Real Time Operating System

KERNEL MASTERS

Content

- **Introduction to RTOS**
  - What is OS?
  - Types of OS?
  - What is Real Time System?
  - Types of Real Time System?
- **Tasks vs Process vs Threads**
- **Threads Classifications**
- **Threads Schedulers**

RTOS Prerequisites

- TIMERS
- Function Pointers
- OS Basics

What is Operating System?

- **Operating System:** is a software program that can interface between user and hardware.

- **Type of OS:**
  - **RTOS**
    - Example: FREE RTOS, ucos, vxworks
  - **GPOS**
    - Example: Windows, Linux , UNIX

GPOS vs RTOS

- System
  - General Purpose System
    - GPOS
      - Windows, Linux
  - Real Time System
    - RTOS
      - RT Linux, FREE RTOS, vxwroks

What Does an Operating System Do?

- **Multitasking**
  - processes cooperate to share processor time
    - willing to periodically check if other process requires action
  - requires careful programming
  - ‘buggy’ process can stall system

What Does an Operating System Do?

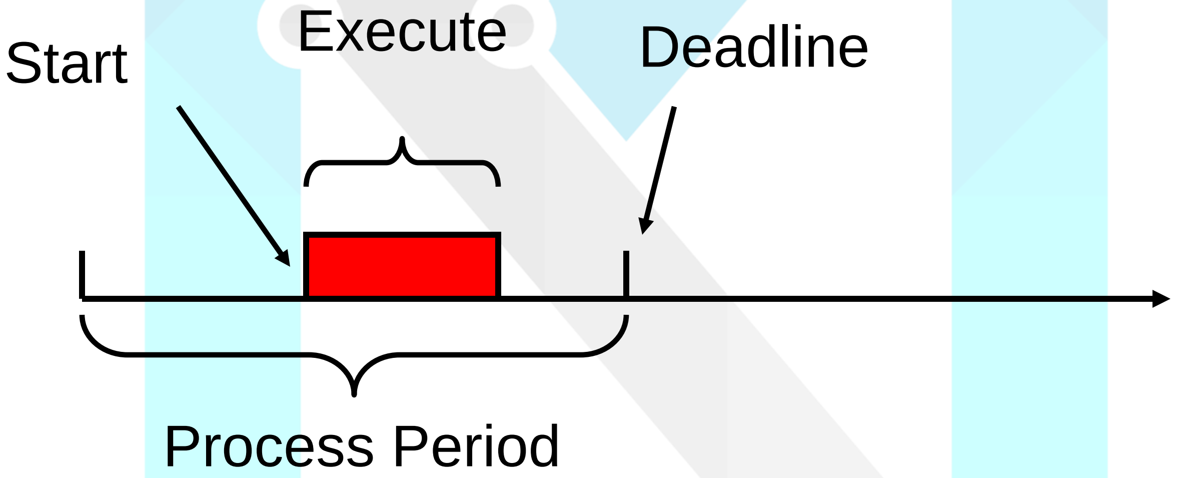
- **Preemptive Multitasking**
  - operating system shifts from process to process
    - *time slicing*
    - intervals determined by clock interrupt
  - operating system maintains *context* for each process, *e.g.*, registers, memory allocation
    - task control blocks
  - processes assigned *priority* for access to system resources

What Does an Operating System Do?

- **Scheduling with process priority**
  - – each process has fixed priority
  - – RTOS *scheduler* is able to preempt running process if higher priority process activated
    - I/O interrupt produces/requires data conversion
  - – scheduler runs highest priority process
    - process typically runs to completion
    - may exclude lower priority processes
      - – exclusion avoidance schemes available

What Does an Operating System Do?

- Process attributes
  - execution time, process period, and deadline



Tasks and processes

- A task is a functional description of a connected set of operations.
- (Task can also mean a collection of processes)

- A process is a unique execution of a program.
  - Several copies of a program may run simultaneously or at different times.
- A process has its own state:
  - registers;
  - memory.
- The operating system manages processes.

KERNEL MASTERS

Why multiple processes?

- Multiple tasks means multiple processes.
- Processes help with timing complexity:
  - – multiple rates
    - multimedia
    - automotive
  - – asynchronous input
    - user interfaces
    - communication systems

KERNEL MASTERS

adapted from Wolf

Multi-rate systems

- Tasks may be synchronous or asynchronous.
- Synchronous tasks may recur at different rates.
- Processes run at different rates based on computational needs of the tasks.

