THE UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

# Towards the Design of Certifiable Mixed-Criticality Systems

Sanjoy Baruah, **Haohan Li**
*The University of North Carolina*

Leen Stougie
*Vrije Universiteit*

# Outline

- **Motivation**
  - Certification requirements in embedded systems
- **Model**
  - Definition of mixed-criticality system
  - Hardness of feasibility test
- **Solution**
  - Why EDF and criticality-monotonic fail
  - OCBP: A new algorithm

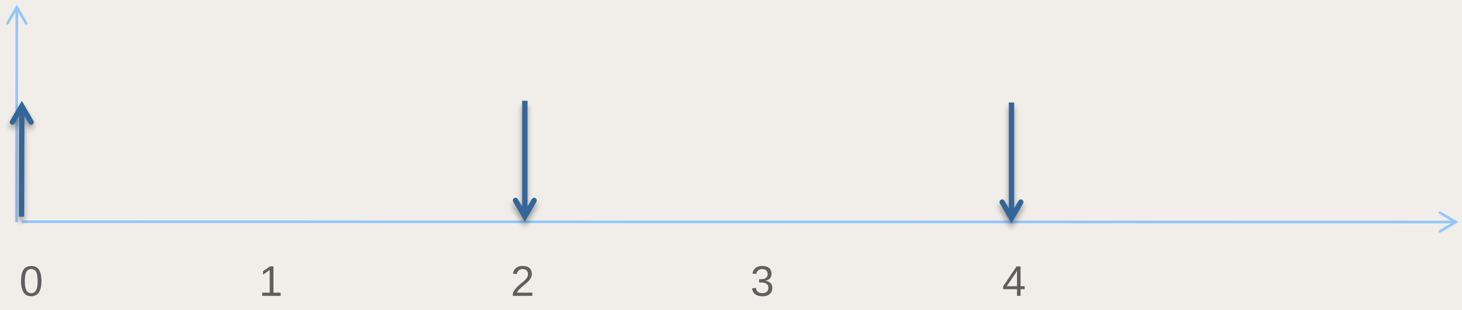
# Motivation

- An example for classic real-time jobs
  - Uniprocessor
  - Preemptive
  - Hard real-time
  - Given finite instance of jobs
    - One-pass job set
    - Specified release times and deadlines

# Example

- An example for classic real-time jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 1 |
| $J_4$ | 0 | 4 | 1 |



# Example

- An example for classic real-time jobs

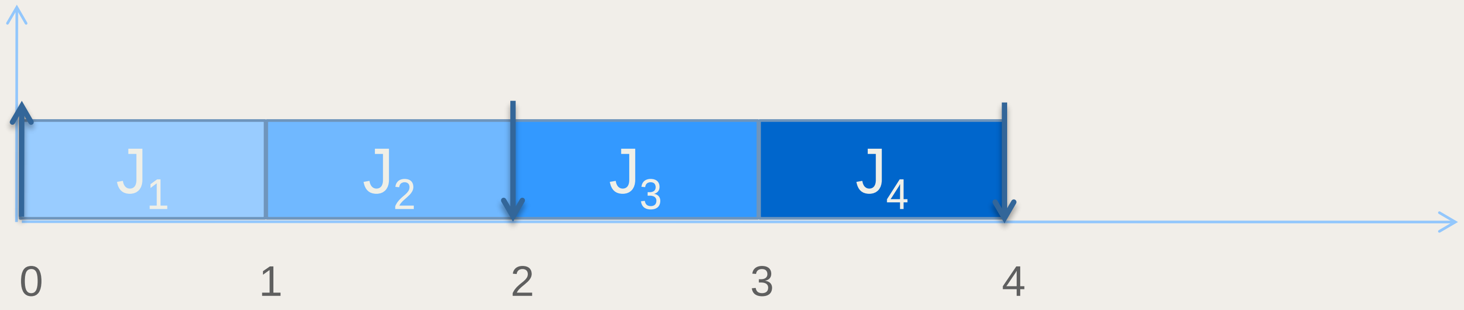
| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 1 |
| $J_4$ | 0 | 4 | 1 |

- We can schedule them using earliest-deadline-first(EDF) strategy optimally

# Example

- An example for classic real-time jobs

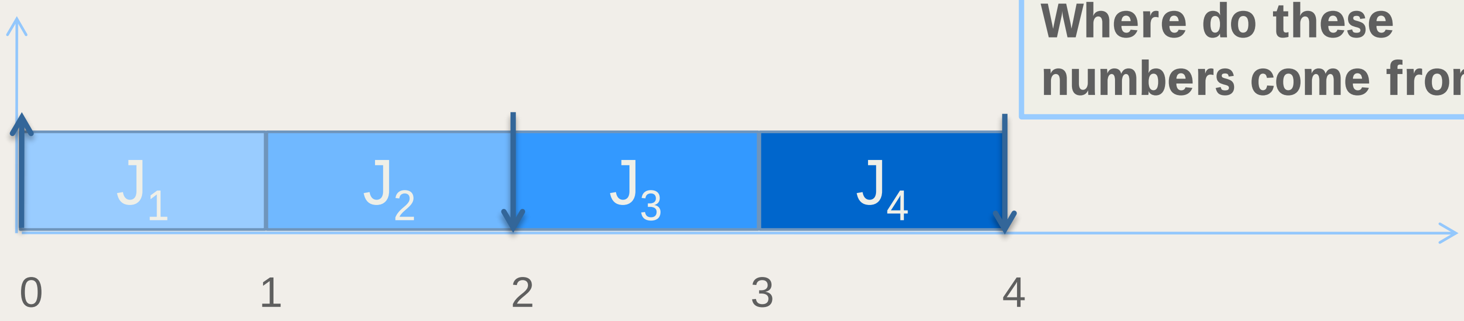
| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 1 |
| $J_4$ | 0 | 4 | 1 |



# Example

- An example for classic real-time jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 1 |
| $J_4$ | 0 | 4 | 1 |




Where do these numbers come from?

