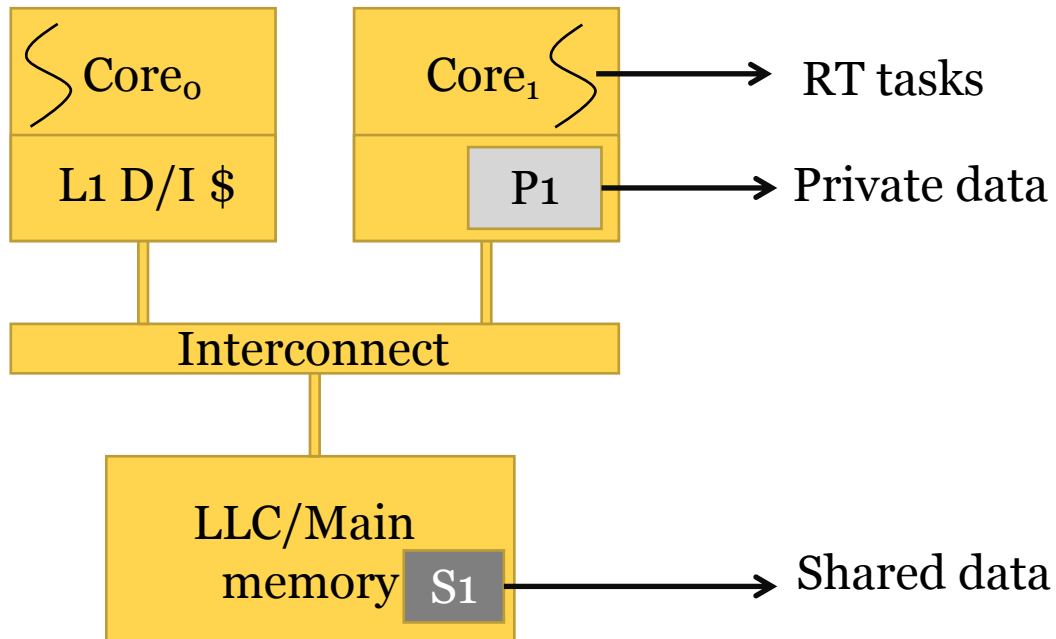


Predictable Cache Coherence for Multi-Core Real-Time Systems

Mohamed Hassan, **Anirudh M. Kaushik** and Hiren Patel

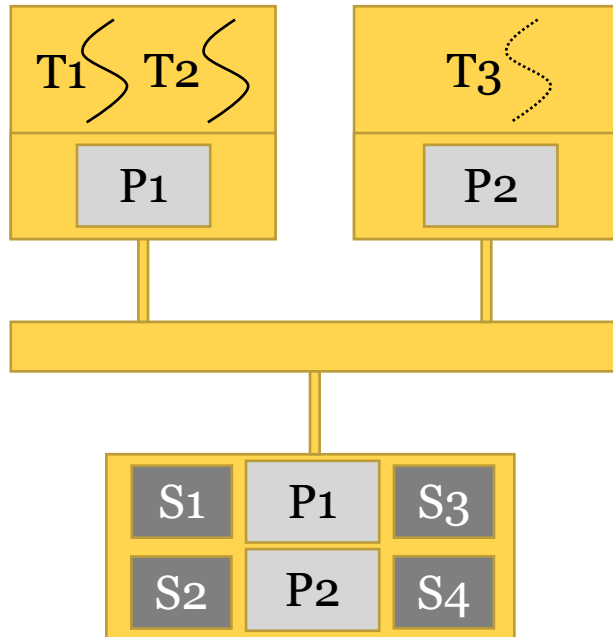
RTAS 2017

Motivation: Data sharing in multi-core real-time systems



Motivation: Data sharing in multi-core real-time systems

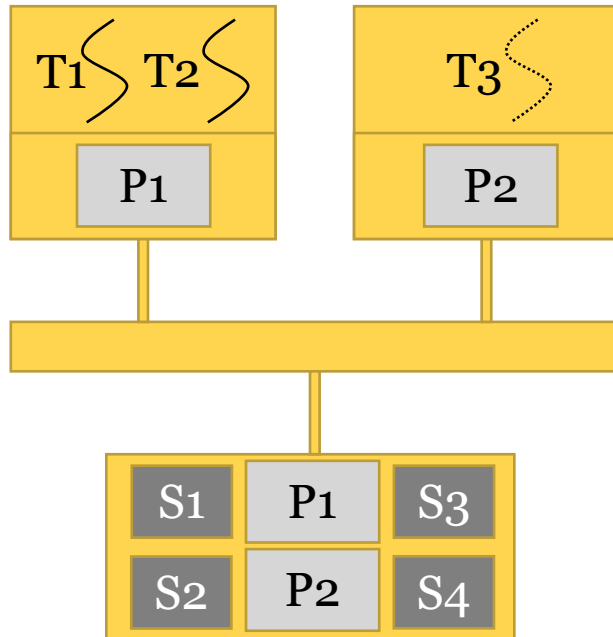
No caching of shared data
[RTSS'09, RTNS'10]



- ✓ Simpler timing analysis
- ✗ Hardware support
- ✗ Long execution time

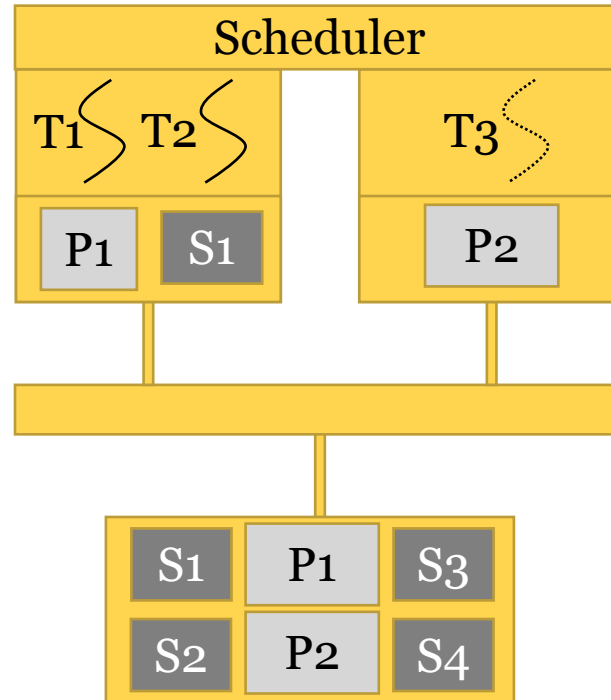
Motivation: Data sharing in multi-core real-time systems

No caching of shared data
[RTSS'09, RTNS'10]



- ✓ Simpler timing analysis
- ✗ Hardware support
- ✗ Long execution time

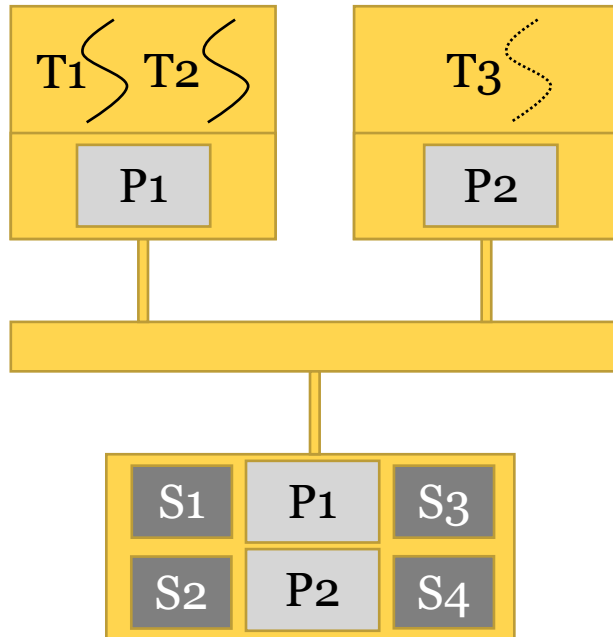
Task scheduling on shared data
[ECRTS'10, RTSS'16]



- ✓ Private cache hits on shared data
- ✓ No hardware support
- ✗ Limited multi-core parallelism
- ✗ Changes to OS scheduler

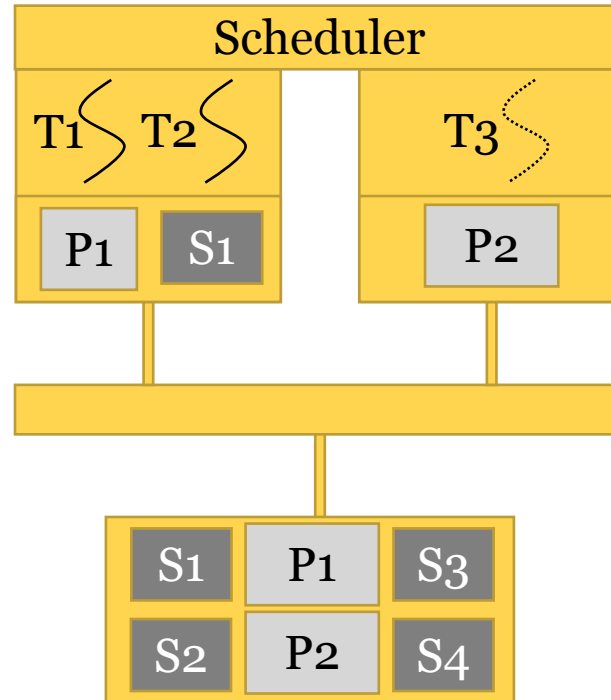
Motivation: Data sharing in multi-core real-time systems

No caching of shared data
[RTSS'09, RTNS'10]



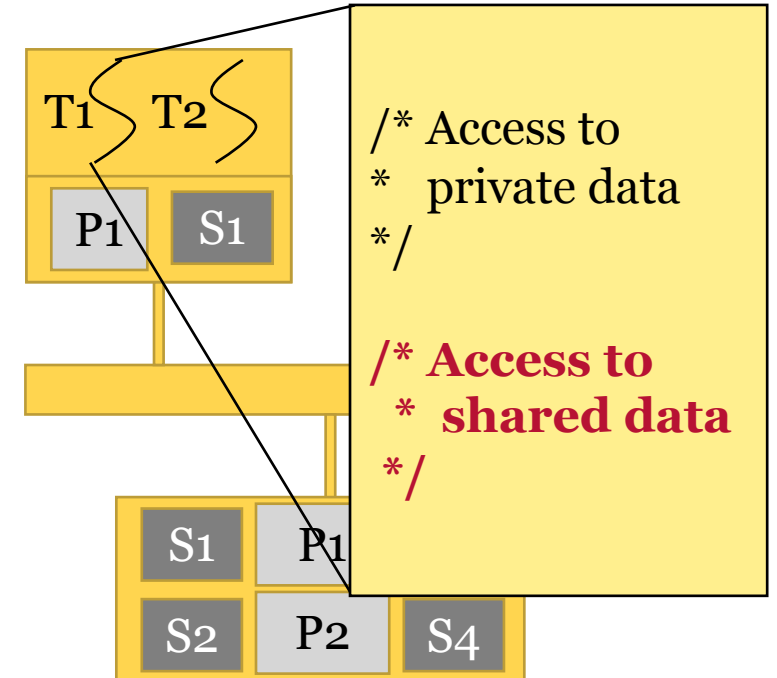
- ✓ Simpler timing analysis
- ✗ Hardware support
- ✗ Long execution time

Task scheduling on shared data
[ECRTS'10, RTSS'16]



- ✓ Private cache hits on shared data
- ✓ No hardware support
- ✗ Limited multi-core parallelism
- ✗ Changes to OS scheduler

Software cache coherence
[SAMOS'14]



- ✓ Private cache hits on shared data
- ✗ Hardware support
- ✗ Source code changes

Motivation: Data sharing in multi-core real-time systems

No caching of shared data

[RTSS'09, RTNS'10]

Task scheduling on shared data

[ECRTS'10, RTSS'16]

Software cache coherence

[SAMOS'14]

Disabling simultaneous cached shared data for **timing analysis** at the cost of **lower performance and OS and application changes.**

```
/* Access to  
* private data  
*/  
/* Access to  
* shared data  
*/
```

- ✓ Simpler timing analysis
- ✗ Hardware support
- ✗ Long execution time

- ✓ Private cache hits on shared data
- ✓ No hardware support
- ✗ Limited multi-core parallelism
- ✗ Changes to OS scheduler

- ✓ Private cache hits on shared data
- ✗ Hardware support
- ✗ Source code changes

Motivation: Data sharing in multi-core real-time systems

No caching of shared data

[RTSS'09, RTNS'10]

Task scheduling on shared data

[ECRTS'10, RTSS'16]

Software cache coherence

[SAMOS'14]

This work:

Allowing **predictable simultaneous cached shared data accesses** through hardware cache coherence, and provide **significant performance improvements with no changes to OS and applications.**

