CS433: Internet of Things
NCS463: Internet of Things

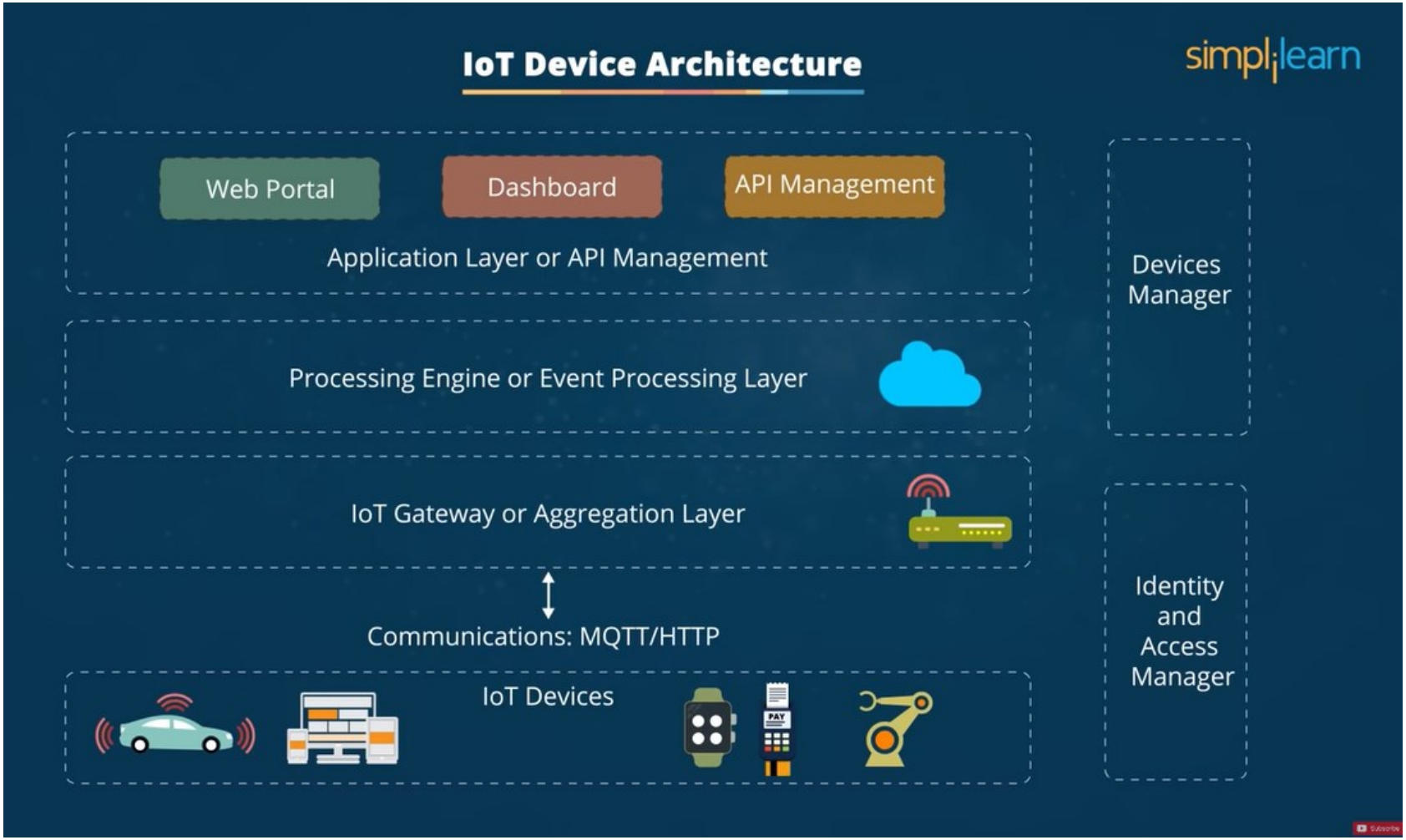
Dr. Ahmed Shalaby

http://bu.edu.eg/staff/ahmedshalaby14

Things: Sensors & Actuators

- **Things**
  - Examples
  - Design Issues
  - Interface
- **Processing Unit**
- **Programming Model**
- **Real Time Operating System (RTOS)**

Internet of Things – Architecture



Things: Sensors & Actuators

- **Sensor** is a device that measures a physical quantity

  → Input / “Read from the physical world”

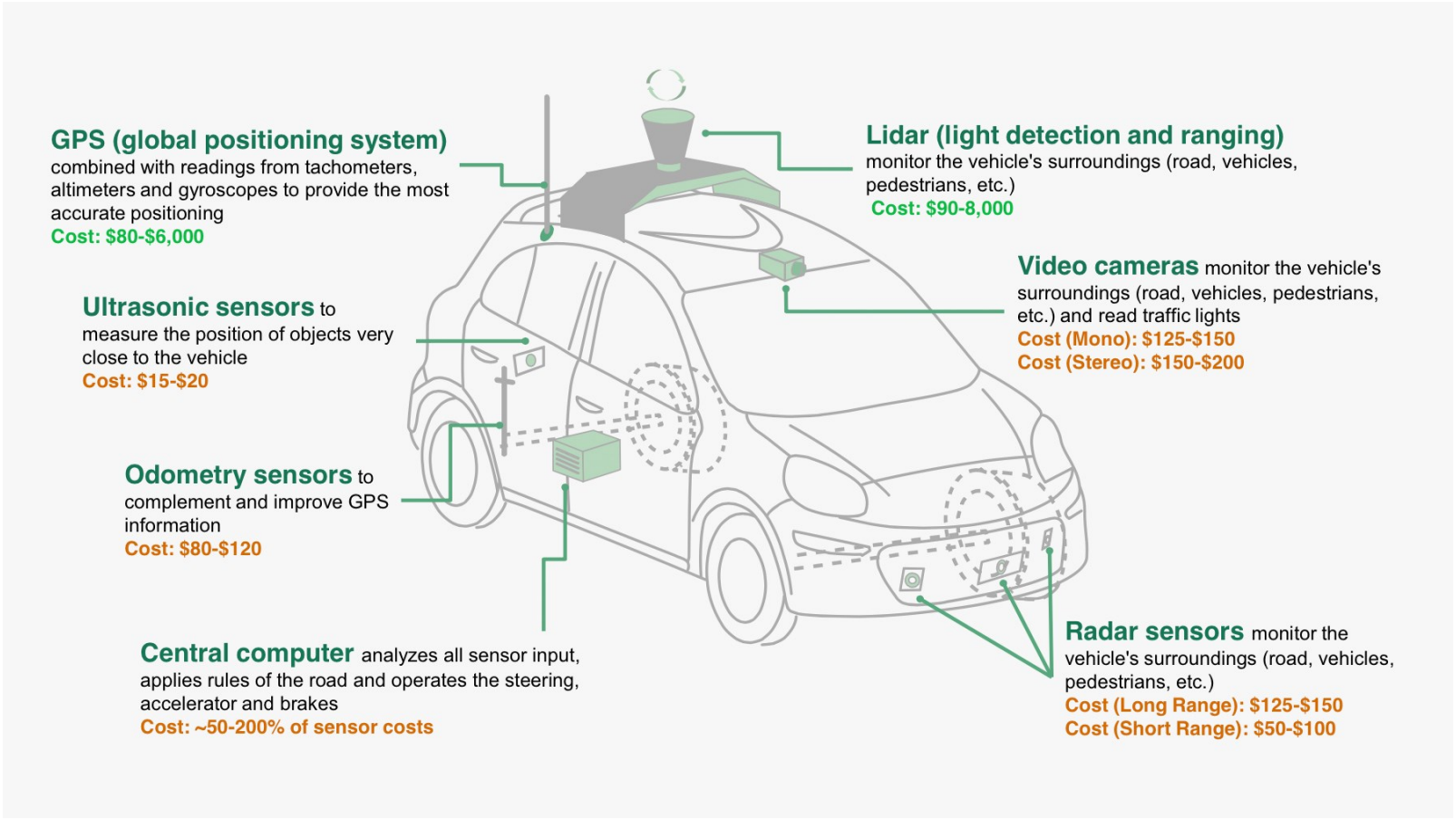
  - Cameras
  - Accelerometers
  - Gyroscopes
  - Strain gauges
  - Microphones
  - Magnetometers
  - Radar/Lidar
  - Chemical sensors
  - Pressure sensors
  - Switches

