Task Cooperation

- Tasks need to cooperate
  - share resources
  - synchronise actions
  - exchange information
- This affects the operation of a schedule
  - schedulability analysis

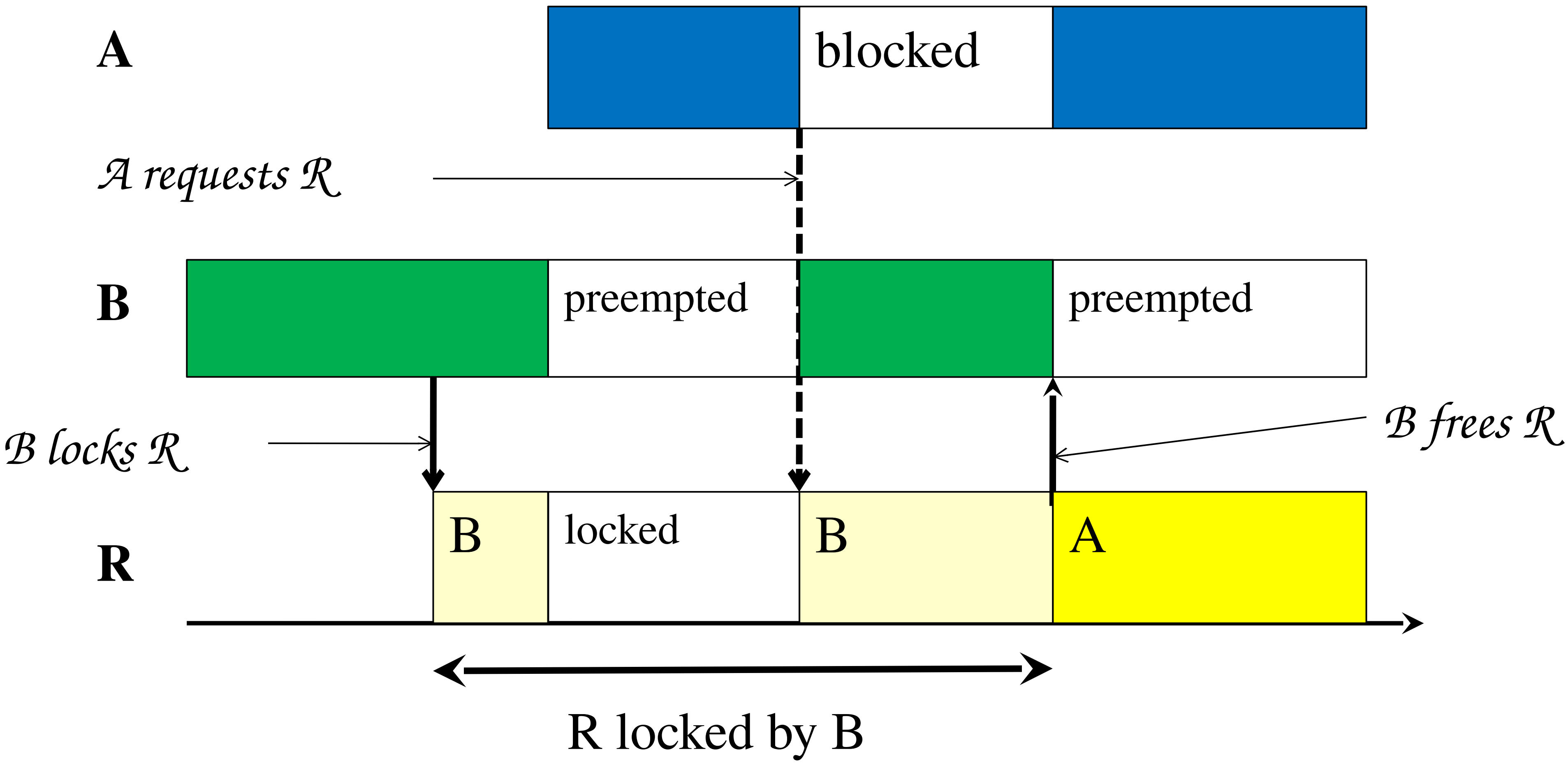
Simple Blocking Example

- Assume two tasks , A and B
- priority(A) > priority(B)
- Both share a non-preemptible resource, R
- B uses R for t units of time
- Assume B is running and has locked R
- A now starts to run and after a short period attempts to access R
- A is blocked until B frees R

