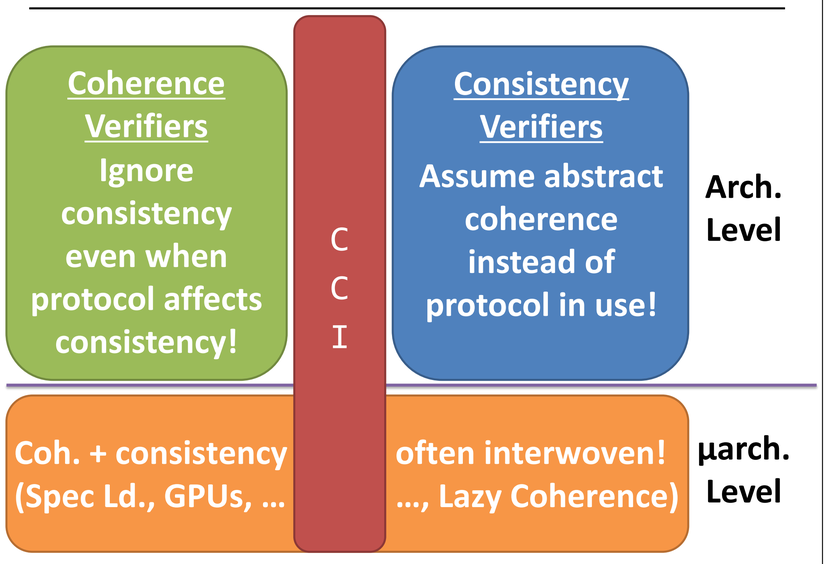
CCICheck: Using μhb Graphs to Verify the Coherence-Consistency Interface

Yatin A. Manerkar, Daniel Lustig, Michael Pellauer (NVIDIA), Margaret Martonosi

PRINCETON UNIVERSITY

Nominated for Best Paper!

The Need for CCI Verification



- **Coherence:** propagate writes to other cores
- **Consistency:** ordering rules for rd./write visibility
- Independent verification of coherence and consistency leaves **verification gap at CCI!**

CCI Mismatch Example ("Peekaboo")

- Prefetch + Livelock Avoidance Mechanism + Inv Before Use = **Consistency Violation!**

`mp` Litmus Test

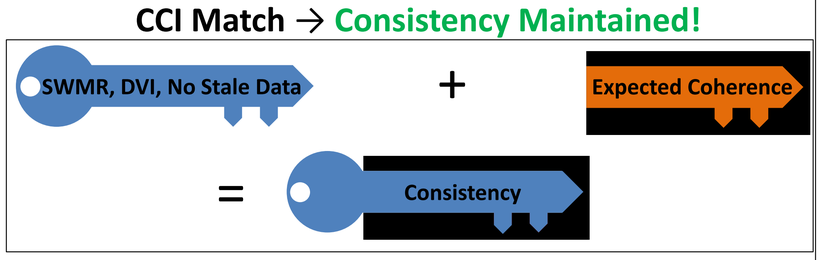
Core 0	Core 1
(i1) St [x] ← 1	(i3) Ld r1 ← [y]
(i2) St [y] ← 1	(i4) Ld r2 ← [x]
Under TSO: Forbid r1=1, r2=0	

Core 0	Core 1
	x: Prefetch miss; issue GetS request
x: Downgrade to S, send [x = 0] to core 1	
x: Issue GetM	
	x: Receive and note Inv, send Inv-Ack (**before data has arrived**)
x: Receive Inv-Ack, perform store x = 1	
y: Perform store y = 1	
	y: Load miss; issue GetS request
y: Downgrade to S, send [y = 1] to core 1	
	y: Perform load y = 1
	x: Load miss; wait for (now stale) data currently in transit
	x: Receive data [x = 0]; perform load x = 0

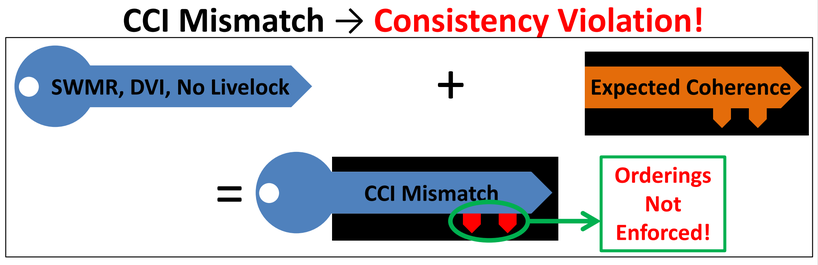
Coherence-Consistency Interface (CCI)

CCI = guarantees that coherence protocol **provides** to rest of microarchitecture + memory ordering guarantees that rest of microarchitecture **expects** from coherence protocol

CCI Match → Consistency Maintained!



