Real-Time Systems

Michael Roitzsch

Resource Access Protocols

29/11/12

TECHNISCHE UNIVERSITÄT DRESDEN

Problems: Priority Inversion (1)

Assumptions:

Jobs use resources in a mutually exclusive manner (critical section CS).
Preemptive priority-driven scheduling, fixed priorities.

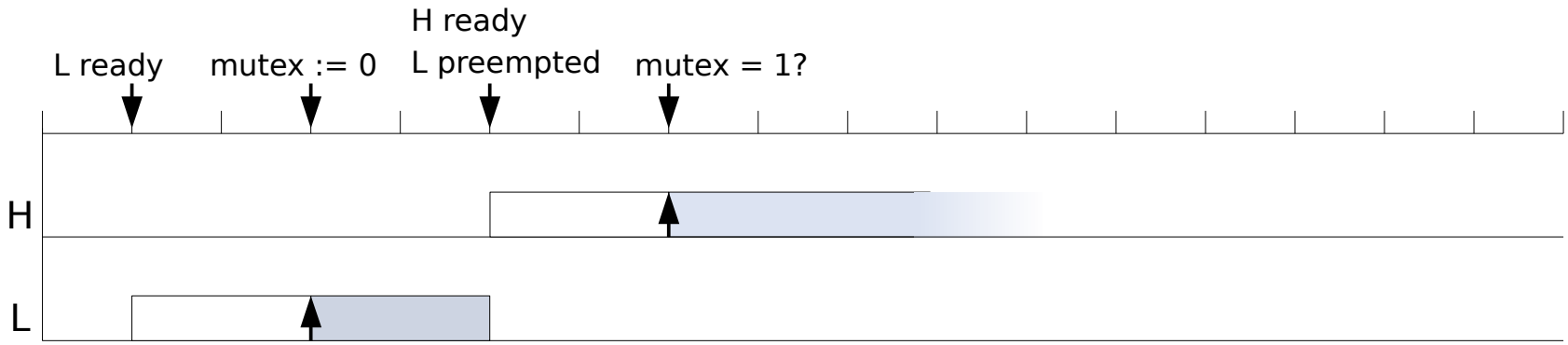
1 processor.

