KU

THE UNIVERSITY OF KANSAS

Lecture 20: Mutual Exclusion and Scheduling

EECS 388 – Fall 2022

Lecture notes are in parts based on slides created by Prof. Heechul Yun

Context

Recommended Reading - Chapter 12.4 and 11.2.3

Lee and Seshia "Introduction to Embedded Systems"

- Add a bit of complexity
  - How resource sharing affect scheduling?

- Mutual Exclusion
- Race Condition, Critical Section, lock

Race Condition

Initial condition: *counter = 5*

Thread 1

```
R1 = load (counter);
R1 = R1 + 1;
counter = store (R1);
```

Thread 2

```
R2 = load (counter);
R2 = R2 – 1;
counter = store (R2);
```

- What are the possible outcome?

T1 $\overset{5}{R1}$ = load (counter)

$\overset{5}{R2}$ = load (counter)

→ $\overset{6}{R1}$ = R1 + 1

→ $\underset{4}{R2}$ = R2 - 1

6 → count   store (R1)

Race Condition

Initial condition: *counter = 5*

```
R1 = load (counter);
R1 = R1 + 1;
counter = store (R1);
R2 = load (counter);
R2 = R2 – 1;
counter = store (R2);
```

counter = 5

```
R1 = load (counter);
R1 = R1 + 1;
R2 = load (counter);
R2 = R2 – 1;
counter = store (R1);
counter = store (R2);
```

counter = 4

```
R1 = load (counter);
R1 = R1 + 1;
R2 = load (counter);
R2 = R2 – 1;
counter = store (R2);
counter = store (R1);
```

counter = 6

- Why this happens?

Race Condition

- A situation when two or more threads **read and write** shared data at the same time
- Correctness depends on the execution order

Thread 1

```
R1 = load (counter);
R1 = R1 + 1;
counter = store (R1);
```

Thread 2

```
R2 = load (counter);
R2 = R2 – 1;
counter = store (R2);
```

- How to prevent race conditions?

Critical Section

- Code sections of potential race conditions

Thread 1

```
Do something
..
R1 = load (counter);
R1 = R1 + 1;
counter = store (R1);
...
Do something
```

Thread 2

```
Do something
..
R2 = load (counter);
R2 = R2 – 1;
counter = store (R2);
..
Do something
```