# Motivation

- Execution time is estimated
- With different tools we'll get different estimations
- Sometimes a part of the system must pass certification from authorities. They will simulate the system to check validity.
- Authorities may use more pessimistic estimations

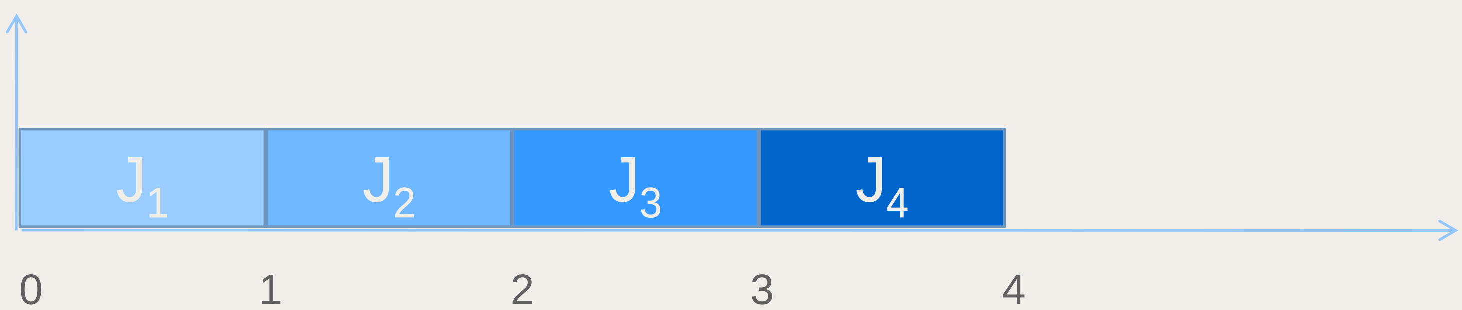
# Motivation

- An example for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 1 |
| $J_4$ | 0 | 4 | 1 |

Not critical, like camera, radio, heater

Safety-critical, like flight control system

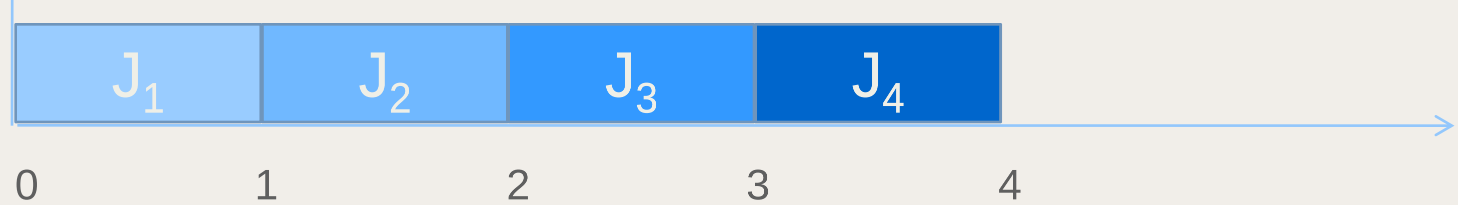


# Motivation

- An example for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | ⇒ 2 |
| $J_4$ | 0 | 4 | ⇒ 2 |

