MONASH University

Information Technology

FIT2100 Operating Systems

REAL-TIME PROCESS SCHEDULING

Week 6 - Supplementary
Semester 2 2024
(Reading: Tanenbaum, Chapter 2)

Dr Charith Jayasekara
Faculty of Information Technology

SCHEDULING IN REAL-TIME SYSTEM

SCHEDULING IN REAL-TIME SYSTEM

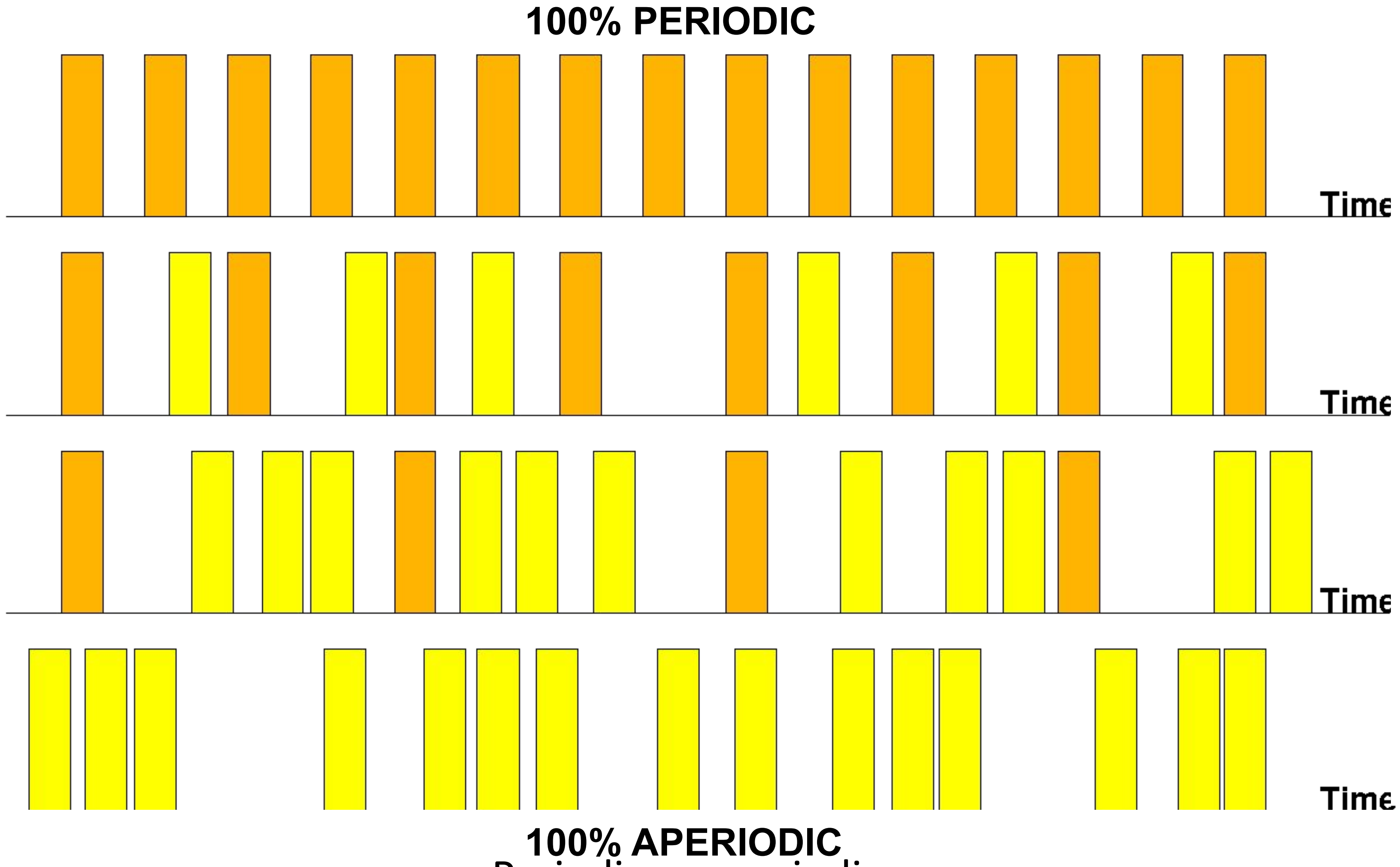
- Timeliness is of critical importance
- Categories
    - **Hard real time**
        - ❑ there are absolute deadlines that must be met
    - **Soft real time**
        - ❑ meaning that missing an occasional deadline is undesirable, but tolerable

    - In both categories, real-time behavior is achieved by dividing the program into a number of processes, the execution time of which is predictable and known in advance. These processes are generally short lived and can run to completion in well under a second. When an external event is detected, it is the job of the scheduler to schedule the processes in such a way that all deadlines are met.