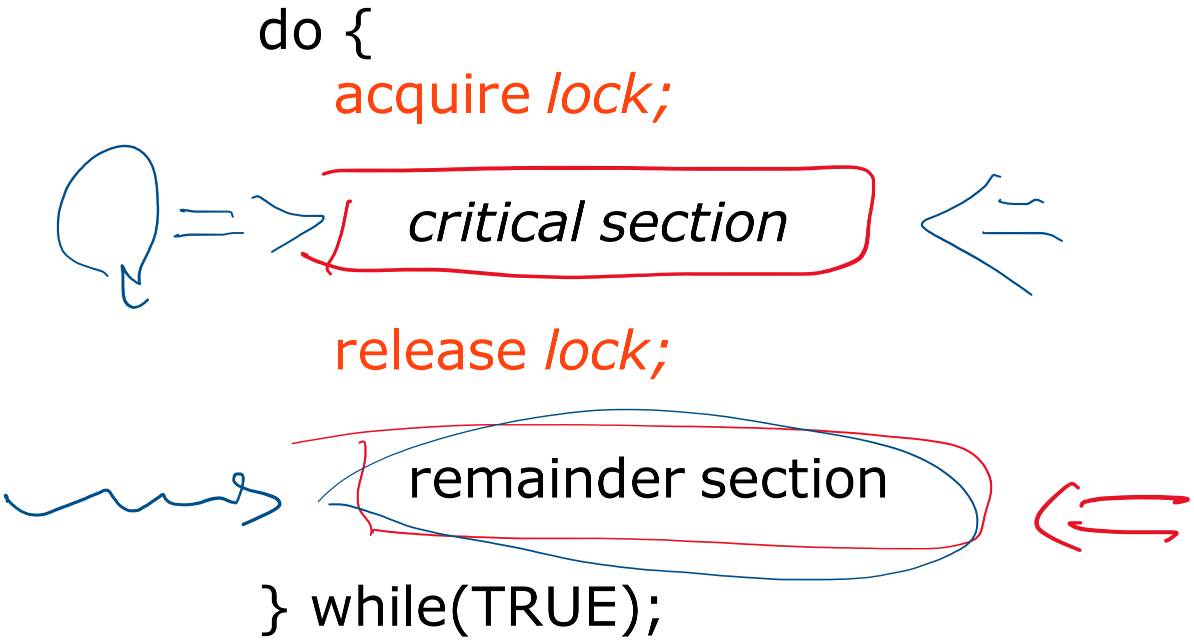
Critical sections

Mutual Exclusion

- A property that requires only one thread can enter its critical section at a time among multiple concurrent threads
- Lock (mutex) is a mechanism to provide mutual exclusion

Lock

- General solution
  - Protect critical section via a lock
  - Acquire on enter, release on exit



```
do {
    acquire lock;

        critical section

    release lock;

        remainder section

} while(TRUE);
```

Scheduling and Mutual Exclusion

What happens when tasks share resources and use mutual exclusion to guard access to those resources?

