Scheduling issues in mixed-criticality systems

Sanjoy Baruah

The University of North Carolina at Chapel Hill

Scheduling issues in mixed-criticality systems

Integrated environments: Multiple systems on a shared platform

Why integrated architectures?

- Can provide a range of functionalities

Separate implementations are inefficient

- Size Weight and Power (SWaP) constraints

Dealing with mixed criticalities

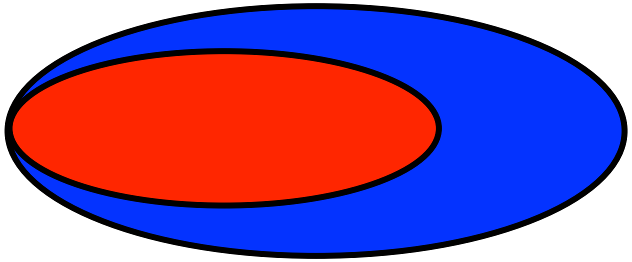
Mixed criticalities: an example

Some sub-systems are more important than others

- Automotive example: ABS vs car stereo

Different sub-systems have different certification requirements

- Defense avionics example. Flight-critical and mission-critical functionalities



Flight critical: certified by Certification Authorities

Mission-critical: validated by design team

Example: Determining worst-case execution time (WCET)

- flight-critical certification: cycle-counting under pessimistic assumptions
- mission-critical validation: extensive experimentation

Current practice: ARINC-653 “space-time partitioning”

- time partitioning: different time-slots are reserved for the flight-critical and the mission-critical sub-systems

Mixed criticalities: an example

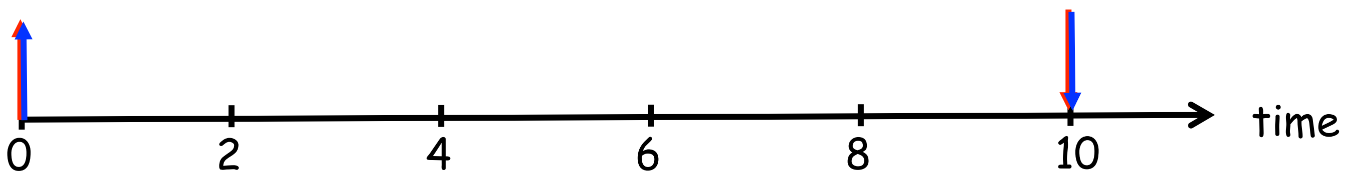
J_1 is flight-critical; J_2 is mission-critical
Both arrive at t=0; have deadlines at t=10
WCET of J_1 is 6; WCET of J_2 is 5

$6 + 5 > 10 \Rightarrow$ not schedulable

But…

- flight-criticality certification does <u>not</u> need J_2 to meet its deadline
- for mission-critical validation, J_1's WCET of 6 may be <u>too pessimistic</u>
  * Suppose J_1's WCET, obtained by extensive experimentation, is 4

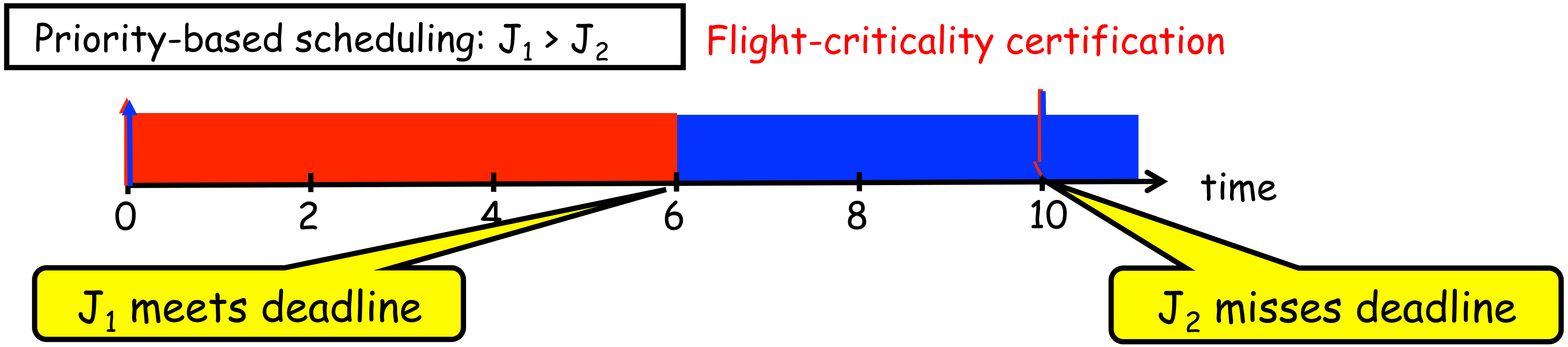
Priority-based scheduling: $J_1 > J_2$



Mixed criticalities: an example

J_1 is flight-critical; J_2 is mission-critical
Both arrive at t=0; have deadlines at t=10
WCET of J_1 is 6; WCET of J_2 is 5

But…
- flight-criticality certification does not need J_2 to meet its deadline
- for mission-critical validation, J_1's WCET of 6 may be too pessimistic
  * Suppose J_1's WCET, obtained by extensive experimentation, is 4



Mixed criticalities: an example

J_1 is flight-critical; J_2 is mission-critical
Both arrive at t=0; have deadlines at t=10
WCET of J_1 is 6; WCET of J_2 is 5

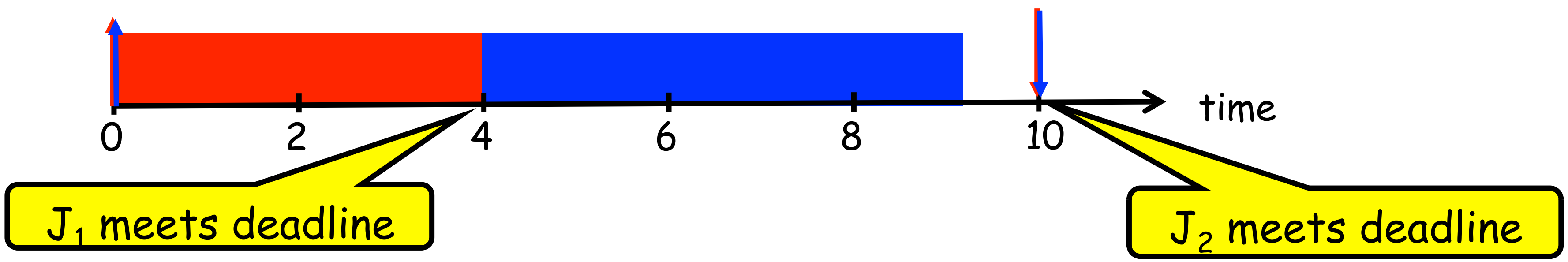
Validated at both criticalities

But…

- flight-criticality certification does not need J_2 to meet its deadline
- for mission-critical validation, J_1's WCET of 6 may be too pessimistic
    * Suppose J_1's WCET, obtained by extensive experimentation, is 4

Priority-based scheduling: $J_1 > J_2$

Mission-critical validation



Mixed criticalities

J_1 is flight-critical; J_2 is mission-critical
Both arrive at t=0; have deadlines at t=10
WCET of J_1 is 6; WCET of J_2 is 5

The same system is being validated, twice

Flight-critical certification	Mission-critical validation
at a very high level of assurance of only a subset of the system	at a lower level of assurance of the entire system

What are the right models, algorithms, and metrics for MC scheduling?

"Design-time resource reclaiming"

Mixed criticalities

J_1 is flight-critical; J_2 is mission-critical
Both arrive at t=0; have deadlines at t=10
WCET of J_1 is 6; WCET of J_2 is 5

The same system is being validated, twice

Flight-critical certification	Mission-critical validation
at a very high level of assurance	at a lower level of assurance
of only a subset of the system	of the entire system

What are the right models, algorithms, and metrics for MC scheduling?

OUTLINE

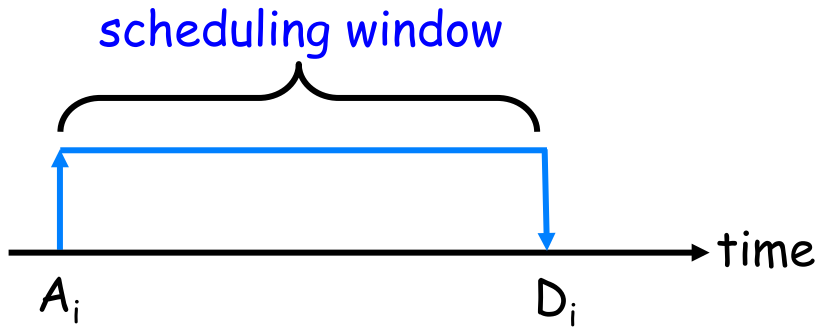
Restricted MC systems: models, algorithms, and metrics

Models, algorithms, and metrics for generalizations to the basic model

The mixed-criticality job model

Job J_i

- arrival time A_i
- deadline D_i
- criticality level L_i
- WCET function $C_i(1)$, $C_i(2)$, …



A positive integer

- larger values = greater criticality

Defense avionics: 2 (3?) criticalities

- safety-critical; mission-critical; non-critical

Civilian aviation (DO-178B): 5 criticalities

-catastrophic; hazardous; major; minor; no effect

Automotive systems (ISO 26262): 4 criticalities

The mixed-criticality job model

Job J_i

- arrival time A_i
- deadline D_i
- criticality level L_i
- WCET function $C_i(1)$, $C_i(2)$, …

$C_i(j)$: The worst-case execution time of job J_i, estimated at a level of assurance consistent with the j^{th} criticality level

(WCET-estimation tools and techniques are criticality level-specific)

Assume $C_i(j) \leq C_i(j+1)$ for all j

The mixed-criticality job model

Job J_i

- arrival time A_i
- deadline D_i
- criticality level L_i
- WCET function $C_i(1)$, $C_i(2)$, …

The MIXED-CRIT SCHEDULING PROBLEM: Given an instance $\{J_1, J_2, \ldots, J_n\}$ of mixed-criticality jobs, determine an appropriate scheduling strategy

CERTIFICATION CRITERION: Job J_i should meet its deadline when each job J_k executes for at most $C_k(L_i)$, for all J_i.

