CPU SCHEDULING

RONG ZHENG

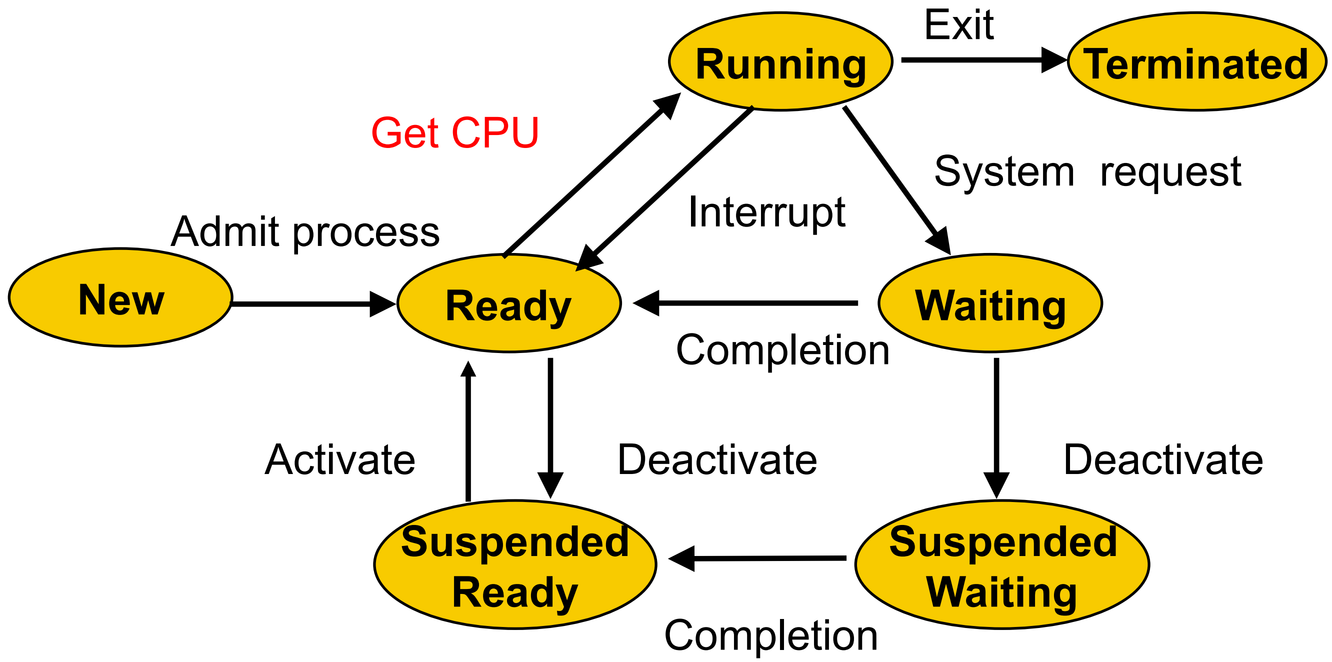
OVERVIEW

Why scheduling?

Non-preemptive vs Preemptive policies

FCFS, SJF, Round robin, multilevel queues with feedback, guaranteed scheduling

SHORT-TERM, MID-TERM, LONG-TERM SCHEDULER



Long-term scheduler: admission control

Mid-term scheduler: who gets to be loaded in memory

Short-term scheduler: who (in ready queue) gets to be executed

SCHEDULING METRICS

Waiting time: Waiting time is the sum of the periods spent waiting in the ready queue.

Turnaround time: The interval from *the time of submission* of a process to *the time of completion* is the turnaround time.

- The sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

Response time (interactive processes): the time from the submission of a request until the first response is produced.

Throughput: number of jobs completed per unit of time

- Throughput related to turningaround time, but not same thing:

CRITERIA OF A GOOD SCHEDULING POLICY

Maximize throughput/utilization

Minimize response time, waiting time

- Throughput related to response time, but not same thing

No starvation

- Starvation happens whenever some ready processes never get CPU time

Be fair

- How to measure fairness

Tradeoff exists

DIFFERENT TYPES OF POLICIES

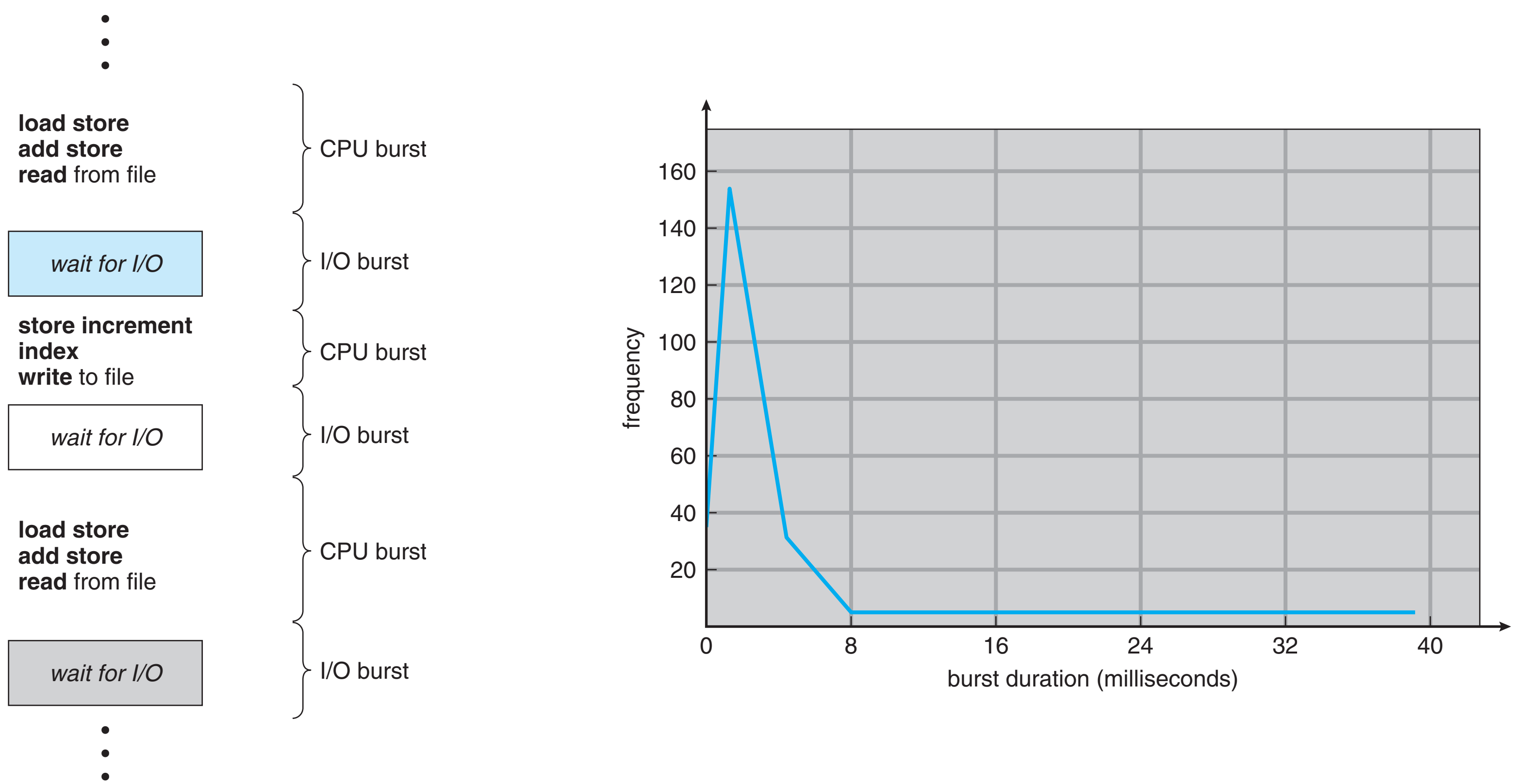
A non-preemptive CPU scheduler will never remove the CPU from a running process

