Certification-cognizant Scheduling in Integrated Computing Environments

Sanjoy Baruah

The University of North Carolina at Chapel Hill

Supported by the National Science Foundation, the Army Research Office, the Air Force Office of Scientific Research, and the Air Force Research Laboratory

Certification-cognizant Scheduling in Integrated Computing Environments

Many real-time systems perform safety-critical functions

Certification authorities (CAs) ensure system safety

Aviation:

- Federal Aviation Authority (FAA)
- European Aviation Safety Agency (EASA)

Medical devices:

- Food and Drug Administration (FDA)

Certification-cognizant Scheduling in Integrated Computing Environments

Many real-time systems perform safety-critical functions

Certification authorities (CAs) ensure system safety

CAs tend to be very conservative…

…can require over-provisioning of computing resources

Certification-cognizant Scheduling in Integrated Computing Environments

Multiple functionalities on a shared platform

Why integrated computing environments?

- Can support a wider range of functionalities

Separate implementations are inefficient

- Size Weight and Power (SWaP) constraints

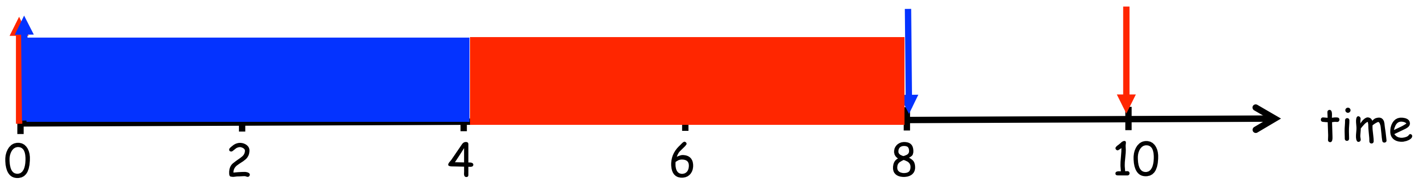
Certification-cognizant Scheduling
in
Integrated Computing Environments

An example

2 jobs – J_1 and J_2 – on a preemptive processor
Both arrive at t=0; have deadlines at t=10 and t=8
WCET of J_1 is 4; WCET of J_2 is 4

[worst-case execution requirement]

Earliest Deadline First (EDF) schedule:



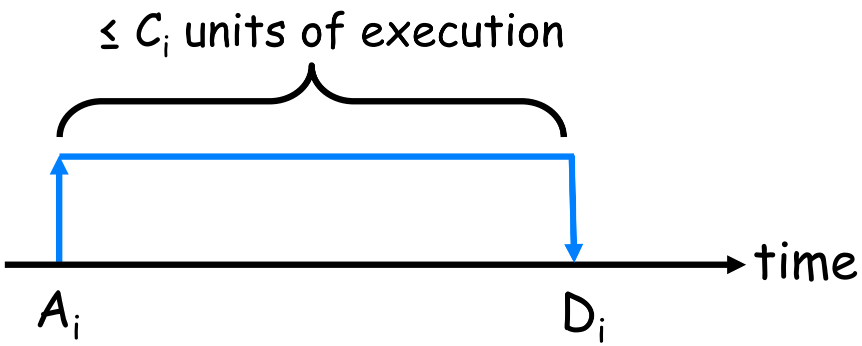
An example

A_i D_i

2 jobs – J_1 and J_2 – on a preemptive processor
Both arrive at t=0; have deadlines at t=10 and t=8
WCET of J_1 is 4; WCET of J_2 is 4

C_i's

* Only J_1 subject to certification
* The job model: $J_i = (A_i, C_i, D_i)$

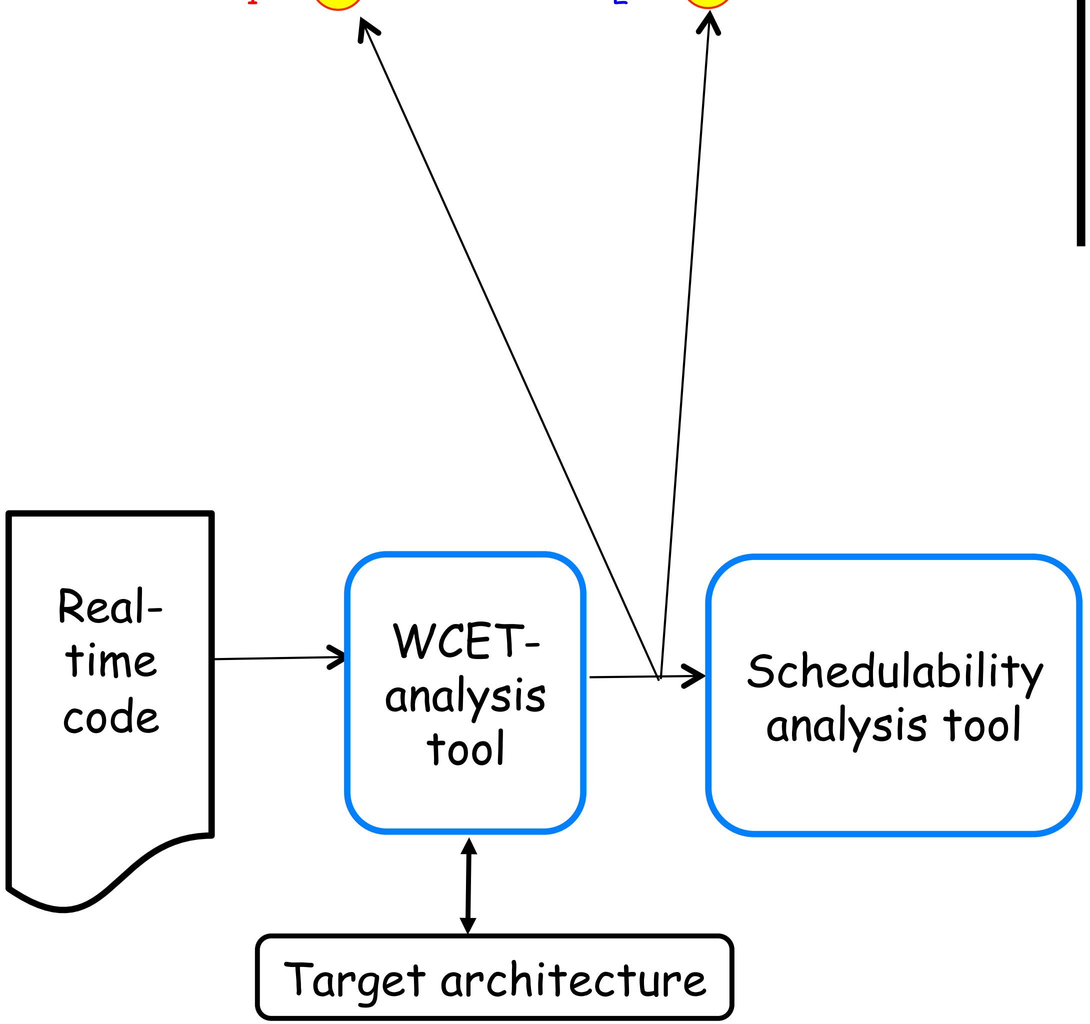


An example

2 jobs – J_1 and J_2 – on a preemptive processor
Both arrive at t=0; have deadlines at t=10 and t=8
WCET of J_1 is 4; WCET of J_2 is 4

* Only J_1 subject to certification
* The task model: $J_i = (A_i, C_i, D_i)$

* A_i and D_i: from requirement specs, and inter-job dependencies
* C_i: by worst-case execution time (WCET) analysis



An example

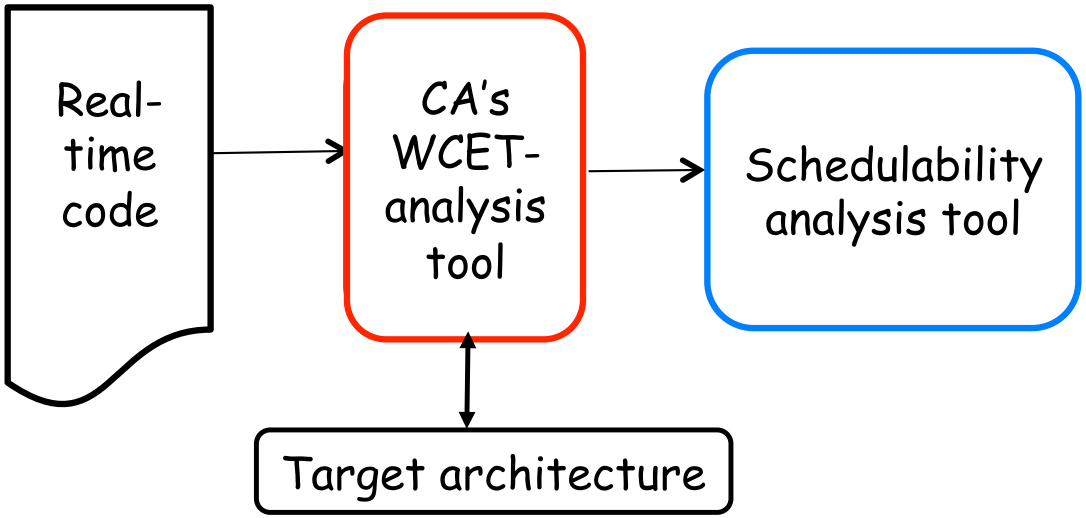
2 jobs – J_1 and J_2 – on a preemptive processor
Both arrive at t=0; have deadlines at t=10 and t=8
WCET of J_1 is 4; WCET of J_2 is 4

