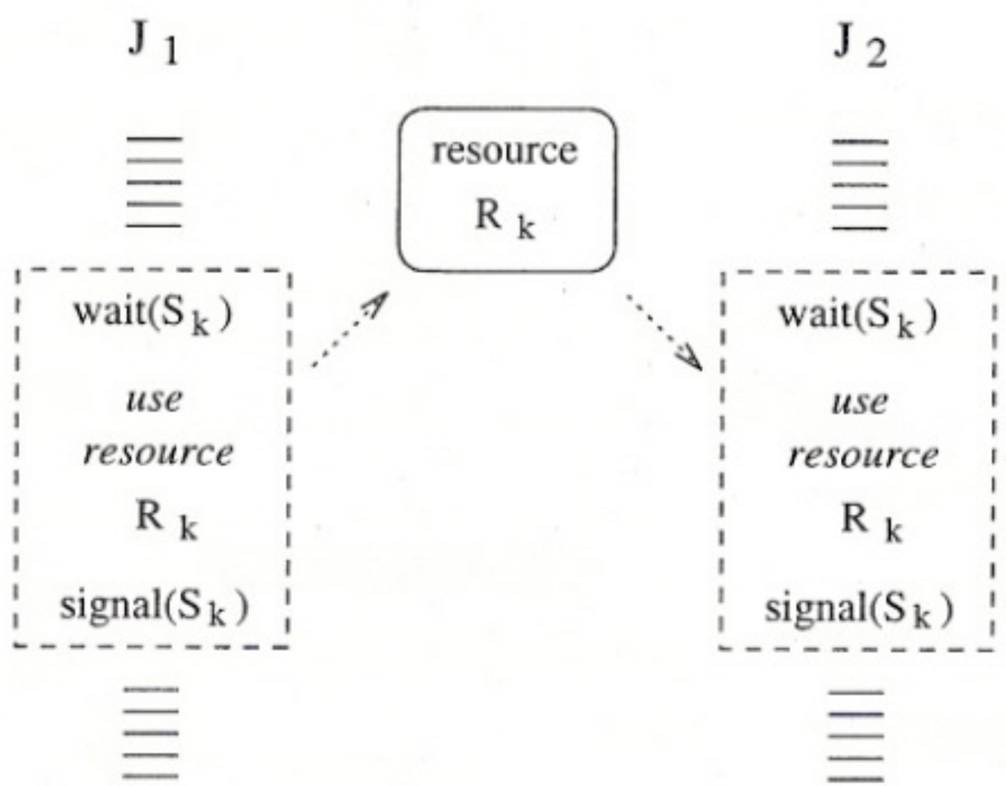
Introduction to
Real-Time Computer Systems

Lecture 9

Sharing Resources

- Many real-time tasks must access resources (data, devices, files, etc.) that are shared among them.
- The resources are locked for integrity before usage.
- Low priority job may lock it first. And then high priority job will have to wait for the resource, causing extra delays.



Mars Rover Priority Inversion

- **Priority Inversion on Mars**

http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/

http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/Authoritative_Account.html

- YouTube

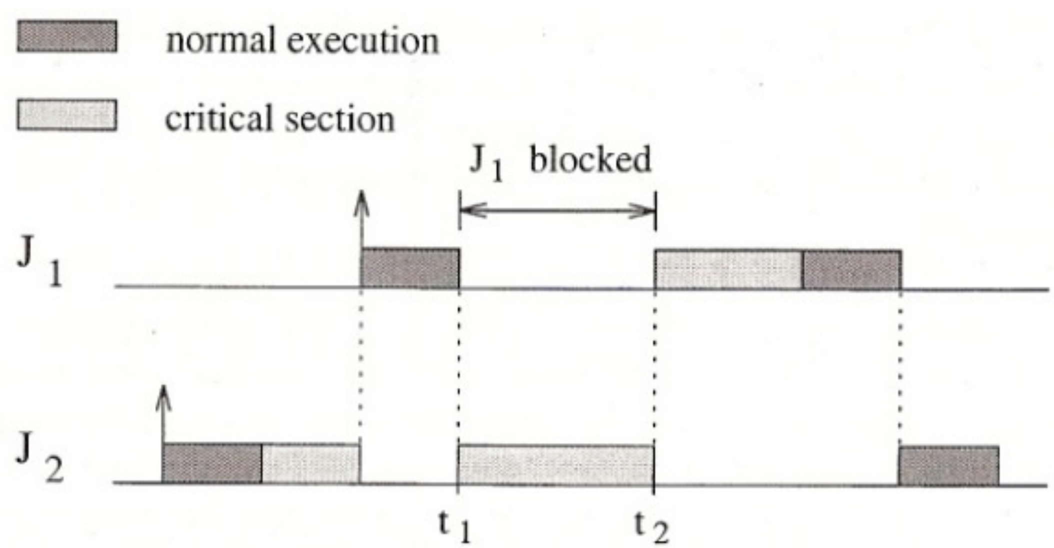
http://www.youtube.com/watch?v=lyx7kARrGeM