Declaration

for the following slides

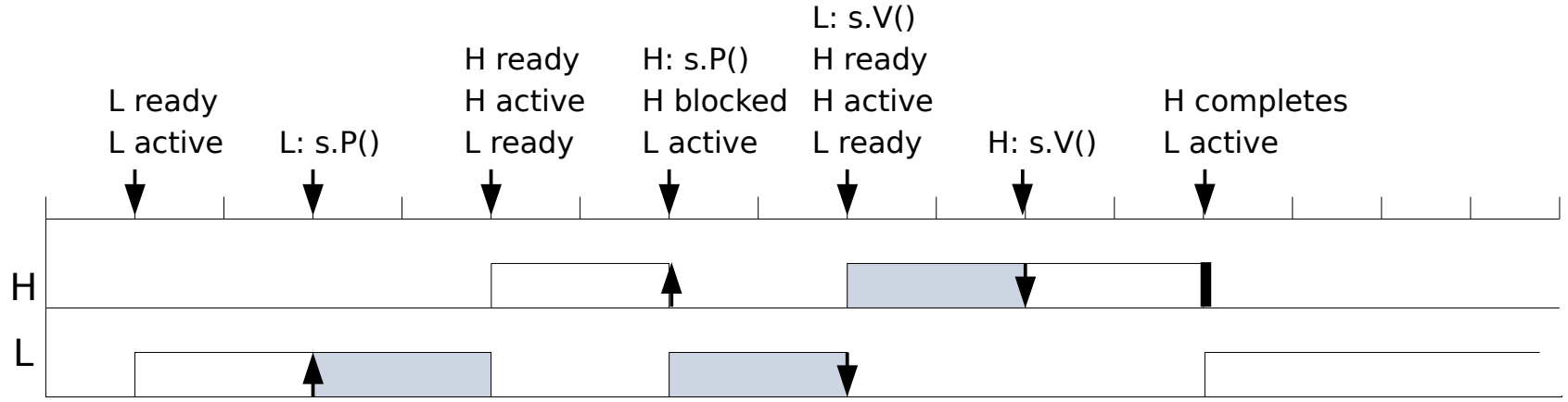
L(R) ↑ U(R) ↓

- Busy waiting and priority inversion

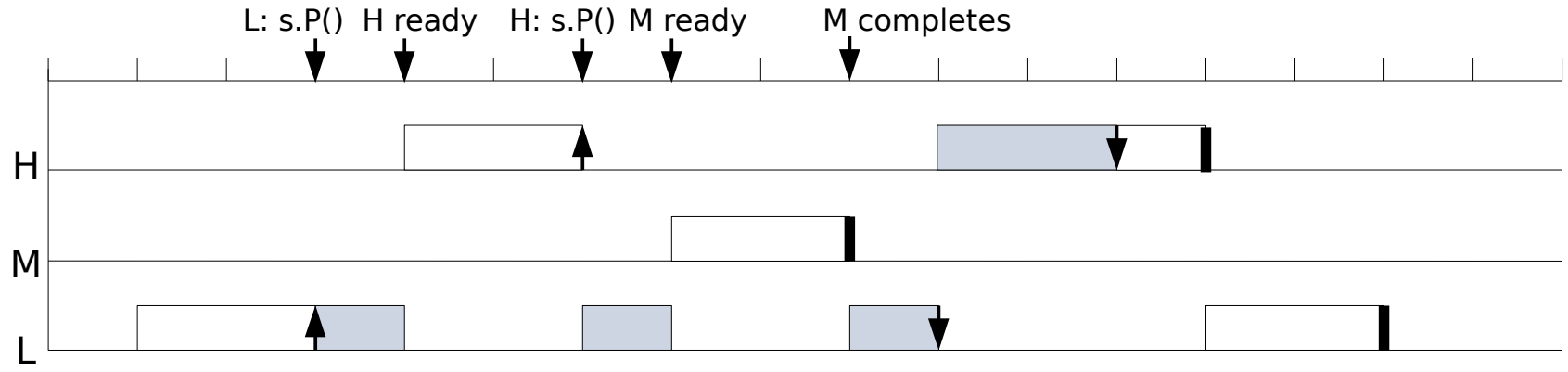


Problems: Priority Inversion (2)

