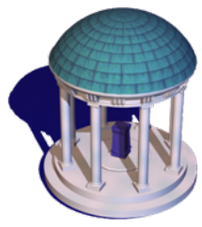
Robust Real-Time Multiprocessor Interrupt Handling Motivated by GPUs

Glenn Elliott
gelliott@cs.unc.edu

Jim Anderson
anderson@cs.unc.edu



The University *of* North Carolina *at* Chapel Hill

Work supported by NSF grants CNS 1016954 and CNS 1115284; ARO grant W911NF-09-1-0535; AFOSR grant FA9550-09-1-0549; and AFRL grant FA8750-11-1-0033.

Real-Time GPUs

Real-Time GPUs

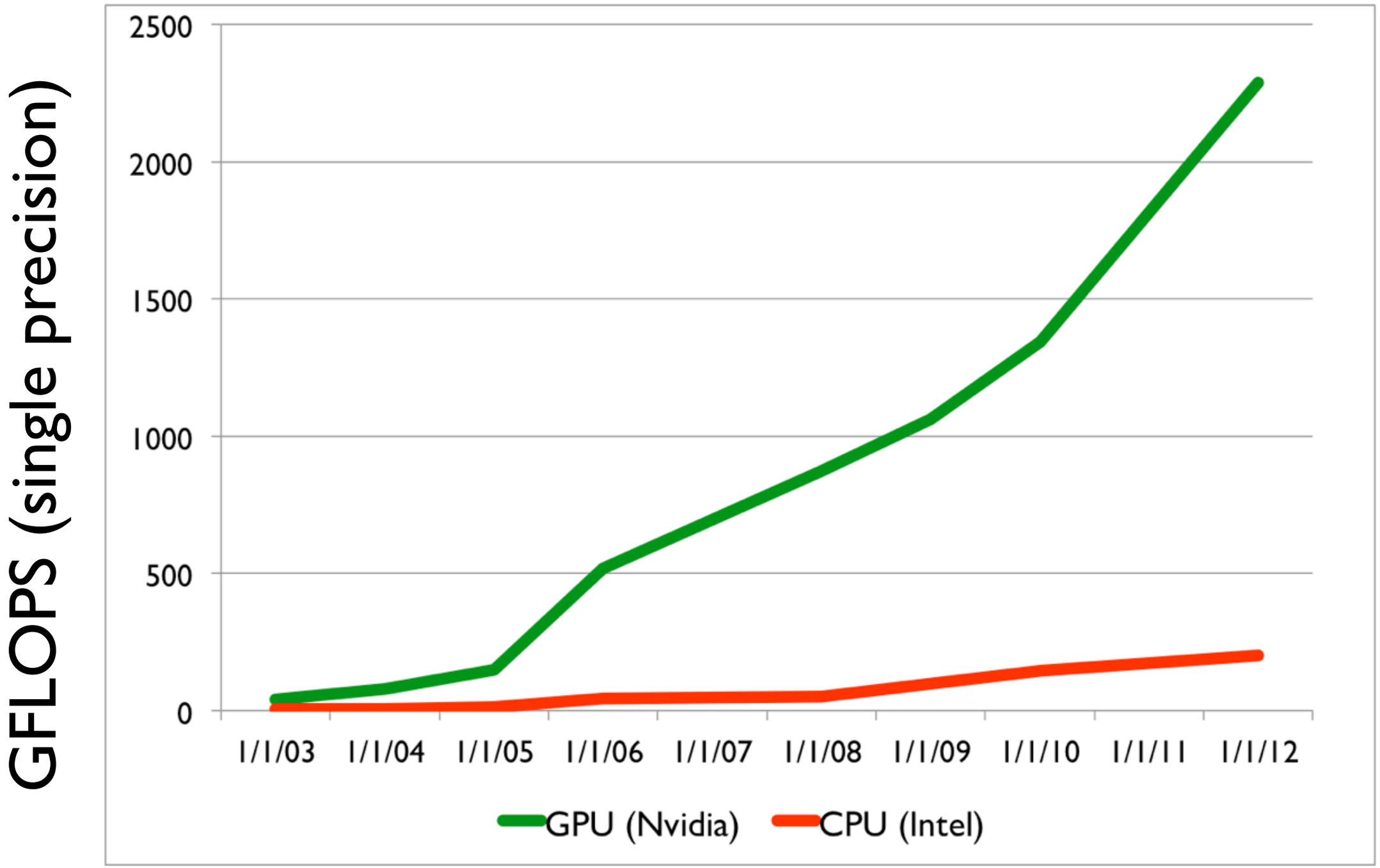
- Graphics processing units (GPUs) can now be programmed with high-level languages to solve general purpose problems
- Practice called "GPGPU"

Real-Time GPUs

- Graphics processing units (GPUs) can now be programmed with high-level languages to solve general purpose problems
  - Practice called "GPGPU"

- **Why use GPUs in real-time systems?**

Primary Motivation: Performance



Domains for GPUs

- GPUs excel at data parallel problems
  - Digital signal processing
  - Matrix-like computations
  - Sorting and searching

Future Automotive Applications

- Vehicle and pedestrian detection
- Object tracking
- Fusion of video, laser, and radar sensor data
- **Clear real-time implications!**

Target Platform

Target Platform

- We want to develop a system using components available today

Target Platform

- We want to develop a system using components available today
- Current state of technology motivates the following platform:

Target Platform

- We want to develop a system using components available today
- Current state of technology motivates the following platform:
  - Multicore system with one or more GPUs

Target Platform

- We want to develop a system using components available today
- Current state of technology motivates the following platform:
  - Multicore system with one or more GPUs
  - Soft real-time

Target Platform

- We want to develop a system using components available today
- Current state of technology motivates the following platform:
  - Multicore system with one or more GPUs
  - Soft real-time
  - Linux-based operating system

Challenges: I/O Device

Challenges: I/O Device

1. Managed by an operating system driver
   - Usually closed source
   - Not originally designed for real-time use

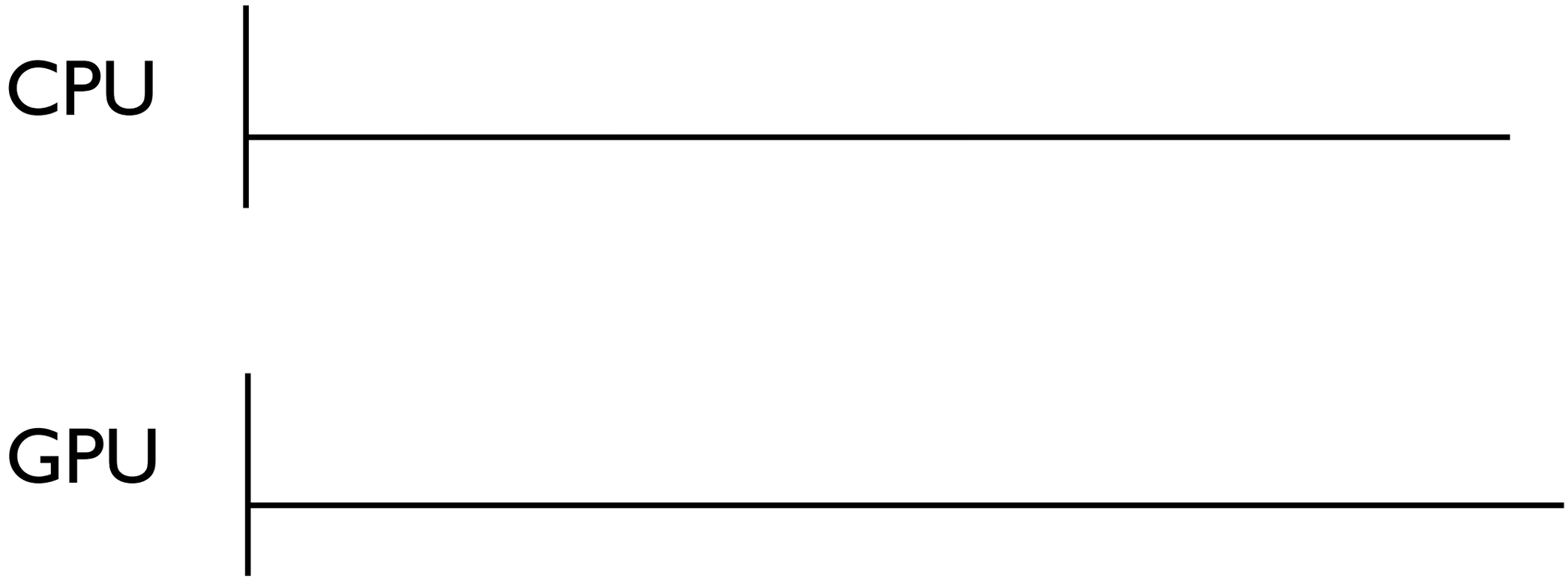
Challenges: I/O Device

1. Managed by an operating system driver
   - Usually closed source
   - Not originally designed for real-time use
2. Not directly schedulable like a CPU
   - Allocation/arbitration issues

Challenges: I/O Device

1. Managed by an operating system driver
   - Usually closed source
   - Not originally designed for real-time use
2. Not directly schedulable like a CPU
   - Allocation/arbitration issues
3. Interrupt-driven communication

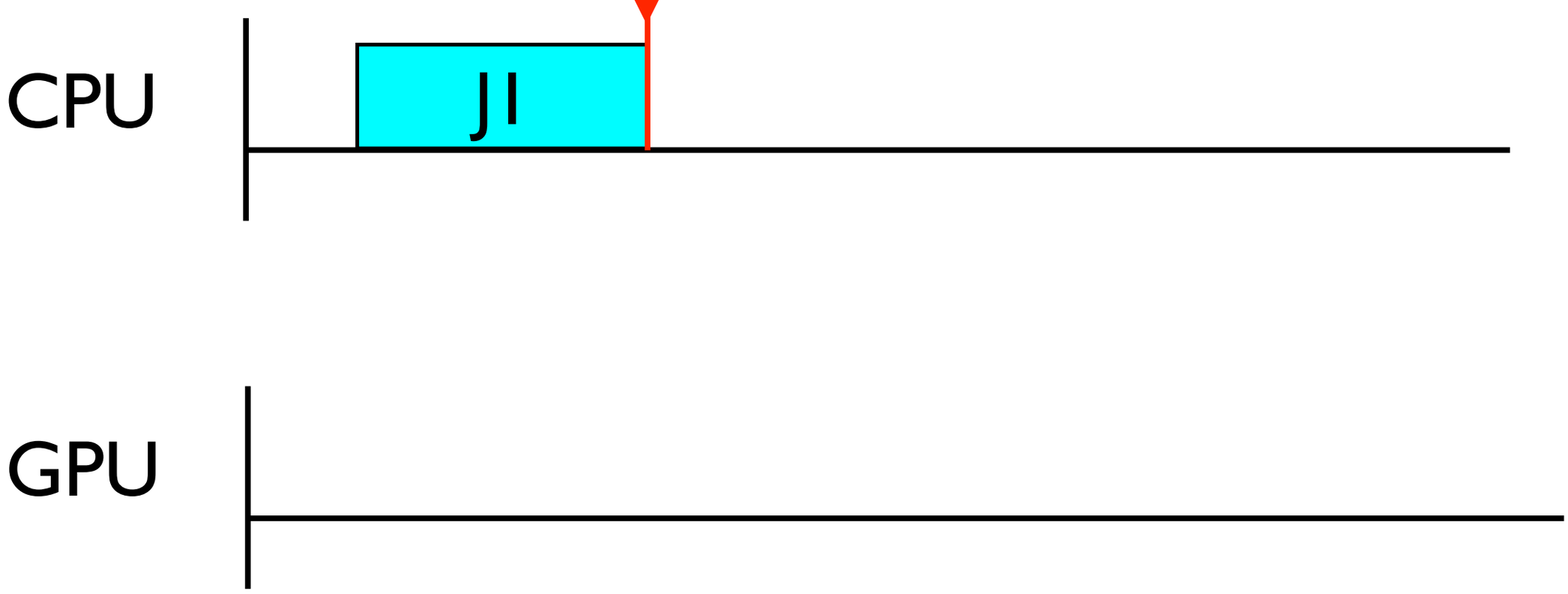
Synchronous GPU Usage Pattern



Synchronous GPU Usage Pattern

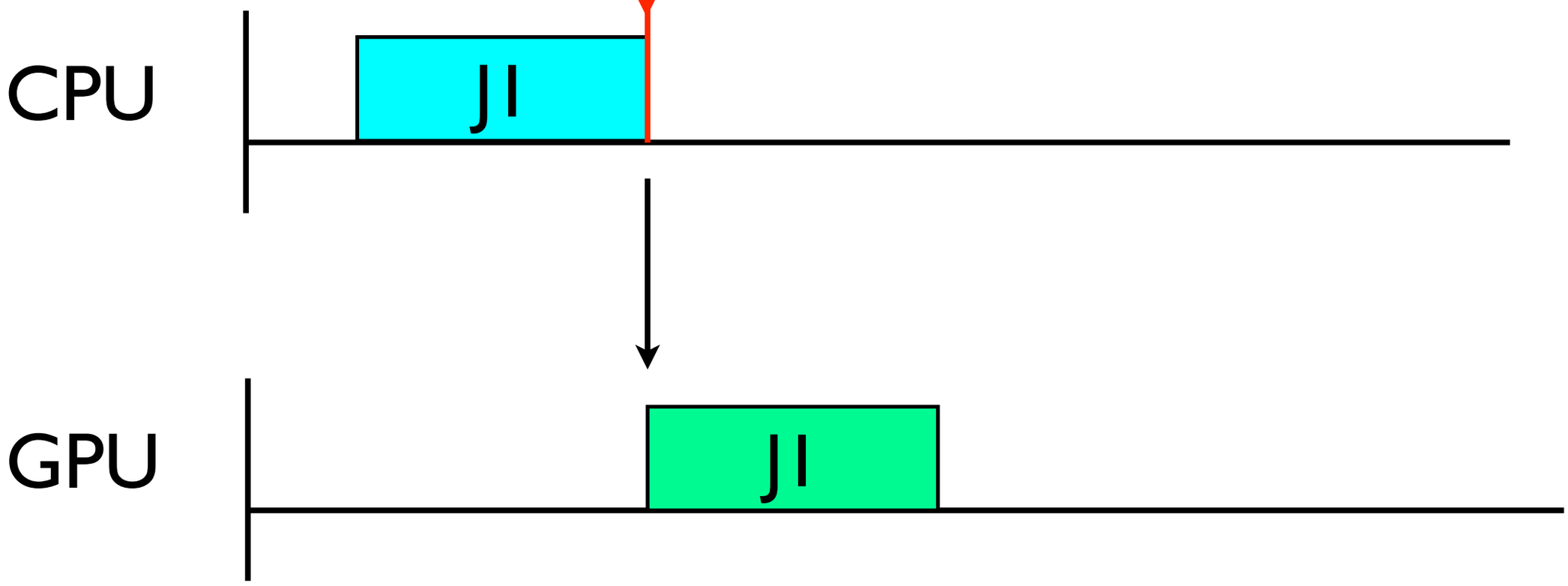
Synchronous GPU Usage Pattern

J1 sends work to the GPU and blocks waiting for results.



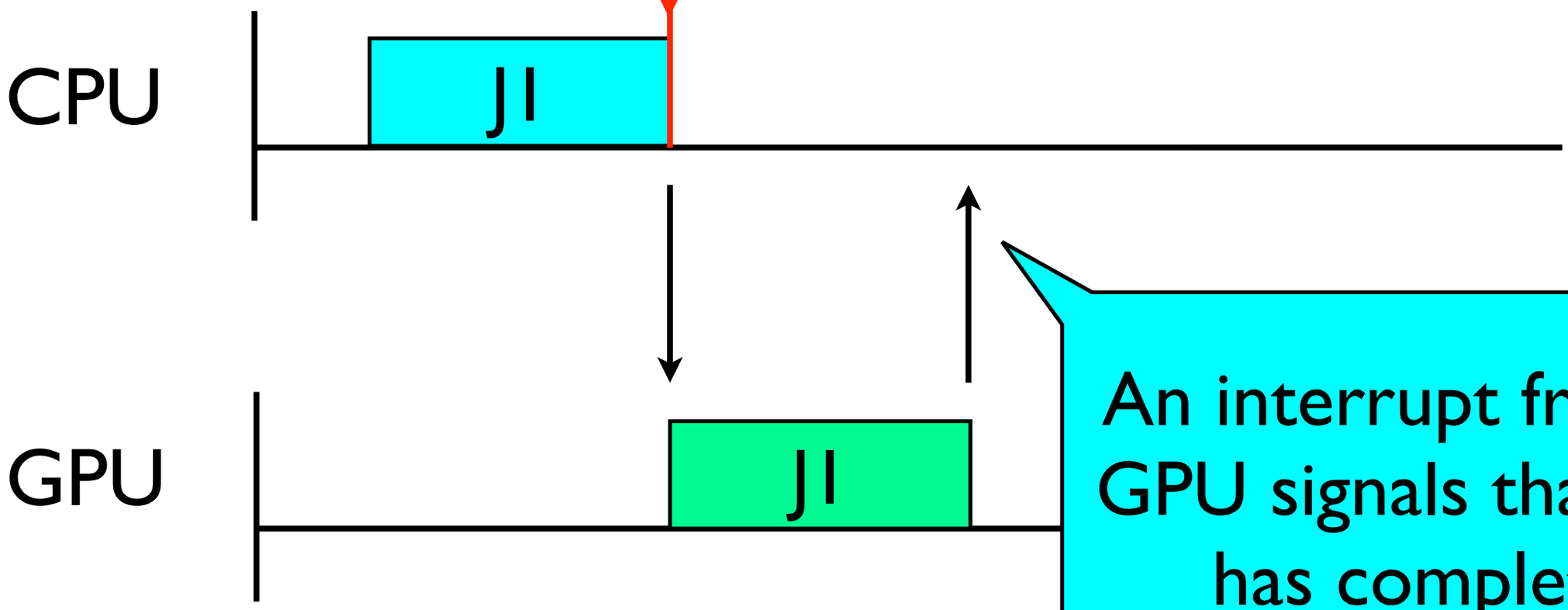
Synchronous GPU Usage Pattern

J1 sends work to the GPU and blocks waiting for results.



Synchronous GPU Usage Pattern

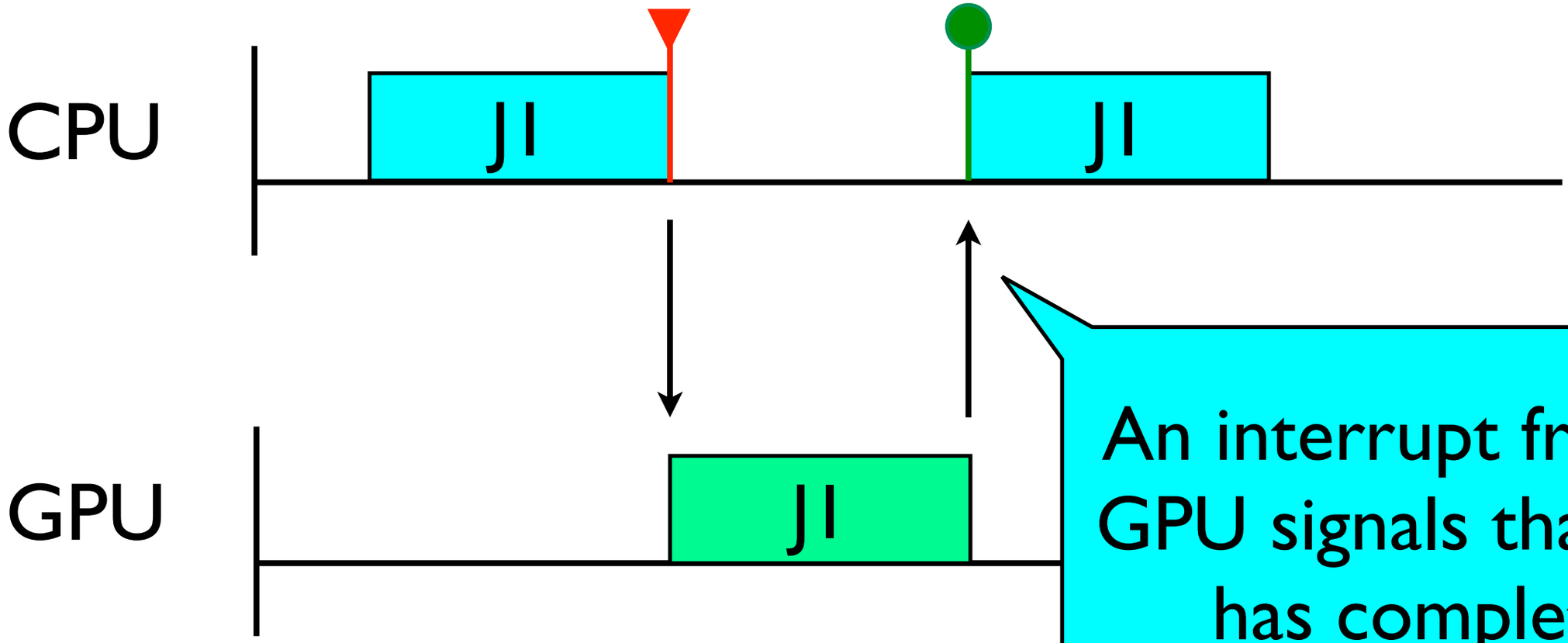
J1 sends work to the GPU and blocks waiting for results.



An interrupt from the GPU signals that work has completed. (Handler not depicted.)

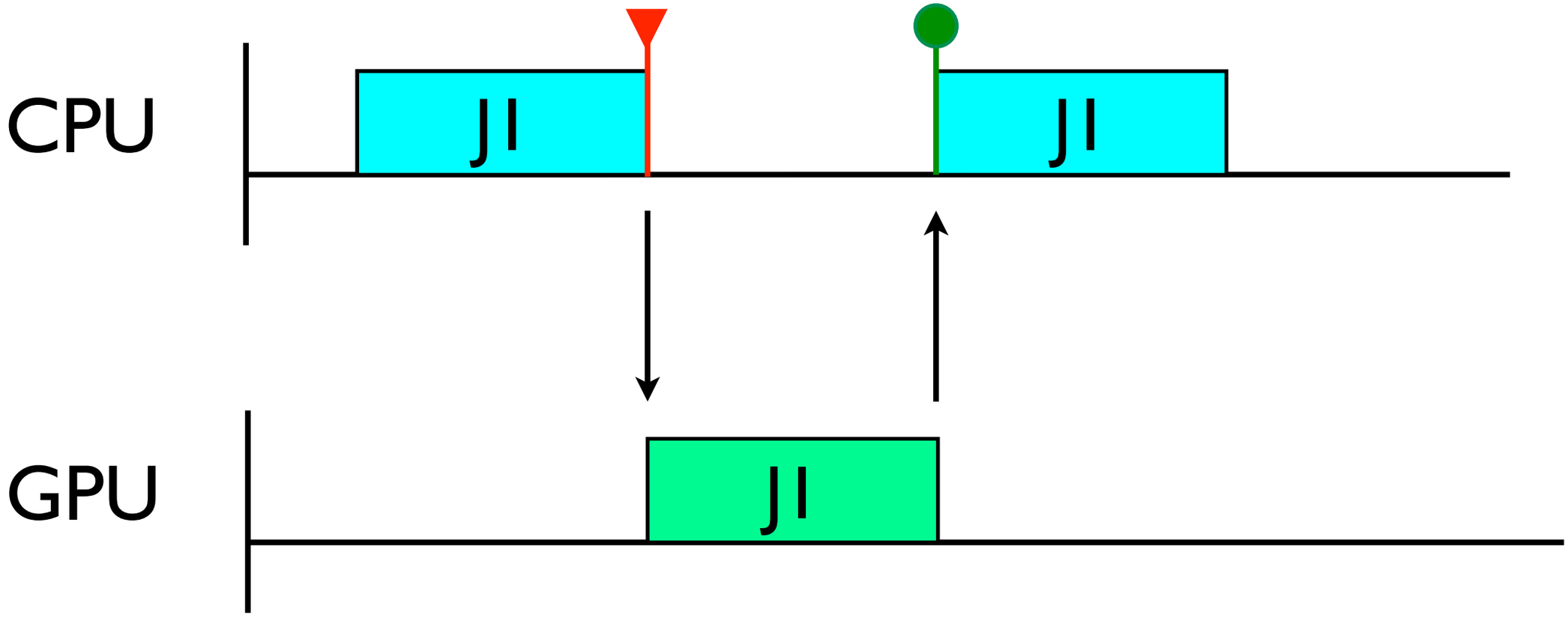
Synchronous GPU Usage Pattern

J1 sends work to the GPU and blocks waiting for results.



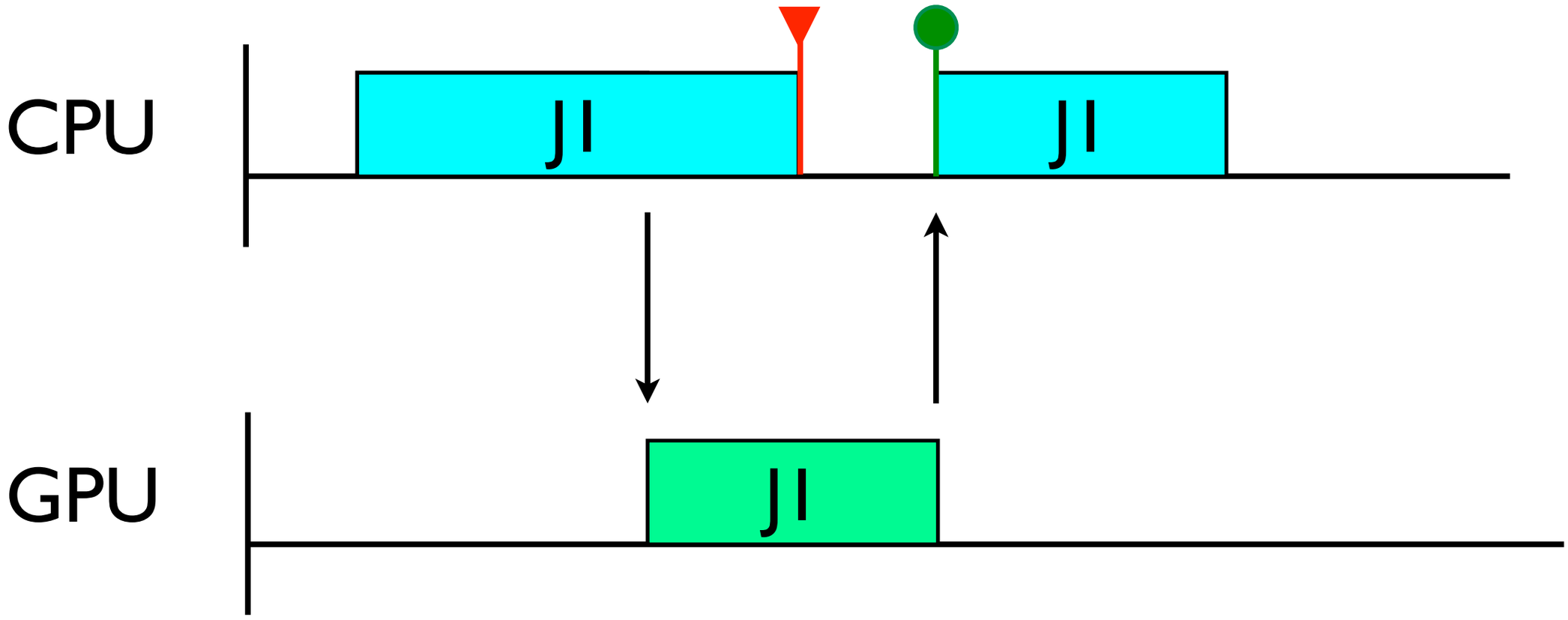
An interrupt from the GPU signals that work has completed. (Handler not depicted.)

Asynchronous GPU Usage Pattern

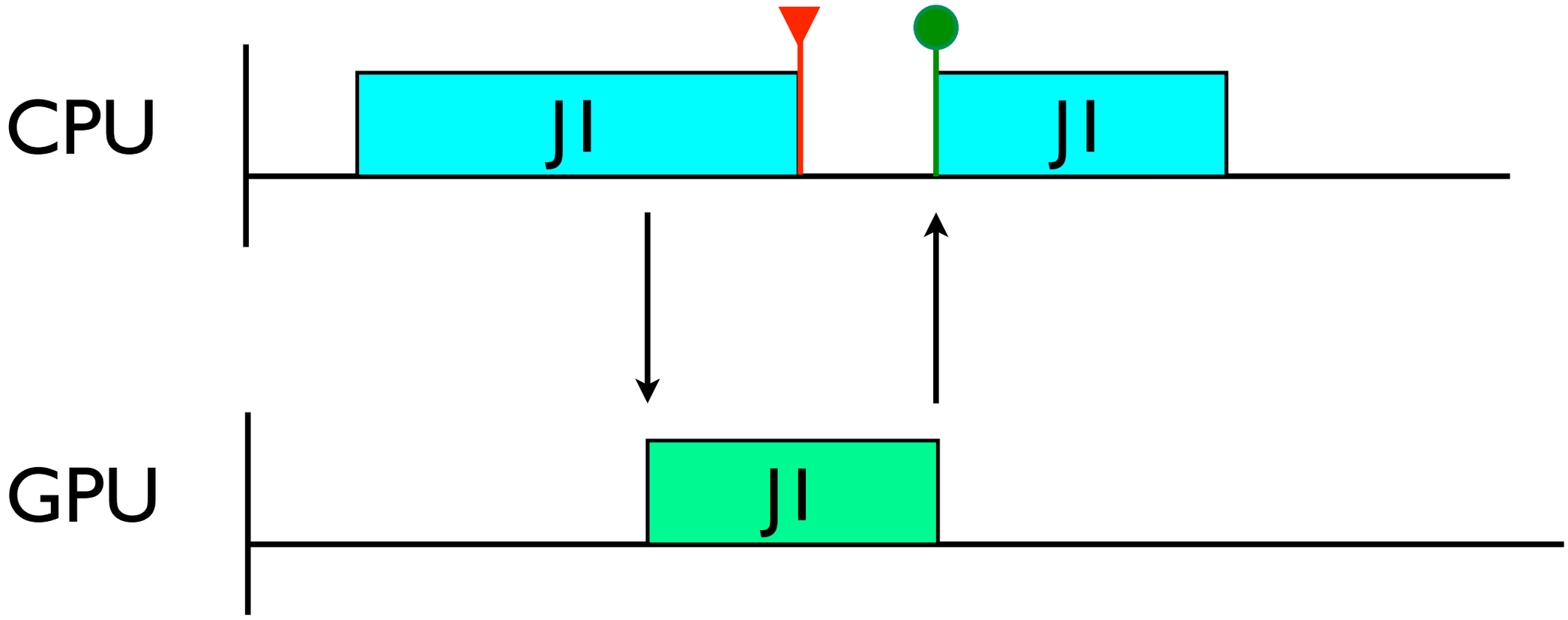


Asynchronous GPU Usage Pattern

J1 may continue executing before blocking...

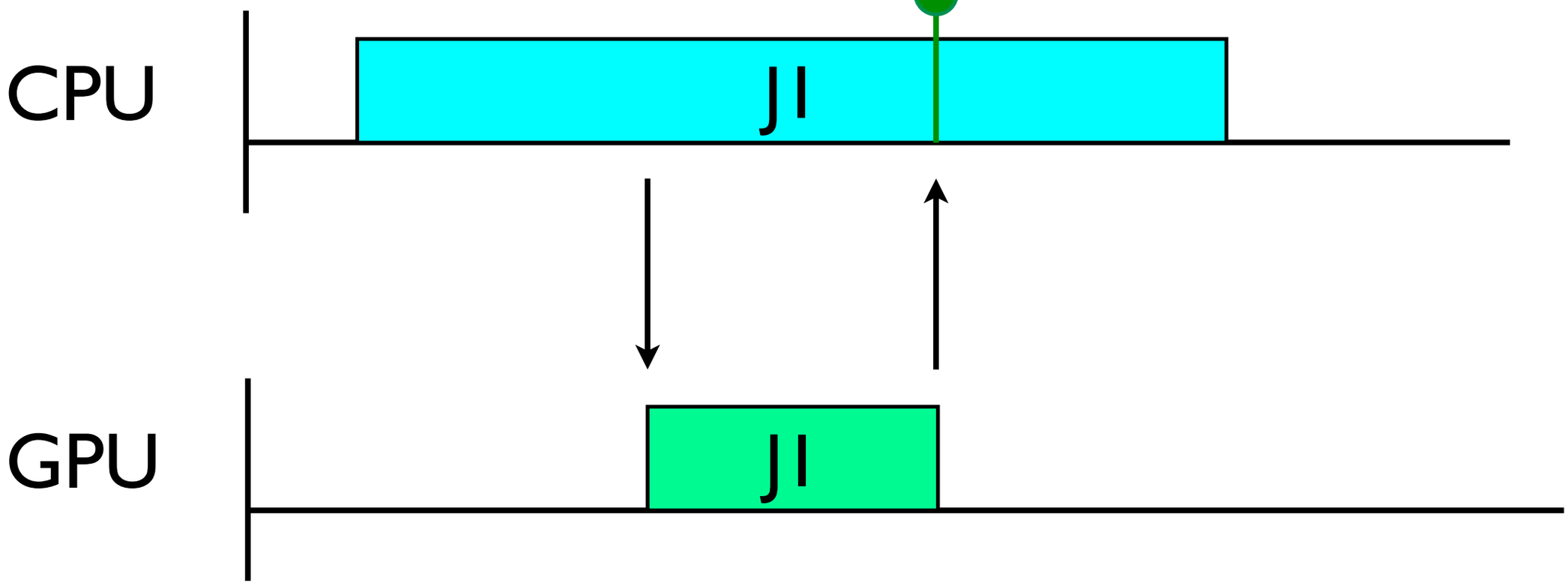


Asynchronous GPU Usage Pattern



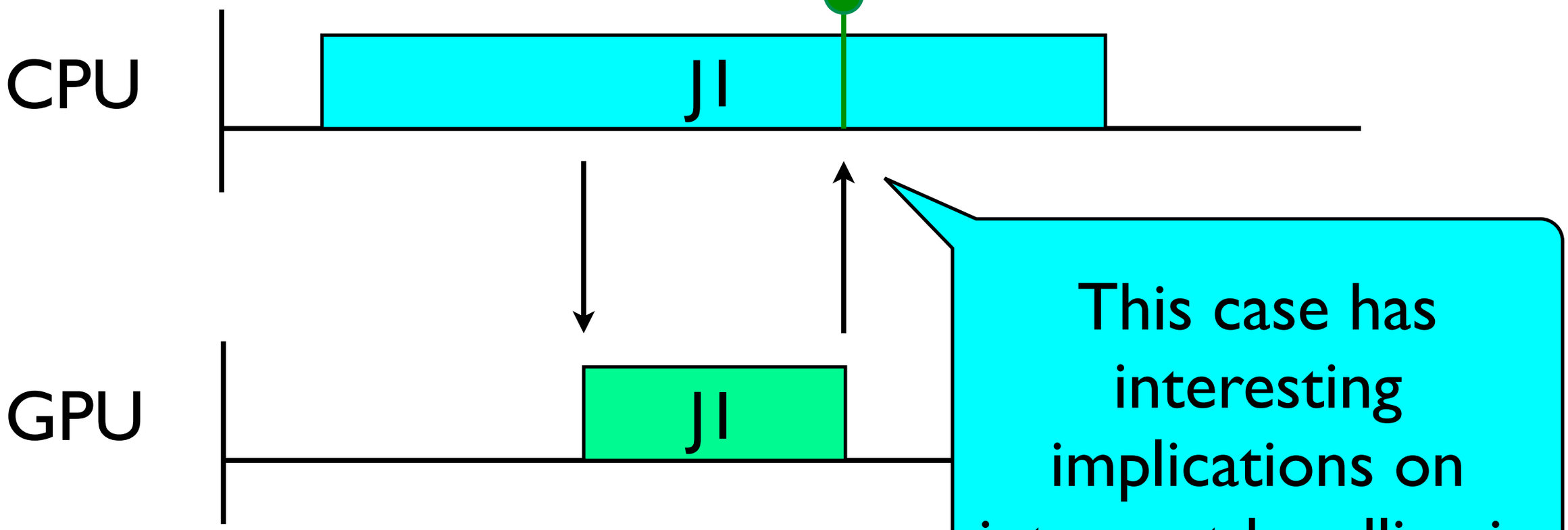
Asynchronous GPU Usage Pattern

...or may never block if GPU finishes before J1 needs results.



Asynchronous GPU Usage Pattern

...or may never block if GPU finishes before J1 needs results.



This case has interesting implications on interrupt handling in global schedulers.

