Leveraging Cache Coherence to Detect and Repair False Sharing On-the-fly

MICRO'24

Vipin Patel, Swarnendu Biswas, and Mainak Chaudhuri

PROSPAR, Computer Science and Engineering,
Indian Institute of Technology Kanpur

Artifact available at:
https://zenodo.org/records/13293424
Or Scan the QR code

False Sharing in Real World Applications

Bugs Home | Report a bug | Advanced search | Saved searches | Tags

Bug #87350	**CACHE_LINE_SIZE in innodb should be 128 on aarch64**
Submitted:	9 Aug 2017 2:31
Reporter:	jared ZHU
Status:	Verified
Category:	MySQL Server: InnoDB storage engine
Version:	8.0.2,5.6,5.7
Assigned to:	
Tags:	CACHE_LINE_SIZE

se sharing in boost::detail::spinlock_pool?

d 8 years, 9 months ago Active 8 years, 8 months ago Viewed 2k times

I came across this SO question and reading it over eventually led me to look at `boost::detail::spinlock_pool`.

The purpose of `boost::detail::spinlock_pool` is to reduce potential contention for a global

JDK-7029167 : add support for conditional card marks

Type: Enhancement
Component: hotspot
Sub-Component: compiler
Affected Version: hs21

Priority: P3
Status: Closed
Resolution: Fixed
OS: solaris_10
CPU: x86

Submitted: 2011-03-18
Updated: 2011-07-29
Resolved: 2011-05-10

```
class LhsPadding
{
    protected long p1, p2, p3, p4, p5, p
}

class Value extends LhsPadding
{
    protected volatile long value;
}

class RhsPadding extends Value
{
    protected long p9, p10, p11, p12, p13, p14, p15;
}

/** <p>Concurrent sequence class used for tracking the progress
public class Sequence extends RhsPadding
```

False sharing in net_device and device
False sharing causes contention for read-only structure fields.
False sharing in page
False sharing causes contention for read-mostly structure fields.

Relates :

Description

False sharing induced by card table marking

The Netflix Case Study

Seeing through hardware counters: a journey to threefold performance increase

Netflix Technology Blog · Follow

Published in Netflix TechBlog · 10 min read · Nov 10, 2022

By Vadim Filanovsky and Harshad Sane

The Netflix Case Study

Seeing through hardware counters: a journey to threefold performance increase

JDK / JDK-8180450

secondary_super_cache does not scale well

Resolved

Details

Type:	Bug	Resolution:	Fixed
Priority:	P3	Fix Version/s:	23
Affects Version/s:	9, 10, 11, 12, 13, 17, 20, 21, 22		
Component/s:	hotspot		
Labels:	amazon-interest, hs-comp-triaged, jdk21-defer-request, jdk21-defer-yes, oracle-triage-14, performance		
Subcomponent:	compiler		
Resolved In Build:	b19		

Description

On some workloads, updates to the Klass::secondary_super_cache field cause excessive cache line invalidation traffic, with noticeable slowdowns.

People

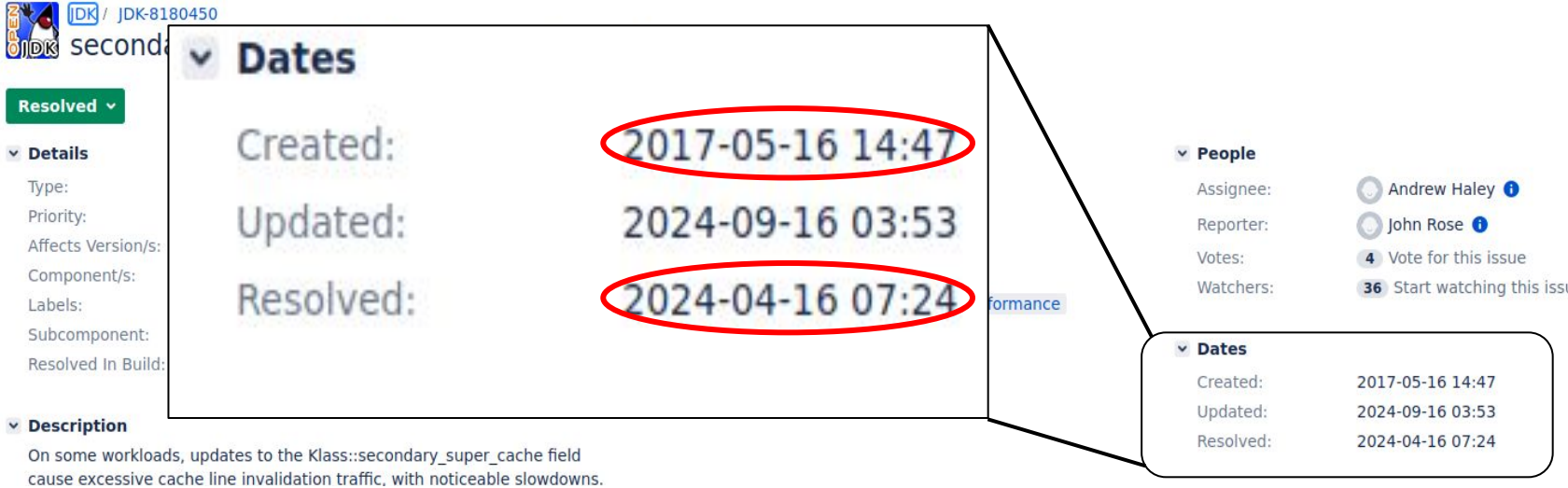
Assignee:	Andrew Haley
Reporter:	John Rose
Votes:	4 Vote for this issue
Watchers:	36 Start watching this issue

Dates

Created:	2017-05-16 14:47
Updated:	2024-09-16 03:53
Resolved:	2024-04-16 07:24

The Netflix Case Study

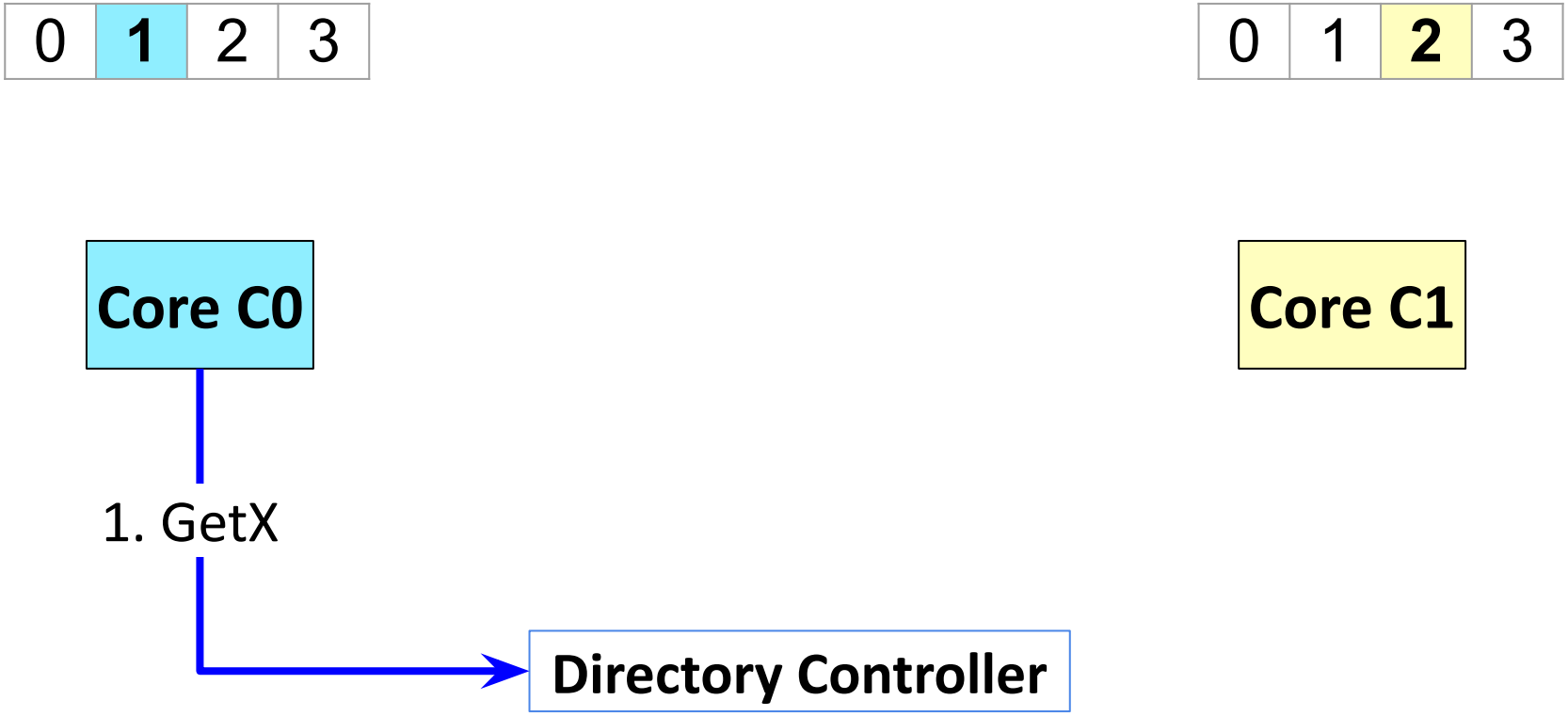
Seeing through hardware counters: a journey to threefold performance increase



