Timing Guarantees

INTRODUCTION TO REAL-TIME SCHEDULING

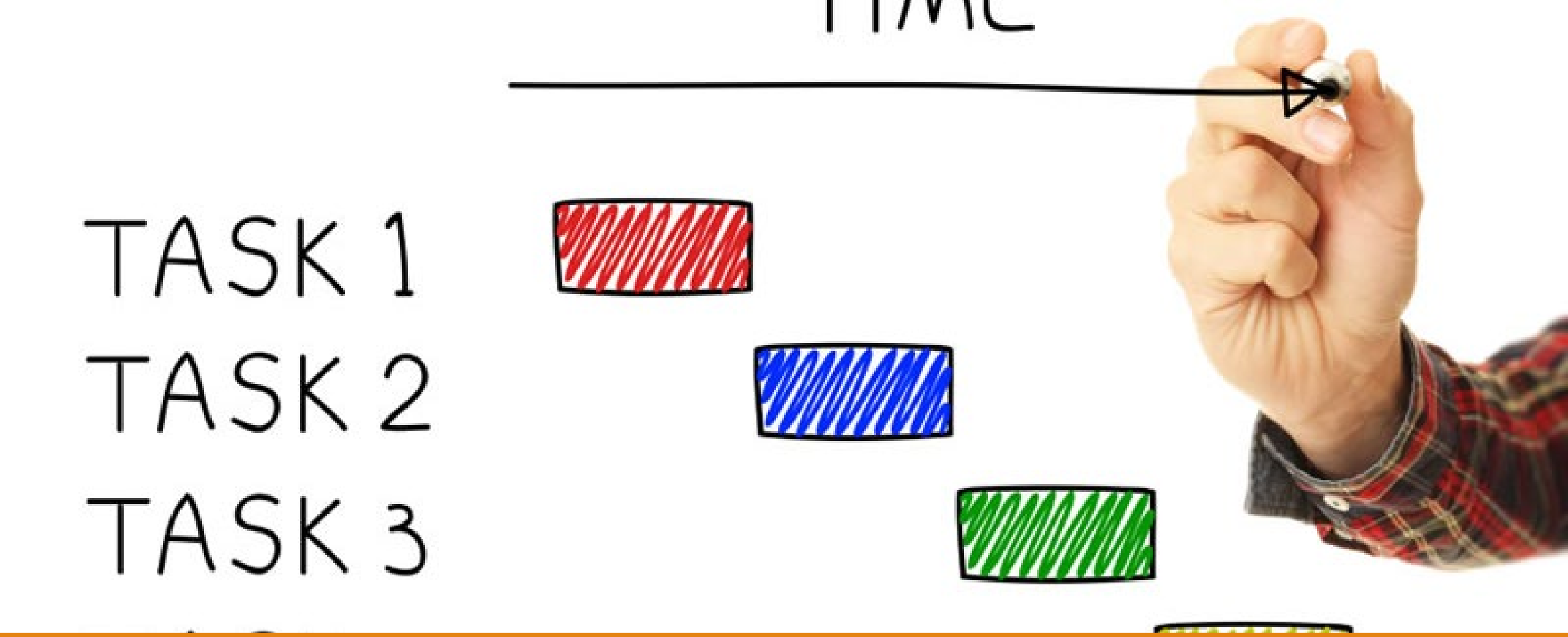
The Task Model for Edge AI

Most AI applications in the IoT space read data from sensors (e.g., read a video frame or a window of time-series measurements) then perform processing on it and report the results

This cyclic processing gives rise to the “periodic task” (scheduling) model

Typically, processing of each frame/window of data must finish before the next one arrives. Thus, the periodic task has a “deadline” (usually at the end of the period)

Multiple tasks may be involved (e.g., multiple encoders of different modalities, or multiple AI modules that perform different functions)



Part I: Scheduling Periodic Tasks

The Schedulability Question: A Drive-by-Wire Example

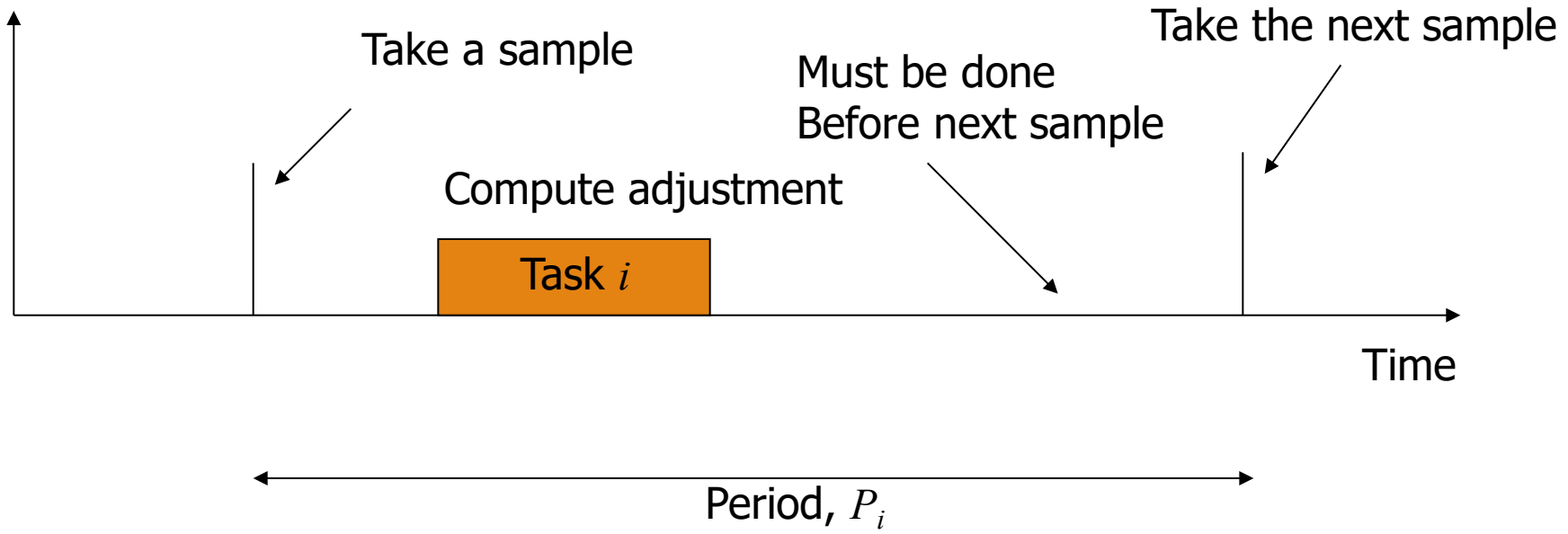
Consider an AI-driven system for an autonomous robot that features three tasks and one GPU (which is the main resources needed for task execution)

- Navigation guidance is computed every 10 ms – wheel positions adjusted accordingly (computing the adjustment takes 4.5 ms of GPU time)
- Threats and obstacles are reassessed every 4 ms – breaks adjusted accordingly (computing the adjustment takes 2ms of GPU time)
- Optimal speed is computed every 15 ms – robot speed is adjusted accordingly (computing the adjustment takes 0.45 ms of GPU time)
- For safe operation, adjustments must always be computed before the next sample is taken

Is it possible to always compute all adjustments in time?

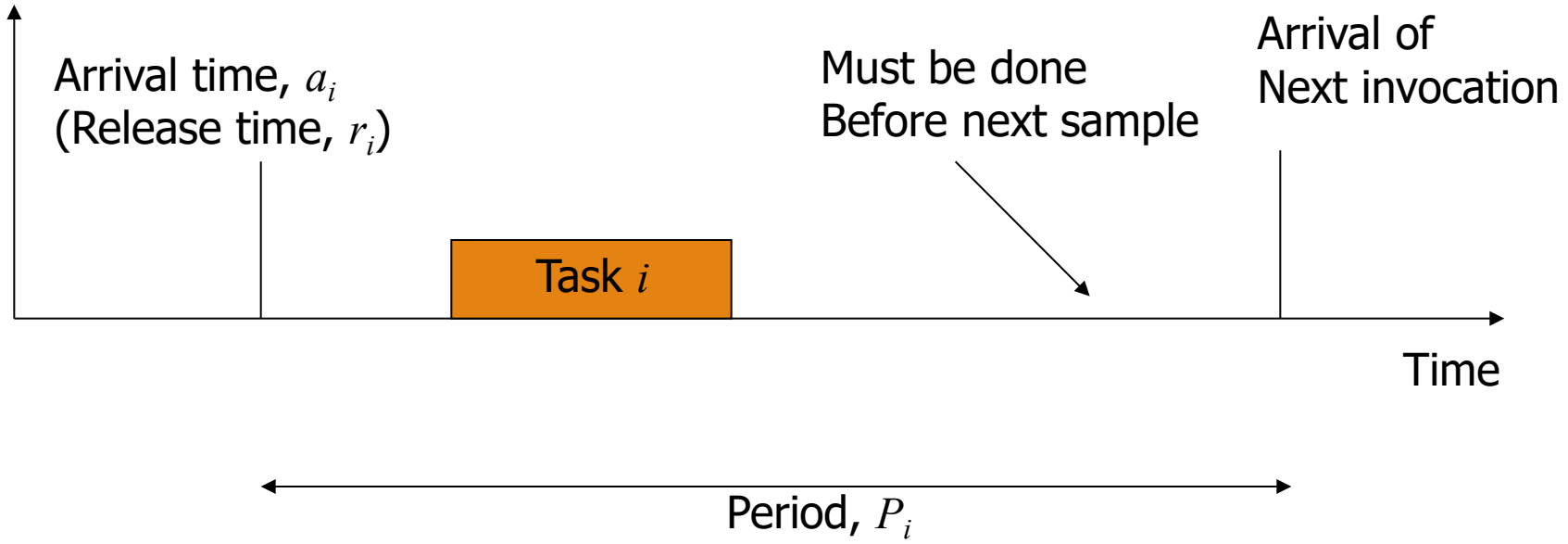
Some Terminology

Tasks, periods, arrival-time, deadline, execution time, etc.



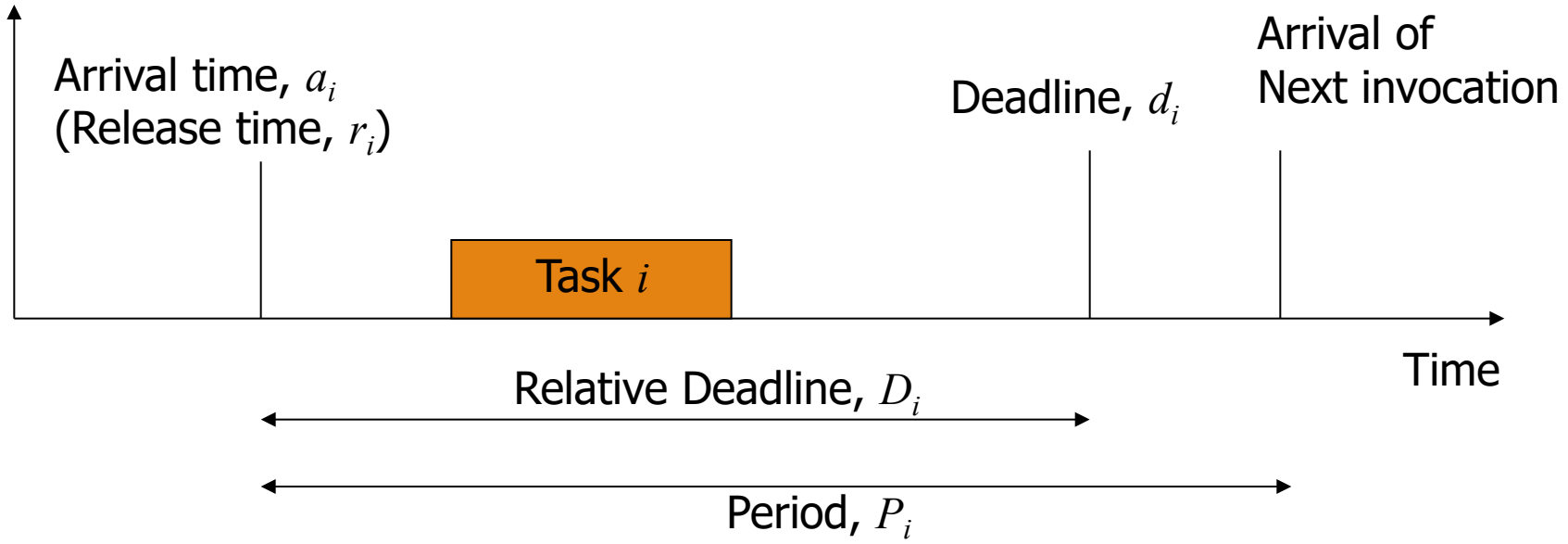
Some Terminology

Tasks, periods, arrival-time, deadline, execution time, etc.



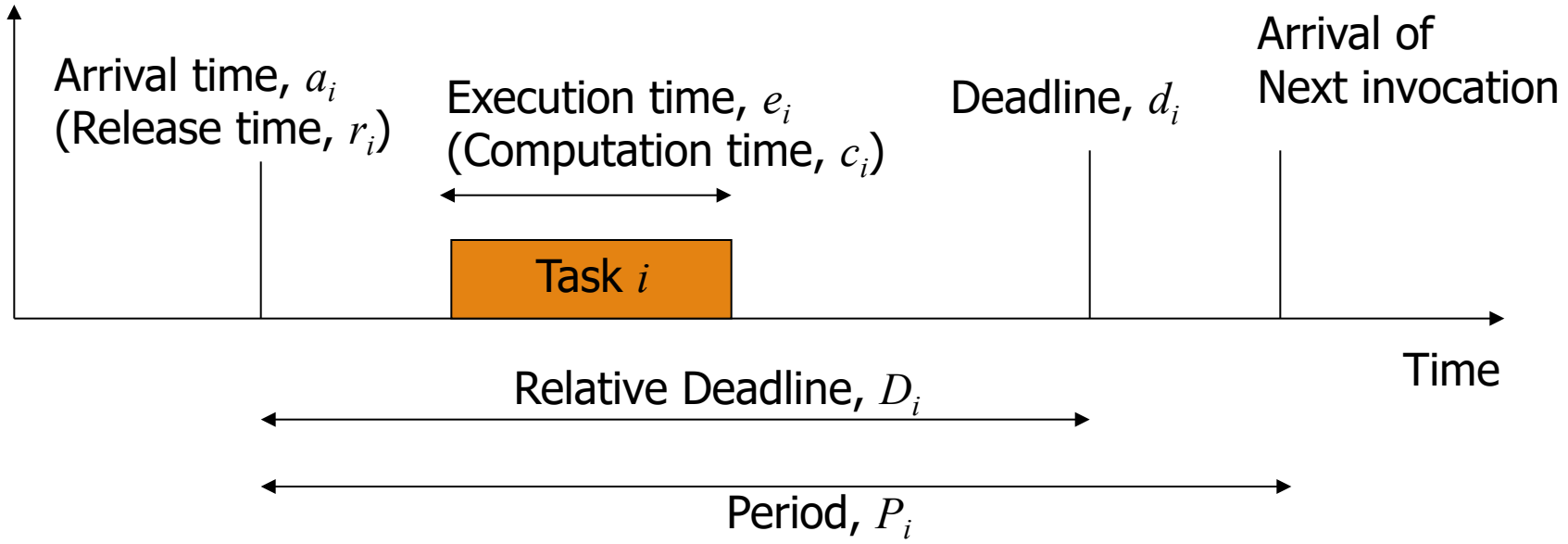
Some Terminology

Tasks, periods, arrival-time, deadline, execution time, etc.



