Co-designing *reliability* and *performance* for datacenter memory

Adarsh Patil

Doctoral Examination

30th May 2023

Advisor:
Vijay Nagarajan (UoE)
Co-examiners:
Vilas Sridharan (AMD)
Antonio Barbalace (UoE)

THE UNIVERSITY *of* EDINBURGH
informatics

About me: My journey so far…

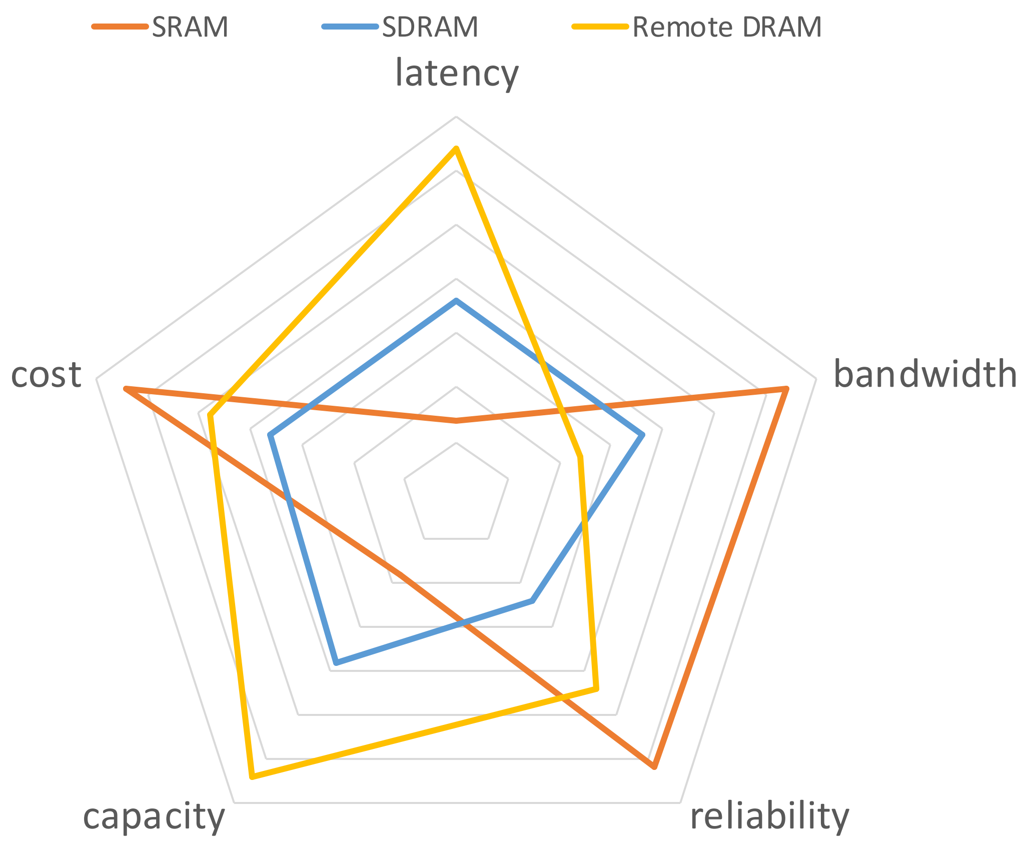
(2012–14) Datacenter Infra Team – Virtualization & Linux Engineering
solutions architect: platform benchmarking, performance analysis

(2014–17) Masters by Research – HAShCache [TACO ‘18], TLB reach [arXiv]
memory architecture: DRAM cache, heterogeneous SoCs, virtual mem [Advisor: Prof. R. Govindarajan]

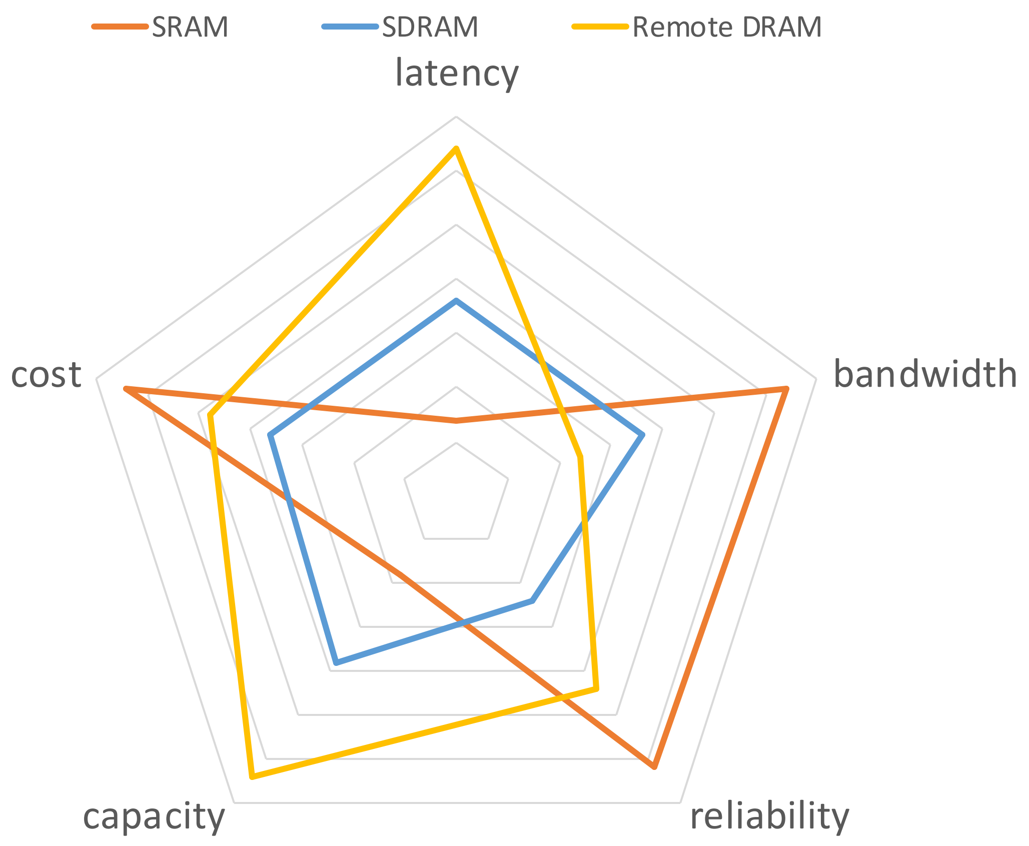
(2017–19) Research Scientist – HPC ecosystem and applications team
application understanding: s/w optimization, h/w architecture for next gen

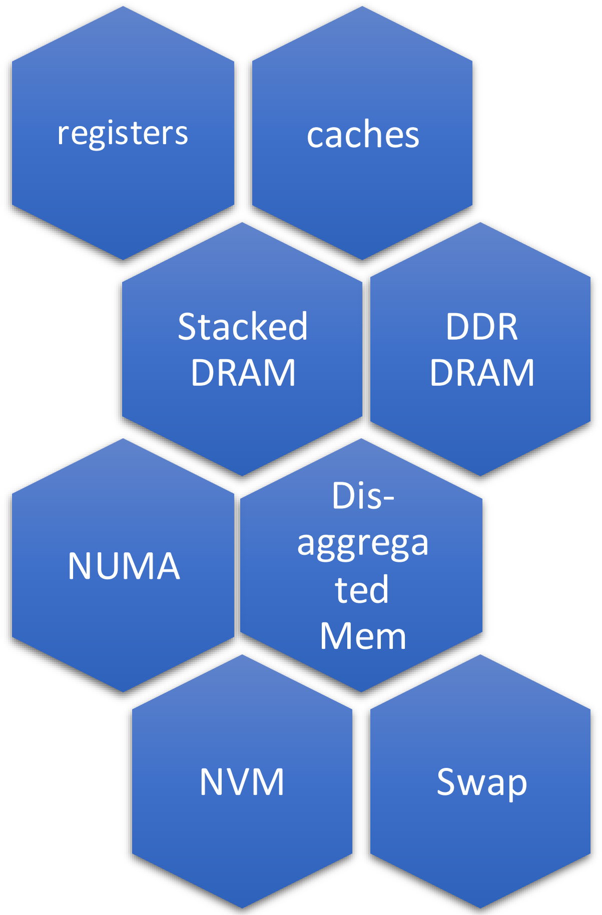
(2019–now) – Co-designing reliability and performance for the datacenter
holistic approach: integrating hardware + application

Memory – a perpetual conundrum!

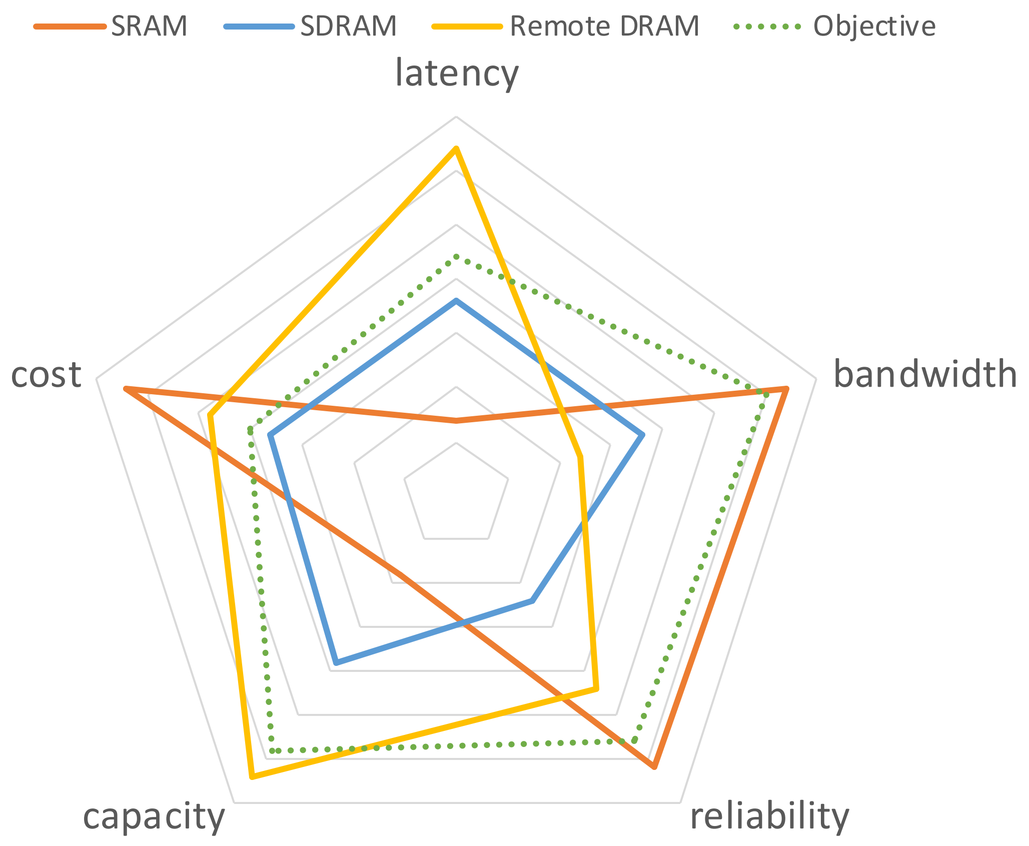


Memory – a perpetual conundrum!

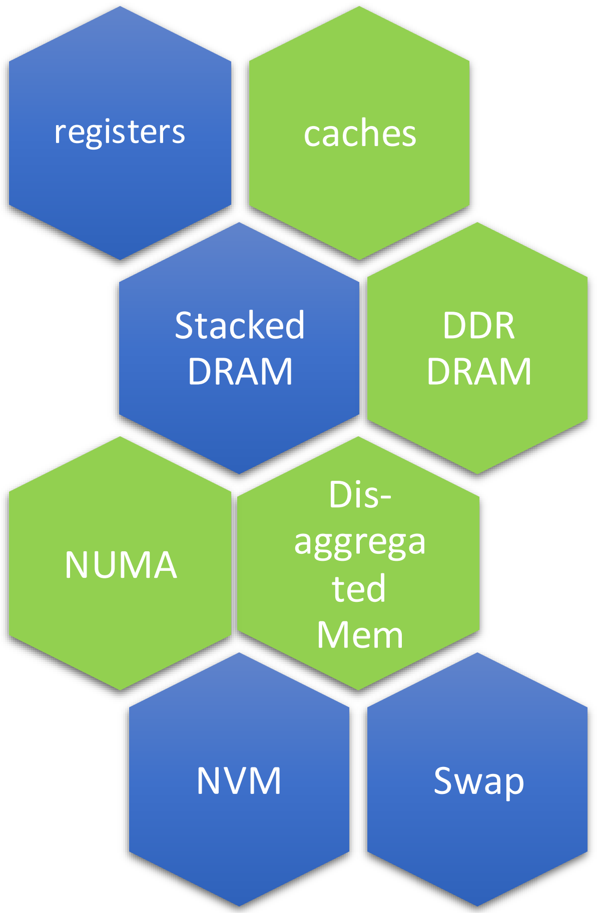




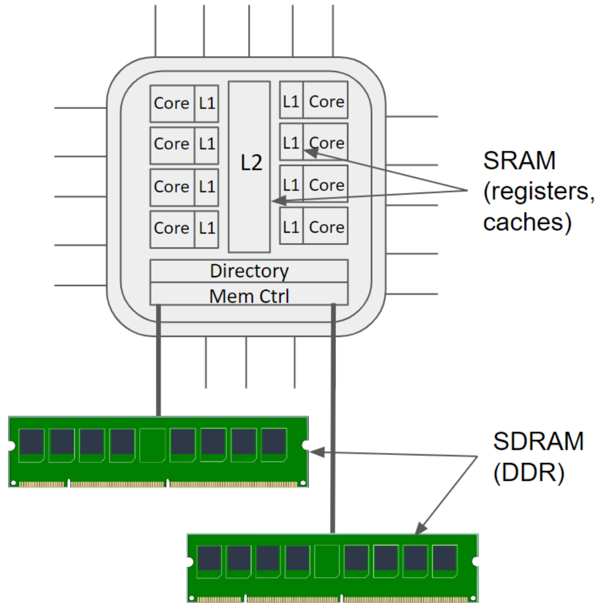
Thesis objective



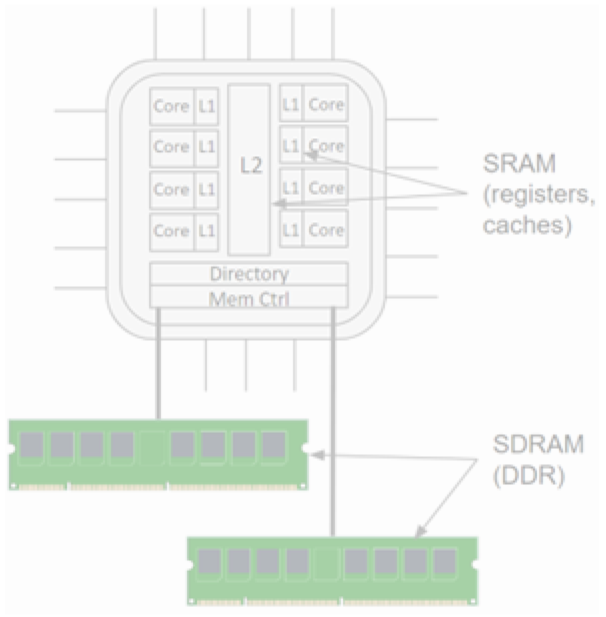
High-performance, high-reliability memory



