Memory Consistency Model-Aware Cache Coherence for Heterogeneous Hardware

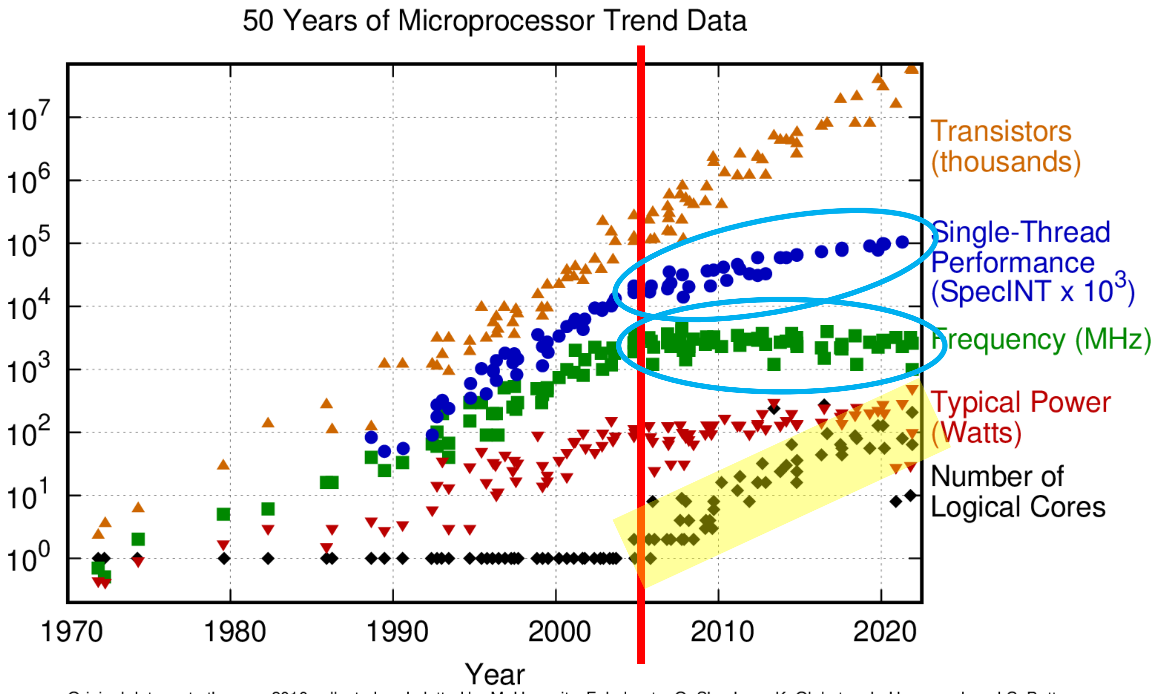
RACHEL CLEAVELAND AND CAROLINE TRIPPEL

STANFORD UNIVERSITY

2024 Formal Methods in Computer-Aided Design
Prague, Czech Republic
October 17, 2024

Stanford University

Modern Trends in Hardware Design



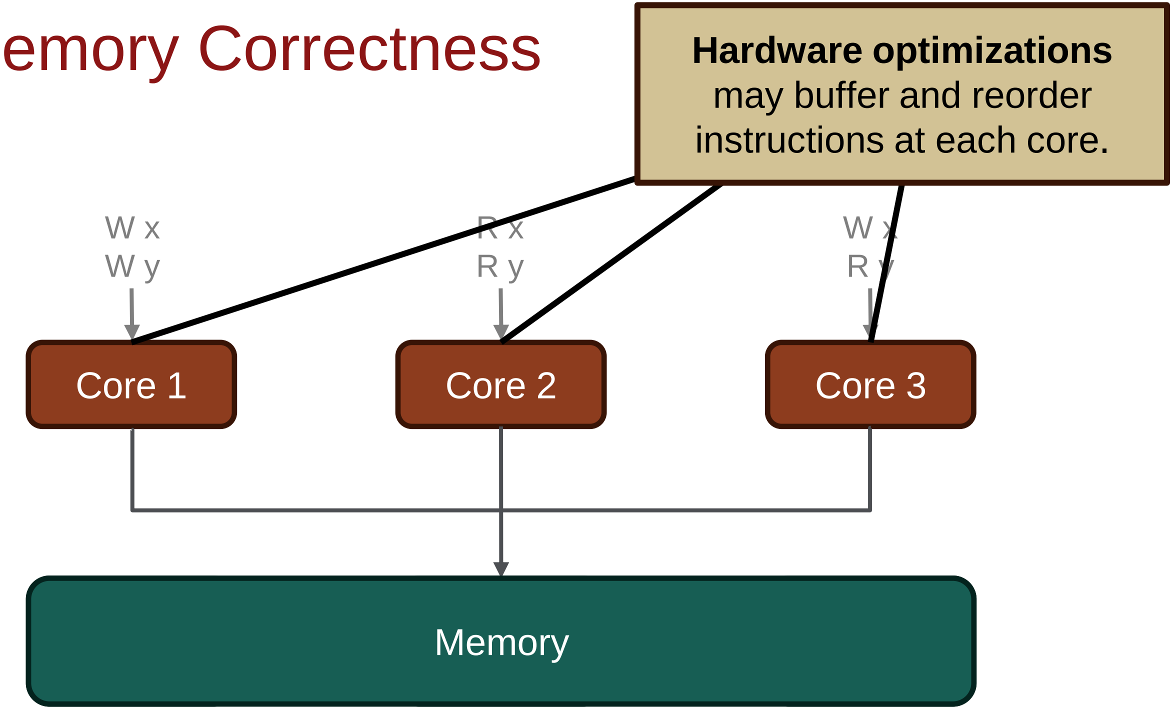
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

Parallelism and Memory Correctness

Parallelism

+

Shared Memory



Hardware optimizations may buffer and reorder instructions at each core.

Parallelism and Memory Correctness

Parallelism

+

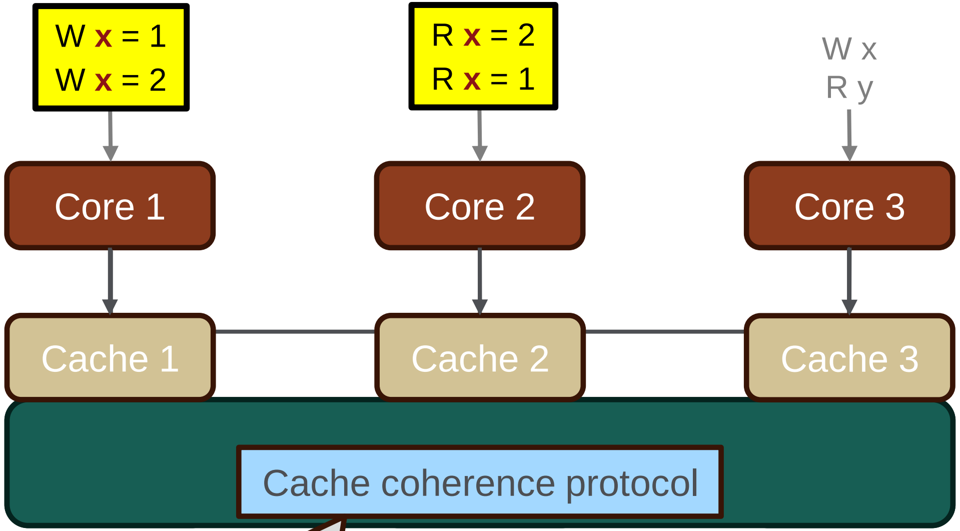
Shared Memory

+

Caching

+

Optimizations



Cache coherence protocols minimally maintain **coherence** (all cores agree on a total order on same-address accesses).

Parallelism and Memory Correctness

Parallelism

+

Shared Memory

+

Caching

+

Optimizations

W **x** = 1
W **x** = 2

R **x** = 2
R **x** = 1

Parallelism and Memory Correctness

Parallelism

+

Shared Memory

+

Caching

+

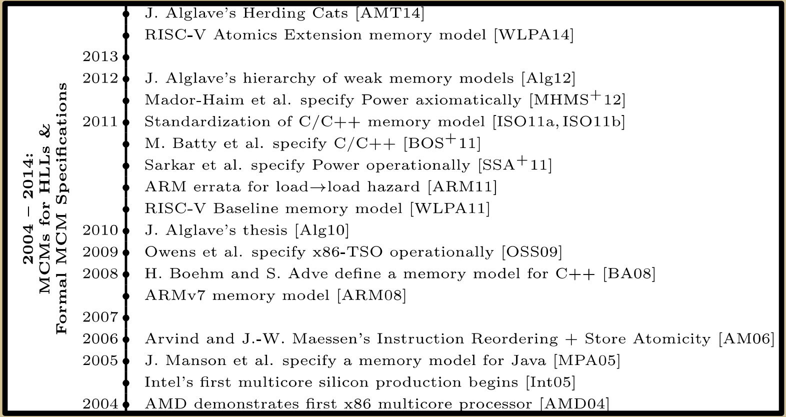
Optimizations

+

Heterogeneity

Th
W
W

Th
W
W f



2004 – 2014: MCMs for HLLs & Formal MCM Specifications

J. Alglave's Herding Cats [AMT14]
RISC-V Atomics Extension memory model [WLPA14]
2013
2012 J. Alglave's hierarchy of weak memory models [Alg12]
Mador-Haim et al. specify Power axiomatically [MHMS+12]
2011 Standardization of C/C++ memory model [ISO11a, ISO11b]
M. Batty et al. specify C/C++ [BOS+11]
Sarkar et al. specify Power operationally [SSA+11]
ARM errata for load→load hazard [ARM11]
RISC-V Baseline memory model [WLPA11]
2010 J. Alglave's thesis [Alg10]
2009 Owens et al. specify x86-TSO operationally [OSS09]
2008 H. Boehm and S. Adve define a memory model for C++ [BA08]
ARMv7 memory model [ARM08]
2007
2006 Arvind and J.-W. Maessen's Instruction Reordering + Store Atomicity [AM06]
2005 J. Manson et al. specify a memory model for Java [MPA05]
Intel's first multicore silicon production begins [Int05]
2004 AMD demonstrates first x86 multicore processor [AMD04]

ctures
nce,
tcome.

ower

y

x86

Memory consistency models (MCMs) define the ordering requirements between *all* memory operations in a program.

Stanford University

Heterogeneity in Modern Hardware

Parallelism

+

Shared Memory

+

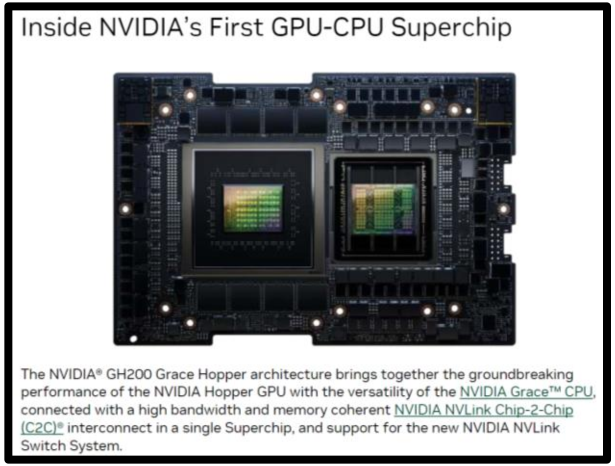
Caching

+

Optimizations

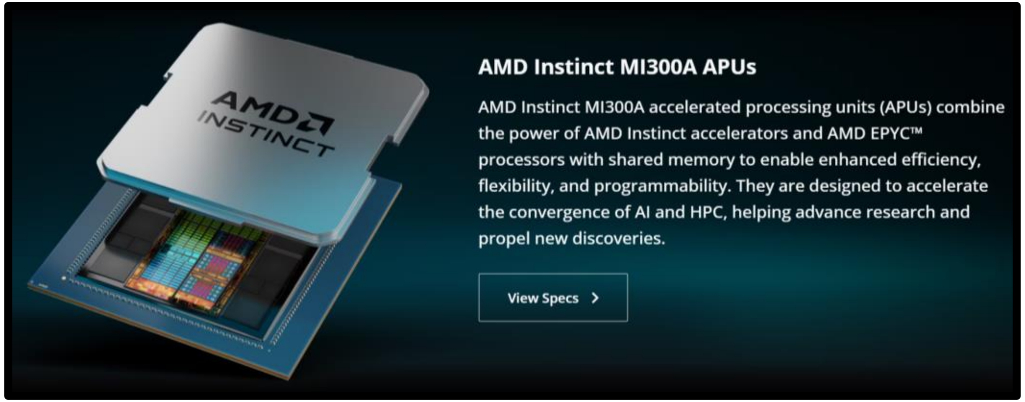
+

Heterogeneity



Inside NVIDIA's First GPU-CPU Superchip

The NVIDIA® GH200 Grace Hopper architecture brings together the groundbreaking performance of the NVIDIA Hopper GPU with the versatility of the NVIDIA Grace™ CPU, connected with a high bandwidth and memory coherent NVIDIA NVLink Chip-2-Chip (C2C)® interconnect in a single Superchip, and support for the new NVIDIA NVLink Switch System.



AMD Instinct MI300A APUs

AMD Instinct MI300A accelerated processing units (APUs) combine the power of AMD Instinct accelerators and AMD EPYC™ processors with shared memory to enable enhanced efficiency, flexibility, and programmability. They are designed to accelerate the convergence of AI and HPC, helping advance research and propel new discoveries.

