Real Time Operating Systems

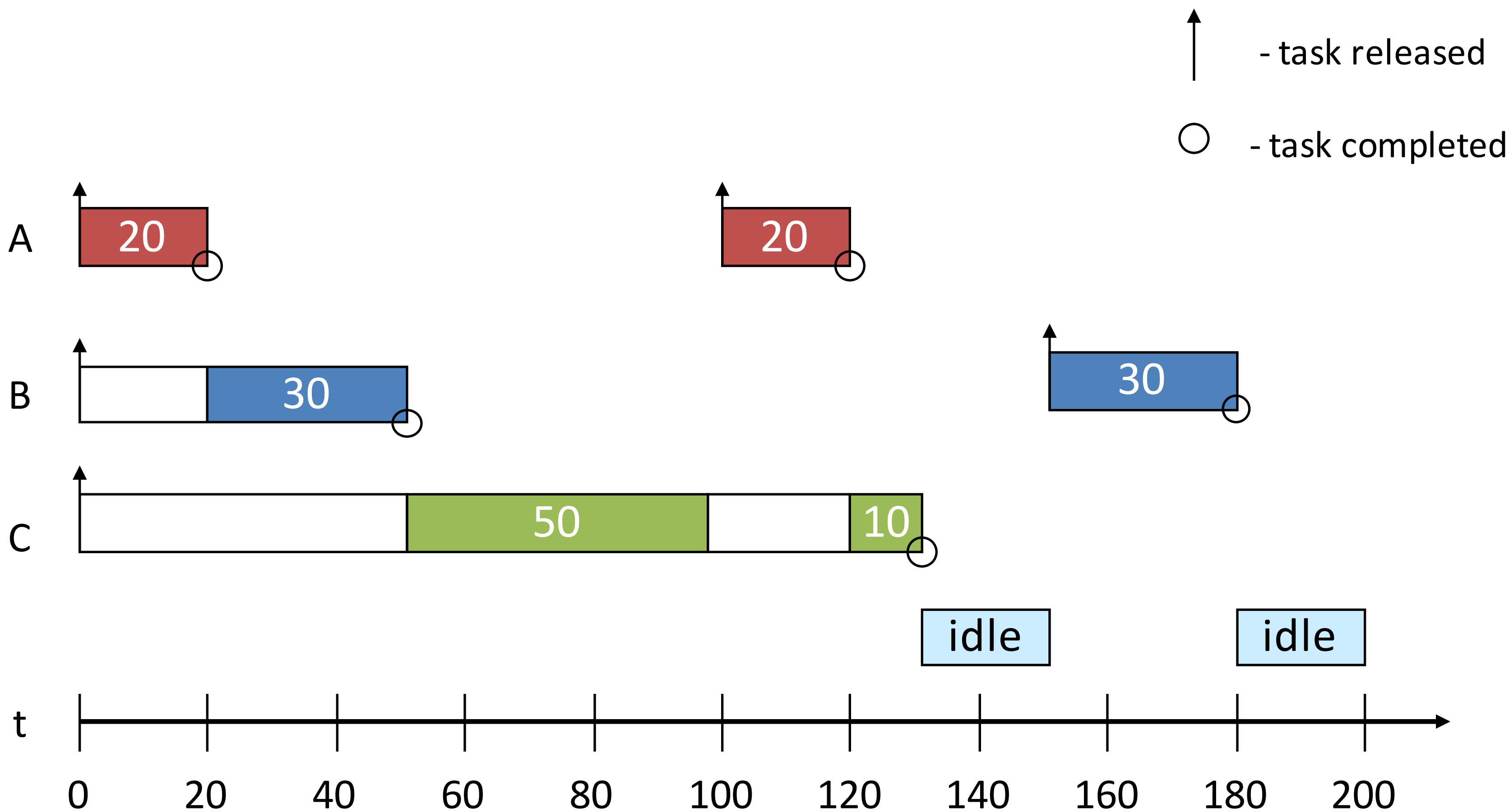
Scheduling & Schedulers

ROYAL MILITARY COLLEGE OF CANADA
ELECTRICAL & COMPUTER
ENGINEERING

GÉNIE ÉLECTRIQUE
ET GÉNIE INFORMATIQUE
COLLÈGE MILITAIRE ROYAL DU CANADA

RMC CMR

Tasks Model



- task released
- task completed
A
20
20
B
30
30
C
50
10
idle
idle
t
0
20
40
60
80
100
120
140
160
180
200

Scheduling Decisions

- **assignment**
- **ordering**
- **timing**

Scheduling

- Scheduling algorithms - **Policy**
  - the **rules** under which tasks are assigned to processors are defined by the system scheduling algorithm
- The **scheduler** - **Mechanism**
  - the module which implements these algorithms and protocols

When to take the decisions?

