Locking Protocol & Multiprocessor Scheduling

(Most slides are from Jim Anderson Real-Time course)

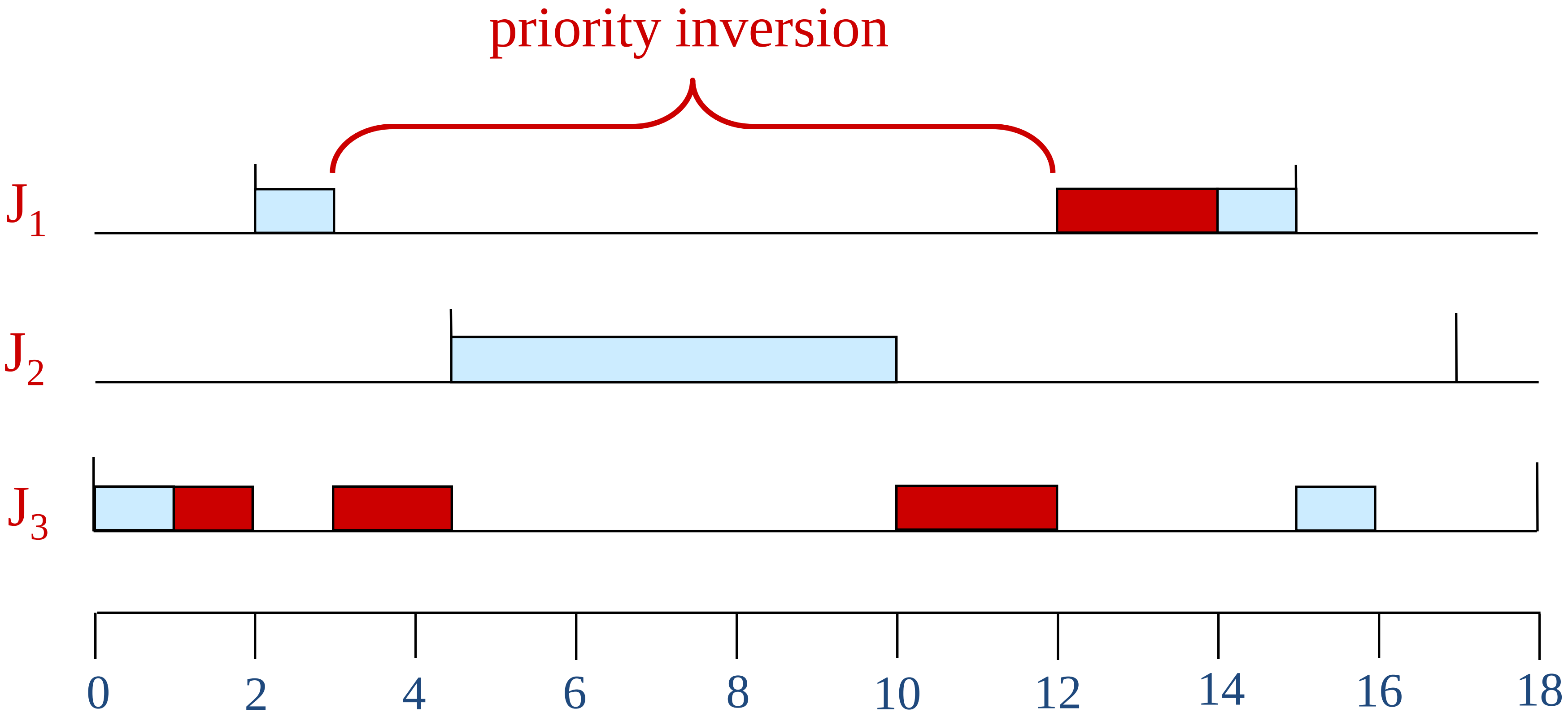
Resources & Locking Protocols

- We continue to consider single-processor systems.
- For simplicity, we will assume there is only one kind of lock request.
- Two jobs have a **resource conflict** if some of the resources they require are the same.
- A matching lock/unlock pair is a **critical section**

Priority Inversions

When tasks share resources, there may be priority inversions.

