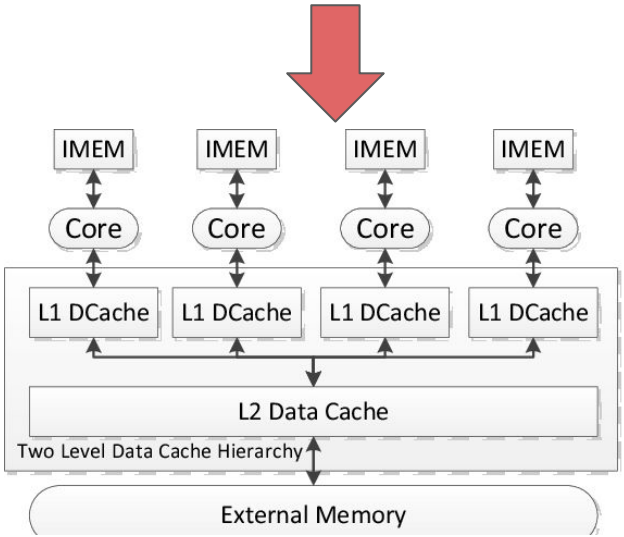
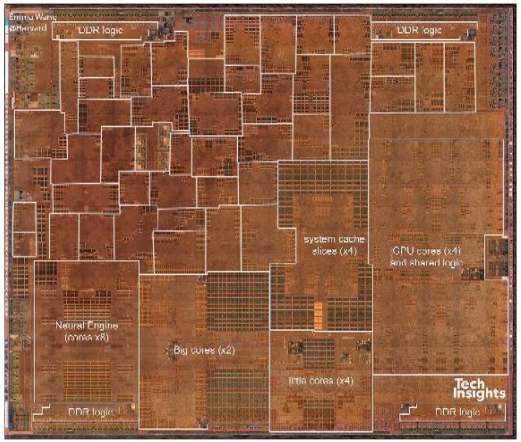
Memory Consistency Model-Aware Cache Coherence for Heterogeneous Shared Memory Systems

Rachel Cleaveland
AHA Meeting
2/7/2024

Cache Coherent Shared Memory for Heterogeneous Hardware



2019 Apple A12 (iPhone XS, XS Max, XR)
40+ accelerators

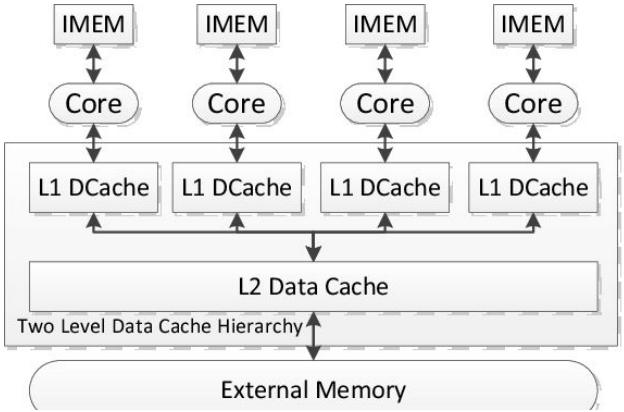


Parallel accesses to the same address must be managed by a cache coherence protocol in order to maintain memory correctness.

Industry designs and standards: NVLink-C2C[1], CXL[2], CAPI[3], CCIX[4]

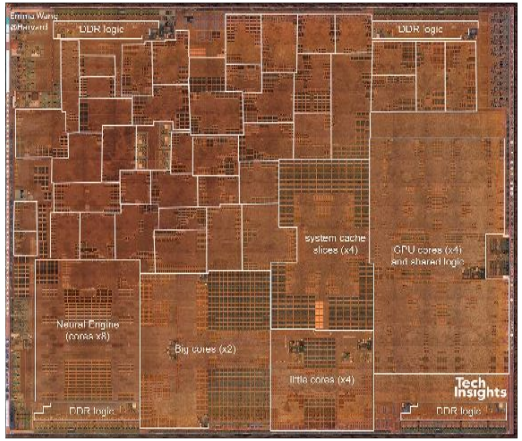
[1] Nvidia Grace Hopper Superchip Architecture. Technical report, Nvidia Corporation, Santa Clara, CA, 2022.
[2] Debendra Das Sharma and Siamak Tavallaei. Compute express link 2.0 white paper. CXL. 2020.
[3] J. Stuecheli, B. Blaner, C. R. Johns, and M. S. Siegel. Capi: A coherent accelerator processor interface. *IBM Journal of Research and Development*, 2015.
[4] Cache coherent interconnect for accelerators (ccix). https://www.ccixconsortium.com/. Accessed: 2023-08-14.

Cache Coherent Shared Memory for Heterogeneous Hardware



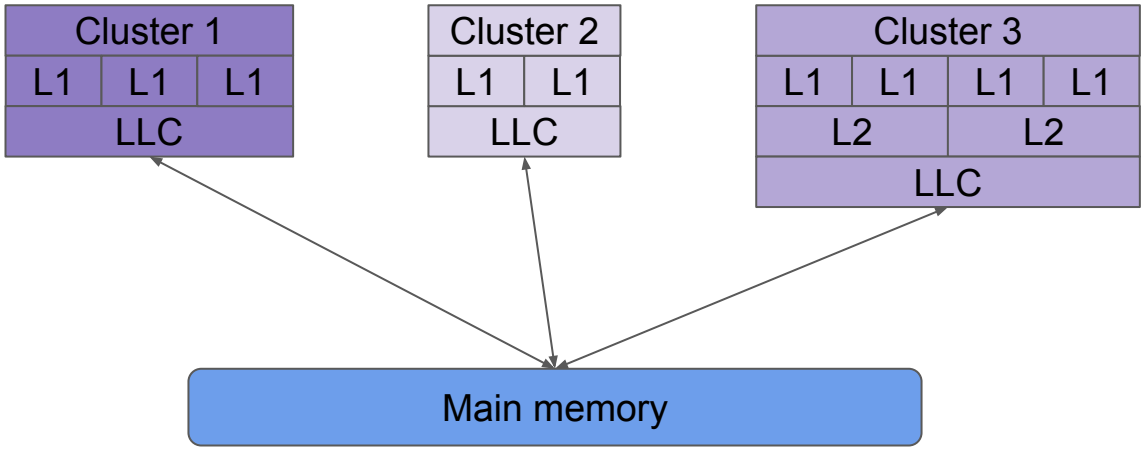
Parallel accesses to the same address must be managed by a cache coherence protocol in order to maintain memory correctness.

2019 Apple A12 (iPhone XS, XS Max, XR)
40+ accelerators



Cache Coherent Shared Memory in Heterogeneous Systems Is Hard

The processing elements which make up modern heterogeneous systems (e.g. Systems-on-Chip (SoCs), multi-chiplet designs, data centers) feature different **memory consistency models (MCMs).**



Overview of Memory Consistency Models

Memory consistency models (MCMs) govern the **ordering and visibility of shared memory accesses** in parallel programs. In doing so, they define what executions are permitted for these programs.

Thread 1	Thread 2	Thread 3	Thread 4
W x = 1	W x = 2	R x = 1 R x = 2	R x = 2 R x = 1

Memory coherence requires a total order on all same-address memory accesses, agreed upon by all cores

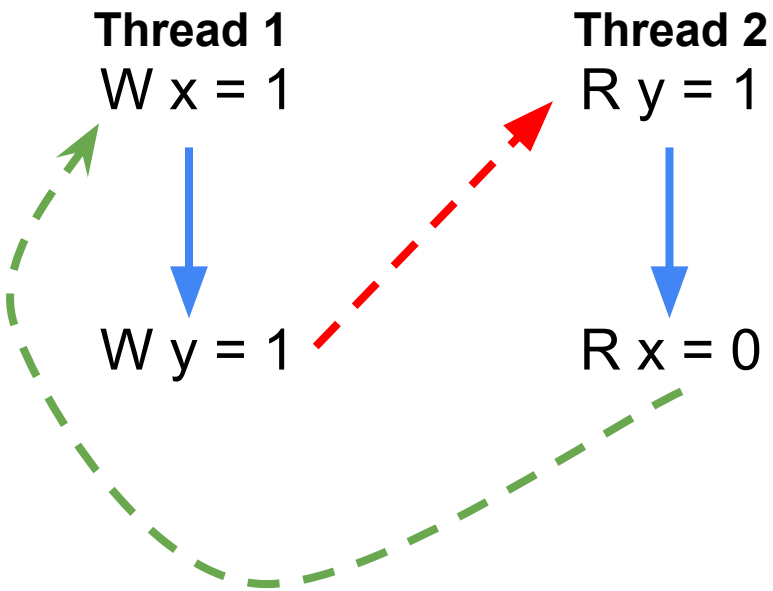
Memory consistency defines ordering requirements on accesses to different addresses

Thread 1	T
1: Wx = 1	3: Ry = 1
2: Wy = 1	4: Rx = 0

Figure 1. Message Passing (MP) litmus test. Memory locations contain zero initially. The outcome is permitted by some MCMs (e.g., ARMv8 [9]), but not others (e.g., x86-TSO [7]).

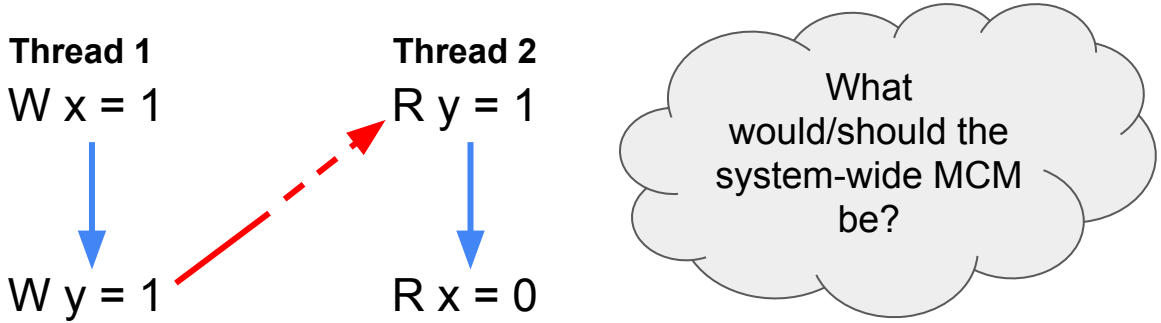
Overview of Memory Consistency Models

MCMs are often specified axiomatically by defining “happens-before” relations between instructions and using these relations to restrict which program executions are permitted.



Cache Coherent Shared Memory in Heterogeneous Systems Is Hard, Continued

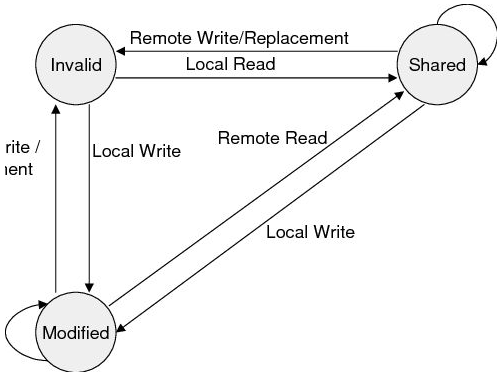
The different system components will make differing assumptions and guarantees about the order in which shared memory accesses are allowed to perform and propagate.

