Chapter 11

Resource Access Protocols

Objectives

What are you about to learn?

Knowledge Objectives

- Understand the problems that come with *dependent* task sets.
- Be able to explain the phenomenon of priority inversion.
- Understand the two resource access protocol “priority inheritance protocol” and “priority ceiling protocol”.
- Know about some scheduling anomalies in dependent task sets.

Skill Objectives

- Ability to define priority ceilings for a given task set.
- Ability to draw the plan for a dependent task set using either the priority inheritance mechanism or priority ceiling protocol, and the RM algorithm.

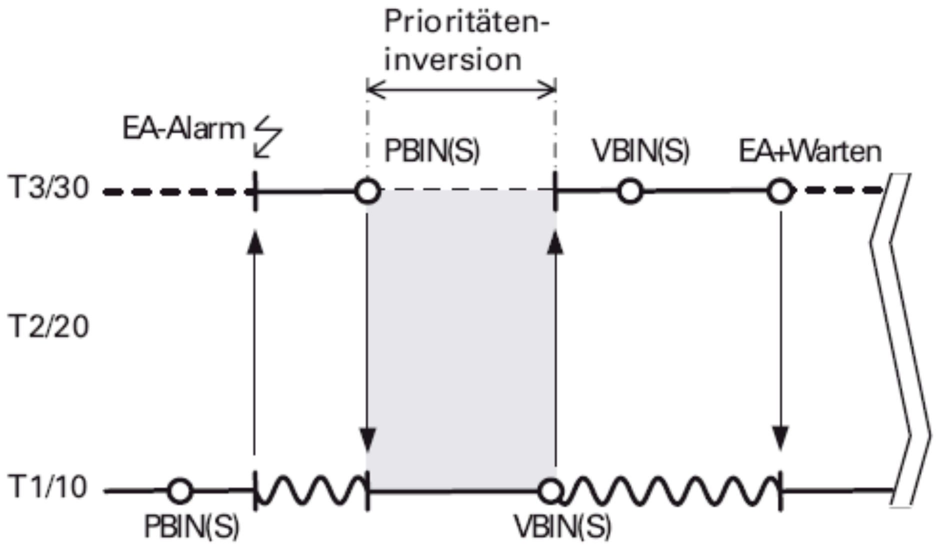
11.1 Scheduling Dependent Task Sets

Tasks become dependent on each other if they access the same resource (beside the processing unit itself).

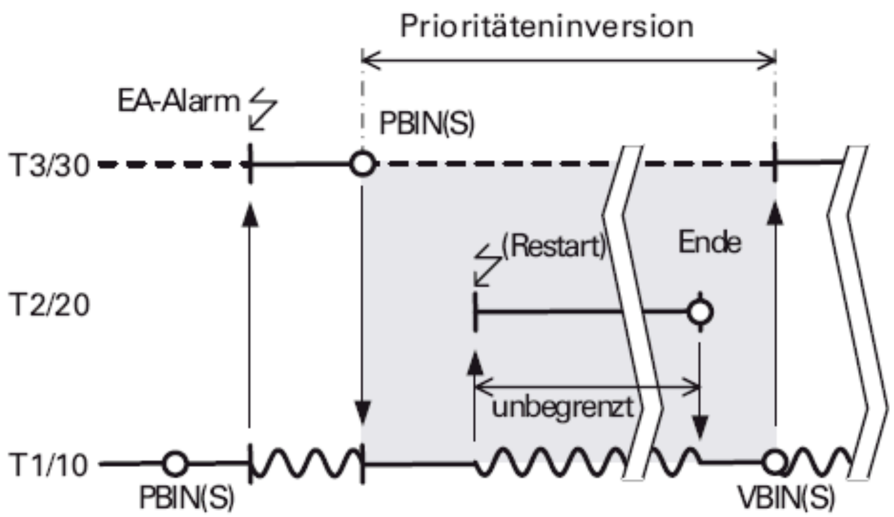
Scheduling dependent task sets can introduce additional challenges. One problem that appears with priority based scheduling is priority inversion.

