Lecture Note #6: More on Task Scheduling
EECS 571
Principles of Real-Time Embedded Systems

Kang G. Shin
EECS Department
University of Michigan

Mars Pathfinder Timing Hiccups?

- **When: landed on the Mar on July 4, 1997**
- **Mission: gathering and sending back voluminous data to Earth.**
- **Problem: resetting the entire computer system, each reset resulting in data loss (``software glitches?'')**

- **System Used and Cause Found:**
  - Wind River System's VxWorks that supports preemptive priority scheduling.
  - ``Information bus'' ($\equiv$ shared memory) accessed via mutexes. The *bus management task* T_1 ran frequently with high priority
  - *Meteorological data collection task* T_3 is an infrequent, low-priority task, and uses the info bus to publish its data. (It acquires mutex, writes data to the bus, then releases the mutex.)
  - $\exists$ a medium-priority, long *commercial task* T_2.
  - Scenario: T_1 blocks on the mutex held by T_3 and in the meantime T_2 preempts T_3; not executing T_1 for certain time triggers a watchdog timer resetting the system.

- **How was the problem solved: A short C program was uploaded to the spacecraft which changed parameter initialization.**

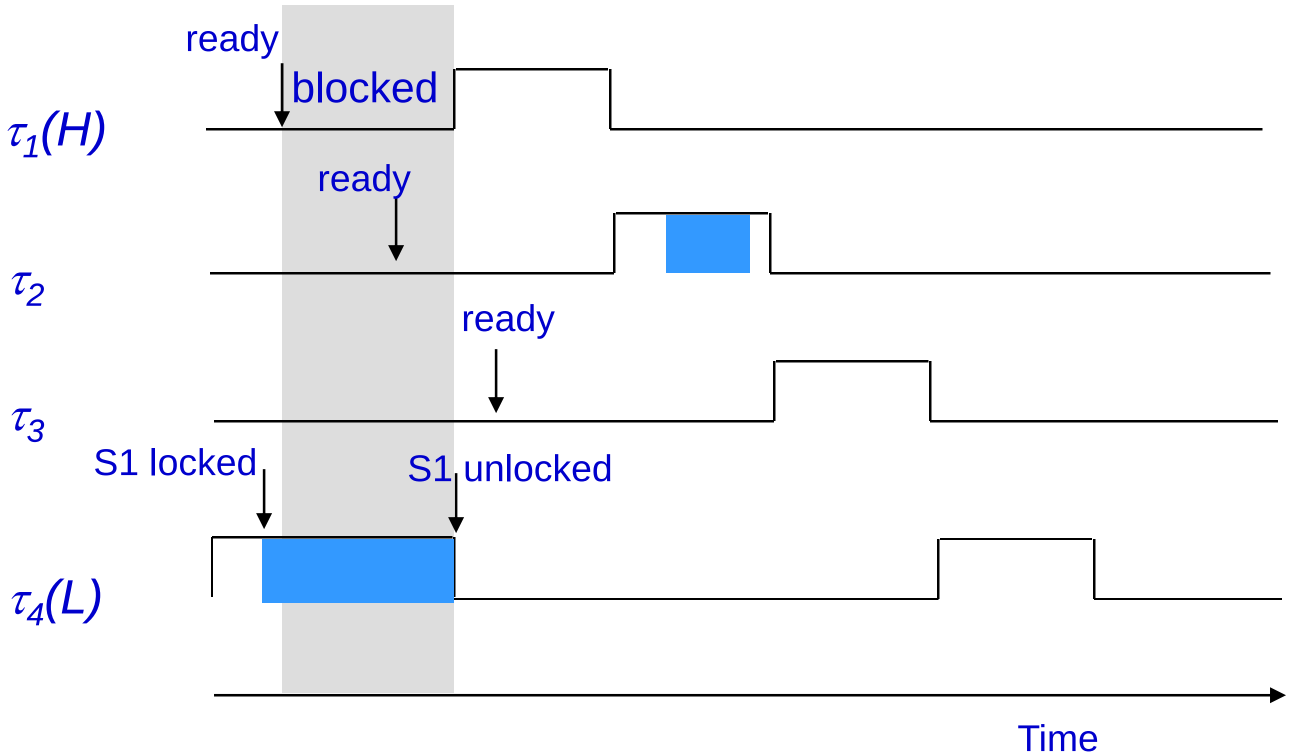
Synchronization Protocols

- Non-preemption
- Basic priority inheritance
- Highest locker's priority
- Priority ceiling
- All of these prevent unbounded priority inversion.