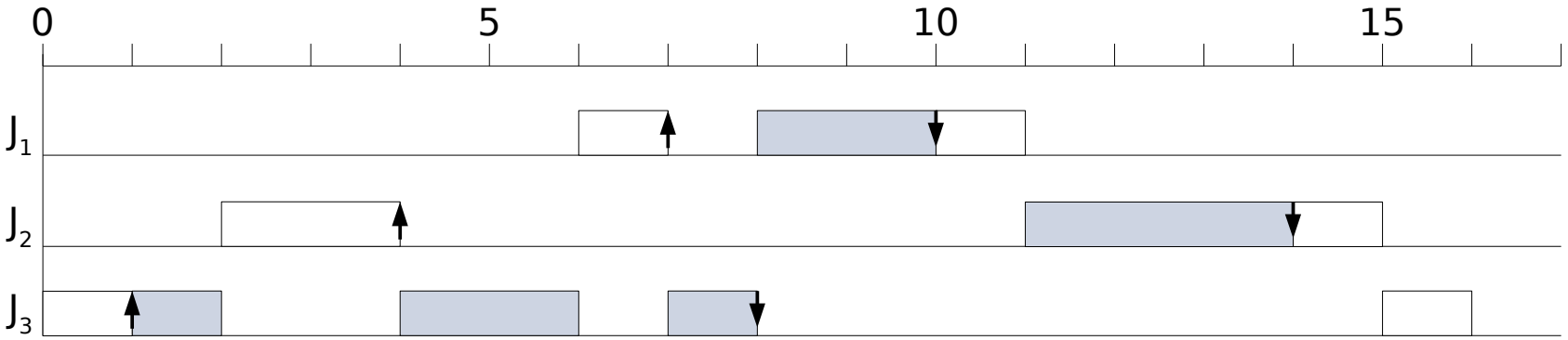
- Semaphores and priority inversion



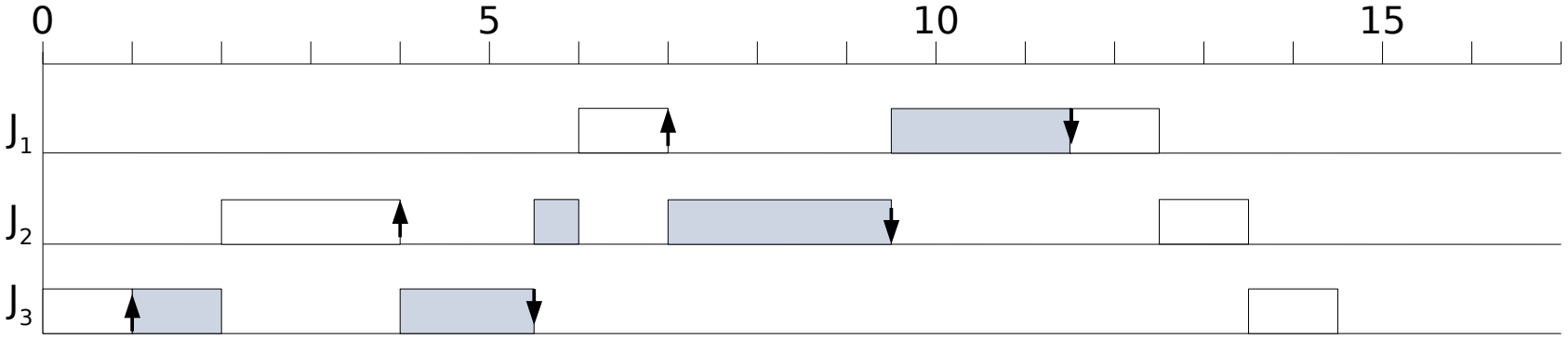
M: medium-prioritized job (not using *s*)



Problems: Timing Anomalies



Reduction of resource usage of J_3 by 1.5:



Problems: Deadlocks

- Deadlocks

Assumptions and Notations

1 processor, preemptive priority-driven scheduling,
jobs are not self-suspending

- $\boldsymbol{R_1,\ldots,R_r}$ resources; nonpreemptable, exclusive

- $\mathbf{L}(\boldsymbol{R_k})$, $\mathbf{U}(\boldsymbol{R_k})$ acquire/release of R_k; release: LIFO
  $\uparrow\boldsymbol{R_k}$ $\downarrow\boldsymbol{R_k}$

- $\boldsymbol{J_1,\ldots,J_n}$ jobs
  $\boldsymbol{J_h, J_l}$: job of high/low priority

  $\boldsymbol{p_1,\ldots,p_n}$ "assigned" priorities (highest priority: 1);
  w.l.o.g. J_i ordered according to priorities

  $\boldsymbol{p_i(t)}$ current priority of J_i

Assumptions and Notations

- **Jobs conflict with one another**
  operate with the a common resource

- **Jobs contend for a resource**
  one job requests the resource that another job already has

- **Blocked job**
  scheduler does not grant the requested resource

- **Priority inversion**
  J_l executes while J_h is blocked

Priority Inheritance - Protocol

for preemptive priority-driven scheduling Sha et al., 1990

- **Basic Priority-Inheritance Protocol**

  (1) ***Scheduling Rule***

  A ready job J is scheduled according to its current priority $p(t)$;
  at release time t: $p(t) := p$.

  (2) ***Allocation Rule***

  J requests R at time t.

  (a) R free: R is allocated to J until J releases R.

  (b) R not free: request is denied, J becomes blocked.

  (3) ***Priority-Inheritance Rule***

  When J becomes blocked by J_l, then J_l inherits the current priority of J, i.e. $p_l(t) := p(t)$.