**The authorities may estimate differently**

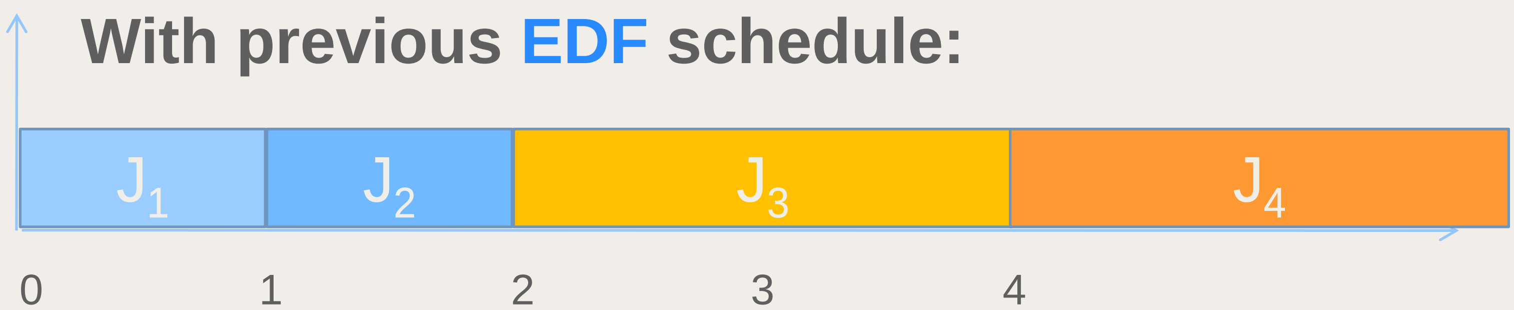


# Motivation

- An example for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

**With previous EDF schedule:**

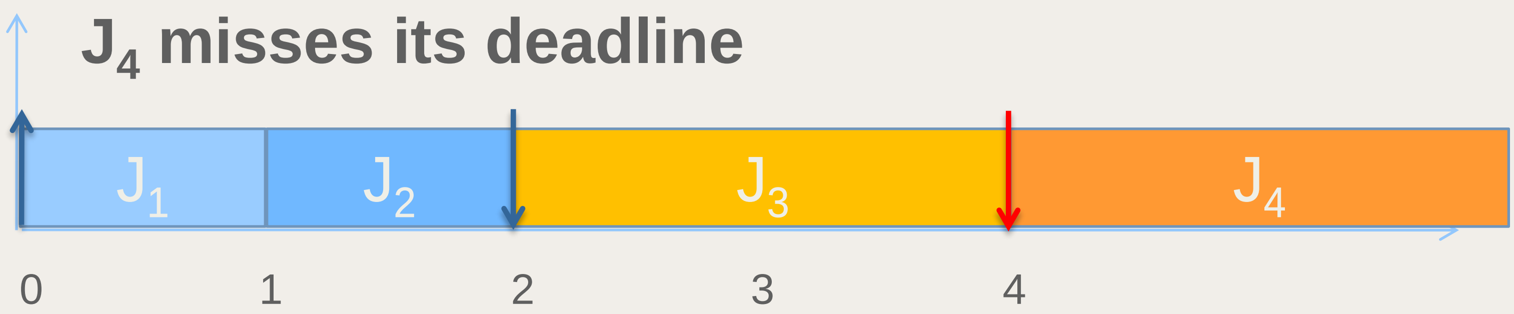


# Motivation

- An example for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

**$J_4$ misses its deadline**



# Motivation

- An example for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

**These two jobs don't have to be certified**

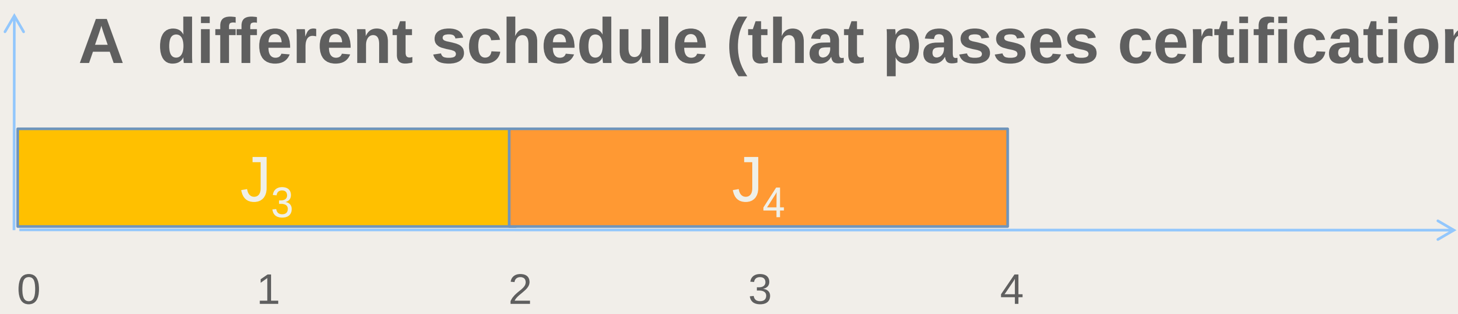
# Motivation

- An example for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

**These two jobs don't have to be certified**

**A different schedule (that passes certification):**



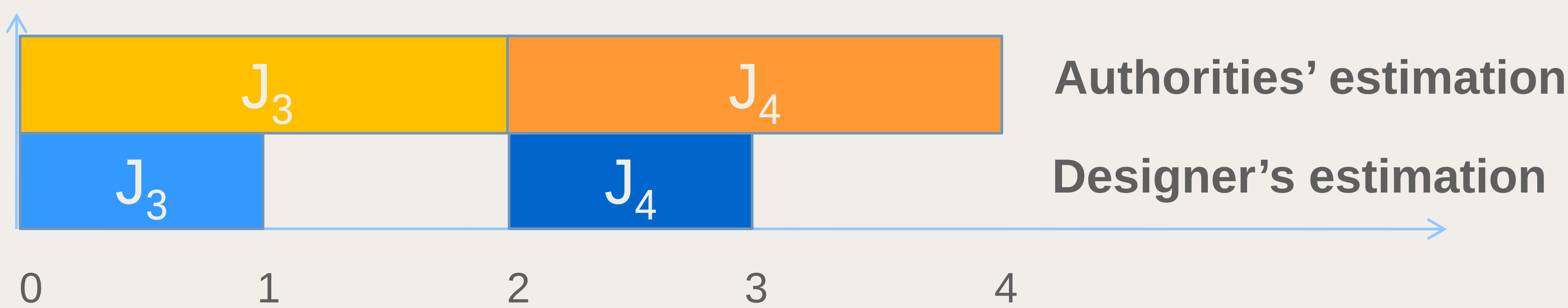
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

# Motivation

- An example for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

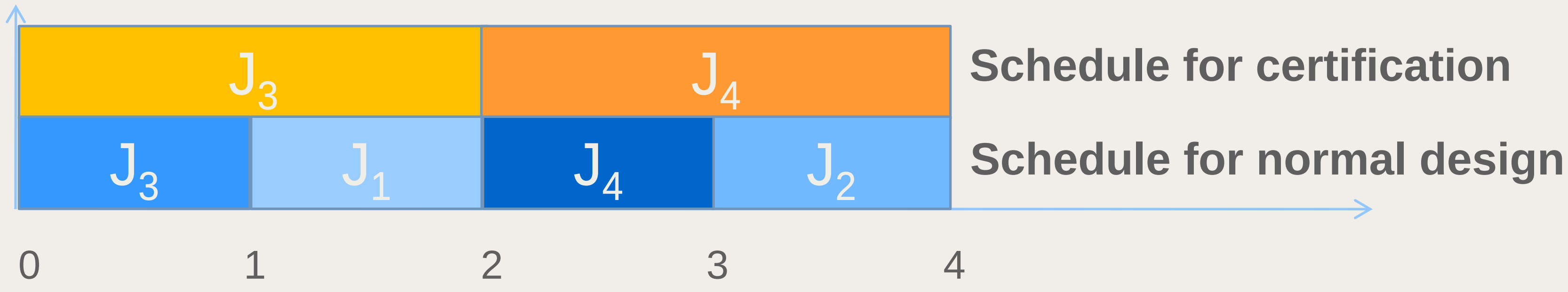
**These two jobs don't have to be certified**