CA's tool more pessimistic

-E.g., based on worst-case analysis

(Designer's tool may use simulation experiments)

CERTIFICATION:



An example

2 jobs – J_1 and J_2 – on a preemptive processor
Both arrive at t=0; have deadlines at t=10 and t=8
WCET of J_1 is 4; WCET of J_2 is 4

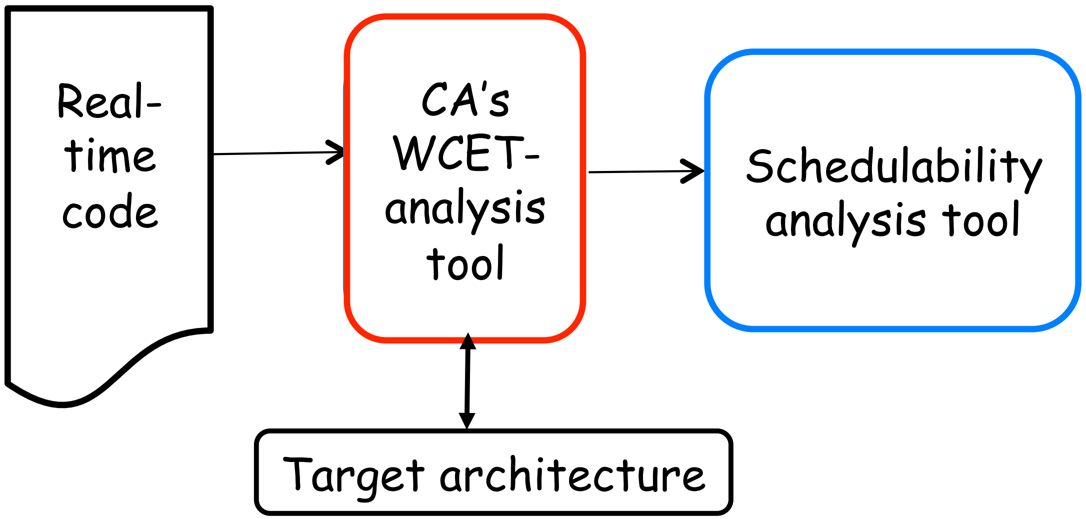
By CA's tool
WCET of J_1 is 8; WCET of J_2 is 6

CA's tool more pessimistic

Determined by sys. designer's tool

$\boxed{8 + 4 = 12 > 10}$

CERTIFICATION:



An example

2 jobs – J_1 and J_2 – on a preemptive processor
Both arrive at t=0; have deadlines at t=10 and t=8
WCET of J_1 is 4; WCET of J_2 is 4

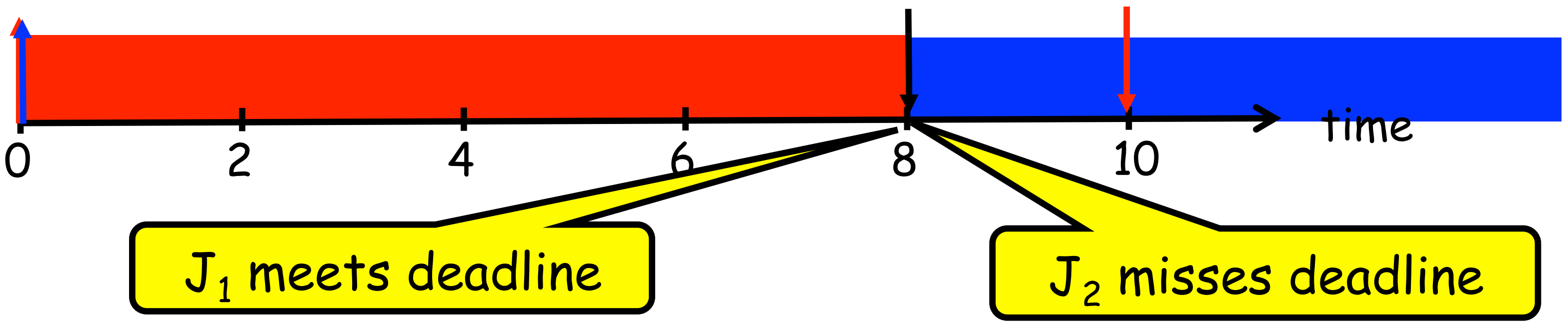
Determined by sys. designer's tool

By CA's tool
WCET of J_1 is 8; WCET of J_2 is 6

Only J_1 is subject to certification

CERTIFICATION: system passes certification

Priority-based scheduling: J_1 > J_2



An example

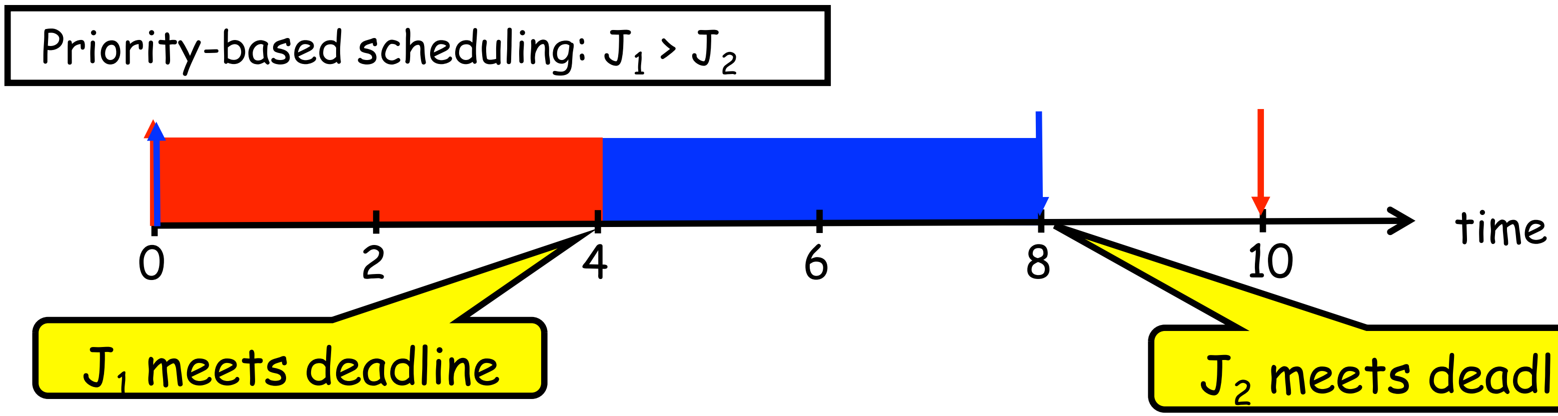
2 jobs – J_1 and J_2 – on a preemptive processor
Both arrive at t=0; have deadlines at t=10 and t=8
WCET of J_1 is 4; WCET of J_2 is 4

By CA's tool
WCET of J_1 is 8; WCET of J_2 is 6

Determined by sys. designer's tool

Both J_1 and J_2 should meet their deadlines

DESIGN VALIDATION: system validated correct



MIXED CRITICALITY (MC) systems

The same system is being analyzed, twice

Certification	System design 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, methods, and metrics for MC scheduling?