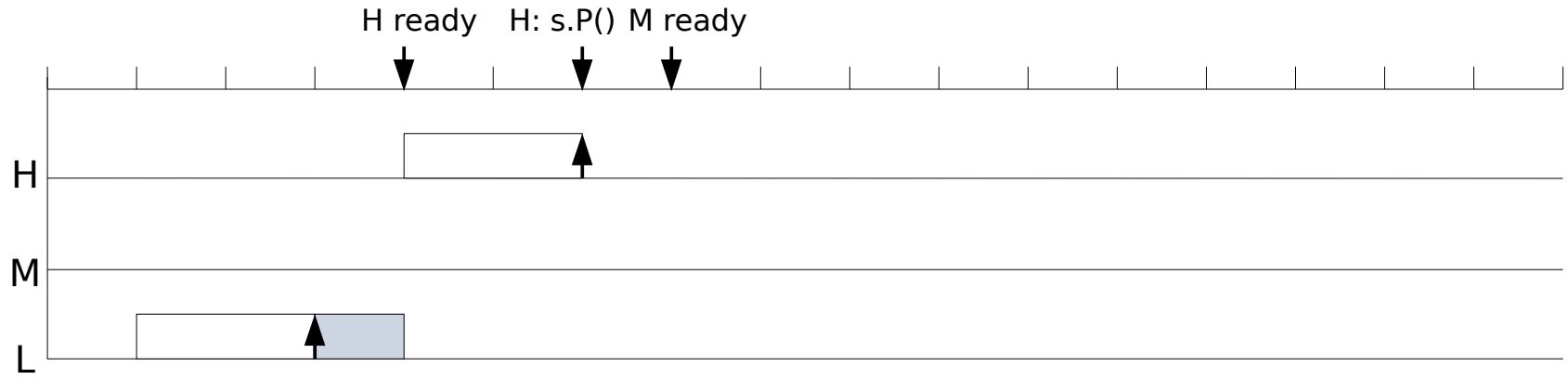
  J_l executes at this priority until it releases R at time $\tilde{t}$.

  Now the priority of J_l returns to its previous priority:

  $p_l(\tilde{t}) := p_l(t')$ t': time when J_l acquires R.

Priority Inheritance - Example

- 2 jobs: no effect!
- 3 jobs:

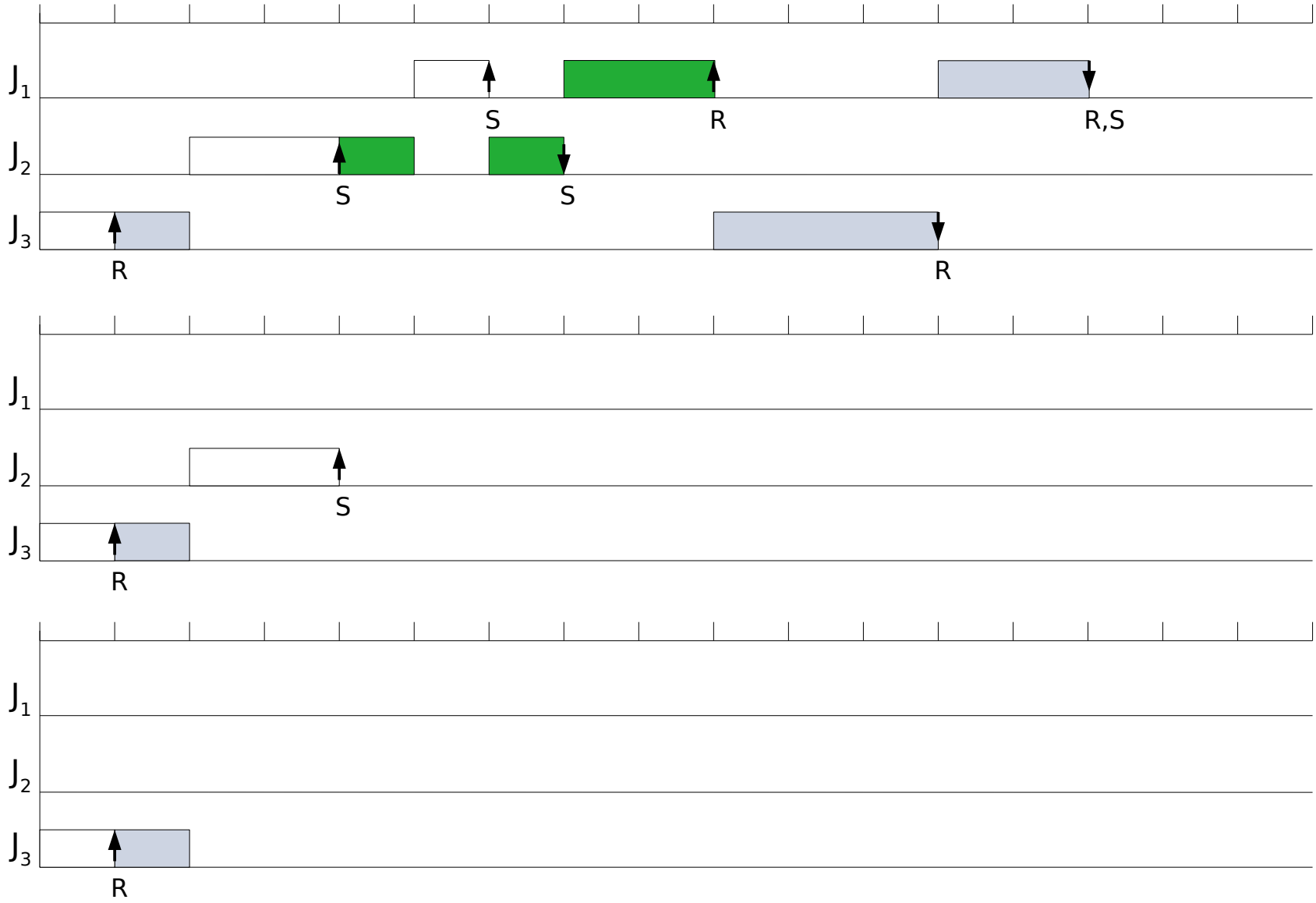


Priority Inheritance – Properties (1)

Properties

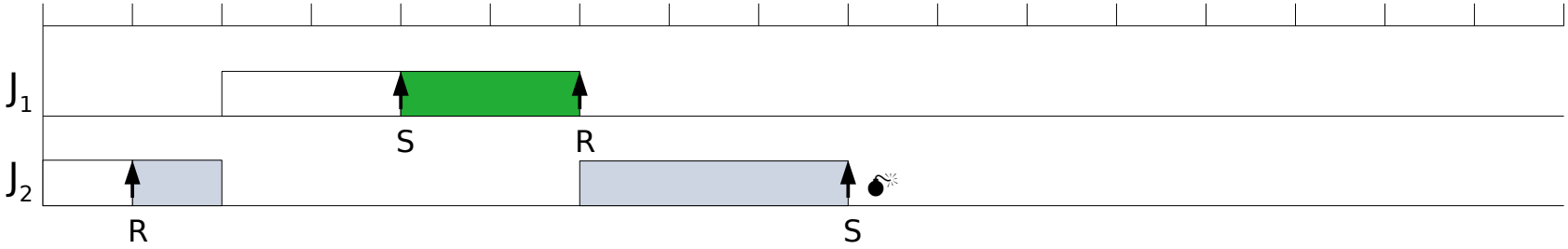
- Priority inheritance is transitive.
- No unbounded uncontrolled priority inversion.
- Priority inheritance does not reduce the blocking times for the high prioritized jobs as much as possible.

Priority Inheritance – Properties (2)



Priority Inheritance – Properties (3)

Priority inheritance does not prevent deadlocks.



Priority Ceilings - Notations
Basic Priority-Ceiling Protocol

Sha/Rajkumar/Lehoczky, 1990

- **Assumptions and Notations**
  - 1 processor, preemptive priority-driven scheduling
    no self-suspension.
  - Assigned priorities p_i are fixed.
    priorities: natural numbers, 1 highest, Ω lowest priority.
  - The resources required by all jobs are known a priori.

- ***P(R)*** **priority ceiling of *R***
  highest priority of all jobs that require R.

- $\hat{P}(t)$ **priority ceiling of the system at time *t***
  highest priority ceiling of all resources that are in use at time t.

Priority Ceilings - Protocol
Basic Priority-Ceiling Protocol

(1) ***Scheduling Rule***

At release time t^{rel} of J: $p(t^{rel}) := p$.

(2) ***Allocation Rule***

J requests R at time t.

(a) R held by another job: request denied, J blocked ("on R").

(b) R free:

(α) $p(t) \succ \hat{P}(t)$: R is allocated to J.

(β) otherwise: R is allocated to J only if J is the job holding the resource(s) R' with $P(R') = \hat{P}(t)$.
Otherwise the job becomes blocked.

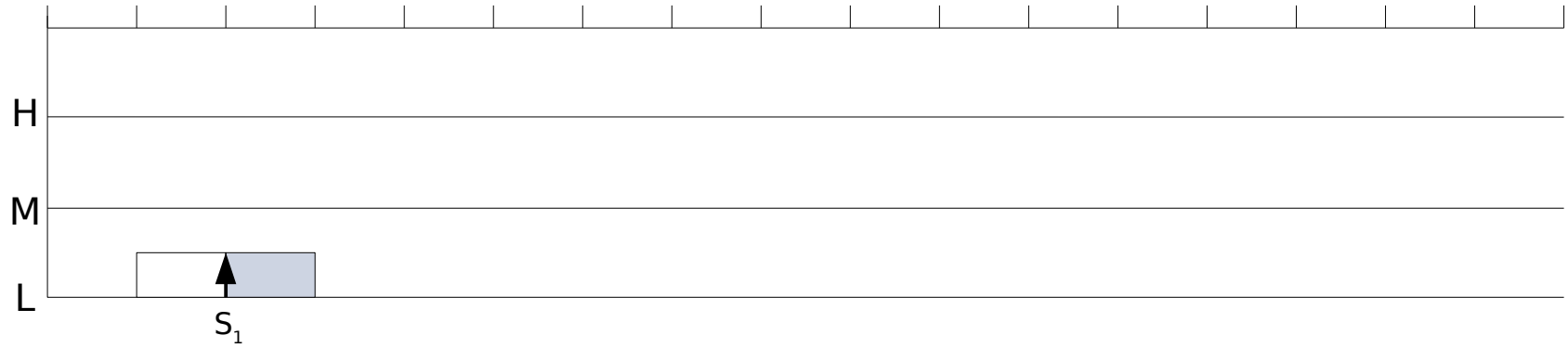
(3) ***Priority-Inheritance Rule***

When J becomes blocked by J_l, J_l inherits J's current priority $p(t)$.

J_l (preemptively) executes at this priority until it releases every resource whose priority ceiling is at least $p(t)$.

At that time, J_l's priority returns to $p_l(t')$
(t': time when it was granted the resource).

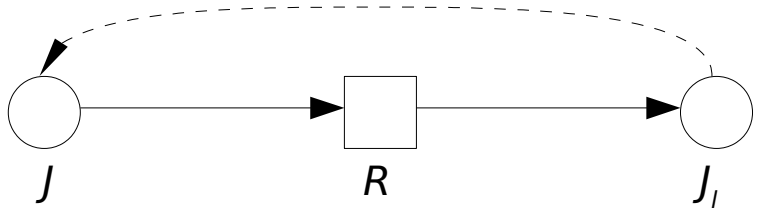
Priority Ceilings - Example
Basic Priority-Ceiling Protocol



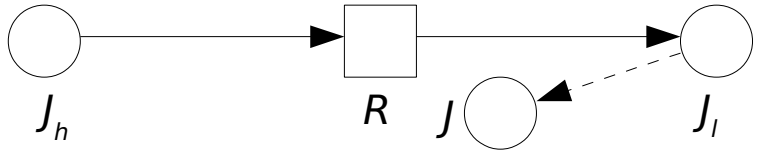
Priority Ceilings - Properties
Basic Priority-Ceiling Protocol

- **Properties**
  - Difference to priority inheritance: *three* ways to blocking

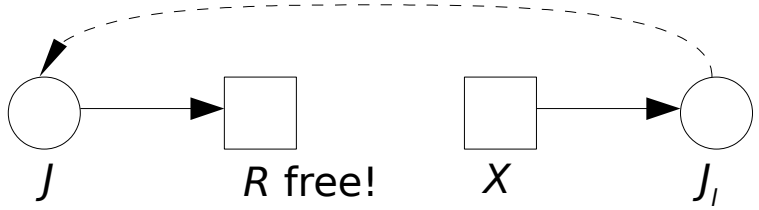
direct:



inheritance:



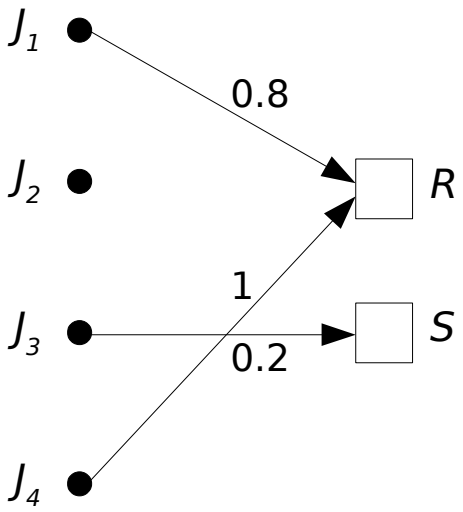
ceilings:

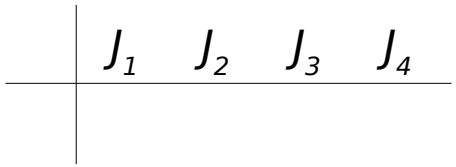


  - Deadlocks can never occur.
  - There can be no transitive blocking.

Priority Ceilings - Example
Basic Priority-Ceiling Protocol

- A job can be blocked for at most *one* resource request.
  Computation of blocking time – Example:





Stack-Based Priority-Ceiling Protocol

- **Further Assumptions**
  - Common run-time stack for all jobs (no self-suspending!).
  - Stack space of an active job is on the top of the stack (preemption!).
  - Stack space is freed when the job completes.

- **Protocol**

  (0) $\hat{P}(t) = \Omega$, when all R are free.

  $\hat{P}(t)$ is updated whenever a resource is allocated or freed.

  (1) ***Scheduling Rule***

  After J is released, it is blocked until $p \succ \hat{P}(t)$.

  Priority-driven scheduling based on assigned priorities. **(!)**

  (2) ***Allocation Rule***

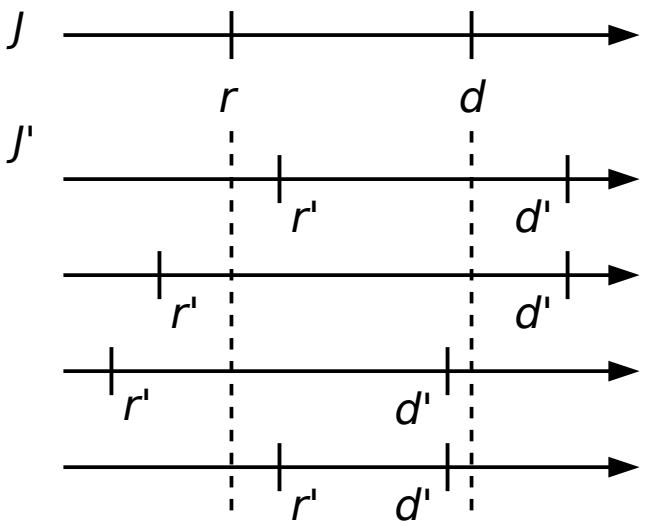
  Whenever a job requests a resource, it is allocated the resource. **(!!)**

- **Properties**
  - When a job begins to execute, all the resources it will ever need are free.
  - Both protocols result in the same longest blocking time of a job.
  - Deadlocks can never occur.

Preemption-Ceiling Protocol (1)
for Deadline-Driven Systems

- **Properties of EDF**
  - Priorities are static on job level using EDF for periodic tasks.
  - Job J with shorter relative deadline can never be preempted by job J' with larger relative deadline

    → J_l can block J_h only when J_h can preempt J_l.



