EEE499 - Real-Time Embedded System Design

Schedulability Part 3

Simple Task Model

- Assumptions:
  1. ~~Tasks are periodic and the period is constant~~
  2. Completion-time < period
  3. Tasks are independent
  4. Runtime is known and deterministic
  5. all system overheads are negligible or deemed to be included in task computation times
  6. Critical instant - defined as the maximum load condition when all tasks release together

- Constraints
  1. ~~Deadline = period~~
  2. fixed set of tasks
  3. Preemptive

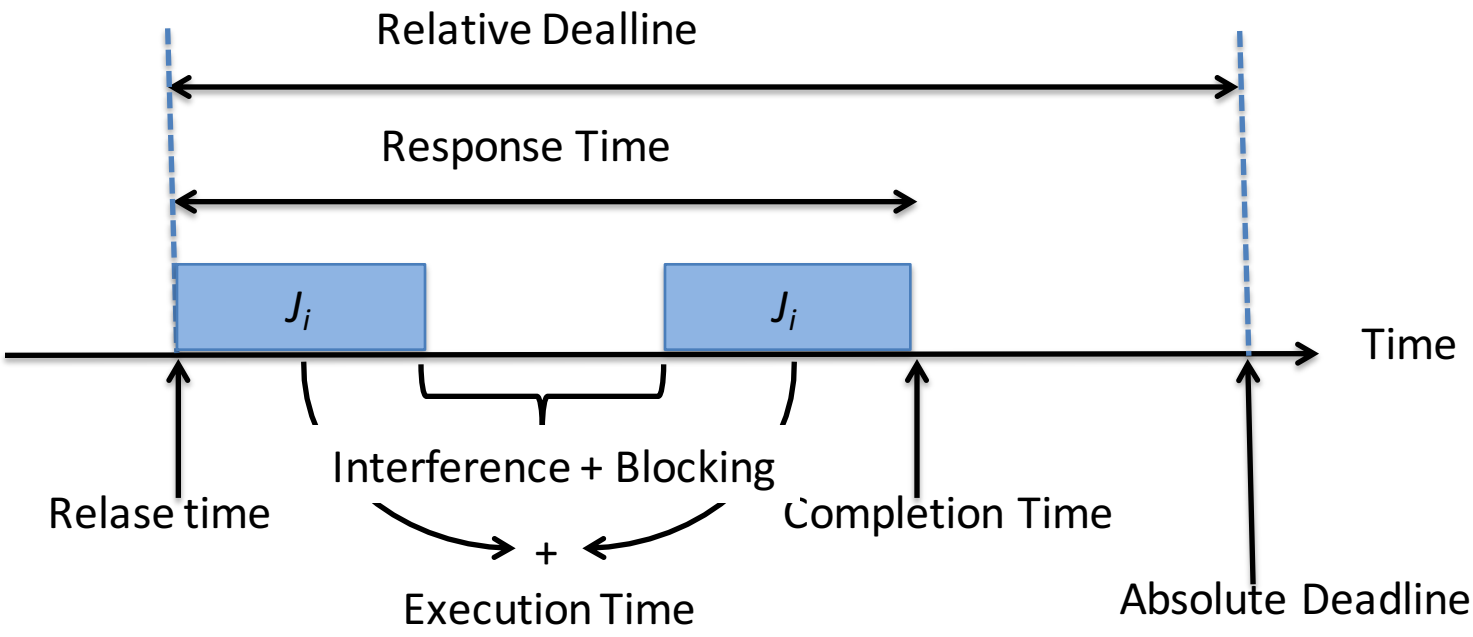
Task Dependency

- The assumption that tasks are independent is not reasonable for all real systems
  - Tasks normally share common resources with semaphores and monitors
  - Tasks must often synchronize
- Tasks can be suspended for a future event that depends on one or more other tasks

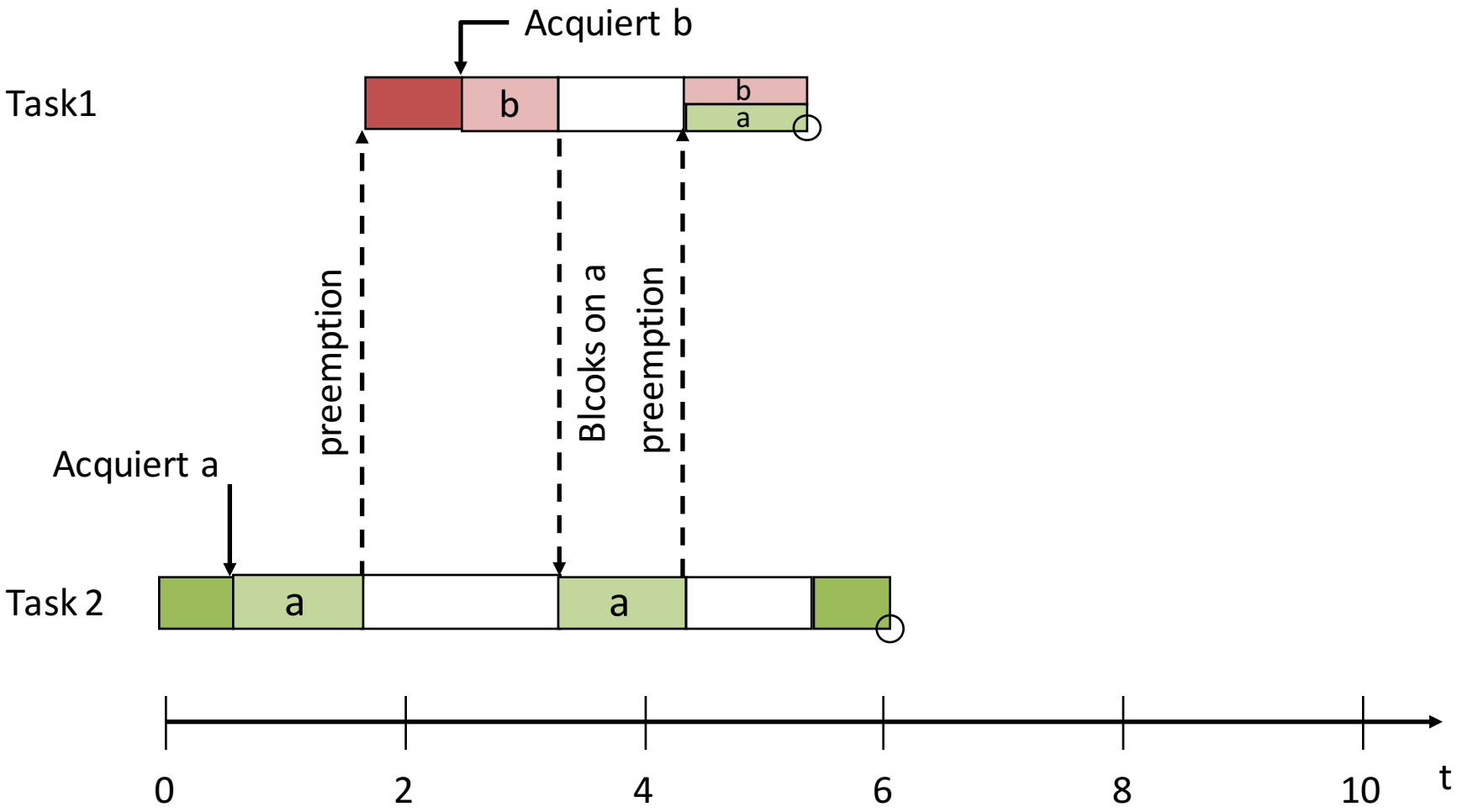
When will this dependence affect the scheduling criterion?