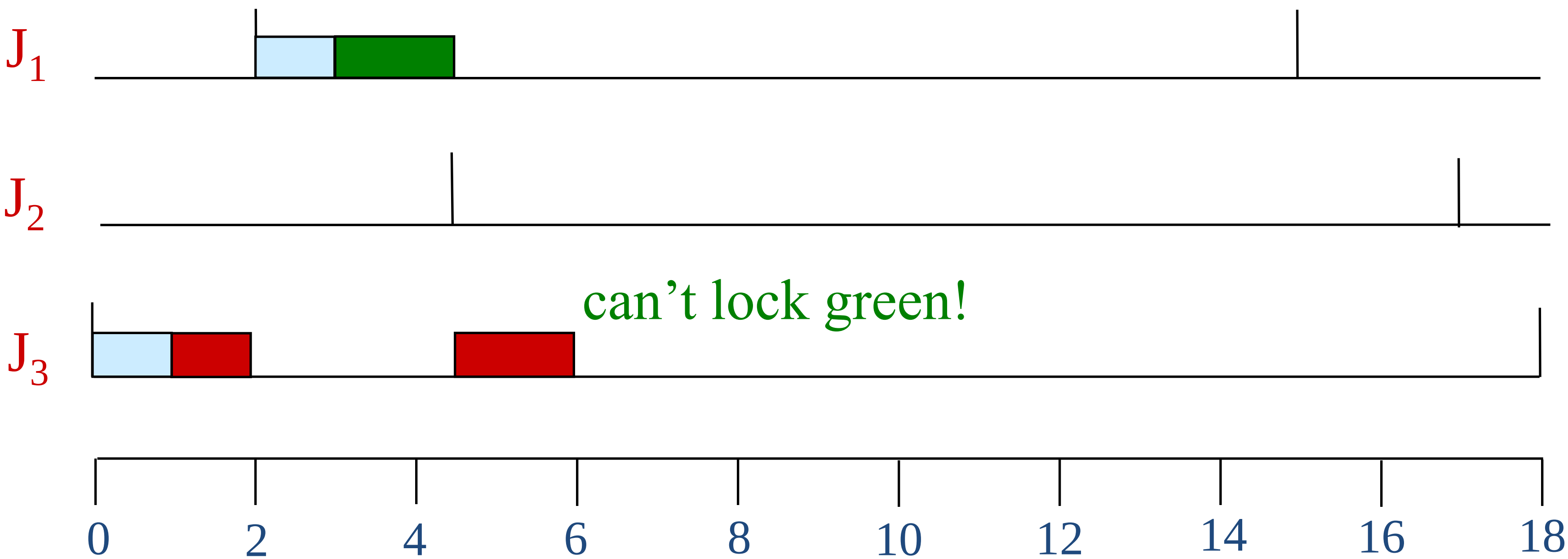
Example:



Deadlocks

When tasks share resources, deadlocks may be a problem.

Example: J_1 accesses green, then red (nested). J_3 accesses red, then green (nested).



Resource Access Control Protocols

- We now consider several protocols for allocating resources that control priority inversions and/or deadlocks.
  1 Nonpreemptive Critical Section Protocol
  2 The Priority Inheritance Protocol
  3 The Priority Ceiling Protocol
  4 Stack Resource Policy

