# Deadline-aware Offloading for High Throughput Accelerators

**Tsung Tai Yeh**, Matthew D. Sinclair,
Bradford M. Beckmann, Timothy G. Rogers

國立交通大學
National Chiao Tung University

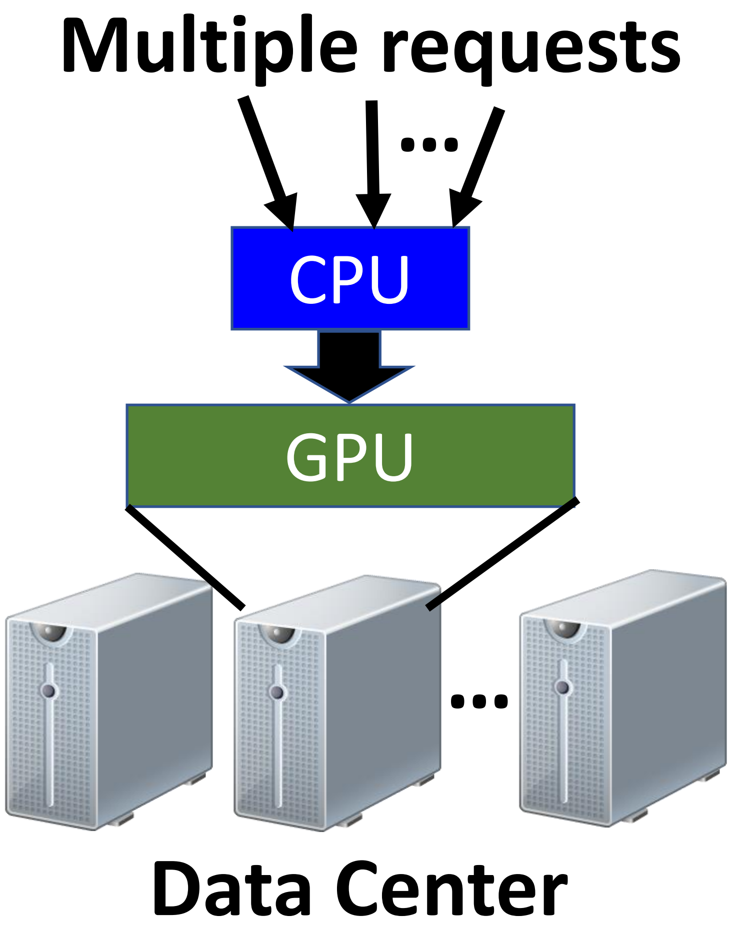
WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

AMD

PURDUE
UNIVERSITY

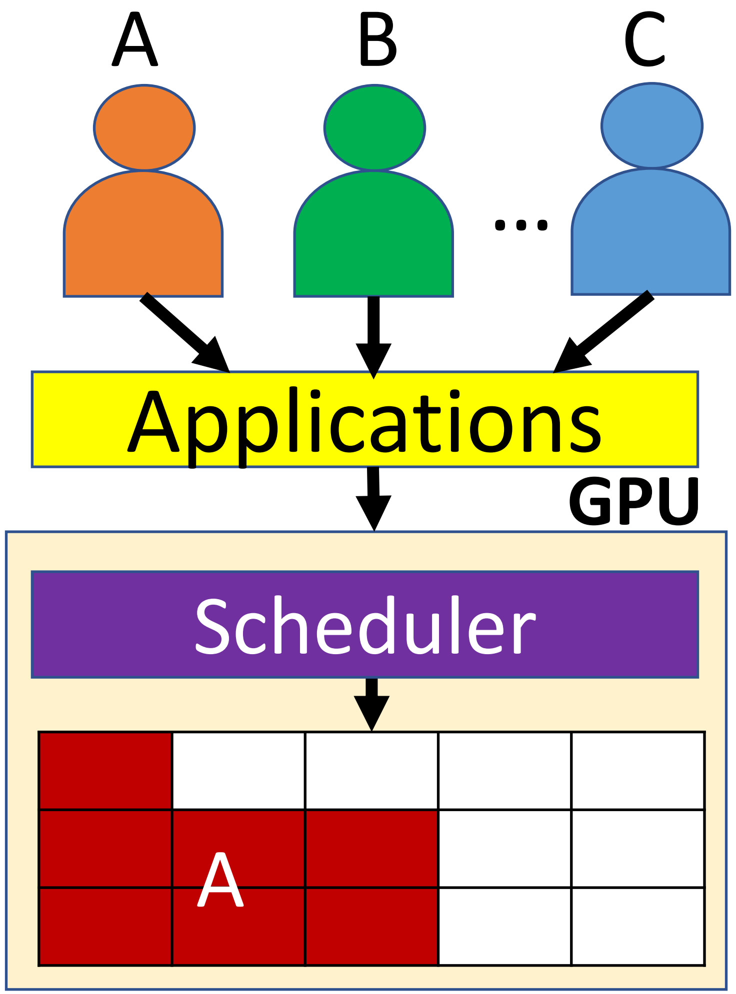
# Motivation

- **Emerging data center workloads**
  - Compute-intensive
  - Highly data parallel
  - Have tight deadlines
  - GPUs increasingly used at data centers
- **Applications**
  - Network processing
  - DNN inference and others
- **GPU streams**
  - Concurrent kernel execution
  - Improves occupancy but difficult to meet different deadlines



