Chapter 4-2: Processes and Operating Systems

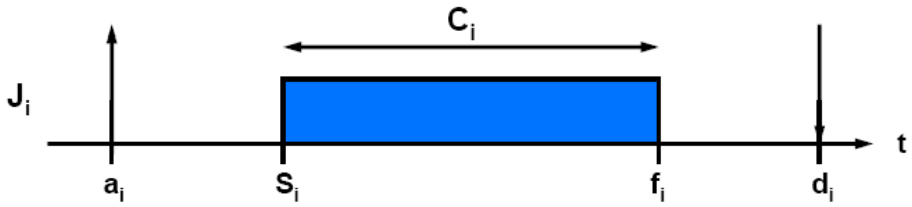
Soo-Ik Chae

Topics

- Earliest deadline first (EDF)
- Least laxity first (LLF)
- EDF (periodic)
- Priority inversion
- Priority inversion protocol (PIP)

Parameters for a real-time task J

- **Arrival time** a_i :
  the time J_i becomes ready for execution
  also called *request time* or *release time*, denoted by r_i
- **Computation time** C_i:
  time necessary for execution without interruption
- **Deadline** d_i:
  time before which task has to have completed its execution
- **Start time** S_i:
  time at which J_i starts its execution
- **Finishing time** f_i:
  time at which J_i finishes its execution



Typical parameters of a real-time task

Earliest Deadline First (EDF)

- Different arrival times: Preemption is allowed, which potentially reduces lateness.
- **Theorem** [Horn74]: Given a set of *n* independent tasks with arbitrary arrival times, any algorithm that at any instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.

Proof of Horn's Theorem

- Let A be an arbitrary algorithm producing σ, $\sigma \neq \sigma_{EDF}$
- Preemption allowed => each task may be executed in disjointed time intervals of unit length. Assume that the execution time of each task can be divided into time slices of unit length
- Let $\sigma(t)$ = a task executing in slice [t, t+1)

  E(t) = a ready task that, at t, has earliest deadline

  $t_E(t)$ = time ($\geq$ t) at which the next slice of E(t) begins its execution in the current schedule σ.

  $\sigma \neq \sigma_{EDF}$ => $\exists$ t in σ ; $\sigma(t) \neq E(t)$

We show that interchanging the position of $\sigma(t)$ and E(t) cannot increase the maximum lateness. If σ starts at t=0 and D = max{d_j} σ_{EDF} can be obtained from by at most D transposition Q.E.D.

Proof of Horn's Theorem: interchanging

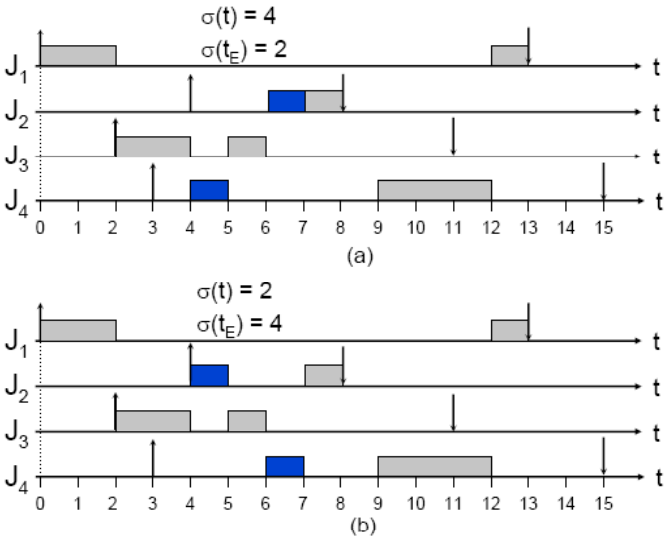
Note: We shortly write t_E instead of $t_E(t)$

```
Algorithm : interchange
 {
      for( t=0 to D-1 ) {
         if (σ (t) ) ≠ E(t) ) {
               σ (t_E)  =  σ(t) ;
               σ (t)    =   E(t) ;
         }
      }
 }
```

$\sigma(t)$ = task executing in slice [t, t+1)
E(t) = ready task that, at t, has earliest deadline
t_E = time ($\geq$t) at which the next slice of E(t) begins its execution in the current schedule

i.e. for each time slice t it is checked whether $\sigma(t) = E(t)$
if not the slice at t and E(t) are exchanged

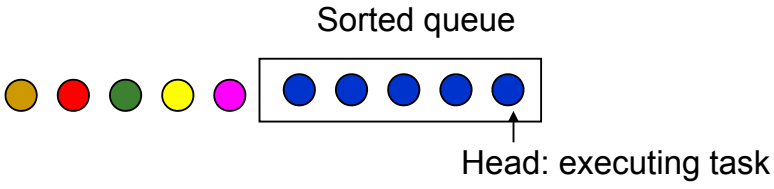
Illustration of interchanging



Proof of the optimality of the EDF algorithm.
(a) schedule s at time t = 4
(b) new schedule obtained after a transposition

EDF scheduling algorithm

- **Earliest deadline first** (EDF) algorithm:
  - Each time a new ready task arrives:
  - It is inserted into a queue of ready tasks, sorted by their **absolute** deadlines. Task at head of queue is executed.
  - If a newly arrived task is inserted at the head of the queue, the currently executing task is preempted.
- Inserting a newly arriving task into an ordered list : *O*(log *n*);
  - n tasks => total complexity: O(n log n).