Recall Response Time

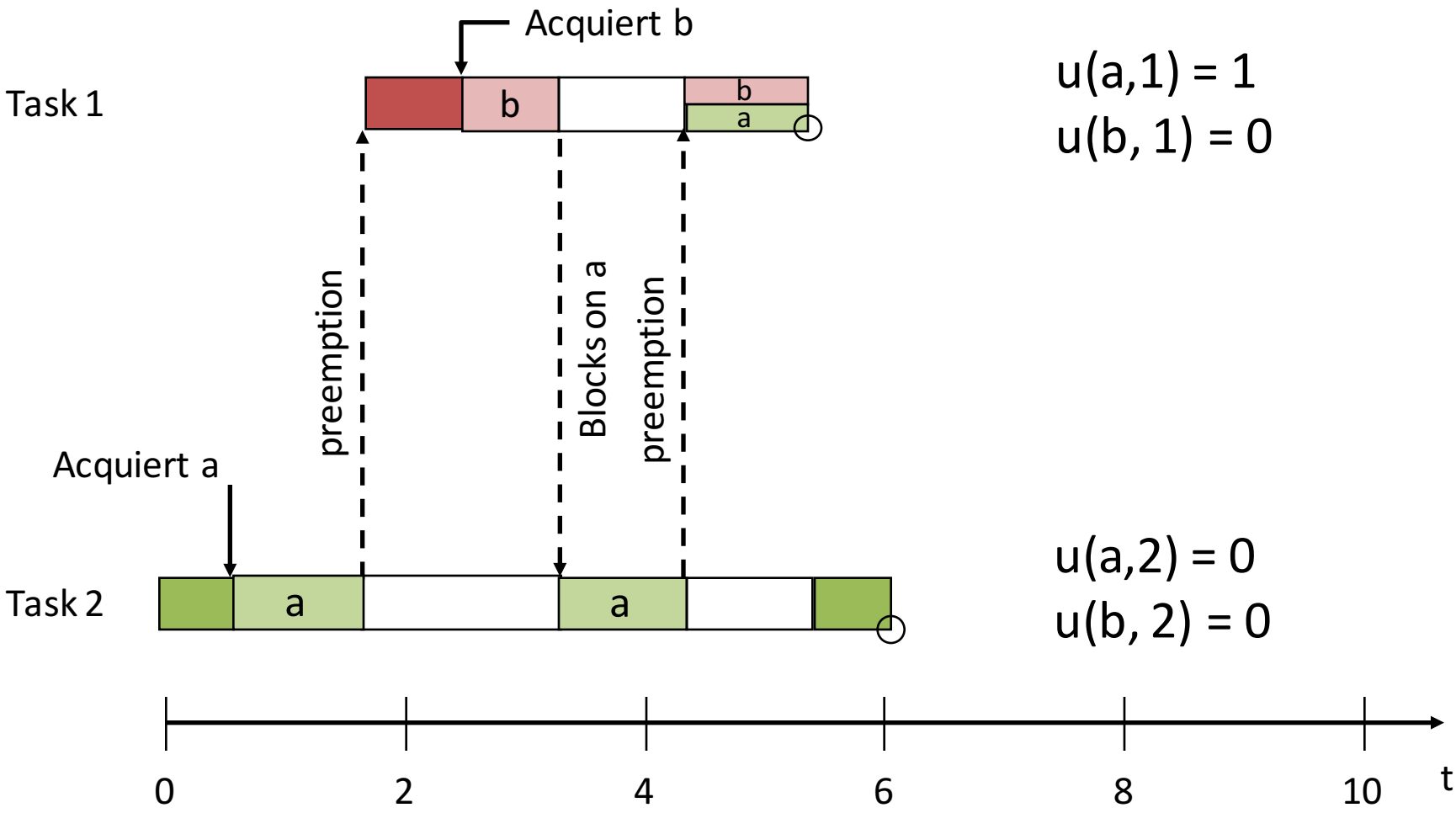
$$R_i = e_i + I_i + B_i \qquad (1)$$



BlockingTime



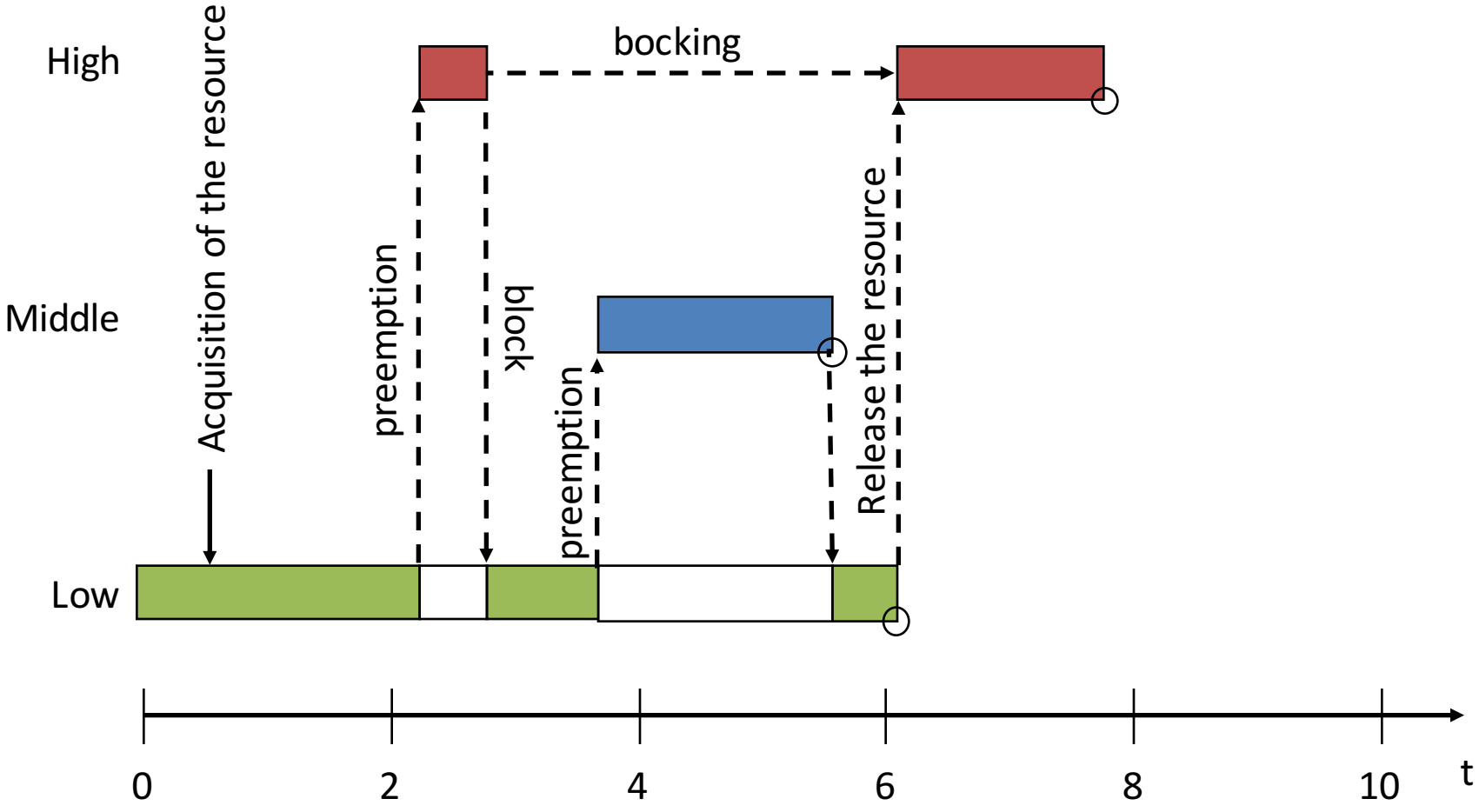
Blocking Time



u(a,1) = 1
u(b, 1) = 0

u(a,2) = 0
u(b, 2) = 0

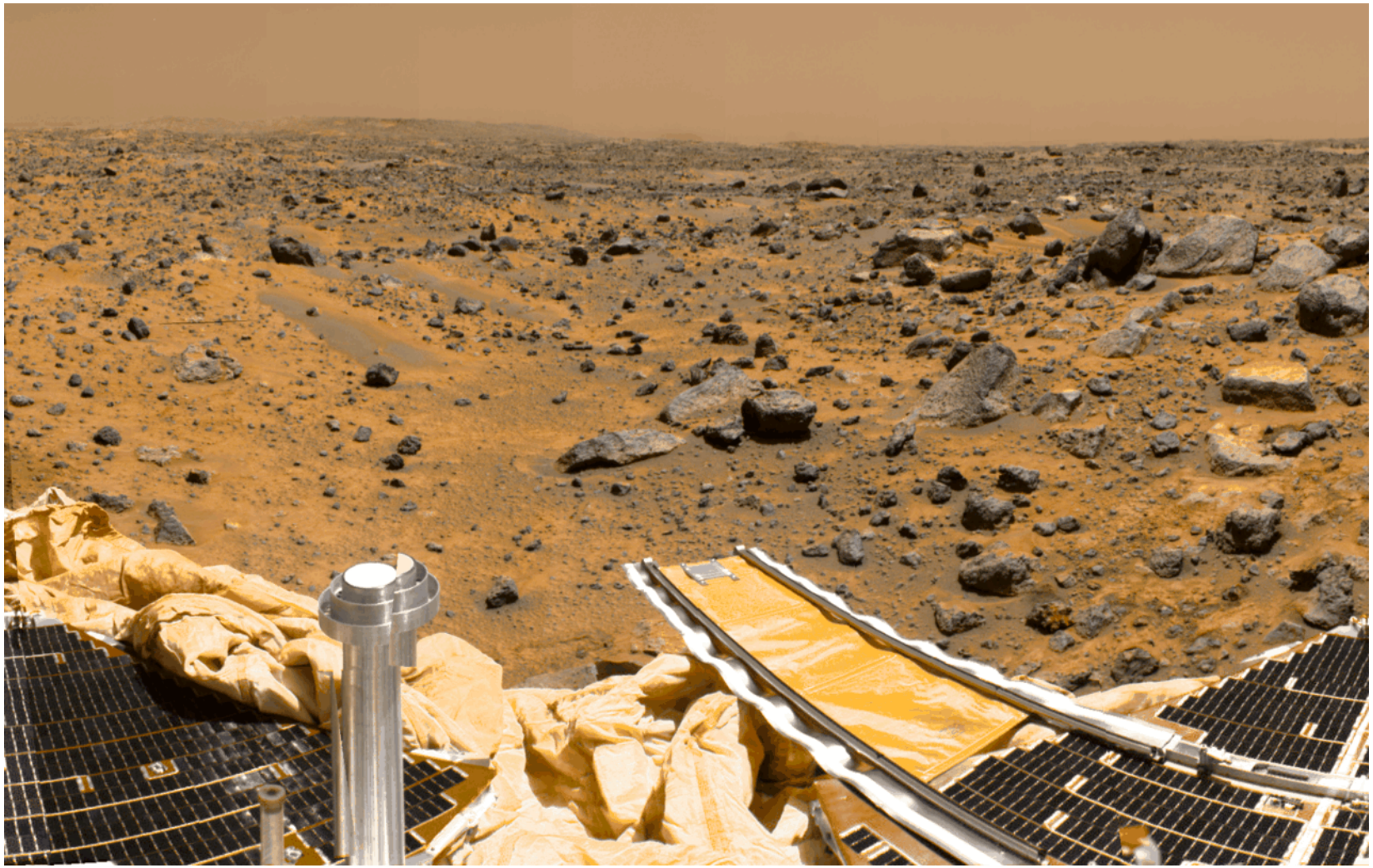
Priority Inversion



Priority Inversion

- Can occur when a high priority task and a low priority task share a common resource
  - The low priority task gets exclusive access to the shared resource
  - Higher priority task causes preemption of lowest priority task but gets blocked while waiting for resource to be released
  - Meanwhile a medium priority task still causes the preemption of the low priority task, further delaying the execution of the high priority task
- There is a priority override because the medium task is guaranteed a higher priority service than the highest priority task that is blocked