- Will wait until the process releases the CPU because i) It issues a system call, or ii) It terminates
- Obsolete

A preemptive CPU scheduler can temporarily return a running process to the ready queue whenever another process requires that CPU in a more urgent fashion

- Has been waiting for too long
- Has higher priority

JOB EXECUTION



With time slicing, thread may be forced to give up CPU before finishing current CPU burst. Length of slices?

FIRST- COME FIRST-SERVED (FCFS)

Simplest and easiest to implement

- Uses a FIFO (First-in-first-out) queue

Previously for non-preemptive scheduling

Example: single cashier grocery store

Process	Burst Time
P_1	24
P_2	3
P_3	3

FCFS

Suppose processes arrive in the order: P1 , P2 , P3

Waiting time for P1 = 0; P2 = 24; P3 = 27

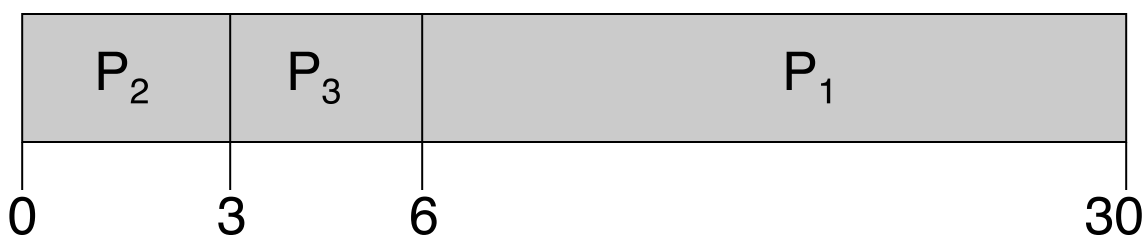
Average waiting time: (0 + 24 + 27)/3 = 17

Average completion time: (24 + 27 + 30)/3 = 27

Convoy effect: short process behind long process

FCFS (CONT'D)

Suppose processes arrive in the order: P2, P1 , P3



Waiting time? P1 = 6; P2 = 0; P3 = 3

Average waiting time? 3

Average completion time? (3+6+30)/3 = 13

Good to schedule to shorter jobs first

SHORTEST JOB FIRST (SJF)

Gives the CPU to the process requesting the least amount of CPU time

- Will reduce average wait
- Must know ahead of time how much CPU time each process needs
- Provably achieving shortest waiting time among non-preemptive policies
- Need to know the execution time of processes ahead of time – not realistic!

ROUND ROBIN

All processes have the same priority

Similar to FCFS but processes only get the CPU for a fixed amount of time T_{CPU}

- Time slice or time quantum

Processes that exceed their time slice return to the end of the ready queue

The choice of is T_{CPU} important

- Large → FCFS
- Small → Too much context switch overhead

EXAMPLE OF RR

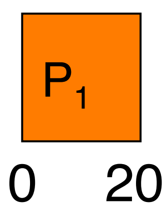
Process	Burst Time	Remaining Time
P_1	53	53
P_2	8	8
P_3	68	68
P_4	24	24

