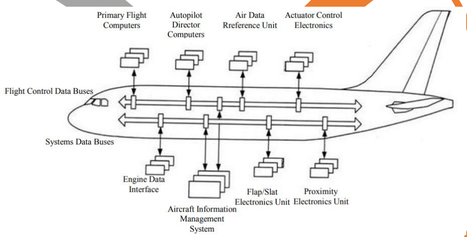
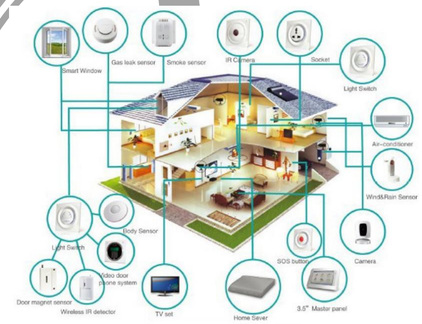
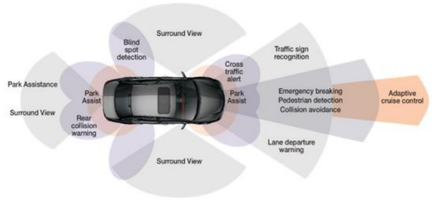
The Best of All Worlds: Improving Predictability at the Performance of Conventional Coherence with No Protocol Modifications

Salah Hessien, **Mohamed Hassan**





Data is a Key in all modern applications



MOTIVATION

Ignore

- Adopts an independent-task model → No communication amongst tasks

Prevent

- Enforcing complete isolation between tasks.
  - At the shared cache: strict cache partitioning and coloring
  - At the DRAM: bank privatization

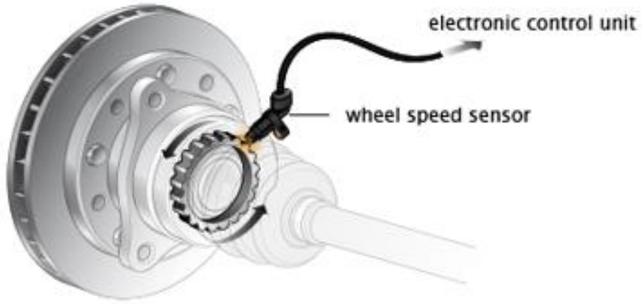
Common Approach

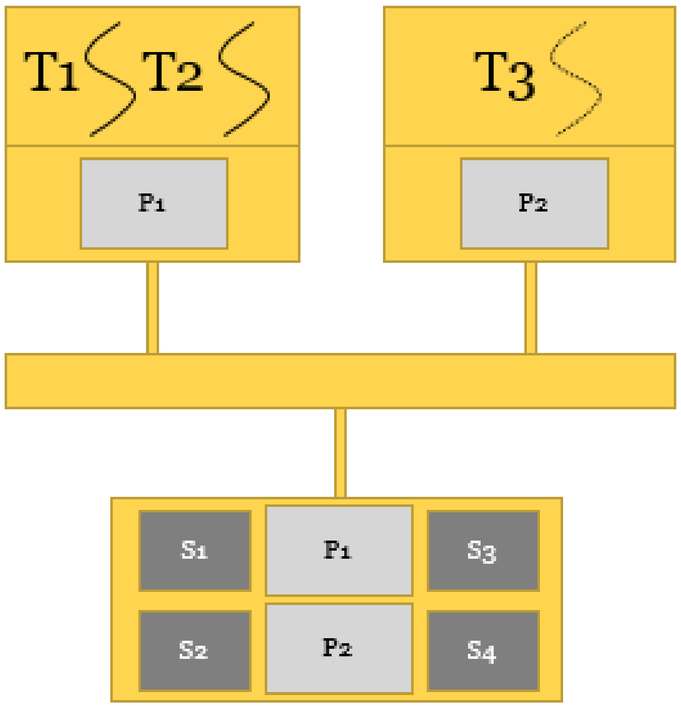
Data Sharing

Common Approach

Data Sharing

- May result in a poor memory or cache utilization
  - e.g.: a task has conflict misses, while other partitions may remain underutilized
- Does not scale with increasing number of cores
  - e.g.: number of PEs ≤ number of DRAM banks
- Not viable in emerging systems due to increased functionality and massive data



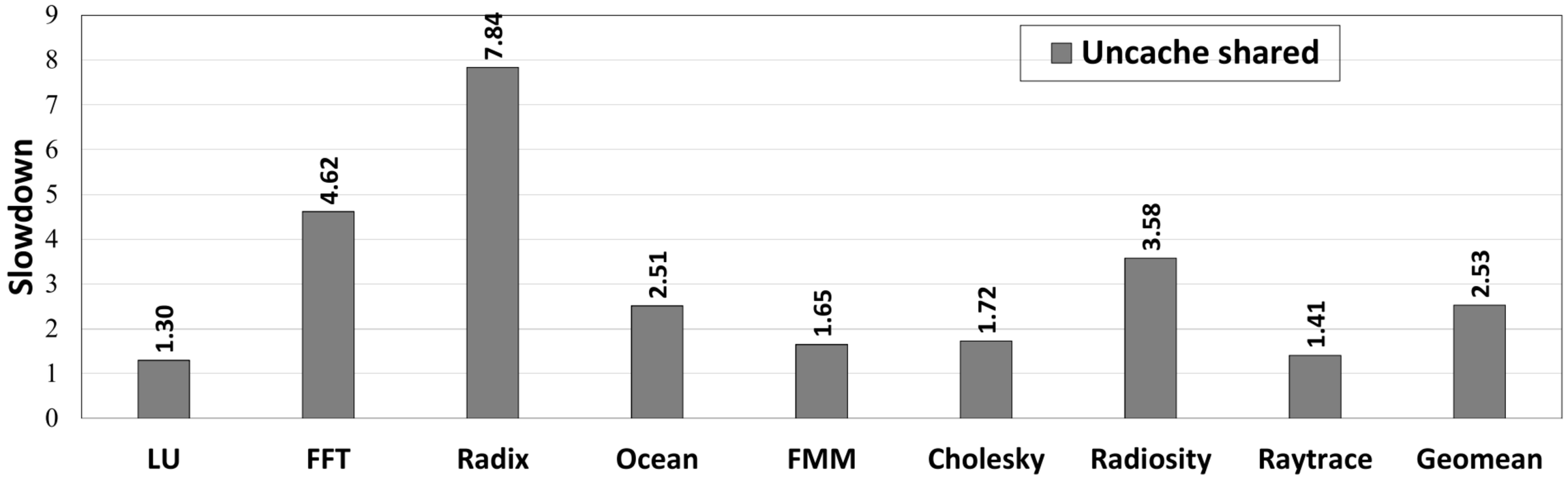


✓ Simpler timing analysis
✗ Hardware changes
✗ Long execution time

Solution:

No caching of shared data

[Hardy et al., RTSS'09]
[Lesage et al., RTNS'10]
[Bansal et al., arXiv'19]
[Chisholm et al., RTSS'16]

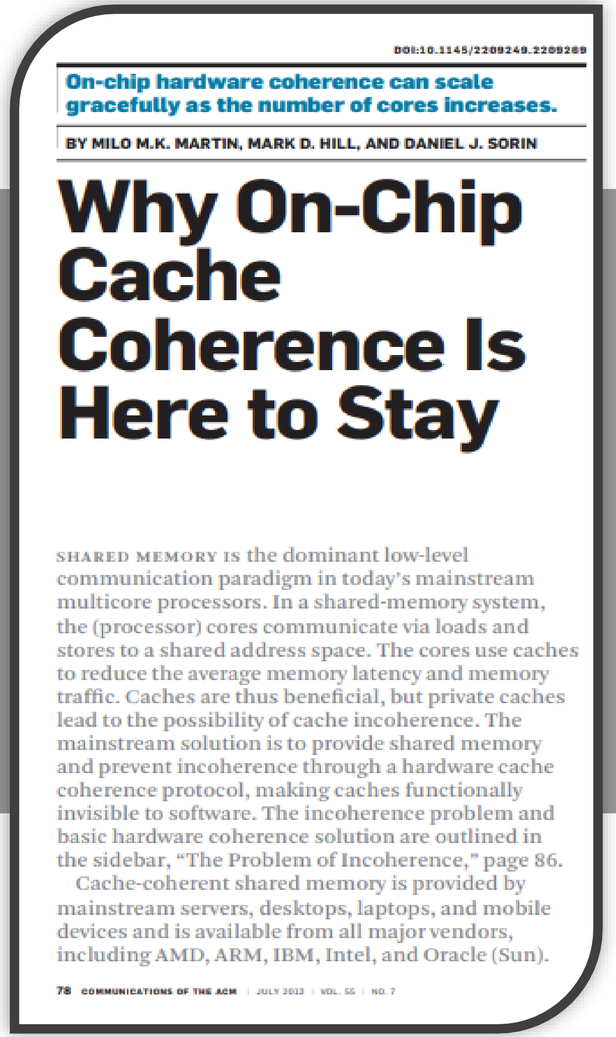


Solution:

No caching of shared data

[Hardy et al., RTSS'09]
[Lesage et al., RTNS'10]
[Bansal et al., arXiv'19]
[Chisholm et al., RTSS'16]

The mainstream solution is to provide shared memory and prevent incoherence through a hardware cache coherence protocol, making caches functionally invisible to software.



DOI:10.1145/2209249.2209269

On-chip hardware coherence can scale gracefully as the number of cores increases.

BY MILO M.K. MARTIN, MARK D. HILL, AND DANIEL J. SORIN

Why On-Chip Cache Coherence Is Here to Stay

SHARED MEMORY IS the dominant low-level communication paradigm in today's mainstream multicore processors. In a shared-memory system, the (processor) cores communicate via loads and stores to a shared address space. The cores use caches to reduce the average memory latency and memory traffic. Caches are thus beneficial, but private caches lead to the possibility of cache incoherence. The mainstream solution is to provide shared memory and prevent incoherence through a hardware cache coherence protocol, making caches functionally invisible to software. The incoherence problem and basic hardware coherence solution are outlined in the sidebar, "The Problem of Incoherence," page 86.

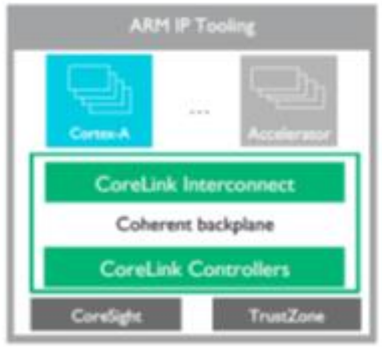
Cache-coherent shared memory is provided by mainstream servers, desktops, laptops, and mobile devices and is available from all major vendors, including AMD, ARM, IBM, Intel, and Oracle (Sun).

Coherence is the norm in COTS platforms

Data Sharing

Heterogeneous compute requires coherency

- Flexible heterogeneous architecture
  - Blend compute and acceleration for target solution
- Fast, reliable transport to shared memory
  - Maximize throughput, minimize latency
- Accelerate SoC deployment
  - IP designed, optimized and validated for systems



ARM

Coherence is the Industry's Choice

Data Sharing

oday's SoCs include a mix of CPU cores, computing clusters, GPUs and other computing resources and specialized accelerators.
Getting heterogeneous processors to communicate efficiently is a daunting design challenge. ***A popular approach is to use high-performance and power-efficient shared-memory communication and a sophisticated on-chip cache-coherent interconnect.*** This presentation will introduce a new technology that automates the architecture design process, supports CHI and ACE in one design, and uses advanced machine-learning algorithms to create an optimal pre-verified cache-coherent solution.

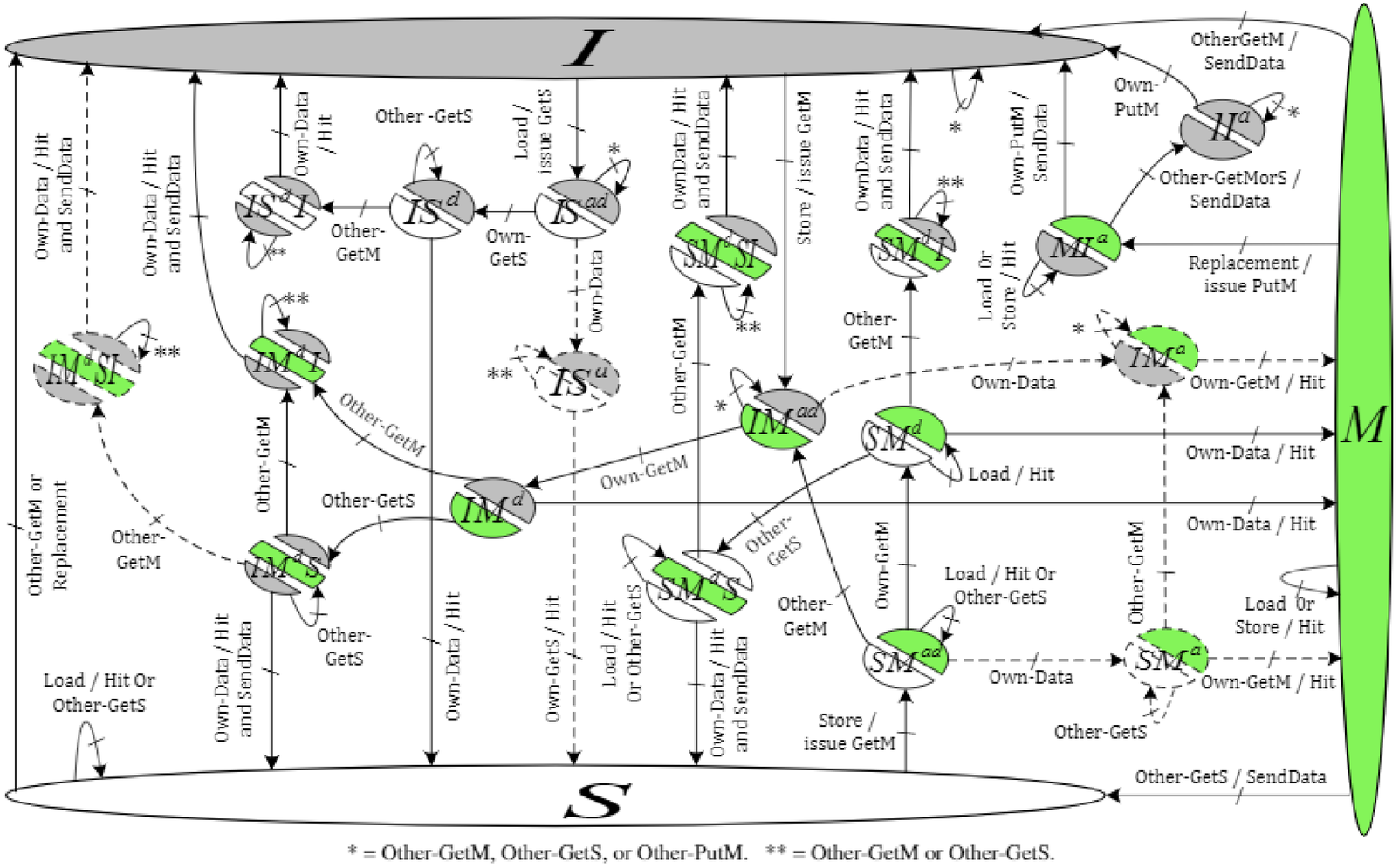
Coherence is the Industry's Choice

Data Sharing

Autonomous driving requirements are mandating the simultaneous use of multiple types of processing units to efficiently execute sophisticated image processing, sensor fusion, and machine learning/AI algorithms.
This presentation introduces ***new coherency platform technology that enables the integration of heterogeneous cache coherent hardware accelerators and CPUs***, using a mixture of ARM ACE, CHI, and CHI Issue B protocols, into systems that meet both the requirements of high compute performance and ISO 26262-compliant functional safety.