Non-preemption Protocol

τ_2:{…P(S1)…V(S1)…}

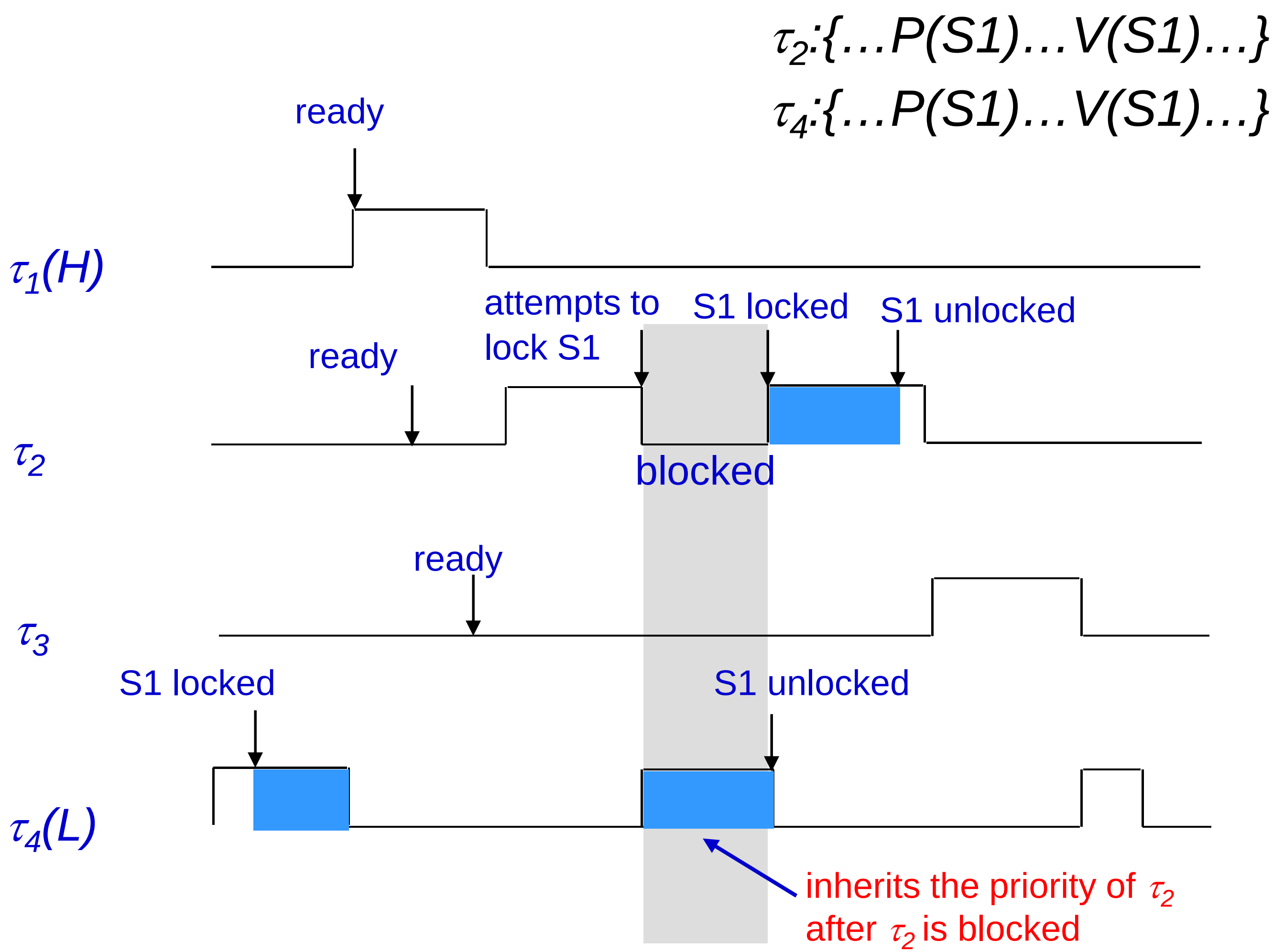
τ_4:{…P(S1)…V(S1)…}



Advantages and Disadvantages

- **Advantages:**
  - Simplicity
  - Use with fixed-priority and dynamic-priority systems
  - No prior knowledge about resource requirement by each task
  - Good when all critical sections are short
- **Disadvantages:**
  - Every task can be blocked by every lower priority task, even when there is no resource sharing between the tasks.
  - Blocking time: $max(cs_i)$

Basic Inheritance Protocol (BIP)



Some Notations

- **J_i is the i-th job of periodic task T.**
- **π_i = job J_i's assigned priority**
- **$\pi_i(t)$ = current (at time t) priority of J_i**
- **If the decision to change the priority of J_i is made at $t = t_1$ then**
  - $\pi_i(t_1^-)$ = priority at and immediately before t_1,
  - $\pi_i(t_1^+)$ = priority immediately after the priority change