The WCET of J_k, computed at J_i's criticality level

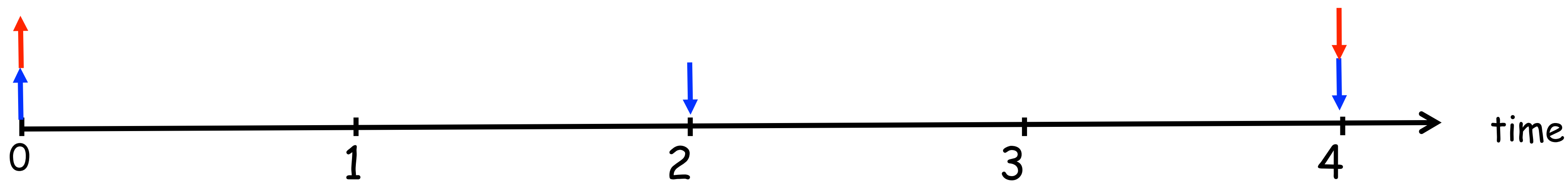
MC scheduling: An example

J_i:	L_i	A_i	$C_i(1)$	$C_i(2)$	D_i
J_1:	1				
J_2:	1				
J_3:	2				
J_4:	2				

1 → LO
2 → HI

MC scheduling: An example

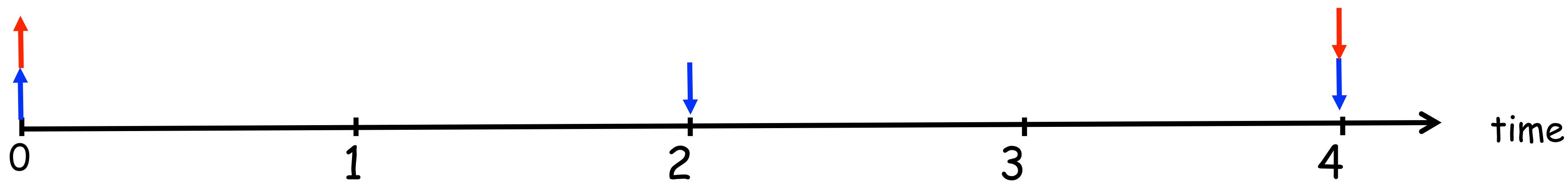
J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	1	2
J_2:	LO	0	1	1	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	2	4



Schedule for LO-criticality behavior - Earliest Deadline First (EDF)
Schedule for HI-criticality behavior - Any work-conserving algorithm

MC scheduling: An example

J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	1	2
J_2:	LO	0	1	1	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	2	4



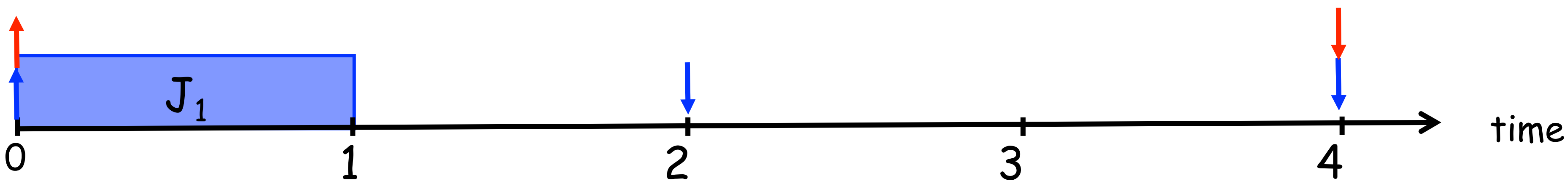
Schedule for LO-criticality behavior ✓
Schedule for HI-criticality behavior ✓

Schedule for <u>BOTH</u> behaviors?

MC scheduling: An example

J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	1	2
J_2:	LO	0	1	1	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	2	4

Earliest Deadline First (EDF) scheduling

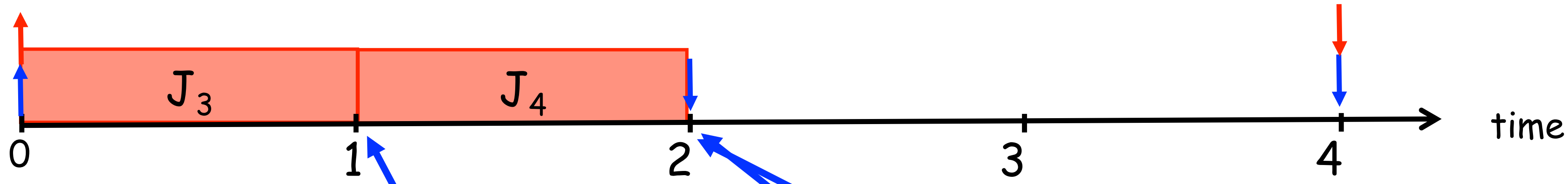


HI-criticality certification: must fit 4 units of work <u>here</u>

MC scheduling: An example

J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	1	2
J_2:	LO	0	1	1	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	2	4

Criticality-Monotonic scheduling



J_3 completes execution

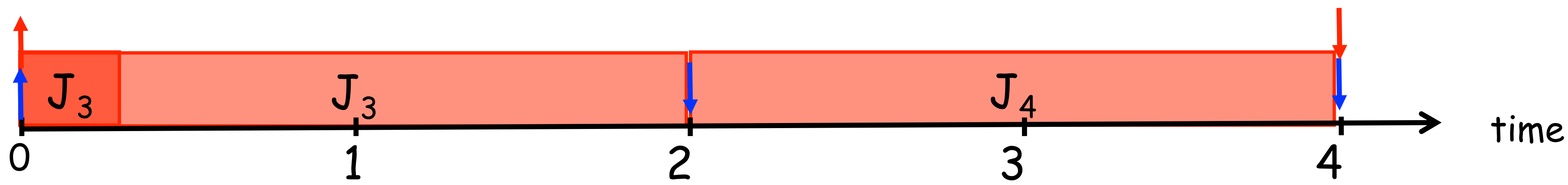
J_4 completes execution

LO-criticality validation: J_1 misses its deadline

MC scheduling: An example

J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	1	2
J_2:	LO	0	1	1	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	2	4

If J_3 does <u>not</u> complete by 1:

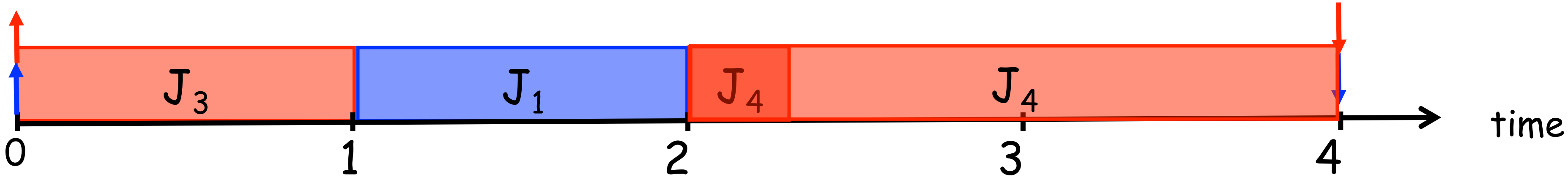


MC scheduling: An example

J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	1	2
J_2:	LO	0	1	1	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	2	4

If J_3 completes by 1:

If J_4 does not complete by 3:



MC scheduling: An example

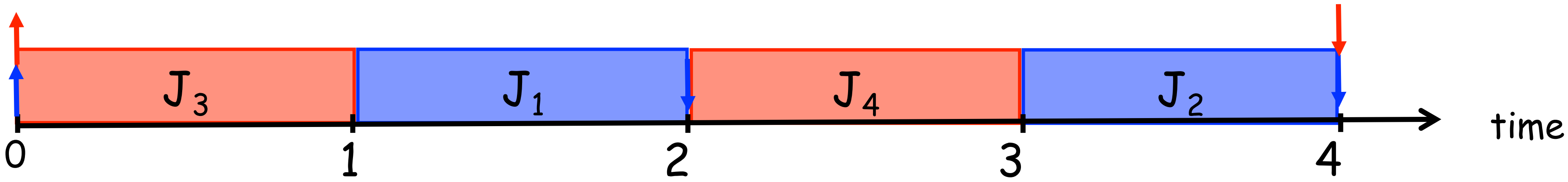
J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	1	2
J_2:	LO	0	1	1	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	2	4

<u>A correct strategy</u>:

- execute J_3 first
- if J_3 executes for ≤ 1, J_1 is next
- J_4 is next
- J_2 executes last