- Detailed Technical Description

http://retis.sssup.it/~marko/2009/mars_explorer.pdf

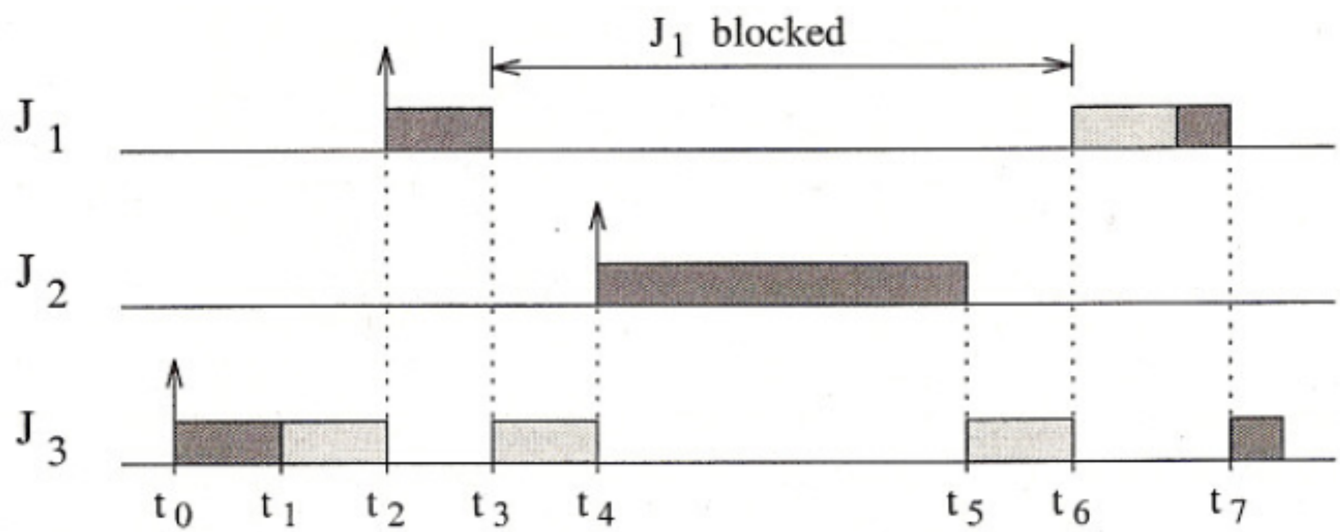
Priority Inversion, 1^{st} case

- Assumption: Whoever locks a file for performing a write, no one else should be able to access the file.
  - Suppose J_1 and J_2 modify the same file.
  - J_1 has a higher priority than J_2.
  - J_1 will be delayed by J_2, *between* t_1 *and* t_2



Priority Inversion, 2nd case

- J_1 must wait for J_3 to release the lock before it can continue.
  - Normally, J_1 is taking out of system contention. If the system then finds the next highest priority job and it's not J_3…