Priority inversion occurs when a low priority task T1 locks a resource that is then requested by a high priority task T3 (bounded priority inversion). T1 will block T3 via the shared resource, which is the same as if T1 has a higher priority than T3 (inversion of priority).

The priority inversion becomes unbounded when a third task T2 with a priority between those of T1 and T3 preempts T1. This is also called a livelock.



(a) begrenzte Prioritäteninversion



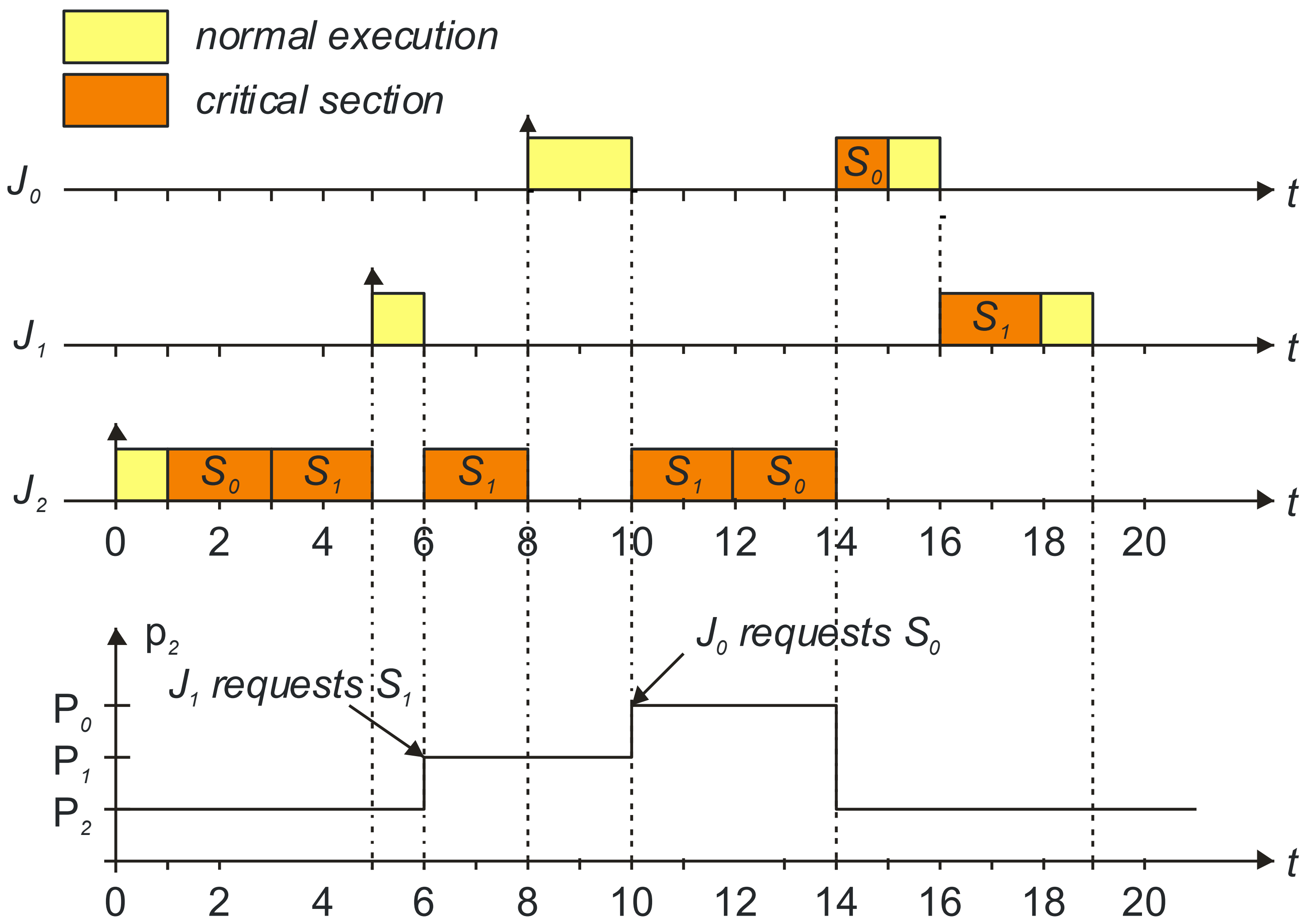
(a) unbegrenzte Prioritäteninversion

11.2 The Priority Inheritance Protocol

The priority inheritance protocol can be defined as follows:

- Jobs are scheduled based on their active priorities. Jobs with the same priority are executed on a first come first serve basis.
- When job J_i tries to enter a critical section $z_{i,j}$ and resource $R_{i,j}$ is already held by a lower priority job, J_i will be blocked. J_i is said to be blocked by the task that holds the resource. Otherwise, J_i enters the critical section $z_{i,j}$.
- When a job J_i is blocked on a semaphore, it transmits its active priority to the job, say J_k, that holds that semaphore. Hence, J_k resumes and executes the rest of its critical section with priority $p_k = p_i$. J_k is said to inherit the priority of J_i. In general, a task inherits the highest priority of the jobs blocked by it.
- When J_k exits a critical section, it unlocks the semaphore, and the highest-priority job, if any, blocked on that semaphore is awakened. Moreover, the active priority of J_k is updated as follows: if no other jobs are blocked by J_k, p_k is set to its nominal priority P_k, otherwise it is set to the highest priority of the jobs blocked by J_k.
- Priority inheritance is transitive; that is if a job J_3 blocks a job J_2, and J_2 blocks a job J_1, then J_3 inherits the priority of J_1 via J_2.

See the following example with three jobs with priorities $P_0 > P_1 > P_2$.



normal execution
critical section
J_0
S_0
t
J_1
S_1
t
J_2
S_0
S_1
S_1
S_1
S_0
t
0 2 4 6 8 10 12 14 16 18 20
p_2
J_0 requests S_0
J_1 requests S_1
P_0
P_1
P_2
t
0 2 4 6 8 10 12 14 16 18 20

11.2 The Priority Inheritance Protocol

Let B_i be the maximum blocking time, due to lower-priority jobs, that a job J_i may experience. Then a set of n periodic tasks using the priority inheritance protocol can be scheduled by the rate monotonic algorithm if

$$\forall i, 1 \le i \le n, \sum_{k=1}^{i} \frac{C_k}{T_k} + \frac{B_i}{T_i} \le i(2^{1/i} - 1)$$

This can be simplified to a less tight condition

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \max\left(\frac{B_1}{T_1}, \ldots, \frac{B_n}{T_n}\right) \le n(2^{1/n} - 1)$$

Example:

	C_i	T_i	B_i
J_1	1	2	1
J_2	1	4	1
J_3	2	8	0

In this case the periods of the tasks are harmonic, the utilization bound for rate monotonic scheduling becomes 100%.

11.3 The Priority Ceiling Protocol

The priority ceiling protocol can be defined as follows:

- Each semaphore S_k is assigned a priority ceiling $C(S_k)$ equal to the priority of the highest-priority job that can lock it. $C(S_k)$ is a static value that can be computed off-line.
- Let J_i be the job with the highest priority among all jobs ready to run; thus J_i is assigned the processor.
- Let S^* be the semaphore with the highest priority ceiling among all the semaphores currently locked by jobs other than J_i, and let $C(S^*)$ be its ceiling.
- To enter a critical section guarded by a semaphore S_k, J_i must have a priority higher than $C(S^*)$. If $P_i \leq C(S^*)$, the lock on S_k is denied and J_i is said to be blocked on semaphore S* by the job that holds the lock on S^*.
- When a job J_i is blocked on a semaphore, it transmits its priority to the job, say J_k, that holds that semaphore. Hence, J_k resumes and executes the rest of its critical section with the priority of J_i. J_k is said to inherit the priority of J_i.
- When J_k exits a critical section, it unlocks the semaphore and the highest-priority job, if any, blocked on that semaphore is awakened. Moreover, the active priority of J_k is updated as follows: if no other jobs are blocked by J_k, p_k ist set to the nominal priority P_k; otherwise, it is set to the highest priority of the jobs blocked by J_k.
- Priority inheritance is transitive; that is if a job J_3 blocks a job J_2, and J_2 blocks a job J_1, then J_3 inherits the priority of J_1 via J_2.

