Real-Time Scheduling

Jobs, Servers and Shared Resources · ES/CPSF

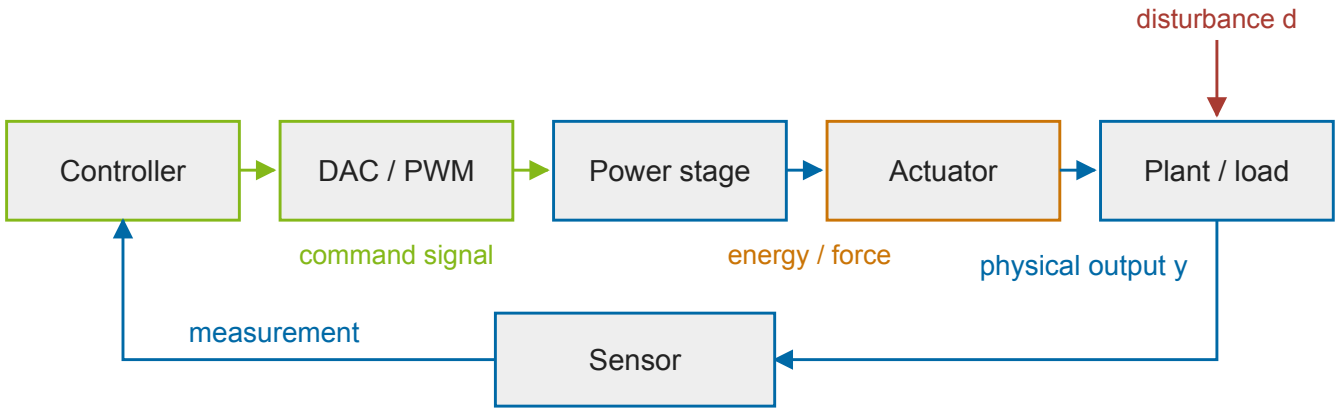
Peter Ulbrich

Course website

Winter Semester 2026/27

Timing the Motor Controller

In Chapter 5, the control task was released every **2 ms** and had **1 ms** to compute and publish a command.



- **The control job needs at most 0.55 ms of CPU time. We must bound what happens when other work shares that CPU.**

Workload and Analysis Assumptions

A timing claim applies to a specific workload, platform and scheduling policy.

Initially	Consequence
One processor	The available CPU time in an interval is its length.
Independent jobs	Releases are given by the workload. Jobs need no lock and wait for no input after release.
Given execution-time bounds	Each job can require its full bound.
Ideal dispatch	Scheduler and switch costs are omitted in the initial model.

- **Make each assumption explicit before using a test.**

Learning Objectives

- Construct and check schedules for independent jobs.
- Use processor demand and response-time analysis for recurrent tasks.
- Explain how a server controls interference from event-driven work.
- Trace a conflicting access, use a mutex and identify a deadlock.
- Explain how a resource protocol gives a blocking bound for the timing analysis.

- **Follow a timing requirement from the schedule to the proof obligation.**

Release Times, Preemption and Priority

What changes when we move from a one-off job list to a real-time controller?

Question	Cases we will use
When does work become ready?	One common release, different job-specific releases, or recurrent releases with a period or minimum separation.
Can another job take over?	Nonpreemptive jobs keep the CPU until they finish in this model. Preemptive jobs may be interrupted and resumed.
How is ready work ranked?	Fixed **task** priorities in RM and DM. **Job** priorities based on absolute deadlines in EDF.

- **Specify the release model, preemption rule and priority rule separately.**

Chapter Map: Scheduling Policies

6.1 Scheduling Policies

RM, DM and EDF on the same workloads

6.2 EDF Schedulability

Utilization, short deadlines and interval demand

6.3 Fixed-Priority Analysis

Critical instant and response-time iteration

6.4 Irregular Events and Servers

Service budgets and replenishment

6.5 Concurrency and Shared Resources

Mutual exclusion, deadlock and blocking

From Tasks to Recurring Jobs

For task τ_i and its kth job $J_{i,k}$, recall the three timing parameters.

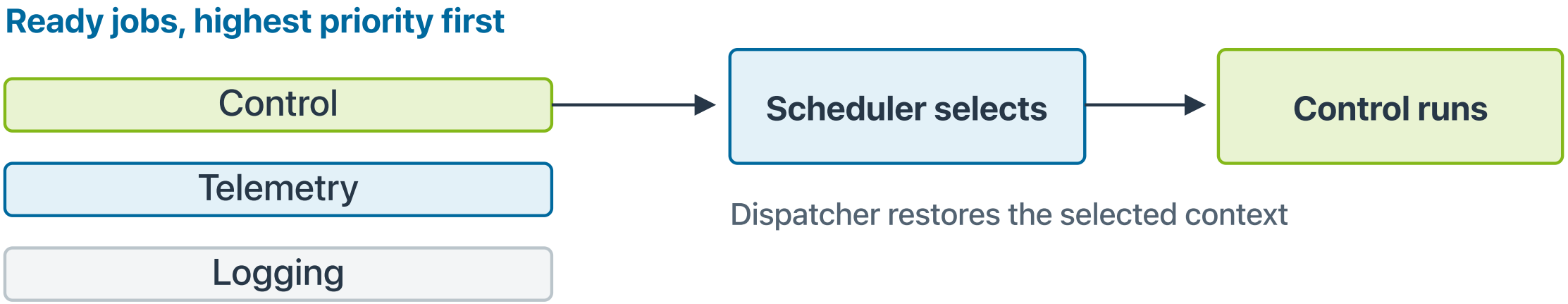
Parameter	Meaning
C_i	Maximum CPU time of each job
T_i	Exact period for periodic releases, minimum separation for sporadic releases
D_i	Relative deadline, so $d_{i,k} = r_{i,k} + D_i$

Implicit deadline: $D_i = T_i$. Constrained deadline: $D_i \leq T_i$.

- **A sporadic guarantee covers every release sequence consistent with its minimum separation.**

Fixed Priorities on a Ready Queue

The OSEK tasks in Chapter 5 had configured priorities. When their jobs become ready, the scheduler uses that order.



- **A ready job uses its task's assigned priority P_i. The scheduler selects the highest-priority ready job.**

Rate Monotonic and Deadline Monotonic

Two tasks have different periods and relative deadlines. Which task receives higher priority?

Task	Period T_i	Relative deadline D_i
A	4 ms	4 ms
B	6 ms	2 ms

Assignment	Rule	Higher priority here
Rate Monotonic (RM)	Shorter period first	A
Deadline Monotonic (DM)	Shorter relative deadline first	B

- **With implicit deadlines, $D_i = T_i$, the two rankings coincide.**

The Priority Rule Can Decide Feasibility

A and B both release a job at time 0. A needs 1.5 ms of CPU time and B needs 1.0 ms. Recall $T_A = 4$, $D_A = 4$, $T_B = 6$ and $D_B = 2$ ms.

Assignment	CPU order after release	First response times
RM	A, then B	$R_A = 1.5$ ms and $R_B = 2.5$ ms. B misses.
DM	B, then A	$R_B = 1.0$ ms and $R_A = 2.5$ ms. Both meet their deadlines.

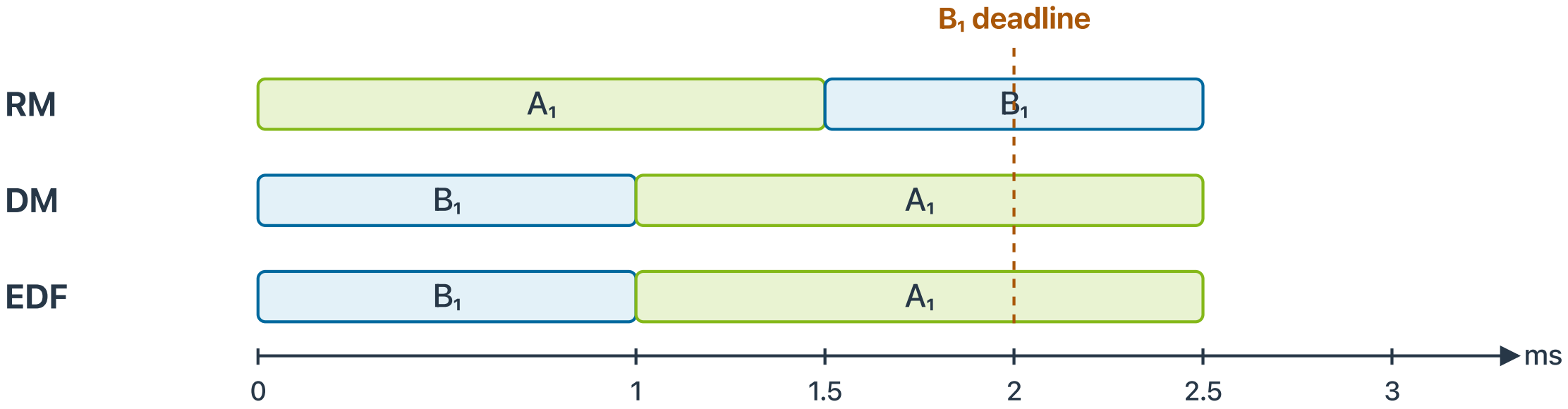
- **When a longer-period task has a tighter deadline, DM can give it the higher priority.**

Independent tasks on one preemptive CPU, no overhead. For this example the synchronous releases are the critical instant.