Heterogeneity and Memory Correctness

Parallelism

+

Shared Memory

+

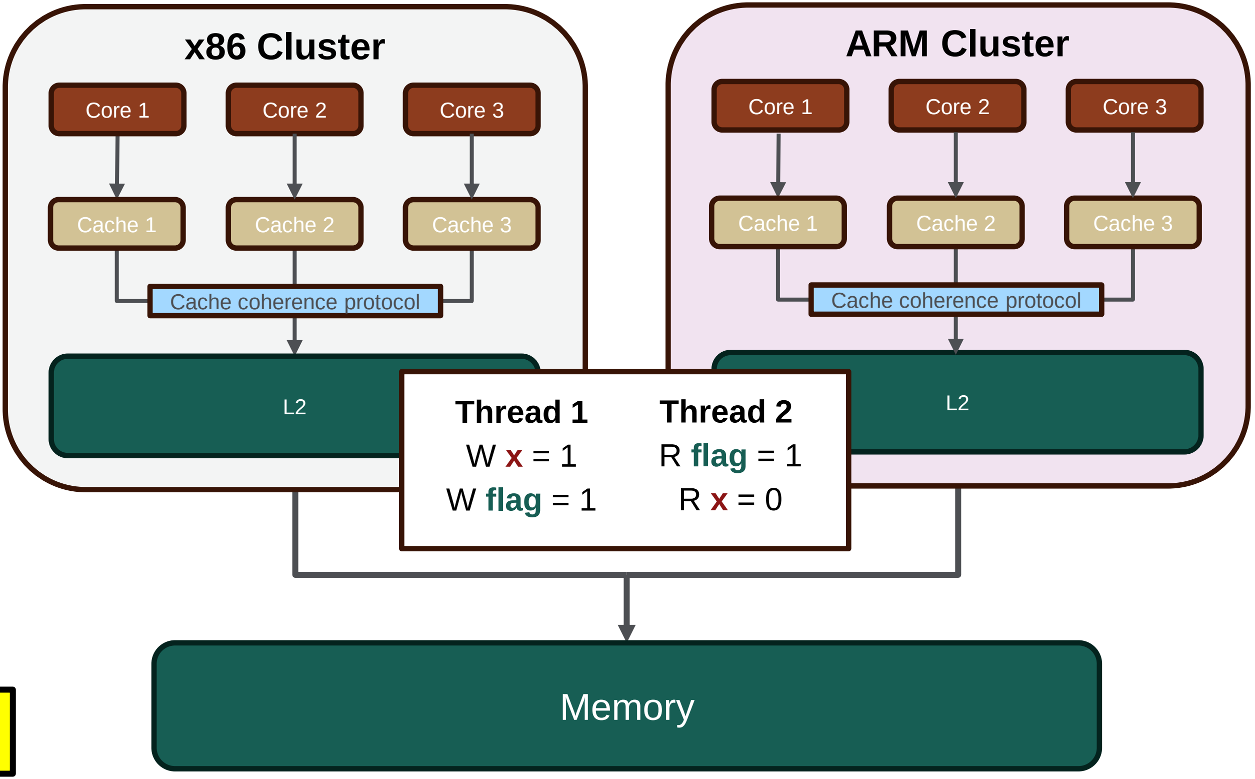
Caching

+

Optimizations

+

Heterogeneity



Thread 1	Thread 2
W **x** = 1	R **flag** = 1
W **flag** = 1	R **x** = 0

Heterogeneity and Memory Correctness

Parallelism

+

Shared Memory

+

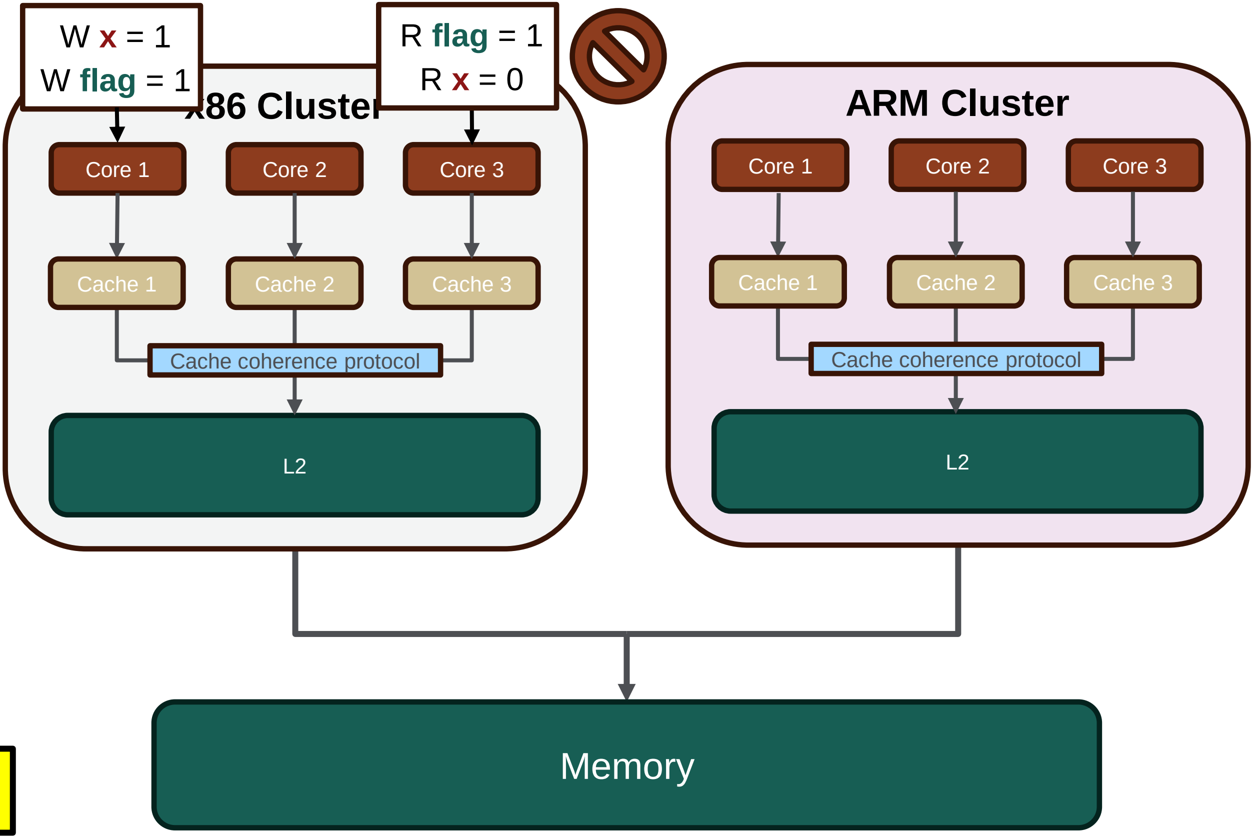
Caching

+

Optimizations

+

Heterogeneity



Heterogeneity and Memory Correctness

Parallelism

+

Shared Memory

+

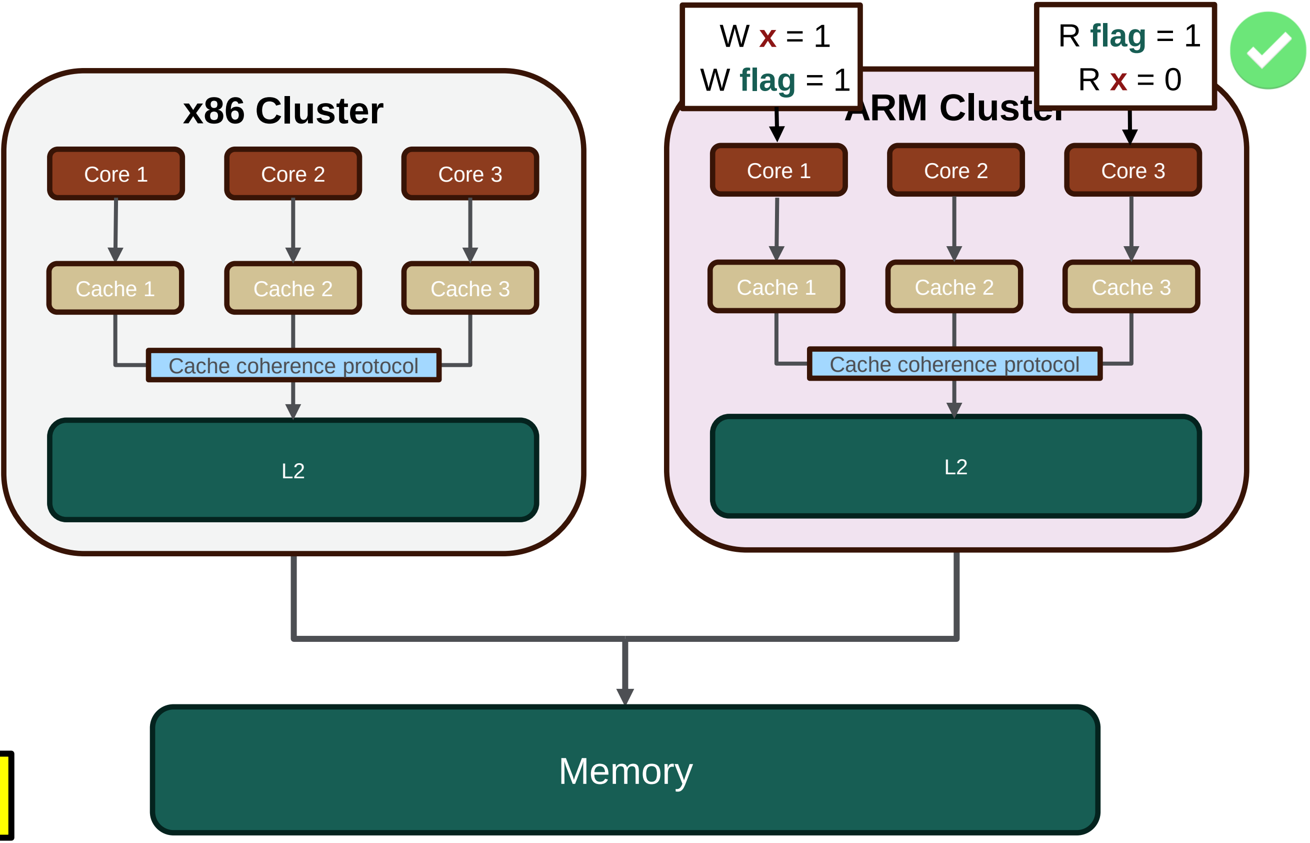
Caching

+

Optimizations

+

Heterogeneity



Heterogeneity and Memory Correctness

Parallelism

+

Shared Memory

+

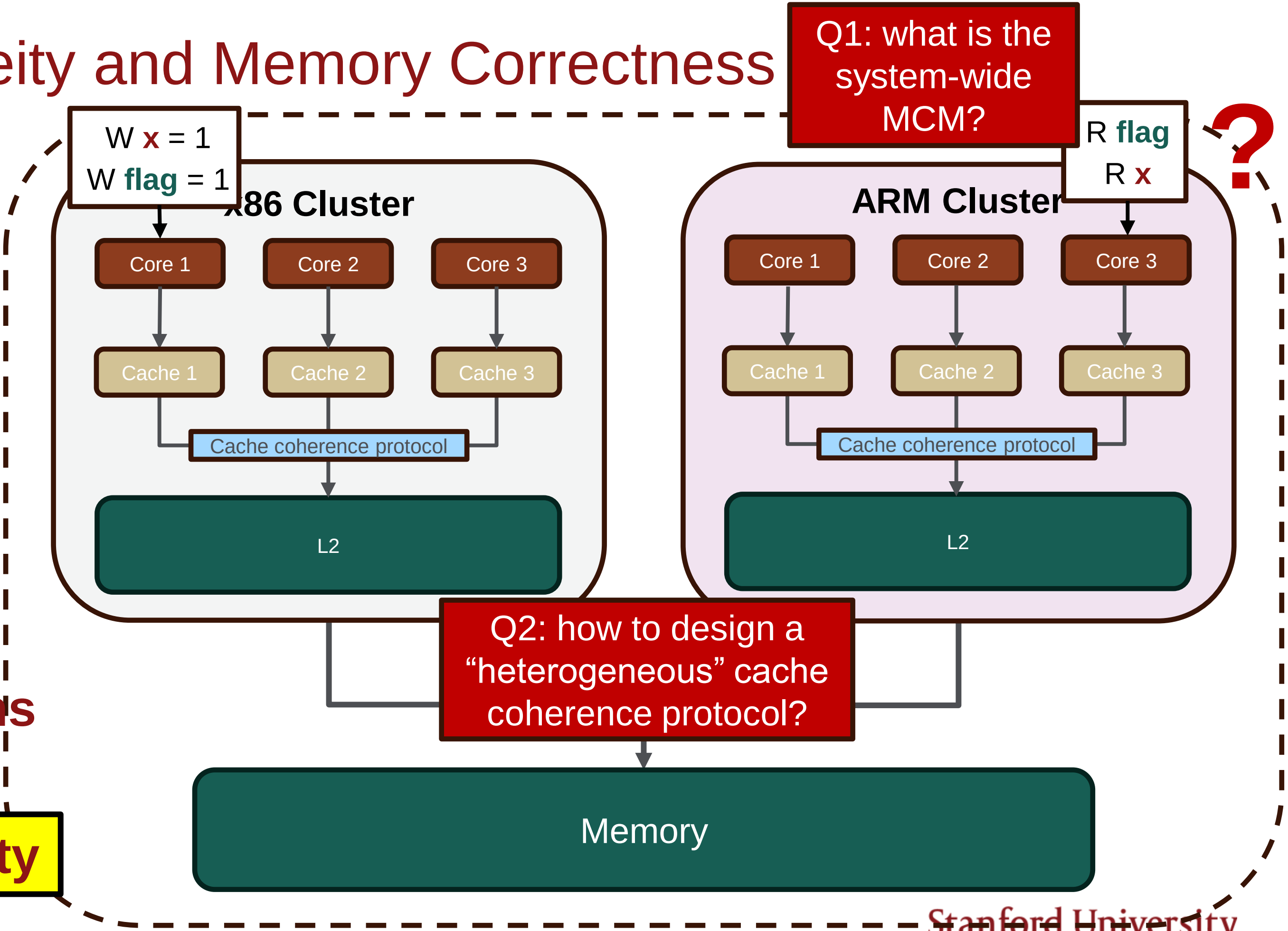
Caching

+

Optimizations

+

Heterogeneity



Q1: what is the system-wide MCM?

Q2: how to design a “heterogeneous” cache coherence protocol?

Industrial Approach: Coherence Interfaces

Parallelism

+

Shared Memory

+

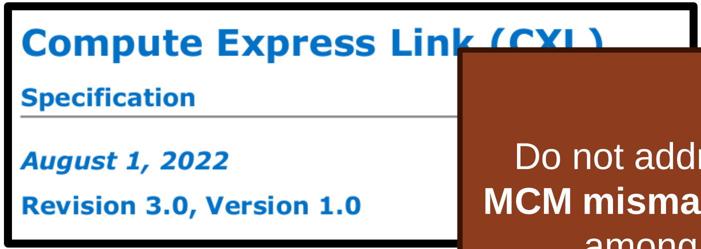
Caching

+

Optimizations

+

Heterogeneity

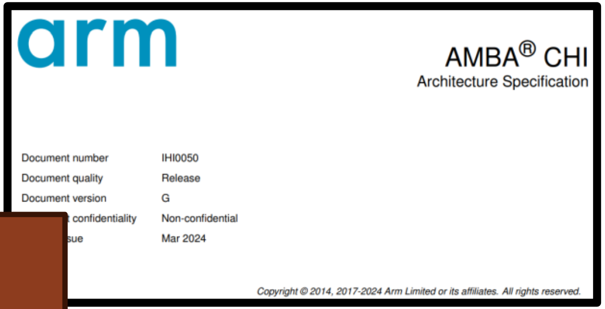


Compute Express Link (CXL)

Specification

August 1, 2022

Revision 3.0, Version 1.0



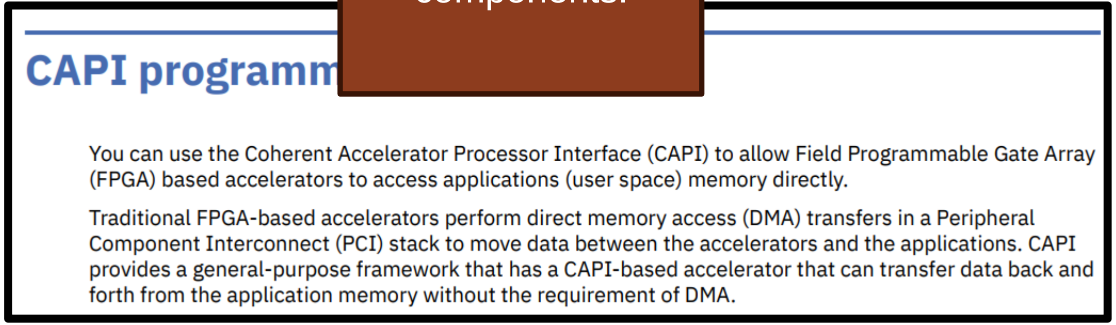
arm

AMBA® CHI
Architecture Specification

Document number	IHI0050
Document quality	Release
Document version	G
confidentiality	Non-confidential
sue	Mar 2024

Copyright © 2014, 2017-2024 Arm Limited or its affiliates. All rights reserved.

Do not address **MCM mismatches** among components!



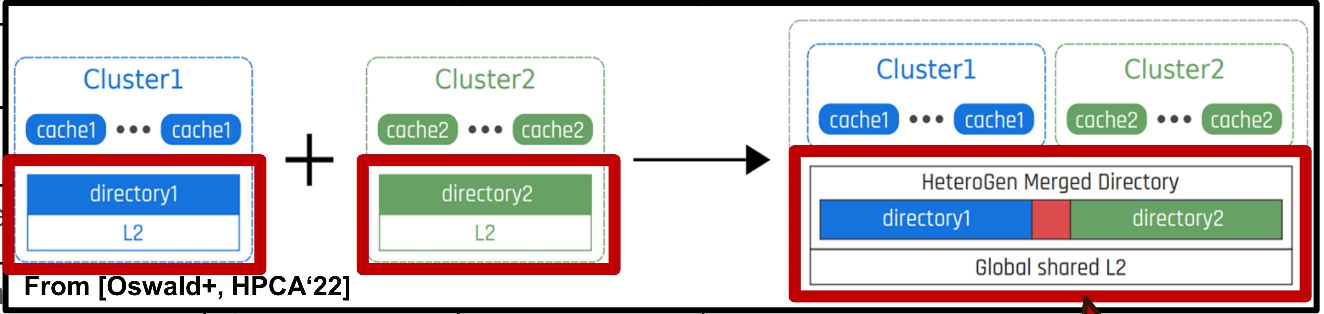
CAPI programm

You can use the Coherent Accelerator Processor Interface (CAPI) to allow Field Programmable Gate Array (FPGA) based accelerators to access applications (user space) memory directly.

Traditional FPGA-based accelerators perform direct memory access (DMA) transfers in a Peripheral Component Interconnect (PCI) stack to move data between the accelerators and the applications. CAPI provides a general-purpose framework that has a CAPI-based accelerator that can transfer data back and forth from the application memory without the requirement of DMA.

Current Approaches

	Approach	Cache Coherent	Memory Consistency
Industrial	CXL[1]	✓	✘
	2]		
	[3]		
	nde		
	rossin		
	HeteroGen[6]	✓	✓



From [Oswald+, HPCA‘22]

Synthesizes a fresh MCM-aware coherence protocol (and MCM) for each system it unifies.

Directly merges clusters’ memory systems and coherence protocols

[1] Debendra Das Sharma and Siamak Tavallaei. Compute Express Link 2.0. 2020.
[2] ARM. AMBA CHI Architecture Specification. 2024.
[3] J. Stuecheli et al. CAPI: A Coherent Accelerator Processor Interface. *IBM Journal of Research and Development*, 2015.
[4] Johnathan Alsop, Matthew Sinclair, and Sarita Adve. Spandex: A flexible interface for efficient heterogeneous coherence. *ISCA* 2018.
[5] Lena E. Olson, Mark D. Hill, and David A. Wood. Crossing guard: Mediating host-accelerator coherence interactions. *ASPLOS*, 2017.
[6] Nicolai Oswald et al. Heterogen: Automatic synthesis of heterogeneous cache coherence protocols. *HPCA*, 2022.

Current Approaches

				MemGlue's Features		
	Approach	Cache Coherent	Memory Consistency	Modular	Verifiable	Polite
Industrial	CXL[1]	✓	✗	✓	✓	✗
	CHI[2]	✓	✗	✓	✓	✗
	CAPI[3]	✓	✗	✓	✓	✗
Academic	Spandex[4]	✓	✗	✓	✓	✗
	Crossing Guard[5]	✓	✗	✓	✓	✗
	HeteroGen[6]	✓	✓	✗	✗	✗
Our work	MemGlue	✓	✓	✓	✓	✓

[1] Debendra Das Sharma and Siamak Tavallaei. Compute Express Link 2.0. 2020.
[2] ARM. AMBA CHI Architecture Specification. 2024.
[3] J. Stuecheli et al. CAPI: A Coherent Accelerator Processor Interface. *IBM Journal of Research and Development*, 2015.
[4] Johnathan Alsop, Matthew Sinclair, and Sarita Adve. Spandex: A flexible interface for efficient heterogeneous coherence. *ISCA* 2018.
[5] Lena E. Olson, Mark D. Hill, and David A. Wood. Crossing guard: Mediating host-accelerator coherence interactions. *ASPLOS*, 2017.
[6] Nicolai Oswald et al. Heterogen: Automatic synthesis of heterogeneous cache coherence protocols. *HPCA*, 2022.

Roadmap

MemGlue Design Principles

Ordered MemGlue (Ordered Interconnect Network)

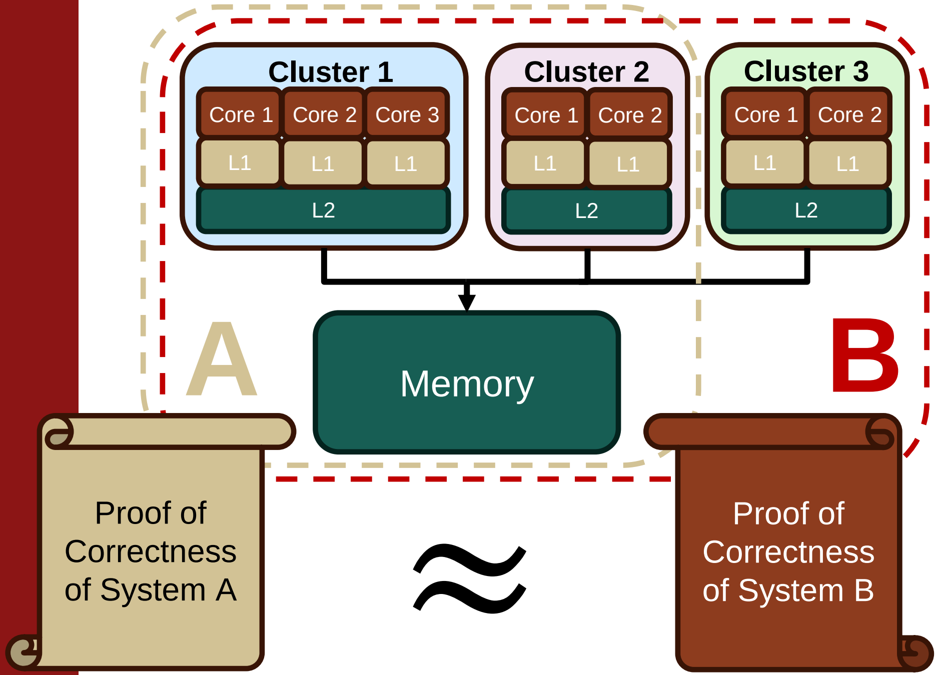
Unordered MemGlue (Unordered Interconnect Network)

Experimental Evaluation & Results

- Bounded proof of correctness (litmus testing)
- Complete proof of correctness (manual)

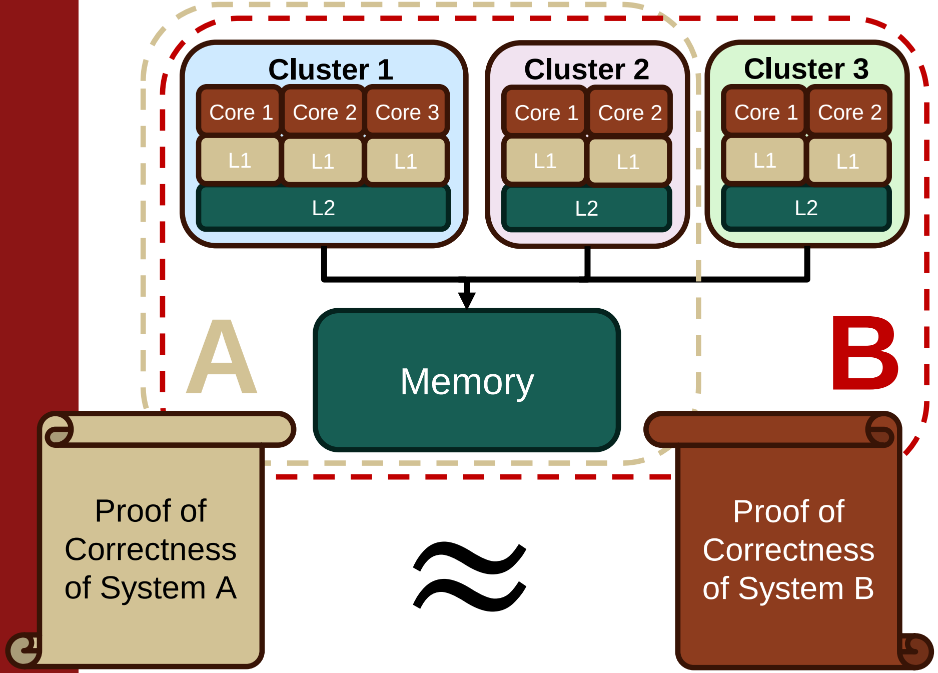
MemGlue Design Principles

Principles 1 & 2: A heterogeneous cache coherence protocol should be **modular** and **verifiable**.



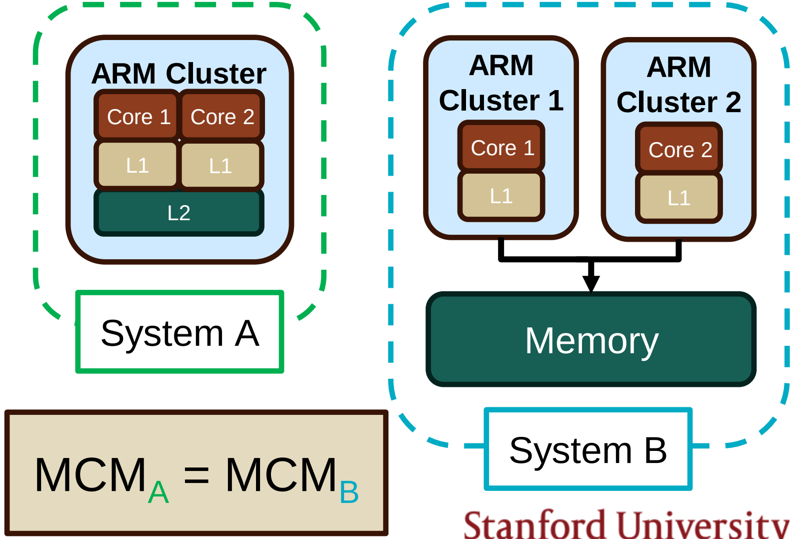
MemGlue Design Principles

Principles 1 & 2: A heterogeneous cache coherence protocol should be **modular** and **verifiable**.



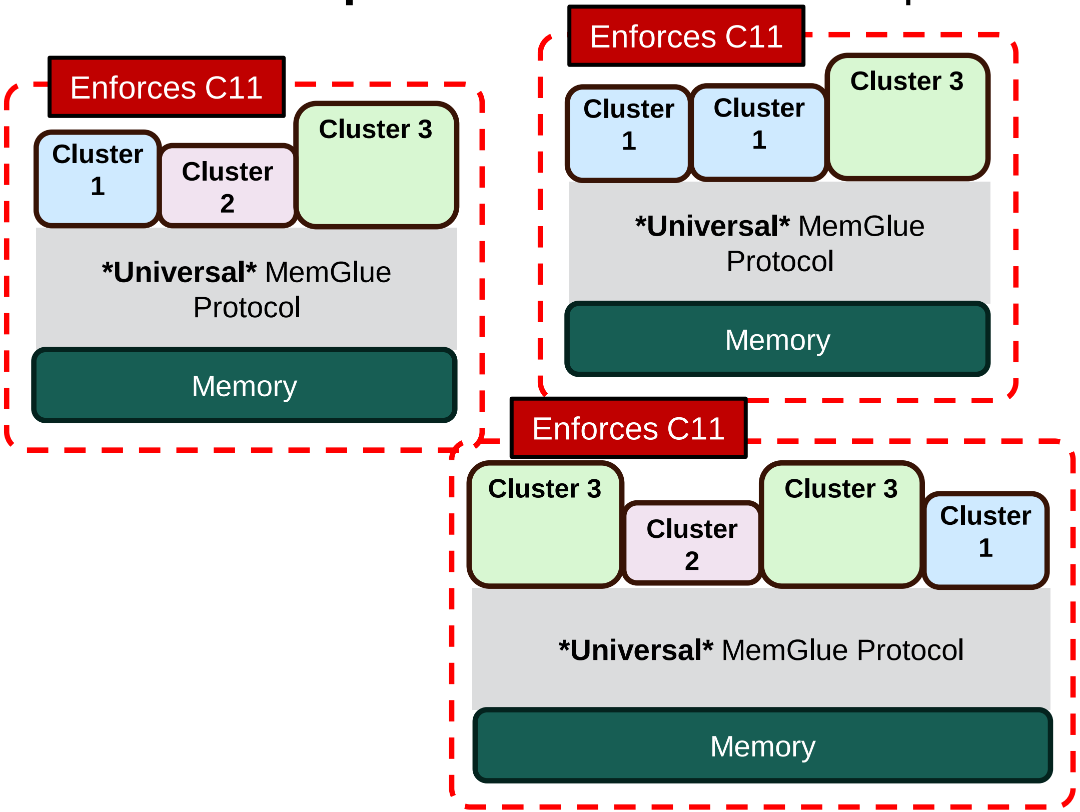
Principle 3: A heterogeneous cache coherence protocol should be **polite**.

- Any local MCM or protocol supported.
- Intra- and inter-cluster performance minimally restricted.



Principles 1 & 2: Modularity and Verifiability

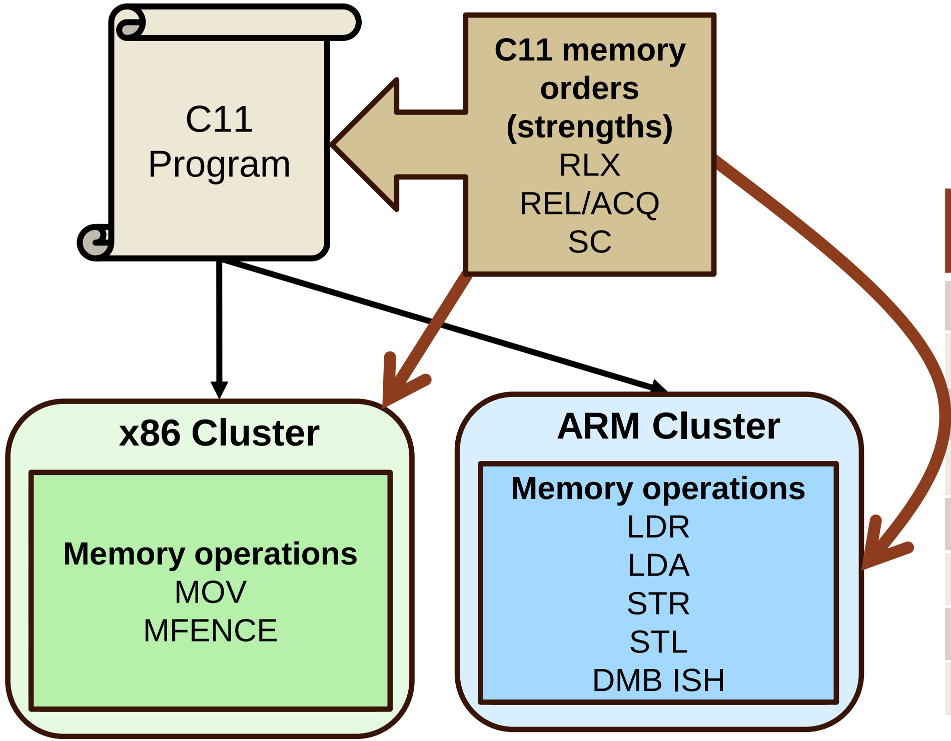
A **universal protocol** addresses Principles 1 & 2.