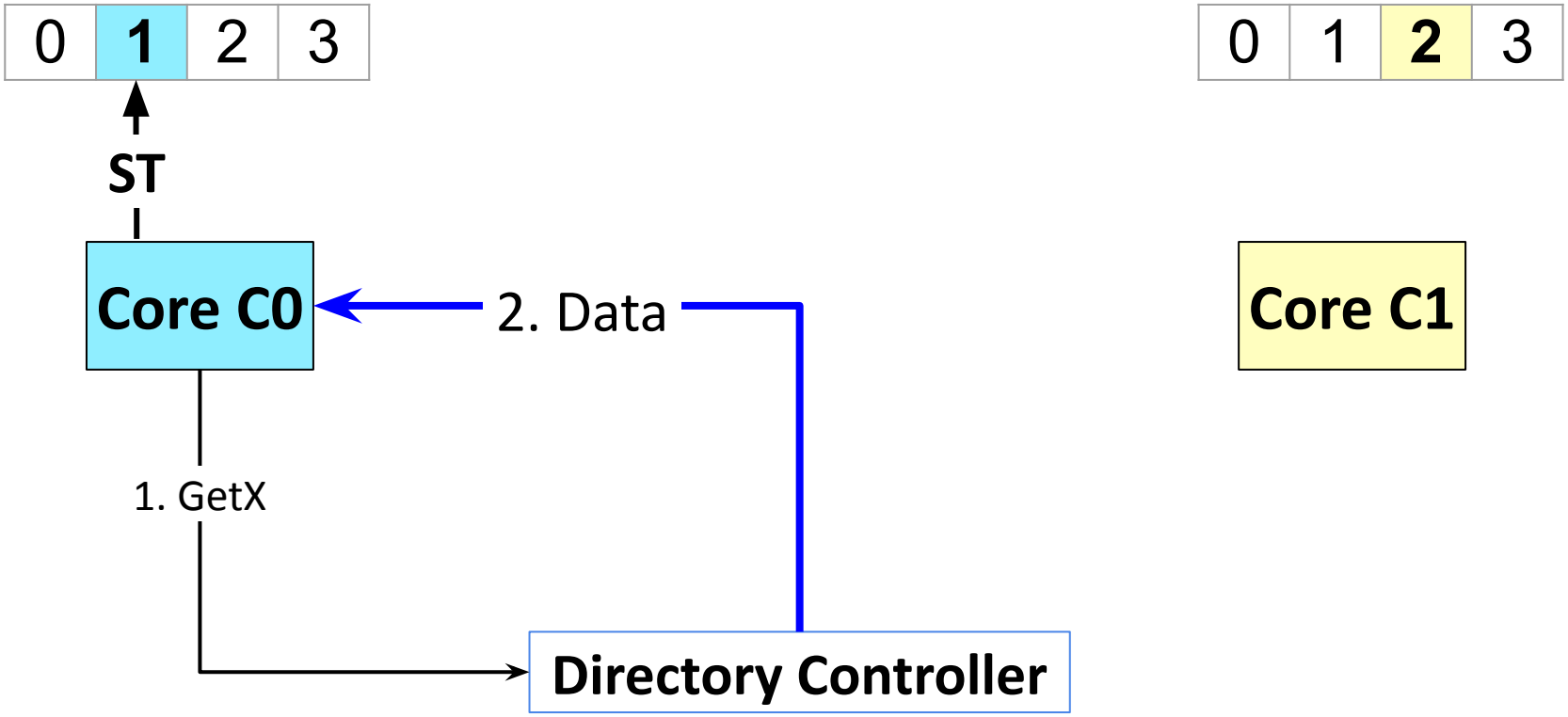
Cache Contention due to False Sharing

Cache Contention due to False Sharing

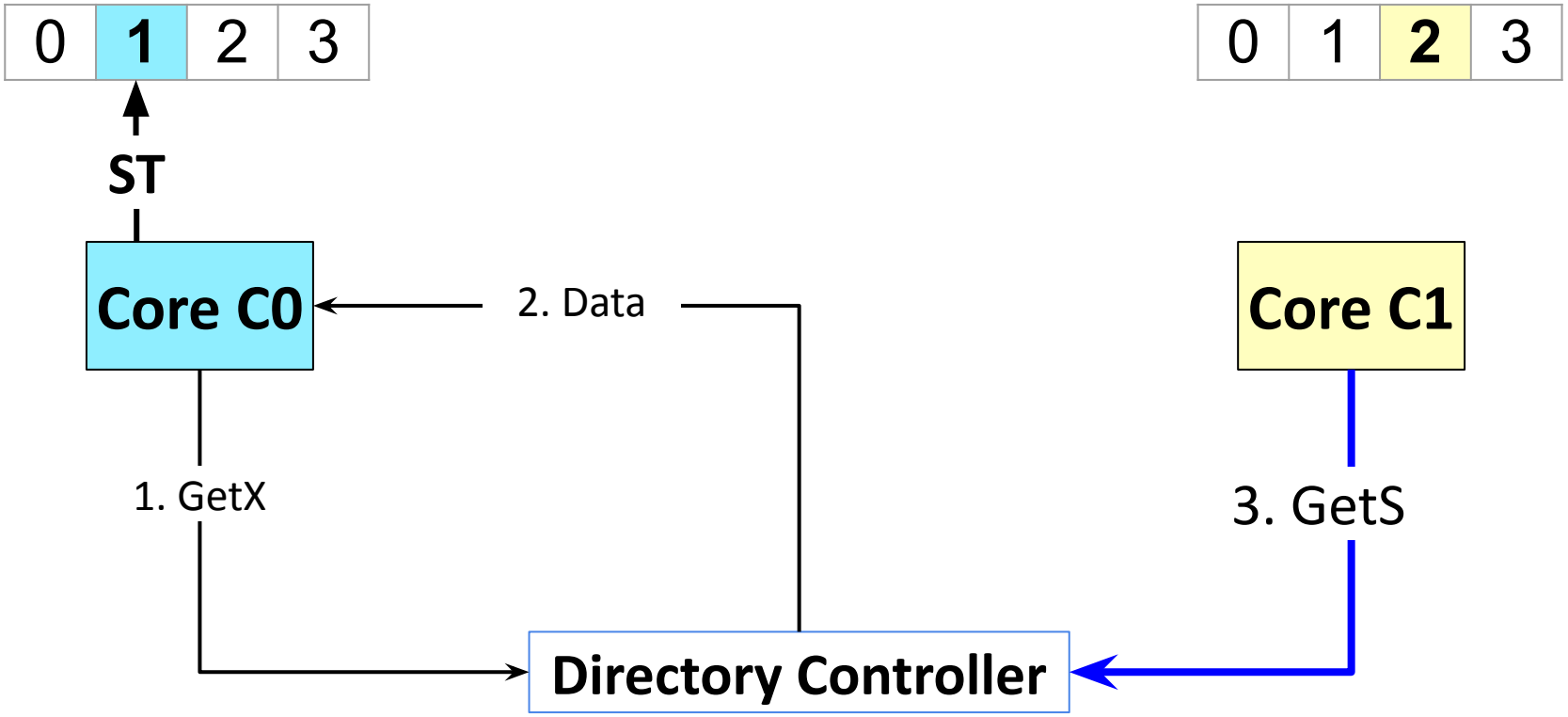
Cache Contention due to False Sharing



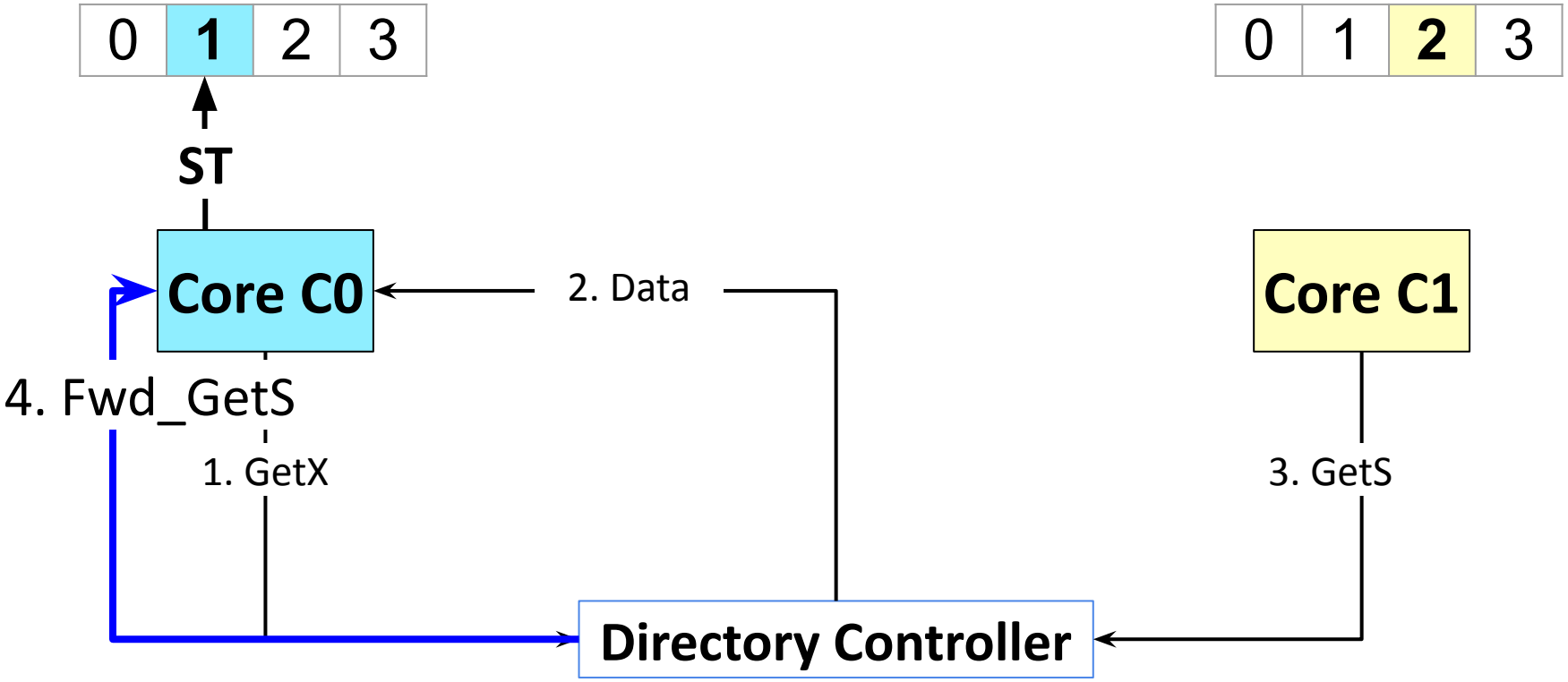
Cache Contention due to False Sharing



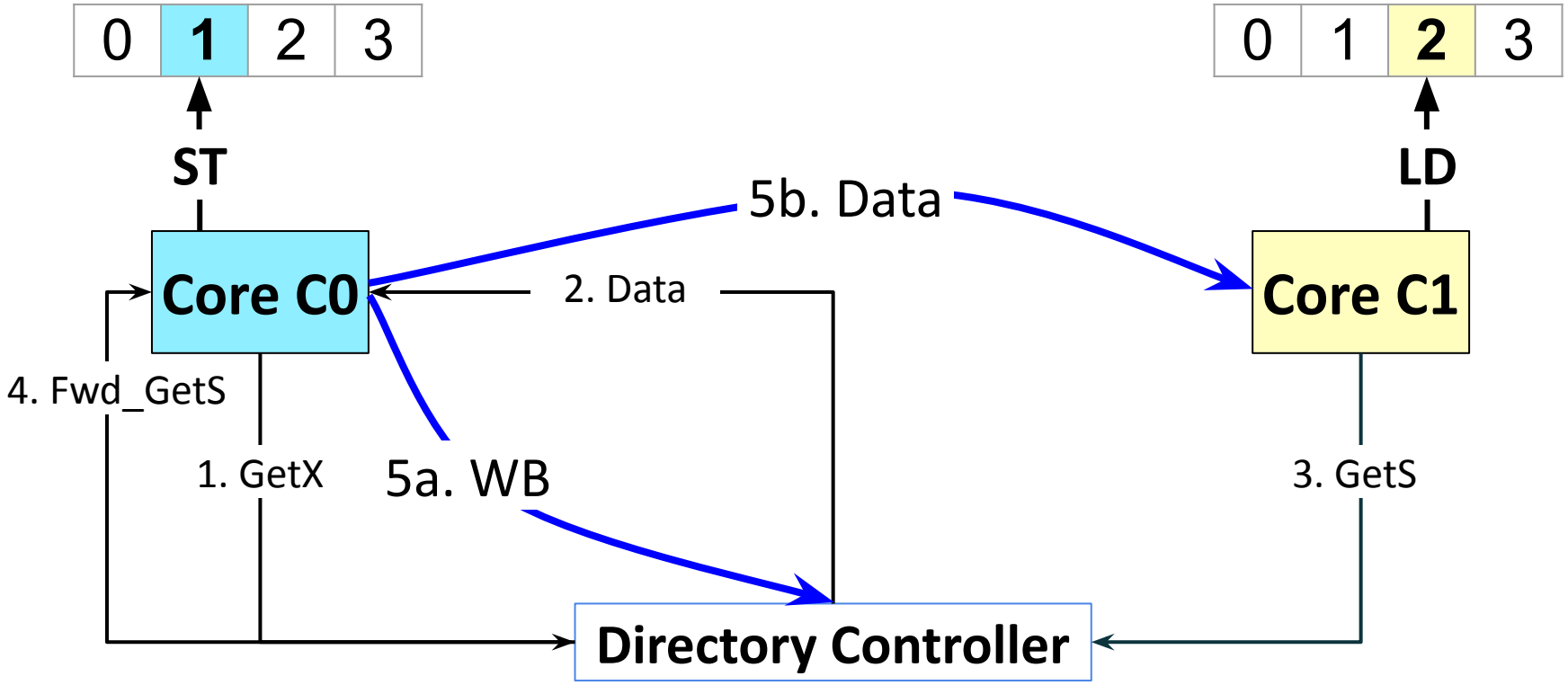
Cache Contention due to False Sharing



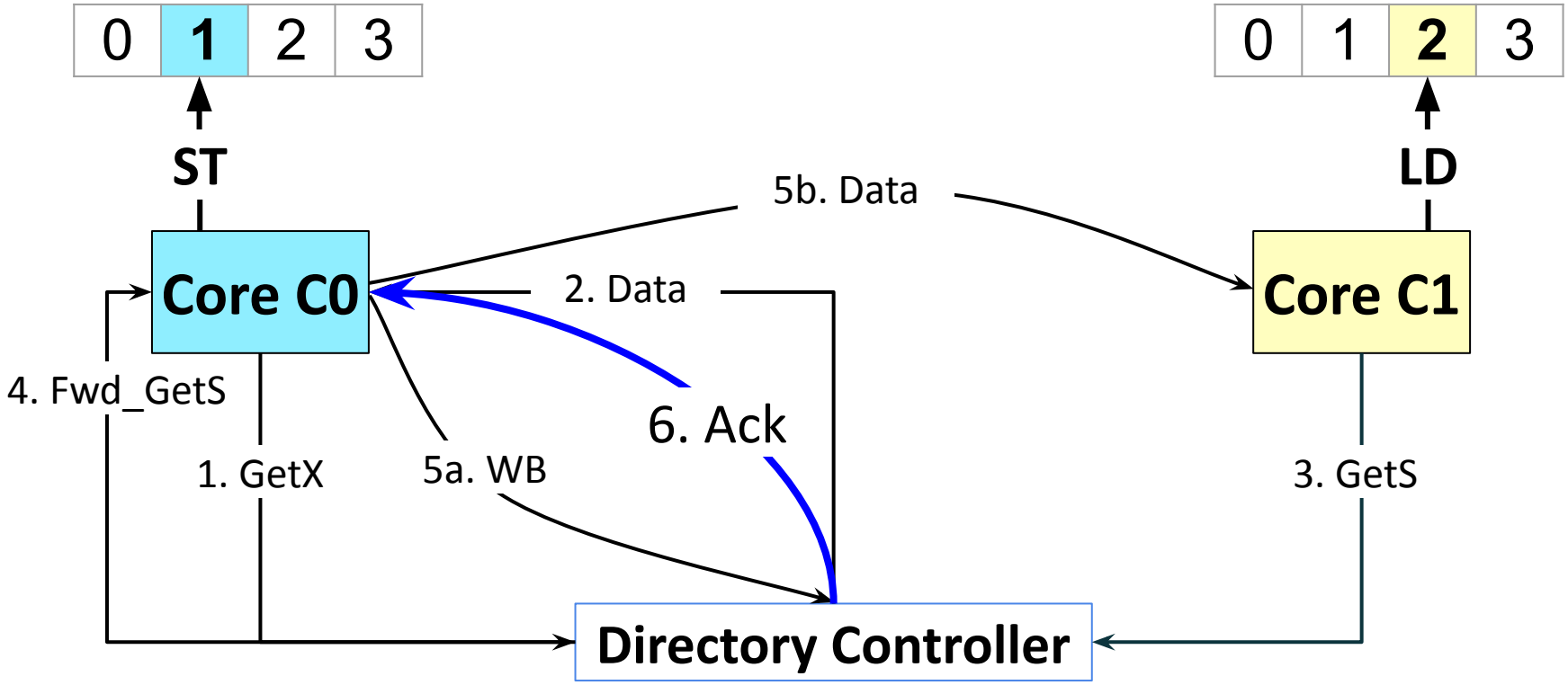
Cache Contention due to False Sharing



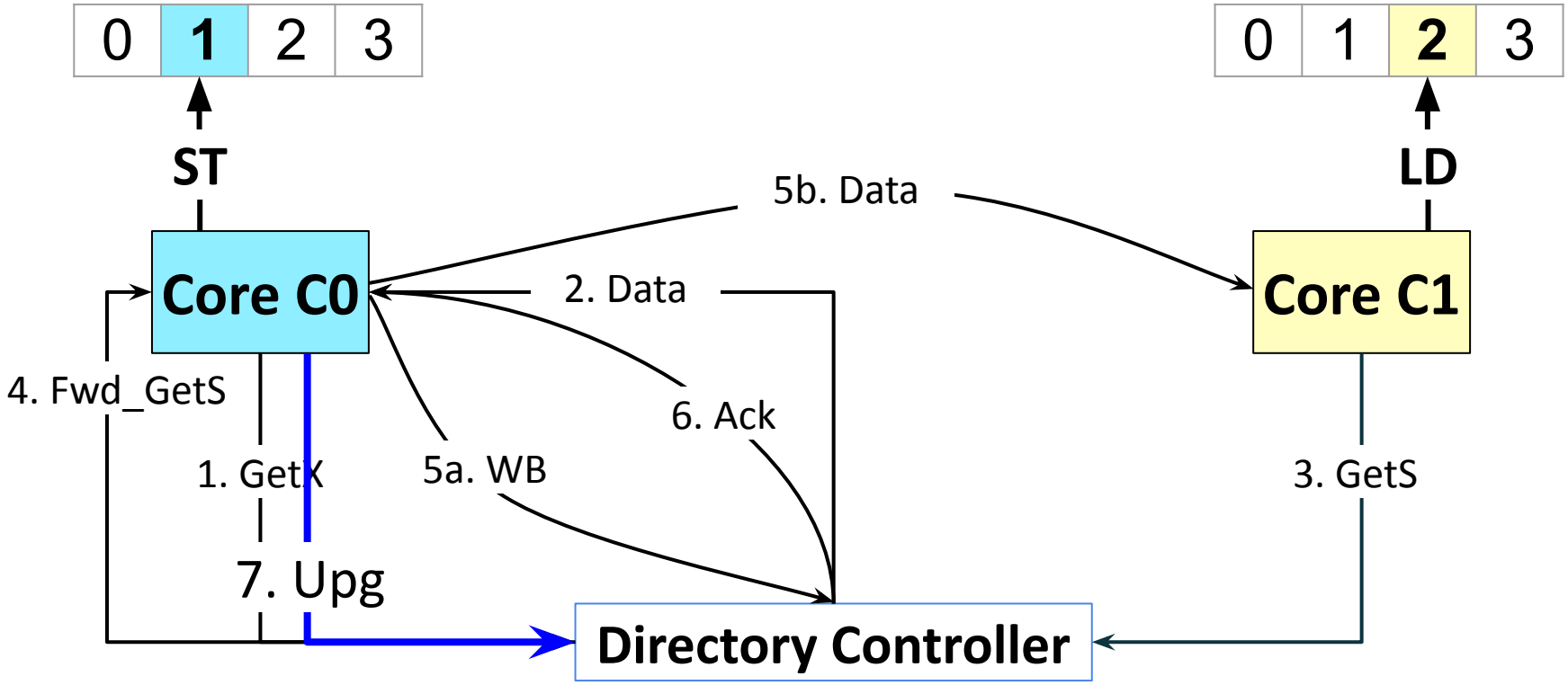
Cache Contention due to False Sharing



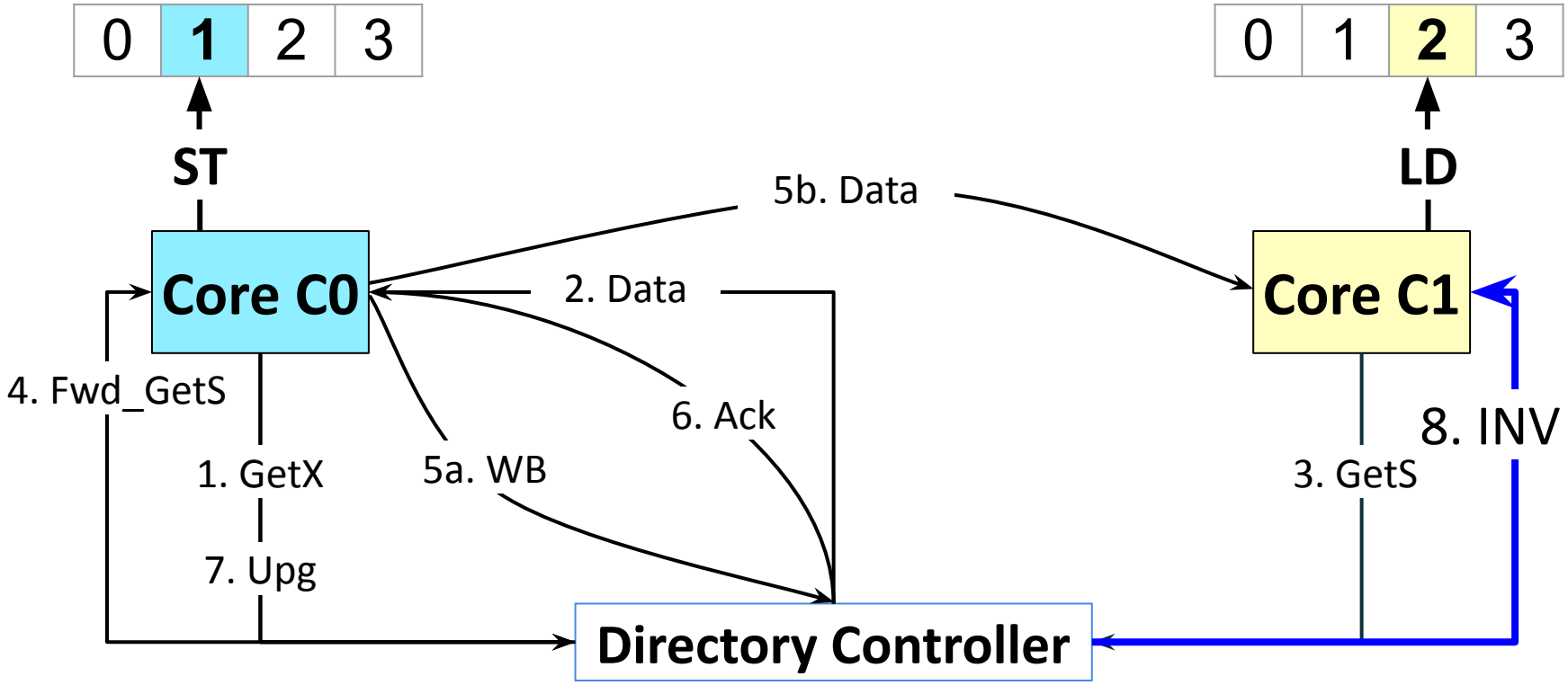
Cache Contention due to False Sharing



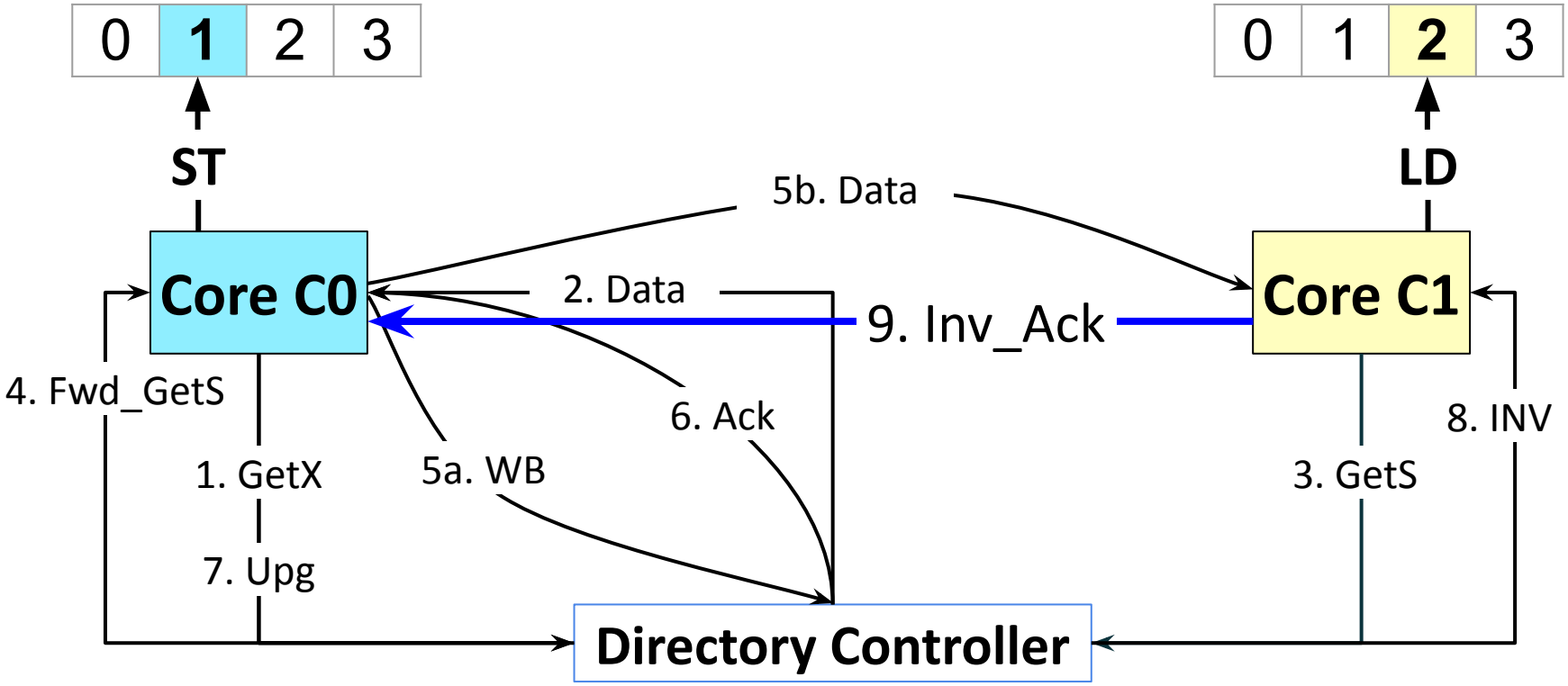
Cache Contention due to False Sharing



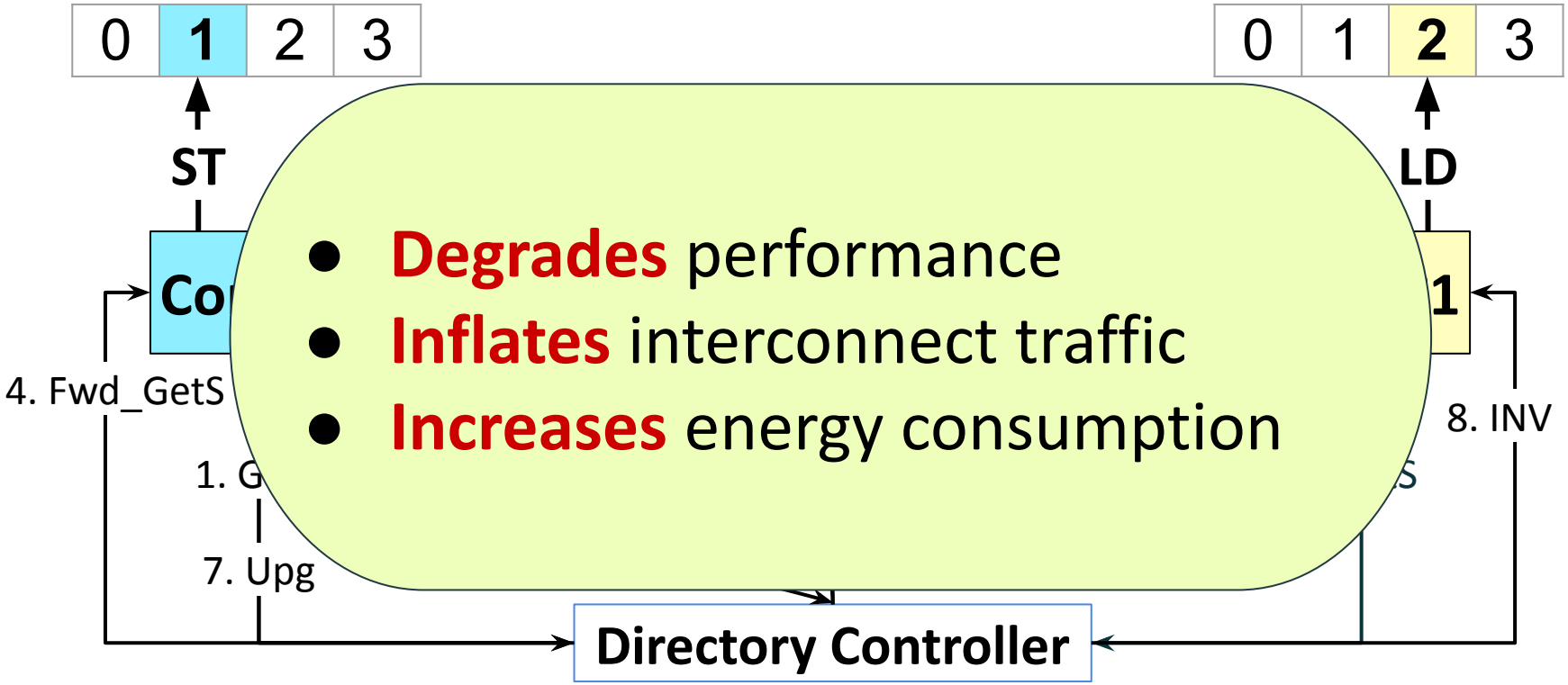
Cache Contention due to False Sharing



Cache Contention due to False Sharing

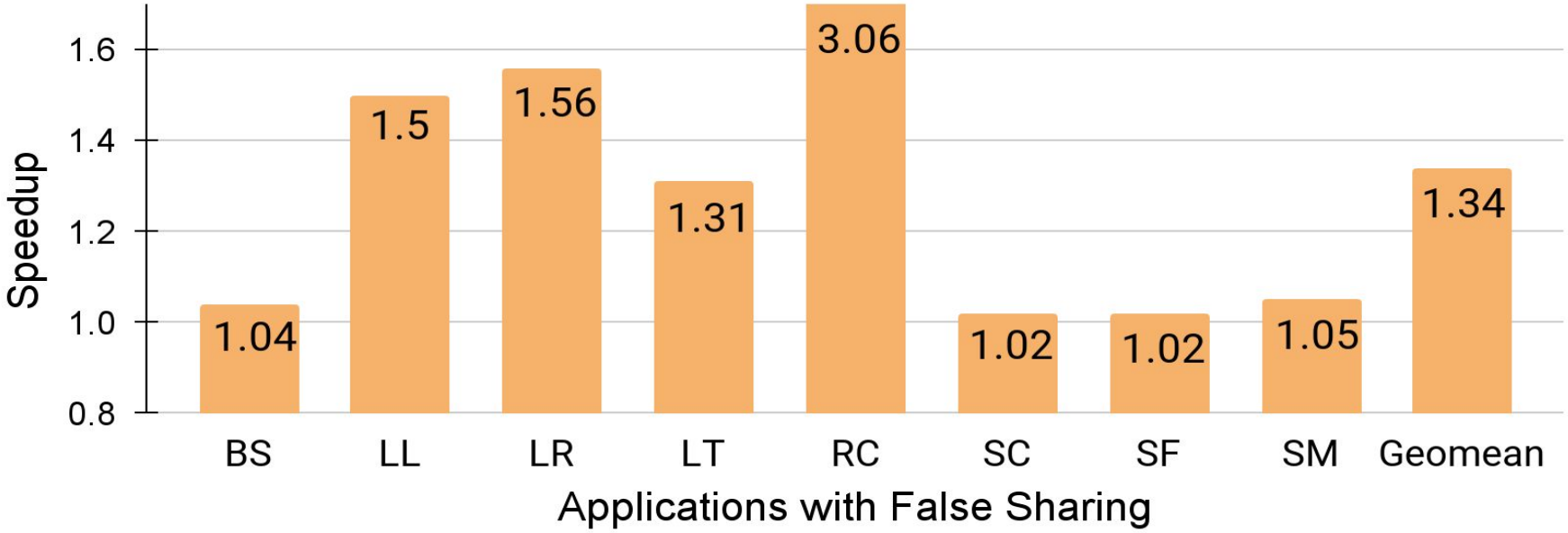


Cache Contention due to False Sharing



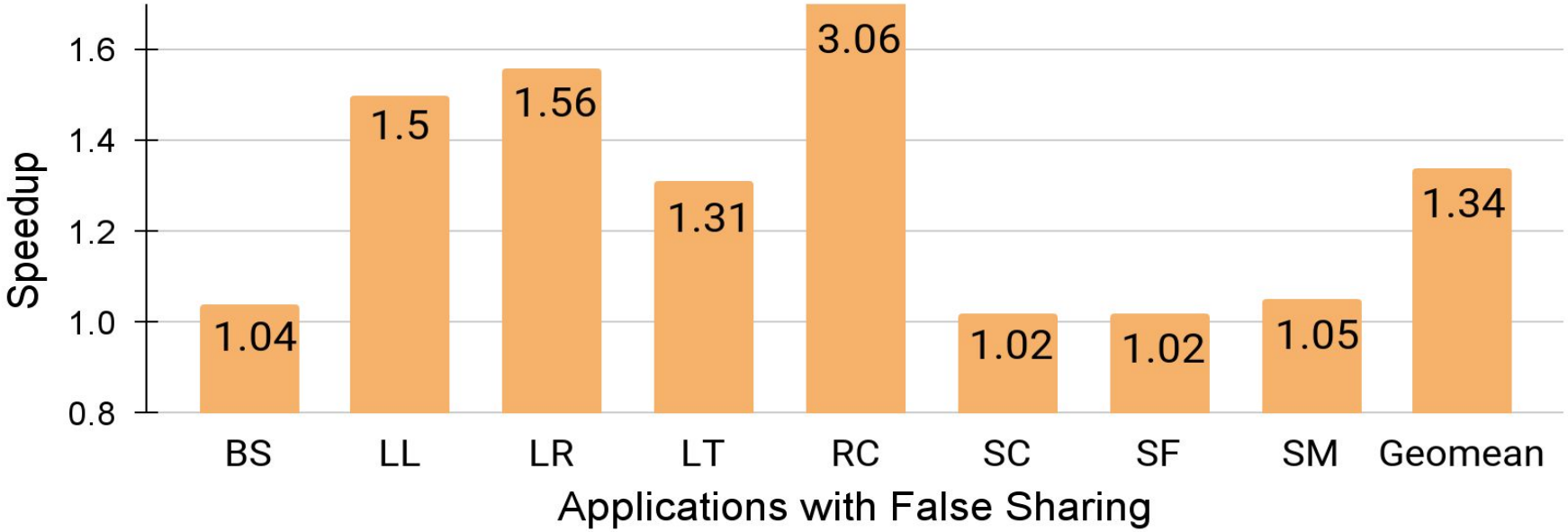
- **Degrades** performance
- **Inflates** interconnect traffic
- **Increases** energy consumption

Speedup by Manually Fixing False Sharing



Avg speedup **1.34x** and max speedup of **3.06x**

Speedup by Manually Fixing False Sharing



Savings: **84%** in interconnect traffic and **25%** in energy consumption

Challenges in False Sharing Elimination

Challenges in False Sharing Elimination

- **Heap Organization[1]**

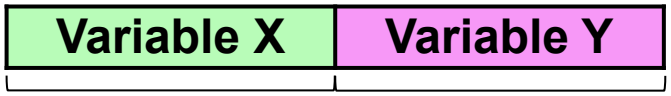
  For these applications, the memory allocator is often a bottleneck that severely limits program performance and scalability on multiprocessor systems. Previous allocators suffer from problems that include poor performance and scalability, and **heap organizations that introduce false sharing**.