We have Blocking

Illustration

priority(A) > priority(B)



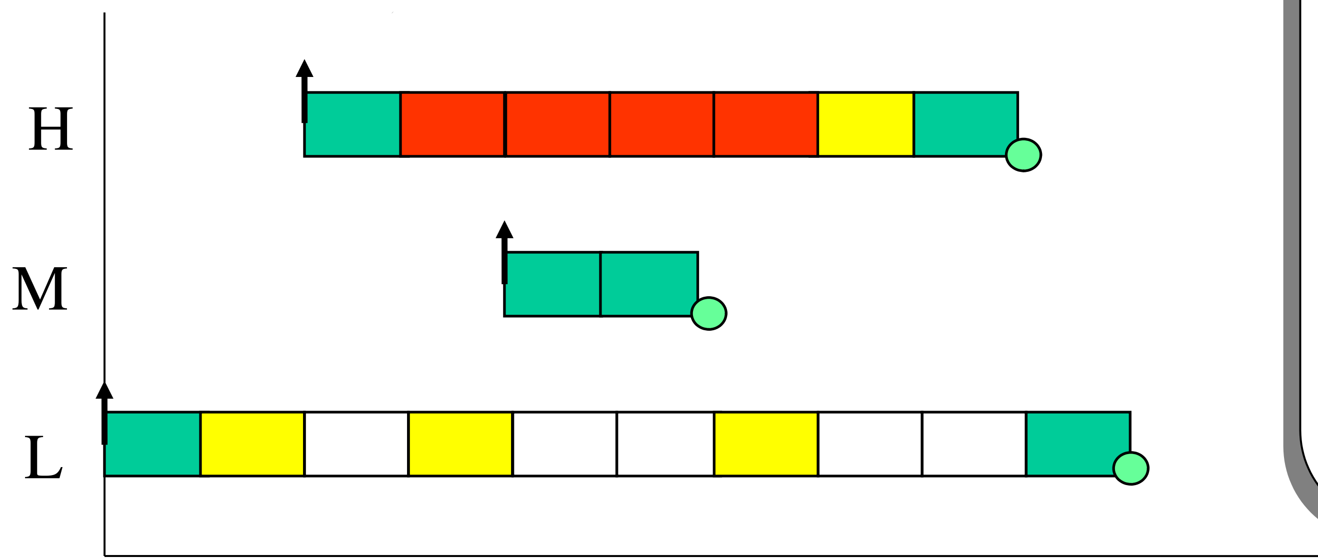
Note

- In following examples we introduce offsets, (release times) so that points can be illustrated.

Example #1

- Example
  - Let H be high priority task, M be a medium priority task, L low priority task
  - H and L share a critical section
  - L is running and locks critical section, R
  - H arrives next, runs and then requests R
  - Request refused since R already locked
  - M (Medium, independent of L) arrives and suspends L
  - …

Example #1



Execution requirements:

H : ERE
M : EE
L : ERRRE

Executing — Executing with R locked — Preempted — Blocked

Example #2

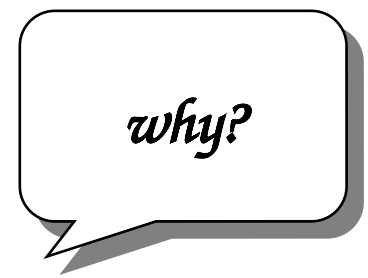
- Scenario:

task	T	D	C
A	50	10	5
B	500	500	250
C	3000	3000	1000

- Utilisation ~93%
- Schedulable if no blocking with worst-case response times of 5, 280, 2500 respectively

Example #2 continued

- Assume A and C share some data
- access to the data is protected by a semaphore, S
- each require the data for 1 unit of time
- Scenario:
  - C is running and at time t locks S
  - immediately after A is activated and gets the cpu
  - at t+2 B is activated but must wait for the cpu
  - at t+3 A attempts to lock S but cannot and is suspended
  - B now runs and completes at t + 253
  - C runs and frees S at t+254
  - A gets in, (at last) BUT has missed its deadline!