Interrupt Handling

Interrupt Handling

- May arrive at unpredictable moments

Interrupt Handling

- May arrive at unpredictable moments
- Interrupt preempts currently scheduled task *and* prevents this task from resuming until interrupt is handled

Interrupt Handling

- May arrive at unpredictable moments
- Interrupt preempts currently scheduled task *and* prevents this task from resuming until interrupt is handled
- A CPU must acknowledge interrupt and may often perform additional computations

Interrupt Handling

- A CPU must acknowledge interrupt and may often perform additional computations

Interrupt Handling

- A CPU must acknowledge interrupt and may often perform additional computations
- Handling often *split*:

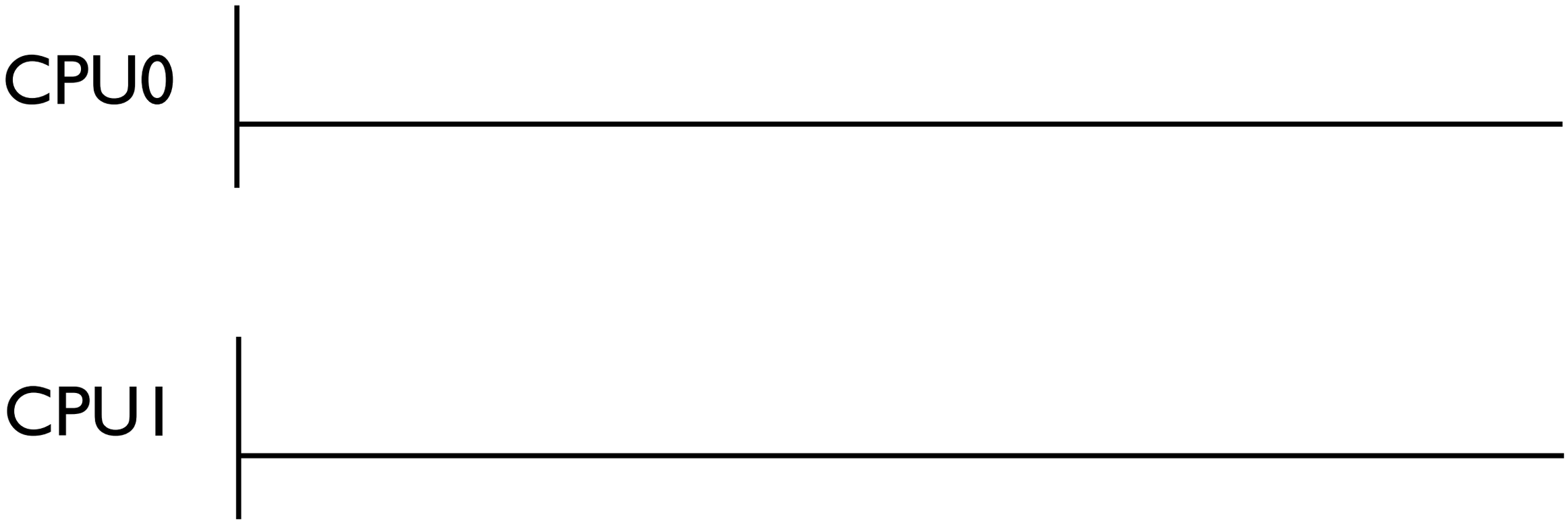
Interrupt Handling

- A CPU must acknowledge interrupt and may often perform additional computations
- Handling often *split*:
  - **Top Half**: performs acknowledgement

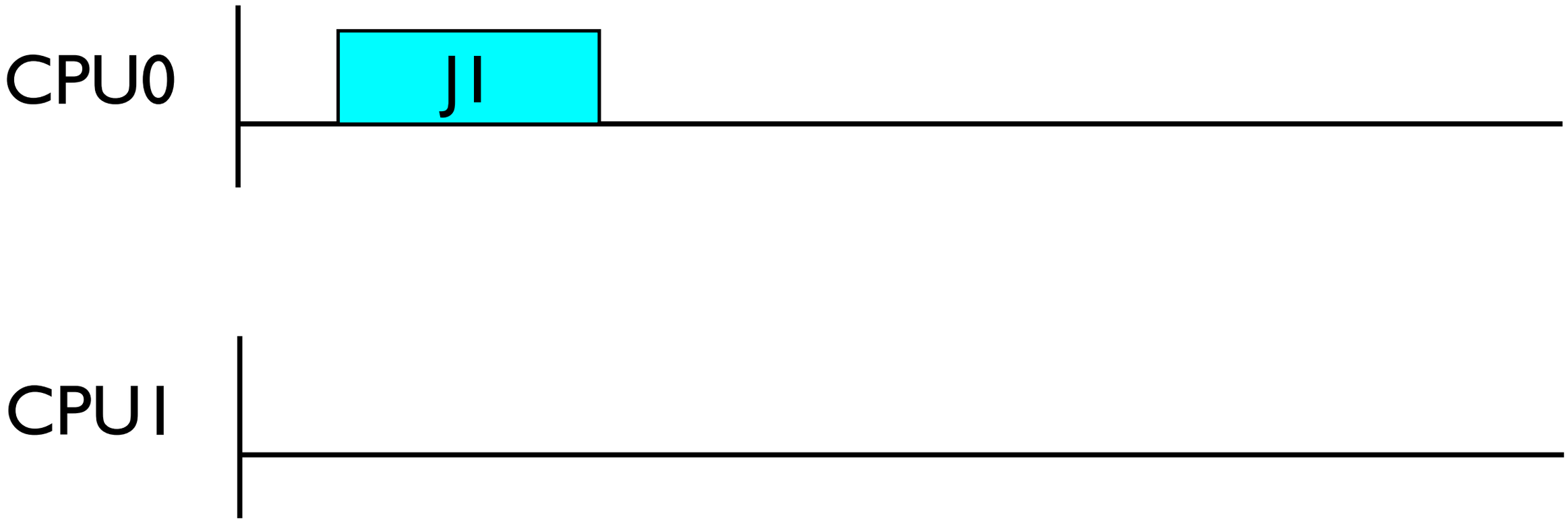
Interrupt Handling

- A CPU must acknowledge interrupt and may often perform additional computations
- Handling often *split*:
  - **Top Half**: performs acknowledgement
  - **Bottom Half**: performs computations

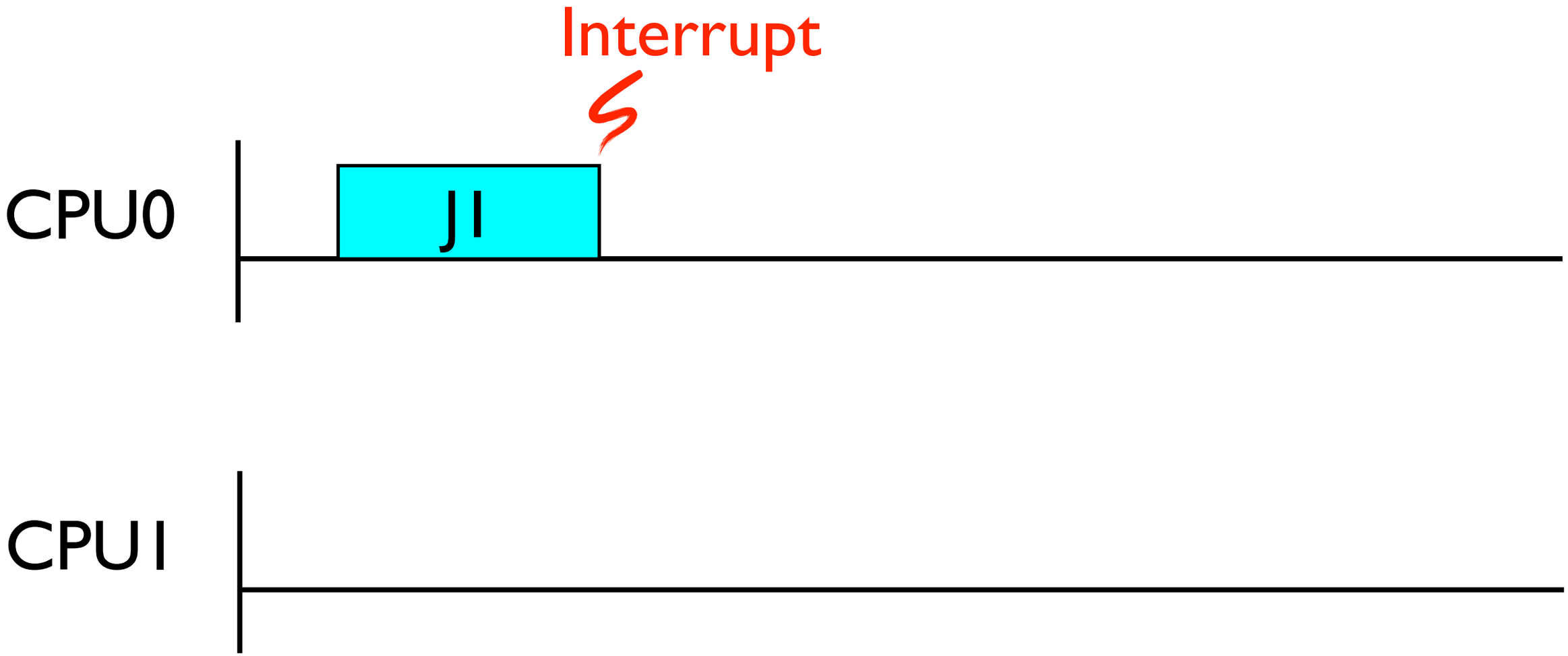
Interrupt Handling



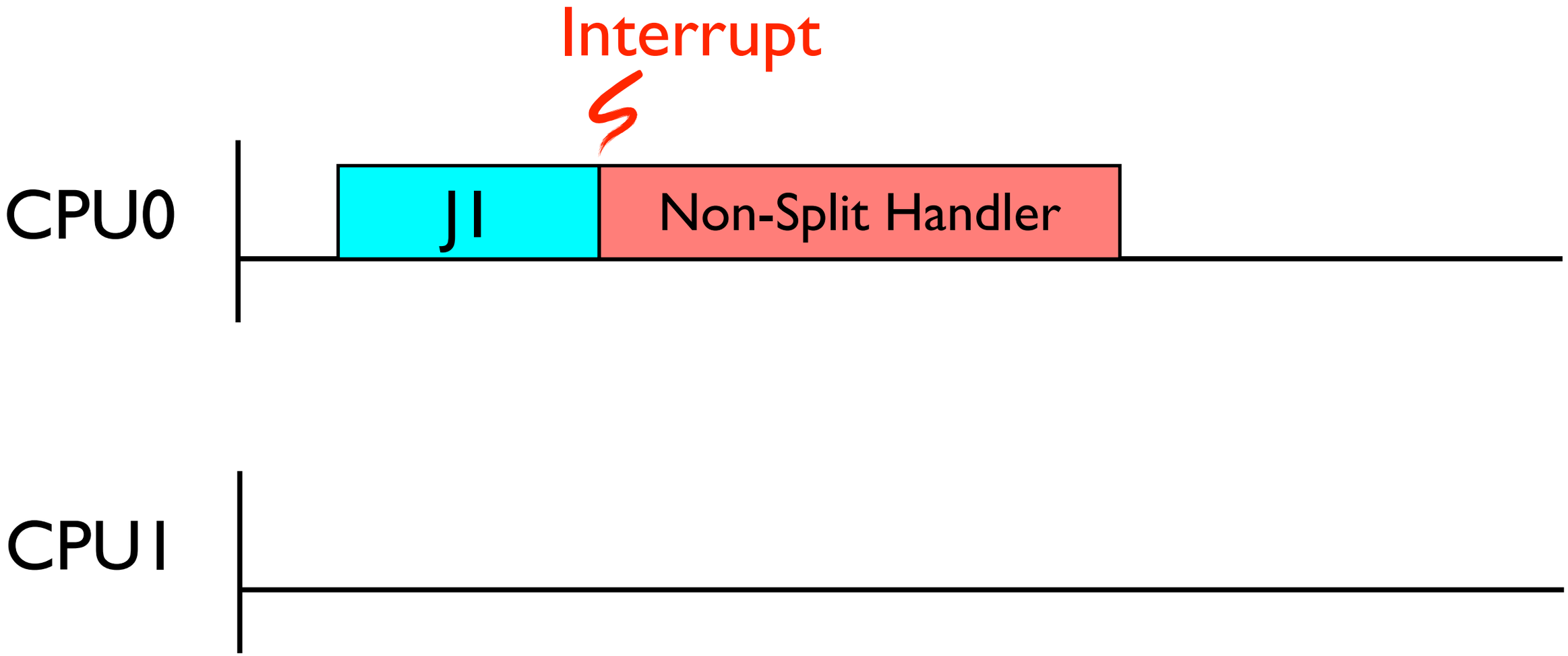
Interrupt Handling



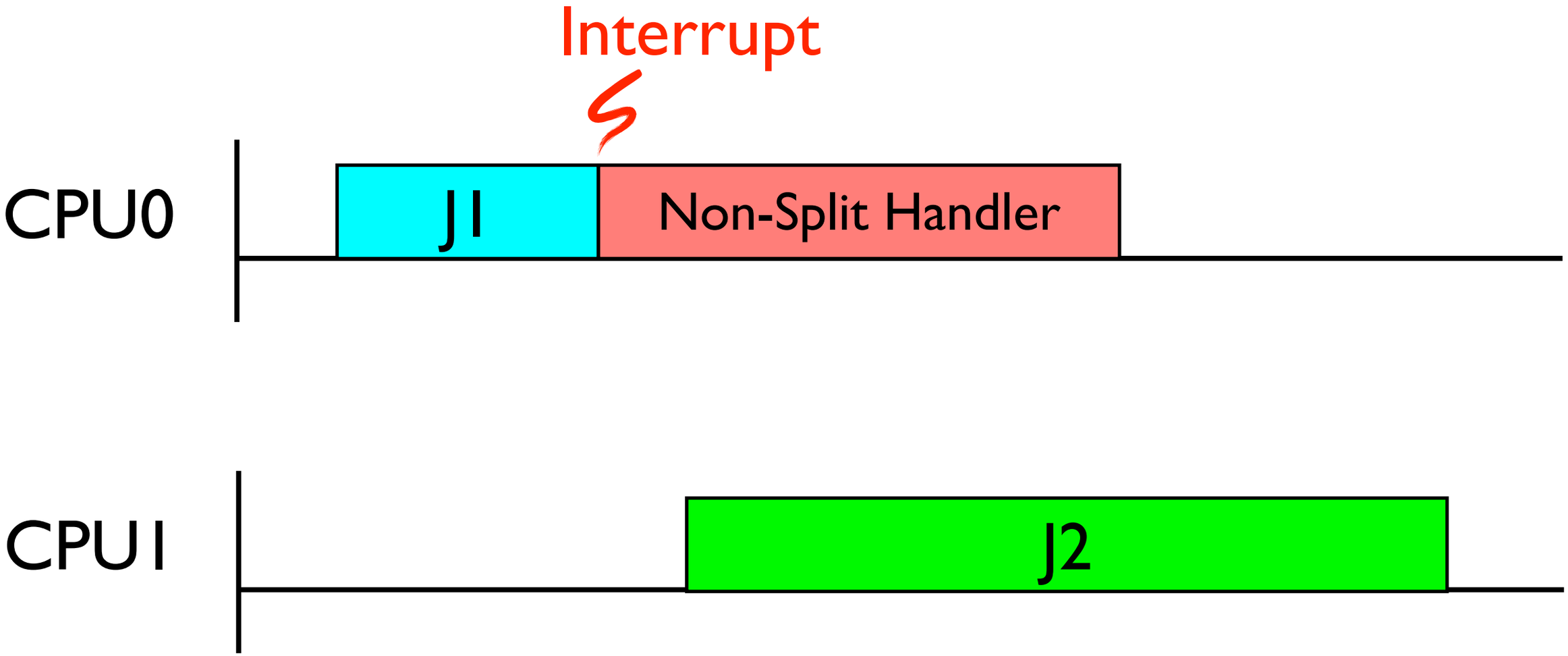
Interrupt Handling



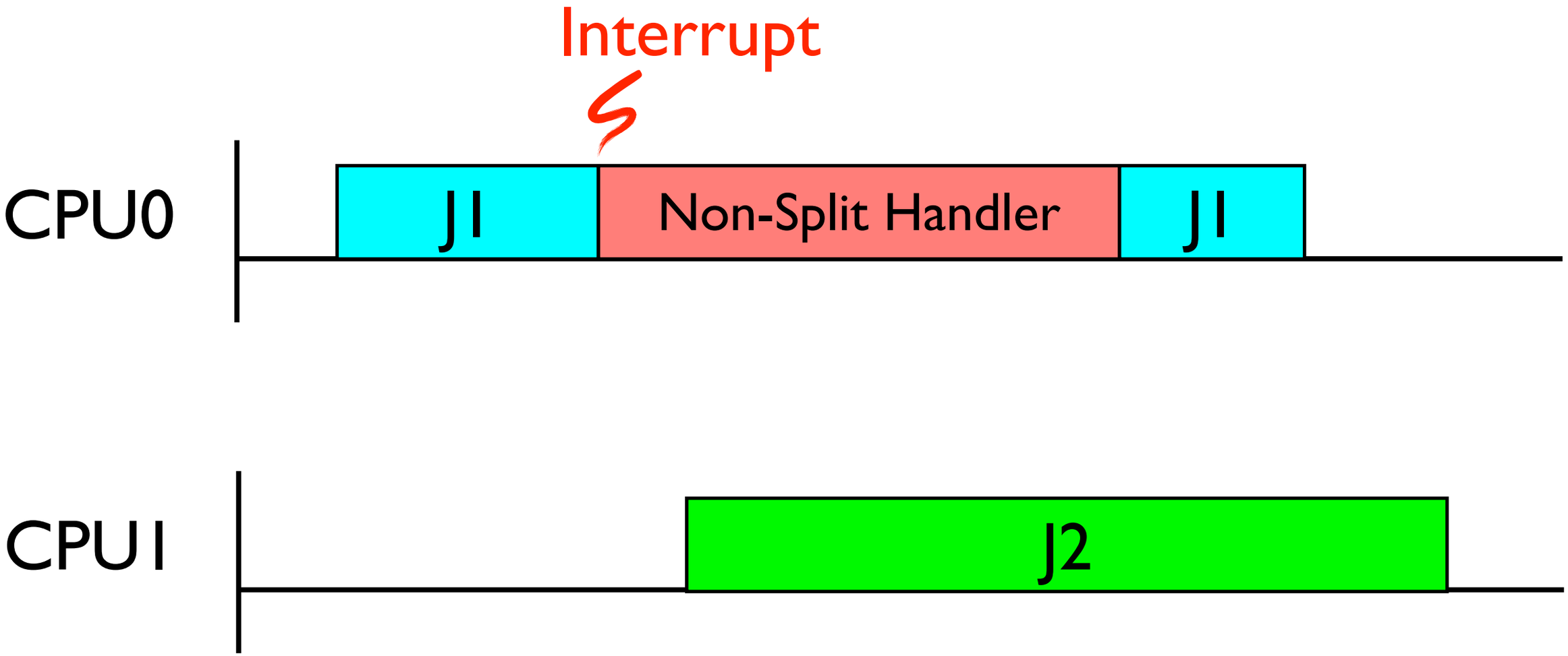
Interrupt Handling



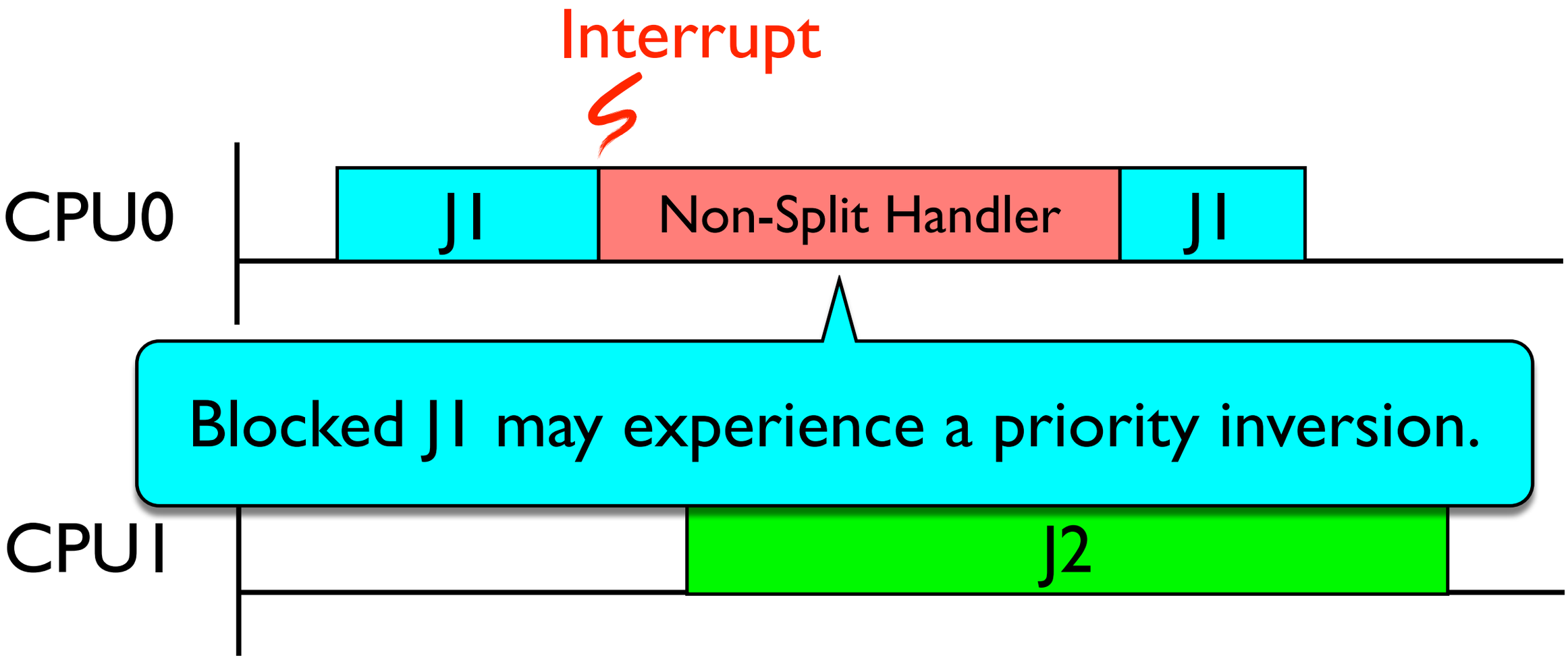
Interrupt Handling



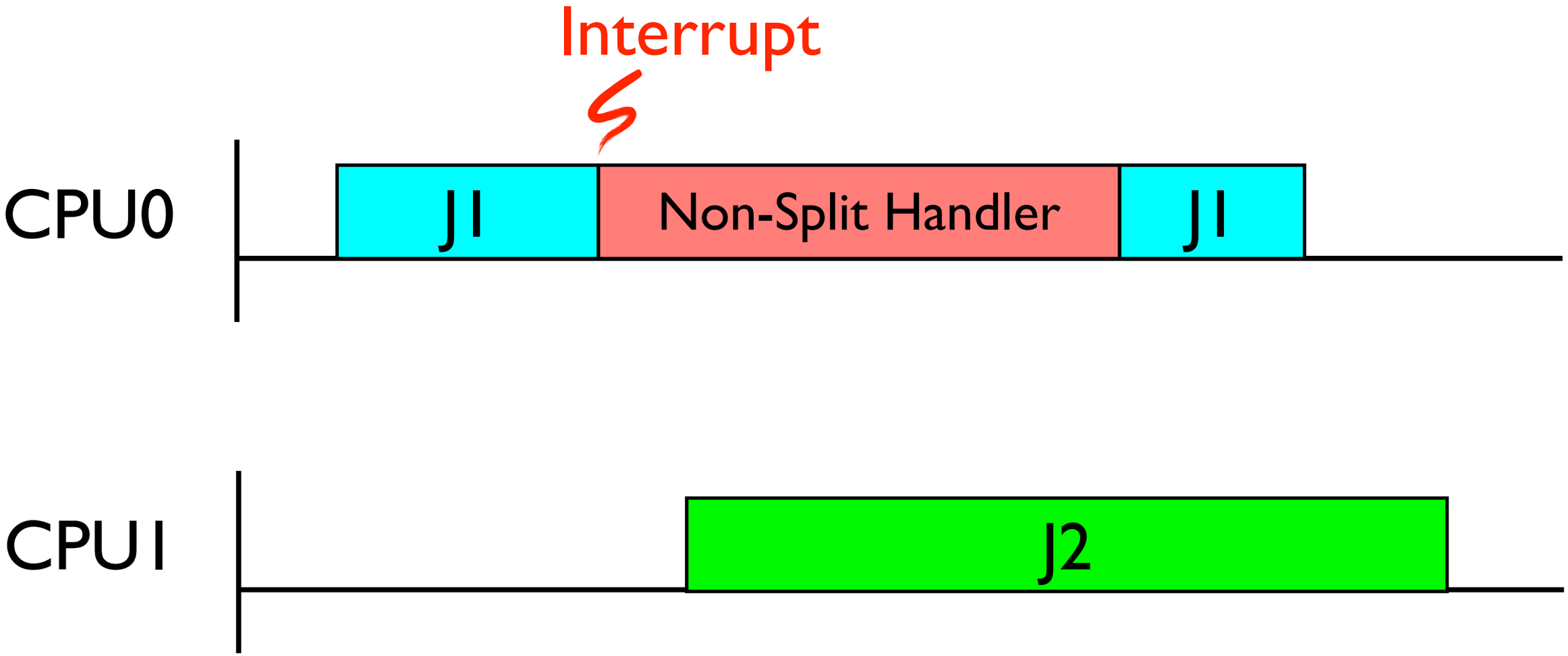
Interrupt Handling



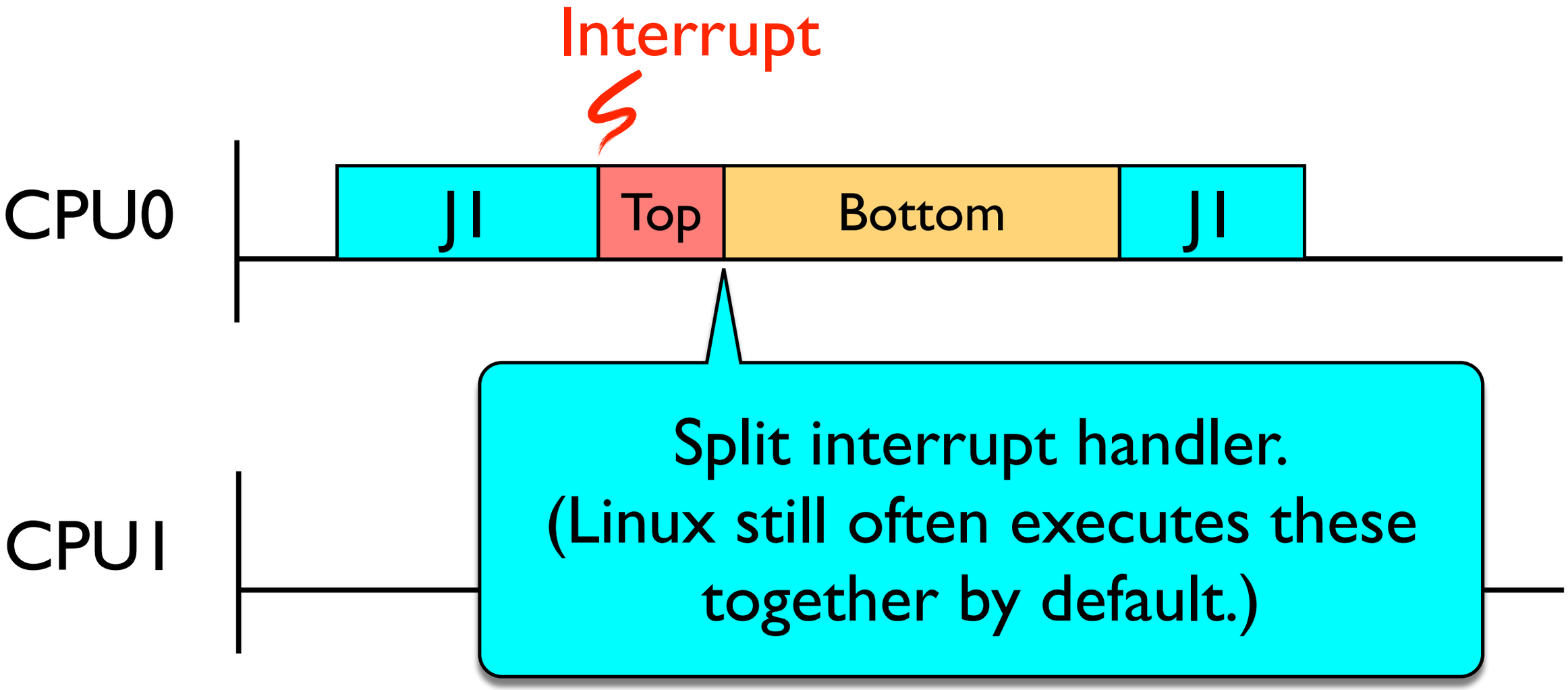
Interrupt Handling



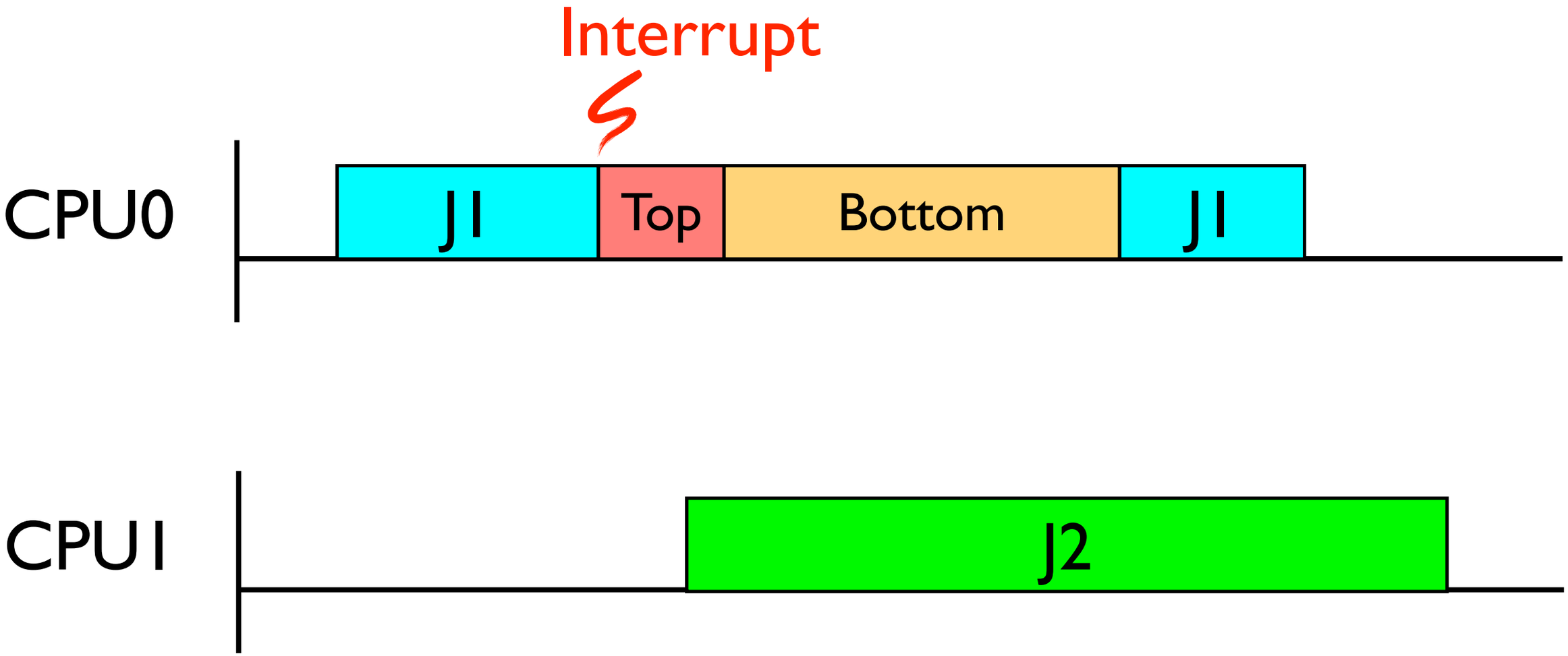
Interrupt Handling



Interrupt Handling

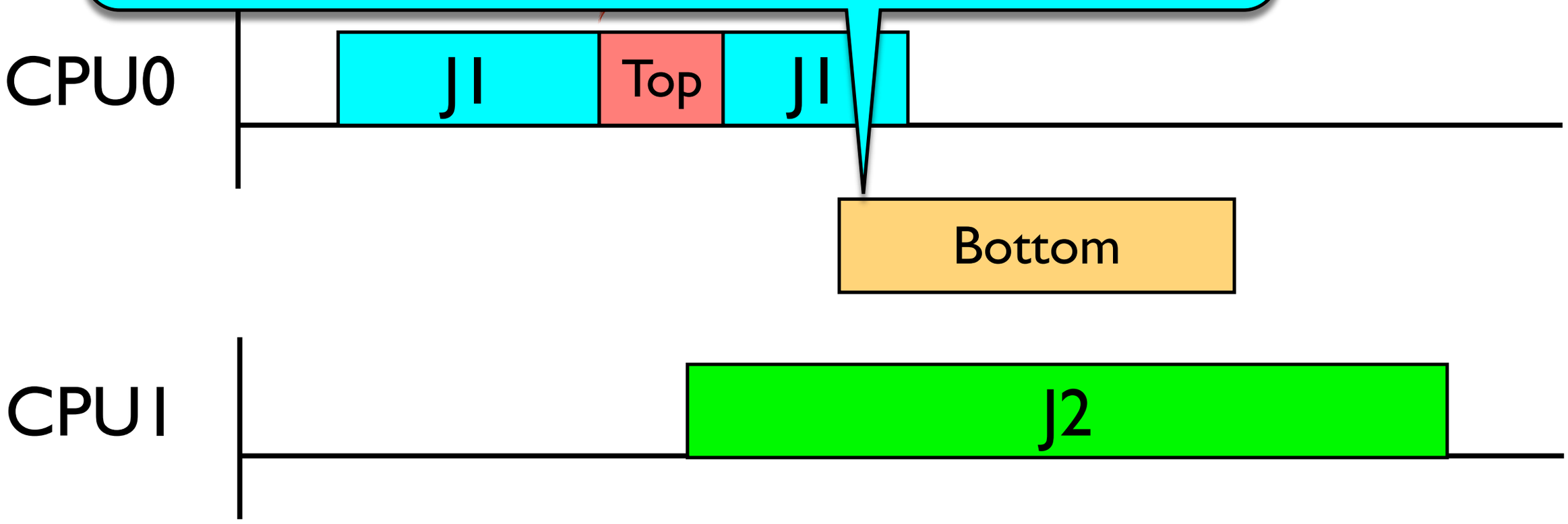


Interrupt Handling



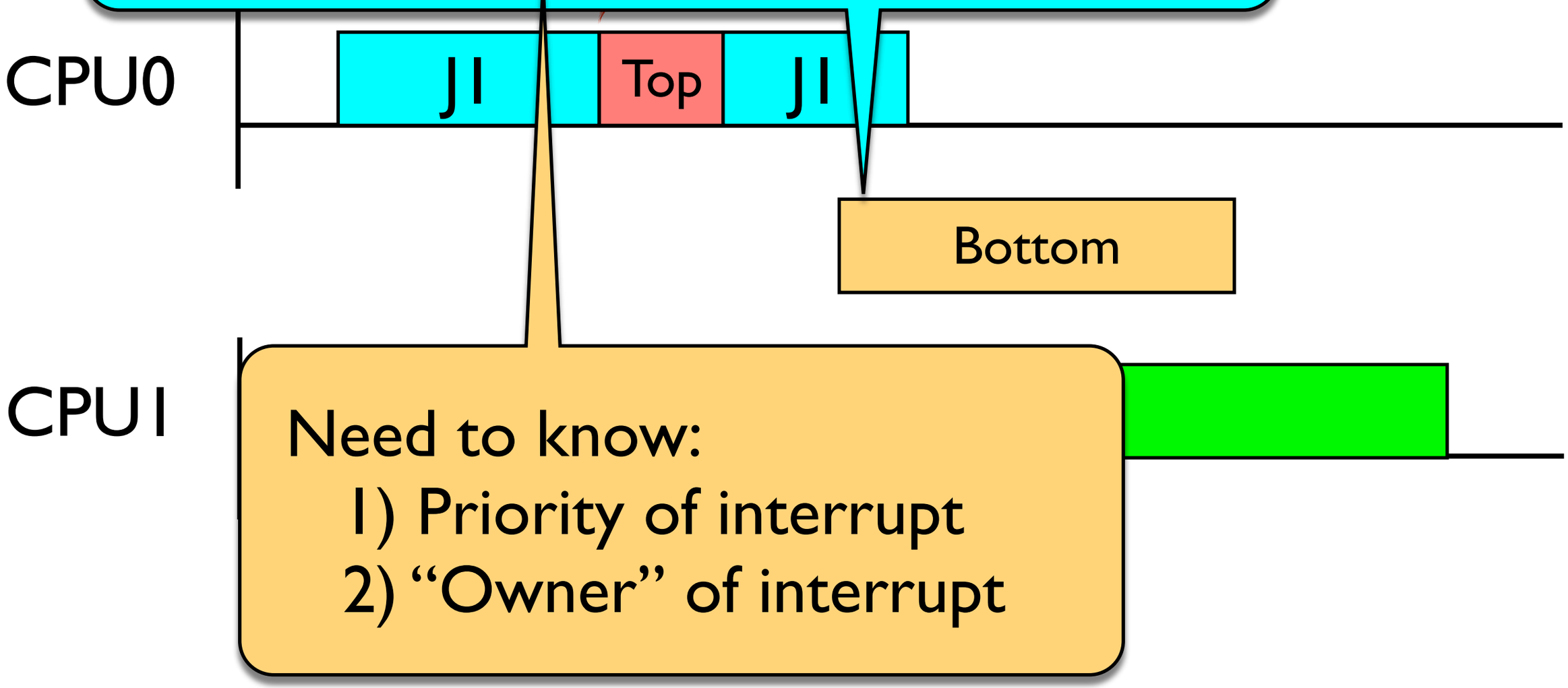
Real-time approaches (usually) schedule bottom halves in a thread (fixed or inherited priority) or in a container.

