Orchestrating Coherence and Consistency in Heterogeneous Shared Memory Systems

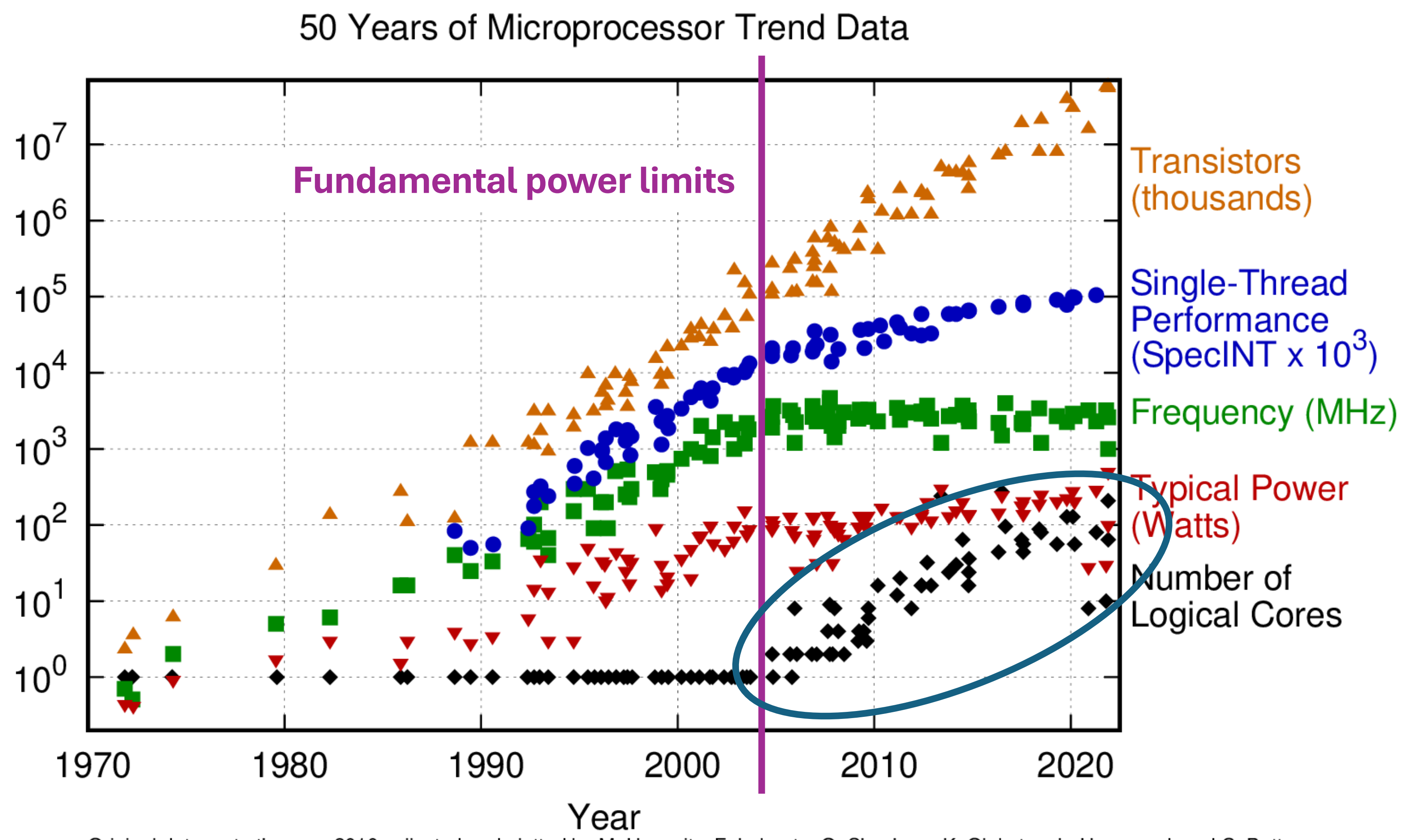
Caroline Trippel

Assistant Professor of Computer Science & Electrical Engineering

Stanford Differentiated Access Memories Project

May 10, 2024

20 years ago, fundamental power limits forced a move to multi-core processors



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

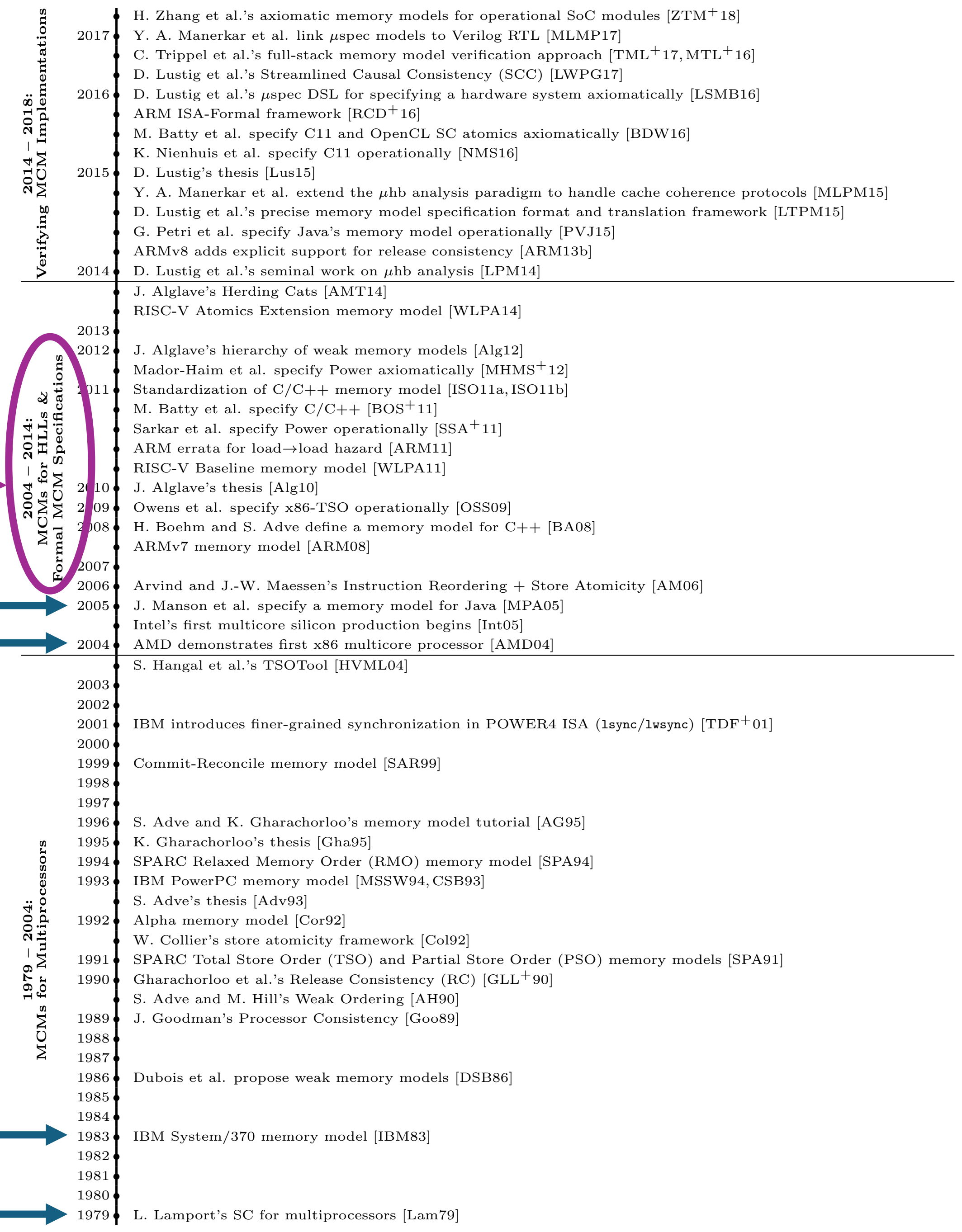
Memory consistency model formalization efforts

2005: Intel's first **multicore** silicon production begins

2004: AMD demonstrates first x86 **multicore** processor

1983: **IBM System/370** memory consistency model

1979: Lamport's **sequential consistency** for multiprocessors



2014 – 2018: Verifying MCM Implementations

- H. Zhang et al.'s axiomatic memory models for operational SoC modules [ZTM$^+$18]
- 2017 Y. A. Manerkar et al. link μspec models to Verilog RTL [MLMP17]
- C. Trippel et al.'s full-stack memory model verification approach [TML$^+$17, MTL$^+$16]
- D. Lustig et al.'s Streamlined Causal Consistency (SCC) [LWPG17]
- 2016 D. Lustig et al.'s μspec DSL for specifying a hardware system axiomatically [LSMB16]
- ARM ISA-Formal framework [RCD$^+$16]
- M. Batty et al. specify C11 and OpenCL SC atomics axiomatically [BDW16]
- K. Nienhuis et al. specify C11 operationally [NMS16]
- 2015 D. Lustig's thesis [Lus15]
- Y. A. Manerkar et al. extend the μhb analysis paradigm to handle cache coherence protocols [MLPM15]
- D. Lustig et al.'s precise memory model specification format and translation framework [LTPM15]
- G. Petri et al. specify Java's memory model operationally [PVJ15]
- ARMv8 adds explicit support for release consistency [ARM13b]
- 2014 D. Lustig et al.'s seminal work on μhb analysis [LPM14]

2004 – 2014: MCMs for HLLs & Formal MCM Specifications

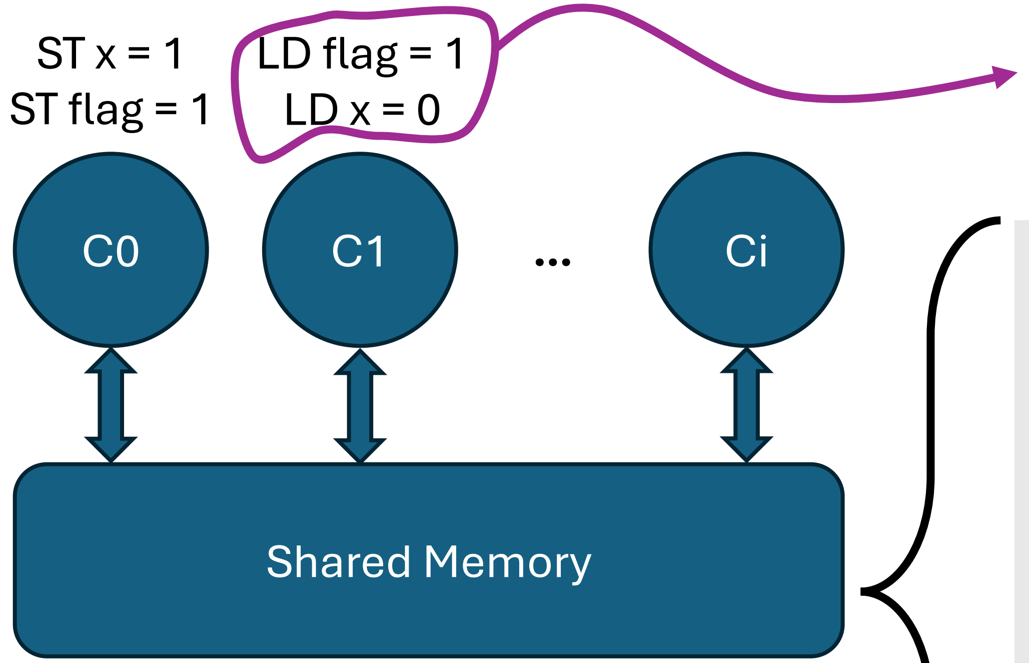
- J. Alglave's Herding Cats [AMT14]
- RISC-V Atomics Extension memory model [WLPA14]
- 2013
- 2012 J. Alglave's hierarchy of weak memory models [Alg12]
- Mador-Haim et al. specify Power axiomatically [MHMS$^+$12]
- 2011 Standardization of C/C++ memory model [ISO11a, ISO11b]
- M. Batty et al. specify C/C++ [BOS$^+$11]
- Sarkar et al. specify Power operationally [SSA$^+$11]
- ARM errata for load$\rightarrow$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]

1979 – 2004: MCMs for Multiprocessors

- S. Hangal et al.'s TSOTool [HVML04]
- 2003
- 2002
- 2001 IBM introduces finer-grained synchronization in POWER4 ISA (`lsync`/`lwsync`) [TDF$^+$01]
- 2000
- 1999 Commit-Reconcile memory model [SAR99]
- 1998
- 1997
- 1996 S. Adve and K. Gharachorloo's memory model tutorial [AG95]
- 1995 K. Gharachorloo's thesis [Gha95]
- 1994 SPARC Relaxed Memory Order (RMO) memory model [SPA94]
- 1993 IBM PowerPC memory model [MSSW94, CSB93]
- S. Adve's thesis [Adv93]
- 1992 Alpha memory model [Cor92]
- W. Collier's store atomicity framework [Col92]
- 1991 SPARC Total Store Order (TSO) and Partial Store Order (PSO) memory models [SPA91]
- 1990 Gharachorloo et al.'s Release Consistency (RC) [GLL$^+$90]
- S. Adve and M. Hill's Weak Ordering [AH90]
- 1989 J. Goodman's Processor Consistency [Goo89]
- 1988
- 1987
- 1986 Dubois et al. propose weak memory models [DSB86]
- 1985
- 1984
- 1983 IBM System/370 memory model [IBM83]
- 1982
- 1981
- 1980
- 1979 L. Lamport's SC for multiprocessors [Lam79]

Memory consistency (and **cache coherence**) for homogeneous compute, homogeneous memory

Is this a legal program outcome?

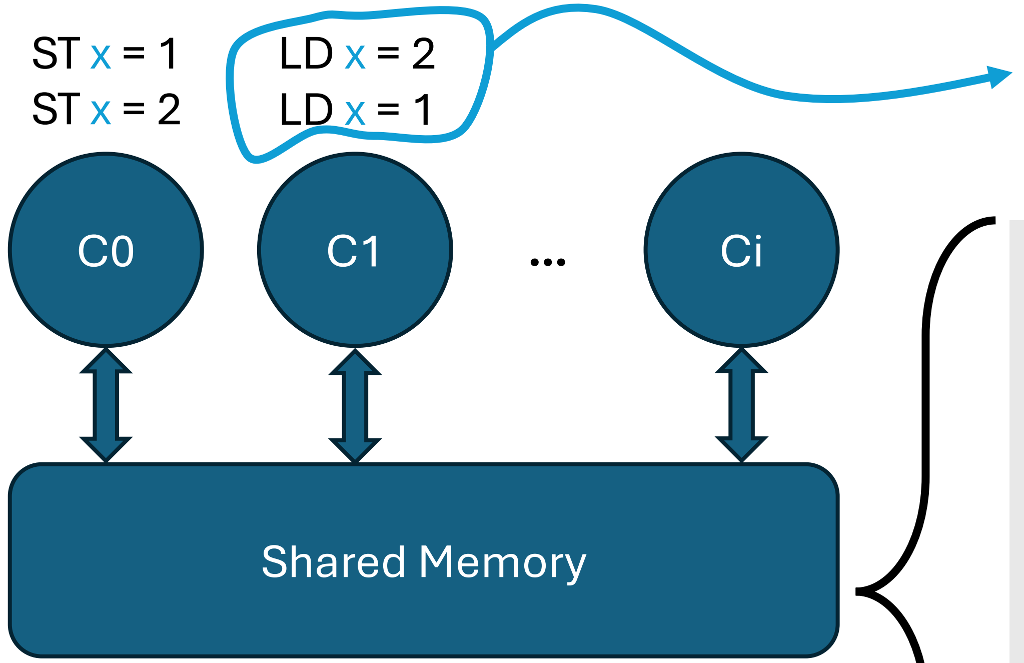


Forbidden: SC, x86-TSO, RVTSO
Permitted: Arm, Power, RVWMO, PTX

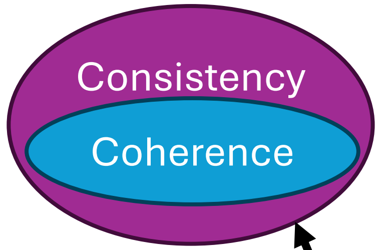
"For a shared memory machine, the **memory consistency model [MCM]** defines the architecturally visible behavior of its memory system. Consistency definitions provide **rules about loads and stores** (or memory reads and writes) and **how they act upon memory**. As part of supporting a memory consistency model, many machines also provide **cache coherence protocols** that ensure that multiple cached copies of data are kept up-to-date."

Memory consistency (and cache coherence) for homogeneous compute, homogeneous memory

C1 reads values for x in a different order than C0 writes them!



Forbidden: All cache-coherent architectures



"For a shared memory machine, the **memory consistency model [MCM]** defines the architecturally visible behavior of its memory system. Consistency definitions provide **rules about loads and stores** (or memory reads and writes) and **how they act upon memory**. As part of supporting a memory consistency model, many machines also provide **cache coherence protocols** that ensure that multiple cached copies of data are kept up-to-date."

Orchestrating correct parallel program execution for homogeneous compute, homogeneous memory

```
atomic_bool flag; // C11 memory model
atomic_int x;

// thread 0                    // thread 1
x.store(1, RLX)                if (flag.load(ACQ) == true)
flag.store (true, REL)            assert (x.load(RLX) == 1)
```

X X

High-level language (HLL) MCMs specify the **ordering requirements** of memory operations in a program.

C11 MCM says that assert cannot fail.

HLL-to-ISA MCM compiler mappings

```
// core 0                      // core 1
ST X = 1                         LD flag = r1
                                 cmp r1, #1
ST flag = 1                      bne end

                                 LD X → r2
                               end:
```

? ?

Instruction set architecture (ISA) MCM specifies the **ordering guarantees** of memory operations executing on hardware.

Forbidden: SC, x86-TSO, RVTSO
Permitted: Arm, Power, RVWMO, PTX

Orchestrating correct parallel program execution for homogeneous compute, homogeneous memory

```
atomic_bool flag;
atomic_int x;

// thread 0                       // thread 1
x.store(1, RLX)                   if (flag.load(ACQ) == true)
flag.store (true, REL)              assert (x.load(RLX) == 1)
```

High-level language (HLL) MCMs specify the **ordering requirements** of memory operations in a program.

C11 MCM says that assert cannot fail.

HLL-to-ISA MCM compiler mappings

```
// core 0                         // core 1
ST X = 1                            LD flag = r1
FENCE                               cmp r1, #1
ST flag = 1                         bne end
                                    FENCE
                                    LD X → r2
                                  end:
```

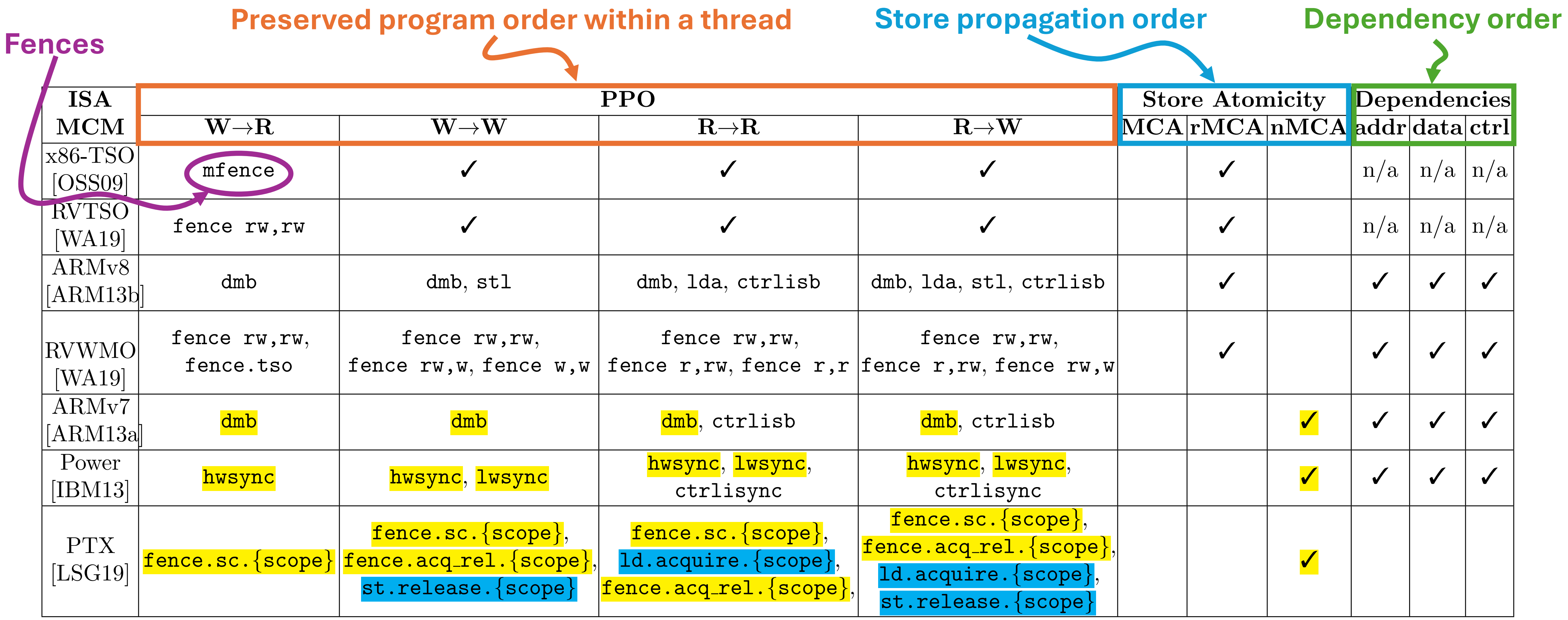
Need **fences** to forbid illegal execution.

Instruction set architecture (ISA) MCM specifies the **ordering guarantees** of memory operations executing on hardware.

Forbidden: SC, x86-TSO, RVTSO

Permitted: Arm, Power, RVWMO, PTX

Landscape of ISA Memory Consistency Models


Fences
Preserved program order within a thread
Store propagation order
Dependency order

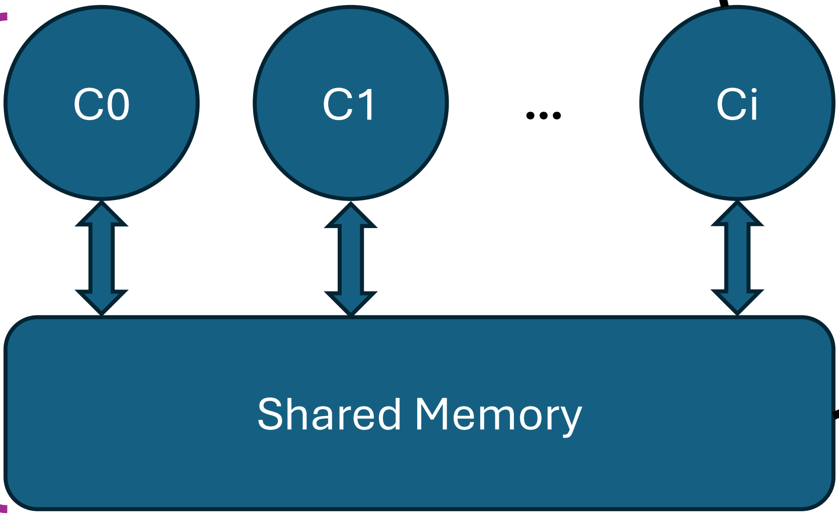
ISA MCM	PPO				Store Atomicity			Dependencies		
	W→R	W→W	R→R	R→W	MCA	rMCA	nMCA	addr	data	ctrl
x86-TSO [OSS09]	mfence	✓	✓	✓		✓		n/a	n/a	n/a
RVTSO [WA19]	fence rw,rw	✓	✓	✓		✓		n/a	n/a	n/a
ARMv8 [ARM13b]	dmb	dmb, stl	dmb, lda, ctrlisb	dmb, lda, stl, ctrlisb		✓		✓	✓	✓
RVWMO [WA19]	fence rw,rw, fence.tso	fence rw,rw, fence rw,w, fence w,w	fence rw,rw, fence r,rw, fence r,r	fence rw,rw, fence r,rw, fence rw,w		✓		✓	✓	✓
ARMv7 [ARM13a]	dmb	dmb	dmb, ctrlisb	dmb, ctrlisb			✓	✓	✓	✓
Power [IBM13]	hwsync	hwsync, lwsync	hwsync, lwsync, ctrlisync	hwsync, lwsync, ctrlisync			✓	✓	✓	✓
PTX [LSG19]	fence.sc.{scope}	fence.sc.{scope}, fence.acq_rel.{scope}, st.release.{scope}	fence.sc.{scope}, ld.acquire.{scope}, fence.acq_rel.{scope},	fence.sc.{scope}, fence.acq_rel.{scope}, ld.acquire.{scope}, st.release.{scope}			✓			

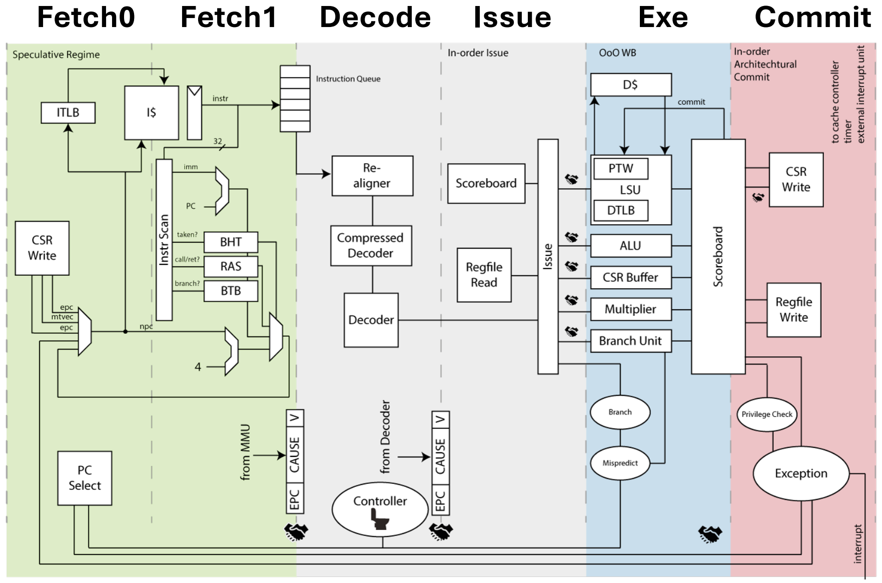
Challenge #1: How do we ensure that microarchitecture correctly implements its ISA MCM?

```
// core 0        // core 1
ST X = 1            LD flag = r1
FENCE               cmp r1, #1
ST flag = 1         bne end
                    FENCE
                    LD X → r2
                 end:
```

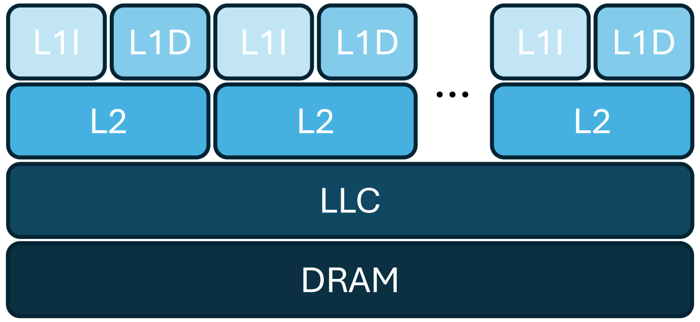
"stores update memory"
"stores update memory in program order"

SOTA: Top-down verification: Teams of engineers manually encode **formal MCM properties**, map down to RTL signals, and evaluate with model checkers to get **bounded proofs**.





RISC-V CVA6 **Core Microarchitecture** [Zaruba & Benini, GitHub'19]



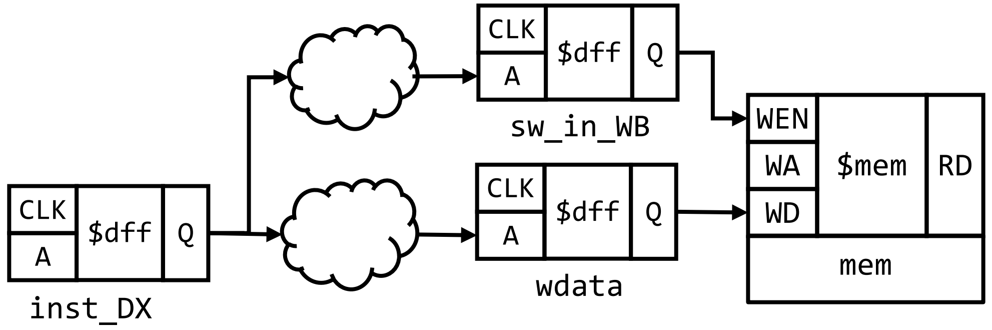
Cache coherent shared memory

Our Approach: Bottom-up, Push-button Formal Verification of Hardware MCM Implementations

[Hsiao+, MICRO'21]

Design Metadata
+

Case Study:
RISC-V multi-V-scale [Magyar, GitHub'16]



#1 **Static Netlist Analysis**

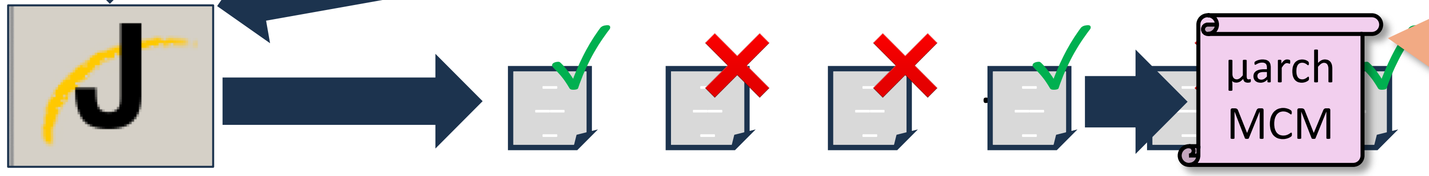
#2 **Over-approximation** of MCM "state update" and "ordering" guarantees

A0: State element **s** will never be updated by instruction **i0** with opcode **op**.

```
P0: assume (first |-> ( (`PCR_0 != pc0 [*0:$]) ##1
        (`PCR_0 == pc0 [*1:$]) ##1 (`PCR_0 != pc0) ));
P1: assume (first |-> s_eventually(`PCR_<stage(s)> == pc0));
P2: assume (`PCR_0 == pc0 |-> `IFR == i0);
P3: assume (opcode(i0) == op);
A0: assert (`PCR_<stage(s)> == pc0 |-> s == $past(s));
```

```
P4: assume (`PCR_0 == pc0 |-> `IFR == i0);
P5: assume (opcode(i0) == op);
P6: assume (first |-> strong((`IFR == `NOP &&
        `PCR_0 != pc0 [*0:$]) ##1 (`PCR_0 == pc0) ) );
A1: assert (first |-> s_eventually( (`PCR_<stage> == pc0) ##1
        (!(`PCR_<stage> == pc0)) ));
```

#3 **SystemVerilog Assertion (SVA)**

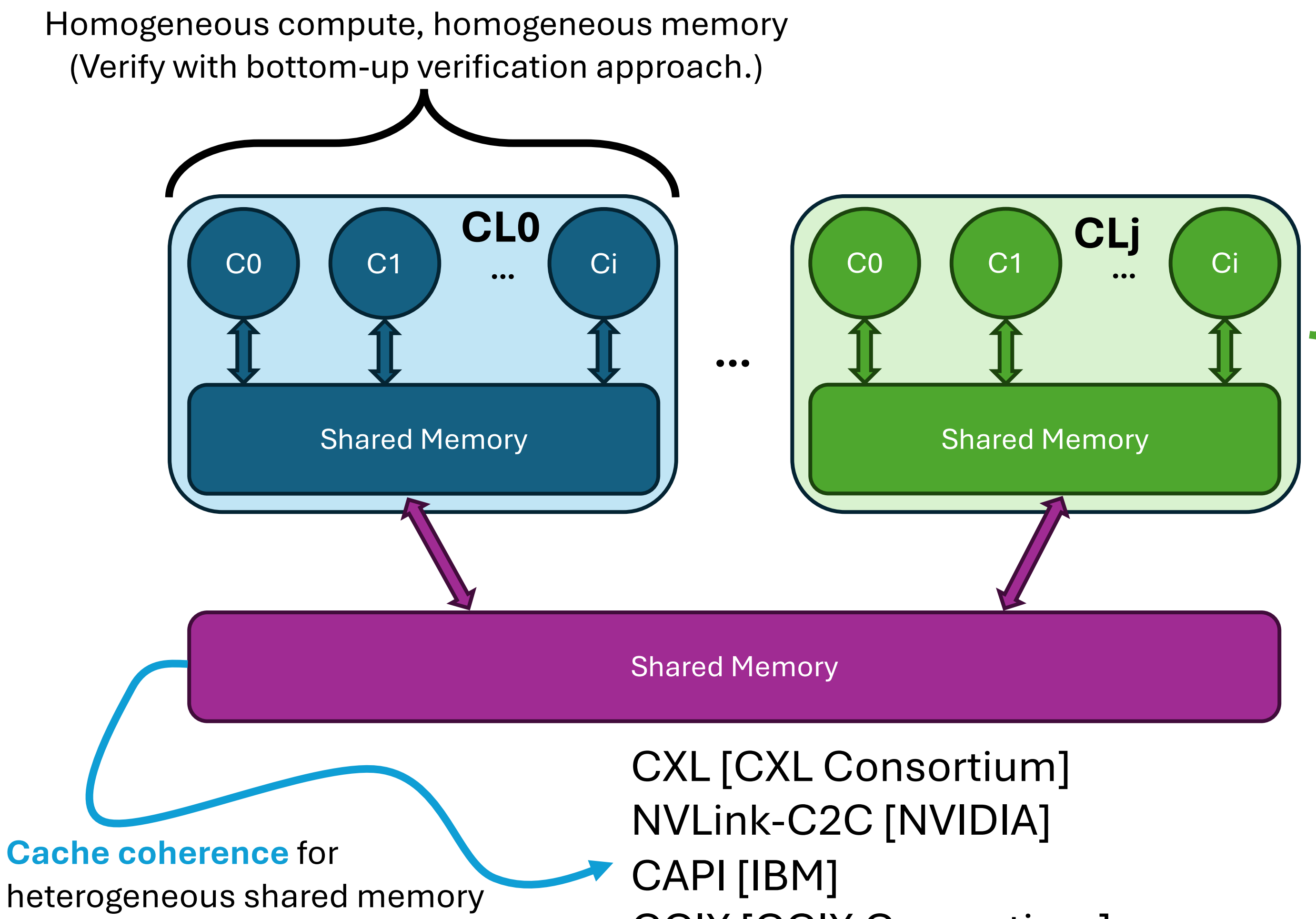


#4 **Model Checkers**

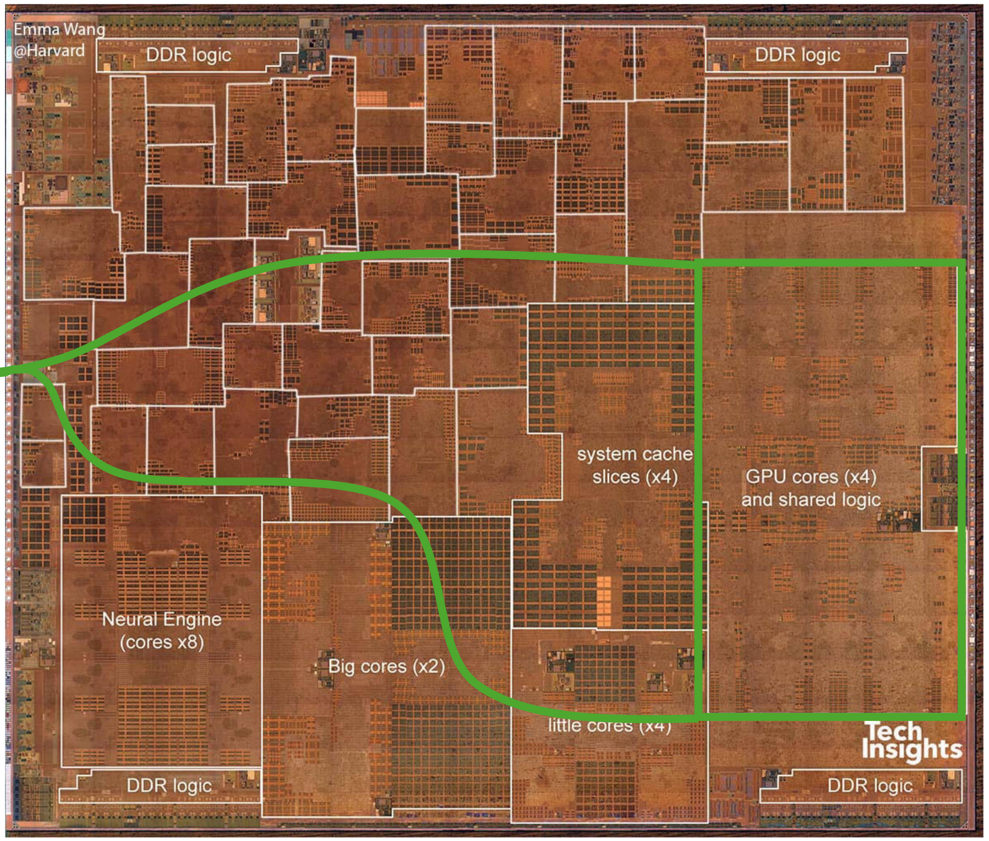
#5 **Sound & complete** formal specification

Multi-V-scale μarch MCM synthesis: **~7 mins** for **122 SVAs** with **no bounded proofs**. SOTA hits 11-hour timeout on many proofs [Manerkar, MICRO'17].

Cache coherence for heterogeneous compute, homogeneous memory

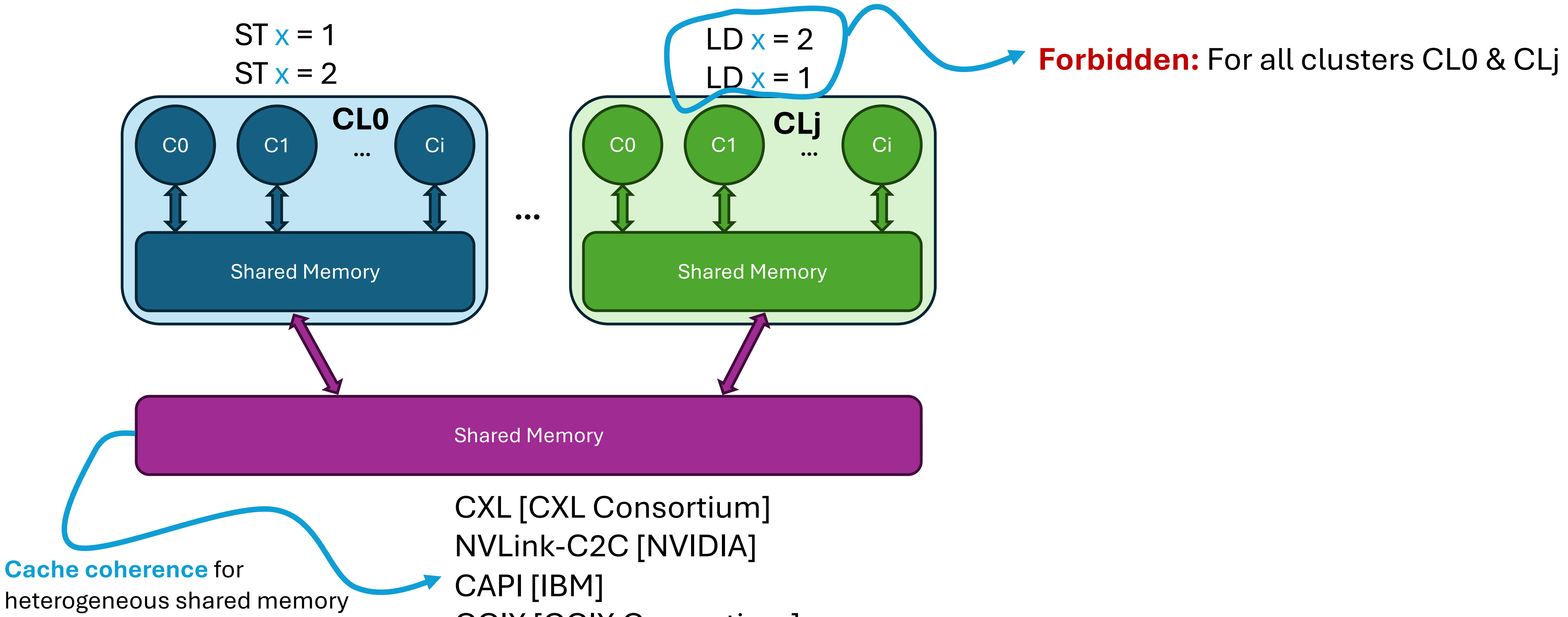

Homogeneous compute, homogeneous memory
(Verify with bottom-up verification approach.)
CL0
C0
C1
...
Ci
Shared Memory
...
CLj
C0
C1
...
Ci
Shared Memory
Shared Memory
Cache coherence for heterogeneous shared memory

CXL [CXL Consortium]
NVLink-C2C [NVIDIA]
CAPI [IBM]
CCIX [CCIX Consortium]
...



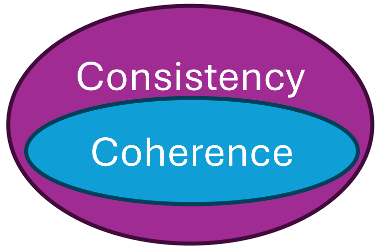
2019 Apple A12 (iPhone XS, XS Max, XR)
40+ accelerators

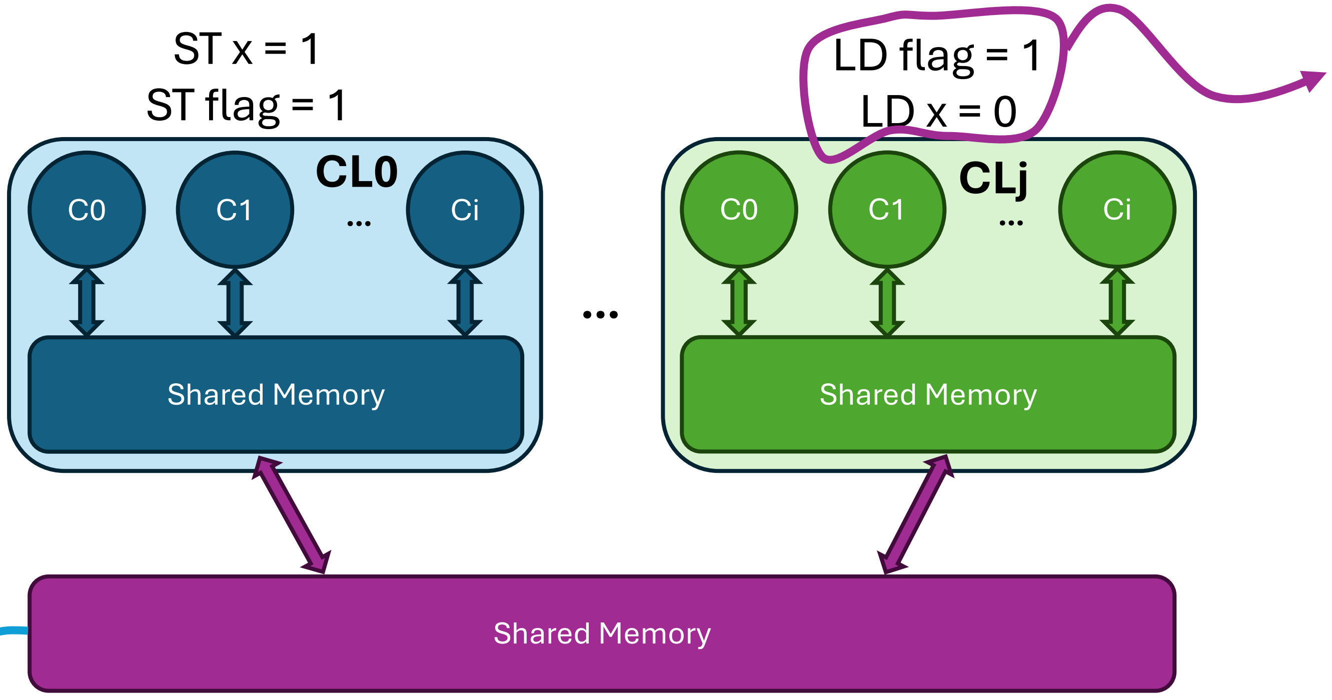
Cache coherence for heterogeneous compute, homogeneous memory



CXL [CXL Consortium]
NVLink-C2C [NVIDIA]
CAPI [IBM]
CCIX [CCIX Consortium]
...

Memory consistency for heterogeneous compute, homogeneous memory



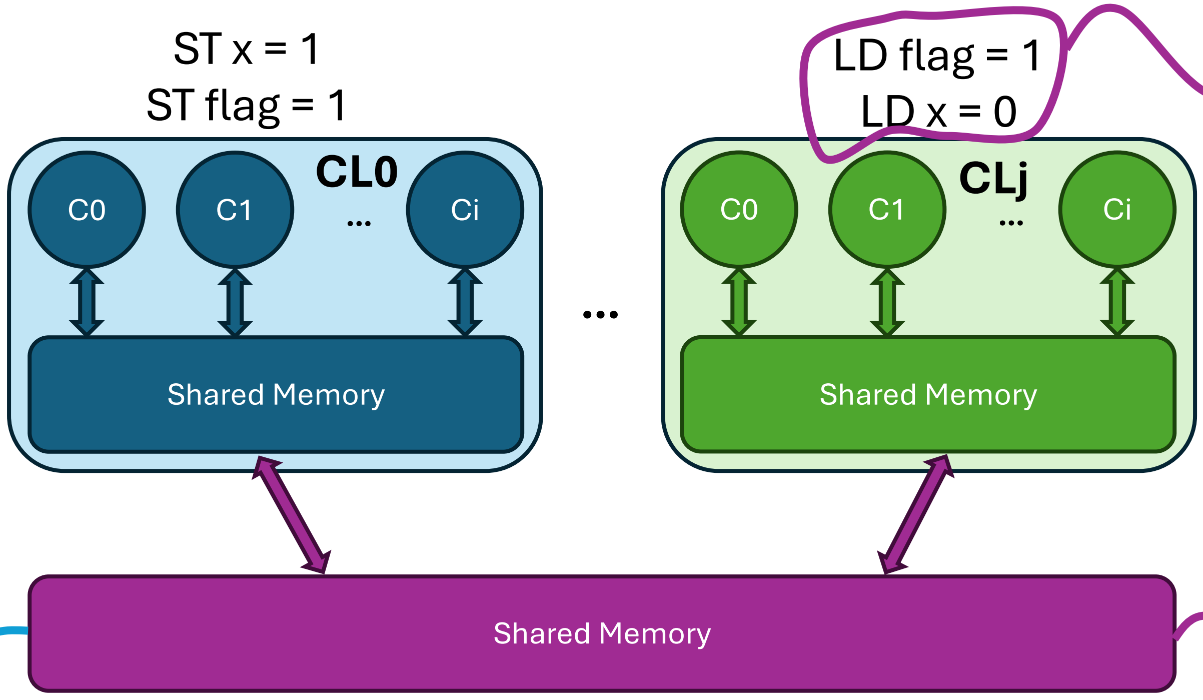


Forbidden: For some CL0 & CLj
Permitted: For some CL0 & CLj

Cache coherence for heterogeneous shared memory

CXL [CXL Consortium]
NVLink-C2C [NVIDIA]
CAPI [IBM]
CCIX [CCIX Consortium]
...

Challenge #2: How should we fuse heterogeneous clusters while upholding their MCM guarantees?

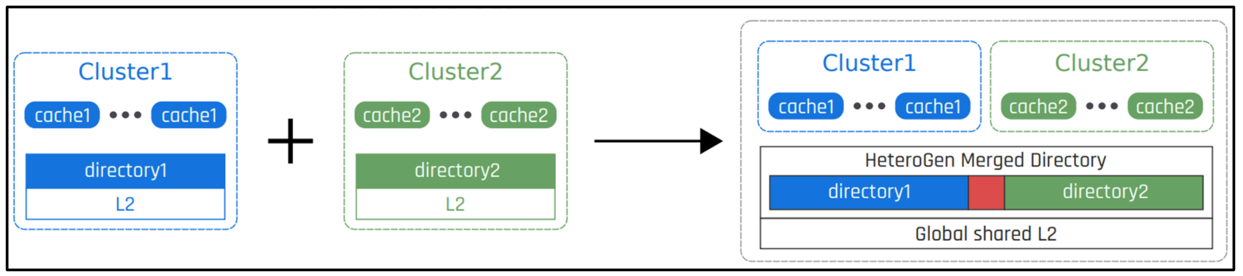


Forbidden: For some CL0 & CLj
Permitted: For some CL0 & CLj

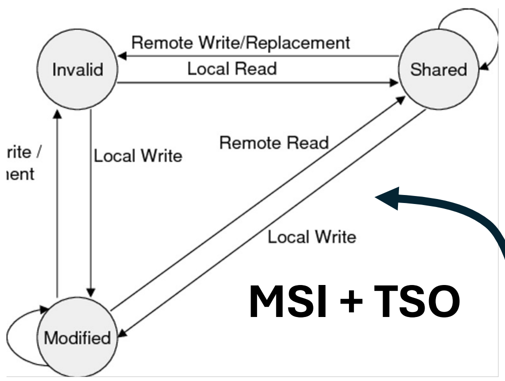
SOTA: Redesign & re-verify memory system for every new combination of compute clusters (e.g., [Oswald+,

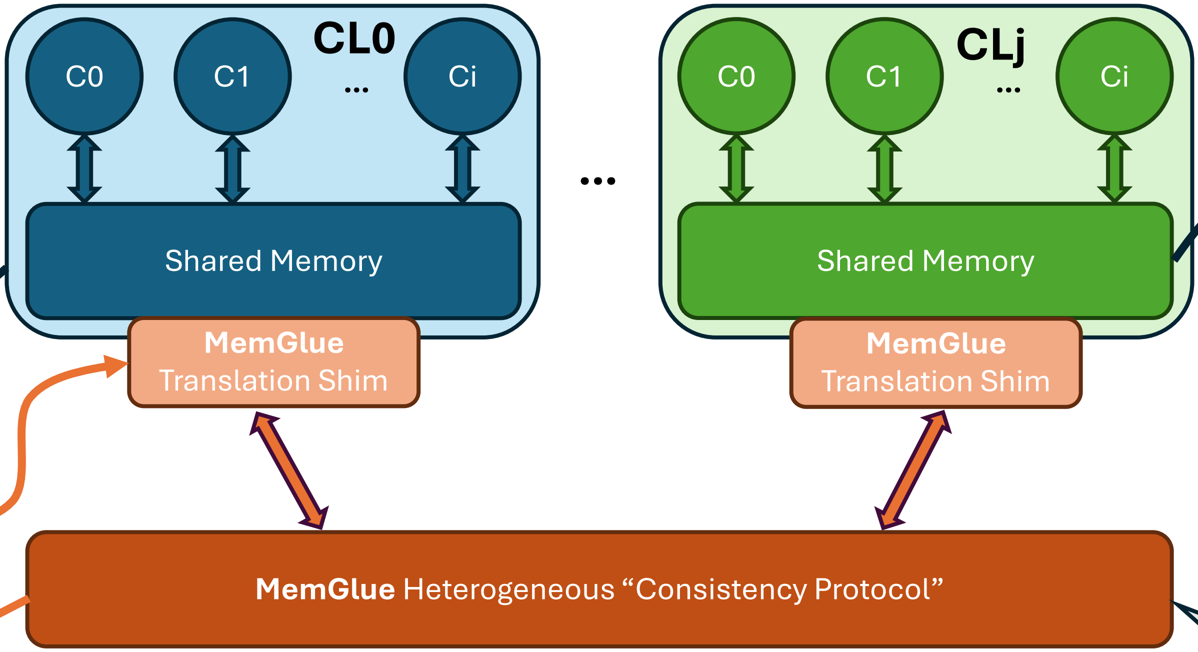
Cache coherence for heterogeneous shared memory

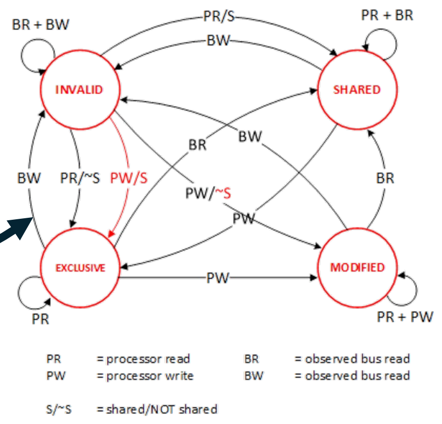
CXL [CXL Consortium]
NVLink-C2C [NVIDIA]
CAPI [IBM]
CCIX [CCIX Consortium]
...



Our Approach: Modular **MCM-aware** coherence protocol that is designed once & verified once[Cleaveland+, In Preparation]







Lazy MESI + Armv7

Translation shim: translates local coherence protocol messages into MemGlue messages, accounting for cluster's MCM.

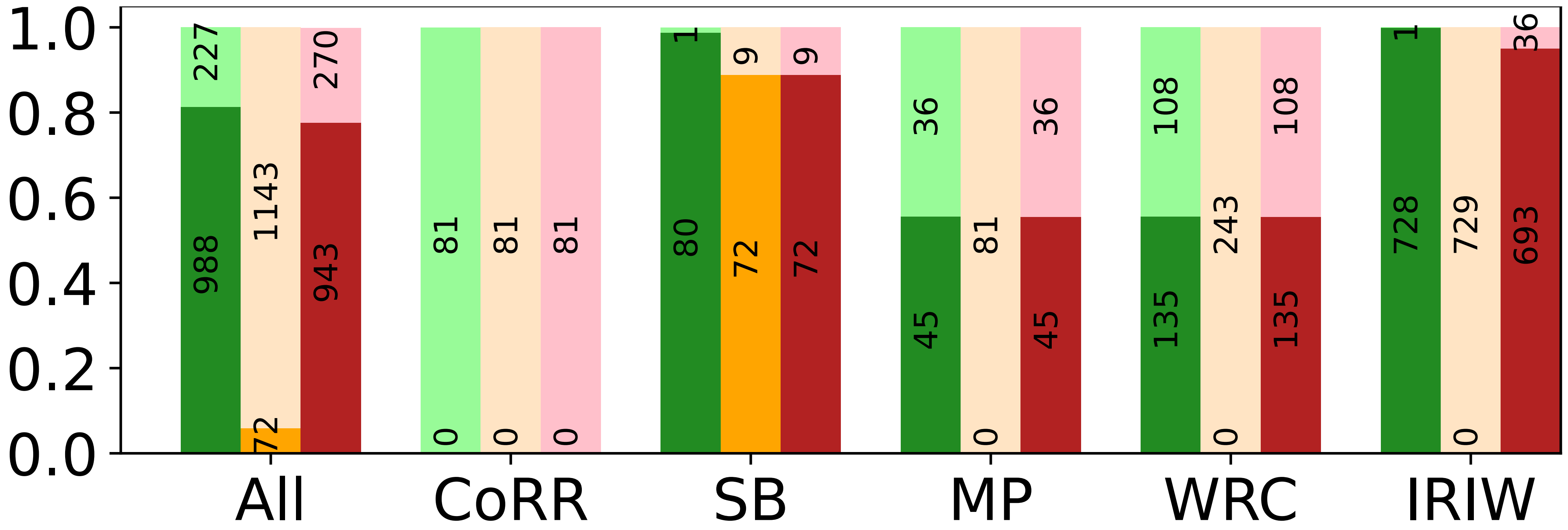
MemGlue protocol:

- Coherence protocol that enforces the C11 MCM (a form of release consistency) globally.
- *Does not* enforce single-writer multiple reader, so as to fully exploit relaxed ordering of cluster MCMs
- Any C11-compatible cluster can be "plugged in"
- Currently "update-based" for producer-consumer sharing

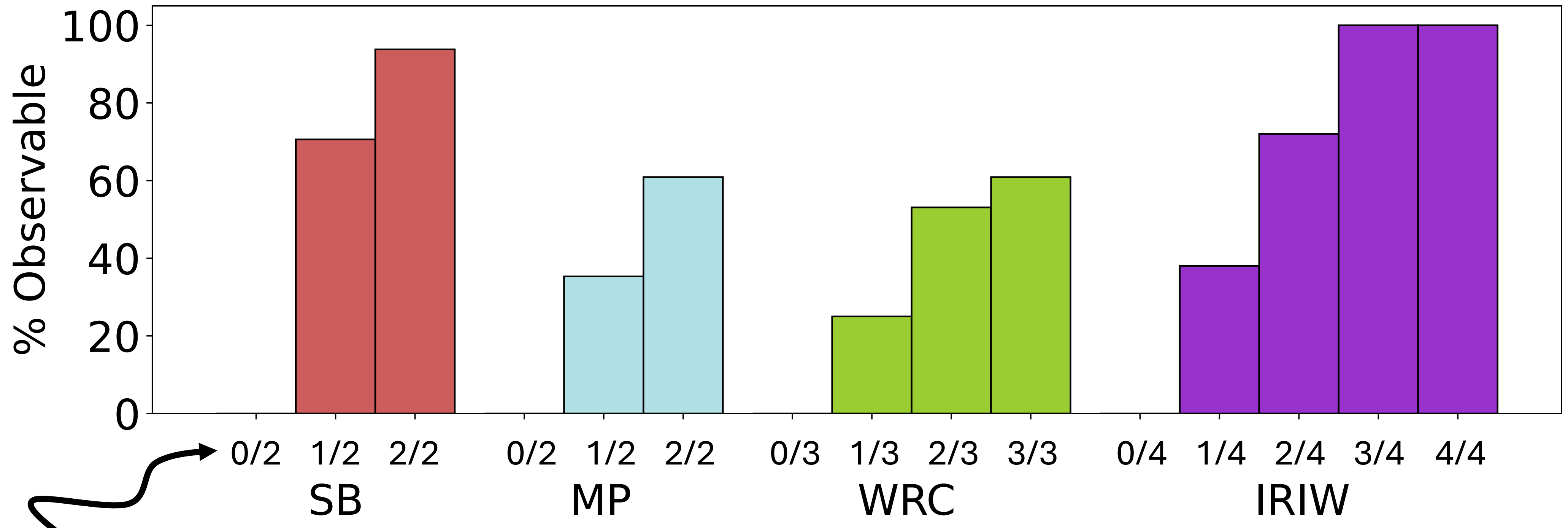
Co-design opportunity: By having software **checkIn** and **checkOut** coherence units (e.g., cache lines), we can reduce protocol traffic.

Preliminary Results: MemGlue nearly matches C11 ordering semantics for 6,738 litmus tests

- Manual complete proof that MemGlue enforces C11 for all programs
- Bounded model checker proof (Murphi) for 6,738 litmus test programs
- Dark/light colors: permitted/forbidden
- **Green:** C11, **Yellow:** Ordered MemGlue, **Red:** Unordered MemGlue **(our proposal)**



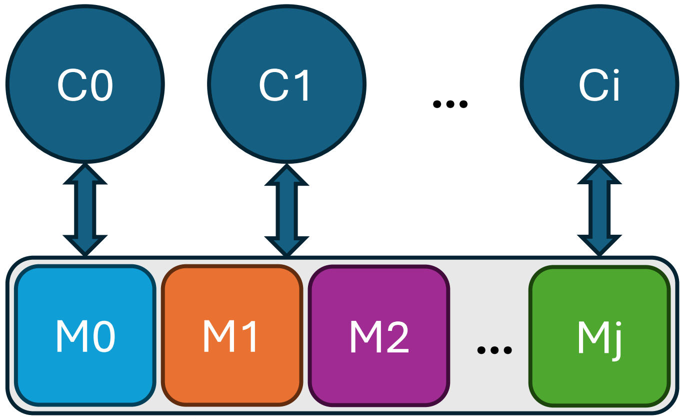
Preliminary Results: MemGlue exploits weak ordering behavior as permitted by clusters



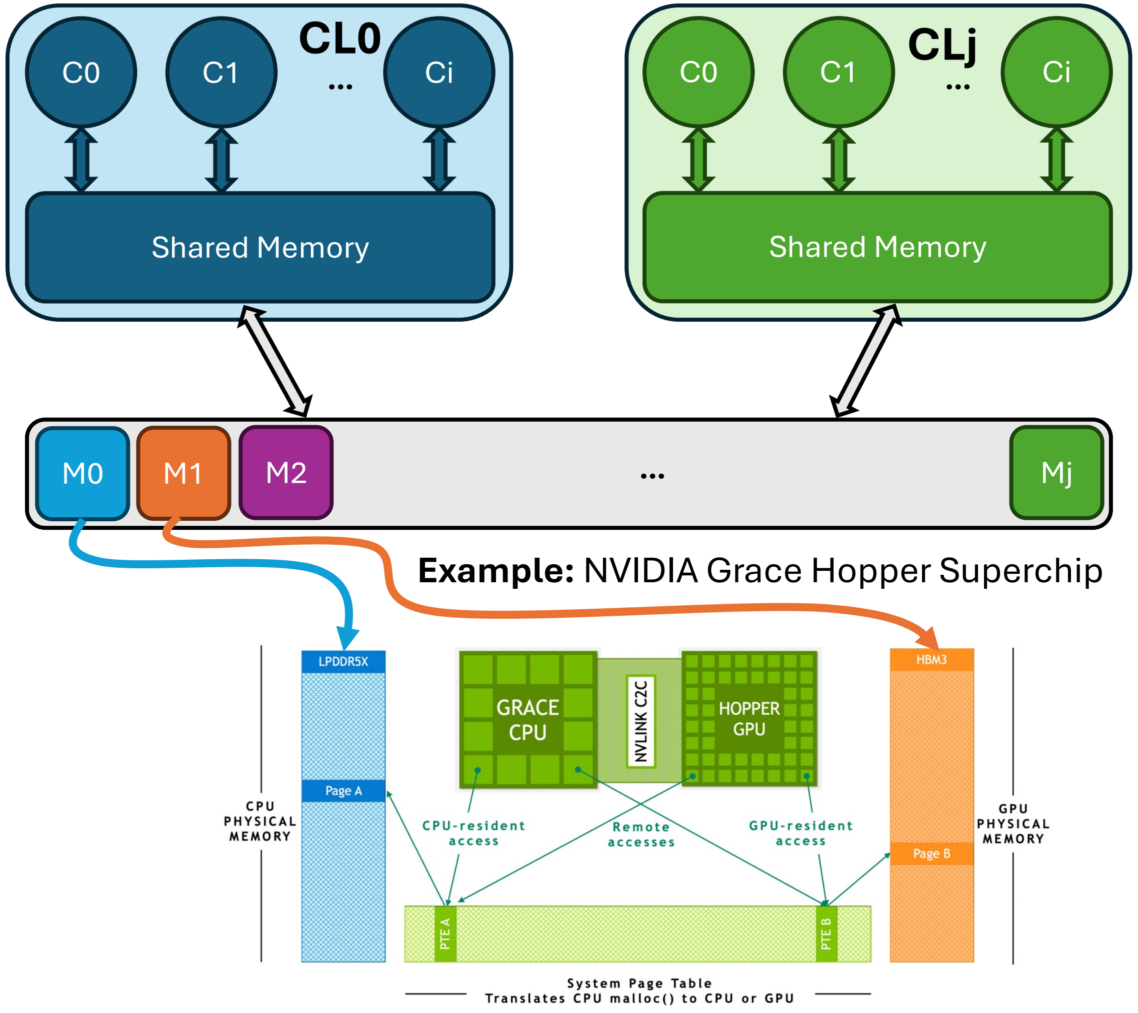
Weak threads / **Strong** threads

Next steps: Implement MemGlue as a hardware prototype.

Opportunities for Differentiated Access (Shared) Memory Architectures



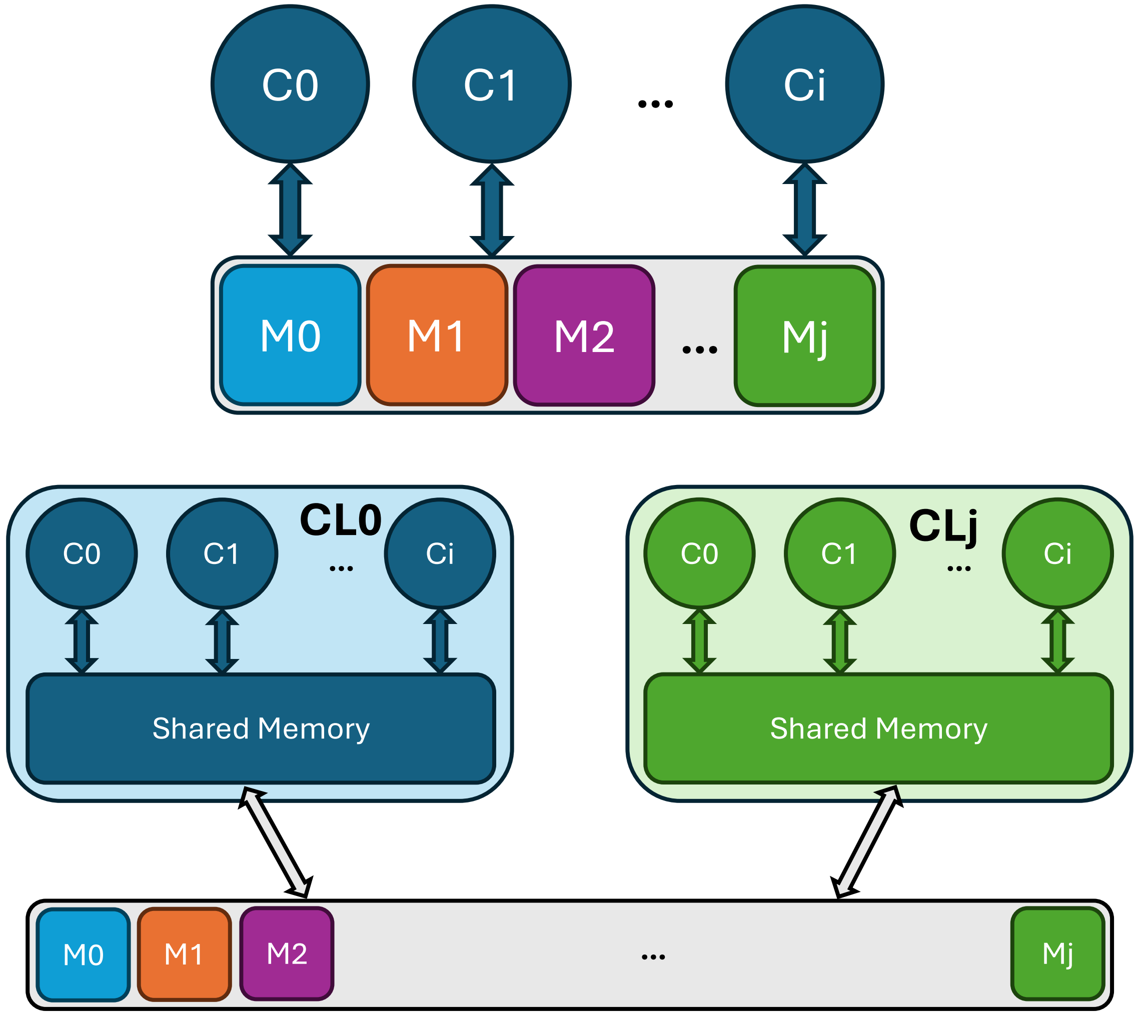
Co-design opportunity: Data structure granularity coherence to amortize metadata and protocol communication overheads.



Example: NVIDIA Grace Hopper Superchip

https://developer.nvidia.com/blog/nvidia-grace-hopper-superchip-architecture-in-depth/

Challenges for Differentiated Access (Shared) Memory Architectures



- **Challenge #3:** Formalizing **new reordering behaviors** for software:
  - **Concurrency created within a thread** if data structures are mapped to distinct memories.
  - **Persistency mismatches** between heterogeneous memories.
  - **Data-structure granularity coherence**
  - **Bounded de-synchronization** may be permissible for certain applications (e.g., ML)
- **Challenge #4:** Designing **new safety-nets** to recover ordering when needed

Summary of Shared Memory Research Challenges

Homogeneous multicores

Heterogeneous SoCs, SiPs, datacenters

		Compute	
		Homogeneous	Heterogeneous
Memory	Homogeneous	① Push-button verification of hardware MCM implementations	② Modular MCM-aware coherent shared memory for heterogeneous shared memory
	Heterogeneous	③ Formally specifying architectural MCMs for DAMs ④ Orchestrating correct shared memory parallelism on DAMs	

DAM Systems

Key takeaway: We're just getting to the point of specifying/verifying memory consistency in non-DAM systems…DAM systems will make these problems much harder!