Task Interactions and Blocking

- If a task is suspended waiting for a lower-priority task to complete some required computation then the priority model is, in some sense, being undermined
- It is said to suffer priority inversion
- If a task is waiting for a lower-priority task, it is said to be blocked
- Can not predict number of blockings so can have problems with static allocation of priority

Priority Inversion

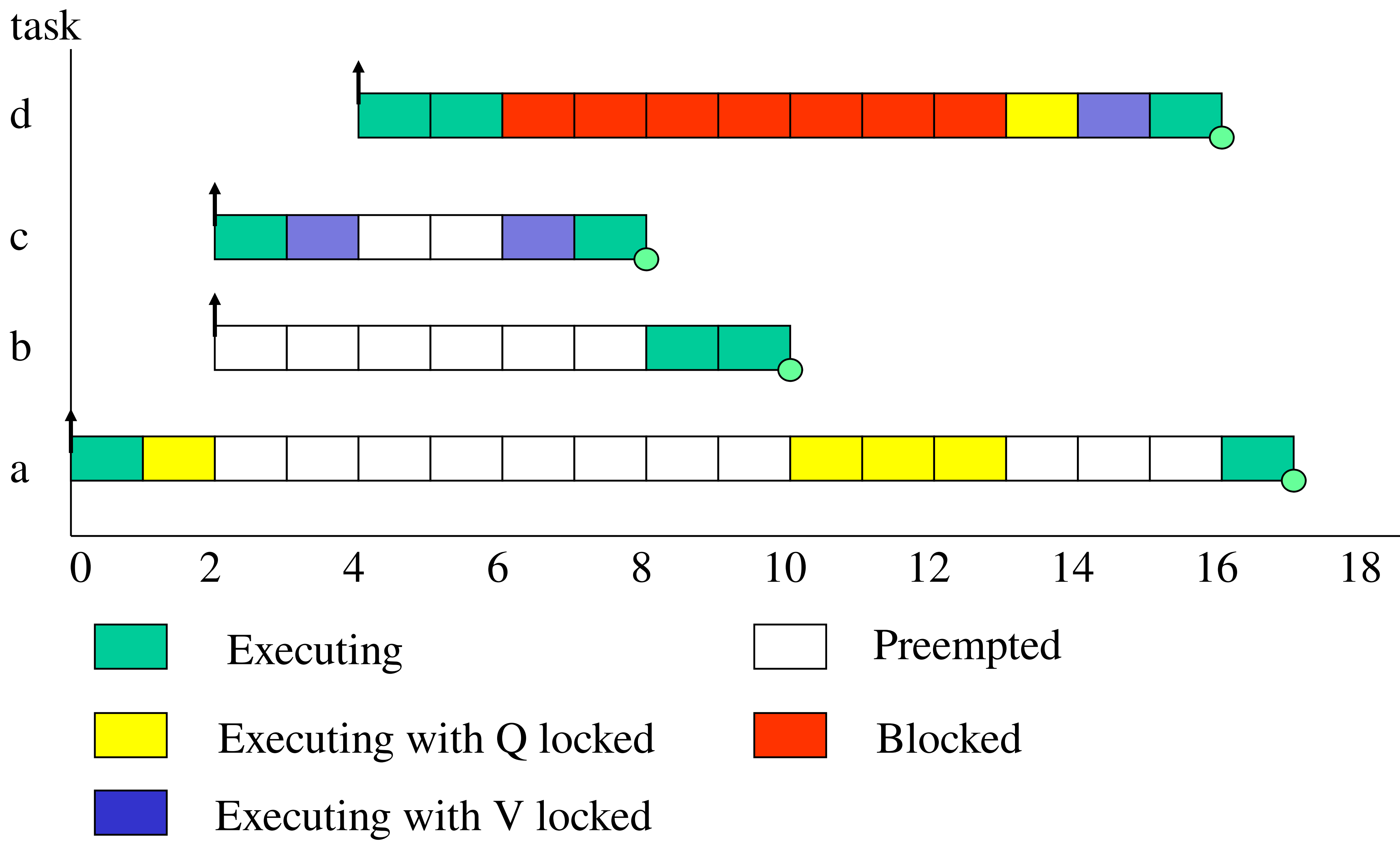
- The situation where a lower priority task can delay a higher priority task
- In example 2 the problem is limited to the effect of B
- In general there may be many other tasks with priorities between A and C so the blocking could be considerable
- The priority scheduling scheme then degenerates to a FIFO algorithm does not take into account importance of tasks

Priority Inversion example #3

- To illustrate an extreme example of priority inversion, consider the executions of four periodic tasks: `a`, `b`, `c` and `d`; and two resources: `Q` and `V`

task	Priority	Execution Sequence	Release Time
a	**1**	**EQQQQE**	**0**
b	**2**	**EE**	**2**
c	**3**	**EVVE**	**2**
d	**4**	**EEQVE**	**4**

Example #3 - Priority Inversion



Possible solutions?

