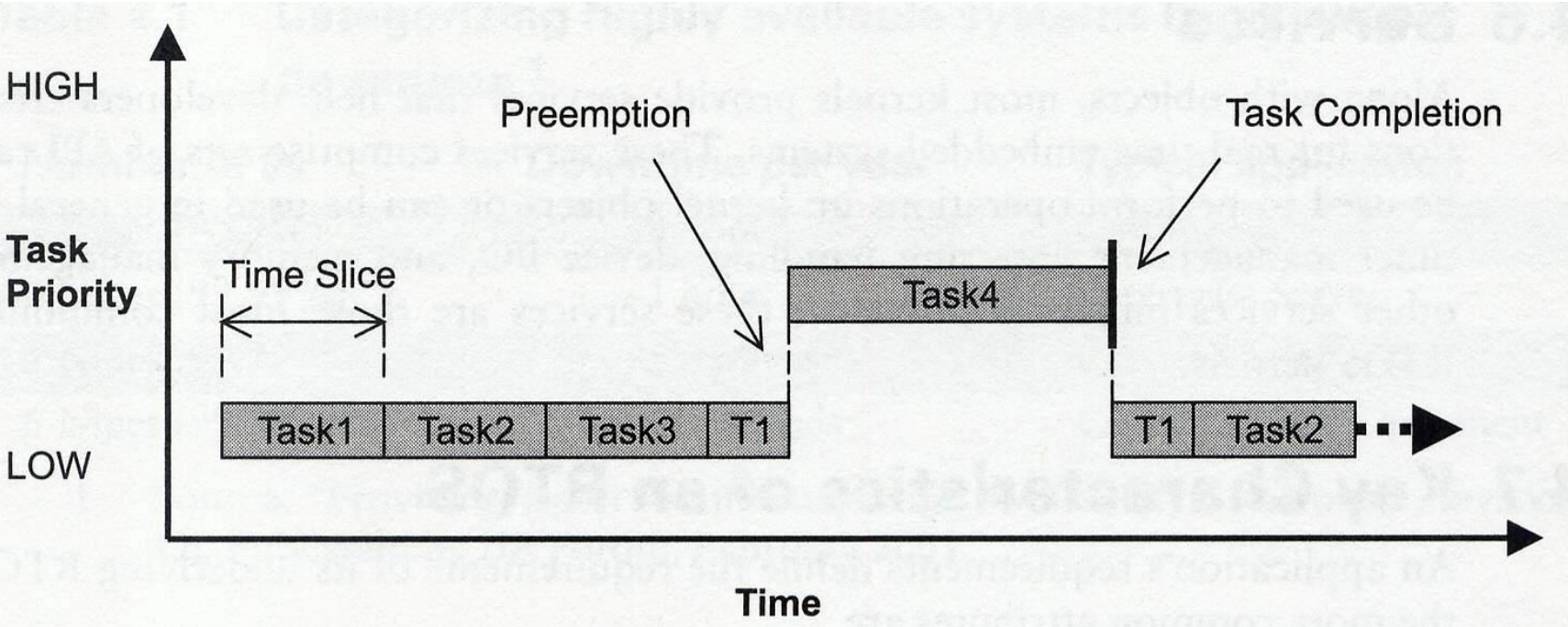
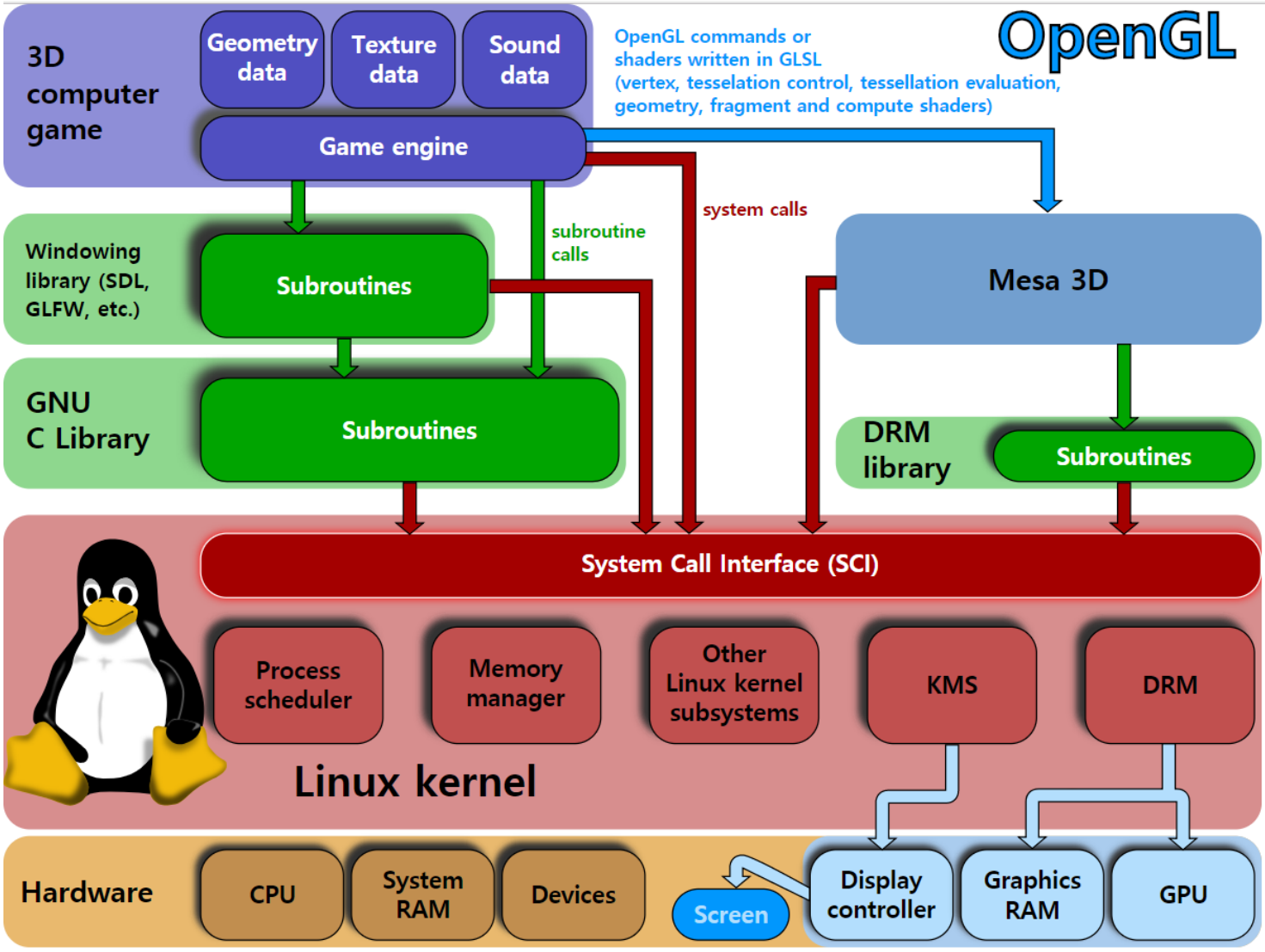


Figure 4.5 Round-robin and preemptive scheduling.