- **Ω = nonexistent priority, lower than the lowest priority**

Terminology and Rules

- At time t_1, job J_i requests resource R_k.
- $R_k \rightarrow J_l$: Resource R_k is held by job J_l
- $J_i \rightarrow R_k$: Job J_i is blocked waiting for resource R_k to be released ($J_i \rightarrow R_k \rightarrow J_l$)
- ***Scheduling Rules*:**
  - Ready jobs are scheduled on processors preemptively according to their current priorities, $\pi_i(t)$.
  - Upon release of a job, its priority is equal to its assigned priority
    - if J_i is released at $t = t'$, then $\pi_i(t') = \pi_i$
- ***Resource allocation*:**
  - If a resource is free, then it is allocated when it is requested
  - if not free, then the request is denied and the requesting job is blocked

Priority Inheritance Rules

- ***Scheduling Rule:*** **same as the assumptions**
- ***Allocation Rule:*** **same as the assumptions**
- ***Priority-Inheritance Rule:***
  - if $J_i \rightarrow R_k \rightarrow J_l$ and $\pi_l(t_1^-)$ = priority of J_l at $t = t_1$
  - then $\pi_l(t_1^+) = \pi_i(t_1)$
  - until J_l releases R_k at t_2 when $\pi_l(t_2^+) = \pi_l(t_1^-)$

- **L. Sha, R. Rajkumar, J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization",** ***IEEE Transactions on Computers*****, Vol. 39, No. 9, pp. 1175-1185, 1990**

Properties of Priority Inheritance

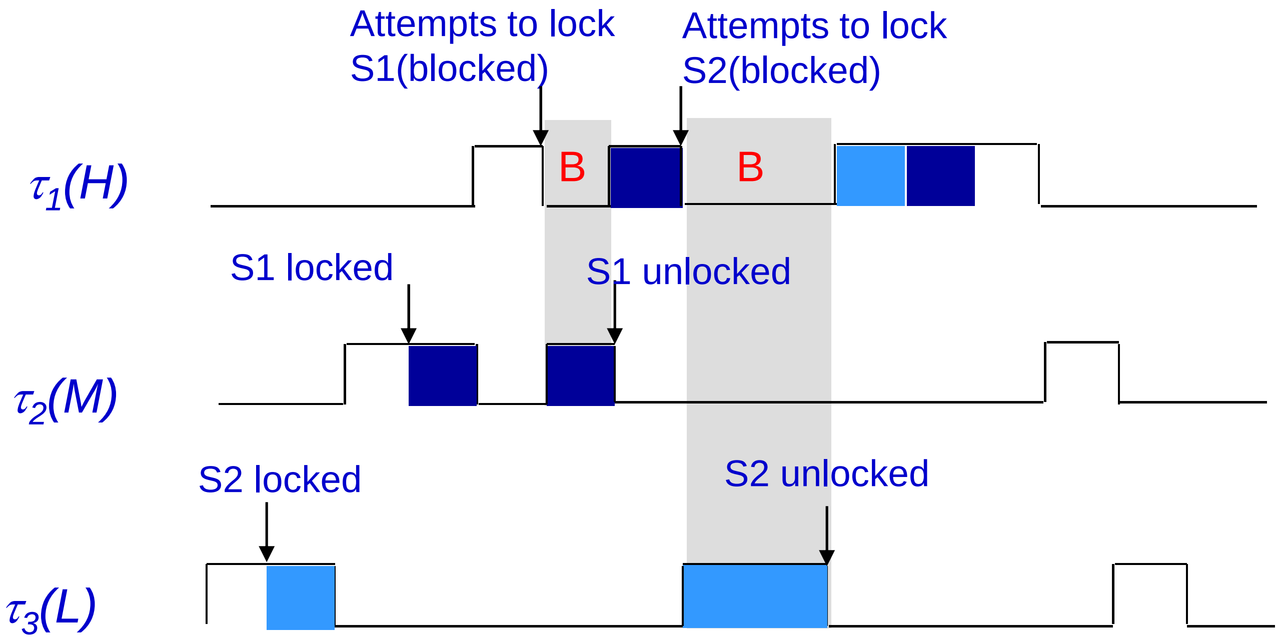
- For each resource (semaphore), a list of blocked tasks must be stored in a priority queue.
- A task (job) τ_i uses its assigned priority, and inherits the highest dynamic priority of all the tasks it blocks when it is in its critical section and blocks some higher priority tasks.
- Priority inheritance is *transitive*; that is, if task τ_i blocks τ_j and τ_j blocks τ_k , then τ_i can inherit the priority of τ_k.
- When τ_i releases a resource, which priority should it use?
- Chained/nested blocking if requesting multiple resources (nested mutex requests)
- Direct blocking and *indirect* (inheritance) blocking (when the lower priority task inherits the higher priority task's priority).