RR schedule

EXAMPLE OF RR

Process	Burst Time	Remaining Time
P_1	53	33
P_2	8	8
P_3	68	68
P_4	24	24

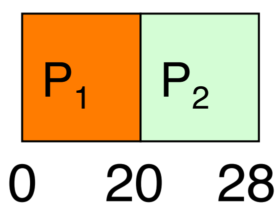
RR schedule



EXAMPLE OF RR

Process	Burst Time	Remaining Time
P_1	53	33
P_2	8	0
P_3	68	68
P_4	24	24

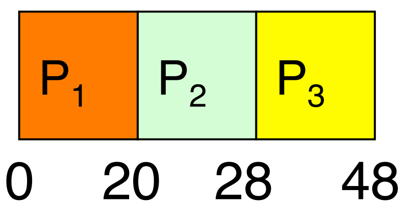
RR schedule



EXAMPLE OF RR

Process	Burst Time	Remaining Time
P_1	53	33
P_2	8	0
P_3	68	48
P_4	24	24

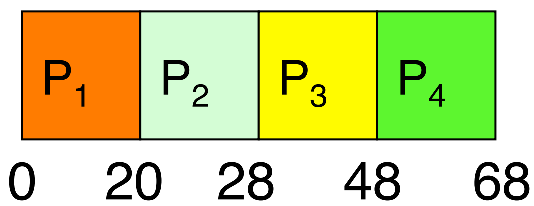
RR schedule



EXAMPLE OF RR

Process	Burst Time	Remaining Time
P_1	53	33
P_2	8	0
P_3	68	48
P_4	24	4

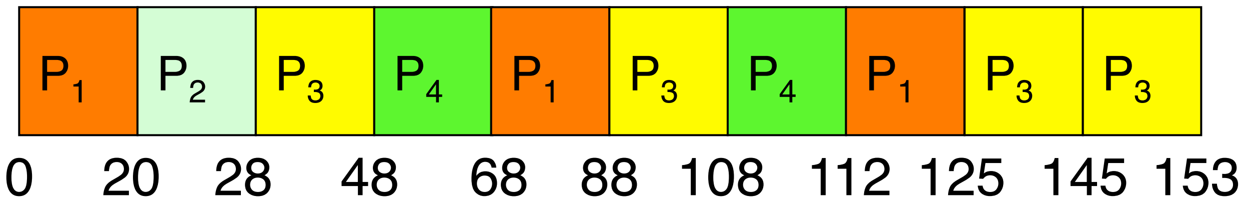
RR schedule



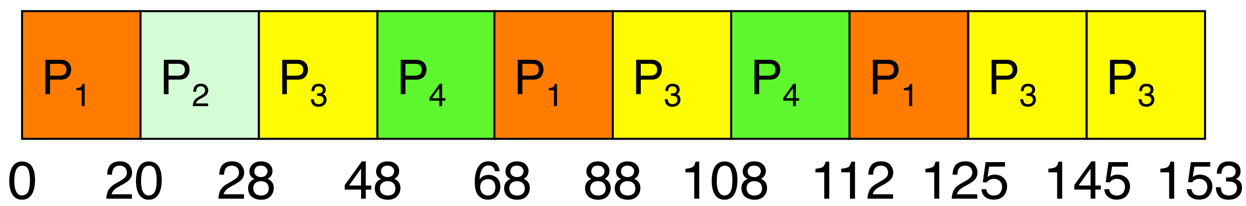
EXAMPLE OF RR

Process	Burst Time	Remaining Time
P_1	53	0
P_2	8	0
P_3	68	0
P_4	24	0

RR schedule



RR WITH QUANTUM = 20



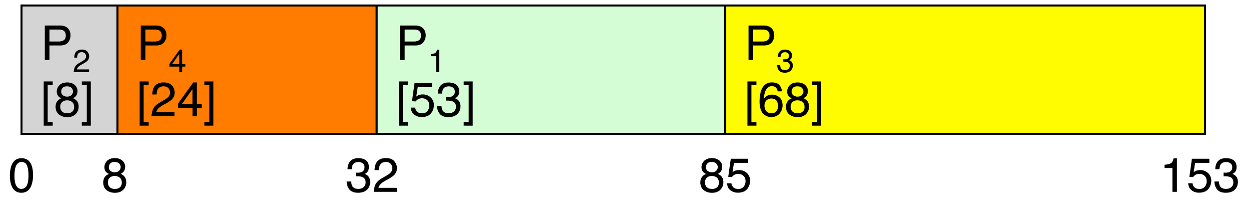
Waiting time for P1=(68-20)+(112-88)=72
P2=(20-0)=20
P3=(28-0)+(88-48)+(125-108)=85
P4=(48-0)+(108-68)=88

Average waiting time = (72+20+85+88)/4=66¼

Average completion time = (125+28+153+112)/4 = 104½

WITH DIFFERENT TIME QUANTUM

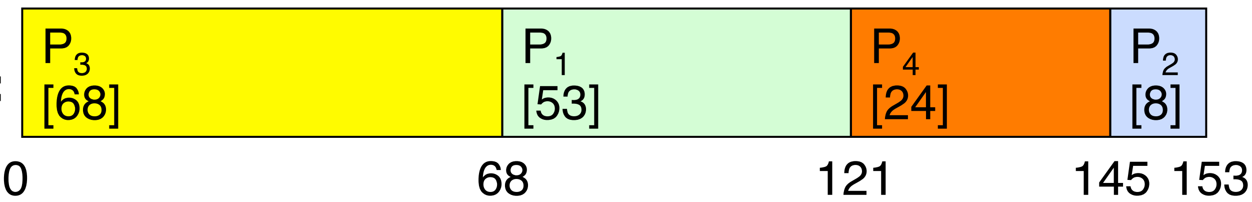
Best FCFS:



	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	Best FCFS	32	0	85	8	31¼
Completion Time	Best FCFS	85	8	153	32	69½

WITH DIFFERENT TIME QUANTUM

Worst FCFS:



	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	Best FCFS	32	0	85	8	31¼
	Worst FCFS	68	145	0	121	83½
Completion Time	Best FCFS	85	8	153	32	69½
	Worst FCFS	121	153	68	145	121¾

WITH DIFFERENT TIME QUANTUM

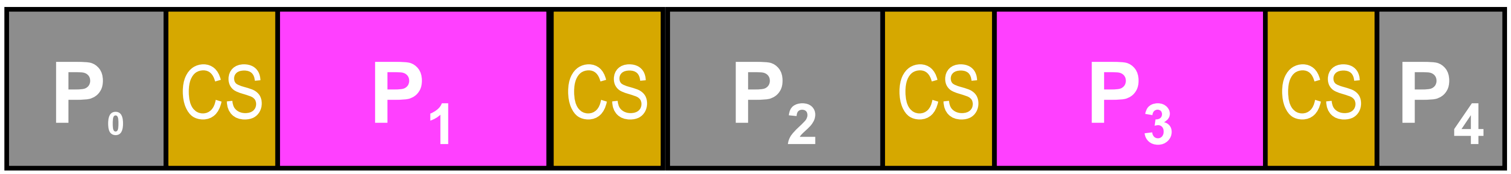
Worst FCFS:

P_3 [68]	P_1 [53]	P_4 [24]	P_2 [8]
0	68	121	145 153

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_1	P_3	P_1	P_3	P_1	P_3	P_3	P_3
0	8	**16**	24	32	40	48	56	64	72	**80**	88	96	104	112	120	128	**133**	141	149 **153**

	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	Q = 8	80	8	85	56	57¼
	Q = 10	82	10	85	68	61¼
	Q = 20	72	20	85	88	66¼
	Worst FCFS	68	145	0	121	83½
Completion Time	Best FCFS	85	8	153	32	69½
	Q = 1	137	30	153	81	100½
	Q = 5	135	28	153	82	99½
	Q = 8	133	16	153	80	95½
	Q = 10	135	18	153	92	99½
	Q = 20	125	28	153	112	104½
	Worst FCFS	121	153	68	145	121¾

IN REALITY



The completion time is long with context switches

- More harmful to long jobs

Choice of slices:

- Typical time slice today is between 10ms – 100ms
- Typical context-switching overhead is 0.1ms – 1ms
- Roughly 1% overhead due to context-switching

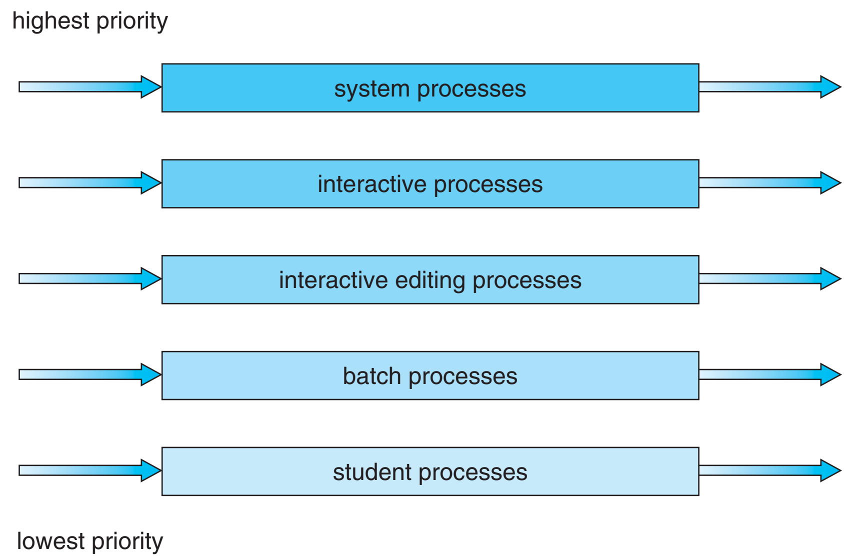
MULTILEVEL QUEUES WITH PRIORITY

Distinguish among

- Interactive processes – High priority
- I/O-bound processes – Medium priority
    - Require small amounts of CPU time
- CPU-bound processes – Low priority
    - Require large amounts of CPU time (number crunching)