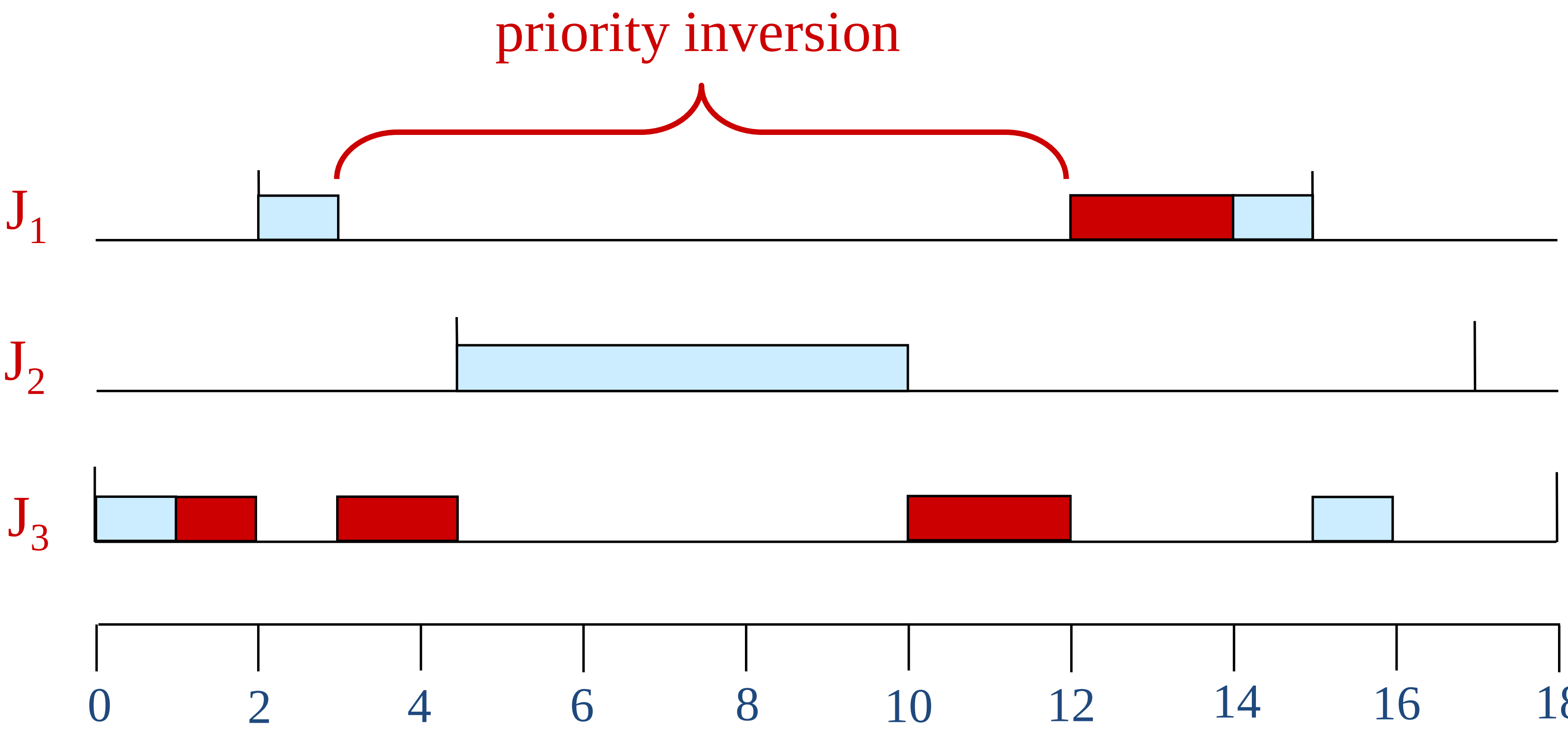
Nonpreemptive Critical Section Protocol

- The simplest protocol: just execute each critical section nonpreemptively

The Priority Inheritance Protocol

When tasks share resources, there may be priority inversions.

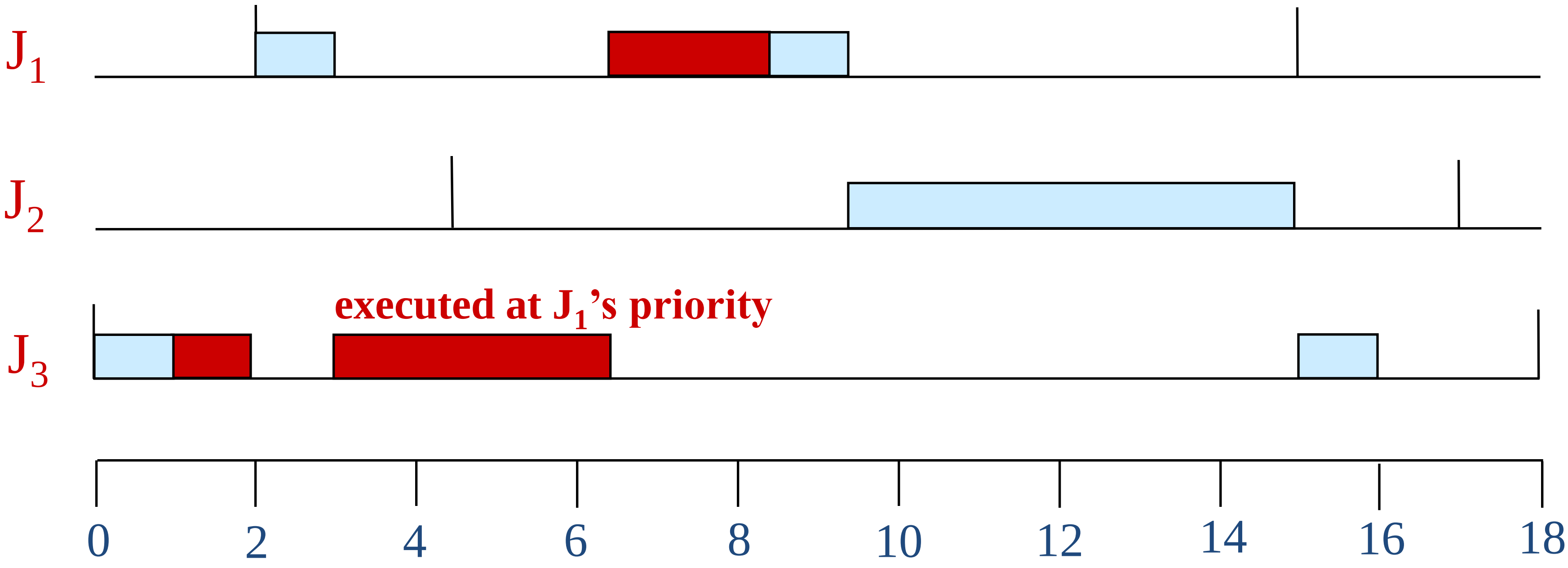
Example:



The Priority Inheritance Protocol

Priority Inheritance Protocol: When a low-priority job blocks a high-priority job, it *inherits* the high-priority job's priority.