Preservation of Schedulability in EDF

- Corollary

  If σ is schedulable, then σ_{EDF} is as well.

Proof:

We show that each transposition preserves schedulability

At each instance:

each slice be - either anticipated

- or postponed up to t_E

If a slice is anticipated the feasability of this task is preserved

Assume a slice of J_i is postponed to t_E

σ feasible => $(t_E+1) \leq d_E$

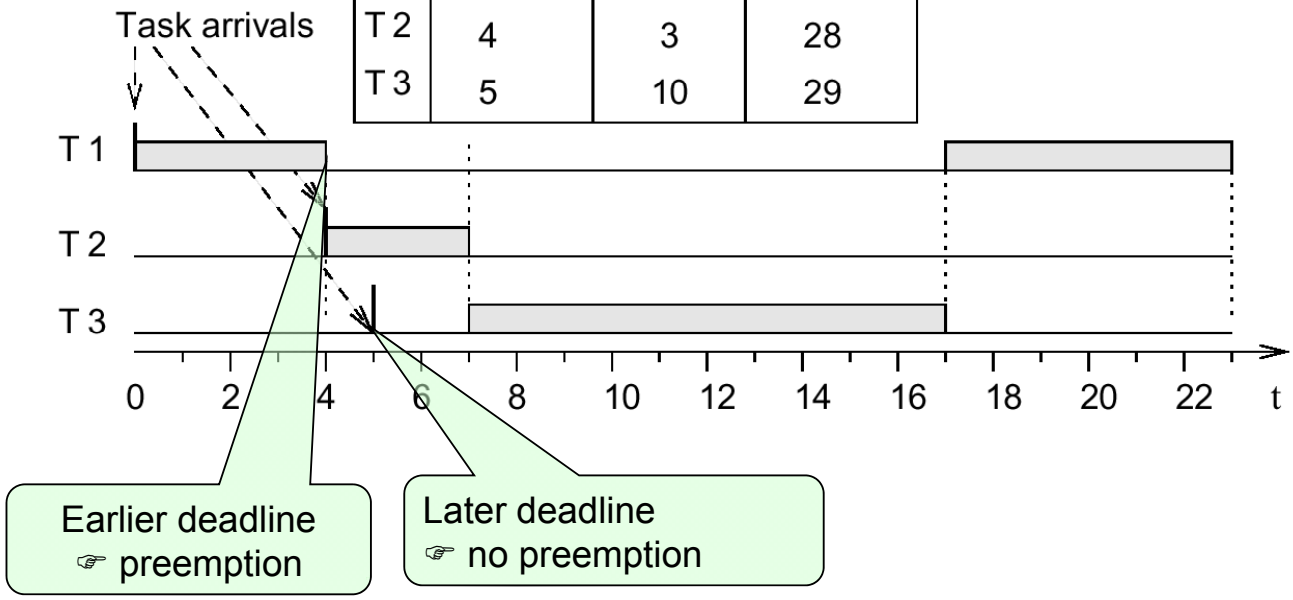
=>$(\forall_j: d_E < d_j)$ $(t_E+1) < d_j$

=> J_i postponed to t_E is feasible

q.e.d.

