

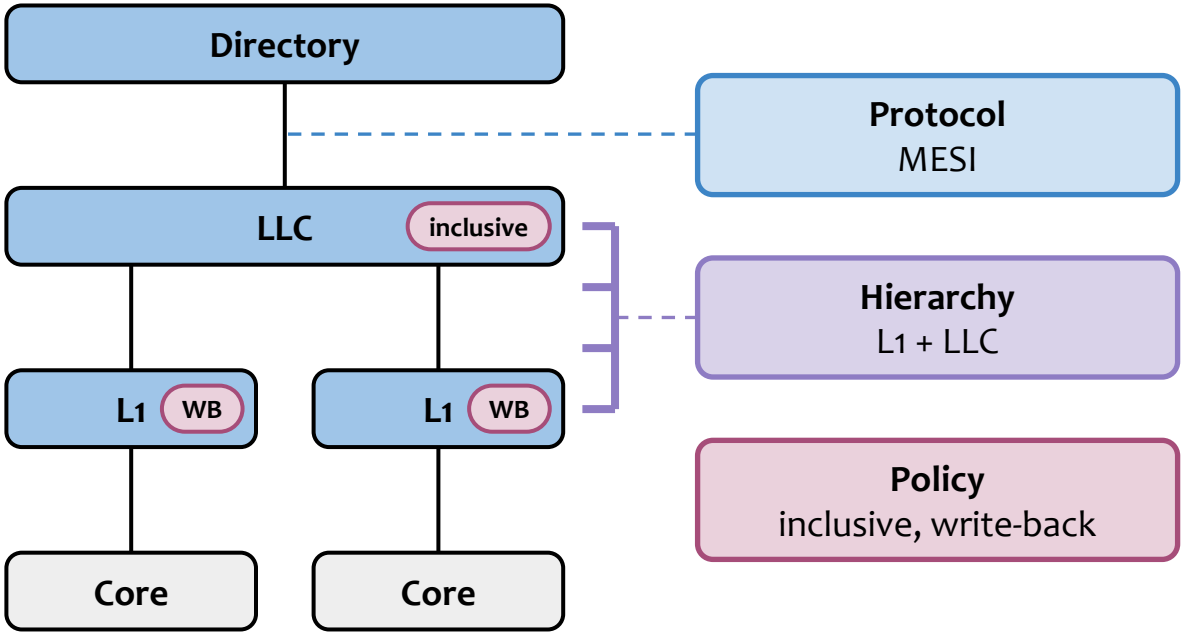
Synapse

synapse-coherence.github.io

Bob's job: Designing cache coherence controllers

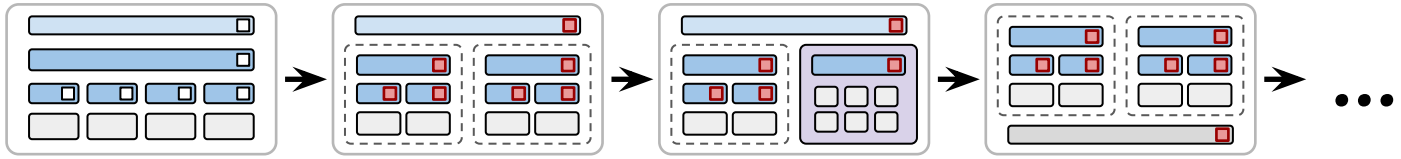


Computer architect

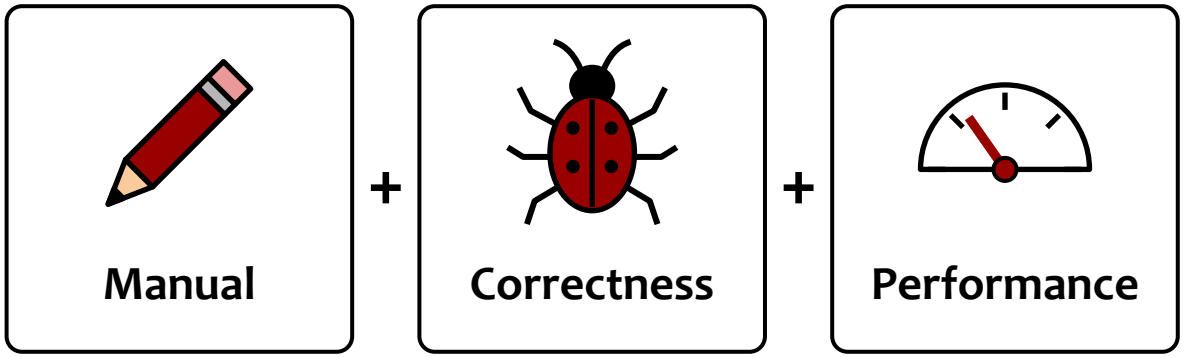


Bob's problem: Rebuilding controllers every time





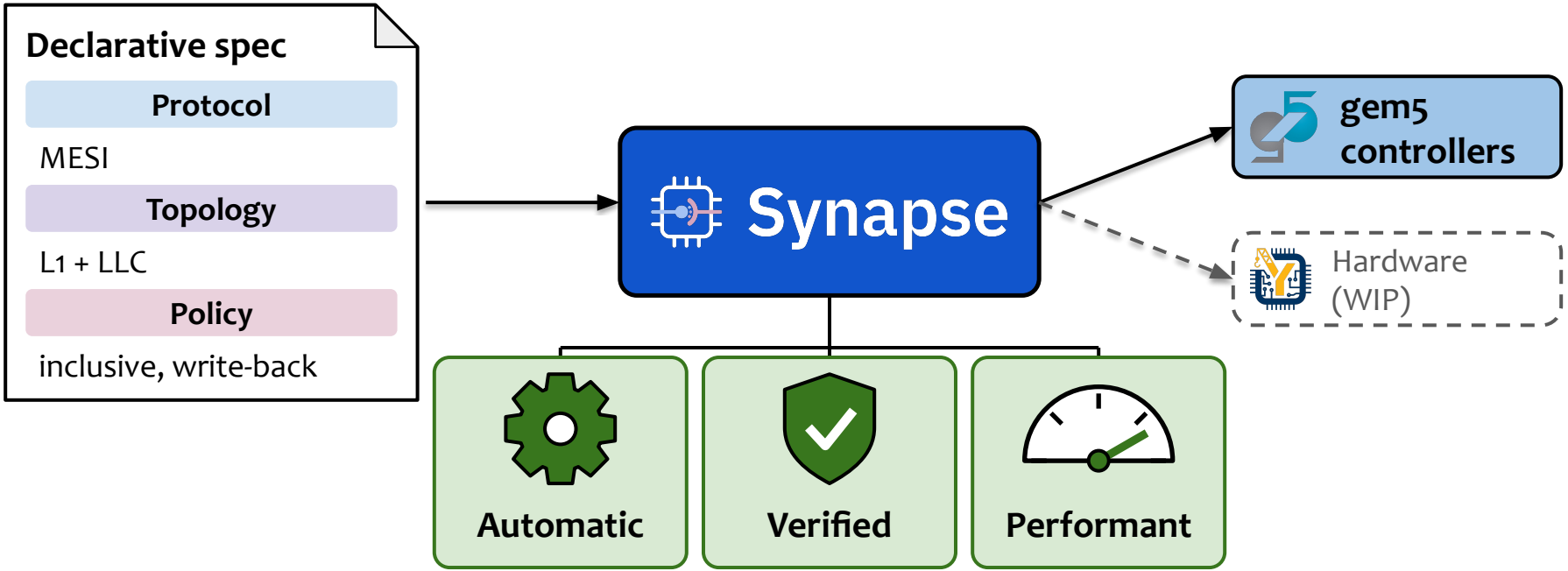
From scratch, for all new architectures…