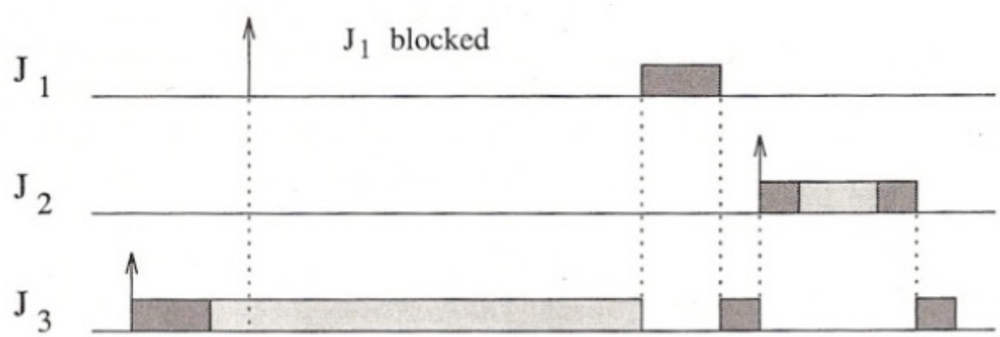


Priority Inversion, Problem Model

- Assumptions
  - We know what resources will be used by each job
  - Each resource has only one copy
  - We don't know exactly when resources are locked
  - There is a maximum length of time that resources are locked
- Problem
  - Given a set of real-time jobs with all above parameters, can they be scheduled to meet deadline using RM or EDF?

Priority Inversion Solution I: NPP

- Non-Preemptive Protocol: When you lock a resource, you become the highest priority in the scheduler.
  - Similar to the OS Kernel mode.



  - Still delays J_l, but not as much (no middle priority job can execute)
  - Simple, but causes unnecessary delay

Highest Locker Protocol (HLP)

- A semaphore is given an "active priority" when it is created; the programmer should choose the highest priority of any thread that may attempt to lock the semaphore .
- As soon as a thread enters synchronized code, its (active) priority is raised to the semaphore's priority.
  - If, through programming error, a thread has a higher base priority than the ceiling of the semaphore it is attempting to enter, then an exception is thrown.

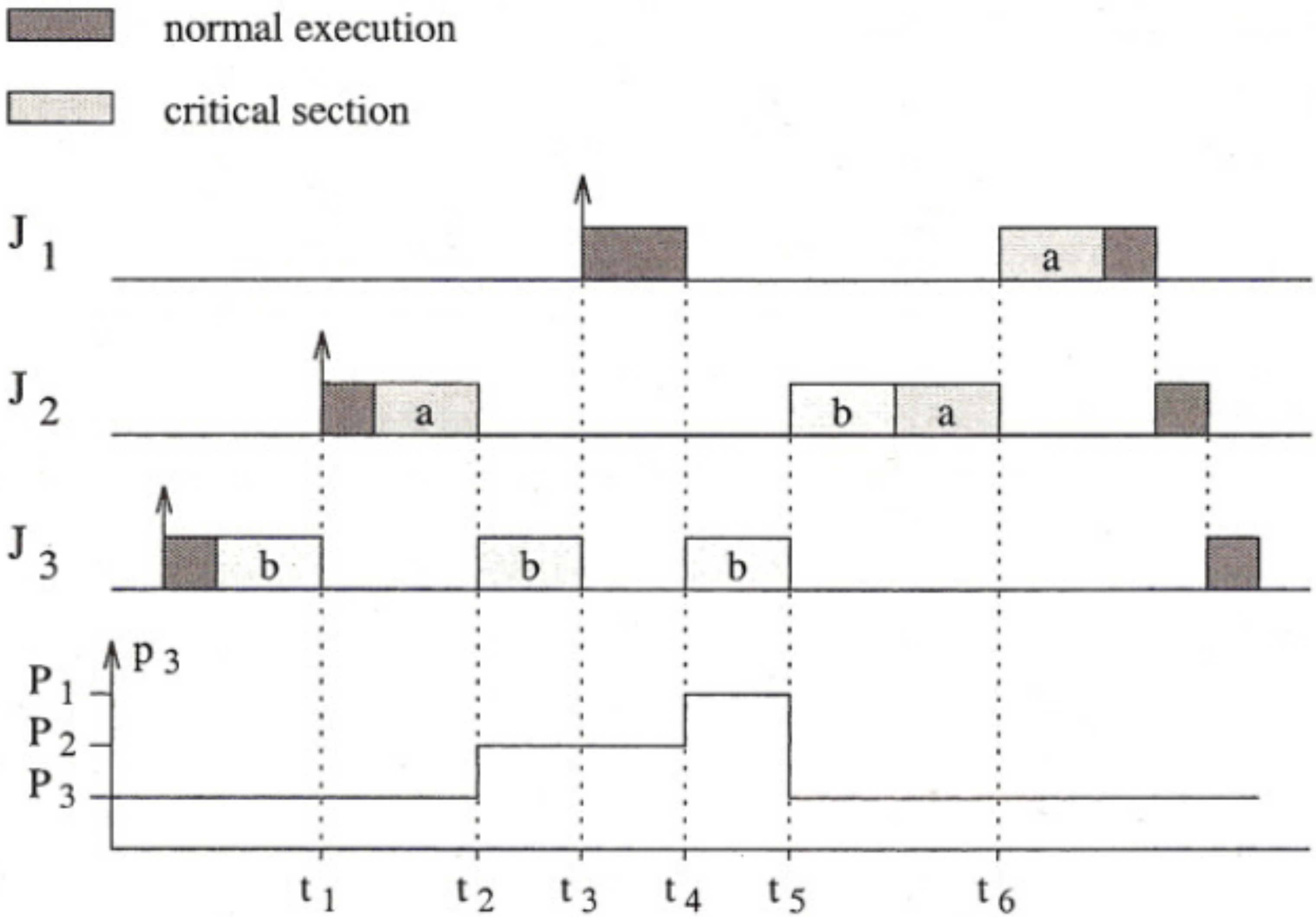
Highest Locker Protocol (HLP)