If J_3 completes by 1:

If J_4 completes by 3:



The complexity of MC scheduling

Given an instance of mixed-criticality jobs, determining whether an appropriate scheduling strategy exists for it is NP-hard in the strong sense

- For uniprocessors as well as multiprocessors
- Upon both preemptive and non-preemptive processors
- Even if there are only two distinct criticality levels
- And all jobs arrive simultaneously

Coping with intractability

Given an instance of mixed-criticality jobs, determining whether an appropriate scheduling strategy exists for it is NP-hard in the strong sense

Focus on dual criticality instances:

Each job is either HI-criticality or LO-criticality

$J_i = (L_i, A_i, C_i(\text{LO}), C_i(\text{HI}), D_i)$

$L_i \in \{\text{LO}, \text{HI}\}$

Coping with intractability

Given an instance of mixed-criticality jobs, determining whether an appropriate scheduling strategy exists for it is NP-hard in the strong sense

Focus on dual criticality instances:

Each job is either HI-criticality or LO-criticality

- For ease of presentation
- Important special case: HI-crit. jobs need certification; LO-crit. jobs do not
- Already intractable
- All techniques & results generalize to more criticality levels

A preemptive uniprocessor scheduling algorithm

Dual-criticality instance I = {J_1, J_2, …, J_n}

Assign priorities by Lawler's technique (Audsley's algorithm)

- recursively find a lowest-priority job

```
    I' := I
L1: Ji := a job that may be assigned lowest priority in I'
    I' := I' - {Ji}
    if I' is not empty then goto L1
```

A preemptive uniprocessor scheduling algorithm

Dual-criticality instance $I = \{J_1, J_2, \ldots, J_n\}$

Assign priorities by Lawler's technique (Audsley's algorithm)

- recursively find a lowest-priority job

```
Ji := a job that may be assigned lowest priority in I'
```

J_i may be assigned lowest priority if it meets its deadline as the lowest-priority job, when each job J_k executes for $C_k(L_i)$ time units

The WCET of J_k, computed at J_i's criticality level

A preemptive uniprocessor scheduling algorithm

Dual-criticality instance $I = \{J_1, J_2, \ldots, J_n\}$

Assign priorities by Lawler's technique (Audsley's algorithm)

- recursively find a lowest-priority job

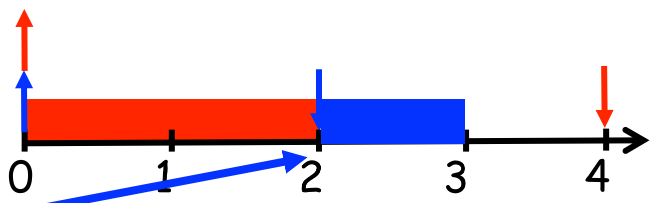
```
Ji := a job that may be assigned lowest priority in I'
```

J_i may be assigned lowest priority if it meets its deadline as the lowest-priority job, when each job J_k executes for $C_k(L_i)$ time units

An example:

J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	2	2
J_2:	HI	0	2	2	4

Can J_1 be lowest priority? - no!



J_1 misses its deadline

A preemptive uniprocessor scheduling algorithm

Dual-criticality instance $I = \{J_1, J_2, \ldots, J_n\}$

Assign priorities by Lawler's technique (Audsley's algorithm)

- recursively find a lowest-priority job

```
Ji := a job that may be assigned lowest priority in I'
```

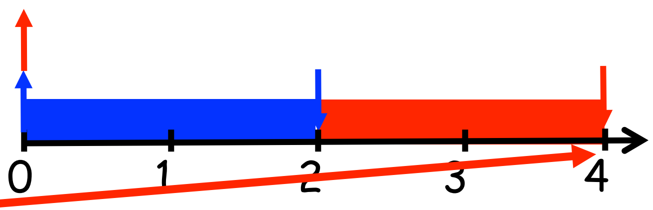
J_i may be assigned lowest priority if it meets its deadline as the lowest-priority job, when each job J_k executes for $C_k(L_i)$ time units

An example:

J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	2	2
J_2:	HI	0	2	2	4

Can J_1 be lowest priority? - no!

Can J_2 be lowest priority? - yes



J_2 meets its deadline

A preemptive uniprocessor scheduling algorithm

Dual-criticality instance $I = \{J_1, J_2, \ldots, J_n\}$

Assign priorities by Lawler's technique (Audsley's algorithm)

- recursively find a lowest-priority job

```
Ji := a job that may be assigned lowest priority in I'
```

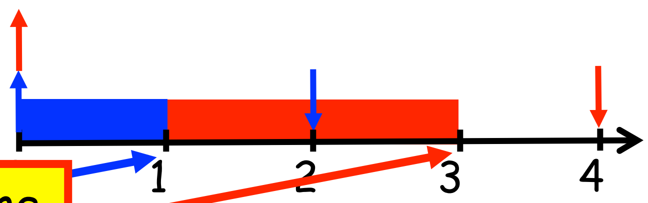
J_i may be assigned lowest priority if it meets its deadline as the lowest-priority job, when each job J_k executes for $C_k(L_i)$ time units

An example:

J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	2	2
J_2:	HI	0	2	2	4

Priority ordering: $J_1 > J_2$

LO-criticality certification:



J_1 meets its

J_2 meets its deadline

A preemptive uniprocessor scheduling algorithm

Dual-criticality instance $I = \{J_1, J_2, \ldots, J_n\}$

Assign priorities by Lawler's technique (Audsley's algorithm)

- recursively find a lowest-priority job

```
Ji := a job that may be assigned lowest priority in I'
```

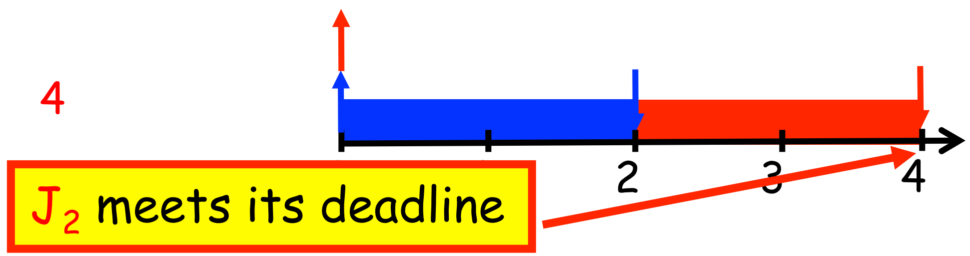
J_i may be assigned lowest priority if it meets its deadline as the lowest-priority job, when each job J_k executes for $C_k(L_i)$ time units

An example:

J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	2	2
J_2:	HI	0	2	2	4

Priority ordering: $J_1 > J_2$

HI-criticality certification:



A preemptive uniprocessor scheduling algorithm

Dual-criticality instance $I = \{J_1, J_2, \ldots, J_n\}$

Assign priorities by Lawler's technique (Audsley's algorithm)
- recursively find a lowest-priority job

```
Ji := a job that may be assigned lowest priority in I′
```

J_i may be assigned lowest priority if it meets its deadline as the lowest-priority job, when each job J_k executes for $C_k(L_i)$ time units

OCBP: Own Criticality-Based Priorities

Dual-criticality instance $I = \{J_1, J_2, \ldots, J_n\}$

Assign priorities by Lawler's technique (Audsley's algorithm)

- recursively find a lowest-priority job

```
Ji := a job that may be assigned lowest priority in I'
```

J_i may be assigned lowest priority if it meets its deadline as the lowest-priority job, when each job J_k executes for $C_k(L_i)$ time units

PROPERTIES:

* Polynomial runtime
  - $O(n^3 \log n)$ naive; $O(n^2)$

*Quantitative performance bound
  - assuming some run-time support
  - based on system load parameter

Run-time support for mixed criticalities

Does the run-time system police the execution of jobs?

$C_i(HI) \gg C_i(LO)$ for LO-criticality jobs

WCET at HI criticality

WCET at LO criticality

If run-time system can enforce execution budgets

assign the LO-criticality job J_i a budget of $C_i(LO)$

$C_i(HI) = C_i(LO)$ for LO-criticality job J_i

- But policing and budgeting overhead costs must be accounted for

- Policing and budget-enforcement functionalities are HI-criticality

The load parameter

For "regular" real-time instances:

demand(I, $[t_1, t_2)$) ≡ cumulative execution requirement of jobs of instance I over the time interval $[t_1, t_2)$

$$\boxed{\text{load}(I) \equiv \max_{\text{all } [t1,t2)} \left\{ \text{demand}(I,[t_1,t_2)) \Big/ (t_2-t_1) \right\}}$$

RESULT: Any regular (i.e., non-MC) instance I is feasible on a preemptive uniprocessor if and only if load(I) ≤ 1