One queue per priority

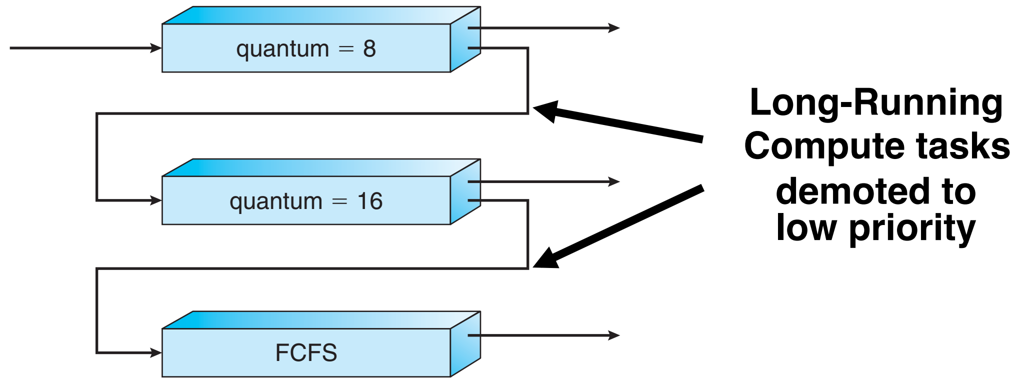
- Different quantum for each queue



Allow higher priority processes to take CPU away from lower priority processes

1. How do we know which is which?
2. What about starvation?

MULTI-LEVEL FEEDBACK SCHEDULING



Use past behavior to predict future

- First used in Cambridge Time Sharing System (CTSS)
- Multiple queues, each with a different priority
  - Higher priority queues often considered “foreground” tasks
- Each queue has its own scheduling algorithm
  - e.g., foreground – RR, background – FCFS
  - Sometimes multiple RR priorities with quantum increasing exponentially (highest:1ms, next:2ms, next: 4ms, etc.)

Adjust each job’s priority as follows (details vary)

- Job starts in highest priority queue
- If timeout expires, drop one level
- If timeout doesn’t expire, push up one level (or to top)

MULTI-PROCESSOR SCHEDULING

Multi-processor on a single machine or on different machines (clusters)

- Process affinity: avoid moving data around
- Load balancing
- Power consumption

SCHEDULING IN LINUX

Traditionally

- Multi-level feedback queue
- RR within each queue

Modern implementation

- Processes can be assigned one of three priority levels: Real Time (highest), Kernel, or Time Shared (lowest)
- Time shared processes use multi-level feedback queue
- Priority levels of time-shared processes can be adjusted (relatively) via *nice* command
- For SMP, support process affinity and load balancing

REAL-TIME SCHEDULING

REAL-TIME SYSTEMS

Systems whose correctness depends on their temporal aspects as well as their functional aspects

- Control systems, automotive …

Performance measure

- Timeliness on timing constraints (deadlines)
- Speed/average case performance are less significant.

Key property

- Predictability on timing constraints

Hard vs soft real-time systems

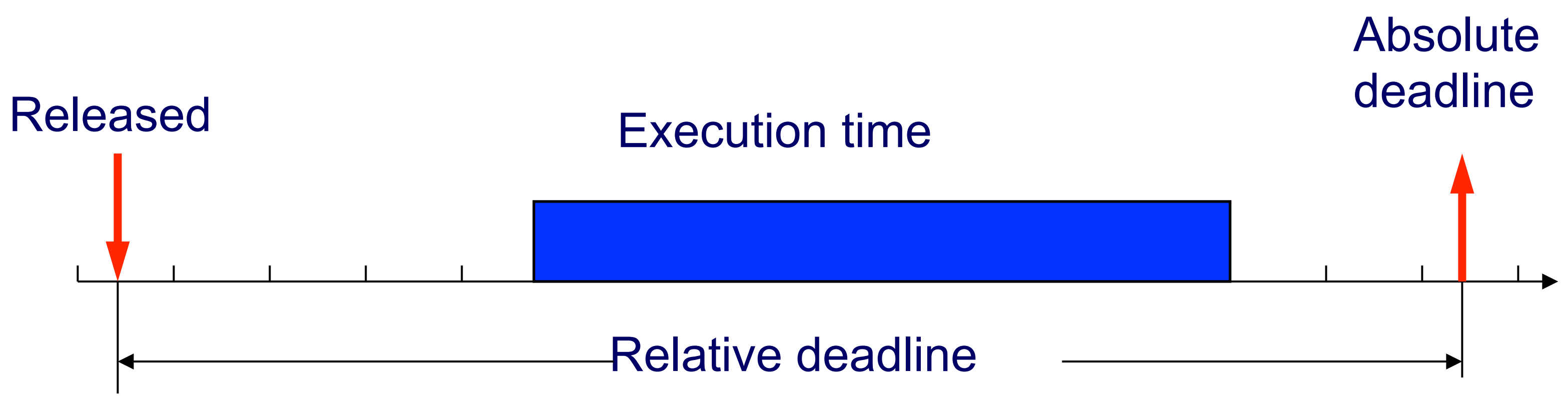
REAL-TIME WORKLOAD

Job (unit of work)

- a computation, a file read, a message transmission, etc

Attributes

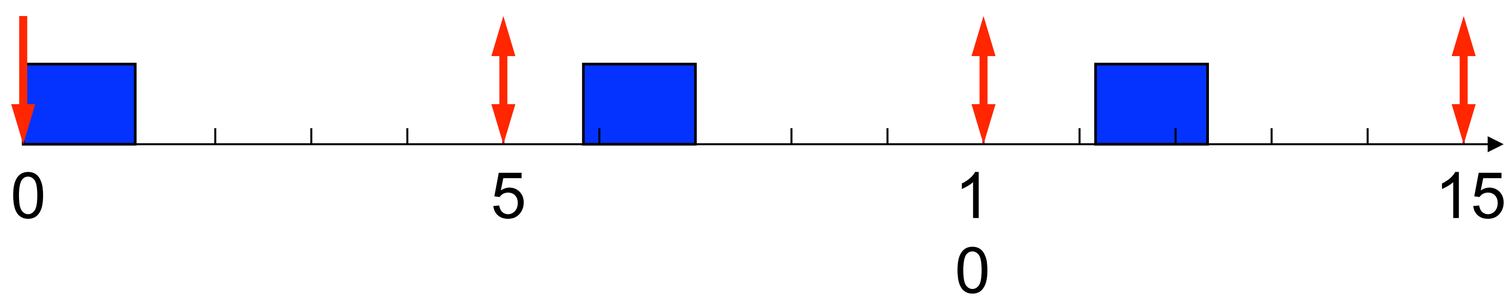
- Resources required to make progress
- Timing parameters