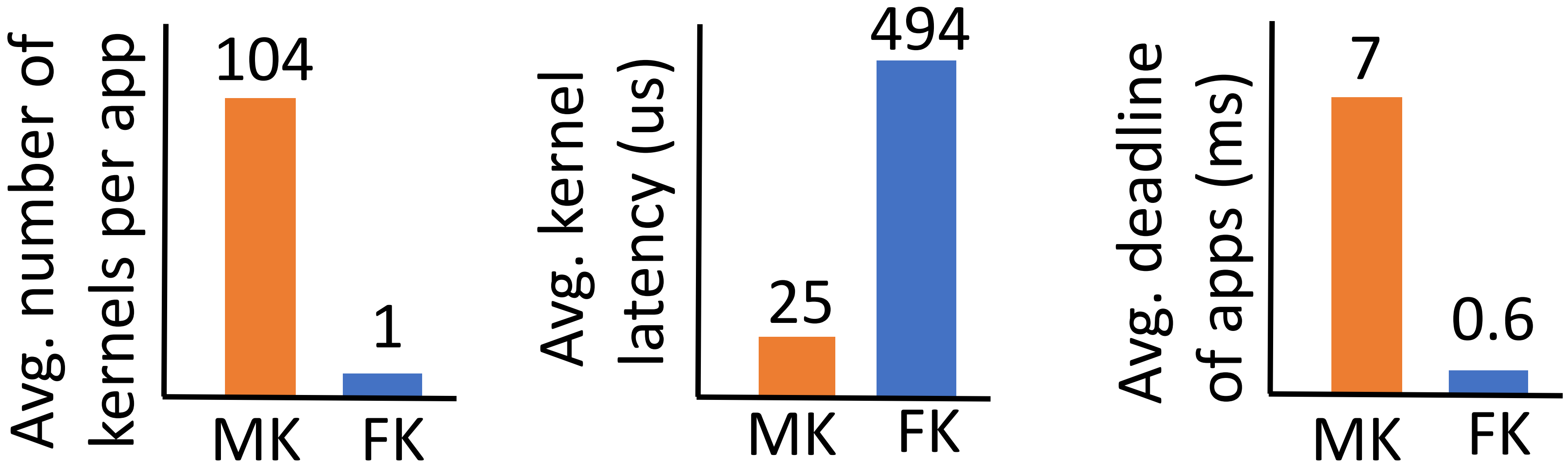
# Motivation

- **Medium parallelism**
  - A single job cannot fully utilize entire GPU
- **GPU inefficient for latency-driven workloads**
  - High host scheduling overhead
  - Static priority assigned by programmers
- **Requirement**
  - Need to carefully co-schedule requests



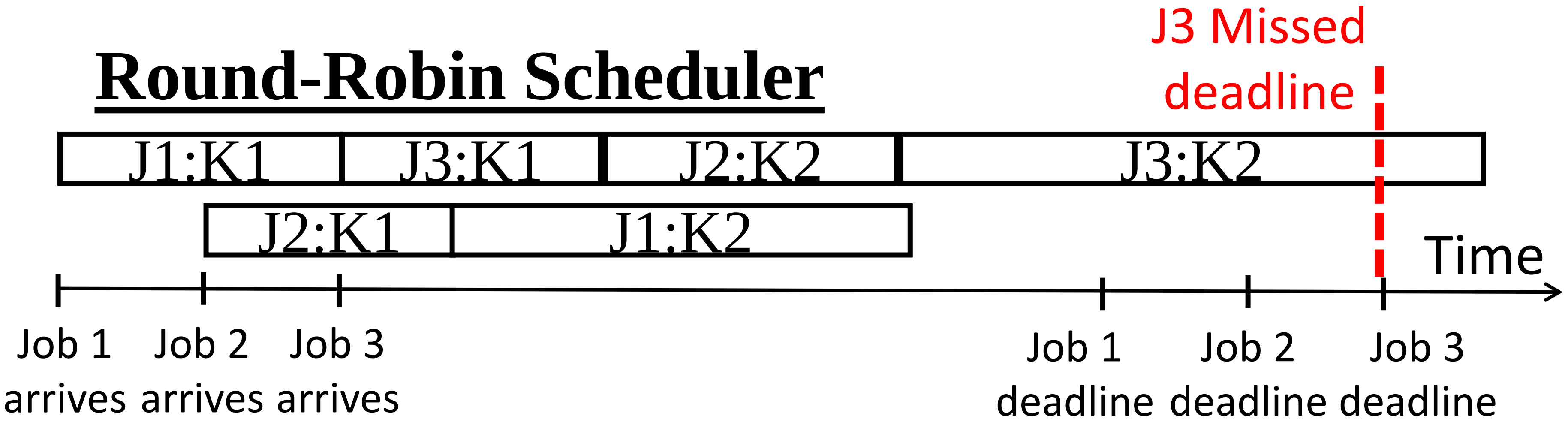
# Additional Characteristics of GPU Applications

- **Many-kernel (MK) applications** (e.g., RNN inference)
  - Relatively small, short kernels that have stringent deadlines
- **Few-kernel (FK) applications** (e.g., Personal Assistants, Network)
  - Bigger kernels with longer deadlines