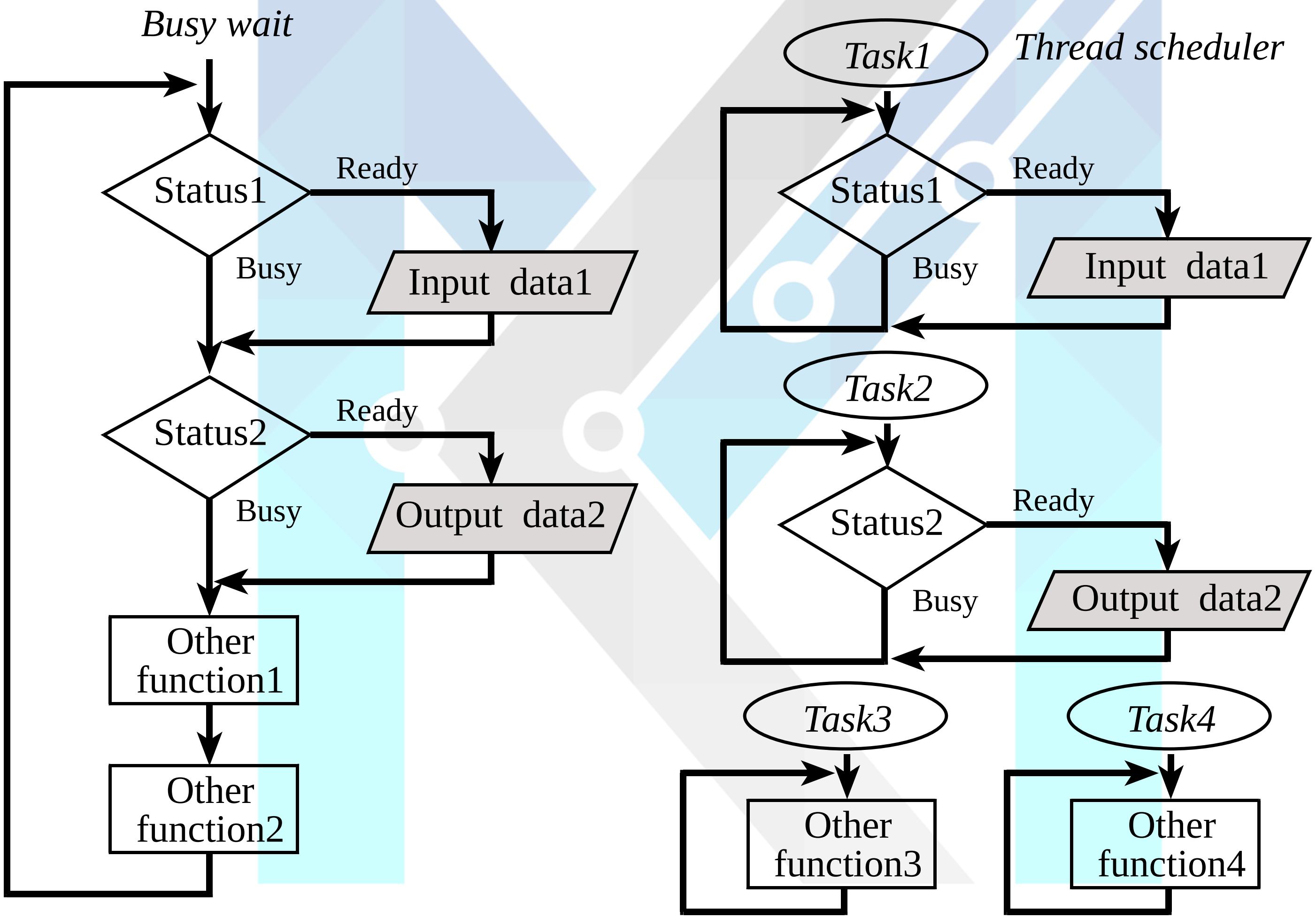
Real Time Operating System

- Real Time Operating Systems
  - Critical – system fails if time constraints not met
  - Dependent – occasional failure to meet time constraint is acceptable
  - Flexible – performance degrades as time constraints missed

What is Real Time System?

- **Real Time System:** System output not only depends on the results and also depends on deliverable time.
  - Real time systems are time deterministic Systems.
  - Real time systems are not a fastest systems.
- **Type of Real Time Systems:**
  - **Hard Real Time System:** Failure to meet deadline causes system failure.
    - Example: Automation plant, Aerospace, Defense
  - **Soft Real Time System:** Failure to meet deadline causes degraded response rather than system failure.
    - Example: Online Reservation
  - **Firm Real Time System:** Late response is useless but some late responses can be tolerated.