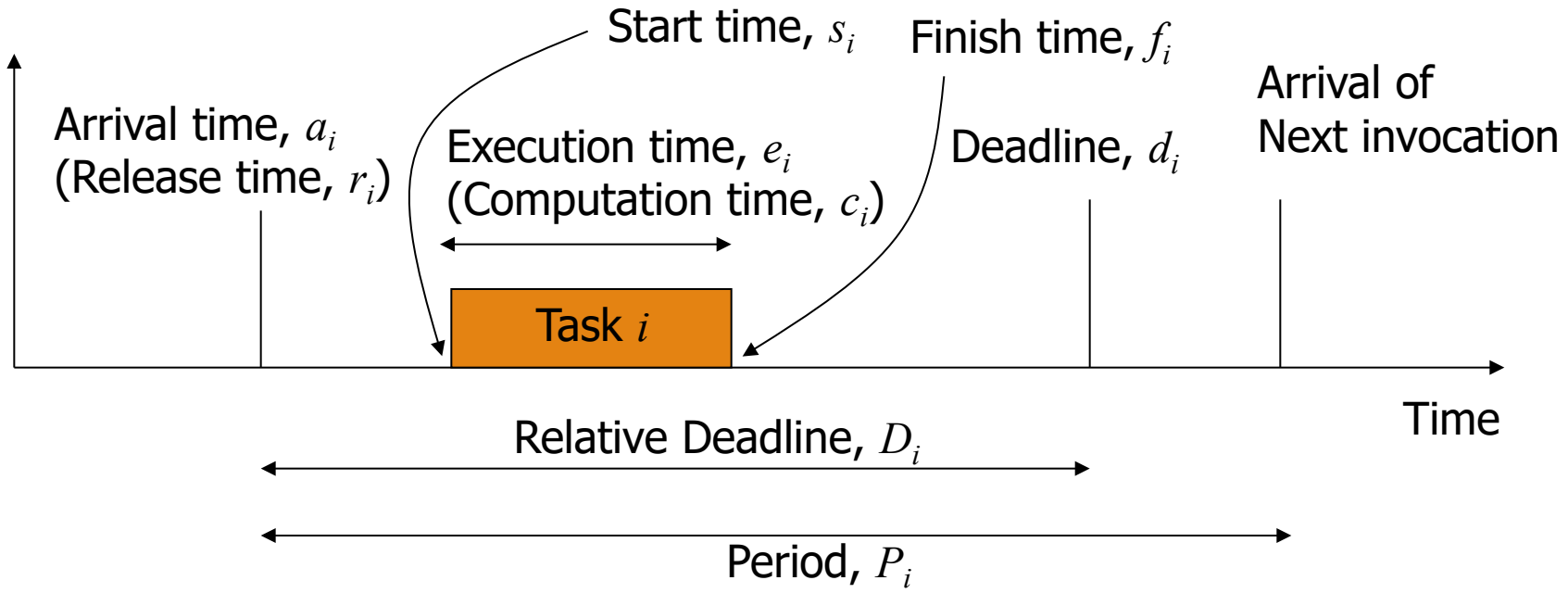
Some Terminology

Tasks, periods, arrival-time, deadline, execution time, etc.



Some Terminology

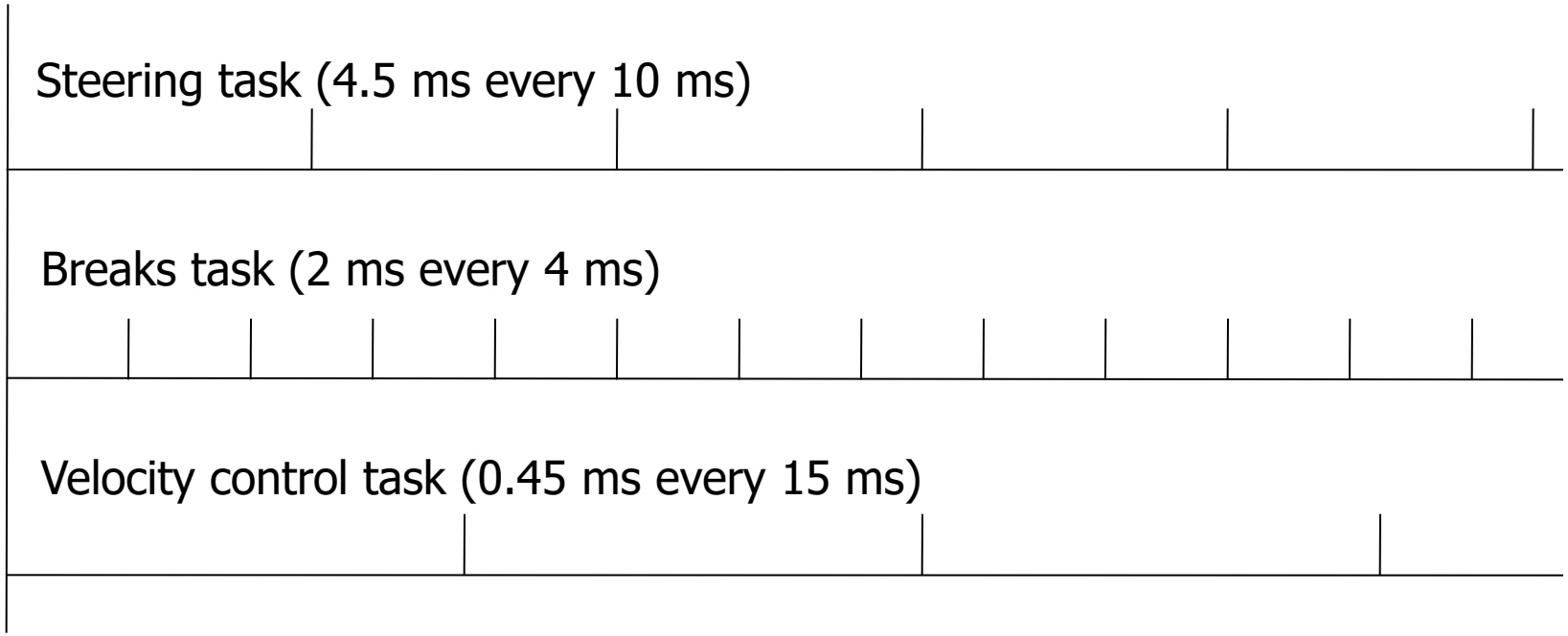
Tasks, periods, arrival-time, deadline, execution time, etc.



Back to

Drive-by-Wire Example

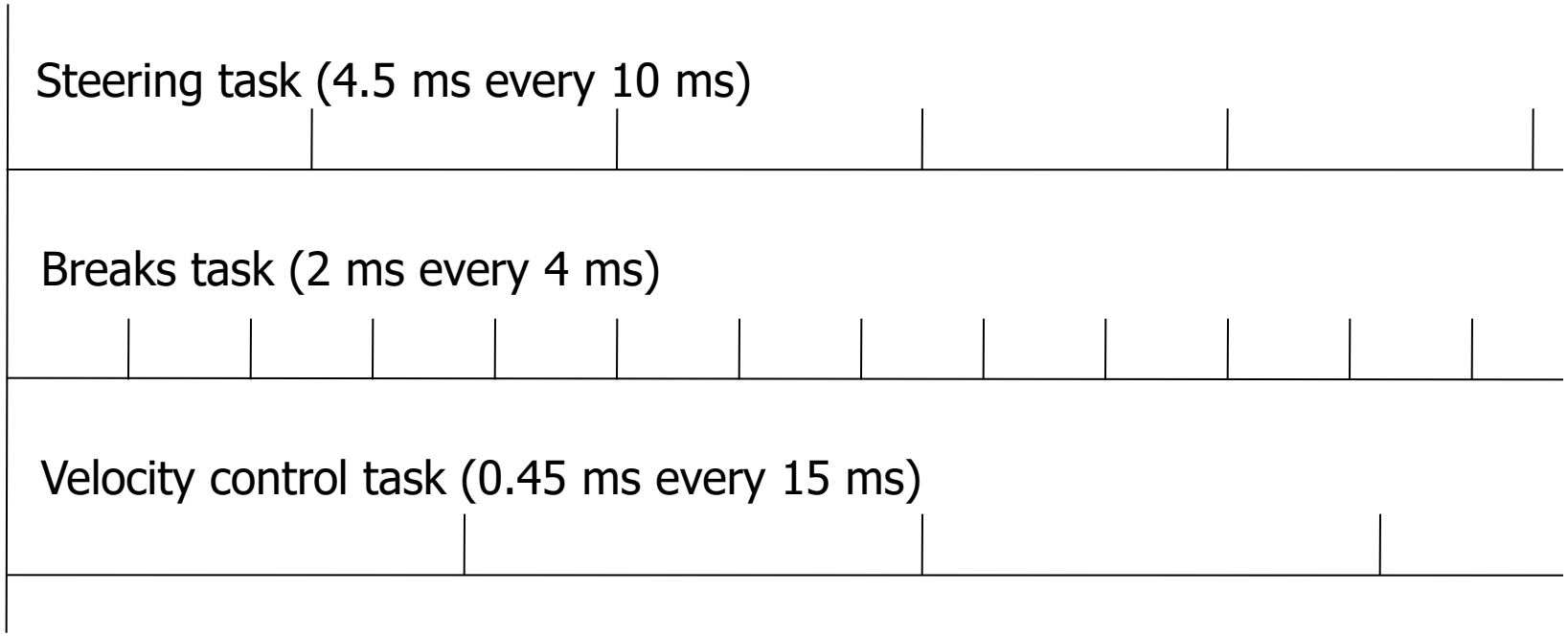
Find a schedule that makes sure all task invocations meet their deadlines. Assume one GPU is available for all tasks.



Back to

Drive-by-Wire Example

Sanity check #1: Is the GPU over-utilized?

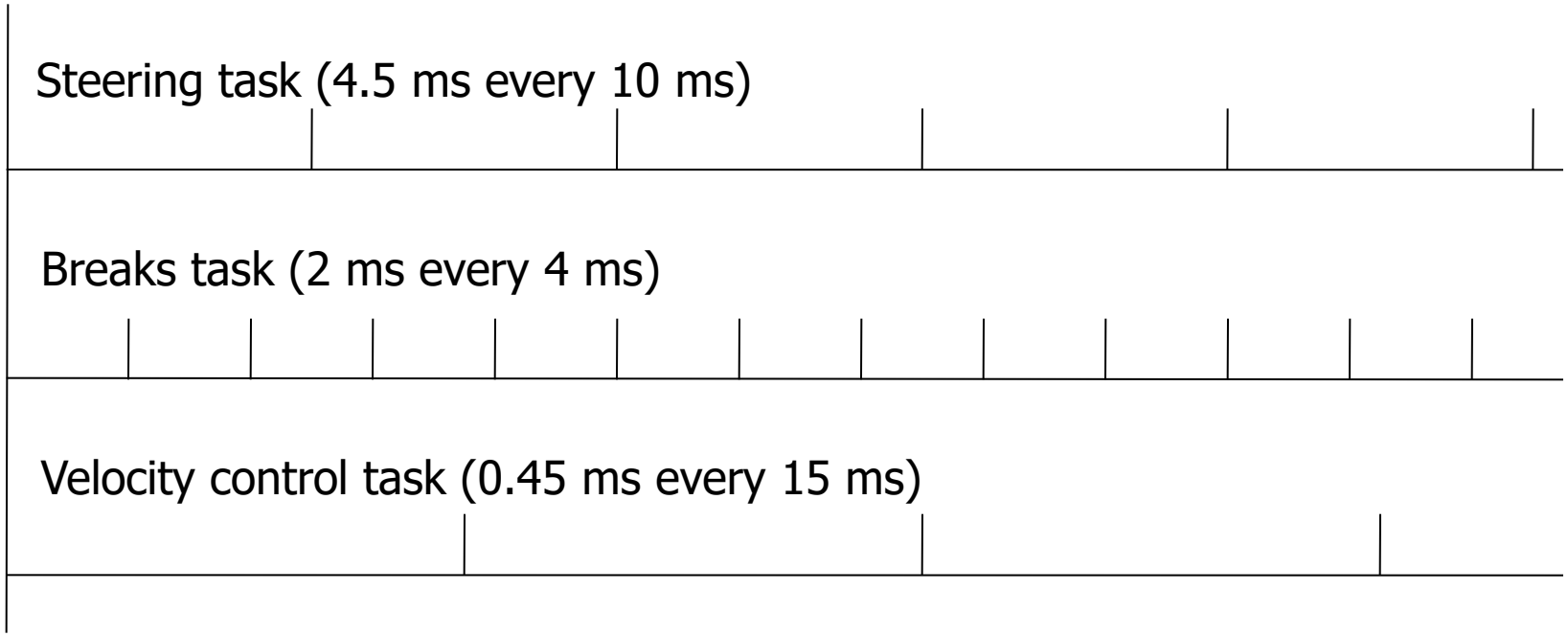


Back to

Drive-by-Wire Example

Sanity check #1: Is the GPU over-utilized?

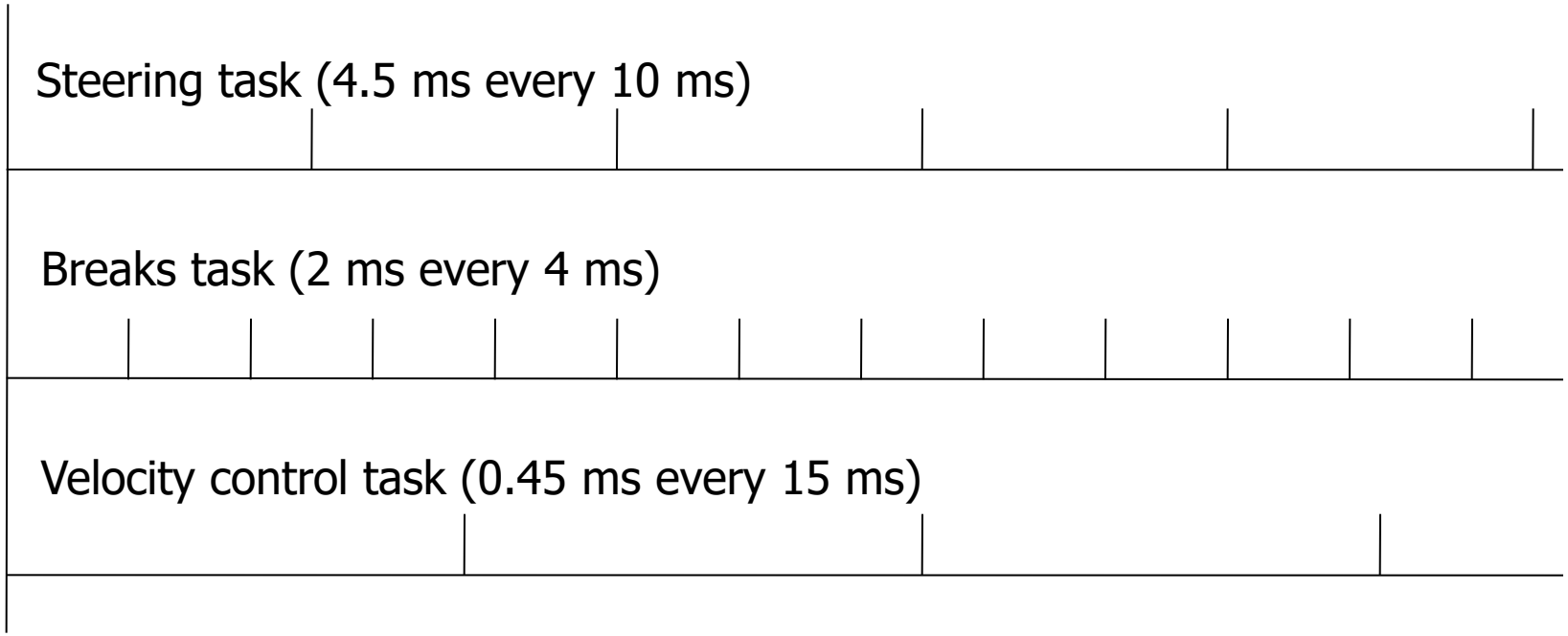
Utilization = 4.5/10 + 2/4 + 0.45/15 < 100%



Periodic Task Scheduling

In what order should tasks be executed?

How to assign priorities to tasks?

