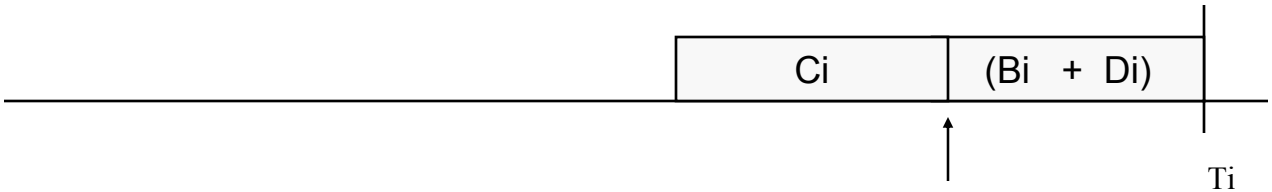
Review 1

- In the lecture on blocking and pre-periodic deadline, we learned that we need to account for the duration of blocking and earlier deadline experienced by task i. There are two ways to accommodate (Bi+Di).
- In the exact test, we just move the deadline forward from Ti to Ti - (Bi+Di) to accommodate the effect of early deadline and blocking EXACTLY.
-
- In the utilization test, we cannot adjust the task deadline directly. We need to adjust the task utilization instead. So we inflate Ci to (Ci + Bi + Di) and check if the task is still schedulable.
  - If it is, then the inflated task must finish by Ti.
  - If with additional (Bi + Di) the task finishes by Ti, then with Ci alone the task must finish (Bi+Di) unit before Ti.
  - Hence Ti can take the delay Bi and finish by (Ti – Di).



Review 2

- Why not reduce Ti to (Ti – Bi – Di)?
- Example: C1 = 0.305 T1 = 1
- C2 = 1, T2 = 2, D2 = 0.4
- C3 = 5, T3 = 1000

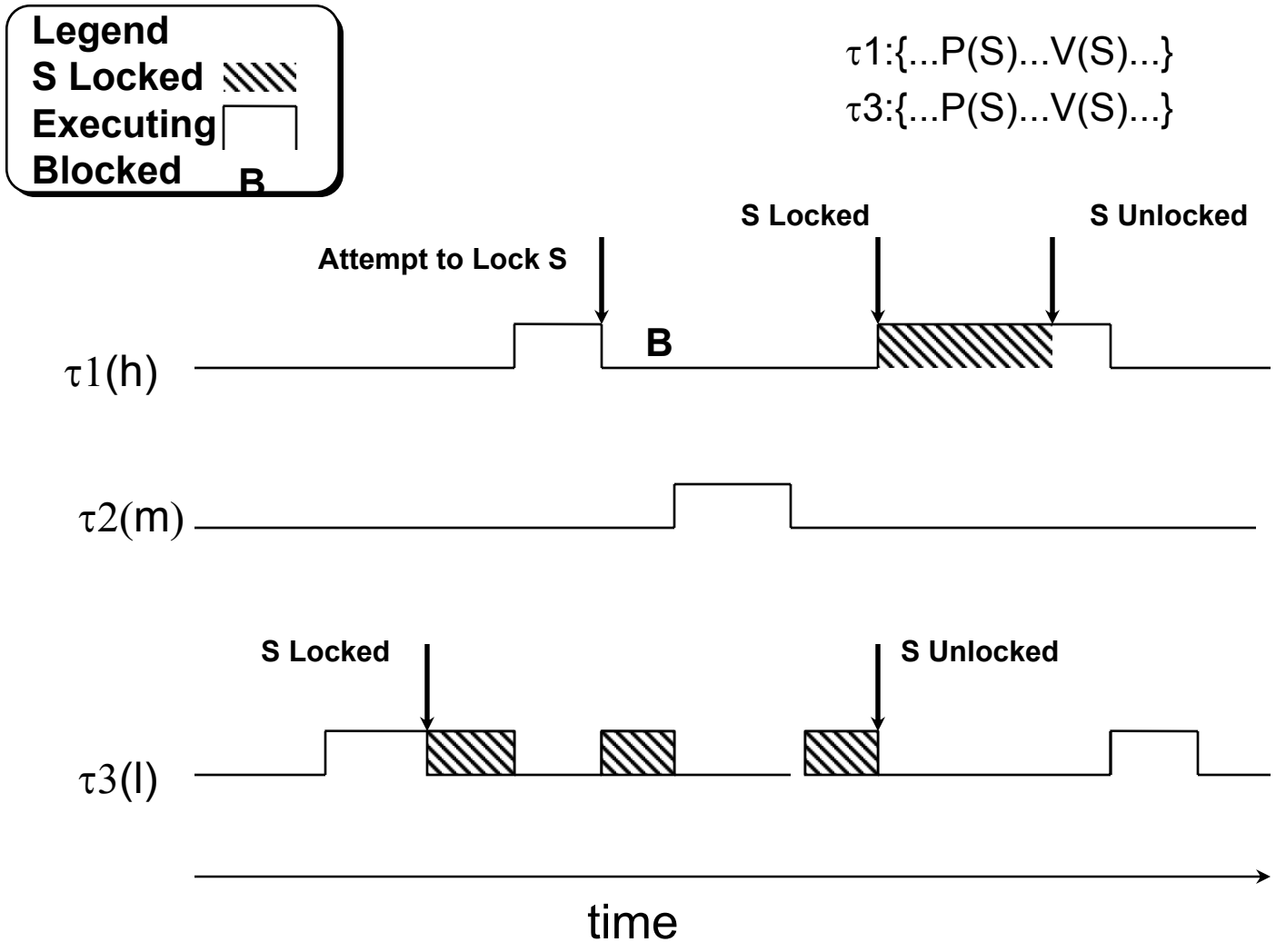
- Note T1 and T2 are harmonic, hence U(2) = 1.