Example of Chained/nested Blocking (BIP)

τ_1:{…P(S1)…P(S2)…V(S2)…V(S1)…}

τ_2:{…P(S1)…V(S1)…}

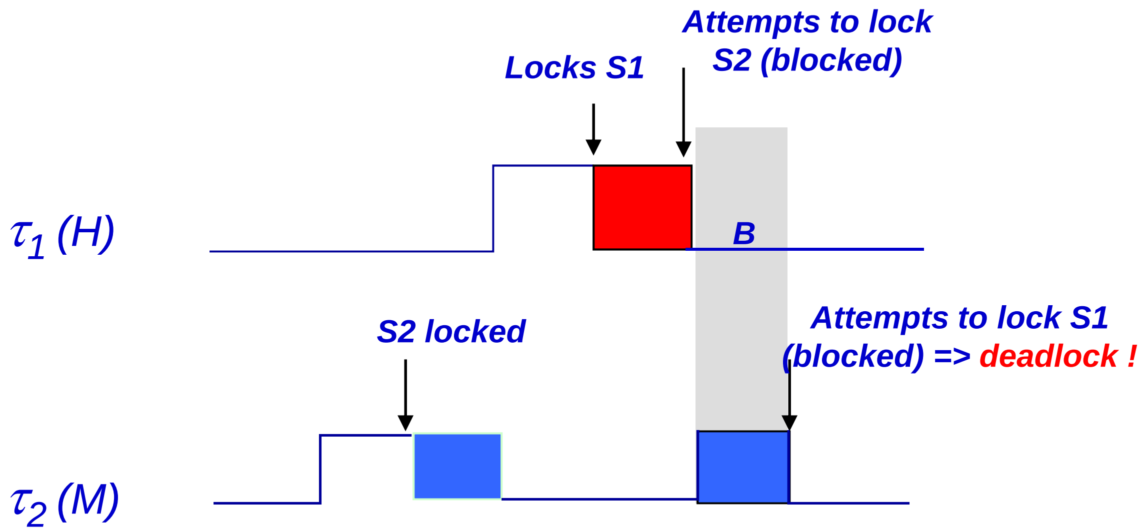
τ_3:{…P(S2)…V(S2)…}



Deadlock: Using BIP

τ_1 :{…P(S1)…P(S2)…V(S2)…V(S1)..}

τ_2 :{…P(S2)…P(S1)…V(S1)…V(S2)..}



Blocking Time Under BIP

- **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).. }

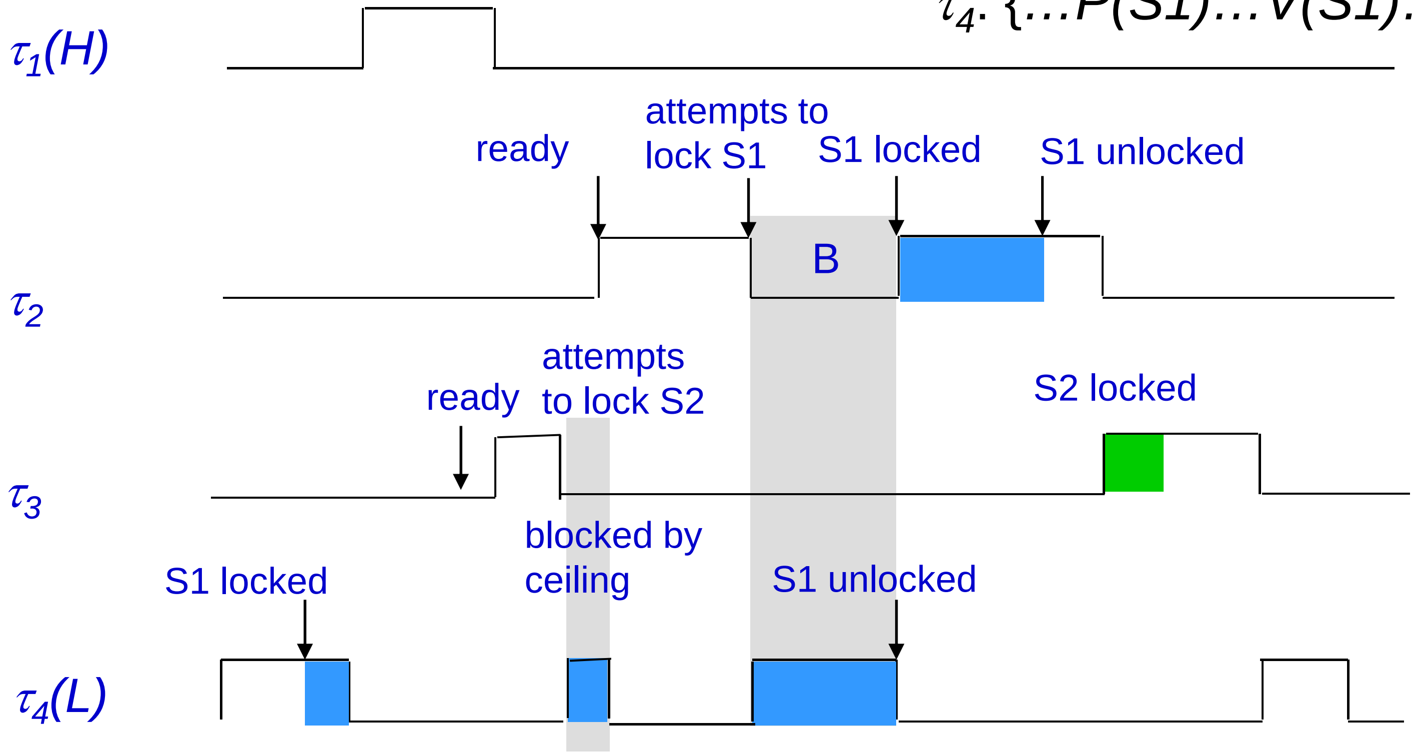
	direct blocking by			indirect blocking by			blocking time		
	T2	T3	T4	T2	T3	T4	T2	T3	T4
T1	N	Y	Y					5	7
T2		N	Y		Y	Y		5	7
T3			Y			Y			7

