C³: CXL Coherence Controllers for Heterogeneous Architectures

Anatole Lefort, David Schall, Nicolò Carpentieri, Julian Pritzi,
Soham Chakraborty, Nicolai Oswald, Pramod Bhatotia

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

The Promise of CXL

In a world of scalable data centers

- **Memory** is a **critical resource** for **data centers,**
- But is **not efficiently managed:**

Over-provisioning[1]

Low avg. usage (~40%)[2]

Stranded memory (up to 25%)[3]

- **$$$ wasted** in **energy** and **hardware**

CXL promises **on-demand allocation** from **remote memory chassis**

[1] Reidys et al., *Coach: Exploiting Temporal Patterns for All-Resource Oversubscription in Cloud Platforms*, ASPLOS'25
[2] Li et al., *Pond: CXL-Based Memory Pooling Systems for Cloud Platforms*, ASPLOS'23
[3] Tirmazi et al., *Borg: the next generation,* EuroSys'20

The CXL memory abstraction

Hardware-based main memory disaggregation (remote DRAM pooling)

The CXL memory abstraction

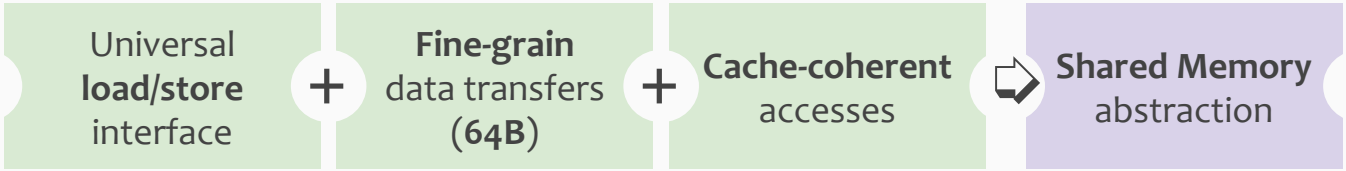
Hardware-based main memory disaggregation (remote DRAM pooling)

CPUs access CXL-based remote memory "*just like a regular DIMM*"**:**

The CXL memory abstraction

Hardware-based main memory disaggregation (remote DRAM pooling)

CPUs access CXL-based remote memory "*just like a regular DIMM*":

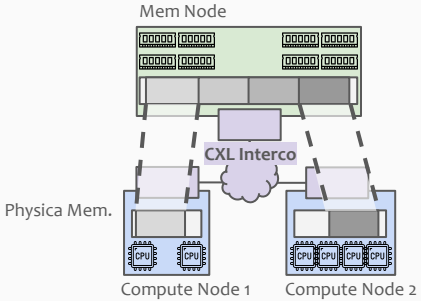


The CXL memory abstraction

Hardware-based main memory disaggregation (remote DRAM pooling)

CPUs access CXL-based remote memory "*just like a regular DIMM*":

Universal **load/store** interface + **Fine-grain** data transfers **(64B)** + **Cache-coherent** accesses ⇨ **Shared Memory** abstraction



An hardware-based abstraction:

- Hosts **map** remote **CXL memory** regions **as physical ranges**
- **Data** moves **transparently** to local CPU **caches**

The CXL memory abstraction

Hardware-based main memory disaggregation (remote DRAM pooling)

CPUs access CXL-based remote memory "*just like a regular DIMM*":

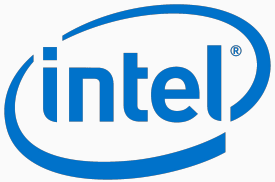
Universal **load/store** interface + **Fine-grain** data transfers (**64B**) + **Cache-coherent** accesses ⇨ **Shared Memory** abstraction

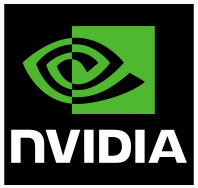
But, Is CXL ready for modern data centers?

No, **Modern Data Centers** are **Heterogeneous!**

x86, ARM, GPUs, Domain-specific accelerators

… and the trend only keeps growing!



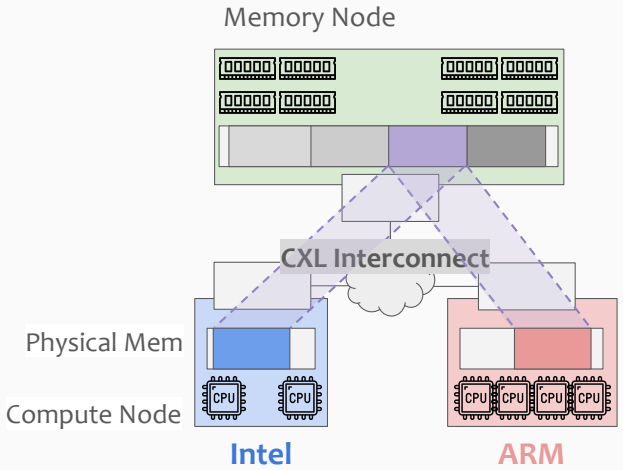


Current **CXL hardware** (& specifications)
do not support heterogeneous architectures

Why is CXL hard for heterogeneous architectures?

Mismatch of CC protocols and memory consistency models (MCMs)

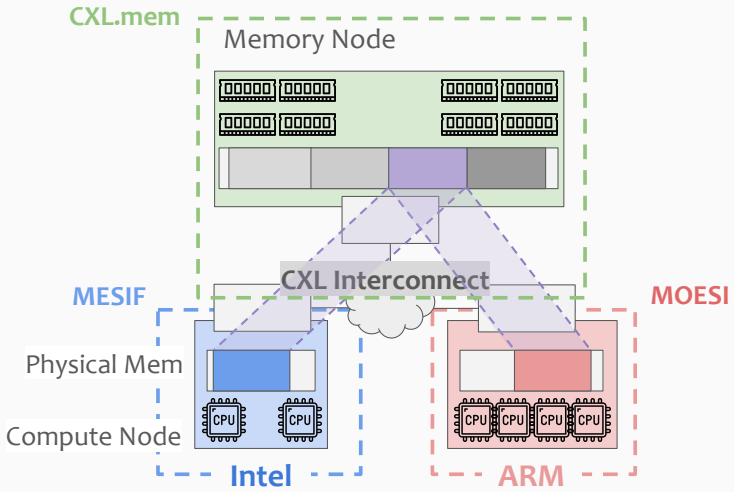
- ➡ **Each host architecture** is tailored for a CC protocol and MCM
- ➡ **CXL** protocol specifications **do not cover** interoperation with heterogeneous architectures



Why is CXL hard for heterogeneous architectures?

Mismatch of CC protocols and memory consistency models (MCMs)

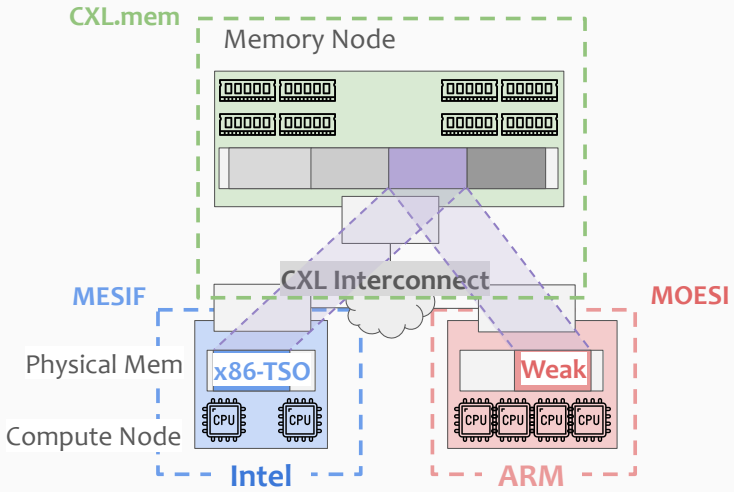
- ➡ **Each host architecture** is tailored for a CC protocol and MCM
- ➡ **CXL** protocol specifications **do not cover** interoperation with heterogeneous architectures



Why is CXL hard for heterogeneous architectures?

Mismatch of CC protocols and memory consistency models (MCMs)

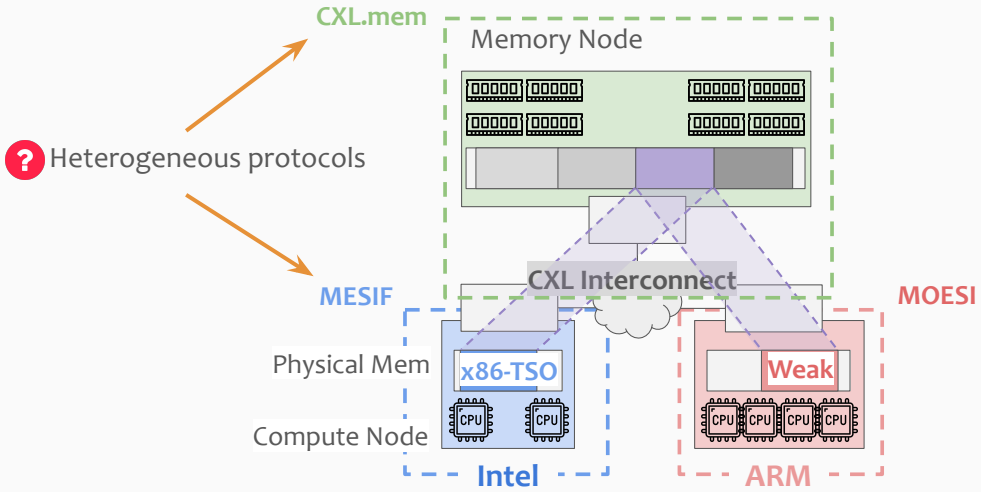
- ➡ **Each host architecture** is tailored for a CC protocol and MCM
- ➡ **CXL** protocol specifications **do not cover** interoperation with heterogeneous architectures



Why is CXL hard for heterogeneous architectures?