Should handler preempt J2?



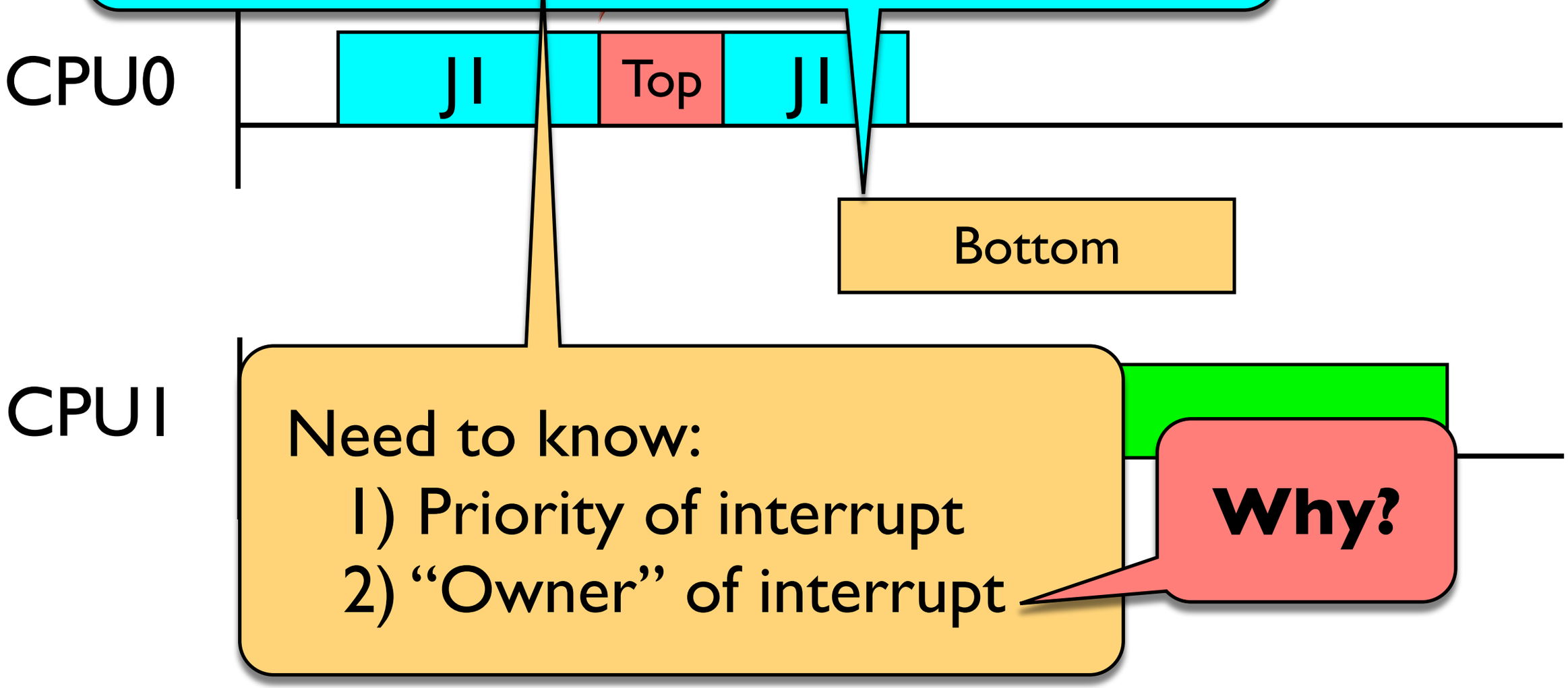
Real-time approaches (usually) schedule bottom halves in a thread (fixed or inherited priority) or in a container.

Should handler preempt J2?

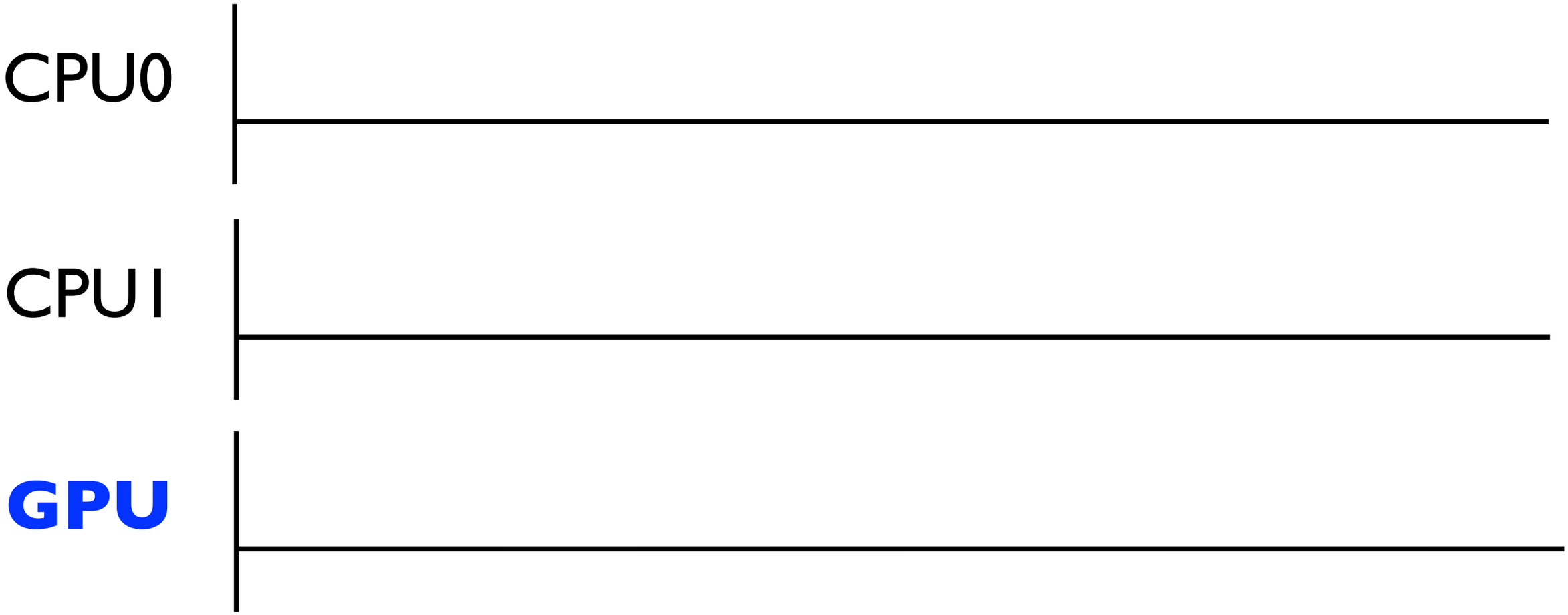


Real-time approaches (usually) schedule bottom halves in a thread (fixed or inherited priority) or in a container.

Should handler preempt J2?

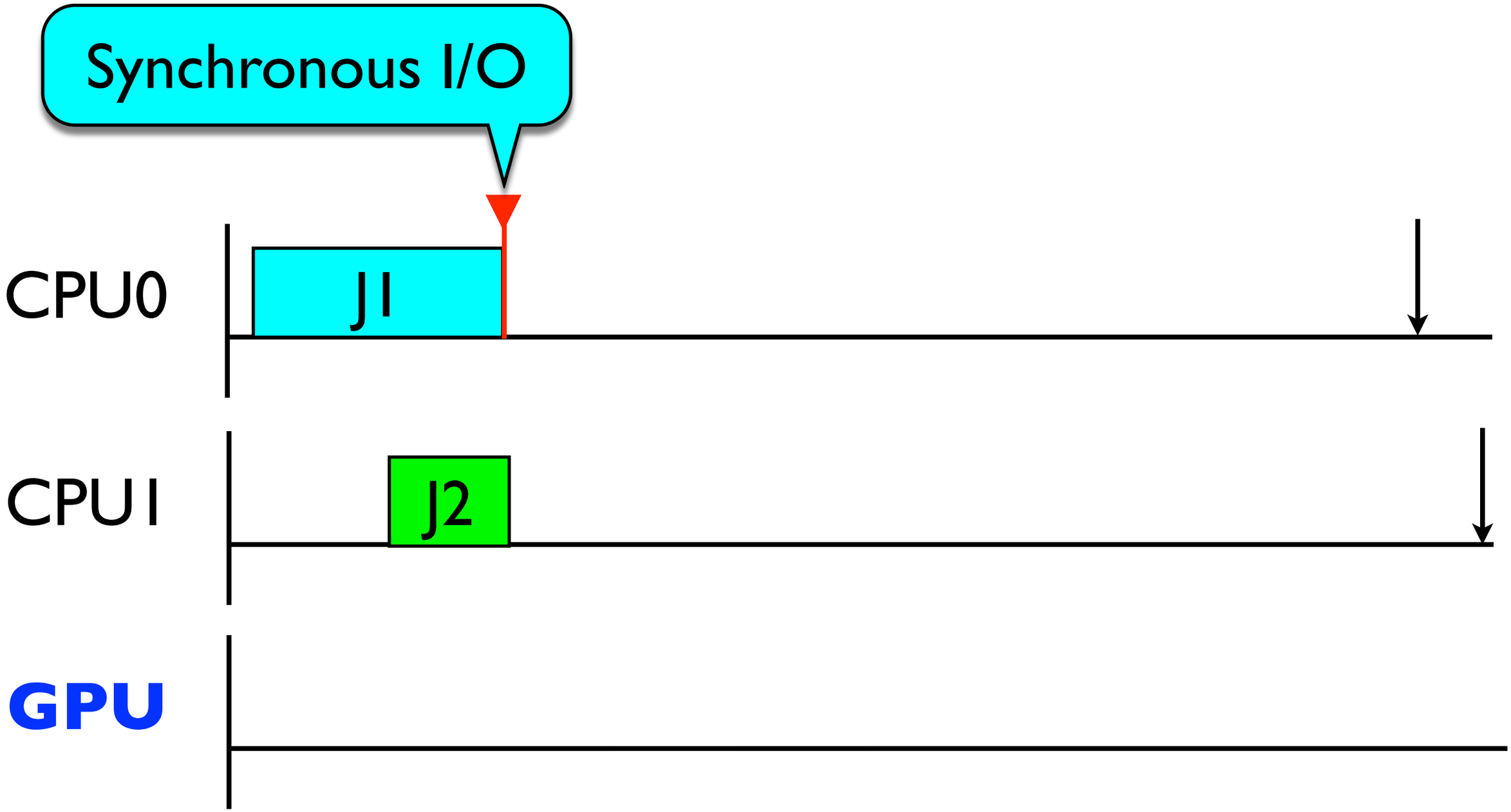


Interrupt Ownership

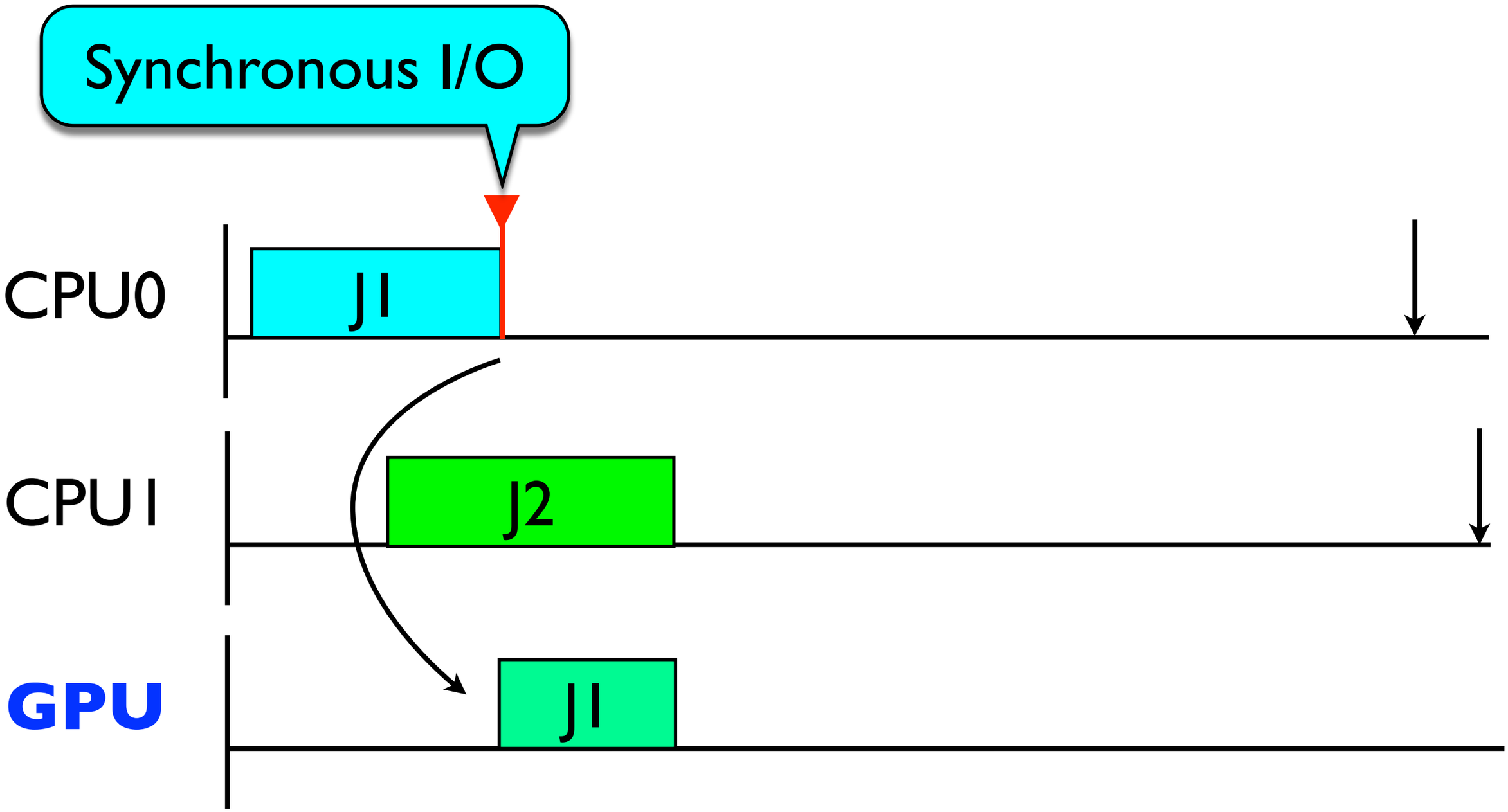


Interrupt Ownership

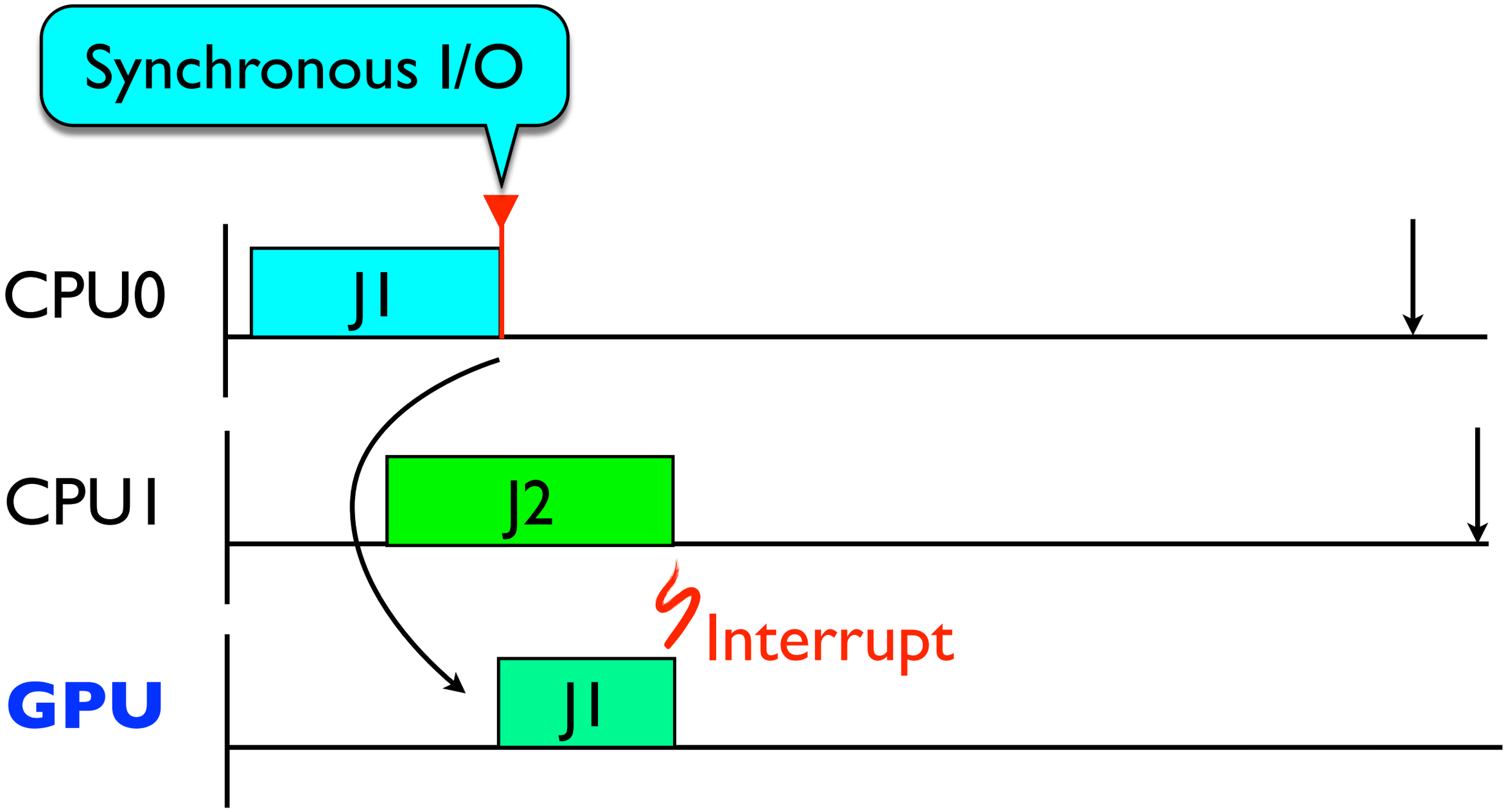
Interrupt Ownership



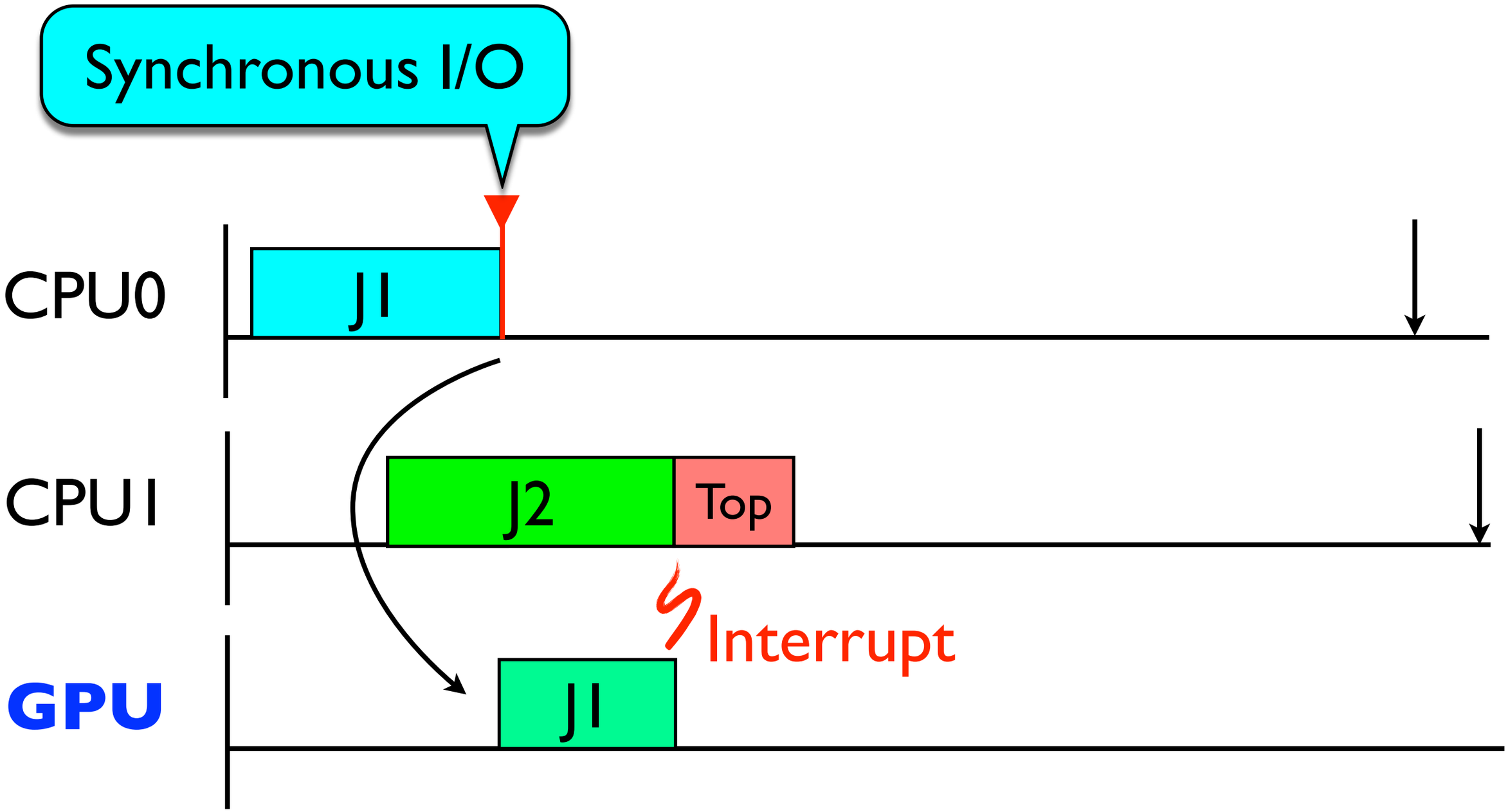
Interrupt Ownership



Interrupt Ownership



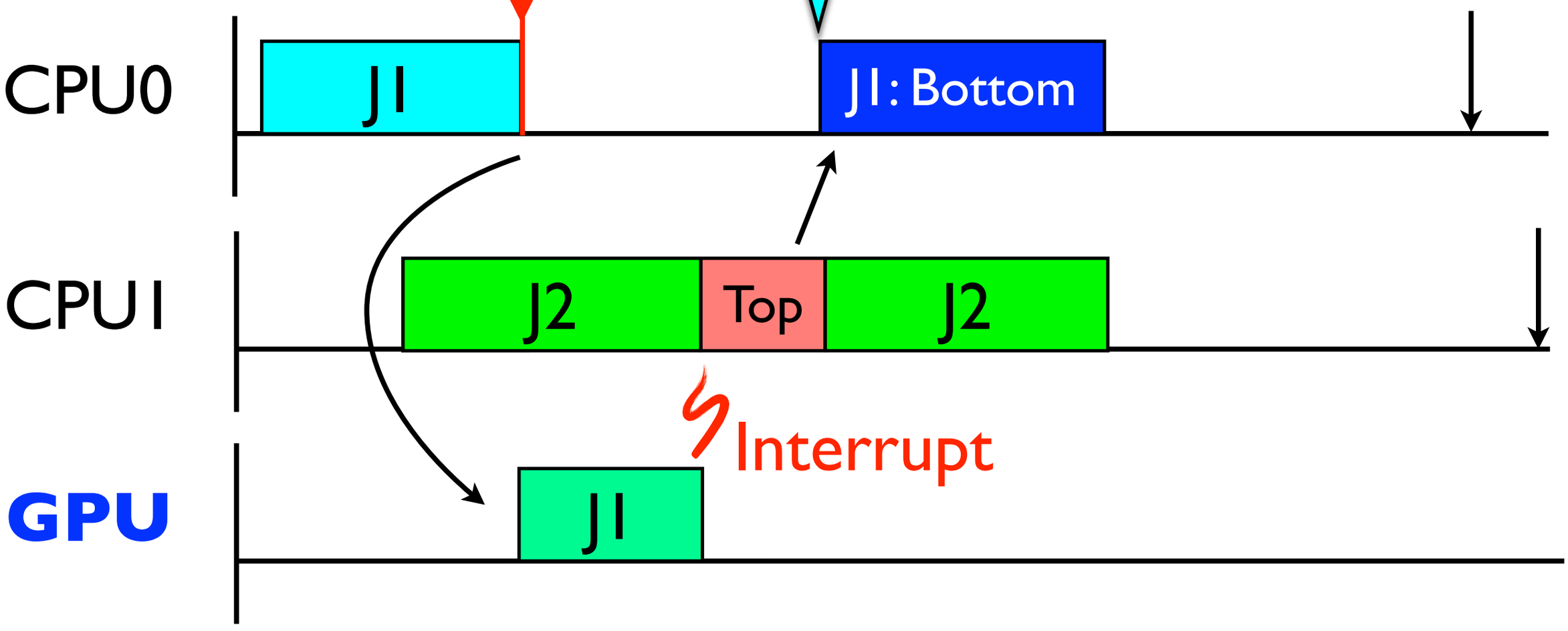
Interrupt Ownership



Interrupt Ownership

Synchron

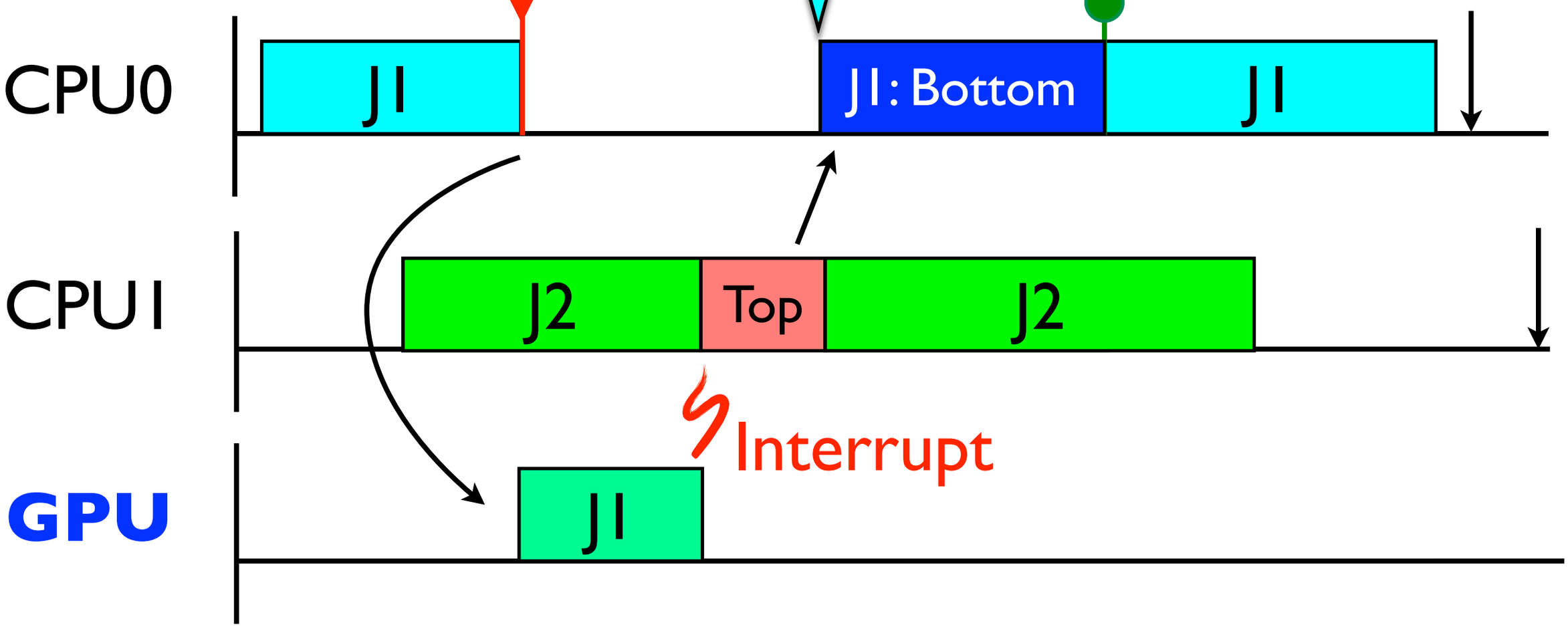
Schedule bottom half.
Thread inherits priority of J1.



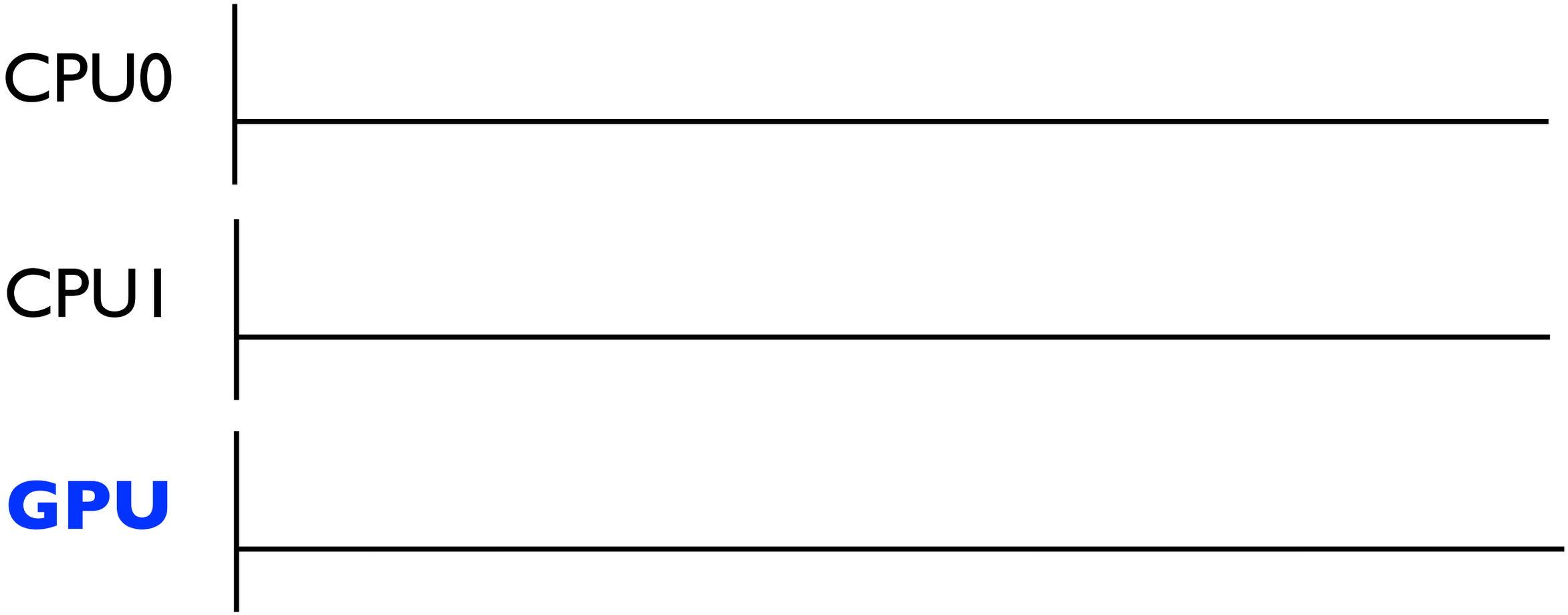
Interrupt Ownership

Synchron

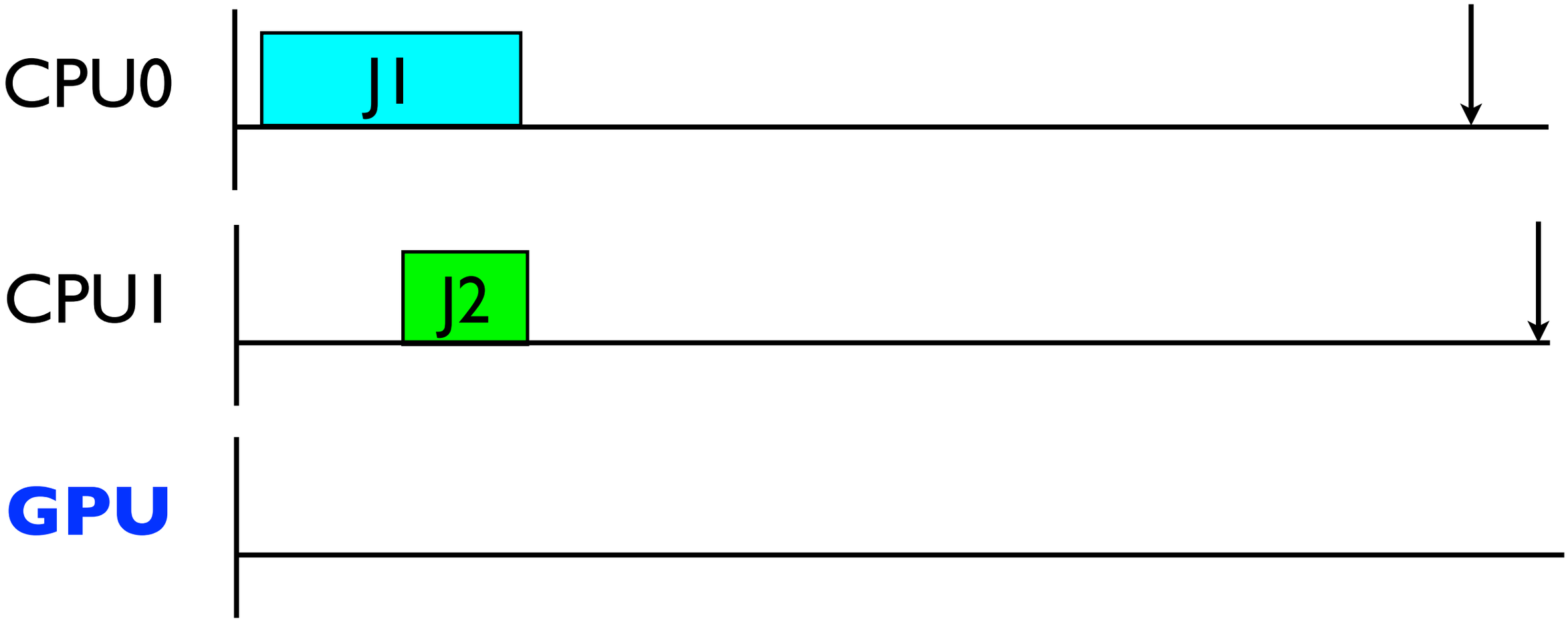
Schedule bottom half.
Thread inherits priority of J1.