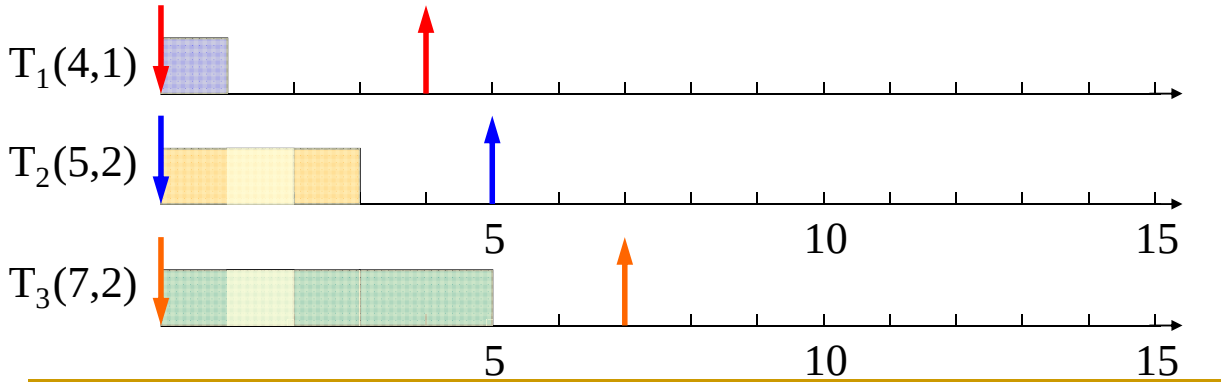
EDF Example

	arrival	duration	deadline
T 1	0	10	33
T 2	4	3	28
T 3	5	10	29



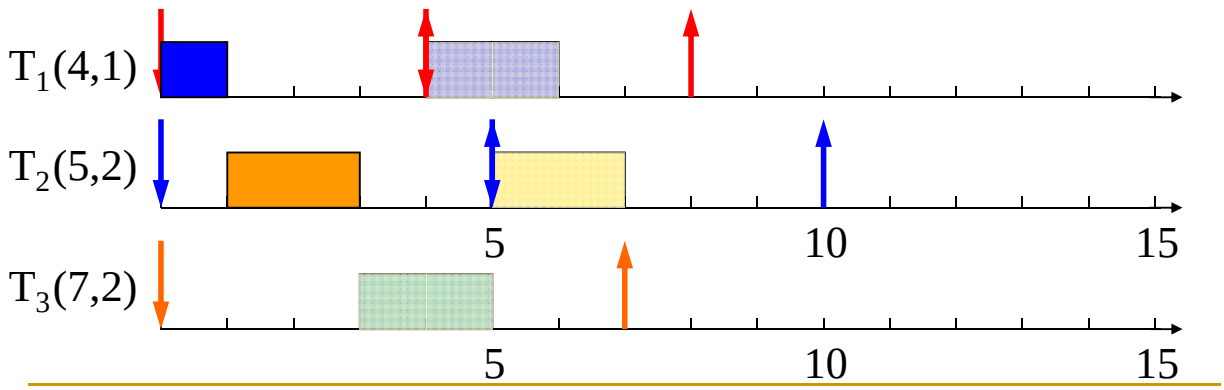
EDF (Earliest Deadline First)

- Optimal dynamic priority scheduling
- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline



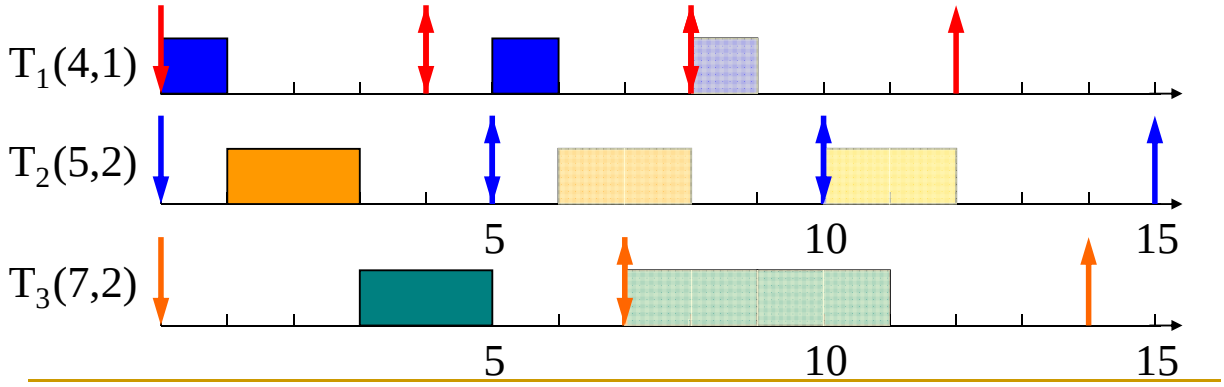
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline



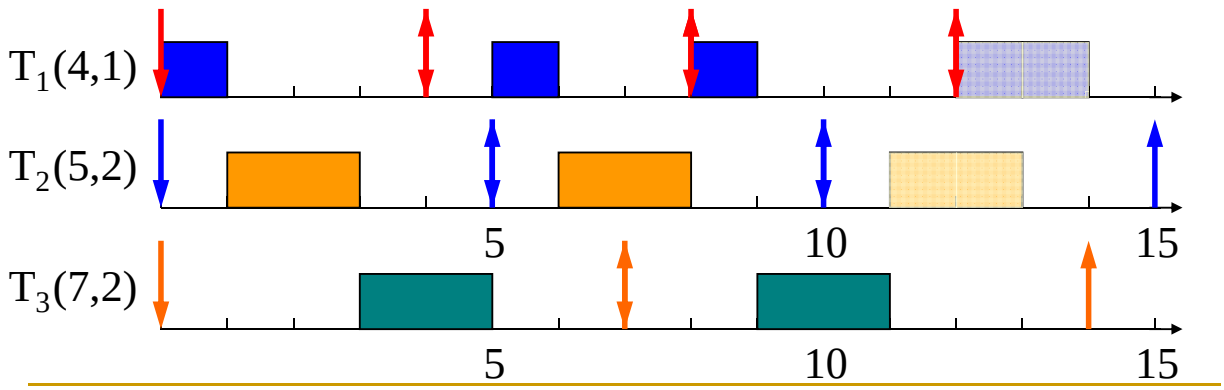
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline



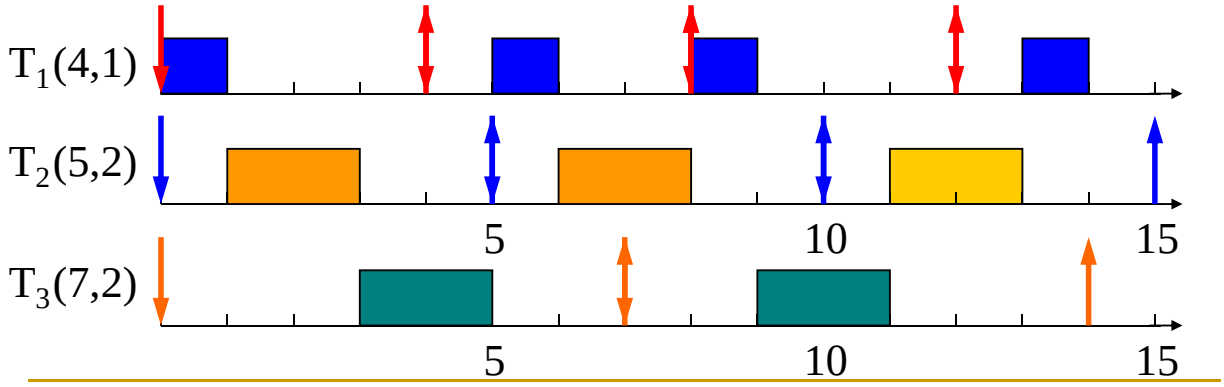
EDF (Earliest Deadline First)

- 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

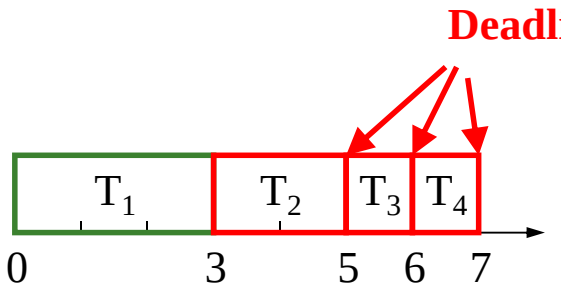
$$\sum U_i \leq 1$$

Liu & Layland,

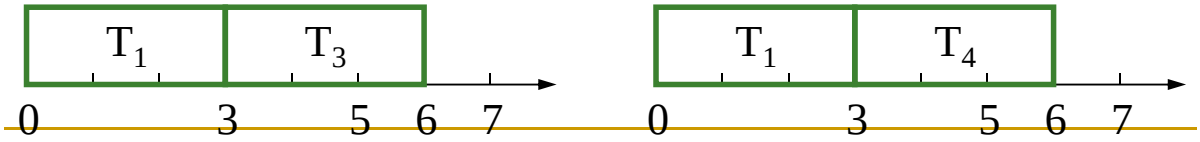
"Scheduling algorithms for multi-programming in a hard-real-time environment", Journal of ACM, 1973.

EDF – Overload Conditions

- Domino effect during overload conditions
  - Example: $T_1(4,3)$, $T_2(5,3)$, $T_3(6,3)$, $T_4(7,3)$



Better schedules :



RM vs. EDF

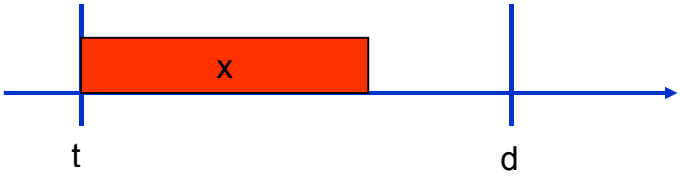
- Rate Monotonic
  - Simpler implementation, even in systems without explicit support for timing constraints (periods, deadlines)
  - Predictability for the highest priority tasks
- EDF
  - Full processor utilization
  - Misbehavior during overload conditions
- For more details: Buttazzo, “Rate monotonic vs. EDF: Judgement Day”, EMSOFT 2003.

Least laxity first (LLF) scheduling

- also called as least slack time first (LST) or minimum laxity first (MLF)
- Priorities = decreasing function of the laxity
- The less laxity, the higher the priority. Dynamically changing priority; preemptive.
- Laxity or slack: difference between remaining computation time and time until deadline.
  - Process with smallest laxity has highest priority.
- Unlike EDF, takes into account computation time in addition to deadline.