- Design time
- Run-time

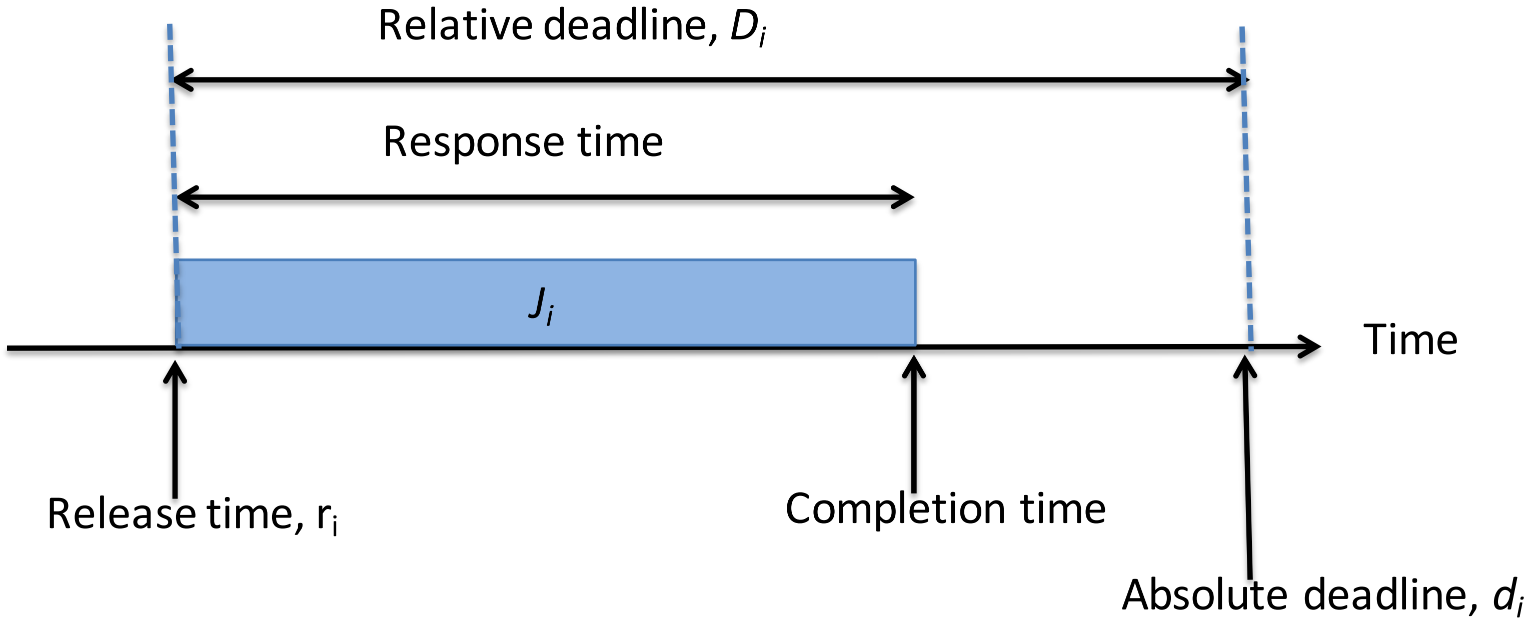
Types of Schedulers

- Fully static scheduler
- Static order scheduler (off-line scheduler)
- Static assignment scheduler
- fully-dynamic scheduler

(Static assignment scheduler and fully-dynamic scheduler: on-line schedulers)

