Real-time Operating System(RTOS) and Scheduling Algorithms

Embedded Systems – 5

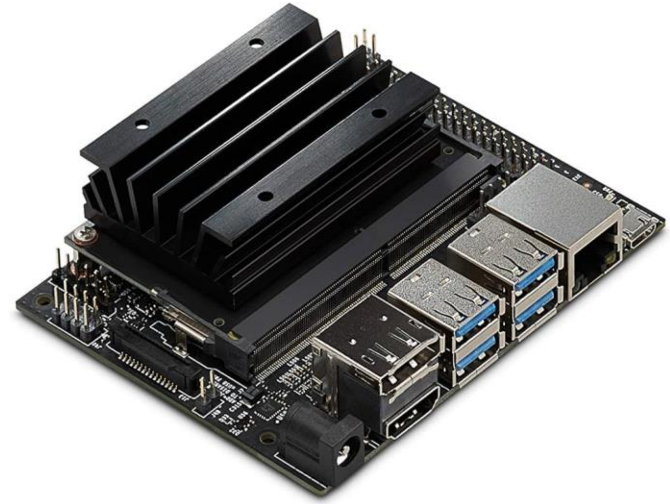
Hemant Ghayvat
Department of Computer Science and Media Technology
Hemant.ghayvat@lnu.se

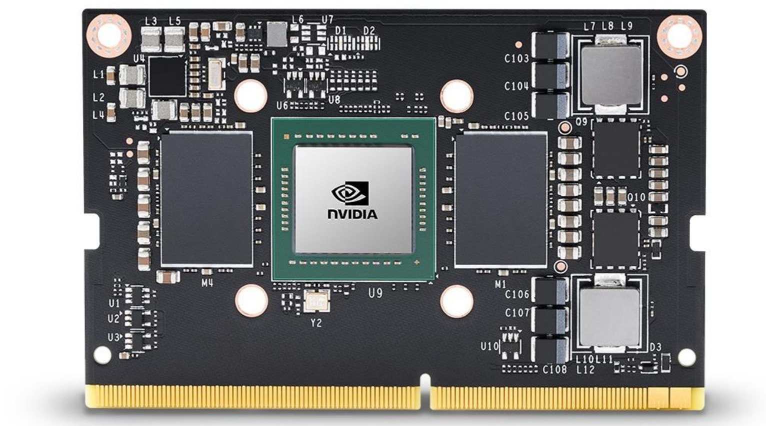
A system is said to be **Real Time** if it is required to complete it’s work & deliver it’s services according to specific deadline .

Real time operation generally means the execution keeps up with the pace of events

- A speech or video sample of 10 second, if processed in 10 second or in less

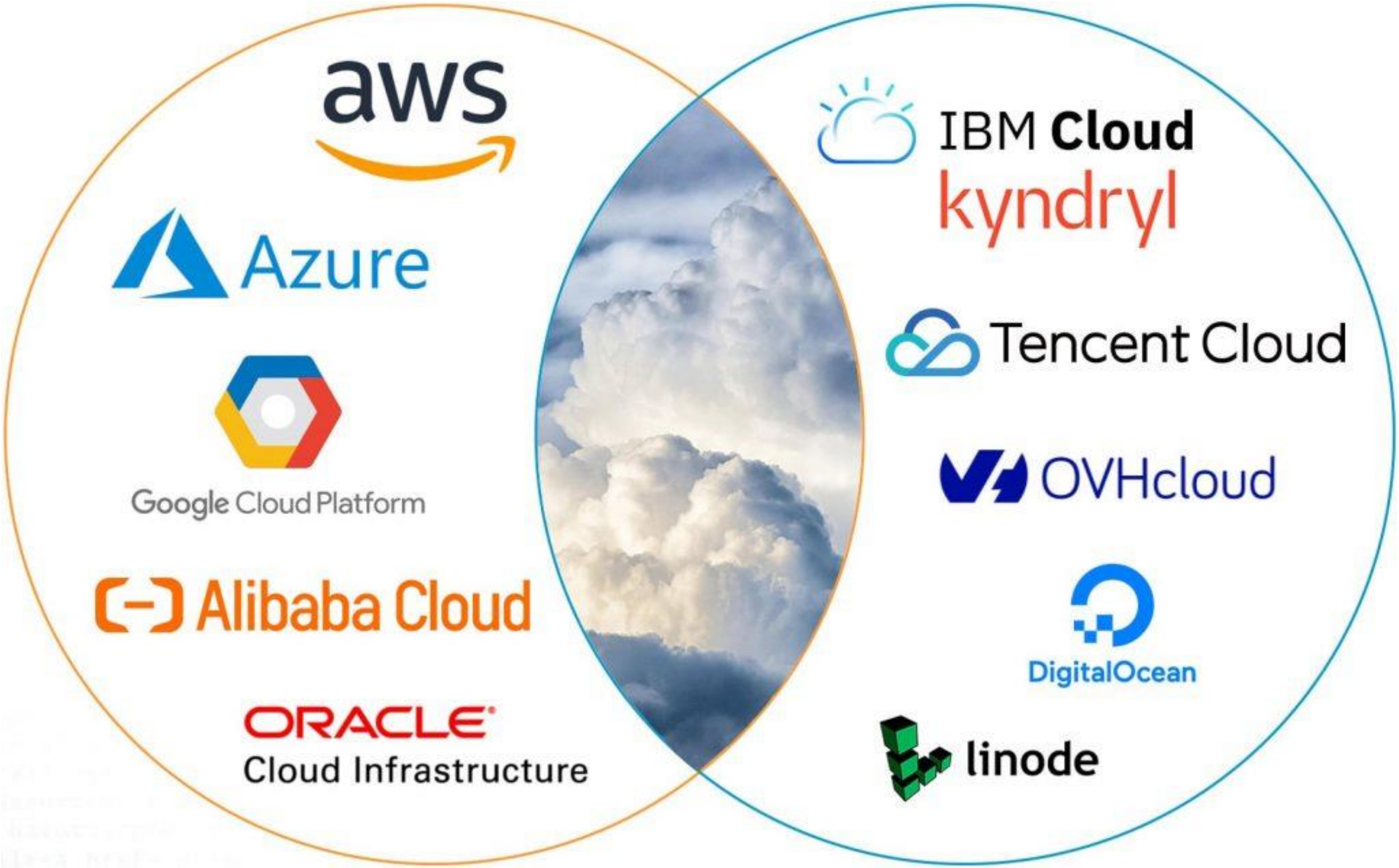
Example – Airbag System in Cars (Hard Real-Time)
Anti-lock Braking System (ABS) in Vehicles (Hard Real-Time)