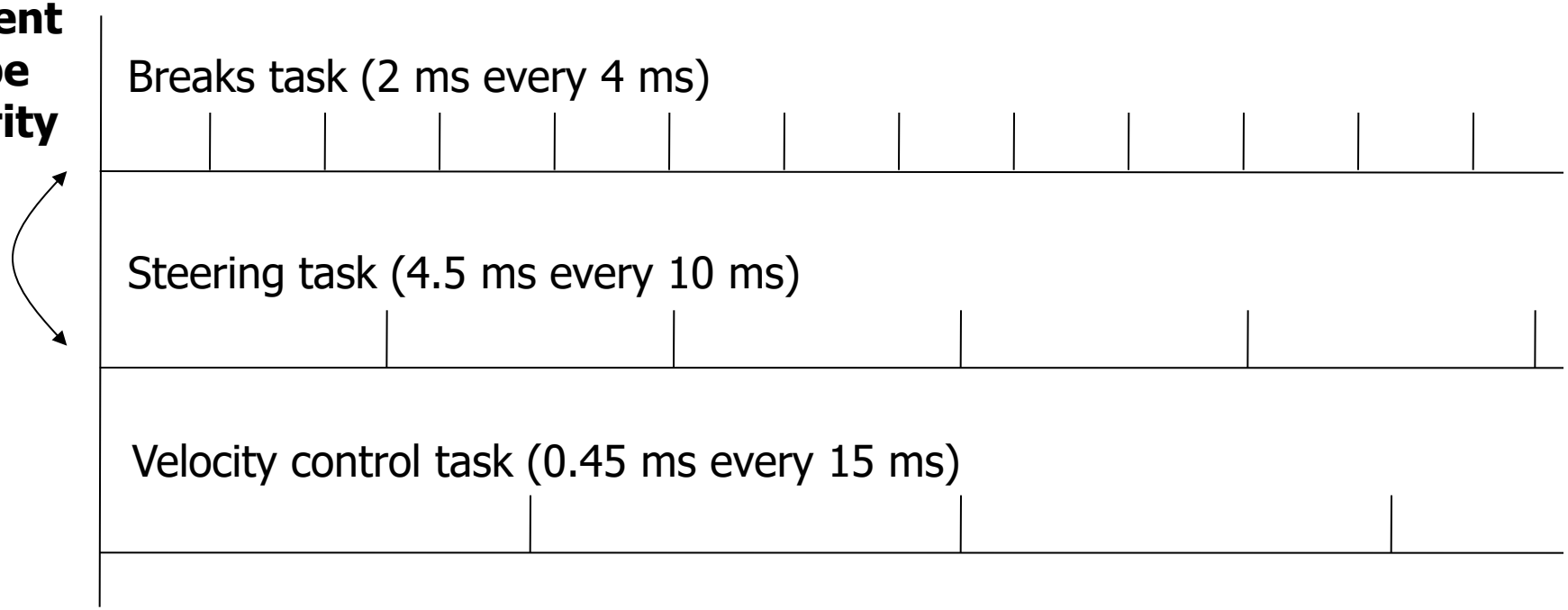


Periodic Task Scheduling

Decision #1: In what order should tasks be executed?

Consider rate monotonic scheduling: Smaller period → higher priority

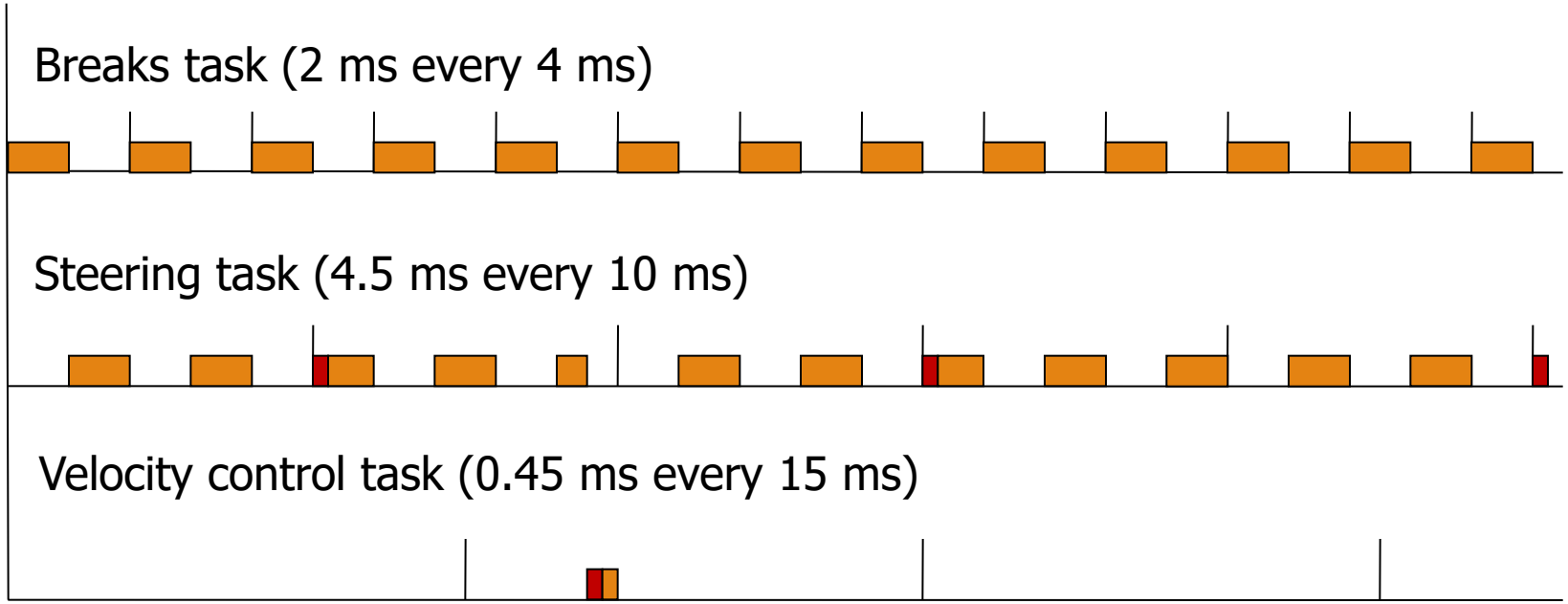
Intuition: Urgent tasks should be higher in priority



Periodic Task Scheduling

Note that in this example, deadlines are missed!

(Although Utilization < 100%)

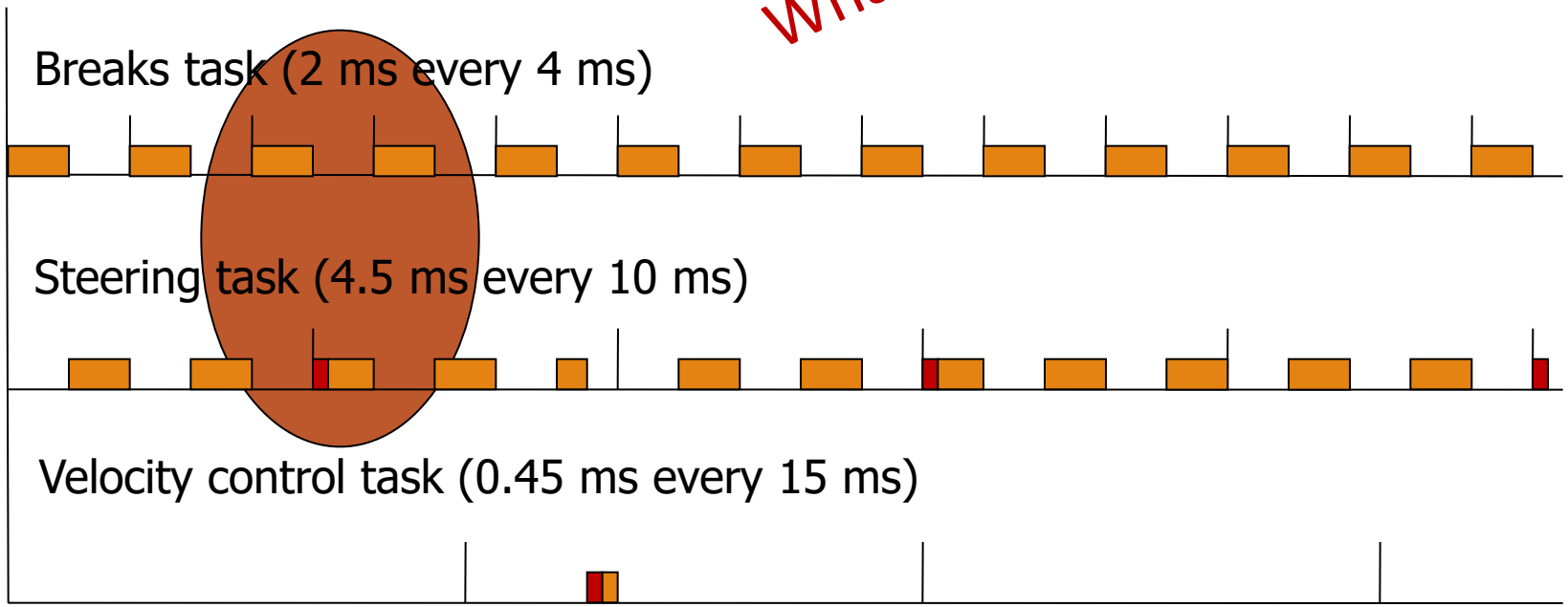


Periodic Task Scheduling

Deadlines are missed!

Average Utilization < 100%

What went wrong?



Periodic Task Scheduling

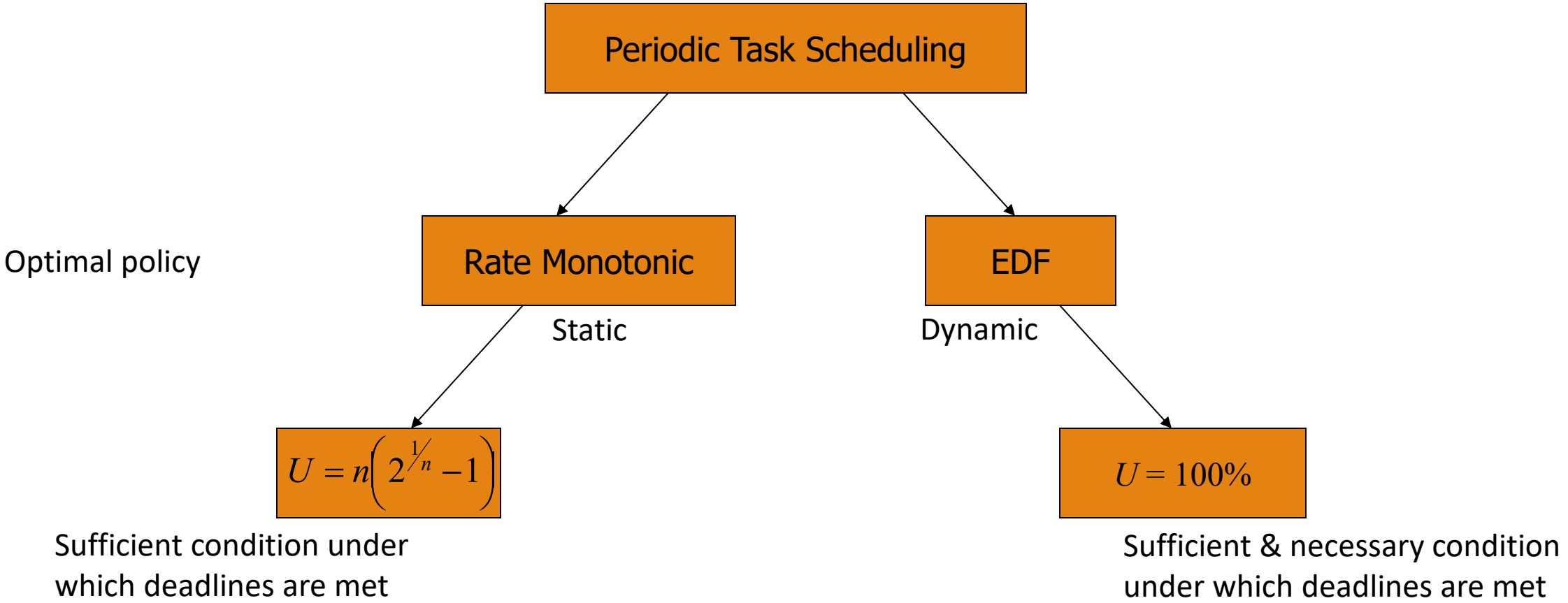
What is the optimal *static priority* scheduling policy? Under which conditions is it able to meet all deadline?

- Static priority means: All invocations of the same task have the same priority
- Optimal (static policy) means: It can meet deadlines whenever any other static priority policy can meet deadlines.

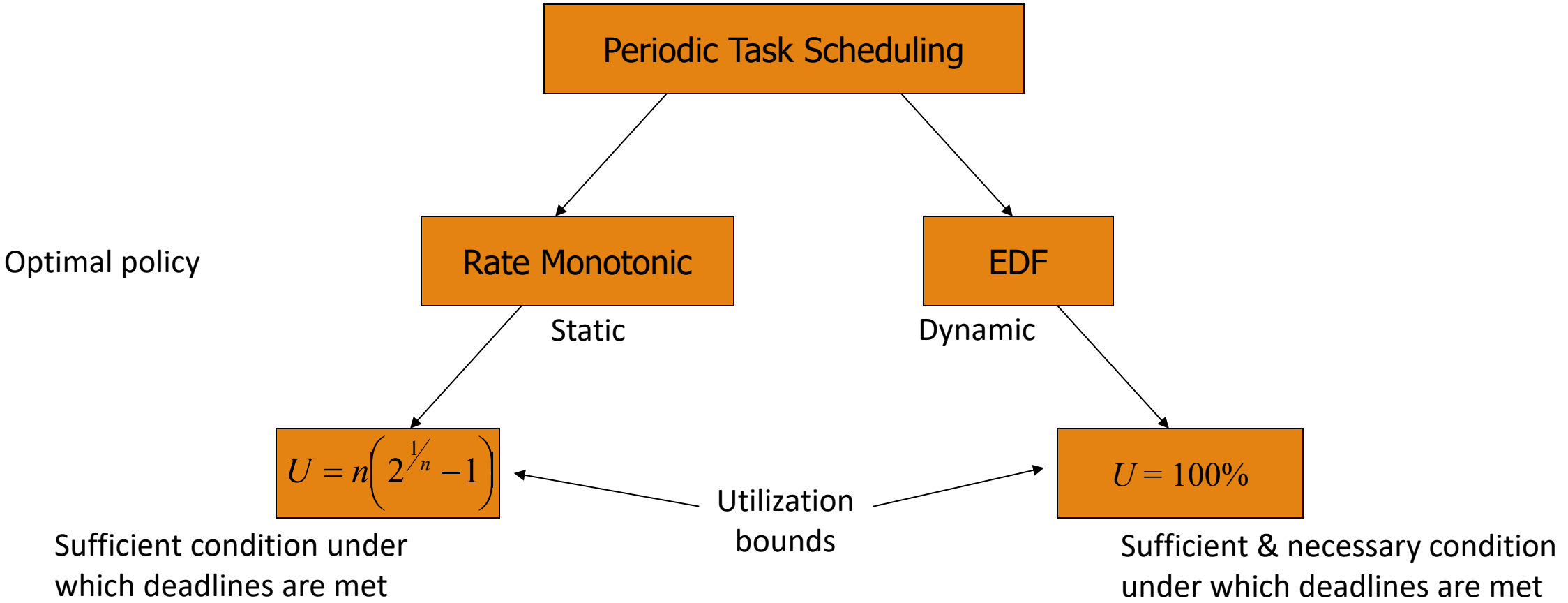
What is the optimal *dynamic priority* scheduling policy? Under which conditions is it able to meet all deadline?

- Dynamic priority means: Each separate task invocation has its own priority
- Optimal (dynamic policy) means: It can meet deadlines whenever any other dynamic priority policy can meet deadlines.