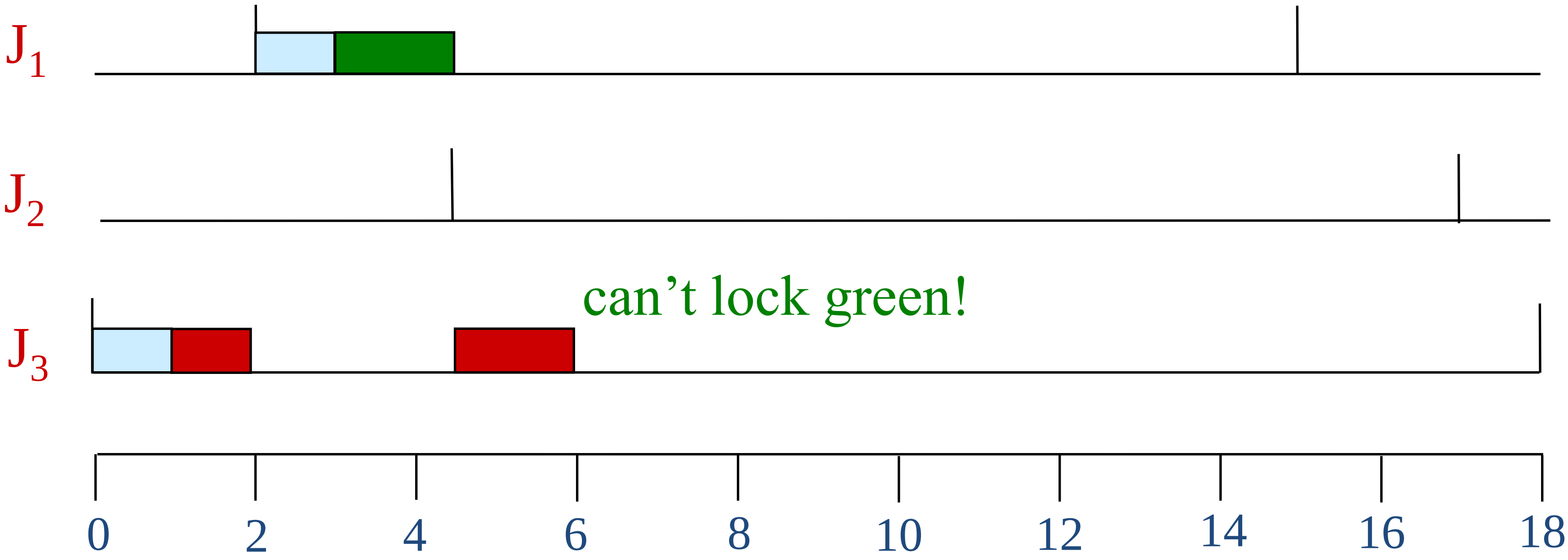
This prevents an untimely preemption by a medium-priority job.



Deadlocks

When tasks share resources, deadlocks may be a problem.

Example: J_1 accesses green, then red (nested). J_3 accesses red, then green (nested).



PIP Definition

Each job J_k has an **assigned priority** (e.g., RM priority) and a **current priority** $\pi_k(t)$.

1. Scheduling Rule: Ready jobs are scheduled on the processor preemptively in a priority-driven manner according to their current priorities. At its release time t, the current priority of every job is equal to its assigned priority. The job remains at this priority except under the condition stated in rule 3.

2. Allocation Rule: When a job J requests a resource R at time t,

(a) if R is free, R is allocated to J until J releases it, and

(b) if R is not free, the request is denied and J is blocked.

3. Priority-inheritance Rule: When the requesting job J becomes blocked, the job J_l that blocks J inherits the current priority of J. The job J_l executes at its inherited priority until it releases R (or until it inherits an even higher priority); the priority of J_l returns to its priority $\pi_l(t')$ at the time t' when it acquired the resource R.