EDF on the Same Job Releases

How would **Earliest Deadline First (EDF)** schedule the same releases of A and B?

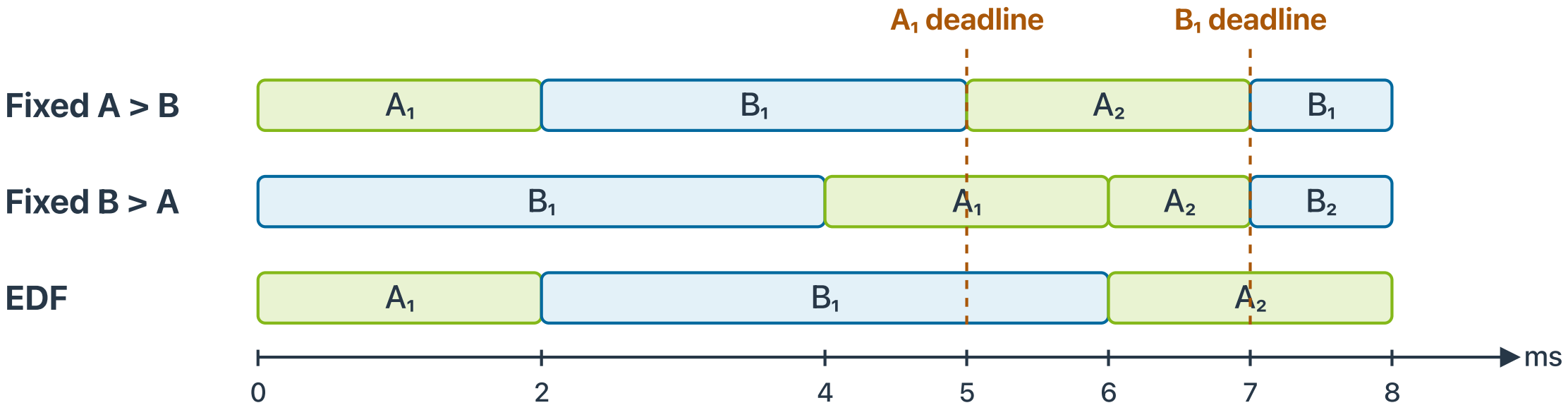
A_k and B_k denote the kth jobs of tasks A and B.



- **EDF chooses the ready job with the earliest absolute deadline. Here that is B_1, due at 2 ms.**

When Both Strict Fixed Orders Miss

Two tasks release recurrent jobs with implicit deadlines. For (T, C, D) in ms, $A = (5, 2, 5)$ and $B = (7, 4, 7)$.

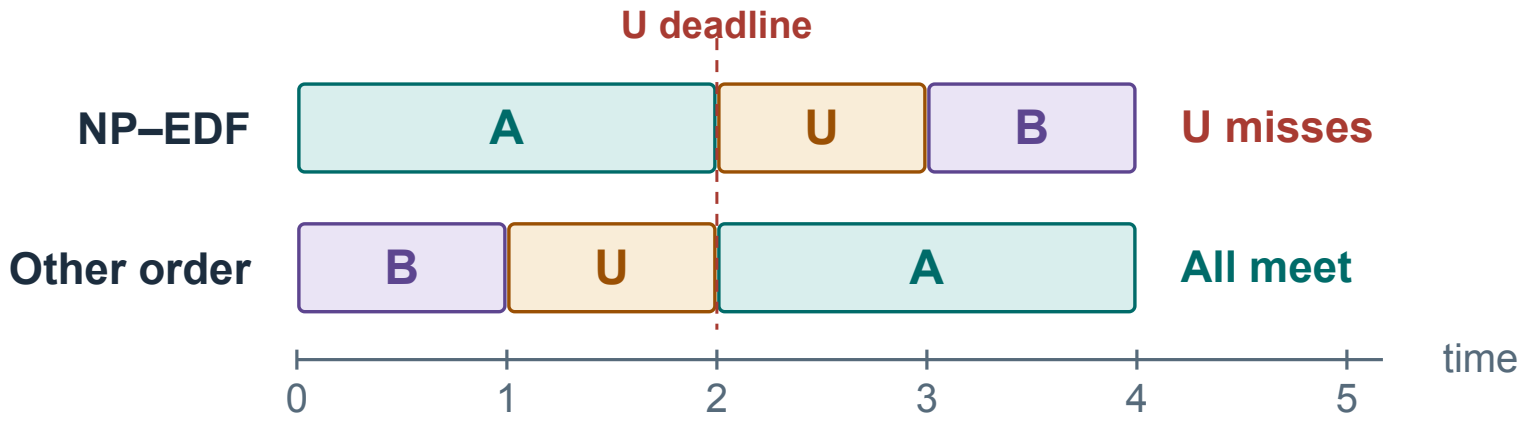


- **At 5 ms, EDF lets B finish first: its pending job is due at 7 ms, the newly released A job at 10 ms.**

Distinct task priorities in the fixed-priority cases. Synchronous releases on one fully preemptive CPU. A guarantee under EDF follows in the next section.

Nonpreemptive EDF Can Choose the Wrong Job

Each job is given as (r, C, d) (release, CPU demand, absolute deadline). $A = (0, 2, 4)$, $B = (0, 1, 6)$ and urgent job $U = (1, 1, 2)$.



- **Both schedules keep the CPU busy. The earlier deadline at time 0 leads to a later miss when the job cannot be preempted.**

Chapter Map: EDF Schedulability

6.1 Scheduling Policies

RM, DM and EDF on the same workloads

6.2 EDF Schedulability

Utilization, short deadlines and interval demand

6.3 Fixed-Priority Analysis

Critical instant and response-time iteration

6.4 Irregular Events and Servers

Service budgets and replenishment

6.5 Concurrency and Shared Resources

Mutual exclusion, deadlock and blocking

Implicit Deadlines: An Exact EDF Test

Independent recurrent tasks with $D_i = T_i$ on one preemptive EDF CPU.

$$U = \sum_i U_i \leq 1 \quad \Longleftrightarrow \quad \text{all deadlines are guaranteed.}$$

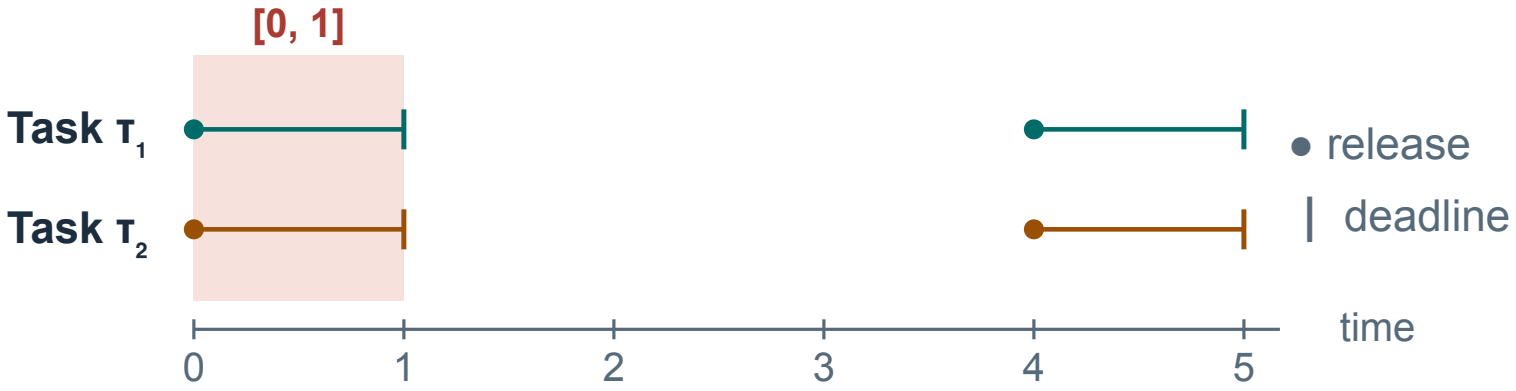
Why	CPU time
Long-run necessity	Jobs released every T_i require a CPU share of $U_i = C_i/T_i$.
Interval sufficiency	Every job with both endpoints inside an interval occupies T_i of its task's release-to-deadline span. These spans do not overlap for the same task.

- **With implicit deadlines, the utilization bound also controls demand in each interval.**

For tasks A and B, $U = 2/5 + 4/7 = 34/35 < 1$. Liu and Layland (1973) for periodic tasks.

Short Deadlines Can Overload a Short Interval

Both tasks have $(C_i, T_i, D_i) = (1, 4, 1)$. Their combined utilization is $U = 0.5$.



- **At a simultaneous release, $[0, 1]$ needs 2 units of CPU time and offers 1.**

The EDF Interval Test

On one preemptive CPU, consider a finite set of independent jobs J_k. Each has release r_k, CPU demand C_k and absolute deadline d_k.