Priority Ceiling Protocol (PCP)

τ_2: {…P(S1)…V(S1)…}

τ_3: {…P(S2)…V(S2)…}

τ_4: {…P(S1)…V(S1)…}



Basic Priority Ceiling Rules (1)

- $\Pi(R)$ = priority ceiling of resource R – the highest priority of the tasks that request R
- $\Pi_S(t)$ = system priority ceiling – the highest priority ceiling of all resources that *are in use* at time t
- *Scheduling Rule:* same as the assumptions
- *Allocation Rule:*
  - if $J_i \rightarrow R_k \rightarrow J_l$ at $t \leq t_1$ then block J_i *(no change)*
  - if R_k becomes free at t_1,
    - if $\pi_i(t_1) > \Pi_S(t_1)$, then $R_k \rightarrow J_i$
    - else (i.e., $\pi_i(t_1) \leq \Pi_S(t_1)$)

      if for some $R_x \rightarrow J_i$ and $\Pi(R_x) = \Pi_S(t_1)$, then $R_k \rightarrow J_i$

      [J_i holds resource R_x whose priority ceiling is $\Pi_S(t_1)$]
    - else deny and block ($J_i \rightarrow R_k$)

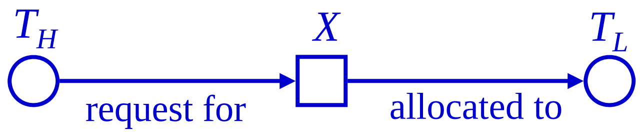
Basic Priority Ceiling Rules (2)

- *Priority-Inheritance Rule:*
  - if $J_i \rightarrow R_k$ at $t = t_1$ and is blocked by J_l (and $\pi_l(t_1^-)$ = priority of J_l)
    - either $R_k \rightarrow J_l$, (J_l holds the resource R_k)

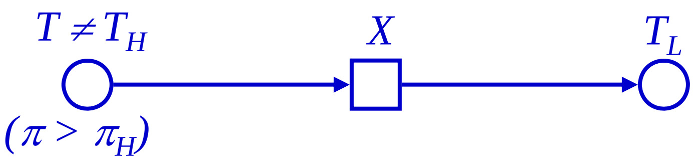
      or $J_l \rightarrow R_x$ and $\Pi(R_x) = \Pi_S(t_1) \geq \pi_i(t_1)$
  - then $\pi_l(t_1^+) = \pi_i(t_1)$ (*inherited priority*)
  - until J_l releases all R_x with $\Pi(R_x) \geq \pi_i(t_1)$, $\pi_l(t_2^+) = \pi_l(t_1^-)$ at $t = t_2$.

Blocking in PCP

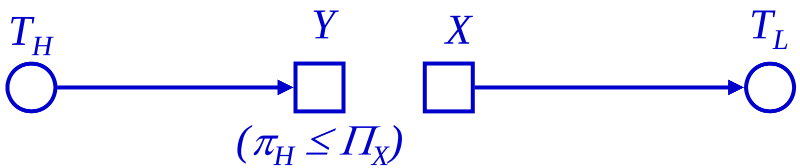
- **A task T_H can be blocked by a lower-priority task T_L in three ways:**
  - directly, i.e.,



  - when T_L inherits a priority higher than the priority π_H of T_H.



