

Lecture 13:

Real-Time Scheduling

CS 3281

Daniel Balasubramanian,
Shervin Hajiamini, Sandeep Neema, and Bryan Ward

Motivation – Cyber-Physical Systems



Surgical Robotics



Industrial Internet of Things (IIoT)



Power and Utilities



Satellites



Autonomous Vehicles



Drones & DoD Systems

Real-Time Systems

Enterprise Systems



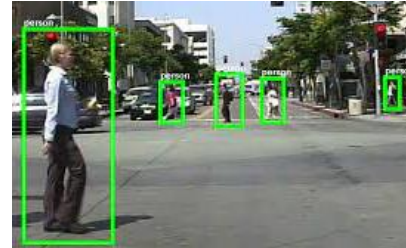
Servers, desktops, web browsing, emails, etc.

"Real Fast" Systems



Interactive processing, i.e., video games

Soft Real-Time System



Pedestrian Detection

Hard Real-Time System



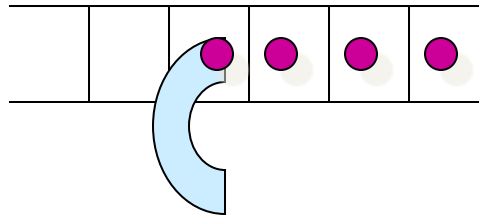
Arc-Flash Relays: ~2ms to break circuit



Interaction with the physical world requires keeping time with the real world.
Many CPS, especially safety- and mission-critical systems have strict timing requirements.

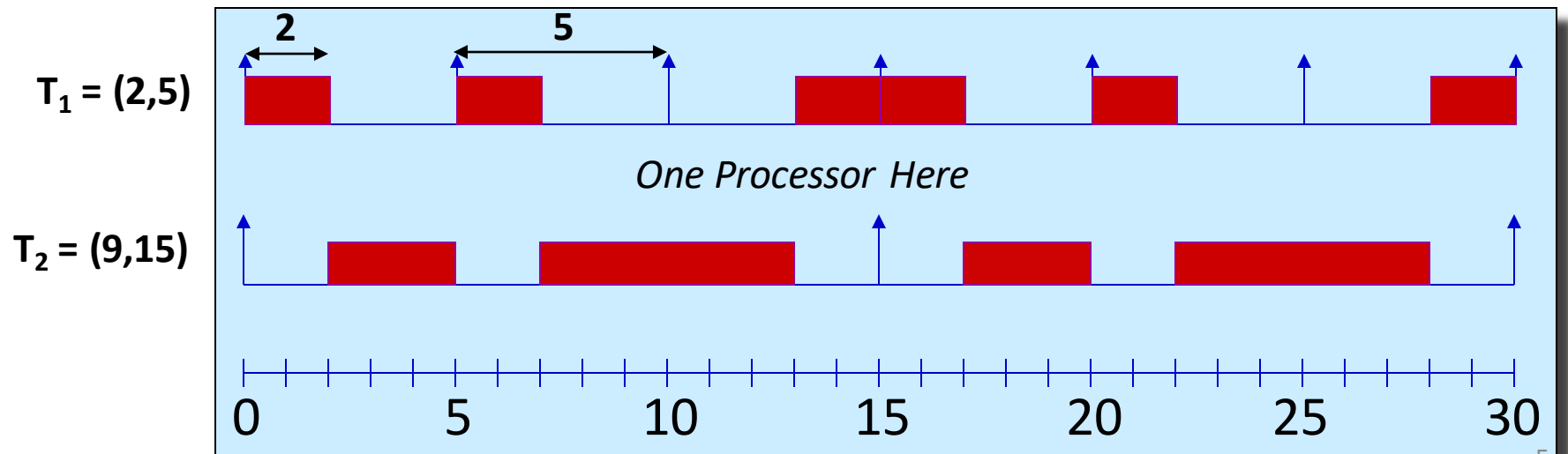
What is a Real-Time System?

- A system with a dual notion of correctness:
 - *Logical correctness* (“it does the right thing”);
 - *Temporal correctness* (“it does it on time”).
- A system wherein *predictability* is as important as *performance*.
- Real-time systems are designed based on worst case, rather than average case
- **A simple example:** A robot arm picking up objects from a conveyor belt.



