A Synchronous IPC Protocol for Predictable Access to Shared Resources in Mixed-Criticality Systems

RTSS'14
December 4, 2014

Max Planck Institute for Software Systems

Björn B. Brandenburg
bbb@mpi-sws.org

LITMUS[RT]
Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

How To Synchronize Access to Shared Resources?

(such as data structures, OS services, I/O ports, …)

Part 1

Locking is the wrong approach *in mixed-criticality (MC) systems.*

Part 2

MC-IPC*:* ***Predictable*** *IPC for* ***Cross-Criticality*** *Resource Sharing*

Part 3

A Case Study: Freedom-from-Interference *Despite Failures, DoS,…*

Synchronization vs. Isolation in Mixed-Criticality Systems

Isolation: The Core MC Requirement

Freedom from Interference

High-criticality tasks not negatively affected by other (low-criticality) tasks.

Isolation: The Core MC Requirement

Freedom from Interference

High-criticality tasks not negatively affected by other (low-criticality) tasks.

"analyze & trust"

"isolate & enforce"

Isolation: The Core MC Requirement

Freedom from Interference

High-criticality tasks not negatively affected by other (low-criticality) tasks.

“analyze & trust”

Design Time Proof Obligation
Show that a (**lower-criticality**) task ***does not*** cause interference.

At runtime
Trust *that other tasks do not cause unpredictable interference.*

“isolate & enforce”

Mandate Strict Isolation
Ensures that a (**lower-criticality**) task ***cannot*** cause interference.

At runtime
Unpredictable interference ***impossible*** *due to* ***isolation***.

Isolation: The Core MC Requirement

The Problem with "Analyze & Trust"

Need to ***establish absence of interference*** at the ***level of assurance*** of the **highest-criticality** task that ***could be interfered*** with.

➔ *we're back to applying the **highest standards** to **all** tasks…*

"analyze & trust"

Design Time Proof Obligation

Show that a (**lower-criticality**) task ***does not*** cause interference.

At runtime

***Trust** that other tasks do not cause unpredictable interference.*

"isolate & enforce"

Mandate Strict Isolation

Ensures that a (**lower-criticality**) task ***cannot*** cause interference.

At runtime

*Unpredictable interference **impossible** due to **isolation**.*

Isolation: The Core MC Requirement

Benefit of Isolation

Need to apply **highest standards only** when validating the **isolation mechanism** itself.

But **NOT** to each isolated (**lower-criticality**) task.

“analyze & trust”

Design Time Proof Obligation

Show that a (**lower-criticality**) task ***does not*** cause interference.

At runtime

Trust *that other tasks do not cause unpredictable interference.*

“isolate & enforce”

Mandate Strict Isolation

Ensures that a (**lower-criticality**) task ***cannot*** cause interference.

At runtime

Unpredictable interference ***impossible*** *due to* ***isolation****.*

Isolation: The Core MC Requirement

Freedom from Interference

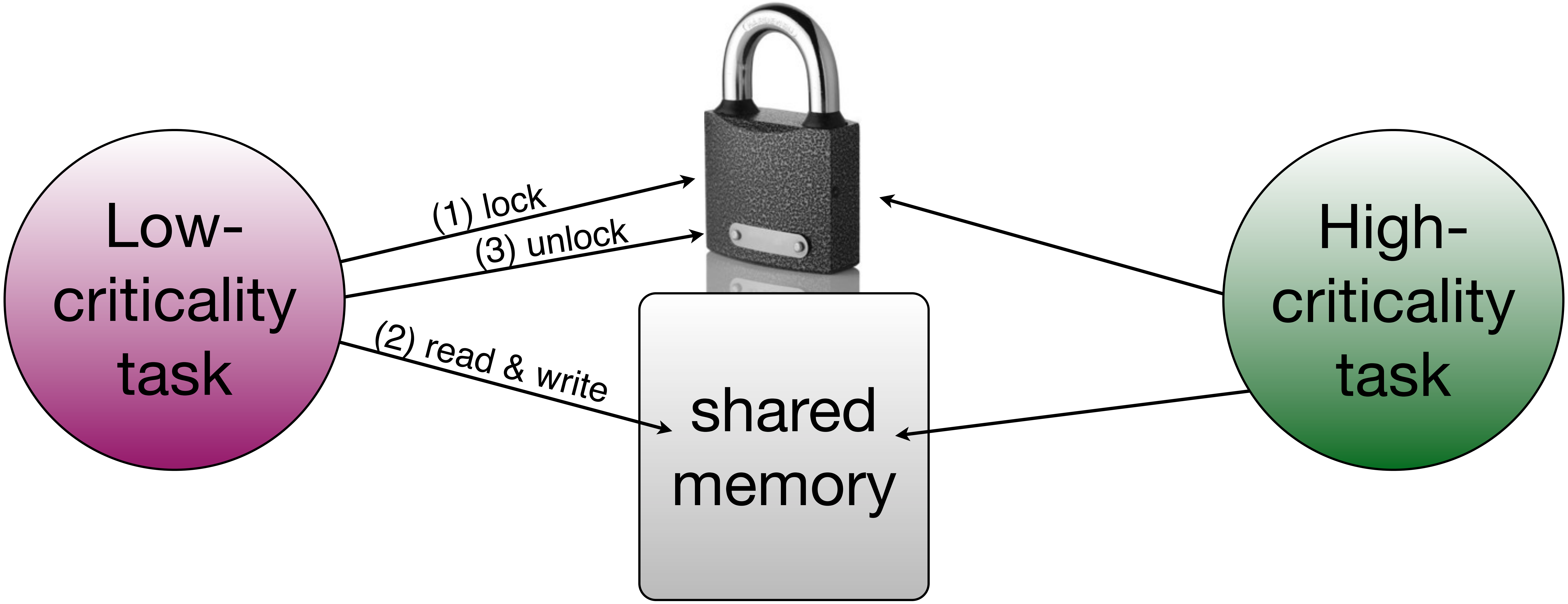
High-criticality tasks not negatively affected by other (low-criticality) tasks.

➔ **logical** isolation + **temporal** isolation

Locking Implies Trust

Real-time locking protocols: PIP, PCP, MPCP, FMLP, OMLP …

➡ Applicable across multiple criticalities?

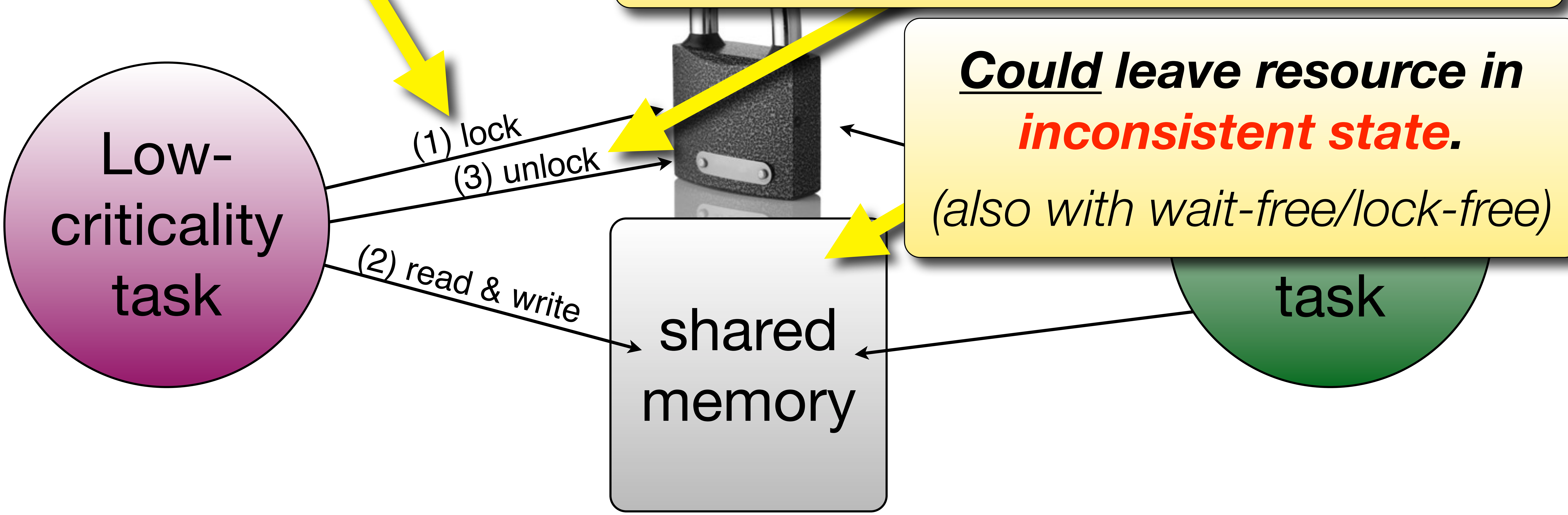


Locking Implies Trust

Could **access** ***without locking*** **first.**

Could fail to unlock
(or hold lock for arbitrary duration).

Could **leave resource in** ***inconsistent state*****.**
(also with wait-free/lock-free)

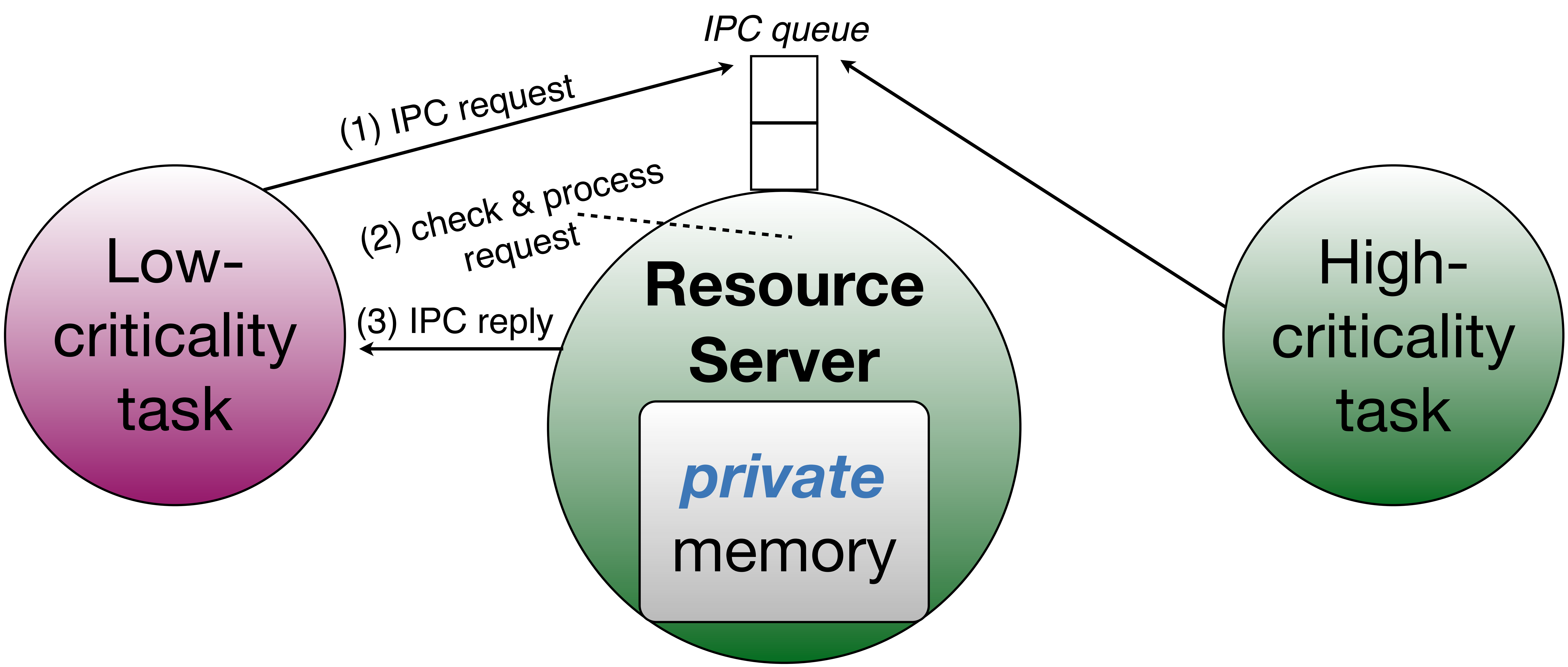


Better: Message Passing / IPC

Isolate resource in resource server process to mediate access

➡ Use *inter-process communication* (IPC) to request service

➡ Canonical approach in μ-kernels…

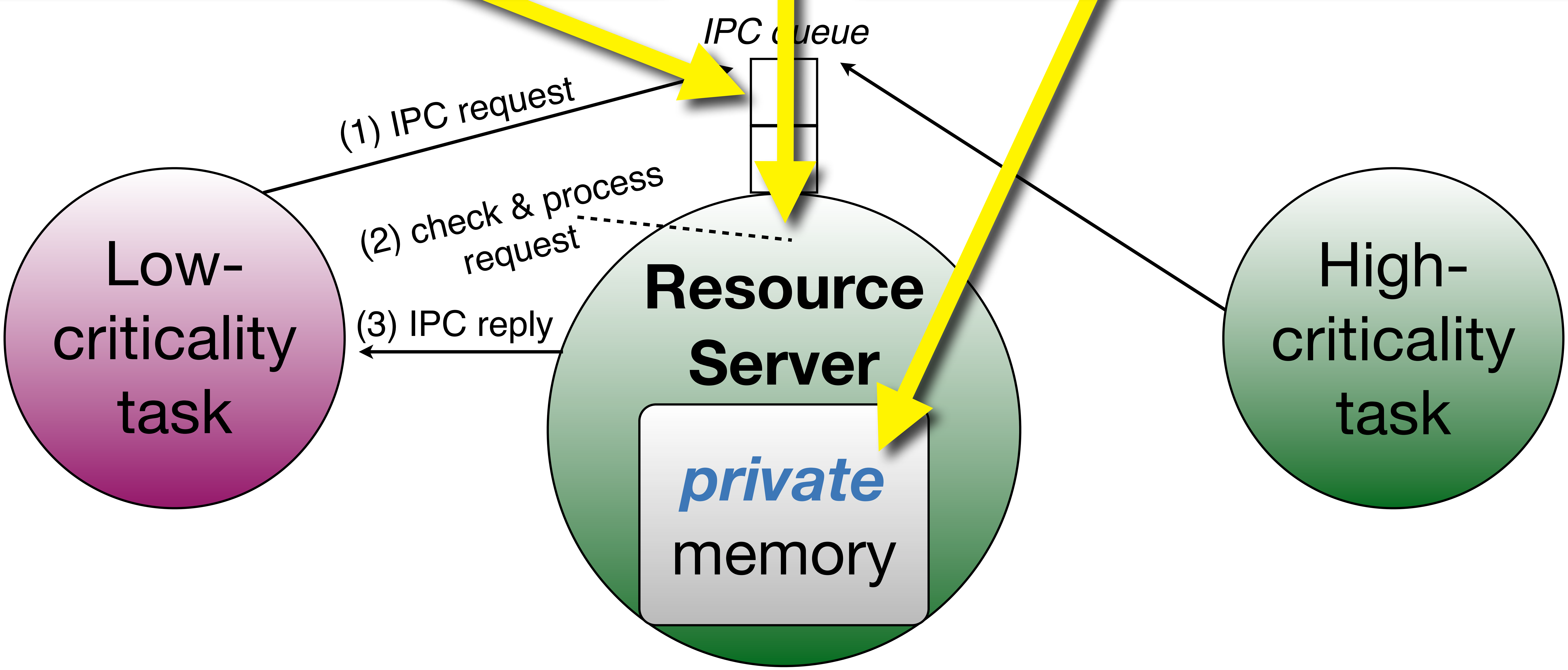


Better: … / IPC

Untrusted client cannot occupy resource indefinitely.

Untrusted client cannot bypass synchronization.

Untrusted client cannot leave resource in inconsistent state.



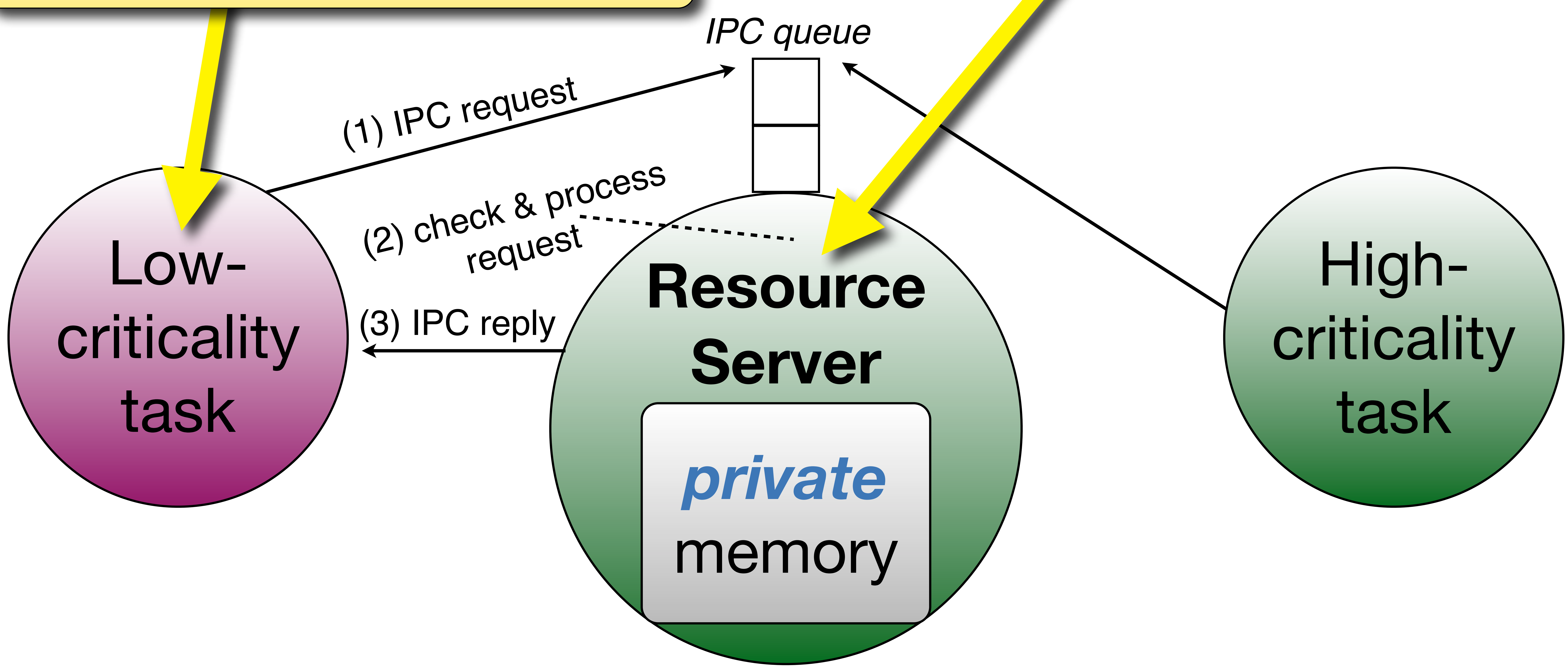
Logical isolation: Must still **validate and trust the server** at **highest-criticality level** of assurance…

Isolate resource in resource server process to mediate access

…ication (IPC) to request service

…ernels…

…but not all clients!

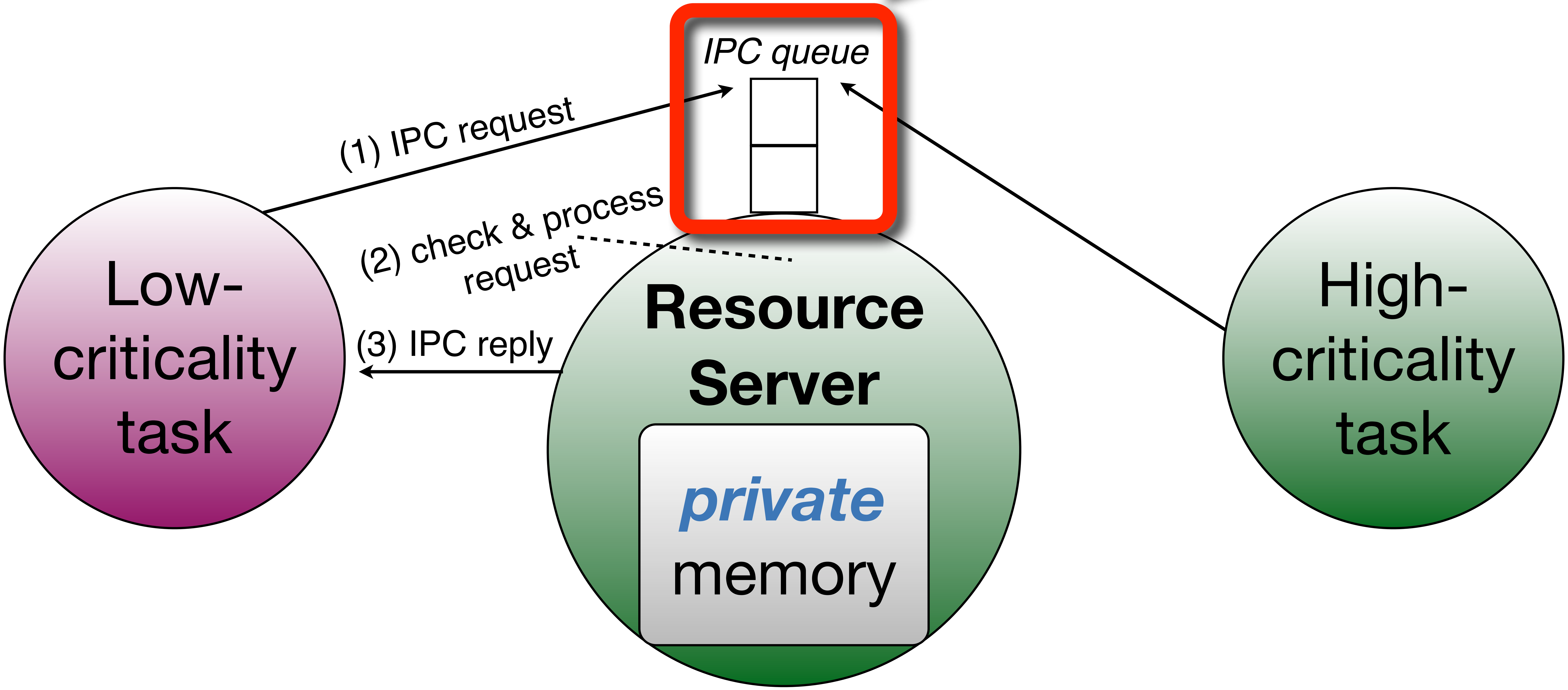


***Temporal Isolation:* What is the *maximum IPC queueing delay*?**

…and what happens if clients misbehave!?

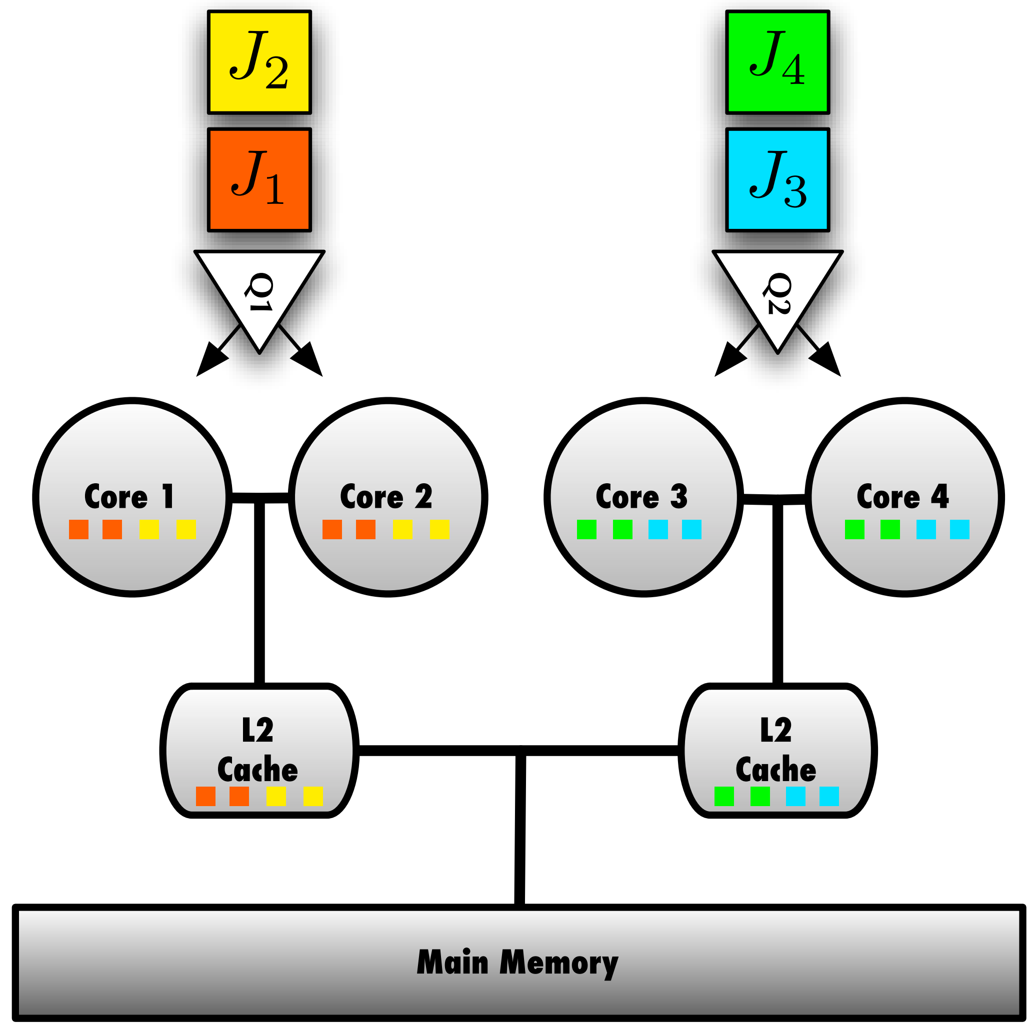
Isolate resource in resource server process to mediate access

- ➡ Use *inter-process communication* (IPC) to request service
- ➡ Canonical approach in *μ*-kernels…



MC-IPC
An IPC Protocol for Mixed-Criticality Systems

System Topology: Clustered Scheduling



***m* = 4**

***K* = 2**

m_1 = 2

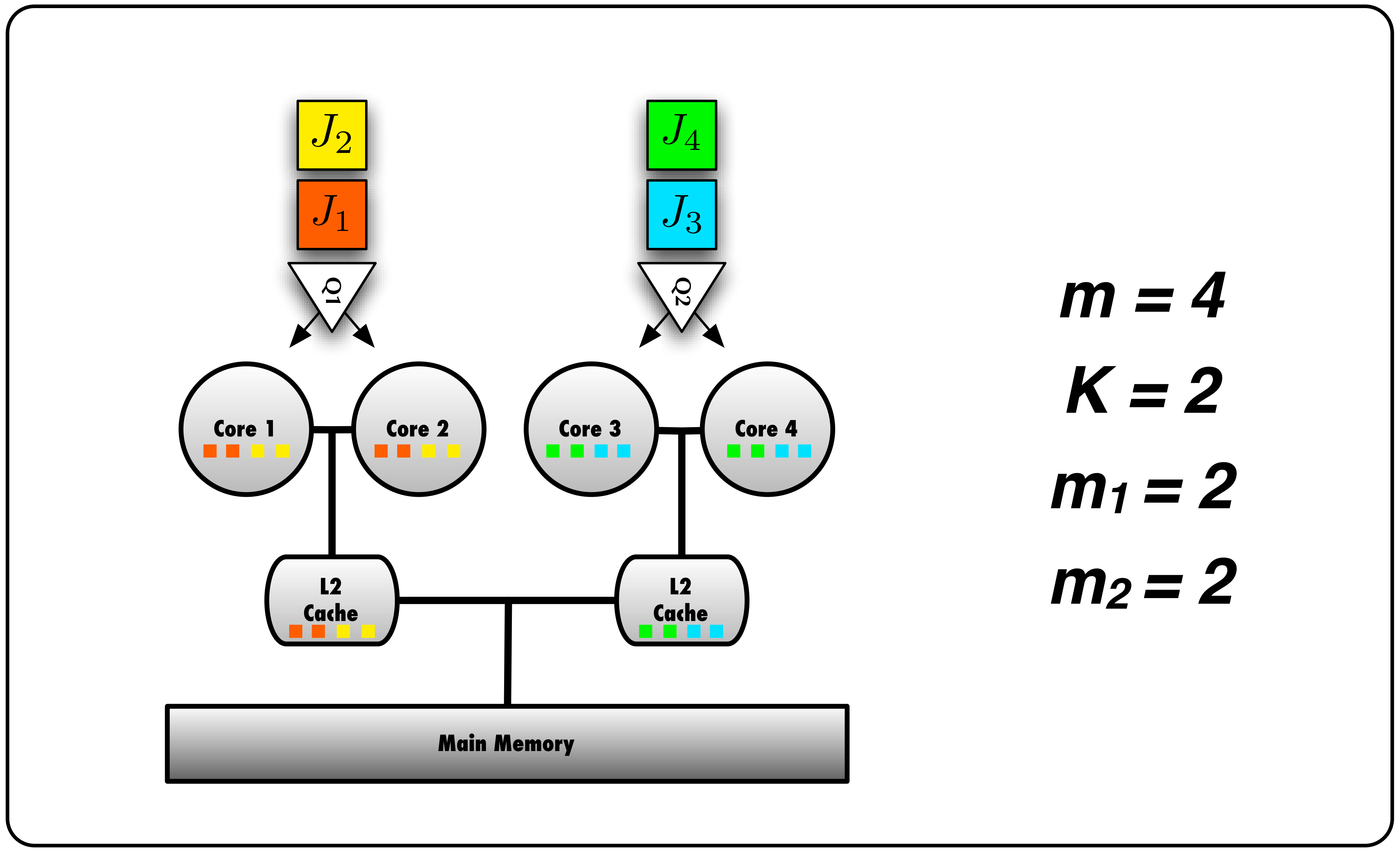
m_2 = 2

Shared-Memory <u>multiprocessor</u> platform

- ➡ Organized into K clusters $C_1, C_2, \ldots, C_K$
- ➡ m_i processors in cluster C_i
  - *non-uniform clusters* permitted

Partitioned Scheduling: m_i = *1* in each cluster (= partition)

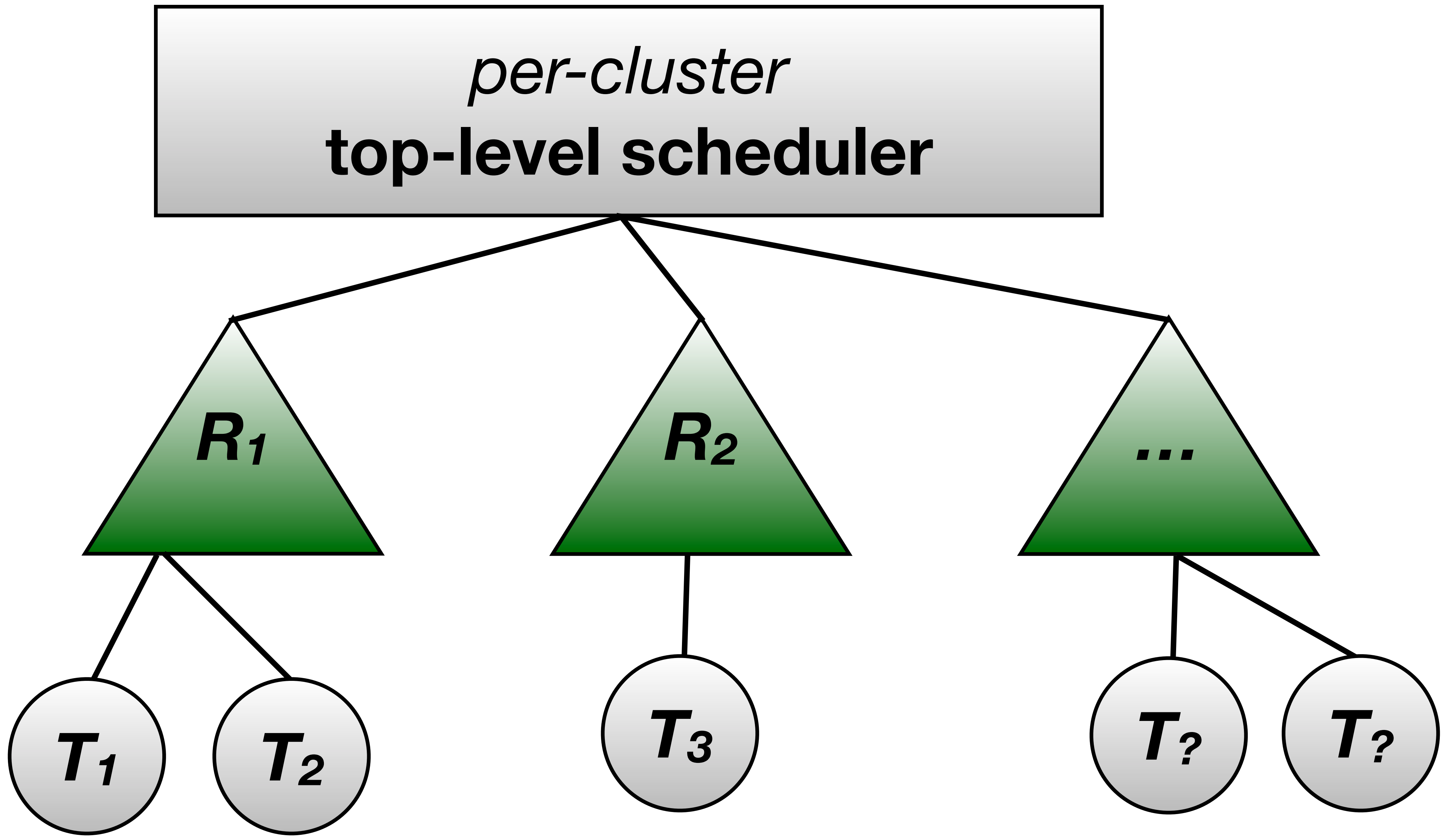
Special case most common in practice (and used in evaluation).