PRESENTATION PLAN

- A model for representing simple MC workloads
- An algorithm for scheduling such MC systems
- A metric for quantifying the effectiveness of this algorithm
- Generalizations to the model
- Algorithms for scheduling in these generalized models
- Evaluating these algorithms

The mixed-criticality job model

Job J_i

scheduling window

Level	Failure Condition	Interpretation
A	Catastrophic	Failure may cause a crash
B	Hazardous	Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the plane due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers
C	Major	Failure is significant, but has a lesser impact than a Hazardous failure (for example, leads to passenger discomfort rather than injuries)
D	Minor	Failure is noticeable, but has a lesser impact than a Major failure (for example, causing passenger inconvenience or a routine flight plan change)
E	No Effect	Failure has no impact on safety, aircraft operation, or crew workload.

ime

Previous example: 2 criticalities

- needs certification; does not need certification

Civilian aviation (DO-178B): 5 criticalities

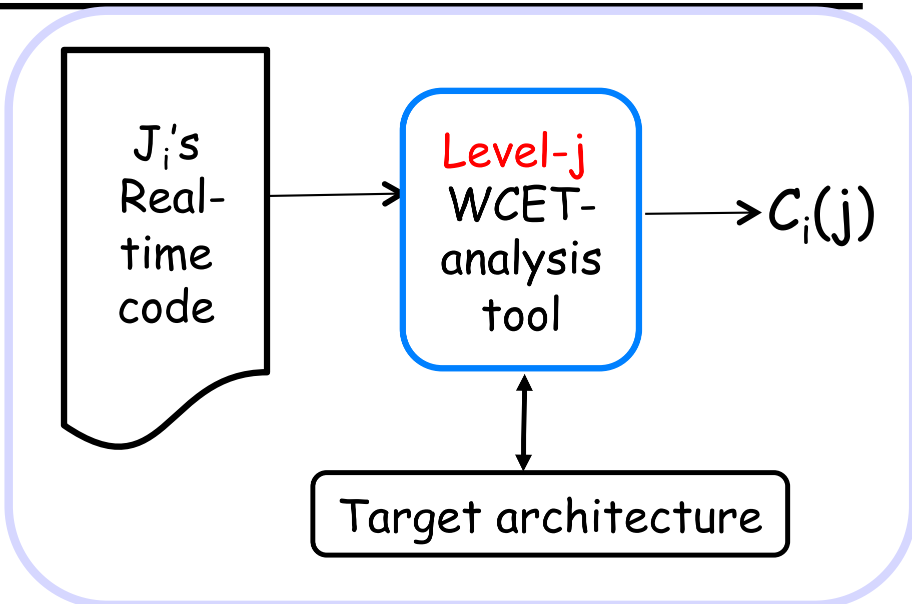
-catastrophic; hazardous; major; minor; no effect

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 $C_i(1), C_i(2), \ldots$



$C_i(j)$: 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

- upper bounds: the greater the desired degree of confidence, the larger the value

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

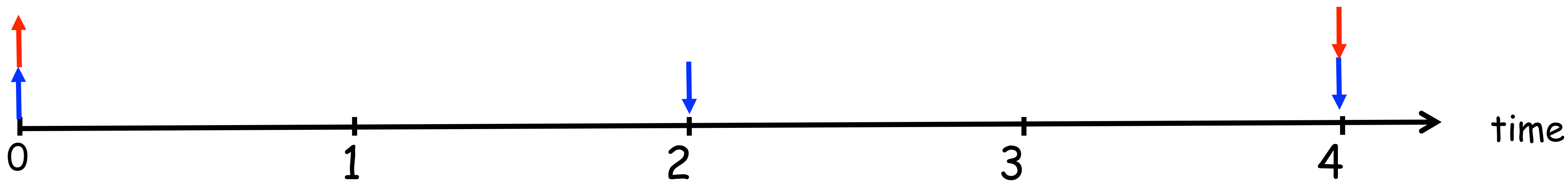
MC scheduling: An example

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

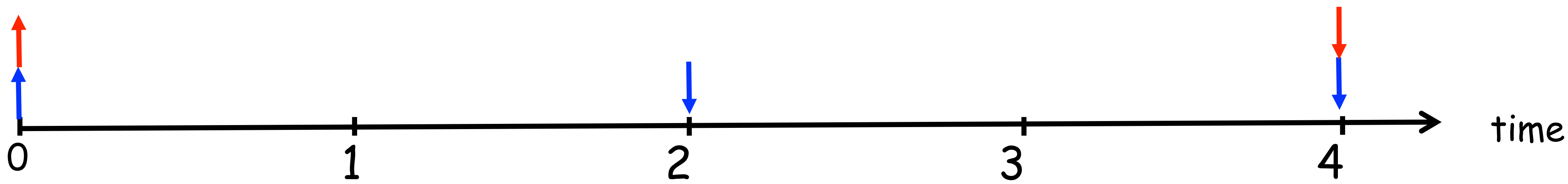
J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	1	2
J_2:	LO	0	1	2	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	2	4



Schedule for LO-criticality behavior - Earliest Deadline First (EDF)
Schedule for HI-criticality behavior - Criticality Monotonic scheduling

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	2	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	2	4



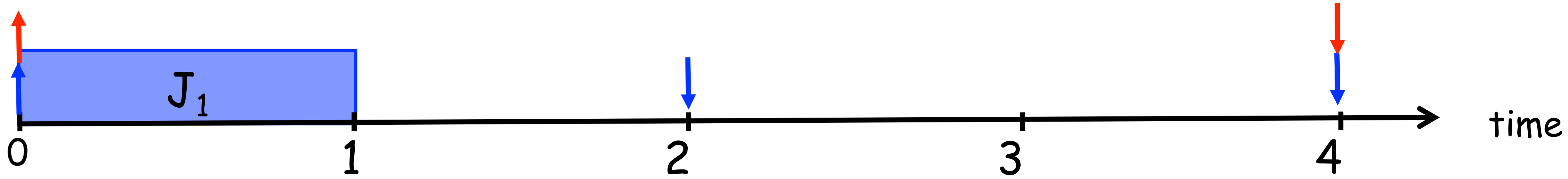
Schedule for LO-criticality behavior ✓
Schedule for HI-criticality behavior ✓

Single scheduling strategy for both 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	2	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	2	4

Earliest Deadline First (EDF)

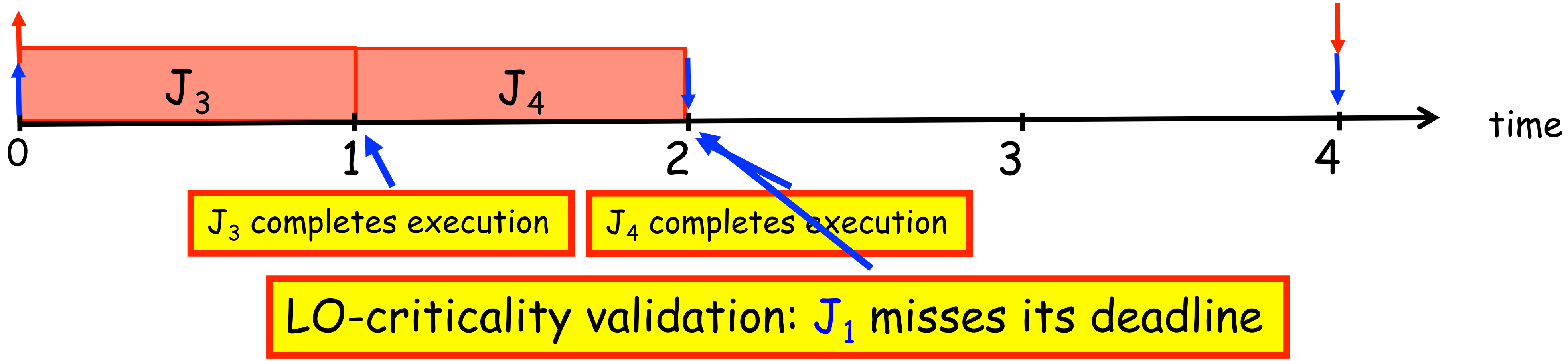


HI-criticality certification: must fit 4 units of work here

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	2	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	2	4

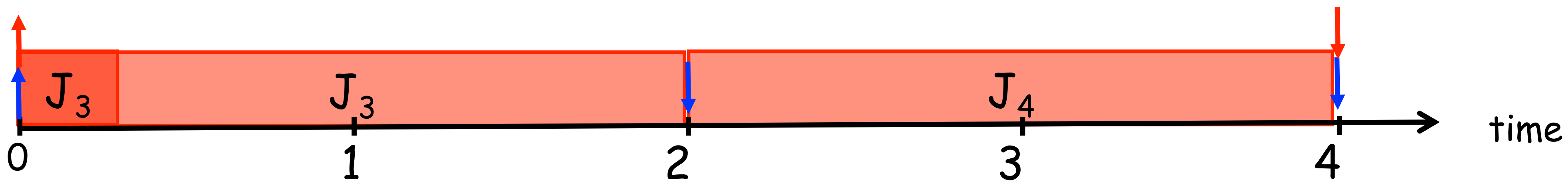
Criticality-Monotonic



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	2	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:

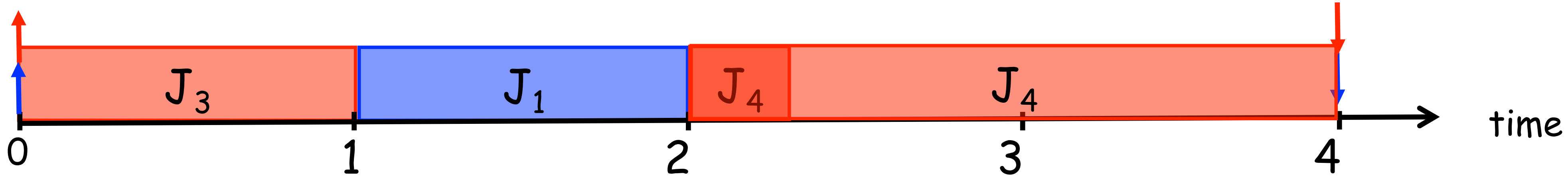


MC scheduling: An example

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	2	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 <u>not</u> complete by 3:



MC scheduling: An example

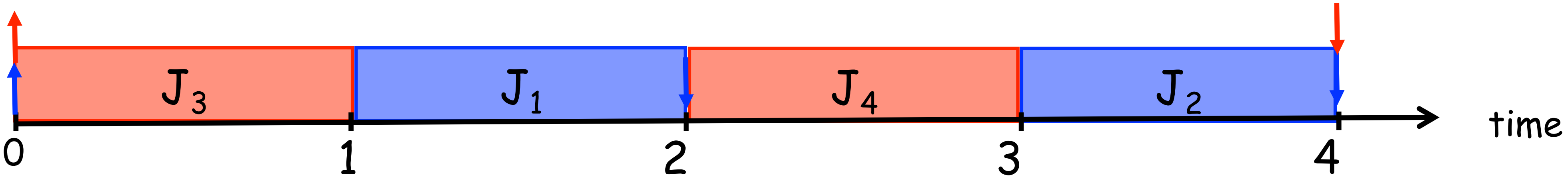
J_i:	L_i	A_i	$C_i(LO)$	$C_i(HI)$	D_i
J_1:	LO	0	1	1	2
J_2:	LO	0	1	2	4
J_3:	HI	0	1	2	4
J_4:	HI	0	1	2	4

A correct strategy:

- 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
- Already intractable
- All techniques & results generalize to more criticality levels

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)

1. find a lowest-priority job
2. remove from instance
3. repeat on remaining instance

[Proof of correctness: On preemptive processors, lower-priority jobs do not impact the scheduling of higher-priority jobs.]

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)

```
    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

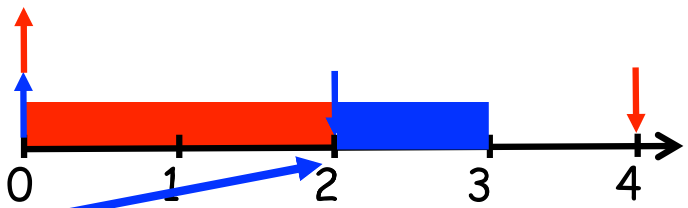
```
J_i := a job that may be assigned lowest priority in I'
```

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'
```

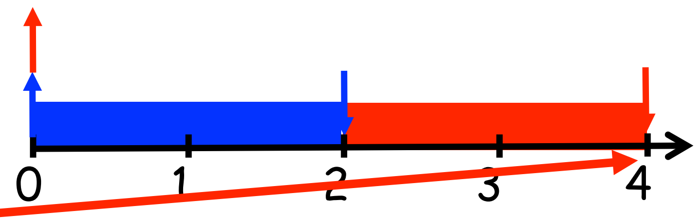
J_i may be assigned lowest priority if it meets its deadline as the lowest-priority job, when each job J_k executes for $C_k(L_i)$ time units