Problem statement

Wouldn’t it be nice to automatically synthesize these coherence controllers?

Our promise: Automatic controller synthesis



Synapse

Decoupling Protocol, Topology, and Policy for Automated Synthesis of Coherence Controllers

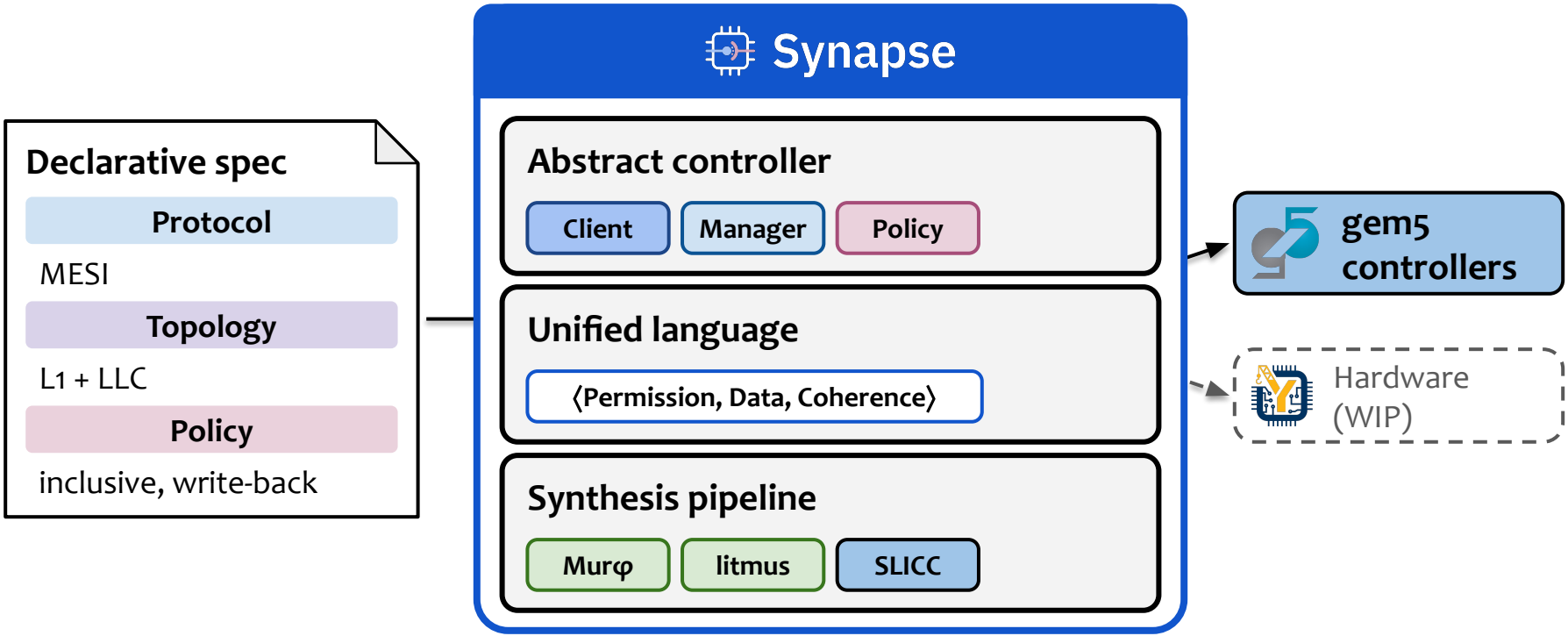
Nicolò Carpentieri, Anatole Lefort,
David Schall, Julian Pritzi, Pramod Bhatotia