Generalization to dual-criticality instances

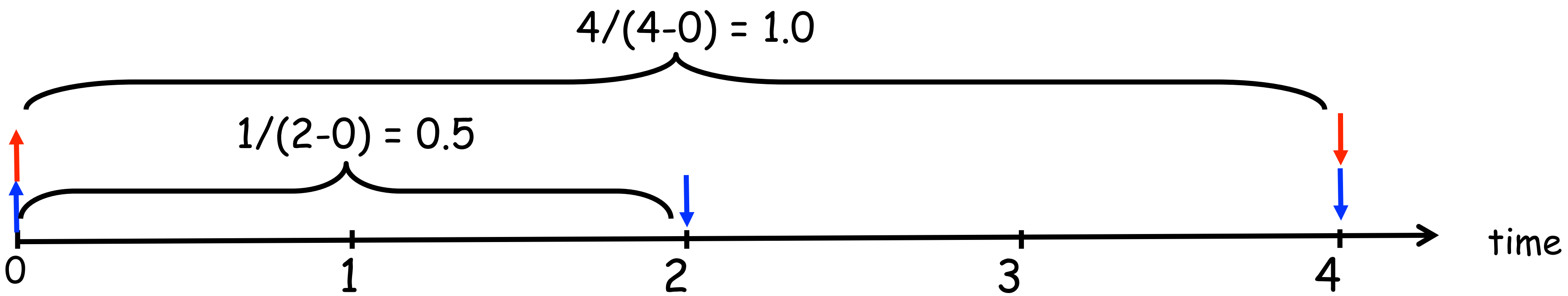
*$\text{load}_{LO}(I)$ - load "expected" by system designer
(all jobs; LO-criticality WCET's)

*$\text{load}_{HI}(I)$ - load to be certified
(only HI-criticality jobs; HI-criticality WCET's)

The load parameter: an example

J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	1	2
J_2:	LO	0	1	1	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	1	4

$load_{LO} = \max(0.5, 1.0) = 1.0$



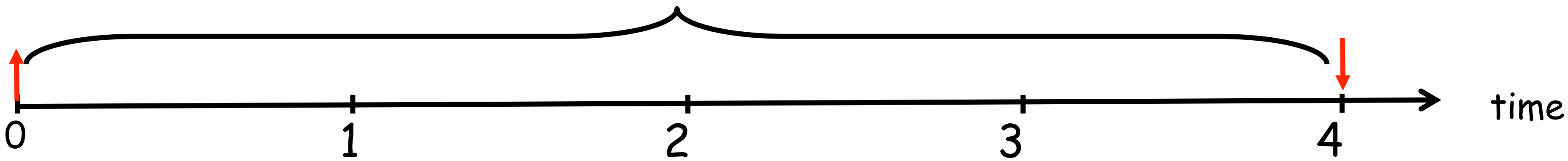
The load parameter: an example

J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	1	2
J_2:	LO	0	1	1	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	1	4

$load_{LO} = \max(0.5, 1.0) = 1.0$

$load_{HI} = 0.75$

(2+1)/(4-0) = 0.75



The load parameter: an example

J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	1	2
J_2:	LO	0	1	1	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	1	4

$\text{load}_{LO} = \max(0.5, 1.0) = 1.0$

$\text{load}_{HI} = 0.75$

This instance I has low-criticality load $\text{load}_{LO}(I) = 1.00$

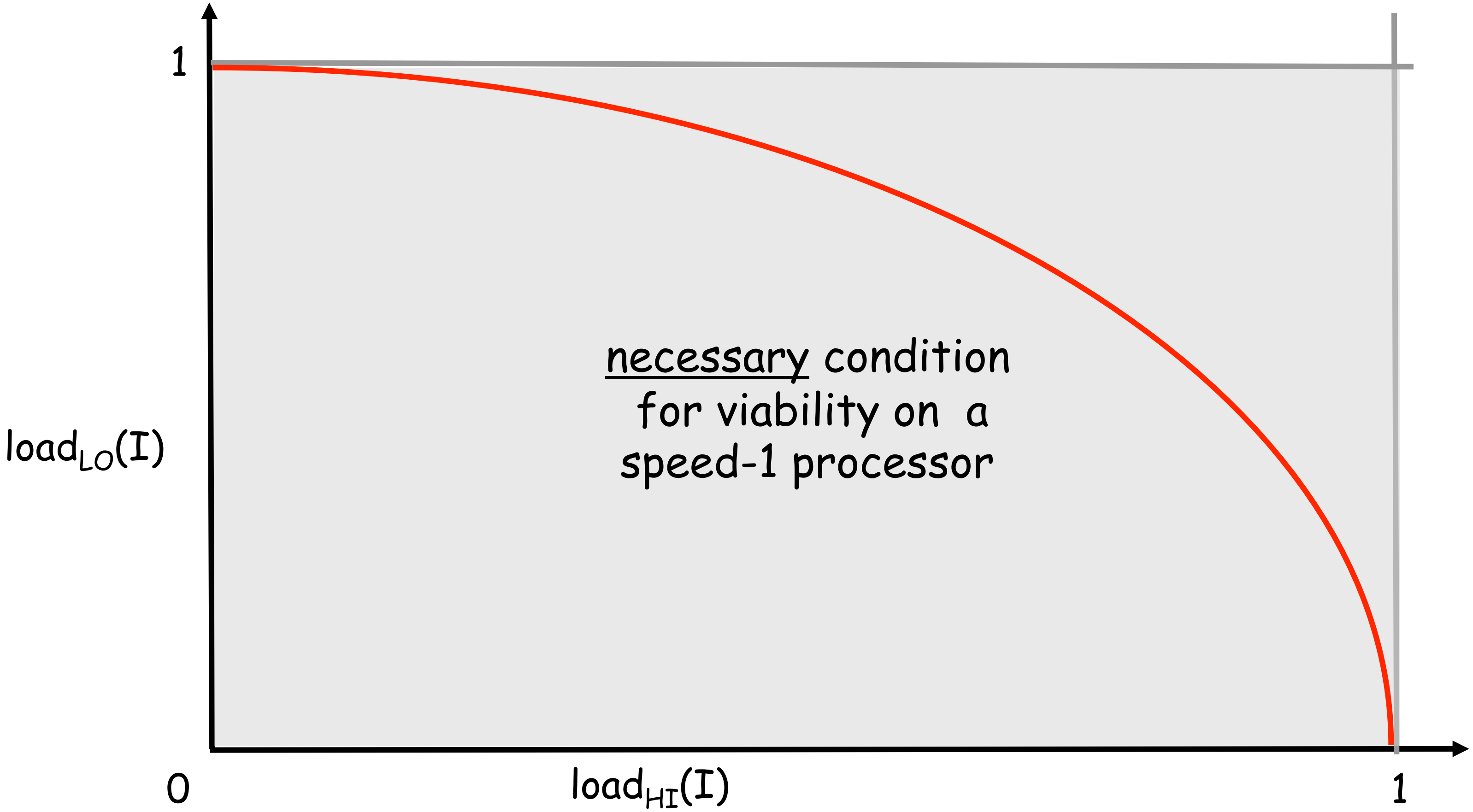
and high-criticality load $\text{load}_{HI}(I) = 0.75$

OCBP: A sufficient schedulability condition

RESULT: Algorithm OCBP schedules any dual-criticality instance I satisfying

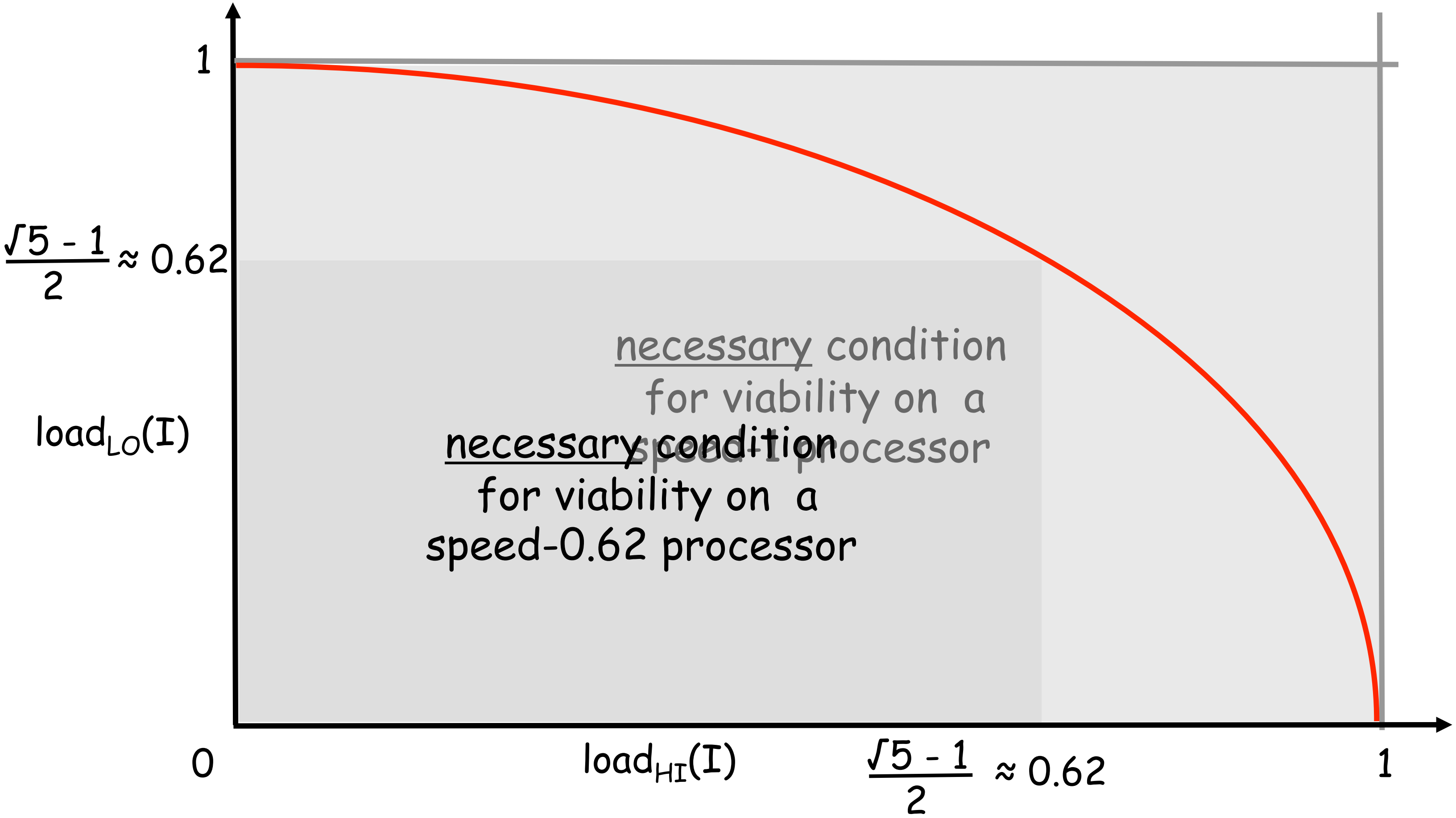
$$\text{load}_{HI}(I) + \text{load}_{LO}(I)^2 \leq 1$$