# Motivation

- An example for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

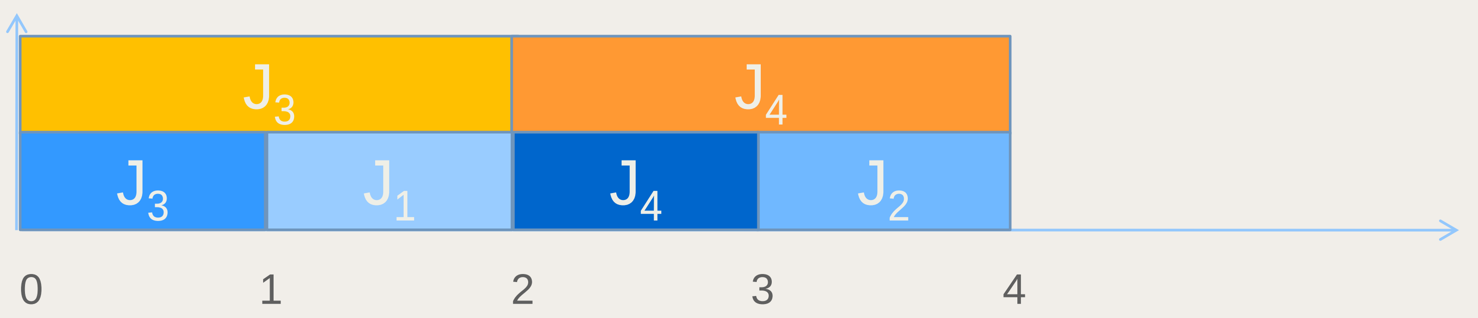


# Motivation

- An example for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

**Can we use one strategy to solve the problem?**

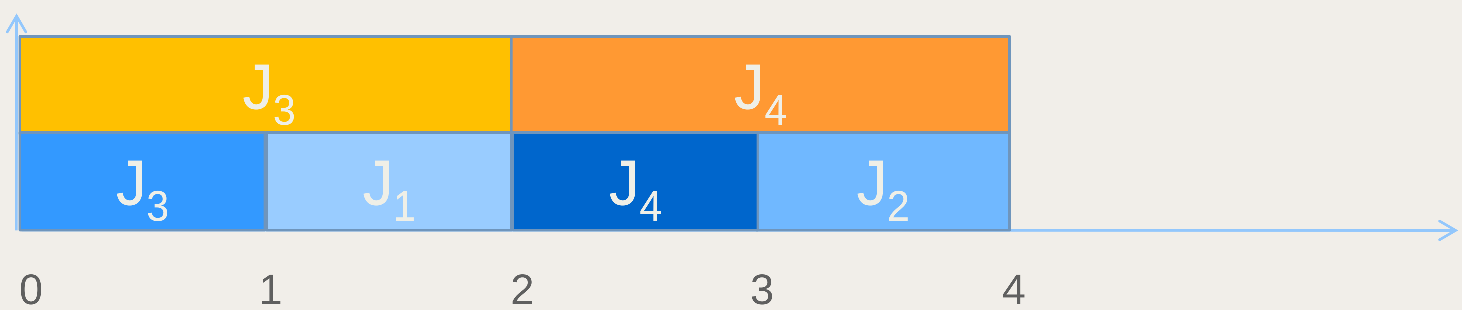


# Motivation

- An example for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

**Yes, we can, by using a priority + criticality strategy (in this example).**



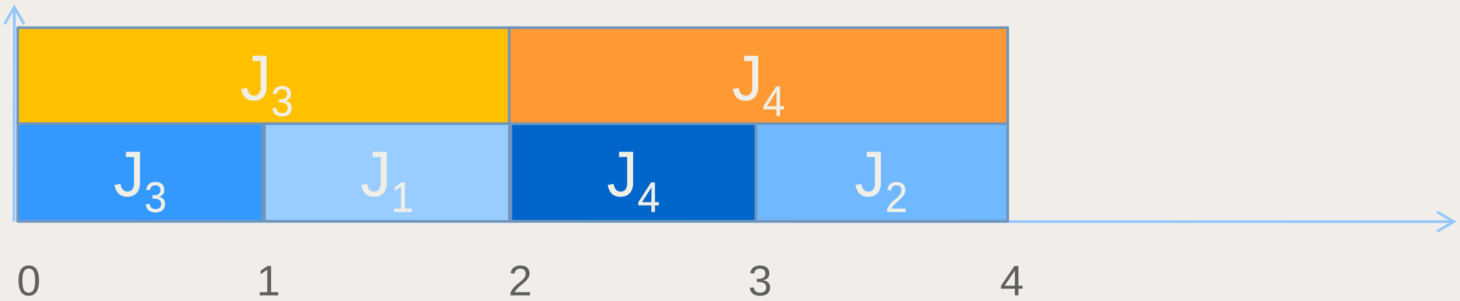
# Motivation

- An example for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

**A feasible static priority list:**

**$J_3 > J_1 > J_4 > J_2$**



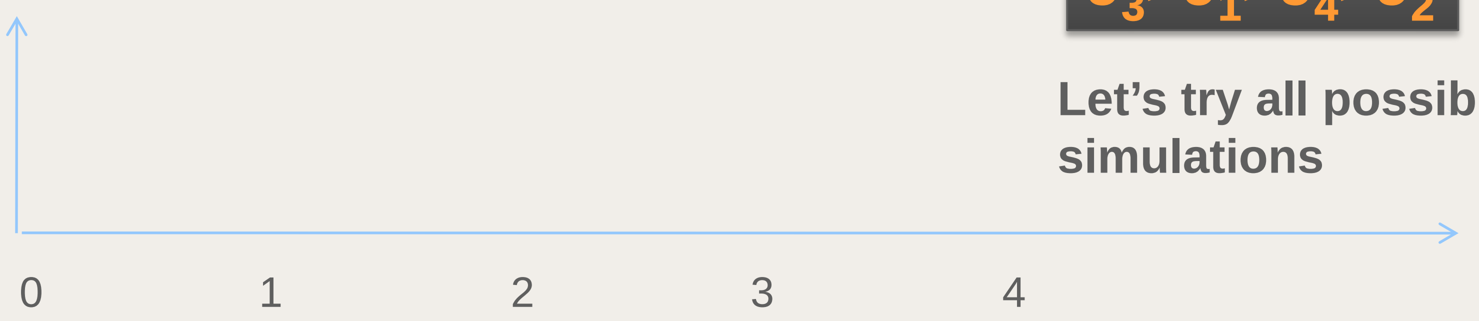
