Revolutionizing Mobile and Cloud via Coherence

Vijay Nagarajan

THE UNIVERSITY *of* EDINBURGH
informatics

Nicolai Oswald

Adarsh Patil

Mahesh Dananjaya

Carr Reece

Theo Olausson

Antonis Katsarakais

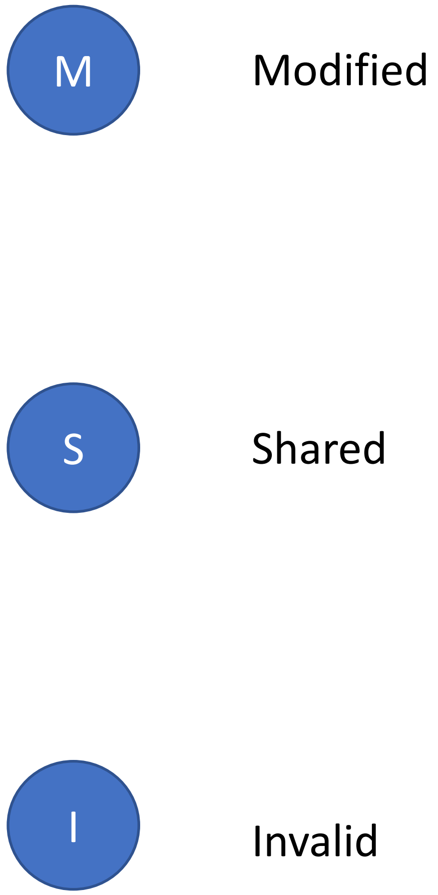
Vasilis Gavrielatos

Boris Grot

Dan Sorin

Nikos Nikoleris

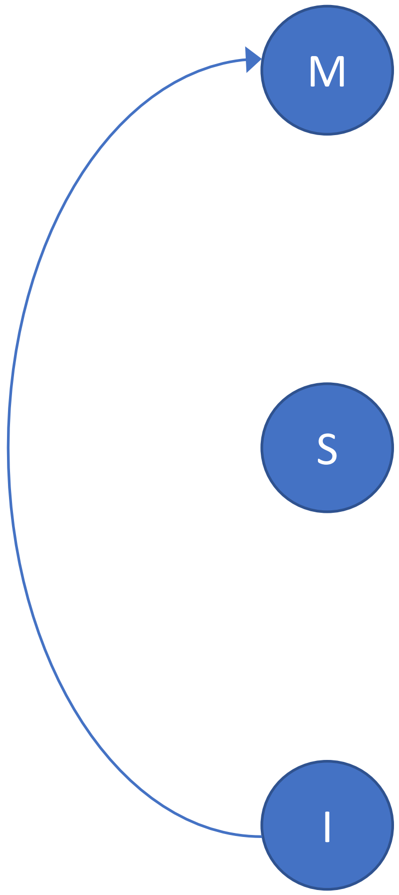
Tobias Grosser

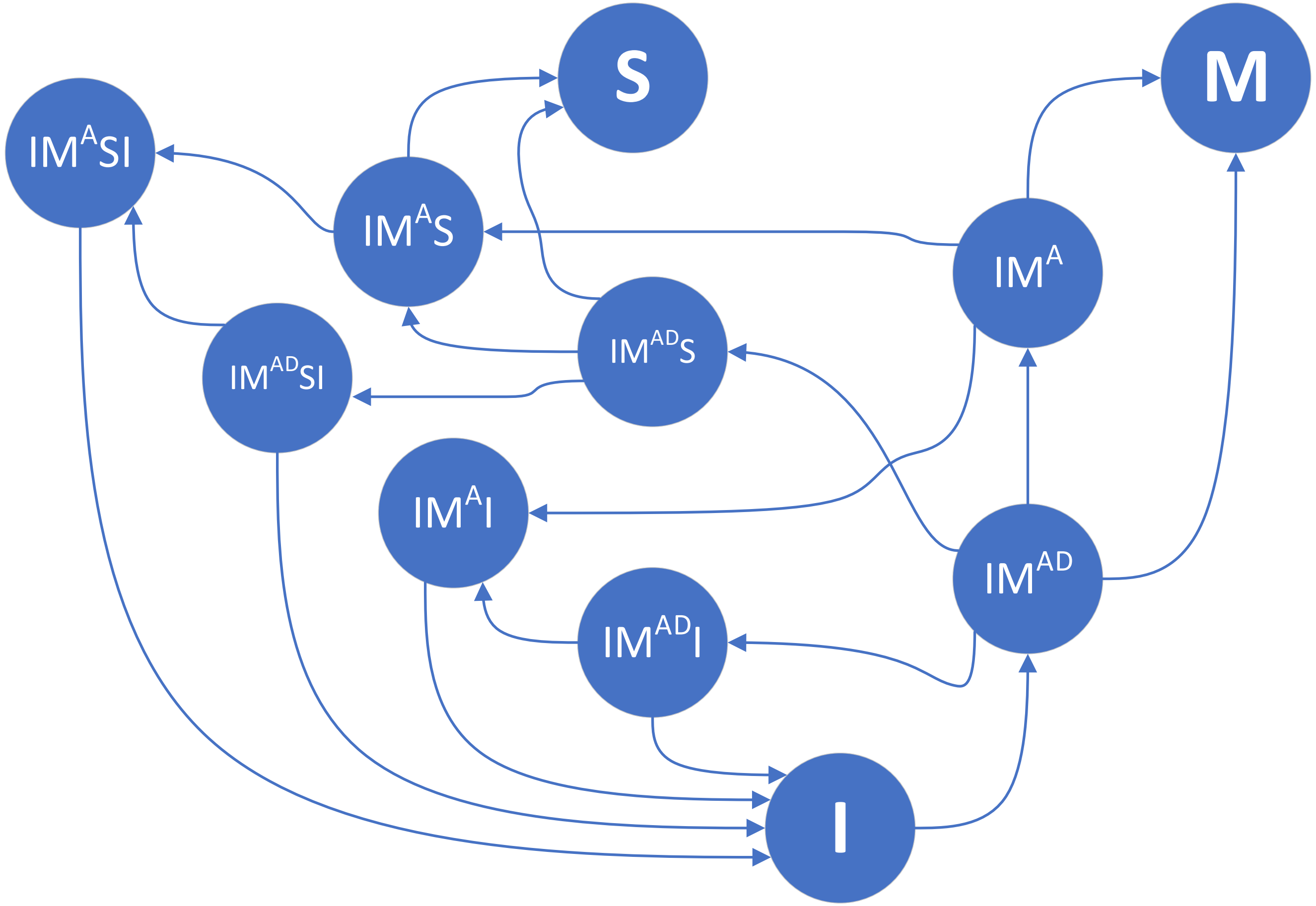


M Modified

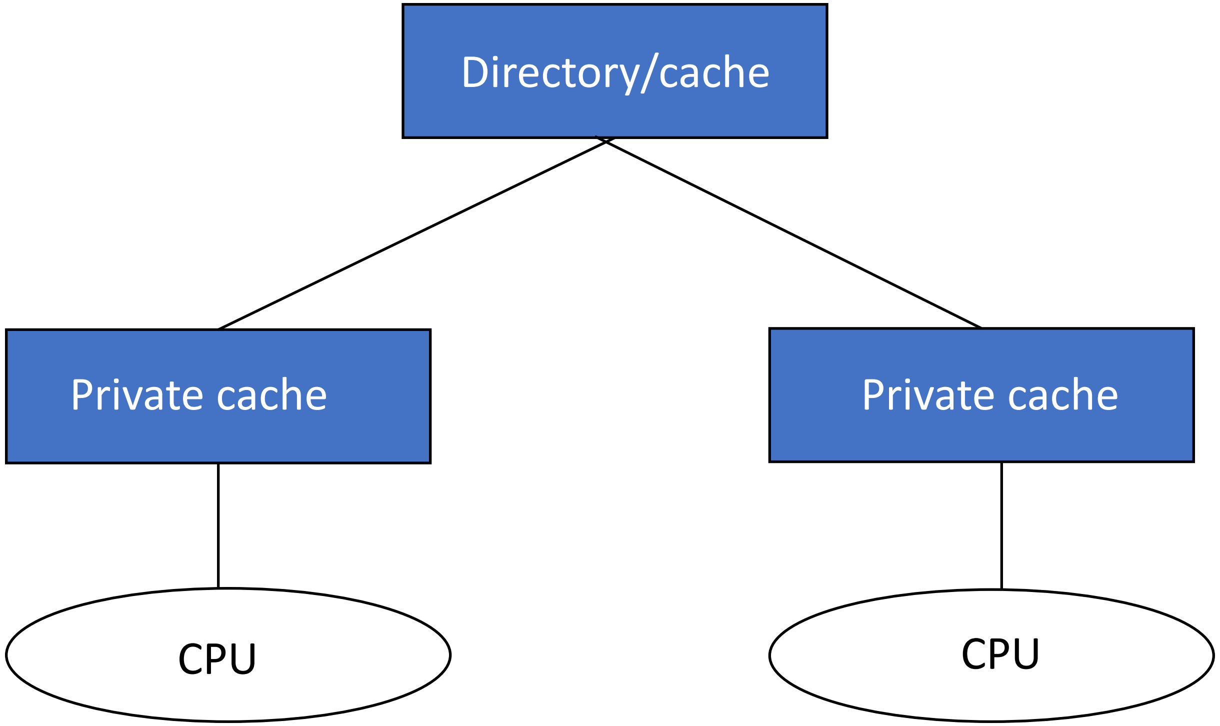
S Shared

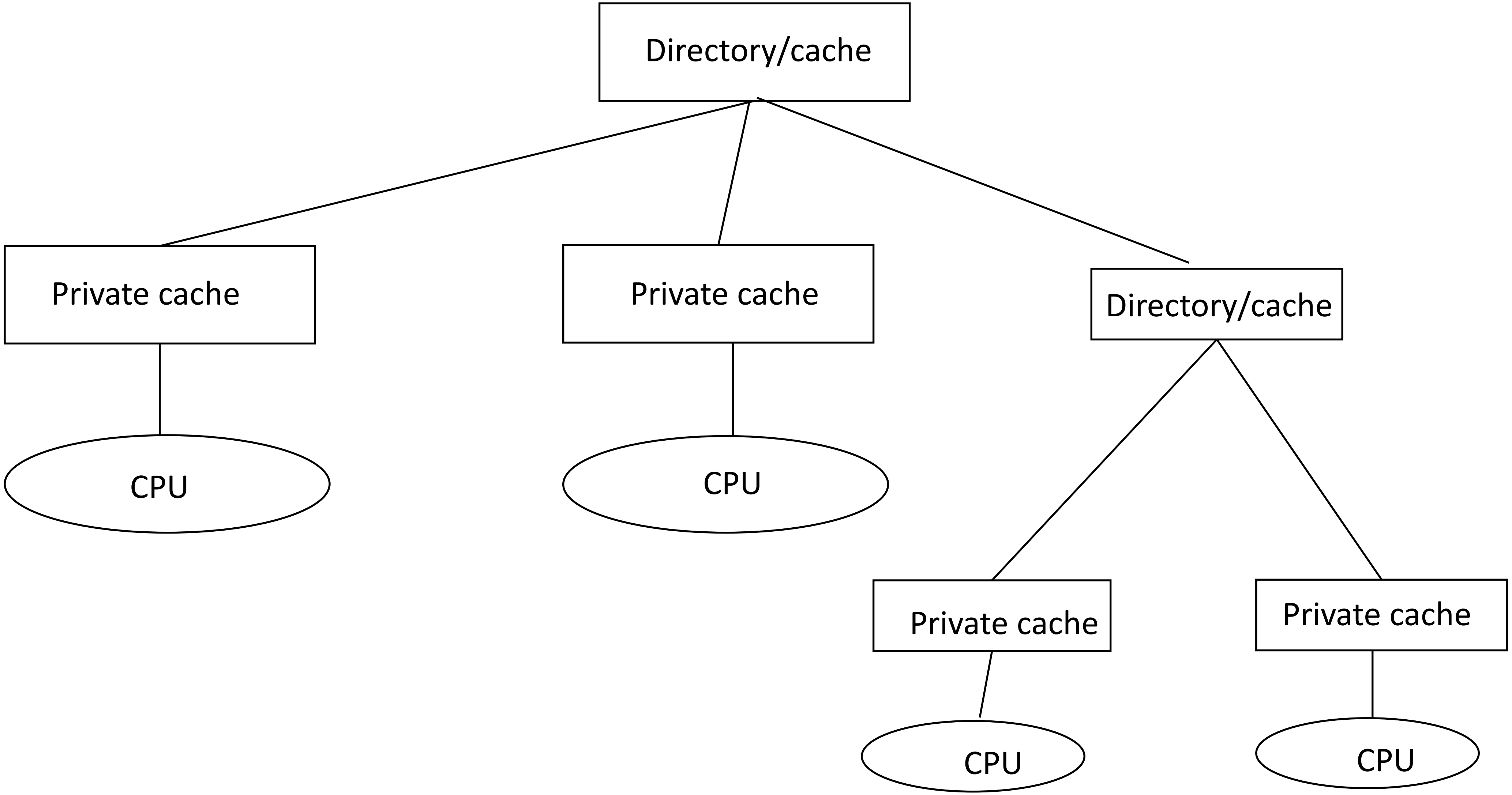
I Invalid

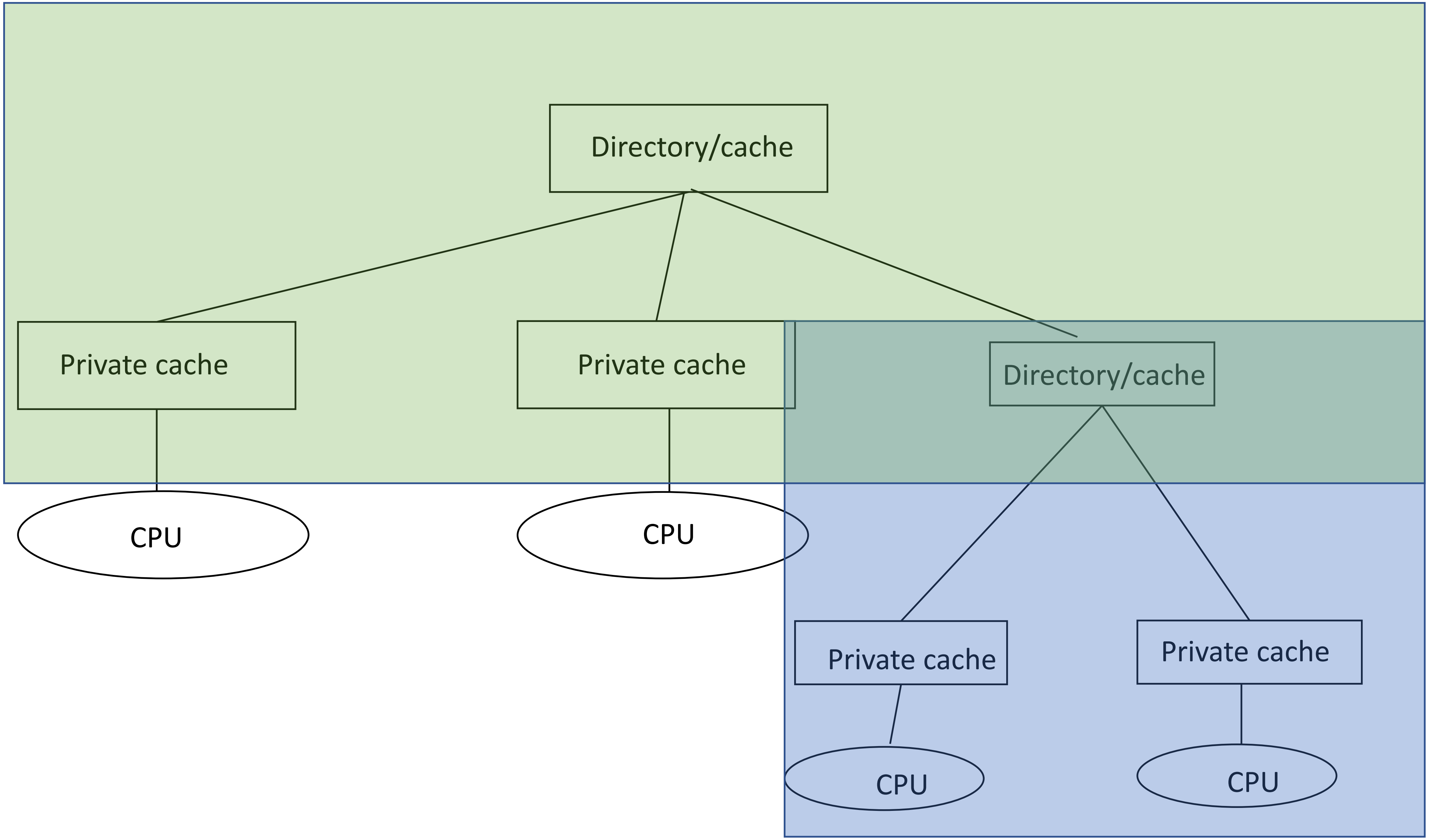

M
S
I



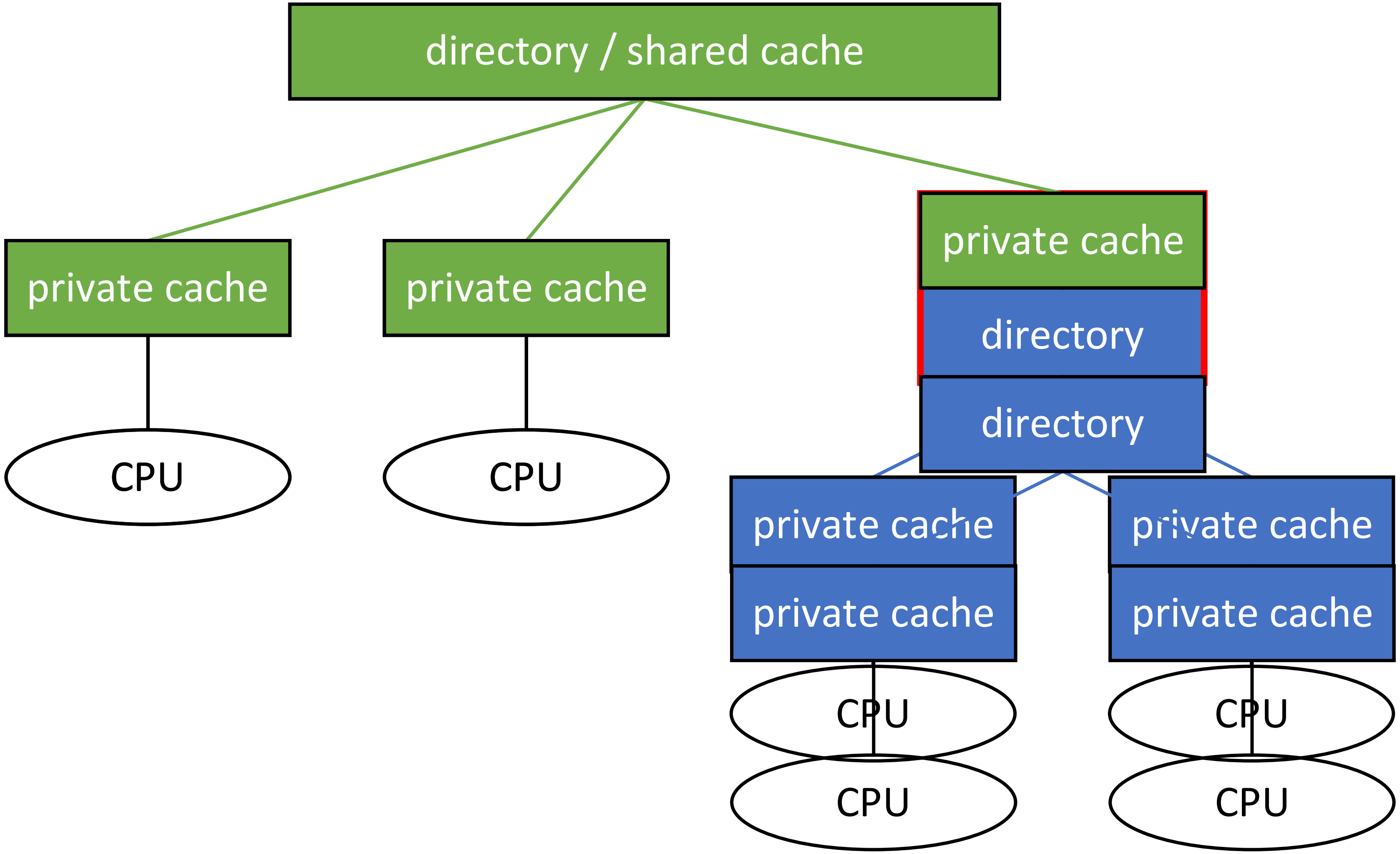
Concurrency!

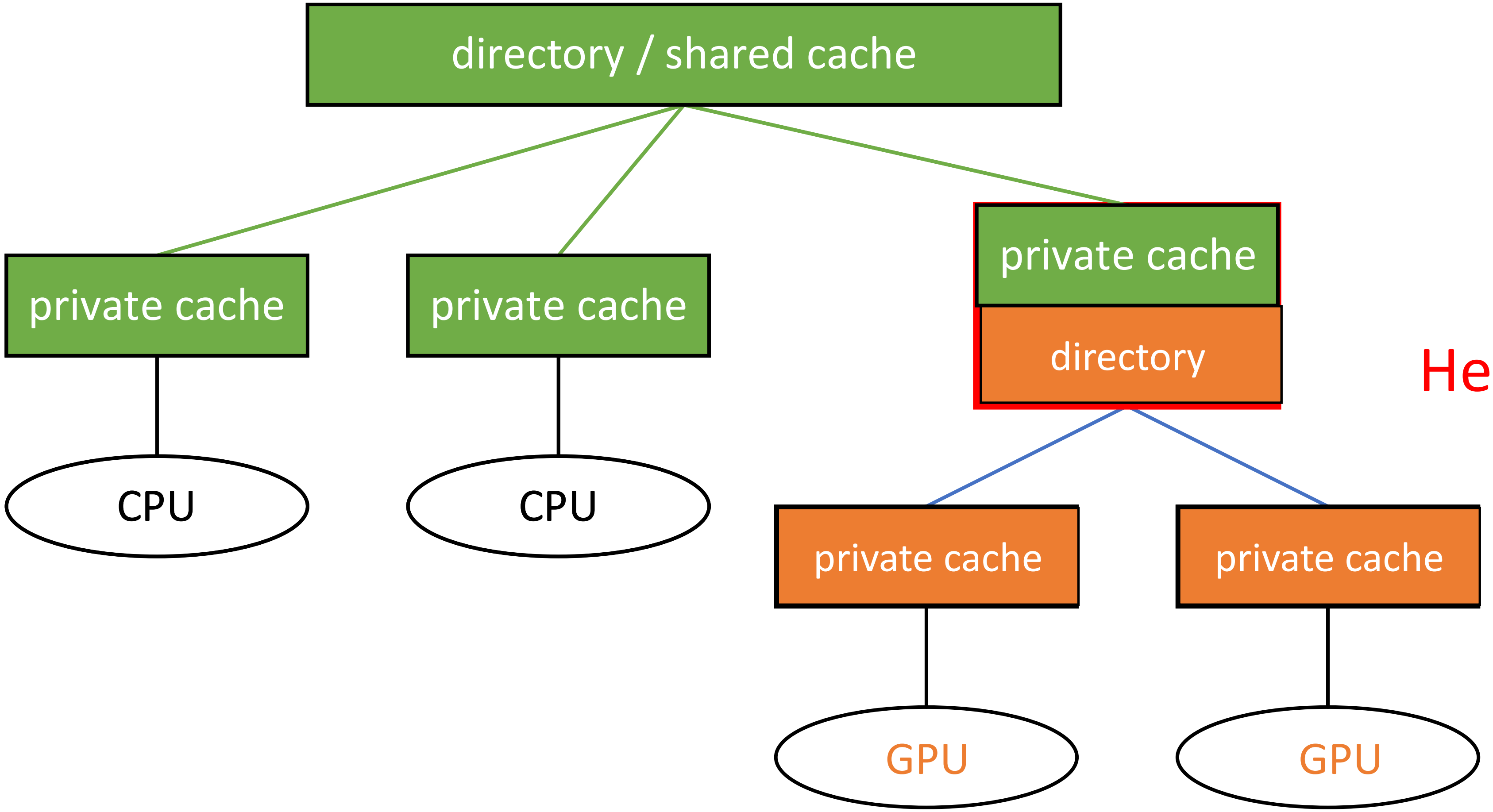

Directory/cache
Private cache
Private cache
CPU
CPU


Directory/cache
Private cache
Private cache
Directory/cache
CPU
CPU
Private cache
Private cache
CPU
CPU


Directory/cache