  - When T_H requests a resource and the priority ceiling of all resources held by T_L is equal to or higher than π_H:

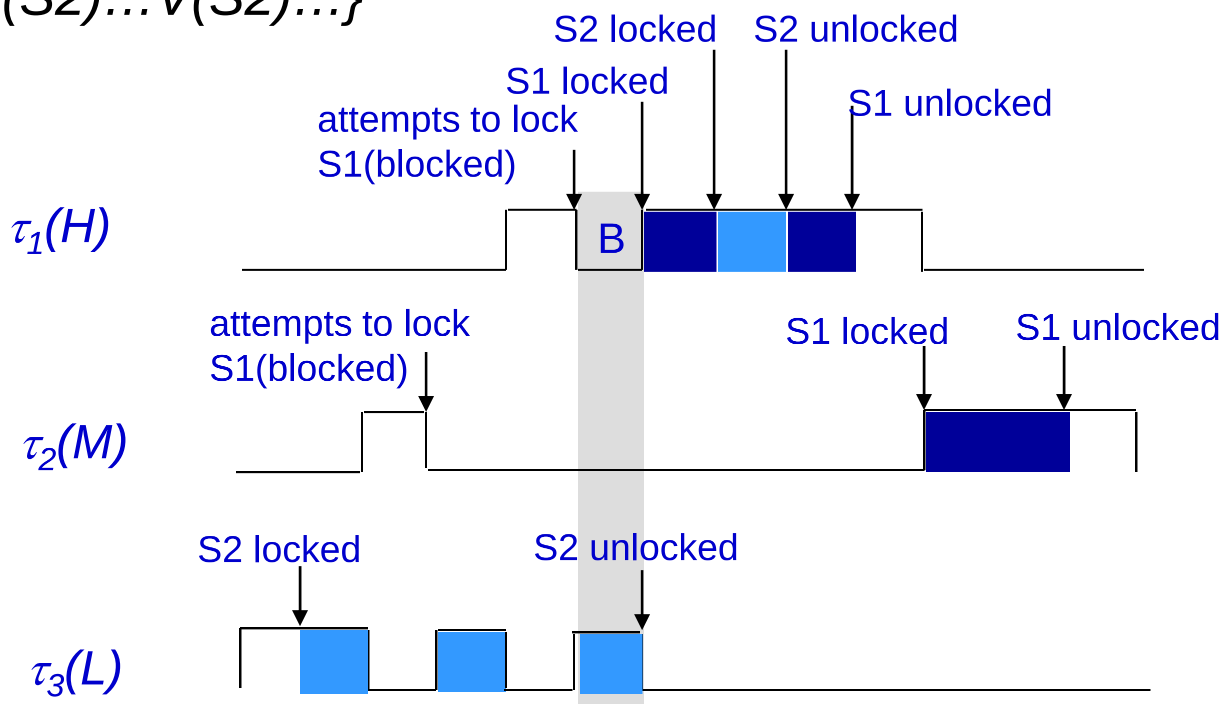


Blocked At Most Once (PCP)