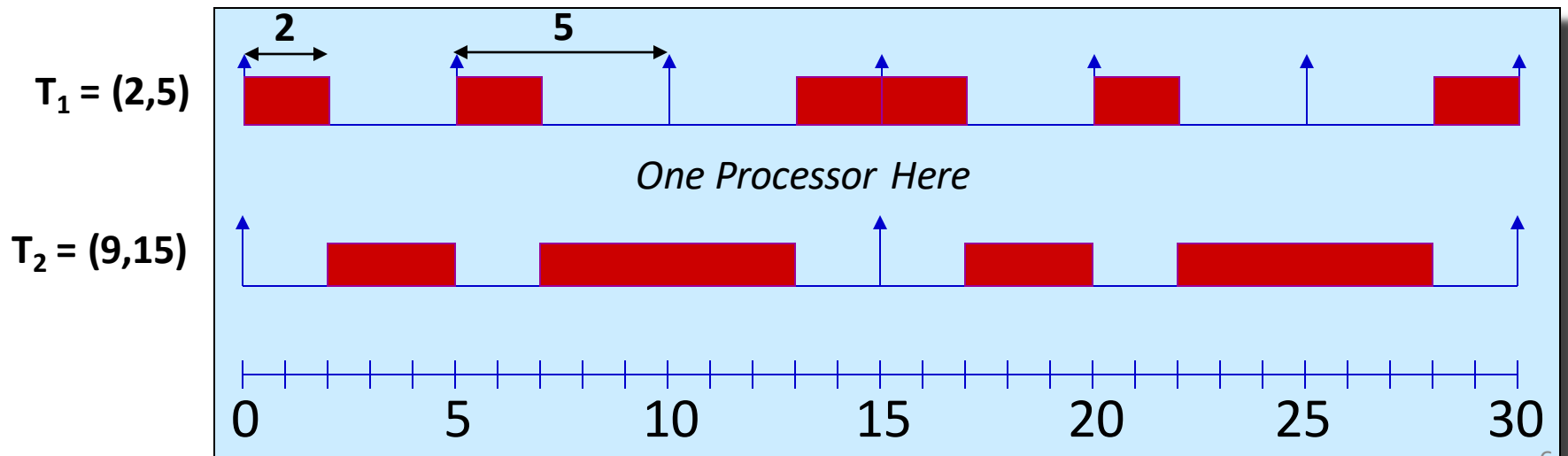
Periodic Task Systems

- Set τ of periodic tasks scheduled on M cores:



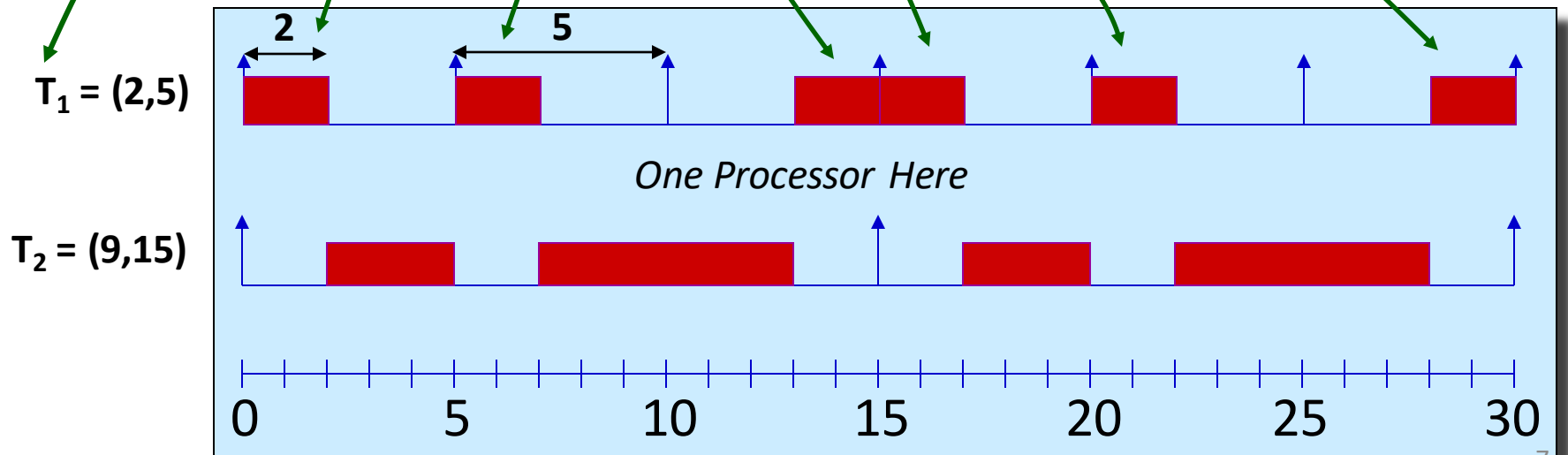
Periodic Task Systems

- Set τ of **periodic tasks** scheduled on **M cores**:
 - Task $T_i = (e_i, p_i)$ releases a **job** with exec. cost e_i every p_i time units.
 - T_i 's **utilization** (or **weight**) is $u_i = e_i/p_i$.
 - Total utilization** is $U(\tau) = \sum_{T_i} e_i/p_i$.



