Analysis of Memory-Contention in Heterogeneous COTS MPSoCs

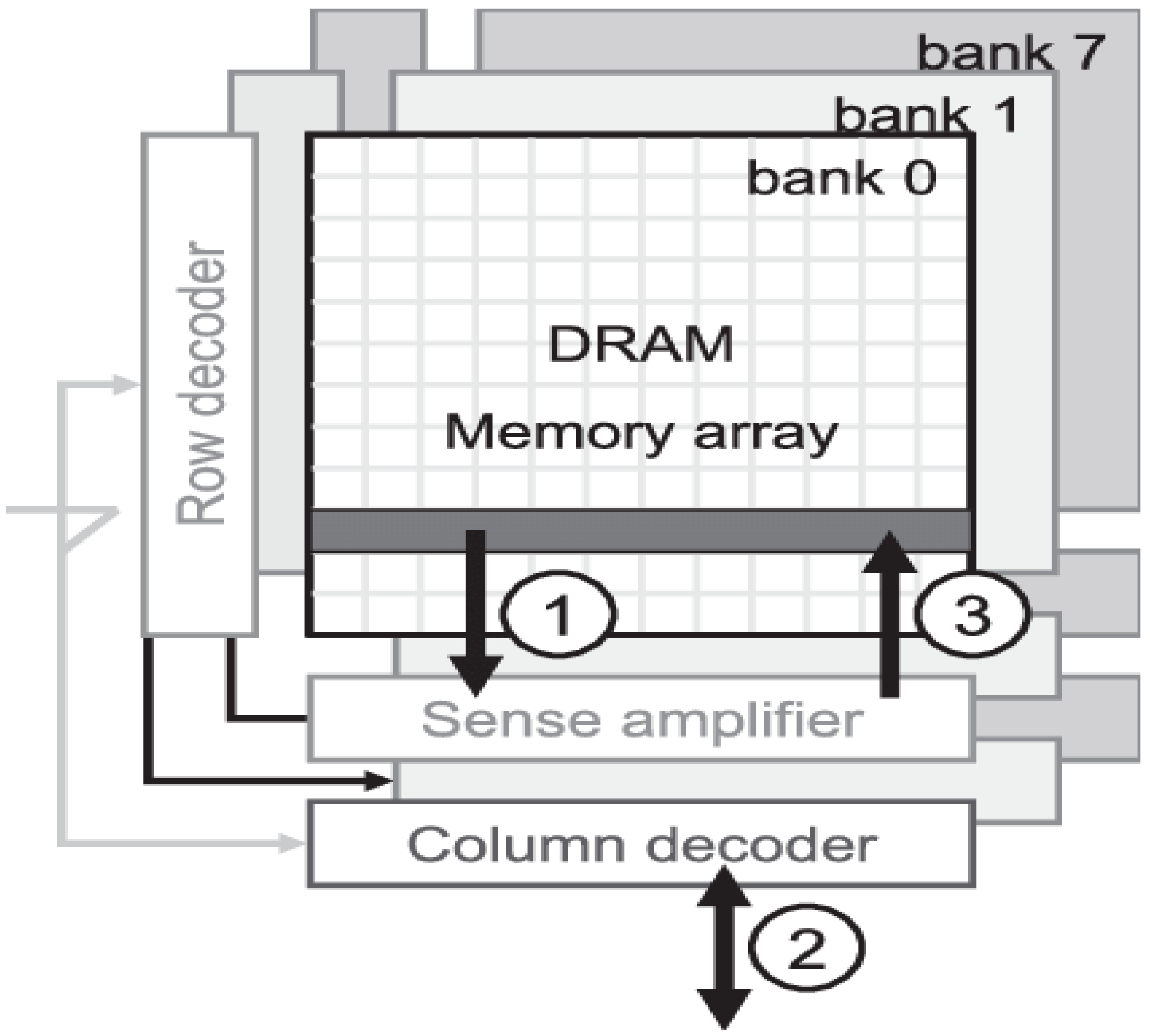
Mohamed Hassan and Rodolfo Pellizzoni

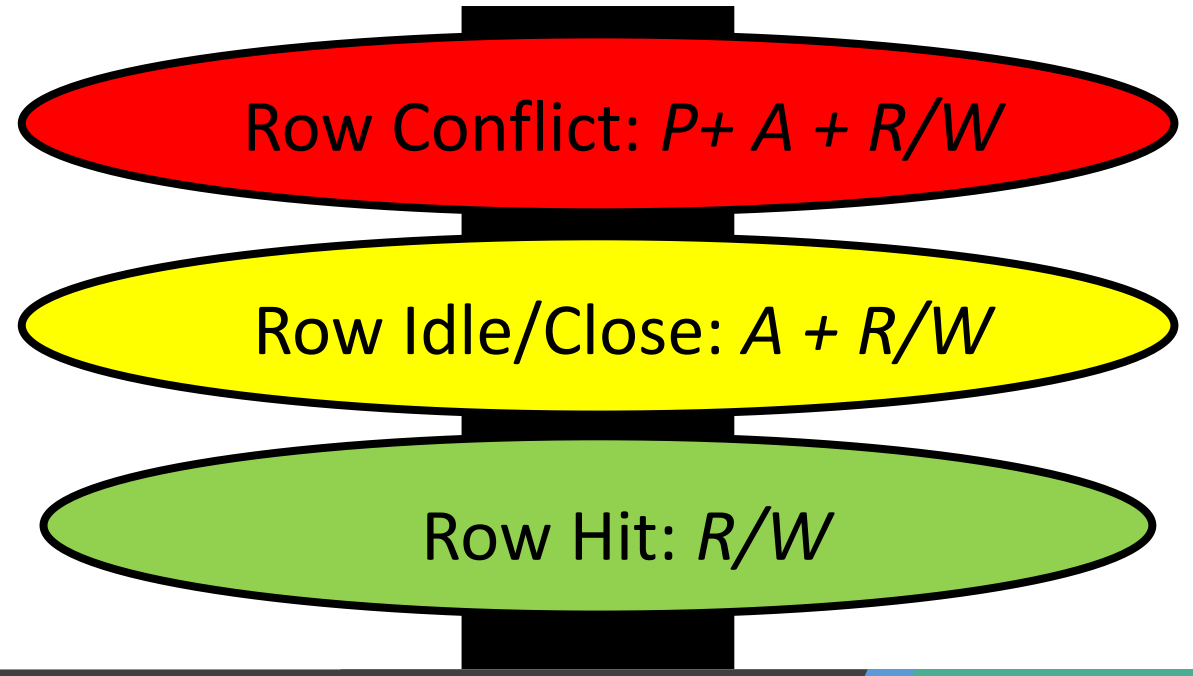
https://gitlab.com/FanusLab/memory-contention-analysis


DRAMs
01
Big Picture
02
Motivation
03
Proposed
04
Results
05

Outline

- DRAM Consists of multiple banks
- The memory controller (MC) manages accesses to DRAM
- A request in general consists of:
  - ACTIVATE (A) command:
    - Bring data row from cells into sense amplifiers
  - Read/Write (R/W) commands:
    - To read/write from specific columns in the sense amplifiers
  - PRECHARGE (P) command:
    - to write back a previous row in the sense amplifiers before bringing the new one



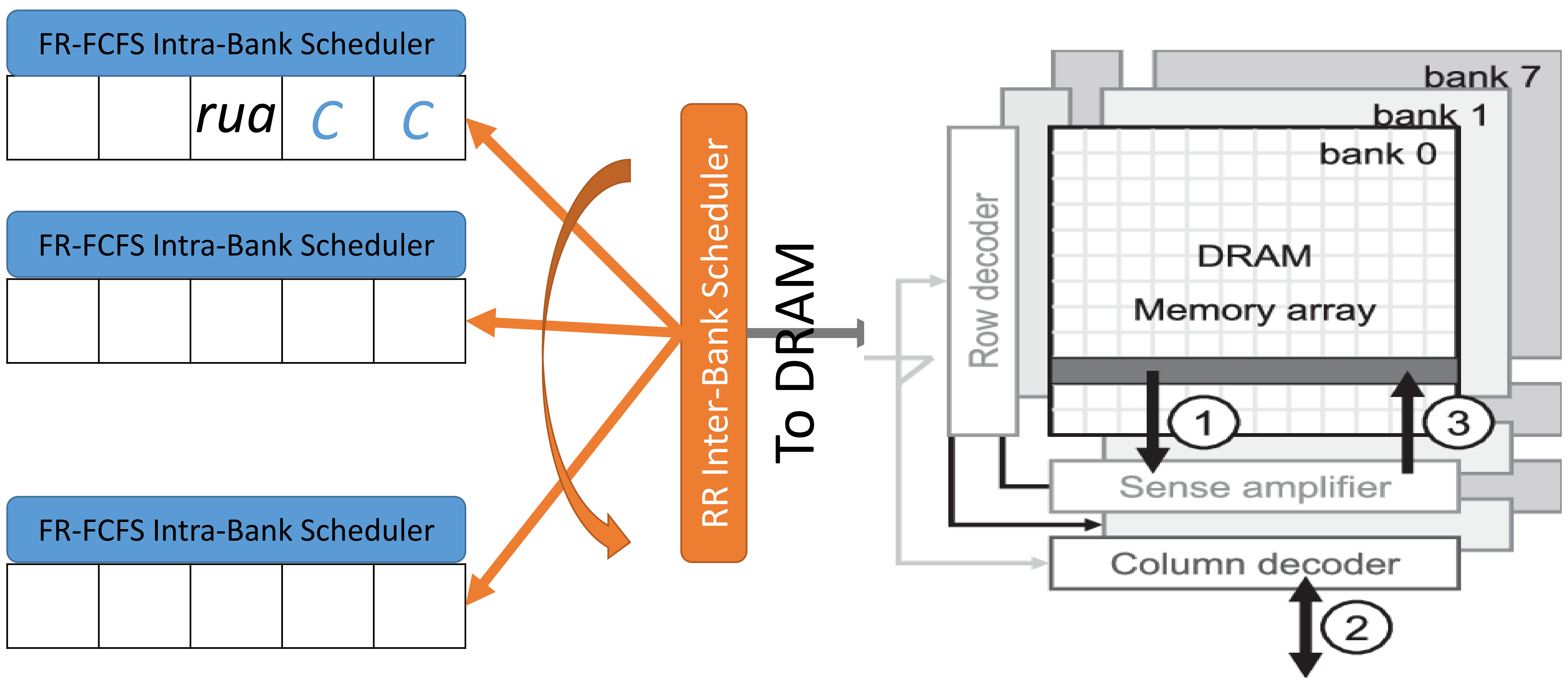


Background

DRAM

Intra-bank interfering requests:

1. Intra-bank conflict requests

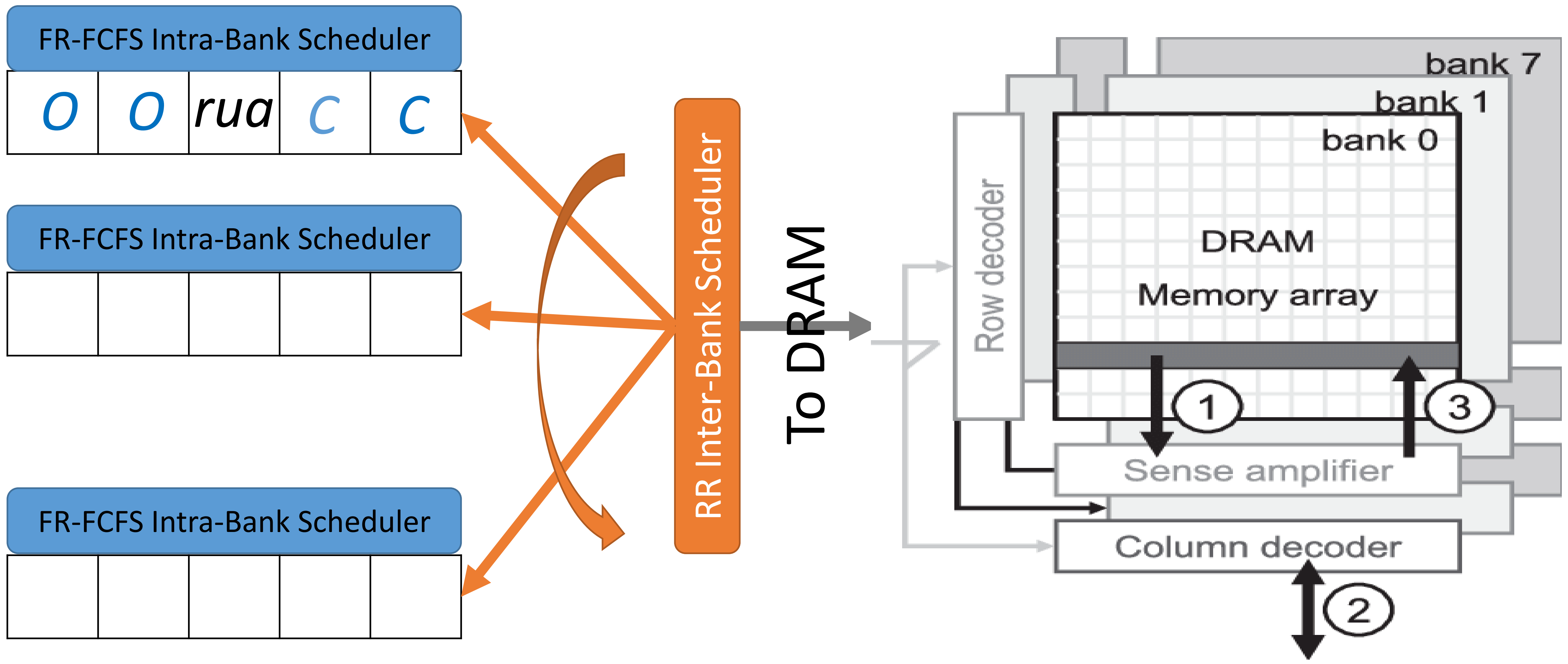


Request Types

Big Picture

Intra-bank interfering requests:

1. Intra-bank conflict requests
2. Intra-bank reorder requests



Request Types

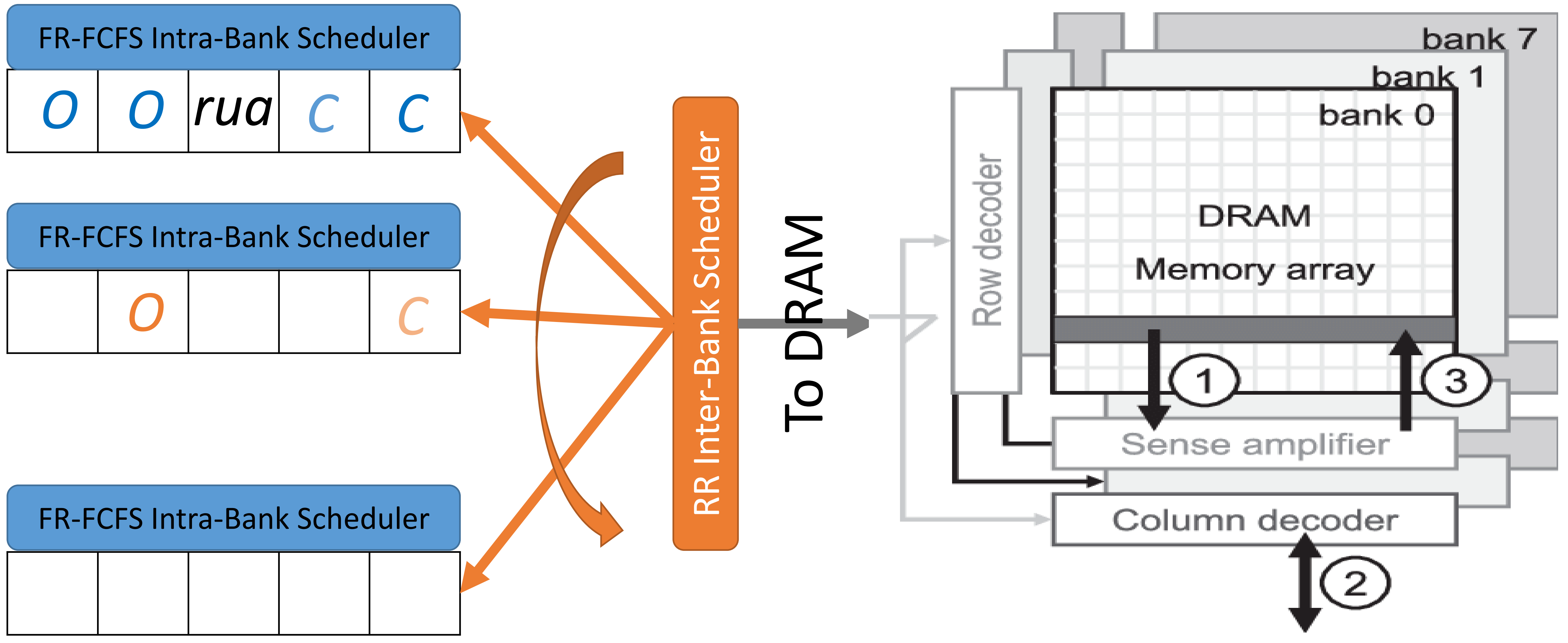
Big Picture

Intra-bank interfering requests:

1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:

3. Inter-bank close requests
4. Inter-bank open requests



Request Types

Big Picture

Intra-bank interfering requests:

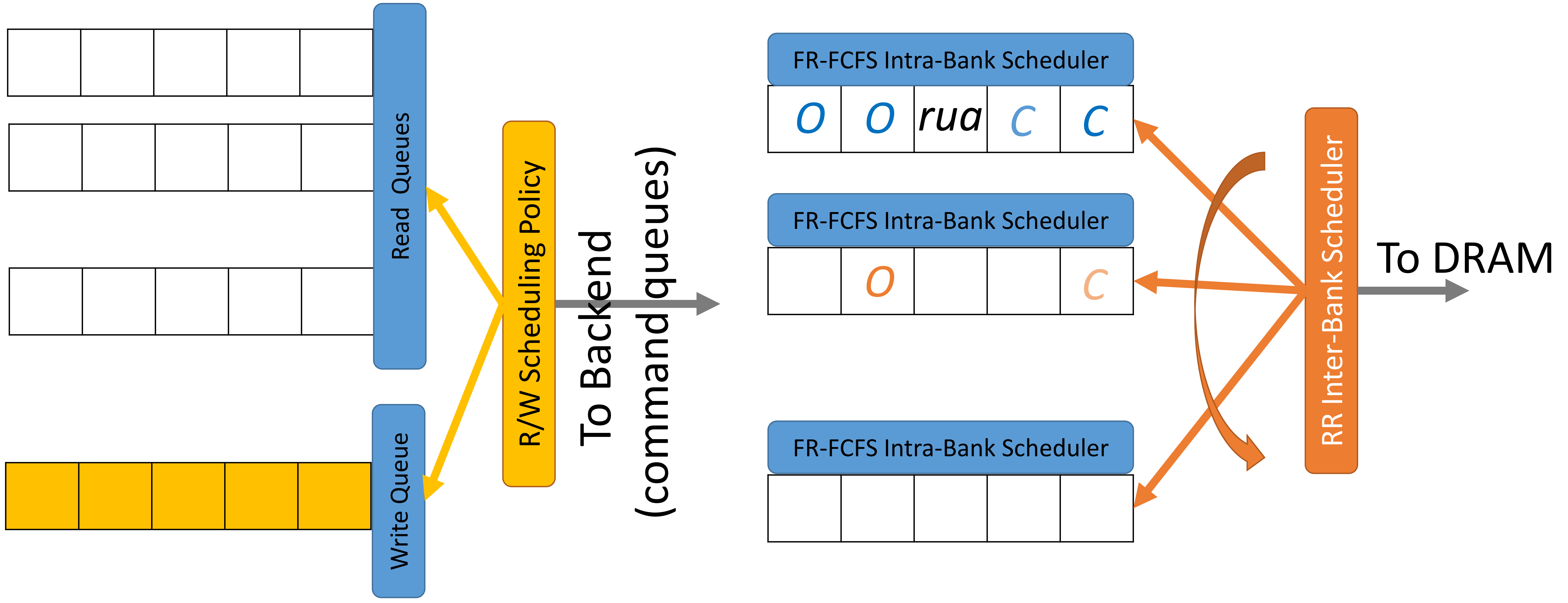
1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:

3. Inter-bank close requests
4. Inter-bank open requests

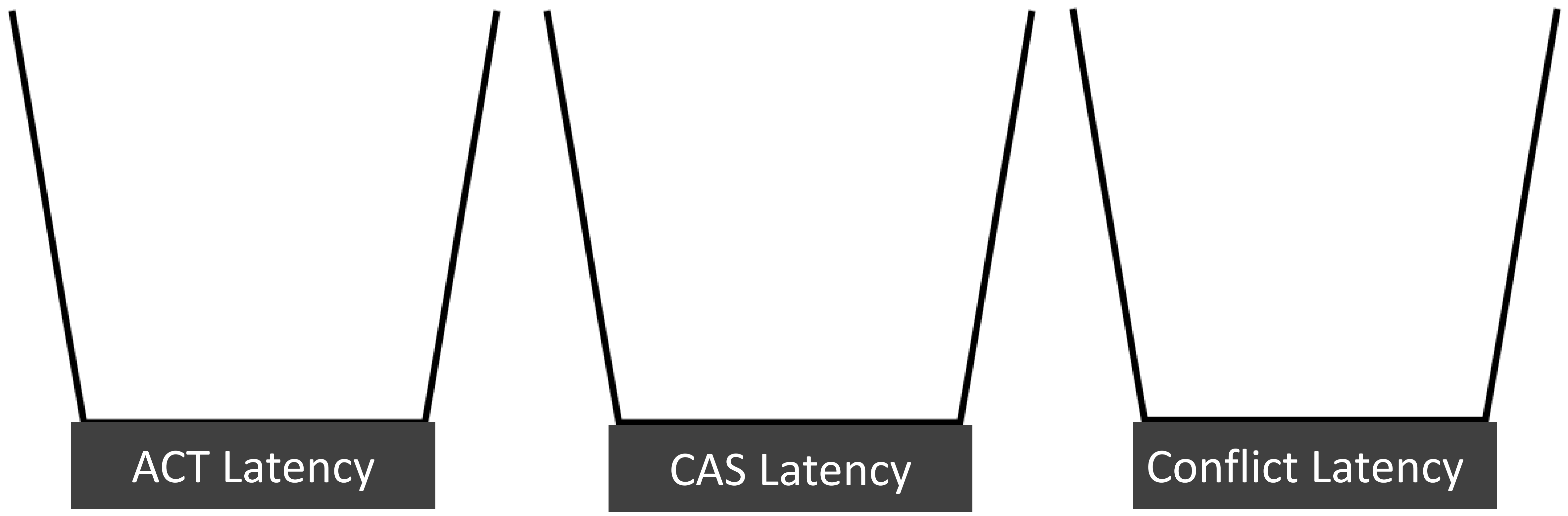
Write batching

5. Write batching requests



Request Types

Big Picture



Latency Buckets (Components)

Big Picture

Intra-bank interfering requests:

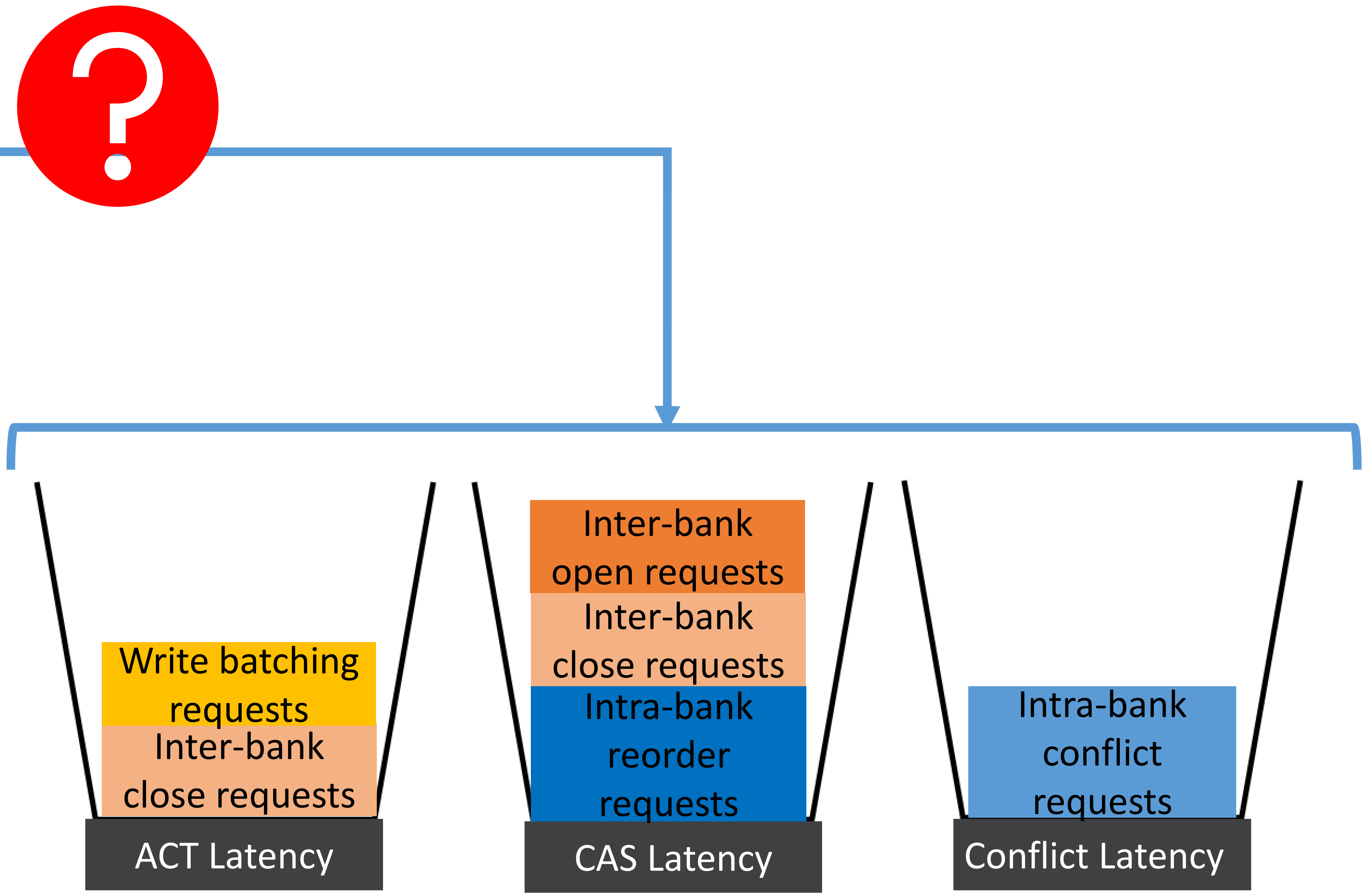
1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:

3. Inter-bank close requests
4. Inter-bank open requests

Write batching

5. Write batching requests



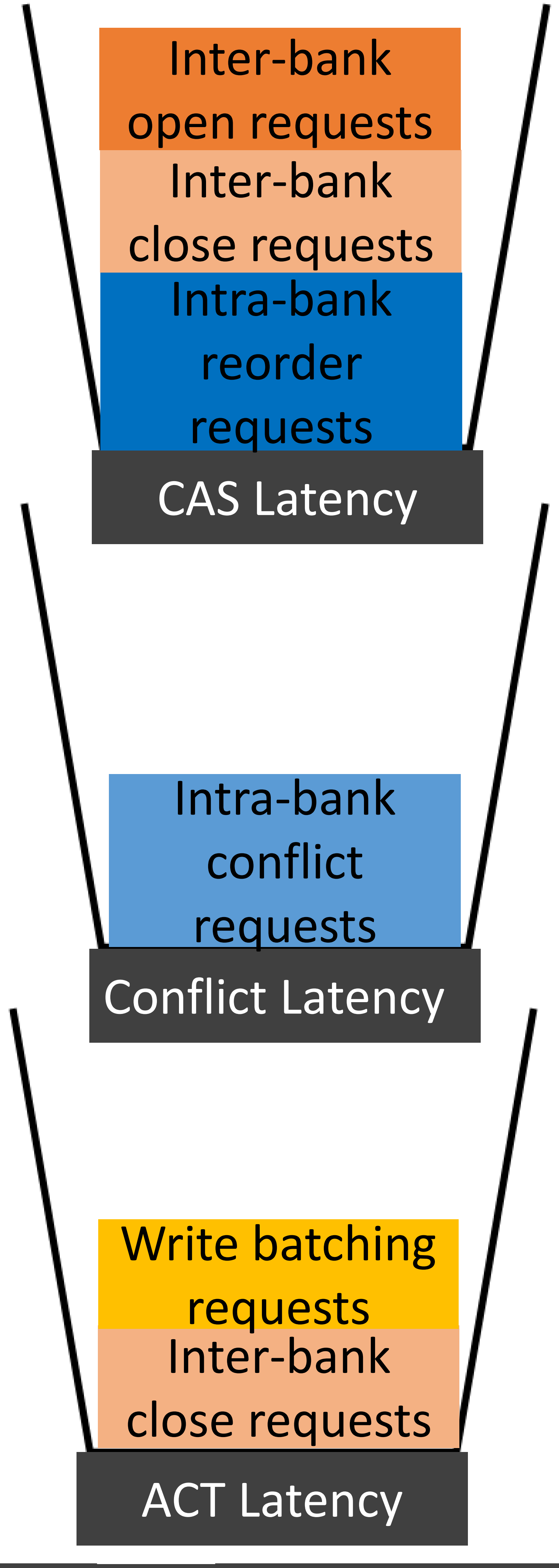
Latency Buckets (Components)

Big Picture

Request-Dr Analysis

- What is the worst-case of each of these components can be suffered by a single request? → ***$WCL^{per\text{-}req}$***
- Assuming nothing at all about interfering tasks
  - (i.e., infinite number of interfering requests)
- Then obtain total memory latency assuming we know **the total number of interfered requests**

  → ***$WCL^{tot} = \#Reqs \times WCL^{per\text{-}req}$***



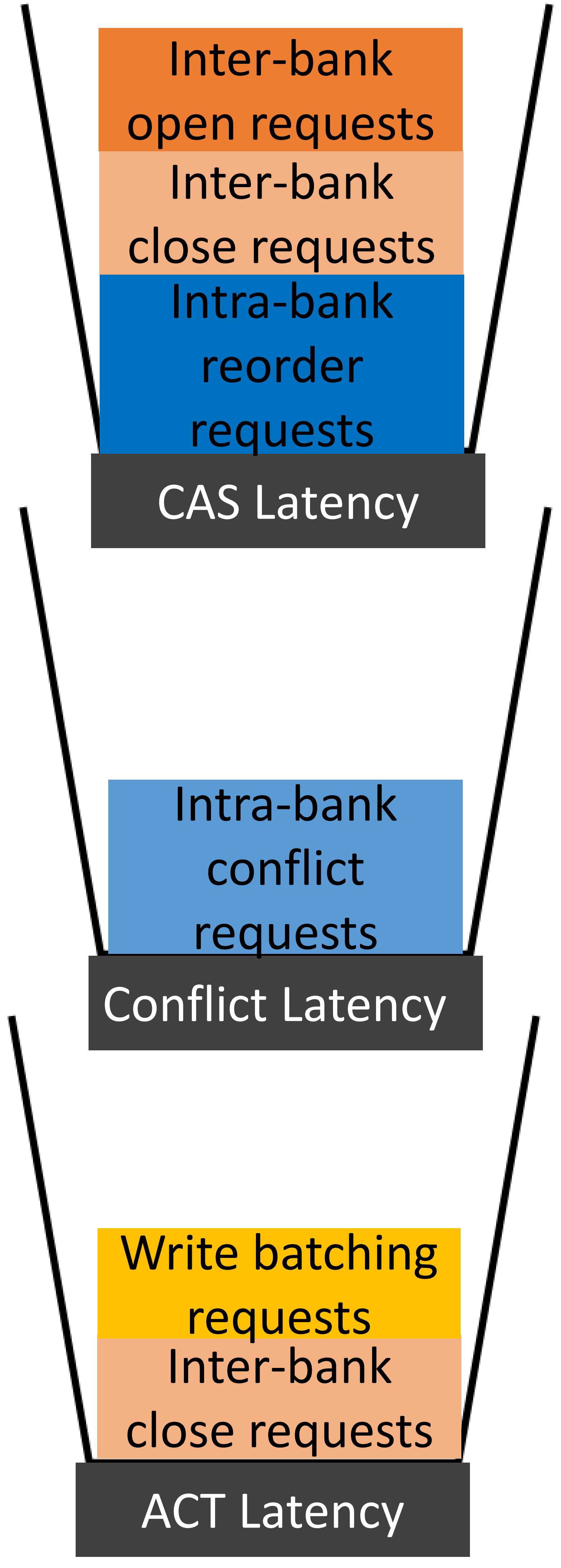
Request-Driven vs Job-Driven Analysis

Motivation

Request-Dr Analysis

- What is the worst-case of each of these components can be suffered by a single request? → ***$WCL^{per\text{-}req}$***
- Assuming nothing at all about interfering tasks
  - (i.e., infinite number of interfering requests)
- Then obtain total memory latency assuming we know **the total number of interfered requests**

→ ***$WCL^{tot} = \#Reqs \times WCL^{per\text{-}req}$***

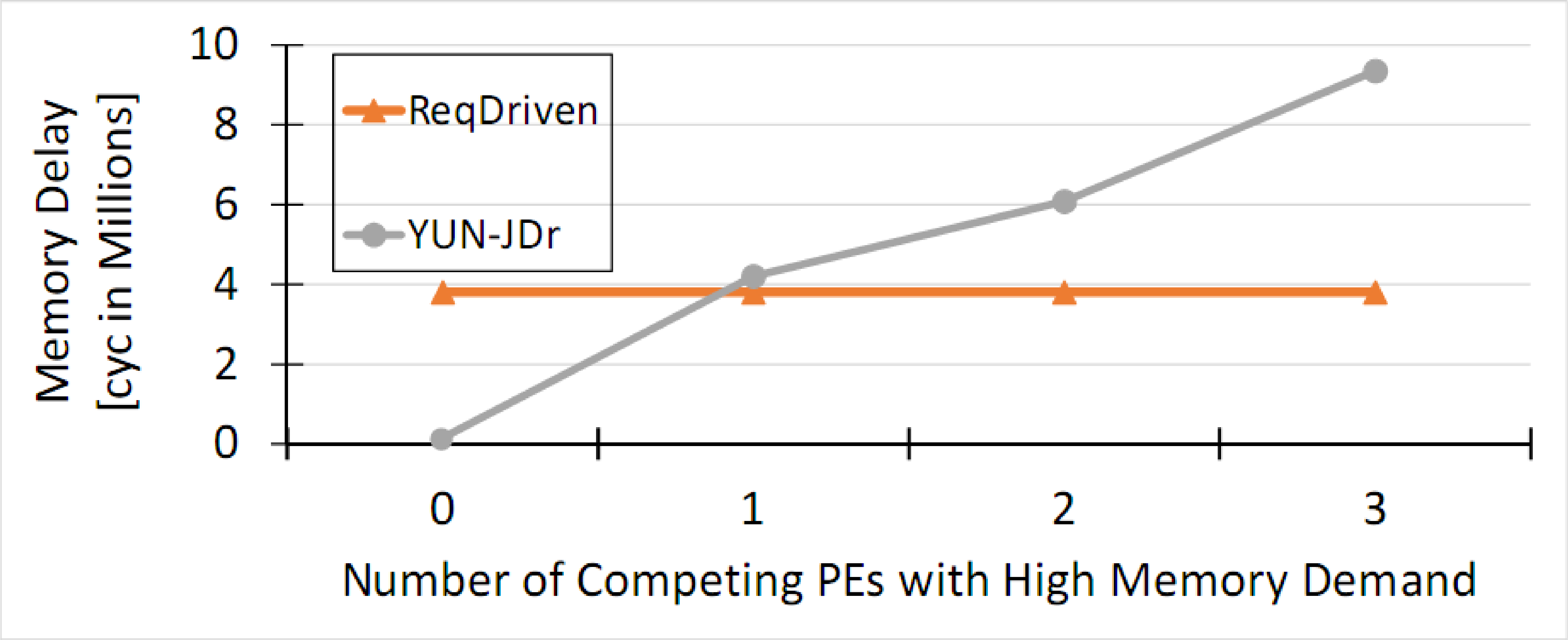


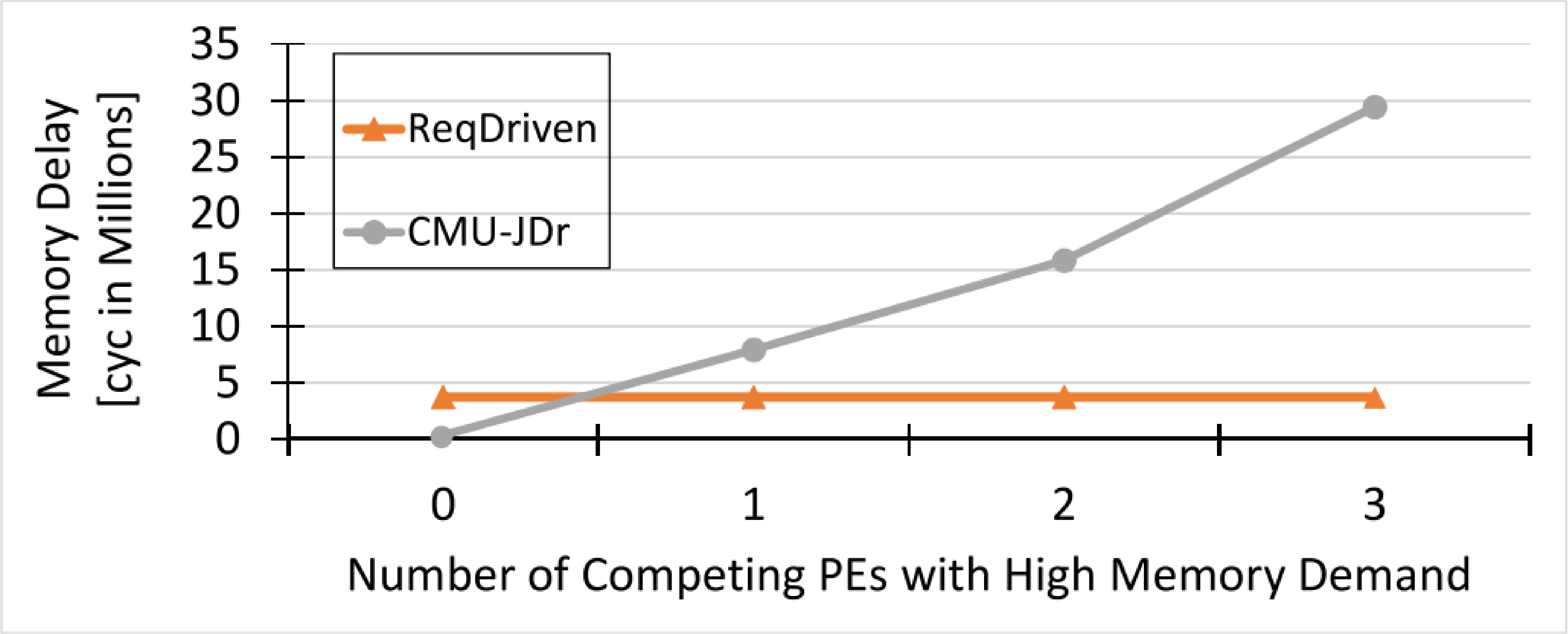
Job-Dr Analysis

- What is the worst-case of each of these components can be suffered by the total task assuming **we know the number of interfering requests**?
- Assuming nothing at all about # interfered requests
  - (i.e., infinite number of interfered requests)

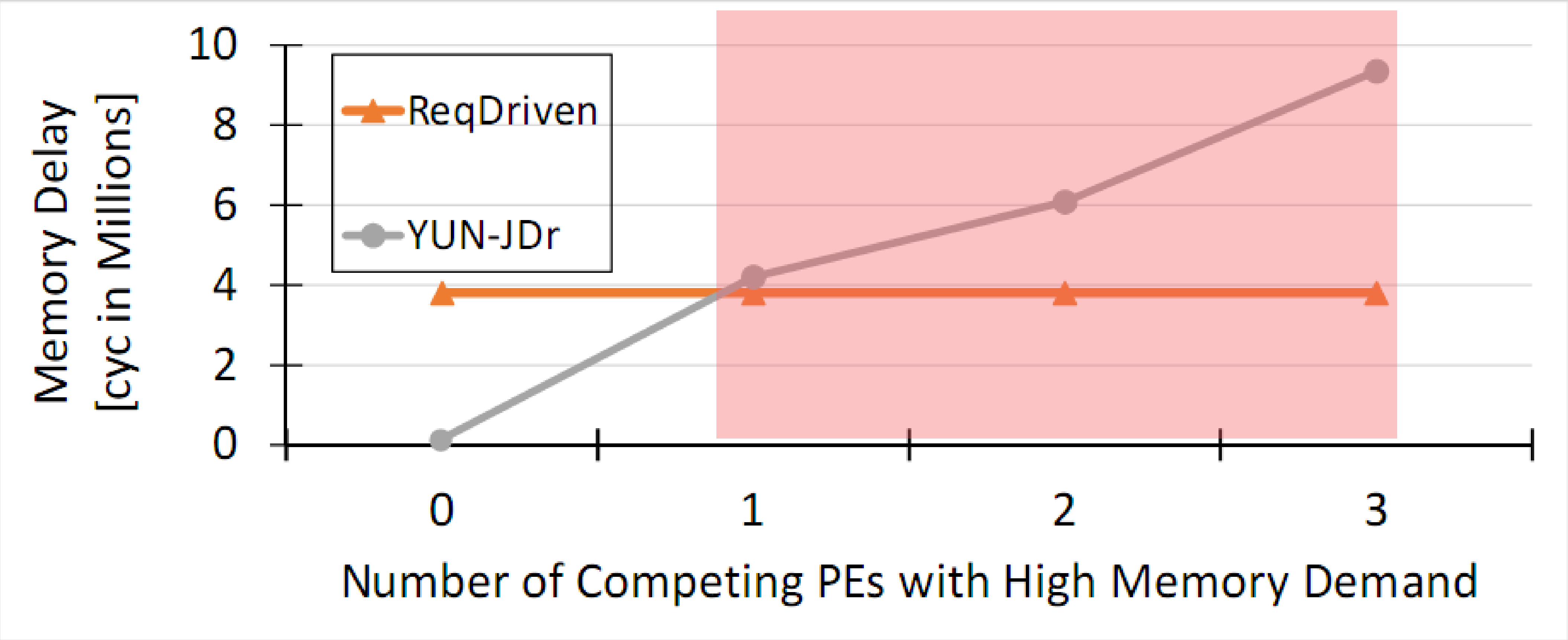
Request-Driven vs Job-Driven Analysis

Motivation



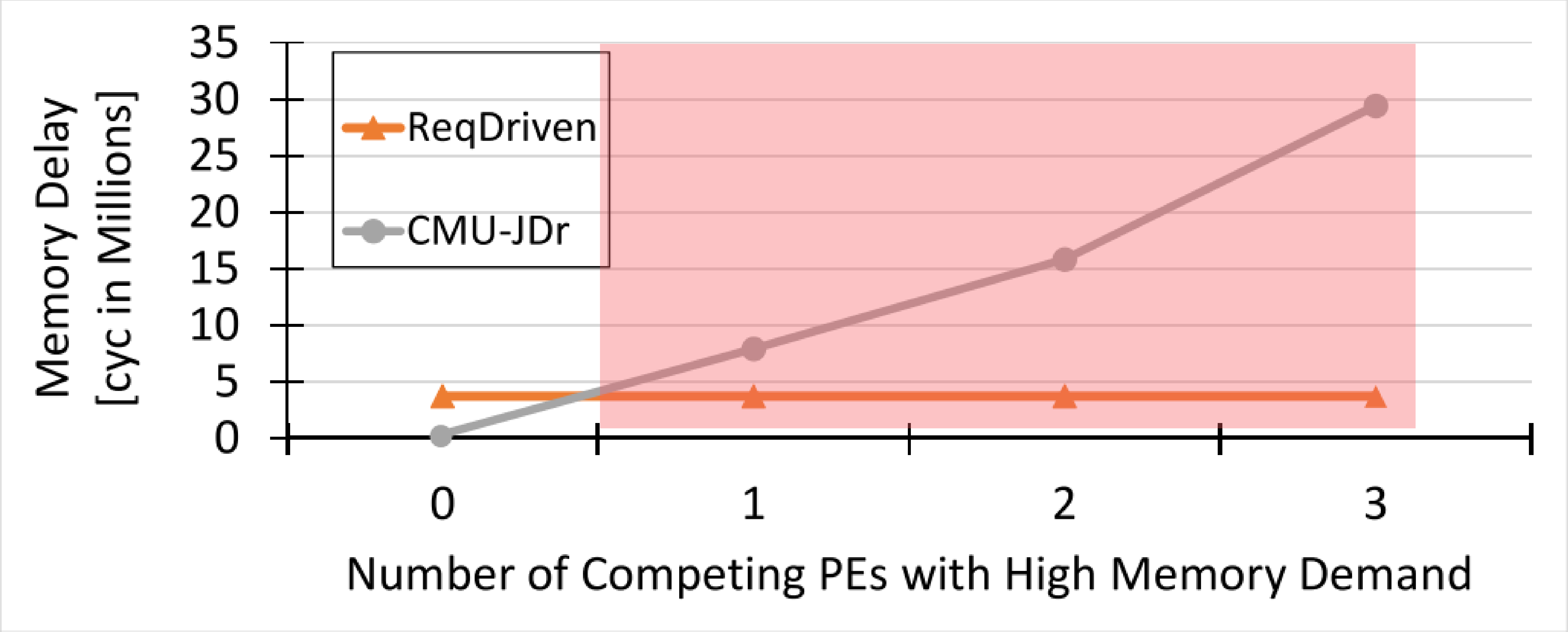


Request-Driven vs Job-Driven Analysis



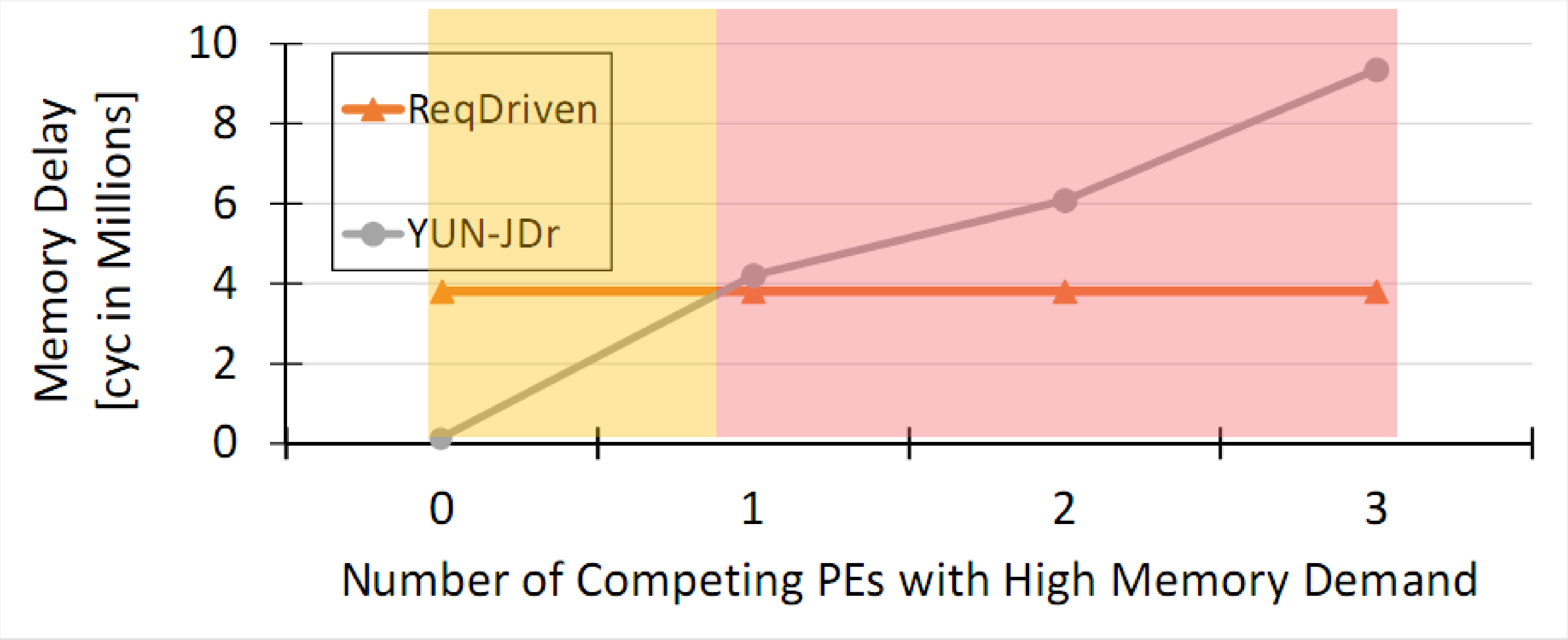
Relatively small number of ***interfered*** requests