Linux

- Source code freely available
- Robust and reliable
- Modular, configurable, scalable
- Superb support for networking and Internet
- No runtime licenses
- Large pool of skilled developers

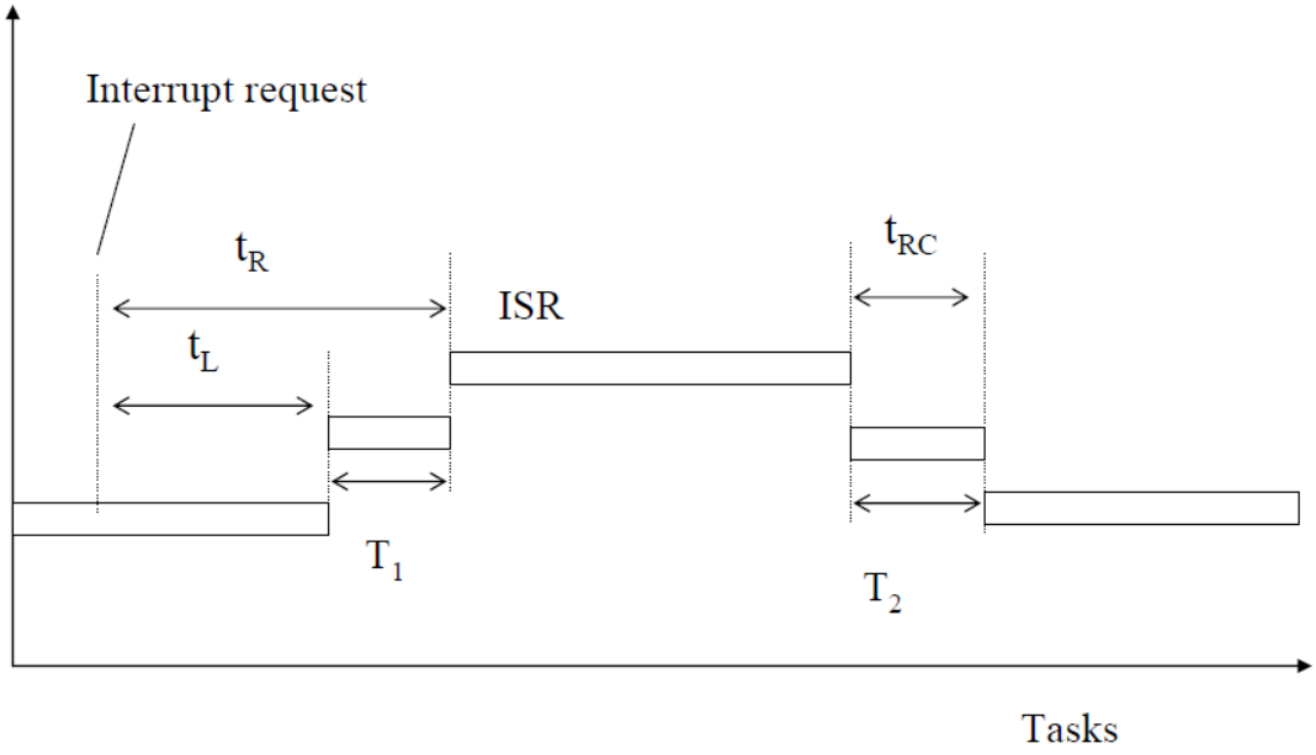

3D computer game
Geometry data
Texture data
Sound data
Game engine
OpenGL commands or shaders written in GLSL (vertex, tesselation control, tessellation evaluation, geometry, fragment and compute shaders)
OpenGL
system calls
subroutine calls
Windowing library (SDL, GLFW, etc.)
Subroutines
Mesa 3D
GNU C Library
Subroutines
DRM library
Subroutines
System Call Interface (SCI)
Process scheduler
Memory manager
Other Linux kernel subsystems
KMS
DRM
Linux kernel
Hardware
CPU
System RAM
Devices
Screen
Display controller
Graphics RAM
GPU

Interrupt Latency

- Traditional UNIX Operating systems suffer from large interrupt latency
- How to reduce the interrupt latency?
  - Make kernel highly preemptible by changing its internal structure (minimizing interrupt disabling) or adding a set of preemption points
  - Microkernel approach

Definition of Interrupt Latency

- Interrupt latency(t_L), response(t_R), and recovery (t_{RC}) times, T_1: time for saving CPU contexts, T_2: time for restoring CPU contexts



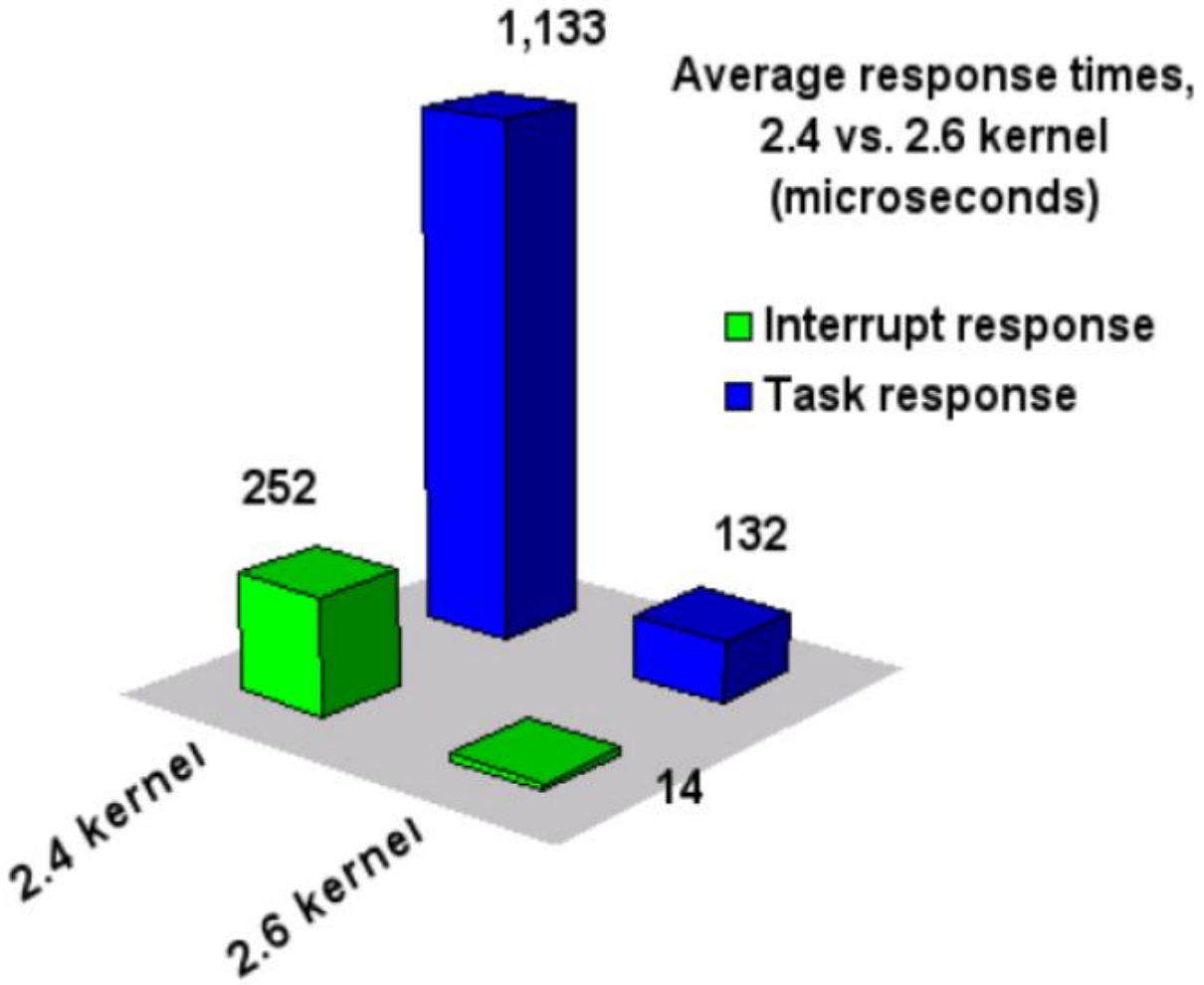
Linux and Real-Time

- Linux is not Real-Time
- Monolithic Kernel: The Linux kernel uses coarse grained synchronization, which allows a kernel task exclusive access to some data for long periods. This could delay the execution of any POSIX real-time task that needs access to that same data.
- Not Preemptible in Kernel Mode: The Linux kernel does not preempt the execution of any task during system calls. If a low priority process is in the middle of a system call and a message is received for a real-time process, the message will unfortunately be held in the queue until the system call completes, despite its low priority.