Shared-Memory <u>multiprocessor</u> platform

- ➡ Organized into K clusters $C_1, C_2, \ldots, C_K$
- ➡ m_i processors in cluster C_i
  - *non-uniform clusters* permitted

Temporal Isolation: Reservation-Based Scheduling



Reservation-Based Scheduling

- ***R_1, R_2,…***
- top-level scheduler chooses reservation
- reservation chooses task

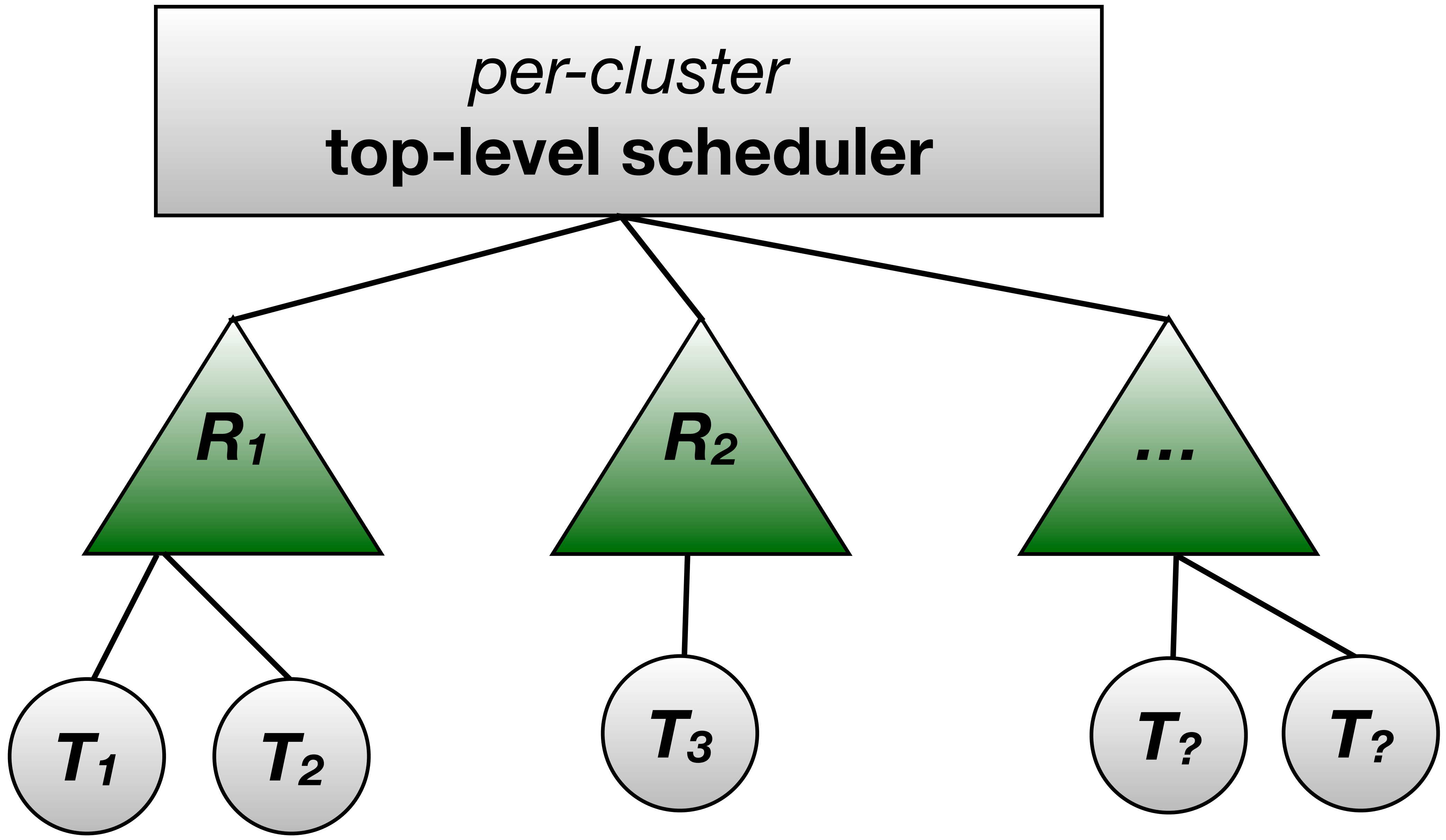
Each reservation has a…

- current **priority**
- current **budget**
- one or more **client tasks**

Specific type of reservation intentionally left undefined.

- Compatible examples:
  - *polling reservation*
  - *table-driven reservation*
  - *constant bandwidth server* (CBS)
  - …

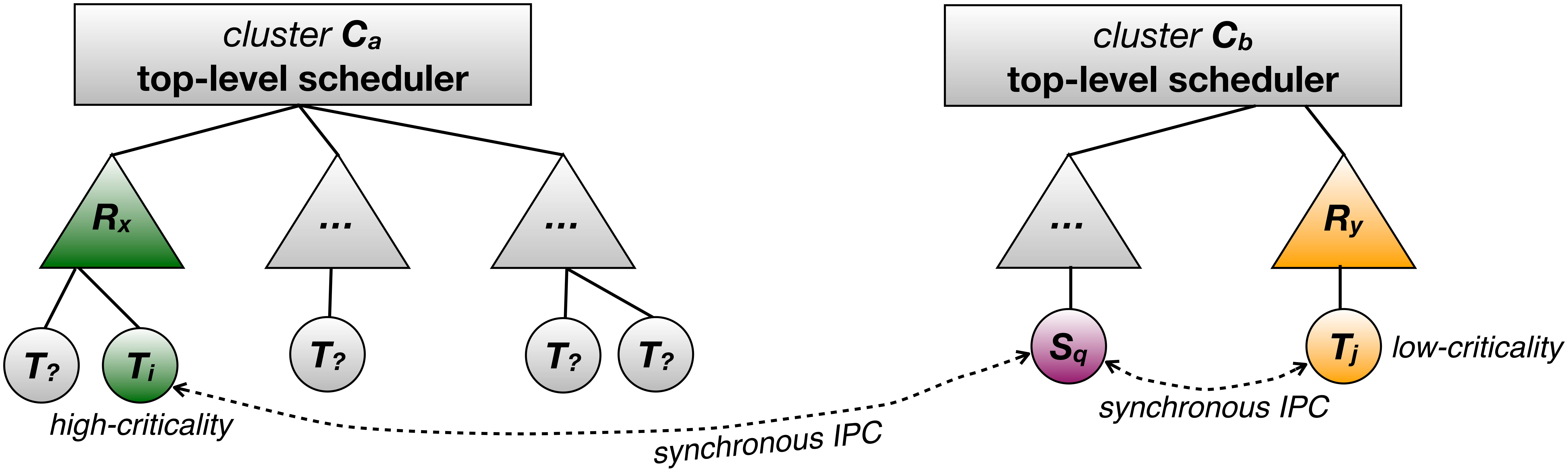
Reservation Assumptions



Assumptions

- **A1**: the **priority** of a reservation ***changes only*** when its budget is ***exhausted*** or ***replenished***.
- **A2**: a reservation selected for service by the top-level scheduler **consumes budget** even if all clients are blocked on IPC (***idling rule***).

Resource Servers



Shared resource servers

➡ each shared resource ***q*** encapsulated in resource server S_q

Logical Isolation

➡ OS: isolate server in private protection domain (***private address space***)

➡ server implementation: ***reject requests*** that are illegal / illformed / non-sensical

Temporal Isolation

➡ server implementation: bounded maximum operation length

➡ IPC protocol must ensure **bounded IPC delay (= bandwidth consumption)**

What Could Violate Temporal Isolation?

P1: The resource server may be preempted indefinitely
➡ *Need to ensure timely IPC request completion…*

P2: Clients may attempt to monopolize server
➡ *Need to prevent starvation… (➔ FIFO queue?)*

P3: There may be an unpredictable number of contending clients
➡ *Need to respect priority of requesting clients… (➔ priority queue?)*

P4: A client may run out of budget while waiting for server
➡ *Need to prevent backlog of "stale" clients…*

P5: Best-effort background tasks may need to access server
➡ *Some system services inherently shared (e.g., network stack)…*

What Could Violate Temporal Isolation?

No existing design is resilient to all of these issues.

P3: There may be an unpredictable number of contending clients

➡ *Need to respect priority of requesting clients… (➔ priority queue?)*

P4: A client may run out of budget while waiting for server

➡ *Need to prevent backlog of "stale" clients…*

P5: Best-effort background tasks may need to access server

➡ *Some system services inherently shared (e.g., network stack)…*

What Could Violate Temporal Isolation?

No existing design is resilient to all of these issues.

MC-IPC: we can ***tolerate all causes*** by combining **prior techniques** and simple **commonsense solutions** in just the right way…

P1: Ensuring Server Progress

Server preempted while replying to IPC
= Lock-holder preempted while executing critical section.

➔ *apply well-known techniques for ensuring **lock-holder progress***

P1: Ensuring Server Progress

Server preempted while replying to IPC
= Lock-holder preempted while executing critical section.

Ensuring timely IPC completion: Bandwidth Inheritance

- ➡ let server execute on budget of any client (implicit in ***idling rule***).
- ➡ When preempted, **migrate server** to processor of waiting client.

Well-known solution

- ➡ "Helping" / "timeslice donation" in L4/Fiasco
  - *Hohmuth & Härtig, 2001.*
- ➡ Multiprocessor Bandwidth Inheritance (MBWI)
  - *Faggioli et al., 2010 & 2012.*

M. Hohmuth and H. Härtig,"Pragmatic nonblocking synchronization for real-time systems," in Proc. USENIX ATC'01, 2001.
D. Faggioli, G. Lipari, and T. Cucinotta, "Analysis and implementation of the multiprocessor bandwidth inheritance protocol," Real-Time Systems, vol. 48, no. 6, pp. 789–825, 2012.

P2 & P3: Dealing With Unknown Contention

An unknown number of tasks may issue requests at arbitrary rates.

P2 & P3: Dealing With Unknown Contention

An unknown number of tasks may issue requests at arbitrary rates.

Need to ***respect priorities within each cluster****,*
but also need to ensure ***fairness among clusters****.*

➔ *can apply* ***OMIP*** *three-level queue structure [ECRTS'13]*

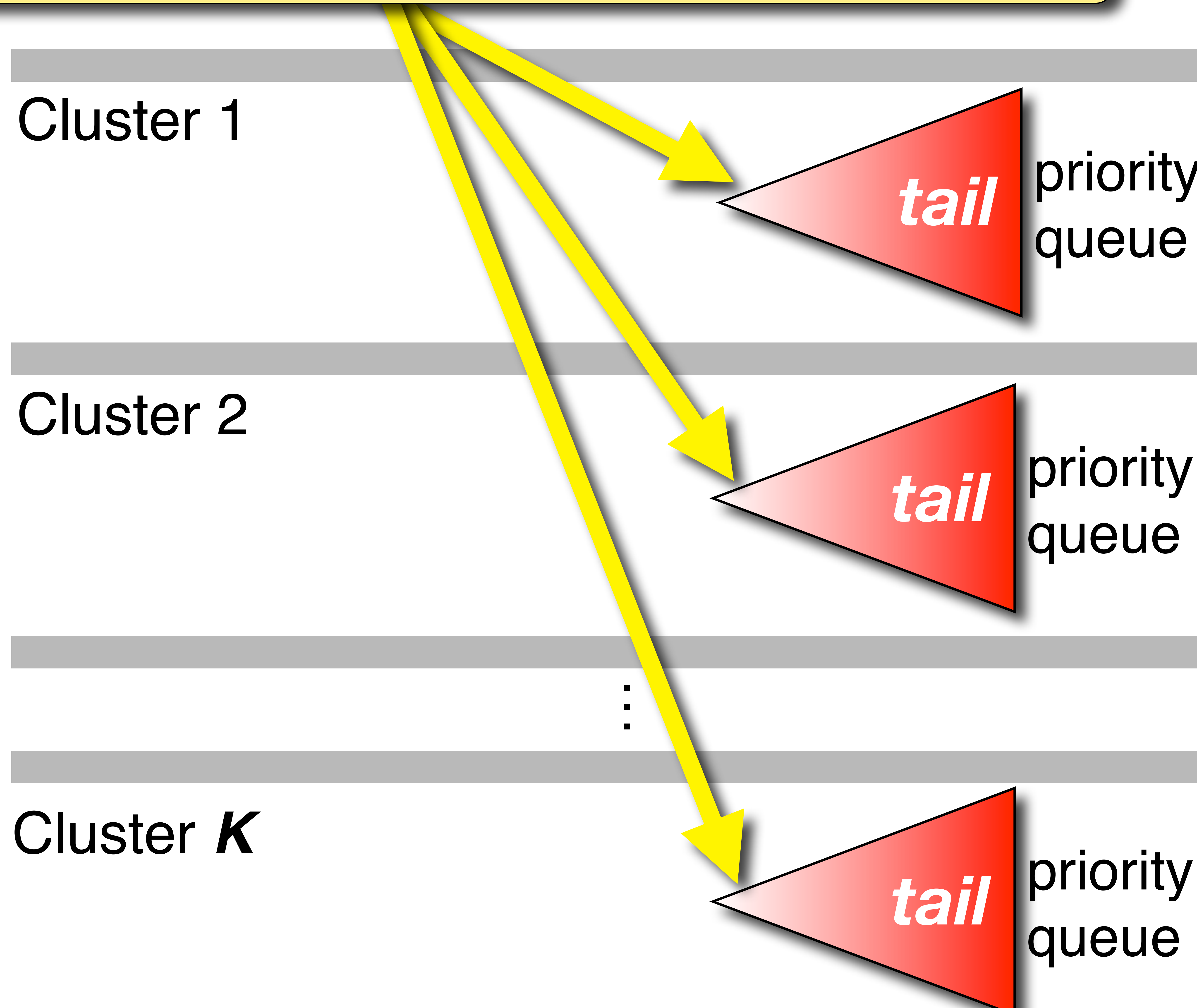
OMIP: —, "A Fully Preemptive Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications", In Proc. ECRTS'13, July 2013.

P2 & P3: Dealing With Unknown Contention

An unknown number of tasks may issue requests at arbitrary rates.

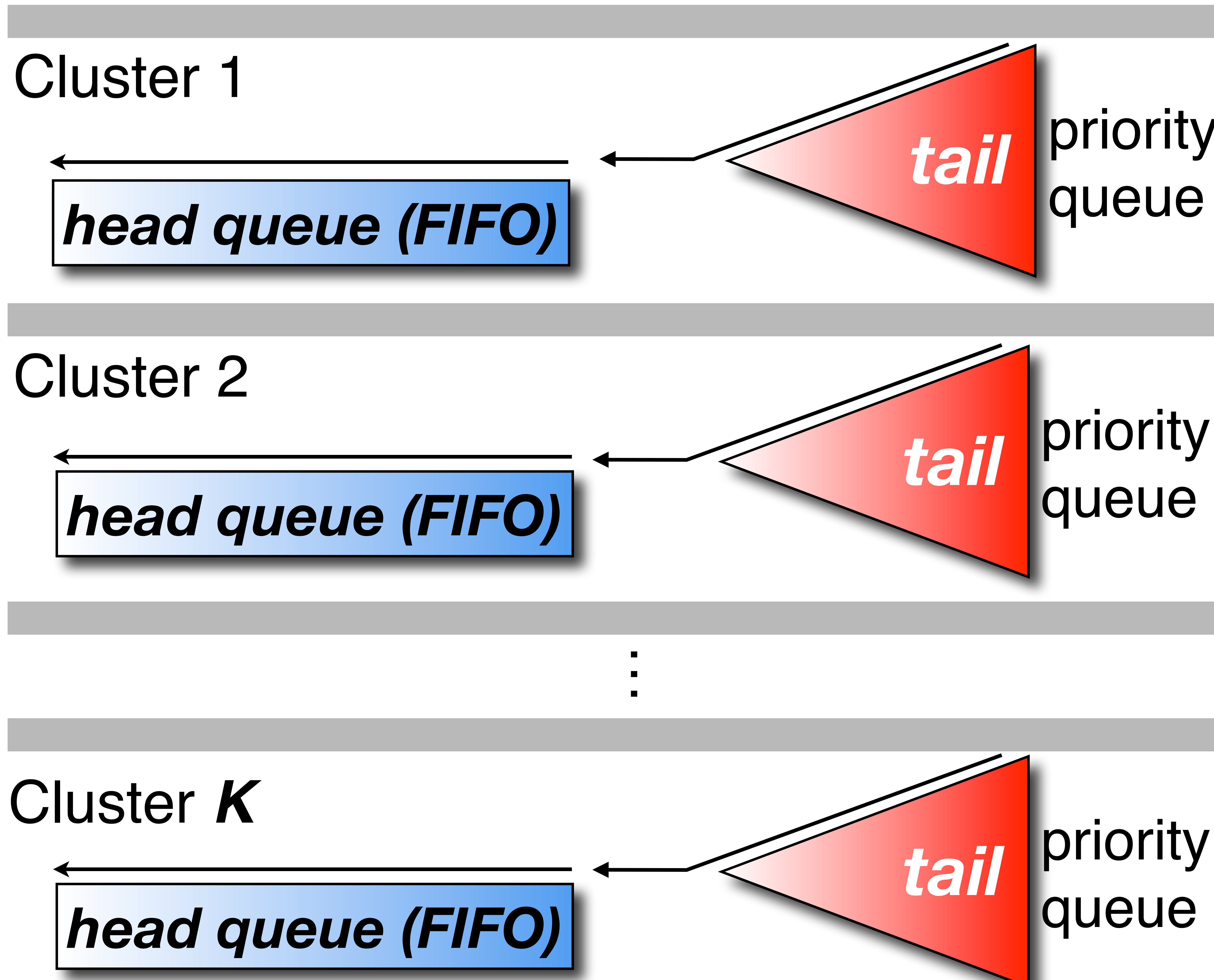
Clients ordered by current reservation priority.

*Priority-ordered w.r.t. **top-level scheduler**,*
***not** reservation-internal priorities (if any): important for **proof**.*



P2 & P3: Dealing With Unknown Contention

An unknown number of tasks may issue requests at arbitrary rates.

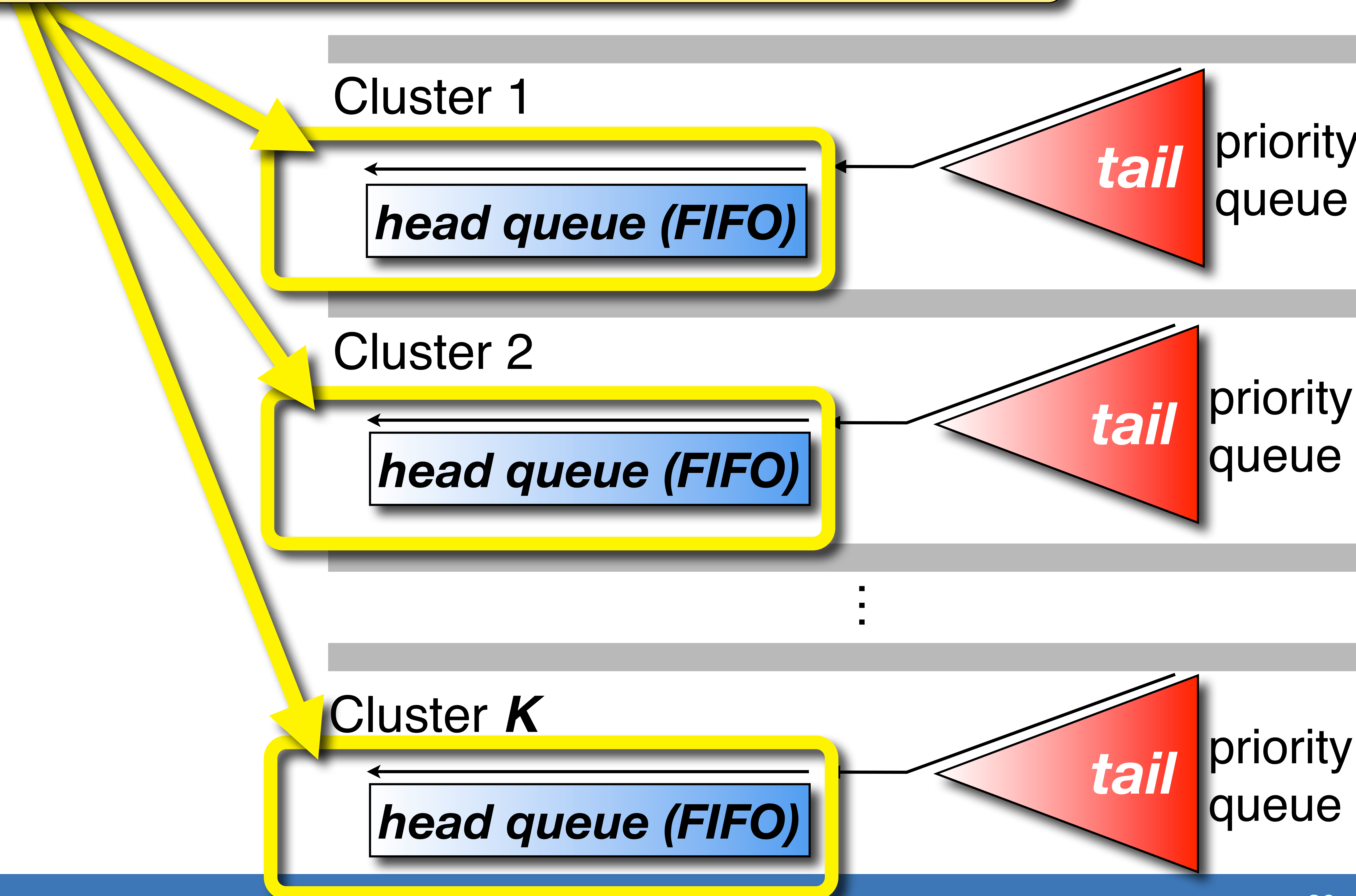


...ention

...ary rates.

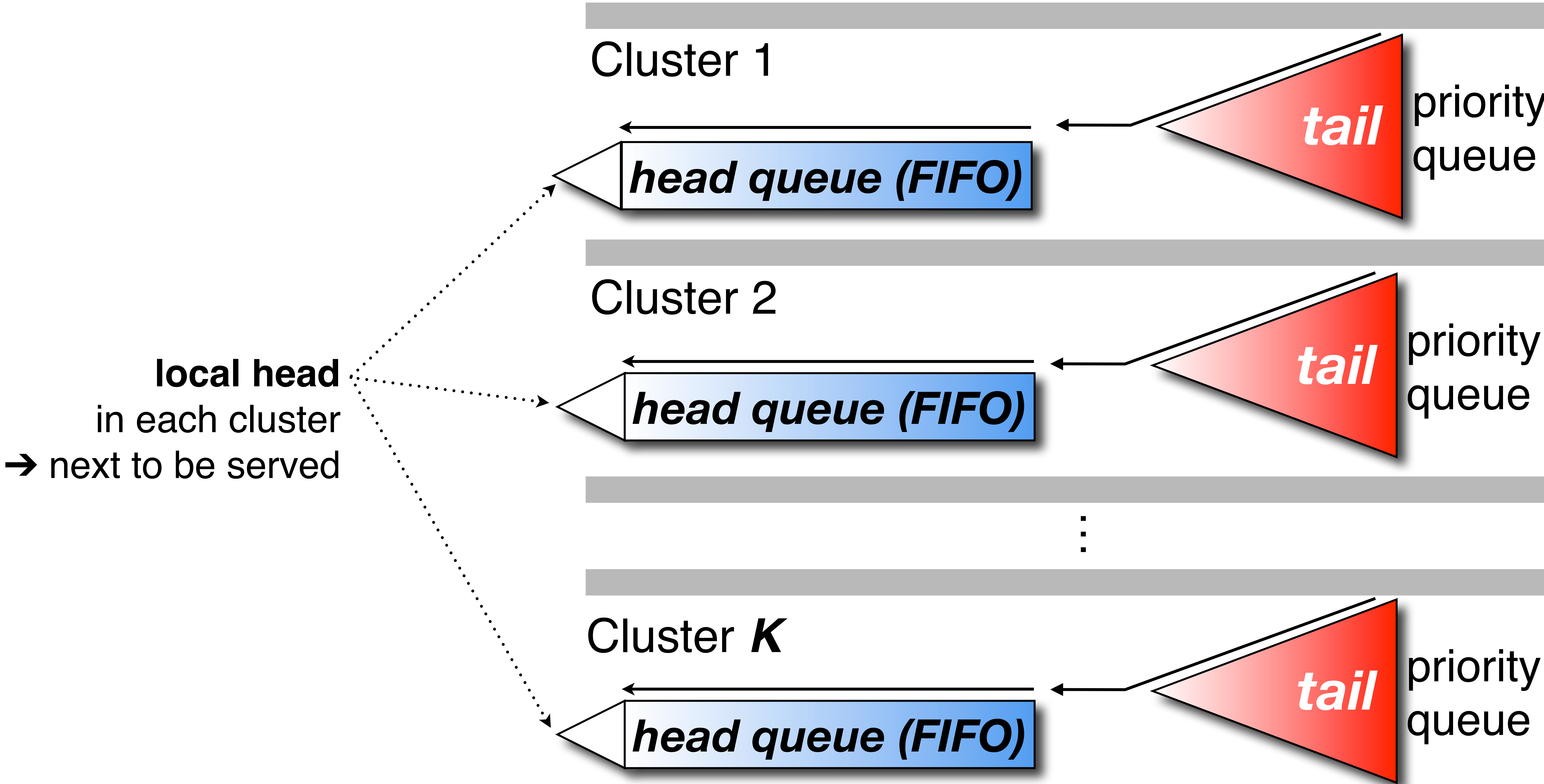
Bounded length: at most m_i **- 1** jobs in each **head queue**.

(m_i = number of cores in cluster)



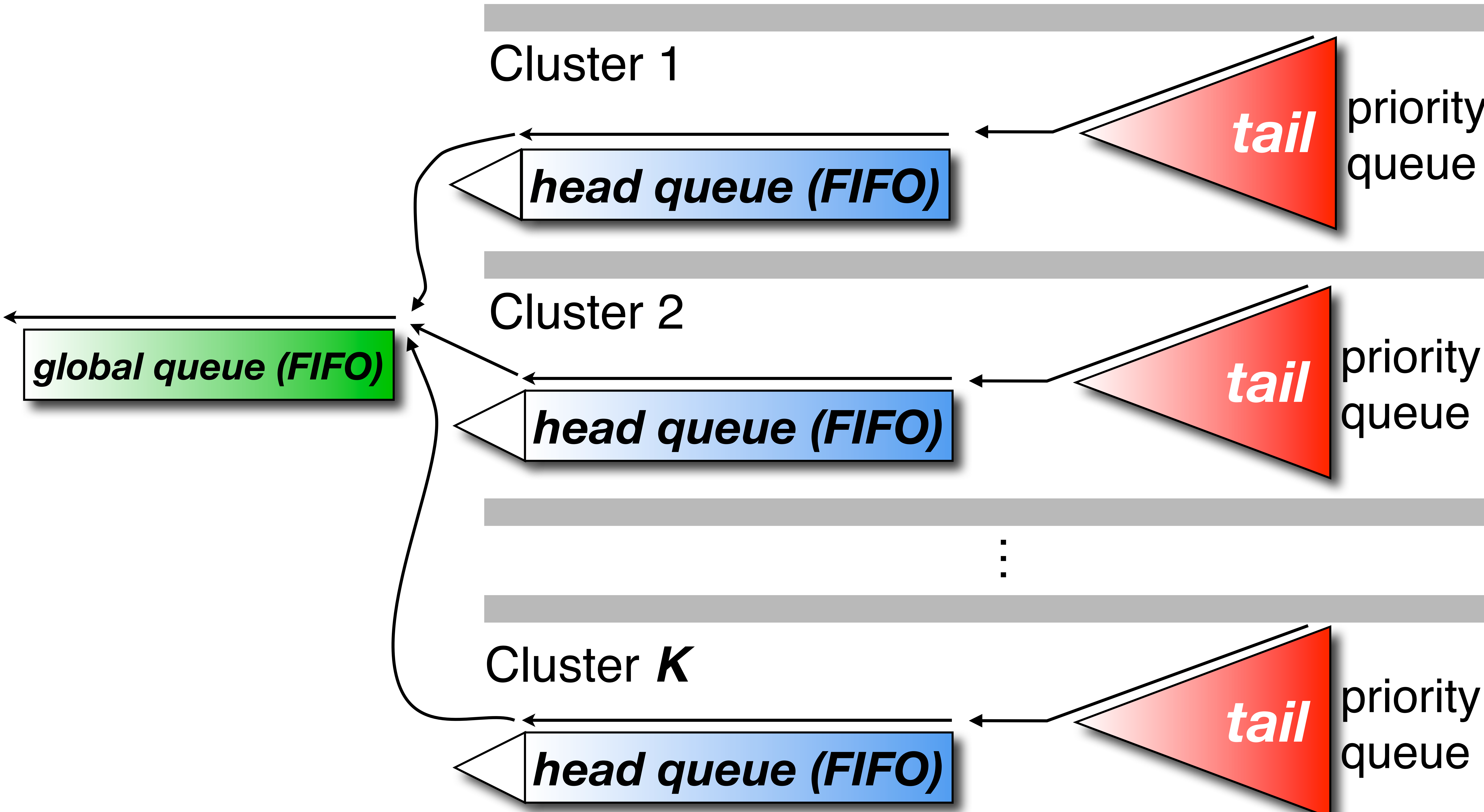
P2 & P3: Dealing With Unknown Contention

An unknown number of tasks may issue requests at arbitrary rates.



P2 & P3: Dealing With Unknown Contention

An unknown number of tasks may issue requests at arbitrary rates.

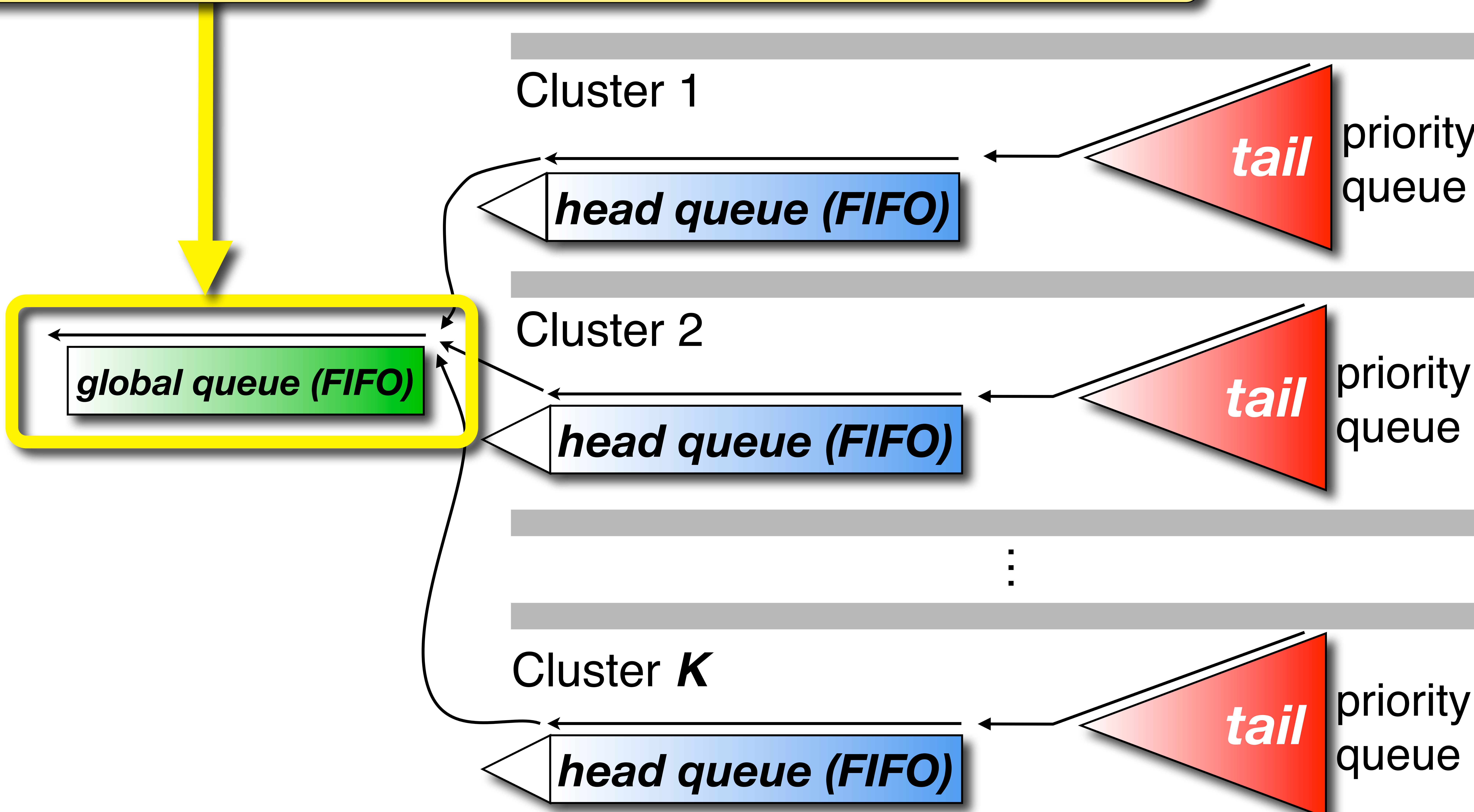


Bounded length: at most ***K* - *1*** jobs in **global queue**.