# Mixed-Criticality Schedule

- A solution for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

$J_3 > J_1 > J_4 > J_2$

Let's try all possible simulations

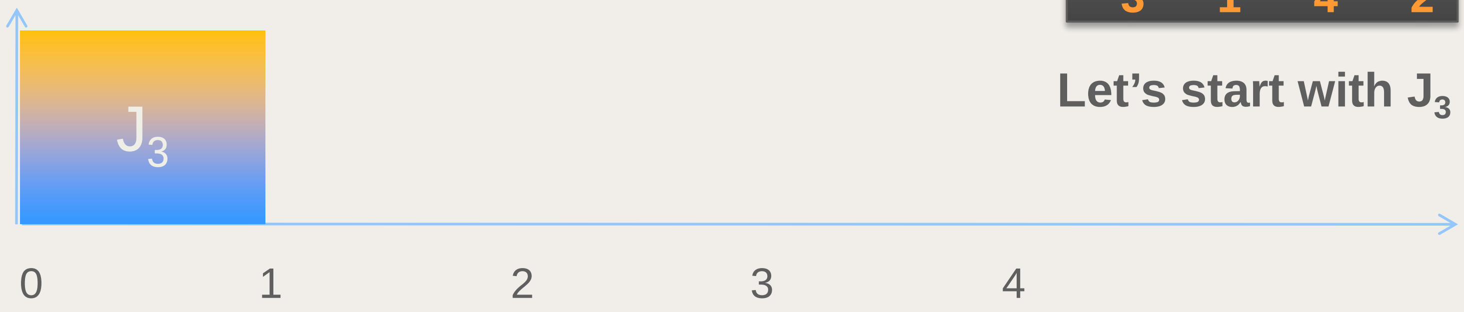


# Mixed-Criticality Schedule

- A solution for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

$J_3 > J_1 > J_4 > J_2$



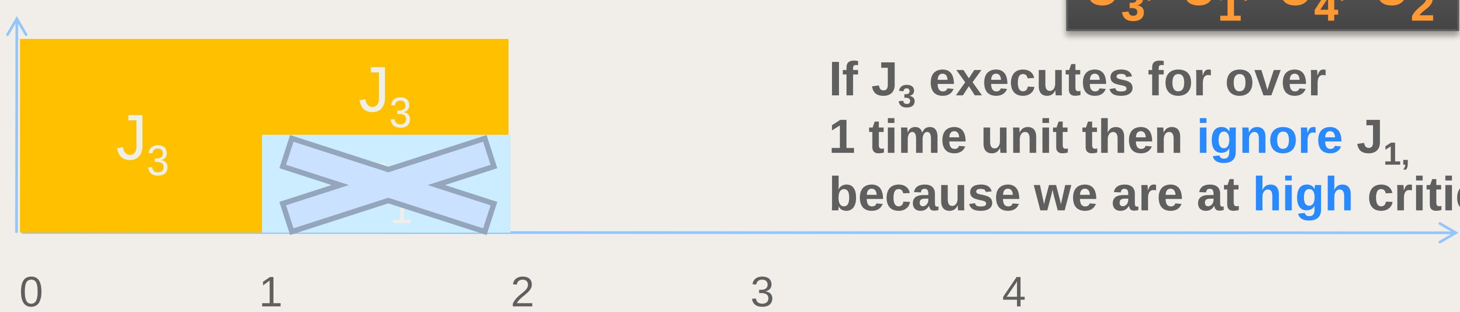

Let's start with $J_3$

# Mixed-Criticality Schedule

- A solution for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

$J_3 > J_1 > J_4 > J_2$



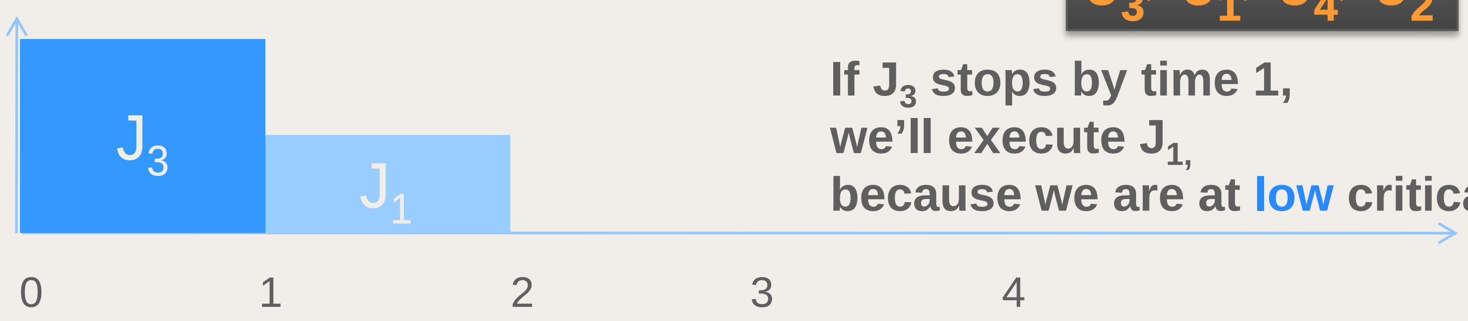

If $J_3$ executes for over 1 time unit then **ignore** $J_1$, because we are at **high** criticality.