The Priority Ceiling Protocol

- **Two key assumptions:**
  - The assigned priorities of all jobs are fixed (as before).
  - The resources required by all jobs are known *a priori* before the execution of any job begins.
- **Definition:** The **priority ceiling** of any resource R is the highest priority of all the jobs that require R, and is denoted $\Pi(R)$.
- **Definition:** The **current priority ceiling** $\Pi'(R)$ of the system is equal to the highest priority ceiling of the resources currently in use, or Ω if no resources are currently in use (Ω is a priority lower than any real priority).

PCP Definition

1. **Scheduling Rule:**

 (a) At its release time t, the current priority $\pi(t)$ of every job J equals its assigned priority. The job remains at this priority except under the conditions of rule 3.

 (b) Every ready job J is scheduled preemptively and in a priority-driven manner at its current priority $\pi(t)$.

2. **Allocation Rule:** Whenever a job J requests a resource R at time t, one of the following two conditions occurs:

 (a) R is held by another job. J's request fails and J becomes blocked.

 (b) R is free.

  (i) If J's priority $\pi(t)$ is higher than the current priority ceiling $\Pi'(t)$, R is allocated to J.

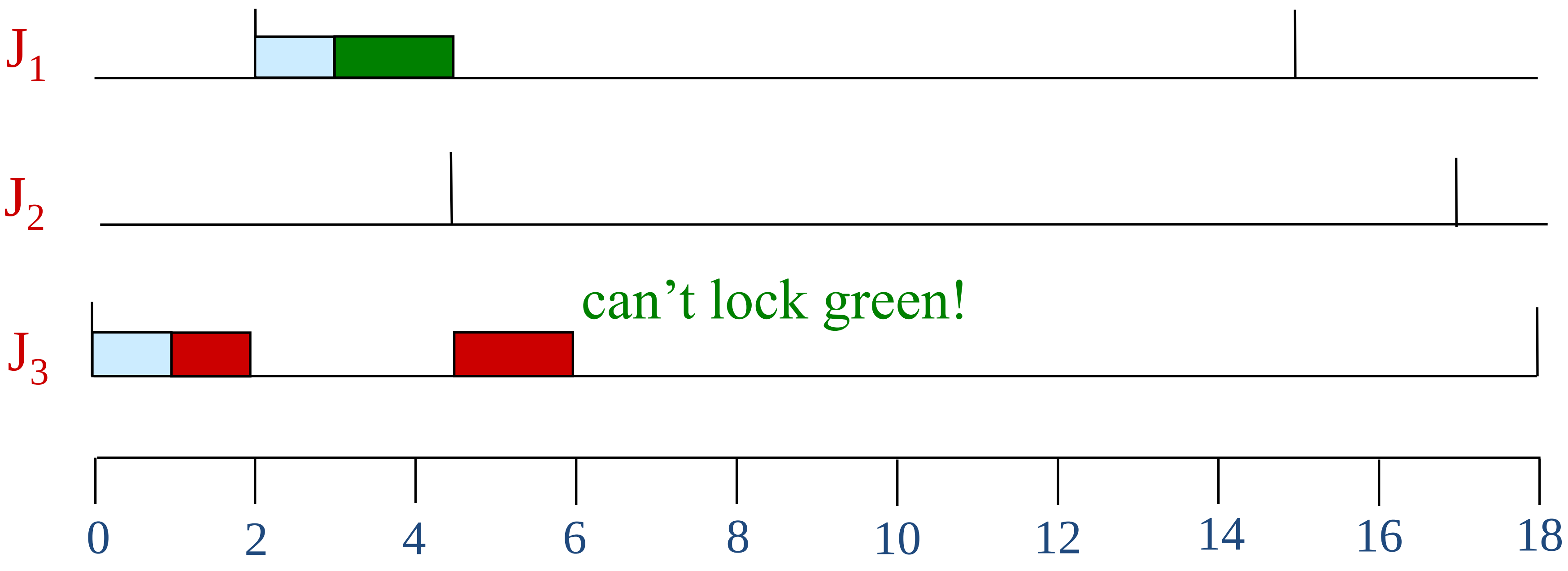
  (ii) If J's priority $\pi(t)$ is not higher than the ceiling $\Pi'(t)$, R is allocated to J only if J is the job holding the resource(s) whose priority ceiling equals $\Pi'(t)$; otherwise, J's request is denied and J becomes blocked.

3. **Priority-inheritance Rule:** When J becomes blocked, the job J_l that blocks J inherits the current priority $\pi(t)$ of J. J_l executes at its inherited priority until it releases every resource whose priority ceiling is $\geq \pi(t)$ (or until it inherits an even higher priority); at that time, the priority of J_l returns to its priority $\pi(t')$ at the time t′ when it was granted the resources.

Deadlocks

When tasks share resources, deadlocks may be a problem.