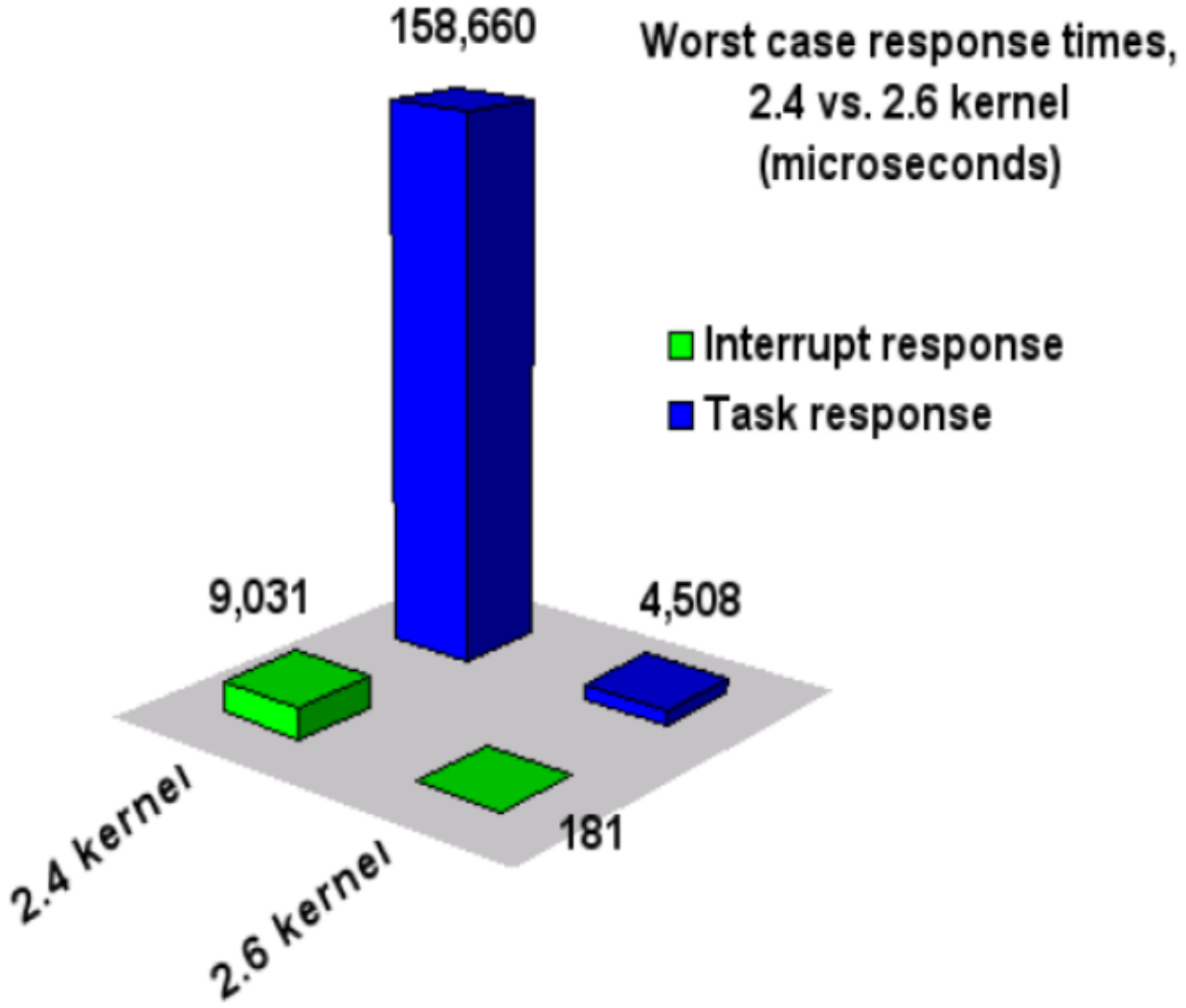
Linux and Real-Time

- Resource Lock: Linux makes high priority tasks wait for low priority tasks to release resources. For example, if any process allocates the last network buffer and a higher priority process needs a network buffer to send a message, the higher priority process must wait until some other process releases a network buffer before it can send its message.
- Priority Scheduling: The Linux scheduling algorithm will sometimes give the most unimportant and nicest process a time slice, even in circumstances when a higher priority process is ready to execute.

Linux Kernel 2.4 vs 2.6



Linux Kernel 2.4 vs 2.6



Cortex-M4 Processor Overview

with ARM Processors and Architectures

Introduction

ARM

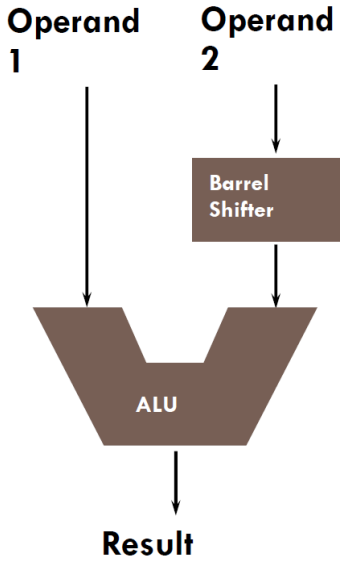
- **ARM was developed at Acorn Computer Limited of Cambridge, UK (between 1983 & 1985)**
  - RISC concept introduced in 1980 at Stanford and Berkeley
- **ARM founded in November 1990**
  - Advanced RISC Machines
- **Best known for its range of RISC processor cores designs**
  - Other products – fabric IP, software tools, models, cell libraries - to help partners develop and ship ARM-based SoCs
- **ARM does not manufacture silicon**
  - Licensed to partners to develop and fabricate new micro-controllers
    - Soft-core

ARM Architecture

- **Based upon RISC Architecture with enhancements to meet requirements of embedded applications**
  - A large uniform register file
  - Load-store architecture
    - `LDR` , `STR` : "Load register" and "Store register"
    - Examples:
      - `LDR R1, [R0]` ; load into R1 the content of the memory location whose address is in R0
      - `STR R1, [R0]` ; store the contents of R1 into the memory location whose address is in R0
  - Fixed length instructions
  - 32-bit processor (v1-v7), 64-bit processor (v8)
  - Good speed/power
  - High code density