Coherence is the Industry's Choice

Data Sharing

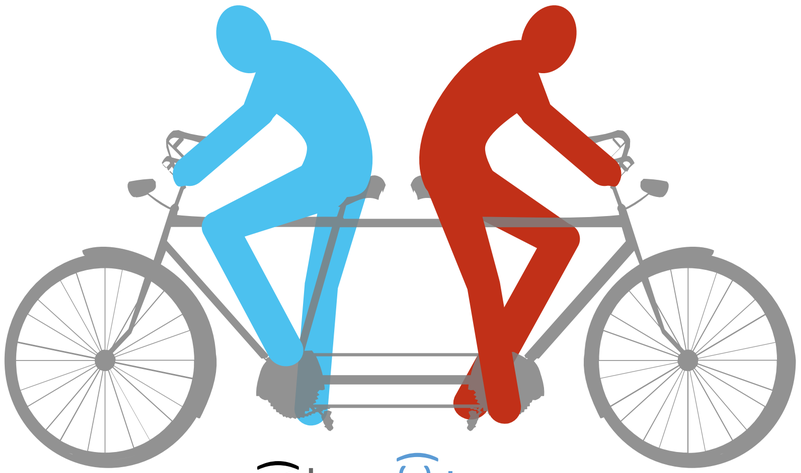


Coherence is A complex Machinery

Data Sharing

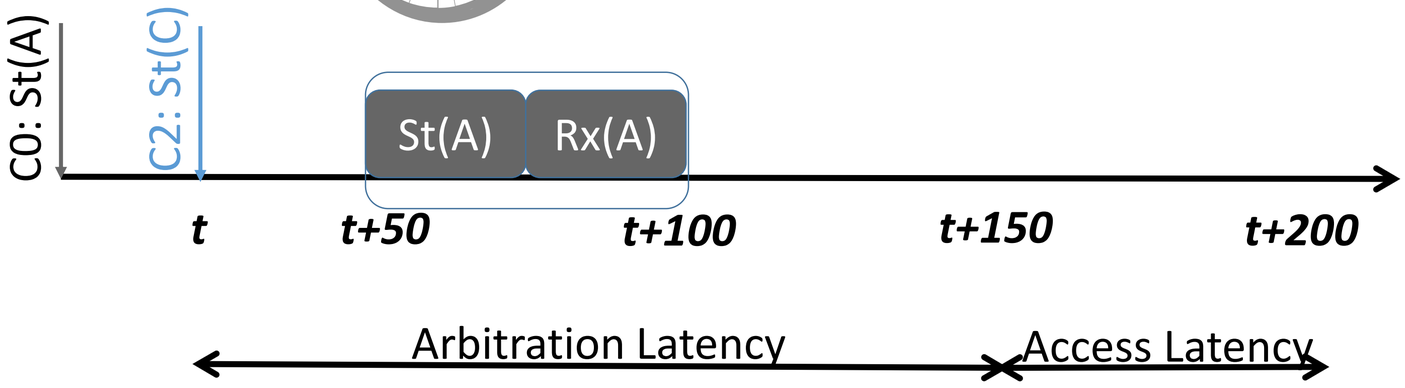
Conventional COTS:

- High performance arbitration
  - FCFS, Split-Tx, Priority-Based, ..
- Shared data is entirely supported through a hardware cache coherence protocol
- Not Predictable



Traditional Real-Time:

- Predictable by design
  - TDM, RR, WRR, HRR, …
- Does not support shared data, put aside coherence

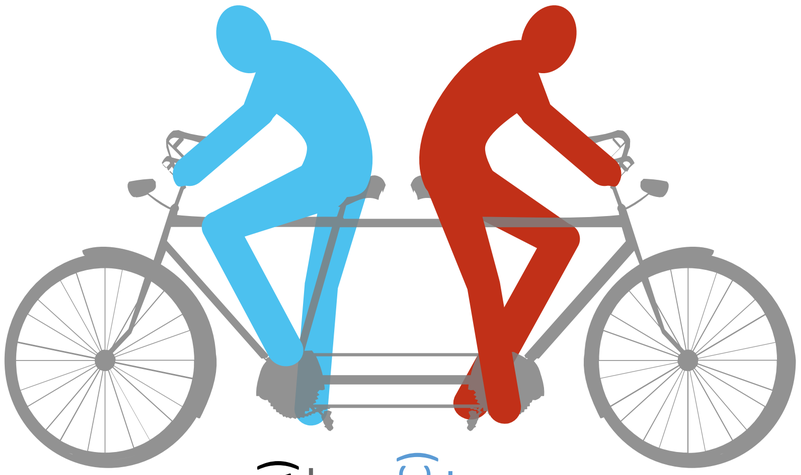


State-of-the-art

Data Sharing

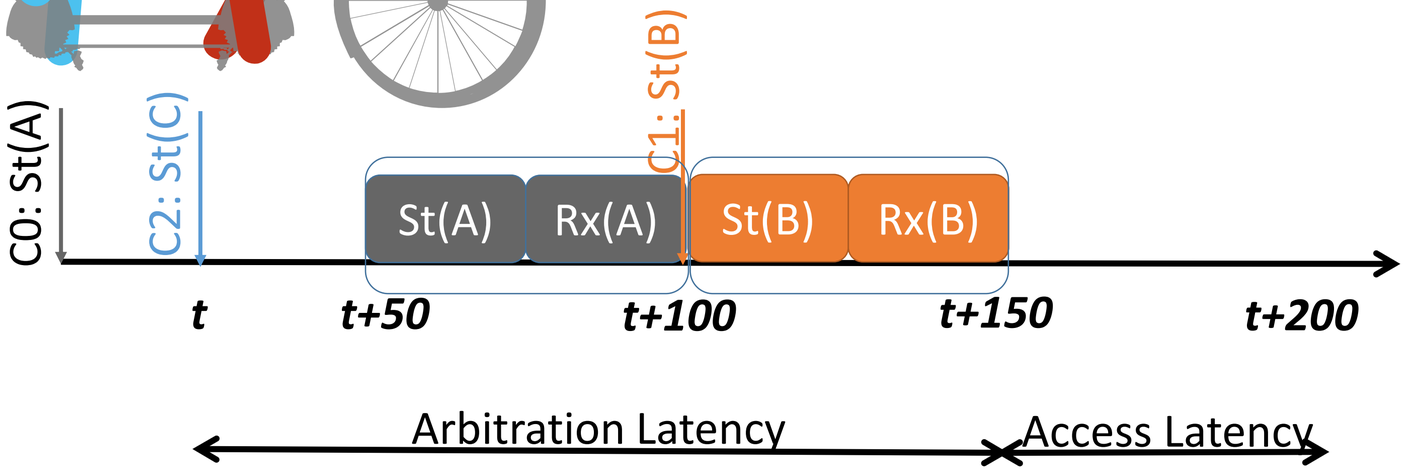
Conventional COTS:

- High performance arbitration
  - FCFS, Split-Tx, Priority-Based, ..
- Shared data is entirely supported through a hardware cache coherence protocol
- Not Predictable



Traditional Real-Time:

- Predictable by design
  - TDM, RR, WRR, HRR, …
- Does not support shared data, put aside coherence

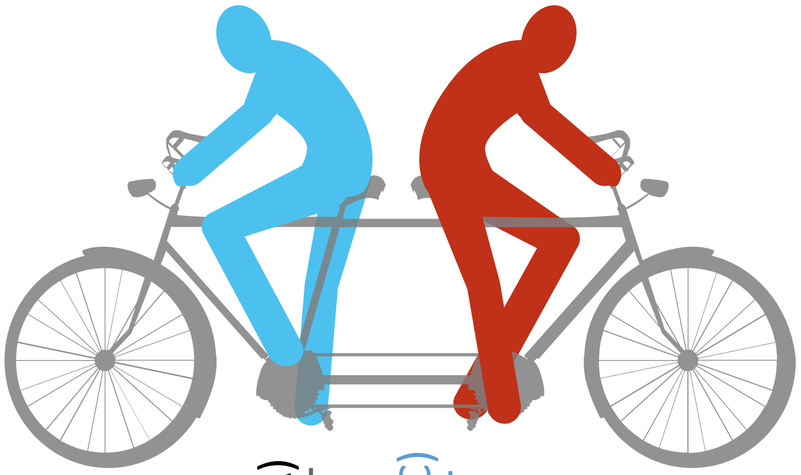


State-of-the-art

Data Sharing

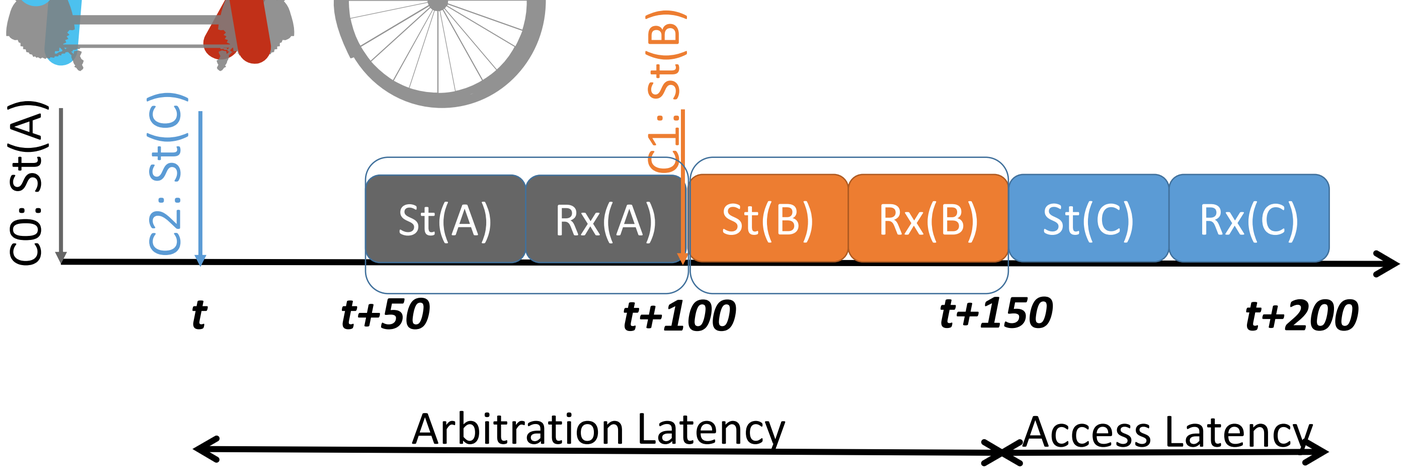
Conventional COTS:

- High performance arbitration ✓
  - FCFS, Split-Tx, Priority-Based, ..
- Shared data is entirely supported through a hardware cache coherence protocol ✓✓
- Not Predictable ✗



Traditional Real-Time:

- Predictable by design ✓✓
  - TDM, RR, WRR, HRR, …
- Does not support shared data, put aside coherence ✗

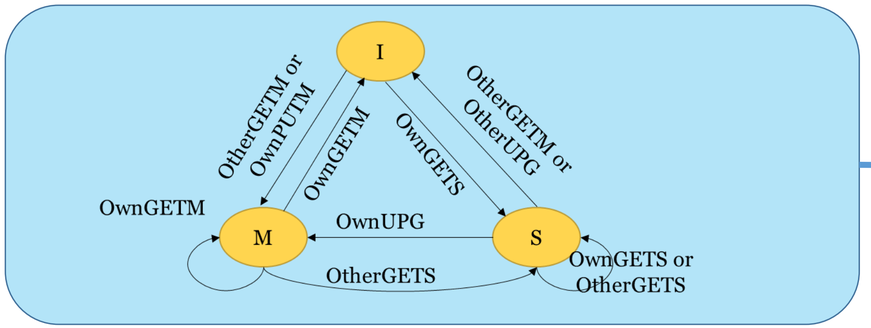


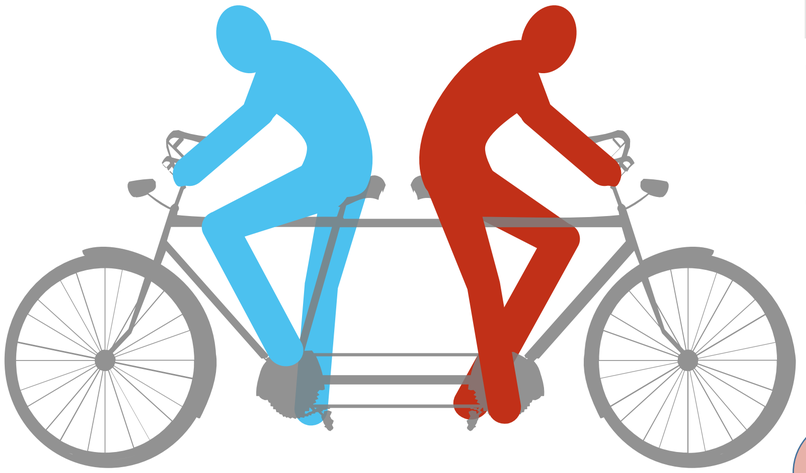
State-of-the-art

Data Sharing

Conventional COTS:

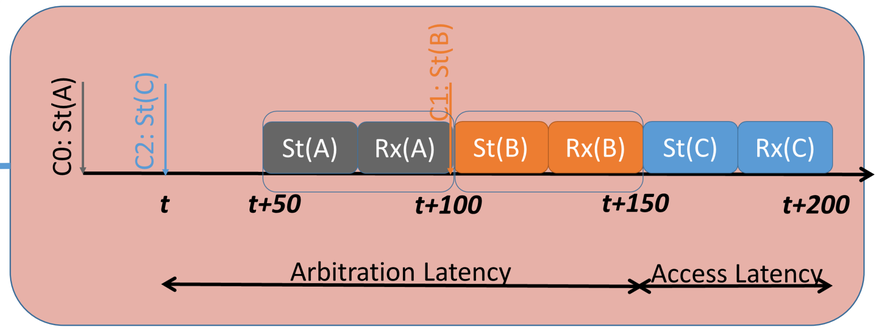
- High performance arbitration
  - FCFS, Split-Tx, Priority-Based, ..
- Shared data is entirely supported through a hardware cache coherence protocol
- Not Predictable





Traditional Real-Time:

- Predictable by design
  - TDM, RR, WRR, HRR, …
- Does not support shared data, put aside coherence