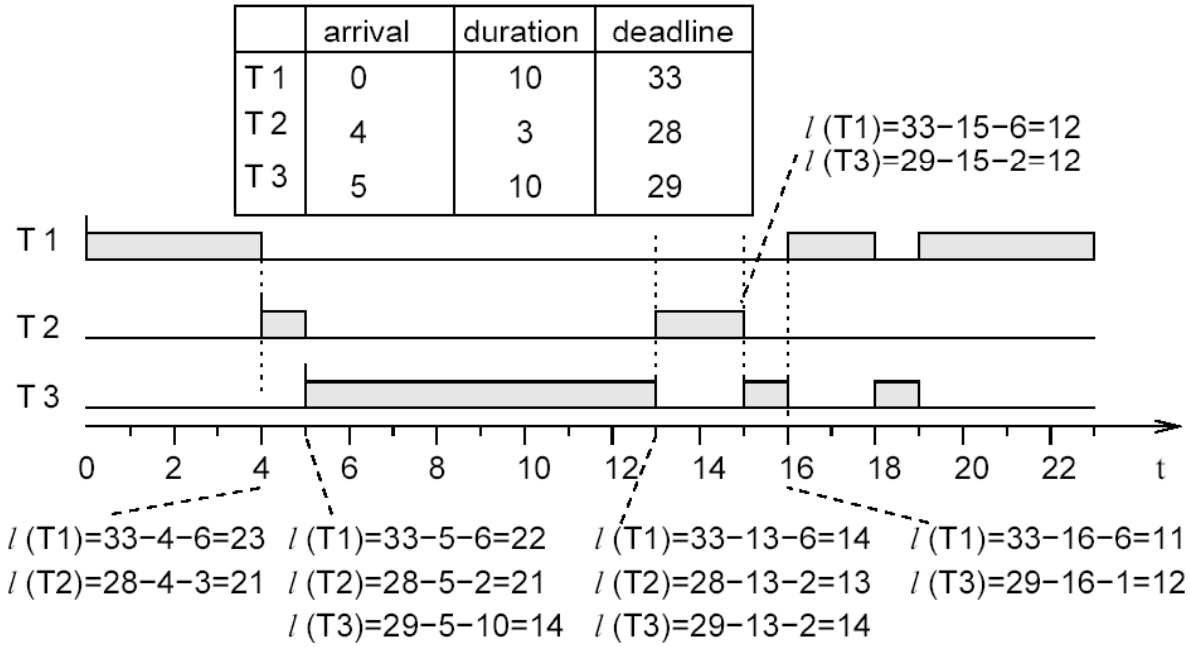
Dynamic Priority Assignment of LLF

- slack = d - $t_{current}$ - x
  - d = deadline
  - $t_{current}$ = time at which slack is computed
  - x = execution time of remaining portion
- scheduler checks slack whenever a new job is released or a job is completed



여유 없는 것(급한 것) 먼저

Least laxity first schedule

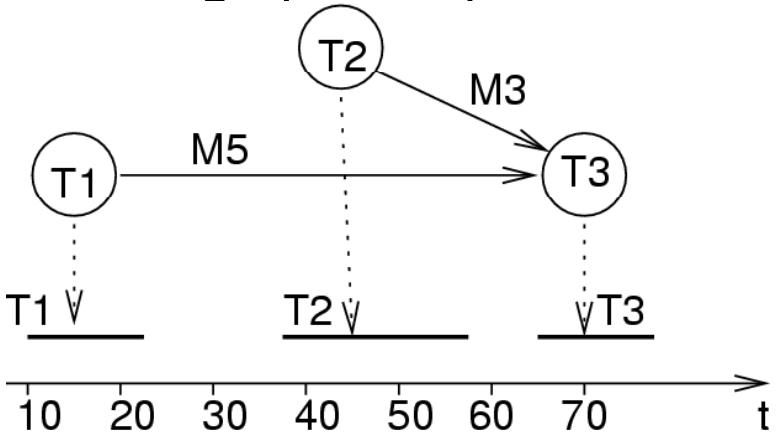


Properties of LLF

- Need to re-compute laxity just at task arrival times.
- Overhead for calls of the scheduler.
- More context switches.
- Detects missed deadlines early.
- LLF is also an optimal scheduling for mono-processor systems.
- Dynamic priorities ☞ cannot be used with a fixed priority OS.
- LLF scheduling requires the knowledge of the execution time.

Scheduling with precedence constraints

- Task graph and possible



Schedule can be stored in table.

Scheduling with precedence constraints

- In general: NP=hard problem
- For special cases, polynomial time algorithm is possible

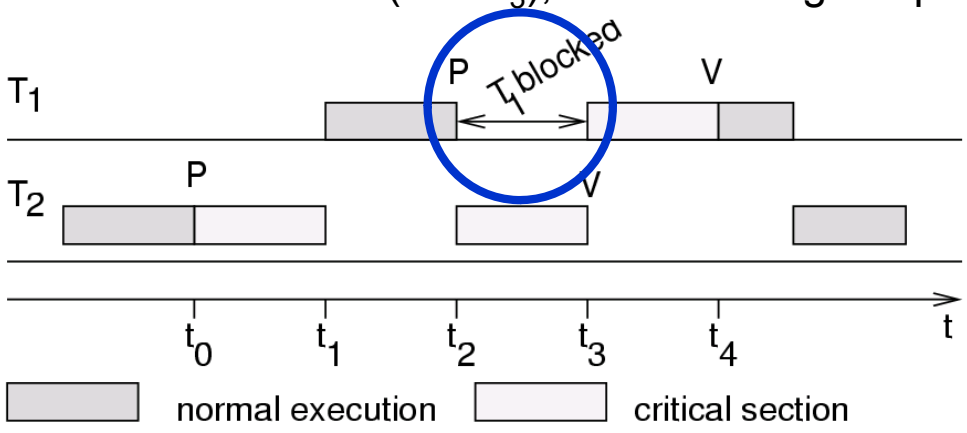
- Latest Deadline First (LDF)
- Given J, a set of tasks with a DAG describing their precedence. Arrival times are assumed to be simultaneous.

Priority inversion

- RMS and EDF assume that a process could always be preempted
  - But that is not true in practical systems
- If a process includes a critical section, then it cannot be preempted while it is in the critical section.
- A critical section is often used to protect the process's access to a shared resource.
- The critical section causes higher-priority processes to be excluded from executing.
- Priority inversion: external resources can make a low-priority process continue to execute as if it had higher priority.