Single thread vs multithread



KERNEL MASTERS

Single thread vs multithread

Single Thread

- Process Contains single thread
- It is a busy wait loop
- Doesn’t support OS Concepts

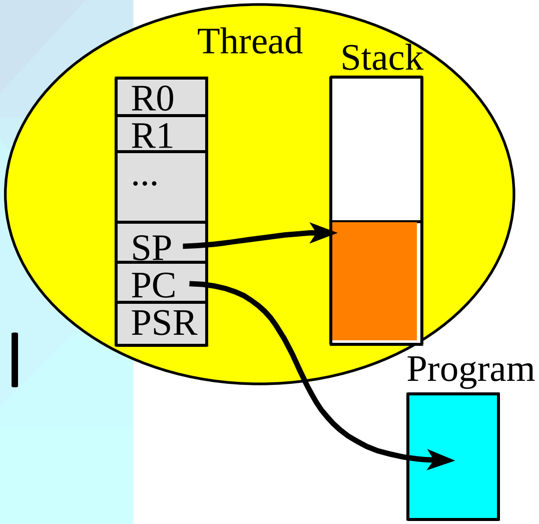
Multi Thread

- Process contains multiple threads
- It doesn't need a busy wait loop.
- Support OS Concepts.
- Support Scheduler Algorithms
  - Periodic Algorithm
  - Aperiodic Algorithm
  - Priority Algorithm

KERNEL MASTERS

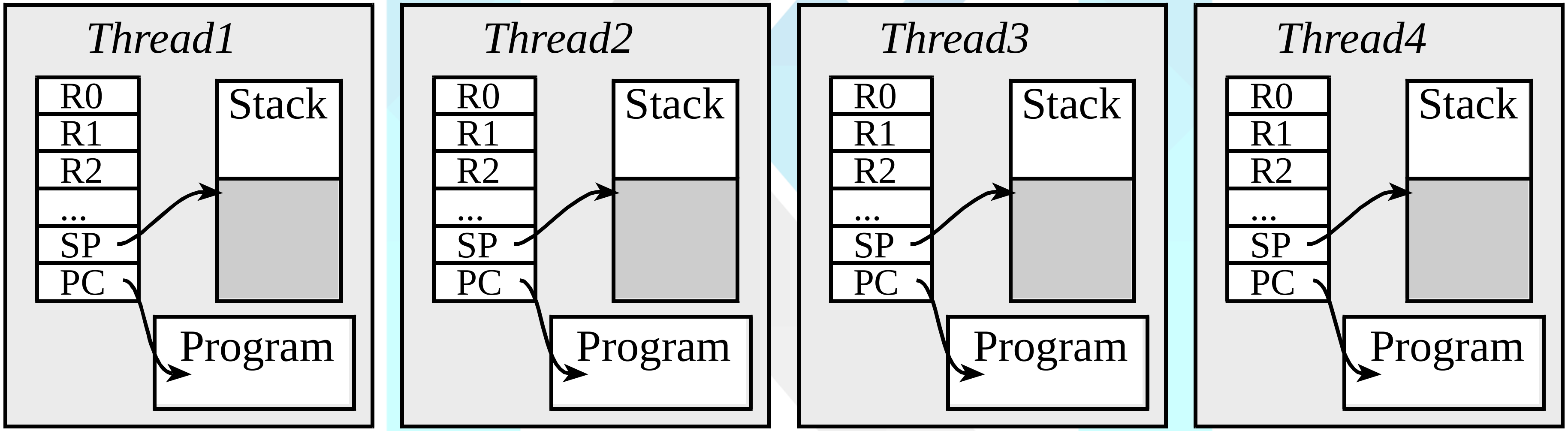
Thread or Light-weight process

- Thread is a single unit of Execution of a software task
- Has its own registers
- Has its own stack
- Local variables are private
- Threads cooperate for common goal
- Private global variables
  - Managed by the OS
  - Allocated in the TCB (e.g., `Id`)