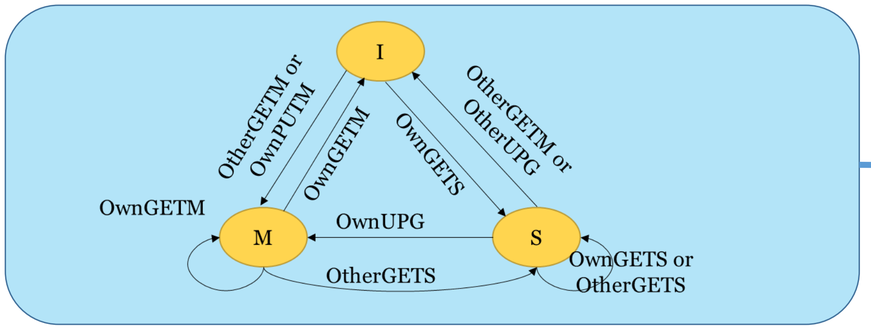
+

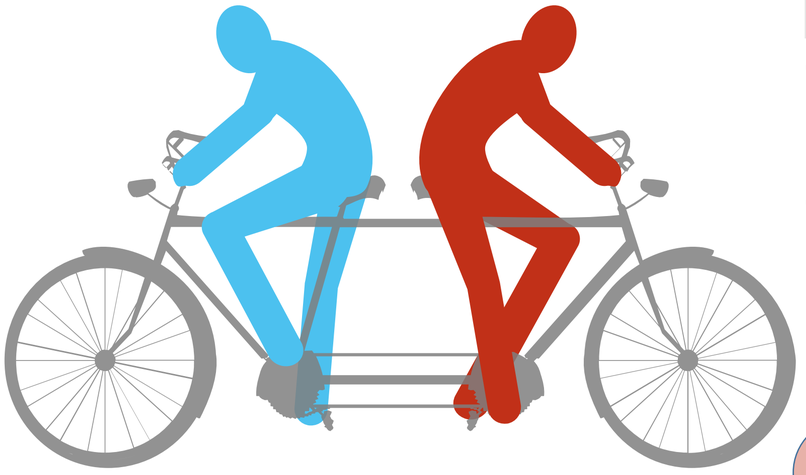
State-of-the-art

Data Sharing

Conventional COTS:

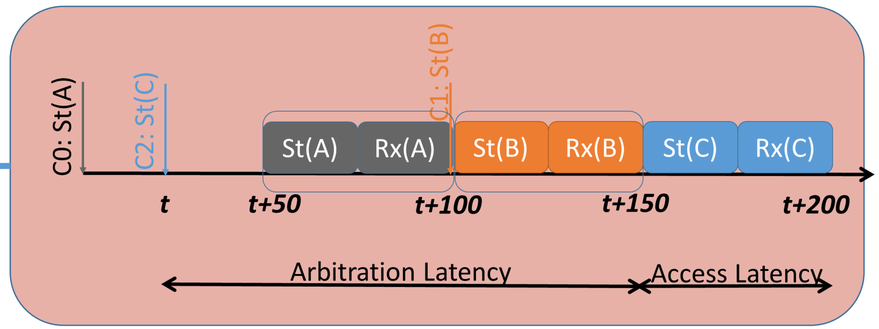
- High performance arbitration ✓
  - FCFS, Split-Tx, Priority-Based, ..
- Shared data is entirely supported through a hardware cache coherence protocol ✓✓
- Not Predictable ✗





Traditional Real-Time:

- Predictable by design ✓✓
  - TDM, RR, WRR, HRR, …
- Does not support shared data, put aside coherence ✗



+

Unpredictable

[RTAS'17] Mohamed Hassan, Anirudh M. Kaushik, Hiren Patel, "Predictable Cache Coherence for Multi-Core Real-Time Systems"

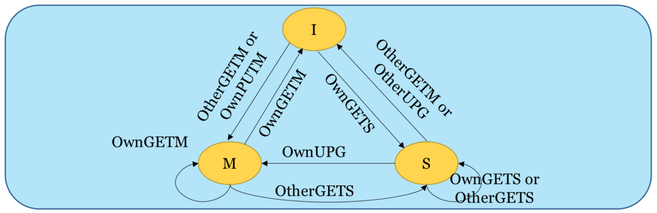
State-of-the-art

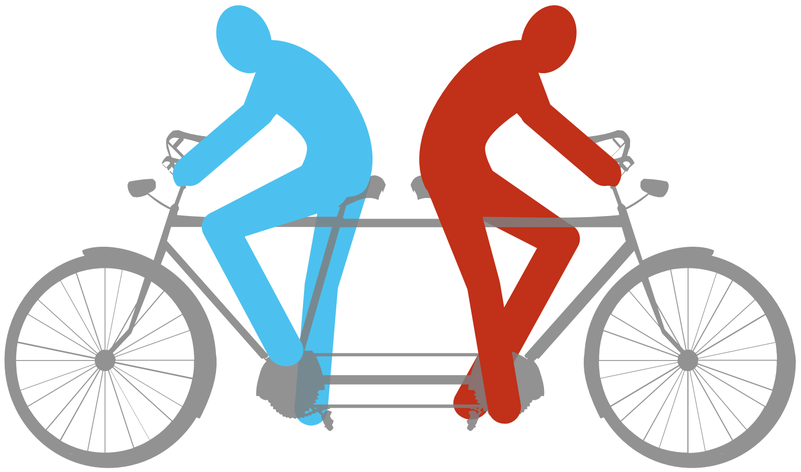
Data Sharing

State-of-the-art

Conventional COTS:

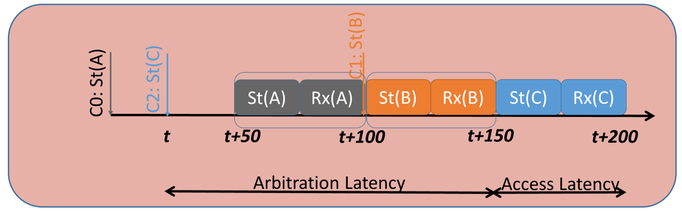
- High performance arbitration ✓
  - FCFS, Split-Tx, Priority-Based, ..
- Shared data is entirely supported through a hardware cache coherence protocol ✓✓
- Not Predictable ✗





Traditional Real-Time:

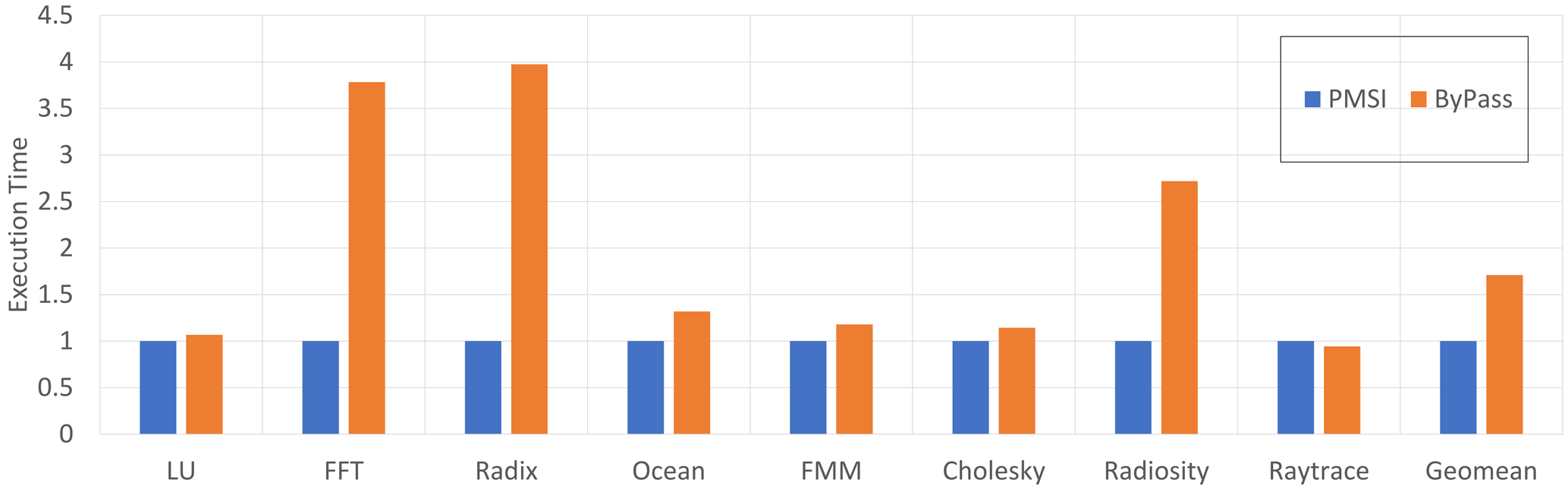
- Predictable by design ✓✓
  - TDM, RR, WRR, HRR, …
- Does not support shared data, put aside coherence ✗



Data-Aware Arbitration:

- Builds on traditional arbitration
- Shared data is supported ✓✓
- BUT requires coherence protocol modifications ✓
- Coherence leads to good performance
- Also achives predictability, but with significant latency bounds ✓
- PMSI [RTAS'17], CARP [RTSS'19], HourGlass [Arxiv'18], PENDULUM [RTSS'19]

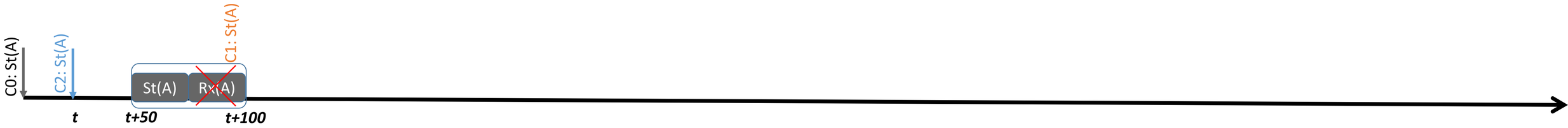
Predictable Coherence



[RTAS'17] Mohamed Hassan, Anirudh M. Kaushik, Hiren Patel, "Predictable Cache Coherence for Multi-Core Real-Time Systems"

Benefit of Coherence: Up to 3x performance

Predictable Coherence

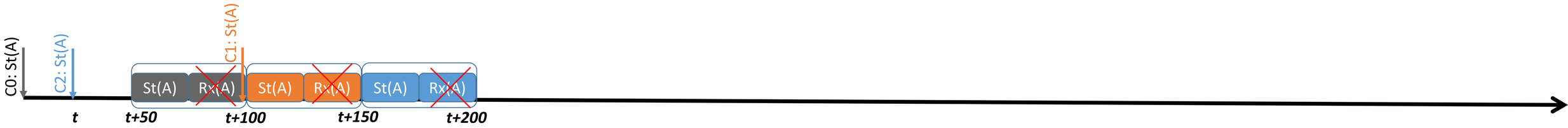


C0 cannot receive A from Mem,
it has to wait for C1 to WB first

- Assume same example from before
- But, let all 3 cores request to access same cache line A
- Assume initially A is modified by C1 in its private cache
- Let as before, our request under analysis is St(A) from C2

Problem: Coherence effect on WC

Predictable Coherence

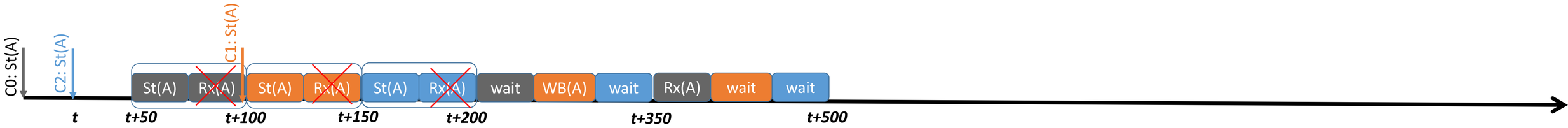


C0 cannot receive A from Mem, it has to wait for C1 to WB first

C1 and C2 has to wait for C0 to conform to the coherence order

Problem: Coherence effect on WC

Predictable Coherence

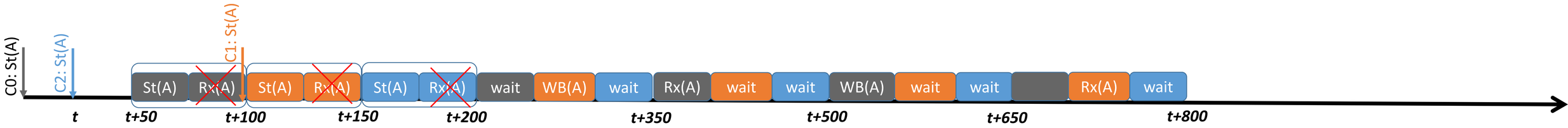


C1 performs the WB

Only then, C0 receives data and finishes its request

Problem: Coherence effect on WC

Predictable Coherence

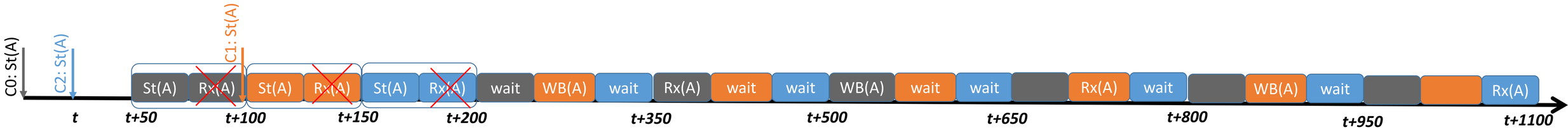


In next period, C0 performs the WB

Only then, C1 receives data and finishes its request

Problem: Coherence effect on WC

Predictable Coherence

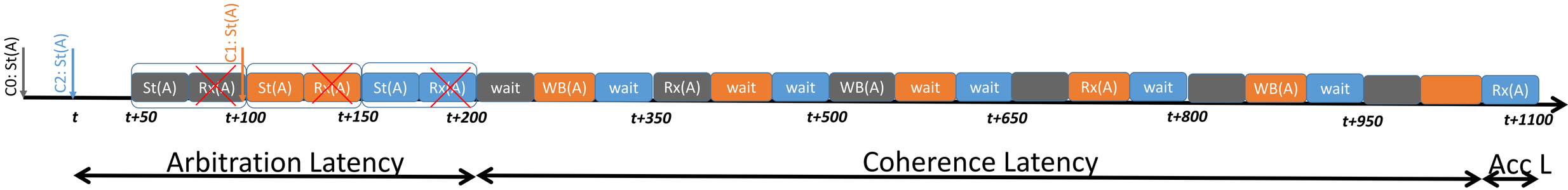


Finally, C1 performs the WB

Afterwards, C2 receives data and finishes its request

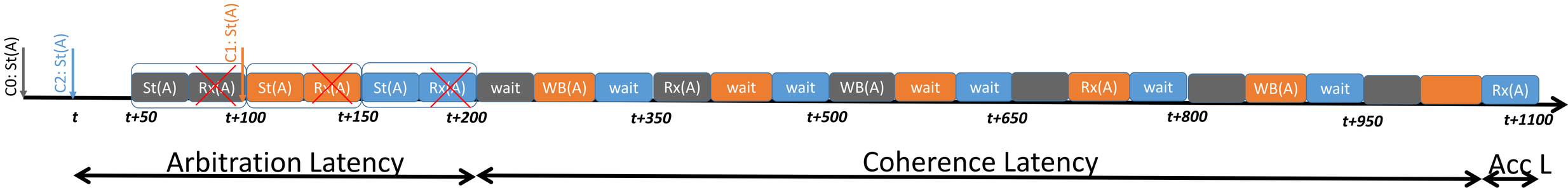
Problem: Coherence effect on WC

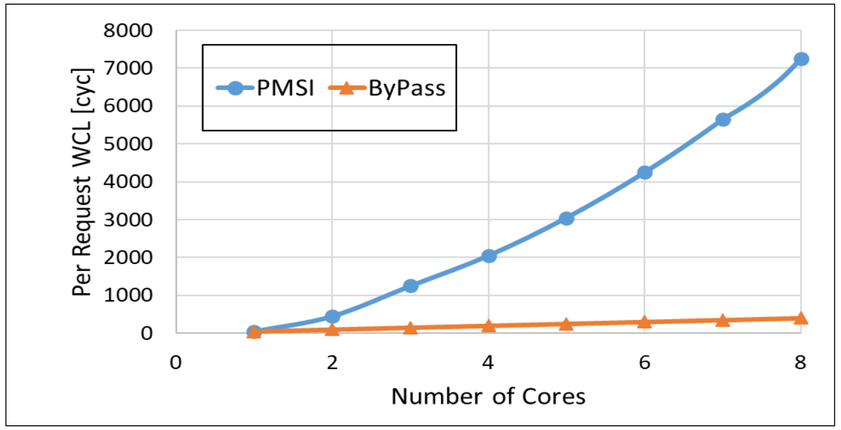
Predictable Coherence



Problem: Coherence effect on WC

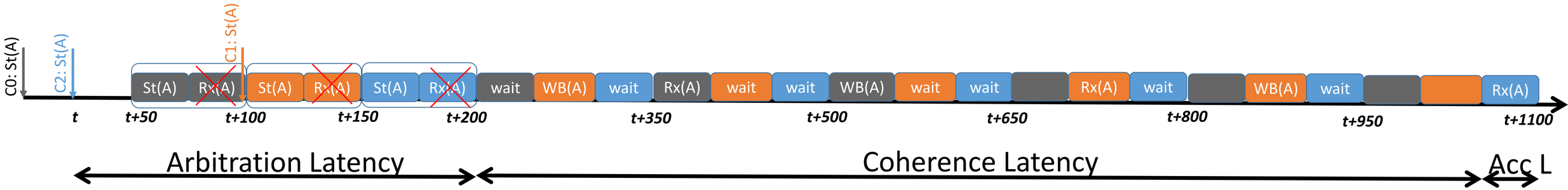
Predictable Coherence



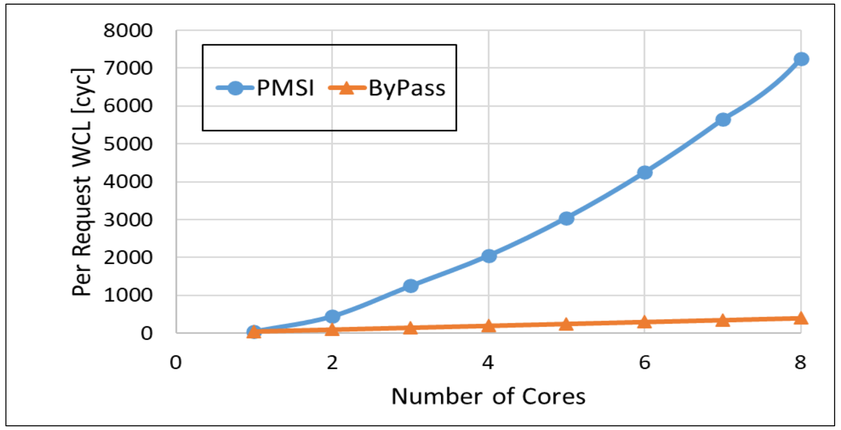


Problem: Coherence effect on WC

Predictable Coherence



- Adopting Traditional Arbitration for coherence results in coupling coherence requests with responses
  - This results in **huge WCLs** of PMSI
  - It also requires the **coherence modifications** to conform to this arbitration, while maintaining predictable behavior
- *Can we achieve predictable coherence without imposing coherence modifications?*
  - **Yes** → by deploying a predictable split-Tx bus

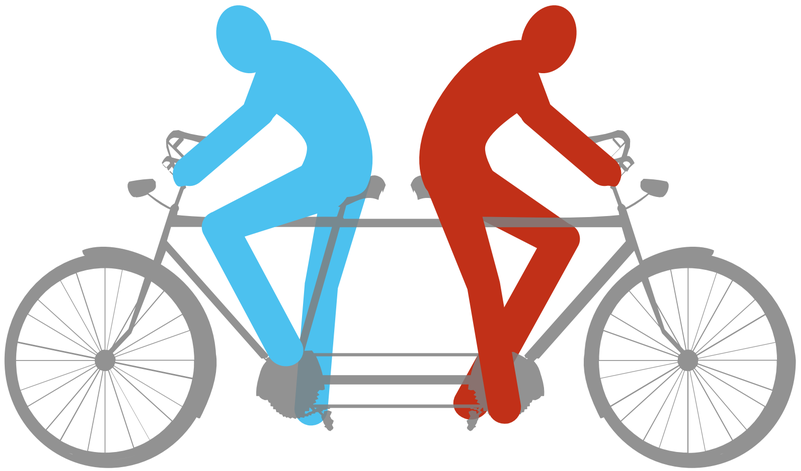


PISCOT Main Idea

Predictable Coherence

Conventional COTS:

- High performance arbitration
  - FCFS, Split-Tx, Priority-Based, ..
- Shared data is entirely supported through a hardware cache coherence protocol
- Not Predictable



Traditional Real-Time:

- Predictable by design
  - TDM, RR, WRR, HRR, …
- Does not support shared data, put aside coherence

PISCOT

- Predictable split-transaction bus
- Shared data is supported
- With ***NO*** protocol modifications
- Achieves better performance than existing coherence solutions
- Also achives predictability, with tight latency bounds

Data-Aware Arbitration:

- Builds on traditional arbitration
- Shared data is supported
- BUT requires coherence protocol modifications
- Coherence leads to good performance
- Also achives predictability, but with significant latency bounds
- PMSI [RTAS'17], CARP [RTSS'19], HourGlass [Arxiv'18], PENDULUM [RTSS'19]

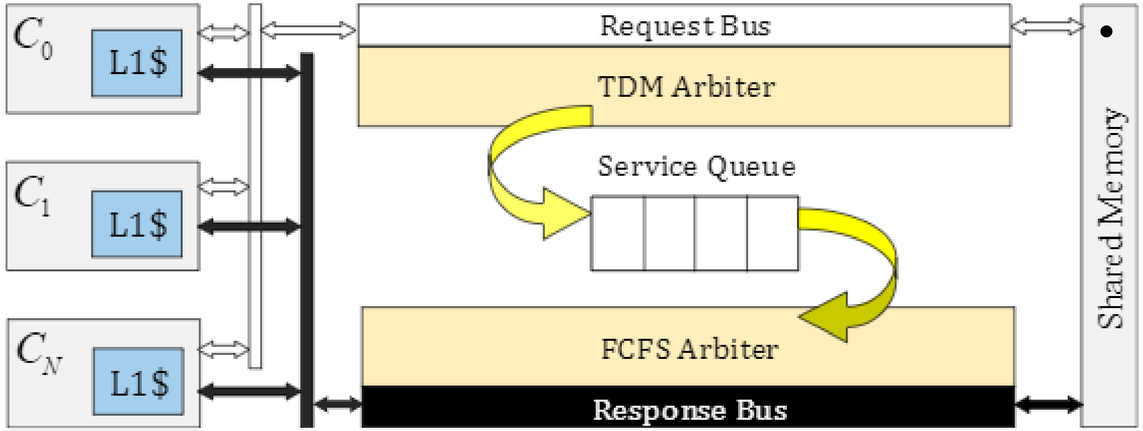
PISCOT vs. State-of-the-art

PISCOT

Unlike COTS systems, the Request Bus Arbiter is a work-conserving TDM arbiter

(1) PISCOT implements a split-transaction bus:

- A Request Bus is responsible for broadcasting the coherence messages initiating memory requests, while
- A Response Bus transfers data as a response to these requests.
- Both buses operate fully in parallel and synchronize using a Service Queue



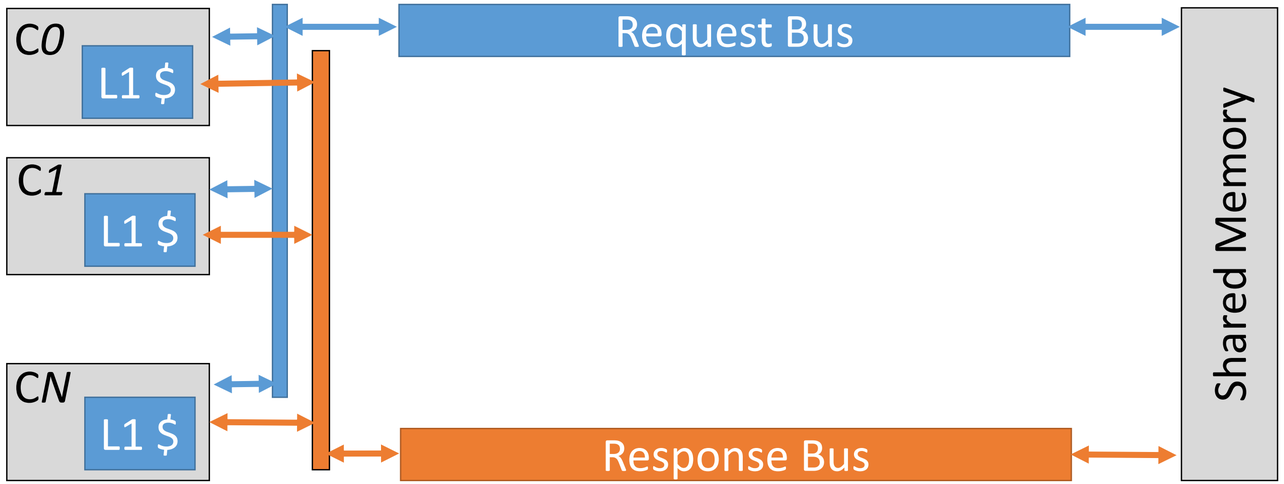
PISCOT High-Level Architecture

PISCOT

(1) Split-Bus

PISCOT implements a split-transaction bus:

- A Request Bus is responsible for broadcasting the coherence messages initiating memory requests, while
- A Response Bus transfers data as a response to these requests.



PISCOT High-Level Architecture

PISCOT

PISCOT High-Level Architecture

(1) Split-Bus

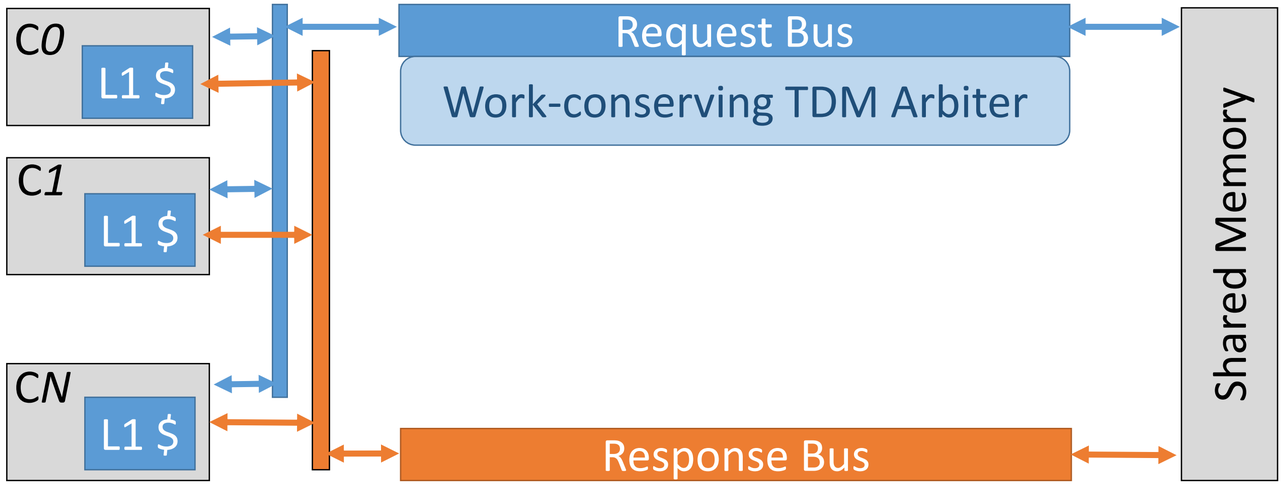
PISCOT implements a split-transaction bus:

- A Request Bus is responsible for broadcasting the coherence messages initiating memory requests, while
- A Response Bus transfers data as a response to these requests.

(2) Request Arbiter

Unlike COTS systems, the Request Bus Arbiter is a work-conserving TDM arbiter

→ *To achieve **PREDICTABILITY***



PISCOT

(1) Split-Bus

PISCOT implements a split-transaction bus:

- A Request Bus is responsible for broadcasting the coherence messages initiating memory requests, while
- A Response Bus transfers data as a response to these requests.

(2) Request Arbiter

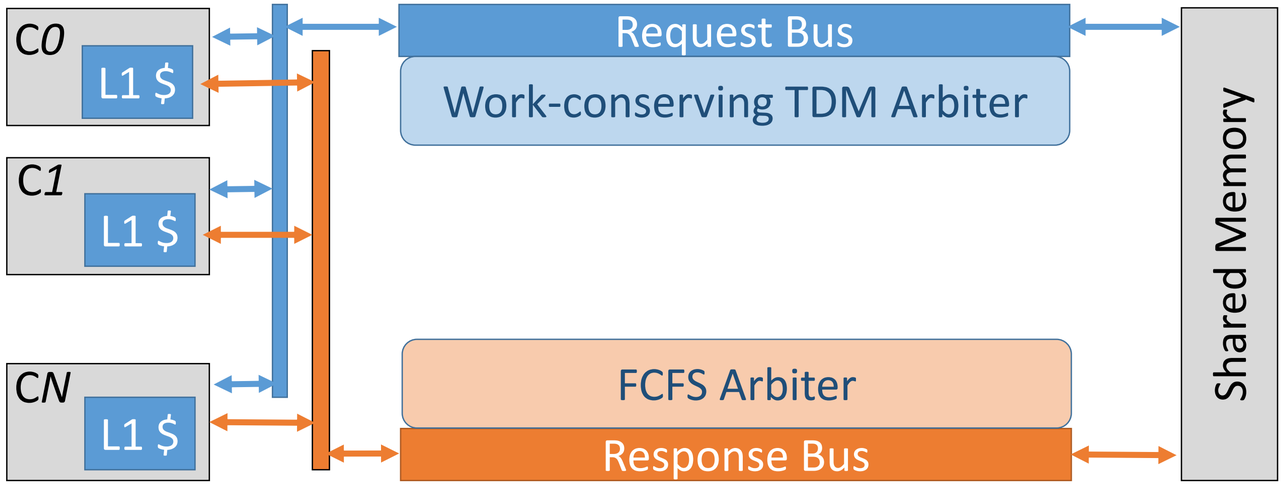
Unlike COTS systems, the Request Bus Arbiter is a work-conserving TDM arbiter

→ *To achieve **PREDICTABILITY***

(3) Response Arbiter

The Response Bus's arbiter is FCFS:

- *serves requests based on their arrival time on the Service Queue*



PISCOT High-Level Architecture

PISCOT

(1) Split-Bus

PISCOT implements a split-transaction bus:

- A Request Bus is responsible for broadcasting the coherence messages initiating memory requests, while
- A Response Bus transfers data as a response to these requests.