Priority Inversion

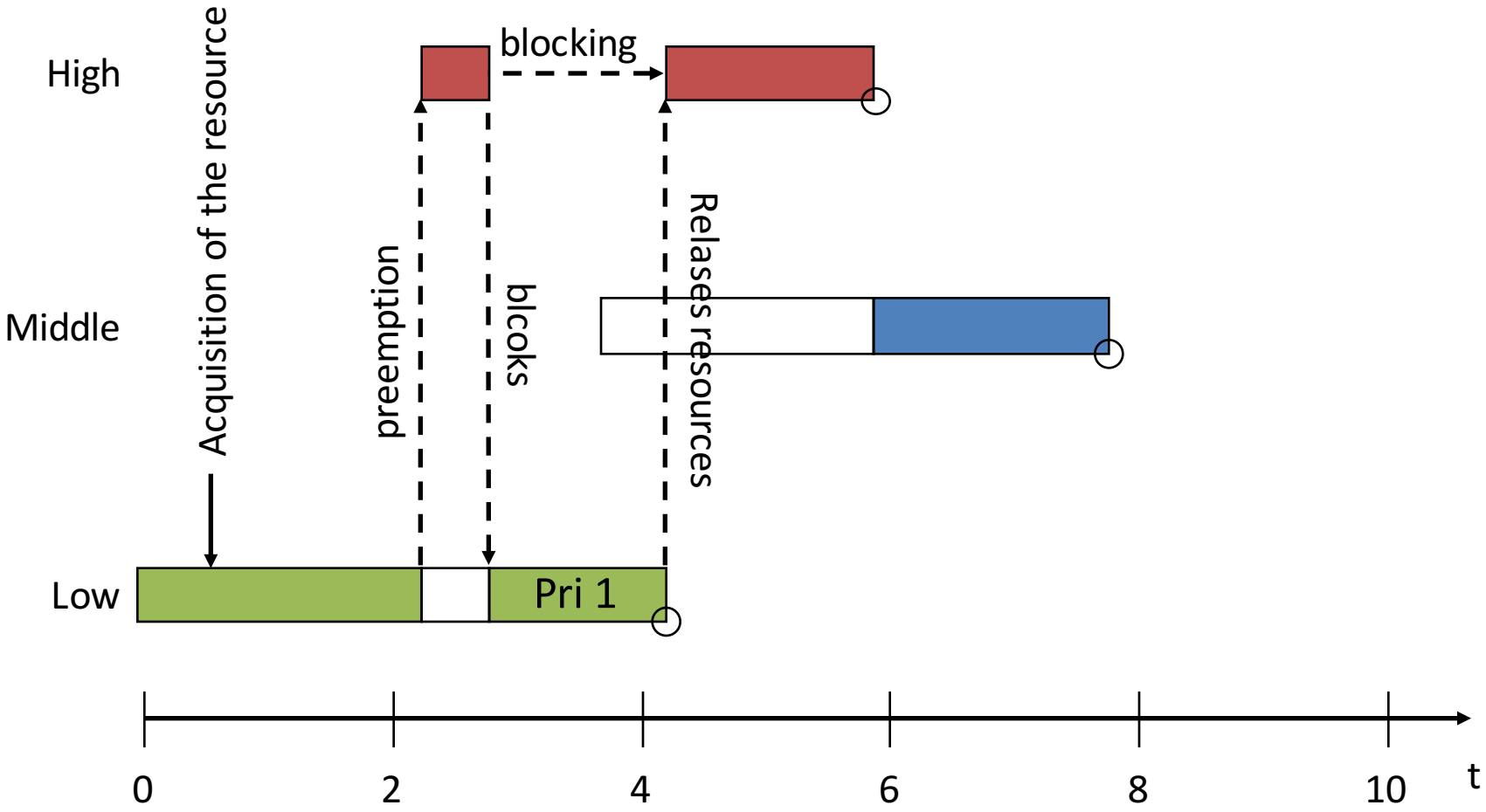


Source: Wikipedia.org

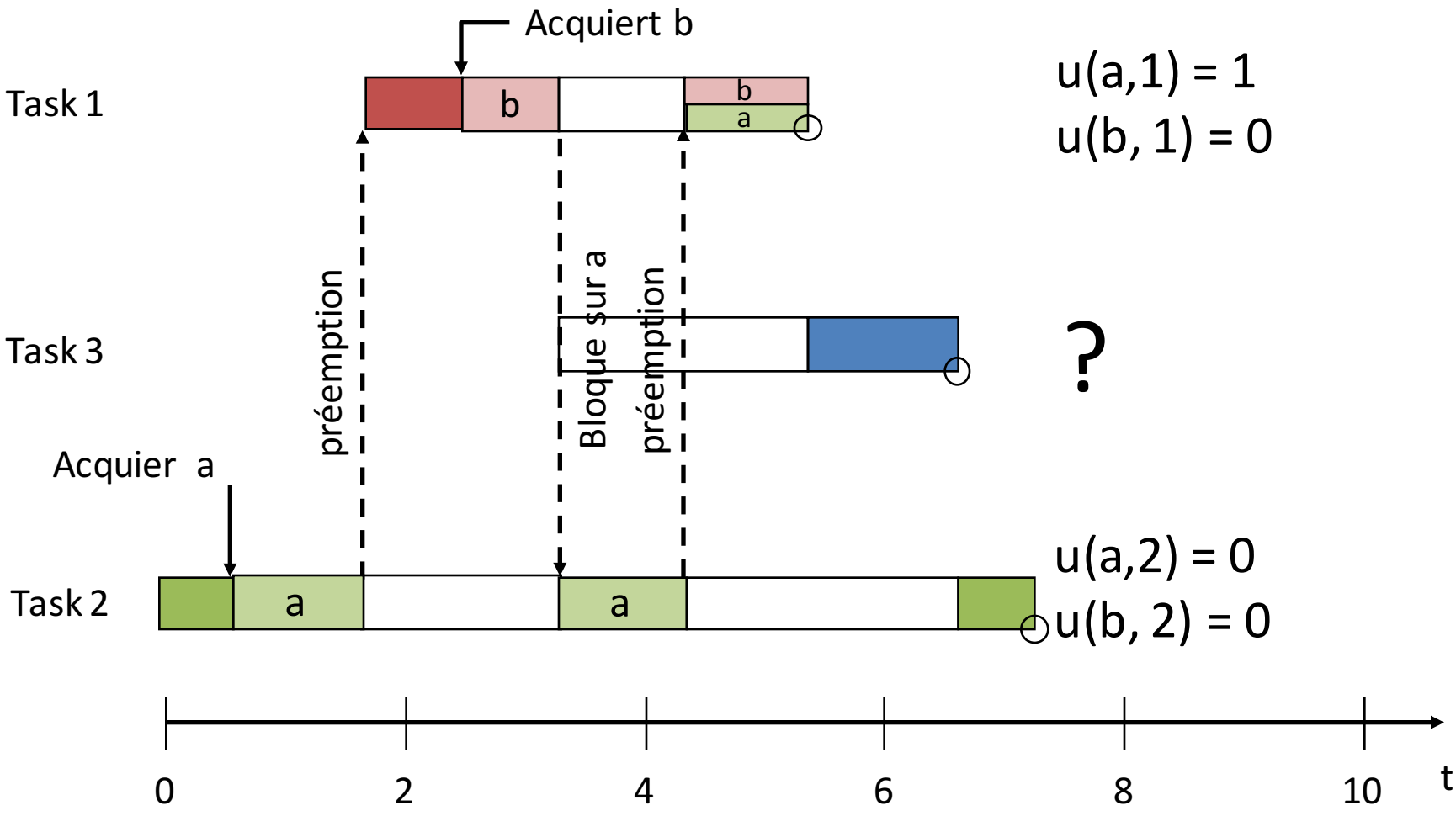
Simple Priority Inheritance

- Is a technique to avoid the priority inversion
  - A task getting a shared resource dynamically gets the priority of the highest priority task with which it shares the resource
  - Inheritance occurs while the higher priority task blocking on the shared resource