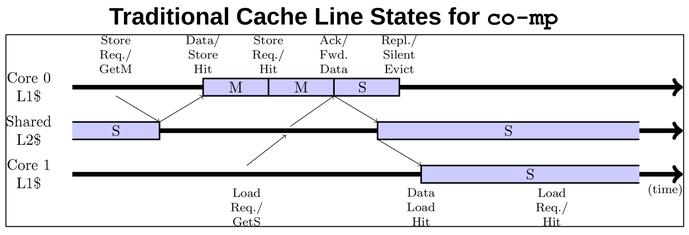
CCI Mismatch → Consistency Violation!



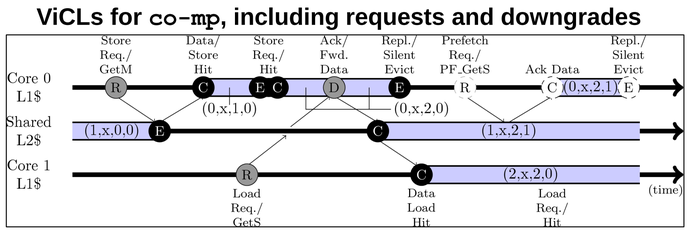
ViCL (Value in Cache Lifetime)

- Models **cache occupancy** and **coh. transitions**
- Formally, a ViCL is a 4-tuple $(cache\_id, address, data\_value, generation\_id)$
- Maps onto period of time (relative to a single cache) over which cache line corresponding to $cache\_id$ and $generation\_id$ holds value $data\_value$ for address $address$.

Traditional Cache Line States for `co-mp`



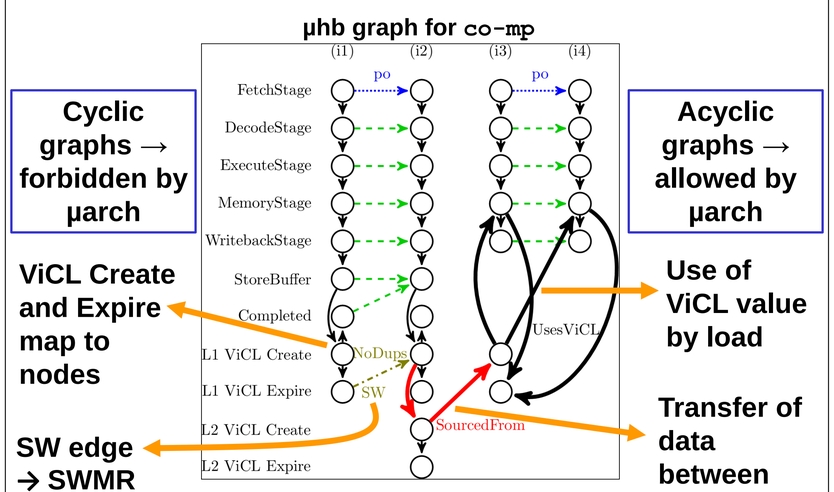
ViCLs for `co-mp`, including requests and downgrades



Microarchitectural happens-before (μhb) graphs with ViCLs

- **Executions** modelled by **μhb graphs**
- **Node** → **microarchitectural event** or **pipeline stage**
- **Edge** → local **happens-before relation** between nodes

μhb graph for `co-mp`



`co-mp` Litmus Test

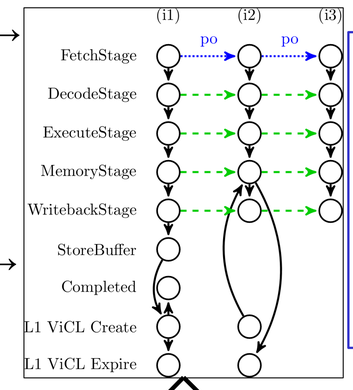
Core 0	Core 1
(i1) St [x] ← 1	(i3) Ld r1 ← [x]
(i2) St [x] ← 2	(i4) Ld r2 ← [x]
In TSO: r1=2, r2=2 Allowed	