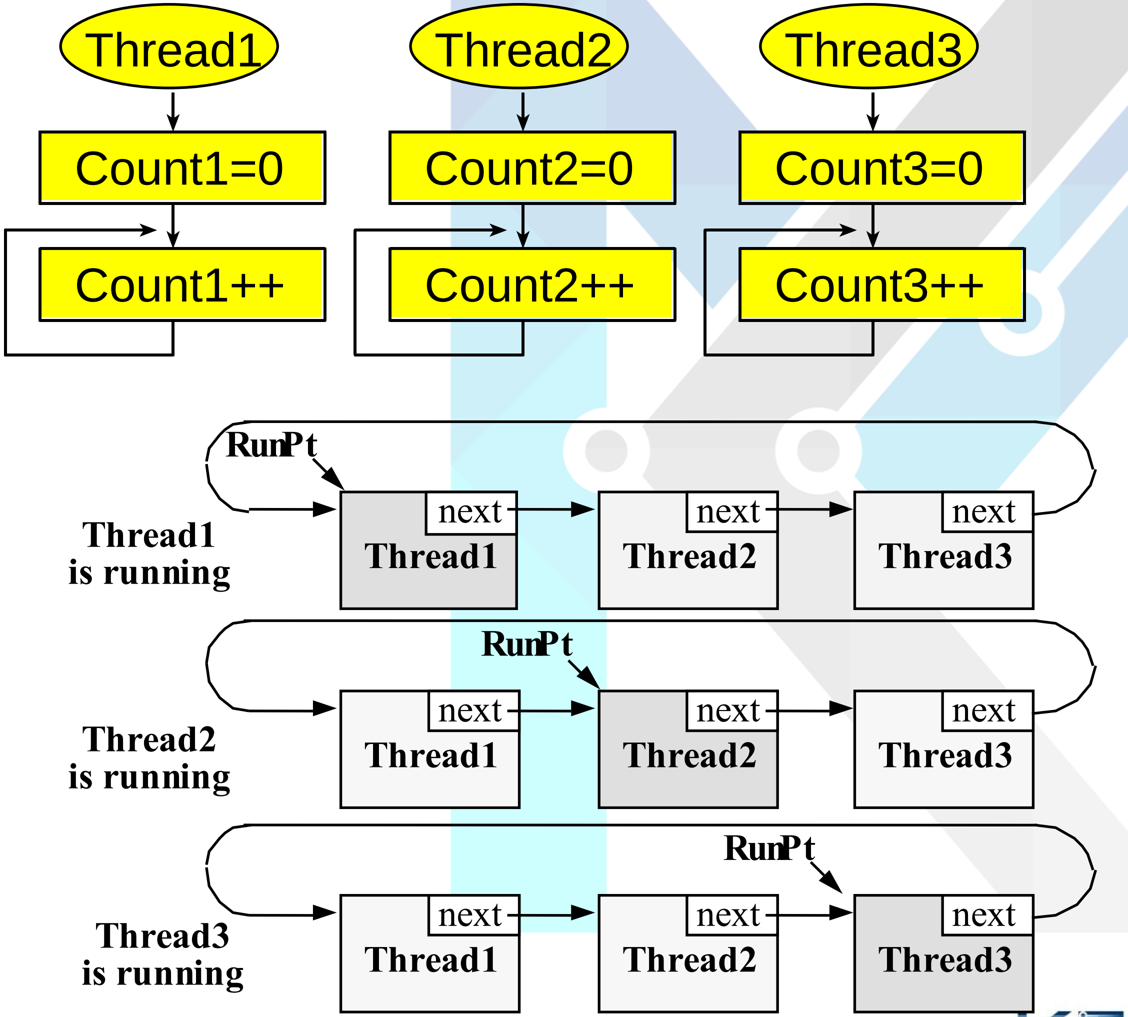


Basic Operation...

What is Thread?