Mismatch of CC protocols and memory consistency models (MCMs)

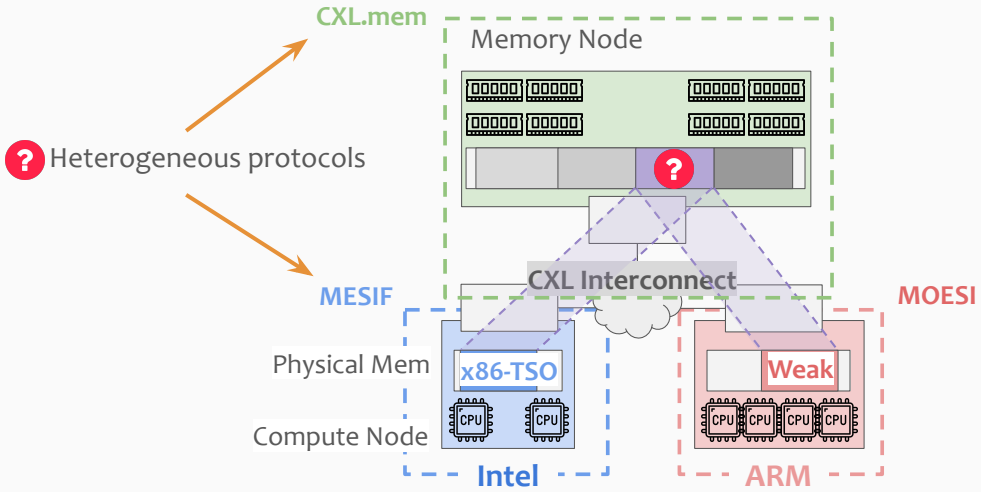
- ➡ **Each host architecture** is tailored for a CC protocol and MCM
- ➡ **CXL** protocol specifications **do not cover** interoperation with heterogeneous architectures



Why is CXL hard for heterogeneous architectures?

Mismatch of CC protocols and memory consistency models (MCMs)

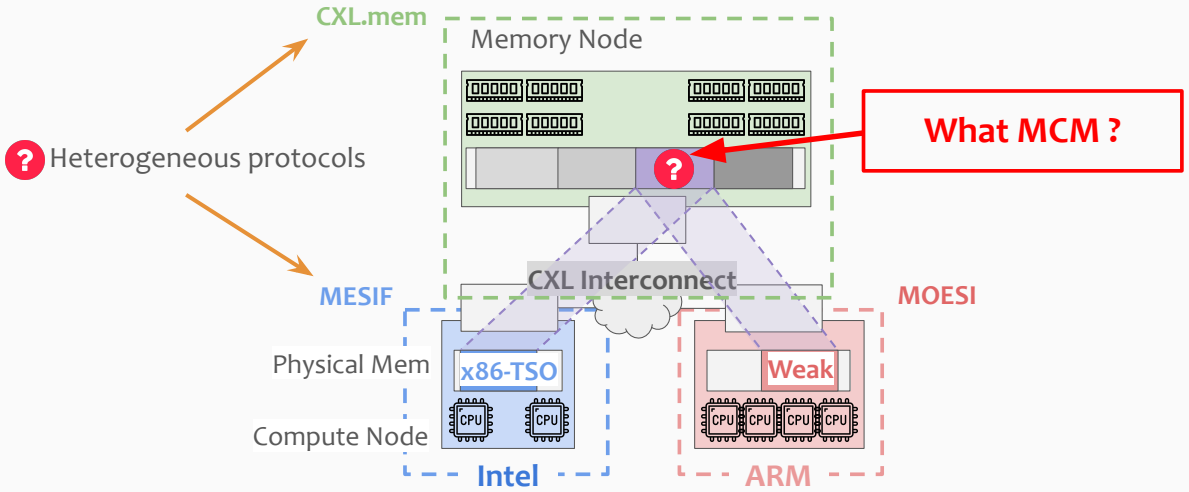
- ➡ **Each host architecture** is tailored for a CC protocol and MCM
- ➡ **CXL** protocol specifications **do not cover** interoperation with heterogeneous architectures



Why is CXL hard for heterogeneous architectures?

Mismatch of CC protocols and memory consistency models (MCMs)

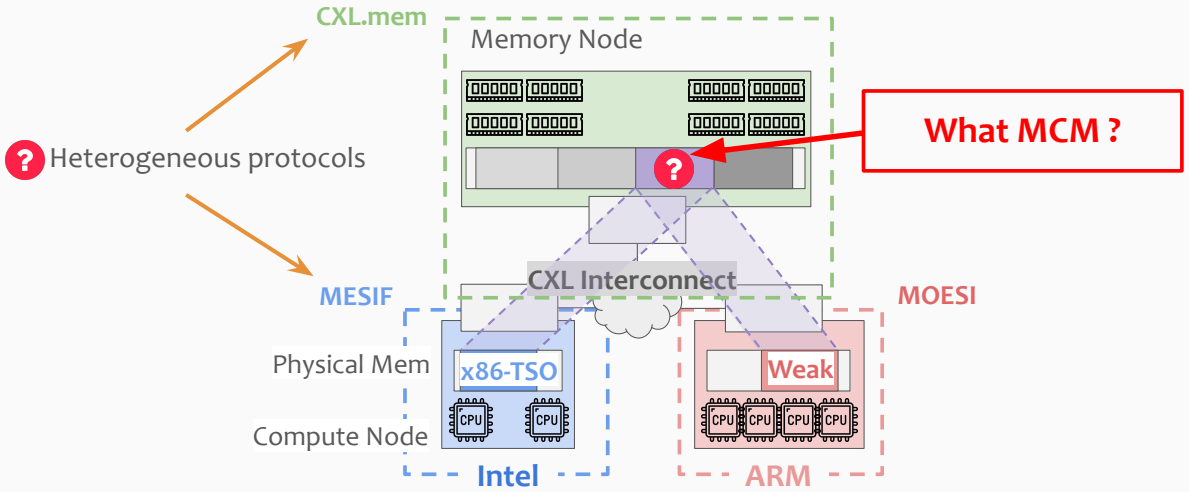
- ➡ **Each host architecture** is tailored for a CC protocol and MCM
- ➡ **CXL** protocol specifications **do not cover** interoperation with heterogeneous architectures



Why is CXL hard for heterogeneous architectures?

Mismatch of CC protocols and memory consistency models (MCMs)

- ➡ **Each host architecture** is tailored for a CC protocol and MCM
- ➡ **CXL** protocol specifications **do not cover** interoperation with heterogeneous architectures

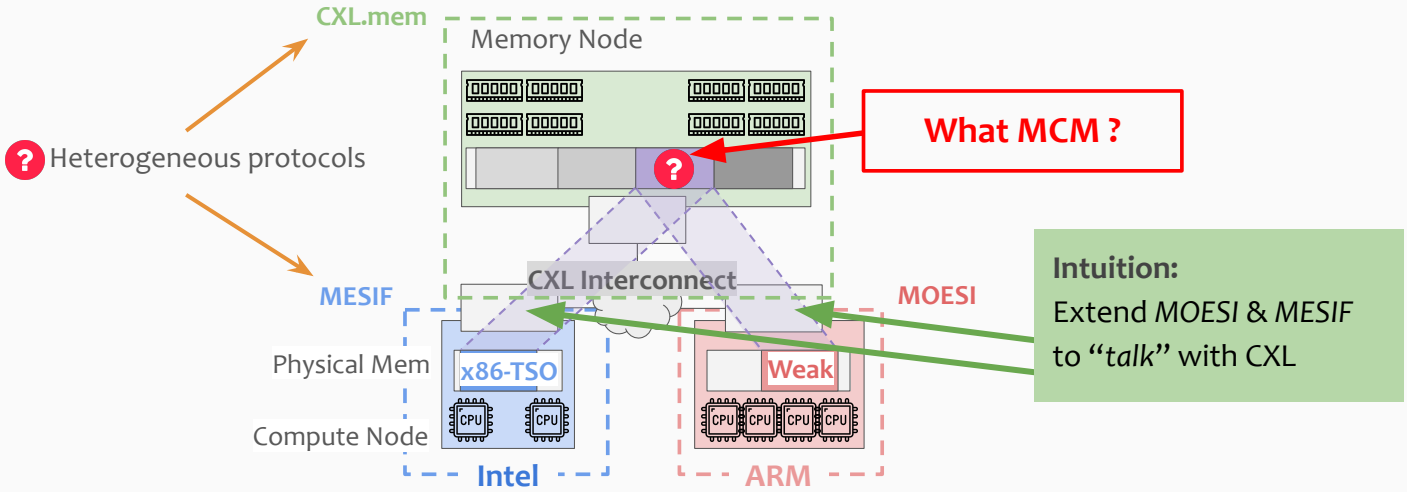


How to **interoperate** vendor-specific **cache-coherence** for CXL, and **avoid memory consistency bugs**?

Why is CXL hard for heterogeneous architectures?

Mismatch of CC protocols and memory consistency models (MCMs)

- ➡ **Each host architecture** is tailored for a CC protocol and MCM
- ➡ **CXL** protocol specifications **do not cover** interoperation with heterogeneous architectures



How to **interoperate** vendor-specific **cache-coherence** for CXL, and **avoid memory consistency bugs**?

The Challenges

How to extend heterogeneous architectures for CXL shared memory?

➪ Vendors must re-design host-specific coherence protocols for CXL interoperability

1 Diverse

Many vendor-specific **coherence protocols** (CC) and **memory consistency models** (MCMs)

2 Semantic gap

Manual & ad-hoc **translation** of **host-specific** coherence requests to CXL protocol

3 Complex protocols

Large state machines (~20 states, ~40 transitions) **tightly coupled** to host's MCMs

Problem Statement: How to systematically and correctly extend heterogeneous architectures for CXL memory?