- **Actuator** is a device that modifies a physical quantity

  Output / “Write to the physical world” →

  - Motor controllers
  - LEDs, lasers
  - LCD and plasma displays
  - Loudspeakers
  - Switches
  - Valves – fluids
  - Solenoids – magnetic

Sensors & Actuators



Source: Wired Magazine

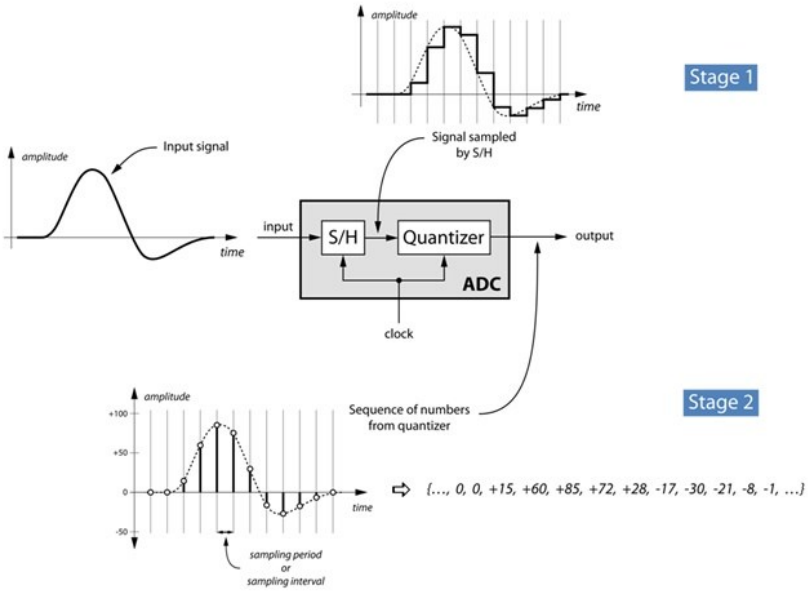
Sensors & Actuators: Design Issues

- **Calibration**
  - Relating measurements to the physical phenomenon
  - Can dramatically increase manufacturing costs
- **Nonlinearity**
  - Measurements may not be proportional to physical phenomenon
  - Correction may be required
  - Feedback can be used to keep the operating point in the linear region
- **Sampling**
  - Aliasing – insufficient sampling frequency - Nyquist sampling rate.
  - Missed events
- **Noise**
  - Analog signal conditioning
  - Digital filtering
  - Introduces latency
- **Failures**
  - Redundancy (sensor fusion problem)
  - Attacks (e.g. Stuxnet attack)

Sensors & Actuators: ADC/DAC

Analog-to-Digital Converter (ADC)

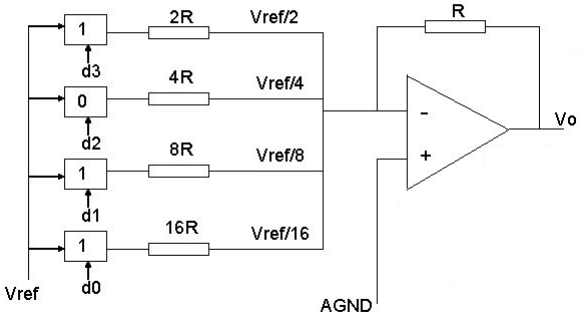
- Takes an analog signal and creates a digital representation of this signal.
- Enables the connection to many different types of sensors, such as distance sensors, temperature sensors, light-level sensors, and so on.



Digital to Analog Converter (DAC)

- takes the digital signals and creates an analog representation of these signals
- Enables the connection to many different types of actuators, such as speakers, Motor controllers, LEDs, lasers, LCD, and so on.

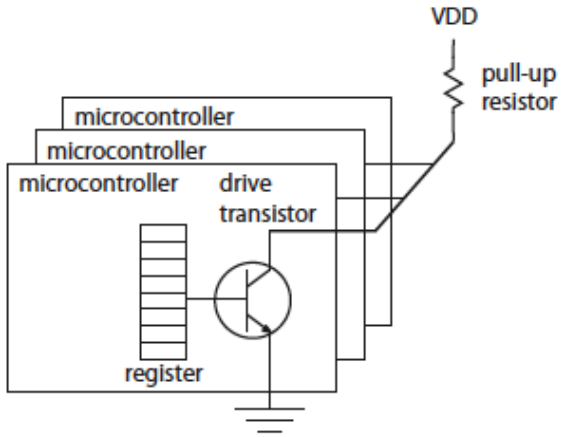
Sum currents with operational amplifier



Sensors & Actuators: Interfaces .

Simple Digital I/O: GPIO

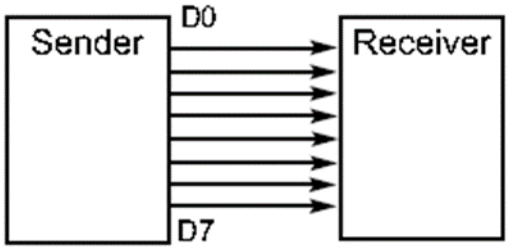
- Open collector circuits are often used on GPIO (general-purpose I/O) pins of a microcontroller.
- The same pin can be used for input and output. And multiple users can connect to the same bus.



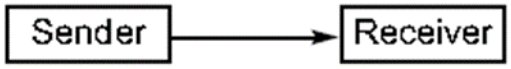
Parallel Transfer

Parallel/Serial Interface

- Parallel
  Transfer a byte of data at a time
  faster, easier
- Serial
  Transfers a bit after another
  cheaper, ideal for a long distance through

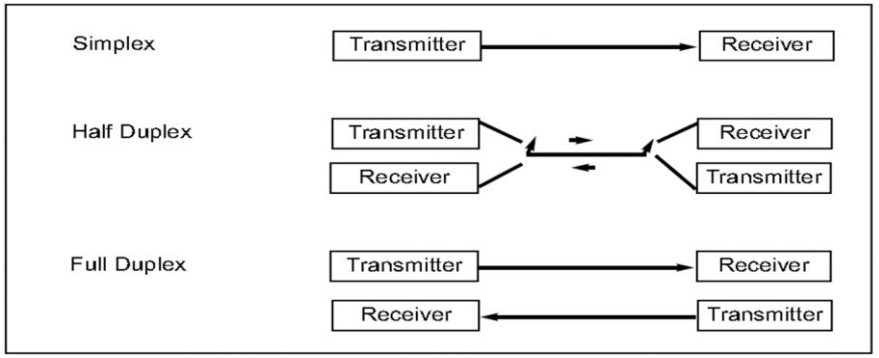


Serial Transfer



Sensors & Actuators: Interfaces ..

- **Direction**
  - Simplex: data can move only in one direction.
  - Half Duplex: data can move in two directions but not at the same time.
  - Full Duplex: data can move in two directions at the same time.

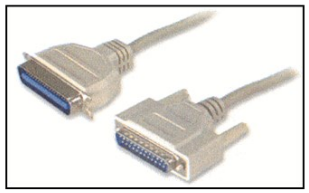


- **Synchronous vs. Asynchronous**
  - Synchronous
    - Clock pulse should be transmitted during data transmission.
    - Only one side generates a clock at the same time.
  - Asynchronous
    - Clock pulse is not transmitted.
    - The two sides should generate a clock pulse.
    - There should be a way to synchronize the two sides.

Sensors & Actuators: Interfaces …

- **Parallel (one wire per bit)**
  - ATA: Advanced Technology Attachment
  - PCI: Peripheral Component Interface
  - SCSI: Small Computer System Interface
  - …
- **Serial (one wire per direction)**
  - RS-232
  - SPI: Serial Peripheral Interface bus
  - I2C: Inter-Integrated Circuit
  - USB: Universal Serial Bus
  - SATA: Serial ATA
  - …
- **Mixed (one or more “lanes”)**
  - PCIe: PCI Express
  - users can connect to the same bus.