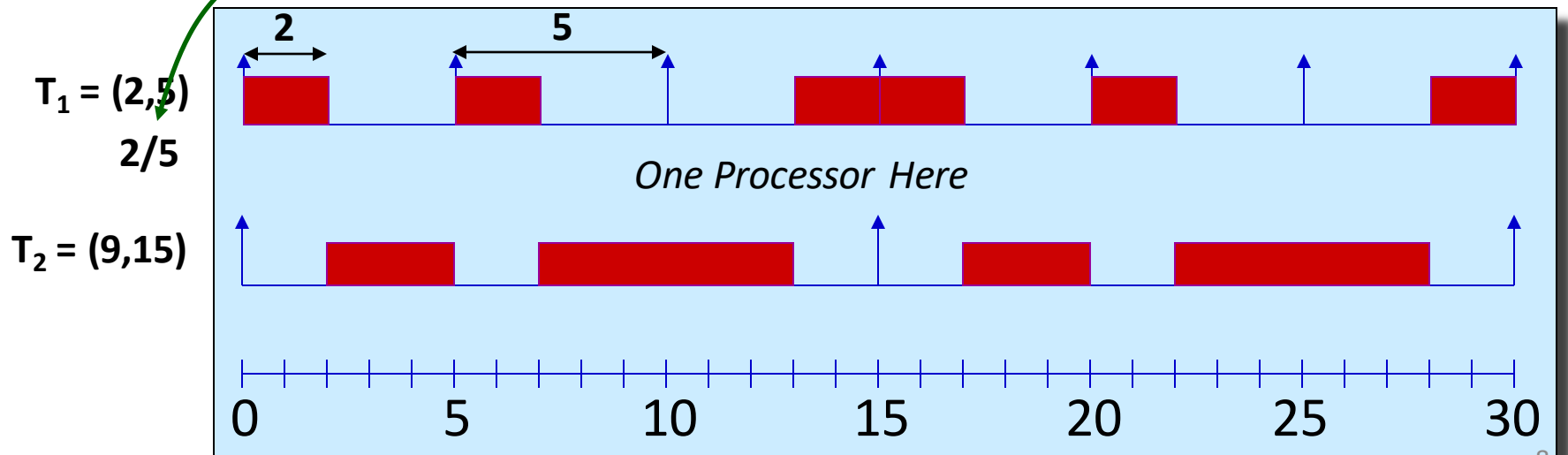
Periodic Task Systems

- Set τ of **periodic tasks** scheduled on **M cores**:
 - Task $T_i = (e_i, p_i)$ releases a **job** with exec. cost e_i every p_i time units.
 - T_i 's **utilization** (or **weight**) is $u_i = e_i/p_i$
 - Total utilization** is $U(\tau) = \sum_T e_i/p_i$.