Our Proposal: C³

C³: CXL Coherence Controllers for **Heterogeneous Architectures**

Our Solution: Pluggable coherence bridges controllers to translate host-specific coherence protocols to CXL and to preserve original memory semantics

1 Genericity

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

2 Non-intrusivity

Avoid internal modifications to **hosts (coherence** & **MCM),** or to **CXL**

3 Correctness

Preserve *by construction* **original host MCMs**

Overview: C³ bridges for CXL interoperability

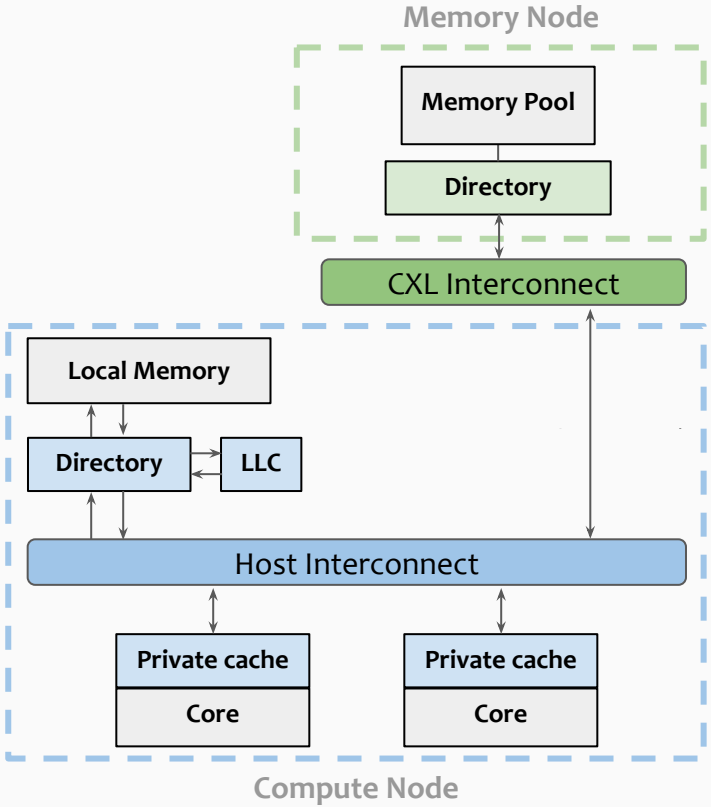
Abstraction: C³ sits at the interface between hosts and CXL

Key ideas:

- C³ logic to perform **semantic** & **context-aware translation** between host and CXL protocol *requests/responses*
- **Preserve host memory orderings**, regardless of other heterogeneous hosts sharing the same CXL region
- **No change required** to host protocol, MCM, CXL protocol, or compiled program binaries

Contributions:

- **Systematic** and **generic methodology** to build host-specific C³ controllers from protocol specifications
- **Correct-by-construction** (formally verified)



Overview: C³ bridges for CXL interoperability

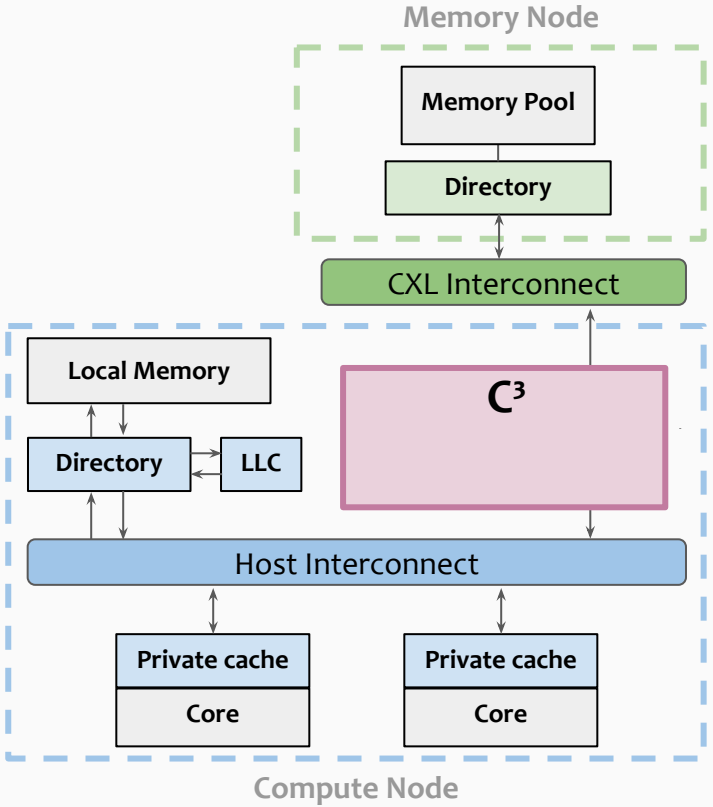
Abstraction: C³ sits at the interface between hosts and CXL

Key ideas:

- C³ logic to perform **semantic** & **context-aware translation** between host and CXL protocol *requests/responses*
- **Preserve host memory orderings**, regardless of other heterogeneous hosts sharing the same CXL region
- **No change required** to host protocol, MCM, CXL protocol, or compiled program binaries

Contributions:

- **Systematic** and **generic methodology** to build host-specific C³ controllers from protocol specifications
- **Correct-by-construction** (formally verified)



Overview: C³ bridges for CXL interoperability

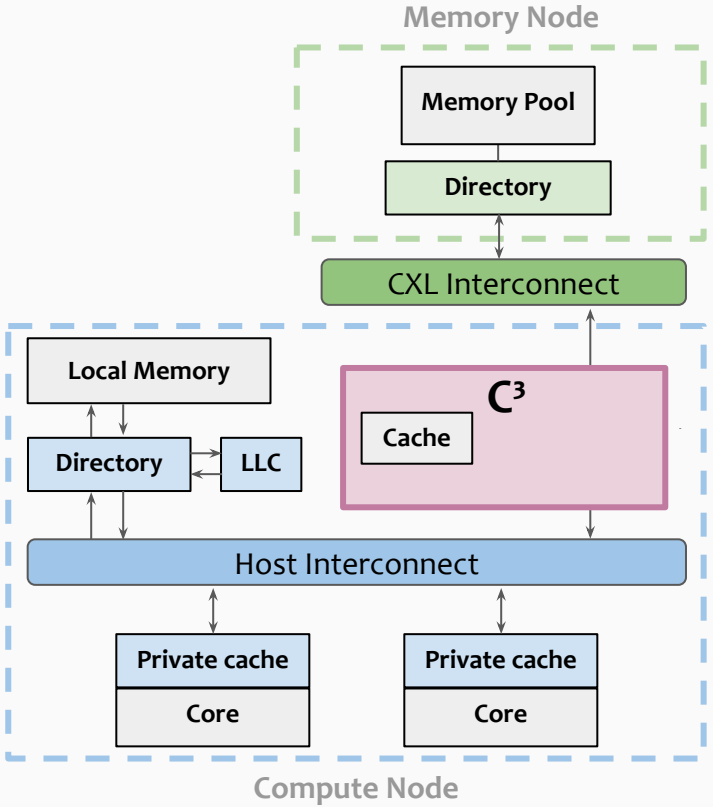
Abstraction: C³ sits at the interface between hosts and CXL

Key ideas:

- C³ logic to perform **semantic** & **context-aware translation** between host and CXL protocol *requests/responses*
- **Preserve host memory orderings**, regardless of other heterogeneous hosts sharing the same CXL region
- **No change required** to host protocol, MCM, CXL protocol, or compiled program binaries

Contributions:

- **Systematic** and **generic methodology** to build host-specific C³ controllers from protocol specifications
- **Correct-by-construction** (formally verified)



Overview: C³ bridges for CXL interoperability

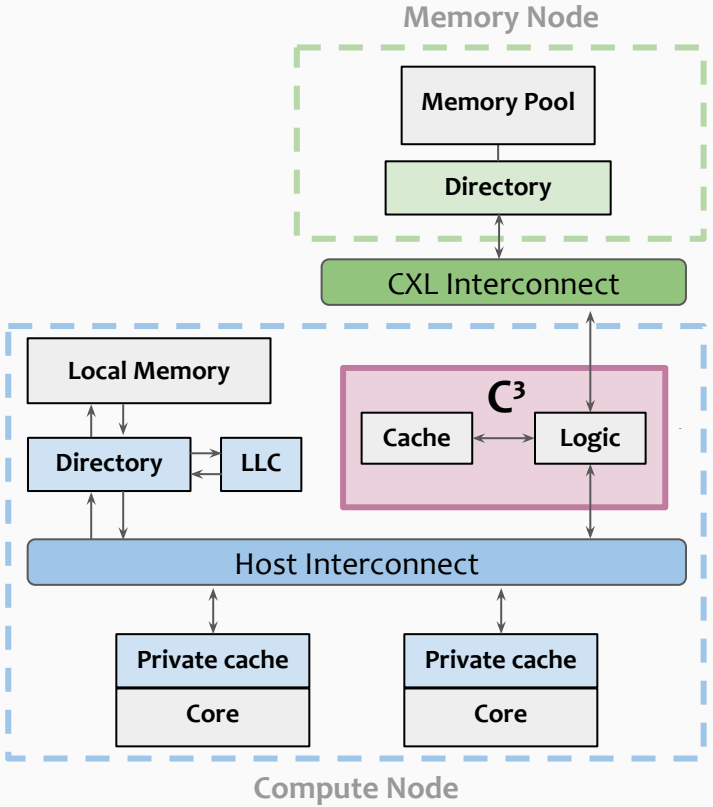
Abstraction: C³ sits at the interface between hosts and CXL

Key ideas:

- C³ logic to perform **semantic** & **context-aware translation** between host and CXL protocol *requests/responses*
- **Preserve host memory orderings**, regardless of other heterogeneous hosts sharing the same CXL region
- **No change required** to host protocol, MCM, CXL protocol, or compiled program binaries

Contributions:

- **Systematic** and **generic methodology** to build host-specific C³ controllers from protocol specifications
- **Correct-by-construction** (formally verified)



Overview: C³ bridges for CXL interoperability

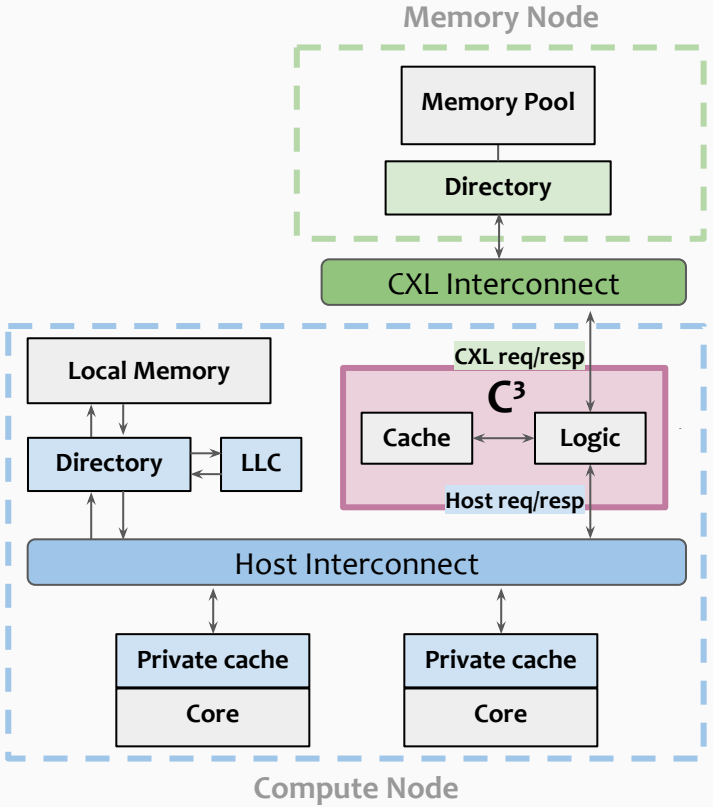
Abstraction: C³ sits at the interface between hosts and CXL

Key ideas:

- C³ logic to perform **semantic** & **context-aware translation** between host and CXL protocol *requests/responses*
- **Preserve host memory orderings**, regardless of other heterogeneous hosts sharing the same CXL region
- **No change required** to host protocol, MCM, CXL protocol, or compiled program binaries

Contributions:

- **Systematic** and **generic methodology** to build host-specific C³ controllers from protocol specifications
- **Correct-by-construction** (formally verified)

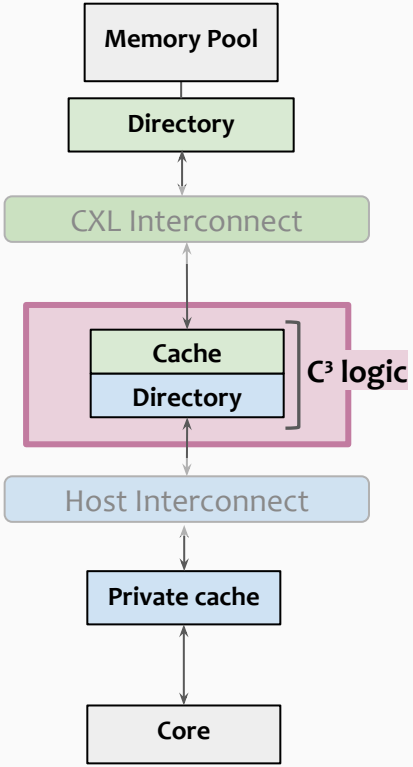


Design: **The C³ *Compound State Machine***

C³ ***combines*** the FSMs from host **directory** & CXL **cache**

C³ ***appears*** as a **directory** to hosts, and **caching client** to CXL

C³ **composes transactions** between **host** protocol and **CXL.mem**

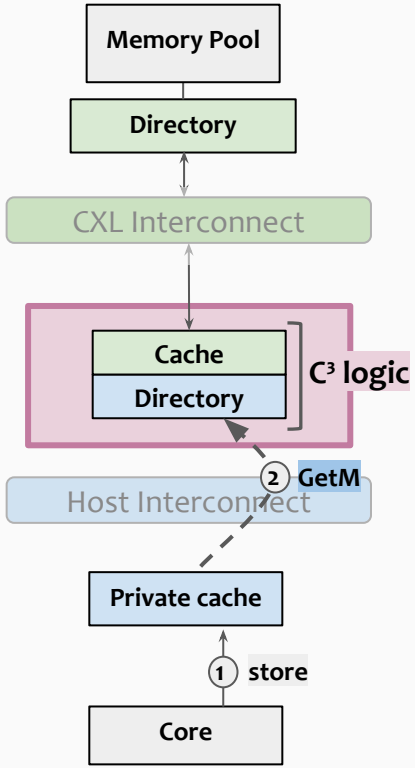


Design: **The C³ *Compound State Machine***

C³ ***combines*** the FSMs from host **directory** & CXL **cache**

C³ ***appears*** as a **directory** to hosts, and **caching client** to CXL

C³ **composes transactions** between **host** protocol and **CXL.mem**



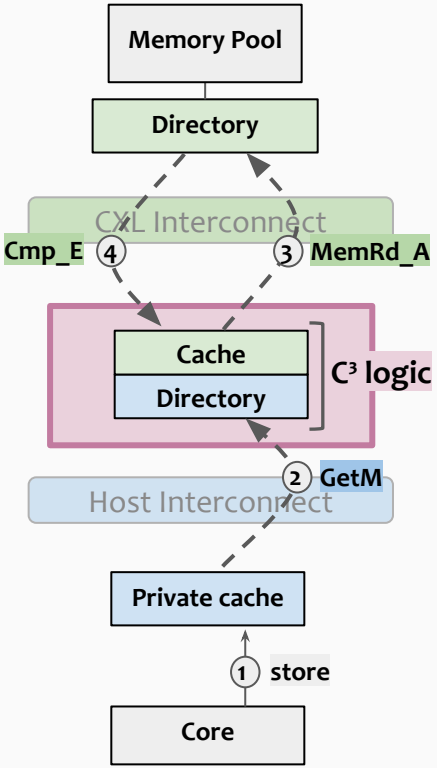
Design: **The C³ *Compound State Machine***

C³ ***combines*** the FSMs from host **directory** & CXL **cache**

C³ ***appears*** as a **directory** to hosts, and **caching client** to CXL

C³ **composes transactions** between **host** protocol and **CXL.mem**

- C³ **propagates** requests that produce **globally visible memory updates**



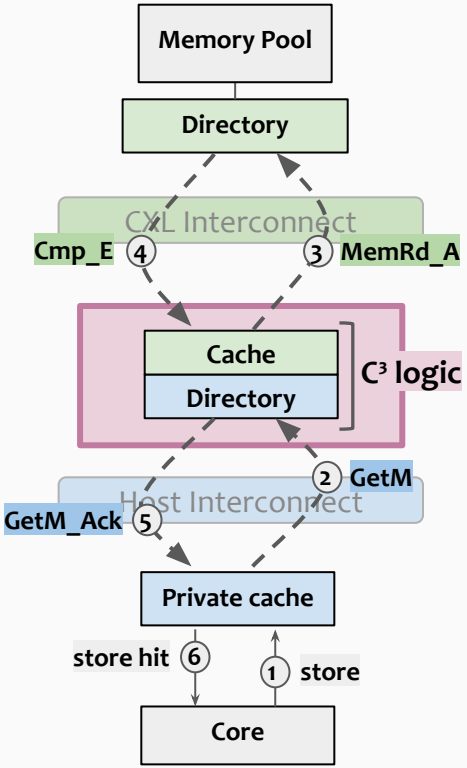
Design: **The C³ *Compound State Machine***

C³ ***combines*** the FSMs from host **directory** & CXL **cache**

C³ ***appears*** as a **directory** to hosts, and **caching client** to CXL

C³ **composes transactions** between **host** protocol and **CXL.mem**

- C³ **propagates** requests that produce **globally visible memory updates**



Design: The C³ *Compound State Machine*

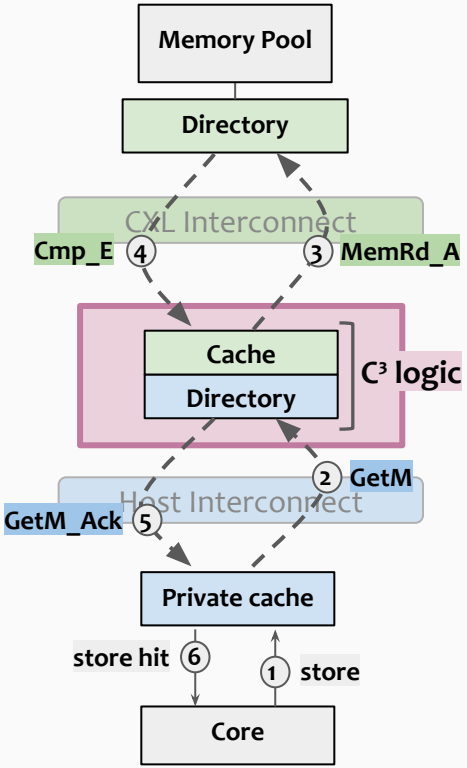
C³ ***combines*** the FSMs from host **directory** & CXL **cache**

C³ ***appears*** as a **directory** to hosts, and **caching client** to CXL

C³ **composes transactions** between **host** protocol and **CXL.mem**

- C³ **propagates** requests that produce **globally visible memory updates**

Example:
- Host `GetM` request (②- ⑤) **encapsulates** the CXL transaction (③- ④)



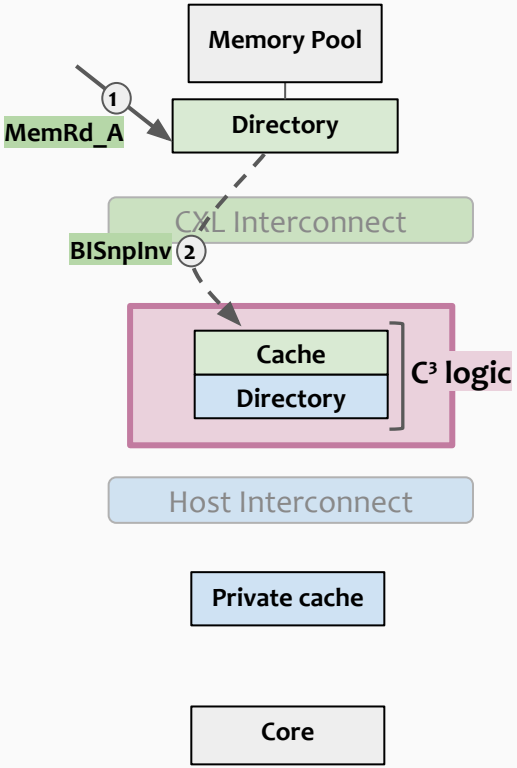
Design: **The C³ *Compound State Machine***

C³ ***combines*** the FSMs from host **directory** & CXL **cache**

C³ ***appears*** as a **directory** to hosts, and **caching client** to CXL

C³ **composes transactions** between **host** protocol and **CXL.mem**

- C³ **propagates** requests that produce **globally visible memory updates**



Example:
- Host `GetM` request (②- ⑤) **encapsulates** the CXL transaction (③- ④)

Design: The C³ *Compound State Machine*

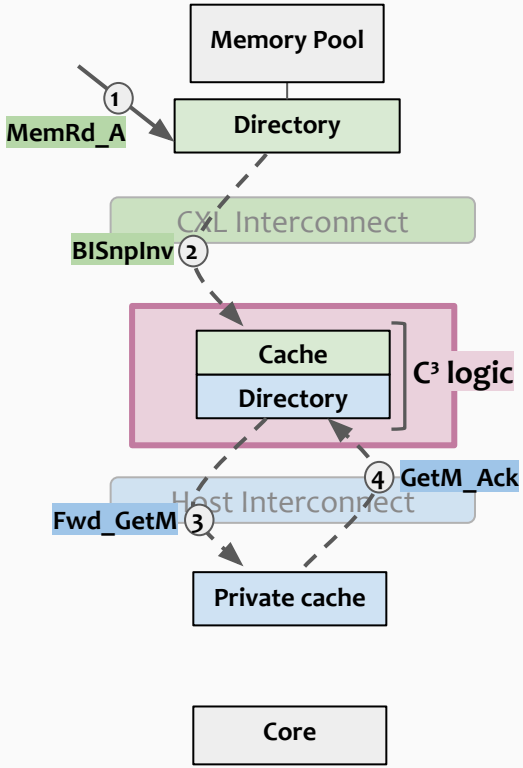
C³ ***combines*** the FSMs from host **directory** & CXL **cache**

C³ ***appears*** as a **directory** to hosts, and **caching client** to CXL

C³ **composes transactions** between **host** protocol and **CXL.mem**

- C³ **propagates** requests that produce **globally visible memory updates**
- C³ **propagates** CXL requests that **change local cache permissions**

Example:
- Host `GetM` request (②- ⑤) **encapsulates** the CXL transaction (③- ④)



Design: **The C³ *Compound State Machine***

C³ ***combines*** the FSMs from host **directory** & CXL **cache**

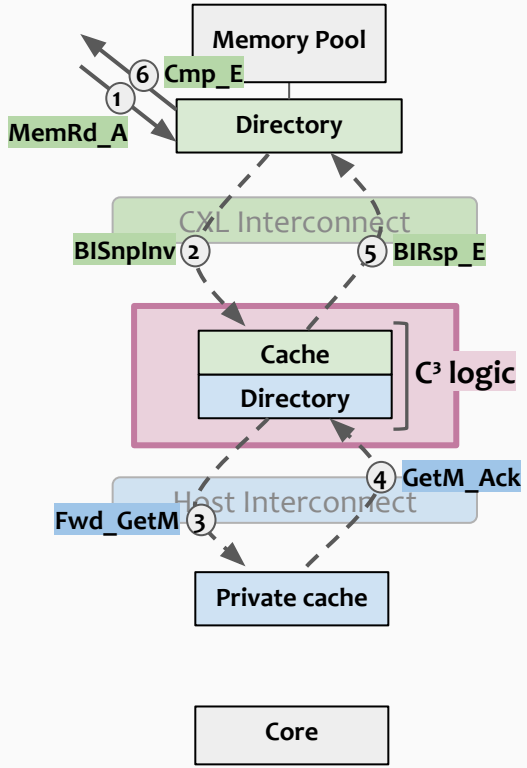
C³ ***appears*** as a **directory** to hosts, and **caching client** to CXL

C³ **composes transactions** between **host** protocol and **CXL.mem**

- C³ **propagates** requests that produce **globally visible memory updates**
- C³ **propagates** CXL requests that **change local cache permissions**

Example:

- Host `GetM` request (②- ⑤) **encapsulates** the CXL transaction (③- ④)



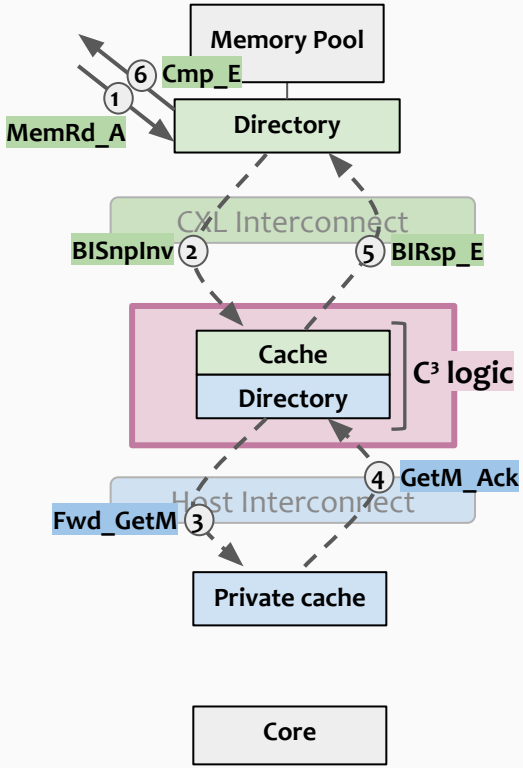
Design: The C³ *Compound State Machine*

C³ ***combines*** the FSMs from host **directory** & CXL **cache**

C³ ***appears*** as a **directory** to hosts, and **caching client** to CXL

C³ **composes transactions** between **host** protocol and **CXL.mem**

- C³ **propagates** requests that produce **globally visible memory updates**
- C³ **propagates** CXL requests that **change local cache permissions**



Example:

- Host `GetM` request (②-⑤) **encapsulates** the CXL transaction (③-④)
- CXL `snoop` request (②-⑤) **encapsulates** the host transaction (③-④)

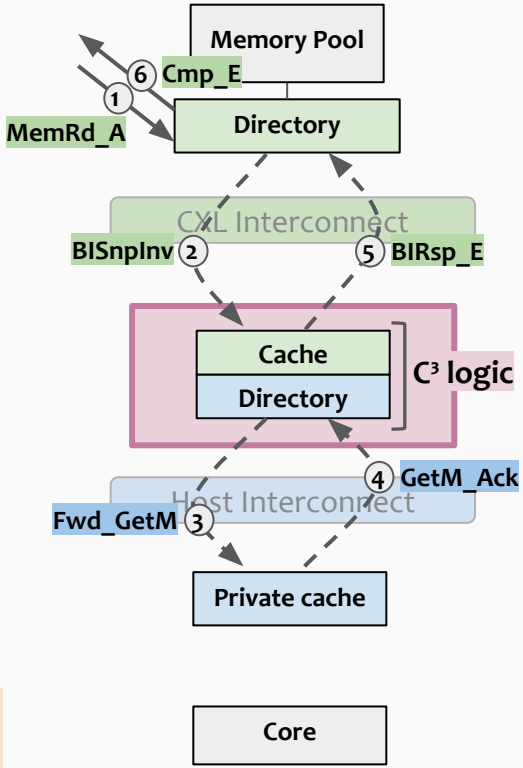
Design: The C³ *Compound State Machine*

C³ ***combines*** the FSMs from host **directory** & CXL **cache**

C³ ***appears*** as a **directory** to hosts, and **caching client** to CXL

C³ **composes transactions** between **host** protocol and **CXL.mem**

- C³ **propagates** requests that produce **globally visible memory updates**
- C³ **propagates** CXL requests that **change local cache permissions**



Example:
- Host `GetM` request (②-⑤) **encapsulates** the CXL transaction (③-④)
- CXL `snoop` request (②-⑤) **encapsulates** the host transaction (③-④)

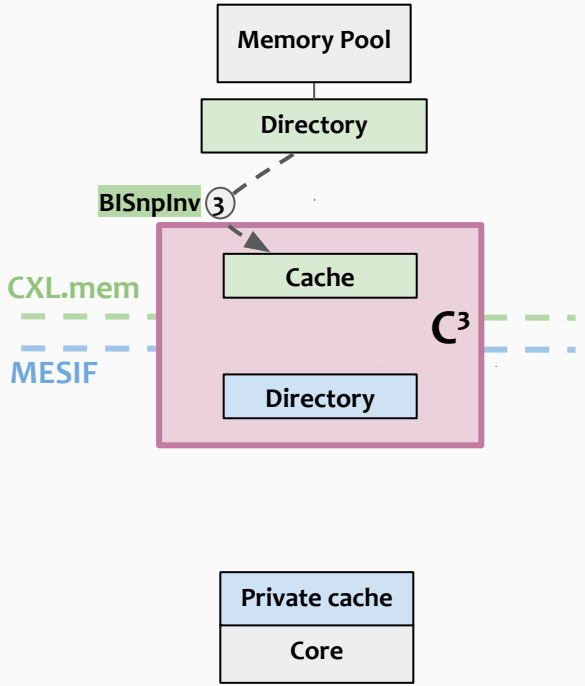
C³ propagation rules: **Atomicity** & **Delegation**

Design: **Semantic Request Translation**

How to build the C³ Compound State Machine?

Idea: Core accesses as **universal semantics** — `load`, `store`, `evict`, `fence`

Transaction translation principle:



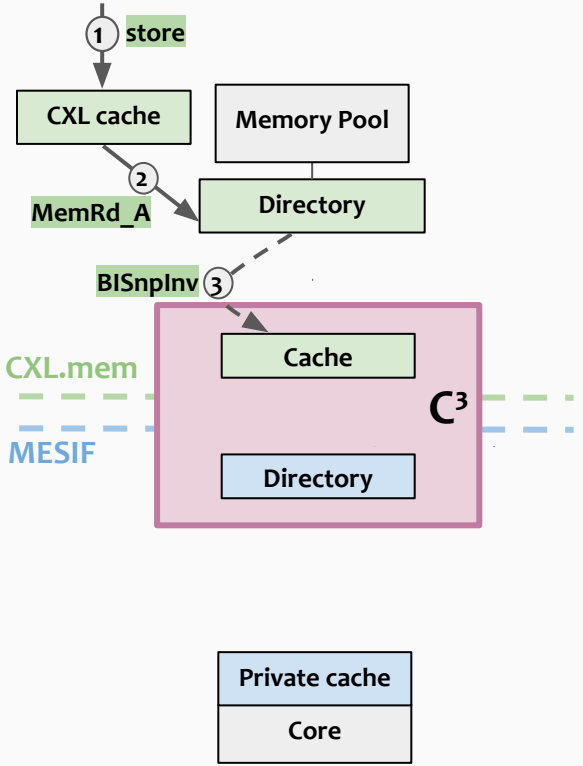
Design: **Semantic Request Translation**

How to build the C³ Compound State Machine?

Idea: Core accesses as **universal semantics** — `load`, `store`, `evict`, `fence`

Transaction translation principle:

- Deduce **access** performed by **original requestor**



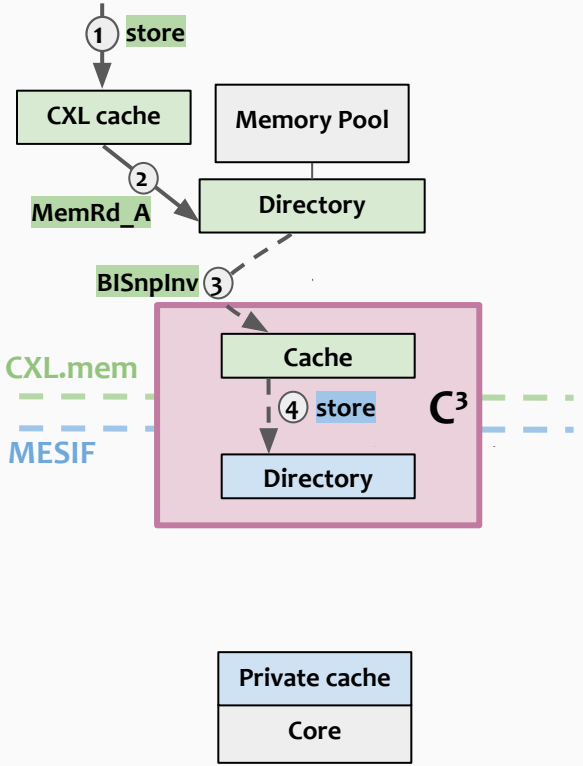
Design: **Semantic Request Translation**

How to build the C³ Compound State Machine?

Idea: Core accesses as **universal semantics** — `load`, `store`, `evict`, `fence`

Transaction translation principle:

- Deduce **access** performed by **original requestor**
- Identify **equivalent access** in remote domain



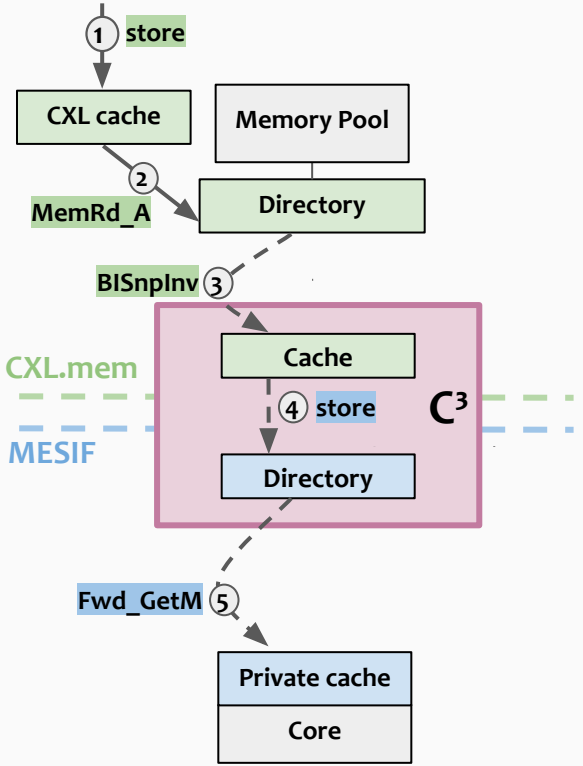
Design: **Semantic Request Translation**

How to build the C³ Compound State Machine?

Idea: Core accesses as **universal semantics** — `load`, `store`, `evict`, `fence`

Transaction translation principle:

- Deduce **access** performed by **original requestor**
- Identify **equivalent access** in remote domain
- **Simulate** equivalent access **to start local transaction**



Design: **Semantic Request Translation**

How to build the C³ Compound State Machine?

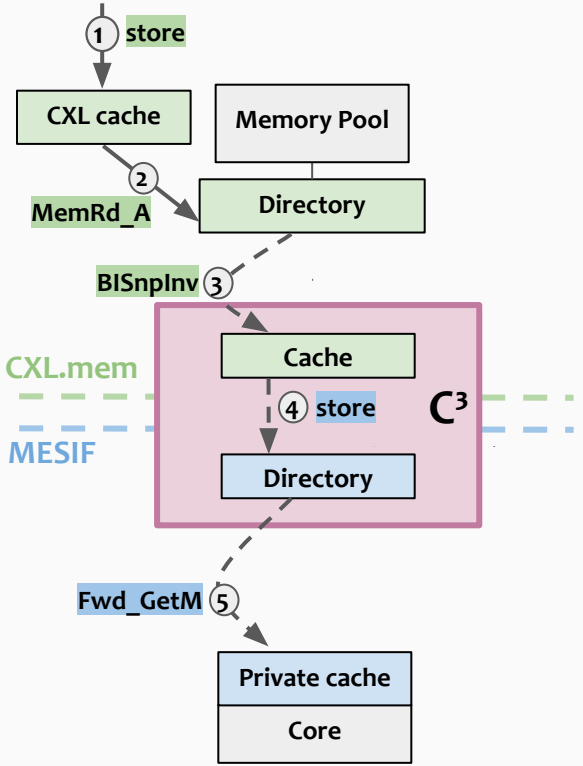
Idea: Core accesses as **universal semantics** — `load`, `store`, `evict`, `fence`

Transaction translation principle:

- Deduce **access** performed by **original requestor**
- Identify **equivalent access** in remote domain
- **Simulate** equivalent access **to start local transaction**

Translation table:

Origin Transaction			Propagated Remote Transaction		
Incoming Message	Current State				



Design: **Semantic Request Translation**

How to build the C³ Compound State Machine?

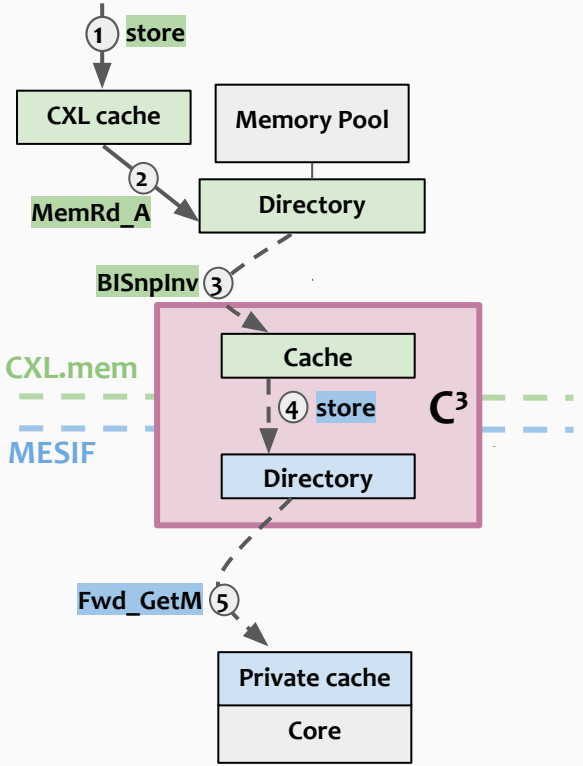
Idea: Core accesses as **universal semantics** — `load`, `store`, `evict`, `fence`

Transaction translation principle:

- Deduce **access** performed by **original requestor**
- Identify **equivalent access** in remote domain
- **Simulate** equivalent access **to start local transaction**

Translation table:

Origin Transaction			Propagated Remote Transaction		
Incoming Message	Current State				
`BISnpInv` ③	`M`, `M`				



Design: **Semantic Request Translation**

How to build the C³ Compound State Machine?

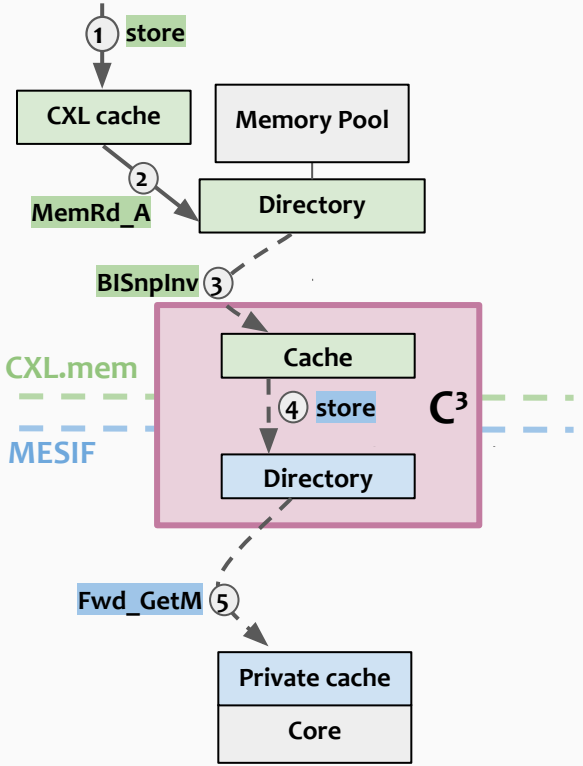
Idea: Core accesses as **universal semantics** — `load`, `store`, `evict`, `fence`

Transaction translation principle:

- Deduce **access** performed by **original requestor**
- Identify **equivalent access** in remote domain
- **Simulate** equivalent access **to start local transaction**

Translation table:

Origin Transaction			Propagated Remote Transaction		
Incoming Message	Current State	Cache access			
`BISnpInv` ③	`M`, `M`	`store` ①			



Design: **Semantic Request Translation**

How to build the C³ Compound State Machine?

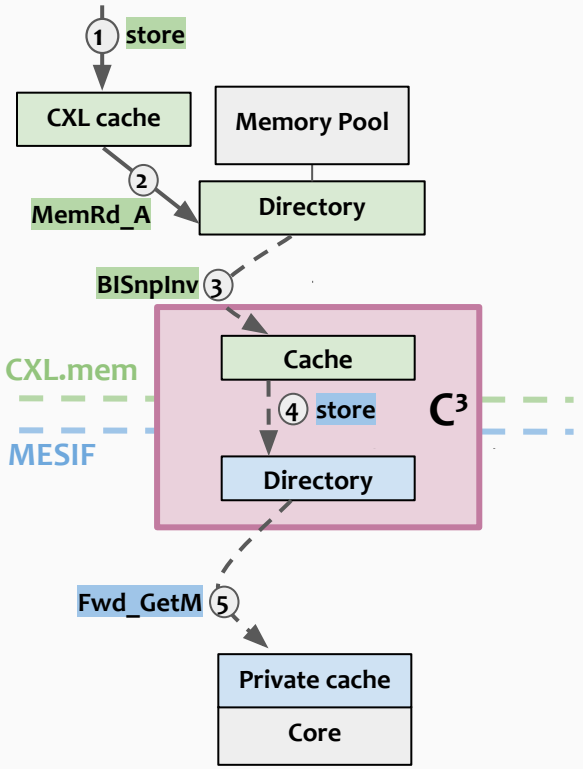
Idea: Core accesses as **universal semantics** — `load, store, evict, fence`

Transaction translation principle:

- Deduce **access** performed by **original requestor**
- Identify **equivalent access** in remote domain
- **Simulate** equivalent access **to start local transaction**

Translation table:

Origin Transaction			Propagated Remote Transaction		
Incoming Message	Current State	Cache access	Translated access		
`BISnpInv` ③	`M, M`	`store` ①	`store` ④		



Design: **Semantic Request Translation**

How to build the C³ Compound State Machine?

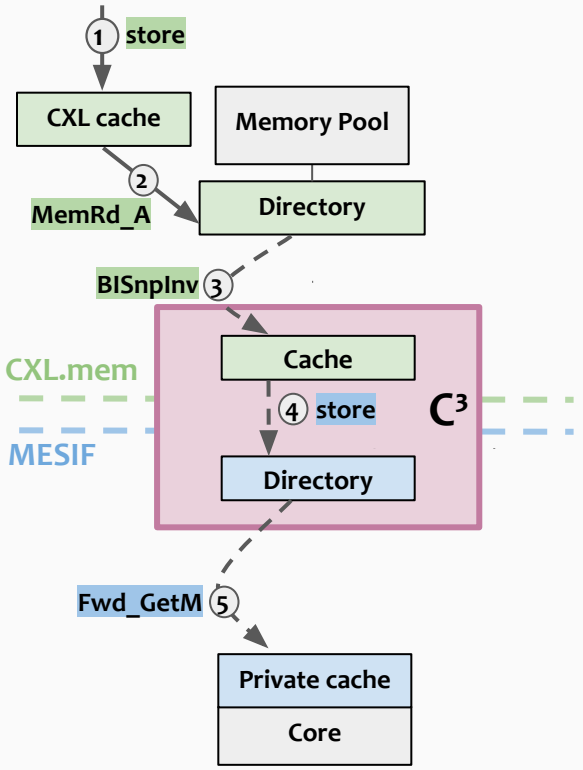
Idea: Core accesses as **universal semantics** — `load`, `store`, `evict`, `fence`

Transaction translation principle:

- Deduce **access** performed by **original requestor**
- Identify **equivalent access** in remote domain
- **Simulate** equivalent access **to start local transaction**

Translation table:

Origin Transaction			Propagated Remote Transaction		
Incoming Message	Current State	Cache access	Translated access	Action	Next state
`BISnpInv` ③	`M`, `M`	`store` ①	`store` ④	Send Fwd_GetM to local owner ⑤	



Design: **Semantic Request Translation**

How to build the C³ Compound State Machine?

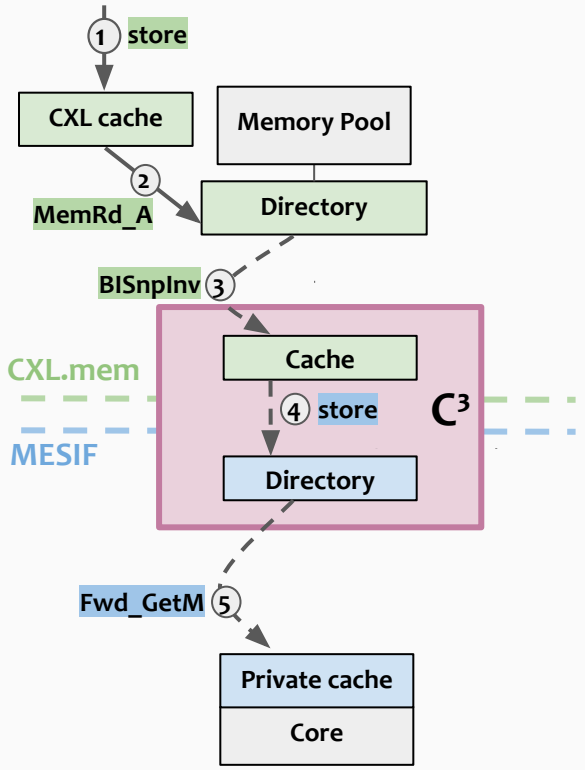
Idea: Core accesses as **universal semantics** — `load`, `store`, `evict`, `fence`

Transaction translation principle:

- Deduce **access** performed by **original requestor**
- Identify **equivalent access** in remote domain
- **Simulate** equivalent access **to start local transaction**

Translation table:

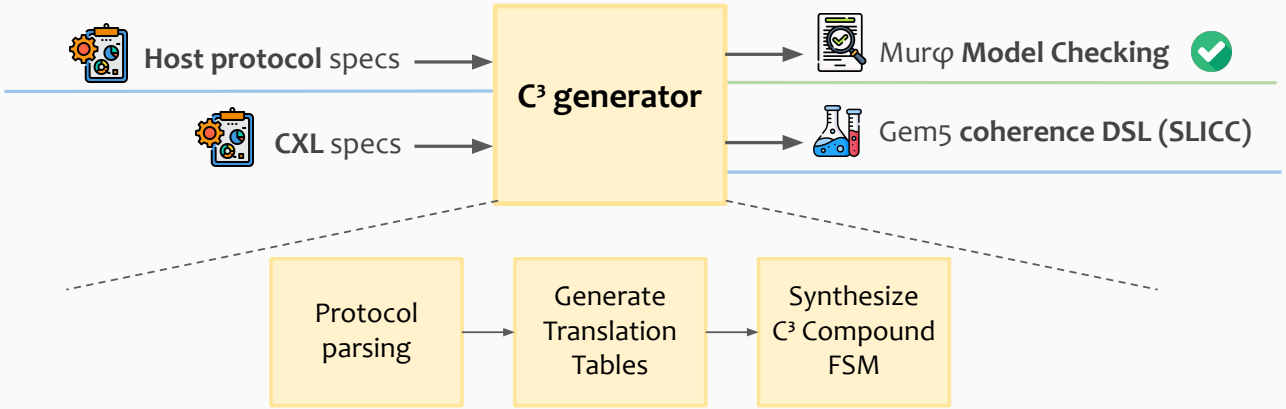
Origin Transaction			Propagated Remote Transaction		
Incoming Message	Current State	Cache access	Translated access	Action	Next state
`BISnpInv`	`M`, `M`	`store`	`store`	Send Fwd_GetM to local owner	MI^A, MI
③		①	④	⑤	



Implementation

Implement C³ controller logic with gem5 cache coherence models (SLICC)

Generate C³ controllers[2] from protocol specifications (PCC[1]):



[1] Oswald et al., *ProtoGen: Automatically Generating Directory Cache Coherence Protocols from Atomic Specifications, ISCA'18*
[2] Lefort et al., *vCXLGen: Automated Synthesis and Verification of CXL Bridges for Heterogeneous Architectures*, ASPLOS'26

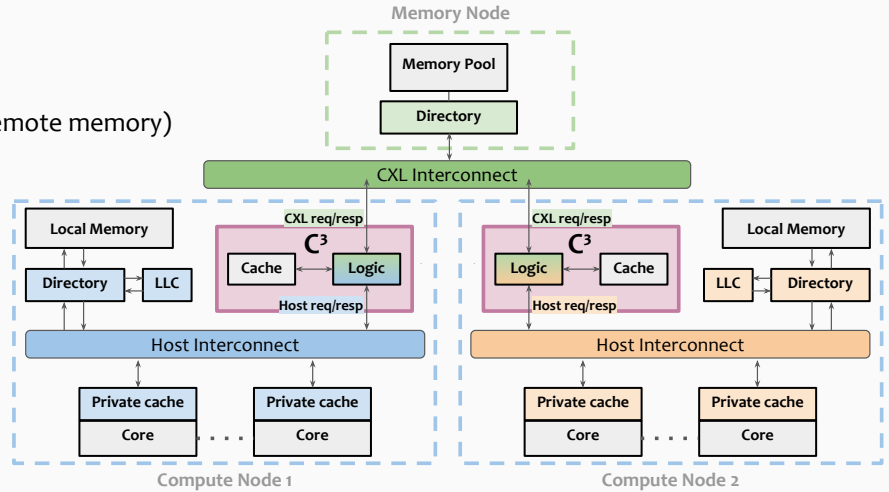
Evaluation: **Methodology**

We evaluate C³ through gem5 simulations:

- **SE** syscall-emulation**, Ruby** memory subsystem, **Garnet** interconnect network models
- **O3** out-of-order CPU models
- **SLICC** DSL to implement <u>all CC controllers</u> (incl. C³)

System model:

3 heterogeneous clusters: **2 hosts** + **1 CXL fabric** (w/ remote memory)



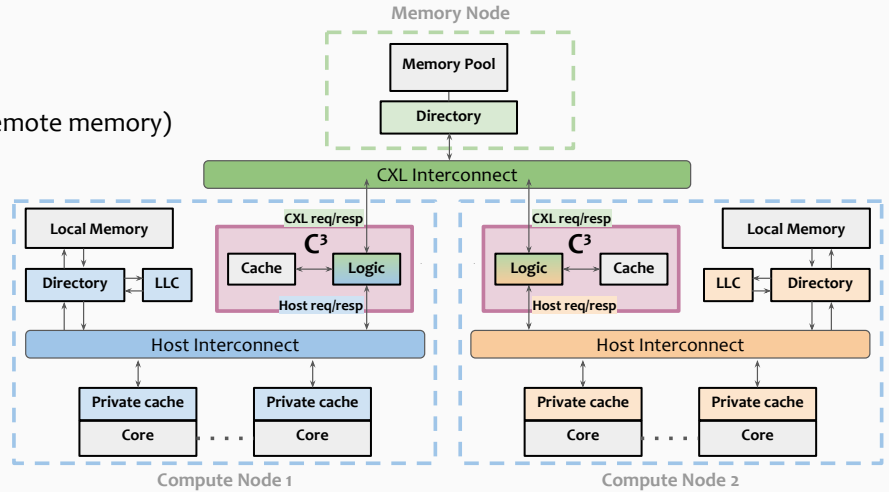
Evaluation: **Methodology**

We evaluate C³ through gem5 simulations:

- **SE** syscall-emulation, **Ruby** memory subsystem, **Garnet** interconnect network models
- **O3** out-of-order CPU models
- **SLICC** DSL to implement <u>all CC controllers</u> (incl. C³)

System model:

3 heterogeneous clusters: **2 hosts** + **1 CXL fabric** (w/ remote memory)



Heterogeneous cluster combinations with C³:

3 host protocols:	**MESI**, **MOESI**, **MESIF**
2 interconnect protocols:	**MESI**, **CXL.mem**
2 host MCMs:	**Arm**, **TSO**

Example combination: **MESI_CXL_MESIF** with Arm_Tso

Evaluation: **Goals**

We answer the following evaluation questions:

- **Correctness:** *the C³ logic (FSM) and SLICC controllers?*
  - *Can C³ correctly **reconcile heterogeneous MCMs**?*
  - *Can C³ correctly **interoperate heterogeneous CC protocols**?*
- **Genericity:** *Is C³ applicable to different heterogeneous host protocols and MCMs?*
- **Performance:** *What are the overheads of the C³ methodology?*

Evaluation: **Correctness**

Does C³ really enforce Compound Memory Consistency in gem5?

Workloads: 7 litmus tests generated with herd7[1] for ARM ISA

Systems: 6 heterogeneous combinations varying 3 **protocols** & 2 **MCMs**

Test	MESI-CXL-MESI			MESI-CXL-MOESI		
	Arm-Arm	TSO-Arm	TSO-TSO	Arm-Arm	TSO-Arm	TSO-TSO
2_2W-sys	✓	✓	✓	✓	✓	✓
IRIW-sys	✓	✓	✓	✓	✓	✓
LB-sys	✓	✓	✓	✓	✓	✓
MP-sys	✓	✓	✓	✓	✓	✓
R-sys	✓	✓	✓	✓	✓	✓
S-sys	✓	✓	✓	✓	✓	✓
SB-sys	✓	✓	✓	✓	✓	✓

Observation: No forbidden outcomes (disallowed by hosts' MCM) after 100,000 executions of each test

Takeaway: C³ successfully preserve **host native MCMs** with CXL memory

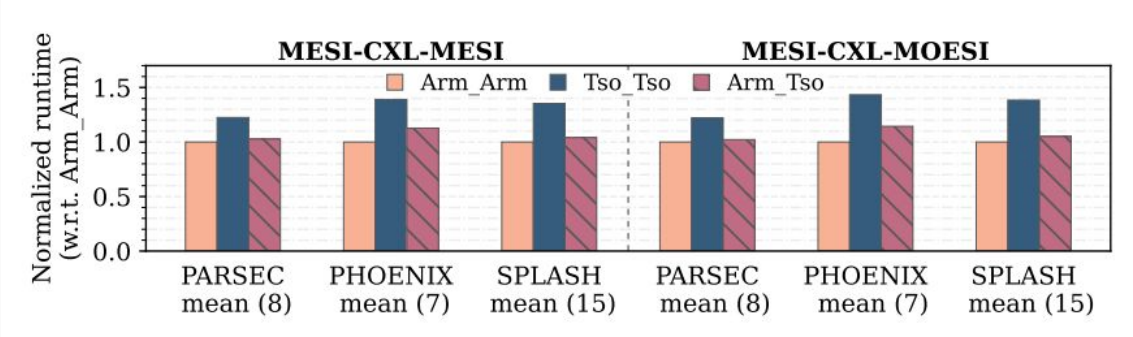
[1] herd7 consistency model simulator: https://diy.inria.fr/www/

Evaluation: **Genericity**

Can C³ reconcile heterogeneous MCMs and protocols?

Workloads: 33 parallel applications (PARSEC, SPLASH-4 & Phoenix suites)

Systems: 6 heterogeneous combinations varying 3 **protocols** & 2 **MCMs**



Observation:

- The weaker the MCM, the faster workloads run (on avg., exec. time: `Arm` < `Arm_TSO` < `TSO`)
- *Weak* MCM is not penalized by *strong* MCM in `Arm_TSO`

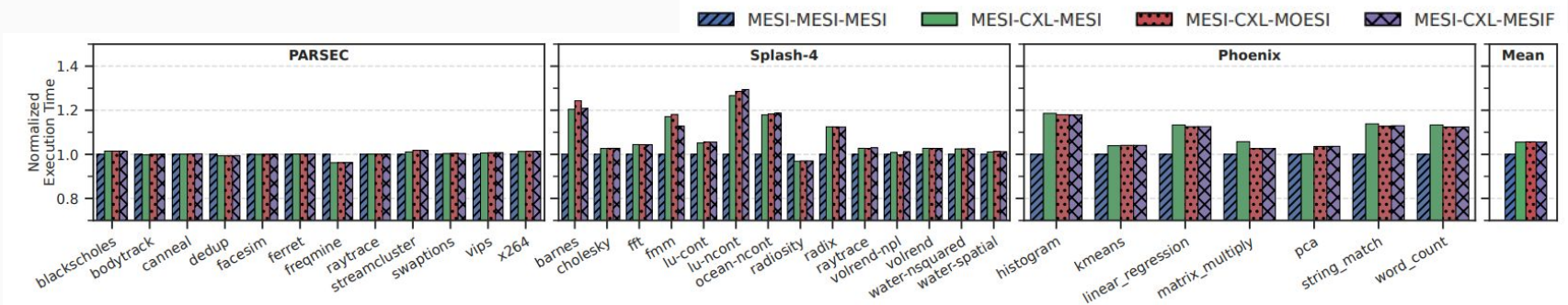
Takeaway: C³ composition of MCMs is **not overly strong** (preserves weak MCM performance)

Evaluation: **Performance**

What are the overheads of C³ controllers?

Workloads: 33 parallel applications (PARSEC, SPLASH-4 & Phoenix suites)

Baseline: unified homogeneous MESI (`MESI-MESI-MESI`) -> conventional MESI LLC instead of C³



Observations:

1- C³ overheads are negligible in most workloads

2- All CXL variants are significantly slower for some workloads (e.g., `barnes`, `lu-ncont`, `histogram`)

Additional analysis: CXL `.mem` is slower than textbook MESI (more handshaking & memory traffic)

Takeaway: C³ logic **overheads are negligible,** `CXL.mem` performs worse than textbook MESI

Conclusion

Motivation:
CXL does not support heterogeneous architectures

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

Solution: C³
Pluggable **coherence bridges** that **translate** per-host **coherence** transactions **to CXL** & preserve original semantics

C³ key ideas:

- **2 propagation rules:** *Delegation* & *Atomicity*: forward coherence transaction effects to other domains
- **Request semantic translation:** leverage correct equivalent transaction in other coherence domains
- **FSM compounding:** Couple FSMs of host directory & CXL cache to implement C³ logic