SCHEDULING IN REAL-TIME SYSTEM

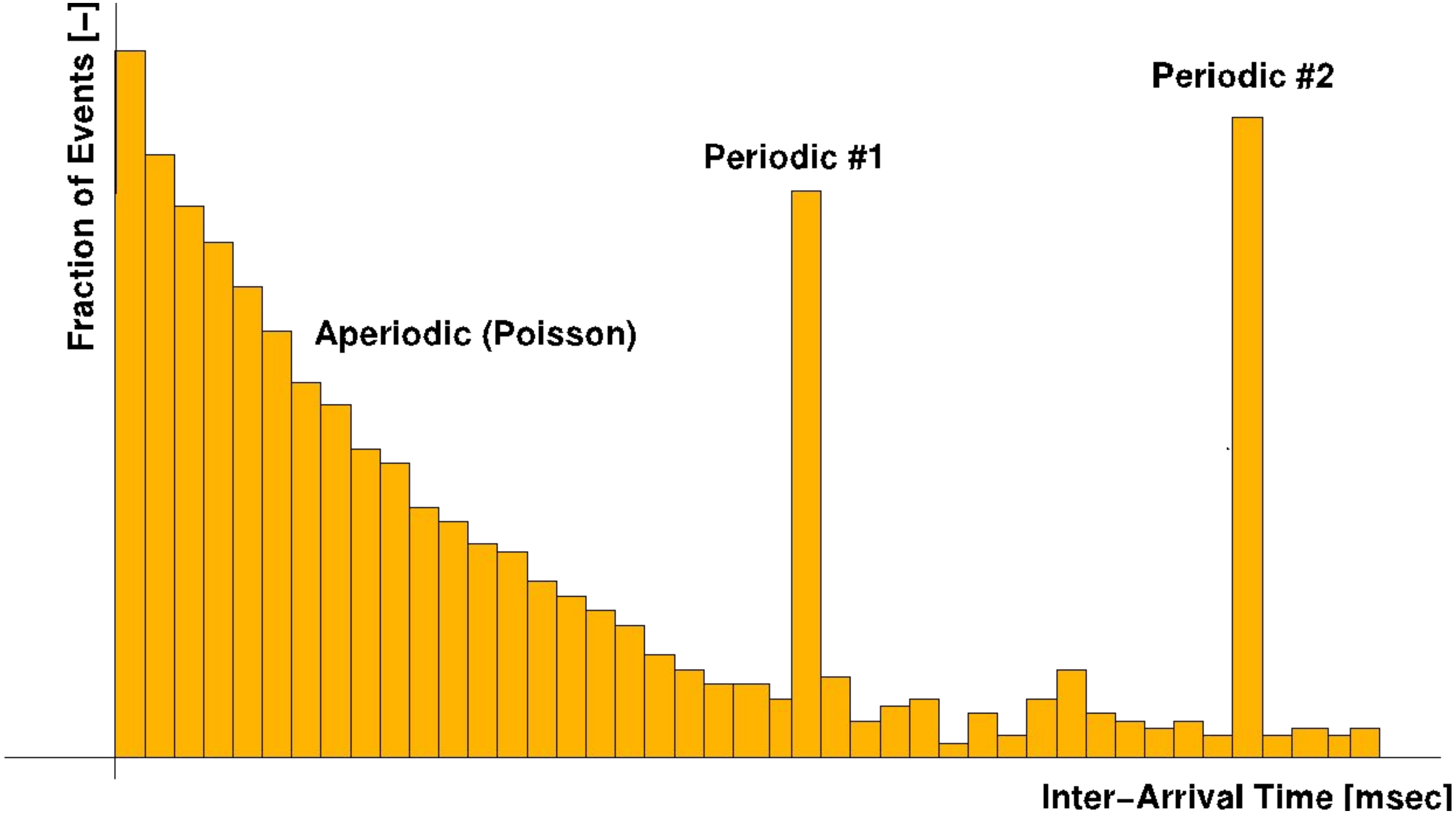
- Periodic or aperiodic
    - Periodic - meaning they occur at regular intervals
    - Aperiodic - meaning they occur unpredictably (e.g. randomly or chaotically)

SCHEDULING IN REAL-TIME SYSTEMS



Periodic or aperiodic

SCHEDULING IN REAL-TIME SYSTEMS



Periodic or aperiodic

SCHEDULING IN REAL-TIME SYSTEMS

- Schedulable satisfies
  - A system may have to respond to multiple periodic event streams. Depending on how much time each event requires for processing, handling all of them may not even be possible.

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

  - For example, if there are *m* periodic events and event *i* occurs with period P_i and requires C_i sec of CPU time to handle each event, then the load can be handled only if A real-time system that meets this criterion is said to be schedulable.
  - This means it can actually be implemented.
  - A process that fails to meet this test cannot be scheduled because the total amount of CPU time the processes want collectively is more than the CPU can deliver

SCHEDULING IN REAL-TIME SYSTEMS

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

- Schedulable satisfies
  - Example, consider a soft real-time system with three periodic events, with periods of 100, 200, and 500 msec, respectively. If these events require 50, 30, and 100 msec of CPU time per event, respectively.
  - The system is schedulable because $0.5 + 0.15 + 0.2 < 1$.
  - If a fourth event with a period of 1 sec is added, the system will remain schedulable as long as this event does not need more than 150 msec of CPU time per event.
  - Implicit in this calculation is the assumption that the context-switching overhead is so small that it can be ignored.
  - If context-switching time is non-negligible, it must be included in the calculation.