1. E. D. Berger et al., “Hoard: a scalable memory allocator for multithreaded applications,” ASPLOS’00

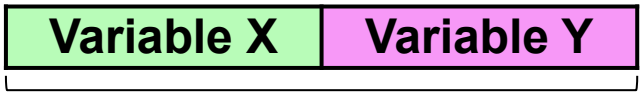
Challenges in False Sharing Elimination

- Heap Organization

- **Cache Organization**



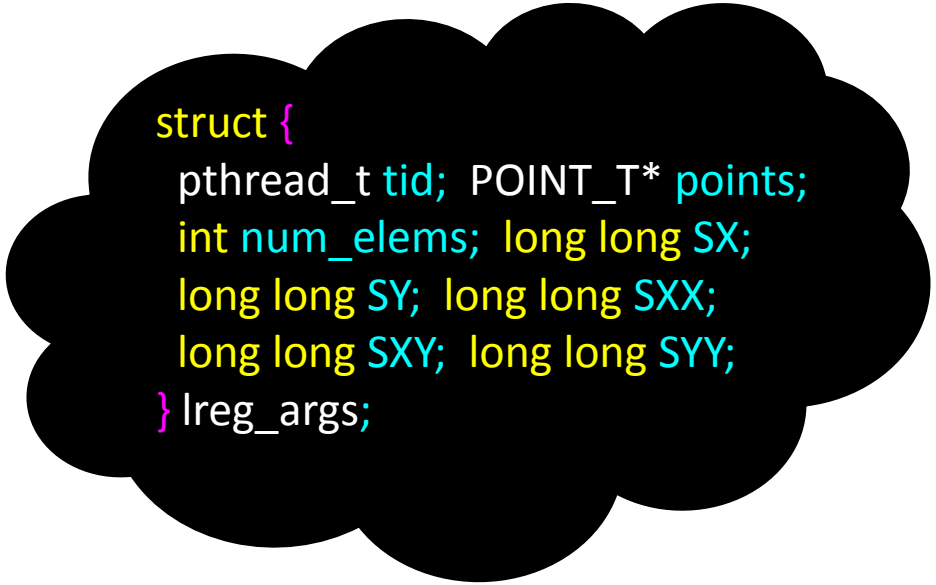
(a) No False Sharing



(b) False Sharing w/ larger cache line size

Challenges in False Sharing Elimination

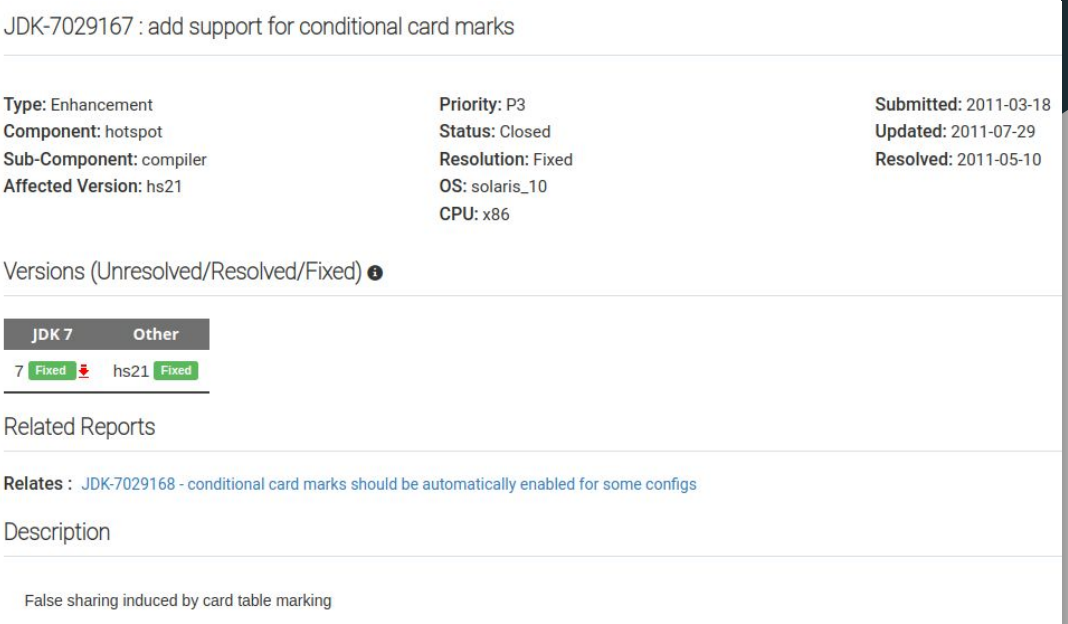
- Heap Organization
- Cache Organization
- **Compiler Optimization[1]**



1. M. Nanavati et al., “Whose Cache Line Is It Anyway? Operating System Support for Live Detection and Repair of False Sharing,” EuroSys ’13

Challenges in False Sharing Elimination

- Heap Organization
- Cache Organization
- Compiler Optimization
- **Runtime Environment[1]**



JDK-7029167 : add support for conditional card marks

Type: Enhancement
Component: hotspot
Sub-Component: compiler
Affected Version: hs21

Priority: P3
Status: Closed
Resolution: Fixed
OS: solaris_10
CPU: x86

Submitted: 2011-03-18
Updated: 2011-07-29
Resolved: 2011-05-10

Versions (Unresolved/Resolved/Fixed)

JDK 7	Other
7 Fixed	hs21 Fixed

Related Reports

Relates : JDK-7029168 - conditional card marks should be automatically enabled for some configs

Description

False sharing induced by card table marking

1. False Sharing induced by Card Table Marking: Online

Challenges in False Sharing Elimination

- Heap Organization
- Cache Organization
- Compiler Optimization
- Runtime Environment
- **Application Language**[1]

4.3.1 Challenges of Runtime Object Modification

Changing the layout of just the contended instances of a class is impossible, as all instances of a class must be **binary compatible**. Moreover, modifications must **respect inheritance**.

1. A. Eizenberg et al., “REMIX: Online Detection and Repair of Cache Contention for the JVM,” PLDI’16

Shortcomings of the existing approaches

Shortcomings of the existing approaches

- **Inflation of memory footprint**

Shortcomings of the existing approaches

- Inflation of memory footprint
- **Introduction of additional instructions**

Shortcomings of the existing approaches

- Inflation of memory footprint
- Introduction of additional instructions
- **Extending support to every language**

Shortcomings of the existing approaches

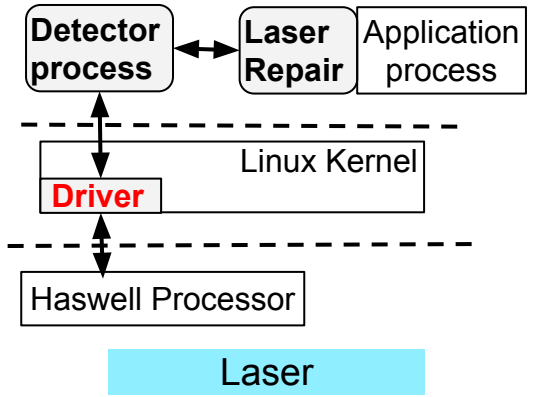
- Inflation of memory footprint
- Introduction of additional instructions
- Extending support to every language
- **Access to application source code**[1]

> 3 Design
>
> In this section, we describe the design of Huron, our hybrid in-house/in-production false sharing detection and repair system. Huron first detects and repairs false sharing in house **using developer test cases**.

1. T. A. Khan et al., “Huron: hybrid false sharing detection and repair,” PLDI’19

Shortcomings of the existing approaches

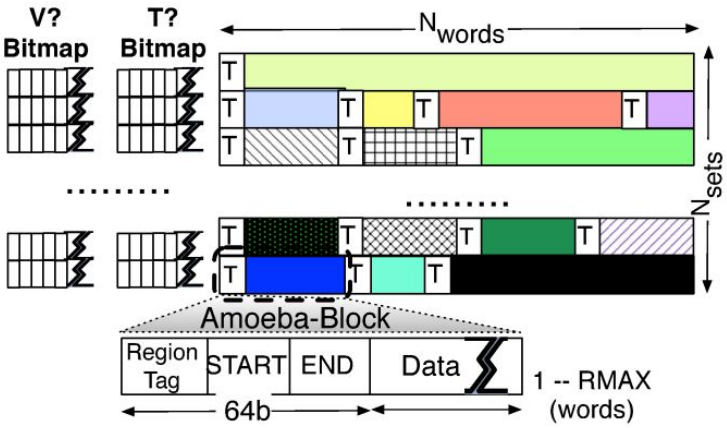
- Inflation of memory footprint
- Introduction of additional instructions
- Extending support to every language
- Access to application source code
- **OS and Kernel modification[1,2]**



1. L. Luo et al., “LASER: Light, Accurate Sharing dEtection and Repair,” HPCA’16
2. T. Liu et al., “Cheetah: Detecting False Sharing Efficiently and Effectively,” CGO’16

Shortcomings of the existing approaches

- Inflation of memory footprint
- Introduction of additional instructions
- Extending support to every language
- Access to application source code
- OS and Kernel modification



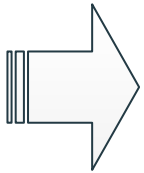
Amoeba Cache Architecture[1]

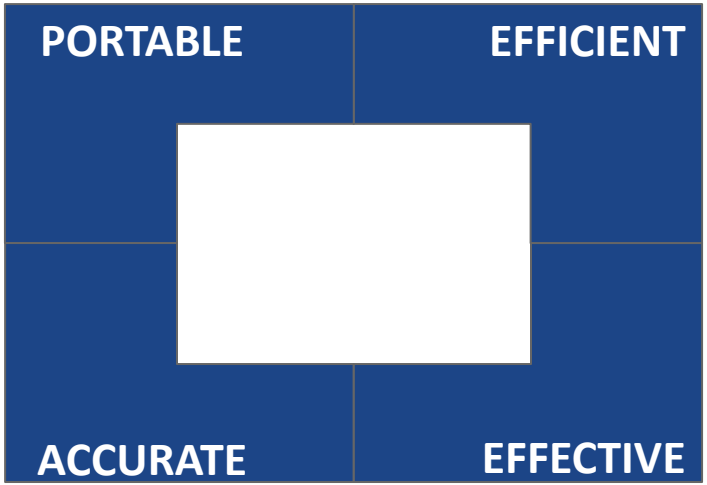
- **Cache organization modification**

1. S. Kumar et al., “Amoeba-Cache: Adaptive Blocks for Eliminating Waste in the Memory Hierarchy,” MICRO’12

Shortcomings of the existing approaches

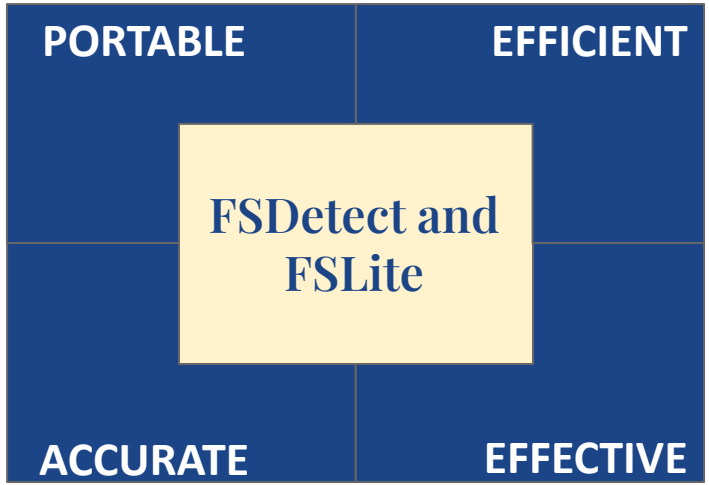
- Inflation of memory footprint
- Introduction of additional instructions
- Extending support to every language
- Access to application source code
- OS and Kernel modification
- Cache organization modification





Shortcomings of the existing approaches

- Inflation of memory footprint
- Introduction of additional instructions
- Extending support to every language
- Access to application source code
- OS and Kernel modification
- Cache organization modification



Key Insights

FSDetect

- Tracks the sharing patterns of block
- Identifies the impactful instance of false sharing

Key Insights

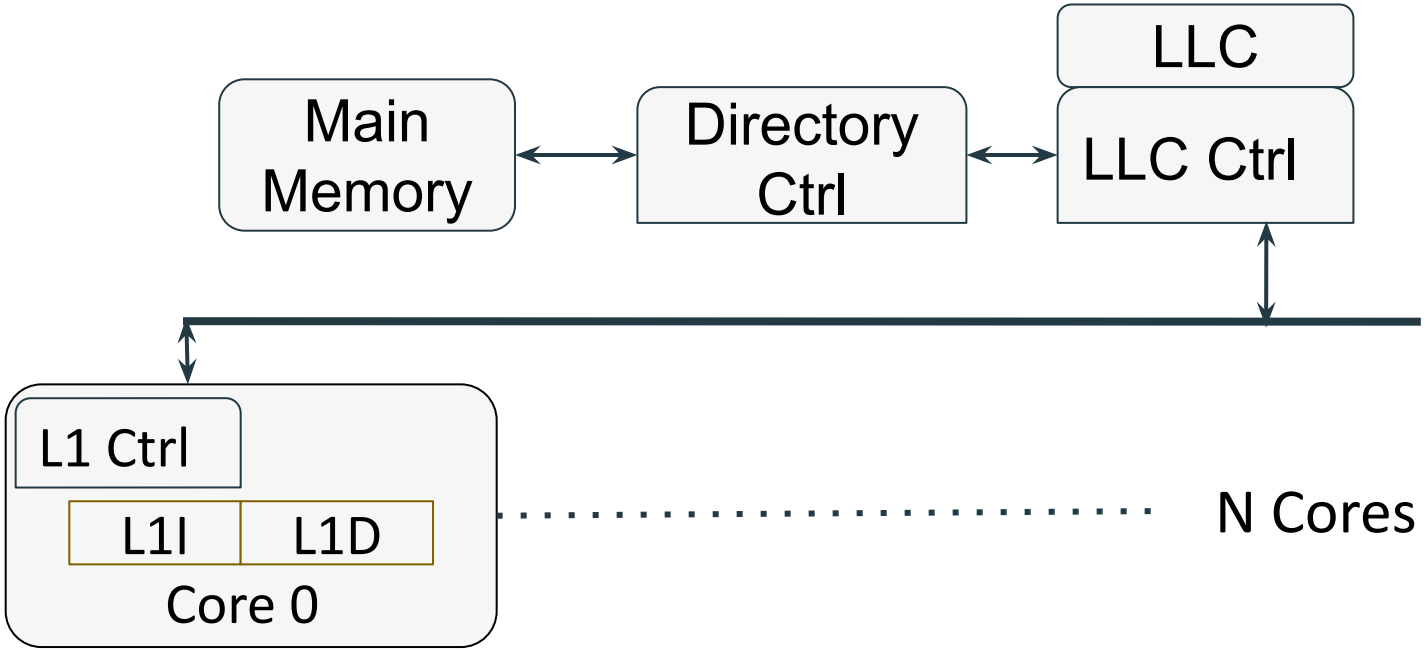
FSDetect

- Tracks the sharing patterns of block
- Identifies the impactful instance of false sharing

FSLite

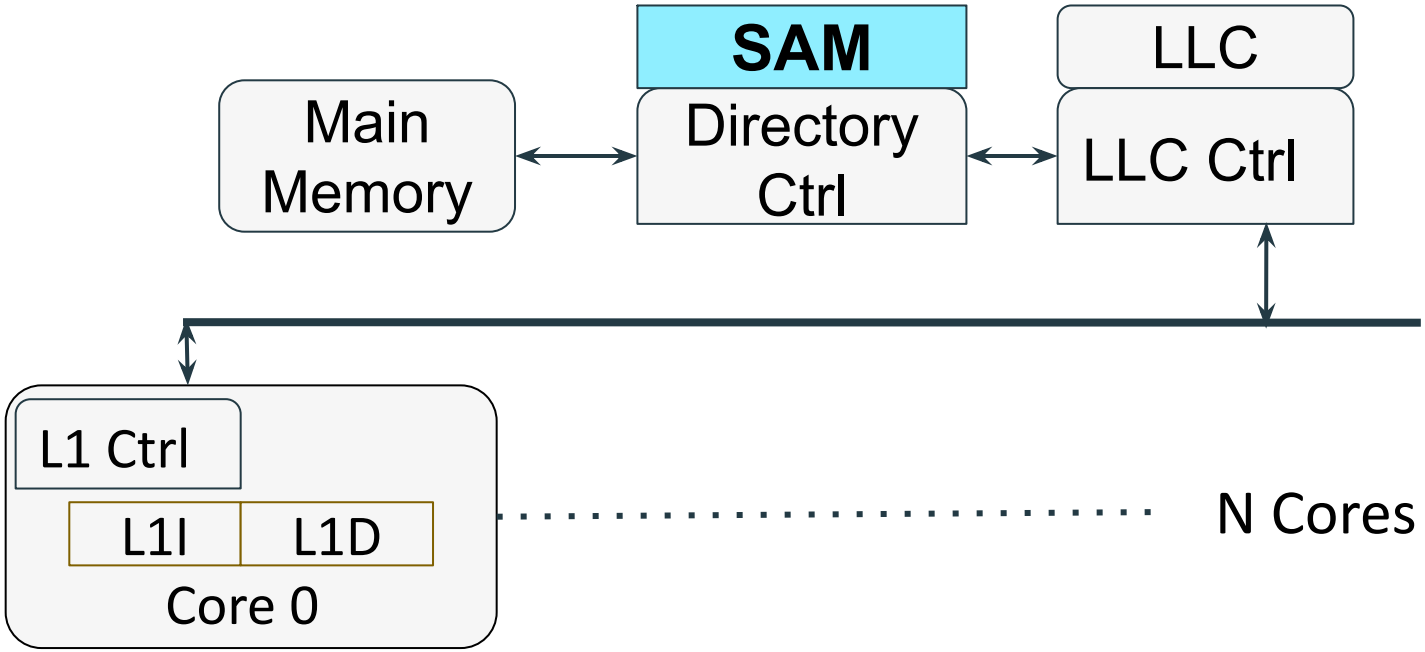
- Eliminates false sharing by **privatizing** the blocks
- Allows multiple writers for the disjoint bytes