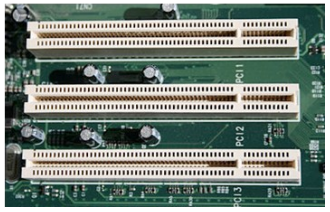
SCSI



USB

RS-232

PCI

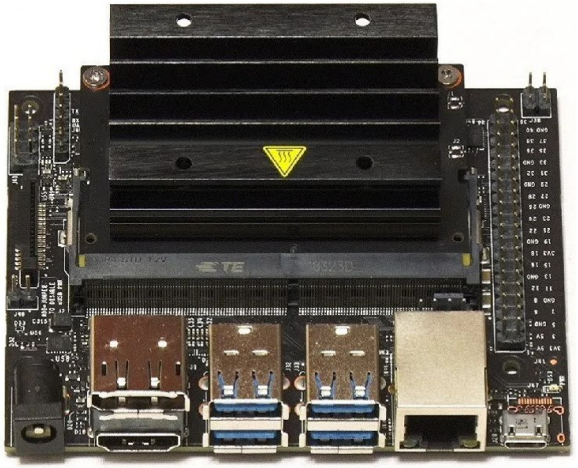


Processing Units

Arduino

Raspberry Pi

BeagleBone

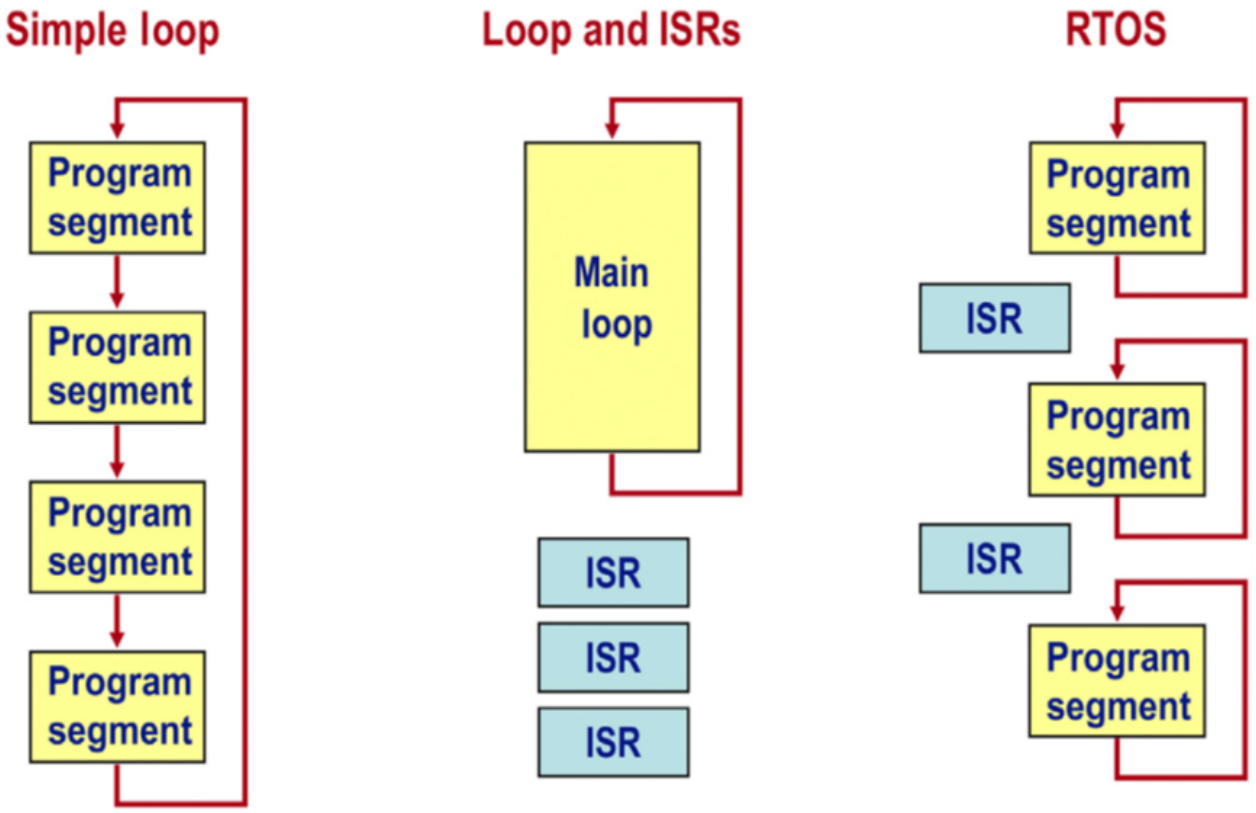


Jetson Nano

❑ PROCESSOR

- Arduino has ATmega328, runs @ 16MHz.
- The Raspberry Pi 4 has a Broadcom BCM2711 system-on-chip, and it runs on a 1.5-GHz quad-core 64-bit ARM Cortex-A72 CPU @ 1.5 GHz.
- The Beaglebone Black has AM335x ARM® Cortex-A8 @1GHz.
- The Jetson Nano runs on a quad-core ARM Cortex-A57 64-bit @ 1.42 GHz.

Processing & Programming Models.

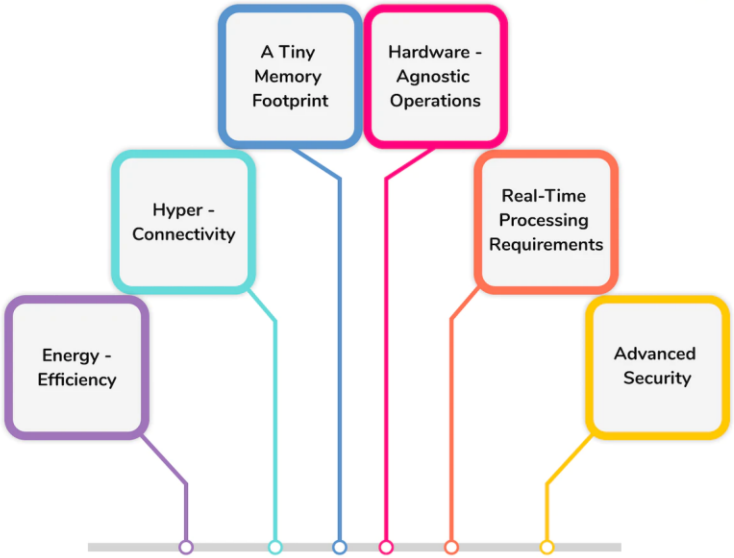


Processing & Programming Models..

- **Simple (Pooling – Loop Structure)**
  - It goes around repeatedly executing the same sequence of actions.
  - Simple - Code is reliable and easy to understand.
  - Performance / Scalability / task priorities.
- **Loop / ISR**
  - consider placing all the non-critical tasks in the main loop and locating the time-sensitive tasks in Interrupt Service Routines (ISRs).
  - Challenge is the distribution of tasks between the main loop and the ISRs.
- **RTOS (Real Time OS)**
  - Divide the application into tasks that run concurrently.
  - The essence of real-time computing is not only that the computer responds to its environment fast enough, but that it responds reliably fast enough.