Private cache
Private cache
Directory/cache
CPU
CPU
Private cache
Private cache
CPU
CPU

Hierarchy!

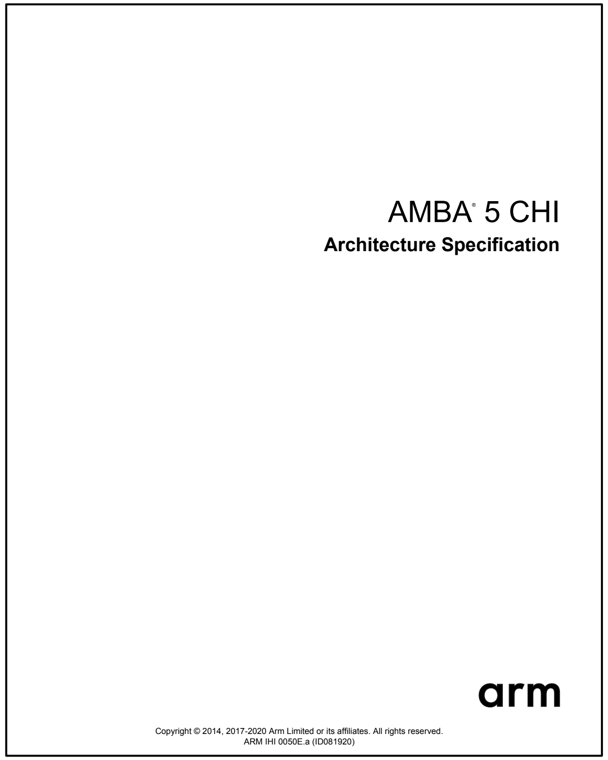

directory / shared cache
private cache
private cache
private cache
directory
directory
CPU
CPU
private cache
private cache
private cache
private cache
CPU
CPU
CPU
CPU



Heterogeneity!

Existing approach and its limitations

- Suppose one wants to build a multiprocessor SoC
  - Read ~100-500 page prose document (e.g., CHI).
  - Implement protocol by hand in Verilog



AMBA® 5 CHI

Architecture Specification

arm

Copyright © 2014, 2017-2020 Arm Limited or its affiliates. All rights reserved.
ARM IHI 0050E.a (ID081920)

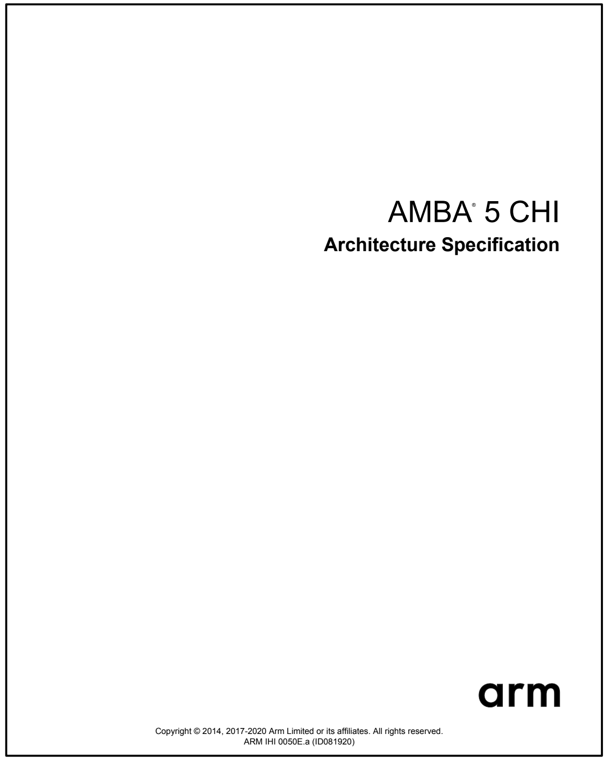
Existing approach and its limitations

- Suppose one wants to build a multiprocessor SoC

  - Read ~100-300 page prose document (Tilelink, CHI).