Main Results in Real-time Scheduling of Periodic Tasks



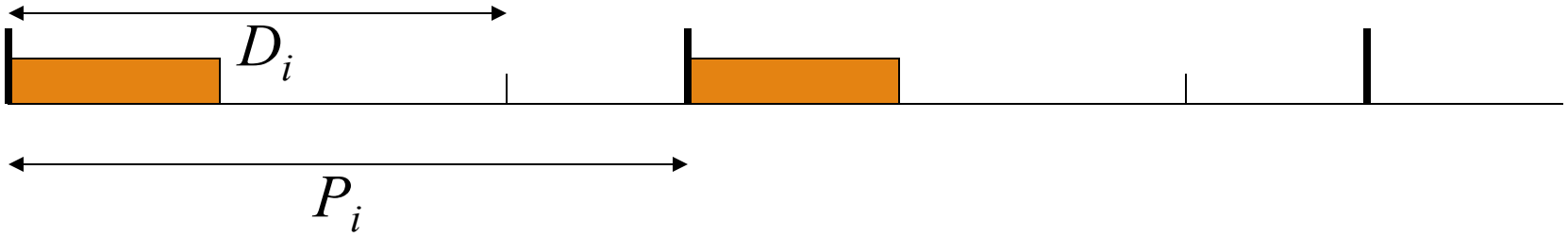
Main Results in Real-time Scheduling of Periodic Tasks



Deadline Monotonic Scheduling

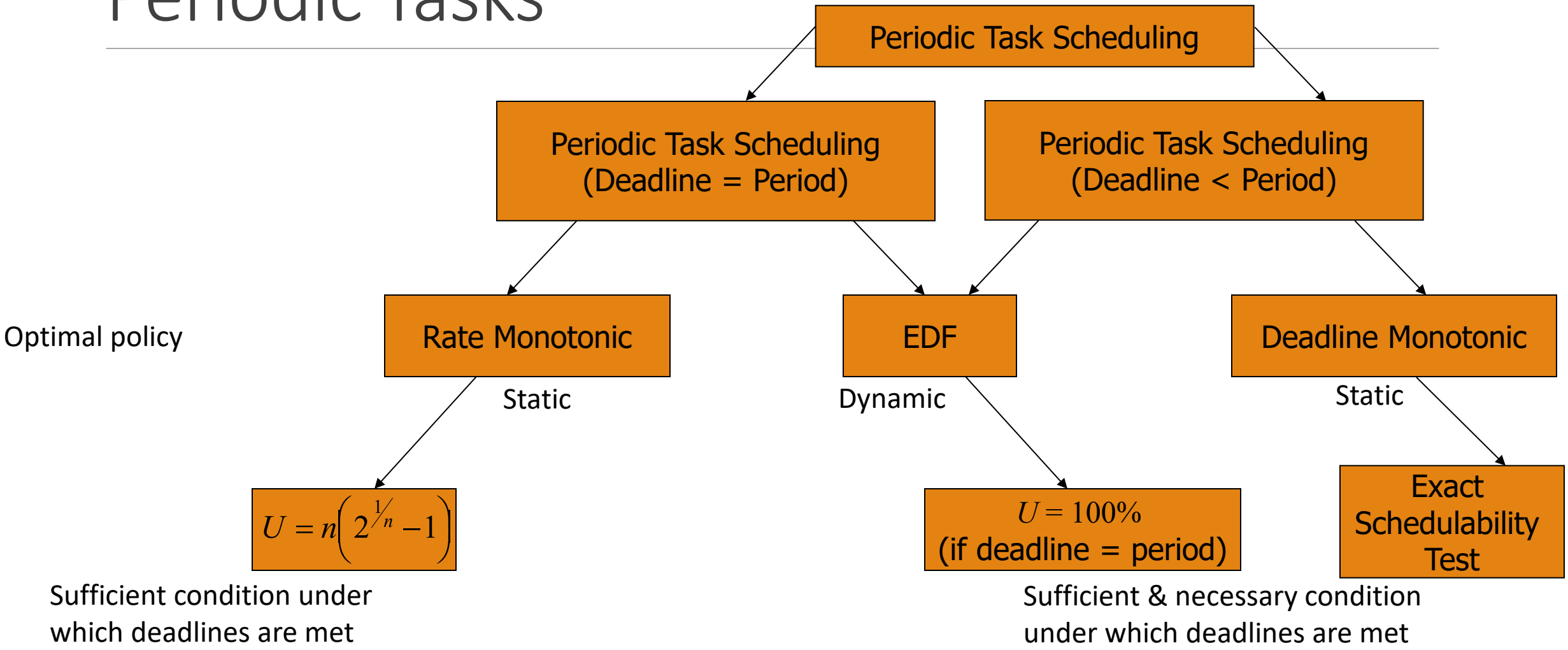
The model is motivated by situations where the sampling period at which the AI algorithm observes the world is larger than the desired reaction time to external stimuli.

Consider a set of periodic tasks where each task, i, has a computation time, C_i, a period, P_i, and a relative deadline $D_i < P_i$.



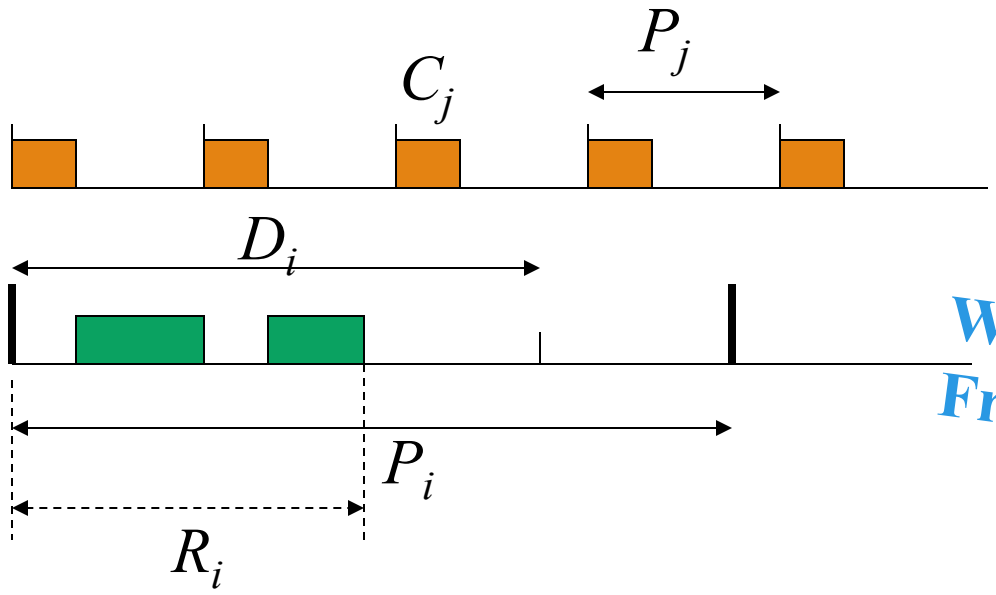
What is the schedulability condition?

Main Results in Real-time Scheduling of Periodic Tasks



The Exact Schedulability Test

Consider interference (with the execution of task, i) caused by higher priority tasks, j:



$$I = \sum_j \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

Worst case interference, *I*, From higher priority tasks

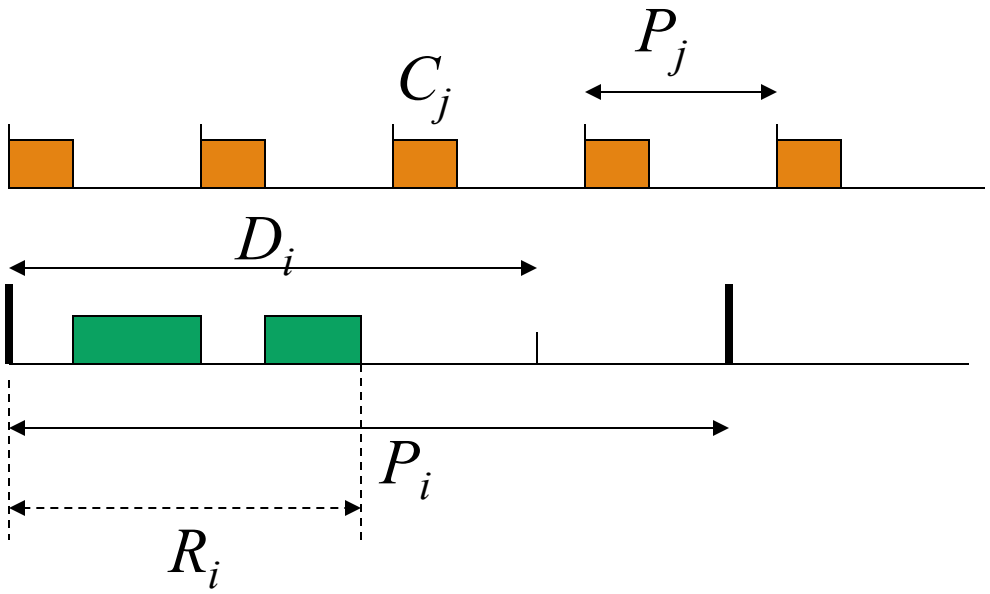
Where:

$$R_i = I + C_i$$

The Exact Schedulability Test: (An Example)

$$I = \sum_j \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

$$R_i = I + C_i$$



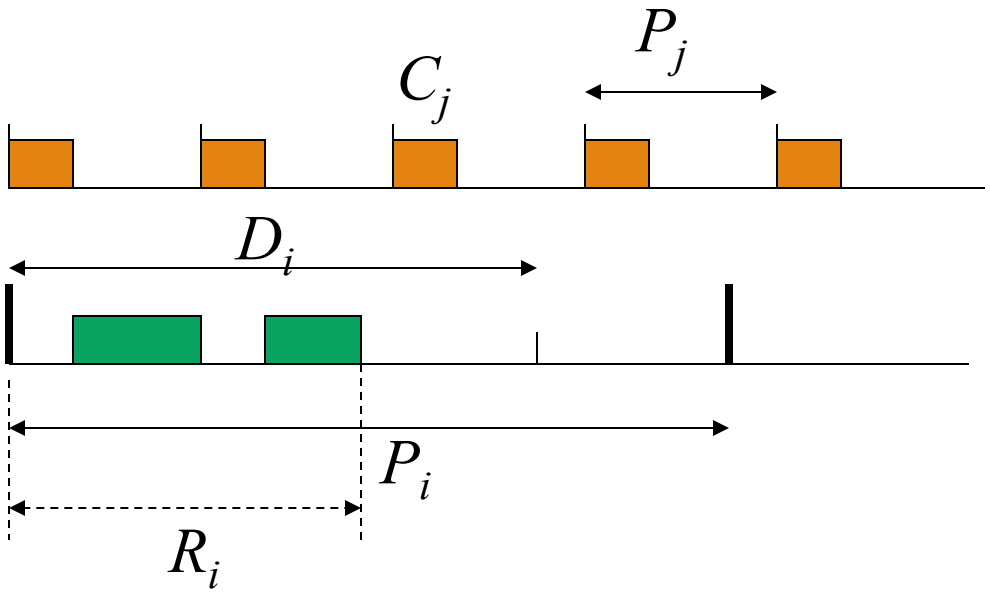
Consider a system of two tasks:

Task 1: P_1=1.7, D_1=0.5, C_1=0.5
Task 2: P_2=8, D_2=3.2, C_2=2

The Exact Schedulability Test: (An Example)

$$I = \sum_j \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

$$R_i = I + C_i$$



$$I^{(0)} = C_1 = 0.5$$

$$R_2^{(0)} = I^{(0)} + C_2 = 2.5$$

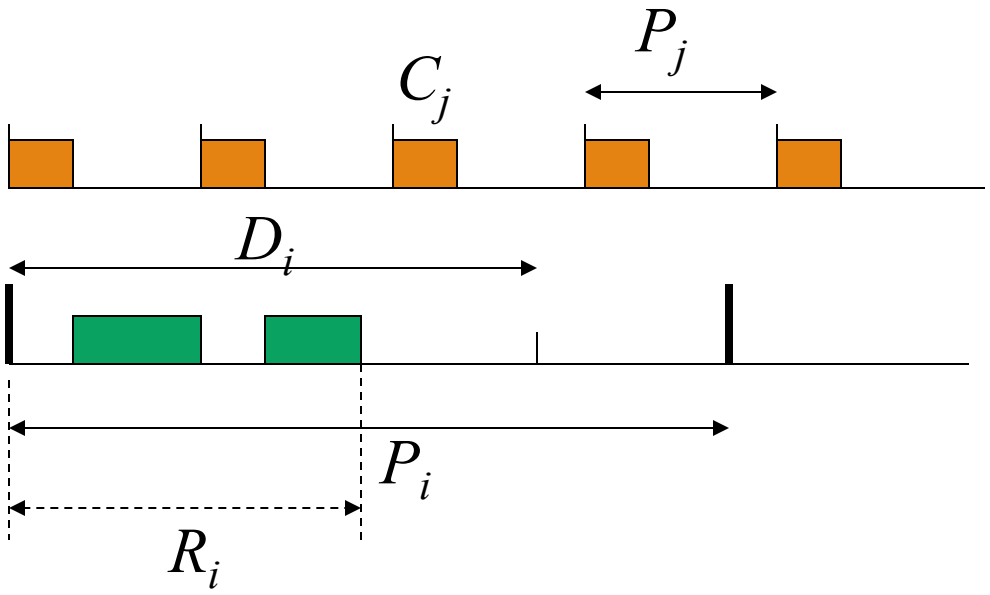
Consider a system of two tasks:

Task 1: P_1=1.7, D_1=0.5, C_1=0.5
Task 2: P_2=8, D_2=3.2, C_2=2

The Exact Schedulability Test: (An Example)

$$I = \sum_j \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

$$R_i = I + C_i$$



Consider a system of two tasks:

Task 1: P_1=1.7, D_1=0.5, C_1=0.5
Task 2: P_2=8, D_2=3.2, C_2=2

$$I^{(0)} = C_1 = 0.5$$

$$R_2^{(0)} = I^{(0)} + C_2 = 2.5$$

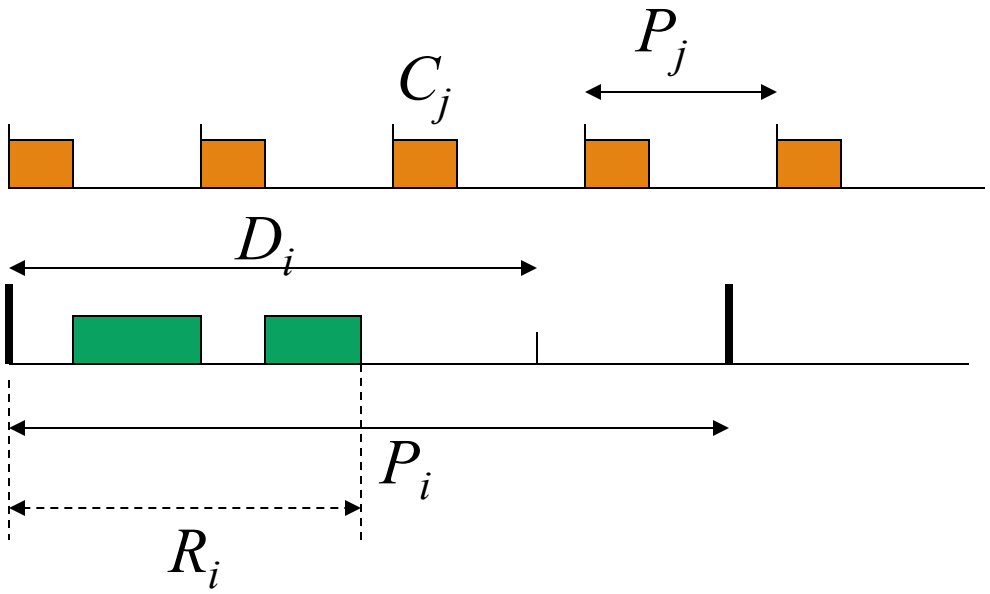
$$I^{(1)} = \left\lceil \frac{R_2^{(0)}}{P_1} \right\rceil C_1 = \left\lceil \frac{2.5}{1.7} \right\rceil 0.5 = 1$$

$$R_2^{(1)} = I^{(1)} + C_2 = 3$$

The Exact Schedulability Test: (An Example)

$$I = \sum_j \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

$$R_i = I + C_i$$



Consider a system of two tasks:

Task 1: P_1=1.7, D_1=0.5, C_1=0.5
Task 2: P_2=8, D_2=3.2, C_2=2

$$I^{(0)} = C_1 = 0.5$$

$$R_2^{(0)} = I^{(0)} + C_2 = 2.5$$

$$I^{(1)} = \left\lceil \frac{R_2^{(0)}}{P_1} \right\rceil C_1 = \left\lceil \frac{2.5}{1.7} \right\rceil 0.5 = 1$$

$$R_2^{(1)} = I^{(1)} + C_2 = 3$$

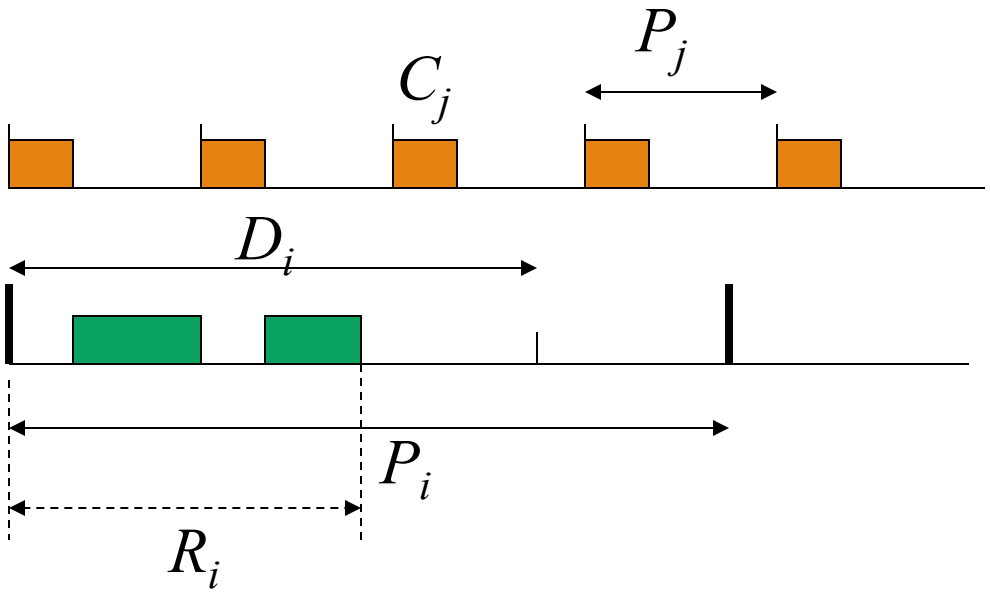
$$I^{(2)} = \left\lceil \frac{R_2^{(1)}}{P_1} \right\rceil C_1 = \left\lceil \frac{3}{1.7} \right\rceil 0.5 = 1$$

$$R_2^{(2)} = I^{(2)} + C_2 = 3$$

The Exact Schedulability Test: (An Example)

$$I = \sum_j \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

$$R_i = I + C_i$$



Consider a system of two tasks:

Task 1: P_1=1.7, D_1=0.5, C_1=0.5
Task 2: P_2=8, D_2=3.2, C_2=2

$$I^{(0)} = C_1 = 0.5$$

$$R_2^{(0)} = I^{(0)} + C_2 = 2.5$$

$$I^{(1)} = \left\lceil \frac{R_2^{(0)}}{P_1} \right\rceil C_1 = \left\lceil \frac{2.5}{1.7} \right\rceil 0.5 = 1$$

$$R_2^{(1)} = I^{(1)} + C_2 = 3$$

$$I^{(2)} = \left\lceil \frac{R_2^{(1)}}{P_1} \right\rceil C_1 = \left\lceil \frac{3}{1.7} \right\rceil 0.5 = 1$$

$$R_2^{(2)} = I^{(2)} + C_2 = 3$$

$3 < 3.2 \rightarrow$ Ok!