✓ Simpler timing analysis

✗ Hardware support

✗ Long execution time

✓ Private cache hits on shared data

✓ No hardware support

✗ Limited multi-core parallelism

✗ Changes to OS scheduler

✓ Private cache hits on shared data

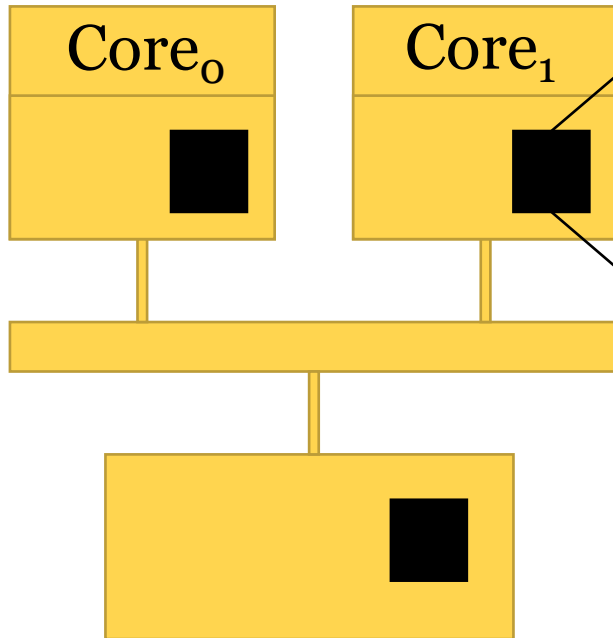
✗ Hardware support

✗ Source code changes

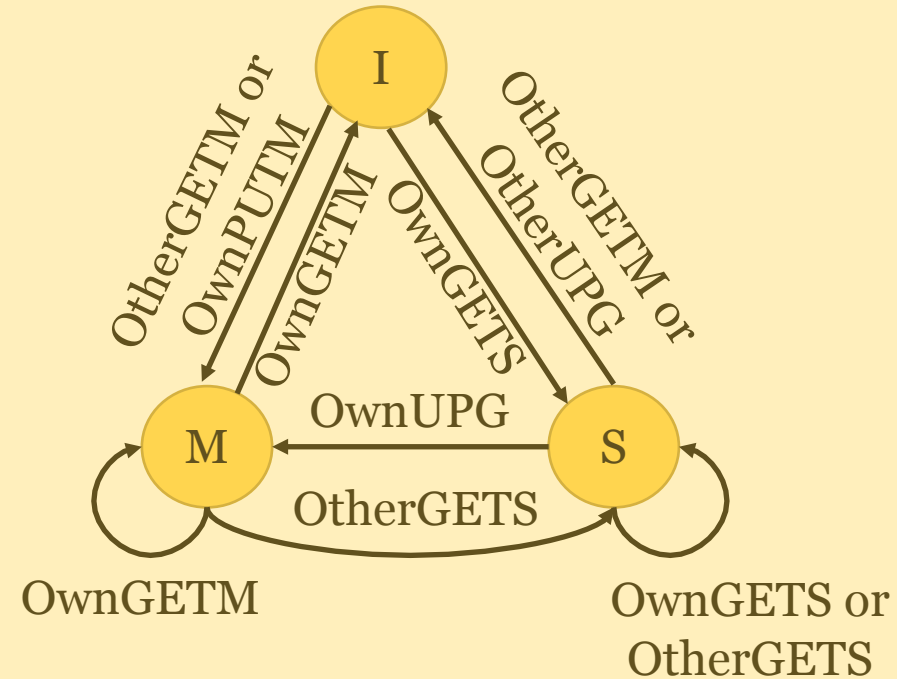
Outline

- Overview of hardware cache coherence
- Challenges of applying conventional hardware cache coherence on RT multi-core systems
- Our solution: **P**redictable **M**odified-**S**hared-**I**nvalid (PMSI) cache coherence protocol
- Latency analysis of PMSI
- Results
- Conclusion

Overview of hardware cache coherence

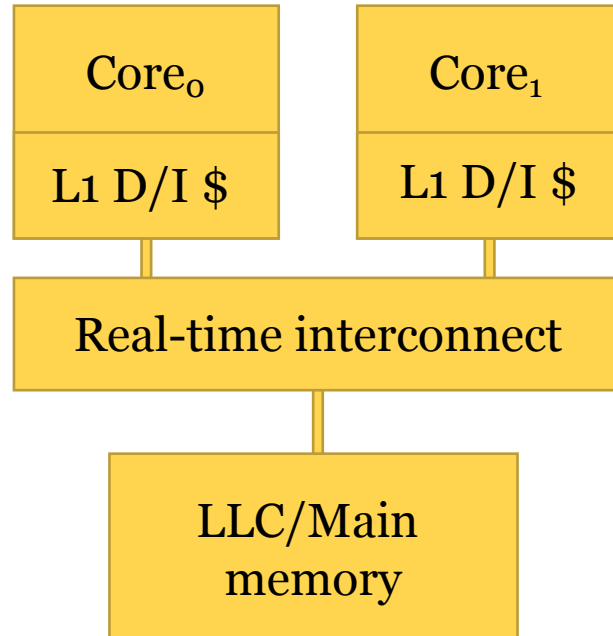


Modified-Shared-Invalid protocol

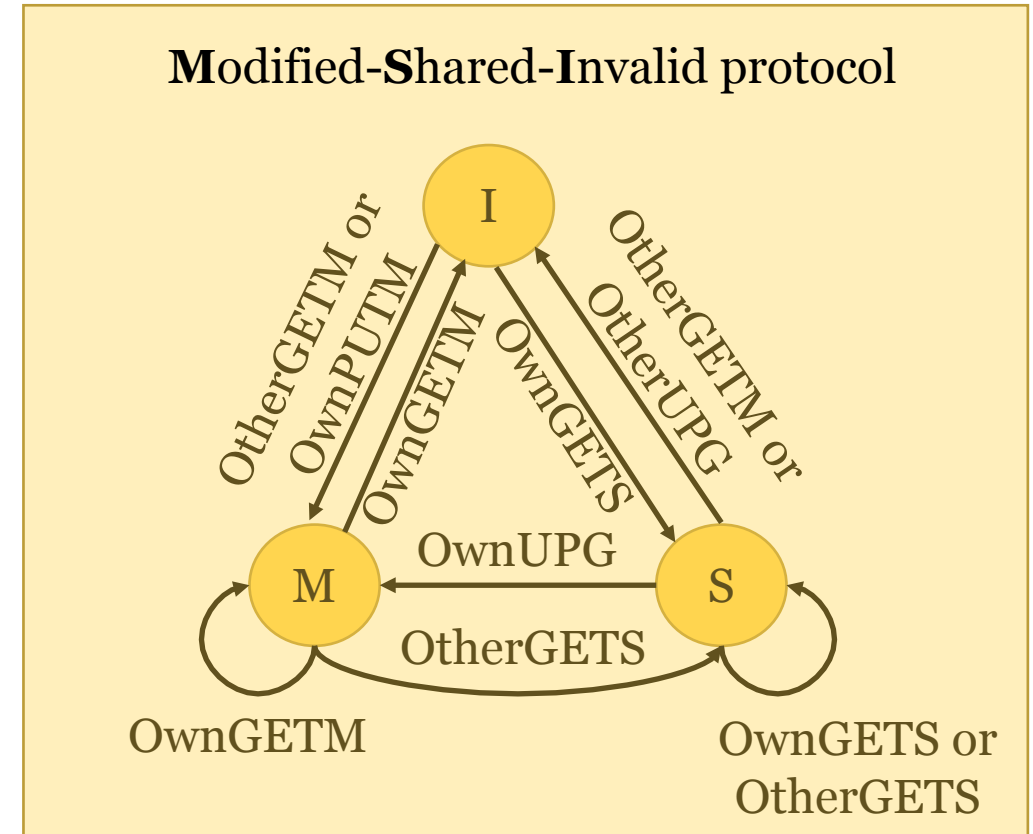


GETS: Memory load
GETM/UPG: Memory store
PUTM: Replacement

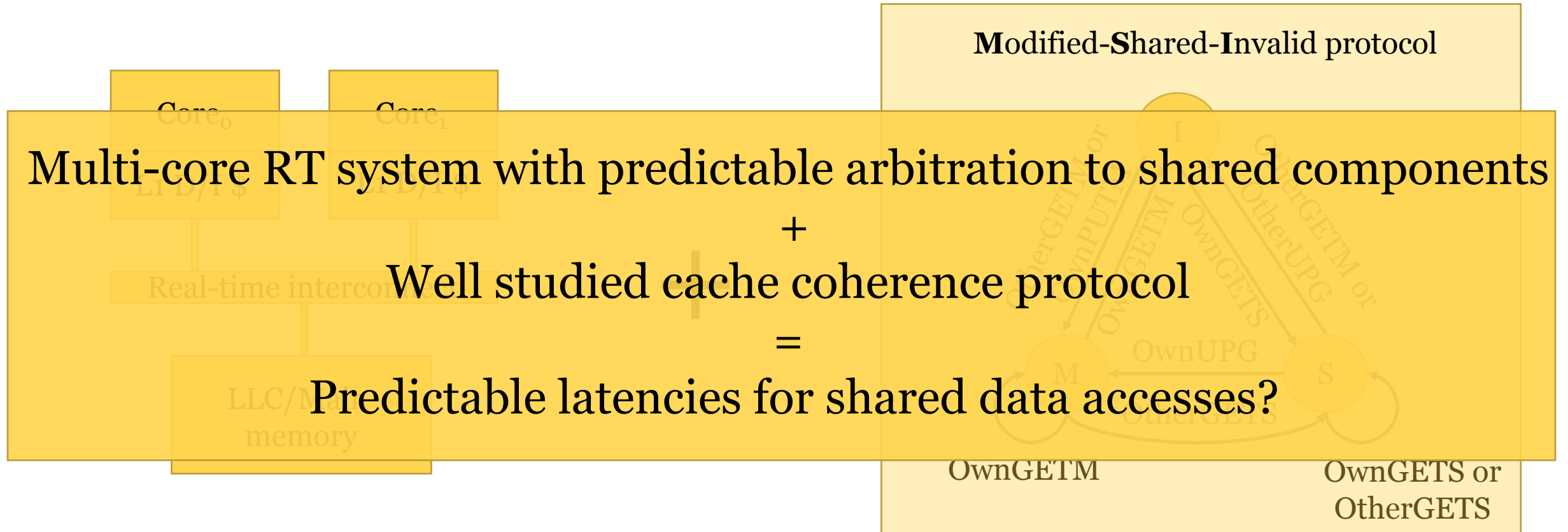
Applying hardware cache coherence to multi-core RT systems



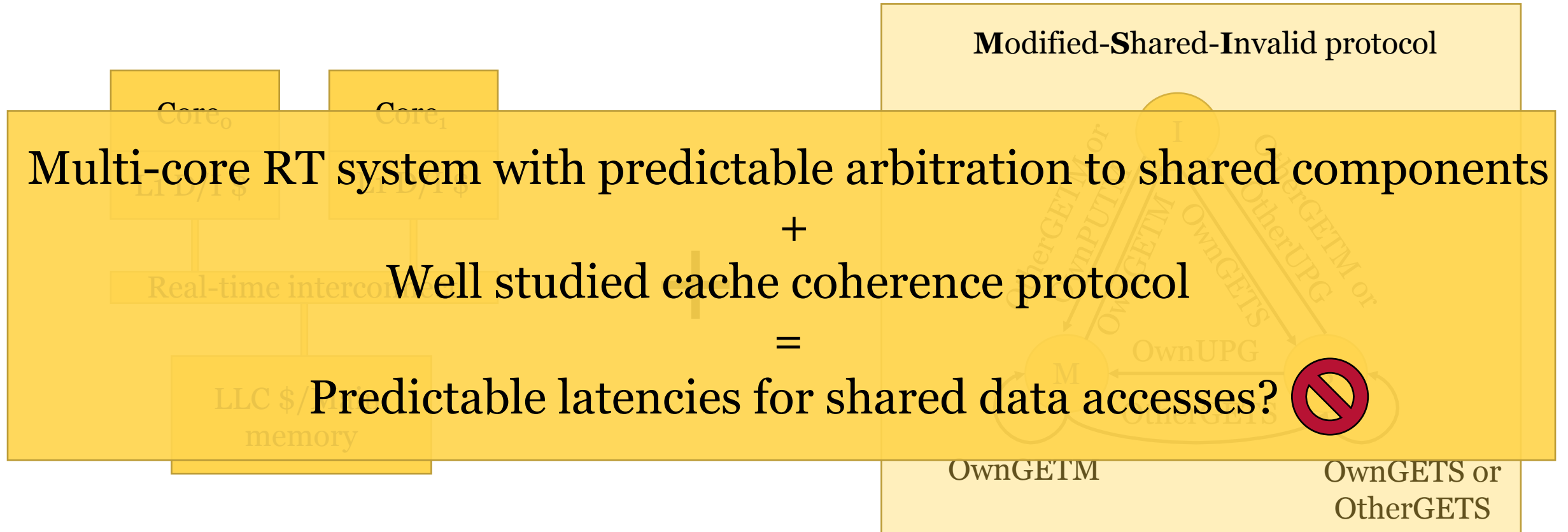
+



Applying hardware cache coherence to multi-core RT systems



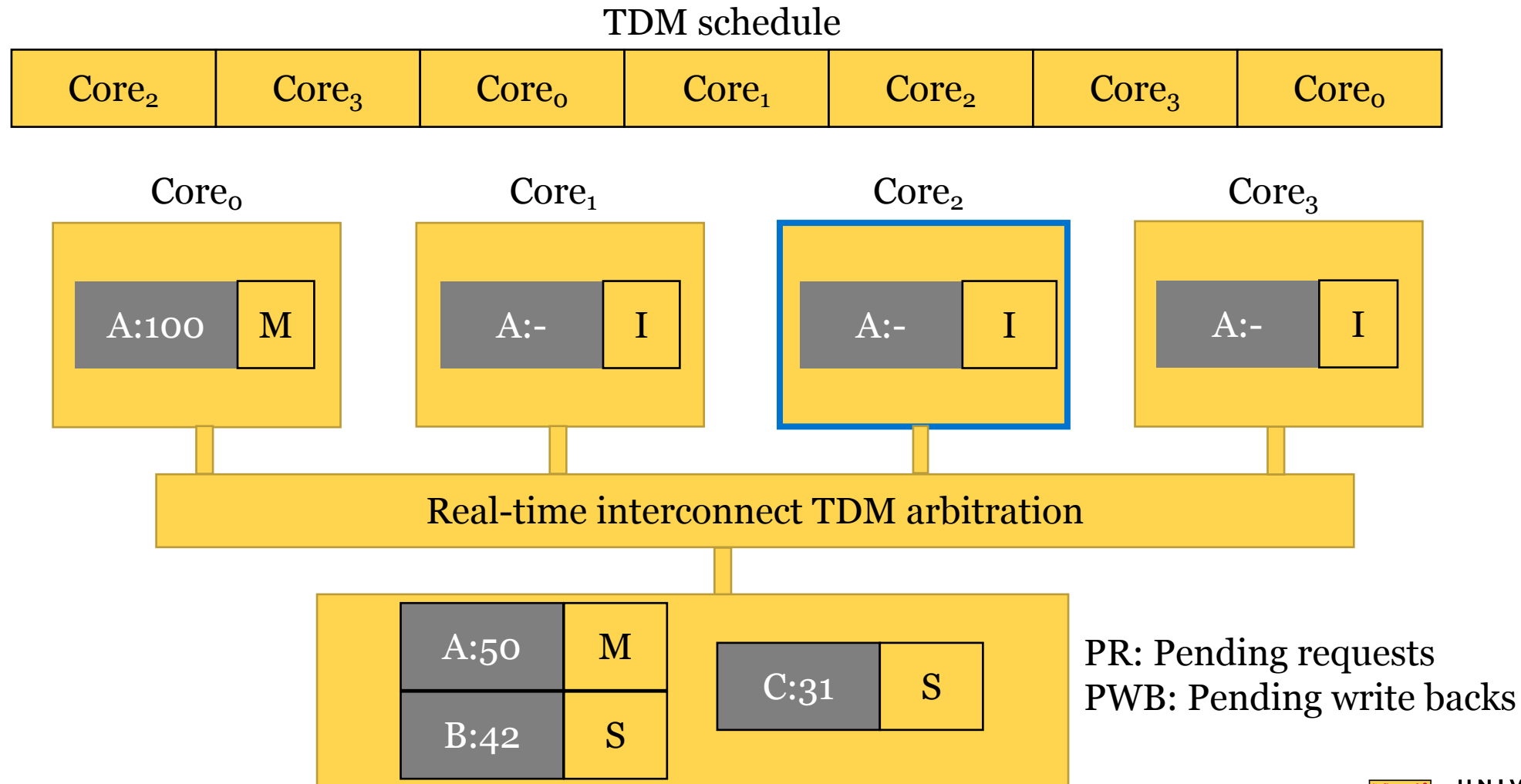
Applying hardware cache coherence to multi-core RT systems



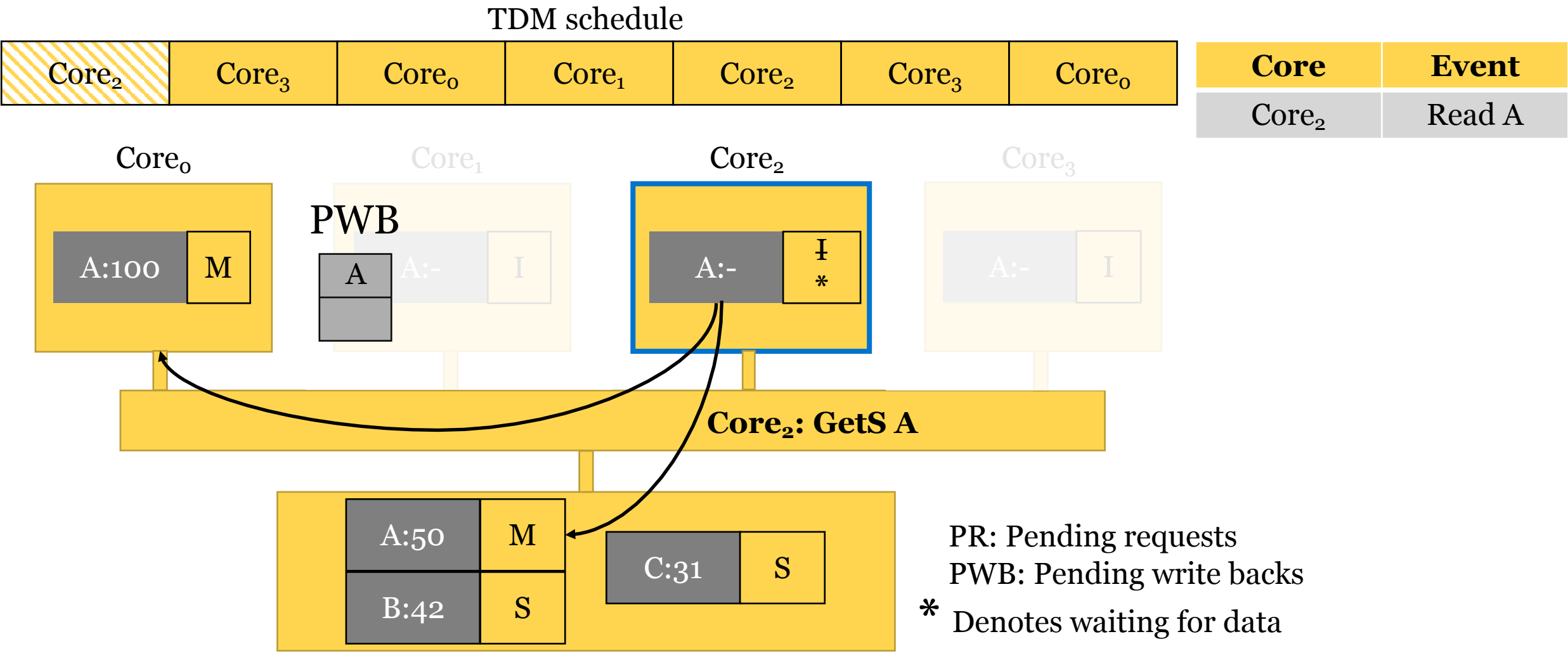
Sources of unpredictable memory access latencies due to hardware cache coherence

1. Inter-core coherence interference on same cache line
2. Inter-core coherence interference on different cache lines
3. Inter-core coherence interference due to write hits
4. **Intra-core coherence interference**

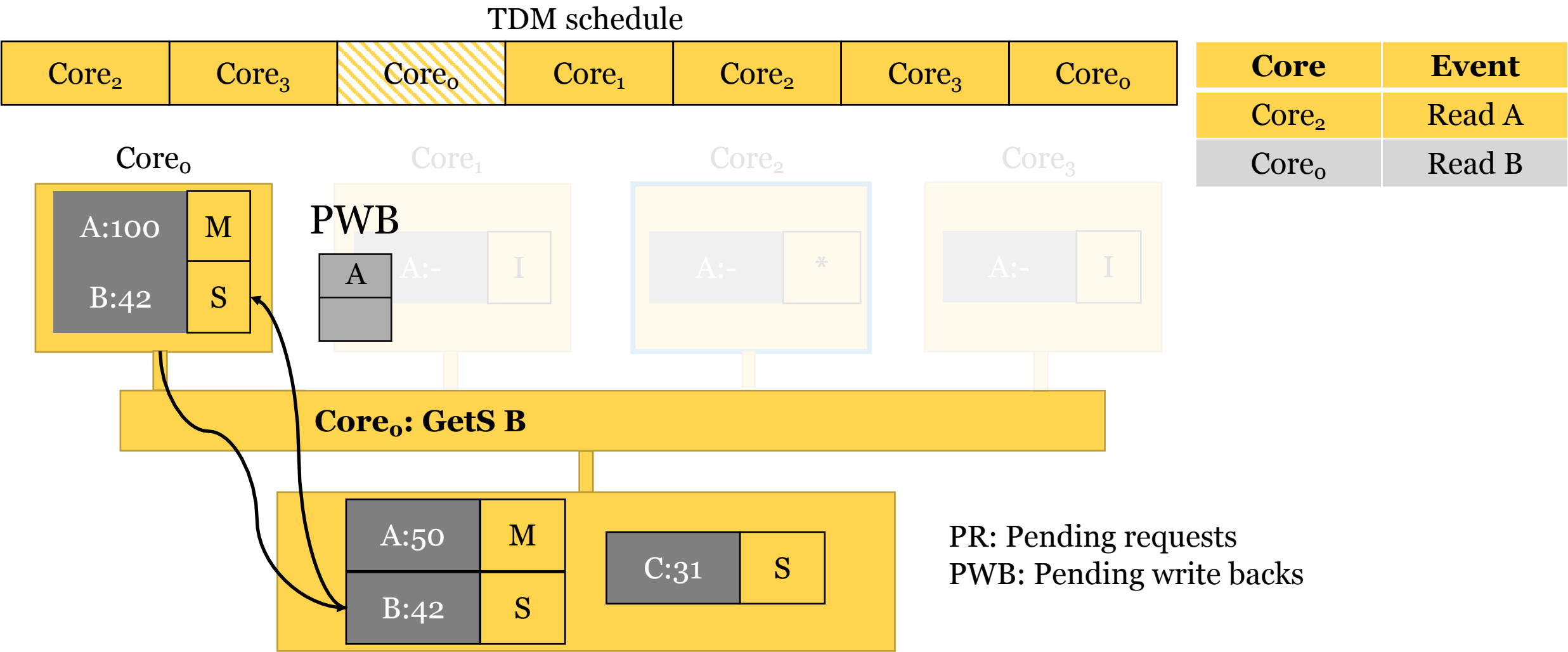
System model



Intra-core coherence interference



Intra-core coherence interference

**Core₀: GetS B**

A:50

 M

B:42

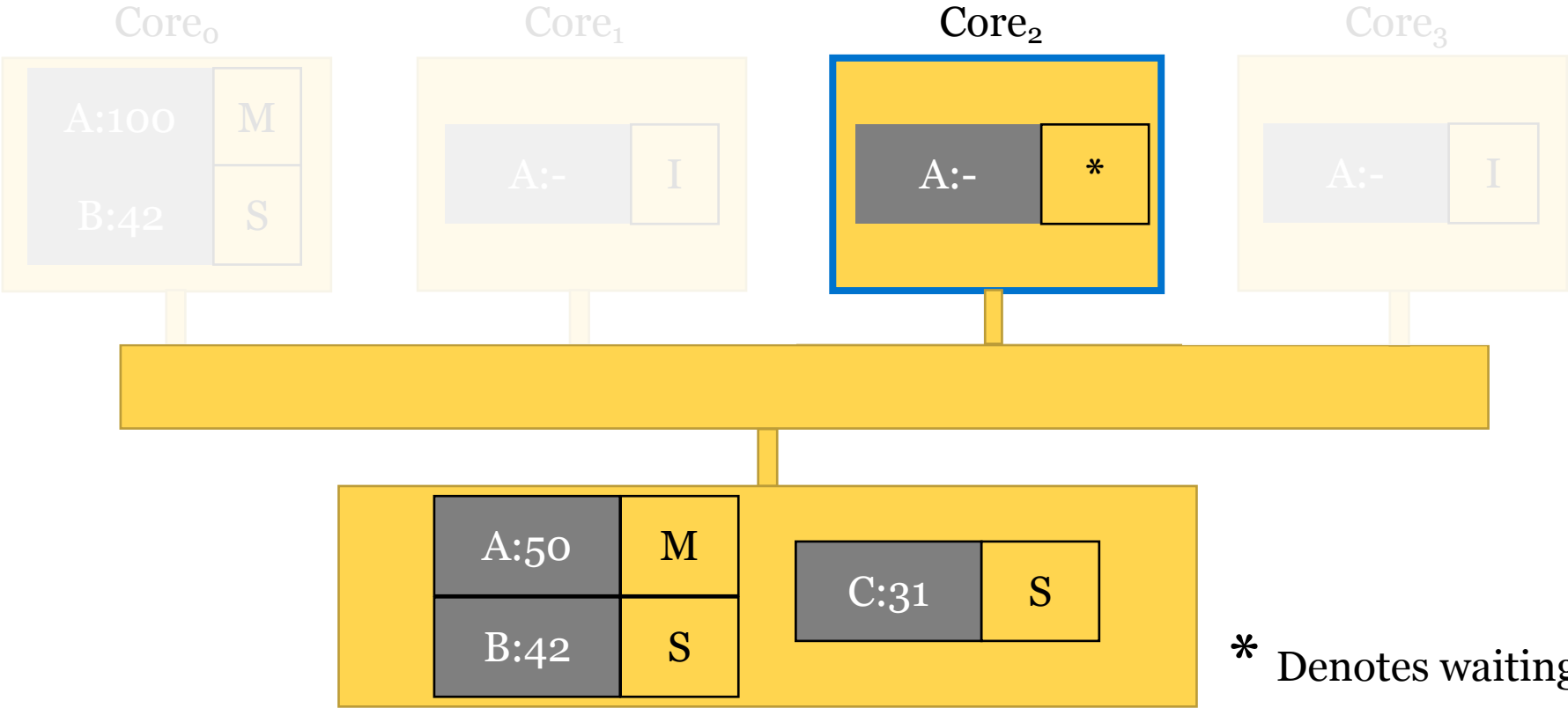
 S

C:31

PR: Pending requests
PWB: Pending write backs

Intra-core coherence interference

TDM schedule

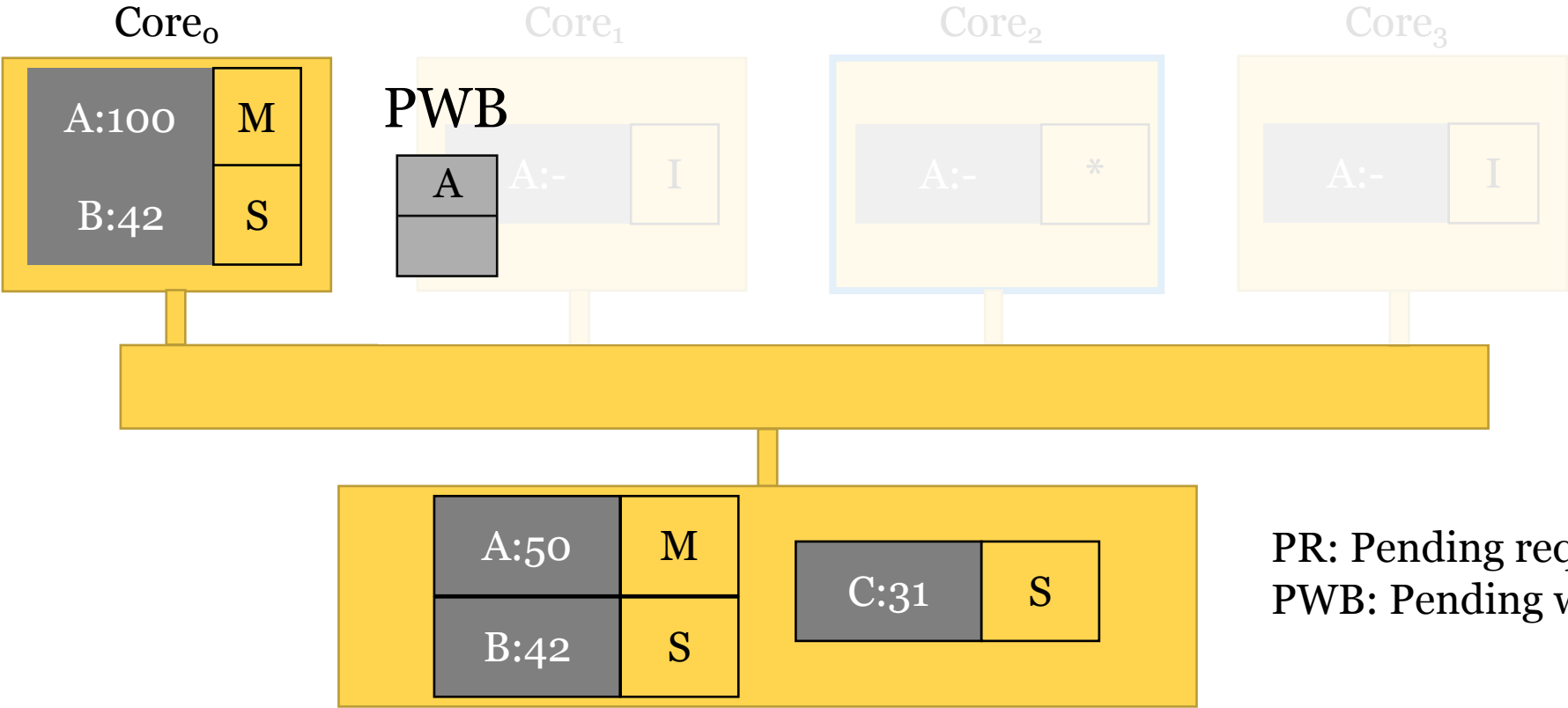


Core	Event
Core ₂	Read A
Core ₀	Read B
Core ₂	Waiting for data

* Denotes waiting for data

Intra-core coherence interference

TDM schedule

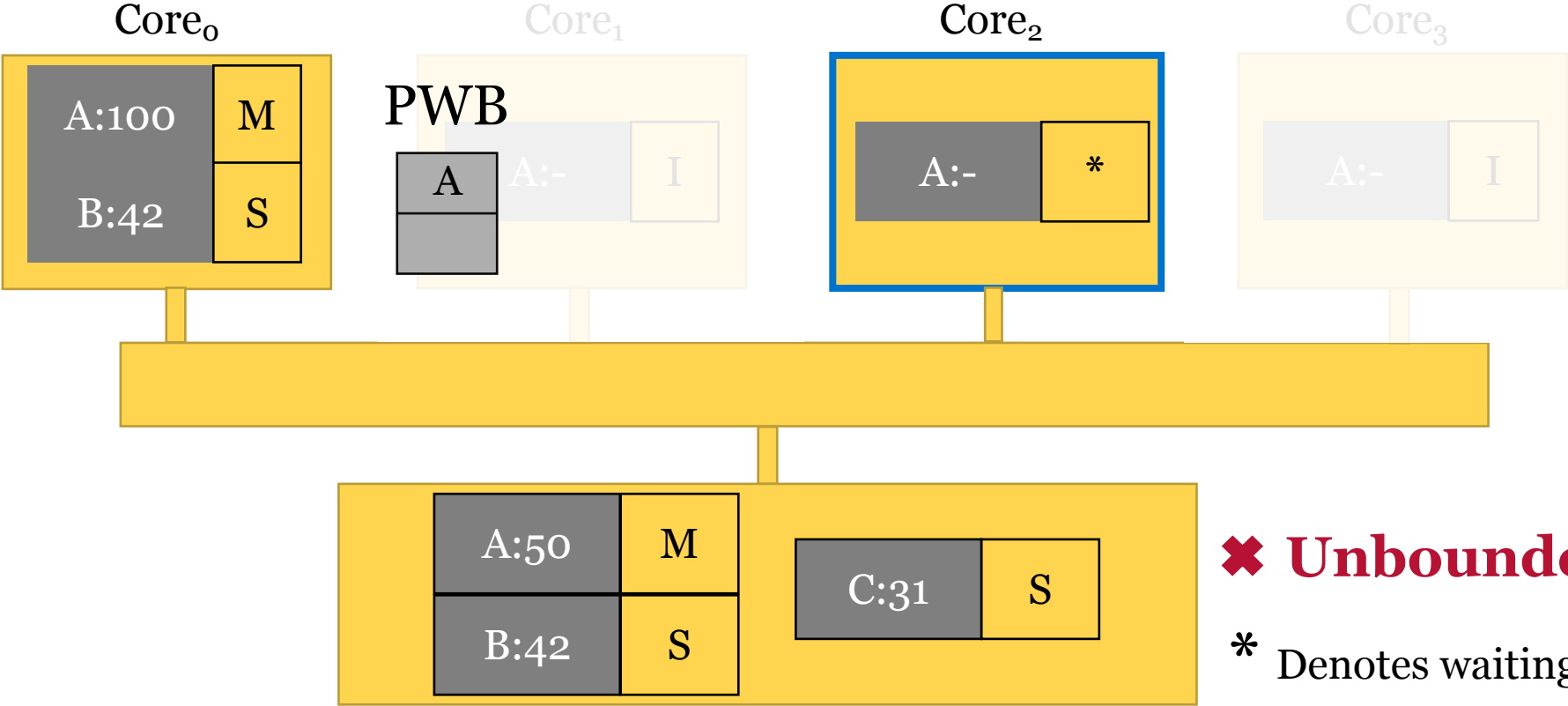


Core	Event
Core ₂	Read A
Core ₀	Read B
Core ₂	Waiting for data
Core ₀	Read C

PR: Pending requests
PWB: Pending write backs

Intra-core coherence interference

TDM schedule

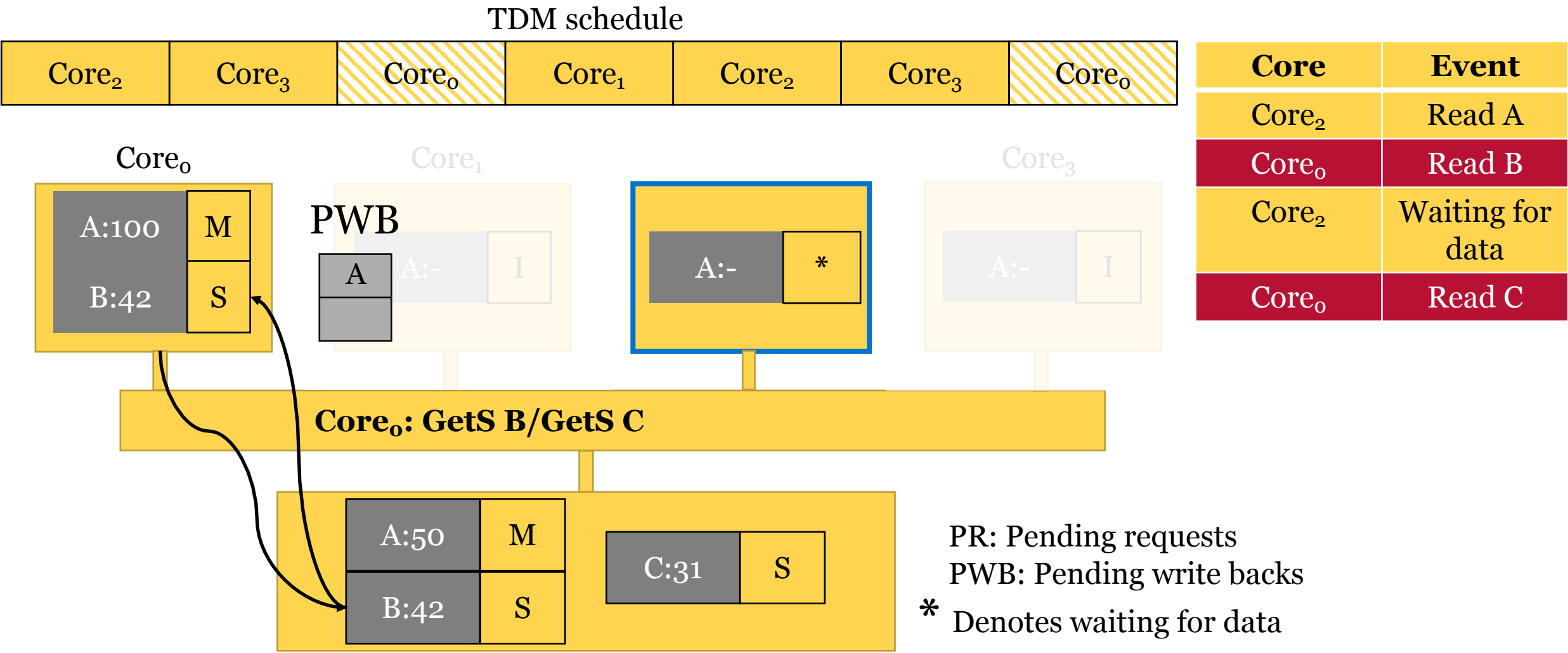


Core	Event
Core ₂	Read A
Core ₀	Read B
Core ₂	Waiting for data
Core ₀	Read C

✖ Unbounded latency for Core₂

* Denotes waiting for data

Intra-core coherence interference



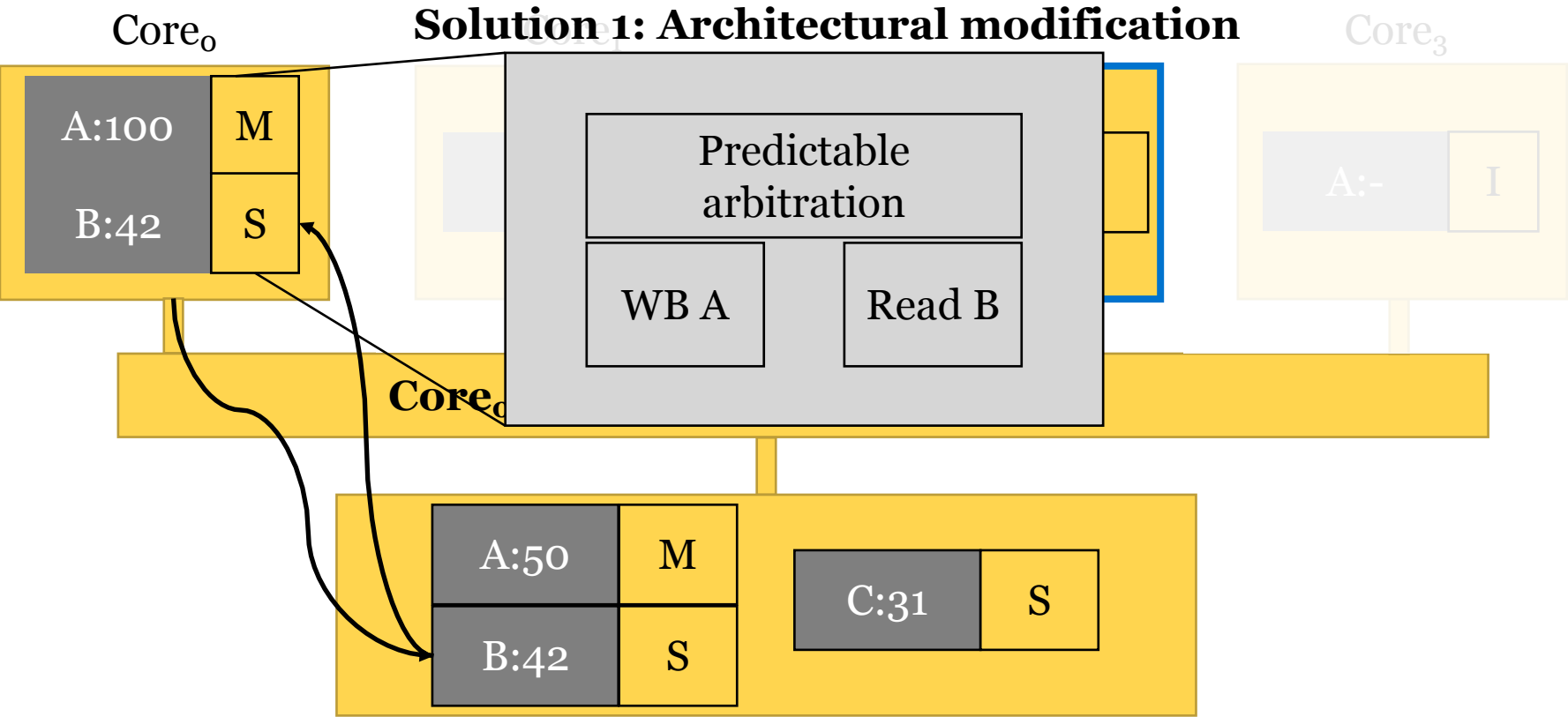
Core	Event
Core ₂	Read A
Core ₀	Read B
Core ₂	Waiting for data
Core ₀	Read C

Intra-core coherence interference

TDM schedule



Solution 1: Architectural modification



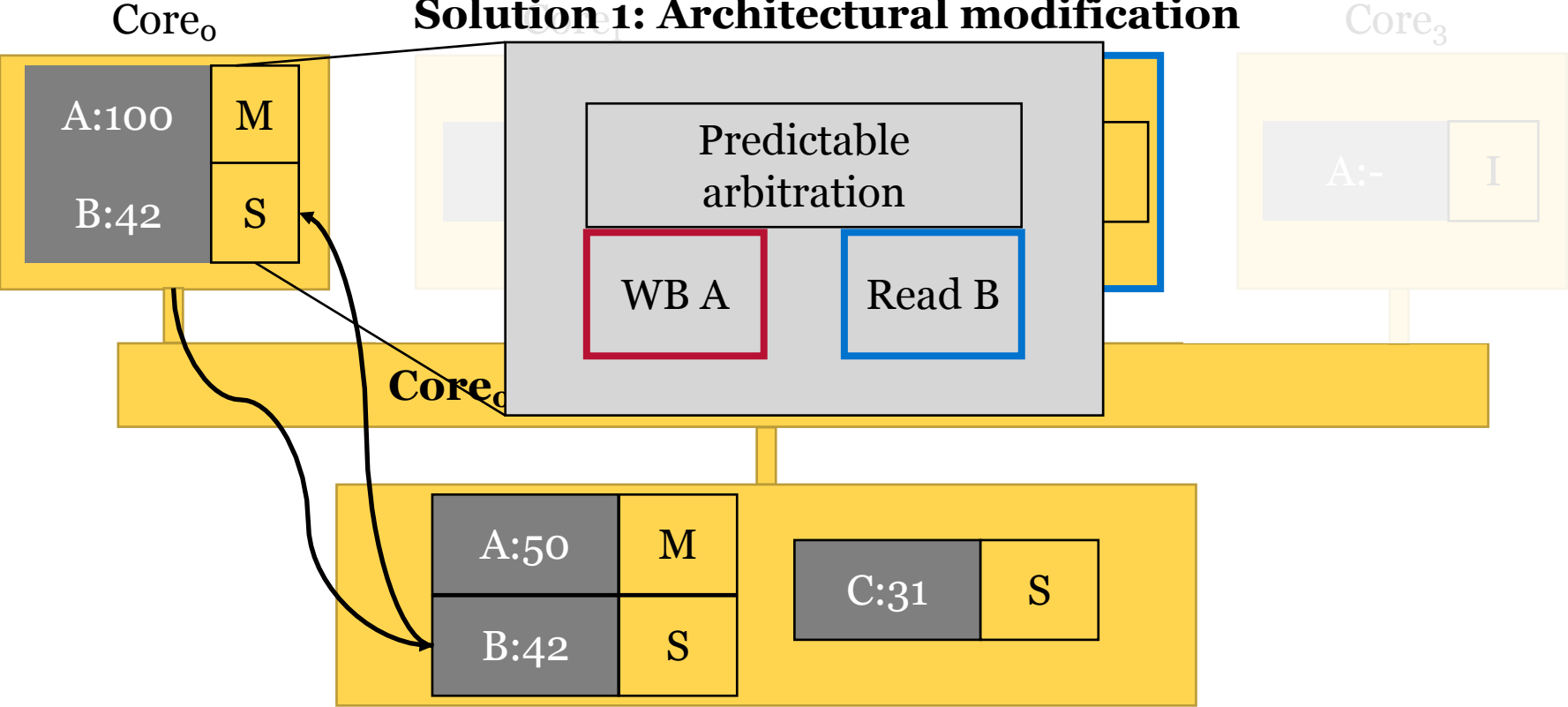
Core	Event
Core ₂	Read A
Core ₀	Read B
Core ₂	Waiting for data
Core ₀	Read C

Intra-core coherence interference

TDM schedule



Solution 1: Architectural modification



Core	Event
Core ₂	Read A
Core ₀	Read B
Core ₂	Waiting for data
Core ₀	Read C

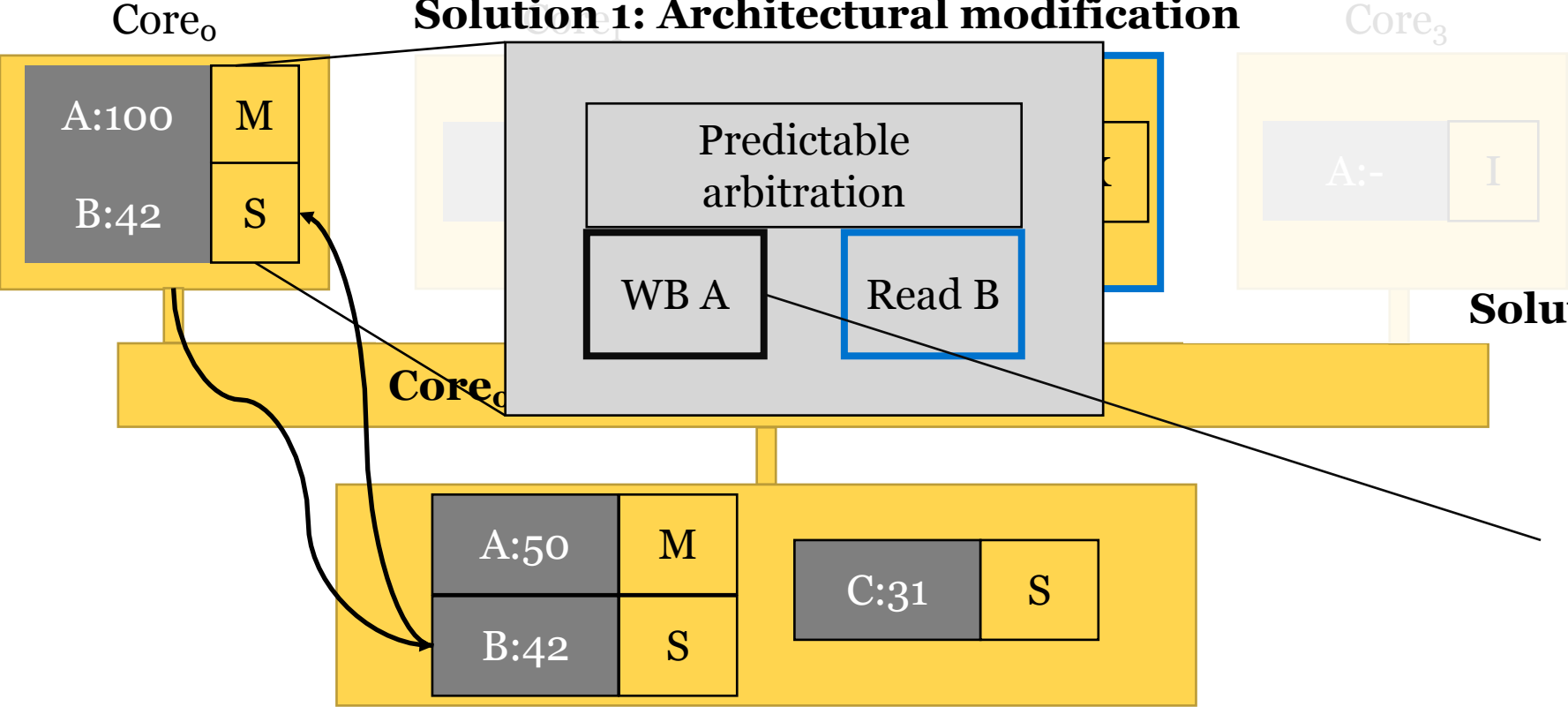
Intra-core coherence interference

TDM schedule



Core	Event
Core ₂	Read A
Core ₀	Read B
Core ₂	Waiting for data
Core ₀	Read C

Solution 1: Architectural modification



Solution 2: Coherence protocol modifications

Need to maintain state to indicate Core₀'s pending WB of A

Sources of unpredictable memory access latencies due to cache coherence

1. Inter-core coherence interference on same cache line

✓ **Architectural modification +
Coherence modifications =**

2. Inter-core coherence interference due to write hits
✓ **Predictable Modified-Shared-Invalidate cache
coherence protocol (PMSI)**

3. Inter-core coherence interference due to write hits
✓ **Predictable latencies for simultaneous shared data access**

✓ **No application/OS changes**

4. Intra-core coherence interference
✓ **Significant performance benefits**

✓ **Solution: Architectural modification to the core + Coherence modifications**

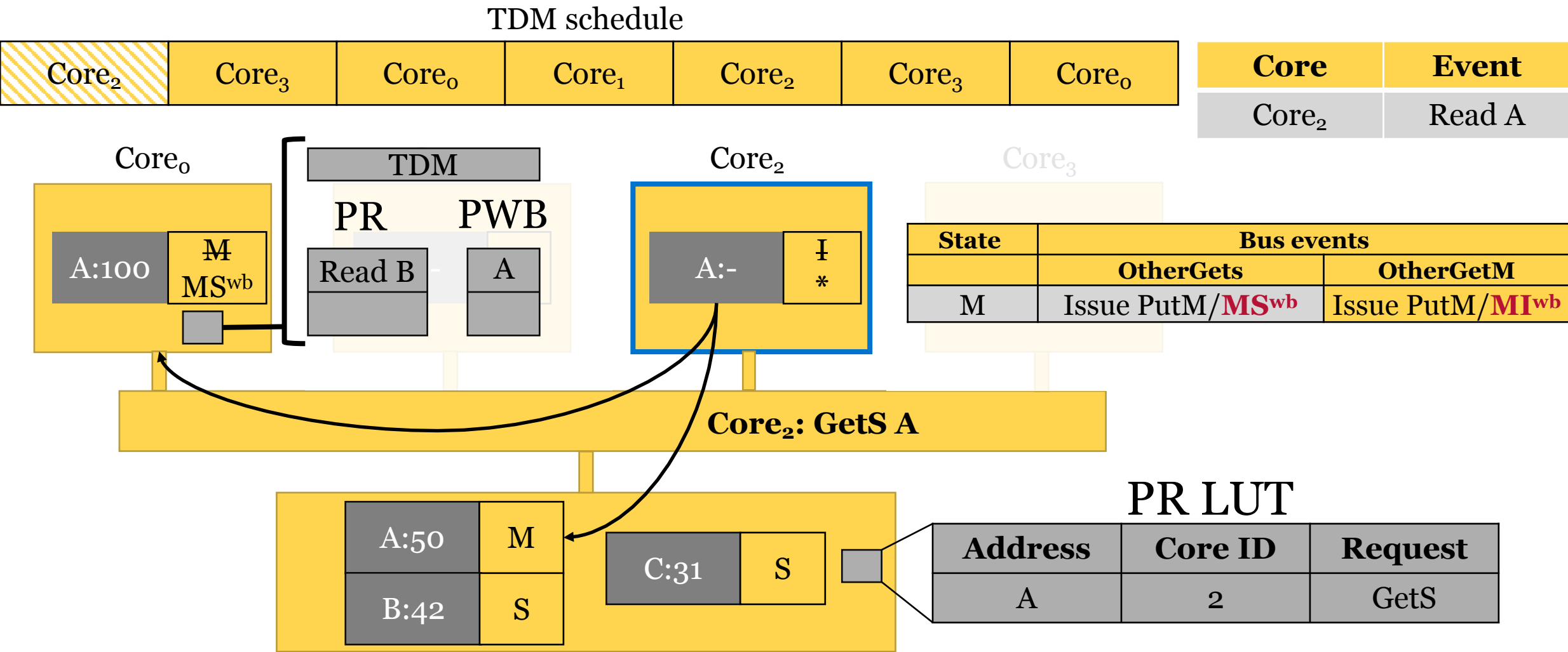
PMSI cache coherence protocol: Protocol modifications

State	Core events			Bus events						
	Load	Store	Replacement	OwnData	OwnUpg	OwnPutM	OtherGetS	OtherGetM	Other Upg	Other PutM
I	Issue GetS/IS ^d	Issue GetM/IM ^d	X	X	X	X	N/A	N/A	N/A	N/A
S	Hit	Issue Upg/ SM^w	X	N/A	X	X	N/A	I	I	X
M	Hit	Hit	Issue PutM/ MI^{wb}	X	X	X	Issue PutM/ MS^{wb}	Issue PutM/ MI^{wb}	X	X
IS ^d	X	X	X	Read/S	X	X	N/A	IS ^d I	IS ^d I	N/A
IM ^d	X	X	X	Write/S	X	X	IM ^d S	IM ^d I	X	N/A
IM ^d I	X	X	X	Write/ MI^{wb}	X	X	N/A	X	X	N/A
IS ^d I	X	X	X	Read/I	X	X	N/A	X	X	N/A
IM ^d S	X	X	X	Write/ MS^{wb}	X	X	N/A	X	X	N/A
SM^w	X	X	X	N/A	Store/ M	X	N/A	I	I	X
MI^{wb}	Hit	Hit	N/A	X	X	Send data/I	N/A	N/A	X	X
MS^{wb}	Hit	Hit	MI^{wb}	X	X	Send data/S	N/A	MI^{wb}	X	X

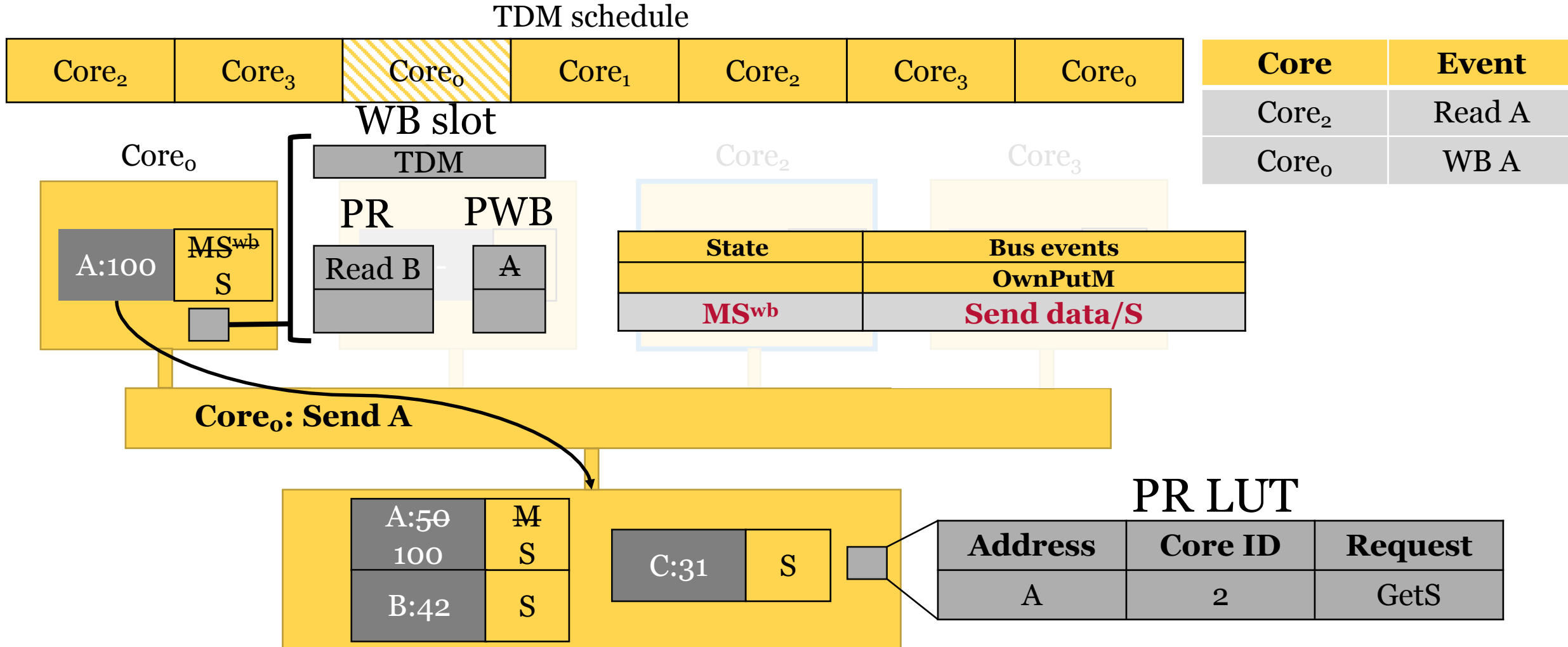
PMSI cache coherence protocol: Protocol modifications

State	Core events			Bus events						
	Load	Store	Replacement	OwnData	OwnUpg	OwnPutM	OtherGetS	OtherGetM	Other Upg	Other PutM
I	Issue GetS/IS ^d	Issue GetM/IM ^d	X	X	X	X	N/A	N/A	N/A	N/A
S	Hit	Issue Upg/ SM^w	X	N/A	X	X	N/A	I	I	X
M	Hit	Hit	Issue PutM/ MI^{wb}	X	X	X	Issue PutM/ MS^{wb}	Issue PutM/ MI^{wb}	X	X
IS ^d	X	X	X	Read/S	X	X	N/A	IS ^d I	IS ^d I	N/A
IM ^d	X	X	X	Write/S	X	X	IM ^d S	IM ^d I	X	N/A
IM ^d I	X	X	X	Write/ MI^{wb}	X	X	N/A	X	X	N/A
IS ^d I	X	X	X	Read/I	X	X	N/A	X	X	N/A
IM ^d S	X	X	X	Write/ MS^{wb}	X	X	N/A	X	X	N/A
SM^w	X	X	X	N/A	Store/ M	X	N/A	I	I	X
MI^{wb}	Hit	Hit	N/A	X	X	Send data/I	N/A	N/A	X	X
MS^{wb}	Hit	Hit	MI^{wb}	X	X	Send data/S	N/A	MI^{wb}	X	X

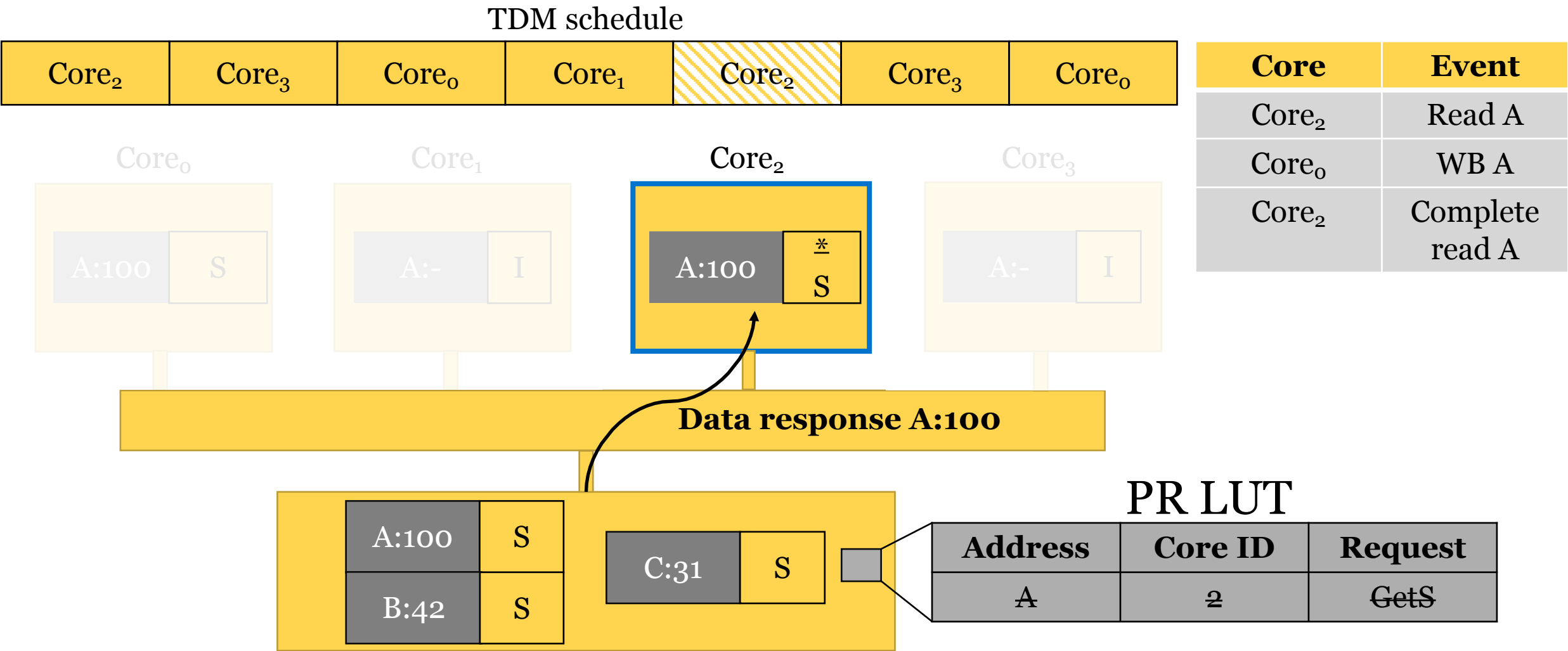
Intra-core coherence interference



Intra-core coherence interference

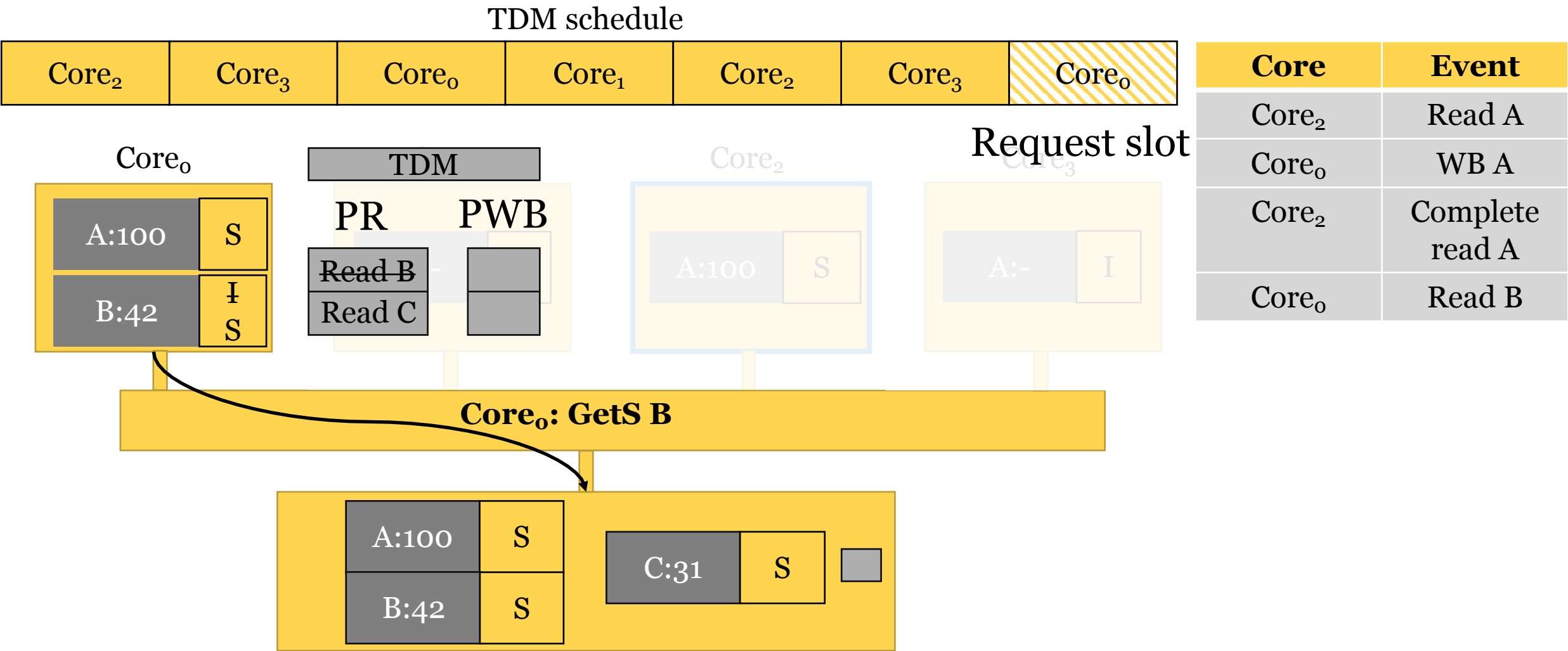


Intra-core coherence interference



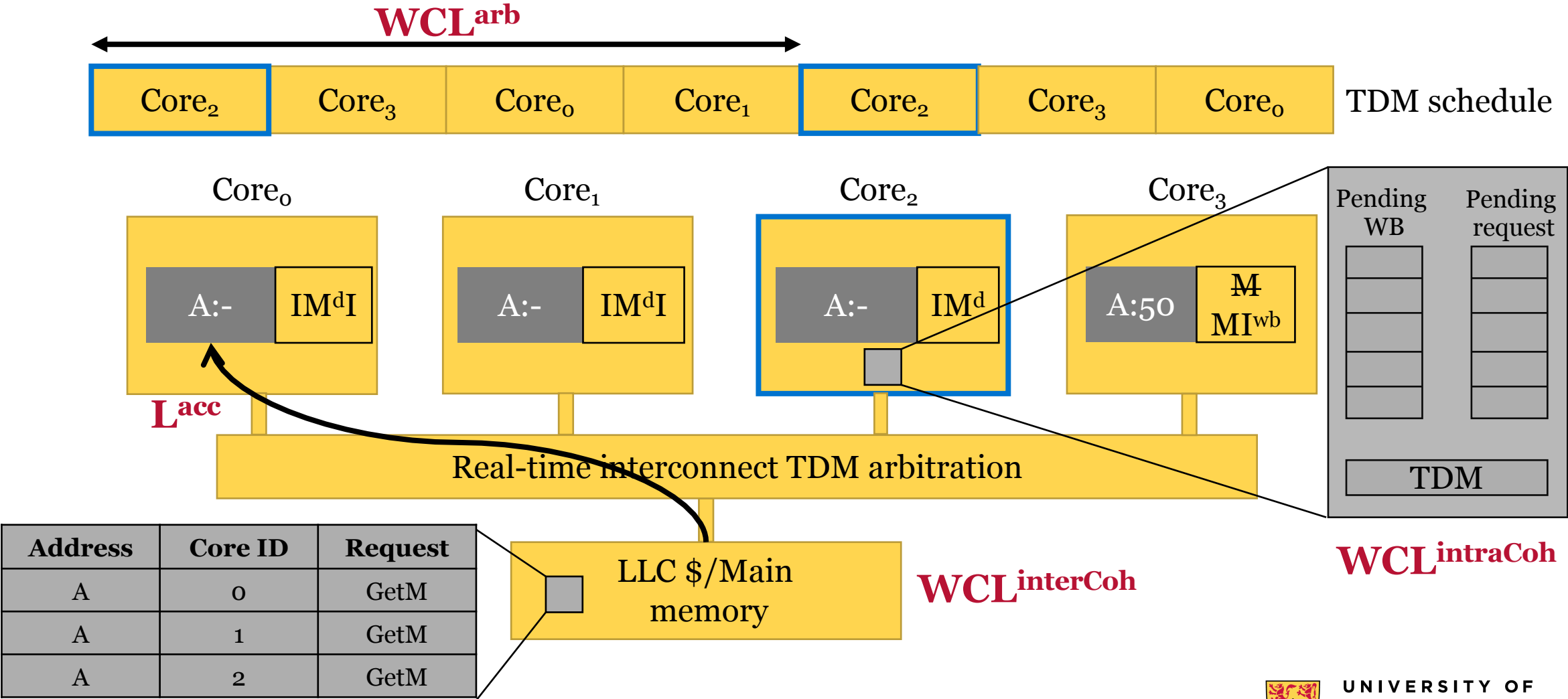
Core	Event
Core ₂	Read A
Core ₀	WB A
Core ₂	Complete read A

Intra-core coherence interference

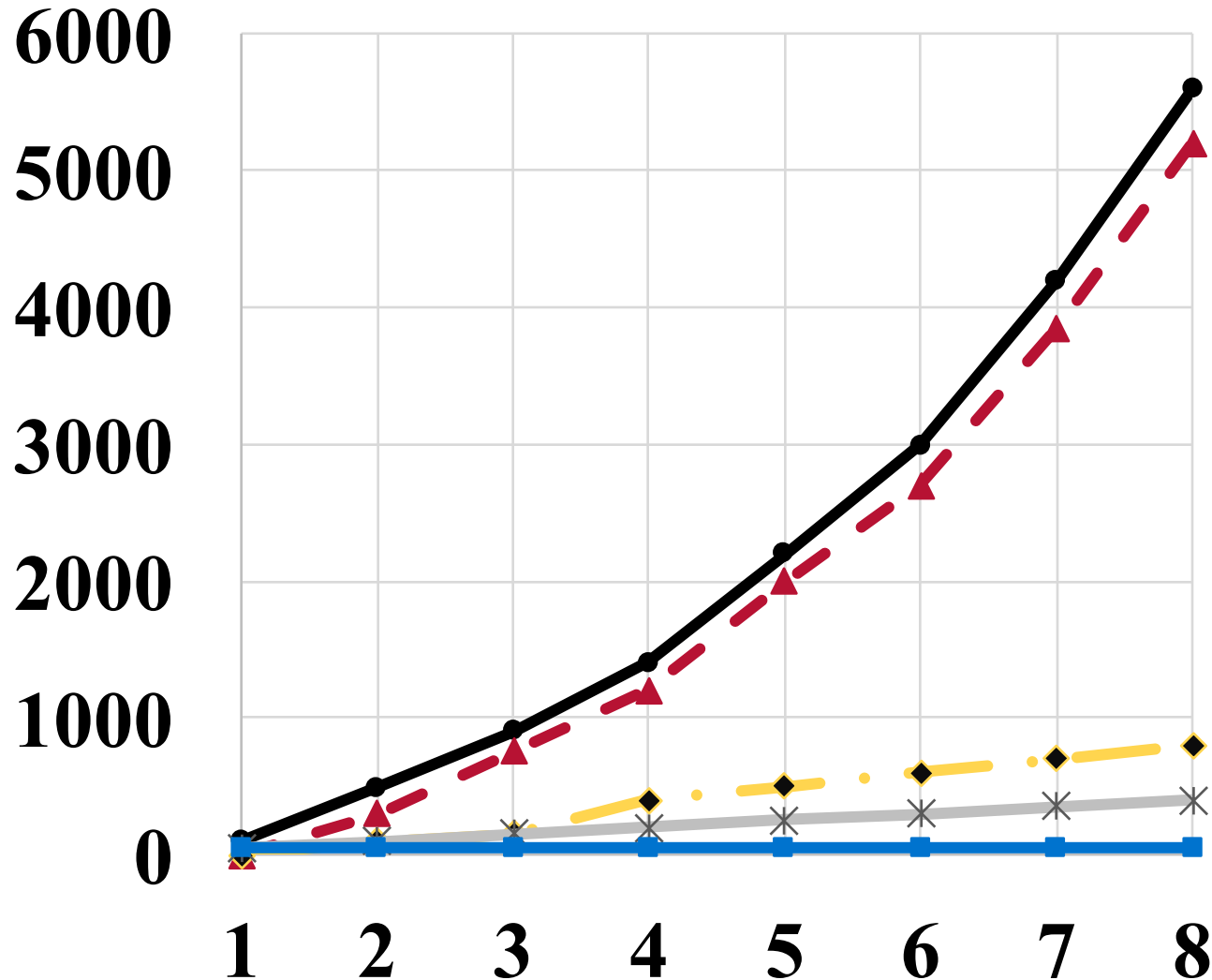


Latency analysis of PMSI

N: Number of cores



Latency analysis of PMSI

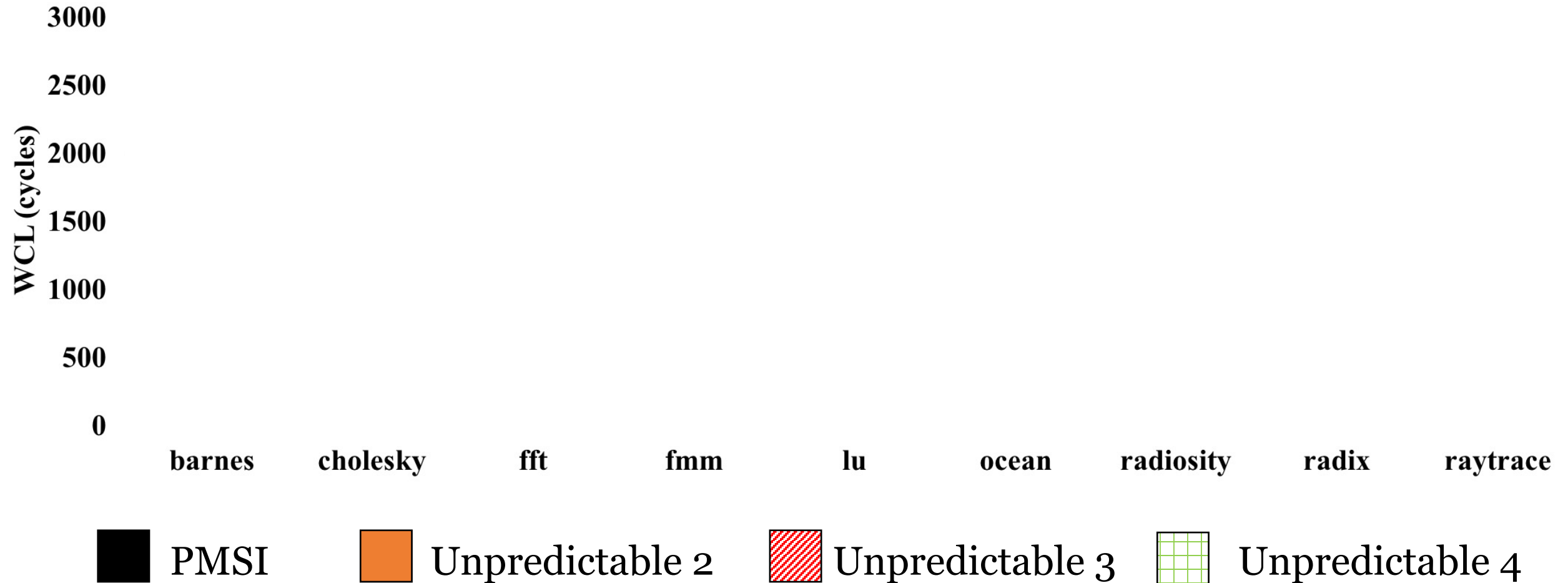


$$\begin{aligned} \text{WCL of memory request} &= \\ &\quad L^{\text{acc}} + \\ &\quad \text{WCL}^{\text{arb}} (\propto N) + \\ &\quad \text{WCL}^{\text{intraCoh}} (\propto N) + \\ &\quad \text{WCL}^{\text{interCoh}} (\propto N^2) \end{aligned}$$

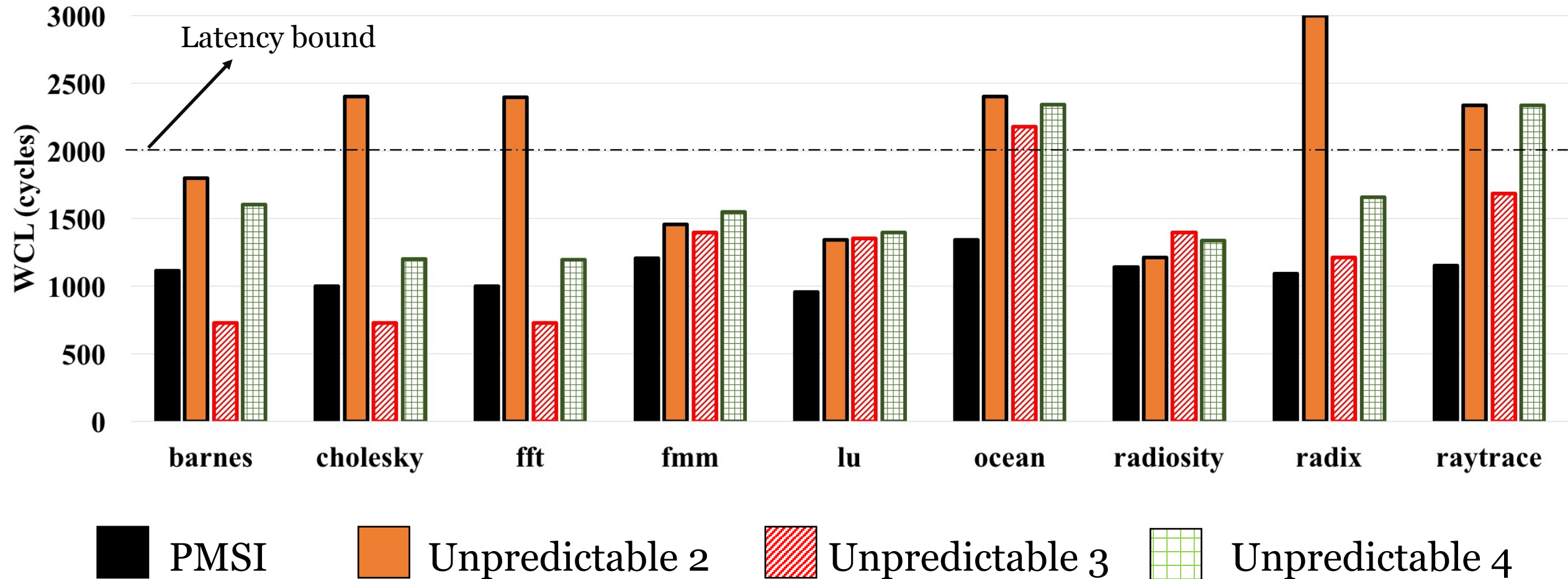
Results

- Cycle accurate Gem5 simulator
- 2, 4, 8 cores, 16KB direct mapped private L1-D/I caches
- In-order cores, 2GHz operating frequency
- TDM bus arbitration with slot width = 50 cycles
- Benchmarks: SPLASH-2, synthetic benchmarks to stress worst-case scenarios
- Simulation framework available at <https://git.uwaterloo.ca/caesr-pub/pmsi>

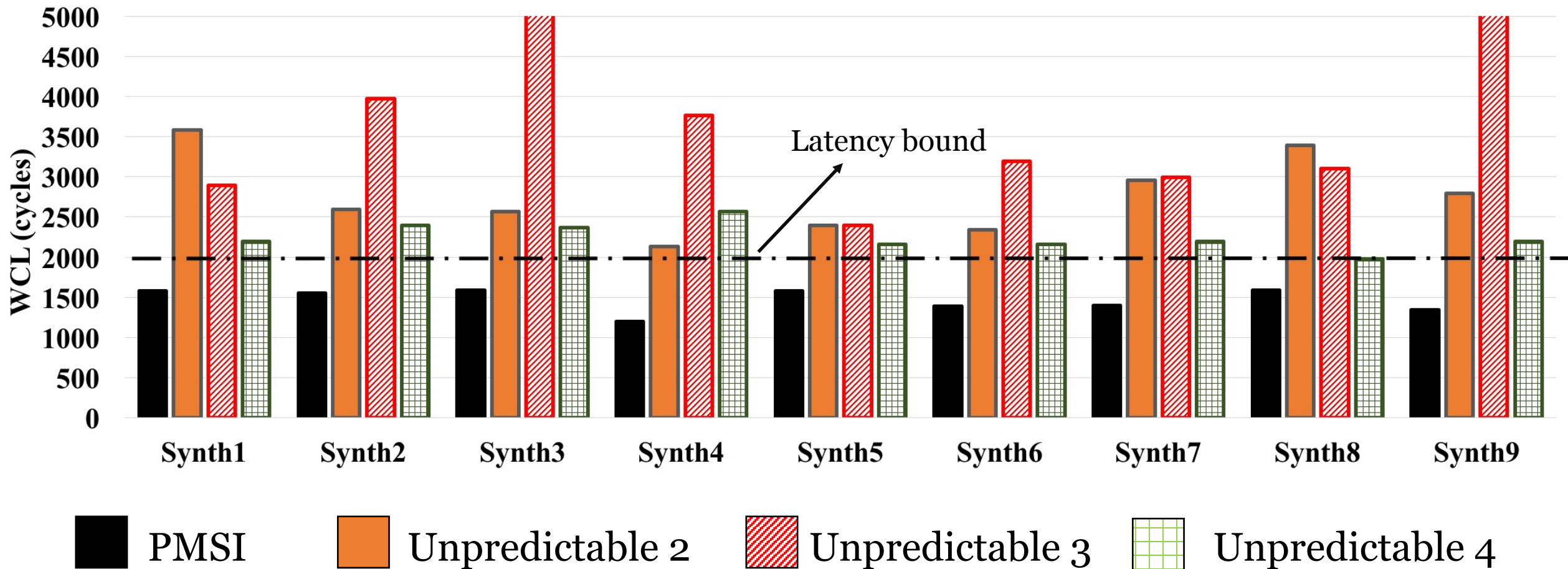
Results: Observed worst case latency per request



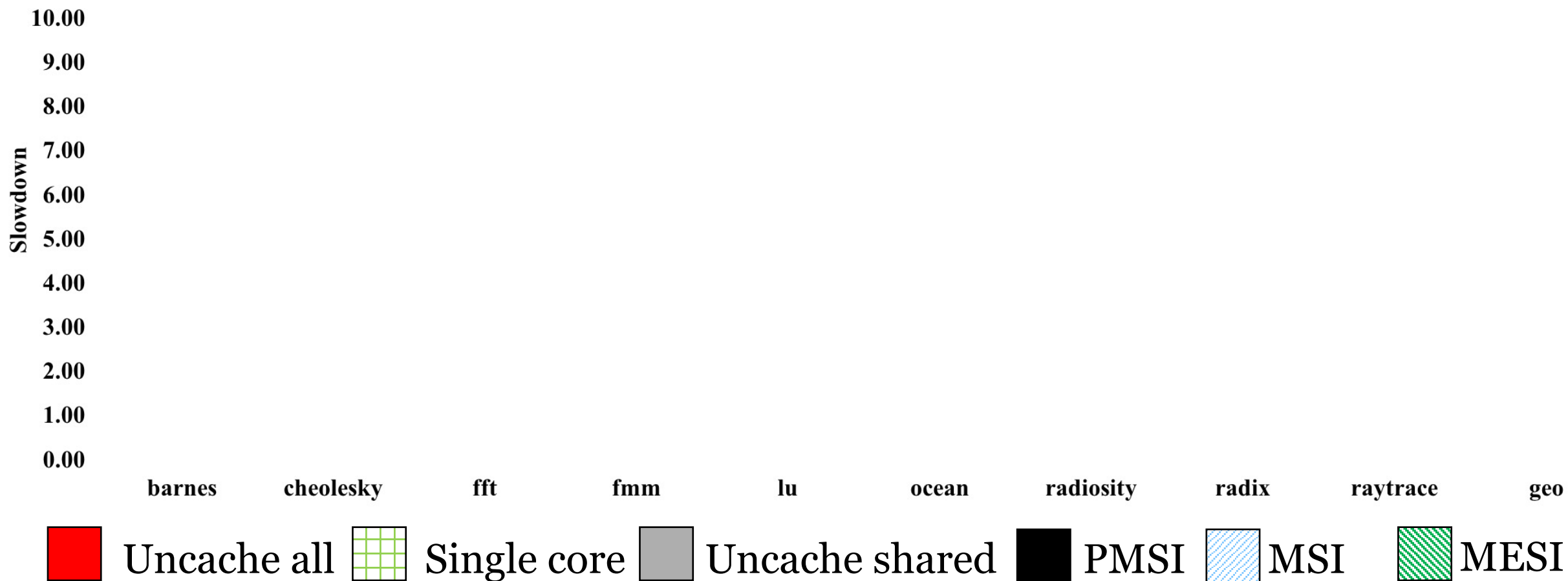
Results: Observed worst case latency per request



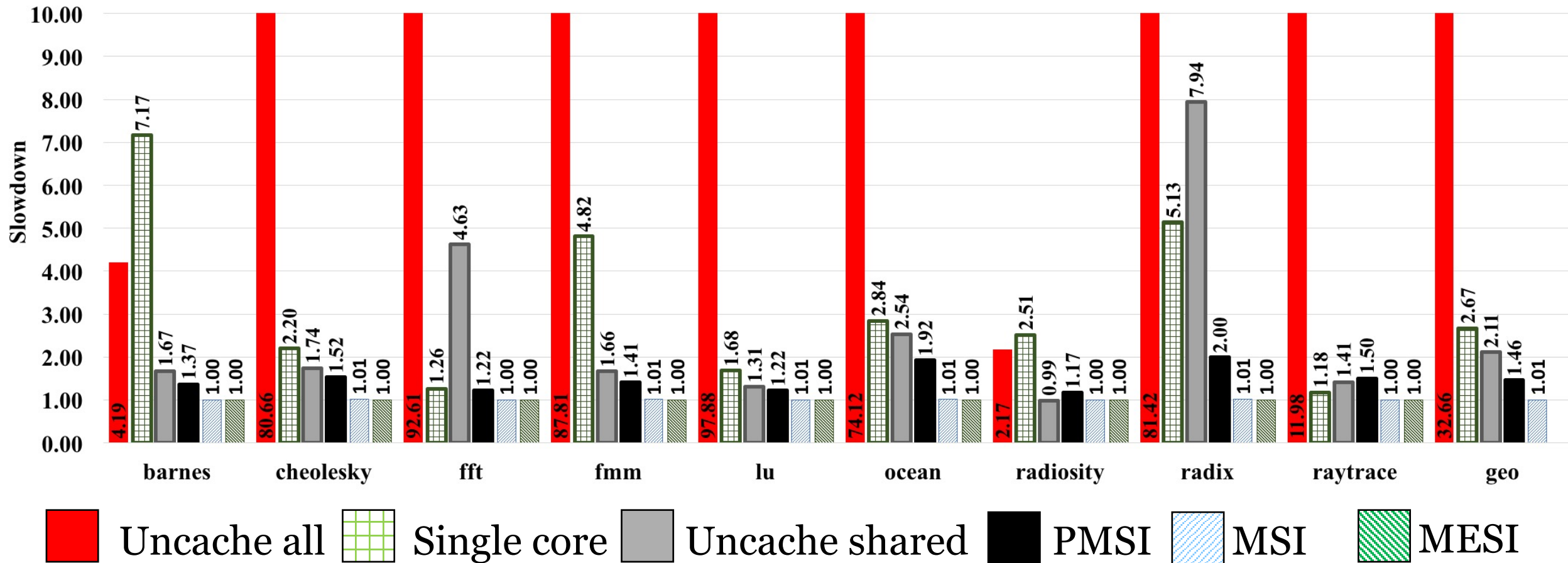
Results: Observed worst case latency per request



Results: Slowdown relative to MESI cache coherence



Results: Slowdown relative to MESI cache coherence



Conclusion

- Enumerate sources of unpredictability when applying conventional hardware cache coherence on multi-core real-time systems
- Propose PMSI cache coherence that allows **predictable simultaneous shared data accesses with no changes to application/OS**
- Hardware overhead of PMSI is ~ **128 bytes**
- WCL of memory access with PMSI $\propto$ square of number of cores ($\propto N^2$)
- **PMSI outperforms prior techniques for handling shared data accesses (upto 4x improvement over uncached-shared), and suffers on average 46% slowdown compared to conventional MESI cache coherence protocol**

UNIVERSITY OF **WATERLOO**



THANK YOU & QUESTIONS?