# Key Challenge 1

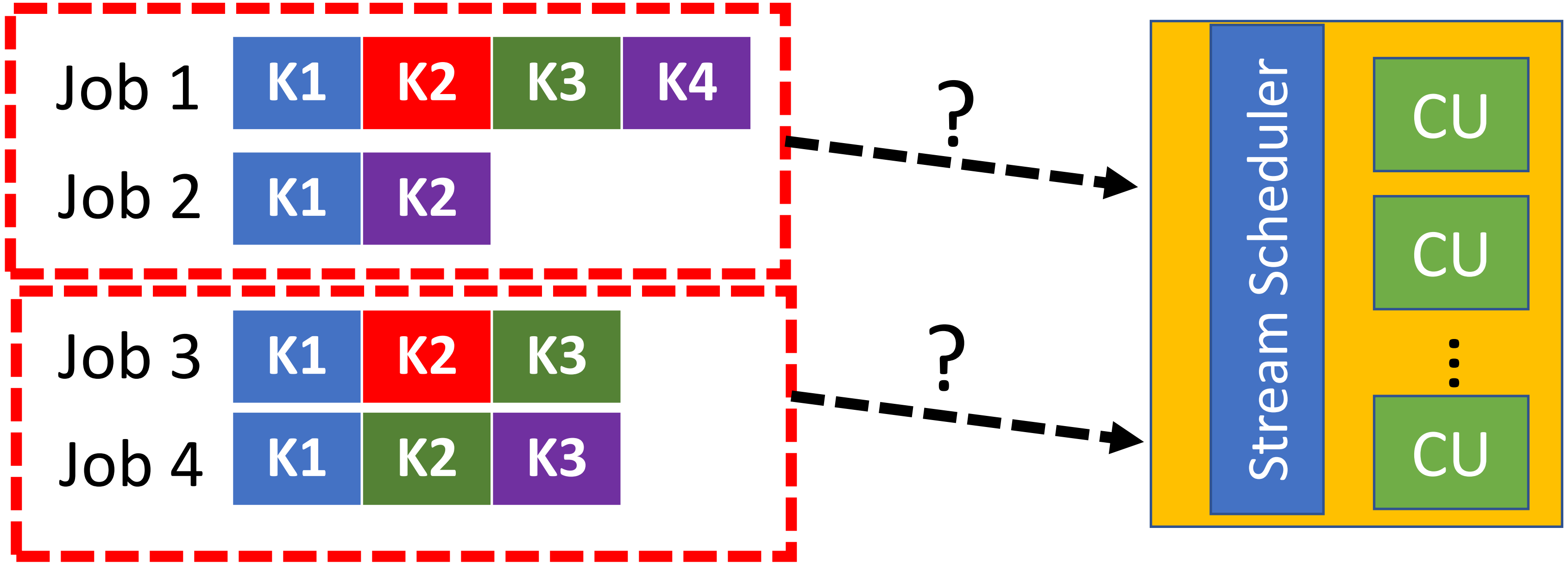
- How to decide job priorities?
  - QoS constraints for laxity-sensitive applications
  - Multiple jobs contend for GPU resources
  - Static priorities can be overly conservative




3 jobs, each with 2 kernels
Assume GPU can execute 2 kernels simultaneously

# Key Challenge 2

- How to avoid oversubscribing the GPU?
  - Slow system response makes it difficult to meet real-time deadlines
  - **Challenge 2A:** How many jobs should be picked?
  - **Challenge 2B:** Which job should be chosen?



# Our Goal

**Minimize** the number of jobs that miss their deadlines while **maximizing** the GPU utilization

# LAX: Deadline-aware Offloading

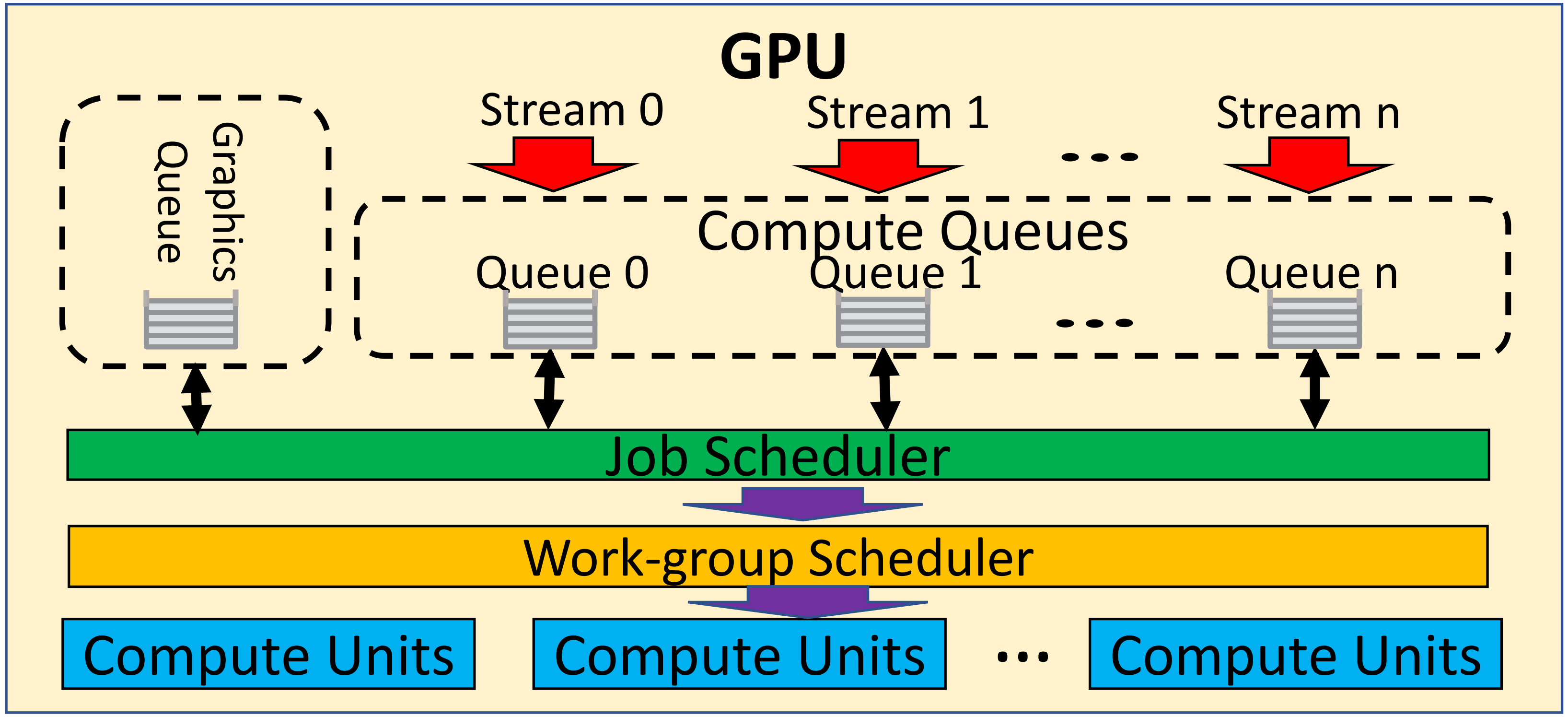
- **Component 1 – Job Scheduling**
  - **Exploit hardware information:**
    - Determine how much contention is occurring
    - Decide how much slack each job has before its deadline
  - **Dynamically reprioritize jobs**
- **Component 2 – Queuing Delay Calculation**
  - Using Little's Law to estimate the capacity of the GPU
  - Predicting the time remaining of each job