REAL-TIME TASK

Task : a sequence of similar jobs

- Periodic task (p,e)
    - Its jobs repeat regularly
    - Period p = inter-release time ($0 < p$)
    - Execution time e = maximum execution time ($0 < e < p$)
    - Utilization U = e/p



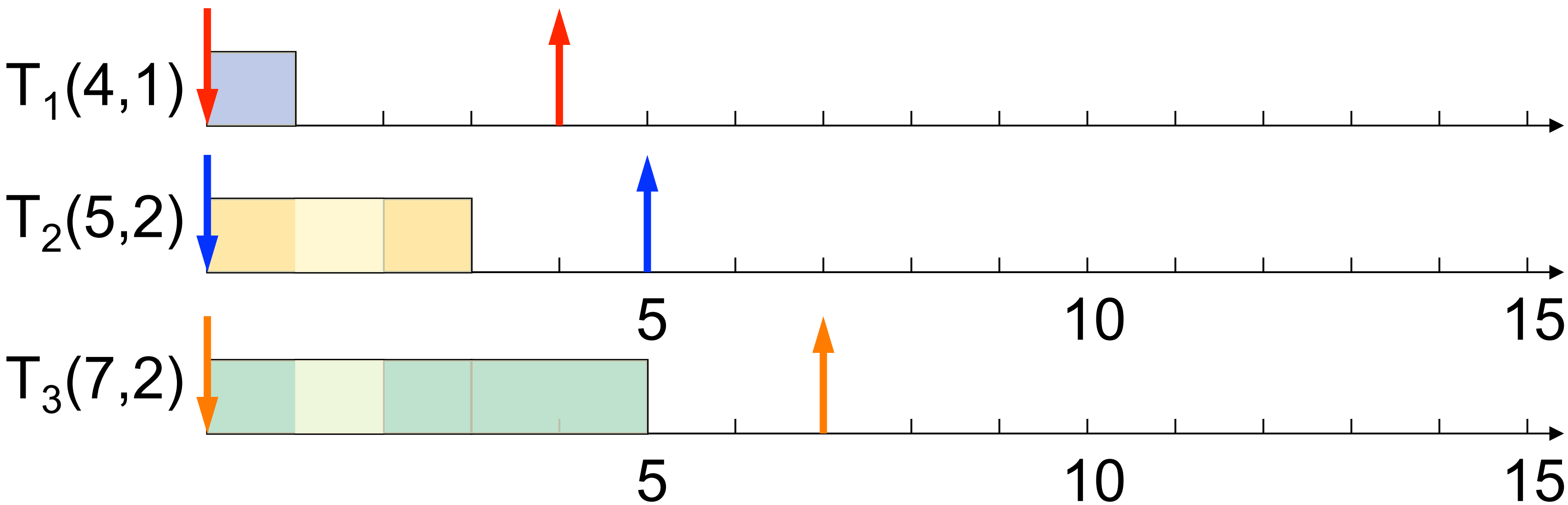
RATE MONOTONIC

Optimal static-priority scheduling

It assigns priority according to period

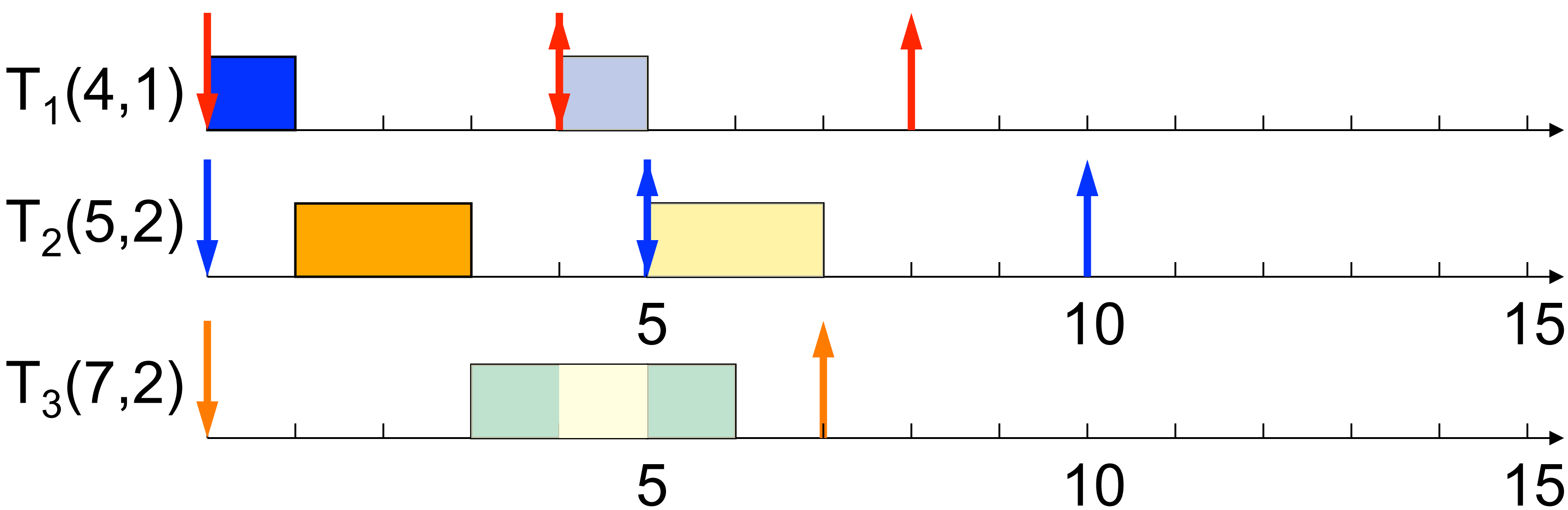
A task with a shorter period has a higher priority