→ Req-Dr wins

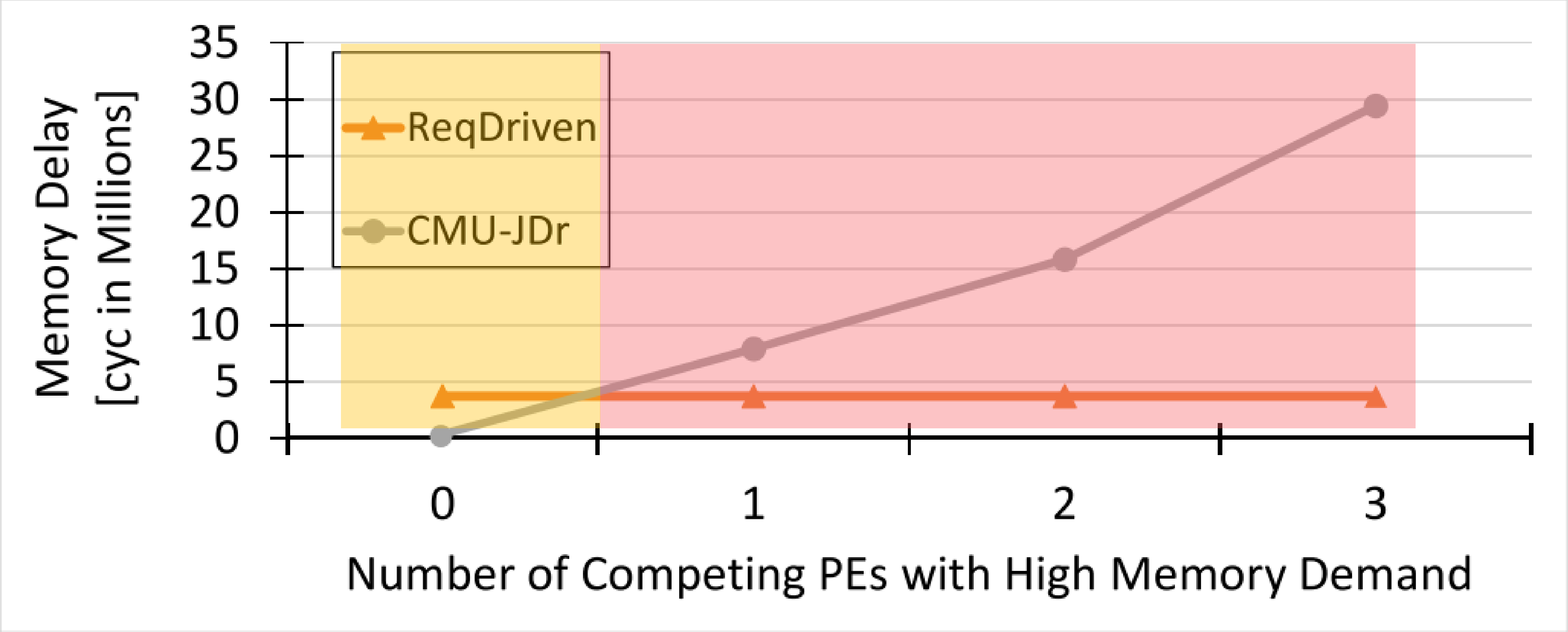


Request-Driven vs Job-Driven Analysis

Motivation



Relatively small number of ***interfered*** requests
→ Req-Dr wins



Relatively small number of ***interfering*** requests
→ Job-Dr wins

Request-Driven vs Job-Driven Analysis

Motivation

State-of-the-art

Motivation

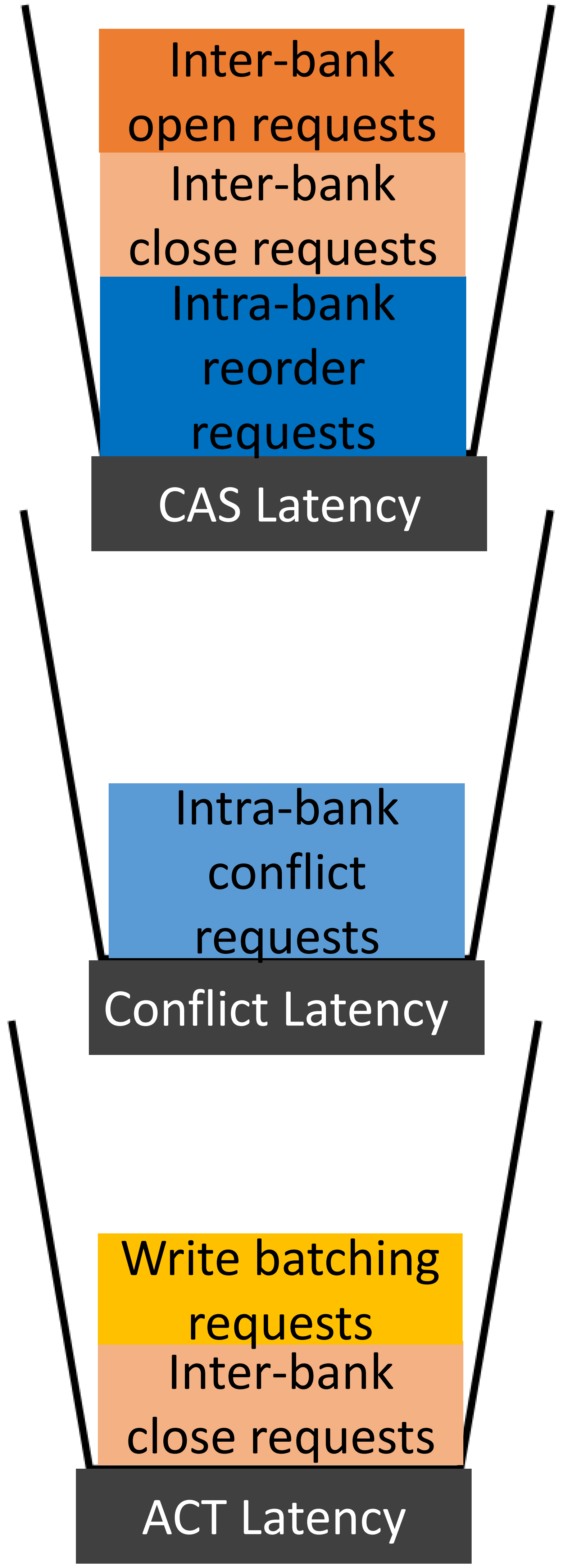
- **[CMU Req- and Job-Dr]** Hyoseung Kim et al. Bounding memory interference delay in COTS-based multi-core systems. RTAS, 2014.
  - Both request- and job-driven analysis
  - A specific COTS platform

- **[Yun Req- and Job-Dr]** Heechul Yun, Rodolfo Pellizzon, and Prathap Kumar Valsan. Parallelism-aware memory interference delay analysis for COTS multicore systems. ECRTS, 2015.
  - Both request- and job-driven analysis
  - A specific COTS platform

- **[Hassan Req-Dr]** Mohamed Hassan and Rodolfo Pellizzoni. Bounding DRAM interference in COTS heterogeneous MPSoCs for mixed criticality systems, EMSOFT, 2018
  - Explores a wide variety of COTS possible configurations (144 platform instances)
  - Only request-driven analysis

What do we do?

- A task-aware
  - COTS-aware
    - Hybrid analysis

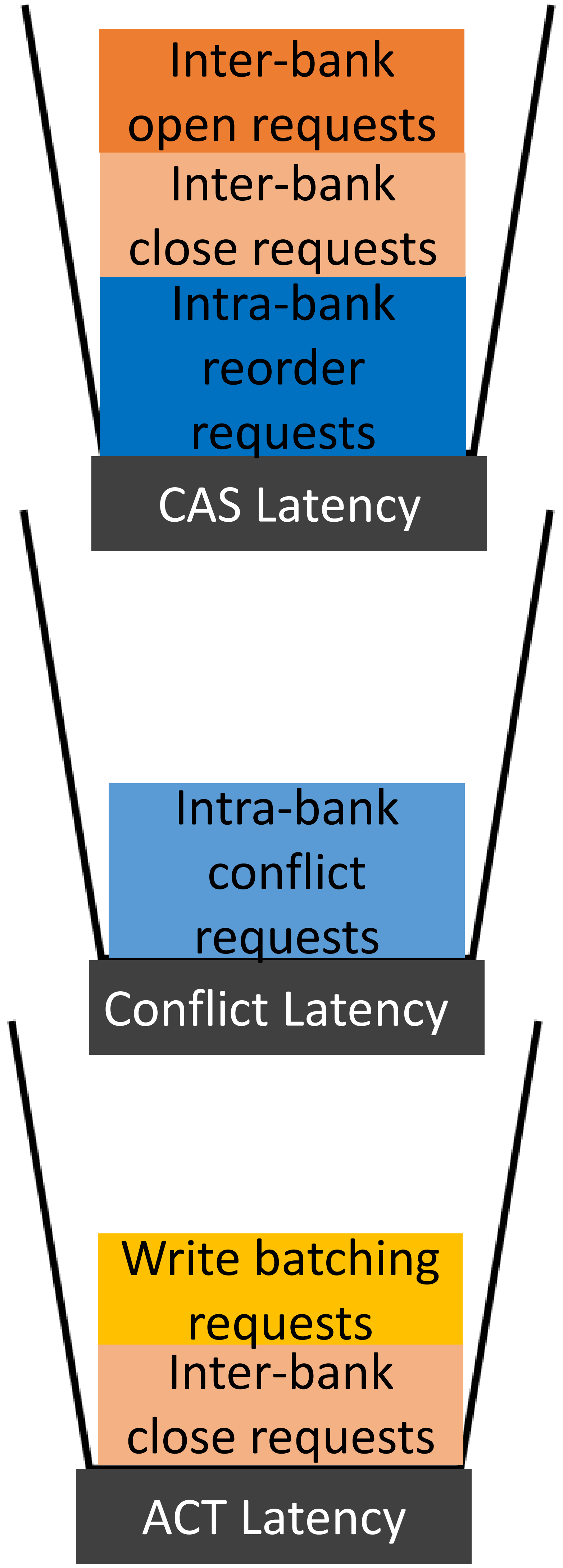


This work

Proposed

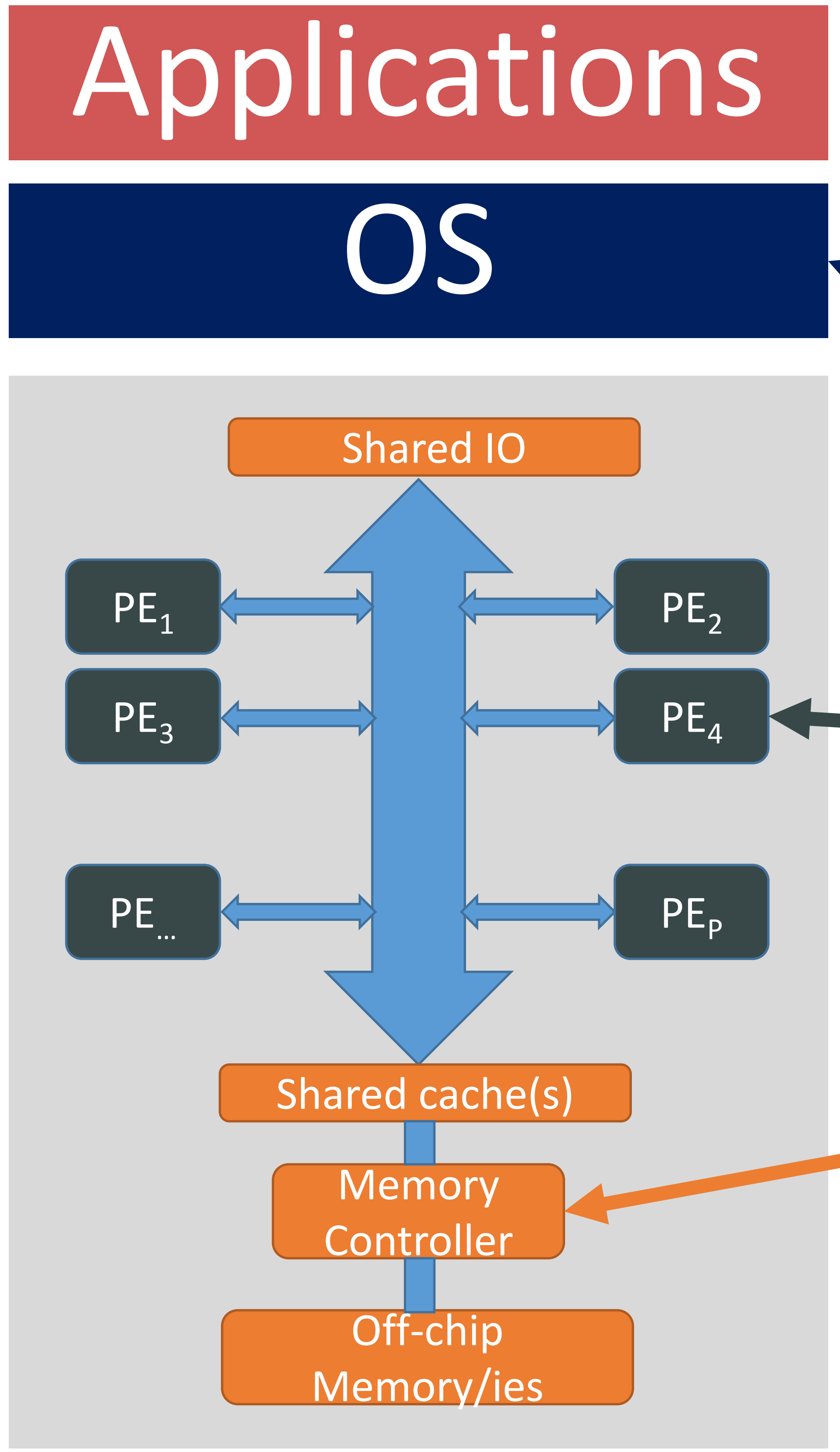
What do we do?

- A task-aware COTS-aware Hybrid analysis
- ***Task-aware:***
  - Account for deferent level of knowledge we have about running tasks:
    - Total number of requests
    - Total number of reads + writes
    - Total number of open (row hits) + close (row misses) requests



Task-Aware Analysis

Proposed



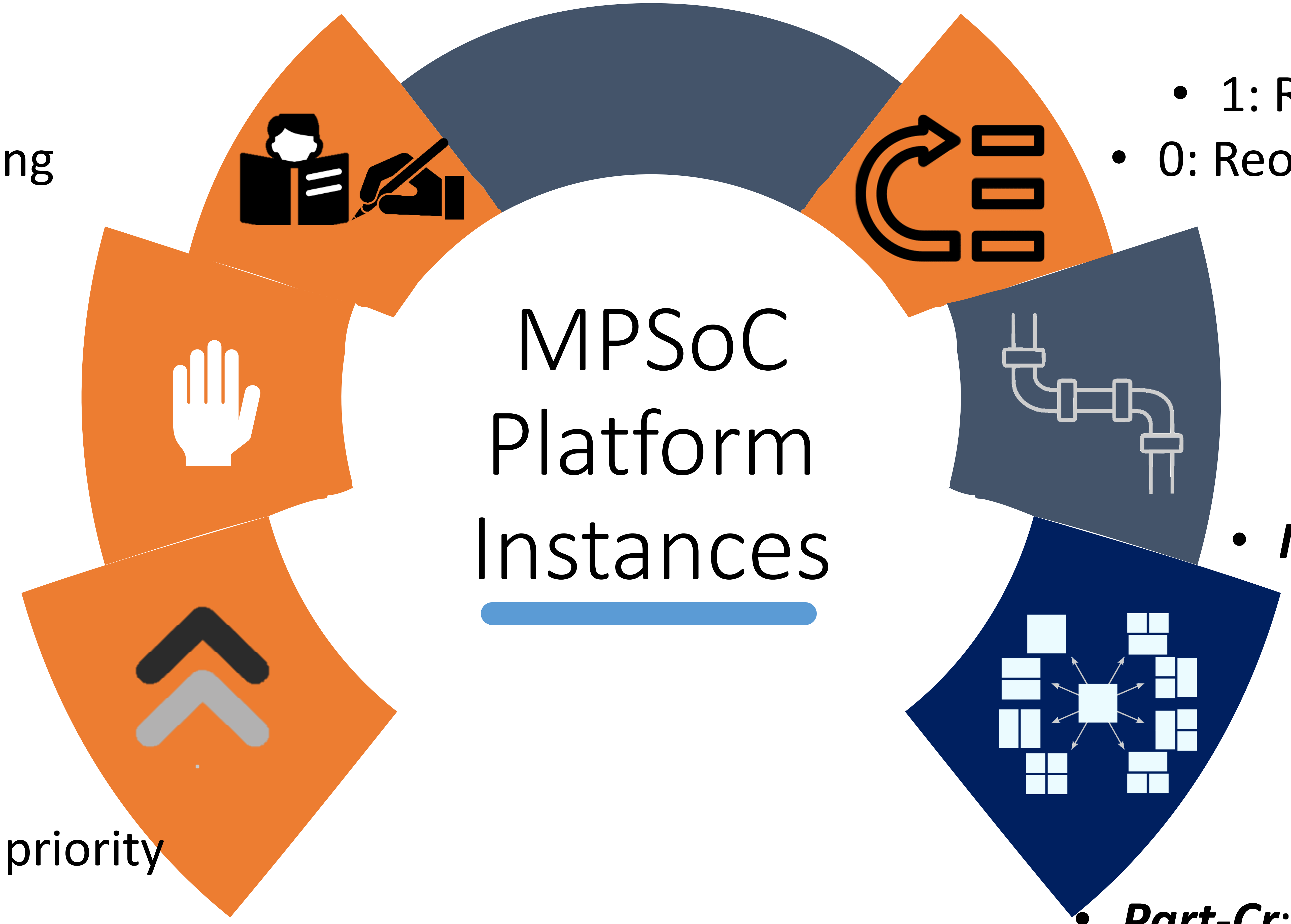
Memory Behavior Depends on?:

- OS Configuration
- PE Architecture
- MC Policies

- Priority:
  - PEs can be given priorities
  - COTS platforms support different priority levels
  - Existing analysis does not account for this
- Intra-bank scheduling
  - FR-FCFS
  - COTS also supports a threshold on reordering to prevent starvation
- Inter-bank scheduling
  - RR across banks
  - Two flavors:
    - Always schedule ready commands of any type (high performance)
    - Reorder only commands of different type (prevent starvation)
- Read/Write arbitration, two flavors:
  - Reads and writes have same priority
  - Serve in batches, where reads have higher priority

COTS-Aware Analysis

Proposed



R/W Reorder
- 1: write batching
- 0: no write batching

Inter-bank Reorder
- 1: Reorder across all commands
- 0: Reorder commands of diff types

FR-FCFS Threshold
- 1: FR-FCFS is capped
- 0: no cap on FR-FCFS

Pipeline
- ***IO-All***: All PEs are In-order
- ***IO-Cr***: Critical PEs are in-order
- ***OOO-All***: All PEs are OOO

Priority
- 1: Critical PEs are higher priority
- 0: no priority

Partitioning
- ***No-Part***: No Partitioning
- ***Part-Cr***: Partition among critical apps
- ***Part-All***: Partition among all apps

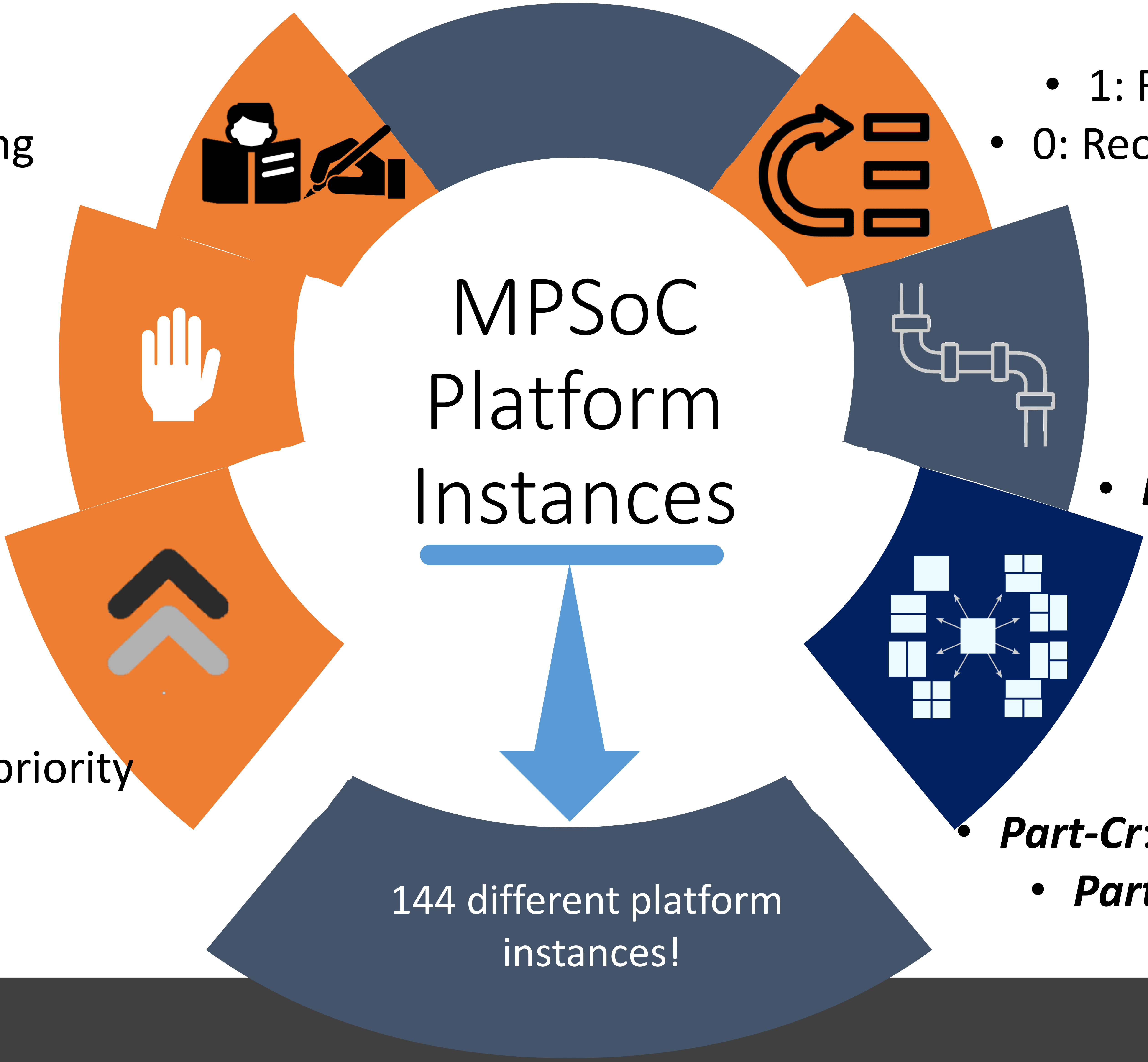
COTS-Aware Analysis

Proposed

R/W Reorder
- 1: write batching
- 0: no write batching

Inter-bank Reorder
- 1: Reorder across all commands
- 0: Reorder commands of diff types

FR-FCFS Threshold
- 1: FR-FCFS is capped
- 0: no cap on FR-FCFS



Pipeline
- ***IO-All***: All PEs are In-order
- ***IO-Cr***: Critical PEs are in-order
- ***OOO-All***: All PEs are OOO

Priority
- 1: Critical PEs are higher priority
- 0: no priority

Partitioning
- ***No-Part***: No Partitioning
- ***Part-Cr***: Partition among critical apps
- ***Part-All***: Partition among all apps

COTS-Aware Analysis

Proposed

What do we do?

- A task-aware COTS-aware Hybrid analysis
- ***Hybrid:***
  - **State-of-the-art**: only running request- or job-Dr analysis or run both and take the min

  - **This work**: construct an optimization framework that blends both request-level and task-level per-core constraints to obtain tighter bounds

Hybrid Analysis

Proposed

Intra-bank interfering requests:

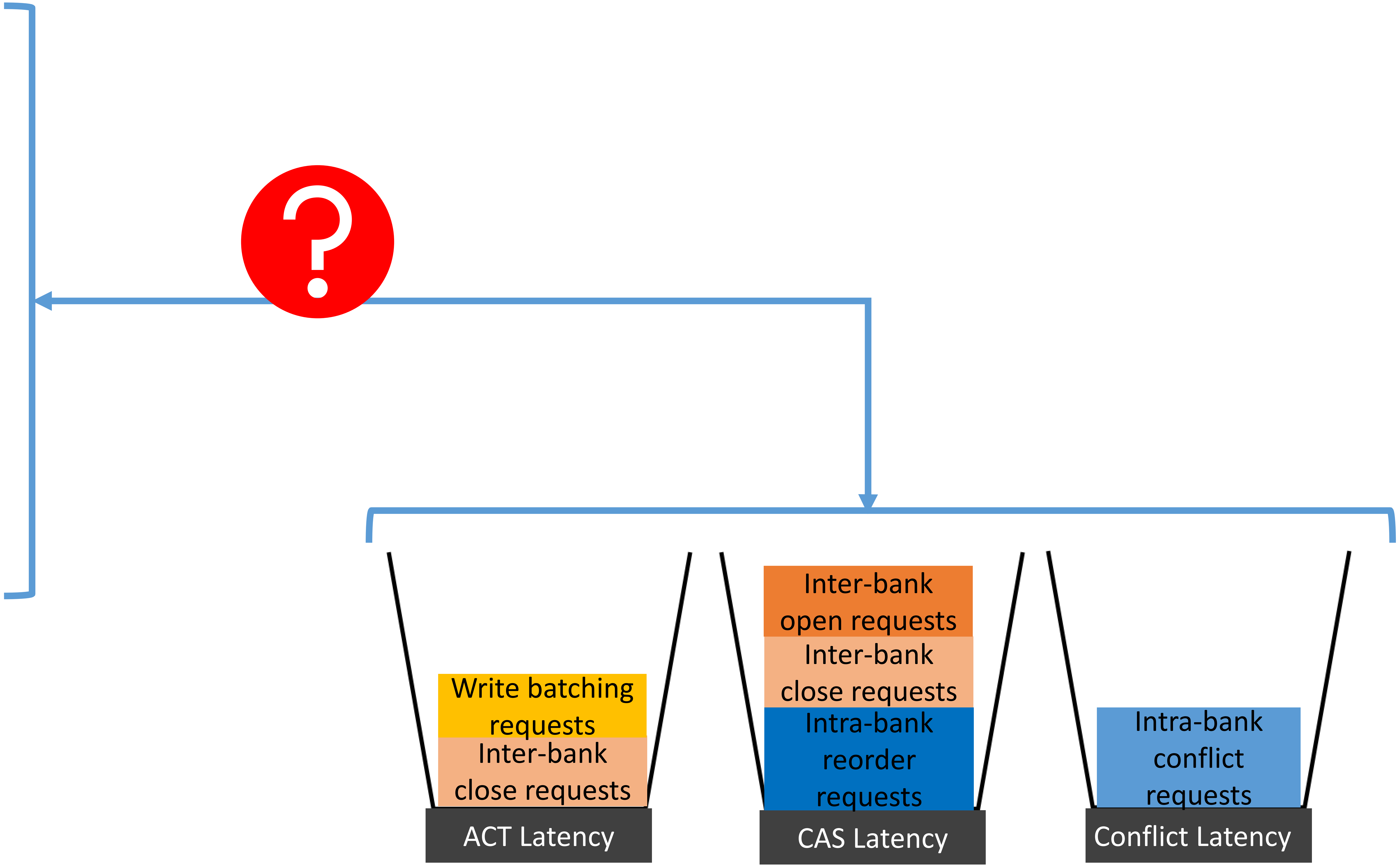
1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:

3. Inter-bank close requests
4. Inter-bank open requests

Write batching

5. Write batching requests



Conflict Req

Proposed

Intra-bank interfering requests:

1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:

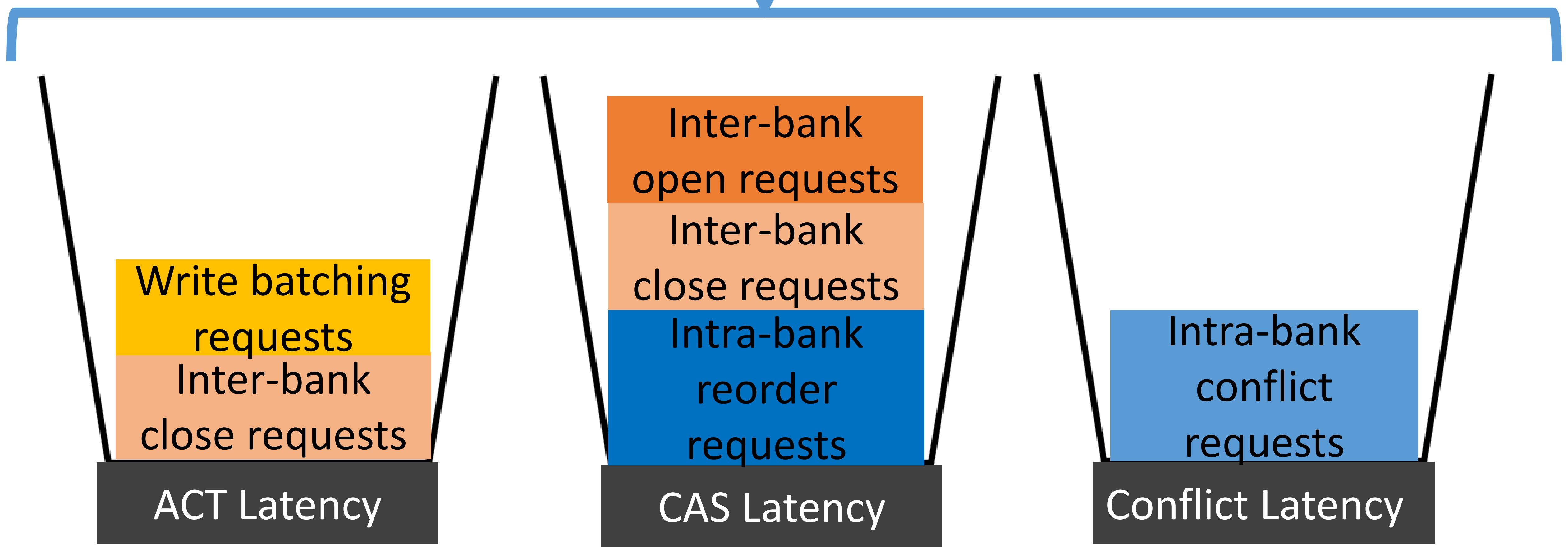
3. Inter-bank close requests
4. Inter-bank open requests

Write batching

5. Write batching requests

Optimization problem:

1. Write Latency components as functions on those requests



Conflict Req

Proposed

Intra-bank interfering requests:

1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:

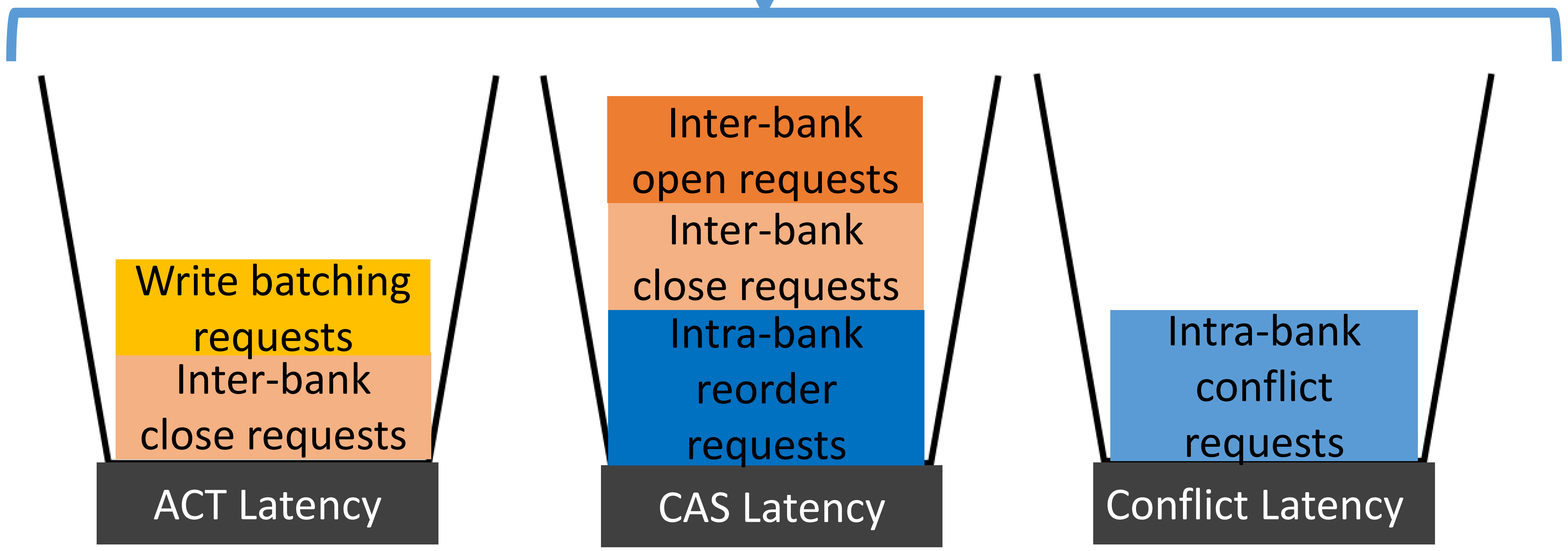
3. Inter-bank close requests
4. Inter-bank open requests

Write batching

5. Write batching requests

Optimization problem:

1. Write Latency components as functions on those requests
2. Define constraints on the number of requests based on request-driven and job driven analysis



Conflict Req

Proposed

Intra-bank interfering requests:

1. Intra-bank conflict requests
2. Intra-bank reorder requests

Inter-bank interfering requests:

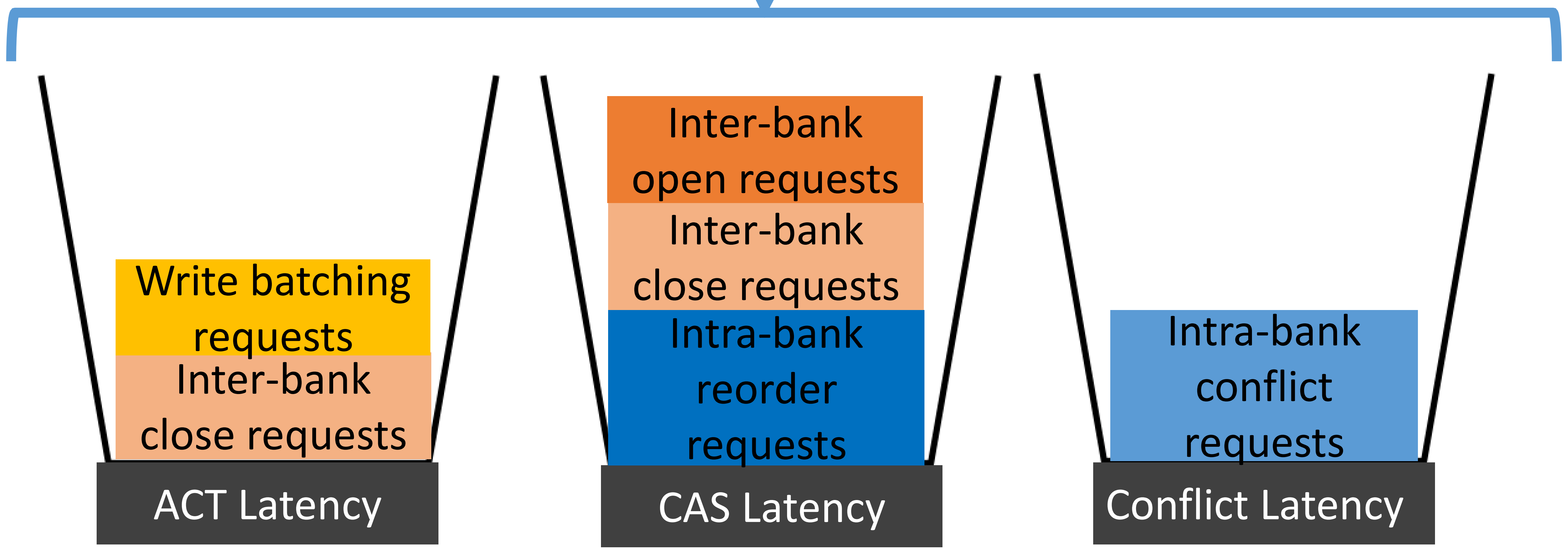
3. Inter-bank close requests
4. Inter-bank open requests

Write batching

5. Write batching requests

Optimization problem:

1. Write Latency components as functions on those requests
2. Define constraints on the number of requests based on request-driven and job driven analysis
3. Maximize total latency (summation of all components)



Conflict Req

Proposed

Intra-bank conflict Read (Write) requests from *p*	Inter-bank close Read (Write) requests from *p* interfering with close *rua*				≤	close Read (Write) requests from p
Intra-bank reorder Read (Write) requests from p	Inter-bank open Read (Write) requests from p interfering with close *rua*				≤	open Read (Write) requests from p
Intra-bank conflict Read (Write) requests from p	Inter-bank close Read (Write) requests from *p* interfering with close *rua*	Intra-bank reorder Read (Write) requests from p	Inter-bank open Read (Write) requests from p interfering with close *rua*	Inter-bank Read (Write) requests from *p* interfering with open *rua*	≤	Total Reads (writes) from *p*
Write batching requests					≤	close Write requests from p

Job-Driven Constraints

Proposed

			Interfering Reqs from Cr Core	Interfering Reqs from nCr Core
Part-All	Priority	FR-FCFS thr	None	None
		No thr		
	No-Priority	FR-FCFS thr		
		No thr		
Part-Cr	Priority	FR-FCFS thr		
		No thr		
	No-Priority	FR-FCFS thr		*
		No thr		unbounded
No-Part	Priority	FR-FCFS thr	*	None
		No thr	Unbounded	
	No-Priority	FR-FCFS thr	*	*
		No thr	Unbounded	Unbounded

- *** Constraint:**
  - If FR-FCFS is with threshold: no more than N^{thr} can cause reorder-interference with request under analysis →

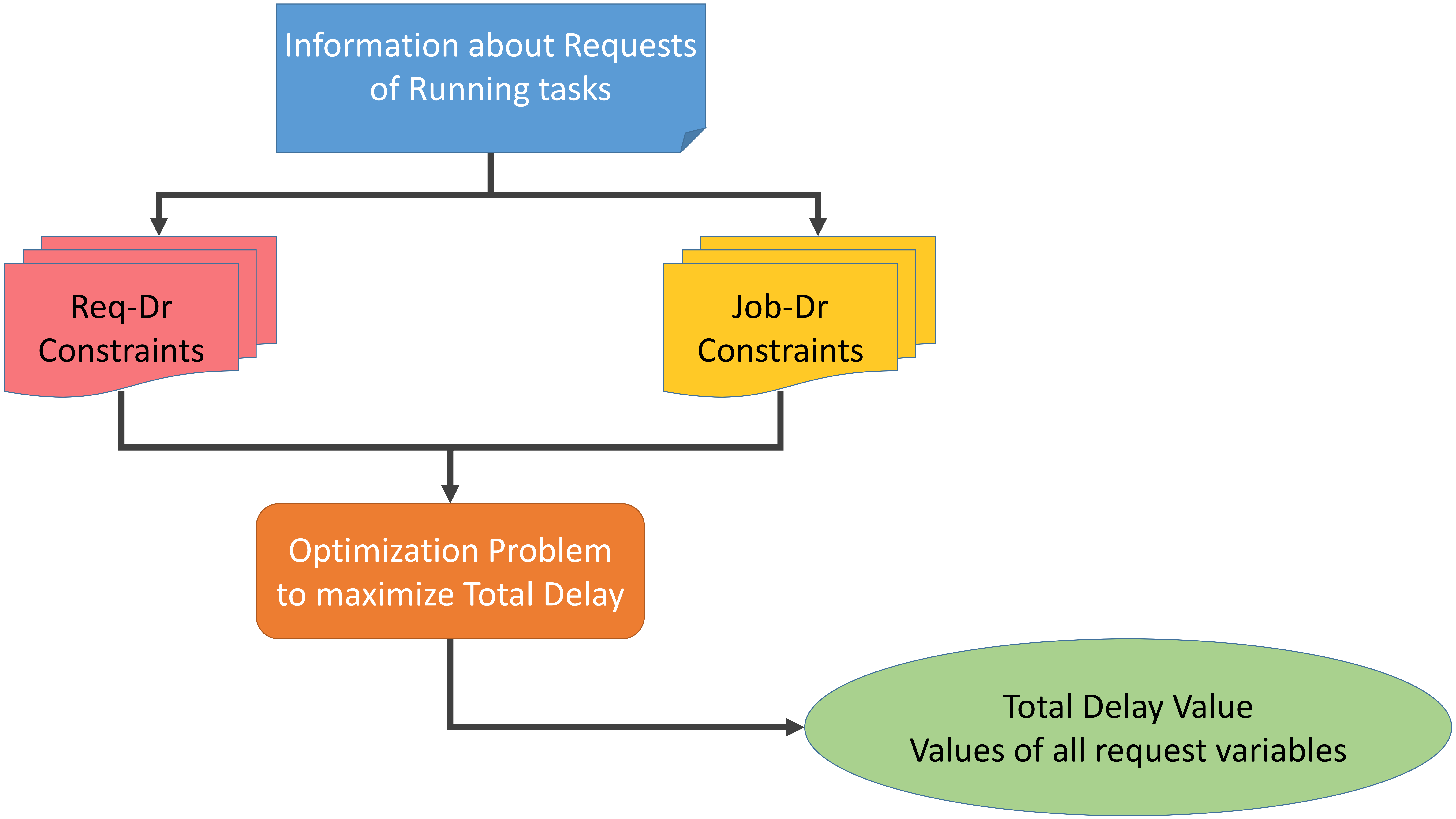
Total # reorder interfering requests from all cores ≤ N^{thr} ×# Interfered

Request-Dr Constraints

Example: Reorder Requests

Intra-bank reorder Read (Write) requests from p

Proposed



Overall Approach

Proposed

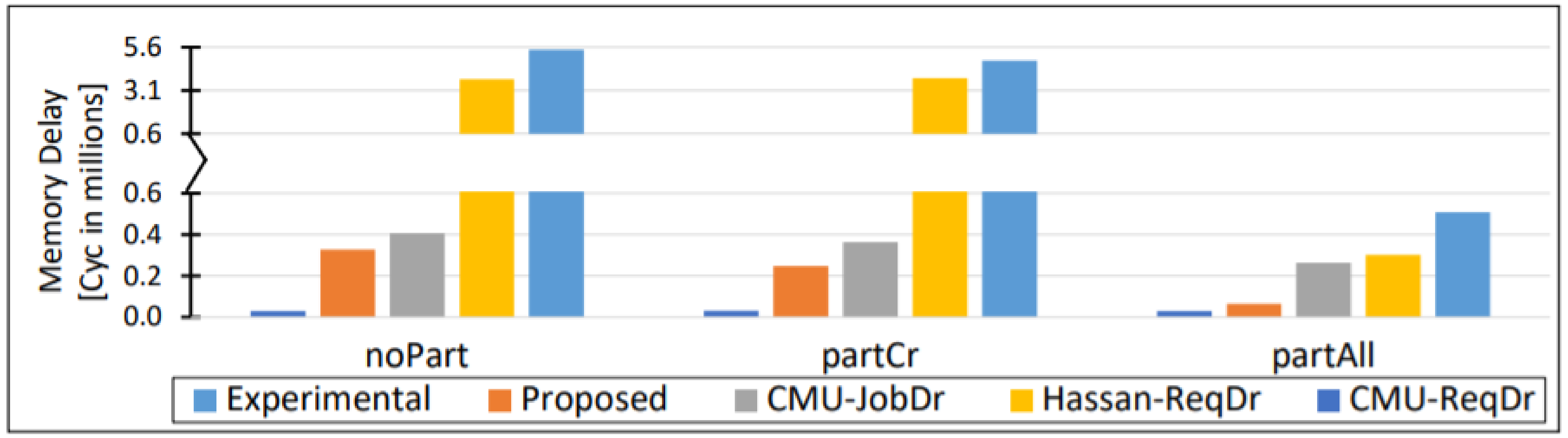
P	4	Pcr	2	Pncr	2
Nthr	8	Wbtch	16	PR	4
NB	8				
NBp	• noPart		8		
	• PartAll		2		
	• PartCr and p is Cr		4		
	• PartCr and p is nCr		8		
NBcr	• noPart of PartCr		8		
	• PartAll		4		
NBncr	• noPart of PartCr		8		
	• PartAll		4		

System Configuration

High				Low			
BM	#Reads	#writes	Total	BM	#Reads	#writes	Total
matrix	280000	38428	318428	rspeed	2000	482	2482
a2time	166000	21751	187751	pntrch	2000	479	2479
aifftr	101000	77234	178234	basefp	2000	478	2478

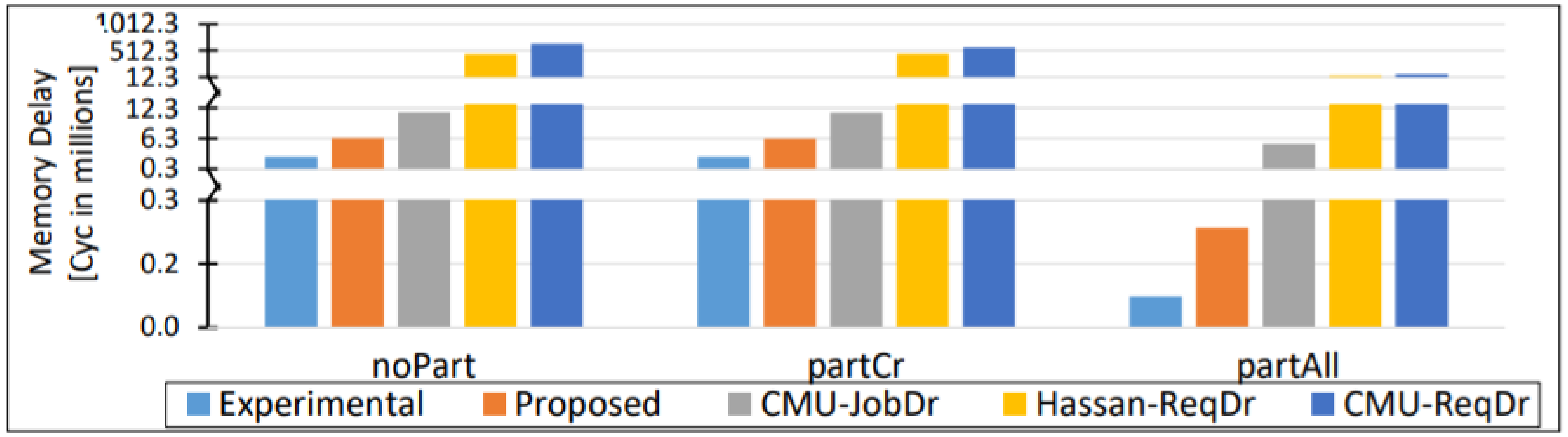
Benchmarks

RESULTS



(a) Low-Low.

(b) Low-High.

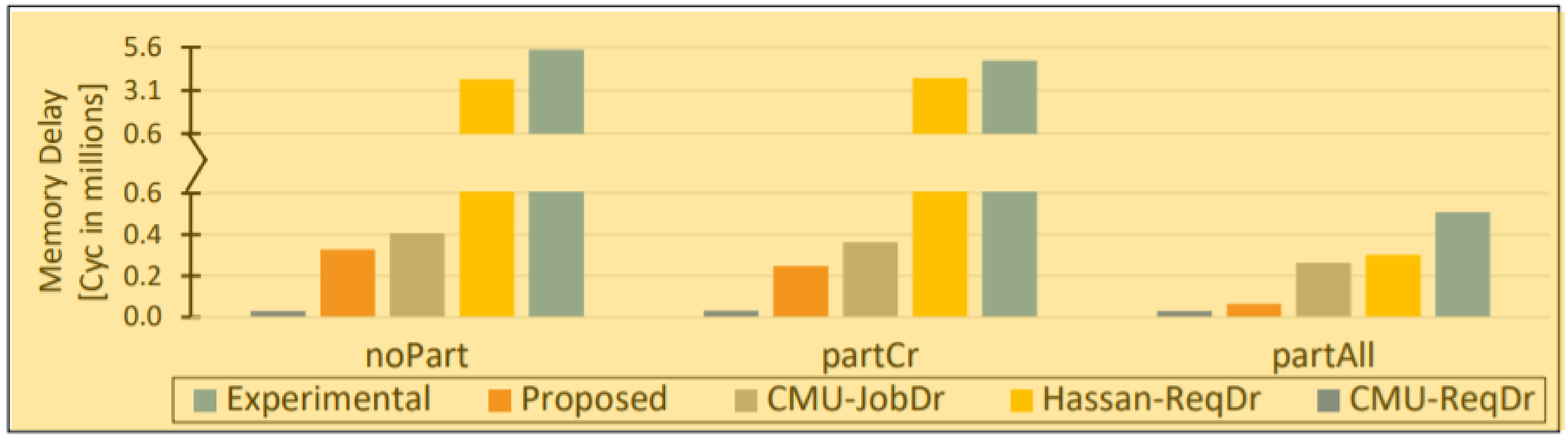


(c) High-Low.

(d) High-High.

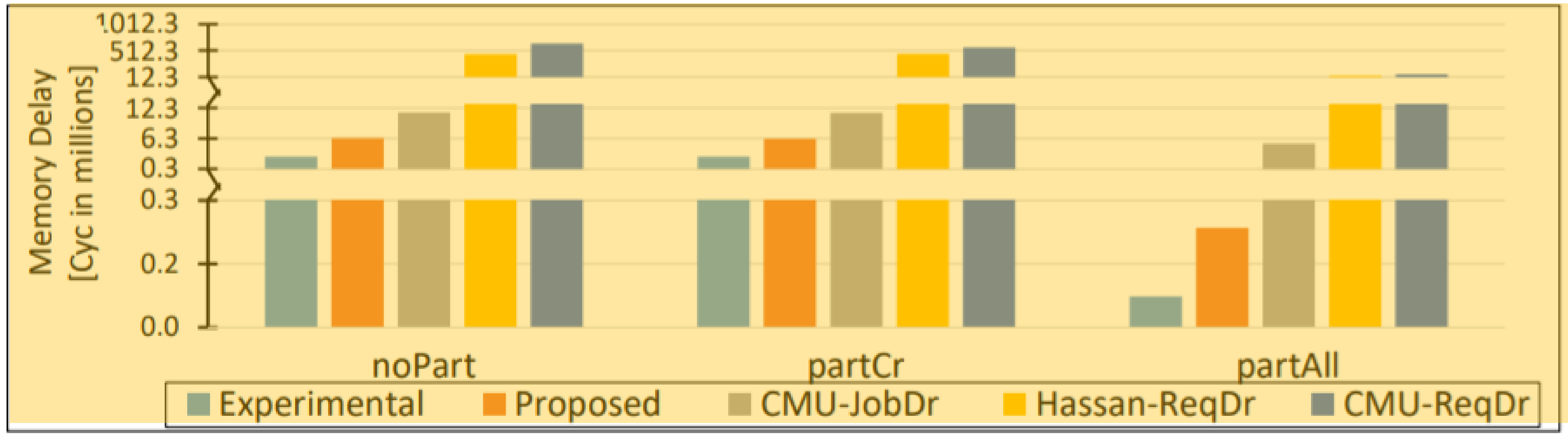
Comparison with [CMU] across its supported platforms

RESULTS



(a) Low-Low.

(b) Low-High.



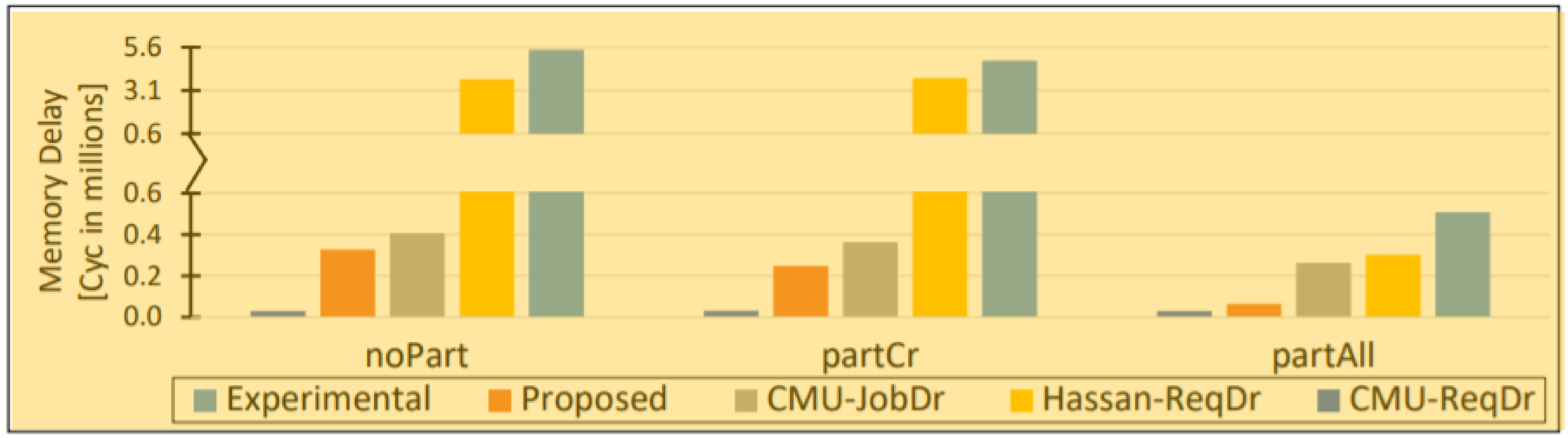
(c) High-Low.

(d) High-High.

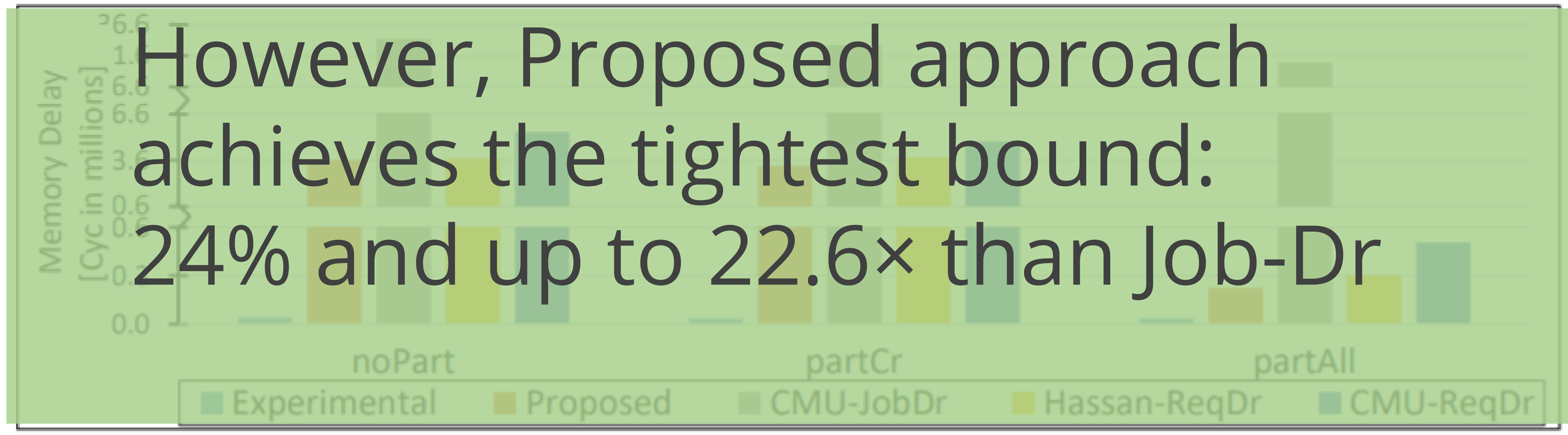
CMU-JobDr is achieving better performance than Req-Dr in these 3

Comparison with [CMU] across its supported platforms

RESULTS

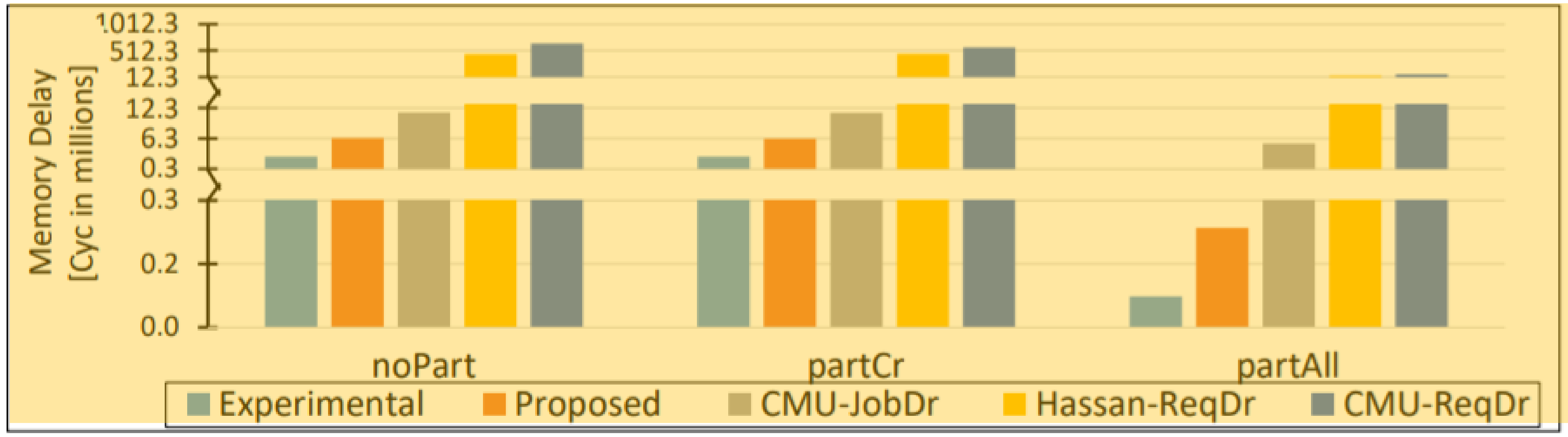


(a) Low-Low.



However, Proposed approach achieves the tightest bound: 24% and up to 22.6× than Job-Dr

(b) Low-High.



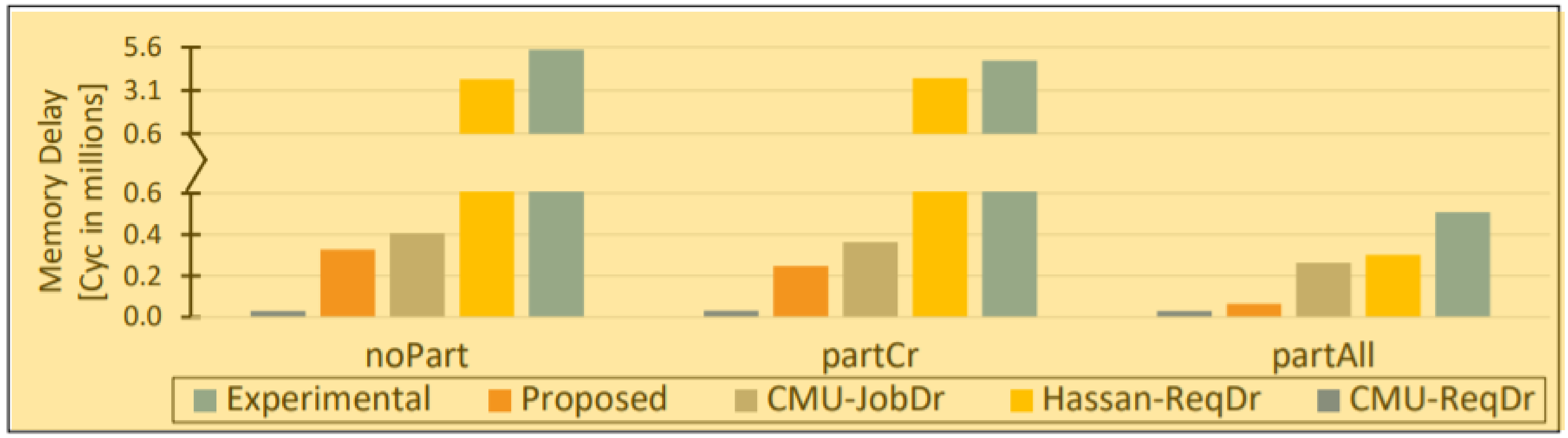
(c) High-Low.

(d) High-High.

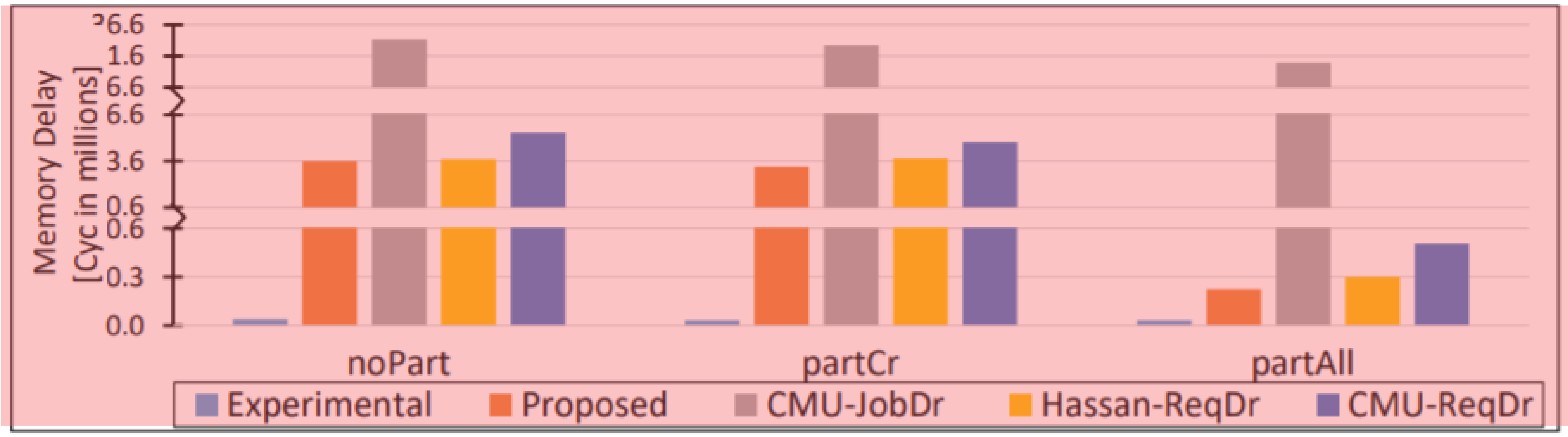
CMU-JobDr is achieving better performance than Req-Dr in these 3

Comparison with [CMU] across its supported platforms

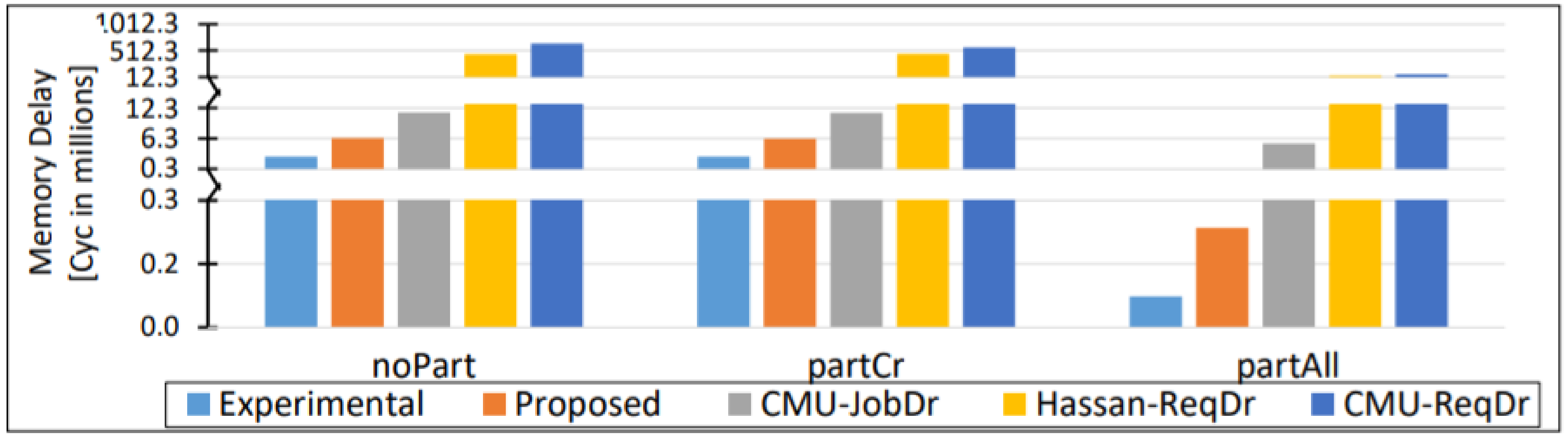
RESULTS



(a) Low-Low.



(b) Low-High.



(c) High-Low.

(d) High-High.

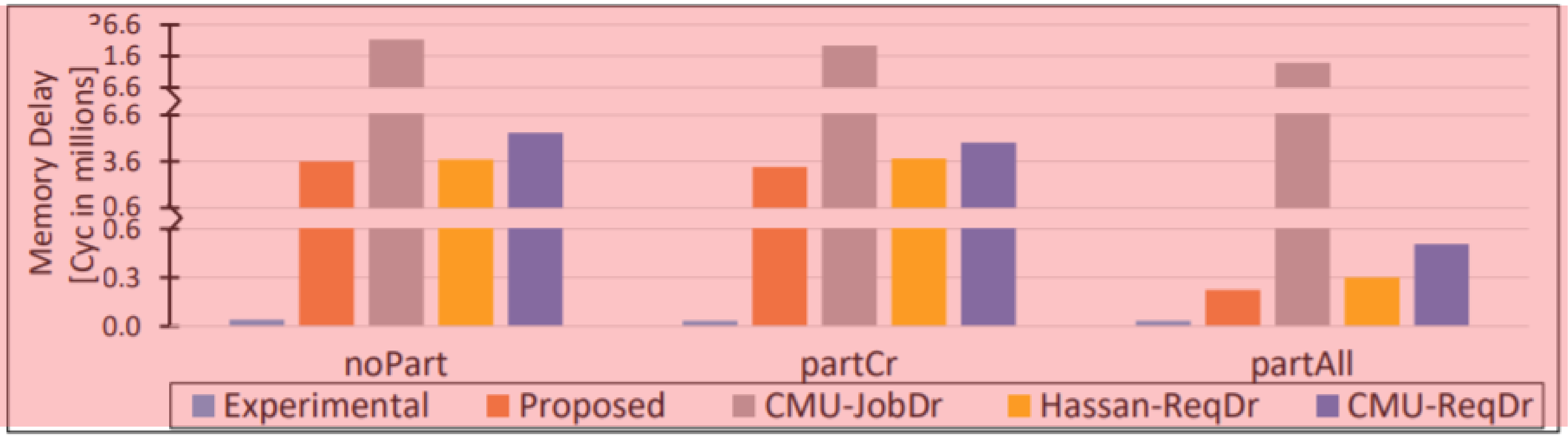
Req-Dr achieves tighter bound than Job-Dr in this

Comparison with [CMU] across its supported platforms

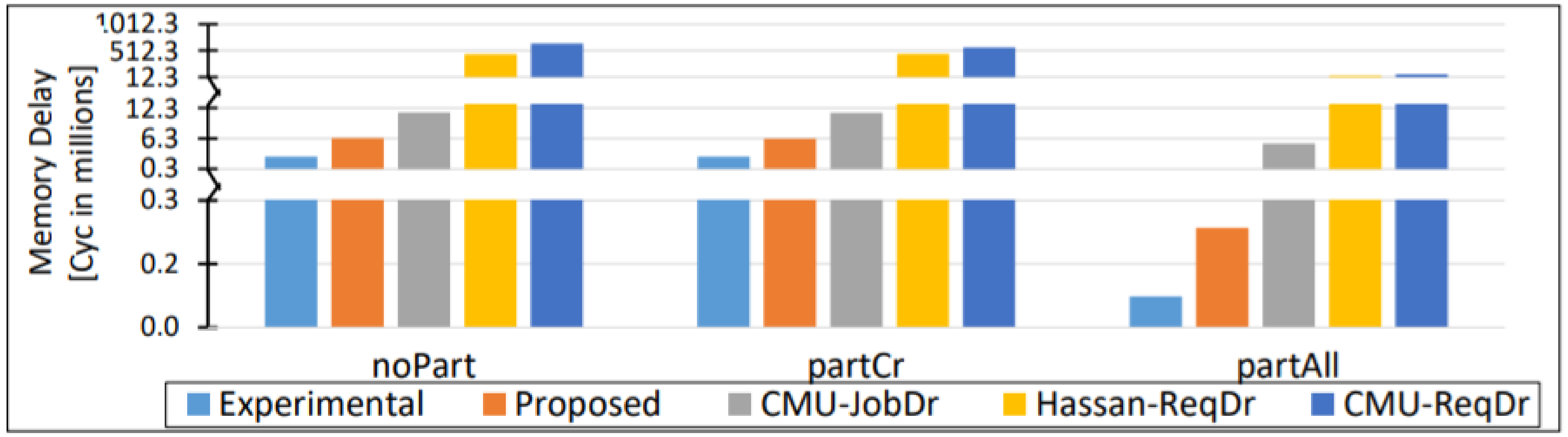
RESULTS

However, Proposed approach achieves the tightest bound (34% tighter than Hassan-ReqDr)

(a) Low-Low.



(b) Low-High.



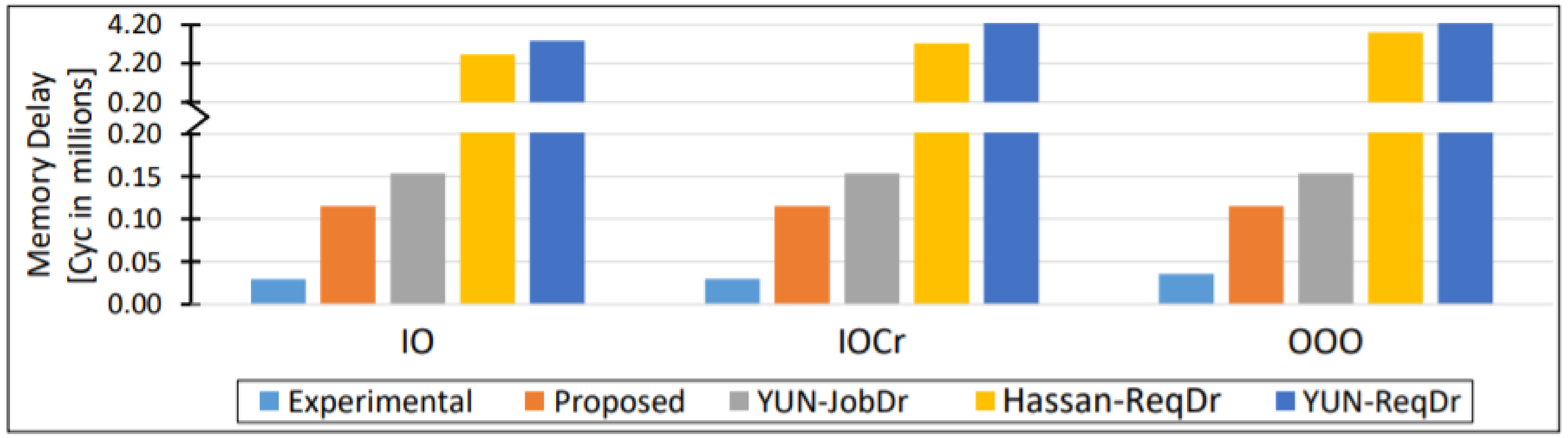
(c) High-Low.

(d) High-High.

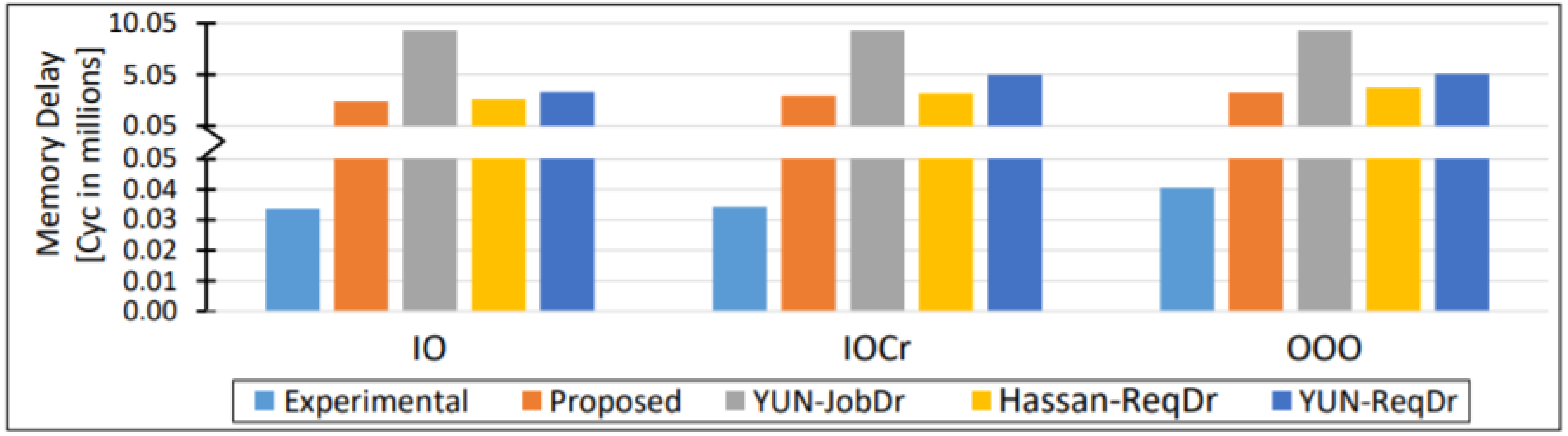
Req-Dr achieves tighter bound than Job-Dr in this

Comparison with [CMU] across its supported platforms

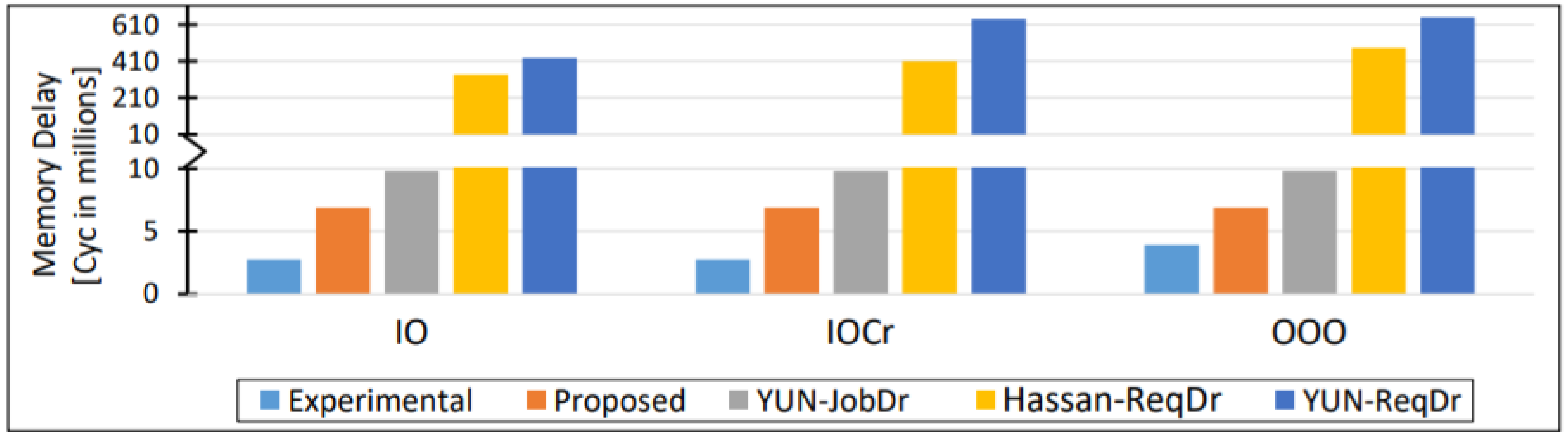
RESULTS



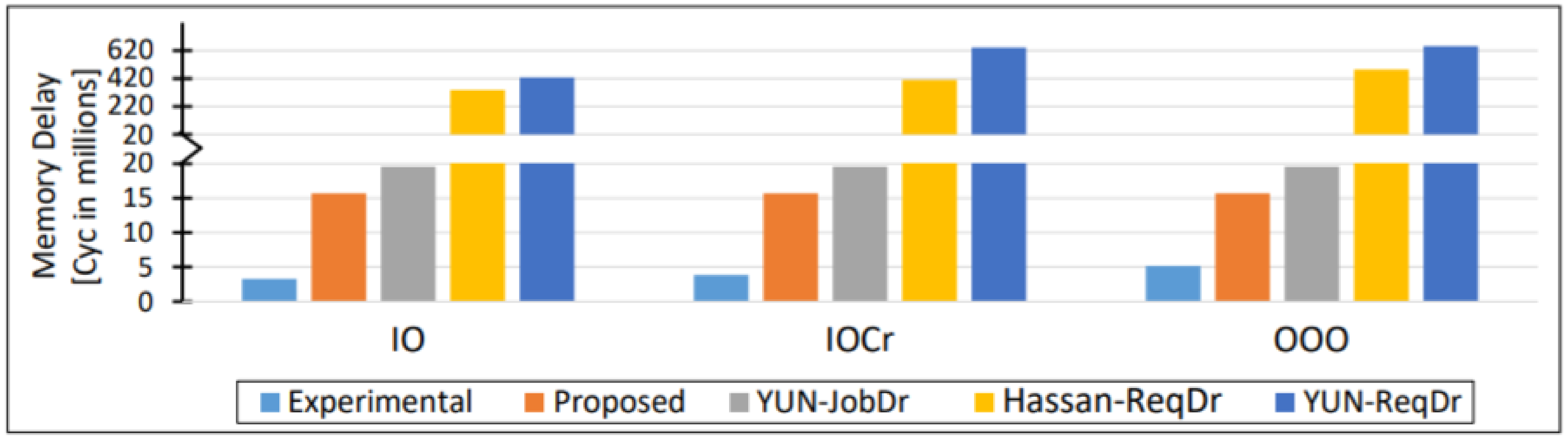
(a) Low-Low.



(b) Low-High.



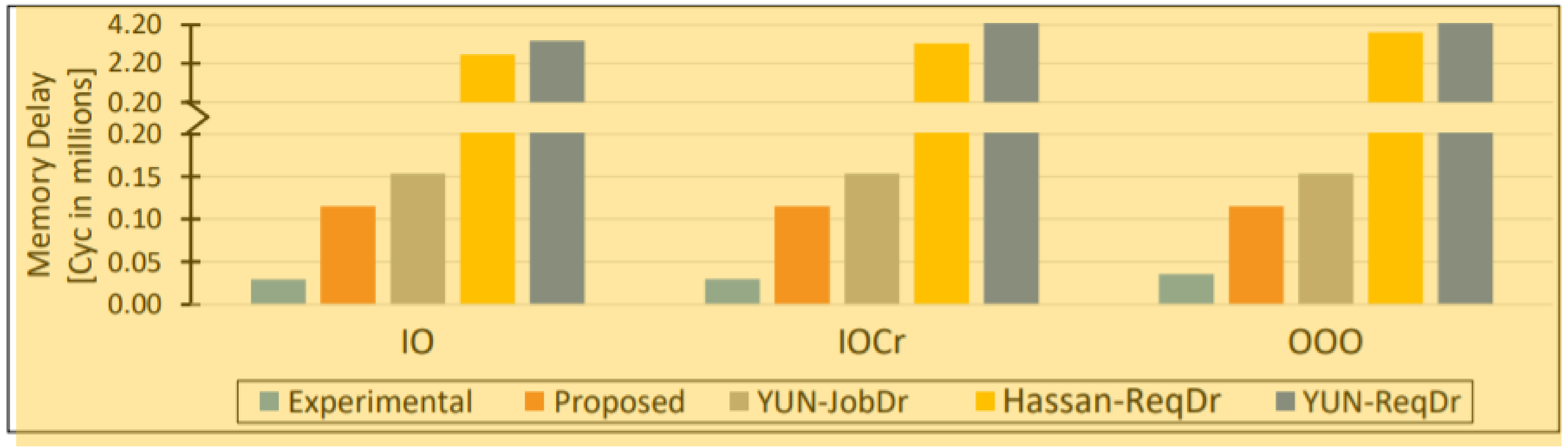
(c) High-Low.



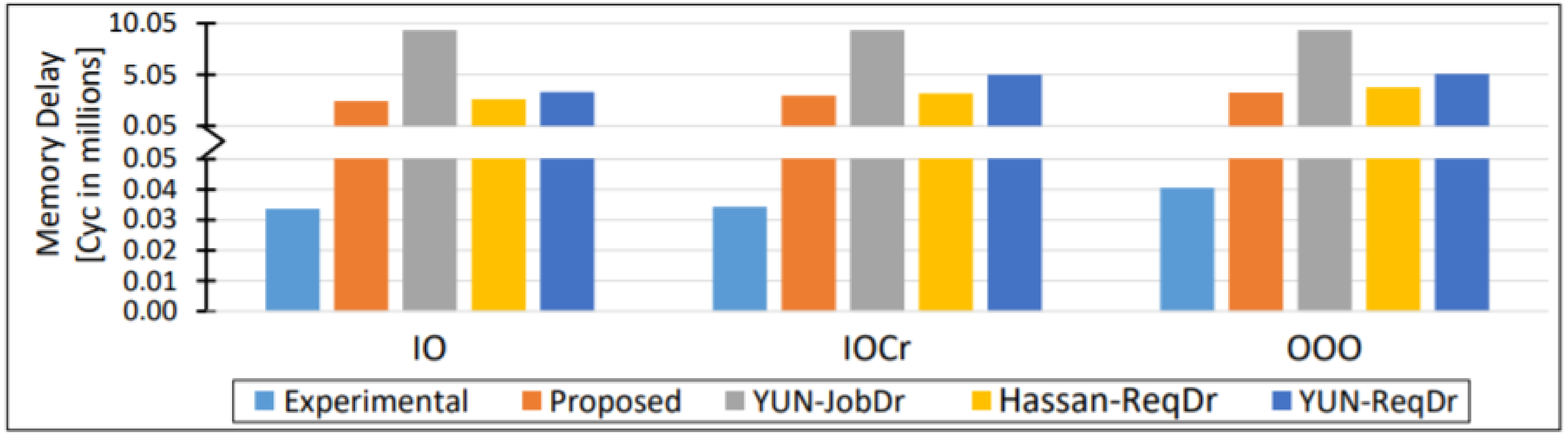
(d) High-High.

Comparison with [YUN] across its supported platforms

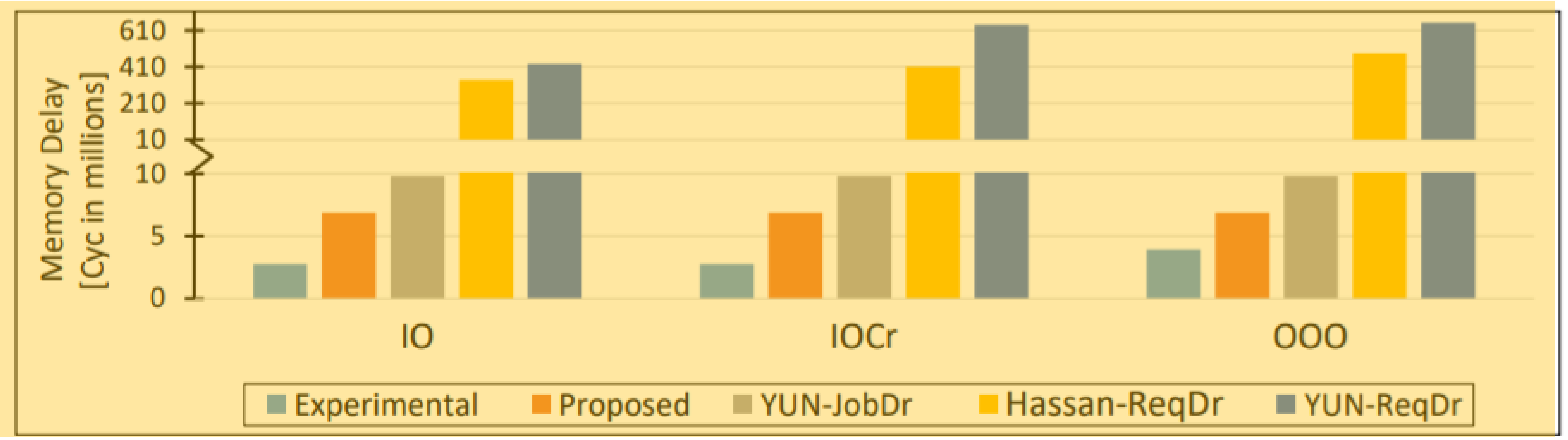
RESULTS



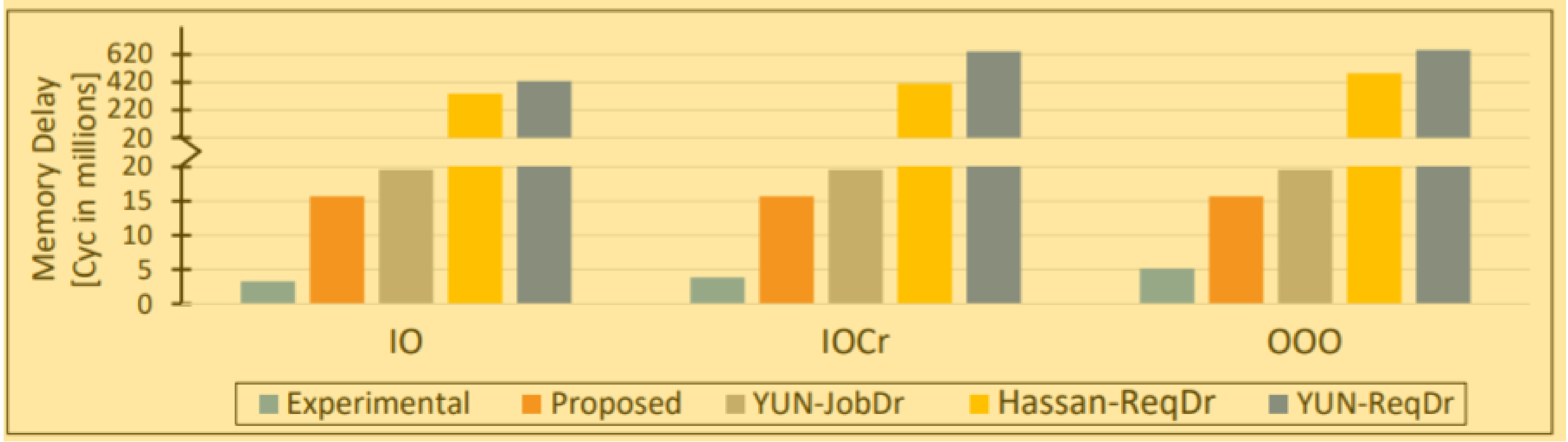
(a) Low-Low.



(b) Low-High.



(c) High-Low.

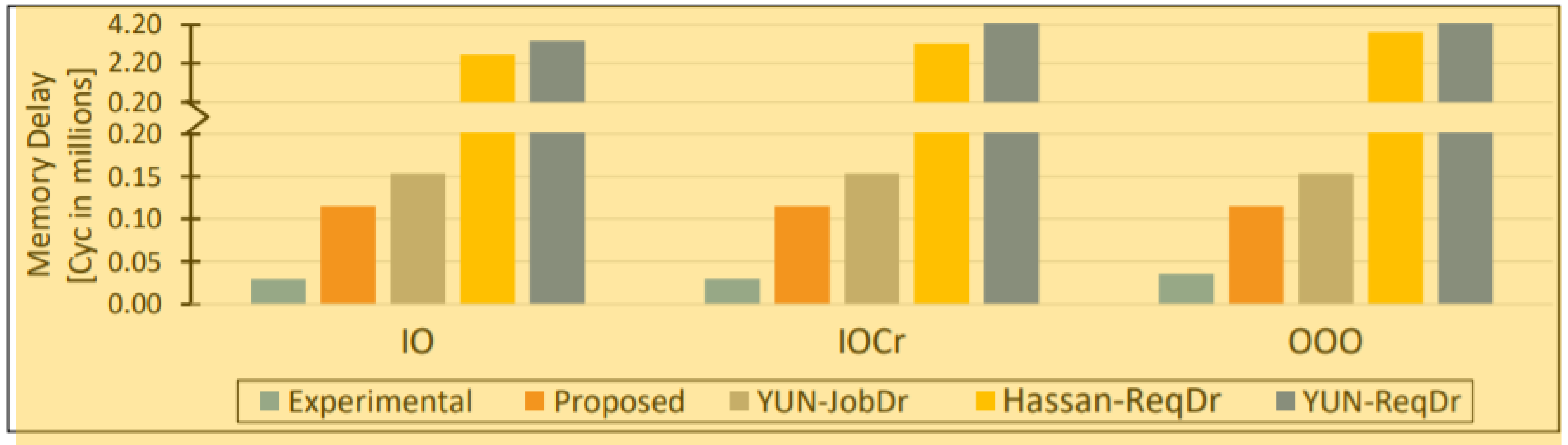


(d) High-High.

YUN-JobDr is achieving better performance than Req-Dr in these 3

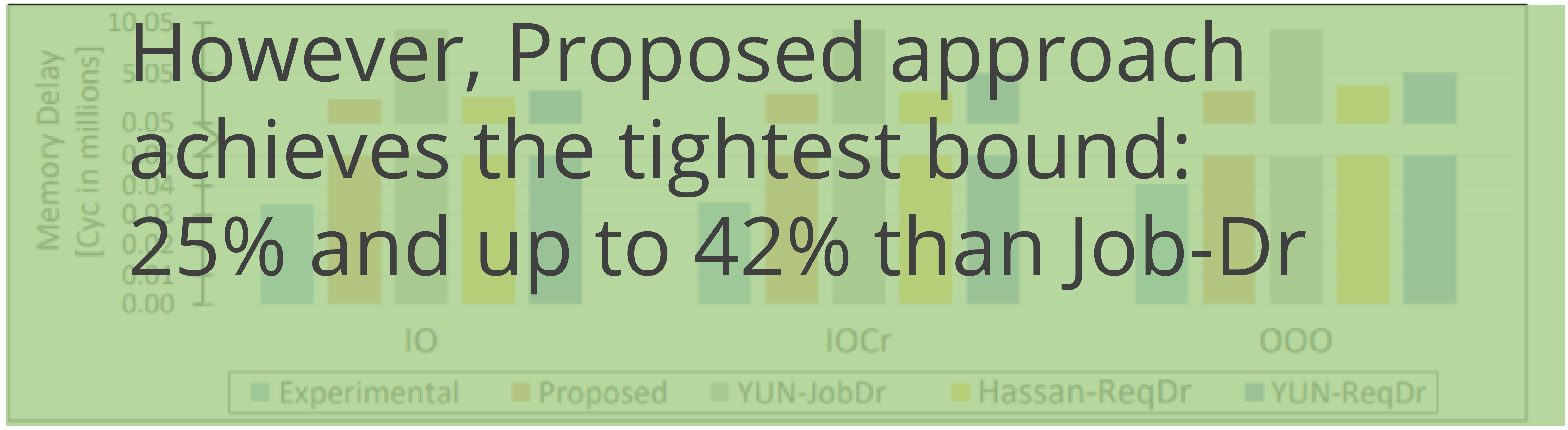
Comparison with [YUN] across its supported platforms

RESULTS

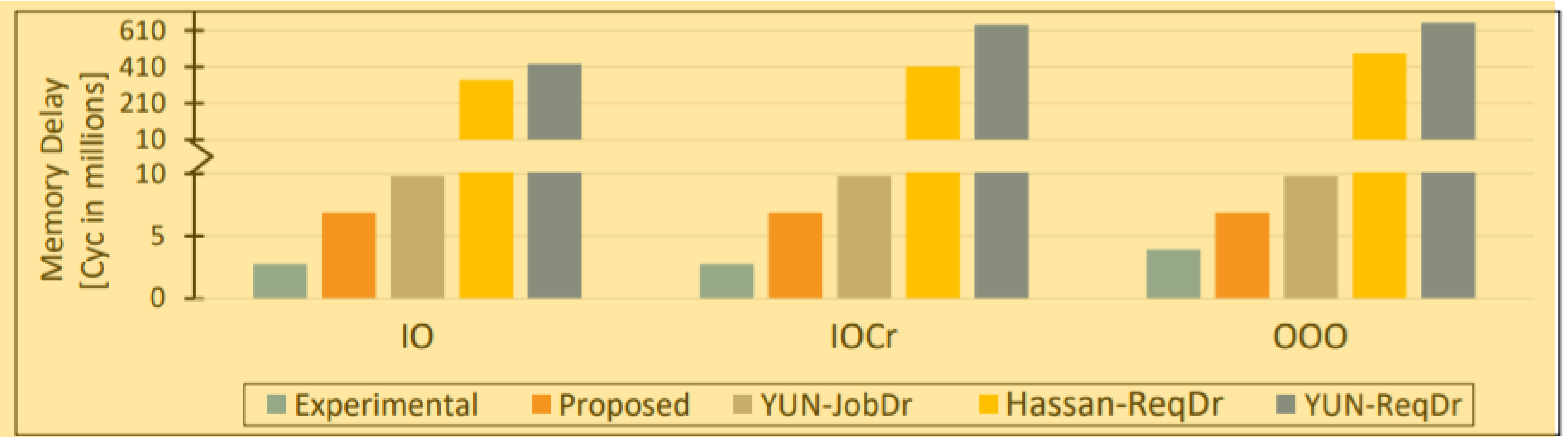


(a) Low-Low.

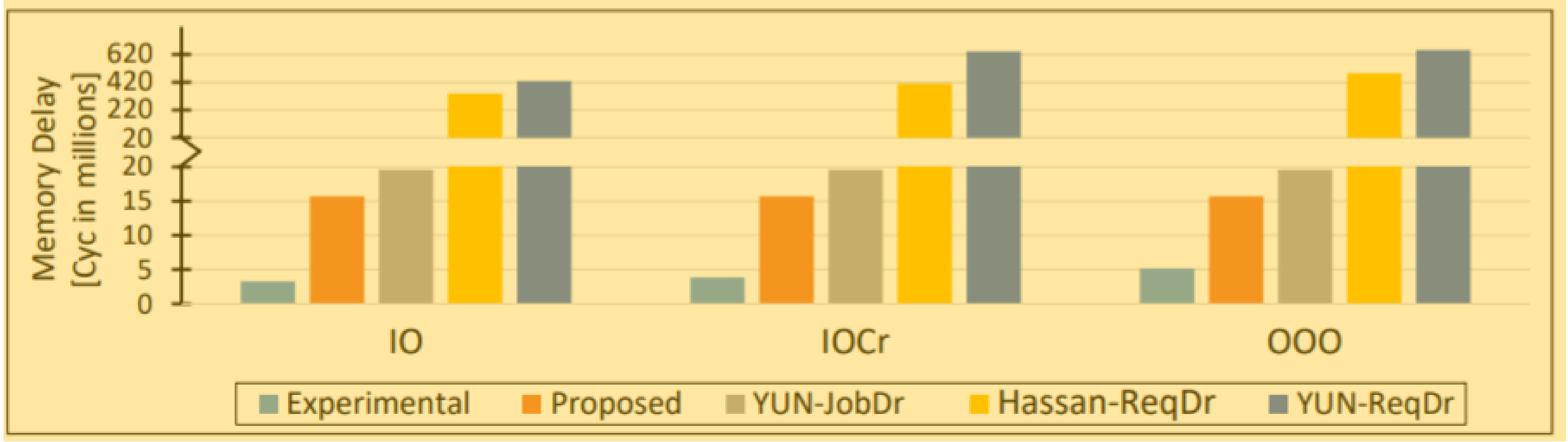
However, Proposed approach achieves the tightest bound: 25% and up to 42% than Job-Dr



(b) Low-High.



(c) High-Low.

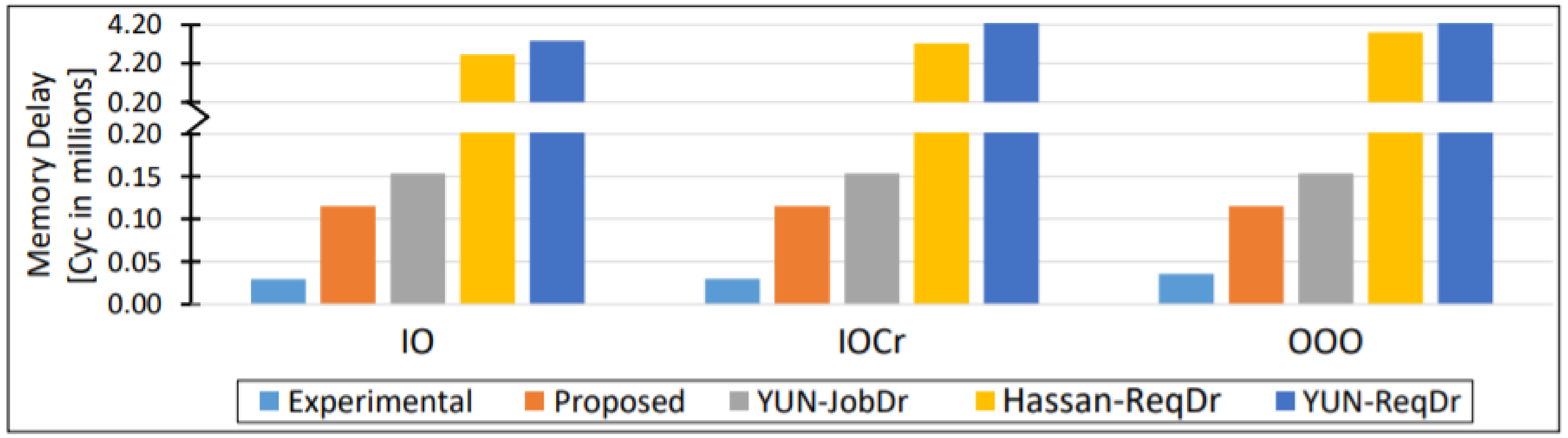


(d) High-High.

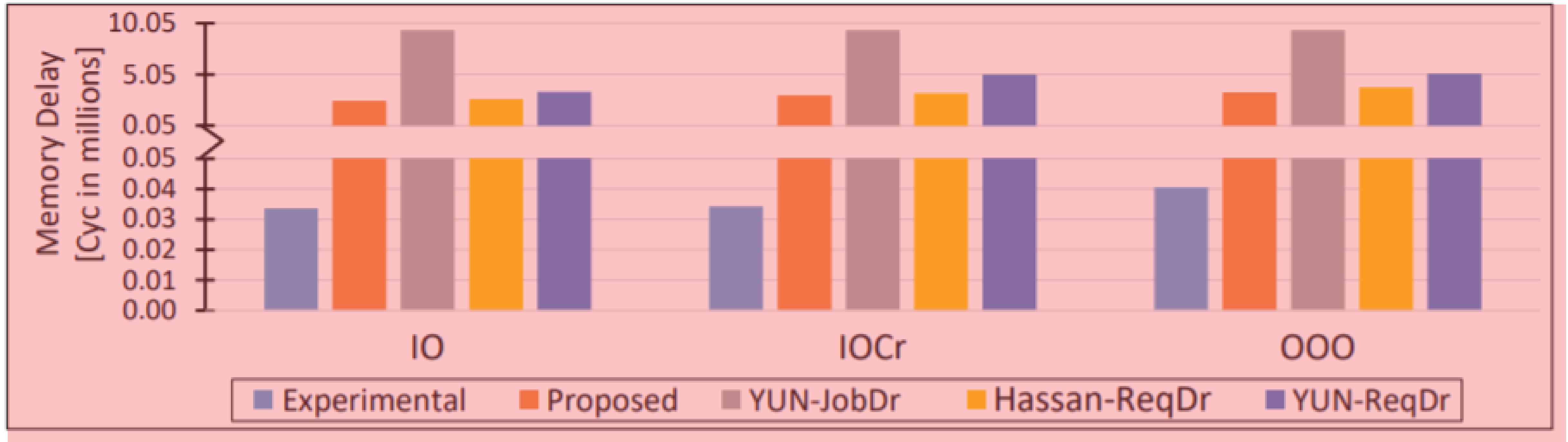
YUN-JobDr is achieving better performance than Req-Dr in these 3

Comparison with [YUN] across its supported platforms

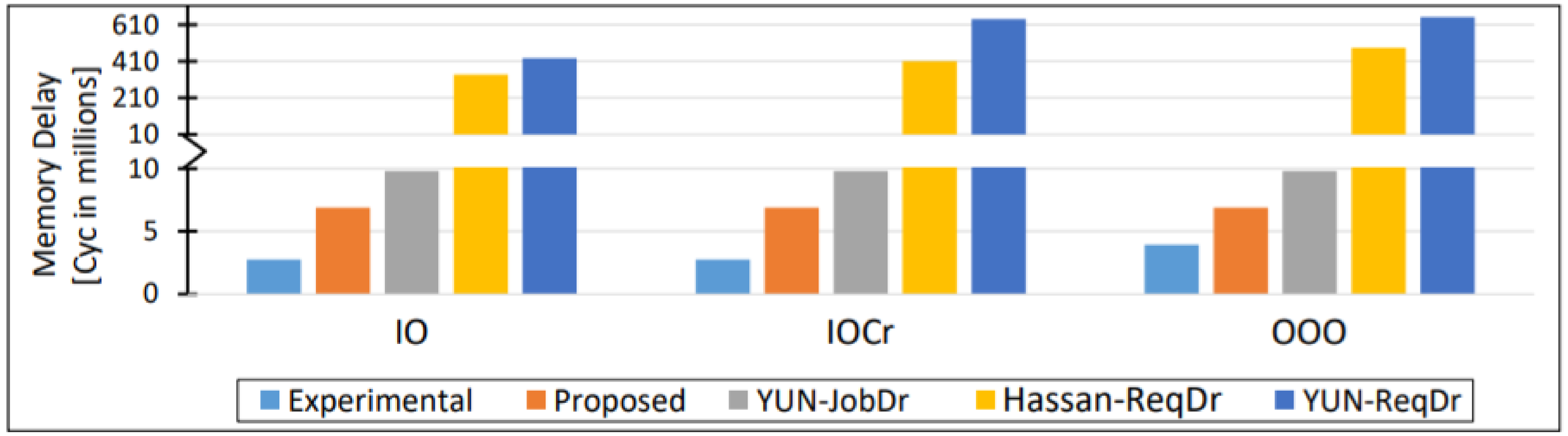
RESULTS



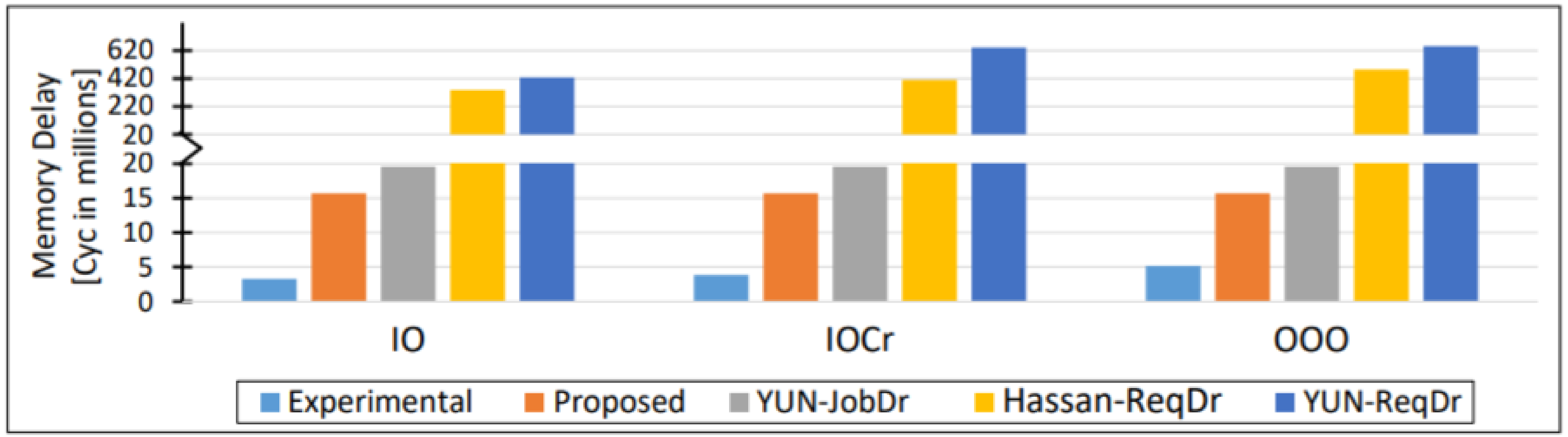
(a) Low-Low.



(b) Low-High.



(c) High-Low.



(d) High-High.

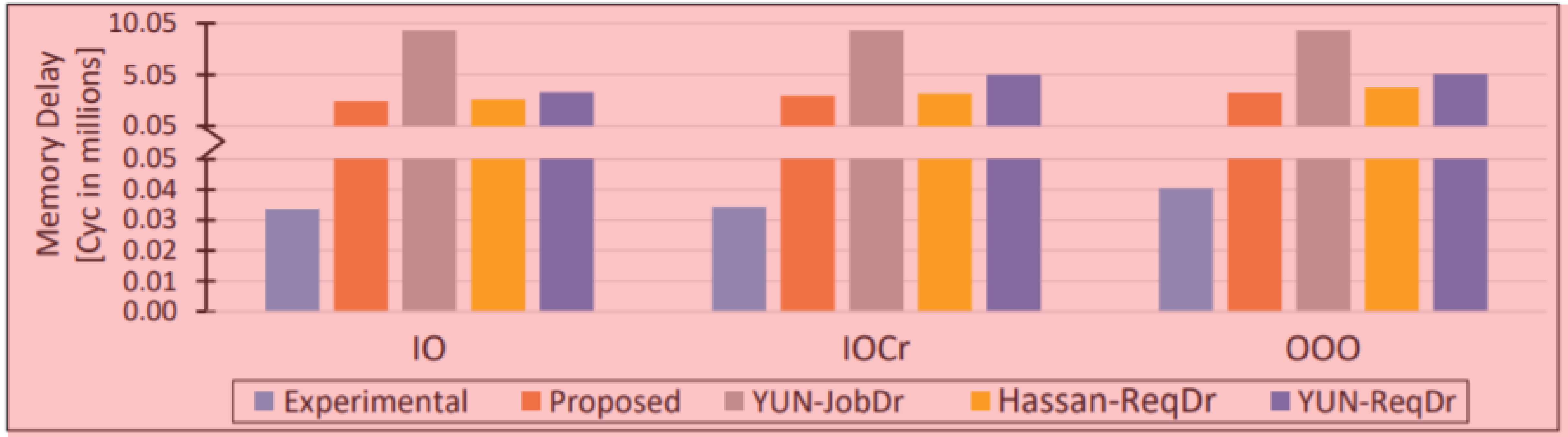
Req-Dr achieves tighter bound than Job-Dr in this

Comparison with [YUN] across its supported platforms

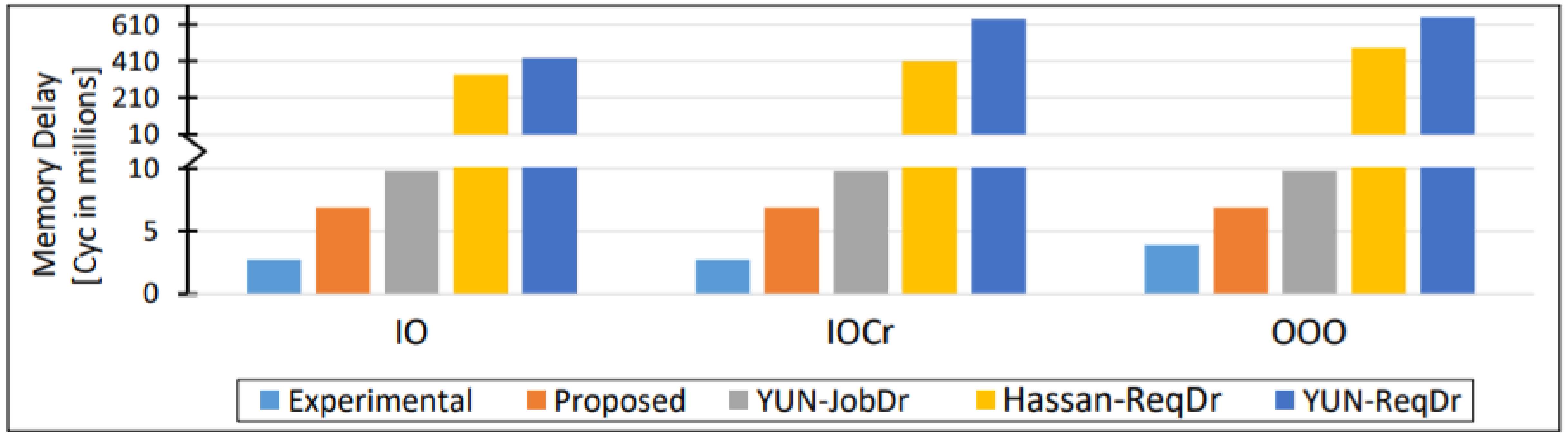
RESULTS

However, Proposed approach achieves the tightest bound (15% tighter than Hassan-ReqDr)

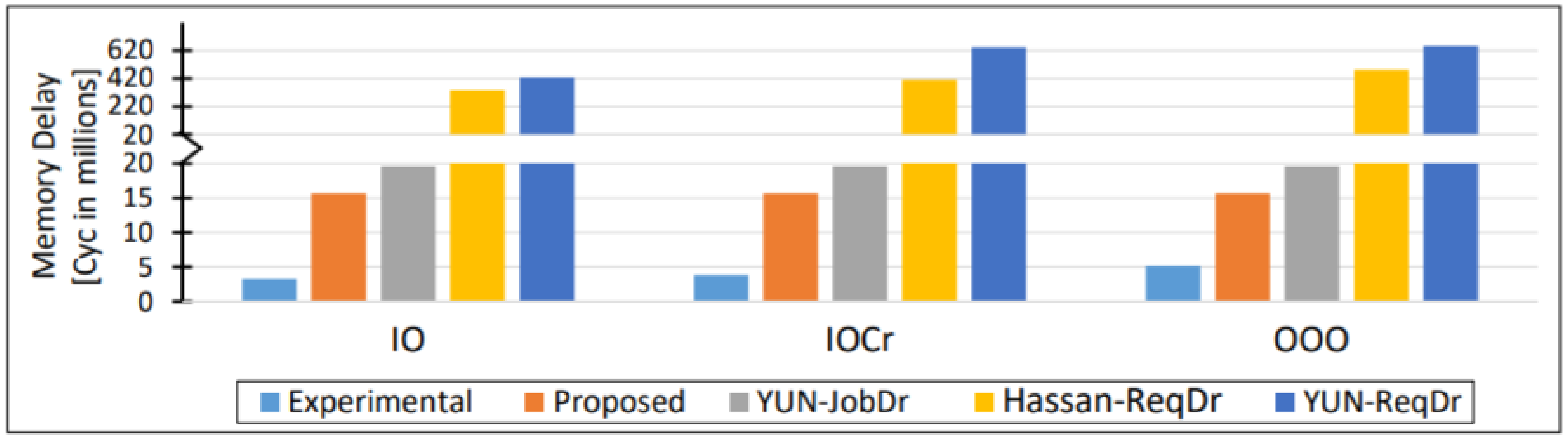
(a) Low-Low.



(b) Low-High.



(c) High-Low.

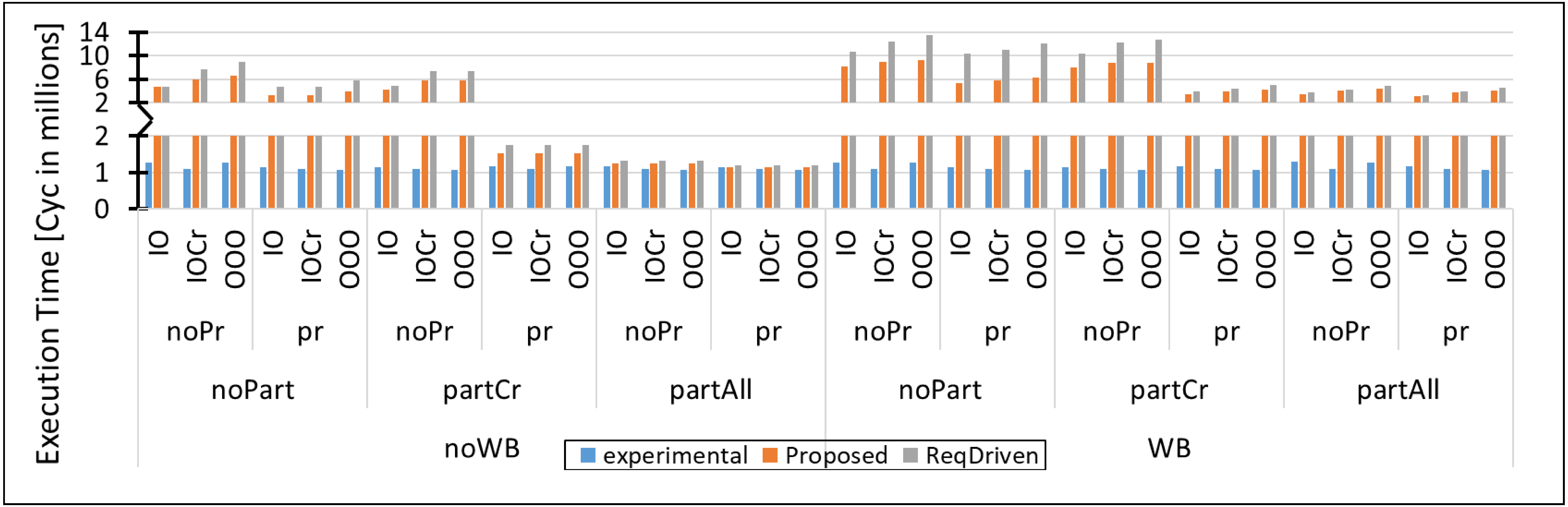


(d) High-High.

Req-Dr achieves tighter bound than Job-Dr in this

Comparison with [YUN] across its supported platforms

RESULTS

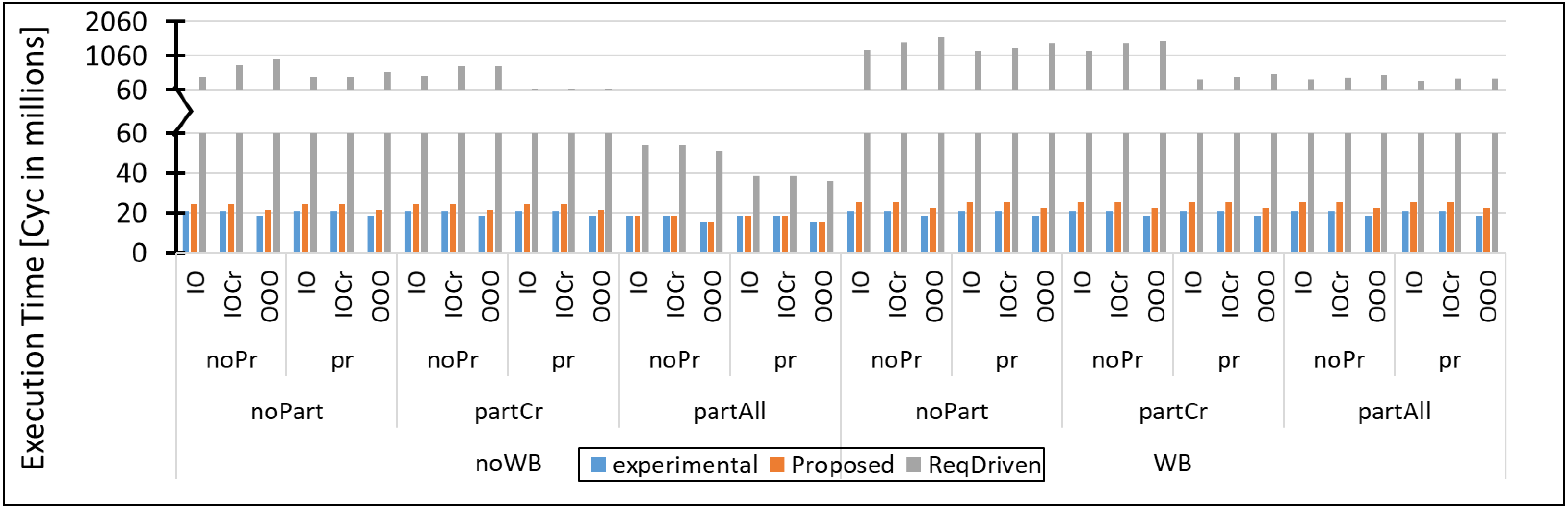


Proposed provides up to 98% and 24% on average tighter bounds across all platform instances

Comparison with Req-Dr across platforms

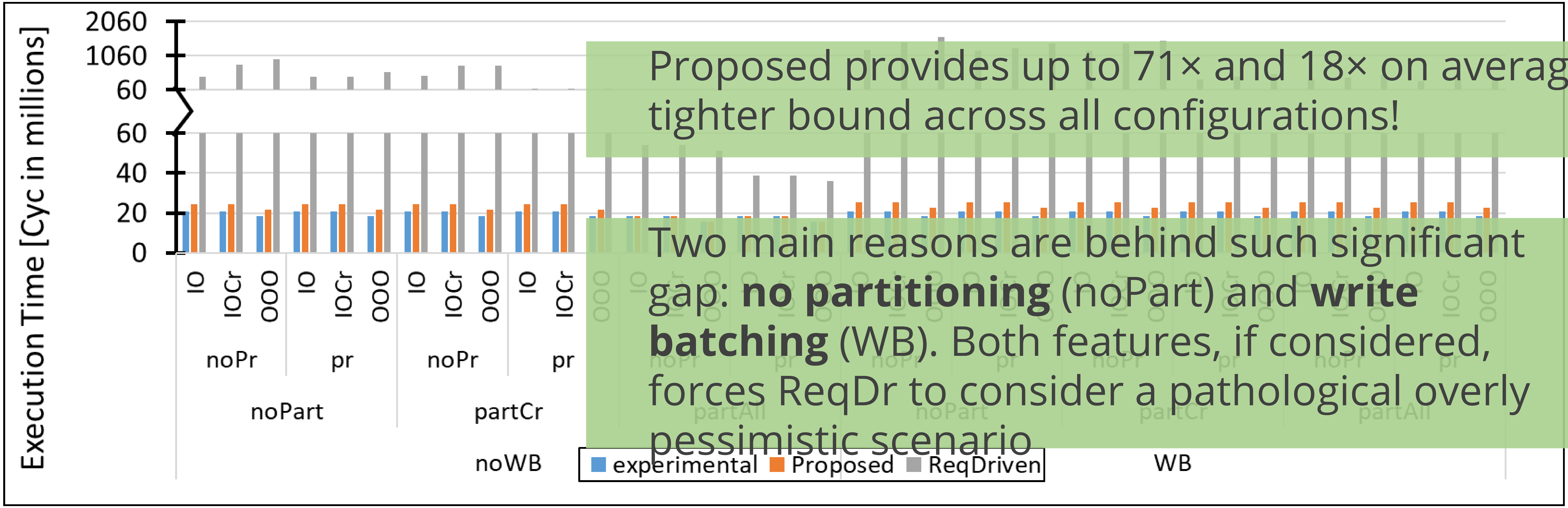
Low-High case

RESULTS



Comparison with Req-Dr across platforms
High-Low case

RESULTS



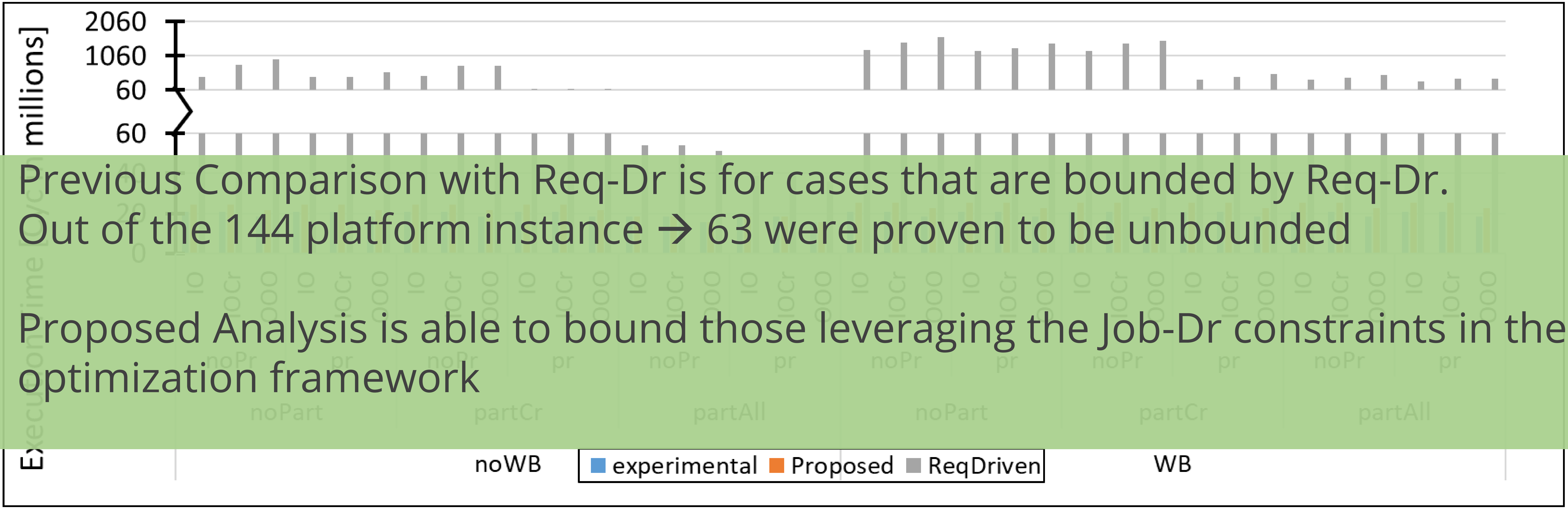
Proposed provides up to 71× and 18× on average tighter bound across all configurations!

Two main reasons are behind such significant gap: **no partitioning** (noPart) and **write batching** (WB). Both features, if considered, forces ReqDr to consider a pathological overly pessimistic scenario

Comparison with Req-Dr across platforms

High-Low case

RESULTS

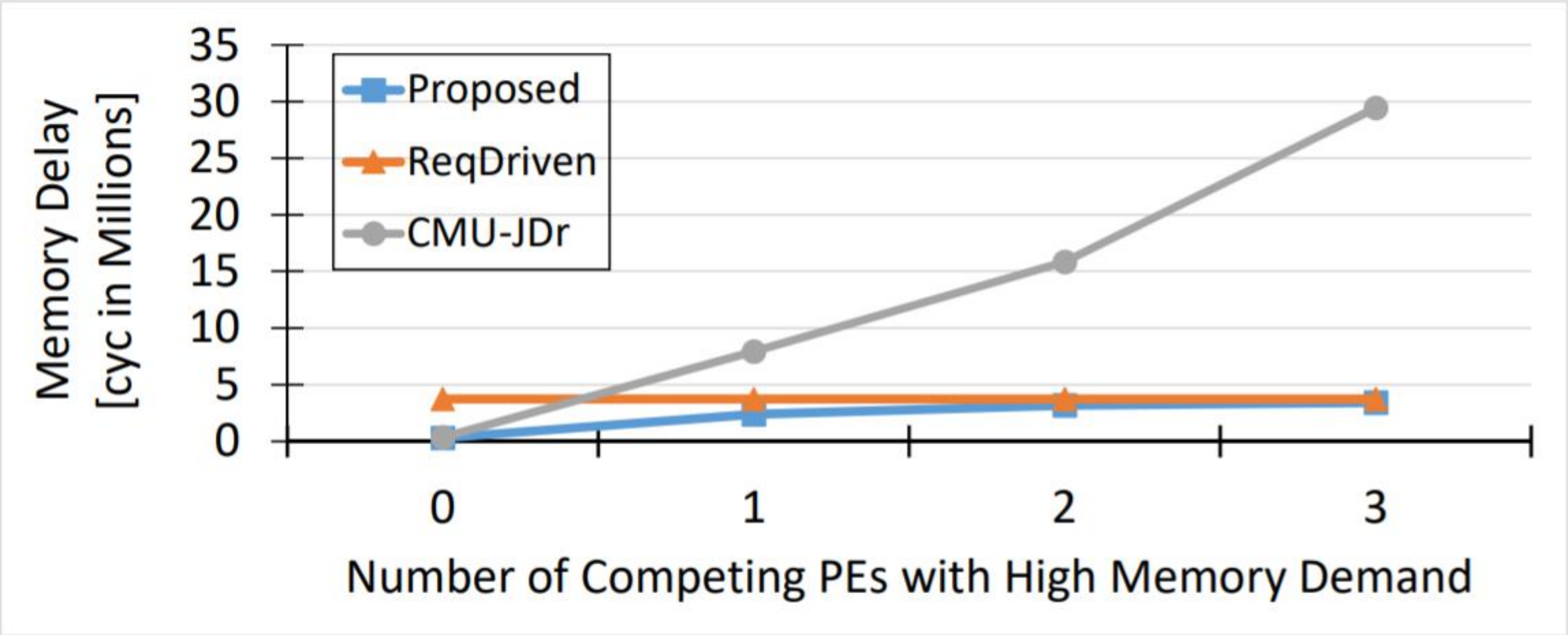


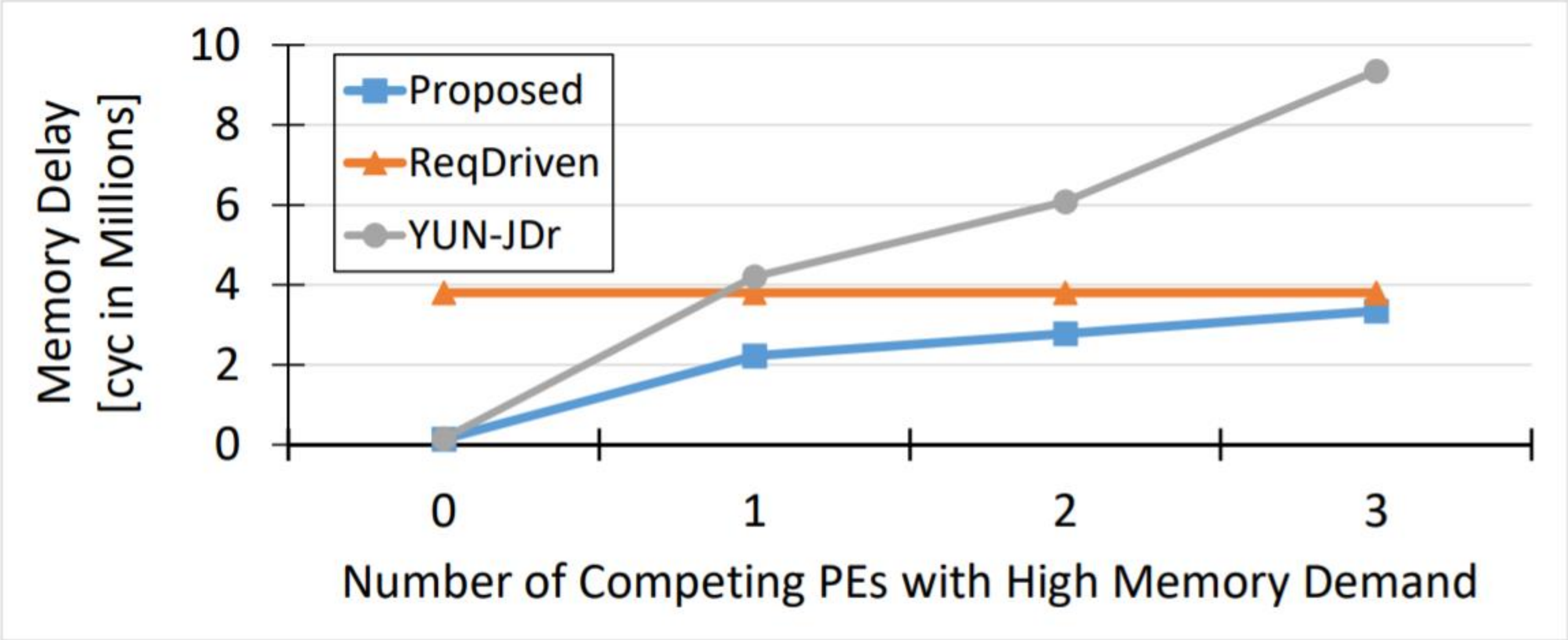
Previous Comparison with Req-Dr is for cases that are bounded by Req-Dr.
Out of the 144 platform instance → 63 were proven to be unbounded

Proposed Analysis is able to bound those leveraging the Job-Dr constraints in the optimization framework

Comparison with Req-Dr across platforms
Unbounded cases by Req-Dr

RESULTS

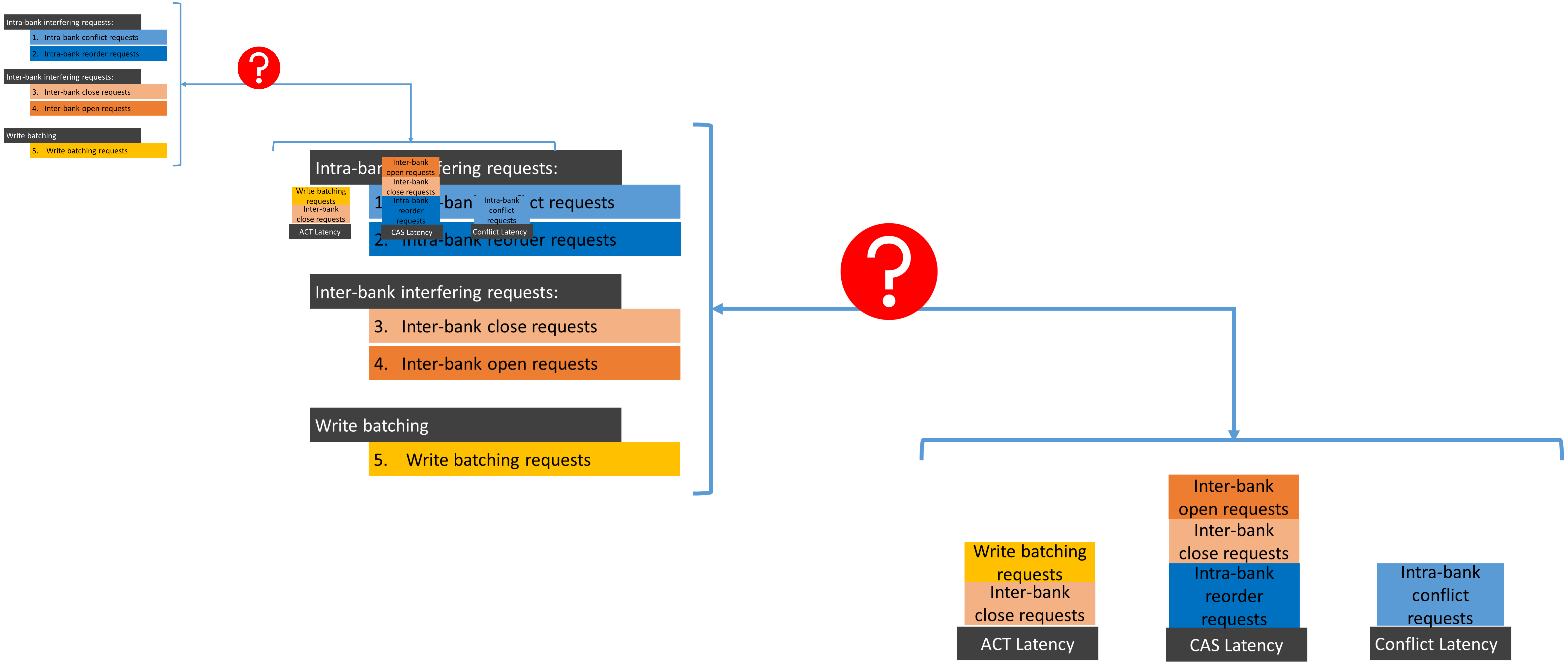




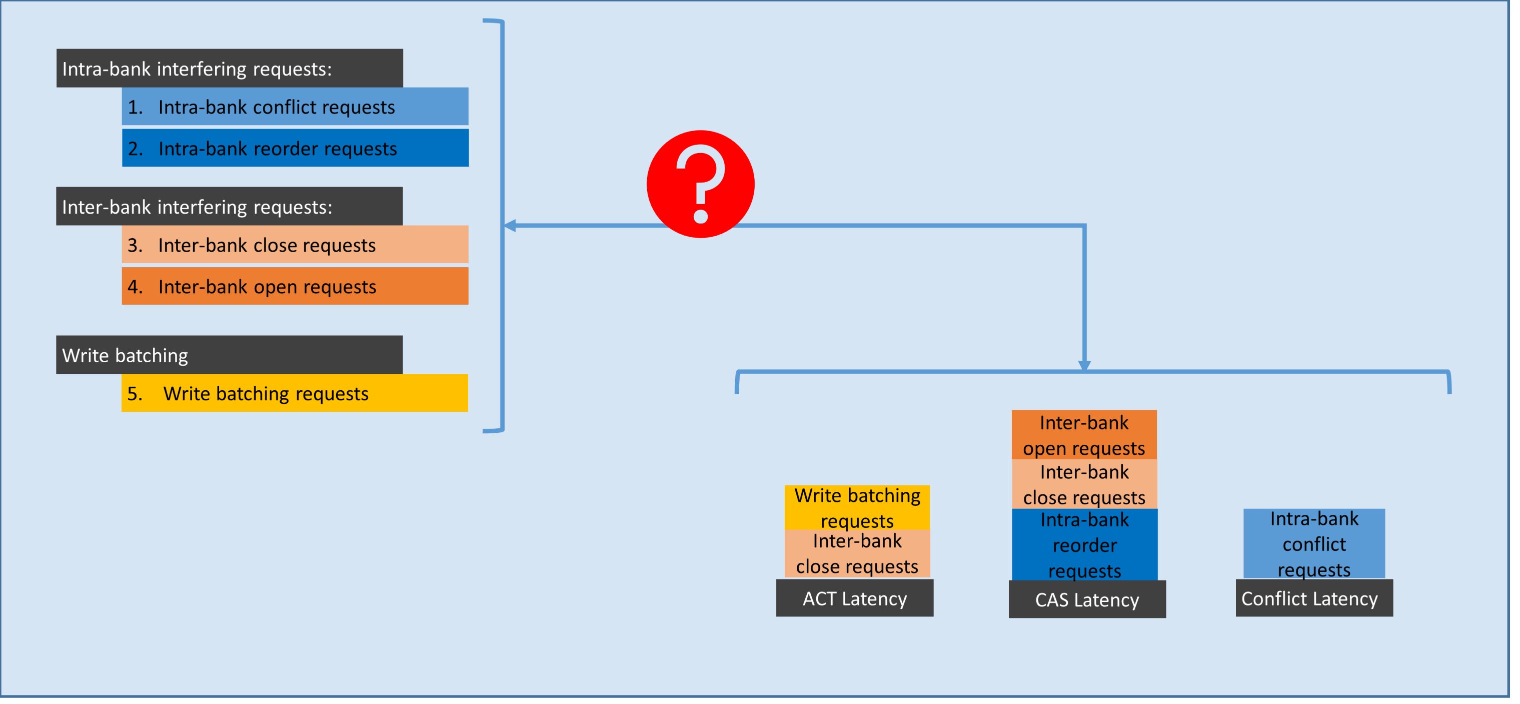
Back to these two figures

How does the proposed hybrid analysis perform?

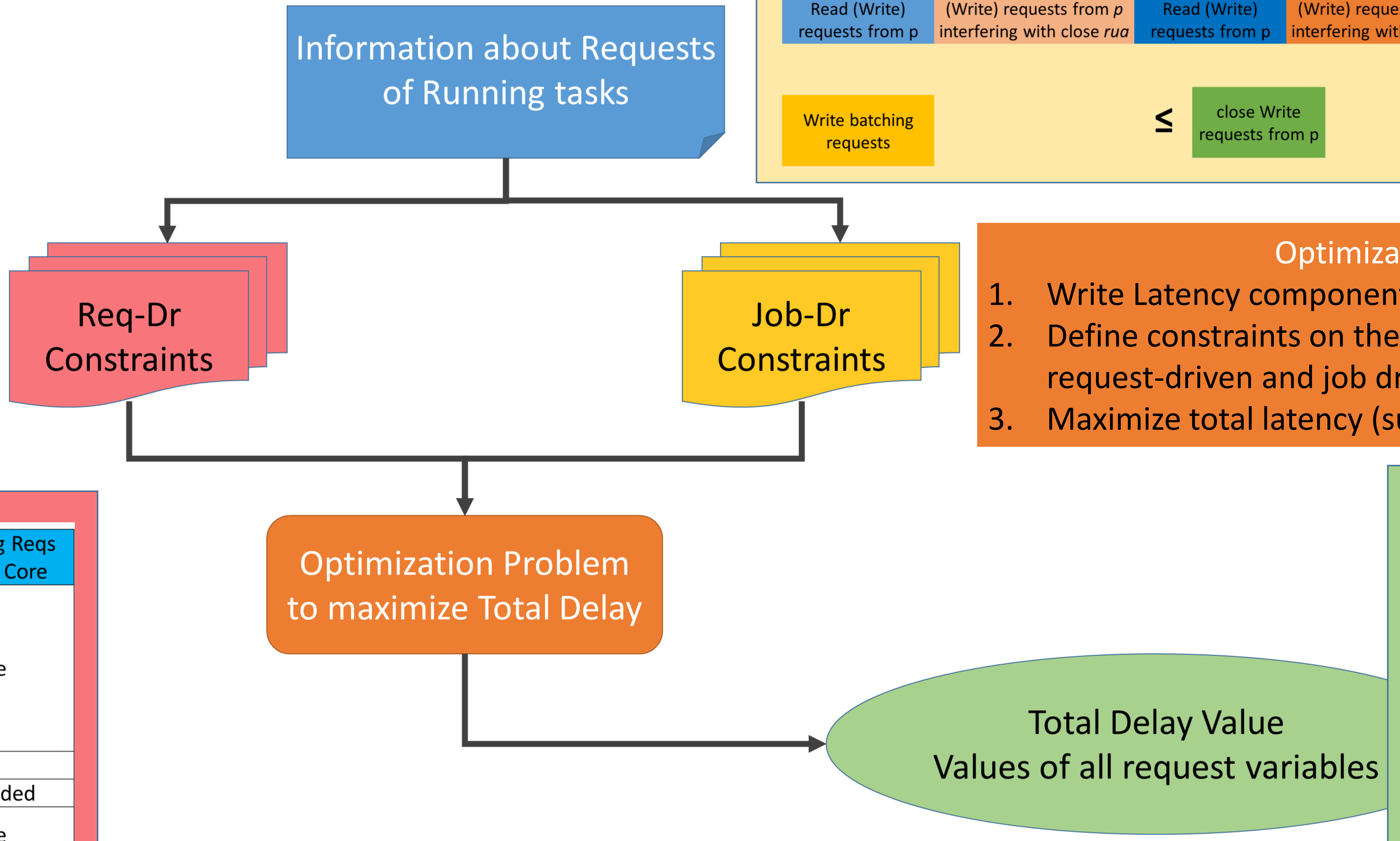
RESULTS


Intra-bank interfering requests:
1. Intra-bank conflict requests
2. Intra-bank reorder requests
Inter-bank interfering requests:
3. Inter-bank close requests
4. Inter-bank open requests
Write batching
5. Write batching requests
Write batching requests
Inter-bank close requests
ACT Latency
Inter-bank open requests
Inter-bank close requests
Intra-bank reorder requests
CAS Latency
Intra-bank conflict requests
Conflict Latency

Summary



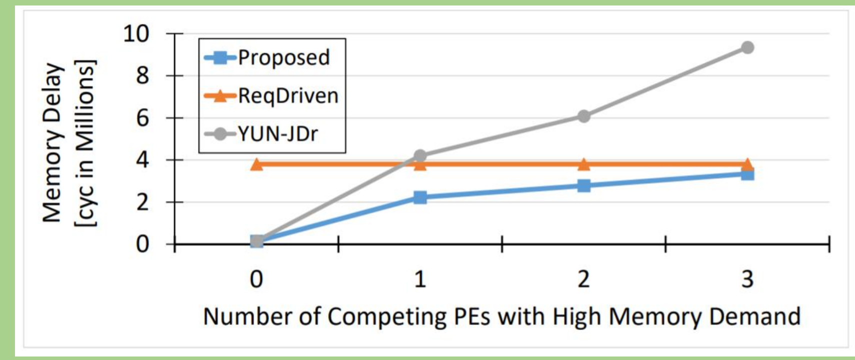
Intra-bank conflict Read (Write) requests from *p*	Inter-bank close Read (Write) requests from *p* interfering with close *rua*		≤	close Read (Write) requests from p
Intra-bank reorder Read (Write) requests from p	Inter-bank open Read (Write) requests from p interfering with close *rua*		≤	open Read (Write) requests from p
Intra-bank conflict Read (Write) requests from p	Inter-bank close Read (Write) requests from *p* interfering with close *rua*	Intra-bank reorder Read (Write) requests from p / Inter-bank open Read (Write) requests from p interfering with close *rua* / Inter-bank Read (Write) requests from *p* interfering with open *rua*	≤	Total Reads (writes) from *p*
Write batching requests			≤	close Write requests from p

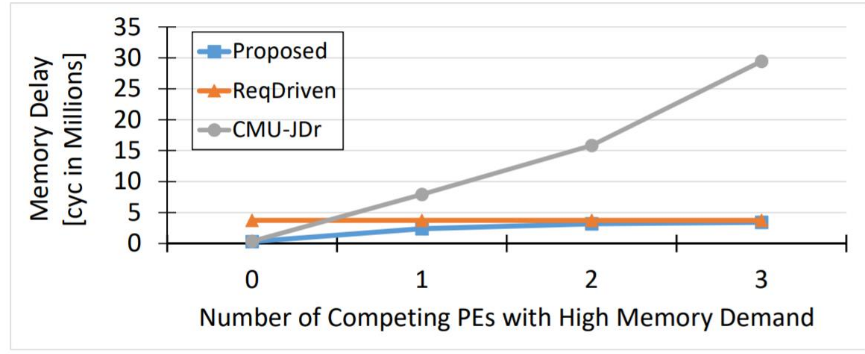


Optimization problem:

1. Write Latency components as functions on those requests
2. Define constraints on the number of requests based on request-driven and job driven analysis
3. Maximize total latency (summation of all components)

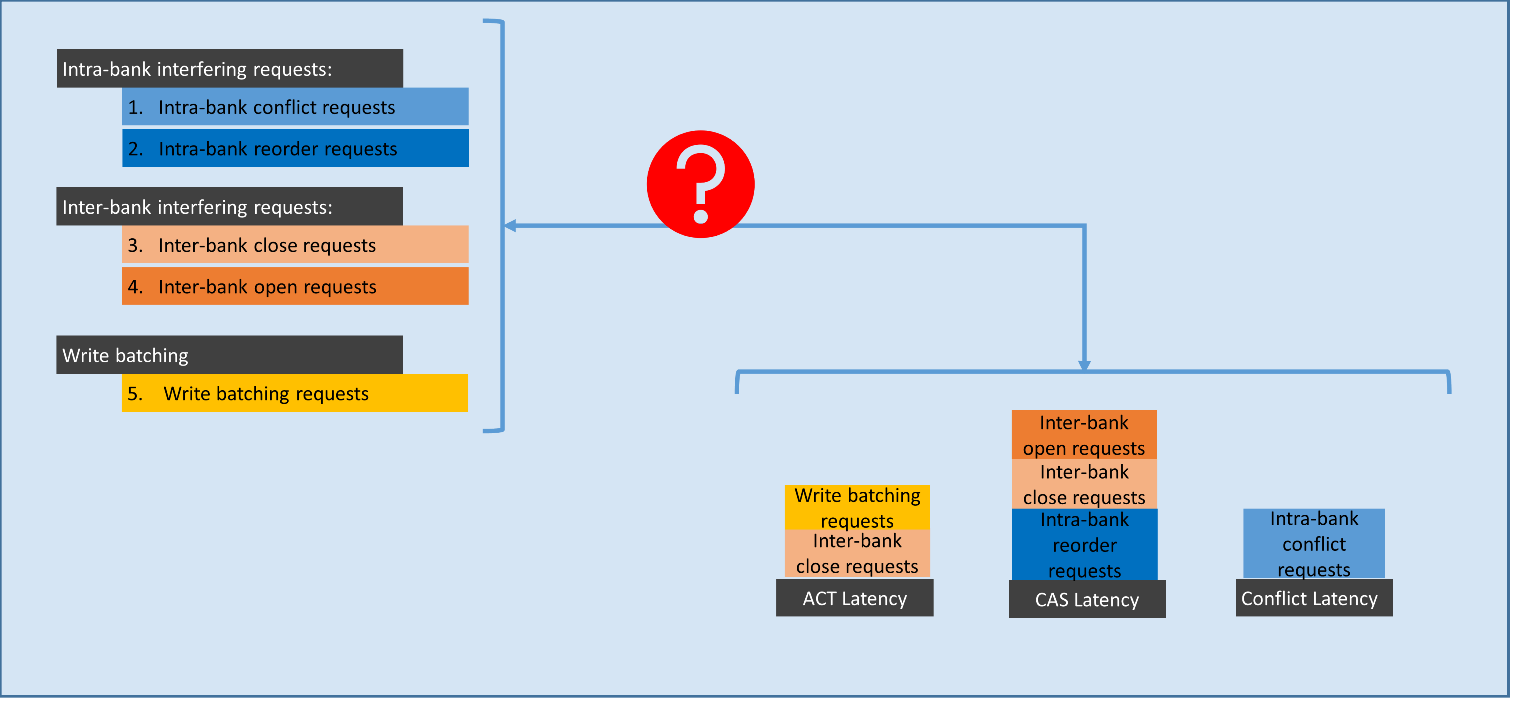
			Interfering Reqs from Cr Core	Interfering Reqs from nCr Core
Part-All	Priority	FR-FCFS thr	None	None
		No thr		
	No-Priority	FR-FCFS thr		
		No thr		
Part-Cr	Priority	FR-FCFS thr		
		No thr		
	No-Priority	FR-FCFS thr		*
		No thr		unbounded
No-Part	Priority	FR-FCFS thr	*	None
		No thr	Unbounded	
	No-Priority	FR-FCFS thr	*	*
		No thr	Unbounded	Unbounded





Summary

https://gitlab.com/FanusLab/memory-contention-analysis

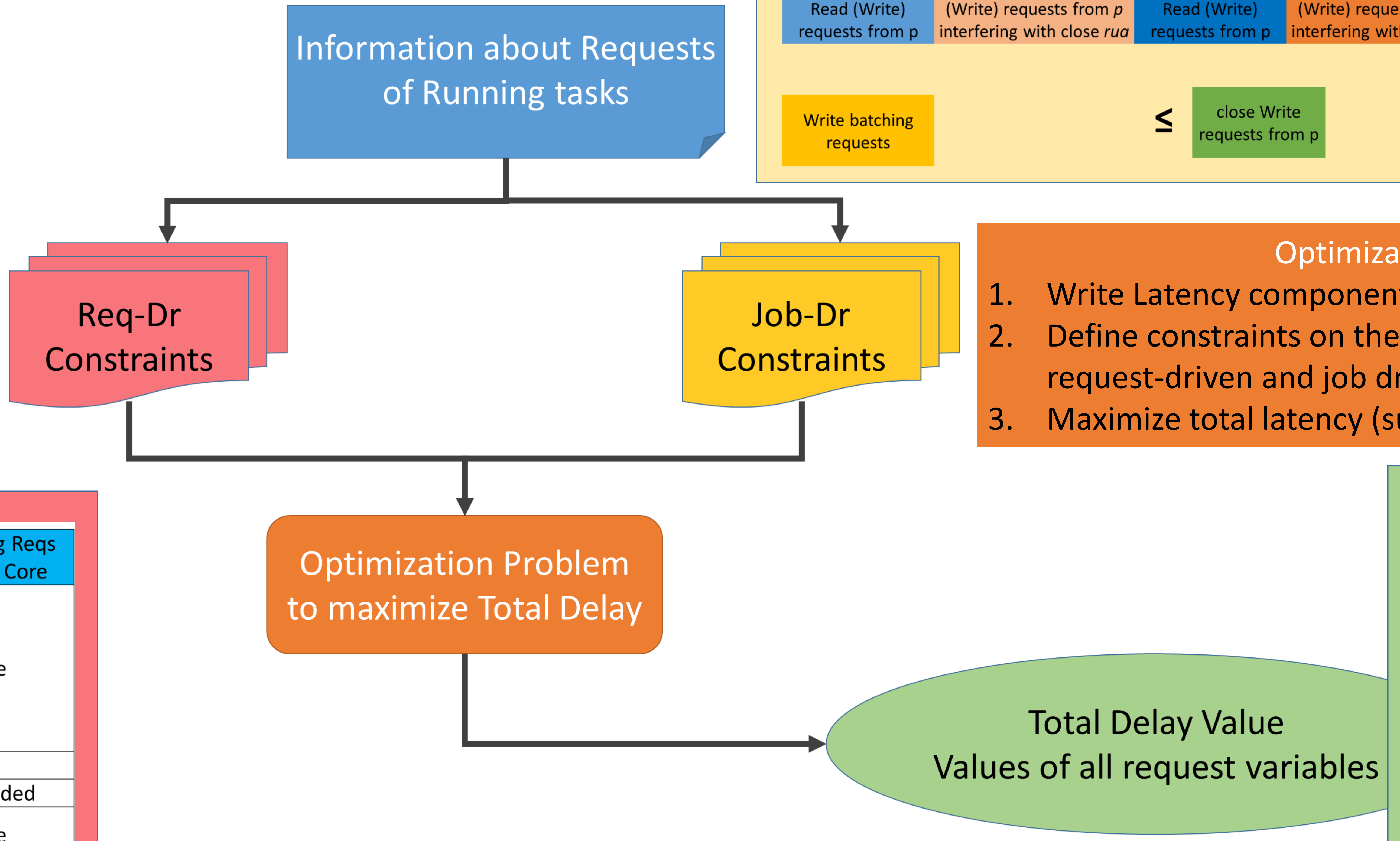


Intra-bank conflict Read (Write) requests from *p* + Inter-bank close Read (Write) requests from *p* interfering with close *rua* ≤ close Read (Write) requests from p

Intra-bank reorder Read (Write) requests from p + Inter-bank open Read (Write) requests from p interfering with close *rua* ≤ open Read (Write) requests from p

Intra-bank conflict Read (Write) requests from p + Inter-bank close Read (Write) requests from p interfering with close *rua* + Intra-bank reorder Read (Write) requests from p + Inter-bank open Read (Write) requests from p interfering with close *rua* + Inter-bank Read (Write) requests from *p* interfering with open *rua* ≤ Total Reads (writes) from *p*

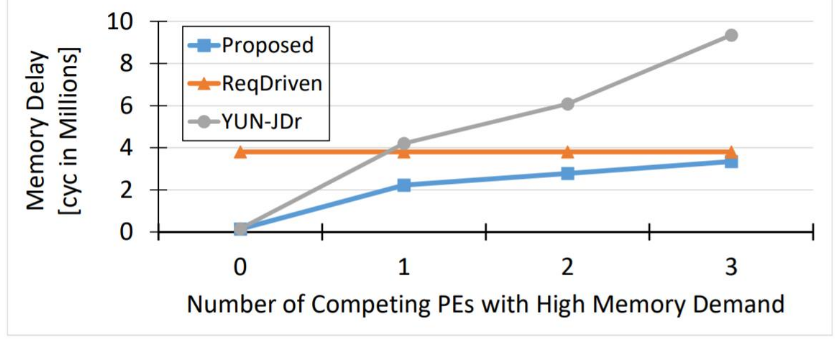
Write batching requests ≤ close Write requests from p

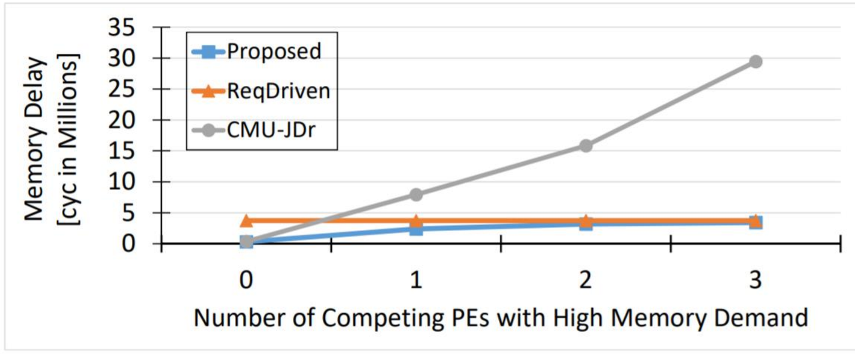


Optimization problem:

1. Write Latency components as functions on those requests
2. Define constraints on the number of requests based on request-driven and job driven analysis
3. Maximize total latency (summation of all components)

			Interfering Reqs from Cr Core	Interfering Reqs from nCr Core
Part-All	Priority	FR-FCFS thr	None	None
		No thr		
	No-Priority	FR-FCFS thr		
		No thr		
Part-Cr	Priority	FR-FCFS thr		
		No thr		
	No-Priority	FR-FCFS thr		*
		No thr		unbounded
No-Part	Priority	FR-FCFS thr	*	None
		No thr	Unbounded	
	No-Priority	FR-FCFS thr	*	*
		No thr	Unbounded	Unbounded





https://gitlab.com/FanusLab/memory-contention-analysis

mohamed.hassan@mcmaster.ca

Summary