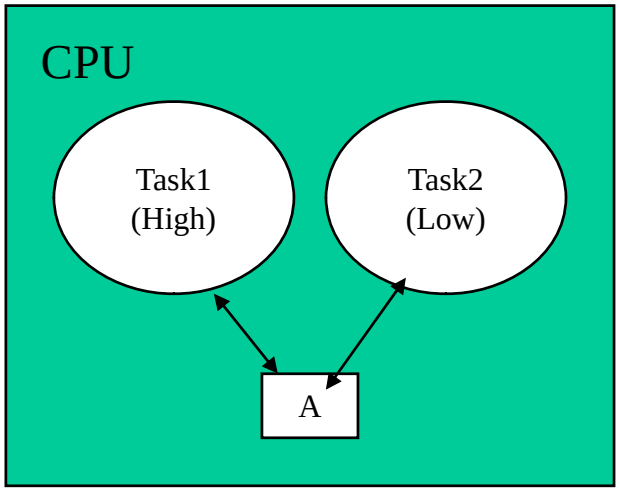
Resources and Resource Access Control (only on Fixed-Priority System)

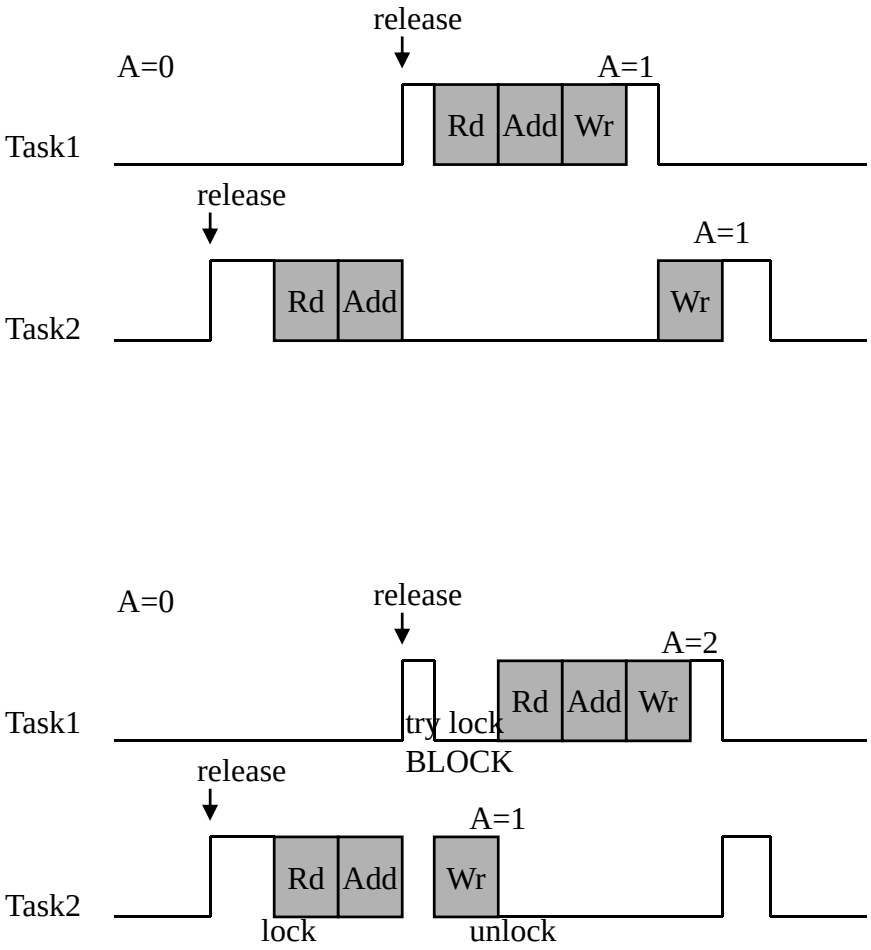
- Chapter 8 -

Overview

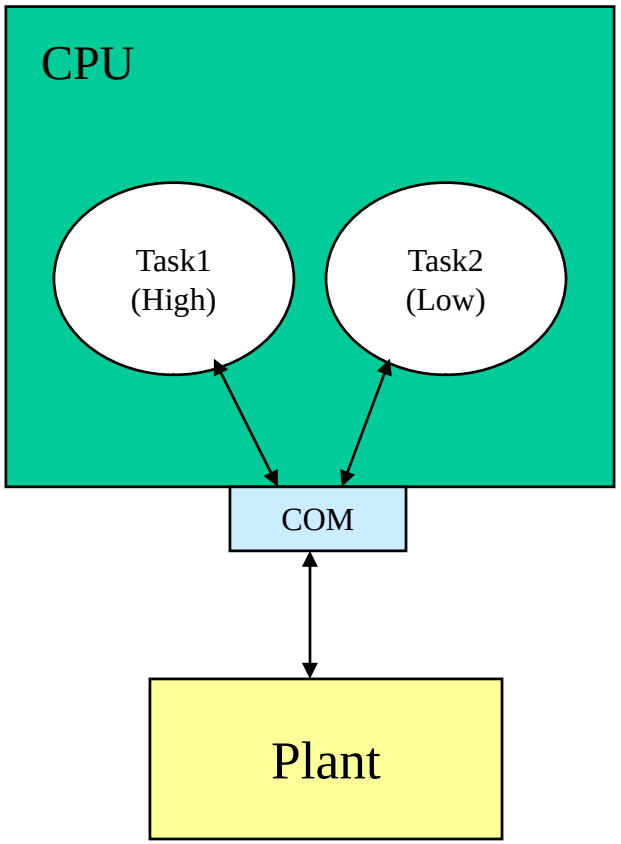
- Resource access (Mutual Exclusive)
  - Priority inversion
  - Unbounded priority inversion
- Resource access control protocol
  - Priority inheritance protocol
  - Priority ceiling protocol

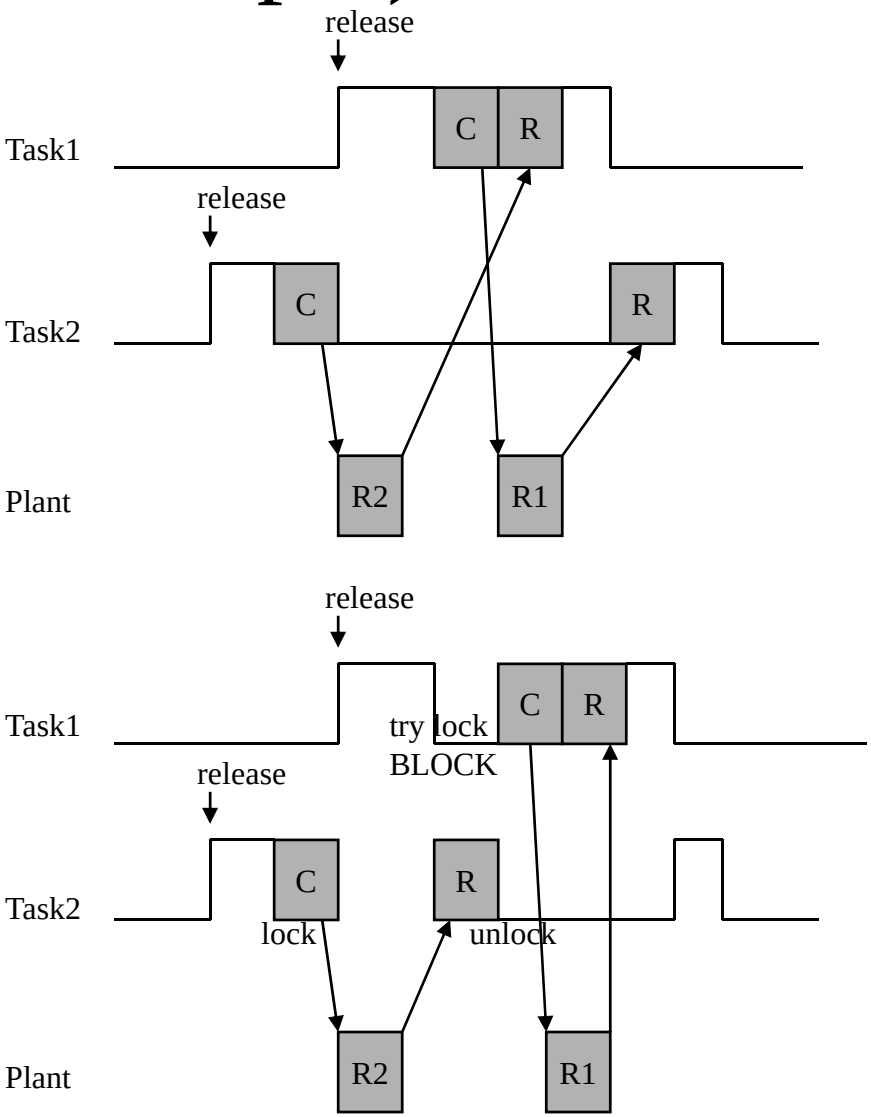
Mutually exclusive resource sharing





Mutually exclusive resource sharing (Another example)

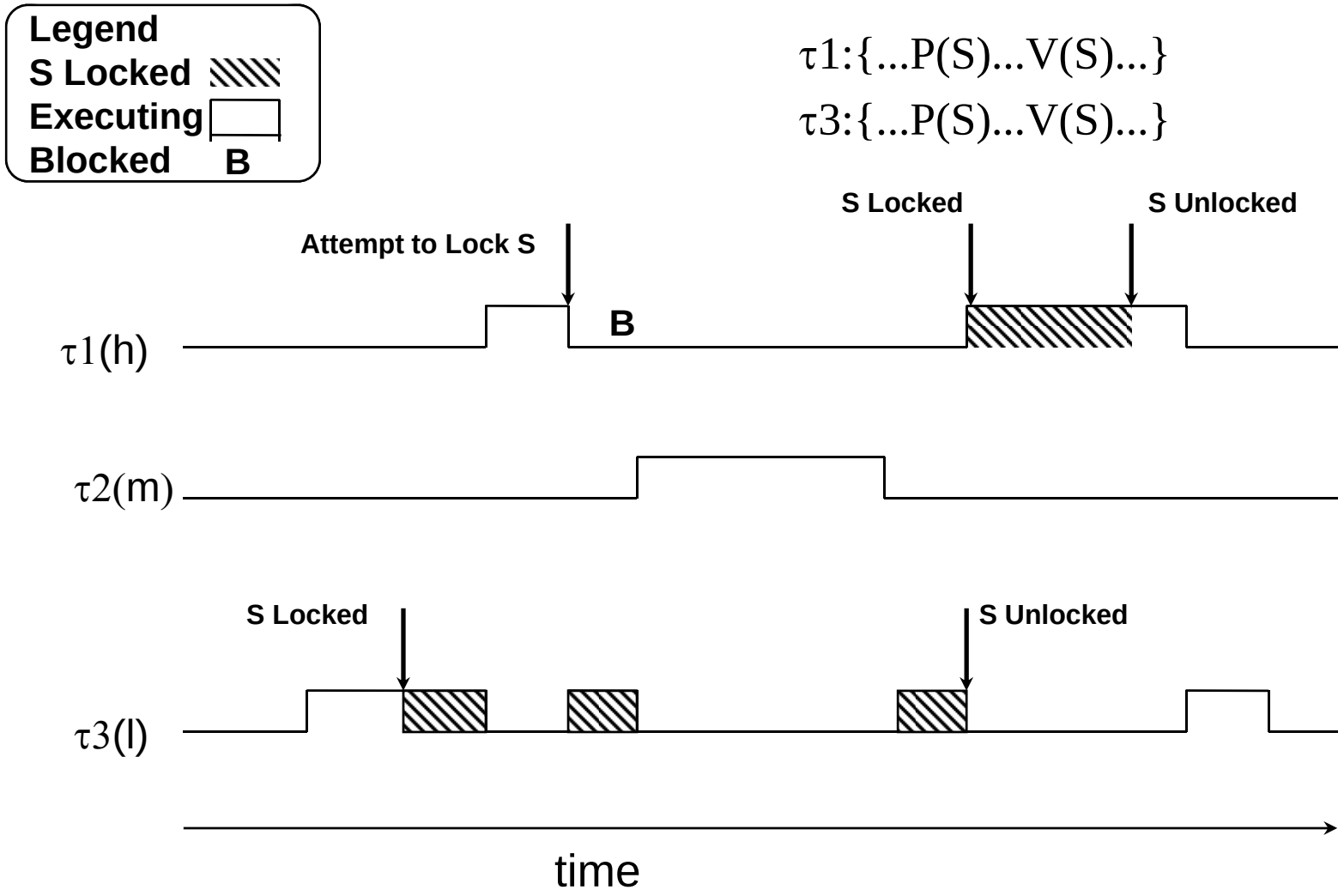




Unbounded Priority Inversion

- When a high priority task is delayed by lower priority tasks, it is said that priority inversion has occurred and the high priority task is blocked by the lower priority task.
- Priority inversion occurs during synchronization.
- When tasks synchronize, we expect delays due to the use of mutual exclusion.
- And we expect that the delay due to mutual exclusion is a function of the duration of the critical sections.
- When the duration of priority inversion is not bounded by a function of the duration of critical sections, unbounded priority inversion is said to occur.

Unbounded Priority Inversion



Mars Pathfinder (Rover)

What really happened on Mars?

- **The Mars Pathfinder mission was widely proclaimed as "flawless" in the early days after its July 4th, 1997 landing on the Martian surface**. Successes included its unconventional "landing" -- bouncing onto the Martian surface surrounded by airbags, deploying the Sojourner rover, and gathering and transmitting voluminous data back to Earth, including the panoramic pictures that were such a hit on the Web. But 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. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".

- **VxWorks provides preemptive priority scheduling of threads**. Tasks on the Pathfinder spacecraft were executed as threads with priorities that were assigned in the usual manner reflecting the relative urgency of these tasks.

- 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 (mutexes).**

- You can read more at http://sssup1.sssup.it/~giorgio/mars/jones.html

What really happened on Mars?

- **The meteorological data gathering task ran as an infrequent, low priority thread, and used the information bus to publish its data. When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex**. If an interrupt caused the information bus thread to be scheduled while this mutex was held, and if the information bus thread then attempted to acquire this same mutex in order to retrieve published data, this would cause it to block on the mutex, waiting until the meteorological thread released the mutex before it could continue. **The spacecraft also contained a communications task that ran with medium priority.**

- 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.

- This scenario is a classic case of “unbounded” priority inversion.

What really happened on Mars?

- **How was this debugged?** VxWorks can be run in a mode where it records a total trace of all interesting system events, including context switches, uses of synchronization objects, and interrupts. After the failure, JPL engineers spent hours and hours running the system on the exact spacecraft replica in their lab with tracing turned on, attempting to replicate the precise conditions under which they believed that the reset occurred. The engineers finally reproduced a system reset on the replica. Analysis of the trace revealed the priority inversion.

- **How was the problem corrected?** When created, a VxWorks mutex object accepts a boolean parameter that indicates whether priority inheritance should be performed by the mutex. The mutex in question had been initialized with the parameter off; had it been on, the low-priority meteorological thread would have inherited the priority of the high-priority data bus thread blocked on it while it held the mutex, causing it be scheduled with higher priority than the medium-priority communications task, thus preventing the priority inversion. **Leaving the "debugging" facilities in the system saved the day. Without the ability to modify the system in the field, the problem could not have been corrected.**

- **Finally, the engineer's initial analysis that "the data bus task executes very frequently and is time-critical -- we shouldn't spend the extra time in it to perform priority inheritance" was exactly wrong. It is precisely in such time critical and important situations where correctness is essential, even at some additional performance cost.**

The importance of good theory/algorithms

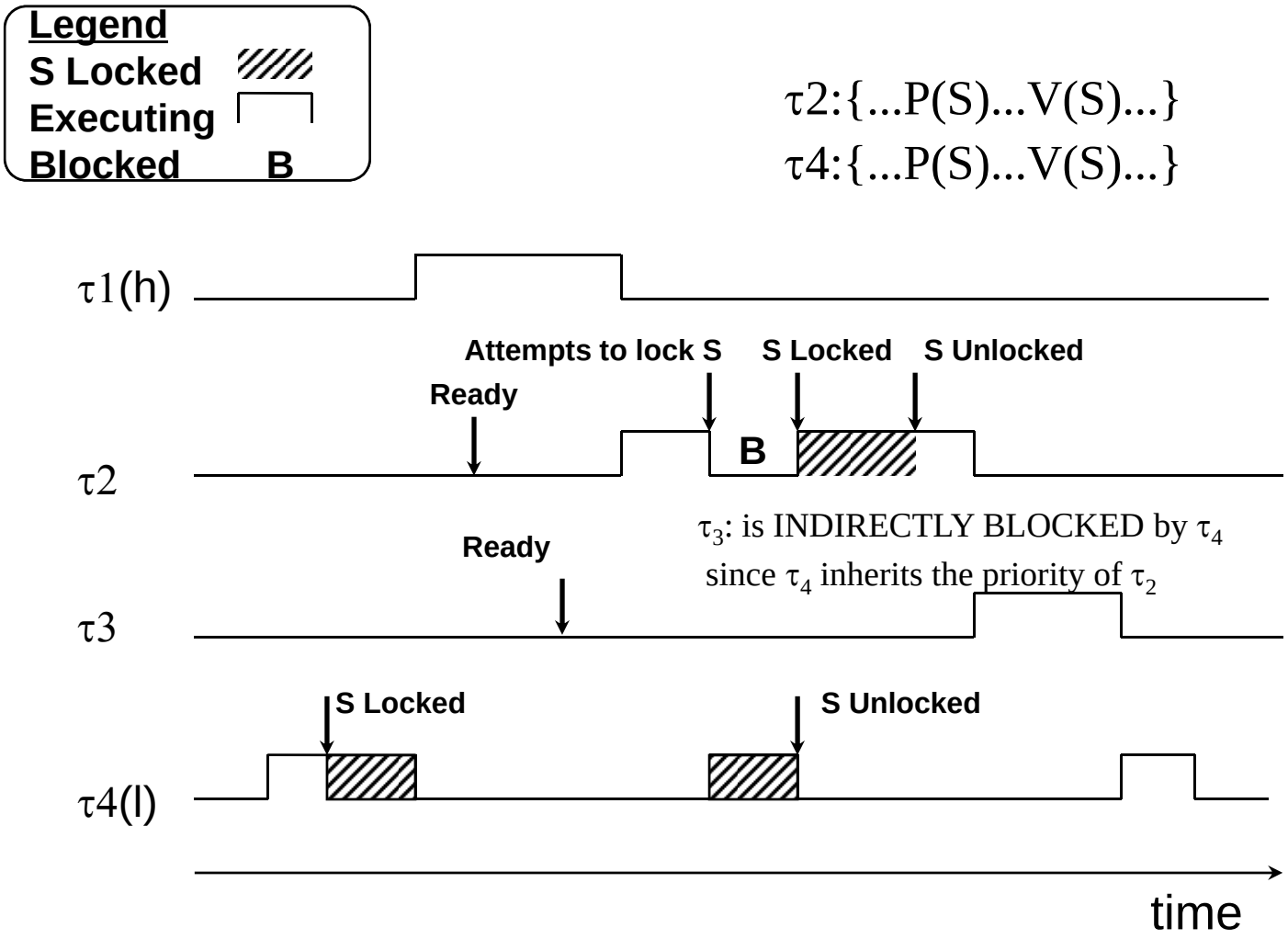
- **The paper that first identified the priority inversion problem and proposed the solution was:**

- L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. In IEEE Transactions on Computers, vol. 39, pp. 1175-1185, Sep. 1990.

Basic Priority Inheritance Protocol

- Rule 1: When a lower priority task blocks higher priority tasks during its critical section, it uses (inherits) the highest priority of all the blocked tasks.
- Rule 2: When a task exits its critical section, it returns to its normally assigned priority
- Rule 3: Priority inheritance is transitive.

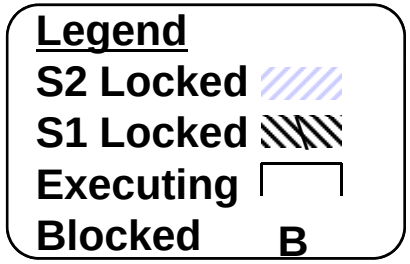
Basic Priority Inheritance Protocol



Properties of Basic Priority Inheritance Protocol

- We shall assume that 1) a job does not self-suspending inside a critical section; and 2) if nested semaphore are used, it will be properly nested. Under PIP,
  - A job can be blocked directly when it shares semaphores with lower priority tasks. It can be blocked by each of the lower priority task once that shares one or more semaphores. The duration is the longest outmost critical section.
  - A job can be blocked indirectly if a higher priority and a lower priority share a semaphore. It occurs when the lower priority task inherits the higher priority task's priority.
  - The total blocking time will be the sum of direct and indirect blocking.
- Forgetting the computation of indirect blocking is a common mistake. Don't make this mistake in your schedulability analyzer.

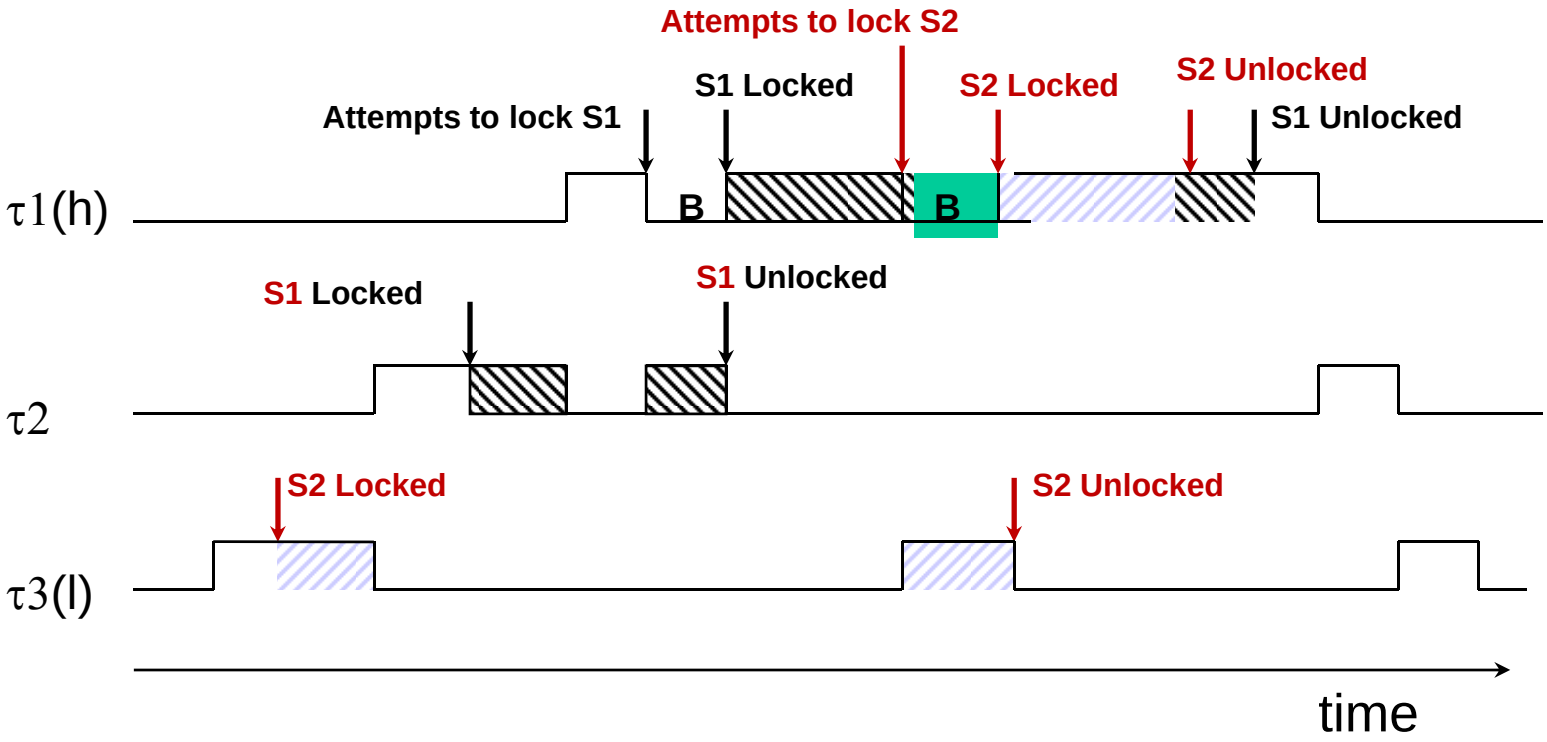
Chained Blocking under PIP



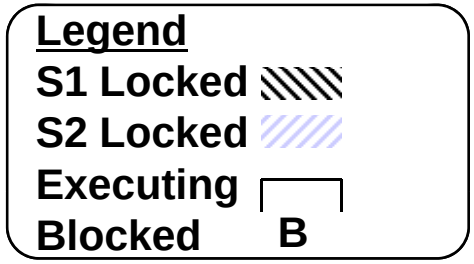
τ1:{...P(S1)...P(S2)...V(S2)...V(S1)...}

τ2:{...P(S1)...V(S1)...}

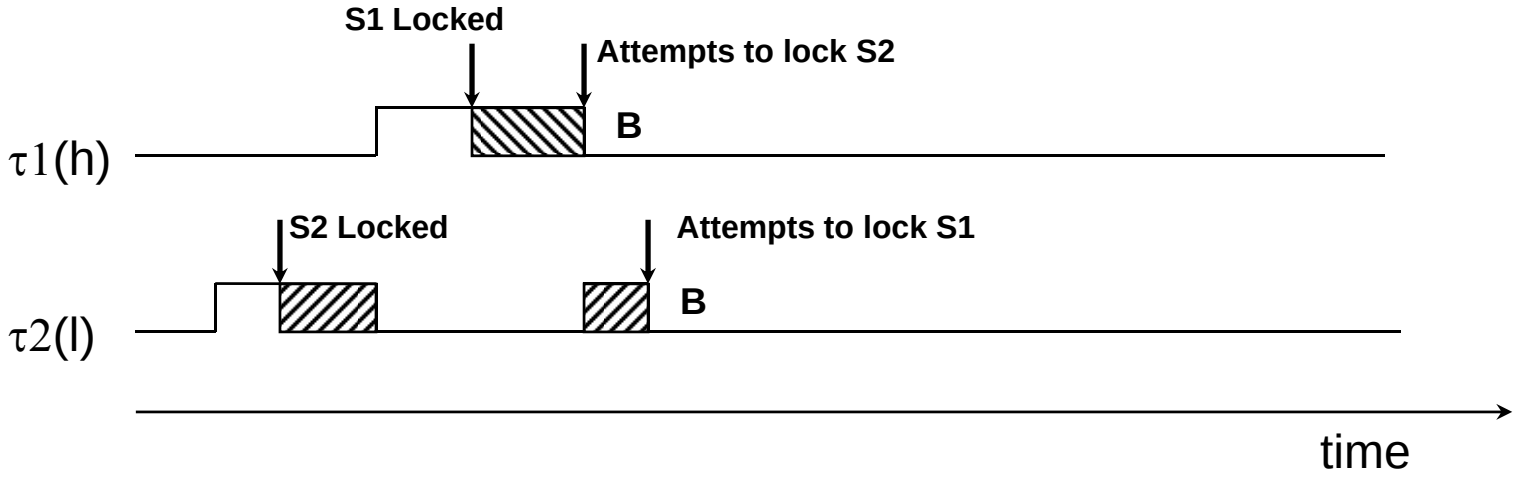
τ3:{...P(S2)...V(S2)...}



Deadlock Under PIP



τ1:{...P(S1)...P(S2)...V(S2)...V(S1)...}
τ2:{...P(S2)...P(S1)...V(S1)...V(S2)...}



Blocking time under PIP

- can be blocked only once by each lower priority task
  - Directly
  - Indirectly

$$b_i(rc) = \sum_{j=i+1}^{n} cs_j rc_{i,j}$$

cs_j is the duration of outermost critical section of task - j

$$rc_{i,j} = \begin{cases} 0 & \text{no block by task - } j \\ 1 & \text{block by task - } j \end{cases}$$

Example

T1 = {.. P(A) .3. P(B) .2. V(B) .1. V(A) ..}

T2 = {.. P(C) .2. V(C) ..}

T3 = {.. P(A) .1. P(B).2.V(B) .2. V(A) .. }

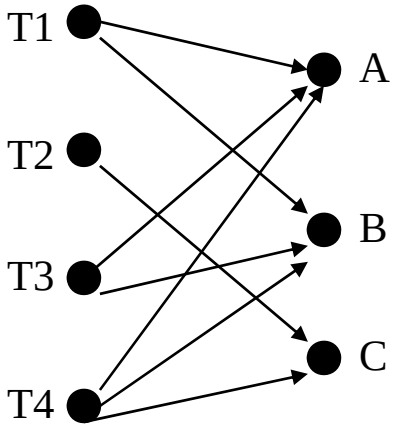
T4 = {..P(A).1.P(C).1.P(B) .3. V(B).1.V(C).1.V(A).. }

CS1=6

CS2=2

CS3=5

CS4=7



	Directly blocked by			Indirectly blocked by			Totally blocked by		
	T2	T3	T4	T2	T3	T4	T2	T3	T4
T1	N	Y	Y					5	7
T2	*	N	Y	*	Y	Y	*	5	7
T3		*	Y		*	Y		*	7

b1=12

b2=12

b3=7

Priority Ceiling Protocol

- Definition 1: A priority ceiling is assigned to each semaphore, which is equal to the highest priority job that may use this semaphore.
- Definition 2: A priority ceiling is active if the semaphore associated with the ceiling is locked.
- Definition 3: For any job J, the current system priority ceiling is the max of (all the active ceiling – ceiling of semaphores hold by J).

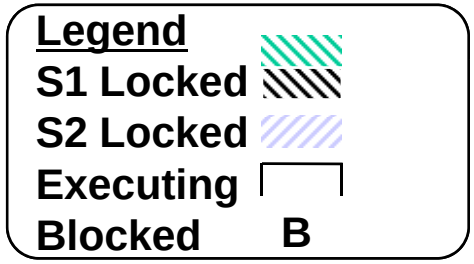
- Rule 1: Each job is scheduled by its assigned priorities.

- Rule 2: (Ceiling rule) A job cannot lock a semaphore unless its priority is higher (not equal) than the current system priority ceiling – *prevent potential chained blocking!* (Locking is allowed for the owner job of the current system ceiling.)

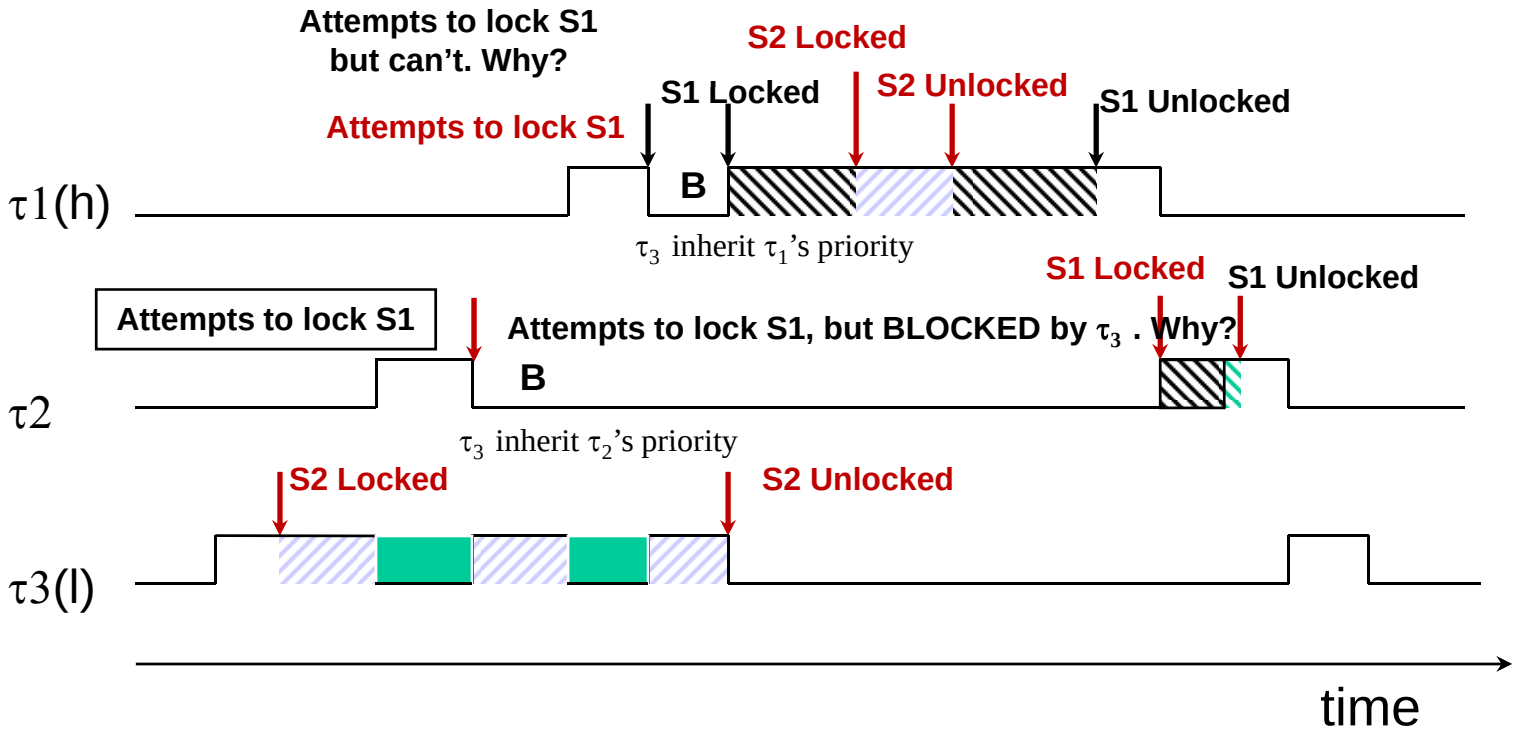
- Rule 3: (Priority inheritance rule) If a job J blocks higher priority jobs via priority ceiling, J inherits the priority of the blocked high priority tasks.

- Rule 4: When a job exits its critical section, it returns to its normal priority.

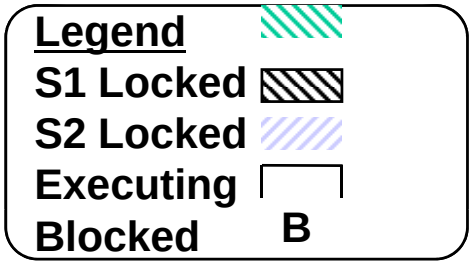
Blocked at Most Once (PCP)



τ1:{...P(S1)...P(S2)...V(S2)...V(S1)...}
τ2:{...P(S1)...V(S1)...}
τ3:{...P(S2)...V(S2)...}

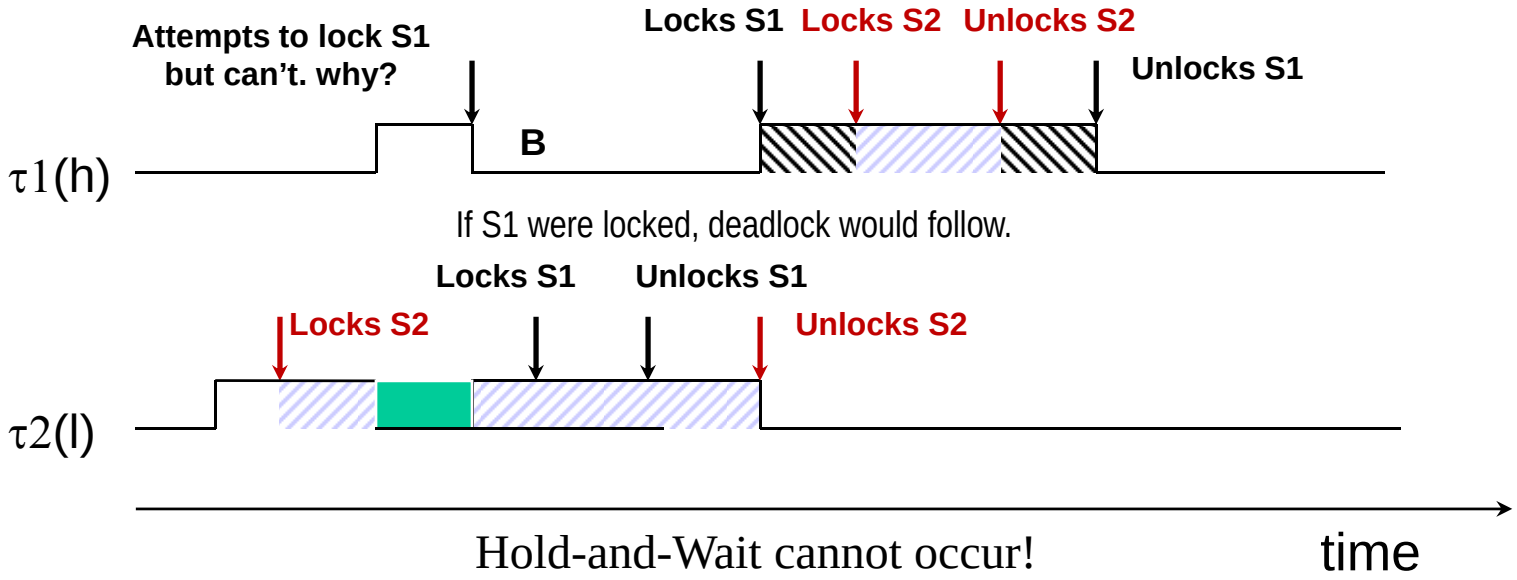


Deadlock Avoidance: Using PCP



τ1:{...P(S1)...P(S2)...V(S2)...V(S1)...}

τ2:{...P(S2)...P(S1)...V(S1)...V(S2)...}



Note: Task τ2 can still lock S1 since it owns the current system ceiling, S1 is not locked by OTHER tasks

Properties of PCP

- Under priority ceiling protocol,
  - A job can be blocked by lower priority jobs at most once no matter how many semaphores they share. In addition, jobs cannot be deadlocked.
  - There is indirect blocking under PCP. But a job can be blocked at most once directly or indirectly.
  - The blocking time of a job is the maximum of all the outermost critical sections that could block the job directly or indirectly
- Failure to account for indirect blocking is a common mistake and don’t make it in your exam or in your schedulability analyzer.

Blocking time under PCP

- can be blocked only once
  - Direct block
  - Priority inheritance block
  - Priority ceiling block

$$b_i(rc) = \max_{j=i+1}^{n} cs_j rc_{i,j}$$

cs_j is the duration of outermost critical section of task - j

$$rc_{i,j} = \begin{cases} 0 & \text{no block by task - } j \\ 1 & \text{block by task - } j \end{cases}$$

Example

T1 = {.. P(X) .10. V(X) ..}

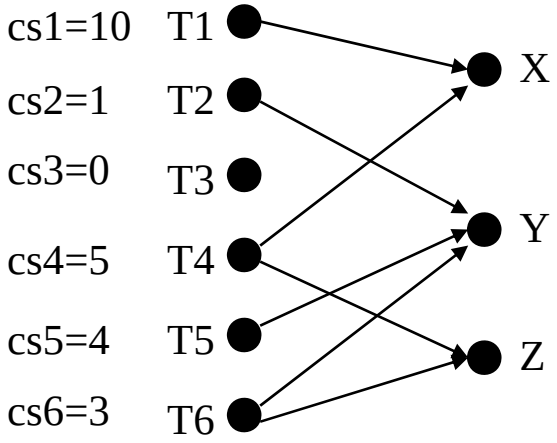
T2 = {.. P(Y) .1. V(Y) ..}

T3 = {.. }

T4 = {..P(X).1.P(Z).2.V(Z).2.V(X).. }

T5 = {..P(Y).4.V(Y).. }

T6 = {..P(Y).1.P(Z).2.V(Z).0.V(Y).. }



	Directly blocked by					Priority-inheritance blocked by					Priority-ceiling blocked by					
	T2	T3	T4	T5	T6	T2	T3	T4	T5	T6	T2	T3	T4	T5	T6	
T1			Y													b1=5
T2	*			Y	Y	*		Y			*		Y			b2=5
T3		*					*	Y	Y	Y		*				b3=5
T4			*		Y			*	Y	Y			*	Y	Y	b4=4
T5				*	Y				*	Y				*	Y	b5=3

Example (More detail analysis)

T1 = {.. P(X) .10. V(X) ..}

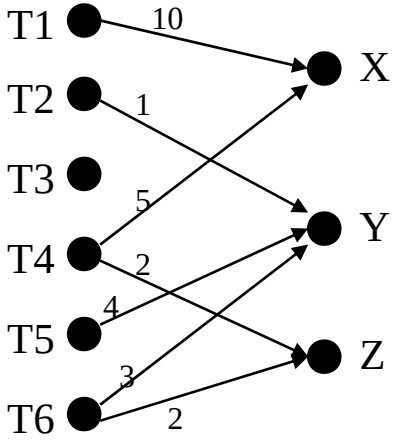
T2 = {.. P(Y) .1. V(Y) ..}

T3 = {.. }

T4 = {..P(X).1.P(Z).2.V(Z).2.V(X).. }

T5 = {..P(Y).4.V(Y).. }

T6 = {..P(Y).1.P(Z).2.V(Z).0.V(Y).. }



	Directly blocked by					Priority-inheritance blocked by					Priority-ceiling blocked by					
	T2	T3	T4	T5	T6	T2	T3	T4	T5	T6	T2	T3	T4	T5	T6	
T1			5													b1=5
T2	*			4	3	*		5			*		5			b2=5
T3		*					*	5	4	3		*				b3=5
T4			*		2			*	4	3			*	4	3	b4=4
T5				*	3				*	3				*	3	b5=3

Schedulability Analysis considering Blocking

- Utilization bound check

  Individual task check $$\sum_{j=1}^{i-1} \frac{e_j}{p_j} + \frac{e_i + b_i(rc)}{p_i} \le U(i) \text{ for all } 1 \le i \le n$$

  System check $$\sum_{i=1}^{n} \frac{e_i}{p_i} + \max_{j=1}^{n} \frac{b_j(rc)}{p_j} \le U(n)$$

- Response time check

$$r_i = e_i + b_i(rc) + \sum_{j=1}^{i-1} \left\lceil \frac{r_i}{p_j} \right\rceil e_j \le d_i \text{ for all } 1 \le i \le n$$