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′
```

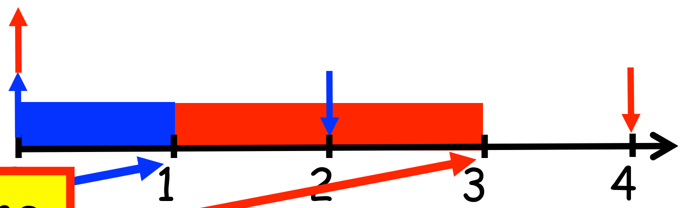
J_i may be assigned lowest priority if it meets its deadline as the lowest-priority job, when each job J_k executes for $C_k(L_i)$ time units

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

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

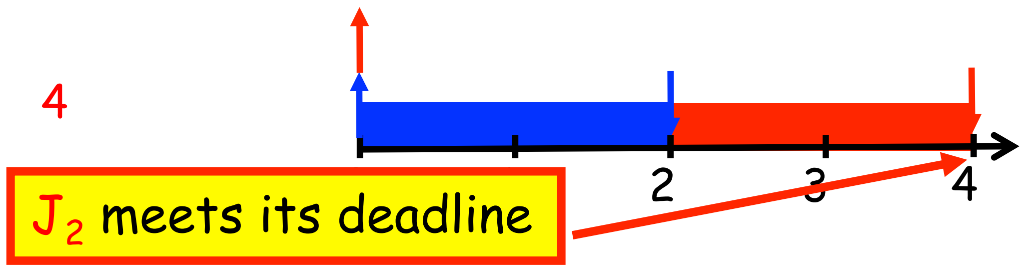
J_i may be assigned lowest priority if it meets its deadline as the lowest-priority job, when each job J_k executes for $C_k(L_i)$ time units

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; $O(n^2)$

* Is a sufficient (not exact) scheduling algorithm

*Quantitative performance bound
(assuming some run-time support)

A quantitative metric - Speedup factor

Schedulable instance I → A polynomial-time algorithm → A schedule for I on <u>faster</u> procs.

NP-hard: Such an algorithm is unlikely

So, seek an approximate algorithm that has polynomial run-time

* Faster processors to compensate for non-optimality of the algorithm

Definition. A scheduling algorithm A has speedup factor equal to s ($s \geq 1$) if any instance that can be scheduled by an optimal algorithm on unit-speed processors, can be scheduled by algorithm A on speed-s processors

* Comparing approximation algorithms: smaller s is better

(An optimal algorithm has $s = 1$)

We seek polynomial-time scheduling algorithms with small speedup factors

Integrated computing environments + certification requirements

Models - finite collection of independent jobs

Methods - OCBP (Own Criticality-Based Priorities)

Metrics - Processor Speedup Factor

PRESENTATION PLAN:

A processor speedup factor for OCBP

Generalizing the model: recurrent task systems

- an algorithm for recurrent task systems: EDF-MD

- a processor speedup factor for EDF-MD

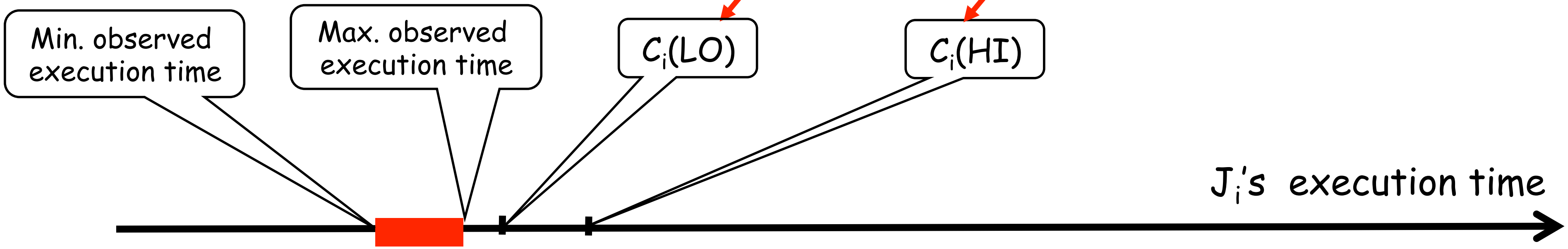
Further generalizing the model: critical sections

WCET: a closer look

LO criticality HI criticality WCET

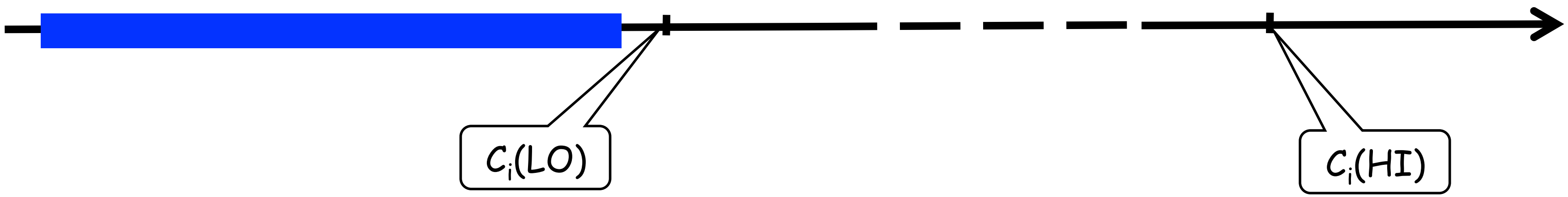
Safety-critical programs (should) exhibit predictable behavior

- -Simple control structures; bounded loops; restricted use of cache, etc.
- Average behavior similar to worst-case behavior