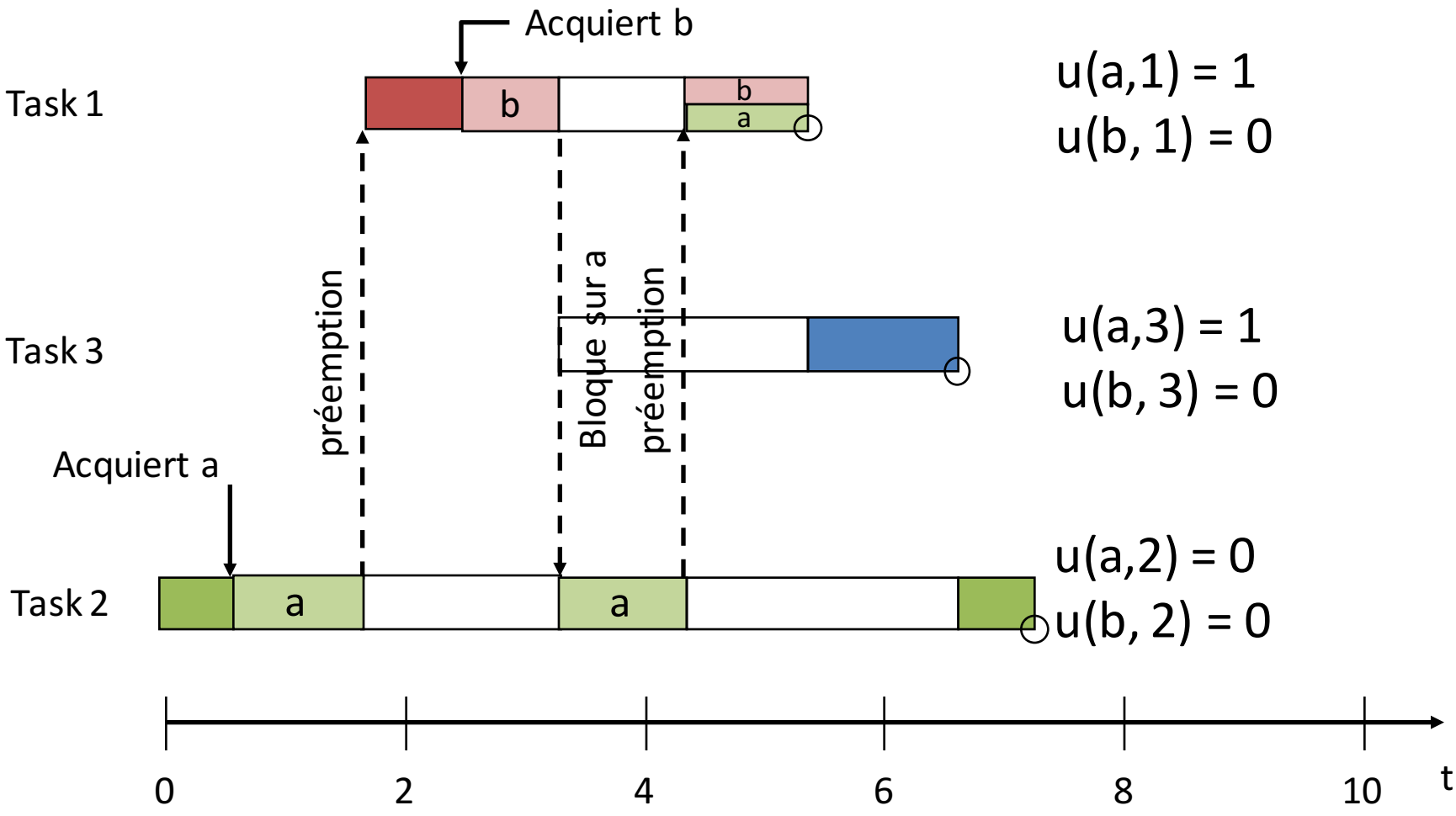
Simple Priority Inheritance



Blocking Time



Blocking Time



Blocking Time

- The blocking time of the task τi is defined as being

$$B_i = \sum_{r=1}^{R} u(r,i) \cdot C_S(r) \qquad (2)$$

  - u (r, i) = 1 if resource r is used by at least one taskpriority of less than i and at least one task (including i) ofhigher priority than iu (r, i) = 0 otherwise
  - and CS (r) is the execution time of the critical section k

Blocking Time