A job with $r_k \geq a$ and $d_k \leq b$ must receive all its CPU time in $[a, b]$.

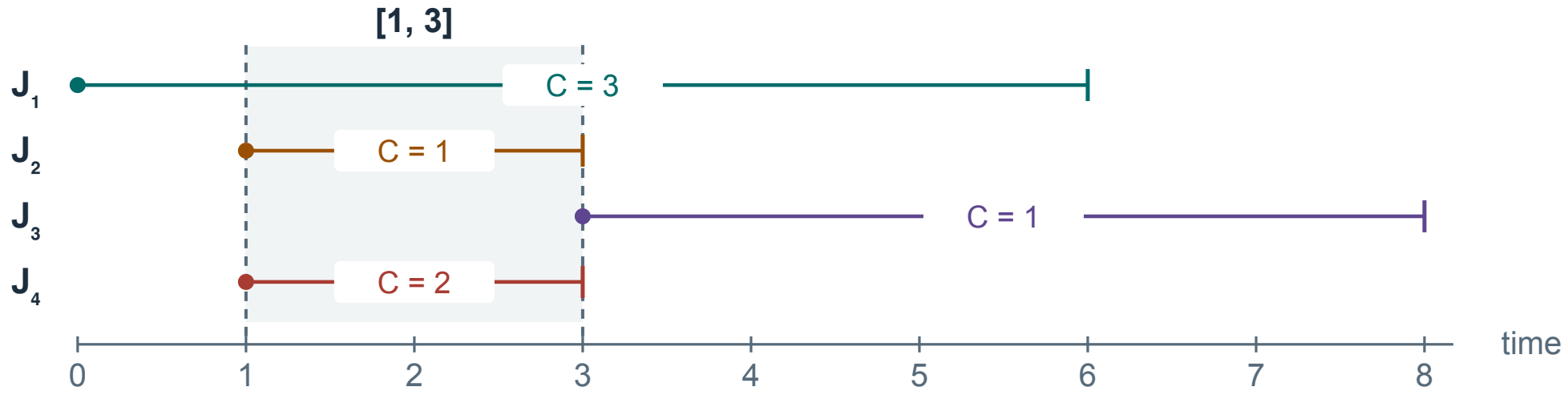
$$\sum_{k:\, r_k \geq a,\, d_k \leq b} C_k \;\leq\; b - a$$

Check pairs with a a release, b a deadline, and $a \leq b$.

- **A failing interval proves overload. For this model, EDF meets all deadlines exactly when every interval passes.**

Find the Overloaded Interval

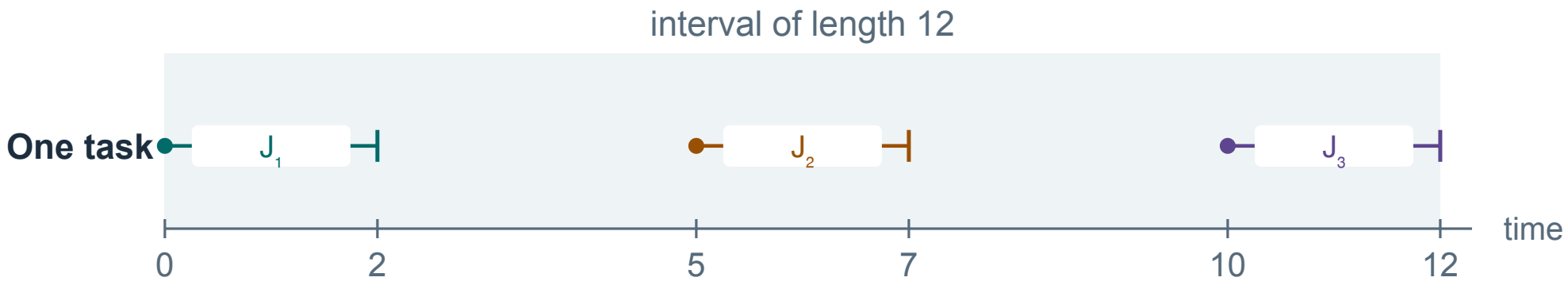
Four one-off jobs have the release and deadline windows shown below. Is the total demand of 7 enough to detect an overload?



- J_2 and J_4 need 3 units of CPU time in the 2 units available from 1 to 3.

Demand Bound: Count Complete Job Windows

One sporadic task has $T_i = 5$, $D_i = 2$ and $C_i = 1$. Here J_k abbreviates $J_{i,k}$ for this task.



The first job needs D_i from release to deadline. Each additional complete window needs another T_i units of interval length.

- **In length 12 there is space for three complete job windows. Their demand is 3.**

EDF Tests Processor Demand at Every Length

Maximum CPU demand of one sporadic task in an interval of length ℓ:

$$\mathrm{dbf}_i(\ell) = C_i \max\left(0, \left\lfloor \frac{\ell - D_i}{T_i} \right\rfloor + 1\right).$$

- **Preemptive EDF guarantees the deadlines exactly when $\sum_i \mathrm{dbf}_i(\ell) \leq \ell$ for every $\ell > 0$.**

For the two tasks on S19, $\mathrm{dbf}_1(1) = \mathrm{dbf}_2(1) = 1$. The length-1 interval fails.

Independent sporadic tasks with $0 < D_i \leq T_i$ on one fully preemptive CPU. Baruah, Mok and Rosier (1990), doi:10.1109/REAL.1990.128746.

Self Check: Same Utilization, Different Windows

Two independent sporadic tasks each have $C_i = 1$ and $T_i = 4$.

Case	Relative deadlines	Which test answers the question?
A	$D_1 = D_2 = 4$	Does $U = 0.5$ establish a guarantee?
B	$D_1 = D_2 = 1$	Which interval shows a deadline miss is possible?

- **Under fixed priorities, we now ask how much higher-priority work can delay one job.**

Chapter Map: Fixed-Priority Analysis

6.1 Scheduling Policies

RM, DM and EDF on the same workloads

6.2 EDF Schedulability

Utilization, short deadlines and interval demand

6.3 Fixed-Priority Analysis

Critical instant and response-time iteration

6.4 Irregular Events and Servers

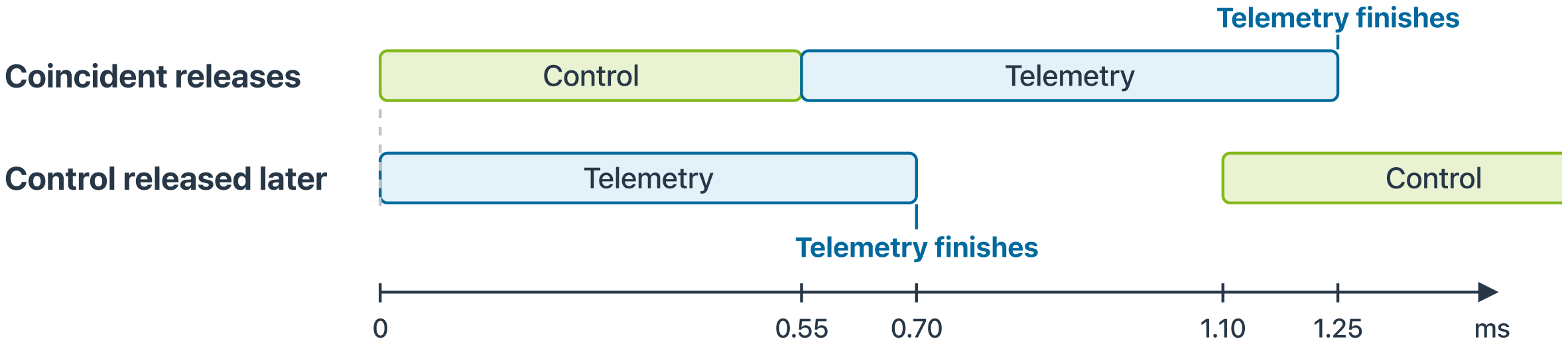
Service budgets and replenishment

6.5 Concurrency and Shared Resources

Mutual exclusion, deadlock and blocking

A Critical Instant for Fixed Priorities

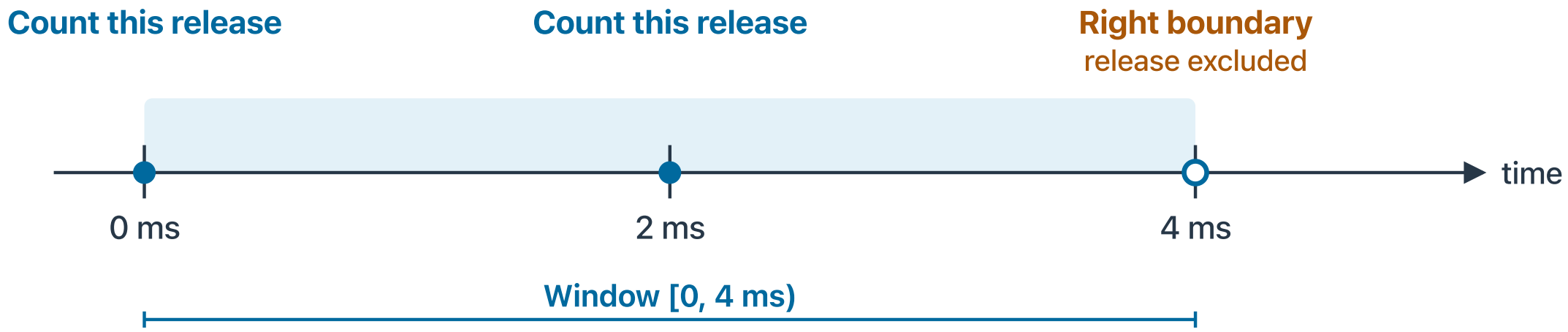
Telemetry is ready. How does the timing of a higher-priority control release affect it?