Handling One-Time Jobs (Aperiodics)

Question: how to execute one-time jobs (called aperiodic jobs) without violating schedulability guarantees given to periodic tasks?

Mixed Periodic and Aperiodic Task Systems

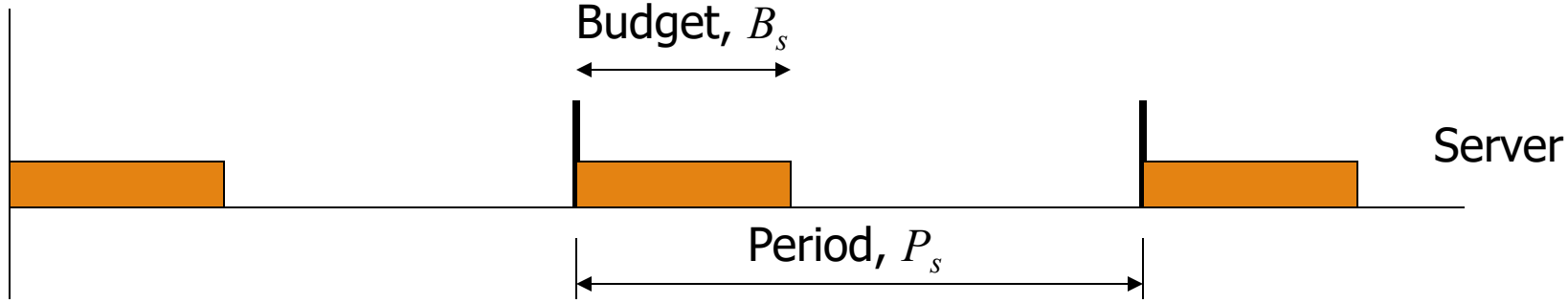
Idea: aperiodic jobs can be served by periodically invoked servers

The server can be accounted for in periodic task schedulability analysis

The server has a period P_s and a budget B_s

Server can serve aperiodic jobs until budget expires

Servers have different flavors depending on the details of when they are invoked, what priority they have, and how budgets are replenished



Mixed Periodic and Aperiodic Task Systems

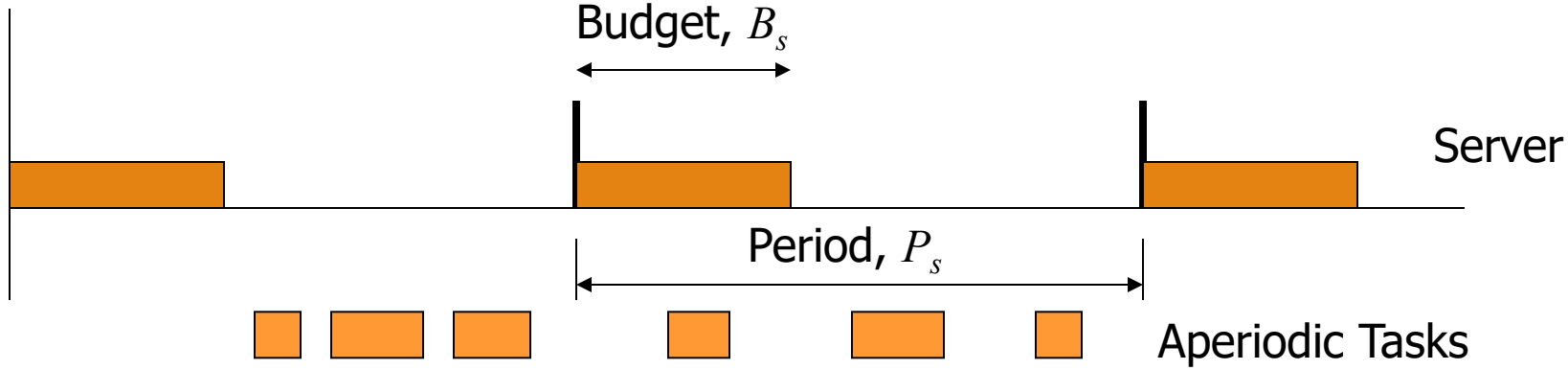
Idea: aperiodic jobs can be served by periodically invoked servers

The server can be accounted for in periodic task schedulability analysis

The server has a period P_s and a budget B_s

Server can serve aperiodic jobs until budget expires

Servers have different flavors depending on the details of when they are invoked, what priority they have, and how budgets are replenished



Mixed Periodic and Aperiodic Task Systems

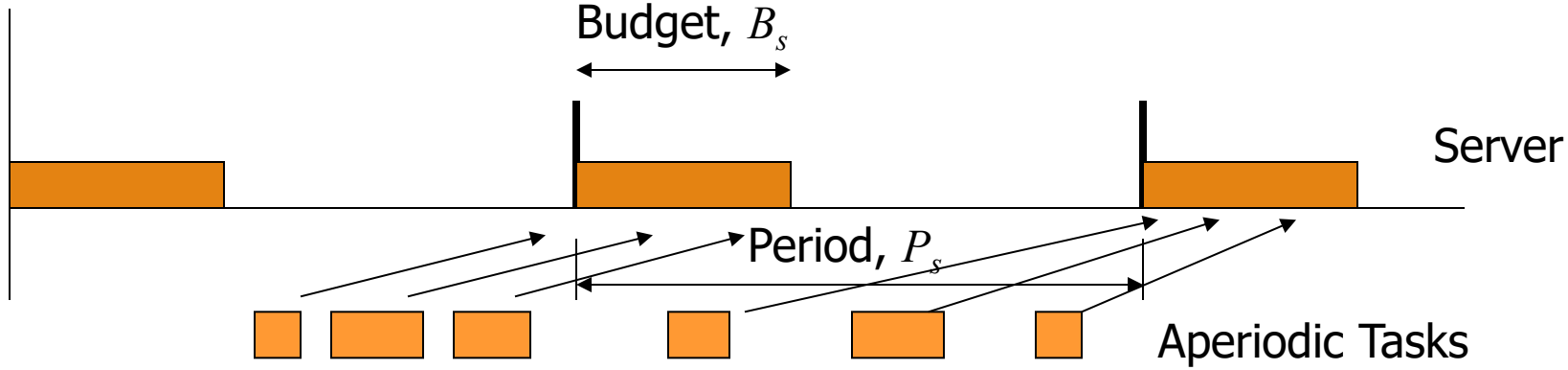
Idea: aperiodic jobs can be served by periodically invoked servers

The server can be accounted for in periodic task schedulability analysis

The server has a period P_s and a budget B_s

Server can serve aperiodic jobs until budget expires

Servers have different flavors depending on the details of when they are invoked, what priority they have, and how budgets are replenished

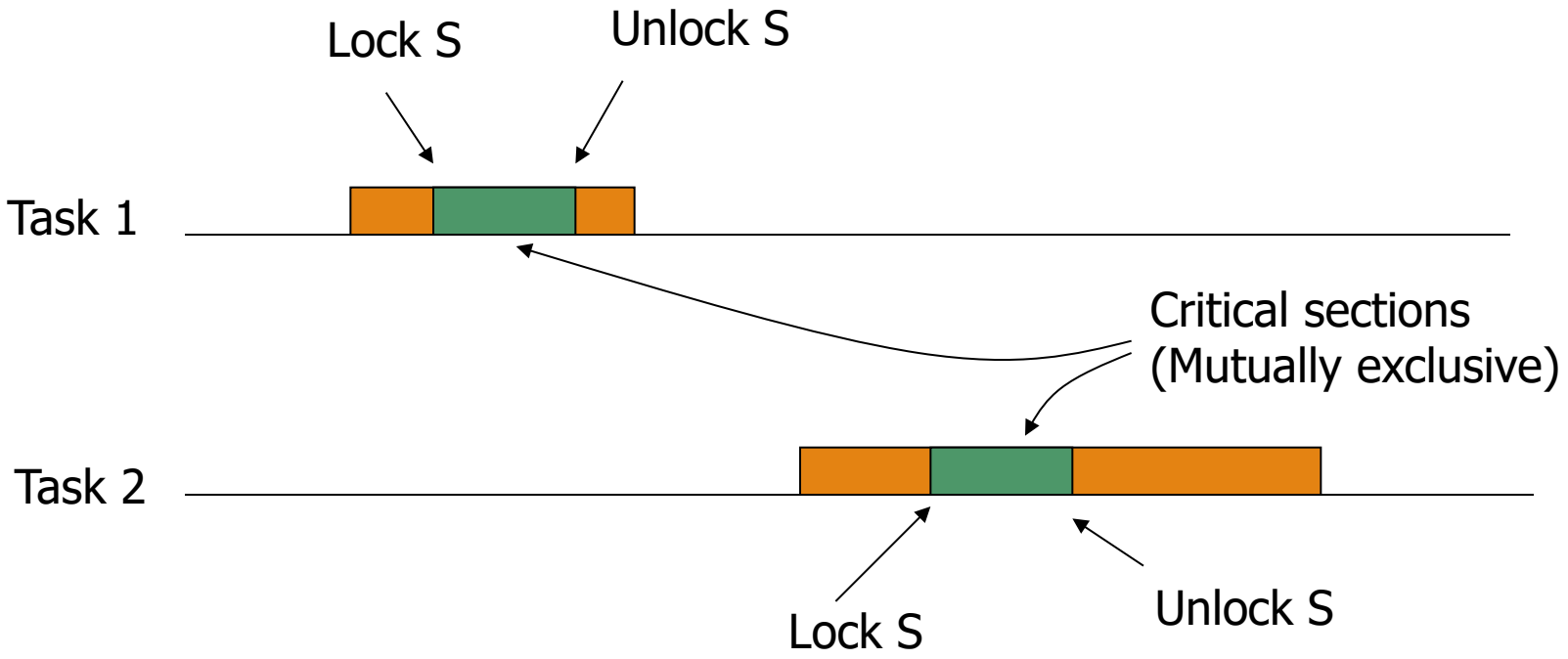


Mutual Exclusion

Question: What if tasks access resources that can only be used sequentially (i.e., in a mutually exclusive mode)?

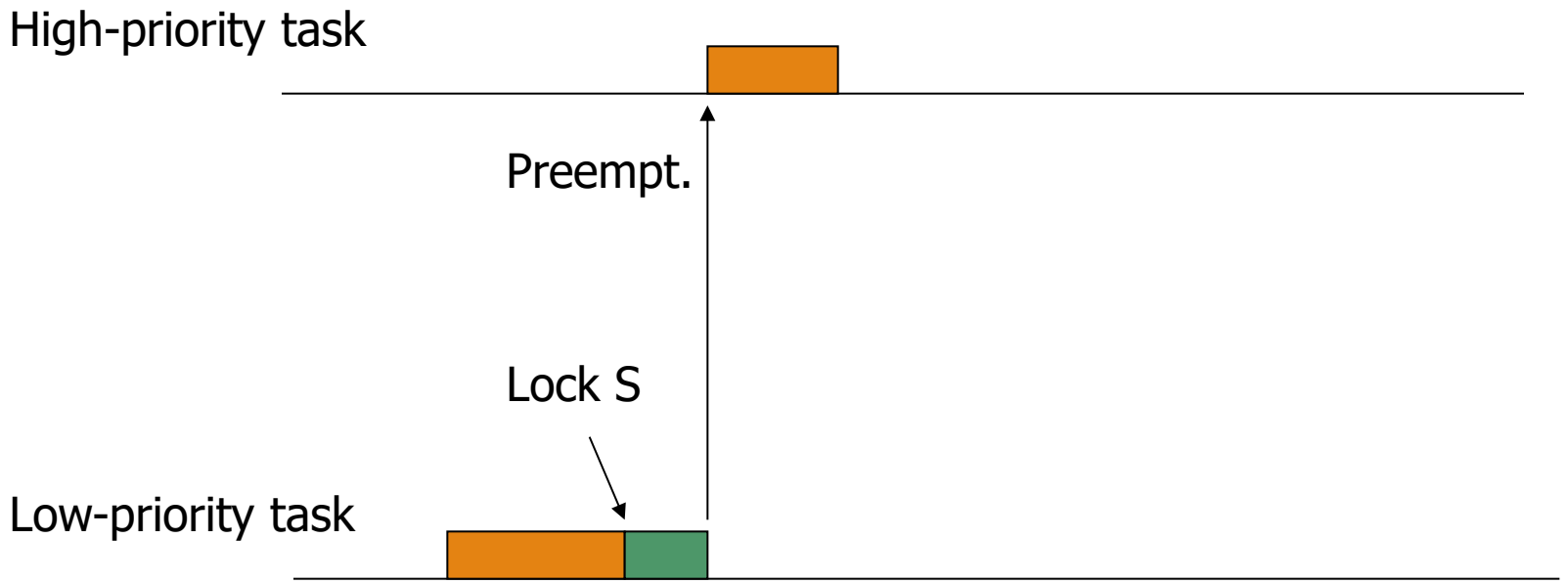
Mutual Exclusion and Non-Preemption Constraints

What if some portion of task execution is locked or not preemptible?