# Mixed-Criticality Schedule

- A solution for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

$J_3 > J_1 > J_4 > J_2$




**If $J_3$ stops by time 1, we'll execute $J_1$, because we are at low criticality.**

# Mixed-Criticality Schedule

- A solution for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

$J_3 > J_1 > J_4 > J_2$

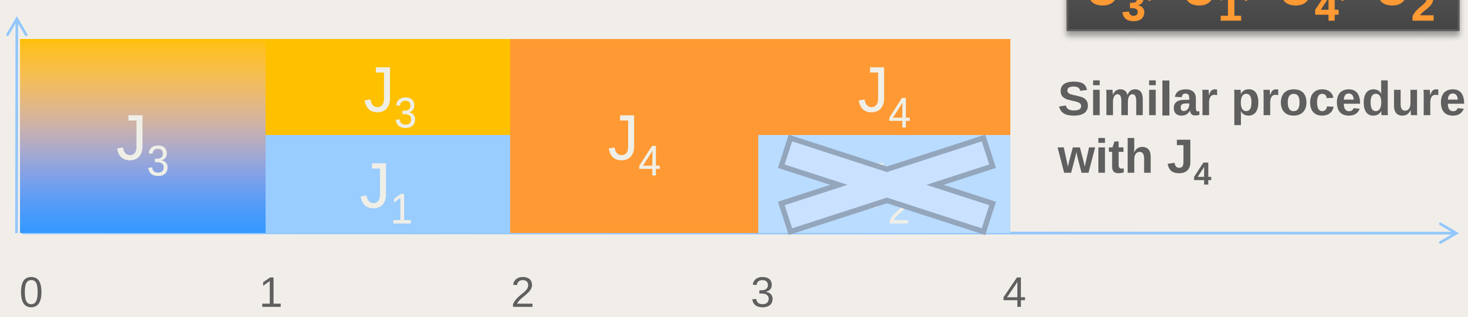

Similar procedure goes with $J_4$

# Mixed-Criticality Schedule

- A solution for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

$J_3 > J_1 > J_4 > J_2$




Similar procedure goes with $J_4$

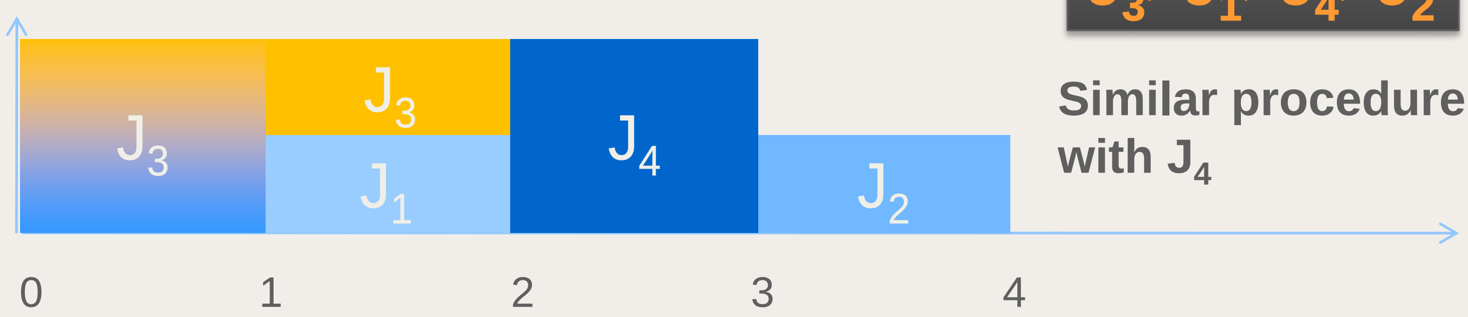
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

# Mixed-Criticality Schedule

- A solution for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

$J_3 > J_1 > J_4 > J_2$




Similar procedure goes with $J_4$

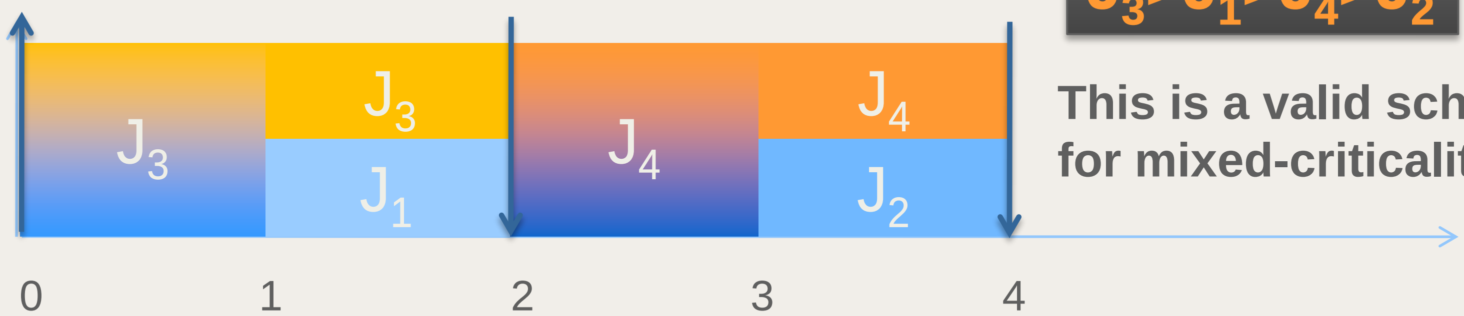
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

# Mixed-Criticality Schedule

- A solution for mixed-criticality jobs

| | Release time($A_i$) | Deadline($D_i$) | Execution time($C_i$) |
|---|---|---|---|
| $J_1$ | 0 | 2 | 1 |
| $J_2$ | 0 | 4 | 1 |
| $J_3$ | 0 | 4 | 2 |
| $J_4$ | 0 | 4 | 2 |

$J_3 > J_1 > J_4 > J_2$




This is a valid schedule for mixed-criticality jobs

# Model

- On the base of classic real-time job model, we add a parameter $x_i$, denoting the **criticality** of this job.

| | Release time($A_i$) | Deadline ($D_i$) | Criticality ($x_i$) | Execution time for low-criticality | Execution time for high-criticality |
|---|---|---|---|---|---|
| $J_1$ | 0 | 2 | Low | 1 | 1 |
| $J_2$ | 0 | 4 | Low | 1 | 1 |
| $J_3$ | 0 | 4 | High | 1 | 2 |
| $J_4$ | 0 | 4 | High | 1 | 2 |

- We assume that low-criticality job never uses more than specified execution time

# Model

- We define a job set as mixed-criticality schedulable (MC-schedulable) if there exists a schedule such that:
  - If every job uses at most specified execution time at low criticality, every job will meet its deadline;
  - If at least one high-criticality job uses more than specified execution time at low criticality, every high-criticality job will meet its deadline.