Key Insight: A universal protocol can be designed to target the **C11 MCM** as the system-wide MCM for any MemGlue-unified system.

Principles 1 & 2: Why C11?

C11 is the seminal **heterogeneous MCM.**

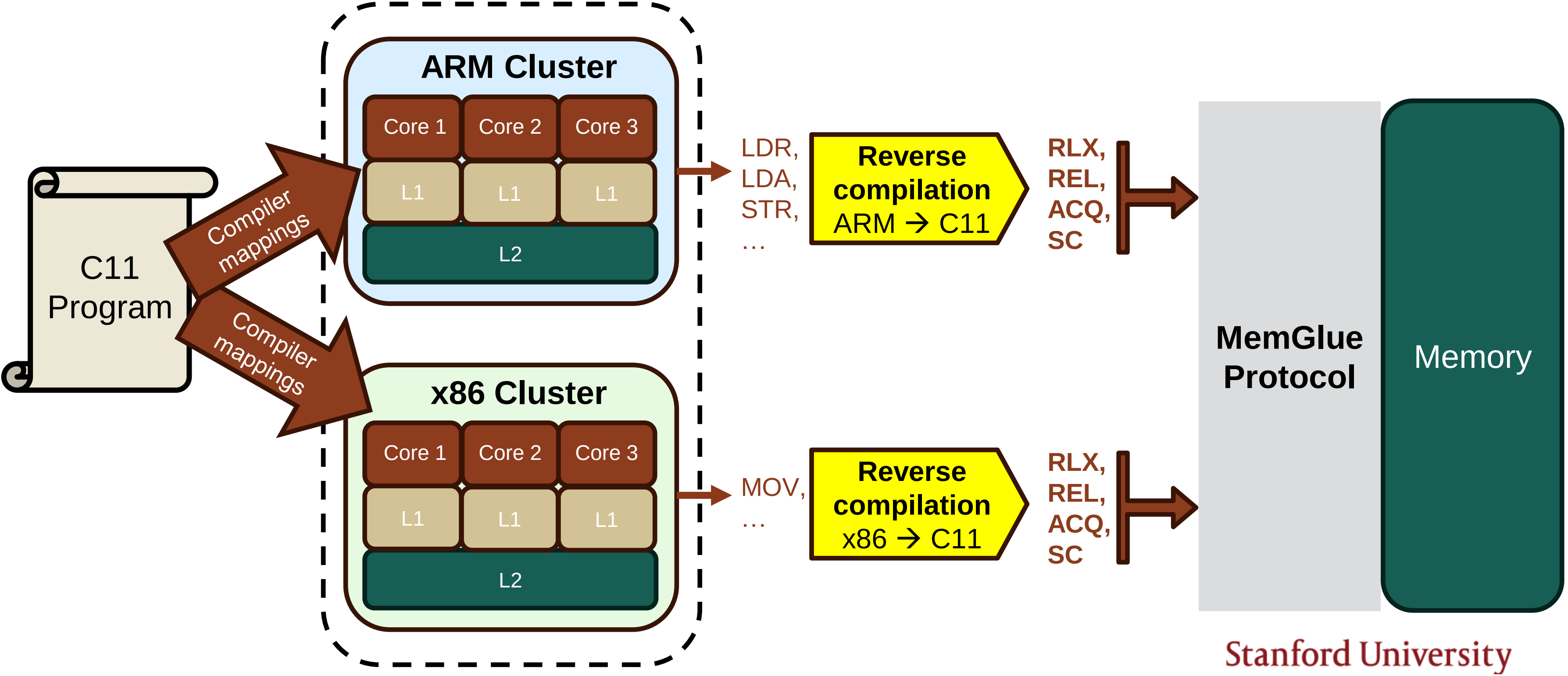


Formally verified compiler mappings translate C11 memory order to ISA instructions.

C/C++11 Operation	x86 Operation	ARM Operation
Load RLX	MOV (from mem)	LDR
Load ACQ	MOV (from mem)	LDA
Load SC	MOV (from mem)	LDA
Store RLX	MOV (to mem)	STR
Store REL	MOV (to mem)	STL
Store SC	MOV (to mem)	STL
Fence ACQ	<ignore>	DMB ISH LD
Fence REL/SC	MFENCE	DMB ISH

Principles 1 & 2: Leveraging the Heterogeneity of C11

MemGlue operates in the unified language of **C11 strengths.**



Principles 1 & 2: Defining C11 Strengths

MemGlue operates in the **unified language of C11 strengths.**

RLX

Few ordering requirements beyond coherence.

Thread 1	**Thread 2**
W_{rlx} **x** = 1	R_{rlx} **flag** = 1
W_{rlx} **flag** = 1	R_{rlx} **x** = 0

Allowed

REL/ACQ

Writes that happen-before a REL are visible to reads that happen-after an ACQ that reads from the REL.

Thread 1	**Thread 2**
W_{rel} **x** = 1	R_{acq} **flag** = 1
W_{rel} **flag** = 1	R_{acq} **x** = 0

Disallowed

SC

All SC instructions are totally ordered.

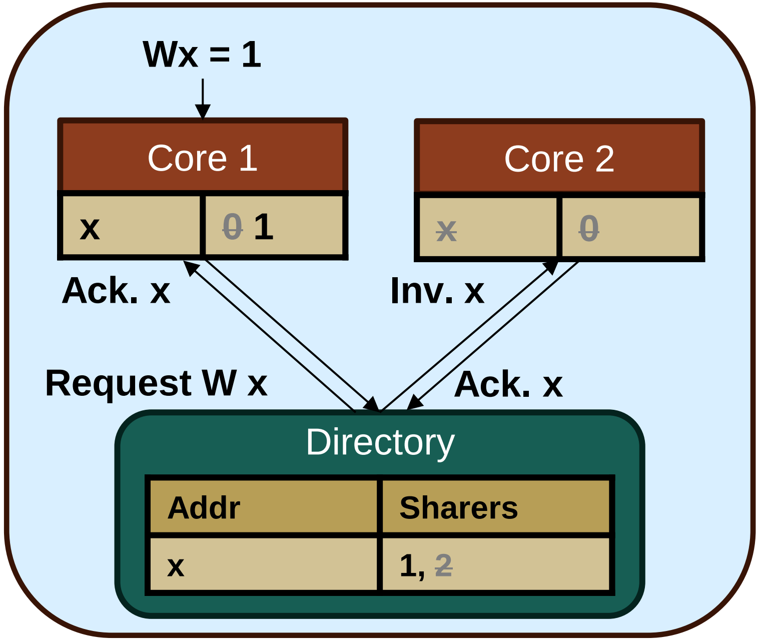
Thread 1	**Thread 2**
W_{sc} **x** = 1	R_{sc} **flag** = 1
W_{sc} **flag** = 1	R_{sc} **x** = 0

Disallowed

Principle 3: Politeness

Implementing MemGlue as an **update-based protocol** (as opposed to an invalidation-based protocol) addresses principle 2.

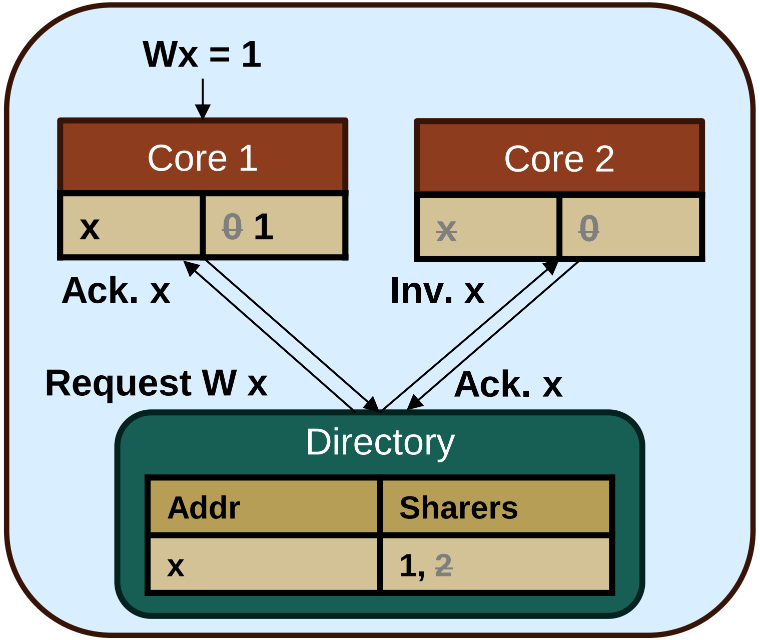
Invalidation-based:



Principle 3: Politeness

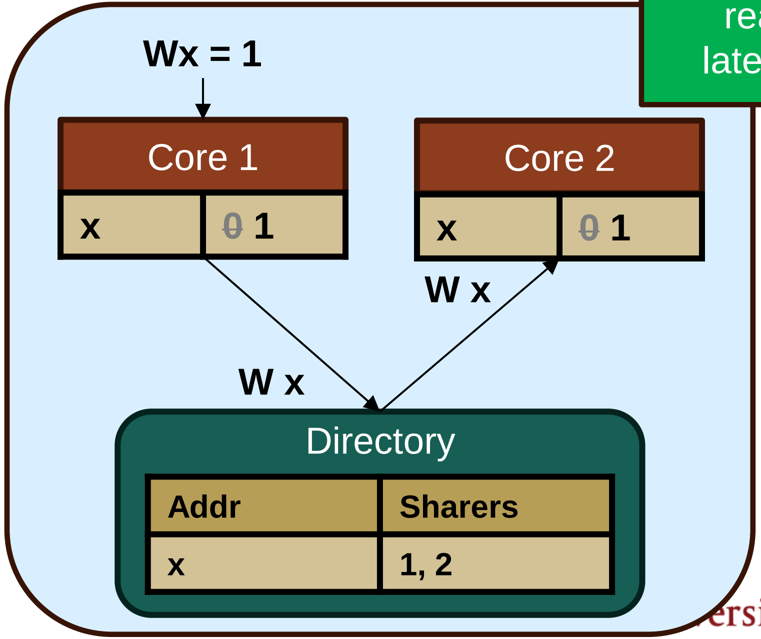
Implementing MemGlue as an **update-based protocol** (as opposed to an invalidation-based protocol) addresses principle 2.

Invalidation-based:



Addr	Sharers
x	1, ~~2~~

Update-based:



Addr	Sharers
x	1, 2

Pro: lower read latency

Principle 3: Politeness

Implementing MemGlue as an **update-based protocol** (as opposed to an invalidation-based protocol) addresses principle 2.

Requirement 1: should not restrict local cluster implementations.		**Requirement 2:** should not restrict inter-cluster performance.	
Allow any local coherence protocol.	Allow any local MCM.	Do not uphold single-write multiple reader invariant.	Performant under producer-consumer communication patterns.

[7] Liqun Cheng and John B Carter. Extending CC-Numa Systems to Support Write Update Optimizations.*SC* 2008.
[8] David B Glasco, Bruce A Delagi, and Michael J Flynn. Update-based cache coherence protocols for scalable shared-memory multiprocessors. *HICSS*, 1994.

Principle 3: Politeness

Implementing MemGlue as an **update-based protocol** (as opposed to an invalidation-based protocol) addresses principle 2.

	Requirement 1: should not restrict local cluster implementations.		**Requirement 2:** should not restrict inter-cluster performance.	
	Allow any local coherence protocol.	Allow any local MCM.	Do not uphold single-write multiple reader invariant.	Performant under producer-consumer communication patterns.
Update-based	✓	✓	✓	
Invalidation-based	✓	✓	✓	

[7] Liqun Cheng and John B Carter. Extending CC-Numa Systems to Support Write Update Optimizations.*SC* 2008.
[8] David B Glasco, Bruce A Delagi, and Michael J Flynn. Update-based cache coherence protocols for scalable shared-memory multiprocessors. *HICSS*, 1994.

Roadmap

MemGlue Design Principles

Ordered MemGlue (Ordered Interconnect Network)

Unordered MemGlue (Unordered Interconnect Network)

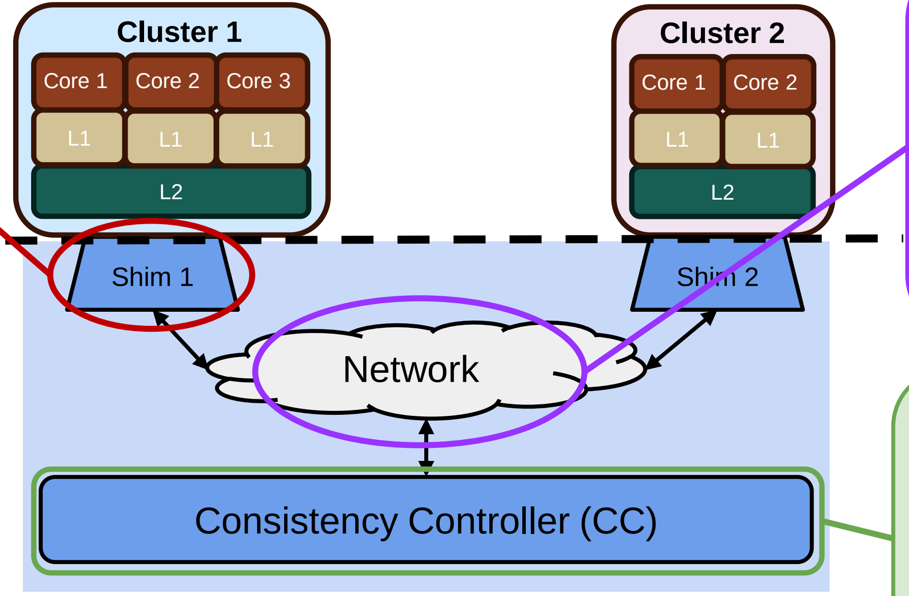
Experimental Evaluation & Results

- Bounded proof of correctness (litmus testing)
- Complete proof of correctness (manual)

MemGlue Overview: Hardware Structures

Shims are responsible for:

1. Reverse compiling cluster instructions to their C11-style analog.
2. Sending writes and read requests on behalf of their cluster.
3. Receiving and propagate write updates from the CC.



Ordered MemGlue: messages between shims and CC arrive in the order they were sent. **Unordered MemGlue:** messages may be reordered by the network.

The Consistency Concroller (CC):

1. Forwards write updates to the necessary shims.
2. Supplies the most up-to-date data on read misses

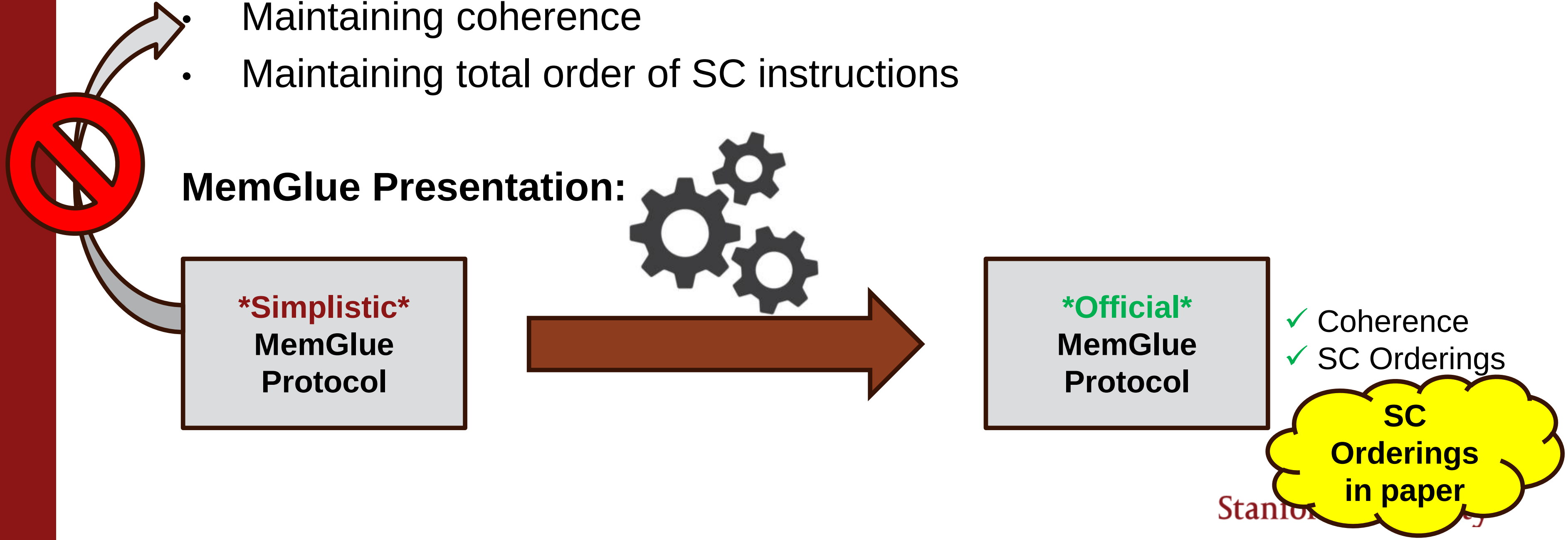
Ordered MemGlue Overview: Challenges

Goal: uphold the C11 MCM for any execution of any program

Primary Challenges:

- Maintaining coherence
- Maintaining total order of SC instructions

MemGlue Presentation:



***Simplistic* MemGlue Protocol**

***Official* MemGlue Protocol**

✓ Coherence

✓ SC Orderings

SC Orderings in paper

Ordered MemGlue By Example

Shim 1

Address	Valid (V/I)	Sync Bit	TS
x			
y			

Shim 2

Address	Valid (V/I)	Sync Bit	TS
x			
y			

Consistency Controller

Address	TS	Data	Sharers
x			
y			

Ordered MemGlue By Example

Shim 1

Address	Valid (V/I)	Data
x	I	
y	I	

Simplistic MemGlue

- Omit metadata
- Omit C11 strengths

Shim 2

Address	Valid (V/I)	Data
x	V	0
y	I	

Shim integration in paper

Consistency Controller

Address	Data	Sharers
x	0	2
y	0	

Ordered MemGlue By Example: Cluster Writes

Wx = 1

Shim 1

Address	Valid (V/I)	Data
x	I	
y	I	

Shim 2

Address	Valid (V/I)	Data
x	V	0
y	I	

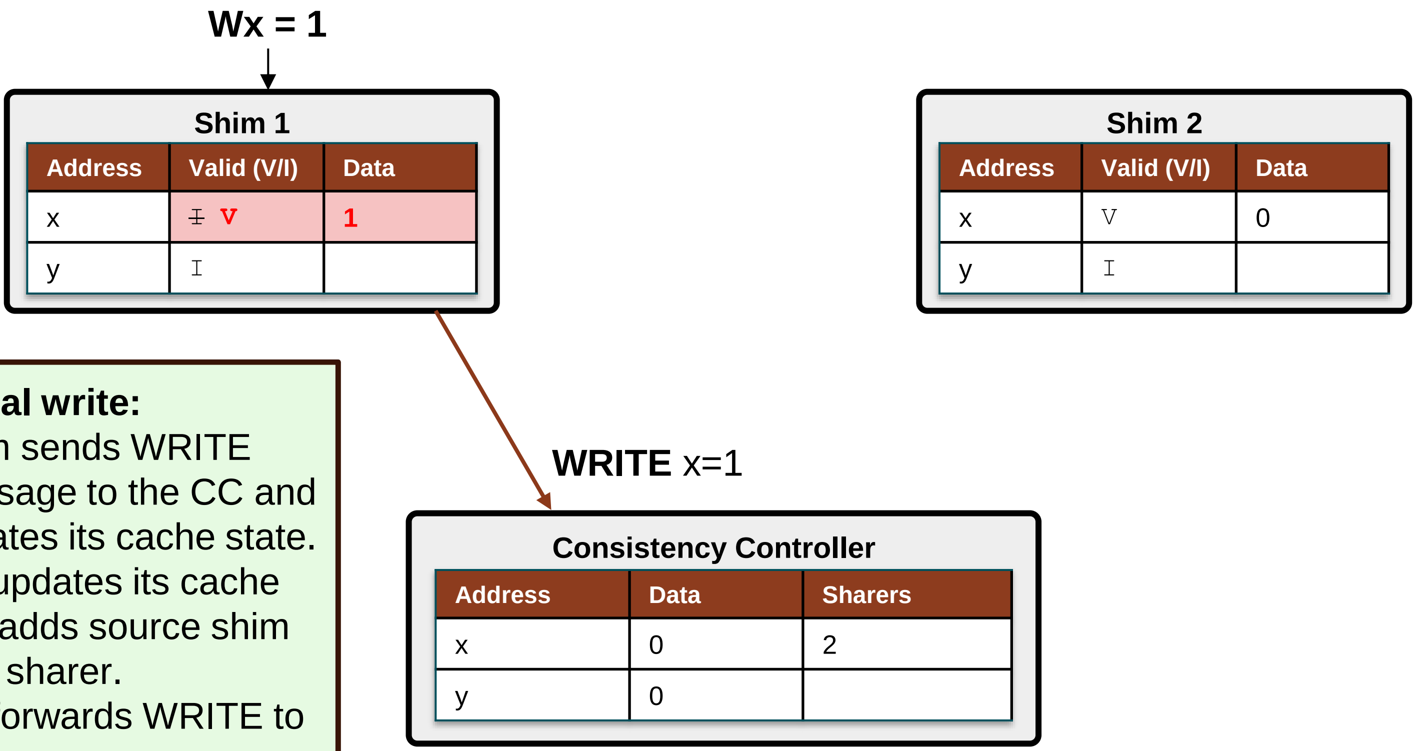
On a local write:

1. Shim sends WRITE message to the CC and updates its cache state.

Consistency Controller

Address	Data	Sharers
x	0	2
y	0	

Ordered MemGlue By Example: Cluster Writes



Shim 1

Address	Valid (V/I)	Data
x	~~I~~ V	1
y	I	

Shim 2

Address	Valid (V/I)	Data
x	V	0
y	I	

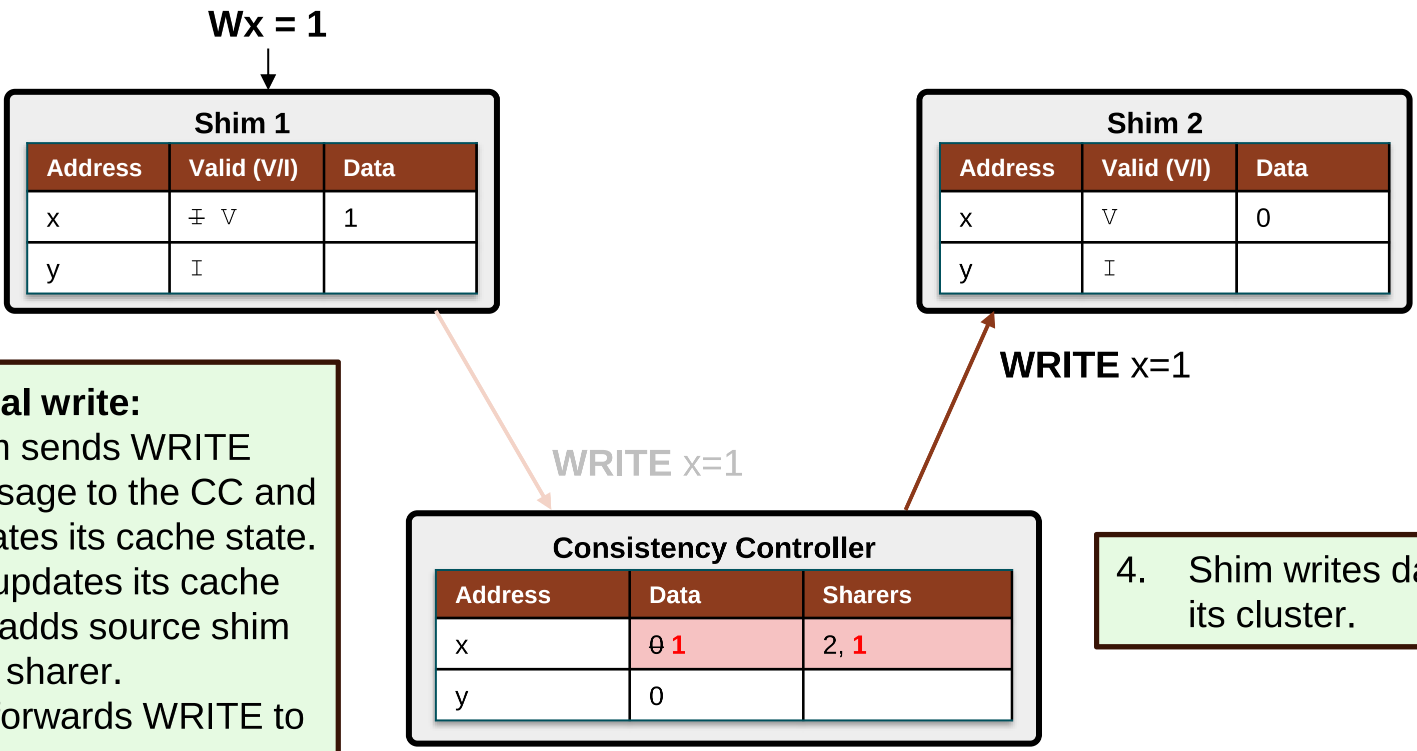
Consistency Controller

Address	Data	Sharers
x	0	2
y	0	

On a local write:

1. Shim sends WRITE message to the CC and updates its cache state.
2. CC updates its cache and adds source shim as a sharer.
3. CC forwards WRITE to sharers.