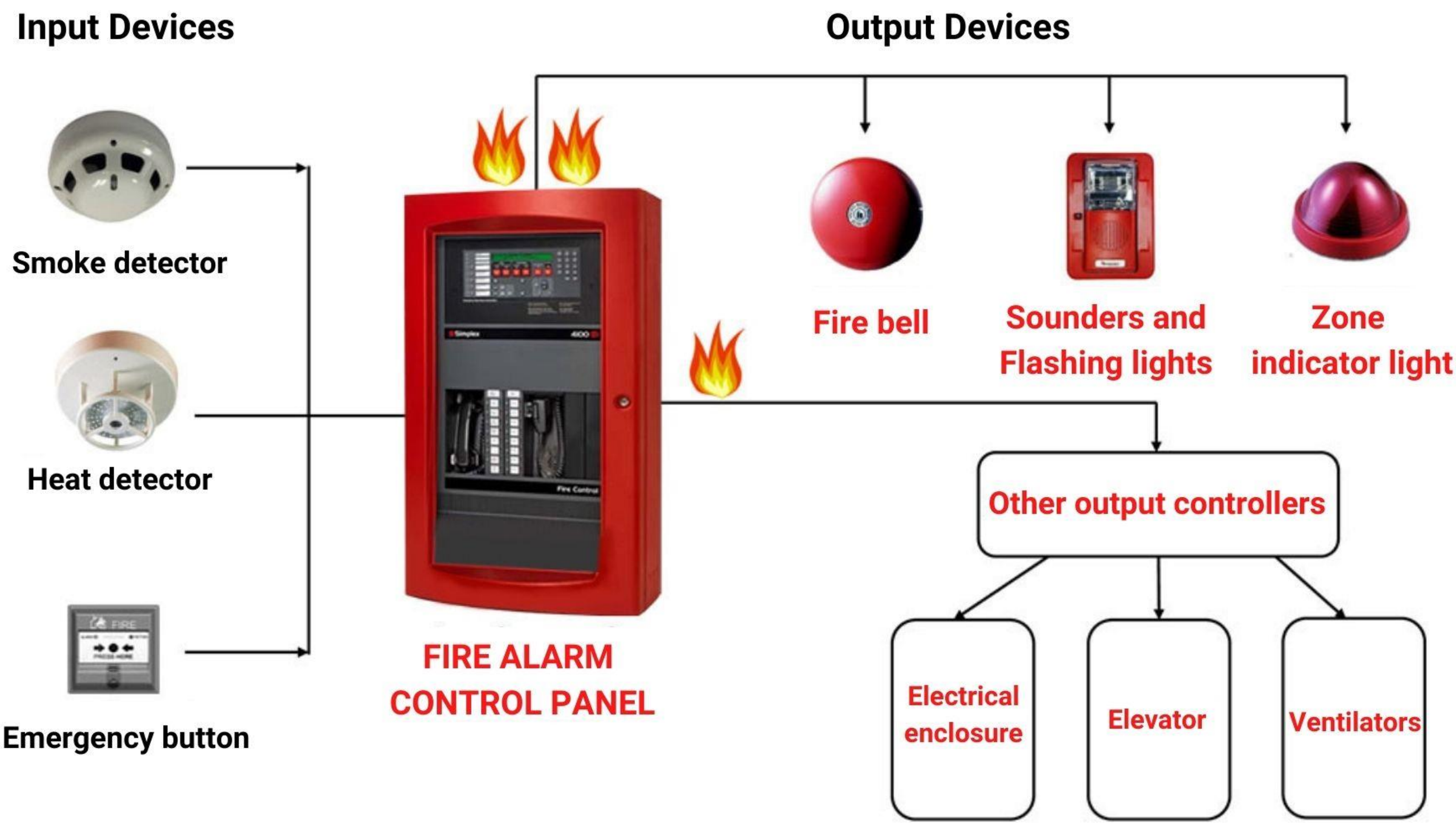


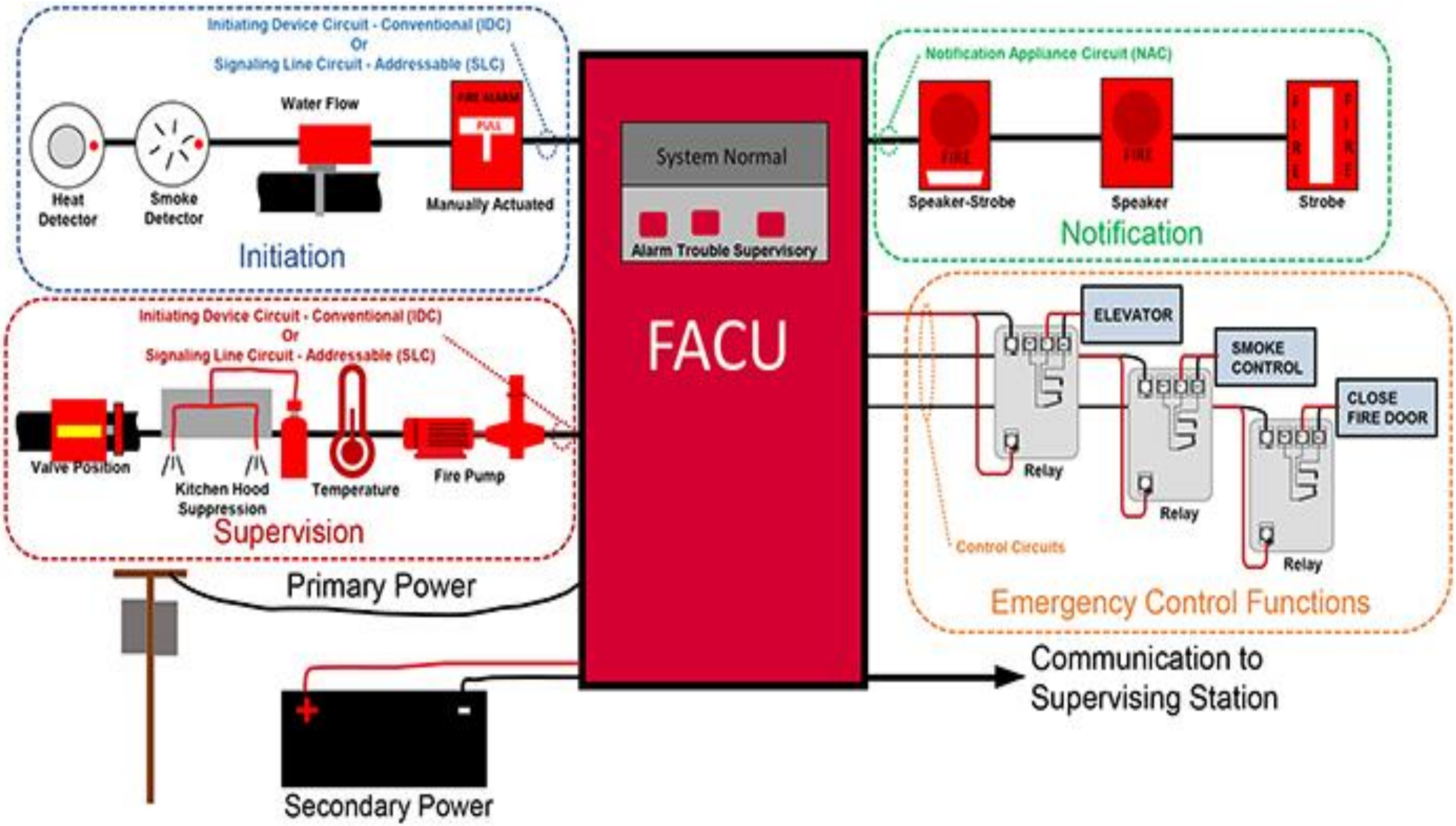
Microcontrollers

AI-Microcontrollers

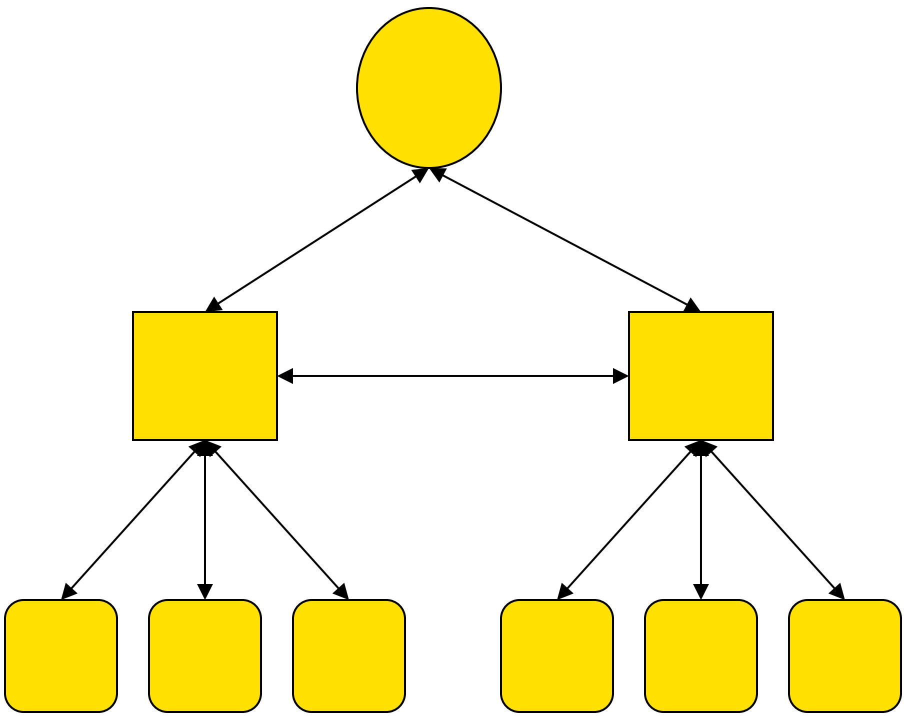

aws
Azure
Google Cloud Platform
Alibaba Cloud
ORACLE
Cloud Infrastructure
IBM Cloud
kyndryl
Tencent Cloud
OVHcloud
DigitalOcean
linode

CONVENTIONAL FIRE ALARM




Initiating Device Circuit - Conventional (IDC)
Or
Signaling Line Circuit - Addressable (SLC)
Water Flow
Heat Detector
Smoke Detector
Manually Actuated
Initiation
System Normal
Alarm Trouble Supervisory
FACU
Notification Appliance Circuit (NAC)
Speaker-Strobe
Speaker
Strobe
Notification
Initiating Device Circuit - Conventional (IDC)
Or
Signaling Line Circuit - Addressable (SLC)
Valve Position
Kitchen Hood Suppression
Temperature
Fire Pump
Supervision
ELEVATOR
SMOKE CONTROL
CLOSE FIRE DOOR
Relay
Relay
Relay
Control Circuits
Emergency Control Functions
Primary Power
Communication to Supervising Station
Secondary Power

Fire alarm system: an example



Central server

TCP/IP over radio

Controllers: ARM based

Low bandwidth radio links

Sensors: microcontroller based

Fire Alarm System

Problem

- Hundreds of sensors, each fitted with Low Range Wireless
  - Sensor information to be logged in a server & appropriate action initiated

Possible Solution

- Collaborative Action
  - Routing
    - Dynamic – Sensors/controllers may go down
    - Auto Configurable – No/easy human intervention.
    - Less Collision/Link Clogging
    - Less no of intermediate nodes
      - » Fast Response Time
    - Secure


EMBEDDED SYSTEM
AND ITS REAL TIME
APPLICATIONS

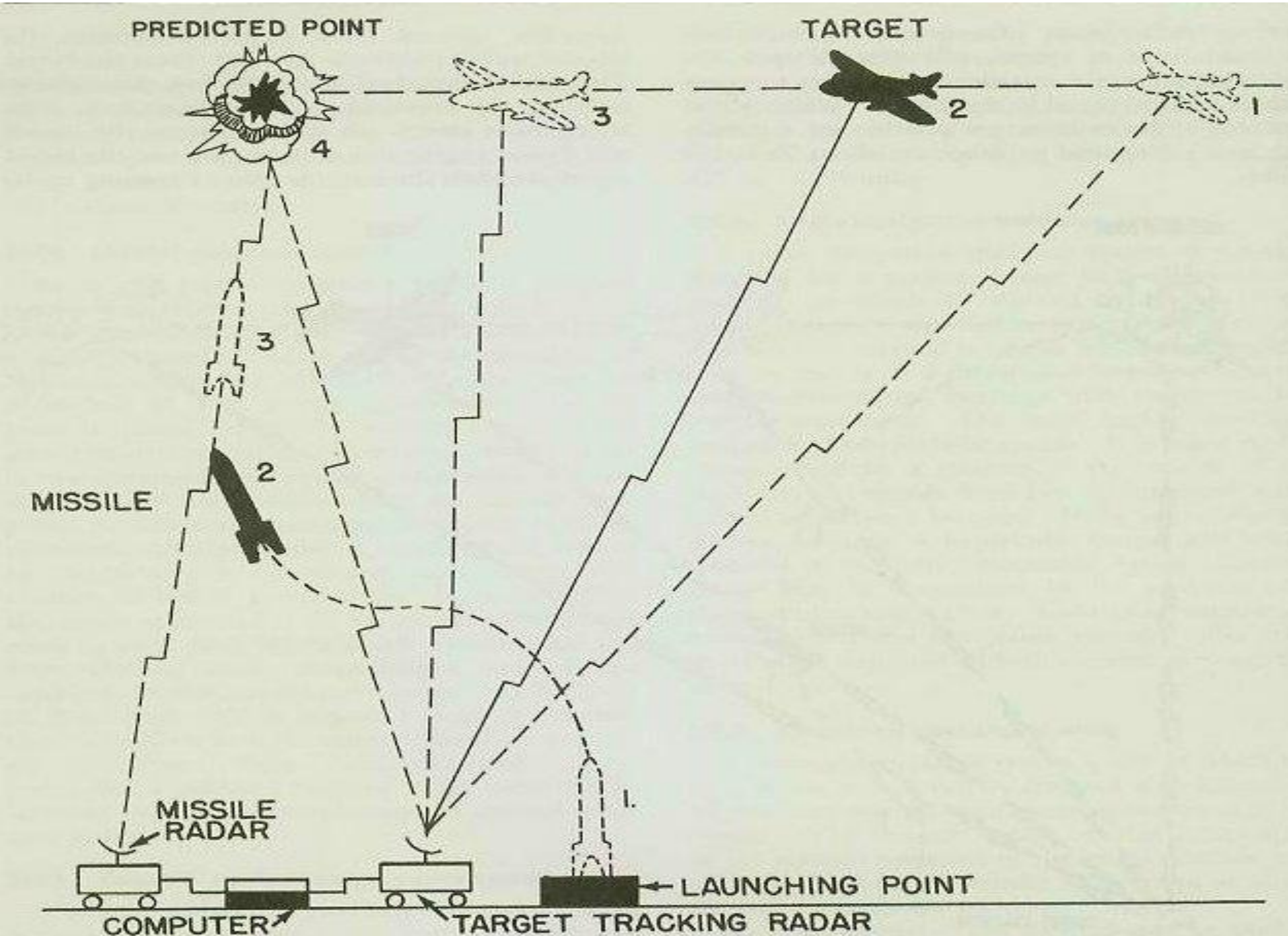


FIGURE 29C2.—Beam rider—two radars.