- **In the independent constrained-deadline model, analyse a simultaneous release with higher-priority tasks and their earliest subsequent releases.**

Counting Higher-Priority Work

While a job of task i is pending, higher-priority job releases can take CPU time away from it.



For higher-priority task j, at most $\lceil w/T_j \rceil$ jobs are released in the synchronous interval $[0, w)$.

- **Multiply the job count by C_j to bound task j's CPU interference.**

Response Time Is a Fixed Point

A longer response time can admit more interference. Compute the response time and the interference together.

$$w_i^{(0)} = C_i, \qquad w_i^{(q+1)} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^{(q)}}{T_j} \right\rceil C_j.$$

- $w_i^{(q)}$ is the response window for task i at iteration $q = 0, 1, \ldots$.
- j ranges over $hp(i)$, the indices of tasks with strictly higher priority than i.
- Stop at the first unchanged value. It gives R_i in the critical-instant model.
- If the estimate exceeds D_i, the test fails.

▶ **Compare the worst-case response time R_i with the relative deadline D_i.**

Worked Example: Control, Telemetry and Logging

The motor controller shares its CPU with two lower-priority tasks. Verify the logging deadline.

Task, highest priority first	T_i	C_i	D_i
Control	2 ms	0.55 ms	1.0 ms
Telemetry	5 ms	0.70 ms	2.1 ms
Logging	10 ms	0.80 ms	4.0 ms

Logging iteration in ms: $0.80 \to 2.05 \to 2.60 \to 2.60$.

- **Logging admits a second release of control. Its response time is 2.60 ms.**

A Quick Sufficient Test for RM

A utilization check for n tasks can prove schedulability without iterating the response times.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{1/n} - 1\right).$$

Implicit-deadline example under RM	Result
$(T_i, C_i) = (4, 1), (5, 2), (20, 5)$	$U = 0.90$ exceeds the three-task bound of about 0.78.
Exact response times in the synchronous model	$(1, 3, 15)$. All three deadlines are met.

- **When the sufficient bound fails, use a more precise test.**

An Exact RM Test for Harmonic Periods

Sort the tasks by period. The periods are **harmonic** if each period is an integer multiple of the preceding one. For example, $2, 6, 12$ are harmonic.

- **For independent periodic tasks with $D_i = T_i$ on one preemptive CPU, RM meets all deadlines if and only if $U \le 1$ when the periods are harmonic.**

At the critical instant, each higher-priority task j releases exactly T_i/T_j jobs in $[0, T_i)$. The CPU demand for the first job of task i and its higher-priority work is

$$C_i + \sum_{j \in hp(i)} \frac{T_i}{T_j} C_j = T_i \left(U_i + \sum_{j \in hp(i)} U_j \right) \le T_i \qquad \text{if } U \le 1.$$

Use a fixed tie-break for equal periods. The analysis assumes zero scheduling overhead.

Harmonic RM at Full Utilization

The tasks release together at time 0, with $D_i = T_i$. Time is in ms.

Task in RM order	Period T_i	CPU demand C_i	Jobs released in $[0, 12)$	CPU time in that window
τ_1	2	1	6	6
τ_2	6	2	2	4
τ_3	12	2	1	2

- $U = 1/2 + 2/6 + 2/12 = 1$. **The periods form a harmonic chain, so RM guarantees all three deadlines.**

Self check. Would the exact harmonic test still apply to periods $4, 6, 12$?

Chapter Map: Irregular Events and Servers

6.1 Scheduling Policies

RM, DM and EDF on the same workloads

6.2 EDF Schedulability

Utilization, short deadlines and interval demand

6.3 Fixed-Priority Analysis

Critical instant and response-time iteration

6.4 Irregular Events and Servers

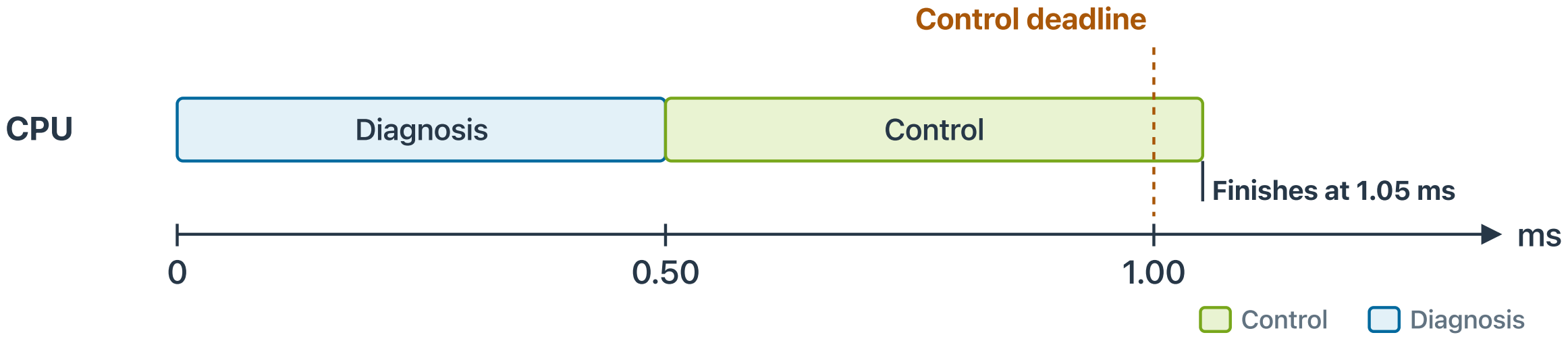
Service budgets and replenishment

6.5 Concurrency and Shared Resources

Mutual exclusion, deadlock and blocking

A Diagnostic Event Delays Control

A motor-control job and a high-priority diagnostic job are released together at $t = 0$.



- **One diagnostic job can consume more than the controller's 0.45 ms of timing margin.**

Arrival Bounds and Admission

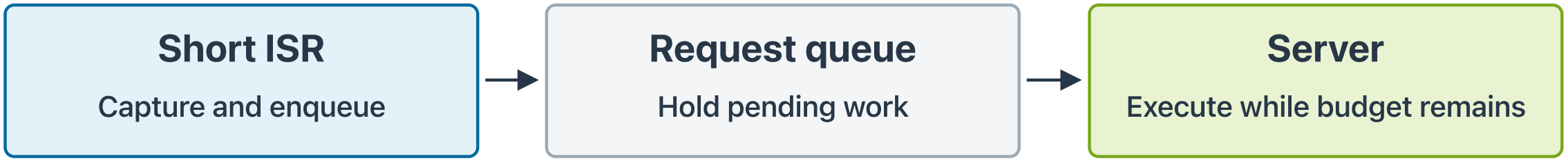
What can we promise when diagnostics arrive at irregular times?

Available assumption	Scheduling implication
A guaranteed minimum gap T_i and known C_i, D_i	Include the sporadic task in the analysis at design time.
An extra job requests a guaranteed deadline at runtime	Admit the job if its deadline and existing commitments can be met.
Service requests need responsive best-effort handling	Queue them for budgeted service and specify what happens on overload.

- **The arrival model bounds incoming demand. Admission determines which extra deadlines the system promises.**

Budgeted Diagnostic Service

A diagnostic interrupt can arrive at any point in the controller's execution.

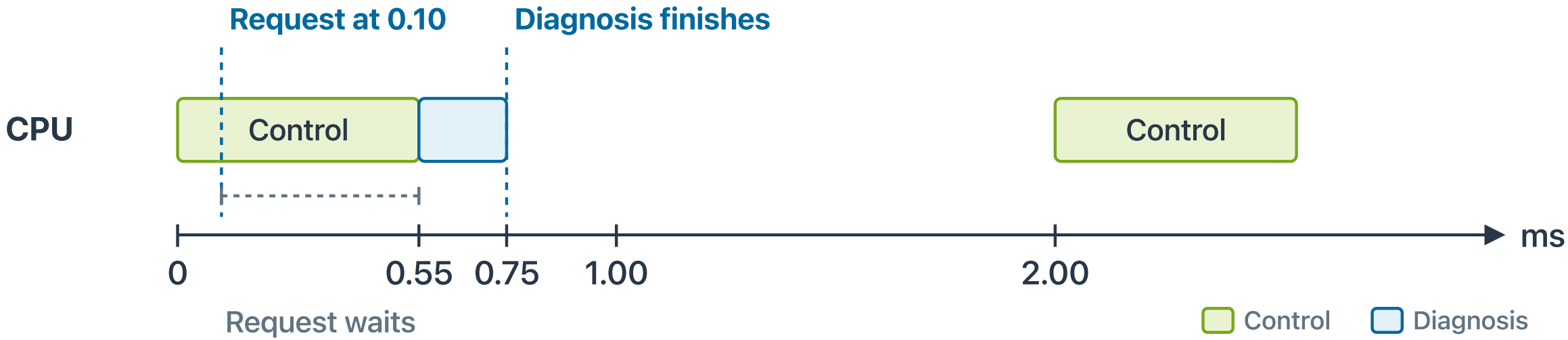


Parameter	Meaning
Q_s	Budget of server s, measured in CPU time
T_s	Period of polling or fixed replenishment, or replenishment delay for the Sporadic Server

- **Keep IRQ work short and account for it separately. The server limits scheduled diagnostic CPU work.**

Background Service

A diagnostic request arrives at 0.10 ms and needs 0.20 ms of CPU time. The control job released at 0 needs 0.55 ms.