Example: J_1 accesses green, then red (nested). J_3 accesses red, then green (nested).

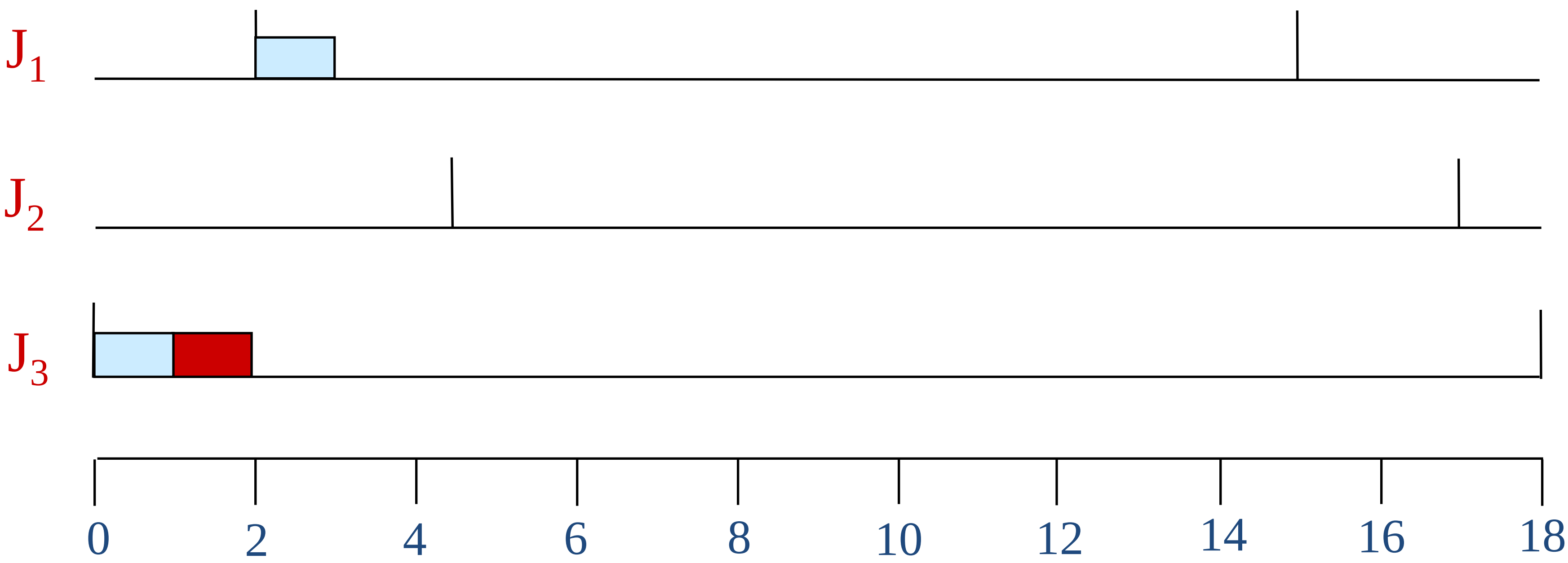


Deadlock Avoidance

With the PIP, deadlock could occur if nested critical sections are invoked in an inconsistent order. Here's an example we looked at earlier.

Example: J_1 accesses green, then red (nested). J_3 accesses red, then green (nested).

want lock green, but cannot



The PCP would prevent J_1 from locking green.