on a preemptive unit-speed processor



OCBP: A sufficient schedulability condition

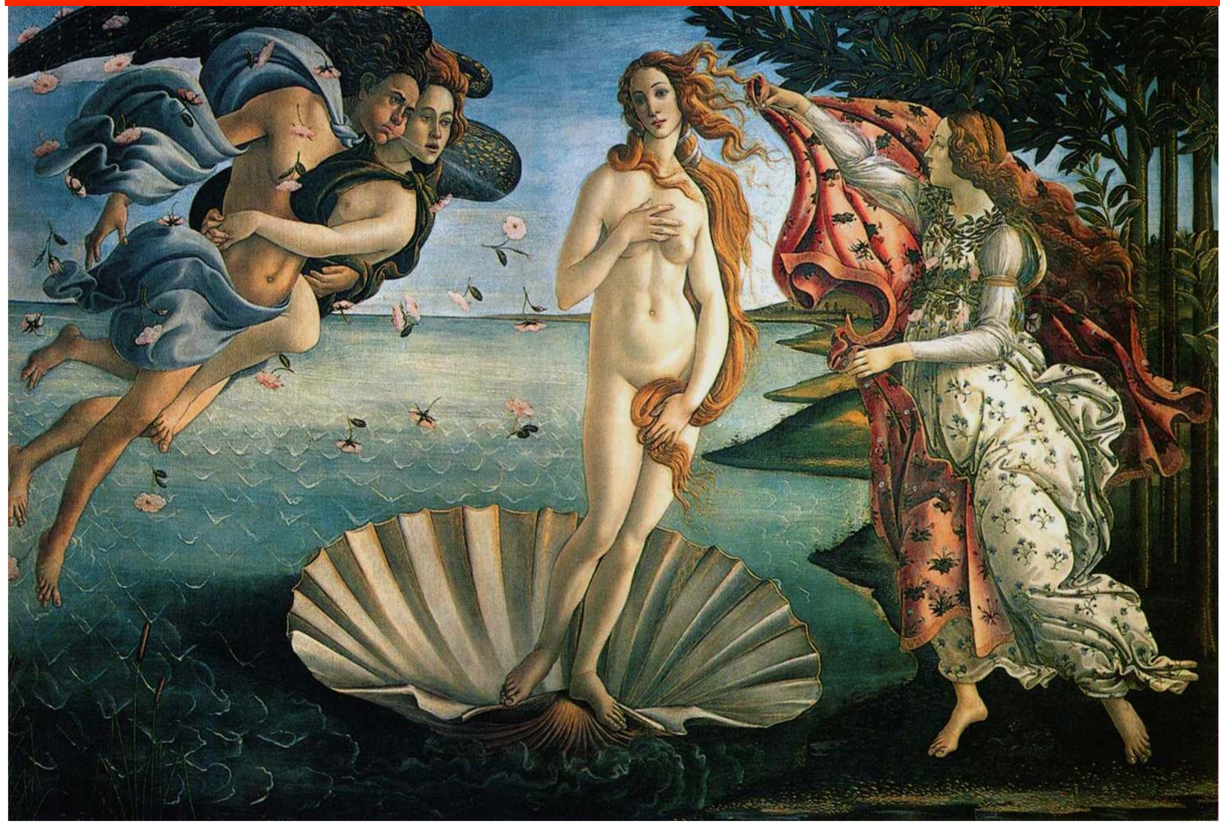
RESULT: Any dual-criticality instance I feasible on a unit-speed processor is OCBP-schedulable on a speed-$\frac{2}{\sqrt{5}-1} = \frac{\sqrt{5}+1}{2}$ ($\approx$ 1.618) processor



OCBP: A sufficient schedulability condition

RESULT: Any dual-criticality instance I feasible on a unit-speed processor is OCBP-schedulable on a speed-$\frac{2}{\sqrt{5}-1} = \frac{\sqrt{5}+1}{2}$ (≈ 1.618) processor

The Golden Ratio: positive solution to $x^2 - x - 1 = 0$



Recurrent tasks

Recurring tasks or processes

- generate jobs
- represent code within an infinite loop

Different tasks are assumed independent

```
for(;;){
        .
        .
        .
        .
        .
}
```

Recurrent tasks: the sporadic tasks model

Task $\tau_i = (D_i, T_i, L_i, [C_i(LO), C_i(HI)])$

- D_i: relative deadline D_i
- T_i: minimum inter-arrival separation ("period")
- $L_i \in \{LO, HI\}$
- $C_i(LO)$, $C_i(HI)$: WCET estimates

Jobs

- first job arrives at any time
- consecutive arrivals at least T_i time units apart
- each job has criticality L_i, and WCET's as specified
- each job has its deadline D_i time units after arrival

The dual-criticality scheduling problem for sporadic task systems: Given a collection $\{\tau_1, \tau_2, \ldots, \tau_n\}$ of dual-criticality sporadic tasks, determine an appropriate scheduling strategy

Algorithms for scheduling systems of recurrent tasks

A classification of priority-based scheduling algorithms

1. Fixed Task Priority (FTP) - e.g., Deadline monotonic (DM)
2. Fixed Job Priority (FJP) - e.g., EDF
3. Dynamic Priority (DP) - e.g., Least Laxity

EASIER TO IMPLEMENT

MORE GENERAL

Algorithms for scheduling systems of recurrent tasks

A classification of priority-based scheduling algorithms

1. Fixed Task Priority (FTP) - e.g., Deadline monotonic (DM)
2. Fixed Job Priority (FJP) - e.g., EDF
3. Dynamic Priority (DP) - e.g., Least Laxity

Optimal FTP for "regular" sporadic task systems

Deadline Monotonic not optimal for mixed-criticality tasks

-S. Vestal (RTSS'07). Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance

Criticality Optimal Priority Assignment (COPA)

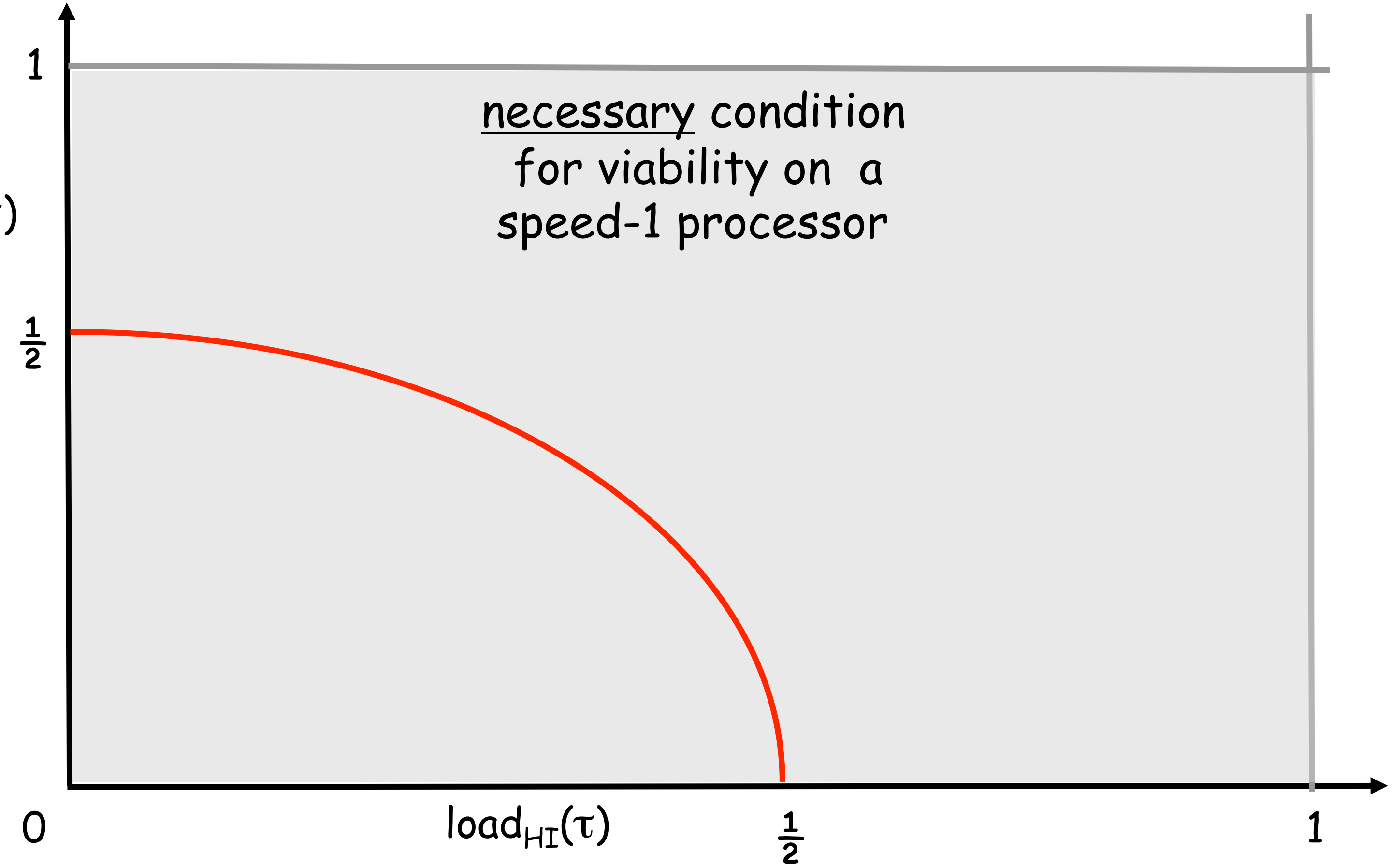
- Application of Lawler's technique to dual-criticality sporadic task systems
- Yields an optimal priority assignment
- Quantitative guarantees, assuming run-time support for budget enforcement