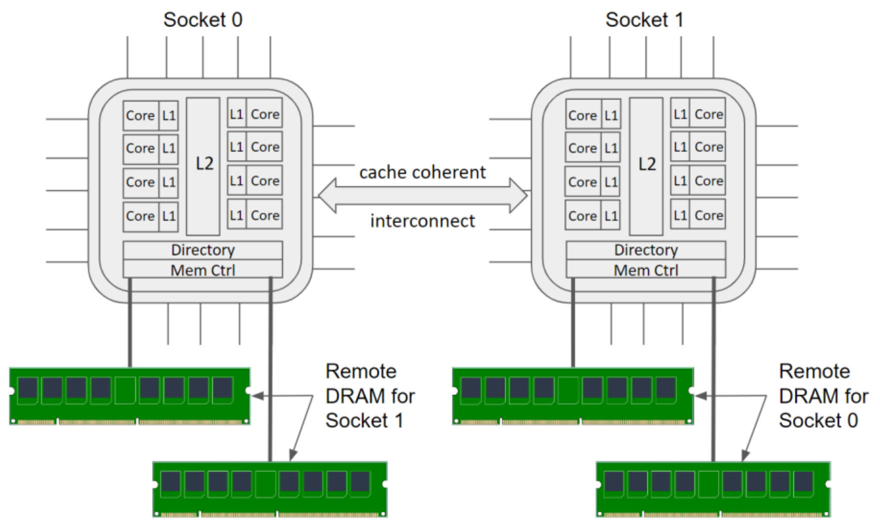
Datacenter memory



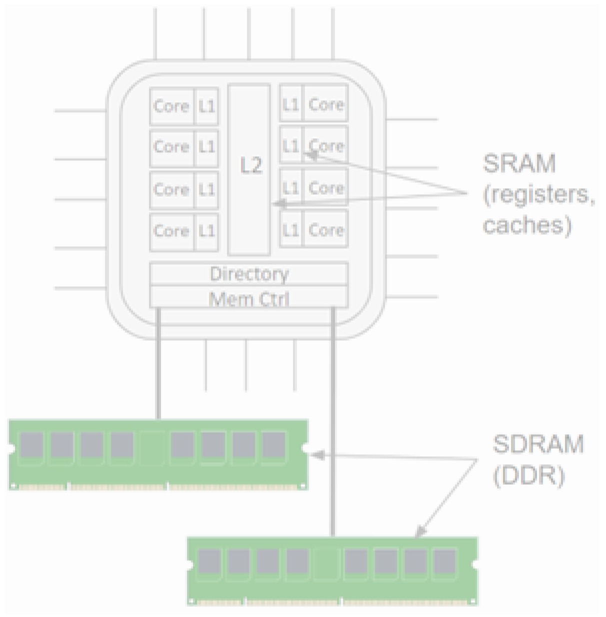
Datacenter memory



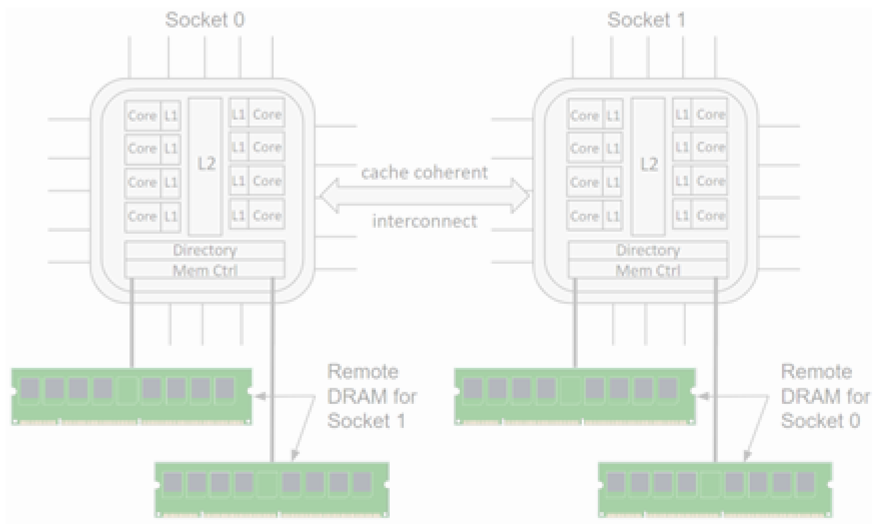
NUMA

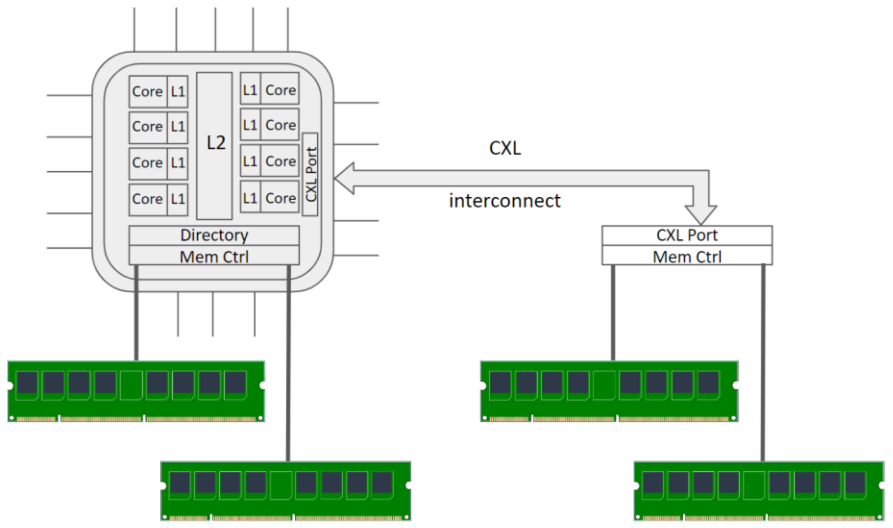


Datacenter memory



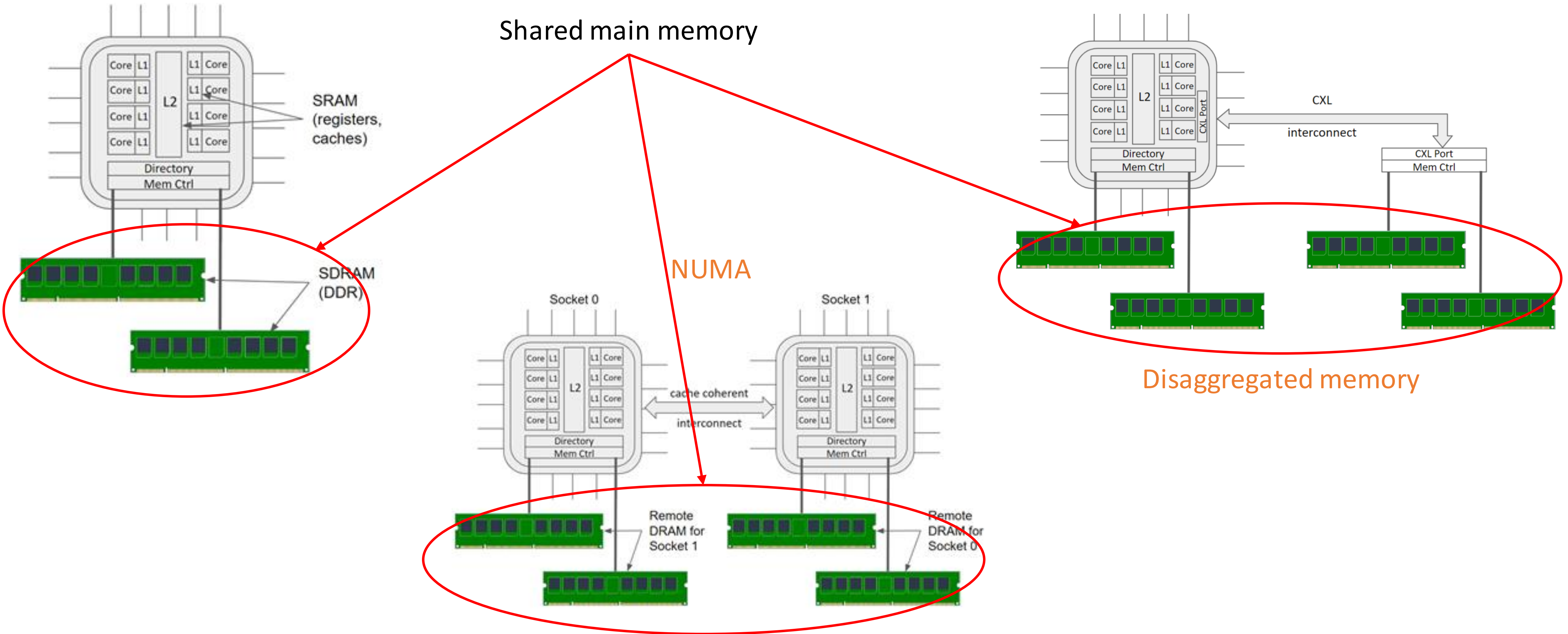
NUMA



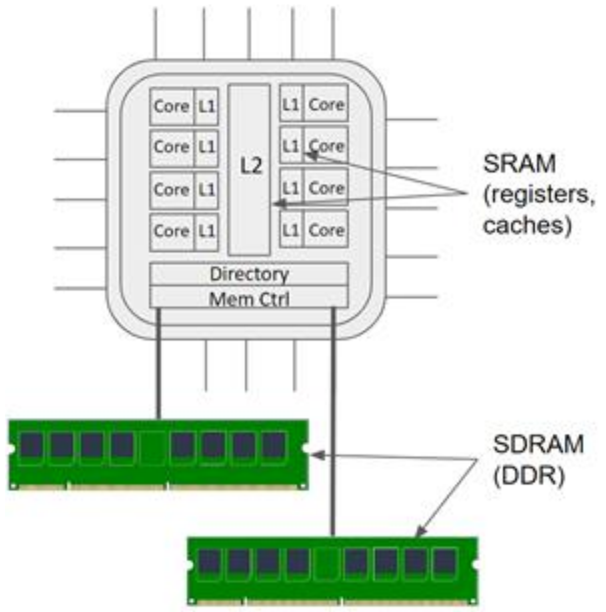


Disaggregated memory

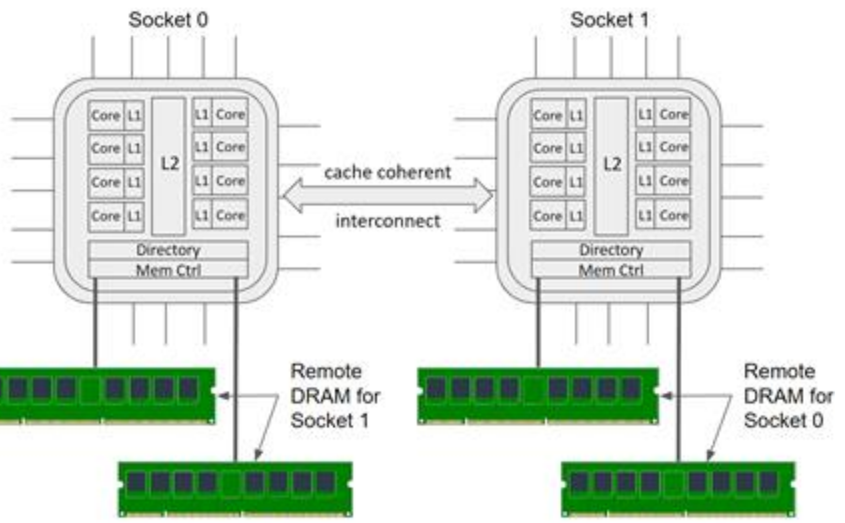
Main memory is comprised of DRAMs

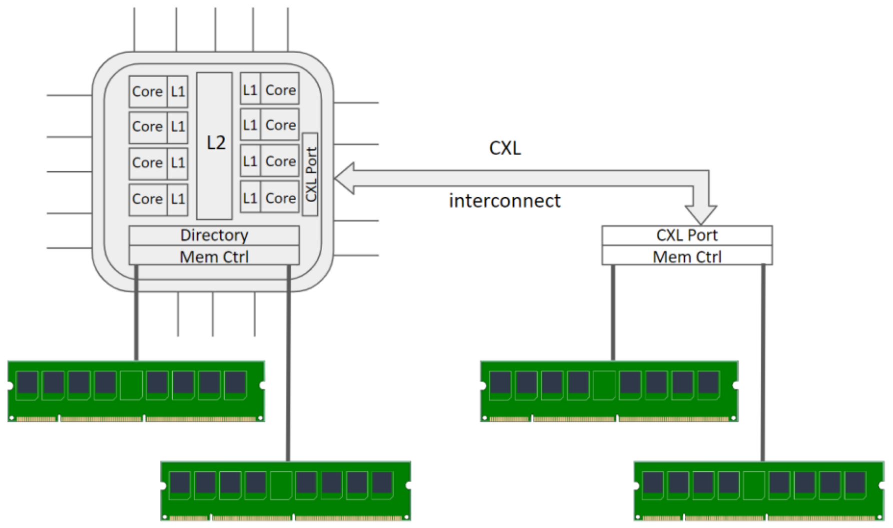


Pervasive coherence protocols



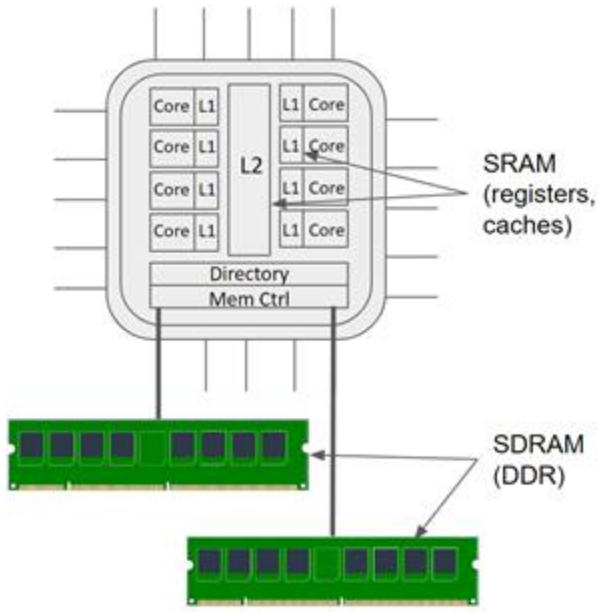
NUMA





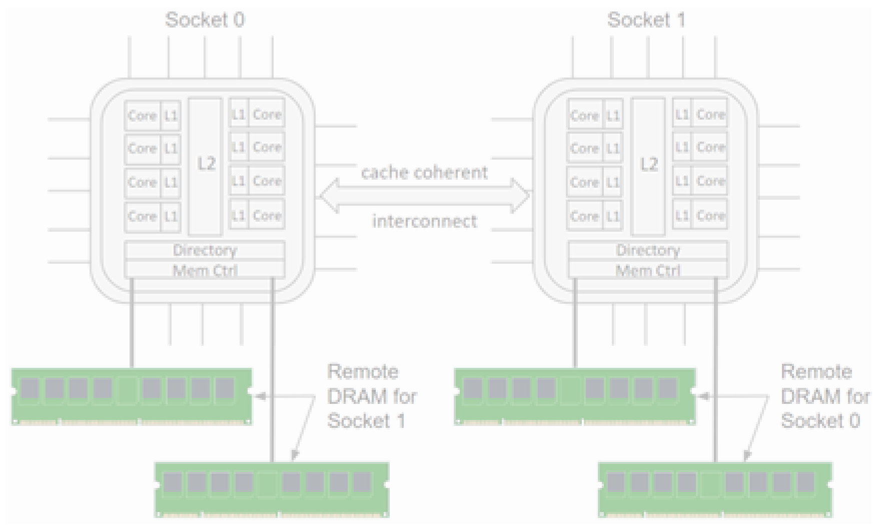
Disaggregated memory

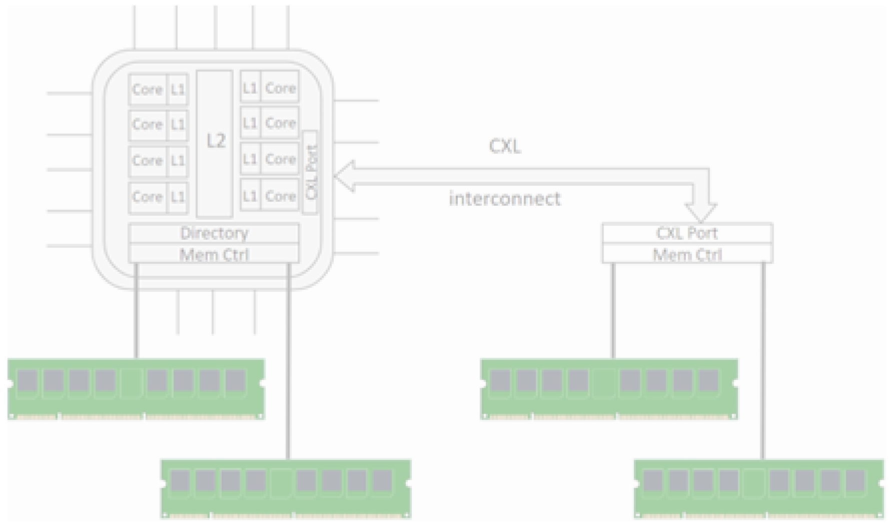
Pervasive coherence protocols



Intra-processor coherence

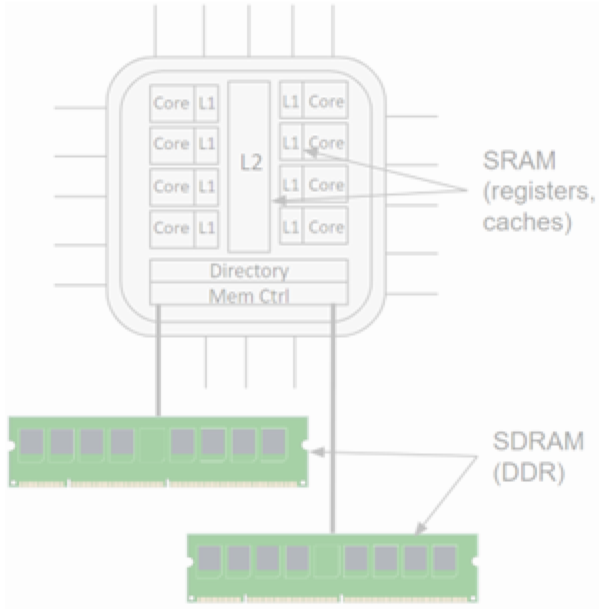
NUMA





Disaggregated memory

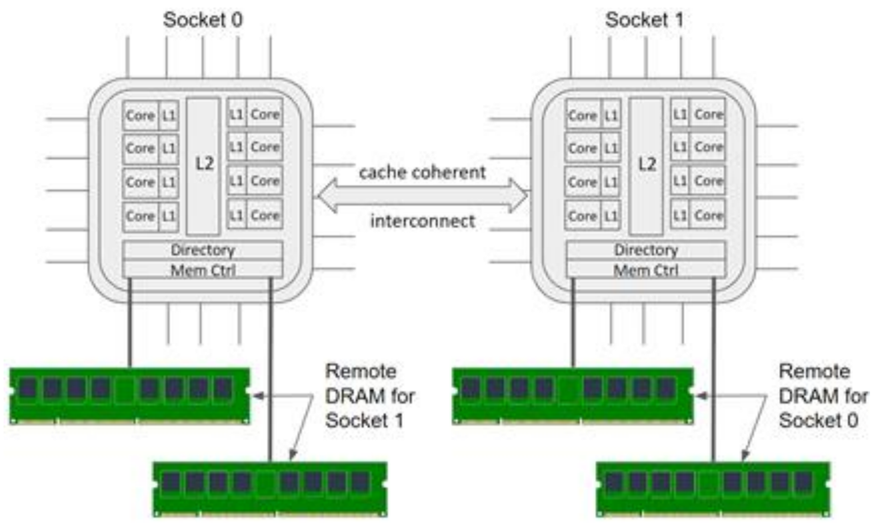
Pervasive coherence protocols

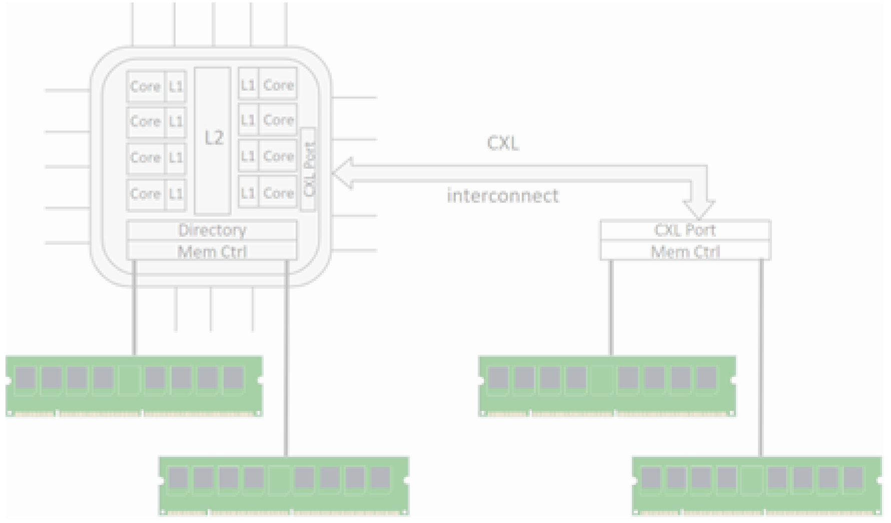


Intra-processor coherence

Inter-processor coherence

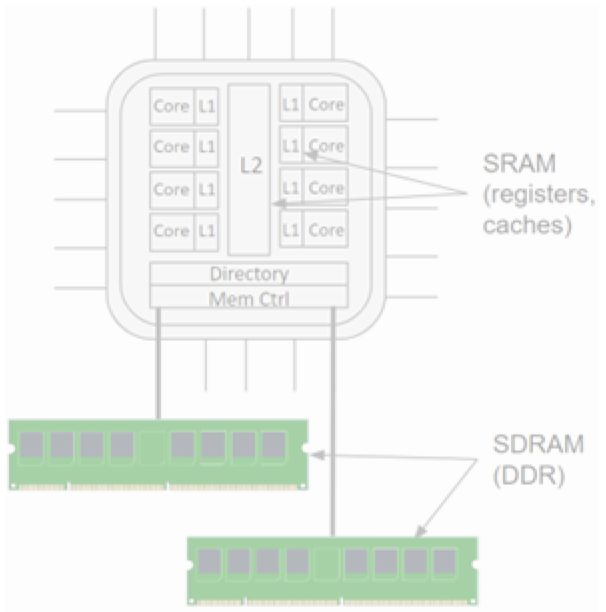
NUMA





Disaggregated memory

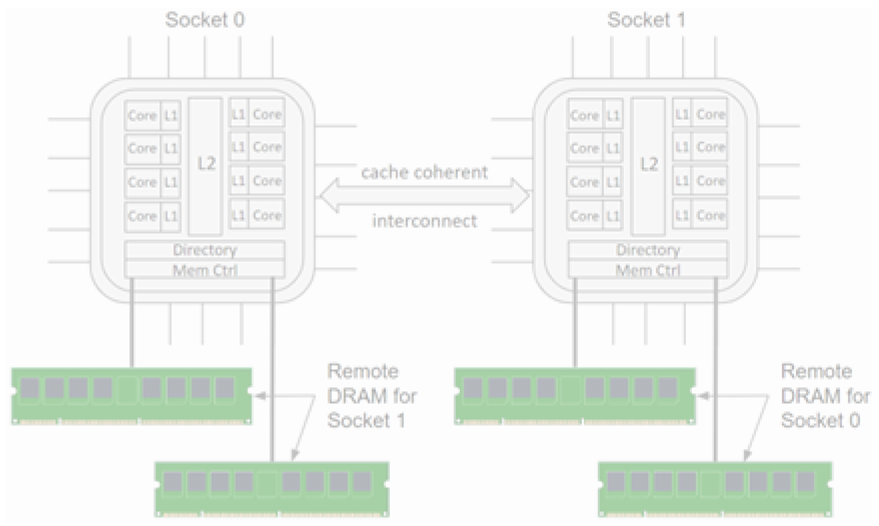
Pervasive coherence protocols

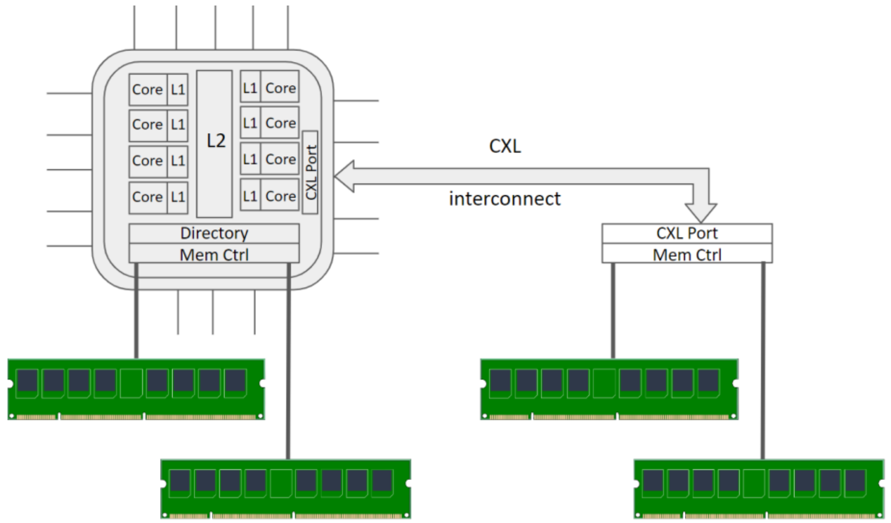


Intra-processor coherence

Inter-processor coherence

NUMA





Disaggregated memory

Inter-server coherence

Thesis insights and contributions

Dvé
Memory
Replication

Employ coherence protocols to improve reliability and performance of DRAM memory **[ISCA ‘21]**

Thesis insights and contributions

Dvé
Memory Replication

Employ coherence protocols to improve reliability and performance of DRAM memory **[ISCA ‘21]**

Āpta
Reliable CXL.mem

Harden the coherence protocols against common modes of failures **[DSN ‘23]**

Thesis insights and contributions

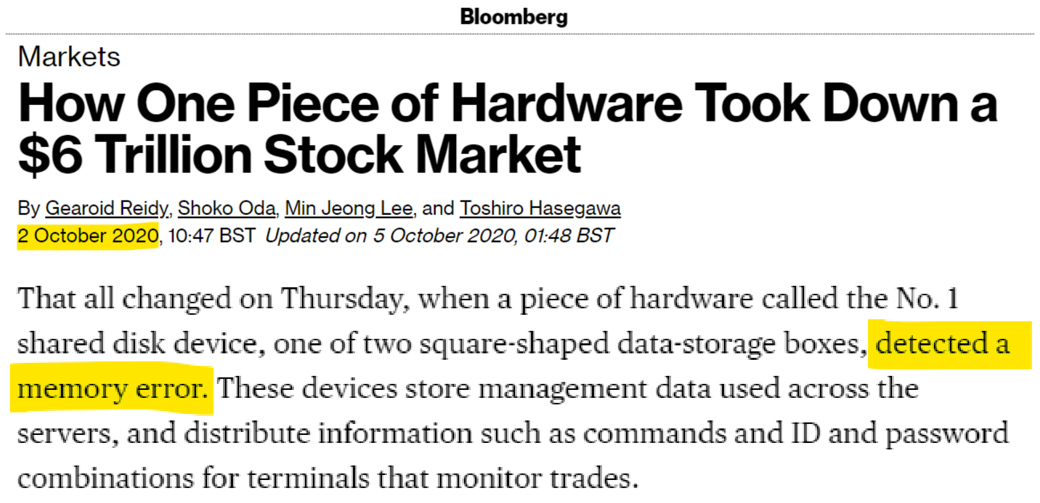
Dvé

Memory Replication

Improving DRAM Reliability and Performance

1

The problem: Increasing DRAM Faults

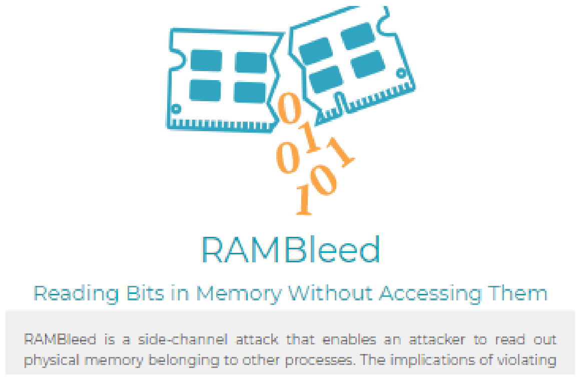

Bloomberg

Markets

How One Piece of Hardware Took Down a $6 Trillion Stock Market

By Gearoid Reidy, Shoko Oda, Min Jeong Lee, and Toshiro Hasegawa