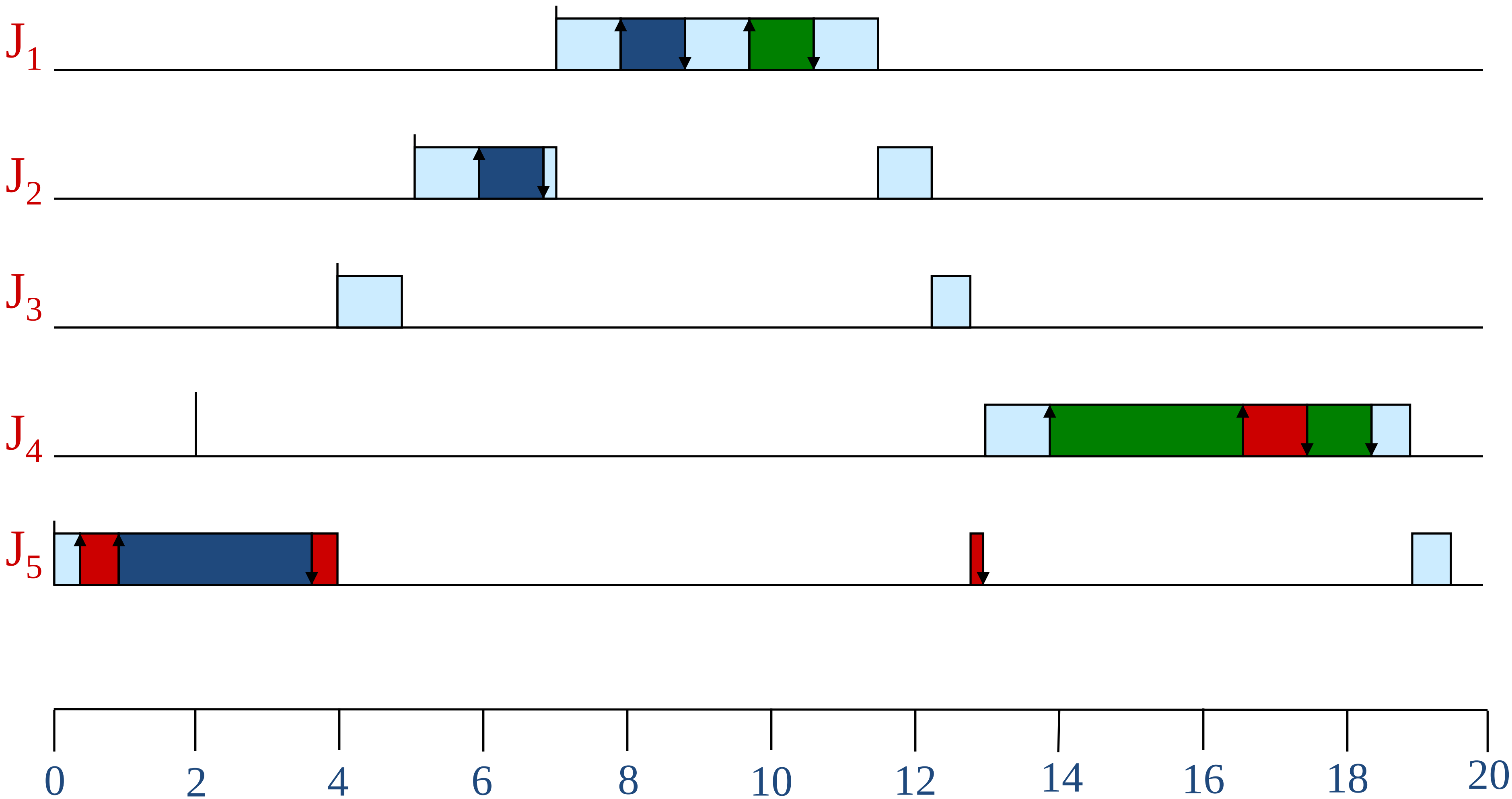
Stack Resource Policy

0. **Update of the Current Ceiling:** Whenever all the resources are free, the ceiling of the system is Ω. The ceiling $\Pi'(t)$ is updated each time a resource is allocated or freed.

1. **Scheduling Rule:** After a job is released, it is blocked from starting executing until its assigned priority is higher than the current ceiling $\Pi'(t)$ of the system. At all times, jobs that are not blocked are scheduled on the processor in priority-driven, preemptive manner according to their assigned priorities.

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

Example



Properties of the SRP

- No job is ever blocked once its execution begins.
  - Thus, there can never be any deadlock.

With the SRP, a job is blocked only before it begins execution, so extra context switches due to blockings are avoided.

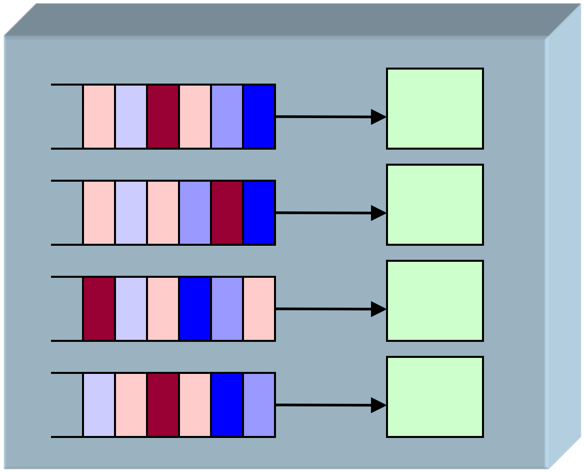
Multiprocessor Scheduling

(Partitioning)

Partition **tasks** so that each task always runs on the same processor.