COPA: A sufficient schedulability condition

RESULT: COPA schedules any dual-criticality sporadic task system τ satisfying

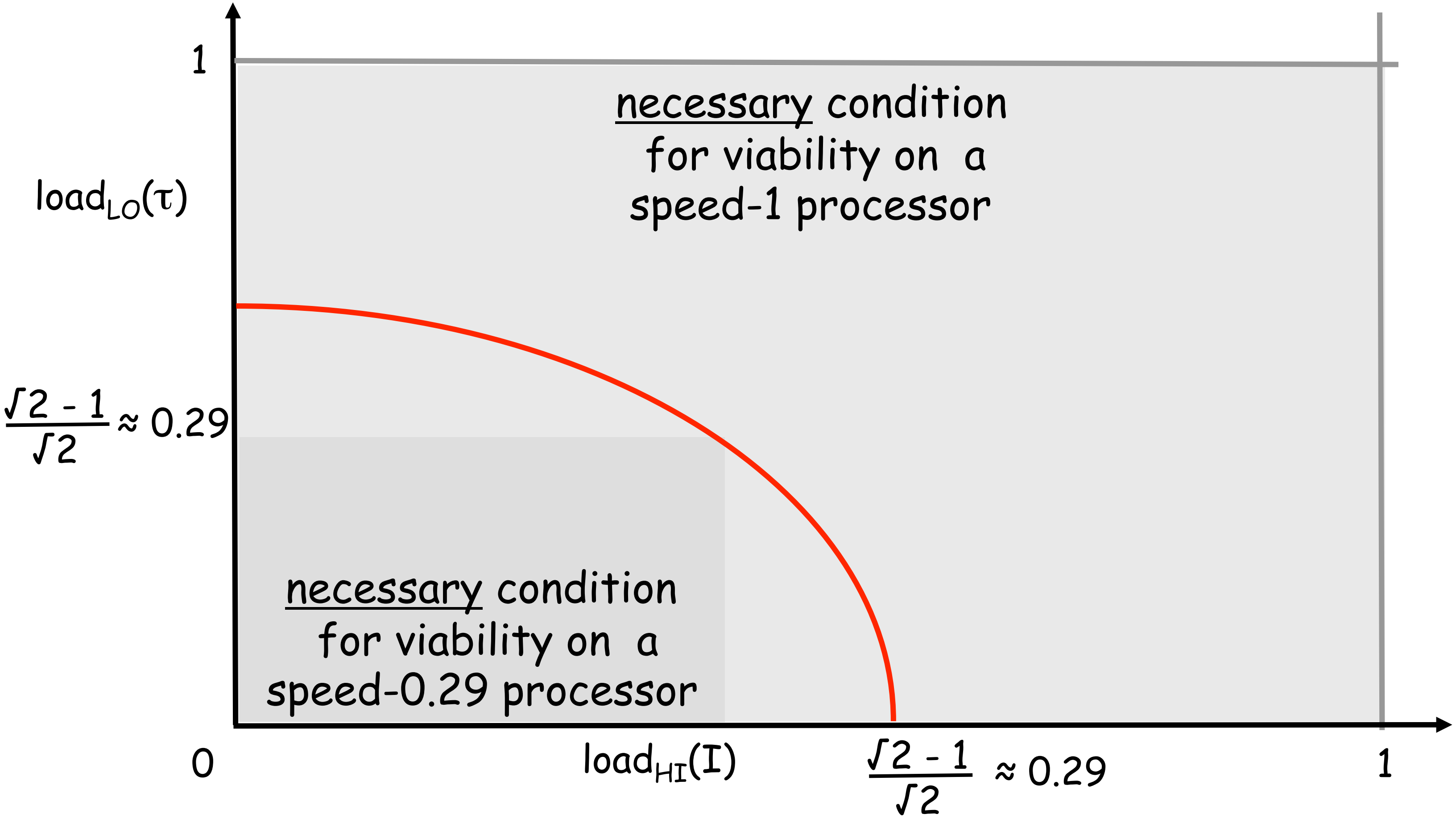
$$\text{load}_{LO}(\tau) + \text{load}_{HI}(\tau) - \text{load}_{LO}(\tau) \times \text{load}_{HI}(\tau) \leq \tfrac{1}{2}$$

on a preemptive unit-speed processor



COPA: A sufficient schedulability condition

RESULT: Any dual-criticality sporadic task system τ feasible on a unit-speed proc. is COPA-schedulable on a speed-$\frac{\sqrt{2}}{\sqrt{2}-1}$ = $2 + \sqrt{2}$ ($\approx$ 3.414) proc



Algorithms for scheduling systems of recurrent tasks

A classification of priority-based scheduling algorithms

1. Fixed Task Priority (FTP) - e.g., Deadline monotonic (DM)
2. Fixed Job Priority (FJP) - e.g., EDF
3. Dynamic Priority (DP) - e.g., Least Laxity

- Criticality Optimal Priority Assignment (COPA) is an optimal FTP algorithm for dual-criticality sporadic task systems

- If run-time system enforces execution quotas for jobs

  Sufficient schedulability condition: $load_{LO}(\tau) + load_{HI}(\tau) - load_{LO}(\tau) \times load_{HI}(\tau) \leq \frac{1}{2}$

  Tight processor speedup bound: $(2 + \sqrt{2})$, ≈ 3.414

Algorithms for scheduling systems of recurrent tasks

A classification of priority-based scheduling algorithms

1. Fixed Task Priority (FTP) - e.g., Deadline monotonic (DM)
2. Fixed Job Priority (FJP) - e.g., EDF
3. Dynamic Priority (DP) - e.g., Least Laxity

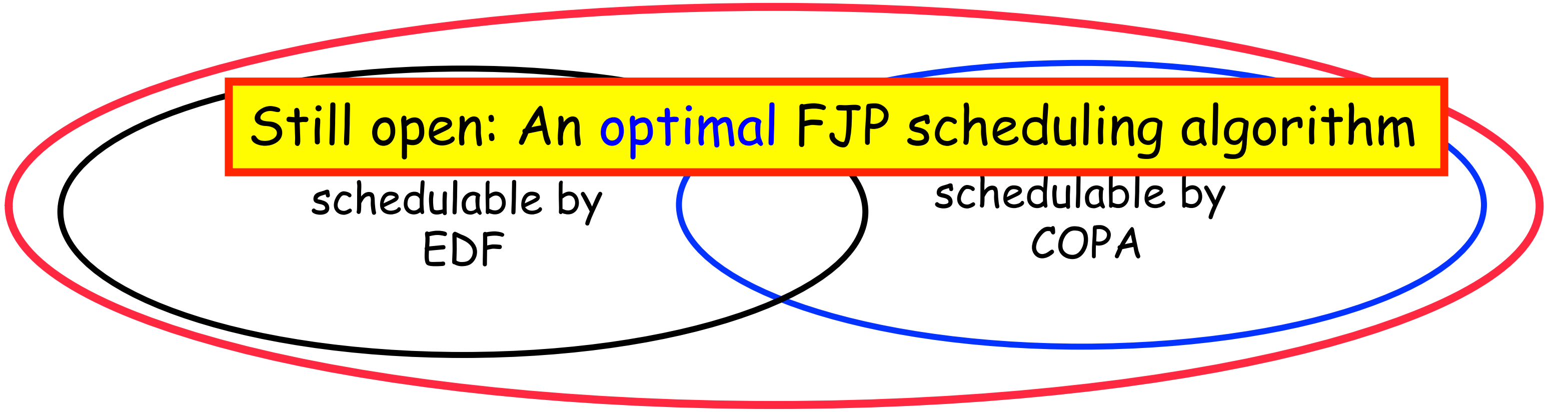
Optimal FJP for "regular" sporadic task systems

* EDF and COPA are incomparable

$\Rightarrow$ EDF is not optimal

* An FJP algorithm that dominates both EDF and COPA

Still open: An optimal FJP scheduling algorithm



Algorithms for scheduling systems of recurrent tasks

A classification of priority-based scheduling algorithms

EASIER TO IMPLEMENT

1. Fixed Task Priority (FTP) - e.g., Deadline monotonic (DM)
2. Fixed Job Priority (FJP) - e.g., EDF
3. Dynamic Priority (DP) - e.g., Least Laxity

Also optimal DP for "regular" sporadic task systems

* There are DP-schedulable dual-criticality sporadic task systems that no FJP algorithm can schedule

$\Rightarrow$ optimality requires DP-scheduling

Open question: What is the minimum degree of dynamism needed for optimality?