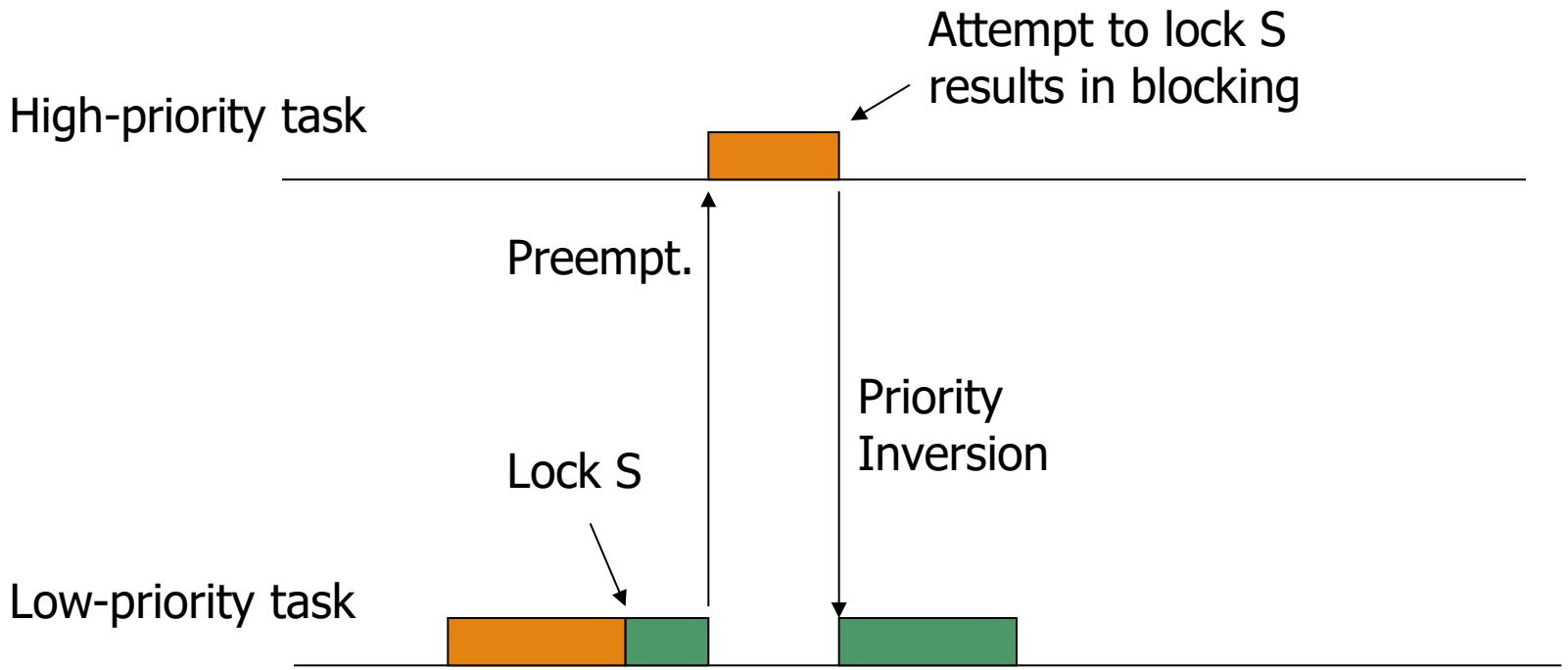
Priority Inversion

Locks and priorities may be at odds. Locking results in priority inversion



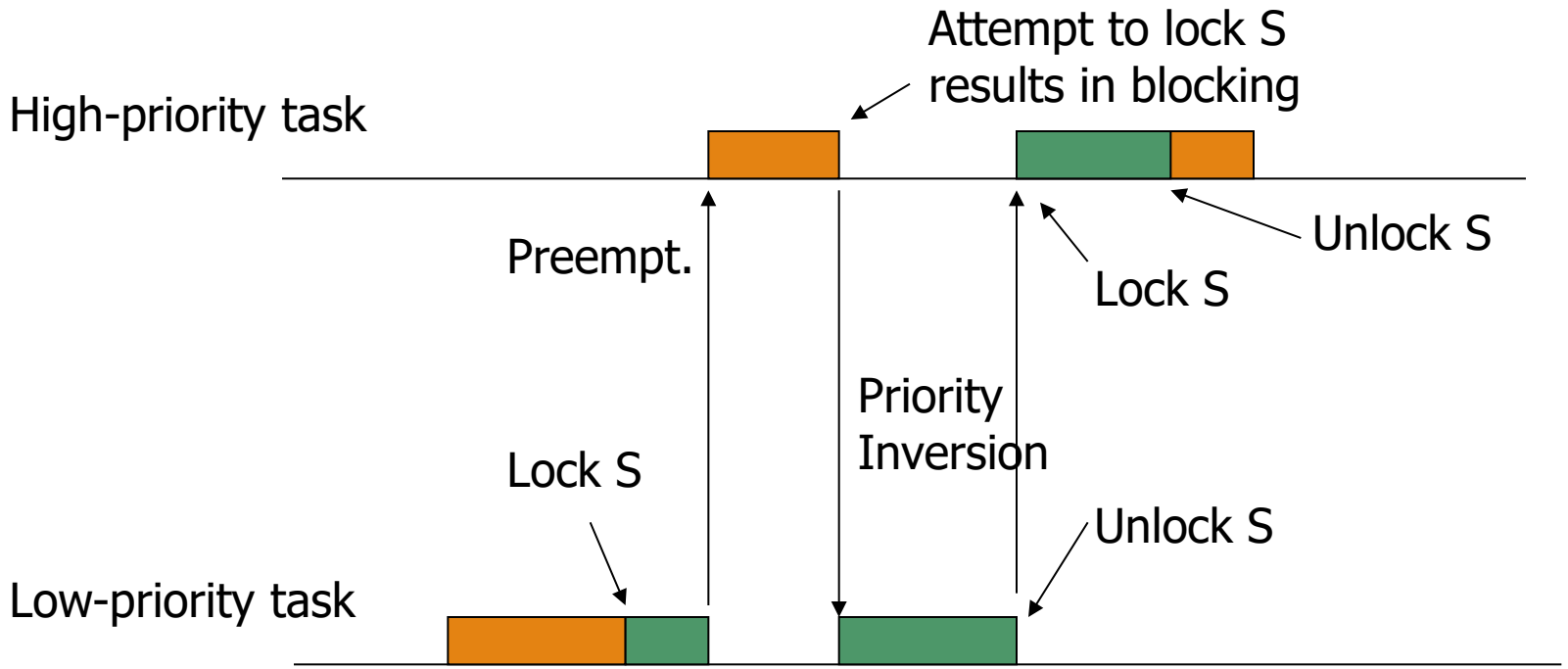
Priority Inversion

Locks and priorities may be at odds. Locking results in priority inversion



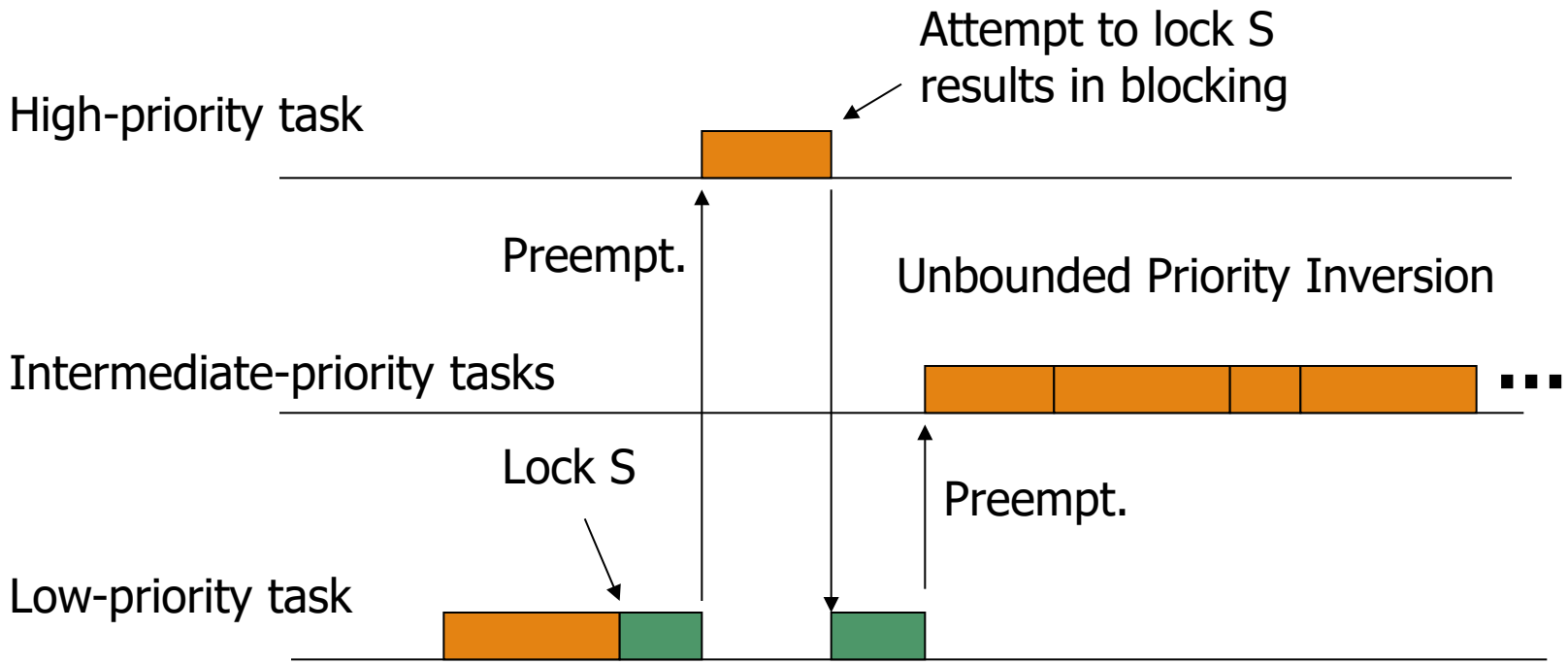
Priority Inversion

How to account for priority inversion?



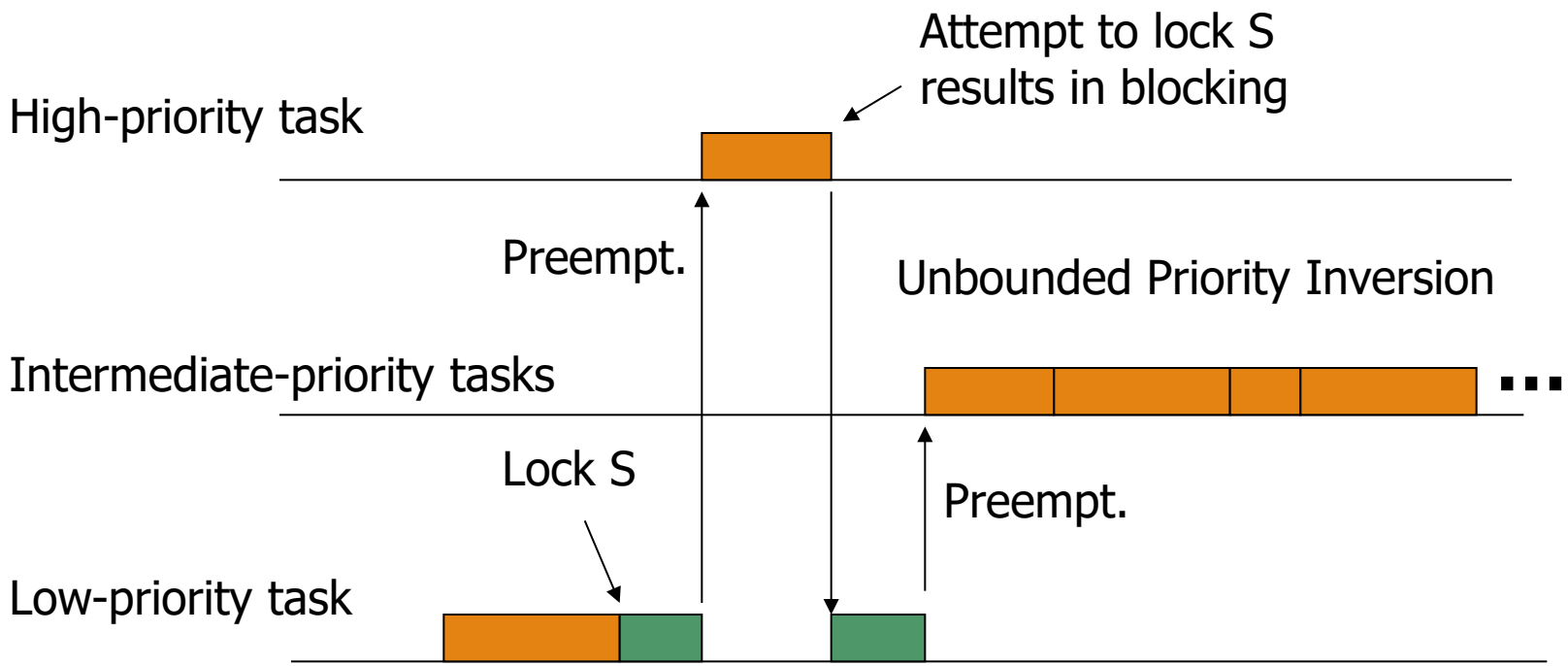
Unbounded Priority Inversion

Consider the case below: a series of intermediate priority tasks is delaying a higher-priority one



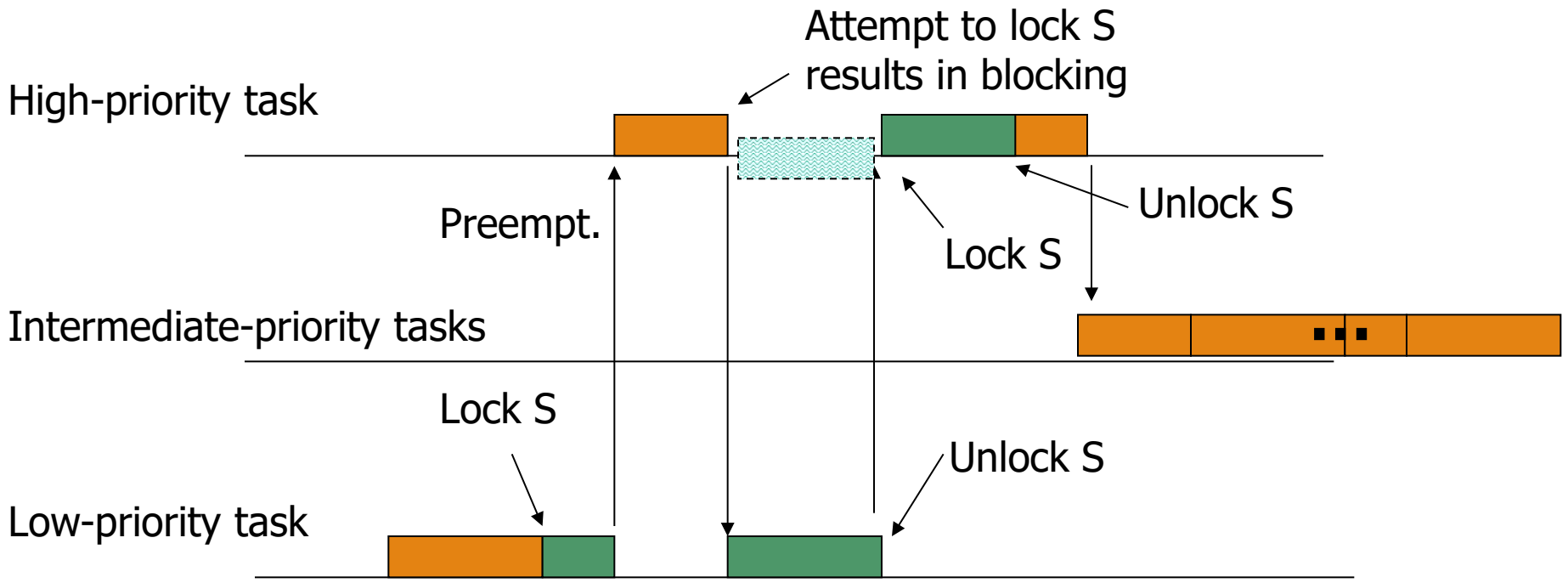
Unbounded Priority Inversion

How to prevent unbounded priority inversion?



Priority Inheritance Protocol

Let a task inherit the priority of any higher-priority task it is blocking



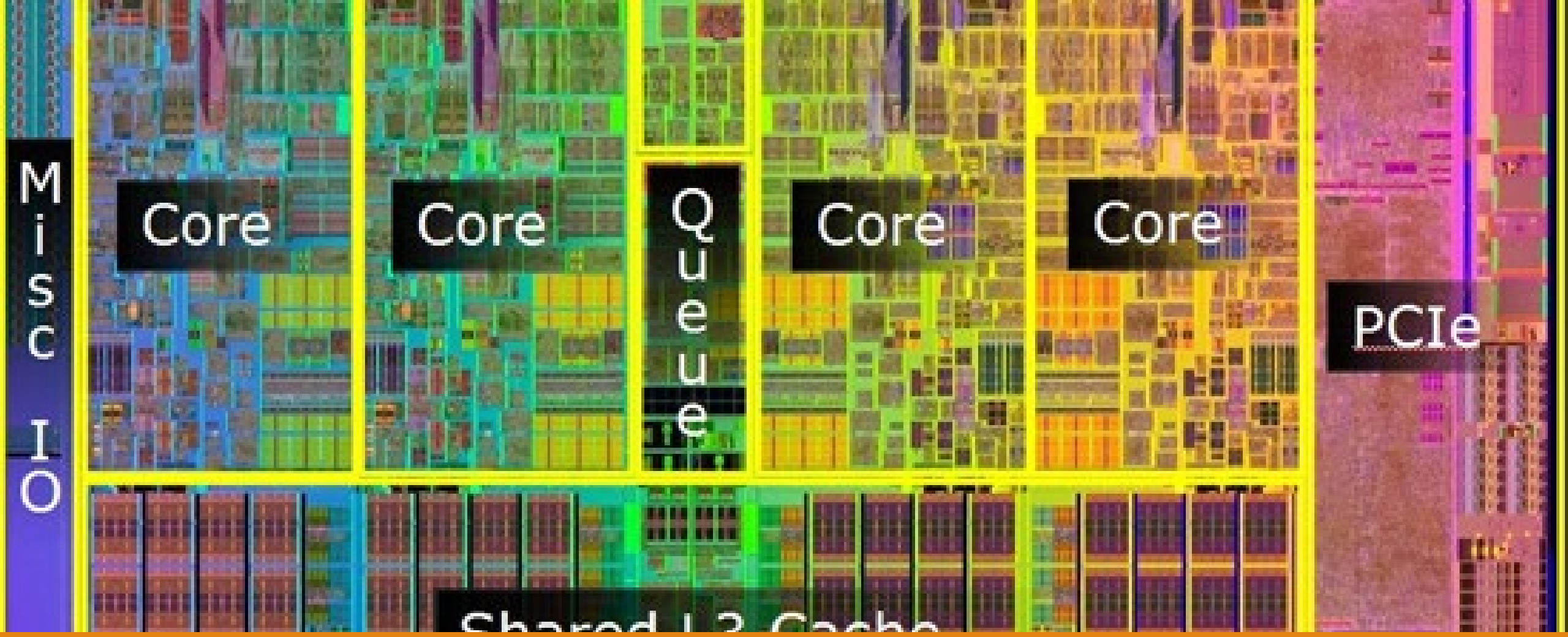
Priority Ceiling Protocol

Definition: The priority ceiling of a semaphore is the highest priority of any task that can lock it

A task that requests a lock R_k is denied if its priority is not higher than the highest priority ceiling of all currently locked semaphores (say it belongs to semaphore R_h)

- The task is said to be blocked by the task holding lock R_h

A task inherits the priority of the top higher-priority task it is blocking



Multicore Scheduling

Question: How do the results generalize to the case with multiple cores are available for task execution?