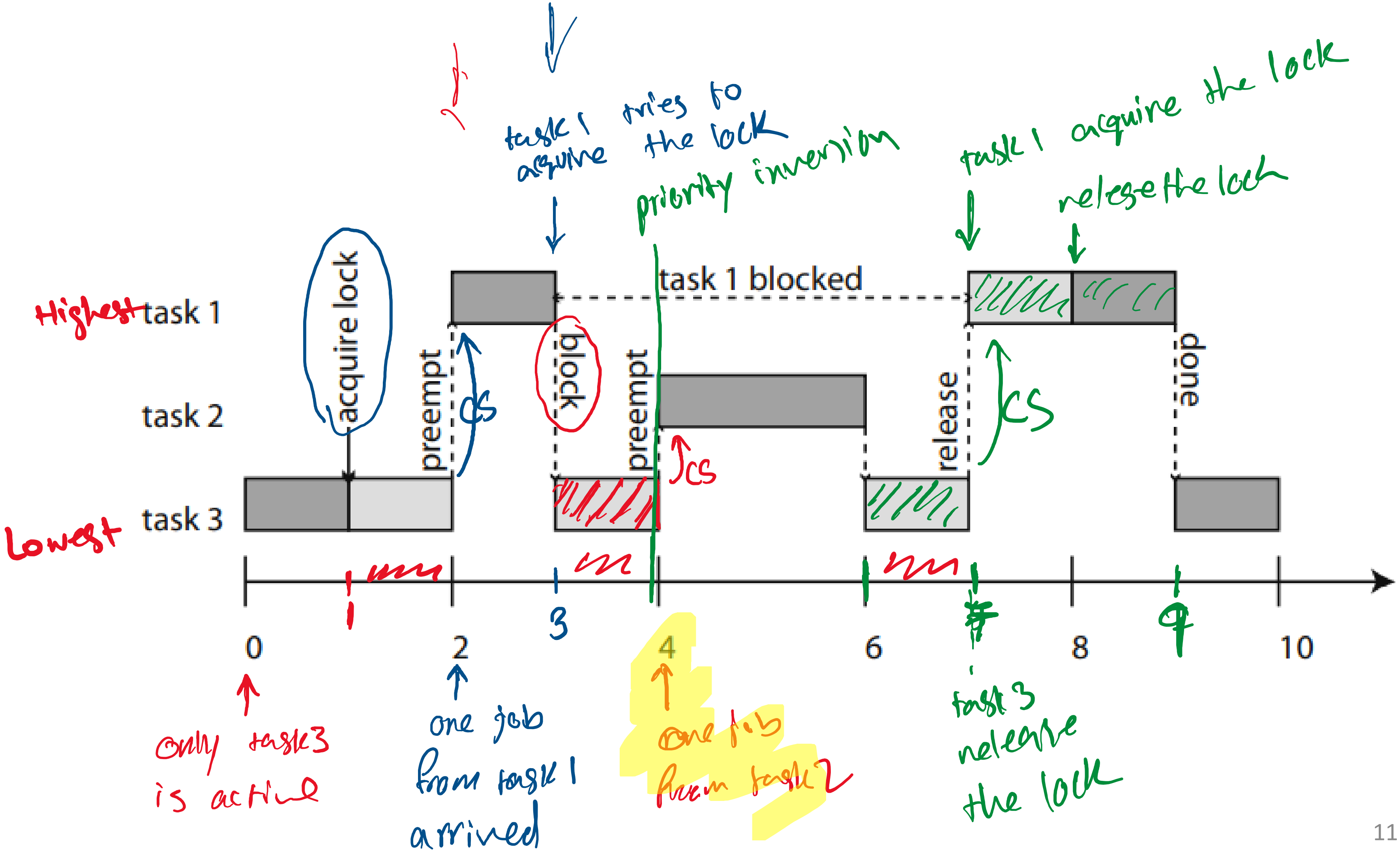
Priority Inversion

- Priority based schedulers always run the task with higher priority

- What if a higher priority task is blocked by a lower priority task?

Priority Inversion: a scheduling anomaly where a high-priority task is blocked while unrelated lower-priority tasks are executing

