Event-Driven Scheduling

(closely following Jane Liu´s Book)

Principles

Admission: Assign priorities to Jobs

At events, jobs are scheduled according to their "priorities"

Important properties:

- decisions, which job to execute next at events (not time instants) such 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 not possible on line)

Restrictions Given Up

some “restrictive” assumptions of time-driven systems are given up:

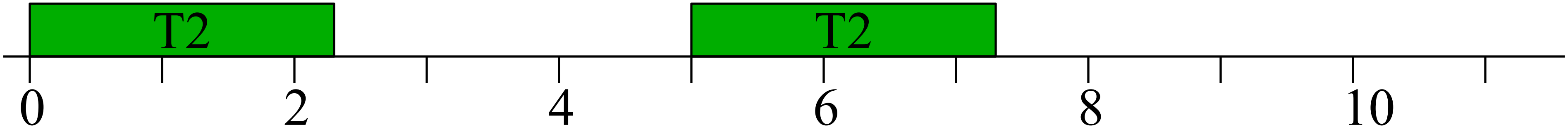
- fixed inter-release times
  - → minimum inter-release times
- fixed number of rt tasks in systems
  - real-time <u>and</u> non real-time, number can vary
- a priori fairly well known parameters
  - tasks come and go, overloading, ...

Priority Assignment Following "Criticality"

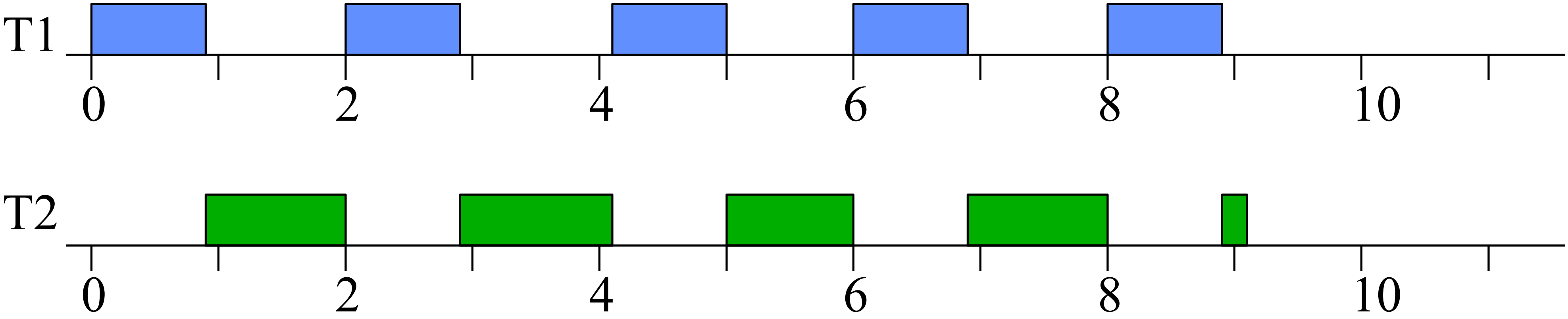
The more critical a task the higher its priority

T1: (2,0.9)   T2: (5,2.3)

T2 more critical than T1



T1 misses deadline in Job 1 and 2/3, unnecessarily ...



Important Variants

- **Static vs dynamic allocation to processors**
  - static:
    jobs are assigned to processors once and stay there
  - dynamic:
    one queue served by all processors (jobs “migrate”)
- **static vs. dynamic priorities**
  - static: jobs do not change their priorities (unless new tasks arrive)
  - dynamic: priorities are recomputed frequently

e.g., FIFO is dynamic priority scheduling

- **preemptive or non preemptive**
  - some tasks
  - all tasks

Preemptive vs. Non-Preemptive Scheduling, Example

2 processors,

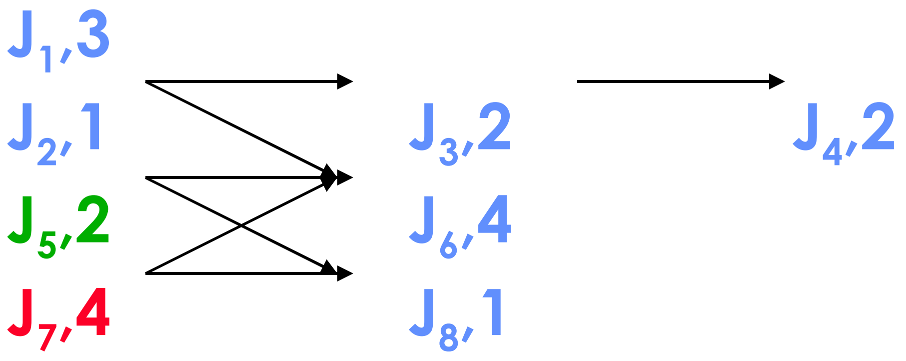
Tasks: Notation used below: J_i ,e_i

release time of J_5 is 4, all others 0; (!)

static priorities, assigned such that:
i < k => Prio(J_i) higher than Prio(J_k)

Tasks can “migrate”

precedence graph:



J_1,3

J_2,1

J_5,2

J_7,4

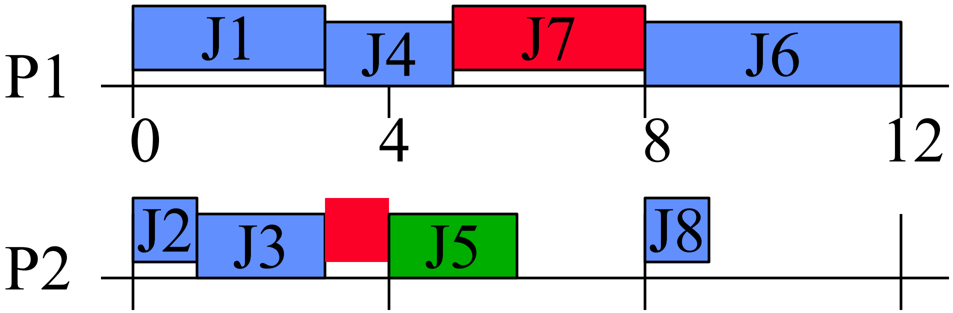
J_3,2

J_6,4

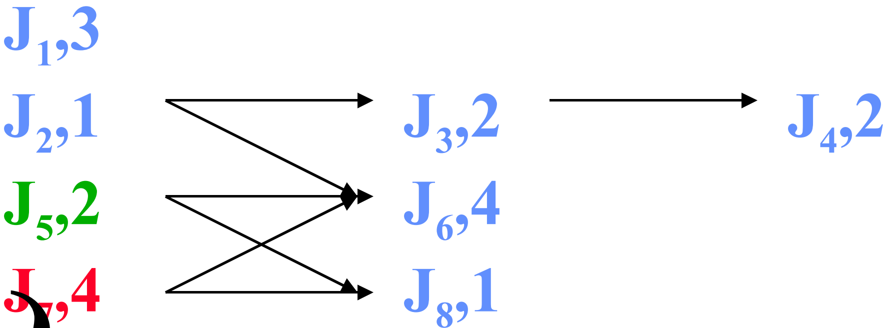
J_8,1

J_4,2

Example, executions



preemptive



J_1,3

J_2,1

J_3,2

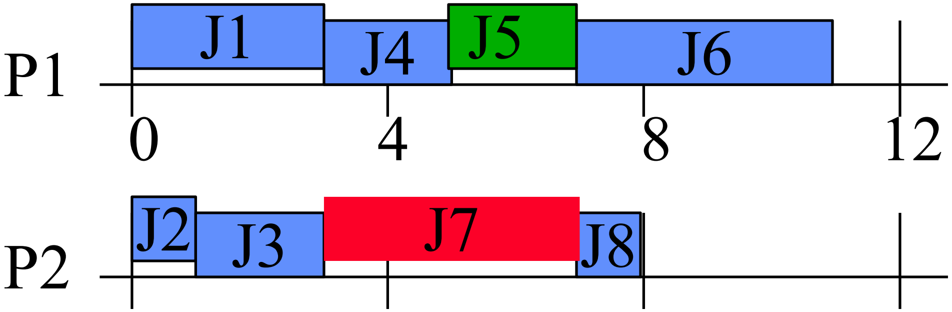
J_4,2

J_5,2

J_6,4

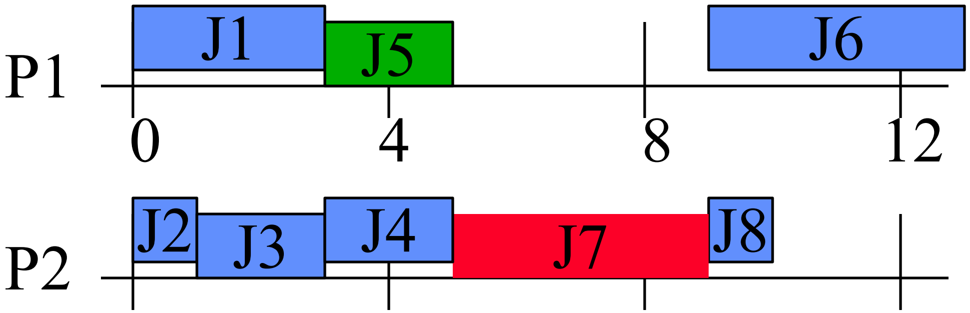
J_7,4

J_8,1

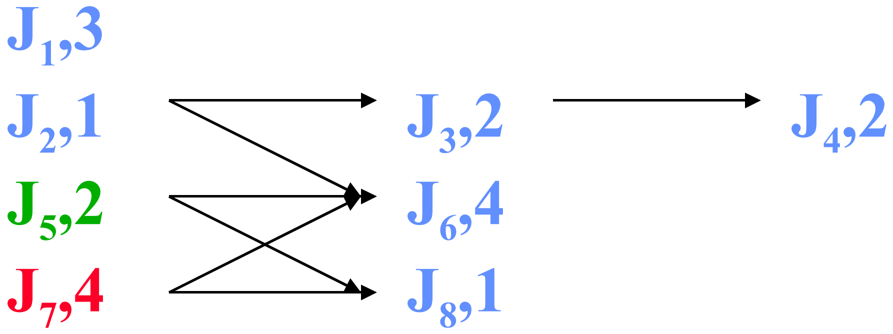


non preemptive

Modified Example: release time of J5 = 0



} non preemptive



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) <= 4/3 * makespan(preemptive)

Effective Release Times and Deadlines

"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

From Now: use effective ...

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)

Earliest Deadline First

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.

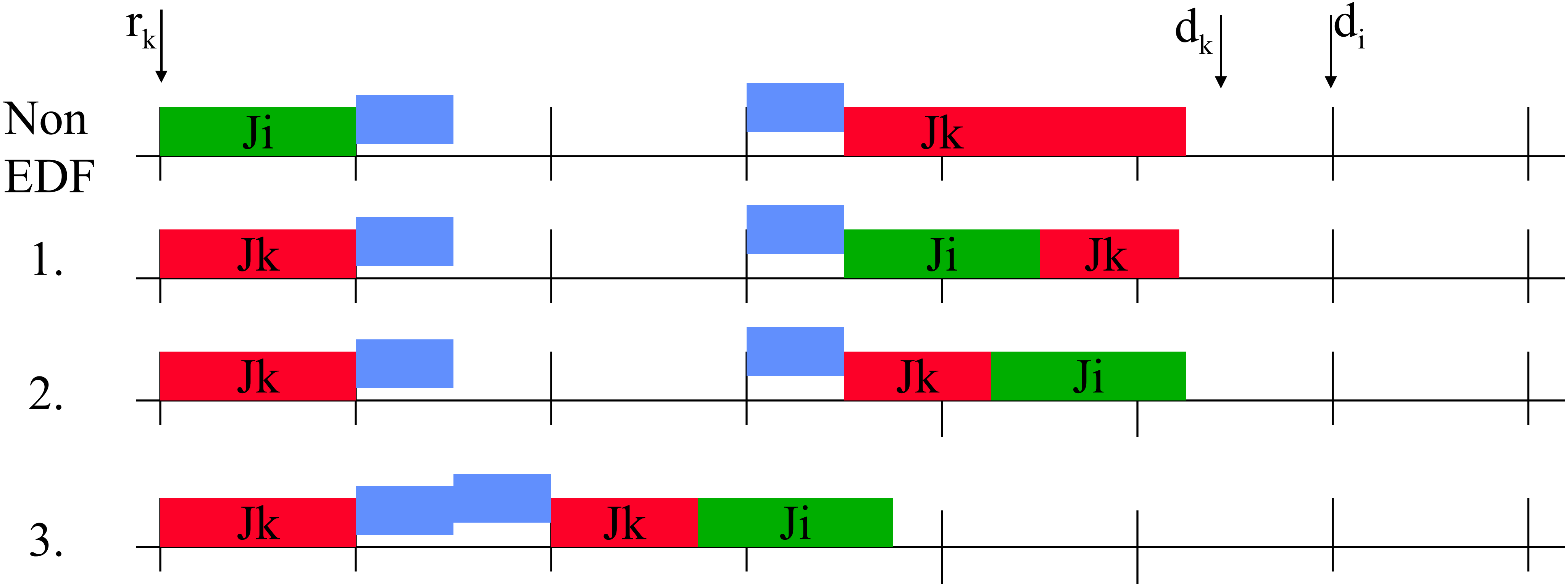
Then: EDF is "optimal", i.e.
if there is a feasible schedule,
there is also one with EDF

EDF Optimality

Proof: (informal)