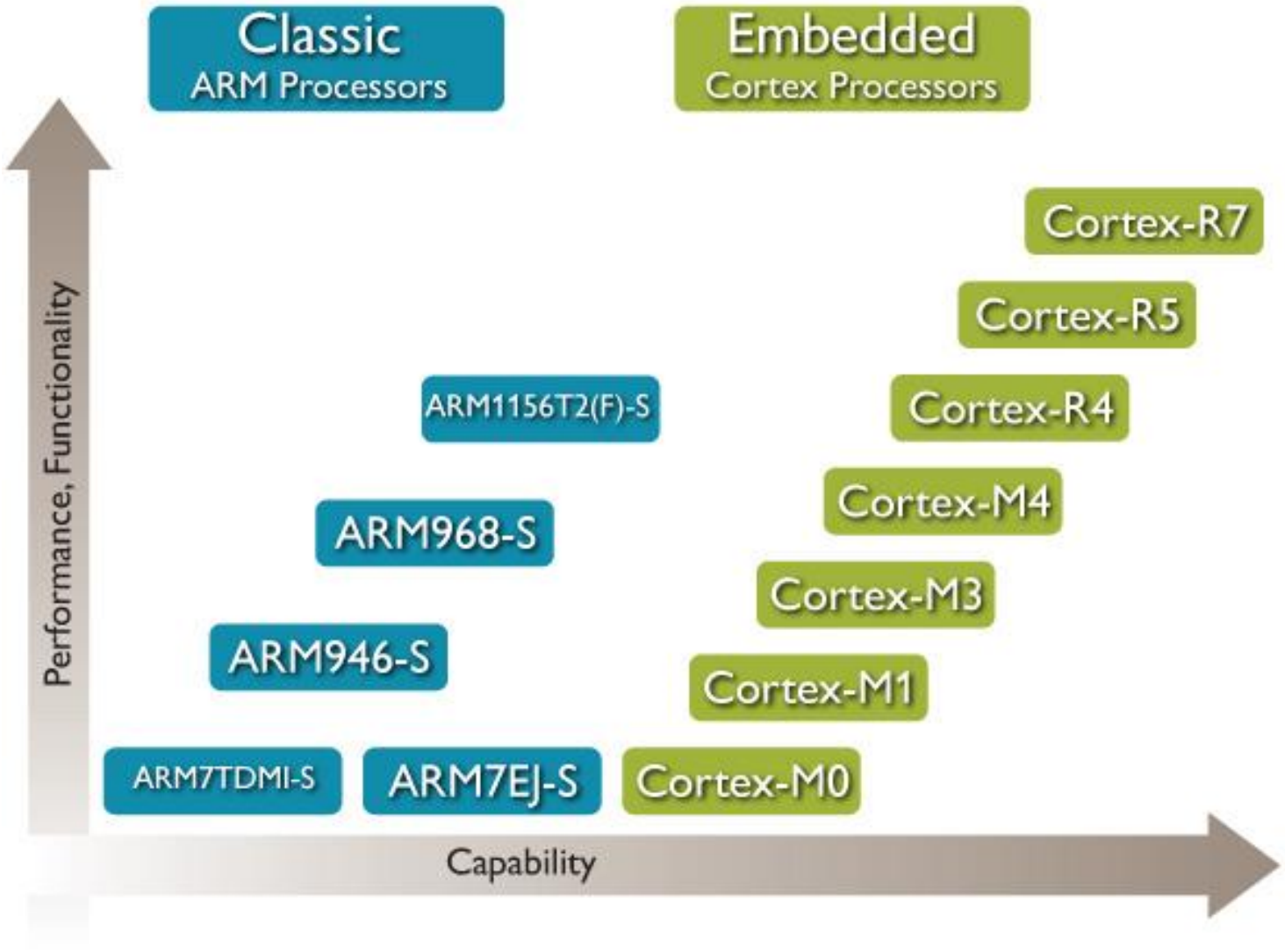
Enhancement to Basic RISC

- **Control over both ALU and shifter for every data processing operations**
  - ADD r2, r3, r4, LSL #2 ; r2 = r3 + (r4 * 4)

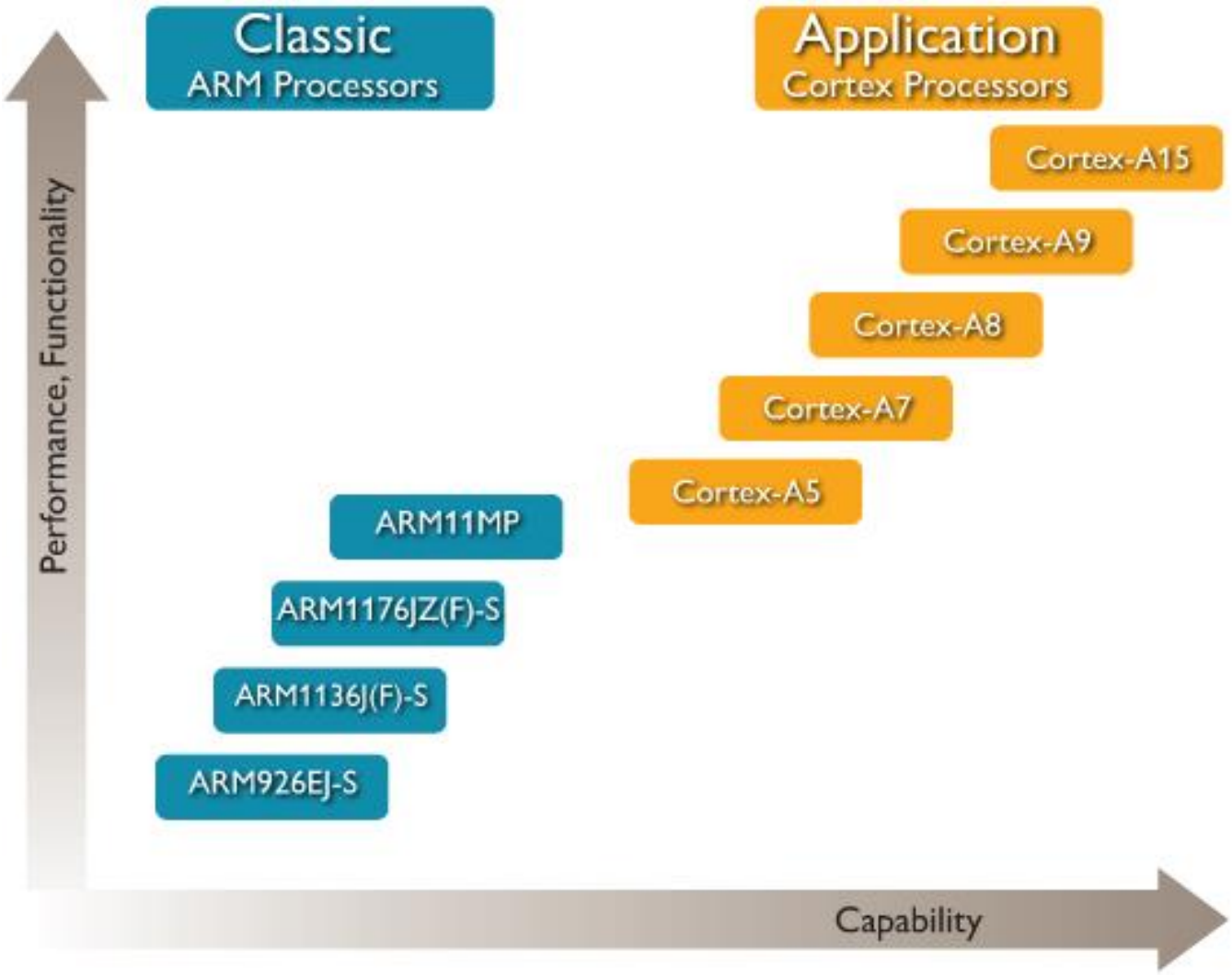


- **Auto-increment and auto-decrement addressing modes**
  - To optimize program loops
- **Load/Store multiple data instructions**
  - To maximize data throughput
  - LDM, STM
- **Conditional execution of instructions**
  - To maximize execution throughput
    - Example: ADDEQ r0, r1, r2
      - ➔ Instruction will only be executed when the zero flag is set to 1