- **Background service uses otherwise idle CPU time. Its delay depends on how long higher-priority work keeps the CPU busy.**

Polling Server

Can a periodic service limit the impact of diagnostics? Use $Q_s = 0.25$ ms and $T_s = 2$ ms at a priority above control.



- **An empty poll gives up the budget. A request just after that poll waits for the next server release.**

Deferrable Server

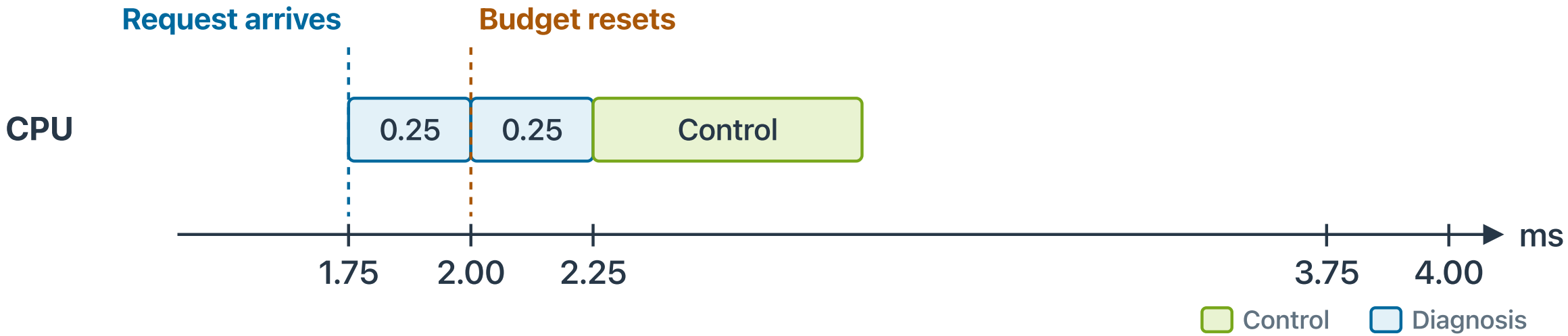
The polling server gave up its budget just before the request arrived. Keep $Q_s = 0.25$ ms available within each 2 ms period.



- **Retain unused budget until the end of the period. A ready request can use it immediately at the server's priority.**

Deferrable Server Interference

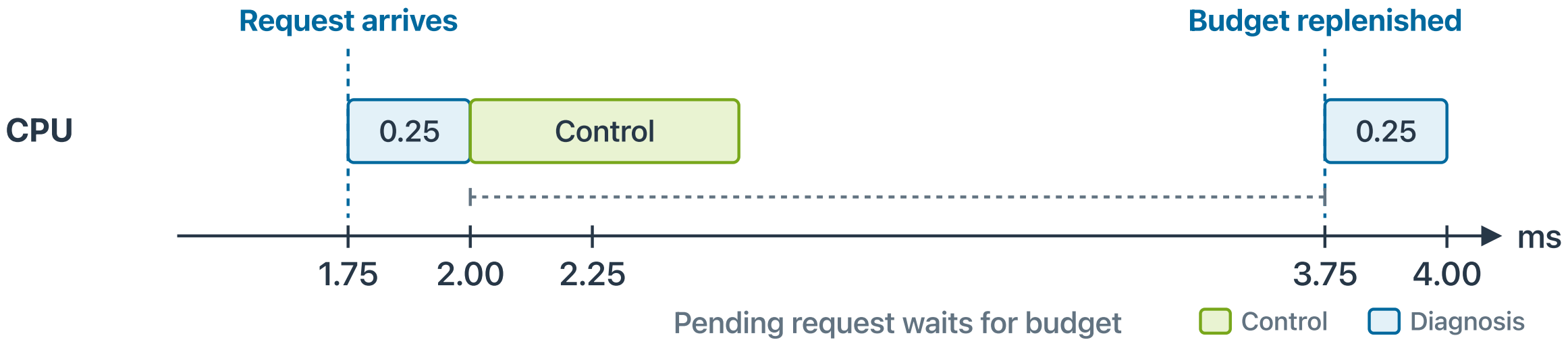
A larger request needs 0.50 ms and arrives at 1.75 ms. The server has used none of its initial 0.25 ms budget.



- **A short window straddling a replenishment can contain two full budgets.**

Sporadic Server Replenishment

Can the server keep unused budget and avoid refilling it at a fixed boundary?



- **Link replenishment to budget use. In this trace the spent 0.25 ms becomes available at $1.75 + 2.00 = 3.75$ ms.**

Four Service Policies on One Trace

Compare the same 0.20 ms request released at 0.10 ms under each policy.

Service	When the queue is empty	Diagnosis starts	First control job finishes
Background	No CPU work	0.55 ms	0.55 ms
Polling Server	Ends the current poll	2.00 ms	0.55 ms
Deferrable Server	Keeps budget within the period	0.10 ms	0.75 ms
Sporadic Server	Keeps available budget	0.10 ms	0.75 ms

- **The first request can see the same service from two servers while their later interference differs.**

The Controller's Bound with a Server

The controller needs 0.55 ms of CPU time by its 1.00 ms deadline. Interrupt and scheduler overhead uses at most 0.05 ms in that window.

$$\underbrace{0.55\,\mathrm{ms}}_{\text{control}} + \underbrace{0.25\,\mathrm{ms}}_{\text{server}} + \underbrace{0.05\,\mathrm{ms}}_{\text{overhead}} = 0.85\,\mathrm{ms}$$

The calculation assumes that the selected server contributes at most 0.25 ms within the controller's 1.00 ms response window.

- **The server's interference bound enters the same response-time argument as higher-priority task execution.**

CPU Time and Shared State

A sample record stores a speed value and a timestamp. One software task publishes it, and control reads it.

- How much CPU time can other work take from control in the next millisecond?
- What can control read if it preempts the writer between two stores?

- **The CPU budget bounds competition for execution time. A shared record also needs a rule for concurrent access.**

Chapter Map: Concurrency and Shared Resources

6.1 Scheduling Policies

RM, DM and EDF on the same workloads

6.2 EDF Schedulability

Utilization, short deadlines and interval demand

6.3 Fixed-Priority Analysis

Critical instant and response-time iteration

6.4 Irregular Events and Servers

Service budgets and replenishment

6.5 Concurrency and Shared Resources

Mutual exclusion, deadlock and blocking

Resources That Jobs Share

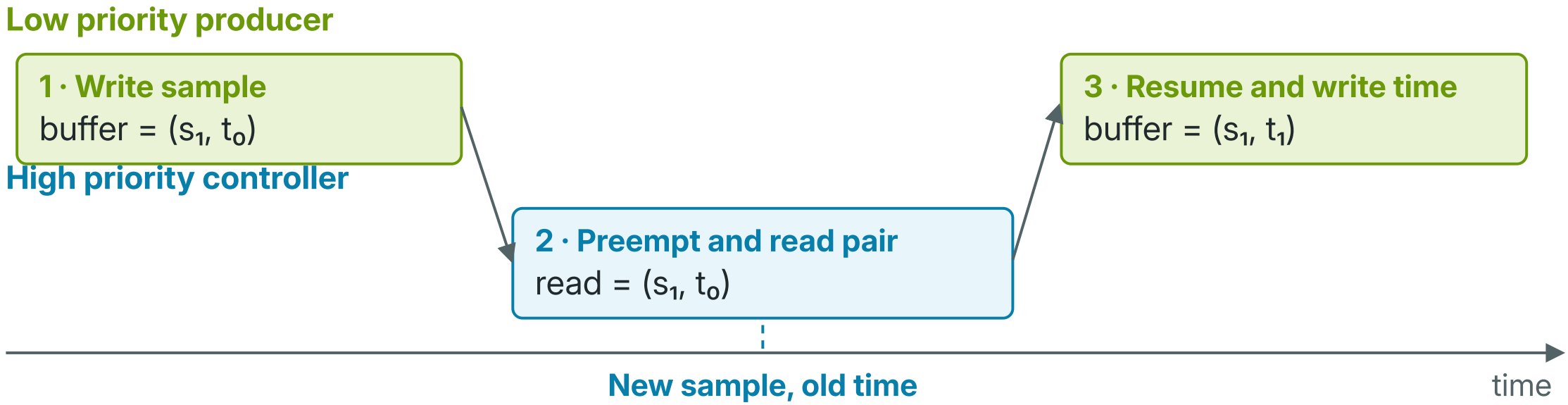
The producer publishes sensor samples. Control reads them. Telemetry may also use the same driver.