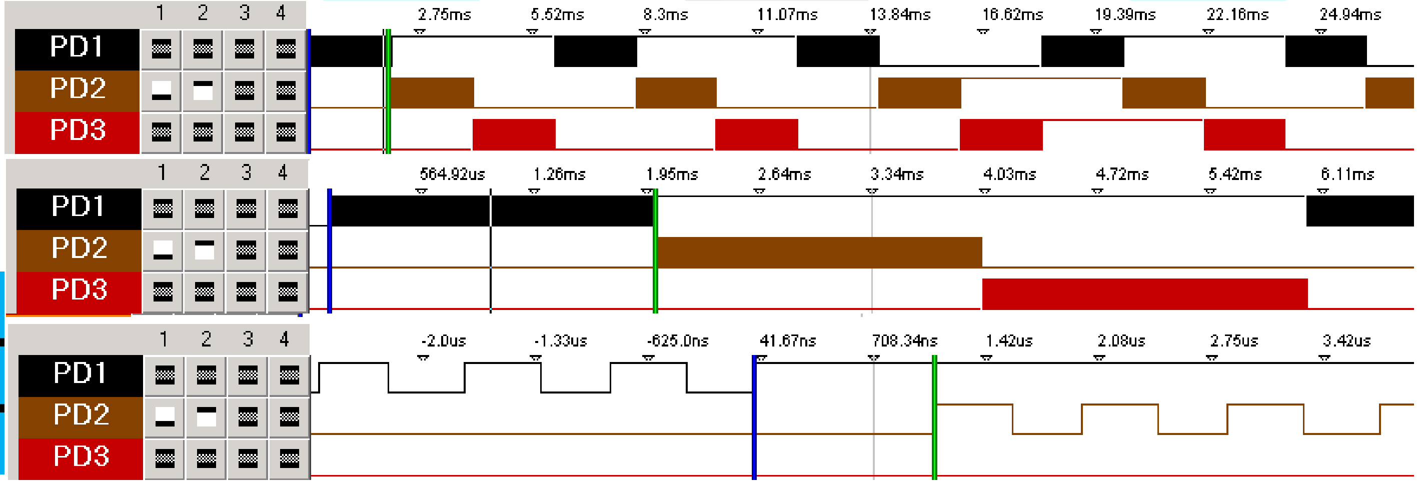
Threads



FIFOs...

Tasks and processes

```
SysTick_Handler                ; 1) Saves R0-R3,R12,LR,PC,PSR
    CPSID   I                  ; 2) Prevent interrupt during switch
    PUSH    {R4-R11}           ; 3) Save remaining regs r4-11
    LDR     R0, =RunPt         ; 4) R0=pointer to RunPt, old thread
    LDR     R1, [R0]           ;    R1 = RunPt
    STR     SP, [R1]           ; 5) Save SP into TCB
    LDR     R1, [R1,#4]        ; 6) R1 = RunPt->next
    STR     R1, [R0]           ;    RunPt = R1
    LDR     SP, [R1]           ; 7) new thread SP; SP = RunPt->sp;
    POP     {R4-R11}           ; 8) restore regs r4-11
    CPSIE   I                  ; 9) tasks run with interrupts
enabled
    BX      LR                 ; 10) restore R0-R3,R12,LR,PC,PSR
```



ds

PERIODIC TASK	**APERIODIC TASK**
It repeats itself after a certain time interval.	It can occur at random instants.
These tasks are controlled by clock interrupts.	These tasks are not controlled by clock interrupts.

Threads Classifications

- Periodic, execution at regular intervals
  - E.g., ADC, DAC, motor control
  - E.g., Check CO levels
- Aperiodic, execution can not be anticipated
  - Execution is frequent
  - E.g., New position detected as wheel turns
- Sporadic, execution can not be anticipated
  - Execution is infrequent
  - E.g., Faults, errors, catastrophes

Thread Scheduler

- List possible thread states
- List possible scheduling algorithms
  - What?
  - How?
  - Why?
- Performance measures
  - Utilization
  - Latency
  - Bandwidth

When to run scheduler??