τ_1:{…P(S1)…P(S2)…V(S2)…V(S1)…}

τ_2:{…P(S1)…V(S1)…}

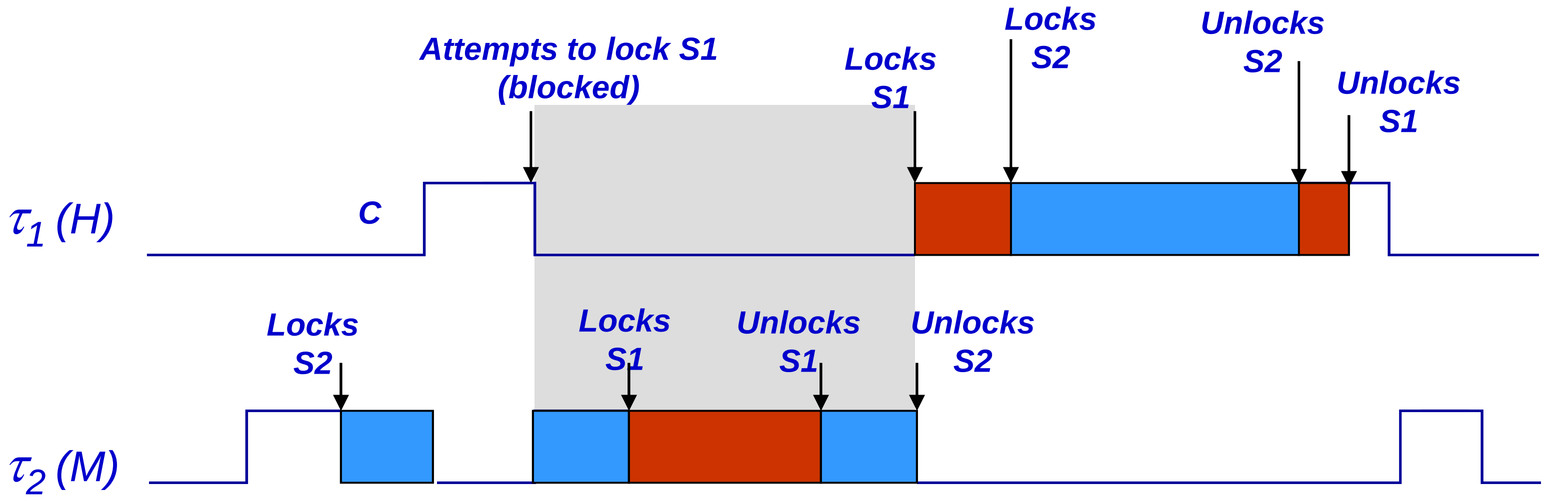
τ_3:{…P(S2)…V(S2)…}



Deadlock Avoidance: Using PCP

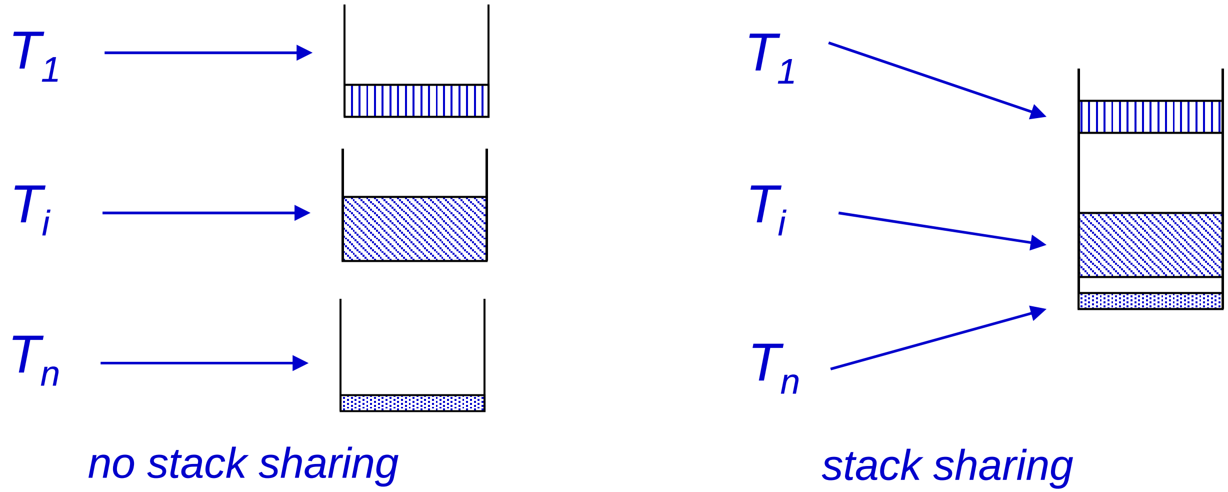
τ_1 :{…P(S1)…P(S2)…V(S2)…V(S1)..}

τ_2 :{…P(S2)…P(S1)…V(S1)…V(S2)..}



Stack Sharing

- Sharing a stack among multiple tasks eliminates stack space fragmentation, making memory savings:



- However:
  - Once job is preempted, it can only resume when it returns to be on top of stack.
  - Otherwise, it may cause a deadlock.
  - Stack becomes a resource that allows for “one-way preemption”.

Stack-Based Priority Ceiling Protocol

- **<u>To avoid deadlocks</u>: Once execution begins, make sure that job is not blocked due to resource access**
  - allow preemption only if the priority is higher than the ceiling of all resources in use

- **Update Current Ceiling in the usual manner**
  - If no resource allocated, $\Pi_S(t) = \Omega$
- **Scheduling Rule:**
  - J_i released and blocked until $\pi_i(t) > \Pi_S(t)$
  - When not blocked, jobs are scheduled in the usual manner.
- **Allocation Rule:**
  - Allocate when requested