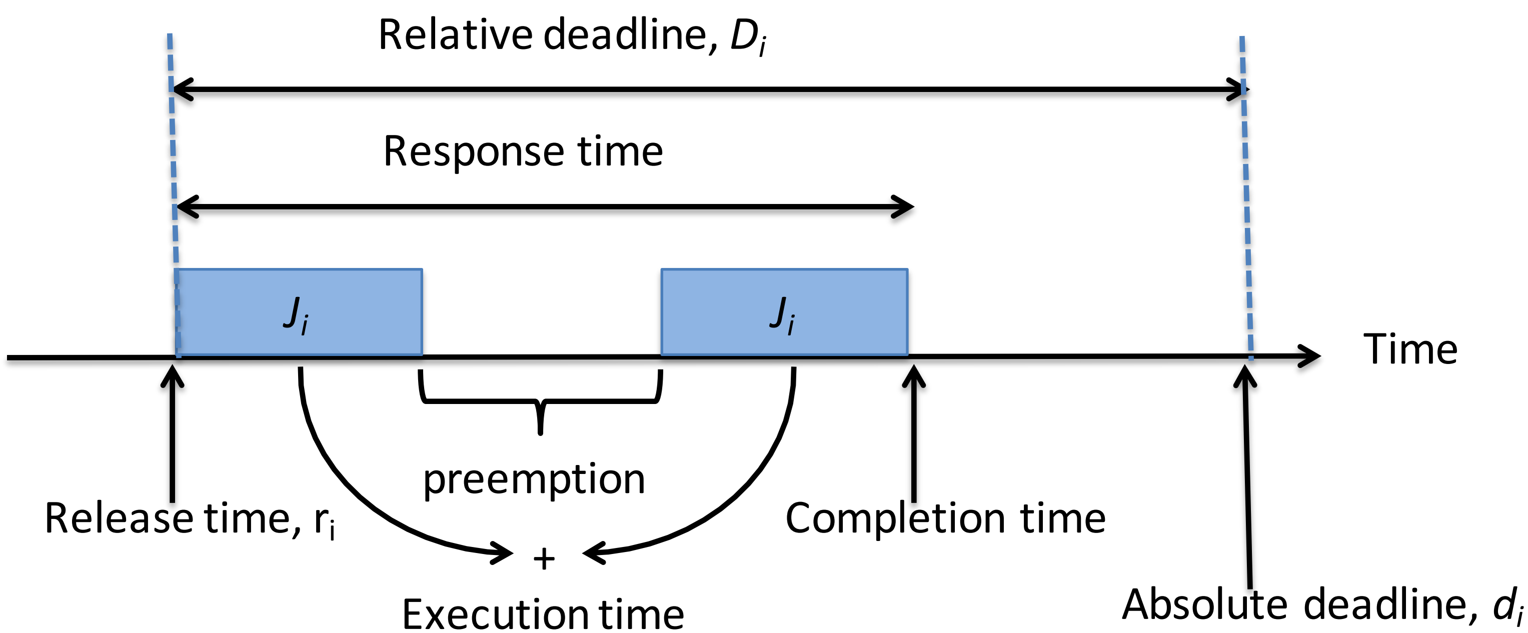
Preemption

Non-preemtive scheduler



Preemption

Preemptive scheduler



Choice of algorithm

The choice of scheduling algorithm depends on the goals

- Meeting all the deadlines
- Minimize the response time
- Maximize utilization
- Combination of goals
- Etc.

Schedules

- Schedules
  - a **schedule** is the assignment by the scheduler of all tasks in the system to the available processors
  - we assume that the scheduler is correct in that ->
- It only produces **valid schedules**
  1. at any time one processor is assigned at most one job
  2. at any time each job is assigned at most one processor*
  3. no job is scheduled before its release time
  4. the total amount of processor time allocated is equal to each job's maximum execution time (or actual)**
  5. all precedence and resource constraints are met

*** implies no parallel processing at job level** **** depends on algorithm**

Schedules

- Feasible schedules
  - a **feasible schedule** is one in which all jobs meet their timing constraints (usually deadlines)
  - to say that a set of jobs is **schedulable** by an algorithm implies that scheduling under this algorithm always produces a feasible schedule
- Optimal algorithms
  - an algorithm which always produces a feasible schedule, if one exists, is said to be optimal

What does this say about a set of jobs which cannot be scheduled by an optimal algorithm?