2 October 2020, 10:47 BST *Updated on 5 October 2020, 01:48 BST*

That all changed on Thursday, when a piece of hardware called the No. 1 shared disk device, one of two square-shaped data-storage boxes, detected a memory error. These devices store management data used across the servers, and distribute information such as commands and ID and password combinations for terminals that monitor trades.


RAMBleed

Reading Bits in Memory Without Accessing Them

RAMBleed is a side-channel attack that enables an attacker to read out physical memory belonging to other processes. The implications of violating

VUSec

ECCPLOIT: ECC MEMORY VULNERABLE TO ROWHAMMER ATTACKS AFTER ALL

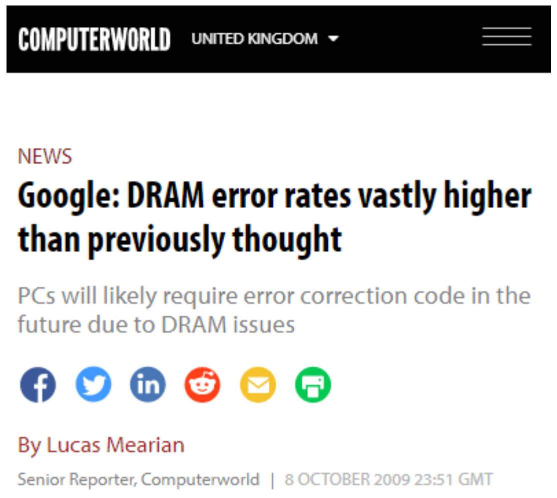
Where many people thought that high-end servers were safe from the (unpatchable) Rowhammer bitflip

IEEE SPECTRUM

Hardware

DRAM's Damning Defects—and How They Cripple Computers

By **Ioan Stefanovici, Andy Hwang and Bianca Schroeder**
Posted 23 Nov 2015 | 16:00 GMT

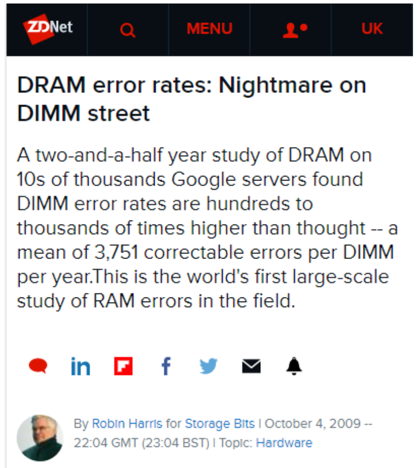

COMPUTERWORLD UNITED KINGDOM

NEWS

Google: DRAM error rates vastly higher than previously thought

PCs will likely require error correction code in the future due to DRAM issues

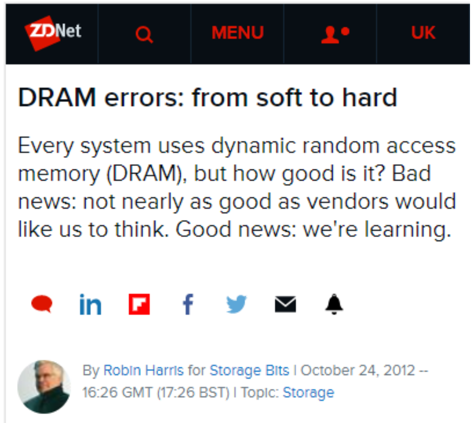
By Lucas Mearian
Senior Reporter, Computerworld | 8 OCTOBER 2009 23:51 GMT


ZDNet MENU UK

DRAM error rates: Nightmare on DIMM street

A two-and-a-half year study of DRAM on 10s of thousands Google servers found DIMM error rates are hundreds to thousands of times higher than thought -- a mean of 3,751 correctable errors per DIMM per year.This is the world's first large-scale study of RAM errors in the field.

By Robin Harris for Storage Bits | October 4, 2009 -- 22:04 GMT (23:04 BST) | Topic: Hardware

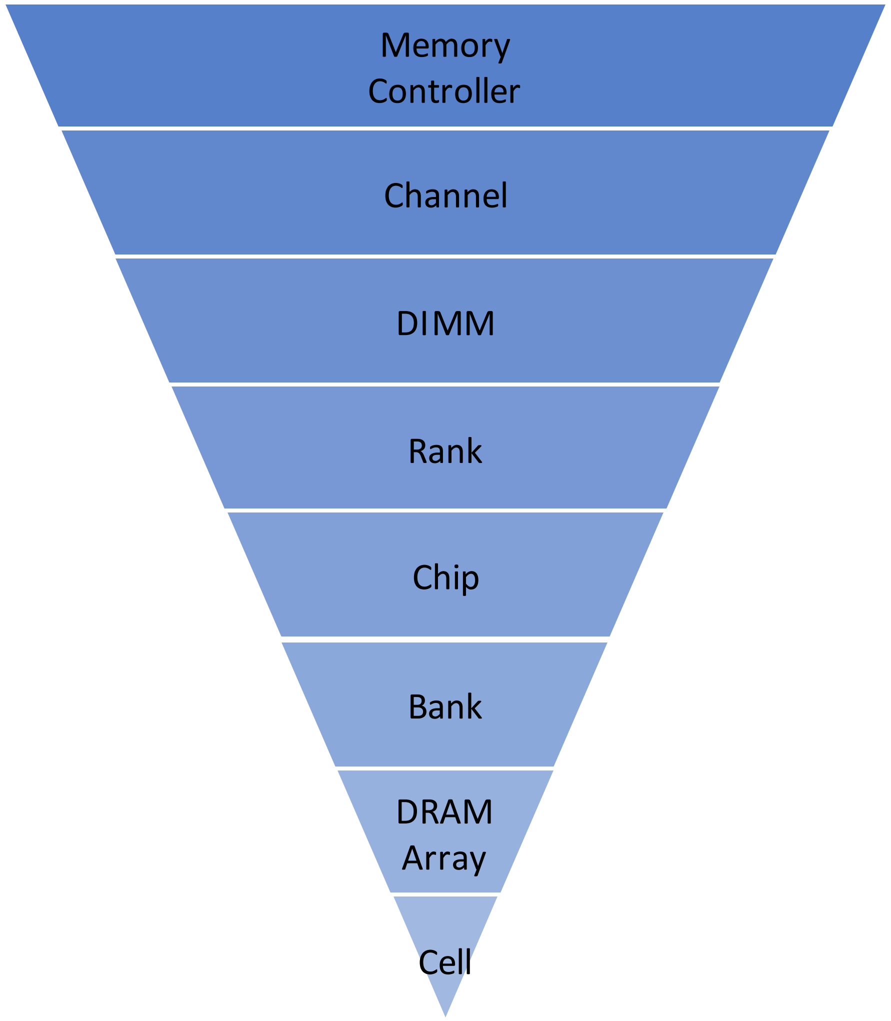

ZDNet MENU UK

DRAM errors: from soft to hard

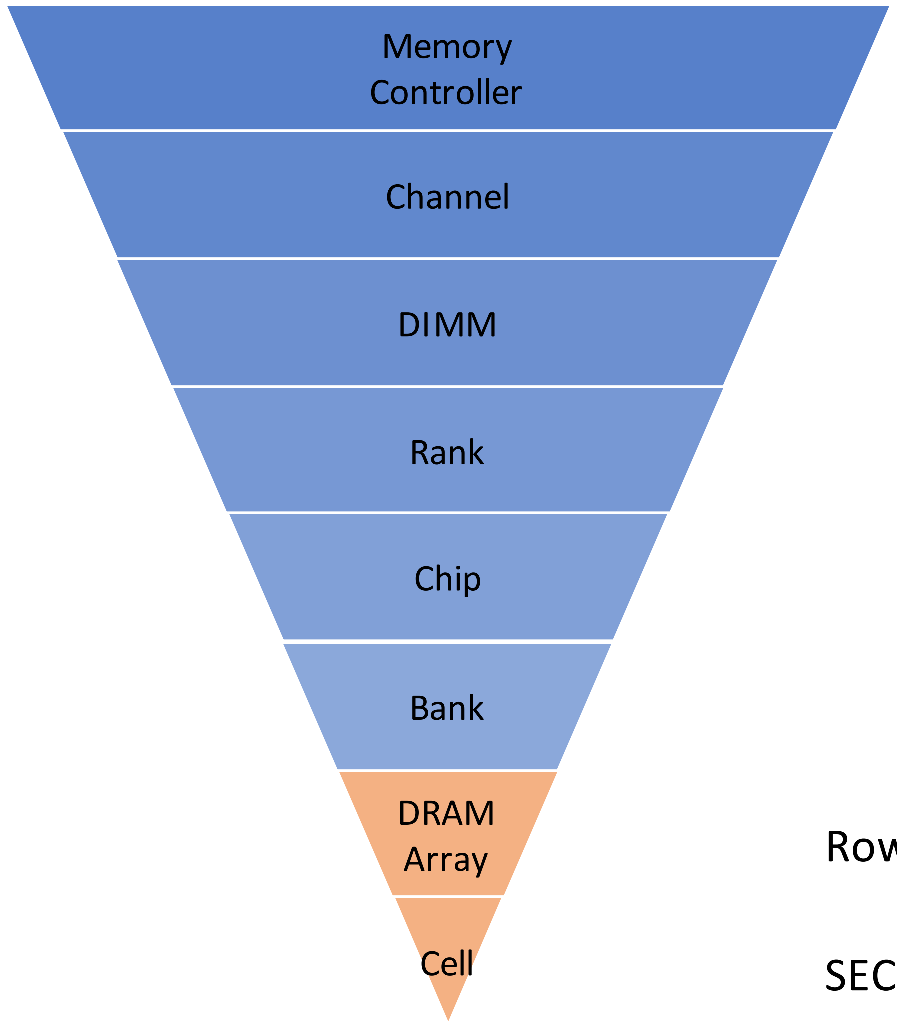
Every system uses dynamic random access memory (DRAM), but how good is it? Bad news: not nearly as good as vendors would like us to think. Good news: we're learning.