"Regular" programs may be more complex

- -Greater unpredictability on behavior; greater variation in run-times



$C_i(\mathrm{HI}) \gg C_i(\mathrm{LO})$ for a less predictable job

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

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~~

If run-time system can enforce execution budgets

$$\text{Budget assigned to } J_i = \begin{cases} C_i(LO), \text{ if } J_i \text{ is a LO-criticality job} \\ C_i(HI), \text{ if } J_i \text{ is a HI-criticality job} \end{cases}$$

$\boxed{C_i(HI) = C_i(LO) \text{ for LO-criticality job } J_i}$

But…

- such systems tend to be more complex
- policing and budgeting overhead costs must be accounted for
- policing and budget-enforcement must be implemented as HI-criticality functionalities

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 $[t_1, t_2)$

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

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

*load_{LO}(I) - load “expected” by system designer

(all jobs; LO-criticality WCET’s)

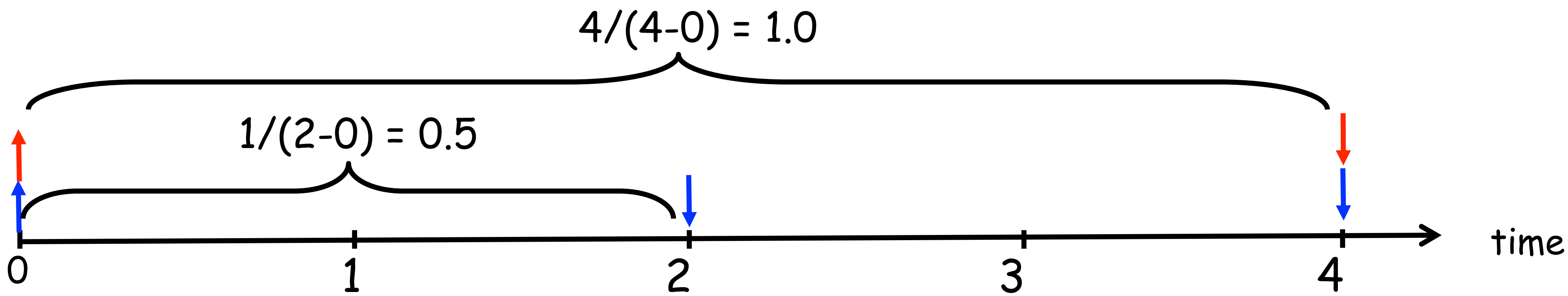
*load_{HI}(I) - load to be certified

(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

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

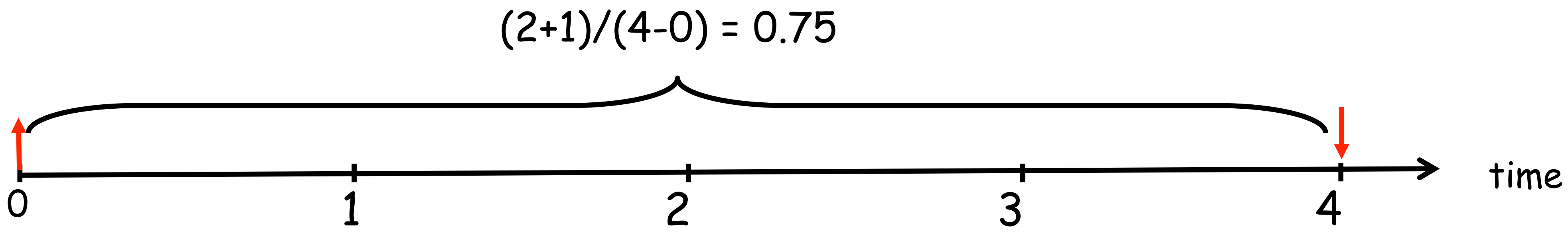


The load parameter: an example

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



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 $\text{load}_{LO}(I) = 1.00$

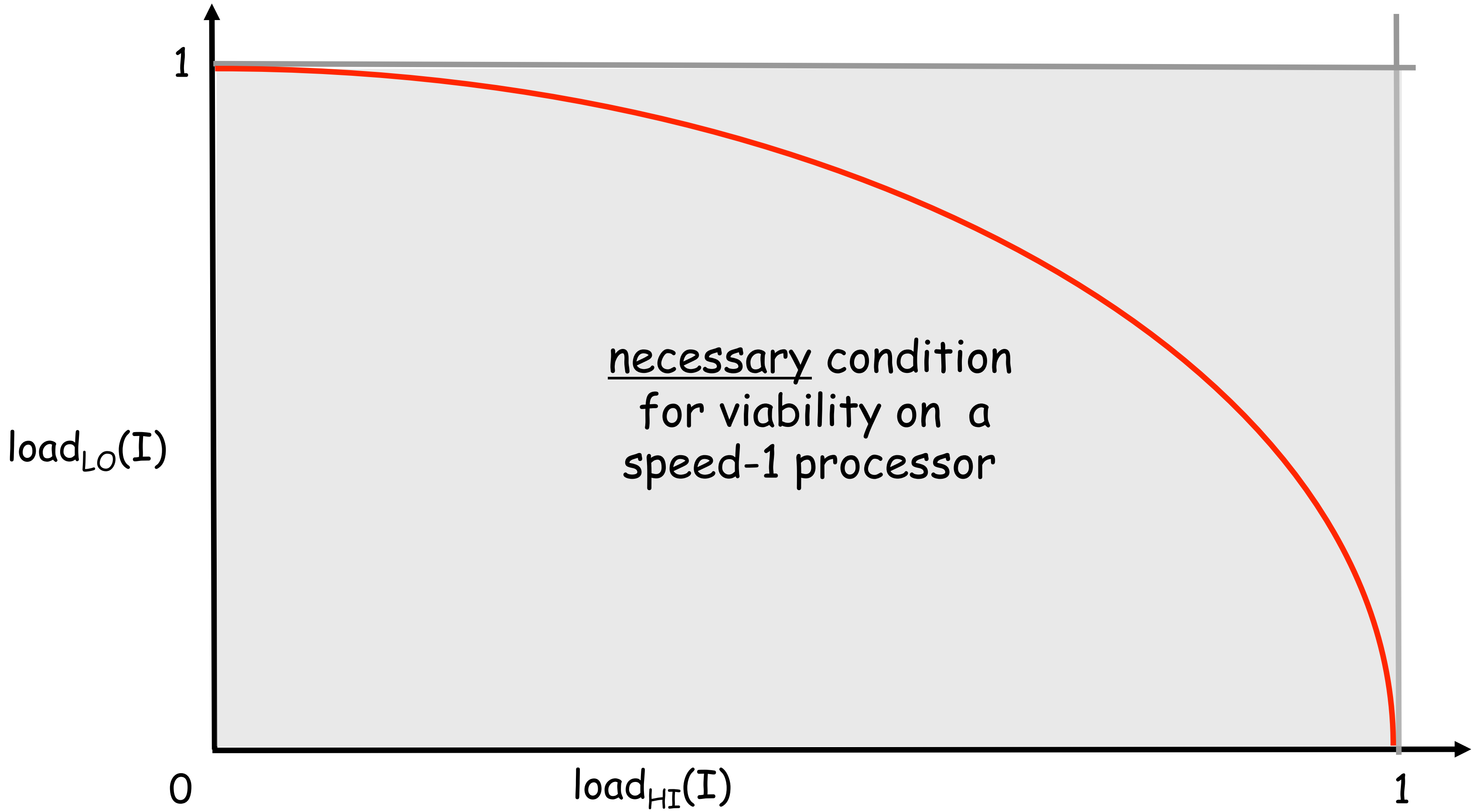
and high-criticality load $\text{load}_{HI}(I) = 0.75$

OCBP: A sufficient schedulability condition

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

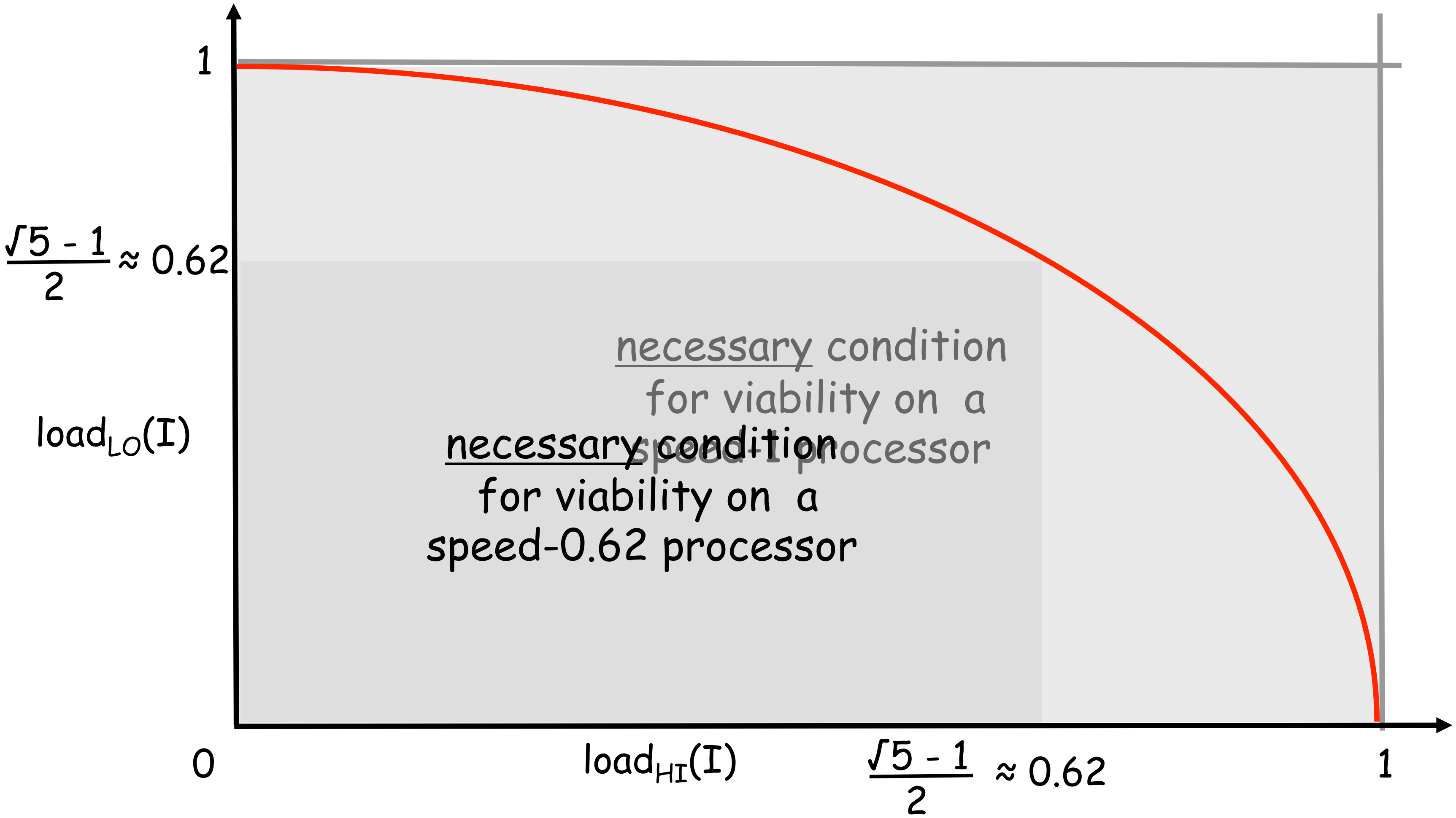
$$\text{load}_{HI}(I) + \text{load}_{LO}(I)^2 \leq 1$$

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-$\frac{2}{\sqrt{5}-1} = \frac{\sqrt{5}+1}{2}$ (≈ 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-$\frac{2}{\sqrt{5} - 1} = \frac{\sqrt{5} + 1}{2}$ ($\approx$ 1.618) processor

The Golden Ratio: positive solution to $x^2 = (x + 1)$

Recurrent tasks

Recurring tasks or processes

- generate jobs
- represent code within an infinite loop

Different tasks are assumed independent