Baseline Architecture



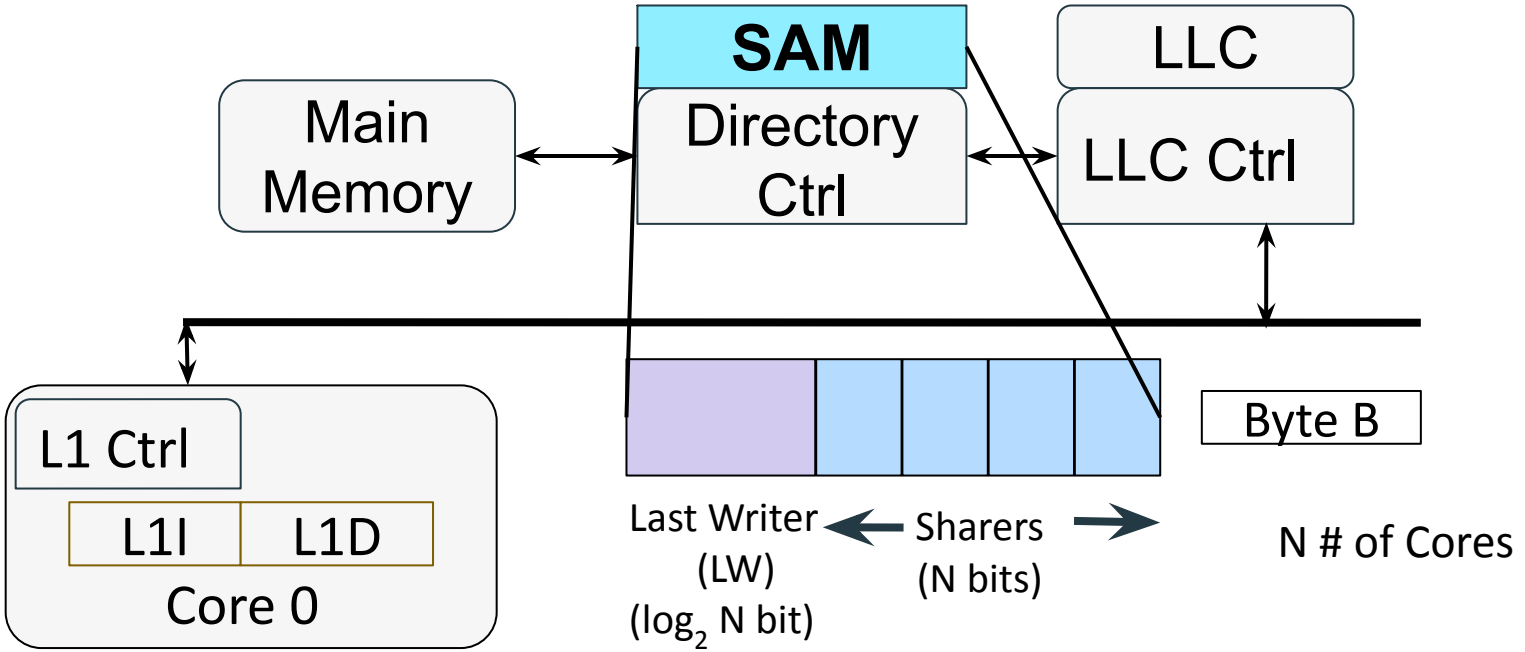
Overview of Architectural Modification

Shared Access Metadata (SAM) to track the access history



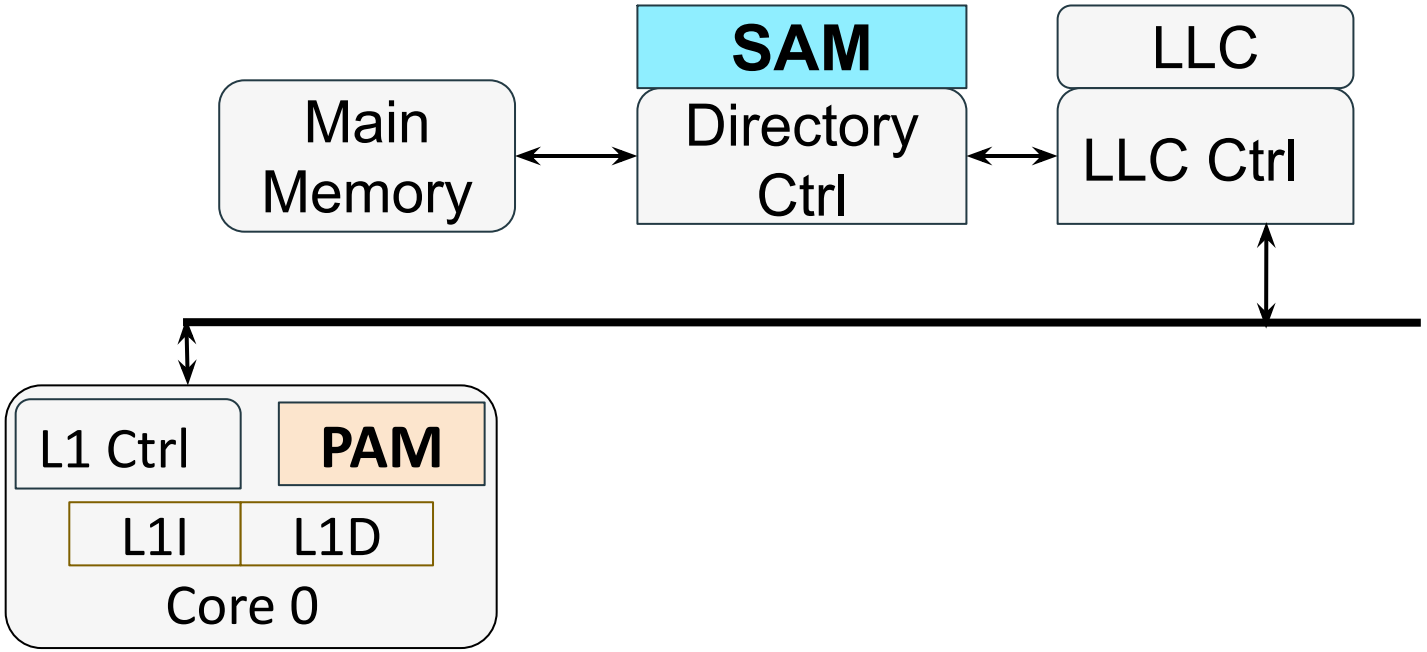
Overview of Architectural Modification

Shared Access Metadata (SAM) to track the access history



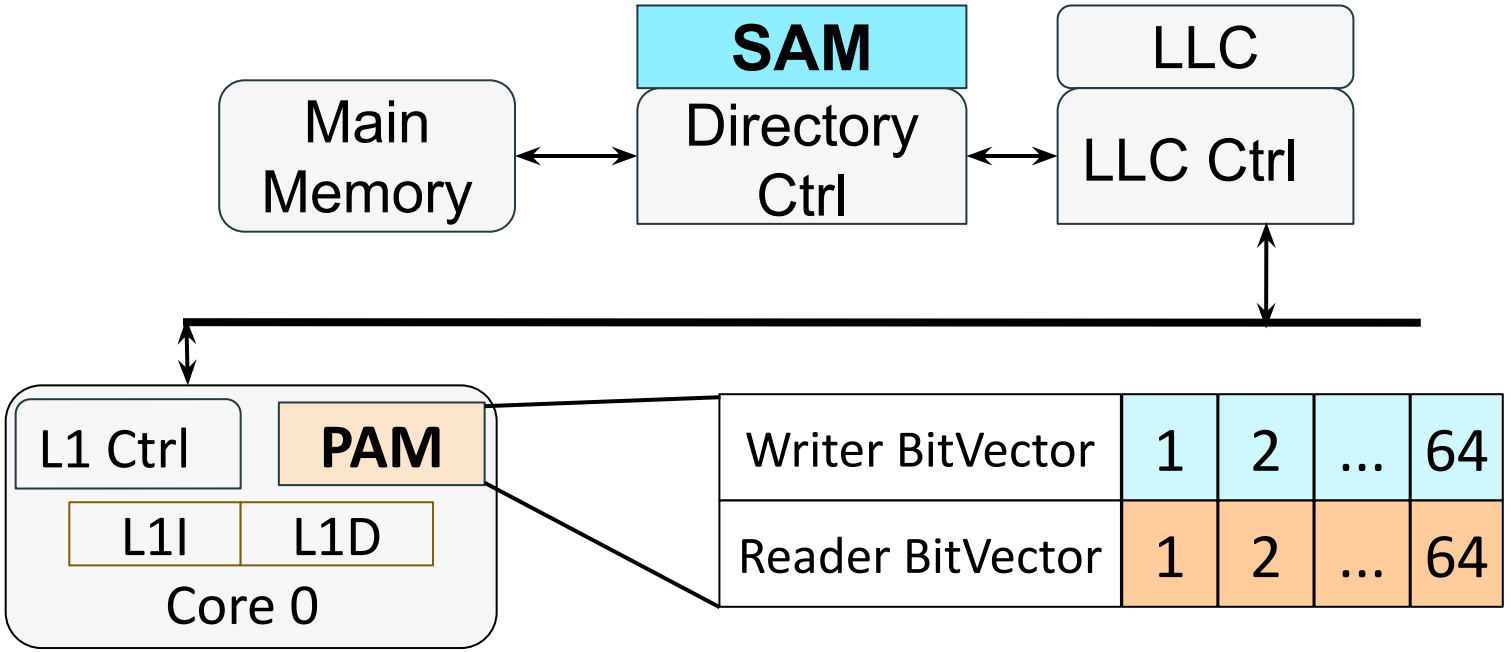
Overview of Architectural Modification

Private Access Metadata (PAM) to track the local access

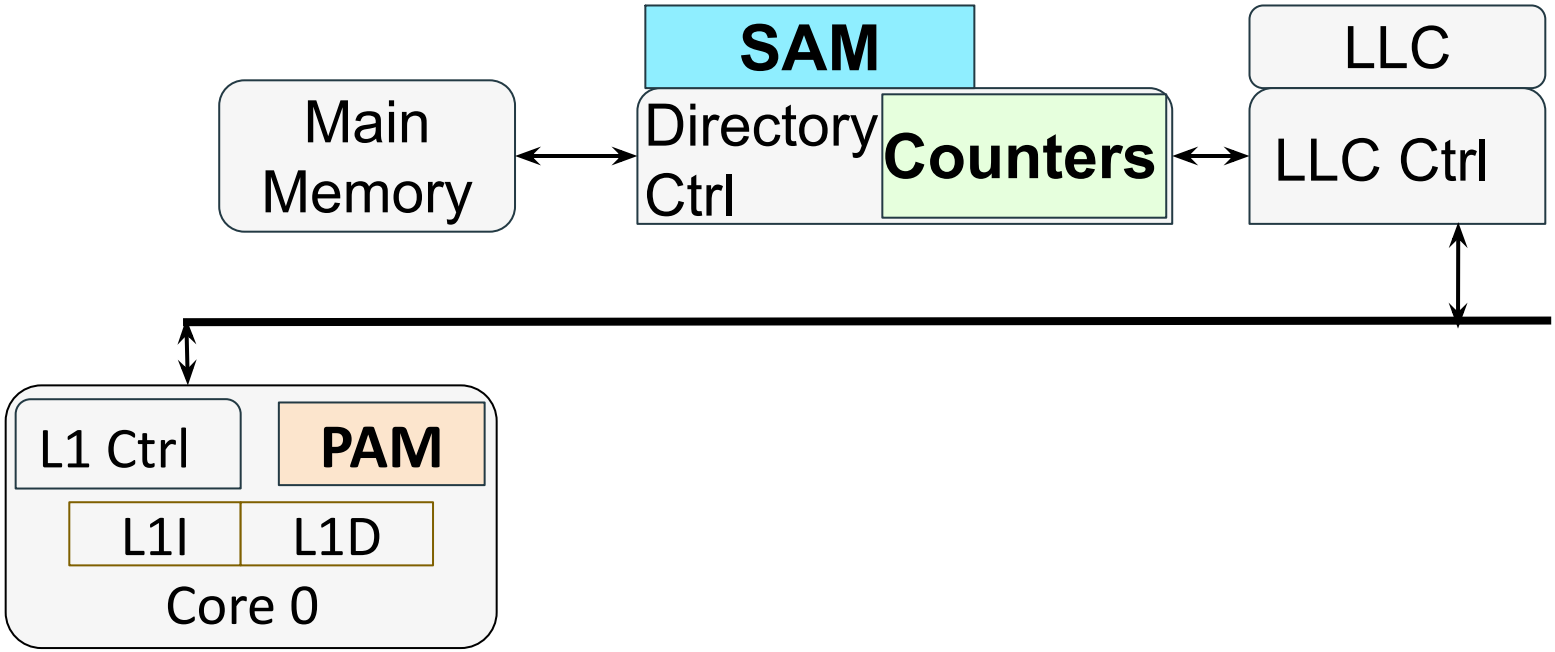


Overview of Architectural Modification

Private Access Metadata (PAM) to track the local access



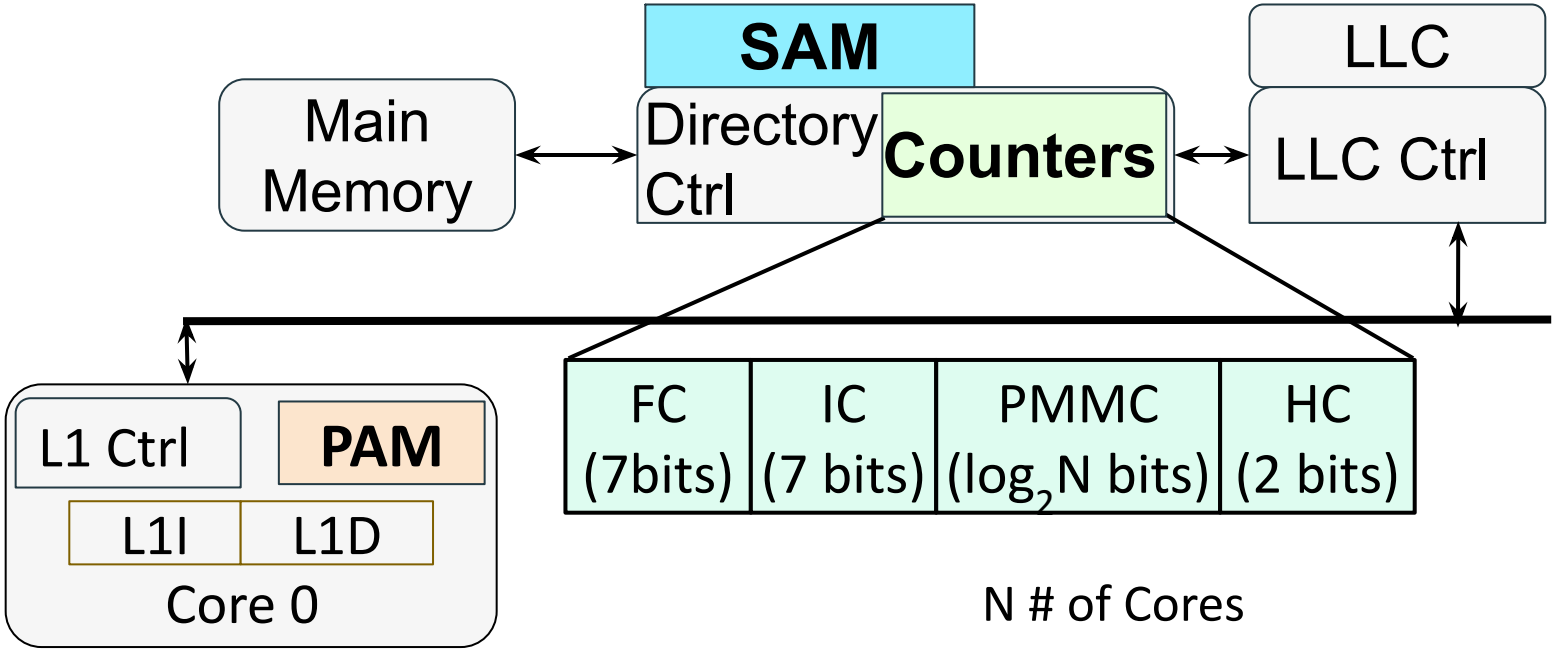
Overview of Architectural Modification



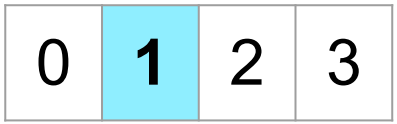
Overview of Architectural Modification

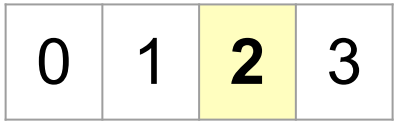
Fetch (FC) and Invalidation (IC) to filter the impactful instance

Pending Metadata Message (PMMC) to track the inflight metadata



Detecting False Sharing in FSDetect





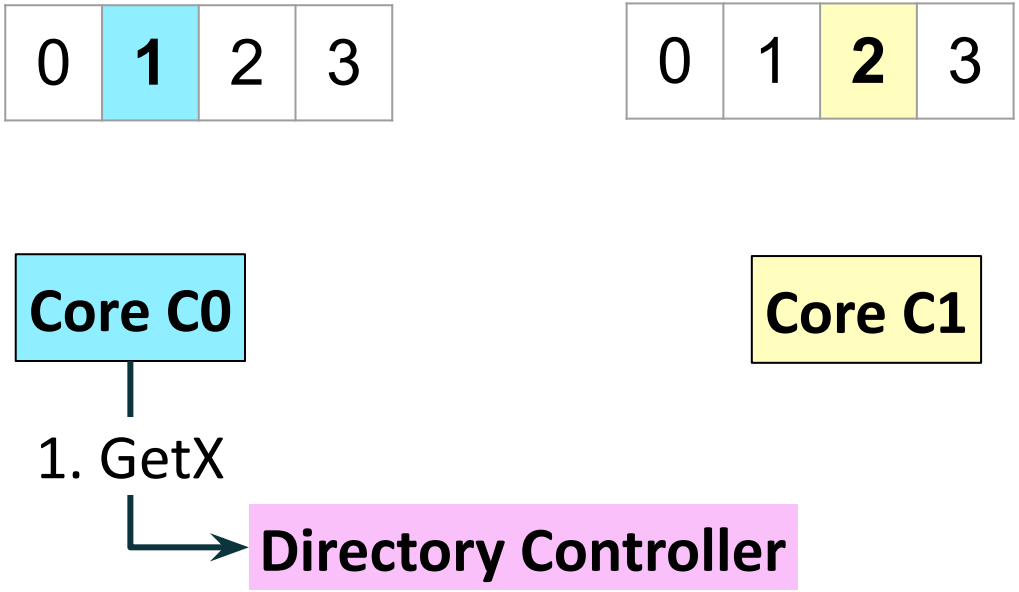
Core C0

Core C1

Directory Controller

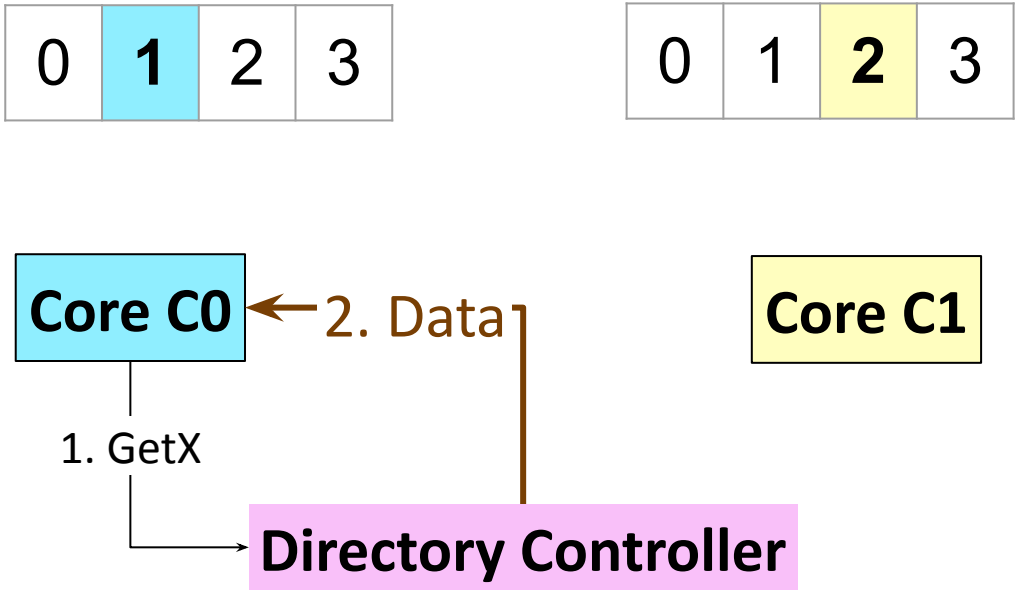
Block B0 (I)

Detecting False Sharing in FSDetect

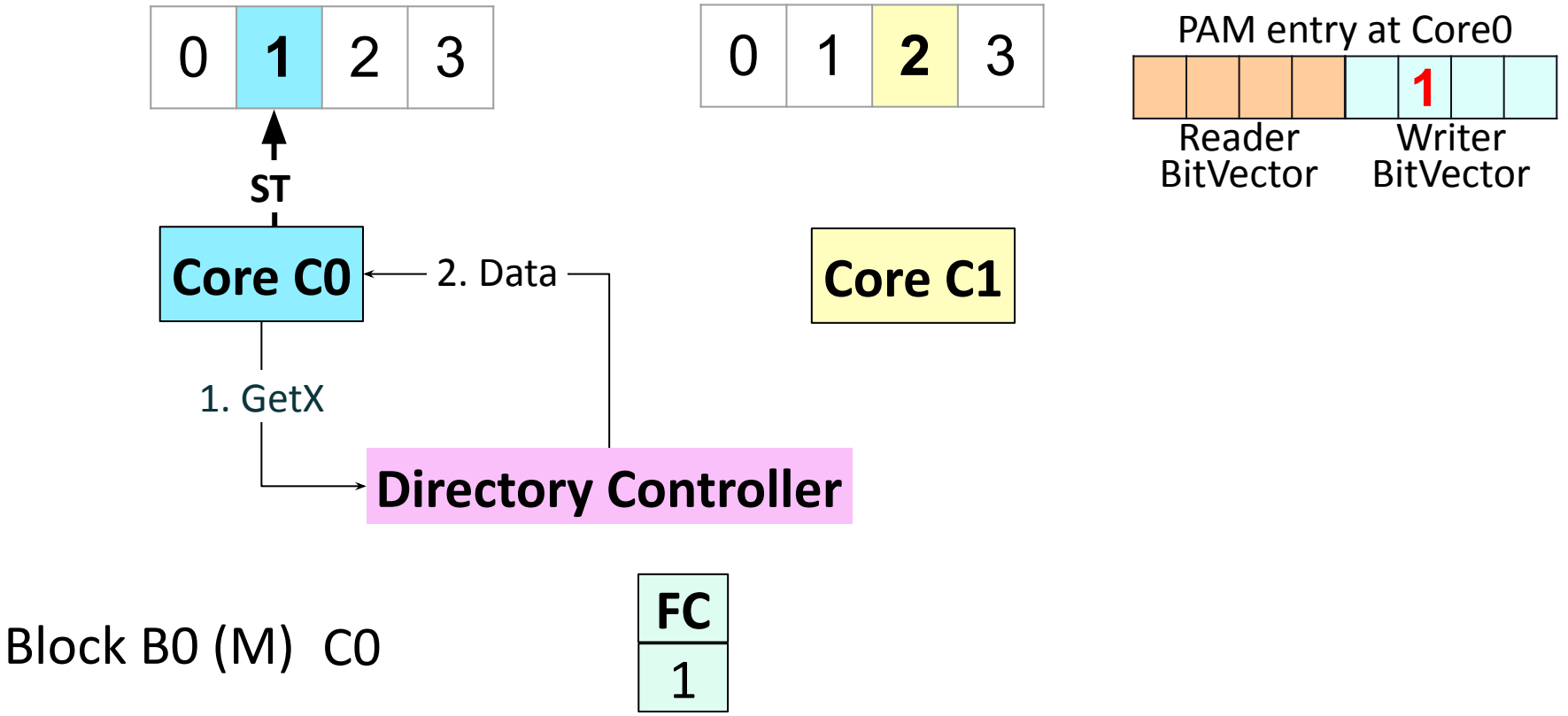


Block B0 (I)