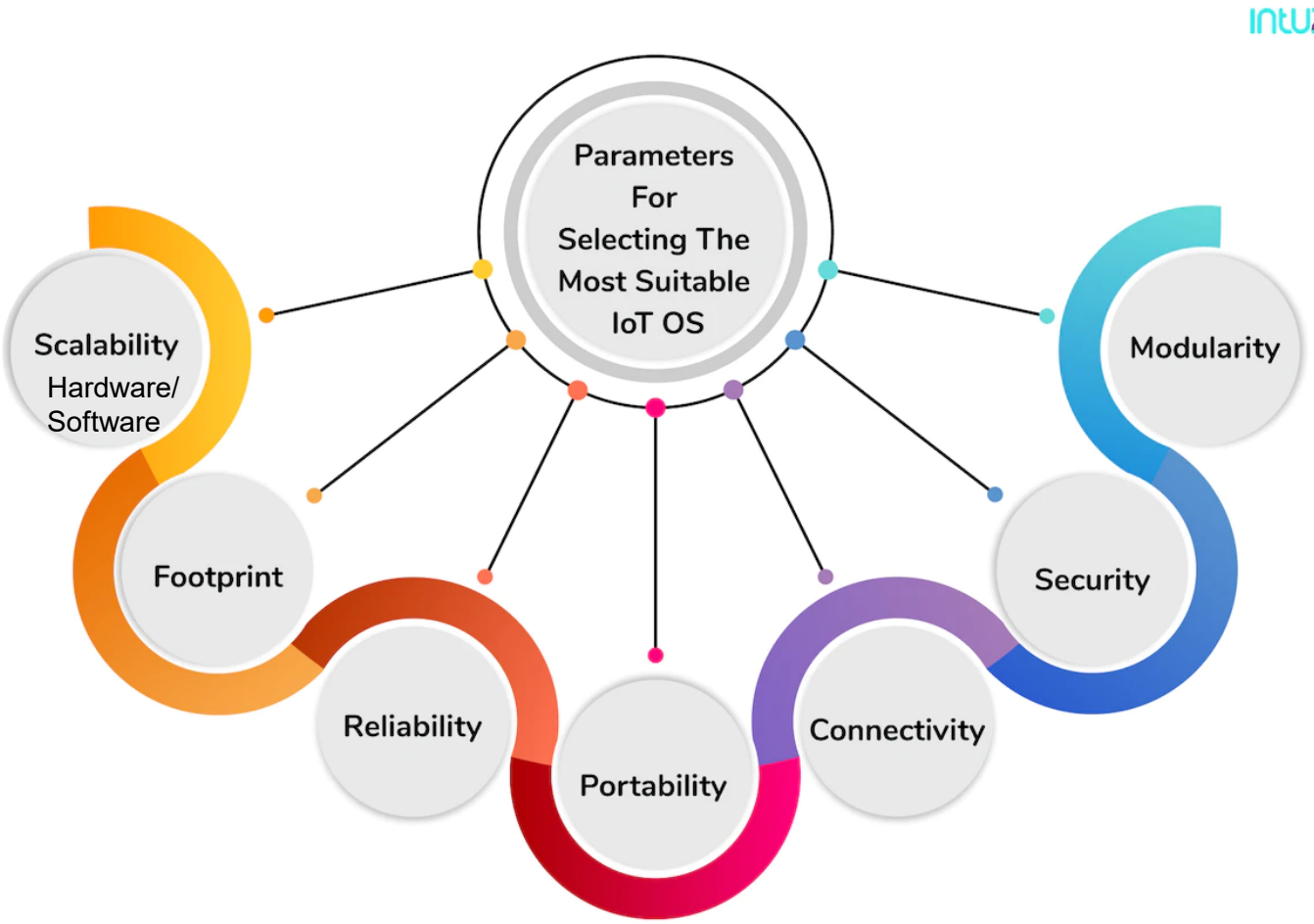
RTOS - IoT Operating Systems

- IoT OS is essential for IoT applications. It enables devices and applications to connect with other systems, such as cloud platforms and services.
- The IoT OS manages limited resources such as memory bandwidth, data volumes, and processing power to transmit, collect and store data. For example, IoT OS is used to control traffic lights, digital televisions, smart meters, ATMs, airplane controls, and elevators, among many other use cases.
- The IoT OS runs the software, and efficiently without any latency on the host IoT device.



Characteristics Of An IoT OS

Top IoT Operating Systems



Guide on Top 17 IoT Operating Systems For IoT Devices | Intuz

IoT Operating Systems Parameters .

- **Scalability:**
  The operating system must be scalable for any type of device. That means both integrators and developers need to be familiar with the operating system when it comes to gateways and nodes.
- **Footprint:**
  Since the devices will always come with a bag of constraints, it is essential to choose an operating system with low power, processing, and memory requirements. The overheads incurred by you should be minimal at the end of the day.
- **Reliability:**
  This is a critical factor to note for mission-critical systems. For instance, Industrial IoT devices are at remote locations and have to work for years without hampering business continuity. Your operating system should be able to fulfill specific certifications for IoT apps.
- **Portability:**
  Operating systems isolate apps from the specifics of the hardware, hence leading to greater portability. Usually, an OS is ported to different interface and hardware platforms to the board support package (BSP) in a standardized format, such as POSIX calls.

IoT Operating Systems Parameters . .

- **Connectivity:**
  This one is obvious, but your operating system should support connectivity protocols such as WiFi, IEEE, Ethernet, and so on. There is no point in IoT apps if they cannot connect without any hassle.
- **Security:**
  The operating system of your choice should be safe and secure to use, allowing you to add on some aspects in the form of SSL support, secure boot, components, and encryption drivers.
- **Modularity:**
  Every operating system must mandatorily have a kernel core. All other functionalities can be included as add-ons if so required by the IoT app you are building.

Top IoT Operating Systems

- **Raspbian Pi:** an operating system used by Raspberry Pi.
- **FreeRTOS:** popular, ported to more than 30 microcontroller platforms
- **Contiki:** IoT OS suitable for low-powered internet connectivity.
- **ARM Mbed OS:**

  user-friendly development environment powered by Node-RED, provides the same set of functionalities as cloud computing platforms such as Amazon Web Services (AWS) and Microsoft Azure.
- **Windows 10 IoT:**

  easy-to-use base for creating and running IoT devices, it automatically downloads and installs all the necessary drivers and programs for your device.
- **Embedded Linux:**

  An enhanced version of the Linux Kernel and an embedded graphics stack are being used by the operating system.
- **TinyOS:** low-power wireless devices.

POSIX RT Extension, Micropython, RIOT OS, Ubuntu Core, Ubuntu MATE, OSMC, Tizen, eLinux OS, RISC OS OPEN and RISC OS Pi, OpenWrt, LibreELEC

Ch6: Processes and Operating Systems

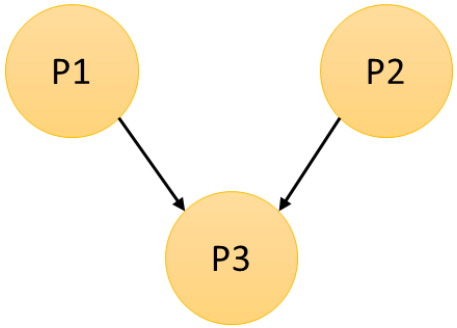
COMPUTERS AS COMPONENTS

Overview

- **Objective:**
  - **Overview:** fundamental abstractions, process and operating system, which are used in building complex applications.
- **Content :**
  - **Multiple Tasks and Multiple Processes**
  - **Preemptive Real-Time Operating Systems**
  - **Priority-Based Scheduling.**
  - **Inter-process Communication Mechanisms**
  - **Evaluating Operating System Performance**
  - **Power Management and Optimization for Processes**

❑ Multiple Tasks and Multiple Processes

- Break the system into multiple tasks in order to manage applications.
- **Tasks and Processes**
  - Process: is a single execution of a program. If we run the same program two different times, we have created two different processes.
    - A process has its own state: (registers; memory)
  - Threads: Processes that share the same address space.
  - Task: can be composed of several processes or threads.
  - Operating System: manages processes, Multiple tasks means multiple processes.



❑ Multiple Tasks and Multiple Processes

- **Multirate Systems**
  - Tasks may be synchronous or asynchronous, Synchronous tasks may recur at different rates.
  - Processes run at different rates based on the computational needs of the tasks.
  - Certain processes must be executed periodically, and each process is executed at its own rate.