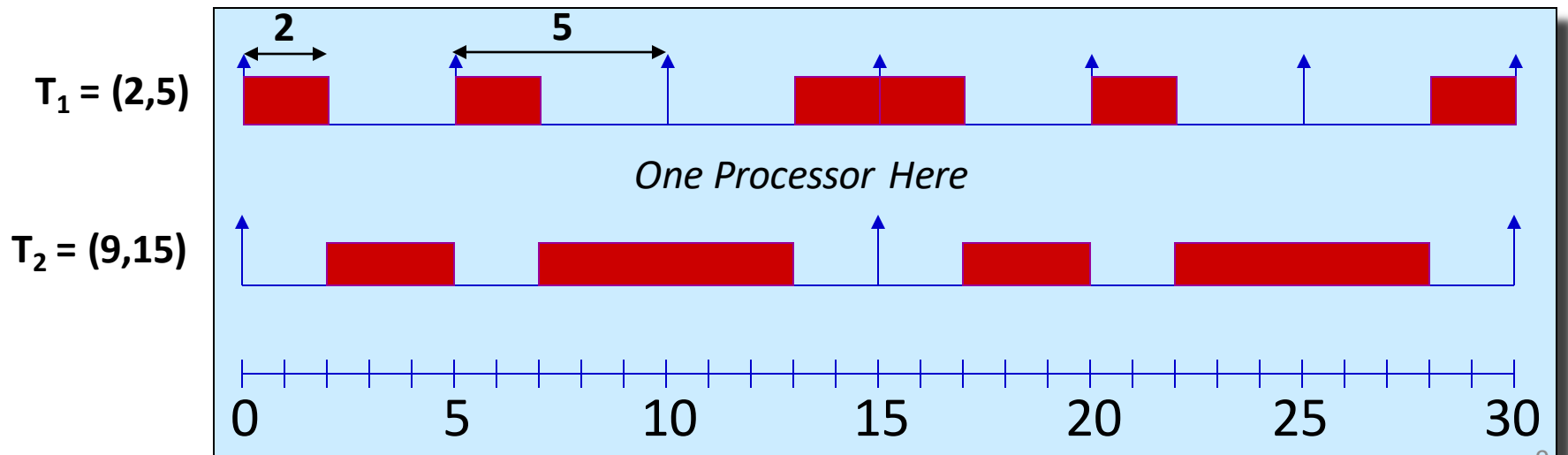
Periodic Task Systems

- Set τ of **periodic tasks** scheduled on **M cores**:
 - Task $T_i = (e_i, p_i)$ releases a **job** with exec. cost e_i every p_i time units.
 - T_i 's **utilization** (or **weight**) is $u_i = e_i/p_i$.
 - Total utilization** is $U(\tau) = \sum_{T_i} e_i/p_i$.



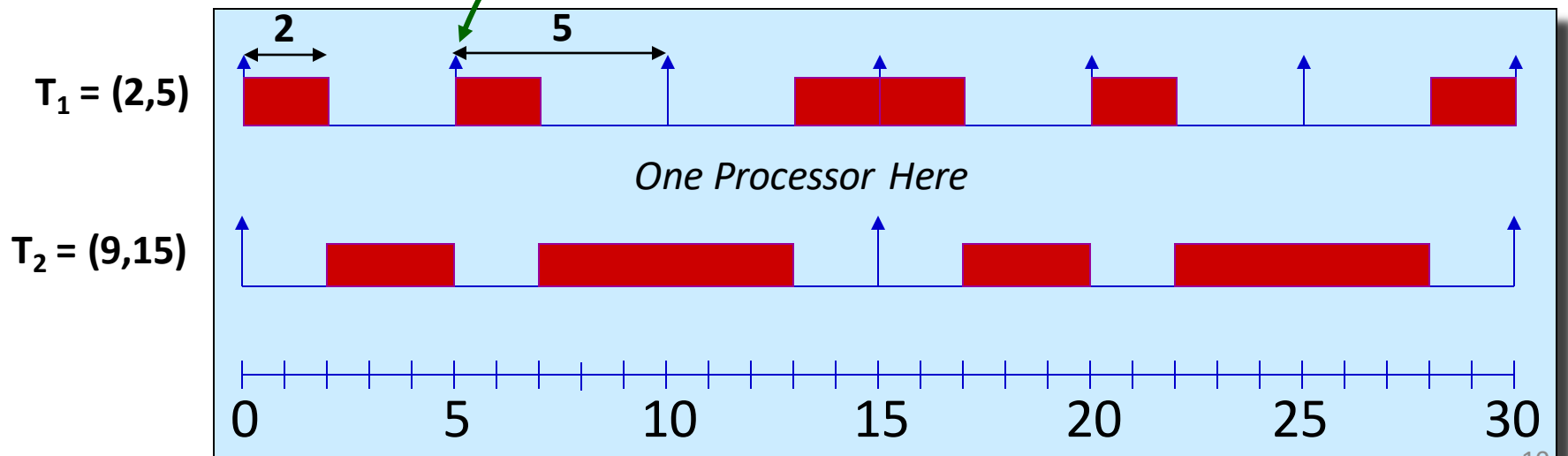
Periodic Task Systems

- Set τ of **periodic tasks** scheduled on **M cores**:
 - Task $T_i = (e_i, p_i)$ releases a **job** with exec. cost e_i every p_i time units.
 - T_i 's **utilization** (or **weight**) is $u_i = e_i/p_i$.
 - Total utilization** is $U(\tau) = \sum_{T_i} e_i/p_i$.
 - Each job of T_i has a **deadline** at the next job release of T_i .