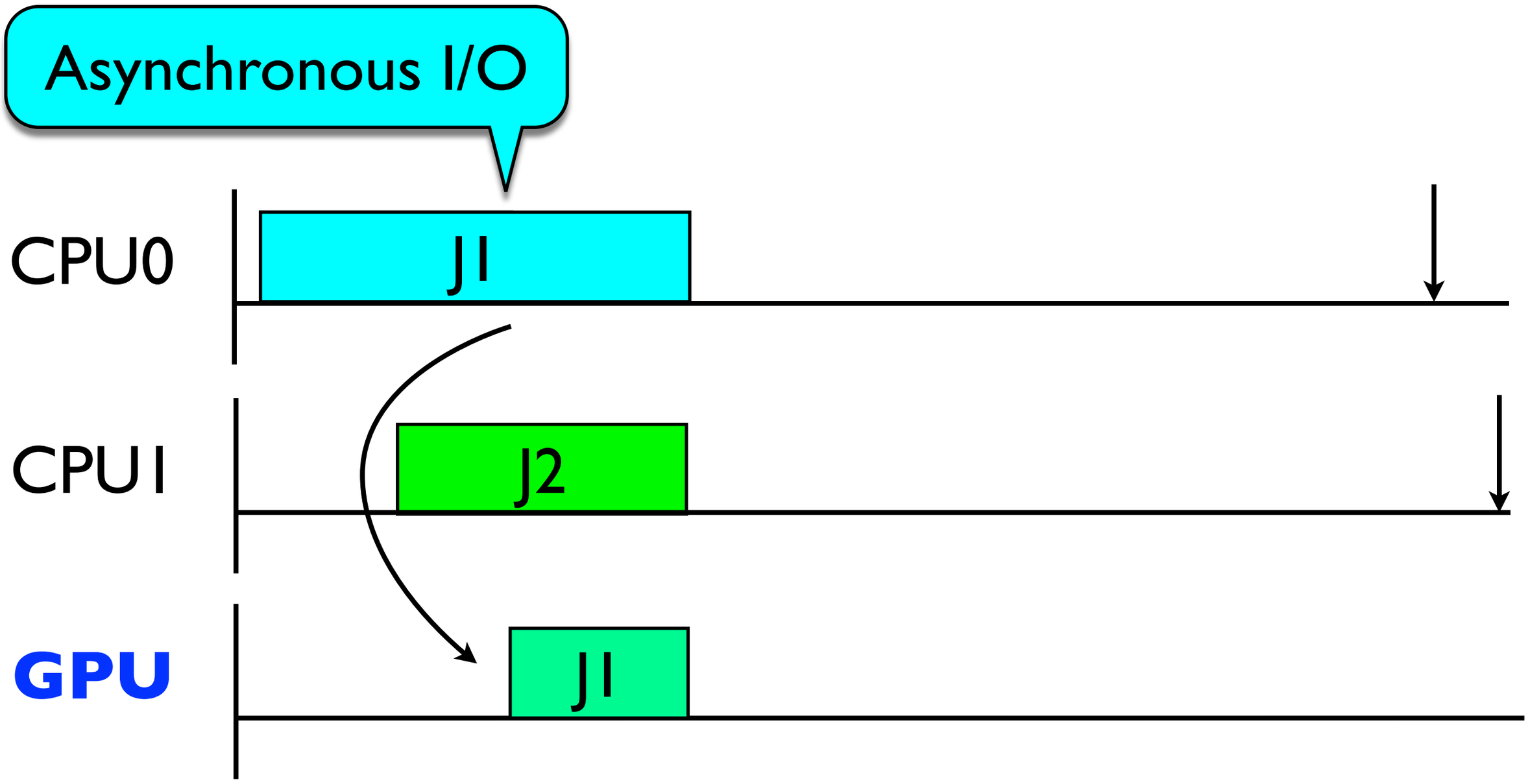
Interrupt Ownership



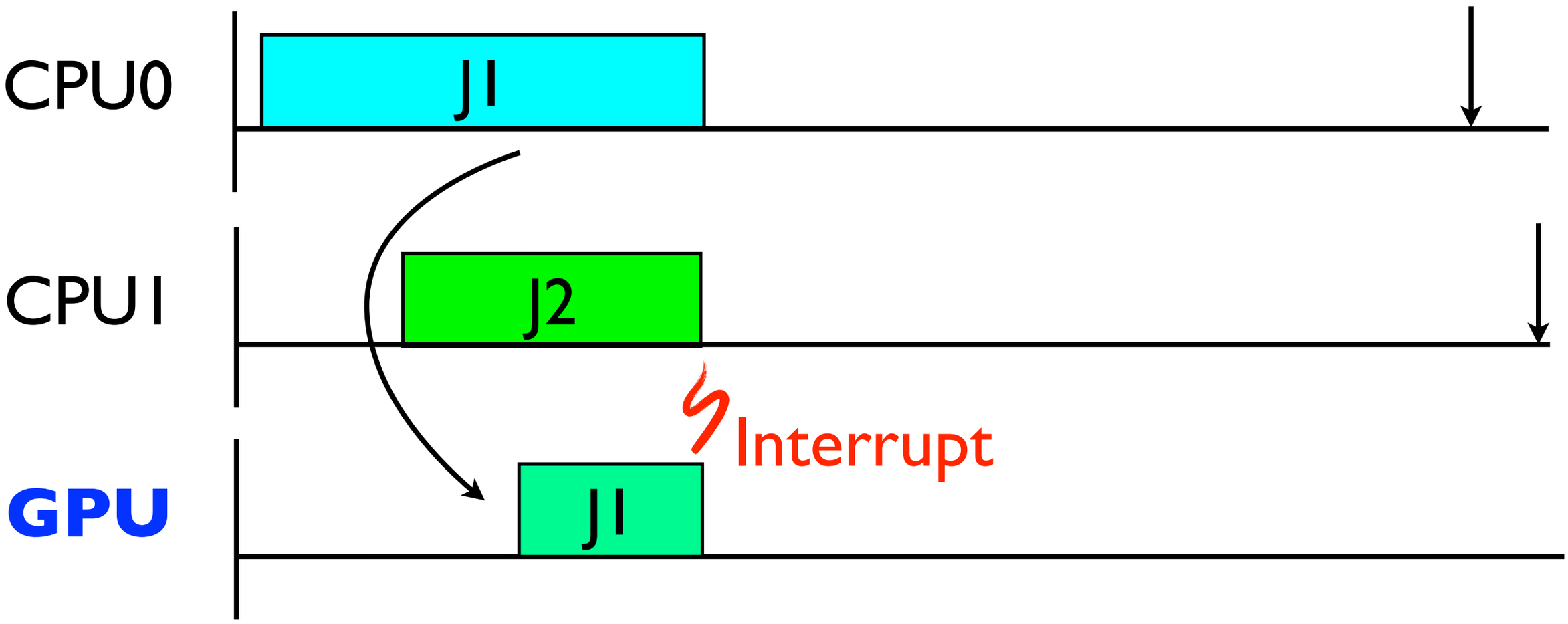
Interrupt Ownership



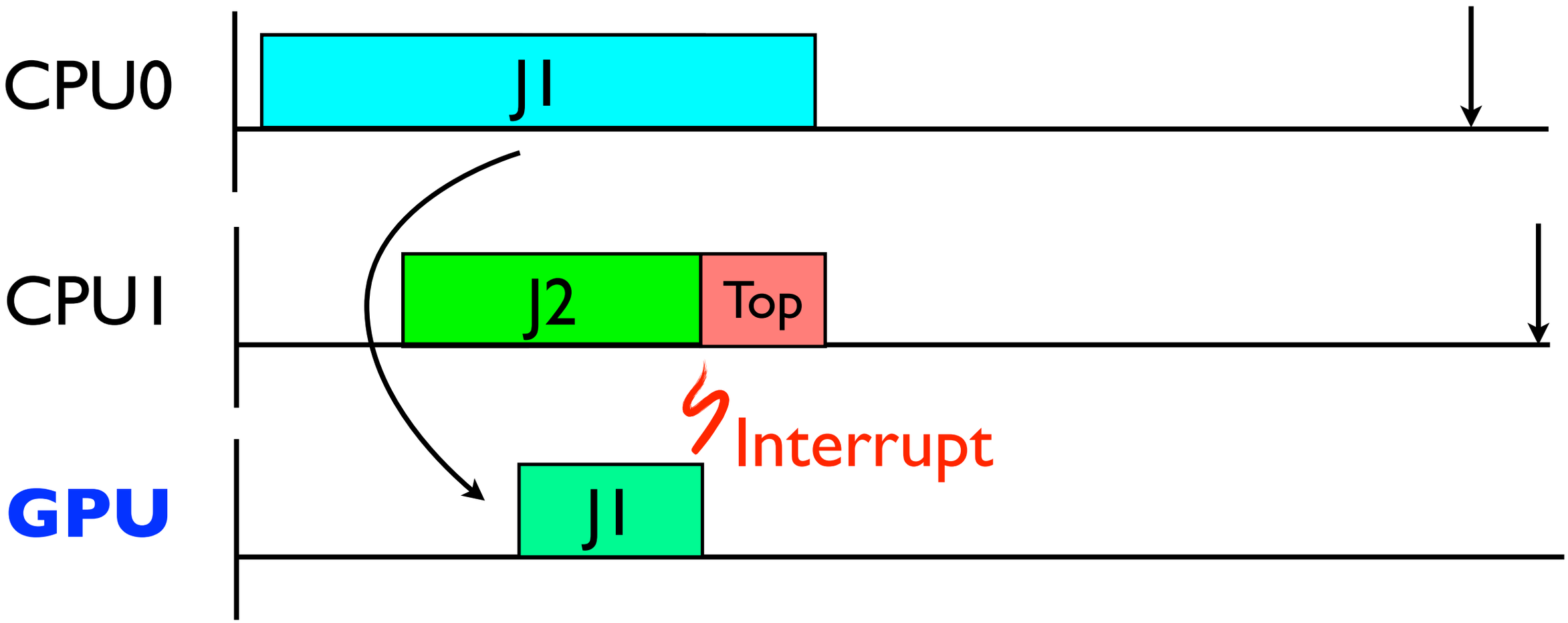
Interrupt Ownership



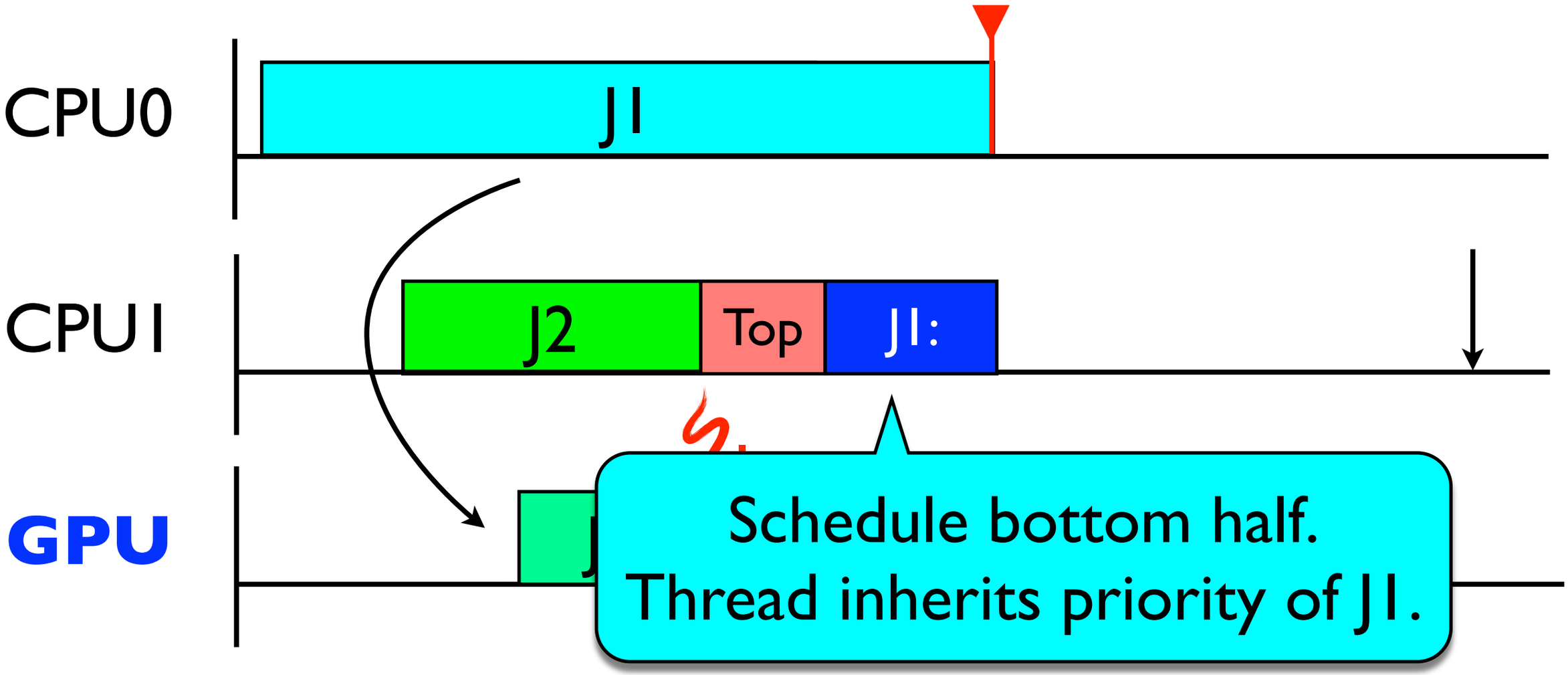
Interrupt Ownership



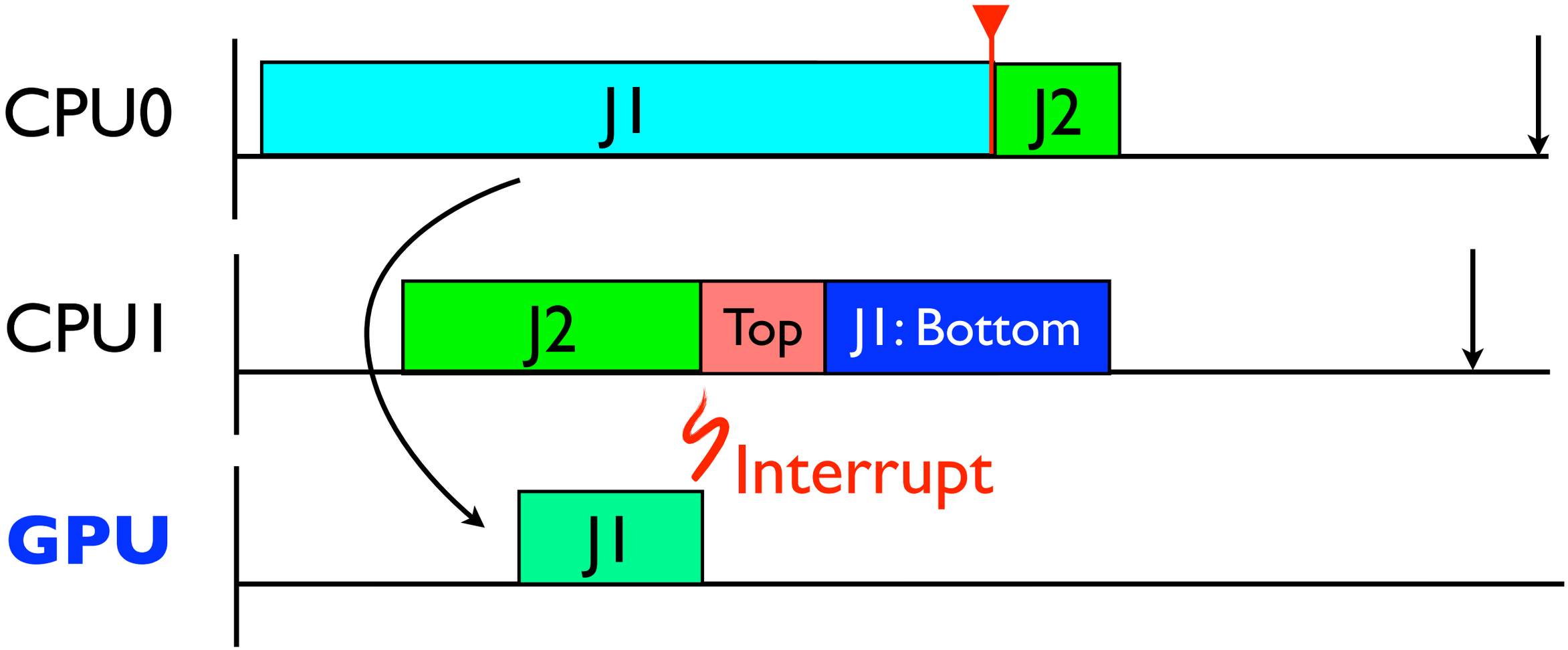
Interrupt Ownership



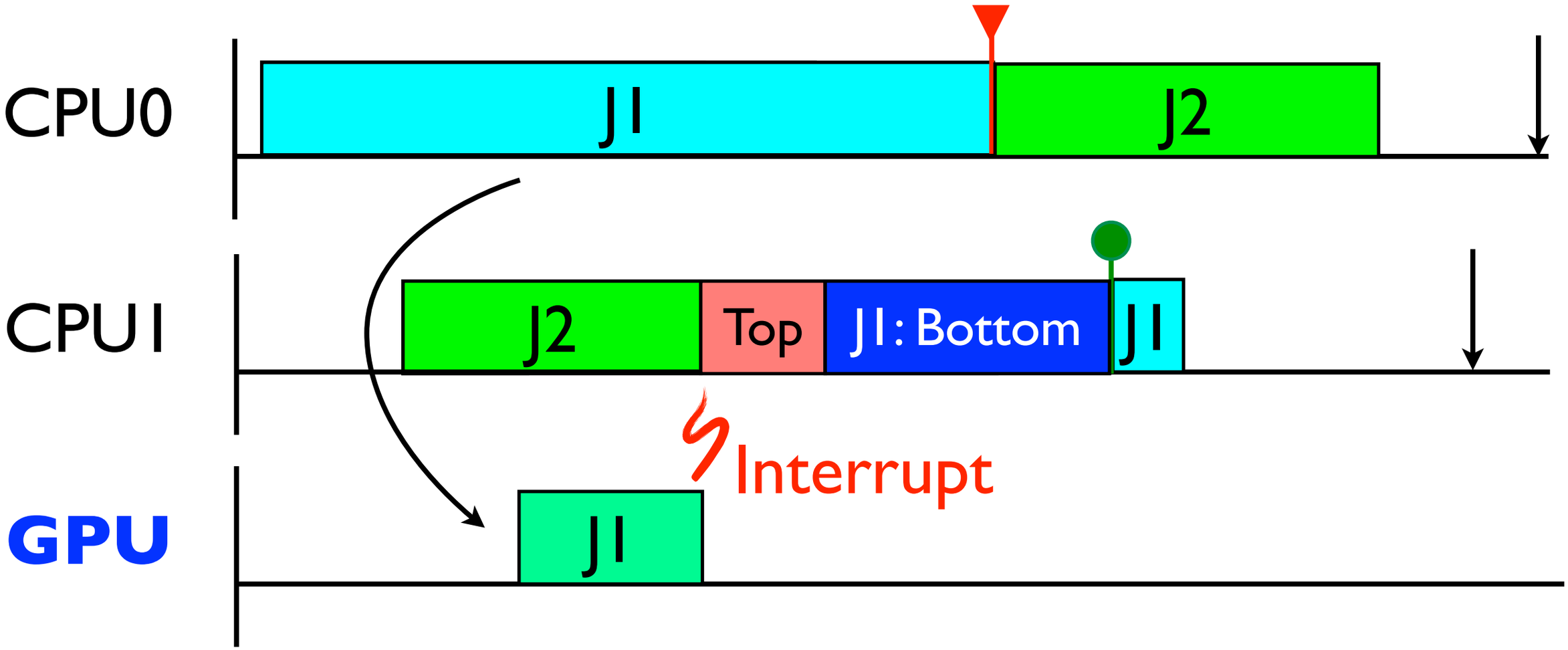
Interrupt Ownership



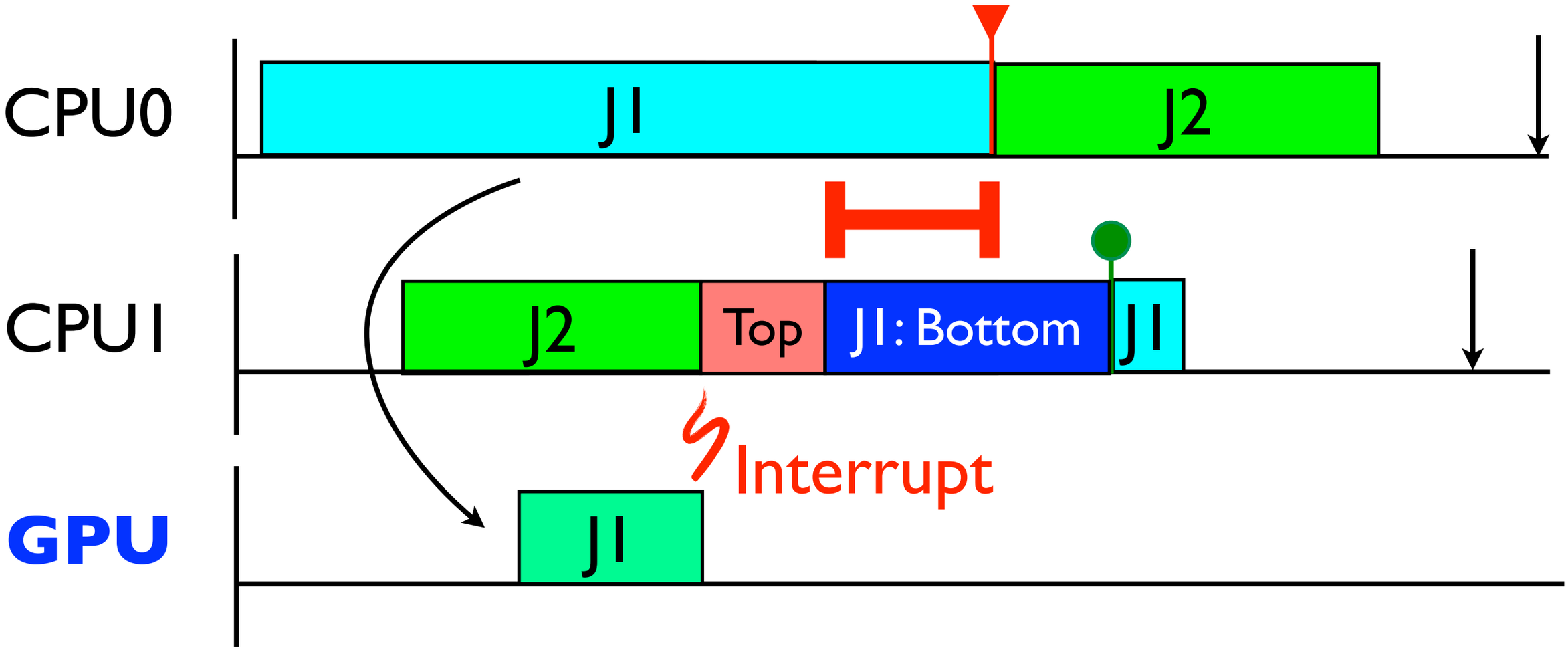
Interrupt Ownership



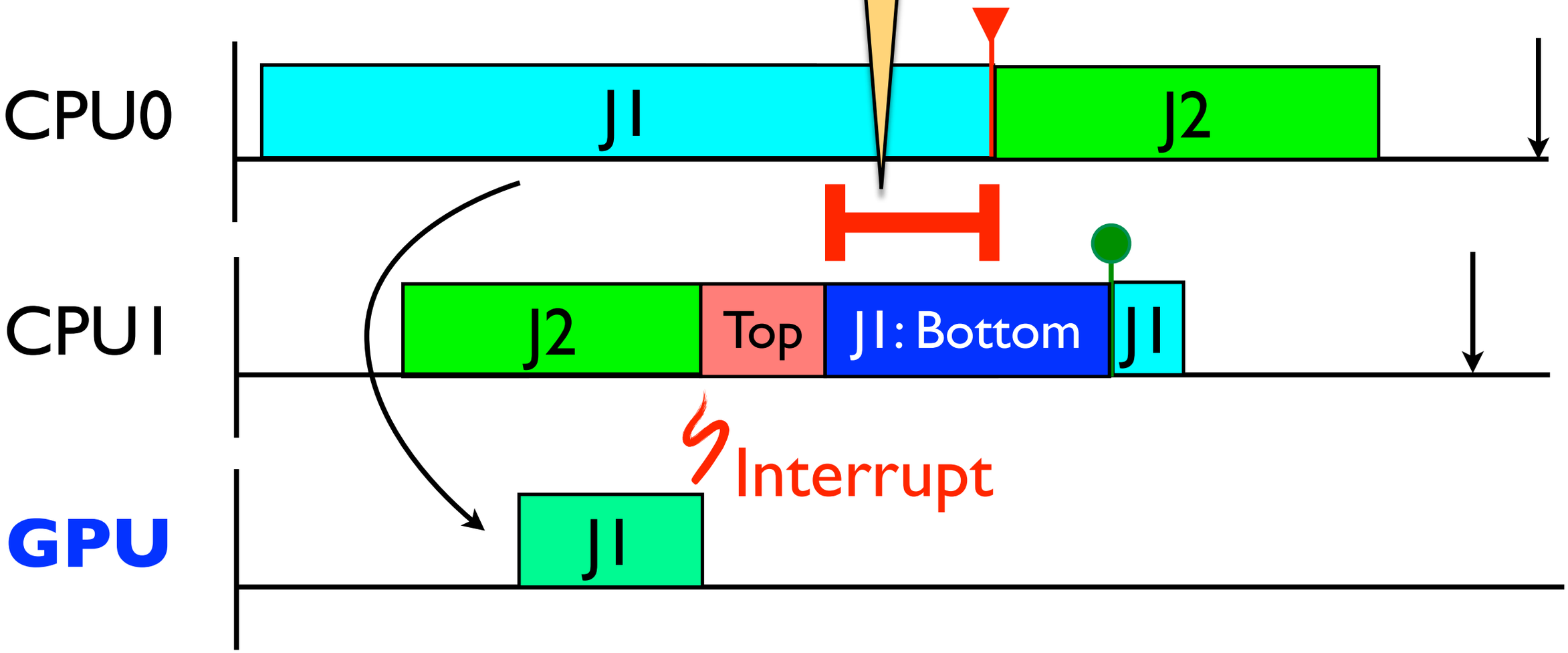
Interrupt Ownership



Interrupt Ownership

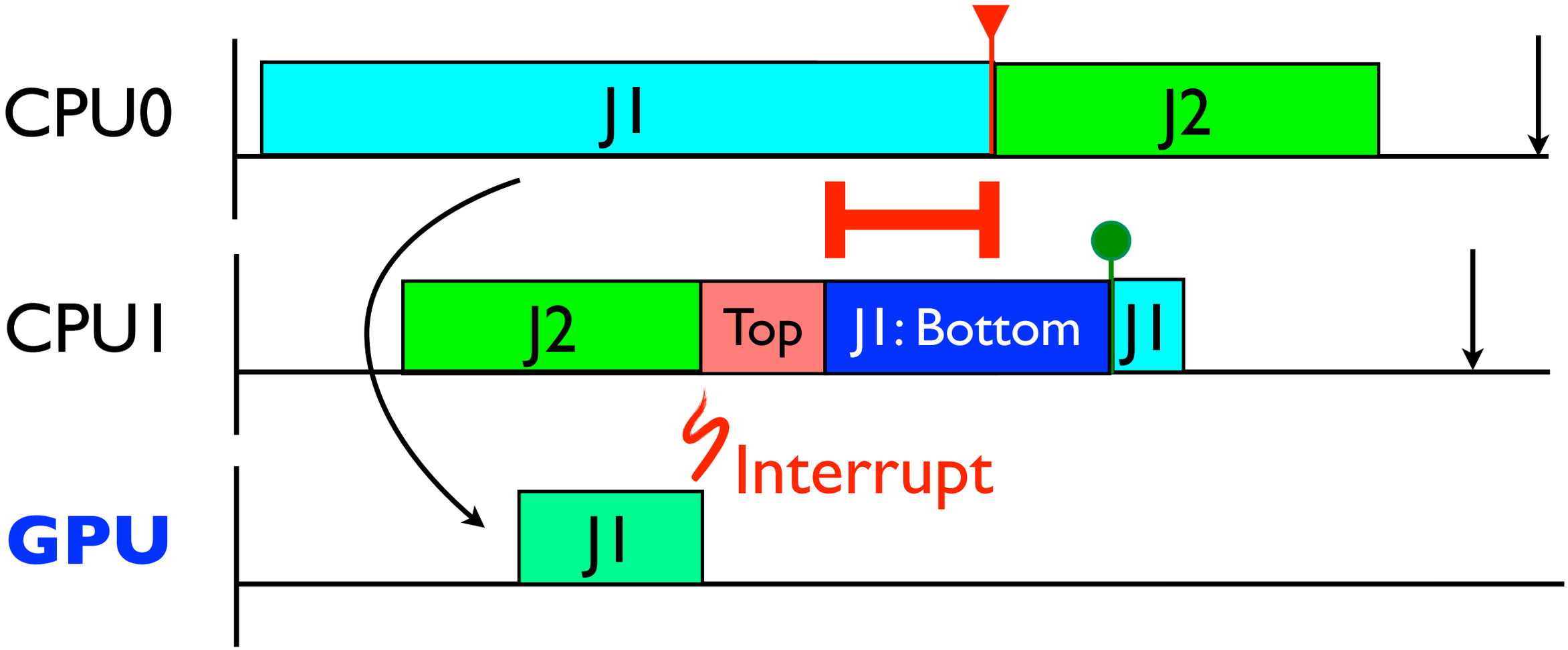


Two threads with same identity!
Breaks single threaded sporadic task model!

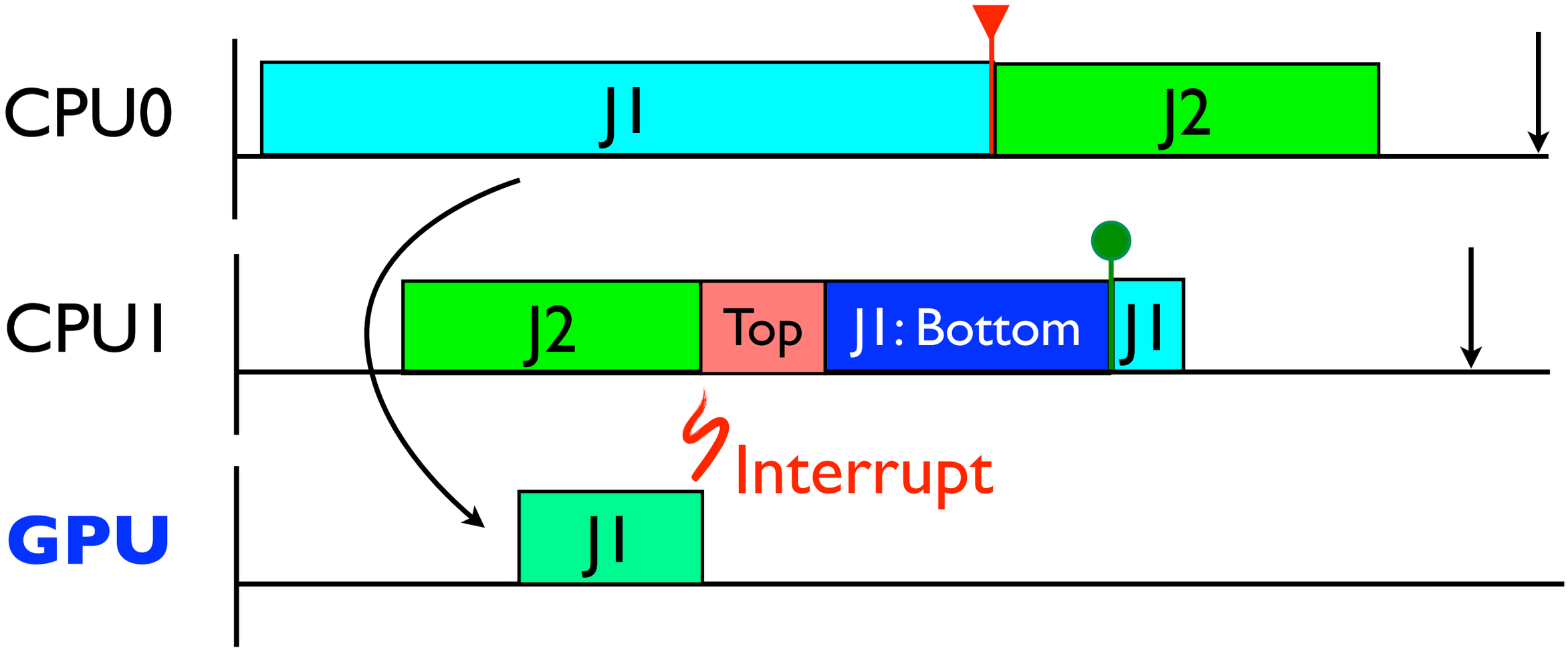


Interrupt Ownership

This can only occur under global scheduler with asynchronous I/O.

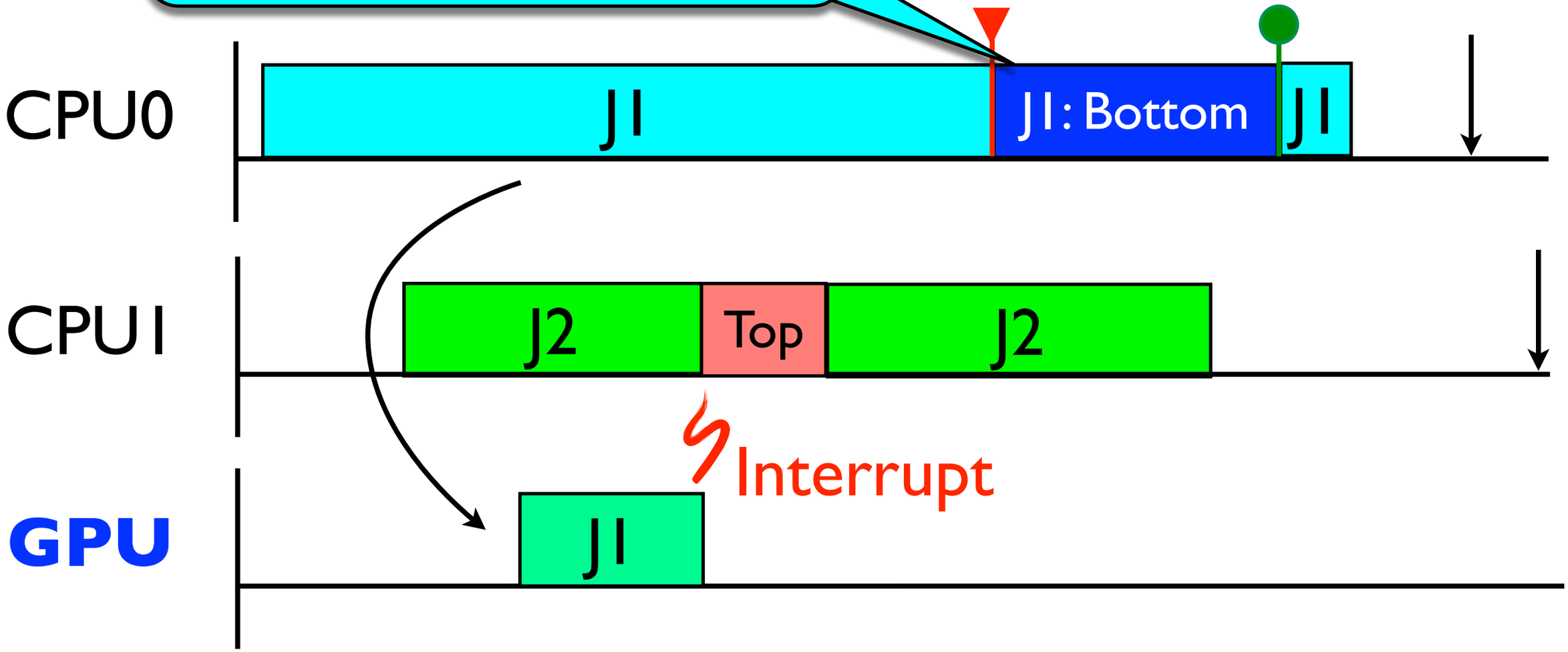


Interrupt Ownership



Interrupt Ownership

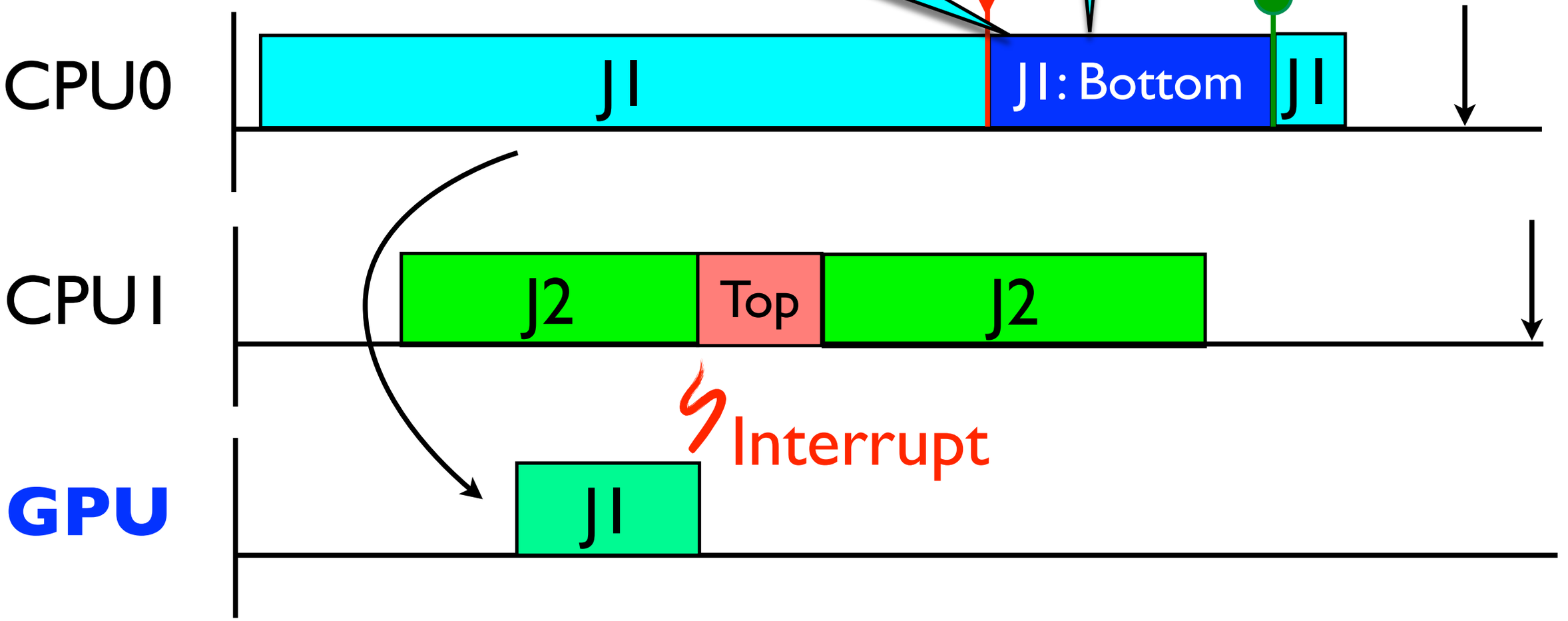
Defer bottom half until J1 suspends, preventing co-scheduling.