Multicore Scheduling

Partitioned

- Each core has statically assigned tasks
  - Better isolation
  - Less effective load sharing (idle time on one core cannot be utilized by another)

Global

- A single queue of tasks is dispatched to whatever core is available
  - Better load sharing
  - Poor isolation

Multicore System Utilization

Utilization, expressed below, for a system of *m* cores can be 0 to *m*:

$$U = \sum_i C_i / P_i$$

Utilization Bound for Partitioned EDF

EDF utilization bound of a partitioned multiprocessor?

Schedulable by partitioned EDF if

$$U \leq (m+1)/2$$

(sufficient condition)

Utilization Bound for Partitioned EDF

There cannot be a better bound than:

$$U \leq (m+1)/2$$

Why?

Utilization Bound for Partitioned EDF

There cannot be a better bound than:

$$U \leq (m+1)/2$$

Why?

Consider *m* tasks of utilization (0.5 + very small value) that arrive first, then one more task of utilization = 0.5. Can the last task be scheduled?

Utilization Bound for Partitioned EDF

What if the largest-utilization task (also called the *heaviest* task) has a utilization no more than U_{max}?

Utilization Bound for Partitioned EDF

- What if the largest-utilization task (also called the *heaviest* task) has a utilization no more than U_{max}?
  - Task set is schedulable if:

$$U \leq \frac{\beta m+1}{\beta+1},$$

where

$$\beta = \left\lfloor \frac{1}{U_{max}} \right\rfloor$$

Utilization Bound for Partitioned EDF

$$U \leq \frac{\beta m+1}{\beta+1}, \text{ where } \beta = \left\lfloor \frac{1}{U_{max}} \right\rfloor$$

Why?

Intuition: Imagine all cores are full of tasks of maximum weight (hence, βU_{max} on each core) then a new task arrives that causes the utilization of a core to barely overflow.

$$U_{bound} = 1 + (m-1)\beta U_{\max} > 1 + (m-1)\beta \frac{1}{\beta+1} = \frac{\beta m+1}{\beta+1}$$

Utilization Bound for Global EDF

Consider a case where *m* very small tasks arrive (each of nearly zero utilization), then a task arrives of utilization = 1. Can the last task be scheduled?

Utilization Bound for Global EDF

Consider a case where *m* very small tasks arrive (each of nearly zero utilization), then a task arrives of utilization = 1. Can the last task be scheduled?

Task set is schedulable if $U \leq 1$

Utilization Bound for Global EDF

What if maximum task utilization is U_{max}?

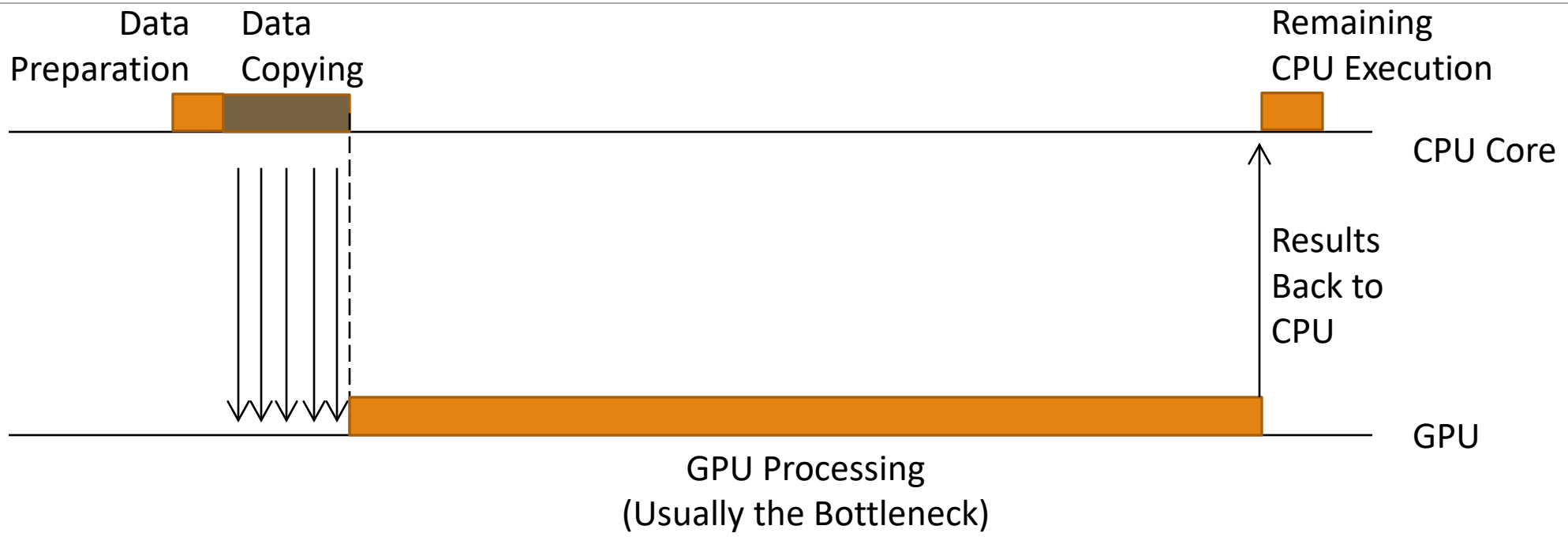
Utilization Bound for Global EDF

- What if maximum task utilization is U_{max}?

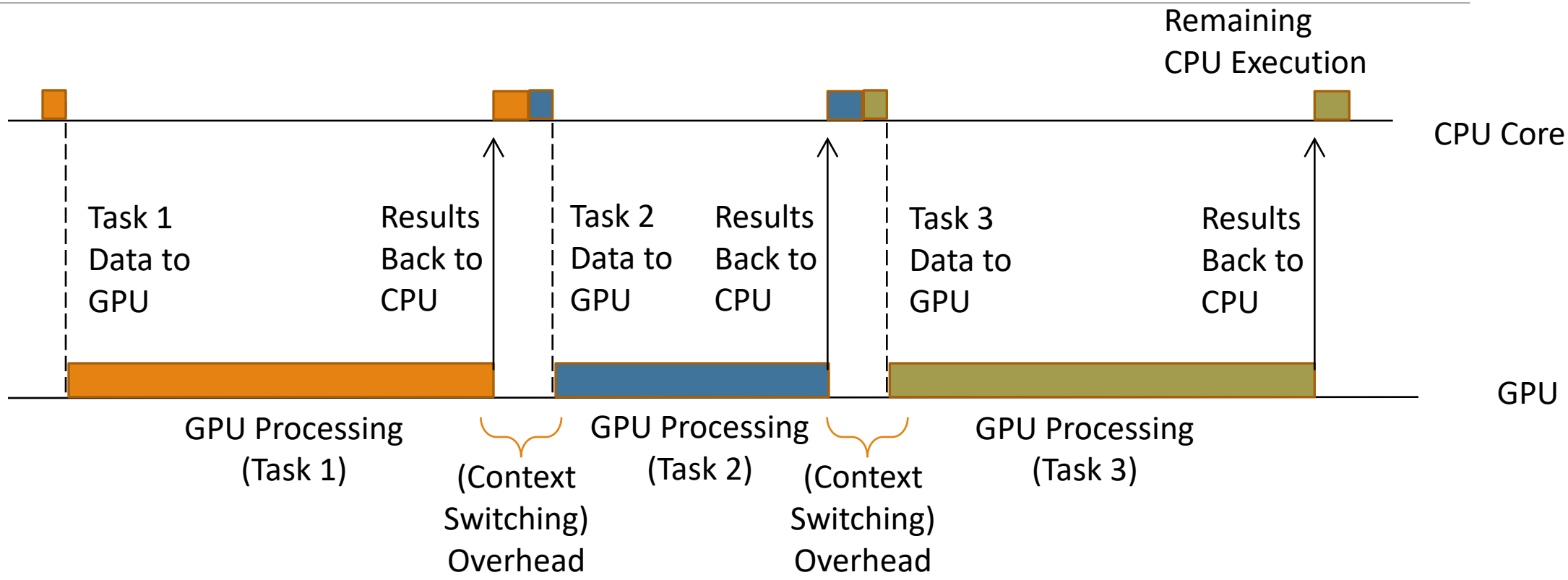
- Task set is schedulable if $U \leq m - (m-1)U_{max}$

Note on Modeling AI Workflows in IoT Contexts

Scheduling Perspective



Scheduling Perspective (Multiple Tasks)



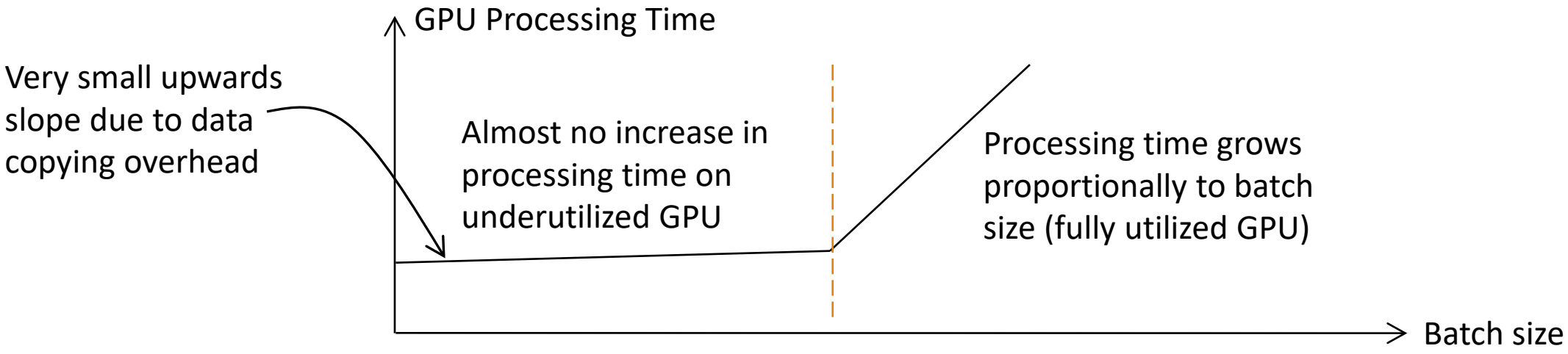
Note: The GPU is similar to a single-core processor with a context switching overhead

Batching

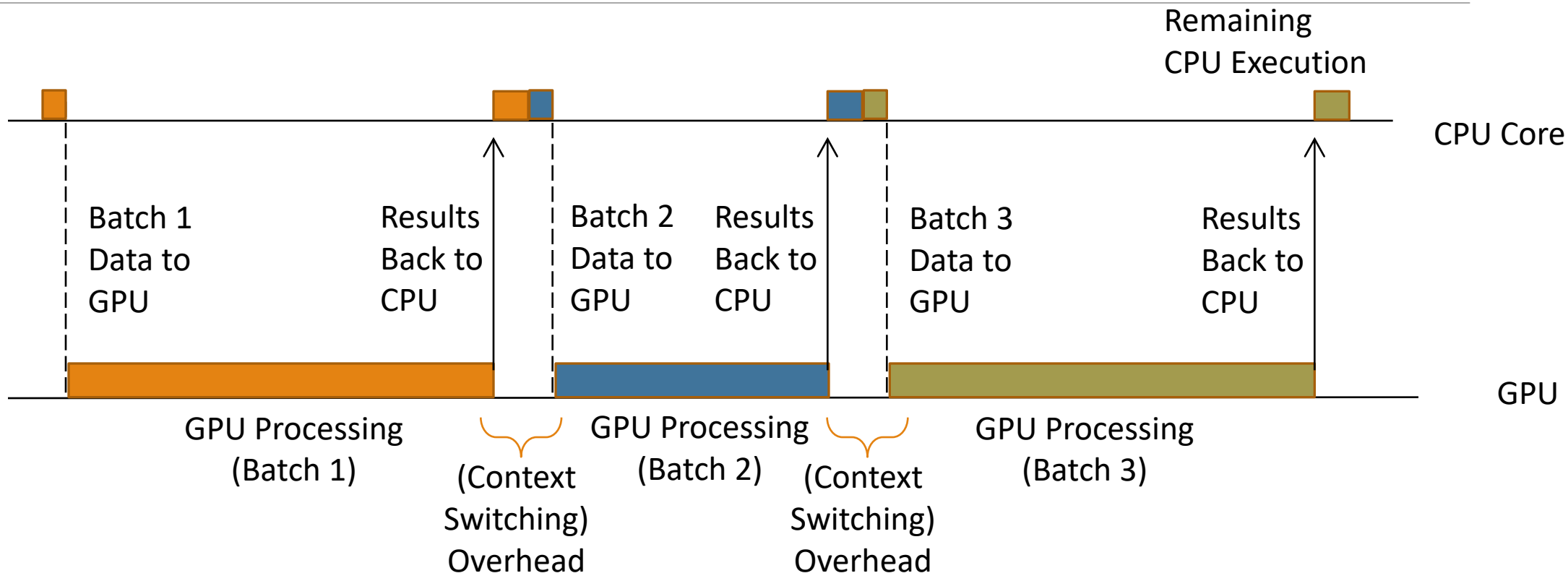
Since the GPU has hundreds of cores, it is often enough to process multiple inputs simultaneously (e.g., frames from multiple security cameras, or point-clouds from multiple LiDARs).

We call such a collection of inputs (simultaneously submitted to the GPU) a *batch*. Thus, GPU task = batch

Rule of Thumb: Increase batch size until the GPU is fully utilized.