Executes a job with the shortest period



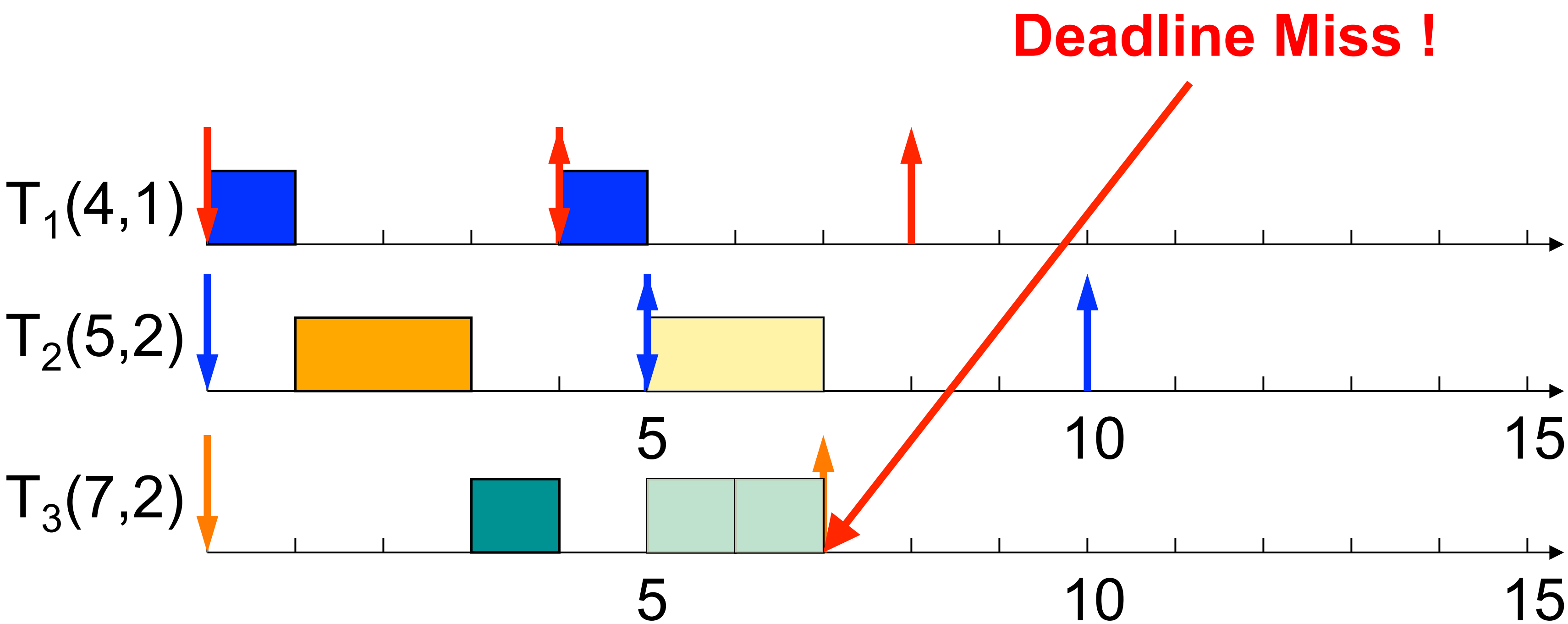
RM (RATE MONOTONIC)

Executes a job with the shortest period



RM (RATE MONOTONIC)

Executes a job with the shortest period

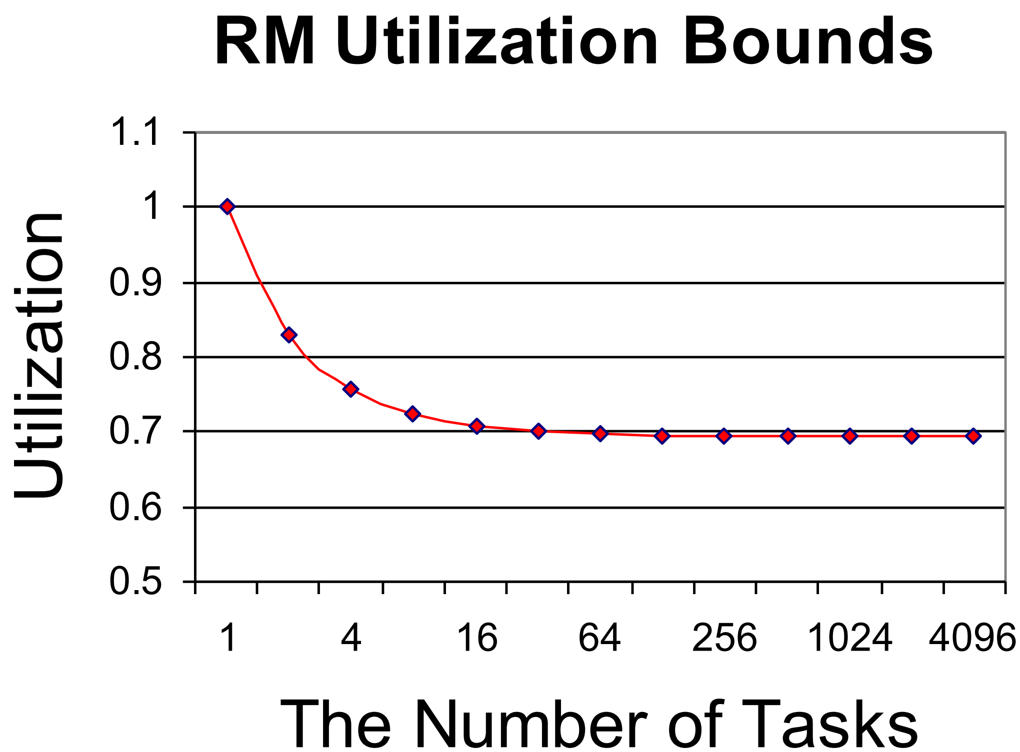


RM – UTILIZATION BOUND

Real-time system is schedulable under RM if

$$\sum C_i/T_i \leq n\,(2^{1/n}-1)$$

C_i is the computation time (work load), T_i is the period

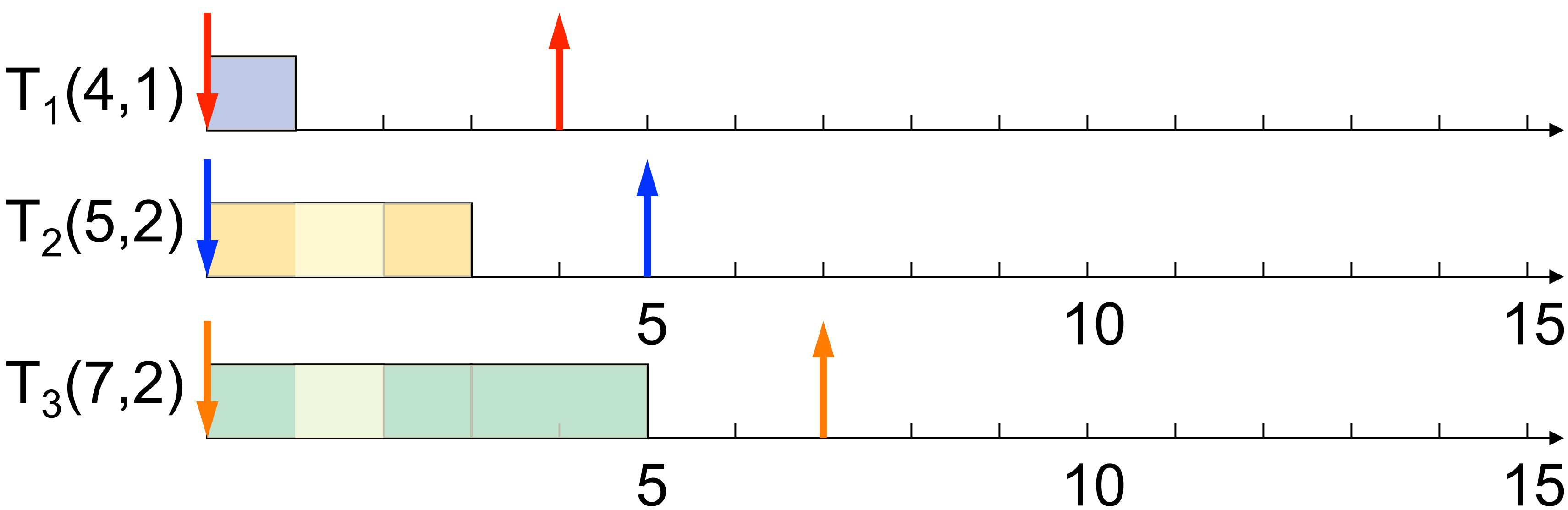


EDF (EARLIEST DEADLINE FIRST)