11.3 The Priority Ceiling Protocol

See the following example with three jobs with priorities $P_0 > P_1 > P_2$.

- Job J_0 sequentially accesses two critical sections guarded by semaphores S_0 and S_1
- Job J_1 accesses only a critical section guarded by semaphore S_2
- Job J_2 uses semaphore S_2 and then makes a nested access to S_1.

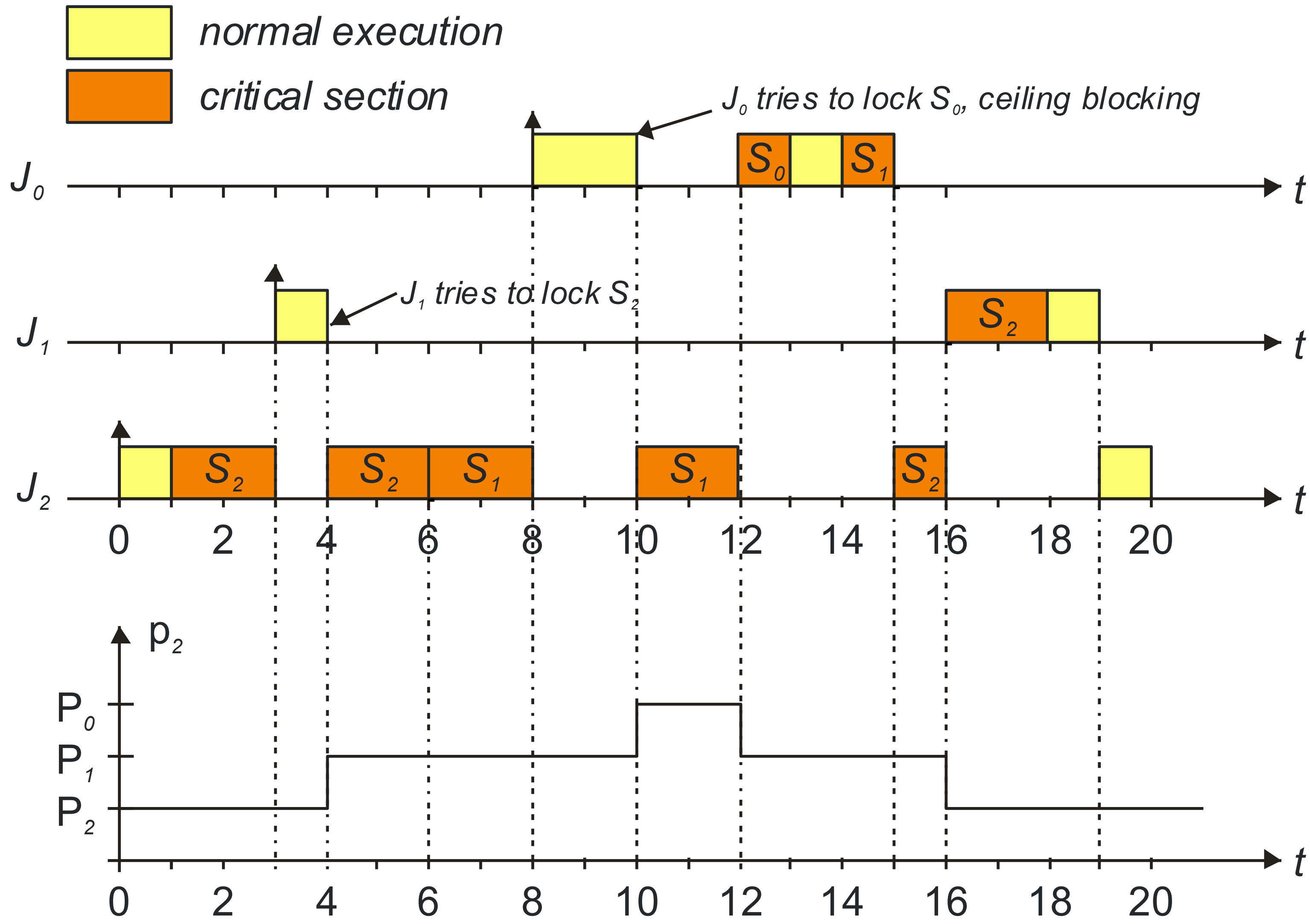
The semaphores are thus assigned the following priority ceilings:

- $C(S_0) = P_0$, since J_0 is the highest-priority task that tries to lock S_0
- $C(S_1) = P_0$, since J_0 is the highest-priority task that tries to lock S_1
- $C(S_2) = P_1$, since J_1 is the highest-priority task that tries to lock S_2

The activation times of the three jobs are as follows:

- J_0 is activated at $t = 8$
- J_1 is activated at $t = 3$
- J_2 is activated at $t = 0$

See the following diagram for the scenario.


normal execution
critical section
J_0 tries to lock S_0, ceiling blocking
J_1 tries to lock S_2
J_0
J_1
J_2
S_0
S_1
S_2
S_2
S_2
S_1
S_1
S_2
t
0 2 4 6 8 10 12 14 16 18 20
p_2
P_0
P_1
P_2

11.3 The Priority Ceiling Protocol

Scenario description:

- At time 0, J_2 is activated, and since it is the only job ready to run, it starts executing and later locks semaphore S_2.
- At time 3, J_1 becomes ready and preempts J_2.
- At time 4, J_1 attempts to lock S_2, but it is blocked by the protocol because P_1 is not greater than $C(S_2)$. Then, J_2 inherits the priority of J_1 and resumes its execution.
- At time 6, J_2 successfully enters its nested critical section by locking S_1. Note that J_2 is allowed to lock S_1 because no semaphores are locked by other jobs.
- At time 8, while J_2 is executing at a priority $p_2 = P_1$, J_0 becomes ready and preempts J_2 because $P_0 > p_2$.
- At time 10, J_0 attempts to lock S_0, which is not locked by any job. However, J_0 is blocked by the protocol because its priority is not higher than $C(S_1)$, which is the highest ceiling among all semaphores currently locked by the other jobs. Since S_1 is locked by J_2, J_2 inherits the priority of J_0 and resumes its execution.
- At time 12, J_2 exits its nested critical section, unlocks S_1, and, since J_0 is awakened, J_2 returns to priority $p_2 = P_1$. At this point, $P_0 > C(S_2)$; hence, J_0 preempts J_2 and executes until completion.
- At time 15, J_0 is completed, and J_2 resumes its execution at a priority $p_2 = P_1$.
- At time 16, J_2 exits its outer critical section, unlocks S_2, and, since J_1 is awakened, J_2 returns to its nominal priority P_2. At this point, J_1 preempts J_2 and executes until completion.

- At time 19, J_1 is completed; thus J_2 resumes its execution.

Blocking time computation of the priority ceiling protocol:

The maximum blocking time of B_i of a job J_i can be computed as the duration of the longest critical section among those belonging to tasks with priority lower than P_i and guarded by a semaphore with ceiling higher than or equal to P_i. If $D_{j,k}$ denotes the duration of the longest critical section of task τ_j among those guarded by semaphore S_k, we can write

$$B_i = \max_{j,k}\left\{D_{j,k} \mid P_j < P_i, C(S_k) \geq P_i\right\}$$

Example: for each job J_i, the duration of the longest critical section among those guarded by the semaphore S_k is denoted by $D_{i,k}$ and is stored in a table. $D_{i,k}$ = 0 means that job J_i does not use semaphore S_k. Semaphore ceilings are in parentheses:

	$S_1\ (P_1)$	$S_2\ (P_1)$	$S_3\ (P_2)$
J_1	1	2	0
J_2	0	9	3
J_3	8	7	0
J_4	6	5	4

The tasks' blocking factors are computed as follows

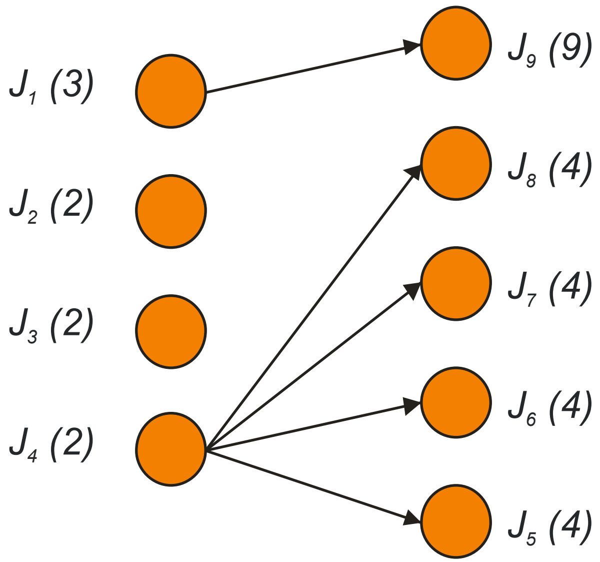
$$
\begin{cases}
B_1 = \max(8,6,9,7,5) = 9 \\
B_2 = \max(8,6,7,5,4) = 8 \\
B_3 = \max(6,5,4) = 6 \\
B_4 = 0
\end{cases}
$$

11.4 Scheduling Anomalies in Dependent Task Sets

When dealing with task sets with precedence relations executed in a multiprocessor environment we encounter some scheduling anomalies:

- adding resources (such as a processor) can make things worse
- relaxing constraints such as less precedence between tasks or lower execution time requirements can make things worse

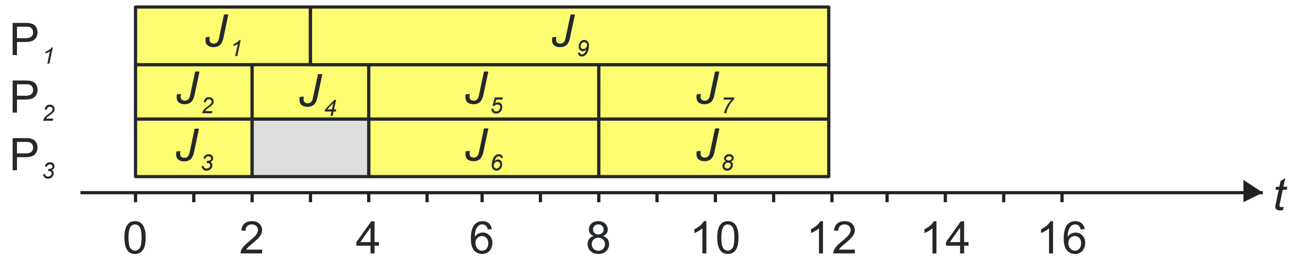
This is illustrated in the following example with nine tasks and the following precedence relationships (values in parentheses are execution times):



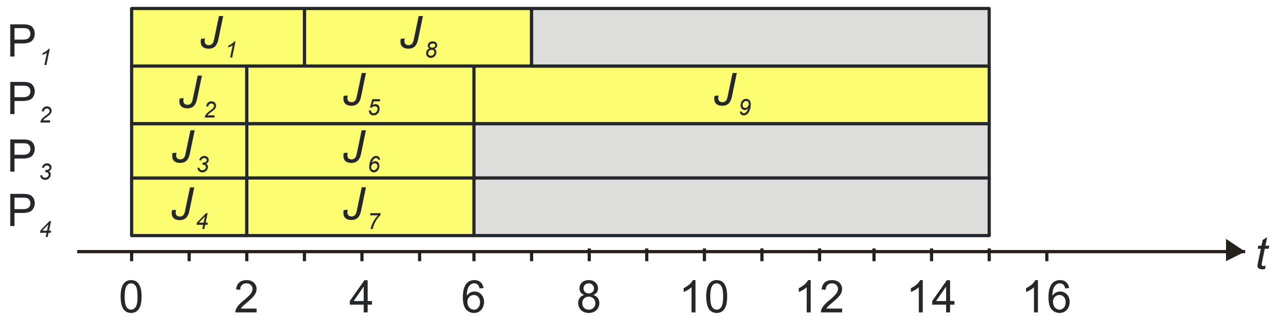
$priority(J_i) > priority(J_j) \quad \forall\, i < j$

values in parantheses are execution times

First case: optimal schedule on three processors P_1, P_2, P_3. Global completion time is 12.



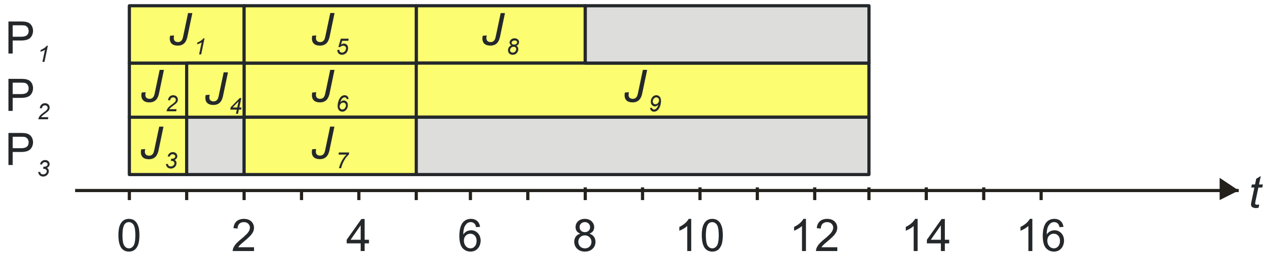
Second case: we add a processor P_4. Tasks are allocated to the first available processor. Global completion time has increased to 15.



Increase to four processors
Allocation to the first available processor

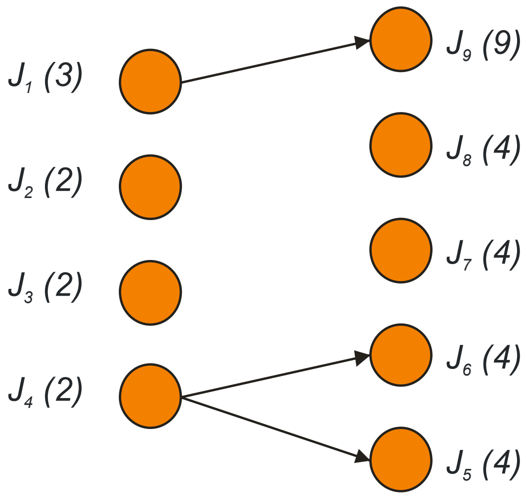
11.4 Scheduling Anomalies in Dependent Task Sets

Third case: we reduce all computation times by one time unit. Tasks are allocated to the first available processor. Global completion time has increased to 13.



Computation time reduced by one time unit

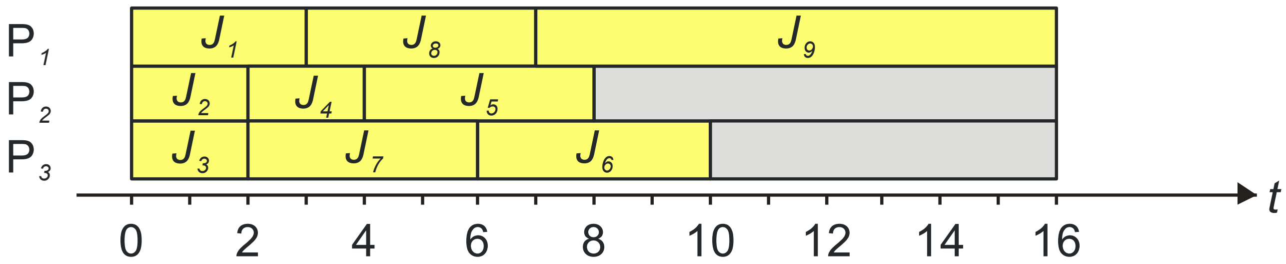
11.4 Scheduling Anomalies in Dependent Task Sets