By Robin Harris for Storage Bits | October 24, 2012 -- 16:26 GMT (17:26 BST) | Topic: Storage

Progression of Reliability Mechanisms



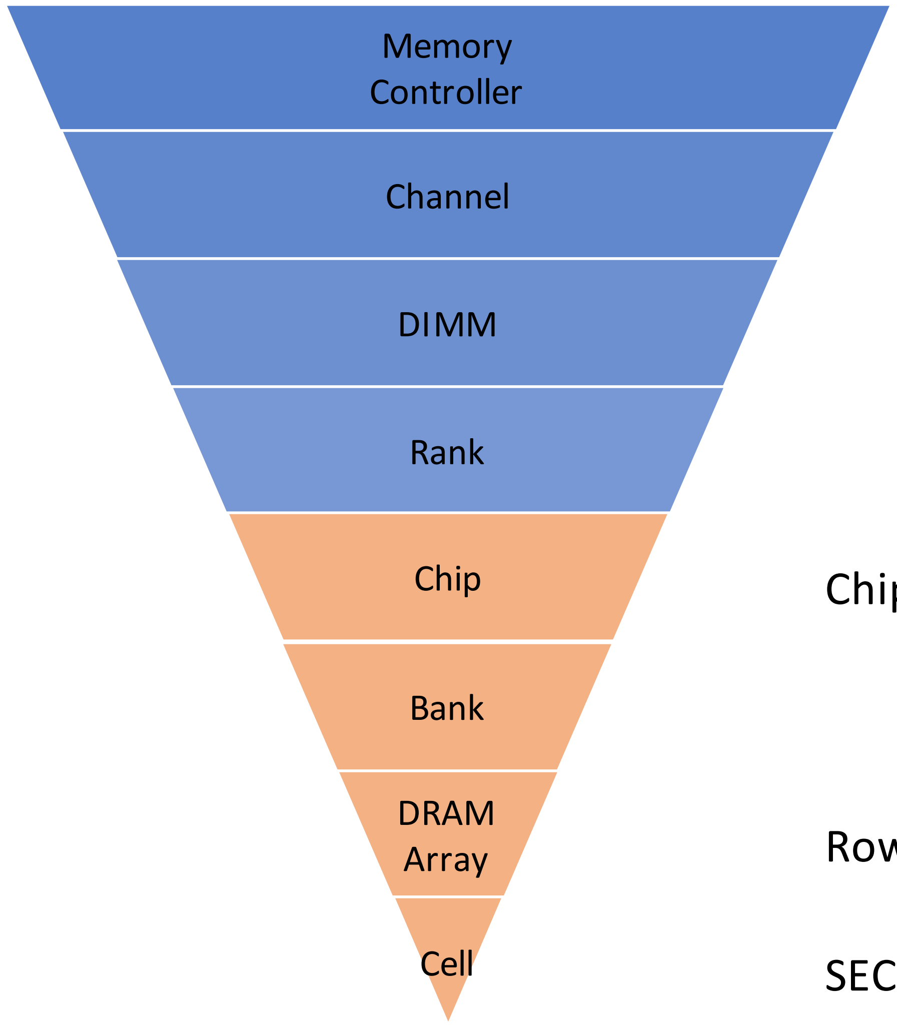
Progression of Reliability Mechanisms



Row/column sparing

SECDED ECC, in-DRAM ECC

Progression of Reliability Mechanisms

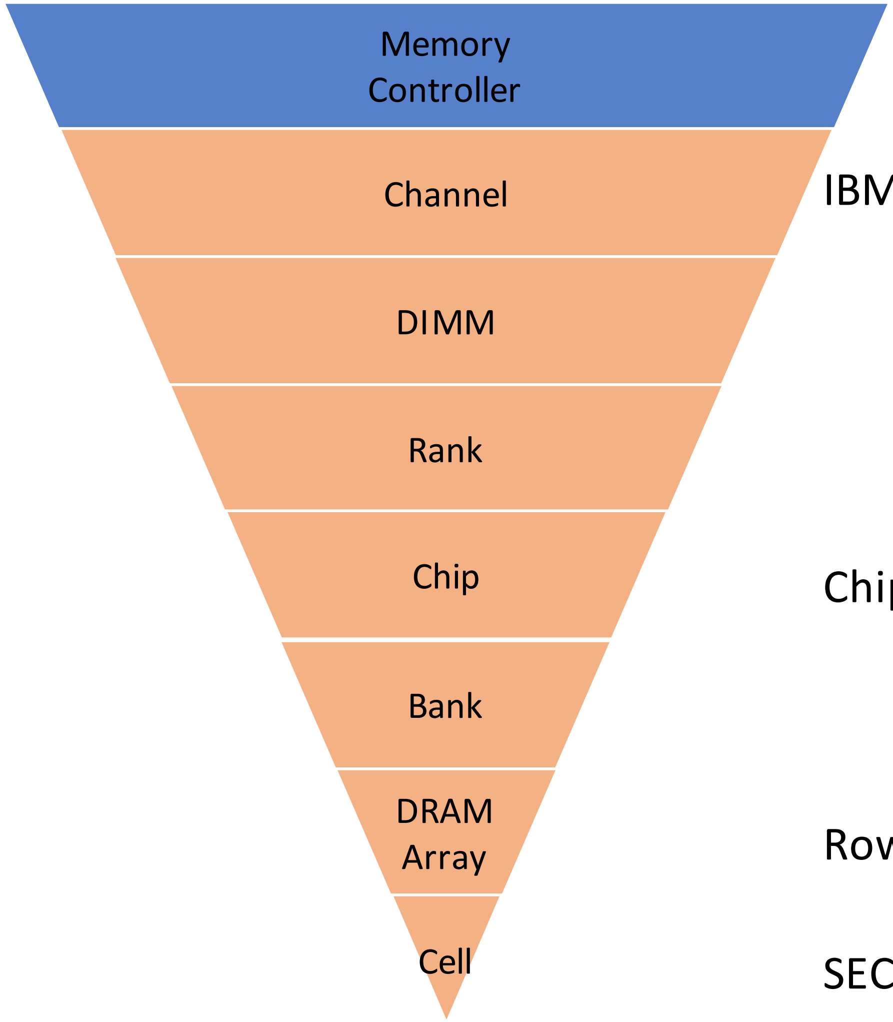


Chipkill ECC / Multi-tier ECC

Row/column sparing

SECDED ECC, in-DRAM ECC

Progression of Reliability Mechanisms



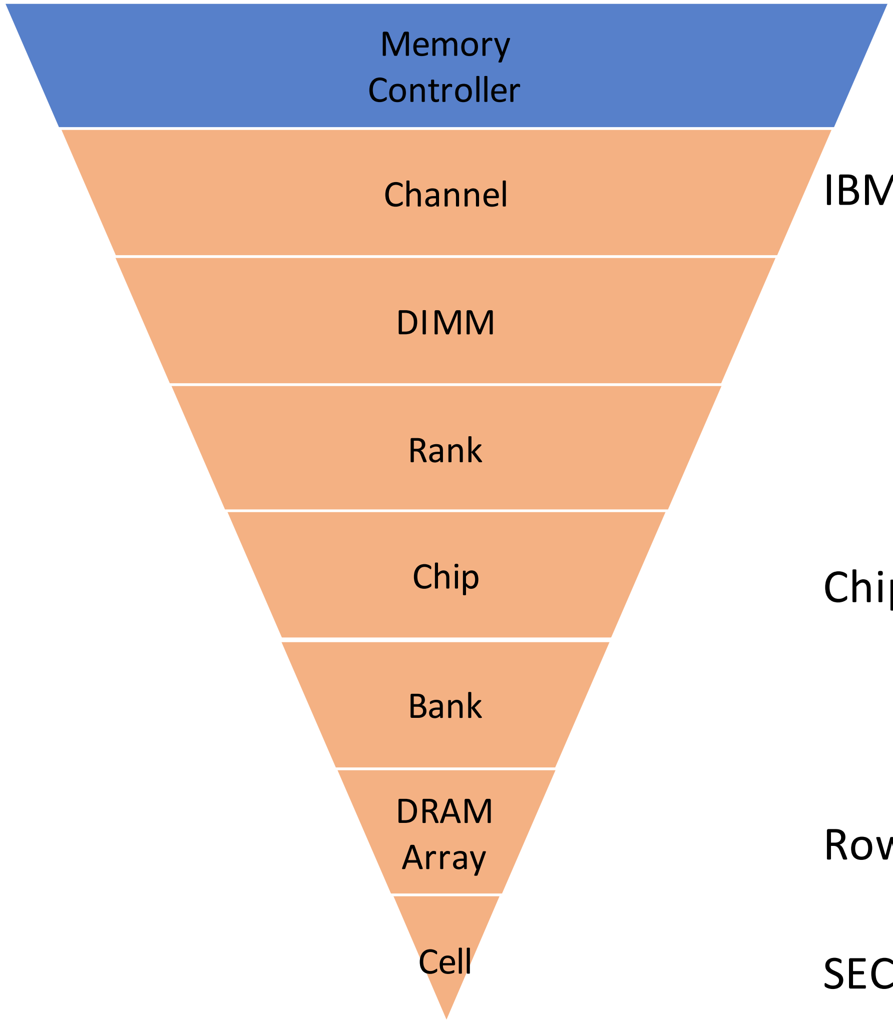
IBM RAIM, Intel Memory Mirroring

Chipkill ECC / Multi-tier ECC

Row/column sparing

SECDED ECC, in-DRAM ECC

Progression of Reliability Mechanisms



IBM RAIM, Intel Memory Mirroring

Chipkill ECC / Multi-tier ECC

Row/column sparing

SECDED ECC, in-DRAM ECC

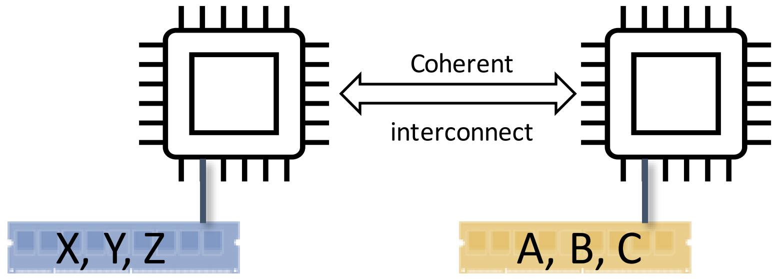
Performance overheads

Replication for Reliability

Dvé insights

- ❑ Full data replica (not ECC code)
- ❑ Keep Replicas as far apart and disjoint as possible
- ❑ Tolerate errors arising from anywhere in the memory path

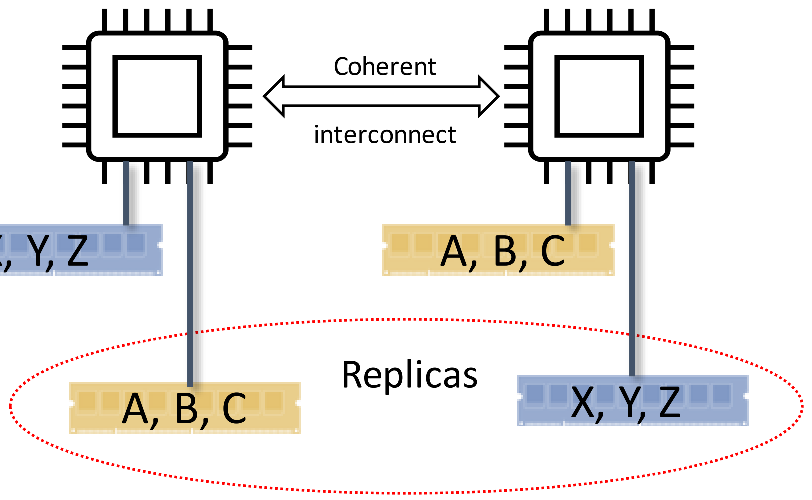
Replication for Reliability



Dvé insights

- ❑ Full data replica (not ECC code)
- ❑ Keep Replicas as far apart and disjoint as possible
- ❑ Tolerate errors arising from anywhere in the memory path

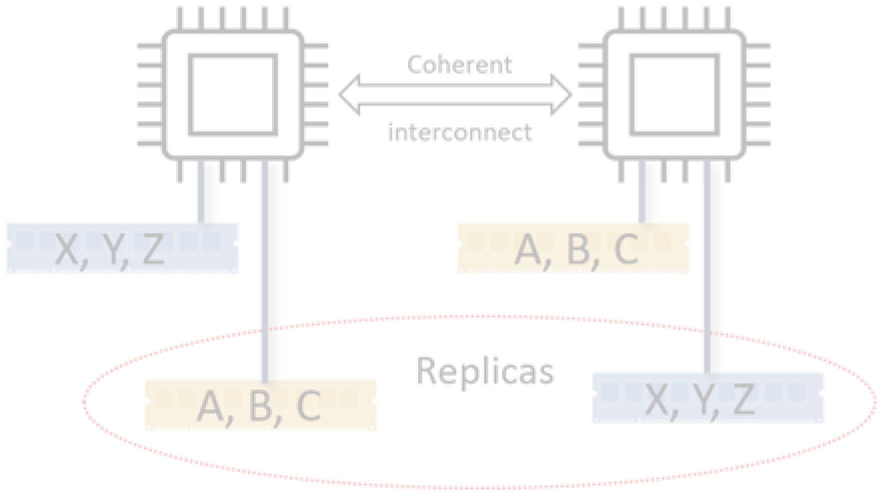
Replication for Reliability



Dvé insights

- ❑ Full data replica (not ECC code)
- ❑ Keep Replicas as far apart and disjoint as possible
- ❑ Tolerate errors arising from anywhere in the memory path

Replication for Reliability

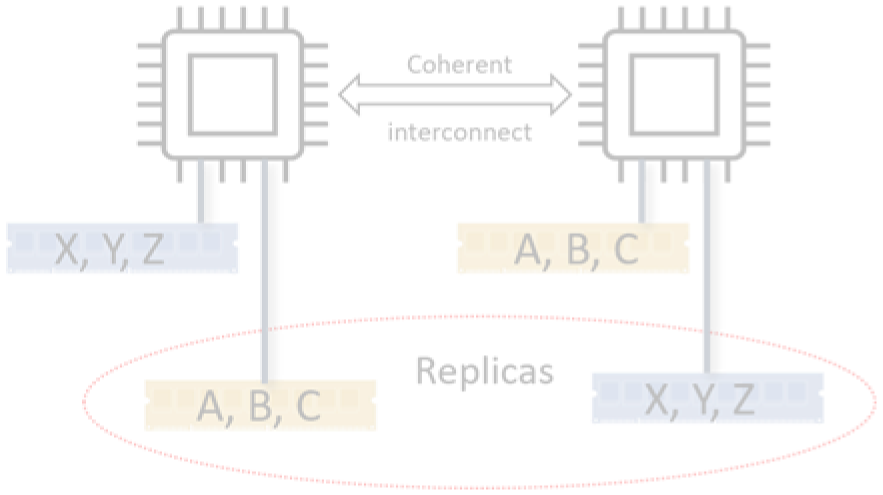


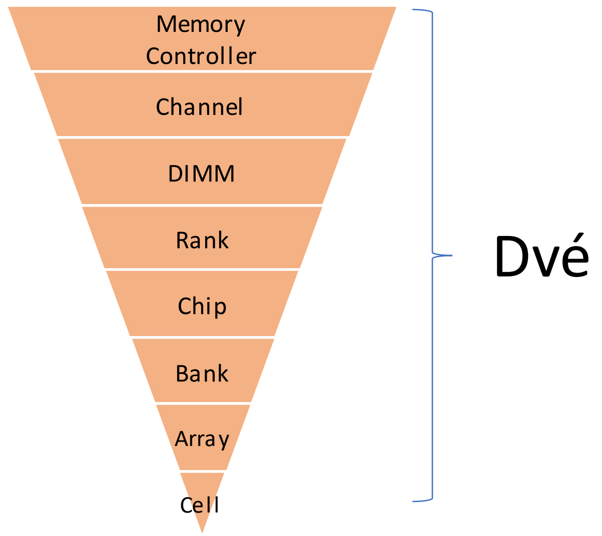
Dvé insights
- ❑ Full data replica (not ECC code)
- ❑ Keep Replicas as far apart and disjoint as possible
- ❑ Tolerate errors arising from anywhere in the memory path

For Detection
- ❑ Existing ECC, CRC, Parity
- ❑ Strong detection-only code
- ❑ Other diagnostic capabilities

For Correction
- ❑ Rely on replica

Replication for Reliability





Dvé insights
- ❑ Full data replica (not ECC code)
- ❑ Keep Replicas as far apart and disjoint as possible
- ❑ Tolerate errors arising from anywhere in the memory path

For Detection
- ❑ Existing ECC, CRC, Parity
- ❑ Strong detection-only code
- ❑ Other diagnostic capabilities