Illustration of Priority Inversion

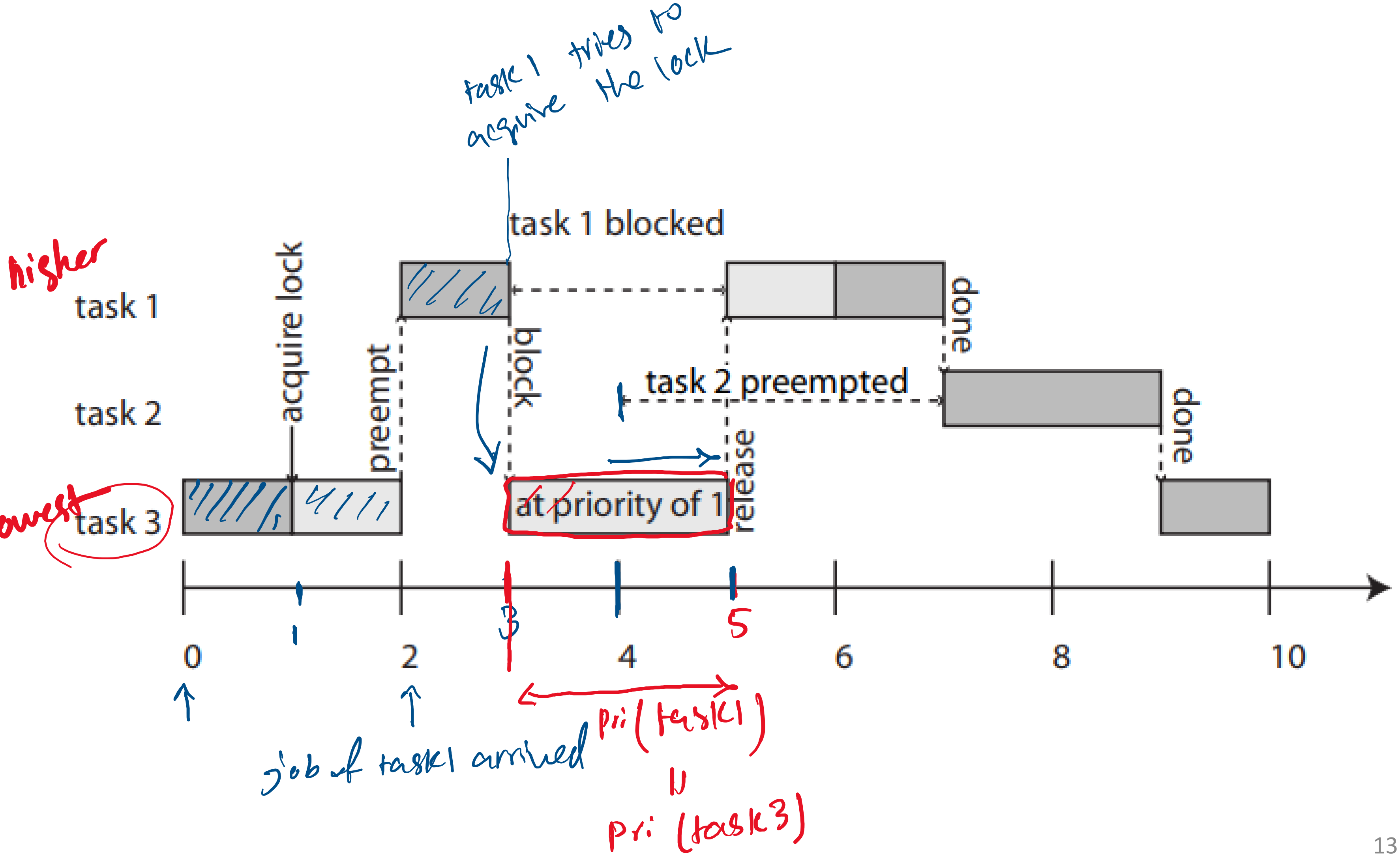


Priority Inheritance Protocol

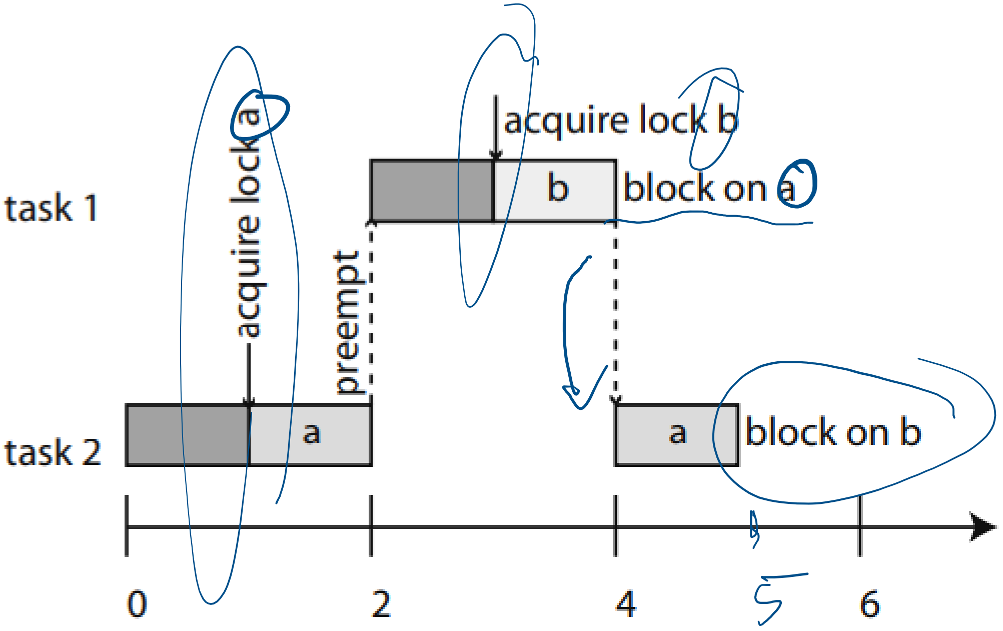
- The task that holds the locks inherits the priority of the task that tries to acquire the lock

- Implication: prevents priority inversion
  - The task that holds the lock cannot be preempted by an intermediate priority task

Illustration of Priority Inheritances



Deadlock!


acquire lock a
preempt
acquire lock b
task 1
b
block on a
task 2
a
a
block on b
0
2
4
5
6