- On leaving the semaphore, the thread has its active priority reset. In simple cases it will set to the thread's previous active priority, but under some circumstances (e.g. a dynamic change to the thread's base priority while it was in the semaphore) a different value is possible
- Benefit: A high priority job will not be affected by those critical sections shared only by low priority jobs.

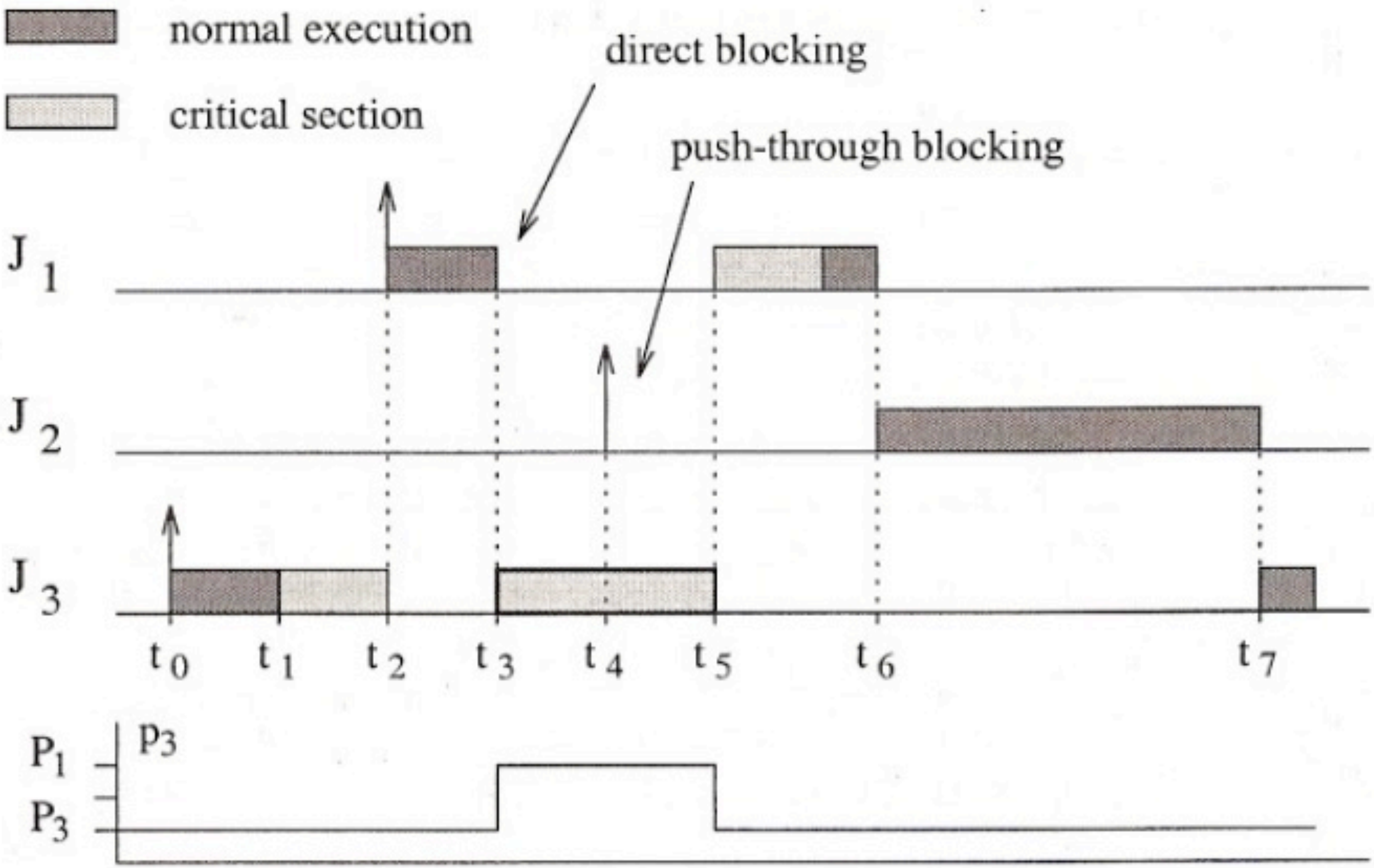
Priority Inheritance Protocol (PIP)