(2) Request Arbiter

Unlike COTS systems, the Request Bus Arbiter is a work-conserving TDM arbiter

→ *To achieve **PREDICTABILITY***

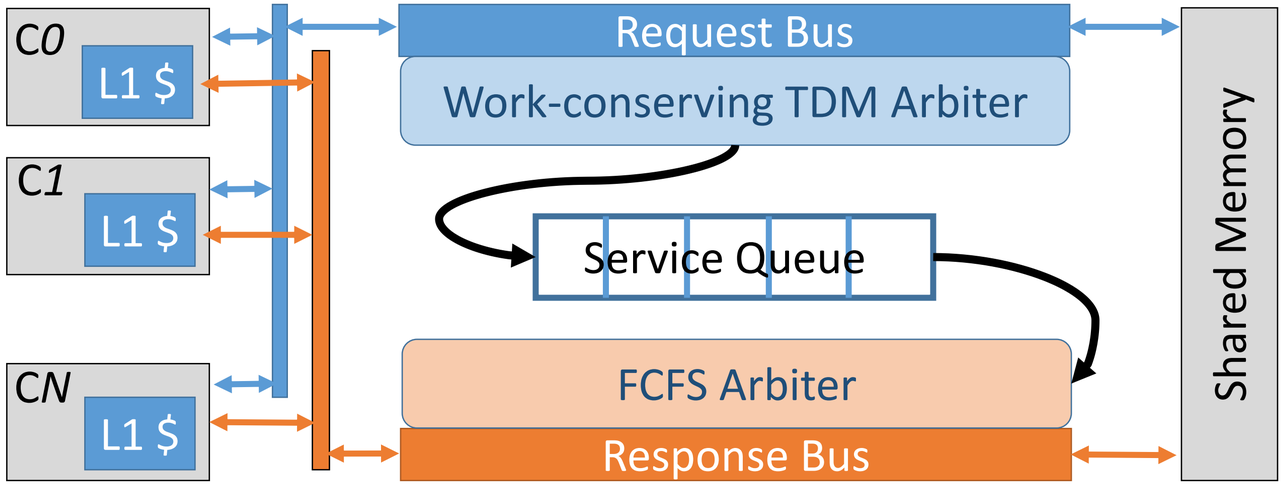
(3) Response Arbiter

The Response Bus's arbiter is FCFS:

- *serves requests based on their arrival time on the Service Queue*

(4) Service Queue

Both buses operate fully in parallel and synchronize using a Service Queue



PISCOT High-Level Architecture

PISCOT

PISCOT High-Level Architecture

(1) Split-Bus

PISCOT implements a split-transaction bus:

- A Request Bus is responsible for broadcasting the coherence messages initiating memory requests, while
- A Response Bus transfers data as a response to these requests.

(2) Request Arbiter

Unlike COTS systems, the Request Bus Arbiter is a work-conserving TDM arbiter

→ *To achieve **PREDICTABILITY***

(3) Response Arbiter

The Response Bus's arbiter is FCFS:

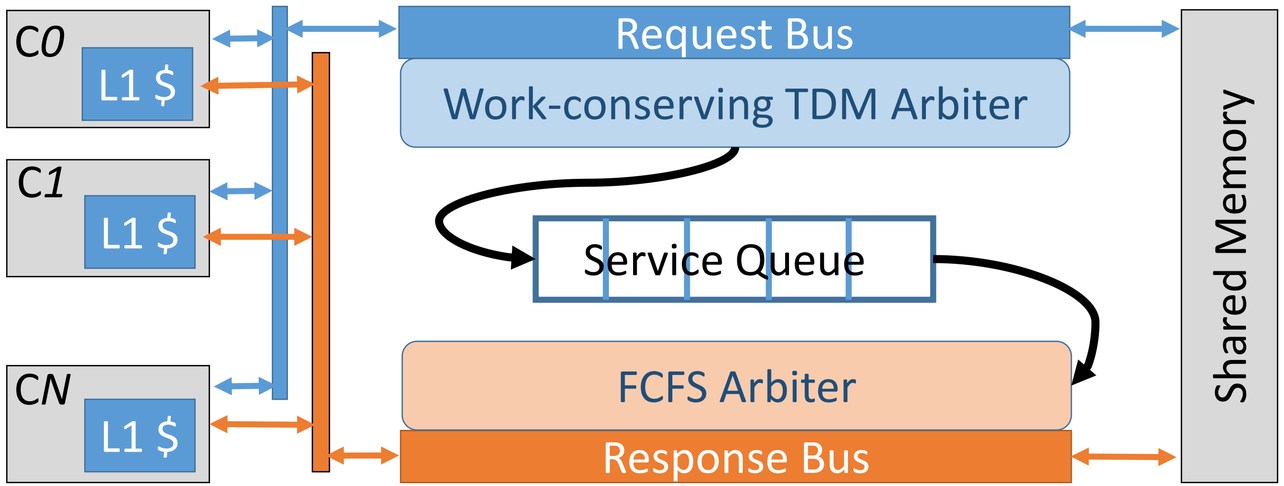
- *serves requests based on their arrival time on the Service Queue*

(4) Service Queue

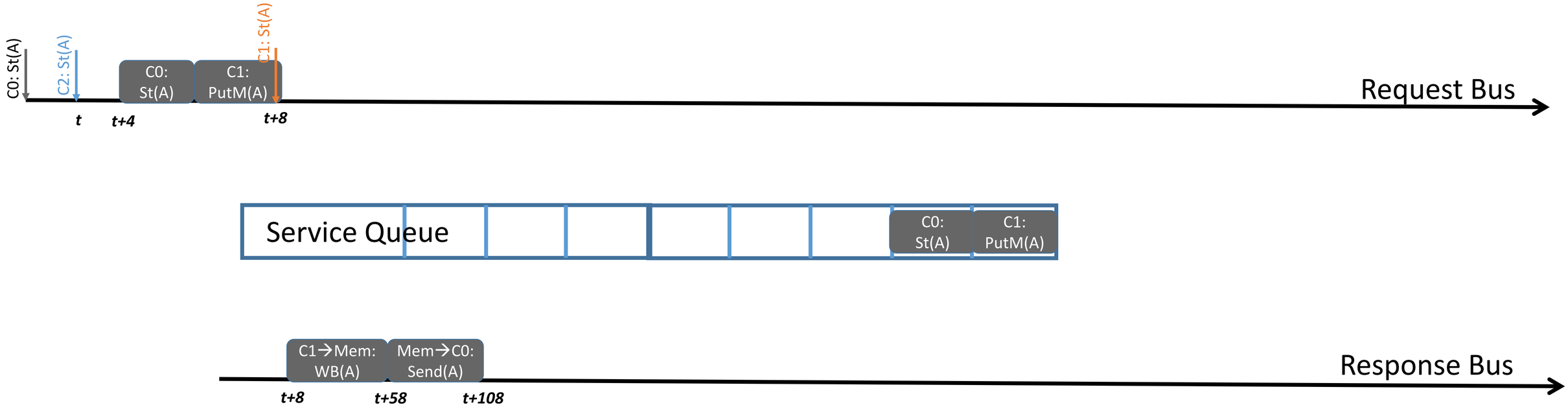
Both buses operate fully in parallel and synchronize using a Service Queue

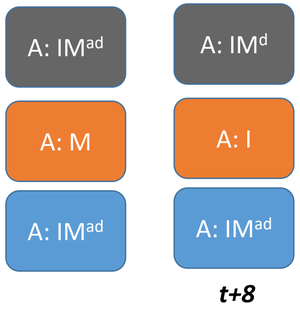
(5) Bounding Interference

- Allows cores to issue multiple outstanding requests.
- However, to limit coherence interference, it only services at most one request from any given core at a time.



PISCOT

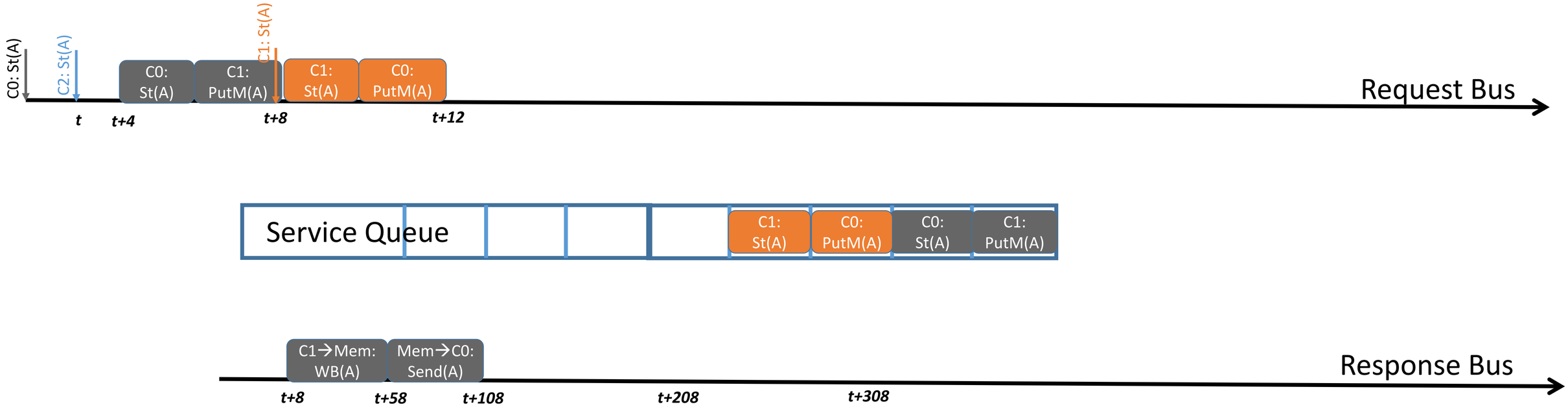


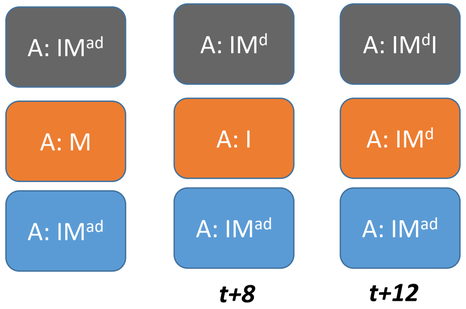


- Assume same example from before
- all 3 cores request to access same cache line A
- Assume initially A is modified by C1 in its private cache
- our request under analysis is St(A) from C2

Problem: Coherence effect on WC

EXAMPLE

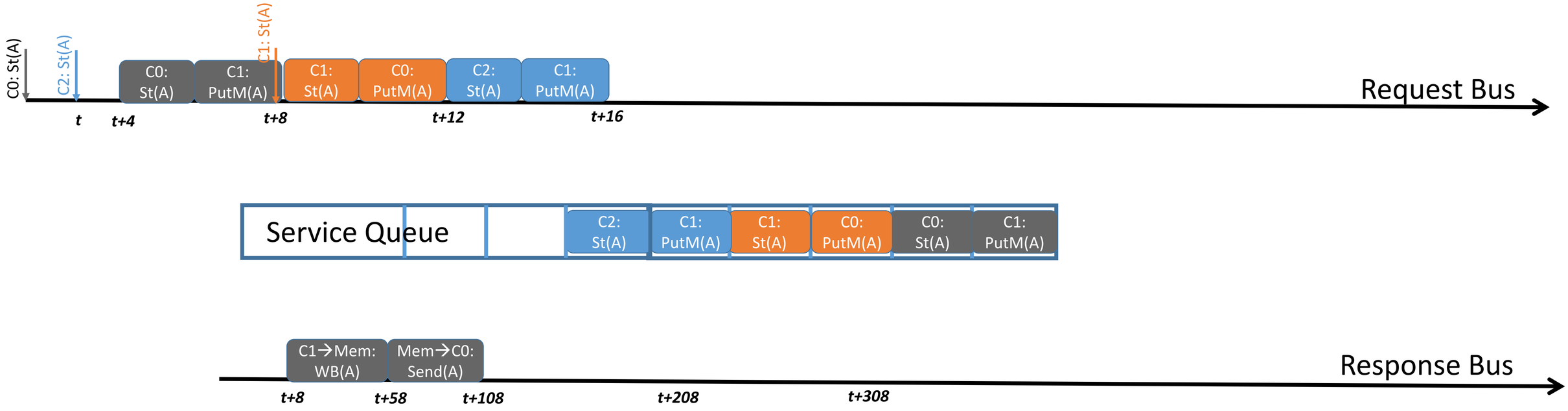


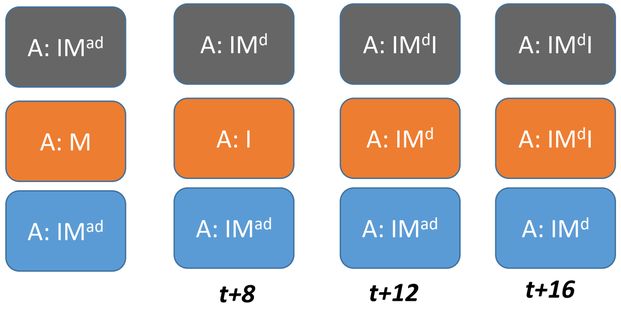


- Assume same example from before
- all 3 cores request to access same cache line A
- Assume initially A is modified by C1 in its private cache
- our request under analysis is St(A) from C2