Resource	What must be coordinated?
CPU	Which ready job executes now?
Sample buffer	Can the reader see an update in progress?
Peripheral or driver state	Can two jobs have overlapping operations on the same device?

- **A shared resource is something multiple jobs need. Its access rules depend on what they do with it.**

One CPU, an Inconsistent Sample

A sensor record pairs sample s with timestamp t. In this trace, subscripts 0 and 1 mark the old and new versions.



- **Preemption can expose an incomplete update, even if individual reads and writes are atomic.**

A Mutex Protects the Pair

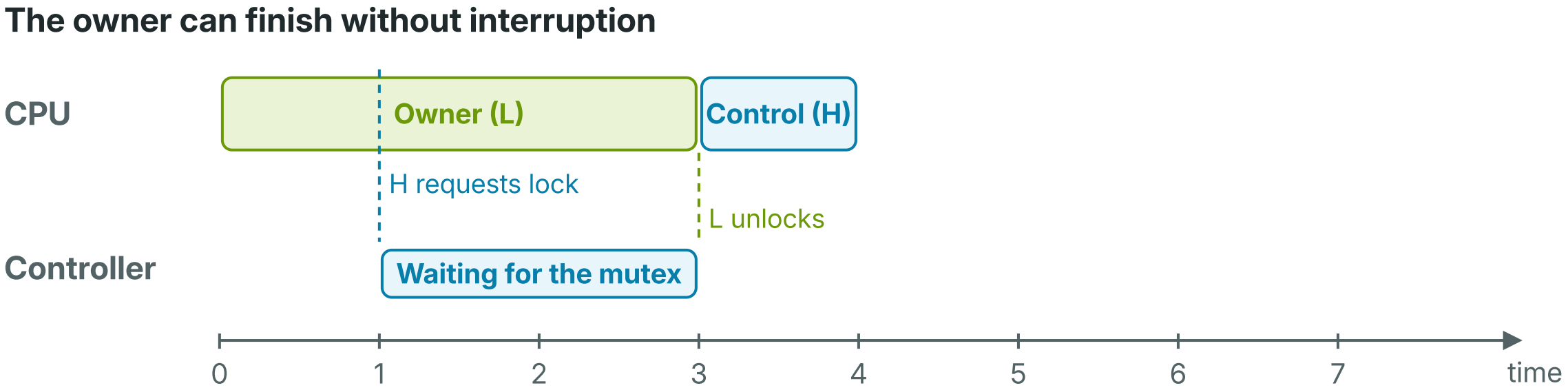
Both jobs must use the same lock around their access to the sample buffer.

Producer	Controller
Lock the buffer mutex.	Lock the buffer mutex.
Write sample and timestamp.	Read sample and timestamp into local variables.
Unlock the buffer mutex.	Unlock the buffer mutex.

- **A critical section is the protected code. A mutex admits one owner at a time.**

Direct Blocking by the Owner

Control needs the sample mutex. The lower-priority producer already owns it.

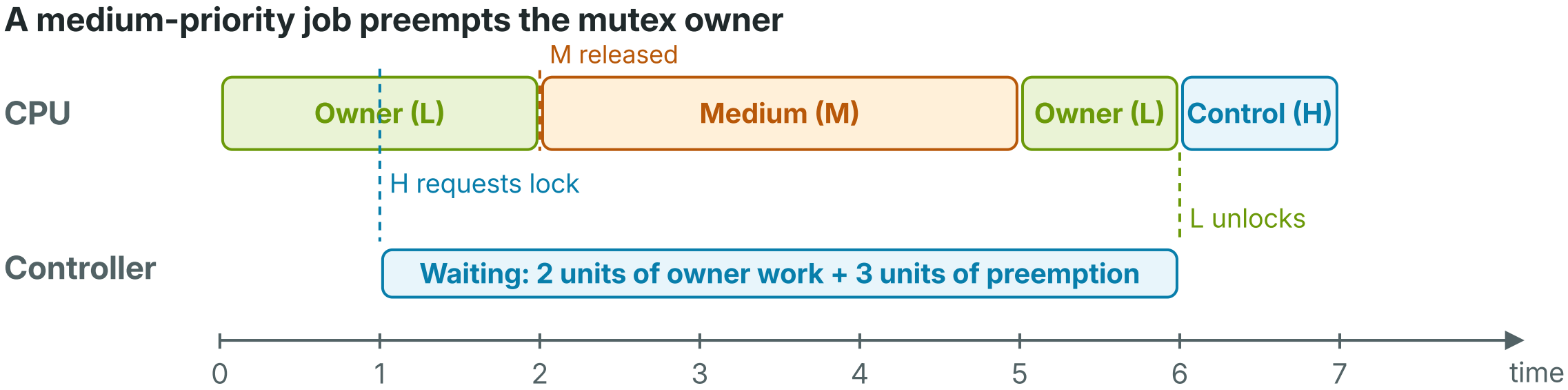


- **The controller is blocked on a lock until the owner releases it.**

The trace omits lock-call and dispatch costs. H and L denote high and low priority.

Priority Inversion Can Extend the Wait

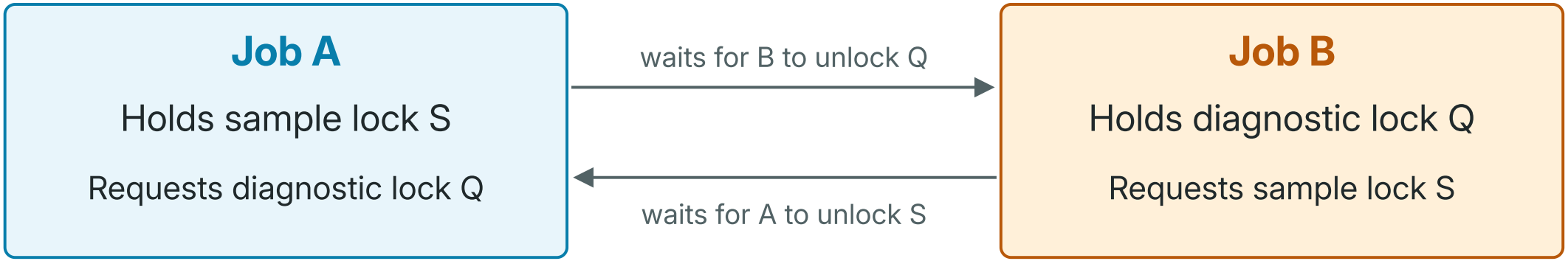
A medium-priority job becomes ready while the controller waits for the mutex owner.



- **The medium-priority job delays the controller indirectly: it preempts the owner that must release the lock.**

Deadlock Is a Cycle of Waiting

Two jobs each own one lock and request the other lock before releasing their own.



Both are waiting. Neither can reach its unlock.

- **In this deadlock, neither job can progress to its unlock, even if the CPU is available.**

Use One Order for Nested Locks

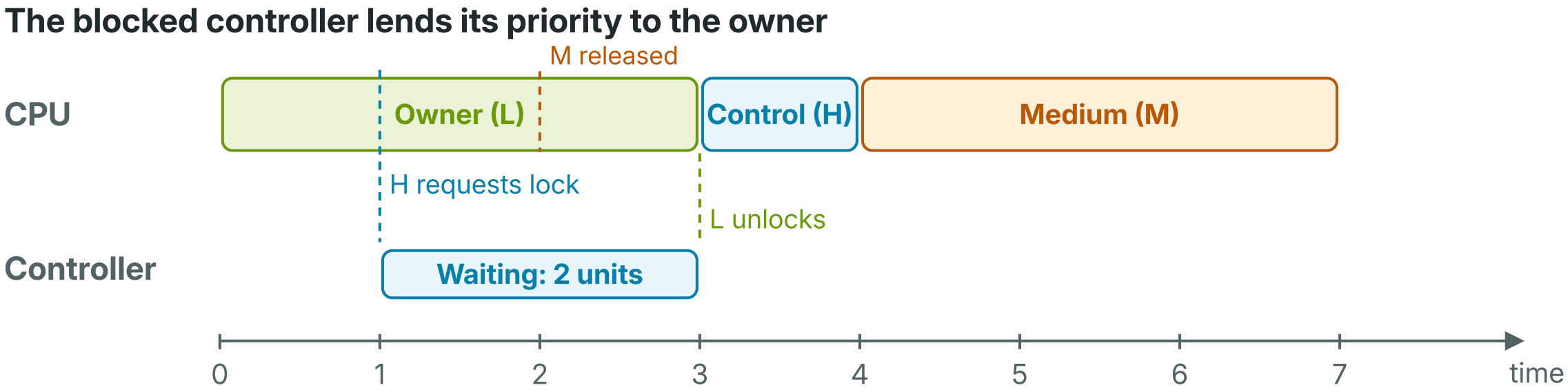
Require every job that needs both locks to acquire the sample lock S before the diagnostic lock Q.