Terms and definitions

Release time (or ready time): This is the time instant at which a task(process) is ready or **eligible** for execution

Schedule Time: This is the time instant when a task gets its **chance** to execute

Completion time: This is the time instant when task completes its execution

Deadline: This is the instant of time by which the execution of task should be completed

Runtime: The time taken without interruption to complete the task, after the task is released

Terms and definitions

Tardiness: Specifies the amount of time by which a task **misses its deadline**. It’s is equal to the difference between completion time and deadline

Laxity: Is defined as **deadline** minus remaining **computation time**. The laxity of task is the maximum amount of time it can wait and still meets its deadline

Hard and Soft Real Time Systems

Hard Real Time System

- Failure to meet deadlines is fatal
- example : Flight Control System, Airbag Deployment

Soft Real Time System

- Late completion of jobs is undesirable but not fatal.
- System performance degrades as more & more jobs miss deadlines
- Video streaming, Audio Processing

Role of an OS in Real Time Systems

Standalone Applications (only firmware)

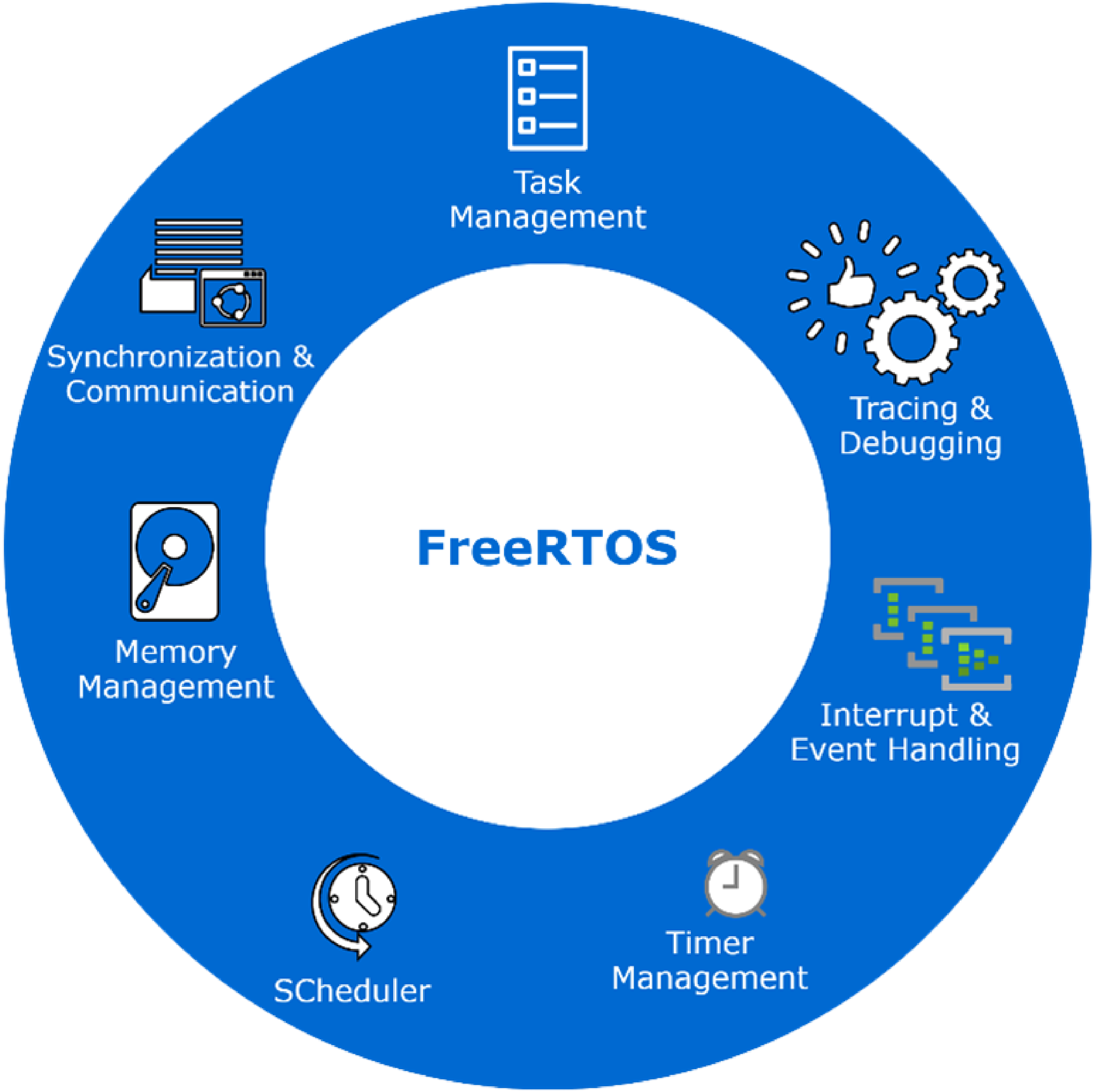
- Often no OS involved
- Micro controller based Embedded Systems

Some Real Time Applications are huge & complex

- Multiple threads
- Complicated Synchronization Requirements
- Filesystem / Network / Windowing support
- OS primitives reduce the software design time

Features of RTOS's

- Scheduling.
- Resource Allocation.
- Interrupt Handling.
- Other issues like kernel size.


FreeRTOS
Task
Management
Tracing &
Debugging
Interrupt &
Event Handling
Timer
Management
SCheduler
Memory
Management
Synchronization &
Communication

Scheduling concerns

Scheduling in Real time in operating systems:

- No of tasks
- Resource Requirements
- Release Time
- Execution time
- Deadlines

Assumptions about Scheduling

CPU scheduling big area of research in early ‘70s

Many implicit assumptions for CPU scheduling:

- One program per user
- One thread per program
- Programs are independent

These are unrealistic but simplify the problem

Does “fair” mean fairness among users or programs?

- If I run one compilation job and you run five, do you get five times as much CPU?
  - Often times, yes!

Goal: dole out CPU time to optimize some desired parameters of the system.

- What parameters?

What is Important in a Scheduling Algorithm?

What is Important in a Scheduling Algorithm?

Minimize Response Time

- Elapsed time to do an operation (job)
- Response time is what the user sees
  - Time to echo keystroke in editor
  - Time to compile a program
  - Real-time Tasks: Must meet deadlines imposed by World

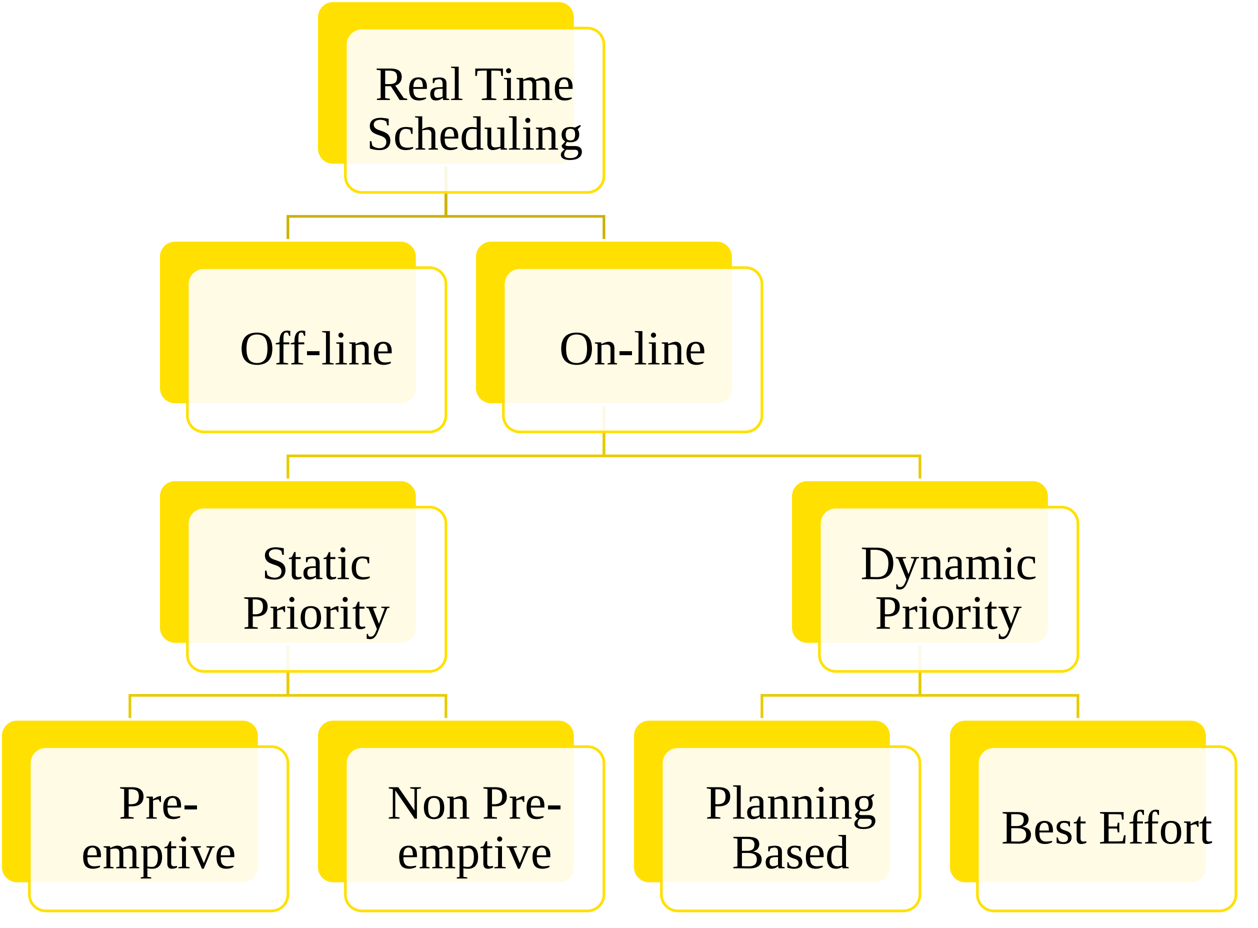
Maximize Throughput

- Jobs per second
- Throughput related to response time, but not identical
  - Minimizing response time will lead to more context switching than if you maximized only throughput
- Minimize overhead (context switch time) as well as efficient use of resources (CPU, disk, memory, etc.)

Fairness

- Share CPU among users in some equitable way
- Not just minimizing average response time

RTOS Scheduling Classification

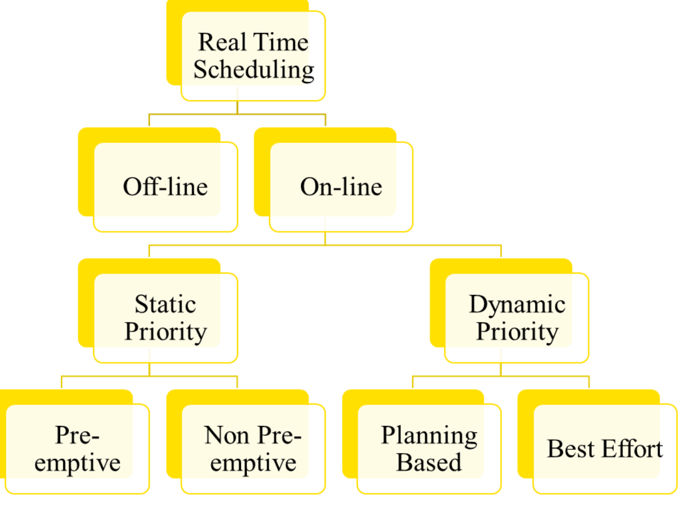


Off-Line Scheduling (Pre-runtime scheduling)

Generate scheduling information prior to system execution (Deterministic System Model)

This scheduling is based on:

- Release time
- Deadlines
- Execution



Disadvantage: Inflexibility, if any parameter changes, the policy will have to be recomputed

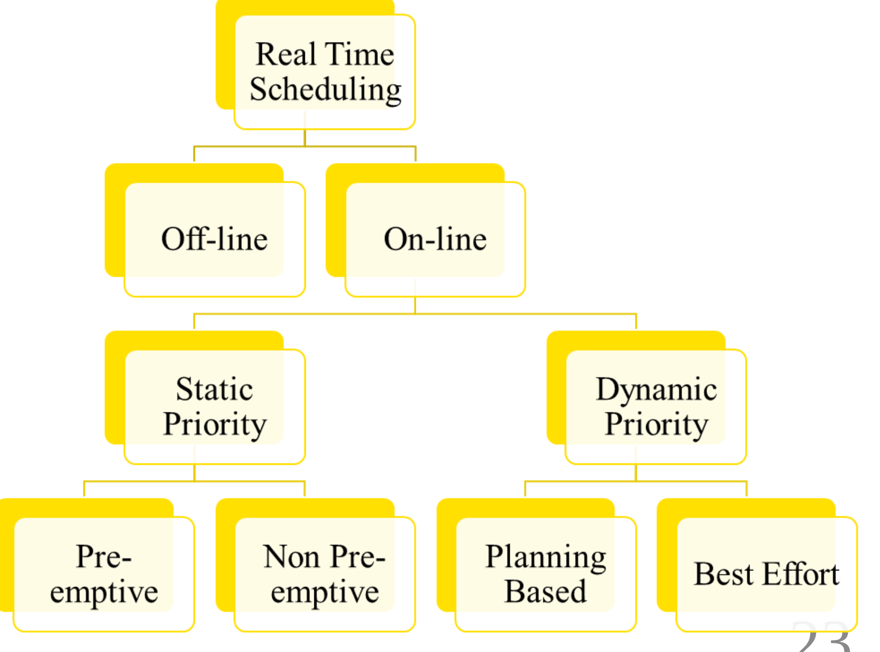
On-Line Scheduling

- Number and types of tasks, associated parameters are not known in advance.
- So, Scheduling must accommodate dynamic changes
- Online Scheduling

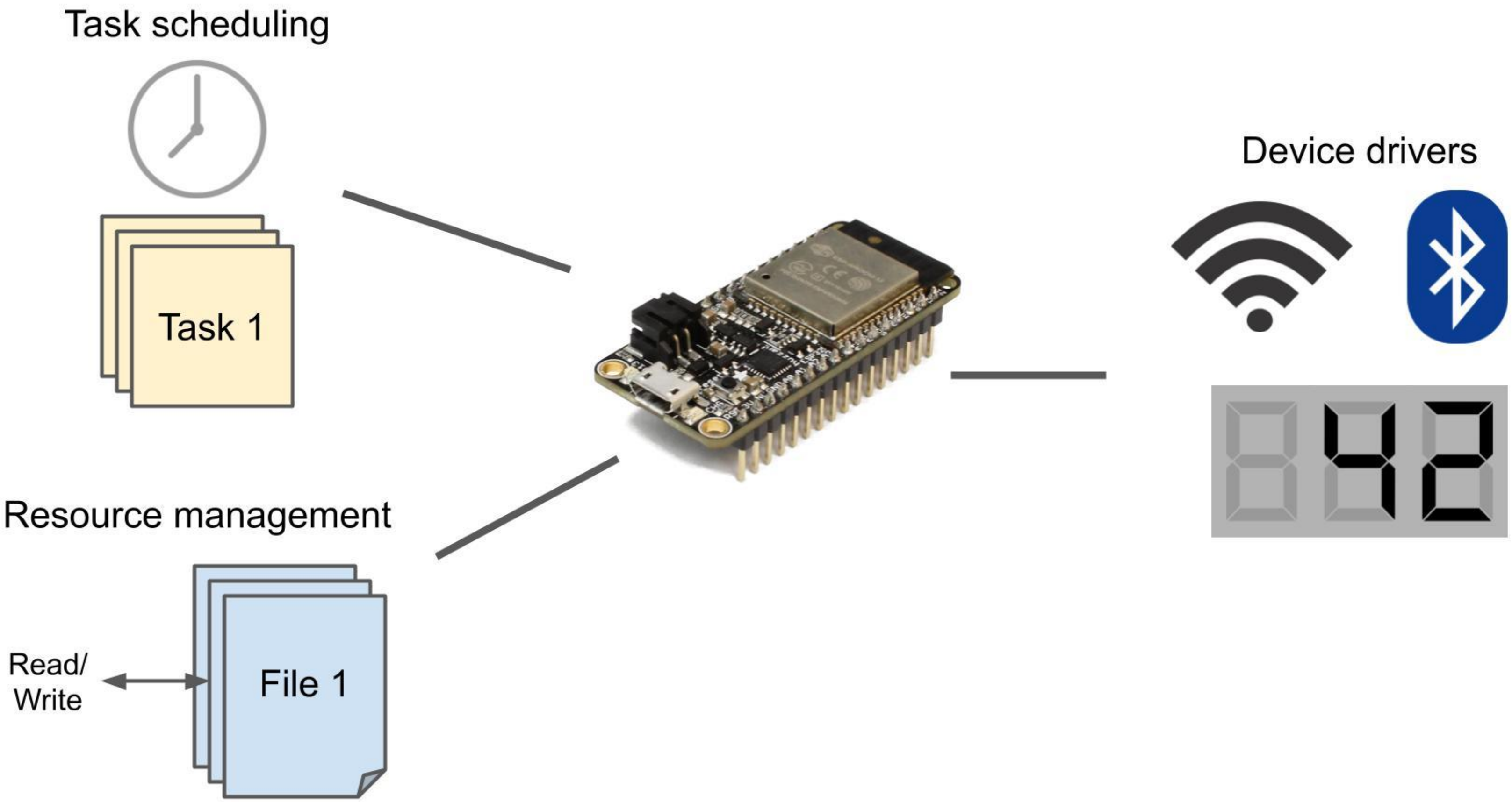
  are of two types:

  – Static Priority

  – Dynamic Priority

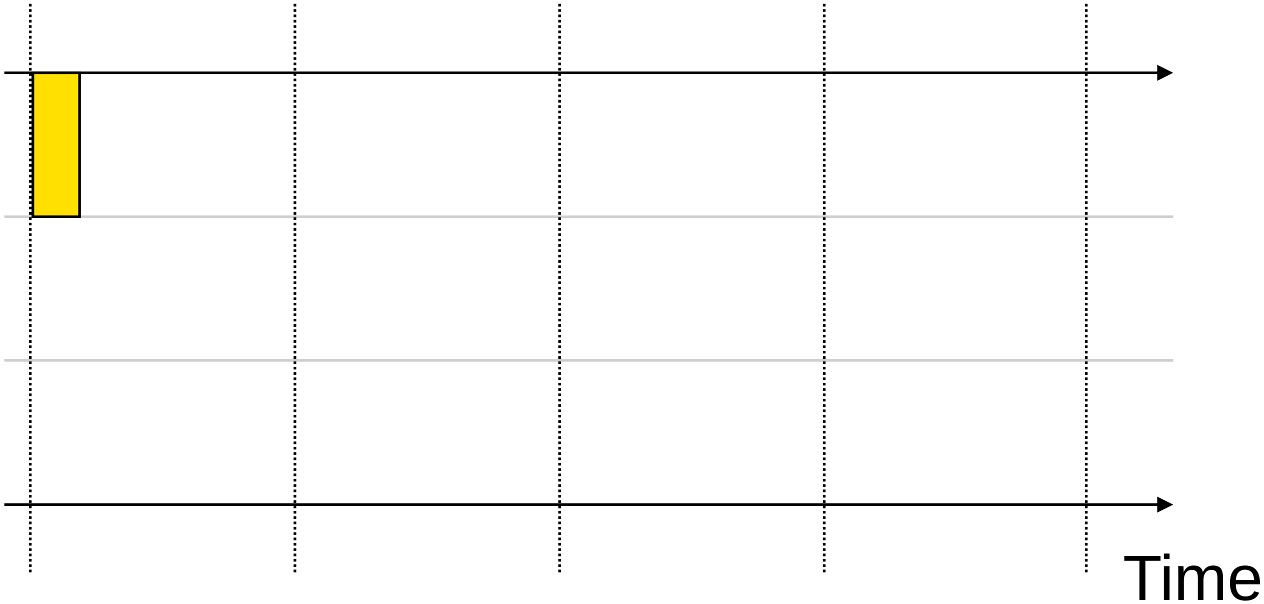


Real-Time Operating System (RTOS)



Preemptive Fixed Priority Scheduling

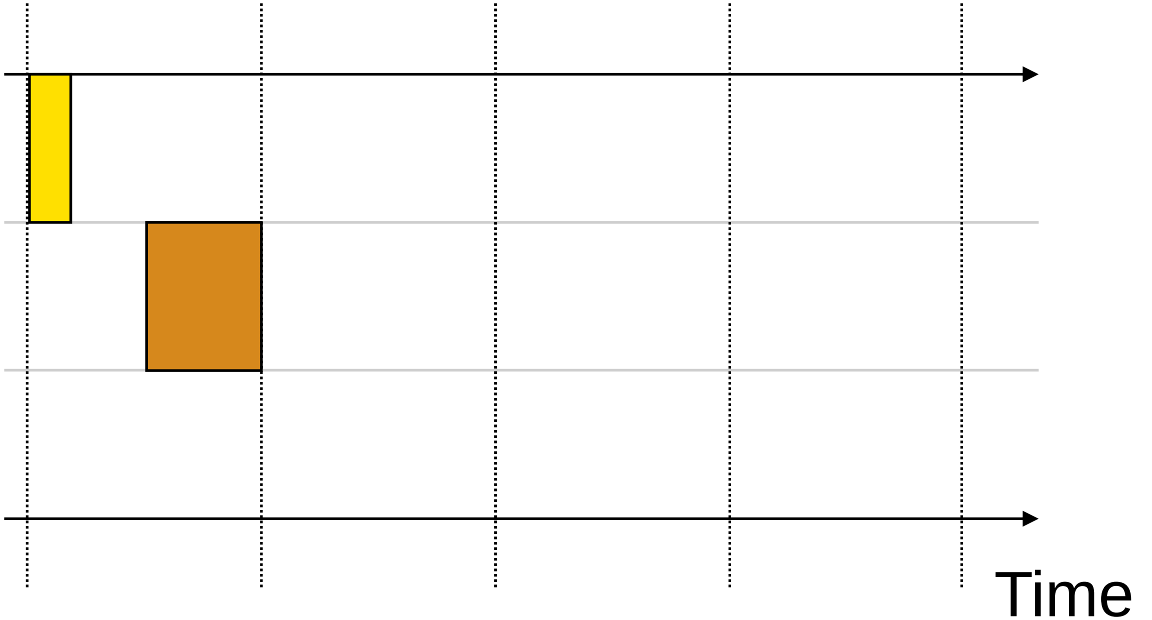
High Priority Task



Low Priority Task

Preemptive Fixed Priority Scheduling

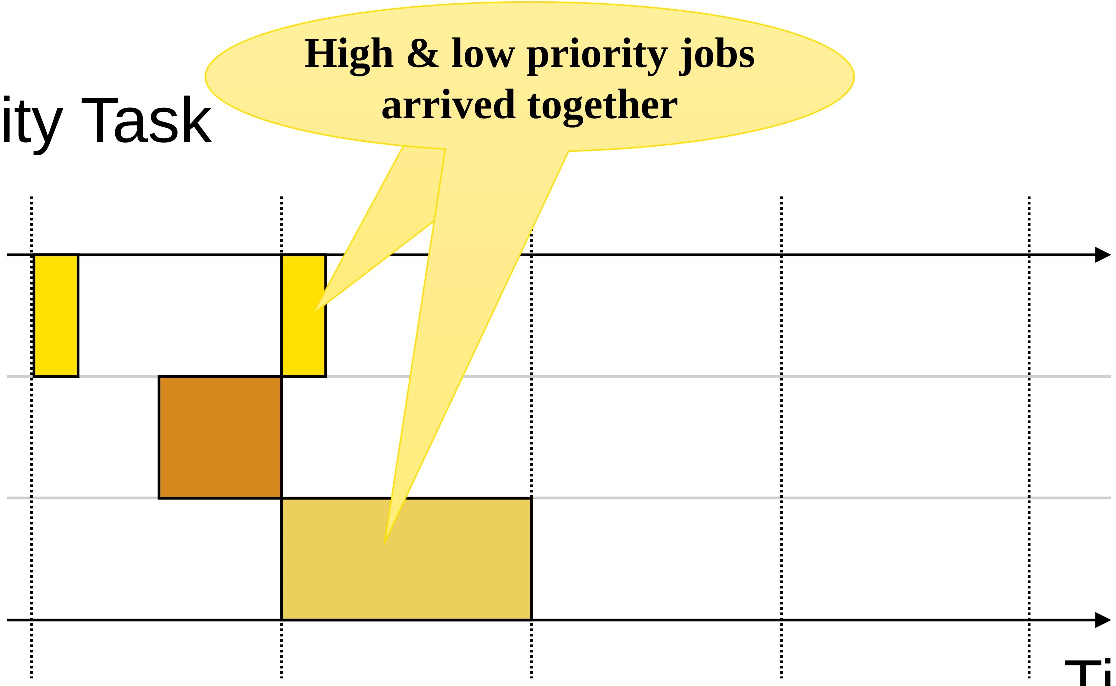
High Priority Task



Low Priority Task

Preemptive Fixed Priority Scheduling

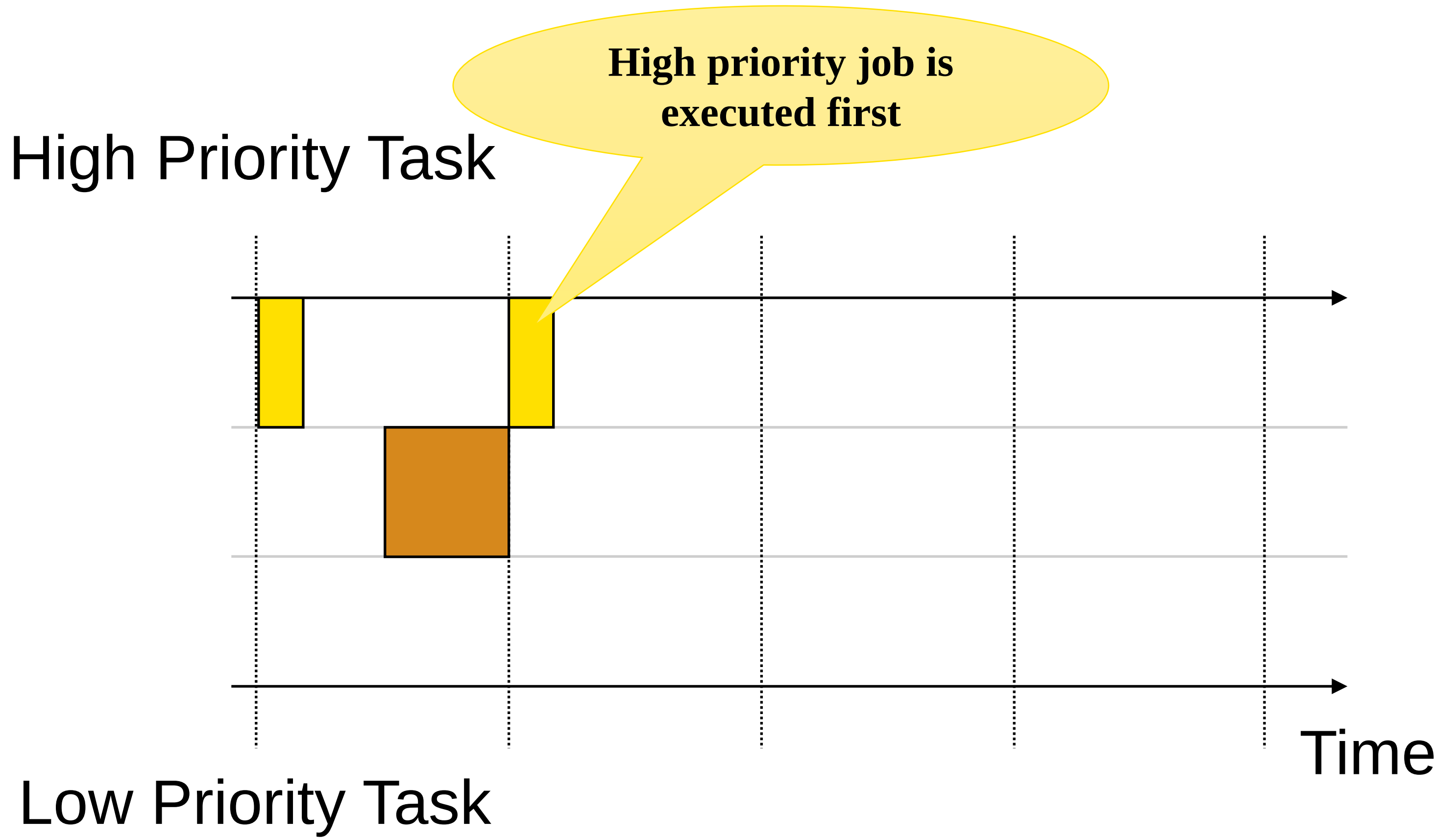
High Priority Task



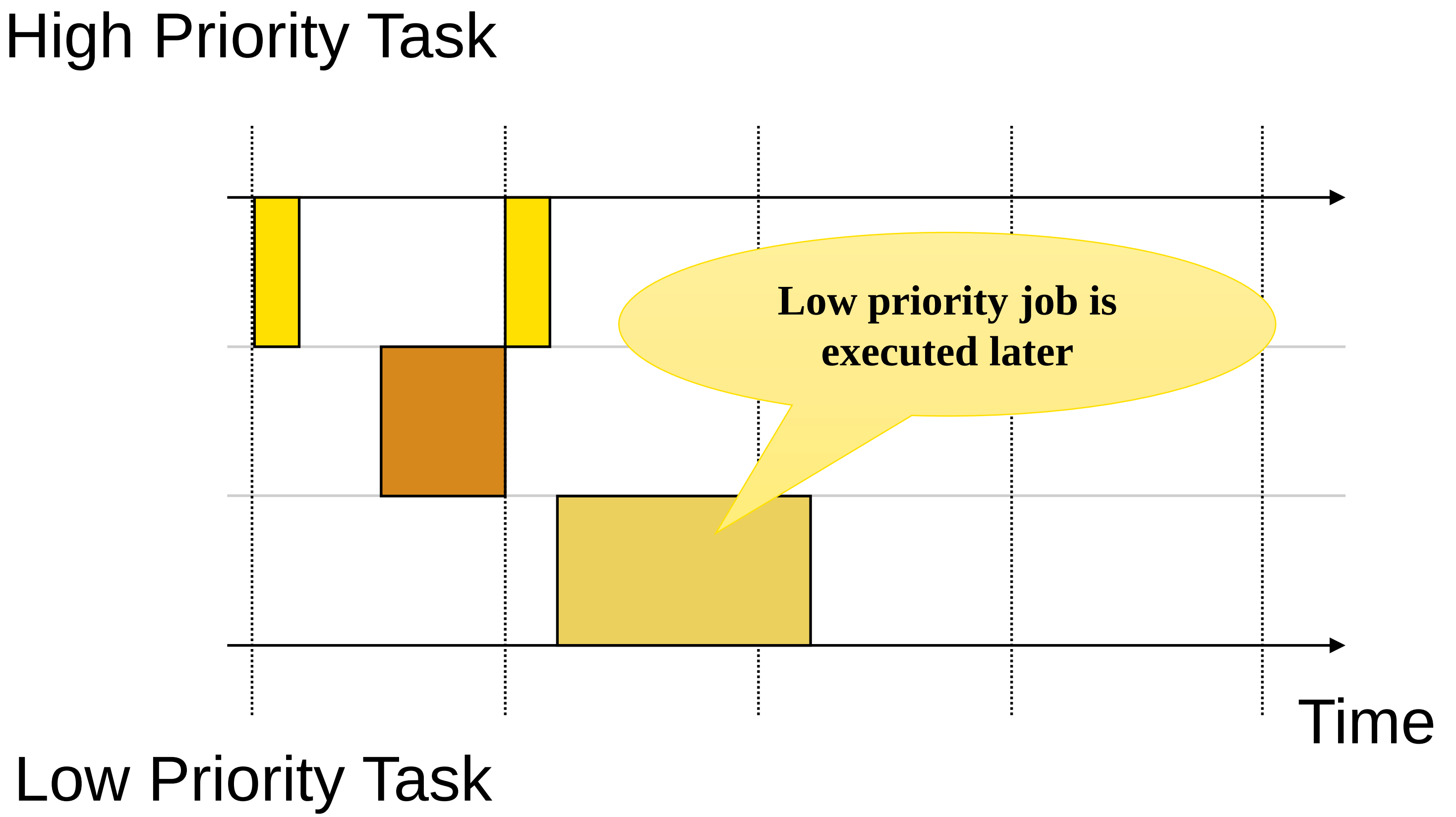
Time

Low Priority Task

Preemptive Fixed Priority Scheduling



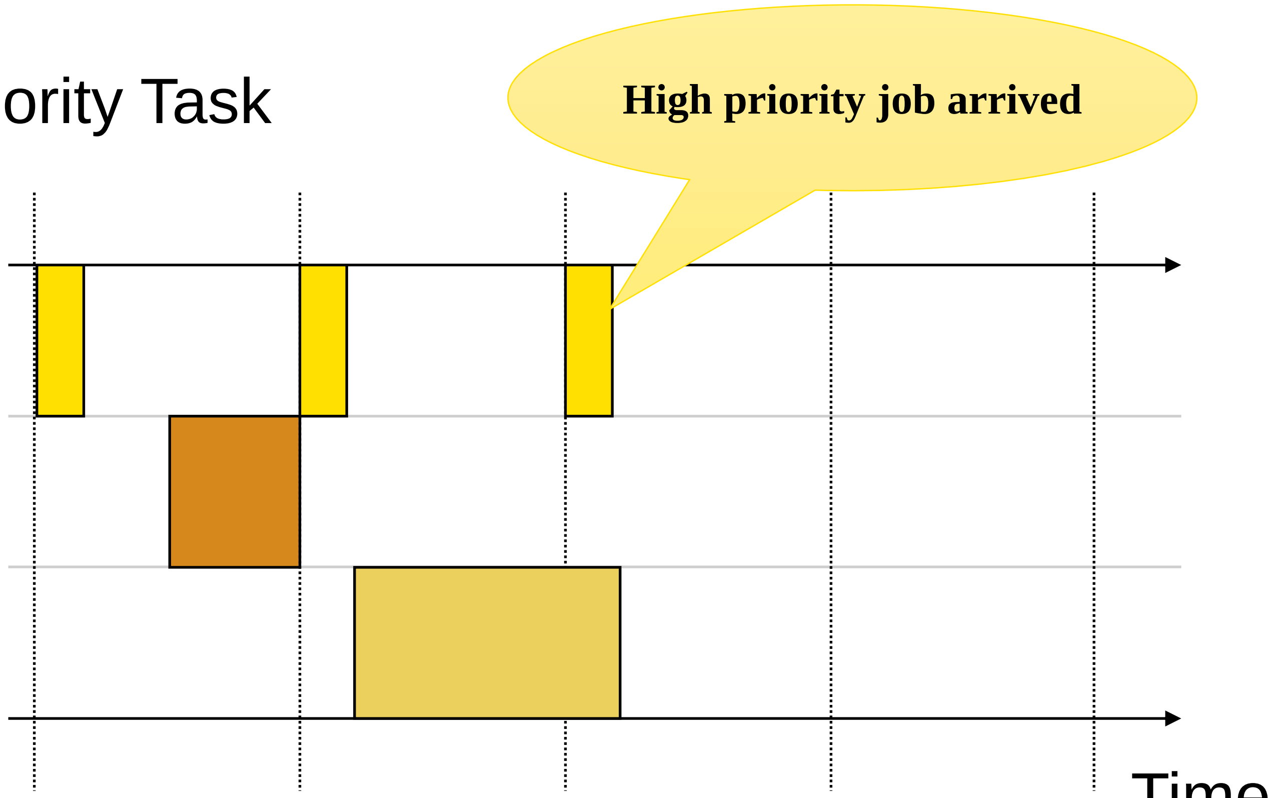
Preemptive Fixed Priority Scheduling



Linnæus University

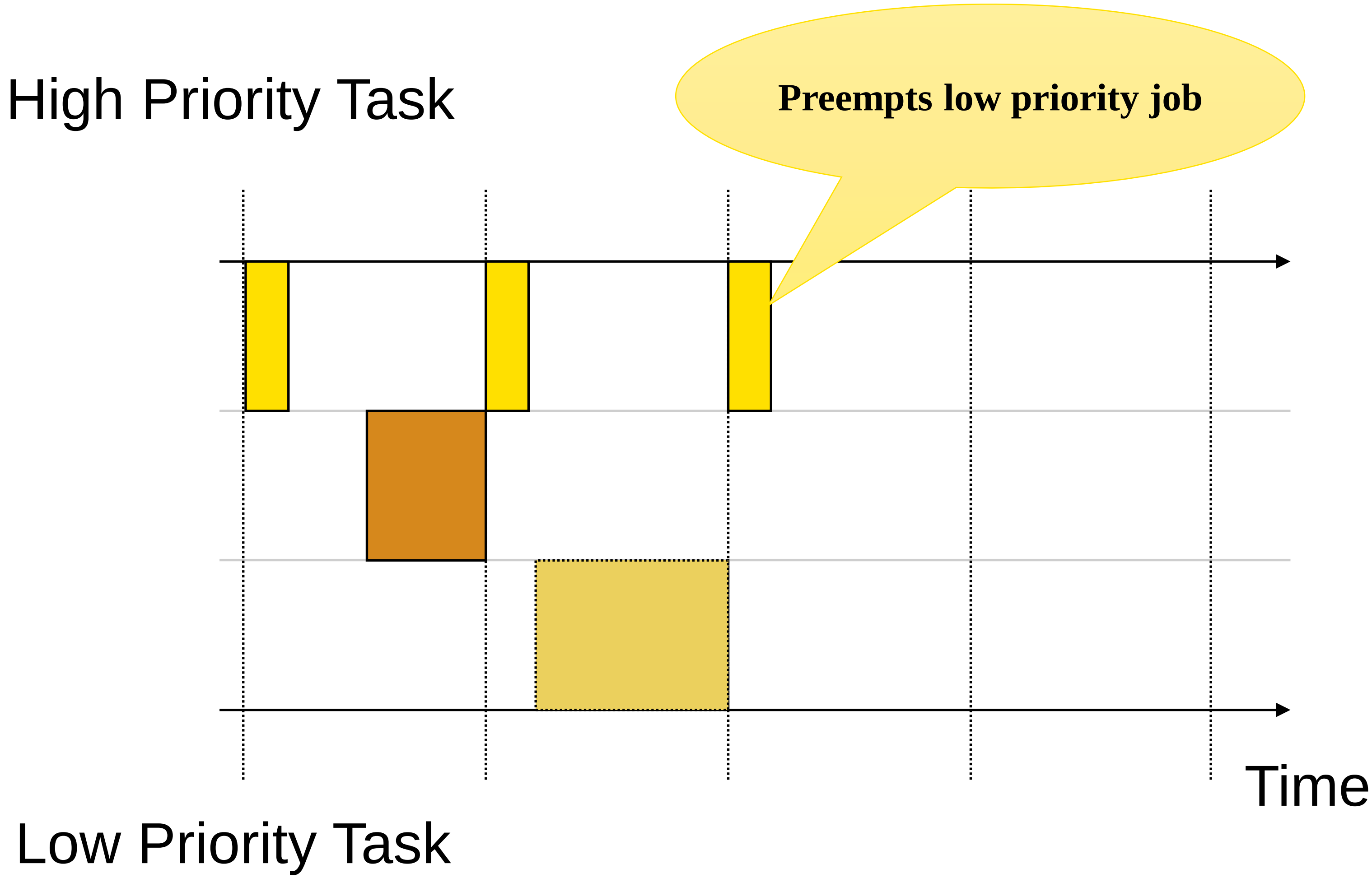
Preemptive Fixed Priority Scheduling

High Priority Task



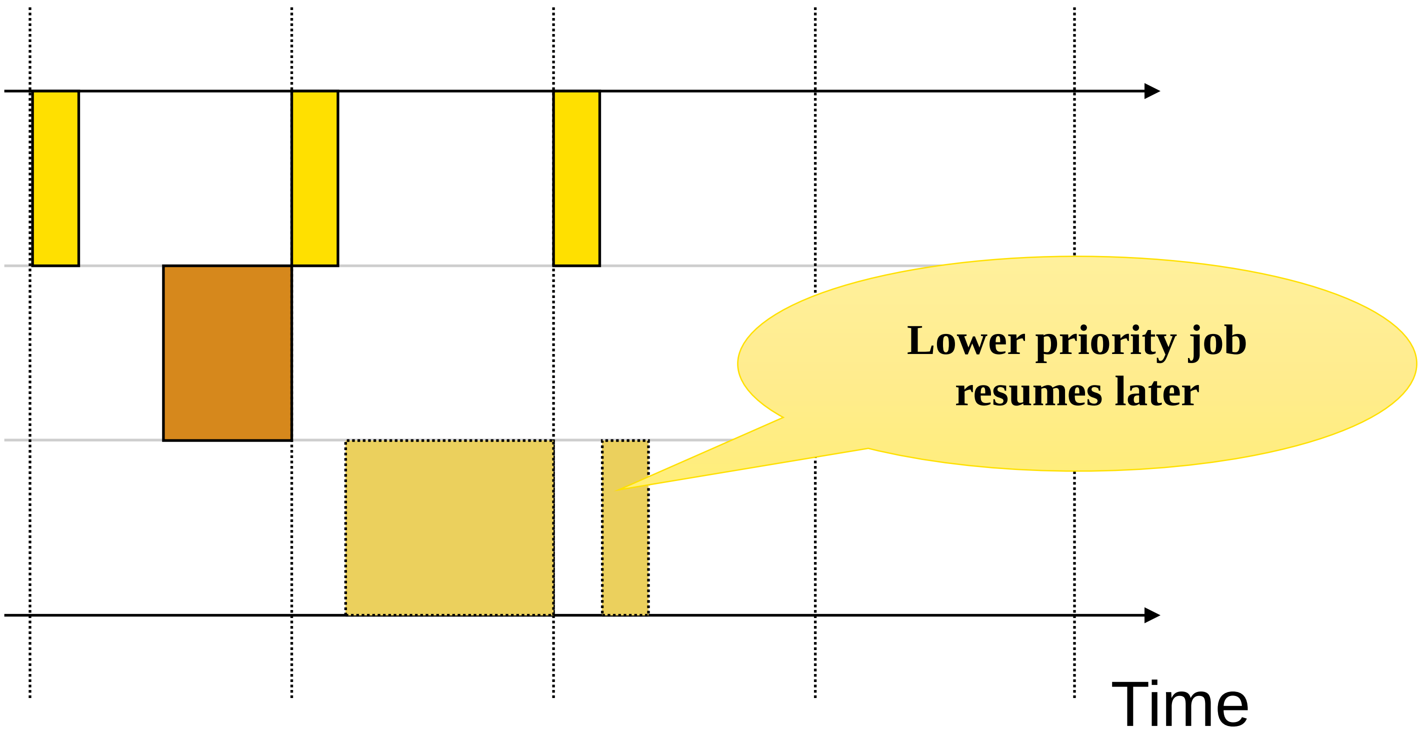
Low Priority Task

Preemptive Fixed Priority Scheduling



Preemptive Fixed Priority Scheduling

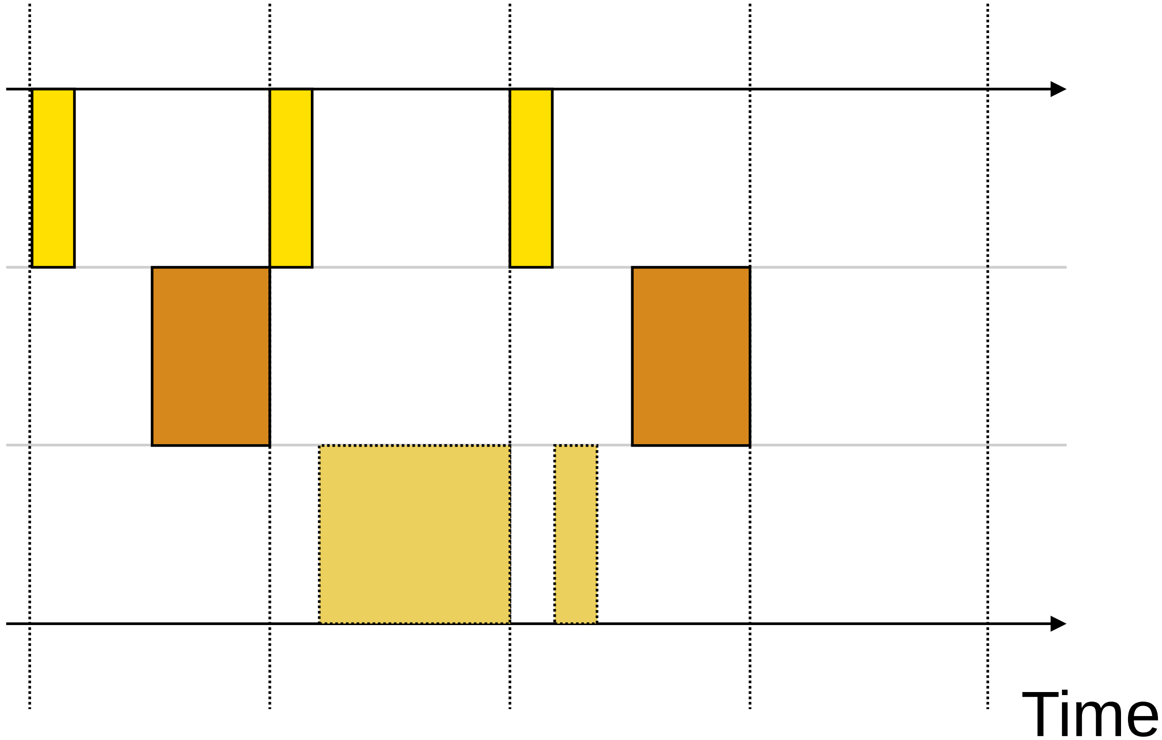
High Priority Task



Low Priority Task

Preemptive Fixed Priority Scheduling

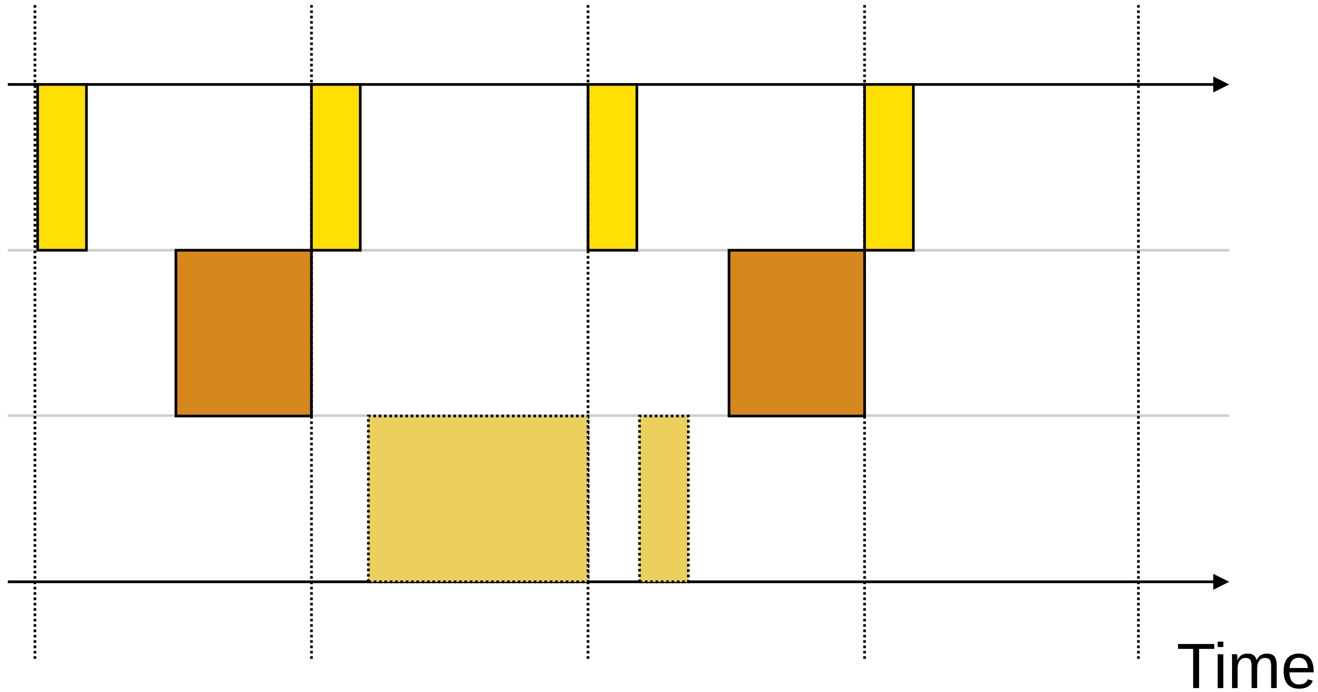
High Priority Task



Low Priority Task

Preemptive Fixed Priority Scheduling

High Priority Task



Low Priority Task

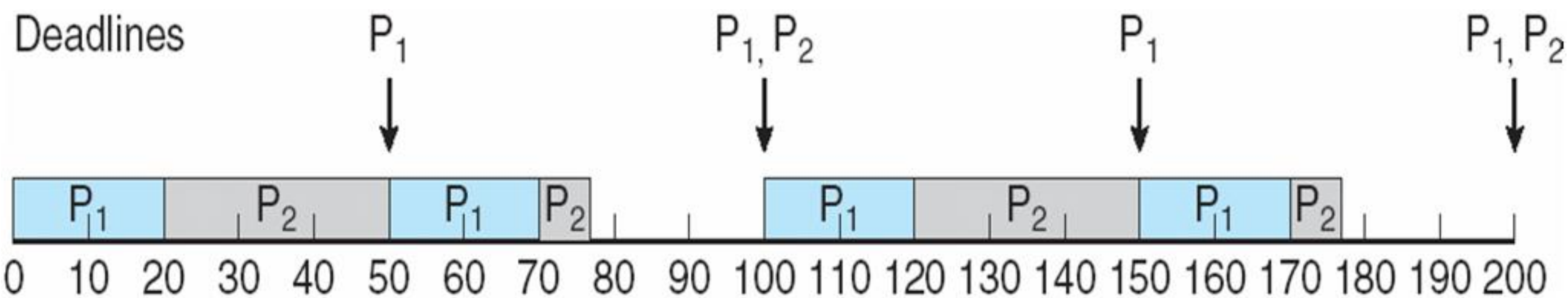
Rate Montonic Scheduling

A priority is assigned based on the inverse of its period

Shorter periods = higher priority

Longer periods = lower priority

P_1 is assigned a higher priority than P_2.



Rate Montonic Scheduling

Example- scenario with three tasks (T1, T2, T3) with the following properties:
T1 has the highest priority and a period of 4ms.
T2 has a medium priority and a period of 5ms.
T3 has the lowest priority and a period of 10ms.
All tasks are assumed to arrive at time 0 and can be preempted.

Step-by-Step Execution
At time 0ms: All three tasks are ready. The scheduler checks their priorities:
T1 has the highest priority, so it starts executing.
At time 1ms: A higher-priority task (T1) is executing. The system continues running T1.
At time 2ms: T1 finishes its execution. Now the scheduler selects the next highest priority task, which is T2.
At time 4ms: T1 becomes ready again since its period is 4ms. Since T1 has a higher priority than T2, it preempts T2.
At time 6ms: T1 finishes its execution. The scheduler resumes T2.
At time 8ms: T1 becomes ready again and preempts T2.
At time 10ms: T3 is ready, but T1 is already executing. Since T3 has the lowest priority, it will have to wait until T1 and T2 complete.
This process continues with T1 preempting lower-priority tasks whenever it becomes ready

Non-Preemptive Fixed/static Priority Scheduling

Example of Non-Preemptive Scheduling

Consider three tasks (T1, T2, and T3) with the following priorities:

- T1: Highest priority
- T2: Medium priority
- T3: Lowest priority

Assuming the tasks are ready to execute as follows:

- All tasks arrive at time 0.
- T1 takes 2ms to complete.
- T2 takes 4ms to complete.
- T3 takes 3ms to complete.

Execution

Step-by-StepTime 0ms: All tasks are ready to execute. Since the system is non-preemptive, the CPU selects the task with the highest priority that is not currently running.

T1 (highest priority) starts executing.

Time 2ms: T1 completes its execution. The scheduler checks for the next highest-priority task that is ready.

T2 (medium priority) starts executing.

Time 6ms: T2 completes its execution. The scheduler checks for the next highest-priority task that is ready.

T3 (lowest priority) starts executing.

Time 9ms: T3 completes its execution.

In this scenario, if a new high-priority task T1+ arrived at time 1ms while T1 was running, it would have to wait until T1 completed, as non-preemptive scheduling does not allow interruptions.

Dynamic Priority Scheduling

Priority of the tasks can be changed depends on the deadlines, system load, state of the task.

Dynamic Priority Scheduling: Planning-Based Dynamic Priority Scheduling

a plan to decide the order in which tasks are executed

Planning-Based Scheduling in an Embedded System (Earliest Deadline First)

Suppose an embedded system has three tasks (T1, T2, T3):

T1 has a flexible deadline of 10ms.

T2 has a strict deadline of 5ms.

T3 has a processing time of 2ms but a soft deadline.

Step-by-Step Execution

1.Initial Planning:

1. When all tasks are initially loaded, the system creates a plan based on their deadlines. For example, it may plan to execute T2 first (strict deadline), then T3 (short processing time), and finally T1.

2.Dynamic Adjustment:

1. While T2 is running, a new task T4 arrives with a critical deadline of 3ms.
2. The system replans: since T4 has a stricter deadline than the remaining tasks, it interrupts the current task (if preemption is allowed) or schedules it to run next.

3.Replanning:

1. As tasks complete or new tasks arrive, the system continues to re-evaluate priorities and adjust the plan to optimize execution.

Dynamic Priority Scheduling: Best Effort

Best-Effort Scheduling is a strategy in which a system attempts to complete as many tasks as possible in the available resources and time but does not guarantee that all tasks will meet their deadlines.

First-Come, First-Served (FCFS)

"Run until Done:" FIFO algorithm

In the beginning, this meant one program runs non-preemtively until it is finished (including any blocking for I/O operations)

Now, FCFS means that a process keeps the CPU until one or more threads block

Example: Three processes arrive in order P1, P2, P3.

- P1 burst time: 24
- P2 burst time: 3
- P3 burst time: 3

Draw the Gantt Chart and compute Average Waiting Time and Average Completion Time.

Scheduling Algorithms: First-Come, First-Served (FCFS)

Example: Three processes arrive in order P1, P2, P3.

- P1 burst time: 24
- P2 burst time: 3
- P3 burst time: 3

Waiting Time

- P1: 0
- P2: 24
- P3: 27

Completion Time:

- P1: 24
- P2: 27
- P3: 30

Average Waiting Time: (0+24+27)/3 = 17

Average Completion Time: (24+27+30)/3 = 27

Scheduling Algorithms: First-Come, First-Served (FCFS)

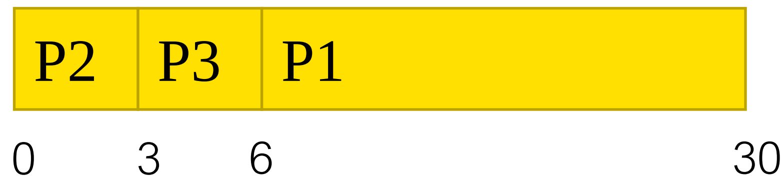
What if their order had been P2, P3, P1?

- P1 burst time: 24
- P2 burst time: 3
- P3 burst time: 3

Scheduling Algorithms: First-Come, First-Served (FCFS)

What if their order had been P2, P3, P1?

- P1 burst time: 24
- P2 burst time: 3
- P3 burst time: 3



Waiting Time

- P1: 0
- P2: 3
- P3: 6

Completion Time:

- P1: 3
- P2: 6
- P3: 30

Average Waiting Time: (0+3+6)/3 = 3 (compared to 17)

Average Completion Time: (3+6+30)/3 = 13 (compared to 27)

Scheduling Algorithms: First-Come, First-Served (FCFS)

Average Waiting Time: (0+3+6)/3 = 3 (compared to 17)

Average Completion Time: (3+6+30)/3 = 13 (compared to 27)

FIFO Pros and Cons:

- Simple (+)
- Short jobs get stuck behind long ones (-)
  - If all you're buying is milk, doesn't it always seem like you are stuck behind a cart full of many items
- Performance is highly dependent on the order in which jobs arrive (-)

How Can We Improve on This?

Round Robin (RR) Scheduling

FCFS Scheme: Potentially bad for short jobs!

- Depends on submit order
- If you are first in line at the supermarket with milk, you don't care who is behind you; on the other hand…

Round Robin Scheme

- Each process gets a small unit of CPU time (time quantum)
  - Usually 10-100 ms
- After quantum expires, the process is preempted and added to the end of the ready queue
- Suppose N processes in ready queue and time quantum is Q ms:
  - Each process gets 1/N of the CPU time
  - In chunks of at most Q ms
  - What is the maximum wait time for each process?

Round Robin (RR) Scheduling

FCFS Scheme: Potentially bad for short jobs!

- Depends on submit order
- If you are first in line at the supermarket with milk, you don't care who is behind you; on the other hand…

Round Robin Scheme

- Each process gets a small unit of CPU time (time quantum)
  - Usually 10-100 ms
- After quantum expires, the process is preempted and added to the end of the ready queue
- Suppose N processes in ready queue and time quantum is Q ms:
  - Each process gets 1/N of the CPU time
  - In chunks of at most Q ms
  - What is the maximum wait time for each process?
    - No process waits more than (n-1)q time units

Round Robin (RR) Scheduling

Round Robin Scheme

- Each process gets a small unit of CPU time (time quantum)
  - Usually 10-100 ms
- After quantum expires, the process is preempted and added to the end of the ready queue
- Suppose N processes in ready queue and time quantum is Q ms:
  - Each process gets 1/N of the CPU time
  - In chunks of at most Q ms
  - What is the maximum wait time for each process?
    - No process waits more than (n-1)q time units

Performance Depends on Size of Q

- Small Q => interleaved
- Large Q is like…
- Q must be large with respect to context switch time, otherwise overhead is too high (spending most of your time context switching!)