Periodic Task Systems

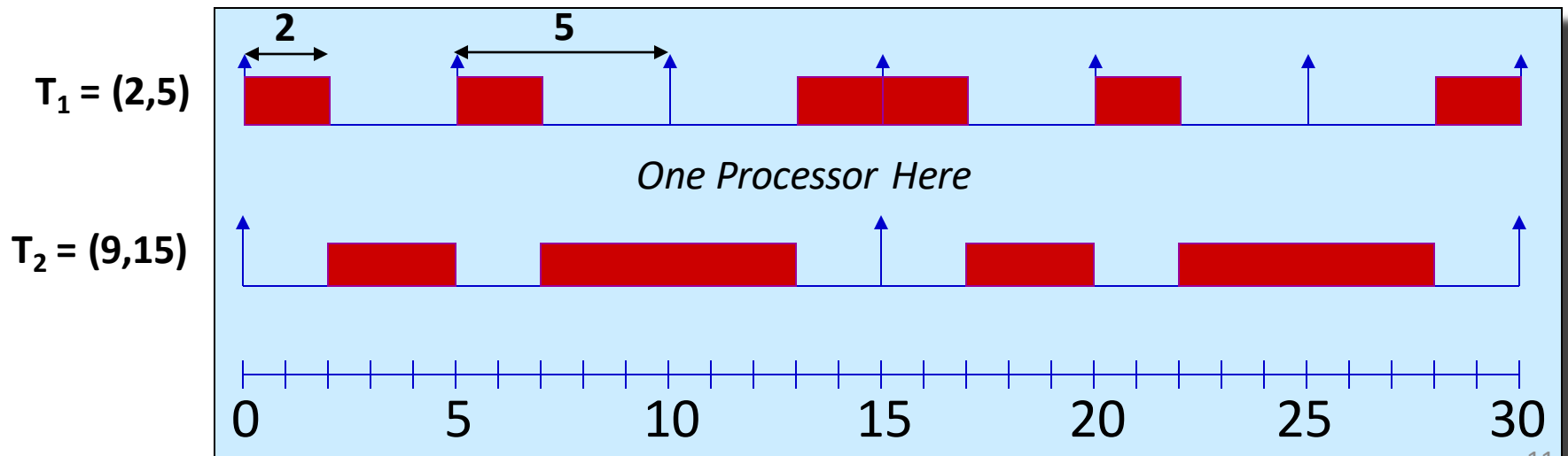
- Set τ of **periodic tasks** scheduled on **M cores**:
 - Task $T_i = (e_i, p_i)$ releases a **job** with exec. cost e_i every p_i time units.
 - T_i 's **utilization** (or **weight**) is $u_i = e_i/p_i$.
 - Total utilization** is $U(\tau) = \sum_{T_i} e_i/p_i$.
 - Each job of T_i has a **deadline** at the next job release of T_i .