  - Implement protocol by hand in Verilog

- Limitations
  - Prose = Imprecise
  - Non-exhaustive and conservative
  - Only MOESI

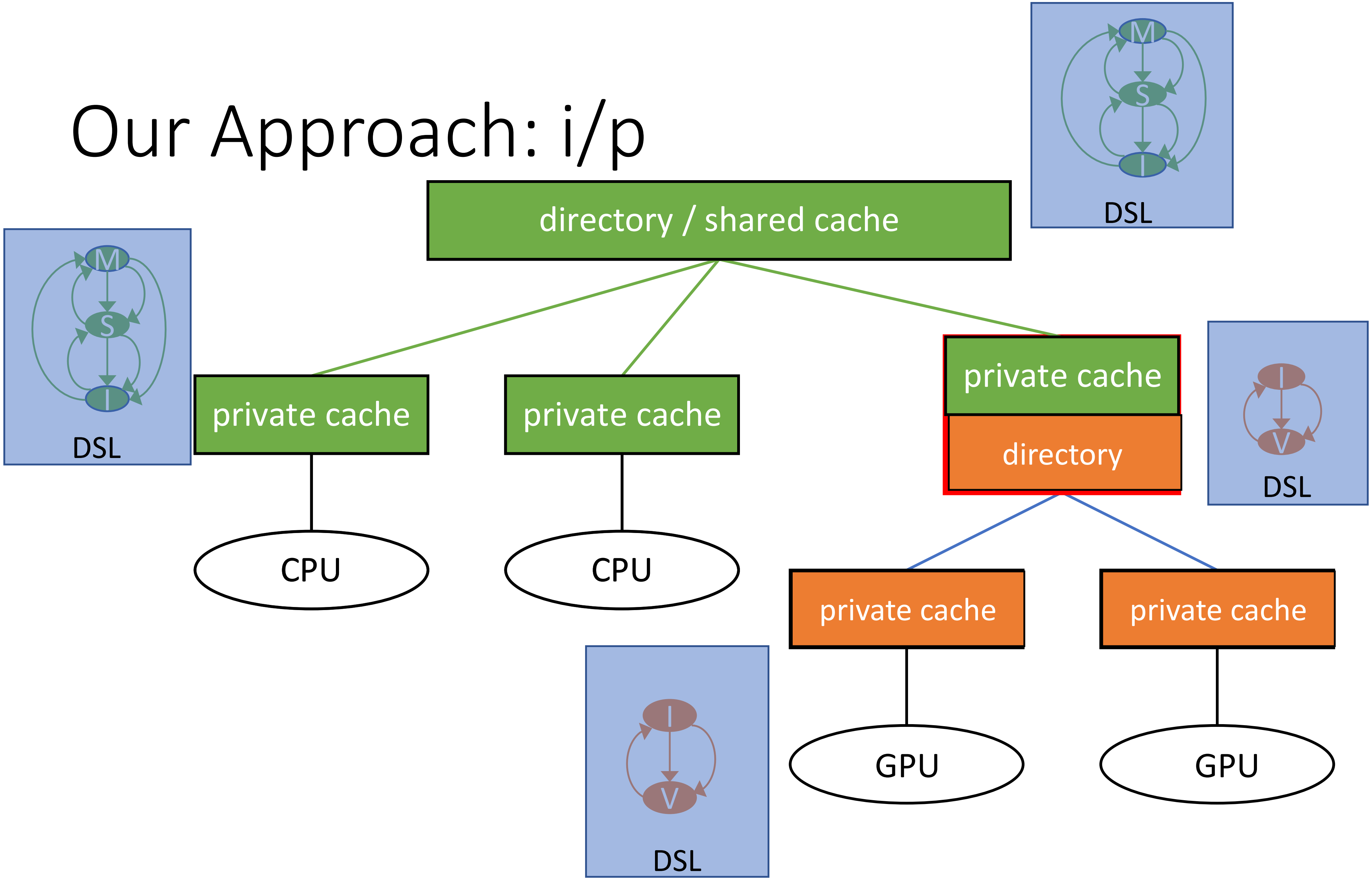


AMBA® 5 CHI
Architecture Specification

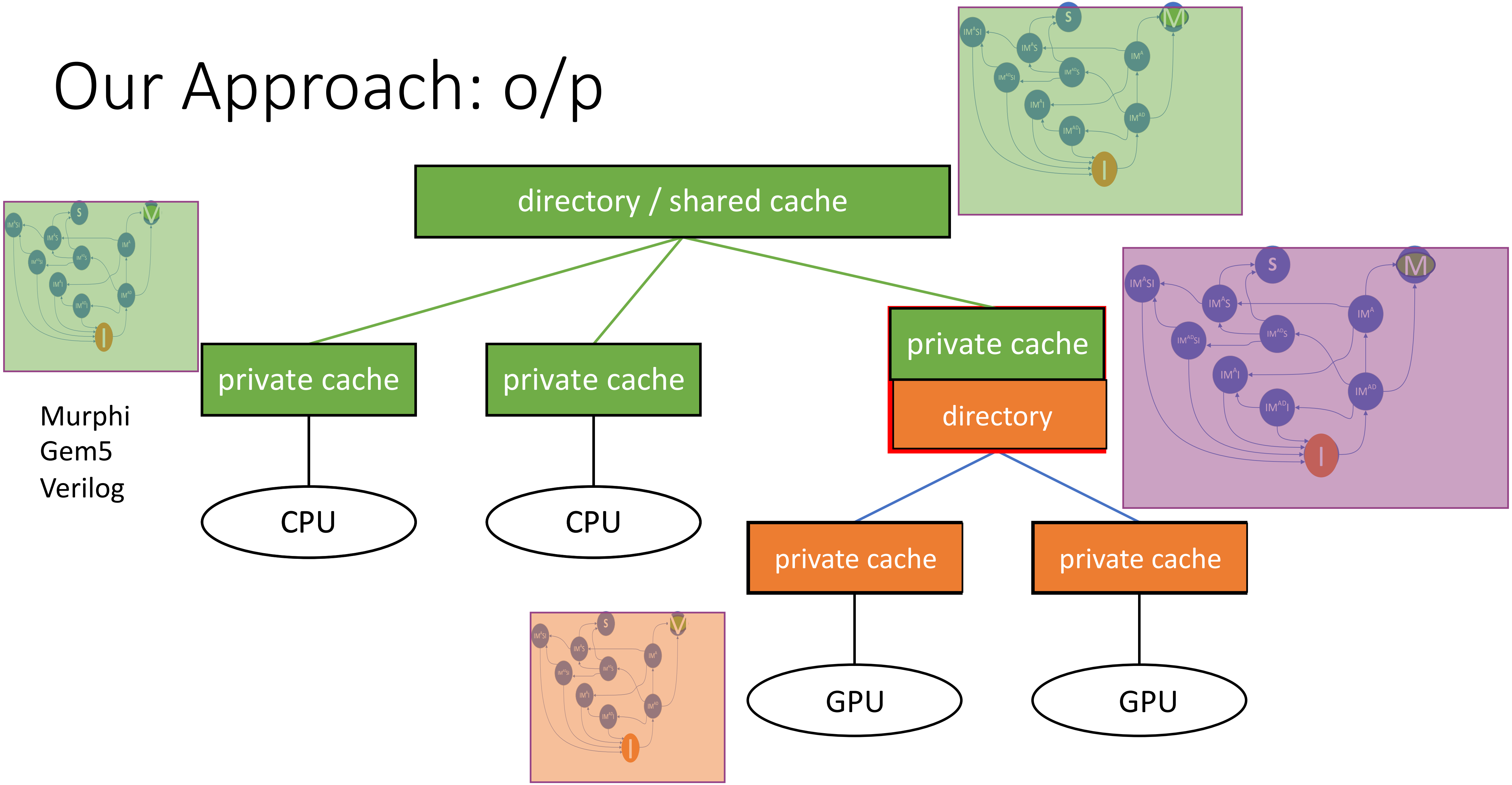
arm

ARM IHI 0050E.a (ID081920)

Our Approach: i/p



Our Approach: o/p

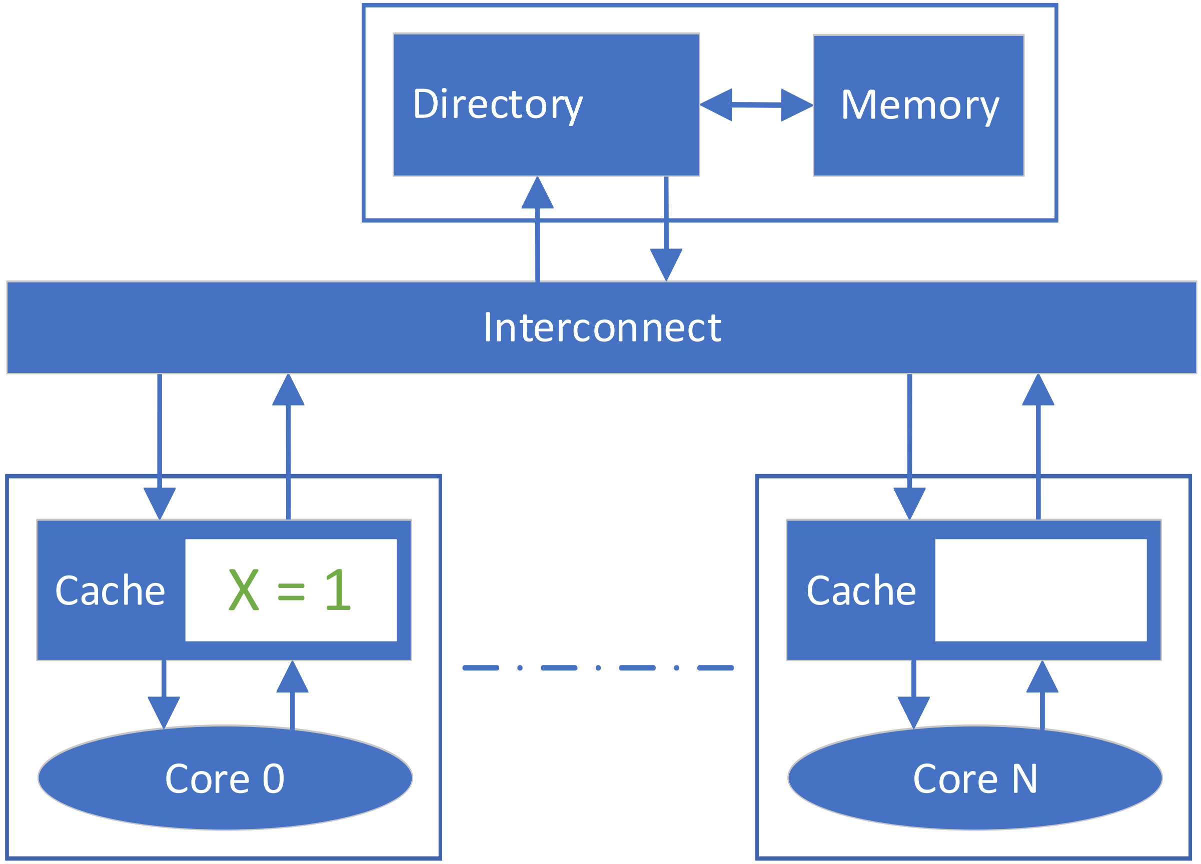


Murphi
Gem5
Verilog

Outline

- Background and Motivation
- Concurrency: ProtoGen
- Hierarchy: HieraGen
- Heterogeneity: HeteroGen
- Coherence for the cloud

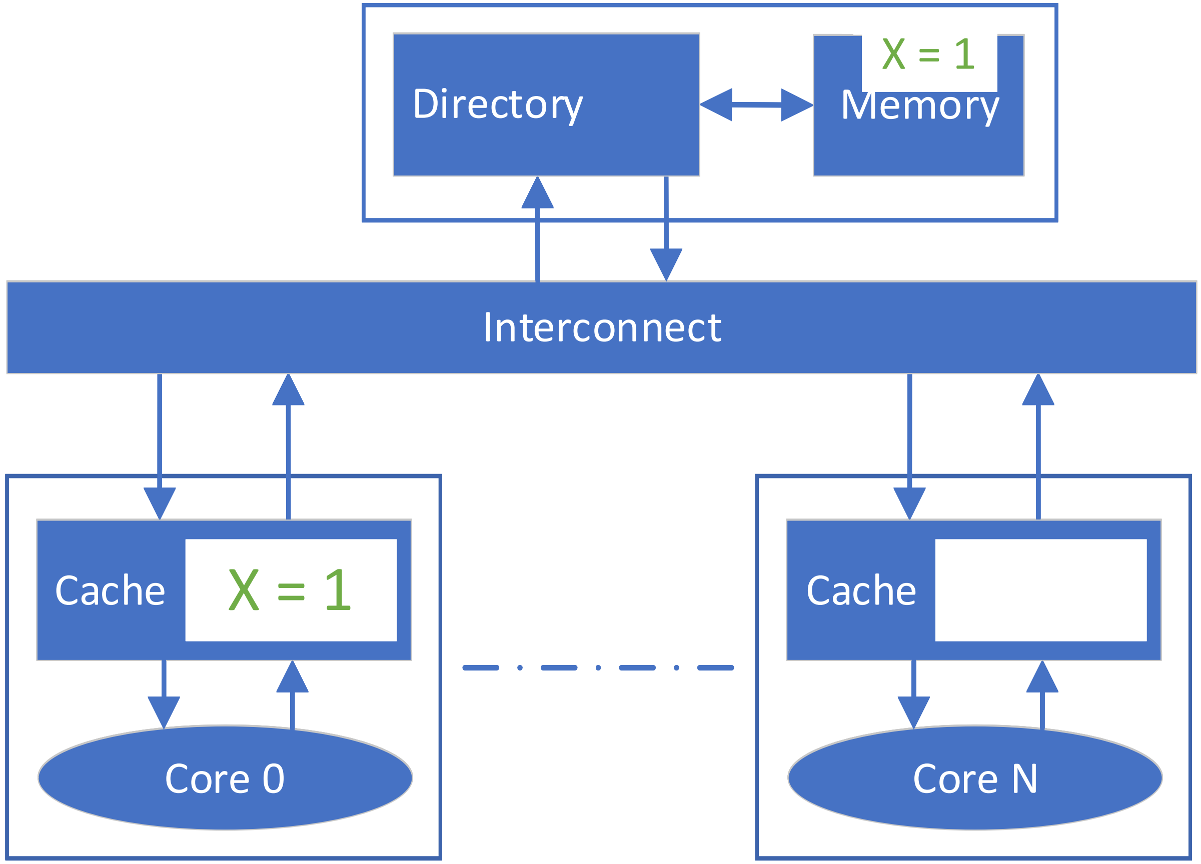
Cache Coherence



- SWMR: single-writer, multiple-reader invariant

Consistency-directed Cache Coherence

Release
(write back)



Acquire
(self-invalidate)

“Cache coherence protocols are notoriously
difficult to design and verif
Memory Systems, 2004]

“The coherence problem is difficult, because it requires coordinating events across nodes” [IEEE Concurrency 2000]

“… directory-based cach
ar

usly

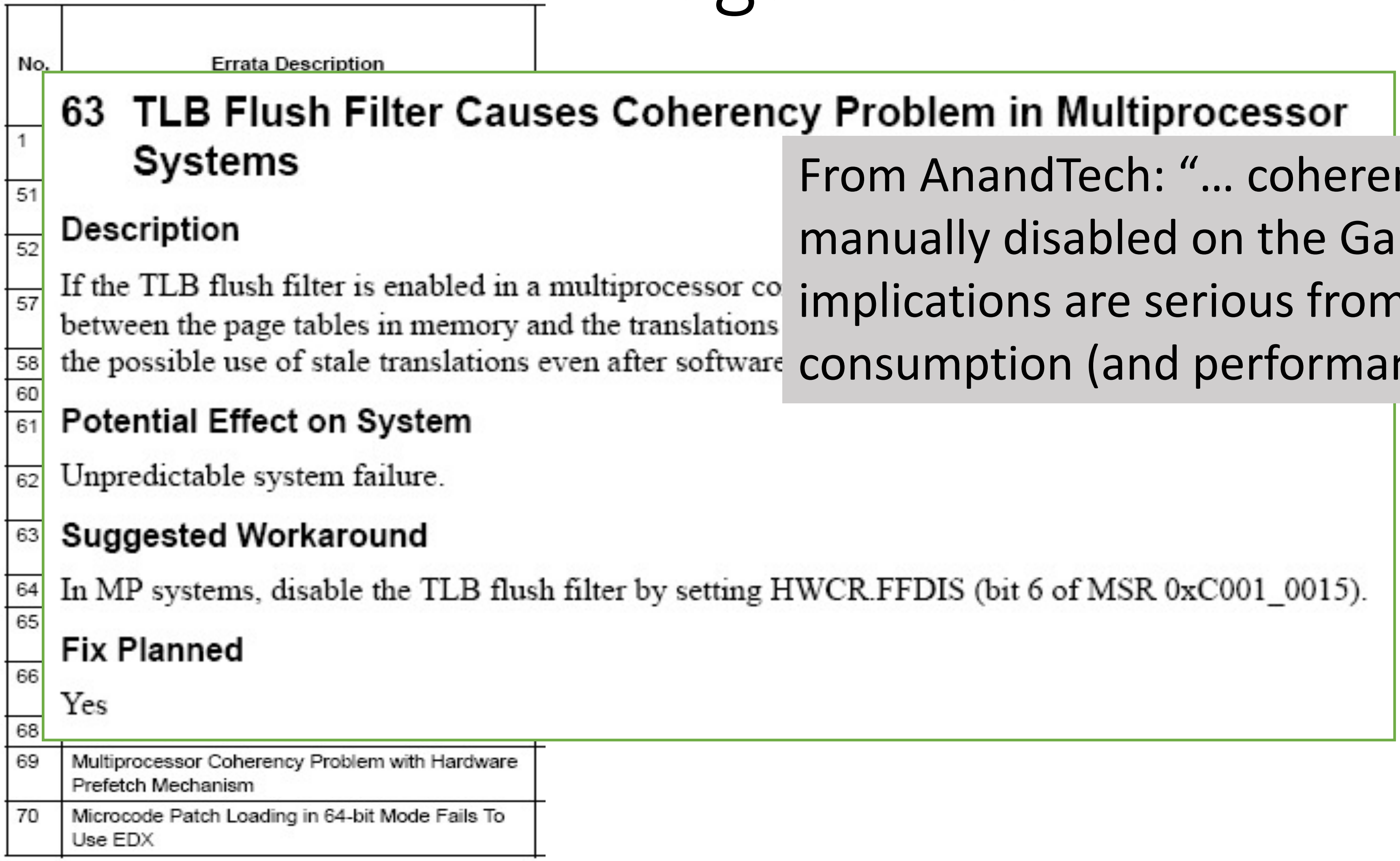
“Sophis
difficult

“… designing and verifying a new hardware coherence protocol is difficult”
[Spandex: A Flexible Interface for Efficient Heterogeneous Coherence - ISCA 2018]

lt to
design and implement correctly” [ASPLOS 2017]

“Cache coherence protocols for distributed shared memory multiprocessors are notoriously difficult to design” [ICFS 1996]

Bugs in the Wild



No.	Errata Description
1	
51	
52	
57	
58	
60	
61	
62	
63	
64	
65	
66	
68	
69	Multiprocessor Coherency Problem with Hardware Prefetch Mechanism
70	Microcode Patch Loading in 64-bit Mode Fails To Use EDX

63 TLB Flush Filter Causes Coherency Problem in Multiprocessor Systems

Description

If the TLB flush filter is enabled in a multiprocessor co
between the page tables in memory and the translations
the possible use of stale translations even after software

Potential Effect on System

Unpredictable system failure.

