

Real-Time Systems

Hermann Härtig

Event-Driven Scheduling

Last updated: 01/12/10

Event-Driven / Priority-Based Scheduling

Outline:

- Principles
- Preemption and Scheduling
- EDF and LST as “dynamic” scheduling methods
 - Few SMP insights
 - Anomalies
- Fixed Priority schedulers
- Admission based on Utilization
- mostly following Jane Liu, Real-Time Systems



Principles

- Admission:
 - check if feasible schedule exists
 - decide how “priorities” are assigned to Jobs
- Scheduling / Dispatching:
At events, jobs are selected according to their priorities

Important properties:

- decisions, which job to execute next at EVENTS
(not time instants)
Events are modeled as releases and completions of jobs
- a (timer) interrupt is an (implementation of a) special event
- never leaves a resource idle intentionally (“greedy”)
- scheduling on line,
admission on line or off line
- scheduling must be simple
(otherwise difficult/not possible on line)

- **Optimality of Schedulers:**
A scheduling method X is called “optimal” in a class of scheduling problems, if the following property holds:
If there a scheduling method Y that produces a feasible schedule than X produces a feasible schedule as well

- some “restrictive” assumptions of time-driven systems are given up:
 - fixed inter-release times
→ minimum inter-release times
 - fixed number of rt tasks in systems
real-time and non real-time, number can vary
 - a priori fairly well known parameters
tasks come and go, overloading, ...

Priority Assignment Following “Criticality”

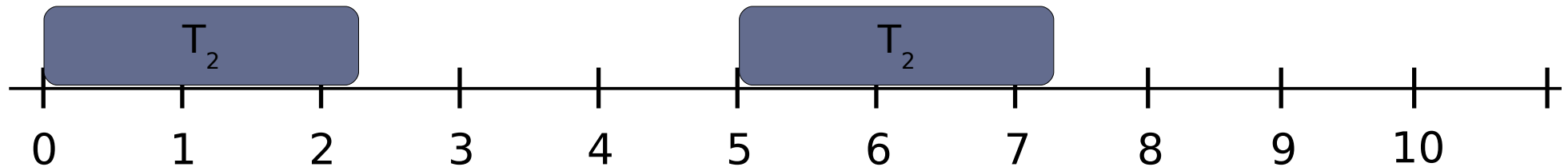
Jane Liu

- The more critical a task, the higher its priority

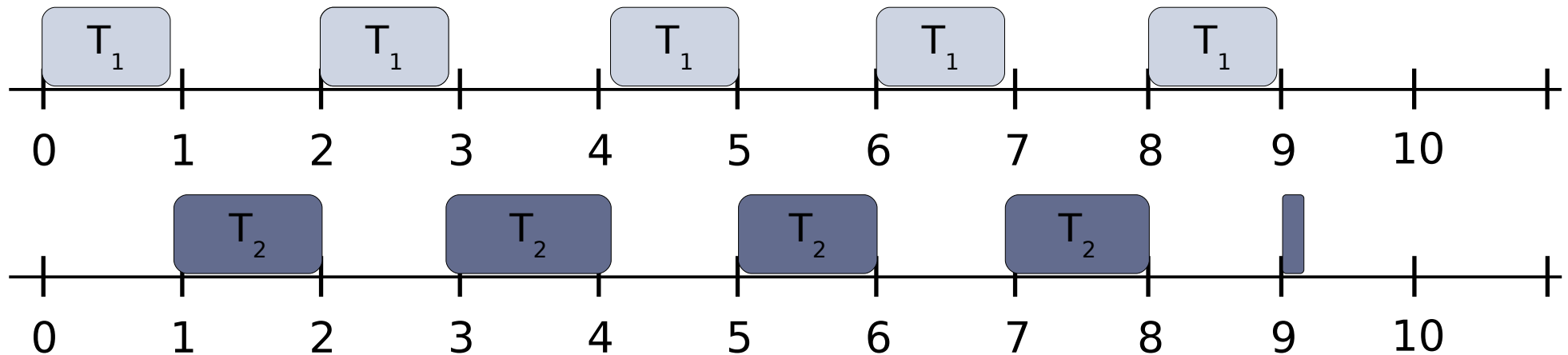
$T_1: (2, 0.9)$

$T_2: (5, 2.3)$

→ T2 more critical than T1

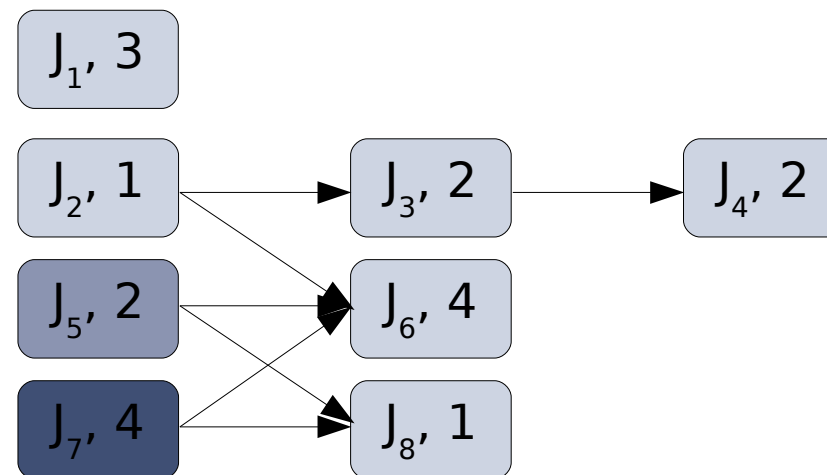


- T_1 misses deadline in Job 1 and 2/3, unnecessarily ...

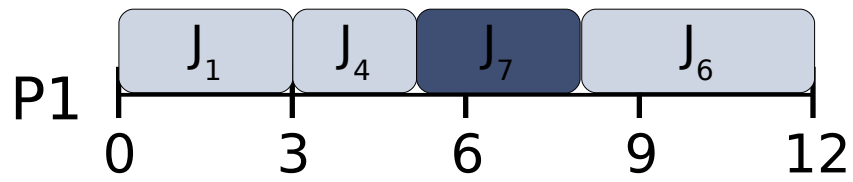


- Static vs dynamic allocation to processors
 - Static: jobs are assigned to processors once and stay
 - Dynamic: jobs “migrate”
 - example: one run queue served by all processors
- preemptive or non preemptive
 - some tasks
 - all tasks