Ordered MemGlue By Example: Cluster Writes



Shim 1

Address	Valid (V/I)	Data
x	~~I~~ V	1
y	I	

Shim 2

Address	Valid (V/I)	Data
x	V	0
y	I	

Consistency Controller

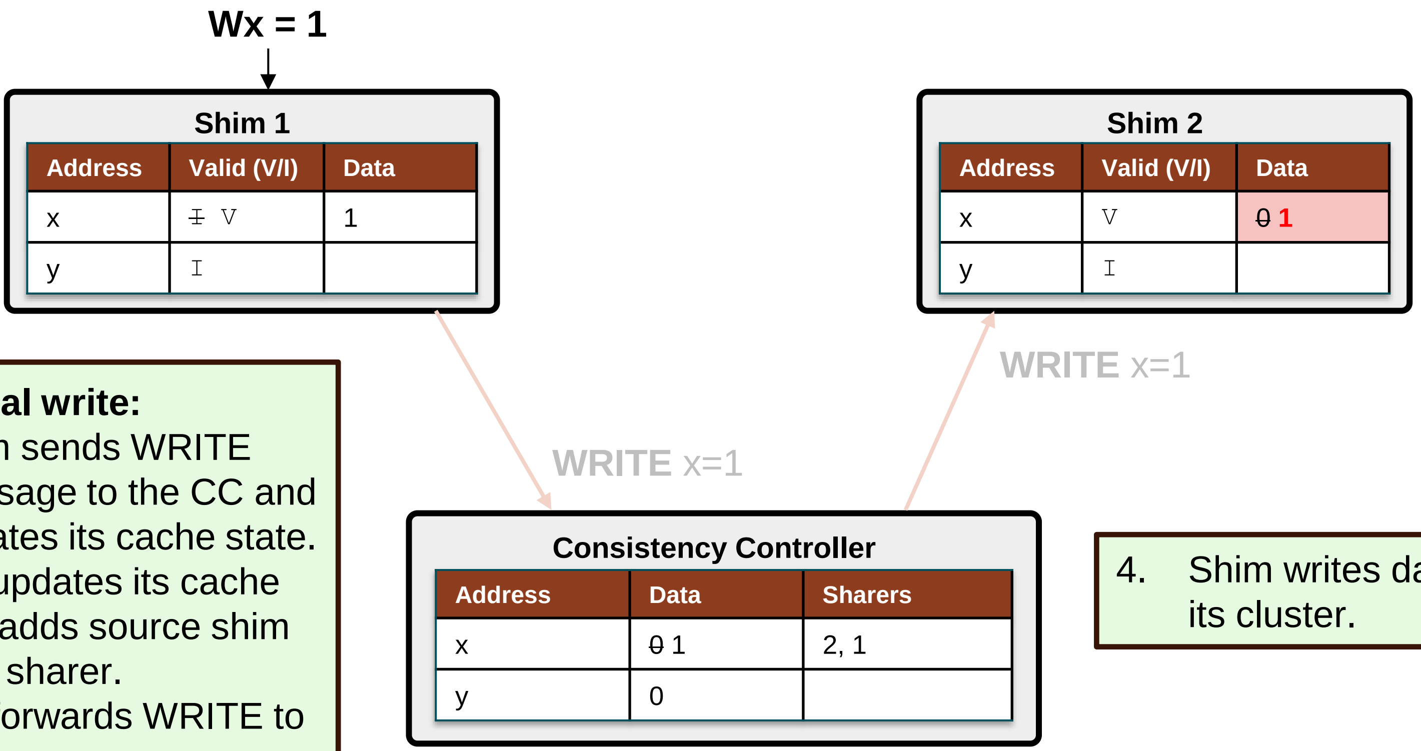
Address	Data	Sharers
x	~~0~~ 1	2, 1
y	0	

On a local write:

1. Shim sends WRITE message to the CC and updates its cache state.
2. CC updates its cache and adds source shim as a sharer.
3. CC forwards WRITE to sharers.

4. Shim writes data within its cluster.

Ordered MemGlue By Example: Cluster Writes



Shim 1

Address	Valid (V/I)	Data
x	~~I~~ V	1
y	I	

Shim 2

Address	Valid (V/I)	Data
x	V	~~0~~ 1
y	I	

Consistency Controller

Address	Data	Sharers
x	~~0~~ 1	2, 1
y	0	

On a local write:

1. Shim sends WRITE message to the CC and updates its cache state.
2. CC updates its cache and adds source shim as a sharer.
3. CC forwards WRITE to sharers.

4. Shim writes data within its cluster.

Ordered MemGlue By Example: Cluster Read Hits

R R ~~x~~ 1

Shim 1

Address	Valid (V/I)	Data
x	~~I~~ V	1
y	I	

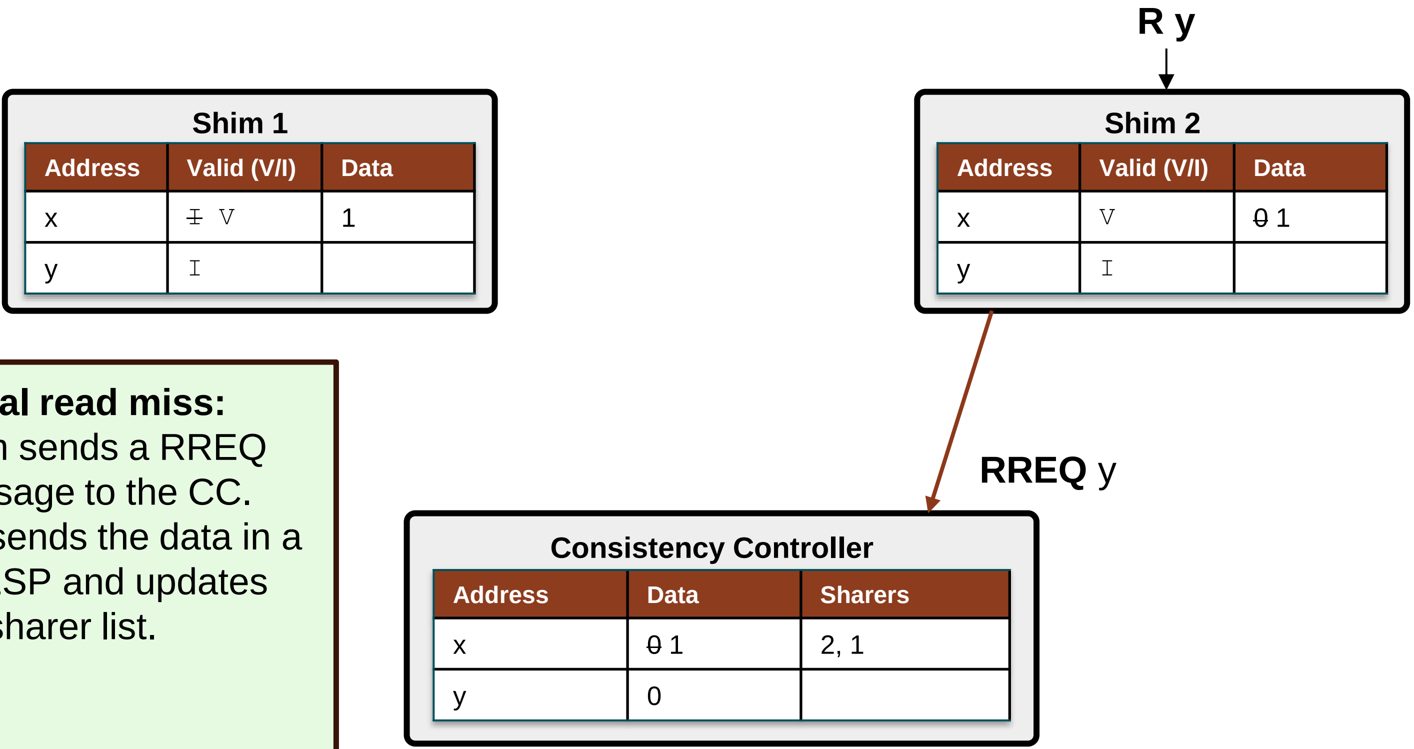
Shim 2

Address	Valid (V/I)	Data
x	V	~~0~~ 1
y	I	

Consistency Controller

Address	Data	Sharers
x	~~0~~ 1	2, 1
y	0	

Ordered MemGlue By Example: Cluster Read Misses



Shim 1

Address	Valid (V/I)	Data
x	~~I~~ V	1
y	I	

Shim 2

Address	Valid (V/I)	Data
x	V	~~0~~ 1
y	I	

Consistency Controller

Address	Data	Sharers
x	~~0~~ 1	2, 1
y	0	

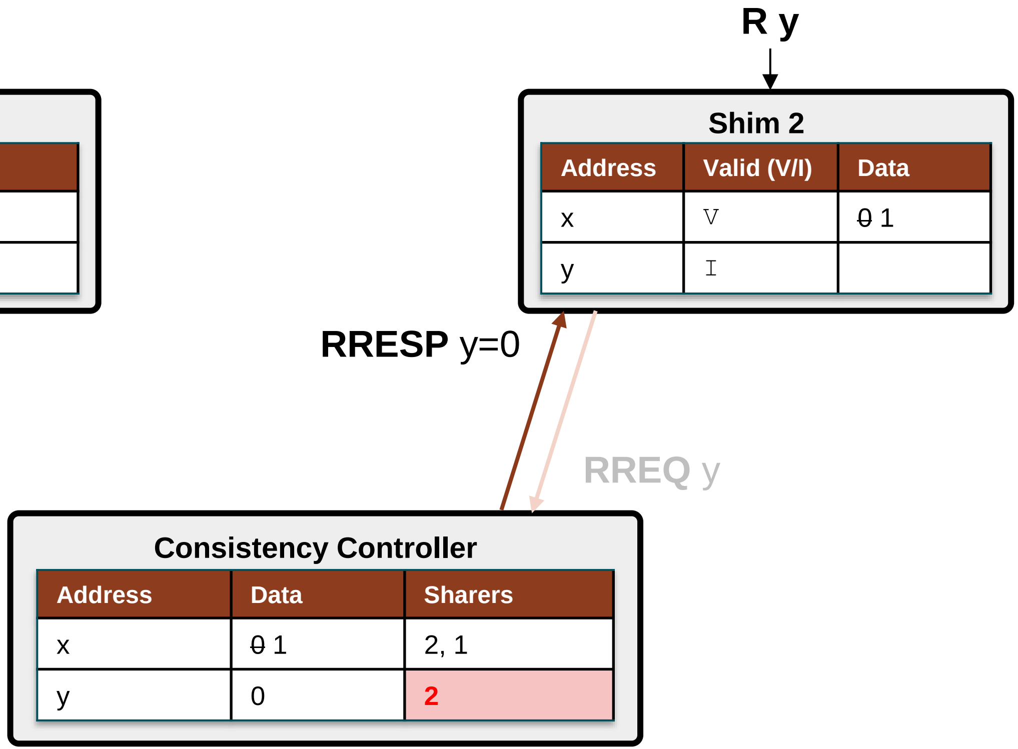
On a local read miss:

1. Shim sends a RREQ message to the CC.
2. CC sends the data in a RRESP and updates the sharer list.

Ordered MemGlue By Example: Cluster Read Misses

Shim 1

Address	Valid (V/I)	Data
x	~~I~~ V	1
y	I	



Shim 2

Address	Valid (V/I)	Data
x	V	~~0~~ 1
y	I	

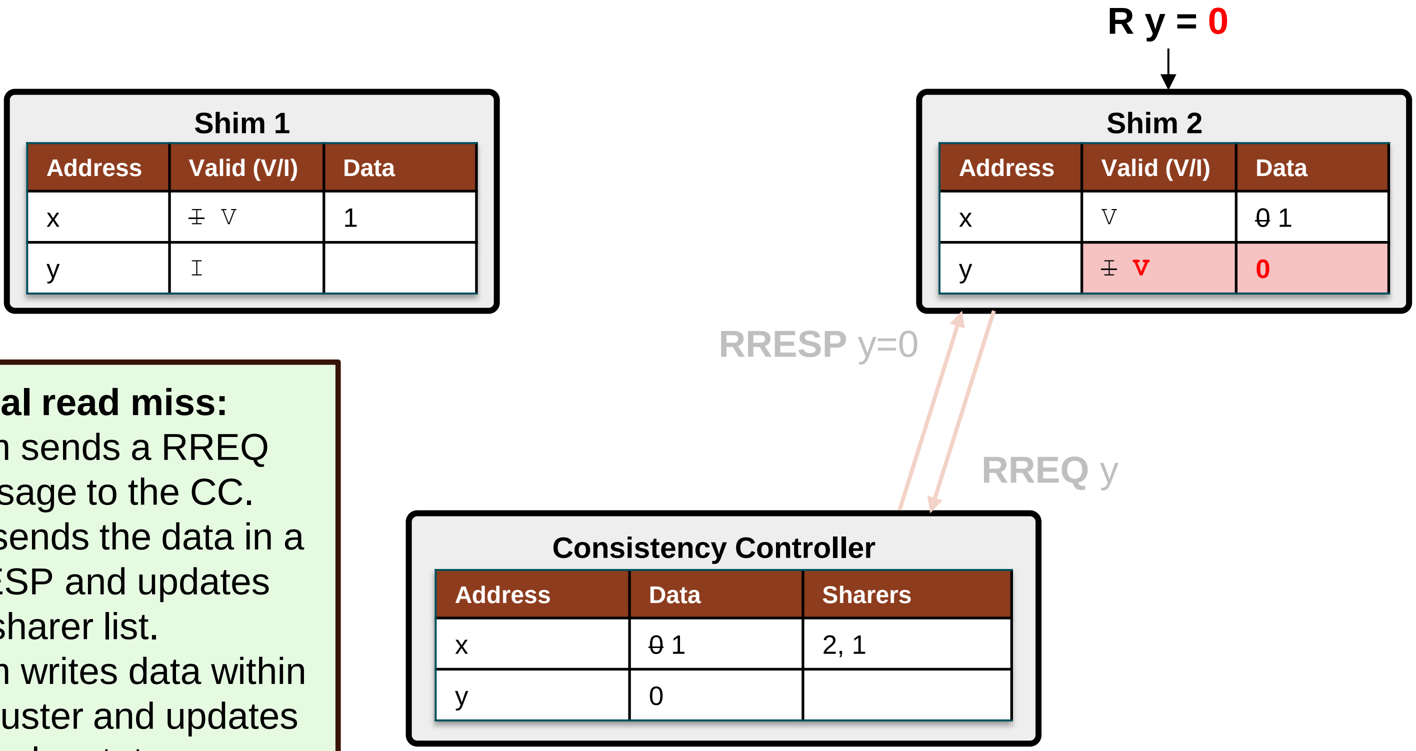
Consistency Controller

Address	Data	Sharers
x	~~0~~ 1	2, 1
y	0	2

On a local read miss:

1. Shim sends a RREQ message to the CC.
2. CC sends the data in a RRESP and updates the sharer list.
3. Shim writes data within its cluster and updates its cache state.

Ordered MemGlue By Example: Cluster Read Misses



R y = 0

Shim 1

Address	Valid (V/I)	Data
x	~~I~~ V	1
y	I	

Shim 2

Address	Valid (V/I)	Data
x	V	~~0~~ 1
y	~~I~~ V	0

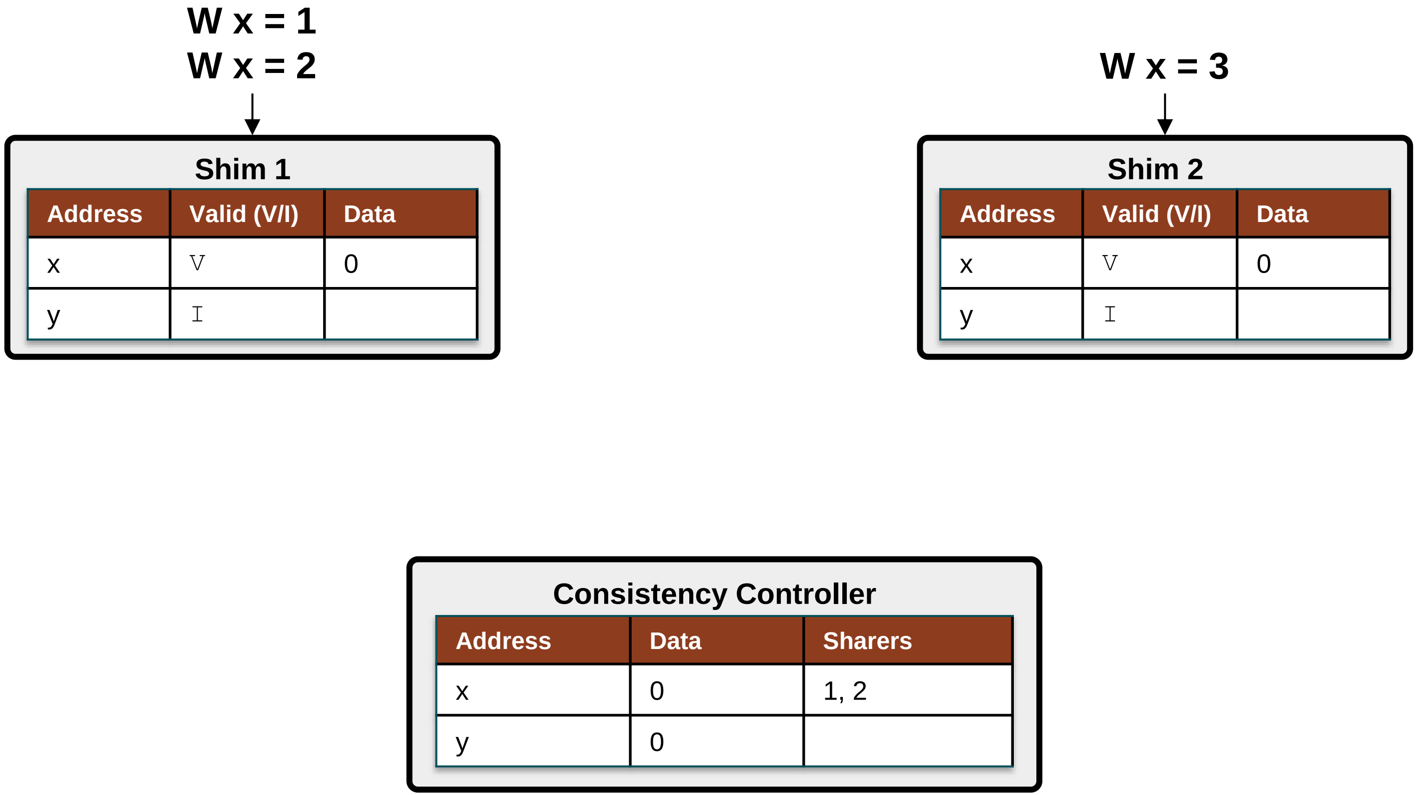
Consistency Controller

Address	Data	Sharers
x	~~0~~ 1	2, 1
y	0	

On a local read miss:

1. Shim sends a RREQ message to the CC.
2. CC sends the data in a RRESP and updates the sharer list.
3. Shim writes data within its cluster and updates its cache state.