Steps:

1. Assign tasks to processors.
2. Schedule tasks on each processor using a uniprocessor algorithm.



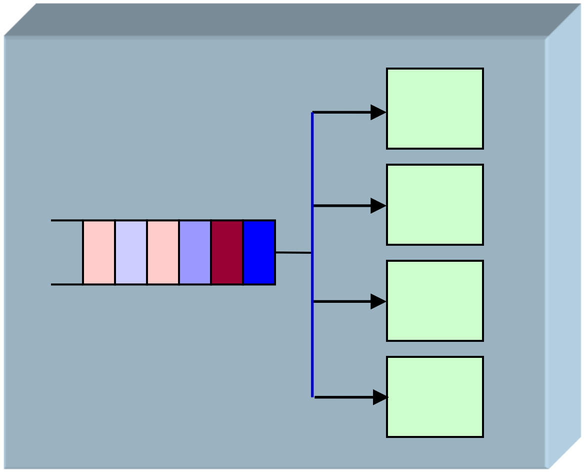
Global Scheduling

(An Alternative to Partitioning)

A single scheduling algorithm is used that schedules all tasks.

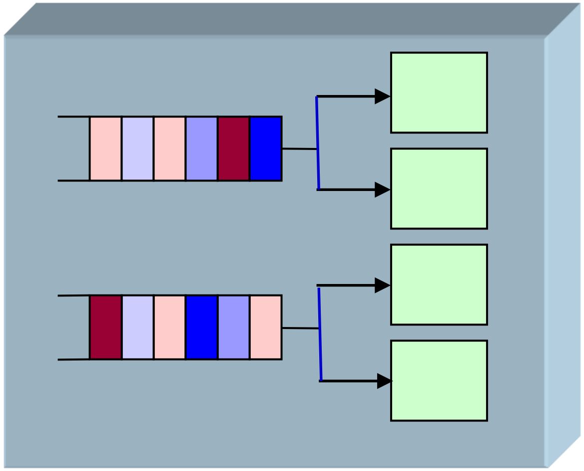
Important Differences:

- A single task queue.
- Tasks may *migrate* among the processors.



Clustered Scheduling

Partition onto clusters of cores, globally schedule within each cluster.



Important Differences:

- Bin packing issues, but to a lesser extent.
- Tasks may migrate among the processors within cluster pool.

Some Example Algorithms

- Uniprocessor scheduling algorithm can still be used with all 3 multiprocessor scheduling approaches.
  - Partitioned-EDF, Global-EDF, Clustered-EDF…

HRT: Optimality is lost

SRT: Tardiness is bounded if:

- Total Utilization $\leq$ m (where m is the number of processors)
- $u_i \leq 1$

Multiprocessor Real-Time Locking

- Spin-Based Locking is used by the flexible multiprocessor locking protocol (FMLP) [Block, et al., 2007]

- Suspension-Based Locking is used by OMLP [Brandenburg, et al., 2010]

Other Multiprocessor Locking Protocols

- For Partitioned Static-Priority Schedulers
  - DPCP [Rajkumar et al. 88, 91]:
  - MPCP [Rajkumar 90, 91]:
- For PEDF
  - Two PCP variants [Chen and Tripathi 94]
  - MSRP [Gai et al. 03]:
- For Global Static-Priority Schedulers
  - PIP [Easwaran and Andersson, 09]
  - P-PCP [Easwaran and Andersson, 09]

References

A. Block, H. Leontyev, B. Brandenburg, and J. Anderson, " A Flexible Real-Time Locking Protocol for Multiprocessors ", Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 47-57, August 2007.

B. Brandenburg and J. Anderson, " Optimality Results for Multiprocessor Real-Time Locking", Proceedings of the 31st IEEE Real-Time Systems Symposium, pp. 49-60, December 2010.

R. Rajkumar, L. Sha, and J. Lehoczky. Real-time synchronization protocols for multiprocessors. Proc. of the 9th Real-Time Systems Symposium, pages 259–269, 1988.

R. Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. Proc. of the 10th International Conference on Distributed Computing Systems, pages 116–123, 1990.

References

R. Rajkumar. Synchronization In Real-Time Systems – A Priority Inheritance Approach. Kluwer Academic Publishers, 1991.

C. Chen and S. Tripathi. Multiprocessor priority ceiling based protocols. Technical Report CS-TR-3252, Univ. of Maryland, 1994.

P. Gai, M. di Natale, G. Lipari, A. Ferrari, C.Gabellini, and P. Marceca. A comparison of MPCP and MSRP when sharing resources in the Janus multiple processor on a chip platform. In Proc. of the 9th IEEE Real-Time And Embedded Technology Application Symposium, pages 189–198, 2003.

A. Easwaran and B. Andersson. Resource sharing in global fixedpriority preemptive multiprocessor scheduling. In Proc. of the 30th IEEE Real-Time Systems Symposium, pages 377–386, 2009.