(*K* = *number of clusters in the system)*

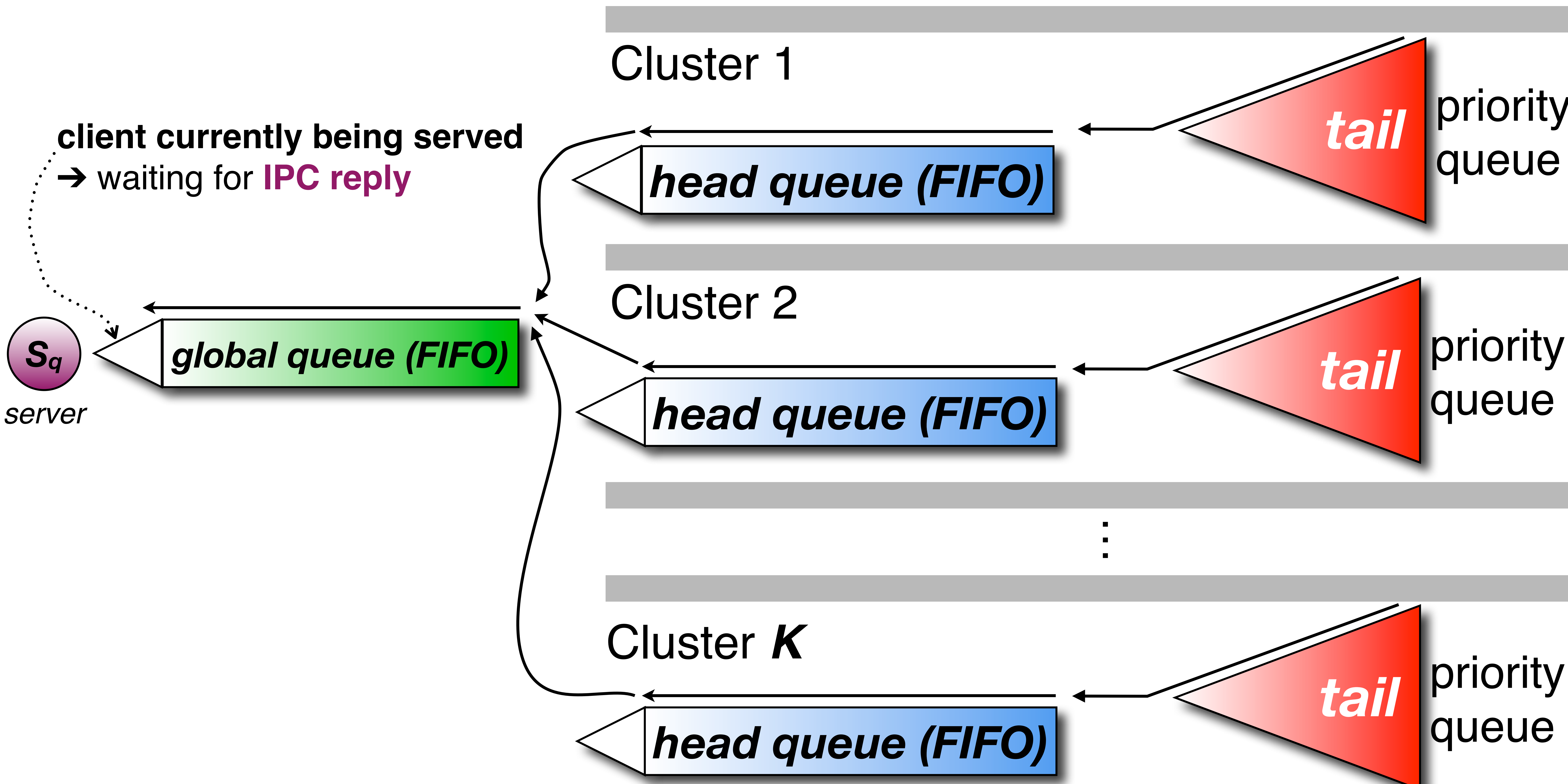
...ention

...ary rates.



P2 & P3: Dealing With Unknown Contention

An unknown number of tasks may issue requests at arbitrary rates.



P4: Dealing With Budget Exhaustion

Idling & bandwidth inheritance: a waiting client can run out of budget.

P4: Dealing With Budget Exhaustion

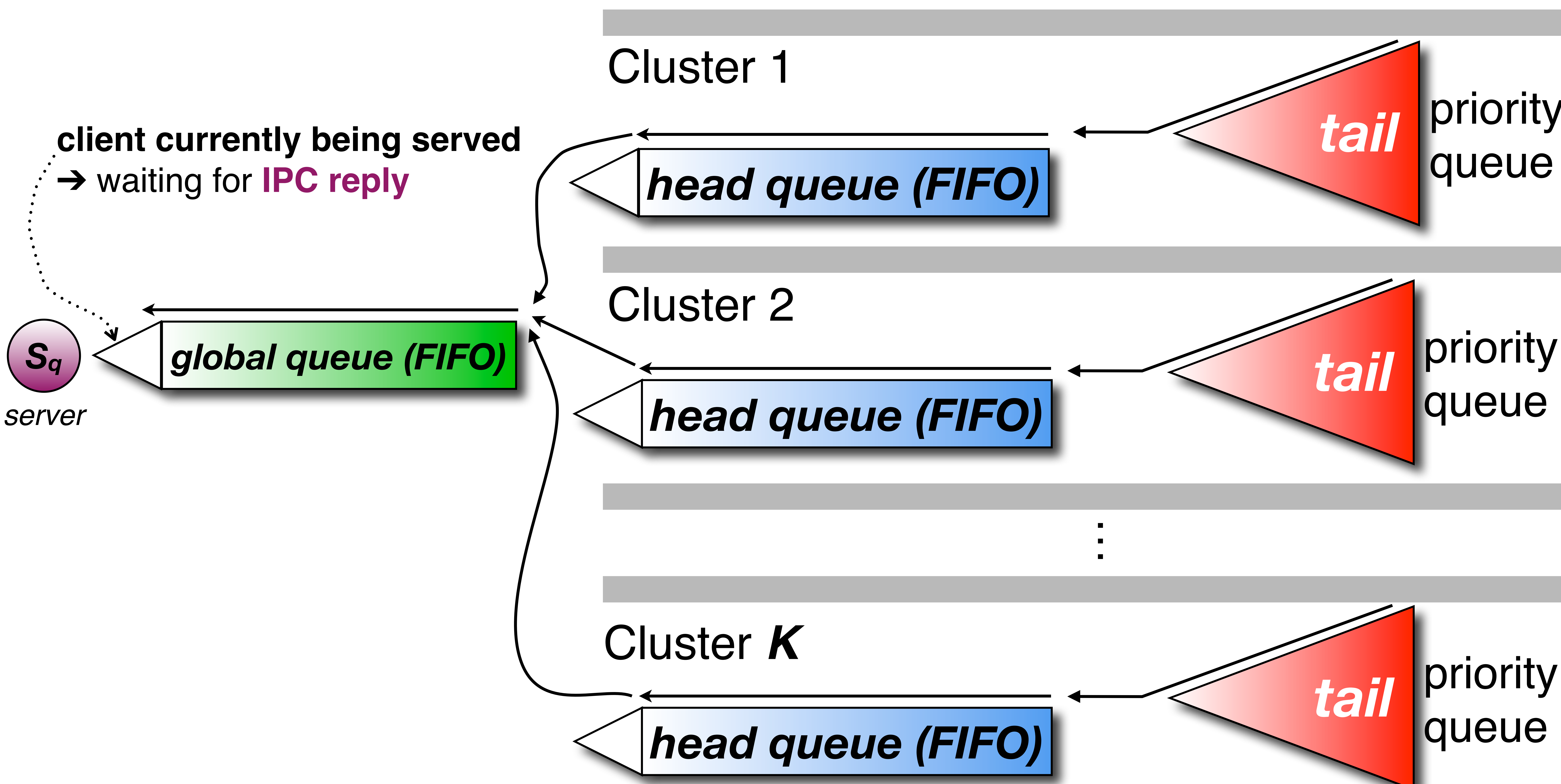
Idling & bandwidth inheritance: a waiting client can run out of budget.

> *With **predictable IPC delay**, a **correct** (high-criticality) task will have been provisioned to **never exceed its budget**.*

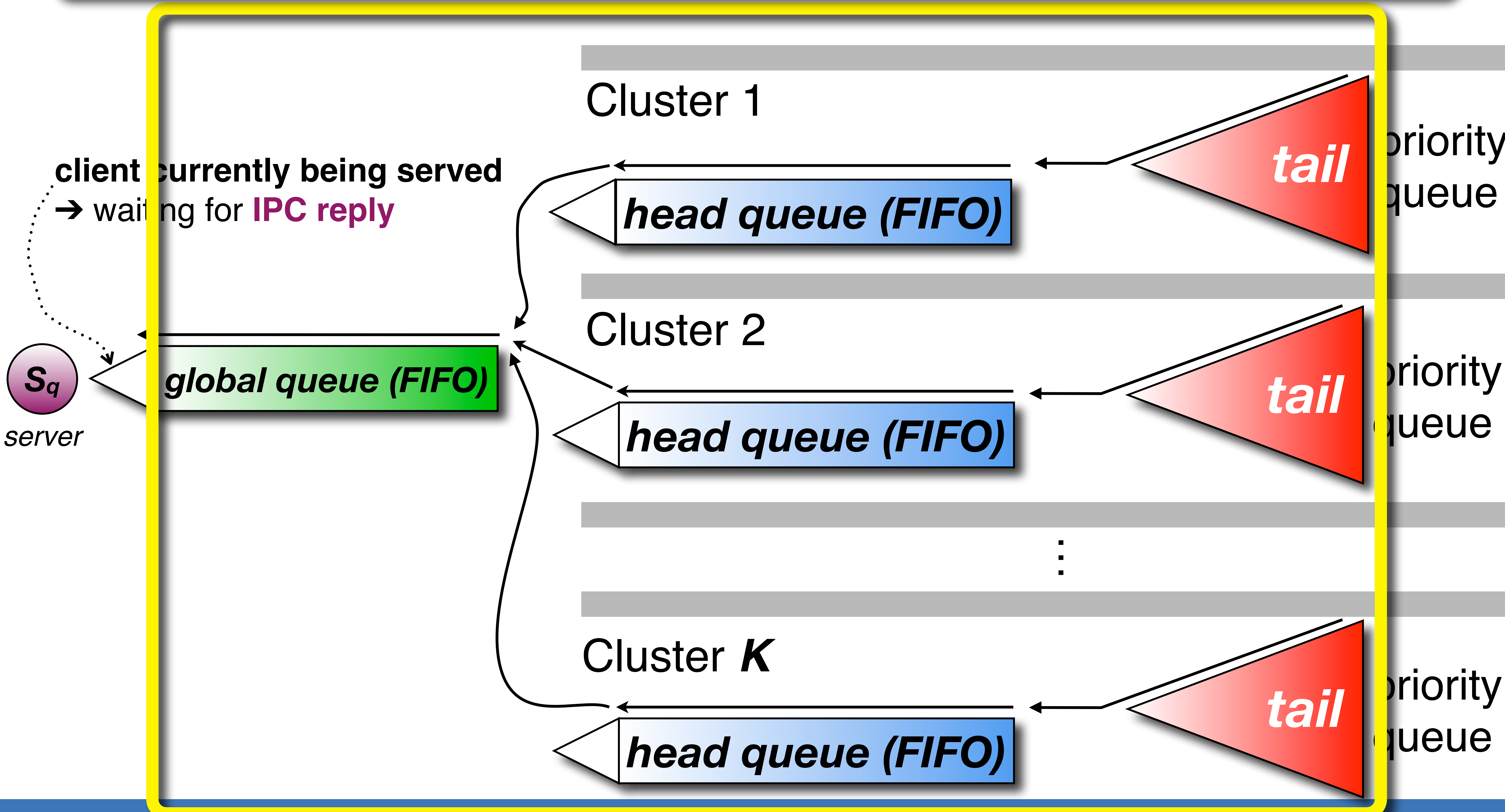
➔ *can **prune** clients from queue **when budget exhausted** & let them reissue requests after budget has been replenished*

P4: Dealing With Budget Exhaustion

Idling & bandwidth inheritance: a waiting client can run out of budget.

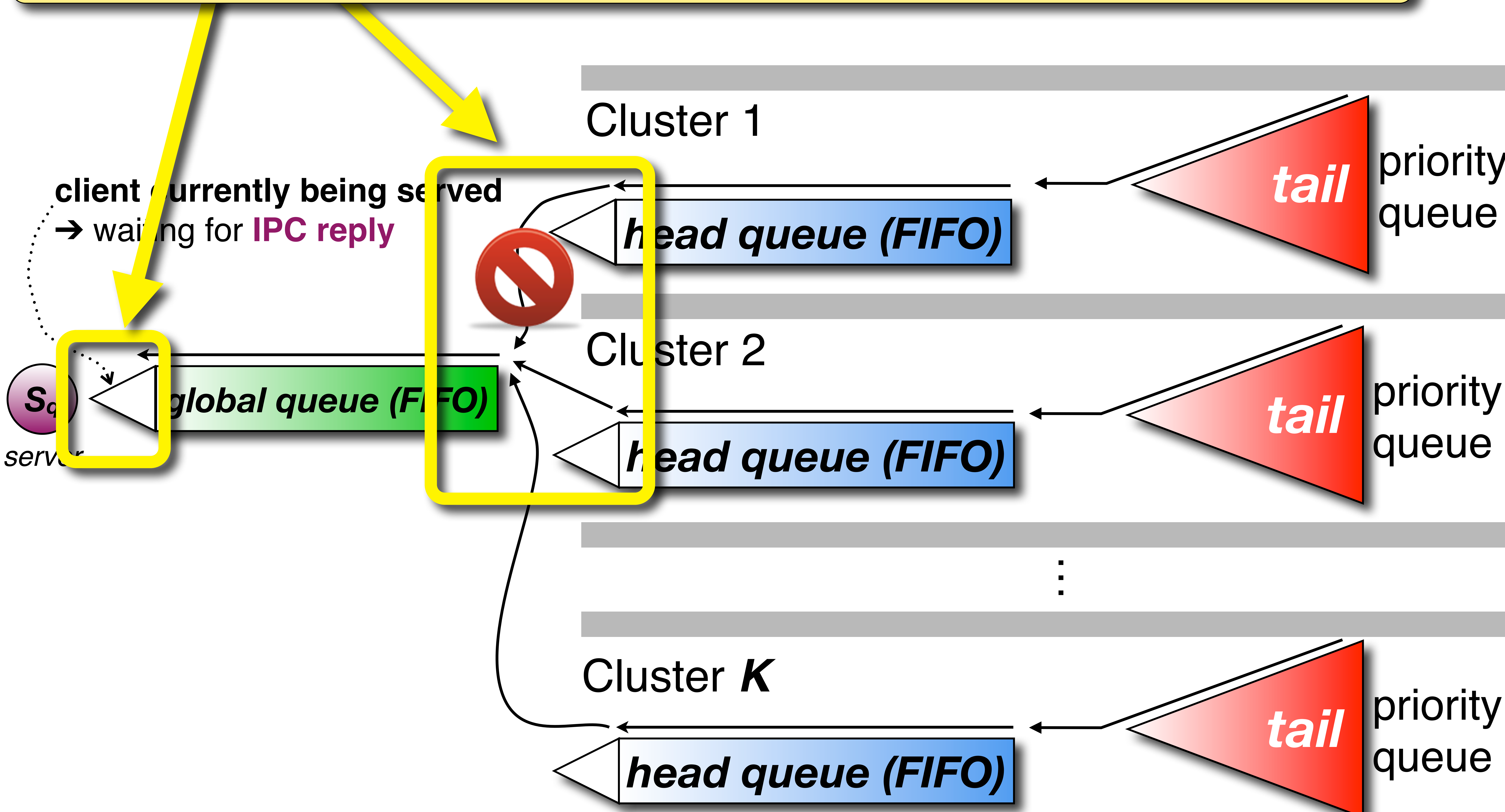


Easy case: client request not yet being served ➔ **simply dequeue**.



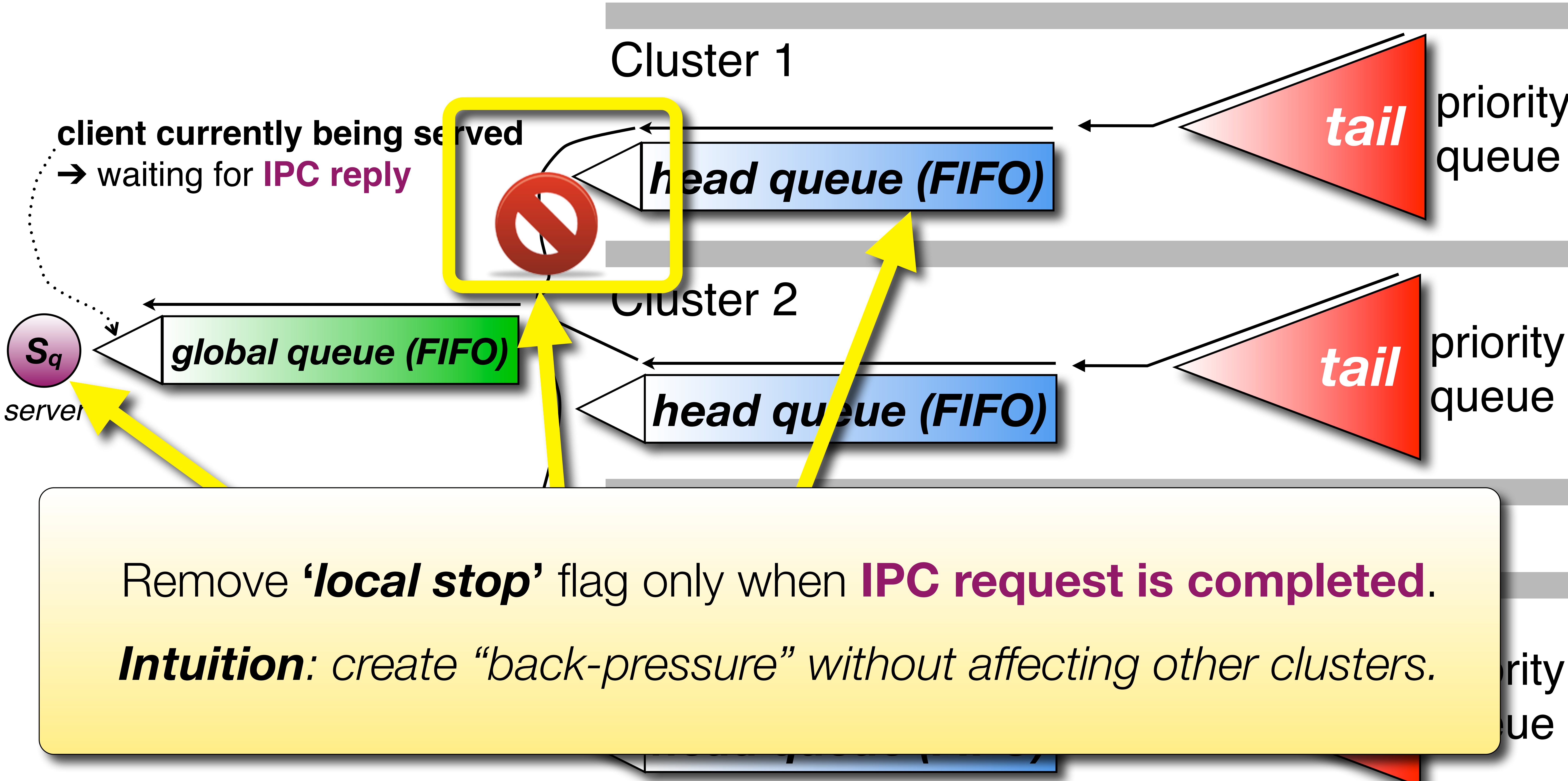
Tricky case: client request **is being served**.

➔ propagate **next task from head queue** to **local head,** but do **not** add to **global queue** yet! (***important for proof***)



P4: Dealing With Budget Exhaustion

Idling & bandwidth inheritance: a waiting client can run out of budget.

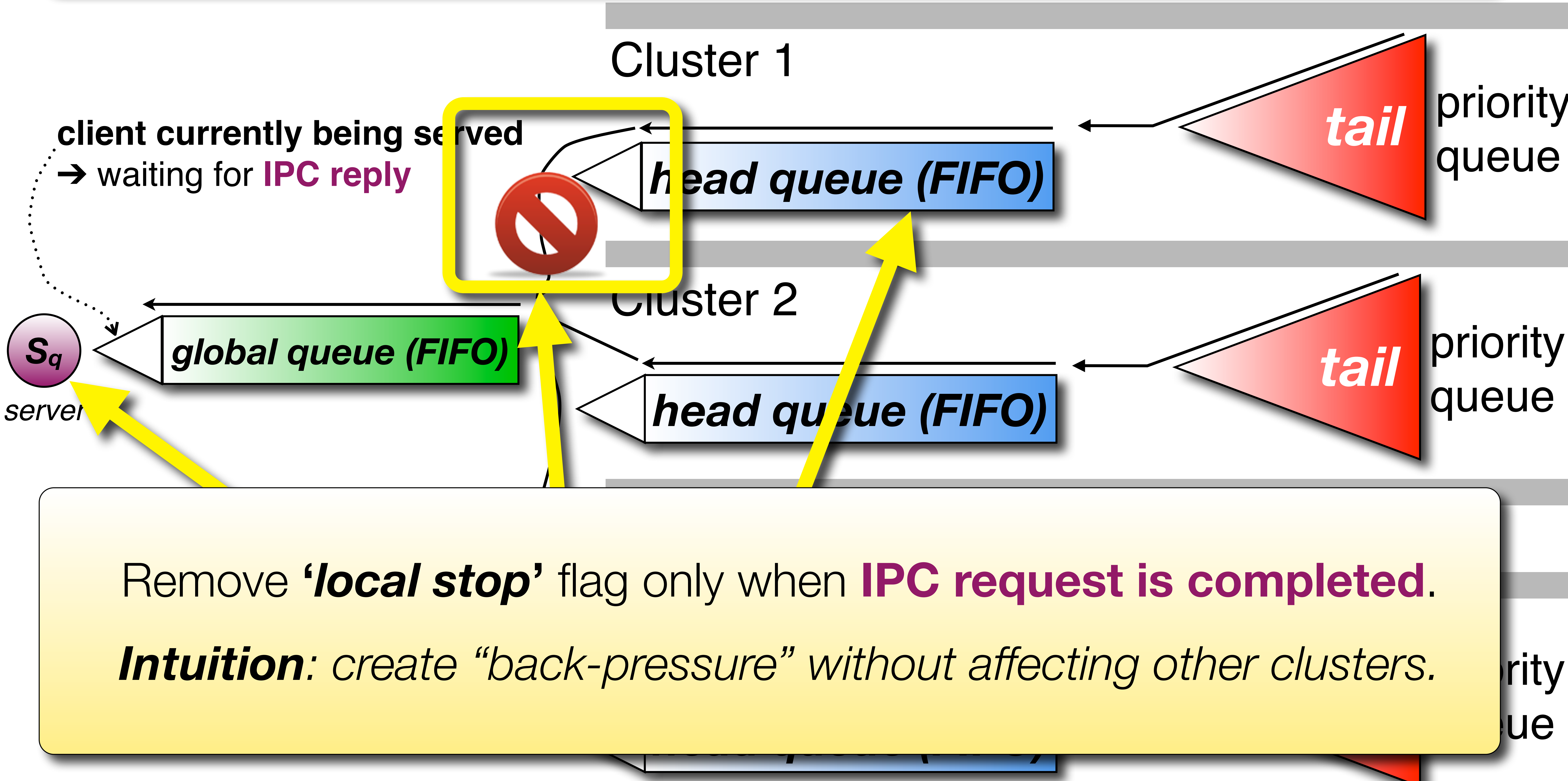


Remove **'*local stop*'** flag only when **IPC request is completed**.

Intuition: *create "back-pressure" without affecting other clusters.*

Added benefit: this mechanism allows us to deal with **budget-less best-effort tasks**.

(see paper for details)



Remove **'*local stop*'** flag only when **IPC request is completed**.

***Intuition**: create "back-pressure" without affecting other clusters.*

MC-IPC: Bandwidth Isolation Guarantee

MC-IPC = MBWI + OMIP Queue + Pruning + Best-Effort Tasks

MC-IPC: Bandwidth Isolation Guarantee

MC-IPC = MBWI + OMIP Queue + Pruning + Best-Effort Tasks

*If a client T_i does **not exhaust** its reservation's budget during a synchronous IPC invocation (= if not pruned),*

then T_i's IPC request to a server S_q is delayed by at most

$$2 \times m_k \times K$$

other requests ("delayed" = forced to expend budget).

m_k — number of cores in local cluster

K — number of clusters

Strict Temporal Isolation

The per-request **IPC delay bound** is **independent of any task parameters**.

No trust implied w.r.t. the **number** of tasks, request **frequencies**, **budgets** of other tasks, **which resources** any other task accesses, etc.

during a synchronous IPC invocation (= if not pruned),

then T_i's IPC request to a server S_q is delayed by at most

$$\boxed{2 \times m_k \times K}$$

other requests ("delayed" = forced to expend budget).

m_k — *number of cores in local cluster*

K — *number of clusters*

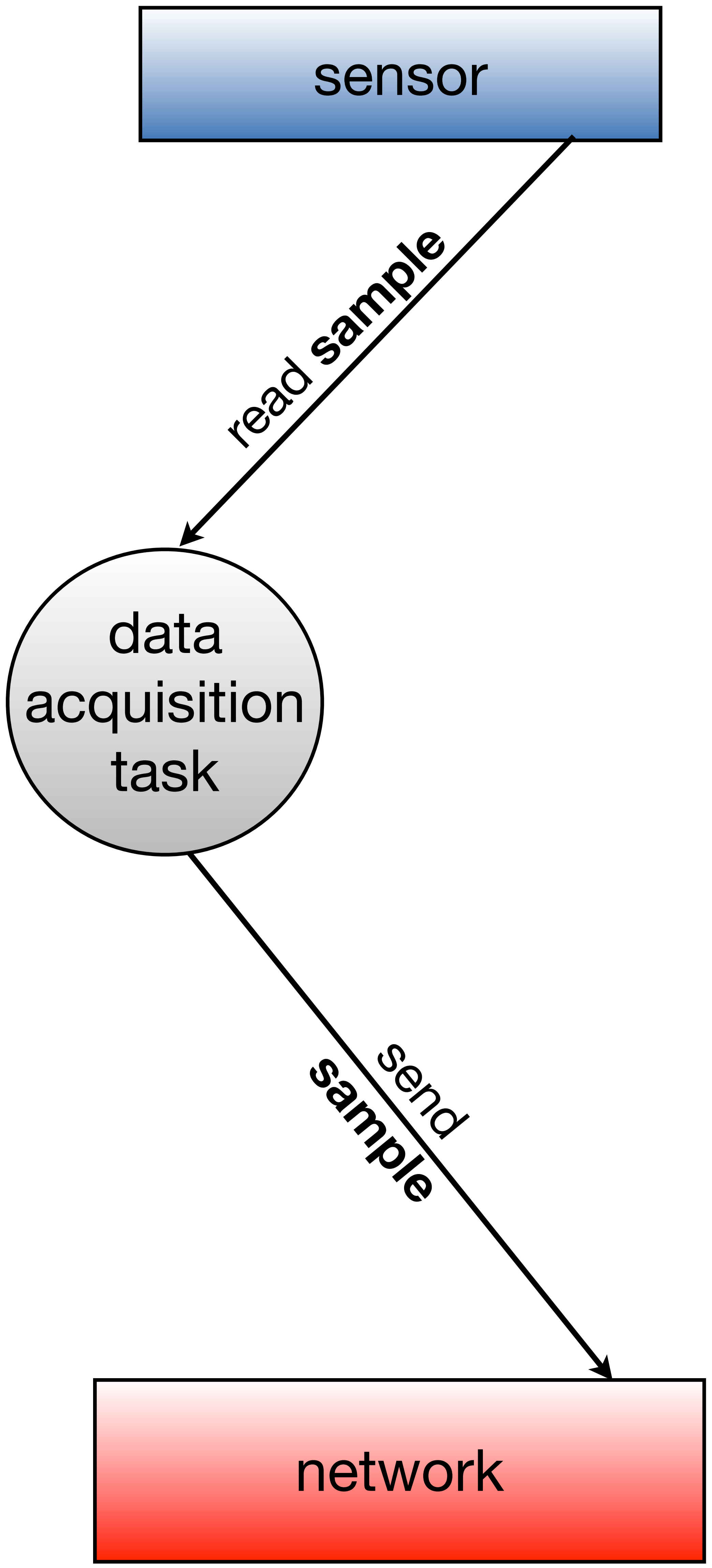
Evaluation: A Case Study

PRIO-IPC vs. FIFO-IPC vs. MC-IPC

Case Study: Access To Signing Service

*measurements of a **real** implementation on a **real** multicore system in a **plausible** scenario*

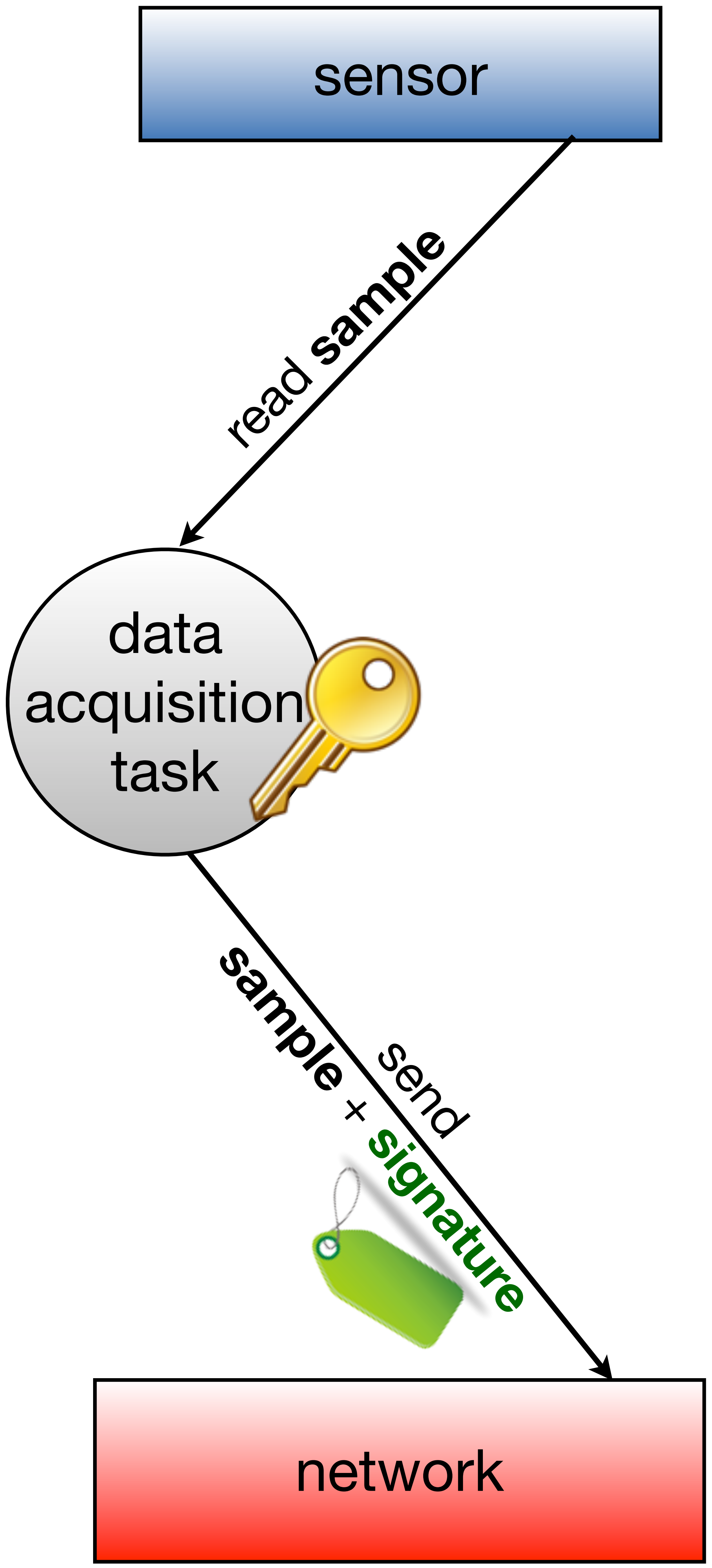
Case Study: Access To Signing Service



Data collection with high integrity requirements

- ➡ multiple **data sources** (sensors, …)
- ➡ forward collected samples for further processing over **untrusted, potentially compromised network**