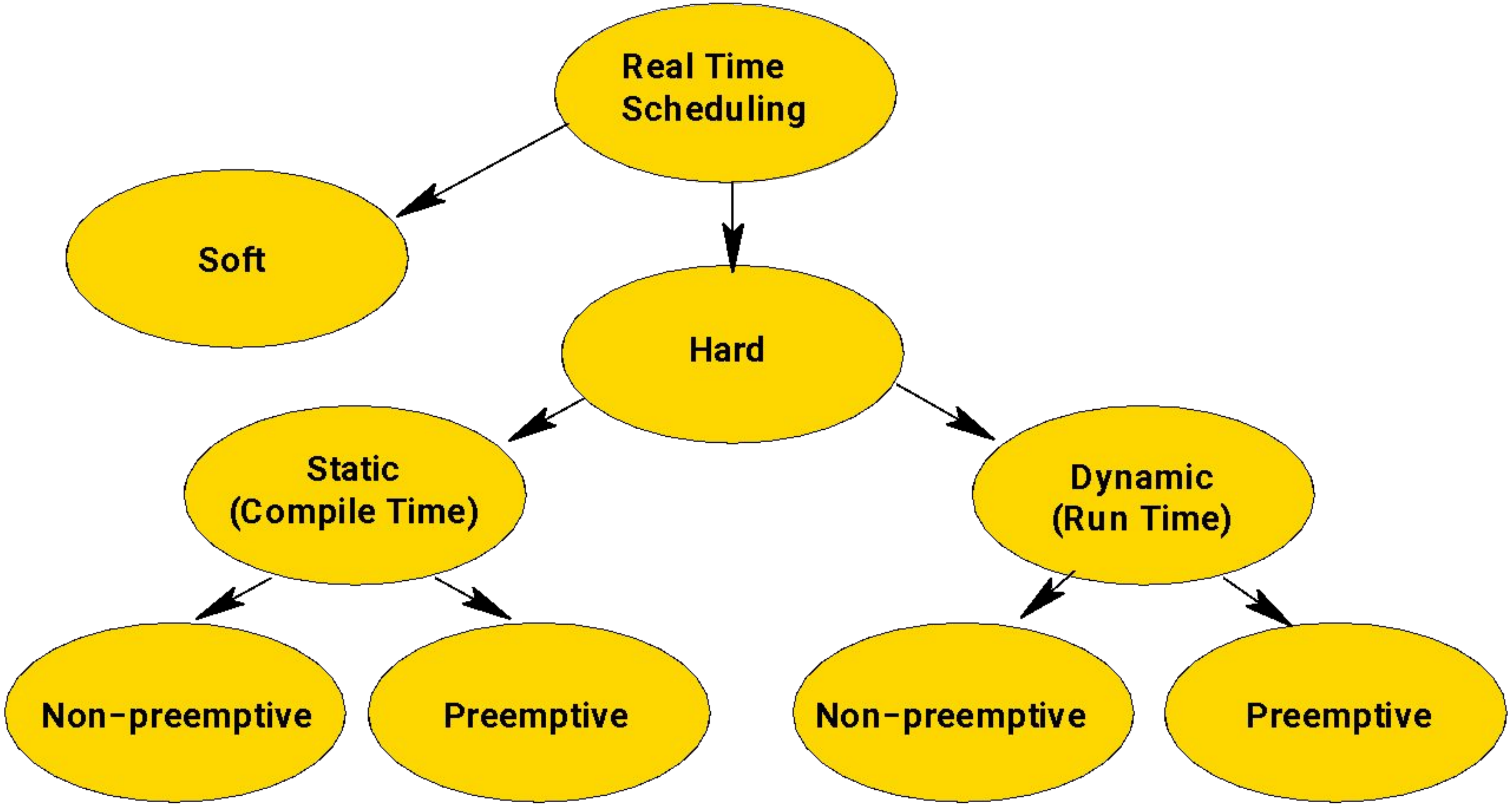
SCHEDULING IN REAL-TIME SYSTEMS

STATIC VS DYNAMIC SCHEDULING

- **Static**
  - Make their scheduling decisions before the system starts running.
  - Static scheduling works only when there is perfect information available in advance about the work to be done and the deadlines that have to be met.
- **Dynamic**
  - Make their scheduling decisions at run time, after execution has started.
  - Dynamic scheduling algorithms do not have the restrictions that apply to static scheduling.

SCHEDULING IN REAL-TIME SYSTEMS

REAL TIME SCHEDULING TAXONOMY



REAL-TIME OPERATING SYSTEMS

Examples of Commonly Used RTOS Types

- **VxWorks** (Wind River Systems), supports x86, MIPS, ARM, PowerPC, SuperH, RISC-V
- **Nucleus** (Siemens EDA), supports ARM, MIPS, PowerPC, RISC-V (replaced VRTX)
- **LynxOS** (Lynx Software Technologies), supports x86, 68k, ARM, PowerPC
- **INTEGRITY** (Green Hills Software), supports ~40 architectures
- **QNX Neutrino** (Blackberry), supports x86-64, ARM32, ARM64
- **Windows IoT** (Microsoft), family of embedded OS variants of Windows
- **FreeRTOS** (Freertos.org), supports 40+ microcontroller architectures
- **RTAI** (RTAI.org), Linux kernel extension, supports x86, PowerPC, ARM, MIPS
- NB in the embedded and IoT sectors, ARM and i86 are common but not exclusive!

ASSESSING RTOS PERFORMANCE

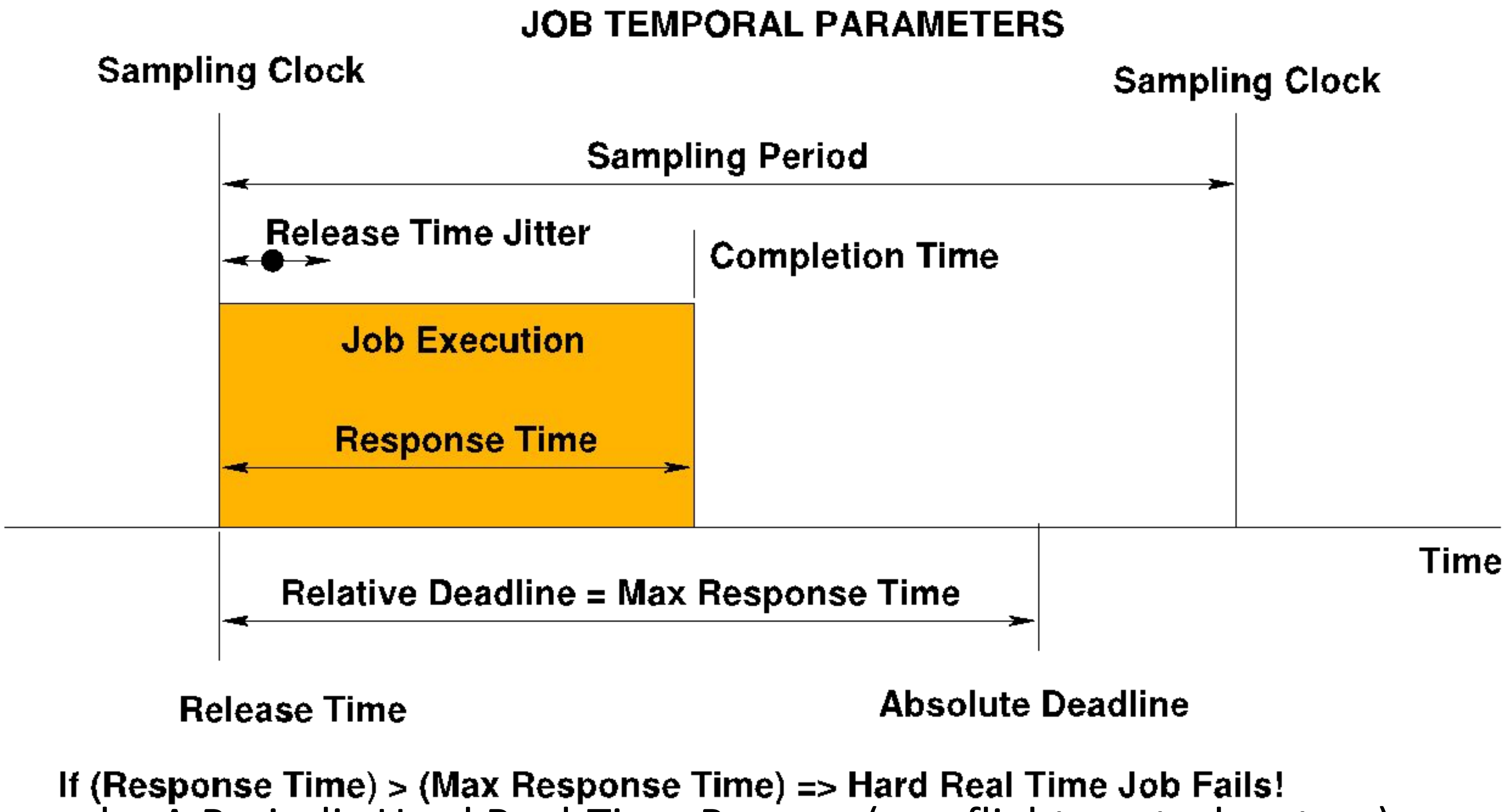
ASSESSING RTOS PERFORMANCE

METRICS

- **Turnaround Time**
  - Time interval from entry to completion
  - The total time a process spends either running or waiting, until it completes.
    - ❑ Total time = Completion time - Arrival time
- **Response Time** (Interrupt Latency)
  - Average time between arrival time, and the process beginning to respond (i.e. returning to the RUNNING state)
  - Critically important for many hard real-time processes

MONASH University

ASSESSING PERFORMANCE



Example: A Periodic Hard Real-Time Process (e.g. flight control system)

ASSESSING RTOS PERFORMANCE

METRICS

- Throughput
  - The average rate at which processes are passing through the system from entry to completion
  - Can be measured as number of processes per time taken, or its inverse.
    - E.g. a system admits and completes 12 processes in 6 seconds.
      - Throughput = 12 processes / 6 sec = 2 proc/sec
      - Or inverse: 6 sec / 12 proc = 0.5 seconds per process

REAL-TIME SCHEDULING ALGORITHMS

SCHEDULING IN REAL-TIME SYSTEMS

Earliest-Deadline-First (EDF) Algorithm

- A way to assign priorities to jobs is on the basis of their deadlines.
- Earliest-Deadline-First (EDF) algorithm is based on the priority assignment whereby the earlier the deadline, the higher the priority.
- This algorithm is important because it is optimal when used to schedule jobs on a processor when preemption is allowed and there is no contention for resources
- When preemption is allowed and jobs do not contend for resources, the EDF algorithm can produce a feasible schedule of a set of jobs J with arbitrary release times and deadlines on a processor, if and only if J has feasible schedules. resources.

SCHEDULING IN REAL-TIME SYSTEMS

EXAMPLE: Earliest-Deadline-First (EDF) Algorithm (Zagan and Gaitan, 2016)

TABLE II

THE PARAMETERS OF A SET OF FIVE TASKS

	a_i	C_i	D_i
τ_5	12	5	18
τ_4	7	8	24
τ_3	4	4	10
τ_2	1	3	11
τ_1	0	2	4

τ_i – task identifier
a_i – task arrival time
C_i – task compute duration
D_i – task deadline

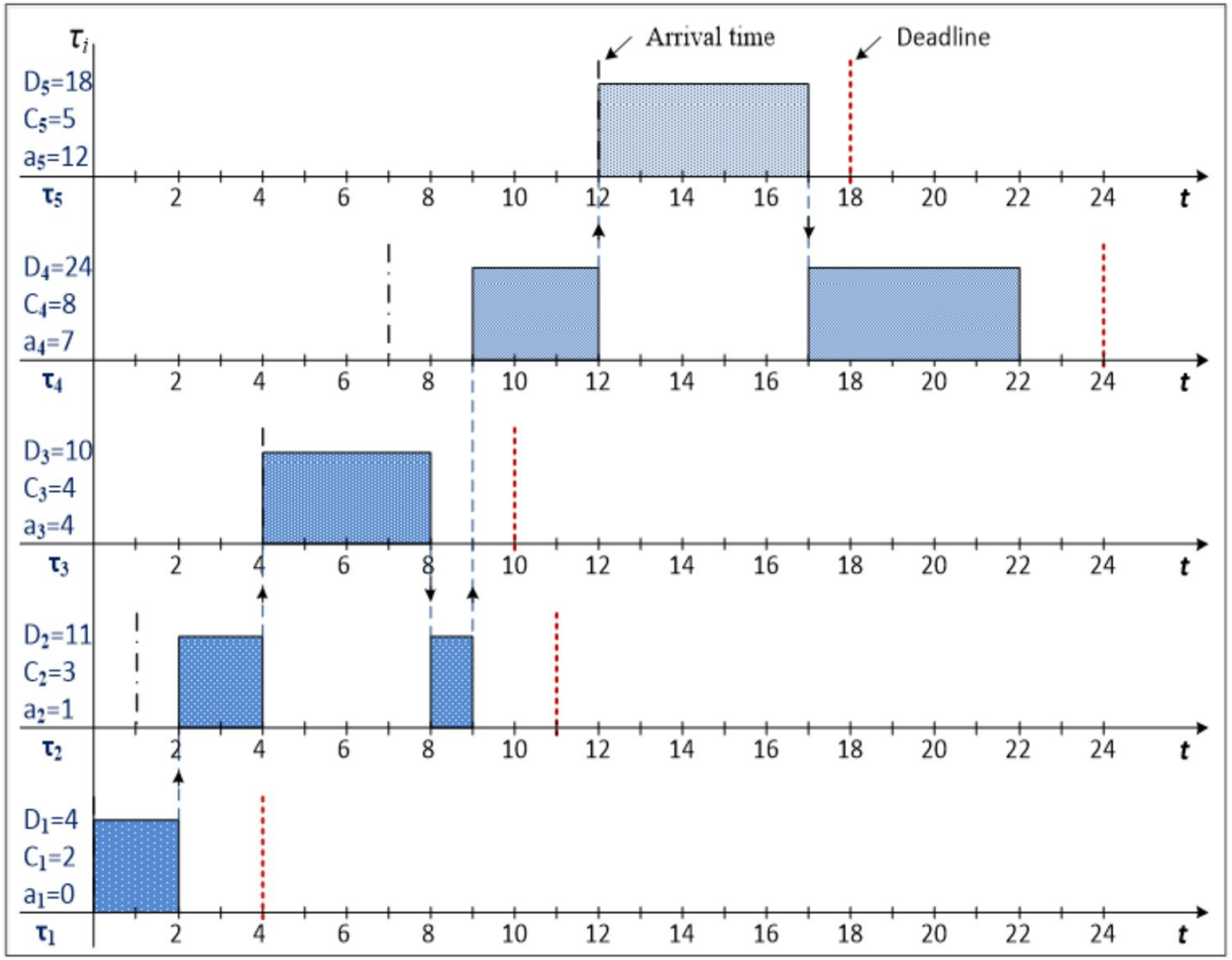


Fig. 1. A scheduling example using the EDF algorithm.

MONASH University

SCHEDULING IN REAL-TIME SYSTEMS

EXAMPLE: Earliest-Deadline-First (EDF) Algorithm (Zagan and Gaitan, 2016)

- Cited: *"Fig. 1 shows an example of scheduling a set of five tasks using the EDF algorithm. At moment t = 0, task τ_1 enters execution, and at moment t = 1, task τ_2 cannot interrupt τ_1 because D1 < D2. Task τ_1 completes execution at time moment t = 2, and at moment t = 4, when τ_2 is being executed, task τ_3 interrupts τ_2 because D3 < D2. To be noted that at time moment t = 7, task τ_4 does not interrupt τ_3 because D3 < D4. When τ_3 completes execution, the CPU is assigned to task τ_2. Task τ_4 is executed at moment t = 9, but it is interrupted at t = 12 by τ_5, because the last one has a lower deadline. Task τ_4 re-enters in execution at moment t = 17, when τ_5 completes its own."*

MONASH University

SCHEDULING IN REAL-TIME SYSTEMS

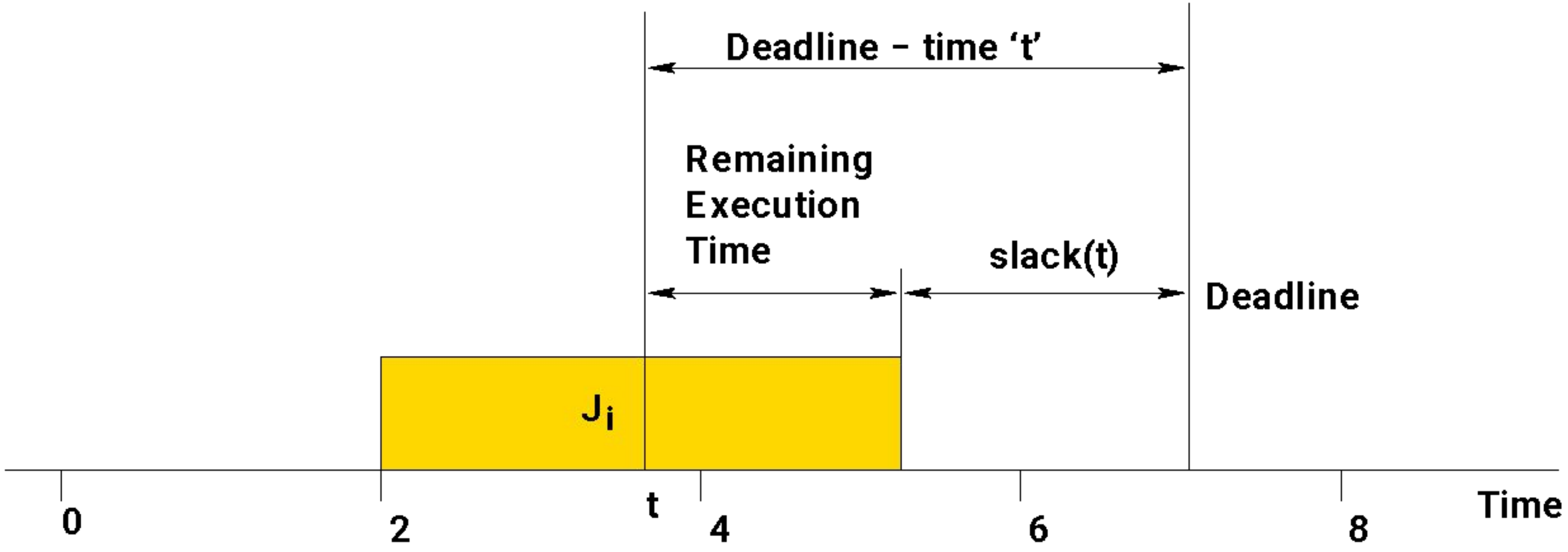
Least-Slack-Time-First (LST) Algorithm

- Another algorithm optimal for scheduling preemptive jobs on one processor is the Least-Slack-Time-First (LST), also called Minimum-Laxity-First (MLF) algorithm.
- At any time t, the slack (or laxity) of a job with deadline d, is equal to d - t minus the time required to complete the remainder of the job.
- slack(t) = (deadline - t) - (execution time remaining(t))
- *slack(t) = deadline - t - etr(t)*

REAL-TIME SCHEDULING ALGORITHMS

LEAST SLACK TIME FIRST ALGORITHM

slack(t) = d_i - t - etr_i(t) ; etr = execution time remaining
slack is re-calculated when a job is released or a job completes



REAL-TIME SCHEDULING ALGORITHMS

LEAST SLACK TIME FIRST ALGORITHM

Execution Time (Feasible Interval]

slack(t) = d_i – t – etr_i(t) ; etr = execution time remaining

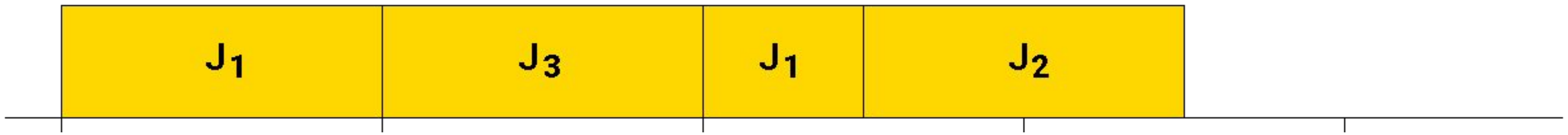
slack is re-calculated when a job is released or a job completes

J_1, t = 0: r_1 = 0, d_1 = 6, etr_1 = 3 => slack time = 6 – 0 – 3 = 3 @ t = 0

J_1, t = 2: r_1 = 0, d_1 = 6, etr_1 = 1 => slack time = 6 – 2 – 1 = 3 @ t = 2
J_3, t = 2: r_3 = 2, d_3 = 7, etr_3 = 2 => slack time = 7 – 2 – 2 = 3 @ t = 2

J_1, t = 4: r_1 = 0, d_1 = 6, etr_1 = 1 => slack time = 6 – 4 – 1 = 1 @ t = 4
NB J_2 cannot be calculated since it is released at t = 5

Pre-emption (arbitrary since identical slack magnitude)

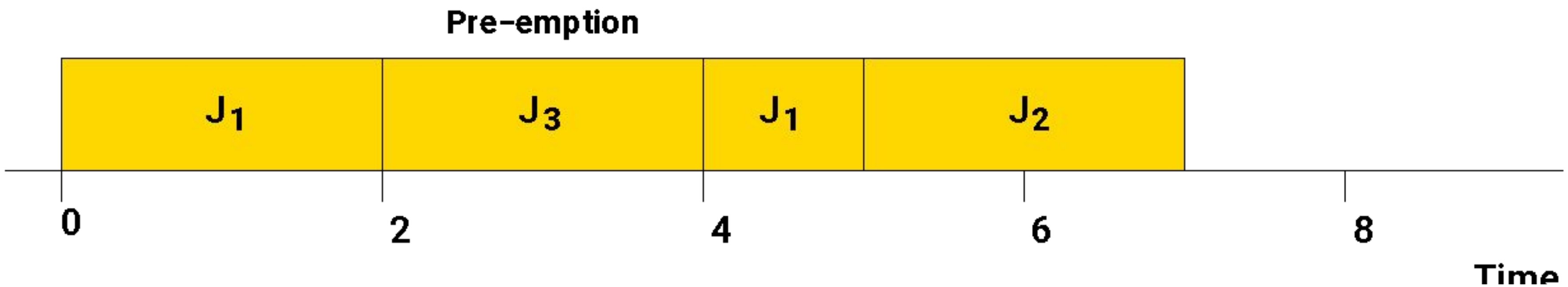


REAL-TIME SCHEDULING ALGORITHMS

LEAST SLACK TIME FIRST ALGORITHM

Execution Time (Feasible Interval]