Stack-Based PCP, cont'd

- **The Stack-Based Priority-Ceiling Protocol is deadlock-free:**
  - When a job begins to execute, all the resources it will ever need are free.
  - Otherwise, $\Pi_S(t)$ would be higher or equal to the priority of the job.
  - Whenever a job is preempted, all the resources needed by the preempting job are free.
  - The preempting job can complete, and then the preempted job can resume.
- **Worst-case blocking time of Stack-Based Protocol is the same as for Basic Priority Ceiling Protocol.**
- **Stack-Based Protocol smaller context-switch overhead**
  - 2 context switches since once execution starts, job cannot be blocked (may be preempted)
  - 4 context switches for PCP since a job may be blocked at most once

Ceiling-Priority Protocol

- **Re-formulation of stack-based priority ceiling protocol for multiple stacks (w/o stack-sharing)**
- **Update Current Ceiling in the usual manner**
- *Scheduling Rule*:
  - No resources held by J_i, $\pi_i(t) = \pi_i$
  - Resource held by J_i, $\pi_i(t) = max(\Pi(R_x))$ for all resources R_x held by J_i
  - FIFO scheduling among jobs with equal priority
- *Allocation Rule*:
  - Allocate when requested

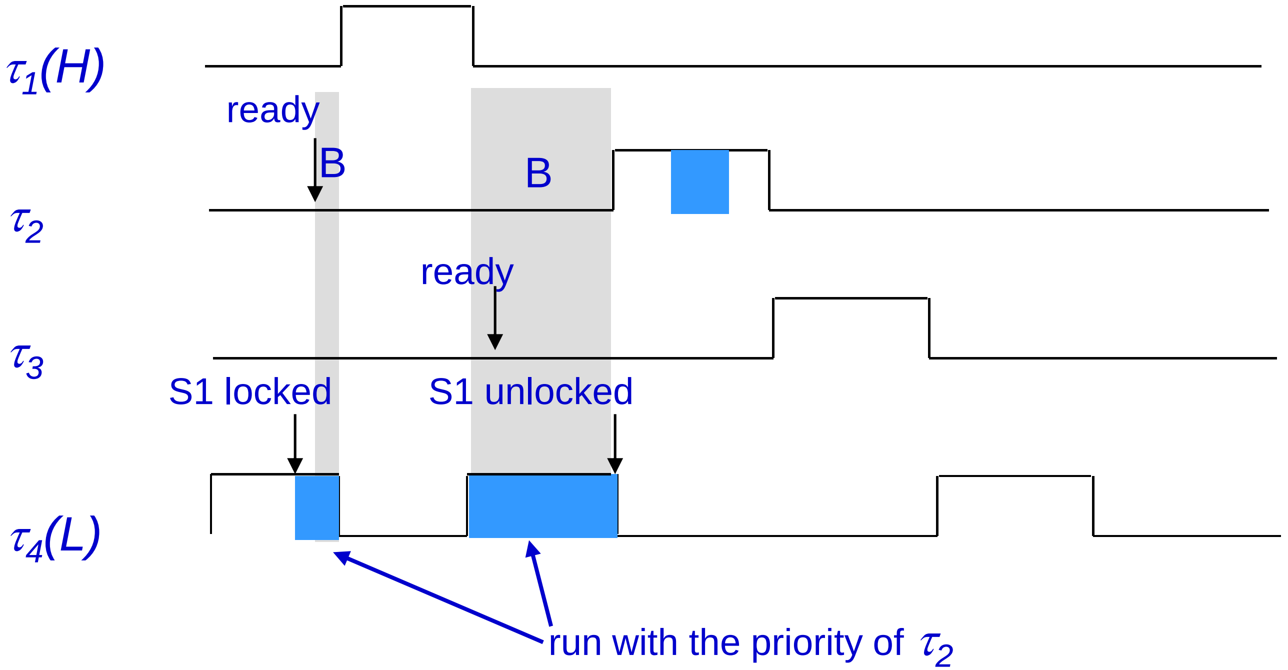
Comparison

- **Worst-case performance of stack-based and basic ceiling protocols are the same**
- **Stack-based version**
  - supported by the Real-Time systems Annex of Ada95
  - Jobs must not self-suspend
- **When jobs do not self-suspend, stack-based and ceiling-priority protocols yield the same schedules.**
- **Stack-based and ceiling-priority have the same worst-case blocking time.**

Highest Locker's Priority Protocol

τ_2:{…P(S1)…V(S1)…}

τ_4:{…P(S1)…V(S1)…}



Summary of Synchronization Protocols

Protocol	Bounded Priority Inversion	Blocked at Most Once	Deadlock Avoidance
Nonpreemptible critical sections	Yes	Yes[1]	Yes[1]
Highest locker's priority	Yes	Yes[1]	Yes[1]
Basic inheritance	Yes	No	No
Priority Ceiling	Yes	Yes[2]	Yes

[1] *Only if tasks do not suspend within critical sections*

[2] *PCP is not affected if tasks suspend within critical sections.*