# Outline

- Motivation
- Background
- Laxity-aware Scheduling (LAX)
- Queuing Delay Estimation
- Evaluation
- Conclusion

# GPU Stream Scheduler and Execution

- Concurrent execution by GPU streams
- Each application (job) is launched by GPU streams
- The stream scheduler determines the priority of each job



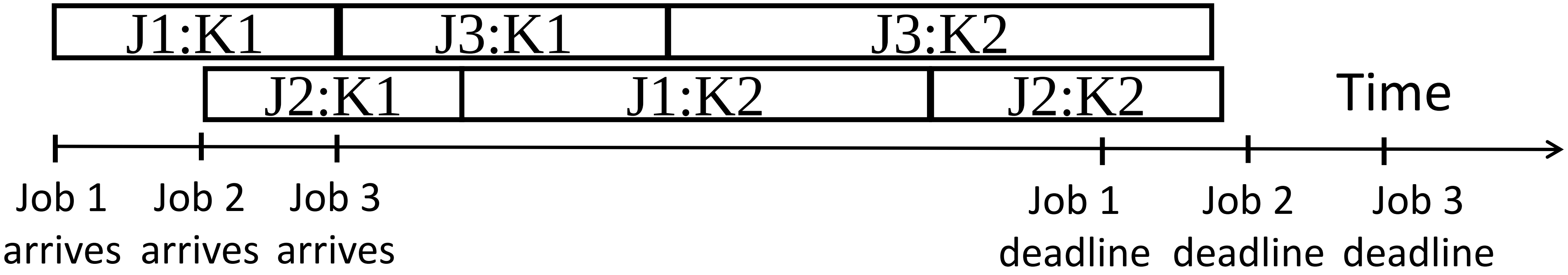
# Outline

- Motivation
- Background
- Laxity-aware Scheduling (LAX)
- Queuing Delay Estimation
- Evaluation
- Conclusion

# Laxity-aware (LAX) Scheduler

- The laxity of a job determines its priority
  - Laxity = Deadline – (TimeRemaining + DurationTime)
  - Laxity tells us the slack in a job's deadline
  - **Challenge 1**: How to predict "TimeRemaining" of a job?
  - **Challenge 2:** How often should update "Laxity"?

**Laxity-aware Scheduler**




3 jobs, each with 2 kernels,
Assume GPU can execute 2 kernels simultaneously

# Laxity-aware Scheduling

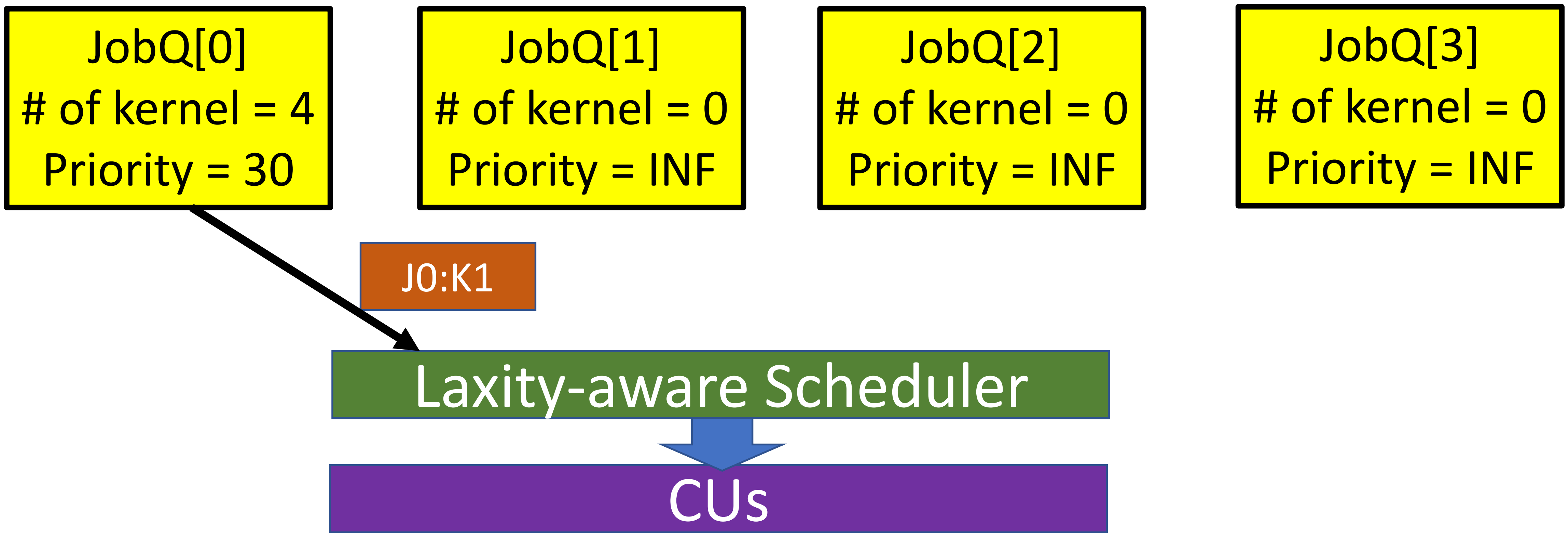
**Time = 0**

| JobQ[0] | JobQ[1] | JobQ[2] | JobQ[3] |
|---|---|---|---|
| # of kernel = 0 | # of kernel = 0 | # of kernel = 0 | # of kernel = 0 |
| Priority = INF | Priority = INF | Priority = INF | Priority = INF |