Interrupt O

Defer bottom half until J1 suspends, preventing co-scheduling.

Account for bottom half execution time as being of J1.



Globally Scheduled GPU Interrupt Handlers

Globally Scheduled GPU Interrupt Handlers

- Real-Time GPU Interrupts:

Globally Scheduled GPU Interrupt Handlers

- Real-Time GPU Interrupts:

1. Thread bottom halves of GPU interrupts, inheriting priority of owners

Globally Scheduled GPU Interrupt Handlers

- Real-Time GPU Interrupts:

1. Thread bottom halves of GPU interrupts, inheriting priority of owners

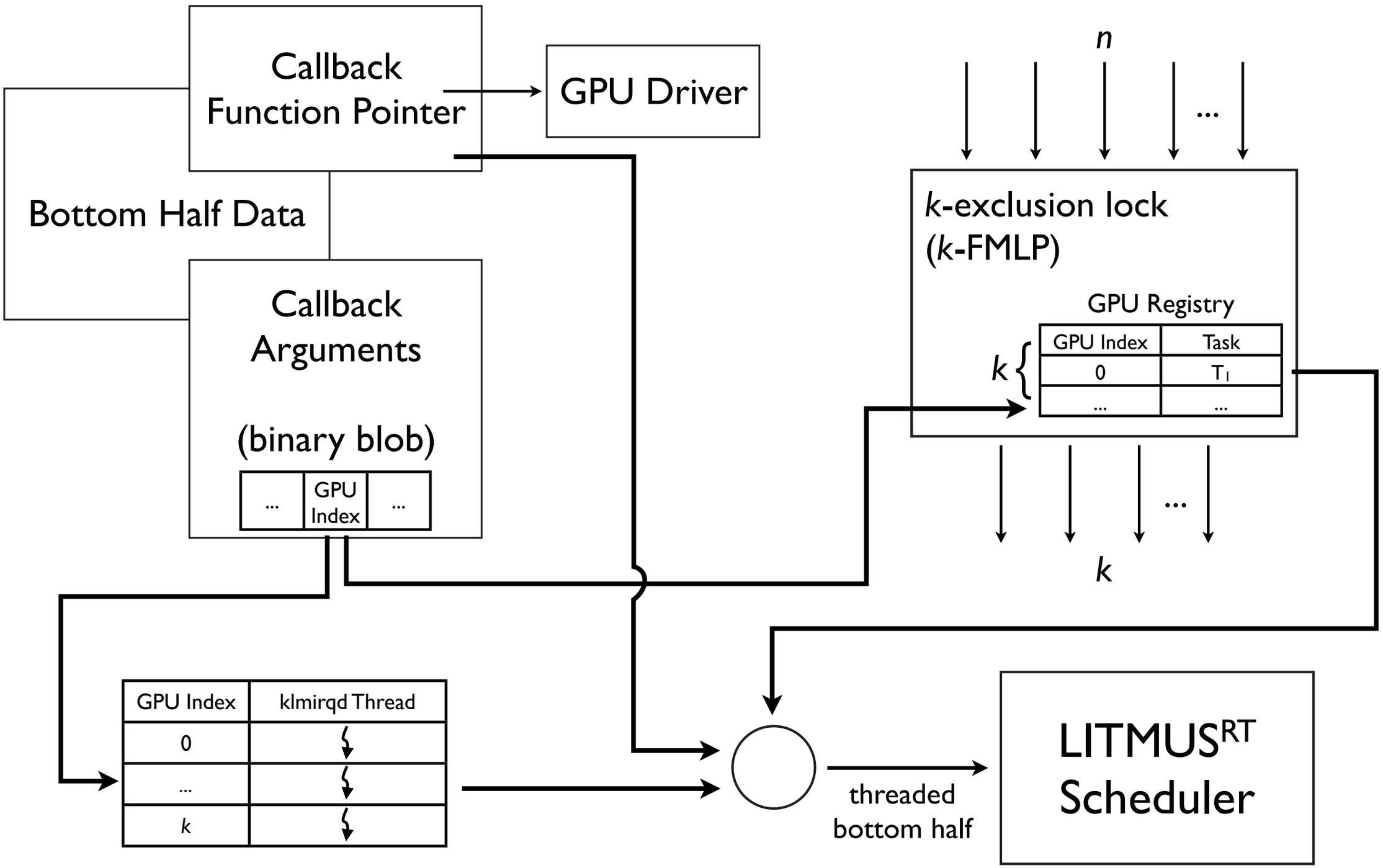
2. Prevent co-scheduling of bottom halves and owners

Globally Scheduled GPU Interrupt Handlers

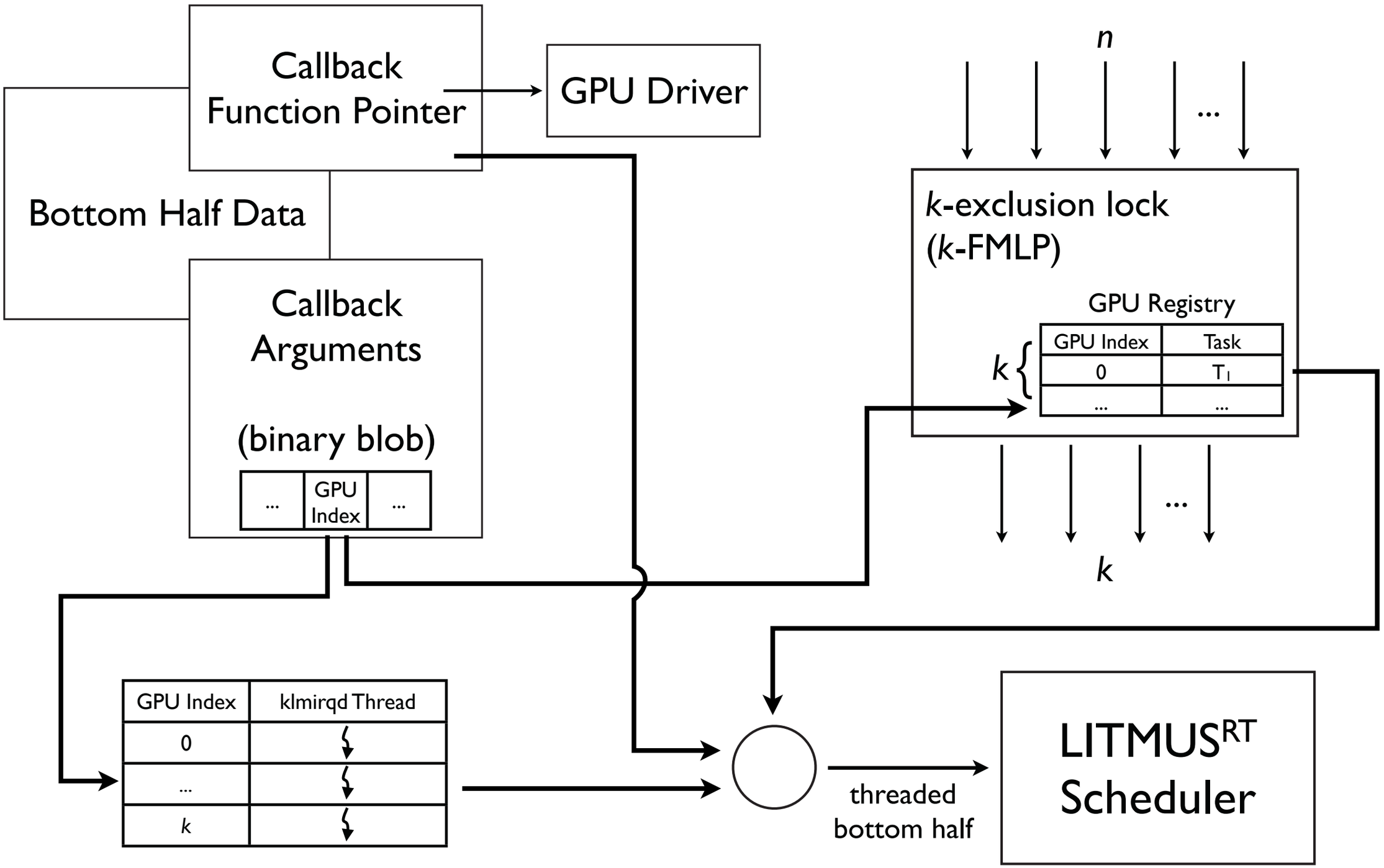
- Real-Time GPU Interrupts:
  1. Thread bottom halves of GPU interrupts, inheriting priority of owners
  2. Prevent co-scheduling of bottom halves and owners
- **PROBLEM: GPU driver is closed source.**
  - Which GPU raised the interrupt?
  - What is the priority of the bottom half?

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

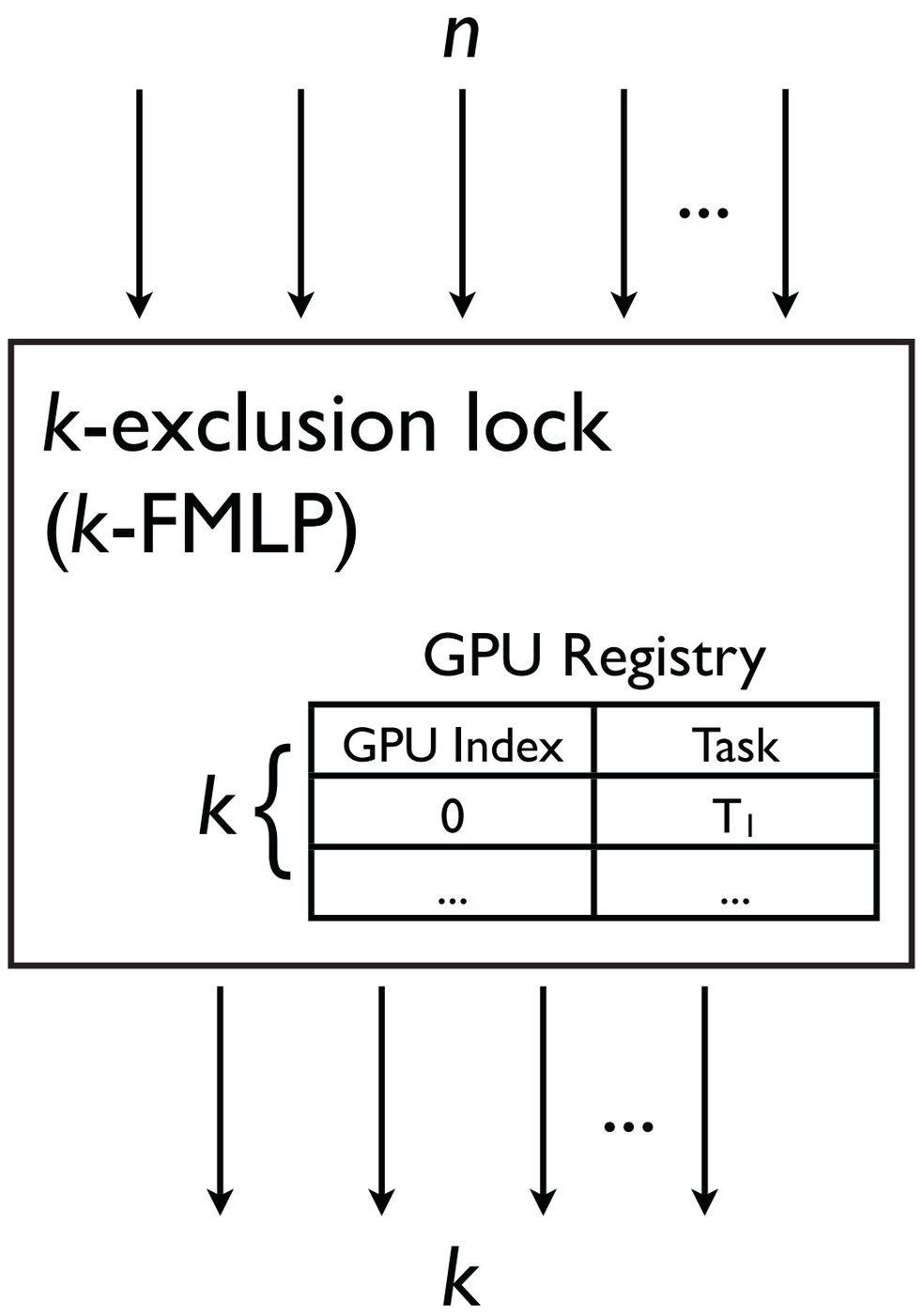
GPU Interrupt Handling



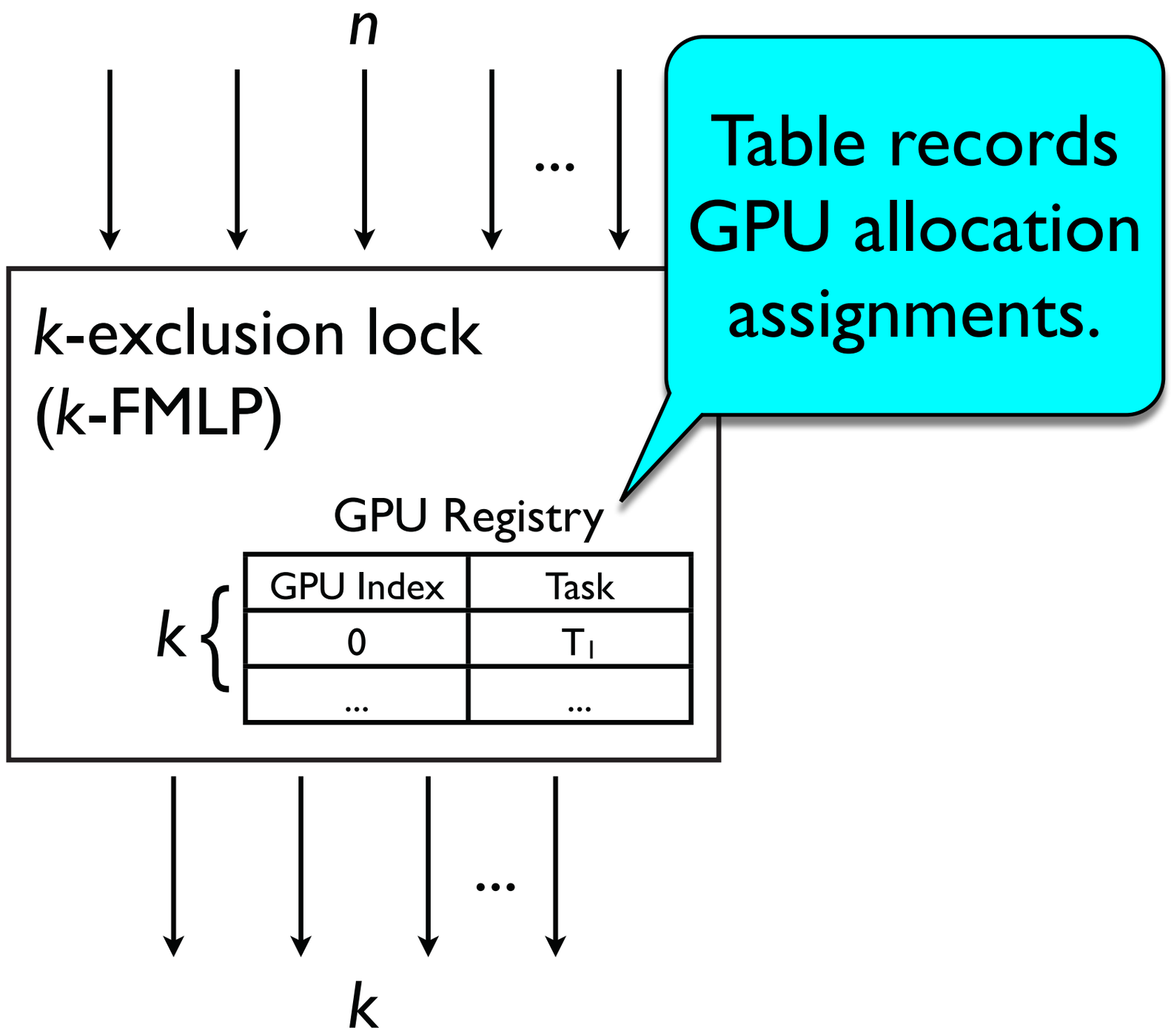
GPU Allocation



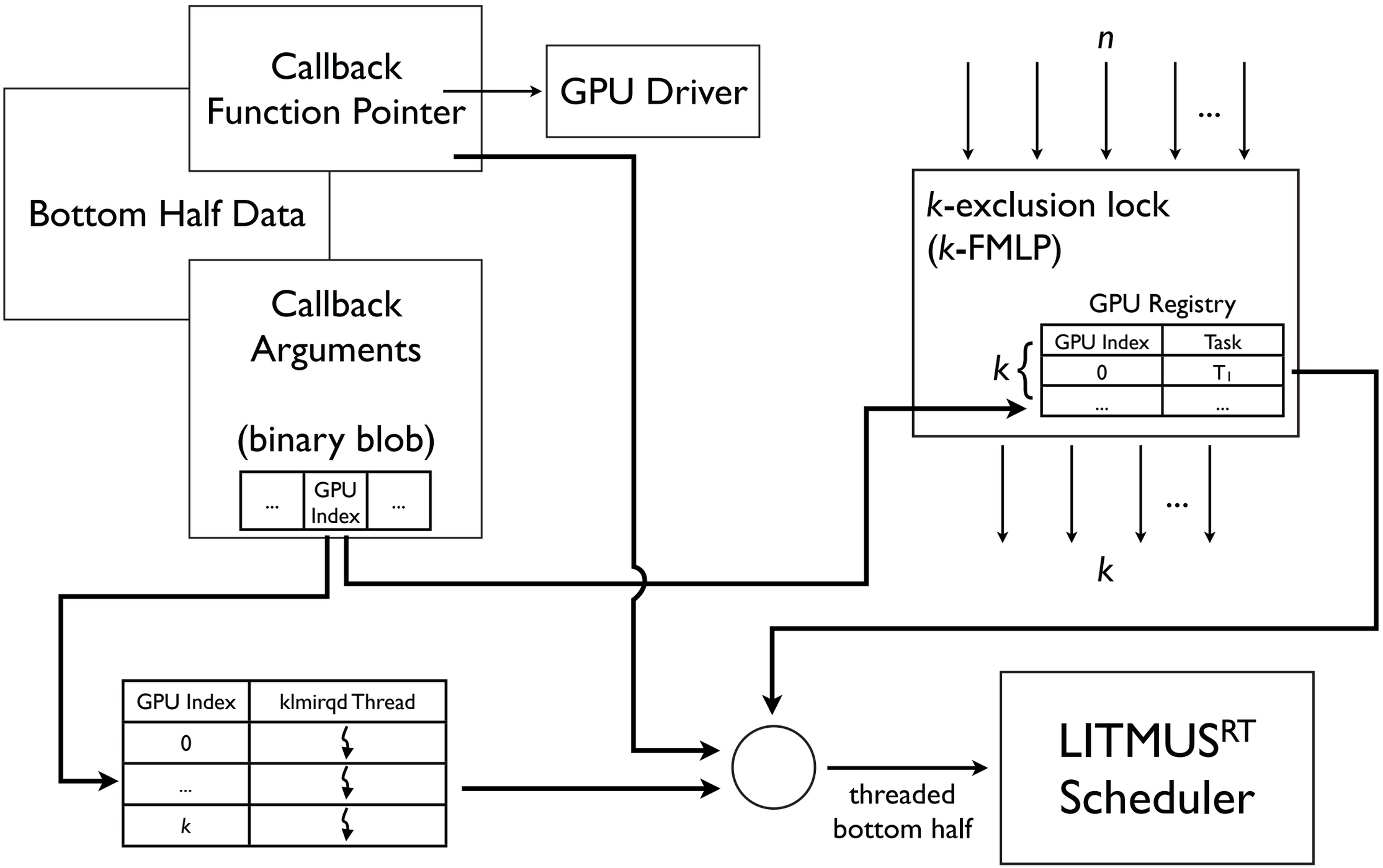
GPU Allocation



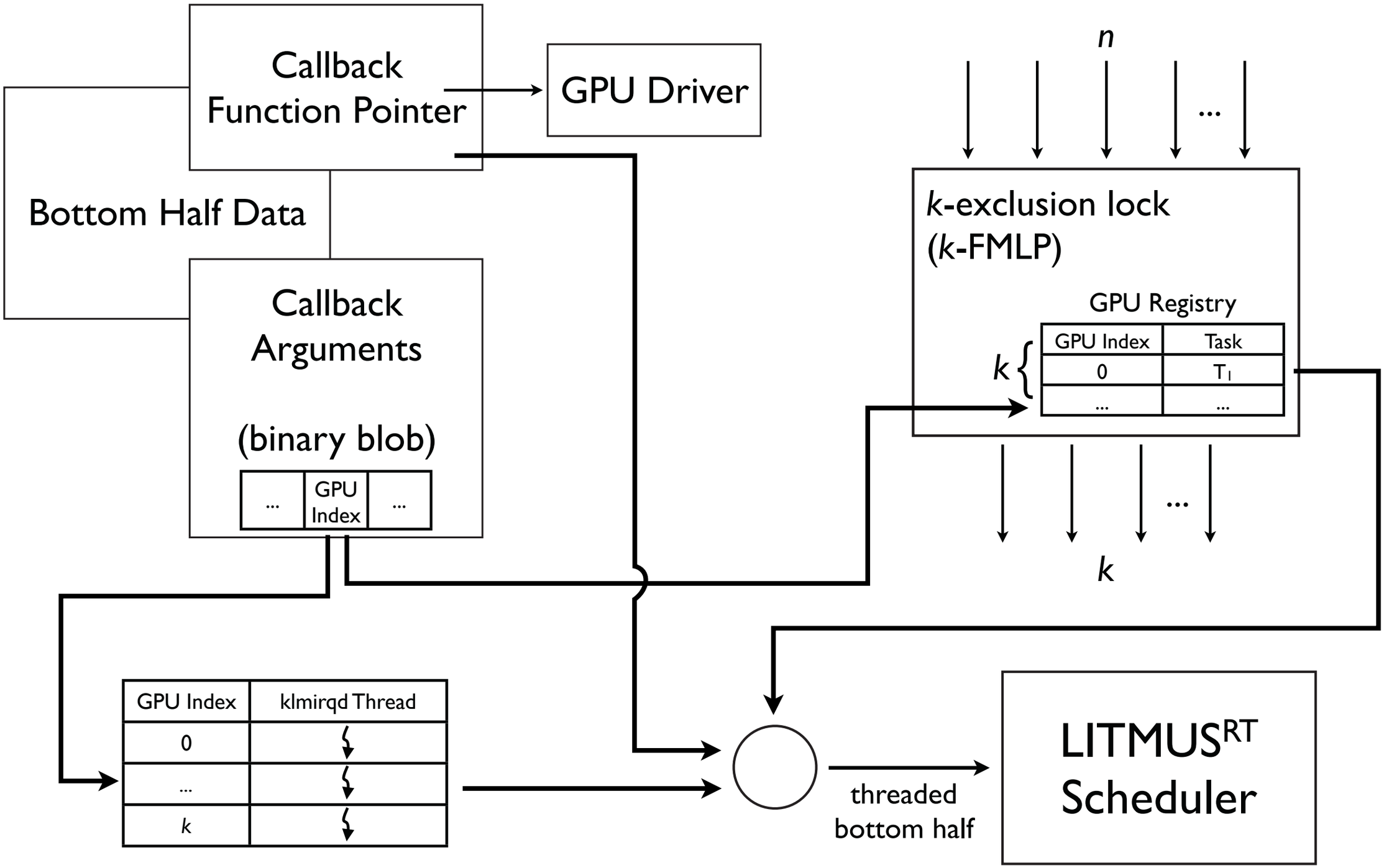
GPU Allocation



GPU Allocation



klmirqd GPU Threads



klmirqd GPU Threads

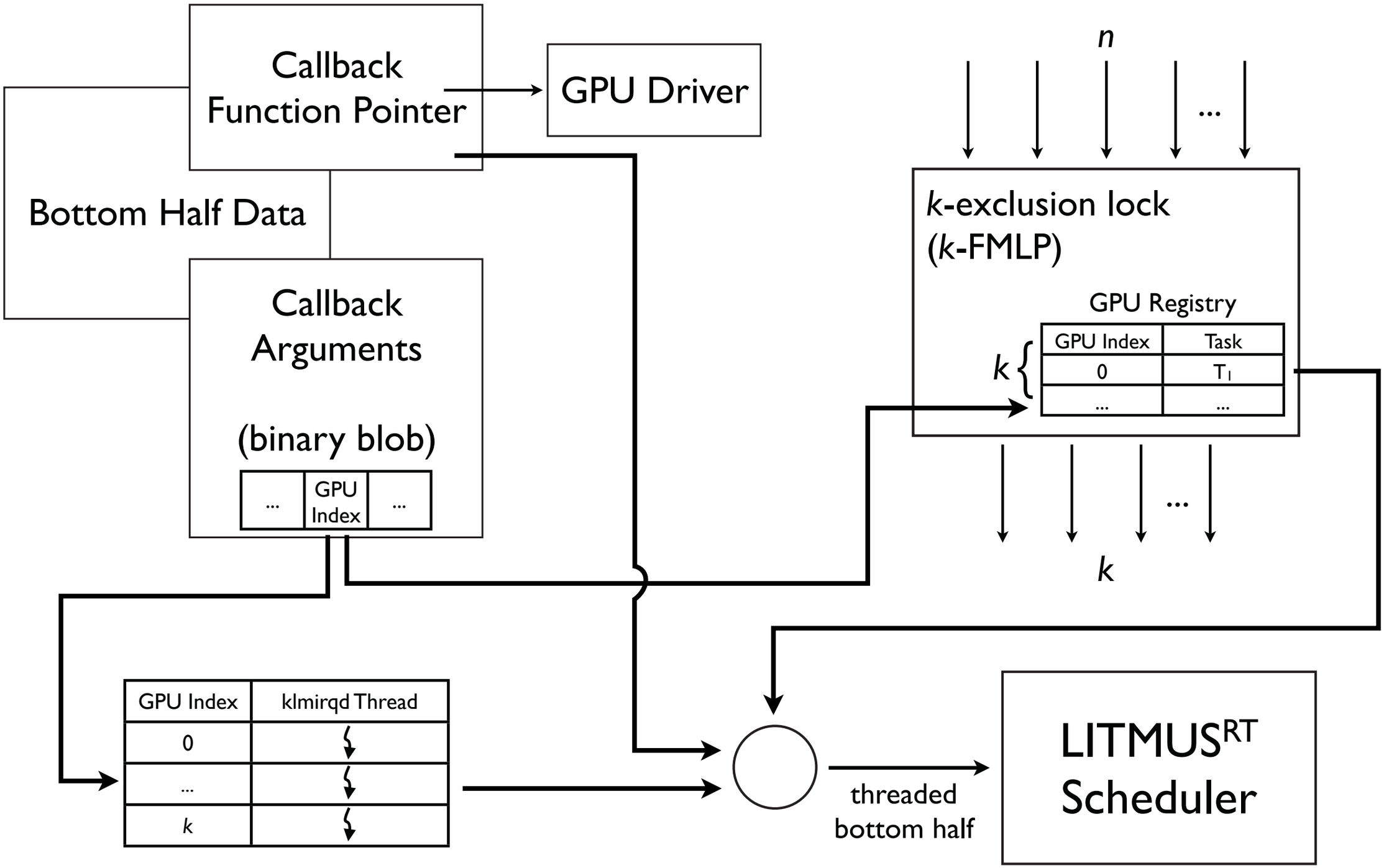
GPU Index	klmirqd Thread
0	
...	
k	

klmirqd GPU Th

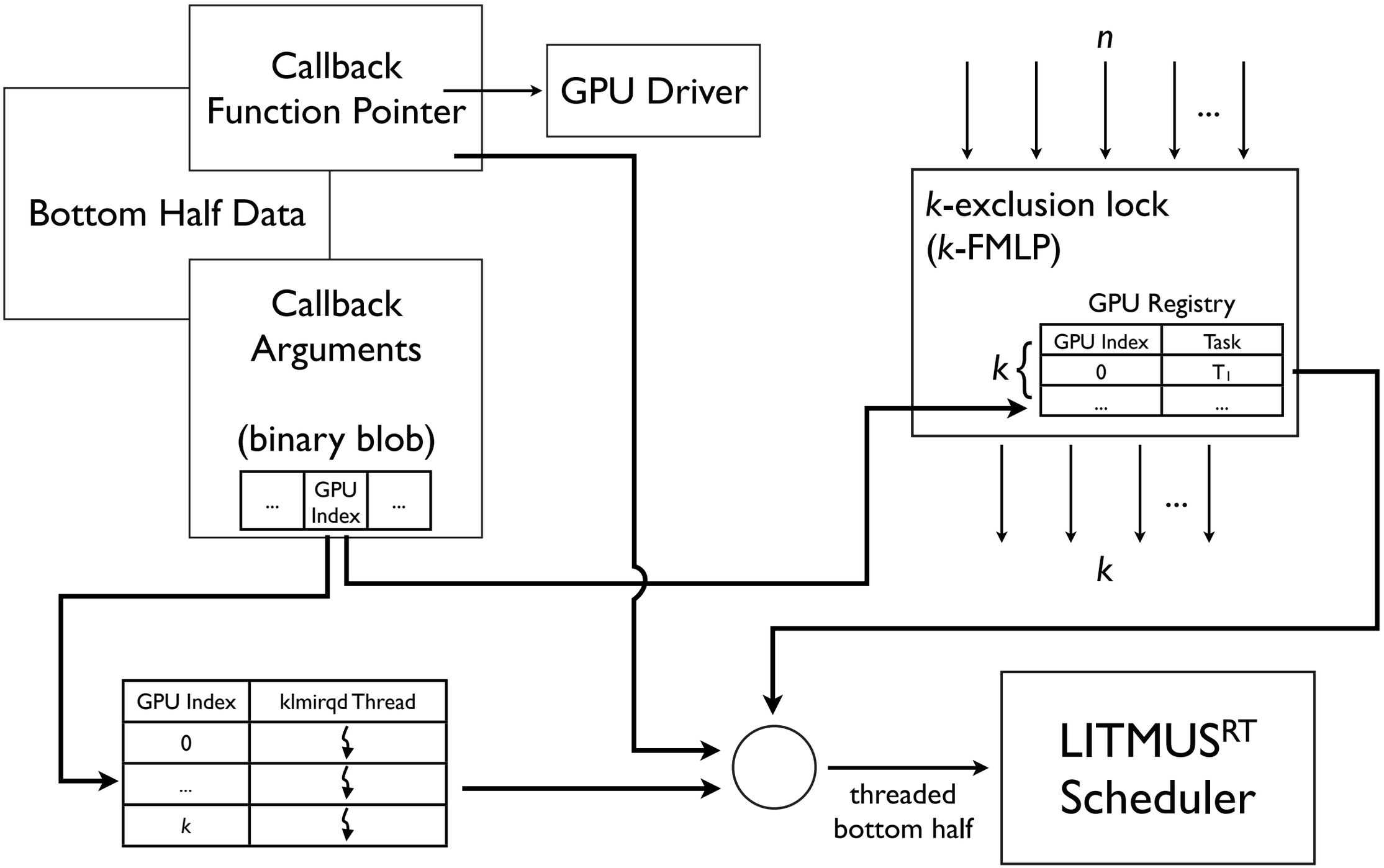
One klmirqd thread per GPU.