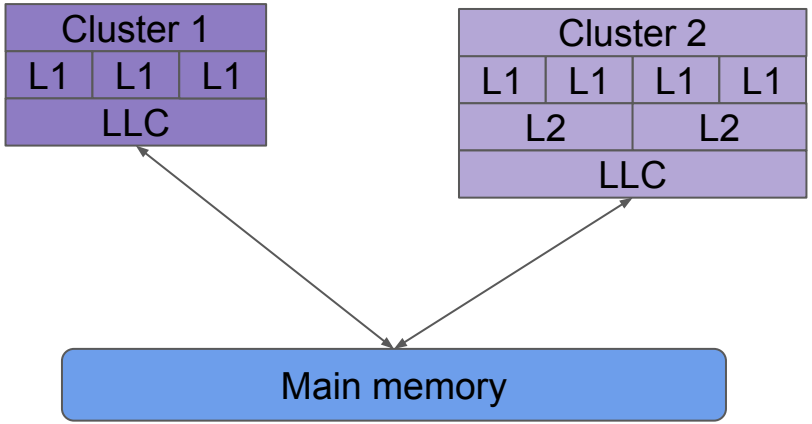


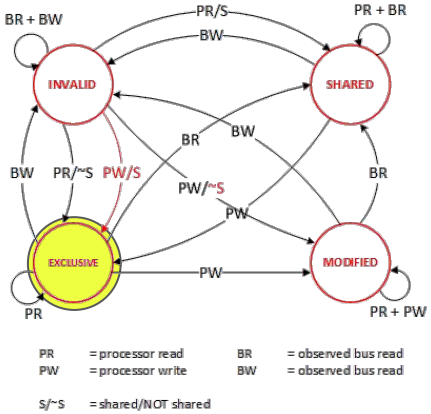
How can all constituent MCMs be honored?

Cache Coherent Shared Memory in Heterogeneous Systems Is Hard, Continued

The different clusters of cores will also have different cache coherence protocols that they implement internally; how can these be merged effectively?



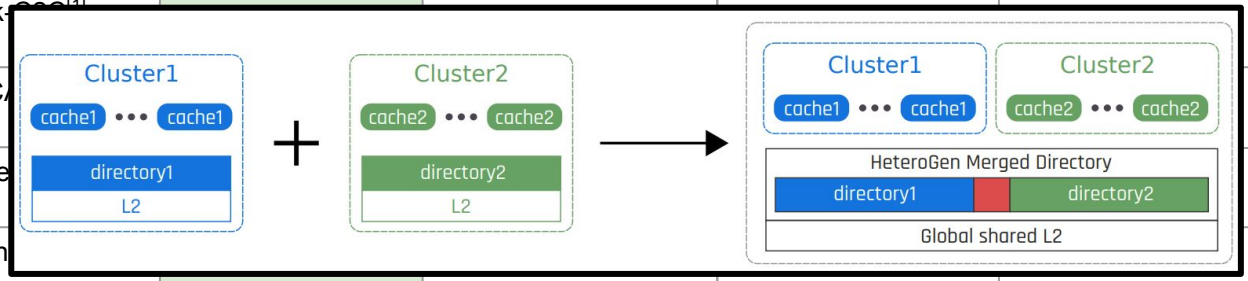




Current Approaches

	Approach	Fine-grained	Memory consistency	Modular	Verifiable
Industrial	Scratchpad memories				
	NVLink-C2C[1]				
	CXL, CA				
	Spande				
	Crossin				
	HeteroGen[4]				
Our Work	MemGlue				

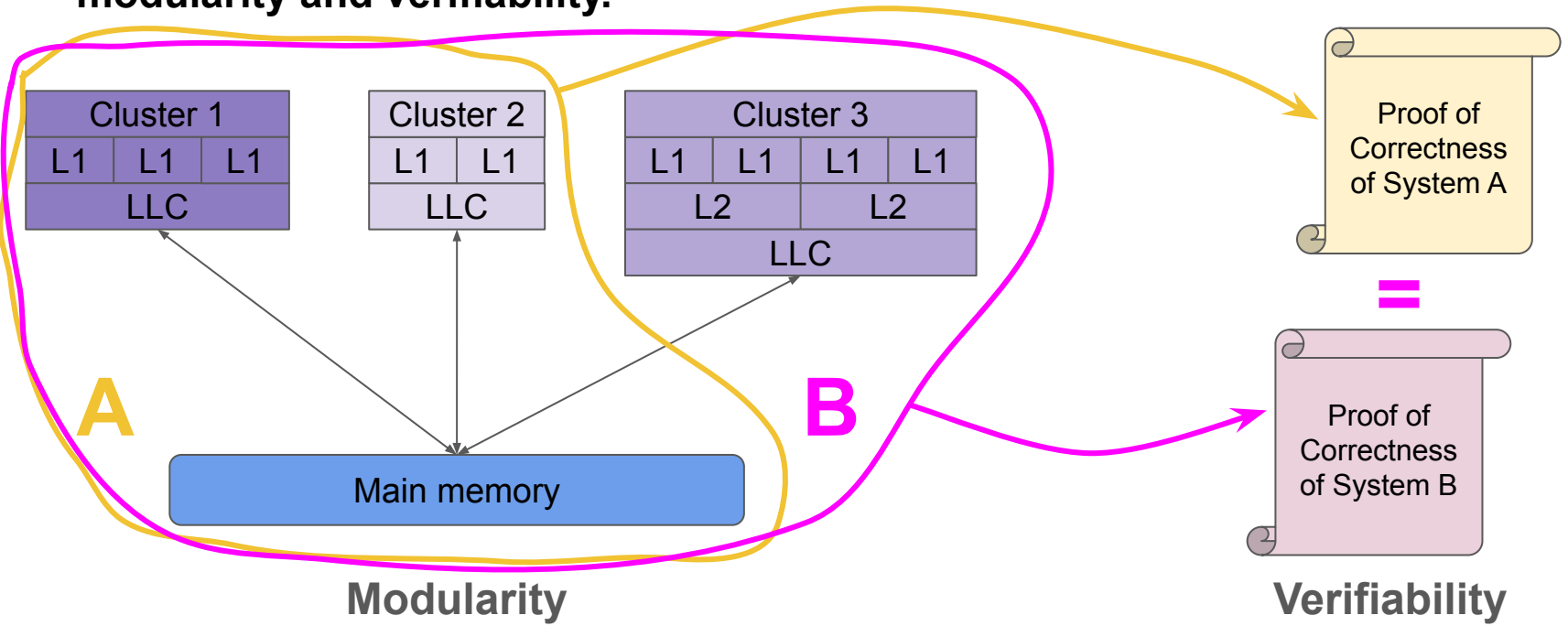
Synthesizes a fresh MCM-aware coherence protocol (and MCM) for each system it unifies



[1] Nvidia Grace Hopper Superchip Architecture. Technical report, Nvidia Corporation, Santa Clara, CA, 2022.
[2] Johnathan Alsop, Matthew Sinclair, and Sarita Adve. Spandex: A flexible interface for efficient heterogeneous coherence. *ISCA* 2018.
[3] Lena E. Olson, Mark D. Hill, and David A. Wood. Crossing guard: Mediating host-accelerator coherence interactions. *ASPLOS*, 2017.
[4] Nicolai Oswald et al. Heterogen: Automatic synthesis of heterogeneous cache coherence protocols. *HPCA*, 2022.

Our Approach: MemGlue

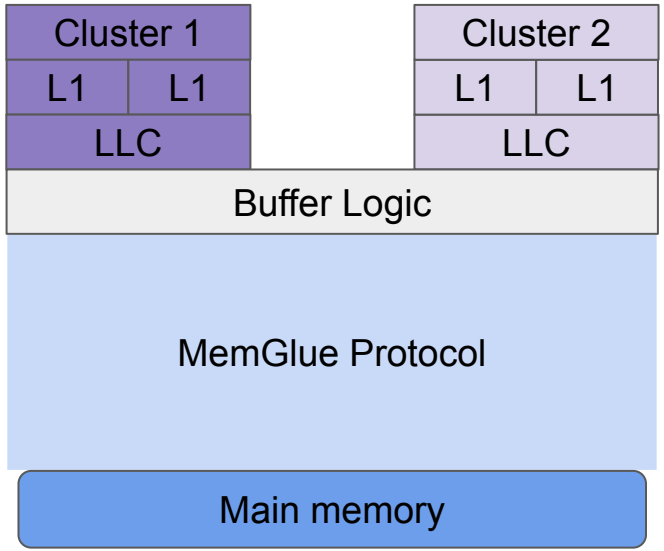
Principle 1: A heterogeneous cache coherence protocol should enable modularity and verifiability.



Our Approach: MemGlue

Principle 1: A heterogeneous cache coherence protocol should enable modularity and verifiability.

Key Insight: These goals can be achieved by using the C11 memory model as the universal memory model for any heterogeneous system unified by MemGlue.

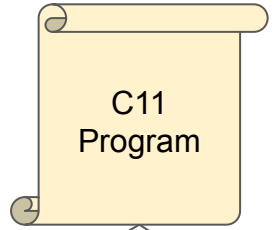


Background: C11 MCM

C11 is the seminal heterogeneous memory consistency model.

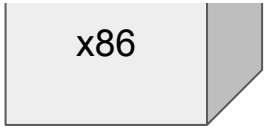
Formally verified compiler mappings (C11 → x86)

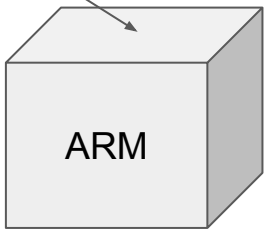
C/C++11 Operation	x86 implementation
Load Relaxed:	MOV (from memory)
Load Consume:	MOV (from memory)
Load Acquire:	MOV (from memory)
Load Seq_Cst:	MOV (from memory)
Store Relaxed:	MOV (into memory)
Store Release:	MOV (into memory)
Store Seq Cst:	(LOCK) XCHG // alternative: MOV (into memory),MFENCE
Consume Fence:	<ignore>
Acquire Fence:	<ignore>
Release Fence:	<ignore>
Acq_Rel Fence:	<ignore>
Seq_Cst Fence:	MFENCE



C11 memory orders
RLX
REL/ACQ
SC

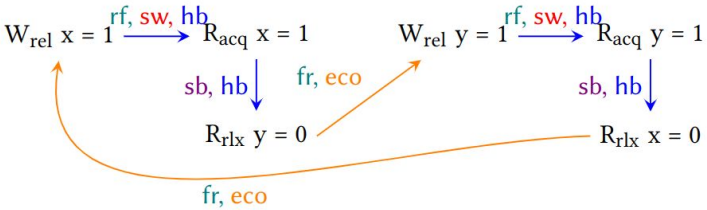
MOV
MFENCE



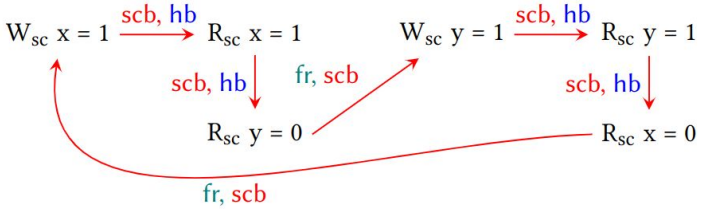


ARM memory operations
LDR
LDA
STR
STL
DMB ISH

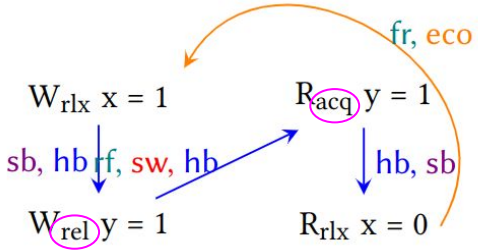
Background: C11 MCM



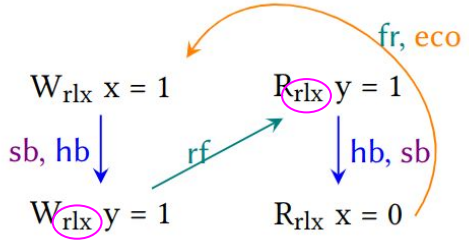
(a) REL-ACQ (Allowed)