Case Study: Access To Signing Service



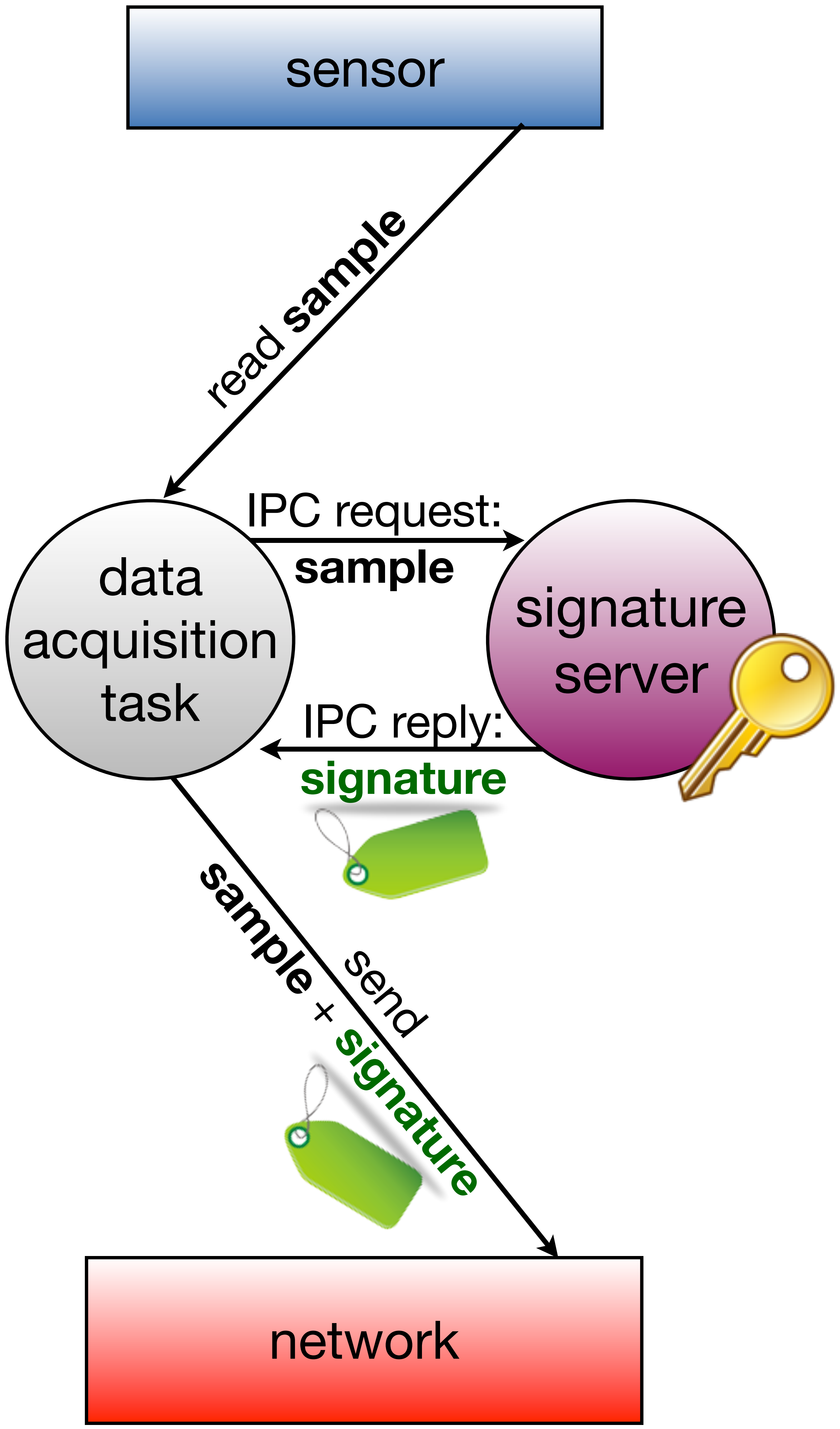
Data collection with high integrity requirements

- ➡ multiple **data sources** (sensors, …)
- ➡ forward collected samples for further processing over **untrusted, potentially compromised network**

To ensure integrity and to prevent playback attacks…

- ➡ …**timestamp** each sample
- ➡ …assign a **sequence number**
- ➡ …add **cryptographic signature**

Case Study: Access To Signing Service



Data collection with <u>high integrity requirements</u>

- ➡ multiple **data sources** (sensors, …)
- ➡ forward collected samples for further processing over **untrusted, potentially compromised network**

To ensure integrity and to prevent playback attacks…

- ➡ …**timestamp** each sample
- ➡ …assign a **sequence number**
- ➡ …add **cryptographic signature**

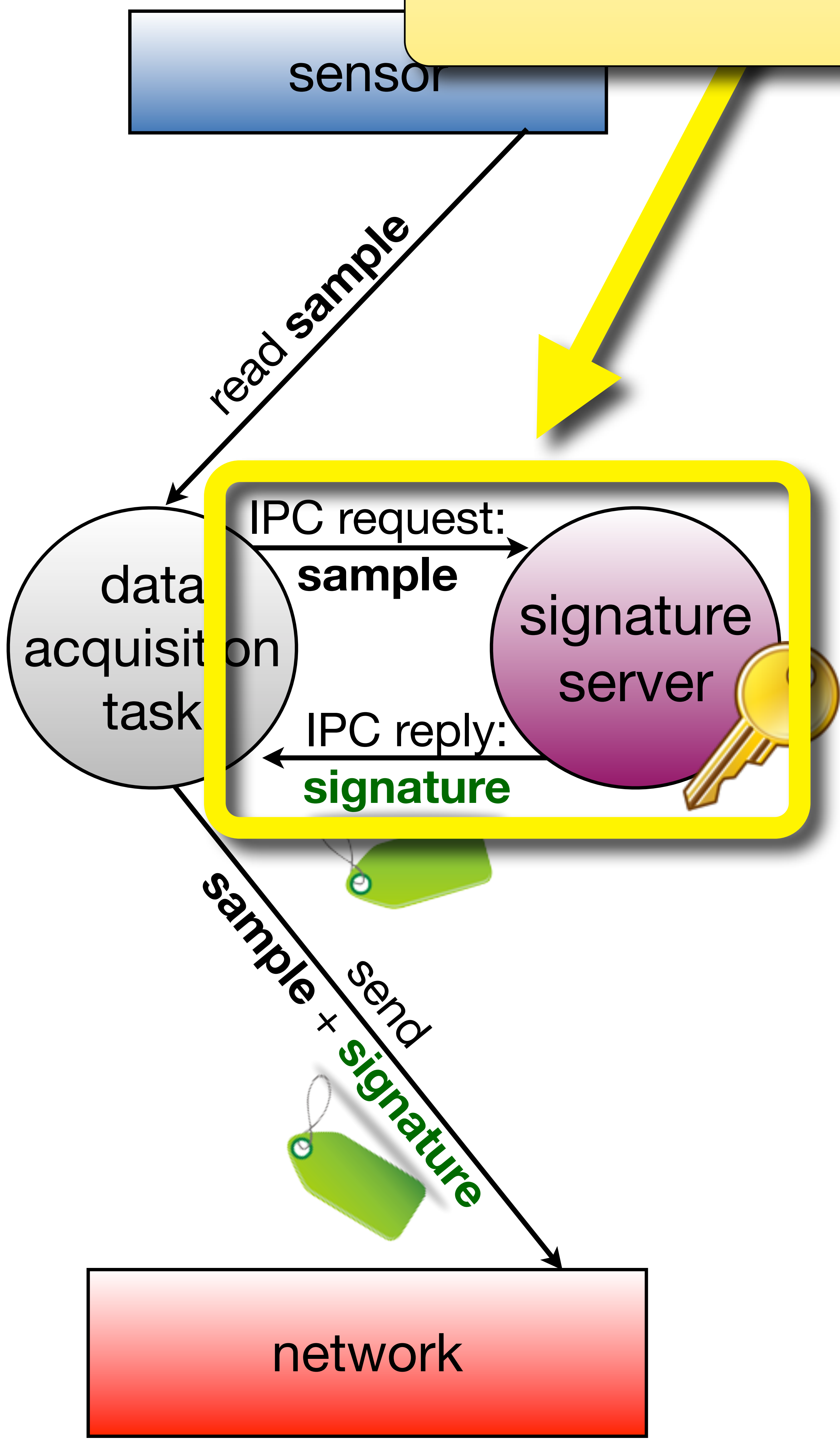
To prevent leakage of cryptographic key material…

- ➡ key material accessible only to **signature server**

The signature server is a shared resource.

If there are multiple data acquisition tasks,
what is the maximum IPC delay?

Case … vice



…irements

- ➡ multiple **data sources** (sensors, …)
- ➡ forward collected samples for further processing over **untrusted, potentially compromised network**

To ensure integrity and to prevent playback attacks…

- ➡ …**timestamp** each sample
- ➡ …assign a **sequence number**
- ➡ …add **cryptographic signature**

To prevent leakage of cryptographic key material…

- ➡ key material accessible only to **signature server**

Case Study: Implementation

Reservation-Based Scheduler

- implemented in LITMUS^RT
- table-driven, sporadic, polling reservations…
- with EDF and FP scheduling

LITMUS^RT
Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

IPC System Calls

- added to LITMUS^RT

Signature Server

- RSA w/ 2048bit keys
- **max. request length**: **≈2ms**

OpenSSL

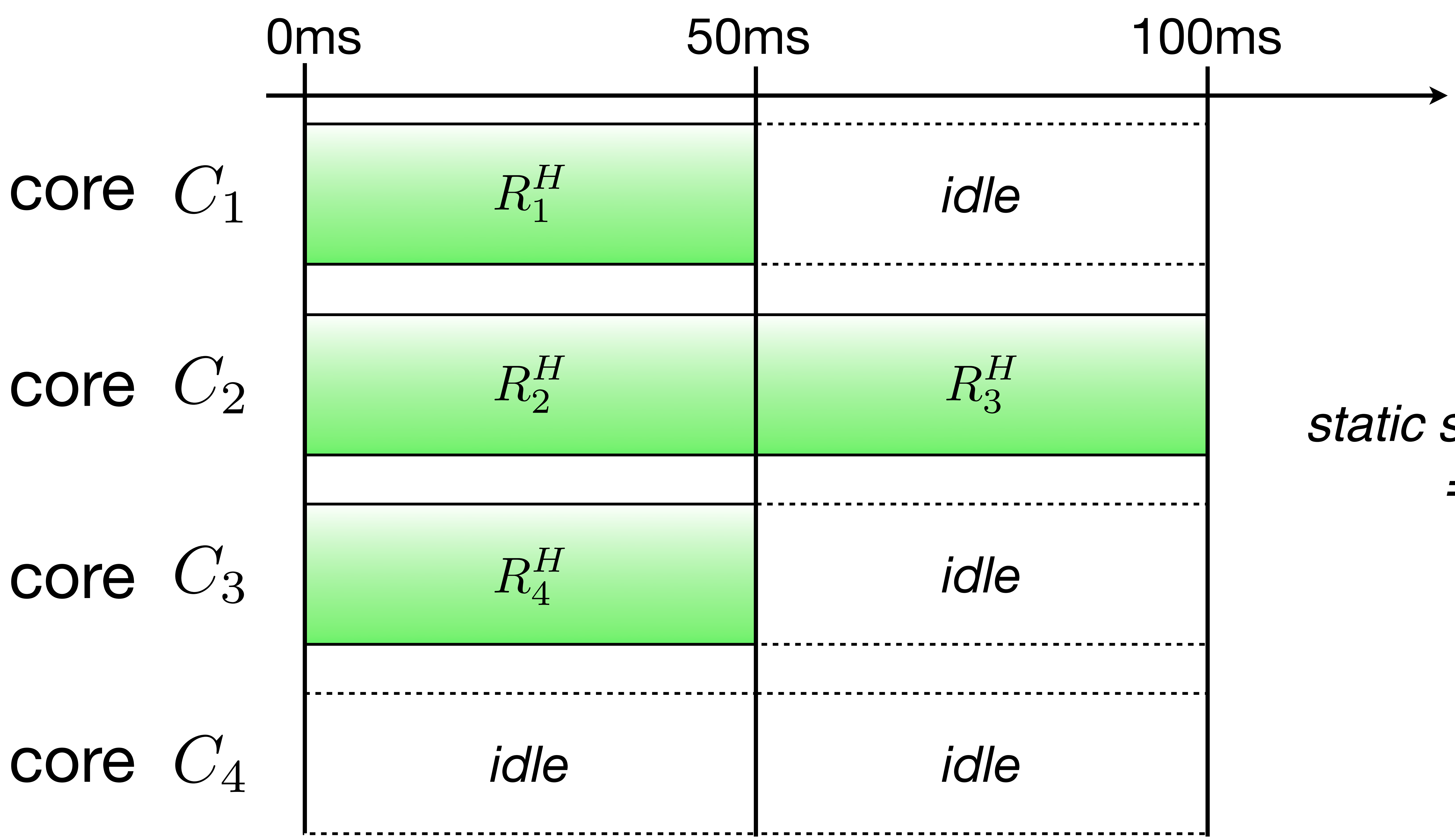
Platform

- 4 Xeon E5-2665 (2.4 Ghz) cores

Case Study: Setup on a 4-Core System

4 high-criticality tasks
➡ provisioned with **table-driven static reservations**

10 low-criticality tasks
➡ provisioned with simple EDF-scheduled **polling reservations**



static schedule length = 100 ms

Low-criticality polling reservations assigned to cores ***C₃*** and ***C₄***;

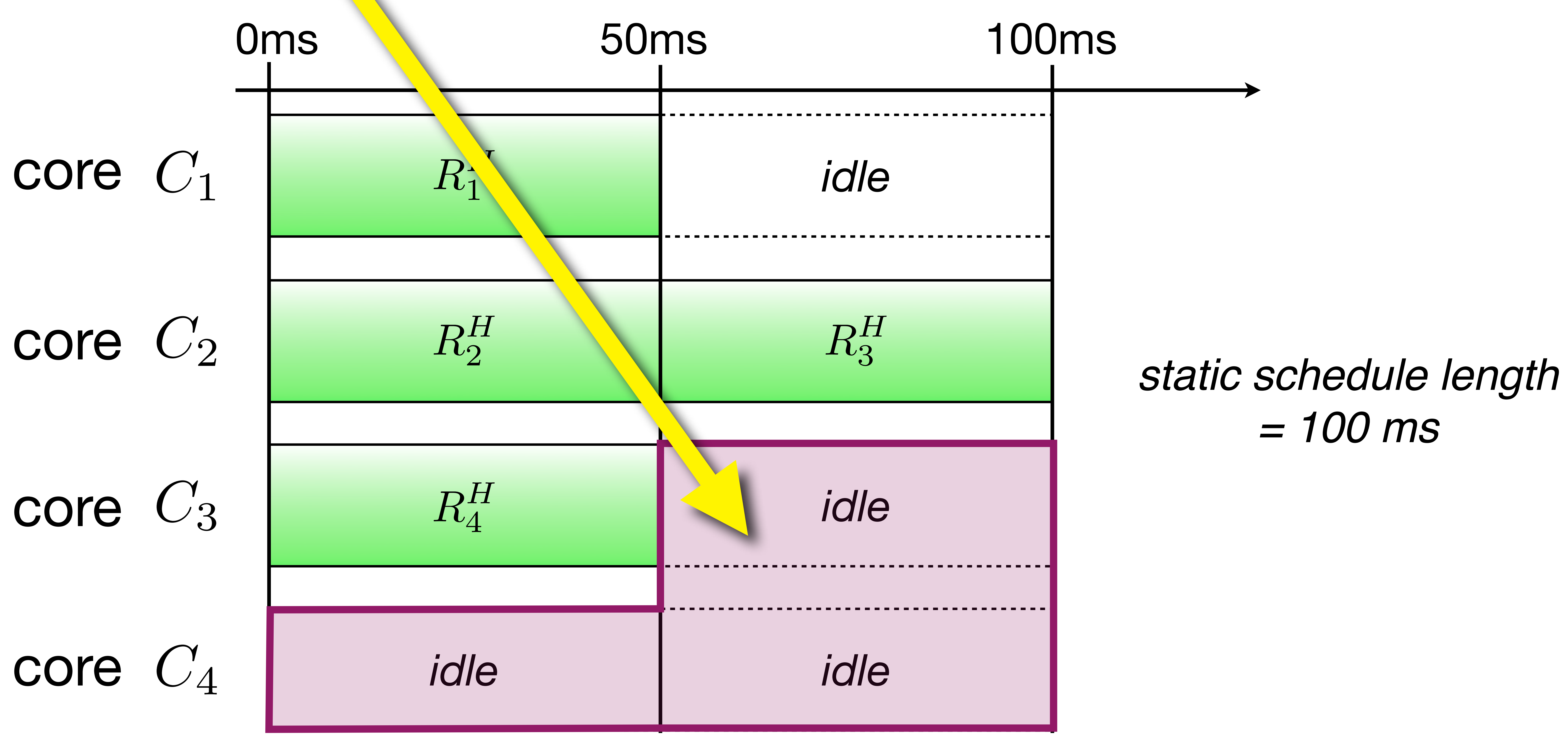
executed only when table-driven reservation is idle.

4 high-criticality tasks

➡ provisioned with **table-driven static reservations**

10 low-criticality tasks

➡ provisioned with simple EDF-scheduled **polling reservations**

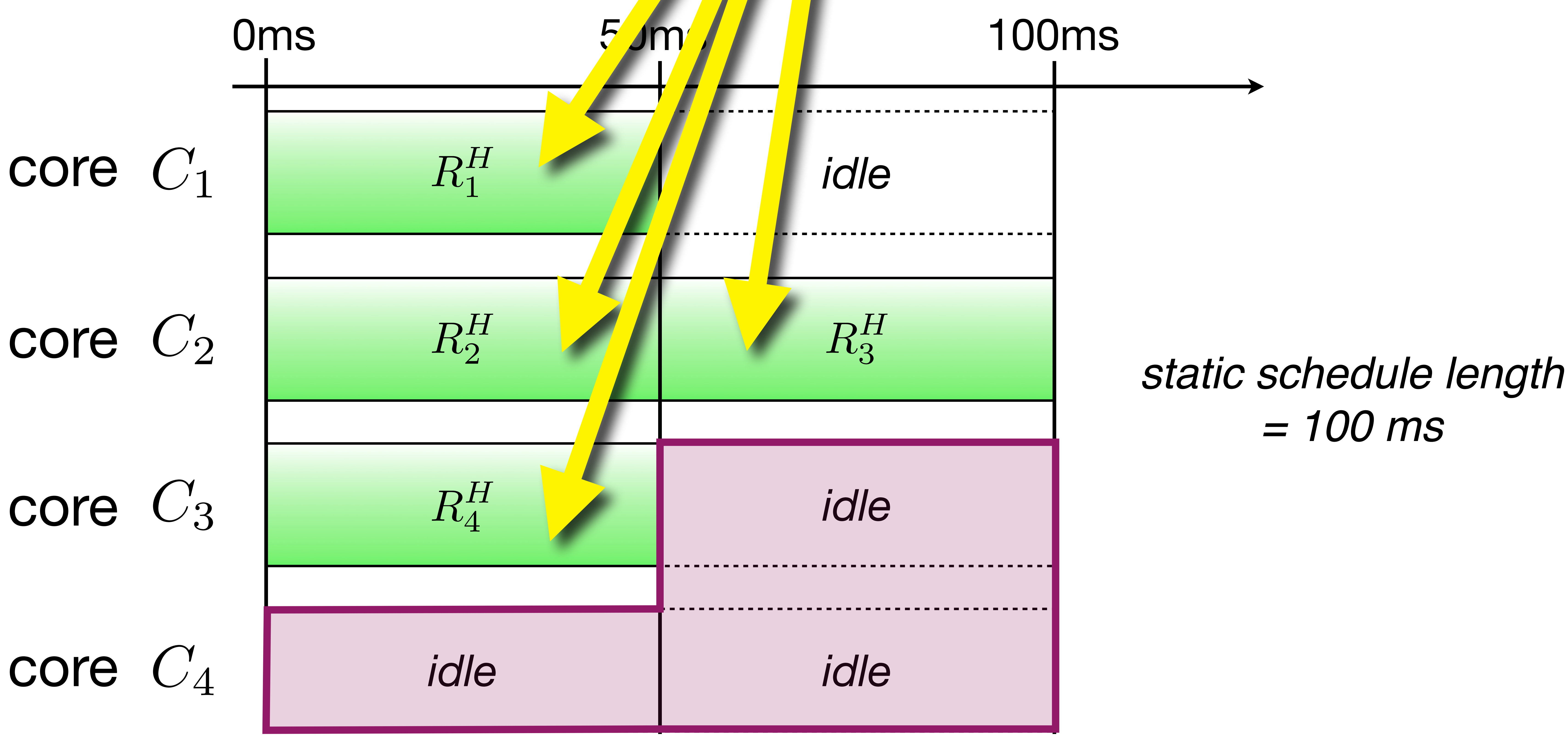


Case Study: Setup on a 4-Core System

4 high-criticality tasks
➡ provisioned with **table-driven**

10 low-criticality tasks
➡ provisioned with simple EDF-scheduled **polling reservations**

One **high-criticality** task each
*with **period = deadline =100ms***



…e System

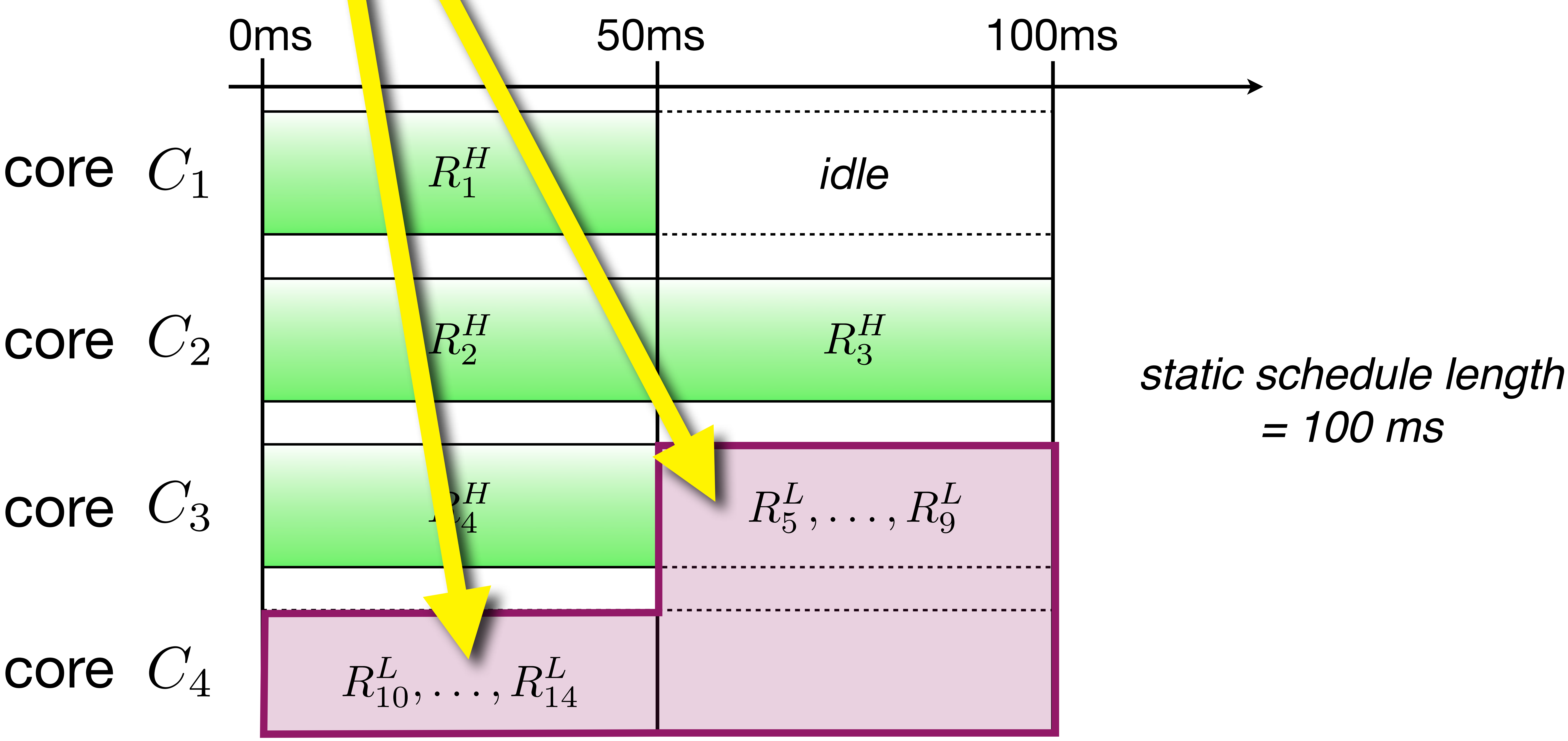
10 low-criticality tasks

one per EDF-based low-criticality reservation

*five per core, with **budget = 20ms** and*
periods = deadline ∈ {100, 120, 250, 500, 1000}ms

10 low-criticality tasks

➡ provisioned with simple EDF-scheduled **polling reservations**



Case Study: Experiment

8 phases of execution, 60 seconds each = 480 seconds

- ➡ 1 phase = 60 seconds of **normal execution**
- ➡ 7 phases = 420 seconds of different **failure modes**
- ➡ measured: **IPC delay** experienced by each task

Case Study: Experiment

8 phases of execution, 60 seconds each = 480 seconds

- ➡ 1 phase = 60 seconds of **normal execution**
- ➡ 7 phases = 420 seconds of different **failure modes**
- ➡ <u>measured</u>: **IPC delay** experienced by each task

Three IPC protocols

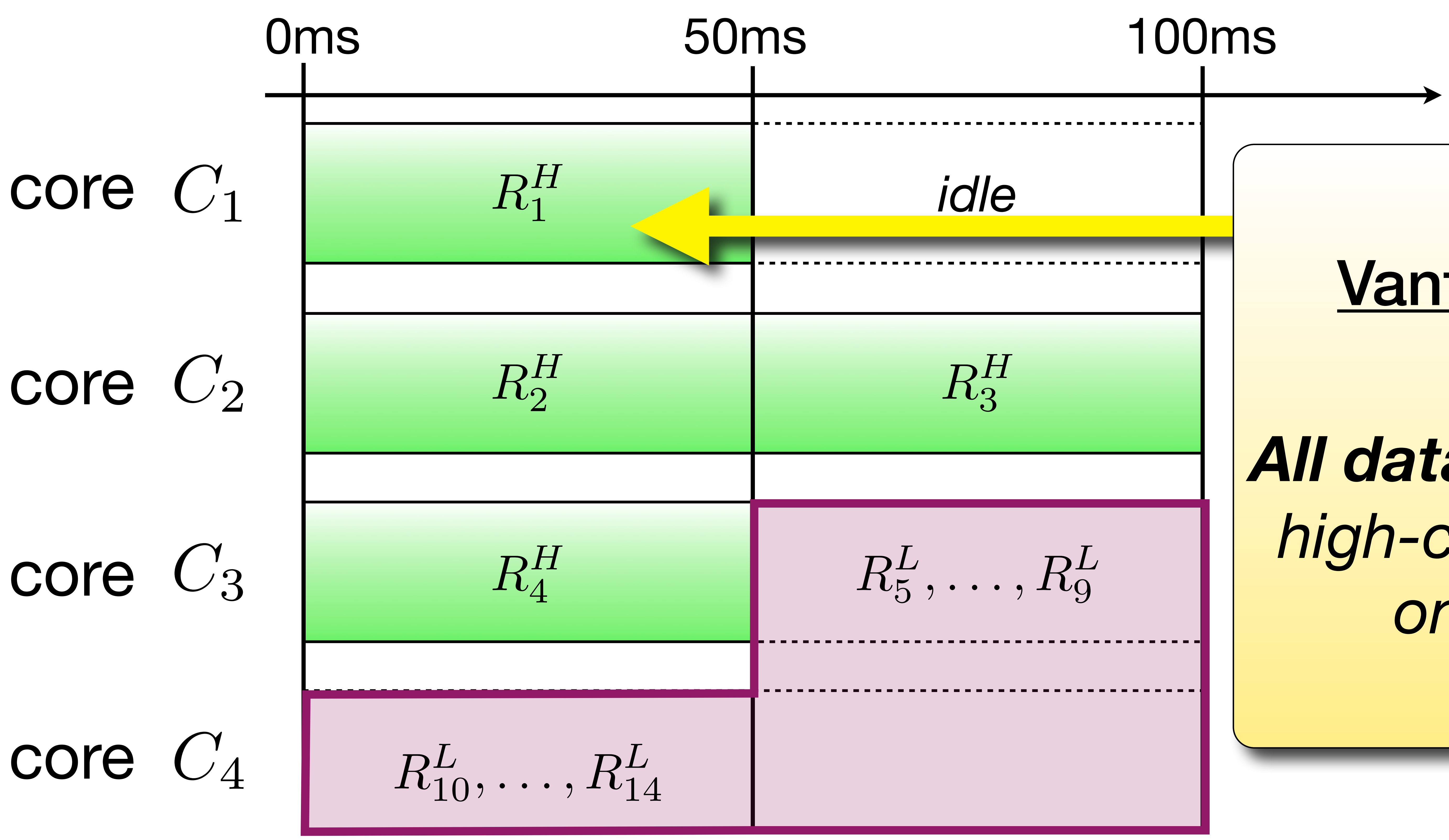
- ➡ <u>MC-IPC</u> (**this paper**)
- ➡ <u>FIFO-IPC</u>
  - serve requests strictly in FIFO order
  - *like Multiprocessor Bandwidth Inheritance Protocol (MBWI)*
- ➡ <u>PRIO-IPC</u>
  - order requests by priority of reservation
  - *like most microkernels (e.g., L4/Fiasco)*
- ➡ 10 runs each

MBWI: D. Faggioli, G. Lipari, and T. Cucinotta, "Analysis and implementation of the multiprocessor bandwidth inheritance protocol," Real-Time Systems, vol. 48, no. 6, pp. 789–825, 2012.

Case Study: Experiment

8 phases of execution, 60 seconds each = 480 seconds

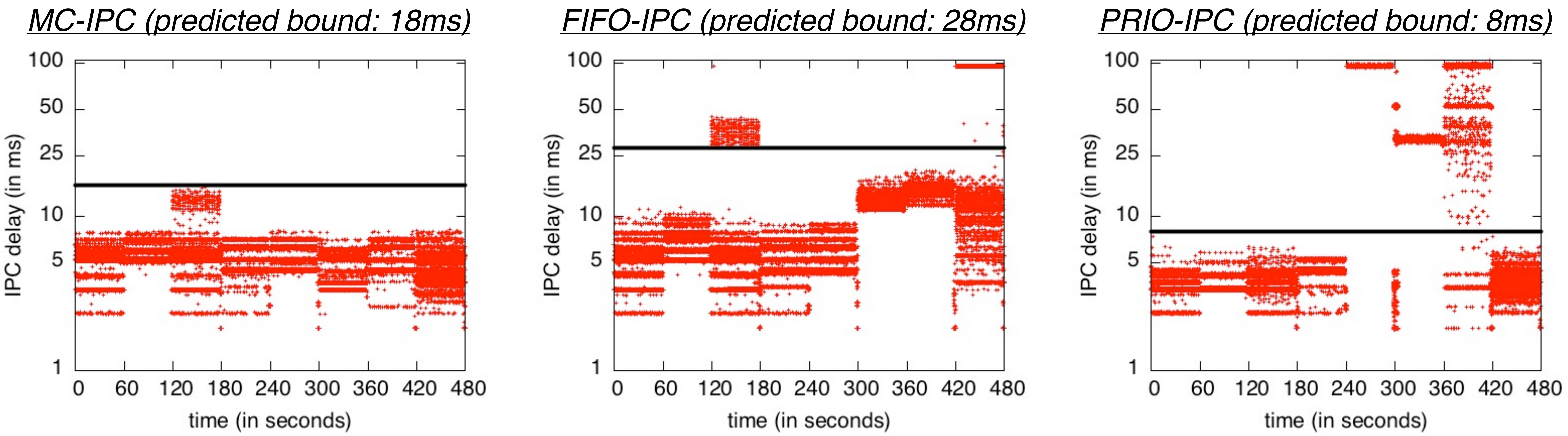
- ➡ 1 phase = 60 seconds of **normal execution**
- ➡ 7 phases = 420 seconds of different **failure modes**
- ➡ measured: **IPC delay** experienced by each task



Vantage Point

***All data** reported for high-criticality task on **core 1**.*

Case Study: Results Overview



8 phases of execution, 60 seconds each = 480 seconds

➡ <u>measured & shown</u>: **IPC delay** experienced by jobs of **task on core 1**

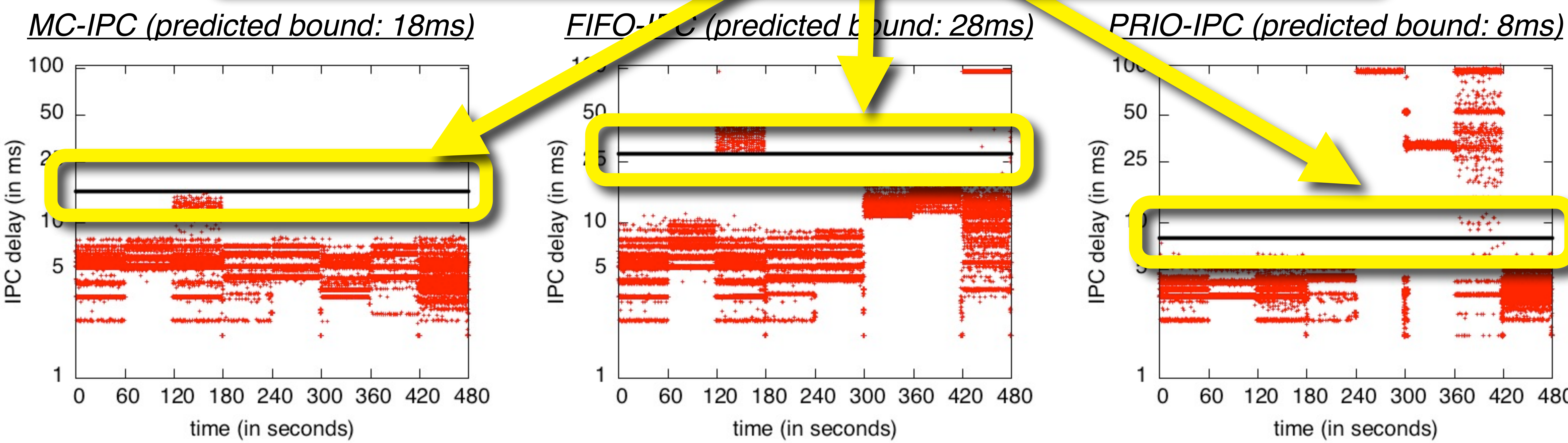
Comparison: measured IPC delay vs. <u>predicted maximum</u>

➡ All three IPC protocols permit ***a priori* IPC delay bounds**

➡ …but they react differently **to task failures / unexpected behavior**.

thick black line = **predicted maximum delay**

Based on RT analysis & specified task set parameters.