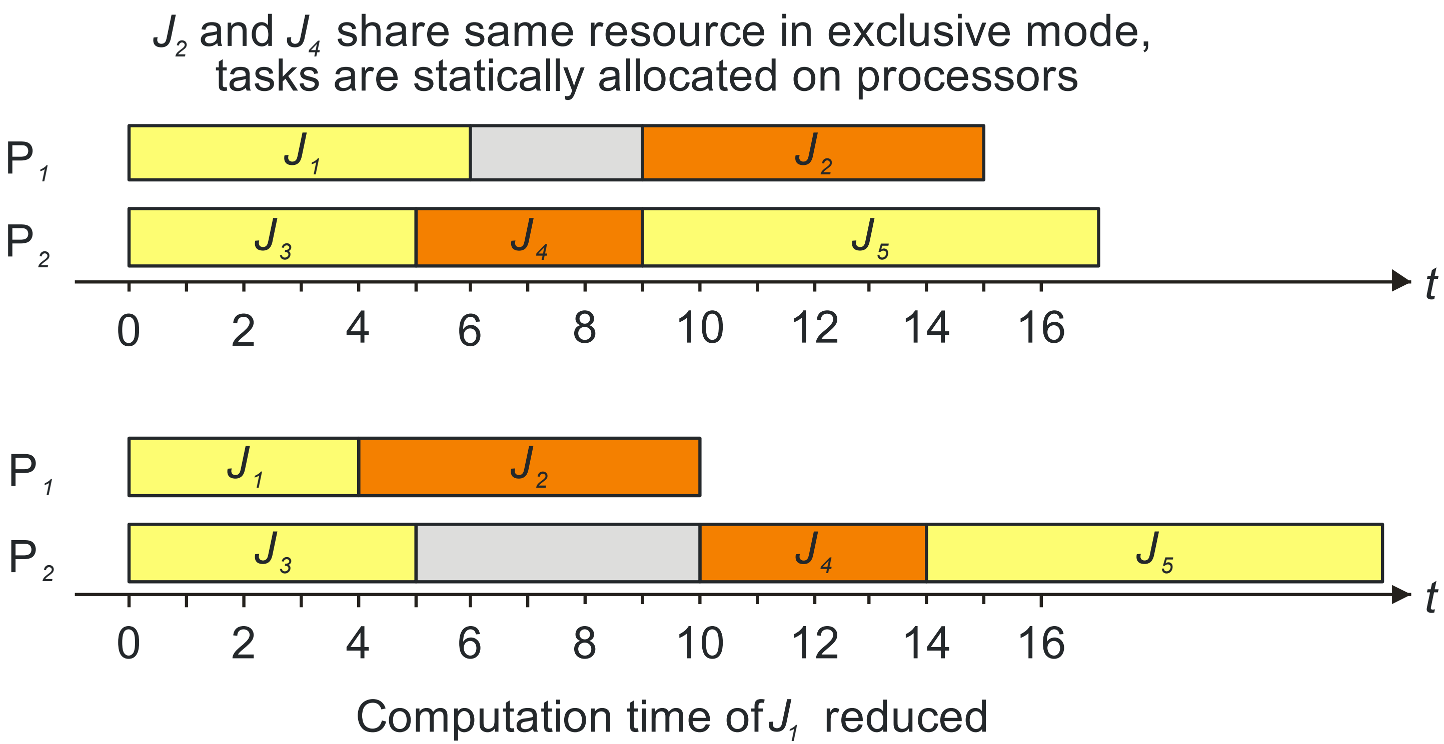
priority(J_i) > priority(J_j) $\forall$ i < j

values in parentheses are execution times

Fourth case: we remove some precedence constraints. Global completion time has increased to 16.



Fifth case: we share a resource in exclusive mode, and reduce computation time. Global completion time increases from 16 to 22.



Points to Remember