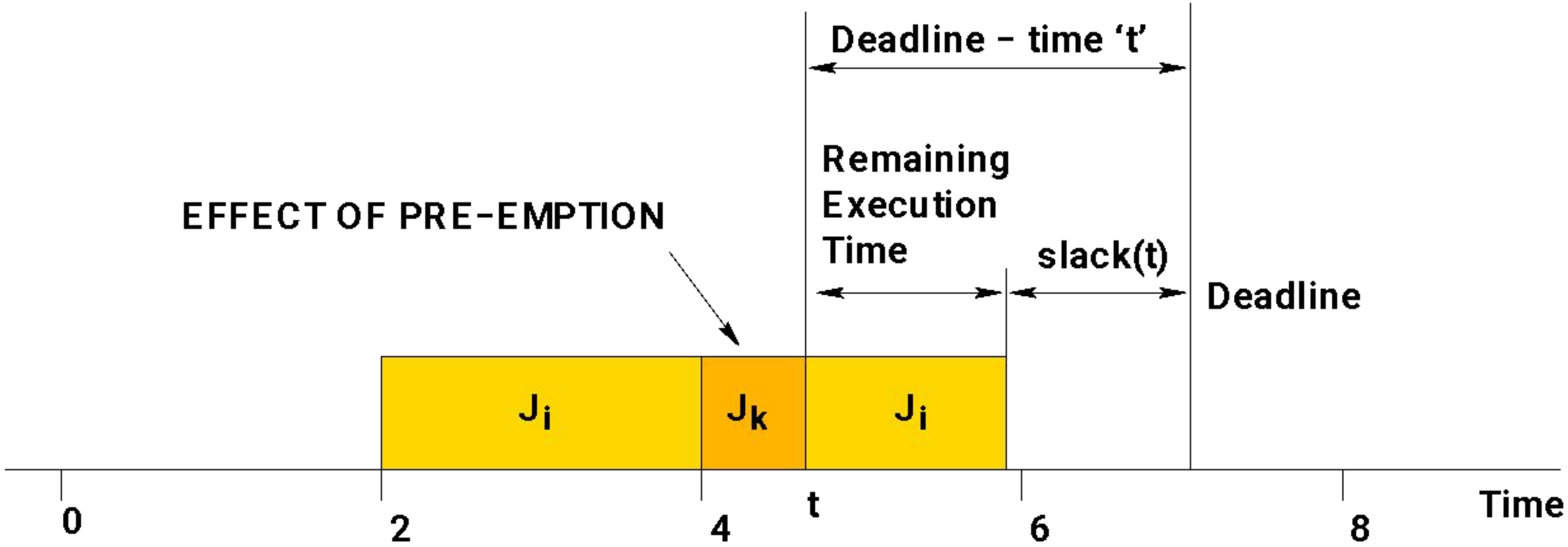
t = 0: r1 = 0, d1 = 6, e1 = 3 => slack time = 3 @ t = 0
i.e. slack time for J1 = 6 – t – (3 – t) while J1 is executing
t = 4: r1 = 0, d1 = 6, remaining e1 = 1 => slack time = 1 @ t = 4
t = 4: r2 = 5, d2 = 8, e2 = 2 => slack time = 4 + 2 = 6 @ t = 4