Approaches to Scheduling

There are generally three broad classes, or approaches to processor scheduling:

- Clock-driven scheduling
- Weighted Round-robin scheduling
- Priority scheduling

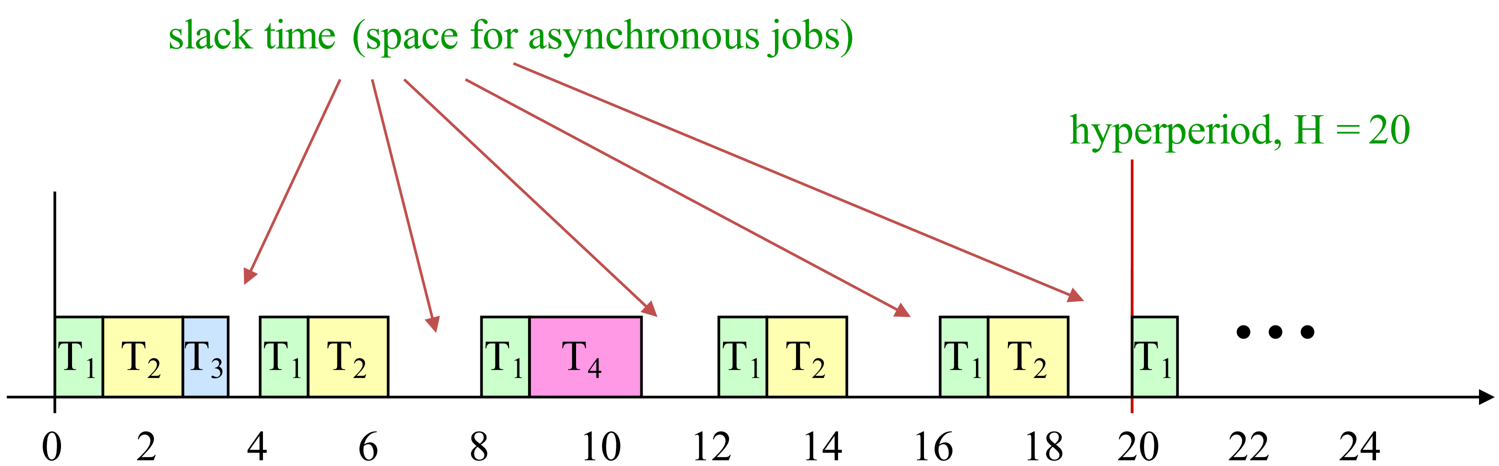
Clock-Driven Scheduling

- At pre-specified time instants, a task or sequence of tasks are scheduled onto the processor(s)
  - job/task scheduling is designed off-line
  - all system job parameters are known a priori and are fixed
  - scheduling overhead is minimal
  - may be implemented using a hardware timer mechanism

Clock-Driven Scheduling

- Observe the schedule below for this system:

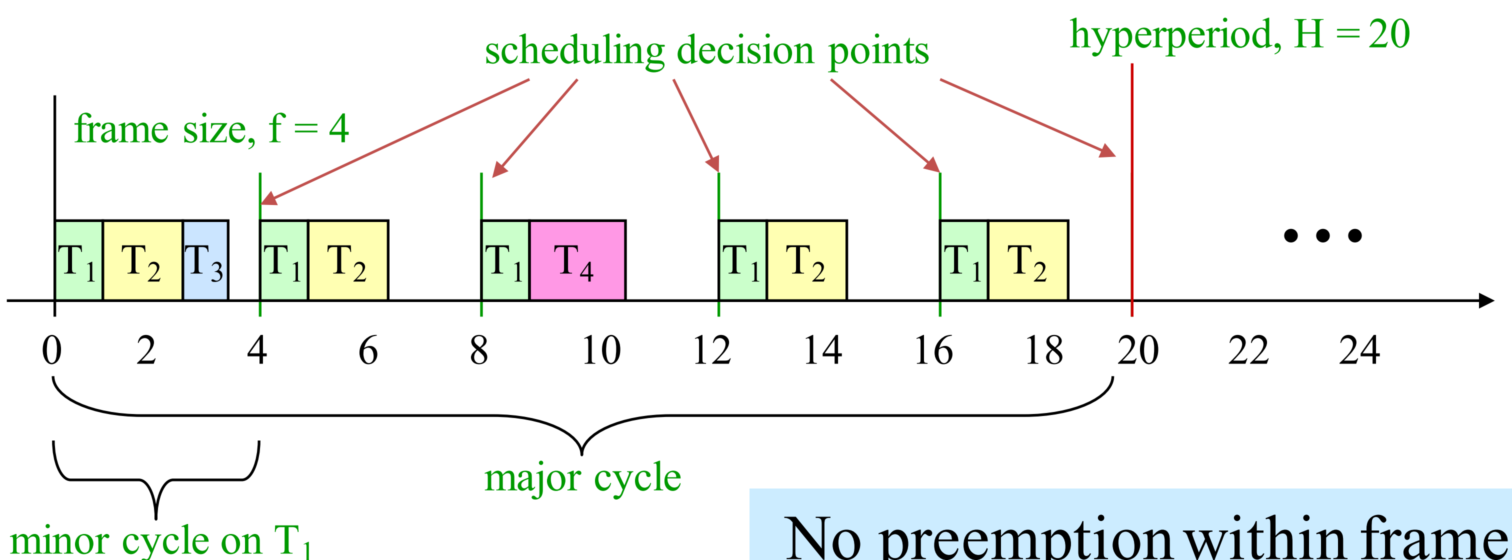
$\{T_1 = (4,1)\ ,\ T_2 = (5,1.5),\ T_3 = (20,1)\ ,\ T_4 = (20,2)\}$