8 phases of execution, 60 seconds each = 480 seconds

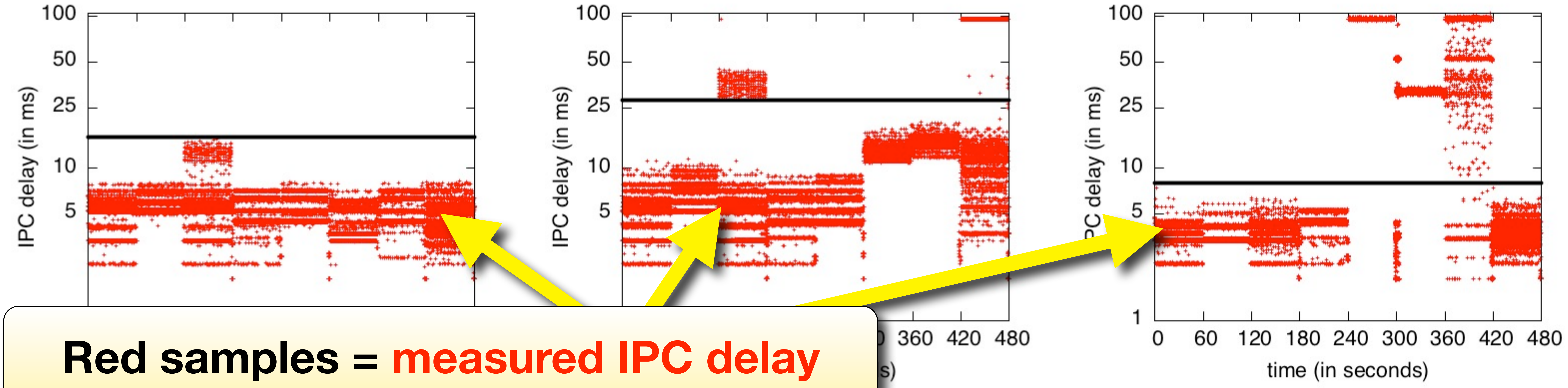
➡ measured & shown: **IPC delay** experienced by jobs of **task on core 1**

Comparison: measured IPC delay vs. predicted maximum

➡ All three IPC protocols permit ***a priori* IPC delay bounds**

➡ …but they react differently **to task failures / unexpected behavior**.

Case Study: Results Overview



Red samples = measured IPC delay
One sample per job of task on core 1.

8 phases of execution, 60 seconds each = 480 seconds

➡ <u>measured & shown</u>: **IPC delay** experienced by jobs of **task on core 1**

Comparison: measured IPC delay vs. <u>predicted maximum</u>

➡ All three IPC protocols permit ***a priori* IPC delay bounds**

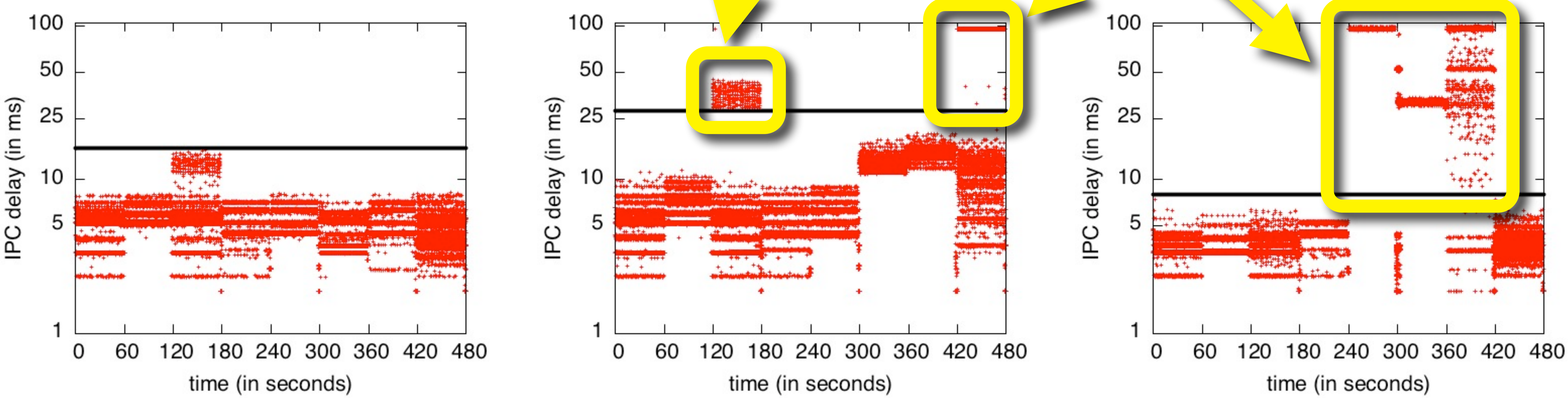
➡ …but they react differently **to task failures / unexpected behavior**.

Case

measured IPC delay > predicted maximum delay

→ *unpredictable IPC delays in case of failures!*

MC-IPC (predicted bound: 18ms) FIFO-IPC (p edicted bound: 28ms) PRIO-IP (predicted bound: 8ms)



8 phases of execution, 60 seconds each = 480 seconds

➡ measured & shown: **IPC delay** experienced by jobs of **task on core 1**

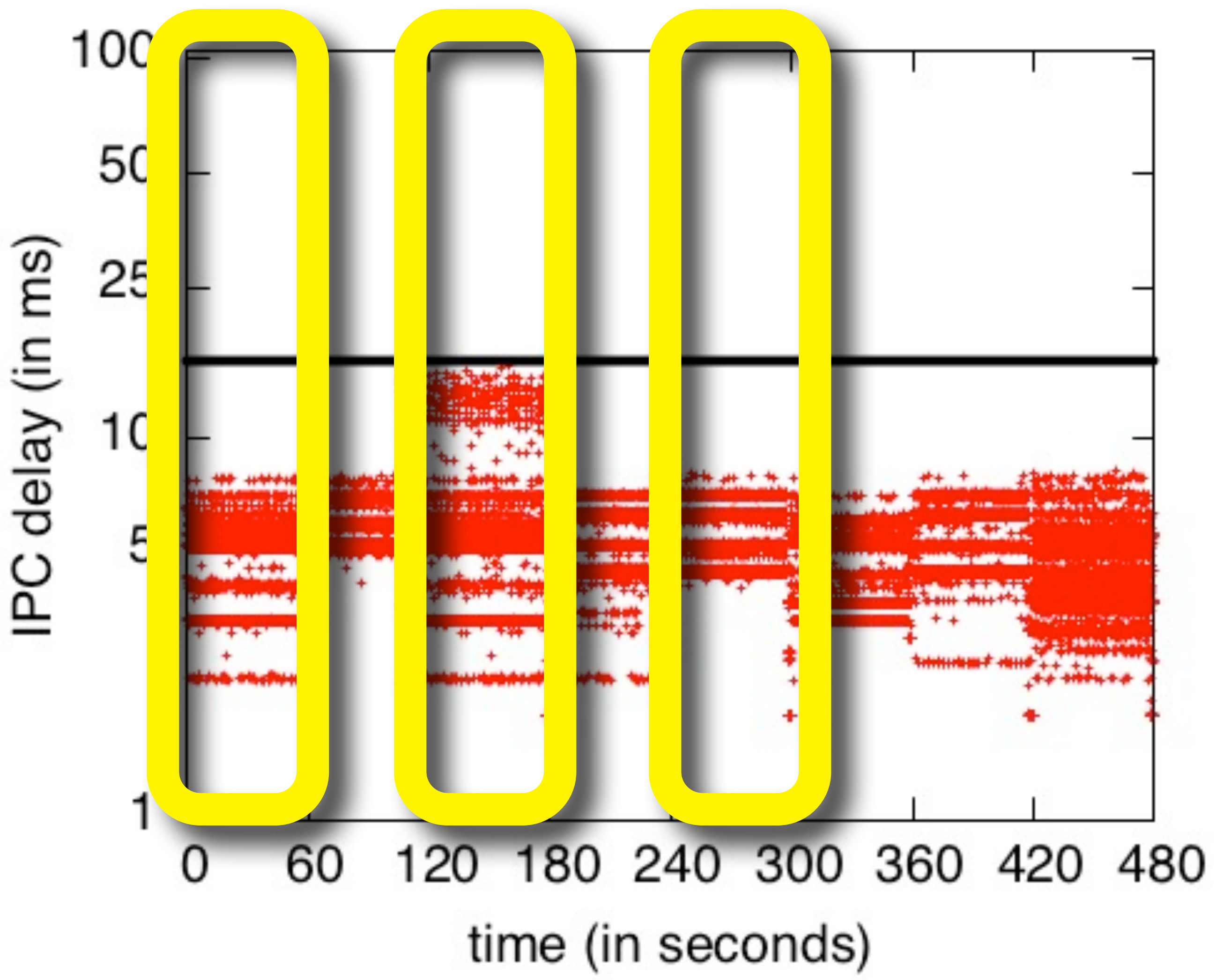
Comparison: measured IPC delay vs. predicted maximum

➡ All three IPC protocols permit ***a priori* IPC delay bounds**

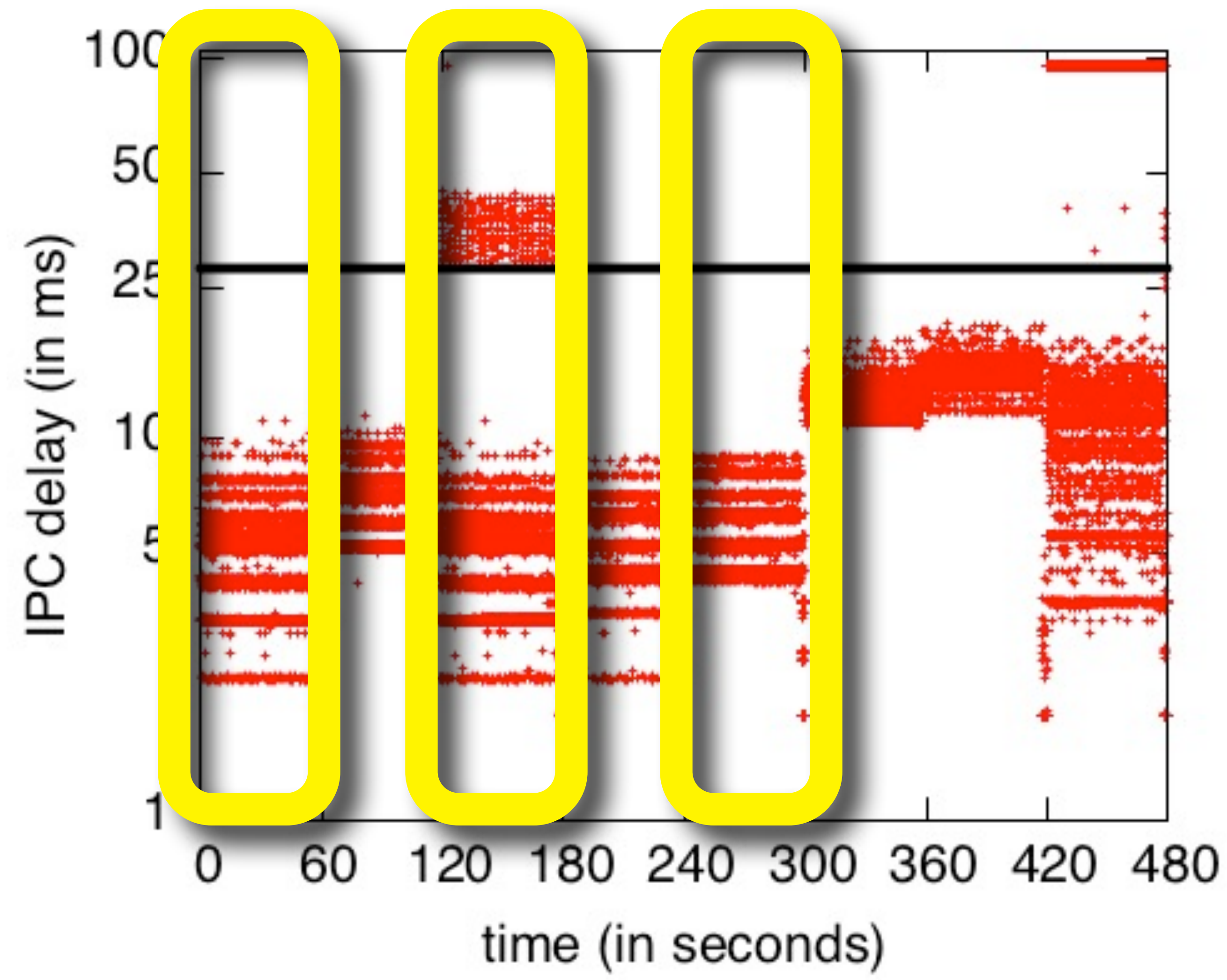
➡ …but they react differently **to task failures / unexpected behavior**.

Case Study: Results Overview

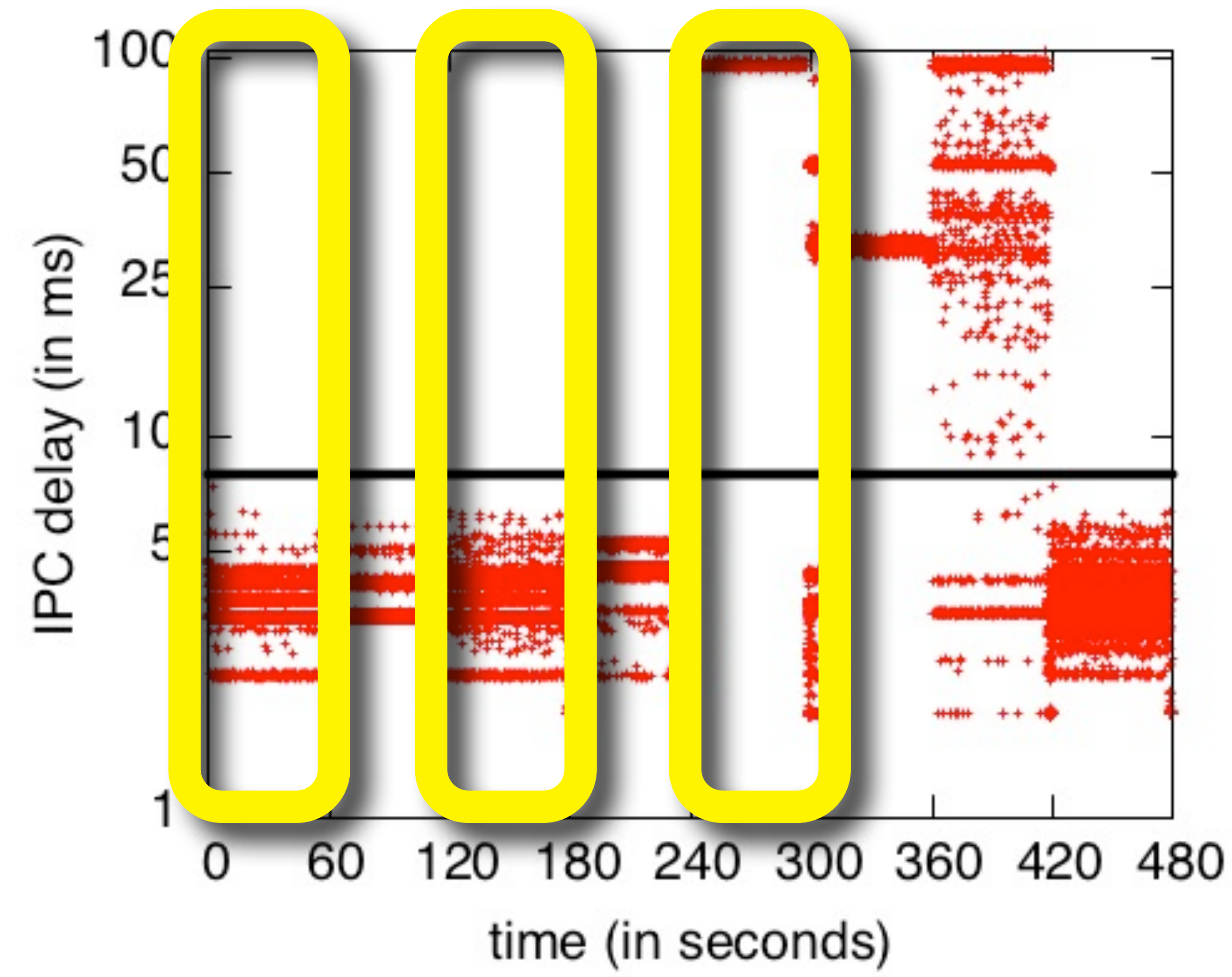
MC-IPC (predicted bound: 18ms)



FIFO-IPC (predicted bound: 28ms)



PRIO-IPC (predicted bound: 8ms)



8 phases of execution, 60 seconds each = 480 seconds

➡ measured & shown: **IPC delay** experienced by jobs of **task on core 1**

In this talk: 3 select phases.

(See paper for discussion of all 8 phases.)

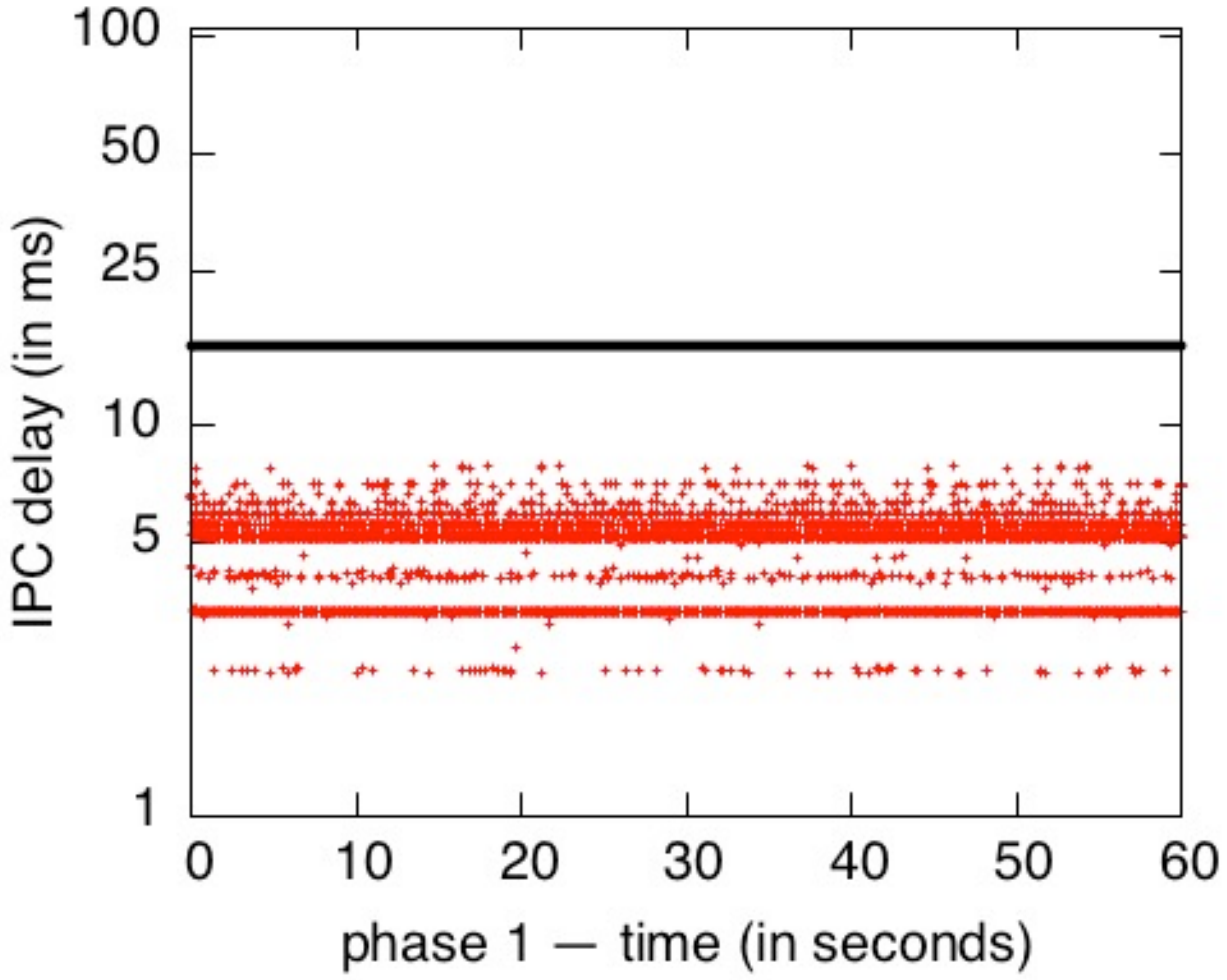
Phase 1: Normal Operation

Reality check:

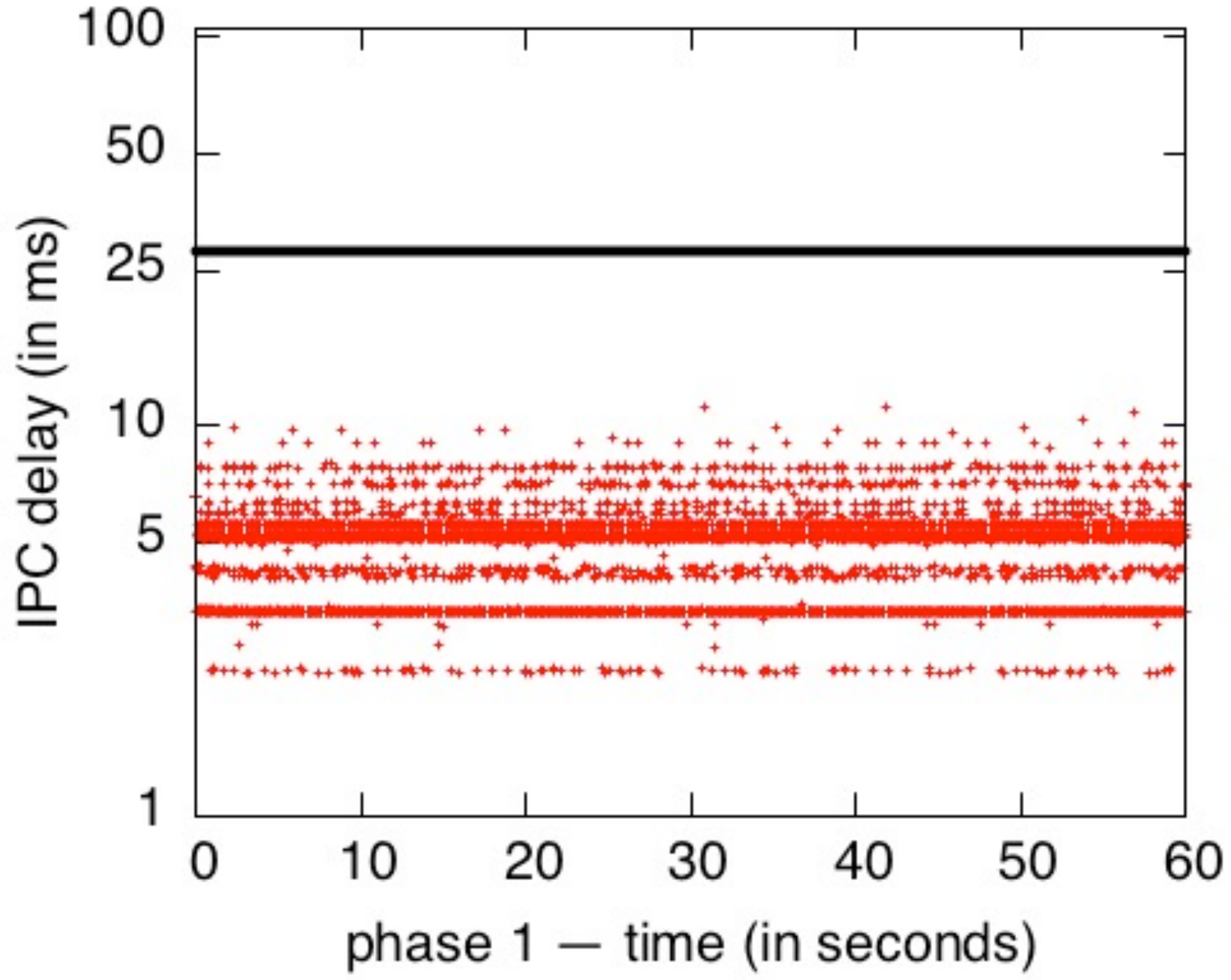
all tasks behave ***exactly as designed*** */* ***as assumed.***

Phase 1: Normal Operation

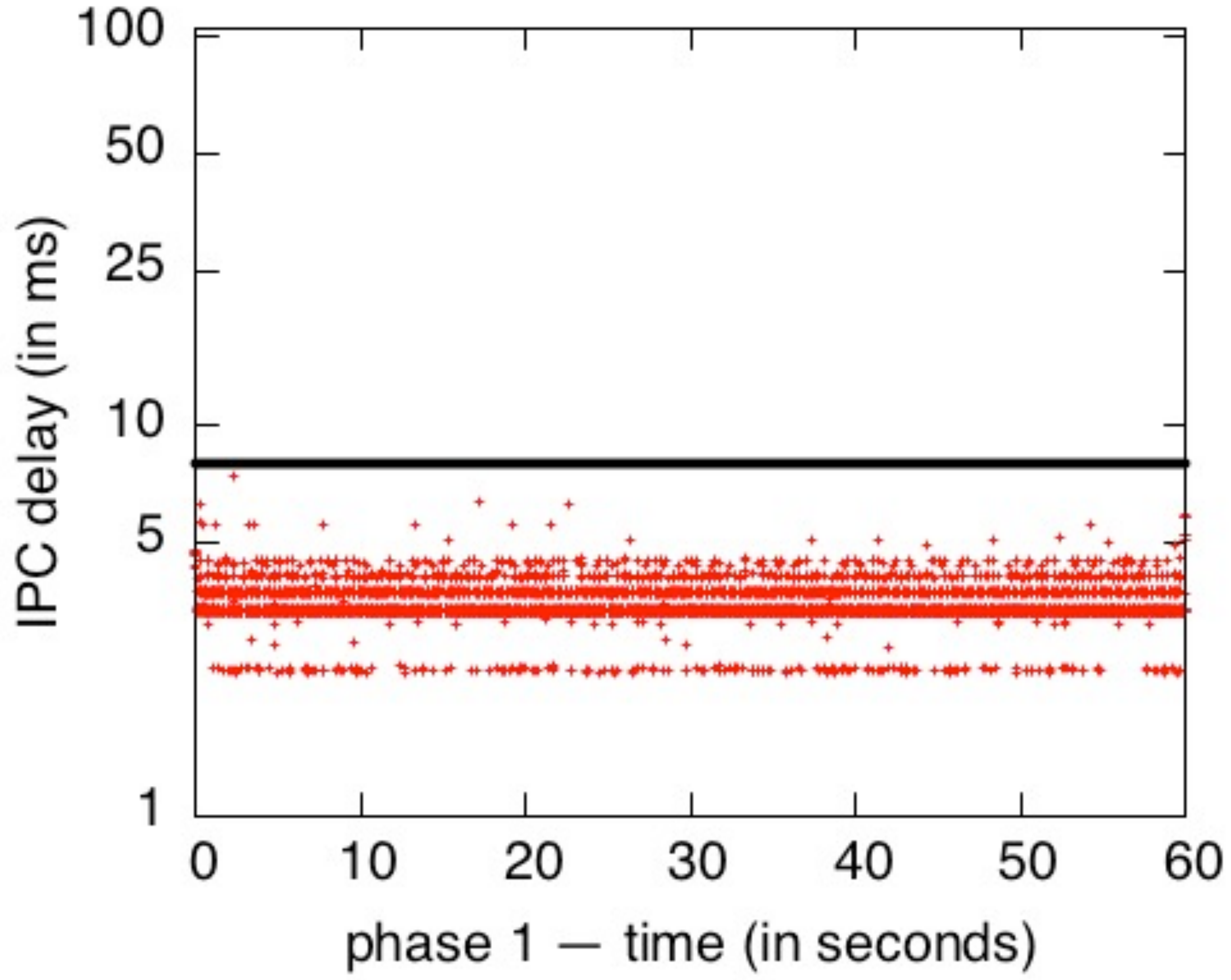
MC-IPC (predicted bound: 18ms)



FIFO-IPC (predicted bound: 28ms)



PRIO-IPC (predicted bound: 8ms)



Tasks Operate Normally

➡ One request to signature server per data-acquisition job

➡ No unexpected tasks

All IPC delays (well) below upper bounds

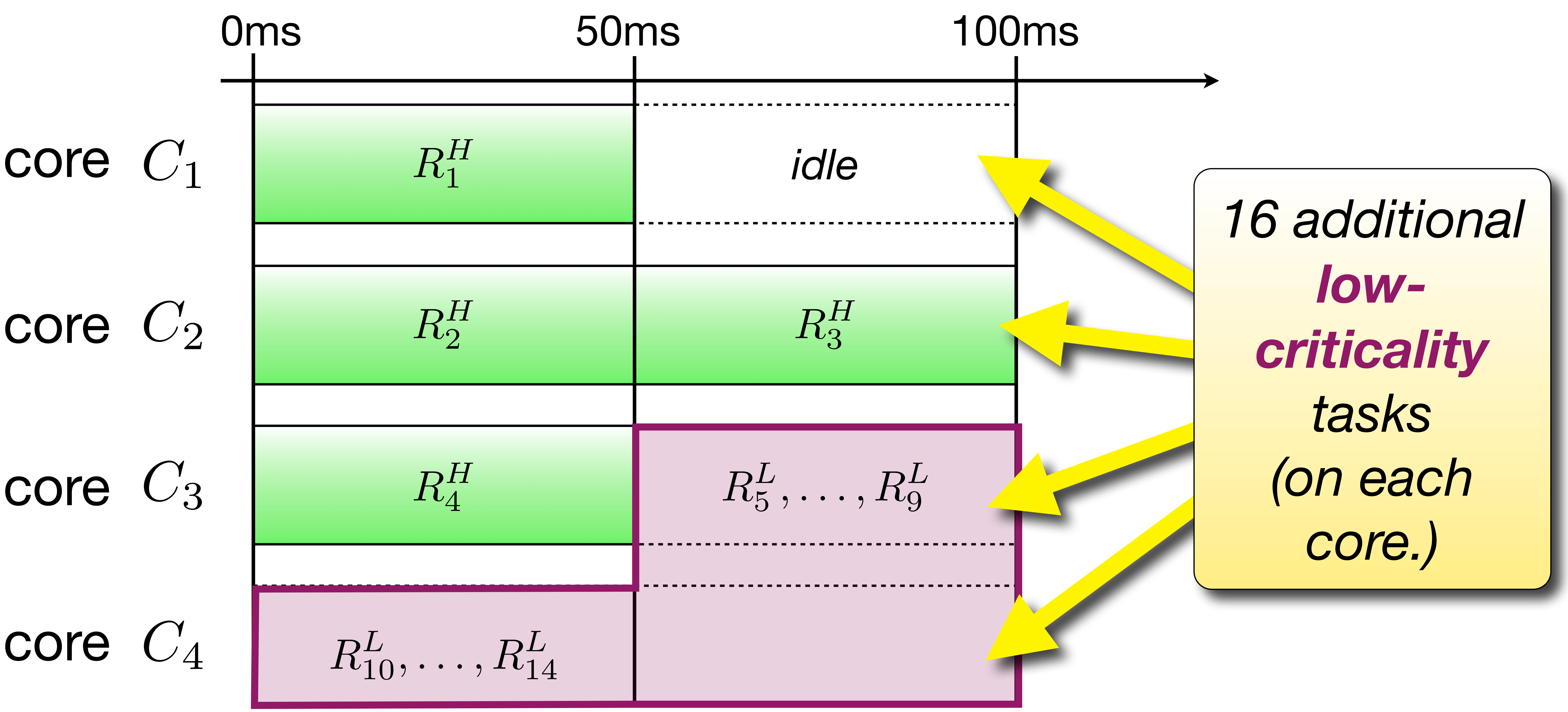
➡ **lowest bound** under PRIO-IPC (8ms)

➡ **most pessimistic** bound with FIFO-IPC (28ms)