Optimal dynamic priority scheduling

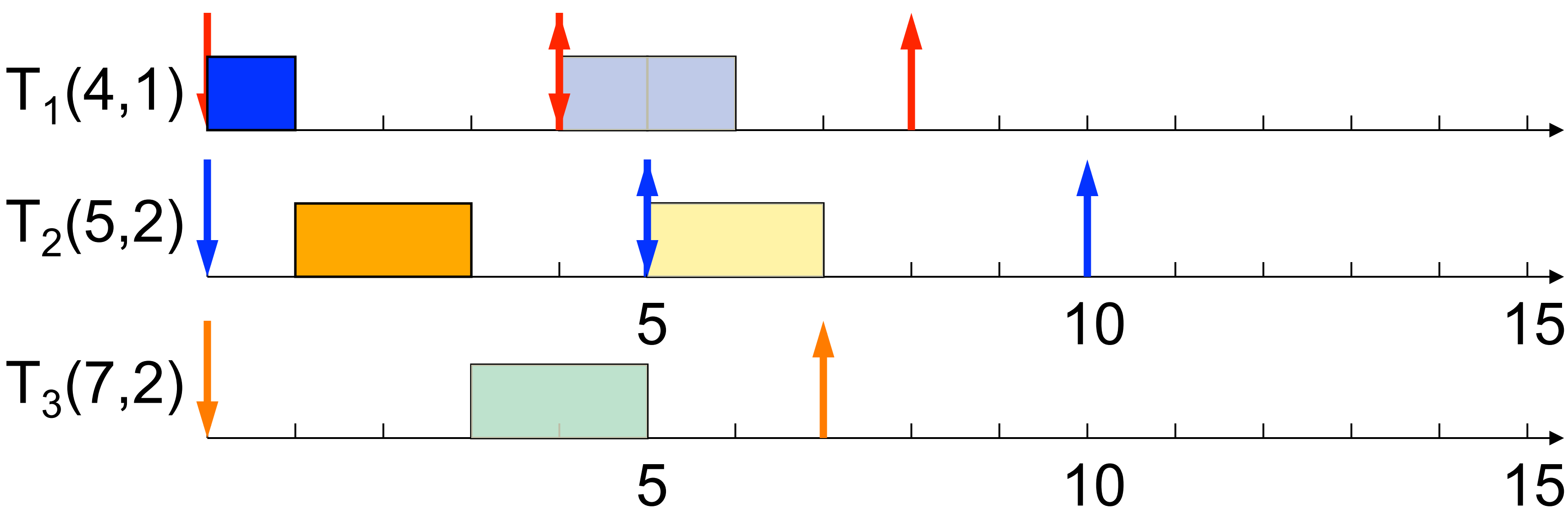
A task with a shorter deadline has a higher priority

Executes a job with the earliest deadline



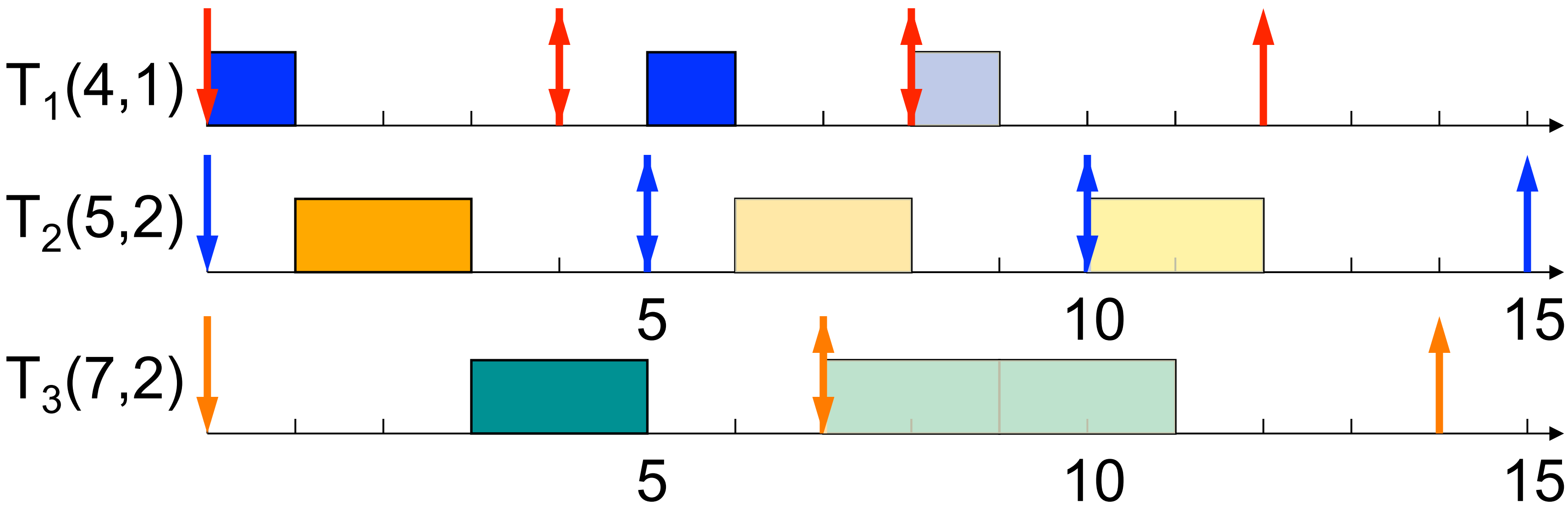
EDF (EARLIEST DEADLINE FIRST)