Periodic Task Systems

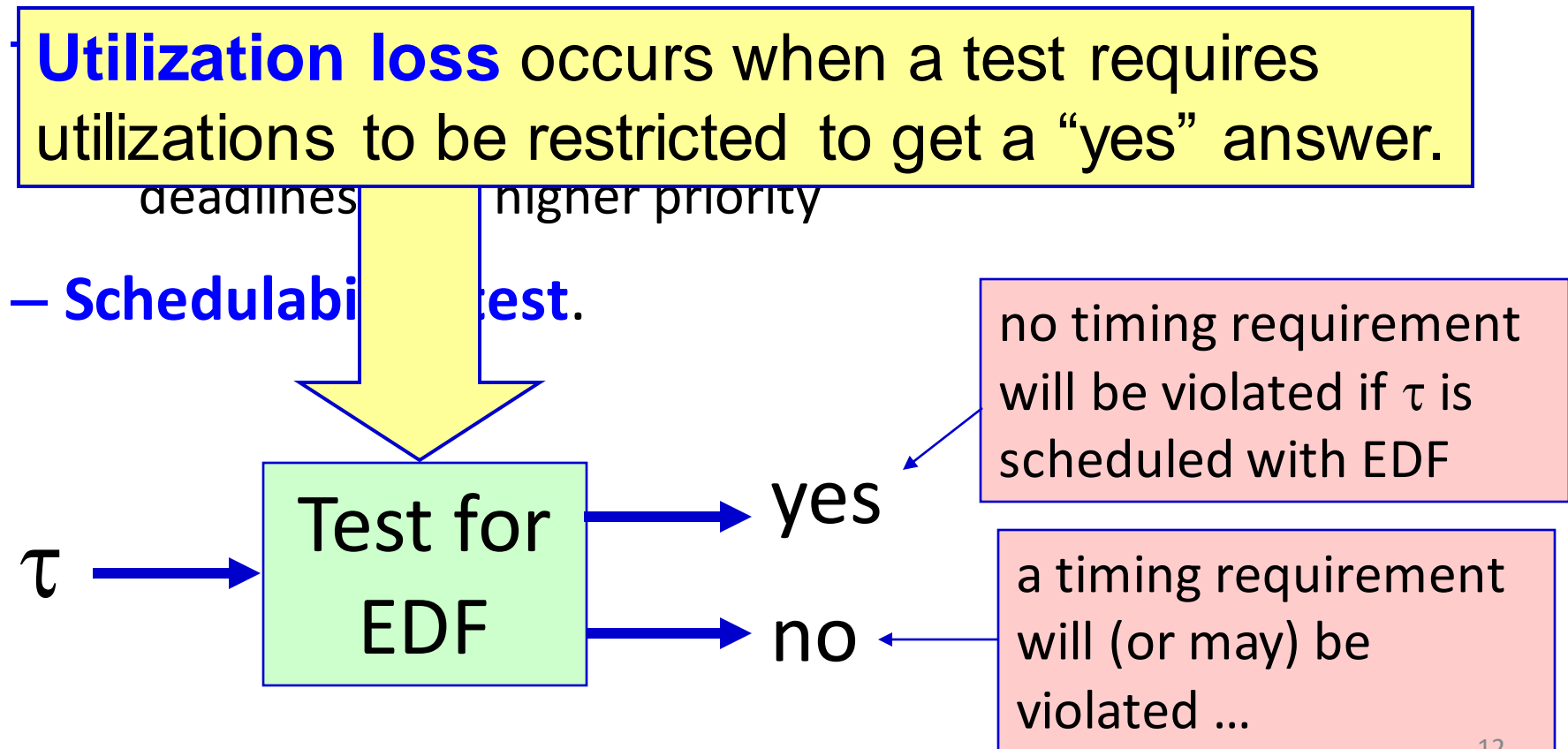
- Set τ of **periodic tasks** scheduled on **M cores**:
 - Task $T_i = (e_i, p_i)$ releases a **job** with exec. cost e_i every p_i time units.
 - T_i 's **utilization** (or **weight**) is $u_i = e_i/p_i$.
 - Total utilization** is $U(\tau) = \sum_{T_i} e_i/p_i$.

This is an example of an **earliest-deadline-first (EDF)** schedule.



Scheduling vs. Schedulability

- W.r.t. scheduling, we actually care about two kinds of algorithms:



Optimality and Feasibility

- A schedule is **feasible** if all timing constraints are met
- A task set T is **schedulable** using scheduling algorithm A if A always produces a feasible schedule for T
- A scheduling algorithm is **optimal** if it always produces a feasible schedule when one exists (under any scheduling algorithm)

Static-Priority Scheduling

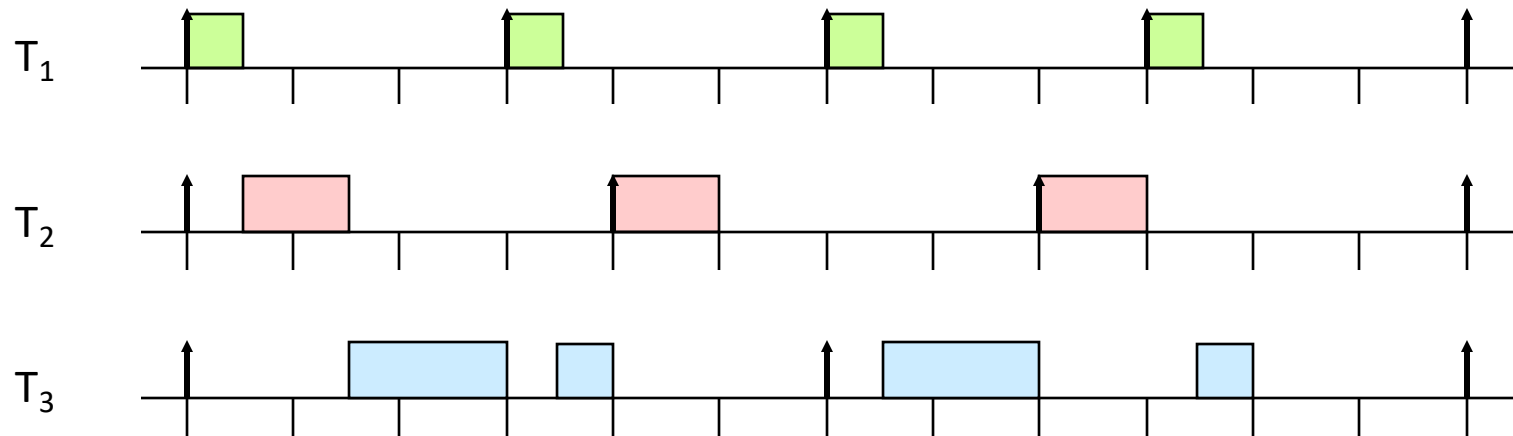
- Under **fixed-priority** scheduling, different jobs of a task are assigned the same priority.
- We will assume that tasks are indexed in decreasing priority order, i.e., **T_i has higher priority than T_k if $i < k$.**
- The ready task with the highest priority is always scheduled.