Suggested Workaround

In MP systems, disable the TLB flush filter by setting HWCR.FFDIS (bit 6 of MSR 0xC001_0015).

Fix Planned

Yes

From AnandTech: "… coherency was broken and manually disabled on the Galaxy S 4. The implications are serious from a power consumption (and performance) standpoint."

Why is Coherence Hard?

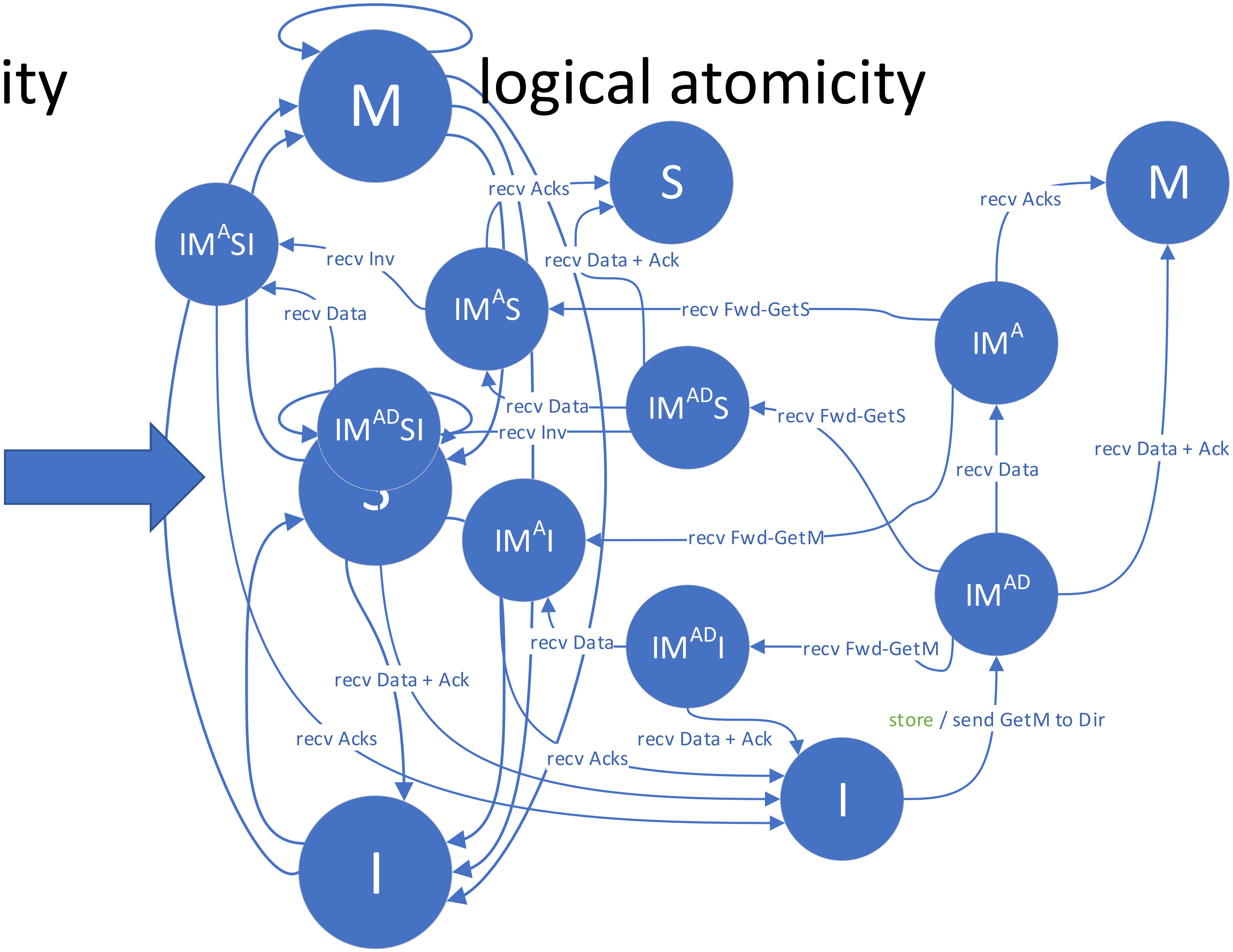
- Concurrency
- Hierarchy
- Heterogeneity

Outline

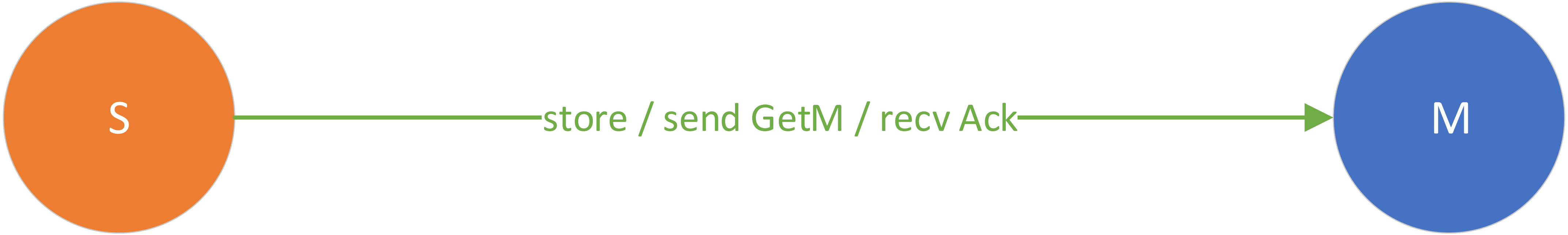
- Background and Motivation
- Concurrency: ProtoGen
- Hierarchy: HieraGen
- Heterogeneity: HeteroGen
- Coherence for the cloud

physical atomicity

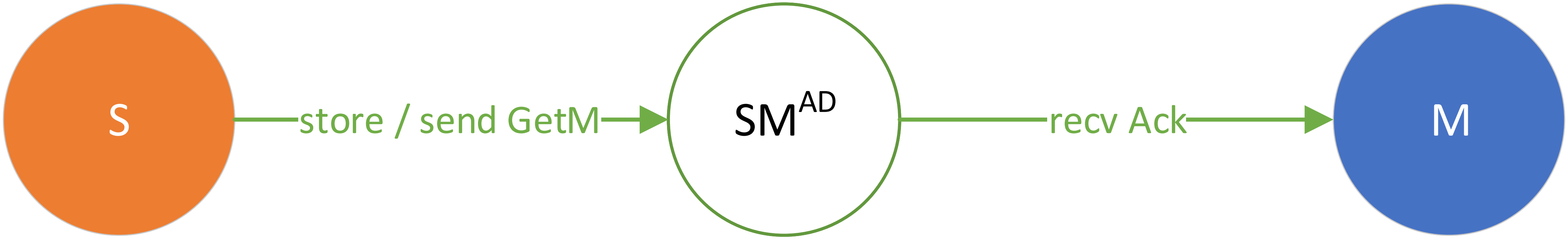
logical atomicity



Atomic S to M Transition

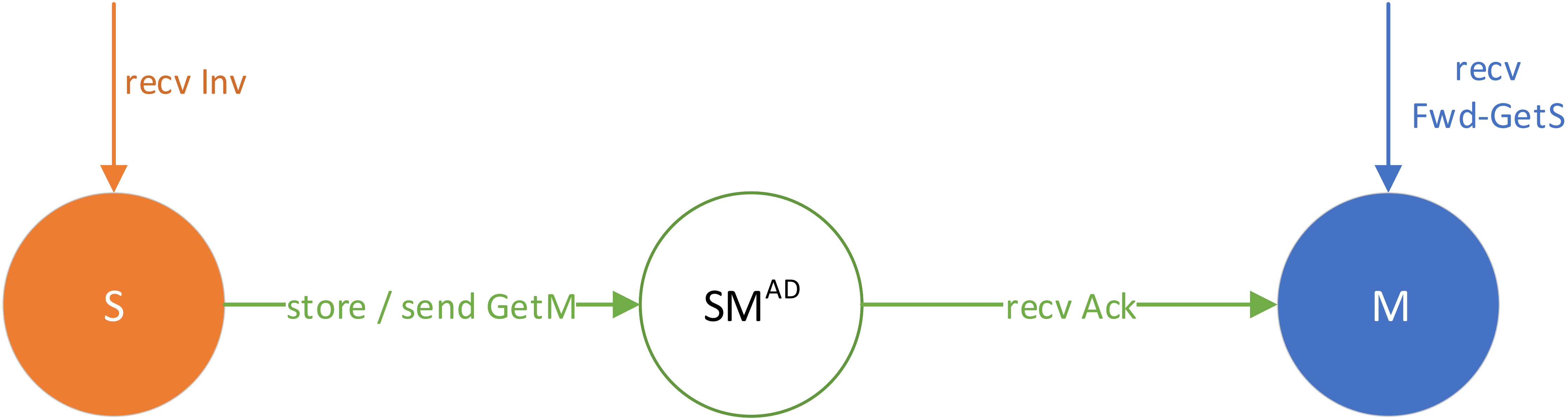


Transient States

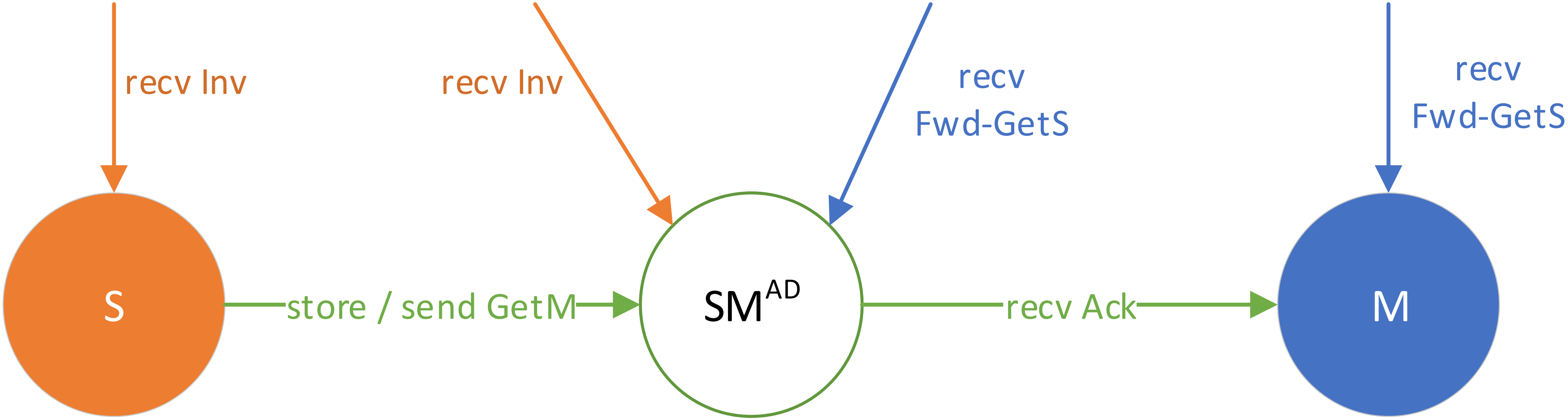


non-atomic transaction

Concurrent Transactions



Concurrent Transactions



non-atomic transactions + concurrency = complexity

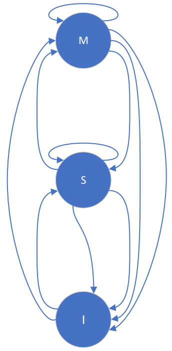
To Summarize…

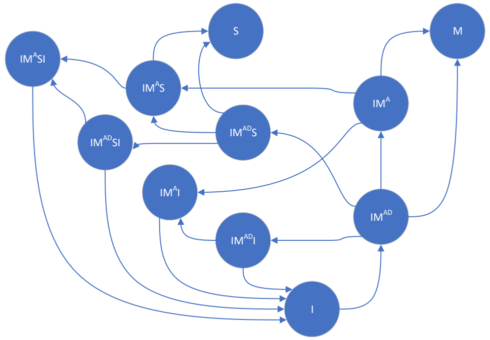
- Stable state protocols assume physically atomic transactions
- Need to support concurrency for performance
- Transient states required to provide logically atomic transactions

Key realization…

- Stable state protocol is a *sequential* specification
- The final protocol is a *non-blocking concurrent* implementation
- Transient states are *synchronization* operations