- 0.305/1 + (1+0.4)/2 = 0.305 + 0.7 > U(2), not schedulable

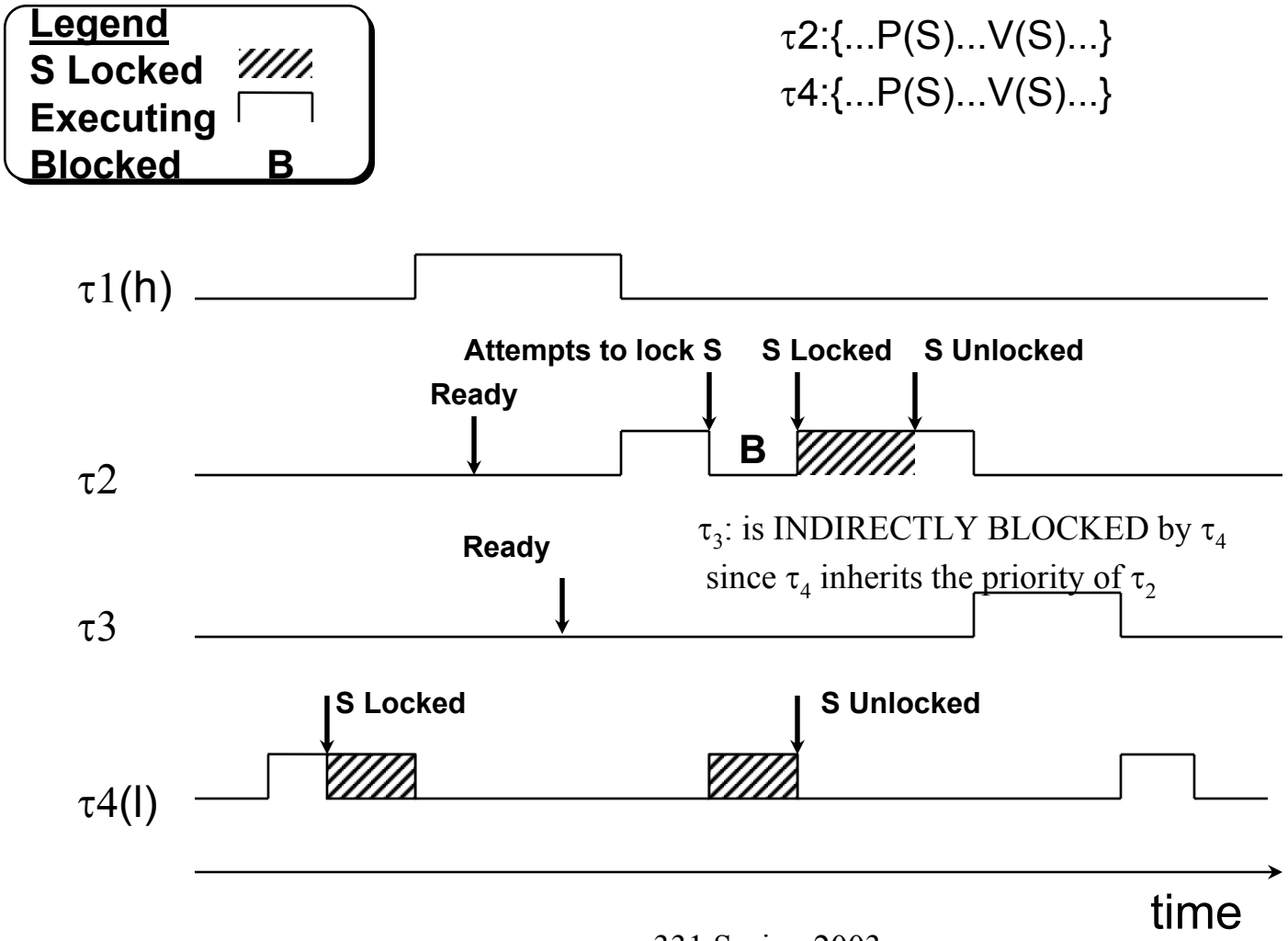
- 0.305/1 + 1/(2 – 0.4) = 0.305 + 0.625 = 0.93 < U(2), schedulable