For Correction
- ❑ Rely on replica

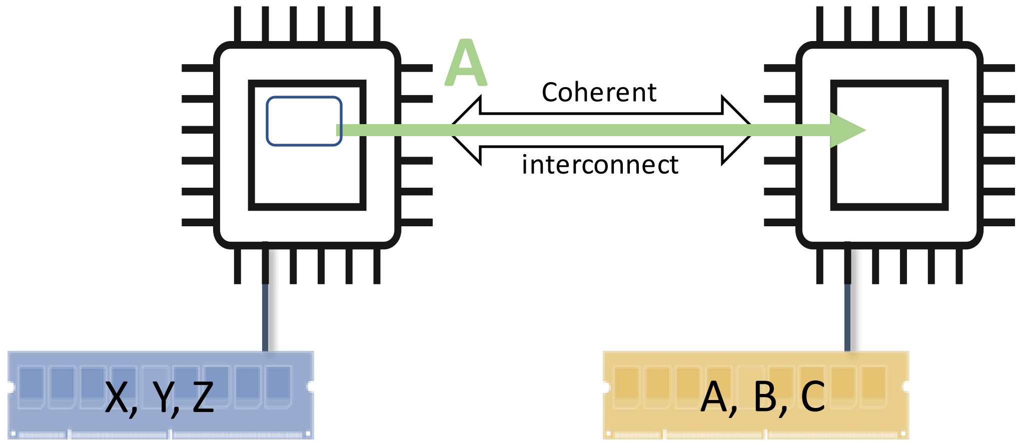
Coherent Replication for Performance

Dvé insights

- ❑ Use replica to improve performance

Dvé
Memory
Replication

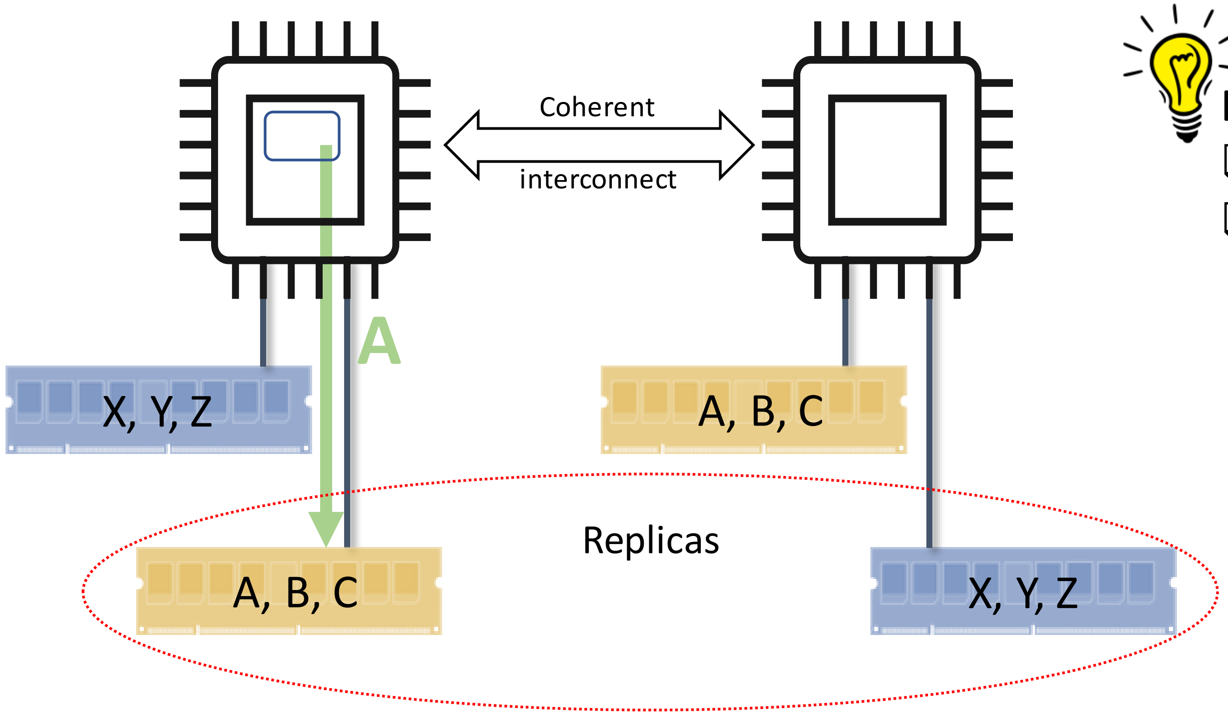
Coherent Replication



Dvé insights

- ❑ Use replica to improve performance

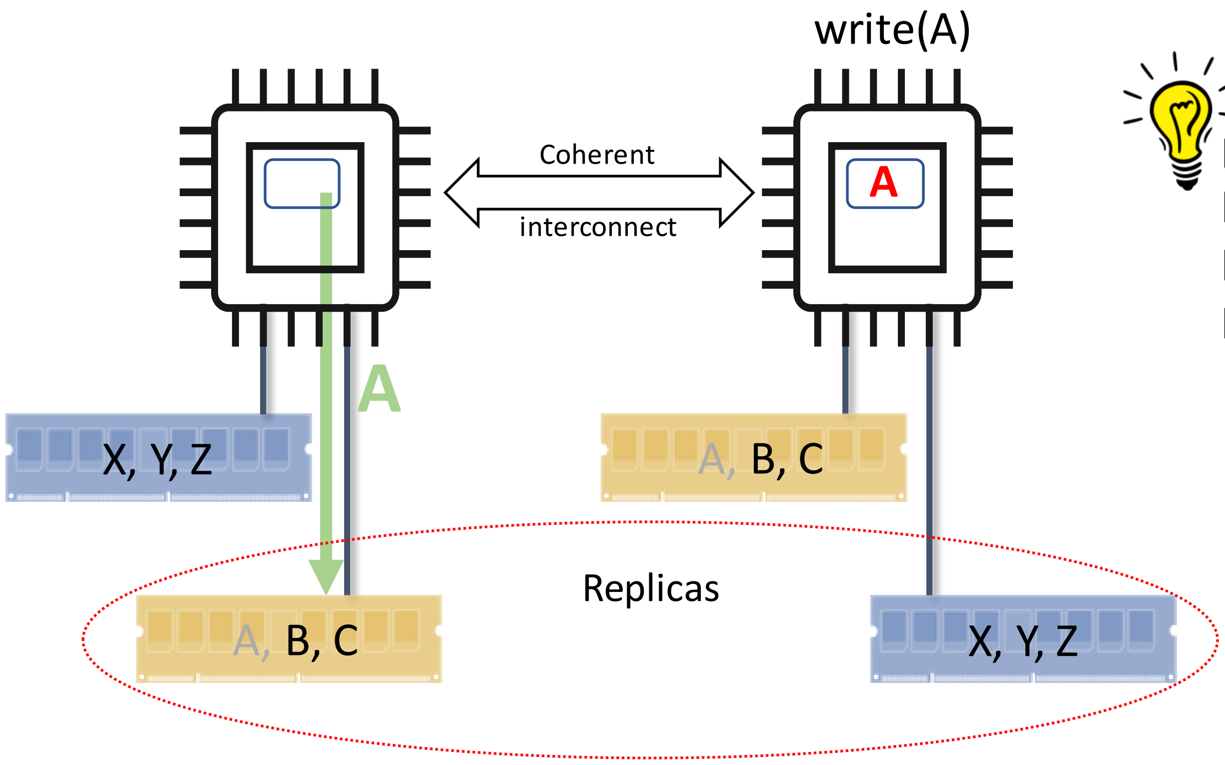
Coherent Replication



Dvé insights

- ❑ Use replica to improve performance
- ❑ Route memory requests to nearest replica

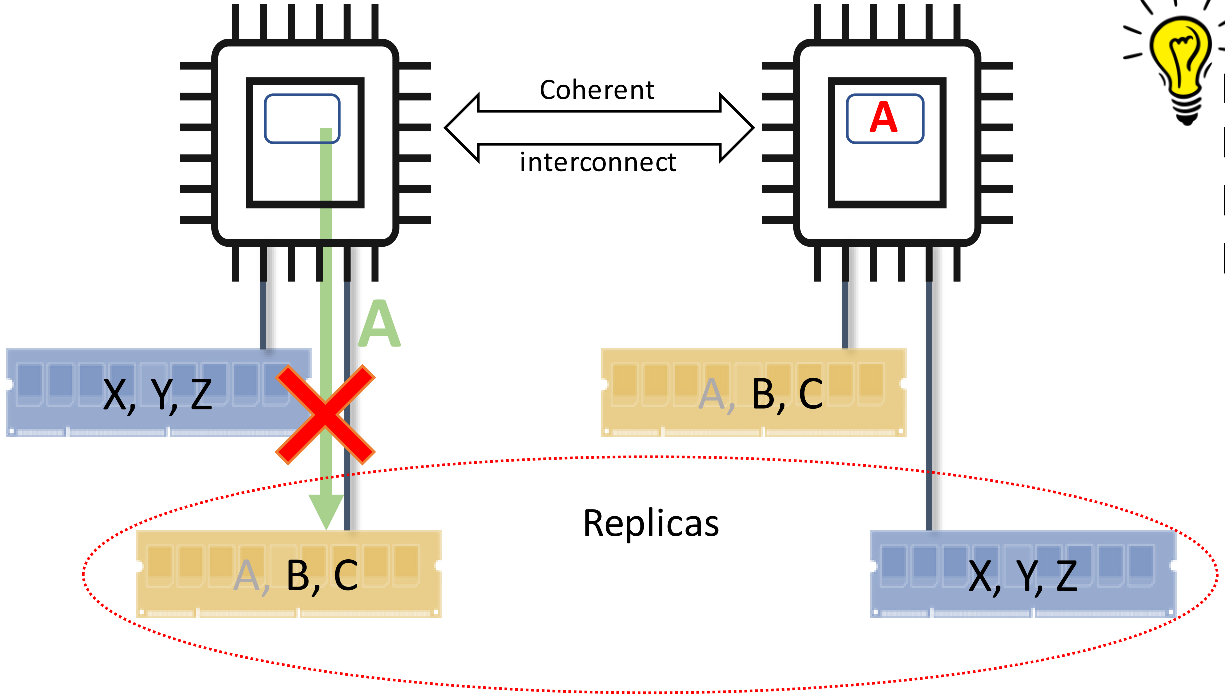
Coherent Replication



Dvé insights

- ❑ Use replica to improve performance
- ❑ Route memory requests to nearest replica
- ❑ Ensure safe access to replica

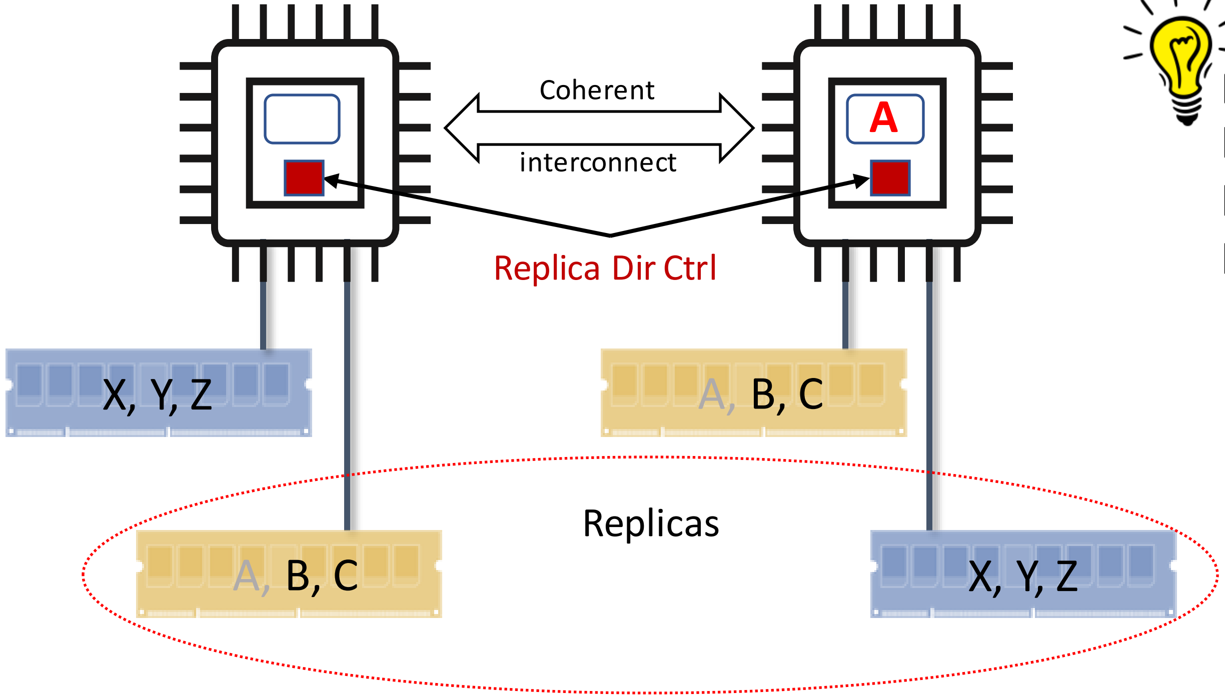
Coherent Replication



Dvé insights

- ❑ Use replica to improve performance
- ❑ Route memory requests to nearest replica
- ❑ Ensure safe access to replica

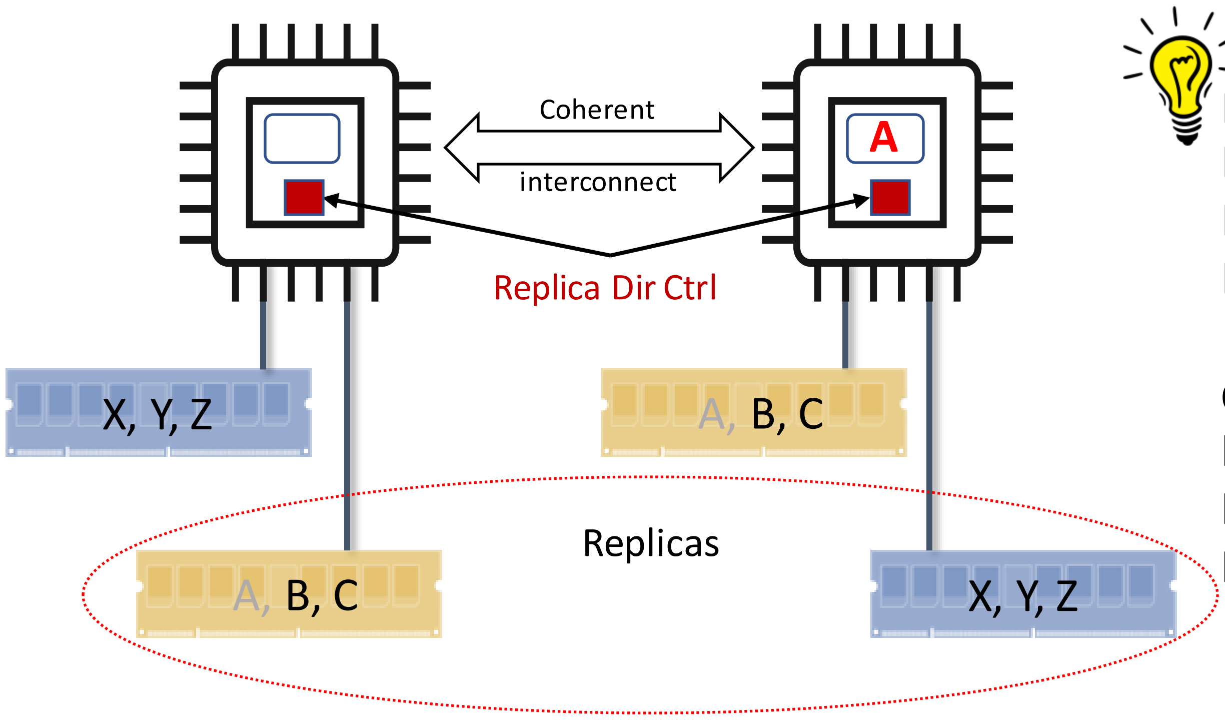
Coherent Replication



Dvé insights

- ❑ Use replica to improve performance
- ❑ Route memory requests to nearest replica
- ❑ Ensure safe access to replica

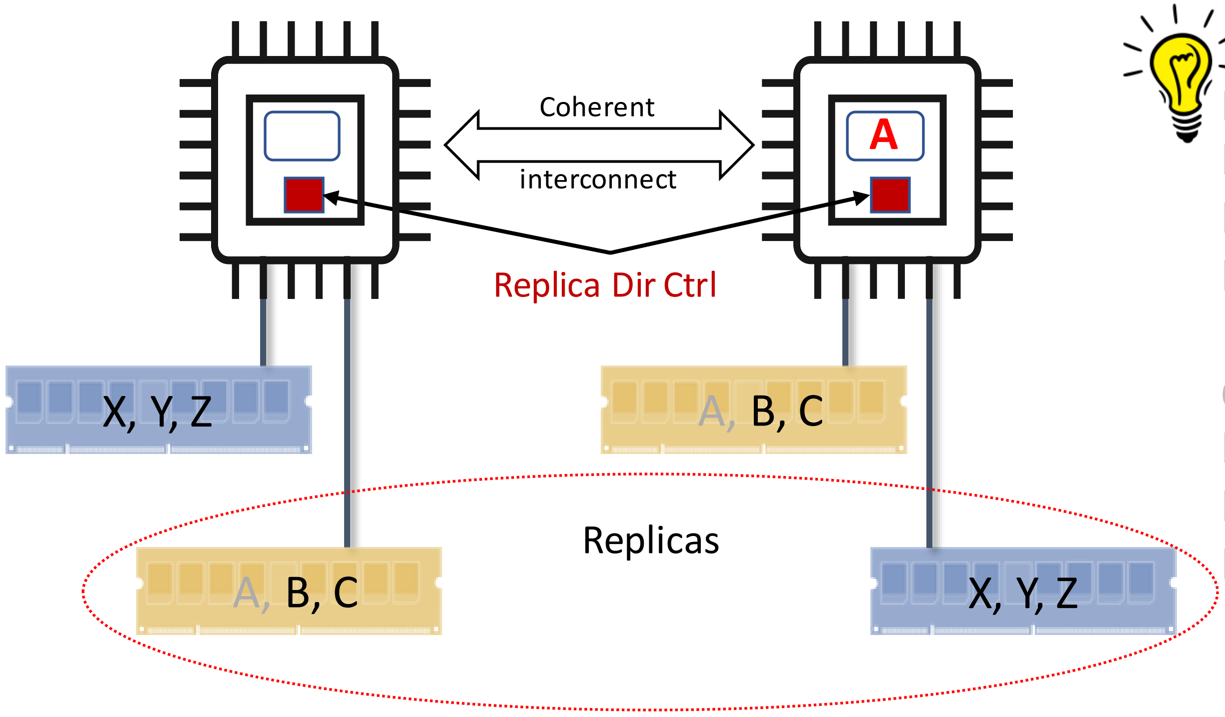
Coherent Replication



Dvé insights
- ❑ Use replica to improve performance
- ❑ Route memory requests to nearest replica
- ❑ Ensure safe access to replica

Coherent Replication
- ❑ Builds on existing cache coherence protocols
- ❑ maintain the replicas in sync (for reliability)
- ❑ provide coherent access to both replicas during fault-free operation (for performance)

Coherent Replication



Dvé insights
- ❑ Use replica to improve performance
- ❑ Route memory requests to nearest replica
- ❑ Ensure safe access to replica

Coherent Replication
- ❑ Builds on existing cache coherence protocols
- ❑ maintain the replicas in sync (for reliability)
- ❑ provide coherent access to both replicas during fault-free operation (for performance)

Mechanisms
- ❑ Allow-based
- ❑ Deny-based

Dvé Memory Replication

Capacity overheads?

Capacity overheads?

Dvé insights

- ❑ Utilize idle memory

Skewed memory utilization

- ❑ 50% of the memory is idle in 90% of the servers
- ❑ Provisioning for peak

Capacity overheads?

Dvé insights

- ❑ Utilize idle memory
- ❑ Overheads applicable only as and when demanded by the application

Skewed memory utilization

- ❑ 50% of the memory is idle 90% of the servers
- ❑ Provisioning for peak

Interface to allocate high-reliability memory

- ❑ Hardware-software co-design
- ❑ OS support

Capacity overheads?