The sporadic task model + shared resources

A dual-criticality sporadic task

- relative deadline
- minimum inter-arrival separation ("period")
- criticality
- worst-case execution requirements

Platform: preemptive uniprocessor

+ additional serially reusable resources

Jobs access shared resources

- within critical sections…which may be nested

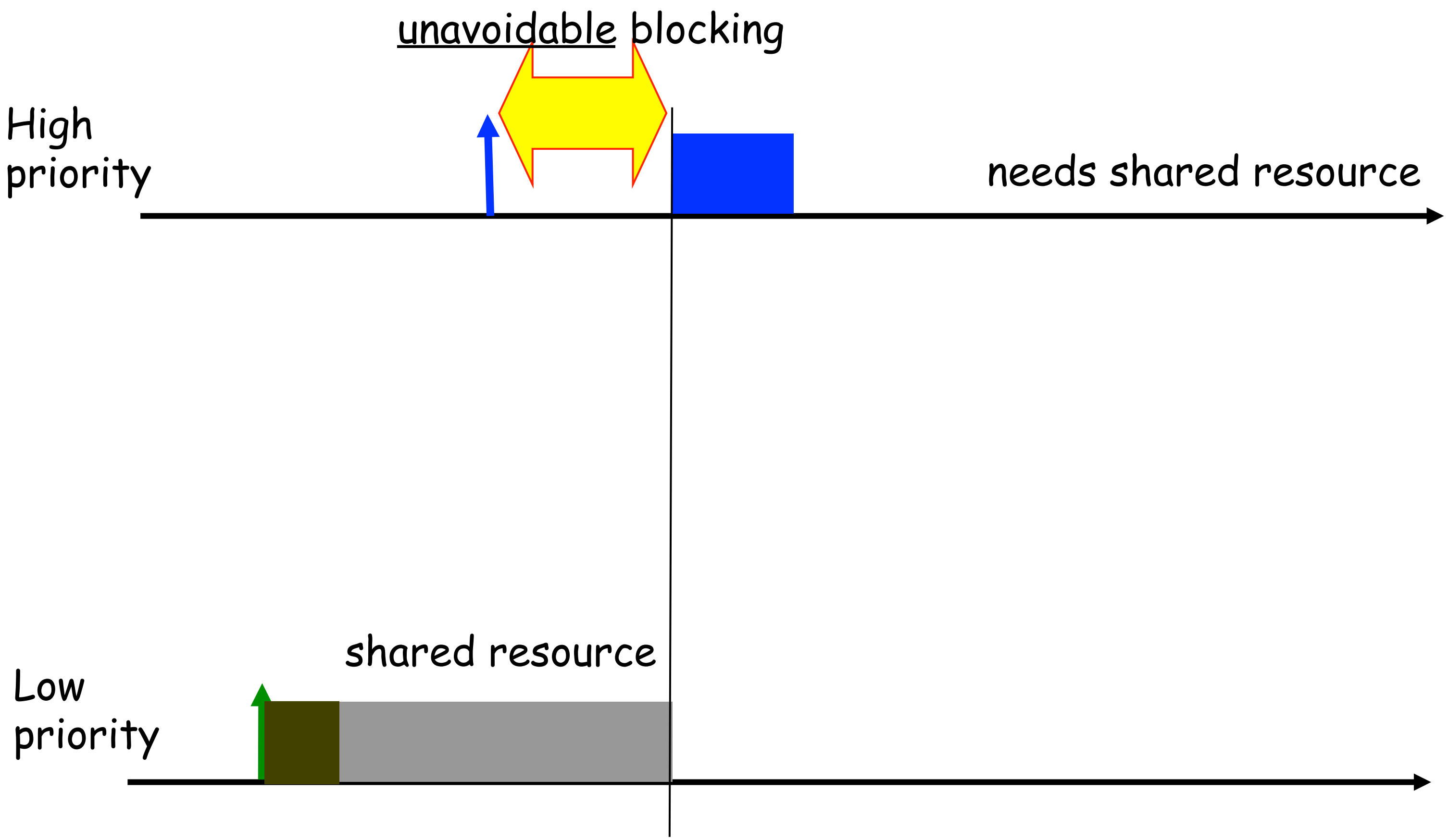
```
for(;;){
 - lock (R1)
       - lock (R3)
       - unlock (R3)
 - unlock (R1)
 - lock (R2)
 - unlock (R2)
}
```