Ordered MemGlue By Example: Timestamps



W x = 1
W x = 2

Shim 1

Address	Valid (V/I)	Data
x	V	0
y	I	

W x = 3

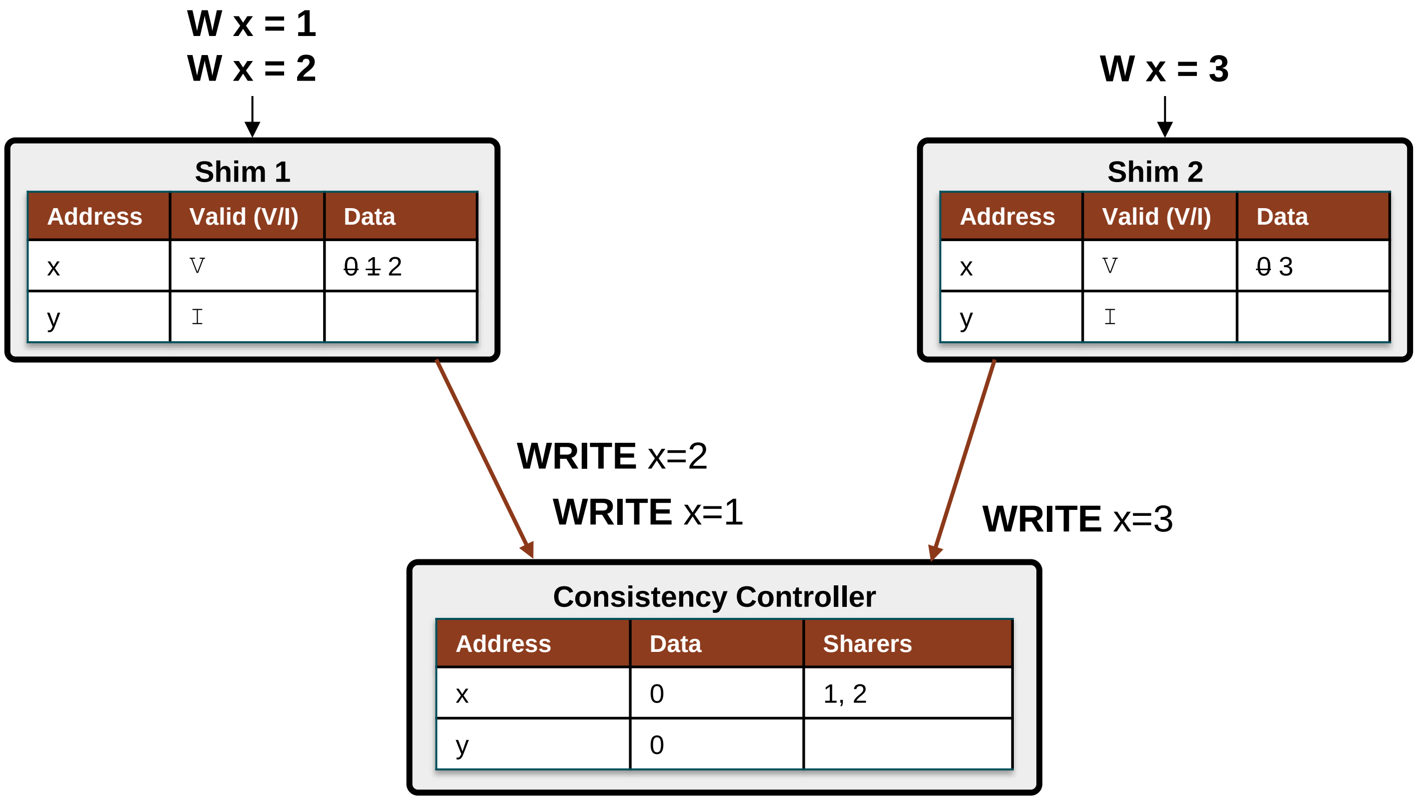
Shim 2

Address	Valid (V/I)	Data
x	V	0
y	I	

Consistency Controller

Address	Data	Sharers
x	0	1, 2
y	0	

Ordered MemGlue By Example: Timestamps



Shim 1

Address	Valid (V/I)	Data
x	V	~~0~~ ~~1~~ 2
y	I	

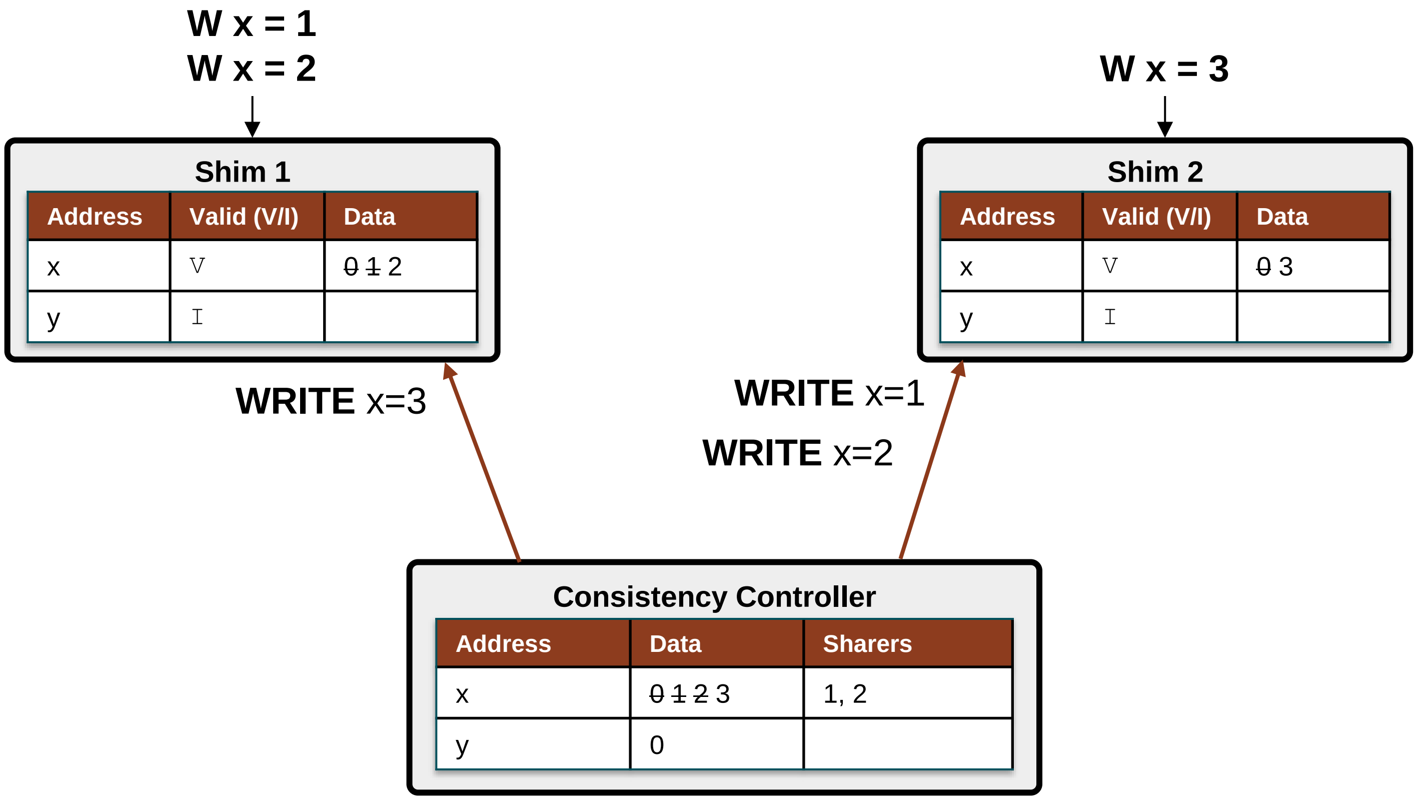
Shim 2

Address	Valid (V/I)	Data
x	V	~~0~~ 3
y	I	

Consistency Controller

Address	Data	Sharers
x	0	1, 2
y	0	

Ordered MemGlue By Example: Timestamps



W x = 1
W x = 2

Shim 1

Address	Valid (V/I)	Data
x	V	~~0~~ ~~1~~ 2
y	I	

W x = 3

Shim 2

Address	Valid (V/I)	Data
x	V	~~0~~ 3
y	I	

WRITE x=3

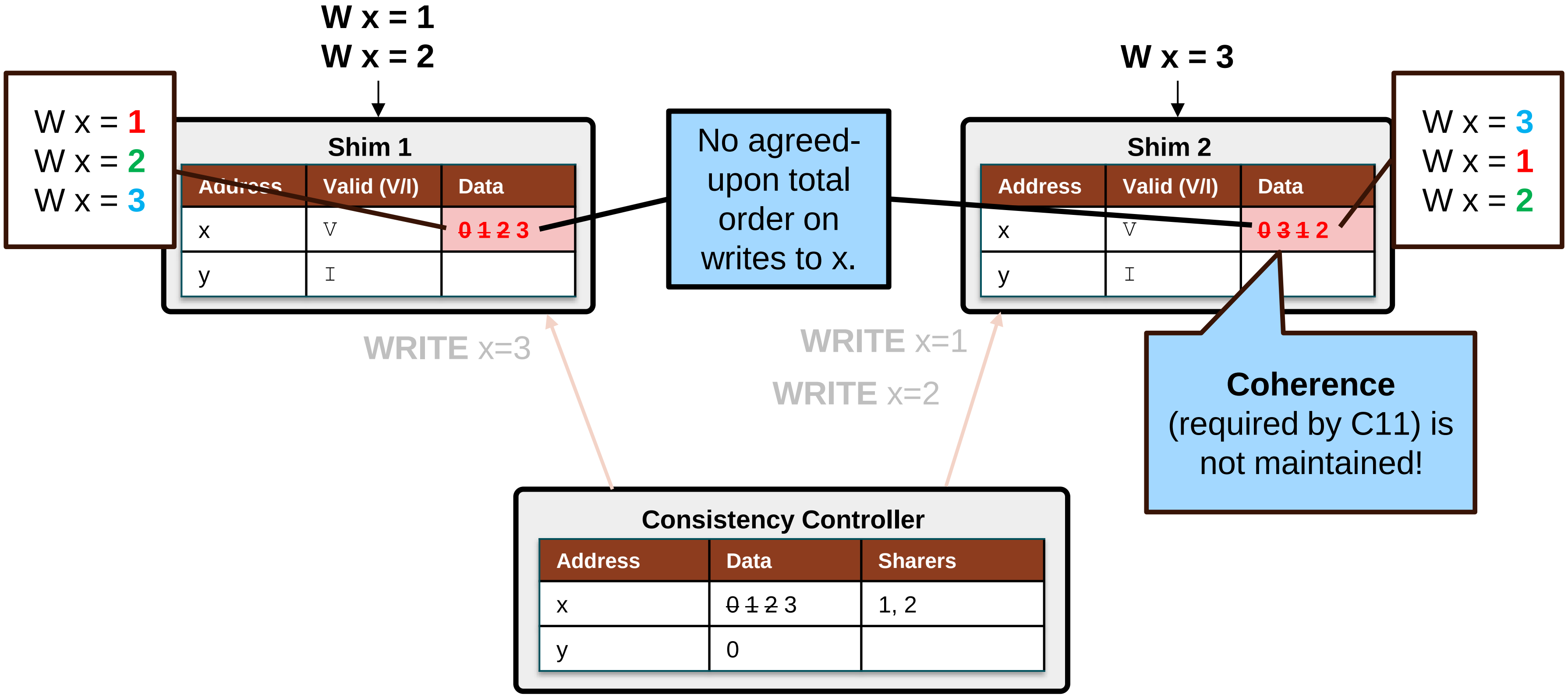
WRITE x=1

WRITE x=2

Consistency Controller

Address	Data	Sharers
x	~~0~~ ~~1~~ ~~2~~ 3	1, 2
y	0	

Ordered MemGlue By Example: Timestamps



W x = 1
W x = 2

W x = 3
W x = 2

W x = 3
W x = 1
W x = 2

Shim 1

Address	Valid (V/I)	Data
x	V	~~0~~ ~~1~~ ~~2~~ 3
y	I	

No agreed-upon total order on writes to x.

Shim 2

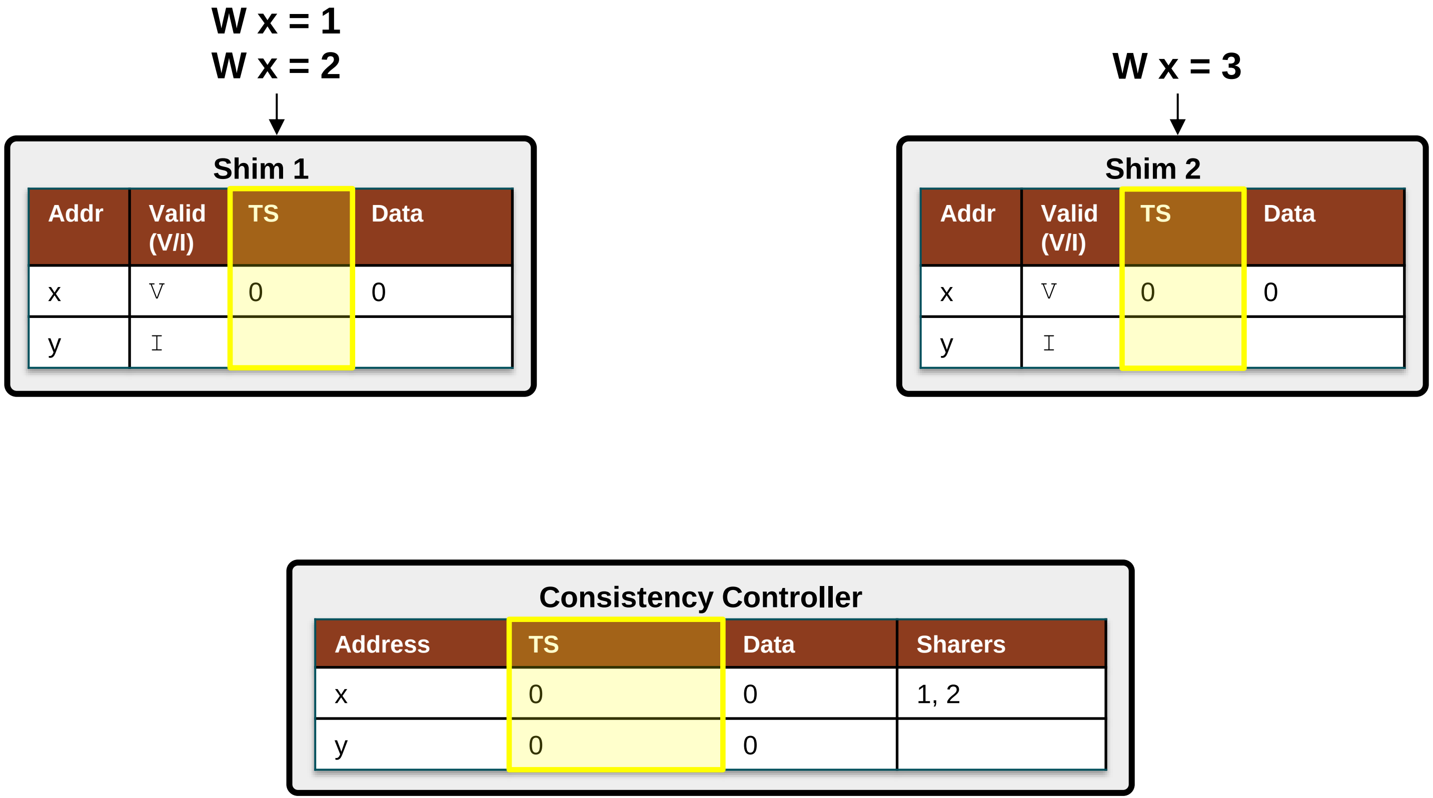
Address	Valid (V/I)	Data
x	V	~~0~~ ~~3~~ ~~1~~ 2
y	I	

Coherence (required by C11) is not maintained!