Dvé insights

- ❑ Utilize idle memory
- ❑ Overheads applicable only as and when demanded by the application

Skewed memory utilization

- ❑ 50% of the memory is idle 90% of the time
- ❑ Provisioning for peak

Interface to allocate high-reliability memory

- ❑ Hardware-software co-design
- ❑ OS support

Flexible trade-off between capacity and reliability

Summary

Replication for Reliability

Lowers DUE by

4x over Chipkill
172x over IBM RAIM
11% over Intel Memory Mirroring

Coherent Replication for Performance

Improves performance by

5% - 117% over baseline NUMA
3% - 107% over an improved Intel mirroring scheme

On-demand Replication

hardware-software co-design
using OS/compiler support

Artifacts available

https://github.com/adarshpatil/dve
https://adar.sh/dve

Thesis insights and contributions

Āpta

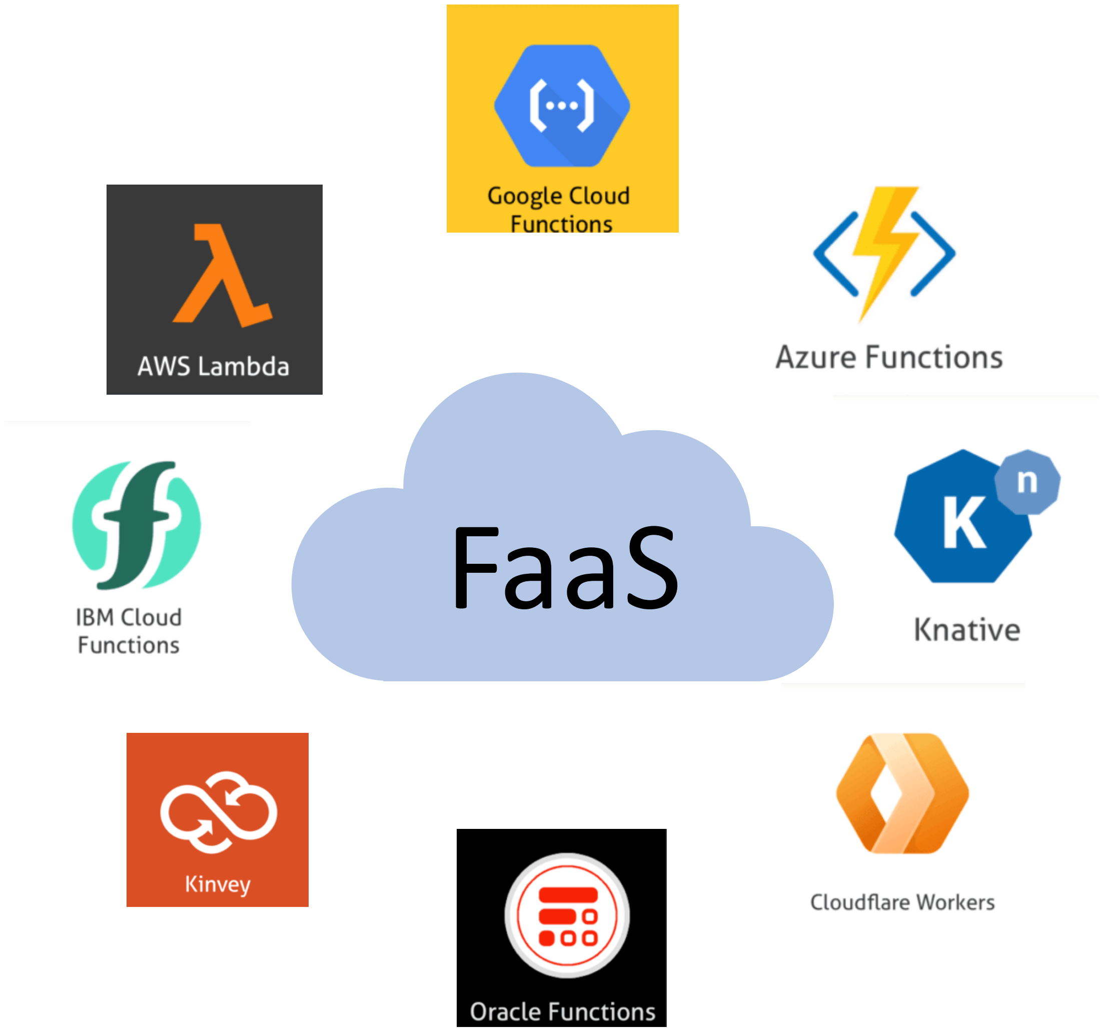
Reliable
CXL.mem

Fault-tolerant disaggregated memory for accelerating FaaS

2

Āpta
Reliable
CXL.mem

"Serverless" Function-as-a-Service



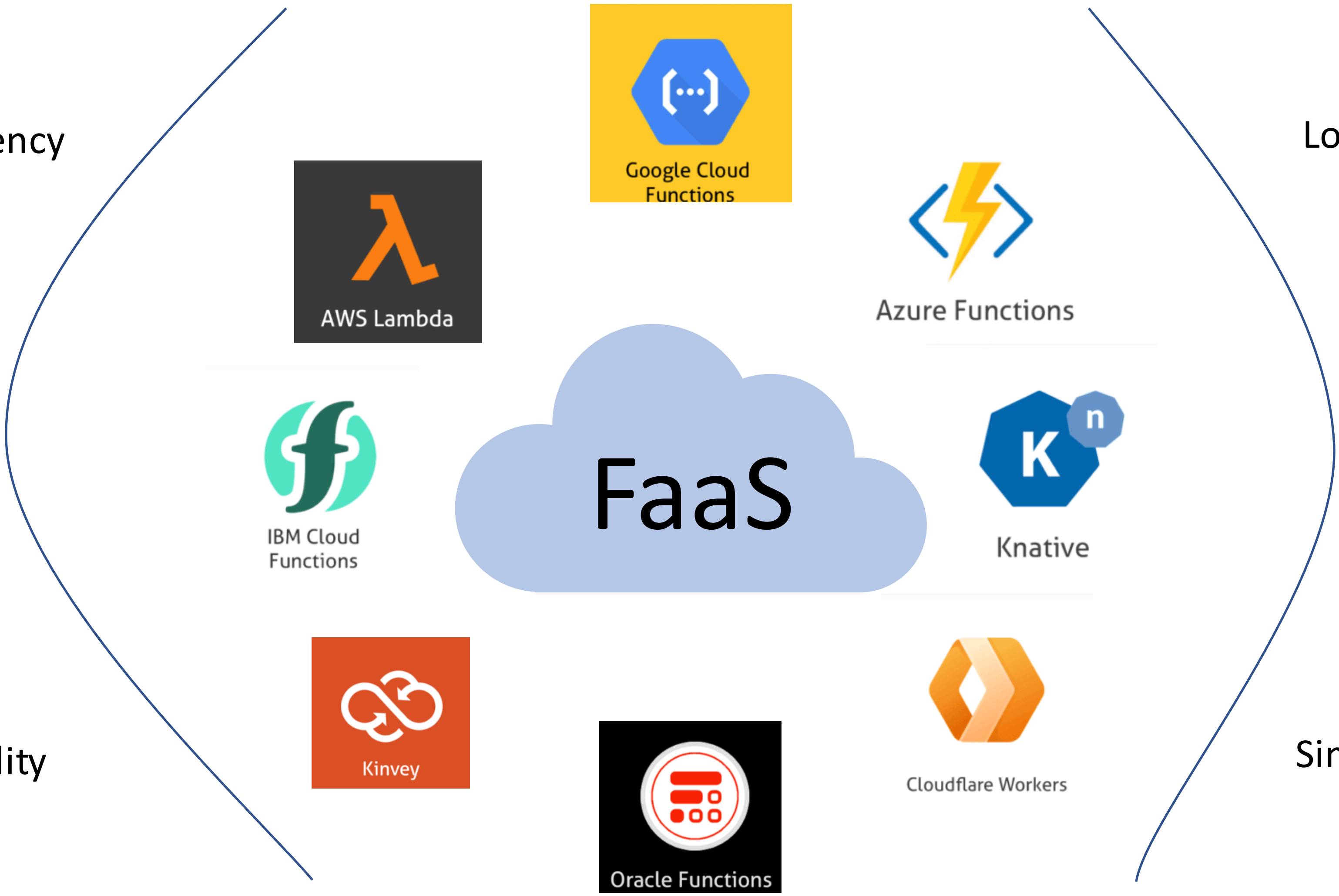
“Serverless” Function-as-a-Service

Efficiency

Elasticity

Productivity

Scalability



Lower TCO

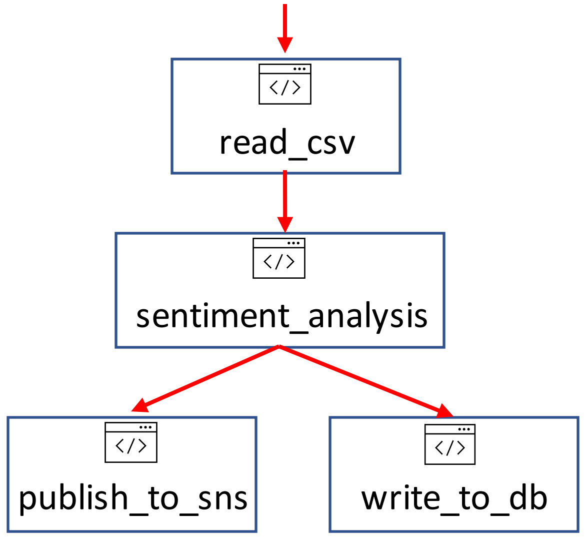
Modularity

Faster time to market

Simplicity

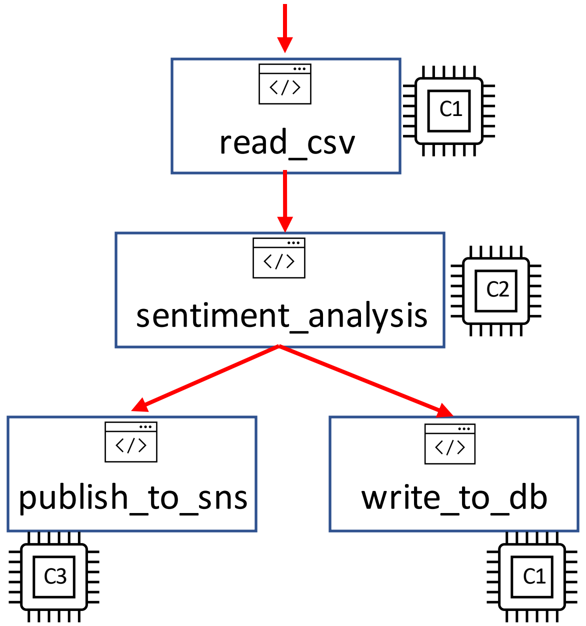
FaaS applications

- State machine workflow of *stateless* functions



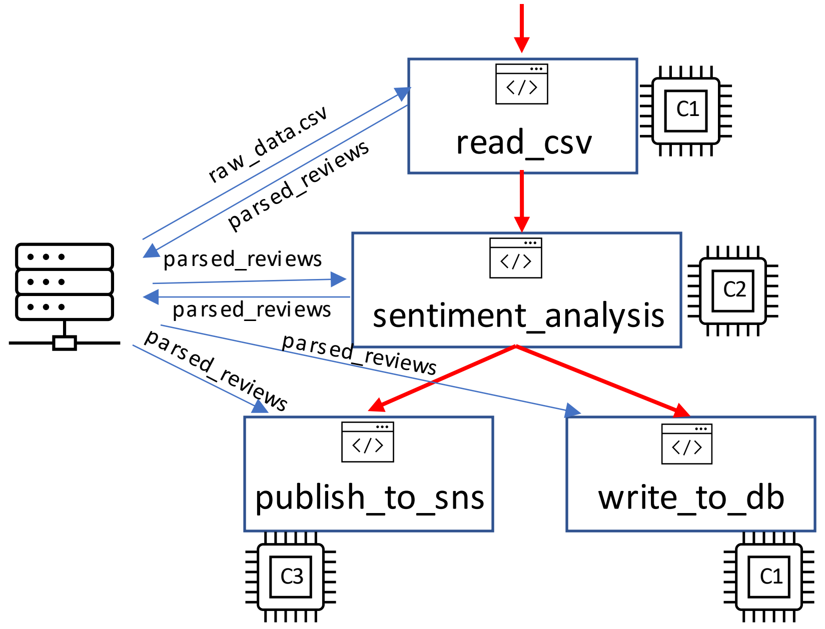
FaaS applications

- State machine workflow of *stateless* functions
- Cloud provider dynamically orchestrates and schedules functions on a fleet of compute servers



Inefficiency of FaaS applications

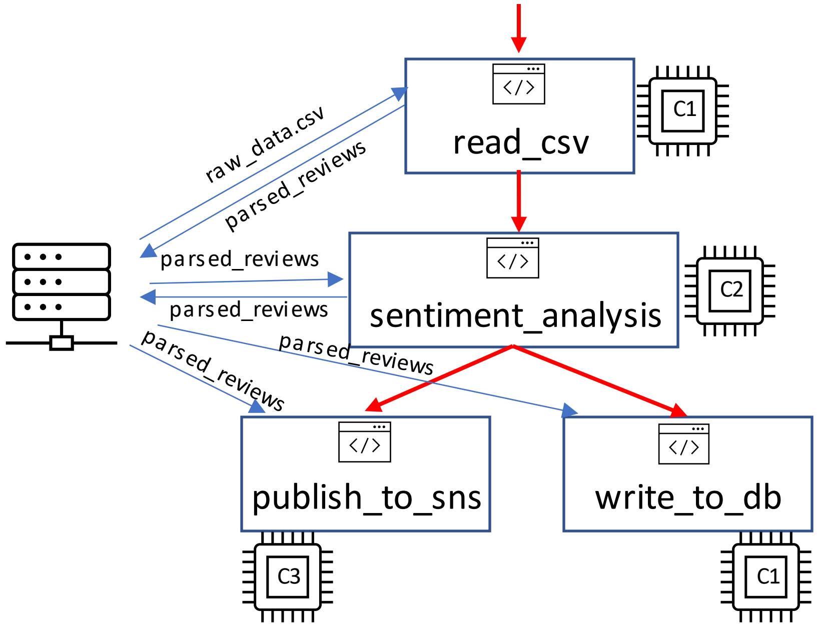
- State machine workflow of *stateless* functions
- Cloud provider dynamically orchestrates and schedules functions on a fleet of compute servers
- State maintained externally as objects in a remote data store



Inefficiency of FaaS applications

- State machine workflow of *stateless* functions
- Cloud provider dynamically orchestrates and schedules functions on a fleet of compute servers
- State maintained externally as objects in a remote data store
  - × Splitting state-compute adds communication overheads

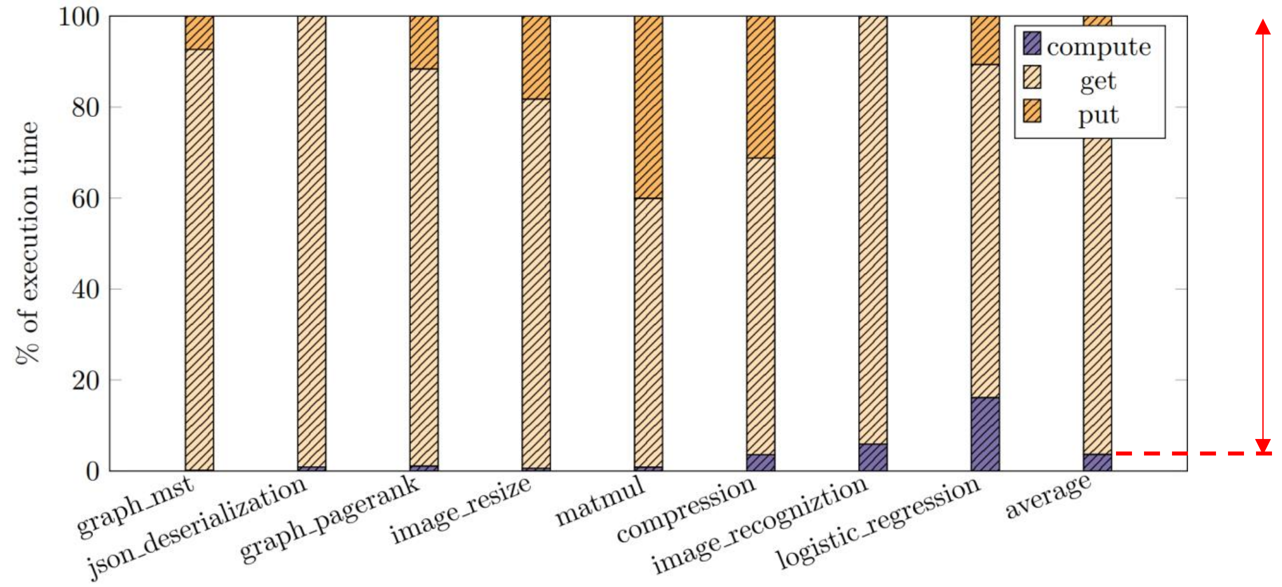
How much?



Āpta
Reliable
CXL.mem

Quantifying communication overheads

- Functions from FunctionBench and SeBS benchmark suites
- Compute - Optimized with Intel OneAPI, run on 16-core Skylake CPUs
- Communication - Amazon S3 object store (median of 100 executions)

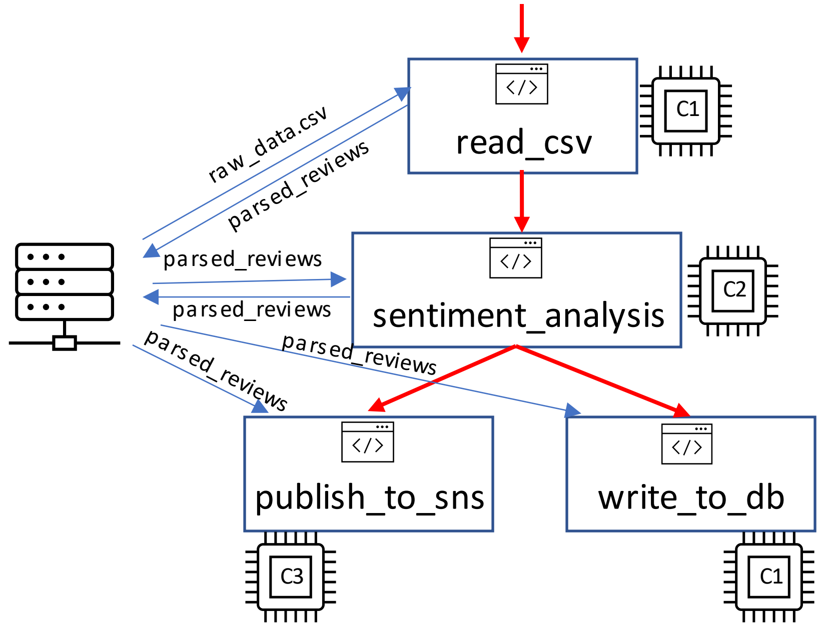


96% of execution time is spent in retrieving data from S3

Inefficiency of FaaS applications

- State machine workflow of *stateless* functions
- Cloud provider dynamically orchestrates and schedules functions on a fleet of compute servers
- State maintained externally as objects in a remote data store
  - × Splitting state-compute adds communication overheads
  - × Communication overheads severely limit performance

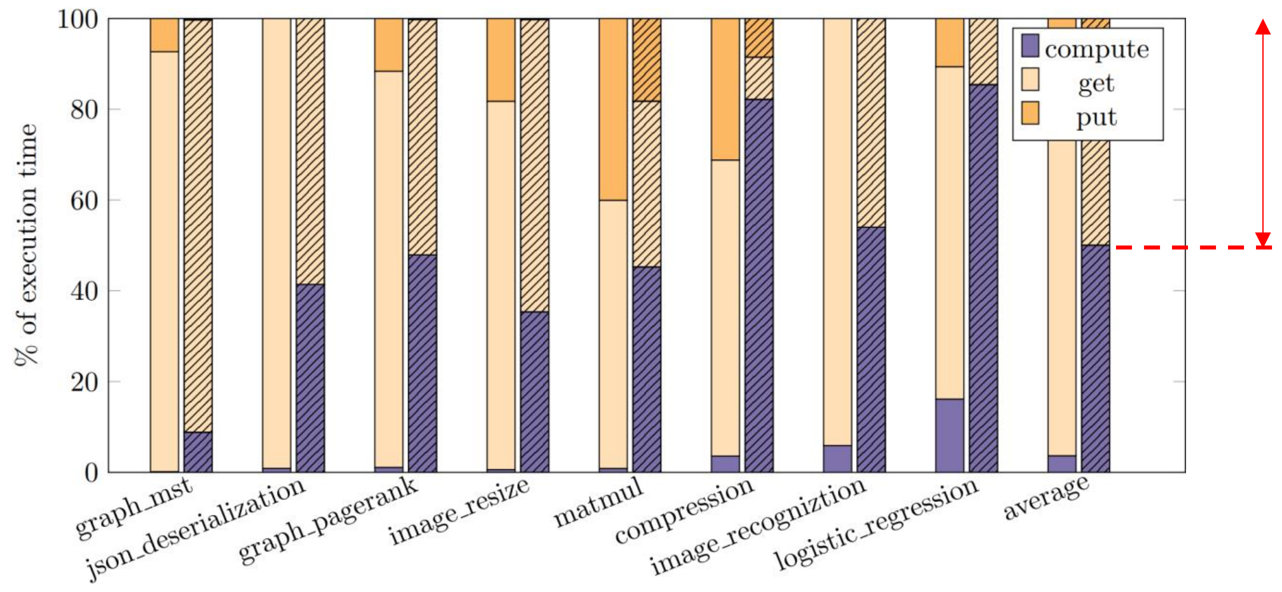
Can we do better?