Round robin
Weighted round robin
Priority

Static
Dynamic
Deterministic

Priority

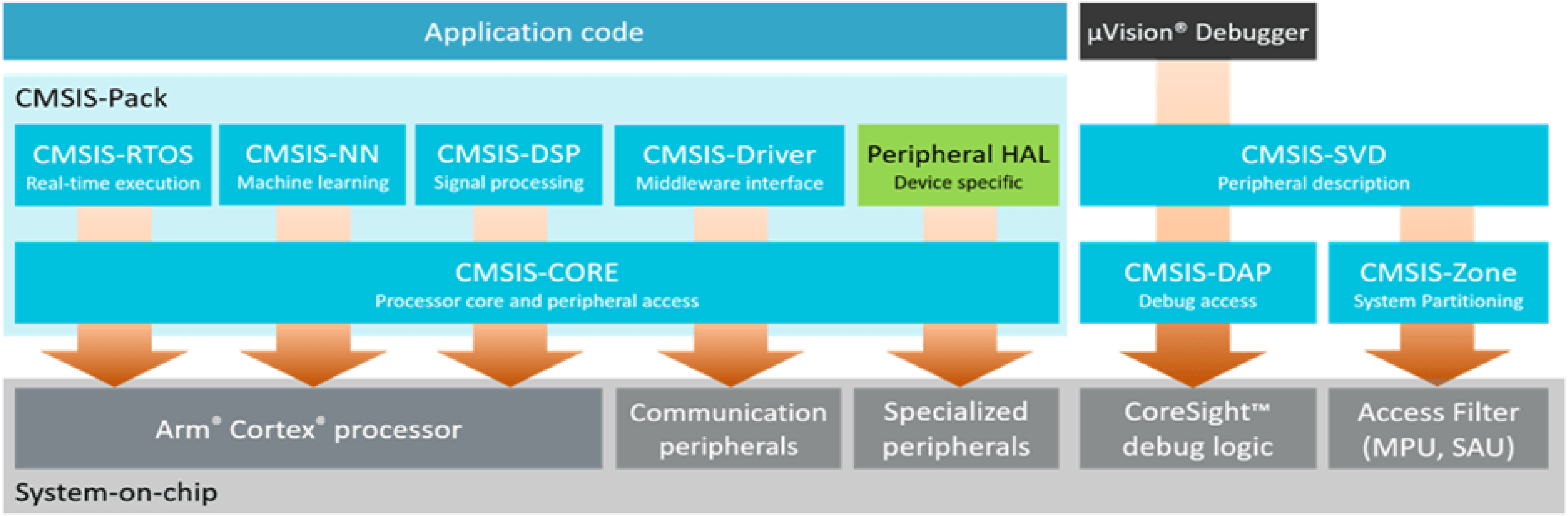
- Execute highest priority first
  - Can you have two tasks at same priority?
- Minimize latency on real-time tasks
- Assign a dollar cost for delays
  - Minimize cost

CMSIS

- The ARM® Cortex™ Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the [Cortex-M processor](https://developer.arm.com)series.
- The CMSIS enables consistent and simple software interfaces to the processor and the peripherals, simplifying software re-use, reducing the learning curve for new microcontroller developers and reducing the time to market for new devices.