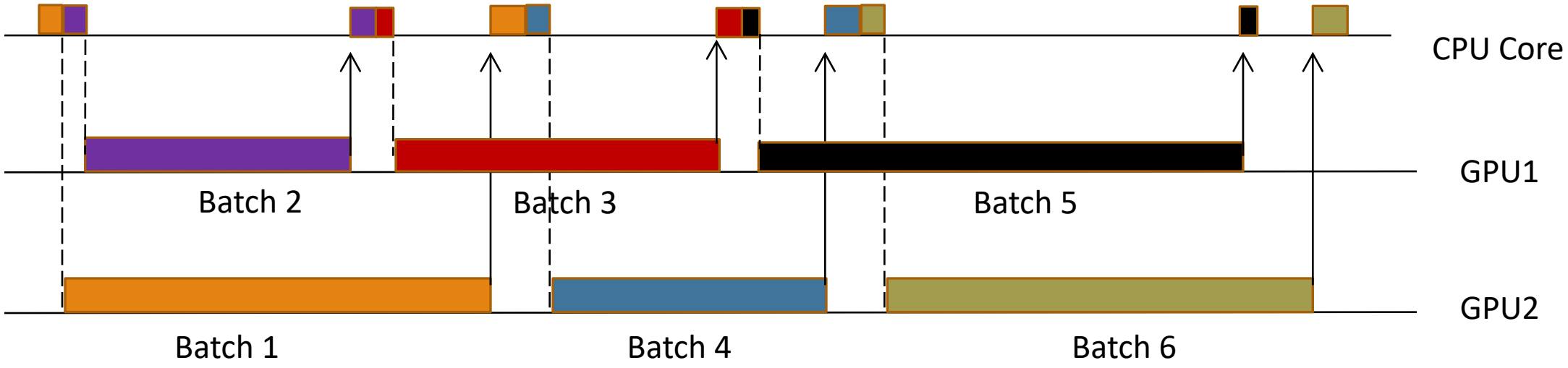


Scheduling Perspective (Multiple Tasks/Batches)
Task = Batch (e.g., multiple images)



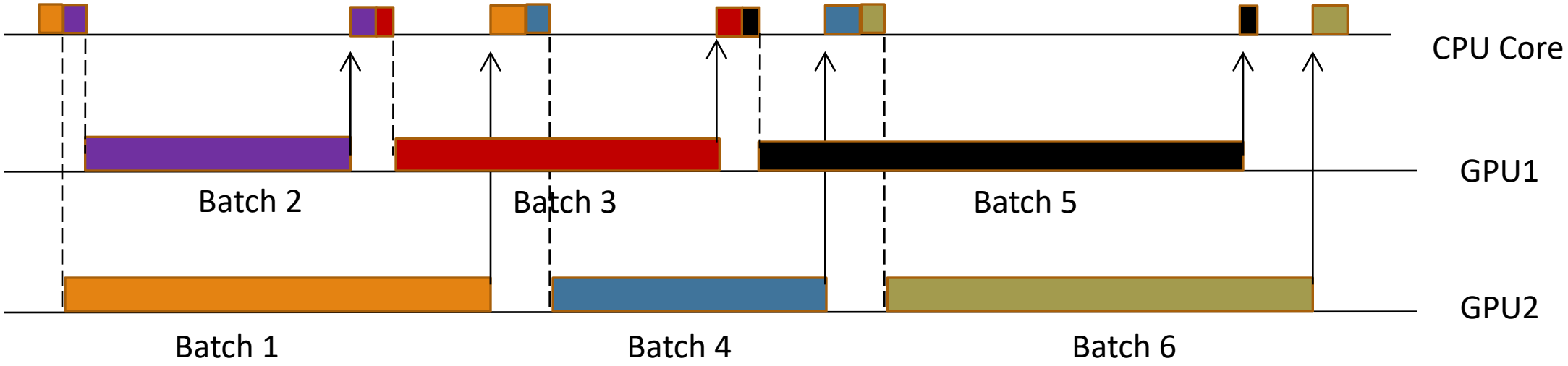
Note: The GPU is similar to a single-core processor with a context switching overhead

Scheduling Perspective (Multiple Batches, Multiple GPUs)



Note: The set of GPUs is similar to a multicore processor with a data-size-dependent context switching overhead

Scheduling Perspective (Multiple Batches, Multiple GPUs)

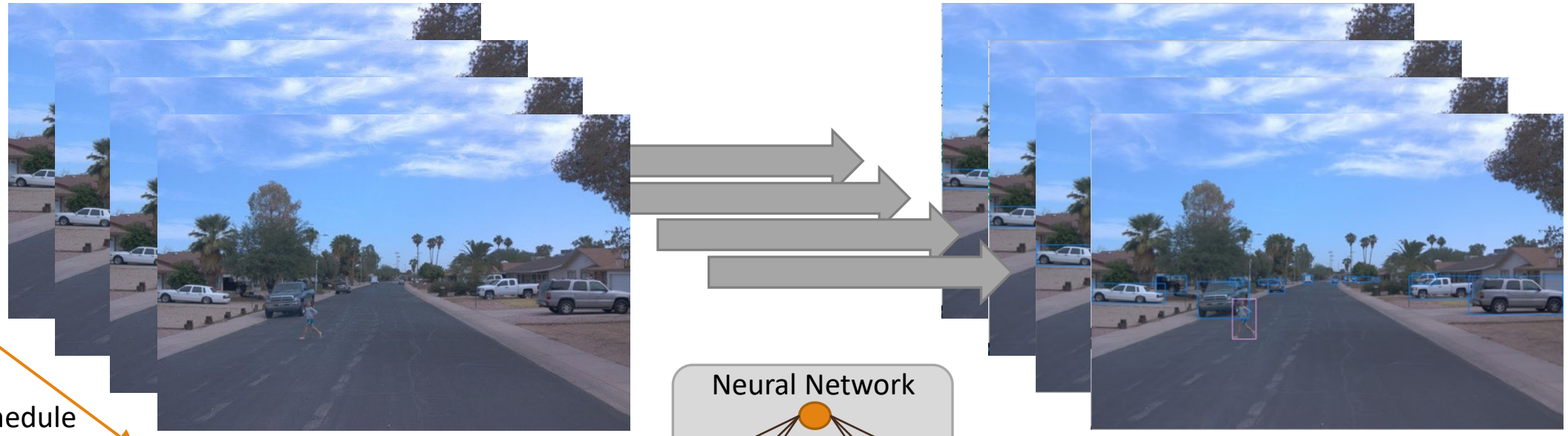


Note: Results of multicore scheduling apply (where “core” = GPU)

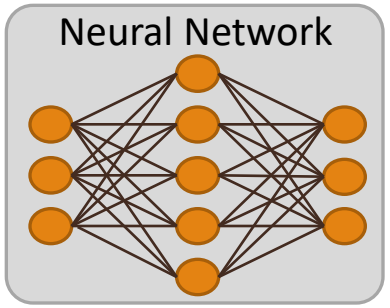
Applications

Prioritizing Attention

Input Frames

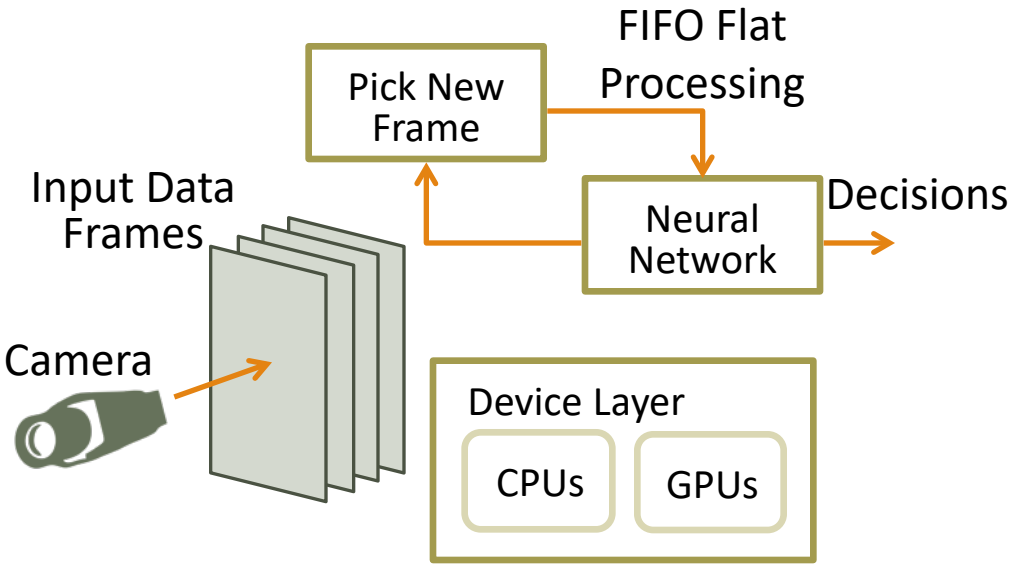


FIFO Schedule

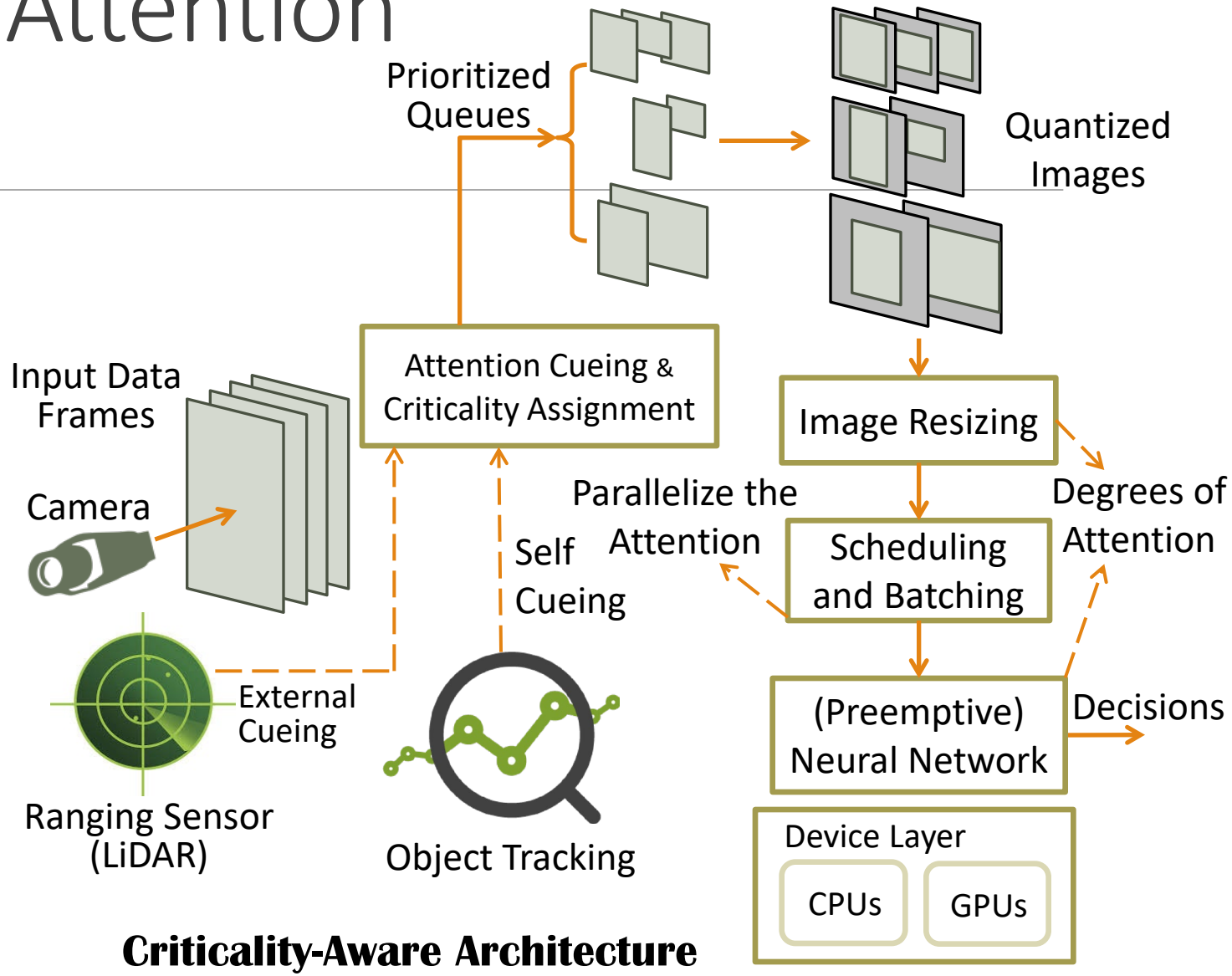


Detection Results

An Architecture for Attention Prioritization



Traditional Architecture

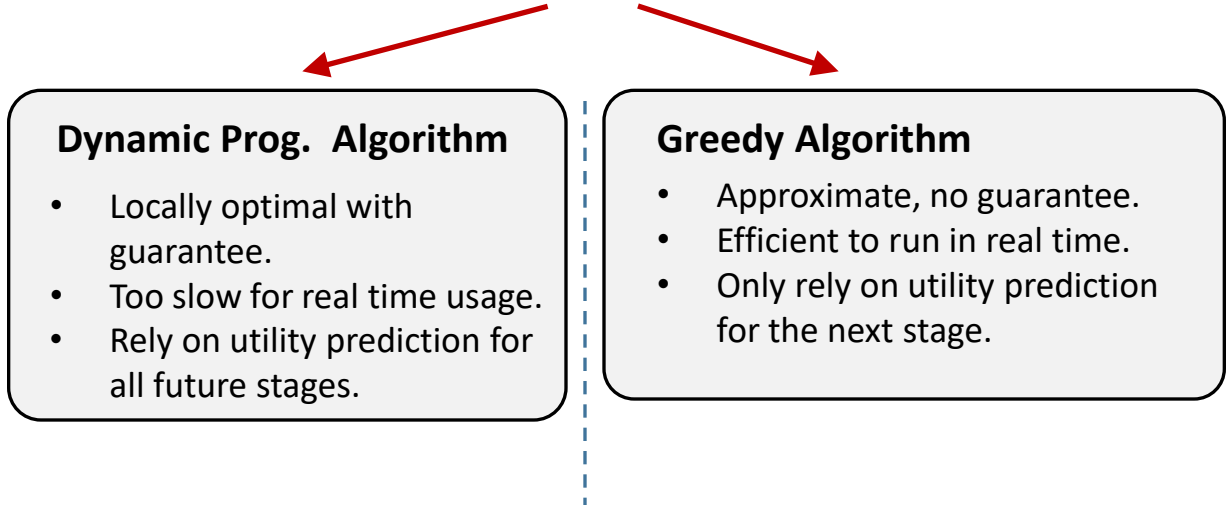


Criticality-Aware Architecture

Attention Scheduling: Mapping Data Processing to Computational/GPU Resources

Find a schedule to **maximize the aggregate system utility** **(predicted accuracy, weighted by criticality)** **subject to deadline and GPU batching constraints**. The schedule decides:

- Task execution order on the GPU (where task = processing of one segment)
- Task execution depth (i.e., number of stages to execute for each task),
- Task batching (which tasks to execute together).



Evaluation

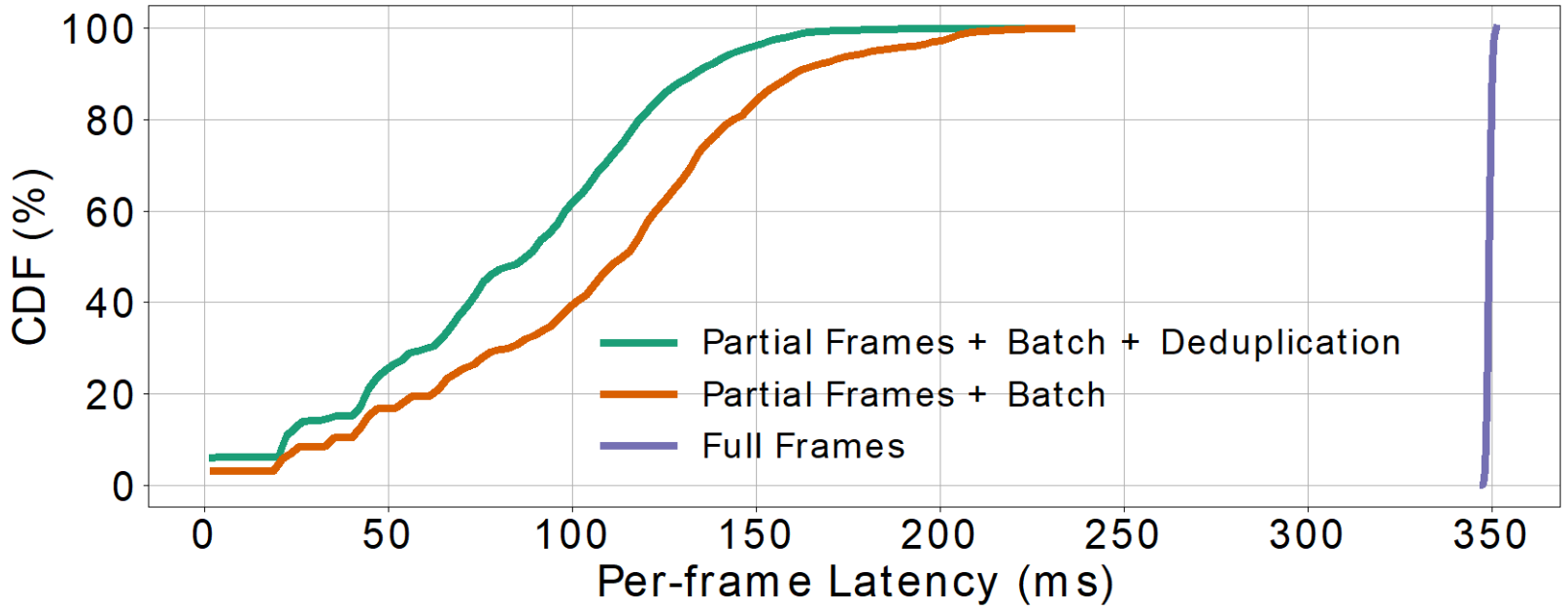
Platform:

- NVIDIA Jetson AGX Xavier SoC.

Dataset:

- Waymo Open Dataset, with 1920 x 1280 images.
- Images originally sampled at 100ms intervals.
- Real-world driving segments collected in diverse geographies and conditions.

Significant reductions in per-frame latency.



*Note: Perfect cueing is assumed above; assumption is relaxed later.