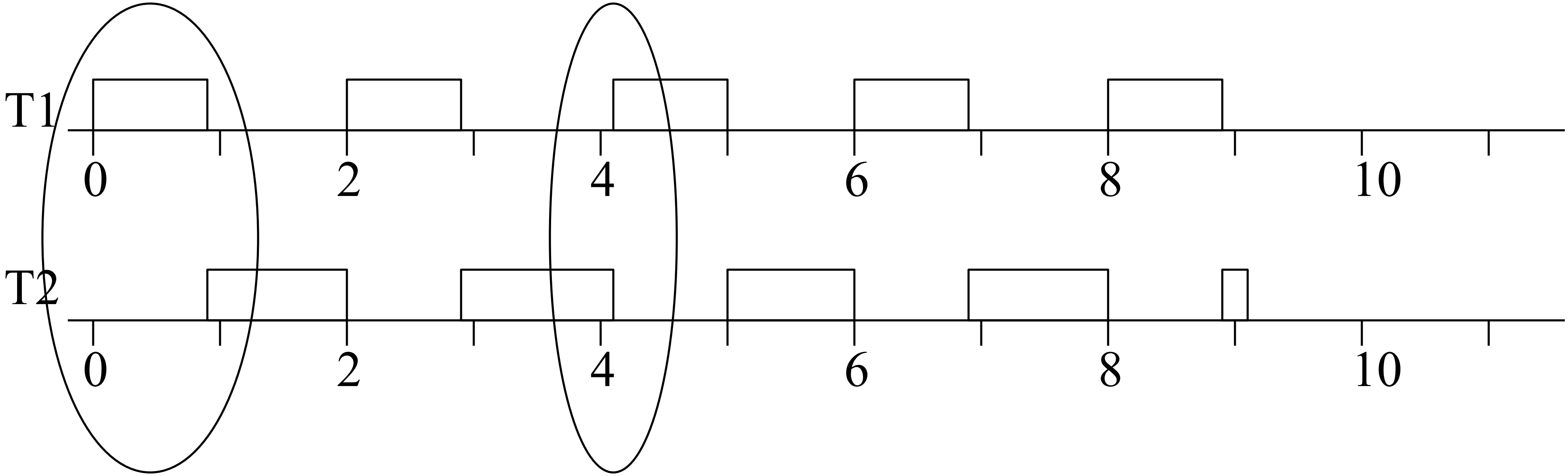
assume a feasible, non EDF schedule

<u>systematically transform it to an EDF schedule (3 steps)</u>



Earliest Deadline First, priority assignment:

fixed per job, dynamic at task level:
the nearer the absolute deadline of a job at release time
the higher the priority
T1: (2,0.9)   T2: (5,2.3)



Latest Release Time (LRT)

Rationale:

no need to complete rt-jobs before deadline
use time für other activities

Idea:

Backwards Scheduling

Run as late as possible

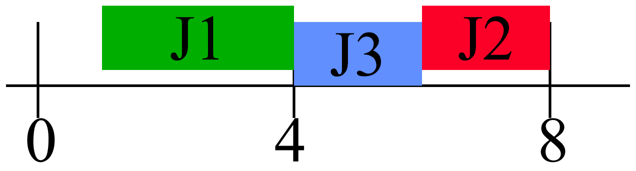
Use latest possible release times as „priority“

optimal (analog EDF-Definiton of Optimality)

Example (Precedence Graph):

J_1,3 (0,6] → J_2,2 (5,8]

J_3,2,(2,7]



Least Slack Time First / Minimum Laxity First

Slack Time = Laxity:
(time to deadline
– remaining time required to reach deadline)

also optimal (analog EDF definition)

Least Slack Time First

dynamic per job, dynamic at task level:

slack time: d - x - t

x remaining execution time of a job
d absolute deadline
t current time

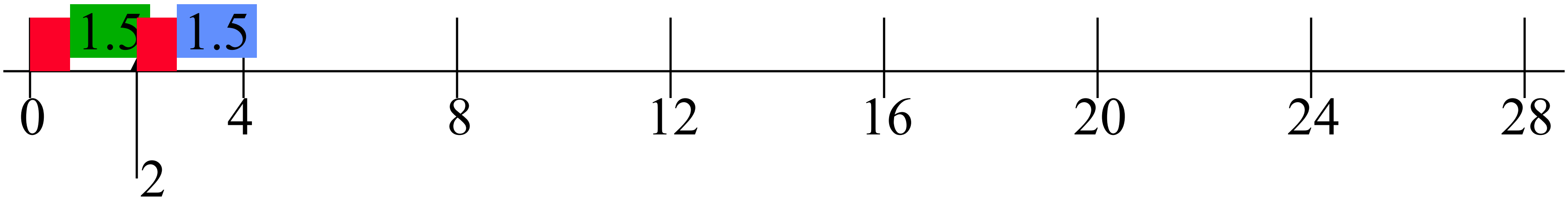
two versions:

- strict:
  slacks are computed at all times (prohibitively slow)
- non-strict:
  slacks computed only at events (release and completion)

scheduler checks slacks of all ready jobs and reorders queue

Non-Strict LST Example

T1: (2,0.75) **T2:(5,1.5)** **T3: (5.1,1.5)**



t=0 all Jobs released
T1,J1:1.25 T2,J1: 3.5 T3,J1: 3.6
d.h. T2,J1 higher priority than T3,J1