`sll` Litmus Test

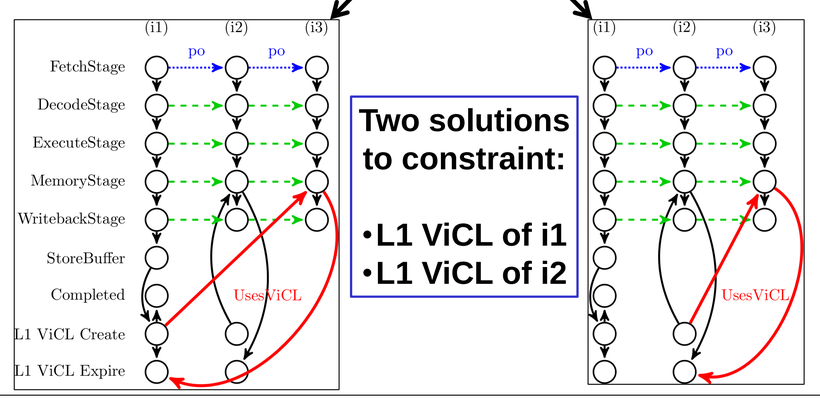
Thread	
(i1)	St [x] ← 1
(i2)	Ld r1 ← [x]
(i3)	Ld r2 ← [x]
Allowed: r1=1, r2=1	

Constraint-Based Enumeration

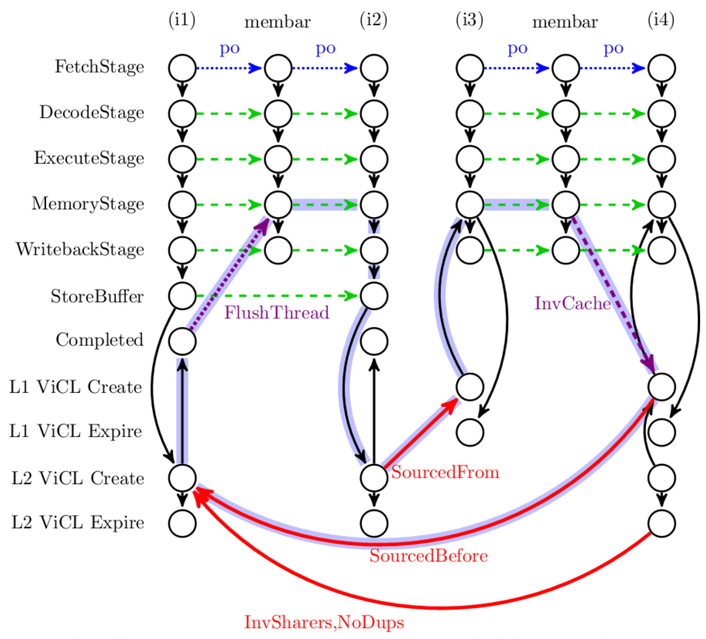
- **Multiple solns** → each further enumerated independently
- **No solns** → **invalid** scenario
- **Cyclic graphs** → **pruned** (can't become acyclic)