GPU Index	klmirqd Thread
0	
...	
k	

klmirqd GPU Threads

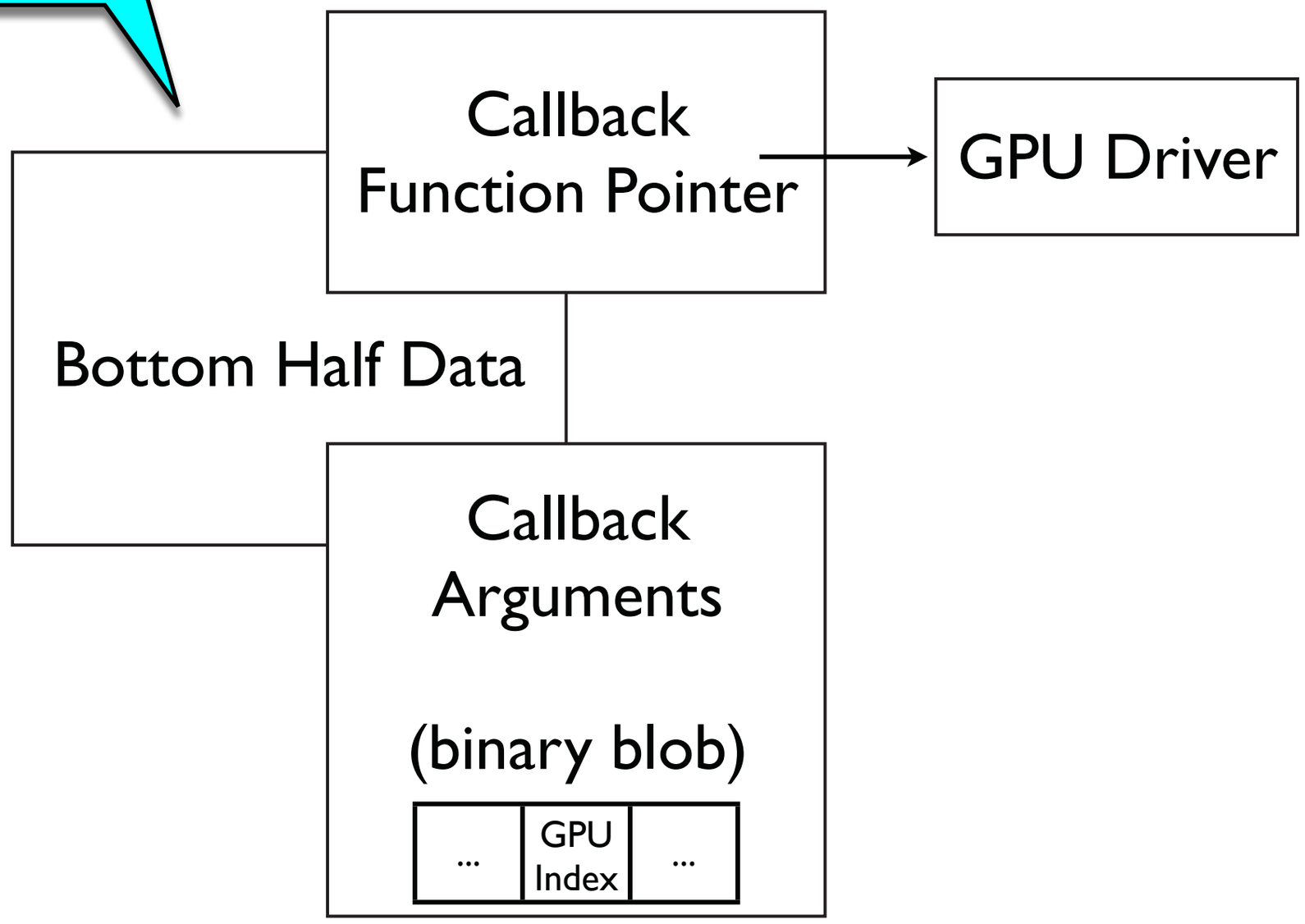


Interrupt Interception

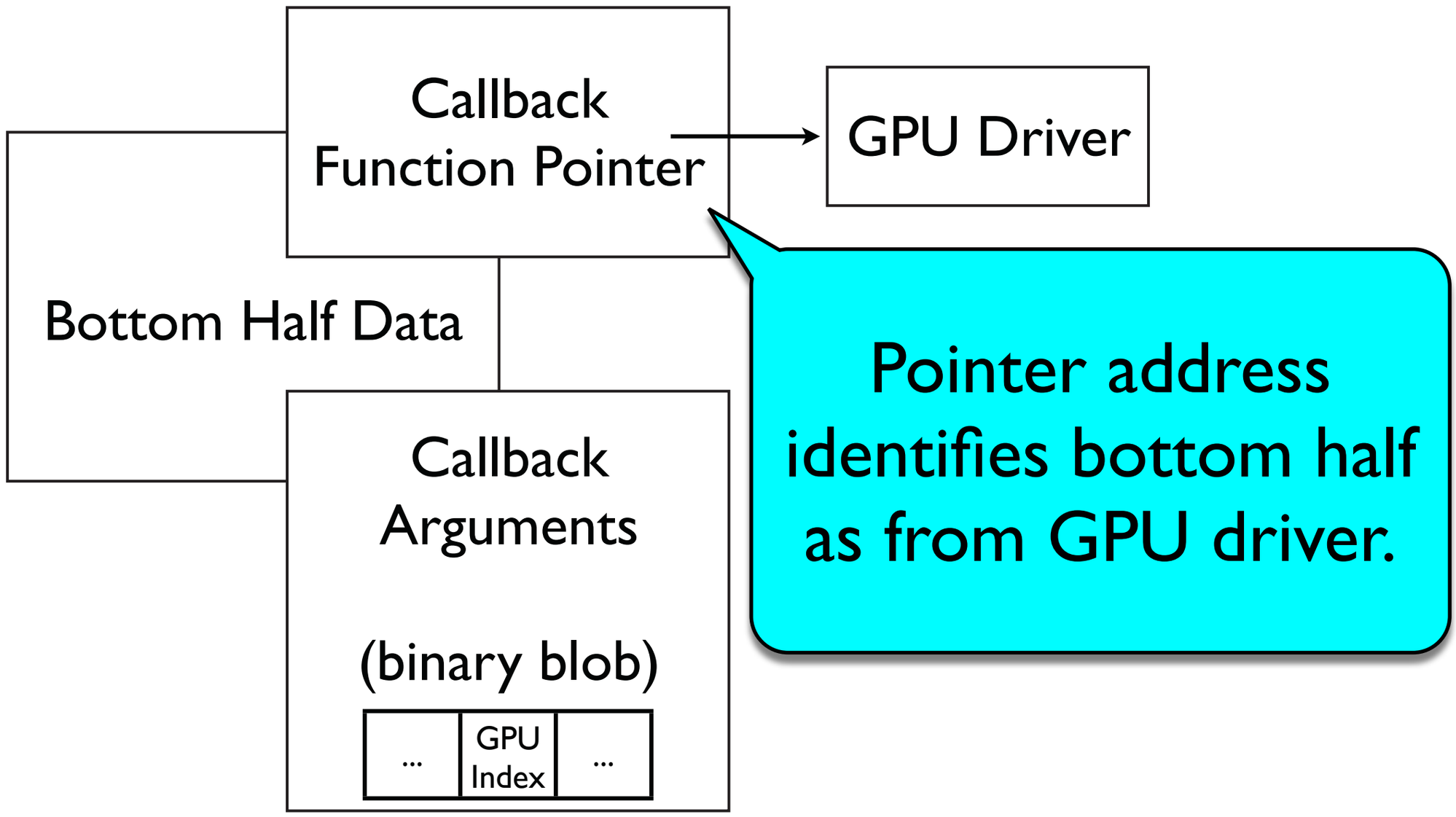


terrupt Interception

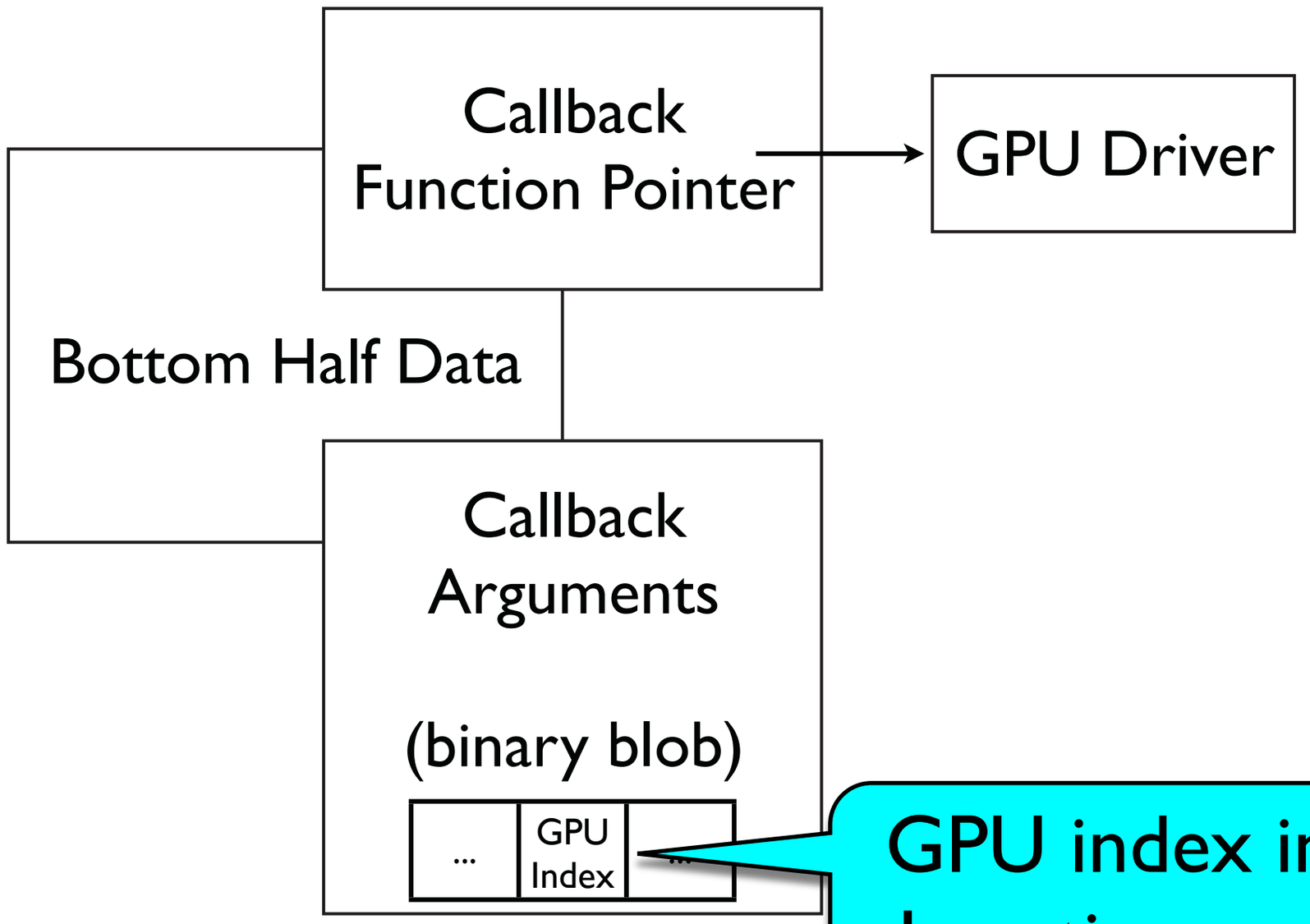
Spawned by top half.



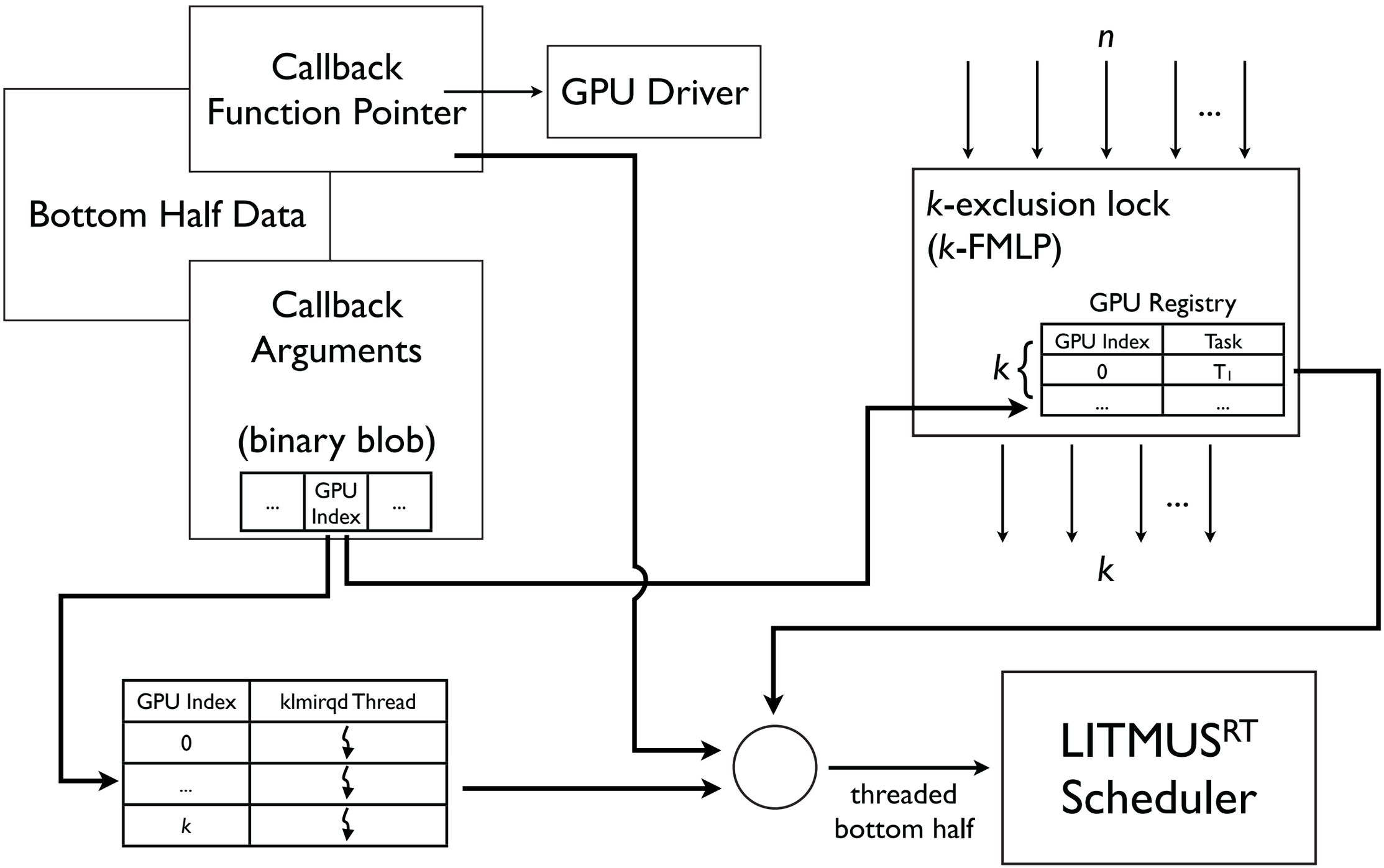
Interrupt Interception



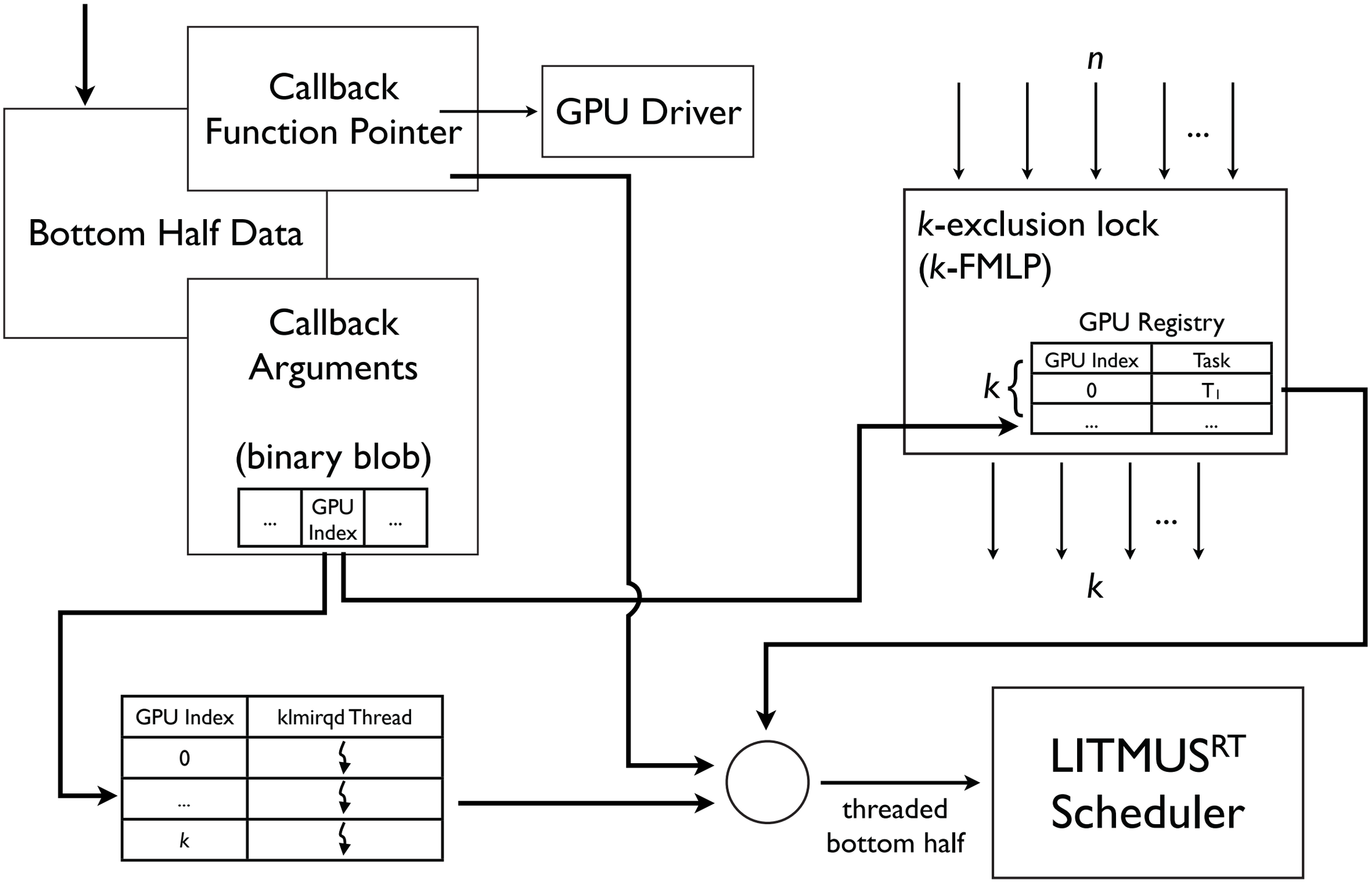
Interrupt Interception



Interrupt Interception



GPU Interrupt Handling



rrupt Handling

GPU bottom half arrives.

Callback
Function Pointer

GPU Driver

Bottom Half Data

Callback
Arguments

(binary blob)

...	GPU Index	...

n

k-exclusion lock
(*k*-FMLP)

GPU Registry

GPU Index	Task
0	T_1
...	...

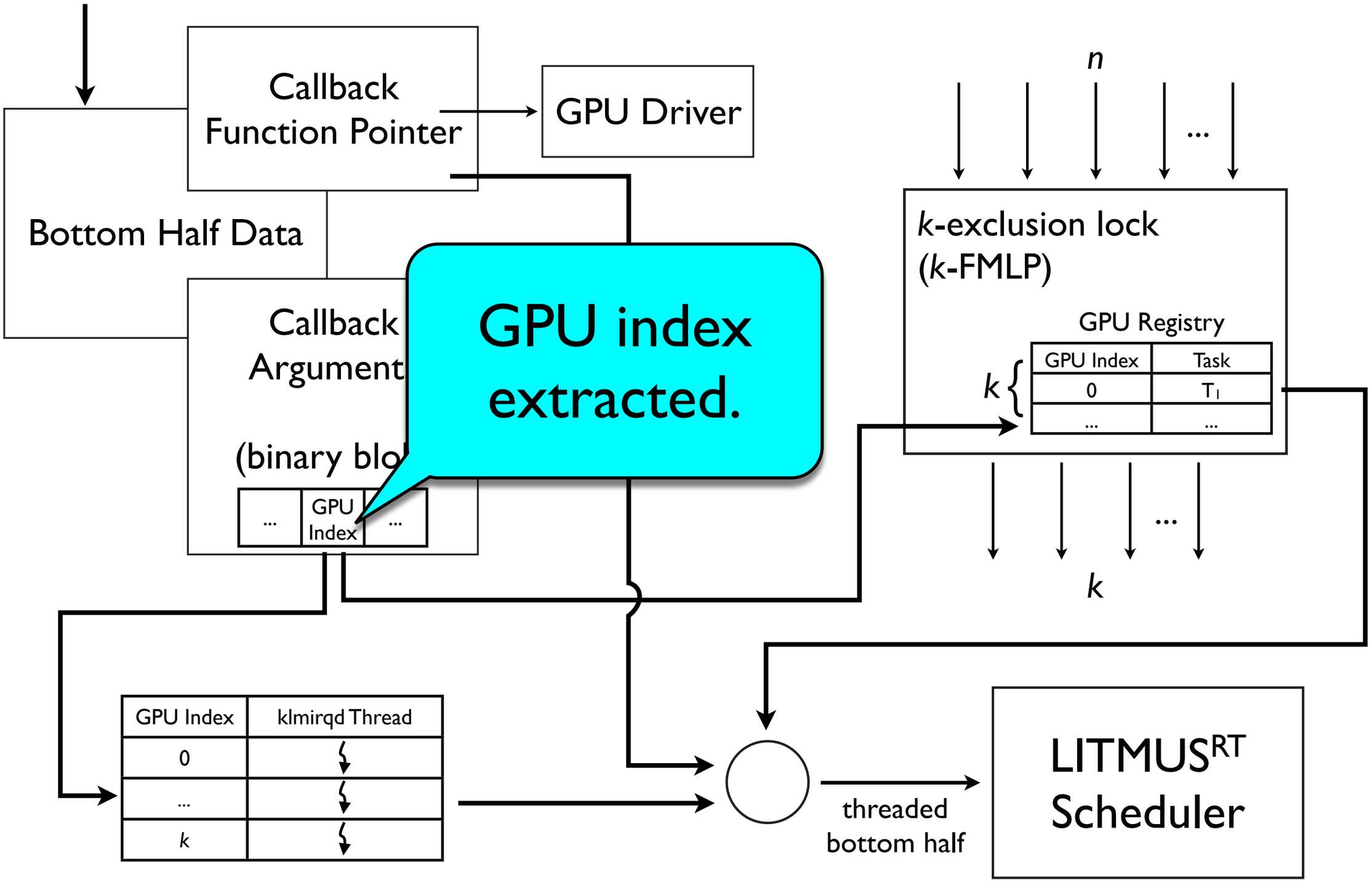
k

GPU Index	klmirqd Thread
0	
...	
k	

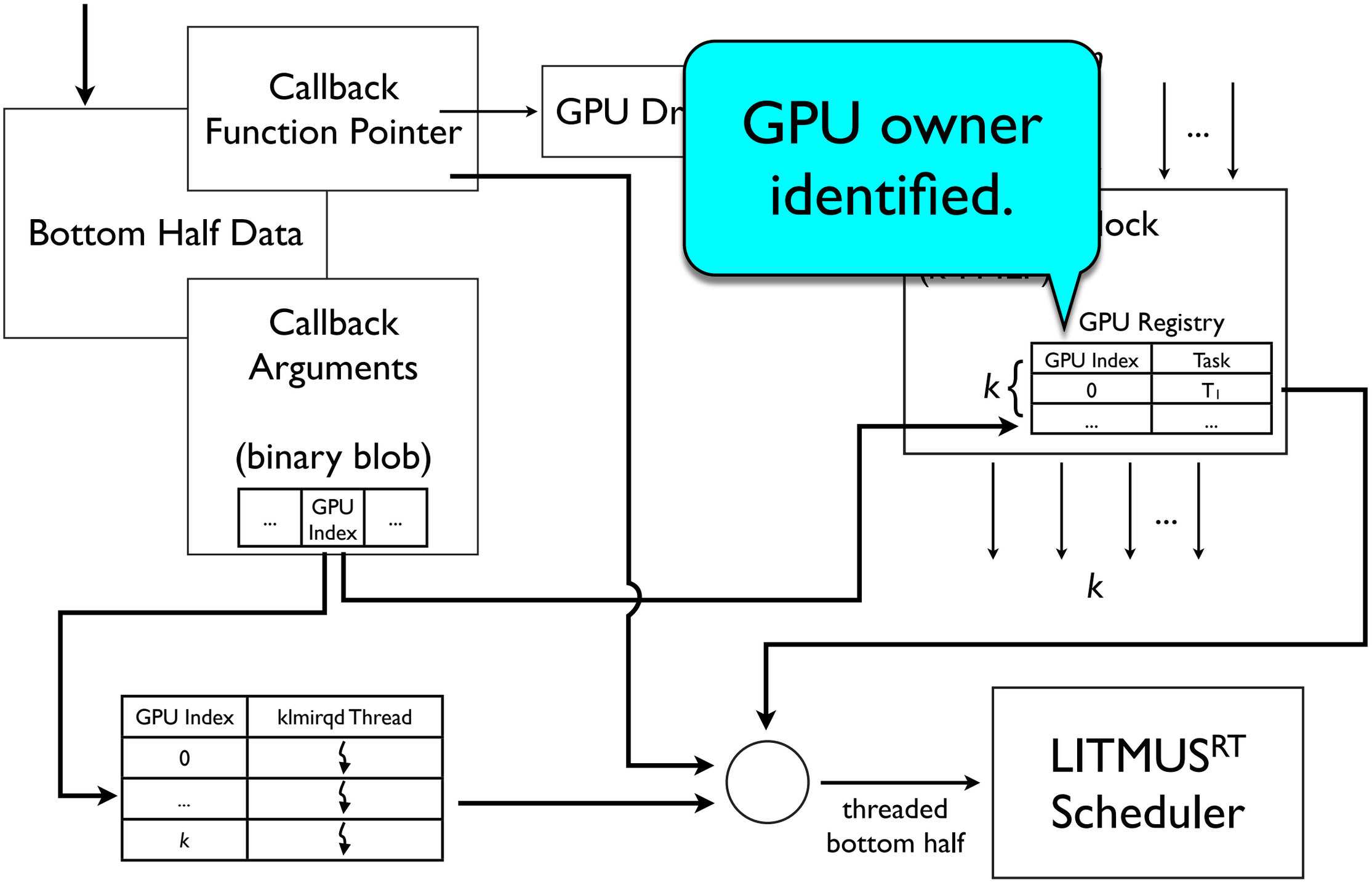
threaded
bottom half

LITMUS^{RT}
Scheduler

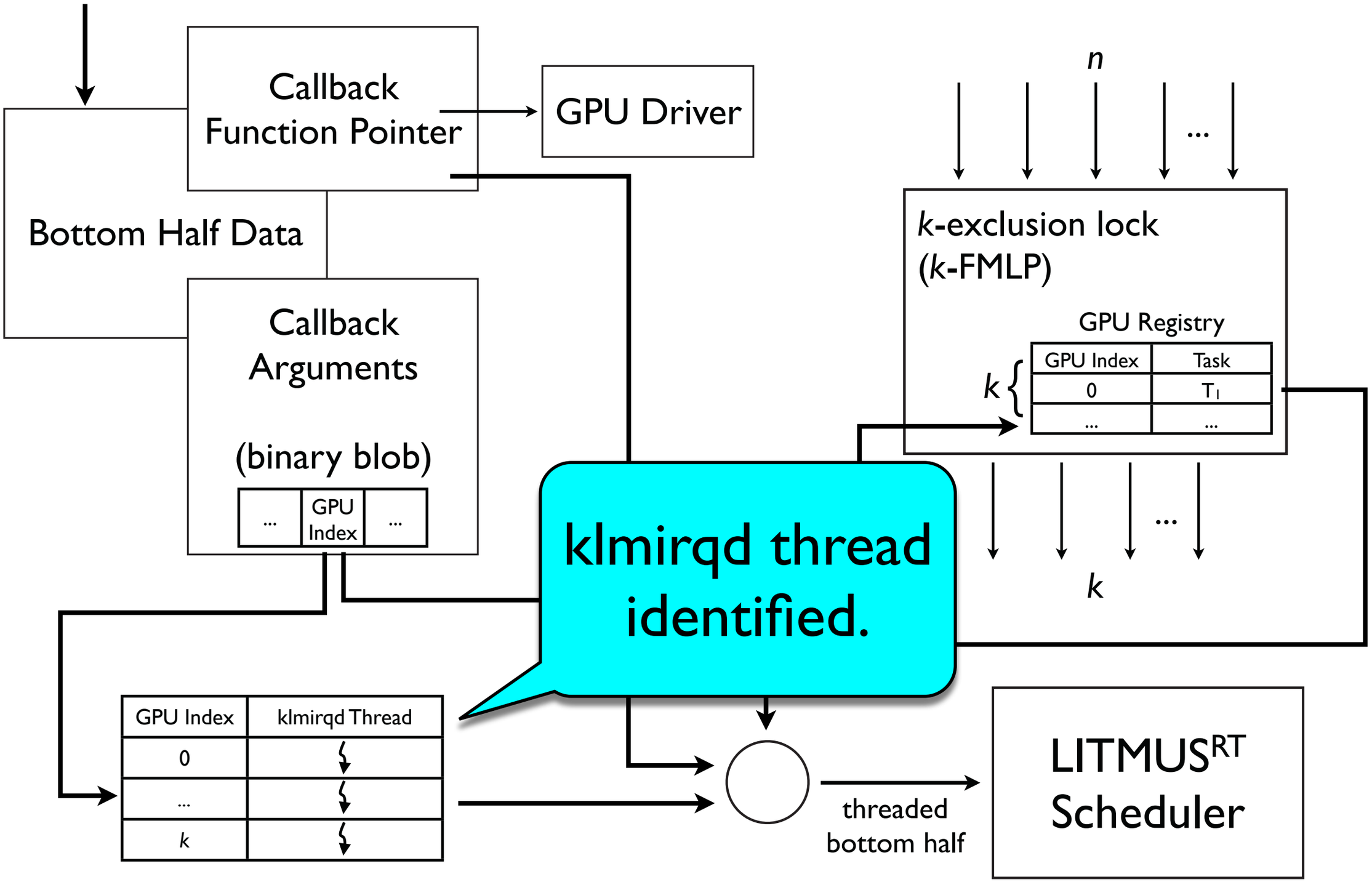
GPU Interrupt Handling



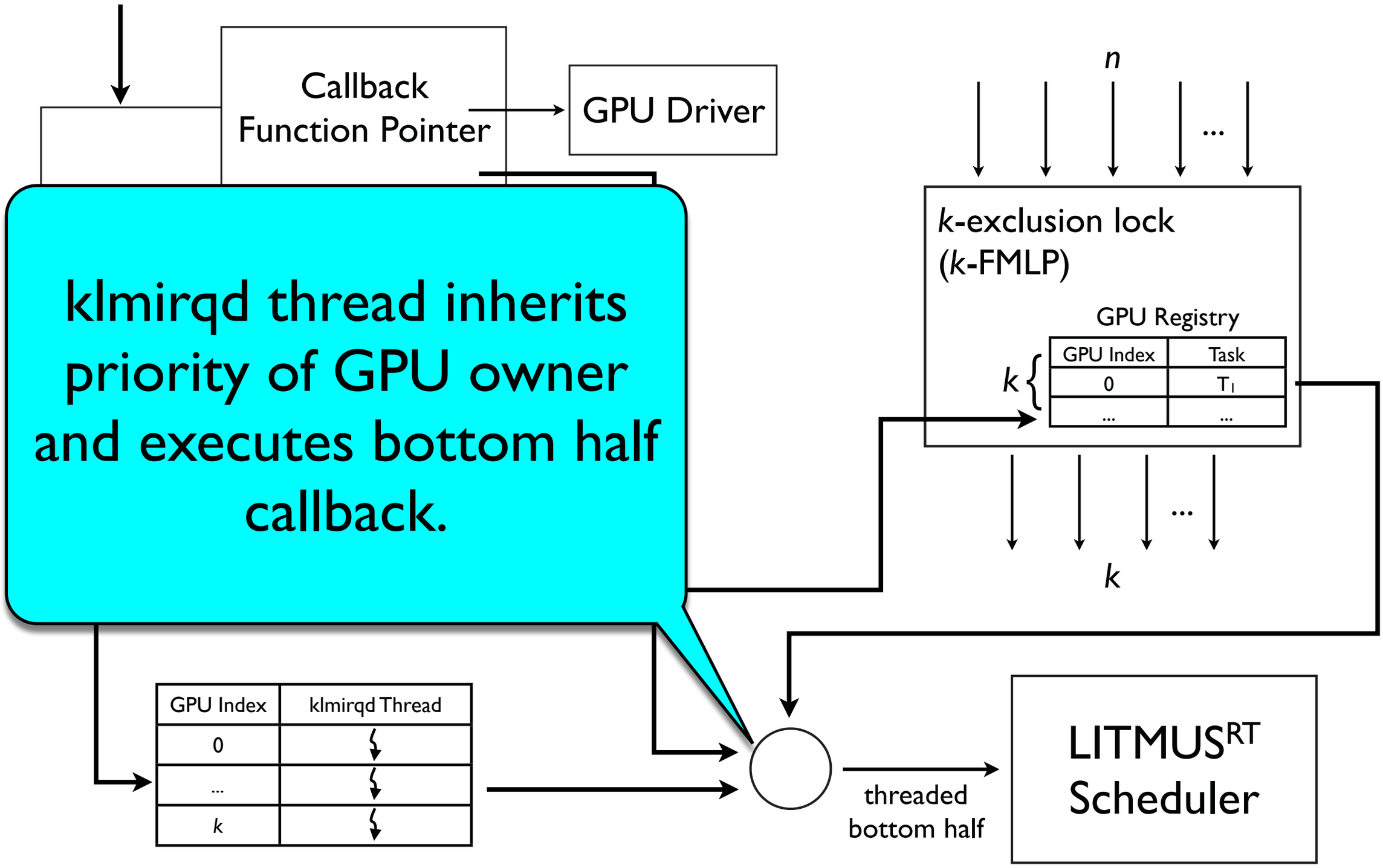
GPU Interrupt Handling



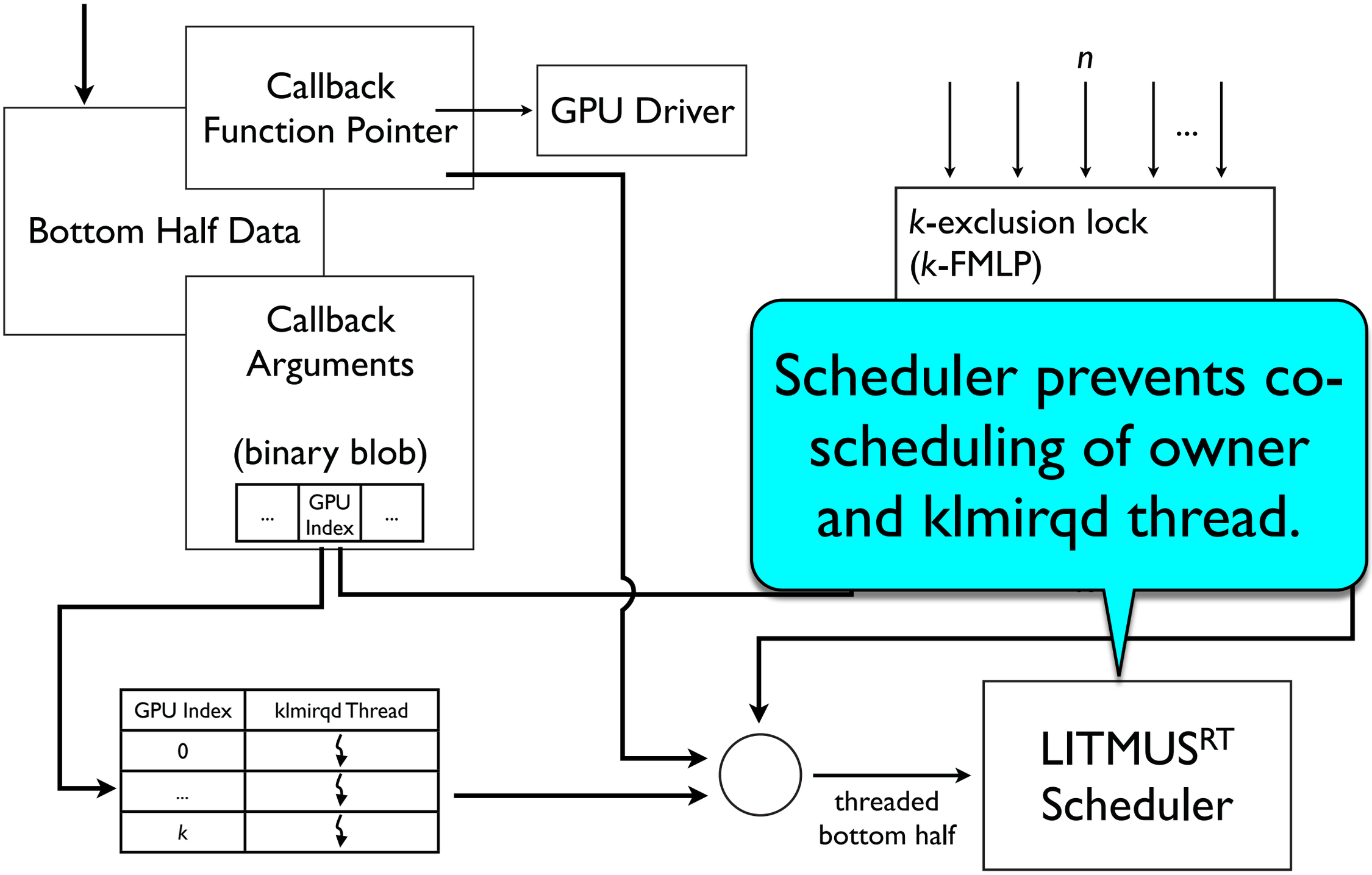
GPU Interrupt Handling



GPU Interrupt Handling



GPU Interrupt Handling



Evaluation

Evaluation

- Test platform:

Evaluation

- Test platform:
  - Two six-core Xeon X5060 processors at 2.67GHz

Evaluation

- Test platform:
  - Two six-core Xeon X5060 processors at 2.67GHz
  - Eight NVIDIA GTX-470 GPUs

Evaluation

- Test platform:
  - Two six-core Xeon X5060 processors at 2.67GHz
  - Eight NVIDIA GTX-470 GPUs
  - Scheduled in clusters along NUMA boundaries

Evaluation

- Test platform:
  - Two six-core Xeon X5060 processors at 2.67GHz
  - Eight NVIDIA GTX-470 GPUs
  - Scheduled in clusters along NUMA boundaries
    - One X5060 and four GPUs per cluster

Effect on Priority Inversions

Effect on Priority Inversions

- Inversions measured in LITMUS^{RT} for:

Effect on Priority Inversions

- Inversions measured in LITMUS^{RT} for:
  - klmirqd

Effect on Priority Inversions

- Inversions measured in LITMUS^{RT} for:
  - klmirqd
  - Standard Linux interrupt handling (SLIH)

Effect on Priority Inversions

- Inversions measured in LITMUS^{RT} for:
  - klmirqd
  - Standard Linux interrupt handling (SLIH)
  - Modified process-aware interrupt (PAI) handling for global scheduling (adapted from Zhang and West, RTSS 2006)

Effect on Priority Inversions

- Inversions measured in LITMUS^{RT} for:
  - klmirqd
  - Standard Linux interrupt handling (SLIH)
  - Modified process-aware interrupt (PAI) handling for global scheduling (adapted from Zhang and West, RTSS 2006)
- Executed 41 task sets with utilizations [7.5, 11.5] each for 2 minutes