# Intractability Result

- Determining whether a given instance is MC-schedulable is NP-hard in the strong sense even if:
  - Every job's release time is exactly the same;
  - Jobs are preemptive.
- Therefore we focus on approximation algorithms

# Solution

- Processor Speed-up Factor
  - A scheduling algorithm has a processor speed-up factor $\Phi$ if
    - it can schedule any MC-schedulable instance
    - on a processor $\Phi$ times as fast
  - We use speed-up factor as a measurement
    - lower factor means better performance for a scheduling strategy
    - $\Phi=1$ means optimal

# Solution

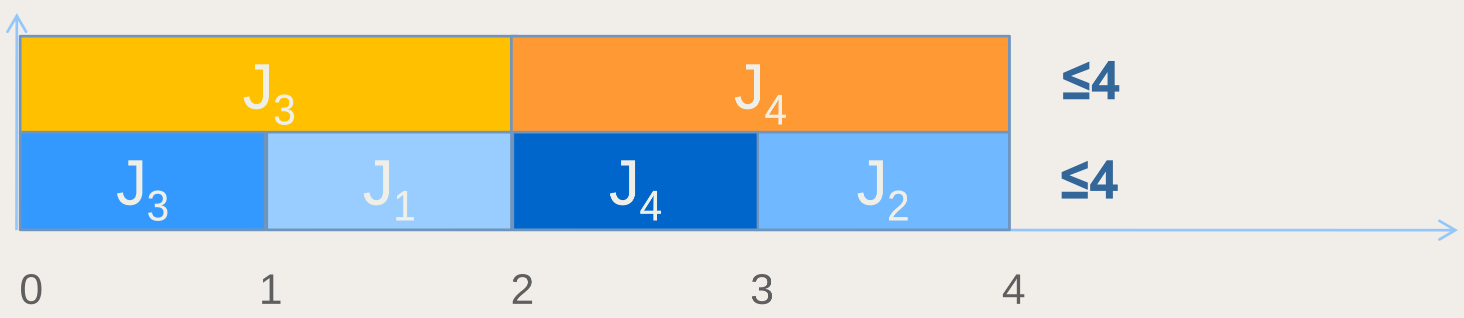
- We seek scheduling algorithms with low processor speed-up factor $\Phi$:
  - On a $\Phi$-speed processor, a job with execution time $t$ will only use $t/\Phi$ processor time
  - When proving a factor $\Phi$ is sufficient (to schedule any MC-schedulable instance),we are actually scheduling full-utilized instances
    - MC-schedulable instances are not over-utilized
      - **We know little about MC-schedulability**

# An Easy Solution

- A schedulability test with $\Phi=2$ is sufficient by worst-case reservation strategy in general.
  - The number 2 is the criticality number
  - Worst-case reservation means we respect the largest execution time estimations, and execute the jobs as a normal system

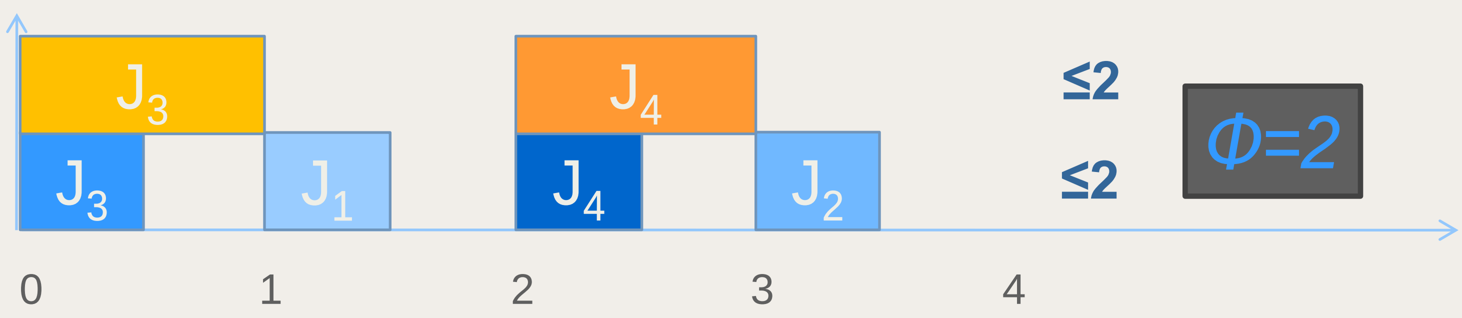
# An Easy Solution

- A schedulability test with *Φ*=2 is sufficient by worst-case reservation strategy in general.
  - Because the time demand in each criticality can not exceed the overall available processor time



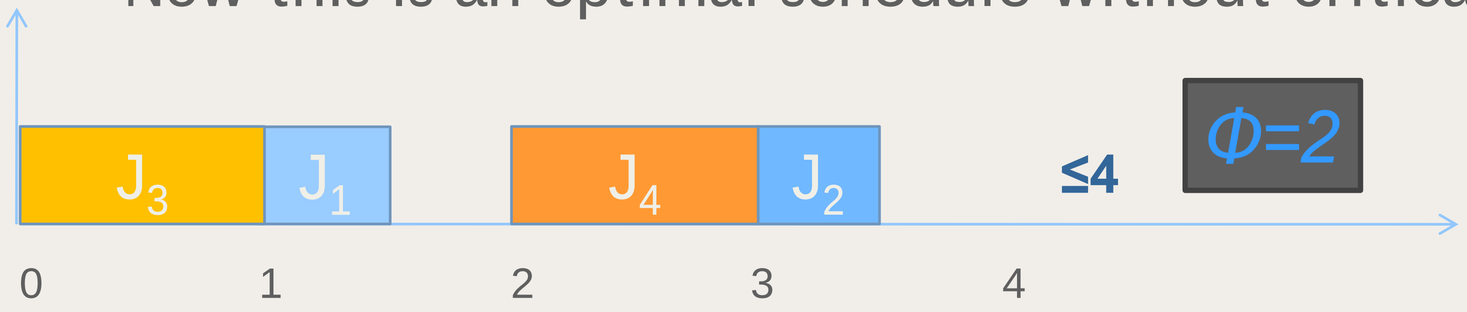
# An Easy Solution

- A schedulability test with $\Phi=2$ is sufficient by worst-case reservation strategy in general.
  - Because the time demand in each criticality can not exceed the overall available processor time