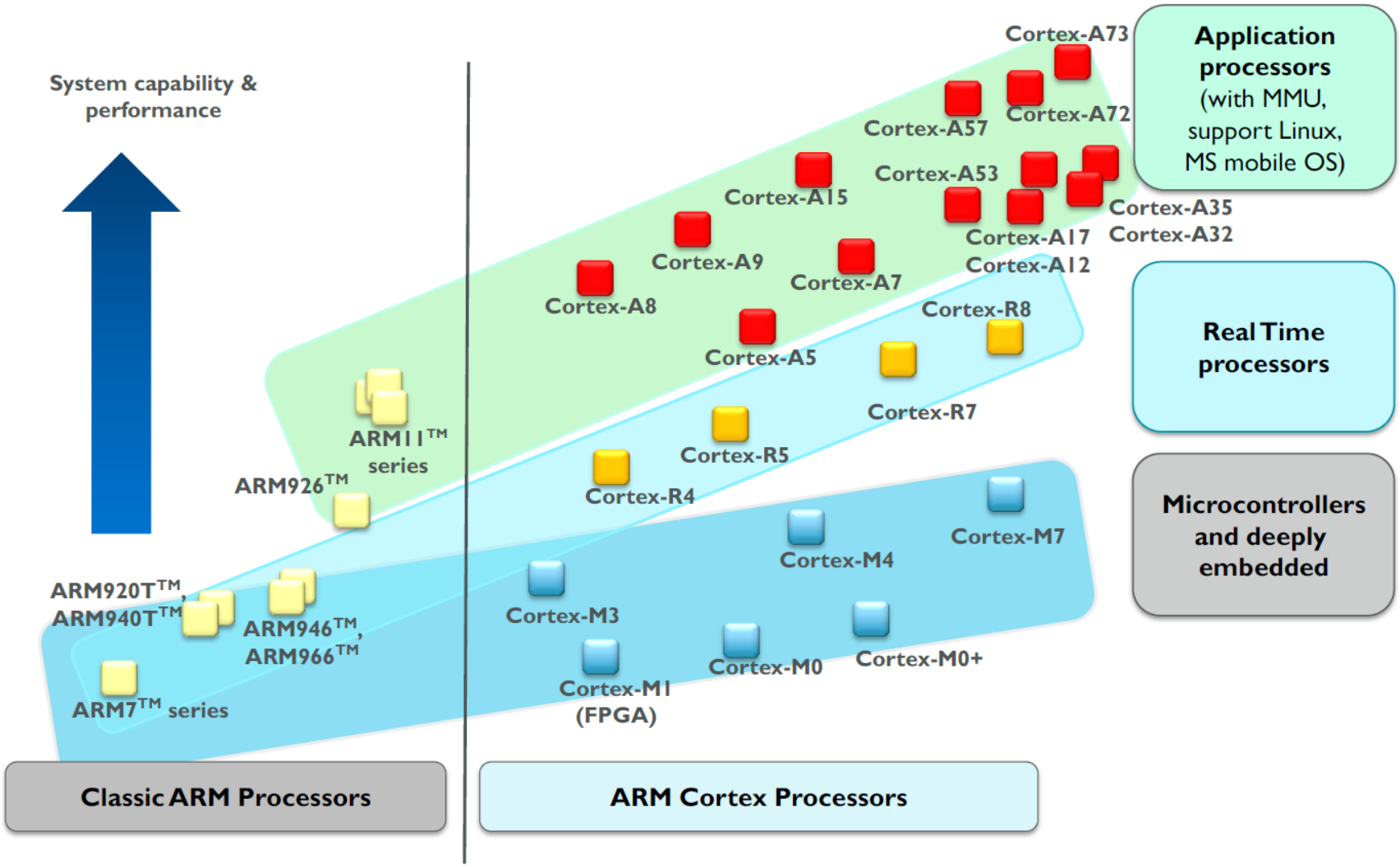
Embedded Processors



Application Processors



ARM Processor Family

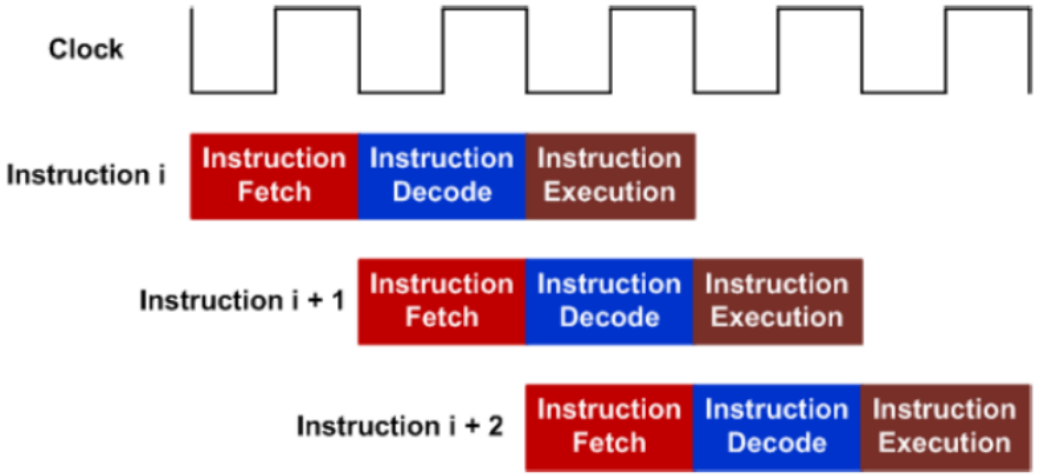


Summary of Processor Characteristics

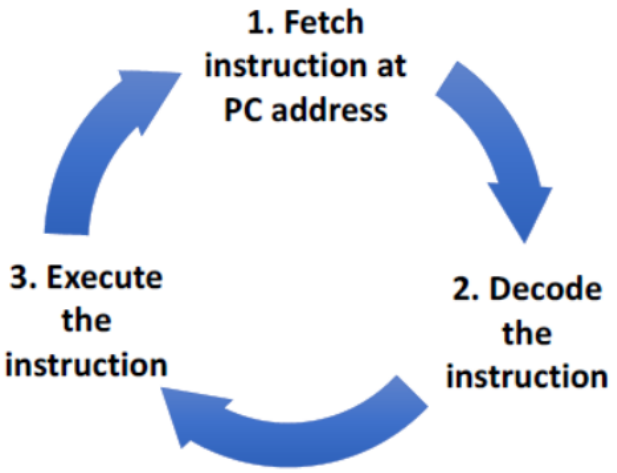
	Application processors	Real-time processors	Microcontroller processors
Design	High clock frequency, Long pipeline, High performance, Multimedia support (NEON instruction set extension)	High clock frequency, Long to medium pipeline length, Deterministic (low interrupt latency)	Short pipeline, ultra low power, Deterministic (low interrupt latency)
System features	Memory Management Unit (MMU), cache memory, ARM TrustZone® security extension	Memory Protection Unit (MPU), cache memory, Tightly Coupled Memory (TCM)	Memory Protection Unit (MPU), Nested Vectored Interrupt Controller (NVIC), Wakeup Interrupt Controller (WIC)
Targeted markets	Mobile computing, smart phones, energy-efficient servers, high-end microprocessors	Industrial microcontrollers, automotives, Hard disk controllers, Baseband modem	Microcontrollers, Deeply embedded systems (e.g. sensors, MEMS, mixed signal IC), Internet of Things (IoT)

Pipeline

- **Pipelining** allows hardware resources to be fully utilized
- One 32-bit instruction or **two 16-bit** instructions can be fetched.