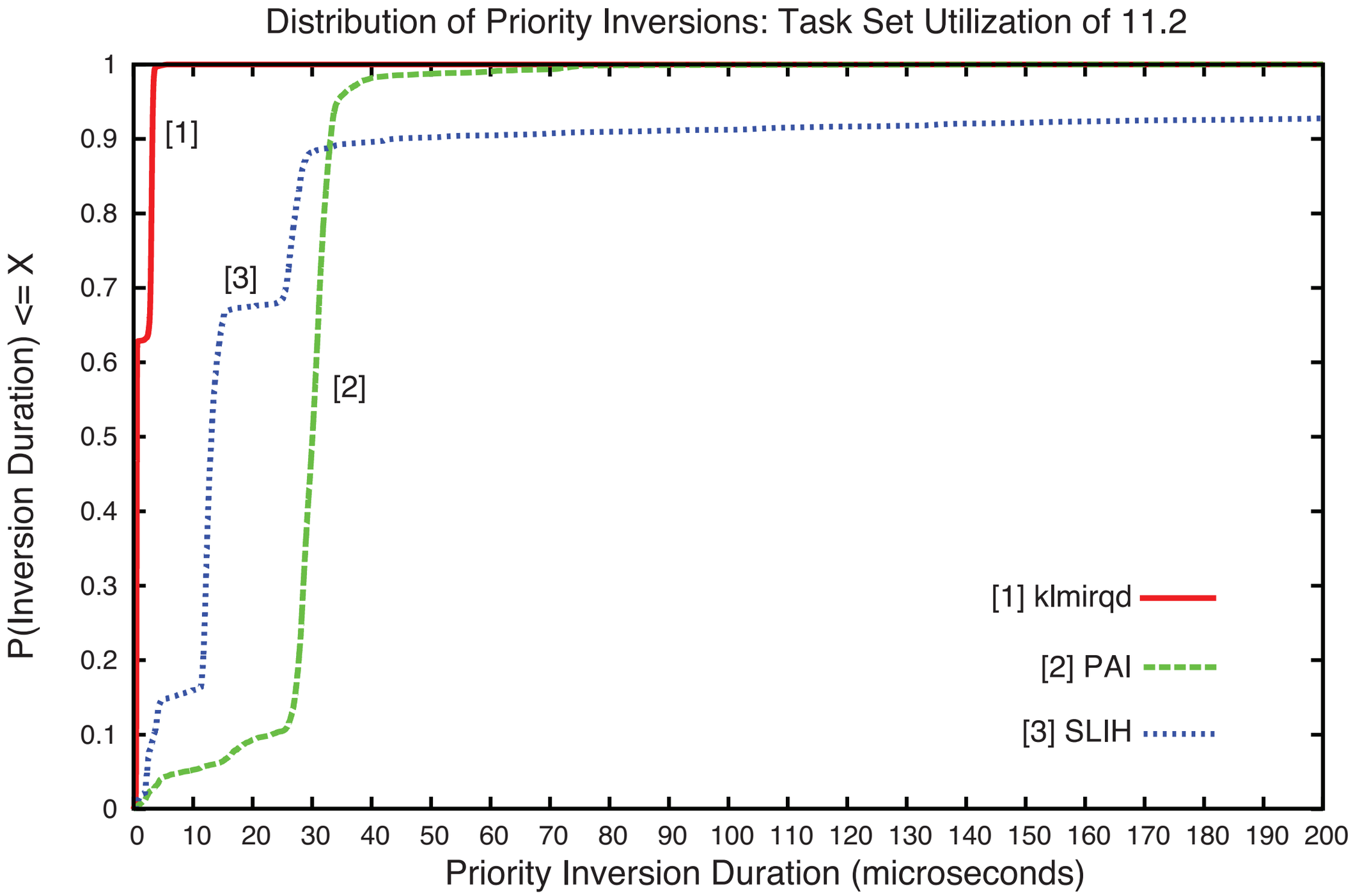
Effect on Priority Inversions

- Inversions measured in LITMUS^{RT} for:
  - klmirqd
  - Standard Linux interrupt handling (SLIH)
  - Modified process-aware interrupt (PAI) handling for global scheduling (adapted from Zhang and West, RTSS 2006)
- Executed 41 task sets with utilizations [7.5, 11.5] each for 2 minutes
- Every task used GPUs asynchronously

Effect on Priority Inversions

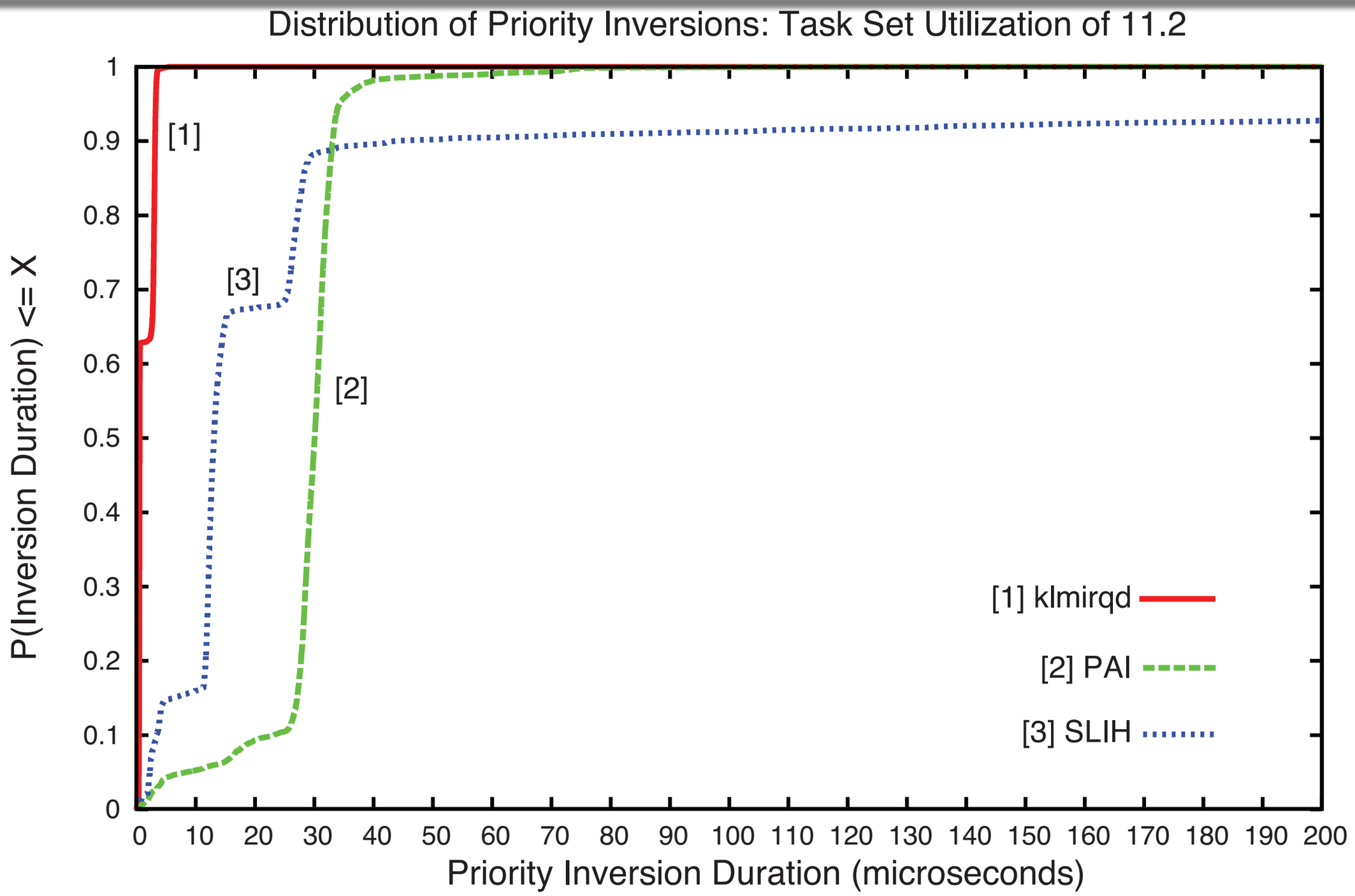
- Inversions measured in LITMUS^{RT} for:
  - klmirqd
  - Standard Linux interrupt handling (SLIH)
  - Modified process-aware interrupt (PAI) handling for global scheduling (adapted from Zhang and West, RTSS 2006)
- Executed 41 task sets with utilizations [7.5, 11.5] each for 2 minutes
- Every task used GPUs asynchronously
- Scheduled under Clustered EDF

Effect on Priority Inversions



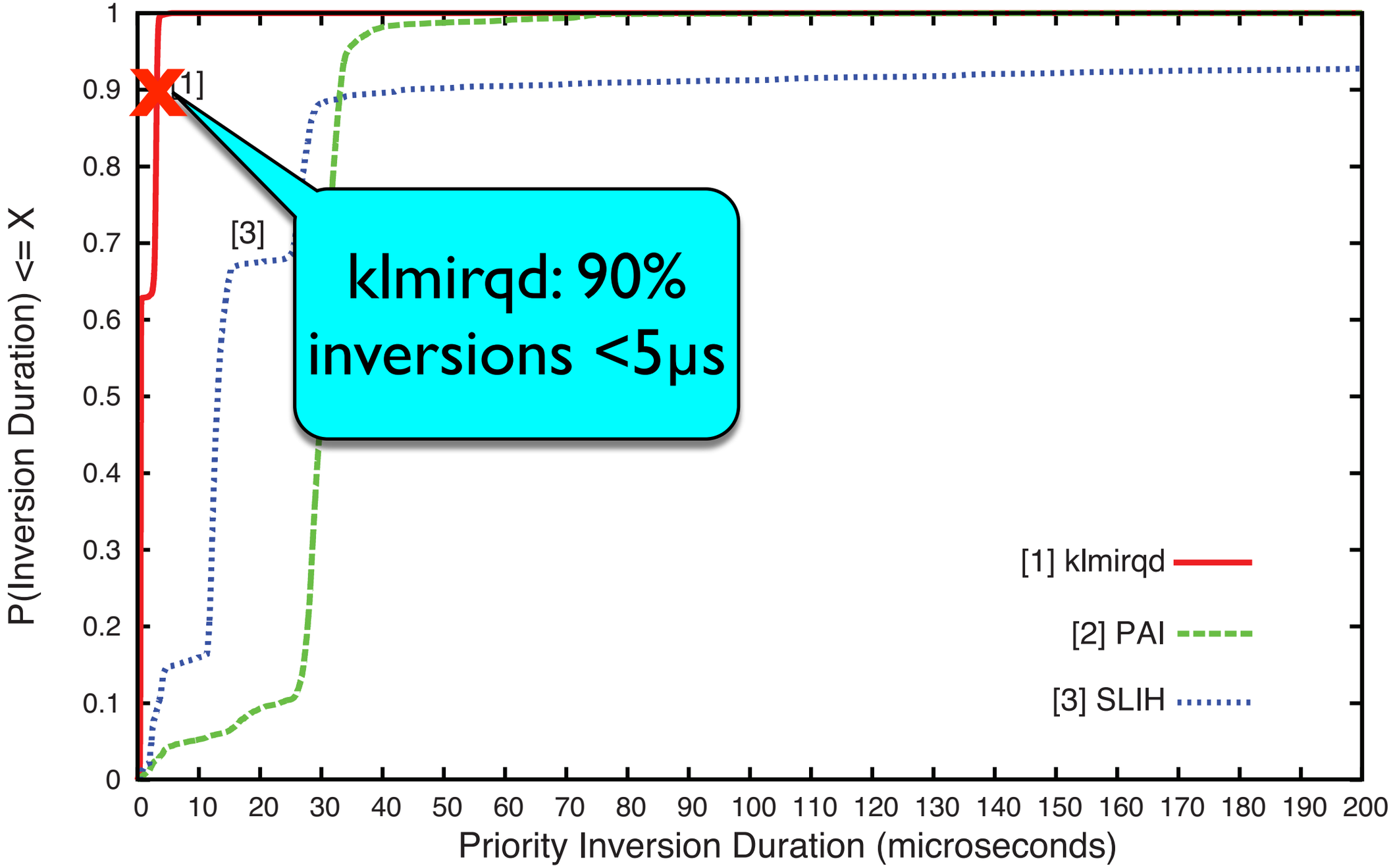
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Priority inversion durations decreased.



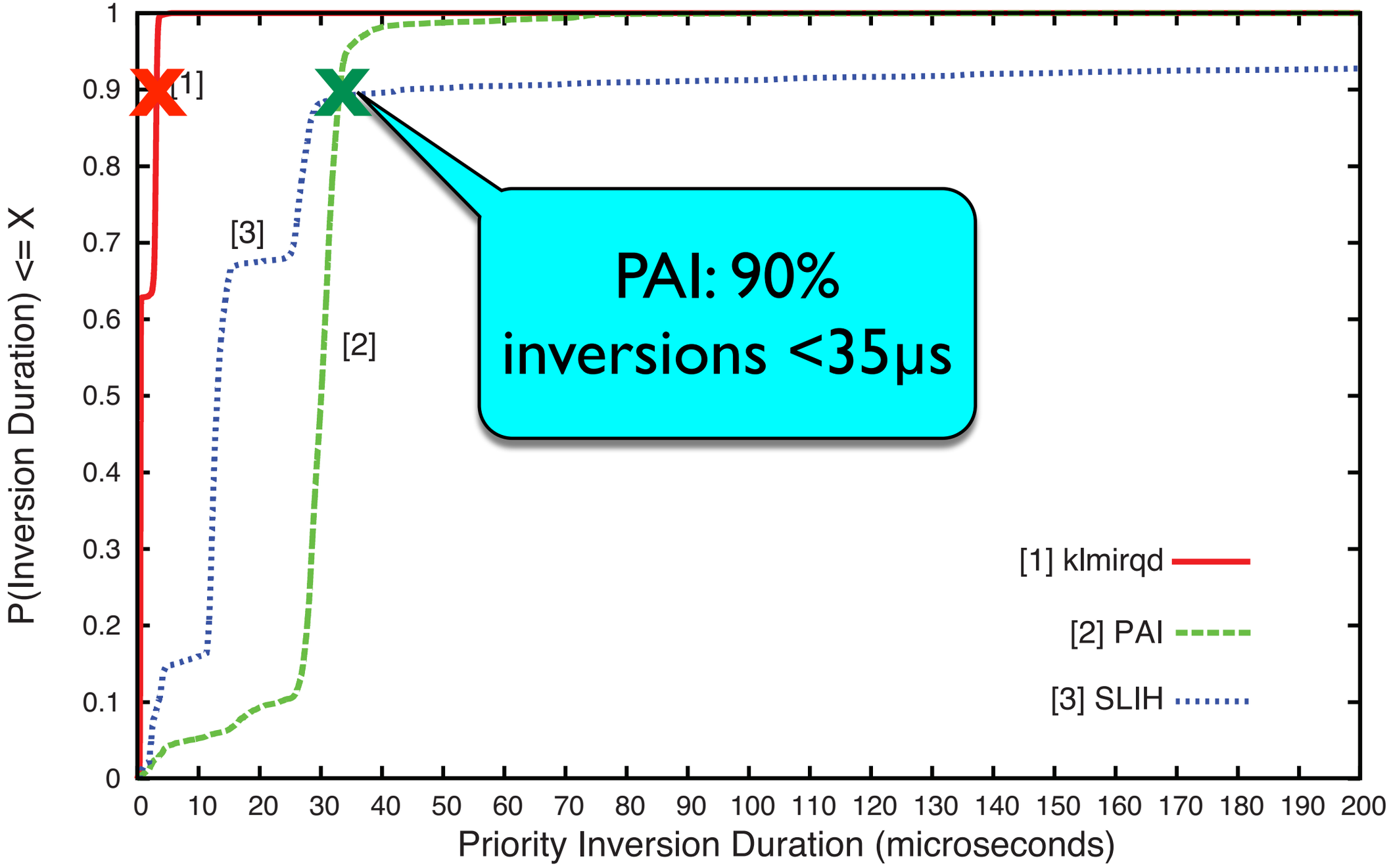
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Priority inversion durations decreased.



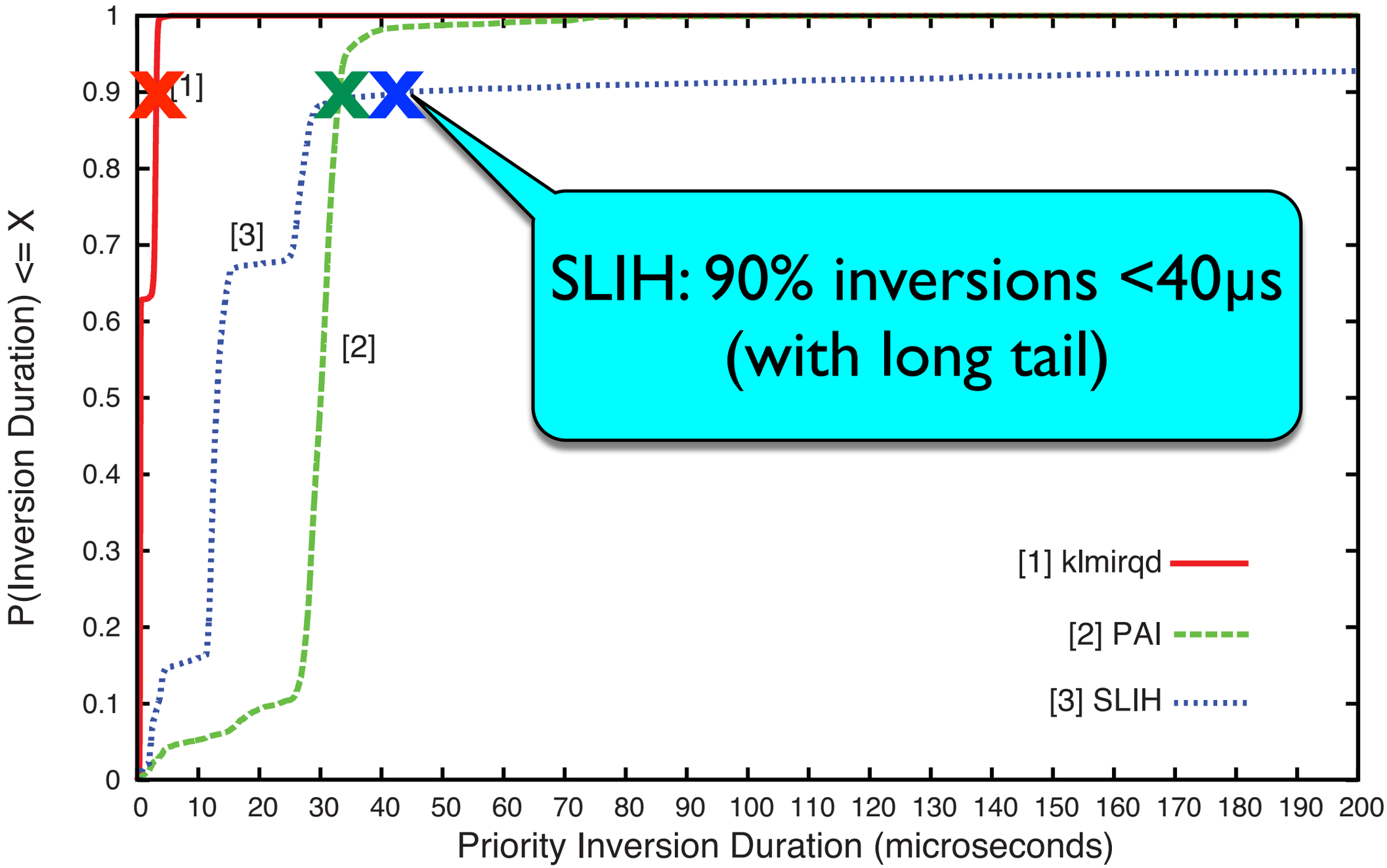
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Priority inversion durations decreased.



THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Priority inversion durations decreased.



SLIH: 90% inversions <40µs (with long tail)

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Overhead-Aware Schedulability Experiments

Overhead-Aware Schedulability Experiments

- Gathered overhead measurements for many system tasks (such as scheduling)

Overhead-Aware Schedulability Experiments

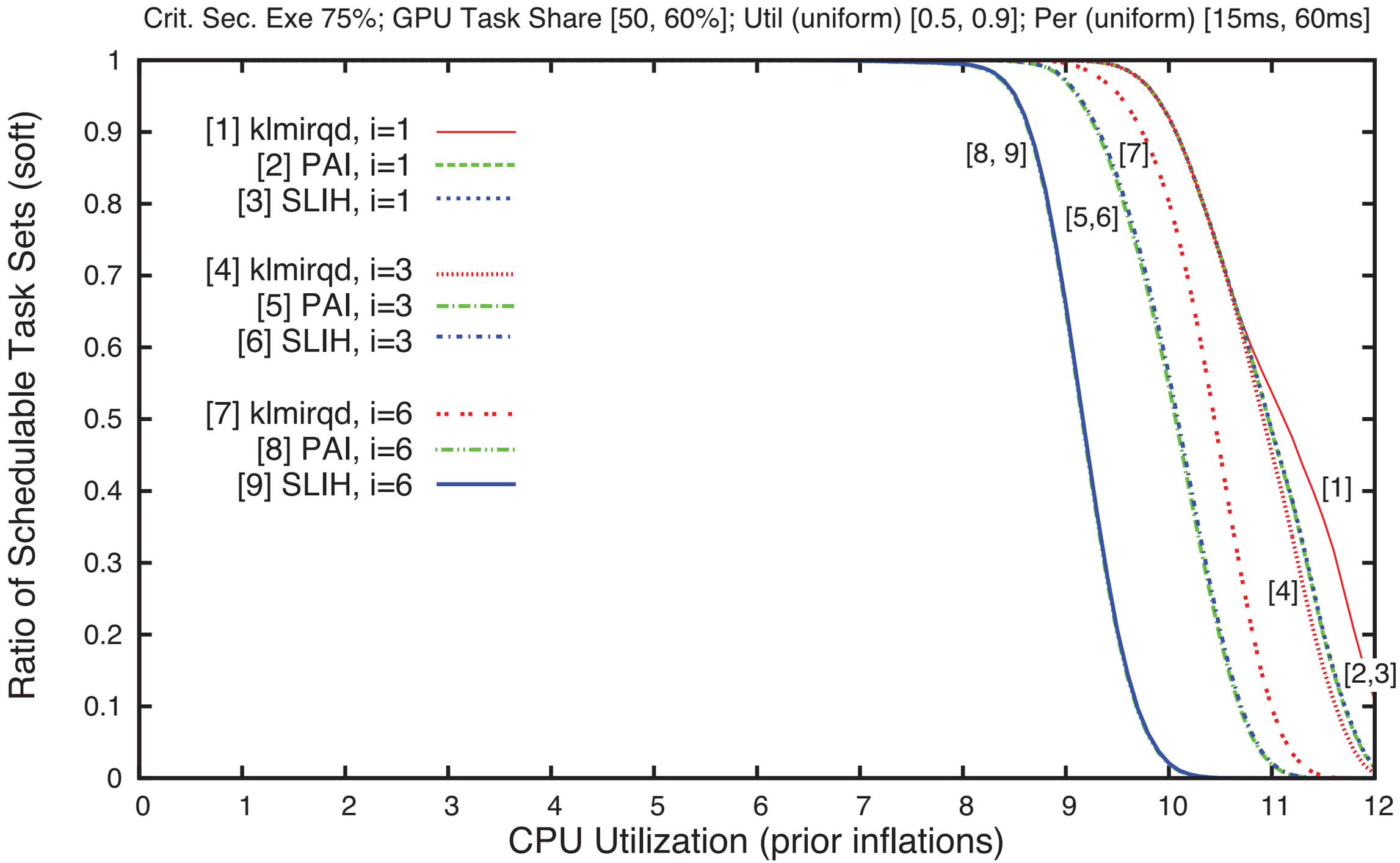
- Gathered overhead measurements for many system tasks (such as scheduling)
- Incorporated overheads into soft real-time schedulability experiments
- Task sets a mix of GPU-using and CPU-only

Overhead-Aware Schedulability Experiments

- Gathered overhead measurements for many system tasks (such as scheduling)
- Incorporated overheads into soft real-time schedulability experiments
  - Task sets a mix of GPU-using and CPU-only
- Different accounting techniques are required for each interrupt handling method

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

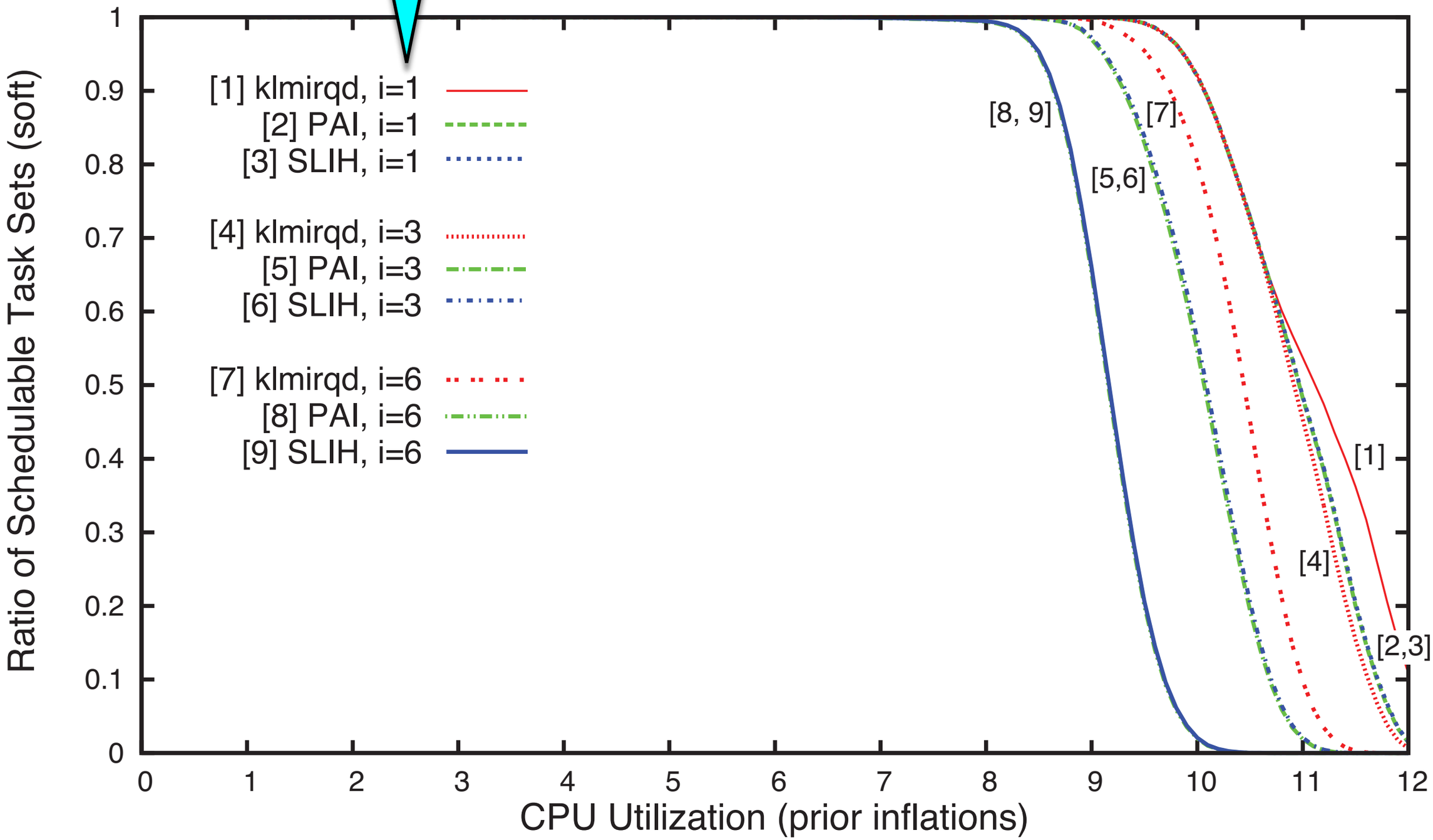
Overhead-Aware Schedulability Experiments



i is number of interrupts generated per job

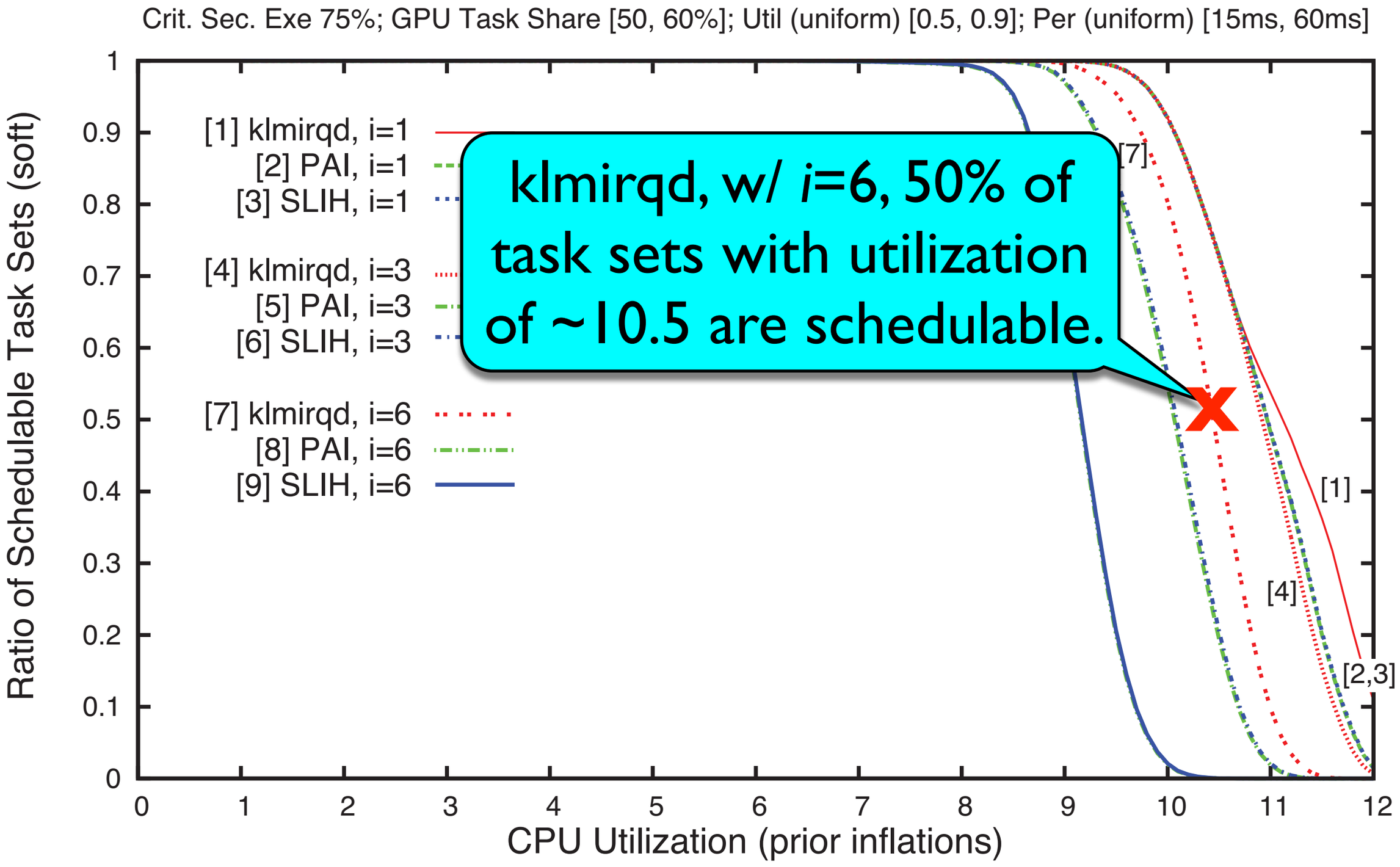
vare Schedulability eriments

60%]; Util (uniform) [0.5, 0.9]; Per (uniform) [15ms, 60ms]



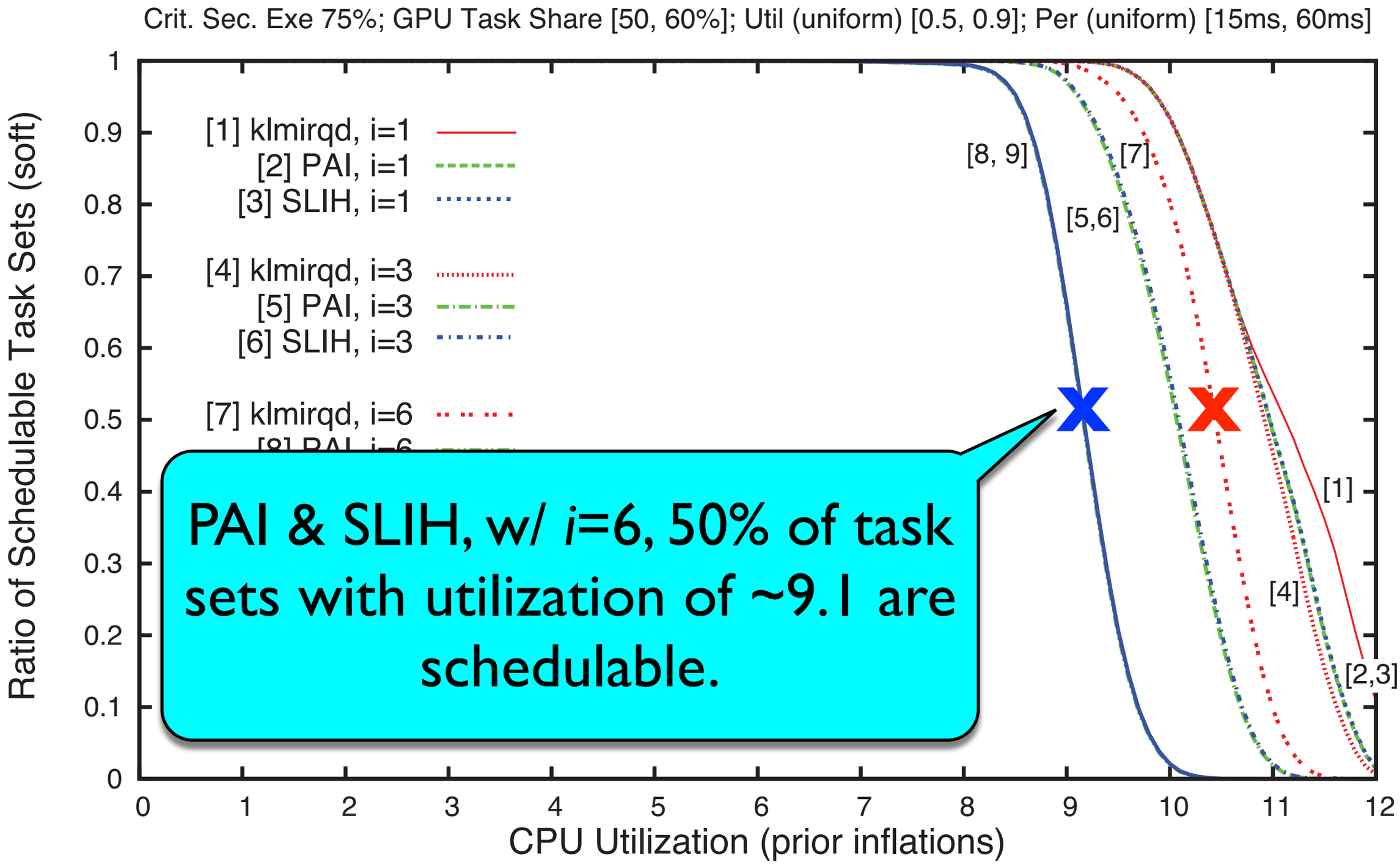
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Overhead-Aware Schedulability Experiments



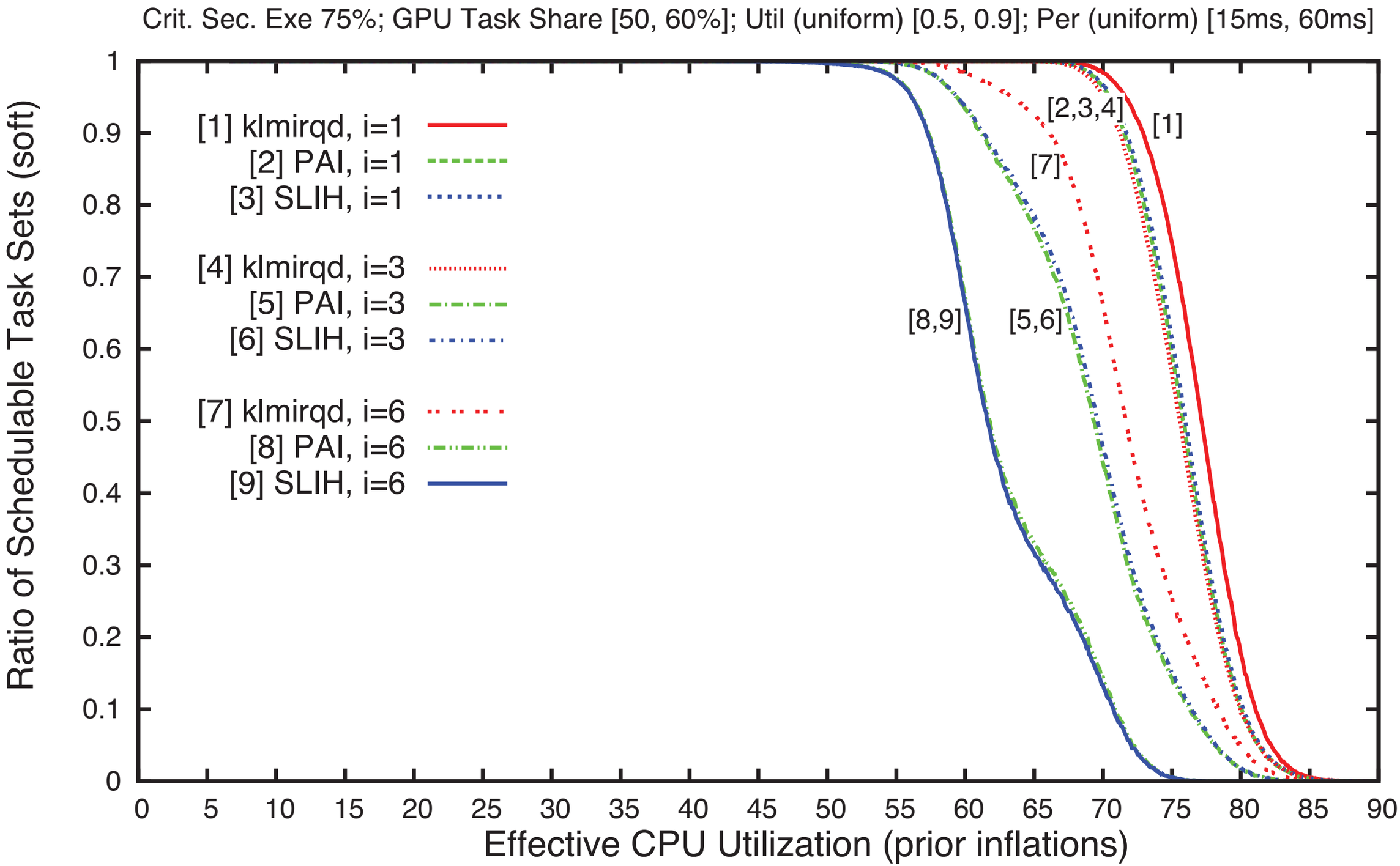
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Overhead-Aware Schedulability Experiments