Phase 3: Surge in Low-Criticality Tasks

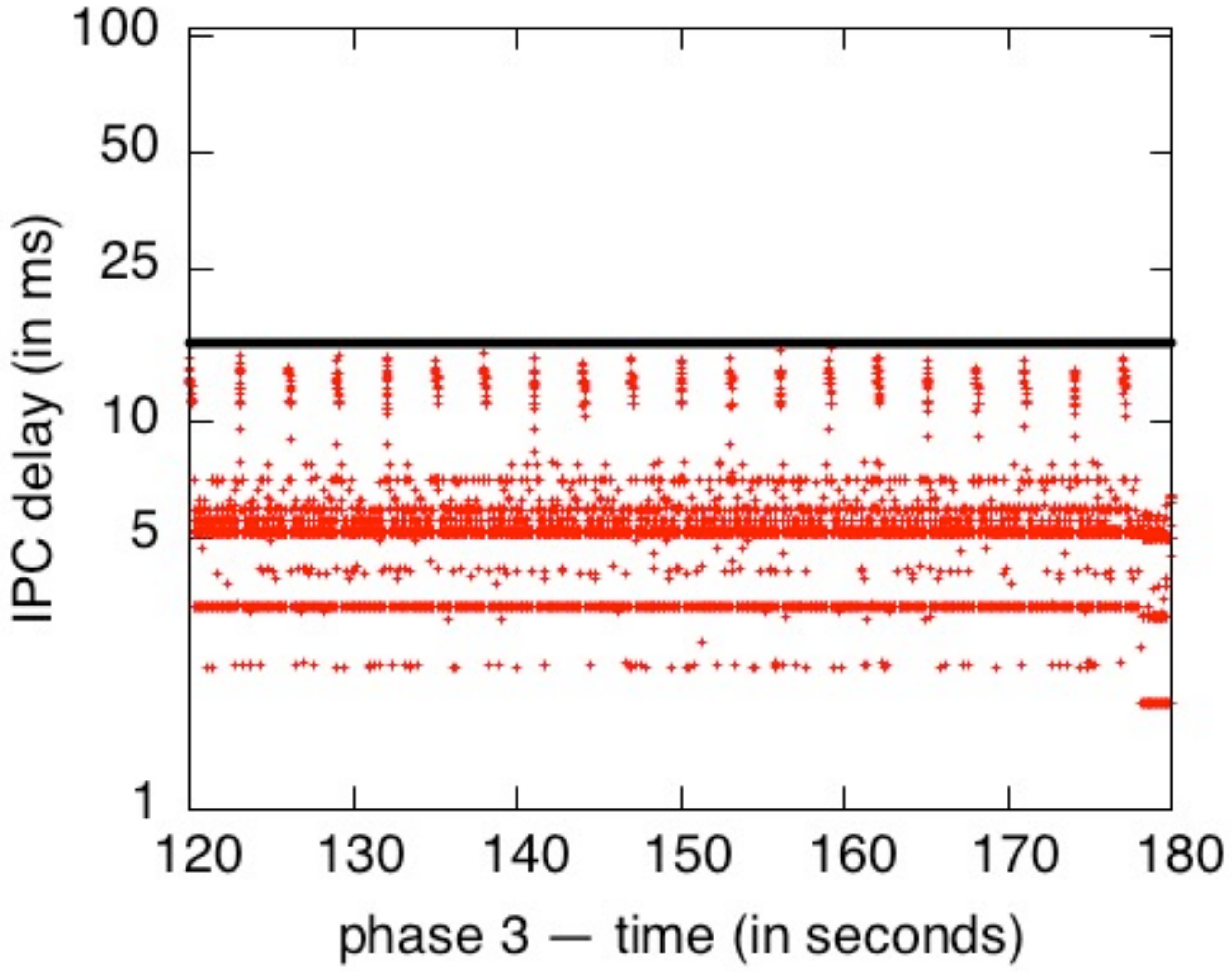
<u>Deviation from original system model:</u>

*additional **low-criticality reservations** with **period = 3000ms** are admitted on all cores.*

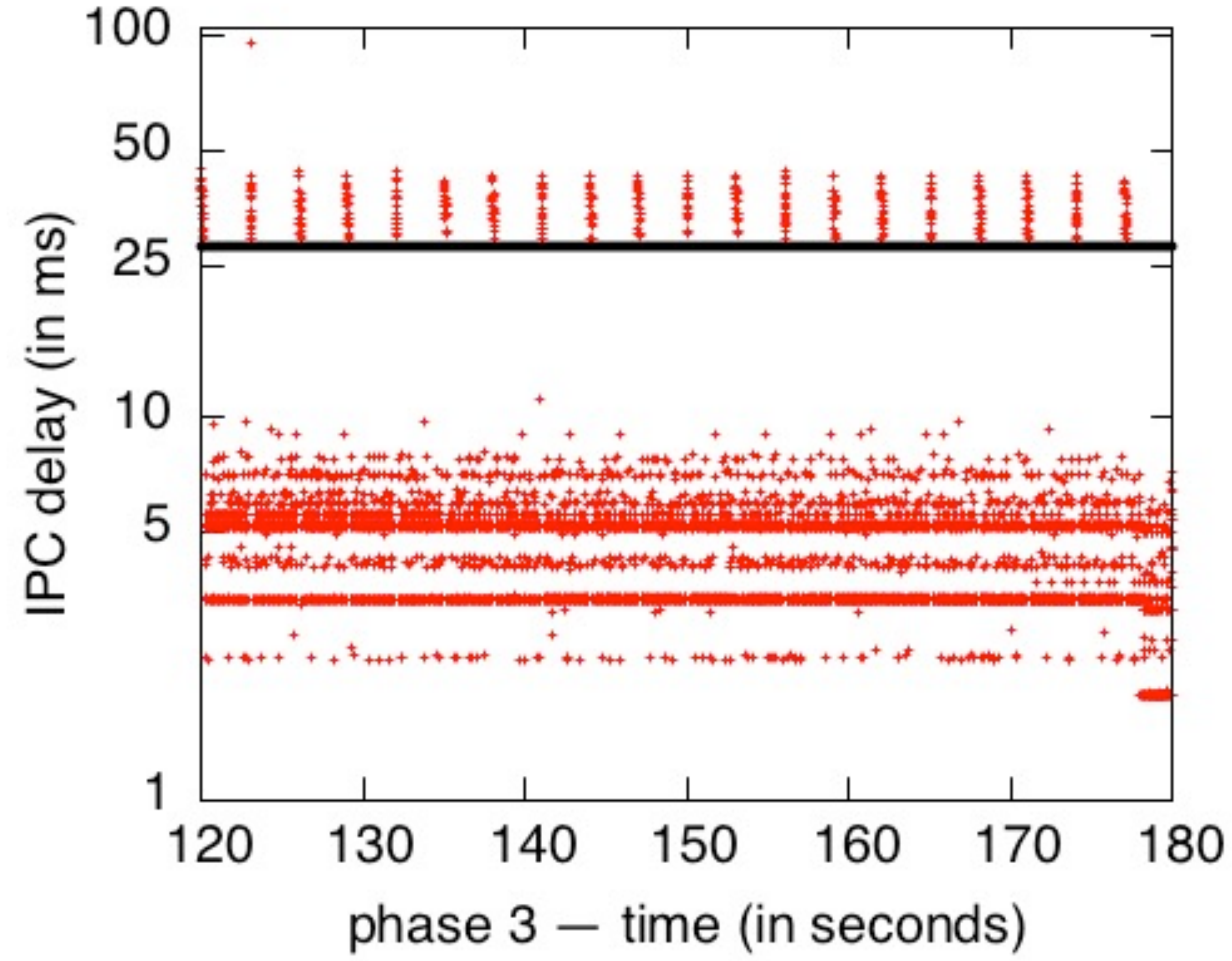


Phase 3: Surge in Low-Criticality Tasks

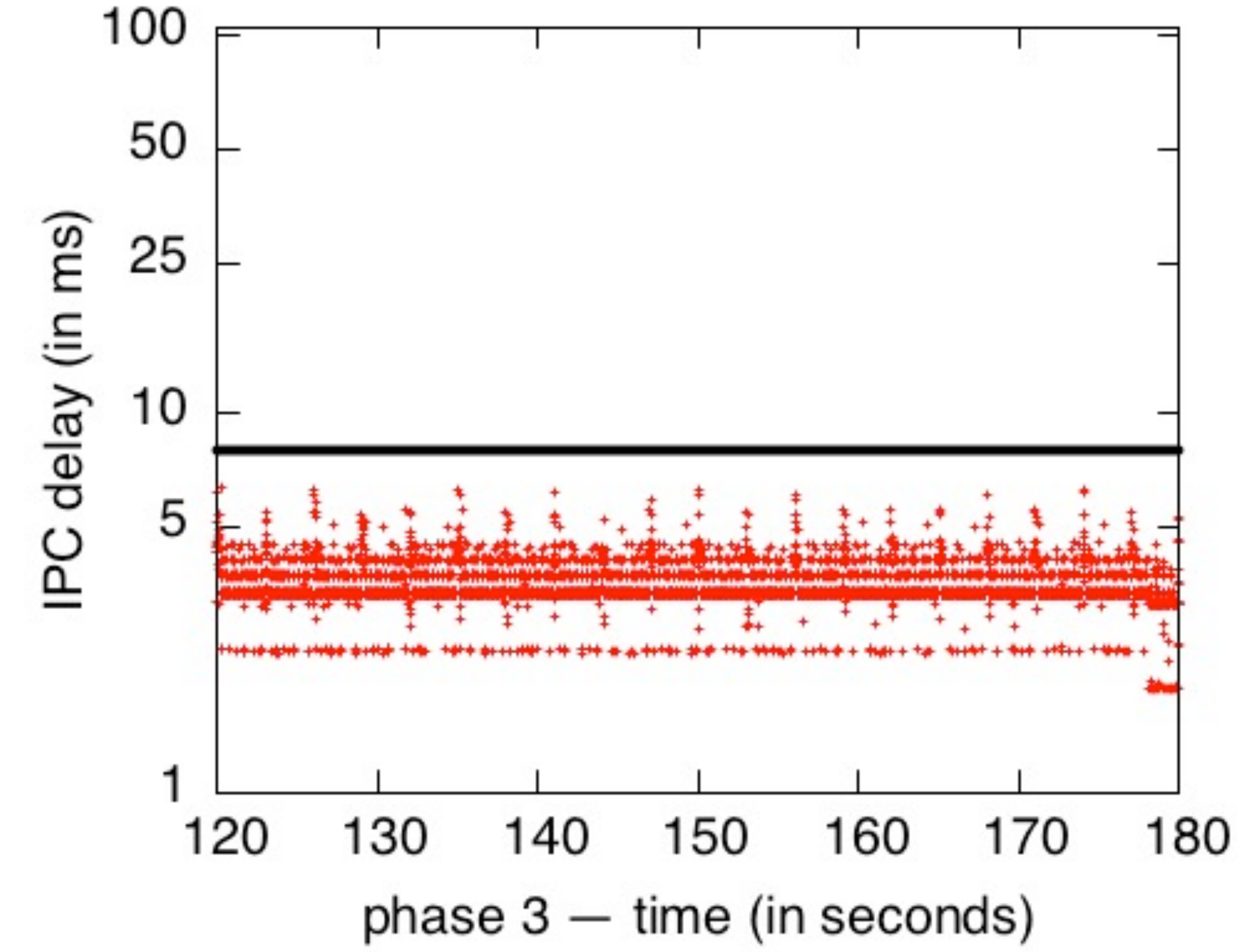
MC-IPC (predicted bound: 18ms)



FIFO-IPC (predicted bound: 28ms)



PRIO-IPC (predicted bound: 8ms)



The system is flooded with additional low-criticality tasks

➡ Each task operates as expected…

➡ …but there are unexpectedly many low-criticality tasks.

Schedulability of high-criticality tasks should not be affected

➡ **table-driven reservations** have statically higher priority anyway.

FIFO-IPC problem: unexpected queue length

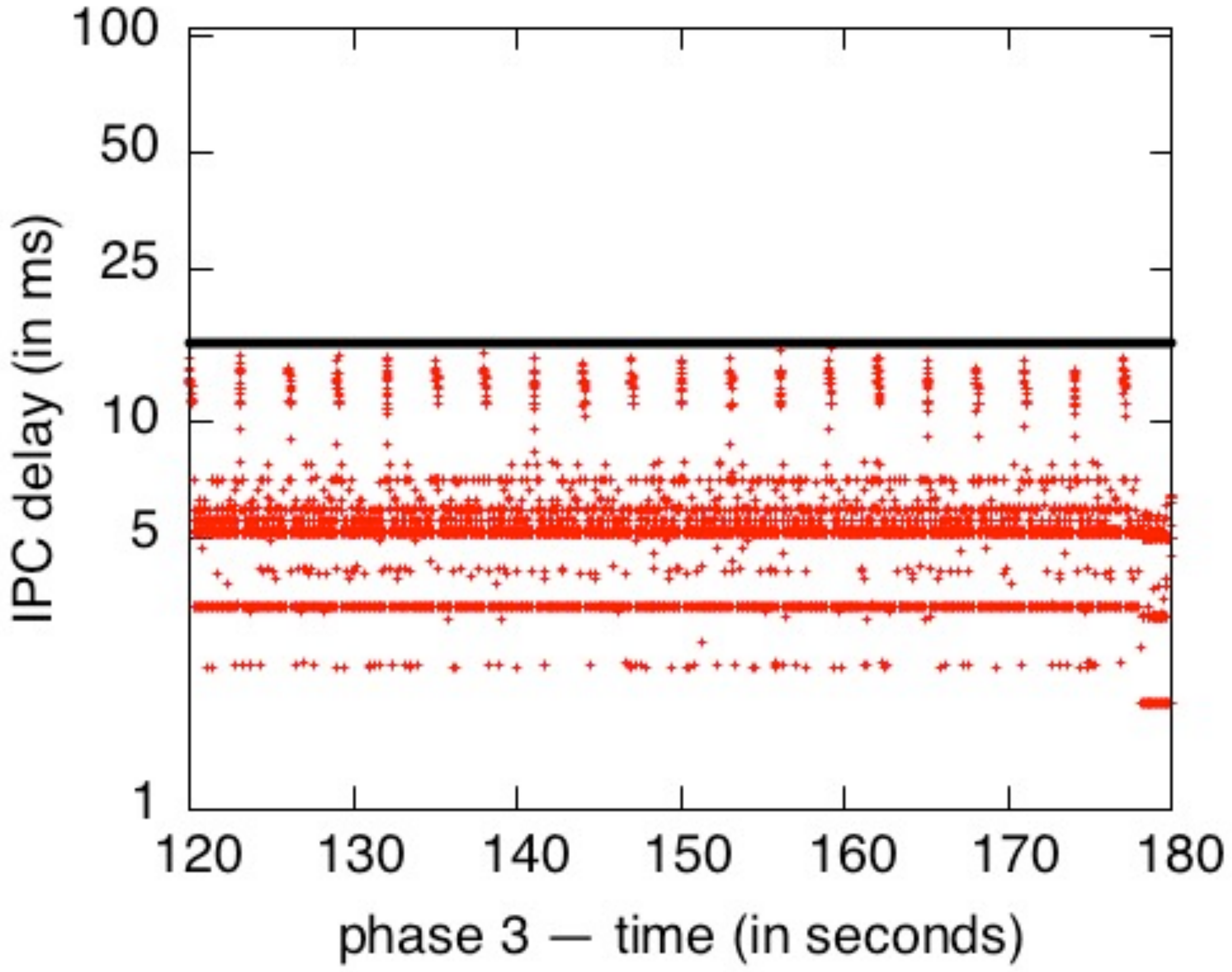
➡ With FIFO-IPC, need to **place trust** in the total number of tasks!

Phase 3

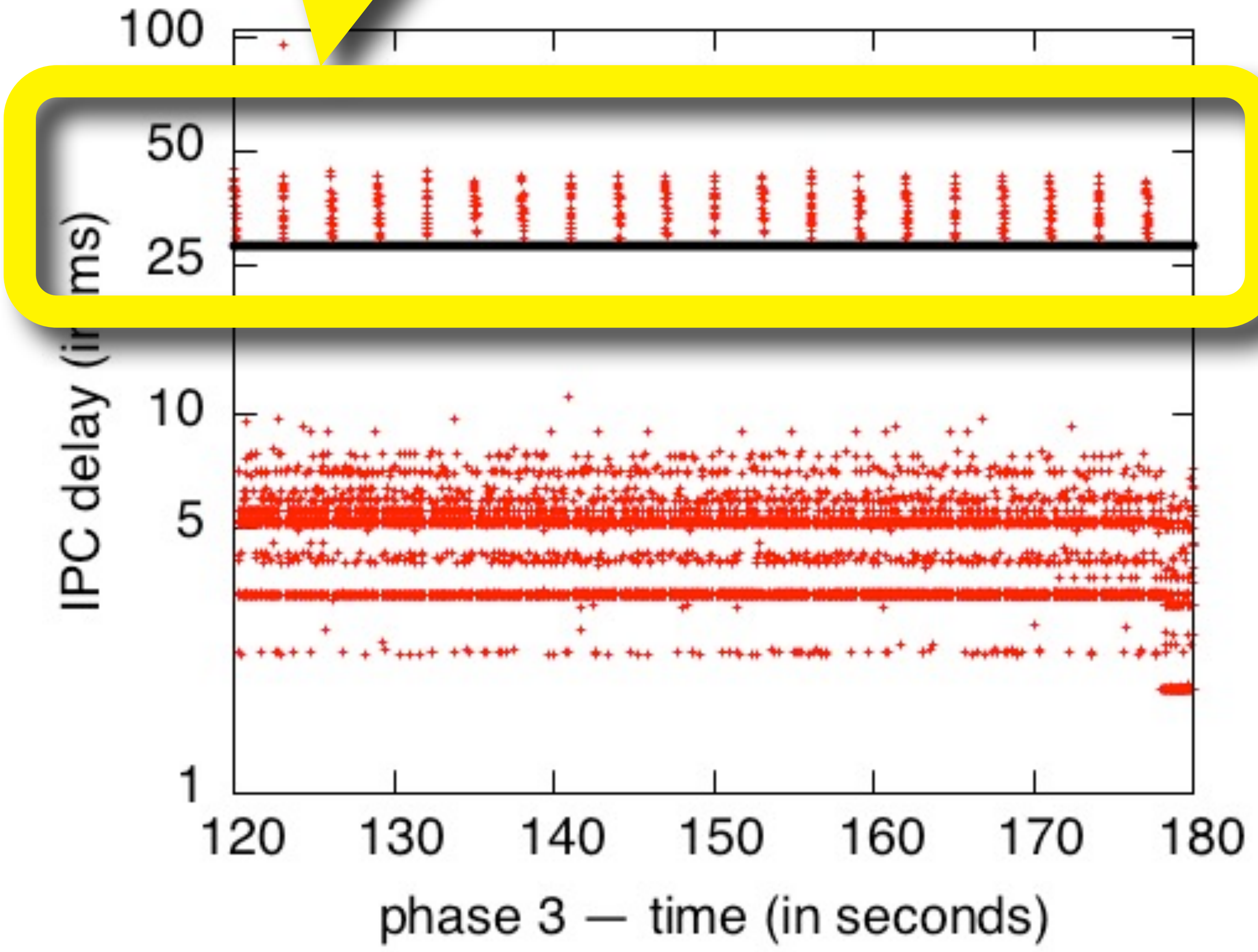
measured IPC delay > predicted maximum delay

→ *unpredictable IPC delays incurred by high-criticality*

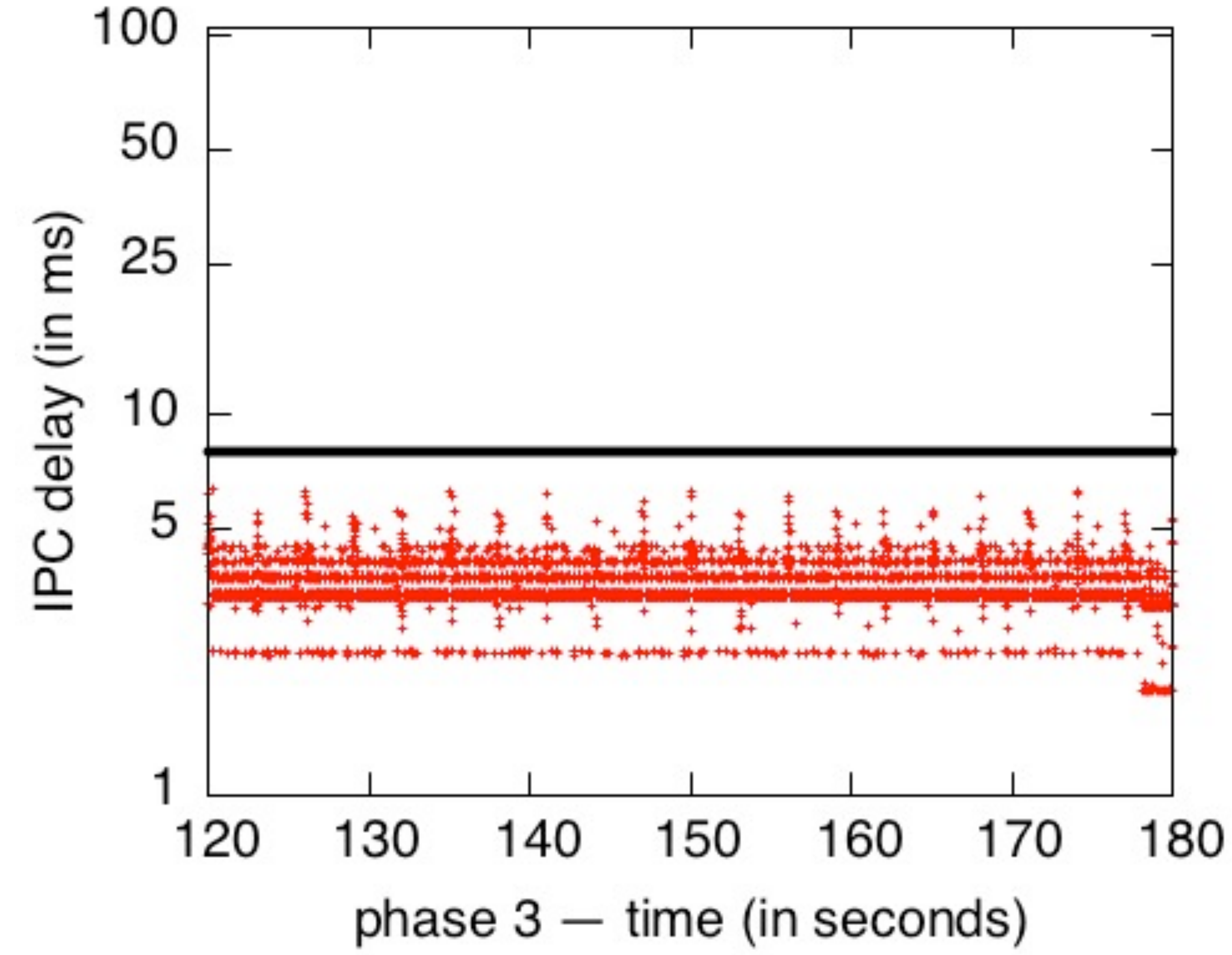
MC-IPC (predicted bound: 18ms)



FIFO-IPC (predicted bound: 28ms)



PRIO-IPC (predicted bound: 8ms)



The system is flooded with additional low-criticality tasks

➡ Each task operates as expected…

➡ …but there are unexpectedly many low-criticality tasks.

Schedulability of high-criticality tasks should not be affected

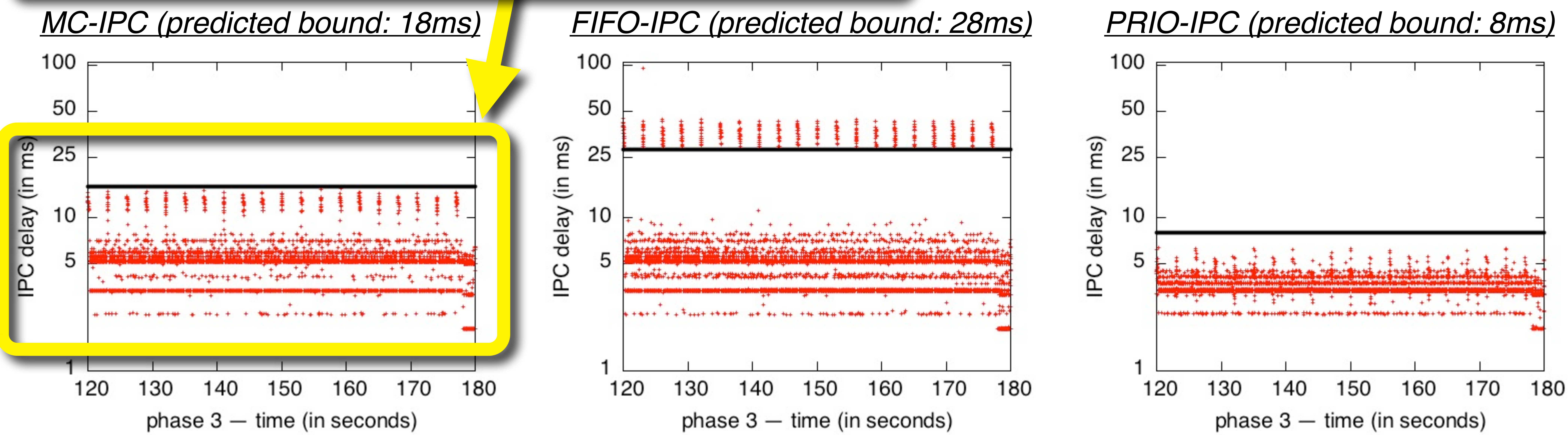
➡ **table-driven reservations** have statically higher priority anyway.

FIFO-IPC problem: unexpected queue length

➡ With FIFO-IPC, need to **place trust** in the total number of tasks!

MC-IPC: somewhat elevated IPC delays, but below predicted bound!

…-Criticality Tasks



The system is flooded with additional low-criticality tasks
- ➡ Each task operates as expected…
- ➡ …but there are unexpectedly many low-criticality tasks.

Schedulability of high-criticality tasks should not be affected
- ➡ **table-driven reservations** have statically higher priority anyway.

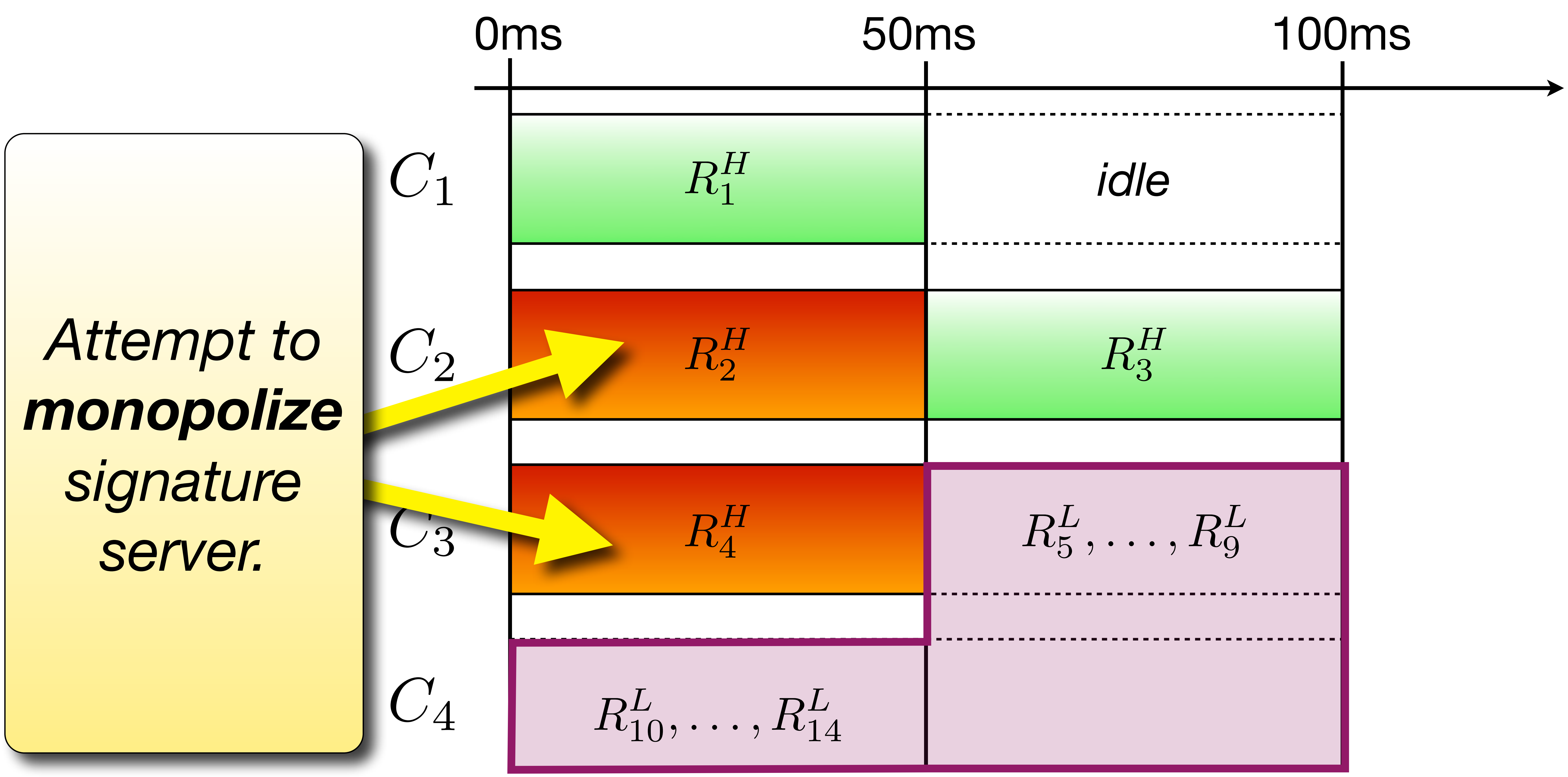
FIFO-IPC problem: unexpected queue length
- ➡ With FIFO-IPC, need to **place trust** in the total number of tasks!