- https://developer.arm.com/tools-and-software/embedded/cmsis

CMSIS



Application code

µVision® Debugger

CMSIS-Pack

CMSIS-RTOS
Real-time execution

CMSIS-NN
Machine learning

CMSIS-DSP
Signal processing

CMSIS-Driver
Middleware interface

Peripheral HAL
Device specific

CMSIS-SVD
Peripheral description

CMSIS-CORE
Processor core and peripheral access

CMSIS-DAP
Debug access

CMSIS-Zone
System Partitioning

Arm® Cortex® processor

Communication peripherals

Specialized peripherals

CoreSight™ debug logic

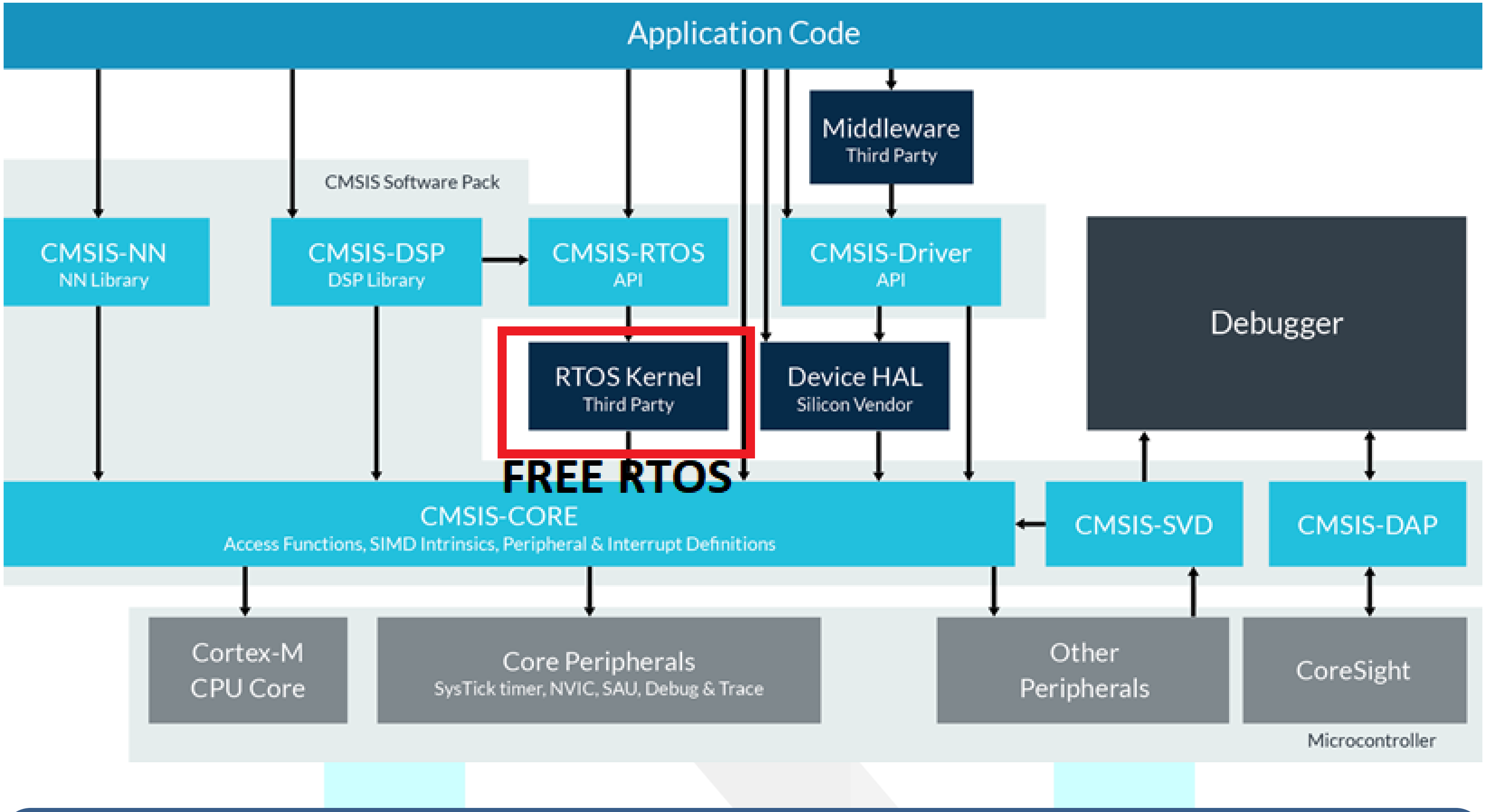
Access Filter (MPU, SAU)

System-on-chip

CMSIS

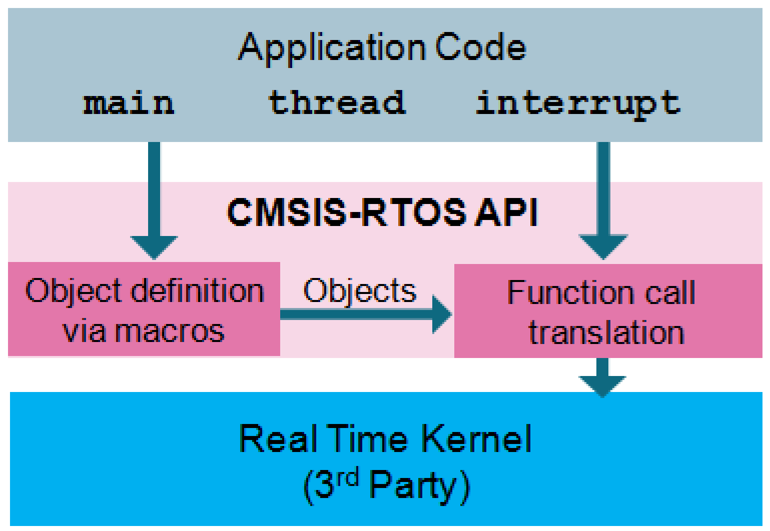
The CMSIS consists of the following components:

- **CMSIS-CORE:** provides an interface to Cortex-M0, Cortex-M3, Cortex-M4, SC000, and SC300 processors and peripheral registers
- **CMSIS-DSP:** DSP library with over 60 functions in fixed-point (fractional q7, q15, q31) and single precision floating-point (32-bit) implementation
- **CMSIS-RTOS API:** standardized programming interface for real-time operating systems for thread control, resource, and time management
- **CMSIS-SVD:** System View Description XML files that contain the programmer’s view of a complete microcontroller system including peripherals



FreeRTOS is our "RTOS Kernel" as depicted in the diagram.
We will be making calls to CMSIS-RTOS (version 2, specifically) in order to control the underlying FreeRTOS.