How do we choose this design? How do we implement this design?

Clock-Driven Scheduling

- Rather than making scheduling decisions at arbitrary times, limit decision making to specific points in the hyperperiod, at frame boundaries



No preemption within frames

Round-Robin Scheduling

- In straight **round robin scheduling**,
  - ready tasks are inserted in a FIFO queue and when they reach the front of the queue they are given an equal slice of processor time
  - tasks are preempted at the end of their slice regardless of completion status
  - therefore in an n job system, each job gets 1/n th of the processor
  - typically processor time slices are quite short with respect to execution times

Weighted Round-Robin Scheduling

- In **weighted round robin scheduling**,
  - each task/job gets *wt* slices of the processor time depending upon the weight of the task
  - therefore in an n job system, job J_i gets $wt_i/(\Sigma wt)$ of the processing time
- Example:

Job	wt	e	% CPU
J_1	2	3	___
J_2	5	7	___
J_3	2	4	___
J_4	1	3	___

What would 1 round of time slice scheduling look like?

Priority-Driven Scheduling

- In a priority-driven system, ready tasks are assigned to processors according to their relative priorities
  - also called greedy scheduling or list scheduling
  - priorities may be static or dynamic
  - tasks may be preemptable or nonpreemptable

Priority-Driven Scheduling

- A **dynamic priority scheduling** algorithm allows for task/job priorities to change at run-time
- With **static priority scheduling** the tasks/jobs have a fixed (generally) a priori priority assignment
  - do not confuse this with dynamic & static systems

Validating the Timing Constraints

- Step 1 – specification correctness
  - Check for consistency/correctness of constraints
- Step 2 – task feasibility
  - Validate that each task can meet its constraints if it were to execute standalone on the processor
- Step 3 – system validation*
  - Validate that all tasks together under the given scheduling & resource management strategy meet their respective constraints

* The difficult part!

The Periodic Task Model

- The periodic task model is a classical **workload** model of real-time systems; one which we will study further in this course
  - the underlying assumption is that each regular or semi-regular function of the system be modeled as a **periodic task**
  - each periodic task (T_i) is defined by its period (p_i) and its worst-case execution time (e_i)
  - a task's **period** is defined as the minimum length of all time intervals between release times of consecutive jobs

The Periodic Task Model

- The **accuracy of the model** is dictated by how closely it resembles the system under study
  - it is quite accurate when the release time jitter is small (best when all tasks are truly periodic) and when the execution times of tasks are well known and have small deviations
  - conversely the accuracy of the model degrades if the release time jitters are high and/or the execution times have high variance

References

[1] Liu, J.W.S., “Real-Time Systems”, Prentice-Hall, 2000.

[2] Lee, E. A., Seshia, S. A. “Introduction to Embedded Systems - A Cyber-Physical Systems Approach”, Second Edition, MIT Press, 2017.