- Now draw a timeline for task 2, and see which one produces the right answer.

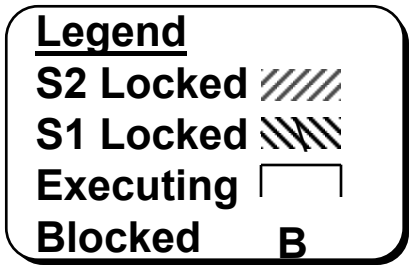
Unbounded Priority Inversion



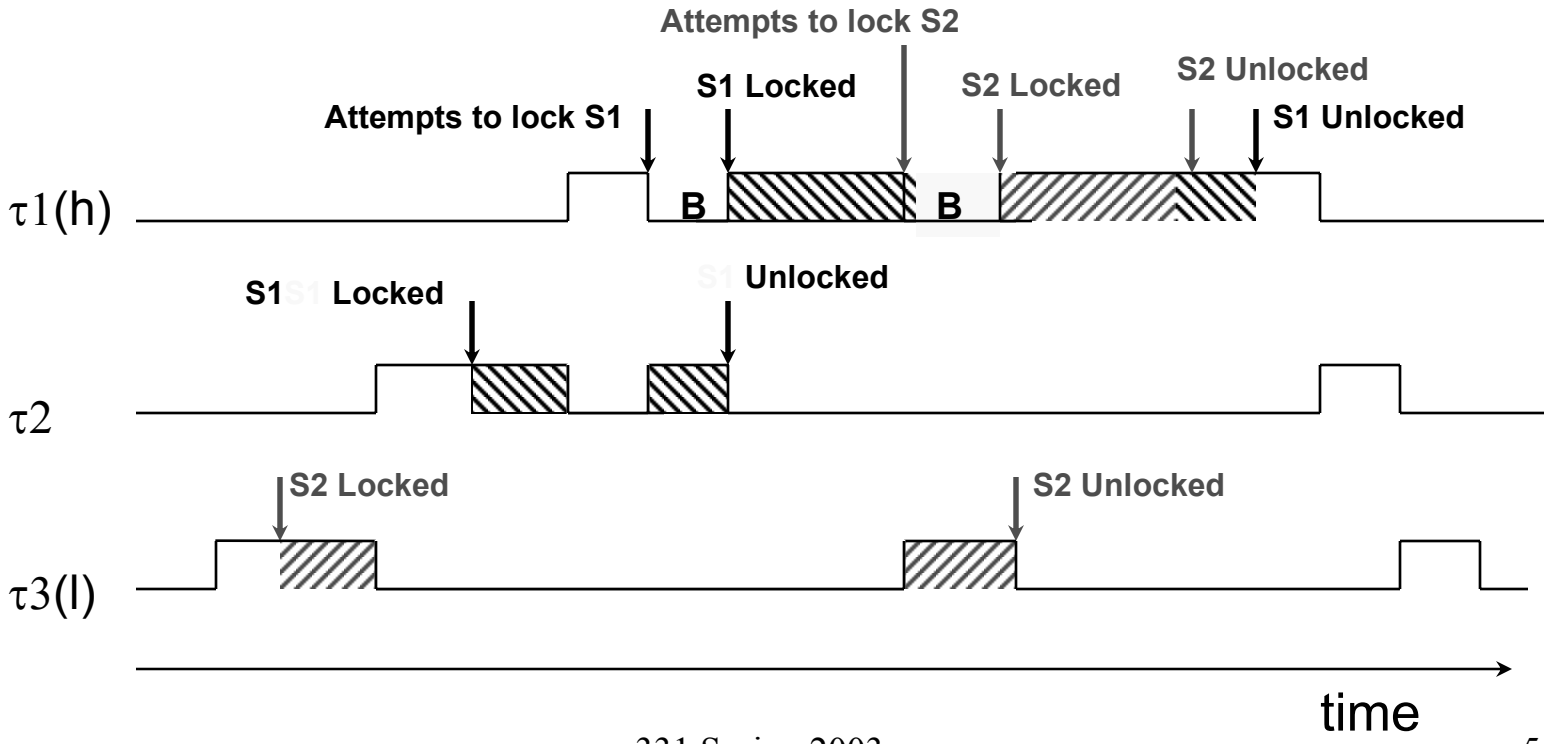
Basic Priority Inheritance Protocol - 1



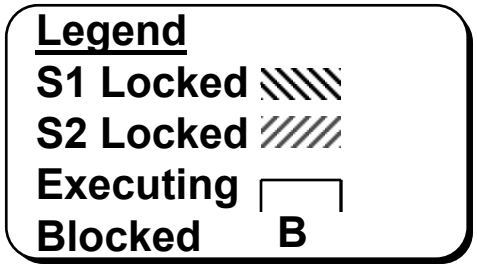
Chained Blocking



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

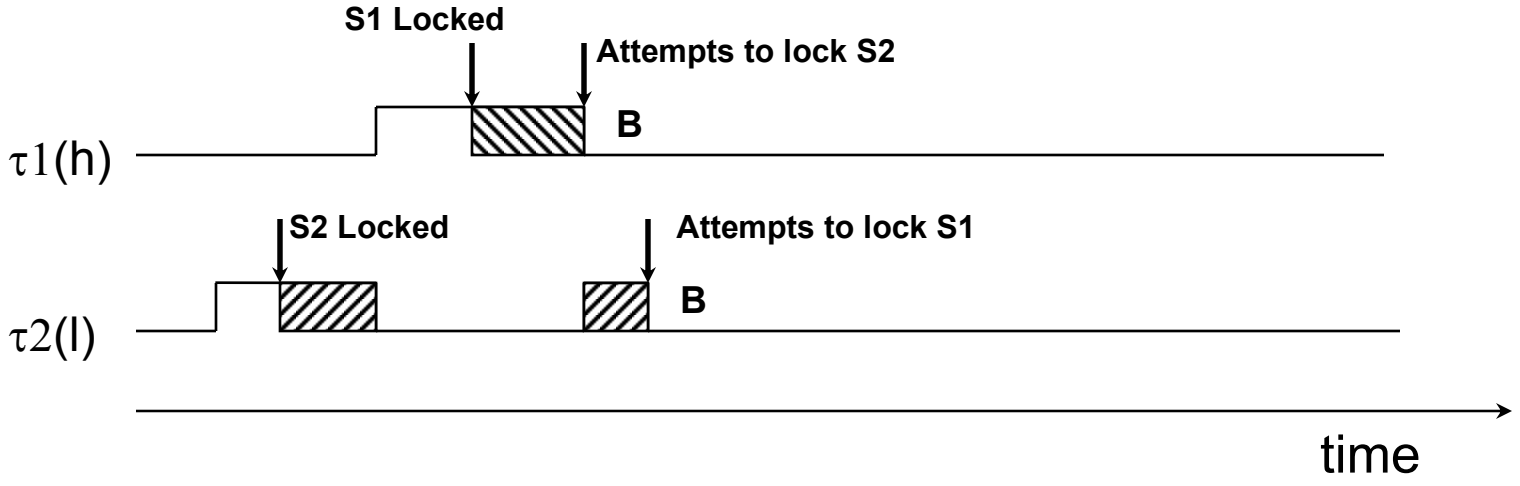