Priority inversion

- Priority T_1 assumed to be higher than priority of T_2.
- If T_2 requests exclusive access first (at t_0), T_1 has to wait until T_2 releases the resource (time t_3), thus inverting the priority:



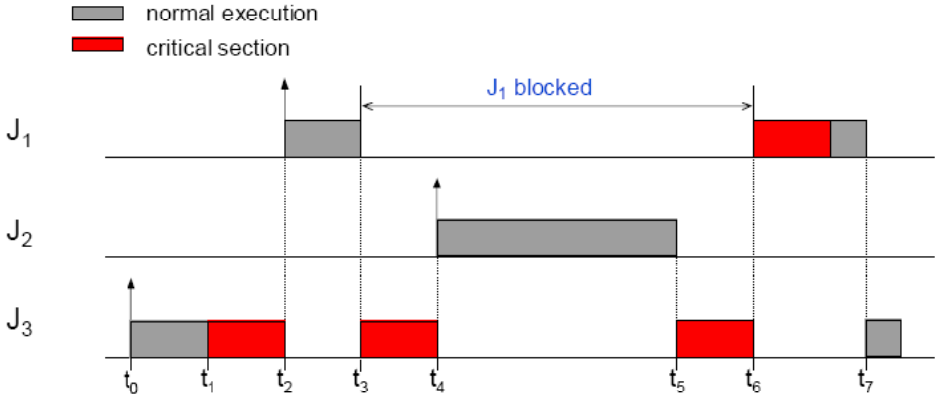
In this example:
duration of inversion bounded by length of critical section of T_2.

Duration of priority inversion with >2 tasks

maximum blocking time of J_1 = duration of J_2 in critical section

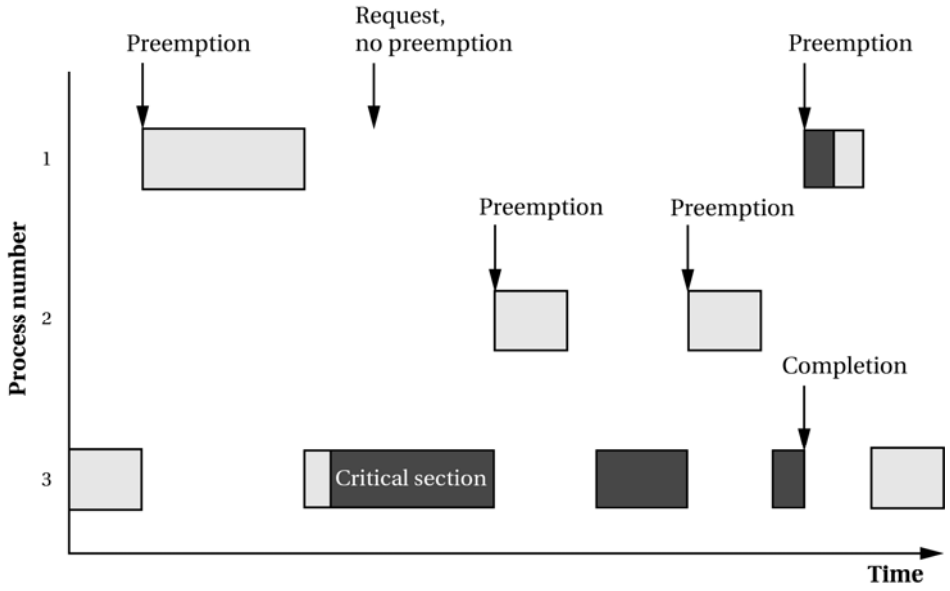
→ unavoidable due to semantics of critical section

However: blocking time may be unbounded if there are tasks with intermediate priority:



Priority inversion occurs in interval $[t_3, t_6]$

Priority inversion example



Critical section is shared with processes 1 and 3

Process 2's preemption can delay process 3 for an arbitrarily long time.

Consequently process 1 is delayed together even though it has higher priority

The MARS Pathfinder problem (1)

- "But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once"." …

The MARS Pathfinder problem (2)

- "VxWorks provides preemptive priority scheduling of threads. Tasks on the Pathfinder spacecraft were executed as threads with priorities that were assigned in the usual manner reflecting the relative urgency of these tasks."
- "Pathfinder contained an "information bus", which you can think of as a shared memory area used for passing information between different components of the spacecraft."
  - A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus. Access to the bus was synchronized with mutual exclusion locks (mutexes)."

The MARS Pathfinder problem (3)

- The meteorological data gathering task ran as an infrequent, low priority thread, … When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex. ..
- The spacecraft also contained a communications task that ran with medium priority."

☞
High priority: retrieval of data from shared memory
Medium priority: communications task
Low priority: thread collecting meteorological data

The MARS Pathfinder problem (4)

- "Most of the time this combination worked fine. However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread. In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running. After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset. This scenario is a classic case of priority inversion."

Priority inheritance protocols

- Sha et al.: basic priority inheritance protocol, priority ceiling protocol.
- Process in a critical section executes at highest priority of any process that shares that critical section (temporarily while in the critical section)
  - PIP reduces the blocking time!
- Possible deadlock: If a process needs several critical sections, it can be blocked sequentially at each of those critical sections by different processes, causing a blocking chain that shows down the higher-priority process.

