vCXLGen

Automated Synthesis and Verification of CXL Bridges for Heterogeneous Architectures

Anatole Lefort*, Julian Pritzi*, Nicolò Carpentieri, David Schall,
Simon Dittrich, Soham Chakraborty, Nicolai Oswald, Pramod Bhatotia
TU Delft NVIDIA

Systems Research Group
https://dse.in.tum.de/
Technical University of Munich

*contributed equally

Heterogeneous compute in modern data centers

Heterogeneous compute in modern data centers

- **Increasing CPU diversity:**

Heterogeneous compute in modern data centers

- **Increasing CPU diversity:**

x86 used to be the norm

Heterogeneous compute in modern data centers

- **Increasing CPU diversity:**

x86 used to be the norm

AMD EPYC

ARM is catching up

arm

Heterogeneous compute in modern data centers

- **Increasing CPU diversity:**

x86 used to be the norm

AMD EPYC

ARM is catching up

arm

Server-grade **RISC-V** CPUs are being launched

Heterogeneous compute in modern data centers

- **Increasing CPU diversity:**

x86 used to be the norm

ARM is catching up

Server-grade **RISC-V** CPUs are being launched

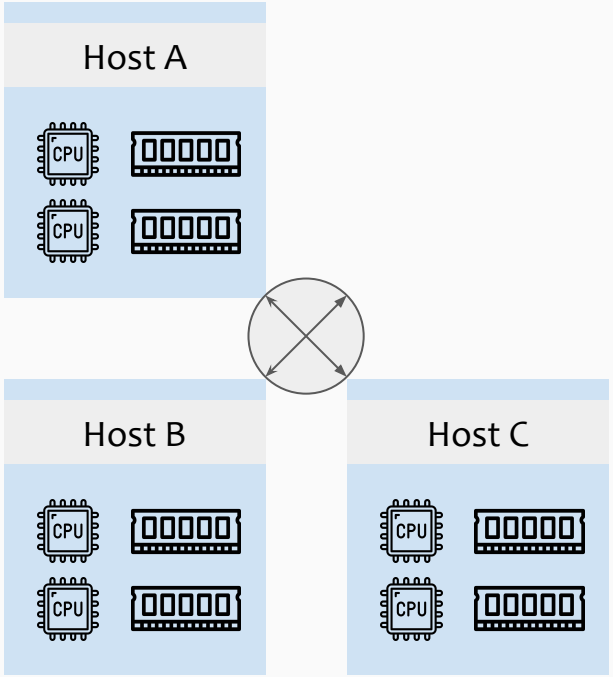
Major players develop their own silicon

Building scalable data centers

With resource disaggregation

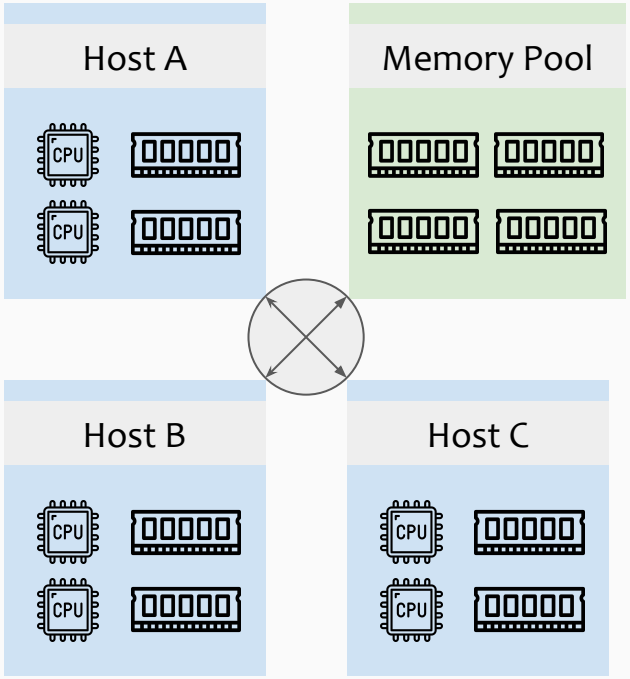
Building scalable data centers

With resource disaggregation



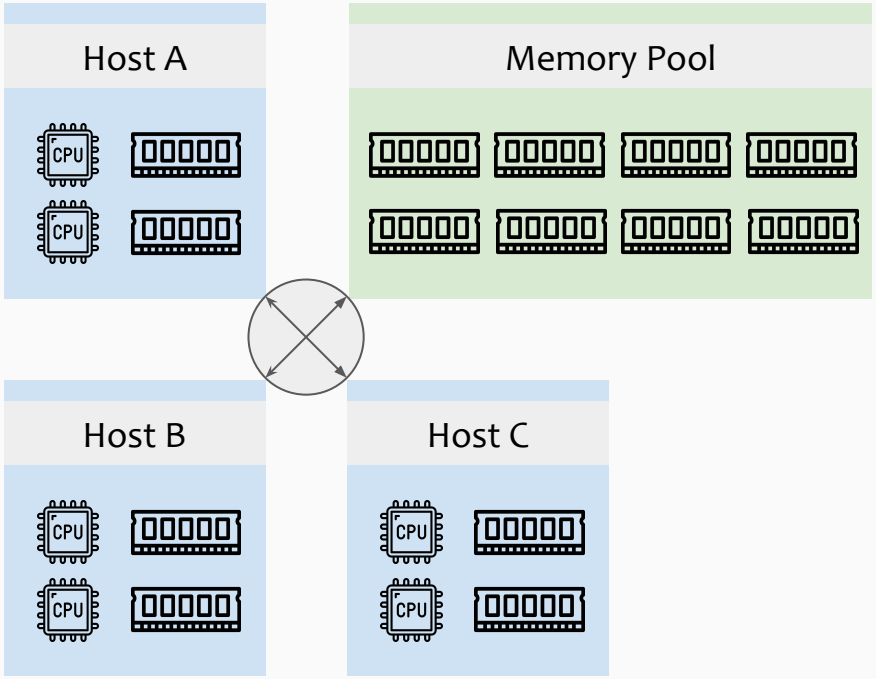
Building scalable data centers

With resource disaggregation



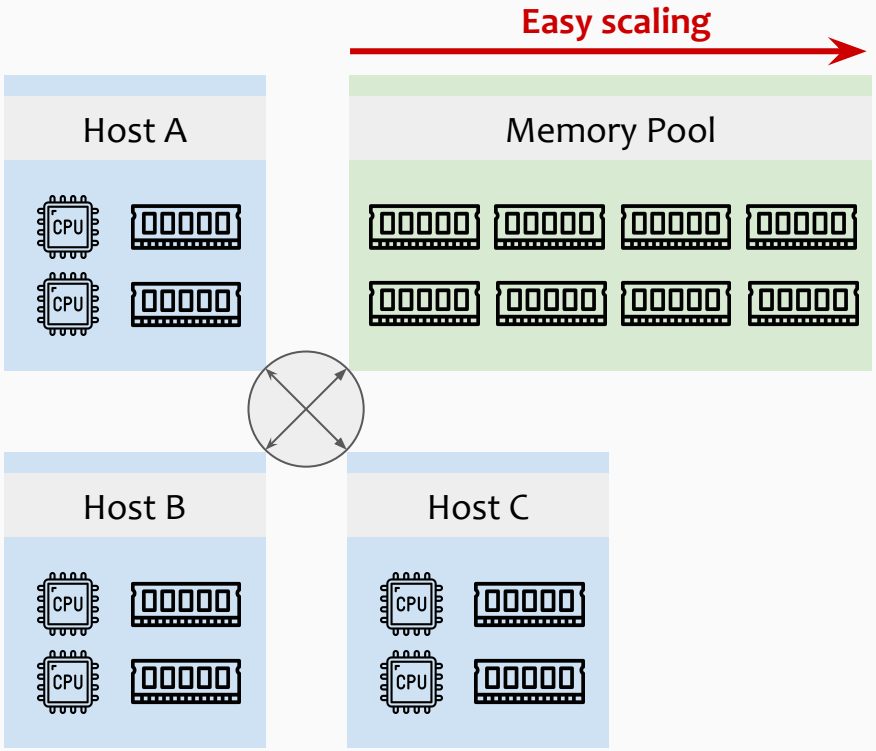
Building scalable data centers

With resource disaggregation



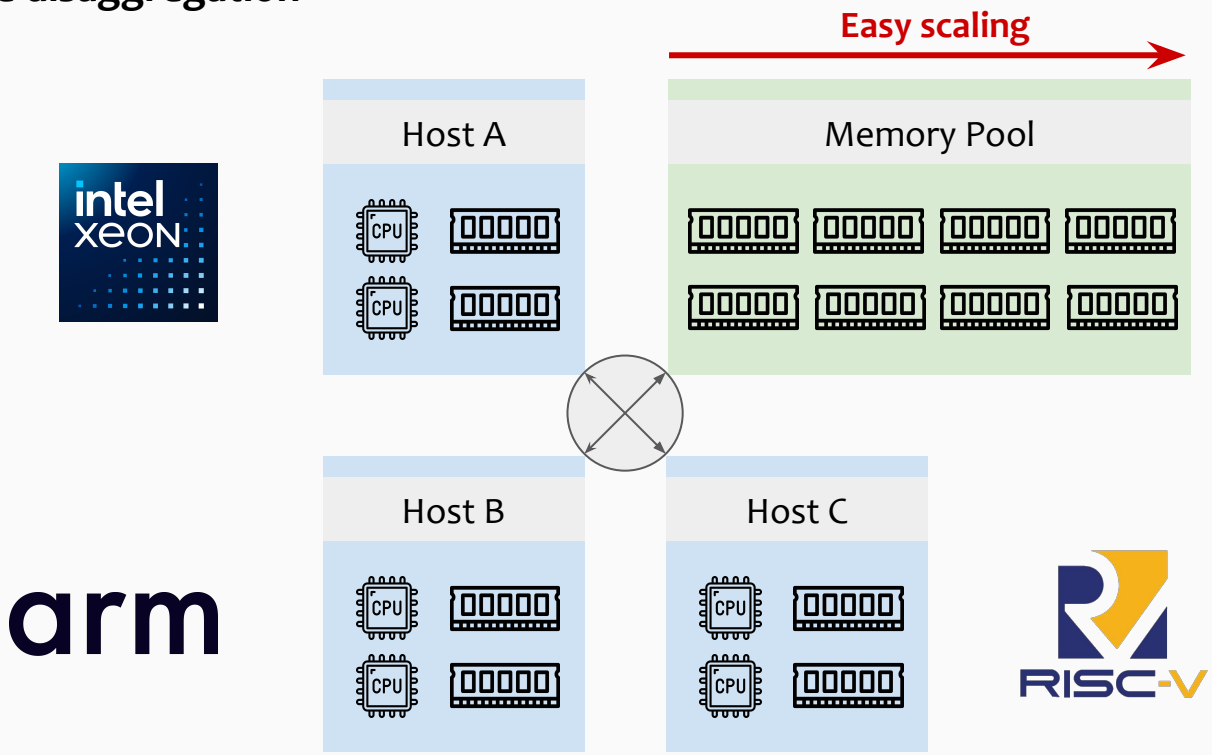
Building scalable data centers

With resource disaggregation



Building scalable data centers

With resource disaggregation



The CXL promise

To modern data centers (DCs)

The CXL promise

To modern data centers (DCs)

- **CXL —** a **standard** for **(remote) memory interconnect**

The CXL promise

To modern data centers (DCs)

- **CXL —** a **standard** for **(remote) memory interconnect**
  - Implemented in hardware
  - *CPU-to-* | *device* | *memory* | *accelerator*

The CXL promise

To modern data centers (DCs)

- **CXL —** a **standard** for **(remote) memory interconnect**
  - Implemented in hardware
  - *CPU-to-* | *device* | *memory* | *accelerator*
  - CXL 3.0 – Multi-host coherent memory

The CXL promise

To modern data centers (DCs)

- **CXL —** a **standard** for **(remote) memory interconnect**
  - Implemented in hardware
  - *CPU-to-* | *device* | *memory* | *accelerator*
  - CXL 3.0 – Multi-host coherent memory

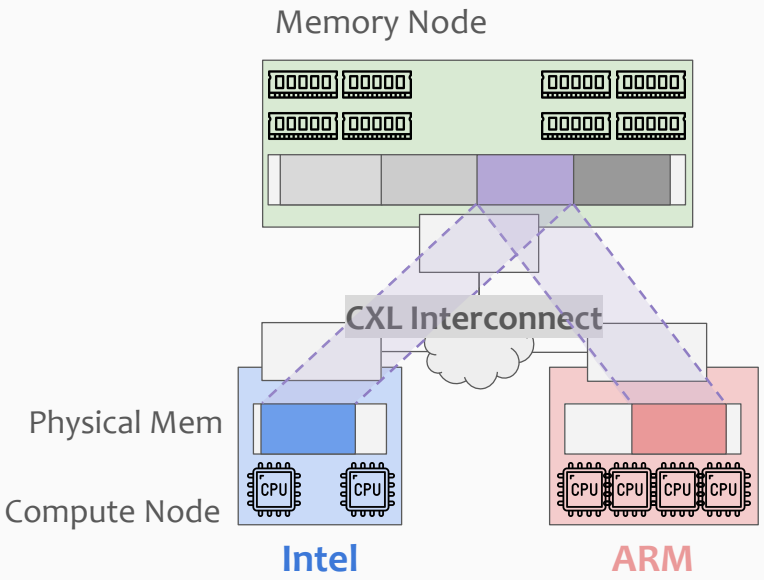
Can CXL compose **heterogeneous architectures ?**

The problem: CXL & heterogeneous DCs

Modern DCs mix CPUs — but CXL lacks interoperability guarantees

The problem: CXL & heterogeneous DCs

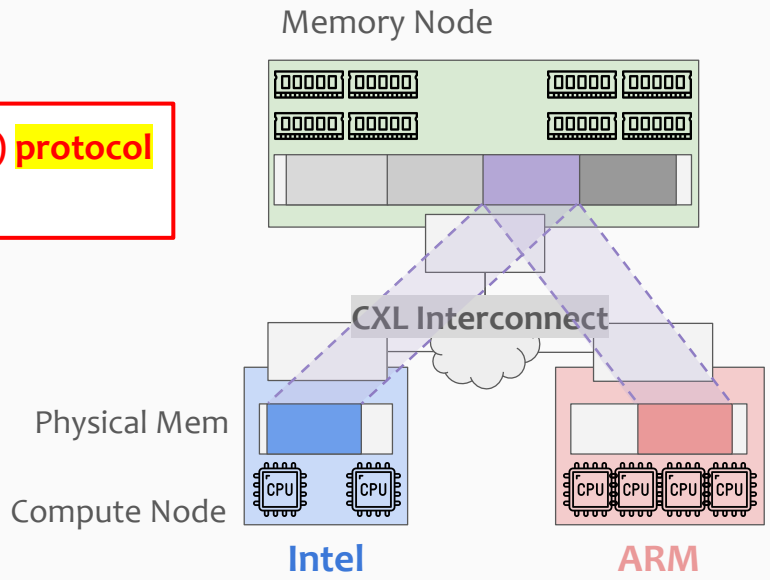
Modern DCs mix CPUs — but CXL lacks interoperability guarantees



The problem: CXL & heterogeneous DCs

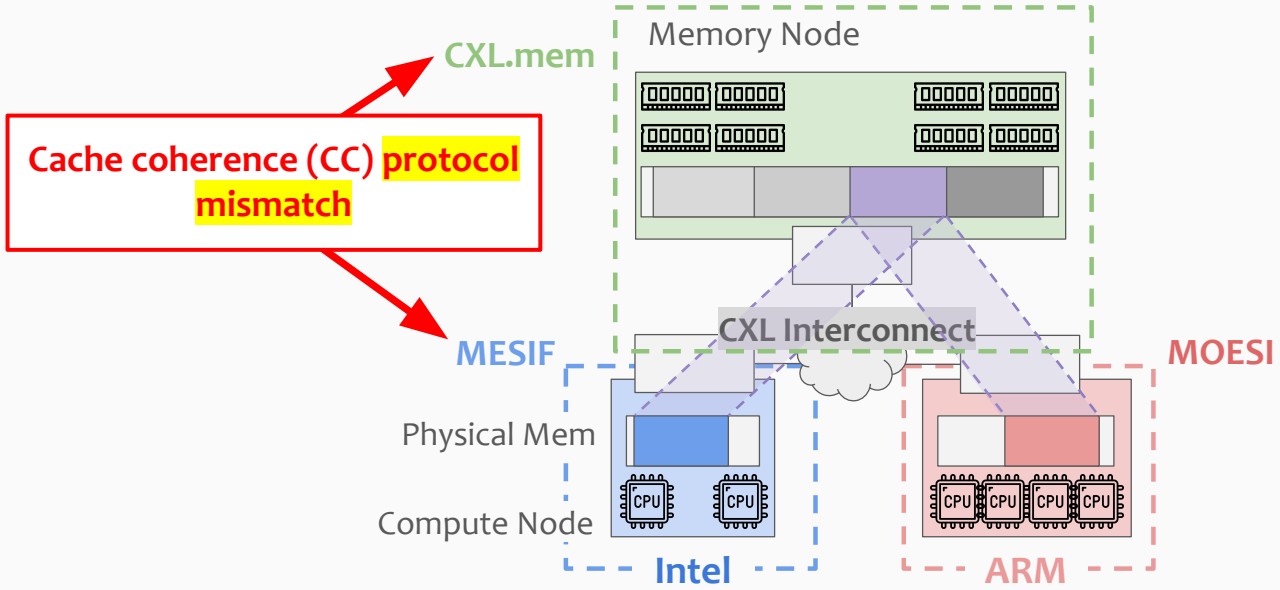
Modern DCs mix CPUs — but CXL lacks interoperability guarantees

Cache coherence (CC) protocol mismatch



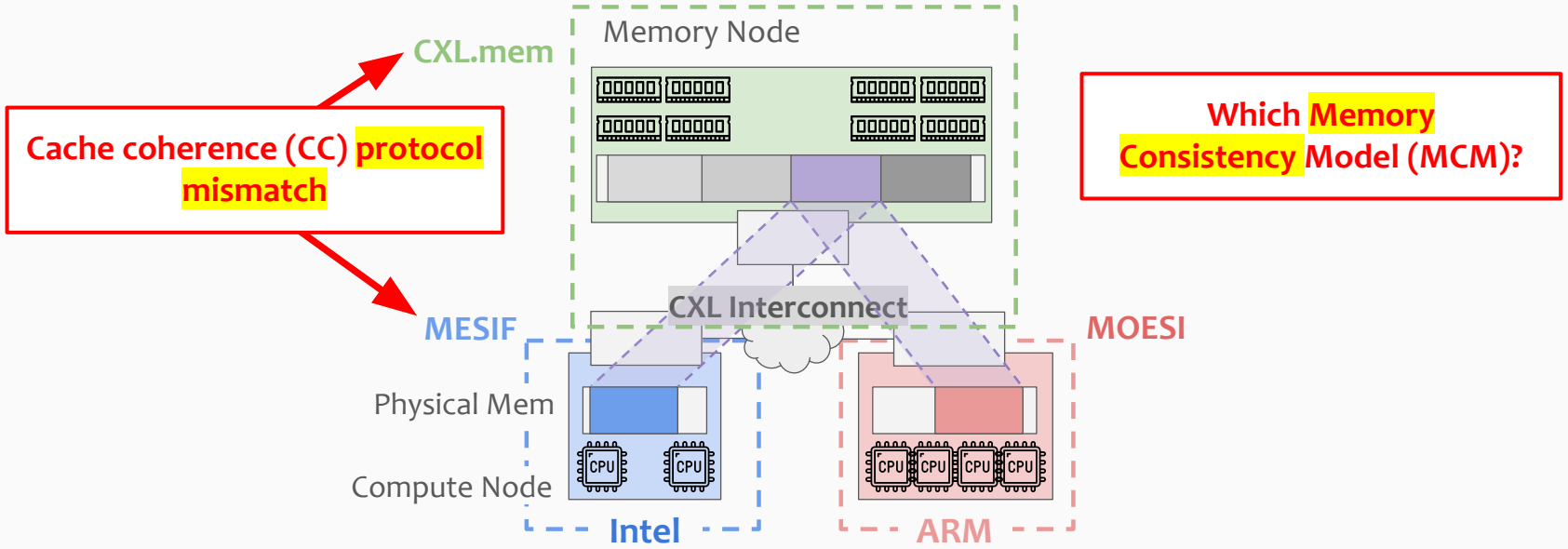
The problem: CXL & heterogeneous DCs

Modern DCs mix CPUs — but CXL lacks interoperability guarantees



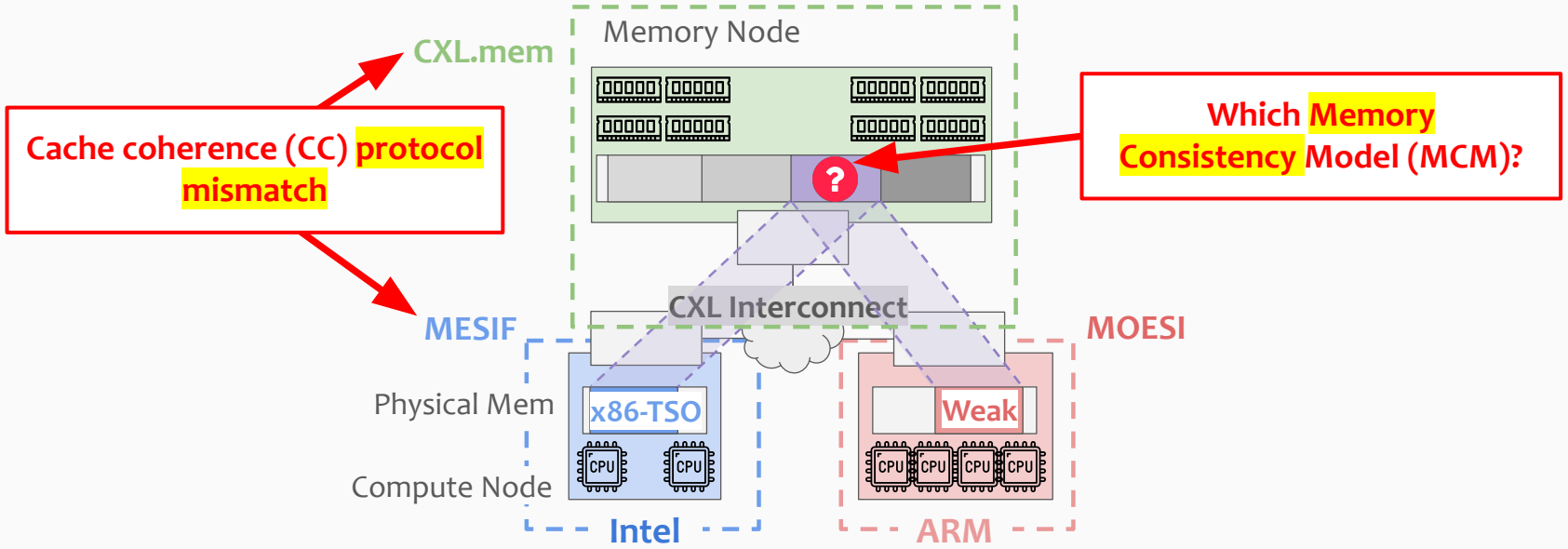
The problem: CXL & heterogeneous DCs

Modern DCs mix CPUs — but CXL lacks interoperability guarantees



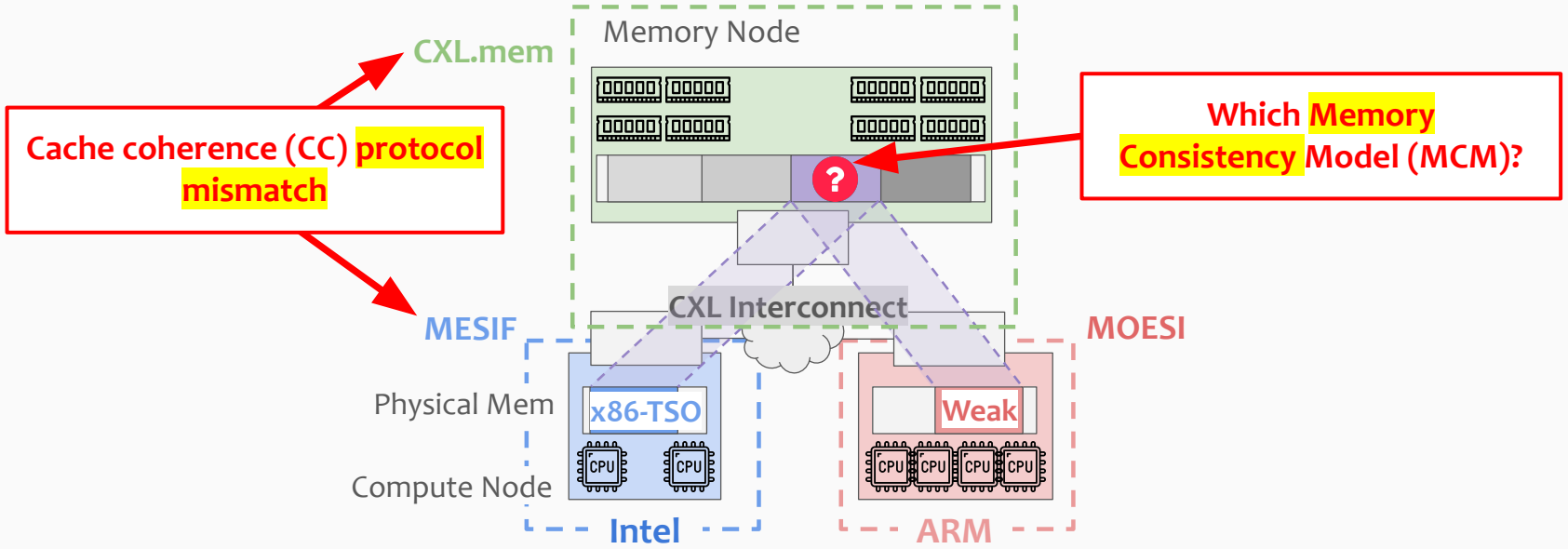
The problem: CXL & heterogeneous DCs

Modern DCs mix CPUs — but CXL lacks interoperability guarantees



The problem: CXL & heterogeneous DCs

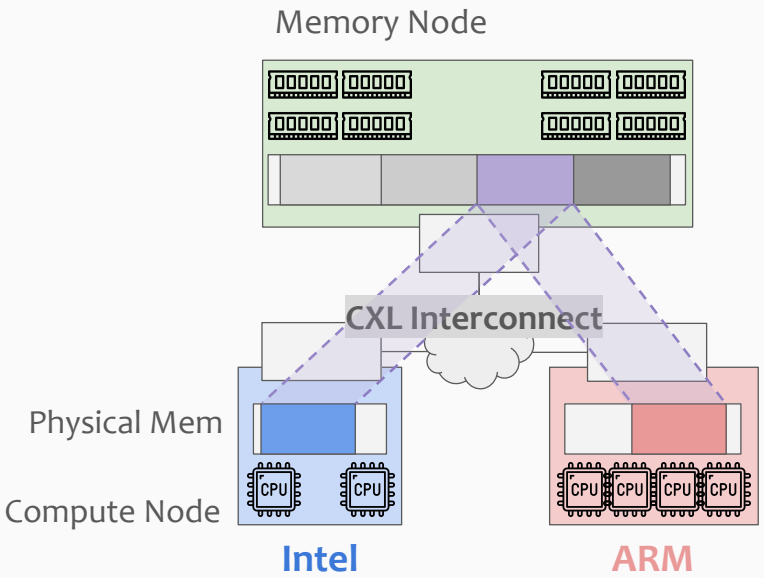
Modern DCs mix CPUs — but CXL lacks interoperability guarantees



No, CXL does not support heterogeneous architectures

The answer: CXL Bridges[1]

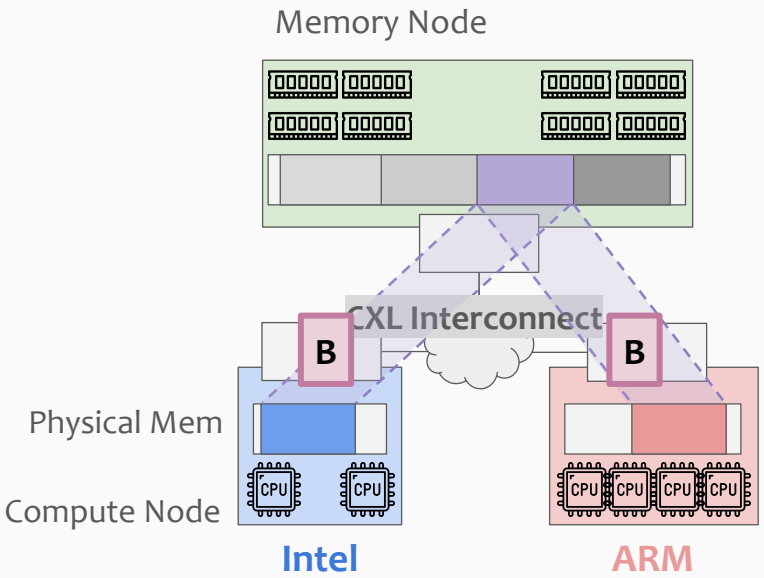
To overcome the **semantic gap** created by **architecture heterogeneity** with **CXL**



[1] *C³: CXL Coherence Controllers for Heterogeneous Architectures*, Lefort et al., HPCA'26

The answer: CXL Bridges[1]

To overcome the **semantic gap** created by **architecture heterogeneity** with **CXL**



[1] *C³: CXL Coherence Controllers for Heterogeneous Architectures*, Lefort et al., HPCA'26

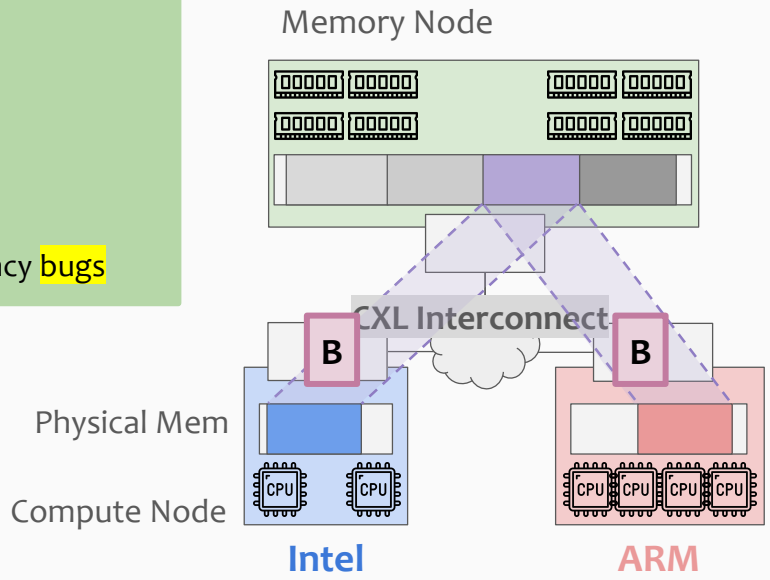
The answer: CXL Bridges[1]

To overcome the **semantic gap** created by **architecture heterogeneity** with **CXL**

Intuition:

1. Translate *MESIF* / *MOESI* to ***"talk"*** with CXL.mem

2. Avoid memory consistency bugs



[1] *C³: CXL Coherence Controllers for Heterogeneous Architectures*, Lefort et al., HPCA'26

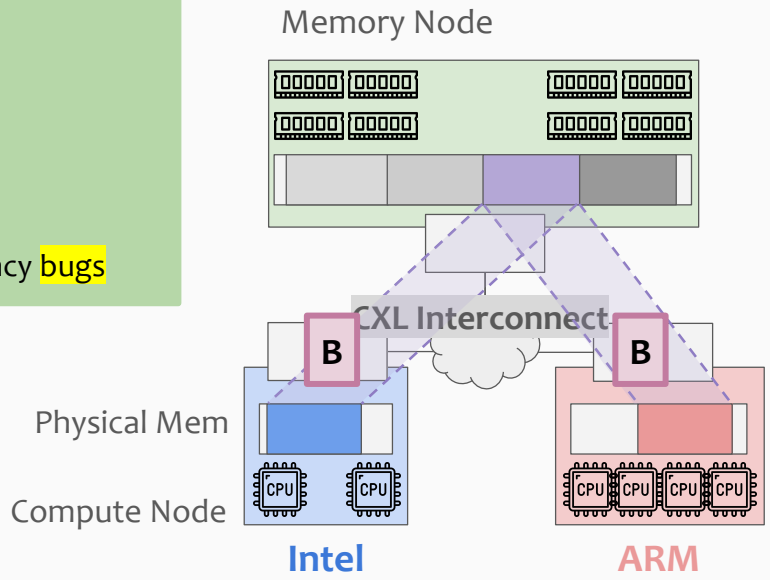
The answer: CXL Bridges[1]

To overcome the **semantic gap** created by **architecture heterogeneity** with **CXL**

Intuition:

1. Translate *MESIF* / *MOESI* to ***"talk"*** with CXL.mem

2. Avoid memory consistency bugs



No guidance from **CXL specifications**

[1] *C³: CXL Coherence Controllers for Heterogeneous Architectures*, Lefort et al., HPCA'26

Challenges: Why are CXL Bridges hard**?**

Challenges: Why are CXL Bridges hard**?**

Hardware vendors are **burdened** with **CXL bridge design**

Challenges: Why are CXL Bridges hard?

Hardware vendors are **burdened** with **CXL bridge design**

1

Semantic gap

Manual & ad-hoc

requests translation

Challenges: Why are CXL Bridges hard?

Hardware vendors are **burdened** with **CXL bridge design**

1 Semantic gap

Manual & ad-hoc
requests translation

2 Complexity

Large state machines
(~40 states, ~250 transitions)

Challenges: Why are CXL Bridges hard?

Hardware vendors are **burdened** with **CXL bridge design**

1 Semantic gap

Manual & ad-hoc
requests translation

2 Complexity

Large state machines
(~40 states, ~250 transitions)

3 Diversity

Many vendor-specific **coherence protocols** & **MCMs**

Challenges: Why are CXL Bridges hard?

Hardware vendors are **burdened** with **CXL bridge design**

1 Semantic gap

Manual & ad-hoc
requests translation

2 Complexity

Large state machines
(~40 states, ~250 transitions)

3 Diversity

Many vendor-specific **coherence protocols** & **MCMs**

Problem Statement: How to systematically and correctly design CXL bridges for heterogeneous architectures?

Proposal: vCXLGen

Solution: Automatic synthesis of "*correct-by-construction*" CXL bridges from coherence protocol specifications for heterogeneous architectures

Proposal: vCXLGen

Solution: Automatic synthesis of "*correct-by-construction*" CXL bridges from coherence protocol specifications for heterogeneous architectures

Design Goals:

Proposal: vCXLGen

Solution: Automatic synthesis of "*correct-by-construction*" CXL bridges from coherence protocol specifications for heterogeneous architectures

Design Goals:

1 Full Automation

Synthesize correct CXL **translations** from specifications

Proposal: vCXLGen

Solution: Automatic synthesis of "*correct-by-construction*" CXL bridges from coherence protocol specifications for heterogeneous architectures

Design Goals:

1 Full Automation

Synthesize correct CXL **translations** from specifications

2 Correctness

Preserve **original MCMs** *by construction*

Proposal: vCXLGen

Solution: Automatic synthesis of "*correct-by-construction*" CXL bridges from coherence protocol specifications for heterogeneous architectures

Design Goals:

1 Full Automation

Synthesize correct CXL **translations** from specifications

2 Correctness

Preserve **original MCMs** *by construction*

3 Genericity

Applicable to any (**existing** and **upcoming**) architectures

Overview: **The vCXLGen** bridge generator

Overview: **The vCXLGen** bridge generator

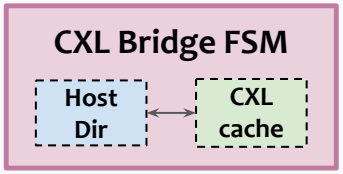
Insight:

CXL Bridge: *compounding of finite state machines from* Host & CXL protocols

Overview: **The vCXLGen** bridge generator

Insight:

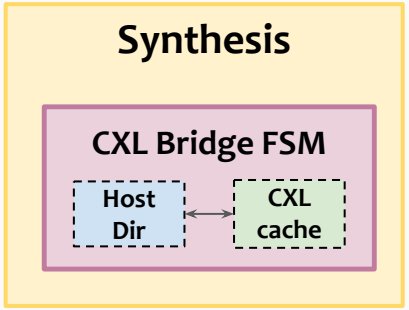
CXL Bridge: *compounding of finite state machines from* Host & CXL protocols



Overview: **The vCXLGen** bridge generator

Insight:

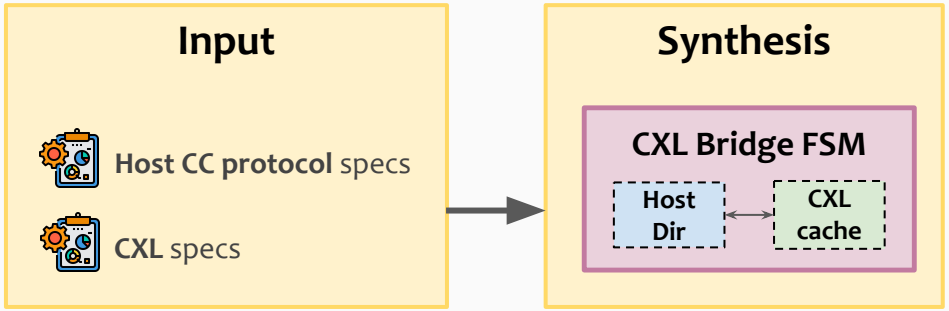
CXL Bridge: *compounding of finite state machines from* Host & CXL protocols



Overview: **The vCXLGen** bridge generator

Insight:

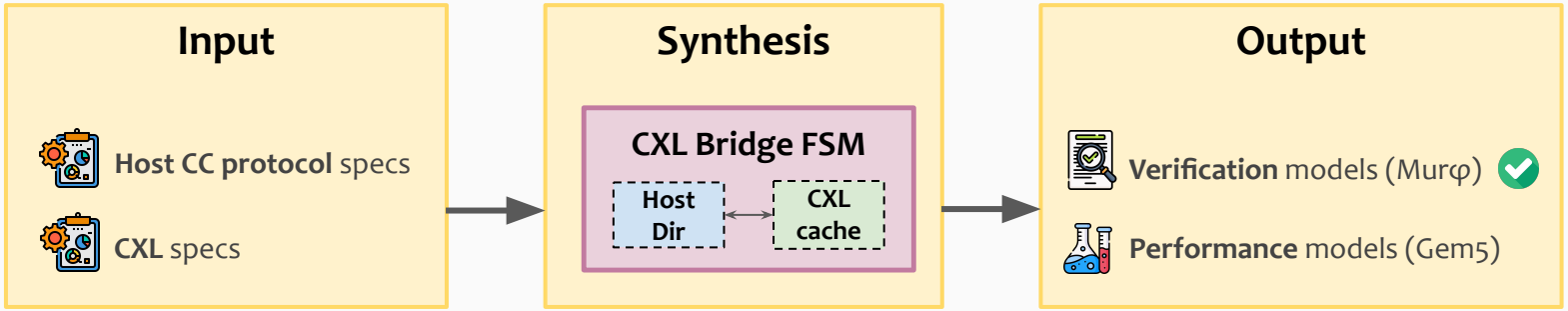
CXL Bridge: *compounding of finite state machines from* Host & CXL protocols



Overview: **The vCXLGen** bridge generator

Insight:

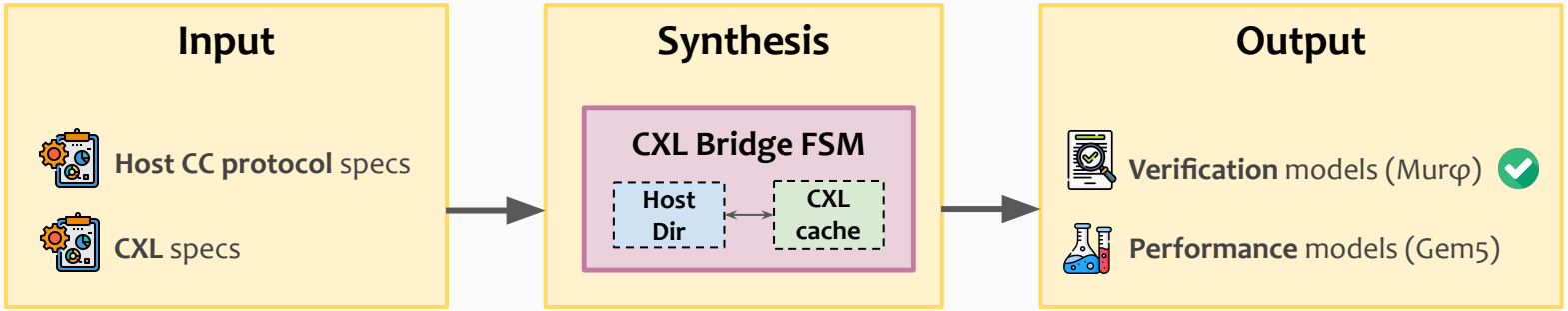
CXL Bridge: *compounding of finite state machines from* Host & CXL protocols



Overview: **The vCXLGen** bridge generator

Insight:

CXL Bridge: *compounding of finite state machines from* Host & CXL protocols

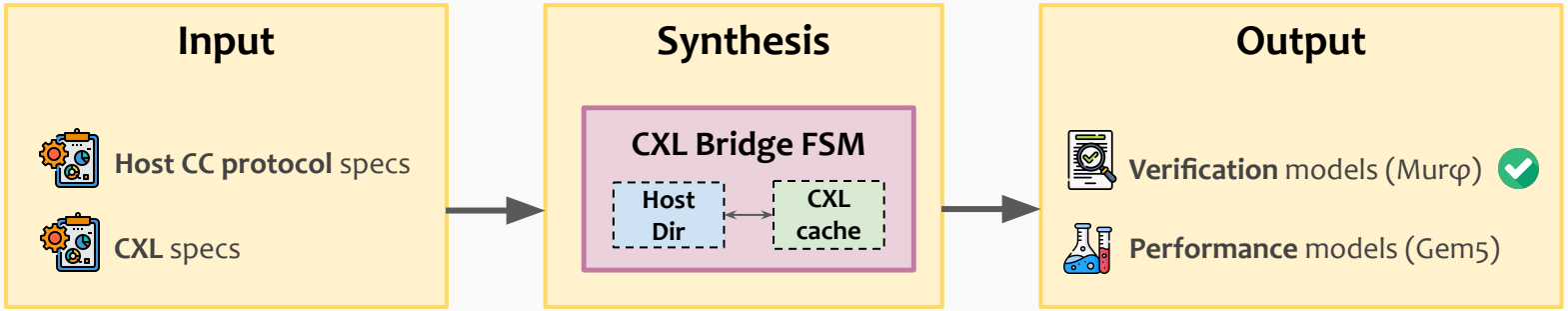


Contributions:

Overview: **The vCXLGen** bridge generator

Insight:

CXL Bridge: *compounding of finite state machines from* Host & CXL protocols



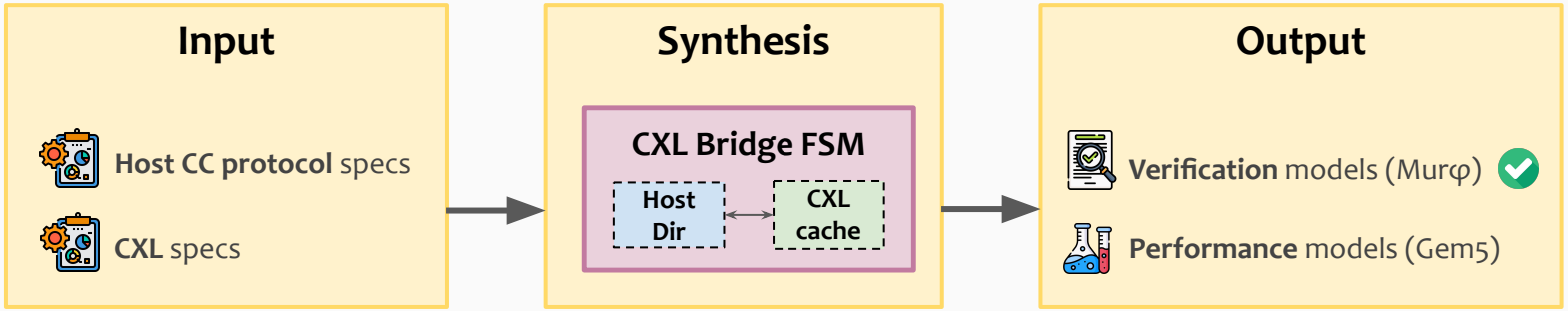
Contributions:

Generic synthesis algorithm
for CXL bridges

Overview: **The vCXLGen** bridge generator

Insight:

CXL Bridge: *compounding of finite state machines from* Host & CXL protocols

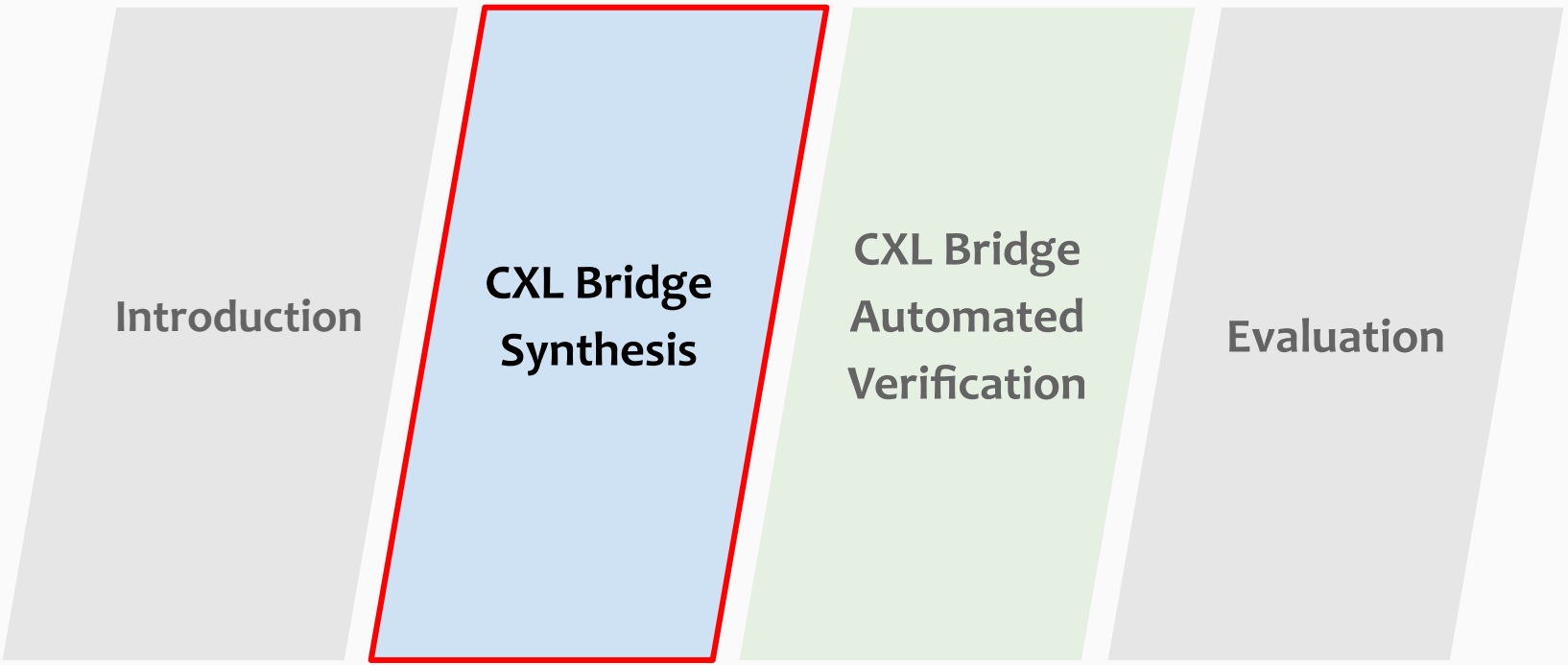


Contributions:

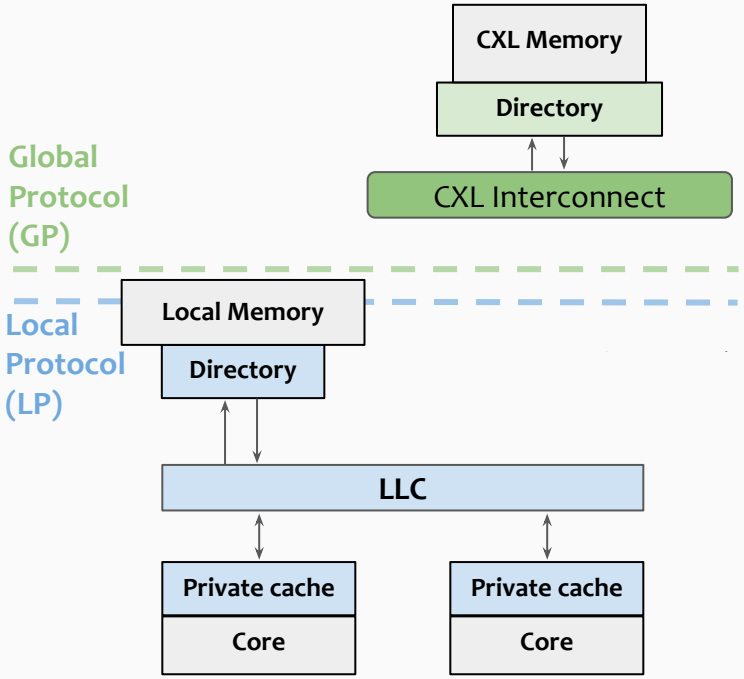
Generic synthesis algorithm for CXL bridges

Automated correctness verification for CXL bridges

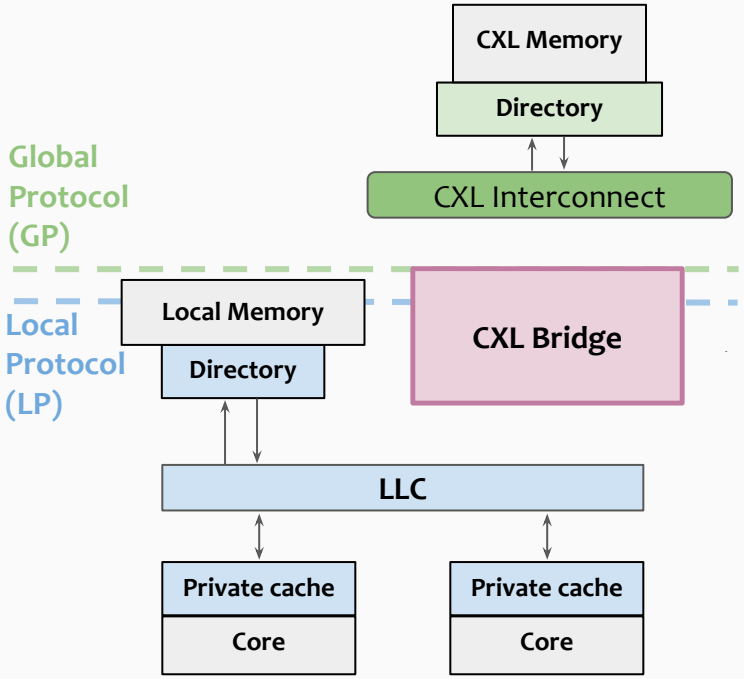
Outline


Introduction
CXL Bridge Synthesis
CXL Bridge Automated Verification
Evaluation

Architecture: **CXL cache coherence bridges**

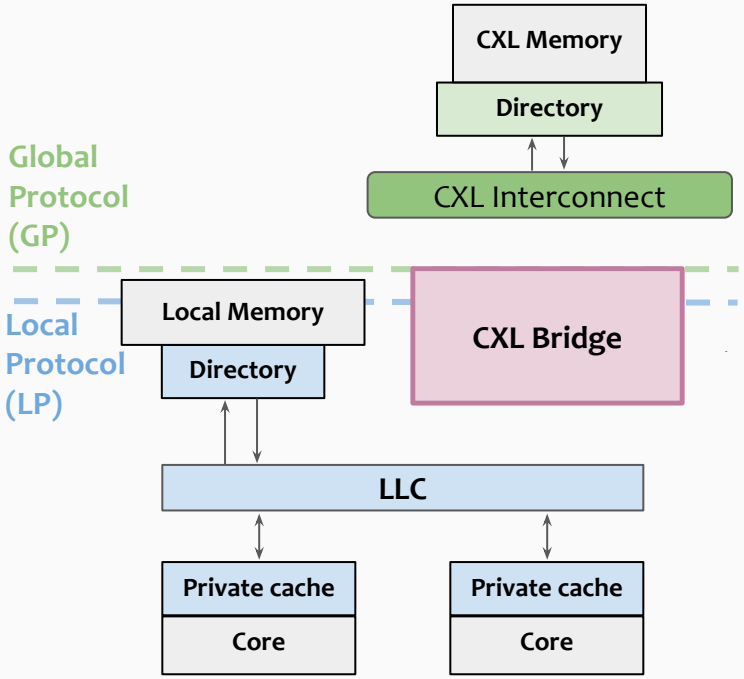


Architecture: **CXL cache coherence bridges**



Architecture: **CXL cache coherence bridges**

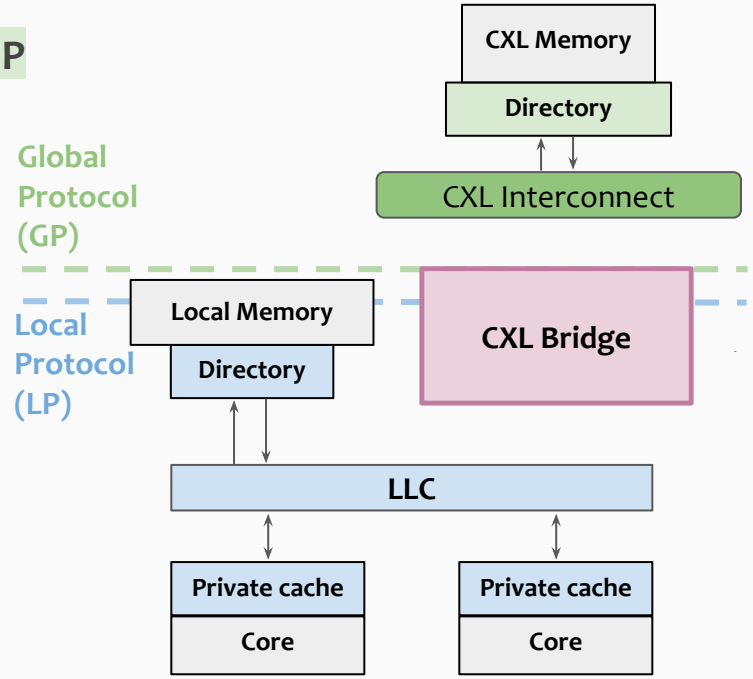
Abstraction: CXL bridges sit at the interface between hosts and CXL



Architecture: **CXL cache coherence bridges**

Abstraction: CXL bridges sit at the interface between hosts and CXL

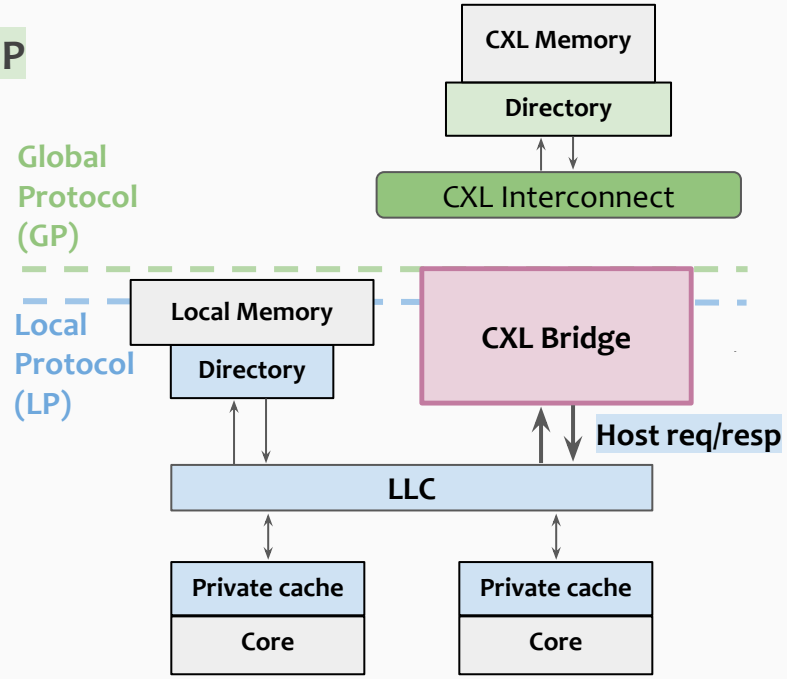
Role: *Propagate* & *Translate* requests between **LP** and **GP**



Architecture: **CXL cache coherence bridges**

Abstraction: CXL bridges sit at the interface between hosts and CXL

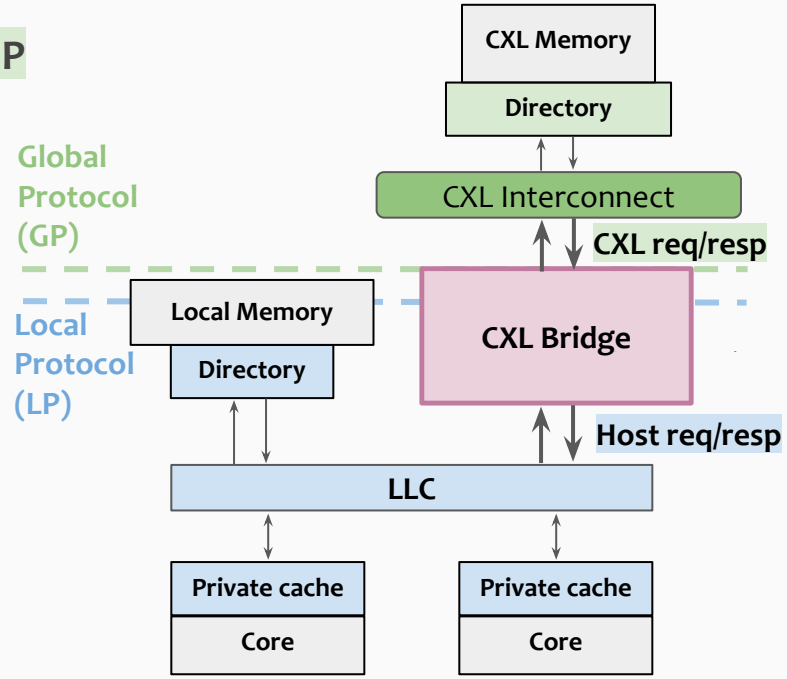
Role: *Propagate* & *Translate* requests between **LP** and **GP**



Architecture: **CXL cache coherence bridges**

Abstraction: CXL bridges sit at the interface between hosts and CXL

Role: *Propagate* & *Translate* requests between **LP** and **GP**

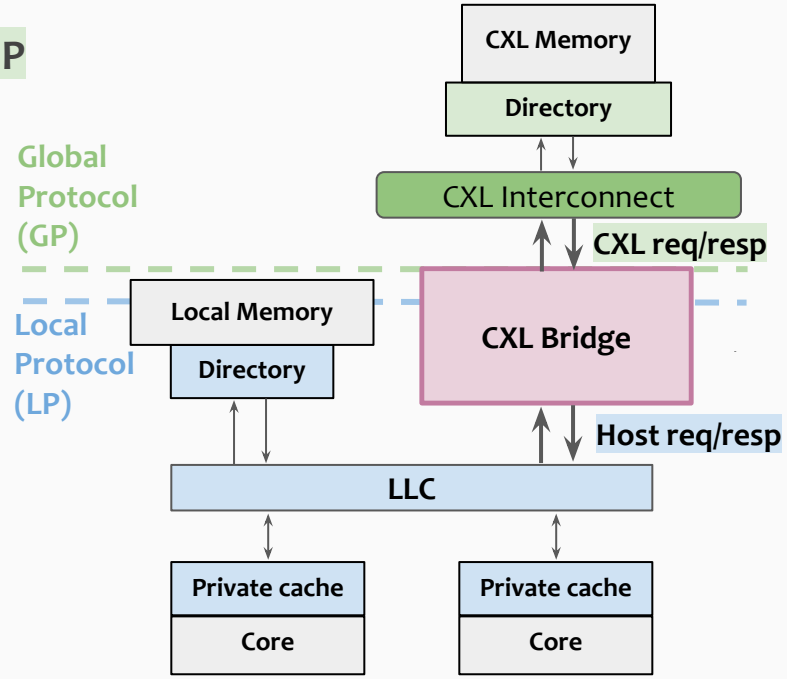


Architecture: **CXL cache coherence bridges**

Abstraction: CXL bridges sit at the interface between hosts and CXL

Role: *Propagate* & *Translate* requests between **LP** and **GP**

Principle: *What requests to propagate?*

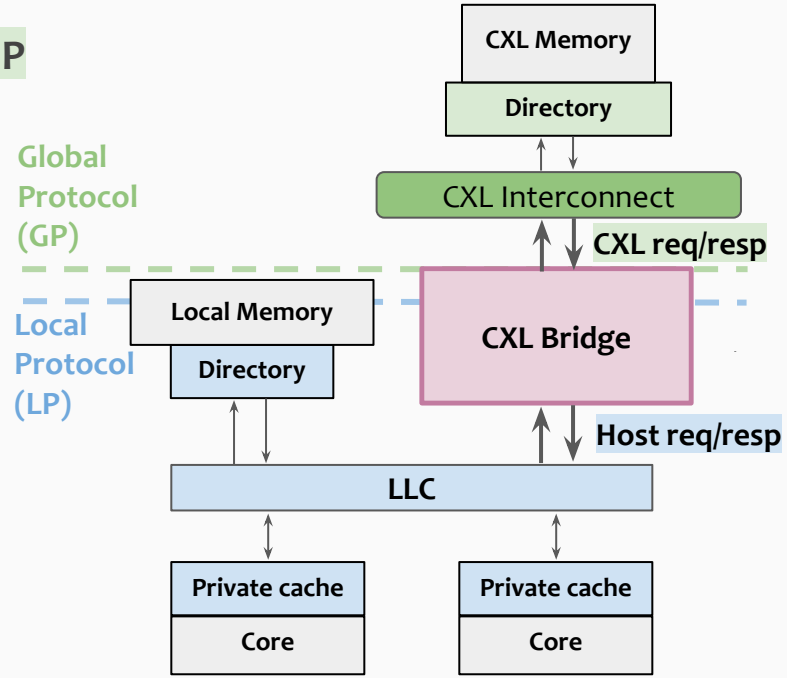


Architecture: **CXL cache coherence bridges**

Abstraction: CXL bridges sit at the interface between hosts and CXL

Role: *Propagate* & *Translate* requests between **LP** and **GP**

Principle: *What requests to propagate?*

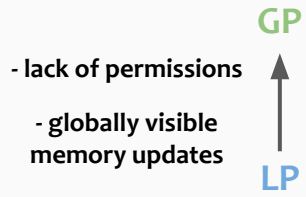


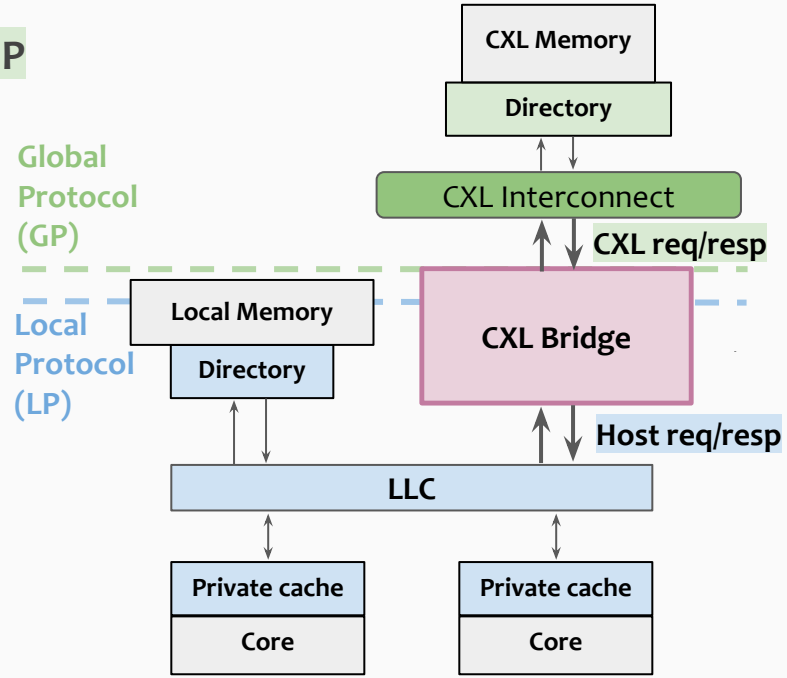
Architecture: **CXL cache coherence bridges**

Abstraction: CXL bridges sit at the interface between hosts and CXL

Role: *Propagate* & *Translate* requests between **LP** and **GP**

Principle: *What requests to propagate?*



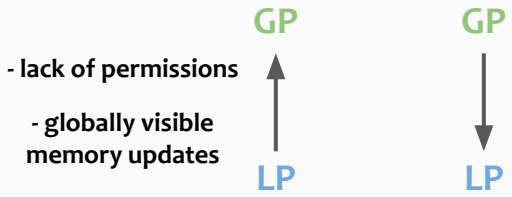


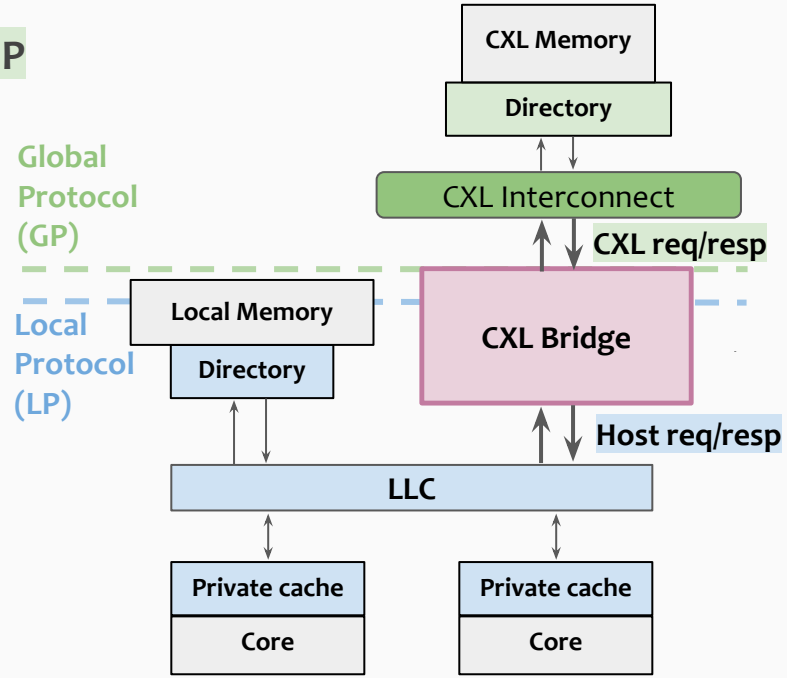
Architecture: **CXL cache coherence bridges**

Abstraction: CXL bridges sit at the interface between hosts and CXL

Role: *Propagate* & *Translate* requests between **LP** and **GP**

Principle: *What requests to propagate?*



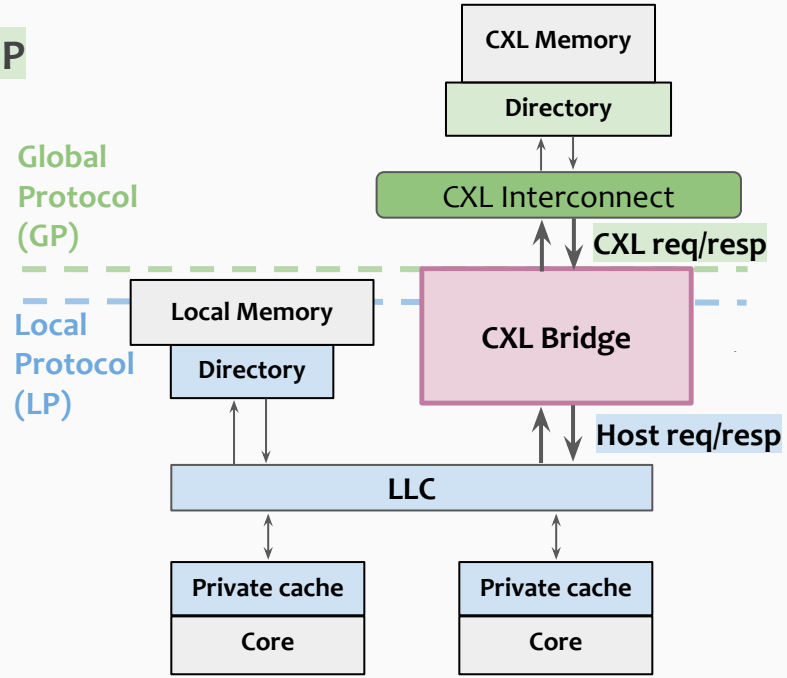


Architecture: **CXL cache coherence bridges**

Abstraction: CXL bridges sit at the interface between hosts and CXL

Role: *Propagate* & *Translate* requests between **LP** and **GP**

Principle: *What requests to propagate?*



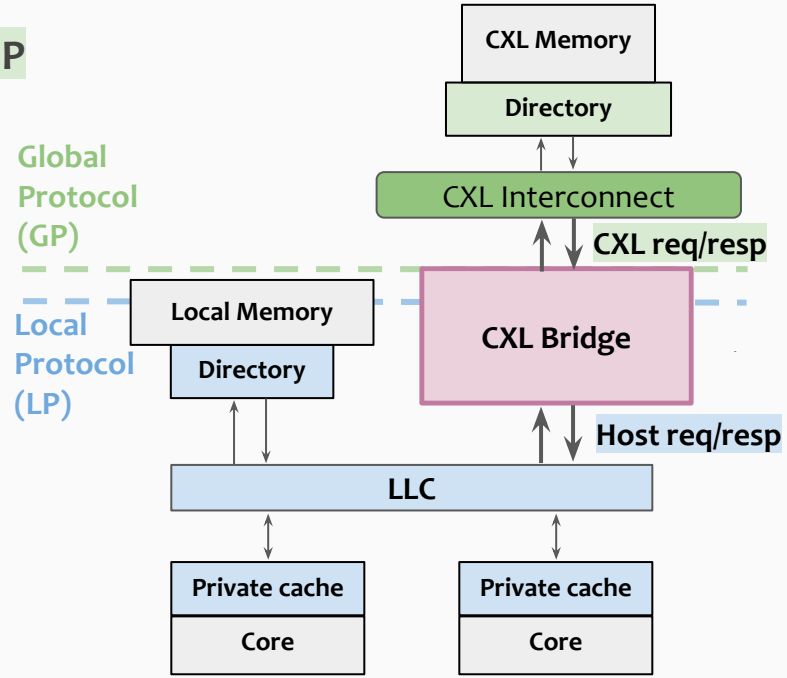
Architecture: **CXL cache coherence bridges**

Abstraction: CXL bridges sit at the interface between hosts and CXL

Role: *Propagate* & *Translate* requests between **LP** and **GP**

Principle: *What requests to propagate?*

Synthesis idea — Detect statically



Architecture: **CXL cache coherence bridges**

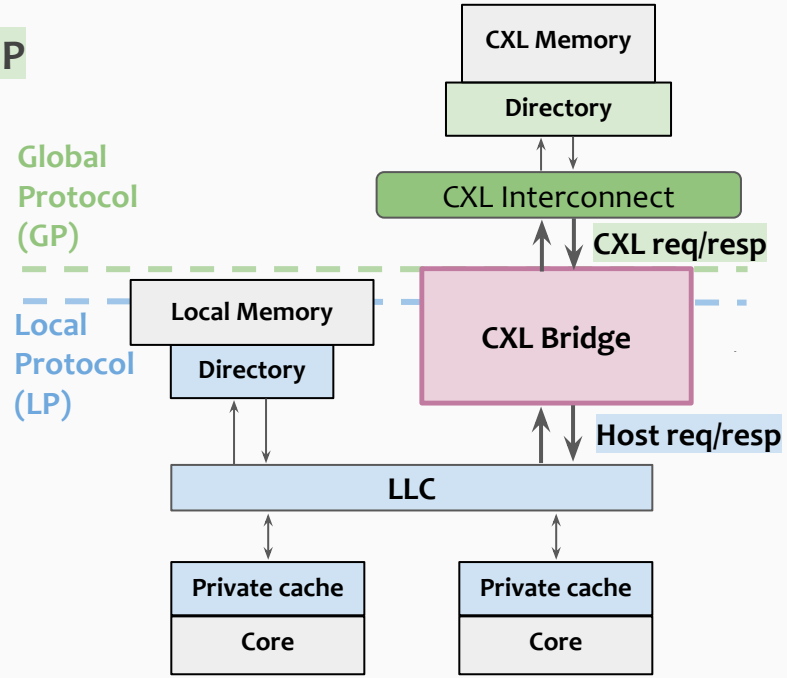
Abstraction: CXL bridges sit at the interface between hosts and CXL

Role: *Propagate* & *Translate* requests between **LP** and **GP**

Principle: *What requests to propagate?*

Synthesis idea — Detect statically

a. Which concrete requests *should propagate*



Architecture: **CXL cache coherence bridges**

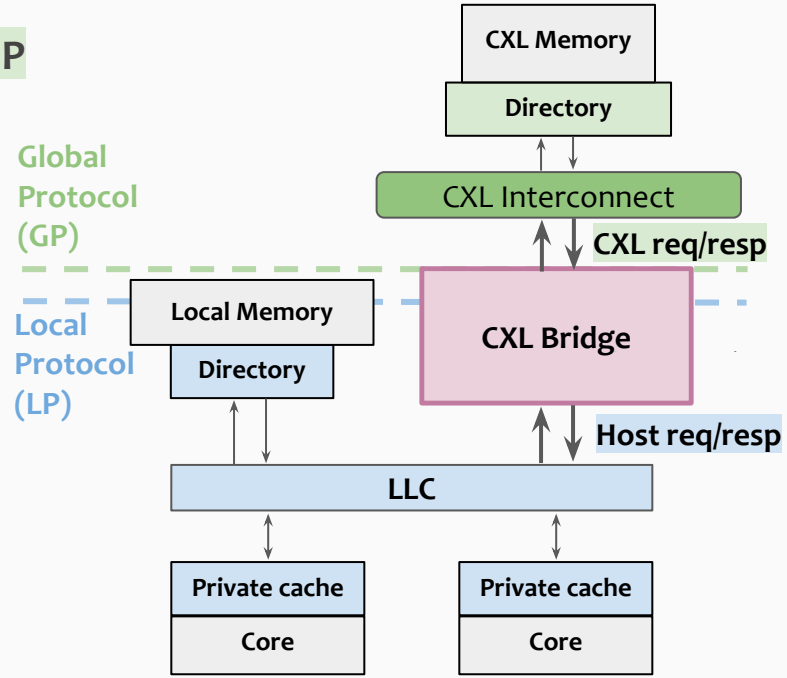
Abstraction: CXL bridges sit at the interface between hosts and CXL

Role: *Propagate* & *Translate* requests between **LP** and **GP**

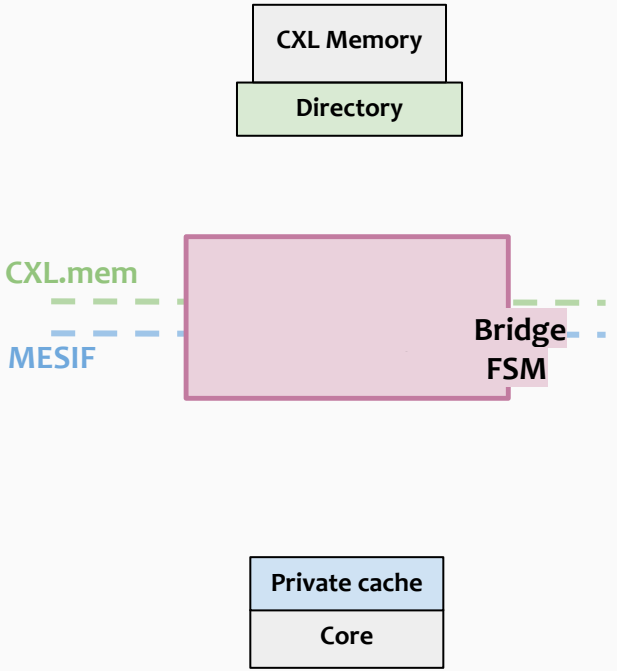
Principle: *What requests to propagate?*

Synthesis idea — Detect statically

a. Which concrete requests *should propagate*
b. *Semantically correct* request translations

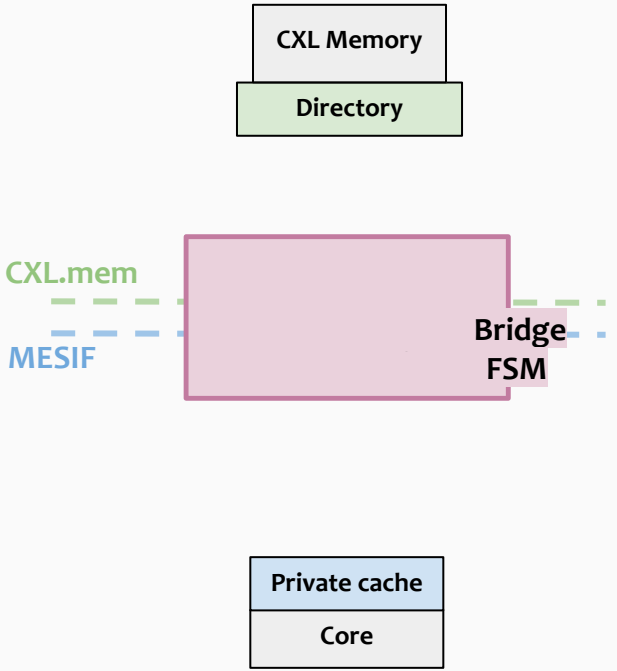


Synthesis: **Semantic translation**



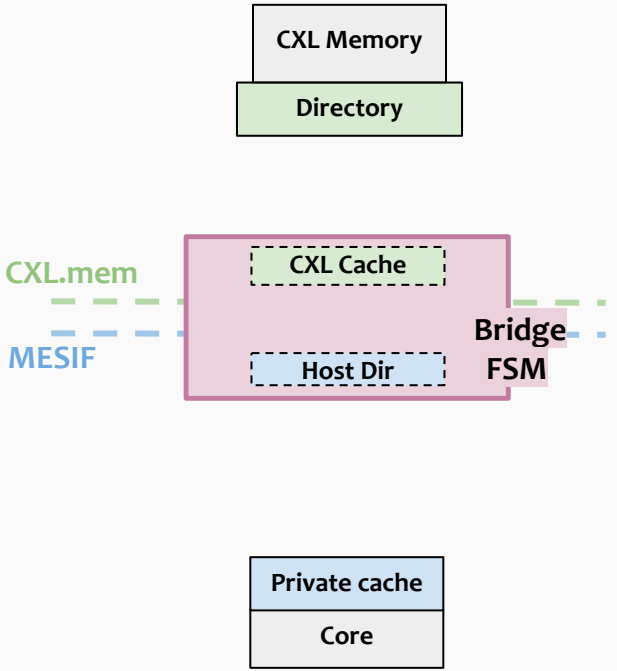
Synthesis: **Semantic translation**

How to generate the bridge FSM?



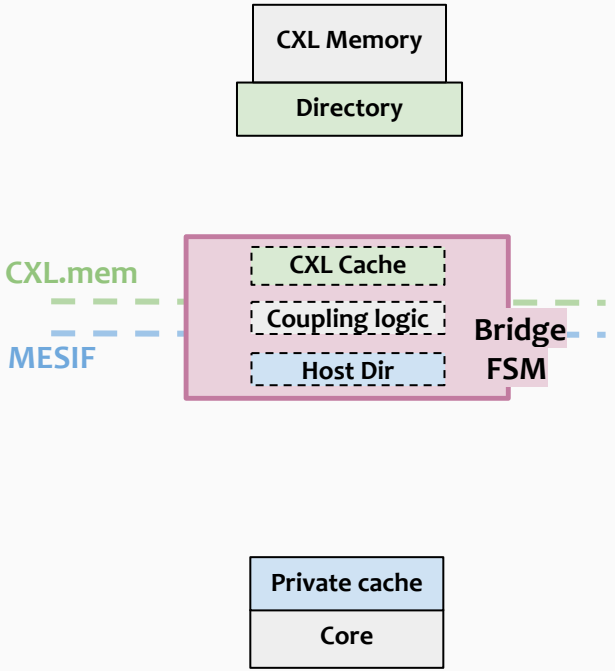
Synthesis: **Semantic translation**

How to generate the bridge FSM?



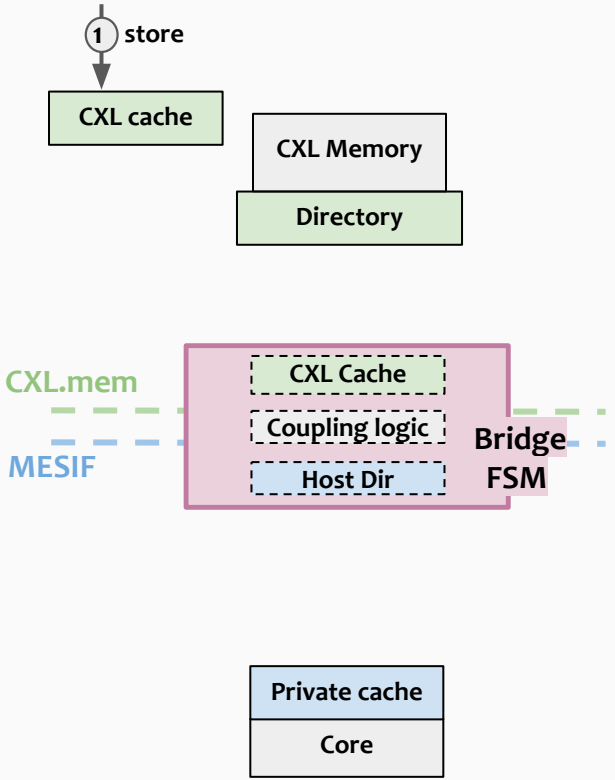
Synthesis: **Semantic translation**

How to generate the bridge FSM?



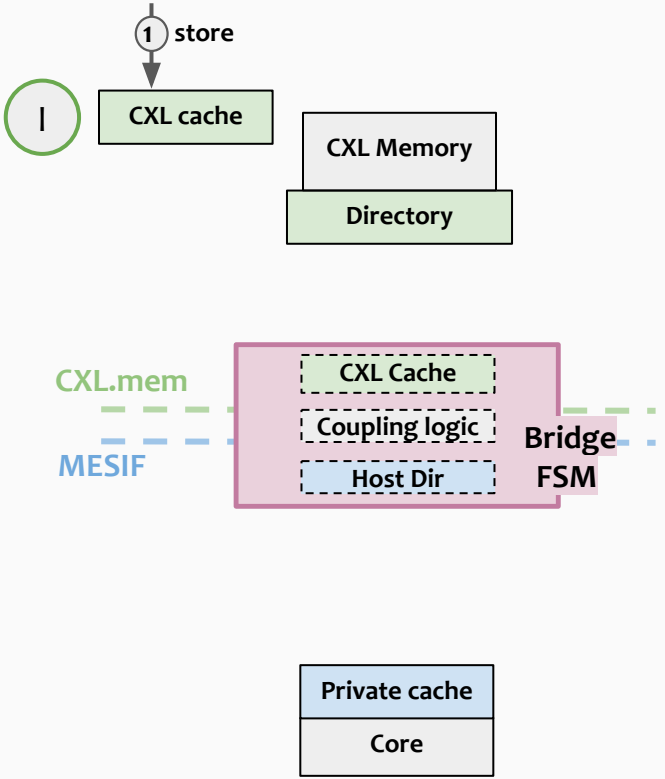
Synthesis: **Semantic translation**

How to generate the bridge FSM?



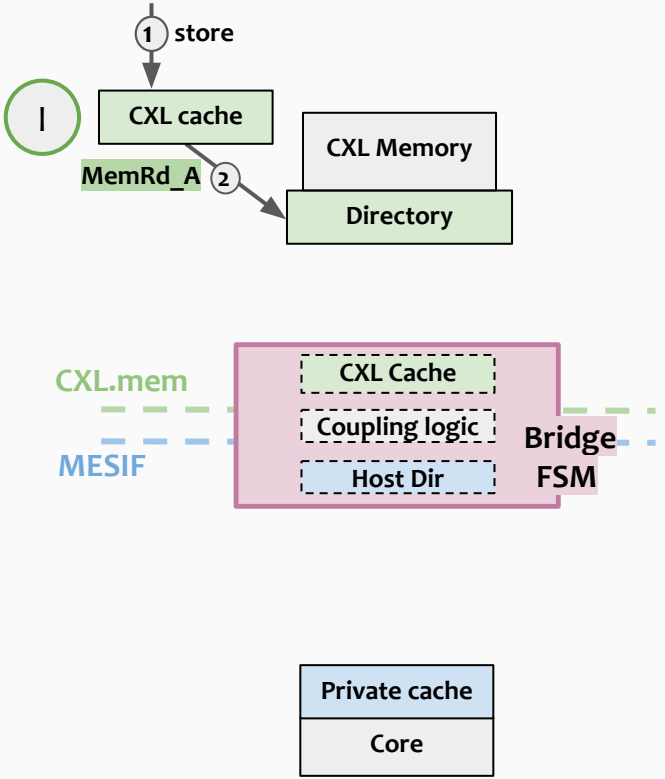
Synthesis: **Semantic translation**

How to generate the bridge FSM?



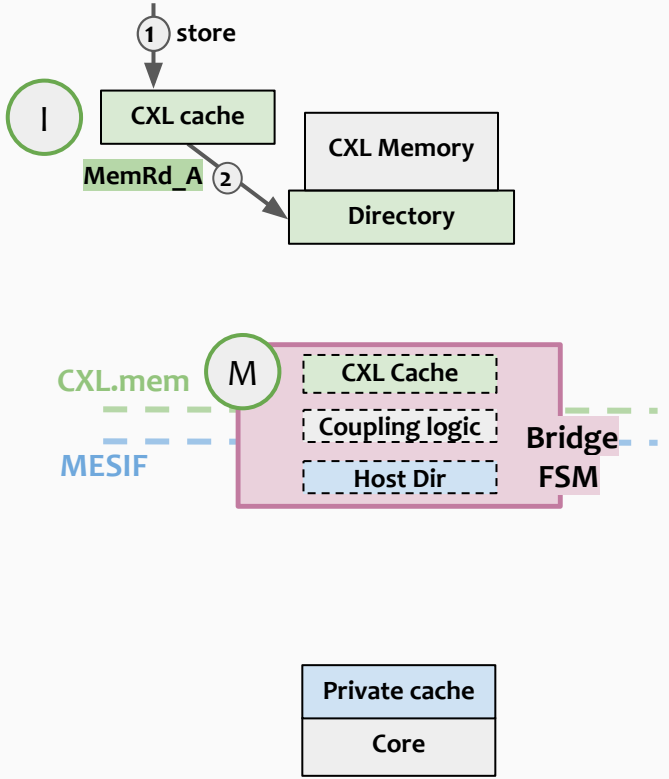
Synthesis: **Semantic translation**

How to generate the bridge FSM?



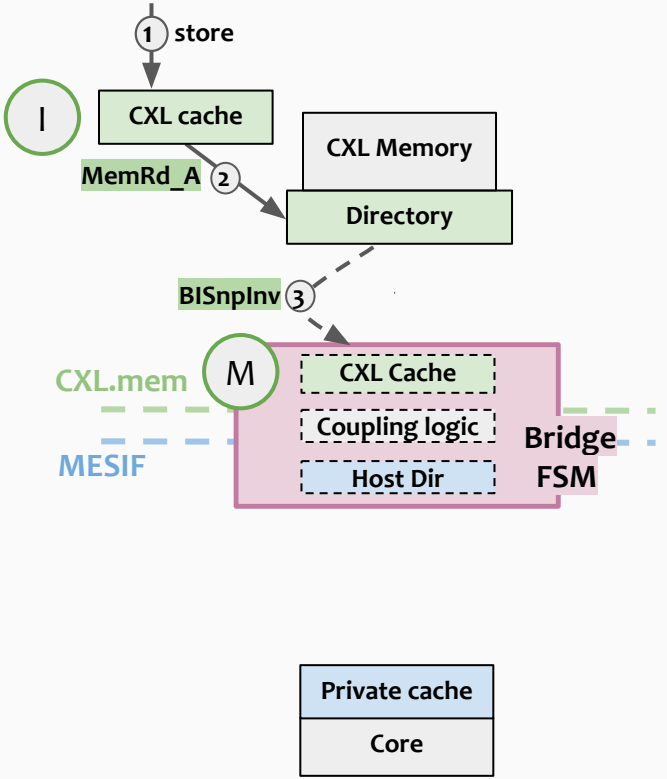
Synthesis: **Semantic translation**

How to generate the bridge FSM?



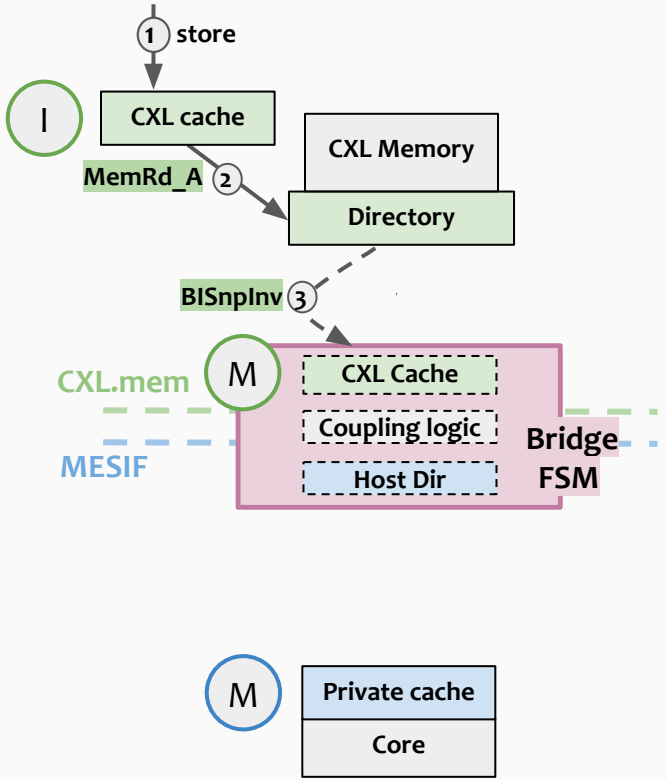
Synthesis: **Semantic translation**

How to generate the bridge FSM?



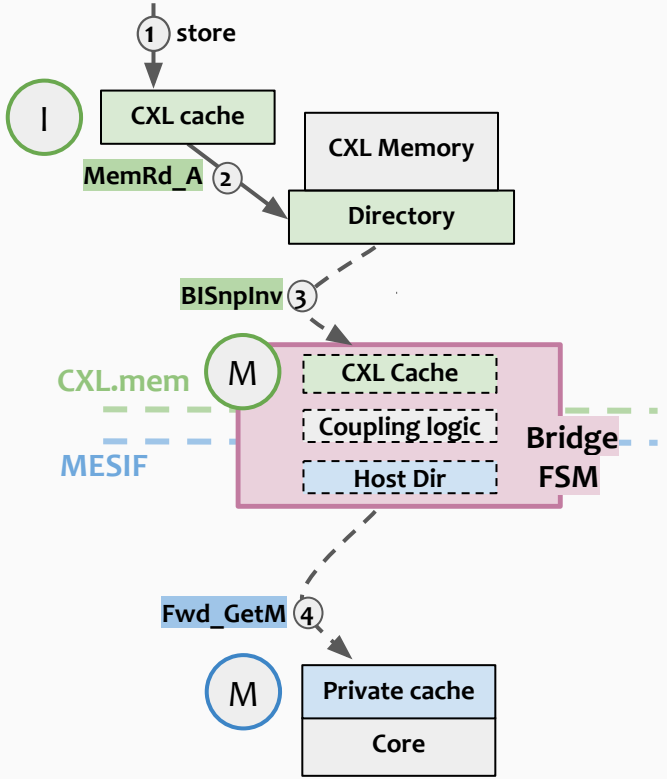
Synthesis: **Semantic translation**

How to generate the bridge FSM?



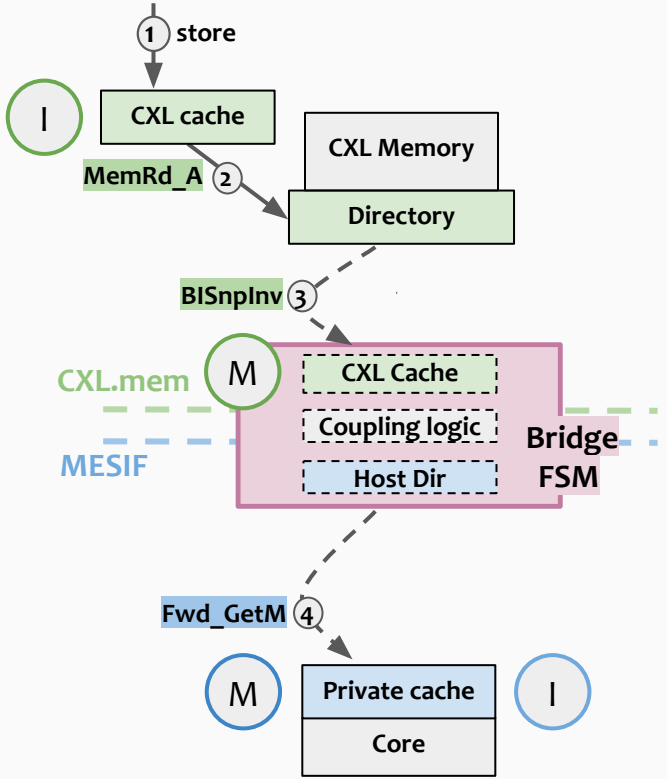
Synthesis: **Semantic translation**

How to generate the bridge FSM?



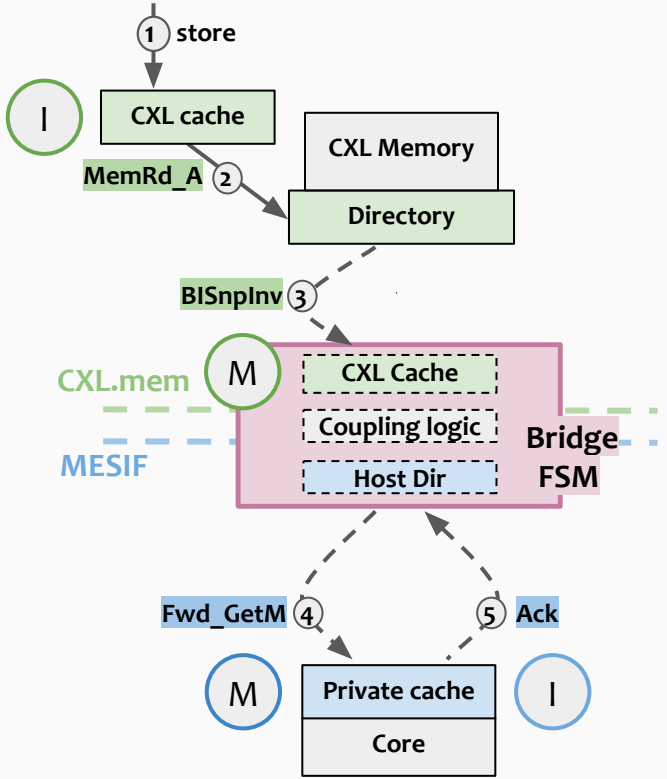
Synthesis: **Semantic translation**

How to generate the bridge FSM?



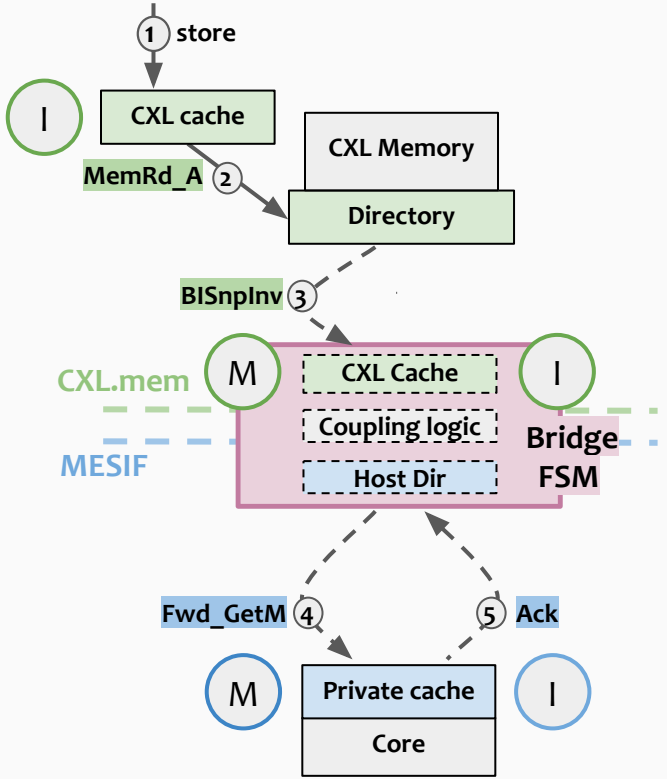
Synthesis: **Semantic translation**

How to generate the bridge FSM?



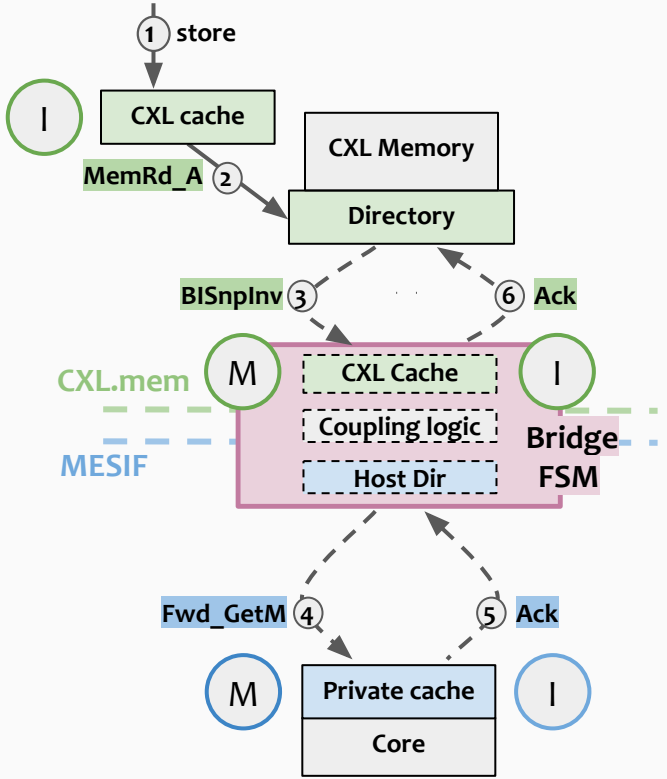
Synthesis: **Semantic translation**

How to generate the bridge FSM?



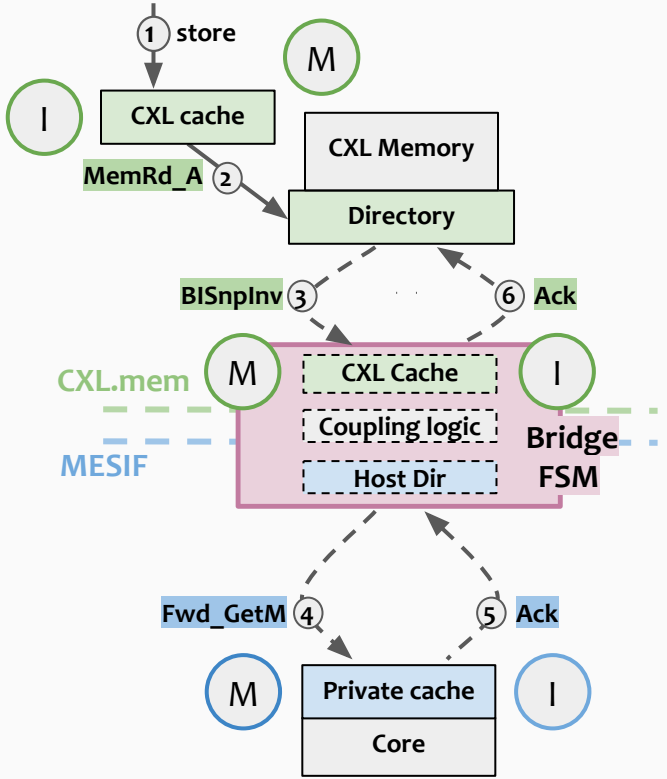
Synthesis: **Semantic translation**

How to generate the bridge FSM?



Synthesis: **Semantic translation**

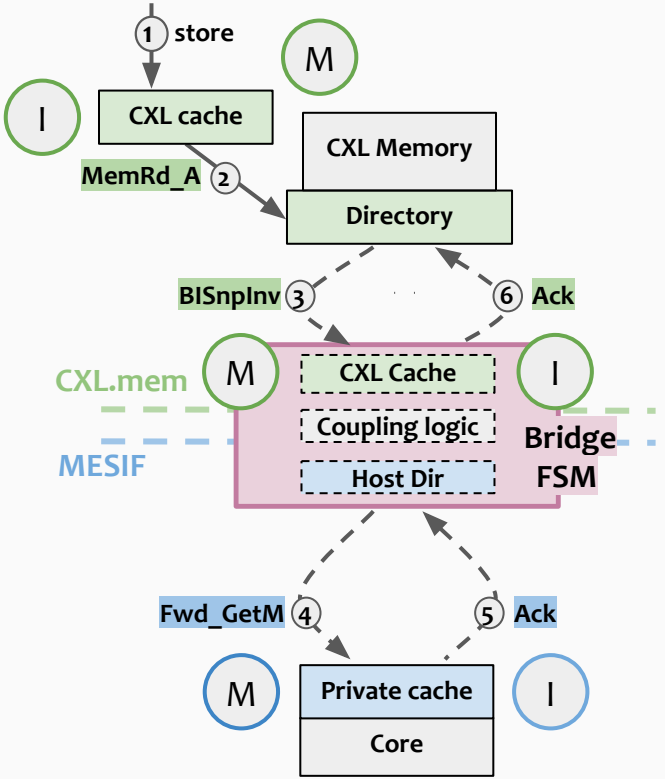
How to generate the bridge FSM?



Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs**:**

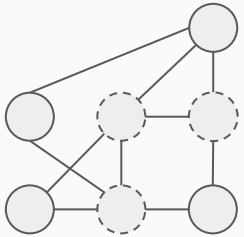


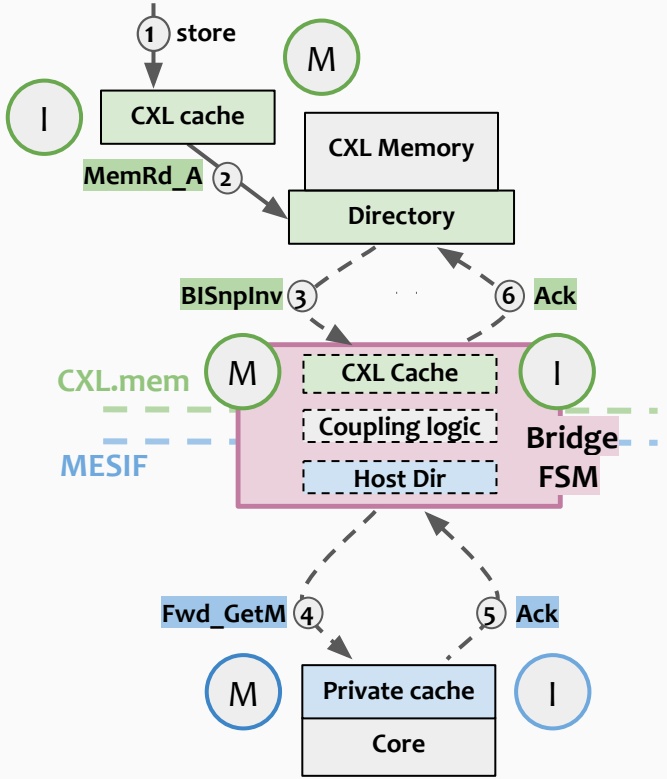
Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs**:**

Host Dir



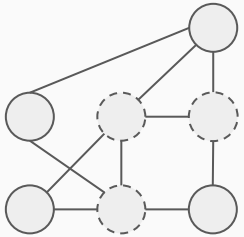


Synthesis: **Semantic translation**

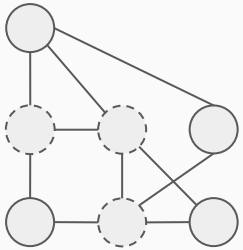
How to generate the bridge FSM?

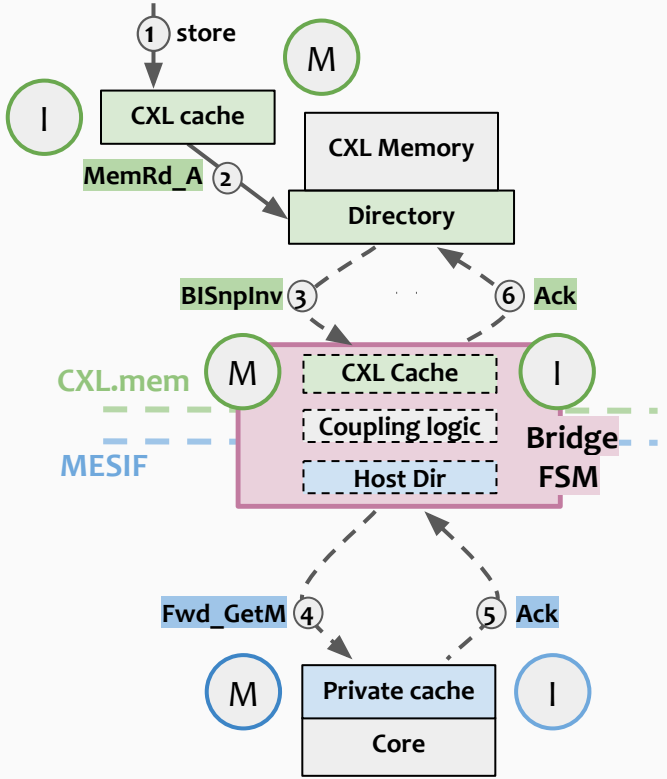
Static analysis of Host Dir & CXL Cache FSMs:

Host Dir



CXL Cache



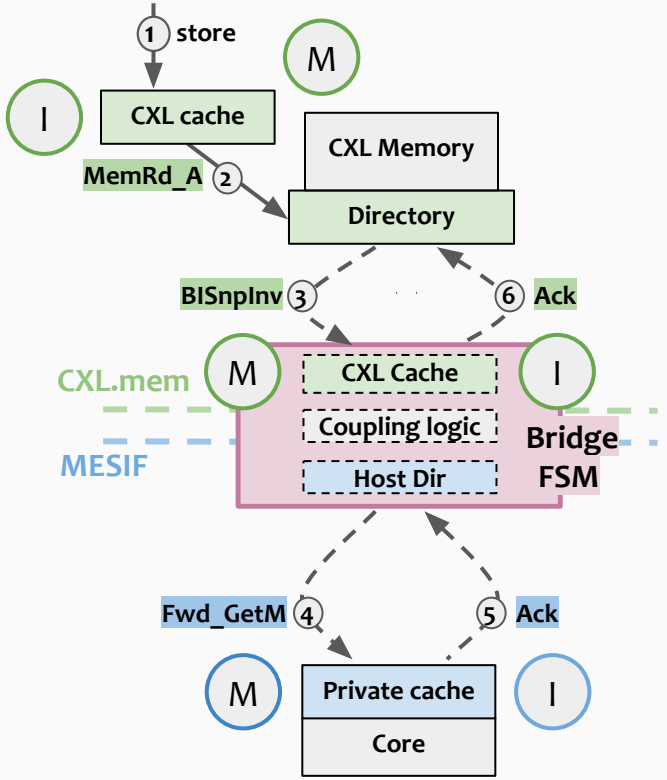


Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:

Host Dir Coupling logic CXL Cache

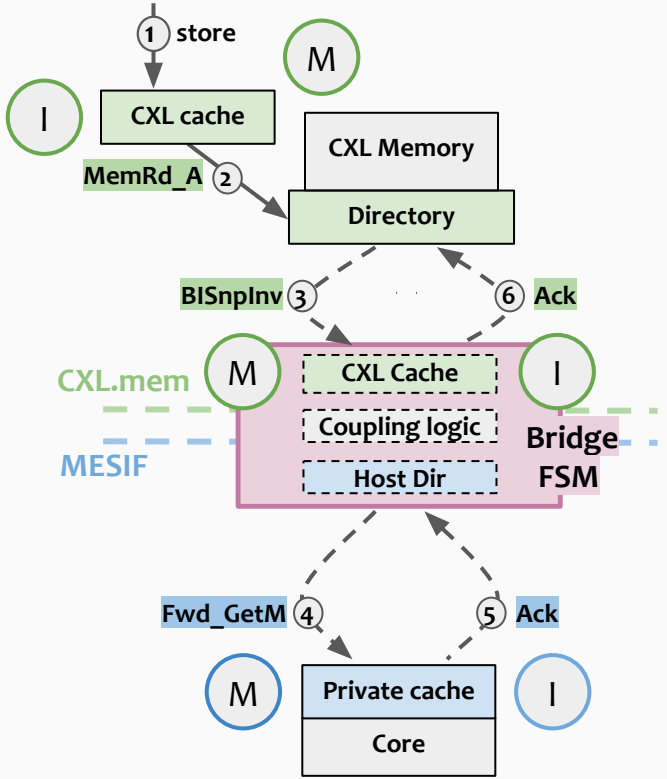


Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:

Host Dir Coupling logic CXL Cache

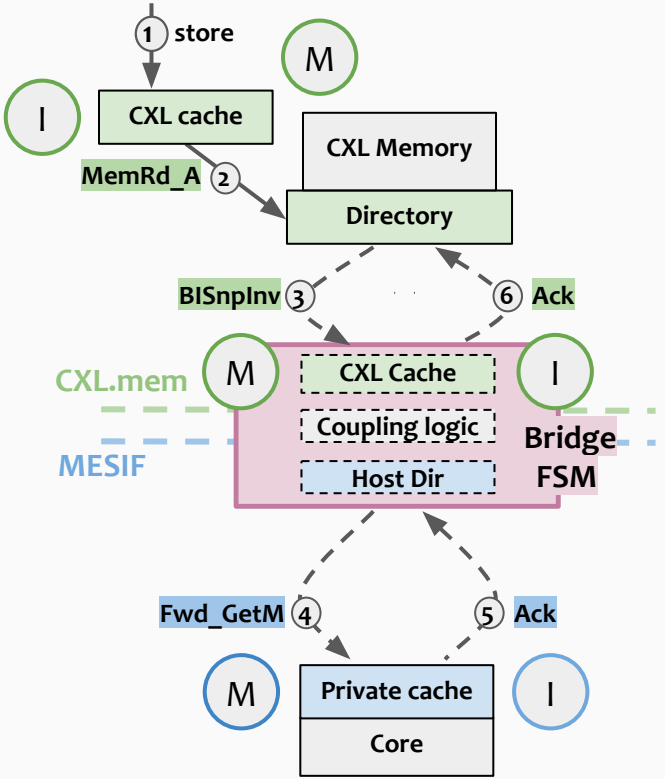


Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:

Host Dir Coupling logic CXL Cache

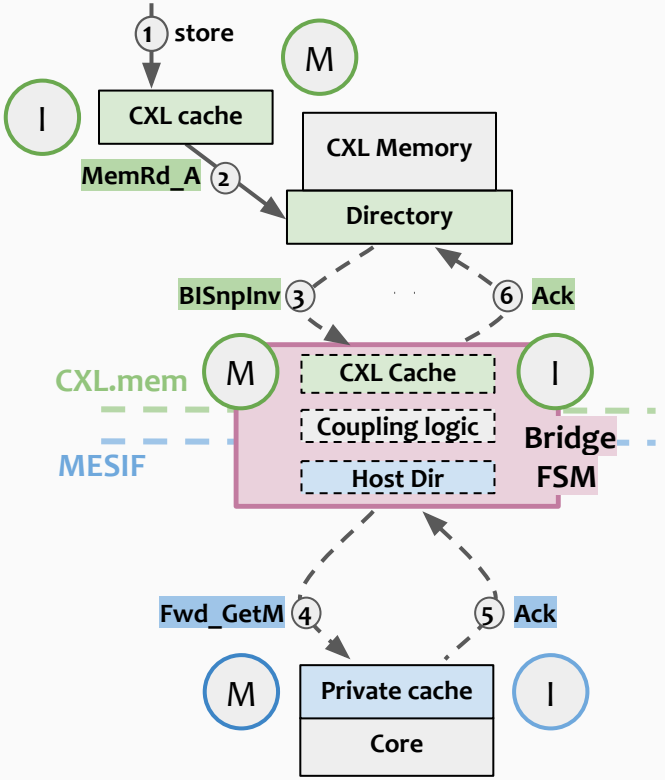


Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:

Host Dir Coupling logic CXL Cache



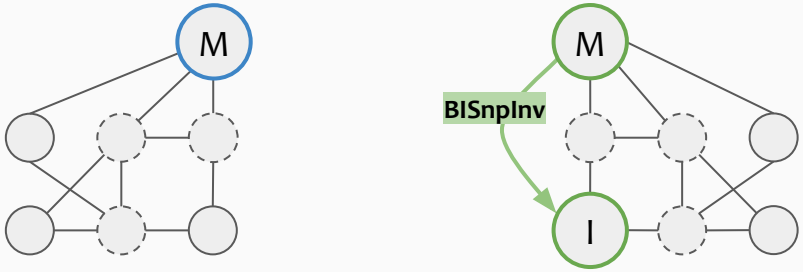
1. *Identify requests that must propagate*

Synthesis: **Semantic translation**

How to generate the bridge FSM?

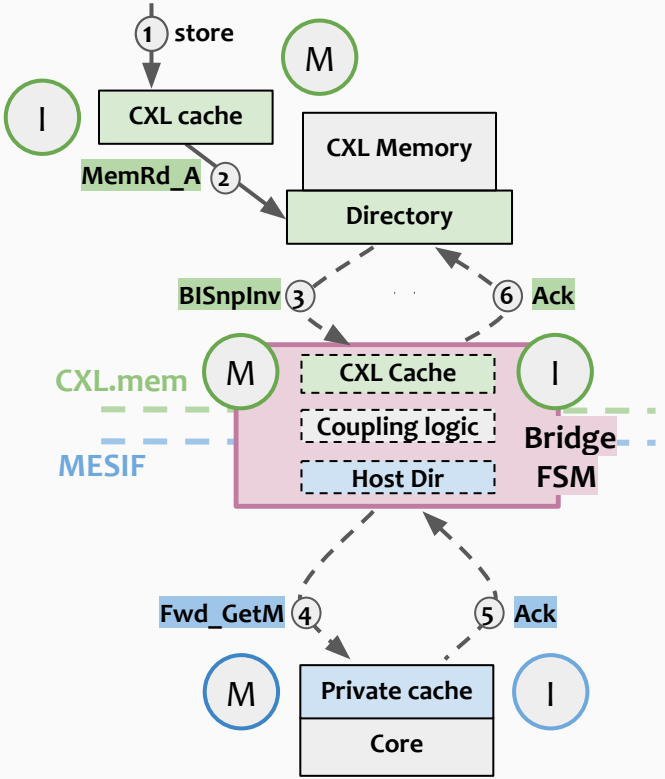
Static analysis of Host Dir & CXL Cache FSMs:

Host Dir | Coupling logic | CXL Cache



(cache permission downgrade)

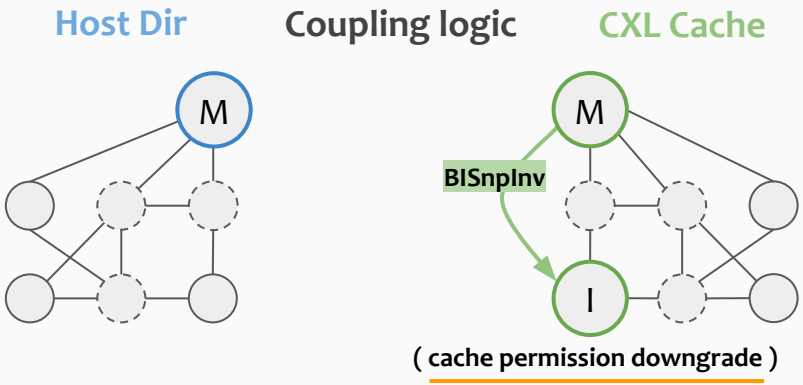
1. *Identify requests that must propagate*



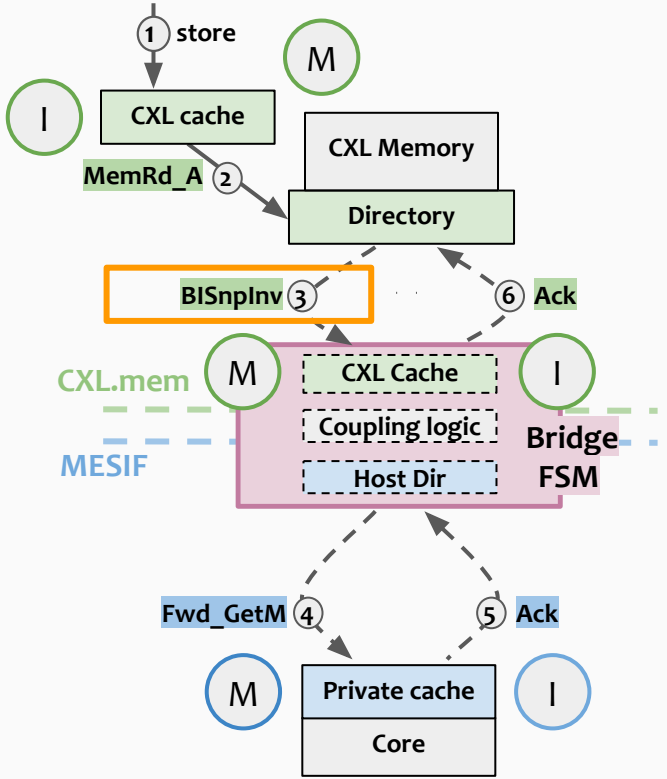
Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:



1. *Identify requests that must propagate*

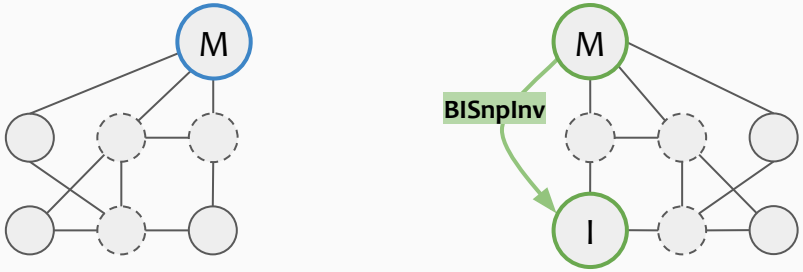


Synthesis: **Semantic translation**

How to generate the bridge FSM?

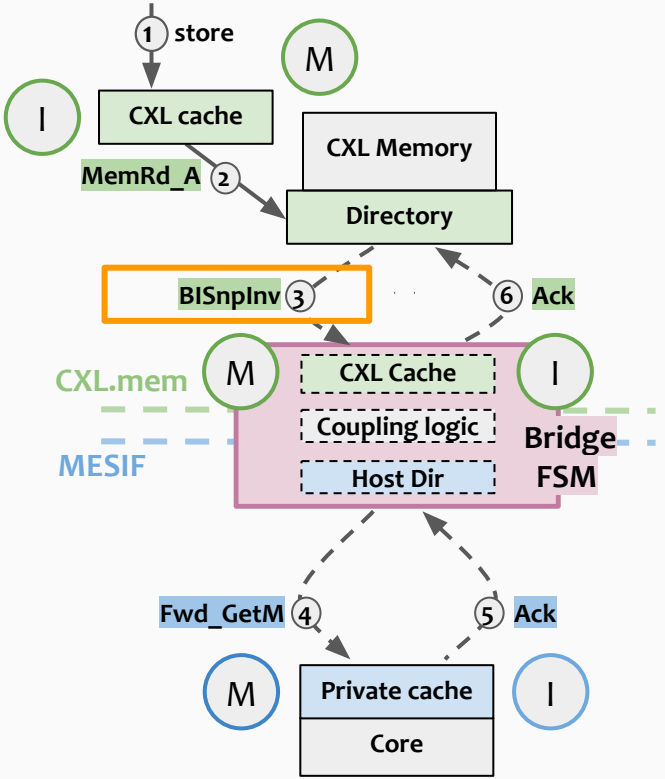
Static analysis of Host Dir & CXL Cache FSMs:

Host Dir | Coupling logic | CXL Cache



(cache permission downgrade)

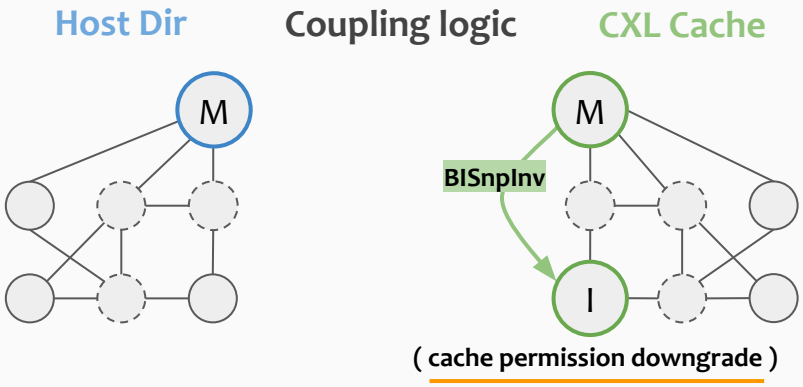
1. *Identify requests that must propagate*
2. *Deduce access of original requestor*



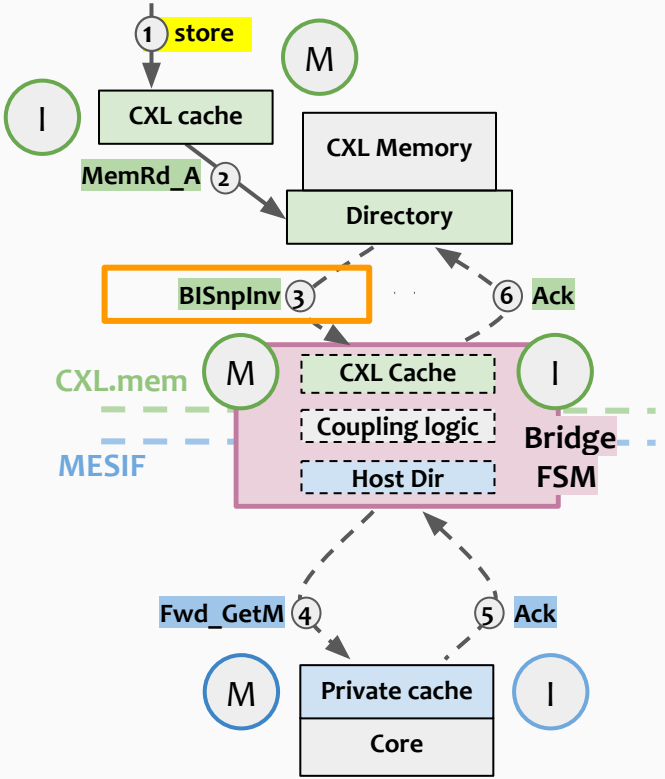
Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:



1. *Identify requests that must propagate*
2. *Deduce access of original requestor*

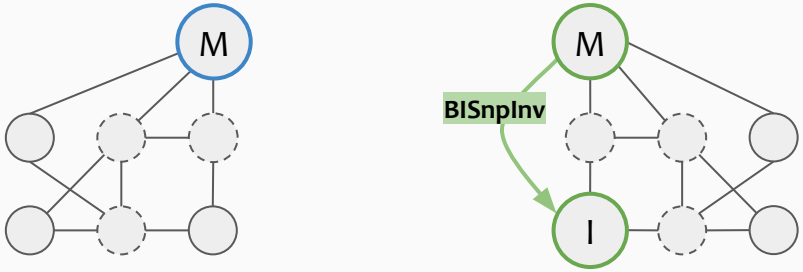


Synthesis: **Semantic translation**

How to generate the bridge FSM?

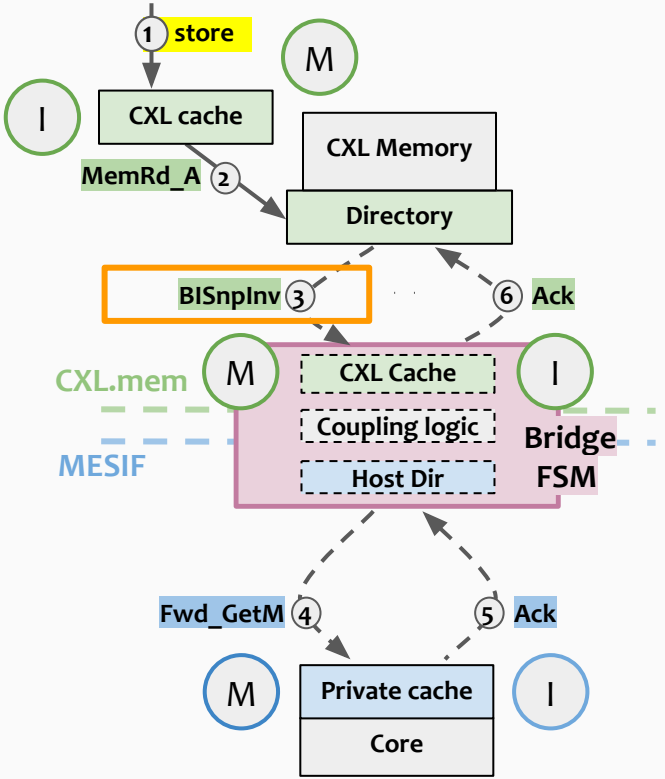
Static analysis of Host Dir & CXL Cache FSMs:

Host Dir | Coupling logic | CXL Cache



(cache permission downgrade)

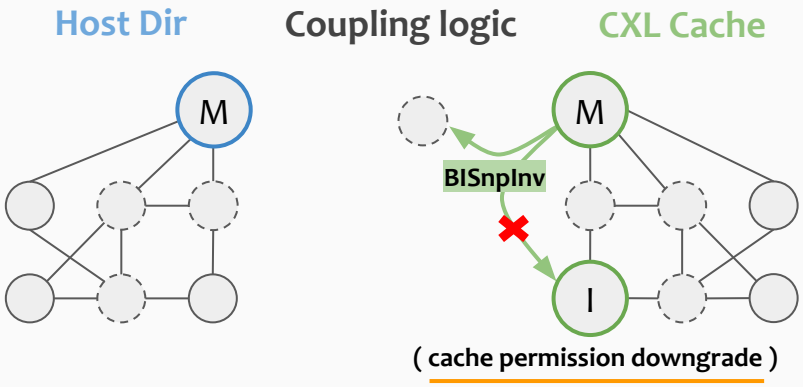
1. *Identify requests that must propagate*
2. *Deduce access of original requestor*
3. *Simulate access to remote FSM*



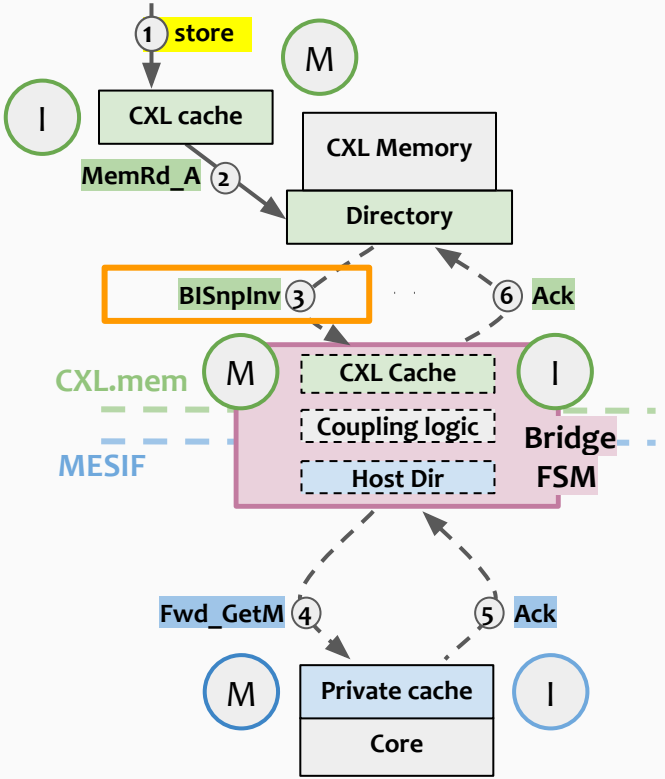
Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:



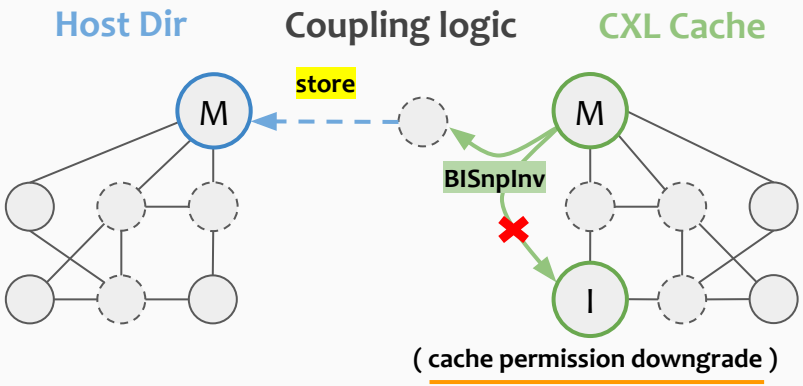
1. *Identify requests that must propagate*
2. *Deduce access of original requestor*
3. *Simulate access to remote FSM*



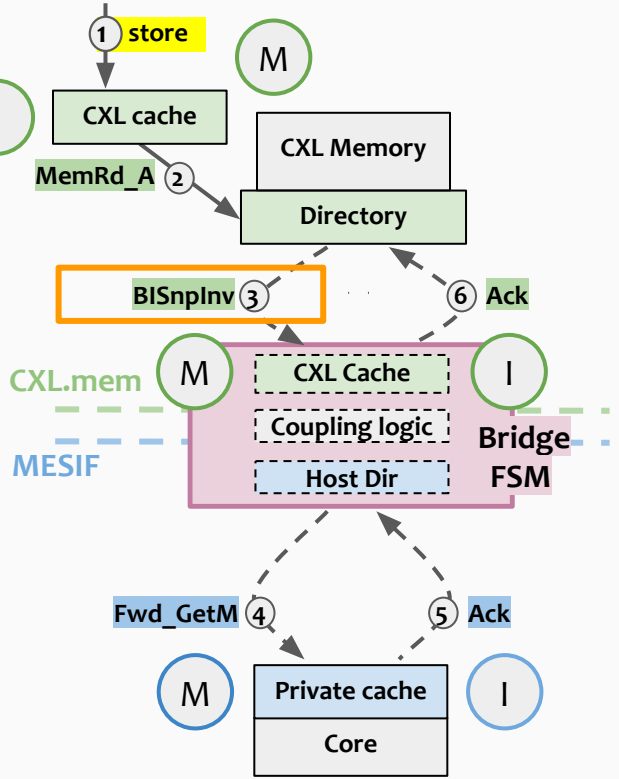
Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:



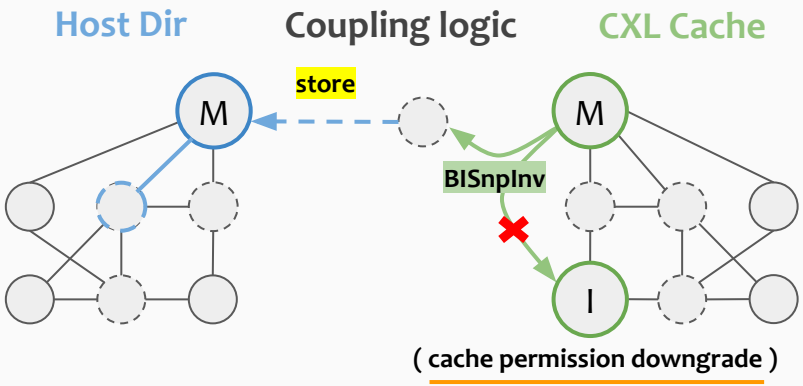
1. *Identify requests that must propagate*
2. *Deduce access of original requestor*
3. *Simulate access to remote FSM*



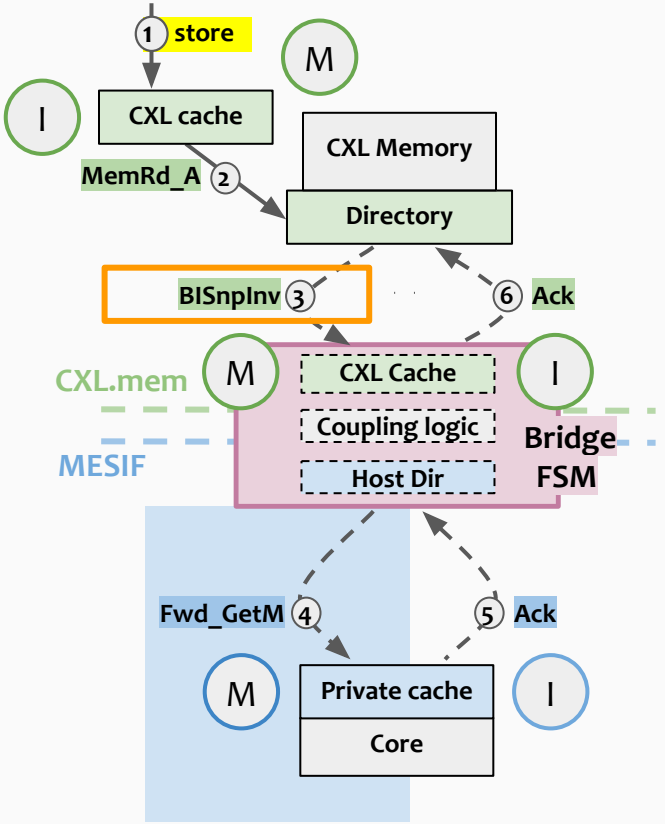
Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:



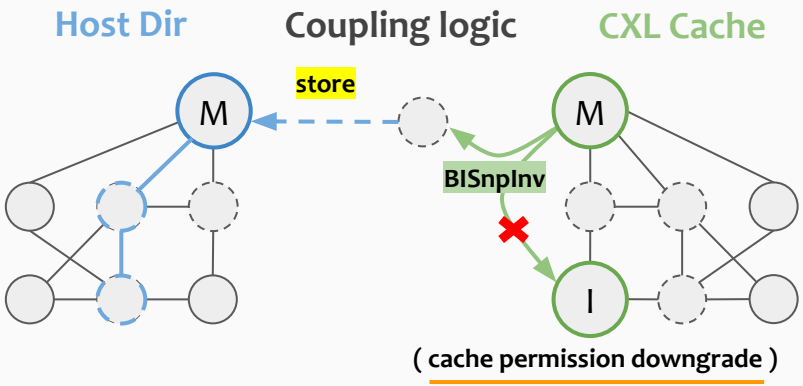
1. *Identify requests that must propagate*
2. *Deduce access of original requestor*
3. *Simulate access to remote FSM*



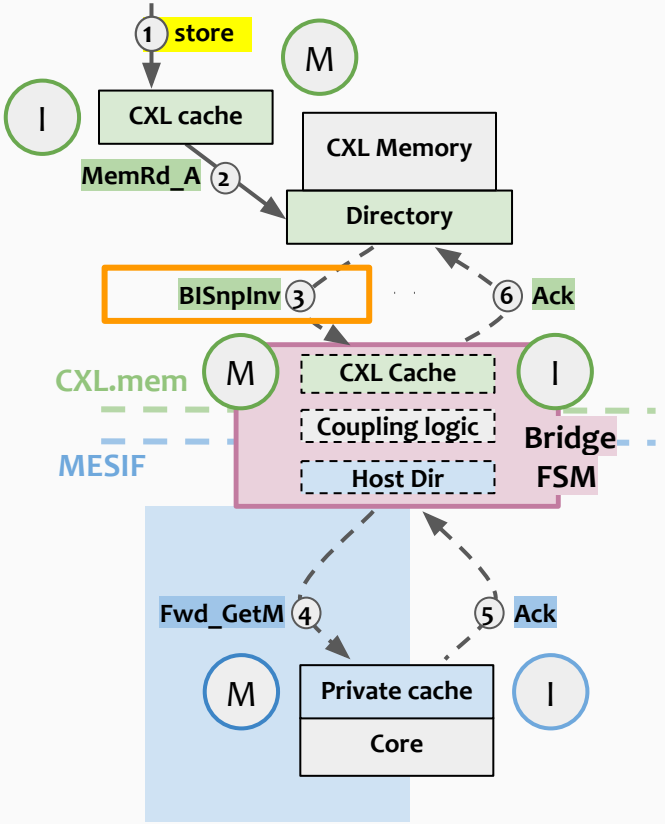
Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:



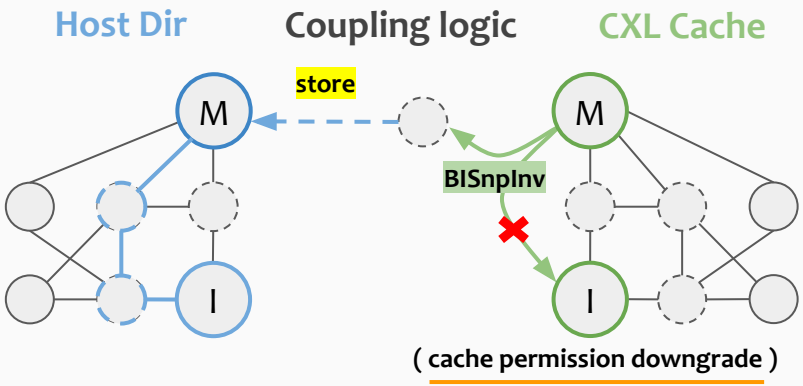
1. *Identify requests that must propagate*
2. *Deduce access of original requestor*
3. *Simulate access to remote FSM*



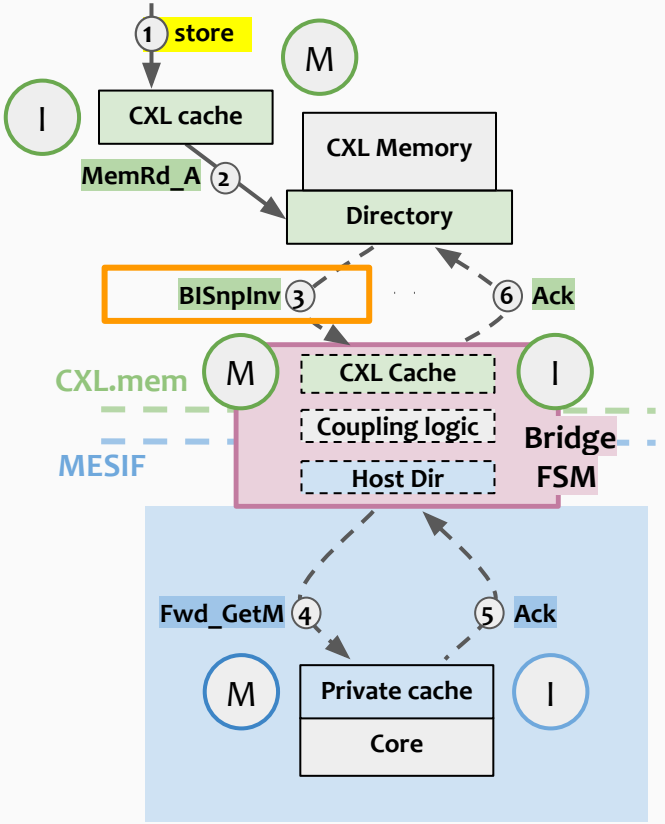
Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:



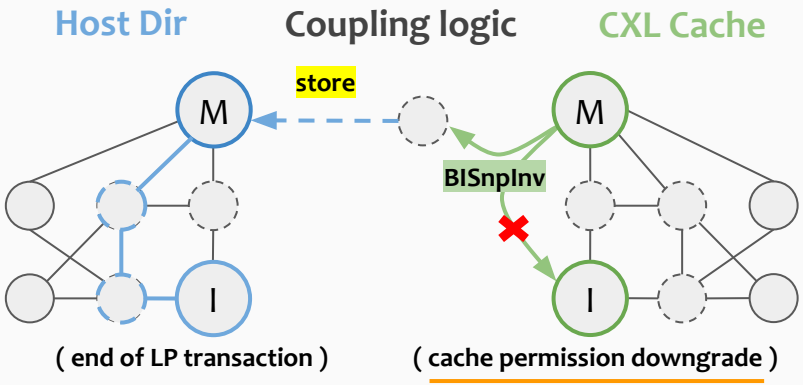
1. *Identify requests that must propagate*
2. *Deduce access of original requestor*
3. *Simulate access to remote FSM*



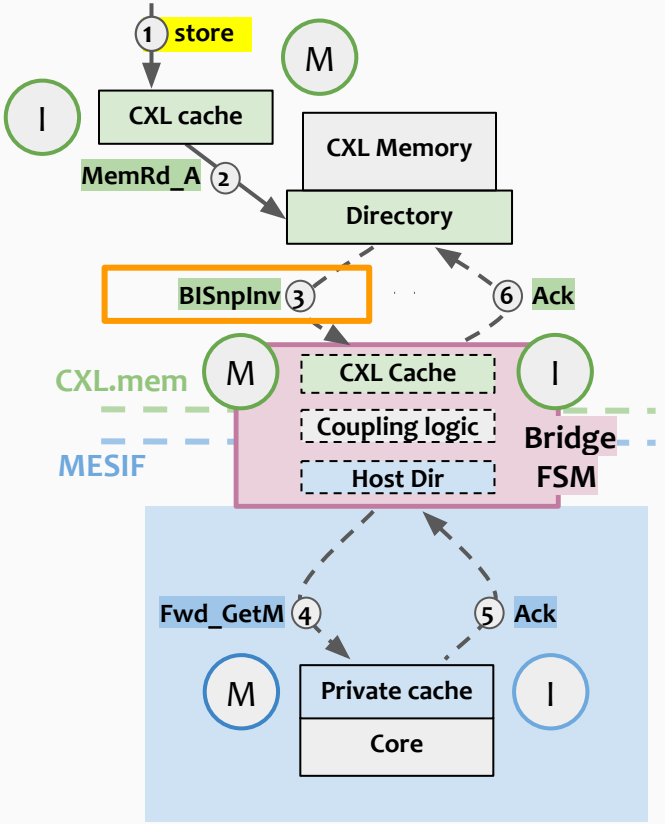
Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:



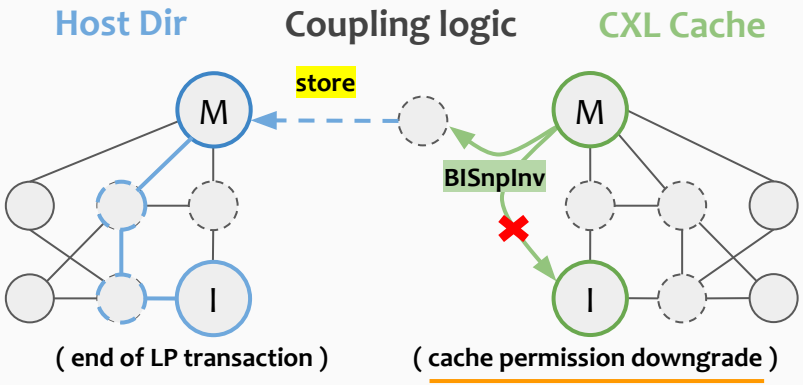
1. *Identify requests that must propagate*
2. *Deduce access of original requestor*
3. *Simulate access to remote FSM*



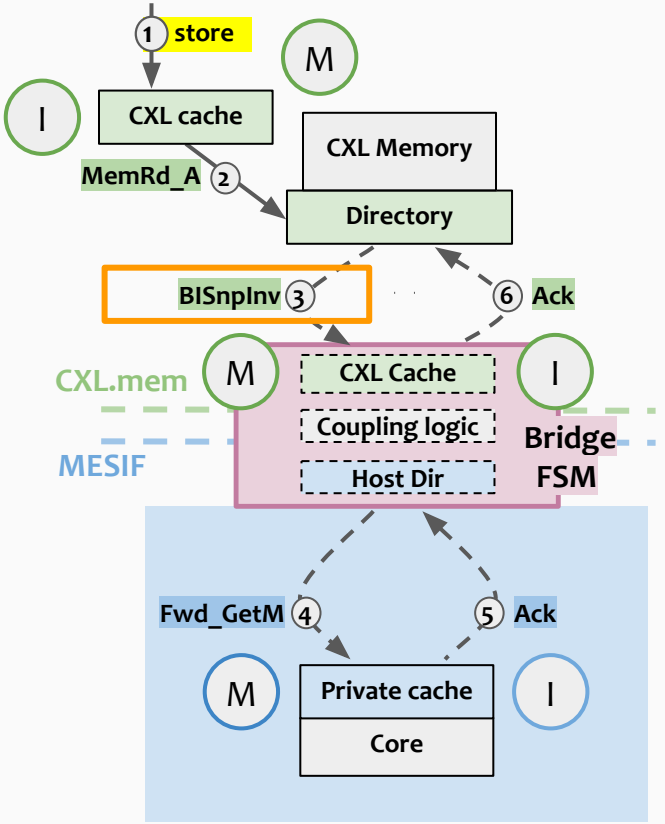
Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:



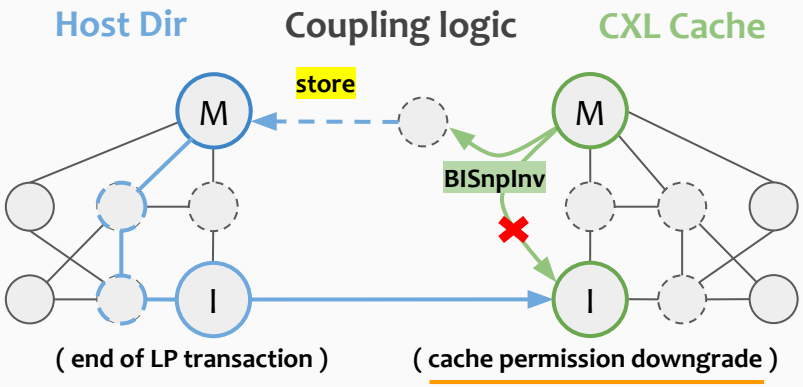
1. *Identify requests that must propagate*
2. *Deduce access of original requestor*
3. *Simulate access to remote FSM*
4. *Resume origin transaction*



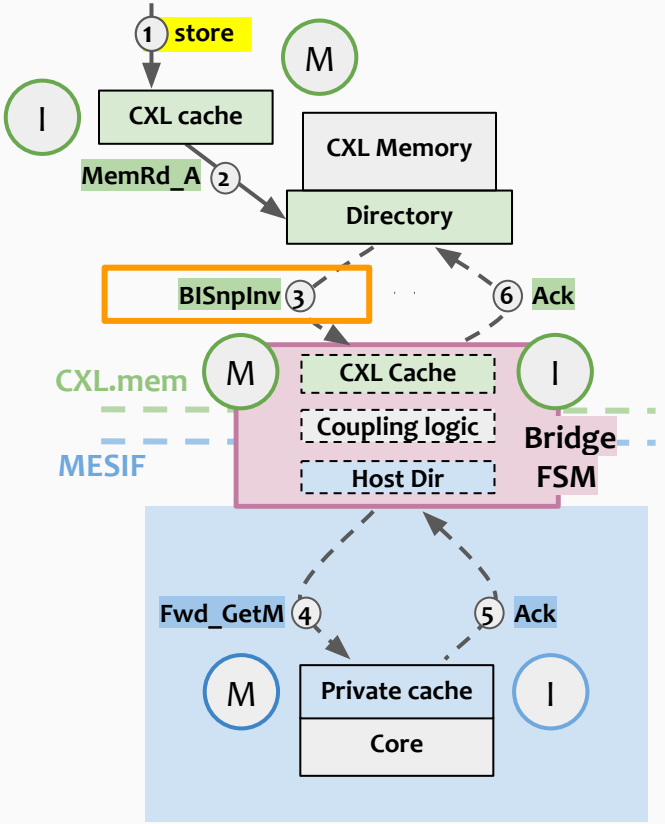
Synthesis: **Semantic translation**

How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:



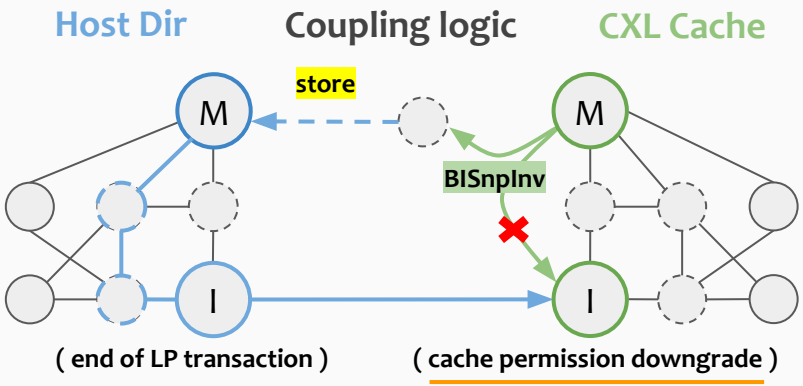
1. *Identify requests that must propagate*
2. *Deduce access of original requestor*
3. *Simulate access to remote FSM*
4. *Resume origin transaction*



Synthesis: **Semantic translation**

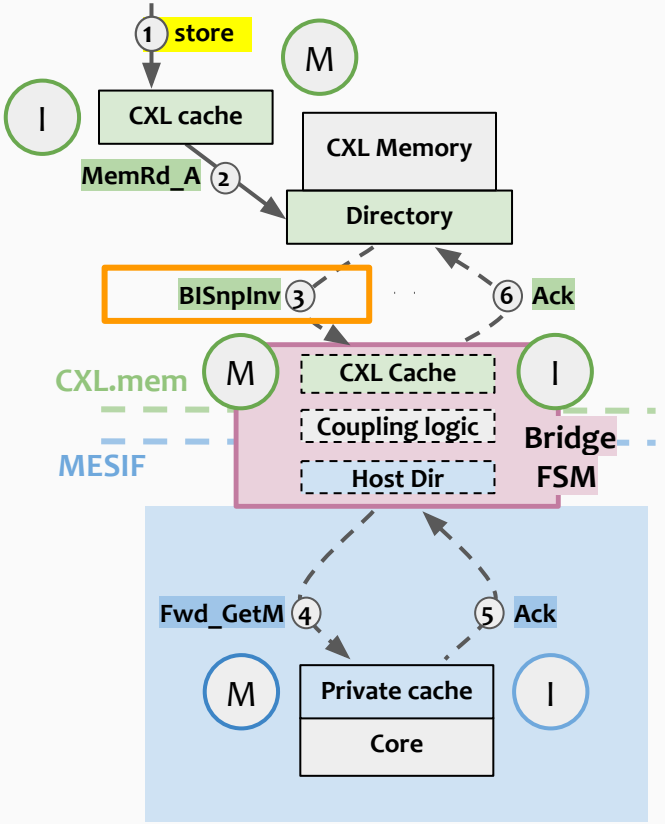
How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:



For all <LP, GP> state tuples

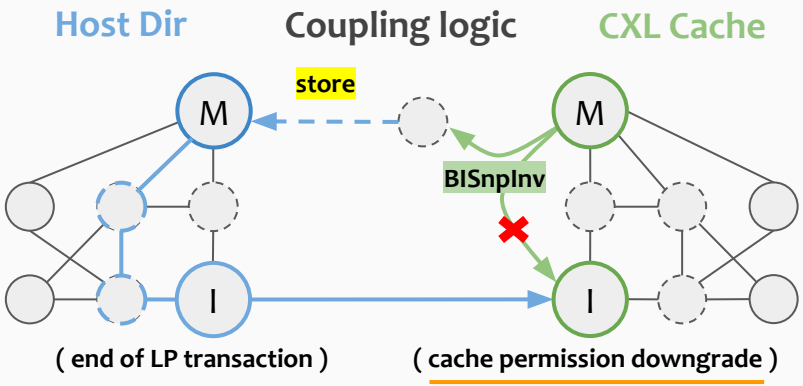
1. *Identify requests that must propagate*
2. *Deduce access of original requestor*
3. *Simulate access to remote FSM*
4. *Resume origin transaction*



Synthesis: **Semantic translation**

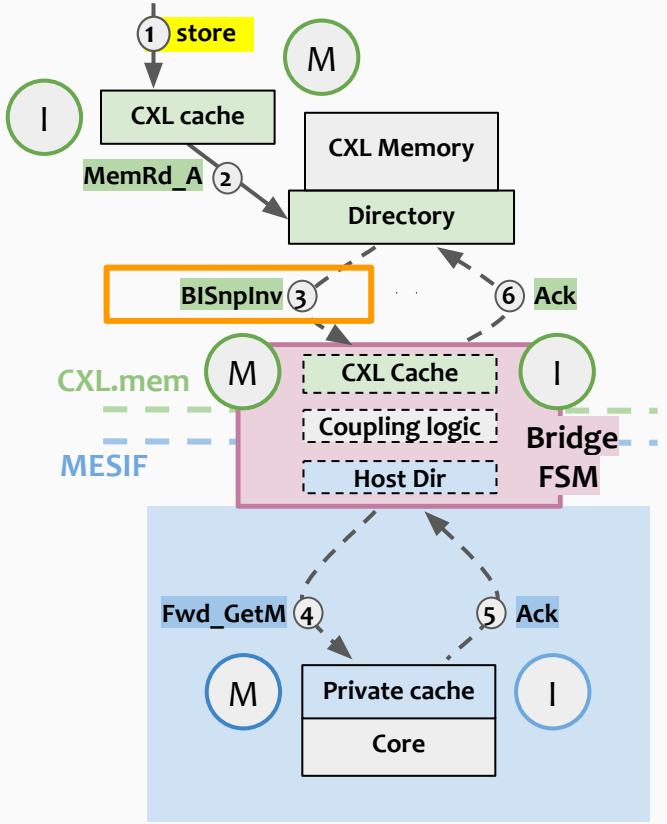
How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:



For all <LP, GP> state tuples

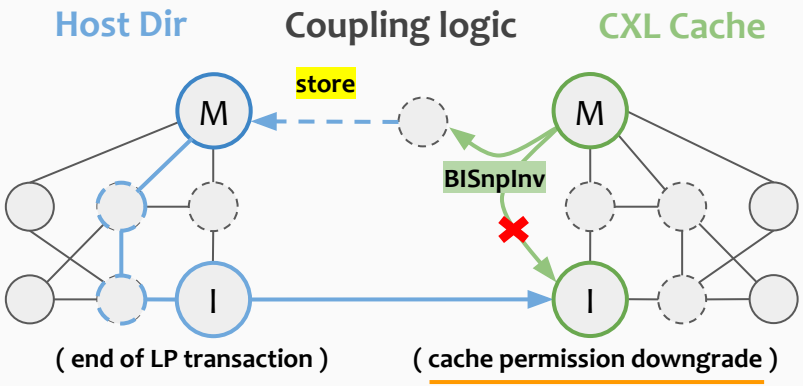
1. *Identify requests that must propagate*] *Detect propagation*
2. *Deduce access of original requestor*
3. *Simulate access to remote FSM*
4. *Resume origin transaction*



Synthesis: **Semantic translation**

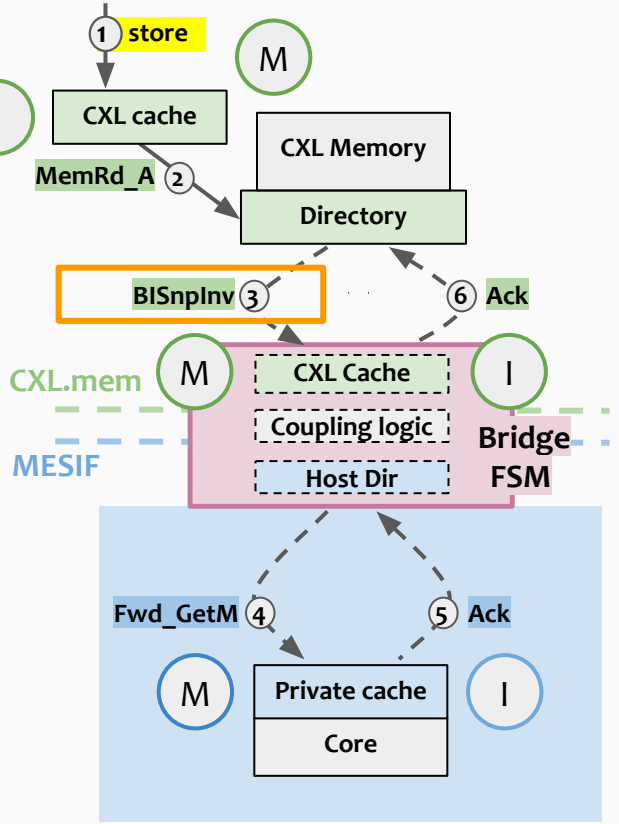
How to generate the bridge FSM?

Static analysis of Host Dir & CXL Cache FSMs:

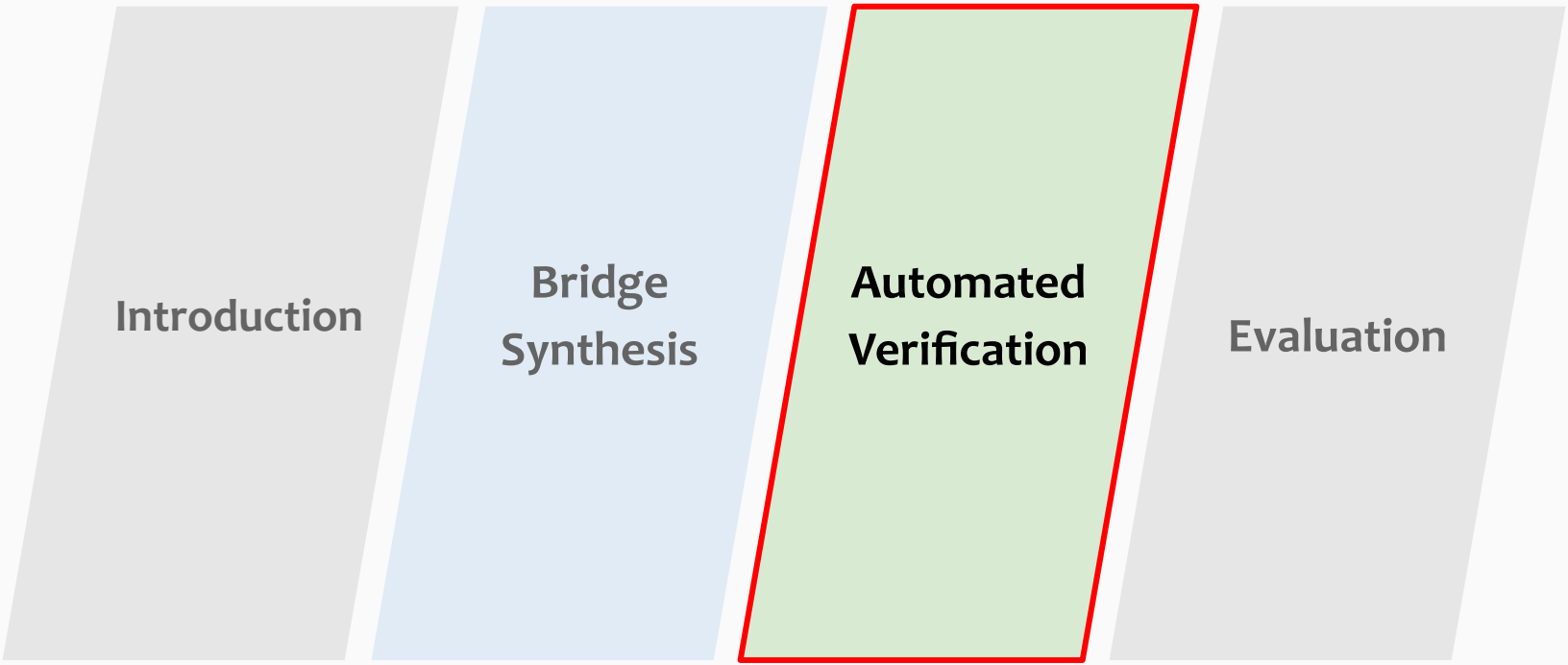


For all <LP, GP> state tuples

1. *Identify requests that must propagate* — *Detect propagation*
2. *Deduce access of original requestor* — *Semantic translation*
3. *Simulate access to remote FSM* — *Semantic translation*
4. *Resume origin transaction*



Outline


Introduction
Bridge Synthesis
Automated Verification
Evaluation

Verification: **Properties**

*Use **model checkers** to verify **heterogeneous** system models:*

1

Safety

Adherence to Compound MCM

2

Liveness

Deadlock-freedom

All memory requests can terminate

Verification: **Properties**

*Use **model checkers** to verify **heterogeneous** system models:*

1

Safety

Adherence to Compound MCM

Compound MCM litmus tests[1]

Explore all litmus test instruction interleavings

2

Liveness

Deadlock-freedom

All memory requests can terminate

Reachability of read/write permission state

Explore all possible instruction interleavings

[1] *Compound Memory Models*, Goens et al., PLDI'23

Verification: **Scalability**

Problem: *State space explosion of the liveness models*

Example:

Setup: MESI and RCC *clusters connected by CXL fabric*

Property: *Deadlock-freedom*

Configuration	Status	# of MC States	RAM usage
2 cores each	**Success**	22 Million	**12 GB**
3 cores each	**Out of Memory**	> 1.4 Billion	**> 1.8 TB**

Verification: **Scalability**

Problem: *State space explosion of the liveness models*

Example:

Setup: MESI and RCC *clusters connected by CXL fabric*

Property: *Deadlock-freedom*

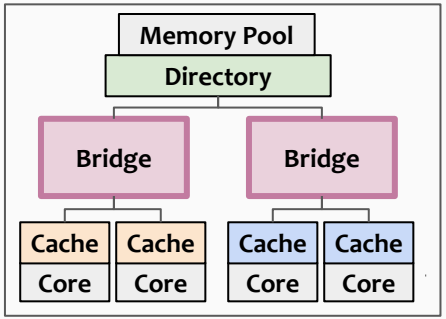
Configuration	Status	# of MC States	RAM usage
2 cores each	**Success**	22 Million	**12 GB**
3 cores each	**Out of Memory**	> 1.4 Billion	**> 1.8 TB**

Full-system liveness models do not scale

Compositional verification

Idea: ***Compositional verification***

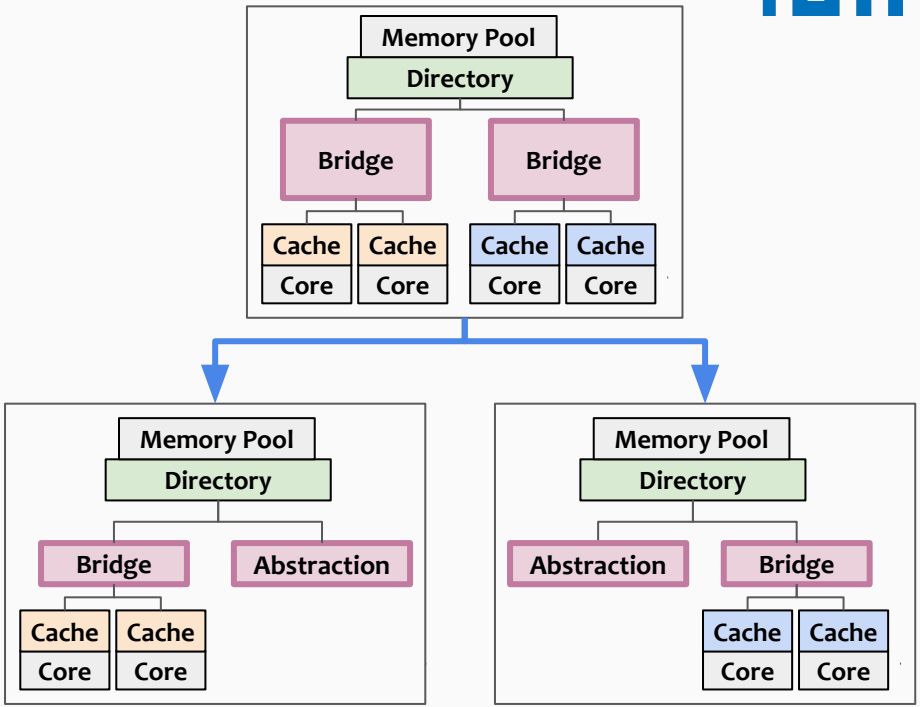
⇒ *Use multiple smaller models*



Compositional verification

Idea: ***Compositional verification***

⇒ *Use multiple smaller models*



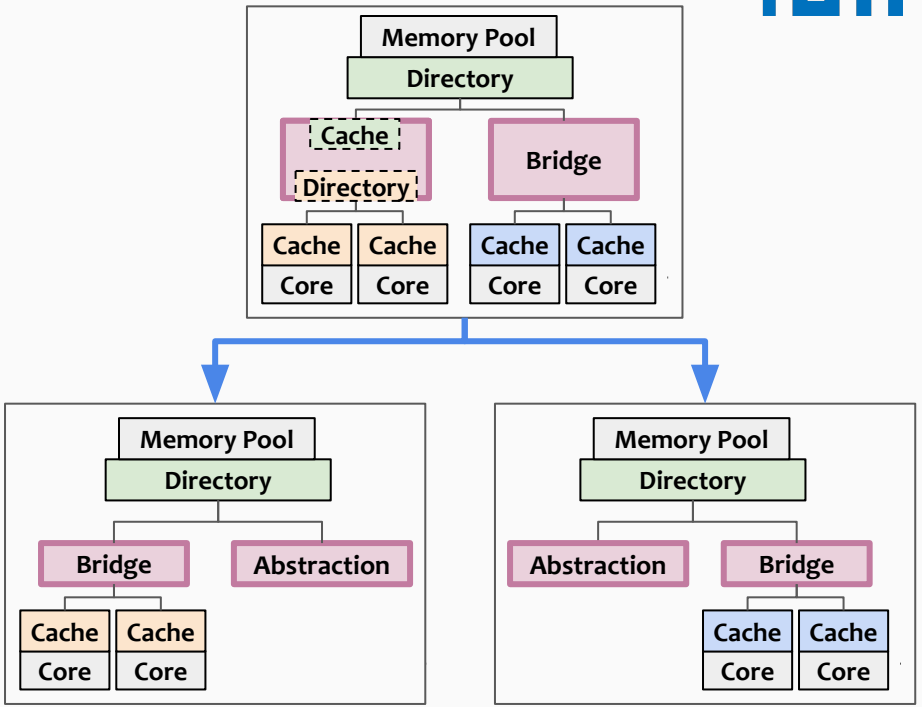
Compositional verification

Idea: ***Compositional verification***

⇒ *Use multiple smaller models*

Insight:

Bridge FSM is a fusion of cache and directory



Compositional verification

Idea: ***Compositional verification***

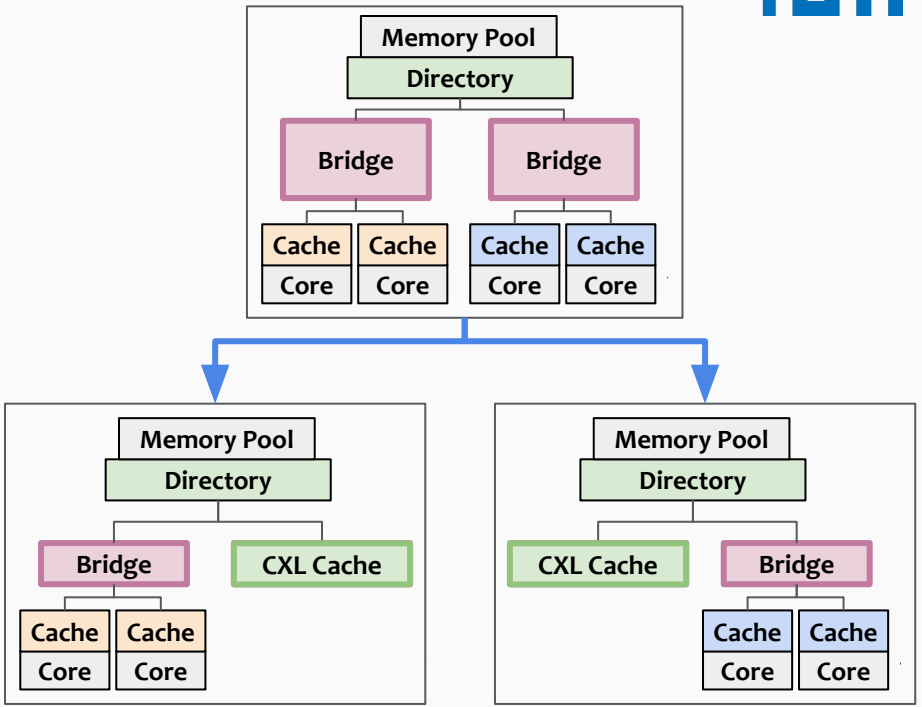
⇒ *Use multiple smaller models*

Insight:

Bridge FSM is a fusion of cache and directory

Approach:

Abstract cluster as a CXL cache



Compositional verification

Idea: ***Compositional verification***

⇒ *Use multiple smaller models*

Insight:

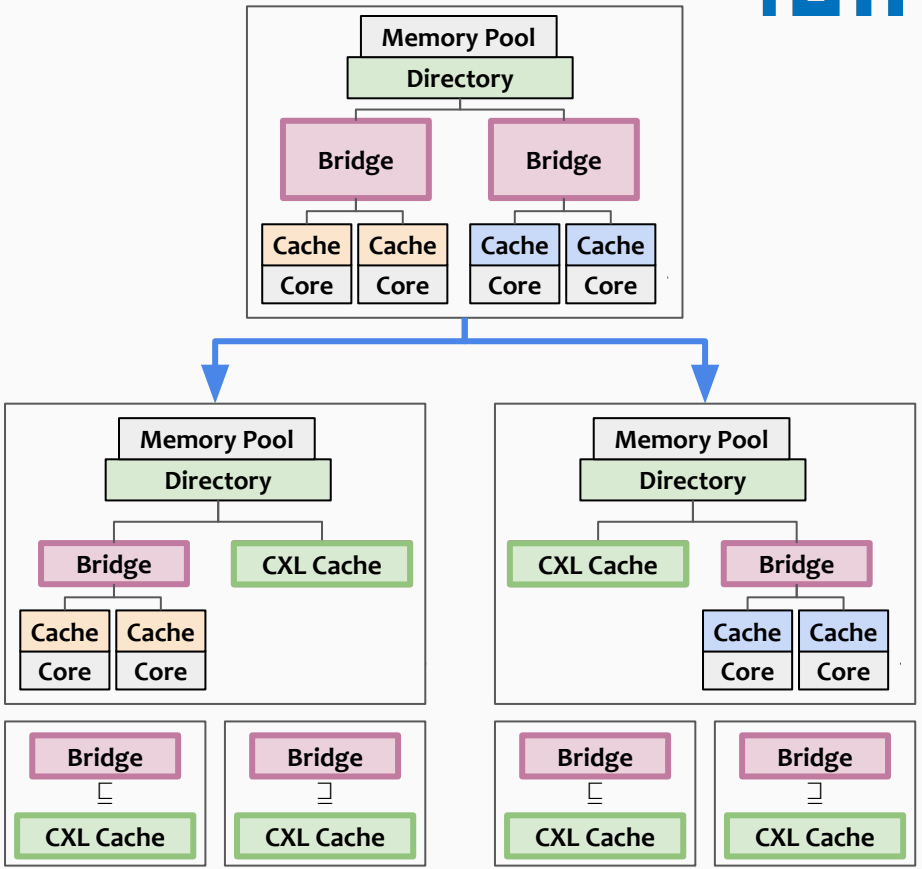
Bridge FSM is a fusion of cache and directory

Approach:

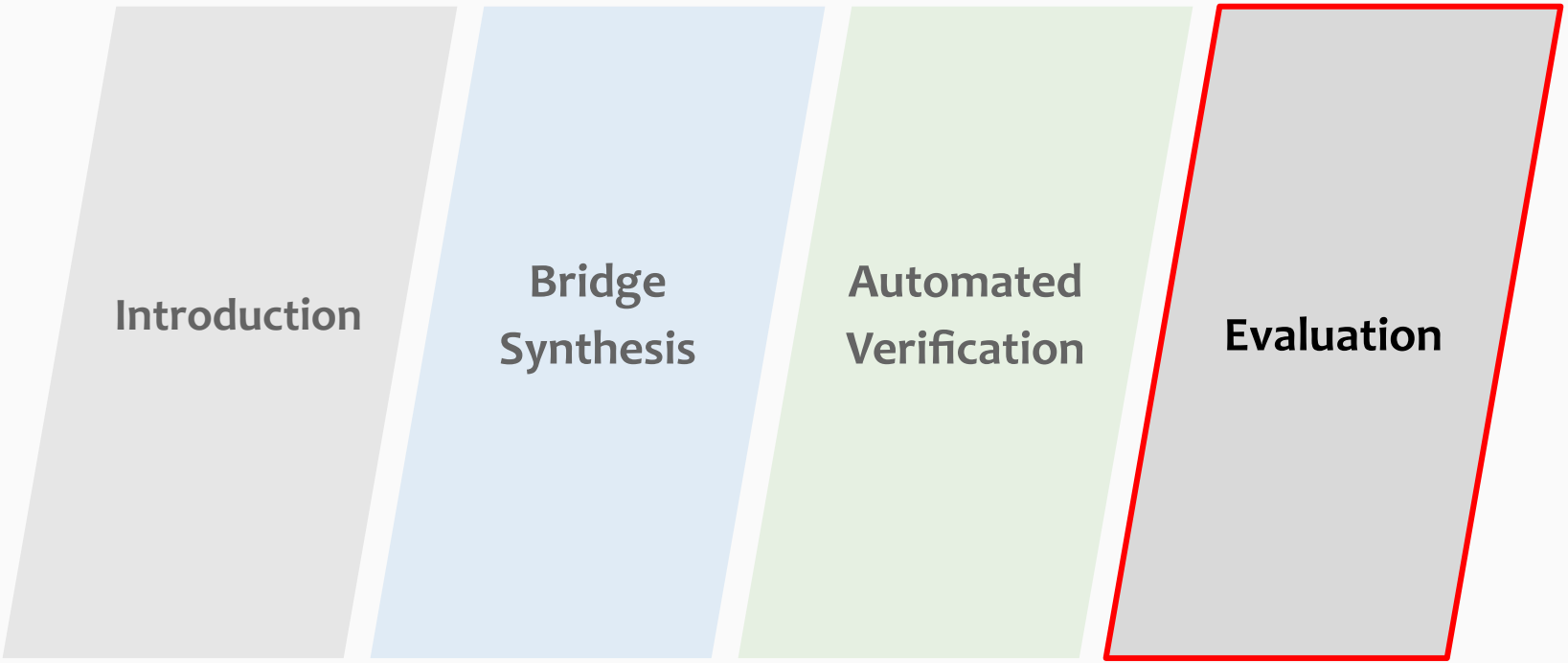
Abstract cluster as a CXL cache

Abstraction Correctness:

Checked by additional models



Outline


Introduction
Bridge Synthesis
Automated Verification
Evaluation

Evaluation: **Goals**

1. **Correctness:**
   Verified safety + liveness using model checkers

2. **Generality:**
   Generated bridges for combinations of protocols:
   CXL, MSI, MESI, MOESI, RCC, RCC-O

3. **Extensibility:**
   Protocols are specified in a few hundred lines:
   CXL ⇒ ~650 SLOC, MESI ⇒ ~350 SLOC, RCC ⇒ ~200 SLOC

Evaluation: **Goals**

1. **Correctness:**
   Verified safety + liveness using model checkers

2. **Generality:**
   Generated bridges for combinations of protocols:
   CXL, MSI, MESI, MOESI, RCC, RCC-O

3. **Extensibility:**
   Protocols are specified in a few hundred lines:
   CXL ⇒ ~650 SLOC, MESI ⇒ ~350 SLOC, RCC ⇒ ~200 SLOC

4. How do the generated **bridges** affect **performance**?

5. Do **Compositional Verification** models improve **scalability**?

Performance evaluation: **Methodology**

Simulate system in gem5:

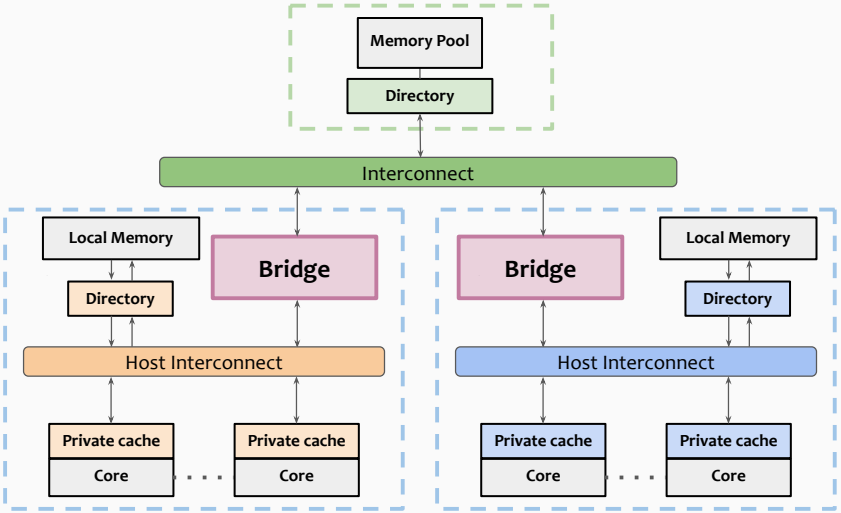
2 host clusters connected to remote memory

Hierarchical setups:

- **MOESI** (gem5)**:** Homogeneous with no bridge
- **MESI-Br:** Homogeneous with vCXLGen bridge
- **CXL-Br:** Heterogeneous with vCXLGen bridge

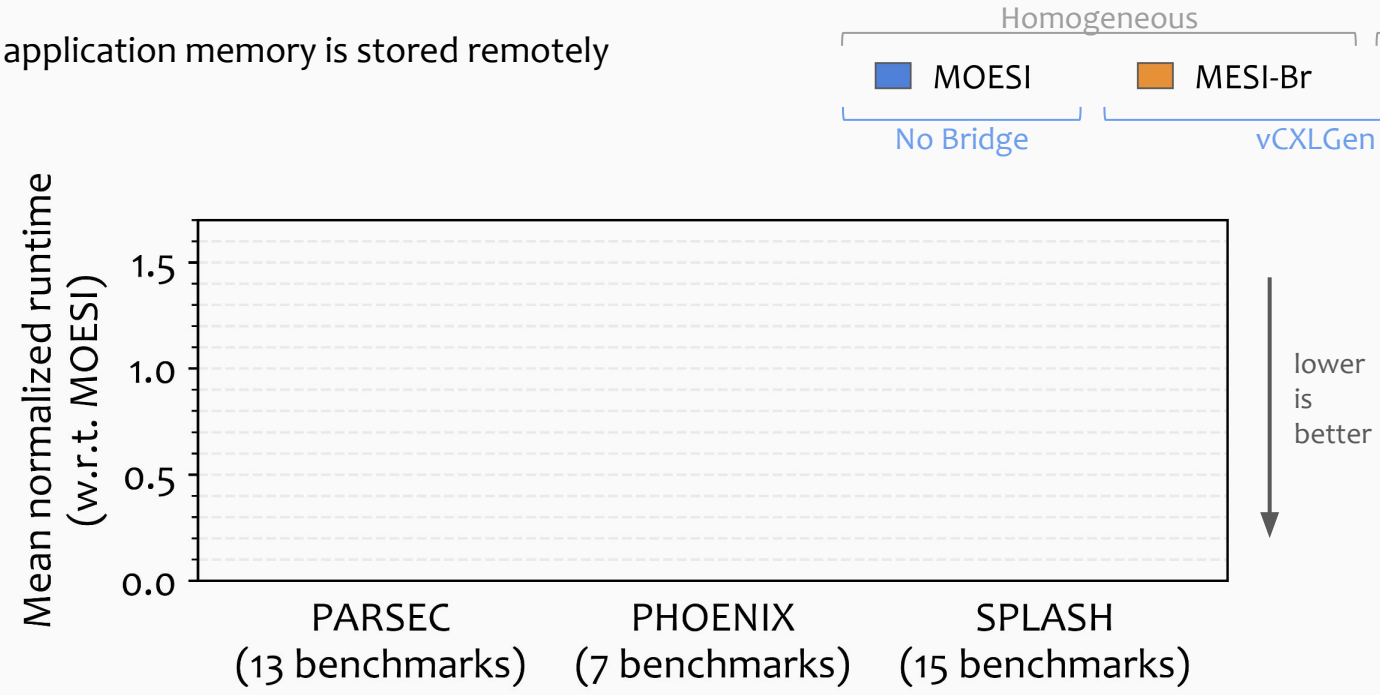
Evaluated scenarios:

1. **Worst-case:** All memory is stored remotely
2. **Distributed KVS:** Only shared memory is stored remotely



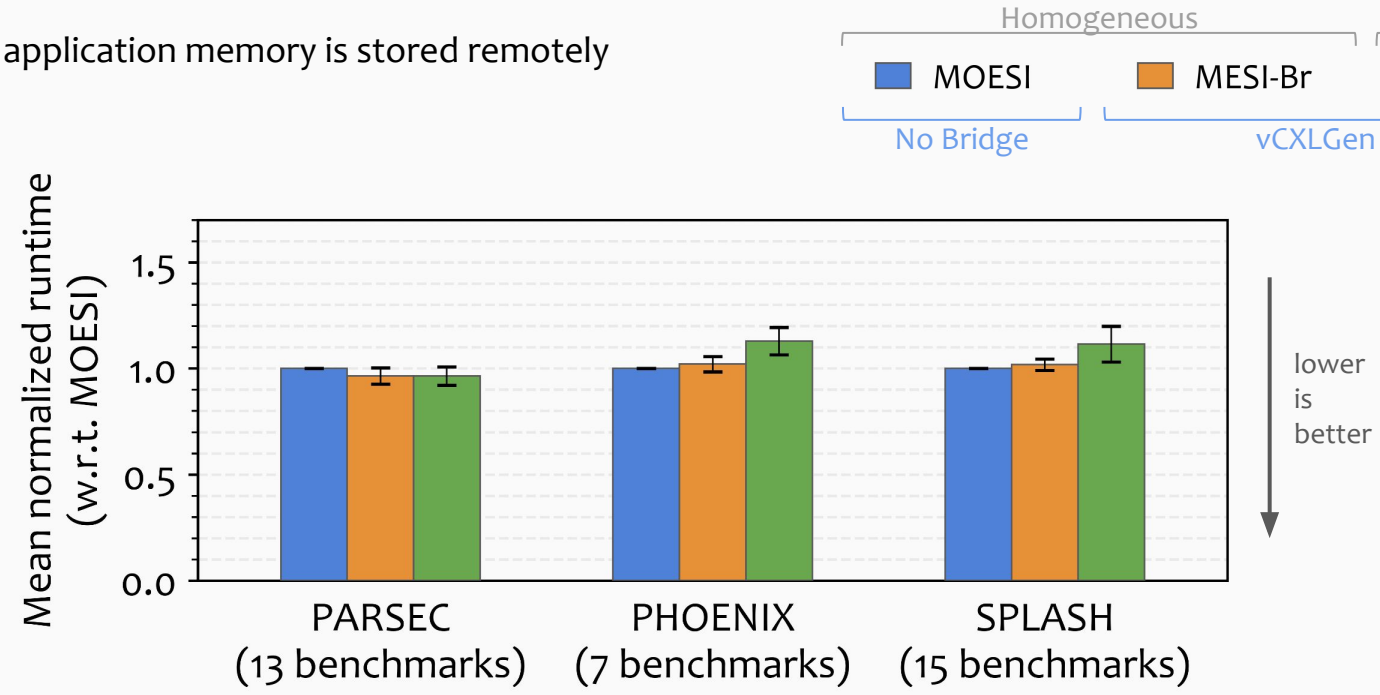
Performance evaluation: **Worst-case overhead**

Setup: All application memory is stored remotely



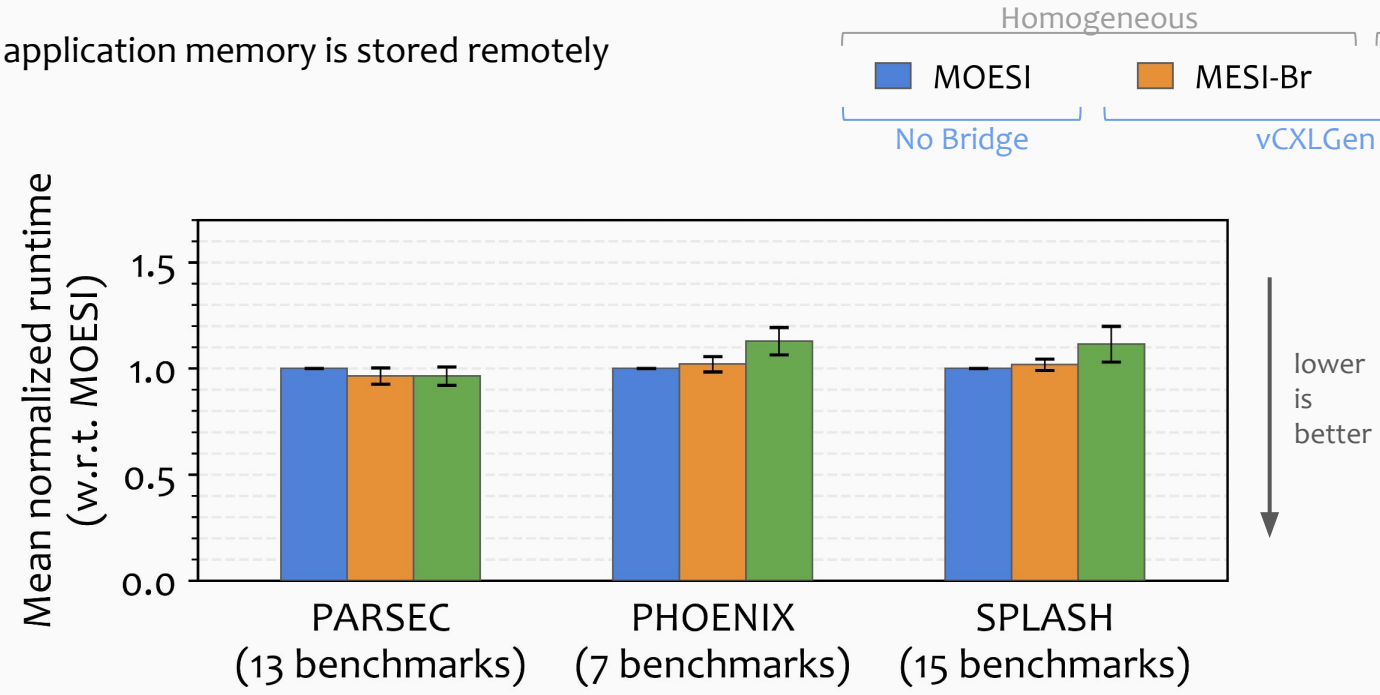
Performance evaluation: **Worst-case overhead**

Setup: All application memory is stored remotely



Performance evaluation: **Worst-case overhead**

Setup: All application memory is stored remotely

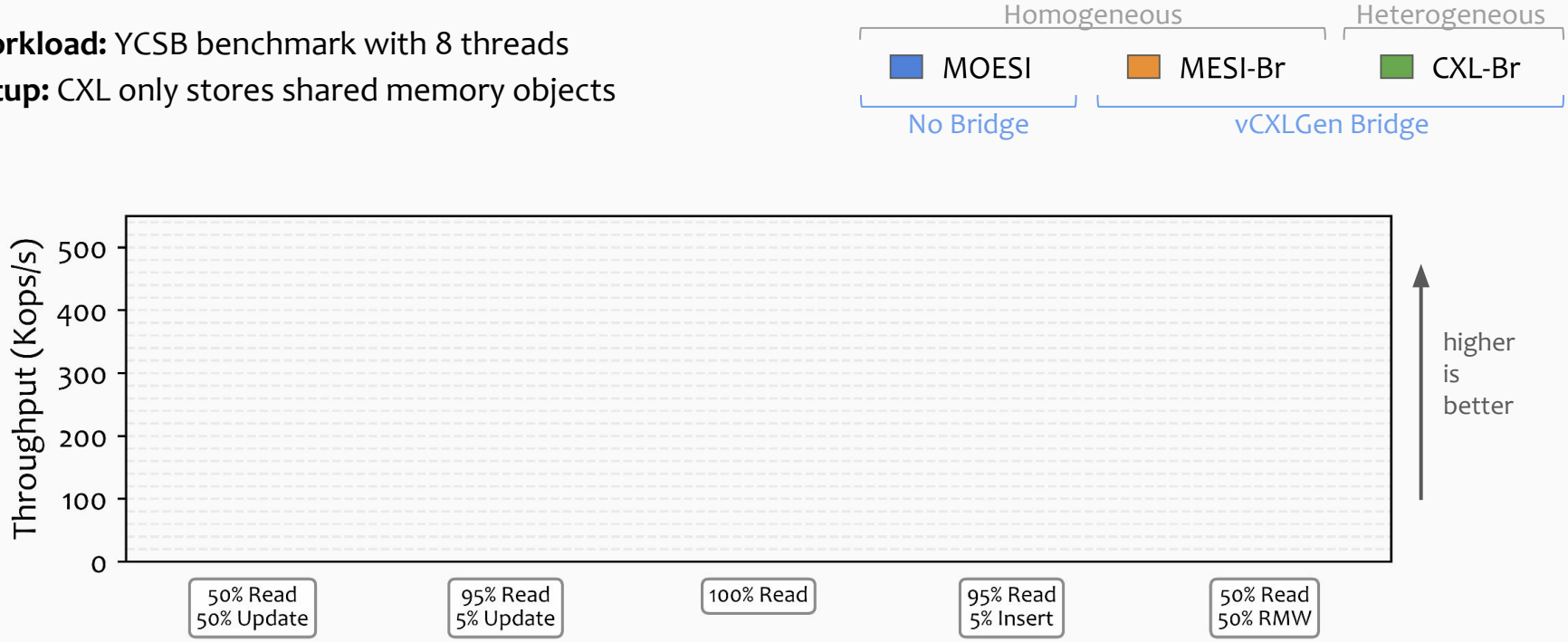


Takeaway: vCXLGen bridges perform **comparably** to a **homogeneous hierarchical MOESI** setup

Performance evaluation: **Distributed KVS**

Workload: YCSB benchmark with 8 threads

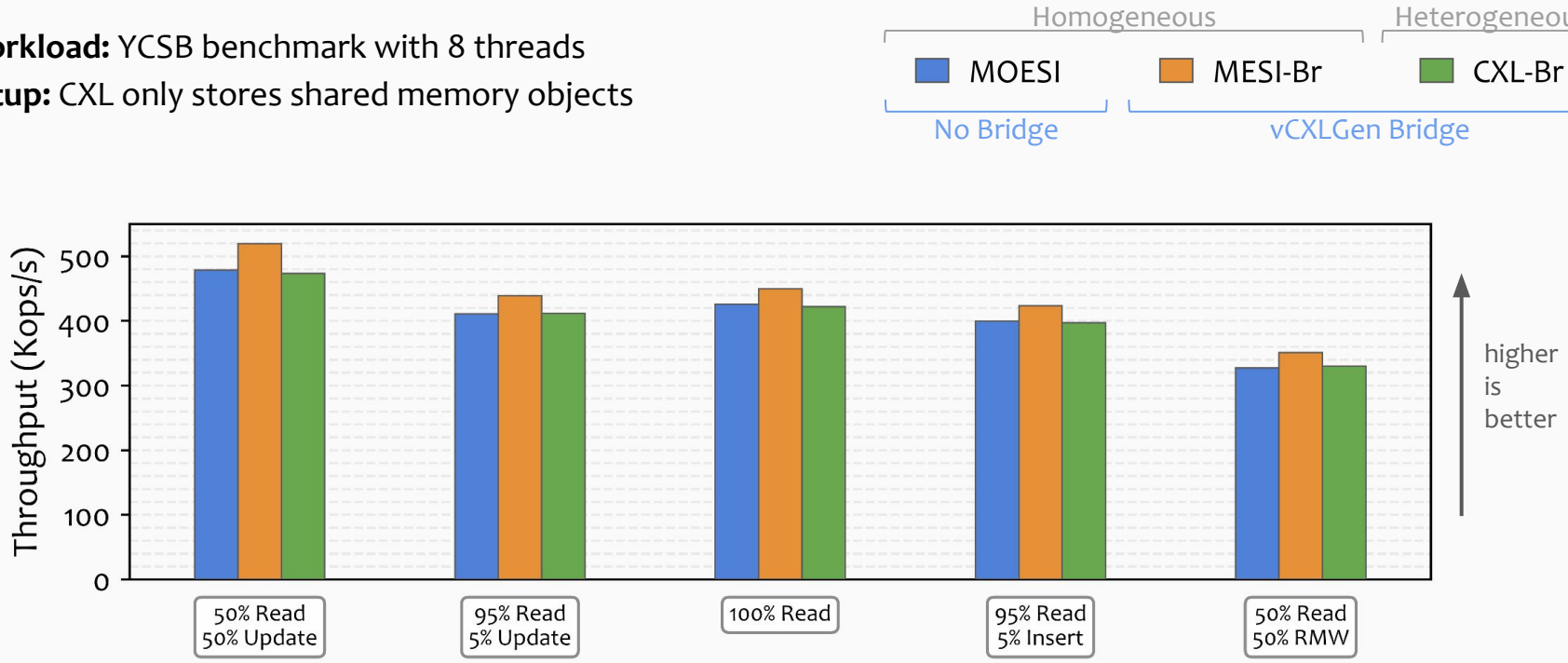
Setup: CXL only stores shared memory objects



Performance evaluation: **Distributed KVS**

Workload: YCSB benchmark with 8 threads

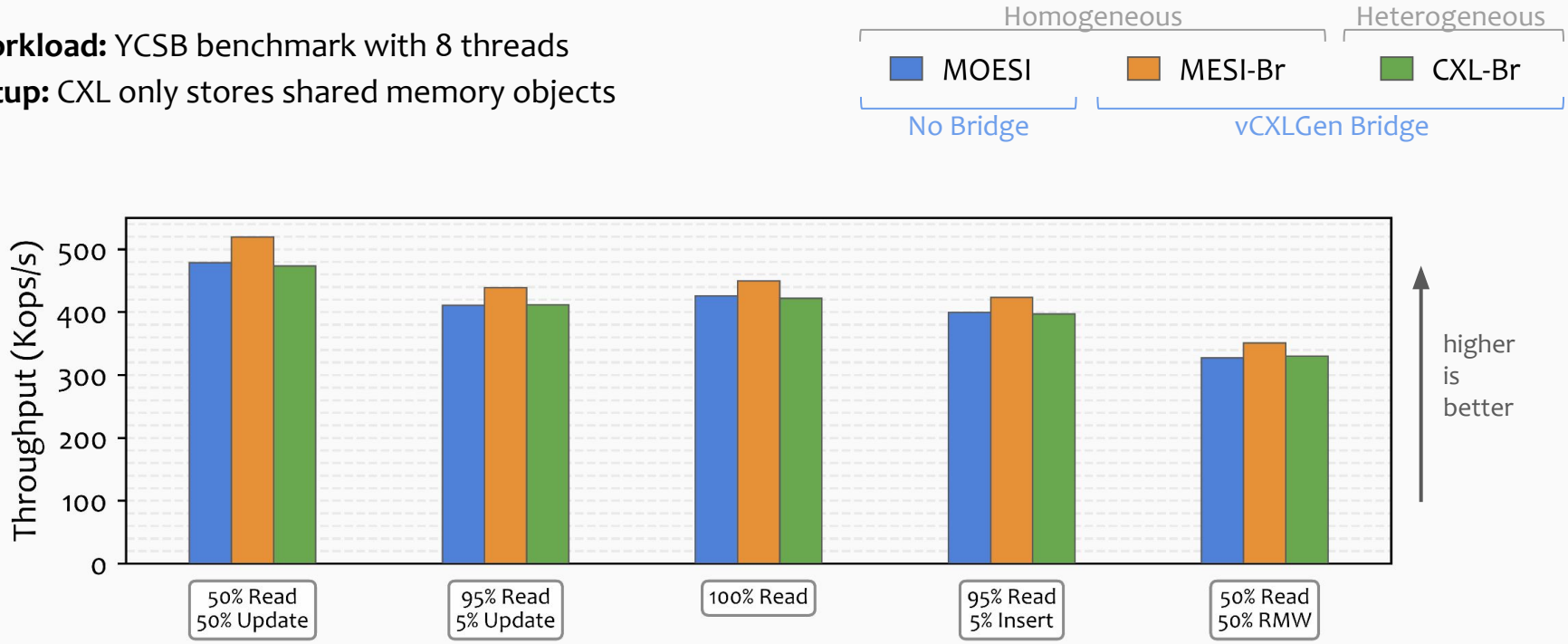
Setup: CXL only stores shared memory objects



Performance evaluation: **Distributed KVS**

Workload: YCSB benchmark with 8 threads

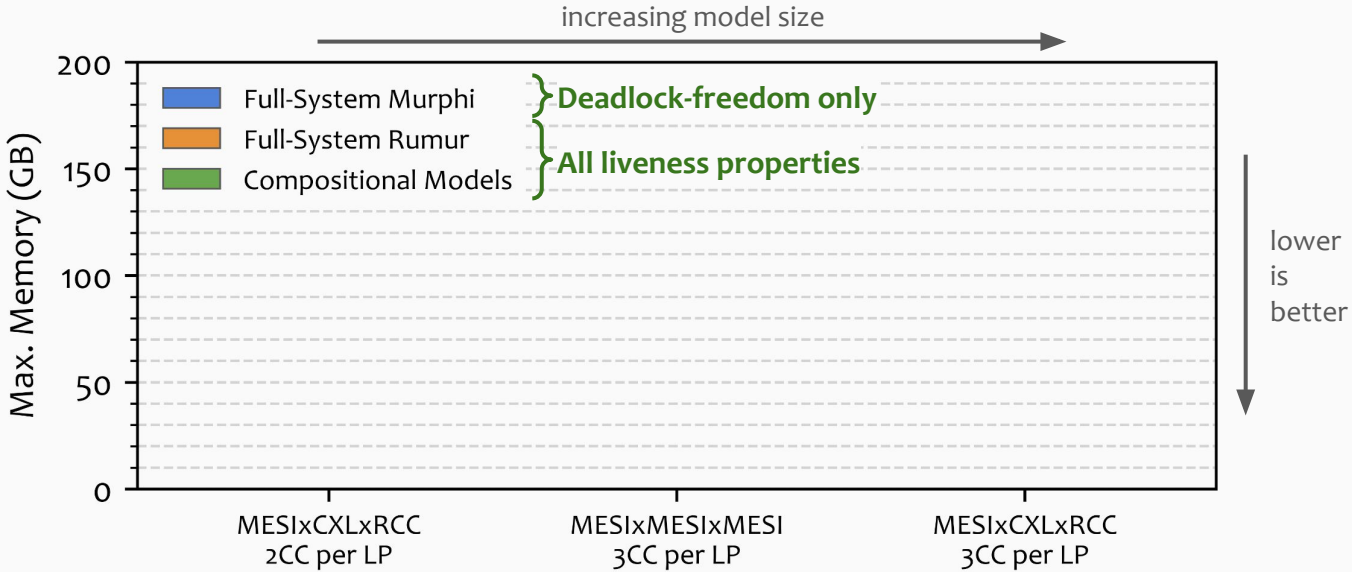
Setup: CXL only stores shared memory objects



Takeaway: vCXLGen-generated bridges **preserve** the **performance** of real-world applications

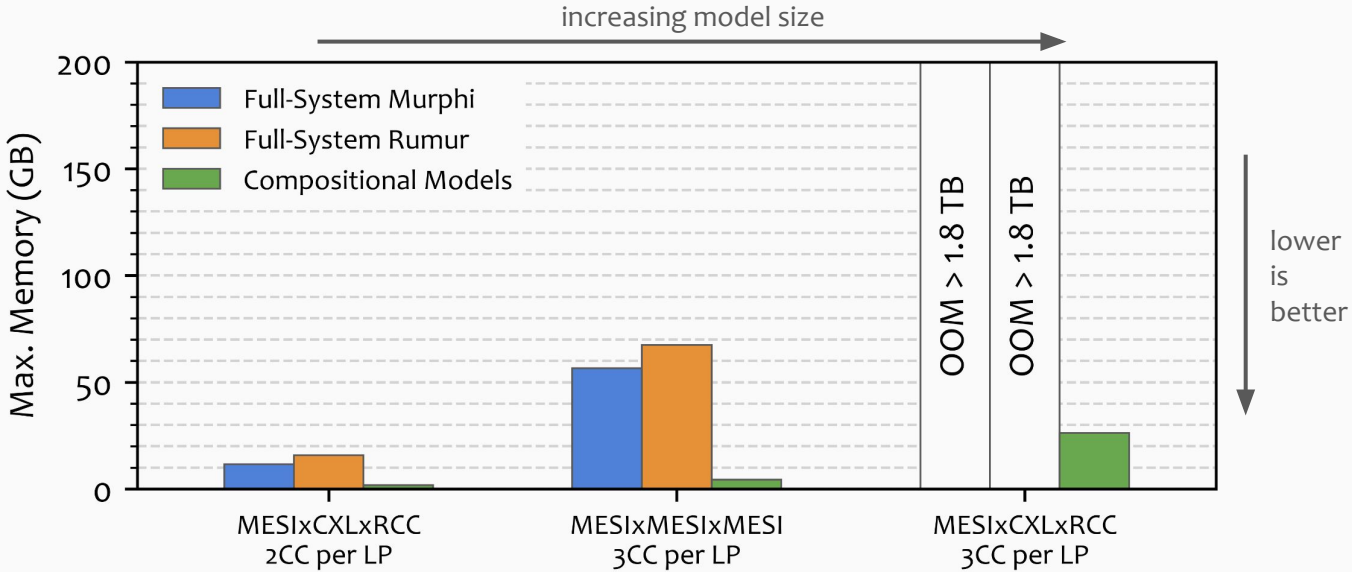
Evaluation: **Verification scalability**

Do vCXLGen's compositional models use less memory?



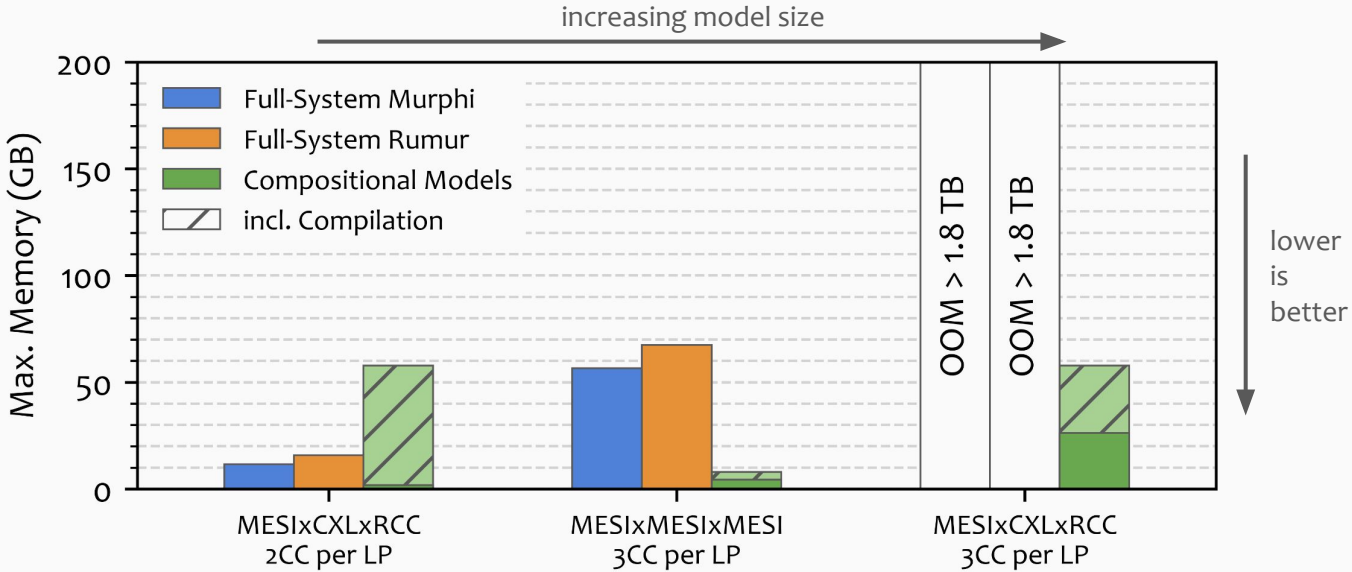
Evaluation: **Verification scalability**

Do vCXLGen's compositional models use less memory?



Evaluation: **Verification scalability**

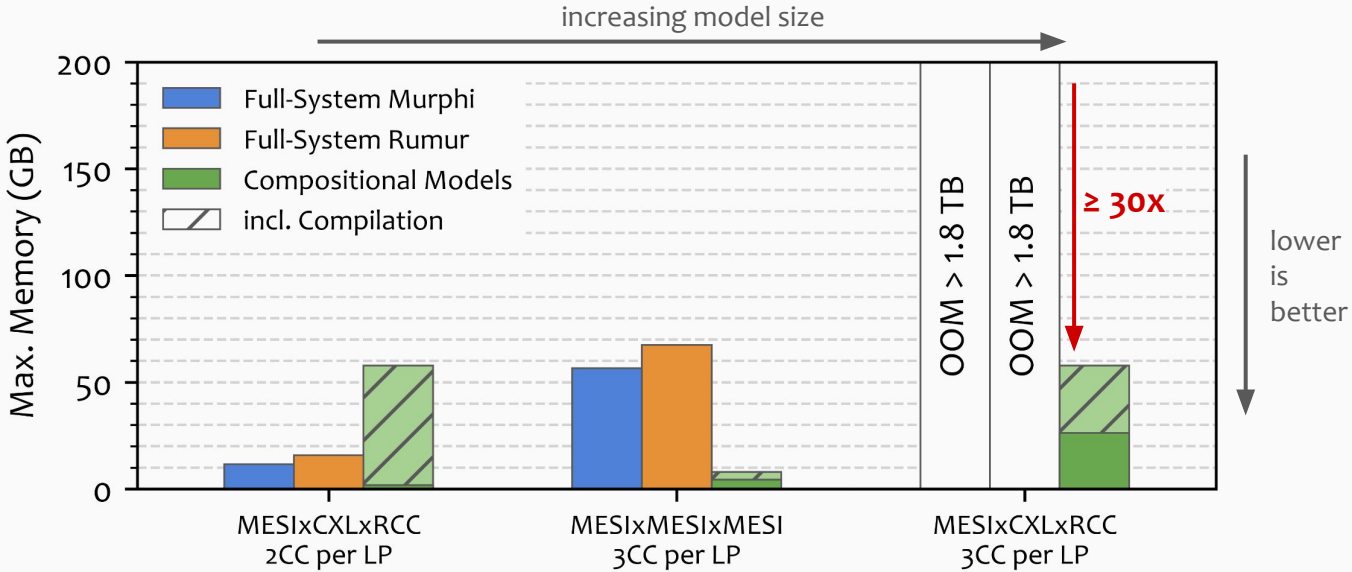
Do vCXLGen's compositional models use less memory?



Observation: *Compilation depends on model complexity*

Evaluation: **Verification scalability**

Do vCXLGen's compositional models use less memory?



Observation: *Compilation depends on model complexity*

Takeaway: Compositional models enable the **verification of larger systems** with **less memory**

Conclusion

Motivation: Current CXL hardware does not support heterogeneous architectures

Problem: How to **automatically** and **correctly** extend heterogeneous architectures for CXL memory?

Solution: vCXLGen — Synthesize *verified* CXL coherence bridges from protocol specifications, that **translate** host protocols to **CXL.mem** & preserve original semantics of heterogeneous architectures

vCXLGen key ideas:

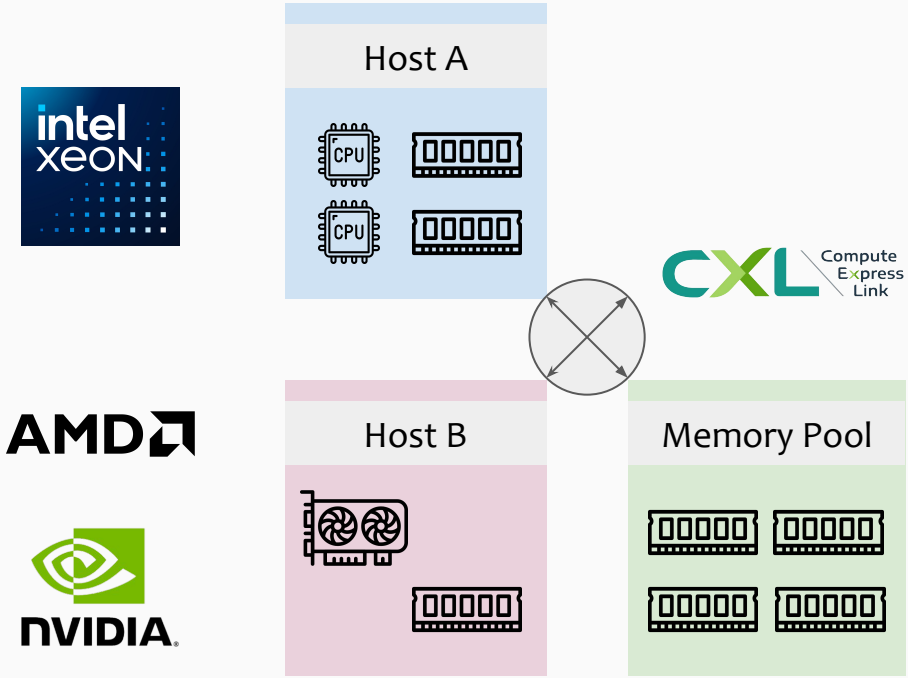
- **Compound State Machine:** combine FSMs of host directory & CXL cache
- **Protocol Semantic Translation:** statically detect *propagation* & *translation* with CXL.mem
- **Compositional Verification:** leverage the bridge's construction principle to decompose verification models

Open problems:

- **Accelerator integration (e.g., GPUs):** CPU-GPU symmetric coherence with CXL 3.0

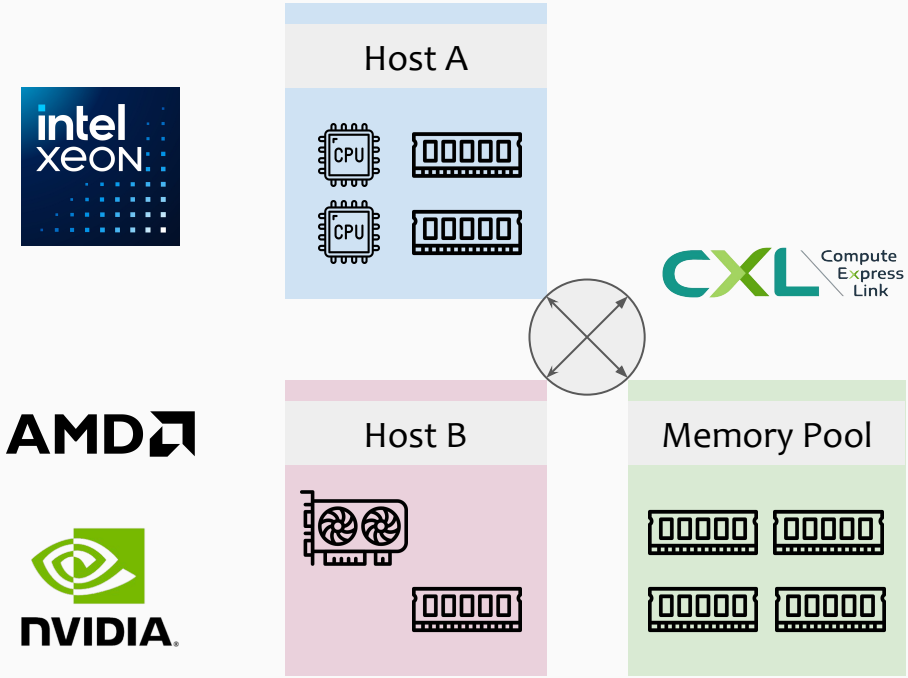
What's next? Accelerators!

E.g., CPU ⟷ GPU **symmetric coherence** with **CXL 3.0**



What's next? Accelerators!

E.g., CPU ⟷ GPU **symmetric coherence** with **CXL 3.0**

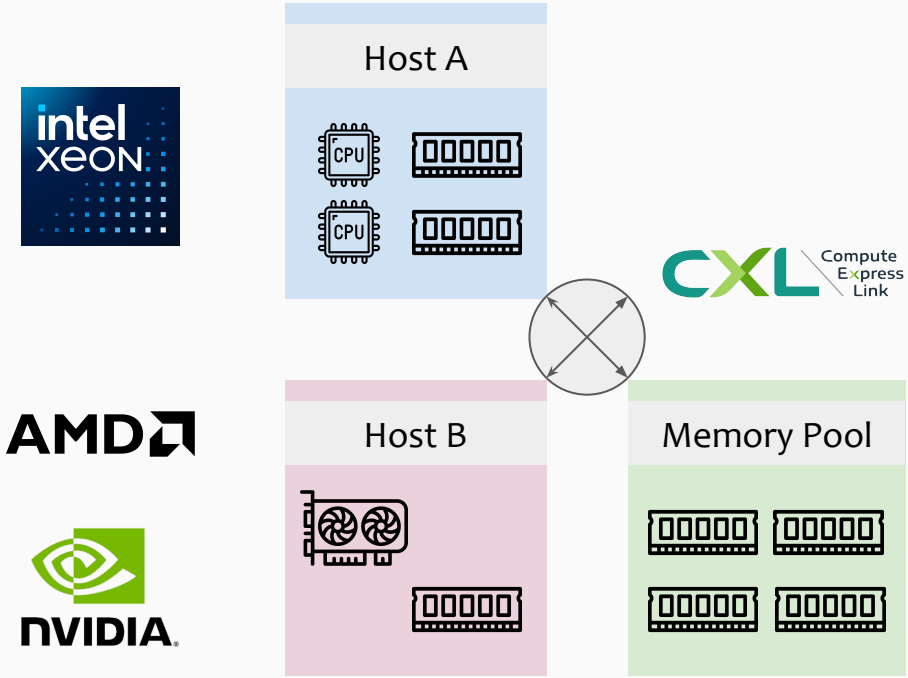


vCXLGen GPU Roadmap:

- ✅ Release-consistency GPU MCMs
- ✅ Update-based GPU protocols
- ✅ Write-through GPU protocols
- ⏳ **Scoped memory models:**

What's next? Accelerators!

E.g., CPU ⟷ GPU **symmetric coherence** with **CXL 3.0**



vCXLGen GPU Roadmap:

- ✅ Release-consistency GPU MCMs
- ✅ Update-based GPU protocols
- ✅ Write-through GPU protocols
- ⌛ **Scoped memory models:**
  - GPUs tag memory operations with a **coherence scope**
    [Block] [GPU] [System]

Backup slides