REAL-TIME SCHEDULING ALGORITHMS

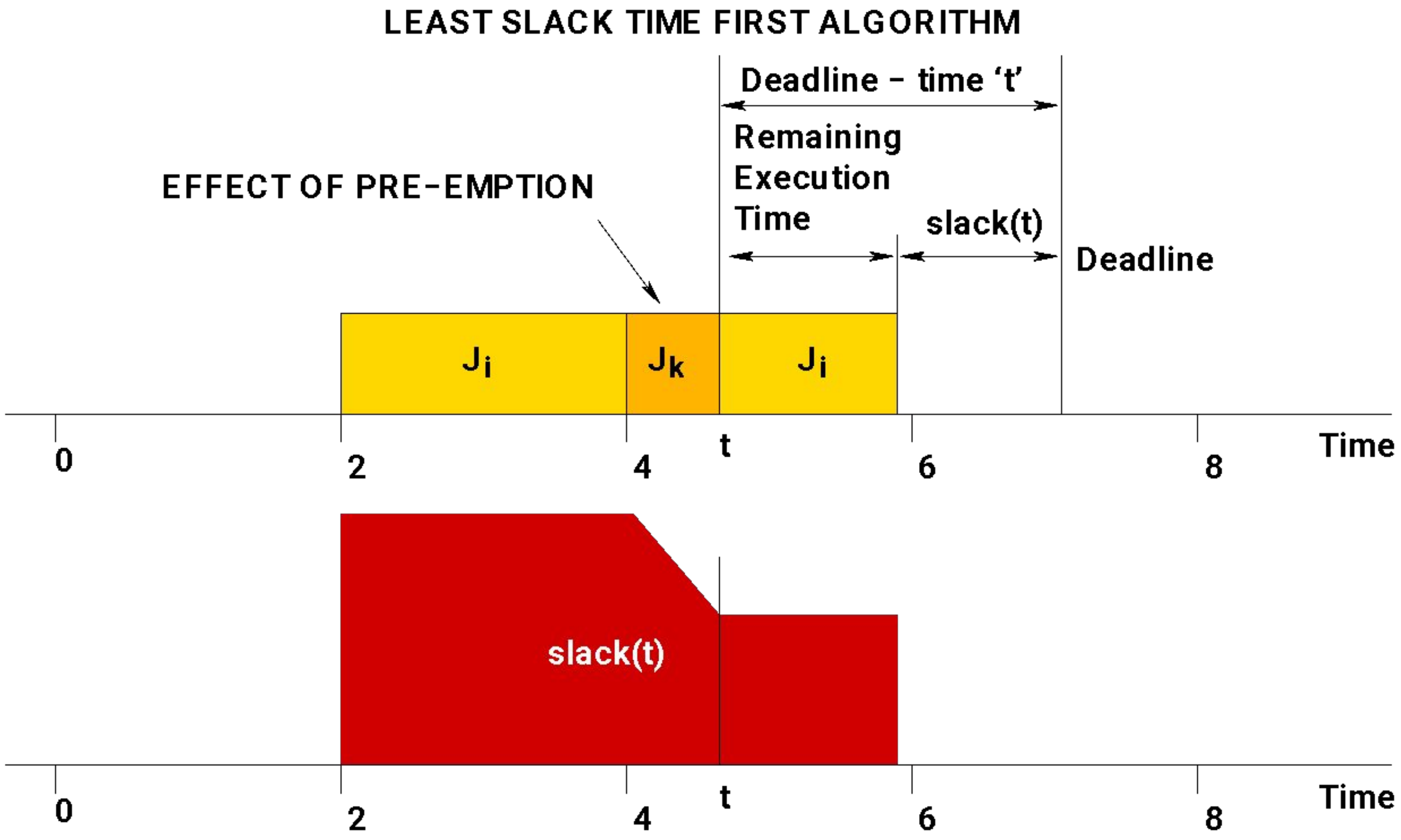
LEAST SLACK TIME FIRST ALGORITHM

slack(t) = d_i - t - etr_i(t) ; etr = execution time remaining
slack is re-calculated when a job is released or a job completes