Constraint: **Load i3 requires source L1 ViCL with same address and data**

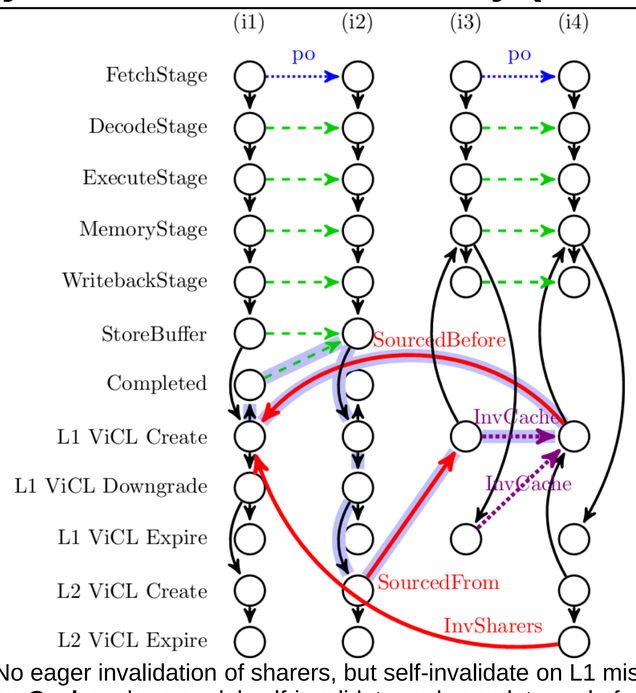


Partial Incoherence (GPU) Case Study



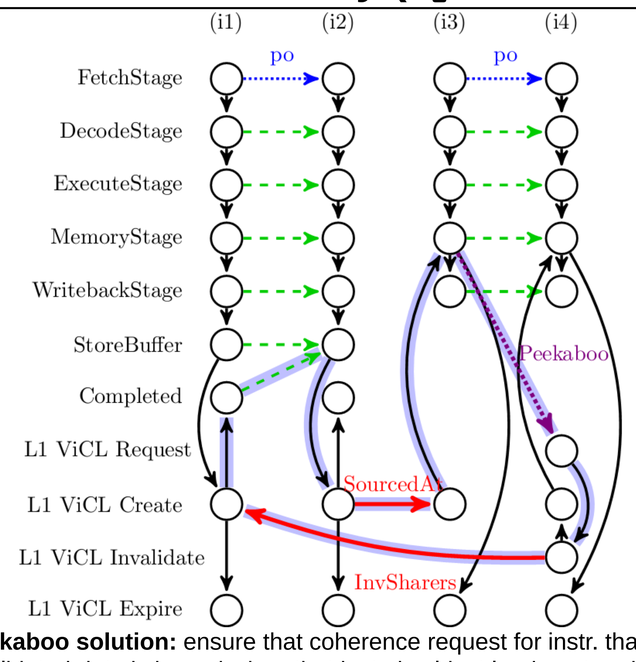
- Counterintuitive outcome for mp with **`membar`** [Alglave et al. ASPLOS15]
- If load-load fence does not ensure **InvCache** ordering, no cycle

Lazy Coherence Case Study (TSO-CC)



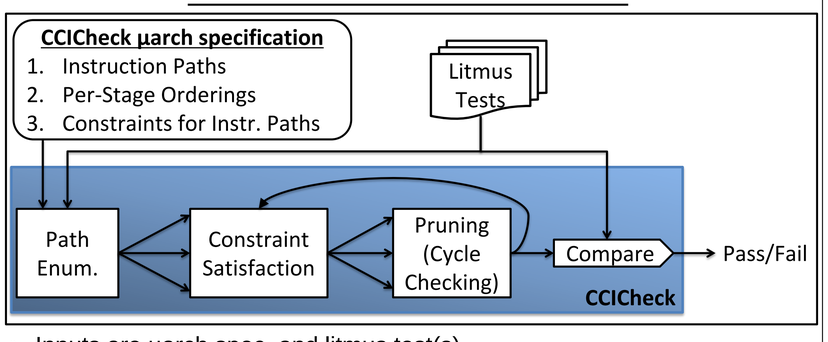
- No eager invalidation of sharers, but self-invalidate on L1 miss
- **InvCache** edges model self-invalidate and complete cycle for **`mp`**

Peekaboo Case Study (`mp` Litmus Test)



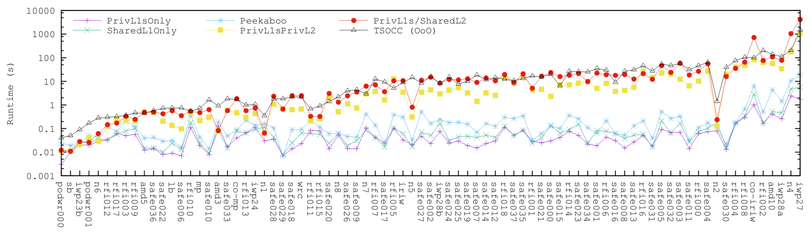
- **Peekaboo solution:** ensure that coherence request for instr. that uses invalidated data is issued when that instr. is oldest load or store in program order [Sorin et al. primer]

CCICheck Toolflow



- Inputs are μarch spec. and litmus test(s)
- 2 high-level enumeration steps: Path Enumeration & Constraint Satisfaction
- Intelligent pruning and unsatisfiable constraint detection keep runtimes scalable

Results



- CCICheck was run on a variety of microarchitectures and coherence protocols across 85 litmus tests
- Geomean test case execution time < 10 seconds on **all** architectures
- Subsequent research used SMT solver-based methods to run most tests in **just a few seconds**! [ASPLOS 2016]

Conclusions

- CCI verification is critical to the correct operation of large or complex parallel systems
- CCICheck's **static CCI-aware microarchitectural consistency verification** is a first step in this direction
- CCICheck uses **μhb graphs** and **exhaustive enumeration** of all possible litmus test executions to verify a microarchitecture
- The **Value in Cache Lifetime (ViCL) abstraction**, **constraint-based enumeration**, and **intelligent pruning** allow comprehensive yet tractable analysis
- CCICheck can handle **partial incoherence**, **lazy coherence**, and a variety of **coherence protocol transitions**
- CCICheck is **open-source** and publicly available at github.com/ymanerka/ccicheck