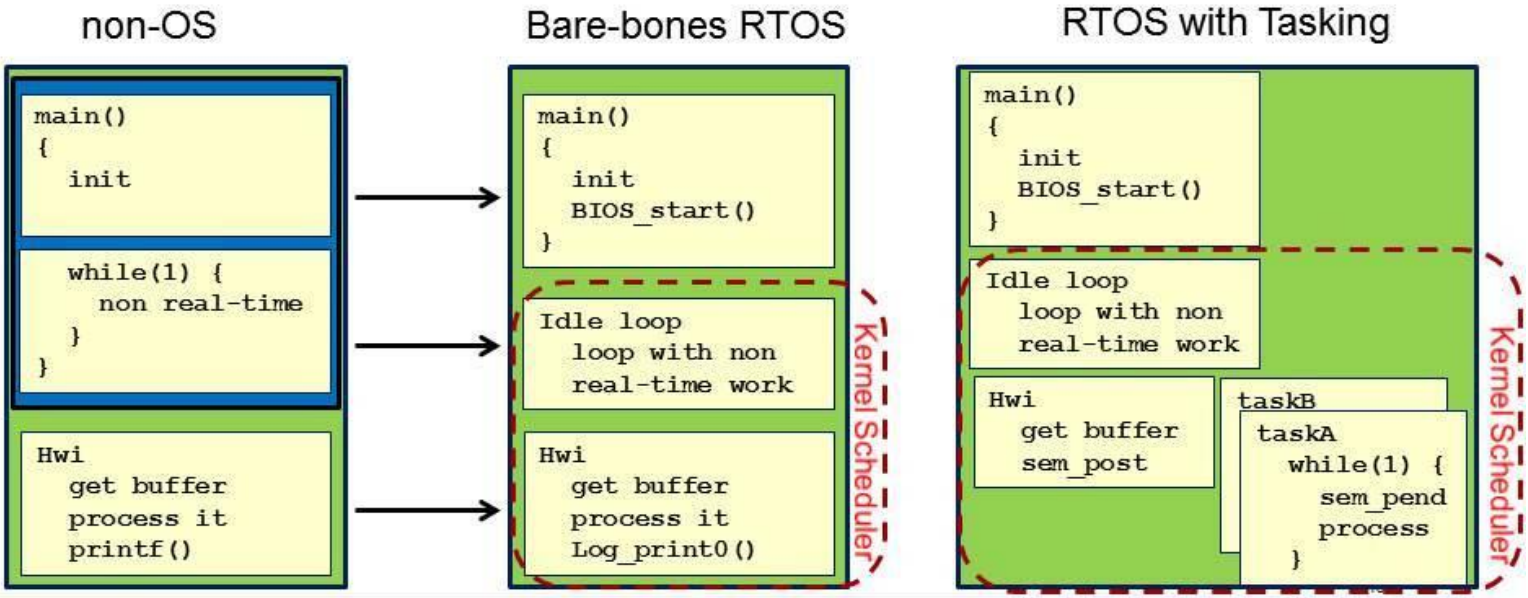
CMSIS-RTOS Interface



KERNEL MASTERS

Non-RTOS vs Bare-bones RTOS vs Full RTOS

Comparison between non-OS and RTOS execution structure.



non-OS

```
main()
{
  init

  while(1) {
    non real-time
  }
}
```

```
Hwi
  get buffer
  process it
  printf()
```

Bare-bones RTOS

```
main()
{
  init
  BIOS_start()
}
```

```
Idle loop
  loop with non
  real-time work
```

```
Hwi
  get buffer
  process it
  Log_print0()
```

Kernel Scheduler

RTOS with Tasking

```
main()
{
  init
  BIOS_start()
}
```

```
Idle loop
  loop with non
  real-time work
```

```
Hwi
  get buffer
  sem_post
```

```
taskB
```

```
taskA
  while(1) {
    sem_pend
    process
  }
```

Kernel Scheduler

RTOS Implementation:

Step1: Program divided in to tasks.

Step2: Implement periodic or non-periodic scheduler algorithm using GPTIMERS & Function Pointers

Smart Weather Monitoring System V3.0

Smart Weather Monitoring System Using RTOS (Periodic algorithm):

Task1: Every 1 sec delay read Date and time from RTC and update in to LCD.

Task2: Every 3 sec delay read Temperature from LM35 sensor and update in to LCD.

Task3: Every 5 Sec delay send temperature to Kernel Masters cloud server using Wi-Fi.