Phase 5: High-Criticality DoS

<u>*Deviation from original system model:*</u>

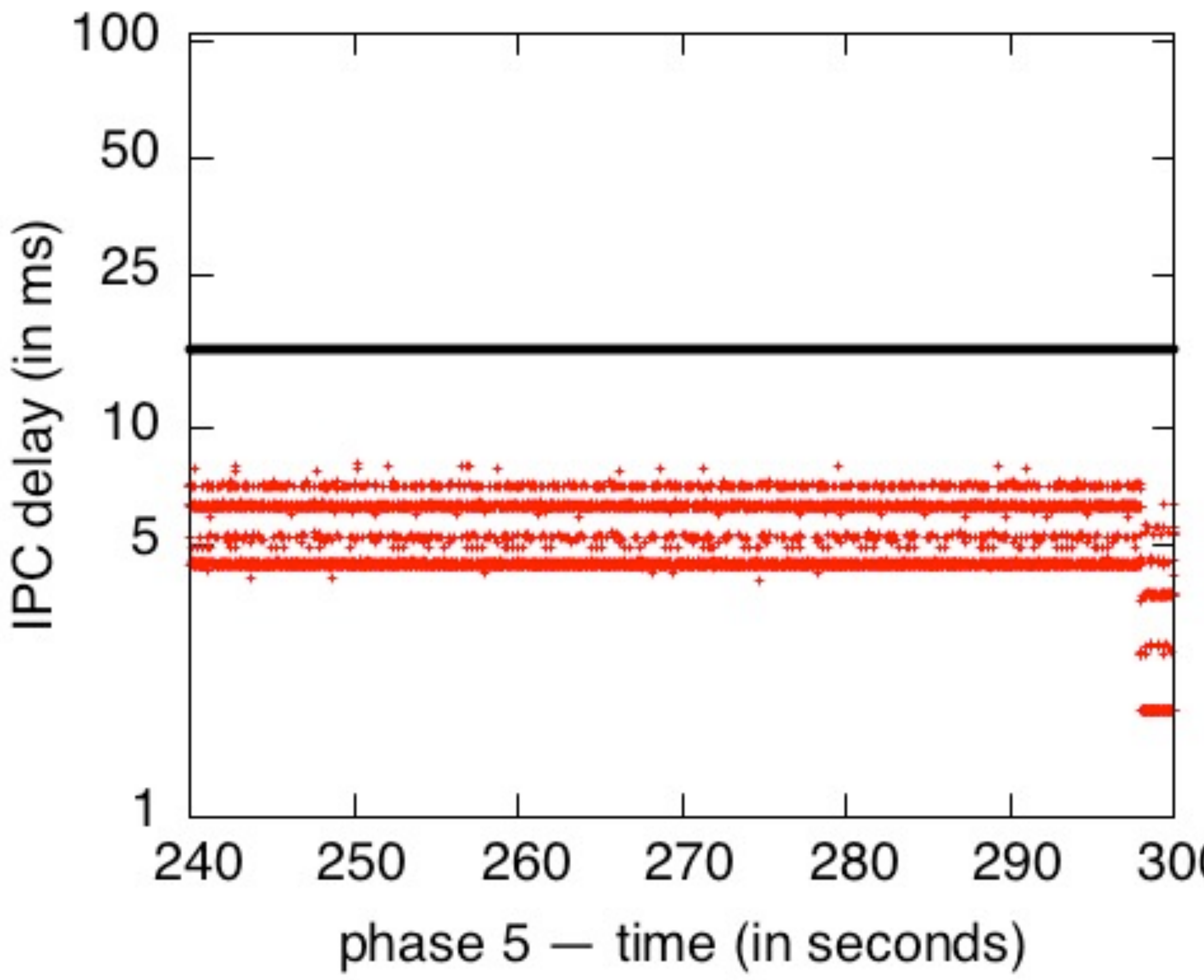
*two **high-criticality tasks** **malfunction**,*
*start invoking signature server **as quickly as possible***

➔ *effectively a **Denial-of-Service** (**DoS**) attack.*

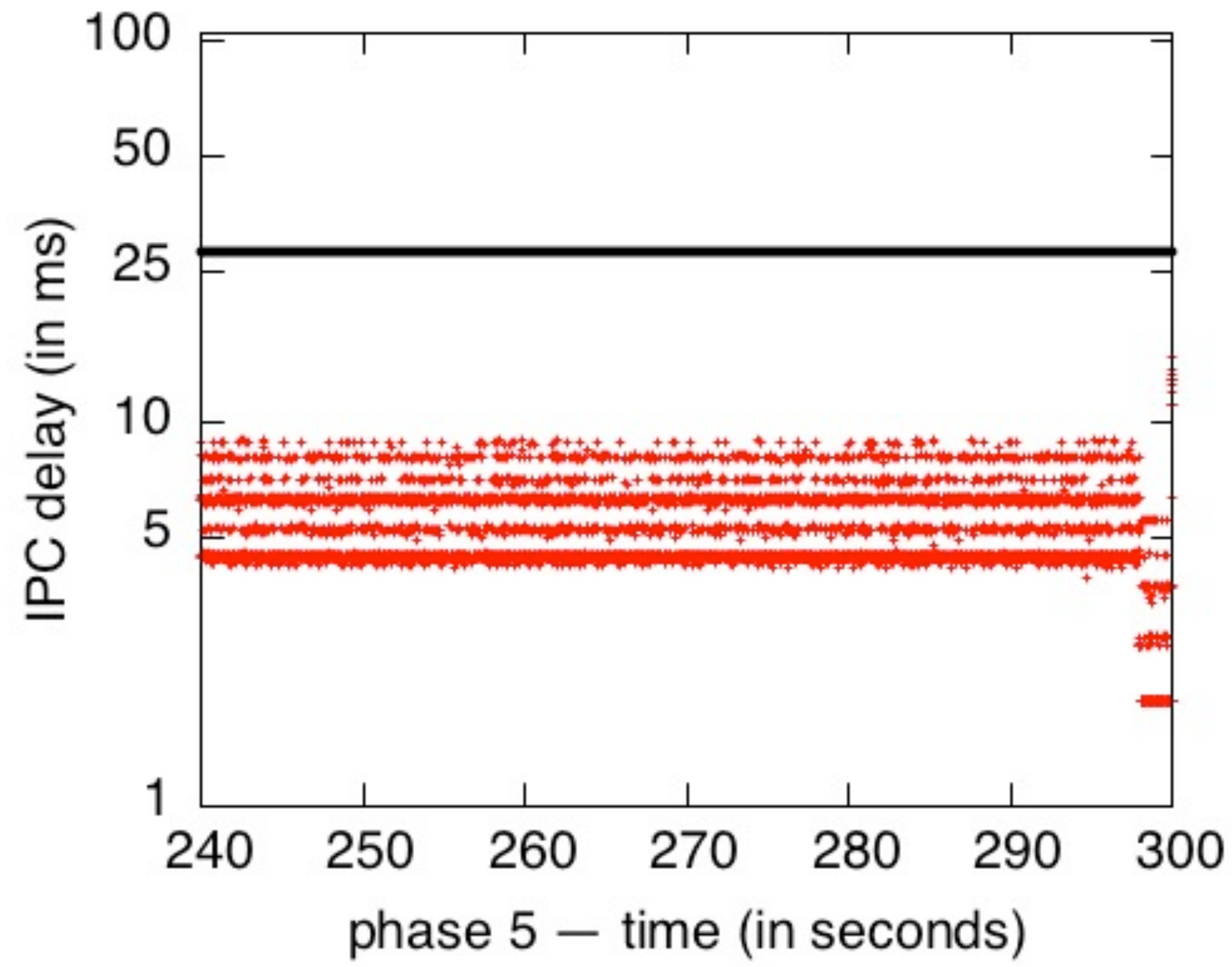


Phase 5: High-Criticality DoS

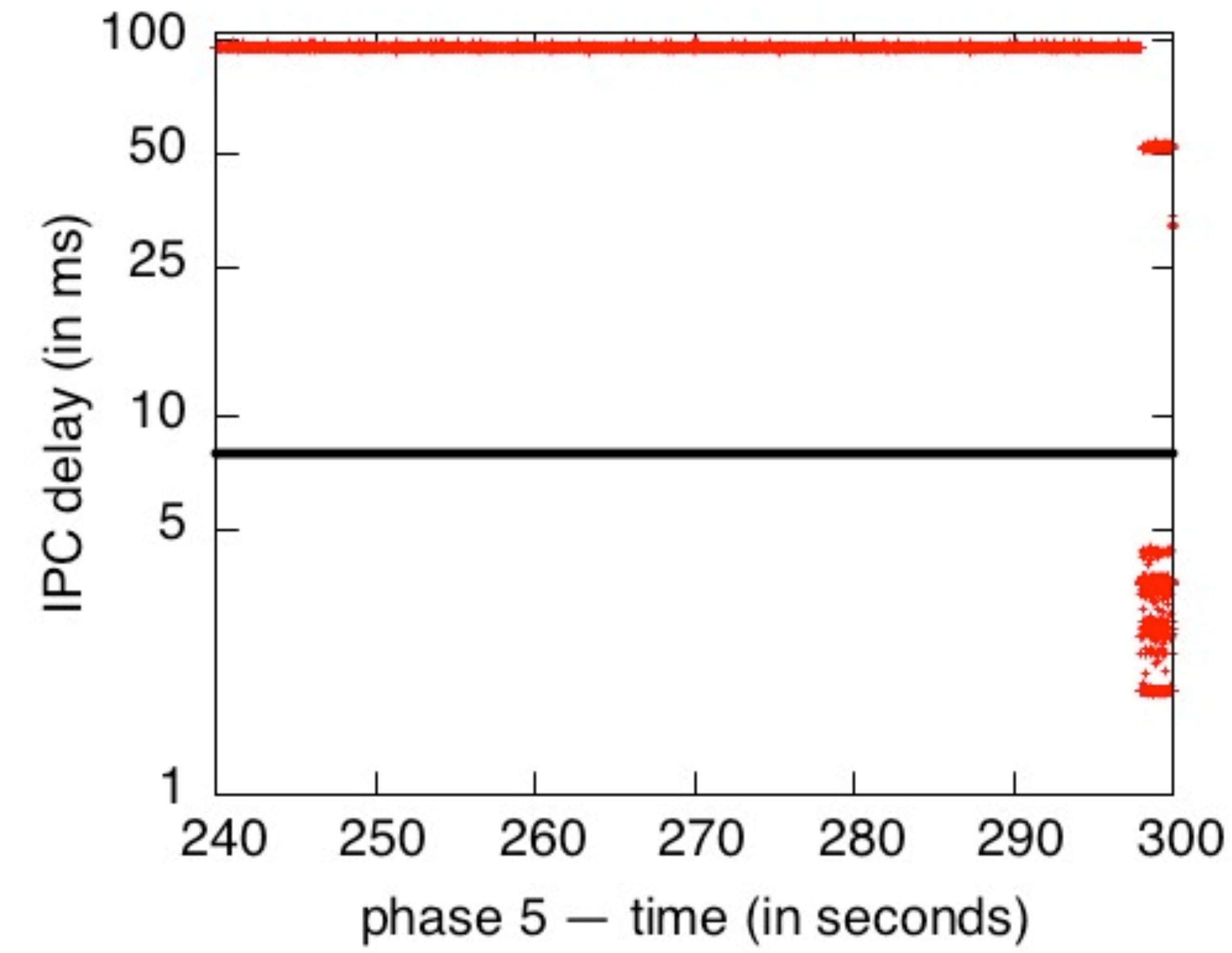
MC-IPC (predicted bound: 18ms)



FIFO-IPC (predicted bound: 28ms)



PRIO-IPC (predicted bound: 8ms)



Two higher-priority tasks of equal criticality monopolize server

➡ PRIO-IPC permits starvation

➡ With PRIO-IPC, need to **place trust** in maximum request frequencies.

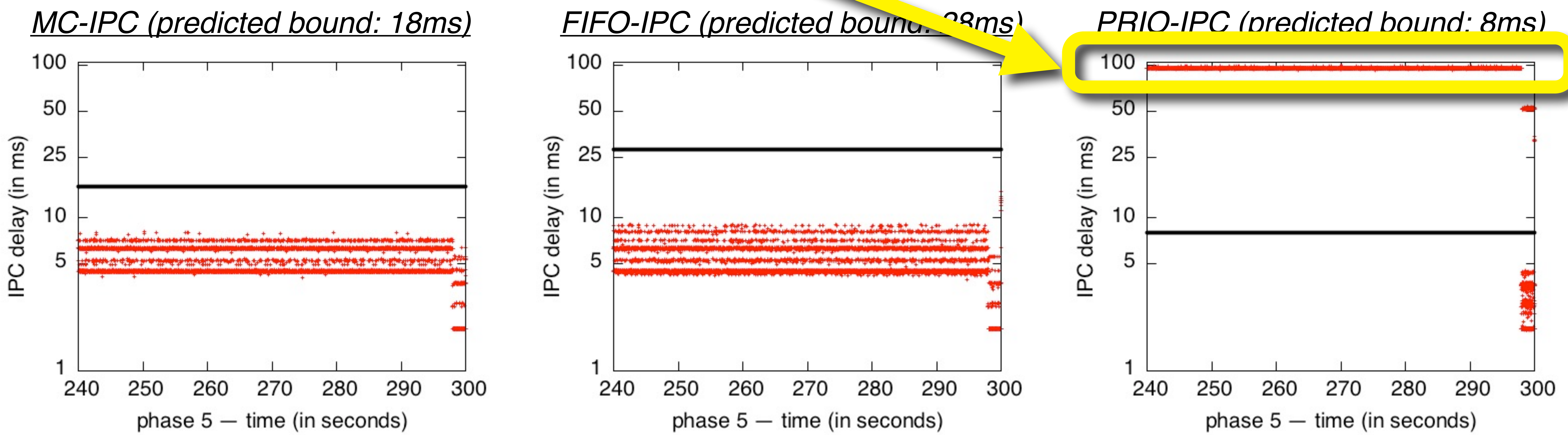
FIFO-IPC

➡ predicted bound and actual IPC delays **independent** of request frequencies

MC-IPC

➡ predicted bound accounts for **arbitrary** request frequencies

measured IPC delay approaching 100ms (= period)
➔ *high-criticality task* on core 1 *starved completely.*



Two higher-priority tasks of equal criticality monopolize server

➡ PRIO-IPC permits starvation

➡ With PRIO-IPC, need to **place trust** in maximum request frequencies.

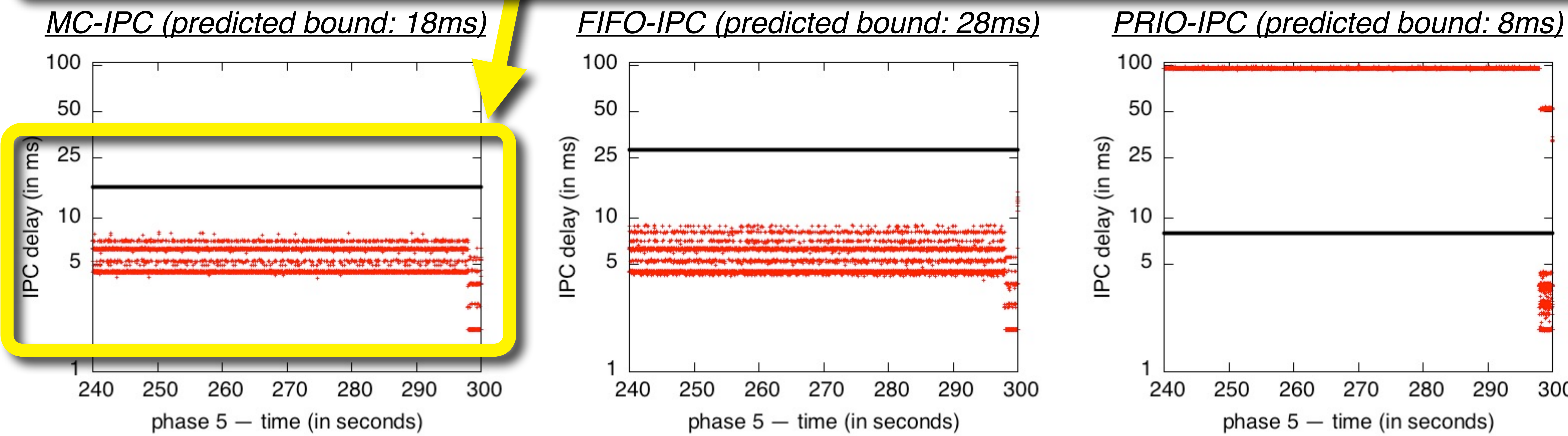
FIFO-IPC

➡ predicted bound and actual IPC delays **independent** of request frequencies

MC-IPC

➡ predicted bound accounts for **arbitrary** request frequencies

MC-IPC: provides freedom-from-interference *even with respect to tasks of* ***equal or higher criticality*** **(= fault containment)**.



Two higher-priority tasks of equal criticality monopolize server

- ➡ PRIO-IPC permits starvation
- ➡ With PRIO-IPC, need to place trust in maximum request frequencies.

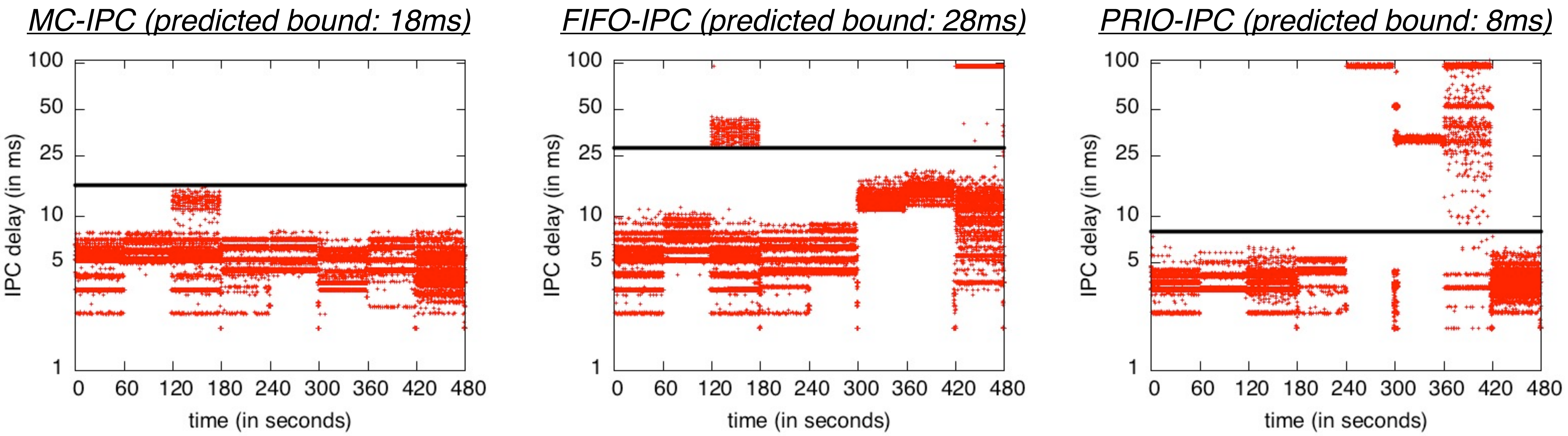
FIFO-IPC

- ➡ predicted bound and actual IPC delays **independent** of request frequencies

MC-IPC

- ➡ predicted bound accounts for **arbitrary** request frequencies

Only MC-IPC Ensures Predicted Worst Case



FIFO-IPC and PRIO-IPC

➡ Vulnerable to deviations from analyzed system model.

MC-IPC: does not require trusting…

➡ …the bound on the **number of tasks** (either high- or low-criticality).

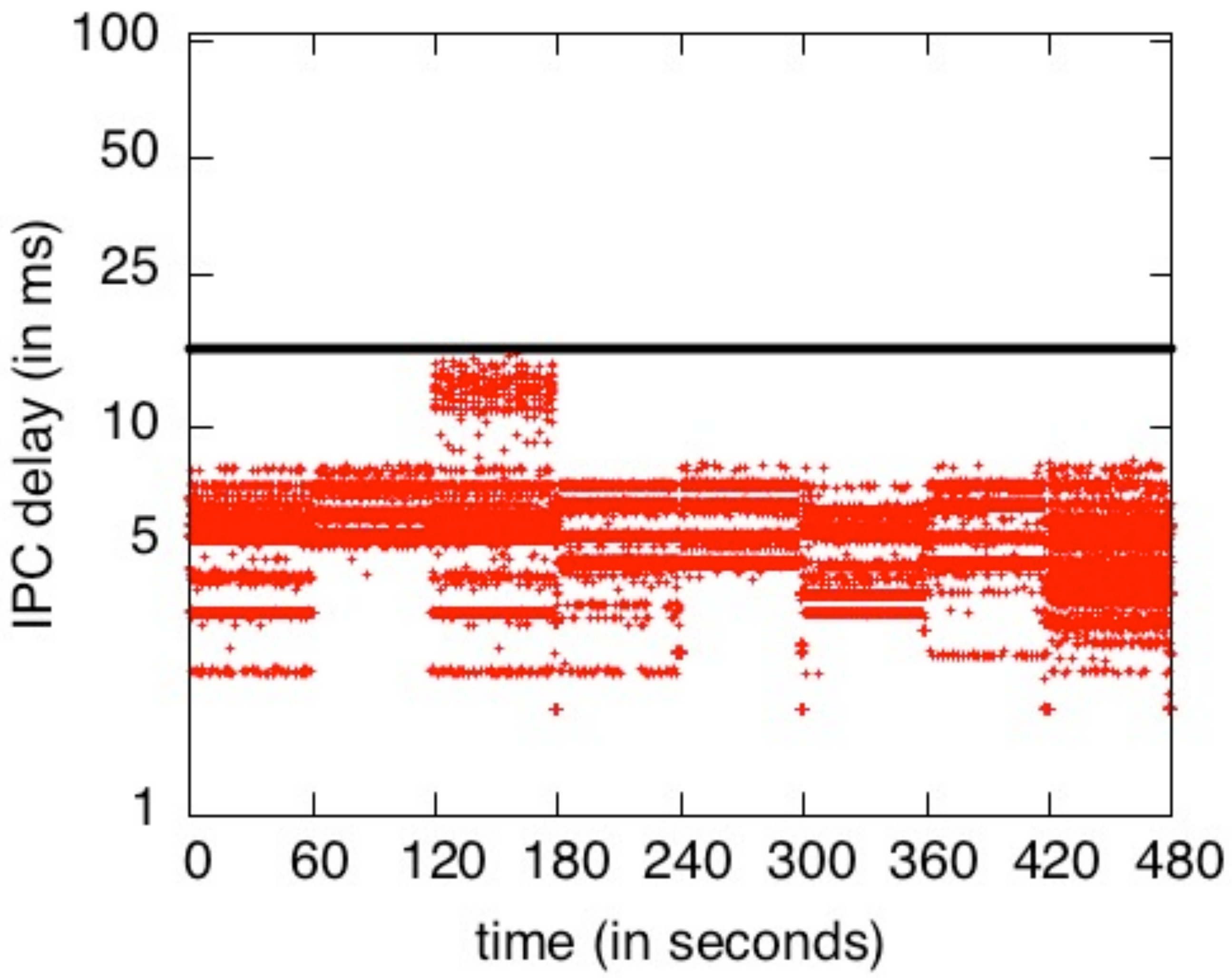
➡ …any bounds on **maximum request frequencies**.

➡ …the behavior of **best-effort or any real-time tasks**.

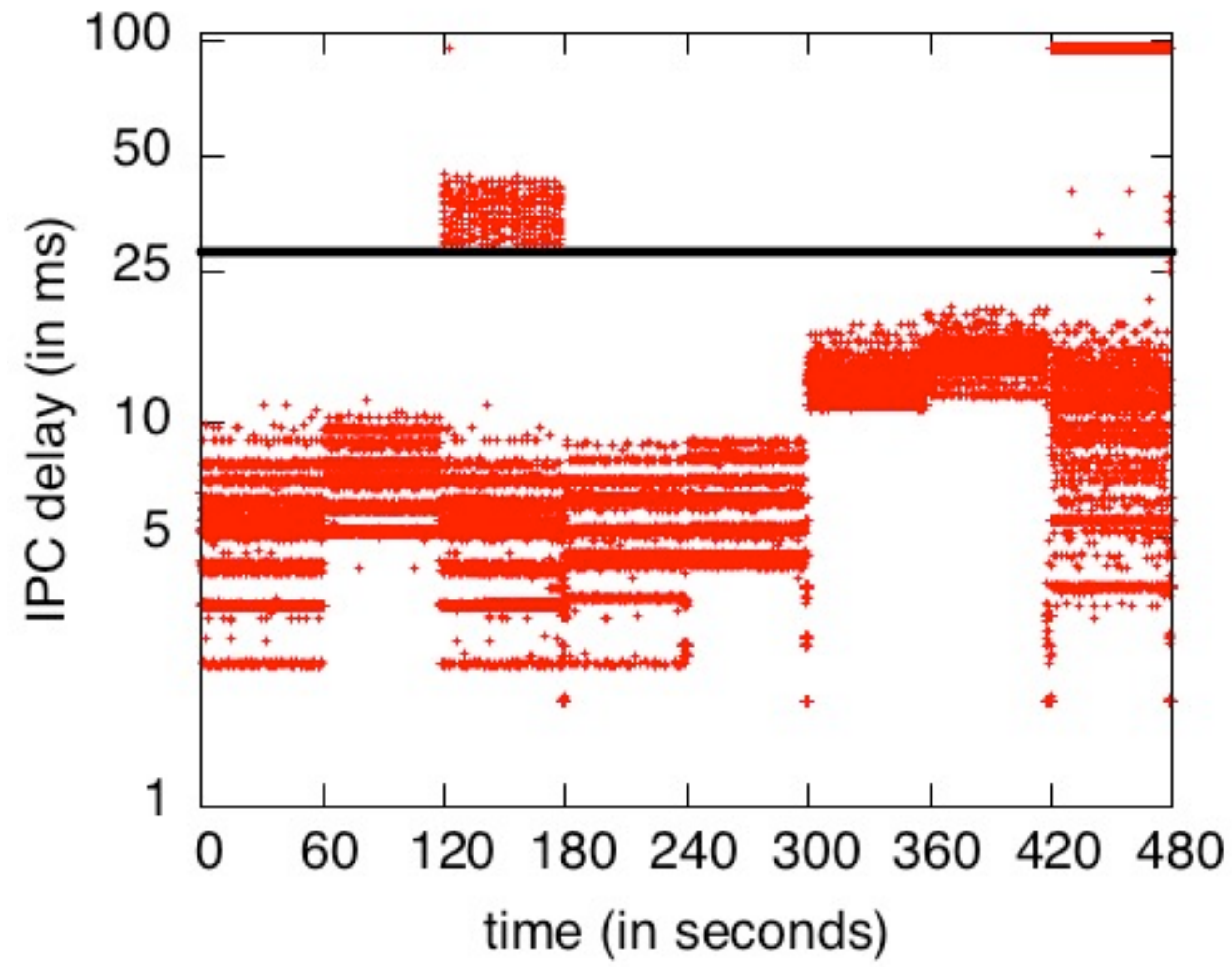
➡ *The trust rests solely with the server (must reject illegal requests).*

Only MC-IPC Ensures Predicted Worst Case

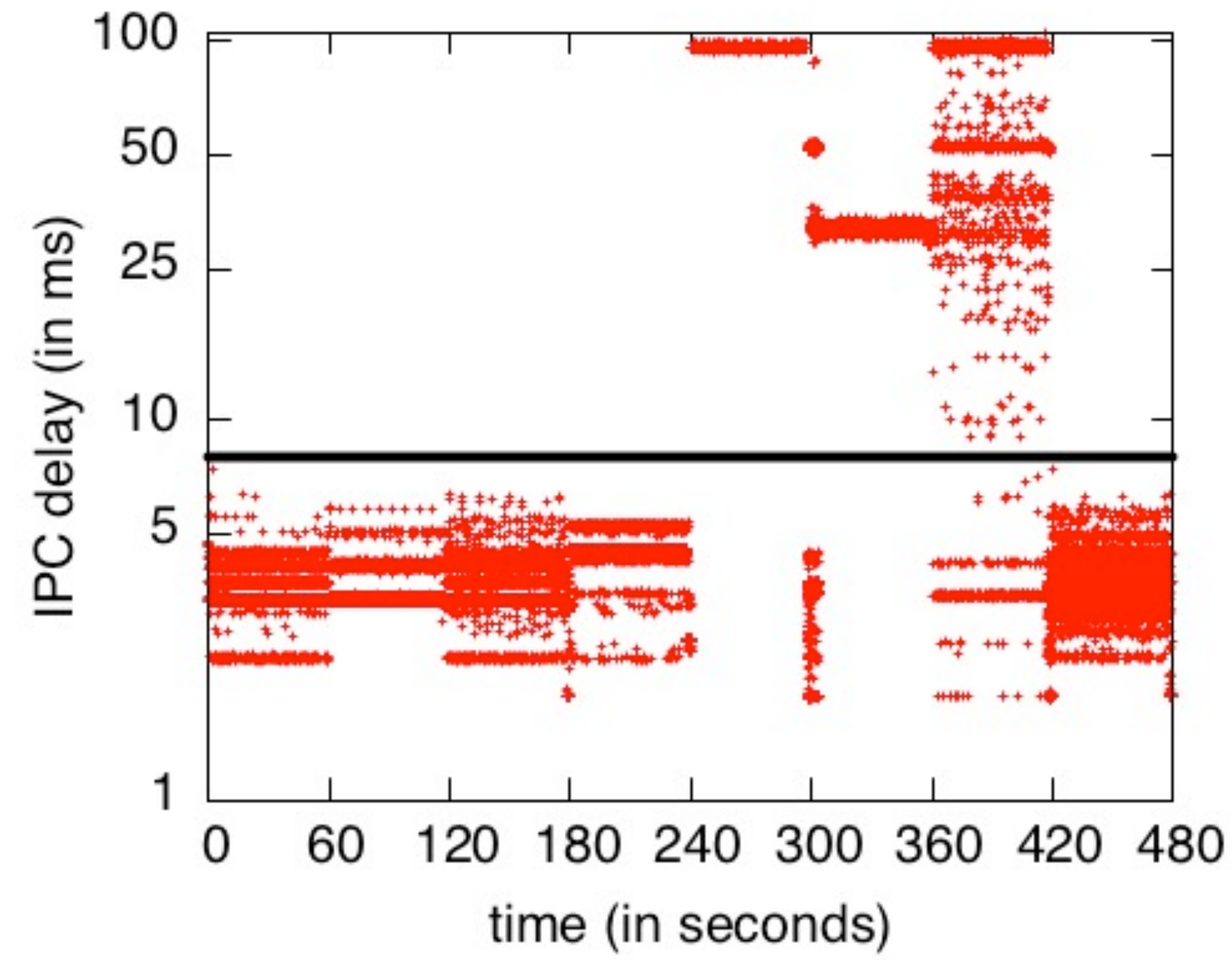
MC-IPC (predicted bound: 18ms)



FIFO-IPC (predicted bound: 28ms)



PRIO-IPC (predicted bound: 8ms)



Analytical Benefits: Integration with Vestal's Model

Because the **MC-IPC** ensures **strict isolation**,
it is simple to **statically reclaim pessimism at lower criticalities**
in the spirit of (and compatible with) **Vestal's model**.

Criticality-dependent IPC costs, criticality-dependent interference,…

See paper for details.

Conclusion

Summary

Locking and lock-free/wait-free synchronization are ill-suited for cross-criticality synchronization.

➡ Explicit synchronization **implies trust** and **violates isolation**.

MC-IPC: Resources may be shared between high- and low-criticality tasks without violating freedom-from-interference

➡ Other tasks do not need to be trusted, only the accessed server.

Prototype and case study in LITMUS^{RT}

➡ (Lack of) isolation easily observed in practical system

Limitations & Future Work

Algorithmic challenge: nested IPC requests (server to server)

- ➡ Not supported by current analysis.
- ➡ Which queue to enqueue in?
  - Cluster of server? Cluster of client? Both? None?

IPC Overheads

- ➡ Prototype in LITMUS^{RT} **not optimized**, not comparable to highly tuned implementations like those found in the L4 family.
- ➡ Even if highly optimized, MC-IPC will likely still be more heavyweight due to additional queue operations.

MC-IPC: freedom-from-interference despite shared resources, despite untrusted tasks, even on multiprocessors.

LITMUS^RT

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

www.litmus-rt.org

Max Planck Institute for Software Systems

Björn B. Brandenburg
bbb@mpi-sws.org

Appendix

P5: Accommodating Best-Effort Tasks

Sharing with best-effort tasks may be unavoidable for system resources.

P5: Accommodating Best-Effort Tasks

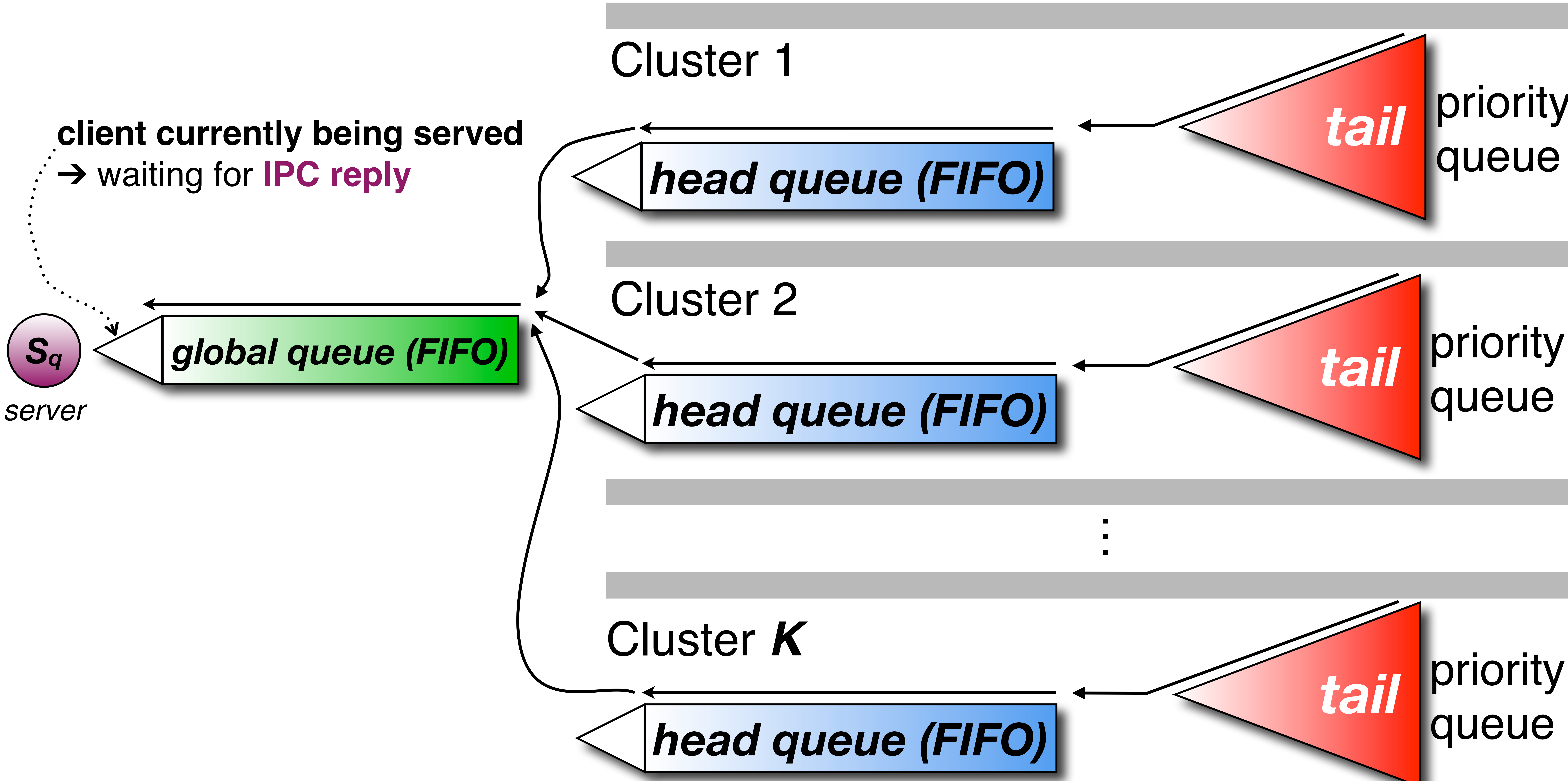
Sharing with best-effort tasks may be unavoidable for system resources.