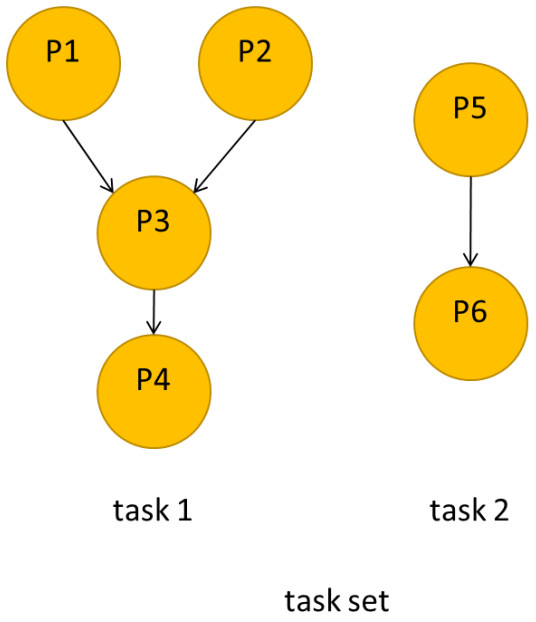
❑ Multiple Tasks and Multiple Processes

- **Real-time systems**
  - Perform a computation to conform to external timing constraints.
  - Deadline frequency:
    - Periodic. executes on (almost) every period
    - Aperiodic. executes on demand
  - Deadline type:
    - Hard: failure to meet deadlines causes system failure.
    - Soft: failure to meet deadline causes a degraded response.
    - Firm: late response is useless.
- **Timing Requirements on Processes**
  - The release time: is the time at which the process becomes ready to execute.
  - The deadline: specifies when a computation must be finished.
  - The period of a process: The time between successive executions.
  - The process's rate is the inverse of its period.

Multiple Tasks and Multiple Processes

- **Timing Requirements on Processes**
  - Before a process can become ready, All the processes on which it depends must complete and send their data to it.

  The data dependencies define a partial ordering on process execution.
  - Task Graph: A set of processes with data dependencies.
  - Communication among processes that run at different rates cannot be represented by data dependencies.



Multiple Tasks and Multiple Processes

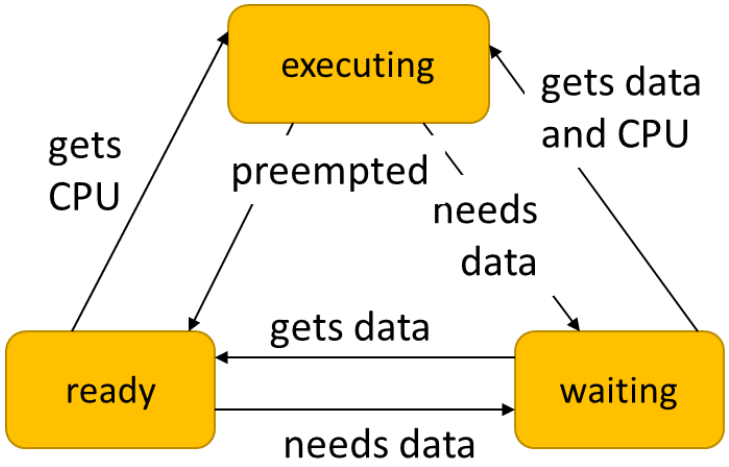
- **Sources of execution time variation:**
  - Data dependencies.
  - Memory system.
  - CPU pipeline.

- **CPU Metrics**
  - The initiation time is the time a process actually starts executing on the CPU.
  - The completion time is the time at which the process finishes its work.
  - CPU utilization: The total execution time of all processes over an interval of time.

$$U = \frac{\text{CPU time for useful work}}{\text{total available CPU time}}. \qquad U = \frac{T}{t}.$$

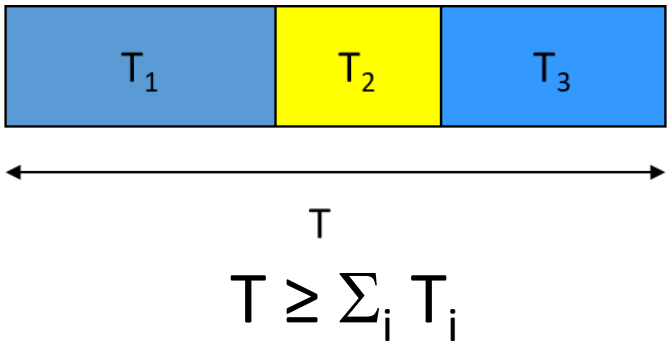
Multiple Tasks and Multiple Processes

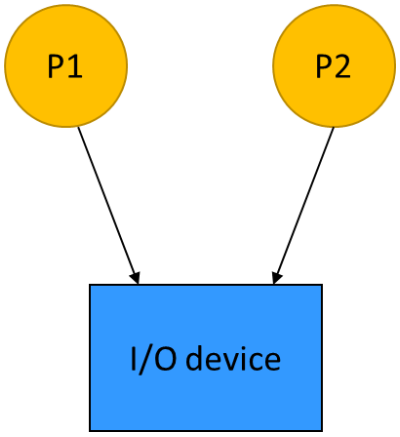
- **Process State and Scheduling**
  - Scheduling: Choosing the order of running processes.
  - Scheduling states: waiting, ready, or executing.
    - Waiting: waiting for data from an I/O device or another process or finish all its work in current period.
    - Ready: receives its required data and when it enters a new period.
    - Executing: It has all its data, is ready to run, and the scheduler selects the process as the next process to run.



Multiple Tasks and Multiple Processes

- **Scheduling Policies**
  - Choosing the right scheduling policy
    - Ensures that the system will meet all its timing requirements,
    - Has influence on the CPU horsepower required to implement the system's functionality.
  - Utilization is one of the key metrics in evaluating a scheduling policy.
  - The best policy depends on the required timing characteristics of the processes being scheduled.
  - Resource constraints make schedulability analysis NP-hard.
  - Must show that the deadlines are met for all timings of resource requests.

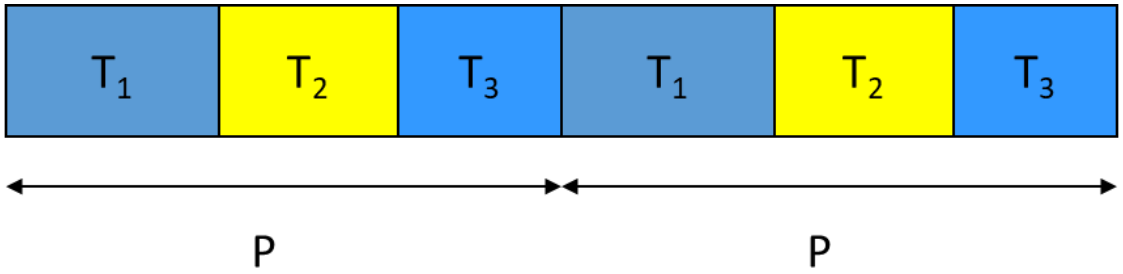




Multiple Tasks and Multiple Processes

- **Scheduling Policies**
  - For periodic processes, the **hyperperiod**, is the Least Common Multiple (LCM) of the periods of all the processes. Ex (P1—1, P2—5, Hyperperiod is 5).

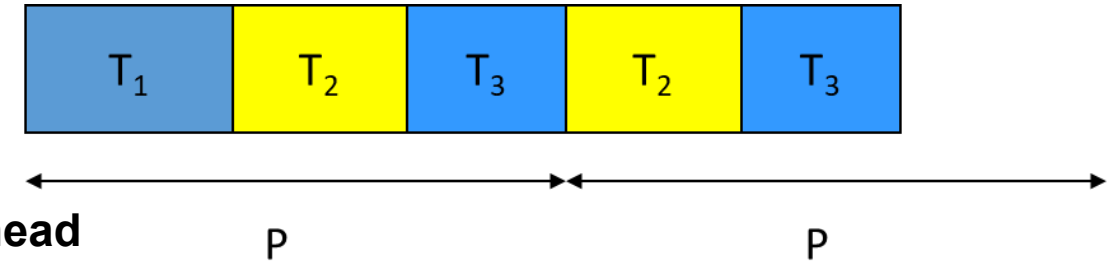
- **Cyclostatic (Time Division Multiple Access) scheduling.**
  - An interval equal to the length of the hyperperiod divided into equal-sized time slots.
  - Two factors affect utilization:
    - the number of time slots used
    - the fraction of each time slot that is used for useful work.
  - Always same CPU utilization (assuming constant process execution times).
  - Can't handle unexpected loads → Must schedule a time slot for aperiodic events.