Pipeline of 32-bit instructions

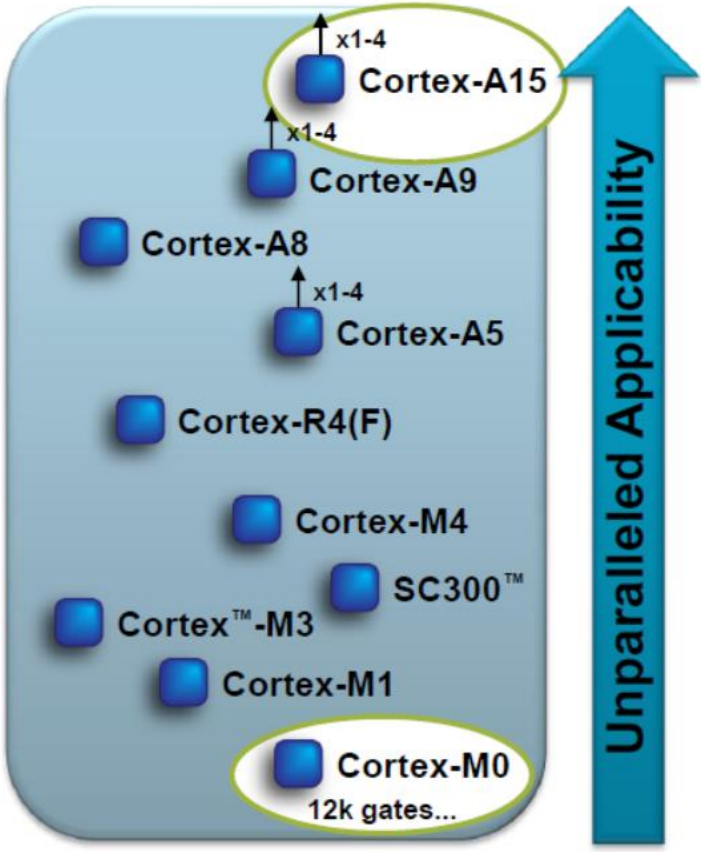


ARM Cortex Advanced Processors

Architectural innovation, compatibility across diverse application spectrum

Cortex™
Low-Power Leadership from ARM®

- ARM Cortex-**A** family:
  - Applications processors for feature-rich OS and 3rd party applications
- ARM Cortex-**R** family:
  - Embedded processors for real-time signal processing, control applications
- ARM Cortex-**M** family:
  - Microcontroller-oriented processors for MCU, ASSP, and SoC applications