No jobs are pushed into job queue

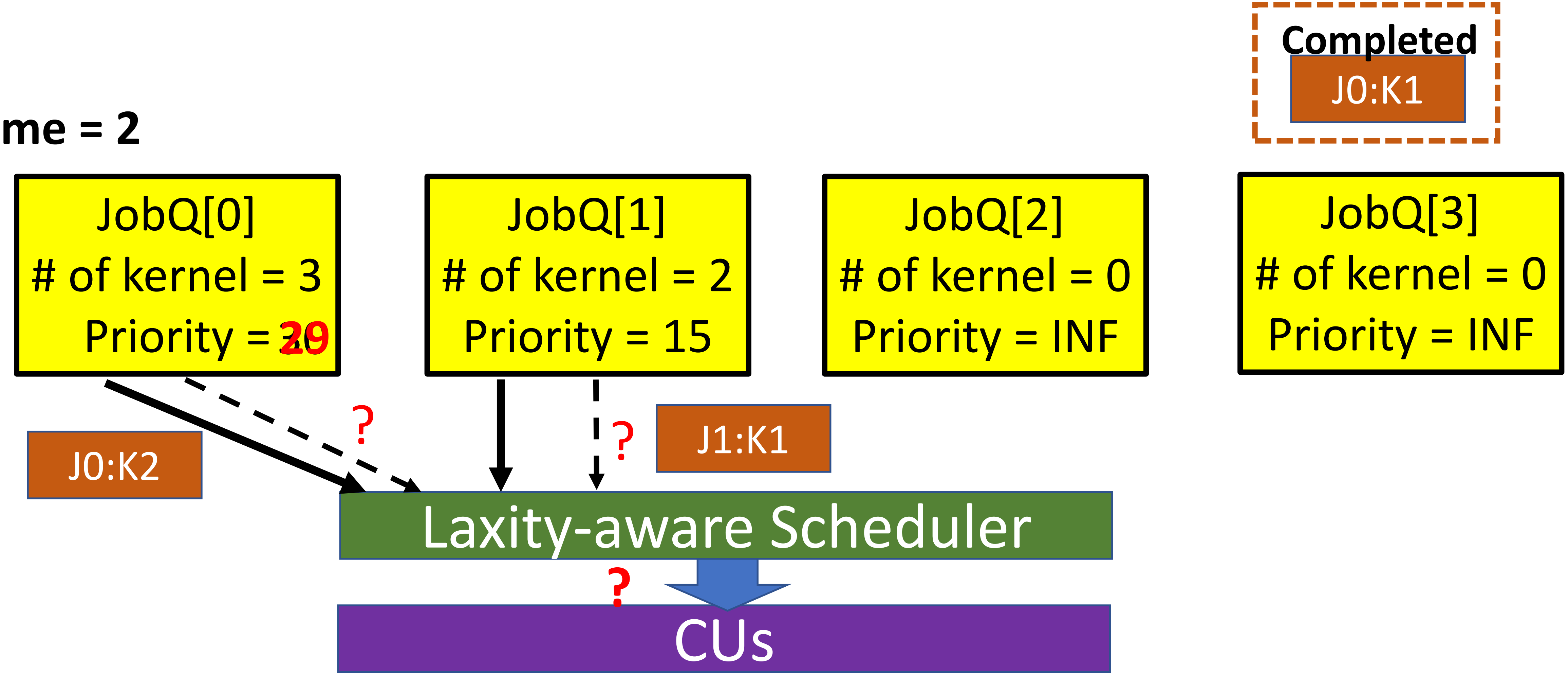
# Laxity-aware Scheduling

**Time = 1**



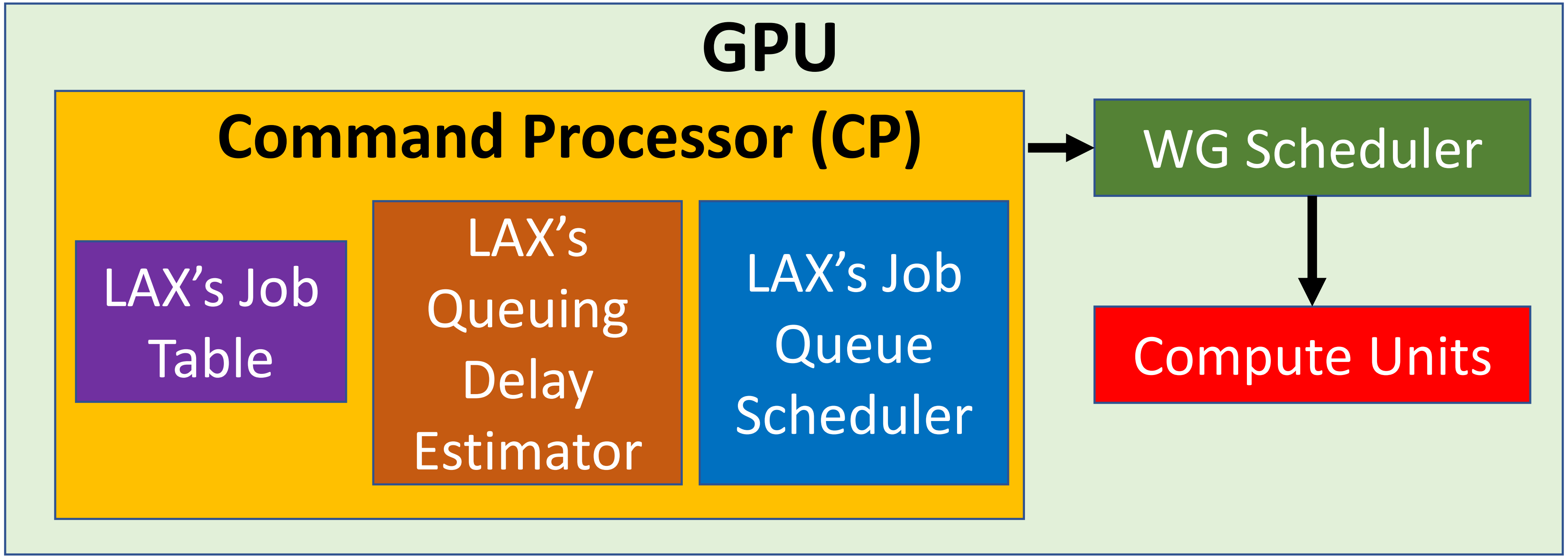
# Laxity-aware Scheduling

**Time = 2**



# LAX Architecture

- Adds an additional hardware table in CP's scratchpad
- Extends the job queue scheduler



# LAX Job Table

- **WG List**
  - Keep total number of workgroup (WG) in each type of kernel used by a job
- **Kernel Profiling Table**
  - Record WG completion rate (# of completed WG/ time)