Āpta
Reliable
CXL.mem

Can we do better?

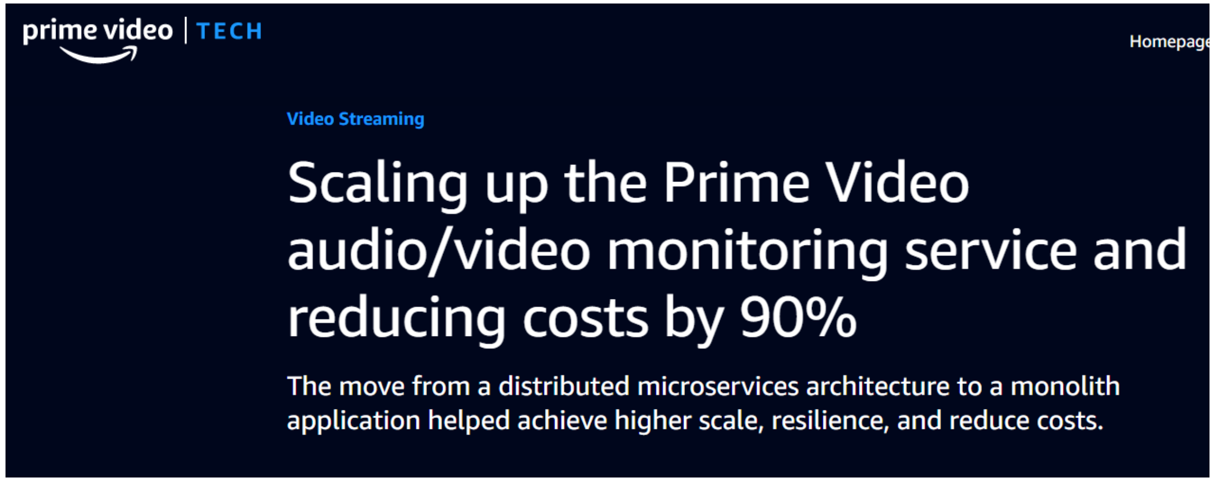
- High-performance in-memory object store
- One-sided RDMA verbs to read/write objects
- Infiniband network (Mellanox ConnectX-3 NIC on PCIe-gen3 x16)



51% of execution time is spent in retrieving data from object store

Āpta
Reliable CXL.mem

The problem: Data communication

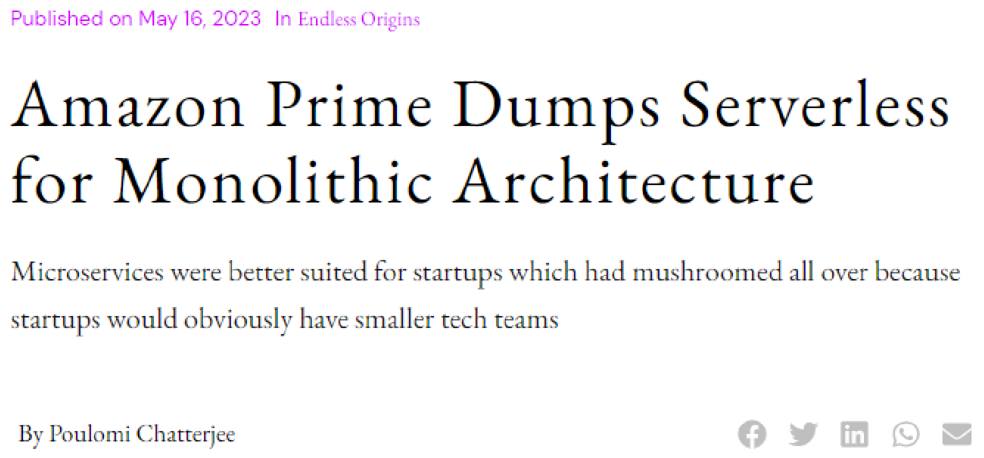

prime video | TECH

Homepage

Video Streaming

Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%

The move from a distributed microservices architecture to a monolith application helped achieve higher scale, resilience, and reduce costs.


Published on May 16, 2023 In Endless Origins

Amazon Prime Dumps Serverless for Monolithic Architecture

Microservices were better suited for startups which had mushroomed all over because startups would obviously have smaller tech teams

By Poulomi Chatterjee

"The two most expensive operations in terms of cost were the orchestration workflow and when data passed between distributed components."

Performance potential for Āpta

Object-granular CXL disaggregated memory

Āpta
Reliable CXL.mem

Performance potential for Āpta

Object-granular CXL disaggregated memory

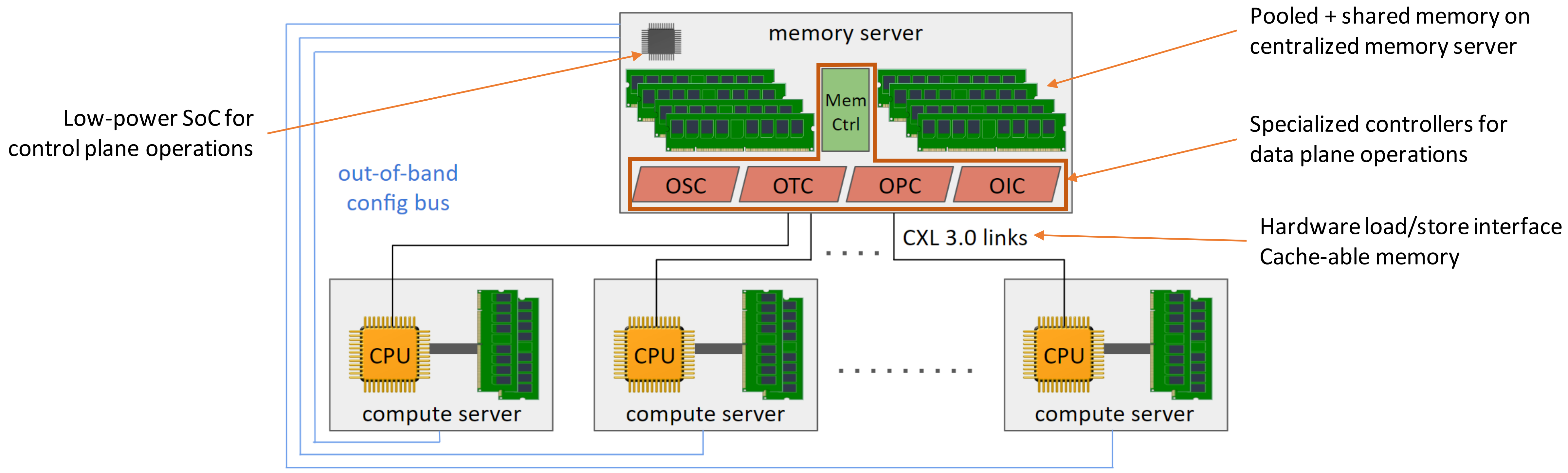
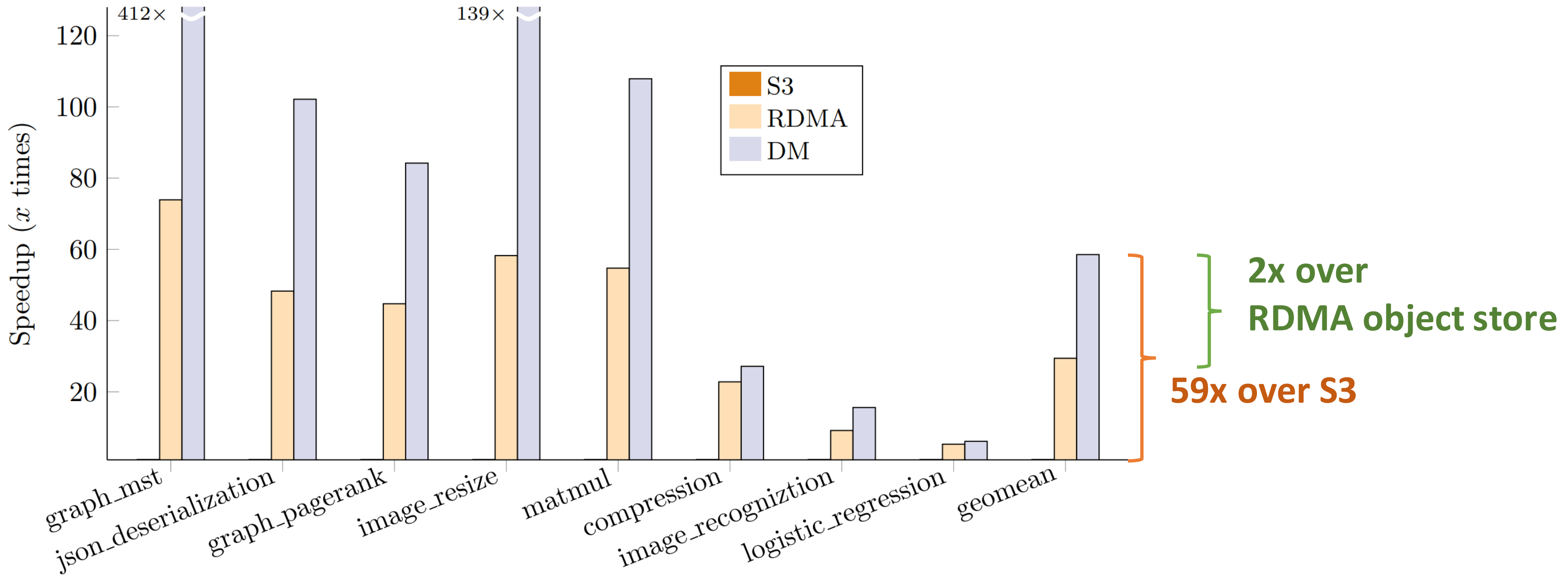

memory server
Mem Ctrl
OSC
OTC
OPC
OIC
out-of-band config bus
CXL 3.0 links
CPU
compute server
CPU
compute server
CPU
compute server
Low-power SoC for control plane operations
Pooled + shared memory on centralized memory server
Specialized controllers for data plane operations
Hardware load/store interface
Cache-able memory

Fig: Āpta system schematic

Performance potential for Āpta

With OpenCAPI-like access latency / bandwidth for DM[†]



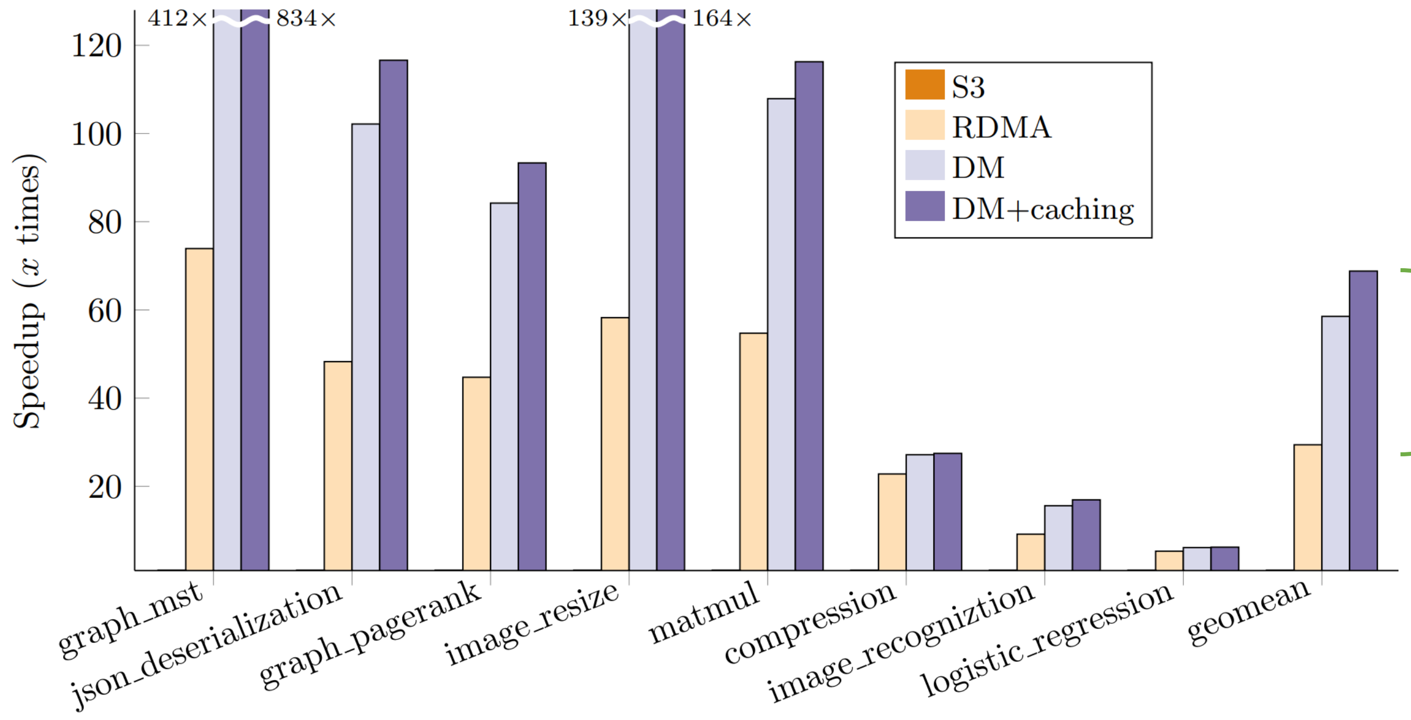
13% communication overheads (Recall 51% for RDMA-based object store)

† ThymesisFlow [MICRO 20]

Āpta
Reliable
CXL.mem

Performance potential for Āpta

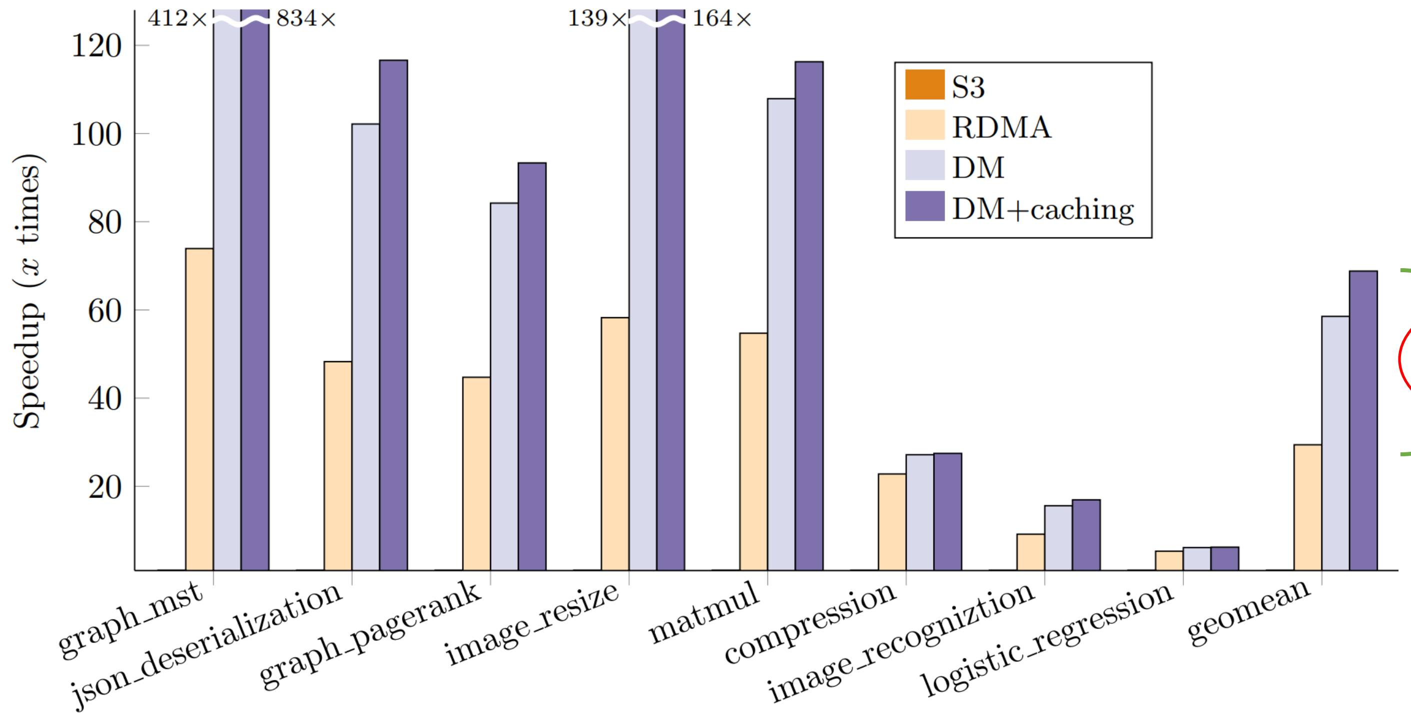
Object caching at compute server



2.3x over RDMA object store

Performance potential for Āpta

Object caching at compute server



2.3x over
RDMA object store

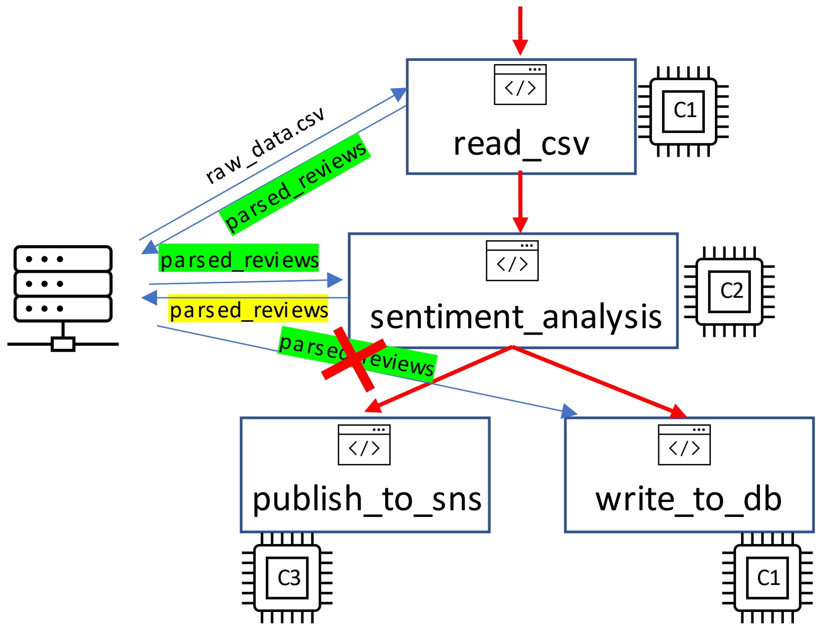
Our target!