Consistency Controller

Address	Data	Sharers
x	~~0~~ ~~1~~ ~~2~~ 3	1, 2
y	0	

Ordered MemGlue By Example: Timestamps



W x = 1
W x = 2

Shim 1

Addr	Valid (V/I)	TS	Data
x	V	0	0
y	I		

W x = 3

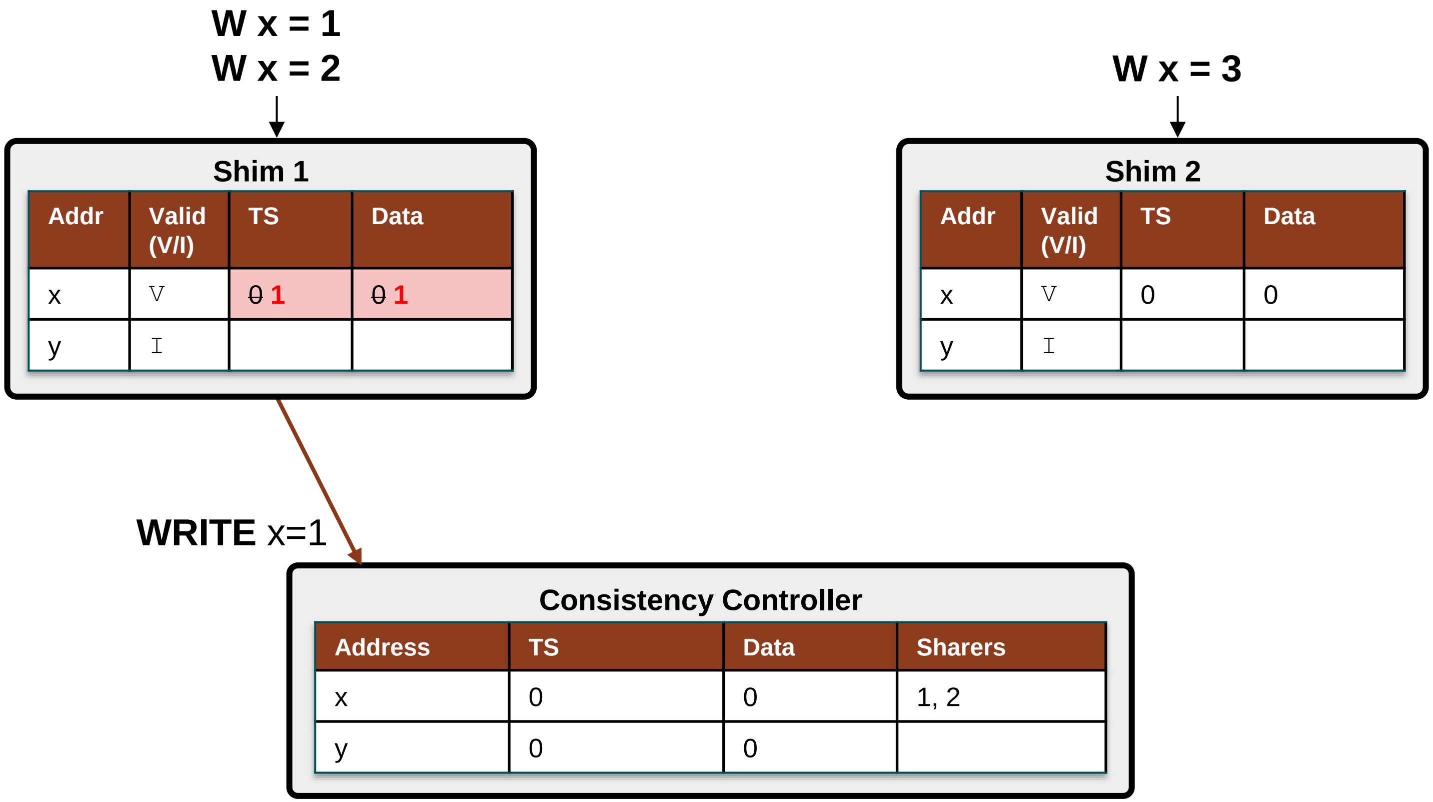
Shim 2

Addr	Valid (V/I)	TS	Data
x	V	0	0
y	I		

Consistency Controller

Address	TS	Data	Sharers
x	0	0	1, 2
y	0	0	

Ordered MemGlue By Example: Timestamps



W x = 1
W x = 2

W x = 3

Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ 1	~~0~~ 1
y	I		

Shim 2

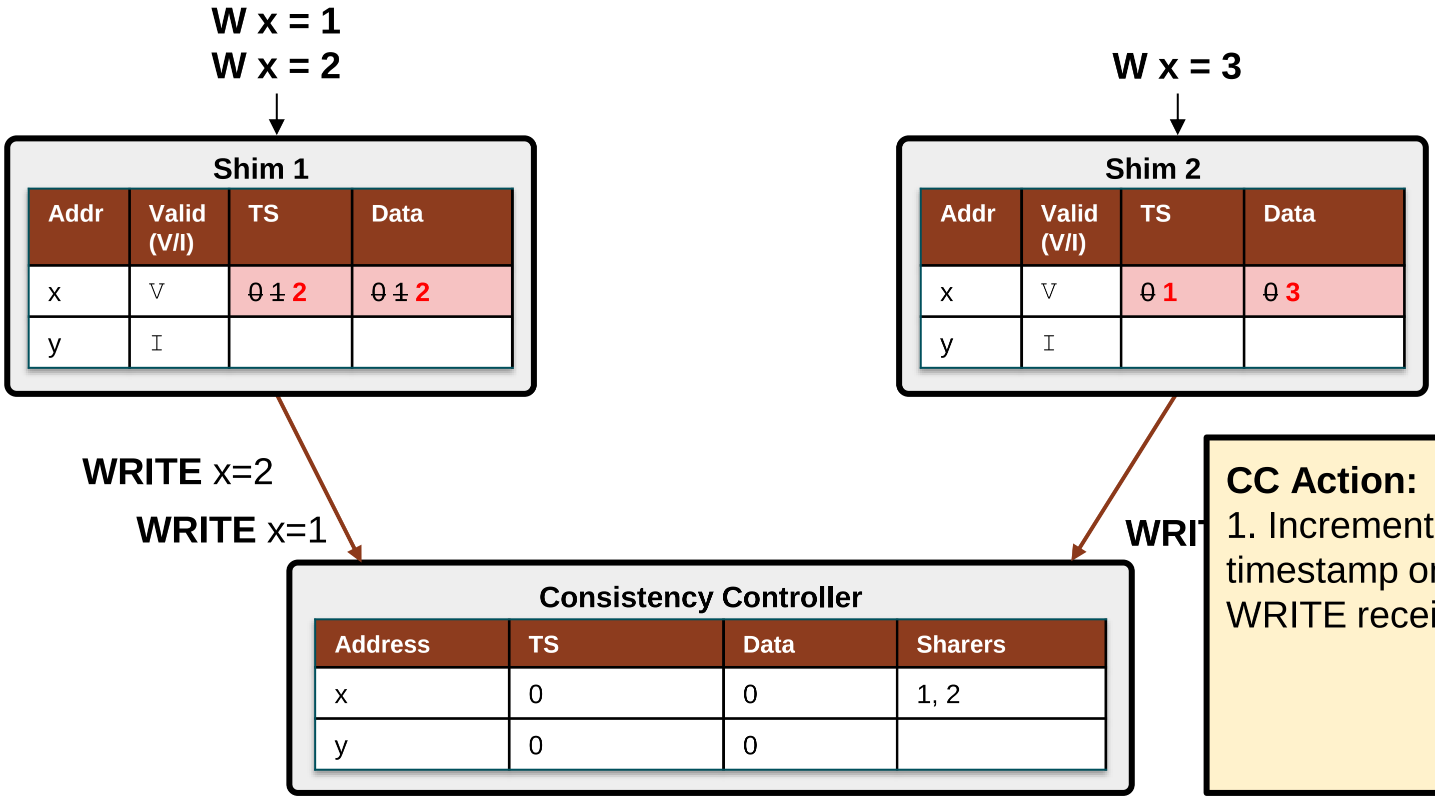
Addr	Valid (V/I)	TS	Data
x	V	0	0
y	I		

WRITE x=1

Consistency Controller

Address	TS	Data	Sharers
x	0	0	1, 2
y	0	0	

Ordered MemGlue By Example: Timestamps



Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ ~~1~~ 2	~~0~~ ~~1~~ 2
y	I		

Shim 2

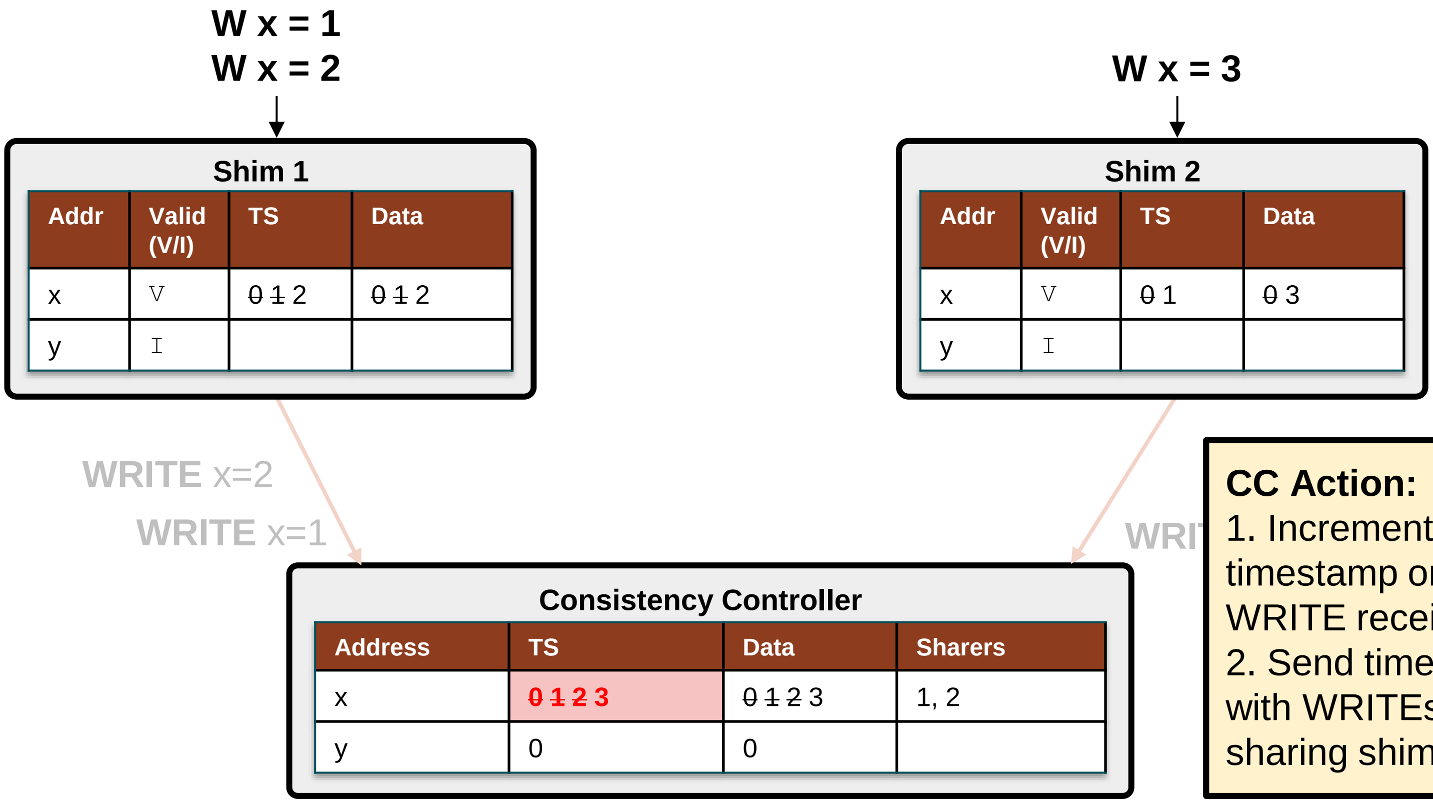
Addr	Valid (V/I)	TS	Data
x	V	~~0~~ 1	~~0~~ 3
y	I		

Consistency Controller

Address	TS	Data	Sharers
x	0	0	1, 2
y	0	0	

CC Action:
1. Increment timestamp on each WRITE received.

Ordered MemGlue By Example: Timestamps



Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ ~~1~~ 2	~~0~~ ~~1~~ 2
y	I		

Shim 2

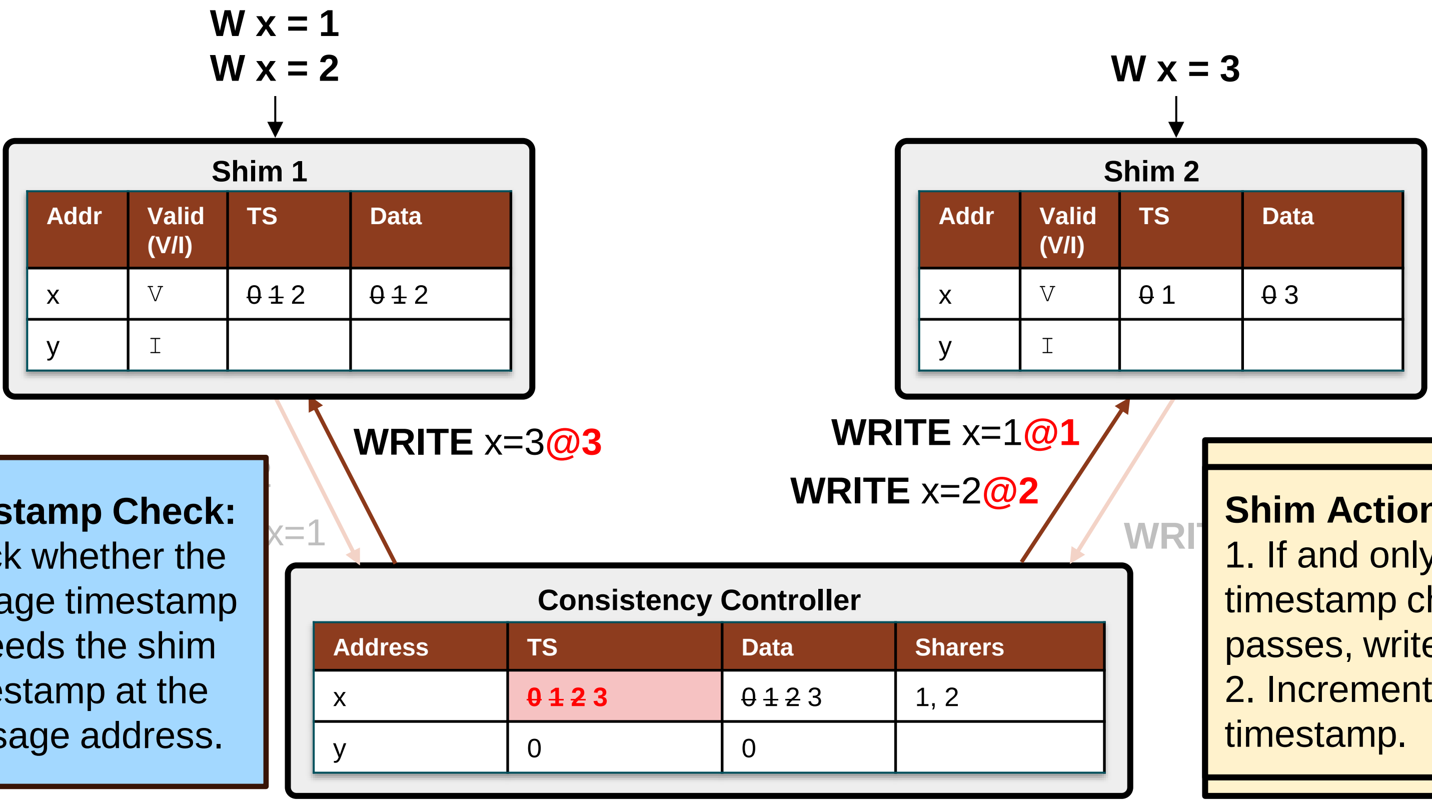
Addr	Valid (V/I)	TS	Data
x	V	~~0~~ 1	~~0~~ 3
y	I		

Consistency Controller

Address	TS	Data	Sharers
x	~~0~~ ~~1~~ ~~2~~ 3	~~0~~ ~~1~~ ~~2~~ 3	1, 2
y	0	0	

CC Action:
1. Increment timestamp on each WRITE received.
2. Send timestamp with WRITEs to sharing shims.

Ordered MemGlue By Example: Timestamps



Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ ~~1~~ 2	~~0~~ ~~1~~ 2
y	I		

Shim 2

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ 1	~~0~~ 3
y	I		

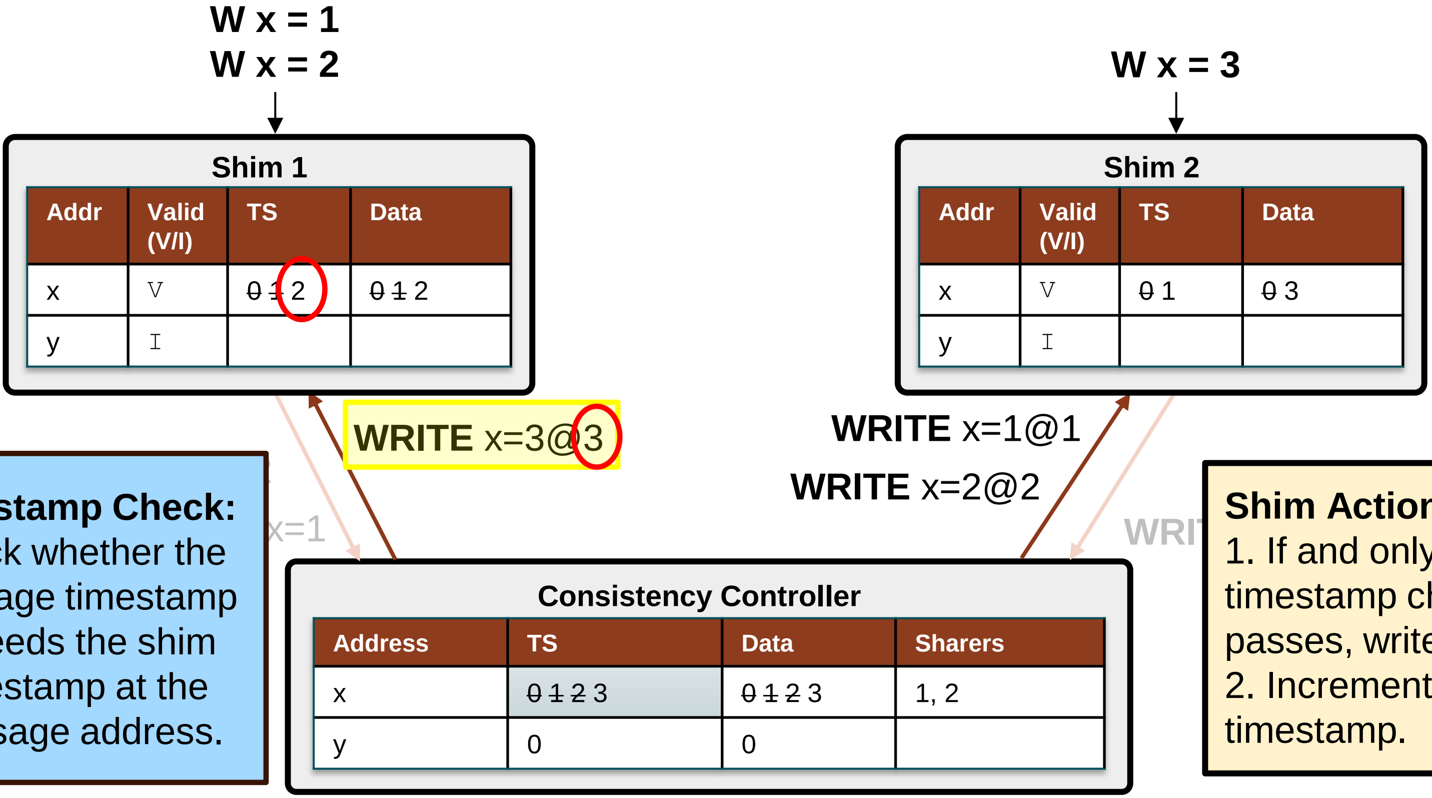
Consistency Controller

Address	TS	Data	Sharers
x	~~0~~ ~~1~~ ~~2~~ 3	~~0~~ ~~1~~ ~~2~~ 3	1, 2
y	0	0	

Timestamp Check: Check whether the message timestamp exceeds the shim timestamp at the message address.

Shim Action:
1. If and only if the timestamp check passes, write the data.
2. Increment the shim timestamp.

Ordered MemGlue By Example: Timestamps



W x = 1
W x = 2

W x = 3

Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ ~~1~~ 2	~~0~~ ~~1~~ 2
y	I		

Shim 2

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ 1	~~0~~ 3
y	I		

WRITE x=3@3

WRITE x=1@1

WRITE x=2@2

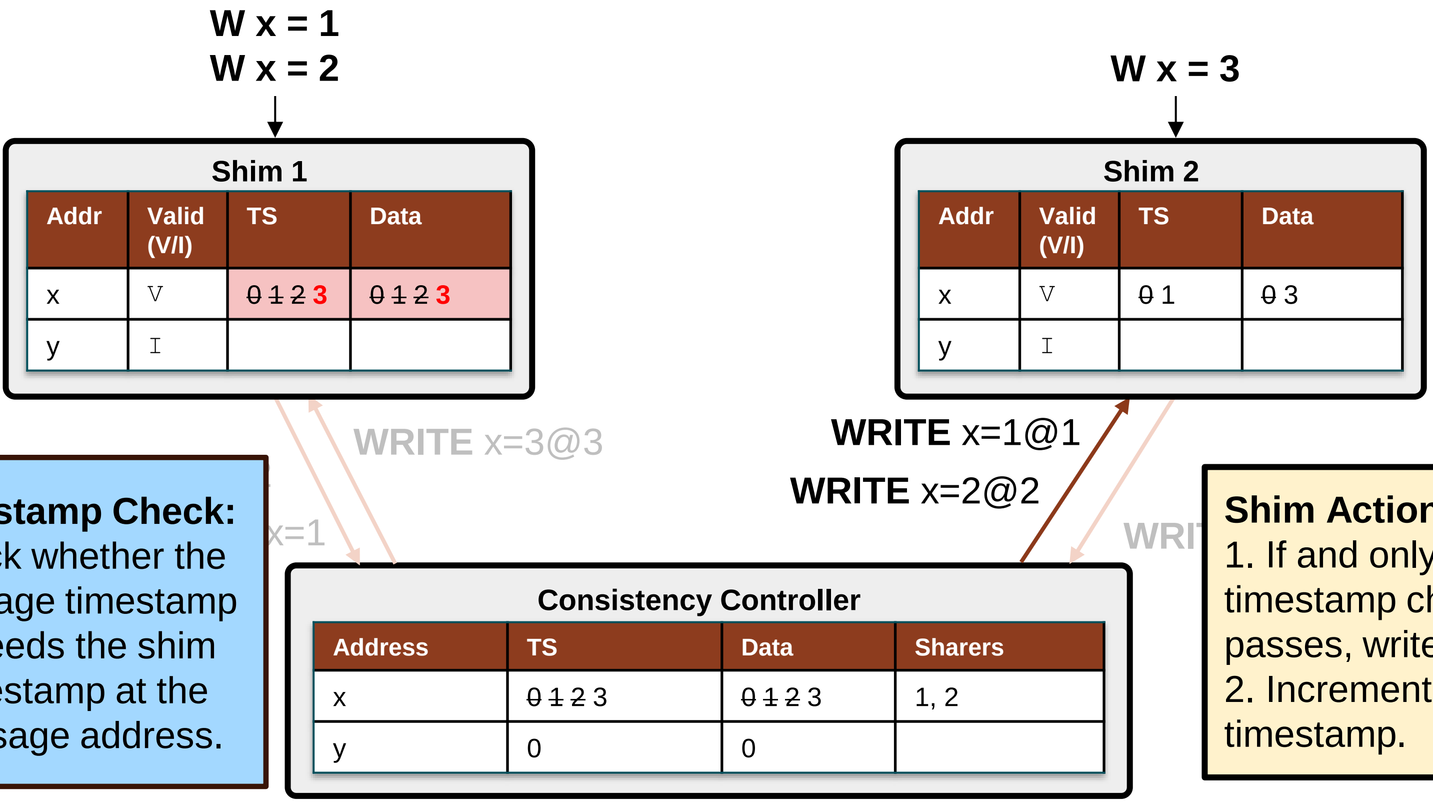
Consistency Controller

Address	TS	Data	Sharers
x	~~0~~ ~~1~~ ~~2~~ 3	~~0~~ ~~1~~ ~~2~~ 3	1, 2
y	0	0	

Timestamp Check: Check whether the message timestamp exceeds the shim timestamp at the message address.

Shim Action:
1. If and only if the timestamp check passes, write the data.
2. Increment the shim timestamp.

Ordered MemGlue By Example: Timestamps



Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ ~~1~~ ~~2~~ 3	~~0~~ ~~1~~ ~~2~~ 3
y	I		

Shim 2

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ 1	~~0~~ 3
y	I		

Consistency Controller

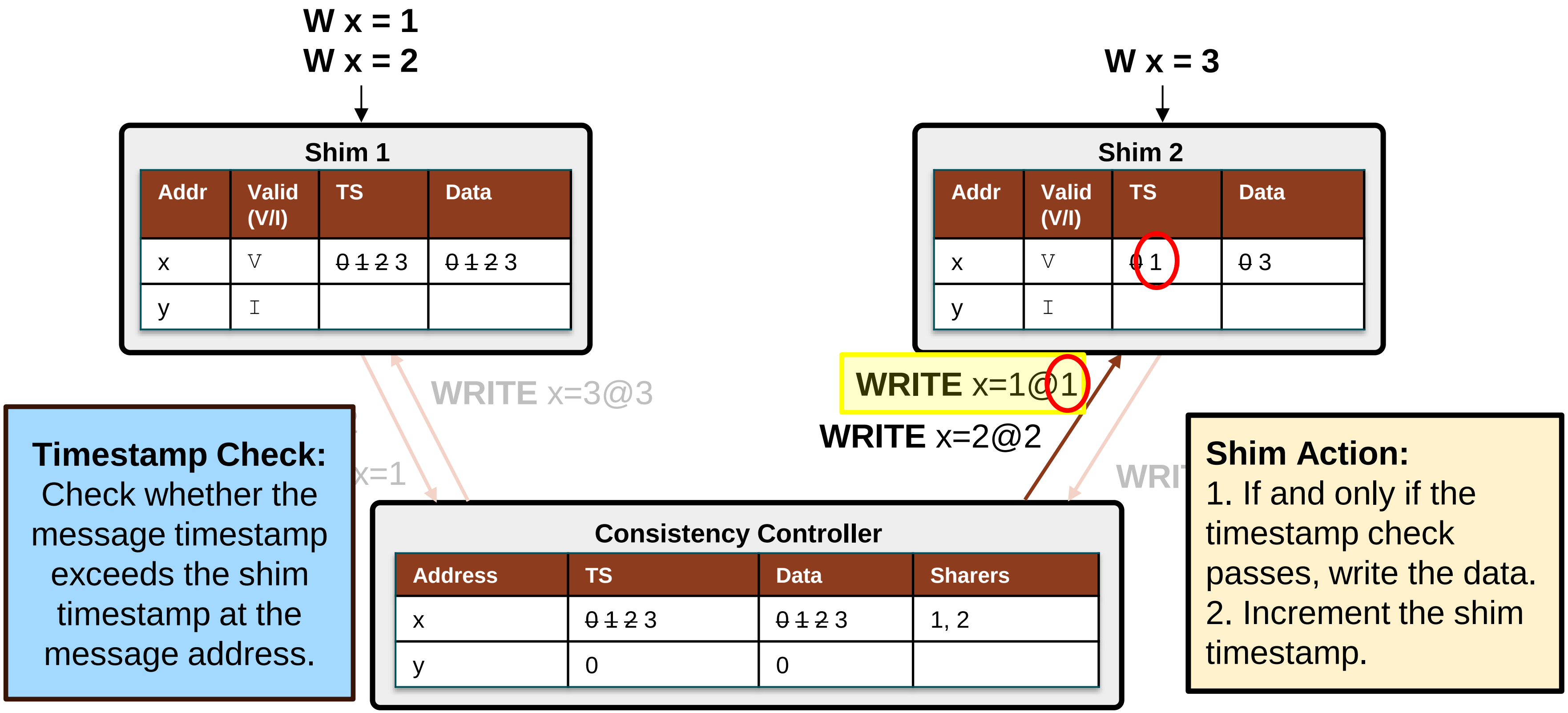
Address	TS	Data	Sharers
x	~~0~~ ~~1~~ ~~2~~ 3	~~0~~ ~~1~~ ~~2~~ 3	1, 2
y	0	0	

Timestamp Check: Check whether the message timestamp exceeds the shim timestamp at the message address.

Shim Action:
1. If and only if the timestamp check passes, write the data.
2. Increment the shim timestamp.