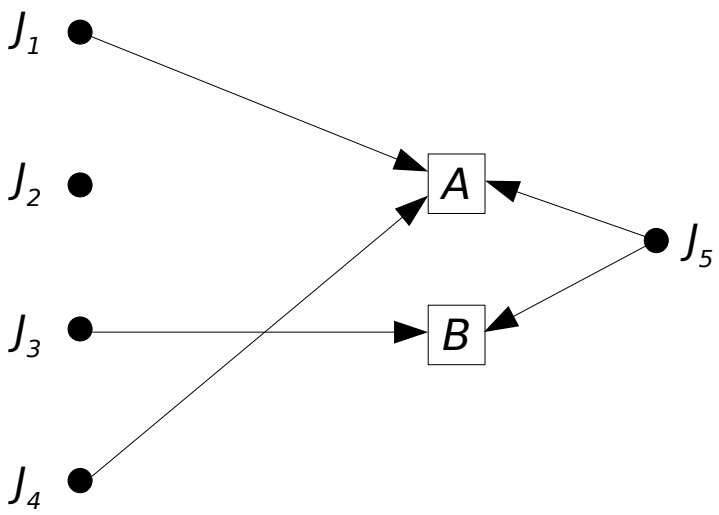
- **Preemption Level (PL) π_J of Job J**
  - ***Requirement***

    It is never possible for J' to preempt J if the PL of J is higher than the PL of J'.
  - ***Validity Condition***

    If $p \succ p'$ and $r > r'$ then $\pi \succ \pi'$.

Preemption-Ceiling Protocol (2)
for Deadline-Driven Systems

- **Example**



Jobs	p_i	r_i	e_i	π_i	π_i'
J_1	1	5	5		
J_2	2	7	3		
J_3	3	4	4		
J_4	4	0	9		
J_5	5	3	4		

Preemption-Ceiling Protocol (3)
for Deadline-Driven Systems

- **EDF-Based Scheduling of Periodic Task Sets**

  A valid preemption-level assignment is according to the relative deadlines d of jobs: the smaller d, the higher π.

  → All jobs in every periodic task in a deadline-driven system have the same preemption level ("fixed preemption-level system").

- **Preemption Ceilings**

  $\Pi(R)$ preemption ceiling of resource R

  $\hat{\Pi}(t)$ preemption ceiling of the system at time t

  analogue priority ceilings

Preemption-Ceiling Protocol (3)
for Deadline-Driven Systems

Stack-Based, Preemption-Ceiling Protocol

(0) $\hat{\Pi}(t) = \Omega$, when all R are free.
$\hat{\Pi}(t)$ is updated whenever a resource is allocated or freed.

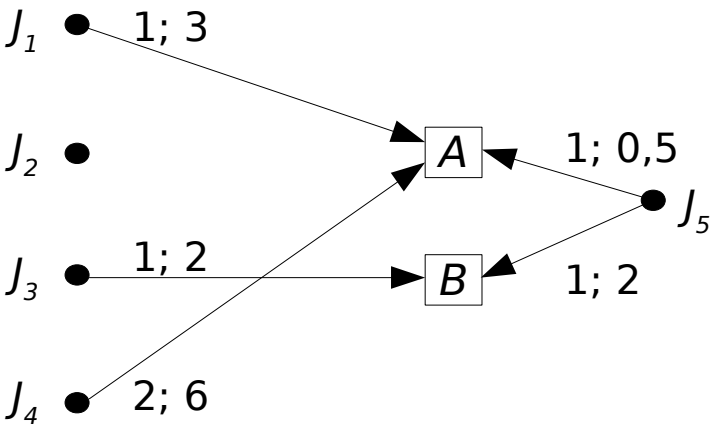
(1) ***Scheduling Rule***
After J is released, it is blocked until
$\pi_J \succ \hat{\Pi}(t)$ and $\pi_J \succ \pi_{J'}$ (J': currently executed job).
Priority-driven scheduling based on “assigned" priorities.

(2) ***Allocation Rule***
Whenever a job requests a resource, it is allocated the resource.

(3) ***Priority-Inheritance Rule***
A job blocking another job from starting inherits the highest priority of all the blocked jobs.

Preemption-Ceiling Protocol (4)

for Deadline-Driven Systems - Example



Jobs	p_i	r_i	d_i	e_i	π_i
J_1		5	13	5	
J_2		7	12	3	
J_3		4	17	4	
J_4		0	24	9	
J_5		3	23	4