- Whenever a job J_2 locks a resource,
  - Takes note of it but doesn't change priority.
  - When another task J_1 requests the locked resource, J_2 inherits (increases) to J_1's priority if J_2 was lower.
  - J_1 goes to the ready queue.
  - When J_2 has unlocked the resource, it resumes the normal priority.

Example of PIP



Two Kinds of Blocking



Two Kinds of Blocking

- **Direct blocking.** It occurs when a high-priority job tries to acquire a resource already held by a lower-priority job.
  - Direct blocking is necessary to ensure the consistency of the shared resources .
- **Push-through blocking.** It occurs when a medium-priority job is blocked by a lower-priority job that has inherited a higher priority from a job it directly blocks.
  - Push-through blocking is to avoid unbounded priority inversion.

Priority Inheritance Blocking Time

- Whenever a high priority job wants a resource locked by a lower priority job, the lower priority job gains the priority of the higher priority job as long as the higher priority job is waiting.
- The high priority job only needs to wait for one additional blocking time.
- So, we need to know the maximum time any lower priority job will hold a lock on the resource.

Blocking Properties

- **Lemma 7.1** A semaphore S_k can cause push-through blocking to job *Ji*, only if S_k is accessed both by a job with priority lower than *Pi* and by a job that has or can inherit a priority equal to or higher than *Pi*.
- **Lemma 7.3** If there are *n lower-priority jobs* that can block a job *Ji*, then *Ji* can be blocked for at most the duration of n critical sections (one for each of the n lower priority jobs), regardless of the number of semaphores used by *Ji* .
  - A job can block another if it is inside a critical section. Once out, it can't enter another since it no longer has a high priority.
- **Lemma 7.4** If there are *m distinct semaphores* that can block a job *Ji*, then *Ji* can be blocked for at most the duration of m critical sections, one for each of the m semaphores.
  - A nested critical section uses the outer most duration since it is the longest.
- **Theorem 7.1 (Sha-Rajkumar-Lehoczky)** Under the Priority Inheritance Protocol, a job J can be blocked for at most the duration of *min*(n, m) critical sections, where n is the number of lower-priority jobs that could block J and m is the number of distinct semaphores that can be used to block J.