Insight





Sequential object **{**

…

…

…

}

Non-blocking concurrent **{**

…

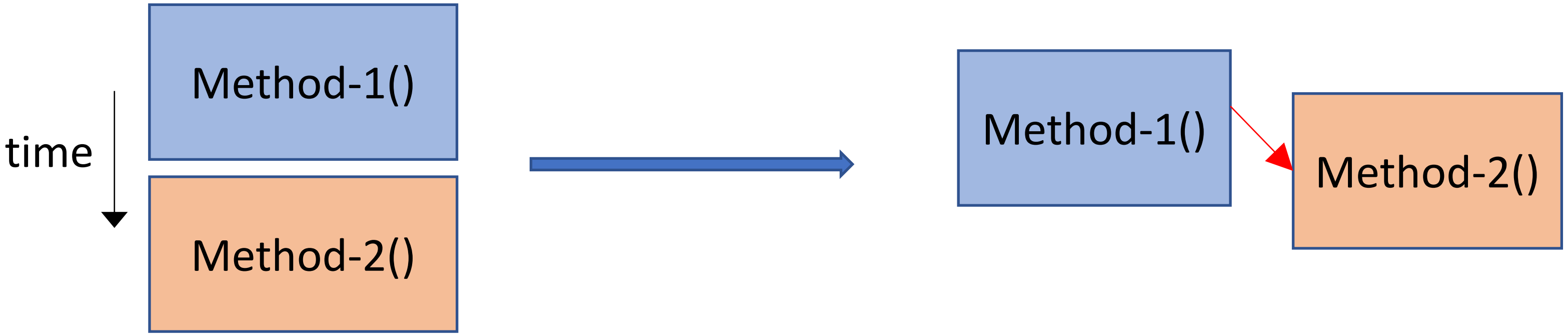
…

…

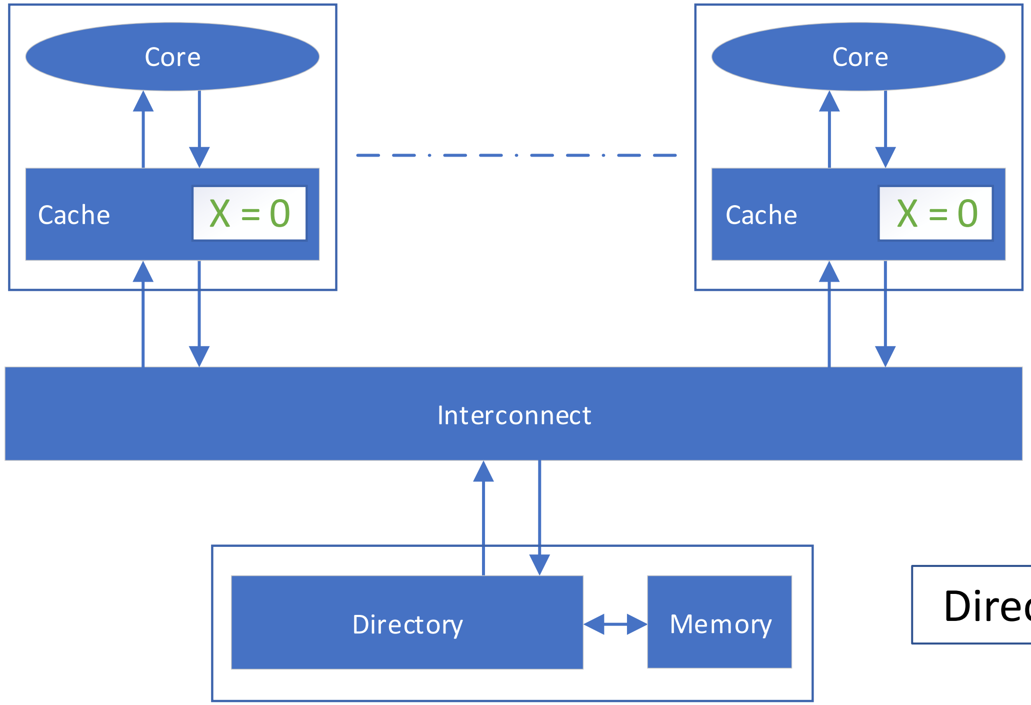
}

No wonder cache coherence protocols are Hard!

Insight



```
concurrent Method-1 {
…
RMW(…); //linearization point
…
}
```



Directory is the Linearization point!

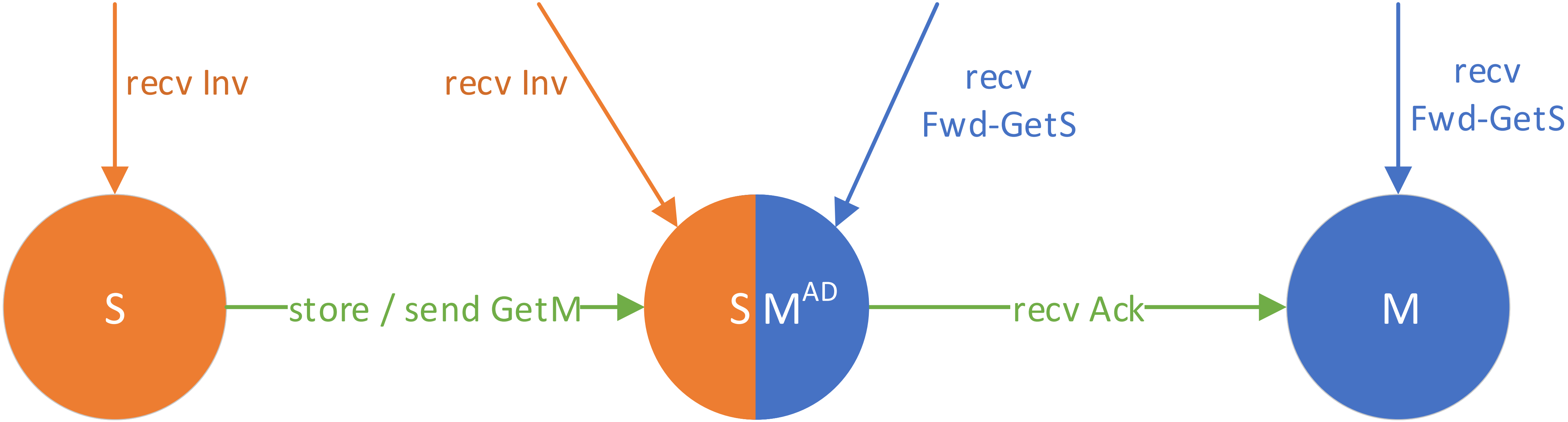
Demystifying Transient States

How do transient states provide logical atomicity?

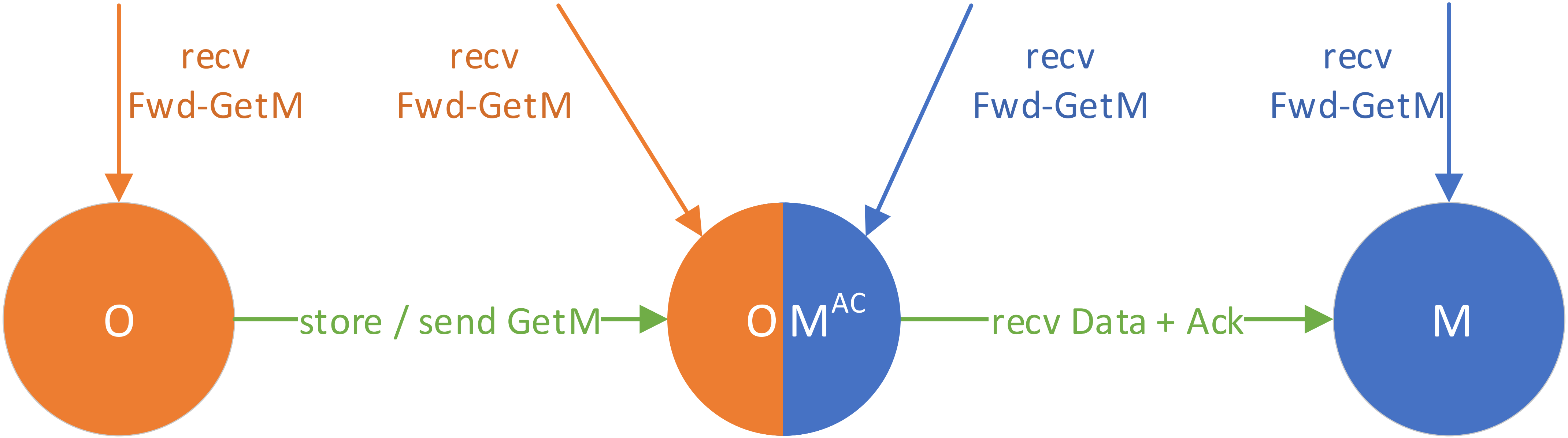
- Convey directory serialization order to caches
- Transient states ensure that caches obey this order

ProtoGen automates by leveraging this insight!

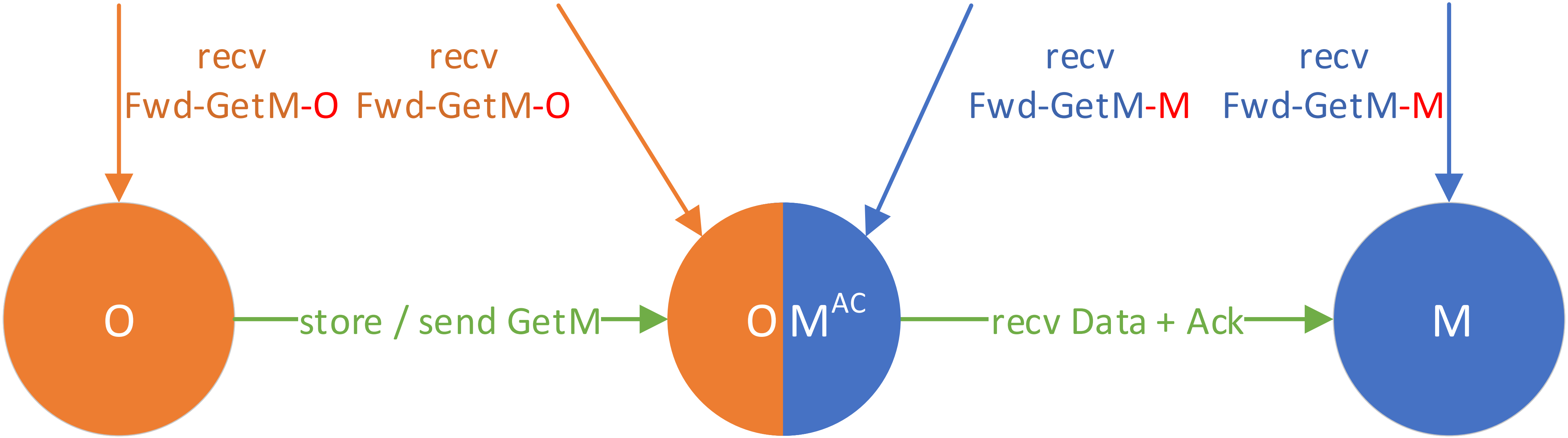
How does cache infer serialization order?



How to resolve name conflicts?



Rename Messages

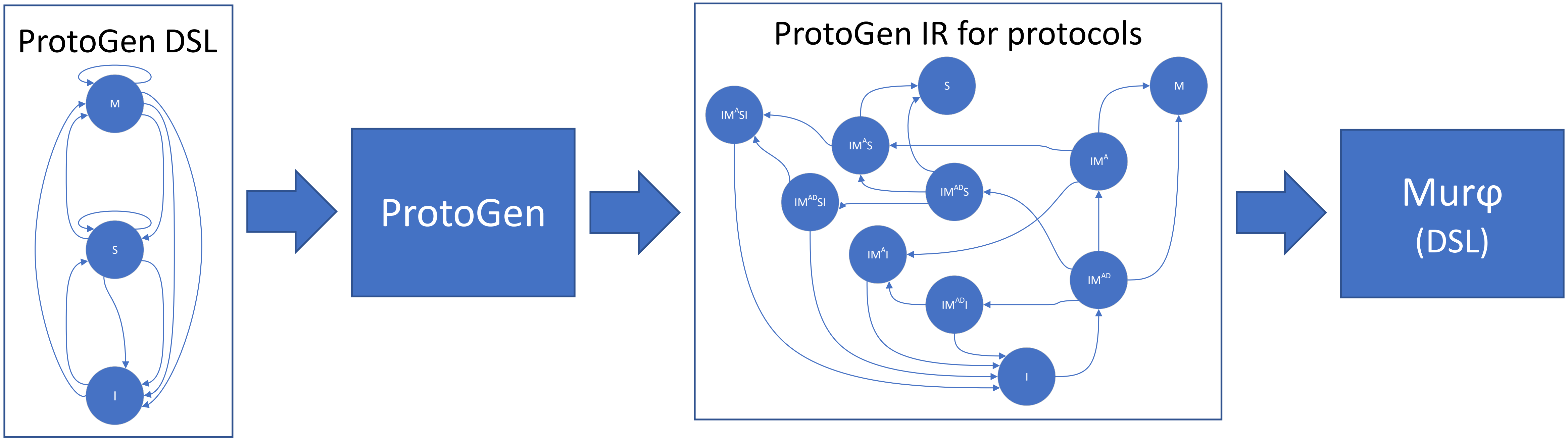


THE UNIVERSITY of EDINBURGH
informatics

ProtoGen Summary

- Infer serialization order from incoming messages
- Rename messages in order to achieve this
- React like in stable state

ProtoGen Tool



ProtoGen DSL

ProtoGen

ProtoGen IR for protocols

Murφ
(DSL)

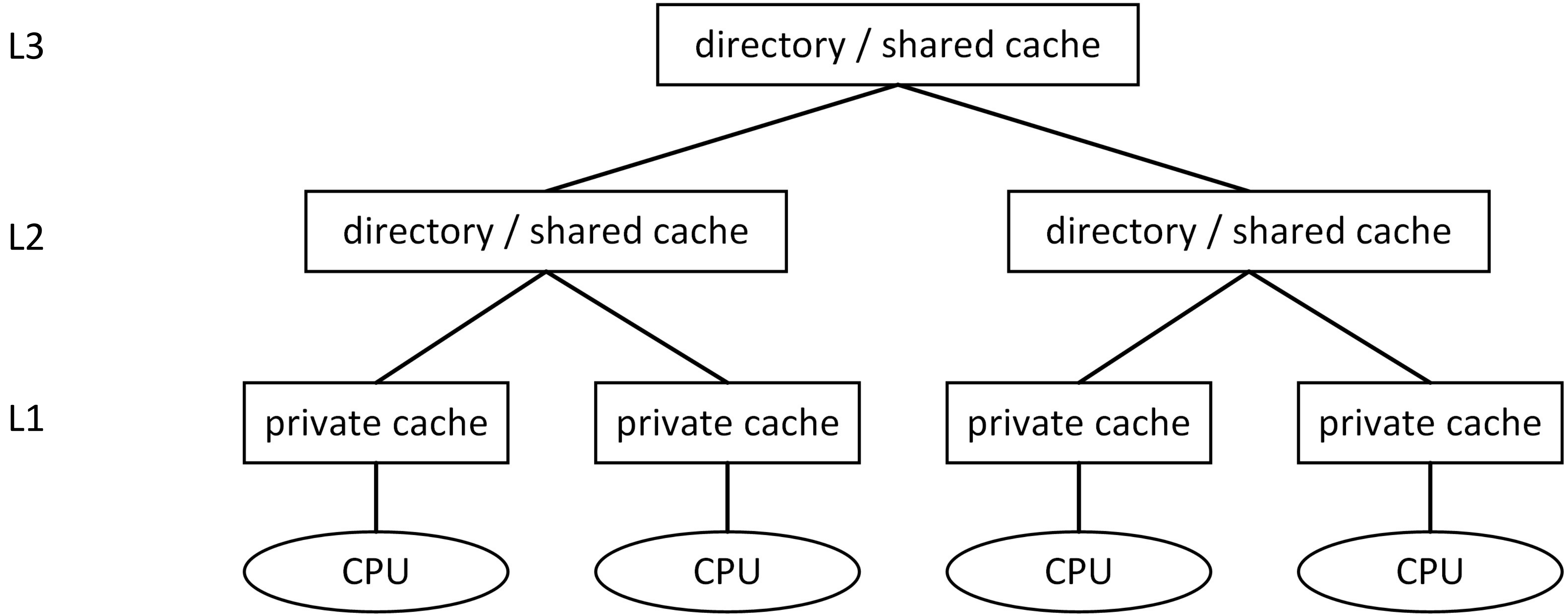
ProtoGen as good (or better) than manually generated protocols

*ISCA'18, IEEE Top Picks Honourable mention

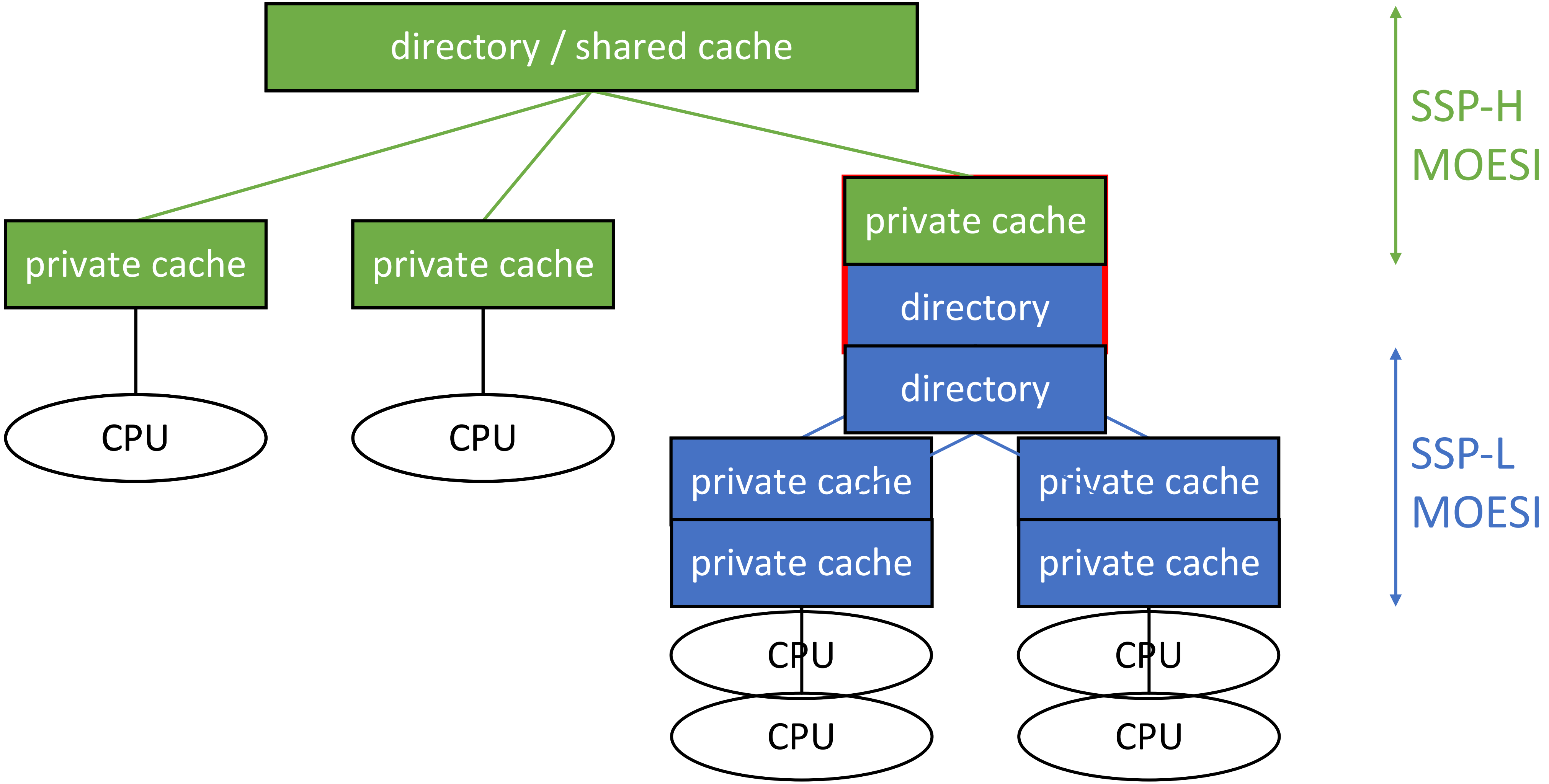
Outline

- Background and Motivation
- Concurrency: ProtoGen
- Hierarchy: HieraGen
- Heterogeneity: HeteroGen
- Coherence for the cloud

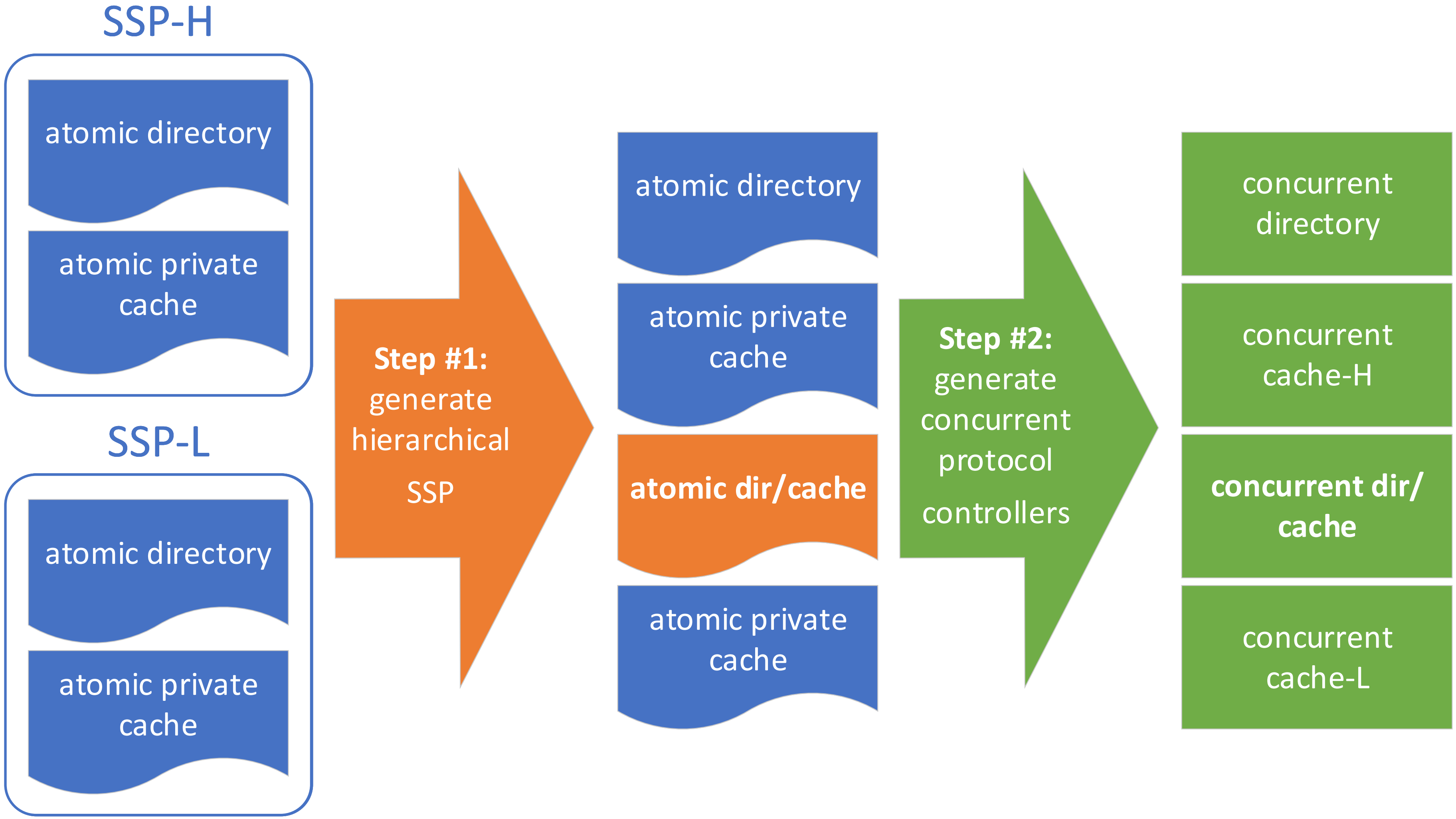
Hierarchical protocols



The Complexity of Hierarchical protocols

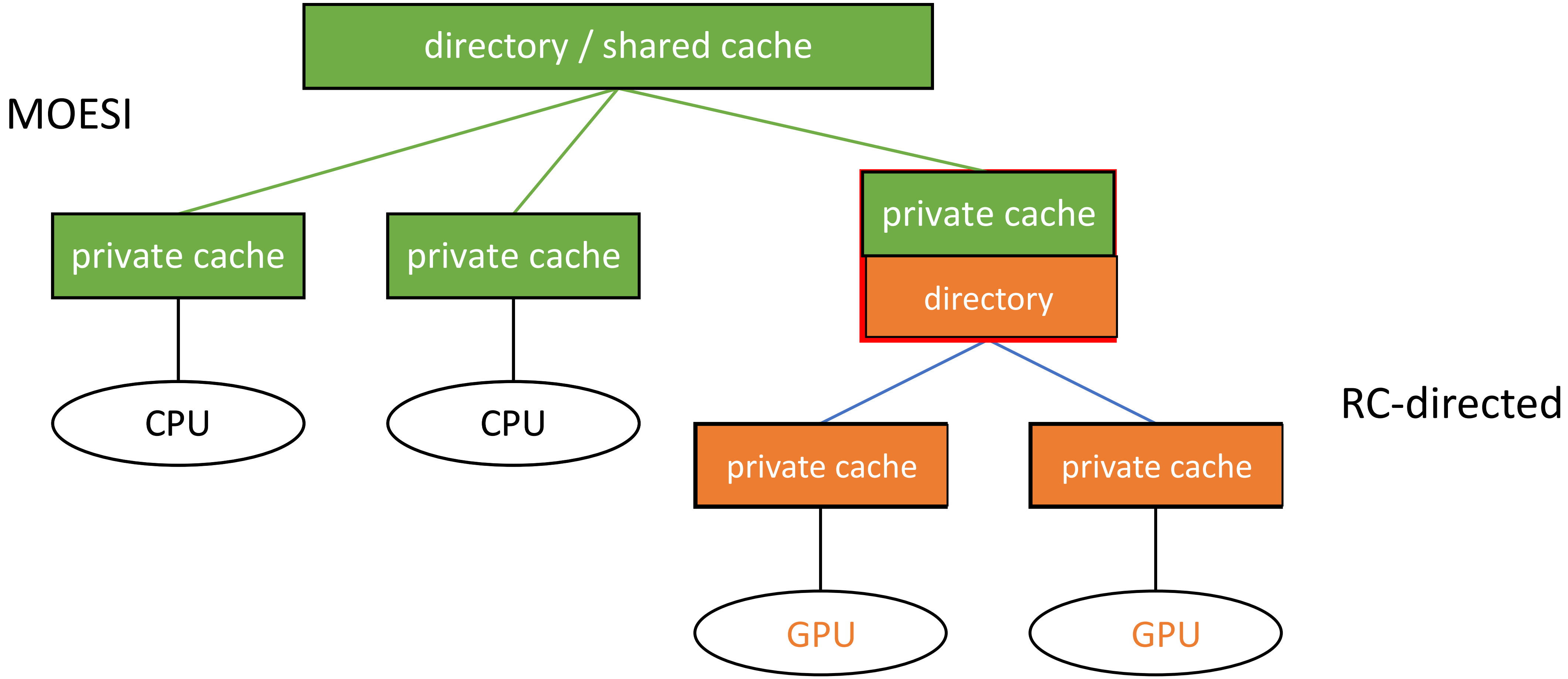


HieraGen* tool flow



*ISCA'20

THE UNIVERSITY of EDINBURGH informatics



MOESI

directory / shared cache

private cache

private cache

private cache

directory

private cache

private cache

CPU

CPU

GPU

GPU

RC-directed

HeteroGen

- How do you stich together two different protocols?
  - HieraGen should work!

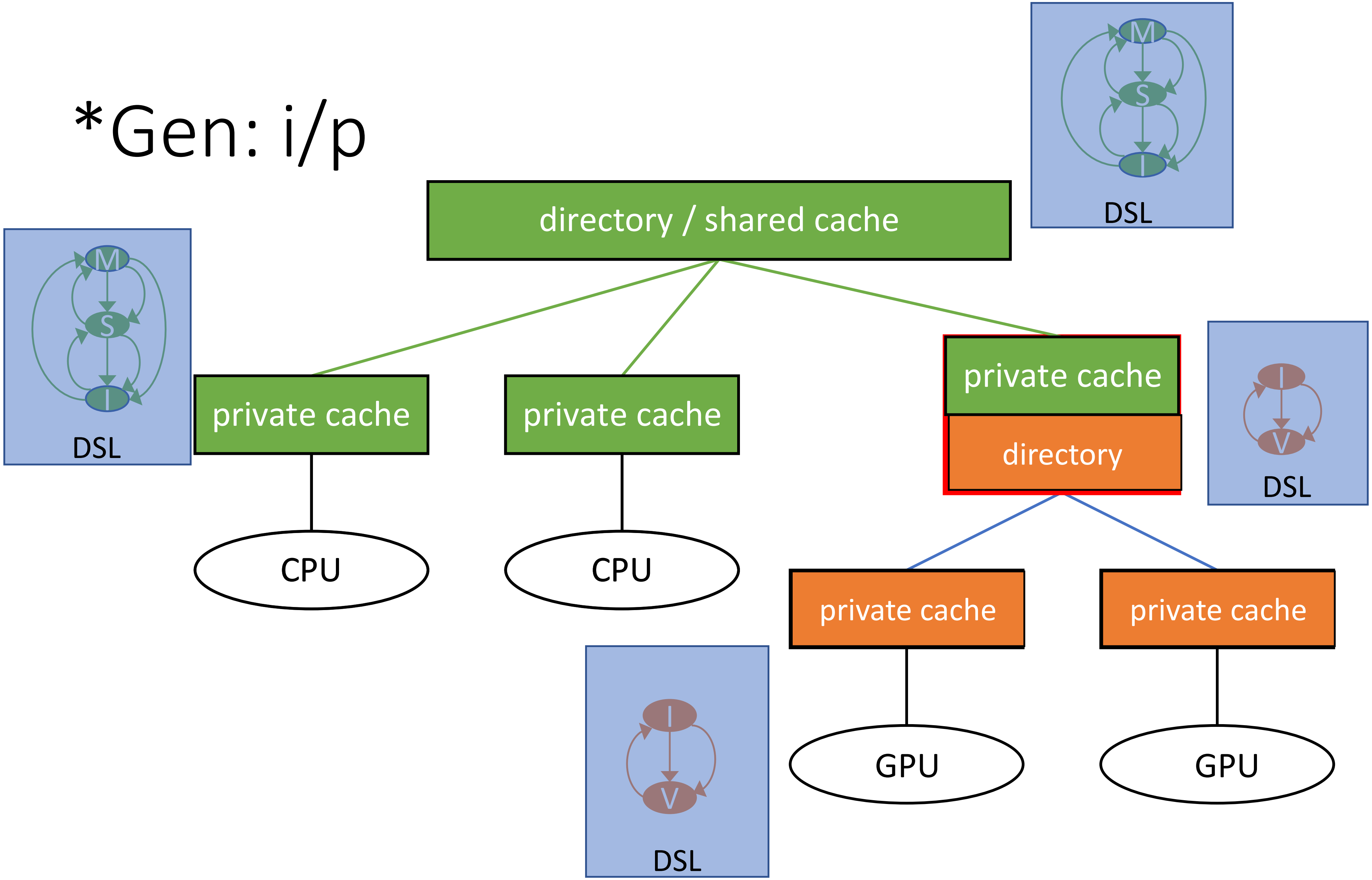
- What is the correctness condition?
  - MOESI style protocols SC
  - RC-style protocols…RC
  - Compound consistency models!