```
for(;;){
      •
      •
      •
      •
      •
}
```

Recurrent tasks: the Liu & Layland (LL) model

Task τ_i = (T_i, L_i, [C_i(LO), C_i(HI)])

- T_i: minimum inter-arrival separation ("period")
- $L_i \in$ {LO, HI}: the criticality level of the task
- C_i(LO), C_i(HI): WCET estimates, at both criticality levels

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 must complete within T_i time units

The dual-criticality scheduling problem for LL task systems: Given a collection { τ_1, τ_2, …, τ_n} of dual-criticality LL tasks, determine a correct scheduling strategy

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 $[t_1, t_2)$

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

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

The utilization parameter of a LL task system

For systems of (non mixed-criticality) LL tasks:

demand(I, $[t_1, t_2)$) ≡ cumulative execution requirement of jobs of instance I ove)

$$U(\tau) = \sum_{\tau_i \in \tau} \frac{C_i}{T_i}$$

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

RESULT: Any regular (i.e., non-MC) LL task system τ is feasible on a preemptive uniprocessor if and only if $U(\tau)$ is ≤ 1

The utilization parameter of a LL task system

For systems of (non mixed-criticality) LL tasks:

$$U(\tau) = \sum_{\tau_i \in \tau} \frac{C_i}{T_i}$$

RESULT: Any regular (i.e., non-MC) LL task system τ is feasible on a preemptive uniprocessor if and only if $U(\tau)$ is ≤ 1

Generalization to dual-criticality LL systems

* $U_{LO}(\tau)$ - as "expected" by system designer (<u>all</u> tasks; LO-criticality WCET's)

$$U_{\mathrm{LO}} = \sum_{all\ \tau_i} \frac{C_i(\mathrm{LO})}{T_i}$$

* $U_{HI}(\tau)$ - to be certified (only HI-crit. tasks; HI-crit. WCET's)

$$U_{\mathrm{HI}} = \sum_{L_i = \mathrm{HI}} \frac{C_i(\mathrm{HI})}{T_i}$$

Scheduling dual-criticality LL tasks on preemptive uniprocessors

Extensions of OCBP to the recurrent tasks model

- yields a speedup bound of ≈ 1.62
- quadratic run-time per scheduling decision

1. Li and Baruah. An algorithm for scheduling certifiable mixed-criticality task systems. RTSS 2010
2. Guan, Ekberg, Stigge and Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. RTSS 2011

Scheduling dual-criticality LL tasks on preemptive uniprocessors

Extensions of OCBP to the recurrent tasks model

- yields a speedup bound of ≈ 1.62
- quadratic run-time per scheduling decision

Earliest Deadline First – Modified Deadlines

EDF-MD: a new scheduling algorithm

- Better (smaller) speedup bound
- better run-time behavior

Algorithm EDF-MD: An example

	L_i	T_i	$C_i(LO)$	$C_i(HI)$
τ_1:	LO	6	3	3
τ_2:	HI	8	2	6

on a unit-speed processor

LO-criticality utilization: $3/6 + 2/8 = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$ $\Rightarrow$ can be scheduled to meet LO-criticality validation requirements

HI-criticality utilization $6/8 = \frac{3}{4}$ $\Rightarrow$ can be scheduled to meet HI-criticality certification requirements