Multiple Tasks and Multiple Processes

Scheduling Policies

- **Round robin scheduling**
  - Round robin uses the same hyperperiod as does cyclostatic.
  - If a process doesn't have useful work to do the round-robin scheduler moves on to the next process in order to fill the time slot with useful work.
  - Round-robin scheduling is often used in hardware such as buses.
  - it is very simple to implement but it provides some amount of flexibility.
  - low scheduling overhead.



- **Schedulability and overhead**

The scheduling process consumes CPU time.

- Not all CPU time is available for processes. Scheduling overhead must be taken into account for the exact schedule. May be ignored if it is a small fraction of the total execution time.

Multiple Tasks and Multiple Processes

- **Running Periodic Processes**
  - Find a programming technique that allows us to run periodic processes.
  - We could pad the loop with useless operations (NOPs) to make the execution time of an iteration equal to the desired period.
  - A timer is a much more reliable way to control the execution of the loop. We would probably use the timer to generate periodic interrupts.
  - Several timers are useful to execute different processes at different rates. Each timer to each rate.
  - A Counters to divide the counter rate. But only for processes at rates simple multiples of each other.

Timed loop

```
void pall(){
  p1();
  p2();
}
```

Multiple timers

```
void pA(){ /* rate A */
  p1();
  p3();
}
void B(){ /* rate B */
  p2();
  p4();
  p5();
}
```

Timer + counter

```
int p2count = 0;
void pall(){
  p1();
  if (p2count >= 2) {
        p2();
        p2count = 0;
        }
  else p2count++;
  p3();
}
```

Multiple Tasks and Multiple Processes

- **Running Periodic Processes**
  - Find a programming technique that allows us to run periodic processes.
  - We could pad the loop with useless operations (NOPs) to make the execution time of an iteration equal to the desired period.
  - A timer is a much more reliable way to control the execution of the loop. We would probably use the timer to generate periodic interrupts.
  - Several timers are useful to execute different processes at different rates. Each timer to each rate.
  - A Counters to divide the counter rate. But only for processes at rates simple multiples of each other.

Timed loop

```
void pall(){
  p1();
  p2();
}
```

Multiple timers

```
void pA(){ /* rate A */
  p1();
  p3();
}
void pB(){ /* rate B */
  p2();
  p4();
  p5();
}
```

Timer + counter

```
int p2count = 0;
void pall(){
  p1();
  if (p2count >= 2) {
        p2();
        p2count = 0;
        }
  else p2count++;
  p3();
}
```

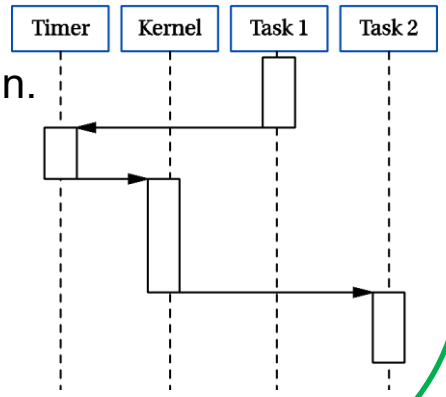
All of these implementations are insufficient.
Need better control over timing.

Preemptive Real-Time Operating Systems

- OS controls resources:
  - who gets the CPU;
  - when I/O takes place;
  - how much memory is allocated.
- OS needs to keep track of:
  - process priorities;
  - scheduling state;
  - process activation record.
- Processes may be created:
  - statically before the system starts;
  - dynamically during execution.
- RTOS executes processes-based timing constraints provided by the system design.
- meet timing constraints accurately is to build a preemptive OS and to use priorities to control what process runs at any given time.
- The main advantage of preemptive scheduling is that it provides better responsiveness to interactive tasks and ensures that critical tasks are executed as soon as possible. This is especially important in real-time operating systems and embedded systems, where certain tasks need to be performed within strict deadlines.

Preemptive Real-Time Operating Systems .

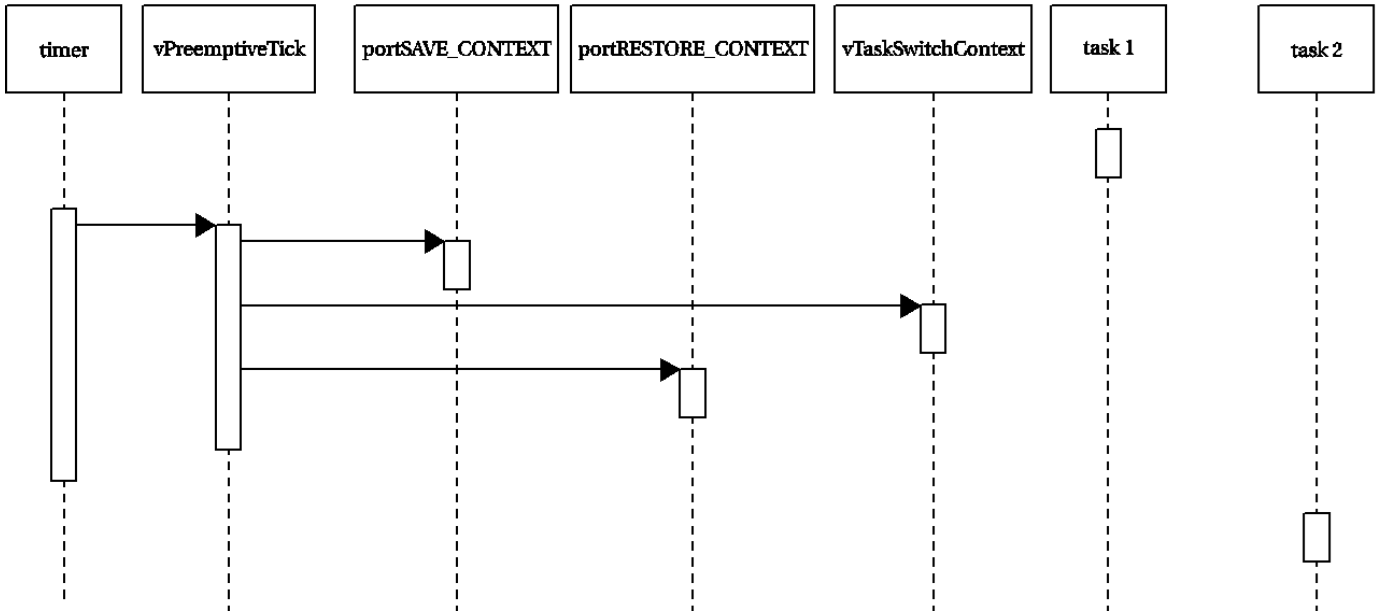
- **Preemption (**Interrupt execution**)**
  - Preemption is an alternative to the C function call as a way to control execution.
  - Create new routines that allow us to jump from one subroutine to another at any point in the program.
  - Use a timer in moving between functions whenever necessary based on the system's timing constraints.
  - Kernel: is part of the OS that determines what process is running.
  - Context: The set of registers that define a process's state.
  - Context Switching: switching from one process's register set to another.
  - Process control block: The data structure that holds the state of the process.
- **Priorities**
  - Kernel selects the highest priority process that is ready to run.
  - Kernel performs a context switch to the new context.



Preemptive Real-Time Operating Systems..