Application Examples

Cortex-A


servers


set-top boxes


netbooks


mobile applications

Cortex-R


disk drives

digital cameras


mobile baseband

Cortex-M

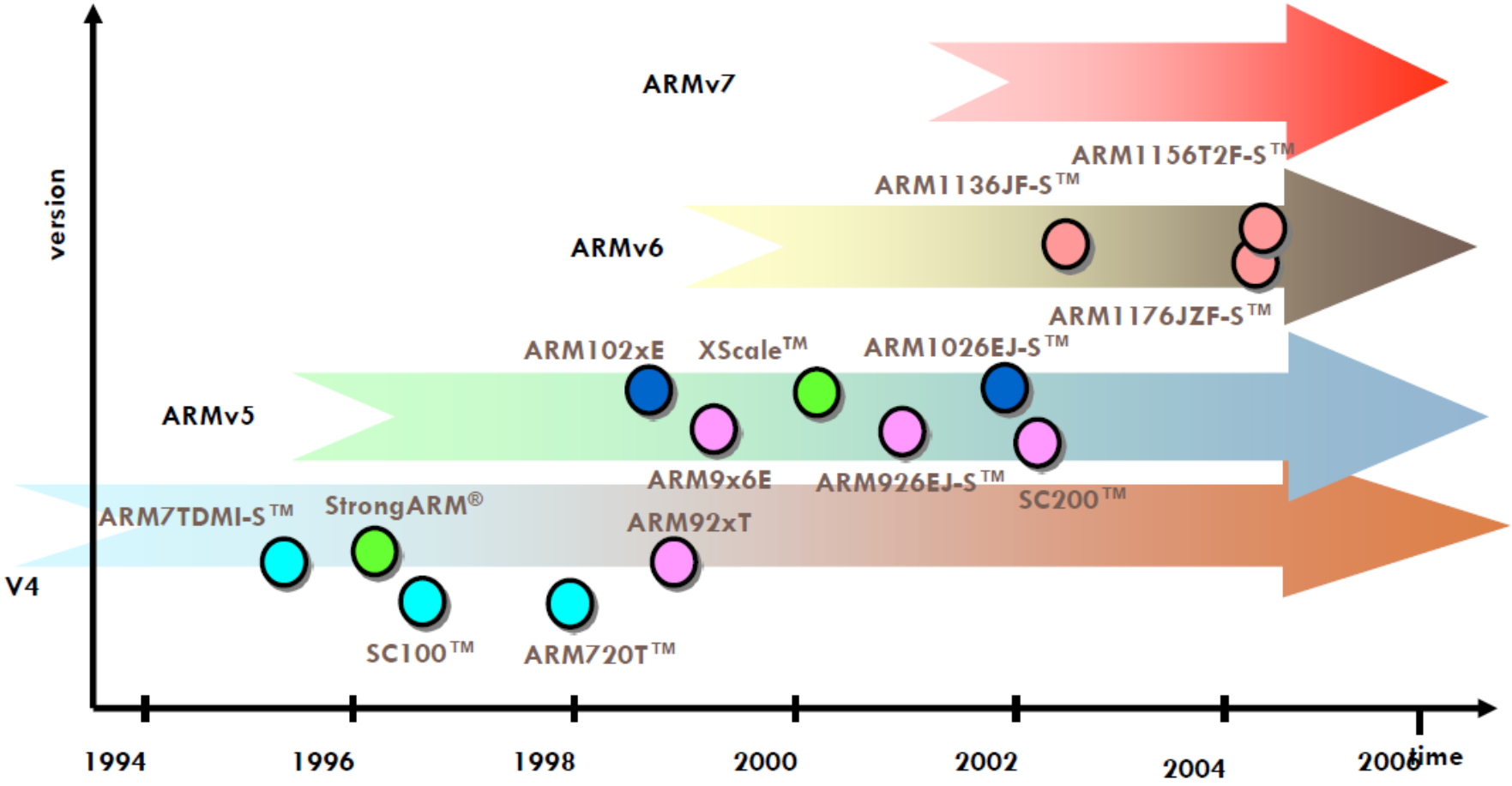

appliances


motors

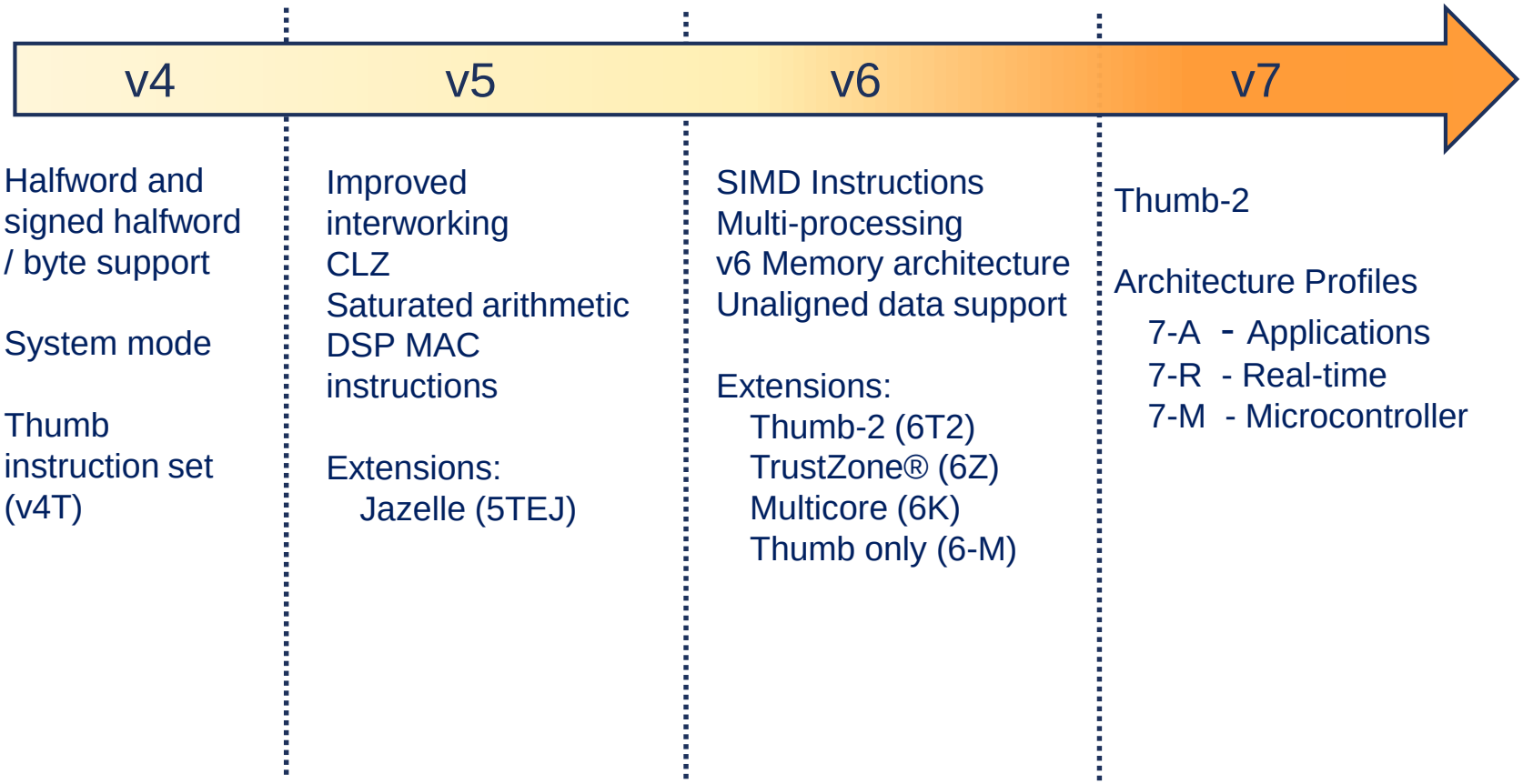

audio

ARM Architecture Overview

Architecture History



Development of the ARM Architecture



- **Note that implementations of the same architecture can be different**
  - Cortex-A8 - architecture v7-A, with a 13-stage pipeline
  - Cortex-A9 - architecture v7-A, with an 8-stage pipeline

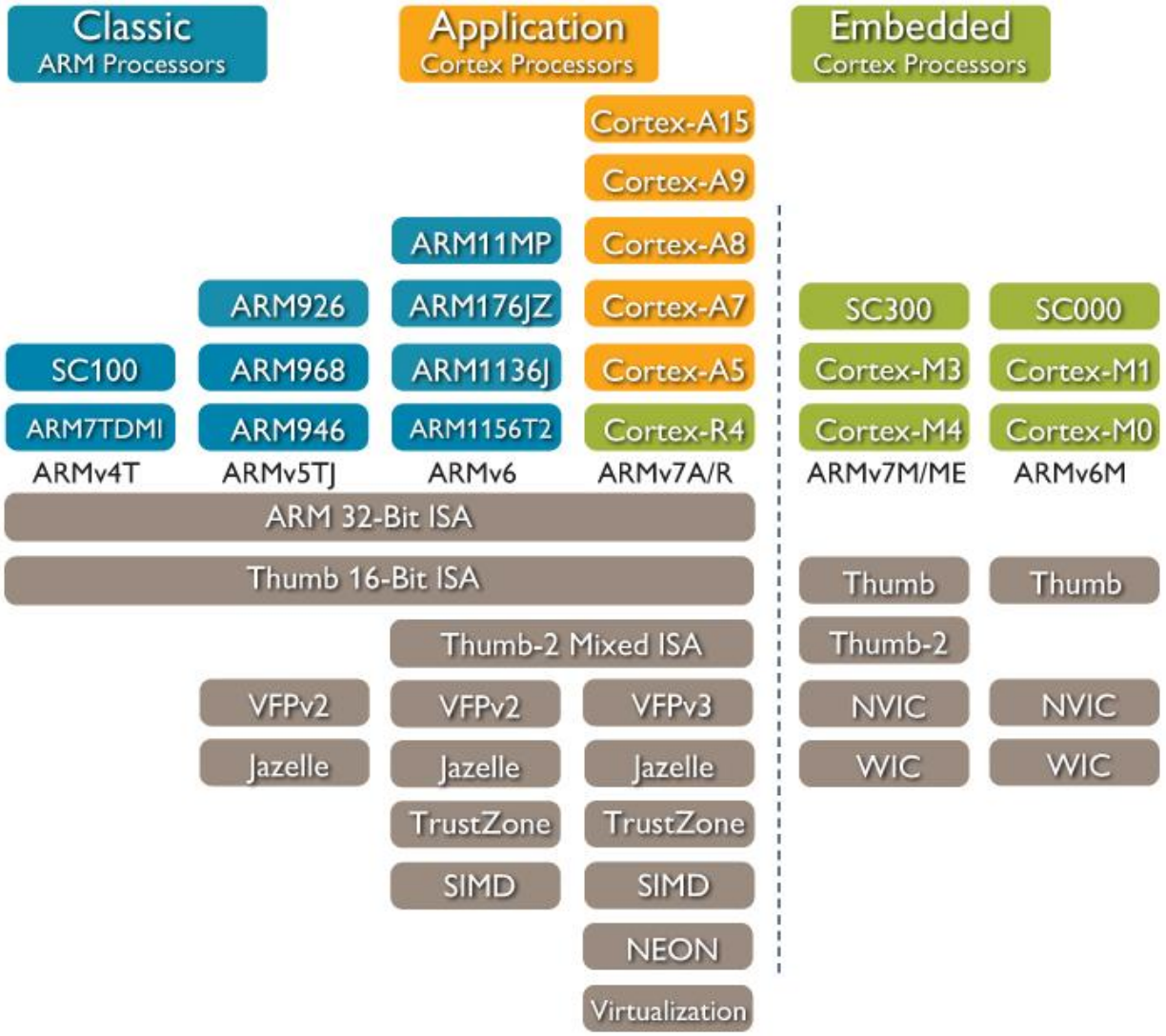
Architecture ARMv7 profiles

- **Application profile (ARMv7-A)**
  - Memory management support (MMU)
  - Highest performance at low power
    - Influenced by multi-tasking OS system requirements
  - TrustZone and Jazelle-RCT for a safe, extensible system
  - e.g. Cortex-A5, Cortex-A9

- **Real-time profile (ARMv7-R)**
  - Protected memory (MPU)
  - Low latency and predictability ‘real-time’ needs
  - Evolutionary path for traditional embedded business
  - e.g. Cortex-R4

- **Microcontroller profile (ARMv7-M, ARMv7E-M, ARMv6-M)**
  - Lowest gate count entry point
  - Deterministic and predictable behavior a key priority
  - Deeply embedded use
  - e.g. Cortex-M3

Which architecture is my processor?