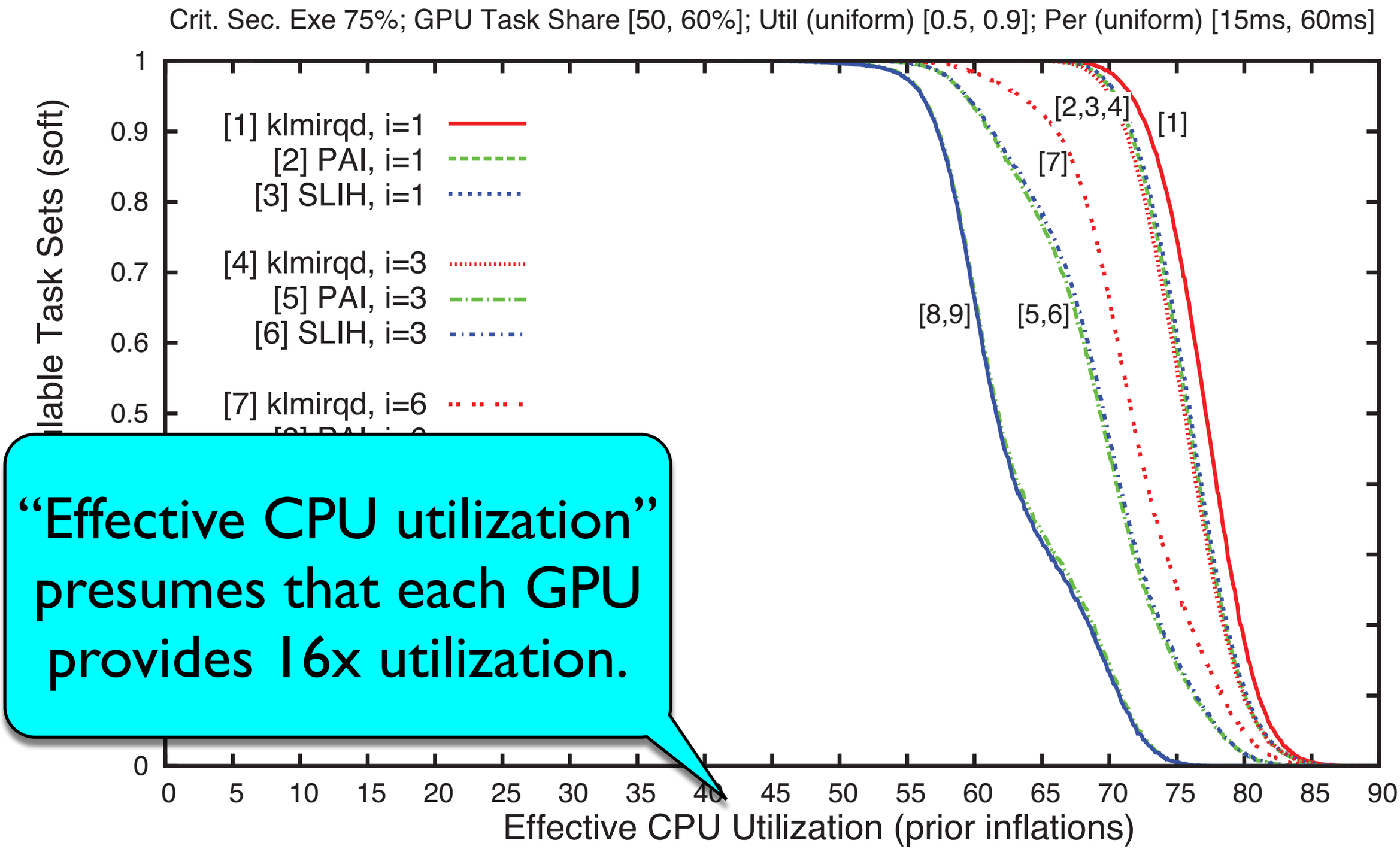
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Overhead-Aware Schedulability Experiments



THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

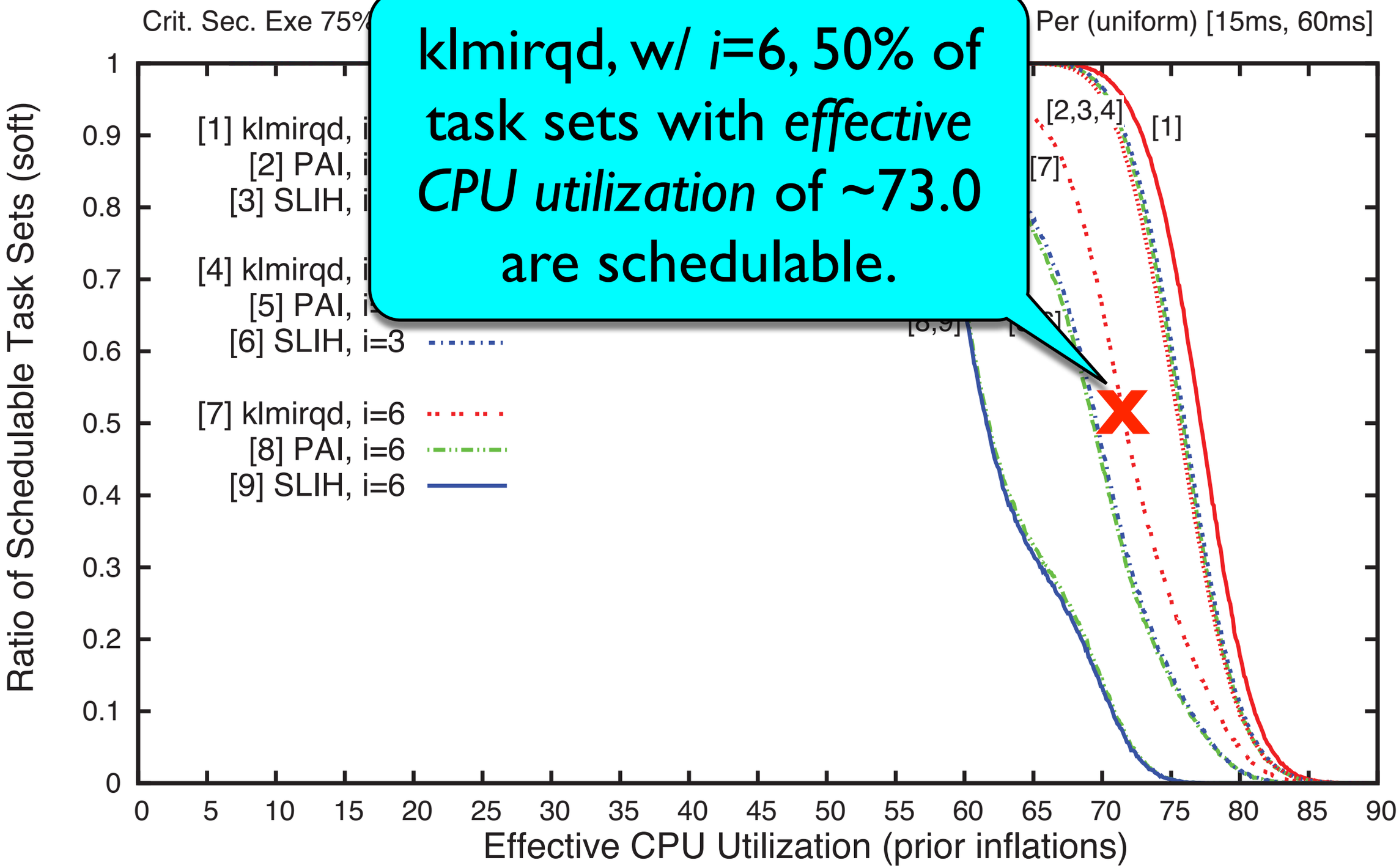
Overhead-Aware Schedulability Experiments



"Effective CPU utilization" presumes that each GPU provides 16x utilization.

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

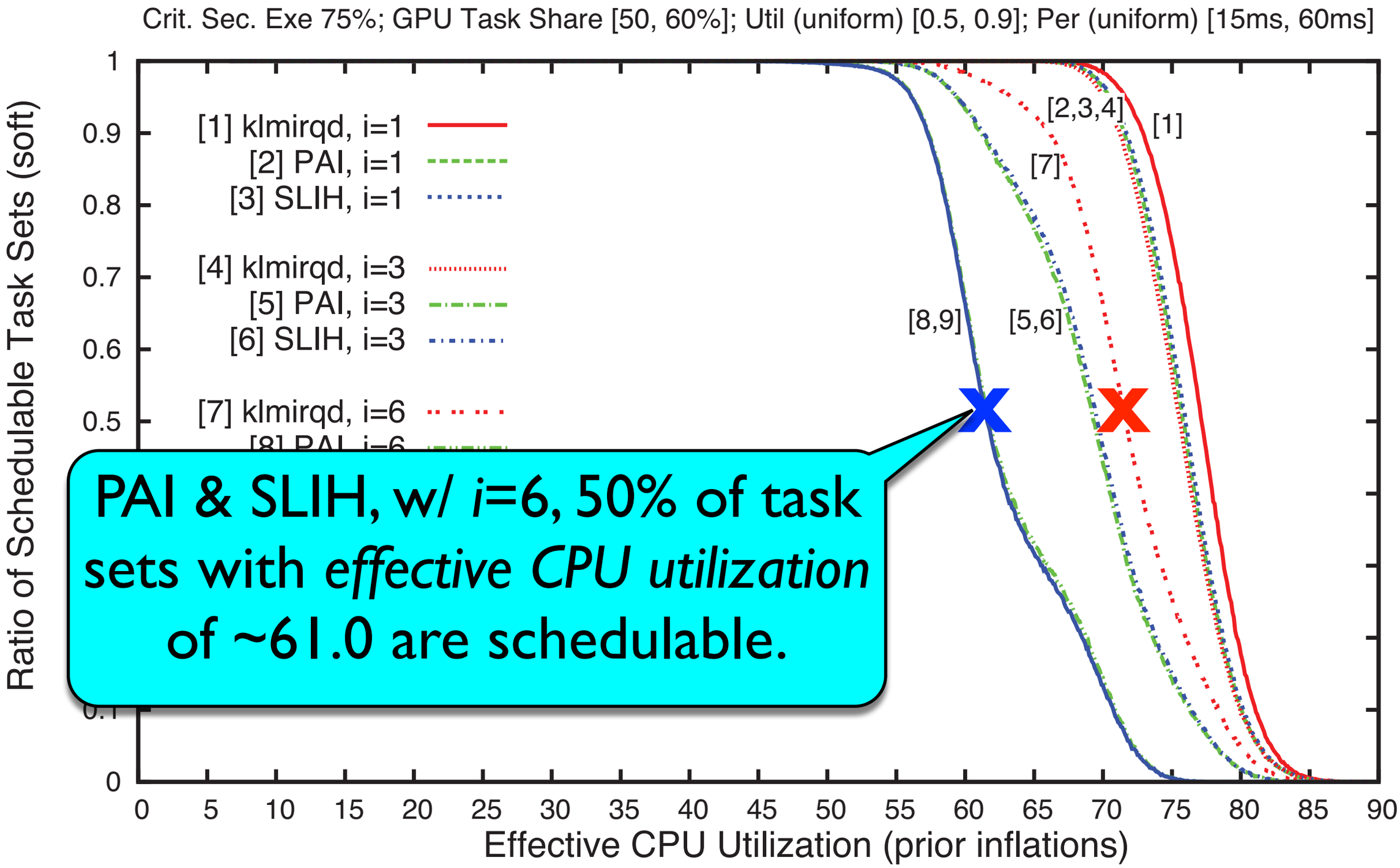
Overhead-Aware Schedulability Experiments



klmirqd, w/ *i*=6, 50% of task sets with *effective CPU utilization* of ~73.0 are schedulable.

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Overhead-Aware Schedulability Experiments



Conclusion

Conclusion

- Developed method for threaded interrupt handling under global scheduling with asynchronous I/O in mind

Conclusion

- Developed method for threaded interrupt handling under global scheduling with asynchronous I/O in mind
- Integrated closed source GPU driver through interrupt interception and decoding

Conclusion

- Developed method for threaded interrupt handling under global scheduling with asynchronous I/O in mind
- Integrated closed source GPU driver through interrupt interception and decoding
- Evaluations indicate klmirqd significantly reduces priority inversions while avoiding schedulability analysis pitfalls

Conclusion

- Developed method for threaded interrupt handling under global scheduling with asynchronous I/O in mind
- Integrated closed source GPU driver through interrupt interception and decoding
- Evaluations indicate klmirqd significantly reduces priority inversions while avoiding schedulability analysis pitfalls
- Source available at www.litmus-rt.org

Thank you!

Questions?

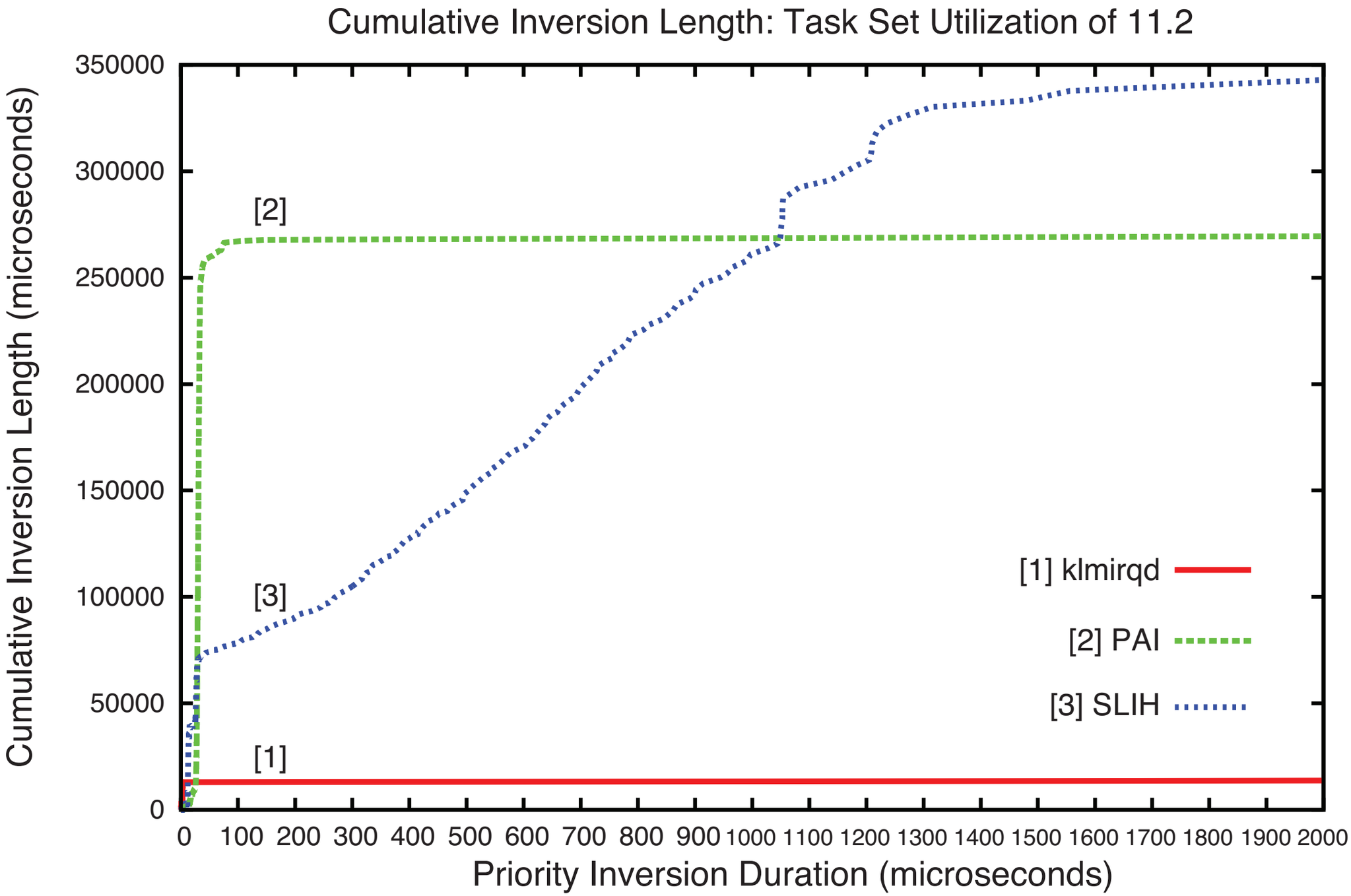
Backup Slides

Measured Overheads

Overhead	Average Time (µs)
Scheduling	0.63
Context Switch	0.36
IPI	0.60
Job Release	0.67
Top Half	**16.44**
Bottom Half	**29.90**
klmirqd Release	1.39
PAI Release	0.13
PAI Scheduling	0.56

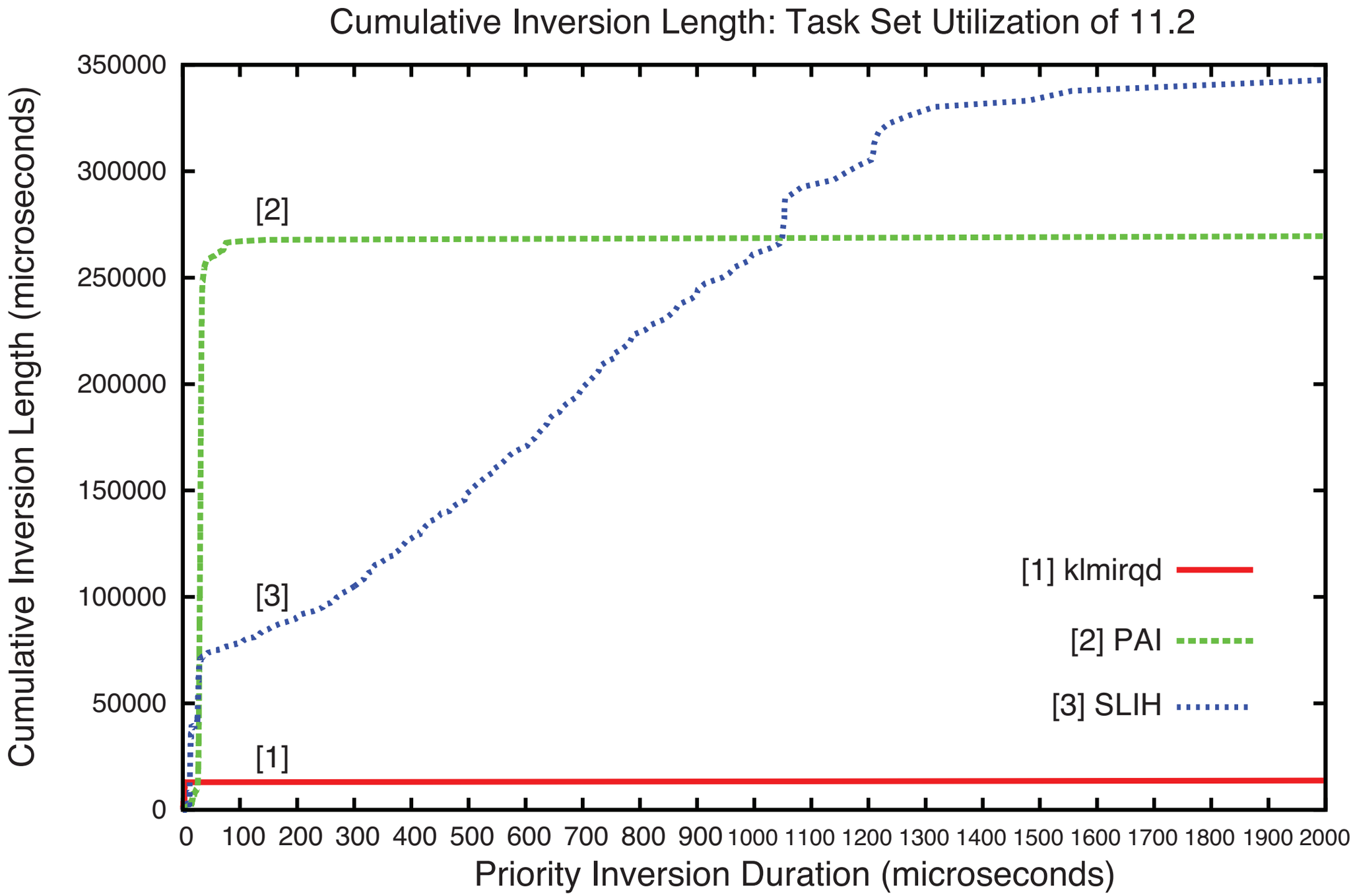
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Effect on Priority Inversions



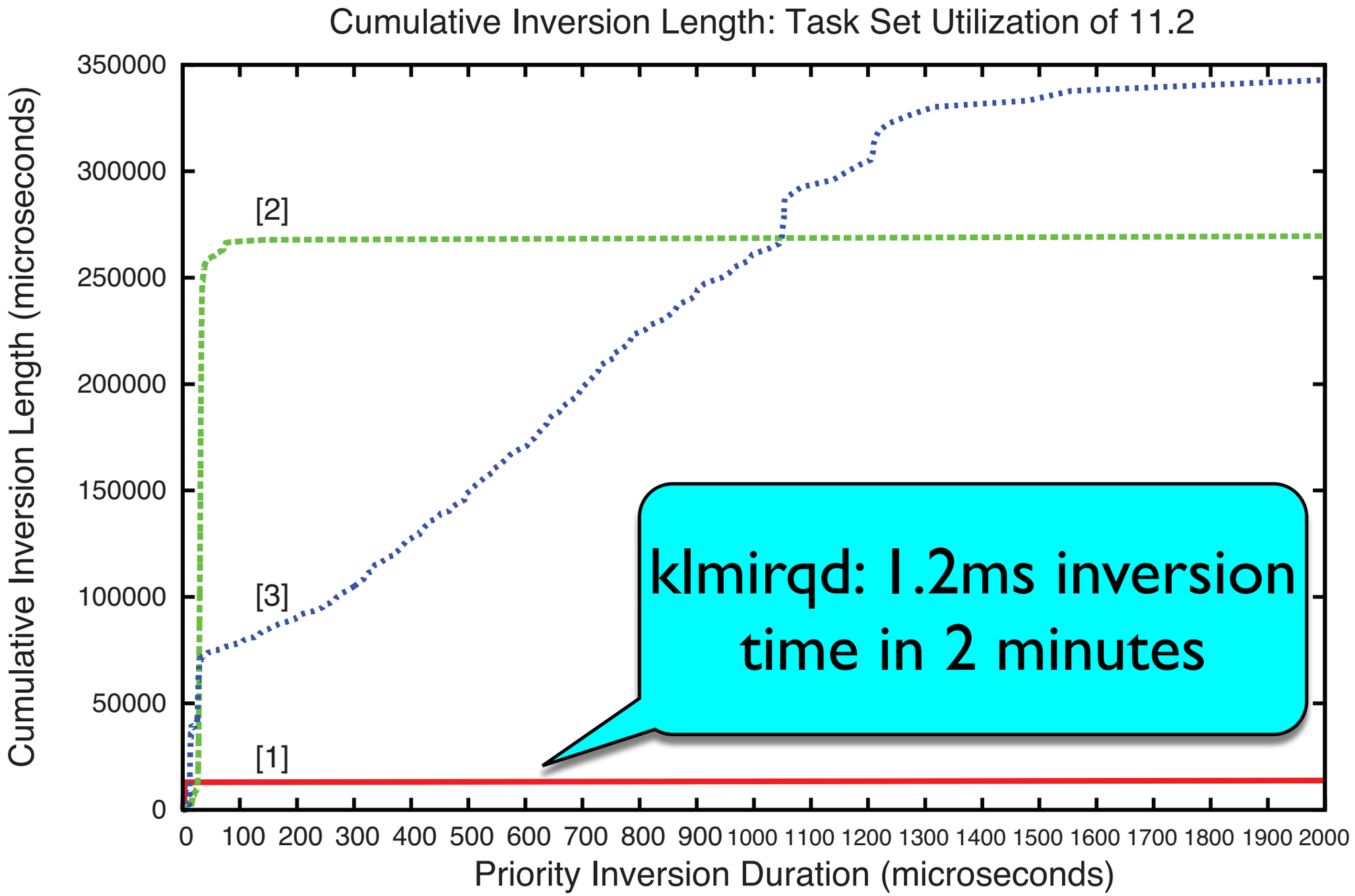
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Total inversion time reduced.



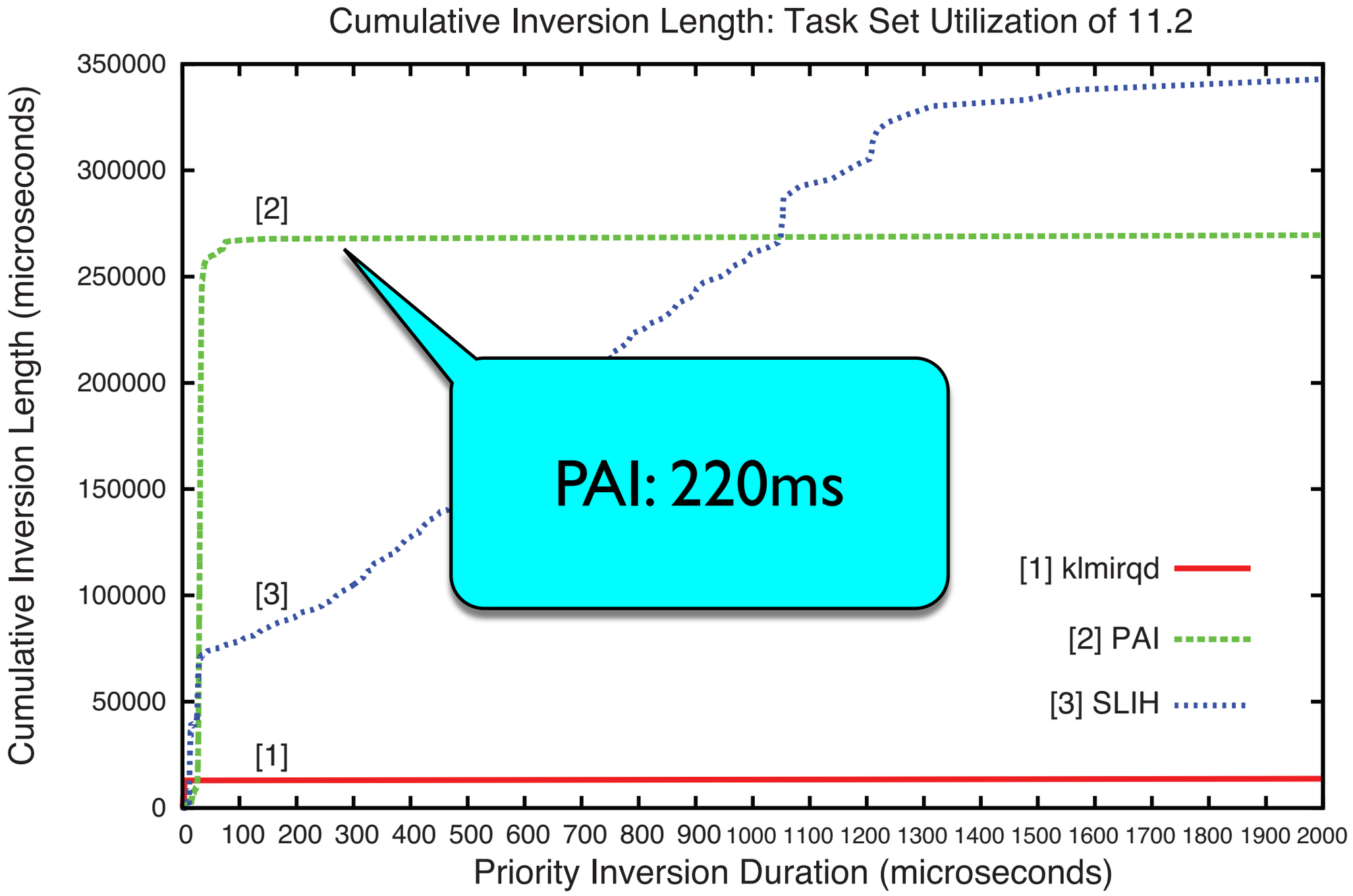
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Total inversion time reduced.

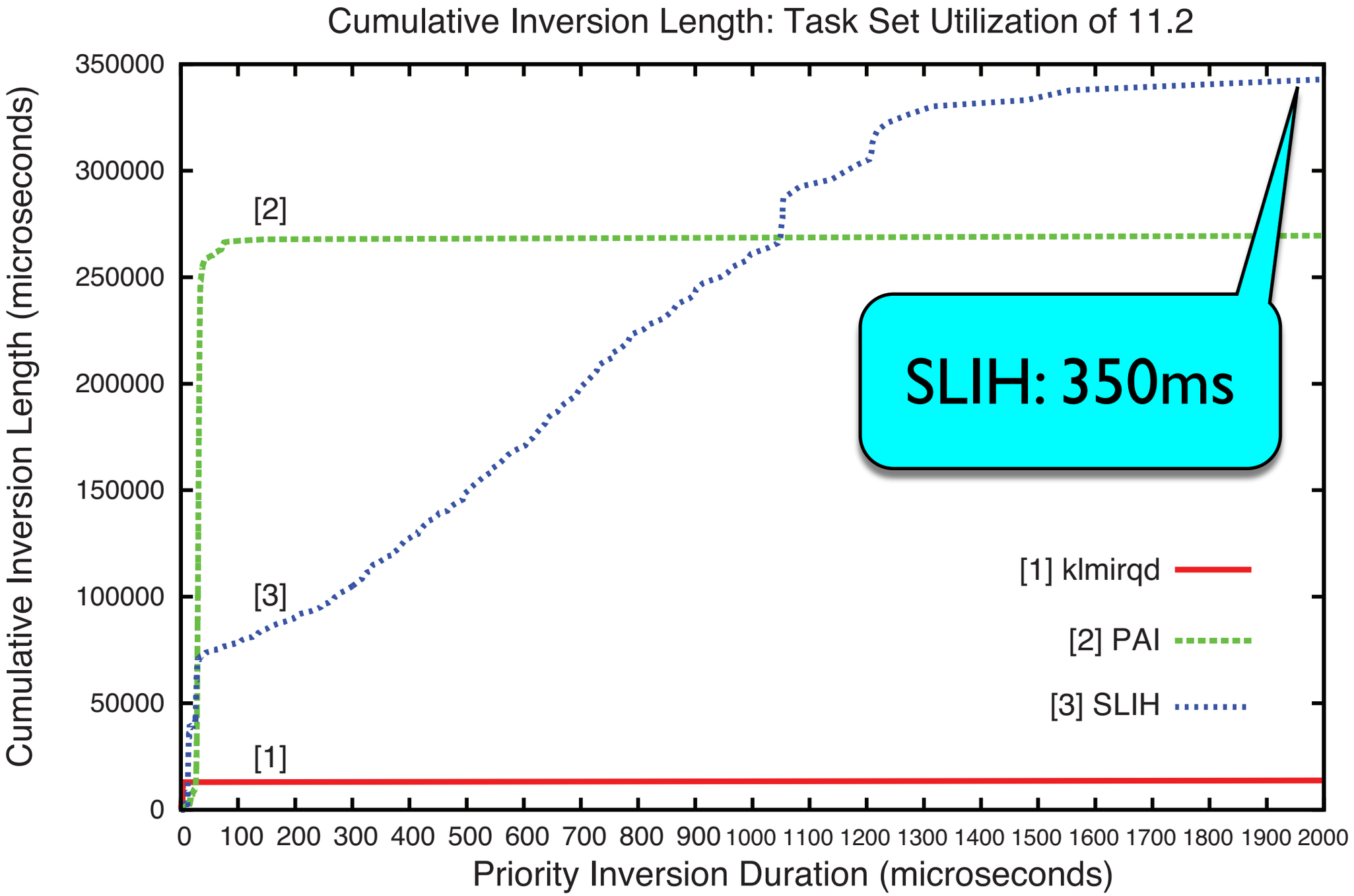


THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Total inversion time reduced.



Total inversion time reduced.

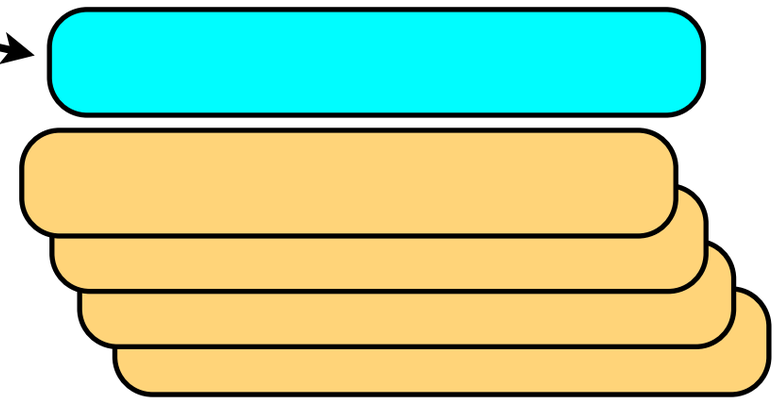


System-Level Evaluation

- Compared klmirqd against SLIH, PAI, and PREEMPT_RT (real-time Linux patch)
- PREEMPT_RT interrupt handler threads have *fixed* priority
- Scheduled using Clustered Rate Monotonic
  - Needed to make fair comparisons to PREEMPT_RT

Pathological Task Set

One GPU-using,
period = 19.9ms



20 CPU-only,
period = 20ms

Four GPU-using,
period = 20.1ms

One "sandwich"
on each cluster.

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

System-Level Evaluation

Scheduler:	C-RM						C-EDF	
Operating System:	PREEMPT_RT		Unmod. Linux	LITMUS^{RT}				
Interrupt Handling Method:	Low Prio. Interrupts (a)	High Prio. Interrupts (b)	SLIH (c)	SLIH (d)	klmirqd (e)	PAI (f)	klmirqd (g)	PAI (h)
Avg. % Miss Per Task								
CPU-Only Tasks	12.5%	12.5%	1.6%	10.0%	10.0%	9.9%	0%	0%
GPU-Using Tasks	10.1%	8.5%	6.8%	0%	0%	0%	0%	0%
Avg. Resp. Time as % Period								
CPU-Only Tasks	22474.5%	24061.0%	8992.1%	55.8%	55.8%	55.6%	55.4%	55.4%
GPU-Using Tasks	23066.1%	34263.5%	61131.7%	46.7%	49.6%	46.2%	46.2%	46.2%

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

System-Level Evaluation

Scheduler:	C-RM						C-EDF	
Operating System:	PREEMPT_RT		Unmod. Linux	LITMUS^{RT}				
Interrupt Handling Method:	Low Prio. Interrupts (a)	High Prio. Interrupts (b)	SLIH (c)	SLIH (d)	klmirqd (e)	PAI (f)	klmirqd (g)	PAI (h)
Avg. % Miss Per Task								
CPU-Only Tasks	12.5%	12.5%	1.6%	10.0%	10.0%	9.9%	0%	0%
GPU-Using Tasks	10.1%	8.5%	6.8%	0%	0%	0%	0%	0%
Avg. Resp. Time as % Period								
CPU-Only Tasks	22474.5%	24061.0%	8992.1%	55.8%	55.8%	55.6%	55.4%	55.4%
GPU-Using Tasks	23066.1%	263.5%	61131.7%	46.7%	49.6%	46.2%	46.2%	46.2%

GPU starvation

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

System-Level Evaluation

Scheduler:	C-RM						C-EDF	
Operating System:	PREEMPT_RT		Unmod. Linux	LITMUS^{RT}				
Interrupt Handling Method:	Low Prio. Interrupts (a)	High Prio. Interrupts (b)	SLIH (c)	SLIH (d)	klmirqd (e)	PAI (f)	klmirqd (g)	PAI (h)
Avg. % Miss Per Task								
CPU-Only Tasks	12.5%	12.5%	1.6%	10.0%	10.0%	9.9%	0%	0%
GPU-Using Tasks	10.1%	8.5%	6.8%	0%	0%	0%	0%	0%
Avg. Resp. Time as % Period								
CPU-Only Tasks	22474.5%	24061.0%	8992.1%	55.8%	55.8%	55.6%	55.4%	55.4%
GPU-Using Tasks	23066.1%	34263.5%	61131.7%	46.7%	49.6%	46.2%	46.2%	46.2%

CPU response time increase