- Prevent pre-emption while a task is inside its critical section
- No task is allowed to enter a critical section if there is a possibility that a higher priority task could be blocked
- Neither satisfactory

Priority Inheritance

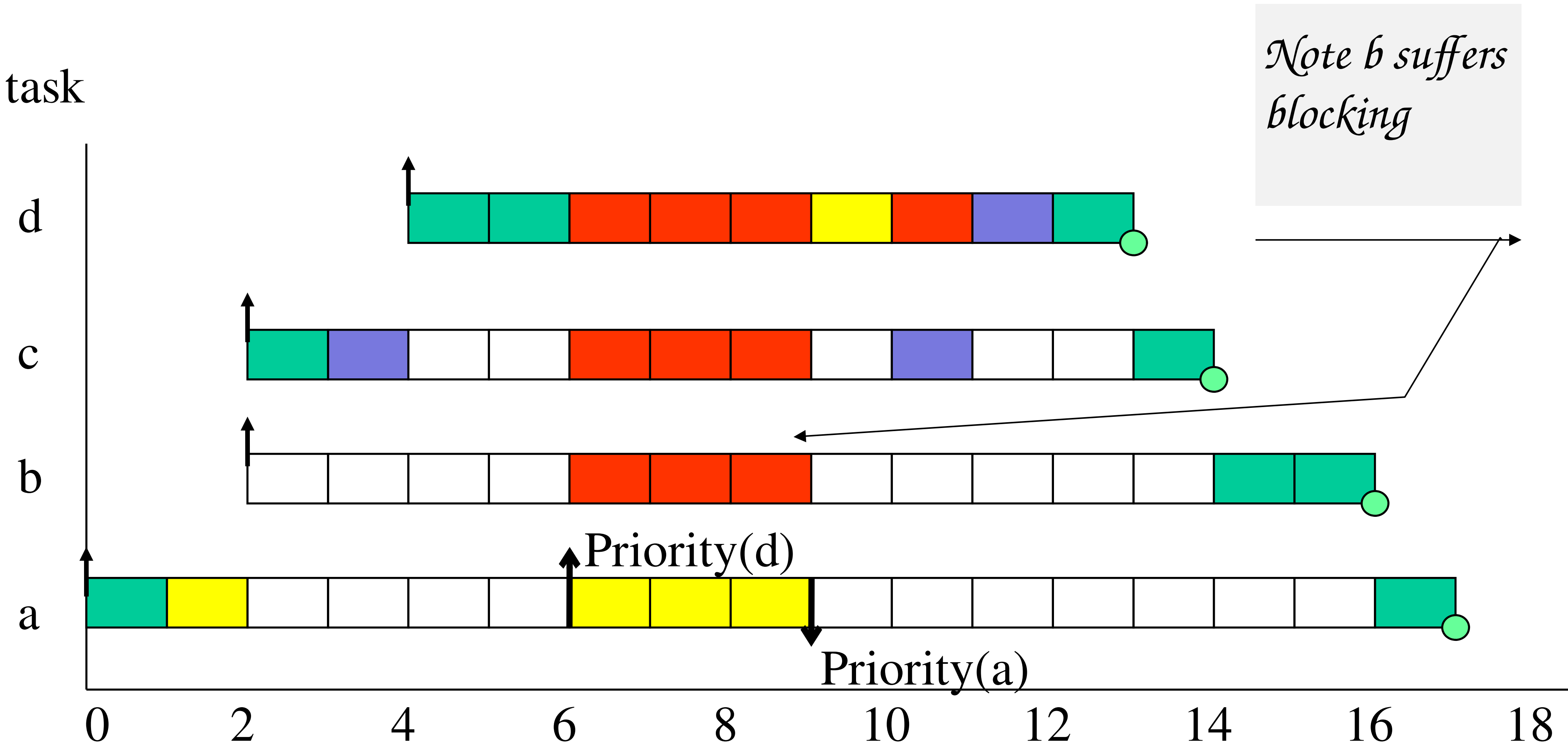
- To prevent a low priority task that is blocking a higher priority task being preempted by tasks of intermediate priority, **priority inheritance protocols** ensure that when ever a task blocks a higher priority task that task **inherits** the higher priority for the duration of the blocking