Problem: Coherence effect on WC

EXAMPLE

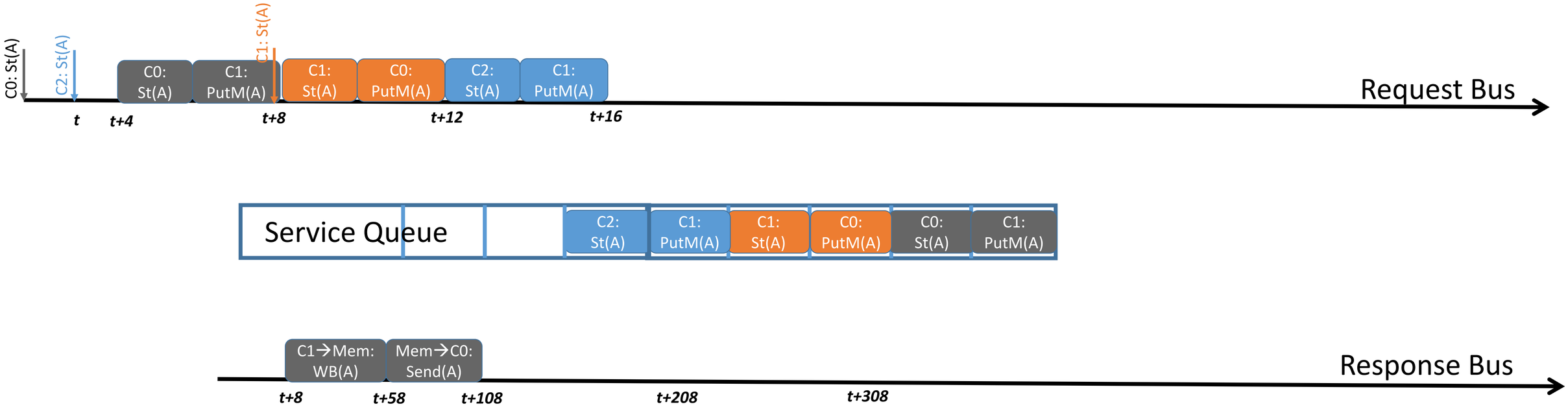


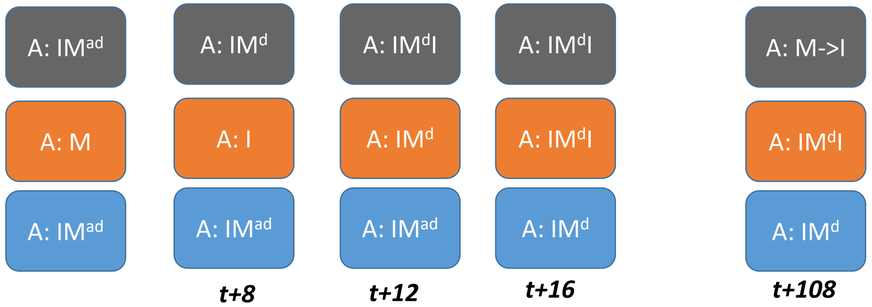


- Assume same example from before
- all 3 cores request to access same cache line A
- Assume initially A is modified by C1 in its private cache
- our request under analysis is St(A) from C2

Problem: Coherence effect on WC

EXAMPLE

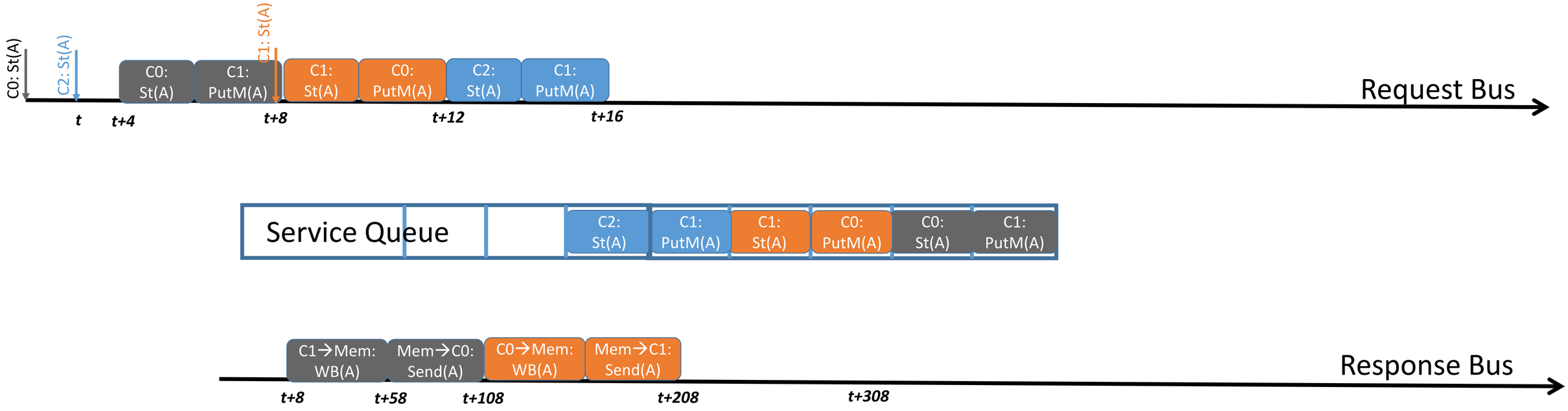


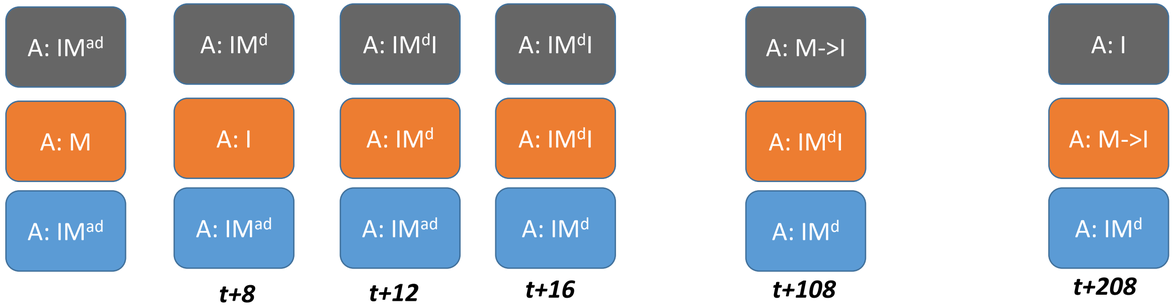


- Assume same example from before
- all 3 cores request to access same cache line A
- Assume initially A is modified by C1 in its private cache
- our request under analysis is St(A) from C2

Problem: Coherence effect on WC

EXAMPLE

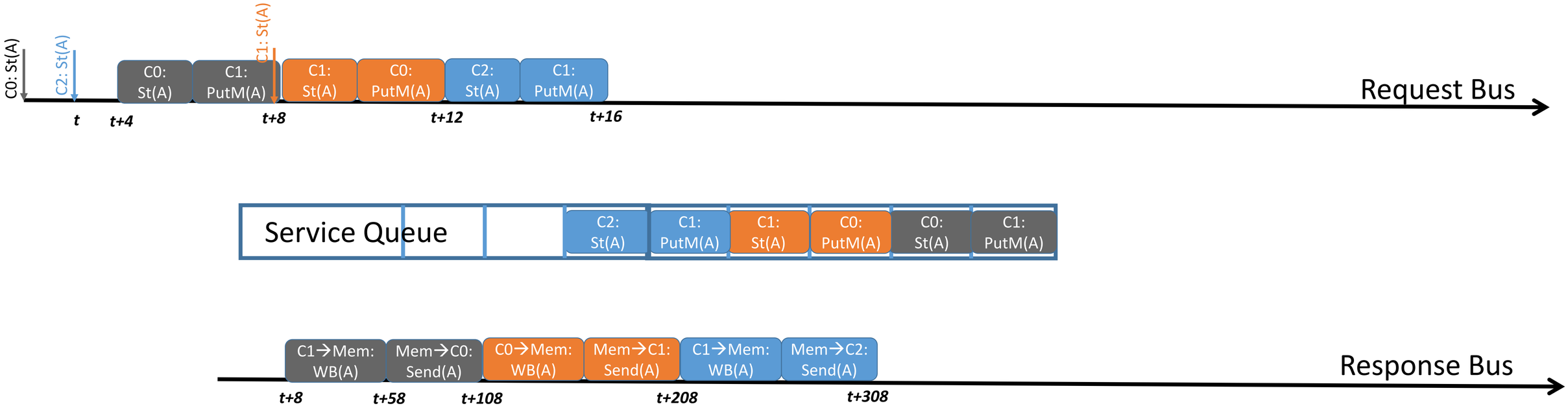


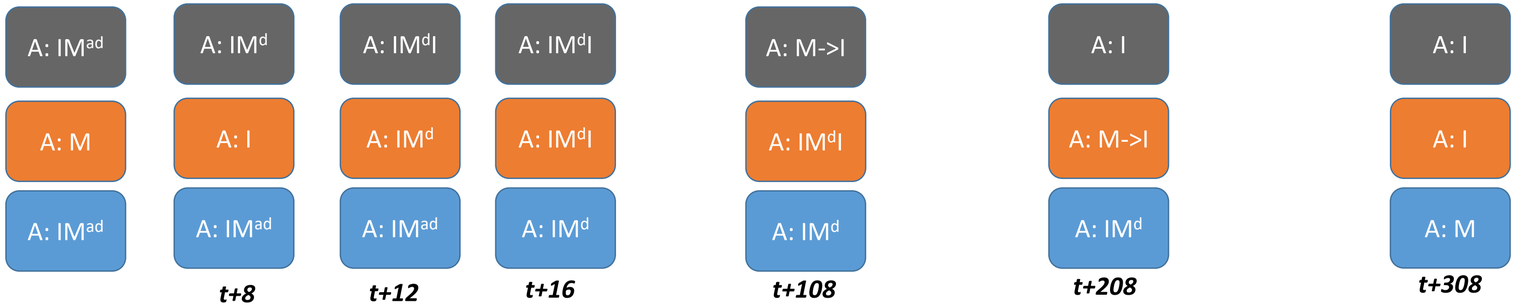


- Assume same example from before
- all 3 cores request to access same cache line A
- Assume initially A is modified by C1 in its private cache
- our request under analysis is St(A) from C2

Problem: Coherence effect on WC

EXAMPLE

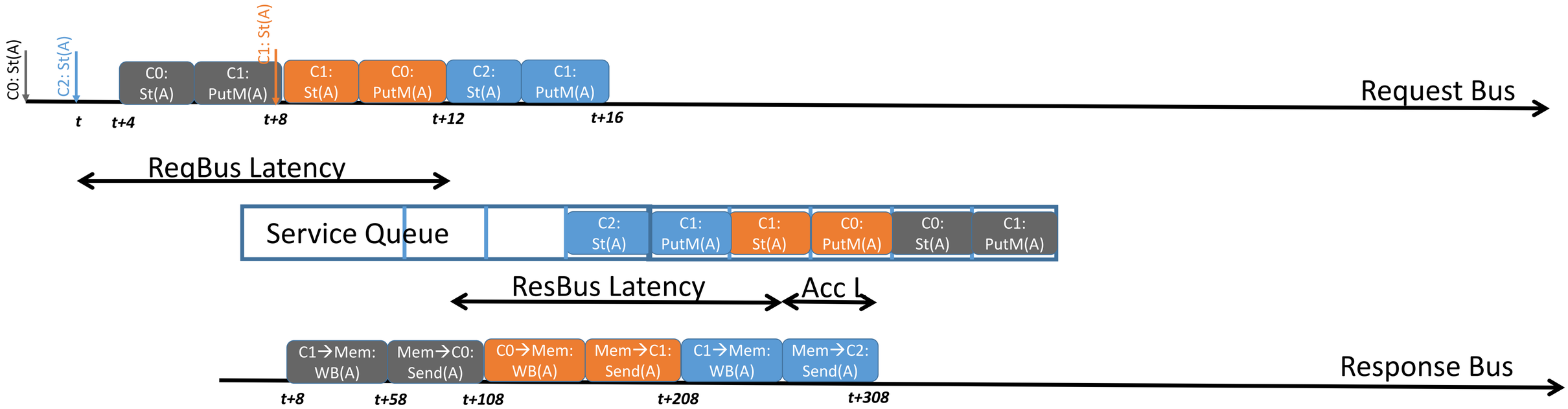


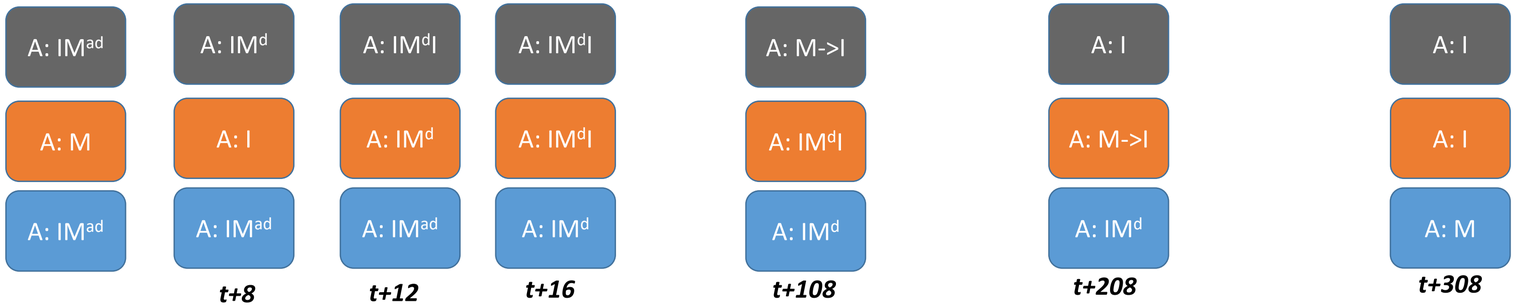


- Assume same example from before
- all 3 cores request to access same cache line A
- Assume initially A is modified by C1 in its private cache
- our request under analysis is St(A) from C2

Problem: Coherence effect on WC

EXAMPLE

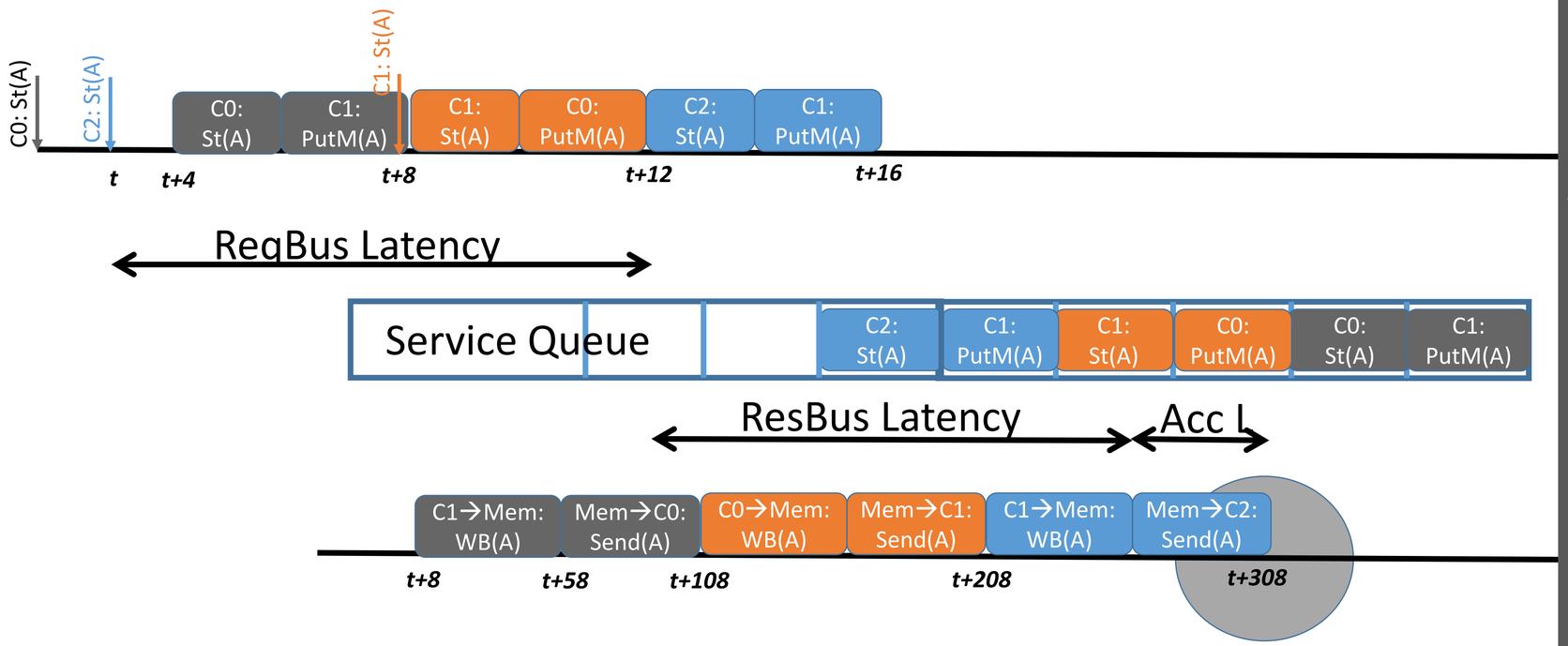


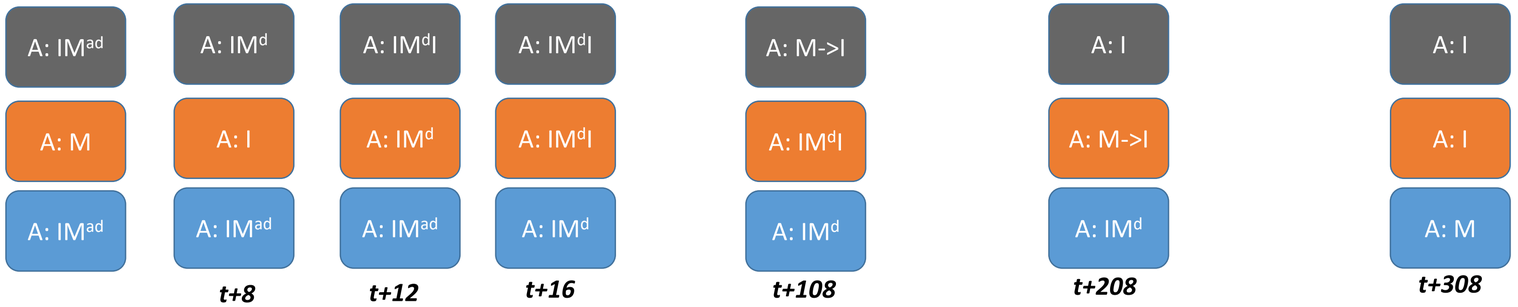


- Assume same example from before
- all 3 cores request to access same cache line A
- Assume initially A is modified by C1 in its private cache
- our request under analysis is St(A) from C2