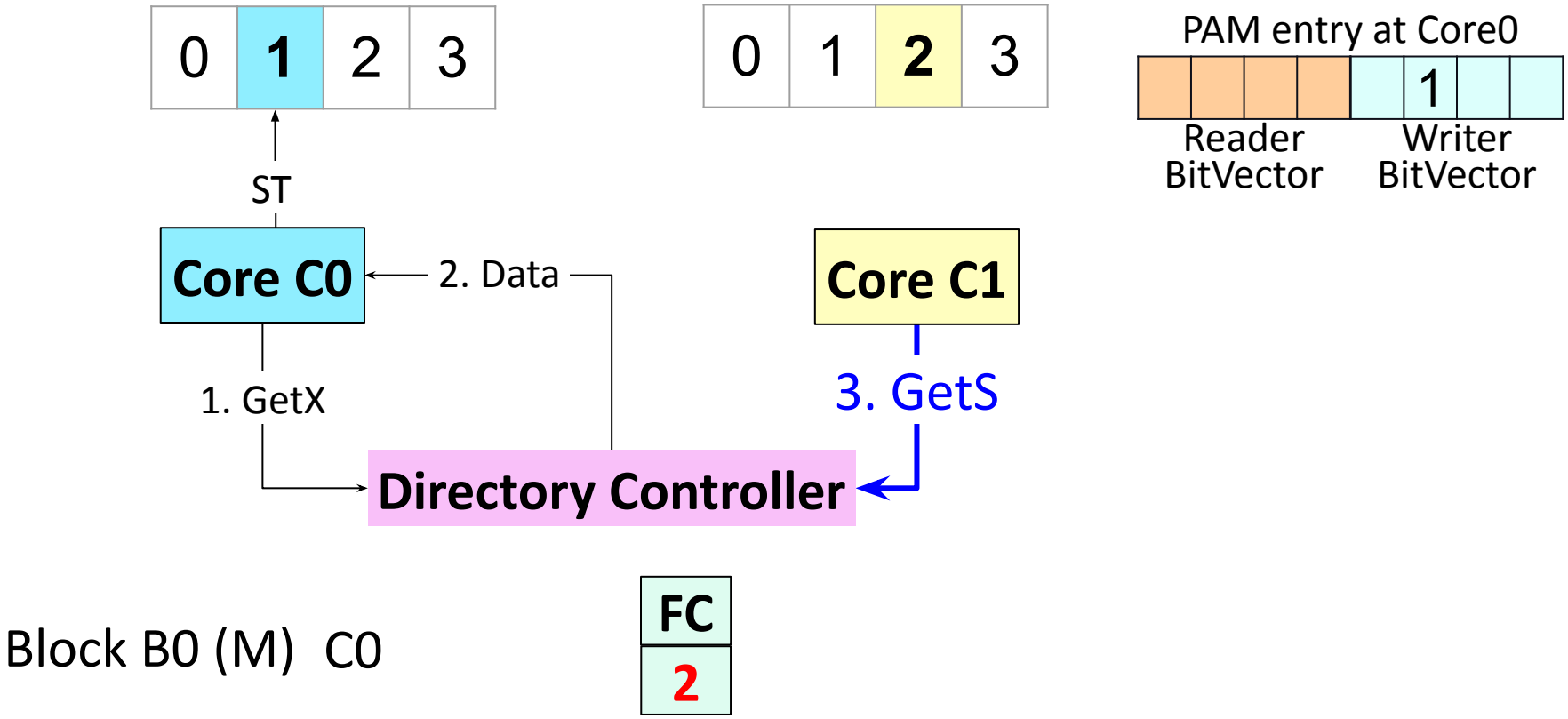
Detecting False Sharing in FSDetect



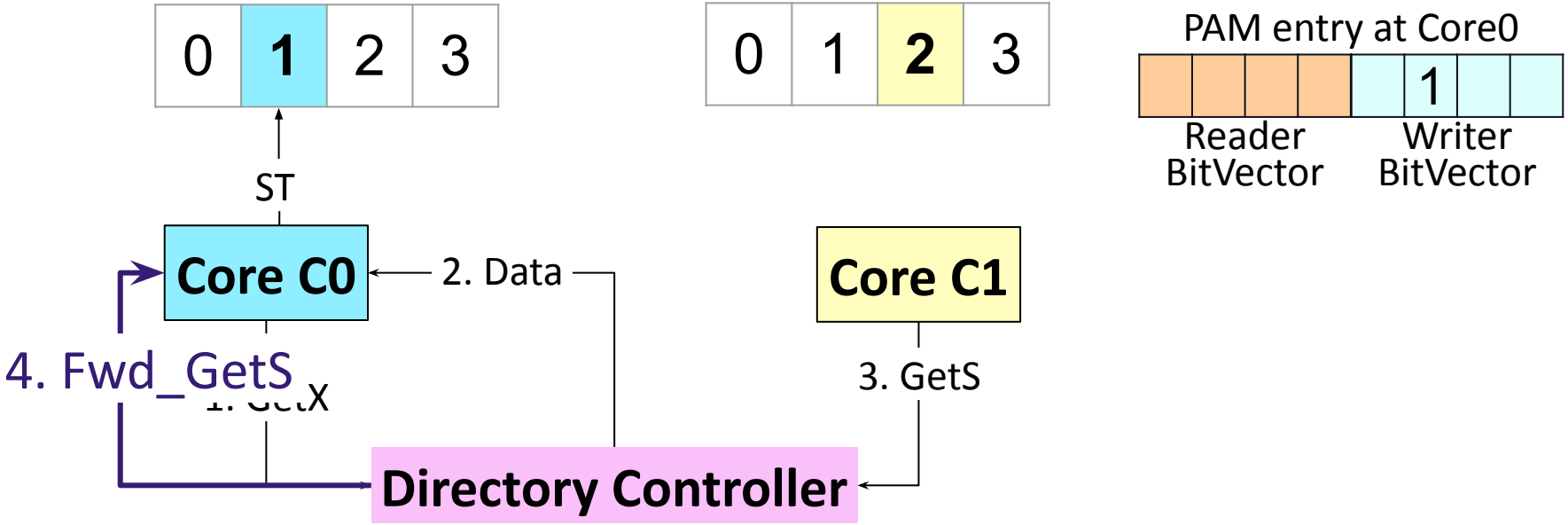
Detecting False Sharing in FSDetect



Detecting False Sharing in FSDetect



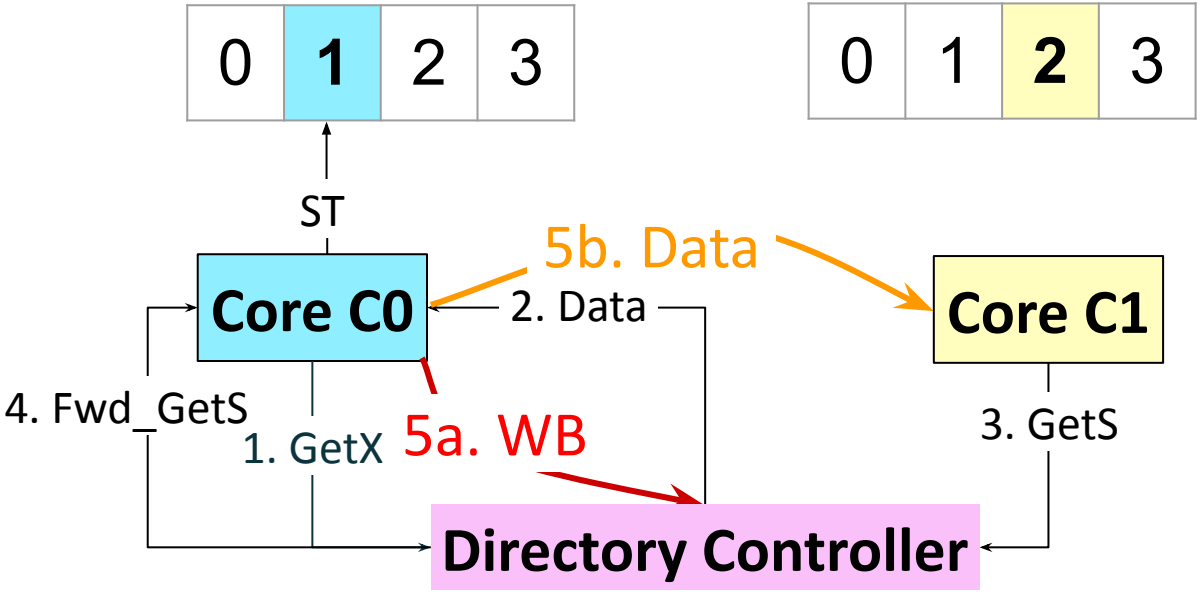
Detecting False Sharing in FSDetect

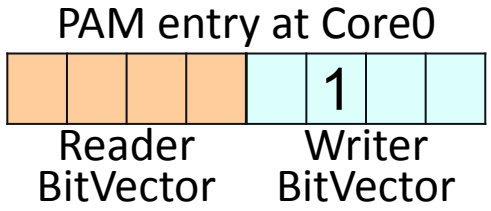


Block B0 (S) C0 **C1**

FC	IC	PMMC
2	1	1

Detecting False Sharing in FSDetect

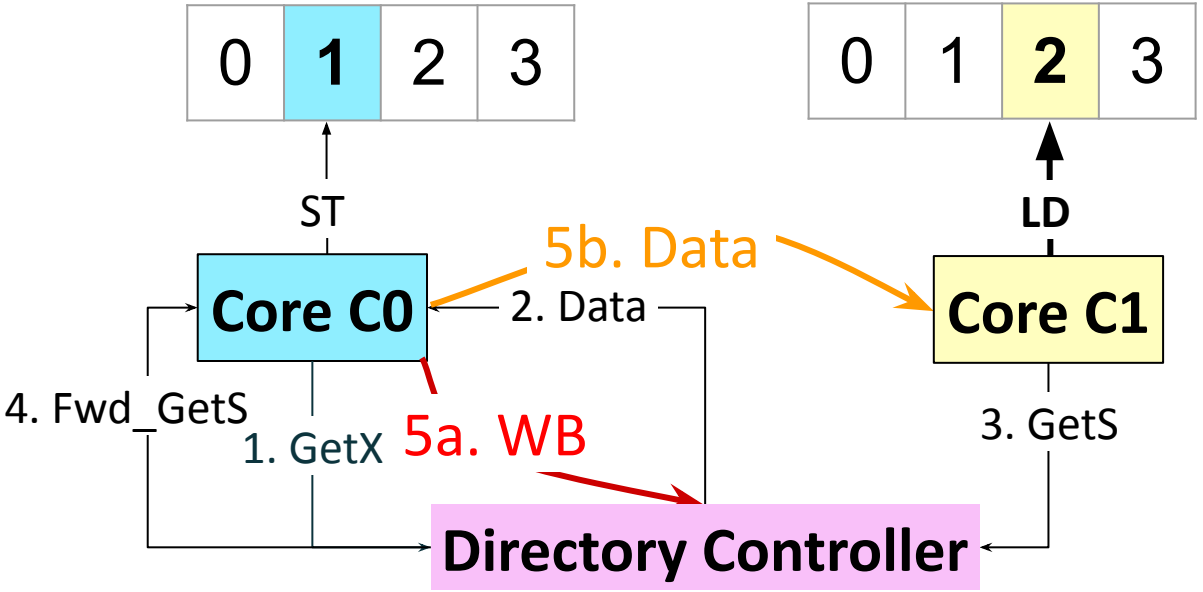


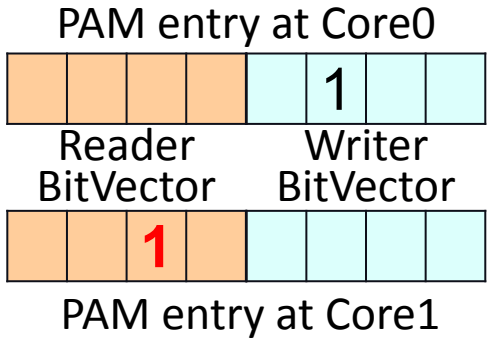


Block B0 (S) C0 C1

FC	IC	PMMC
2	1	1

Detecting False Sharing in FSDetect

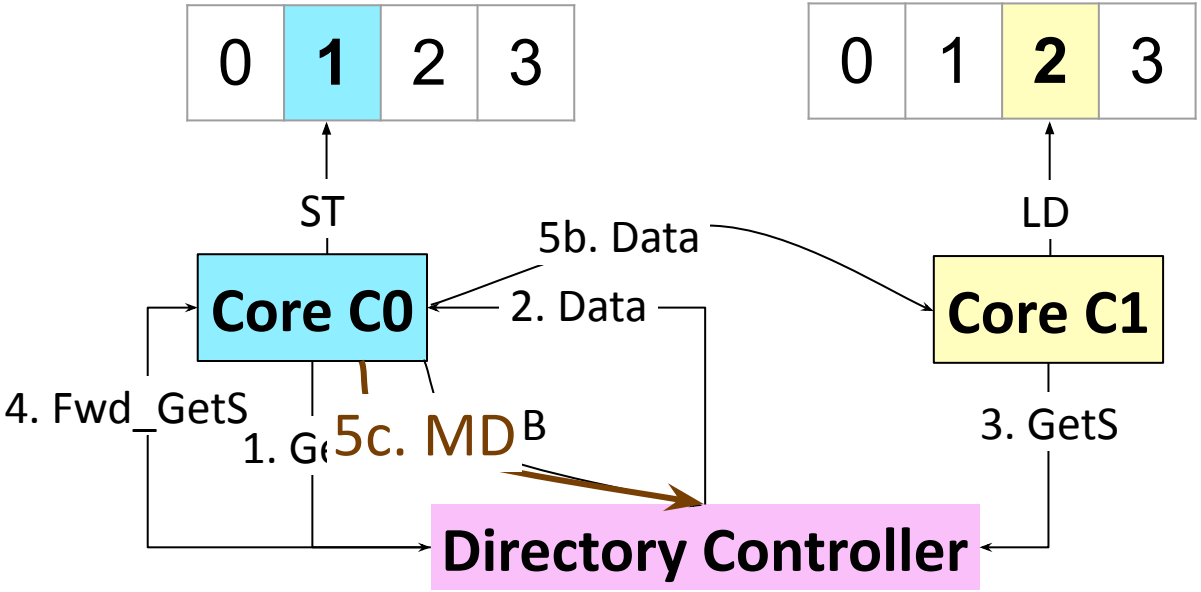


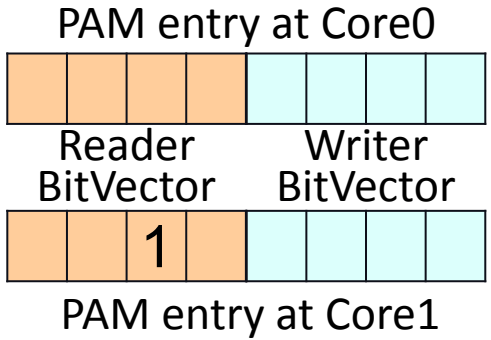


Block B0 (S) C0 C1

FC	IC	PMMC
2	1	1

Detecting False Sharing in FSDetect

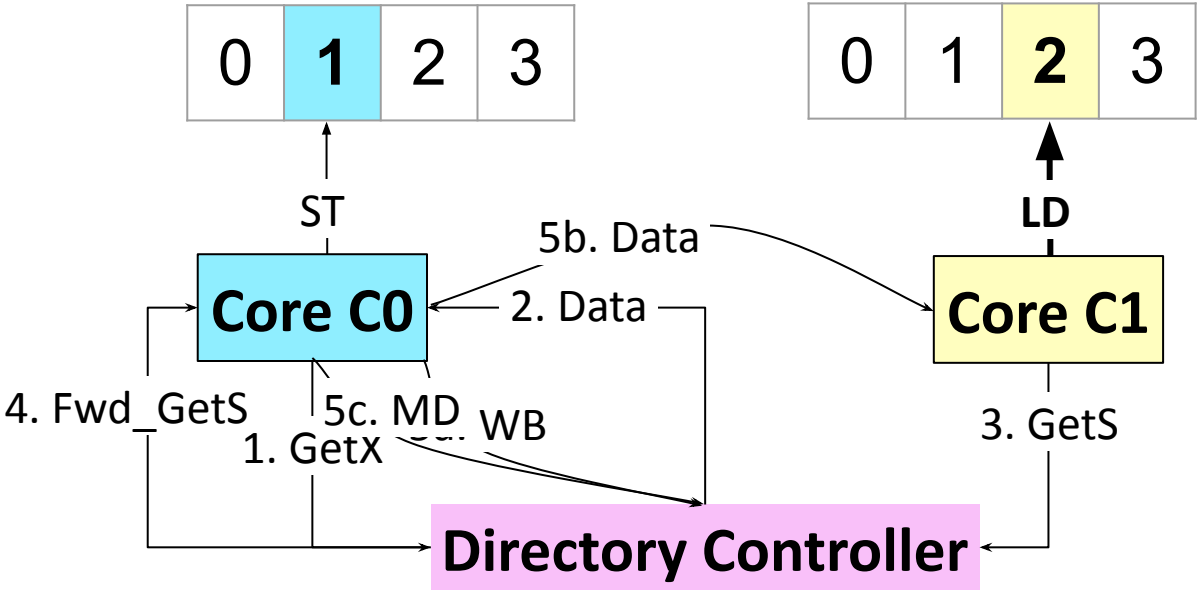


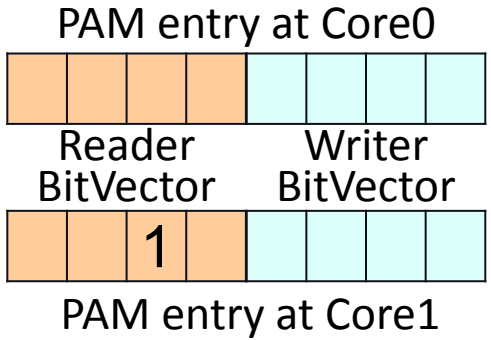


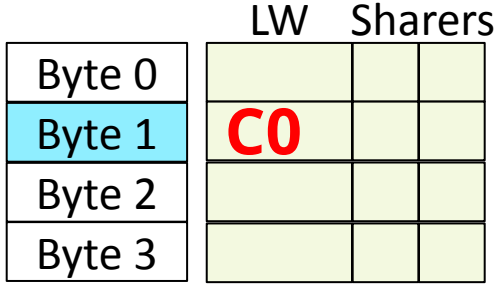
Block B0 (S) C0 C1

FC	IC	PMMC
2	1	1

Detecting False Sharing in FSDetect





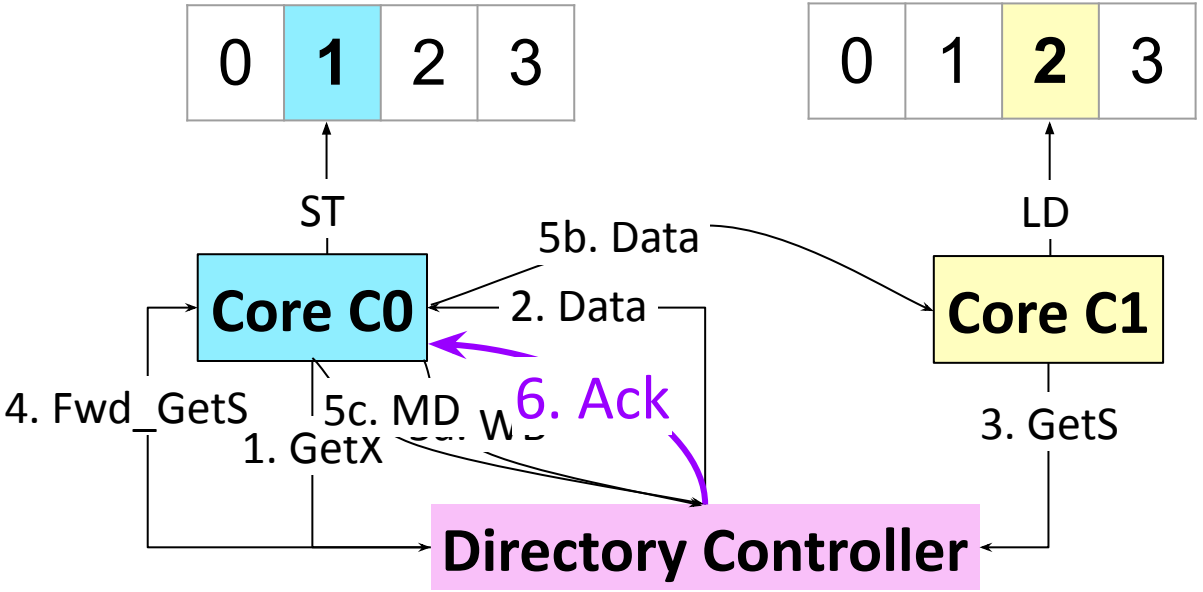


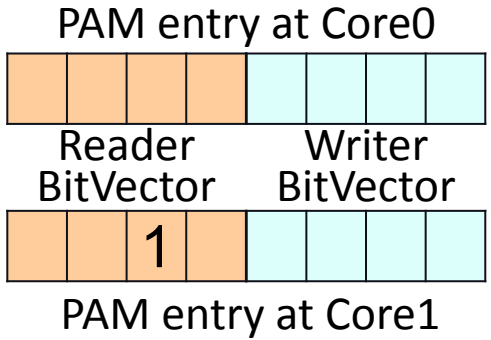
Block B0 (S) C0 C1

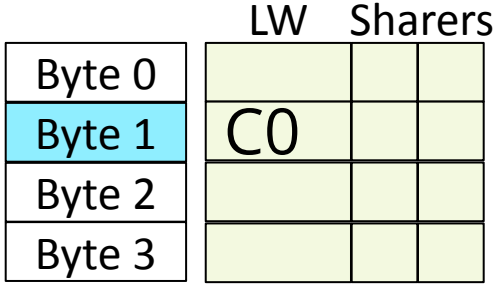
FC	IC	PMMC
2	1	0

SAM table entry for a **Block B0**

Detecting False Sharing in FSDetect





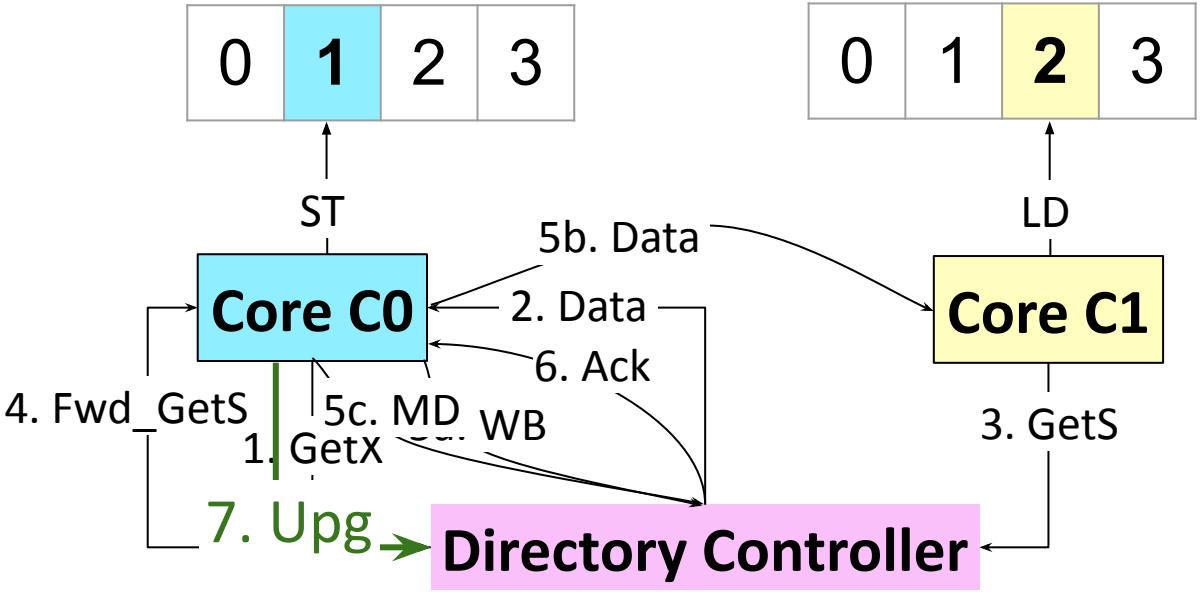


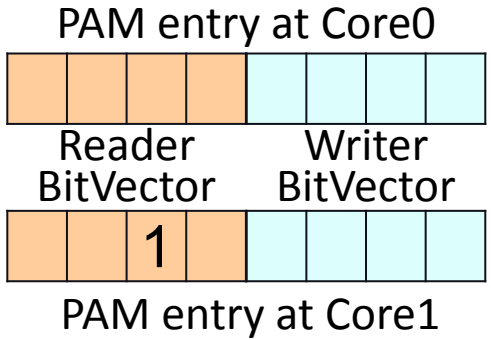
Block B0 (S) C0 C1

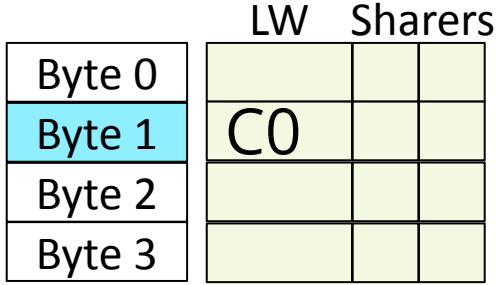
FC	IC	PMMC
2	1	0

SAM table entry for a **Block B0**

Detecting False Sharing in FSDetect





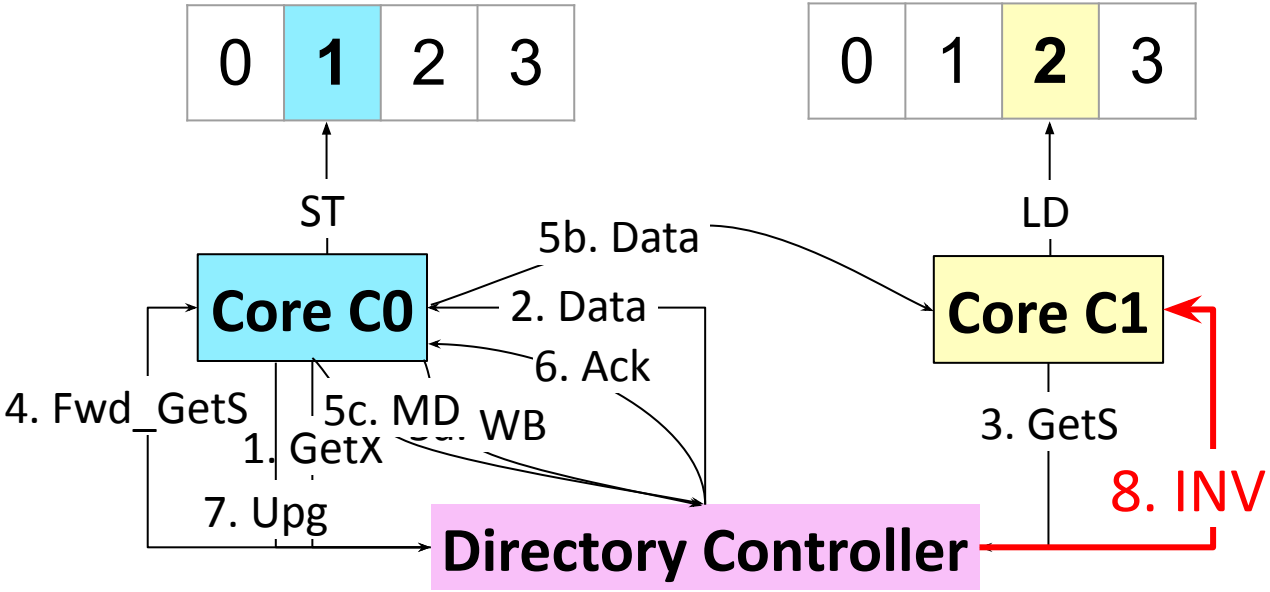


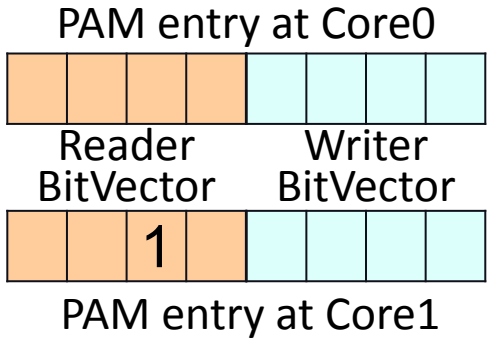
Block B0 (M) C0 ~~C1~~

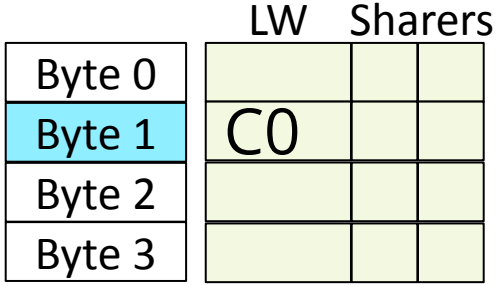
FC	IC	PMMC
3	1	1

SAM table entry for a **Block B0**

Detecting False Sharing in FSDetect





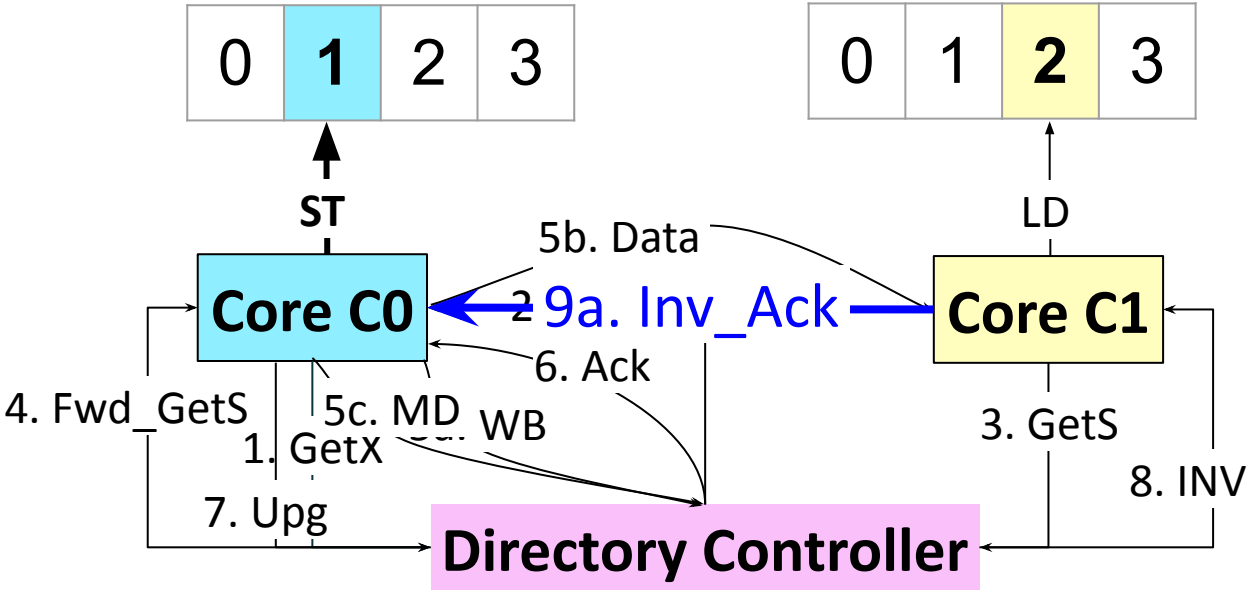


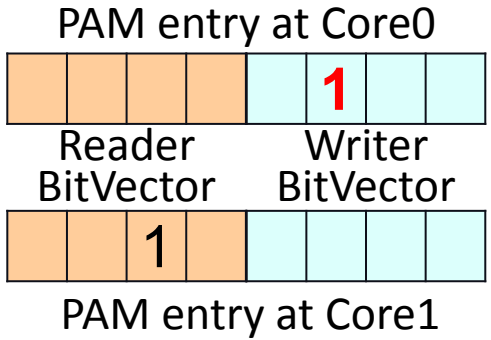
Block B0 (M) C0

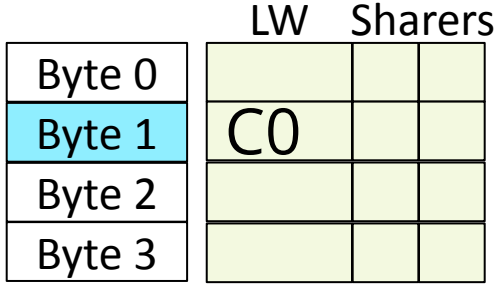
FC	IC	PMMC
3	2	1

SAM table entry for a **Block B0**

Detecting False Sharing in FSDetect





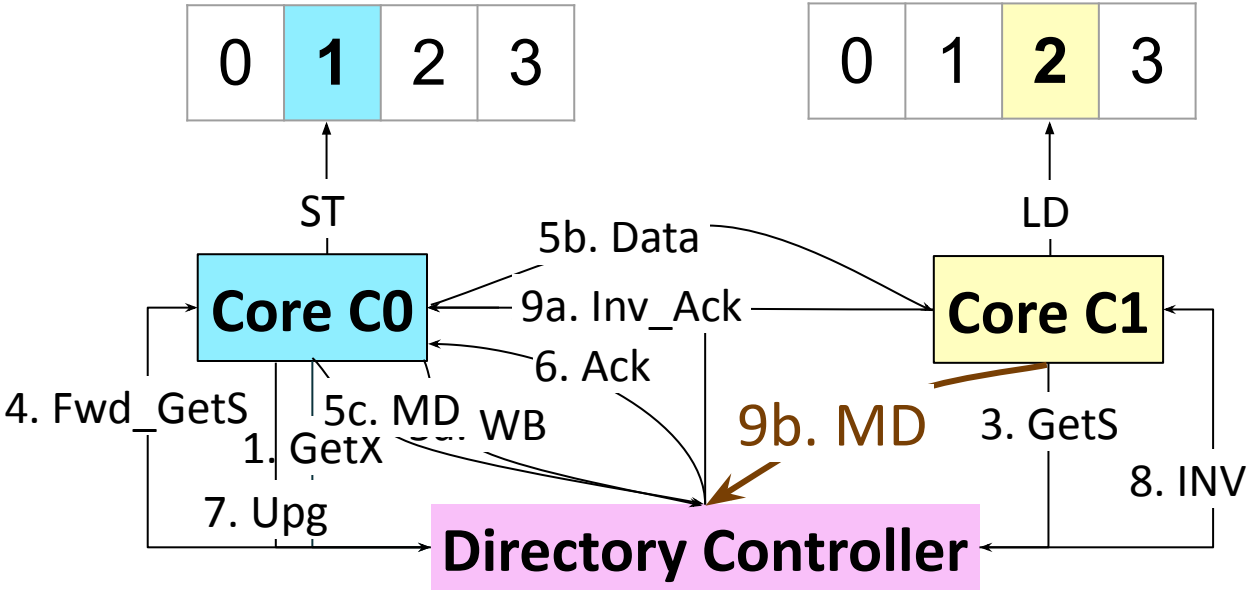


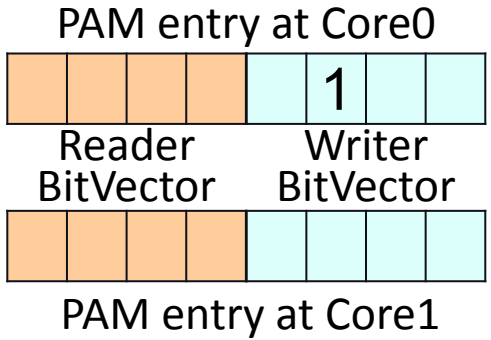
Block B0 (M) C0

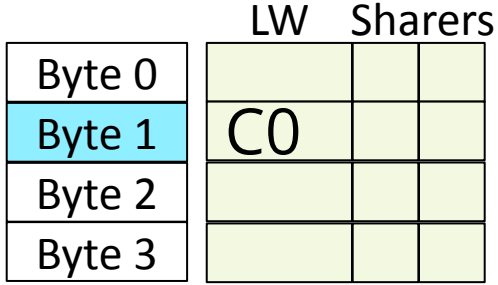
FC	IC	PMMC
3	2	1

SAM table entry for a **Block B0**

Detecting False Sharing in FSDetect





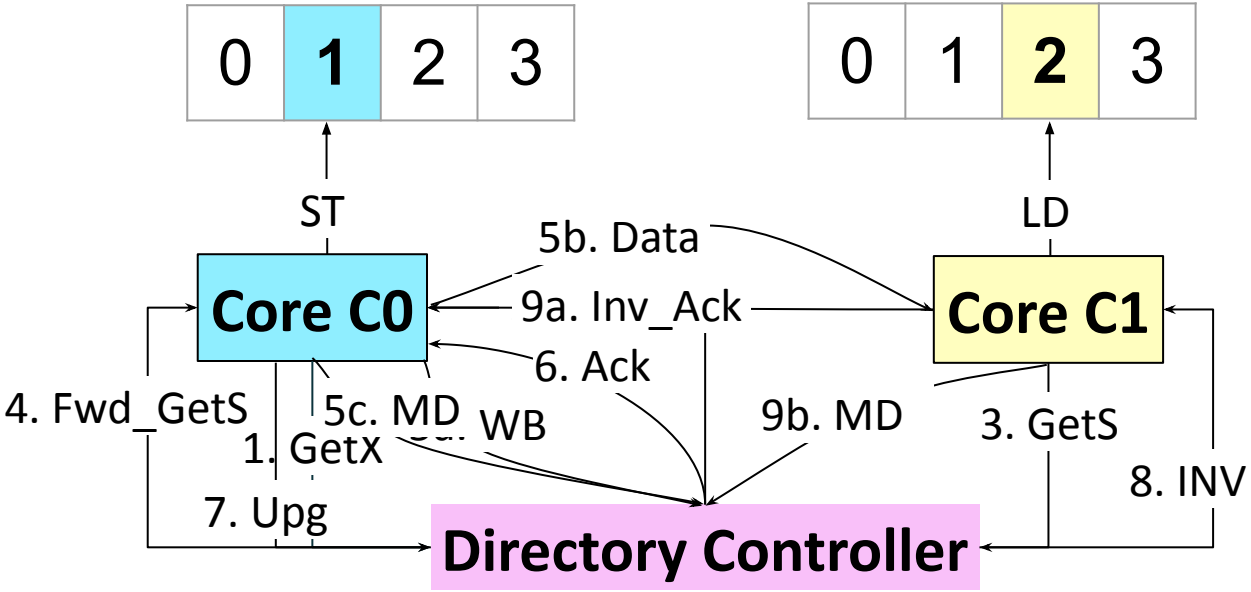


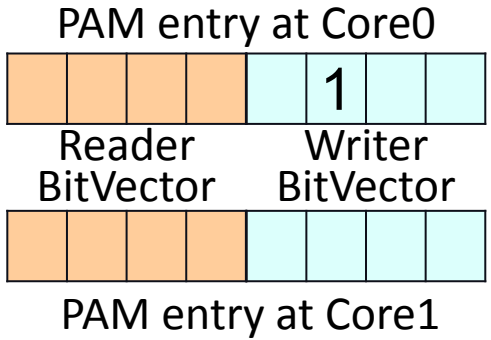
Block B0 (M) C0

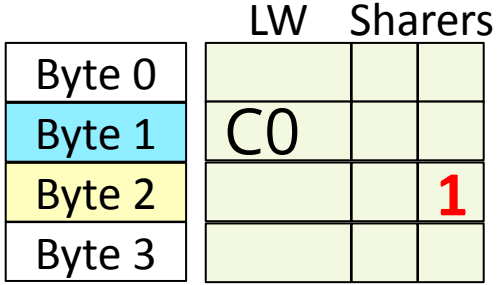
FC	IC	PMMC
3	2	1

SAM table entry for a **Block B0**

Detecting False Sharing in FSDetect





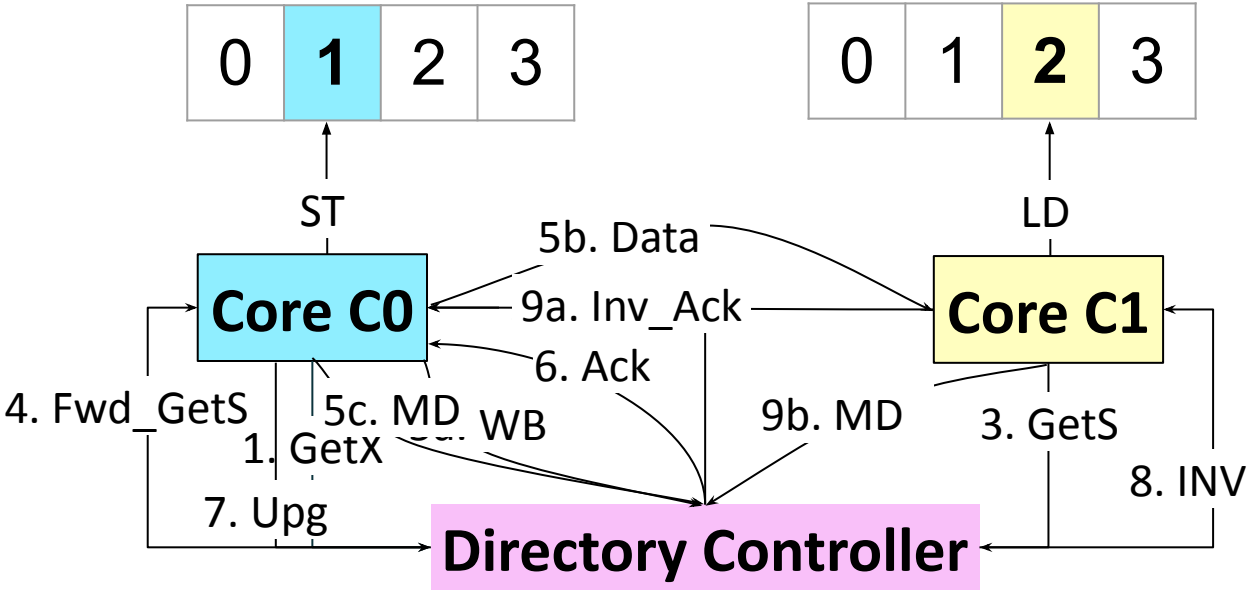


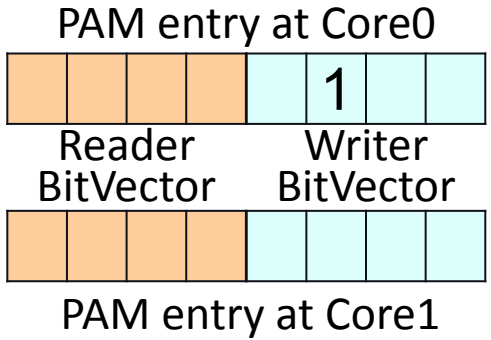
Block B0 (M) C0

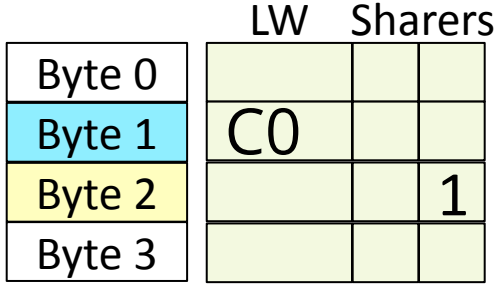
FC	IC	PMMC
3	2	0

SAM table entry for a **Block B0**

Detecting False Sharing in FSDetect





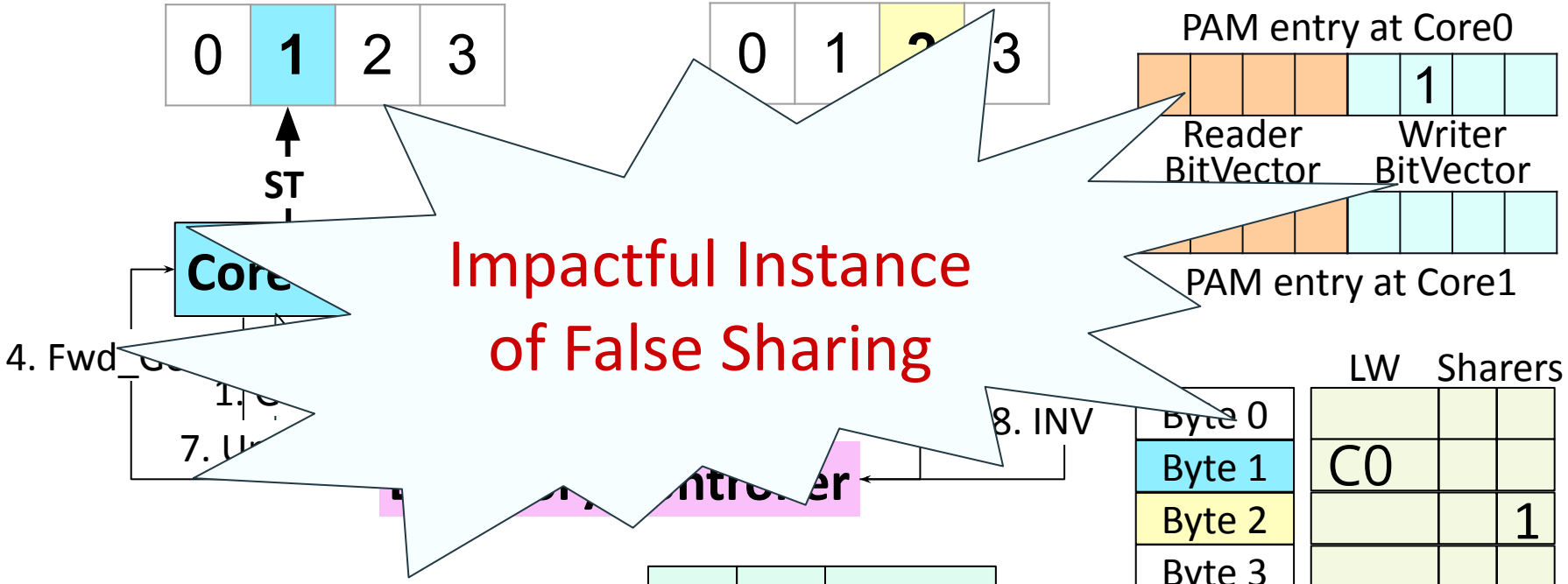


Block B0 (M) C0

FC	IC	PMMC
18	17	0

SAM table entry for a **Block B0**

Detecting False Sharing in FSDetect

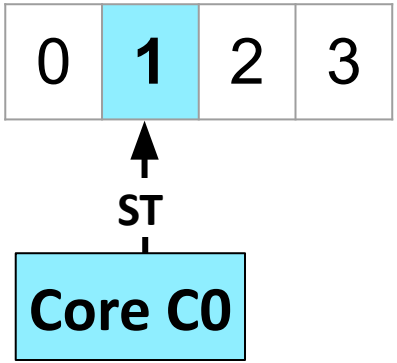


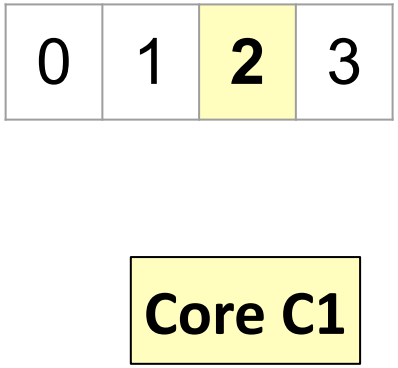
FC	IC	PMMC
18	17	0

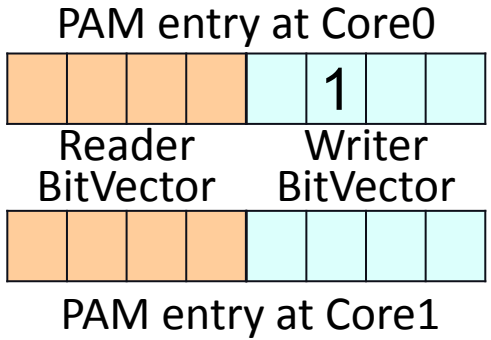
Block B0 (M) C0

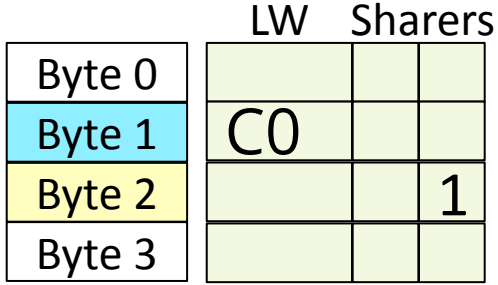
SAM table entry for a **Block B0**

Eliminating False Sharing in FSLite









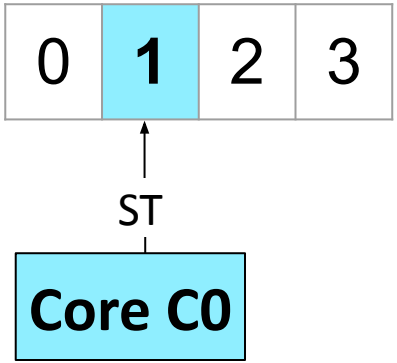
Directory Controller

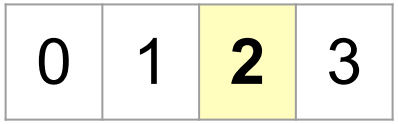
Block B0 (M) C0

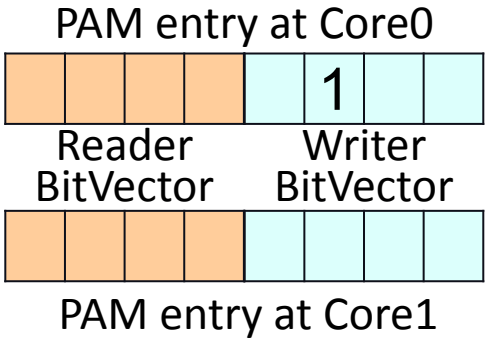
FC	IC	PMMC
18	17	0

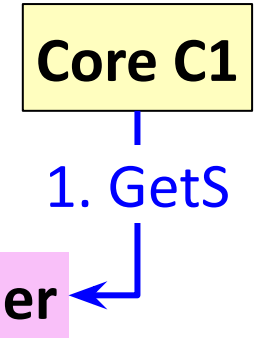
SAM table entry for a **Block B0**

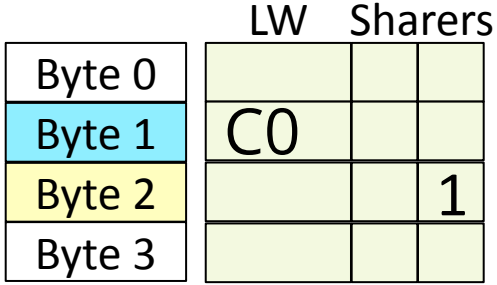
Eliminating False Sharing in FSLite









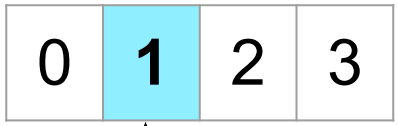


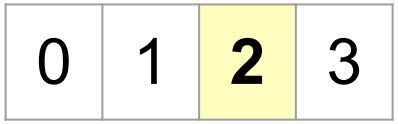
Block B0 (M) C0

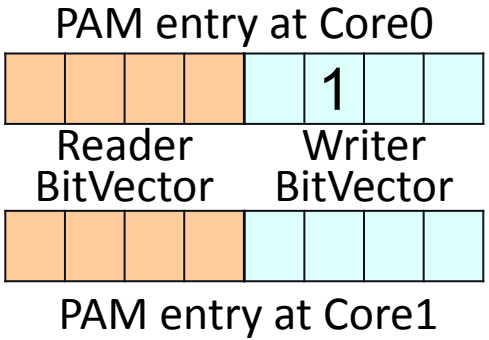
FC	IC	PMMC
18	17	0

SAM table entry for a **Block B0**

Eliminating False Sharing in FSLite







ST

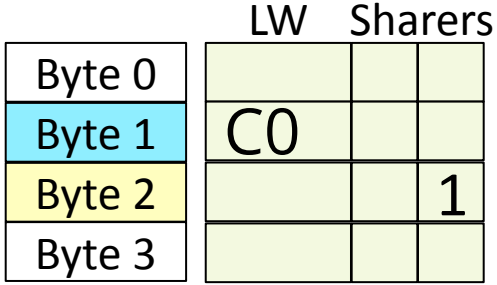
Core C0

2. TR_PRV

Core C1

1. GetS

Directory Controller

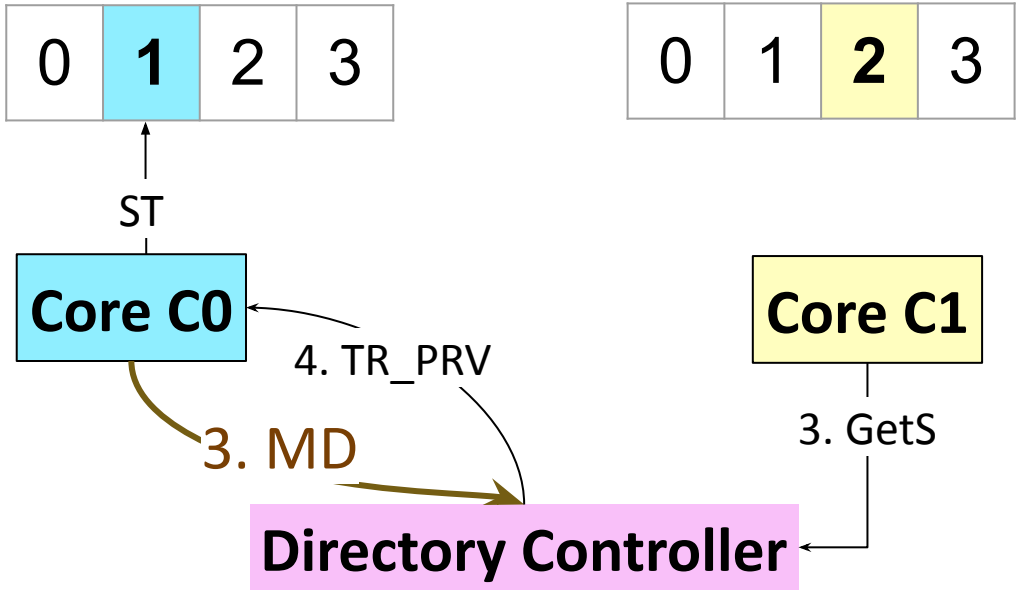


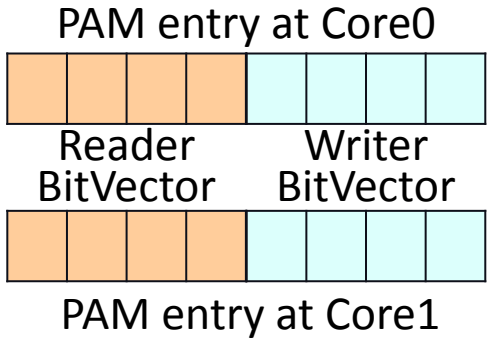
Block B0 (M) C0

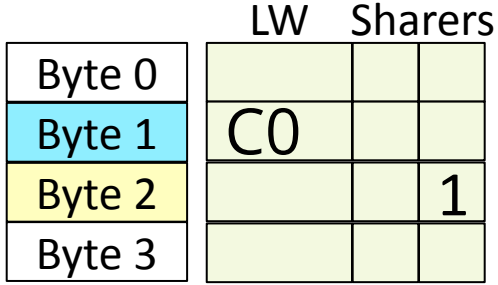
FC	IC	PMMC
18	17	1

SAM table entry for a **Block B0**

Eliminating False Sharing in FSLite





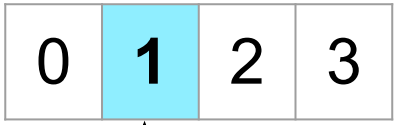


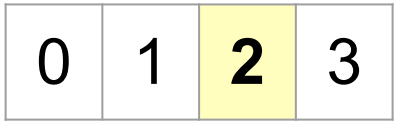
Block B0 (M) C0

FC	IC	PMMC
18	17	1

SAM table entry for a **Block B0**

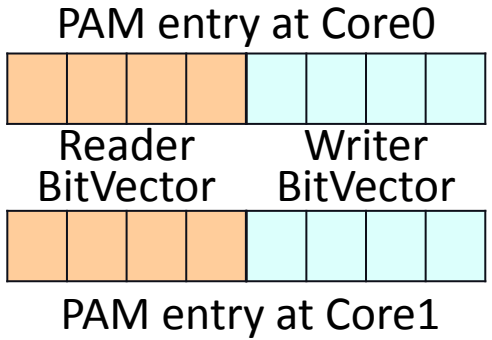
Eliminating False Sharing in FSLite

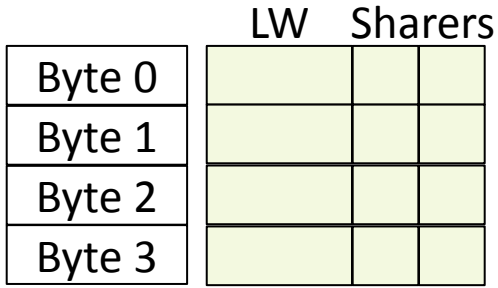




ST

Directory performs a conflict detection check



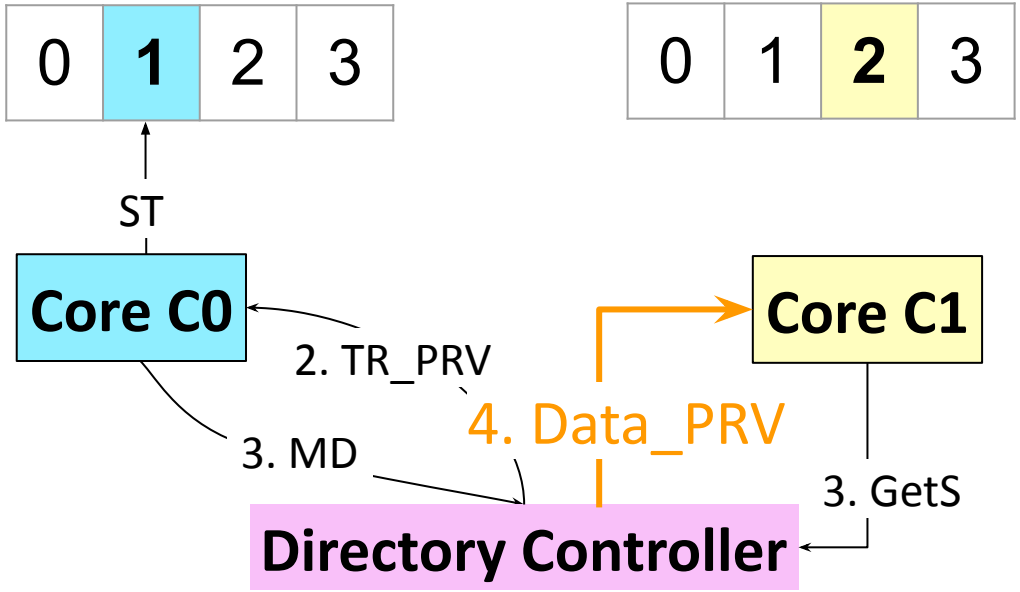


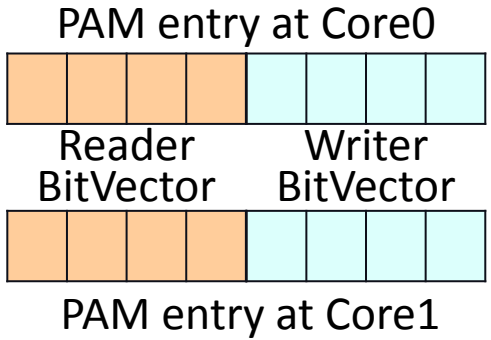
Block B0 (M) C0

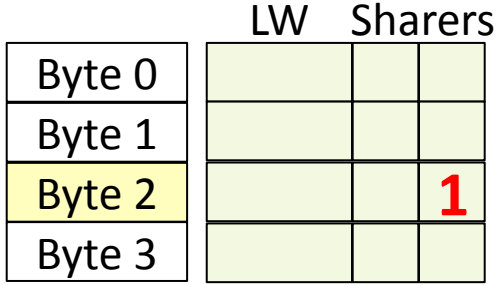
FC	IC	PMMC
18	17	0

SAM table entry for a **Block B0**

Eliminating False Sharing in FSLite





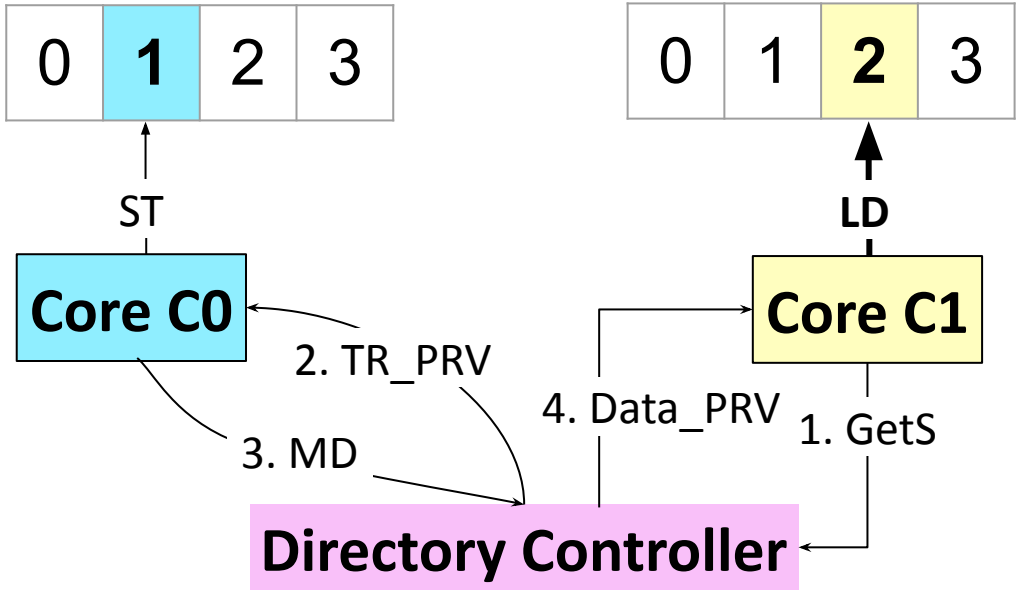


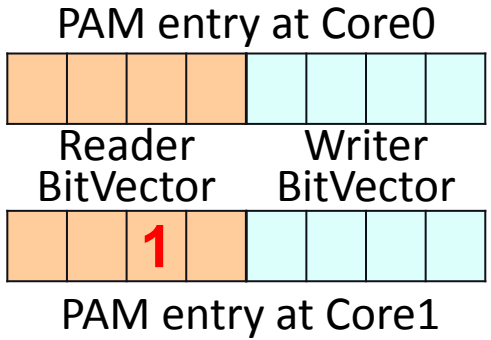
Block B0 (PRV) C0 **C1**

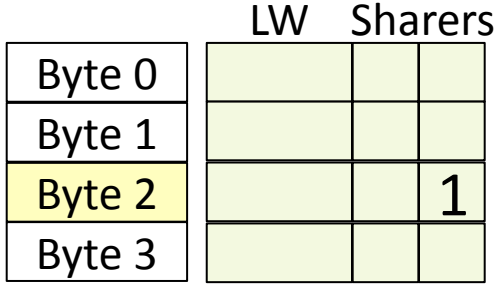
FC	IC	PMMC
18	17	0

SAM table entry for a **Block B0**

Eliminating False Sharing in FSLite