# An Easy Solution

- A schedulability test with $\Phi=2$ is sufficient by worst-case reservation strategy in general.
  - Because the time demand in each criticality can not exceed the overall available processor time
    - Now this is an optimal schedule without criticalities



# An Easy Solution

- A schedulability test with $\Phi$=2 is sufficient by worst-case reservation strategy in general.
  - Because the time demand in each criticality can not exceed the overall available processor time
    - EDF will work now

# Other Solutions

- Classical scheduling algorithms
  - Earliest deadline first(EDF): $\Phi=2$
    - It's exactly the same with worst-case reservation
  - Criticality monotonic: $\Phi=\infty$
    - It may need arbitrarily high speed-up factor to meet all deadlines

# Our Solution

- Own-Criticality-Based-Priority algorithm (OCBP algorithm):
  - A priority-based algorithm
  - Very similar to “Audsley’s Approach”
  - Repeatedly determine which remaining job can be assigned with lowest-priority
    - After an assignment, delete that job from current job set

# OCBP Algorithm

- *J* may be assigned with lowest priority if it meets its deadline as the lowest-priority job, when all other jobs executes for their worst-case execution time at *J*'s criticality.
  - If *J* is of high criticality, it will assume all other jobs use maximum possible execution time;
  - If *J* is of low criticality, it will assume all other jobs use low-criticality execution time.
    - **Otherwise we can just drop *J*.**

# OCBP Algorithm's Example

- Own-Criticality-Based-Priority algorithm:

| | Release time($A_i$) | Deadline ($D_i$) | Criticality ($x_i$) | Execution time for low-criticality | Execution time for high-criticality |
|---|---|---|---|---|---|
| $J_1$ | 0 | 2 | Low | 1 | 1 |
| $J_2$ | 0 | 4 | Low | 1 | 1 |
| $J_3$ | 0 | 4 | High | 1 | 2 |
| $J_4$ | 0 | 4 | High | 1 | 2 |

- For $J_4$, if all other jobs use high-criticality time, total time demand is 6.
  - $J_4$ can be the lowest-priority job only if $\Phi \geq 6/4 = 1.5$.

# OCBP Algorithm's Example

- Own-Criticality-Based-Priority algorithm:

| | Release time($A_i$) | Deadline ($D_i$) | Criticality ($x_i$) | Execution time for low-criticality | Execution time for high-criticality |
|---|---|---|---|---|---|
| $J_1$ | 0 | 2 | Low | 1 | 1 |
| $J_2$ | 0 | 4 | Low | 1 | 1 |
| $J_3$ | 0 | 4 | High | 1 | 2 |
| $J_4$ | 0 | 4 | High | 1 | 2 |

  - For $J_2$, if all other jobs use low-criticality time, total time demand is 4.
    - $J_2$ can be the lowest-priority job without speeding.

# OCBP Algorithm's Example

- Own-Criticality-Based-Priority algorithm:

| | Release time($A_i$) | Deadline ($D_i$) | Criticality ($x_i$) | Execution time for low-criticality | Execution time for high-criticality |
|---|---|---|---|---|---|
| $J_1$ | 0 | 2 | Low | 1 | 1 |
| $J_2$ | 0 | 4 | Low | 1 | 1 |
| $J_3$ | 0 | 4 | High | 1 | 2 |
| $J_4$ | 0 | 4 | High | 1 | 2 |

- After deleting $J_2$, for $J_4$, if all other jobs use high-criticality time, total time demand is 5.
  - Now $J_4$ can be the lowest-priority job only if $\Phi \geq 5/4 = 1.25$.

# OCBP Algorithm's Example

- Own-Criticality-Based-Priority algorithm:

| | Release time($A_i$) | Deadline ($D_i$) | Criticality ($x_i$) | Execution time for low-criticality | Execution time for high-criticality |
|---|---|---|---|---|---|
| $J_1$ | 0 | 2 | Low | 1 | 1 |
| $J_2$ | 0 | 4 | Low | 1 | 1 |
| $J_3$ | 0 | 4 | High | 1 | 2 |
| $J_4$ | 0 | 4 | High | 1 | 2 |

  - Now $J_1$ and $J_3$ can both be the lowest-priority job.

# OCBP Algorithm's Example

- Own-Criticality-Based-Priority algorithm

| | Release time($A_i$) | Deadline ($D_i$) | Criticality ($x_i$) | Execution time for low-criticality | Execution time for high-criticality |
|---|---|---|---|---|---|
| $J_1$ | 0 | 2 | Low | 1 | 1 |
| $J_2$ | 0 | 4 | Low | 1 | 1 |
| $J_3$ | 0 | 4 | High | 1 | 2 |
| $J_4$ | 0 | 4 | High | 1 | 2 |

- Final priority order:
  - $J_1>J_3>J_4>J_2$, or $J_3>J_1>J_4>J_2$.
- We need *$\Phi$=1.25* to make the priority lists valid

# Result

- Our final result is:
  - OCBP algorithm will need at most *Φ=1.618* speed-up factor to schedule any MC-schedulable instance with *2* criticalities;
    - Recalling that optimal schedule needs *Φ=1* and EDF needs *Φ=2*
  - OCBP algorithm runs in polynomial time.

# Future Work

- Extend the current result to periodic/sporadic real-time task model;
- Consider practical issues, like jitters, context-switches, and dependencies;
- Explore new algorithms to schedule mixed-criticality systems.

# Thank you

THE UNIVERSITY
*of* NORTH CAROLINA
*at* CHAPEL HILL