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

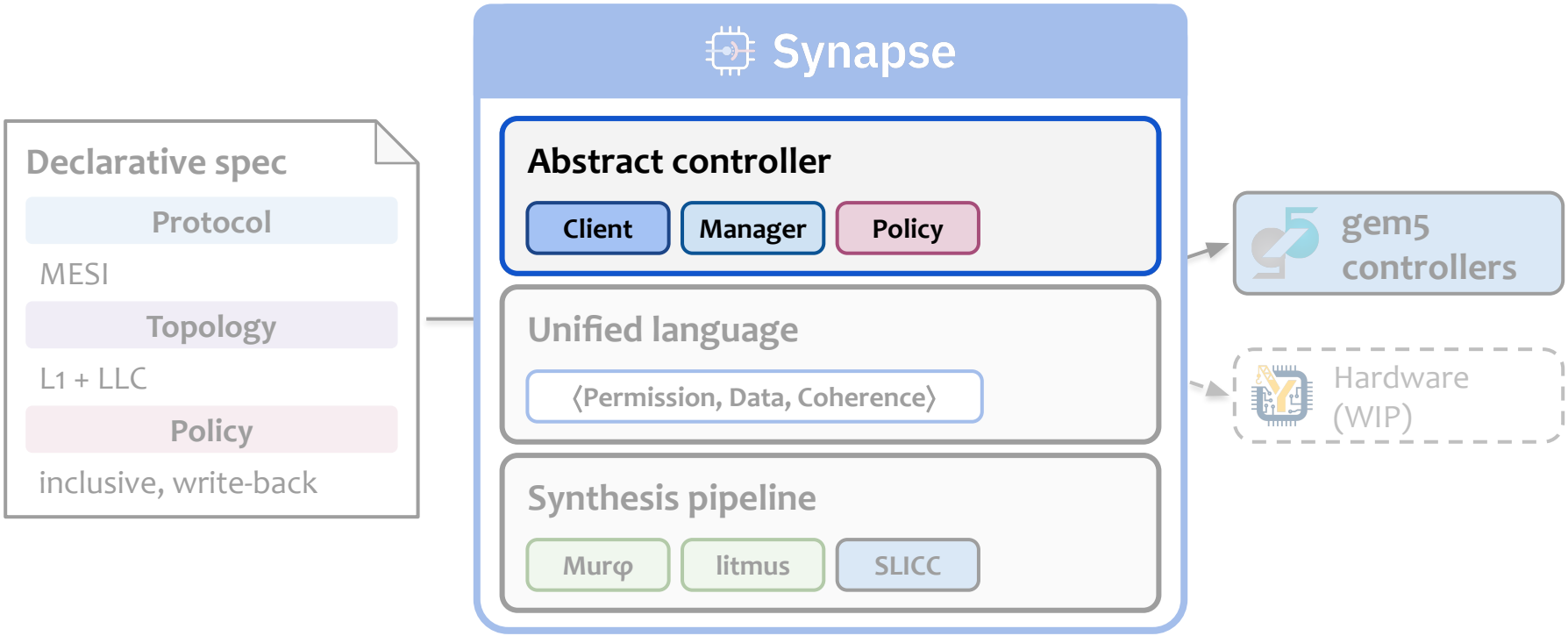


ACM/IEEE MICRO'26, Athens, Greece

Synapse overview



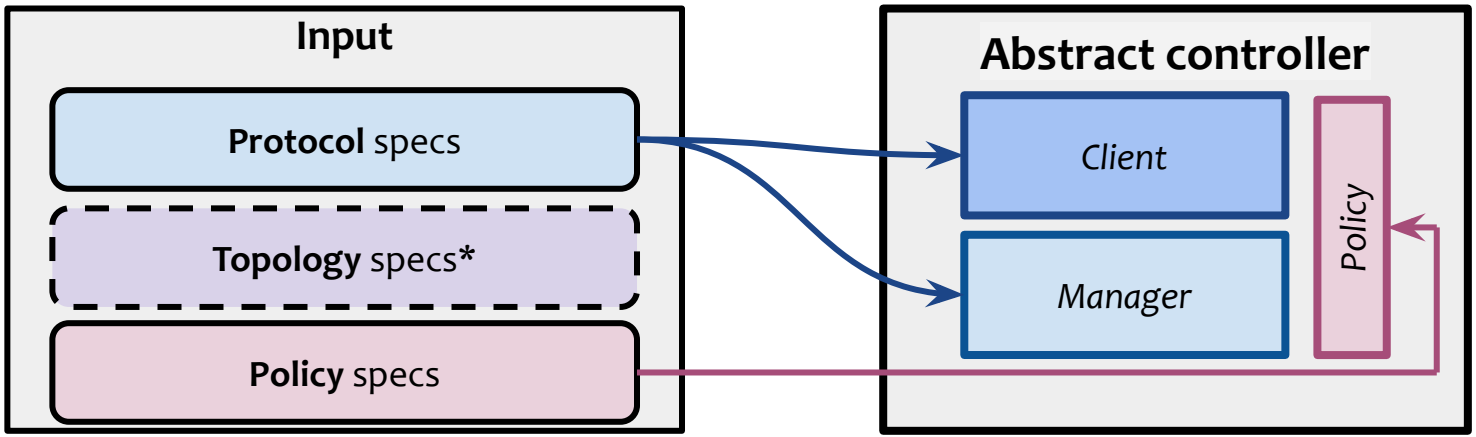
Synapse abstract controller



Synapse abstract controller

Key idea:

Every controller reduces to three FSMs: *client*, *manager* and *policy*



*Topology sets the controller's position

Synapse abstract controller

Key idea:

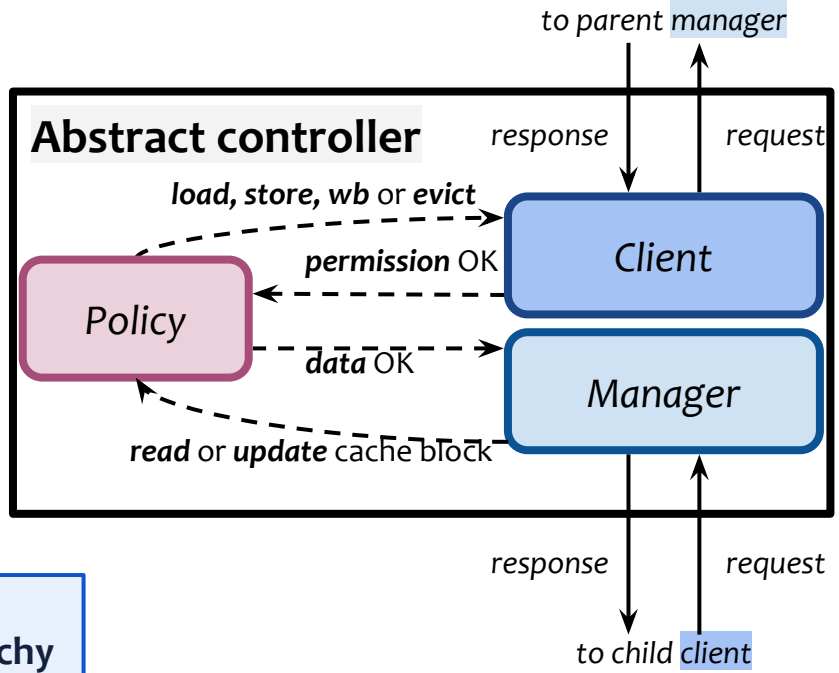
Every controller reduces to three FSMs: *client*, *manager* and *policy*

Principle:

Client: requests data/permissions from the parent/policy (like a *cache*)

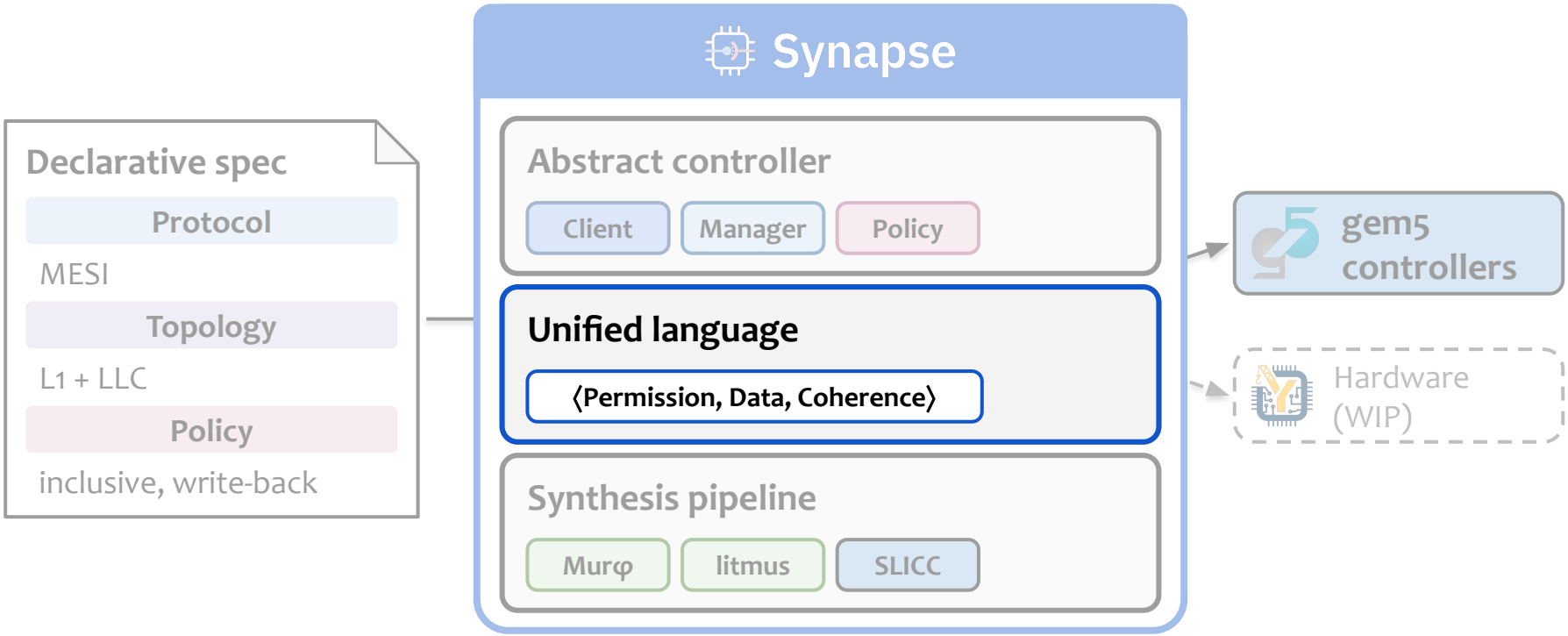
Manager: tracks the state of its children (like a *directory*)

Policy: decides allocation, update, replacement



One building block, placed anywhere in the hierarchy

Synapse unified language

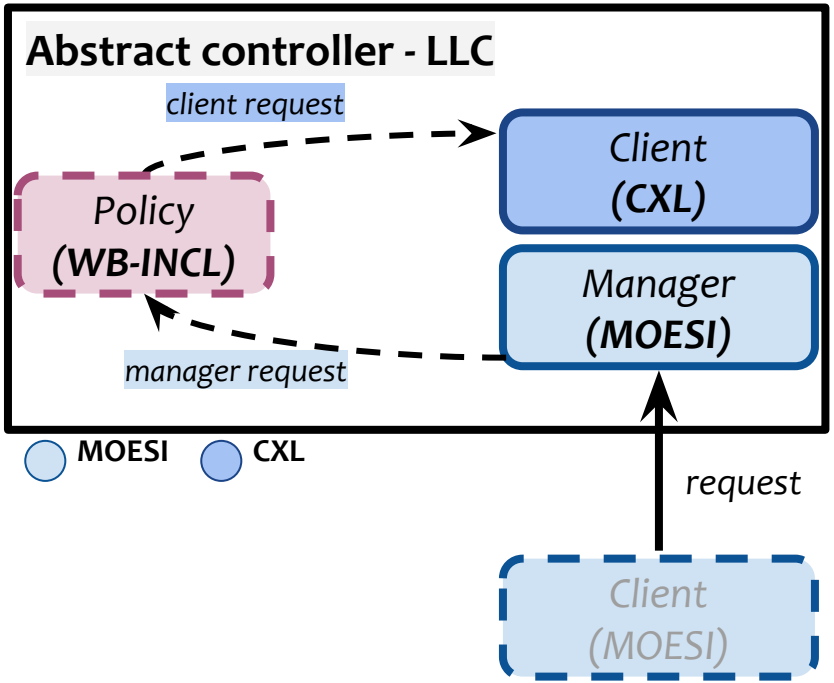


Synapse unified language

How do the client, manager, and policy communicate?

Problem:

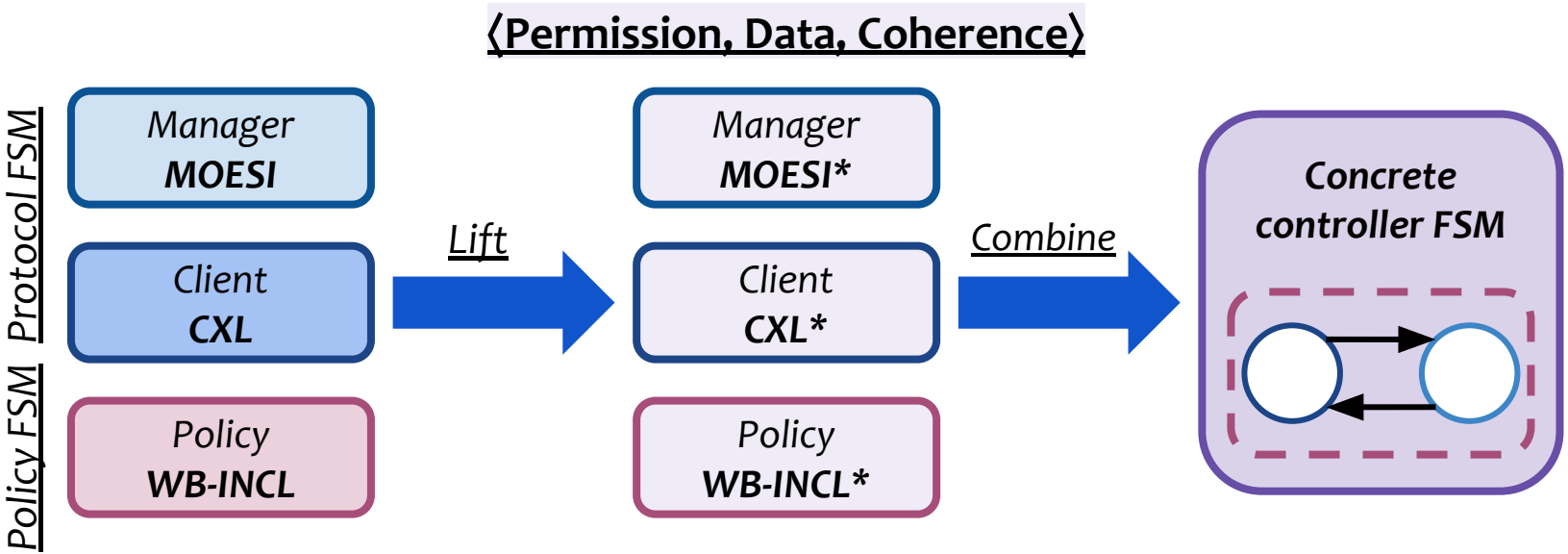
1. ***Policy*** ⟷ ***Manager/Client***
   - *How can a policy request trigger the corresponding manager request?*
2. ***Manager*** ⟷ ***Client***
   - *How can a manager request trigger the corresponding client request across different protocols?*



Synapse unified language

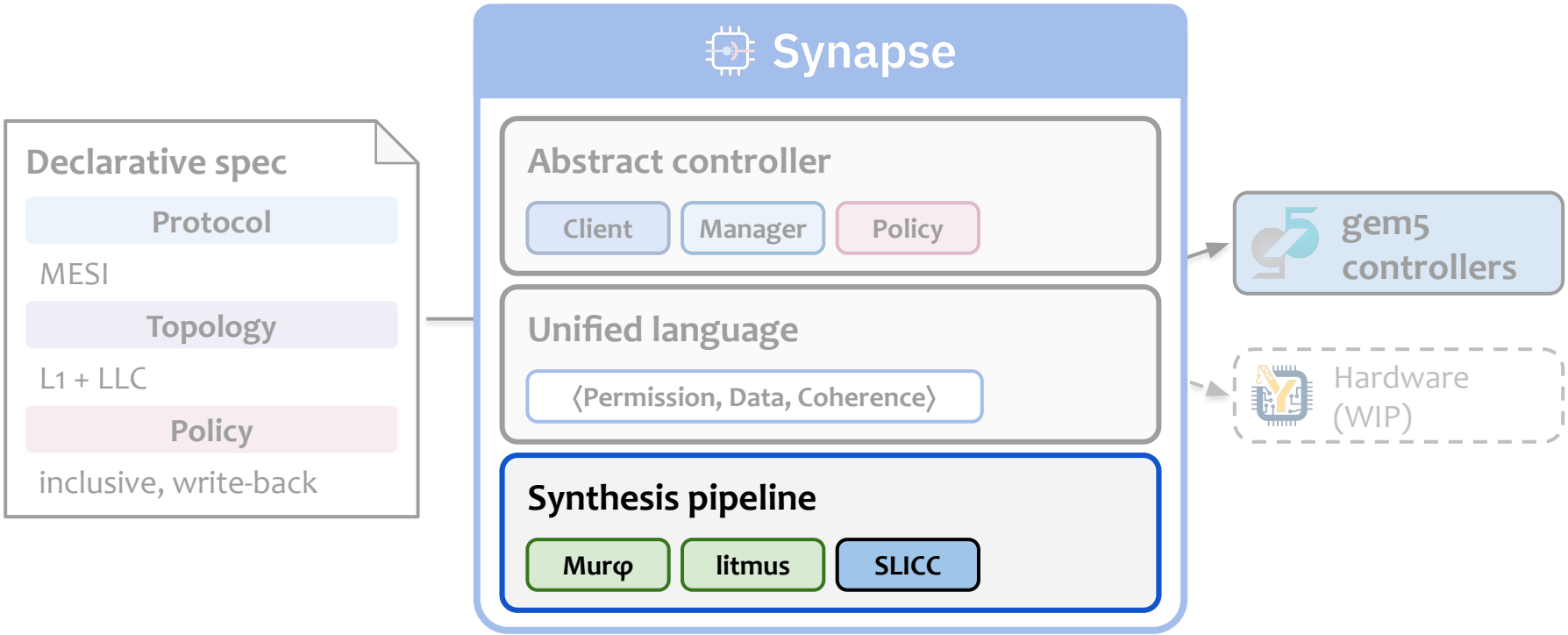
Key idea:

Translate every state and request into a *unified semantic language:*



This language allows building any cache controller

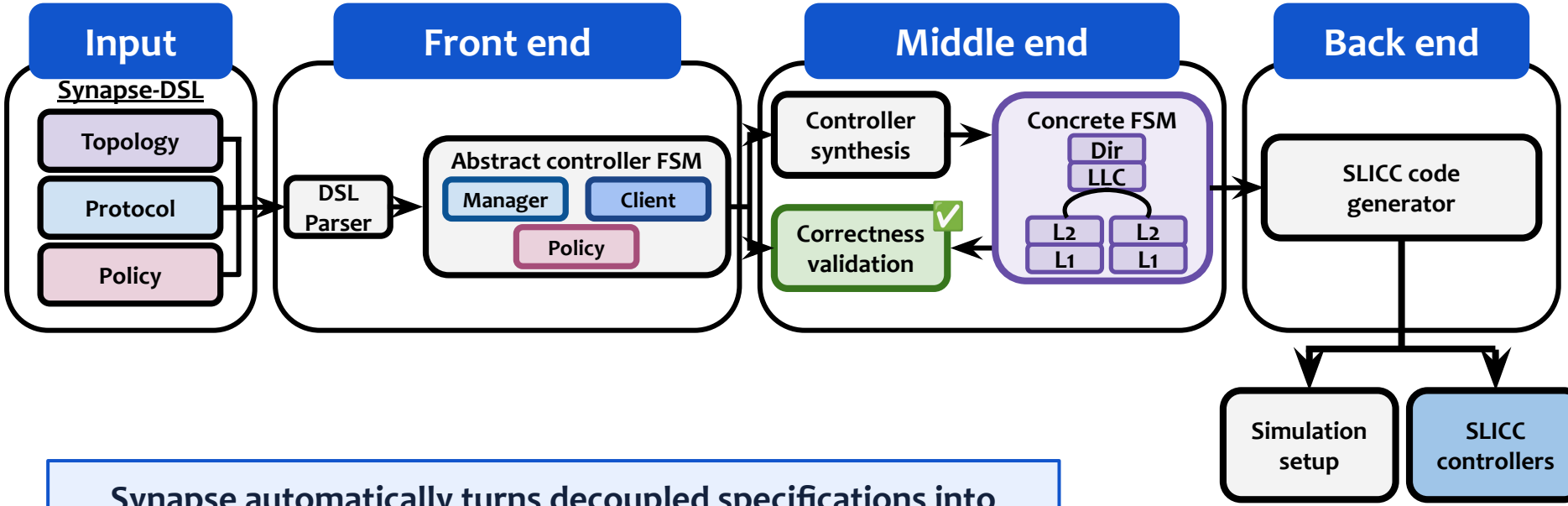
Synapse pipeline



Synapse pipeline

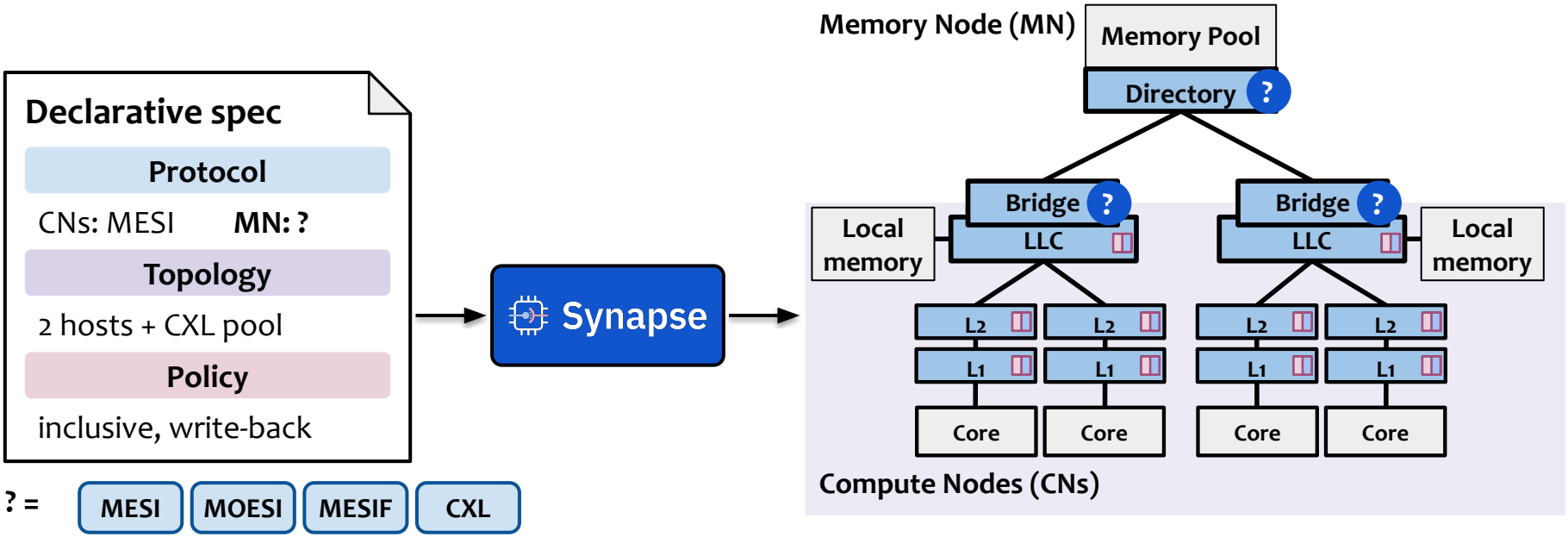
Key idea:

An end-to-end compiler generates verified, gem5-ready controllers



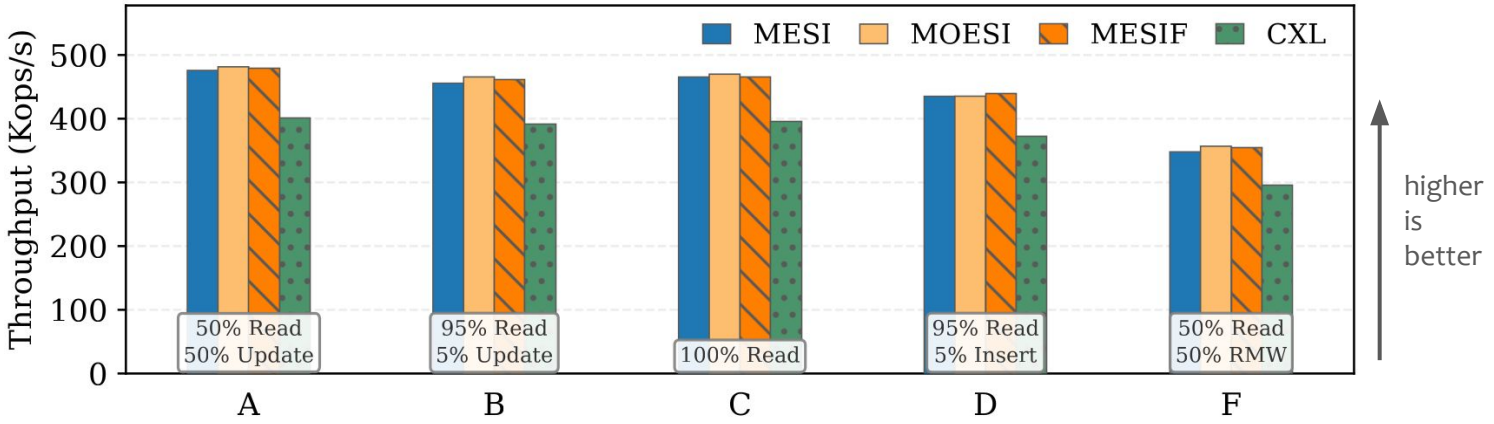
Synapse automatically turns decoupled specifications into verified, gem5-ready coherence controllers

Use case: Multi-host CXL memory pool



Workload: YCSB, 16 threads

Use case: Multi-host CXL memory pool



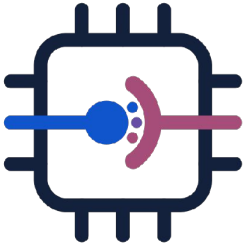
Observations:

- MOESI and MESIF match MESI; CXL.mem is slower on every workload

  Additional analysis: CXL `.mem` is slower as it involves more handshaking & memory traffic

Takeaway: Protocols behave differently; with Synapse, each one is a one-line change

Demo time!



Synapse

synapse-coherence.github.io

Conclusion



Synapse

synapse-coherence.github.io

Automatic synthesis of coherence controllers

↓

For Bob, and for the whole computer architecture community

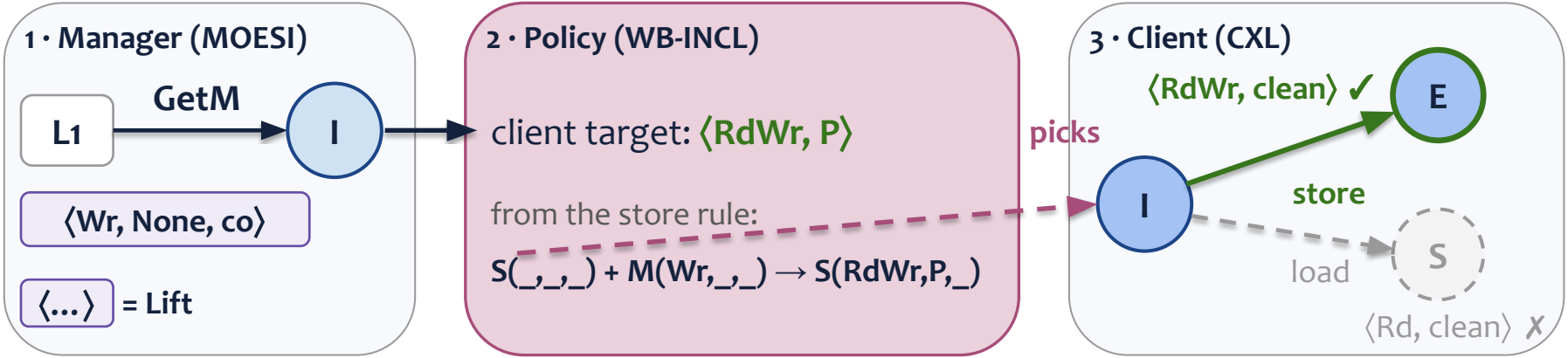
Try Synapse on your architecture, and help us extend it

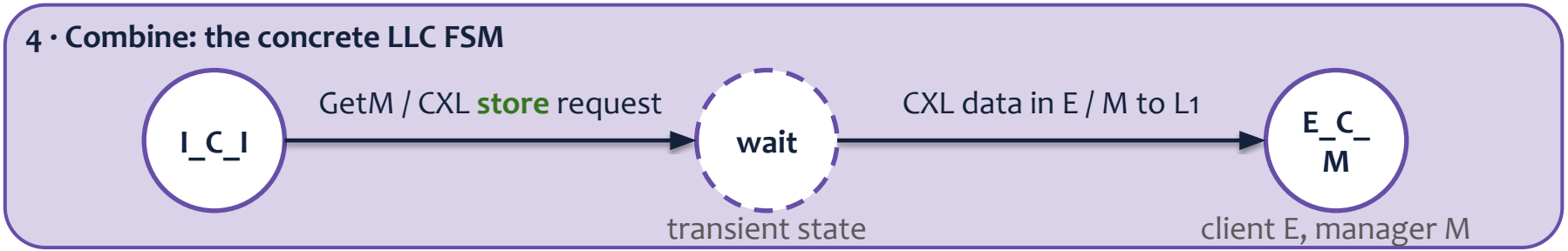


nicolo.carpentieri@tum.de

Backup slides

Lift and Combine: a store at the LLC





Takeaway: The policy sets the target, the labels pick the transaction, Combine fuses them

What Bob writes: three specs

Protocol (MESIF) ~400–1,100 lines

```
Architecture cache {
  Stable{I, S, F, E, M}
  Process(I, load, State){
    msg = Request(GetS, …);
    req.send(msg);
    await {
      when GetF_Ack:
        load; State=F; break;
      when GetM_Ack:
        load; State=E; break;
}}}
```

Policy (LLC, WB-incl.) ~40 lines

```
Rule( S(_,NP,_),     // missing
      M(Rd,None,_), // load
      S(Rd,P,_) )    // readable
```

Topology (excerpt) ~80 lines

```
- memory_node:
    directory: {protocol: CXL}
- compute_node:
    count: 2
    LLC: {protocol: MESI,
          policy: WB_INCL}
```

Correctness validation

Liveness: Murφ

- Deadlock freedom: every reachable state can move on
- Against state explosion: one cluster fully modeled at a time, at most 3 cores

Safety: litmus tests

- Murφ models driven by herd7 litmus tests (SC and RC)
- gem5 on ARM: 2+2W, MP, R, S, SB, 100,000 runs each: no forbidden outcome
- Control: without fences, forbidden outcomes do appear

Takeaway: Every synthesized hierarchy is checked for deadlocks and consistency violations

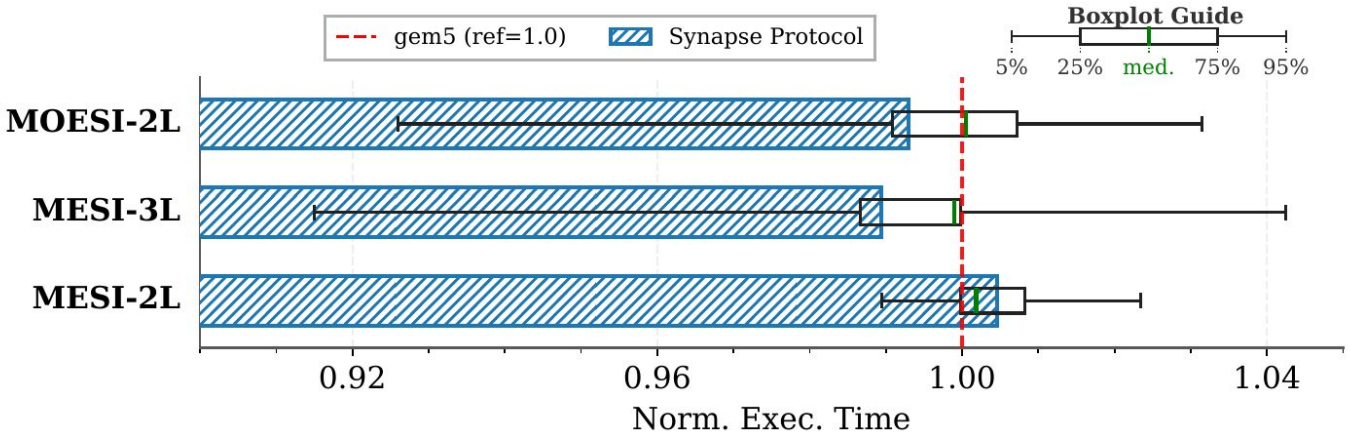
Specs vs. generated controllers

On-core	Off-core	Protocol spec	L1	L2	LLC	Dir	SLICC
MESI	MESI	410	13/41	–	56/327	6/45	7K
MOESI	MOESI	513	17/47	–	128/699	9/52	15K
MESI	MOESI	923	12/36	45/188	83/468	9/52	19K
MESI	MESIF	1,067	12/36	45/188	92/542	9/49	14K
MESI	CXL.mem	1,060	12/36	45/188	64/358	12/74	14K

States/transitions per controller; spec and SLICC in lines of code. Policy ~40 and topology ~80 lines per scenario; – = no L2.

Takeaway: 0.5K–1.2K lines of specs become 7K–19K lines of SLICC

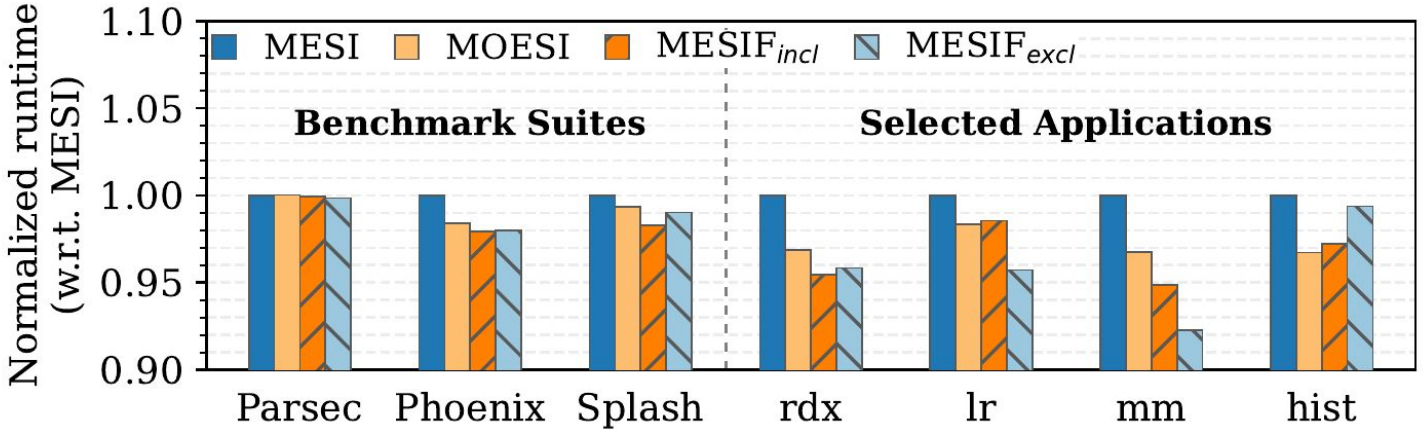
Performance vs. handwritten gem5



- 30 workloads: PARSEC, Phoenix, Splash-4
- Mean difference: MESI-2L +0.5%, MESI-3L −1.1%, MOESI-2L −0.7%
- Outliers: gem5 sends explicit unblock messages; Synapse drops them in MESI

Takeaway: Synthesized controllers match handwritten gem5 within 1.0% on average

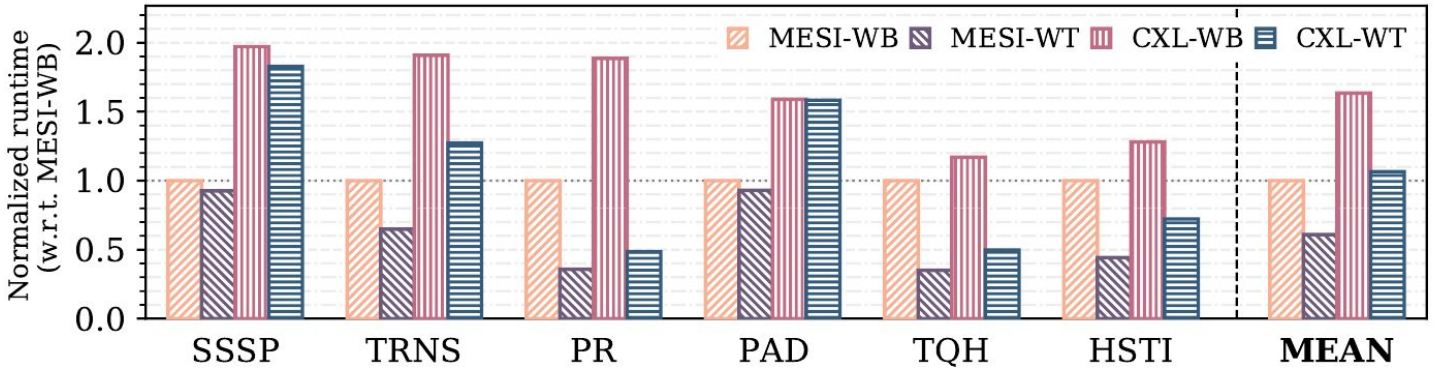
Use case: off-core protocols in NUMA



- 4 sockets, MESI, MOESI, MESIF (incl./excl. LLC)
- O and F states: –0.9% runtime, –13% to –17% memory read bandwidth
- MESIF-incl. fastest (–1.2%): the forwarder stays in the LLC, no extra hops

Takeaway: In NUMA, O and F states still pay off; MESIF most of all

Use case: caching policies in chiplets



- 8 chiplets × 8 accelerator PUs, MESI-RC on-core; MESI or CXL off-core
- Write-through vs. write-back: 1.9× faster with MESI, 1.8× with CXL
- Non-temporal stores skip L1 ownership: memory reads drop 1.5–14×

Takeaway: For streaming accelerators, the policy matters more than the protocol