Verification of Real-Time Systems
Resource Sharing

Jan Reineke

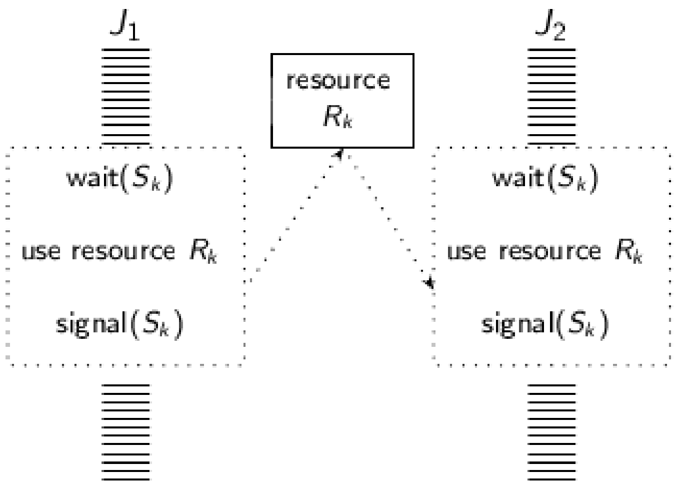
Advanced Lecture, Summer 2015

Resource Sharing

- So far, we have assumed sets of independent tasks.
- However, tasks may share resources
  - to communicate with each other, e.g. through shared memory
  - because resources are sparse, e.g. I/O devices, duplication would be expensive
- Need to ensure mutual exclusion
  - typically by protecting accesses to the shared resource by semaphores

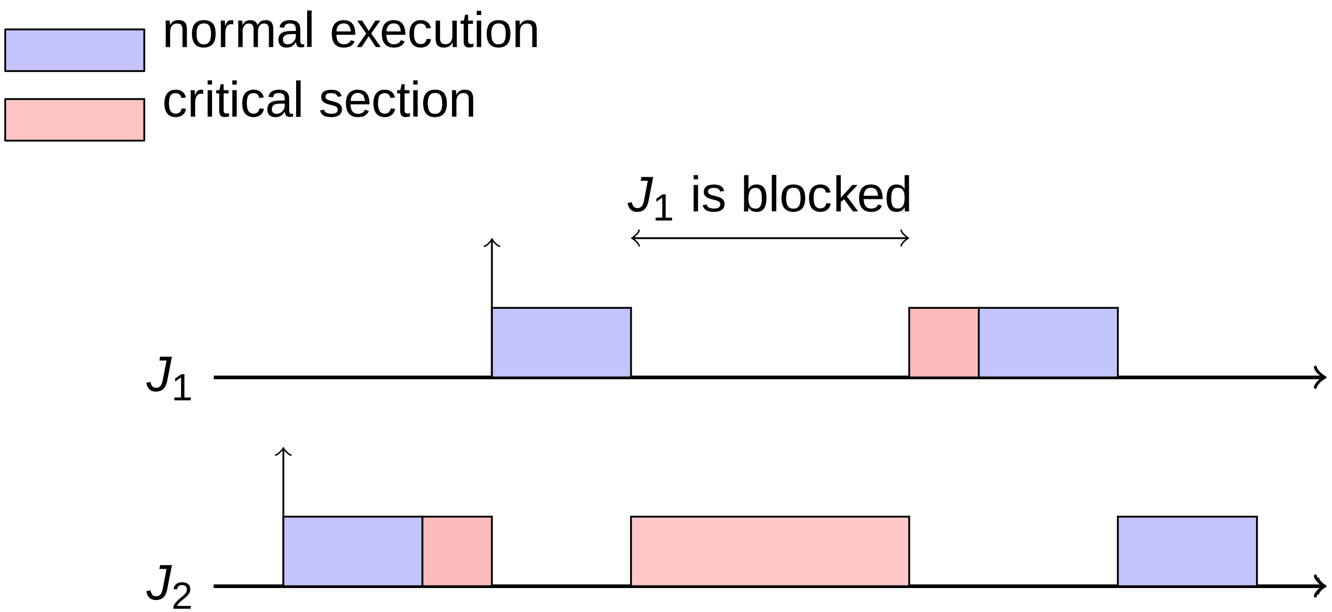
Resource Sharing

- Shared resources:
  - Data structures, variables, main memory area, files, I/O units, the processor, etc.
- Mutual exclusion, critical section
  - When a job enters a critical section of a shared resource, other jobs trying to enter a critical section of the same resource are blocked.

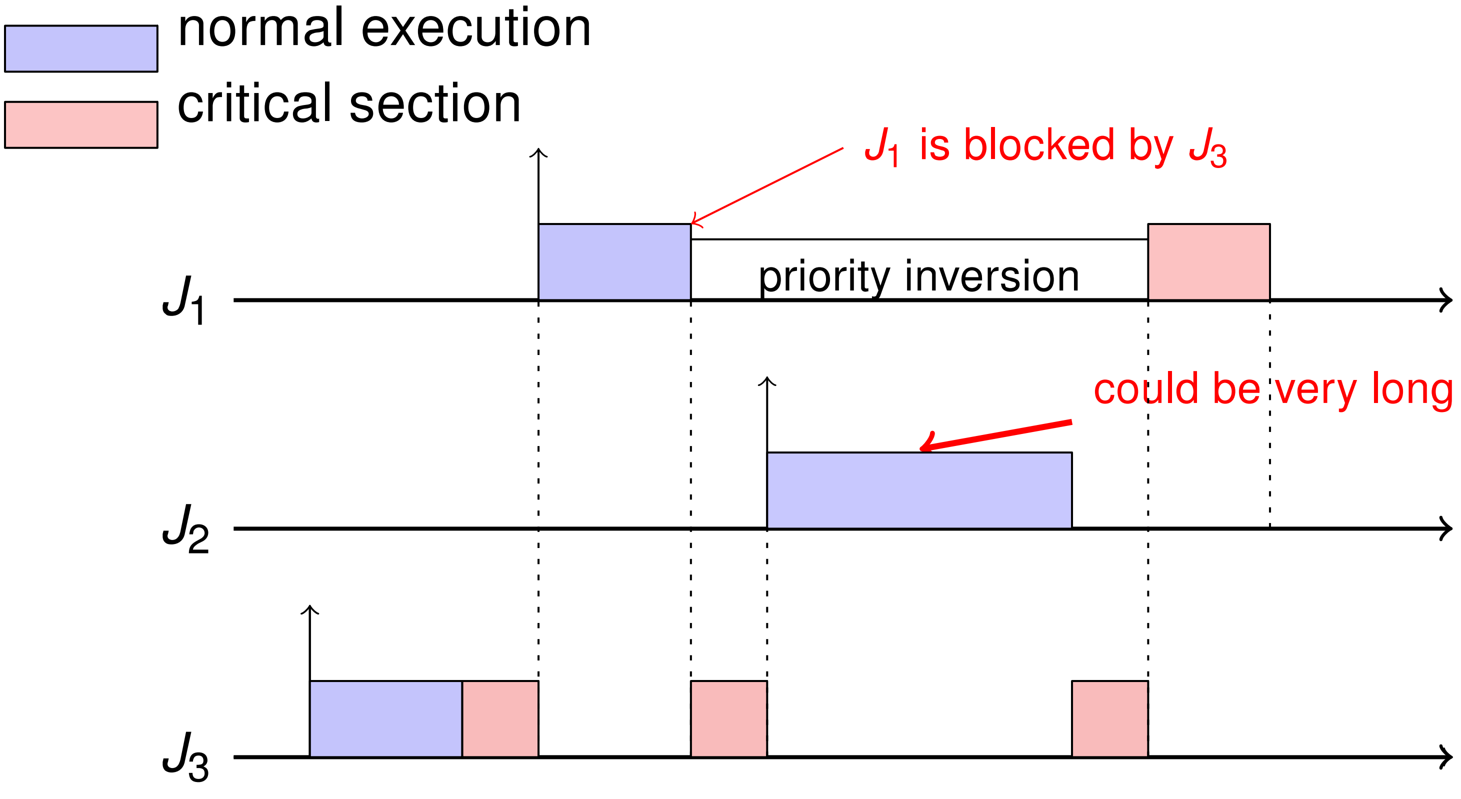


Resource Sharing Affects Scheduling and Schedulability: Priority Inversion

Priority Inversion: a higher priority job is blocked by a lower-priority job.



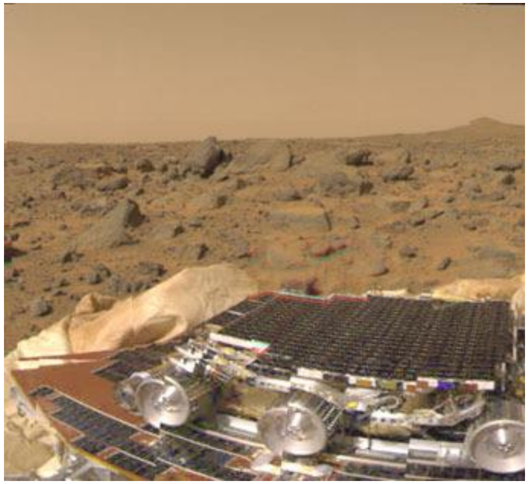
Priority Inversion: Another Example



Priority Inversion in the Real World: Mars Pathfinder

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.



Priority Inversion in the Real World: Mars Pathfinder

"VxWorks provides preemptive priority scheduling of threads. Tasks on the Pathfinder spacecraft were executed as threads with priorities"

"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."

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

Priority Inversion in the Real World: Mars Pathfinder

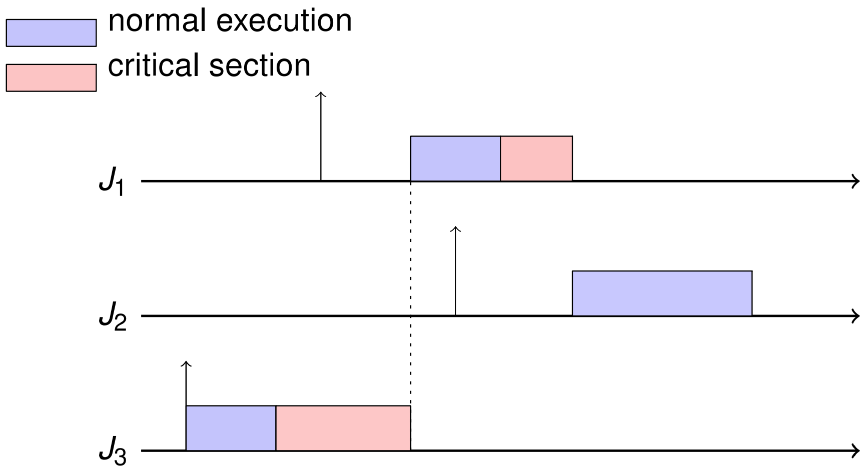
High priority	Medium priority	Low priority
Data retrieval from memory	Communication task	Meteorological data collection

"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."

Naïve solution for Priority Inversion

Disallow preemption during critical sections

- It is simple.
- No deadlocks. Why?
- A high-priority task is blocked for at most one critical section. Why?
- But: it creates unnecessary blocking. Why?



Resource Access Protocols

- Basic Idea:
  - Modify (increase) the priority of those jobs that cause blocking.
  - When a job J_j blocks one or more higher-priority tasks, it temporarily assumes a higher priority.
- Methods:
  - Priority Inheritance Protocol (PIP), for fixed-priority scheduling
  - Priority Ceiling Protocol (PCP), for fixed-priority scheduling
  - Stack Resource Policy (SRP), for both fixed- and dynamic-priority scheduling

Priority Inheritance Protocol (PIP)

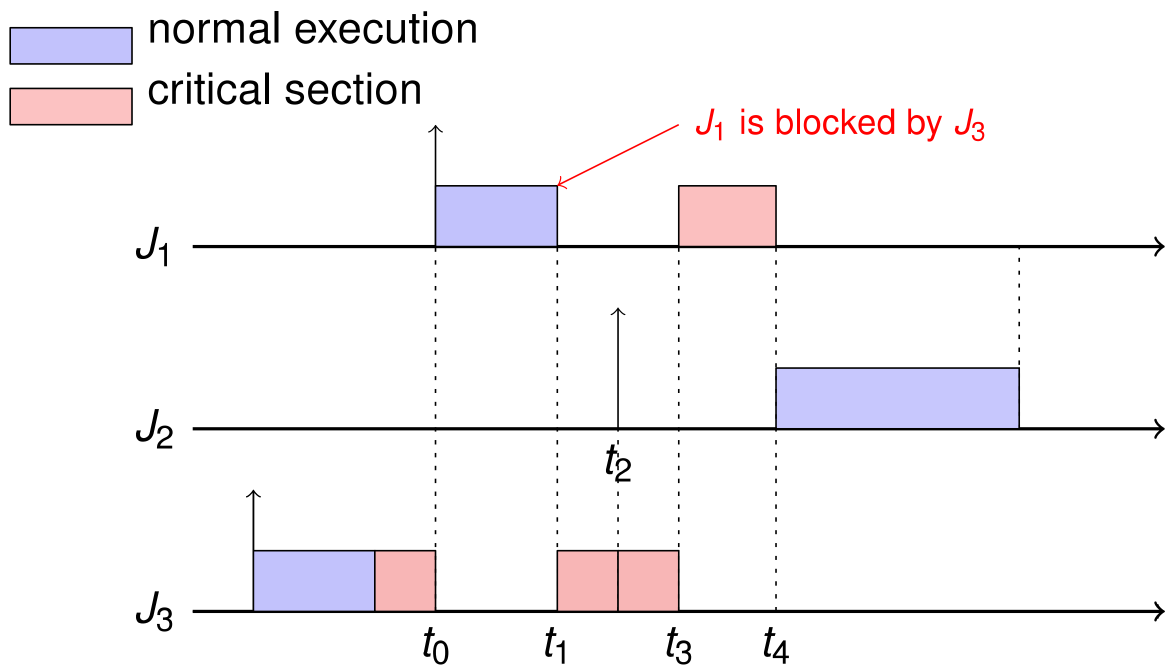
When a lower priority job J_j blocks a higher-priority job, the priority of J_j is promoted to the priority level of the highest-priority job that job J_j blocks.

For example, if the priority order is $J_1 > J_2 > J_3 > J_4 > J_5$,

- When job J_4 blocks jobs J_2 and J_3, the priority of J_4 is promoted to the priority level of J_2.
- When job J_5 blocks J_1 and J_3, its priority level is promoted to the priority level of J_1.

Priority inheritance solved the Mars Pathfinder problem: the VxWorks operating system used in the pathfinder implements priority inheritance. The software was shipped with priority inheritance turned off.

Example of PIP



- t_0: J_1 arrives and preempts J_3
- t_1: J_1 attempts to enter the critical section. J_1 is blocked by J_3 and J_3 inherits J_1's priority
- t_2: J_2 arrives, but has a lower priority than J_3
- t_3: J_3 leaves its critical section, and J_1 now preempts J_3

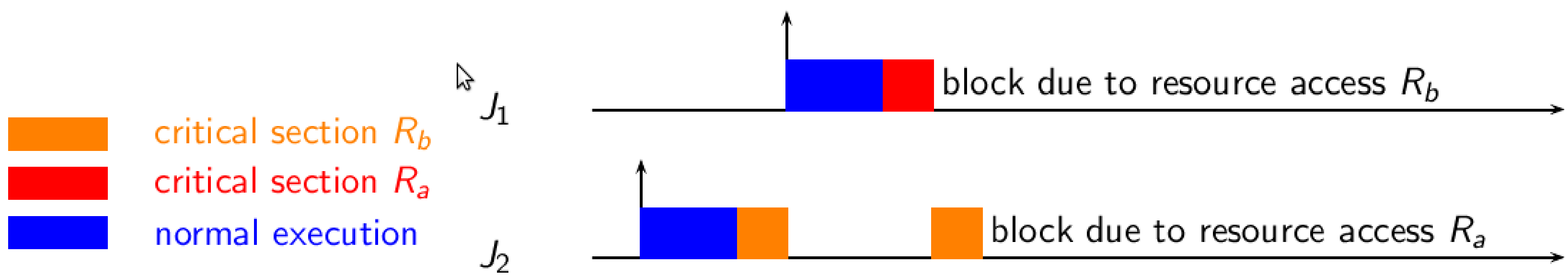
Weaknesses of PIP

Blocking in PIP:

- Direct blocking: higher-priority job tries to acquire a resource held by a lower-priority job
- Push-through blocking: a medium-priority job is blocked by a lower-priority job that has inherited a higher priority

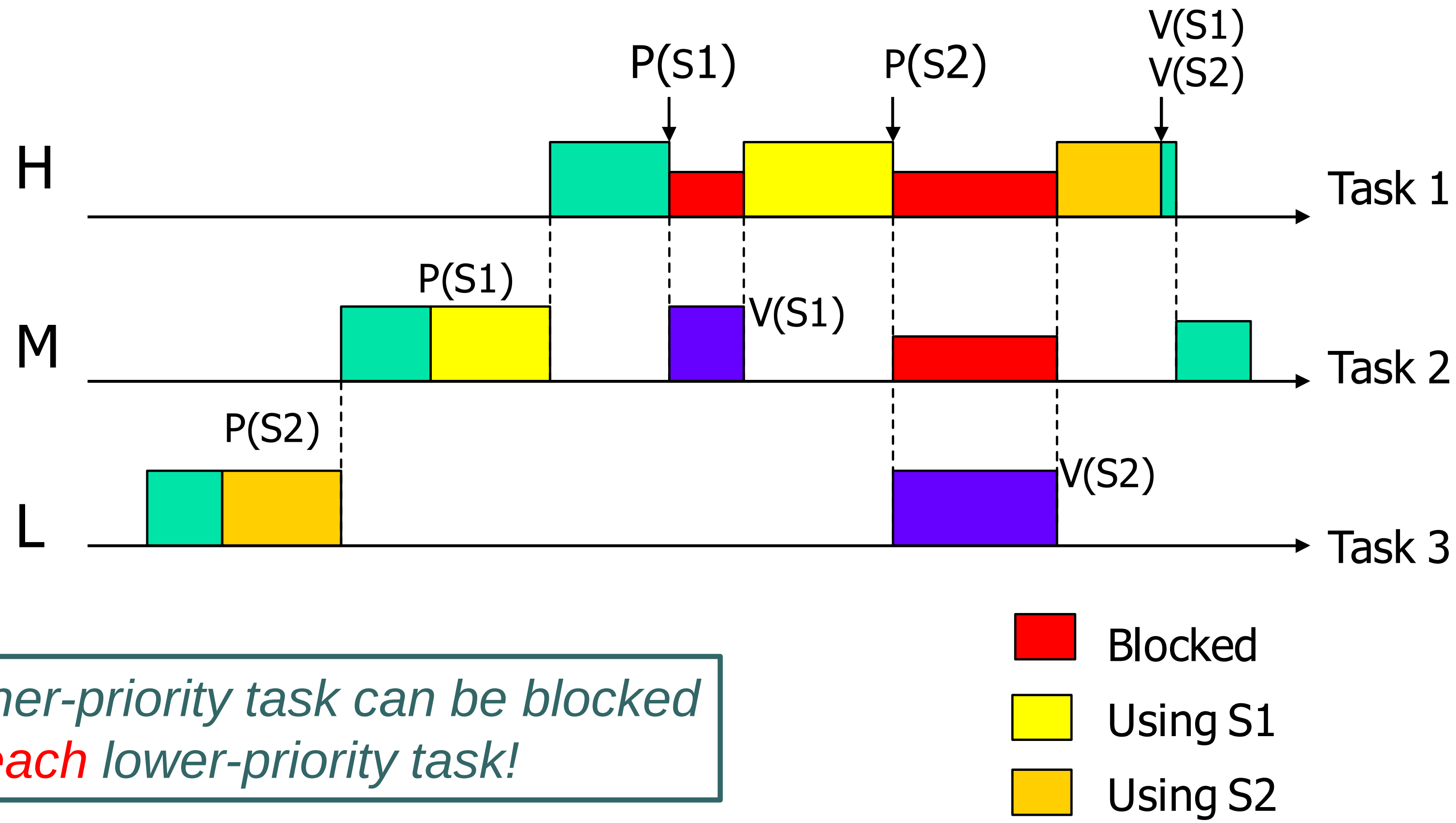
Problems of PIP:

- PIP might cause deadlock if there are multiple resources:



- Under PIP, if there are n lower-priority jobs, a higher-priority job can be blocked for the duration of n critical sections: Chained blocking

PIP: Chained Blocking



Higher-priority task can be blocked by each lower-priority task!

PIP: Blocking-Time Calculation

$Use(S)$ *The set of tasks using semaphore S.*

$CS(k, S)$ *The WCET of the critical section of task k using semaphore S.*

$\Pi(S)$ *The highest priority of the task's using semaphore S.*

The maximal blocking time of task i under PIP:

$$B_i^{PIP} = \sum_k \{\max\{CS(k, S) \mid k \in Use(S)\} \mid \pi(k) < \pi(i) \leq \Pi(S)\}$$

Improvement: Priority Ceiling Protocol (PCP)

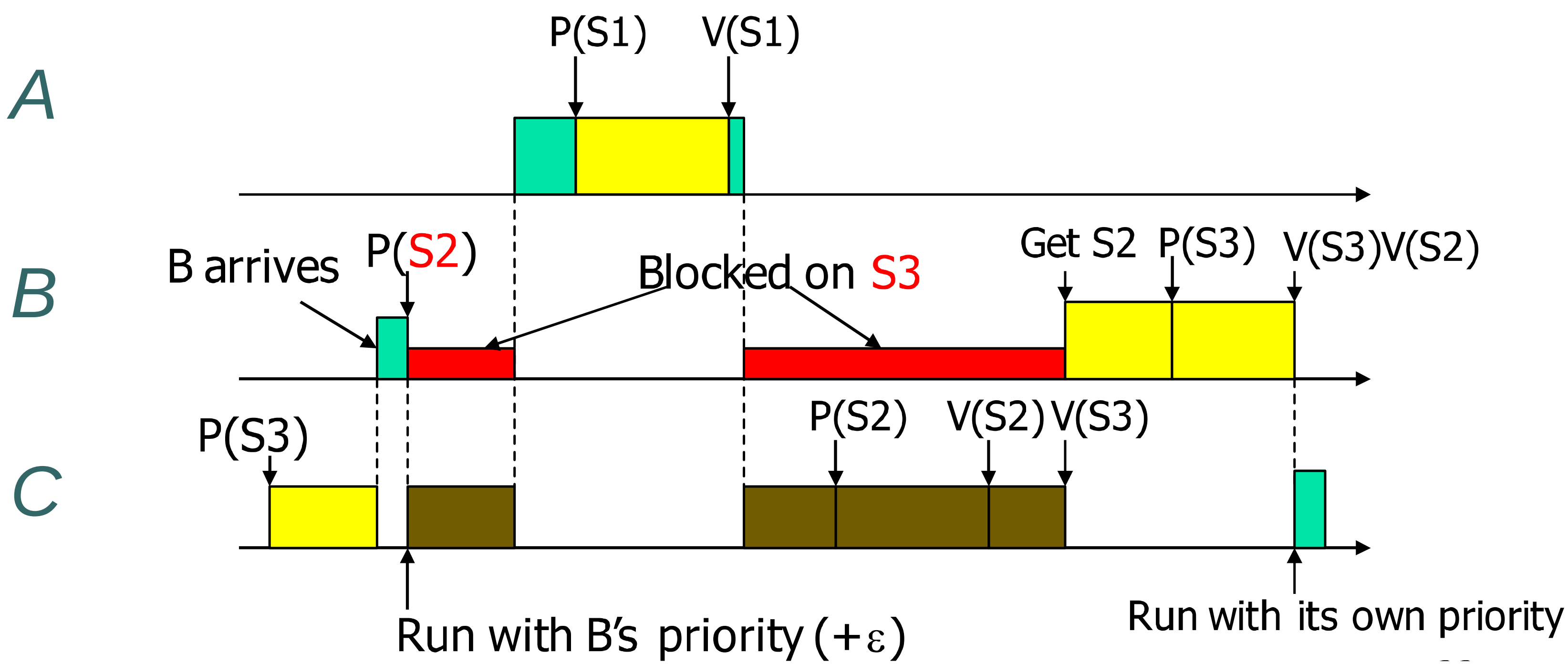
- Two key assumptions:
  - The assigned priorities of all jobs are fixed.
  - The resources required by all jobs are known a priori, i.e., before the execution of any job begins.
- Definition: the priority ceiling of a semaphore R is the highest priority of all the jobs that use R, and is denoted $\Pi(R)$
- Definition: The current priority ceiling $\Pi'(t)$ of the system is equal to the highest priority ceiling of the semaphores in use at time t, or Ω if no resources are in use at time t. (Ω is less than all other priorities.)

Priority Ceiling Protocol: Runtime Behavior

1. Scheduling Rule:
   - At time t when job J is released, the current priority $\pi(t)$ of J is its priority.
   - Every ready job J is scheduled based on its current priority
2. Allocation Rule: When job J requests semaphore S at time t, one of the following conditions occur:
   - S is held by another job and J becomes blocked.
   - S is free:
     - If J's priority $\pi(t)$ is higher than the current priority ceiling $\Pi'(t)$, R is allocated to J.
     - Otherwise, J becomes blocked.
3. Priority-Inheritance Rule: When J becomes blocked, the job J_l that blocks J inherits the current priority $\pi(t)$ of J until it releases every resource whose priority ceiling is $\geq \pi(t)$.

Priority Ceiling Protocol: Example

Task A: … P(S1) … V(S1) …
Task B: … P(S2) … P(S3) … V(S3) … V(S2) …
Task C: … P(S3) … P(S2) … V(S2) … V(S3) …



Beneficial Properties of PCP

Theorem 1: Under PCP, no deadlock can occur.

Why?

Theorem 2: A job can be blocked for at most the duration of one critical section.

Why?

PCP: Blocking Time Calculation

$Use(S)$ *The set of tasks using semaphore S.*

$CS(k, S)$ *The WCET of the critical section of task k using semaphore S.*

The maximal blocking time of task i under PIP:

$$B_i^{PCP} = \max_{k,S}\{CS(k,S) \mid k \in Use(S) \wedge \pi(k) < \pi(i) \leq \Pi(S)\}$$

Priority Inheritance Protocol vs Priority Ceiling Protocol

PIP	PCP
Bounded priority inversion (+)	Bounded priority inversion (+)
May deadlock (-)	Deadlock-free (+)
Up to n blockings (-)	At most one blocking (+)
Easy to implement (+)	Not easy to implement (-)

Schedulability Analysis including Blocking Times

Theorem:

A set of n periodic tasks under PCP can be scheduled by rate-monotonic scheduling, if

$$\forall i, 1 \leq i \leq n, \frac{C_i + B_i}{T_i} + \sum_{j=1}^{i-1} \frac{C_j}{T_j} \leq i(2^{1/i} - 1)$$

where B_i is the worst-case blocking time of task i.

Summary

- Resource sharing may cause priority inversion
- Without further action, priority inversion may be very long
- Priority inheritance and priority ceiling protocols bound the worst-case blocking time
- Can be incorporated into schedulability analysis for rate-monotonic scheduling