Priority Ceiling Protocol

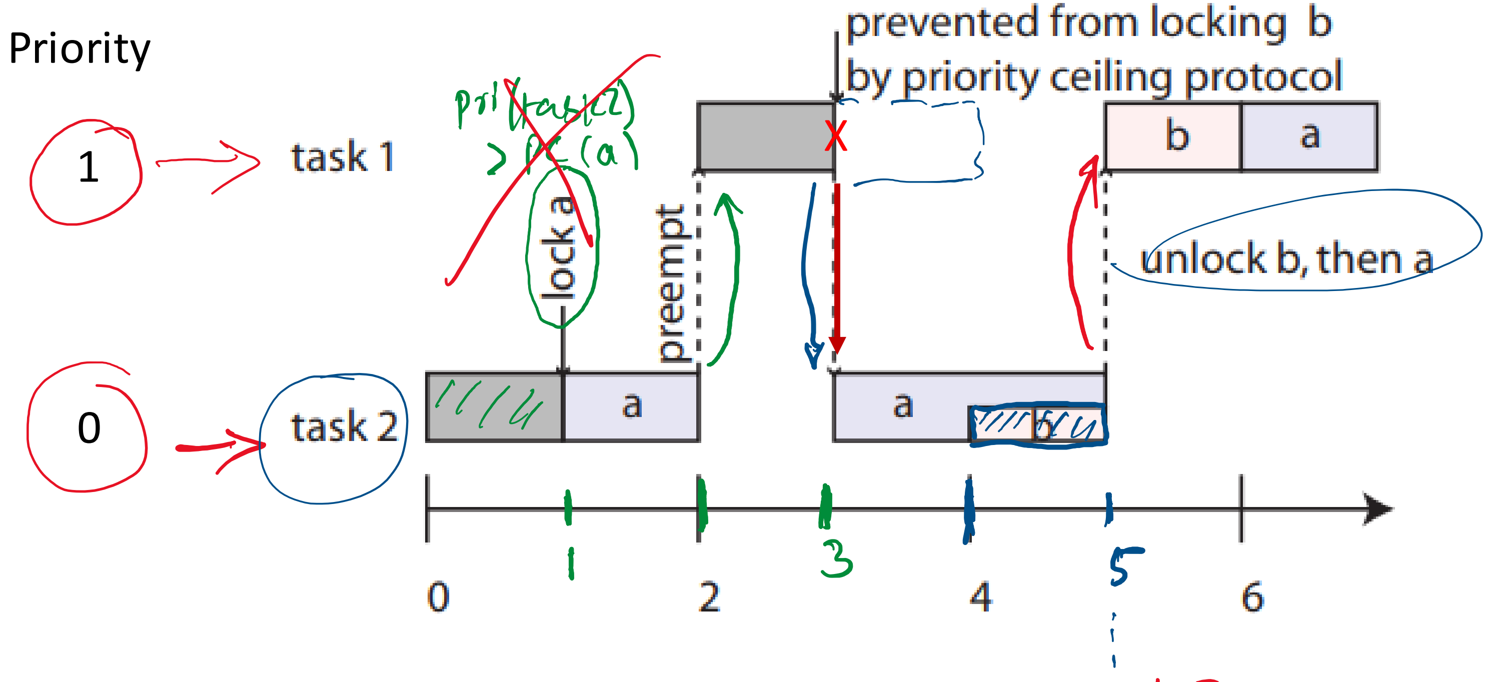
- A task can acquire a lock only if the task's priority is strictly higher than the priority ceilings of all locks currently held by other tasks.
- PC(R) = highest priority of all tasks that may lock R
- Process P can start a new critical section only if: Task's priority > PC of all resources locked by other processes
- Assumption: we know the number of tasks and priorities

which task uses which lock

Illustration of Priority Ceiling Protocol

?

Pri(task1) <= PC(lock a)