Allowed	Rejected by the locking rule
Lock S, then lock Q.	Lock Q, then request S while holding Q.
Release Q, then release S.	Introduce a dependency that contradicts the global order.

- **A global lock order prevents circular wait among locks that follow the rule.**

Priority Inheritance Lets the Owner Run

A high-priority job blocks on a mutex. The owner temporarily inherits the blocked job's priority.



- **The Priority Inheritance Protocol (PIP) lets the owner finish before the medium-priority job can delay the release.**

What Priority Inheritance Leaves Open

A mutex owner may itself be blocked while it holds another lock.

Situation	What priority inheritance changes
H waits for L, and L waits for a third owner.	The inherited priority propagates along the wait chain.
A job meets more than one contested critical section.	Its total blocking can include several critical sections.
A set of jobs wait in a cycle.	Raising priorities leaves every participant waiting.

- **PIP limits preemption of lock owners. The lock structure still determines how many delays can accumulate.**

Set Each Resource's Priority Ceiling

Control, telemetry and the logger run at fixed priorities, in that order from highest to lowest.

Resource	Configured users	Resource ceiling
Sample lock S	Control and logger	Control's priority
Diagnostic lock Q	Telemetry and logger	Telemetry's priority

- **A resource's priority ceiling is the highest base priority of any job that may acquire it.**

Original PCP Also Checks Free Locks

The logger holds S. Telemetry preempts the logger and requests Q, which is free.

1. S is held by another job. Its ceiling is Control's priority.
2. Telemetry's effective priority does not exceed that ceiling. The request for Q waits.
3. The logger inherits Telemetry's priority. It can finish the critical section and release S.

- **Original Priority Ceiling Protocol (O-PCP) admits a free lock only when the requester's effective priority exceeds the ceilings of all locks held by other jobs.**

OSEK Raises Priority at Lock Acquisition

In a correctly configured OSEK task-only example, the logger calls `GetResource(S)`.

Protocol	When does the logger's priority rise?
Original PCP	When a requester actually blocks because of the held lock or its ceiling.
OSEK ceiling rule	As soon as the logger acquires S, up to the resource ceiling if that is higher.

- **With the immediate priority raise, Control may stay Ready while the logger completes its critical section.**

Bound the Blocking Term Under Original PCP

On one CPU under Original PCP, one lower-priority critical section can contribute to a job's blocking bound.

Find the candidates for B_i	Test
Who may hold the lock?	A task with lower base priority than task i.
Which lock counts?	Its ceiling is at least as high as task i's base priority.
How much time?	Take the longest CPU execution bound of the qualifying critical sections. Use zero if none qualify.

- **The blocking bound B_i can include a resource that task i never accesses, through the ceiling rule.**

Blocking Bounds for Control and Telemetry

Control outranks Telemetry, which outranks the logger. In this example, the logger is the only lower-priority task that can hold a qualifying lock for either task.

Logger's critical section	Ceiling	CPU time bound	Can block Control?	Can block Telemetry?
Update sample S	Control	0.20 ms	Yes	Yes
Update diagnostic buffer Q	Telemetry	0.30 ms	No	Yes

- $B_{\text{Control}} = 0.20\,\text{ms}$ **and** $B_{\text{Telemetry}} = 0.30\,\text{ms}$ **under Original PCP.**

The two logger sections are disjoint and included in its CPU demand of 0.80 ms per job.

Add Blocking to Response-Time Analysis

For fixed priorities on one CPU, extend the response-time recurrence from earlier in the chapter.

$$w_i^{(0)} = C_i + B_i, \qquad w_i^{(q+1)} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^{(q)}}{T_j} \right\rceil C_j.$$

Task	Iteration in ms	Deadline check
Control	$0.55 + 0.20 = 0.75$ with no higher-priority interference	$0.75 \le 1.00$
Telemetry	$0.70 + 0.30 = 1.00 \;\rightarrow\; 1.00 + \lceil 1.00/2 \rceil\, 0.55 = 1.55 \;\rightarrow\; 1.55$	$1.55 \le 2.10$

- **Both deadlines are verified with a bound for lower-priority blocking and higher-priority interference.**

Constrained relative deadlines, no release jitter, and no other suspension. Ignore dispatch and lock overhead in this teaching example.

Diagnose a Delay Before Choosing a Protocol

For each trace, explain who holds which resource and which job can still advance.

Observation	Question to ask
Control waits for the sample lock.	Can the owner run and release it?
A medium-priority job runs while the owner is ready.	Which priority does scheduling use for the owner?
Two owners wait for each other’s locks.	Which rule could prevent the cycle?

- **Separate direct blocking, extended priority inversion and deadlock. A timing proof needs a bound for the delays that remain.**

EZS Cheat Sheet · Tasks and Jobs

i identifies a task. $k = 1, 2, \ldots$ counts its successive jobs.

Task notation	Meaning	Job notation	Meaning
τ_i	Task i	$J_{i,k}$	Job k of task i
T_i	Period or minimum inter-arrival time	$r_{i,k}$	Release time
C_i	Execution-time bound per job	$d_{i,k} = r_{i,k} + D_i$	Absolute deadline
D_i	Relative deadline	$f_{i,k}$	Finish time
ϕ_i	Phase of a periodic task	$f_{i,k} - r_{i,k}$	Actual response time
J_i	Release jitter bound	$f_{i,k} \leq d_{i,k}$	Deadline met
R_i	Worst-case response time		

Periodic $\tau_i = (T_i, C_i, D_i, \phi_i)$ · **Sporadic** $\tau_i = (T_i, C_i, D_i)$

Burns Standard Notation (Davis, 2013). Course conventions ϕ_i, $J_{i,k}$, $r_{i,k}$ and $f_{i,k}$.

EZS Cheat Sheet · Scheduling Analysis

i identifies the analysed task and j a higher-priority task. q counts the iteration steps from 0. s identifies a server.

Symbol	Meaning
n, U_i, U	Number of tasks, task utilization $U_i = C_i/T_i$, total utilization $U = \sum_i U_i$
$P_i, hp(i)$	Base priority of task i, indices of tasks with strictly higher priority
$\ell, \mathrm{dbf}_i(\ell)$	Window length, maximum CPU demand of task i from jobs released and due within it
$w, w_i^{(q)}$	Response-window length, its estimate for task i at iteration q
B_i, R_i	Lower-priority blocking bound, worst-case response time
Q_s, T_s	Server budget in CPU time, replenishment period or delay
$C_s = Q_s$	Execution-time bound when modelling a polling server as a periodic task

Burns Standard Notation (Davis, 2013). Course conventions $\mathrm{dbf}_i(\ell)$, $w_i^{(q)}$, Q_s and T_s.

EZS Cheat Sheet · Finite Job Sets and Trace Labels

Symbol	Meaning
J_k	Job k in a finite set of independent jobs
r_k, C_k, d_k, f_k	Release time, CPU demand, absolute deadline, finish time of J_k
$[a, b]$	Test interval from release a to deadline b, with length $b - a$
$L_k = f_k - d_k, L_{\max} = \max_k L_k$	Lateness of J_k, maximum lateness over all jobs

Recurring-job traces. A_k and B_k are the kth jobs of tasks A and B. The demand-bound trace shows J_k for job k of its one sporadic task.

Sensor-buffer trace. In (s_1, t_0), s denotes a sample value and t a timestamp. Subscripts 0 and 1 mark the old and new measurements.

Self Check: What Delays the Controller?

The control code still needs 0.55 ms of CPU time and has a 1 ms deadline.

Observation	Question to answer
A high-priority diagnostic job runs at release.	How much higher-priority demand belongs in the response window?
Diagnosis is moved to a server.	When can it spend its budget, and when is that budget available again?
The control job needs a protected sample.	Which lower-priority critical section can block it? Can its owner reach the unlock?

- **Identify the source of the delay before choosing the timing model.**

From CPU Execution Time to End-to-End Timing

The schedule depends on the execution-time bounds and on where the work runs.

Next question	Model to extend
How do we justify C_i?	Analyse execution paths and platform timing in WCET estimation.
What if a job needs another component's output?	Model precedence and communication between activities.
What if work runs on multiple processors or crosses a bus?	Map work to resources and connect their timing guarantees.

- **Carry the assumptions of each local bound into the system-level timing argument.**

Appendix: A Finite Set of CPU Jobs

Three one-off jobs J_k become ready together. A batch of queued operations for one machine has the same structure.