By combining equations (1) and (2) as well as the equation for interference, we obtain:

(3)

$$w_i^{n+1} = e_i + \sum_{j=1}^{k} \left\lceil \frac{w_i^n}{T_j} \right\rceil \cdot e_j + \sum_{r=1}^{R} u(r,i) \cdot C_S(r)$$

- We still have

$$w_i^0 = e_i$$

Example of a simple priority inheritance

Task	e_i	T_i	D_i
1	3	25	7
2	2	12	-
3	5	17	-
4	6	24	-

r	$C_S(r)$	Task Usage
1	2	1, 3, 4
2	4	2, 4

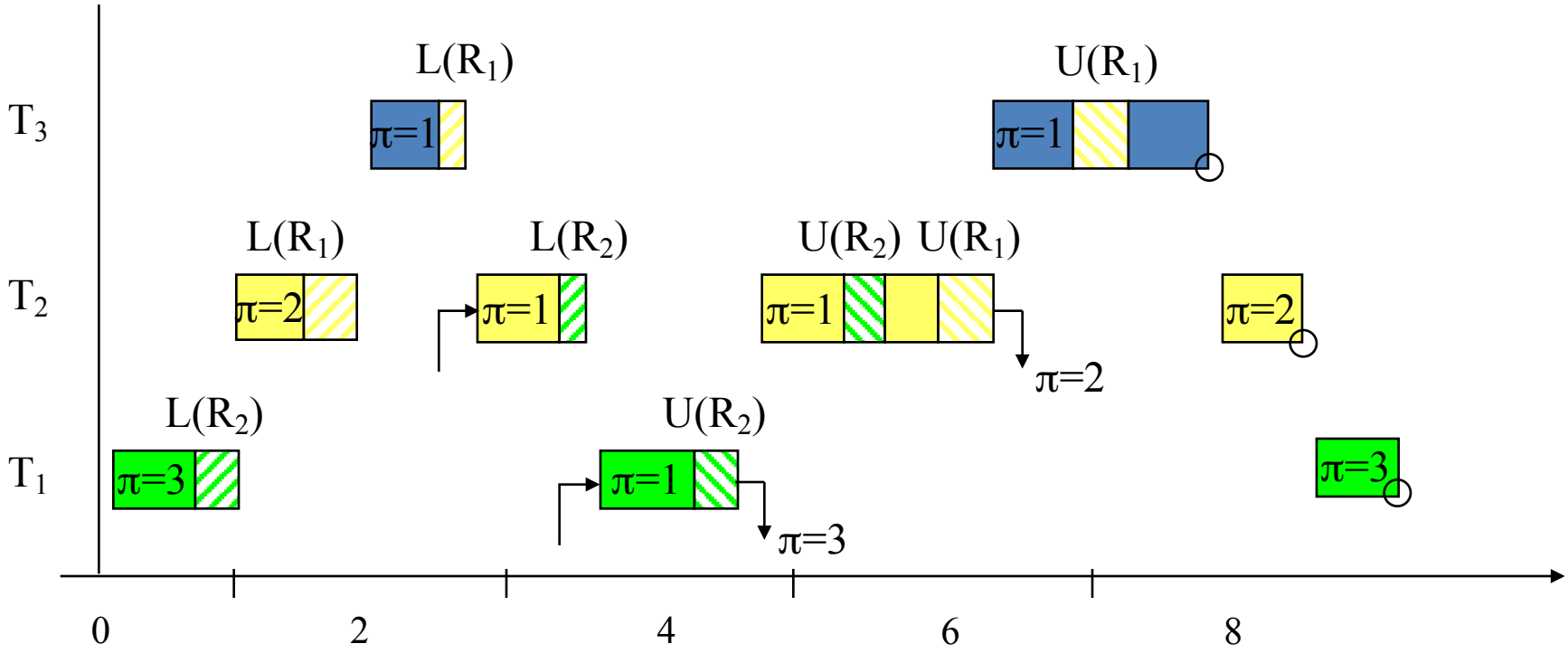
- Use DMPO
- Apply simple priority inheritance
- Is the set of tasks schedulable?