Stanford University

Ordered MemGlue By Example: Timestamps



W x = 1
W x = 2

W x = 3

Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ ~~1~~ ~~2~~ 3	~~0~~ ~~1~~ ~~2~~ 3
y	I		

Shim 2

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ 1	~~0~~ 3
y	I		

WRITE x=3@3

x=1

WRITE x=1@1

WRITE x=2@2

WRI

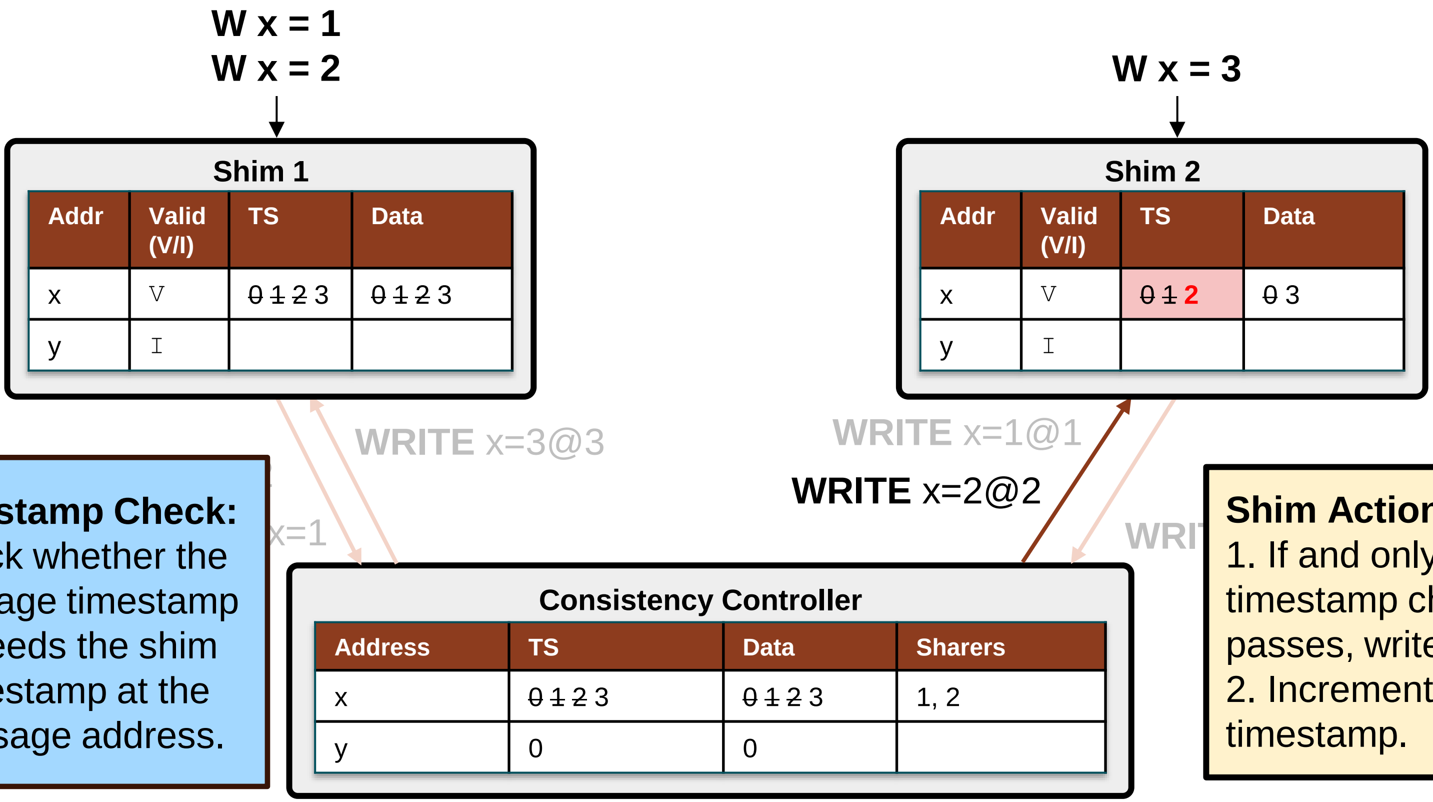
Consistency Controller

Address	TS	Data	Sharers
x	~~0~~ ~~1~~ ~~2~~ 3	~~0~~ ~~1~~ ~~2~~ 3	1, 2
y	0	0	

Timestamp Check: Check whether the message timestamp exceeds the shim timestamp at the message address.

Shim Action:
1. If and only if the timestamp check passes, write the data.
2. Increment the shim timestamp.

Ordered MemGlue By Example: Timestamps



Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ ~~1~~ ~~2~~ 3	~~0~~ ~~1~~ ~~2~~ 3
y	I		

Shim 2

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ ~~1~~ 2	~~0~~ 3
y	I		

Consistency Controller

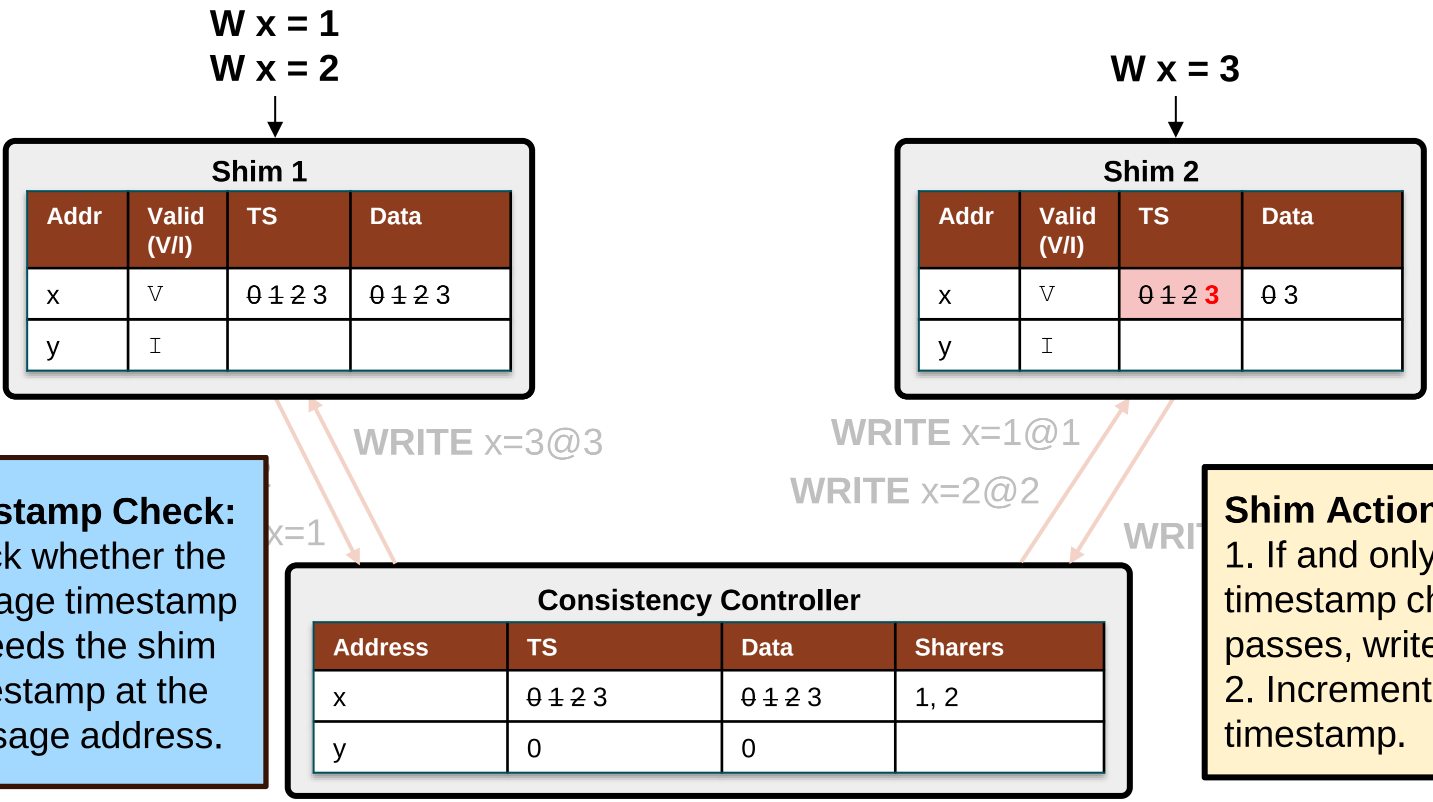
Address	TS	Data	Sharers
x	~~0~~ ~~1~~ ~~2~~ 3	~~0~~ ~~1~~ ~~2~~ 3	1, 2
y	0	0	

Timestamp Check: Check whether the message timestamp exceeds the shim timestamp at the message address.

Shim Action:
1. If and only if the timestamp check passes, write the data.
2. Increment the shim timestamp.

Stanford University

Ordered MemGlue By Example: Timestamps



W x = 1
W x = 2

W x = 3

Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ ~~1~~ ~~2~~ 3	~~0~~ ~~1~~ ~~2~~ 3
y	I		

Shim 2

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ ~~1~~ ~~2~~ 3	~~0~~ 3
y	I		

Consistency Controller

Address	TS	Data	Sharers
x	~~0~~ ~~1~~ ~~2~~ 3	~~0~~ ~~1~~ ~~2~~ 3	1, 2
y	0	0	

Timestamp Check: Check whether the message timestamp exceeds the shim timestamp at the message address.

Shim Action:
1. If and only if the timestamp check passes, write the data.
2. Increment the shim timestamp.

Roadmap

MemGlue Design Principles

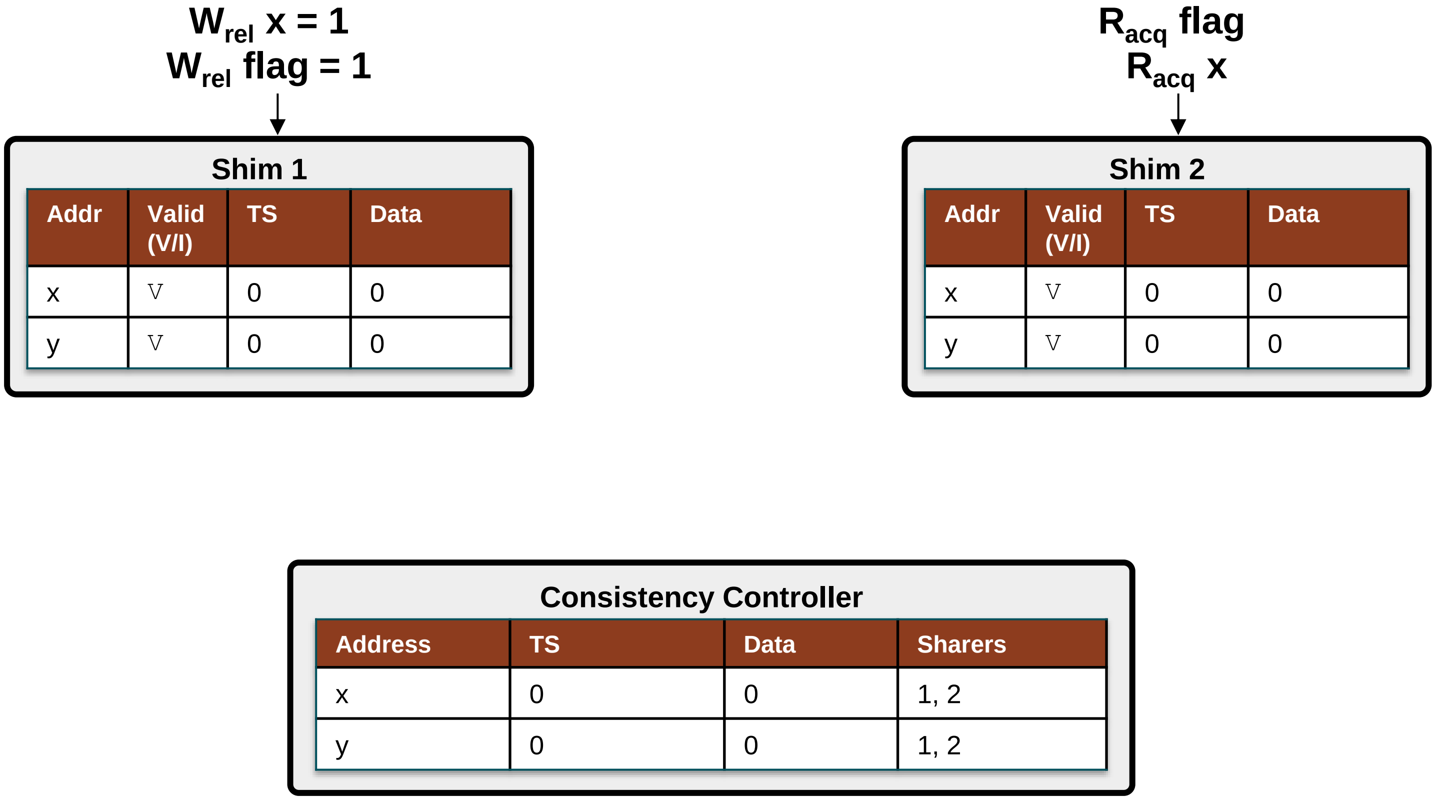
Ordered MemGlue (Ordered Interconnect Network)

Unordered MemGlue (Unordered Interconnect Network)

Experimental Evaluation & Results

- Bounded proof of correctness (litmus testing)
- Complete proof of correctness (manual)

Unordered MemGlue Network Reordering



W_{rel} x = 1
W_{rel} flag = 1

Shim 1

Addr	Valid (V/I)	TS	Data
x	V	0	0
y	V	0	0

R_{acq} flag
R_{acq} x

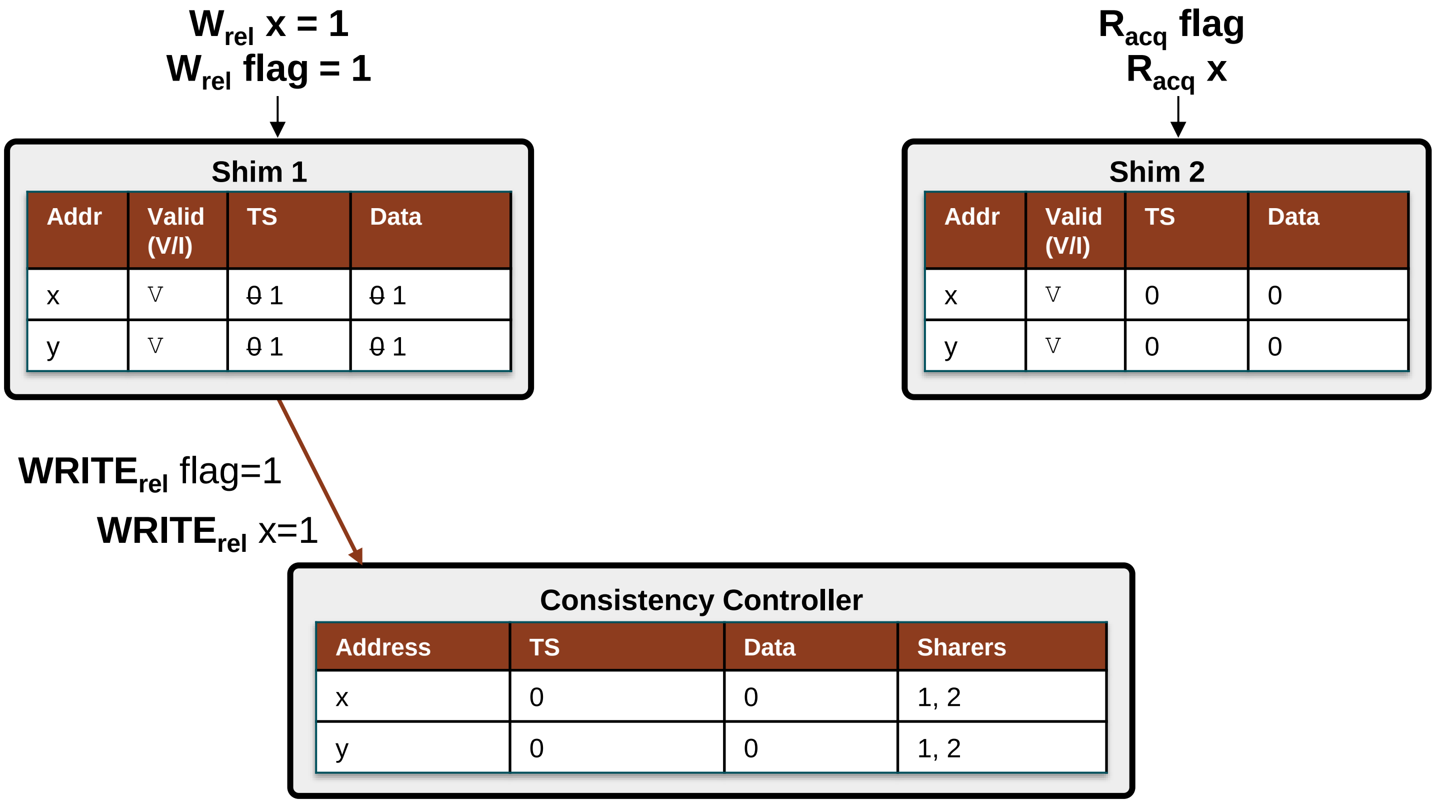
Shim 2

Addr	Valid (V/I)	TS	Data
x	V	0	0
y	V	0	0

Consistency Controller

Address	TS	Data	Sharers
x	0	0	1, 2
y	0	0	1, 2

Unordered MemGlue Network Reordering



W_{rel} x = 1
W_{rel} flag = 1

Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ 1	~~0~~ 1
y	V	~~0~~ 1	~~0~~ 1

R_{acq} flag
R_{acq} x

Shim 2

Addr	Valid (V/I)	TS	Data
x	V	0	0
y	V	0	0

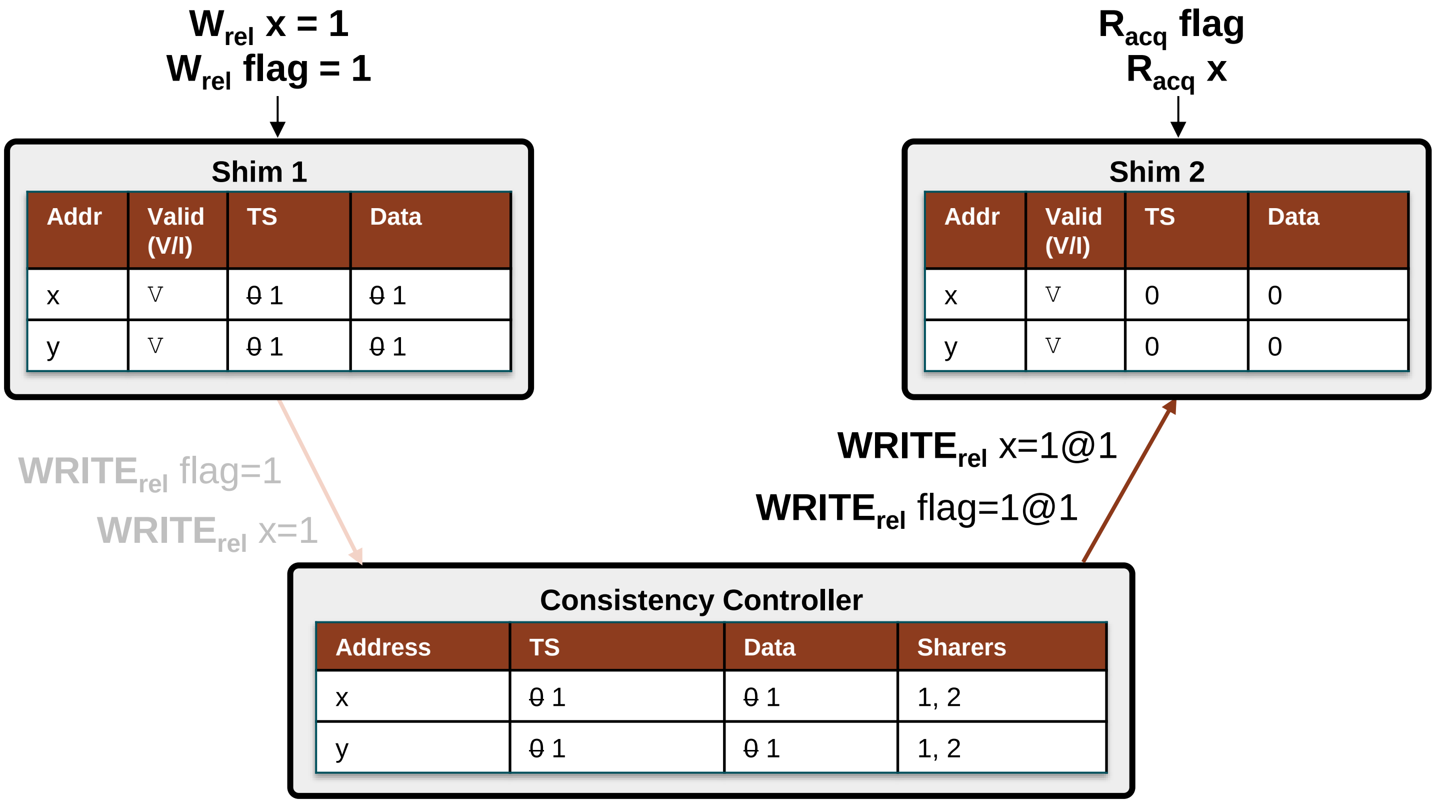
WRITE$_{rel}$ flag=1

WRITE$_{rel}$ x=1

Consistency Controller

Address	TS	Data	Sharers
x	0	0	1, 2
y	0	0	1, 2

Unordered MemGlue Network Reordering



W_{rel} x = 1
W_{rel} flag = 1

R_{acq} flag
R_{acq} x

Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ 1	~~0~~ 1
y	V	~~0~~ 1	~~0~~ 1

Shim 2

Addr	Valid (V/I)	TS	Data
x	V	0	0
y	V	0	0

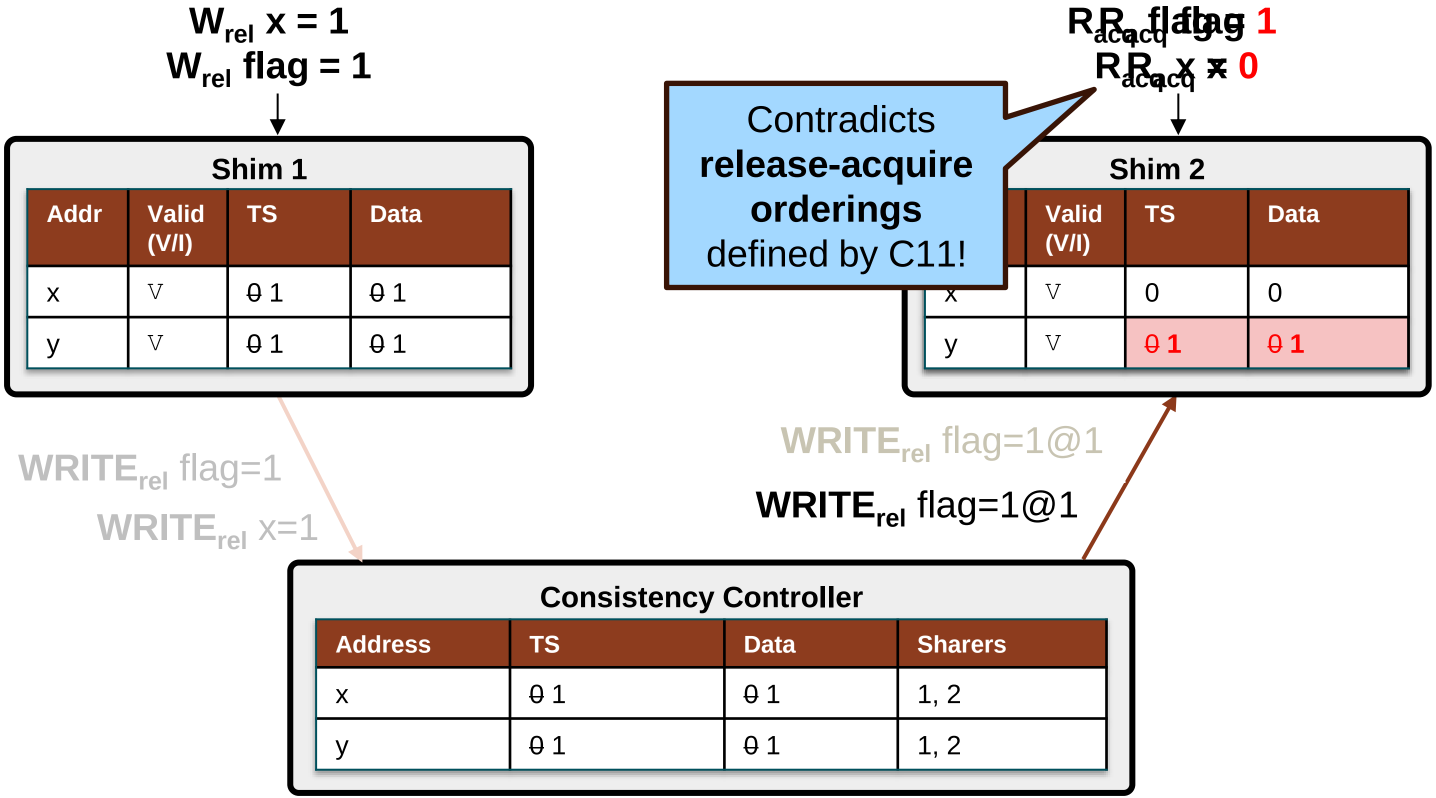
WRITE$_{rel}$ flag=1
WRITE$_{rel}$ x=1

WRITE$_{rel}$ x=1@1
WRITE$_{rel}$ flag=1@1

Consistency Controller

Address	TS	Data	Sharers
x	~~0~~ 1	~~0~~ 1	1, 2
y	~~0~~ 1	~~0~~ 1	1, 2

Unordered MemGlue Network Reordering



W_{rel} x = 1
W_{rel} flag = 1

R_{acq} flag 1
R_{acq} x 0

Contradicts **release-acquire orderings** defined by C11!

Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ 1	~~0~~ 1
y	V	~~0~~ 1	~~0~~ 1

Shim 2

	Valid (V/I)	TS	Data
x	V	0	0
y	V	~~0~~ 1	~~0~~ 1

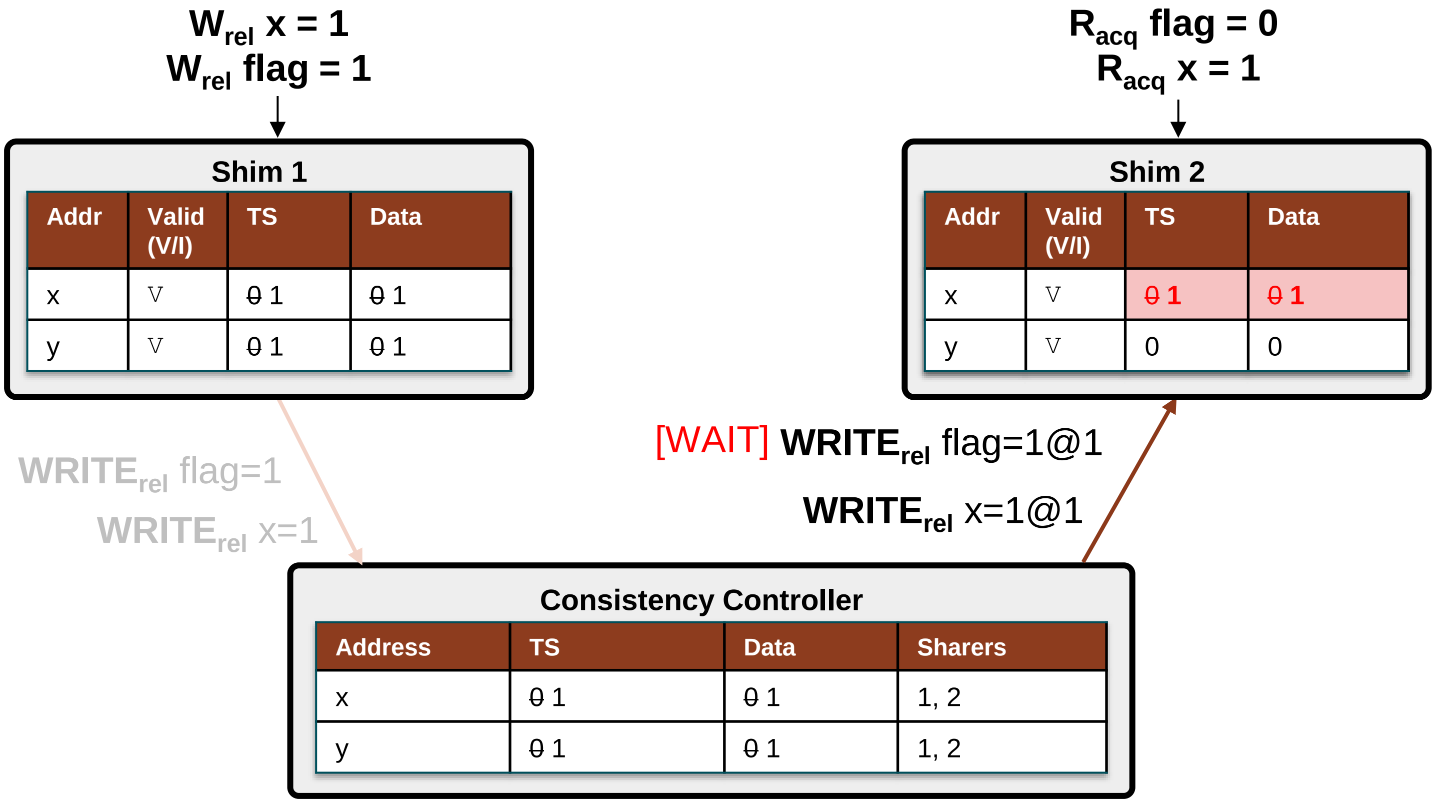
$WRITE_{rel}$ flag=1
$WRITE_{rel}$ x=1

$WRITE_{rel}$ flag=1@1
$WRITE_{rel}$ flag=1@1

Consistency Controller

Address	TS	Data	Sharers
x	~~0~~ 1	~~0~~ 1	1, 2
y	~~0~~ 1	~~0~~ 1	1, 2

Unordered MemGlue Network Reordering



W_{rel} x = 1
W_{rel} flag = 1

R_{acq} flag = 0
R_{acq} x = 1

Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ 1	~~0~~ 1
y	V	~~0~~ 1	~~0~~ 1

Shim 2

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ **1**	~~0~~ **1**
y	V	0	0

$WRITE_{rel}$ flag=1

$WRITE_{rel}$ x=1

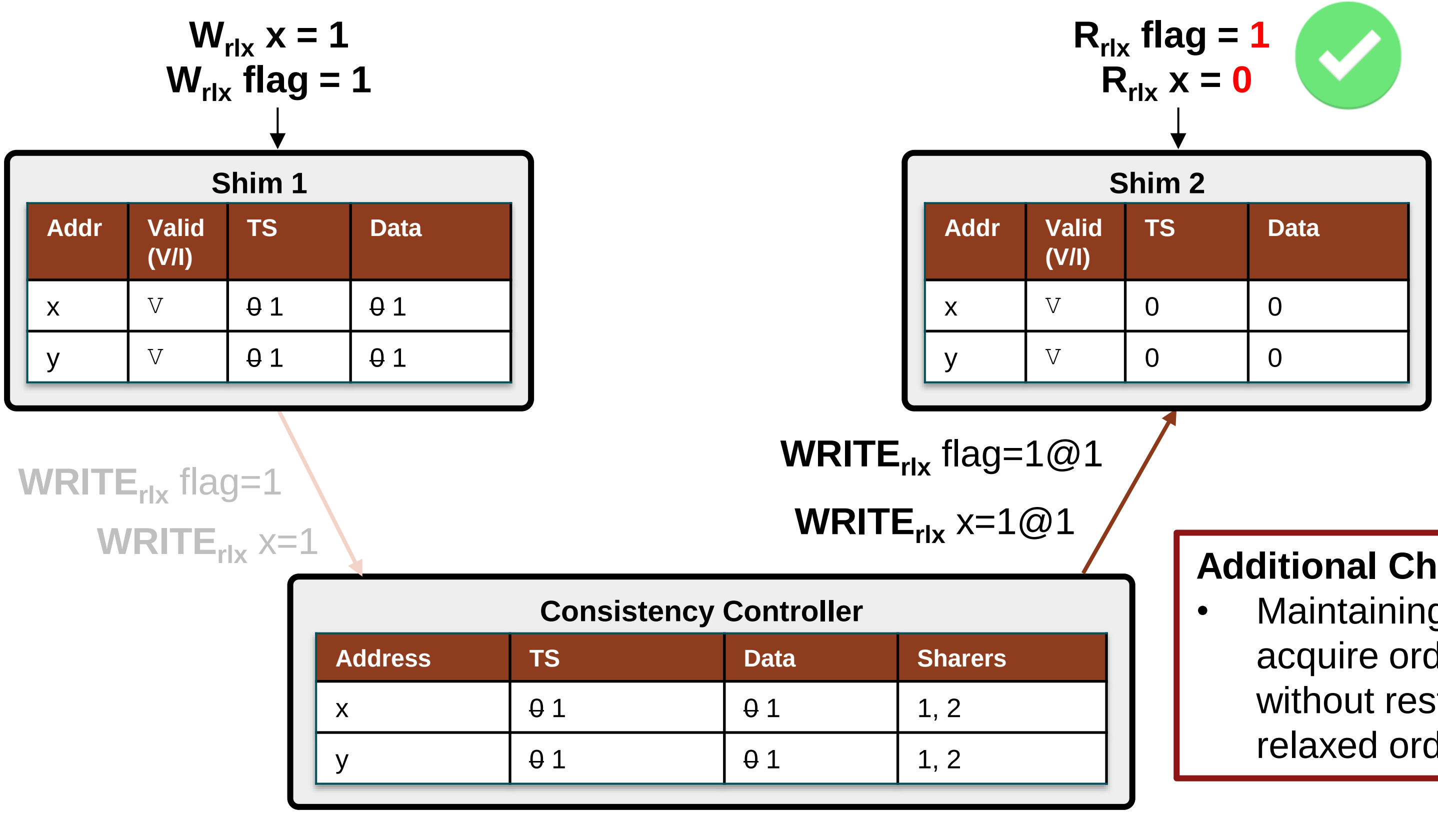
[WAIT] $WRITE_{rel}$ flag=1@1

$WRITE_{rel}$ x=1@1

Consistency Controller

Address	TS	Data	Sharers
x	~~0~~ 1	~~0~~ 1	1, 2
y	~~0~~ 1	~~0~~ 1	1, 2

Unordered MemGlue Network Reordering



W_{rlx} x = 1
W_{rlx} flag = 1

R_{rlx} flag = 1
R_{rlx} x = 0

✓

Shim 1

Addr	Valid (V/I)	TS	Data
x	V	~~0~~ 1	~~0~~ 1
y	V	~~0~~ 1	~~0~~ 1

Shim 2

Addr	Valid (V/I)	TS	Data
x	V	0	0
y	V	0	0

$WRITE_{rlx}$ flag=1

$WRITE_{rlx}$ x=1

$WRITE_{rlx}$ flag=1@1

$WRITE_{rlx}$ x=1@1

Consistency Controller

Address	TS	Data	Sharers
x	~~0~~ 1	~~0~~ 1	1, 2
y	~~0~~ 1	~~0~~ 1	1, 2

Additional Challenge:

- Maintaining release-acquire orderings without restricting relaxed orderings

Unordered MemGlue Metadata

See paper for details

Shim Design

Metadata Cache

Addr	Valid (V/I)	Sync Bit	TS	LWC	RFBufCnt
x					
y					

Counters

icnt	ocnt	fenceCnt

Seen Sets

SeenSet Cache	{}
SeenSet Buffer	{}

Msg Buf

CC Design

Metadata Cache

Address	TS	Data	Sharers	LWS
x				
y				

Msg Buf

Counters

Shim ID	Addr	LWC	fCnt	icnt	ocnt
1	x				
	y				
2	x				
	y				

Seen IDs

Shim ID	Addr	Write ID	Seen ID	Seen Per Shim
1	x			
	y			
2	x			
	y			

Red = metadata from Ordered MemGlue
Tan = metadata added by Unordered MemGlue

Roadmap

MemGlue Design Principles

Ordered MemGlue (Ordered Interconnect Network)

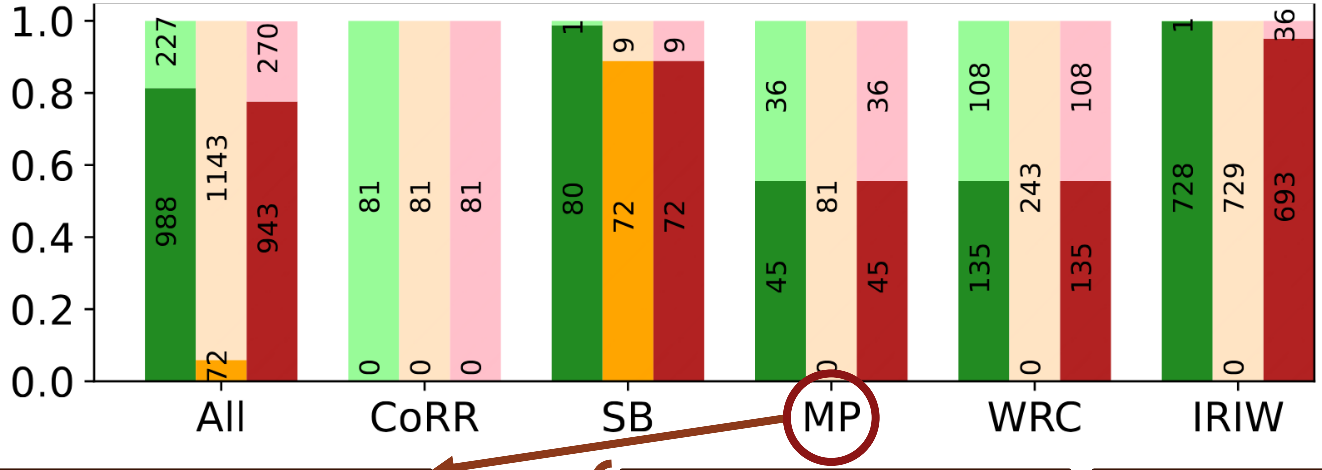
Unordered MemGlue (Unordered Interconnect Network)

Experimental Evaluation & Results

- Bounded proof of correctness (litmus testing)
- Complete proof of correctness (manual)

Results: Litmus Testing for Correctness

We implemented MemGlue in the Murphi model checker and checked its behavior against a suite of 6,738 litmus tests.



Green = C11
Orange = Ordered MemGlue
Red = Unordered MemGlue

Light = disallowed tests
Dark = allowed tests

Thread 1	**Thread 2**
$W_{[\,]}$ **x** = 1	$R_{[\,]}$ **flag** = 1
$W_{[\,]}$ **flag** = 1	$R_{[\,]}$ **x** = 0

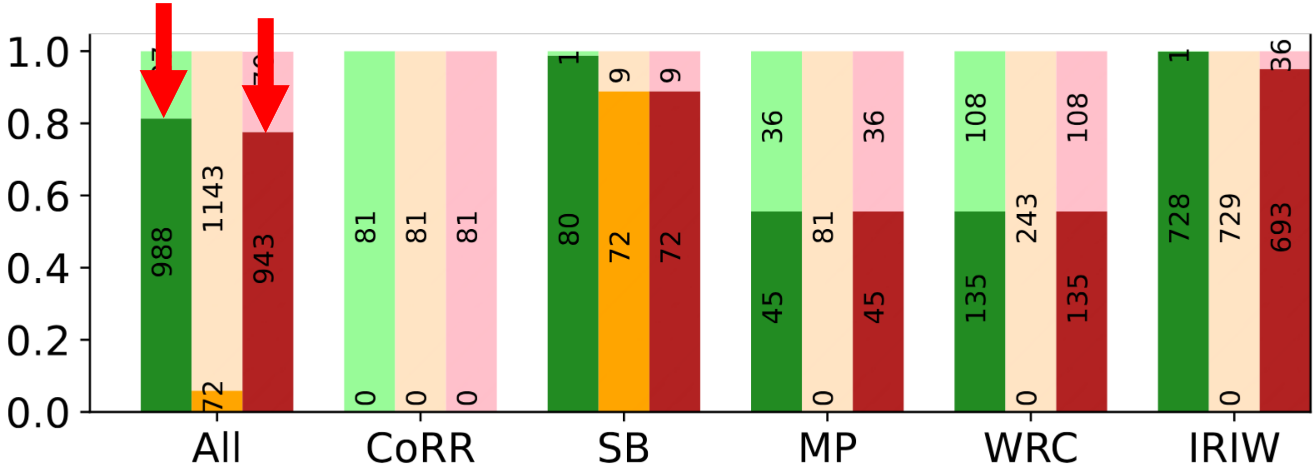
Thread 1	**Thread 2**
W_{rlx} **x** = 1	R_{rlx} **flag** = 1
W_{rlx} **flag** = 1	R_{rlx} **x** = 0

Thread 1	**Thread 2**
W_{rlx} **x** = 1	R_{acq} **flag** = 1
W_{rel} **flag** = 1	R_{sc} **x** = 0

...

Results: Litmus Testing for Correctness

We implemented MemGlue in the Murphi model checker and checked its behavior against a suite of 6,738 litmus tests.



Green = C11
Orange = Ordered MemGlue
Red = Unordered MemGlue

Light = disallowed tests
Dark = allowed tests

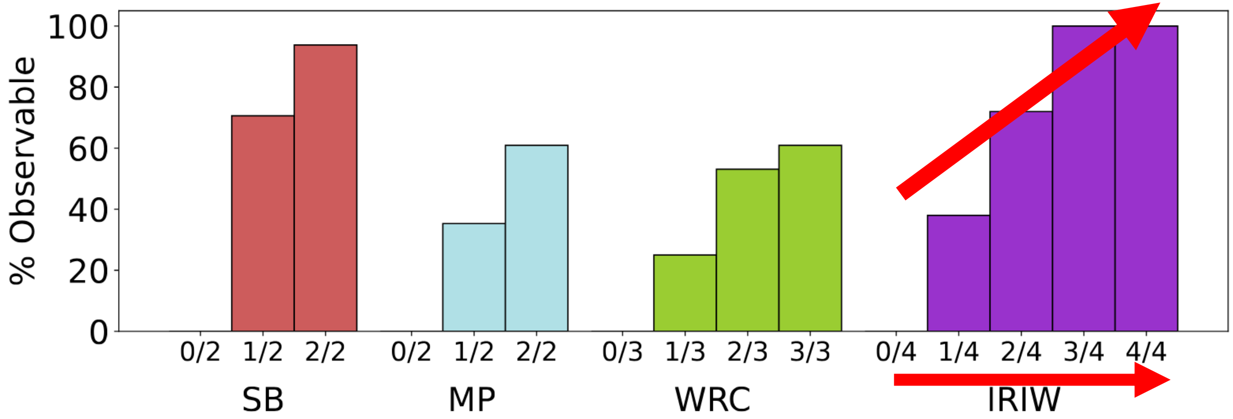
Takeaway 1: Both MemGlue variants uphold C11.

Takeaway 2: Unordered MemGlue effectively leverages relaxed C11 behavior.

Results: Litmus Testing for Politeness

Mapped each test to “strong” clusters and “weak” clusters.

- Strong clusters only emit SC instructions.
- Weak clusters leverage C11 release and relaxed behavior.



Takeaway: MemGlue is polite: it does not overly restrict the system-wide MCM.

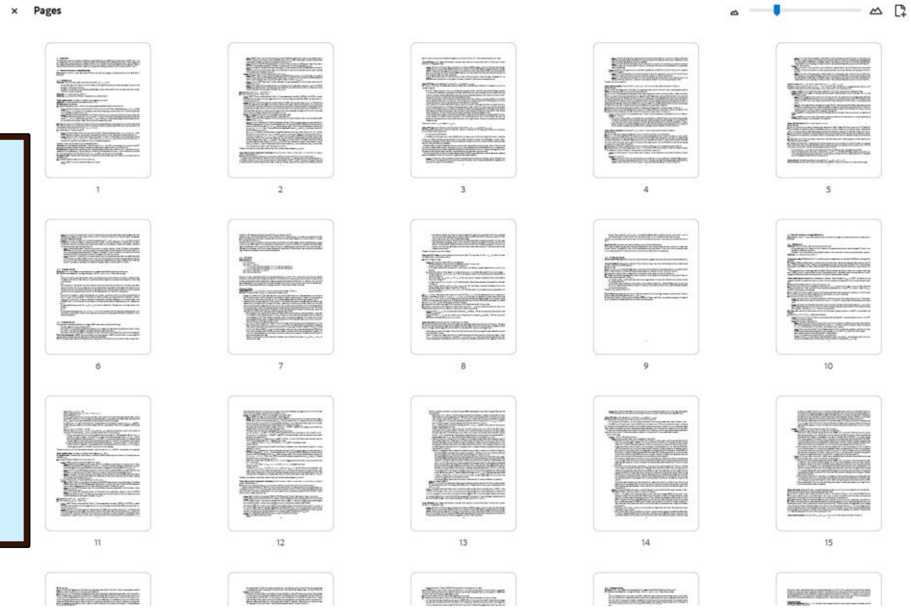
Results: Manual Proof of Correctness

Proof Goal: all program outcomes observable in MemGlue are allowed by the C11 MCM.

C11 MCM is defined **axiomatically.**

$$\begin{aligned}
\texttt{Coherence} &= \texttt{irreflexive}\,(\texttt{hb}\ ;\ \texttt{eco}) \\
\texttt{SC} &= \texttt{acyclic}\,(\texttt{psc}) \\
\texttt{Atomicity} &= \texttt{rmw} \cap (\texttt{fr};\texttt{mo}) = \emptyset \\
\texttt{No-Thin-Air} &= \texttt{acyclic}\,(\texttt{sb} \mid \texttt{rf})
\end{aligned}$$

From [Lahav+, PLDI`17]



Takeaways

See paper:

- Correctly integrating shims into their clusters
- Motivating update-based protocols
- Maintaining C11 under Ordered and Unordered MemGlue

Next steps:

- Performance results via simulation
- Mechanize MemGlue’s proof of correctness
- Define a complete operational model of C11

Takeaways

Conclusions

- MemGlue: MCM-aware cache coherence protocol for heterogeneous systems
  - Modular, verifiable, and polite
  - Targets C11
  - Update-based
- Promising application for update-based protocols

Thank you!

rcleavel@stanford.edu
trippel@stanford.edu

Visit our GitHub repository: