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

Yatin A. Manerkar, Daniel Lustig,
Michael Pellauer*, and Margaret Martonosi

Princeton University *NVIDIA

MICRO-48

Coherence and Consistency

At a high level:

- **Coherence Protocols:** Propagation of writes to other cores
- **Consistency Models:** Ordering rules for visibility of reads and writes

Coherence and Consistency

Coherence Verifiers

Consistency Verifiers

Arch. Level

Coherence and Consistency

Coherence Verifiers

Consistency Verifiers

Arch. Level

Coherence and consistency often interwoven

µarch. Level

Coherence and Consistency

Coherence Verifiers

Ignore consistency even when protocol affects consistency!

Consistency Verifiers

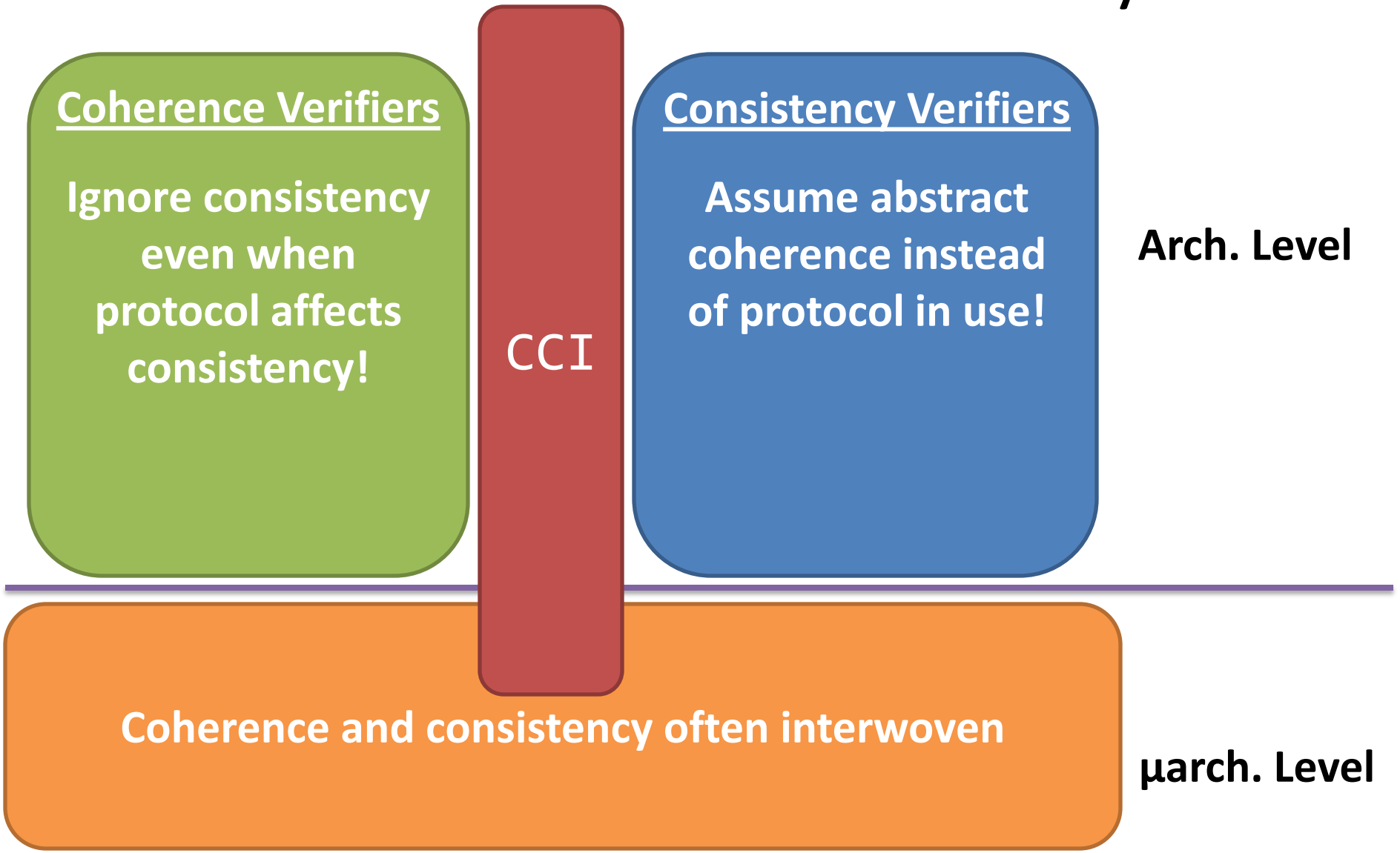
Assume abstract coherence instead of protocol in use!

Arch. Level

Coherence and consistency often interwoven

µarch. Level

Coherence and Consistency



Motivating Example – “Peekaboo”

Motivating Example – "Peekaboo"

1. Invalidation before use
   - Repeated inv before use → livelock [Kubiatowicz et al. ASPLOS 1992]

Motivating Example – "Peekaboo"

1. Invalidation before use
   - Repeated inv before use → livelock [Kubiatowicz et al. ASPLOS 1992]
2. **Livelock avoidance:** allow destination core to perform **one** operation on data when it arrives, **even if already invalidated** [Sorin et al. Primer]
   - Does **not** break coherence
   - Sometimes **intentionally** returns stale data

Motivating Example – "Peekaboo"

1. Invalidation before use
   - Repeated inv before use → livelock [Kubiatowicz et al. ASPLOS 1992]
2. **Livelock avoidance:** allow destination core to perform **one** operation on data when it arrives, **even if already invalidated** [Sorin et al. Primer]
   - Does **not** break coherence
   - Sometimes **intentionally** returns stale data
3. Prefetching

Motivating Example – "Peekaboo"

1. Invalidation before use
   - Repeated inv before use → livelock [Kubiatowicz et al. ASPLOS 1992]

Individual Opt. → **No violation**

Combination of Opts. → **Violation!**

   - Does **not** break coherence
   - Sometimes **intentionally** returns stale data
3. Prefetching

Motivating Example – "Peekaboo"

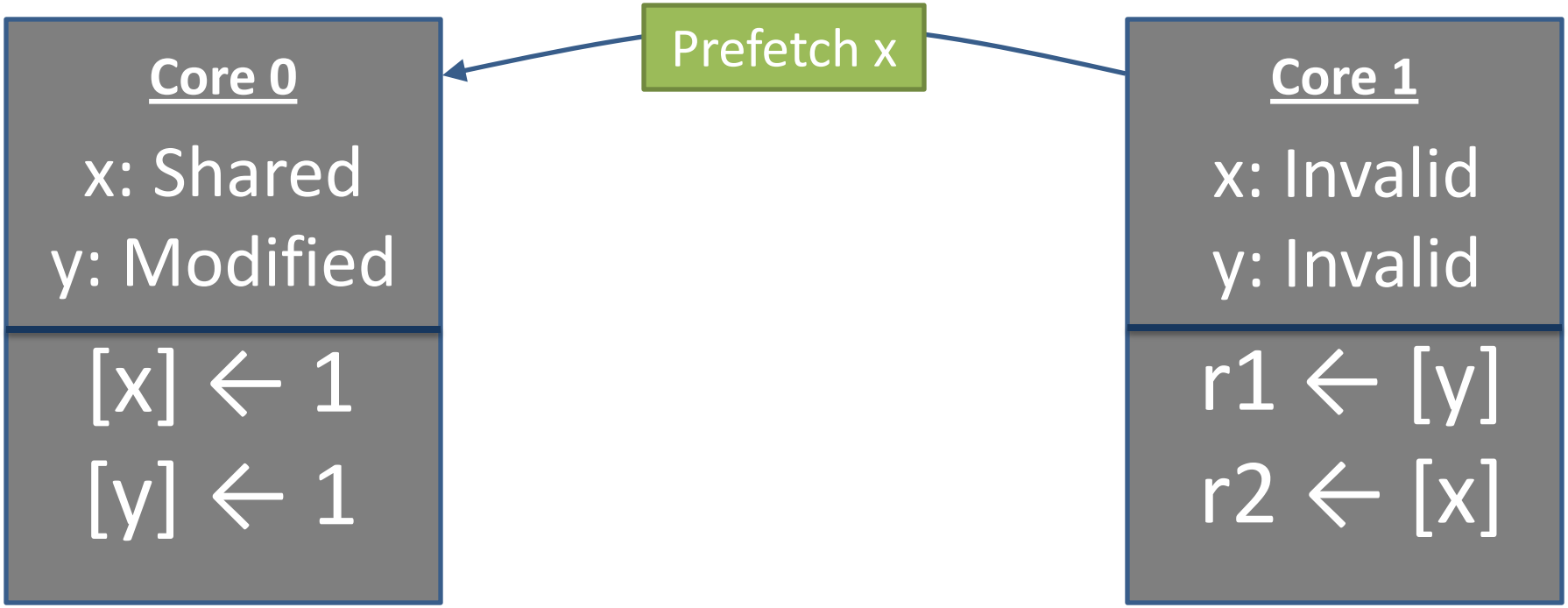
- Consider `mp` with the livelock-avoidance mechanism:

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

Motivating Example – "Peekaboo"

- Consider **mp** with the livelock-avoidance mechanism:

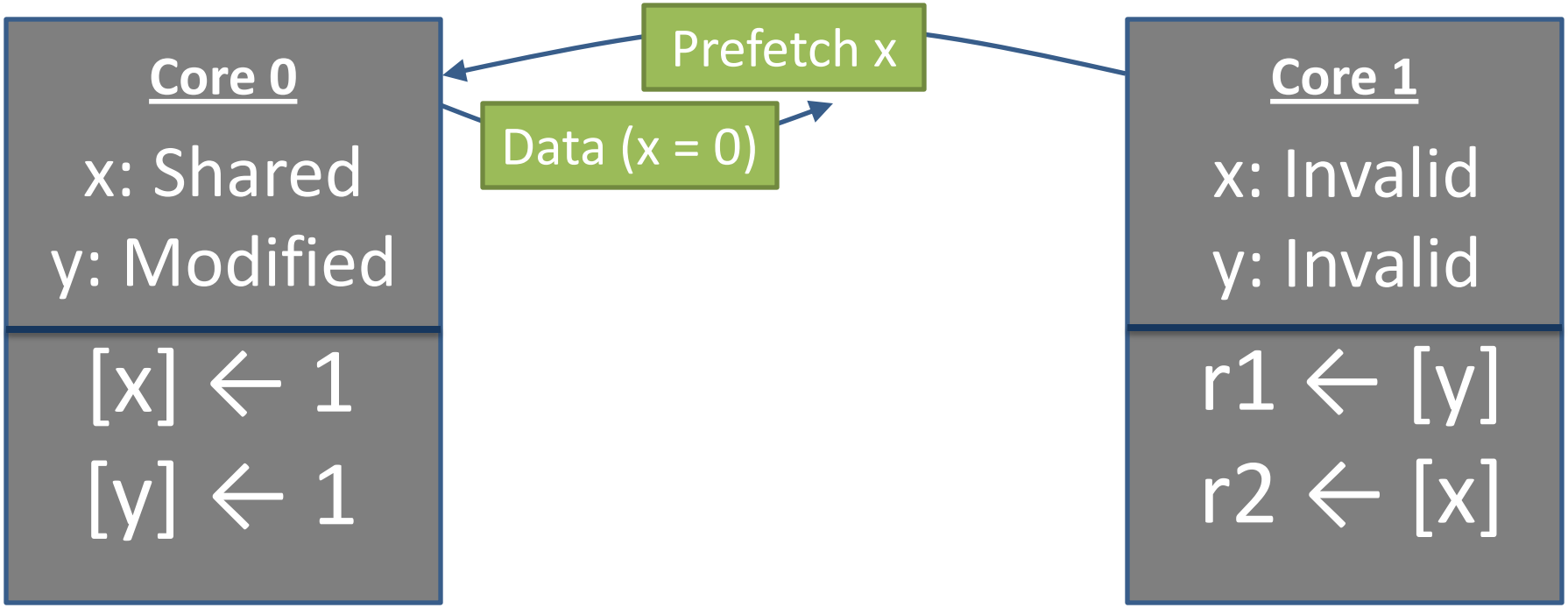
Core 0	Core 1
(i1) St [x] ← 1 (i2) St [y] ← 1	(i3) Ld r1 ← [y] (i4) Ld r2 ← [x]
Under TSO: Forbid r1=1, r2=0	



Motivating Example – "Peekaboo"

- Consider **mp** with the livelock-avoidance mechanism:

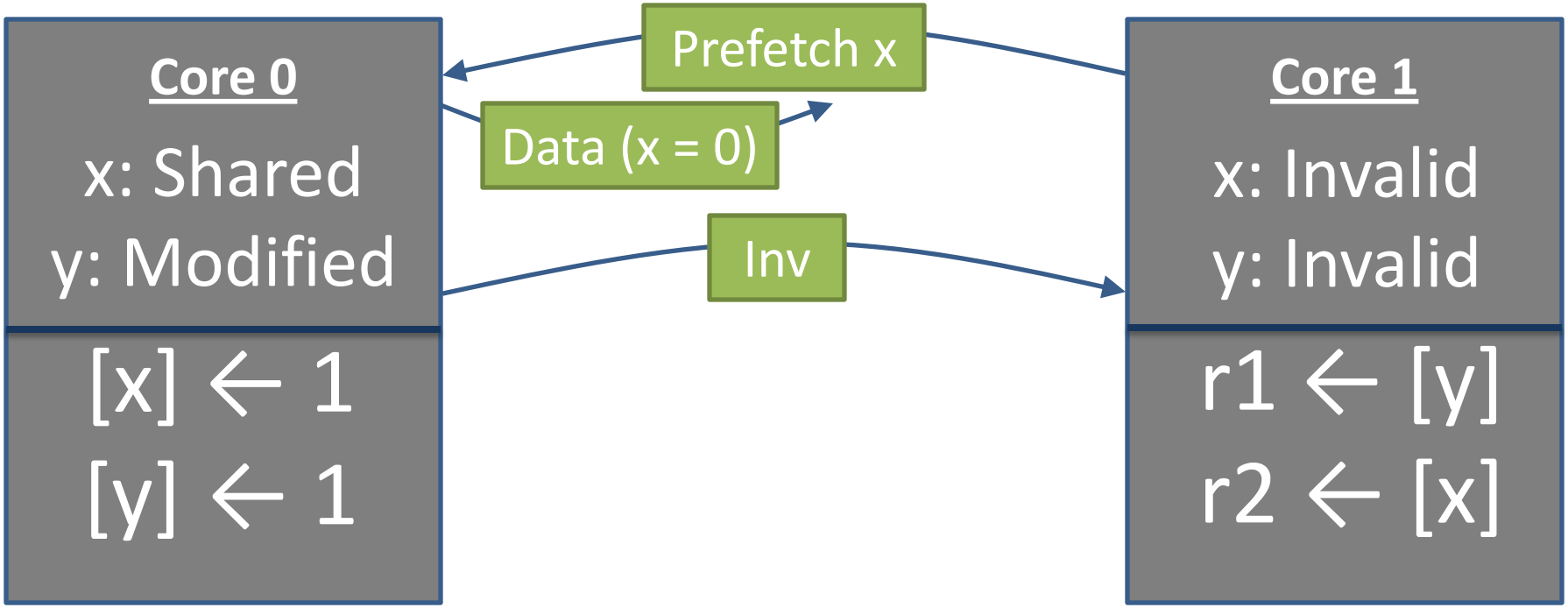
Core 0	Core 1
(i1) St [x] ← 1 (i2) St [y] ← 1	(i3) Ld r1 ← [y] (i4) Ld r2 ← [x]
Under TSO: Forbid r1=1, r2=0	



Motivating Example – "Peekaboo"

- Consider **mp** with the livelock-avoidance mechanism:

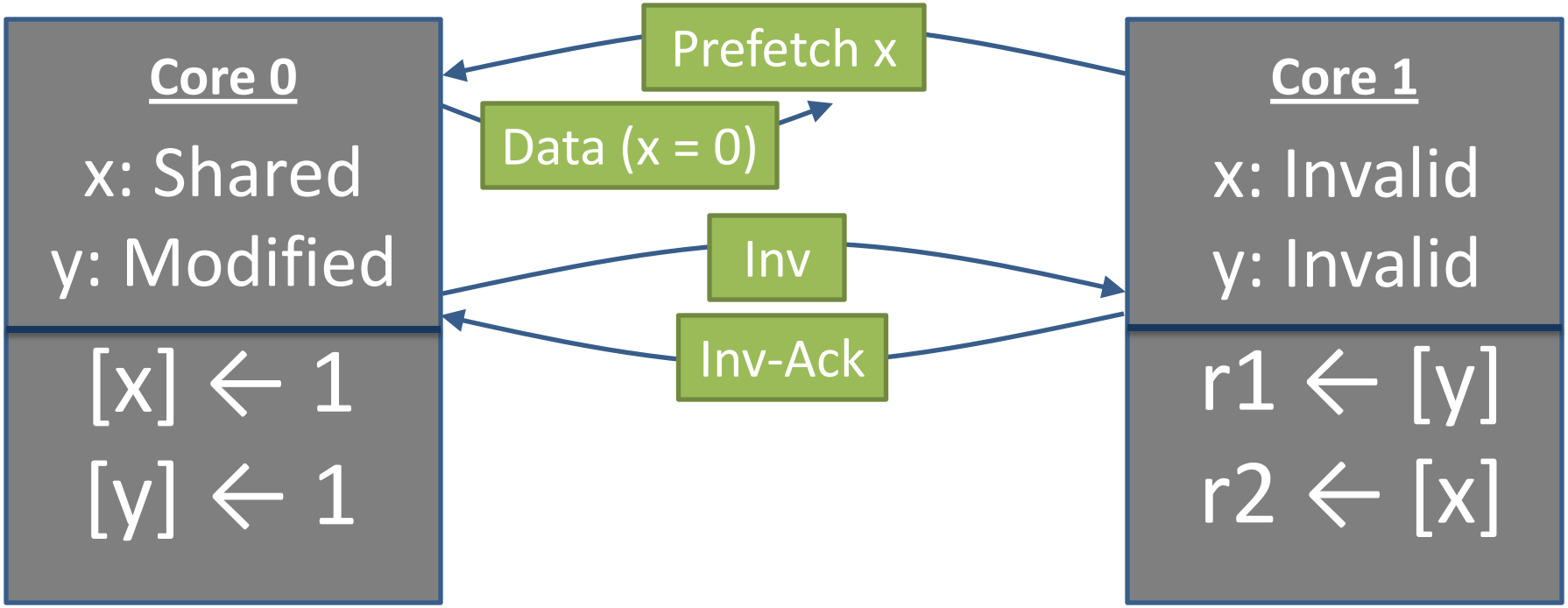
Core 0	Core 1
(i1) St [x] ← 1 (i2) St [y] ← 1	(i3) Ld r1 ← [y] (i4) Ld r2 ← [x]
Under TSO: Forbid r1=1, r2=0	



Motivating Example – "Peekaboo"

- Consider **mp** with the livelock-avoidance mechanism:

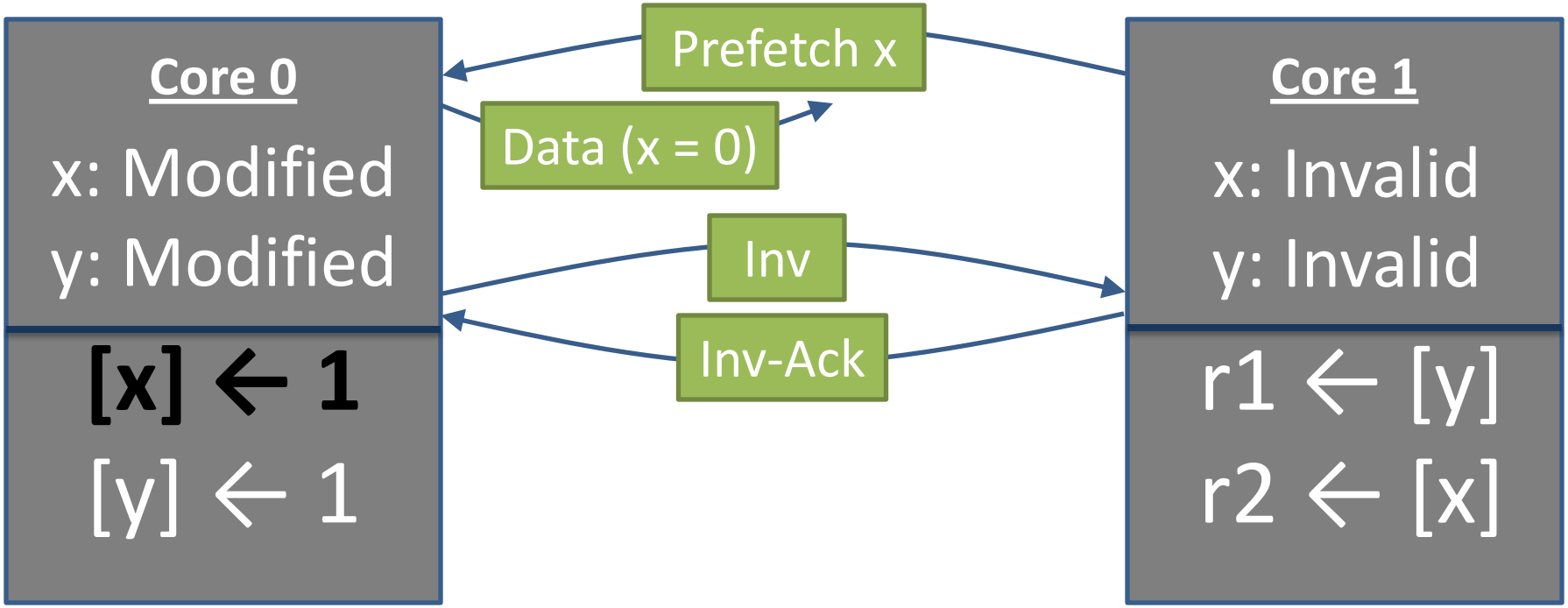
Core 0	Core 1
(i1) St [x] ← 1 (i2) St [y] ← 1	(i3) Ld r1 ← [y] (i4) Ld r2 ← [x]
Under TSO: Forbid r1=1, r2=0	



Motivating Example – "Peekaboo"

- Consider **mp** with the livelock-avoidance mechanism:

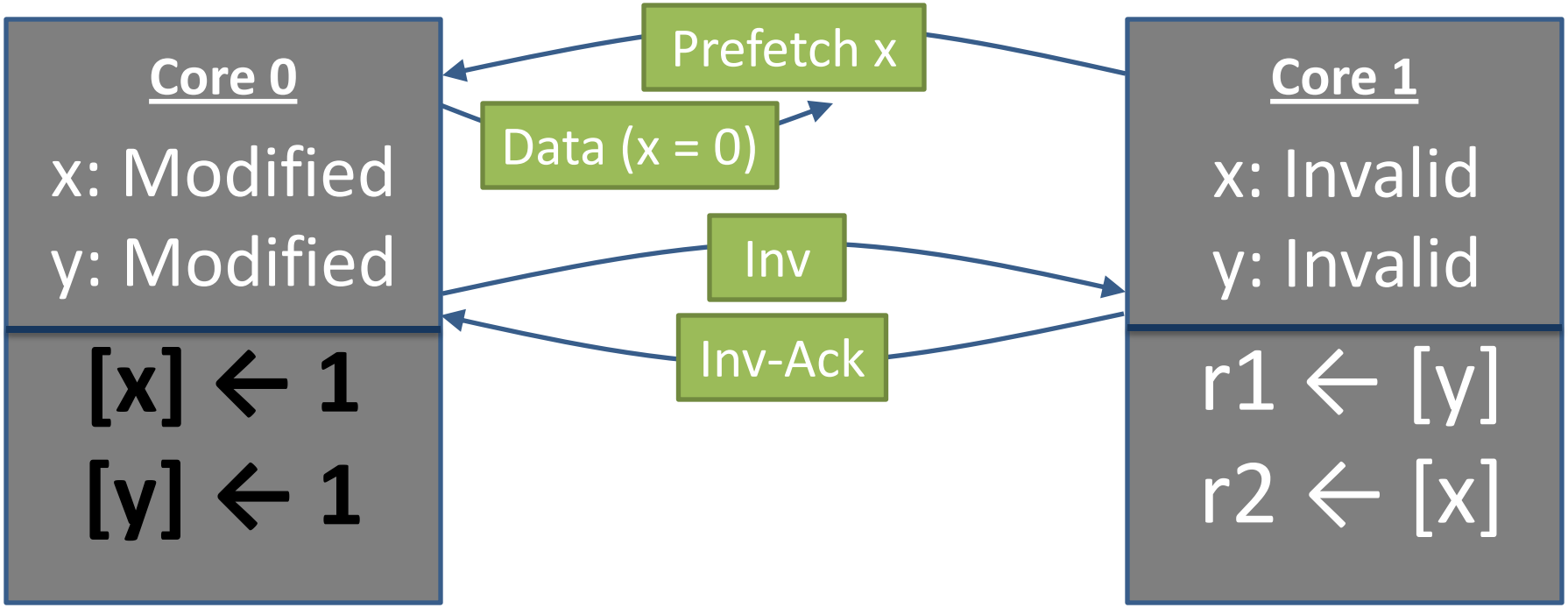
Core 0	Core 1
(i1) St [x] ← 1 (i2) St [y] ← 1	(i3) Ld r1 ← [y] (i4) Ld r2 ← [x]
Under TSO: Forbid r1=1, r2=0	



Motivating Example – "Peekaboo"

- Consider **mp** with the livelock-avoidance mechanism:

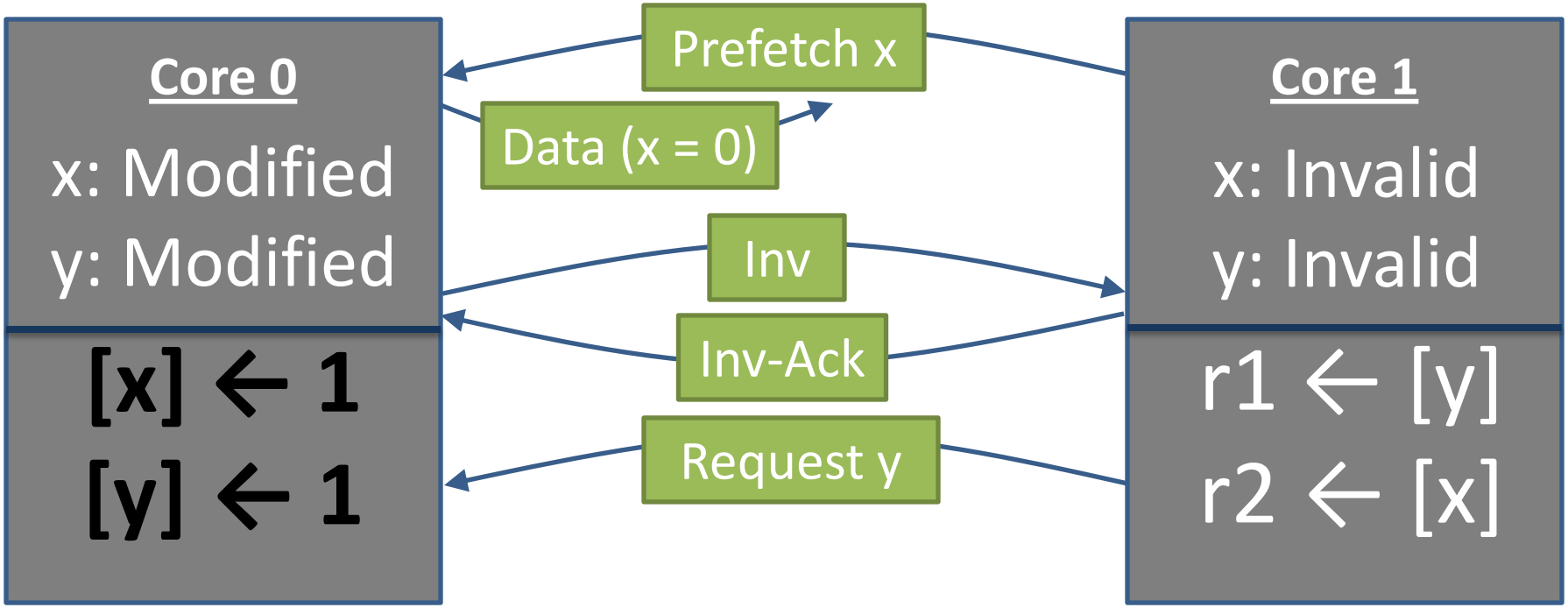
Core 0	Core 1
(i1) St [x] ← 1 (i2) St [y] ← 1	(i3) Ld r1 ← [y] (i4) Ld r2 ← [x]
Under TSO: Forbid r1=1, r2=0	



Motivating Example – "Peekaboo"

- Consider **mp** with the livelock-avoidance mechanism:

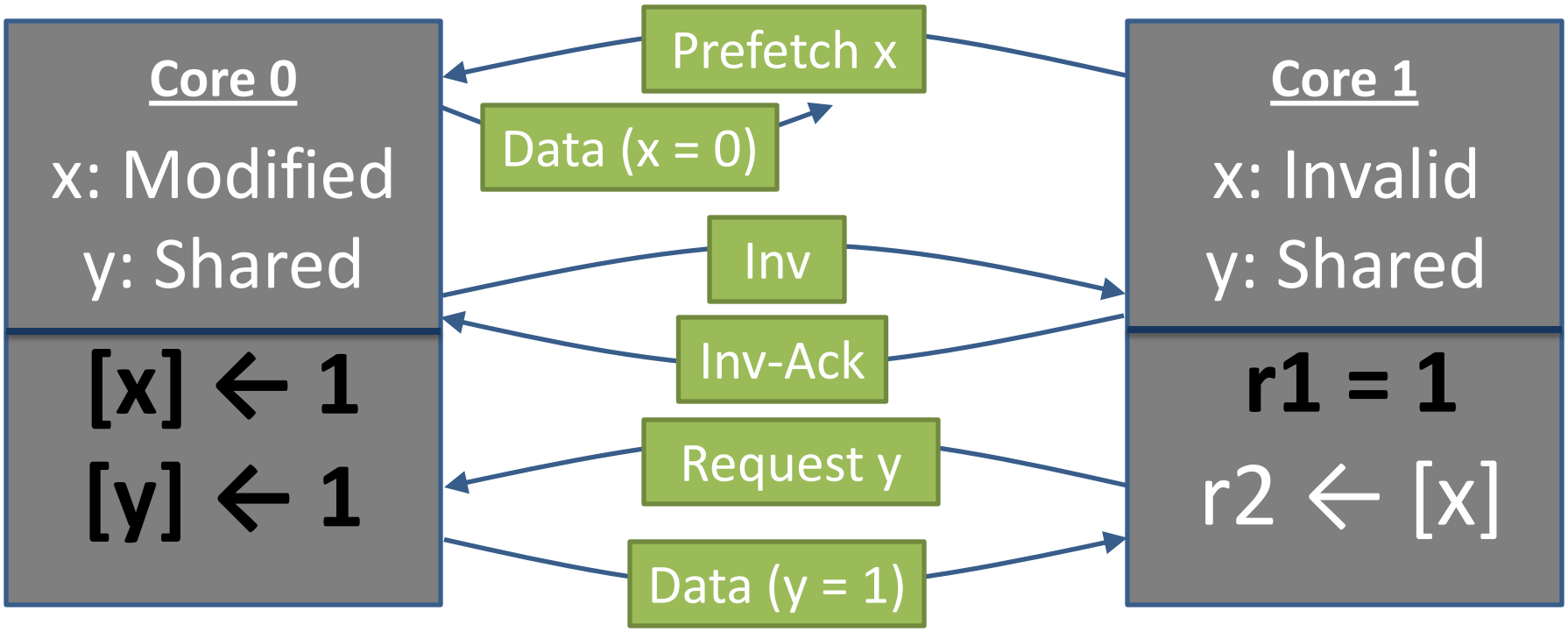
Core 0	Core 1
(i1) St [x] ← 1 (i2) St [y] ← 1	(i3) Ld r1 ← [y] (i4) Ld r2 ← [x]
Under TSO: Forbid r1=1, r2=0	



Motivating Example – "Peekaboo"

- Consider **mp** with the livelock-avoidance mechanism:

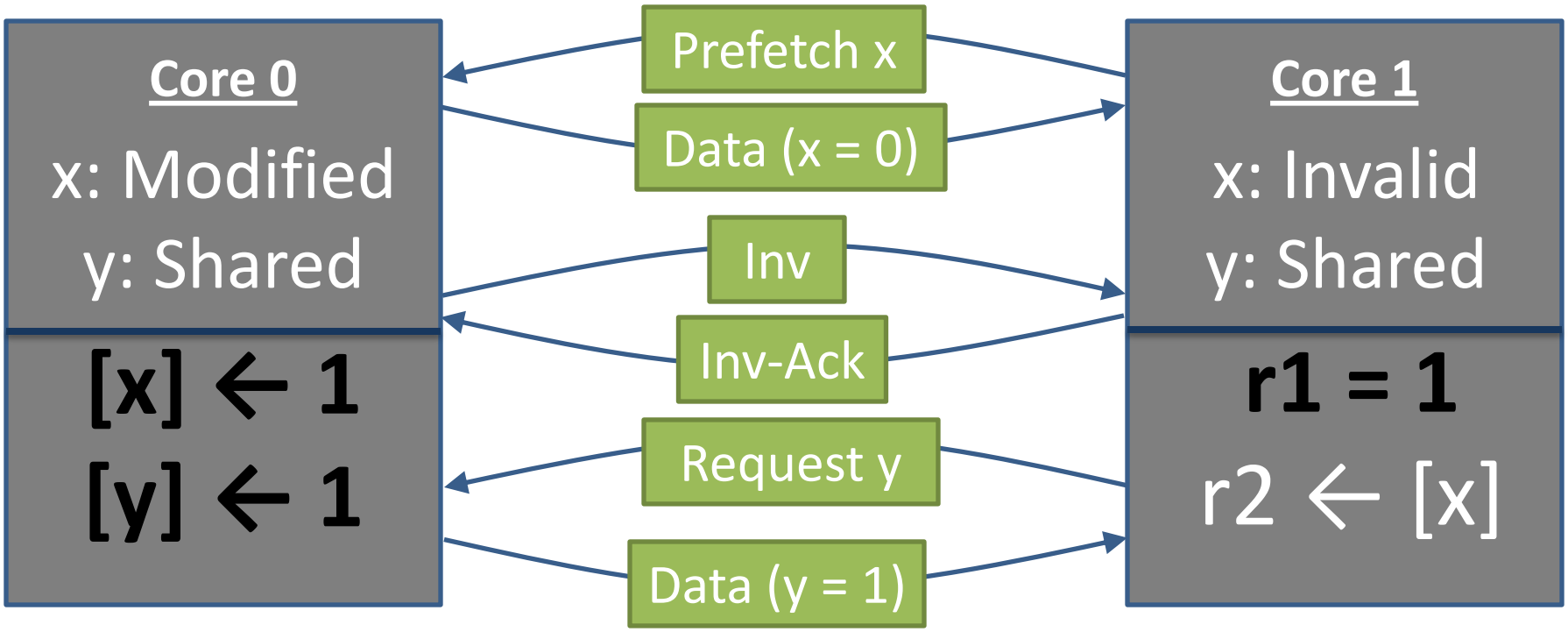
Core 0	Core 1
(i1) St [x] ← 1 (i2) St [y] ← 1	(i3) Ld r1 ← [y] (i4) Ld r2 ← [x]
Under TSO: Forbid r1=1, r2=0	



Motivating Example – "Peekaboo"

- Consider **mp** with the livelock-avoidance mechanism:

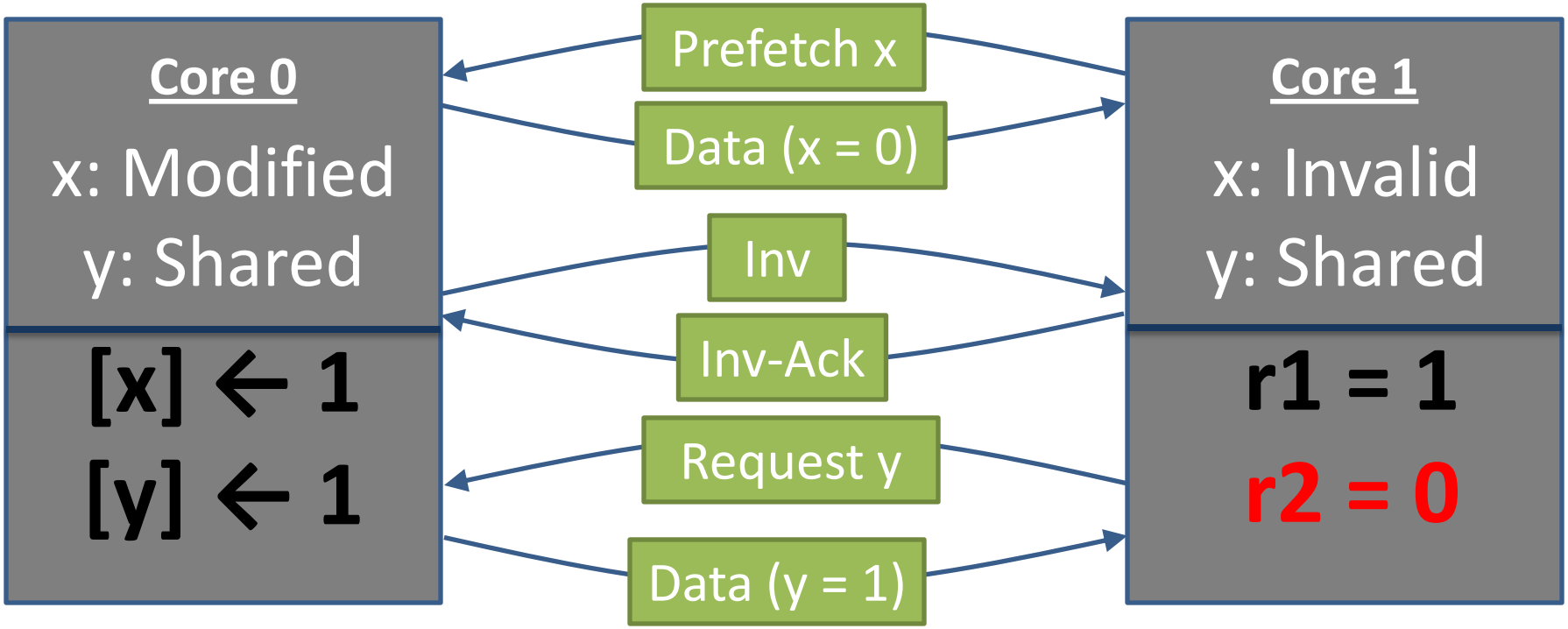
Core 0	Core 1
(i1) St [x] ← 1 (i2) St [y] ← 1	(i3) Ld r1 ← [y] (i4) Ld r2 ← [x]
Under TSO: Forbid r1=1, r2=0	



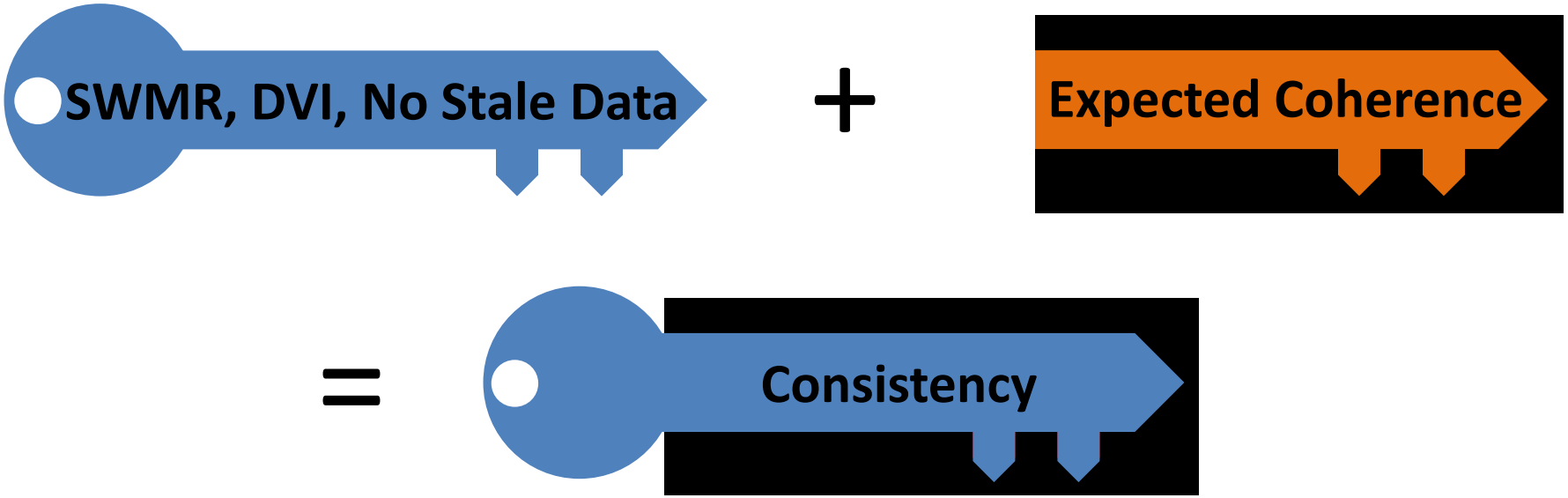
Motivating Example – "Peekaboo"

- Consider **mp** with the livelock-avoidance mechanism:

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

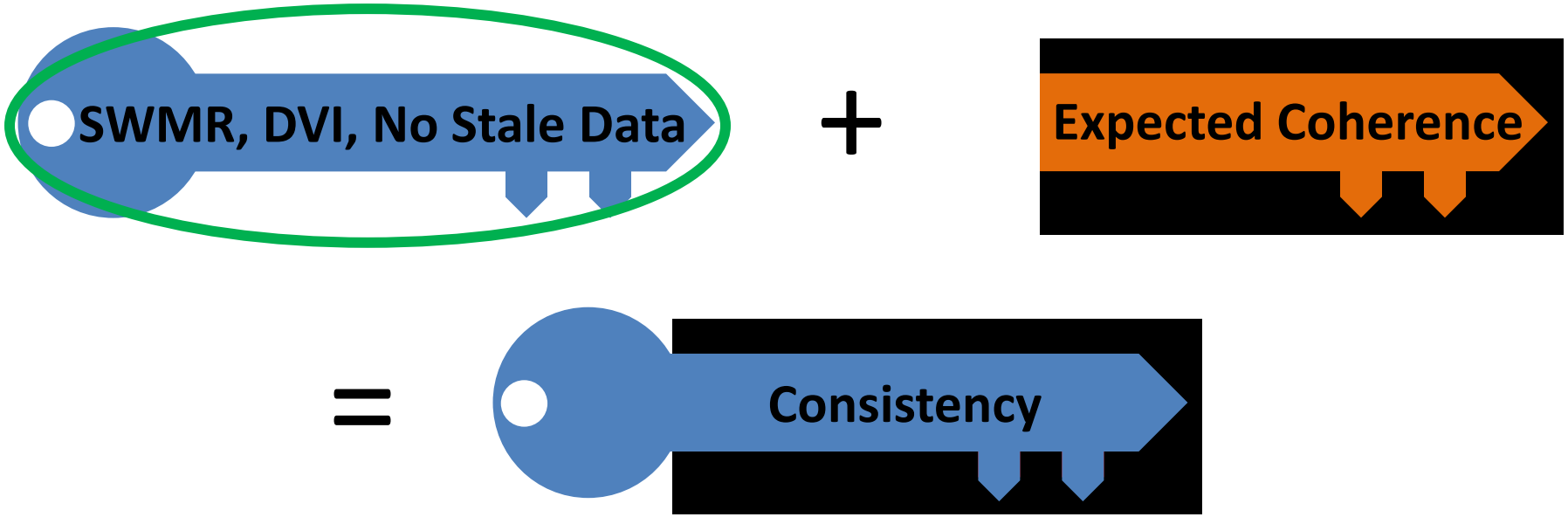


The Coherence-Consistency Interface (CCI)



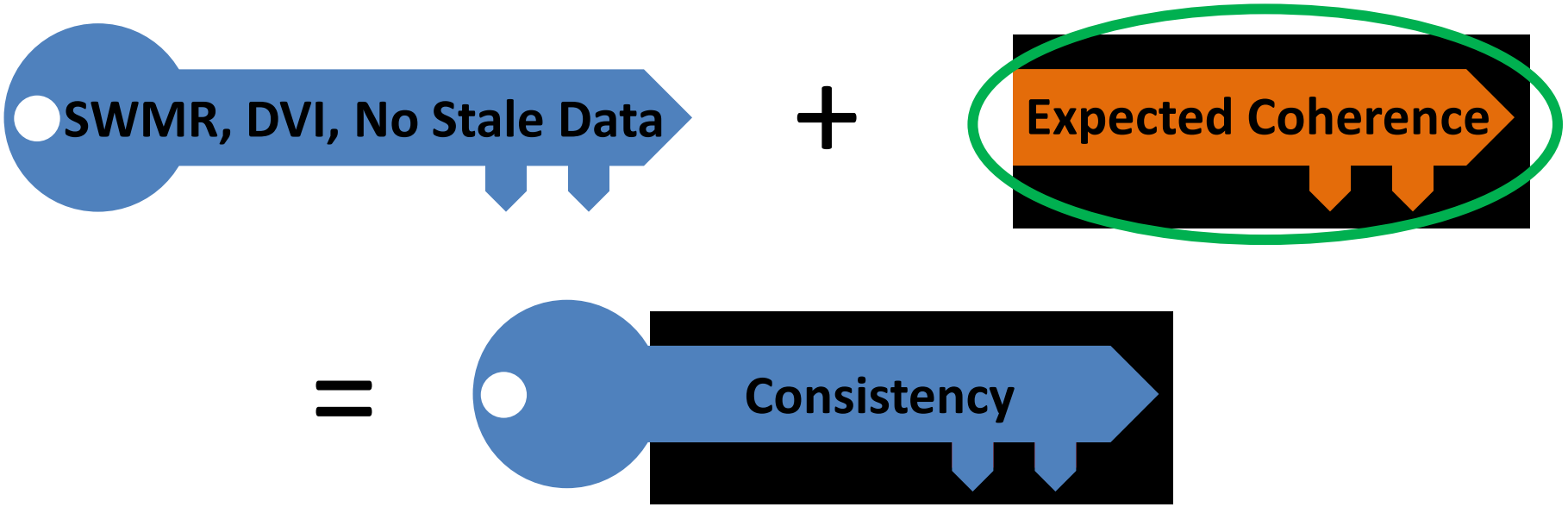
- CCI = guarantees that coherence protocol **provides** to rest of microarchitecture + memory ordering guarantees that rest of microarch. **expects** from coherence protocol

The Coherence-Consistency Interface (CCI)



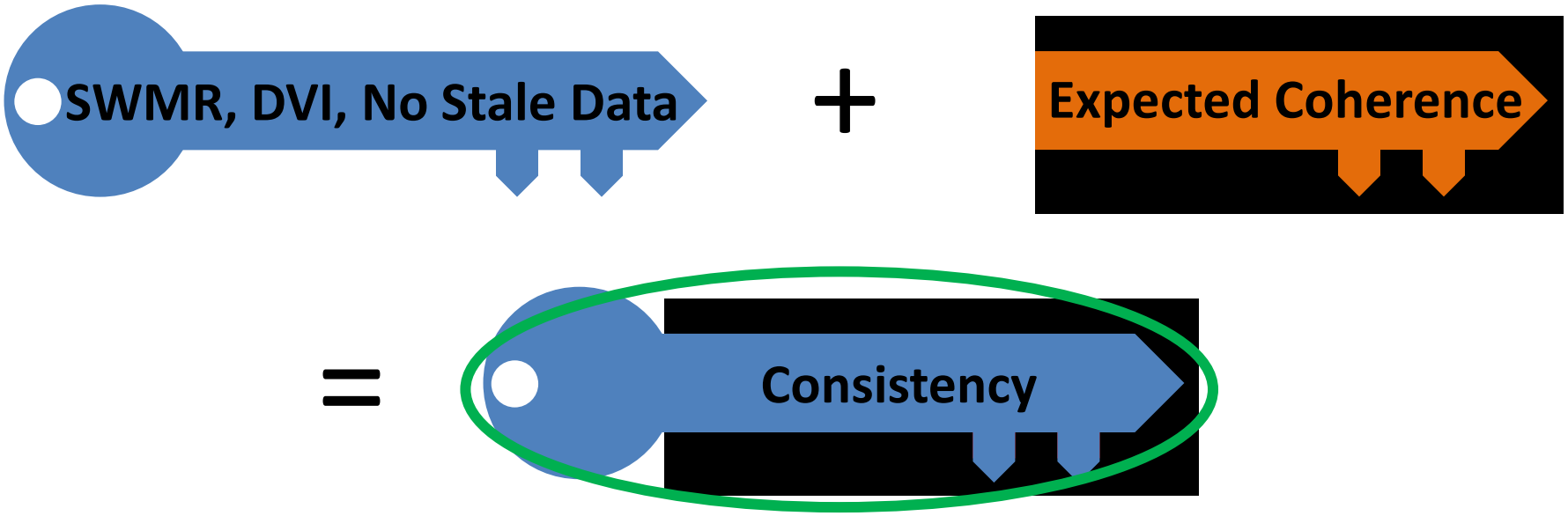
- CCI = guarantees that coherence protocol **provides** to rest of microarchitecture + memory ordering guarantees that rest of microarch. **expects** from coherence protocol

The Coherence-Consistency Interface (CCI)



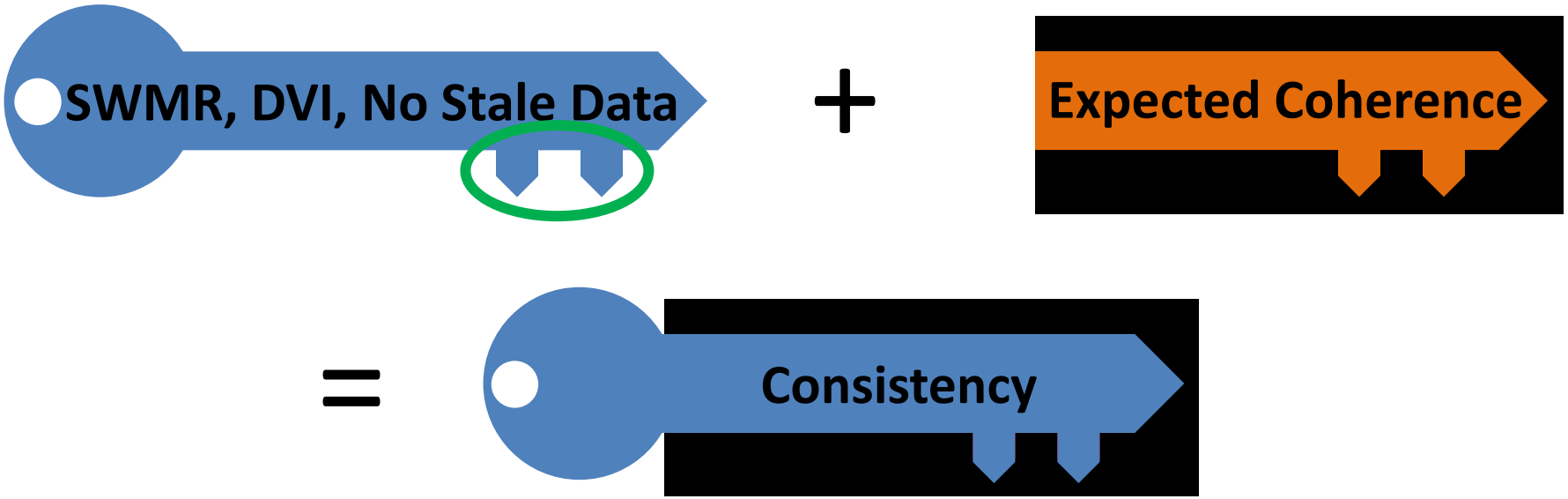
- CCI = guarantees that coherence protocol **provides** to rest of microarchitecture + memory ordering guarantees that rest of microarch. **expects** from coherence protocol

The Coherence-Consistency Interface (CCI)



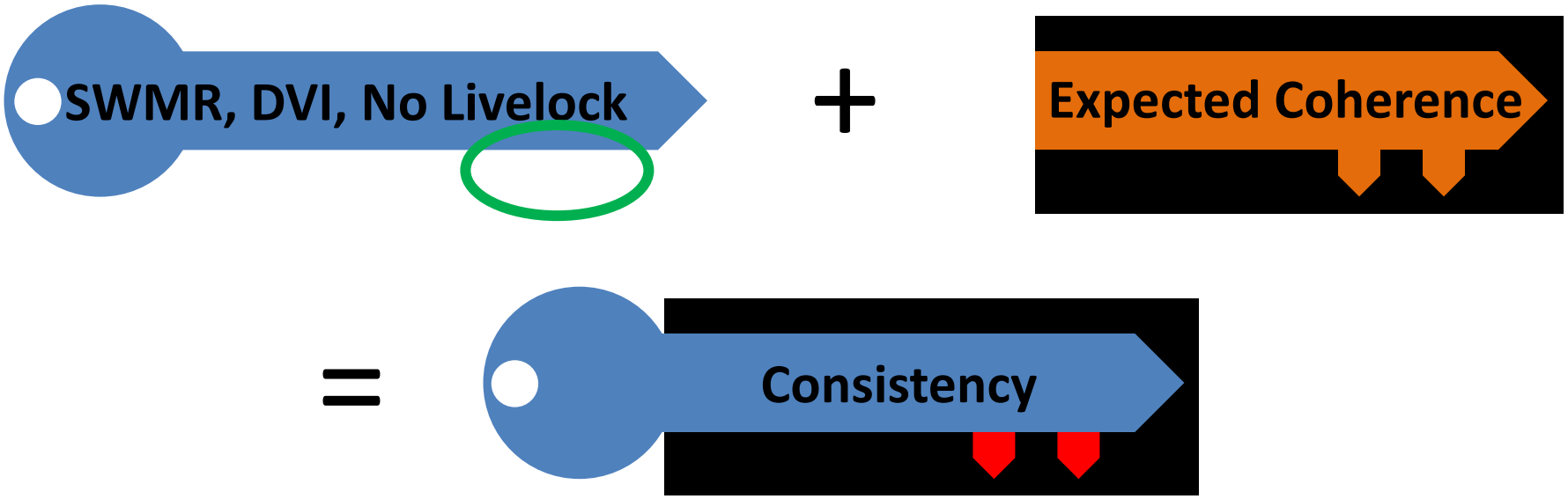
- CCI = guarantees that coherence protocol **provides** to rest of microarchitecture + memory ordering guarantees that rest of microarch. **expects** from coherence protocol

The Coherence-Consistency Interface (CCI)



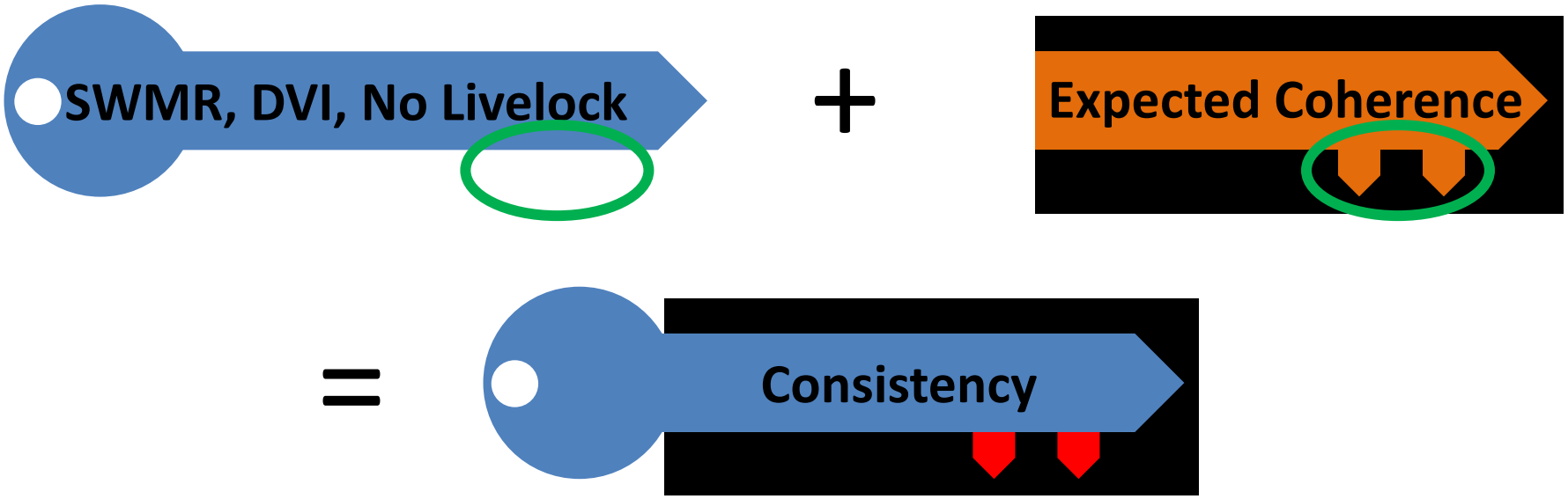
- CCI = guarantees that coherence protocol **provides** to rest of microarchitecture + memory ordering guarantees that rest of microarch. **expects** from coherence protocol

The Coherence-Consistency Interface (CCI)



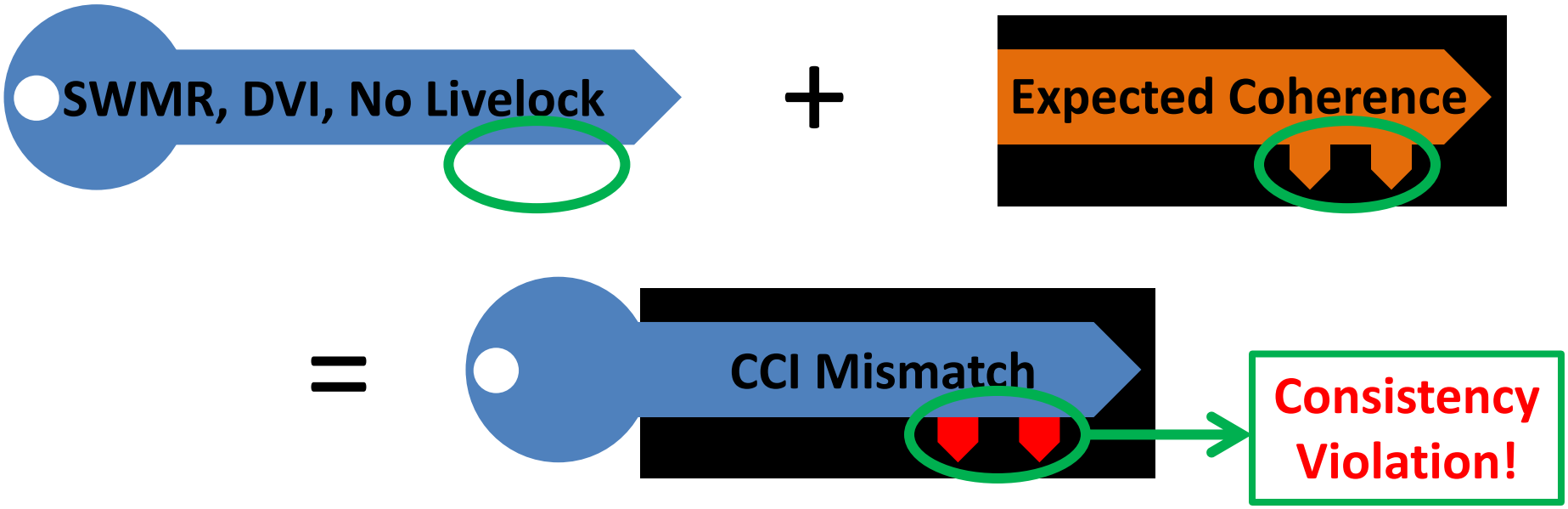
- CCI = guarantees that coherence protocol **provides** to rest of microarchitecture + memory ordering guarantees that rest of microarch. **expects** from coherence protocol

The Coherence-Consistency Interface (CCI)



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

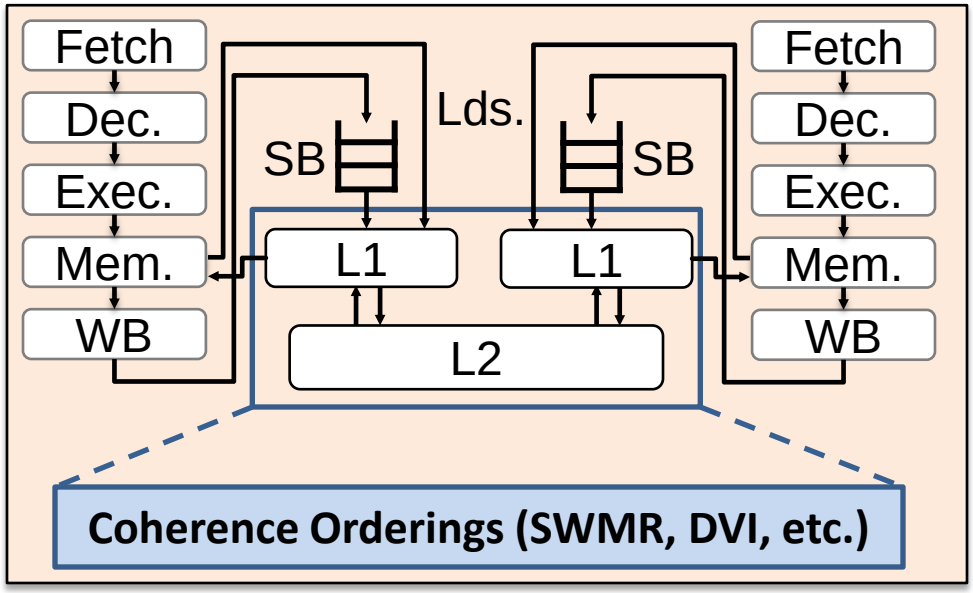
The Coherence-Consistency Interface (CCI)



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

Our Work: CCICheck

Static CCI-aware consistency verification



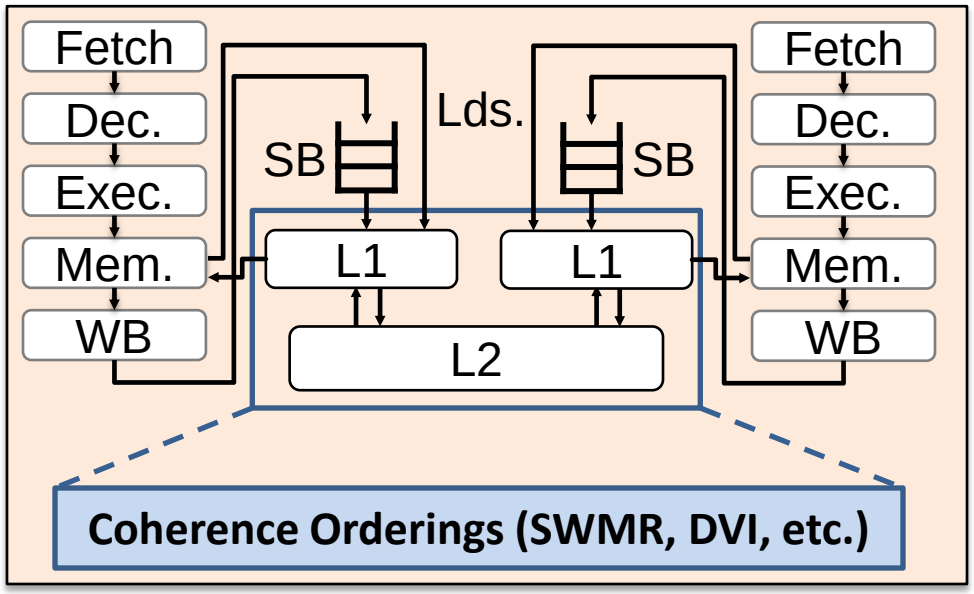
Microarch spec

Litmus Test

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

Our Work: CCICheck

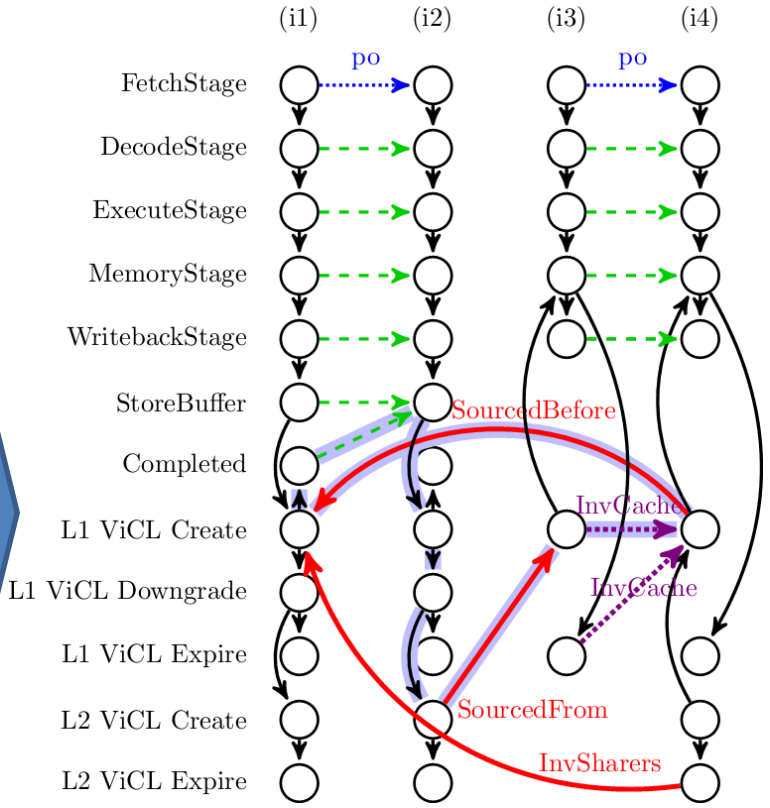
Static CCI-aware consistency verification



Microarch spec

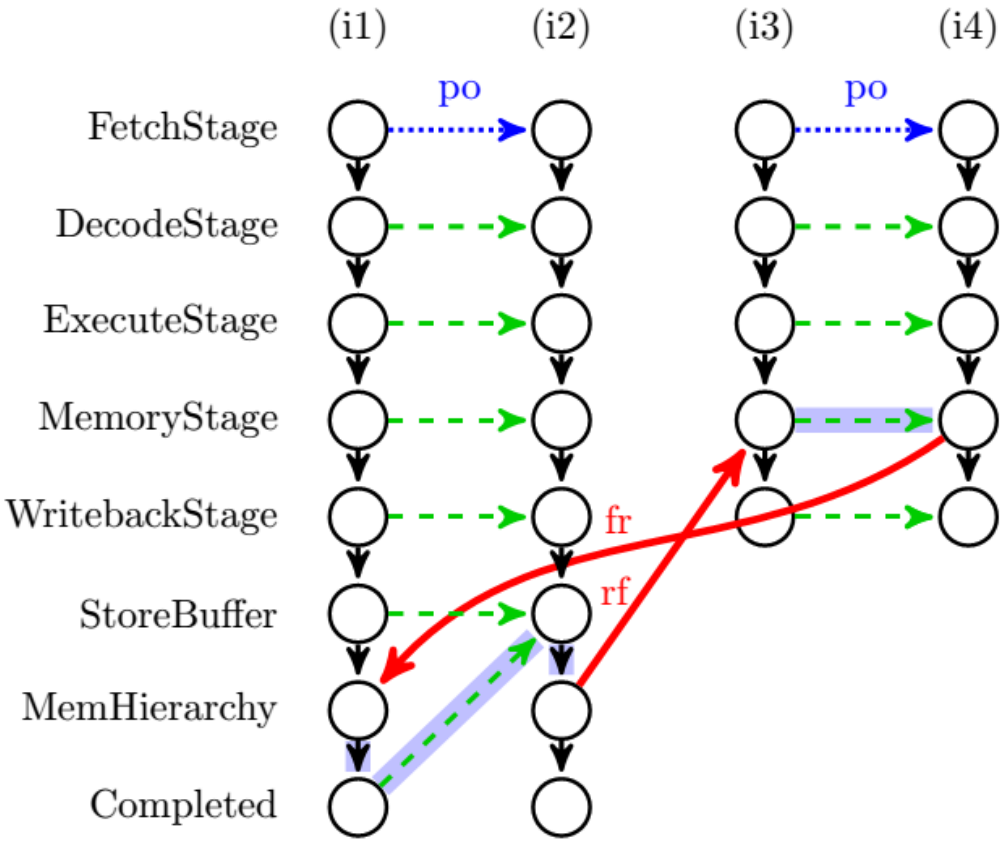
Litmus Test

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	



Microarchitectural happens-before (μhb) graph

Background: PipeCheck



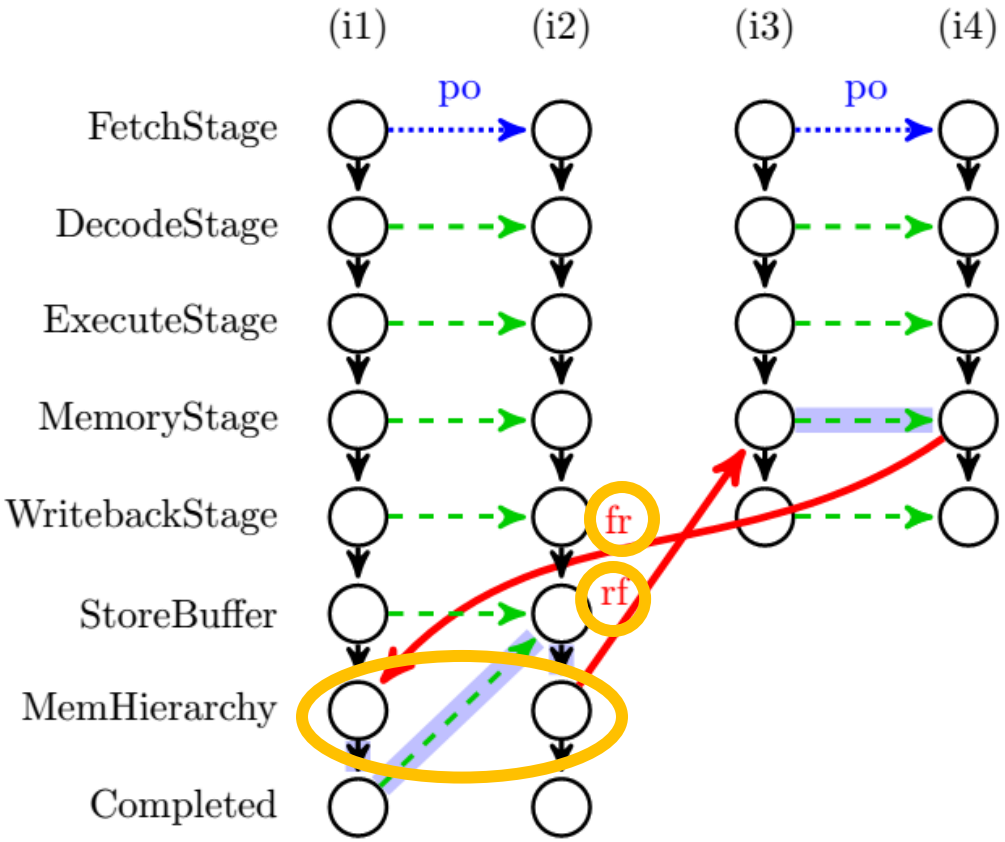
Litmus Test **mp**

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	

- **Exhaustive enumeration** of executions using μhb graphs
- **Cyclic graph →forbidden by μarch**
- **Acyclic graph →allowed by μarch**

[Lustig et al. MICRO-47]

Background: PipeCheck



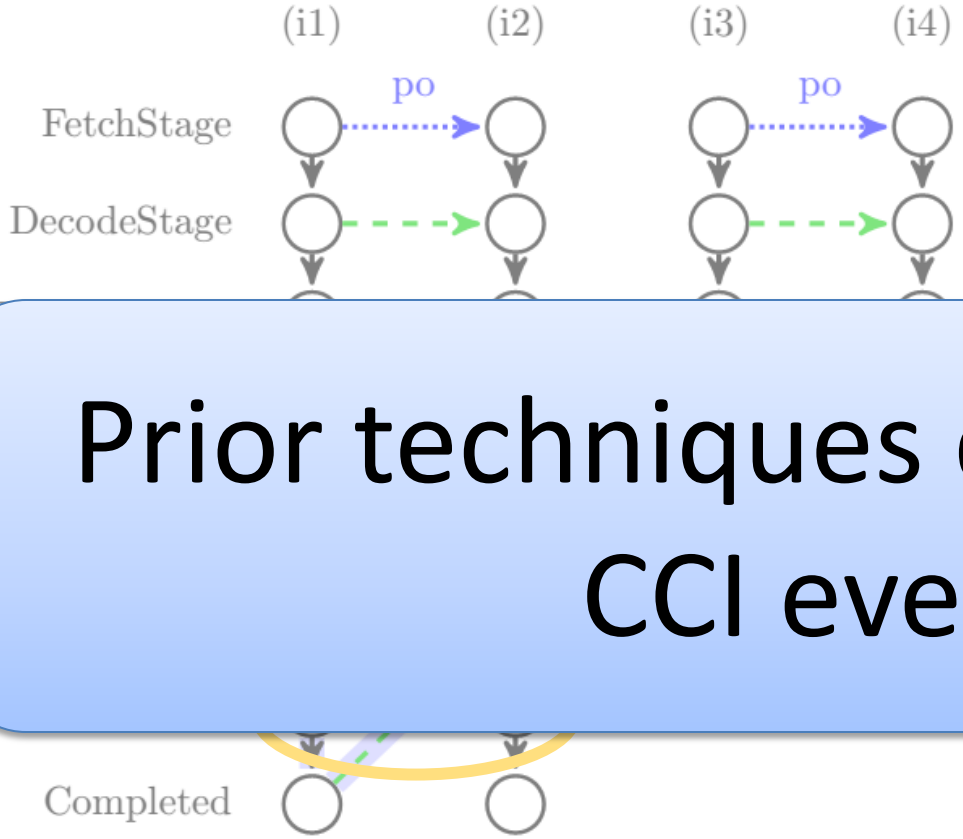
Litmus Test `mp`

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

- **Exhaustive enumeration** of executions using μhb graphs
- **Cyclic graph →forbidden by μarch**
- **Acyclic graph →allowed by μarch**

[Lustig et al. MICRO-47]

Background: PipeCheck



Litmus Test `mp`

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	

- **Exhaustive enumeration** of executions using

Prior techniques cannot model CCI events!

- **Acyclic graph →allowed by μarch**

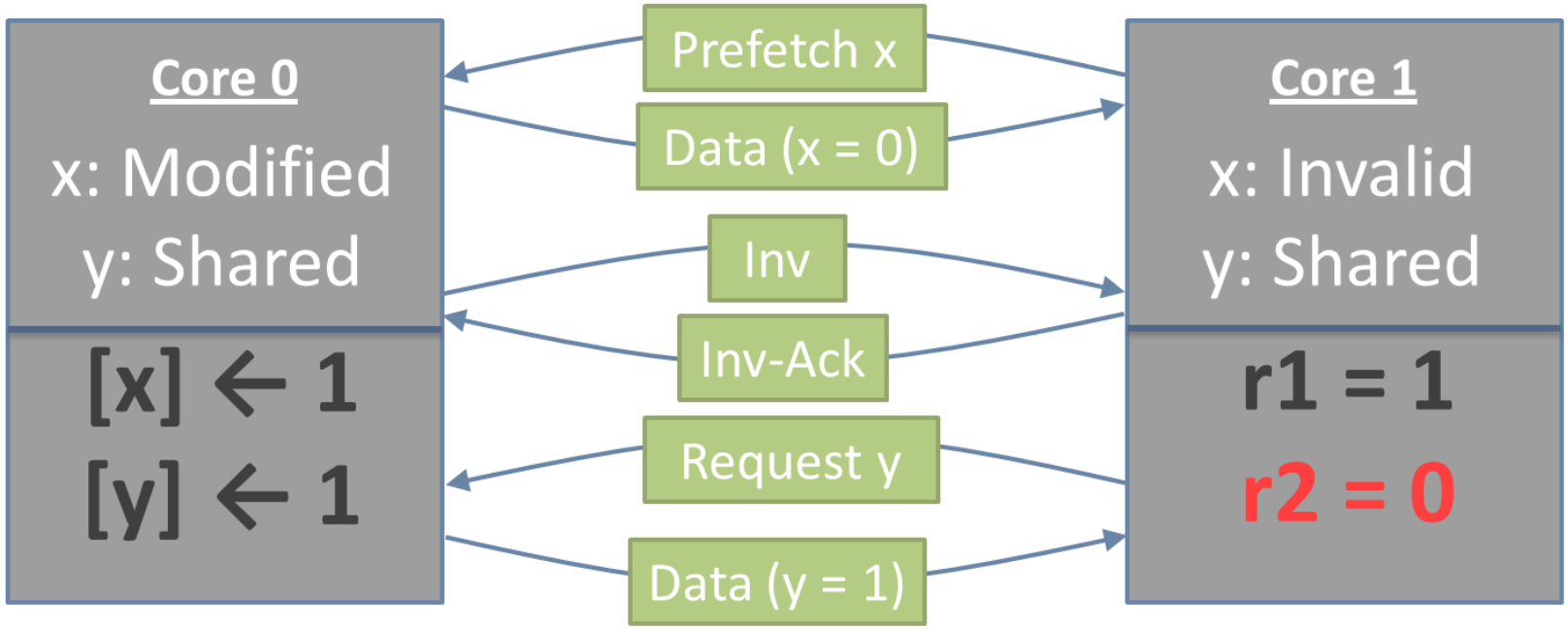
[Lustig et al. MICRO-47]

Modelling CCI Events

- Need to model **per-cache occupancy**
  - Lazy coherence and partial incoherence (e.g. GPUs)
- Need to model **coherence transitions** that relate to consistency (e.g. Peekaboo)

Modelling CCI Events

- Need to model **per-cache occupancy**
  - Lazy coherence and partial incoherence (e.g. GPUs)
- Need to model **coherence transitions** that relate to consistency (e.g. Peekaboo)



ViCL: Value in Cache Lifetime

- 4-tuple:

  (cache_id, address, data_value, generation_id)

- **cache_id** and **generation_id** uniquely identify each cache line
- A ViCL 4-tuple maps on to the period of time over which the cache line serves the data value for the address
- ViCLs start at a **ViCL Create** event and end at a **ViCL Expire** event

ViCL: Value in Cache Lifetime

- 4-tuple:

  (cache_id, address, data_value, generation_id)

- **cache_id** and **generation_id** uniquely identify each cache line
- A ViCL 4-tuple maps on to the period of time over which the cache line serves the data value for the address
- ViCLs start at a **ViCL Create** event and end at a **ViCL Expire** event

ViCL: Value in Cache Lifetime

- 4-tuple:

  (cache_id, address, data_value, generation_id)

- **cache_id** and **generation_id** uniquely identify each cache line
- A ViCL 4-tuple maps on to the period of time over which the cache line serves the data value for the address
- ViCLs start at a **ViCL Create** event and end at a **ViCL Expire** event

ViCL: Value in Cache Lifetime

- 4-tuple:

  (cache_id, address, data_value, generation_id)

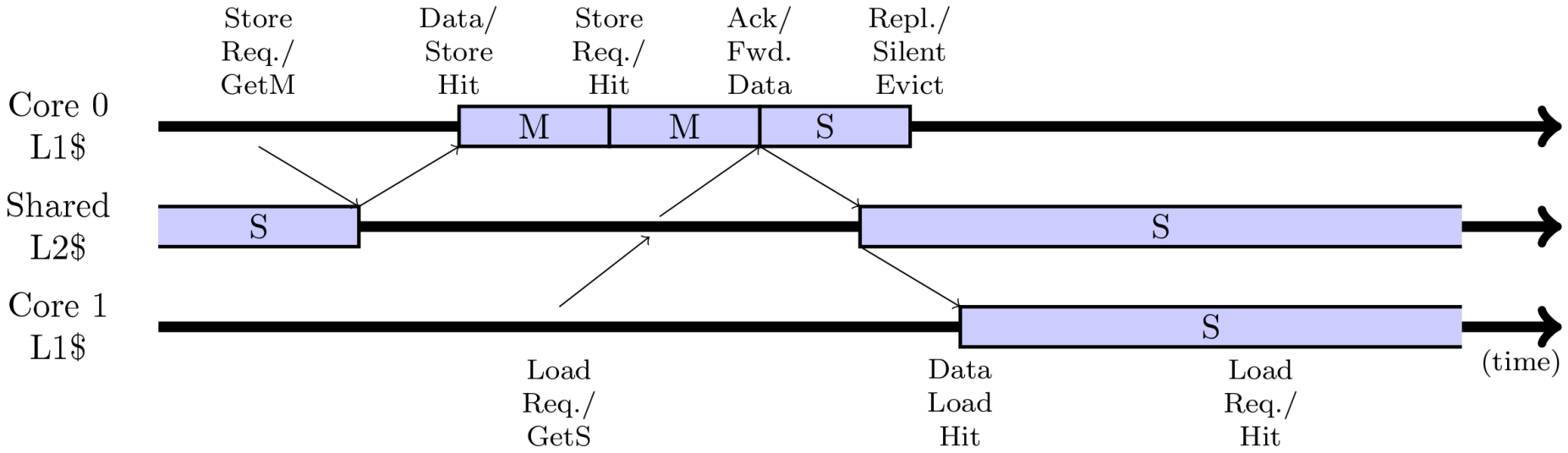
- **cache_id** and **generation_id** uniquely identify each cache line
- A ViCL 4-tuple maps on to the period of time over which the cache line serves the data value for the address
- ViCLs start at a **ViCL Create** event and end at a **ViCL Expire** event

ViCL: Value in Cache Lifetime

Conventional **co-mp** timeline (M = Modified, S = Shared)

Litmus Test **co-mp**

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	

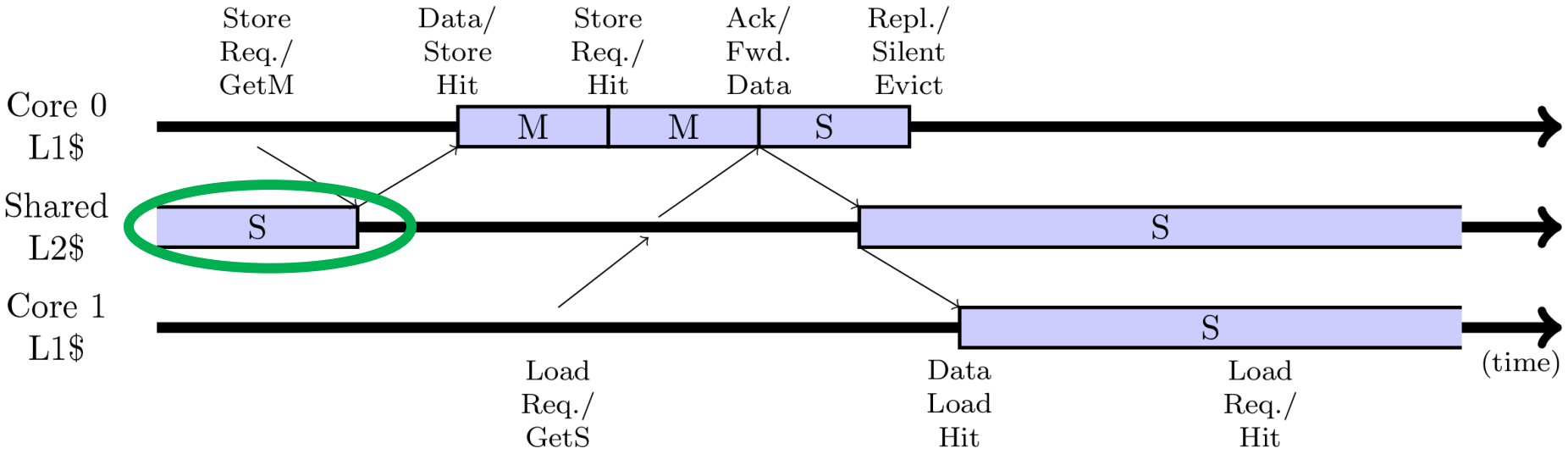


ViCL: Value in Cache Lifetime

Conventional **co-mp** timeline (M = Modified, S = Shared)

Litmus Test **co-mp**

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	

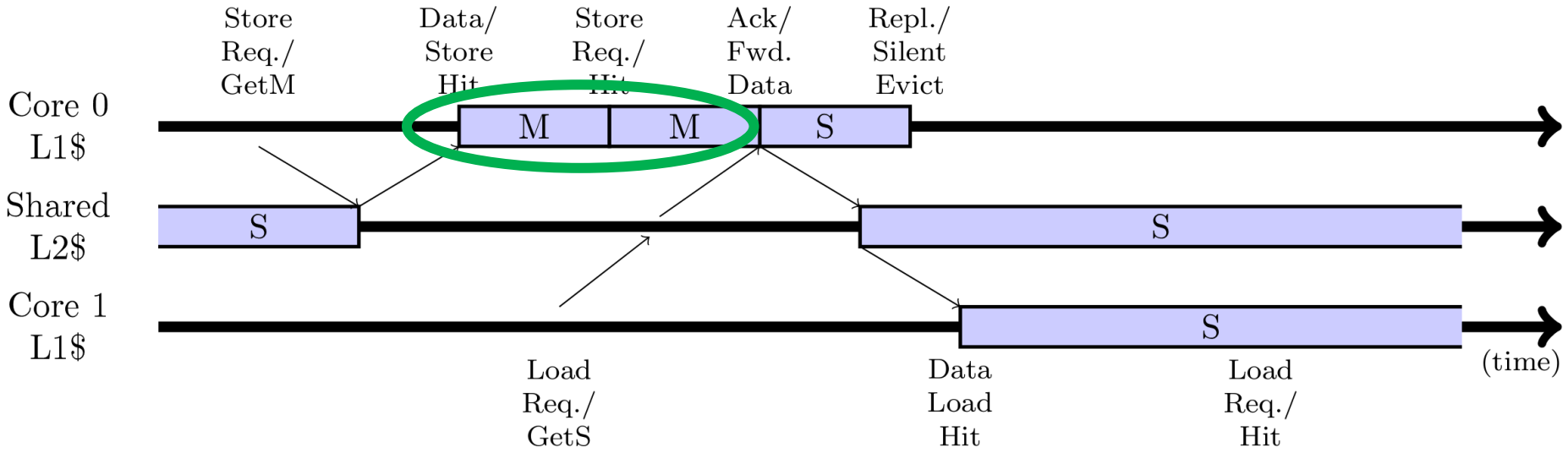


ViCL: Value in Cache Lifetime

Conventional **co-mp** timeline (M = Modified, S = Shared)

Litmus Test **co-mp**

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

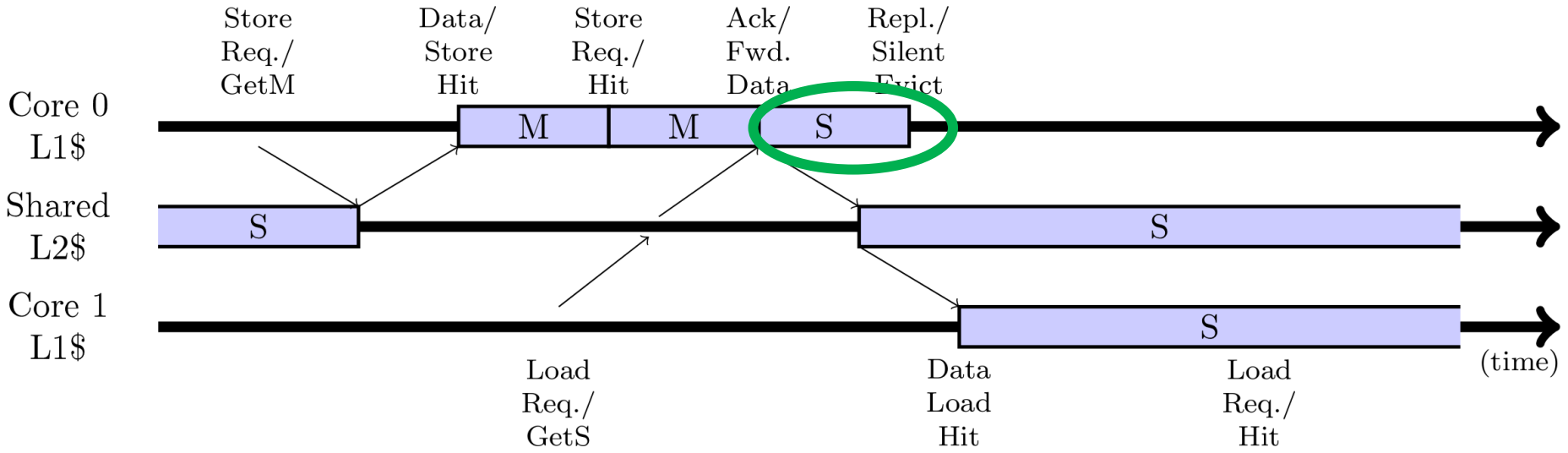


ViCL: Value in Cache Lifetime

Conventional **co-mp** timeline (M = Modified, S = Shared)

Litmus Test **co-mp**

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	

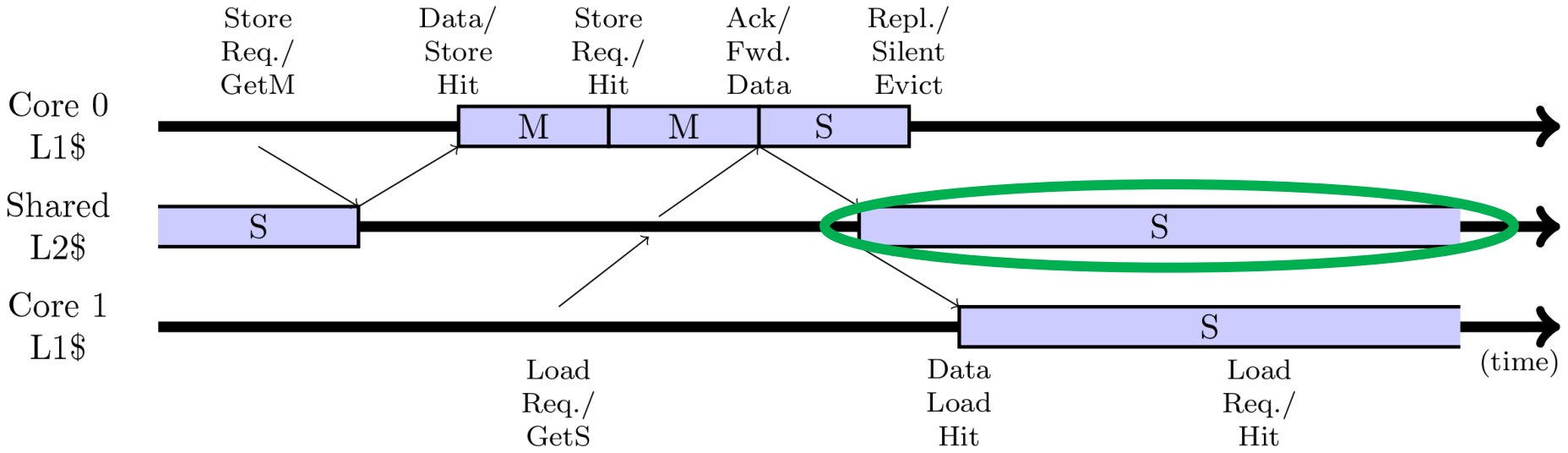


ViCL: Value in Cache Lifetime

Conventional **co-mp** timeline (M = Modified, S = Shared)

Litmus Test **co-mp**

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	

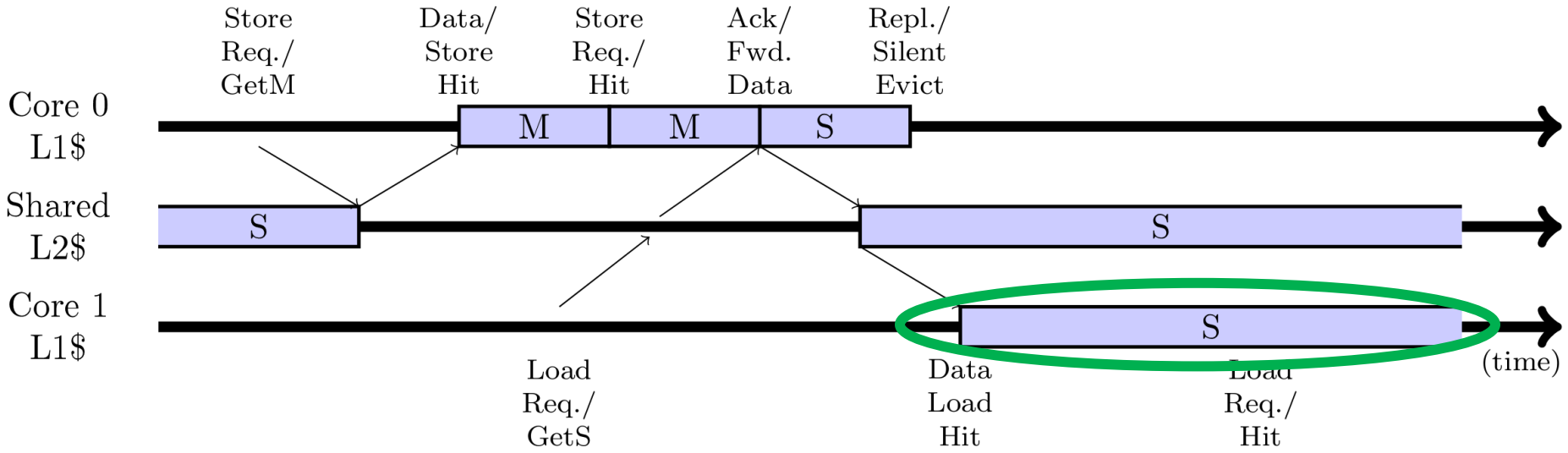


ViCL: Value in Cache Lifetime

Conventional **co-mp** timeline (M = Modified, S = Shared)

Litmus Test **co-mp**

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

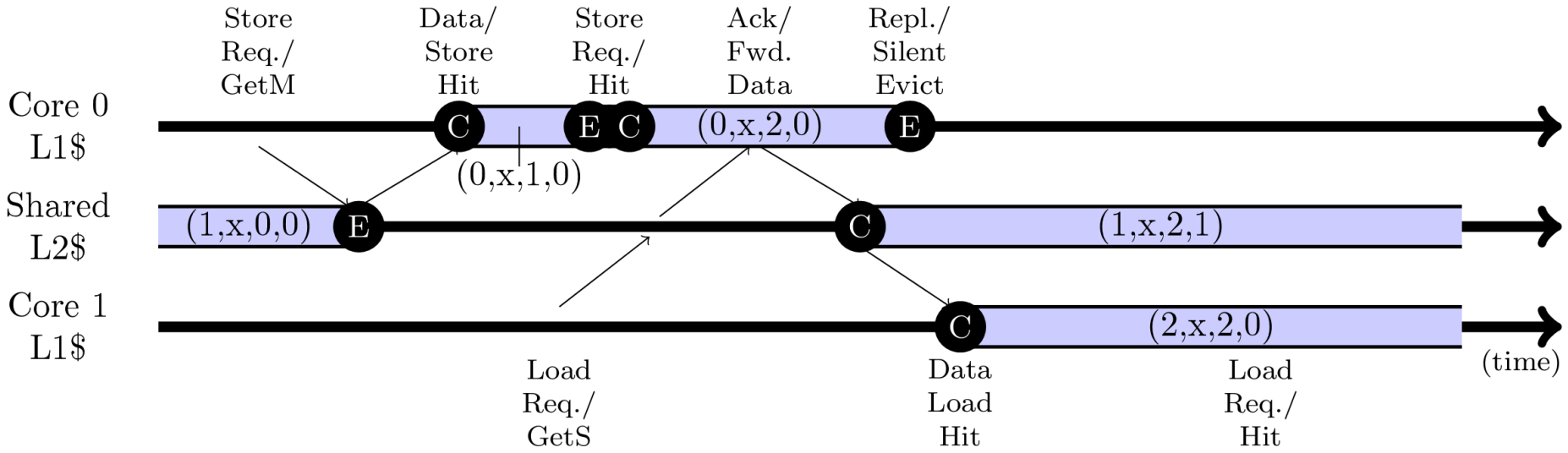


ViCL: Value in Cache Lifetime

Now with ViCLs

Litmus Test **co-mp**

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	



ViCL Nodes

C ViCL Create

E ViCL Expire

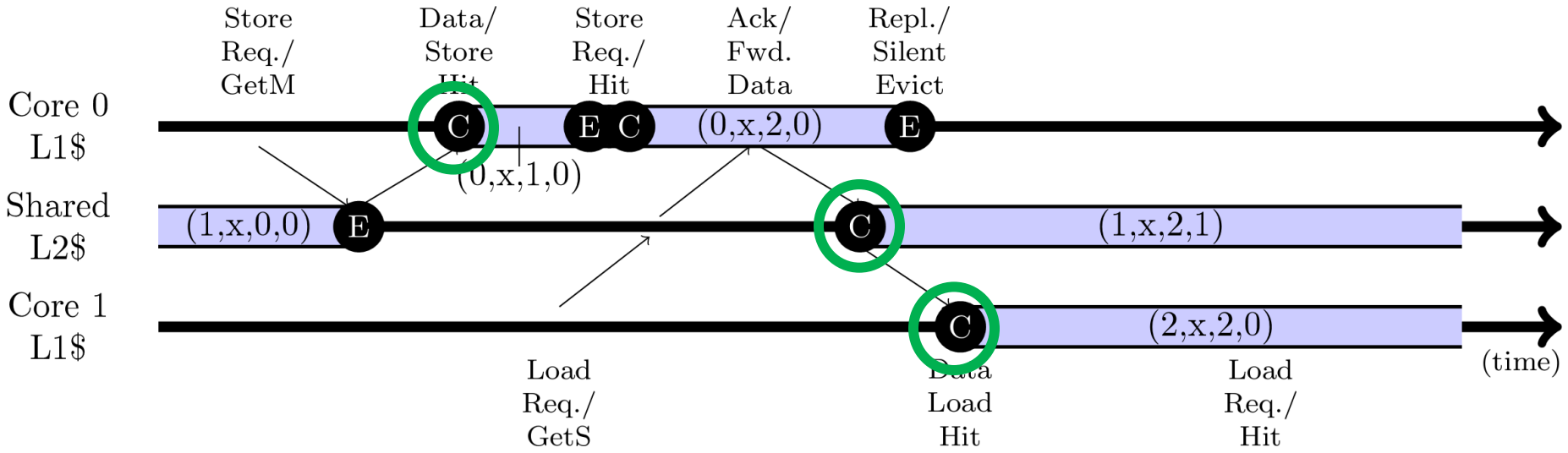
ViCL (cache id, addr, data, gen. id)

ViCL: Value in Cache Lifetime

Now with ViCLs

Litmus Test **co-mp**

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	



ViCL Nodes

C ViCL Create

E ViCL Expire

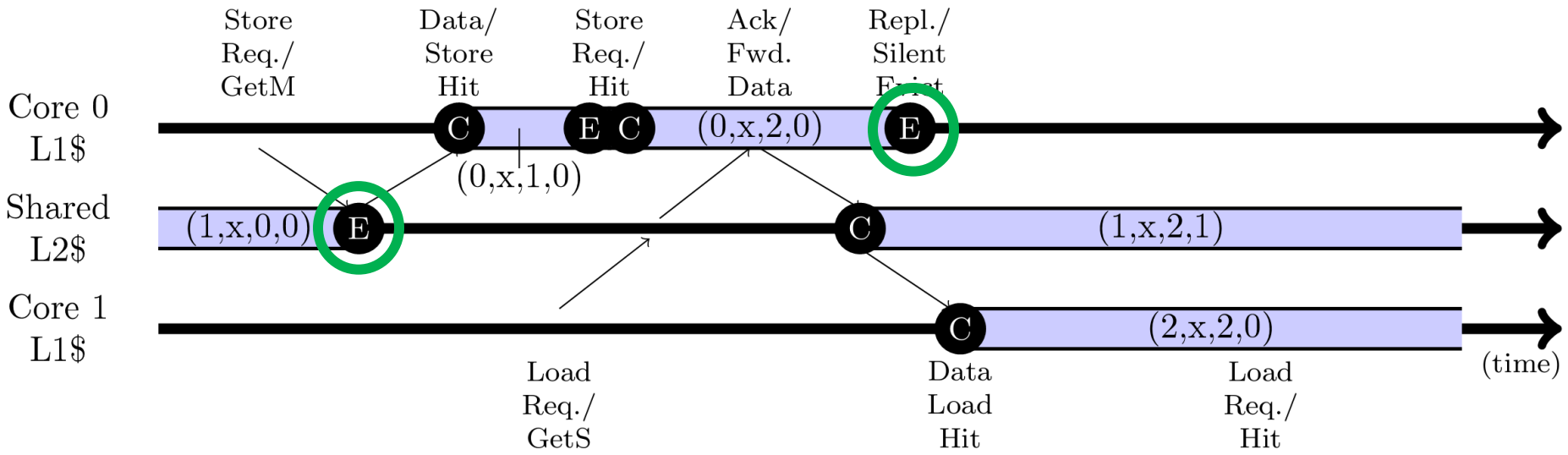
ViCL (cache id, addr, data, gen. id)

ViCL: Value in Cache Lifetime

Now with ViCLs

Litmus Test **co-mp**

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	



ViCL Nodes

C ViCL Create

E ViCL Expire

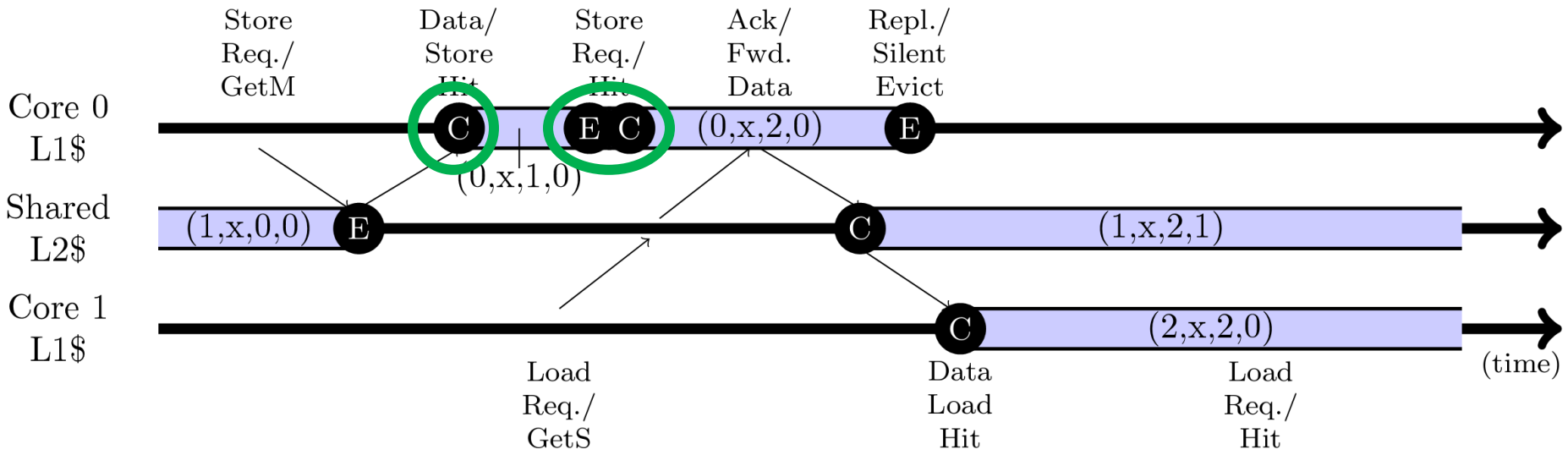
ViCL (cache id, addr, data, gen. id)

ViCL: Value in Cache Lifetime

Now with ViCLs

Litmus Test **co-mp**

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	



ViCL Nodes

C ViCL Create

E ViCL Expire

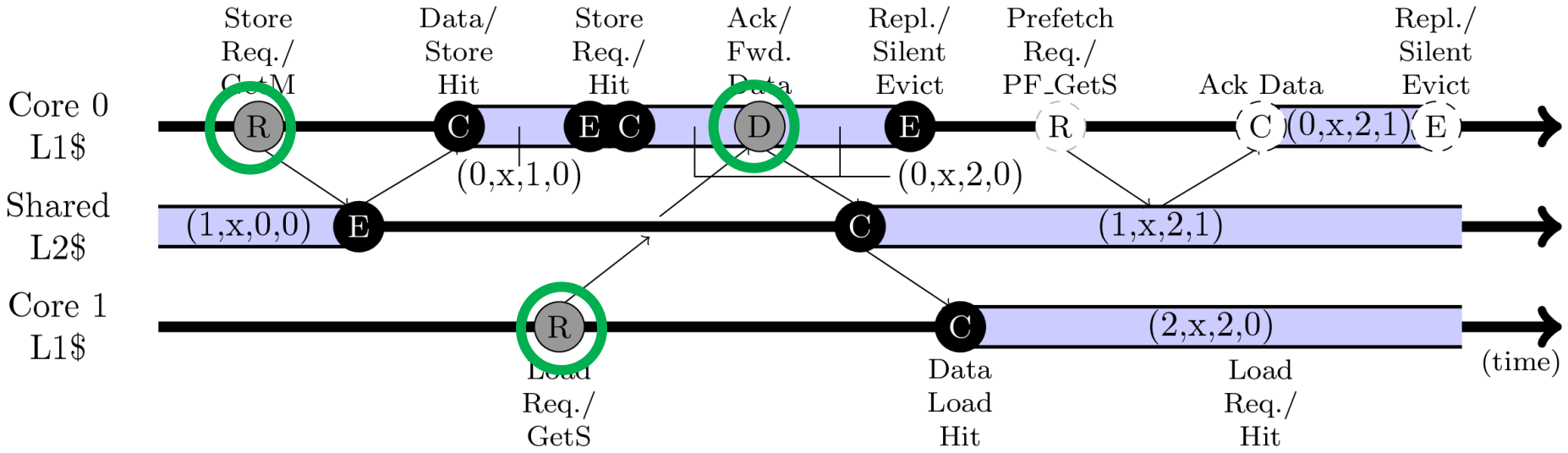
ViCL (cache id, addr, data, gen. id)

ViCL: Value in Cache Lifetime

Can model requests, downgrades, etc.

Litmus Test **co-mp**

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	



Always Enumerated

C ViCL Create

E ViCL Expire

Enumerated as Needed

R $ Line Request

D $ Line Downgrade

Not Enumerated

C ViCL Create

E ViCL Expire

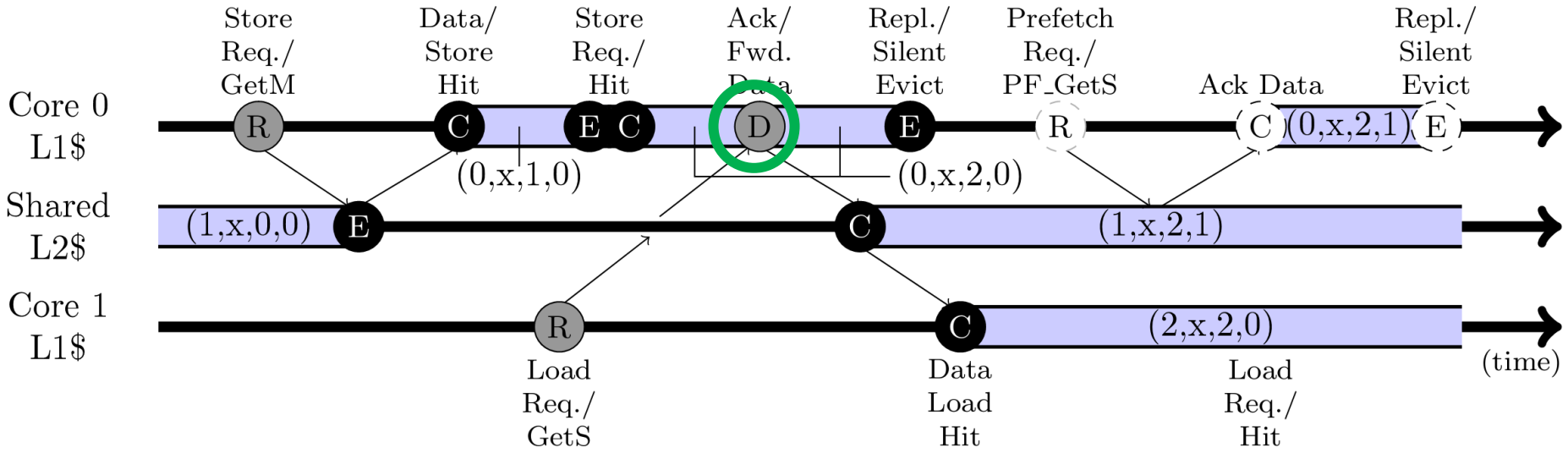
R $ Line Request

ViCL: Value in Cache Lifetime

Can model requests, downgrades, etc.

Litmus Test **co-mp**

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	



Always Enumerated

C ViCL Create — E ViCL Expire

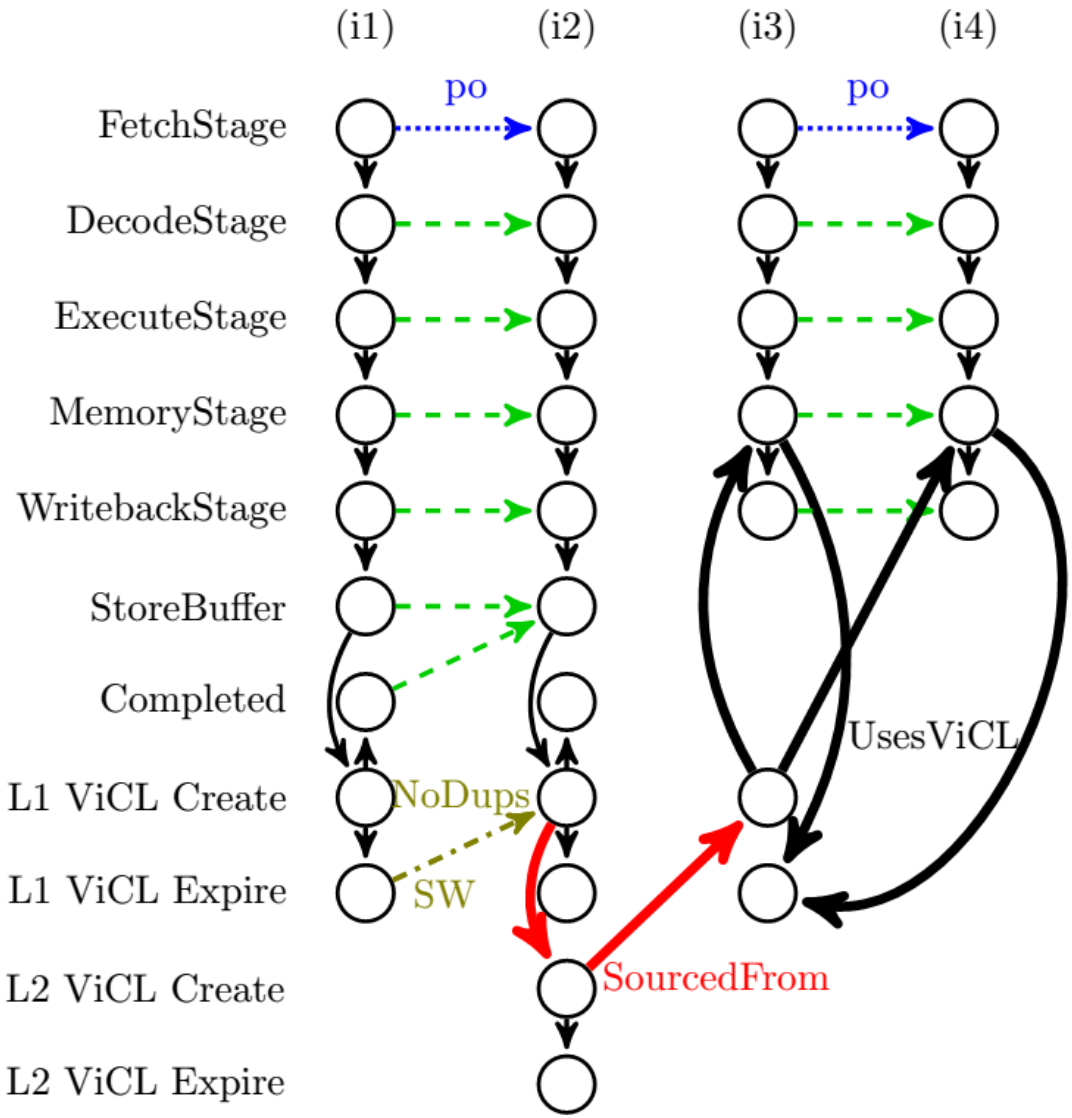
Enumerated as Needed

R $ Line Request — D $ Line Downgrade

Not Enumerated

(C) ViCL Create — (E) ViCL Expire — (R) $ Line Request

ViCLs in μhb Graphs

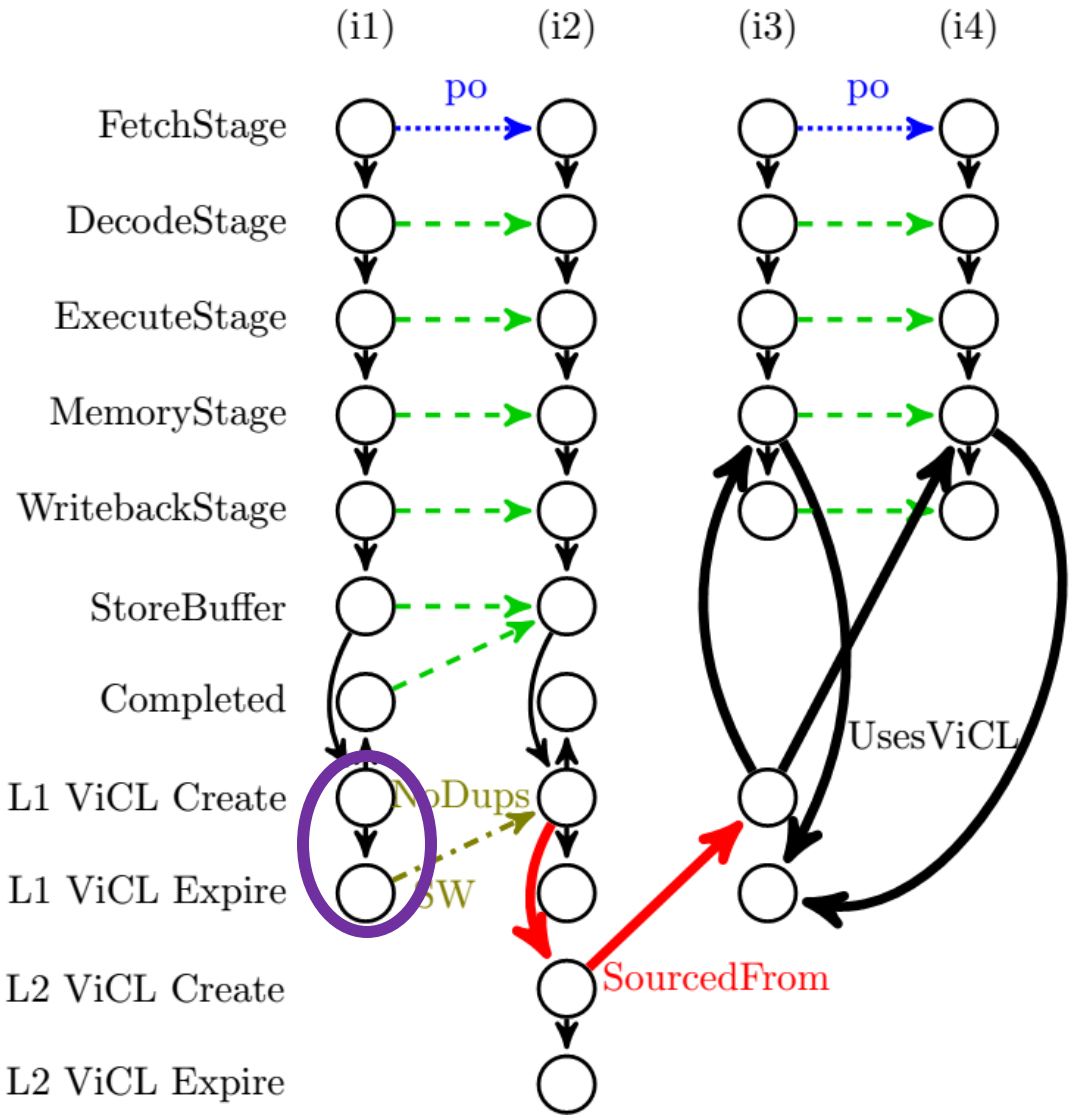


Litmus Test **co-mp**

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

- Use pipeline model from PipeCheck, but add ViCL nodes and edges

ViCLs in μhb Graphs

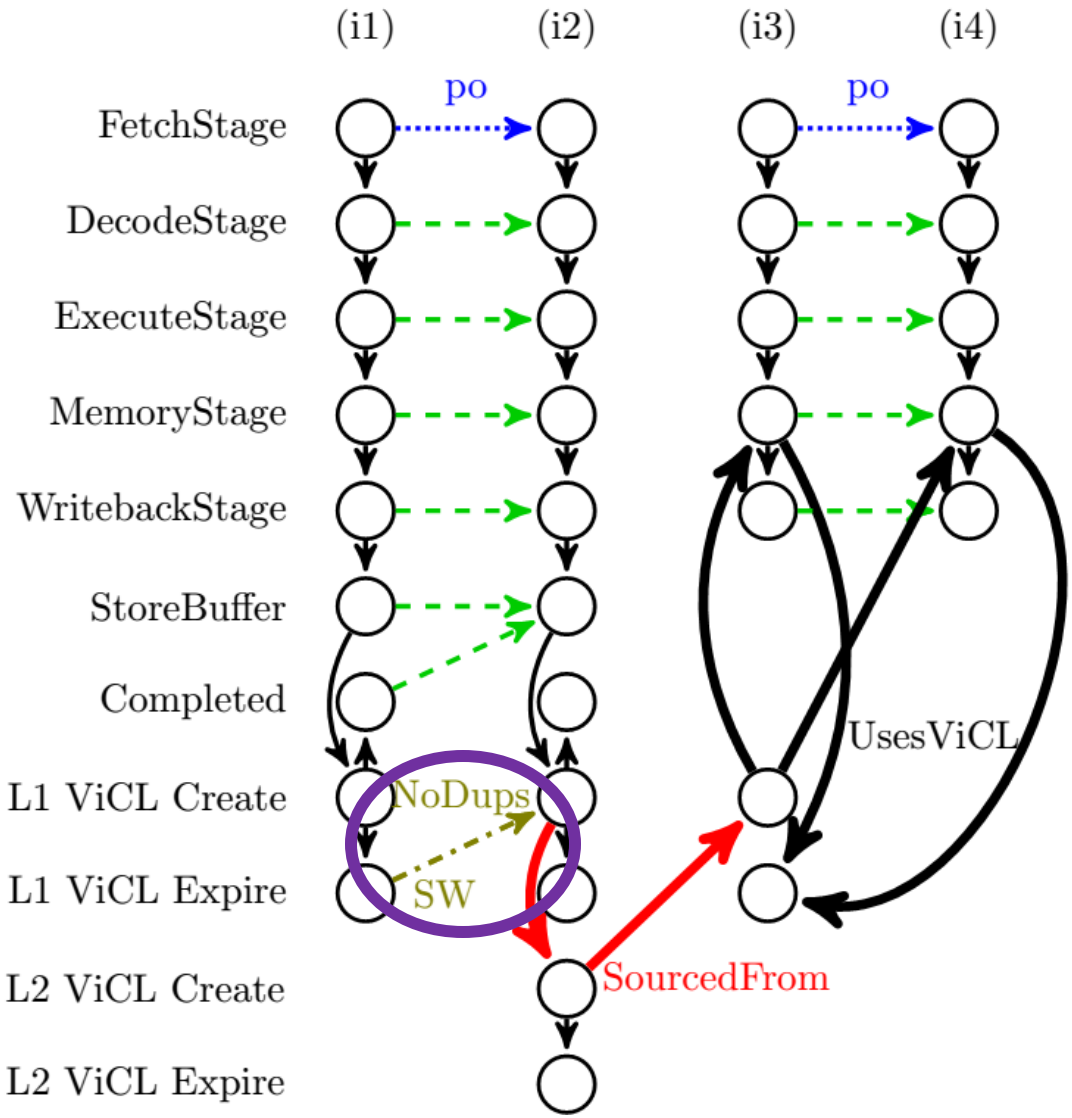


Litmus Test **co-mp**

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

- Use pipeline model from PipeCheck, but add ViCL nodes and edges

ViCLs in μhb Graphs

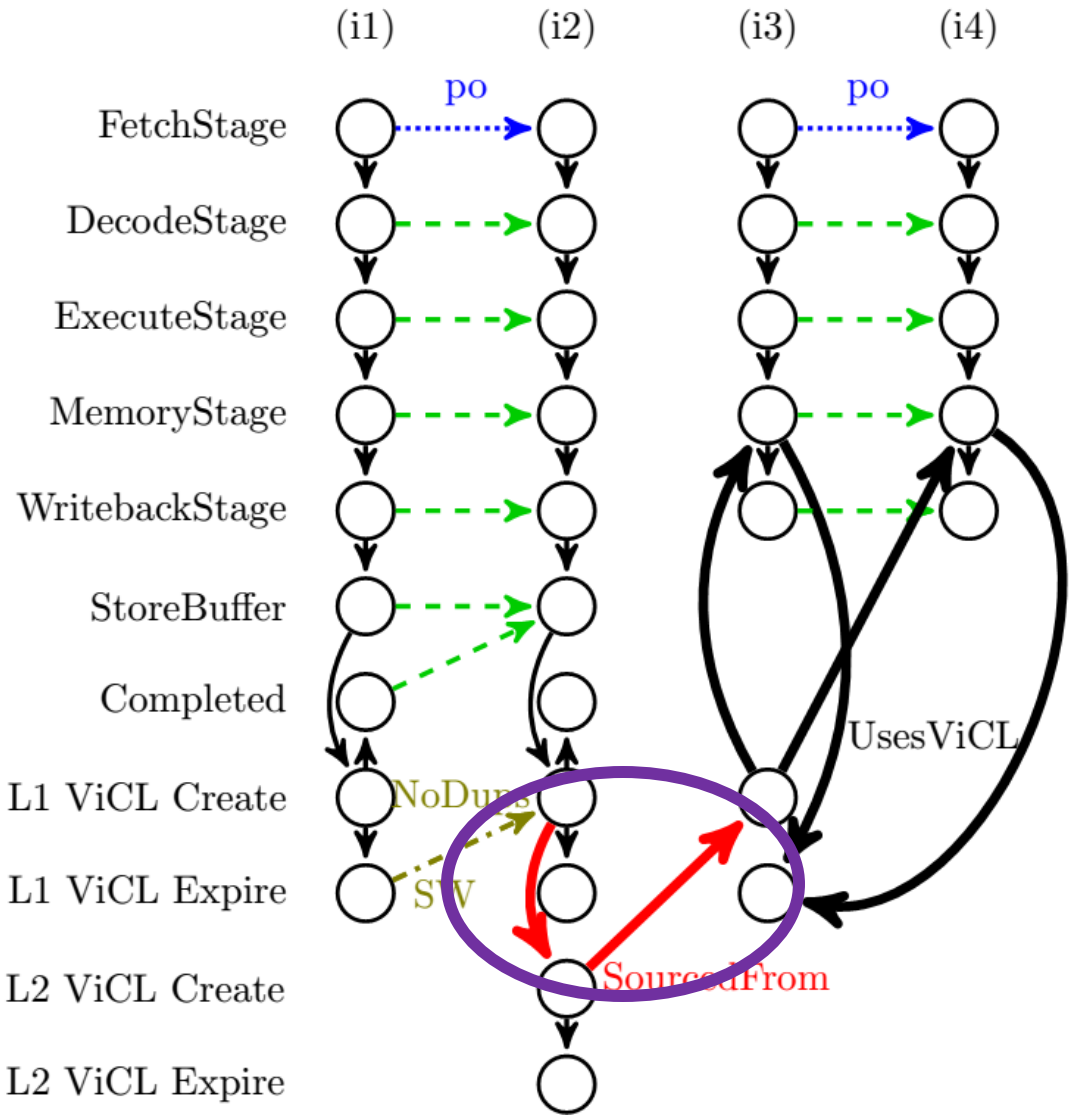


Litmus Test **co-mp**

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

- Use pipeline model from PipeCheck, but add ViCL nodes and edges

ViCLs in μhb Graphs

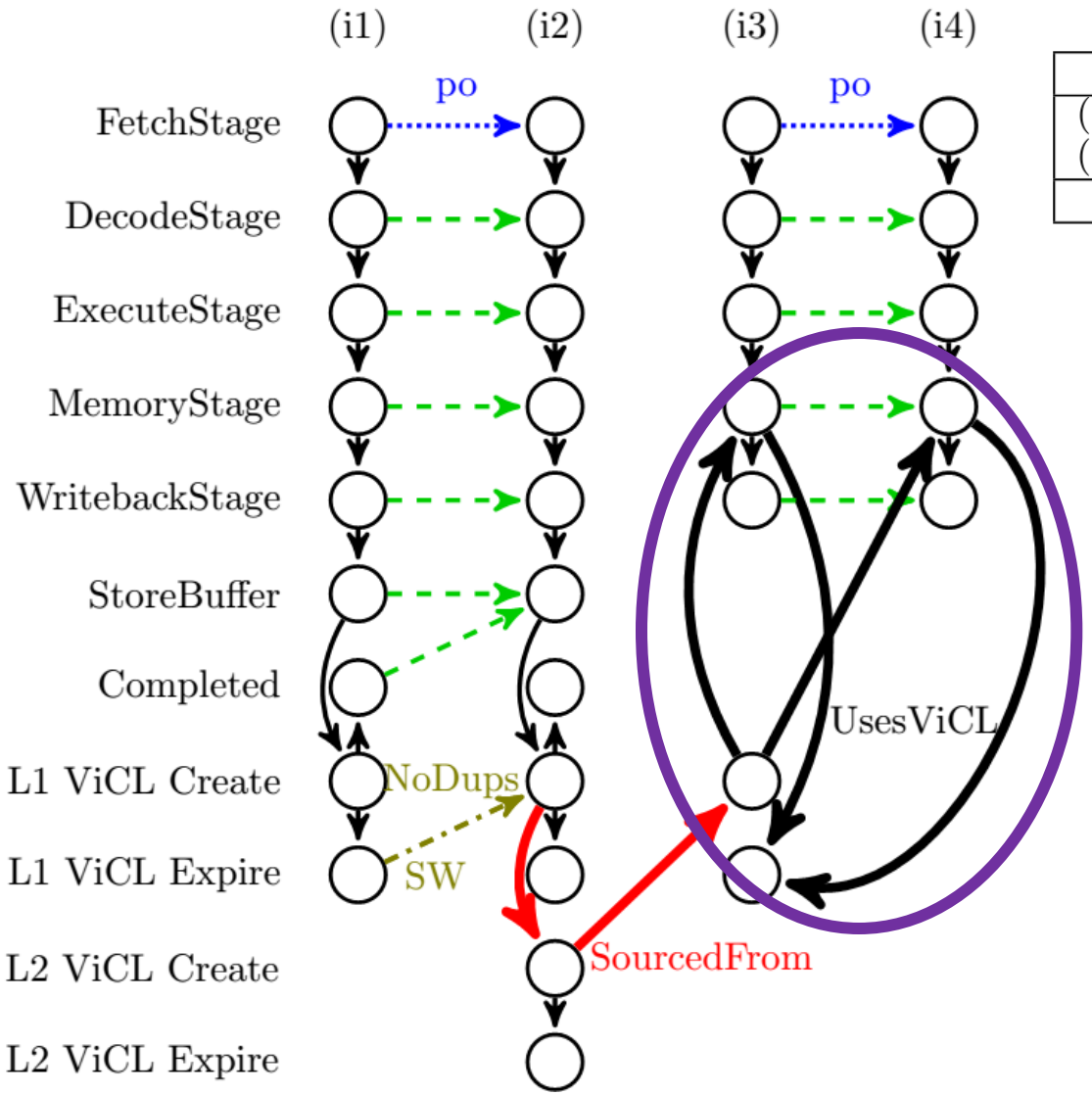


Litmus Test **co-mp**

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

- Use pipeline model from PipeCheck, but add ViCL nodes and edges

ViCLs in μhb Graphs

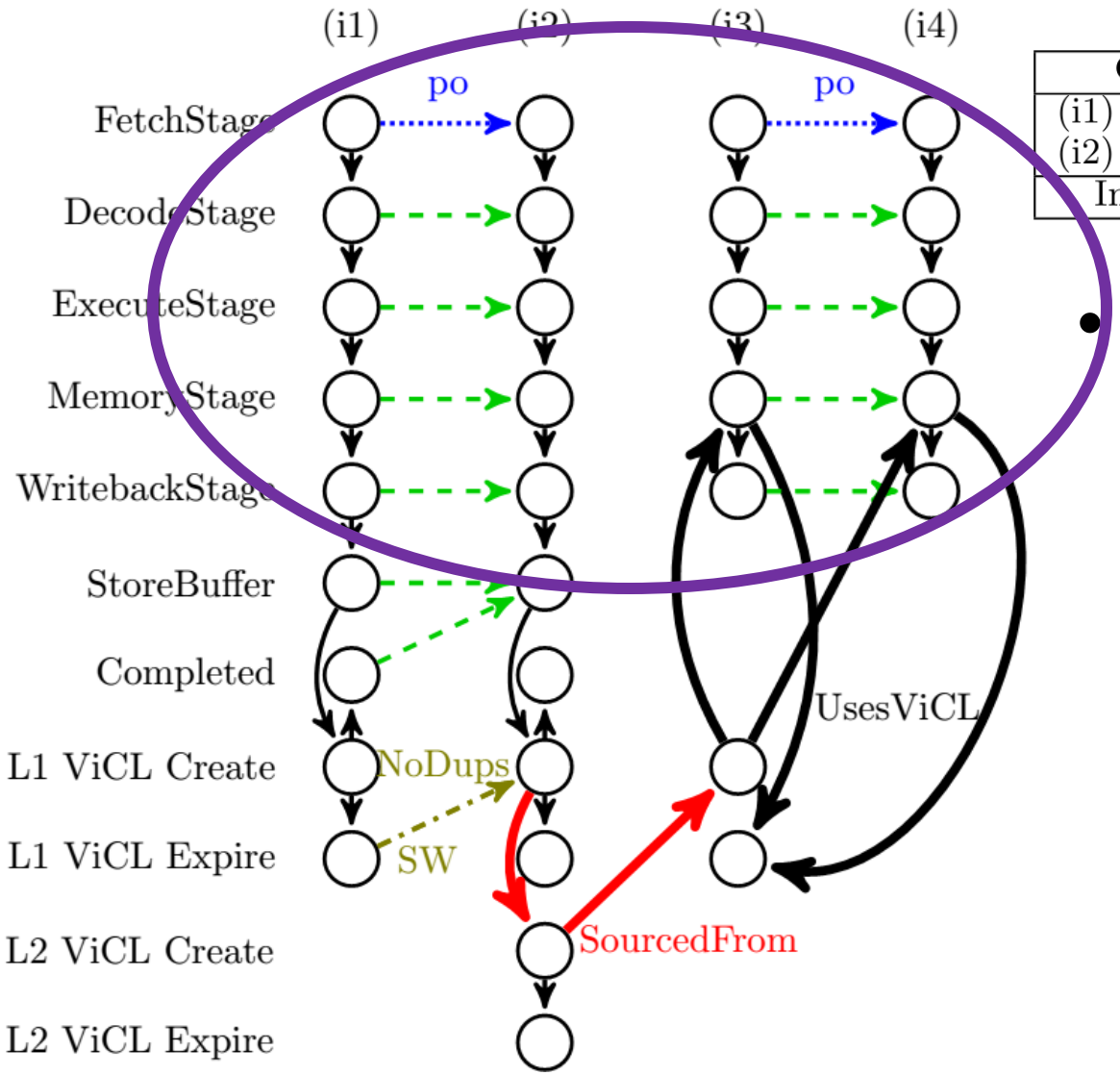


Litmus Test **co-mp**

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

- Use pipeline model from PipeCheck, but add ViCL nodes and edges

ViCLs in μhb Graphs

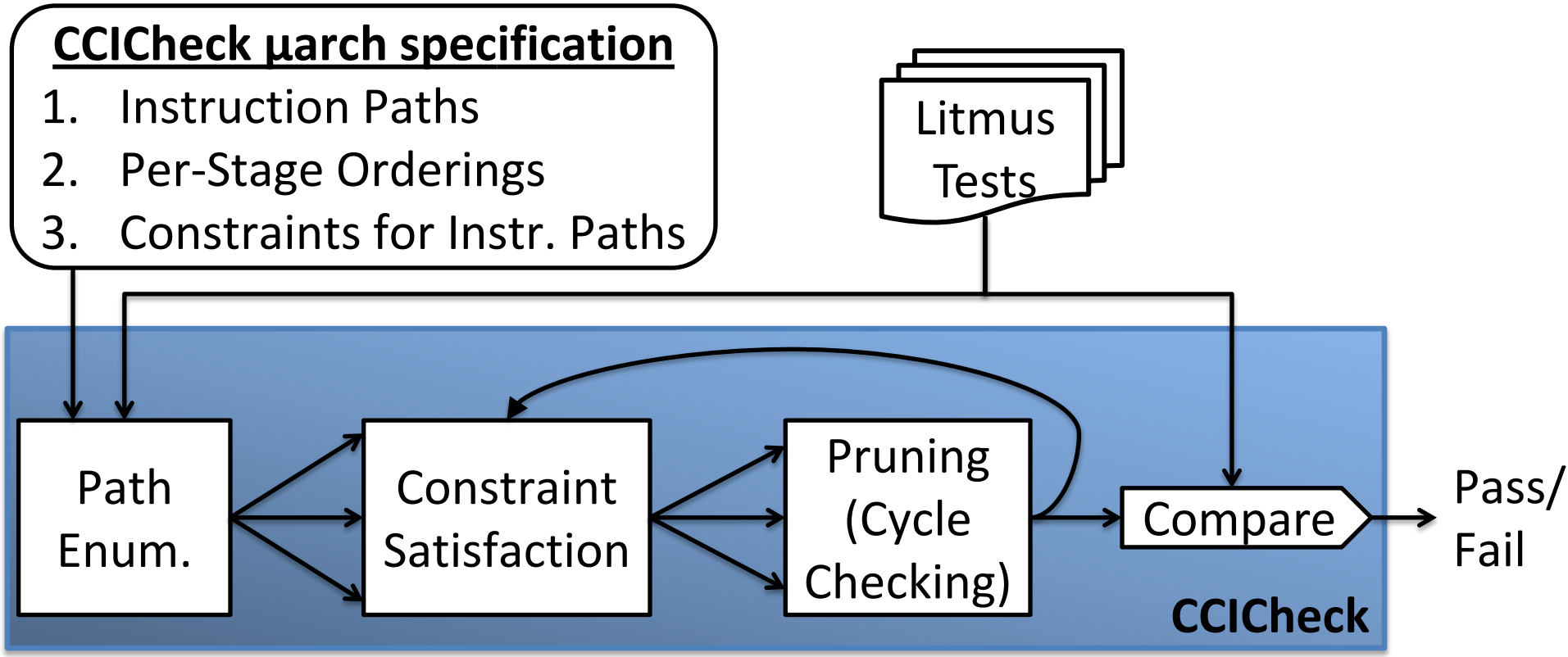


Litmus Test **co-mp**

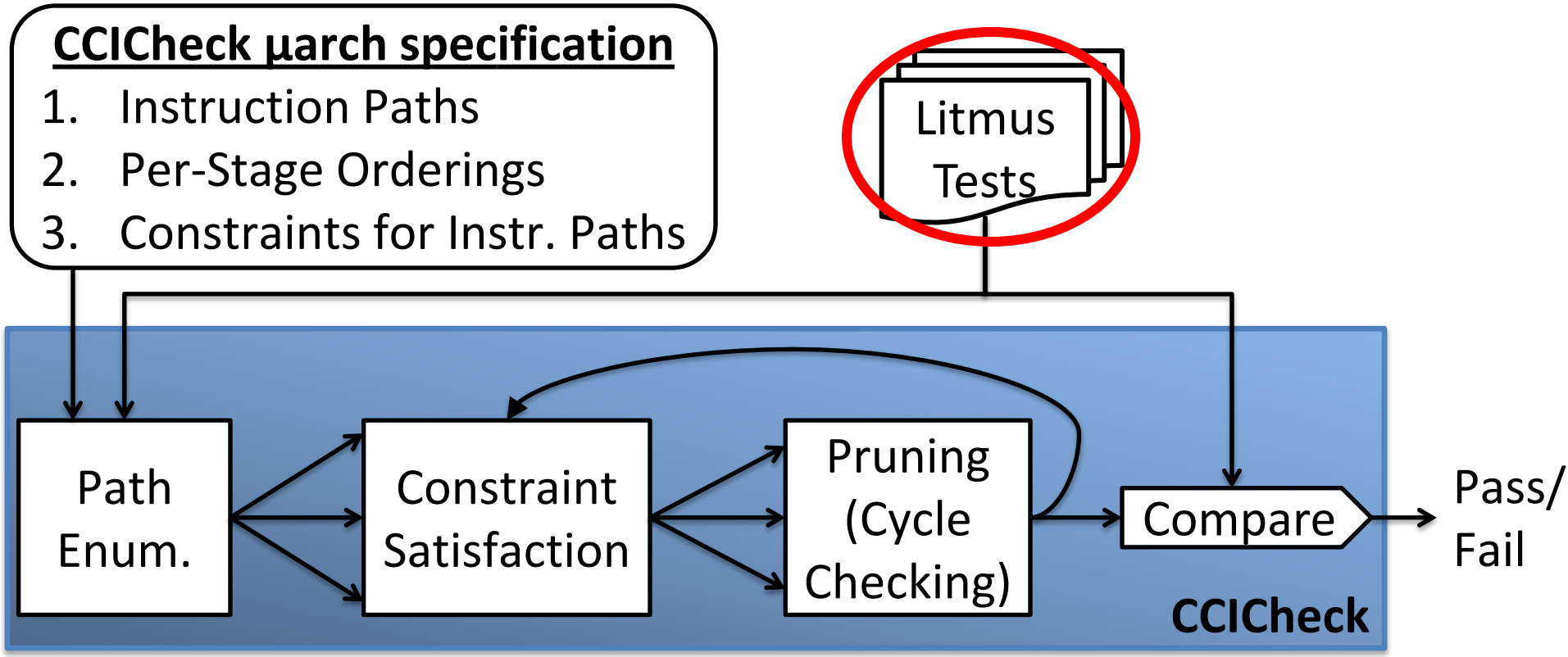
Core 0	Core 1
(i1) St [x] ← 1	(i3) Ld r1 ← [x]
(i2) St [x] ← 2	(i4) Ld r2 ← [x]
In TSO: r1=2, r2=2 Allowed	

- Use pipeline model from PipeCheck, but add ViCL nodes and edges

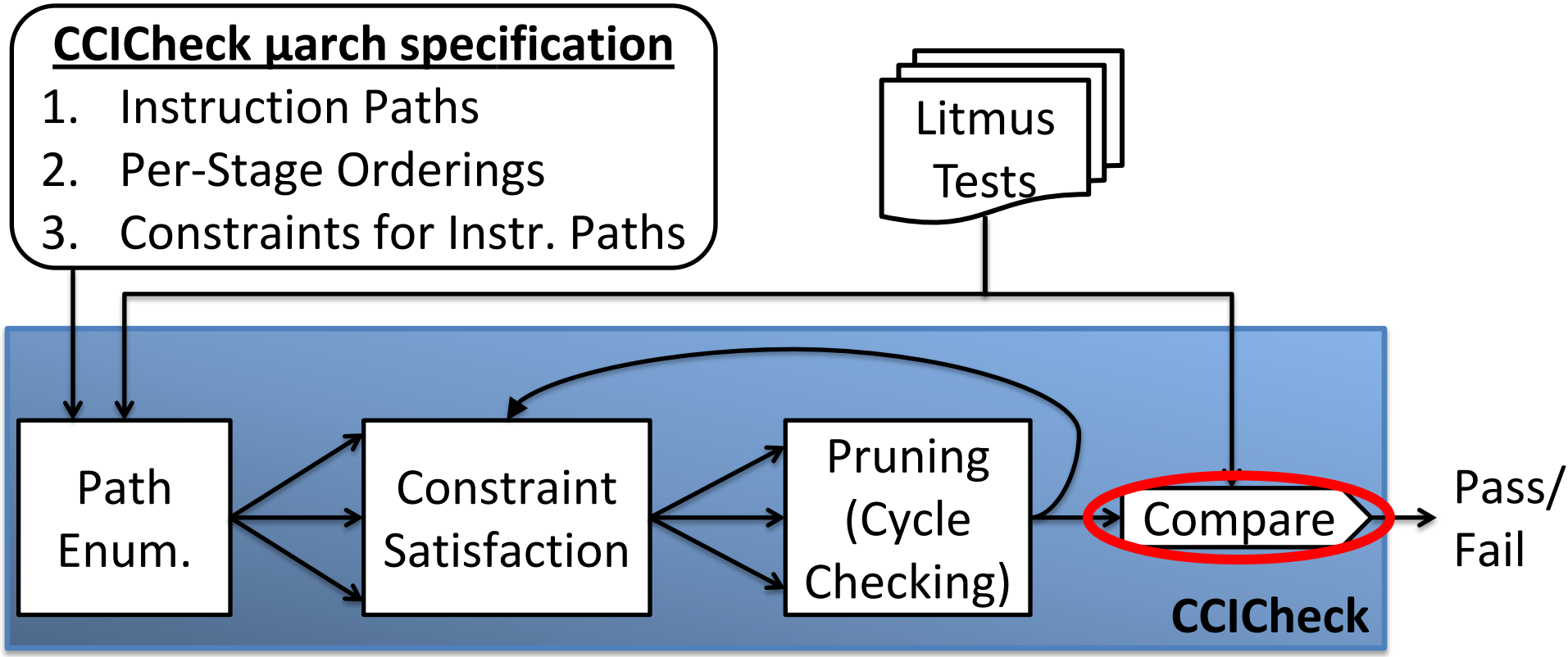
CCICheck Toolflow



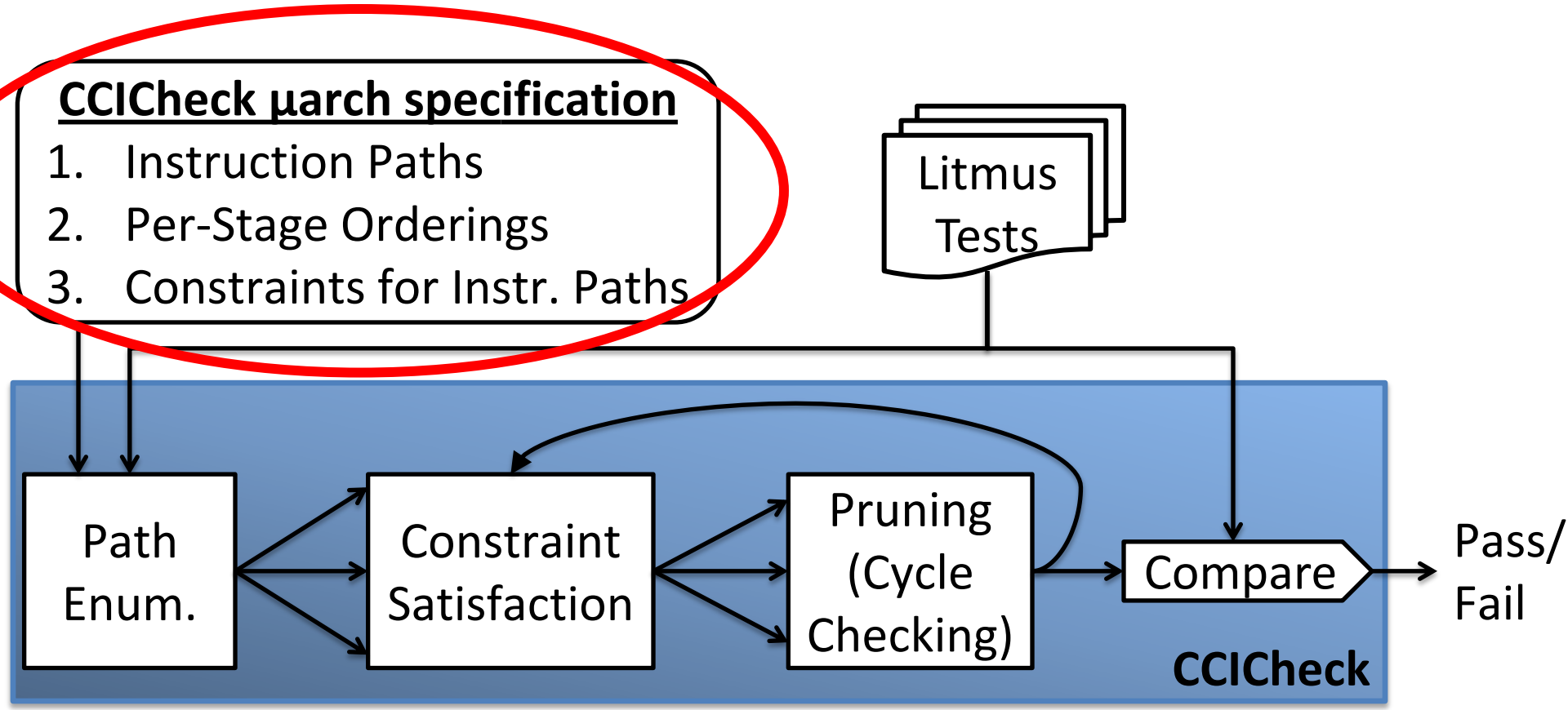
CCICheck Toolflow



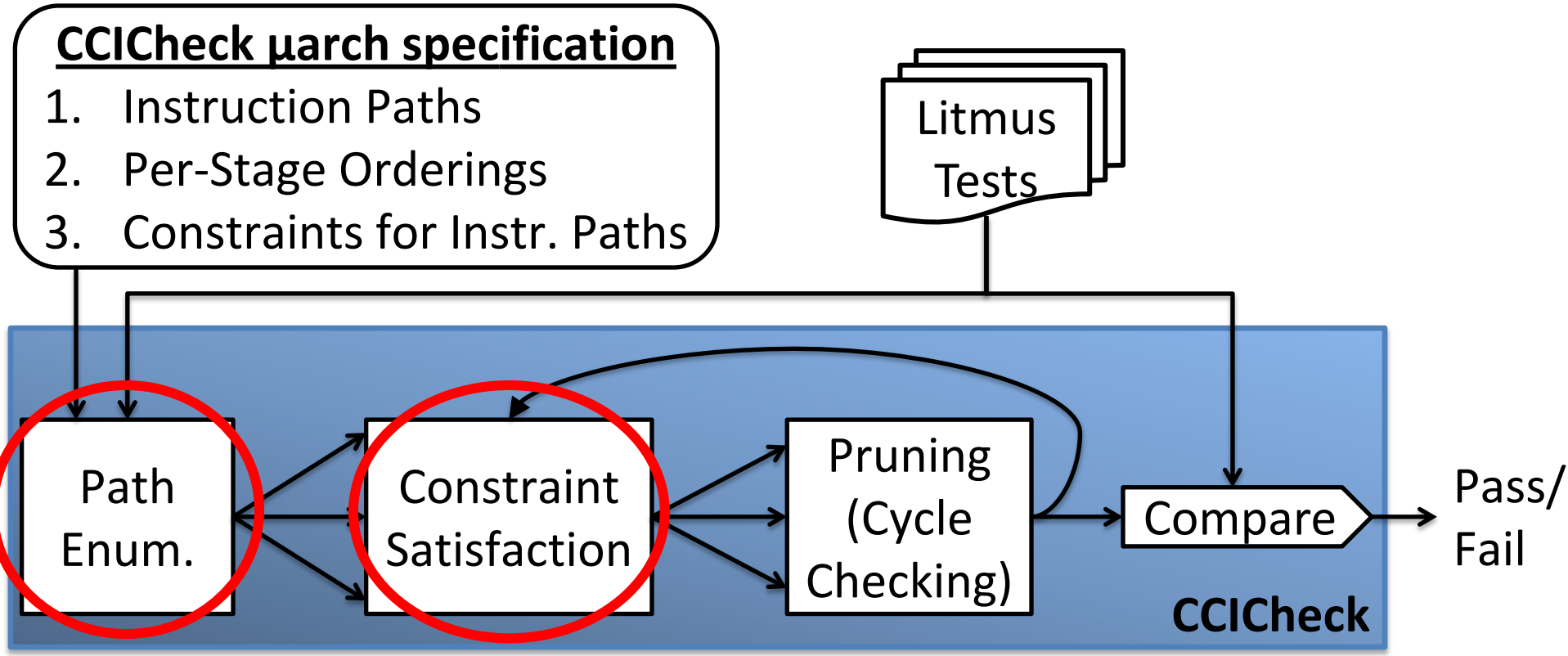
CCICheck Toolflow



CCICheck Toolflow



CCICheck Toolflow



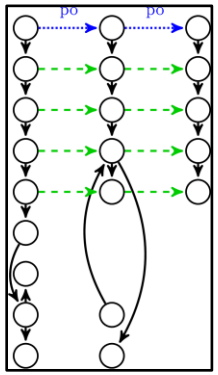
Path Enumeration

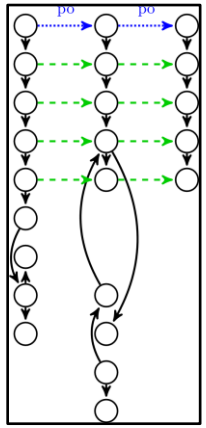
Thread	
(i1)	St [x] ← 1
(i2)	Ld r1 ← [x]
(i3)	Ld r2 ← [x]
Allowed: r1=1, r2=1	

Constraint Satisfaction

Path Enumeration

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

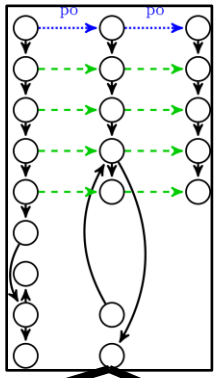


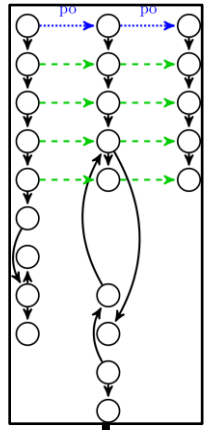


Constraint Satisfaction

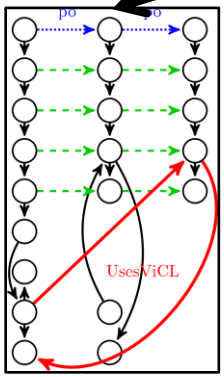
Path Enumeration

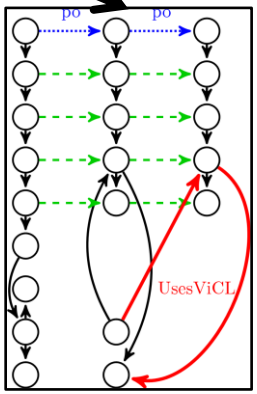
Thread	
(i1)	St [x] ← 1
(i2)	Ld r1 ← [x]
(i3)	Ld r2 ← [x]
Allowed: r1=1, r2=1	

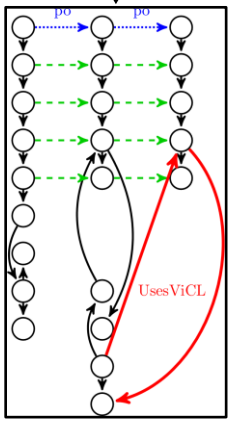




Constraint Satisfaction

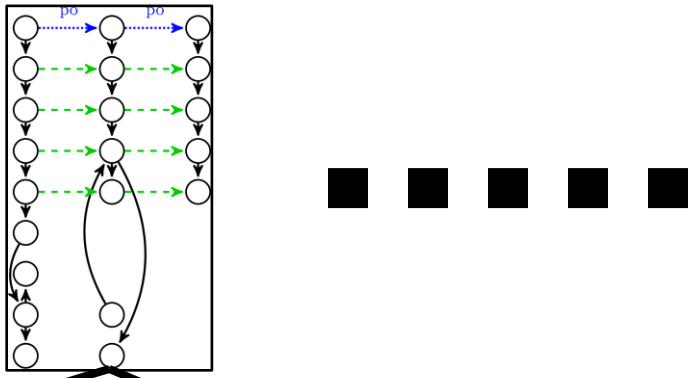


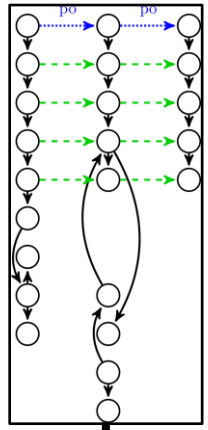




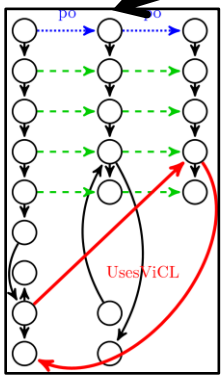
Path Enumeration

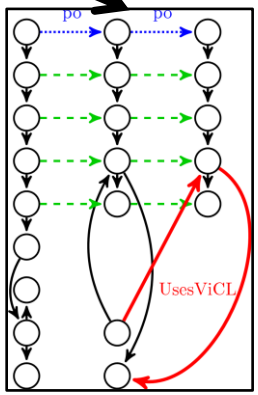
Thread	
(i1)	St [x] ← 1
(i2)	Ld r1 ← [x]
(i3)	Ld r2 ← [x]
Allowed: r1=1, r2=1	

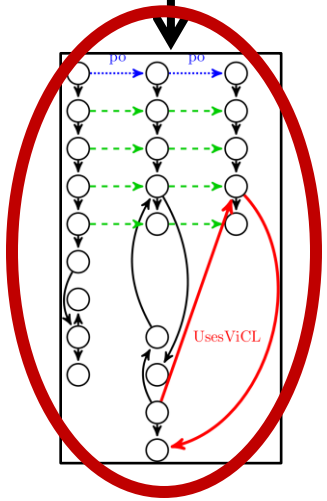




Constraint Satisfaction

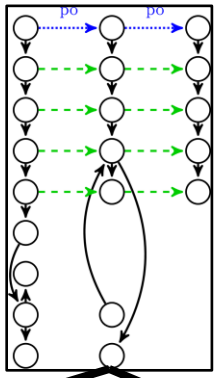


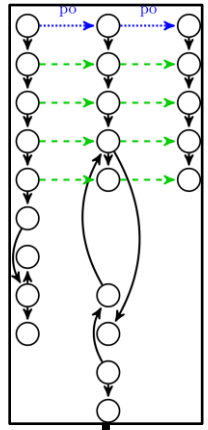




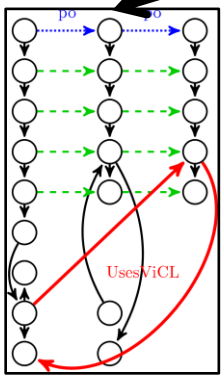
Path Enumeration

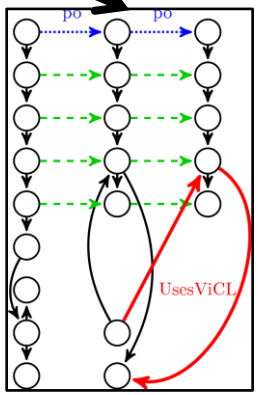
Thread	
(i1)	St [x] ← 1
(i2)	Ld r1 ← [x]
(i3)	Ld r2 ← [x]
Allowed: r1=1, r2=1	

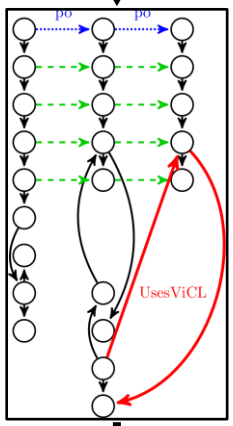




Constraint Satisfaction



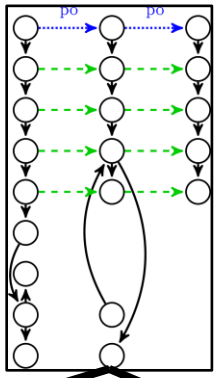


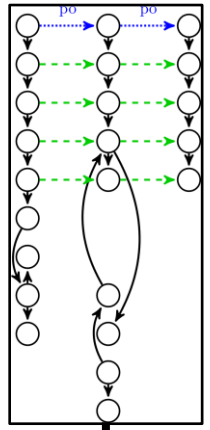


Unsatisfiable Constraint → Invalid Scenario

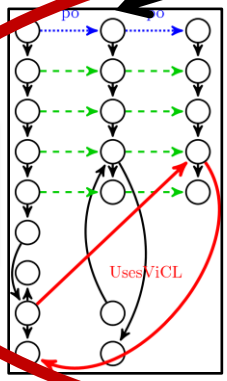
Path Enumeration

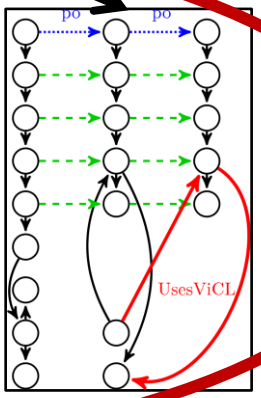
Thread	
(i1)	St [x] ← 1
(i2)	Ld r1 ← [x]
(i3)	Ld r2 ← [x]
Allowed: r1=1, r2=1	

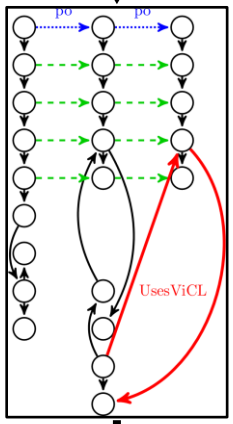




Constraint Satisfaction



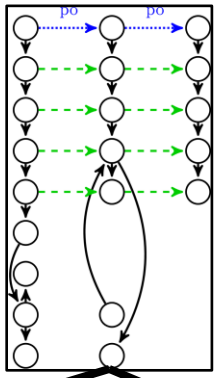


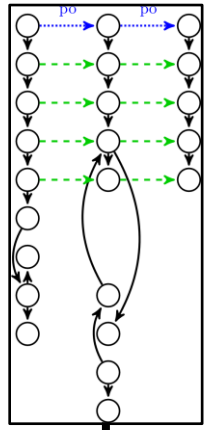


Unsatisfiable Constraint → Invalid Scenario

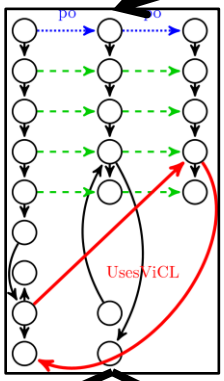
Path Enumeration

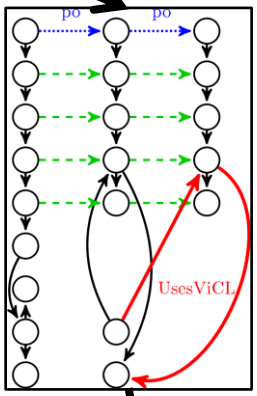
Thread	
(i1)	St [x] ← 1
(i2)	Ld r1 ← [x]
(i3)	Ld r2 ← [x]
Allowed: r1=1, r2=1	

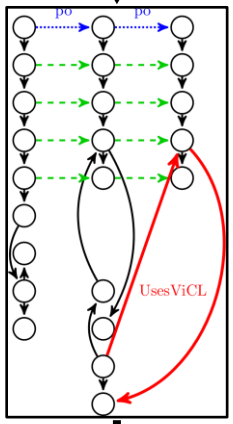


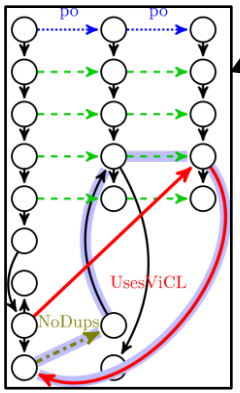


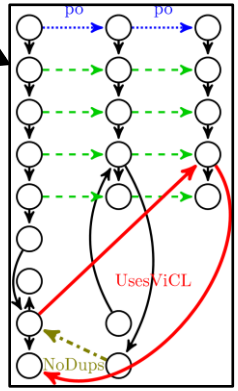
Constraint Satisfaction







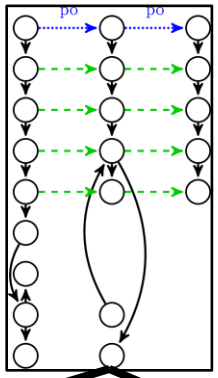


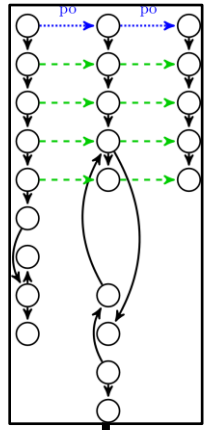


Unsatisfiable Constraint → Invalid Scenario

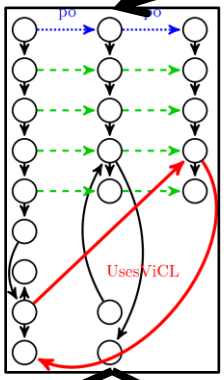
Path Enumeration

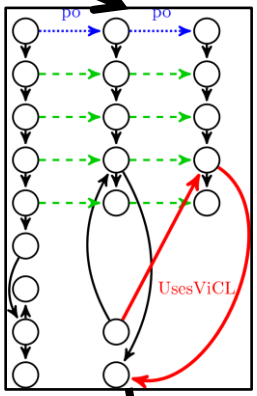
Thread	
(i1)	St [x] ← 1
(i2)	Ld r1 ← [x]
(i3)	Ld r2 ← [x]
Allowed: r1=1, r2=1	

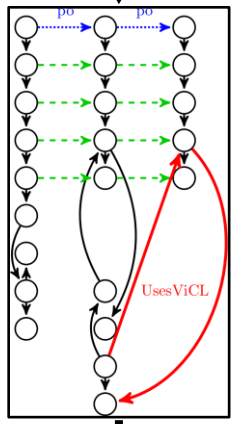


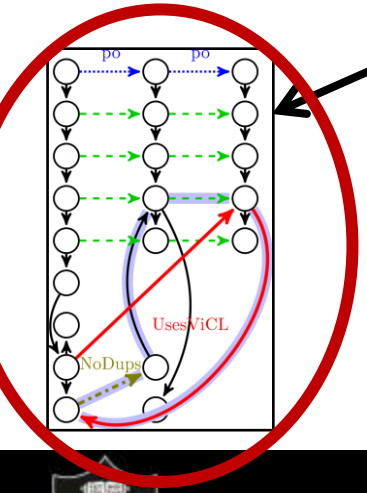


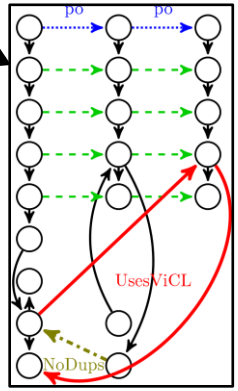
Constraint Satisfaction







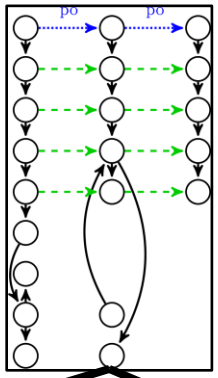


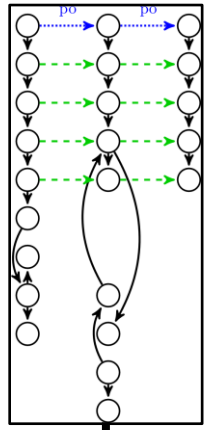


Unsatisfiable Constraint → Invalid Scenario

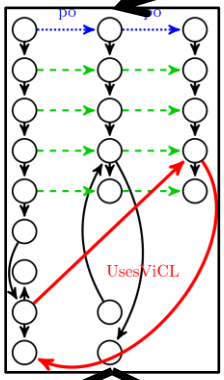
Path Enumeration

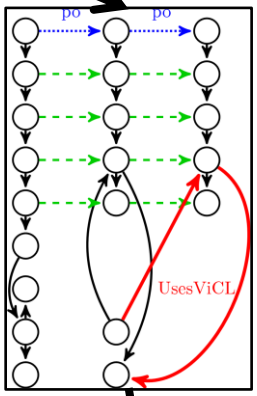
Thread	
(i1)	St [x] ← 1
(i2)	Ld r1 ← [x]
(i3)	Ld r2 ← [x]
Allowed: r1=1, r2=1	

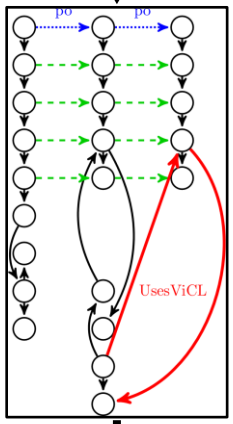


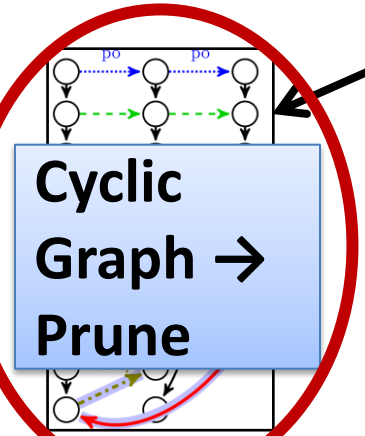


Constraint Satisfaction

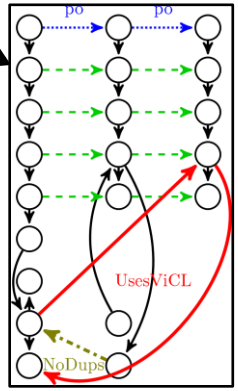








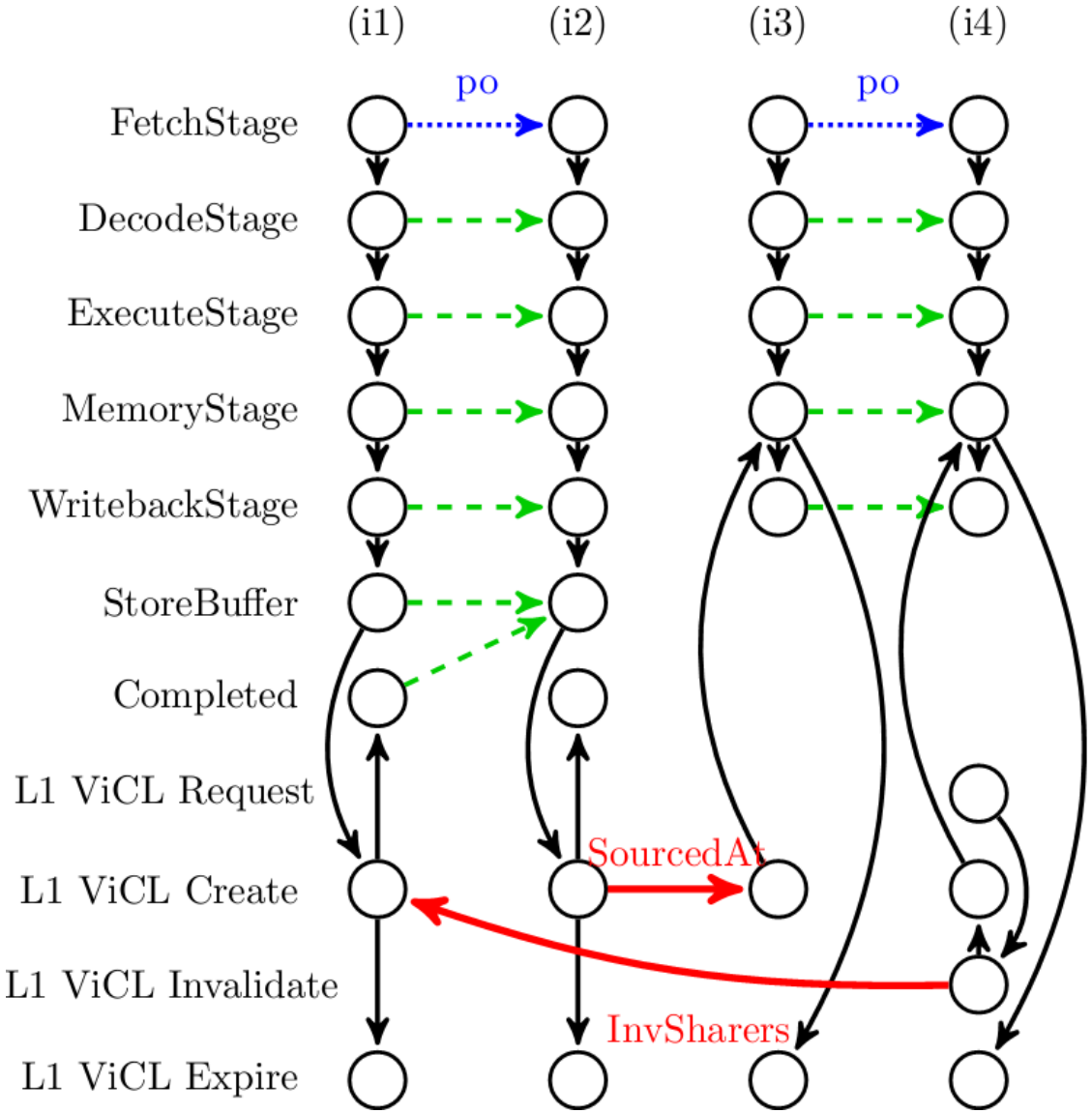
Cyclic Graph → Prune



Unsatisfiable Constraint → Invalid Scenario

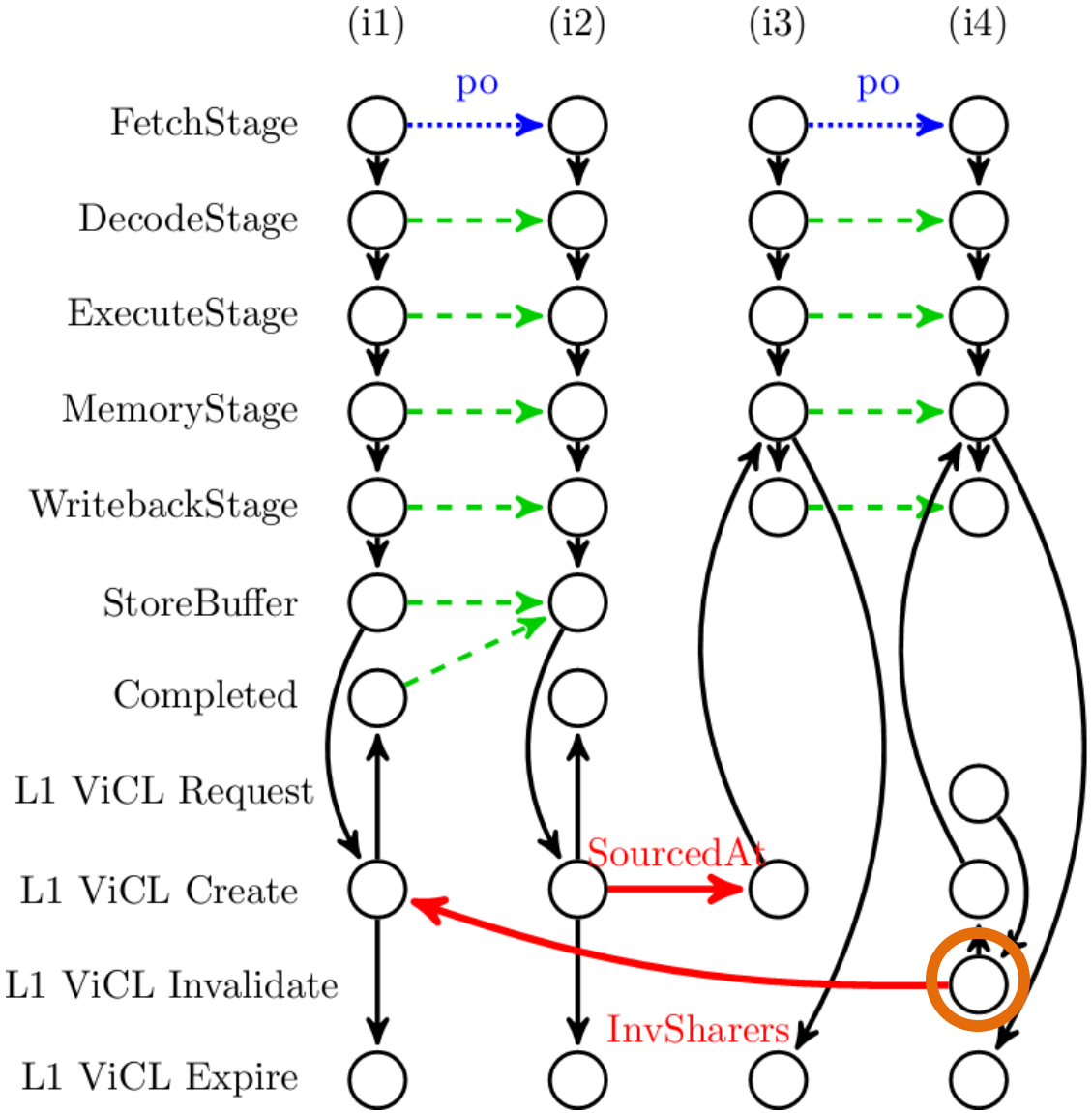
Case Studies and Results

“Peekaboo”



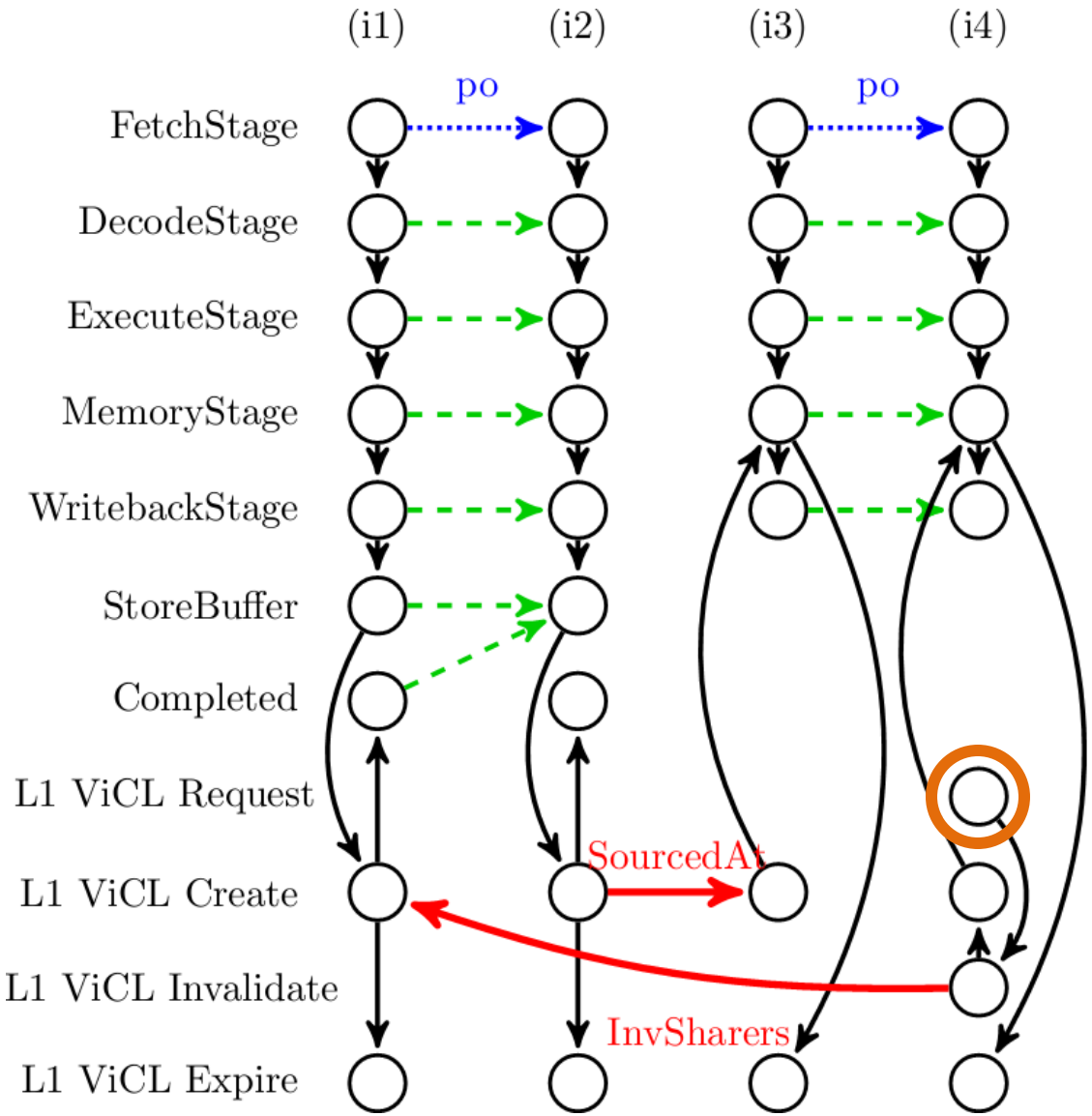
- Livelock prevention mechanism allows use of stale data
- “Peekaboo” edge completes cycle => outcome forbidden
- Consistency maintained

“Peekaboo”



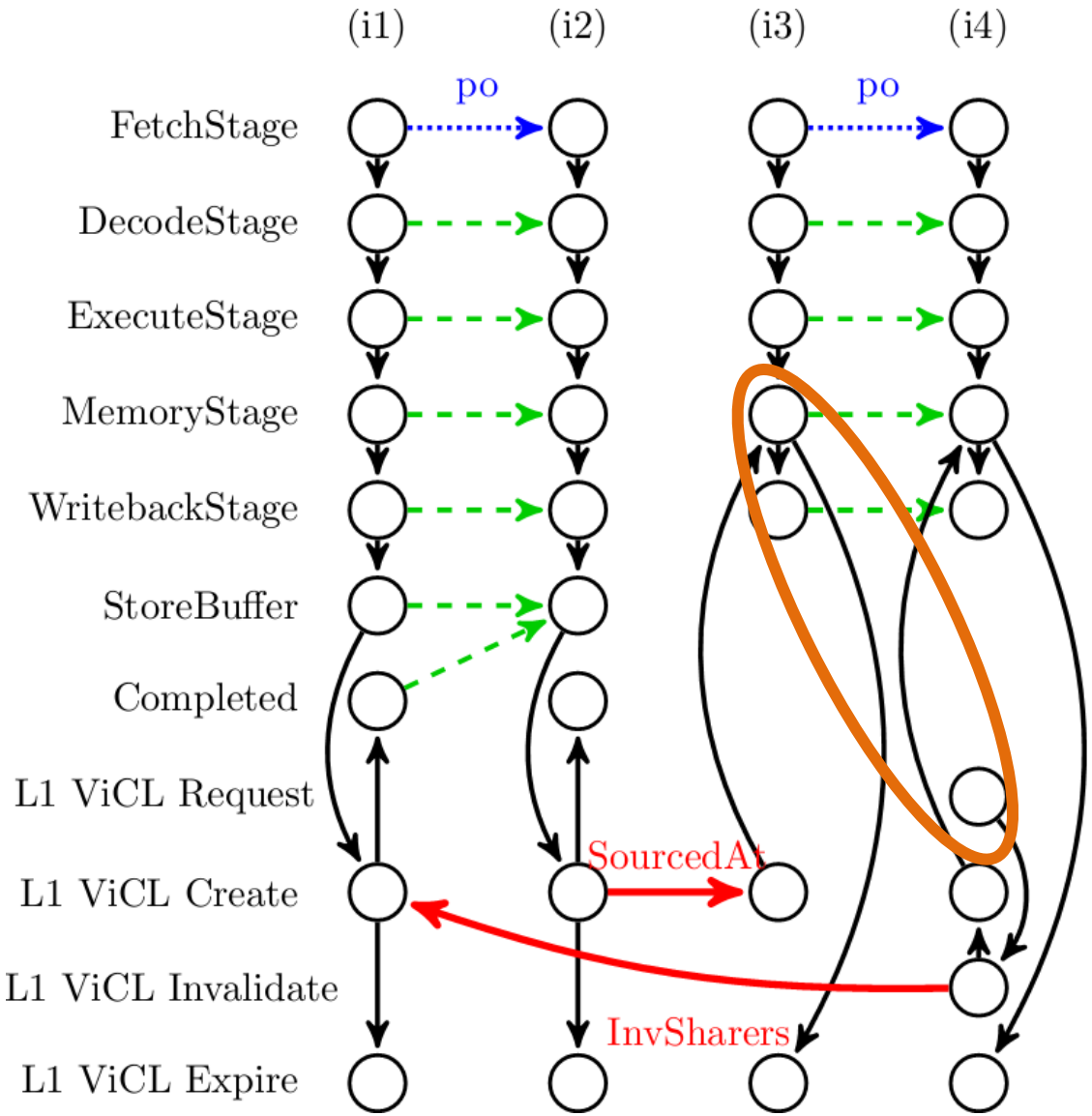
- Livelock prevention mechanism allows use of stale data
- “Peekaboo” edge completes cycle => outcome forbidden
- Consistency maintained

“Peekaboo”



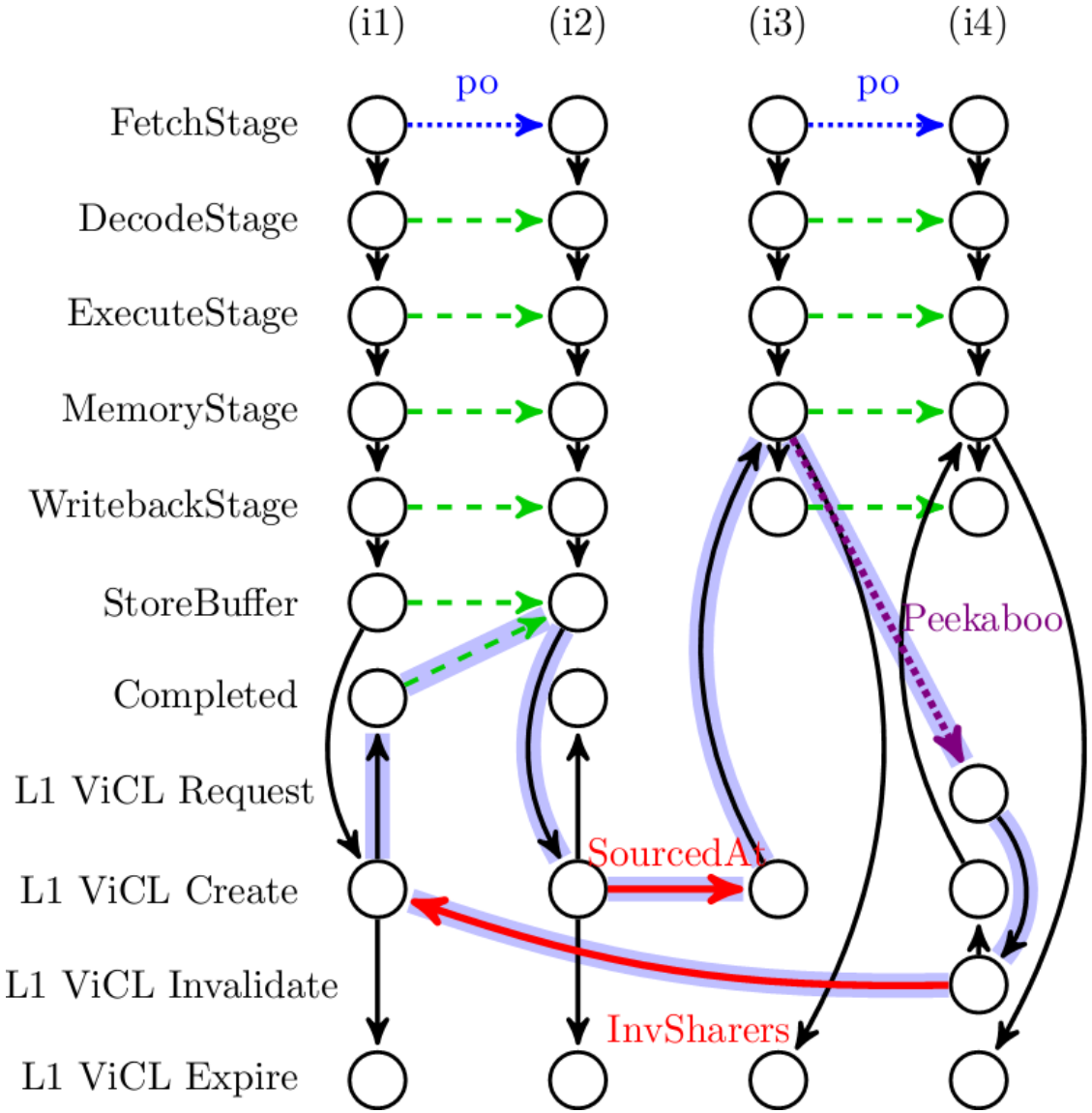
- Livelock prevention mechanism allows use of stale data
- “Peekaboo” edge completes cycle => outcome forbidden
- Consistency maintained

“Peekaboo”



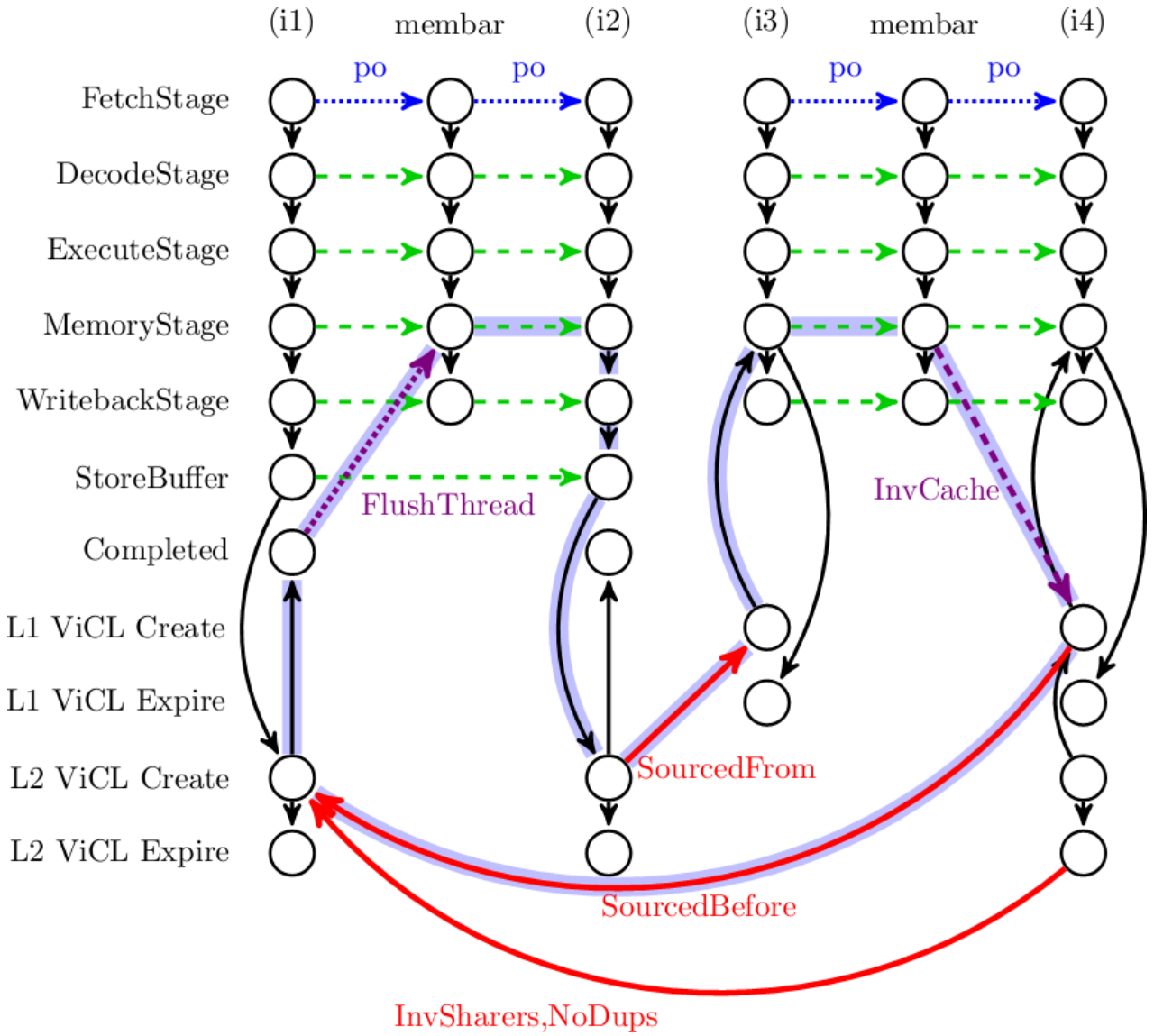
- Livelock prevention mechanism allows use of stale data
- “Peekaboo” edge completes cycle => outcome forbidden
- Consistency maintained

“Peekaboo”



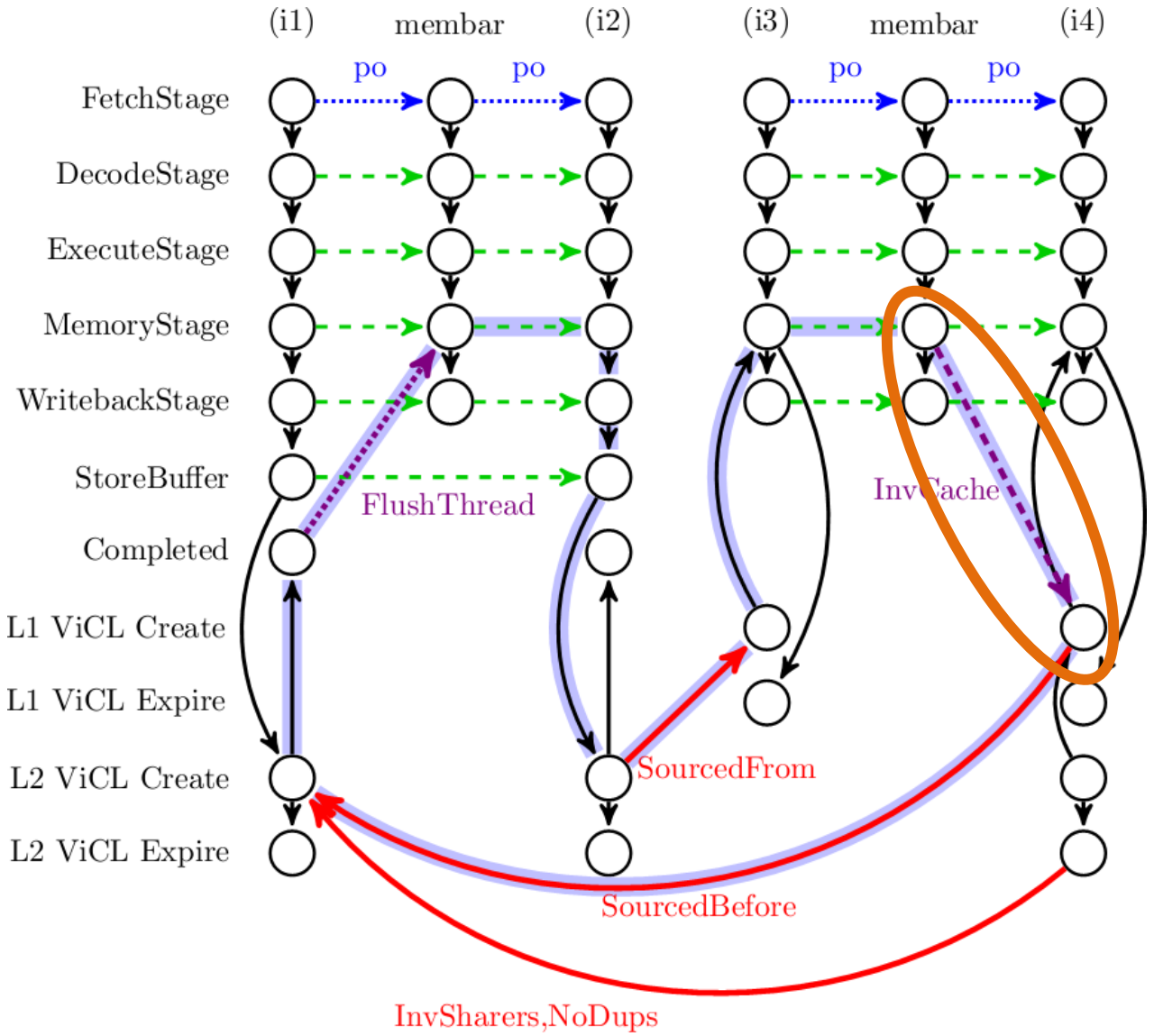
- Livelock prevention mechanism allows use of stale data
- “Peekaboo” edge completes cycle => outcome forbidden
- Consistency maintained