task 2
unlock a
unlock b

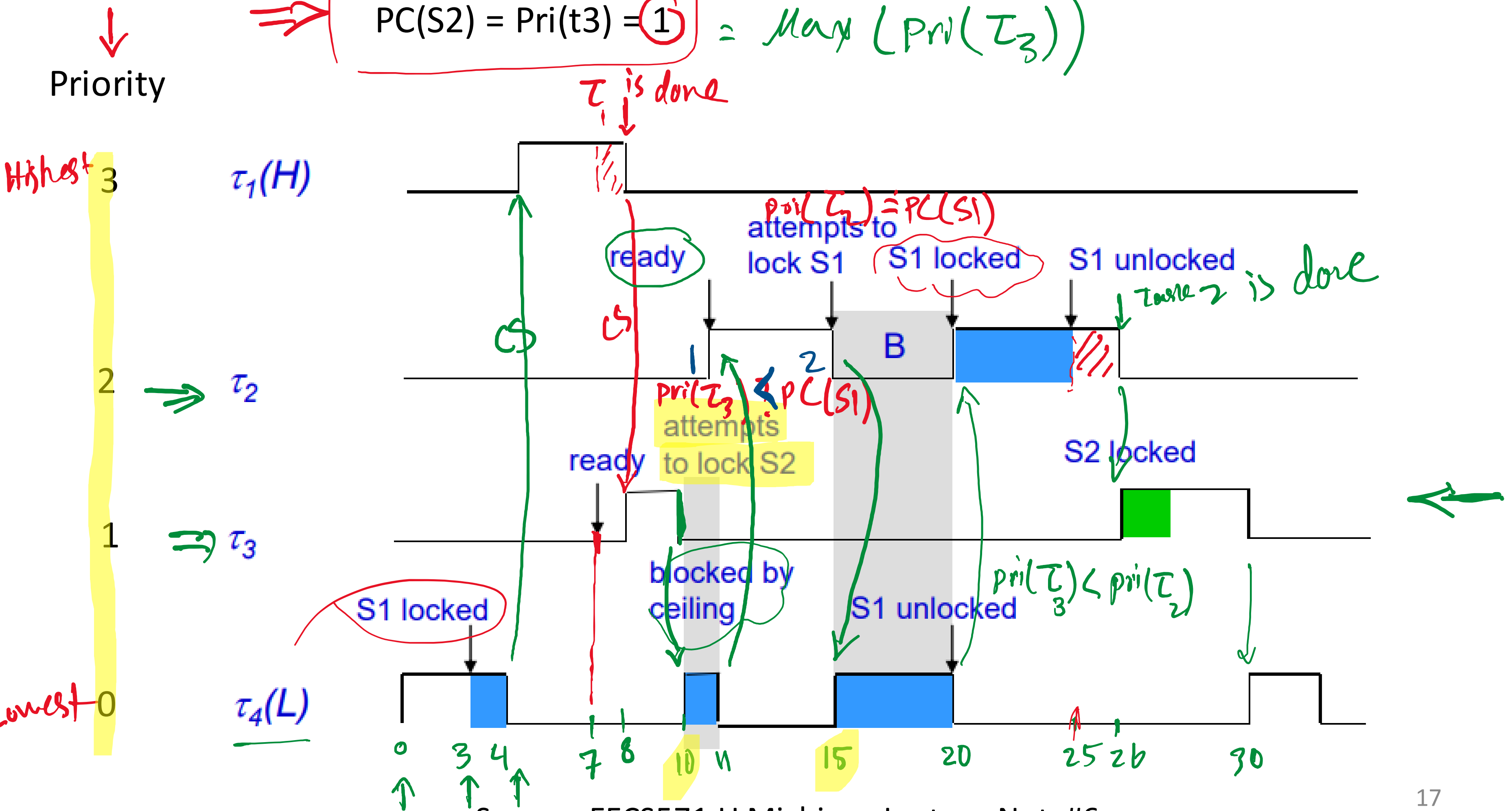
PC(lock a) = Pri(task1) = 1 = Max(0, 1)

PC(lock b) = Pri(task1) = 1 = Max(0, 1)

Another Example

PC(S1) = Pri(t2) = 2 $= \mathrm{Max}(\overset{2}{\mathrm{pri}(\tau_2)}, \overset{0}{\mathrm{pri}(\tau_4)}) = 2$

PC(S2) = Pri(t3) = 1 $= \mathrm{Max}(\mathrm{Pri}(\tau_3))$



Source: EECS571 U Michigan Lecture Note#6

References

- Lee and Seshia. *Intro to Embedded Systems A Cyber Physical System Approach*., 2017. (Chapter 12)