Rate-Monotonic Scheduling

(Liu and Layland)

Priority Definition: Tasks with smaller periods have higher priority.

Example Schedule: Three tasks, $T_1 = (0.5, 3)$, $T_2 = (1, 4)$, $T_3 = (2, 6)$.



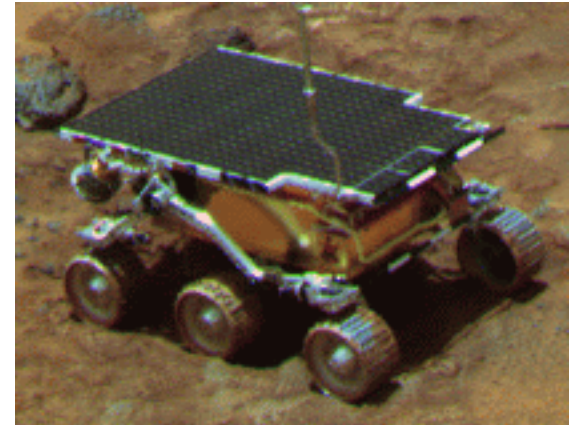
RT Synchronization 101

Priority Inversions

So far we've assumed all jobs are independent.

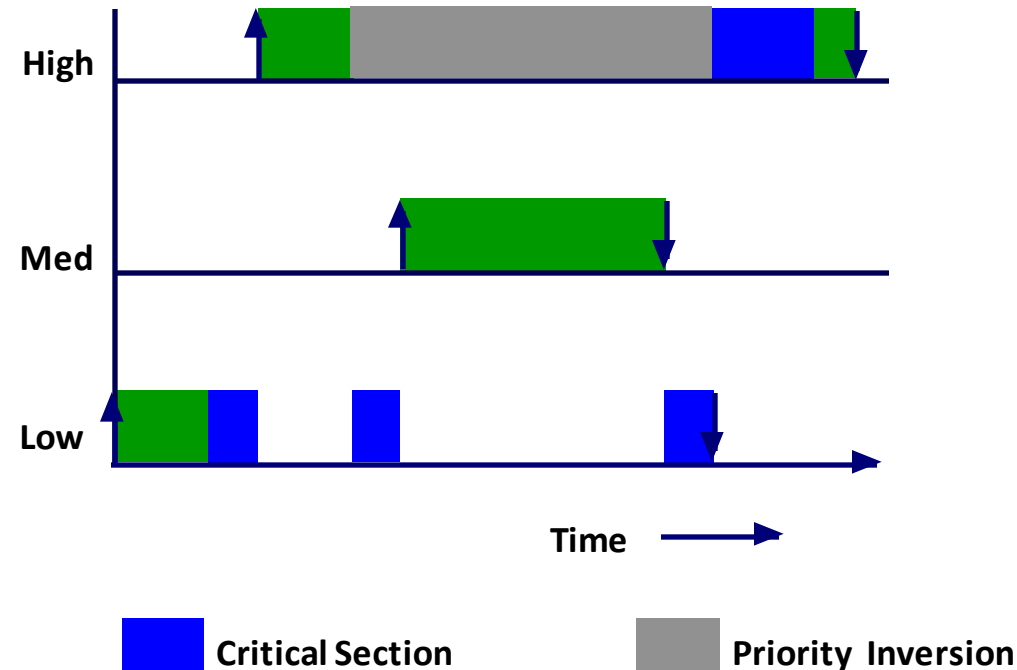
A *priority inversion* occurs when a high-priority job is blocked by a low-priority one.