Priority Inheritance

- Dynamically change priorities
  - priority of L raised to H, once L blocks H
- Limits number of blockings by lower priority tasks
  - If H has **m** critical sections then worst case blocked **m** times
- Problems
  - chains of blocked tasks
  - does not prevent deadlock

Priority Inheritance (ex #3)

- If task p is blocking task q, then q runs with p's priority



Basic Priority Inheritance

- Tasks are assigned a static priority using an appropriate algorithm such as rate or deadline monotonic.
- **Scheduling rule**:
  - ready tasks are scheduled on the processor pre-emptively in a priority driven manner according to their current priority.
  - At release time t, the current priority of every task is equal to its assigned priority.
  - The task remains at that priority except under conditions stated under Priority Inheritance rule.

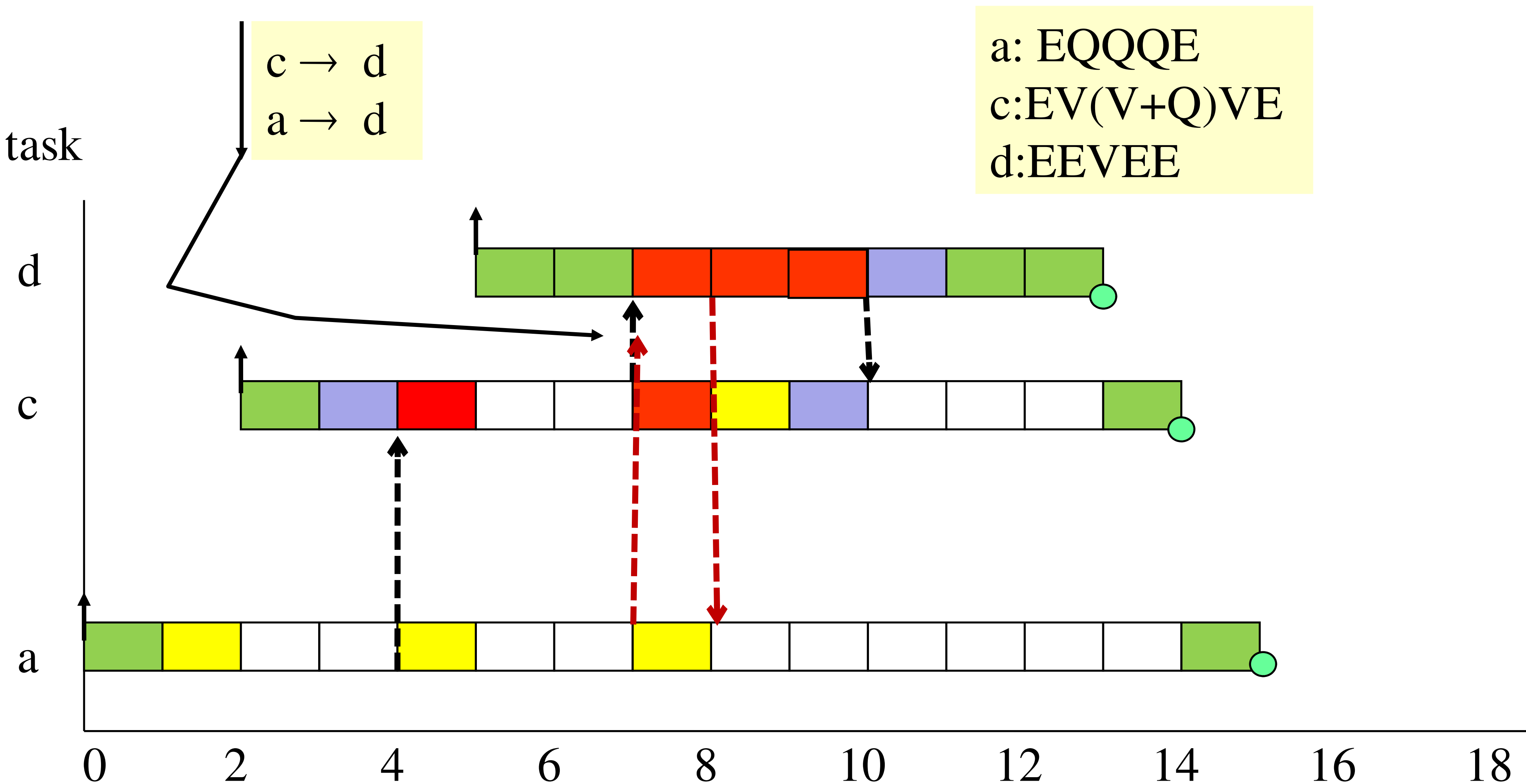
Basic Priority Inheritance

- **Allocation rule**:
  - When a task T requests a resource R at time t,
    - if R is free R is allocated to T until T releases R
    - if R is not free the request is denied and T is blocked.
- **Priority Inheritance rule**:
  - When a requesting task T becomes blocked, the task J which blocks T inherits the current priority of T.
  - The task J executes at its inherited priority until it releases R;
  - At that time the priority of J returns to its priority from the time it blocked T.

Basic Priority Inheritance

- **Weaknesses:**
  - transitive blocking can occur
  - deadlocks are not prevented
  - the blocking time is not reduced to a minimum.

Priority inheritance is transitive



Can have deadlock

- Scenario:
  - H and L share two resources Q and V
  - L locks Q
  - H runs and locks V
  - H tries to lock Q but is blocked
  - L inherits priority(H)
  - L tries to lock V and is refused
  - **We have deadlock**

How could this be avoided?

Deadlock problem

- When using nested critical section, the problem of deadlock can occur; i.e. two or more tasks can be blocked waiting for each other.
- The priority inheritance protocol *does not solve automatically the* problem of deadlock
- Will return to this later.

Computing Blocking Time under Basic Priority Inheritance

- **Theorem 1**
  - Under the priority inheritance protocol, a task can be blocked only once on each different semaphore.
- **Theorem 2**
  - Under the priority inheritance protocol, a task can be blocked by another lower priority task for at most the duration of one critical section.