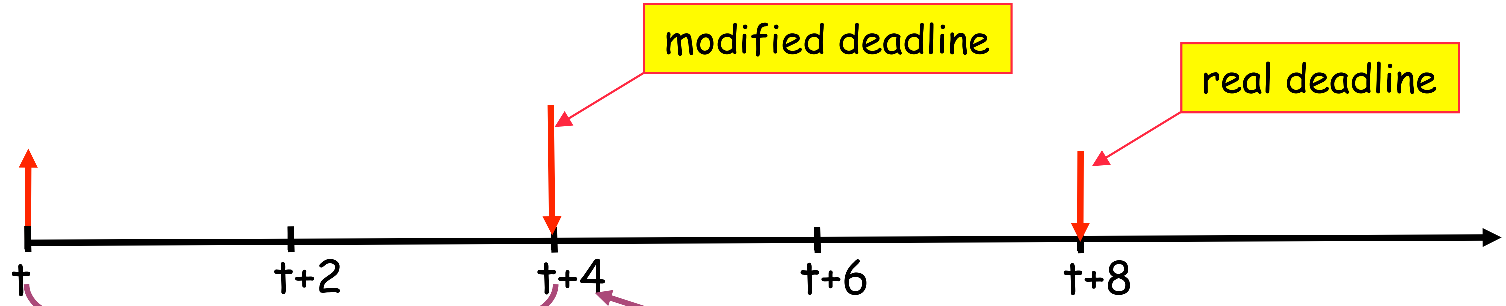
Algorithm EDF-MD: An example

	L_i	T_i	$C_i(LO)$	$C_i(HI)$
τ_1:	LO	6	3	3
τ_2:	HI	8 → 4	2	6

IDEA: Reduce the period of the HI-criticality tasks, while maintaining LO-criticality feasibility

LO-criticality utilization: $3/6 + \cancel{2/8}\,^{2/4} = \frac{1}{2} + \cancel{\frac{1}{4} = \frac{3}{4}}\ = 1$

A job of τ_2:



Receive 2 units of execution here…

either signal completion, or know it's a HI-criticality behavior

$\Rightarrow$

τ_1 does not need to execute any more

Algorithm EDF-MD

1. Pre-processing

   * Scale the periods of all HI-criticality tasks such that U_{LO} becomes 1

   * Scaling factor is $\Big(\sum_{L_i=HI} \frac{C_i(LO)}{T_i}\Big)/\Big(1-(\sum_{L_i=LO} \frac{C_i(LO)}{T_i})\Big)$

2. Initial run-time scheduling (assuming LO-criticality behavior)

   * Schedule according to EDF

     - job deadlines assigned according to the scaled-down periods

3. Run-time scheduling upon transitioning to HI-criticality

   - [i.e., some jobs executes beyond its LO-criticality WCET]

   * Discard all LO-criticality jobs

   * Recompute deadlines for HI-crit. jobs, according to their original periods

   * Future arrivals

     - LO-crit: discard

     - HI-crit: deadlines assigned according to the original periods

Algorithm EDF-MD: Properties

The processor speedup factor of Algorithm EDF-MD is 4/3

- Extended OCBP: ≈ 1.62

number of tasks

Algorithm EDF-MD can be implemented with a run-time complexity equal to O(log N) per scheduling decision

- Extended OCBP: $O(N^2)$ per scheduling decision

Recurrent tasks + shared resources

Workload: Dual-criticality LL tasks

Platform: preemptive uniprocessor
+ additional serially reusable resources

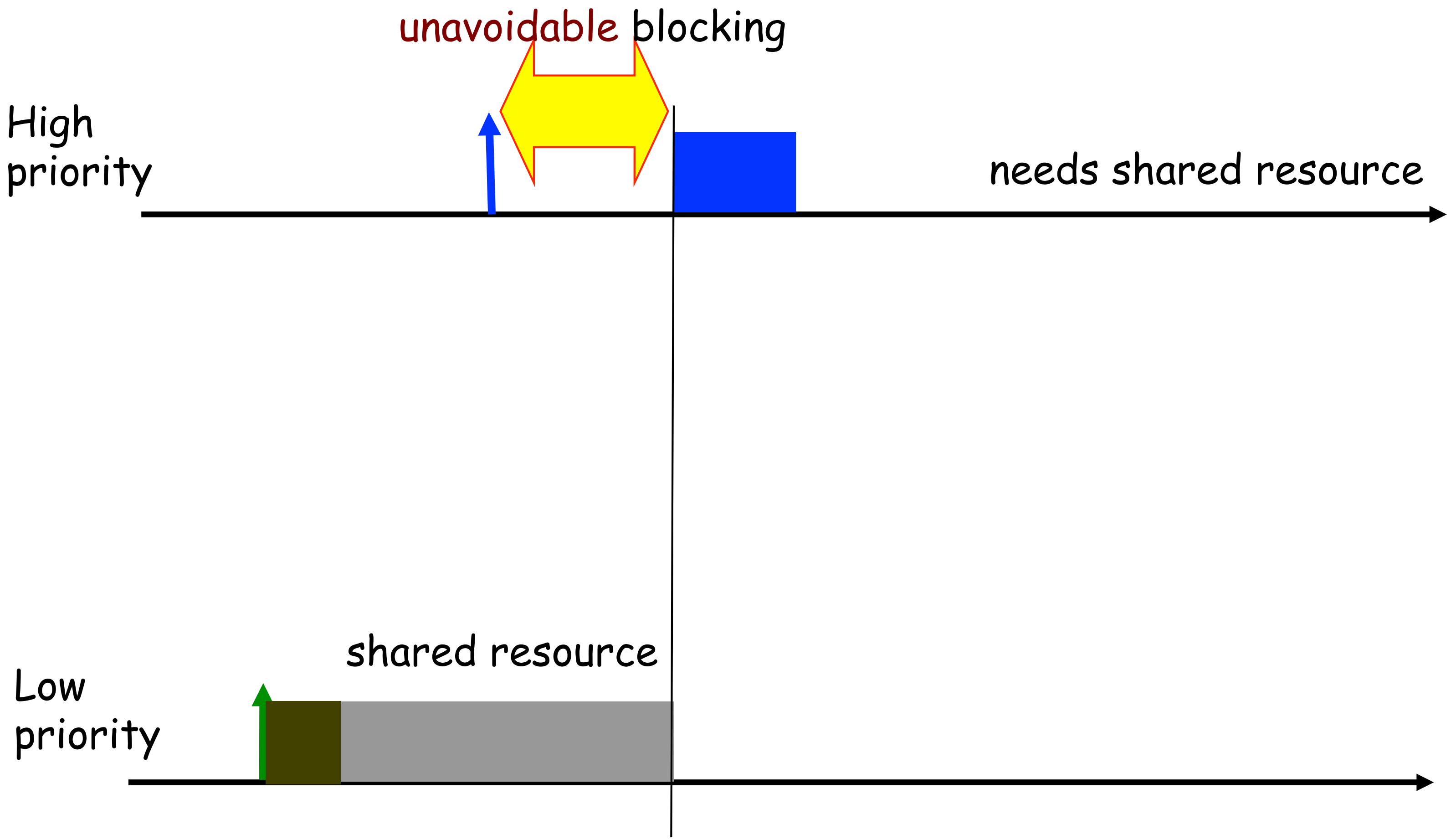
Jobs access shared resources
- within critical sections…which may be nested

Priority Inversion: A lower-priority job executes instead of a higher-priority one

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

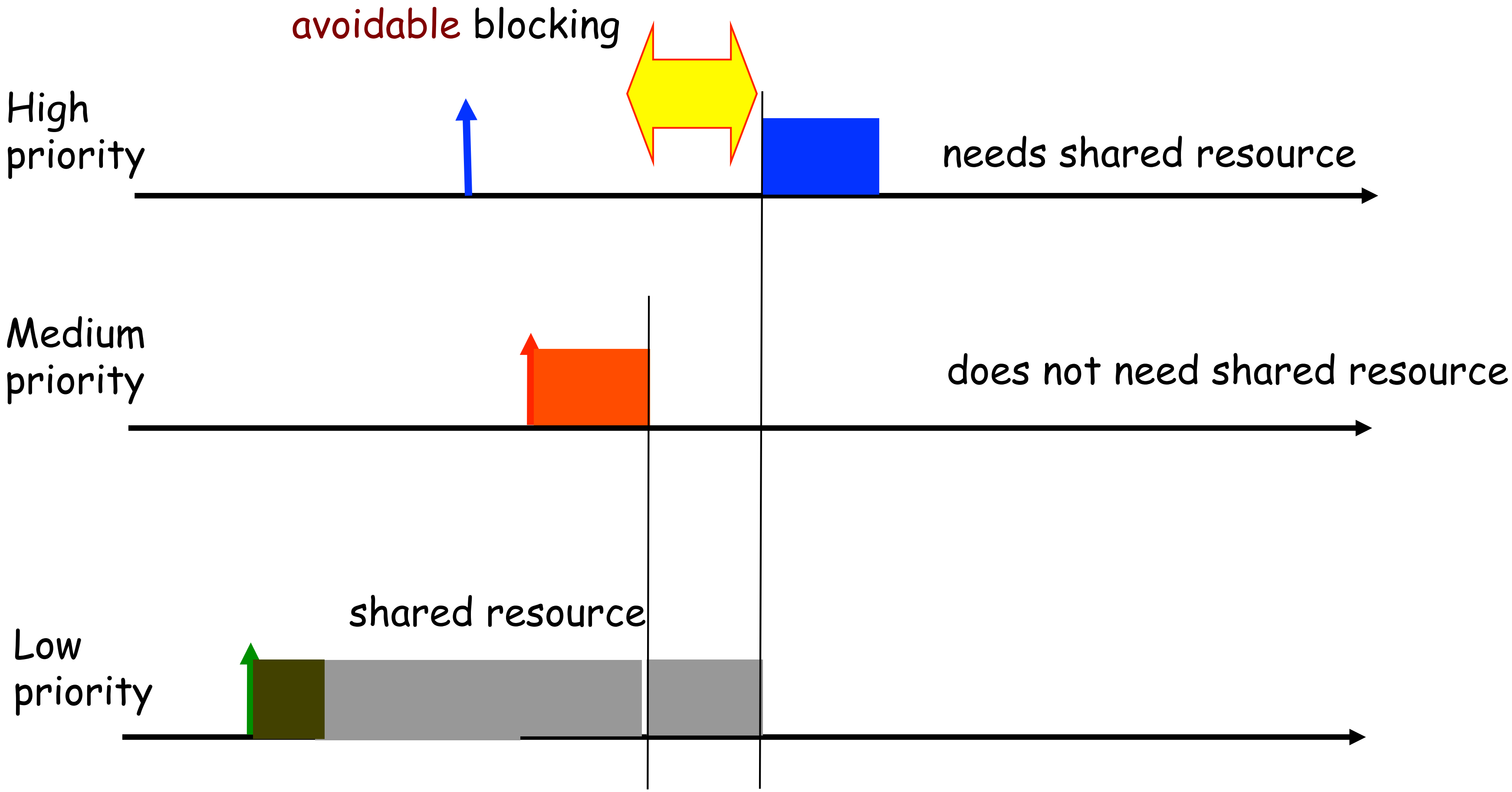
Serially reusable shared resources

Priority inversion and blocking



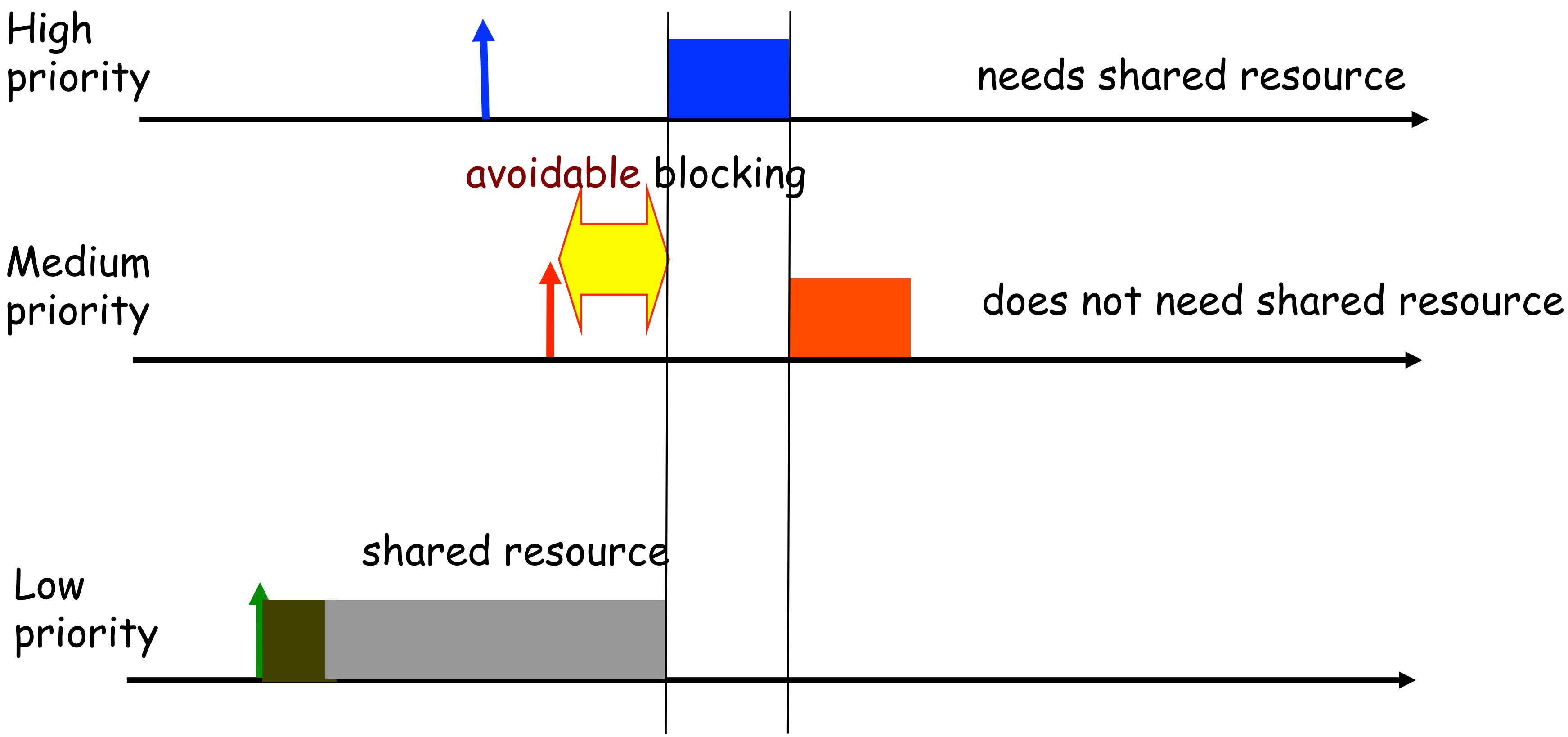
Serially reusable shared resources

Priority inversion and blocking



Serially reusable shared resources

Priority inversion and blocking



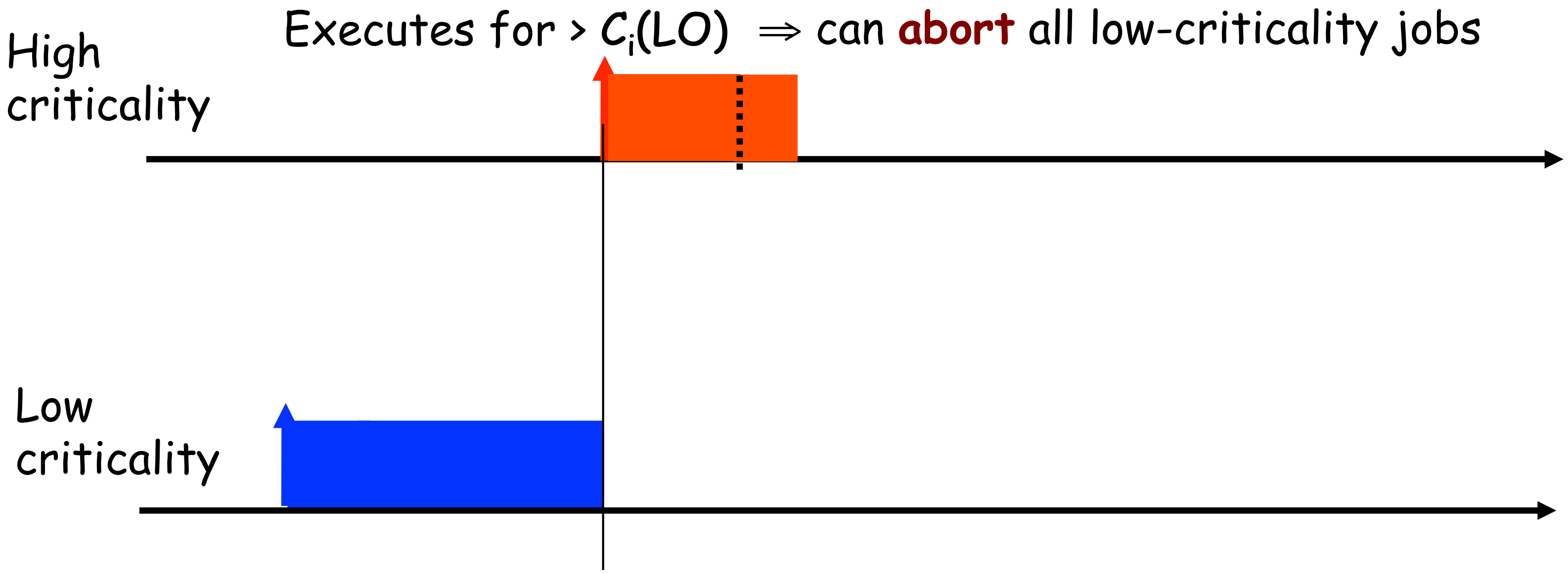
Serially reusable shared resources

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" L&L task systems: if any task system is uniprocessor feasible, then EDF + SRP guarantees to schedule it to meet all deadlines

Serially reusable shared resources

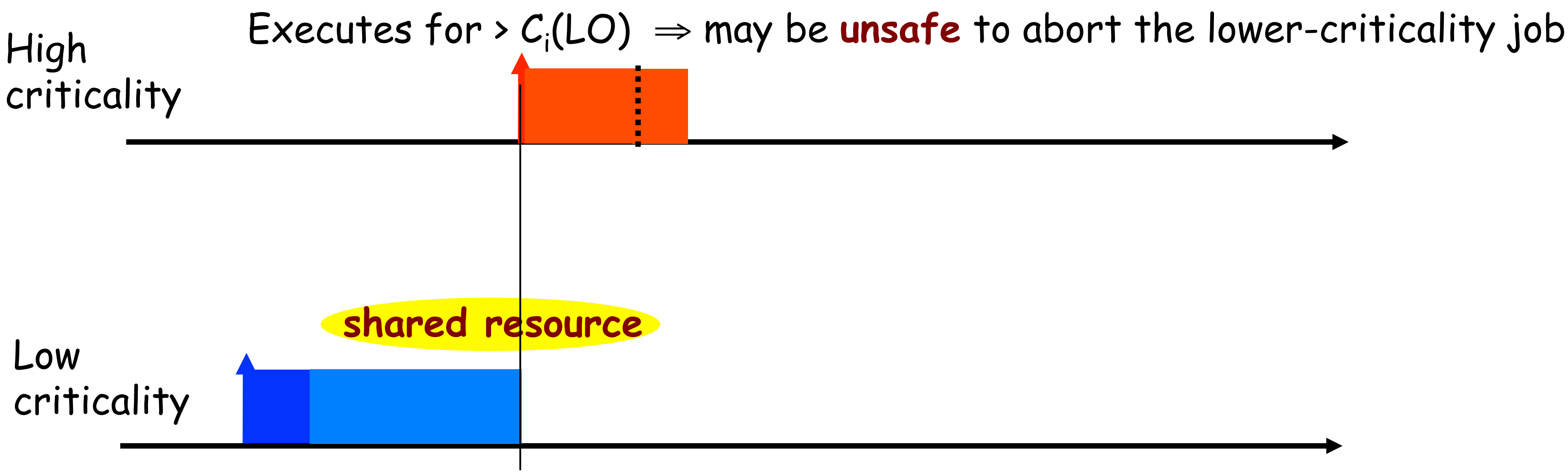
Mixed criticality scheduling without shared resources



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

Current practice: complete isolation amongst applications
is inefficient
- in resource usage: Size, Weight, and Power (SWaP)
- in certification effort

Needed: Certifiably correct techniques for system design and implementation

New models, methods, and metrics for achieving this goal