This is bad because HP jobs usually have more stringent timing constraints.



Mars Pathfinder infamously had a priority inversion when deployed and it almost caused a mission failure. A patch was sent remotely patched to fix.

<https://www.rapitasystems.com/blog/what-really-happened-software-mars-pathfinder-spacecraft>

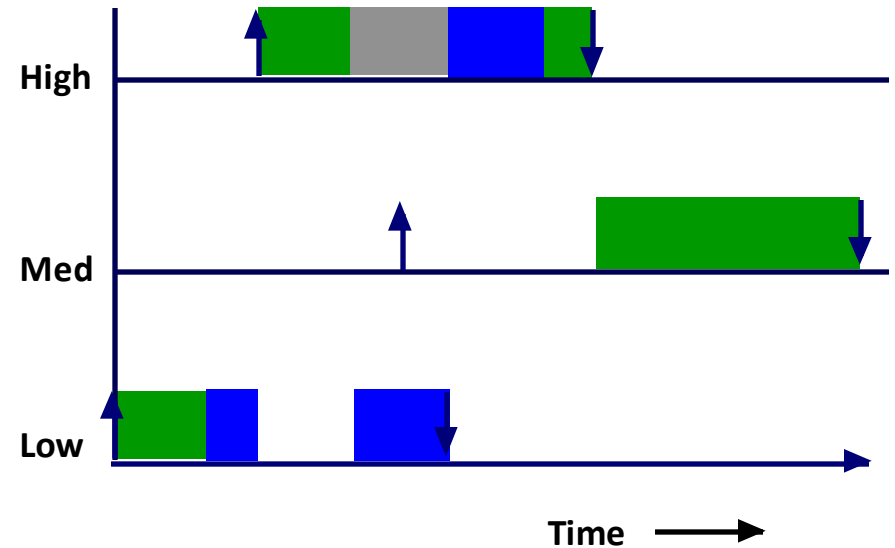
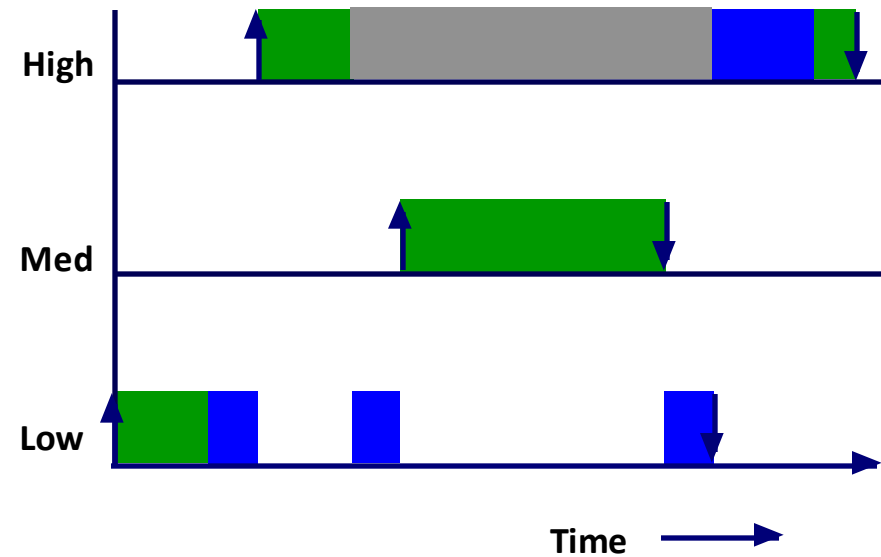
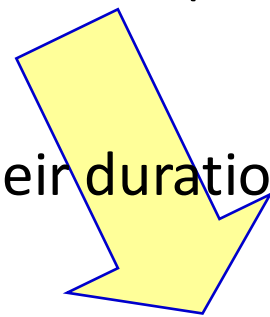


RT Synchronization 101

Priority Inheritance

A Common Solution: Use *priority inheritance* (blocking job executes at blocked job's priority).

Doesn't prevent inversions but limits their duration.



Critical Section



Priority Inversion



Computation Outside of CS's

Summary

- Real-time systems differ from general-purpose ones in that there exist timing requirements
- Common in cyber-physical and safety-critical systems, such as avionics, automotive, and other embedded devices.
- Timing requirements inform how scheduling should be handled
- Many classes of real-time scheduling algorithms
- Analysis complements the scheduling implementation to prove temporal correctness