- **Processes and Context**

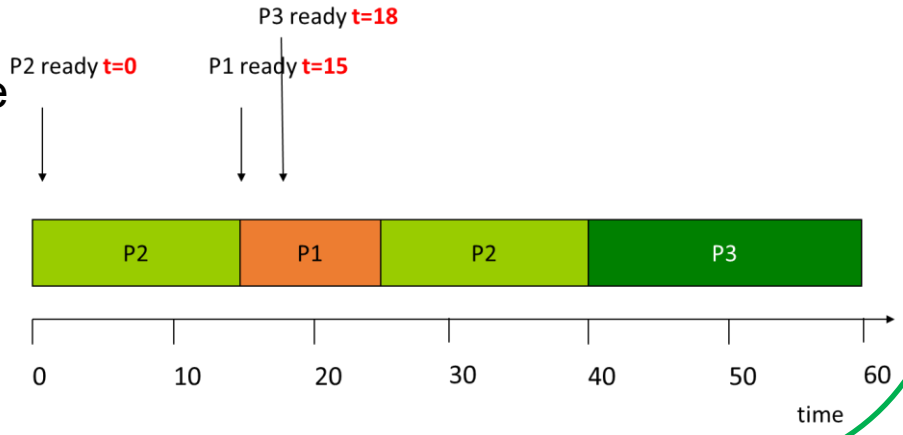
freeRTOS.org context switch




```
portSAVE_CONTEXT(); /* Save the context of the current task. */
vTaskIncrementTick(); /* Increment the tick count - this may wake a task. */
vTaskSwitchContext(); /* Find the highest priority task that is ready to run. */
portRESTORE_CONTEXT();
```

Priority-Based Scheduling

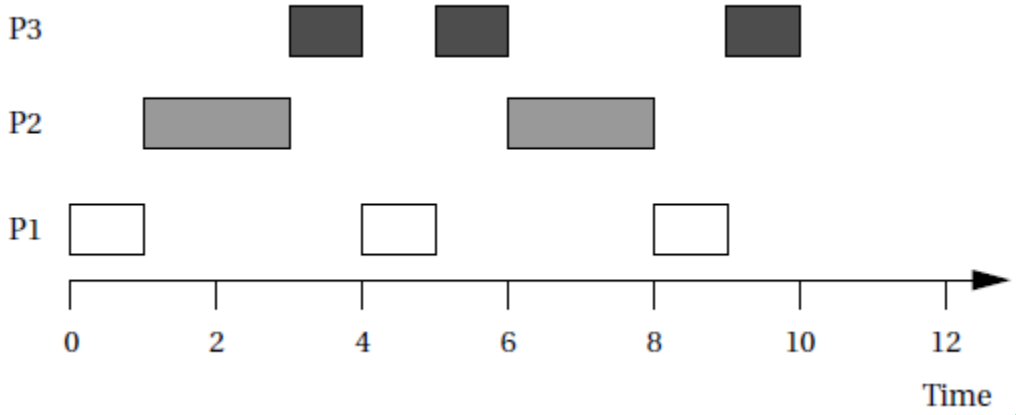
- Each process has a priority.
- CPU goes to the highest-priority process that is ready.
- Determine an algorithm by which to assign priorities to processes.
- Two major ways to assign priority (Static priority, Dynamic Priority).
  - fixed priority.
  - time-varying priorities.
- Priority-driven scheduling Rules
  - Each process has a fixed priority (1 highest);
  - highest-priority ready process gets CPU;
  - process continues until done.
- Priority-driven scheduling Example
  - P1: priority 1, execution time 10
  - P2: priority 2, execution time 30
  - P3: priority 3, execution time 20



Priority-Based Scheduling .

- **Rate-Monotonic Scheduling (RMS)**
  - RMS is static scheduling.
  - provides the highest CPU utilization while ensuring that all processes meet their deadlines.
  - All processes run periodically on a single CPU.
  - **Context switching time is ignored**. (Zero context switch time)
  - There are no data dependencies between processes
  - **The execution time for a process is constant**.
  - The highest-priority ready process is always selected for execution.

Process	Execution time	Period
P1	1	4
P2	2	6
P3	3	12



Priority-Based Scheduling ..

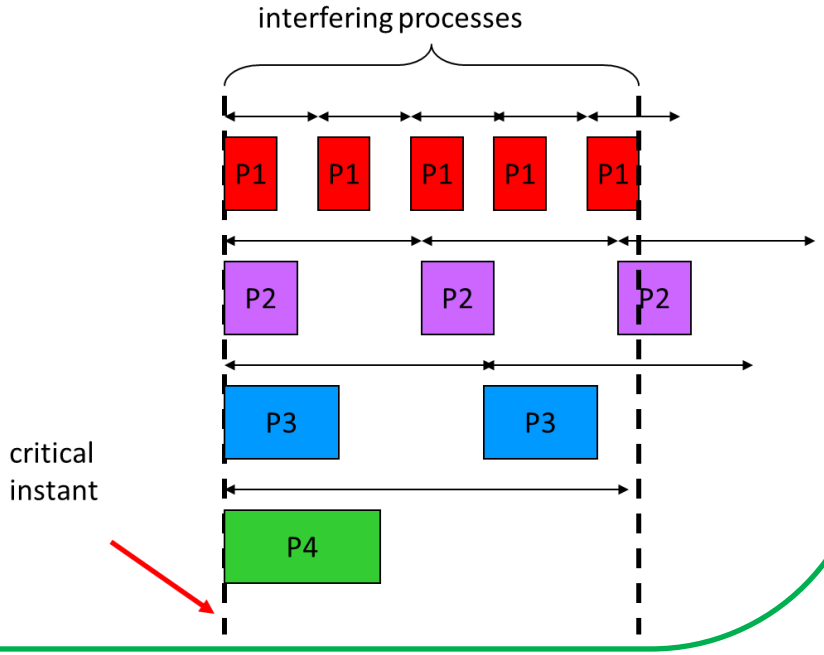
- **Rate-Monotonic Scheduling (RMS)**
  - The process with the shortest period should always be given higher priority.
  - priority inversely proportional to period;
  - RMS is the optimal static-priority schedule.
  - RMS does not always allow the system to use 100% of CPU.

Critical instant: scheduling state that gives the worst response time.

Critical instant occurs when all higher-priority processes are ready to execute.

Efficient implementation:

scan processes;
choose the highest-priority active process.



Priority-Based Scheduling ...

- **Earliest-Deadline-First Scheduling (EDF)**
  - Changes process priorities during execution based on initiation times.
  - **It assigns priorities in order of deadline**. The highest-priority process is the one closest to its deadline.
  - EDF can use 100% of the CPU.
  - The major problem is keeping the processes sorted by time to deadline, which requires recalculating processes at every timer interrupt.

Process	Execution time	Period
P1	1	3
P2	1	4
P3	2	5

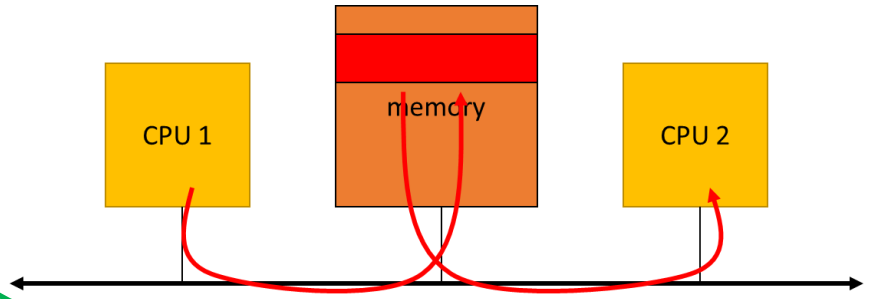
Time	Running process	Deadlines
0	P1	
1	P2	
2	P3	P1
3	P3	P2
4	P1	P3
5	P2	P1
6	P1	
7	P3	P2
8	P3	P1
9	P1	P3

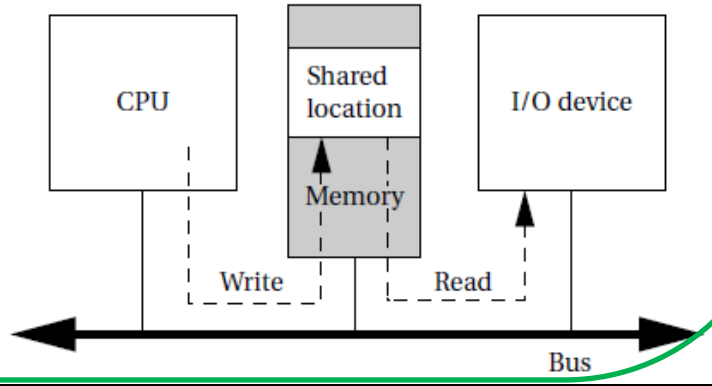
Priority-Based Scheduling ….