Deadlock Under BIP



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

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



Priority Ceiling Protocol

A priority ceiling is assigned to each semaphore, which is equal to the highest priority task that may use this semaphore.

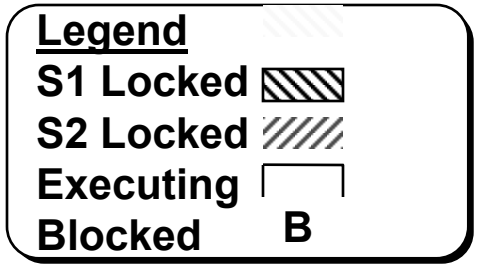
A task can lock a semaphore if and only if its priority is higher than the priority ceilings of all semaphores ALREADY LOCKED by ***other*** tasks.

If a task is blocked by lower priority tasks, the lower priority task inherits its priority.

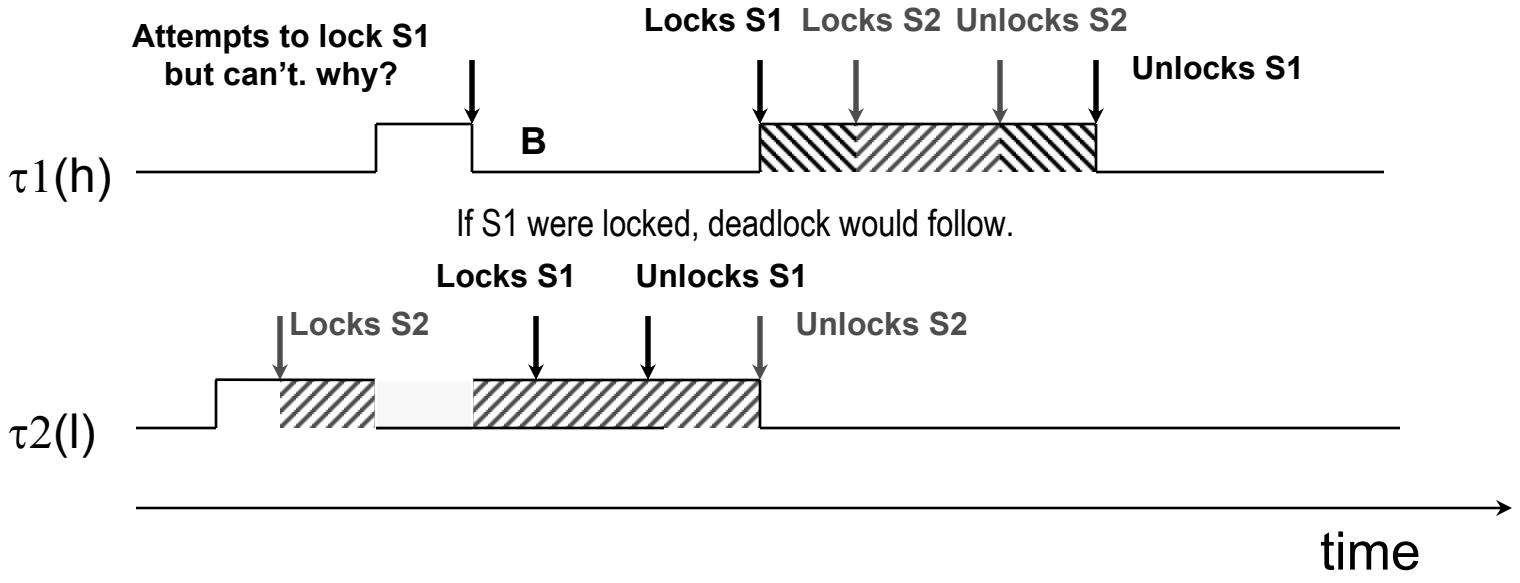
Under priority ceiling protocol, a task can be blocked by lower priority tasks at most once no matter how many semaphores they share. In addition, tasks cannot be deadlocked.

Under priority inheritance protocol, tasks could be deadlocked and chained blocking is a fact of life. But a task is blocked at most by n lower priority tasks sharing resources with it or with higher priority tasks, when there is no deadlock.

Deadlock Avoidance: Using PCP



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

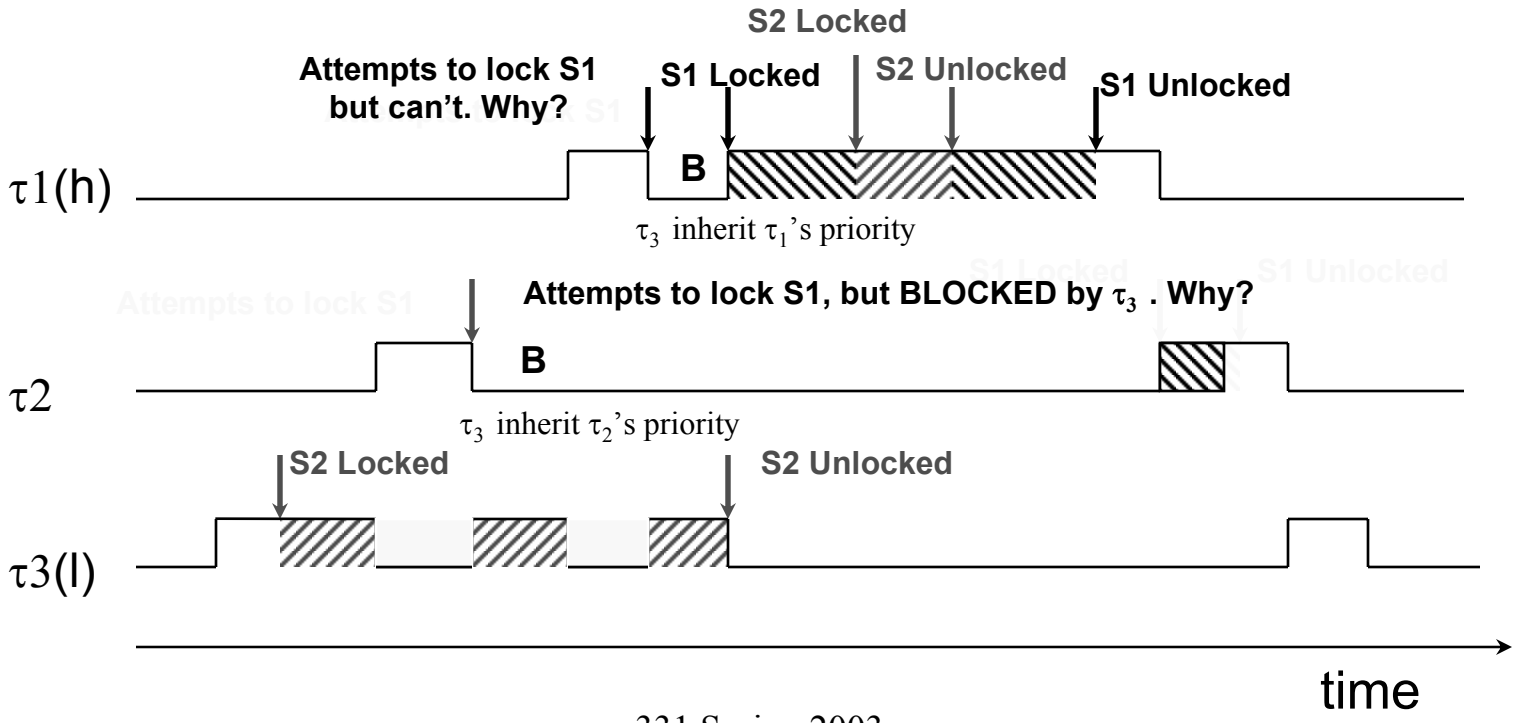


Note: Task τ2 can still lock S1 since it owns the lock, S1 is not locked by OTHER tasks

Blocked at Most Once (PCP)

Legend
S1 Locked
S2 Locked
Executing
Blocked **B**

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



Task Switching and Pre-period Deadline and Blocking

Suppose that a task has D unit of preperiod deadline and blocking time B, we just add them to its execution time for each task in UB test. In exact schedulability test, we will move the deadline from T to (T - D - B)

$$\tau_1 \qquad \frac{(C_1 + 2S + B_1 + D_1)}{T_1} \le U \quad (1)$$

$$\tau_2 \qquad \frac{(C_1 + 2S)}{T_1} + \frac{(C_2 + 2S + D_2 + B_2)}{T_2} \le U \quad (2)$$

$$\tau_3 \qquad \frac{(C_1 + 2S)}{T_1} + \frac{(C_2 + 2S)}{T_2} + \frac{(C_3 + 2S + D_3)}{T_3} \le U \quad (3)$$

Note that B_3 is always zero. It is the task with the longest period and therefore it cannot be blocked by a task with longer period.

Sample Problem

- Suppose that we have three tasks and there are one data structures shared by tasks τ_1 and task τ_2 and another shared by task τ_1 and τ_3. Task τ_2's critical section is 1 unit long while task τ_3's critical section is 2 units long.
- Fill in the blocking times and determine if these 3 tasks are schedulable under PCP. (Note that the critical section is included in the C's, because it is just the part of code that uses the shared data.)

Basic Priority Inheritance

	C	T	B	D
Task τ1	1	4	?	
Task τ2	2	6	?	1
Task τ3	4	13	?	

Priority Ceiling Protocol

	C	T	B	D
Task τ1	1	4	?	
Task τ2	2	6	?	1
Task τ3	4	13	?	

Blocking Under PIP

	C	T	B	D
Task τ1	1	4	3	
Task τ2	2	6	2	1
Task τ3	4	13	0	

Sample Problem with PCP

	C	T	B	D
Task τ1	1	4	2	
Task τ2	2	6	2	1
Task τ3	4	13	0	

$B_1 = 2 = \text{Max}(2,1)$ because of the 'block at most once' property of PCP.

τ_1: $a_o = 1 < (4-2) = 2$ ok

τ_2 $a_0 = 1 + 2 = 3$; $a_1 = 2+\text{ceil}(3/4)*1 = 3 = (6-1-2) = 3$. OK

τ_3 $a_0 = 1+2+4 = 7$,

$a_1 = \text{ceil}(7/4)*1 + \text{ceil}(7/6)*2 + 4 = 10$

$a_2 = \text{ceil}(10/4)*1 + \text{ceil}(10/6)*2 + 4 = 11$

$a_3 = \text{ceil}(11/4)*1 + \text{ceil}(11/6)*2 + 4 = 11 < 13$ Ok

Summary

- In the last lecture and today's lecture, we have learned the issues in real time synchronization:
  - the priority inversion problem
  - the Basic Priority Inheritance Protocol
  - the Priority Ceiling Protocol
  - Schedulability analysis using real time synchronization protocol.
  - direct blocking
  - indirect blocking

  Remember: When a task is in between a high and a low priority tasks sharing a lock, it can be indirectly blocked under both BIP and PCP