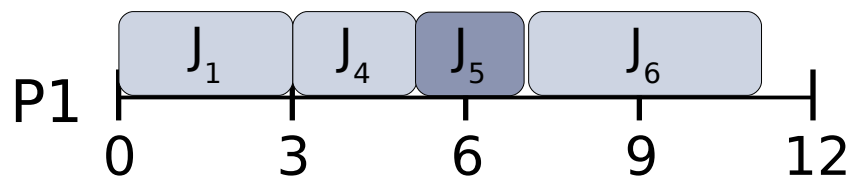
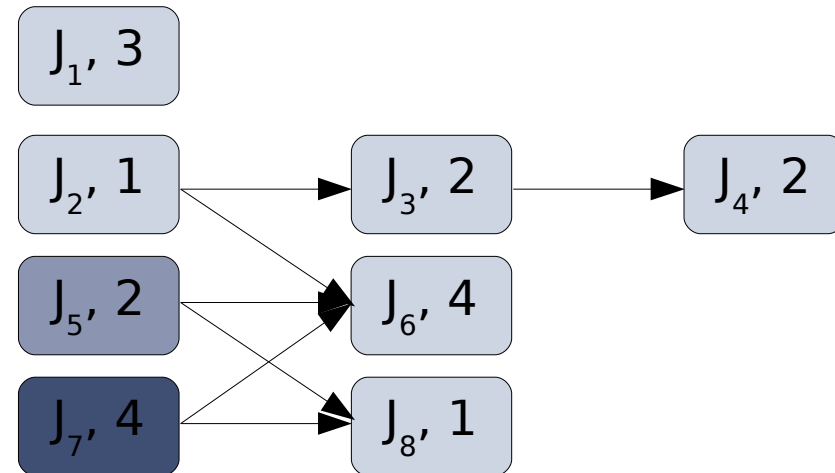
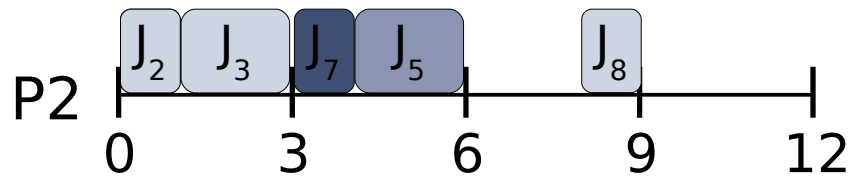
- 2 processors,
- Tasks: Notation used below: J_i, e_i
 - release time of J_5 is 4, all others 0; (!)
- static priorities, assigned such that:
 $i < k \Rightarrow \text{Prio}(J_i)$ higher than $\text{Prio}(J_k)$
- Tasks can “migrate”
- precedence graph:



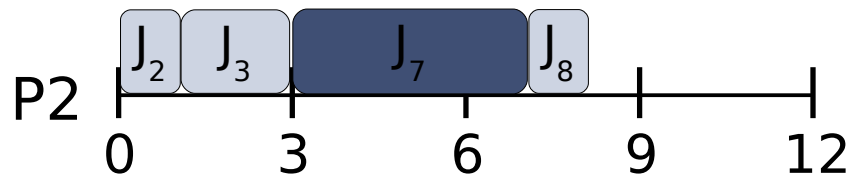
Example, executions



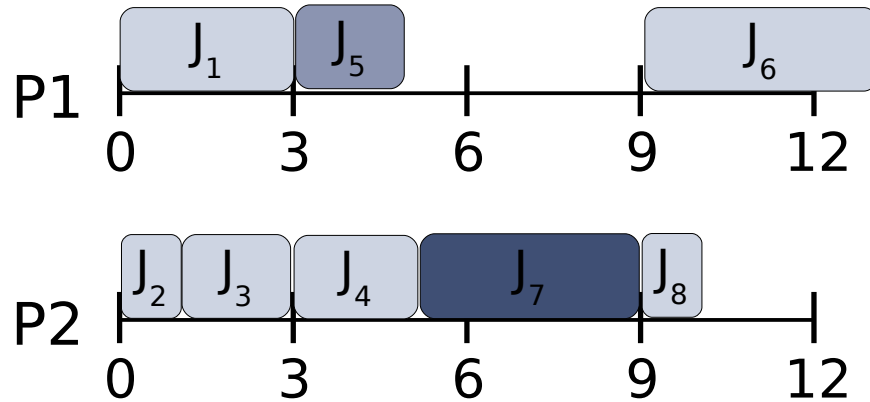
preemptiv



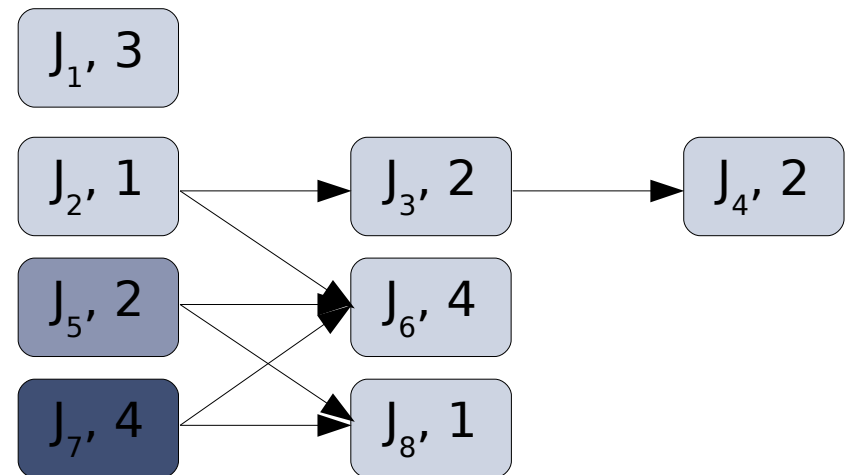
non-
preemptiv



Modified Example: release time of $J_5 = 0$



non-preemptiv



Which is better?

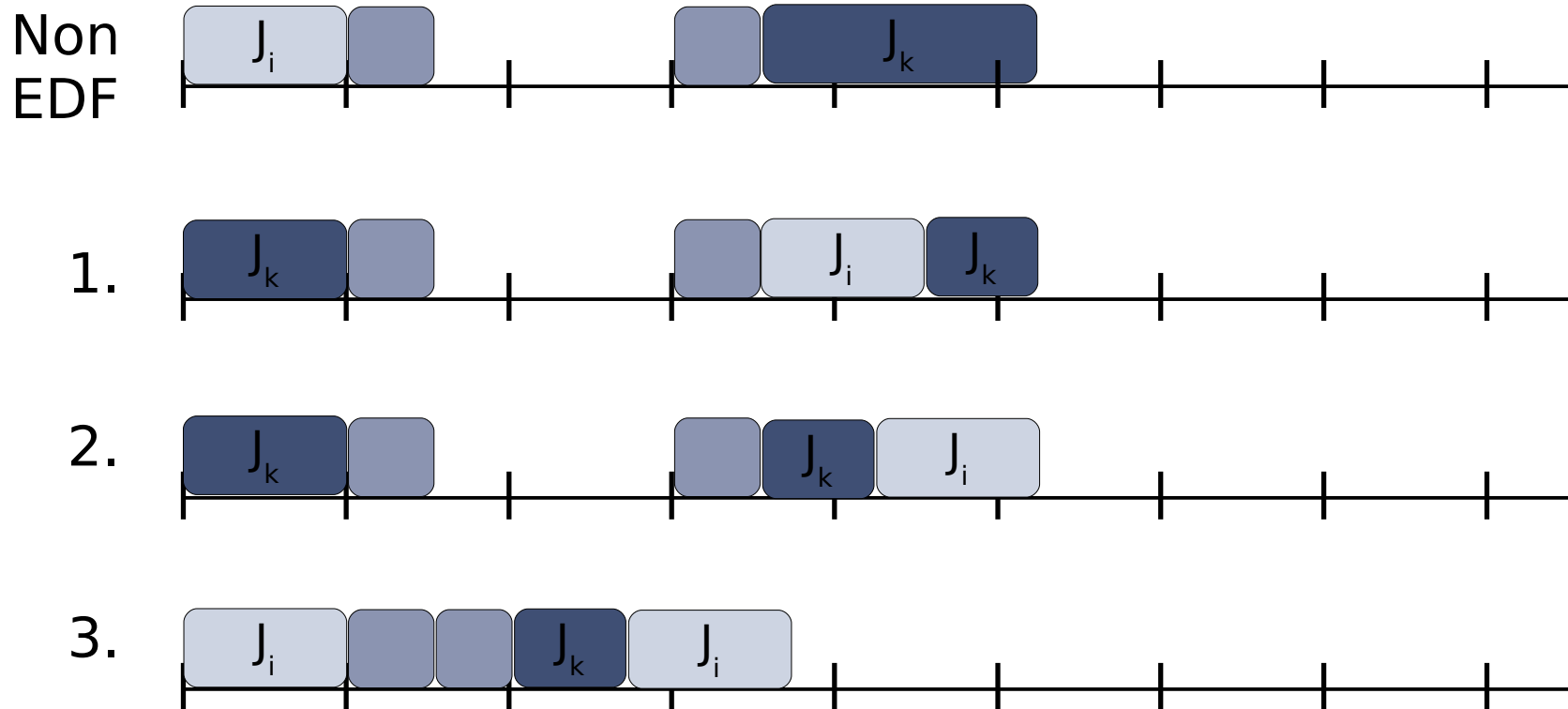
- No *general* answer known!
- If jobs have same release time:
preemptive is better (or equal) in a multiprocessor system if cost for preemption is ignored
- more precise: “makespan” is better
(makespan = response time of job that completes last)
- how much better? Coffman and Garey:
 - 2 processors:
makespan(non-preemptive) $\leq 4/3$ *
makespan(preemptive)

- “Inconsistencies” due to precedence relations
 - a release time given for a job may be later than that of its predecessor
 - a deadline may be earlier than of its successor time

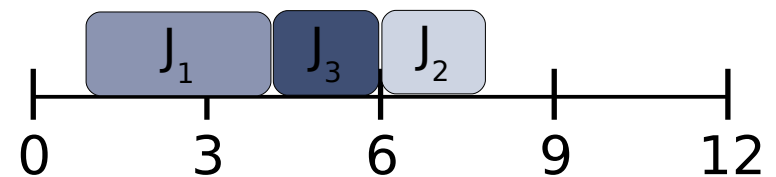
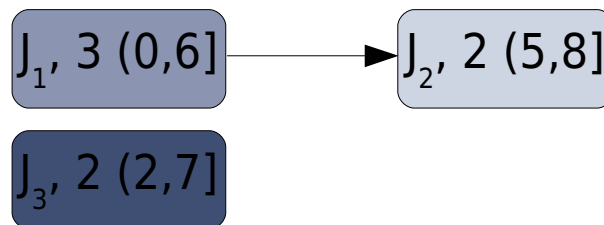
- Effective Release Time:
 - of a job without predecessors: the given release time
 - of a job with predecessors:
max (given release time,
effective release times of all predecessors)
- Effective Deadline:
 - of a job without successor: the given deadline
 - of a job with successor:
min (given deadline,
effective deadlines of all successors)

- Assign priorities at run time ...
 - “the earlier the deadline the higher the priority”
- Theorem:
 - One processor.
 - Jobs preemptable.
 - Jobs do not contend for passive resources.
 - Jobs have arbitrary deadlines, release times.
 - Then: EDF is “optimal”, i.e.
if there is a feasible schedule,
there is also one with EDF

- Proof: (informal)
 - assume a feasible, non EDF schedule
 - systematically transform it to an EDF schedule (3 steps)



- Rationale:
 - no need to complete rt-jobs before deadline
 - use time for other activities
- Idea:
 - Backwards Scheduling
 - Run as late as possible
 - Use latest possible release times as „priority“
 - optimal (analog EDF-Definiton of Optimality)
- Example (Precedence Graph):

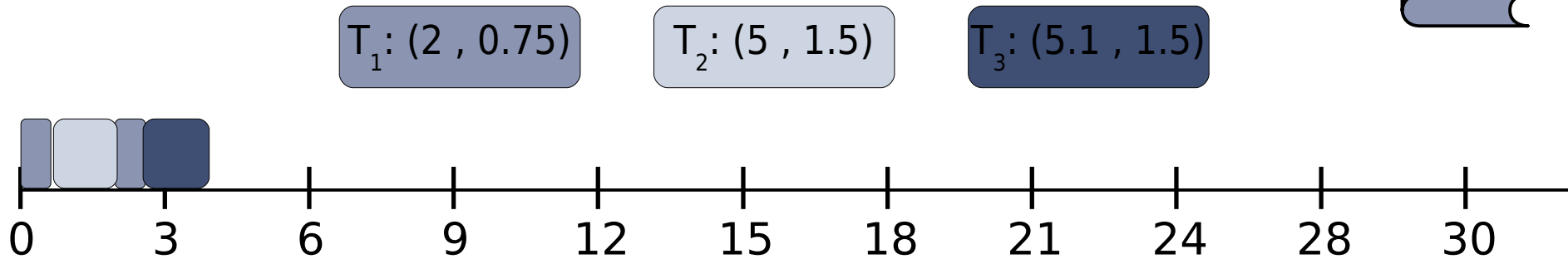


- Slack Time = Laxity:
 - (time to deadline - remaining execution time required to reach deadline)
- slack time: $d - x - t$
 - x remaining execution time of a job
 - d absolute deadline
 - t current time
 - dynamic per job, dynamic at task level (see example)
- also optimal (analog EDF definition)

- two versions:
 - Strict: slacks are computed at all times
 - ***Each instruction (prohibitively slow)***
 - ***Each timer “tick”***
 - Non-strict: slacks computed only at events (release and completion)
- scheduler checks slacks of all ready jobs and reorders queue

Non-Strict LST Example

Jane Liu



- $t=0$ all jobs are released

- T_1, J_1 : **1.25** T_2, J_1 : **3.5** T_3, J_1 :
3.6

- d.h. T_2, J_1 higher priority than T_3, J_1

- $t=2$ T_1, J_2 released

- T_1, J_2 : **1.25** T_2, J_1 : **2.75** T_3, J_1 : **1.6**

- d.h. T_2, J_1 lower priority than T_3, J_1

- $t=2.75$ T_1, J_2 completed

- T_1, J_2 : T_2, J_1 : **2** T_3, J_1 : **0.85**

If no new tasks arrive:

static vs. dynamic priorities

- Task static: Task T, i.e. jobs of T do not change their priorities
- Job static: Jobs do not change their priorities
- Job dynamic: Jobs change their priorities

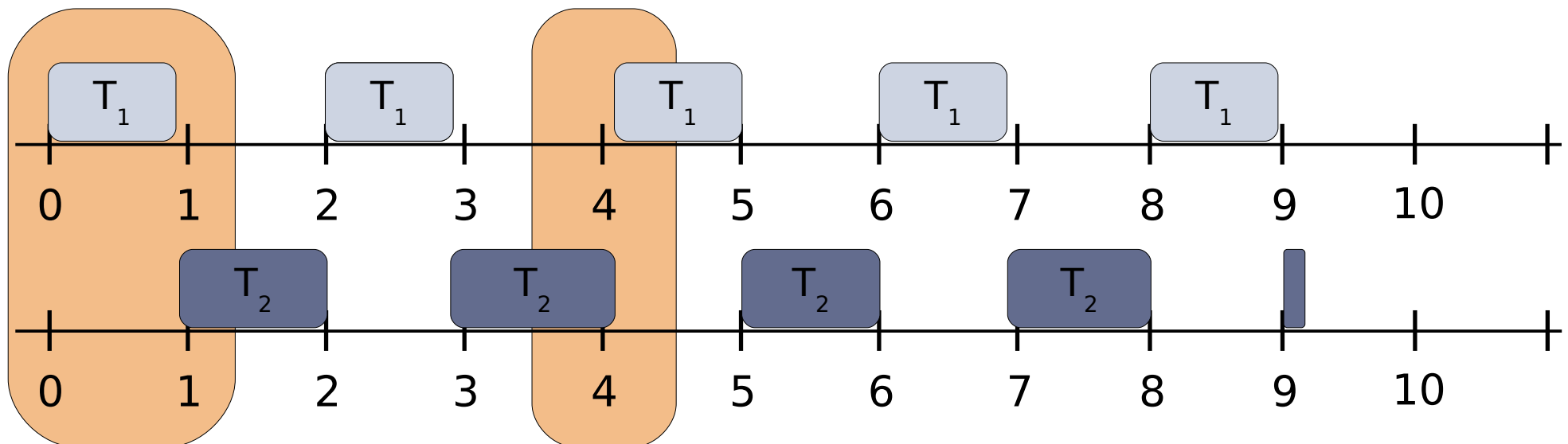
Careful:

Job static is often called dynamic as well

Earliest Deadline First, priority assignment:

Jane Liu

- fixed per job, dynamic at task level:
 - the nearer the absolute deadline of a job at release time
 - the higher the priority
 - $T_1: (2, 0.9)$ $T_2: (5, 2.3)$



EDF and Non - Preemptivity

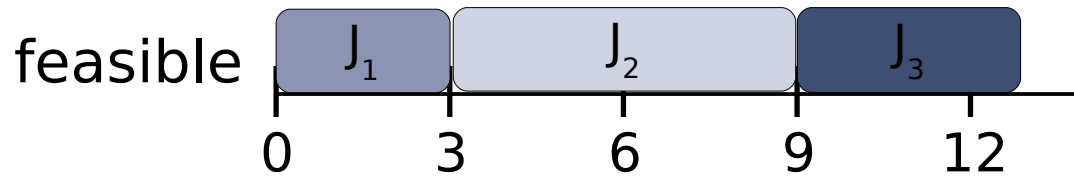
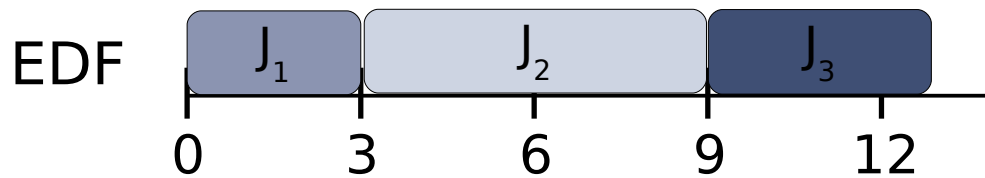
- Job: (release time, execution time, deadline)

$J_1: (0,3,10)$

$J_2: (2,6,14)$

$J_3: (4,4,12)$

release time J_3 J_3 missed Deadline



- EDF is not optimal if jobs are not preemptable

EDF and Multiple Processors

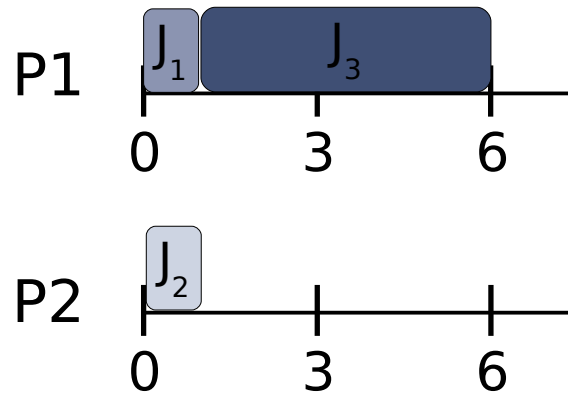
- Job: (release time, execution time, deadline)

$J_1: (0,1,2)$

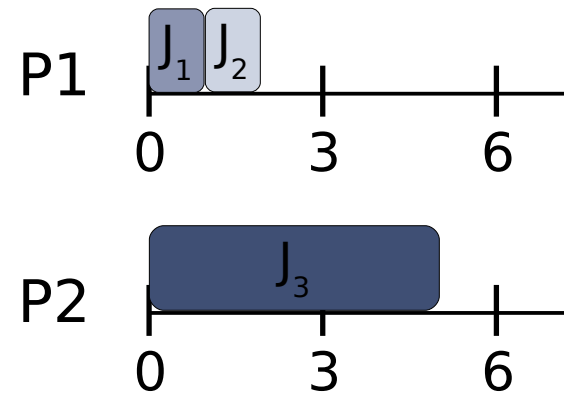
$J_2: (0,1,2)$

$J_3: (0,5,5)$

↓ J_3 missed Deadline



EDF



feasible

- easy for time driven schedulers
- EDF is not optimal for Multiprocessors

- EDF not optimal
general: “static-job” scheduling not optimal
- There are optimal “dynamic-job” schedulers

More later in this course (including references)

Scheduling Anomaly

Jane Liu

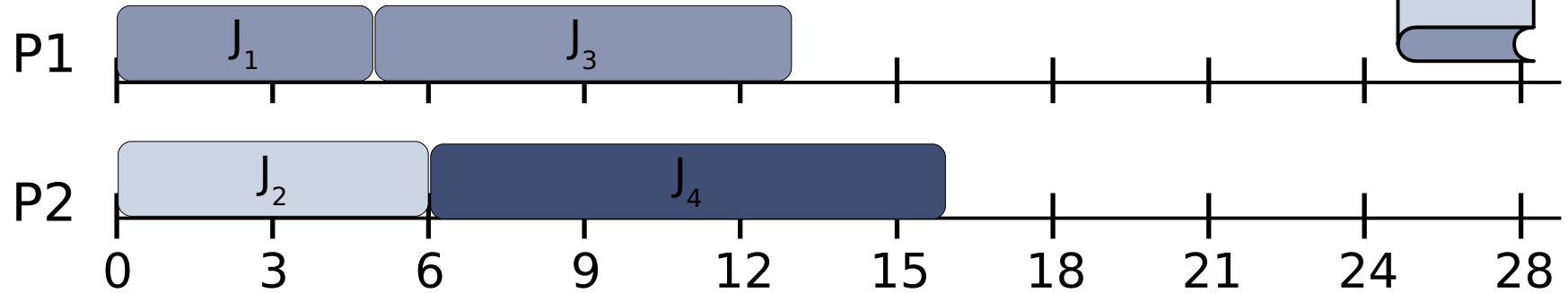
	release	deadline	execution
J_1	0	10	5
J_2	0	10	[2,6] varies
J_3	4	15	8
J_4	0	20	10

- increasing priorities:
 - $i < k \Rightarrow \text{Prio}(J_i)$ higher than $\text{Prio}(J_k)$
- 2 processors, preemptable but not migratable
- intuitive approach:
 - check for worst case(a) and best case(b) execution times and be confident ...

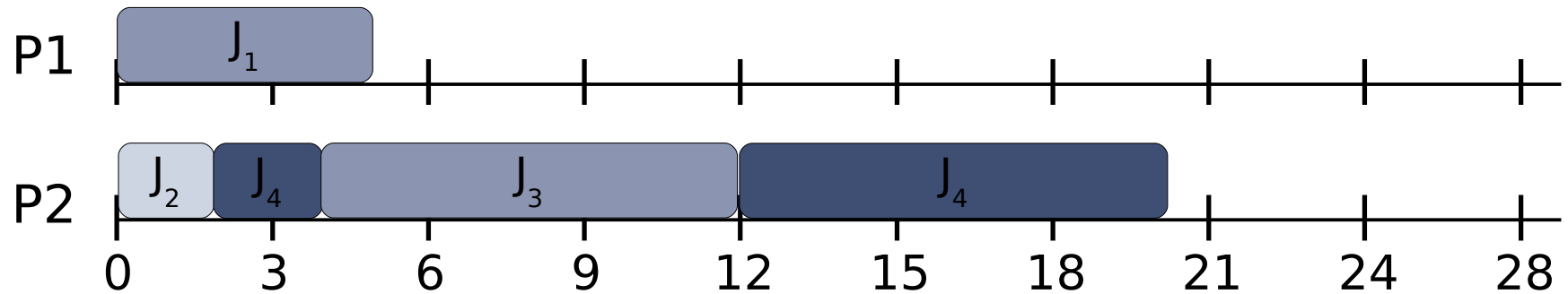
Scheduling Anomaly, cont

Jane Liu

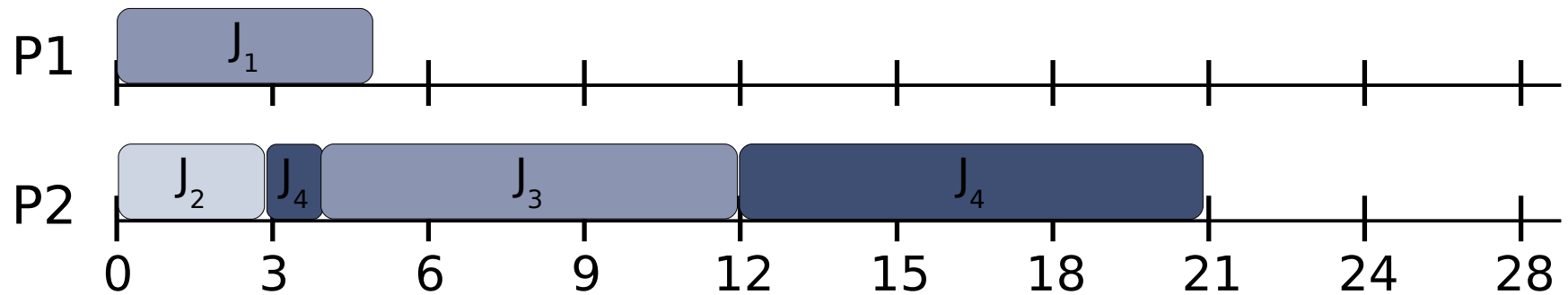
a)



b)



c)



Scheduling Anomaly on One Processor

Jane Liu

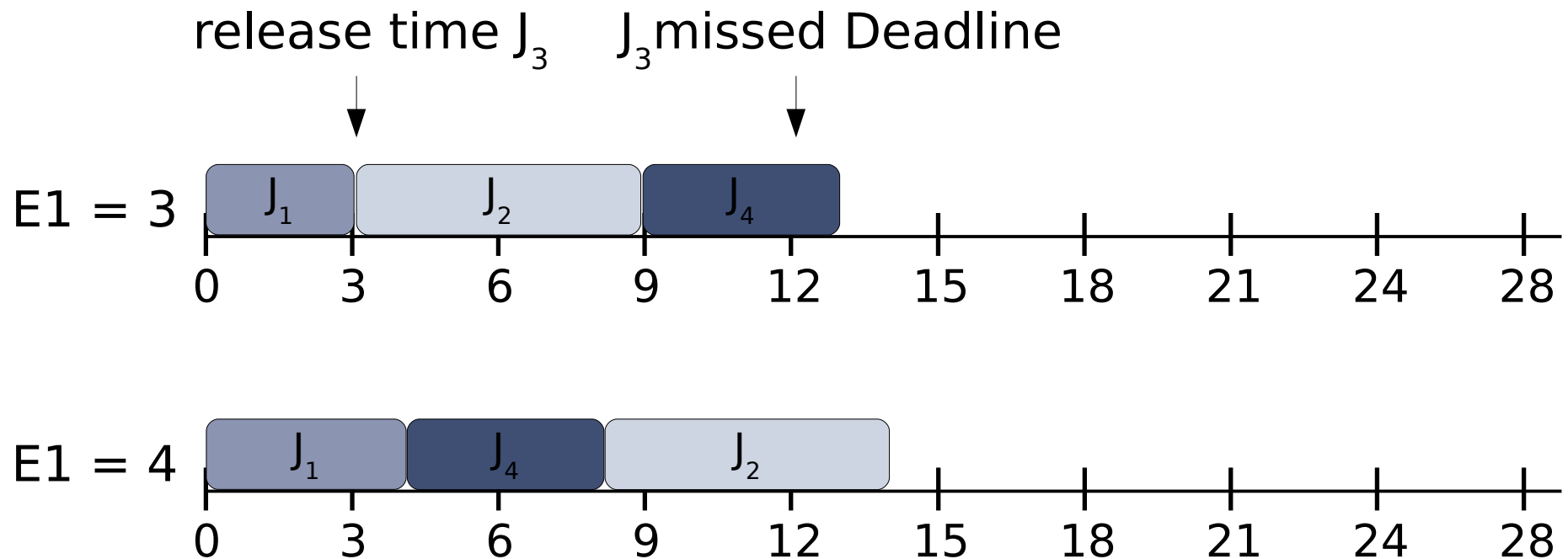
- Job: (release time, execution time, deadline)

$J_1: (0, 3-4, 10)$

$J_2: (2, 6, 14)$

$J_3: (4, 4, 12)$

- Not preemptable



- Informal definition:
 - Given a set of periodic tasks with known minimal and maximal execution times and a scheduling algorithm.

A schedule produced by the scheduler when the execution time of each job has its maximum (**minimum**) value is called a *maximum (minimum) schedule*.

An execution is called *predictable*, if for each actual schedule the start and completion times for each job are bound to be those of the *minimum and maximal schedules*.

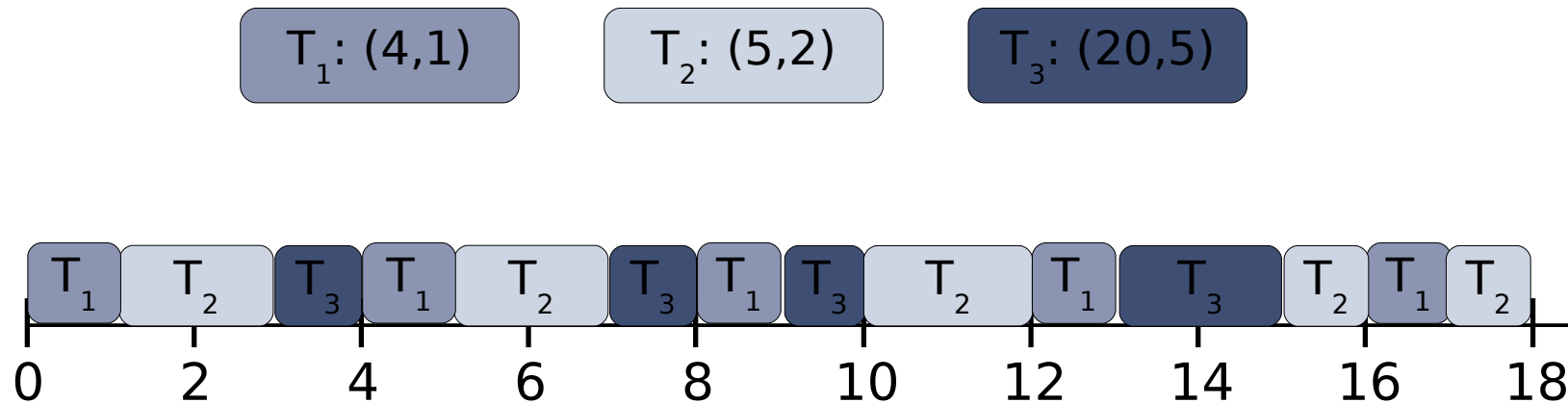
The execution of every job in a set of independent, preemptable jobs with fixed release times is predictable when scheduled in a priority driven manner on one processor.

- Periodic set of tasks with these properties:
 - Tasks are independent
 - one processor
 - no aperiodic tasks
 - preemptable, context switch is negligibly small
 - (period = minimum inter-release times (not fixed))
- Since tasks are independent, tasks can be added (if admitted) and deleted at any time without causing deadline misses.

Rate Monotonic Scheduling

Jane Liu

- fixed priority:
 - the shorter the period the higher the priority (rate: inverse of period)



Deadline Monotonic Scheduling

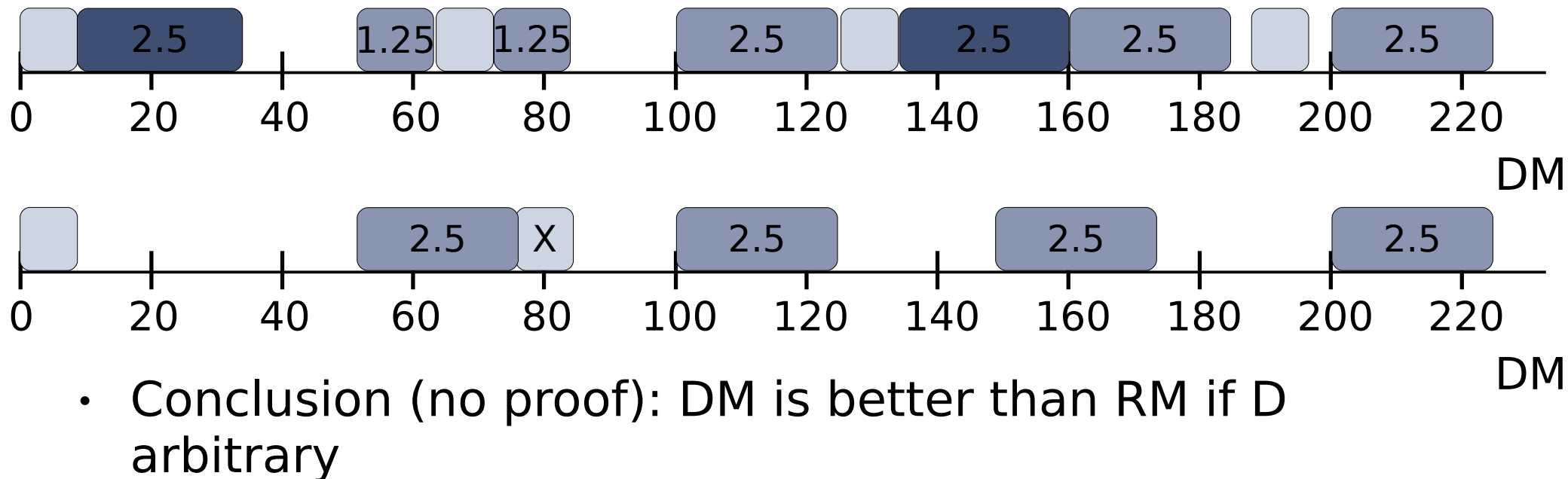
Jane Liu

- fixed priority:
 - the shorter the relative deadline the higher the priority
- example: (ϕ, P, e, D)

$T_1: (50, 50, 25, 100)$

$T_2: (0, 62.5, 10, 20)$

$T_3: (0, 125, 25, 50)$



- Conclusion (no proof): DM is better than RM if D arbitrary

- To do:
 - admission (required before new tasks are admitted)
 - priority assignment (off line / on line)
 - selection of next task (on line)
- restrictions (whether they apply or not)
 - dependencies (precedence, sharing)
 - multiple processors
 - aperiodic, sporadic
- achievable resource utilization: $U = e/p$

EDF and Multiple Processors

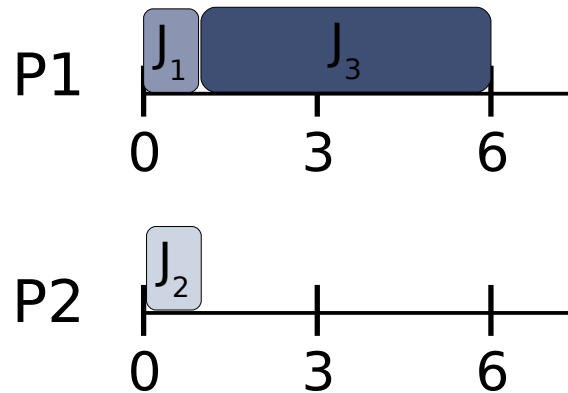
- Job: (release time, execution time, deadline)

$J_1: (0,1,2)$

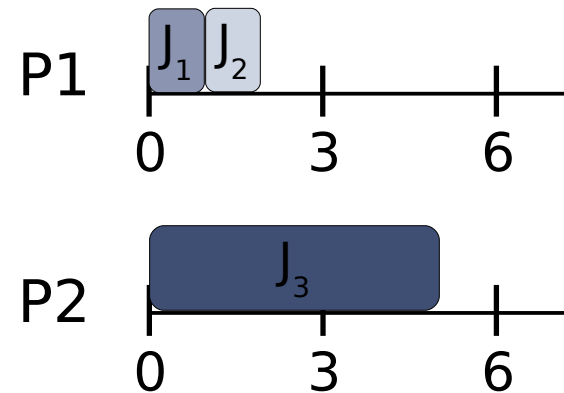
$J_2: (0,1,2)$

$J_3: (0,5,5)$

↓ J_3 missed Deadline



EDF



feasible

- easy for time driven schedulers
- EDF is not optimal for Multiprocessors

Another Multiprocessor Example (Dhall 78)

Jane Liu

m processors, $m+1$ tasks

> 0 , $m*2 < 1$, small

- $T_i, i=1..m$: Period 1, execution time: 2
- T_{m+1} : Period $1+$, execution time: 1
- scheduler: priority (edf)
- allocation: dynamic
- *discuss !*

Pathological case.

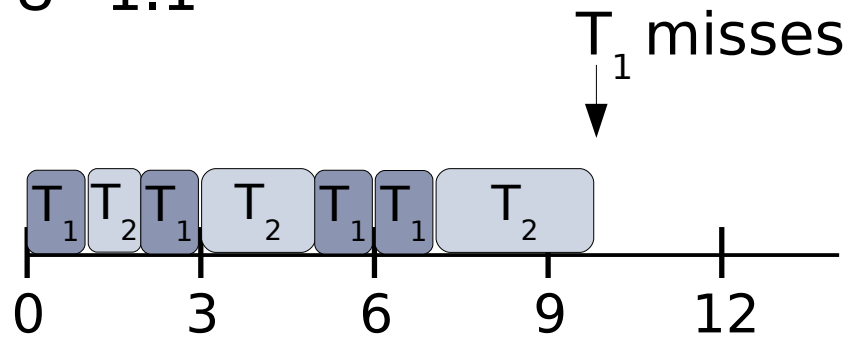
EDF and Overload, examples

Jane Liu

$T_1: (2, 1)$

$T_2: (5, 3)$

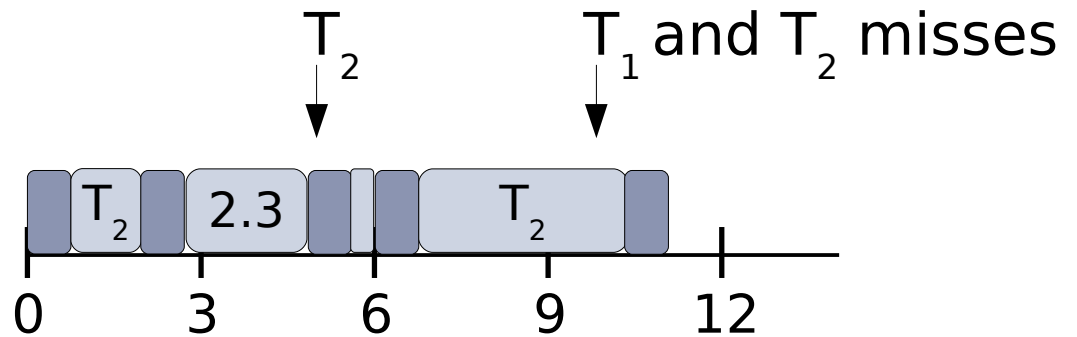
$U=1.1$



$T_1: (2, 0.8)$

$T_2: (5, 3.5)$

$U=1.1$



- No easy way to determine which jobs miss deadline ...

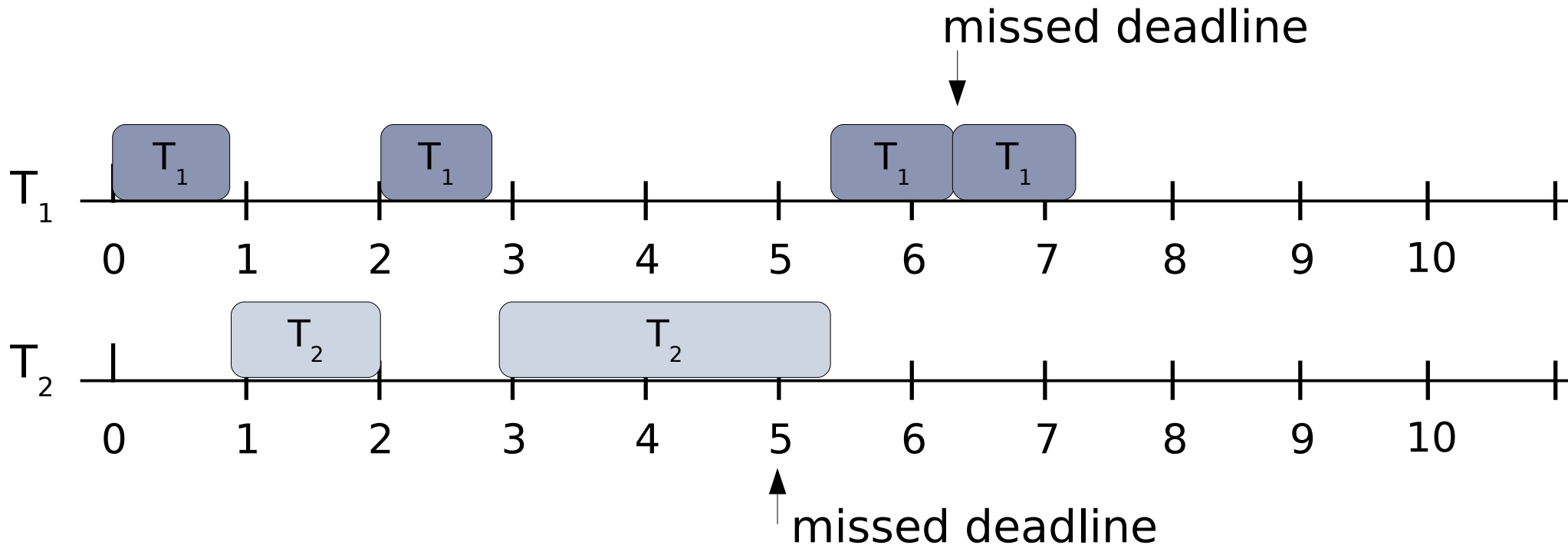
EDF and Overload, one more example

Jane Liu

$T_1: (2, 0.8)$

$T_2: (5, 4.0)$

$U=1.2$



- $J_{2,1}$ continues to execute after deadline and ...
... causes $J_{1,3}$ to miss the deadline

- - in fixed priority systems it is possible to predict which tasks are affected due to overruns

Optimality of Fixed Priority Schedulers

Jane Liu

- T: periodic tasks, independent, preemptable, one proc.
- Deadline Monotonic:
 - relative deadlines \leq periods, in phase
if there is any feasible fixed priority schedule for T,
then Deadline Monotonic is feasible as well
- Rate Monotonic (RMS):
 - relative deadlines = periods
if there is any feasible fixed priority schedule for T,
then Rate Monotonic is feasible as well
simply periodic, i.e.
for all pairs of tasks i, j : if $P_i \leq P_j$ holds $P_j = n * P_i$

A task (P, e) requires e/P of the capacity of a processor

Any scheduler can admit at most up to full capacity:

- For a task set $T_1 .. T_n$: $\sum e_i/p_i \leq 1$ is a necessary but not sufficient condition

Can we establish a bound X such that

- $T_1 .. T_n$: $\sum e_i/p_i \leq X$

is sufficient?

Such bounds are called Schedulability Utilisation SU.

- SU depends on the Scheduling algorithm.
- SU: the higher the better.

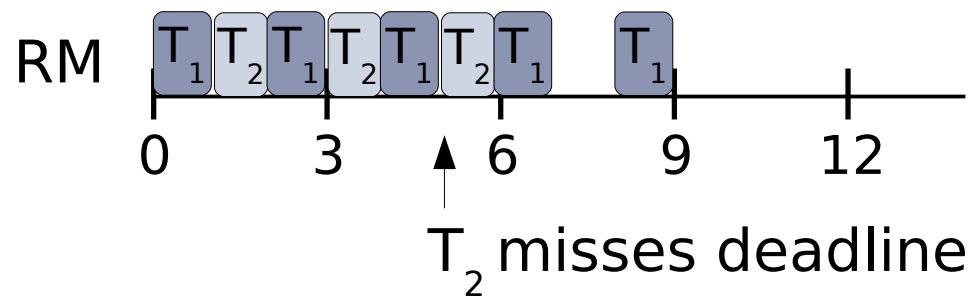
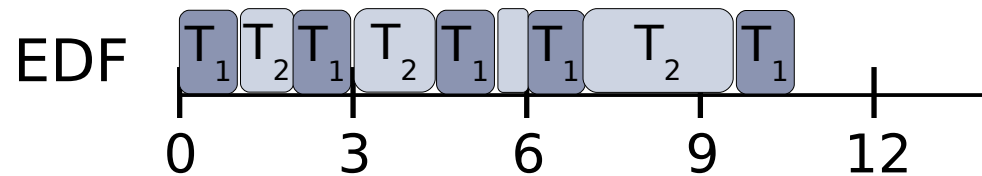
Utilization: RMS ./. EDF

Jane Liu

$T_1: (2, 1)$

$T_2: (5, 2.5)$

$U=1$



- RMS not optimal in general

Some Schedulable Utilization(SU) Results

Jane Liu

- indep. tasks, preemptable, relative deadline=period, 1 processor
- N ... Number of Tasks
- EDF: $SU = 1$
- RMS: $SU = n (2^{1/n} - 1)$ $n \ln(2)$
- RMS (simply periodical, D = P): $SU = 1$

Schedulibility Test for Fixed(!) Priority

Jane Liu

- (case where jobs must complete before end of period)
- Critical Instant Analysis / Time Demand Analysis
- critical instant for task T_i :
 - one of the jobs of T_i is released at same time with a job in every higher priority task ...
- It is sufficient to check a schedule for the critical instant for the longest involved period

(Fixed Prio) Schedulability and Blocking

Jane Liu

- T_i may have to wait for non-preemptable, lower priority task
- b_i : longest non-preemptable portion of all lower prio. Jobs
- Schedulability for all tasks T_i with fixed priority scheduler x $SU_x(i)$:
 - Schedulable Utilisation for scheduling method x with i tasks:
 - $U_i = e_1/p_1 + e_2/p_2 \dots e_i/p_i$
 - $U_i + b_i/p_i \leq SU_x(i)$

Non Negligible Context Switch Time

Jane Liu

- For Job level fixed priority schedulers ... :
 - i.e. each job preempts at most one other job
- 2 context switches:
 - release (when it preempts other)
 - completion
- include CS overhead in wcet:
 - $WCET_i := WCET_{i_original} + 2CS$

(Fixed Prio and) Limited Priority Levels

Jane Liu

- Required: Mapping of
 - Scheduling-Priorities: 1 ... n to
 - Operating System Priorities: $\Pi_1, \Pi_2, \dots, \Pi_m$
- Jobs running with same OS-Prio but different Sched-Prio use:
 - FIFO, Round Robin, ...
- Schedulibility loss ?
 - Notation: Π_i as grid on Scheduling Priorities
 - Example:
 - 10 scheduling priorities, 3 OS priorities
 - possible mapping: $\Pi_1=3, \Pi_2=8, \Pi_3=10$
 - Interpretation: 0,1,2,3 mapped to Π_1 , 4,5,6,7,8 to Π_2 , 9,10 to Π_3
- How is Schedulibility Test affected?

(Fixed Prio and) Limited Priority Levels

Jane Liu

- Mappings:
 - uniformly distributed:
 $k = n/m$
Scheduling Priority X mapped to $\lfloor X/m \rfloor * k$
 - constant ratio:
keep $(\tau_{i-1} + 1) / \tau_i$ as equal as possible

- Rate Monotonic, large n ...
 - $g = \min(\Pi_{i-1} + 1) / \Pi_i$
- $SU_{RM} = \ln(2g) + 1 - g$
- relative schedulibility(rs): relation to $\ln(2)$
- Example:
 - $n = 100000, m = 256$
 - $rs = 0.9986$

=> 256 priorities is it !

Lessons Learned

- Schedulers: static, static and dynamic (RMS, EDF, LST)
- Schedulibility Analysis:
Schedulibility Utilization and Critical Instant
- RMS and EDF are “optimal” under simplistic assumptions
- “Anomalies”
-