Round Robin (RR) Scheduling

Round Robin Scheme

- Each process gets a small unit of CPU time (time quantum)
  - Usually 10-100 ms
- After quantum expires, the process is preempted and added to the end of the ready queue
- Suppose N processes in ready queue and time quantum is Q ms:
  - Each process gets 1/N of the CPU time
  - In chunks of at most Q ms
  - What is the maximum wait time for each process?
    - No process waits more than (n-1)q time units

Performance Depends on Size of Q

- Small Q => interleaved
- Large Q is like FCFS
- Q must be large with respect to context switch time, otherwise overhead is too high (spending most of your time context switching!)

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

The Gantt chart is:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1	
0	4	7	10	14	18	22	26	30

Waiting Time:

- P1: (10-4) = 6
- P2: (4-0) = 4
- P3: (7-0) = 7

Completion Time:

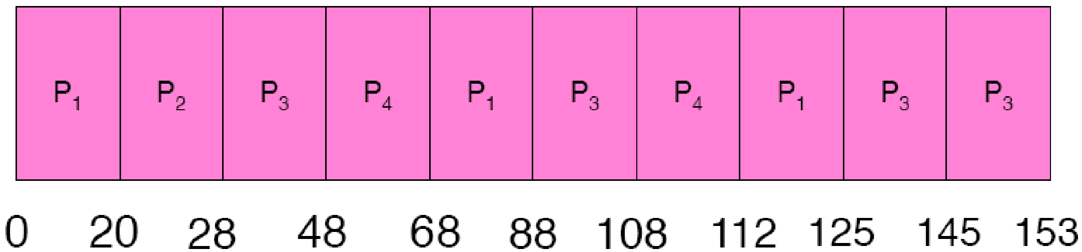
- P1: 30
- P2: 7
- P3: 10

Average Waiting Time: (6 + 4 + 7)/3= 5.67

Average Completion Time: (30+7+10)/3=15.67

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24



A process can finish before the time quantum expires, and release the CPU.

Waiting Time:

- P1: (68-20)+(112-88) = 72
- P2: (20-0) = 20
- P3: (28-0)+(88-48)+(125-108) = 85
- P4: (48-0)+(108-68) = 88

Completion Time:

- P1: 125
- P2: 28
- P3: 153
- P4: 112

Average Waiting Time: (72+20+85+88)/4 = 66.25

Average Completion Time: (125+28+153+112)/4 = 104.5

References

Brad Campbell – bradjc@virginia.edu https://www.cs.virginia.edu/~bjc8c/class/cs6456-f19/
[
Base slide]Pao-Ann Hsiung, 熊博安 Graduate Institute of Computer Science and Information Engineering, National Chung Cheng University, Chiayi 621, Taiwan, R.O.C.

[Boehm73] Boehm, B.W. "Software and its Impact: A Quantitative Assessment," *Datamation*, May 1973, p. 48-59.

[Buchenrieder93] Buchenrieder, K., "Codesign and Concurrent Engineering", Hot Topics, *IEEE Computer*, R. D. Williams, ed., January, 1993, pp. 85-86

[Buck94] Buck, J., et al., "Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems," *International Journal of Computer Simulation*, Vol. 4, April 1994, pp. 155-182.

[Chiodo92] Chiod0, M., A. Sangiovanni-Vincentelli, "Design Methods for Reactive Real-time Systems Codesign," *International Workshop on Hardware/Software Codesign*, Estes Park, Colorado, September 1992.

[Chiodo94] Chiodo, M., P. Giusto, A. Jurecska, M. Marelli, H. C. Hsieh, A. Sangiovanni-Vincentelli, L. Lavagno, "Hardware-Software Codesign of Embedded Systems," *IEEE Micro*, August, 1994, pp. 26-36; © IEEE 1994.

[Chou95] P. Chou, R. Ortega, G. Borriello, "The Chinook hardware/software Co-design System," *Proceedings ISSS*, Cannes, France, 1995, pp. 22-27.

[DeMicheli93] De Micheli, G., "*Extending CAD Tools and Techniques*", Hot Topics, *IEEE Computer*, R. D. Williams, ed., January, 1993, pp. 84

[DeMicheli94] De Micheli, G., "Computer-Aided Hardware-Software Codesign", *IEEE Micro*, August, 1994, pp. 10-16

[DeMichelli97] De Micheli, G., R. K. Gupta, "Hardware/Software Co-Design," *Proceedings of the IEEE*, Vol. 85, No. 3, March 1997, pp. 349-365.

[Ernst93] Ernst, R., J. Henkel, T. Benner, "*Hardware-Software Cosynthesis for Micro-controllers*", *IEEE Design and Test*, December, 1993, pp. 64-75

[Franke91] Franke, D.W., M.K. Purvis. "Hardware/Software Codesign: A Perspective," *Proceedings of the 13th International Conference on Software Engineering*, May 13-16, 1991, p. 344-352; © IEEE 1991

References (Cont.)

Copyright ã 1995-1999 SCRA, Hardware/Software Codesign Overview RASSP Education & Facilitation Program Module 14

Proceedings of the International Workshops / Symposium on HW/SW Codesign, 1993 ~ 2001 (ACM Press & IEEE CS Press).

Hardware Software Co-design of Embedded Systems, F. Balarin, Chiodo, et al., Kluwer Academic Publishers, May 1997.

Co-synthesis of Hardware and Software for Embedded Systems, R. Gupta, Kluwer Academic Publishers, 1995.

Special Issue of the Proceedings of the IEEE on Hardware Software Co-design edited by G. De Micheli, Vol. 85, No. 3, March 1997.

POLIS and Ptolemy tools introduction materials and manuals

[Gajski94] Gajski, D. D., F. Vahid, S. Narayan, J. Gong, *Specification and Design of Embedded Systems*, Prentice Hall, Englewood Cliffs, N J, 07632, 1994

[Gupta92] Gupta, R.K., C.N. Coelho, Jr., G.D. Micheli. "Synthesis and Simulation of Digital Systems Containing Interactive Hardware and Software Components," *29th Design Automation Conference*, June 1992, p.225-230.

[Gupta93] Gupta, R.K., G. DeMicheli, "Hardware-Software Cosynthesis for Digital Systems," *IEEE Design and Test*, September 1993, p.29-40; © IEEE 1993.

[Hermann94] Hermann, D., J. Henkel, R. Ernst, "An approach to the estimation of adapted Cost Parameters in the COSYMA System*", 3rd International Conference on Hardware/Software codesign*, Grenoble, France, September 22-24, 1994, pp. 100-107

[Hood94] Hood, W., C. Myers, "RASSP: Viewpoint from a Prime Developer," *Proceedings 1st Annual RASSP Conference*, Aug. 1994.

[IEEE] All referenced IEEE material is used with permission.

[Ismail95] T. Ismail, A. Jerraya, "Synthesis Steps and Design Models for Codesign," *IEEE Computer*, no. 2, pp. 44-52, Feb 1995.

[Kalavade93] A. Kalavade, E. Lee, "A Hardware-Software Co-design Methodology for DSP Applications," *IEEE Design and Test*, vol. 10, no. 3, pp. 16-28, Sept. 1993.

References (Cont.)

[Parker84] Parker, A.C., “Automated Synthesis of Digital Systems,” *IEEE Design and Test,*, November 1984, p. 75-81.

[RASSP94] *Proceedings of the 1st RASSP Conference*, Aug. 15-18, 1994.

[Rozenblit94] Rozenblit, J. and K. Buchenrieder (editors). Codesign Computer -Aided Software/Hardware Engineering, IEEE Press, Piscataway, NJ, 1994; © IEEE 1994.

[Smith86] Smith, C.U., R.R. Gross. “Technology Transfer between VLSI Design and Software Engineering: CAD Tools and Design Methodologies,” *Proceedings of the IEEE*, Vol. 74, No. 6, June 1986, p.875-885.

[Srivastava91] M. B. Srivastava, R. W. Broderson, “Rapid prototyping of Hardware and Software in a Unified Framework,” *Proceedings ICCAD*, 1991, pp. 152-155.

[Subrahmanyam93] Subrahmanyam, P. A., “Hardware-Software Codesign -- Cautious optimism for the future”, Hot Topics, *IEEE Computer*, R. D. Williams, ed., January, 1993, pp. 84

[Tanenbaum87] Tanenbaum, A.S., *Operating Systems: Design and Implementation*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1987.

[Terry90] Terry, C. “Concurrent Hardware and Software Design Benefits Embedded Systems,” *EDN*, July 1990, p. 148-154.

[Thimbleby88] Thimbleby, H. “Delaying Commitment,” *IEEE Software*, Vol. 5, No. 3, May 1988, p. 78-86.

[Thomas93] Thomas, D.E., J.K. Adams, H. Schmitt, “A Model and Methodology for Hardware-Software Codesign,” *IEEE Design and Test*, September 1993, p.6-15; © IEEE 1993.

[Turn78] Turn, R., “Hardware-Software Tradeoffs in Reliable Software Development,” *11th Annual Asilomar Conference on Circuits, Systems, and Computers,* 1978, p.282-288.

[Vahid94] Vahid, F., J. Gong, D. D. Gajski, “*A Binary Constraint Search Algorithm for Minimizing Hardware During Hardware/Software Partitioning”,* 3rd International Conference on Hardware/Software Codesign, Grenoble, France, Sepetember22-24, 1994, pp. 214-219

[Wolf94] Wolf, W.H. “Hardware-Software Codesign of Embedded Systems,” *Proceedings of the IEEE*, Vol. 82, No.7, July 1994, p.965-989.

References (Cont.)

Additional Reading:

Aylor, J.H. et al., "The Integration of Performance and Functional Modeling in VHDL" in *Performance and Fault Modeling with VHDL,* J. Schoen, ed., Prentice-Hall, Englewood Cliffs, N.J., 1992.

D'Ambrosio, J. G., X. Hu, "Configuration-level Hardware-Software Partitioning for Real-time Embedded Systems", *3rd International Conference on Hardware/Software codesign,* Grenoble, France, September 22-24, 1994, pp. 34-41

Eles, P., Z. Peng, A. Doboli, "VHDL System-Level Specification and Partitioning in a Hardware-Software Cosynthesis Environment", *3rd International Conference on Hardware/Software codesign,* Grenoble, France, September 22-24, 1994, pp. 49-55

Gupta, R.K., G. DeMicheli, "Hardware-Software Cosynthesis for Digital Systems," *IEEE Design and Test*, September 1993, p.29-40.

Richards, M., Gadient, A., Frank, G., eds. *Rapid Prototyping of Application Specific Signal Processors*, Kluwer Academic Publishers, Norwell, MA, 1997

Schultz, S.E., "An Overview of System Design," *ASIC and EDA*, January 1993, p.12-21.

Thomas, D. E, J. K. Adams, H. Schmit, "A Model and Methodology for Hardware-Software Codesign", *IEEE Design and Test*, September, 1993, pp. 6-15

Zurcher, F.W., B. Randell, "Iterative Multi-level Modeling - A Methodology for Computer System Design," *Proceedings IFIP Congress '68,* Edinburgh, Scotland, August 1968, p.867-871.

Lnu.se