- **Estimate job end-to-end latency**
  - ∑ Total_WG_Ki / WG_completion_rate_Ki

LAX's Job Table

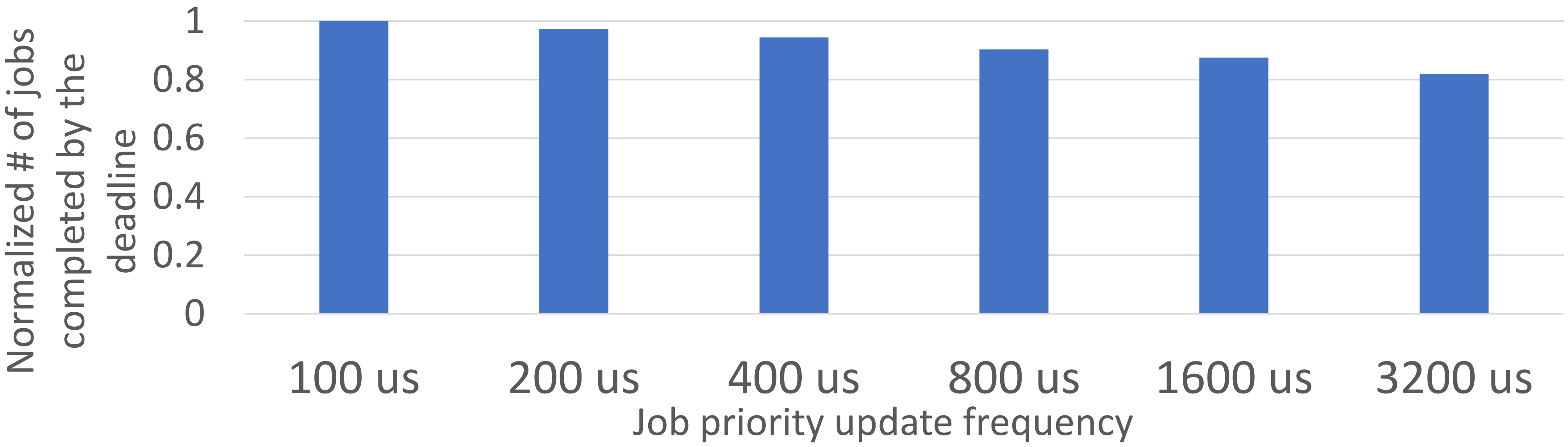
| QID | Priority | WG List | Deadline | Start Time | state |
| --- | --- | --- | --- | --- | --- |
| | | | | | |

| Kernel ID | K1 | K2 | … | K8 |
| --- | --- | --- | --- | --- |
| Total WG | | | … | |

Kernel Profiling Table

| Kernel ID | K1 | K2 | … | K8 |
| --- | --- | --- | --- | --- |
| WG Completion Rate | | | … | |
| Completed WG Count | | | … | |