Problem: Coherence effect on WC

EXAMPLE





Total Latency of 308 cycles compared to 1150 for PMSI

Generally, improves coherence interference from αN^2 to αN

- Assume s
- all 3 core
- Assume i
- our requ

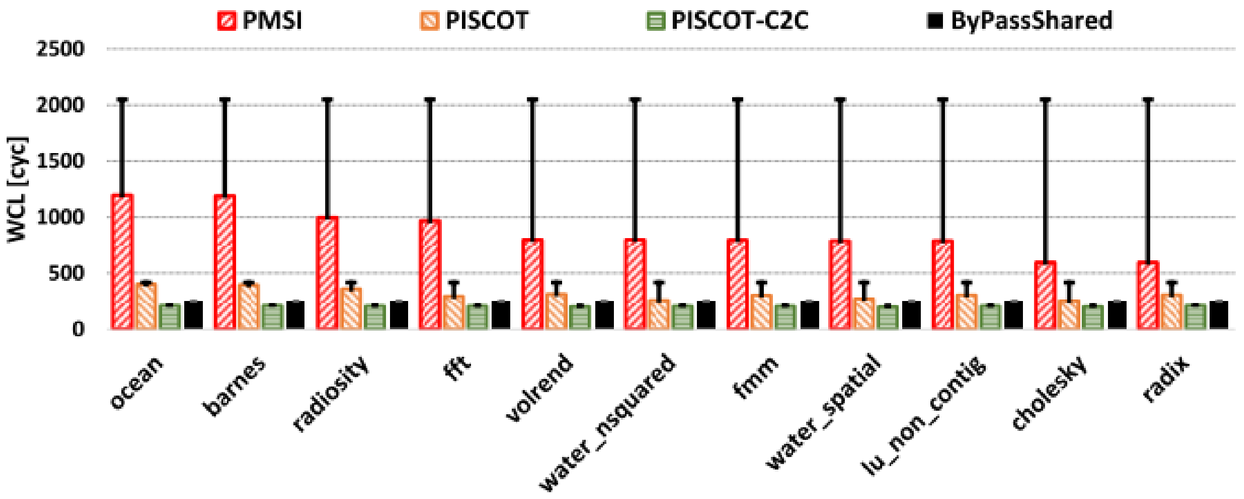
Problem: Coherence effect on WC

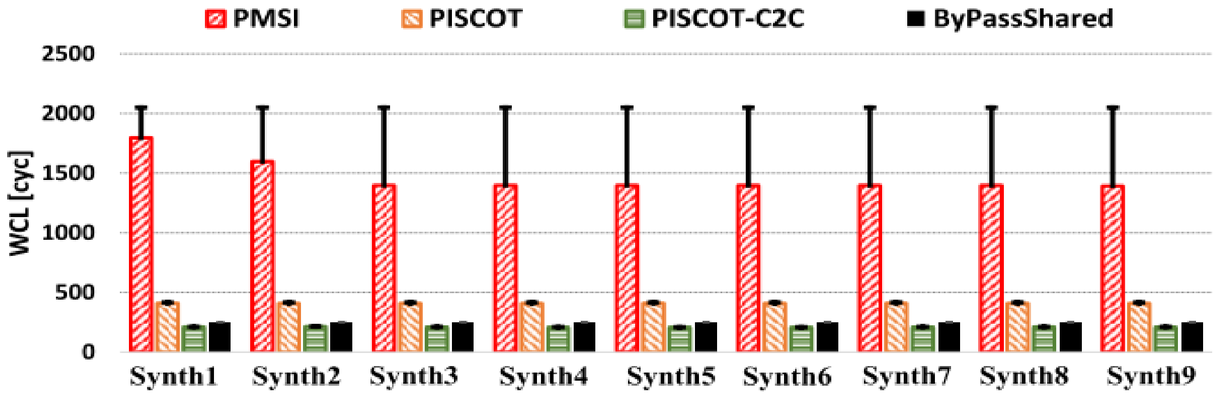
EXAMPLE

Evaluation

Predictability and Latency Bounds

- Four-core system and figures are for SPLASH-3 and synthetics workloads.
- PISCOT shows more than 4× improvement in both the analytical and observed WCL compared to PMSI

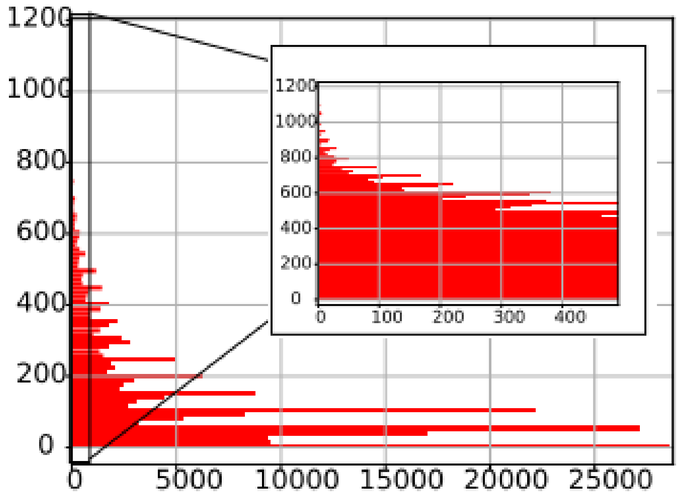




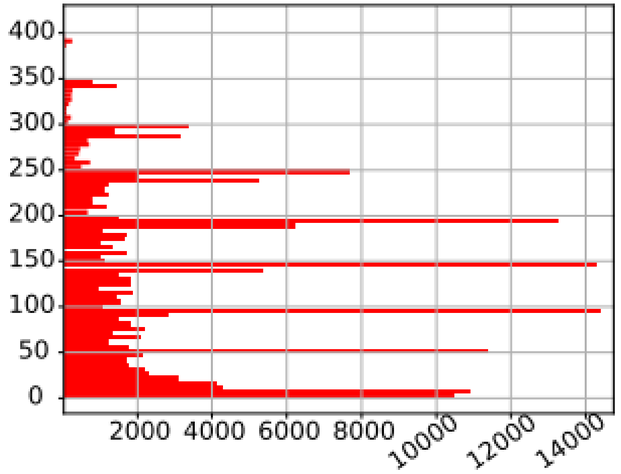
https://gitlab.com/FanosLab/piscot-and-msi-split-bus

Predictability and Latency Bounds

- Four-core system (Fig is for Ocean BM)
- PISCOT is able to avoid the unpredictability/ variability behavior of conventional MSI



(a) MSI



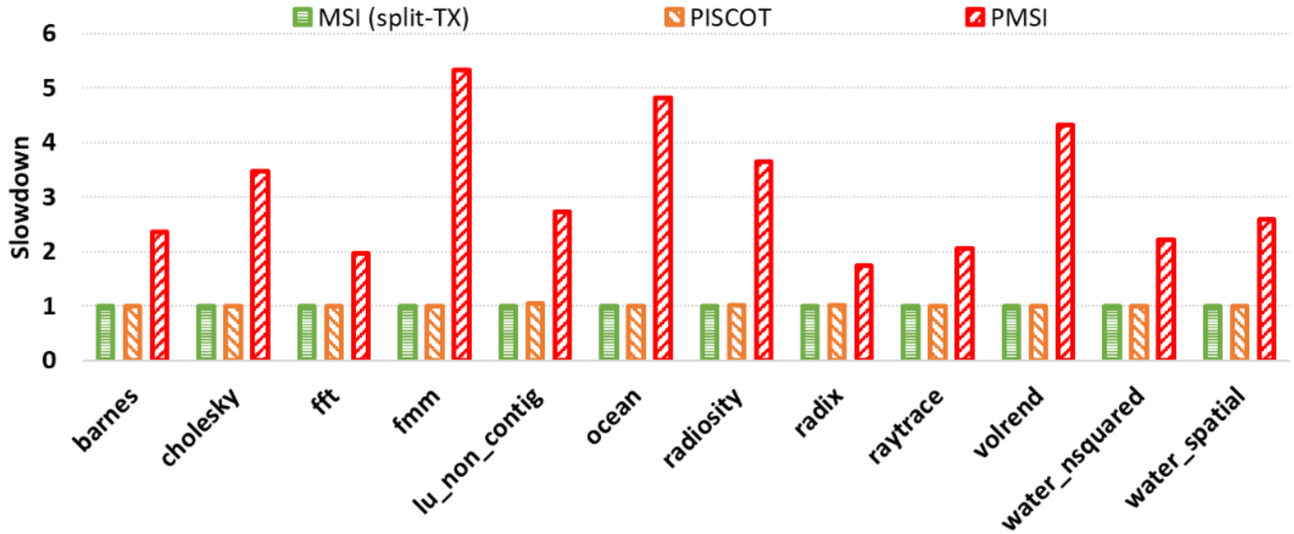
(b) PISCOT

Evaluation

https://gitlab.com/FanosLab/piscot-and-msi-split-bus

Average-Case Performance

- Slowdown compared to conventional MSI
- PMSI's slowdown is 2× on average (and up to 4.3×) across all benchmarks.
  - *coupling of coherence and data transfer*
- PISCOT achieves comparable results with slowdown in the range of 1% – 4%

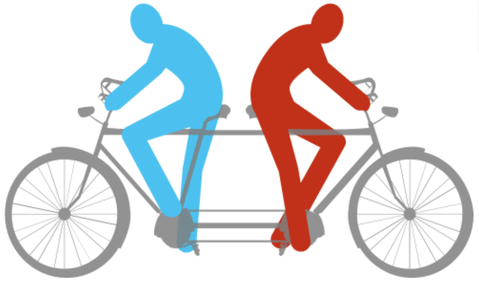


Evaluation

https://gitlab.com/FanosLab/piscot-and-msi-split-bus

Conventional COTS:

- High performance arbitration ✓
  - FCFS, Split-Tx, Priority-Based, ..
- Shared data is entirely supported through a hardware cache coherence protocol ✓✓
- Not Predictable ✗



Traditional Real-Time:

- Predictable by design ✓✓
  - TDM, RR, WRR, HRR, …
- Does not support shared data, put aside coherence ✗

PISCOT

- Predictable split-transaction bus
- Shared data is supported ✓✓
- With ***NO*** protocol modifications ✓✓
- Achieves better performance than existing coherence solutions ✓✓
- Also achives predictability, with tight latency bounds ✓✓

Data-Aware Arbitration:

- Builds on traditional arbitration
- Shared data is supported ✓✓
- BUT requires coherence protocol modifications ✓
- Coherence leads to good performance
- Also achives predictability, but with significant latency bounds ✓
- PMSI [RTAS'17], CARP [RTSS'19], HourGlass [Arxiv'18], PENDULUM [RTSS'19]

PISCOT vs. State-of-the-art

Data Sharing

Summary

SUMMARY

Conventional COTS:
- High performance arbitration ✓
  - FCFS, Split-Tx, Priority-Based, ..
- Shared data is entirely supported through a hardware cache coherence protocol ✓
- Not Predictable ✗

Traditional Real-Time:
- Predictable by design ✓
  - TDM, RR, WRR, HRR, ...
- Does not support shared data, put aside coherence ✗

PISCOT
- Predictable split-transaction bus
- Shared data is supported ✓
- With ***NO*** protocol modifications ✓
- Achieves better performance than existing coherence solutions ✓
- Also achives predictability, with tight latency bounds ✓

Data-Aware Arbitration:
- Builds on traditional arbitration
- Shared data is supported ✓
- BUT requires coherence protocol modifications ✓
- Coherence leads to good performance
- Also achives predictability, but with significan
- PMSI [RTAS'17], CARP [RTSS'19], Hour PENDULUM [RTSS'19]

PISCOT vs. State-of-the-art

(1) Split-Bus

PISCOT implements a split-transaction bus:
- A Request Bus is responsible for broadcasting the coherence messages initiating memory requests, while
- A Response Bus transfers data as a response to these requests.

(2) Request Arbiter

Unlike COTS systems, the Request Bus Arbiter is a work-conserving TDM arbiter

→ *To achieve **PREDICTABILITY***

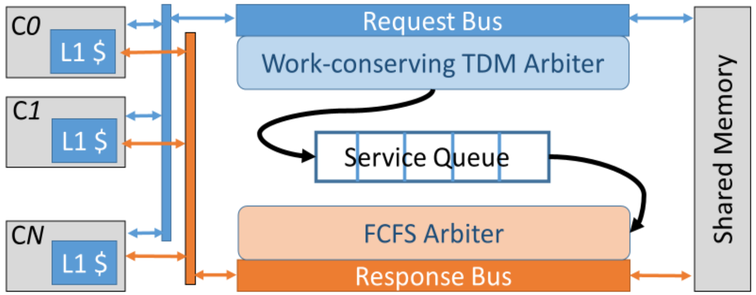
(3) Response Arbiter

The Response Bus's arbiter is FCFS:
- *serves requests based on their arrival time on the Service Queue*

(4) Service Queue

Both buses operate fully in parallel and synchronize using a Service Queue

(5) Bounding Interference
- Allows cores to issue multiple outstanding requests.
- However, to limit coherence interference, it only services at most one request from any given core at a time.