(b) SC-SC (Forbidden)



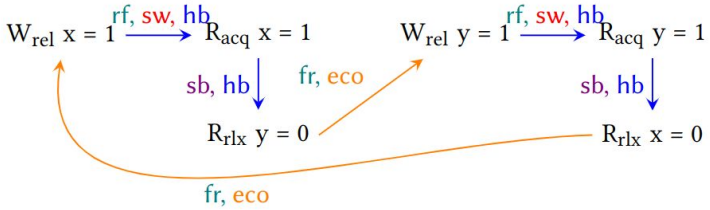
(a) REL-ACQ (Forbidden)



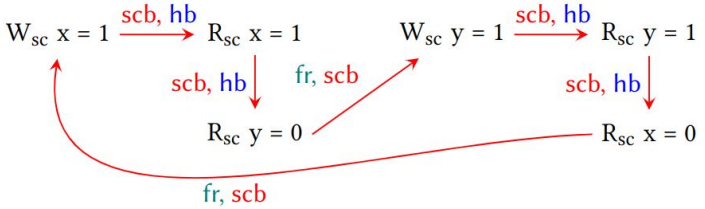
(b) No REL-ACQ (Permitted)

Background: C11 MCM

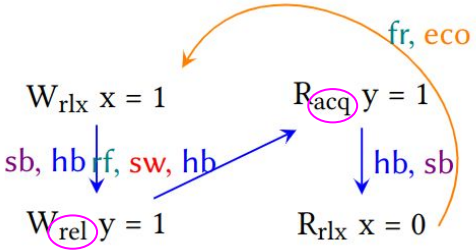
The C11 comprises 4 axioms: Coherence, Atomicity, SC, and No-Thin-Air.



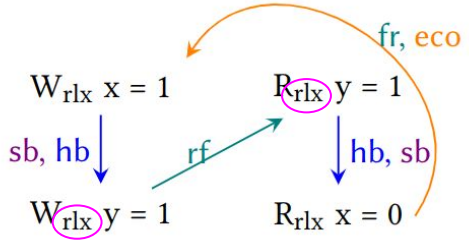
(a) REL-ACQ (Allowed)



(b) SC-SC (Forbidden)



(a) REL-ACQ (Forbidden)



(b) No REL-ACQ (Permitted)

Coherence Axiom: irreflexive (hb;eco)

SC Axiom: acyclic (scb)

Overview: MemGlue and C11

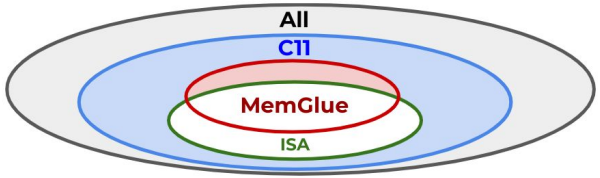
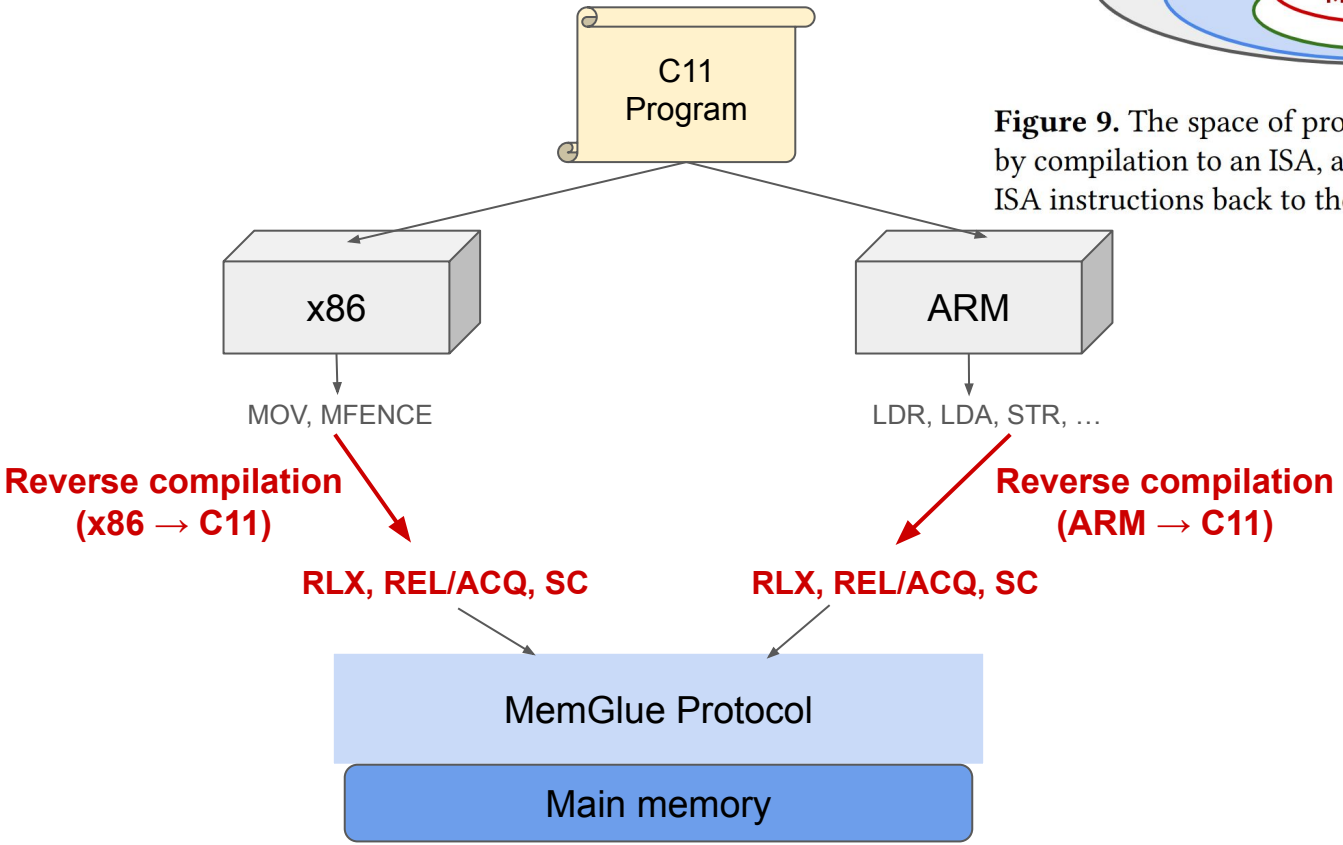
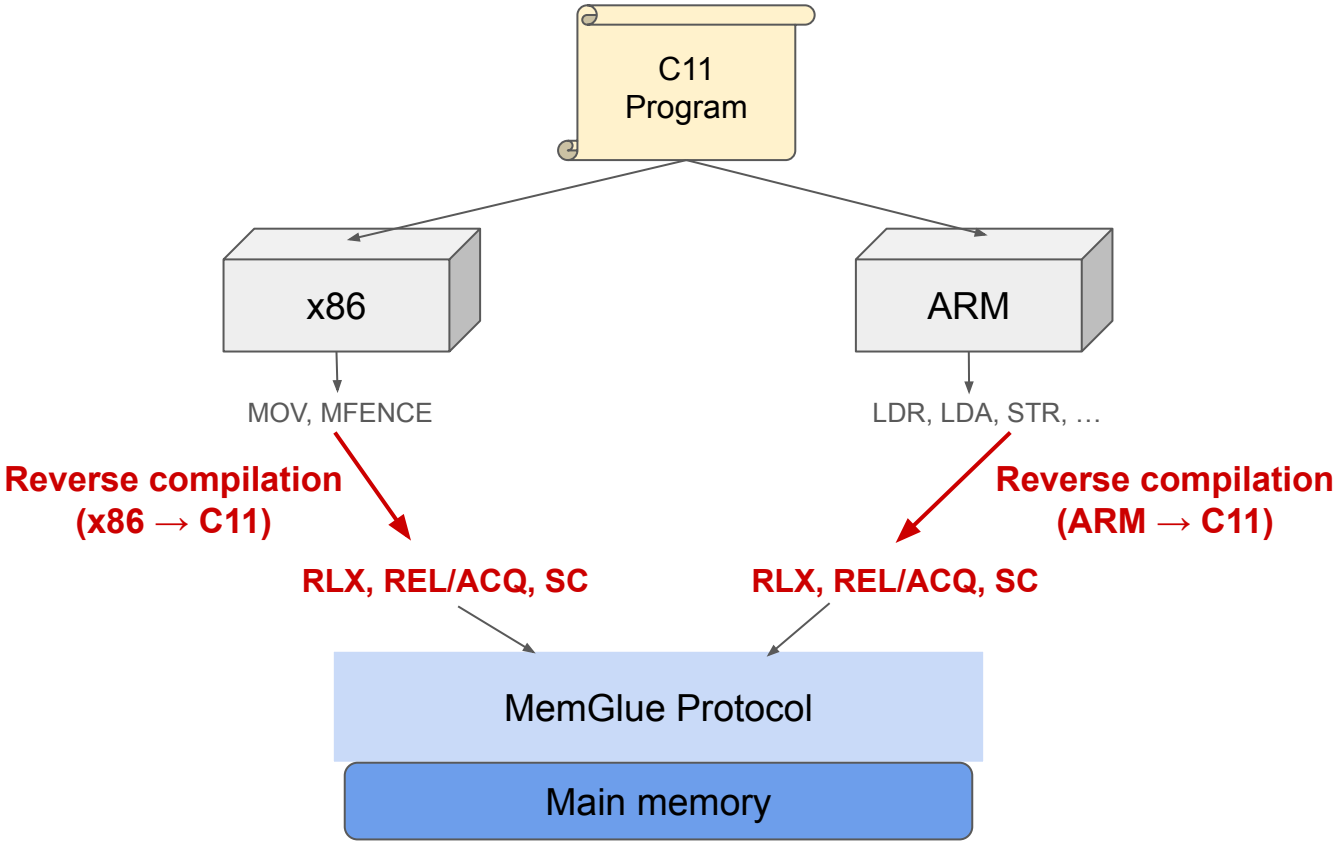


Figure 9. The space of program executions allowed by C11, by compilation to an ISA, and by a reverse-compilation from ISA instructions back to the C11 atomics used by MemGlue.



Overview: MemGlue and C11



Our Approach: MemGlue

Principle 1: A heterogeneous cache coherence protocol should enable modularity and verifiability.