- **Rate-Monotonic Scheduling (RMS) Vs. Earliest-Deadline-First Scheduling (EDF)**
  - EDF extra higher Utilization of CPU But Complex to implement.
  - RMS easier to ensure all deadlines But lower CPU Utilization.
- **Priority Scheduling Problems ?**
  - Missing deadlines, Solutions:
    - Get a faster CPU
    - Redesign the processes to take less execution time.
    - Rewrite the specification to change the deadline.
  - Priority inversion, in which a low-priority process blocks the execution of a higher priority process by keeping hold of its resource. Solution:
    - promote the priority of any process when it requests a resource from the OS.
  - no data dependencies between processes assumption is wrong.
- **Scheduling for low power - DVFS:**
  - First set the clock speed to meet the performance goal in the critical interval.
  - Set clock speed for less-critical intervals in the order of importance.

Inter-process Communication Mechanisms

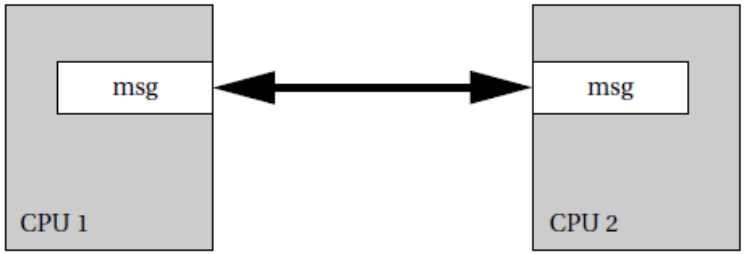
- Process can send a communication in one of two ways: blocking or nonblocking.
  - blocking communication: the process goes into the waiting state until it receives a response.
  - Nonblocking communication: allows the process to continue execution after sending the communication.
- Major styles of interprocess communication: shared memory and message passing.
- **Shared Memory Communication**
  - the CPU wants to send data to the device, it writes to the shared location. The I/O device then reads the data from that location.
  - To avoid shared location conflict between CPU and I/O devices
    - There must be a flag that tells the CPU when the data from the I/O device is ready in order.
    - an atomic test-and-set operation first reads a location and then sets it to a specified value atomic (no interruption).





Inter-process Communication Mechanisms .

- **Message Passing Communication**
  - Each communicating entity has its own message send/receive unit.
  - The message is not stored on the communications link, but rather at the senders/ receivers at the endpoints.
  - freeRTOS.org queues
    - Queues can be used to pass messages.
    - Operating system manages queues.



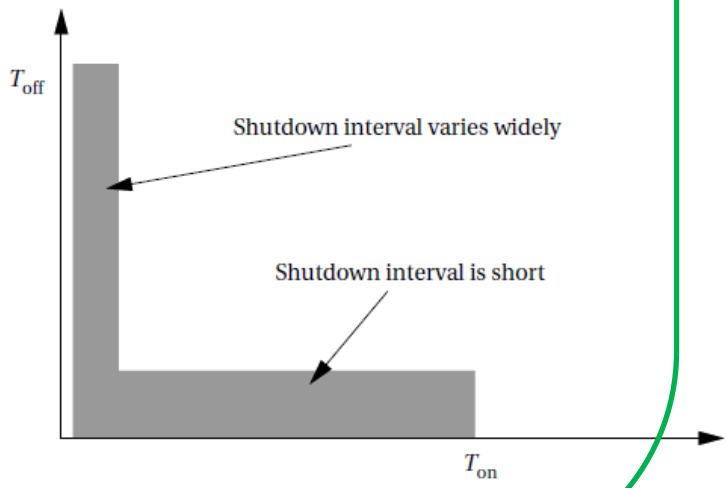
- **Signals**
  - A signal is analogous to a software interrupt.
  - A signal is generated by a process and transmitted to another process by the operating system. It changes the flow of control but does not pass parameters.
  - Unix ^c sends a kill signal to process.

Evaluating Operating System Performance

- Context switching Time.
  - In practice, OS context switch overhead is small (hundreds of clock cycles) relative to many common task periods (ms – ms).
- Interrupt latency and critical sections.
  - Interrupts are turned off in the critical section.
  - Long critical sections add software delays to interrupt latency.
  - Interrupt handling latency has non-zero hardware latency.
  - Interrupt service handler (ISH) is called at interrupt and provides minimal functions.
  - Interrupt service routine (ISR) is a process invoked by ISH and performs most of the device handling and takes time to execute.
- Interrupt priorities and interrupt latency.
- Cache Conflict problem.
- RTOS simulation
  - Some RTOSs provide scheduling simulators.
  - Schedule a mix of processes using I/O traces.
- The Process time variation due to execution (Average, Best, and Worst).
  - Even if individual processes are well-behaved, processes may interfere with each other.

Power Management and Optimization for Processes

- The RTOS and system architecture can use static and dynamic power management mechanisms to help manage the system's power consumption.
- Power modes requires an analysis of the overall system activity.
  - Avoiding a power-down mode can cost unnecessary power.
  - Powering down too soon can cause severe performance penalties.
- Predictive shutdown: The goal is to predict when the next request will be made and to start the system just before that time.
- Several predictive techniques are possible.
- A very simple technique is to use fixed times.
  - if the system does not receive inputs during an interval T_{on}, it shuts down.
  - a powered-down system waits for a period T_{off} before returning to the power-on mode



Conclusion

- Writing a single program that simultaneously satisfies deadlines at multiple rates is too difficult because the control structure of the program.
- fundamental abstractions, process and operating system, which are used in building complex applications.
- A process is a single thread of execution.
- Pre-emption is the act of changing the CPU's execution from one process to another.
- A scheduling policy is a set of rules that determines the process to run.
- Rate-monotonic scheduling (RMS) is a simple but powerful scheduling policy.
- Interprocess communication mechanisms allow data to be passed reliably between processes.

Raspbian - Raspberry Pi OS

- Raspbian is an unofficial port of **Debian** wheezy armhf with compilation settings adjusted to produce code that uses "hardware floating point", the "hard float" ABI and will run on the Raspberry Pi.
- Debian is a free operating system for your computer and includes the basic set of programs and utilities that make your computer run along with many thousands of other packages. **Debian** has a reputation within the Linux community for being very high-quality, stable, and scalable. Debian also has an extensive and friendly user community that can help new users with support for practically any problem.
- The port is necessary because the official Debian wheezy armhf release is compatible only with versions of the ARM architecture later than the one used on the Raspberry Pi (**ARMv7**-A CPUs and higher, vs the Raspberry Pi's ARMv6 CPU).
- The goal of Raspbian is to become the leading OS of choice for all users of the Raspberry Pi.
- Raspbian images are produced by various people. For newcomers we recommend the images provided by the Raspberry Pi foundation.
- Raspbian tries to stay as close to Debian as reasonably possible. Any information you find that applies to Debian will almost certainly apply to the same version of Raspbian.

Source: Raspbian

POSIX – Realtime OS

- Unix was developed in the 1960s at Bell Laboratories to support text processing.
- POSIX (Portable Operating System Interface) is a standard version of Unix.
- Processes may run under different scheduling policies.
- POSIX.1b: Real-time extensions (IEEE Std 1003.1b-1993, later appearing as librt—the Realtime Extensions library)[9]
  - Priority Scheduling
  - Real-Time Signals
  - Clocks and Timers
  - Semaphores
  - Message Passing
  - Shared Memory
  - Asynchronous and Synchronous I/O
  - Memory Locking Interface

Assignment

– Provide a summary of Contiki OS.

Hint:

Contiki Protothreads - YouTube