Cotex-M Processor Family

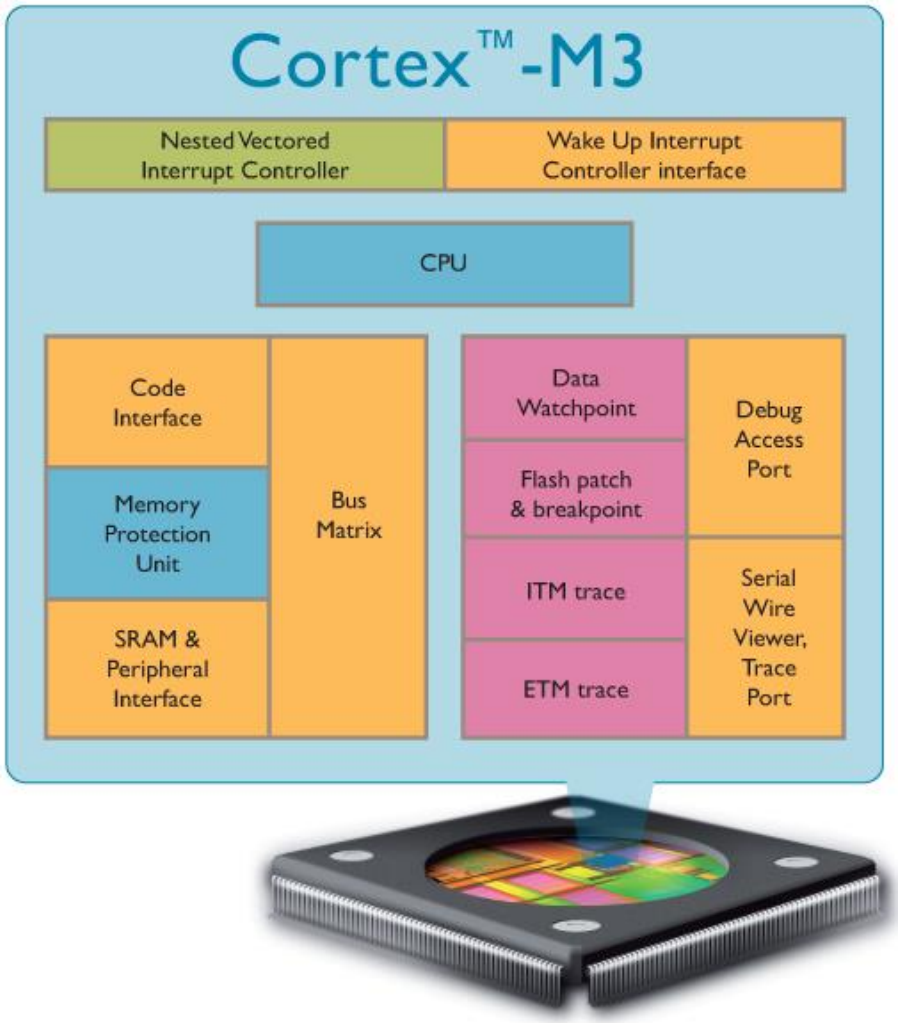
	Descriptions
Cortex-M0	A very small processor (starting from 12K gates) for low cost, ultra low power microcontrollers and deeply embedded applications
Cortex-M0+	The most energy-efficient processor for small embedded system. Similar size and programmer's model to the Cortex-M0 processor, but with additional features like single cycle I/O interface and vector table relocations
Cortex-M1	A small processor design optimized for FPGA designs and provides Tightly Coupled Memory (TCM) implementation using memory blocks on the FPGAs. Same instruction set as the Cortex-M0
Cortex-M3	A small but powerful embedded processor for low-power microcontrollers that has a rich instruction set to enable it to handle complex tasks quicker. It has a hardware divider and Multiply-Accumulate (MAC) instructions. In addition, it also has comprehensive debug and trace features to enable software developers to develop their applications quicker
Cortex-M4	It provides all the features on the Cortex-M3, with additional instructions target at Digital Signal Processing (DSP) tasks, such as Single Instruction Multiple Data (SIMD) and faster single cycle MAC operations. In addition, it also have an optional single precision floating point unit that support IEEE 754 floating point standard
Cortex-M7	High-performance processor for high-end microcontrollers and processing intensive applications. It has all the ISA features available in Cortex-M4, with additional support for double-precision floating point, as well as additional memory features like cache and Tightly Coupled Memory (TCM)

ARMv7-M Architecture

ARMv7-M Profile Overview

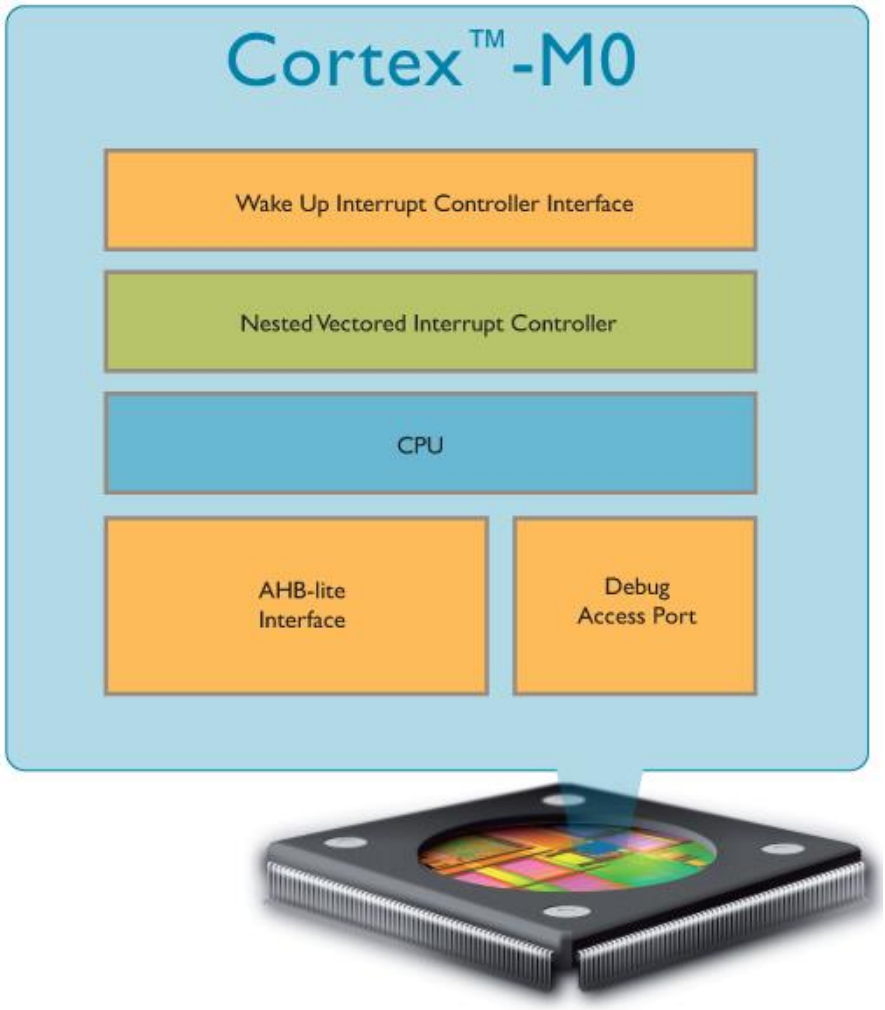
- **v7-M Cores are designed to support the microcontroller market**
  - Simpler to program – entire application can be programmed in C
  - Fewer features needed than in application processors
- **Register and ISA changes from other ARM cores**
  - No ARM instruction set support
  - Only one set of registers
  - xPSR has different bits than CPSR
- **Different modes and exception models**
  - Only two modes: Thread mode and Handler mode
  - Vector table is addresses, not instructions
  - Exceptions automatically save state (r0-r3, r12, lr, xPSR, pc) on the stack
- **Different system control/memory layout**
  - Cores have a fixed memory map
  - No coprocessor 15 – controlled through memory mapped control registers

Cortex-M3



- **ARMv7-M Architecture**
  - Thumb-2 only
- **Fully programmable in C**
- **3-stage pipeline**
- **Optional MPU**
- **AHB-Lite bus interface**
- **Fixed memory map**
- **1-240 interrupts**
  - Configurable priority levels
  - Non-Maskable Interrupt support
  - Debug and Sleep control
- **Serial wire or JTAG debug**
- **Optional ETM**

Cortex-M0



- **ARMv6-M Architecture**
  - 16-bit Thumb-2 with system control instructions
- **Fully programmable in C**
- **3-stage pipeline**
- **AHB-Lite bus interface**
- **Fixed memory map**
- **1-32 interrupts**
  - Configurable priority levels
  - Non-Maskable Interrupt support
- **Low power support**
- **Core configured with or without debug**
  - Variable number of watchpoints and breakpoints

Thumb-2 Technology

- **Thumb-2 ISA was introduced in ARMv7 architecture**
  - Original16-bit Thumb instructions maintain full compatibility with existing code

    +
  - New 16-bit Thumb instructions for improved program flow

    +
  - New 32-bit Thumb instructions for improved performance and code size.
    One 32-bit instruction replaces multiple 16-bit opcodes.
    32-bit instructions are handled in the same mode ~ no interworking required