Partial Incoherence: GPUs



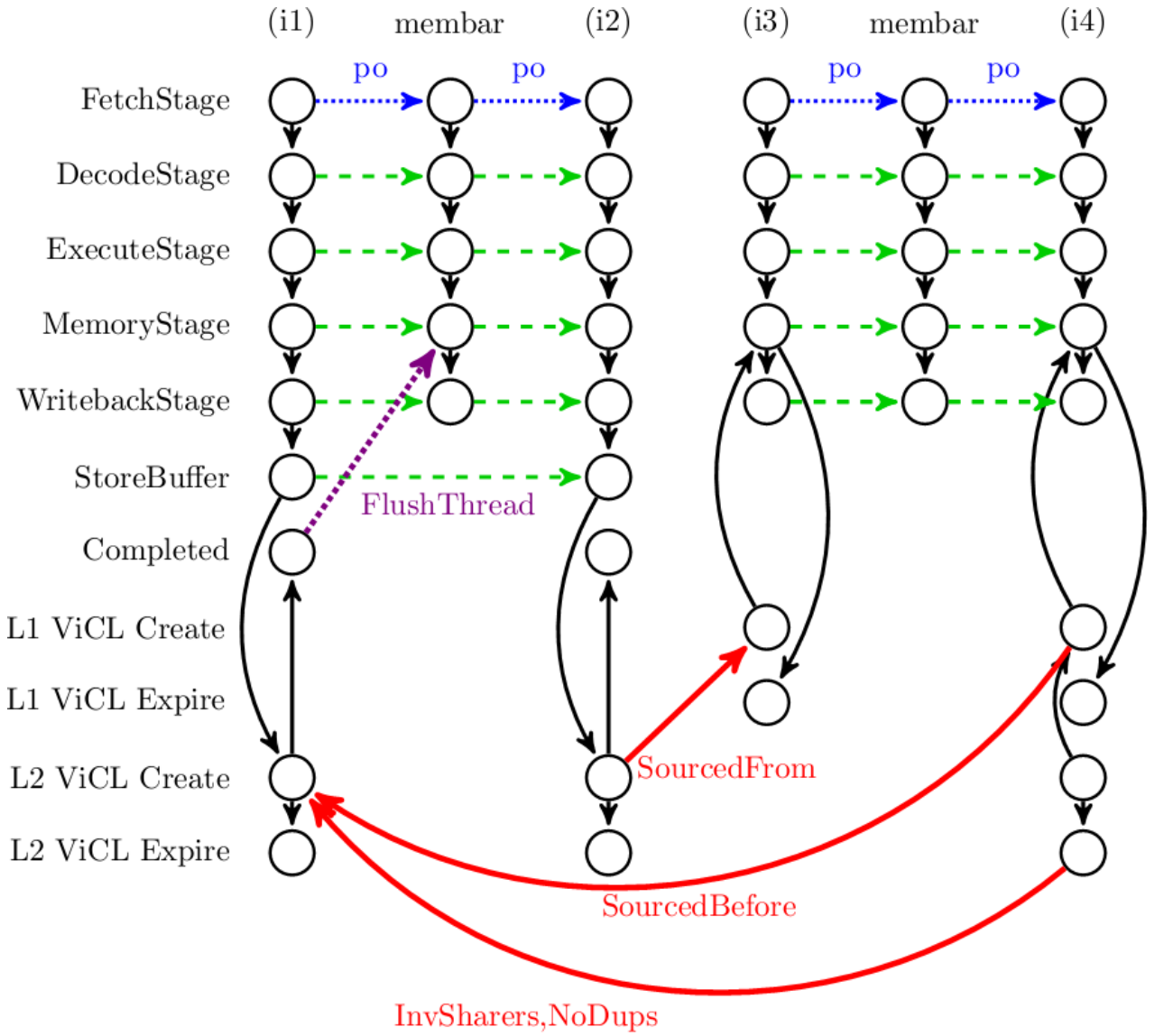
- e.g.: **`mp`** with membar fences [Alglave et al. ASPLOS15]
- If fence does not enforce **InvCache** ordering => no cycle

Partial Incoherence: GPUs



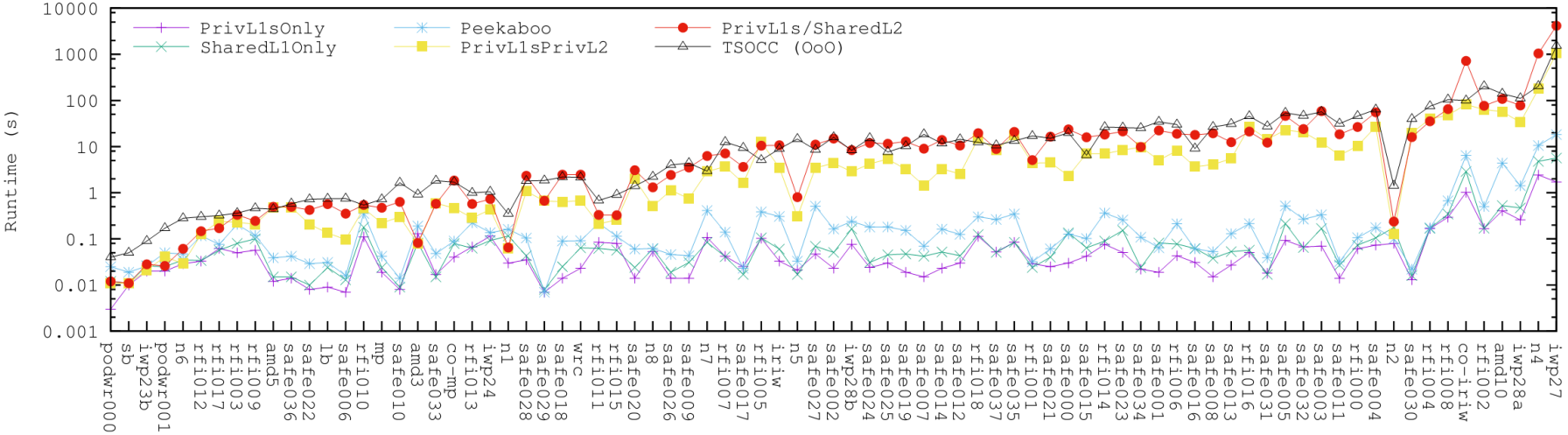
- e.g.: **`mp`** with membar fences [Alglave et al. ASPLOS15]
- If fence does not enforce **InvCache** ordering => no cycle

Partial Incoherence: GPUs



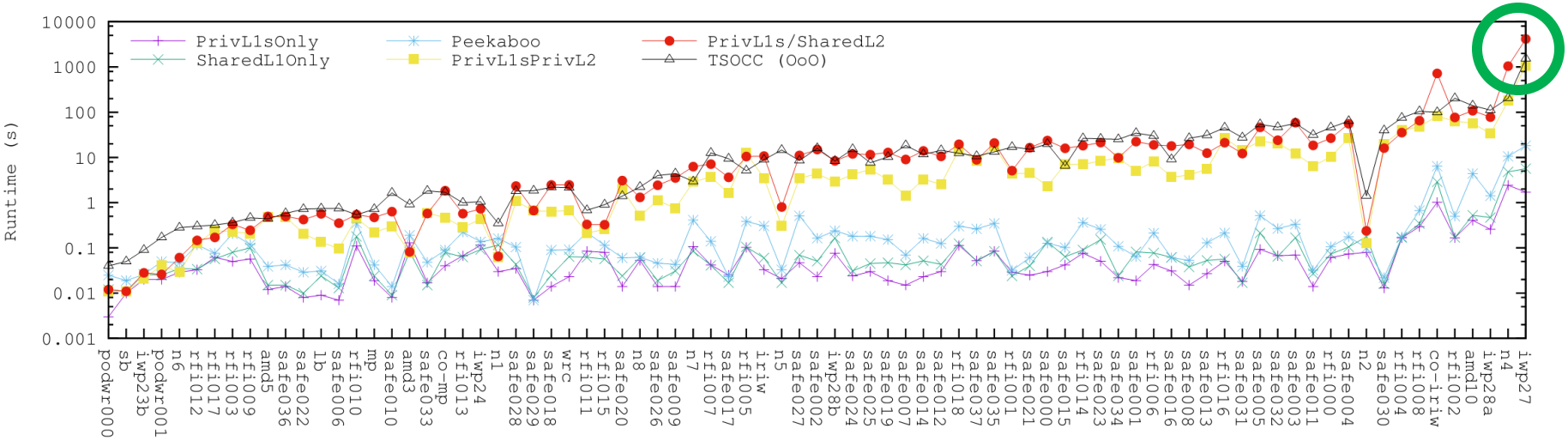
- e.g.: **`mp`** with membar fences [Alglave et al. ASPLOS15]
- If fence does not enforce **InvCache** ordering => no cycle

Verification Times



- Runtimes remain reasonable due to intelligent pruning and unsatisfiable constraint detection
- Subsequent research has used SMT solver-based techniques to run most tests in just **seconds**! [ASPLOS 2016]

Verification Times



- Runtimes remain reasonable due to intelligent pruning and unsatisfiable constraint detection
- Subsequent research has used SMT solver-based techniques to run most tests in just **seconds**! [ASPLOS 2016]

Conclusion

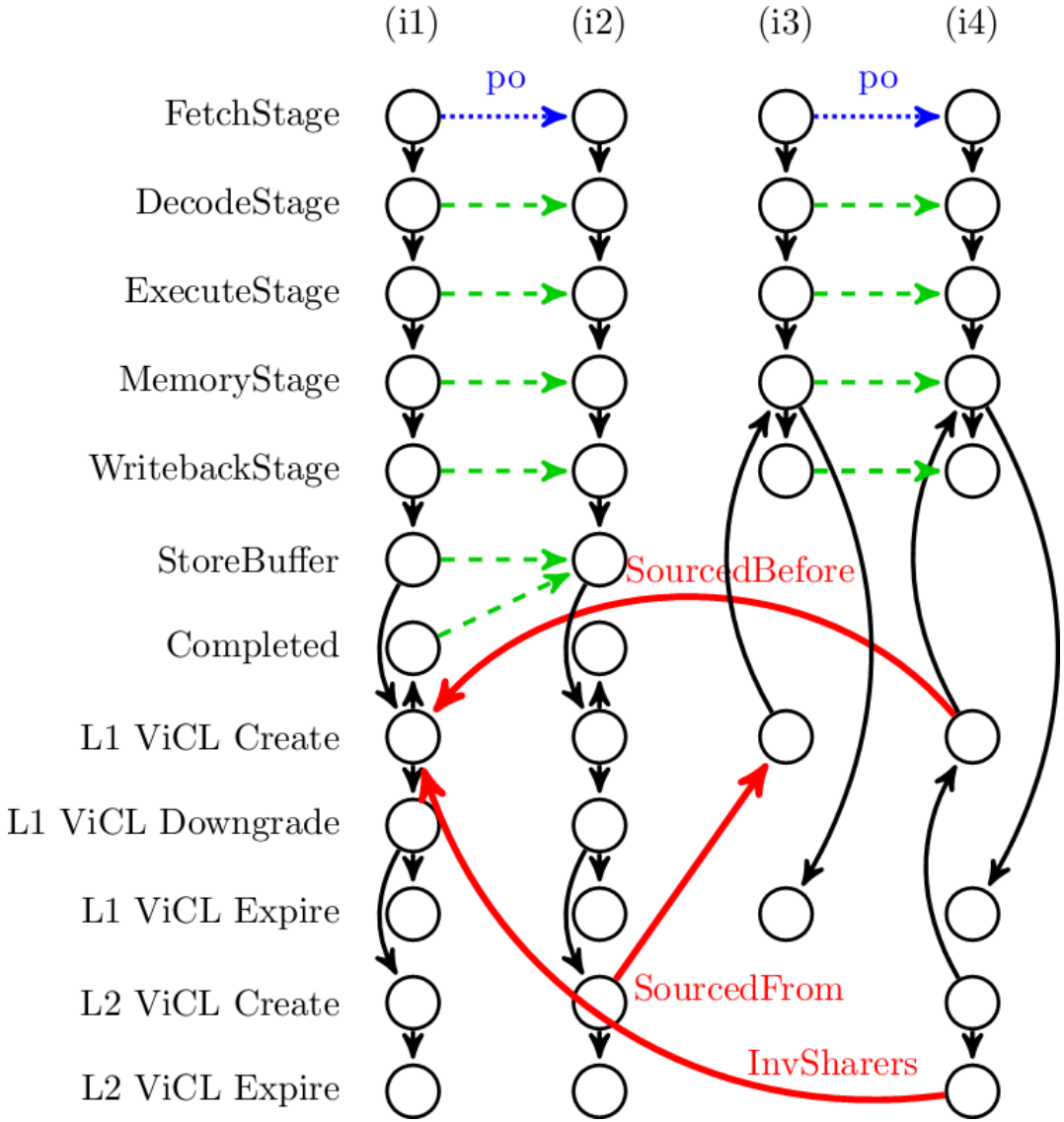
- CCI verification is critical to correct operation of complex parallel systems
- **CCICheck:** static CCI-aware microarchitectural consistency verification
  - Partial incoherence (GPUs), lazy coherence, and more!
- μhb graphs, ViCLs, and constraint-based enumeration
  - **Comprehensive** and **intuitive** μarch modelling
- Allows designers to build correct systems with greater ease and confidence

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

Yatin A. Manerkar, Daniel Lustig,
Michael Pellauer, and Margaret Martonosi

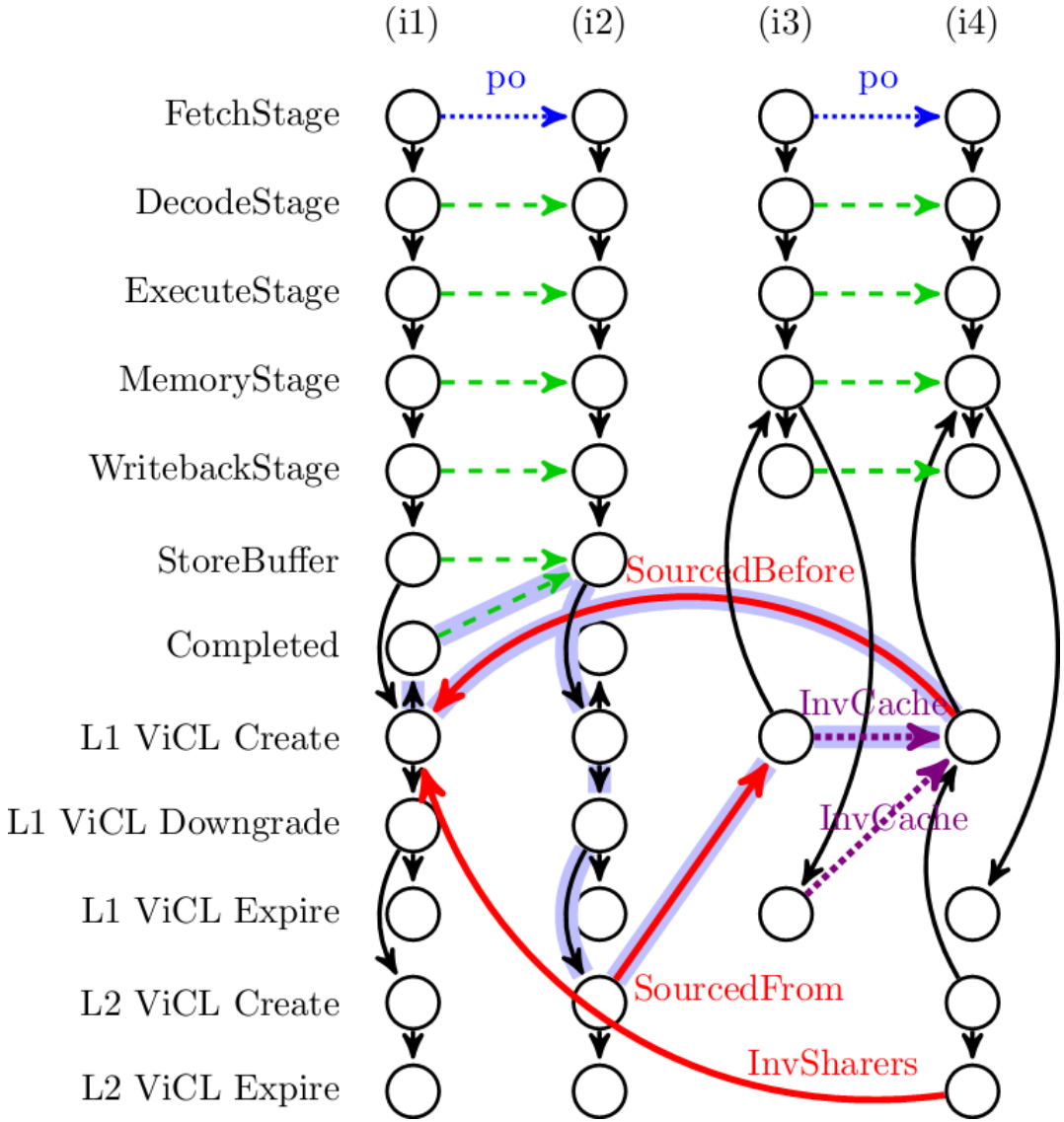
Code available at
https://github.com/ymanerka/ccicheck

Lazy Coherence (TSO-CC)



- No eager invalidation of sharers, but "InvCache" edges model the invalidation of a core's private cache on an L1 miss
- Thus, TSO is maintained

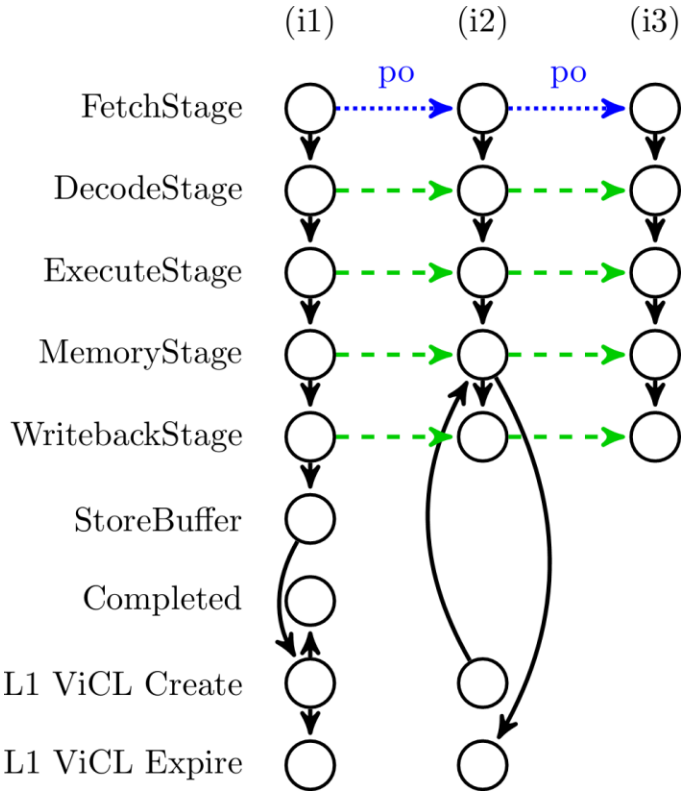
Lazy Coherence (TSO-CC)



- No eager invalidation of sharers, but "InvCache" edges model the invalidation of a core's private cache on an L1 miss
- Thus, TSO is maintained

Constraint-Based Enumeration

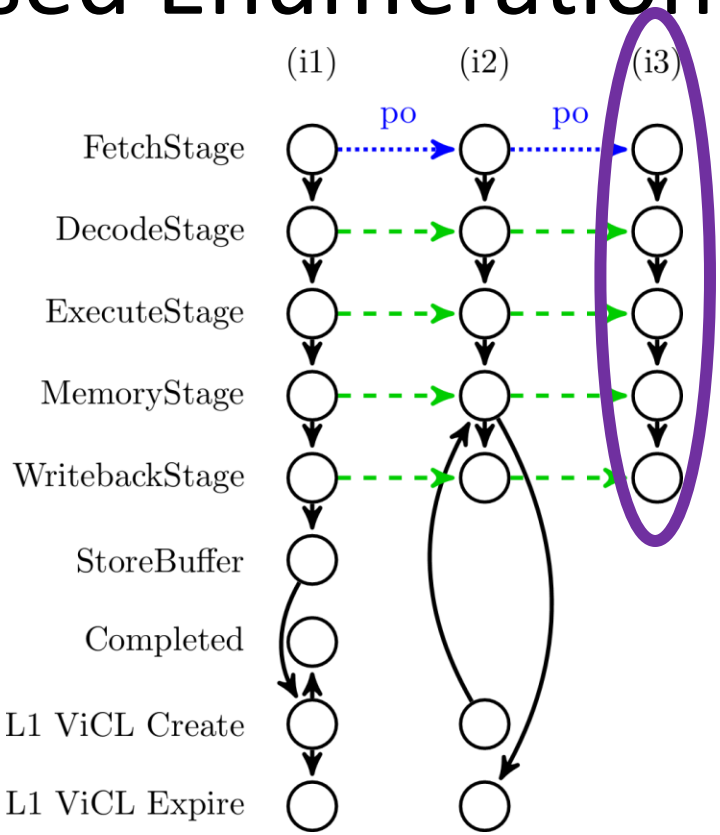
Thread	
(i1)	St [x] ← 1
(i2)	Ld r1 ← [x]
(i3)	Ld r2 ← [x]
Allowed: r1=1, r2=1	



- i3 needs a source for its value
- L1 ViCL with same address and data

Constraint-Based Enumeration

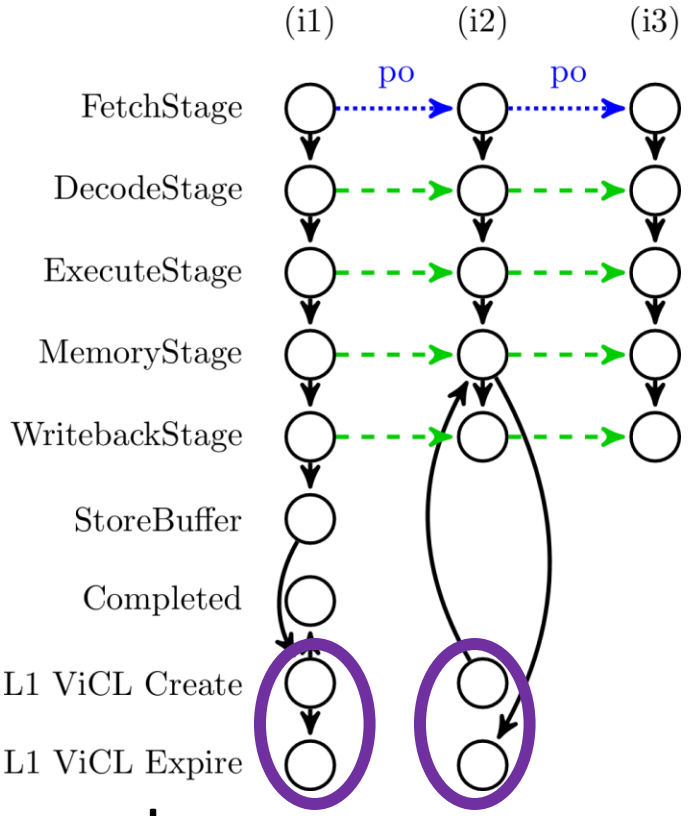
Thread	
(i1)	St [x] ← 1
(i2)	Ld r1 ← [x]
(i3)	Ld r2 ← [x]
Allowed: r1=1, r2=1	



- i3 needs a source for its value
- L1 ViCL with same address and data

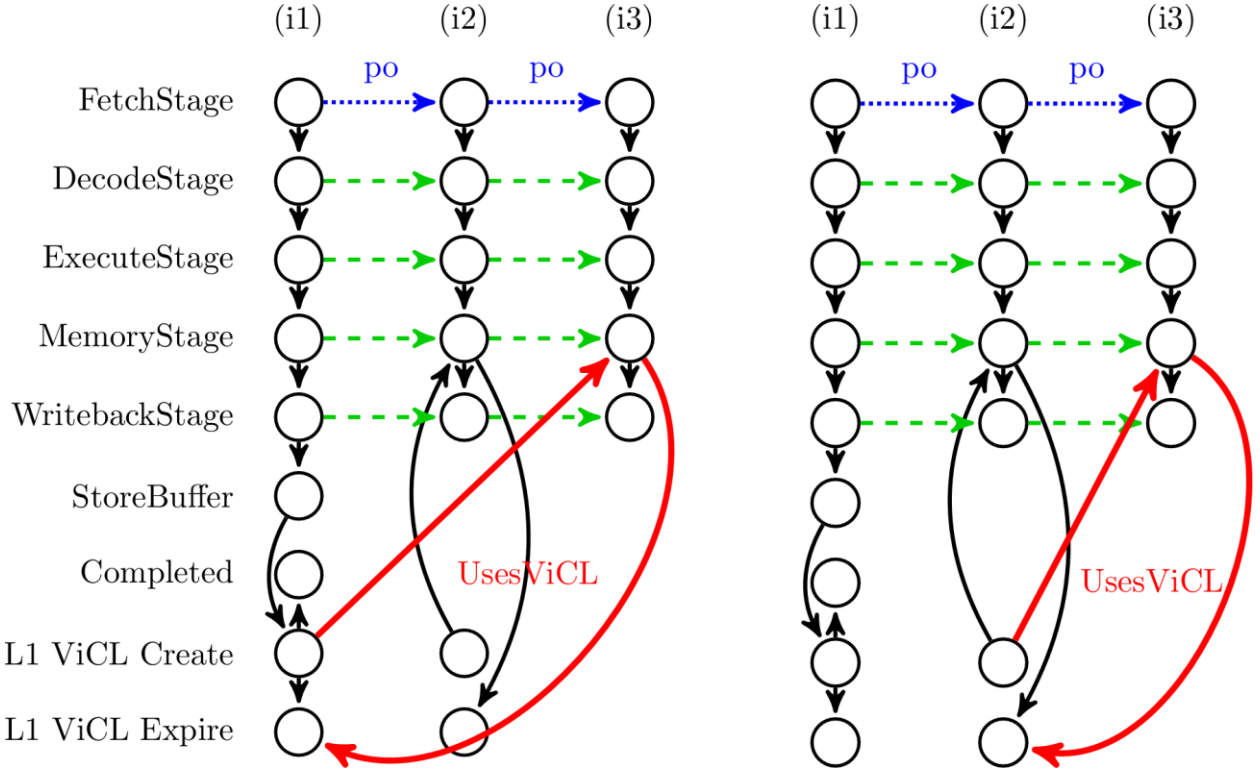
Constraint-Based Enumeration

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



- i3 needs a source for its value
- L1 ViCL with same address and data

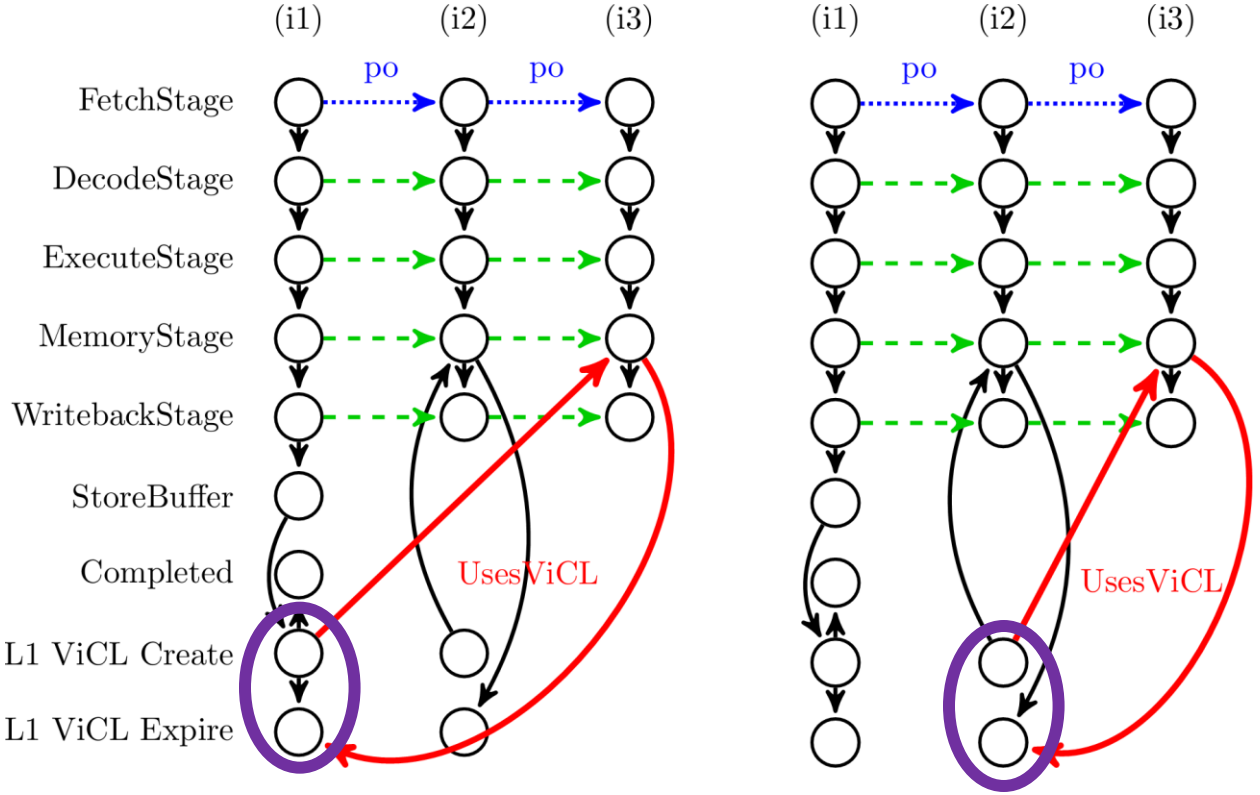
Constraint-Based Enumeration



- i3 needs a source for its value
- L1 ViCL with same address and data

=> Two possibilities enumerated.

Constraint-Based Enumeration



- i3 needs a source for its value
- L1 ViCL with same address and data

=> Two possibilities enumerated.