Summary:

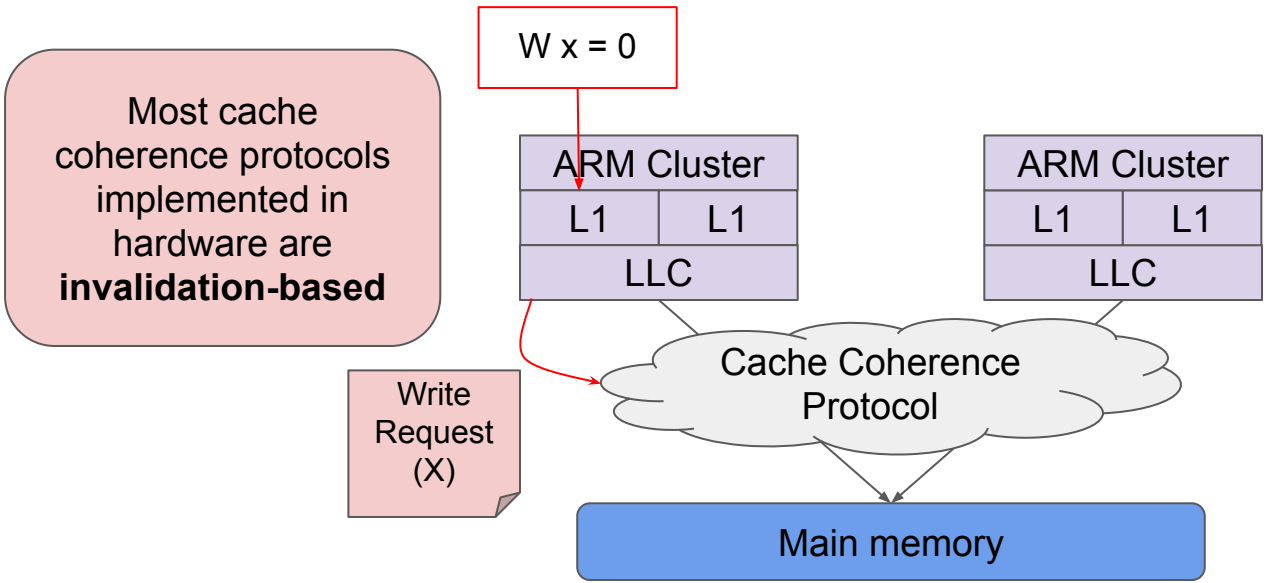
- MemGlue upholds the C11 MCM
- Operates in the language of C11 memory strengths.
  - Reverse-compiles architecture-specific instructions back to C11 instructions.

Our Approach: MemGlue

Principle 2: A heterogeneous cache coherence protocol should not overly restrict the constituent memory consistency models in the system.

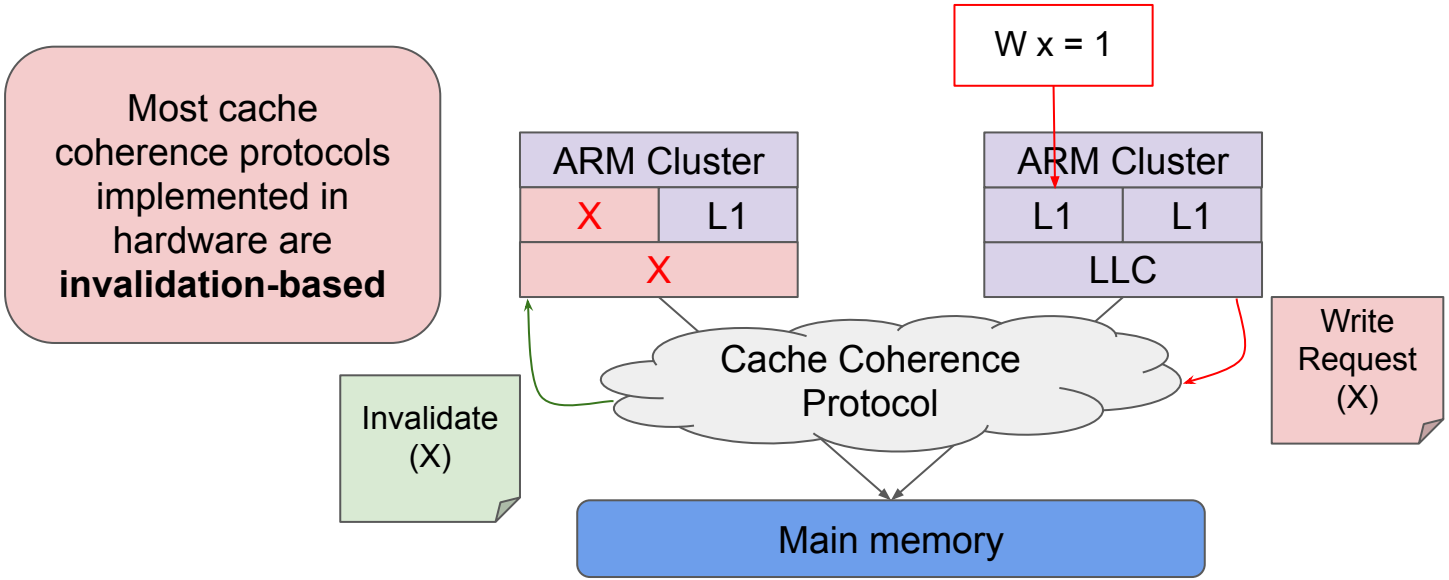
Our Approach: MemGlue

Principle 2: A heterogeneous cache coherence protocol should not overly restrict the constituent memory consistency models in the system.



Our Approach: MemGlue

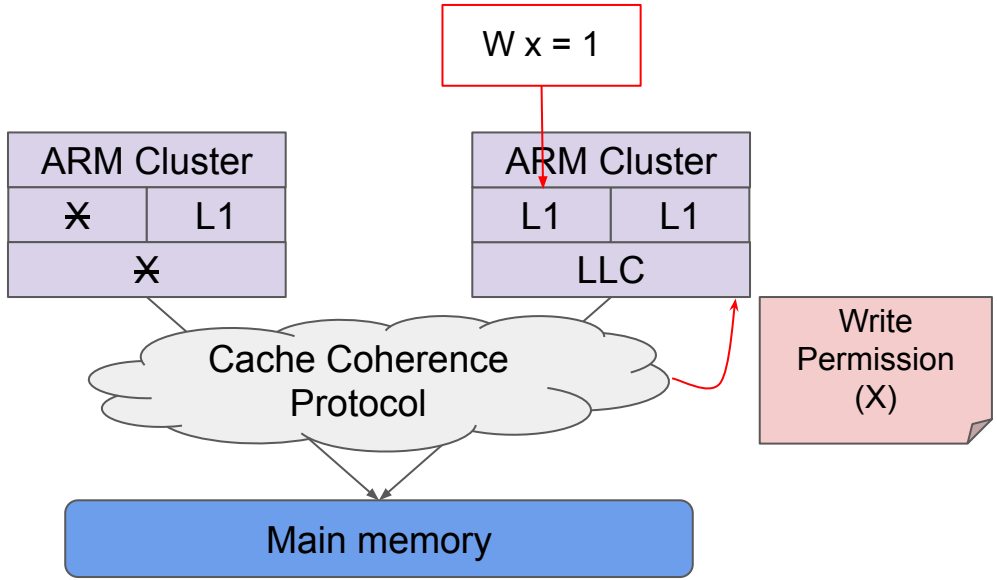
Principle 2: A heterogeneous cache coherence protocol should not overly restrict the constituent memory consistency models in the system.



Our Approach: MemGlue

Principle 2: A heterogeneous cache coherence protocol should not overly restrict the constituent memory consistency models in the system.

Most cache coherence protocols implemented in hardware are **invalidation-based**

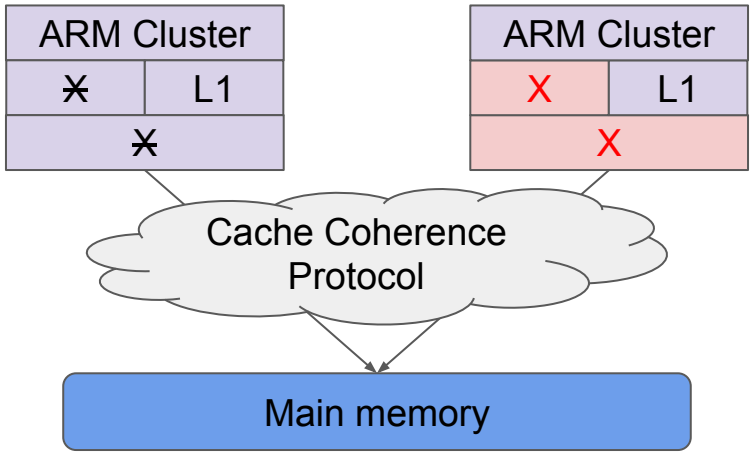


Our Approach: MemGlue

Principle 2: A heterogeneous cache coherence protocol should not overly restrict the constituent memory consistency models in the system.

Most cache coherence protocols implemented in hardware are

Thread 1	Thread 2
1: Wx = 1	3: Ry = 1
2: Wy = 1	4: Rx = 0

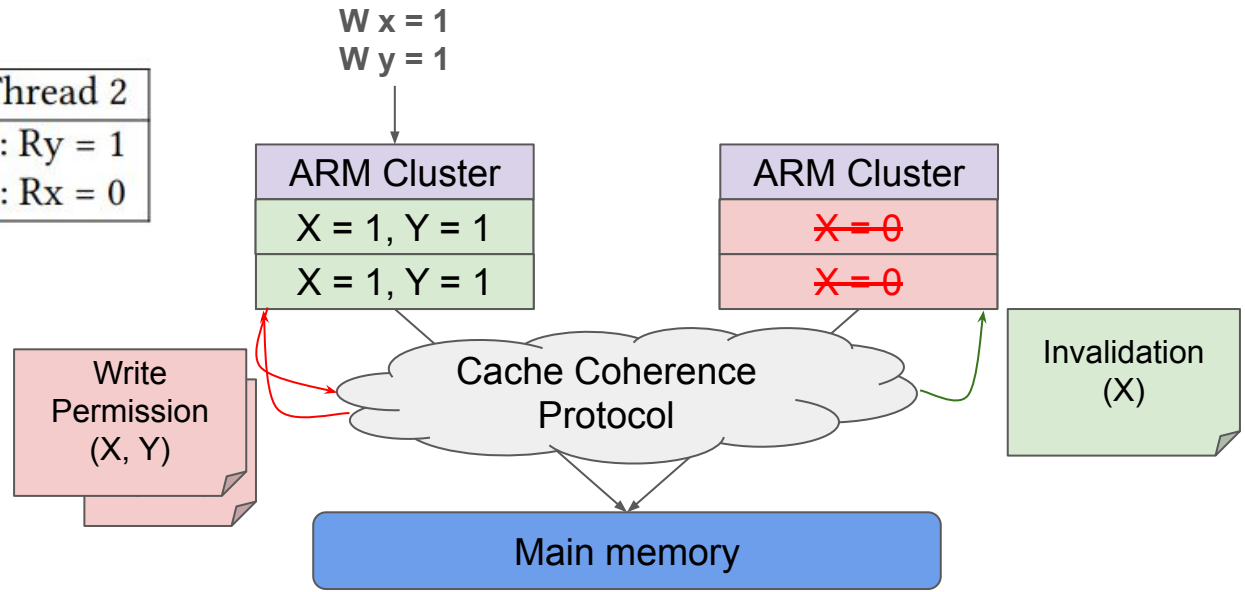


Drawback:
Single writer, multiple reader invariant prevents invalidation-based protocols from leveraging relaxed memory models

Our Approach: MemGlue

Principle 2: A heterogeneous cache coherence protocol should not overly restrict the constituent memory consistency models in the system.