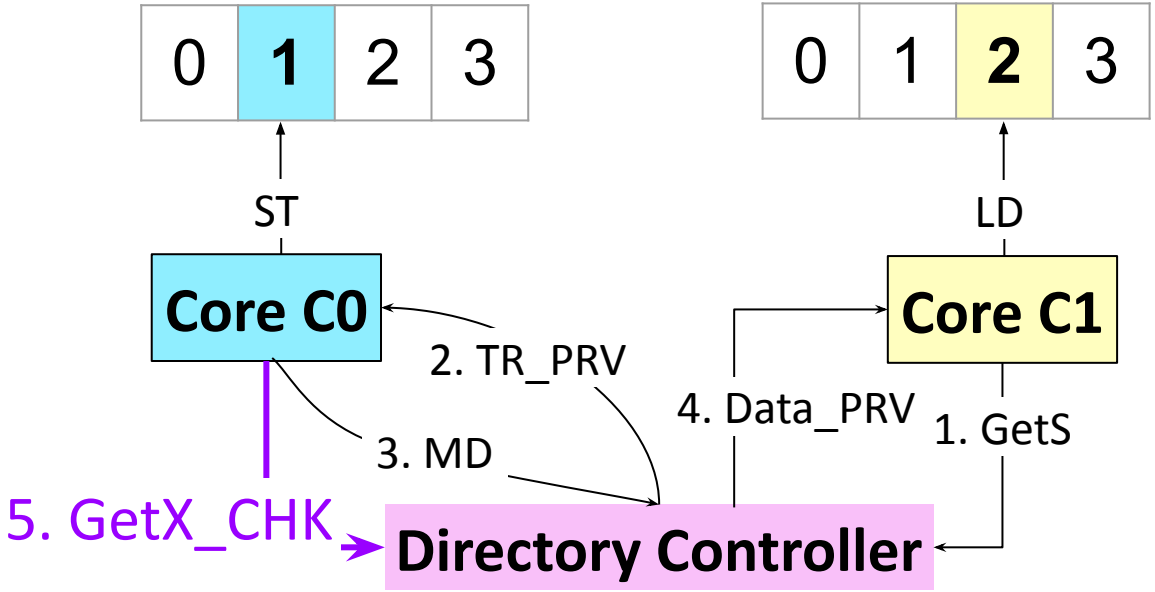


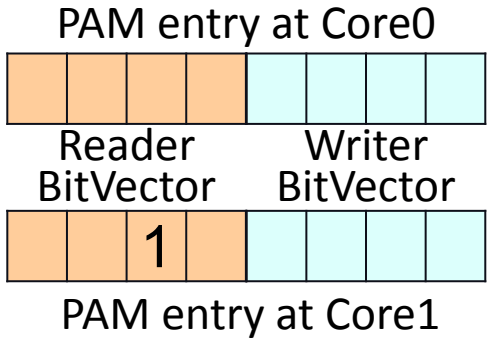
Block B0 (PRV) C0 C1

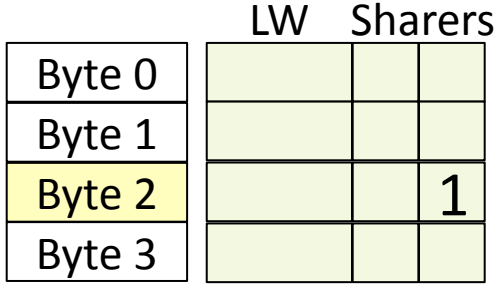
FC	IC	PMMC
18	17	0

SAM table entry for a **Block B0**

Eliminating False Sharing in FSLite





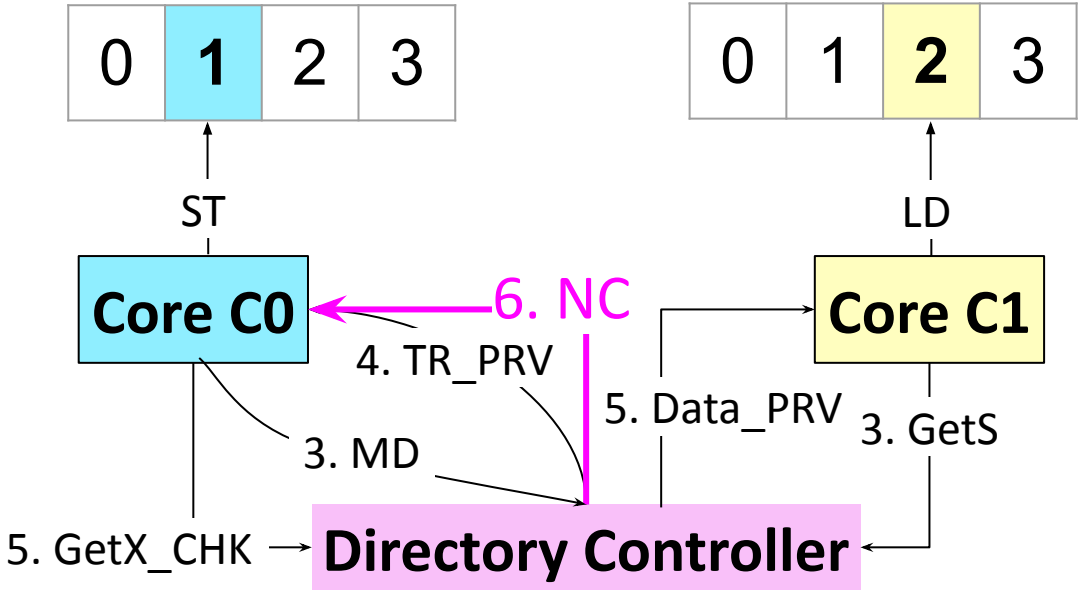


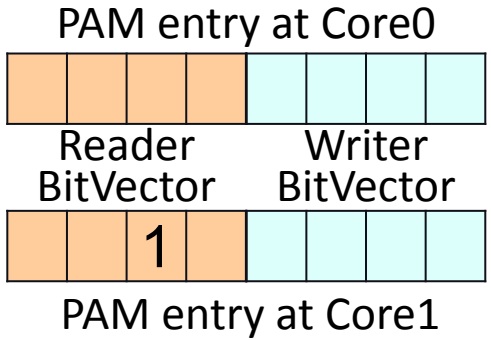
Block B0 (PRV) C0 C1

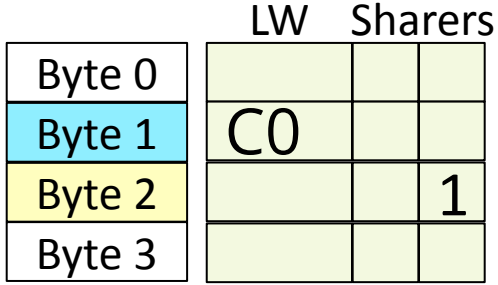
FC	IC	PMMC
18	17	0

SAM table entry for a **Block B0**

Eliminating False Sharing in FSLite





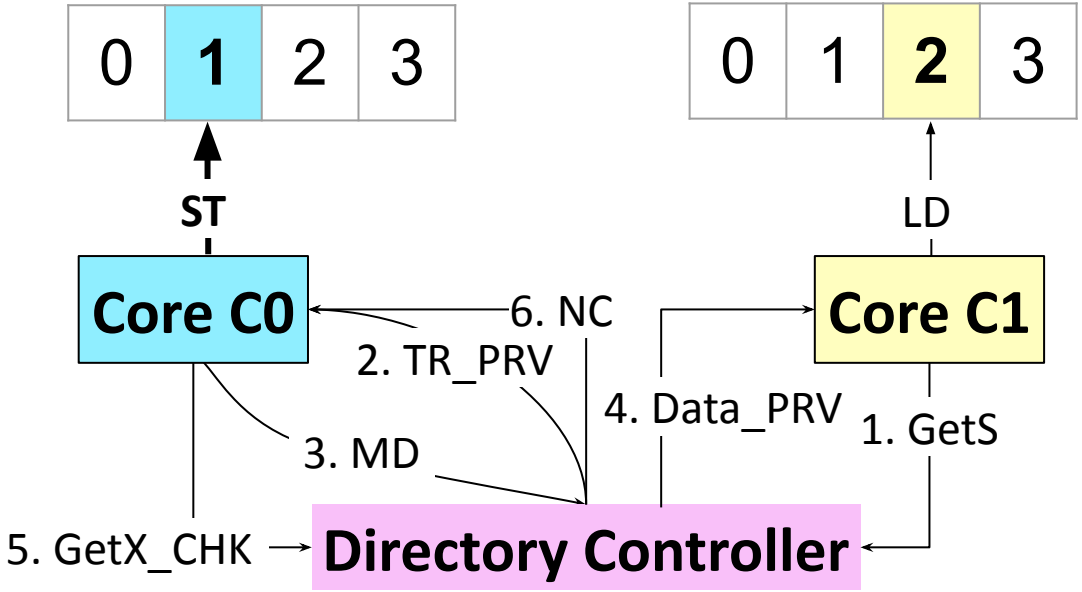


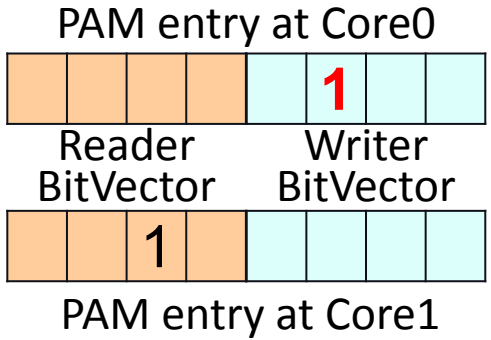
Block B0 (PRV) C0 C1

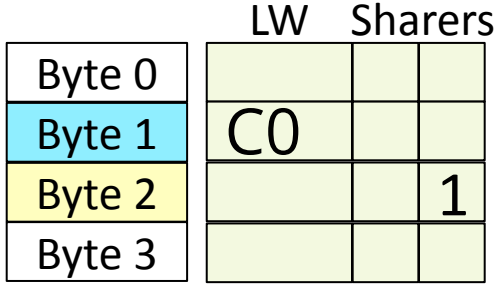
FC	IC	PMMC
18	17	0

SAM table entry for a **Block B0**

Eliminating False Sharing in FSLite





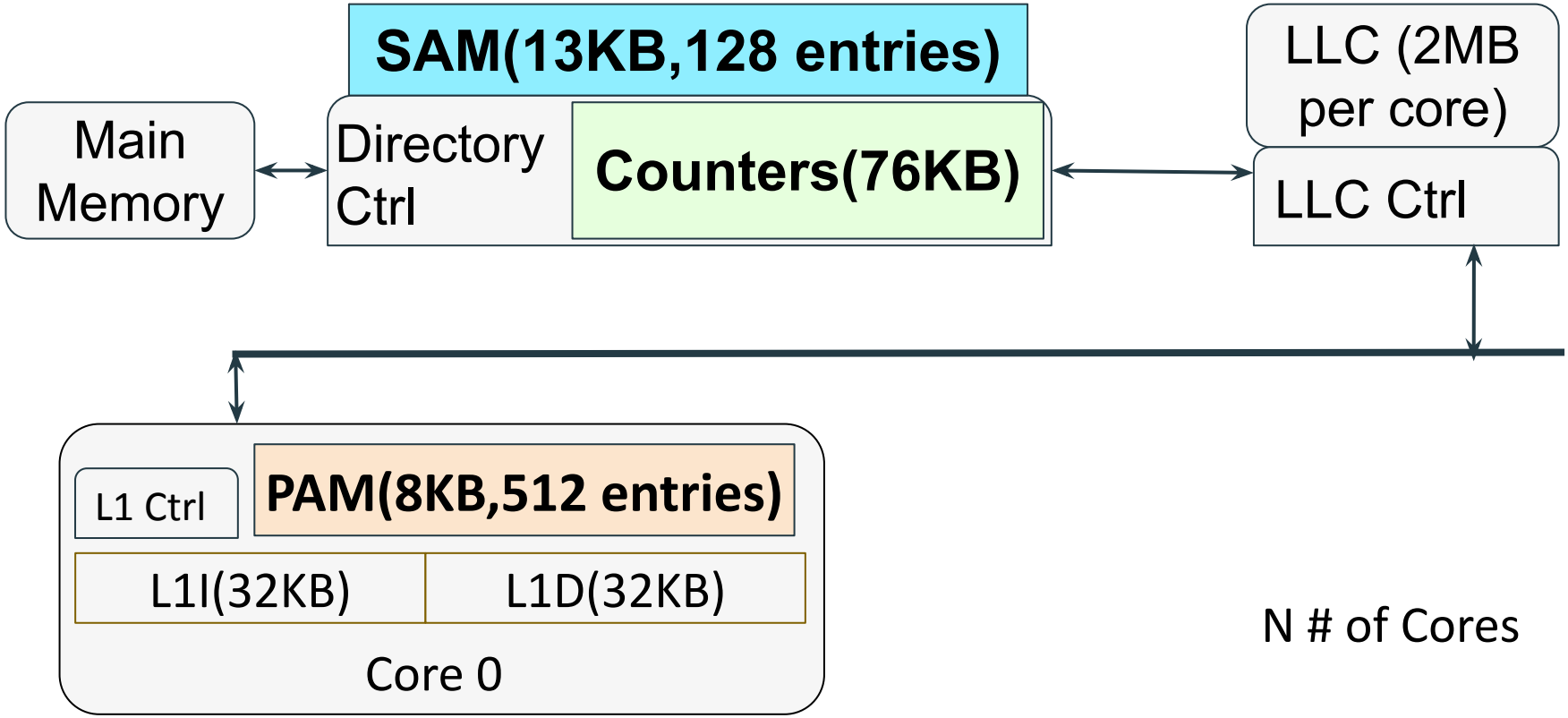


Block B0 (PRV) C0 C1

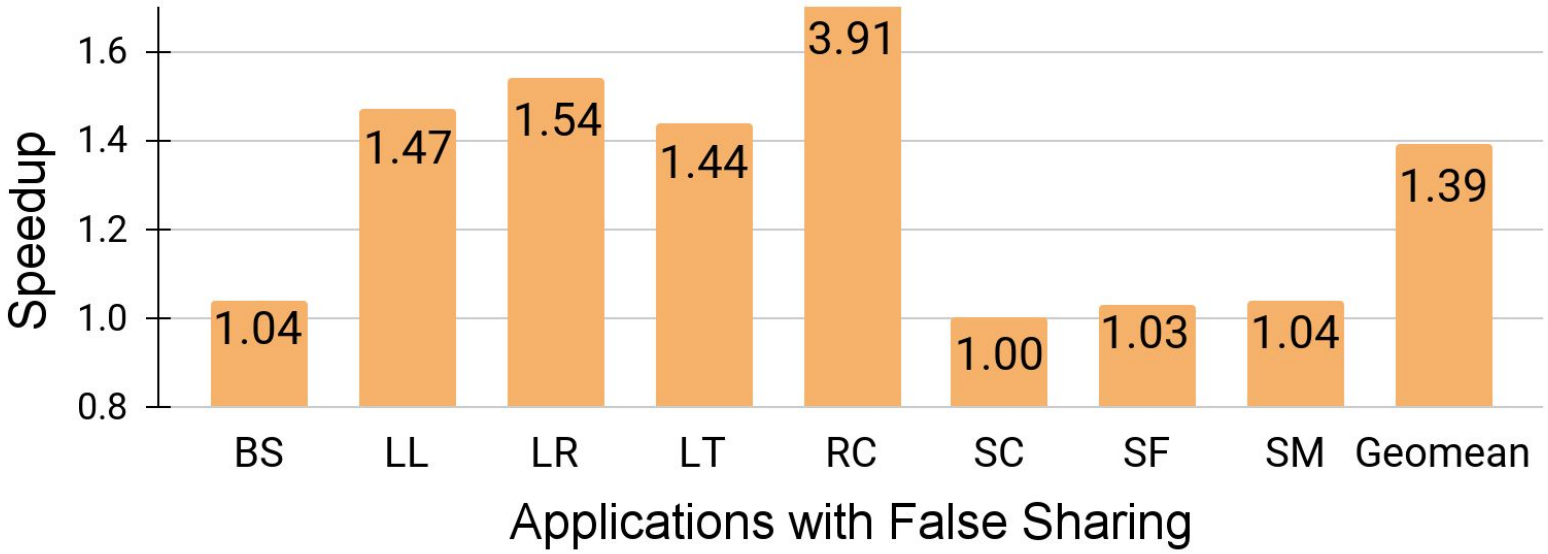
FC	IC	PMMC
18	17	0

SAM table entry for a **Block B0**

Simulation Infrastructure and Configurations

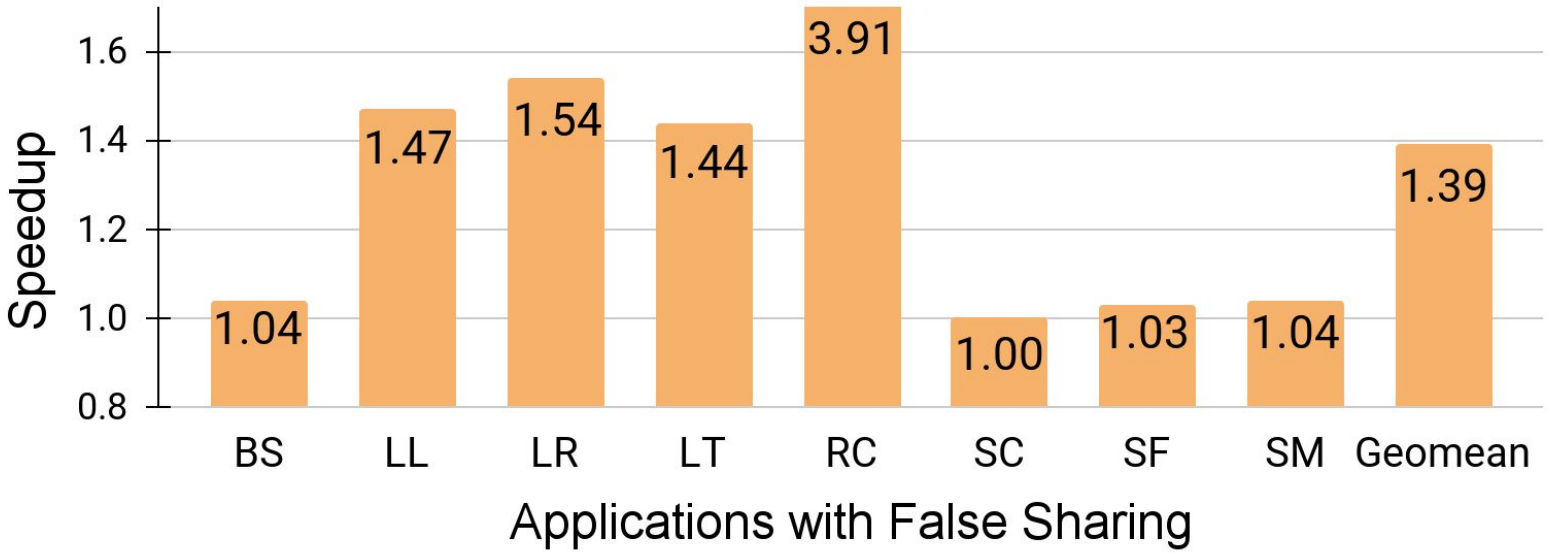


Performance Improvement with FSLite



FSLite achieves on avg **1.39x** speedup and a max speedup **3.91x**

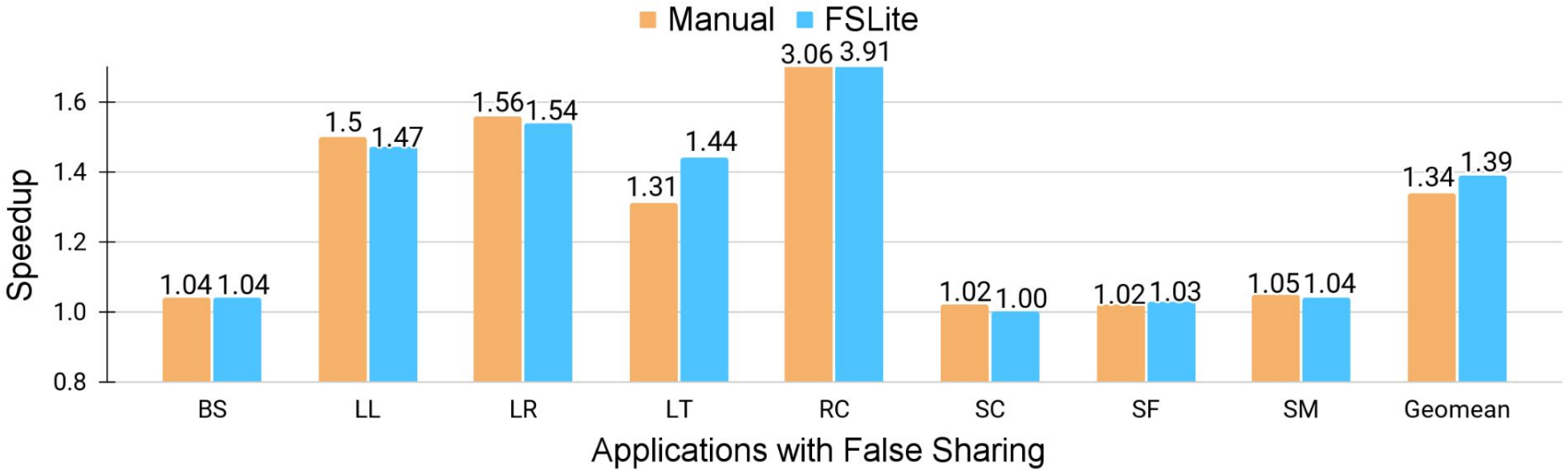
Performance Improvement with FSLite



FSLite achieves on avg **1.39x** speedup and a max speedup **3.91x**

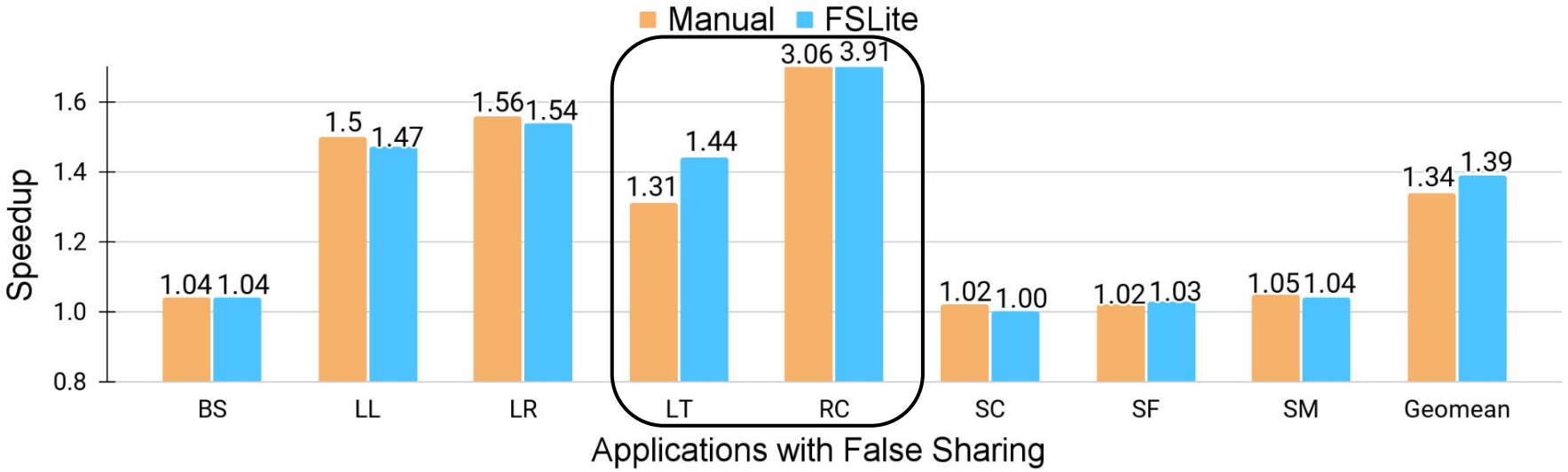
27% energy savings and **89%** less interconnect messages

Comparison with Manual Fix



FSLite outperforms manual fix by **~4%**

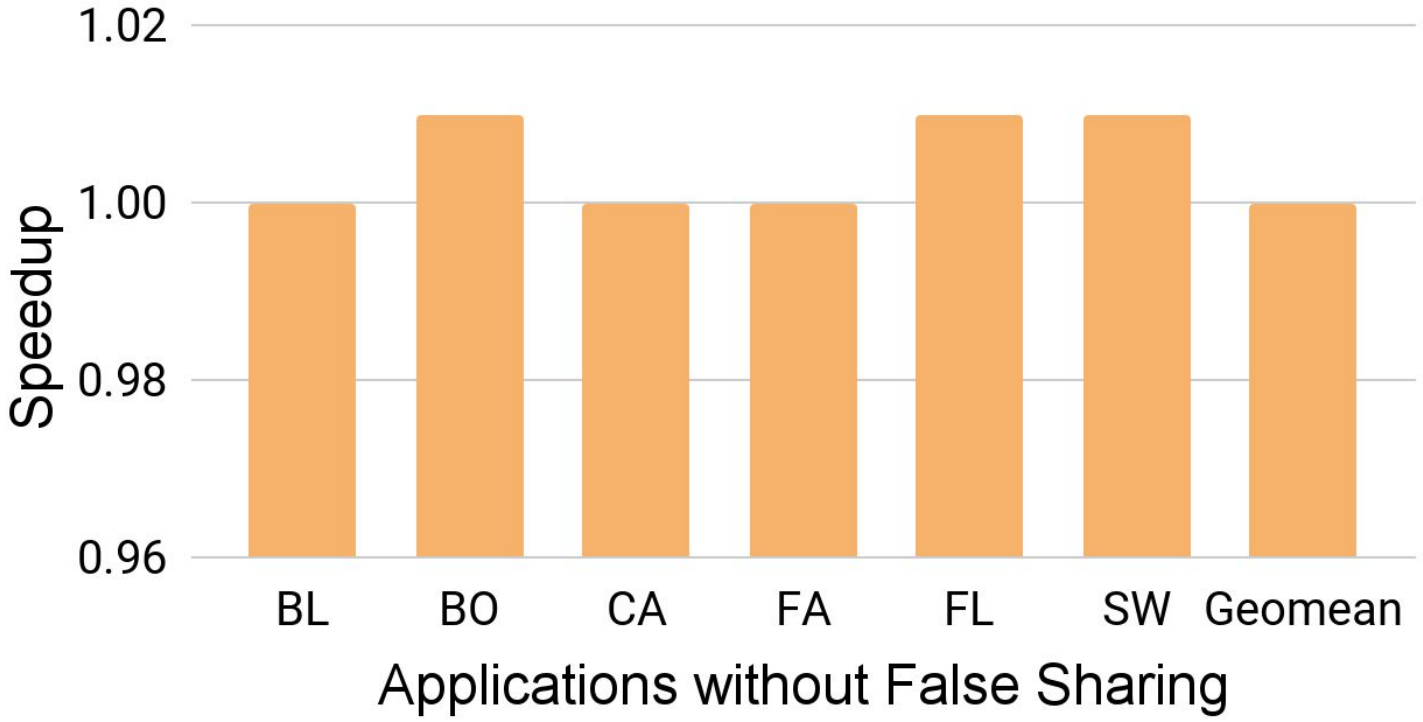
Comparison with Manual Fix



Padding inflates the memory footprint

Padding introduces additional instructions

Impact of FSLite on Applications w/o False Sharing



No slowdown for applications without false sharing

Additional Results

- **FSLite achieves an average speedup of 1.63x for O-o-O cores**

Additional Results

- FSLite achieves an average speedup of 1.63x for O-o-O cores

- **FSLite achieves an average speedup of 1.21x over a 128KB L1D cache baseline across all applications (with and without false sharing)**

Summary

- **False sharing: a common performance bug**

Summary

- False Sharing: a common performance bug

- **Protocol extensions to detect and mitigate false sharing on-the-fly**

Summary

- False Sharing: a common performance bug
- Protocol extensions to detect and mitigate false sharing on-the-fly
- **Independent of application source code and technology stack**

Summary

- False Sharing: a common performance bug
- Protocol extensions to detect and mitigate false sharing on-the-fly
- Independent of application source code and technology stack
- **Outperforms manual fix**

Summary

- False Sharing: a common performance bug
- Protocol extensions to detect and mitigate false sharing on-the-fly
- Independent of application source code and technology stack
- Outperforms manual fix
- **No side-effect on applications without false sharing**

Leveraging Cache Coherence to Detect and Repair False Sharing On-the-fly

MICRO'24

Vipin Patel, Swarnendu Biswas, and Mainak Chaudhuri

PROSPAR, Computer Science and Engineering,
Indian Institute of Technology Kanpur

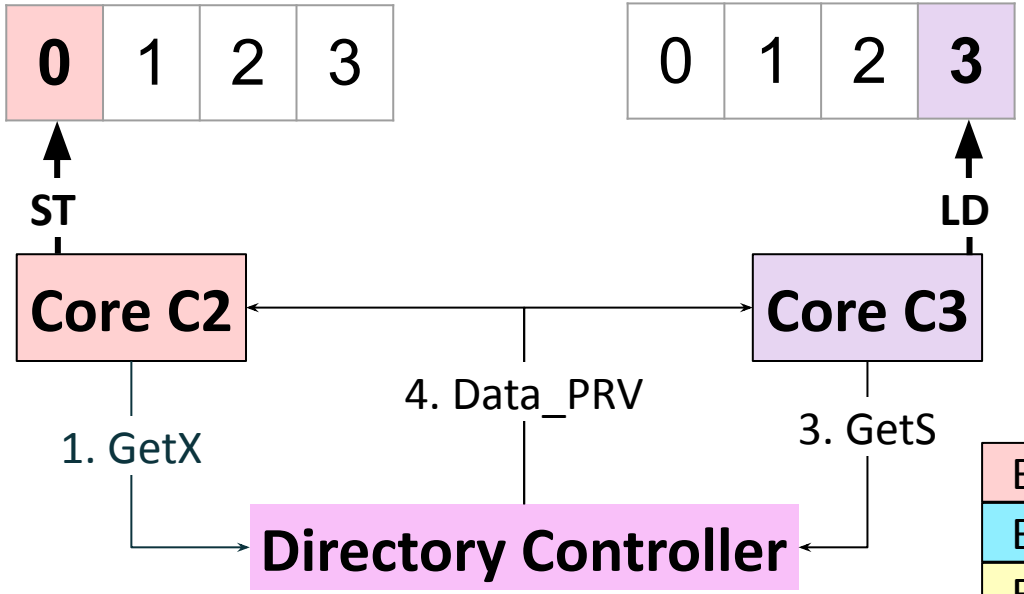
Artifact available at:
https://zenodo.org/records/13293424
Or Scan the QR code

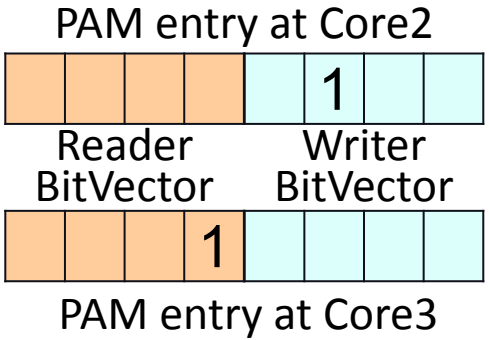
Backup slides

Simulation Infrastructure and Configurations

CPU Type	In-order CPU with 3GHz Frequency
L1D Cache	32 KB per core, 8-way, area: 7.43mm^2
L2(LLC)	2 MB per core, 16-way, area: 13.74m^2
Cache line size	64 bytes
Memory	3 GB DDR3-16000, 8 ranks, 64-bit channel
PAM table	**8 KB (512 entries) per L1D, 8-way**
SAM table	**12.7 KB (128 entries) per LLC slice, 16-way**
Directory Extension	**76 KB per LLC slice**
Conflict Detection	**2 cycles**
Thresholds	**FC:16, IC:16, and HC:3**

Handling Request during Privatization





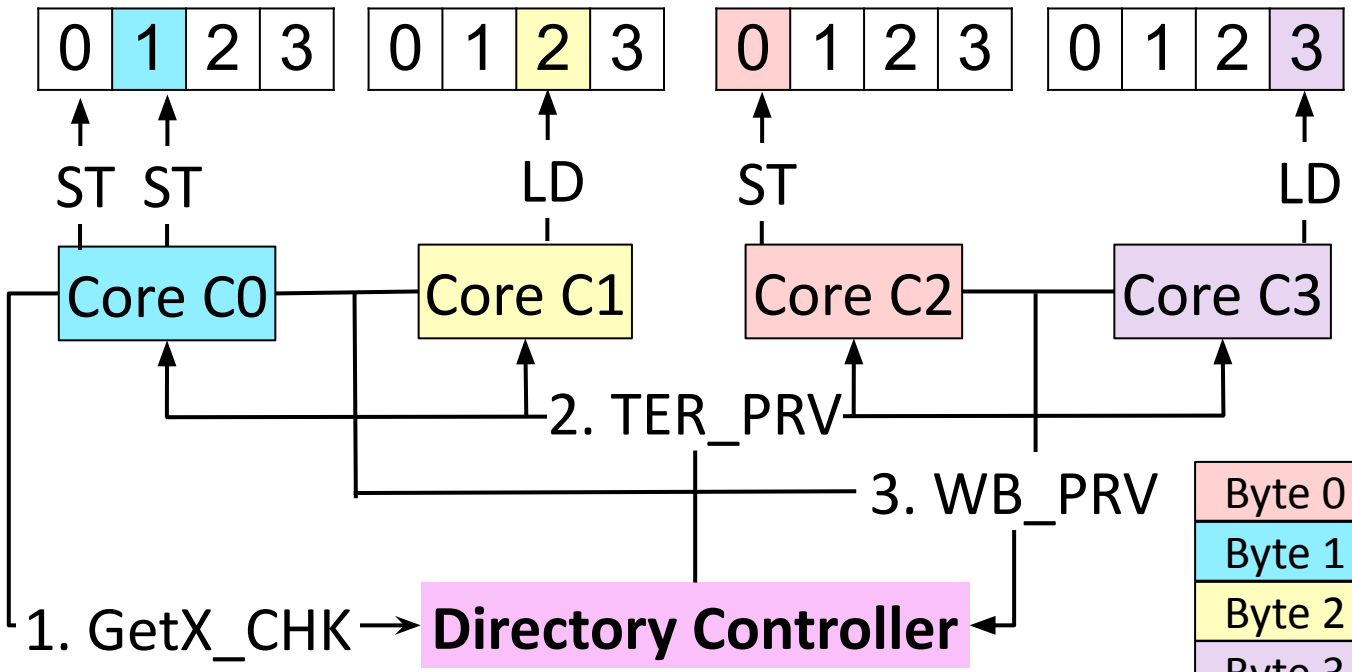
	LW	Sharers			
Byte 0	C2				
Byte 1	C0				
Byte 2			1		
Byte 3					1

SAM table entry for a **Block B0**

Block B0 (PRV) C0 C1 C2 C3

FC	IC	PMMC
18	17	0

Termination of Privatization



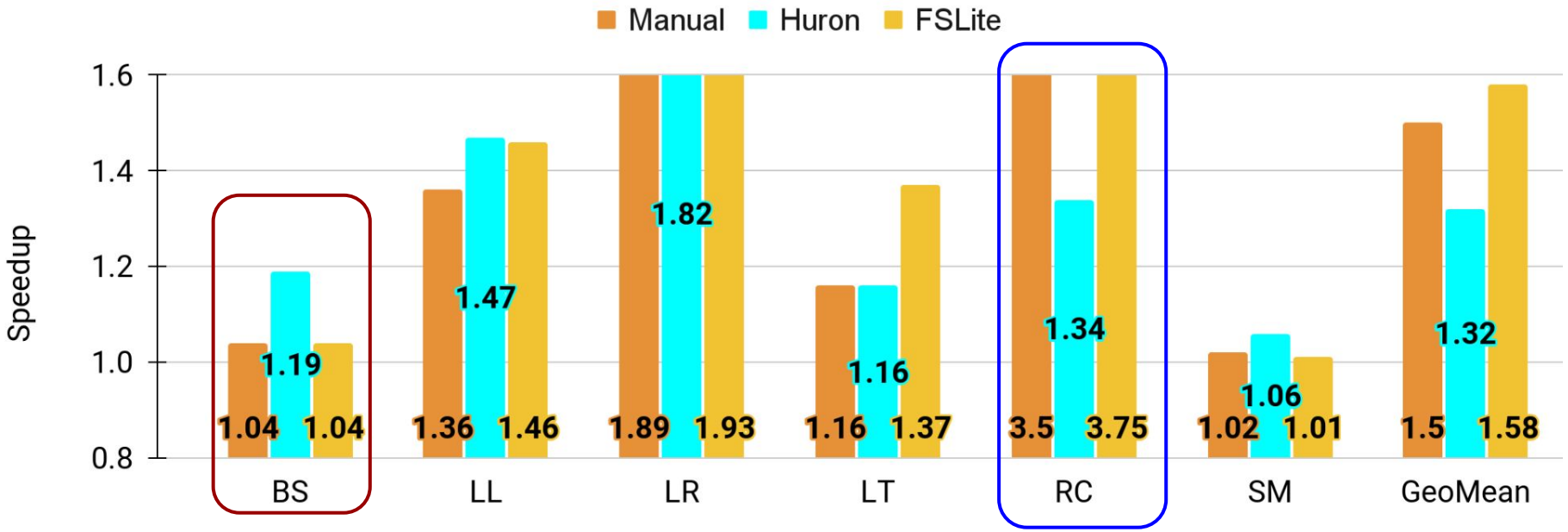
FC	IC	PMMC
0	0	0

	LW	Sharers			
Byte 0	C2				
Byte 1	C0				
Byte 2			1		
Byte 3					1

Block B0 (PRV) C0 C1 C2 C3

SAM table entry for a **Block B0**

Comparison with



15% less committed instruction for BoostSpinlock

The modified binary from Huron **contains false sharing instances**

1. T. A. Khan, Y. Zhao, G. Pokam, B. Mozafari, and B. Kasikci, "Huron: Hybrid False Sharing Detection and Repair," in PLDI, 2019.

Advantage of FSLite over existing approaches

- The coherence actions for the detection are off the critical path
- Application stack agnostic
- No access to application source code
- Unmodified coherence granularity and software guarantees
- Does not inflate memory footprint
- No compiler Support
- No additional instruction