- This means that we have to consider that a task can be blocked more than once, but only once per each resource and once by each task.

Blocking time computation

- We must build a *resource usage table.:*
  - On each row we put a task (decreasing order of priority);
  - On each column we put a resource, in any order;
  - In each cell (i, j) we put
    - the length of the longest critical section of task i on resource Sj ,
    - or 0 if the task does not use the resource.

Blocking time computation

- A task can be blocked only by lower priority tasks:
  - we must consider only the rows below (tasks with lower priority)
- A task can be blocked only on
  - resources that it uses directly,
  - or used by higher priority tasks (*indirect blocking);*
- For each task, we must consider only those columns on which it can be blocked (used by itself or by higher priority tasks).

Example

	Q	R	S	Blocking
A	2	0	0	?
B	0	1	0	?
C	0	0	2	?
D	3	3	1	?
E	1	2	1	?

- Consider A
  - A can be blocked only on Q.
  - Therefore, we must consider only the first column, and take the maximum, which is 3.
- Therefore, $B_A = 3$.

Example

	Q	R	S	Blocking
A	2	0	0	3
B	0	1	0	?
C	0	0	2	?
D	3	3	1	?
E	1	2	1	?

- B can be blocked on Q (*indirect blocking) and on R*.
  - *Therefore, we* must consider the first 2 columns;
  - Consider all cases where two distinct lower priority tasks between 3, 4 and 5 access Q and R,
  - sum the two contributions, and take the maximum;
  - possibilities are:
    - D on Q and E on R: 3 + 2 = 5;
    - D on R and E on Q: 3 + 1 = 4;
- Therefore, $B_B = 5$.

Example

	Q	R	S	Blocking
A	2	0	0	3
B	0	1	0	5
C	0	0	2	?
D	3	3	1	?
E	1	2	1	?

- C can be blocked on Q, R, S
- Work out possible combinations:
  - D on Q and E on R: 5;
  - D on R and E on Q or S: 4;
  - D on S and E on Q: 2;
  - D on S and E on R : 3;
- So blocking for C is 5

Example: final result

	Q	R	S	Blocking
A	2	0	0	3
B	0	1	0	5
C	0	0	2	5
D	3	3	1	2
E	1	2	1	0

Response Time and Blocking

$$R_i = C_i + B_i + I_i$$

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$R_i^{(n+1)} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil C_j$$

Same task
as before with
an extra term

Maybe pessimistic:
A task may not suffer the
maximum blocking

Problems with Basic Priority Inheritance

- Multiple blockings
  - A task can be blocked more than once on different semaphores
- Multiple inheritance
  - when considering nested resources, the priority can be inherited multiple times
- Deadlock
  - In case of nested resources, there can be a deadlock