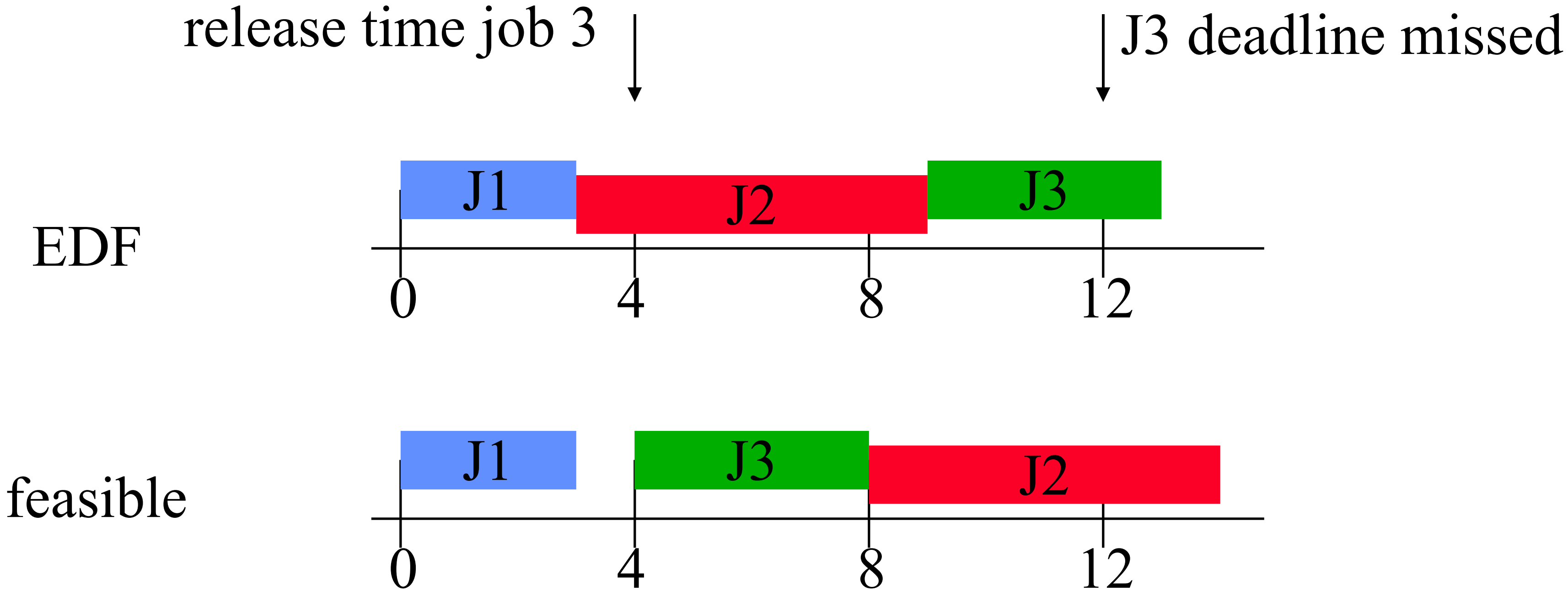
t=2 T1,J2 released
T1,J2:1.25 T2,J1: 2.75 T3,J1: 1.6
d.h. T2,J1 lower priority than T3,J1

t=2.75 T1,J2 completed
T1,J2: T2,J1: 2 T3,J1: 0.85

EDF and Non - Preemptivity

Job: (release time, execution time, deadline)

J1: (0,3,10)   J2: (2,6,14)   J3: (4,4,12)

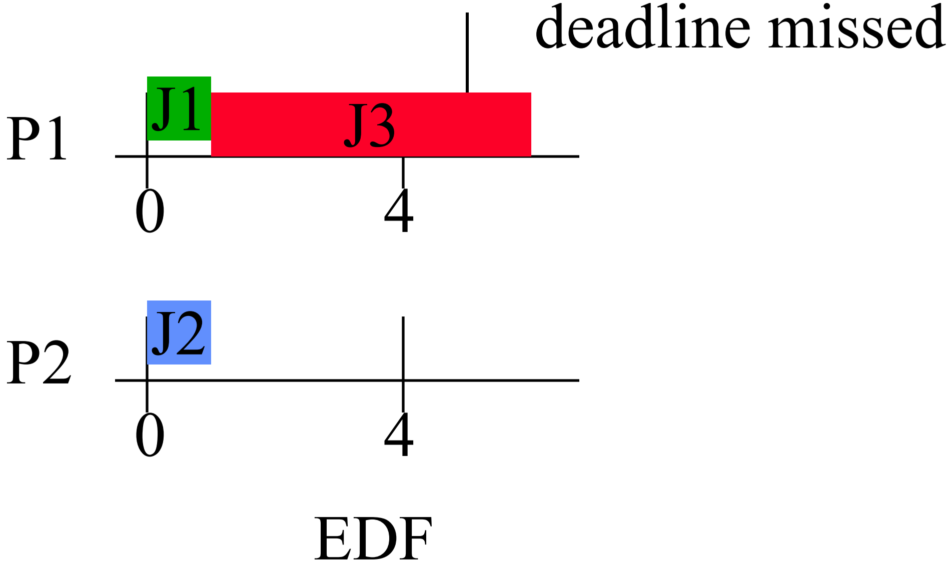


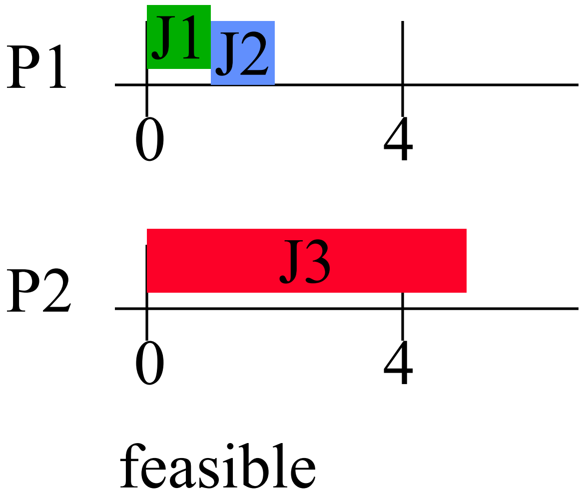
EDF is not optimal if jobs are not preemptable.

EDF and Multiple Processors

Job: (release time, execution time, deadline)

J1: (0,1,2) J2: (0,1,2) J3: (0,5,5)





easy for time driven schedulers

EDF is not optimal for Multiprocessors.

Scheduling Anomaly

	release	deadline	execution	
J1:	0	10	5	
J2:	0	10	[2,6]	varies
J3:	4	15	8	
J4:	0	20	10	

increasing priorities:

$i < k \Rightarrow$ Prio(J_i) higher than Prio(J_k)