*A **best-effort task** **without any guaranteed budget** is no different than a **real-time task** that **exhausted its budget** just after it started being served.*

➔ *bandwidth inheritance takes care of such tasks*

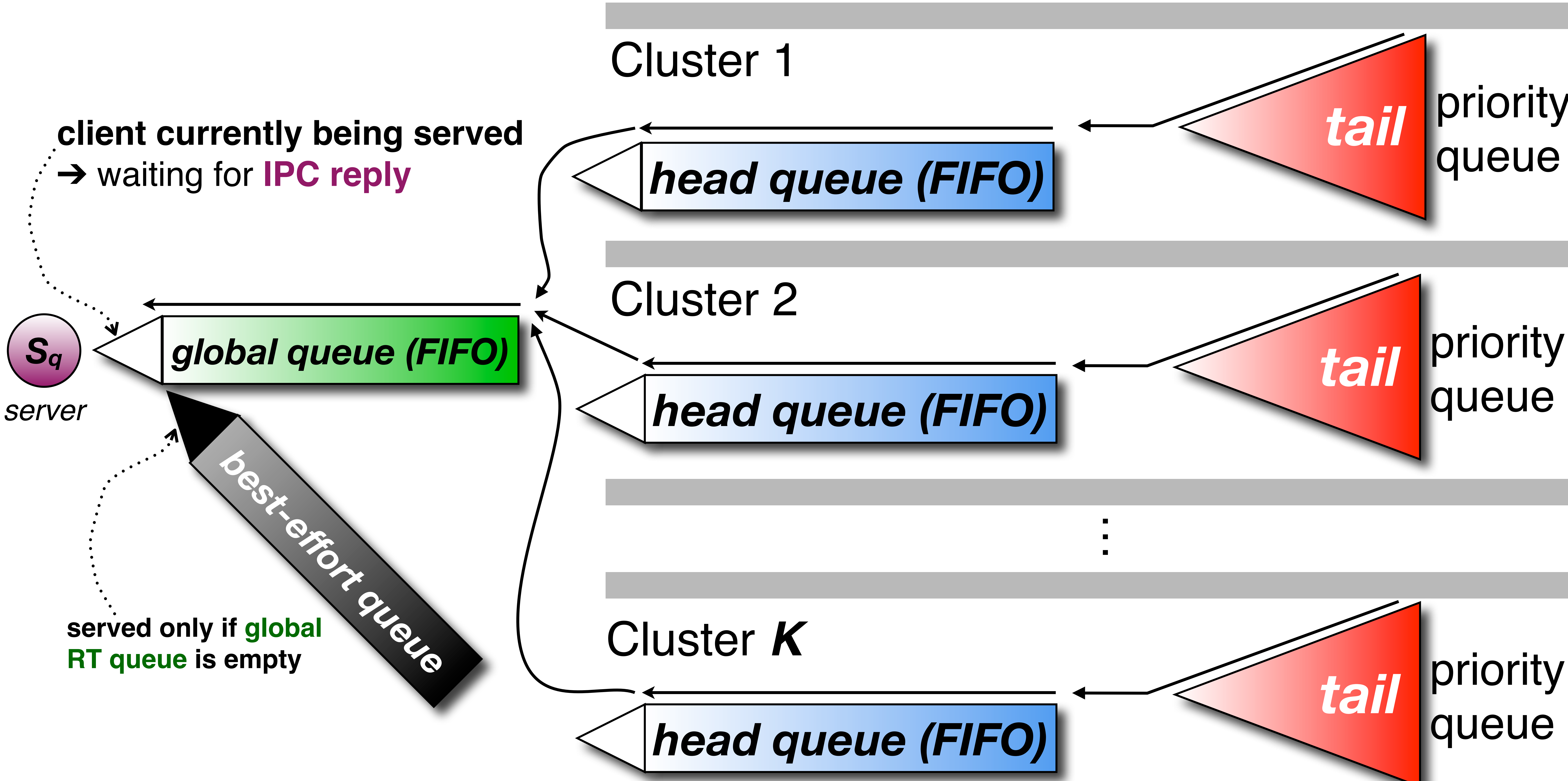
P5: Accommodating Best-Effort Tasks

Sharing with best-effort tasks may be unavoidable for system resources.



P5: Accommodating Best-Effort Tasks

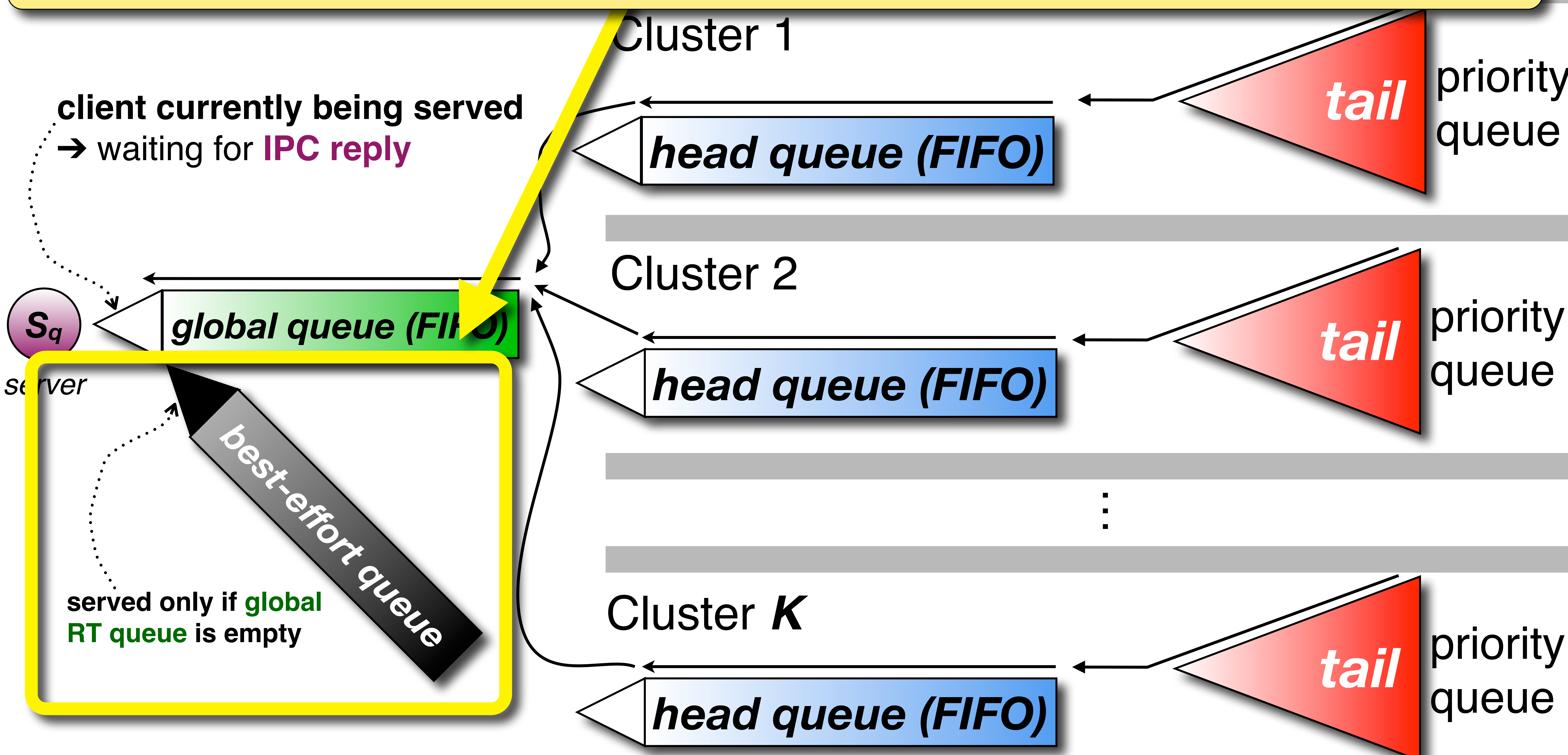
Sharing with best-effort tasks may be unavoidable for system resources.



A real-time task encounters at most one best-effort task.

Analytically, no different than finding S_q “**stuck**” on a **real-time task with an exhausted budget**.

Best-effort queue is global, can be ordered arbitrarily.



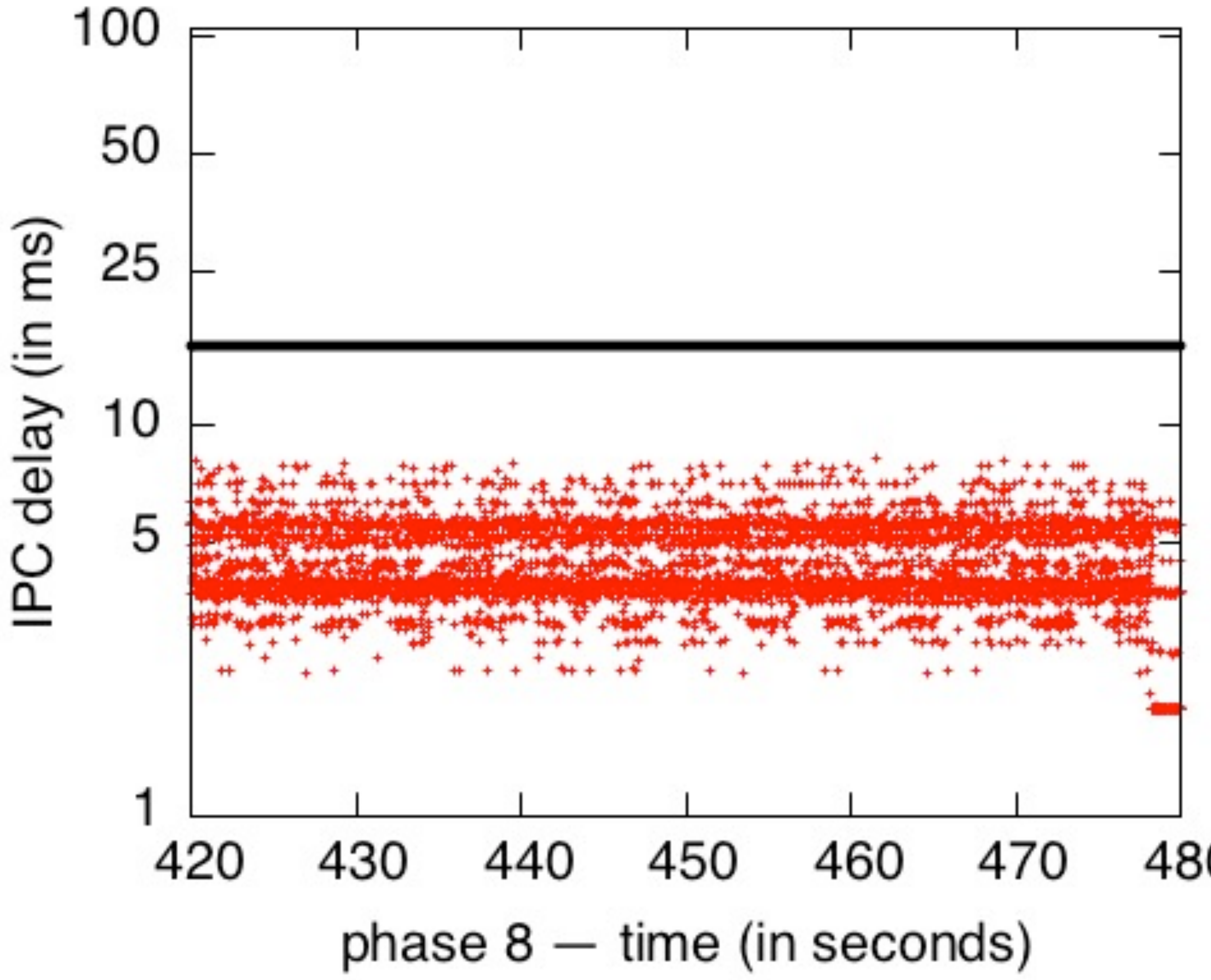
Phase 8: Flood of Best-Effort Tasks

Deviation from original system model:

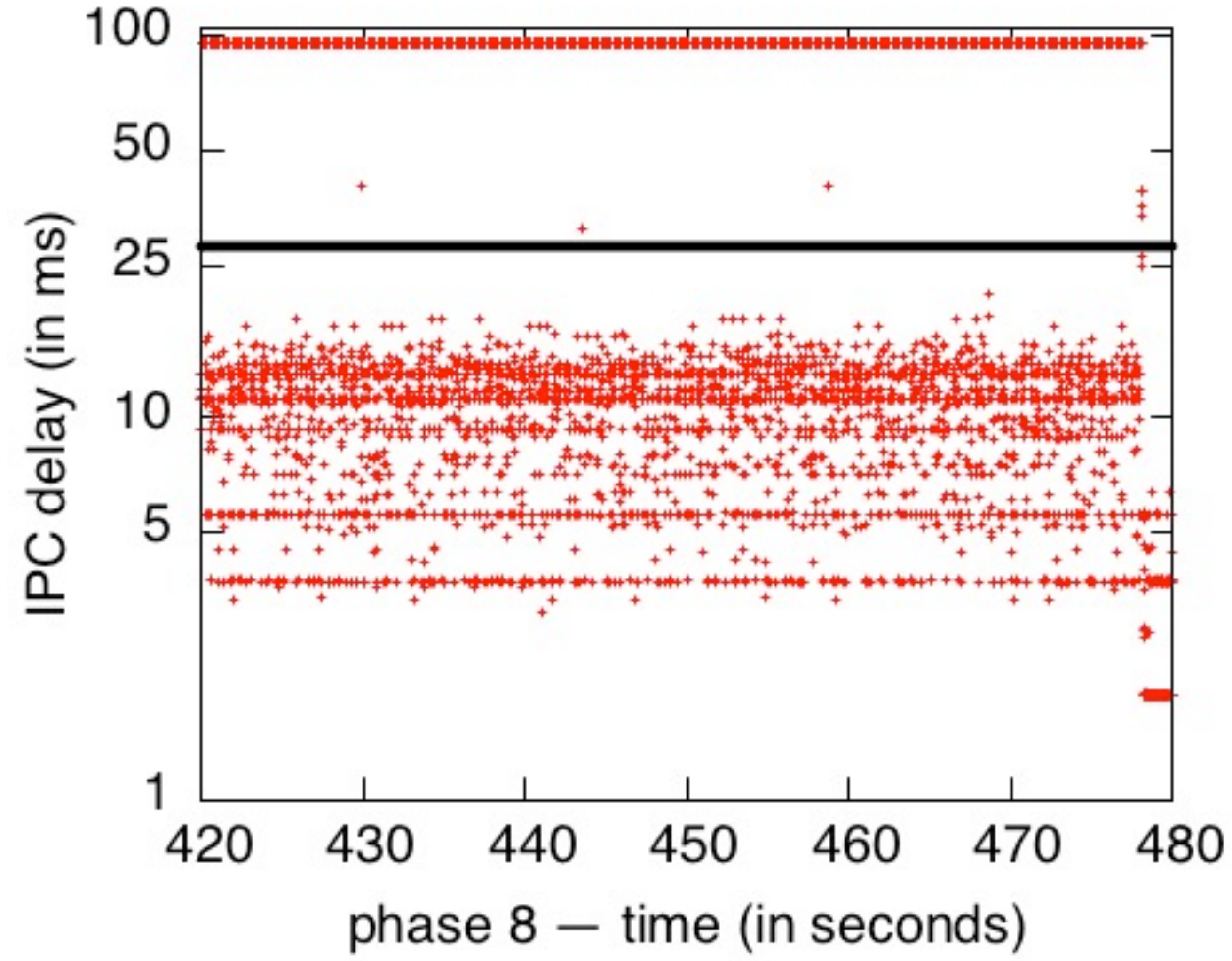
*additional **low-criticality reservations***
*with **period = 3000ms** are admitted on all cores.*

Phase 8: Flood of Best-Effort Tasks

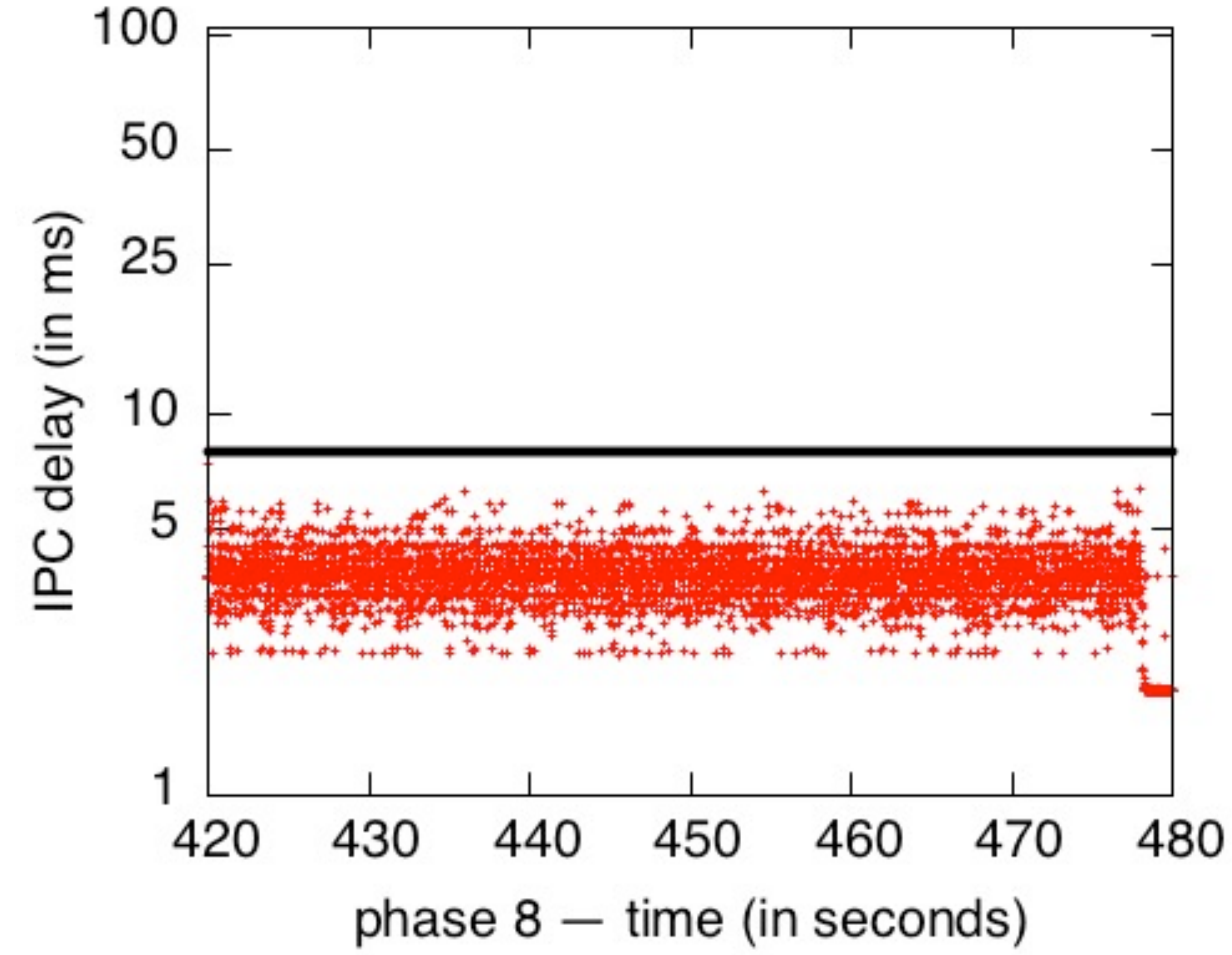
MC-IPC (predicted bound: 18ms)



FIFO-IPC (predicted bound: 28ms)



PRIO-IPC (predicted bound: 8ms)



Best-effort tasks create contention for resource used by real-time tasks

➡ With FIFO-IPC, need to **place trust** in maximum number of background best-effort tasks.

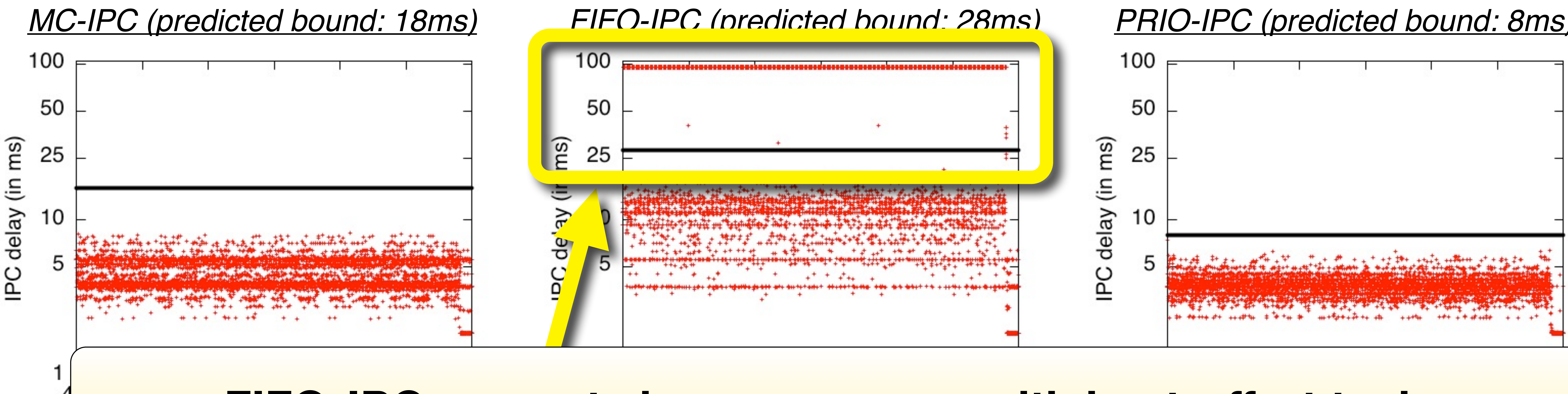
PRIO-IPC

➡ Best-effort tasks have lower priority ➔ **no additional delays**.

MC-IPC

➡ Handles best-effort tasks explicitly ➔ analysis **accounts** for best-effort tasks.

Phase 8: Flood of Best-Effort Tasks



FIFO-IPC: cannot share resources with best-effort tasks
(or must derive high-criticality assurance on maximum number of tasks).

Be

➡ With FIFO-IPC, need to **place trust** in maximum number of background best-effort tasks.

PRIO-IPC

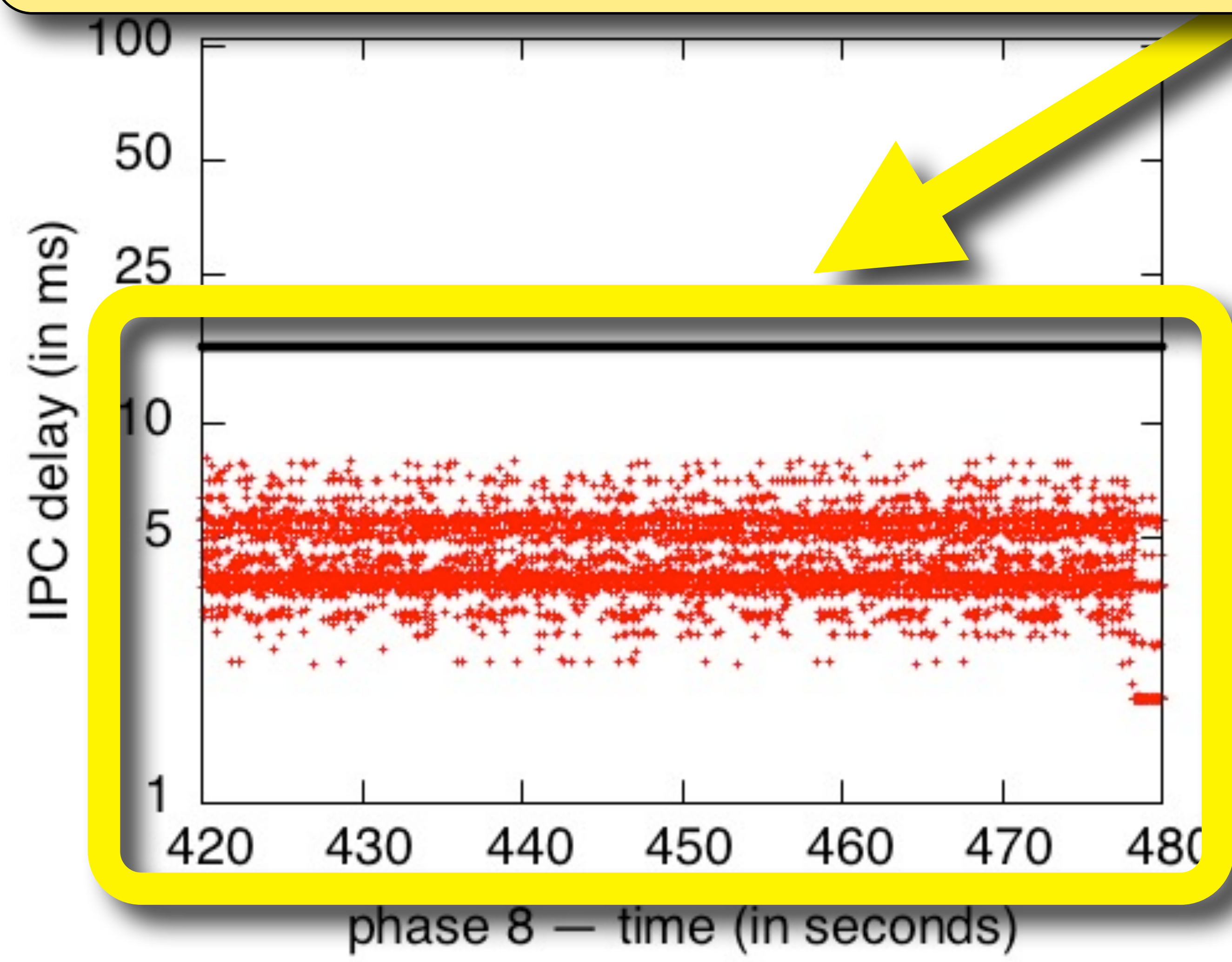
➡ Best-effort tasks have lower priority ➔ **no additional delays**.

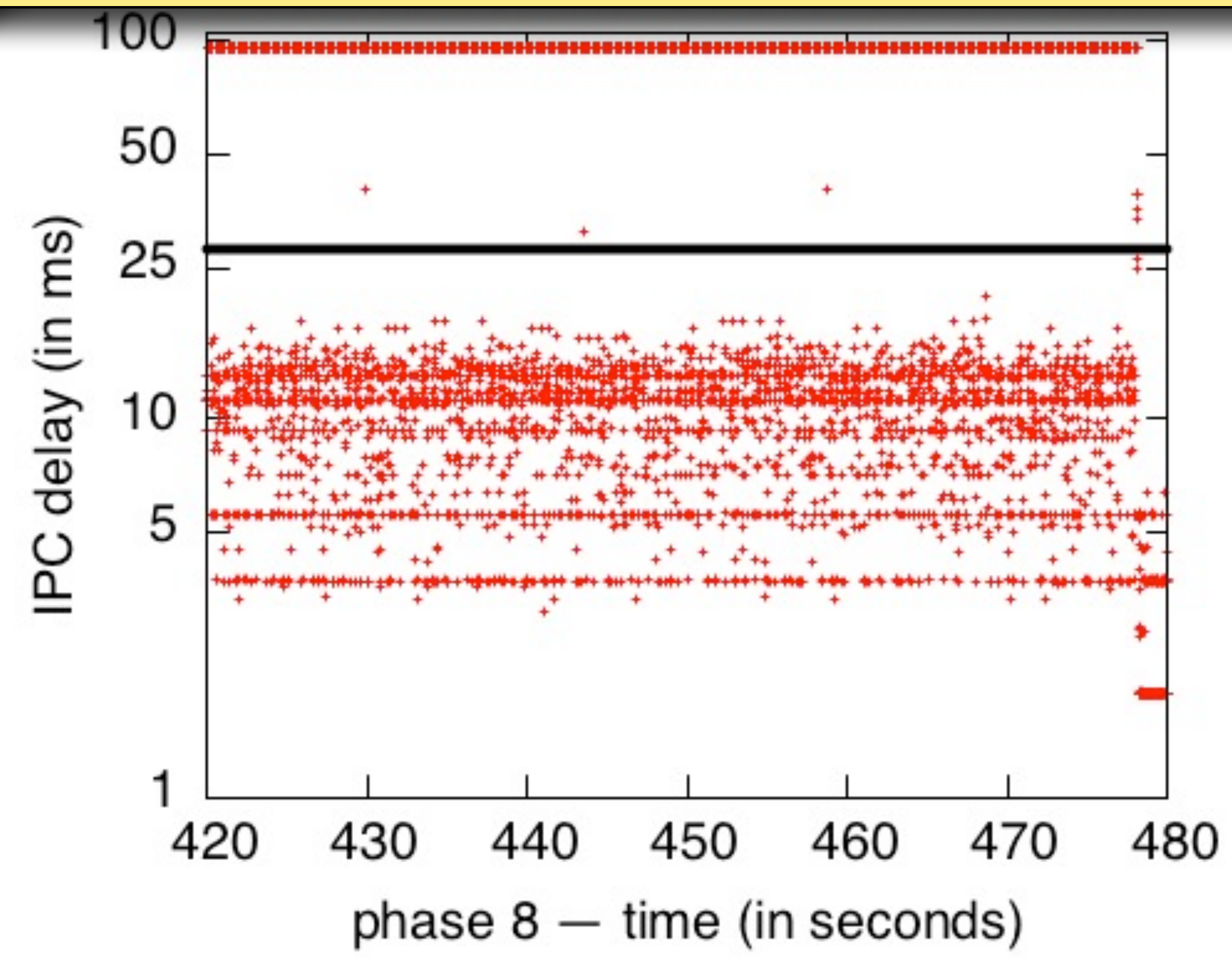
MC-IPC

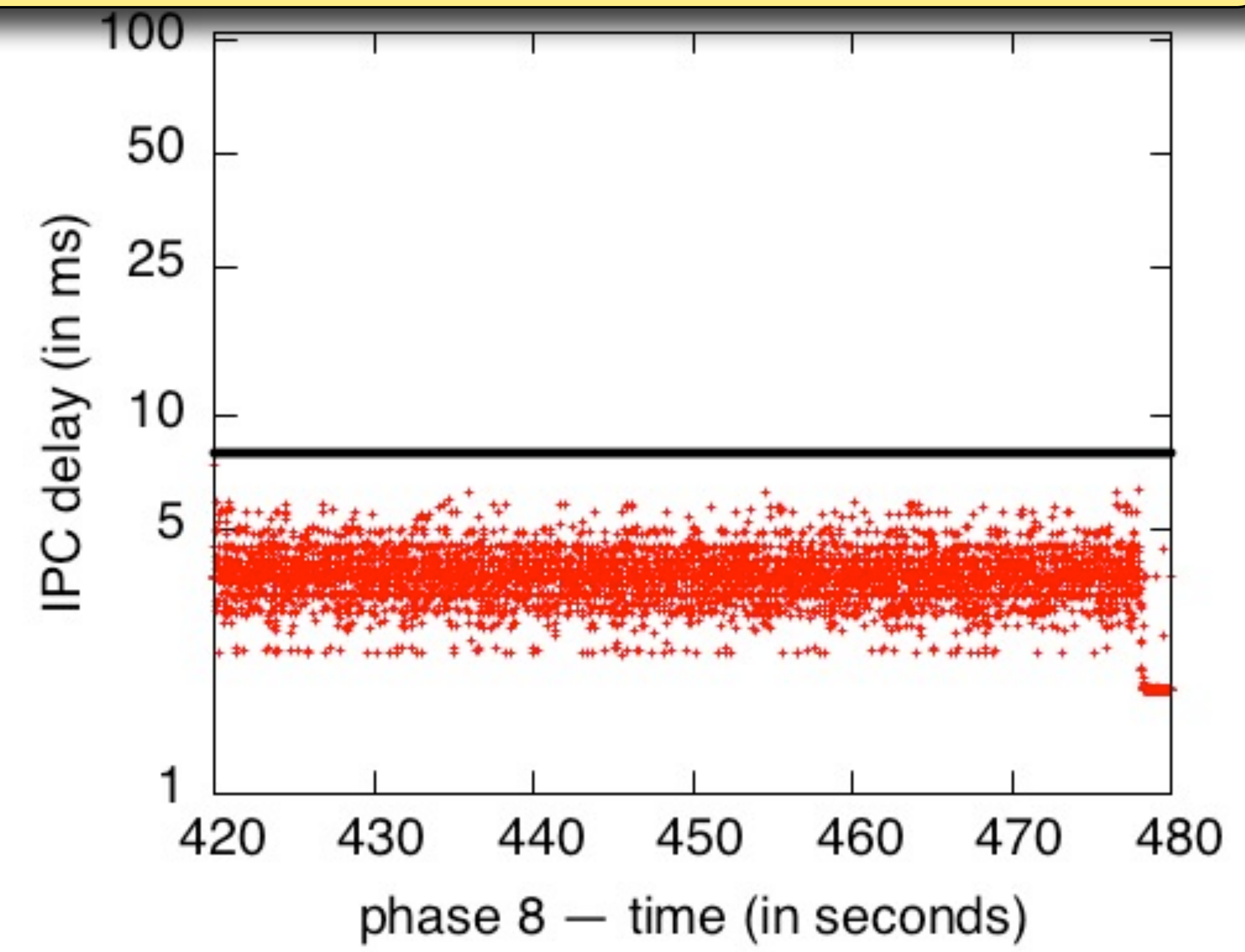
➡ Handles best-effort tasks explicitly ➔ analysis **accounts** for best-effort tasks.

MC-IPC: combines advantages of FIFO-IPC and PRIO-IPC.

*Like PRIO-IPC w.r.t. to **lower-priority/lower-criticality tasks**,*
*like FIFO-IPC w.r.t. **monopolization attempts** and **starvation effects**.*







Best-effort tasks create contention for resource used by real-time tasks

➡ With FIFO-IPC, need to **place trust** in maximum number of background best-effort tasks.

PRIO-IPC

➡ Best-effort tasks have lower priority ➔ **no additional delays**.

MC-IPC

➡ Handles best-effort tasks explicitly ➔ analysis **accounts** for best-effort tasks.