Executes a job with the earliest deadline



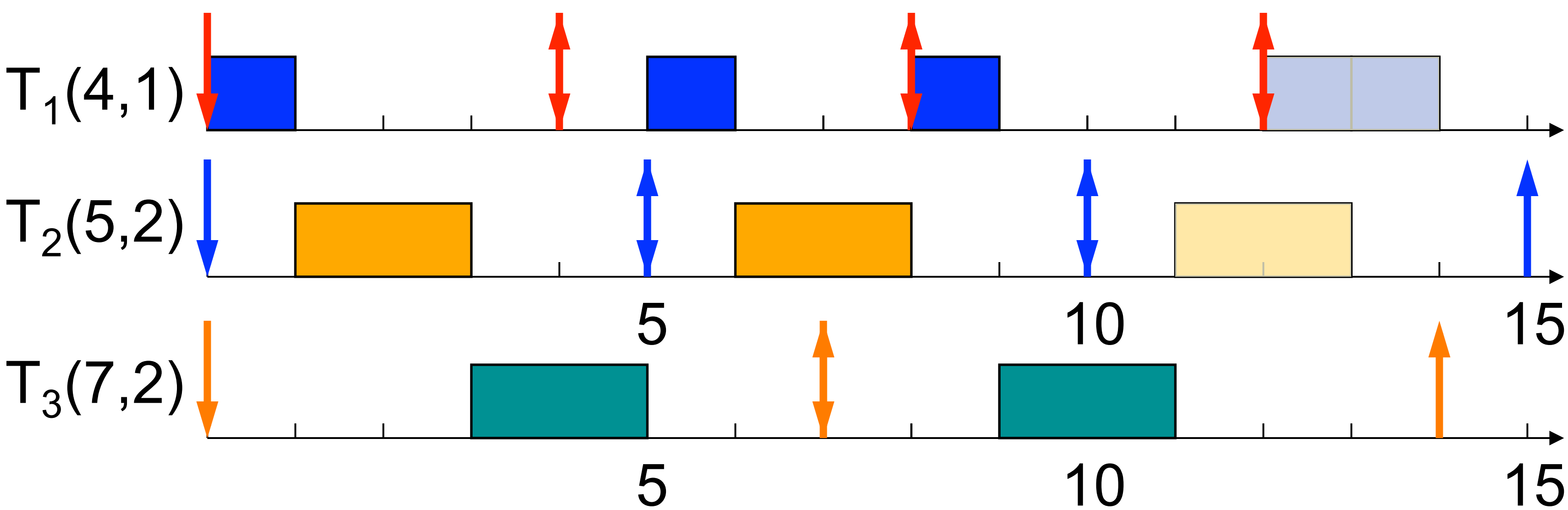
EDF (EARLIEST DEADLINE FIRST)

Executes a job with the earliest deadline



EDF (EARLIEST DEADLINE FIRST)

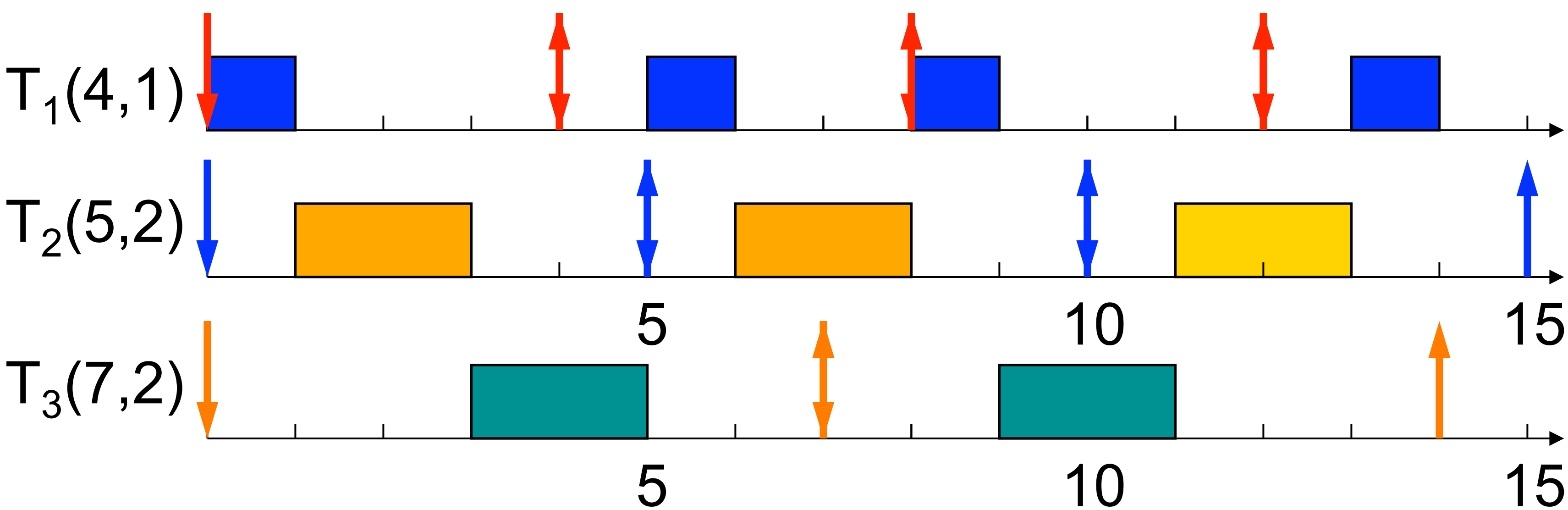
Executes a job with the earliest deadline



EDF (EARLIEST DEADLINE FIRST)

Optimal scheduling algorithm

- if there is a schedule for a set of real-time tasks, EDF can schedule it.



EDF – UTILIZATION BOUND

Real-time system is schedulable under EDF if and only if

$$\sum C_i/T_i \leq 1$$

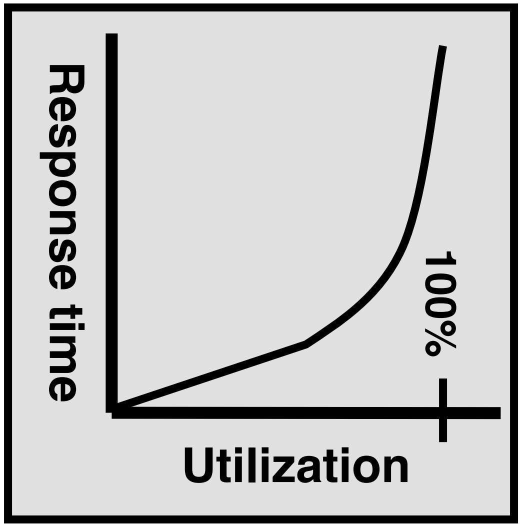
Liu & Layland,

"Scheduling algorithms for multi-programming in a hard-real-time environment", Journal of ACM, 1973.

SUMMARY

Scheduling matters when resource is tight

- CPU, I/O, network bandwidth
- Preemptive vs non-preemptive
- Burst time known or unknown
- Hard vs soft real-time
- Typically tradeoff in fairness, utilization and real-timeliness



COMPARISON

	Utilization (throughput)	Response time	Fairness
FCFS	100%	High	Good
SJF	100%	Shortest	Poor
RR	100%	Medium	Good
Multi-level priority with feedback	100%	Short	Good
RM	$\sum C_i/T_i \leq n\,(2^{1/n}-1)$	-	-
EDF	100%	-	-