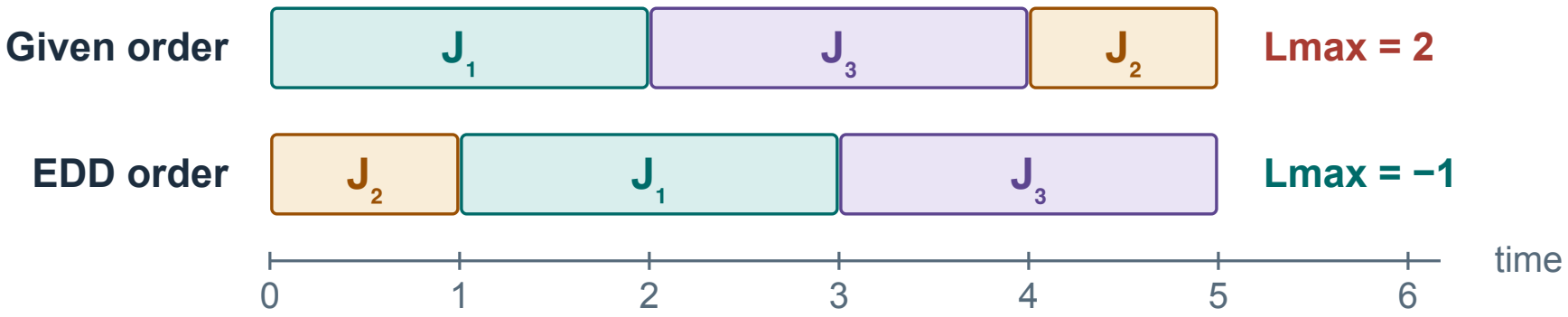
Job	Release r_k	CPU demand C_k	Absolute deadline d_k
J_1	0	2	4
J_2	0	1	3
J_3	0	2	7

With finish time f_k, a job's **lateness** is $L_k = f_k - d_k$. All deadlines are met when $L_{\max} = \max_k L_k \leq 0$.

Each row is a one-off job. Time is in arbitrary units. We schedule on one CPU. Jobs are independent, run without self-suspensions and are charged their stated CPU demand. Dispatch and preemption overhead are zero in this model.

Appendix: Earliest Due Date with a Common Release

At time 0 all three jobs are ready. **Earliest Due Date (EDD)** runs them in increasing order of absolute deadline.



- **With a common release, all jobs are known at the start. One sort by deadline gives an optimal order for maximum lateness.**

Appendix: The EDD Exchange Argument

Suppose adjacent jobs run in the order B, A even though $d_A \leq d_B$.

Ending times	What the swap to A, B changes
A moves earlier	Its lateness cannot increase.
B ends at the old finish time of A	Its lateness is at most the old lateness of A, since $d_B \geq d_A$.
The pair takes the same total time	The other jobs keep their finish times.

- **Eliminate each inverted pair without increasing $L_{\max}$. EDD minimizes maximum lateness.**

Appendix: The EDD Deadline Prefix Test

With all jobs released at time 0, every job with $d_k \leq b$ must fit before deadline b.

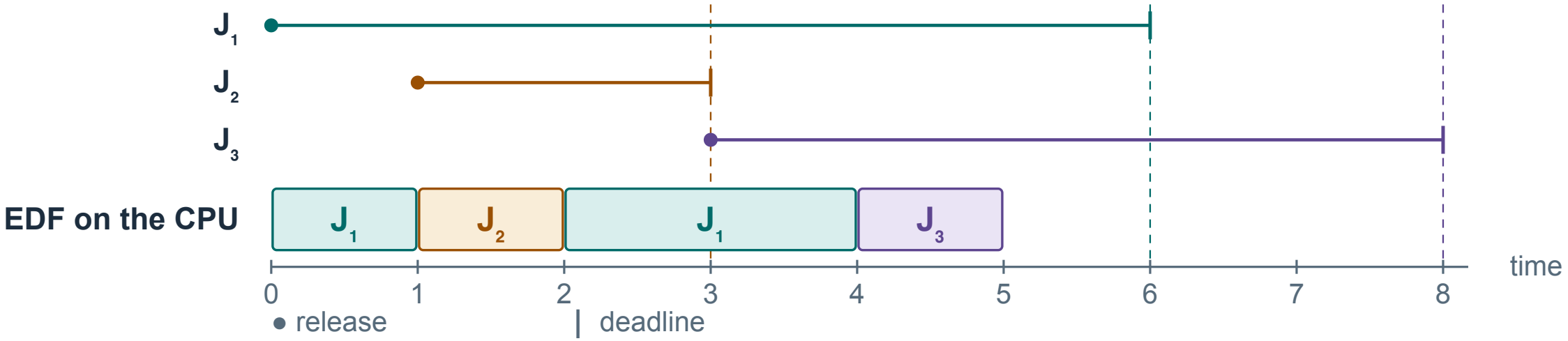
$$\sum_{k:\, d_k \leq b} C_k \;\leq\; b \qquad \text{for each deadline } b.$$

Deadline b	CPU time due by b	Meets the prefix condition?
3	1	$1 \leq 3$
4	$1 + 2 = 3$	$3 \leq 4$
7	$1 + 2 + 2 = 5$	$5 \leq 7$

- **EDD finishes one such prefix after another, so this test is exact.**

Appendix: EDF Selects the Earliest Ready Deadline

Controller, diagnosis and logging can become ready at different times. Preemptive **Earliest Deadline First (EDF)** runs the ready job with the earliest absolute deadline.



- **J_2 preempts J_1 at time 1. J_3 waits at time 3 because J_1 still has the earlier deadline.**

Appendix: Why EDF Can Repair Any Feasible CPU Plan

At the first disagreement, the current plan runs B. EDF would run A, already ready, with $d_A \leq d_B$.

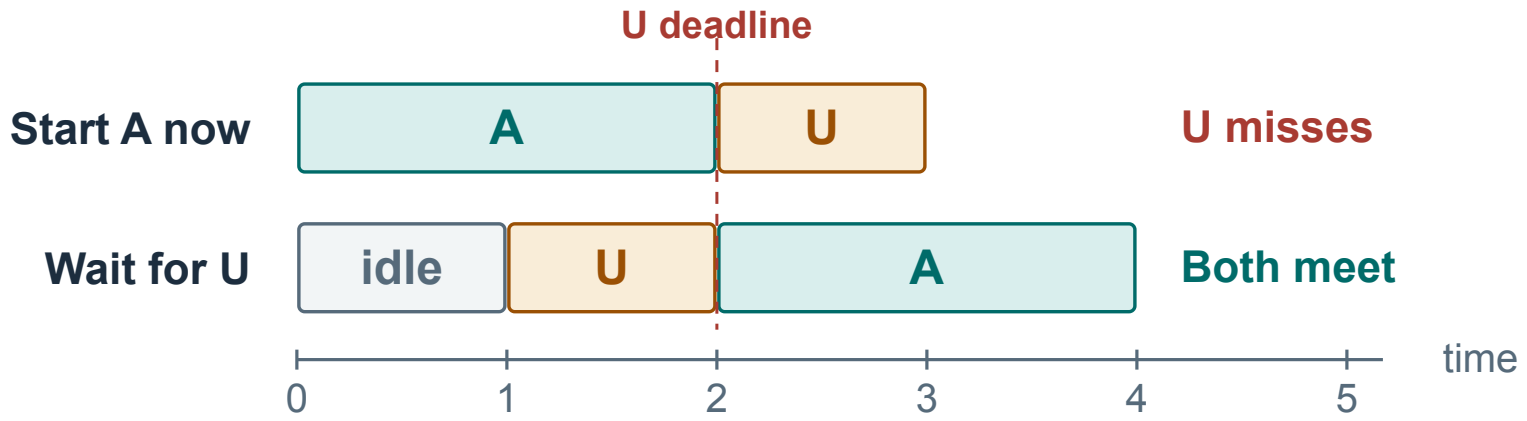
1. Move a later slice of A to this earlier position.
2. Move the displaced slice of B to the old position of A.
3. In a feasible plan that old position ends by $d_A \leq d_B$.

- **With preemption, the exchange preserves deadlines. If a feasible schedule exists, EDF finds one.**

One CPU, finite independent jobs, arbitrary releases, preemption and no dispatch cost. Horn (1974), as presented by Chen A9/12–13.

Appendix: A Nonpreemptive Plan May Need Idle Time

Two jobs remain, given as (r, C, d) (release, CPU demand, absolute deadline). $A = (0, 2, 4)$ and urgent job $U = (1, 1, 2)$.



- **A plan that knows when U will arrive can reserve the first short window for it.**

Reading Guide

Reading in Marwedel, 4th edition

- Independent jobs and recurrent tasks → §§6.2.1 and 6.2.3
- Sporadic events and server motivation → §6.2.5
- Resource access protocols → §§4.2.1–4.2.4
- Follow-up on execution bounds → §5.2.2

- **Marwedel covers the scheduling and locking models. The server papers extend the short introduction in §6.2.5.**

References

- P. Marwedel. *Embedded System Design*, 4th ed., 2021. Springer
- R. I. Davis. *Burns Standard Notation for real time scheduling*, 2013. Author's copy
- P. Ulbrich. *Echtzeitsysteme (EZS)*, FAU Erlangen-Nürnberg, winter semester 2019/20. Lecture slides
- S. Baruah, A. Mok and L. Rosier. *Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor*, 1990. Paper
- N. Audsley et al. *Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling*, 1993. Paper
- B. Sprunt, L. Sha and J. Lehoczky. *Scheduling Sporadic and Aperiodic Events in a Hard Real-Time System*, 1989. Technical report
- L. Sha, R. Rajkumar and J. Lehoczky. *Priority Inheritance Protocols*, 1990. Paper
- OSEK/VDX. *Operating System Specification 2.2.3*, 2005, §8. Specification