[under submission]

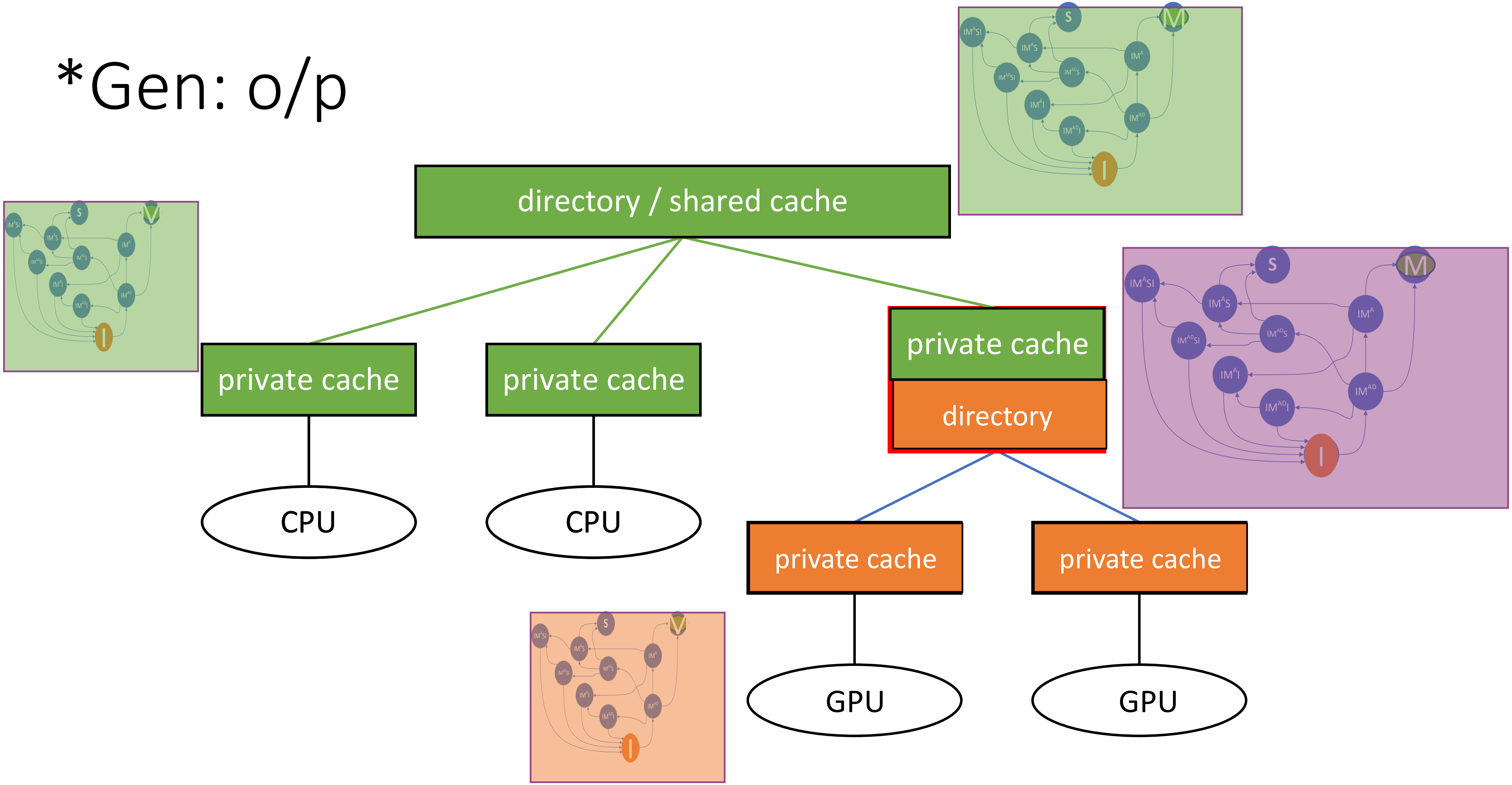
Compound Consistency

- Correctness condition for heterogeneous coherence
- Foundation for heterogeneous consistency
- Each cluster can assume its own memory model

*Gen: i/p


M
S
I
DSL
directory / shared cache
M
S
I
DSL
private cache
private cache
private cache
directory
I
V
DSL
CPU
CPU
private cache
private cache
I
V
DSL
GPU
GPU

*Gen: o/p



directory / shared cache

private cache

private cache

private cache

directory

CPU

CPU

private cache

private cache

GPU

GPU

Outline

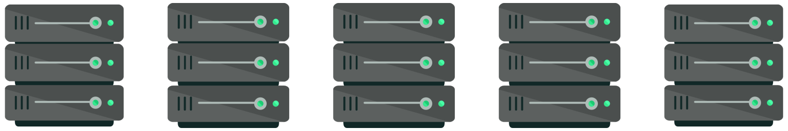
- Background and Motivation
- Concurrency: ProtoGen
- Hierarchy: HieraGen
- Heterogeneity: HeteroGen
- Coherence for the cloud

Datacentre Distributed datastores

In-memory with read/write API

Backbone of online services

redis
APACHE ZooKeeper™
cassandra
hazelcast
riak
apache Ignite™
mongoDB®



Distributed Datastore

Distributed datastores

In-memory with read/write API

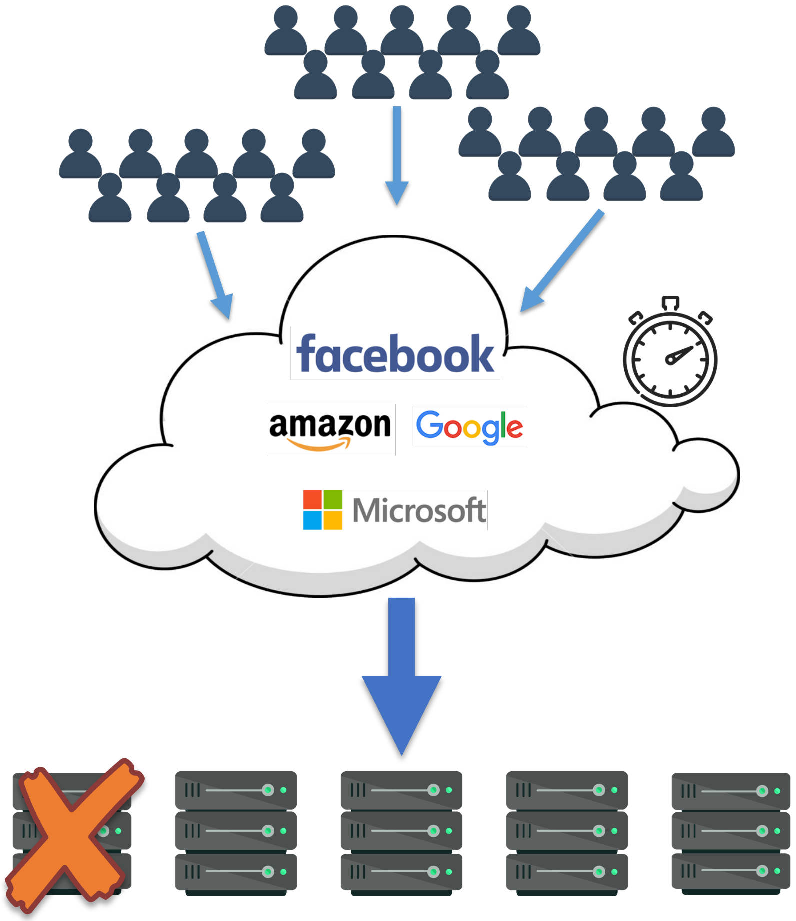
Backbone of online services

Need:

Consistency (Programmability)

High performance

Fault tolerance (Availability)



Distributed Datastore

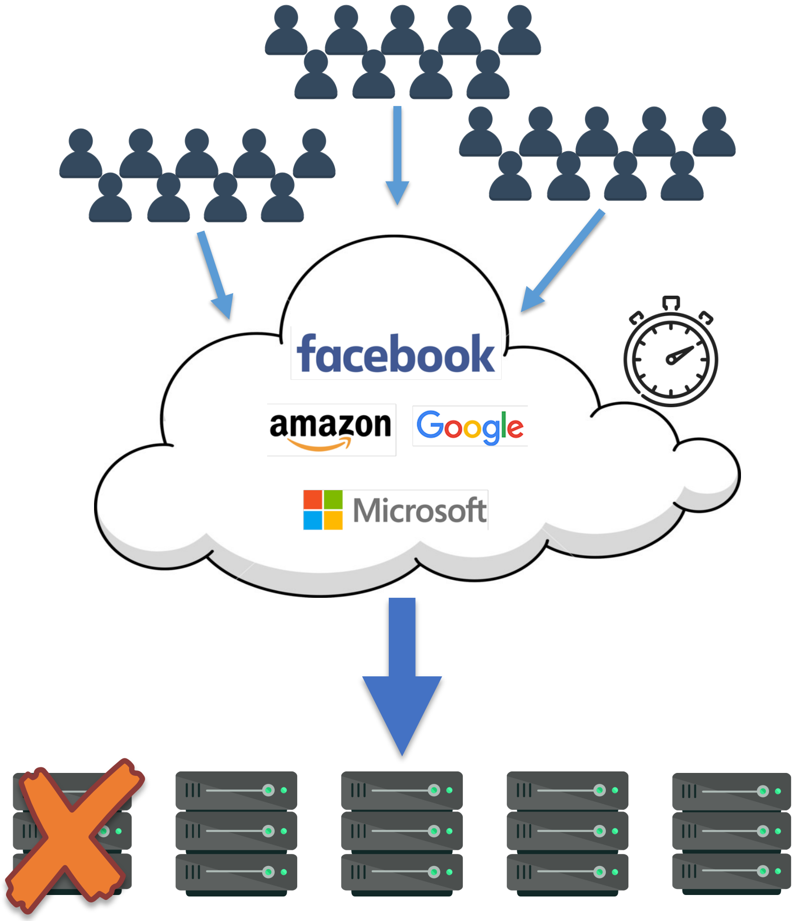
Distributed datastores

In-memory with read/write API

Backbone of online services

Need:

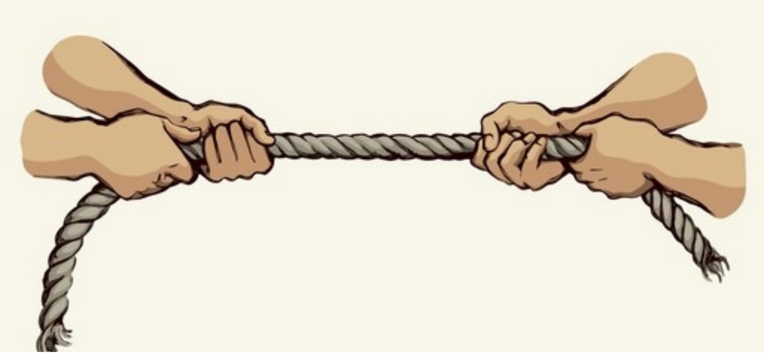
- **Consistency (Programmability)**
- **High performance**
- **Fault tolerance (Availability)**



High-performance Reliable Replication Protocol!

The problem

Performance

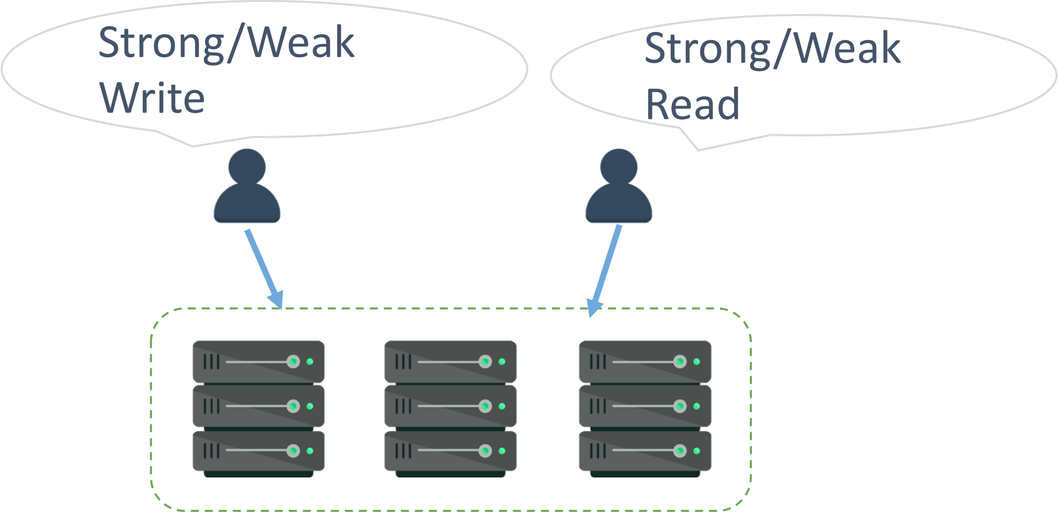


Strong
Consistency

Replication ⇒ Performance vs Consistency

Existing Solution: Multiple Consistency Levels (MCL)

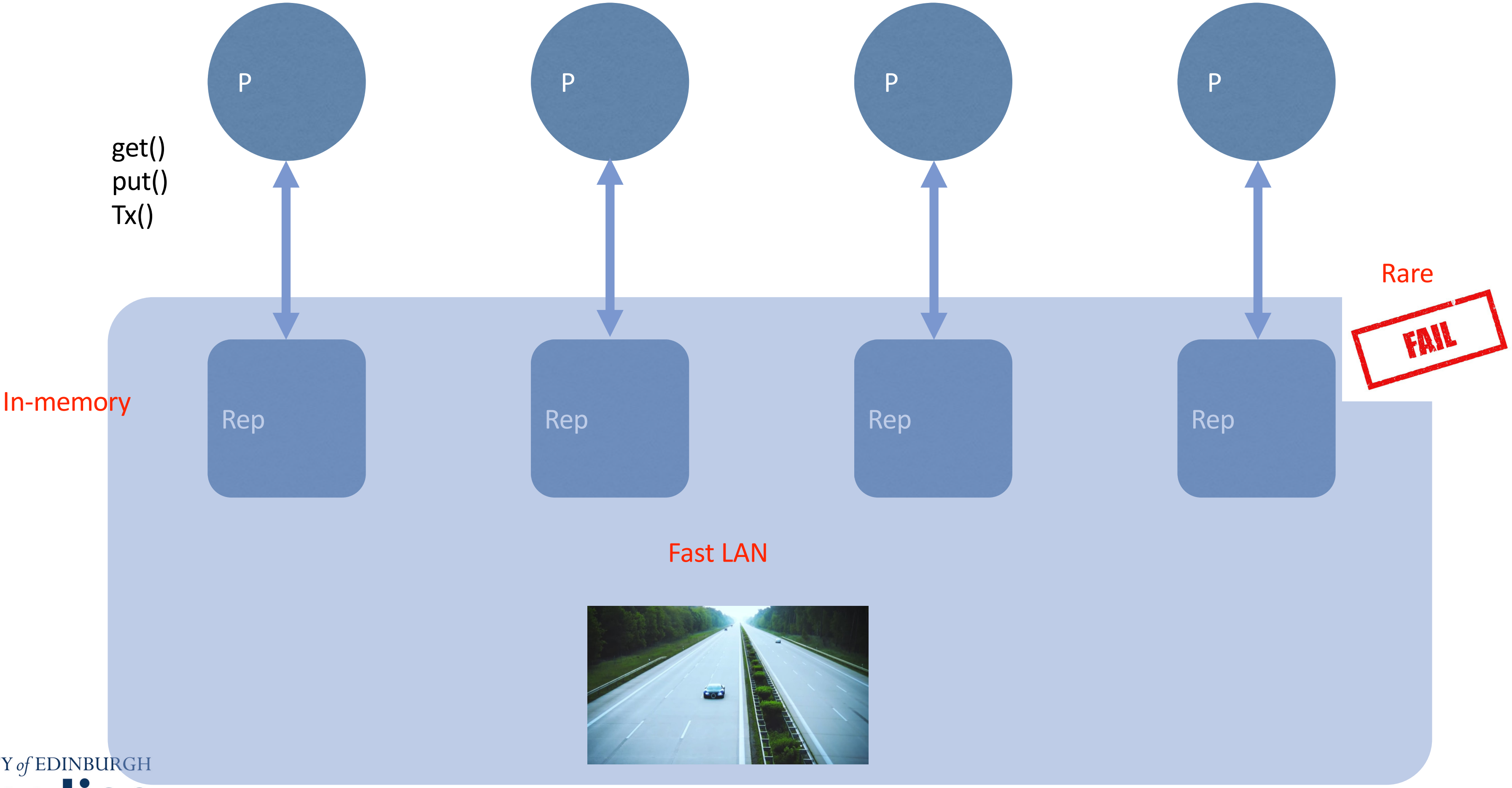
aws	Amazon DB
G	App Engine
YAHOO!	PNUTS
	Manhattan
Microsoft Azure	Pileus



MCL Replicated KVS

What about programmability?

Datacentre Distributed datastore



P

P

P

P

get()
put()
Tx()

Rep

Rep

Rep

Rep

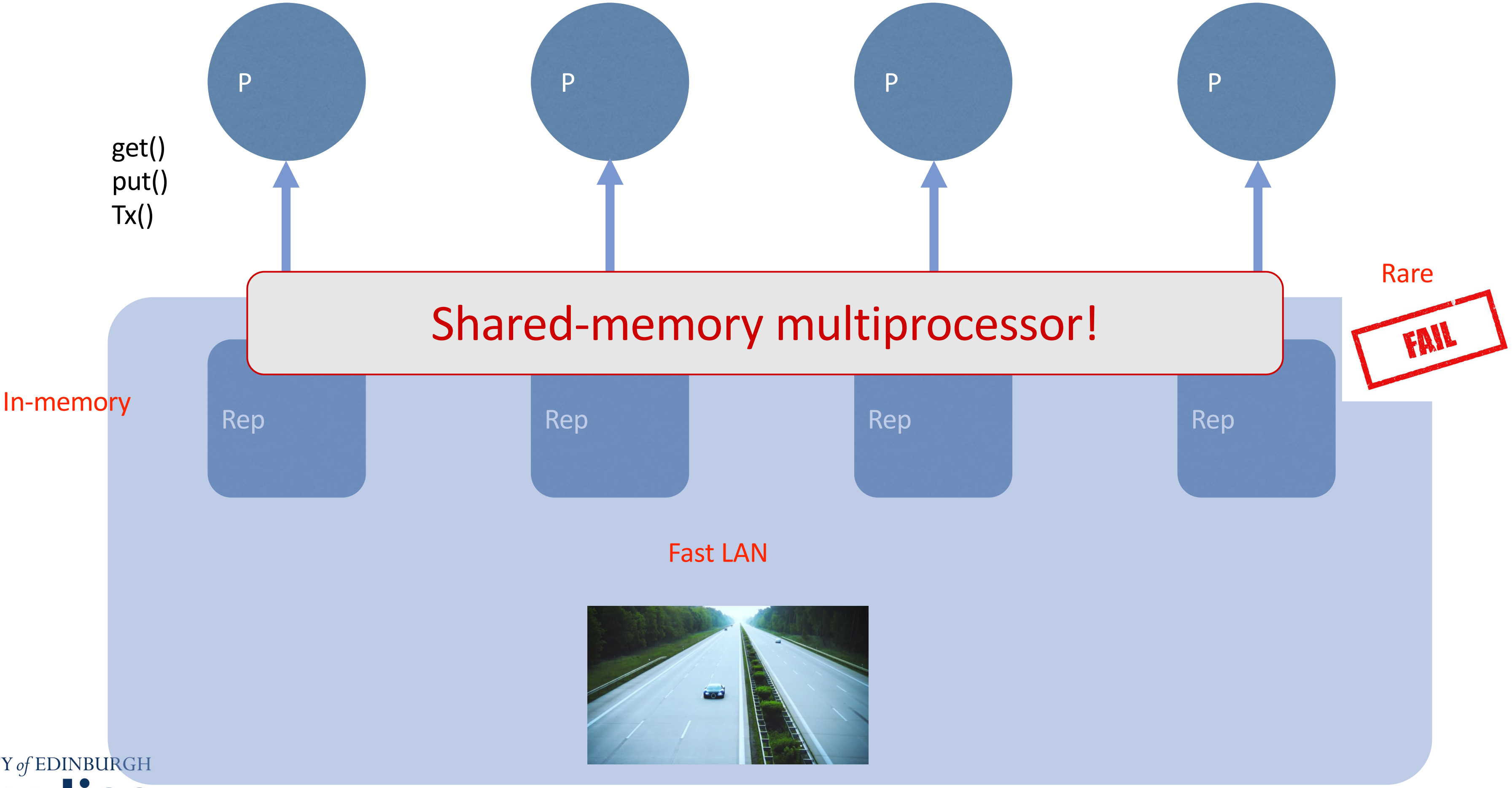
In-memory

Rare

FAIL

Fast LAN

Datacentre Distributed datastore



P

P

P

P

get()
put()
Tx()

Shared-memory multiprocessor!

Rare

FAIL

In-memory

Rep

Rep

Rep

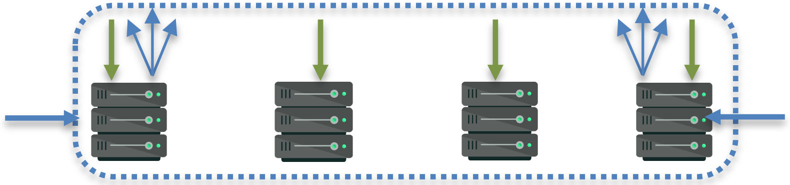
Rep

Fast LAN

Datacentre replication = Fast-path coherence + slow-path consensus?

Hermes [ASPLOS ’20] [Top Picks honorable mention]
Kite [PPoPP’20] [Best paper nominee]

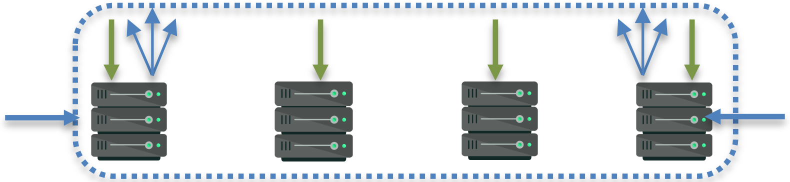
Coherence-inspired Hermes



Broadcast-based invalidating

*ASPLOS'20, IEEE Top Picks Honourable mention

Coherence-inspired Hermes



Broadcast-based invalidating

Local reads

Fast, concurrent writes

Protocol reliable but blocking

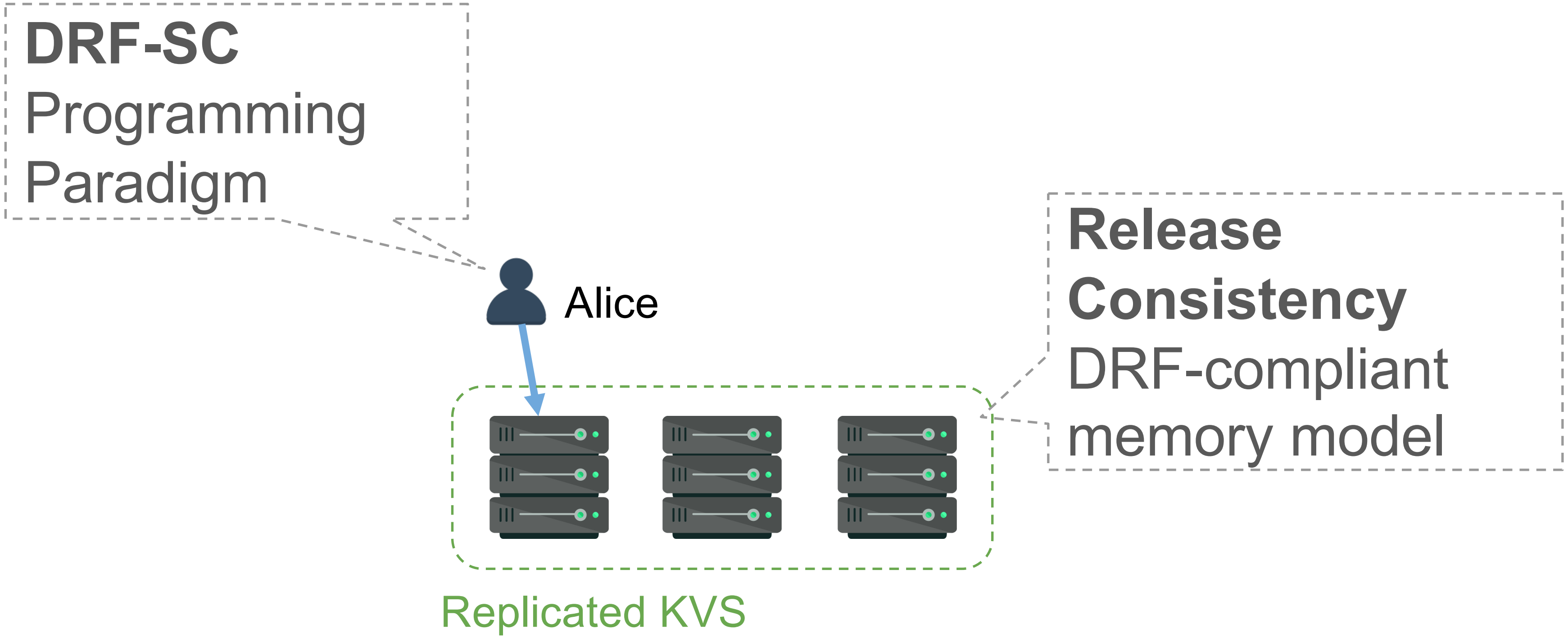
Shared Memory World

Sweet-spot in the Performance-vs-Consistency?

Data-race-free!

Yes, label synchronisation!

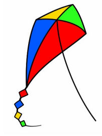
Can we do the same for KVSes?



DRF-SC
Programming Paradigm

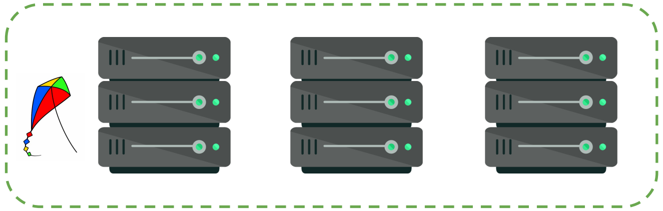
Release Consistency
DRF-compliant memory model

Kite



A Replicated KVS with

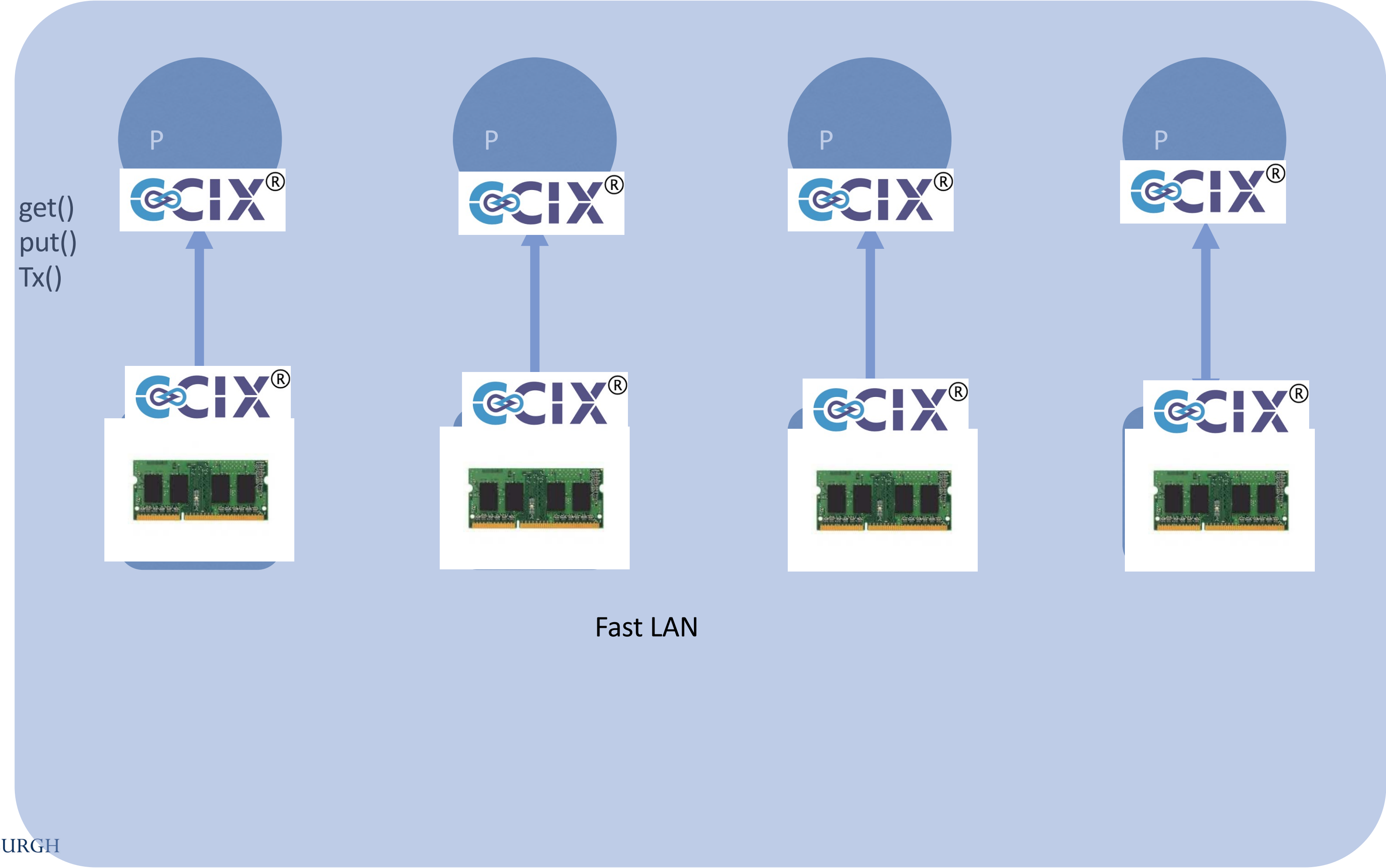
- Release Consistency
- High Availability



Kite Replicated KVS

*PPoPP'20, Best paper nomination

Rethinking Datacentre Memory



Revolutionizing Mobile/Cloud via Coherence

- Raise abstraction of coherence protocol design and automate
  - Concurrency
  - Hierarchy
  - Heterogeneity

- Datacentre coherence a great opportunity but needs new family of high-performance fault-tolerant coherence protocols.