Priority inversion

A lower-priority job executes instead of a higher-priority one

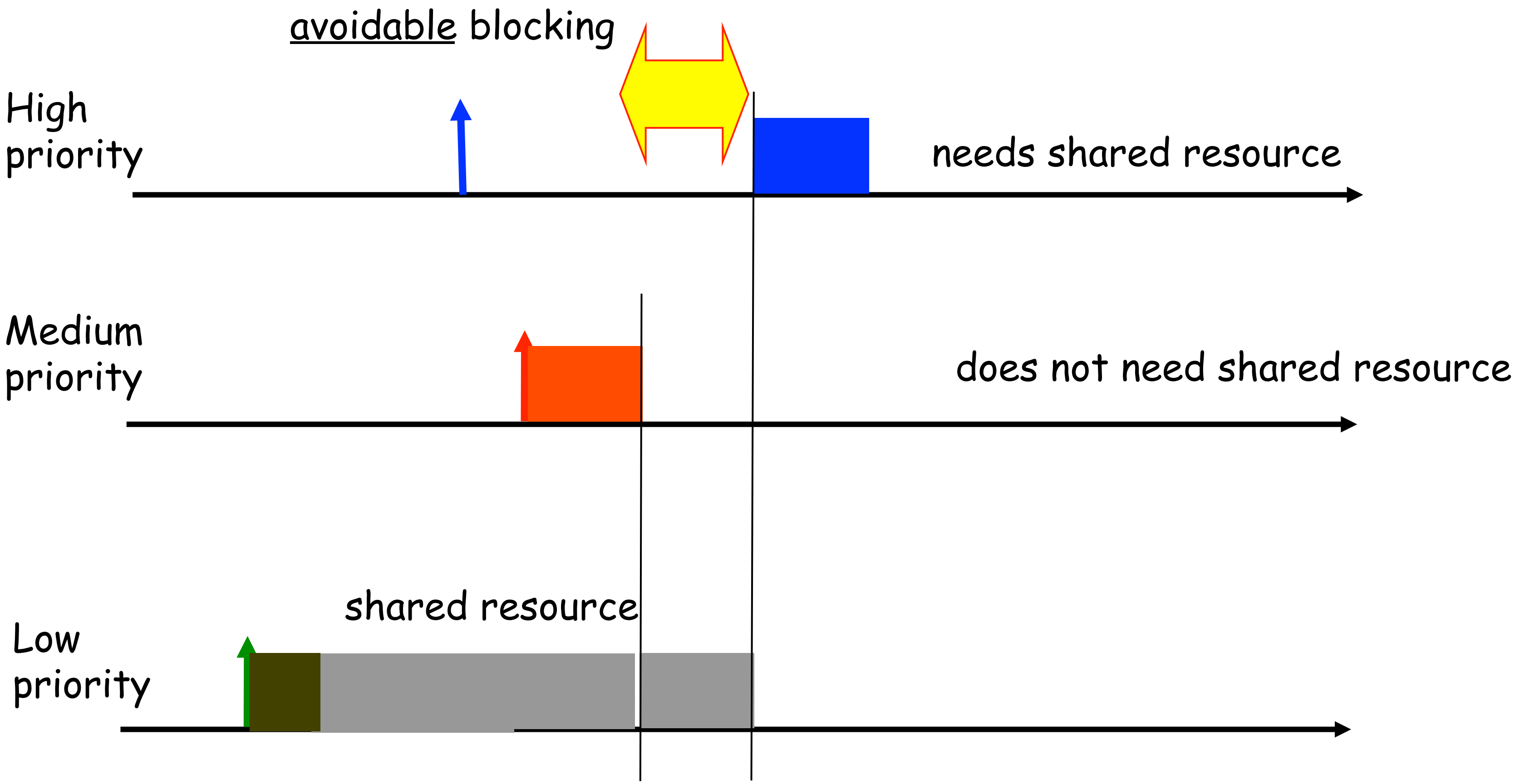
Serially reusable shared resources

Priority inversion and blocking



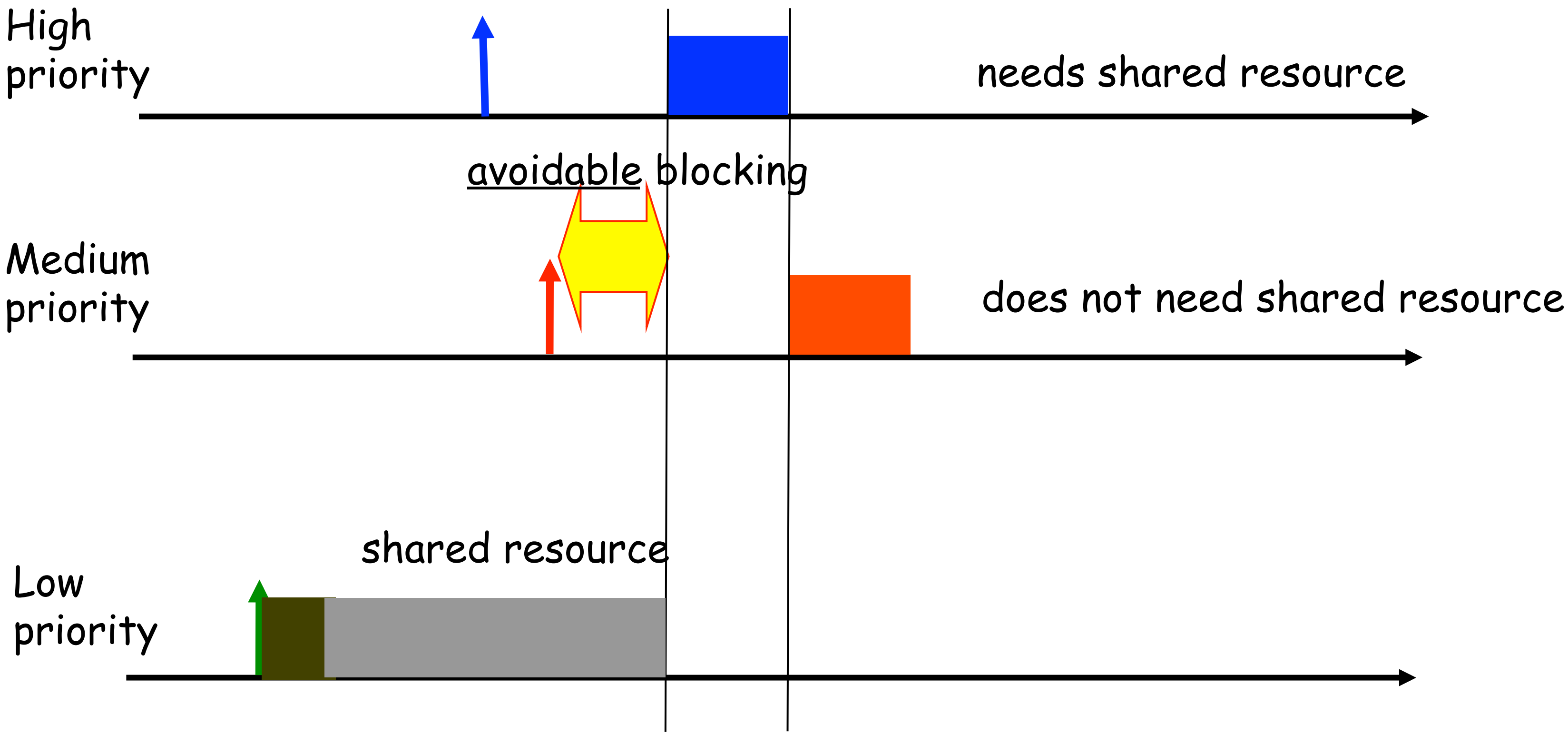
Serially reusable shared resources

Priority inversion and blocking



Serially reusable shared resources

Priority inversion and blocking



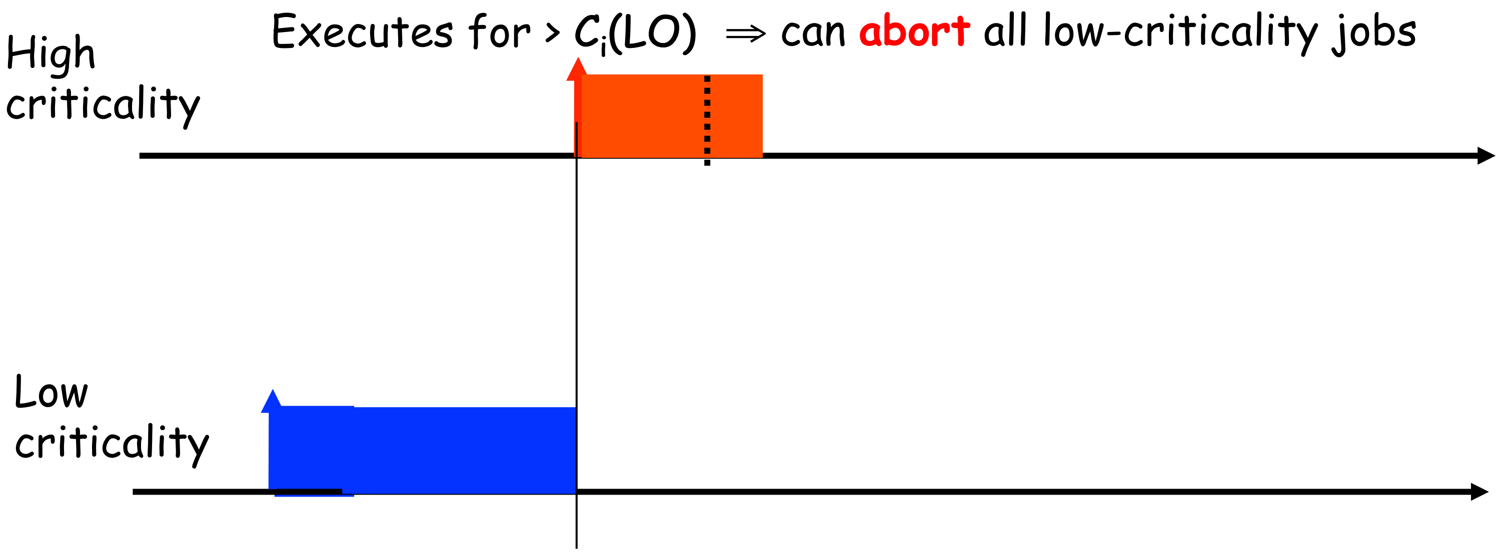
Serially reusable shared resources

Ted Baker. Stack-based scheduling of real-time processes. *Real-Time Systems: The International Journal of Time-Critical Computing* 3(1). 1991.

The STACK RESOURCE POLICY (SRP) is optimal for resource-sharing "regular" sporadic task systems: if any task system is uniproc. feasible, then EDF + SRP guarantees to schedule it to meet all deadlines

Serially reusable shared resources

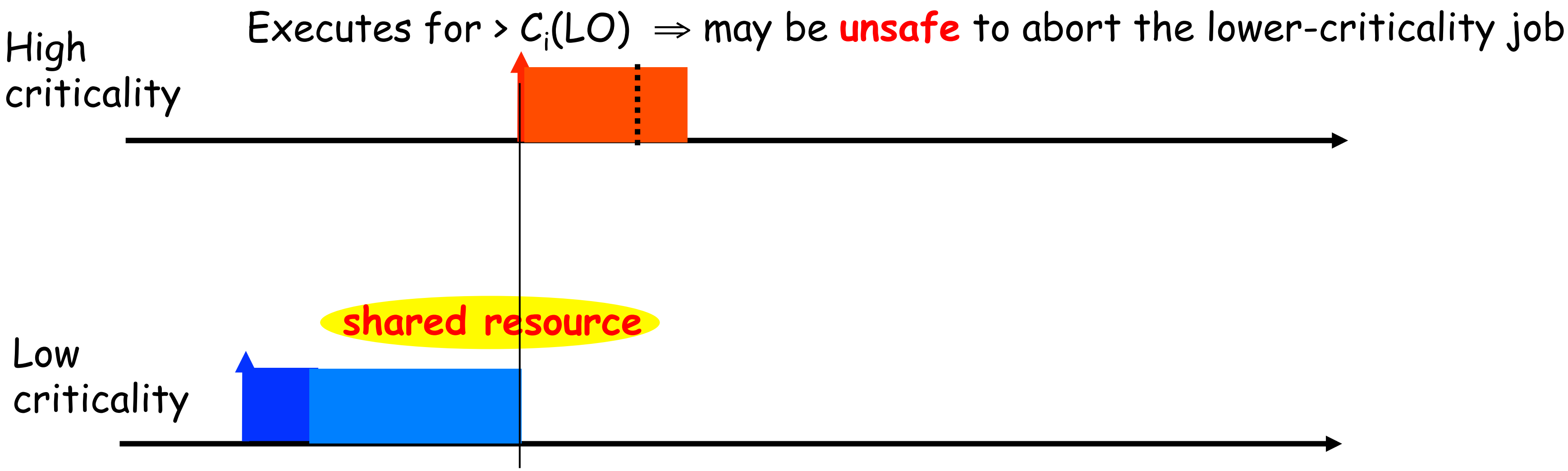
Mixed criticality scheduling without shared resources



Serially reusable shared resources

Mixed criticality scheduling **with** shared resources

Problem: Design an efficient, certifiable strategy for arbitrating access to shared resources for mixed-criticality sporadic task systems



Context and conclusions

Platform-sharing is here to stay

Different certification criteria for different systems

- must be validated to different levels of assurance

Current practice: space-time partitioning

is inefficient

- in resource usage: Size, Weight, and Power (SWaP)
- in certification effort

Needed: Certifiably correct techniques for implementing mixed-criticality systems

- A formal model for mixed-criticality workloads
- generated by recurrent tasks
- that share non-preemptable resources