# How frequently to update priorities?

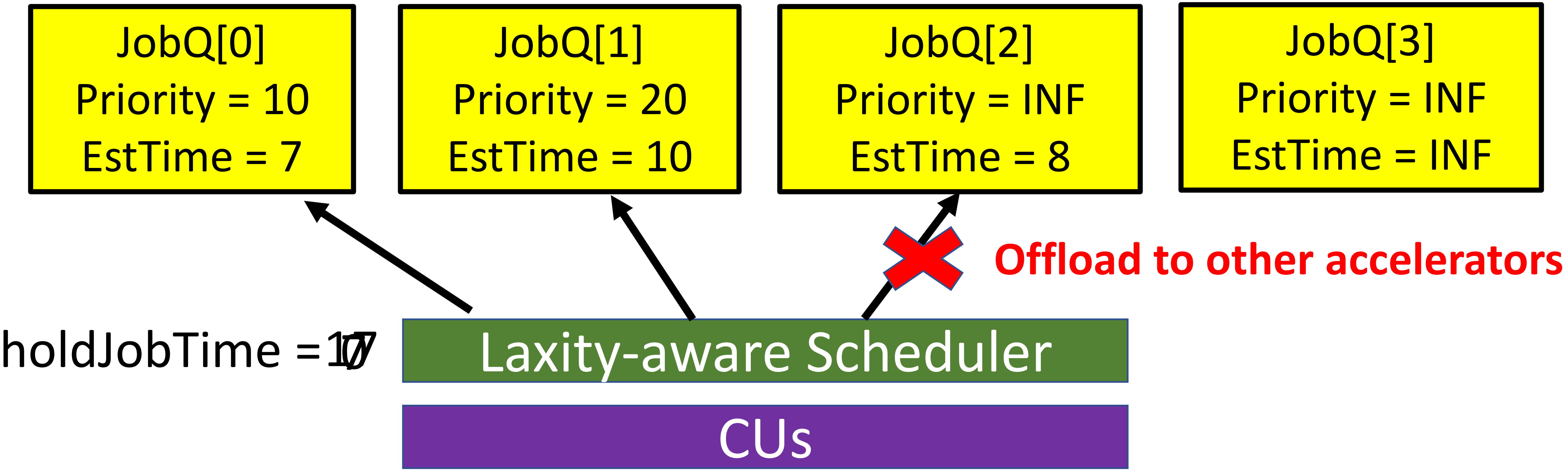



- Frequent priority updates improve performance
- Enables scheduler to quickly adjust priorities as contention changes
- Empirically choose 100 us (priority update frequency)

# Outline

- Motivation
- Background
- Laxity-aware Scheduling (LAX)
- Queuing Delay Estimation
- Evaluation
- Conclusion

# Queuing Delay Estimation




Job 2 is a new job and JobQ[2].deadline = 15

holdJobTime + JobQ[2].EstTime > JobQ[2].deadline

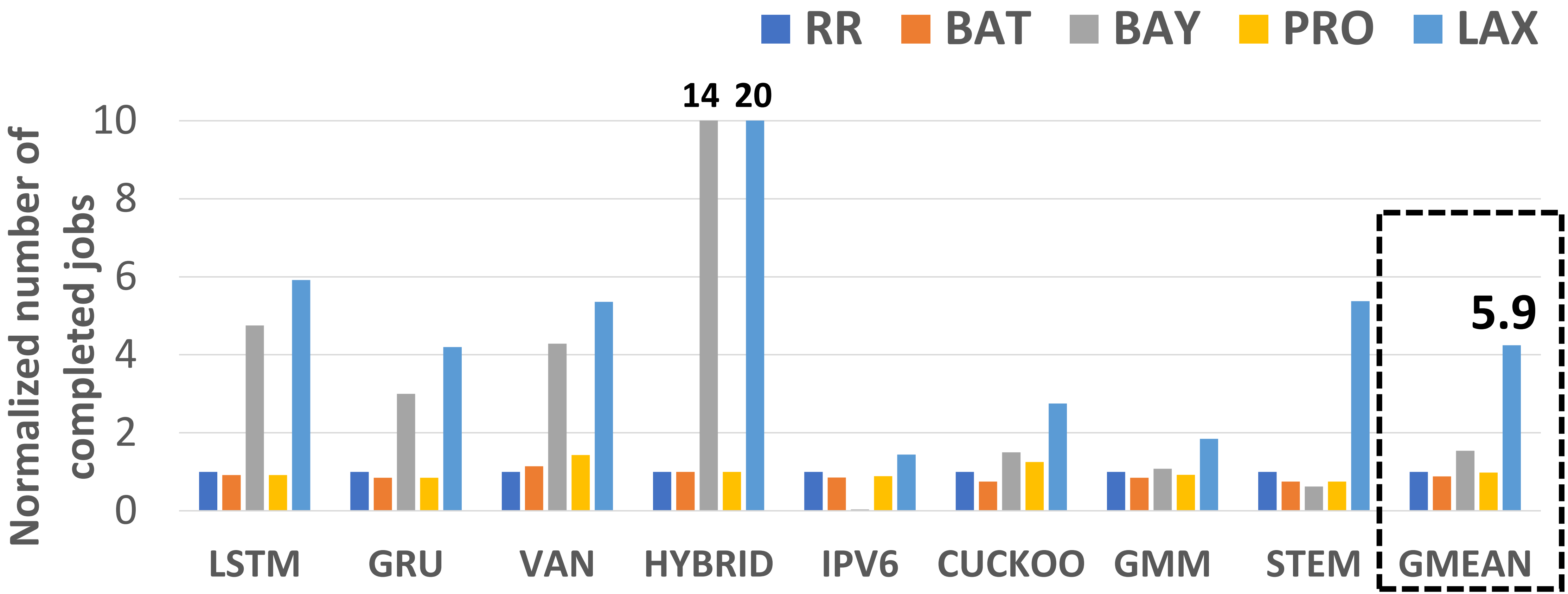
# Outline

- Motivation
- Background
- Laxity-aware Scheduling (LAX)
- Queuing Delay Estimation
- Evaluation
- Conclusion