Priority inheritance protocol (PIP)

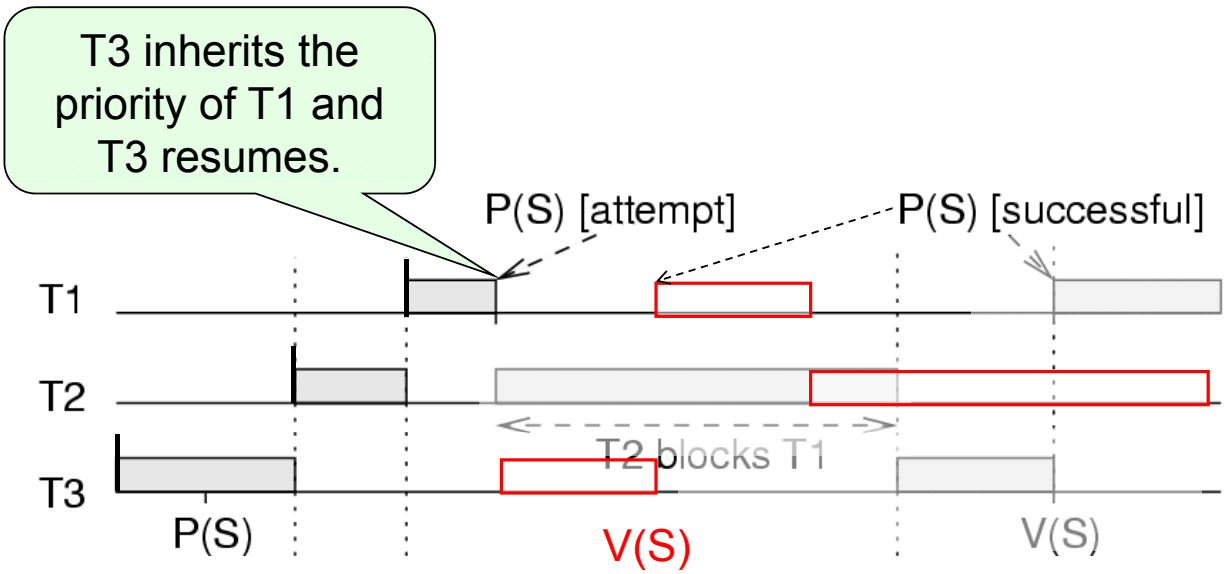
- * A task is scheduled according to its active priority. Tasks with the same priorities are scheduled FCFS.
- A task inherits the highest priority from the tasks it blocks.
- If task T1 executes P(S) but its exclusive access was granted to T2, then T1 will be blocked.
- If priority(T2) < priority(T1), then T2 inherits the priority of T1 so that T2 can release the shared resource earlier by preventing medium-priority tasks from preempting T2 and prolonging the blocking period..
- When T2 executes V(S), its original priority at the point of entry of the critical section as restored.

Priority inheritance protocol

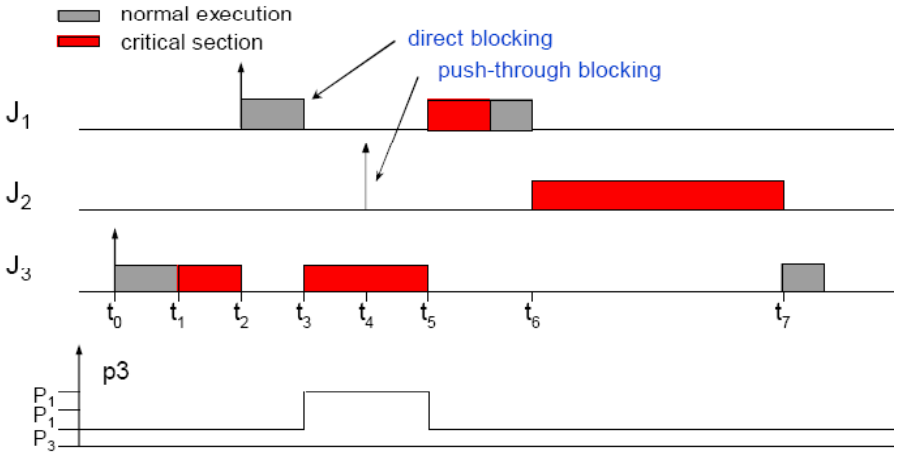
- Priority inheritance is transitive
- Assuming that priority(T1) > priority(T2) > priority(T3)
- If T3 blocks T2 and T2 blocks T1, then T3 inherits the priority of T1.

PIP: Example (1)

- How would priority inheritance affect our example with 3 tasks?



PIP: Example (2)