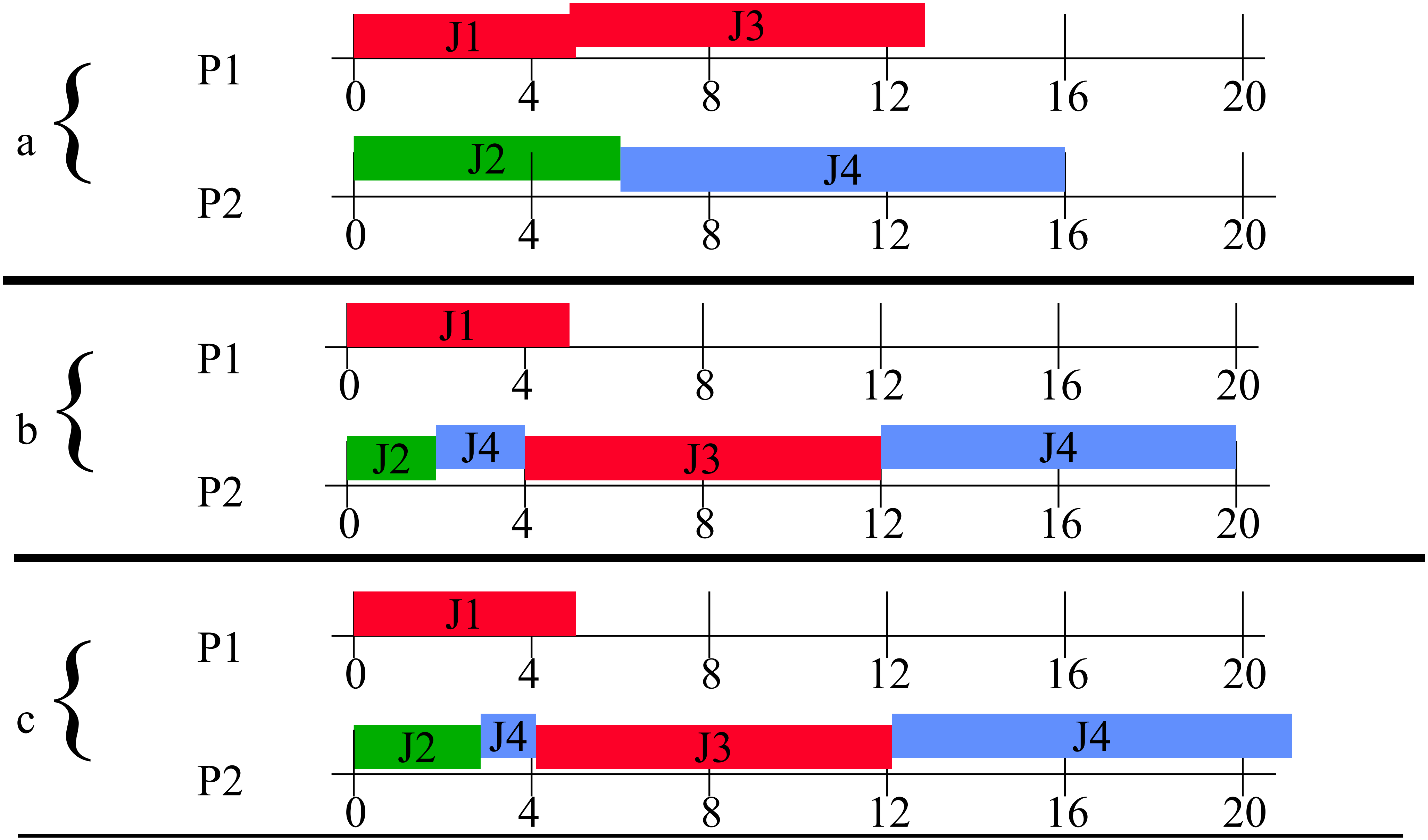
2 processors, preemptable but not migratable

intuitive approach:

check for worst case(a) and best case(b) execution times

and be confident ...

Scheduling Anomaly, cont

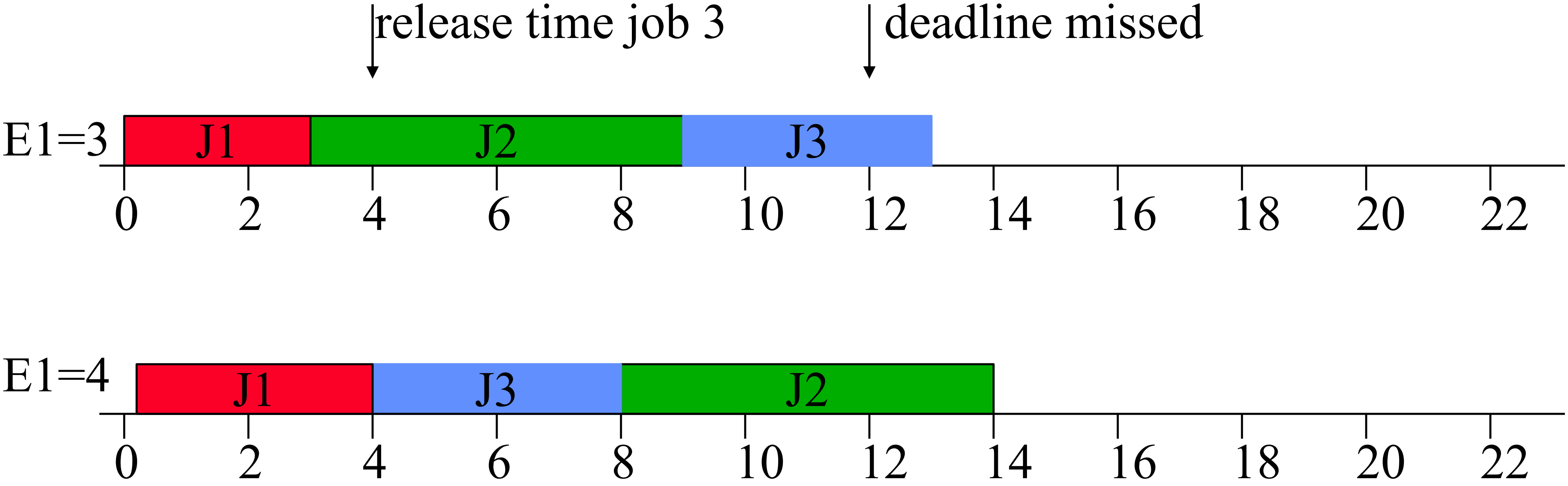


Scheduling Anomaly on One Processor

Job: (release time, execution time, deadline)

J1: (0,3-4,10)   J2: (2,6,14)   J3: (4,4,12)

Not preemptable



Predictable Execution

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 ist 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 be those of the *minimum and maximal schedules*.

Predictable Execution

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.

Validation Algorithms

... determine whether all jobs meet their deadlines

correct or not

accurate or not
- overly pessimistic
- overly optimistic

Assumptions for Next Set of Algorithms

Periodic set of tasks with these properties:

- Tasks are independent
- one processor
- no aperiodic or sporadic 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.

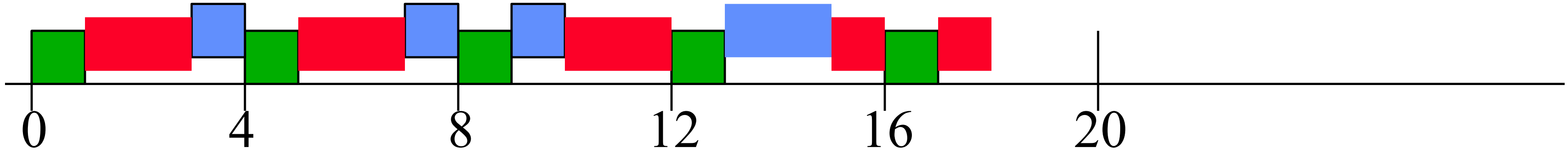
Priority Assignment

- fixed priority:
  fixed for task (and jobs) relativ to other tasks
- dynamic priority:
  priority of tasks changes at release and completion times in relation to other tasks
  - fixed per job
  - dynamic per job

Rate Monotonic Scheduling

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

example: T1: (4,1) T2: (5,2) T3: (20,5)

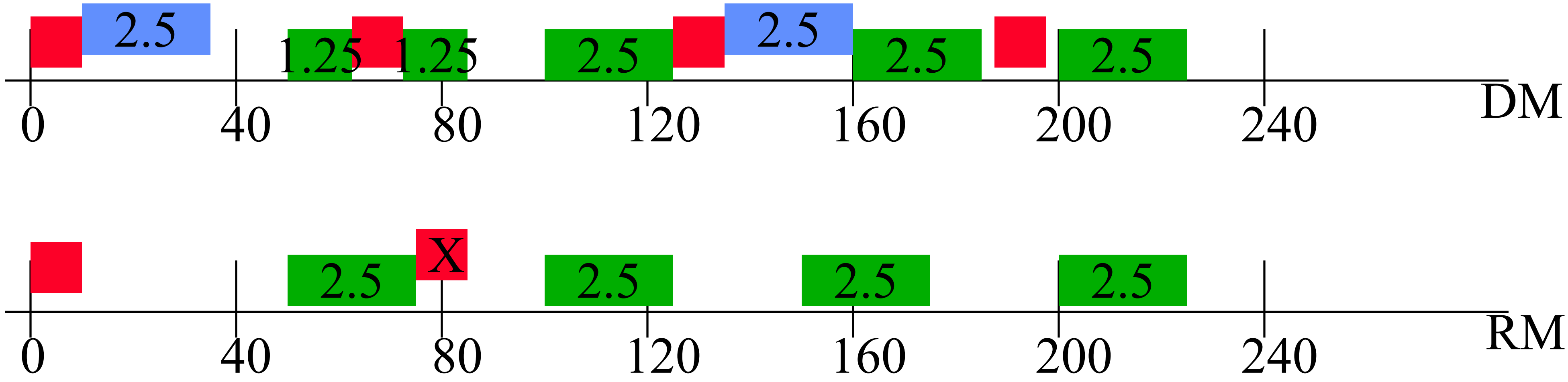


Deadline Monotonic Scheduling

fixed priority:
the shorter the relative deadline the higher the priority

example: (ϕ,P,e,D)

T1: (50,50,25,100) T2: (0,62.5,10,20) T3: (0,125,25,50)



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

(More) Comparison Criteria

- Optimality
- Validation

- Schedulable Utilization(SU) of an algorithm:
  a scheduling algorithm can feasibly schedule any set of periodic tasks on a processor
  if $\varepsilon/p \leq SU$
  SU: the higher the better
  dynamic priority schedulers better than fixed priority

- predictability in the presence of overload:
  in fixed priority systems it is possible to predict which tasks are affected due to overruns

Priority-Driven Scheduling of Periodic Tasks

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)

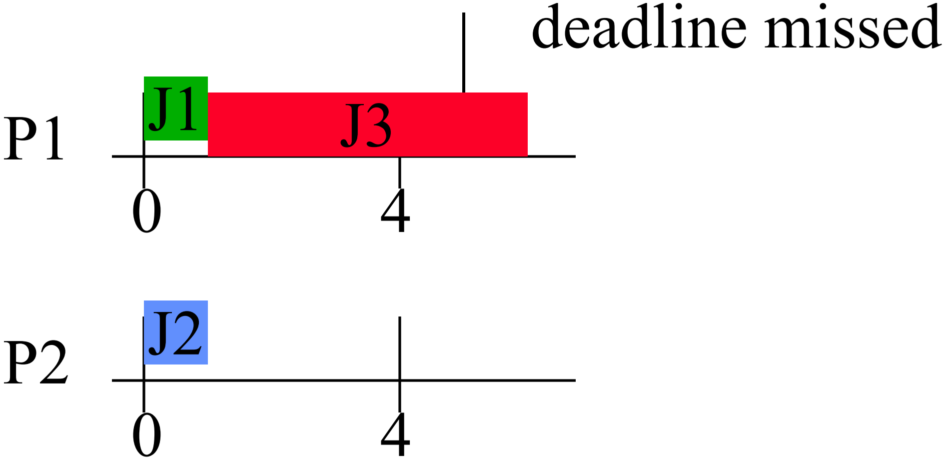
- dependencies (precedence, sharing)
- multiple processors
- aperiodic, sporadic

achievable resource utilization: U=□e/p

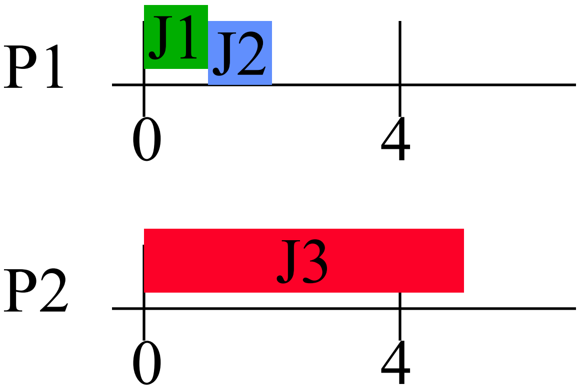
EDF and Multiple Processors

Job: (release time, execution time, deadline)

J1: (0,1,2) J2: (0,1,2) J3: (0,5,5)



EDF



feasible

easy for time driven schedulers

EDF is not optimal for Multiprocessors.

Another Multiprocessor Example

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 or shortest period first)

allocation: dynamic

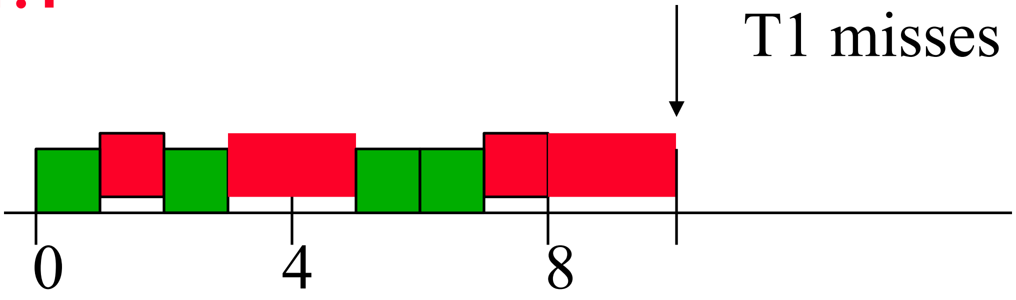
discuss !

Pathological cases, mostly dynamic performs better

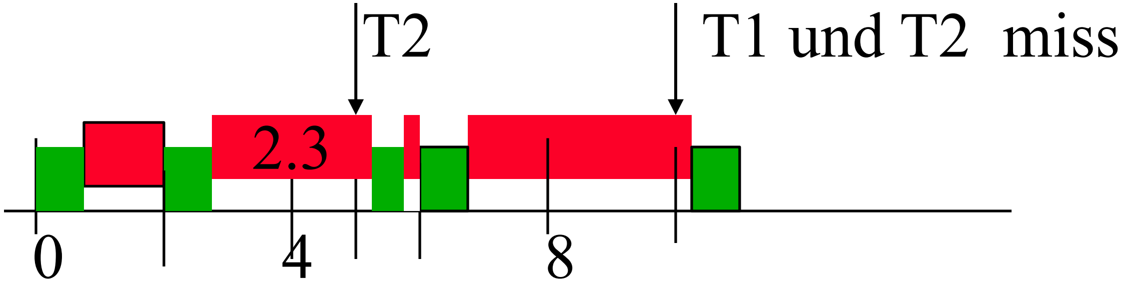
very hard to analyze for worst case

EDF and Overload, examples

T1: (2,1) T2: (5,3) U=1.1



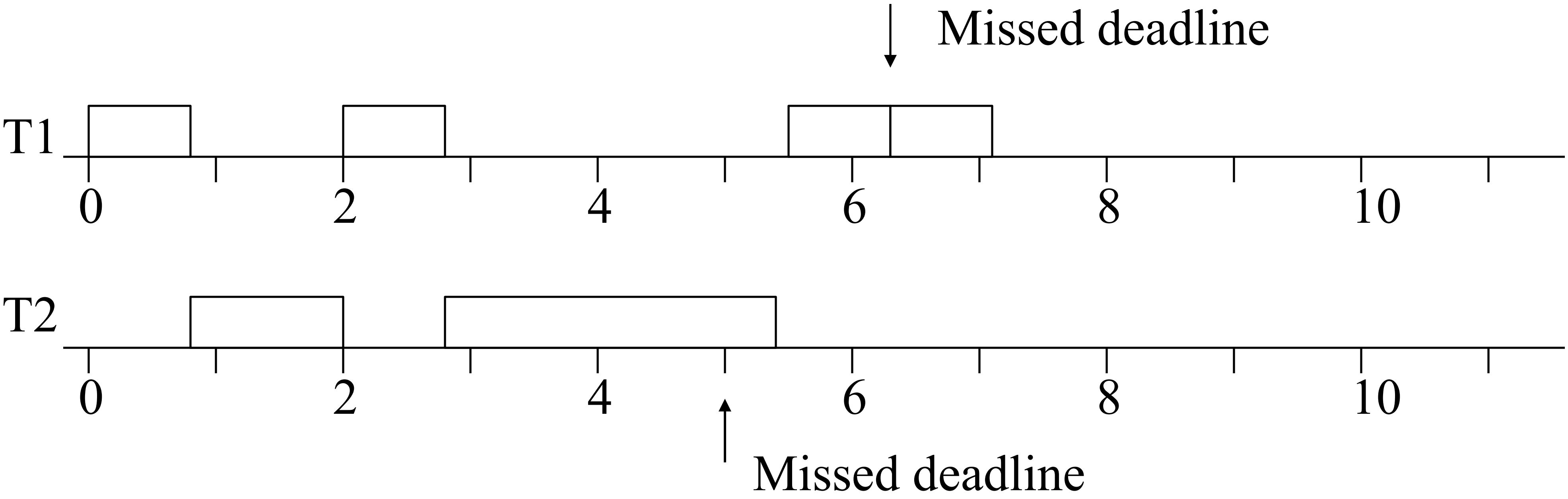
T1: (2, 0.8) T2: (5,3.5) U=1.1



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

EDF and Overload, one more example

T1: (2, 0.8) T2: (5, 4.0) U=1.2

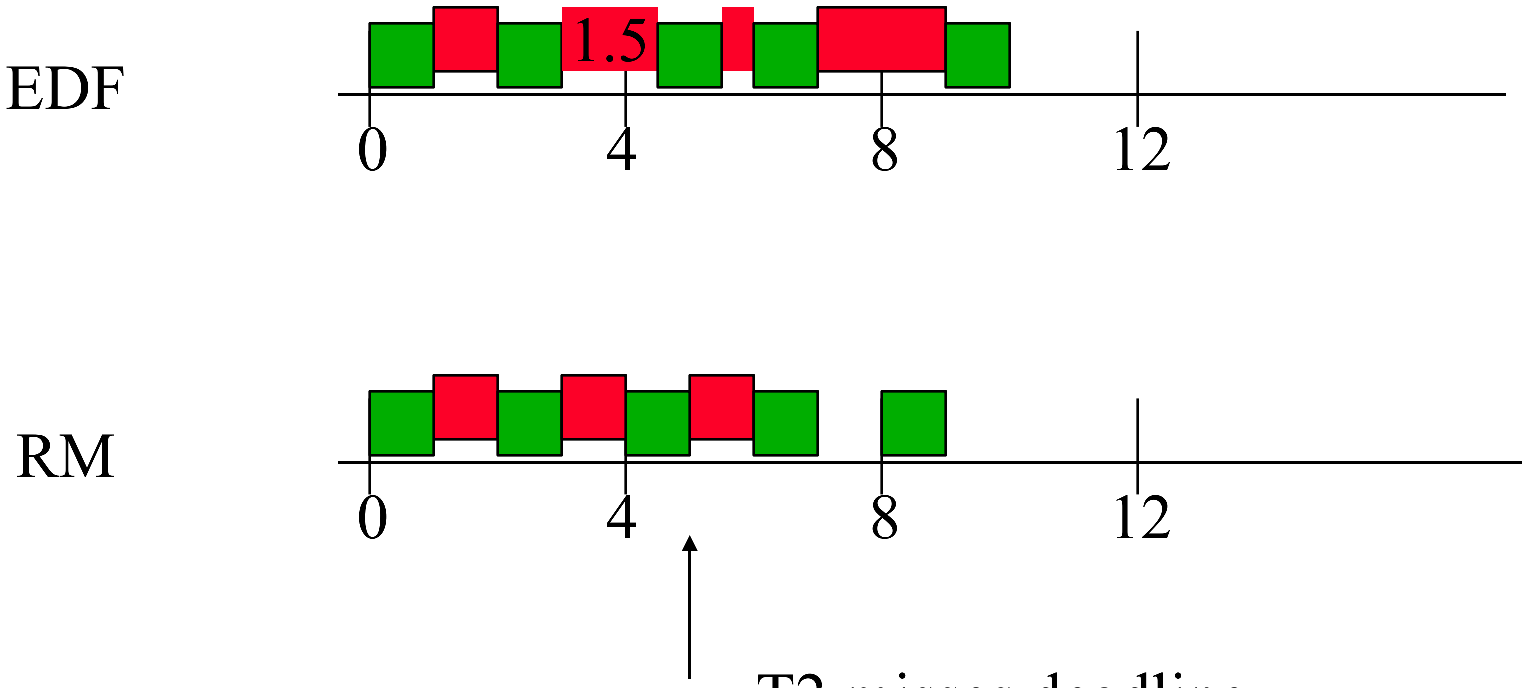


J2,1 continues to execute after deadline and ...
causes J1,3 to miss the deadline

Utilization: RM ./. EDF

T1: (2,1)   **T2: (5,2.5)**

U = 1



T2 misses deadline

RM not optimal in general

Optimality of Fixed Priority Schedulers

T: peridodic tasks, independent, preemptable, one proc.