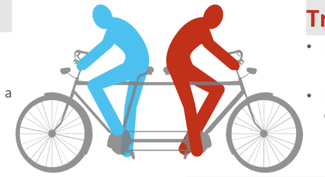
PISCOT High-Level Architecture

Summary

SUMMARY

Conventional COTS:

- High performance arbitration ✓
  - FCFS, Split-Tx, Priority-Based, ..
- Shared data is entirely supported through a hardware cache coherence protocol ✓✓
- Not Predictable ✗



Traditional Real-Time:

- Predictable by design ✓✓
  - TDM, RR, WRR, HRR, …
- Does not support shared data, put aside coherence ✗

PISCOT

- Predictable split-transaction bus
- Shared data is supported ✓✓
- With ***NO*** protocol modifications ✓✓
- Achieves better performance than existing coherence solutions ✓✓
- Also achives predictability, with tight latency bounds ✓✓

Data-Aware Arbitration:

- Builds on traditional arbitration
- Shared data is supported ✓✓
- BUT requires coherence protocol modifications ✓
- Coherence leads to good performance
- Also achives predictability, but with significant latency bounds ✓
- PMSI [RTAS'17], CARP [RTSS'19], HourGlass [Arxiv'18], PENDULUM [RTSS'19]

PISCOT vs. State-of-the-art

Data Sharing

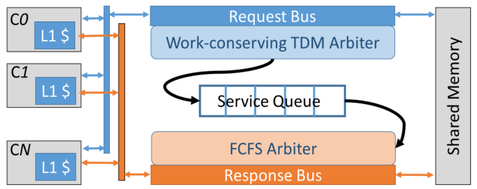
(1) Split-Bus
PISCOT implements a split-transaction bus:
- A Request Bus is responsible for broadcasting the coherence messages initiating memory requests, while
- A Response Bus transfers data as a response to these requests.

(2) Request Arbiter
Unlike COTS systems, the Request Bus Arbiter is a work-conserving TDM arbiter
→ *To achieve **PREDICTABILITY***

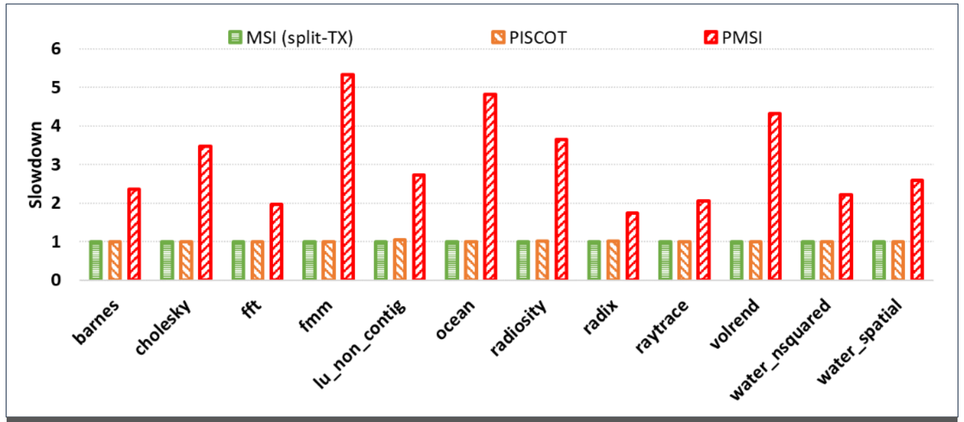
(3) Response Arbiter
The Response Bus's arbiter is FCFS:
- *serves requests based on their arrival time on the Service Queue*

(4) Service Queue
Both buses operate fully in parallel and synchronize using a Service Queue

(5) Bounding Interference
- Allows cores to issue multiple outstanding requests.
- However, to limit coherence interference, it only services at most one request from any given core at a time.



PISCOT High-Level Architecture



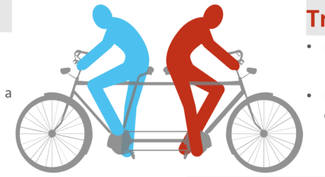
Comparable COTS Performance

Summary

SUMMARY

Conventional COTS:

- High performance arbitration ✓
  - FCFS, Split-Tx, Priority-Based, ..
- Shared data is entirely supported through a hardware cache coherence protocol ✓
- Not Predictable ✗



Traditional Real-Time:

- Predictable by design ✓
  - TDM, RR, WRR, HRR, …
- Does not support shared data, put aside coherence ✗

PISCOT

- Predictable split-transaction bus
- Shared data is supported ✓
- With ***NO*** protocol modifications ✓
- Achieves better performance than existing coherence solutions ✓
- Also achives predictability, with tight latency bounds ✓

Data-Aware Arbitration:

- Builds on traditional arbitration
- Shared data is supported ✓
- BUT requires coherence protocol modifications ✓
- Coherence leads to good performance
- Also achives predictability, but with significant latency bounds ✓
- PMSI [RTAS'17], CARP [RTSS'19], HourGlass [Arxiv'18], PENDULUM [RTSS'19]

PISCOT vs. State-of-the-art

Data Sharing

(1) Split-Bus

PISCOT implements a split-transaction bus:

- A Request Bus is responsible for broadcasting the coherence messages initiating memory requests, while
- A Response Bus transfers data as a response to these requests.

(2) Request Arbiter

Unlike COTS systems, the Request Bus Arbiter is a work-conserving TDM arbiter

→ *To achieve **PREDICTABILITY***

(3) Response Arbiter

The Response Bus's arbiter is FCFS:

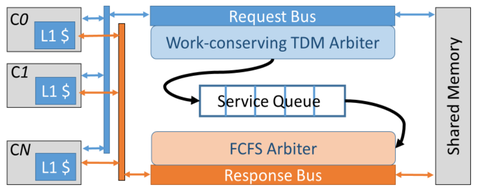
- *serves requests based on their arrival time on the Service Queue*

(4) Service Queue

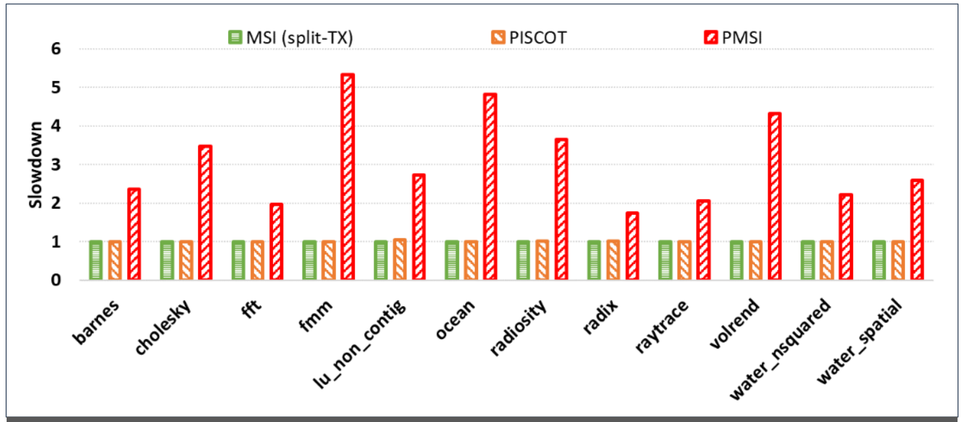
Both buses operate fully in parallel and synchronize using a Service Queue

(5) Bounding Interference

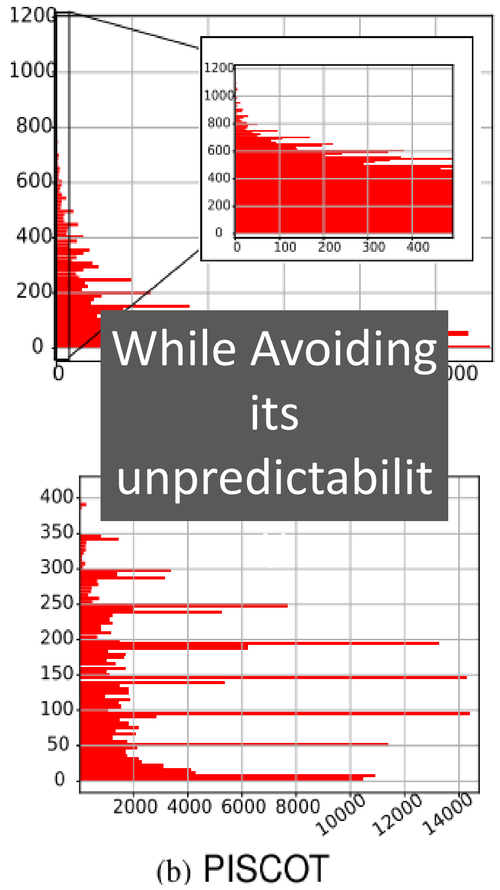
- Allows cores to issue multiple outstanding requests.
- However, to limit coherence interference, it only services at most one request from any given core at a time.



PISCOT High-Level Architecture



Comparable COTS Performance



While Avoiding its unpredictabilit

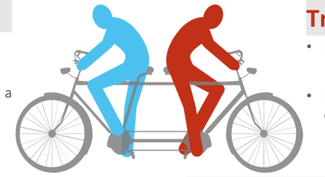
(b) PISCOT

Summary

SUMMARY

Conventional COTS:

- High performance arbitration ✓
  - FCFS, Split-Tx, Priority-Based, ..
- Shared data is entirely supported through a hardware cache coherence protocol ✓
- Not Predictable ✗



Traditional Real-Time:

- Predictable by design ✓
  - TDM, RR, WRR, HRR, …
- Does not support shared data, put aside coherence ✗

PISCOT

- Predictable split-transaction bus
- Shared data is supported ✓
- With ***NO*** protocol modifications ✓
- Achieves better performance than existing coherence solutions ✓
- Also achives predictability, with tight latency bounds ✓

Data-Aware Arbitration:

- Builds on traditional arbitration
- Shared data is supported ✓
- BUT requires coherence protocol modifications ✓
- Coherence leads to good performance
- Also achives predictability, but with significant latency bounds ✓
- PMSI [RTAS'17], CARP [RTSS'19], HourGlass [Arxiv'18], PENDULUM [RTSS'19]

PISCOT vs. State-of-the-art

Data Sharing

(1) Split-Bus

PISCOT implements a split-transaction bus:

- A Request Bus is responsible for broadcasting the coherence messages initiating memory requests, while
- A Response Bus transfers data as a response to these requests.

(2) Request Arbiter

Unlike COTS systems, the Request Bus Arbiter is a work-conserving TDM arbiter

→ *To achieve **PREDICTABILITY***

(3) Response Arbiter

The Response Bus's arbiter is FCFS:

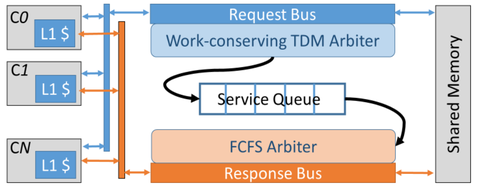
- *serves requests based on their arrival time on the Service Queue*

(4) Service Queue

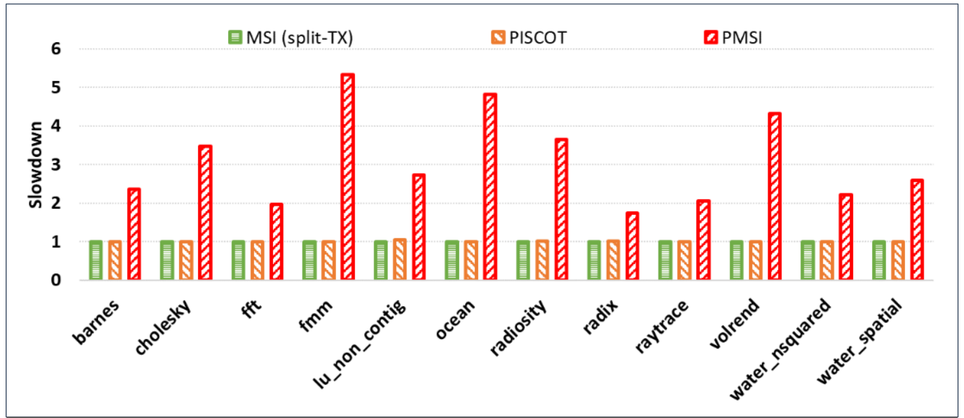
Both buses operate fully in parallel and synchronize using a Service Queue

(5) Bounding Interference

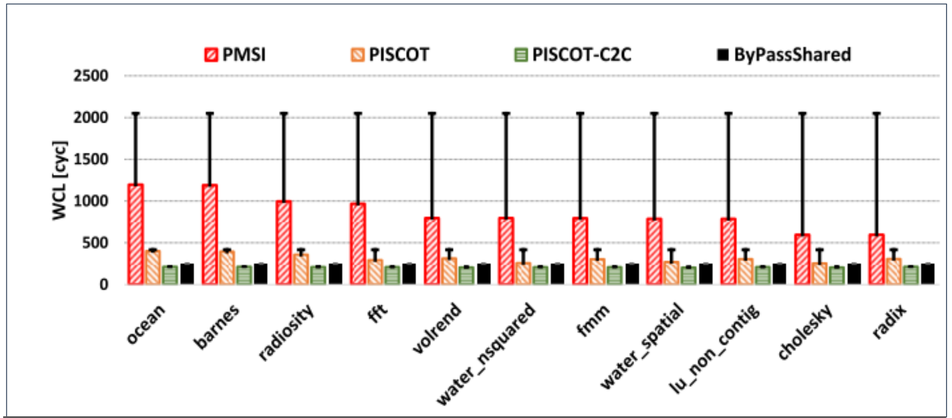
- Allows cores to issue multiple outstanding requests.
- However, to limit coherence interference, it only services at most one request from any given core at a time.



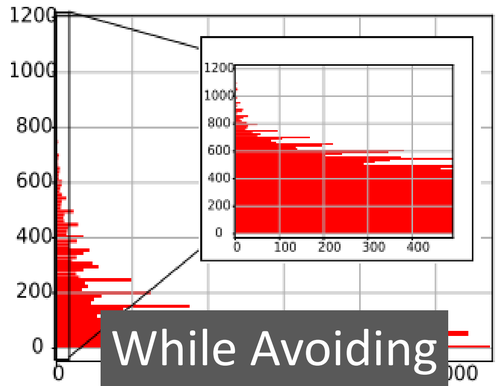
PISCOT High-Level Architecture



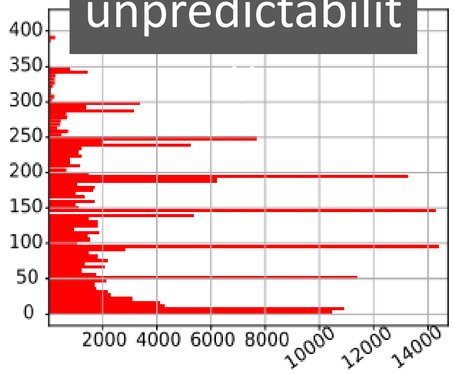
Comparable COTS Performance



And significantly improves coherence Latency Bounds



While Avoiding its unpredictabilit



(b) PISCOT

Summary

SUMMARY