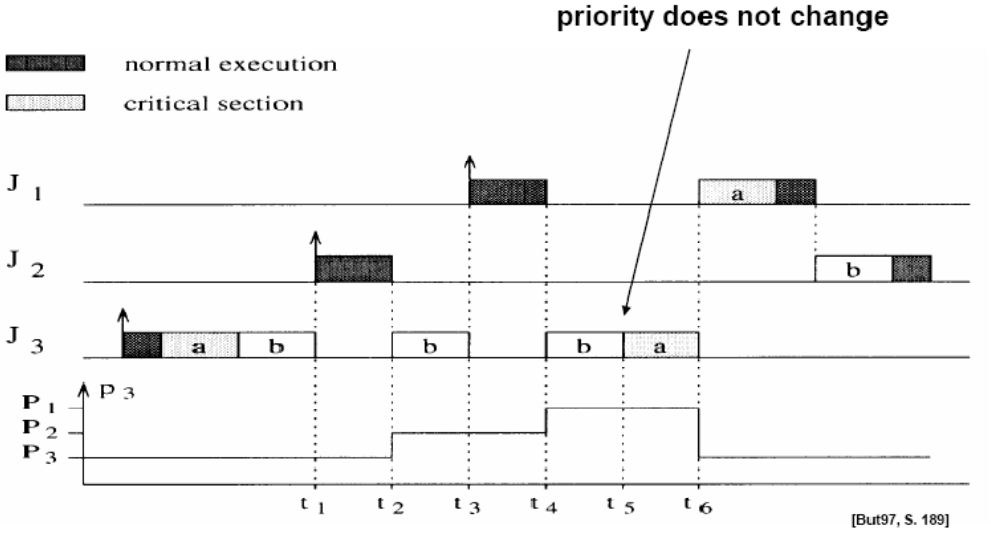


Direct Blocking: **higher-priority job tries to acquire a resource held by a lower-priority job**

Push-through Blocking: **medium-priority job is blocked by a lower-priority job that has inherited a higher priority form a job it directly blocks**

PIP: Example (3)

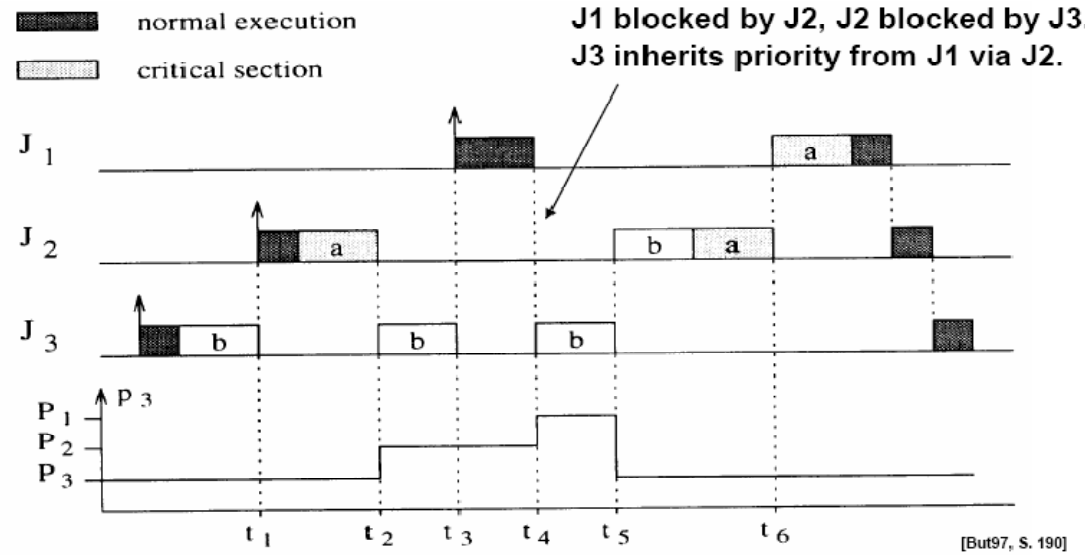
- Example with nested critical sections:



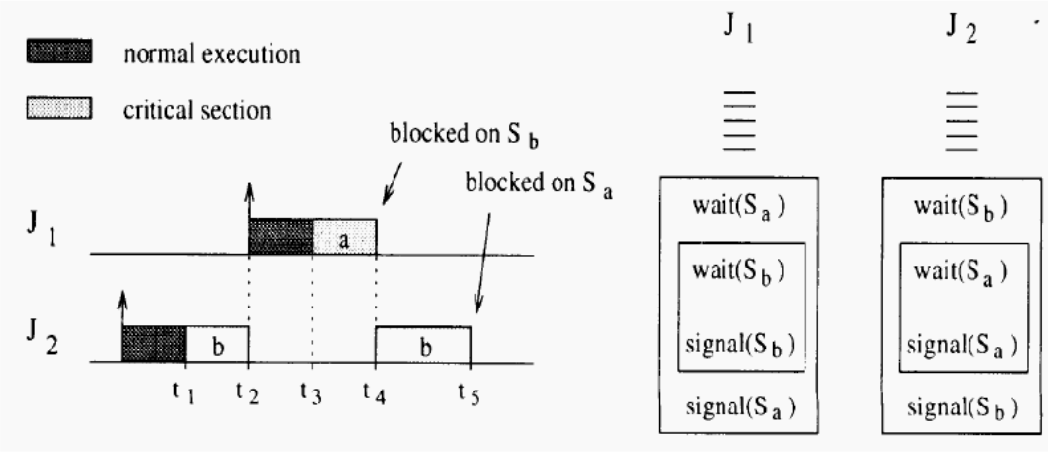
[But97, S. 189]

PIP: Example (4)

- Example of transitive priority inheritance:



PIP: deadlock



Priority inversion on Mars

- Priority inheritance also solved the Mars Pathfinder problem: the VxWorks operating system used in the pathfinder implements a flag for the calls to mutex primitives. This flag allows priority inheritance to be set to “on”. When the software was shipped, it was set to “off”.

The problem on Mars was corrected by using the debugging facilities of VxWorks to change the flag to “on”, while the Pathfinder was already on the Mars [Jones, 1997].