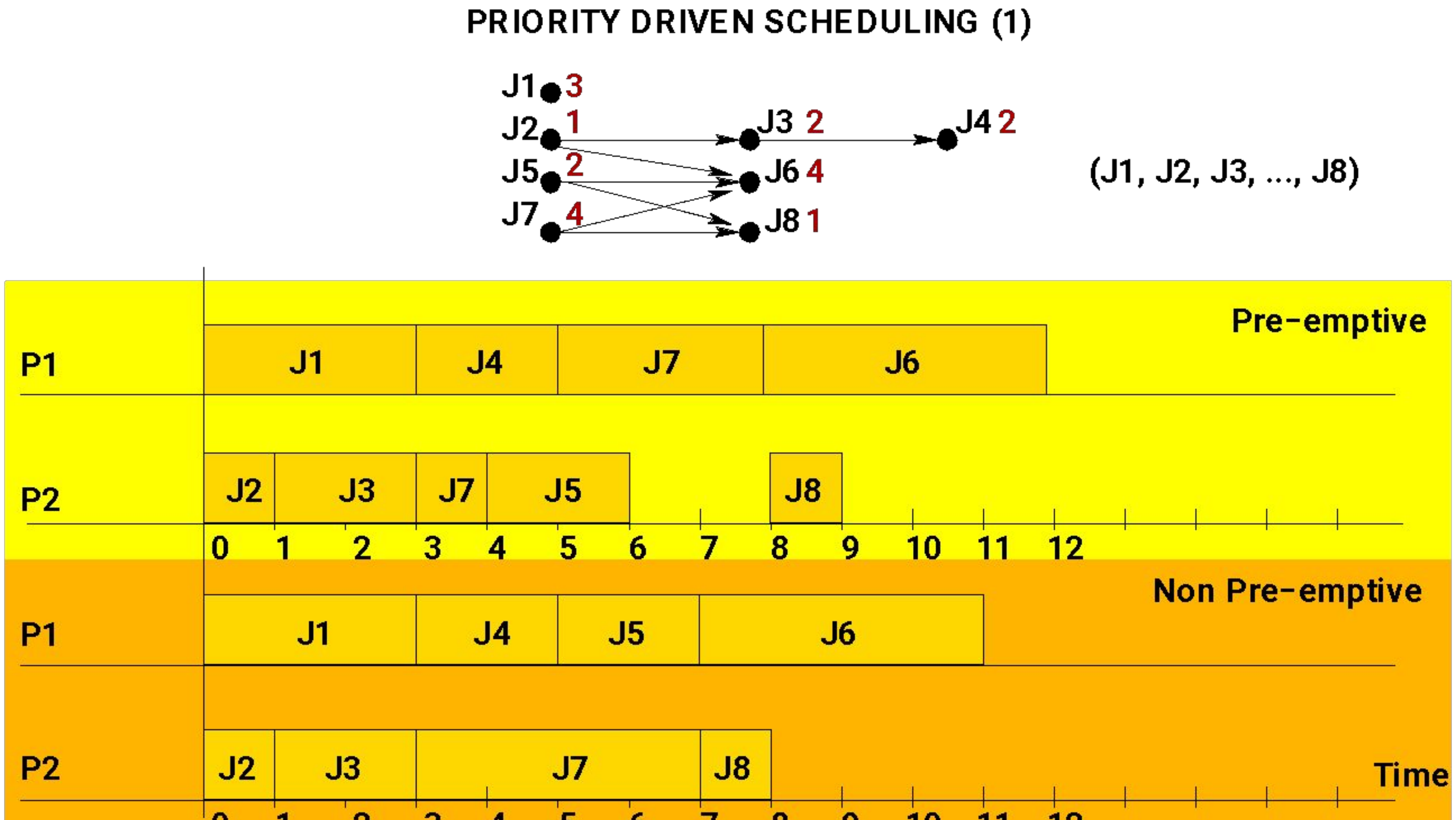


Every time a job is pre-empted its slack time decreases

REAL-TIME SCHEDULING ALGORITHMS

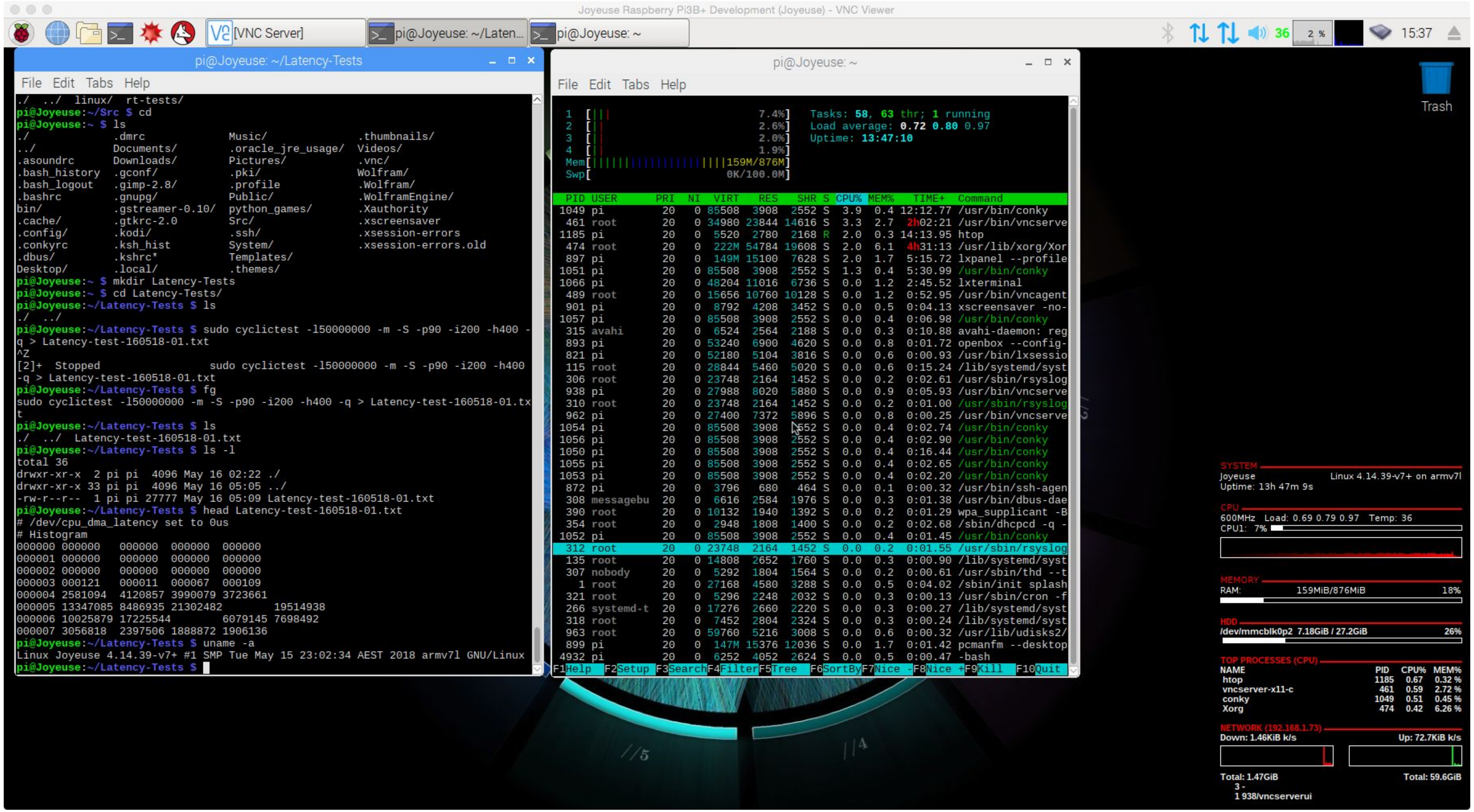


REAL-TIME SCHEDULING ALGORITHMS



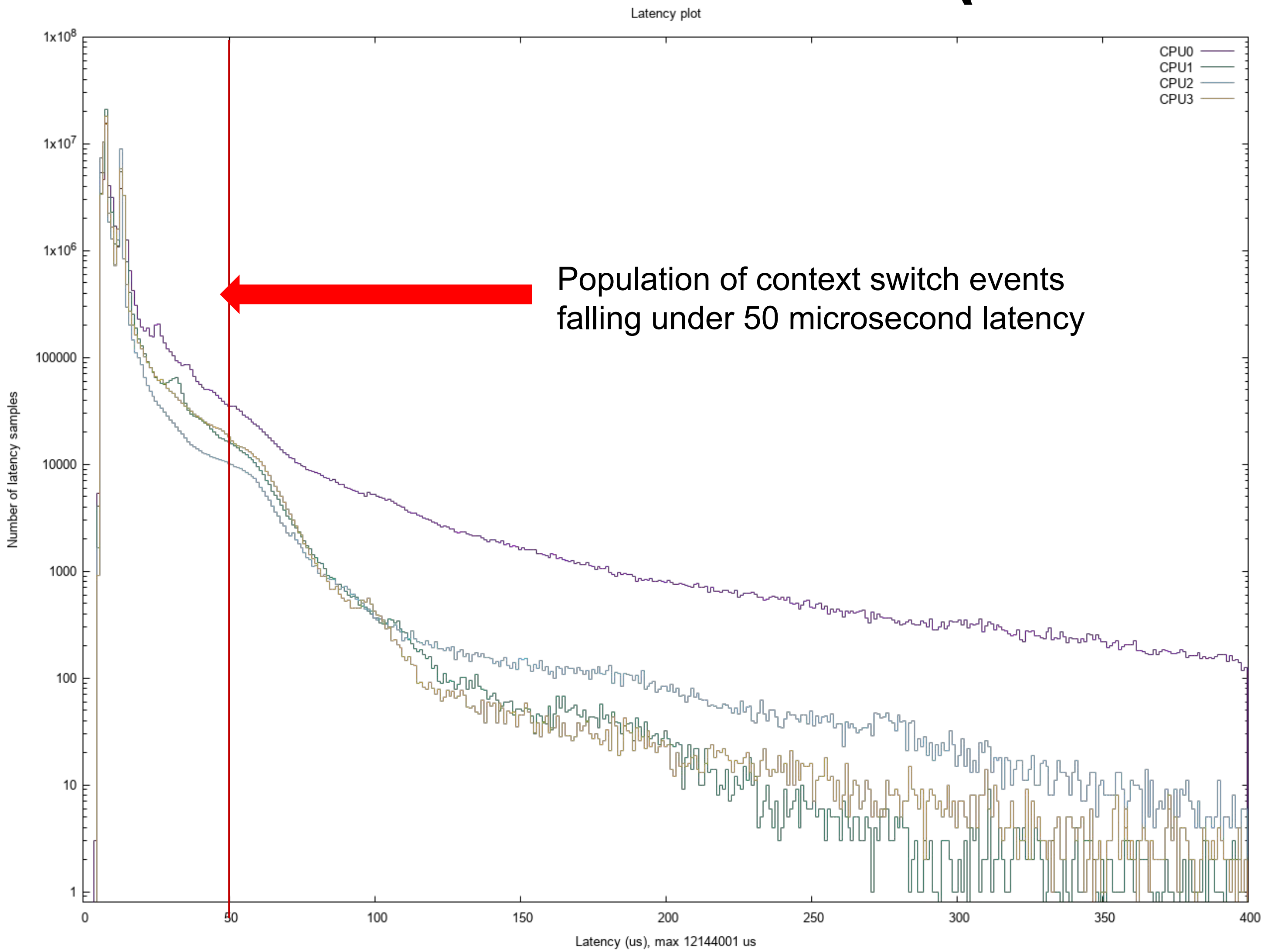
Non-Real-Time Pre-Emptive and Non-Pre-emptive Priority Driven Scheduling

REAL-TIME LATENCY TESTING (RPi3B RTAI)



MONASH University

REAL-TIME LATENCY TESTING (RPi3B RTAI)



MONASH University