Deadline Monotonic:
relative deadlines <= periods, in phase
if there is any feasible fixed priority schedule for T,
then Deadline Mononotic is feasible as well

Rate Monotonic:
relative deadlines = periods
simply periodic, i.e.
for all pairs of tasks i,j: if $P_i <= P_j$ holds $P_j = n * P_i$
RM is schedulable iff U <= 1 (cmp. EDF)

Some Schedulable Utilization(SU) Results

indep. tasks,
preemptable,
relative deadline=period,
one processor

N Number of Tasks

EDF: SU = 1

RMS: $SU = n (2^{1/n} - 1)$ n *J : ln(2)

RMS (simply periodical, D □P): SU = 1

Schedulibility Test for Fixed(!) Priority

(case where jobs must complete before end of period)

Critical Instant Analysis / Time Demand Analysis:

critical instant for task Ti:
one of the jobs of Ti 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 envolved period

(Fixed Prio) Schedulibility and Blocking

Ti may have to wait for
non-preemptable, lower priority task

bi:
longest non-preemptable portion of all lower prio. Jobs

Schedulability for all tasks Ti with fixed priority scheduler x
$SU_x(i)$:
Schedulable Utilisation for scheduling method x with i tasks:

$Ui = e1/p1 + e2/p2 \ldots ei/pi$

$Ui + bi/pi <= SU_x(i)$

Non Negligible Context Switch Time

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:

WCETi := WCETi_original + 2CS

(Fixed Prio and) Limited Priority Levels

Required: Mapping of

- Scheduling-Priorities: 1 ... n to
- Operating System Priorities: E, □2, ... □µ

Jobs running with same OS-Prio but different Sched-Prio use:
FIFO, Round Robin, ...

Schedulibility loss ?

- Notation: □i as grid on Scheduling Priorities
- Example:
  10 scheduling priorities, 3 OS priorities
  possible mapping: □1 =3, □2 = 8, □3 = 10
  Interpretation:
  0,1,2,3 mapped to □1, 4,5,6,7,8 to □2, 9,10 to □3

How is Schedulibility Test affected?

(Fixed Prio and) Limited Priority Levels

Mappings:

- **uniformly distributed:**
  k=n/m
  Scheduling Priority X mapped to $|X/m| * k$

- **constant ratio:**
  keep $(\square_{i-1} + 1)\square_i$ as equal as possible

Schedulilibility Loss

Rate Monotonic, large n ...

- $g = \min(\square_{\iota-1} + 1) / \mathbf{b}_{\iota}$

$SU_{RM} = \ln(2g)+1-g$

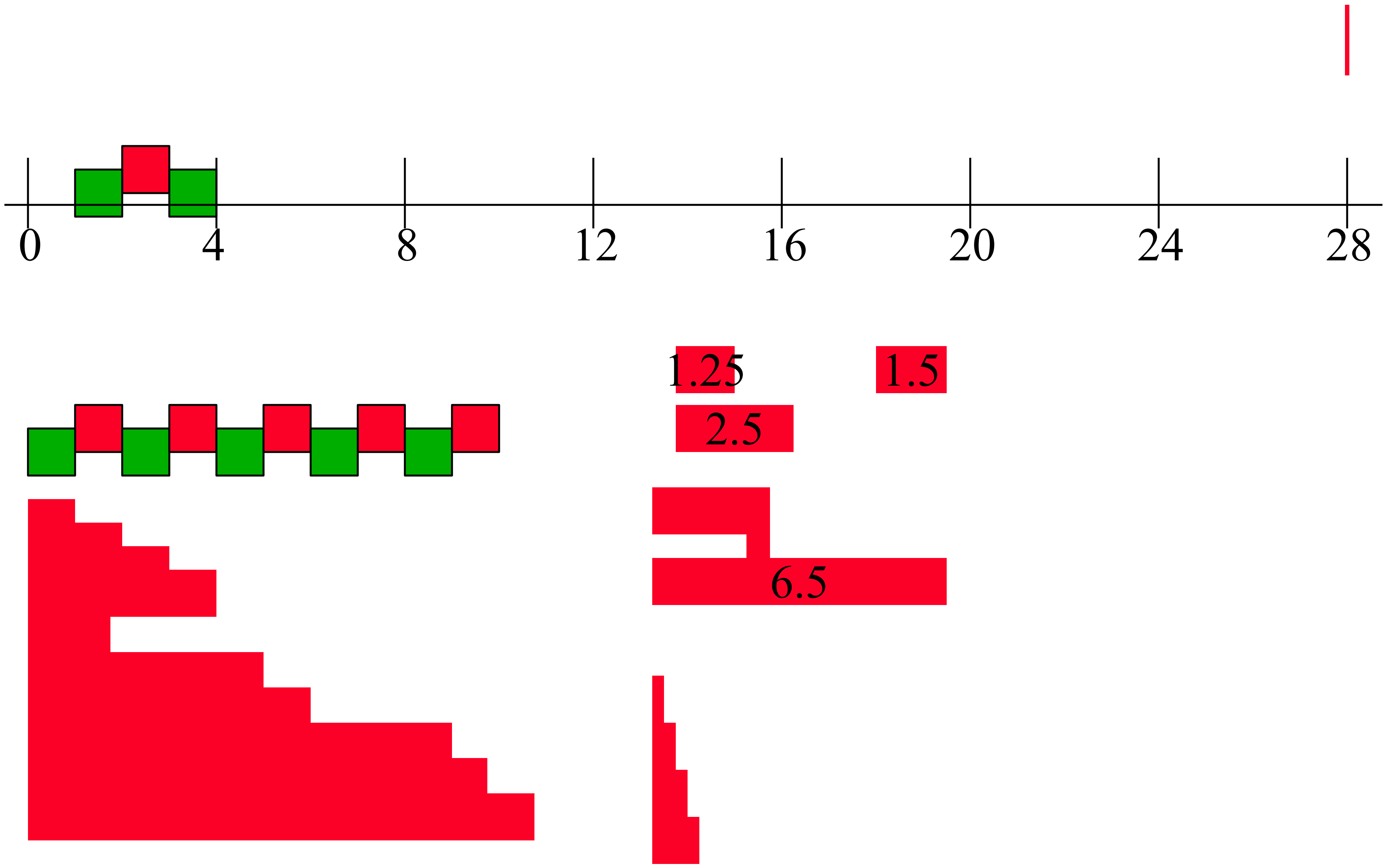
relative schedulibility(rs): relation to ln(2)

example:

n = 100000, m = 256

rs= 0.9986

=> 256 priorities is it !


0
4
8
12
16
20
24
28
1.25
1.5
2.5
6.5