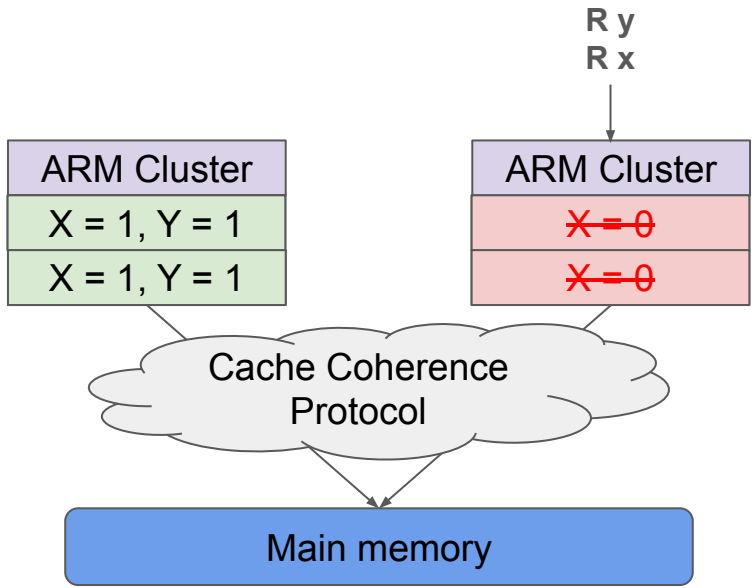
Thread 1	Thread 2
1: Wx = 1	3: Ry = 1
2: Wy = 1	4: Rx = 0



Our Approach: MemGlue

Principle 2: A heterogeneous cache coherence protocol should not overly restrict the constituent memory consistency models in the system.

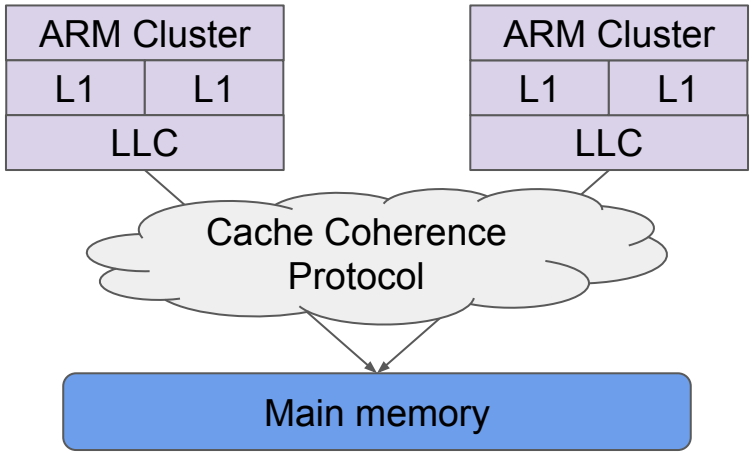
Thread 1	Thread 2
1: Wx = 1	3: Ry = 1
2: Wy = 1	4: Rx = 0



Our Approach: MemGlue

Principle 2: A heterogeneous cache coherence protocol should not overly restrict the constituent memory consistency models in the system.

MemGlue is instead an **update-based protocol**, allowing it to leverage relaxed memory models



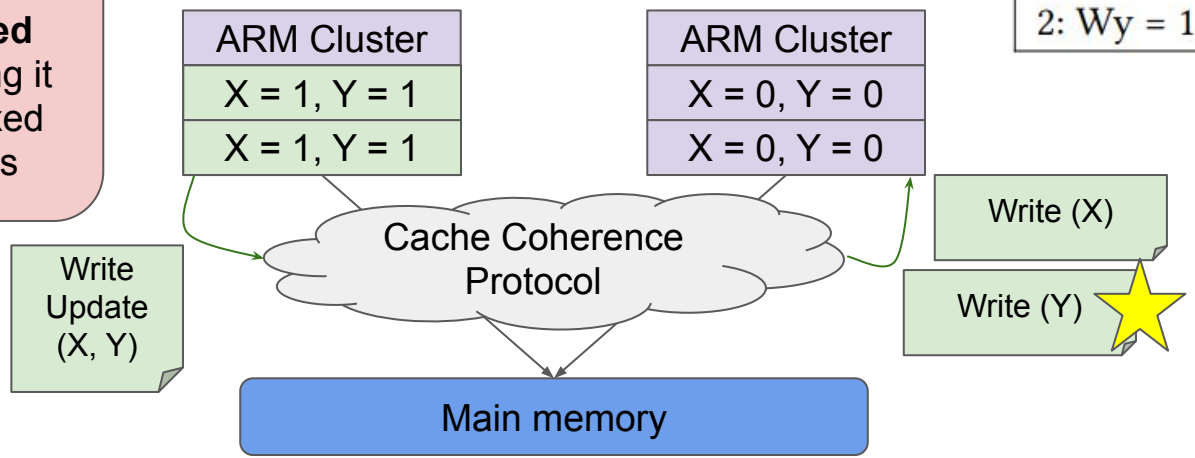
Thread 1	Thread 2
1: Wx = 1	3: Ry = 1
2: Wy = 1	4: Rx = 0

Our Approach: MemGlue

Principle 2: A heterogeneous cache coherence protocol should not overly restrict the constituent memory consistency models in the system.

MemGlue is instead an **update-based protocol**, allowing it to leverage relaxed memory models

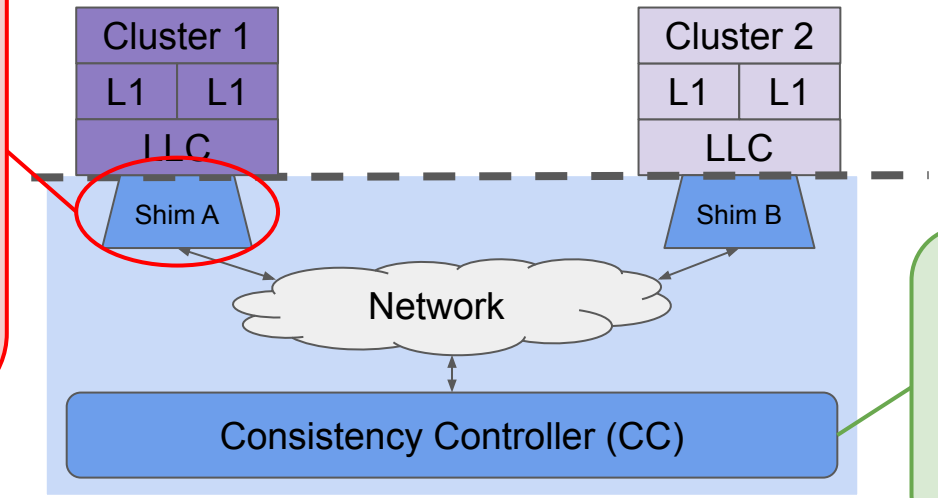
Thread 1	Thread 2
1: Wx = 1	3: Ry = 1
2: Wy = 1	4: Rx = 0



MemGlue Overview: Hardware Structures

Shims are responsible for:

1. Mapping cluster instructions to their MemGlue analog.
2. Deciding which writes may service cluster reads.
3. Forward writes and fences to the CC.



The CC is responsible for establishing a system-wide total order on writes and fences and communicating it to all shims.

(Ordered) MemGlue Overview: Hardware Structures

Shim Design

Cache

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

Consistency Controller Design

Cache

Address	Timestamp	Data	Sharers
x			
y			

MemGlue Overview: Timestamping

Core 1

2) W x = 2

3) W x = 3

Shim	Addr	V/I	TS	Data
	x	V	0	0

Core 2

1) W x = 1

4) W x = 4

Shim	Addr	V/I	TS	Data
	x	V	0	0

Shim Action:
1. Write data
2. Increment timestamp
3. Send write update to CC

CC	Address	TS	Data	Sharers
	x	0	0	1, 2

MemGlue Overview: Timestamping

Core 1

2) W x = 2

3) W x = 3

Shim	Addr	V/I	TS	Data
	x	V	1	2

Core 2

1) W x = 1

4) W x = 4

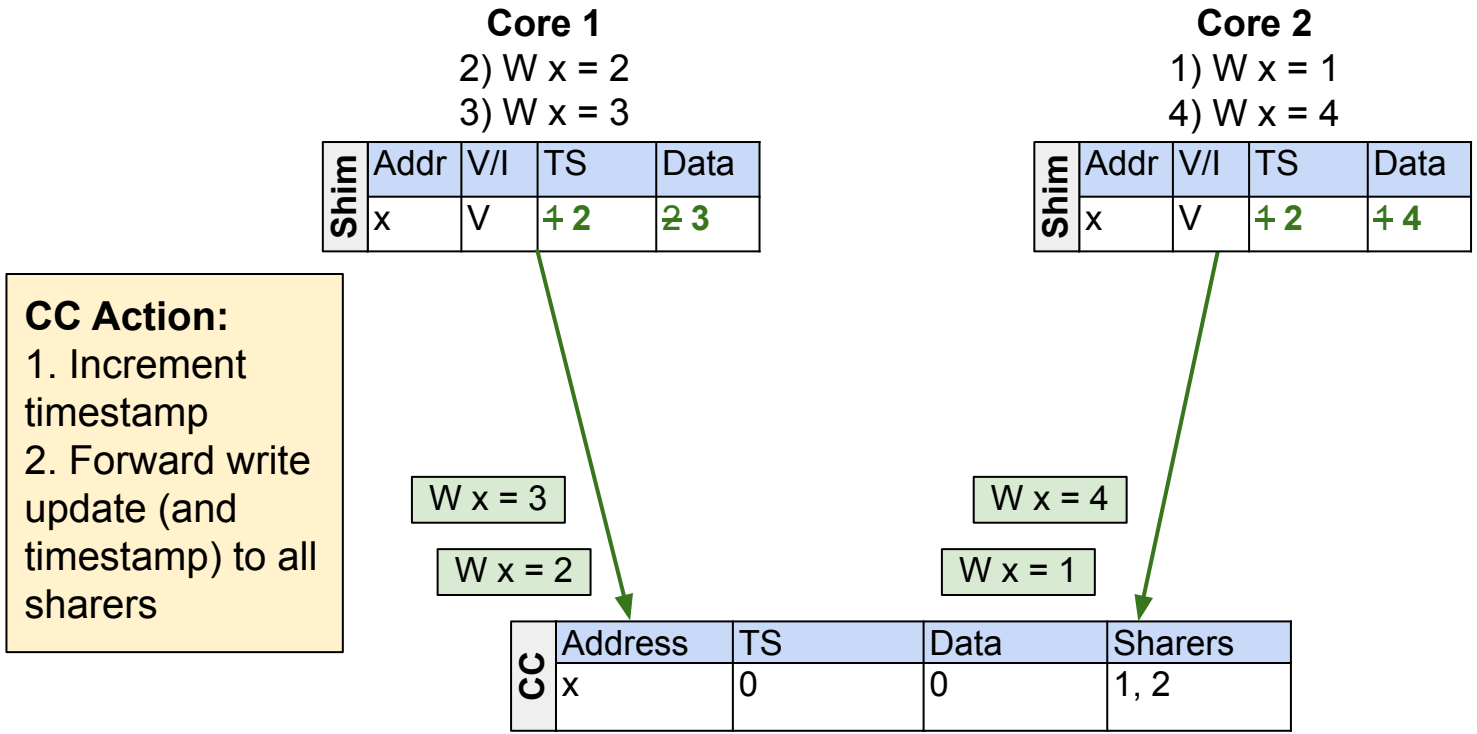
Shim	Addr	V/I	TS	Data
	x	V	0	0

Shim Action:
1. Write data
2. Increment timestamp
3. Send write update to CC

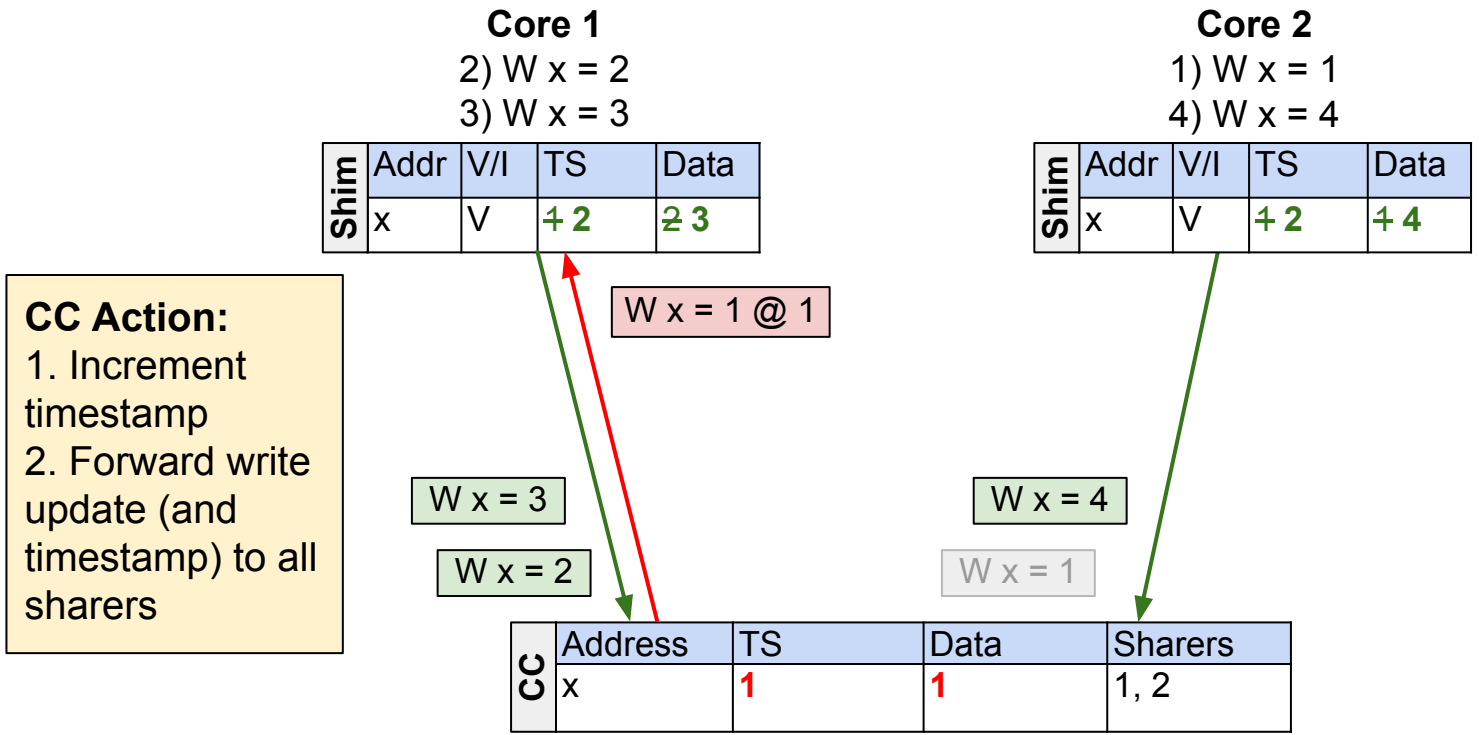
W x = 2

CC	Address	TS	Data	Sharers
	x	0	0	1, 2

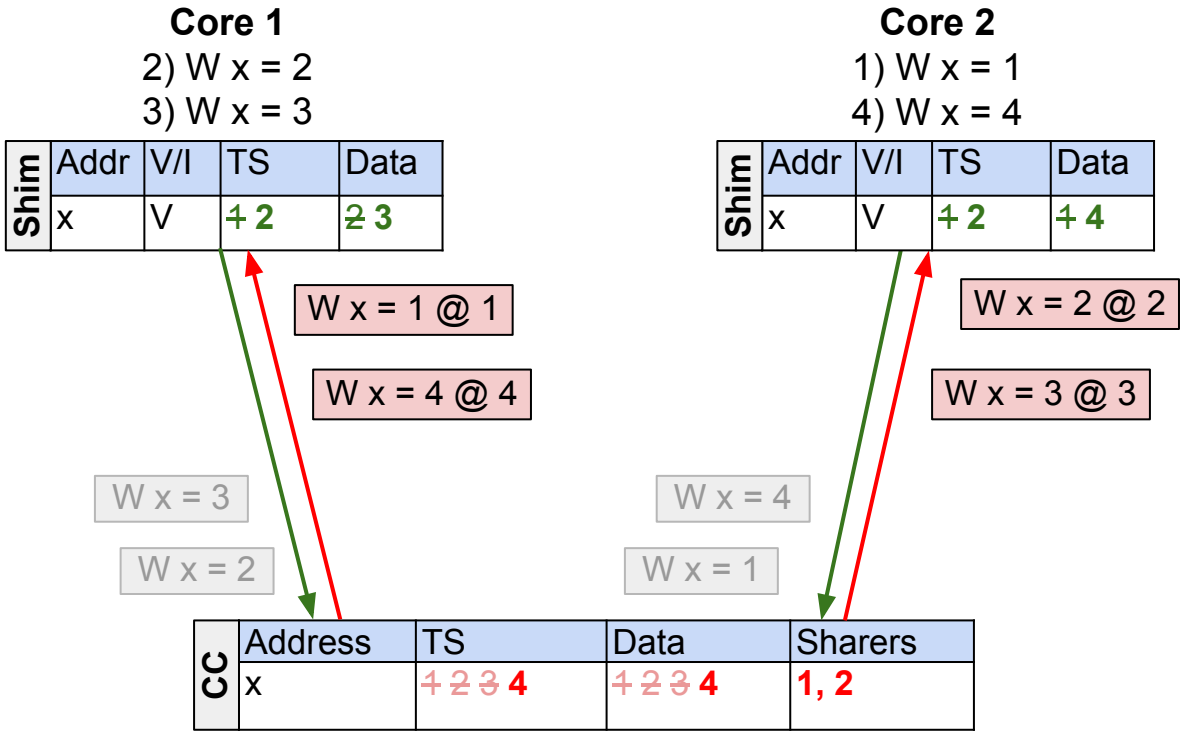
MemGlue Overview: Timestamping



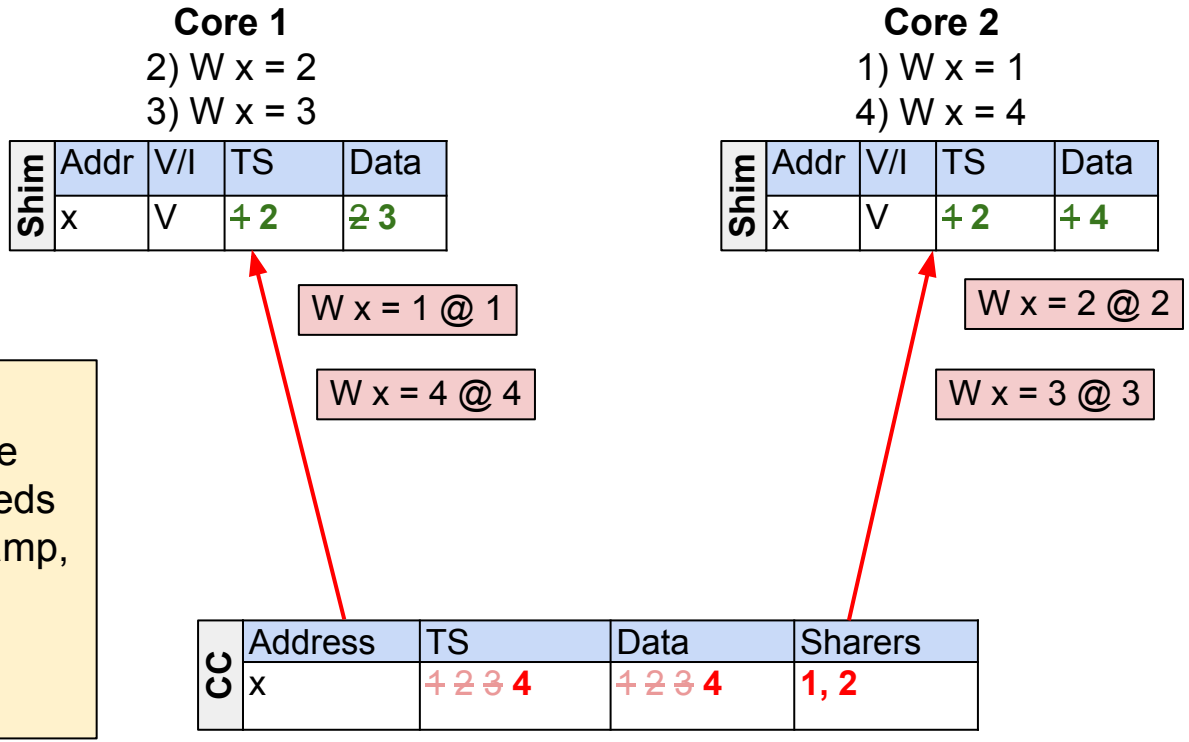
MemGlue Overview: Timestamping



MemGlue Overview: Timestamping

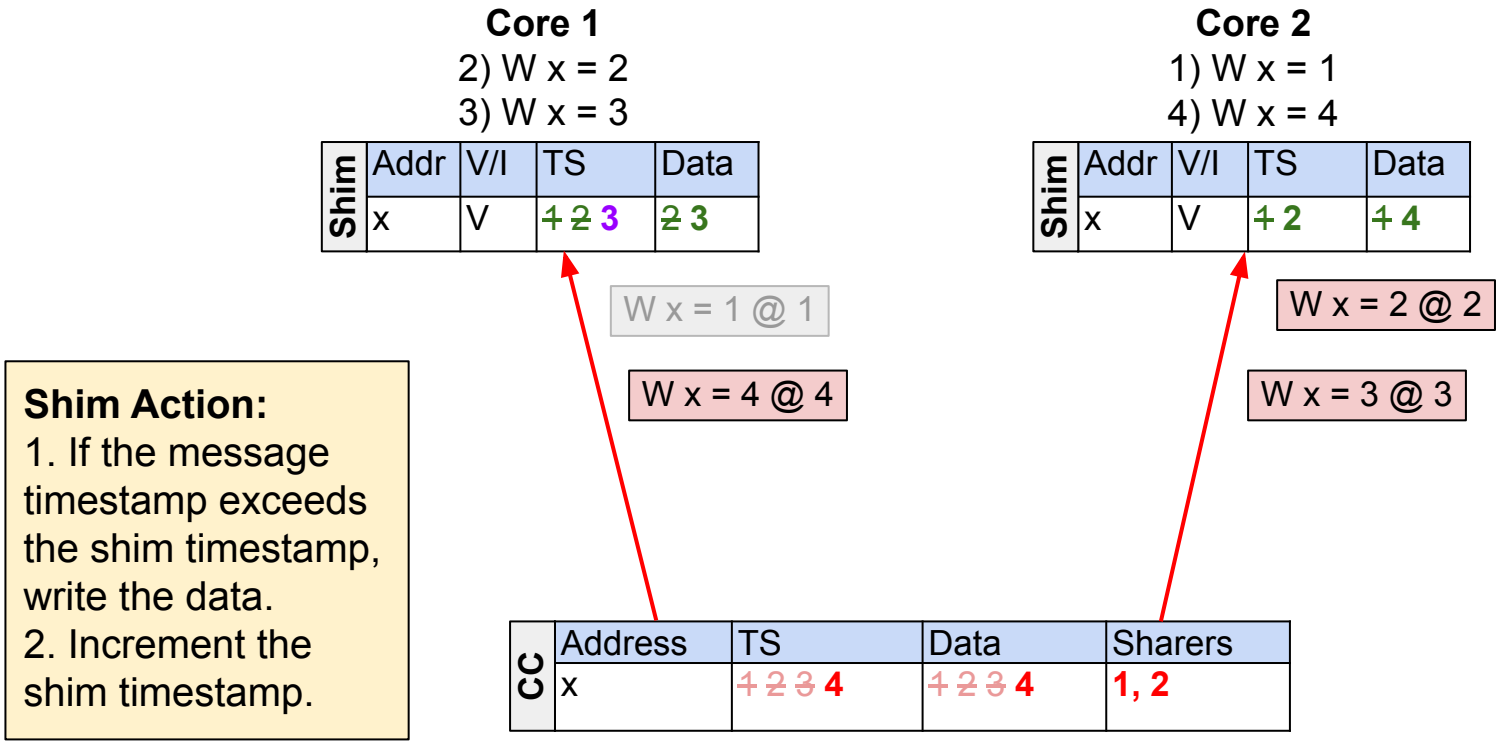


MemGlue Overview: Timestamping



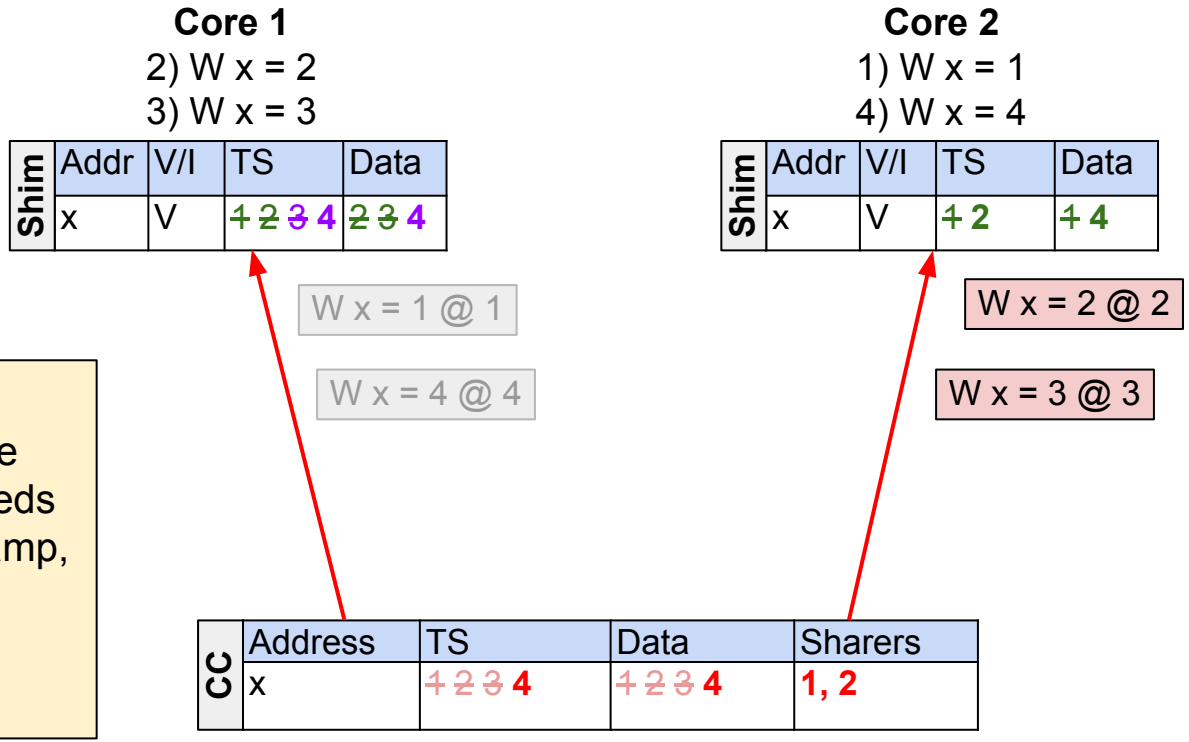
Shim Action:
1. If the message timestamp exceeds the shim timestamp, write the data.
2. Increment the shim timestamp.

MemGlue Overview: Timestamping



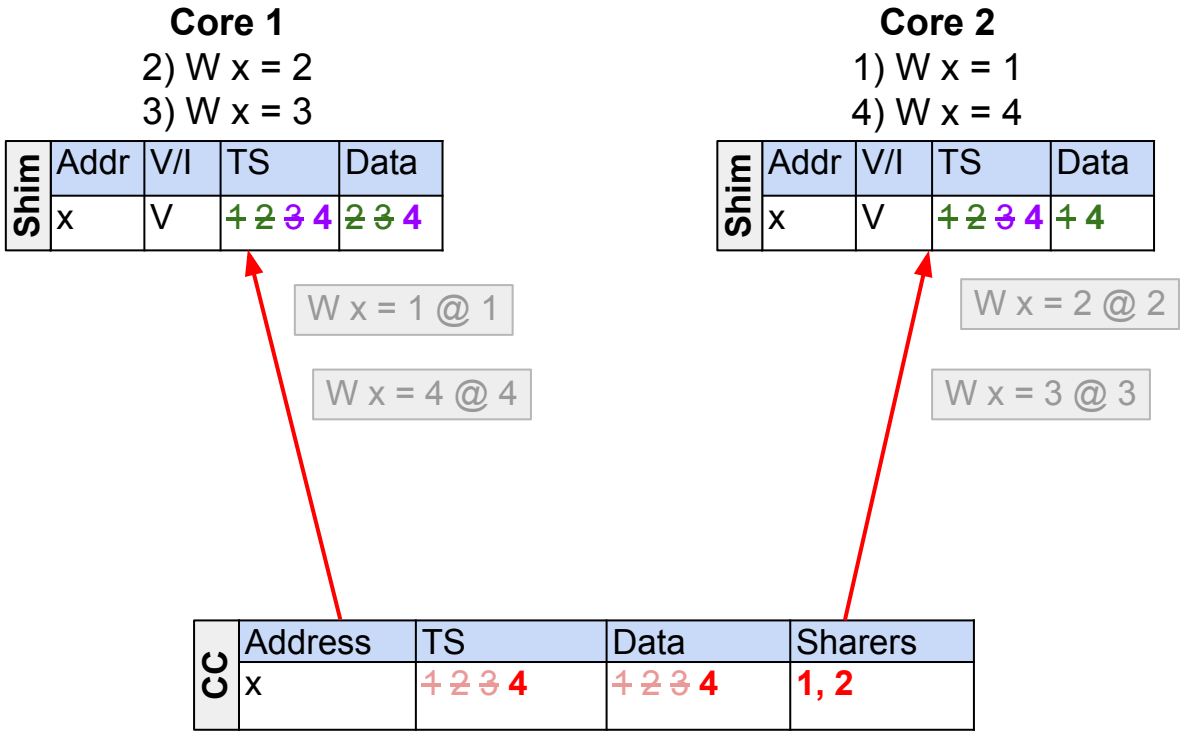
Shim Action:
1. If the message timestamp exceeds the shim timestamp, write the data.
2. Increment the shim timestamp.

MemGlue Overview: Timestamping



Shim Action:
1. If the message timestamp exceeds the shim timestamp, write the data.
2. Increment the shim timestamp.

MemGlue Overview: Timestamping



Unordered MemGlue: Additional Considerations

Messages in Unordered MemGlue can now arrive out-of-order.

Unordered MemGlue tracks additional metadata to reorder messages when necessary, or let them arrive out of order if doing so is permitted by C11.

Core 1
Wrlx x = 1
Wrel y = 1

Shim	Addr	V/I	TS	Data
	x	V	0	0
	y	V	0	0

Core 2
Racq y = 1
Rrlx x = 0

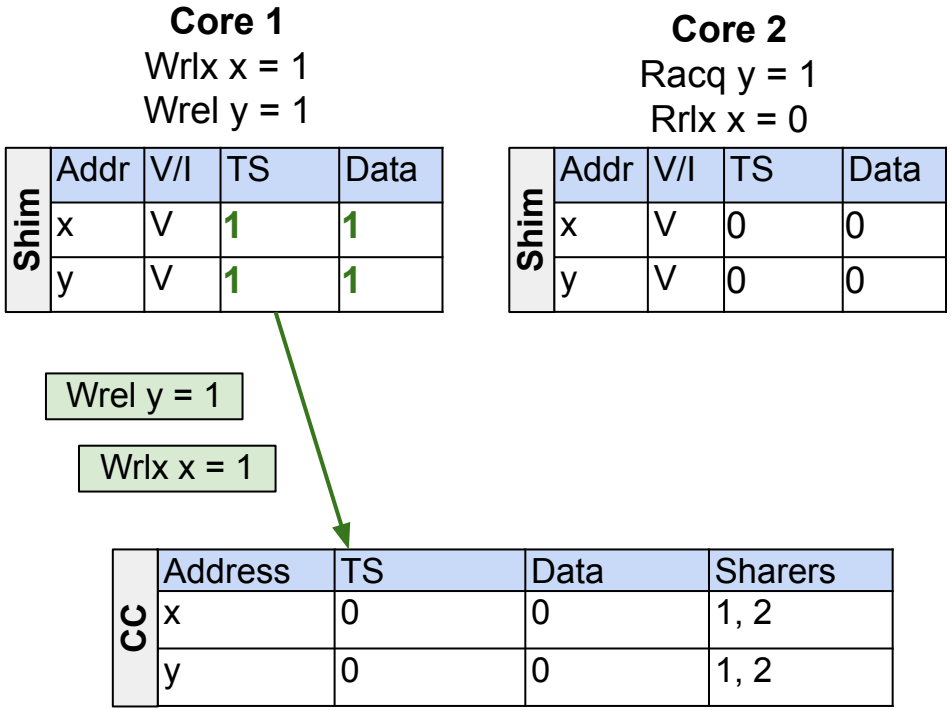
Shim	Addr	V/I	TS	Data
	x	V	0	0
	y	V	0	0

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

Unordered MemGlue: Additional Considerations

Messages in Unordered MemGlue can now arrive out-of-order.

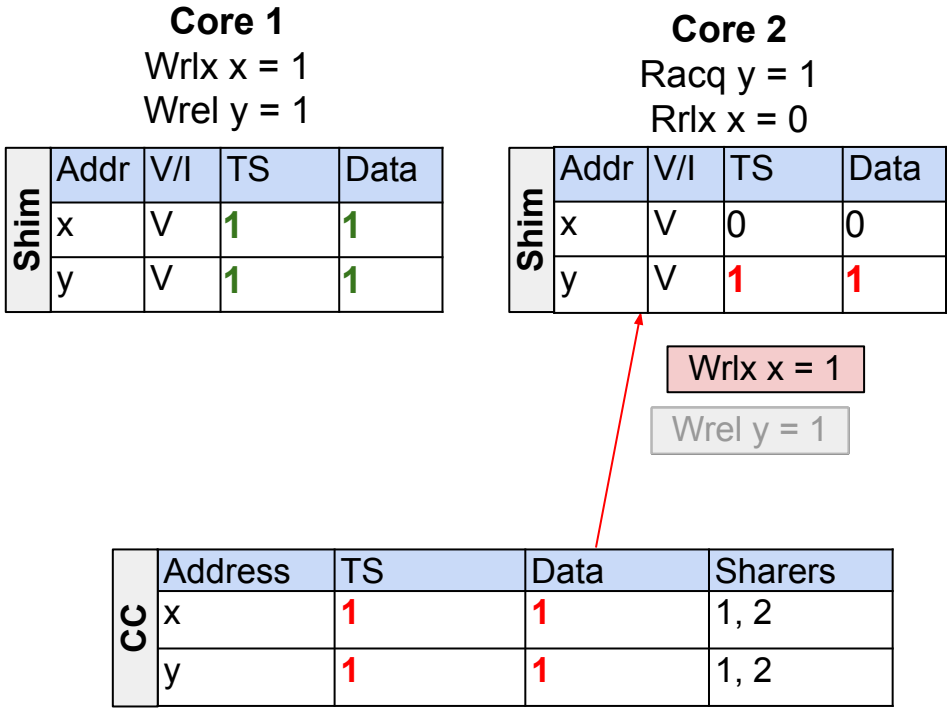
Unordered MemGlue tracks additional metadata to reorder messages when necessary, or let them arrive out of order if doing so is permitted by C11.



Unordered MemGlue: Additional Considerations

Messages in Unordered MemGlue can now arrive out-of-order.

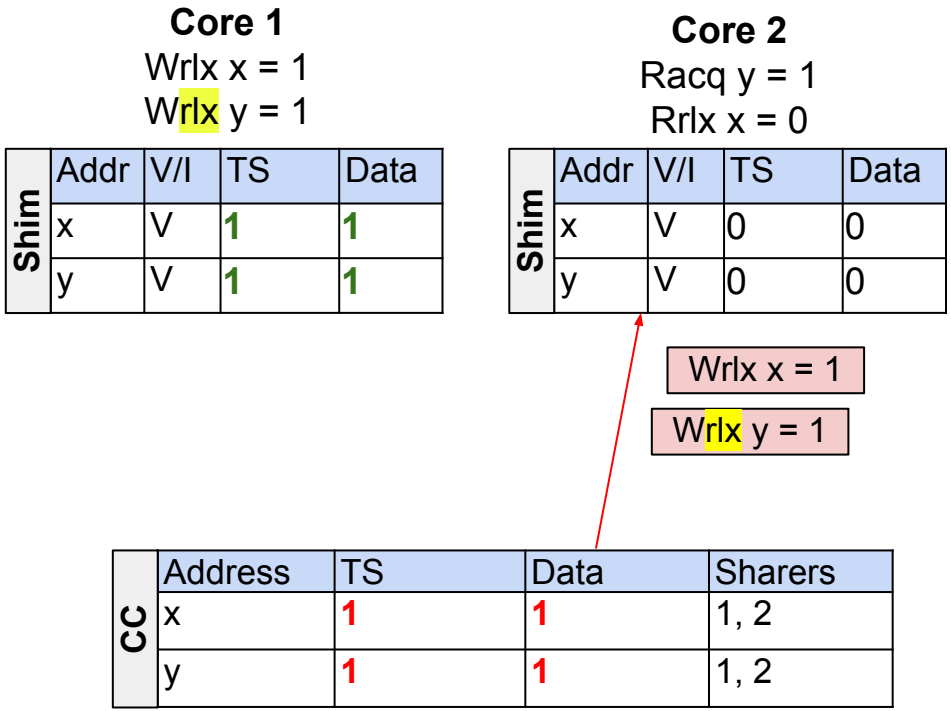
Unordered MemGlue tracks additional metadata to reorder messages when necessary, or let them arrive out of order if doing so is permitted by C11.



Unordered MemGlue: Additional Considerations

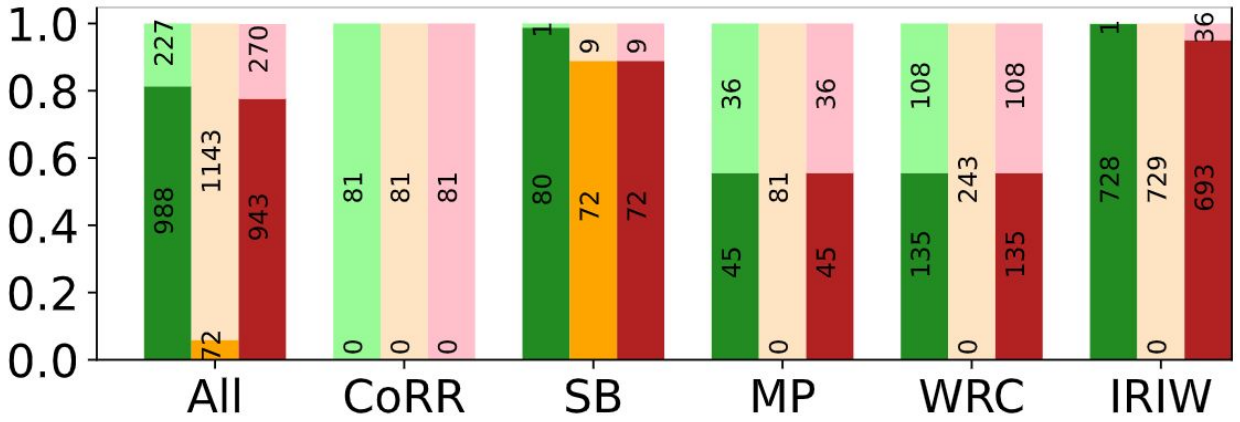
Messages in Unordered MemGlue can now arrive out-of-order.

Unordered MemGlue tracks additional metadata to reorder messages when necessary, or let them arrive out of order if doing so is permitted by C11.



Results

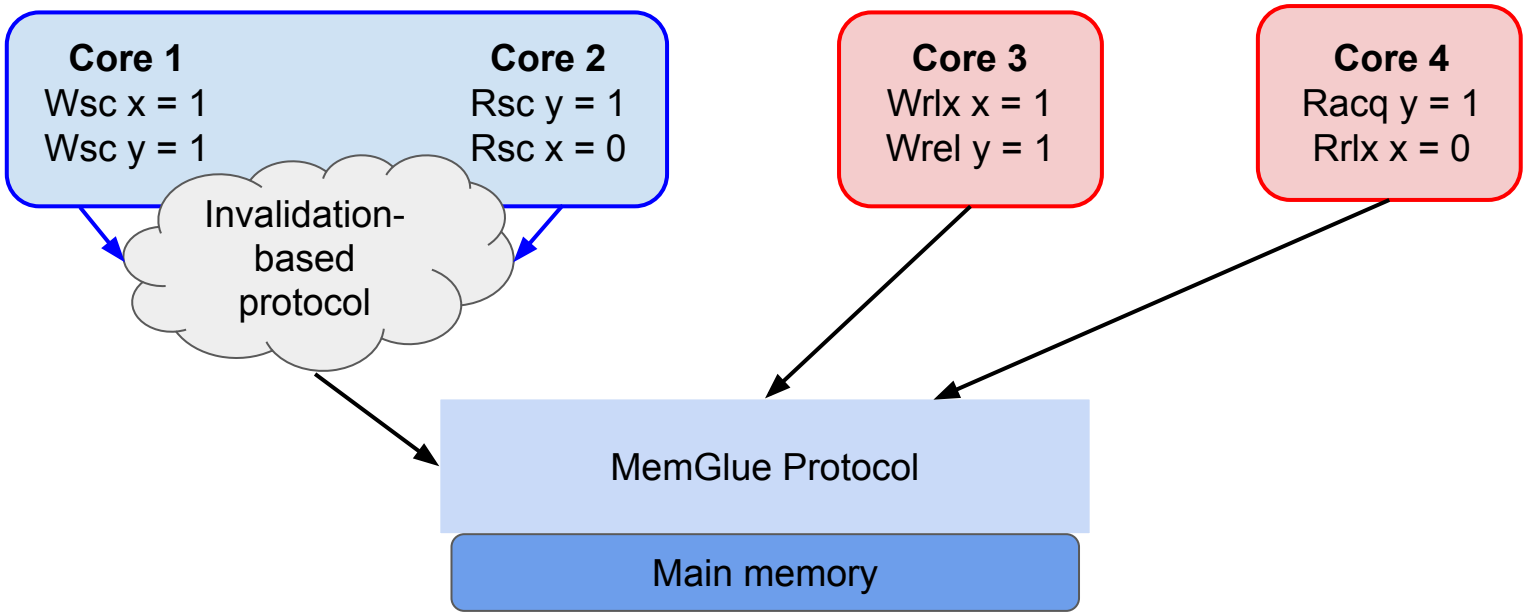
We implemented MemGlue in the Murphi model checker and checked its behavior against a suite of 6,738 litmus tests.



(a) Ordered (yellow) and Unordered (red) MemGlue results. The green columns represent what is permitted in C11. The darker colors are the portion of observable tests; the light colors are the portion that are unobservable.

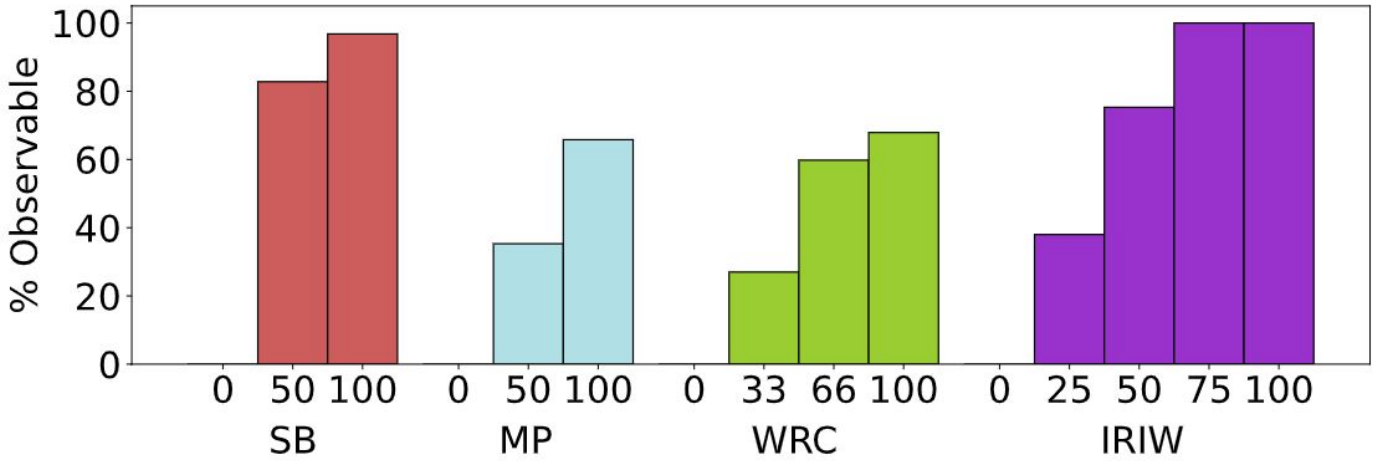
Results

We mapped the litmus tests across “strong” cores (approximating x86-TSO) and “weak” cores (leveraging C11 release and relaxed behavior) to demonstrate that MemGlue allows more behavior as the constituent MCMs become weaker.



Results

We mapped the litmus tests across “strong” cores (approximating x86-TSO) and “weak” cores (leveraging C11 release and relaxed behavior) to demonstrate that MemGlue allows more behavior as the constituent MCMs become weaker.



(c) Results of tests run on strong and weak cores.

Results: Formal Proof

Proof Goal: if a program execution is disallowed by the C11 memory model, then it should not be observable in MemGlue.

→ **Subgoals:** prove that the existence of C11-defined edges between instructions in a program implies that they hit the CC in this order.

E.g. Writes related by “happens-before” hit the CC in hb-order.

Future work: MemGlue As an Operational Model of C11