# Evaluation Methodology

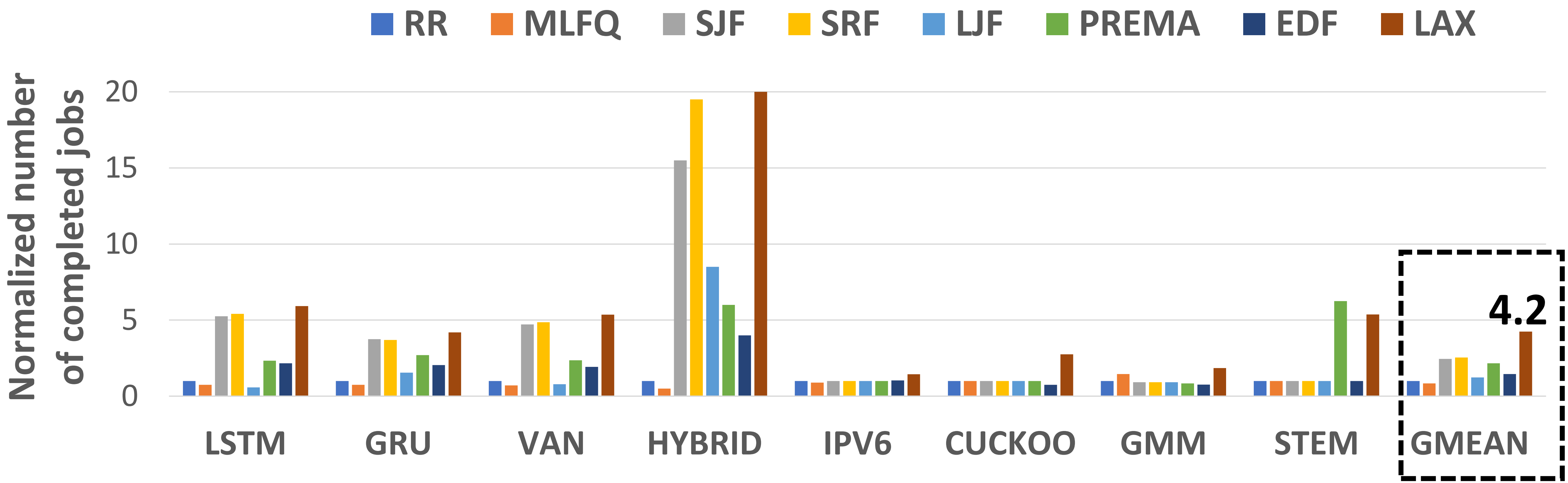
- **Simulator:** gem5-APU
  - 8 CUs, 4 SIMD units per CU
  - 128 compute queues
  - Up to 10 wavefronts per CU
  - Compare LAX to 10 different job scheduling alternatives
- **Workloads:**
  - DeepBench RNNs (Vanilla, GRU, LSTM, Hybrid)
  - G-Opt (Networking: CUCKOO, IPV6)
  - Lucida (IPA: GMM, Stemmer)
  - Each application has different real-time deadlines
  - High, medium, and low arrival rates (exponential distribution)

# CPU-side Scheduling Performance



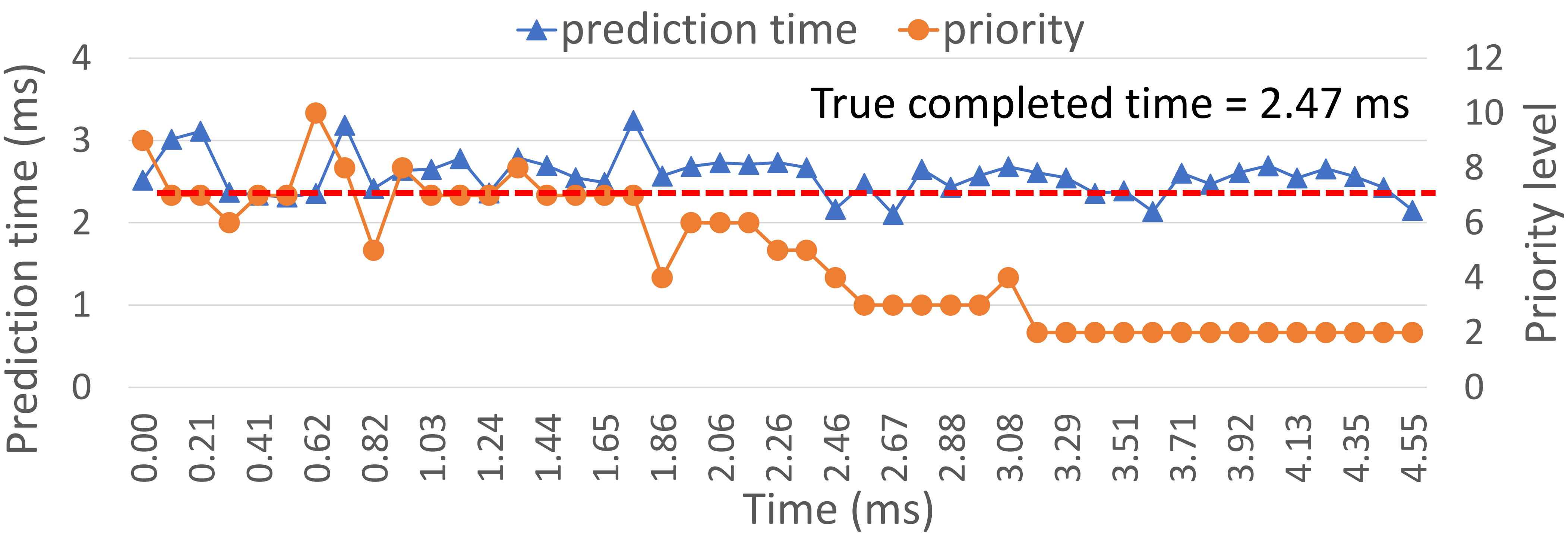

LAX up to **5.9X** geomean better than CPU-side schedulersat the high job arrival rate

# CP-extension Scheduling Performance




LAX up to **4.2X** geomean better than other schedulers that extend CP at the high job arrival rate

# LAX Predictions for a Sample LSTM Job




LAX's predictions have a mean absolute error of 8%

# Additional Studies in the Paper

- **Other Design Considerations**
  - Additional LAX variants examine required level of HW support
- **Sensitivity Studies**
  - Successful job throughput
  - 99-percentile job latency
  - Energy consumption
- **Area estimation:**
  - 4240 bytes of memory for 128 compute-queues

# Conclusion

- **Emerging GPU applications have different characteristics**
  - Real-time constraints, medium amount of parallelism
- **Opportunity**
  - Using stream scheduler to execute jobs simultaneously
- **Problems:**
  - How to decide the priority of jobs?
  - How many jobs should be offloaded?
- **More intelligent scheduler:** Laxity-aware scheduling
  - Predict job completion time and queuing delay
  - Dynamically change job priorities based on their laxity
- **Results:** Complete 1.7X – 5.9X more jobs by their deadlines

## Copyright Disclosure



## Disclaimer

AMD