Deadlock

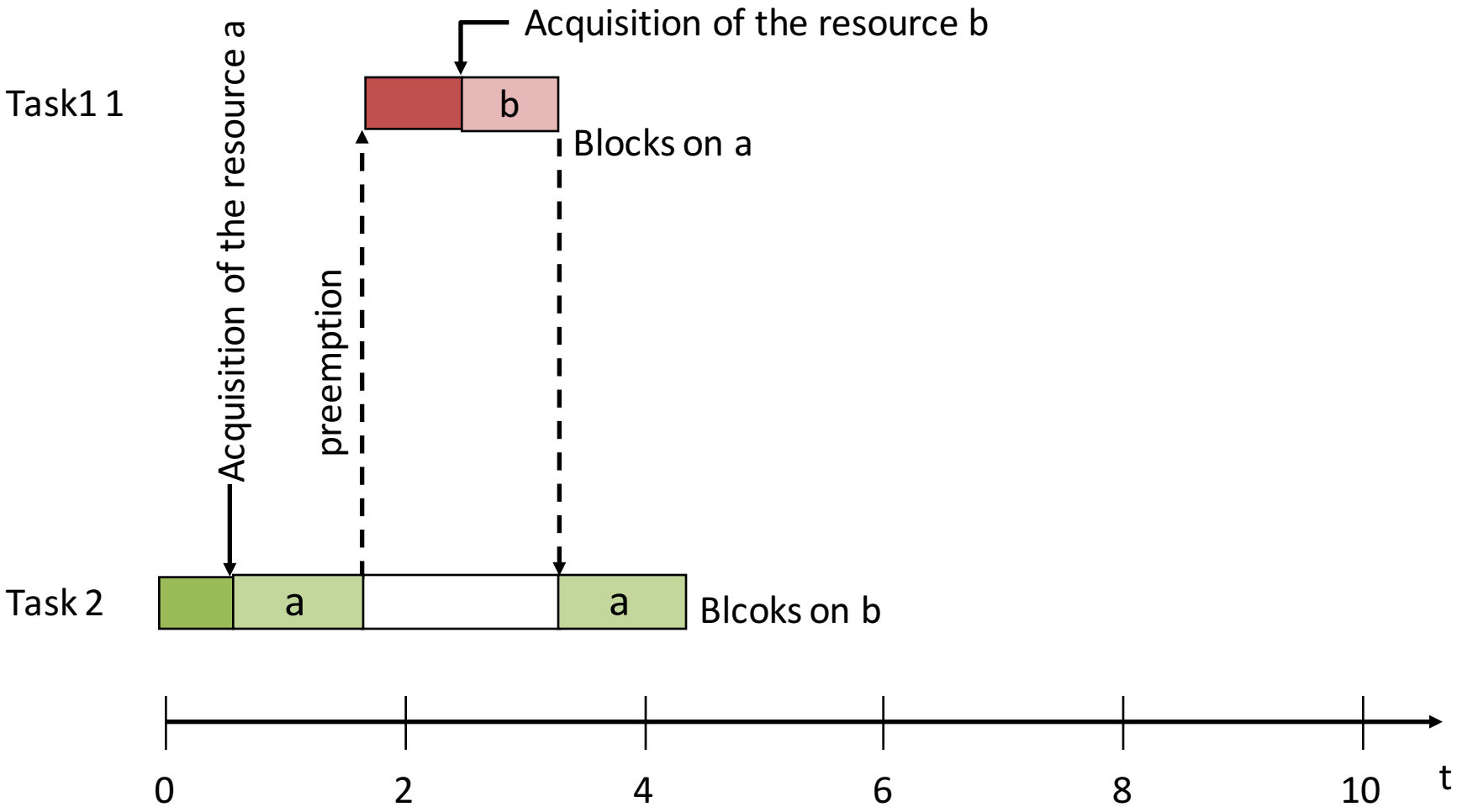
Although using simple priority inheritance will limit the number of deadlocks of a task, it does not prevent transitive blocking or deadlocks.

If the owning task is itself blocked on a second lock, the owner of the second lock can also inherit the priority of the task blocked on the first lock.

Illustrated Transitive Blocking



Deadlock


Acquisition of the resource b
Task1 1
b
Blocks on a
Acquisition of the resource a
preemption
Task 2
a
a
Blcoks on b
0
2
4
6
8
10
t

Ceiling Priority Protocol

- each task has a static default priority
- each resource has a static ceiling value, equal to the maximum priority of the processes which use it
- a task can lock a resource only if its dynamic priority is higher than the ceiling of any currently locked resource (excluding any it already has locked)

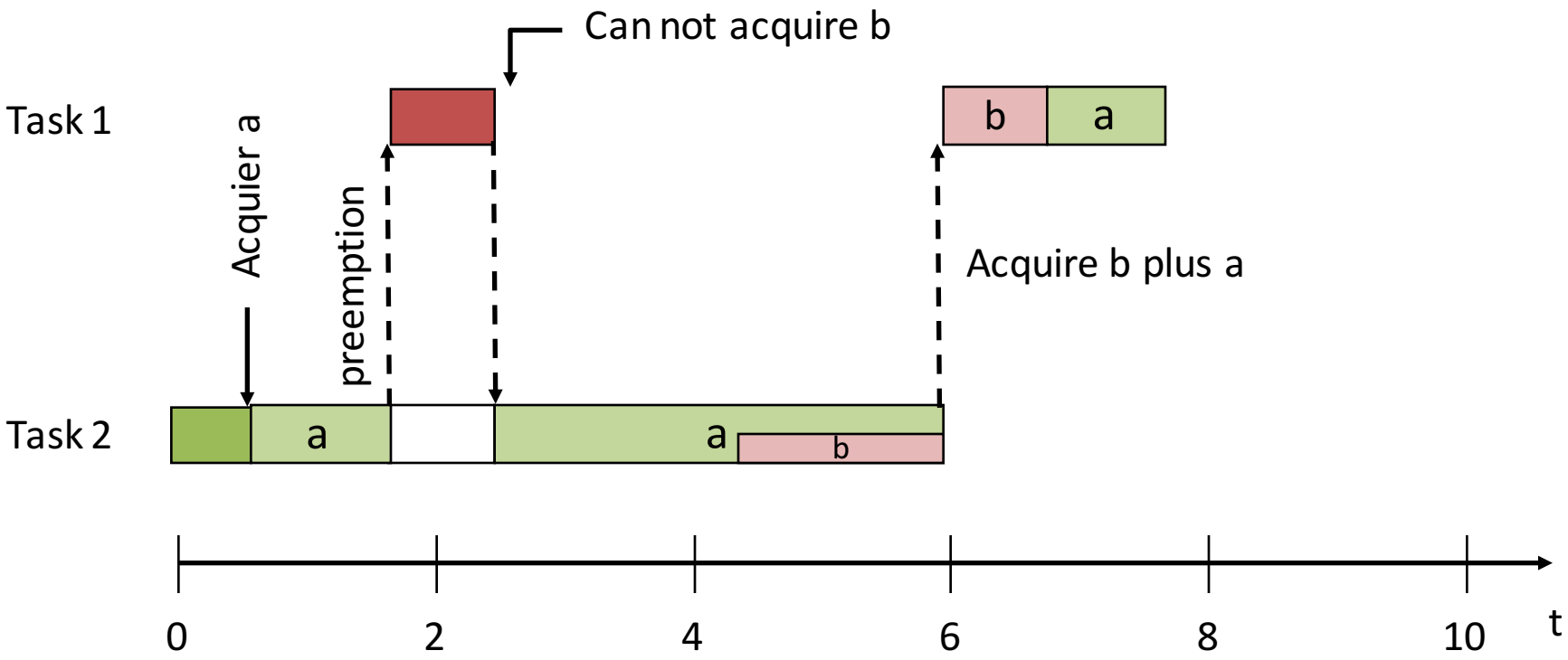
Original:

- each task has a dynamic priority, equal to the maximum of its own static priority and any it inherits **due to blocking** higher priority tasks

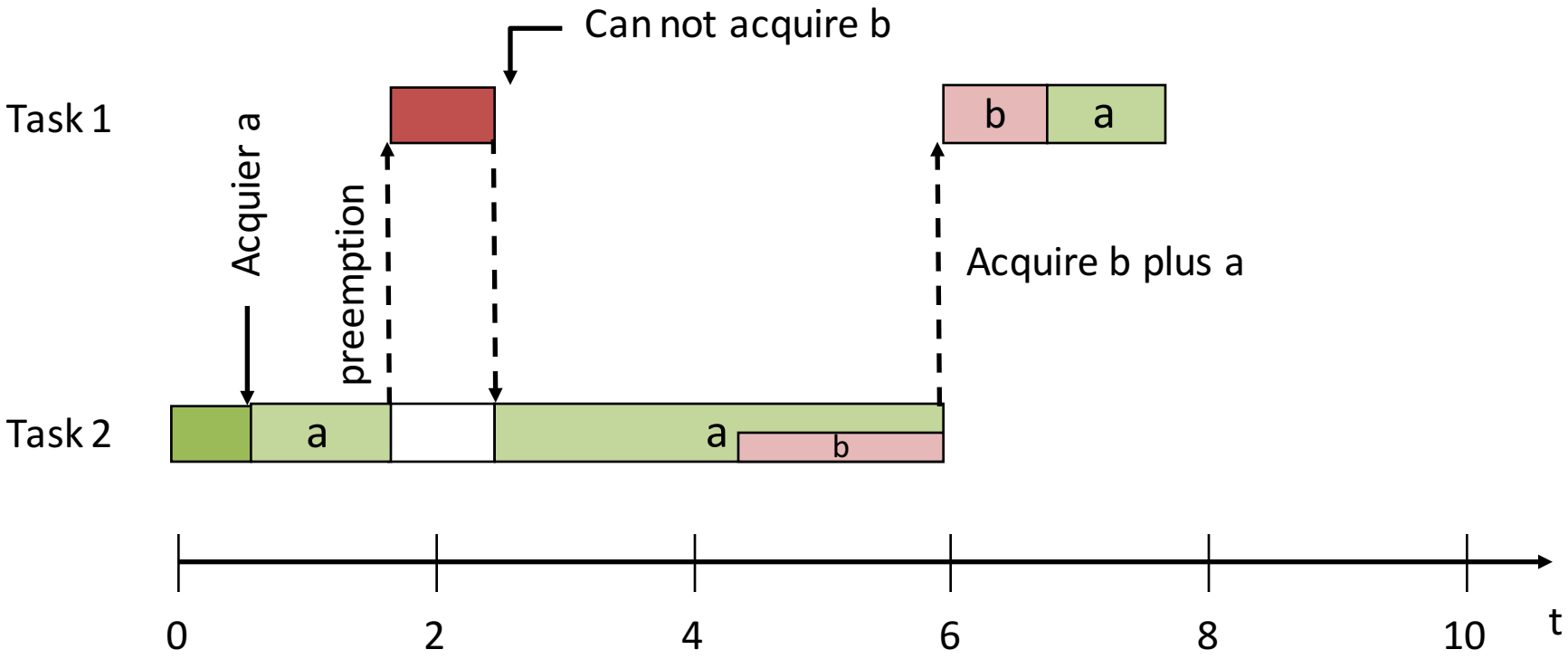
Immediate:

- each task also has a dynamic priority, equal to the maximum of its own static priority and the ceiling values of any resources **it has locked**

Ceiling Priority Protocol



Blocking Time



Ceiling Priority Protocols

- eliminate deadlock : if a low priority task shares 2 resources with a high priority task that can cause dead lock, once it locks one of the resources, it can not get preempted till release the resource, because lower job get equal or higher priority during locking.
- eliminate transitive blocking
  - in fact, for any given task only a single blocking event may occur, therefore
- use a modified blocking time equation:

$$B_i = \max_{k=1}^{K} \{usage(k, i)\ CS(k)\} \qquad (3)$$

Exemple- Ceiling Priority

Task	e_i	T_i	D_i
1	3	25	7
2	2	12	-
3	5	17	-
4	6	24	-

r	$C_S(r)$	Task Usage
1	2	1, 3, 4
2	4	2, 4

- Use DMPO
- Apply Ceiling Priority Protocols
- Is the set of tasks schedulable?

Références

[1] Lee, E. A., Seshia, S. A. “Introduction to Embedded Systems - A Cyber-Physical Systems Approach”, Second Edition, MIT Press, 2017.

[2] Burns, A. and Wellings, A., “*Real-Time Systems and Programming Languages*”, Chapter 13, Addison Wesley, 1997

[3] Gomaa, H., “*Software Design Methods for Concurrent and Real-Time Systems*”, Addison-Wesley, 1993.