The CXL.mem coherence

- Enforcing strong consistency in presence of caching

The CXL.mem coherence

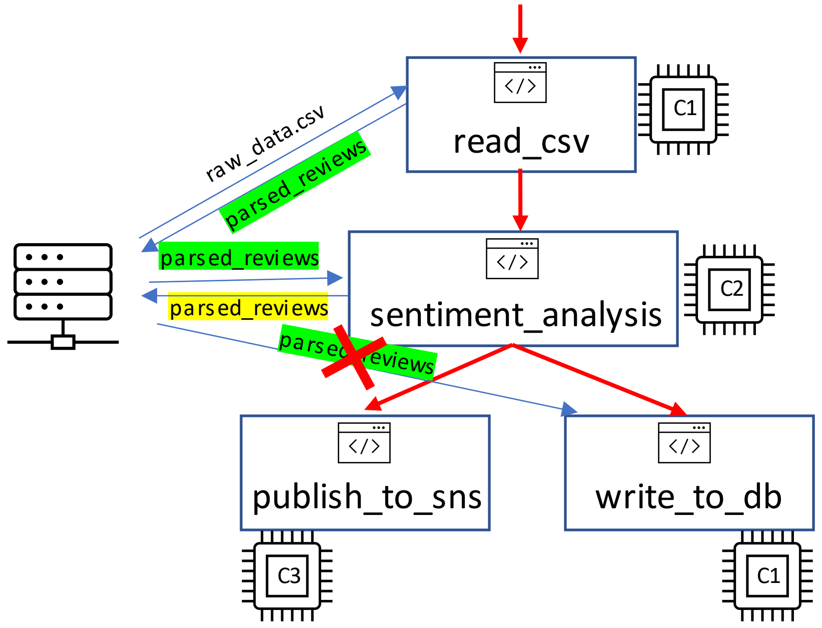
- Enforcing strong consistency in presence of caching



The CXL.mem coherence

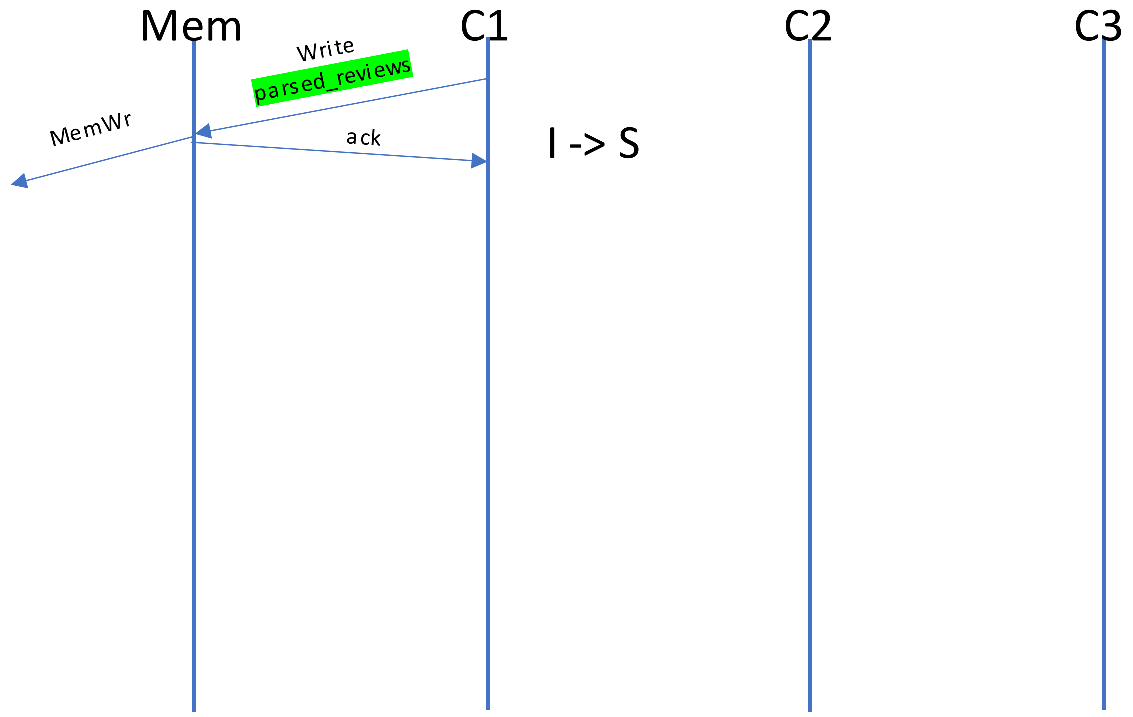
- Enforcing strong consistency in presence of caching

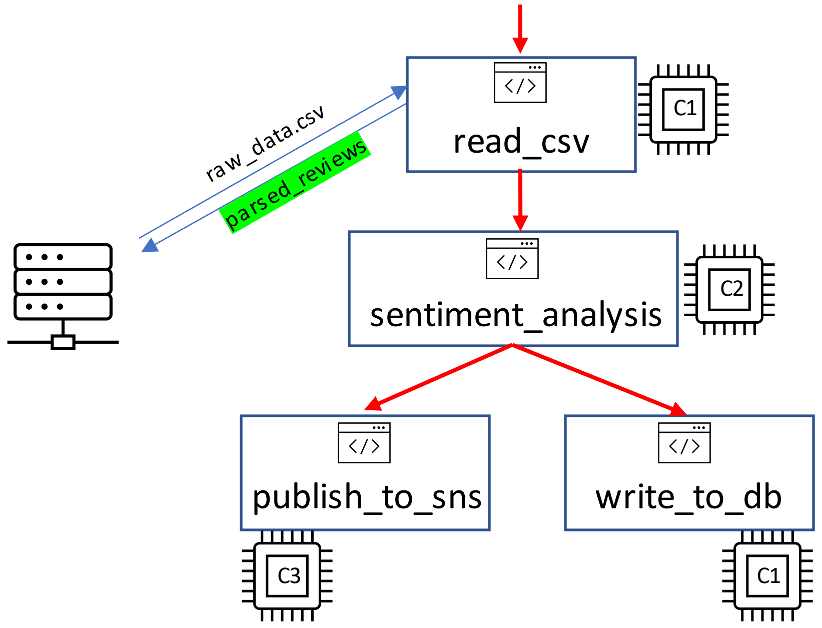
  CXL 3.0 inter-node coherence protocol
  Enforces SWMR invariant



The CXL.mem coherence

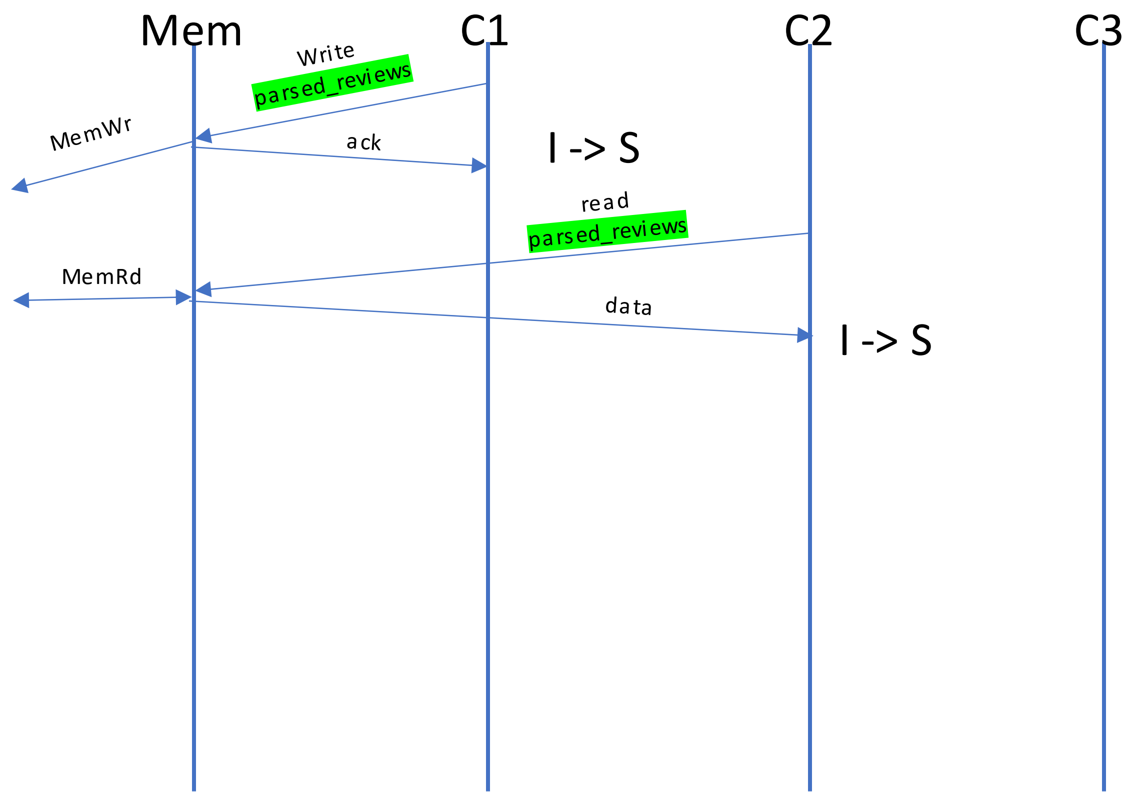
- Enforcing strong consistency in presence of caching
    - CXL 3.0 Inter-node coherence protocol
    - Enforces SWMR invariant

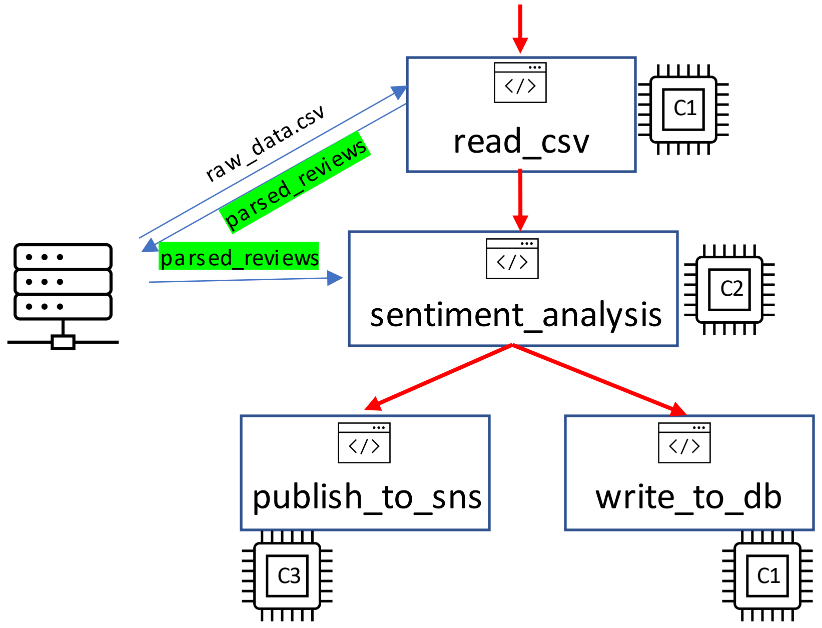




The CXL.mem coherence

- Enforcing strong consistency in presence of caching
  - Inter-node coherence protocol
  - CXL 3.0 protocols enforce SWMR invariant

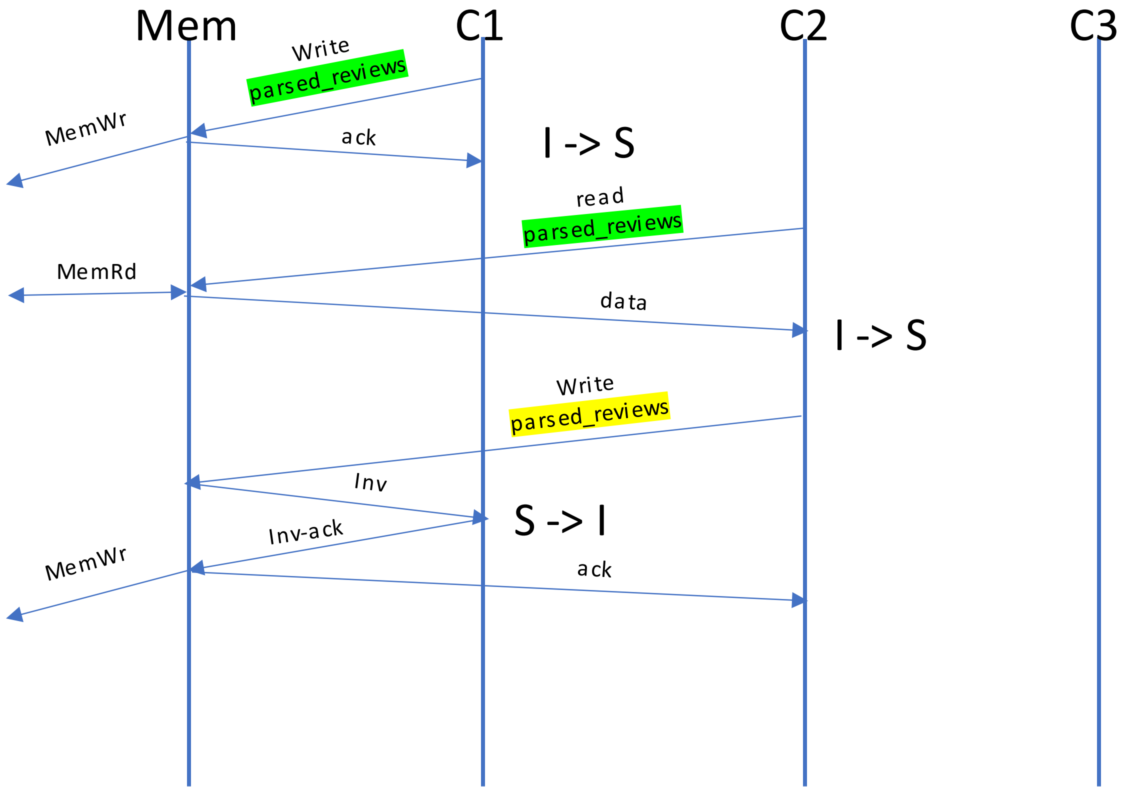


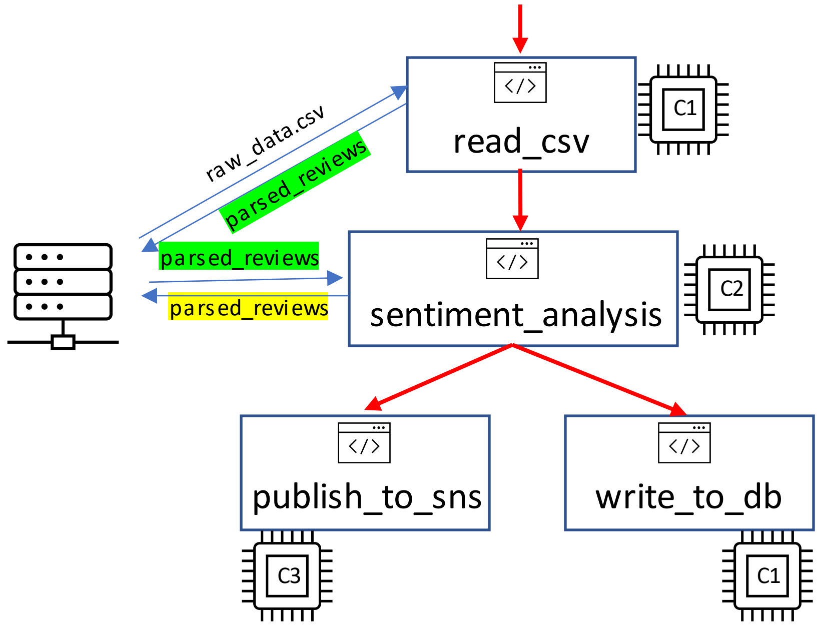


Āpta
Reliable CXL.mem

The CXL.mem coherence

- Enforcing strong consistency in presence of caching
  - Inter-node coherence protocol
  - CXL 3.0 enforce SWMR invariant

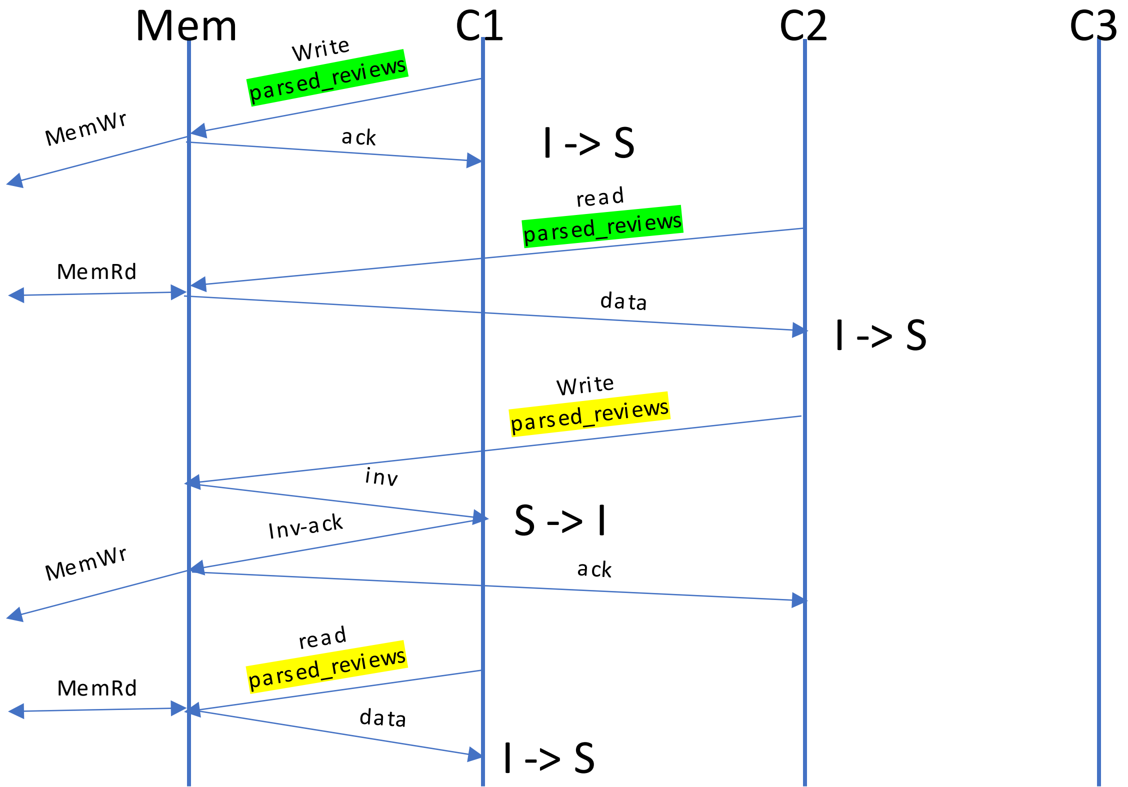


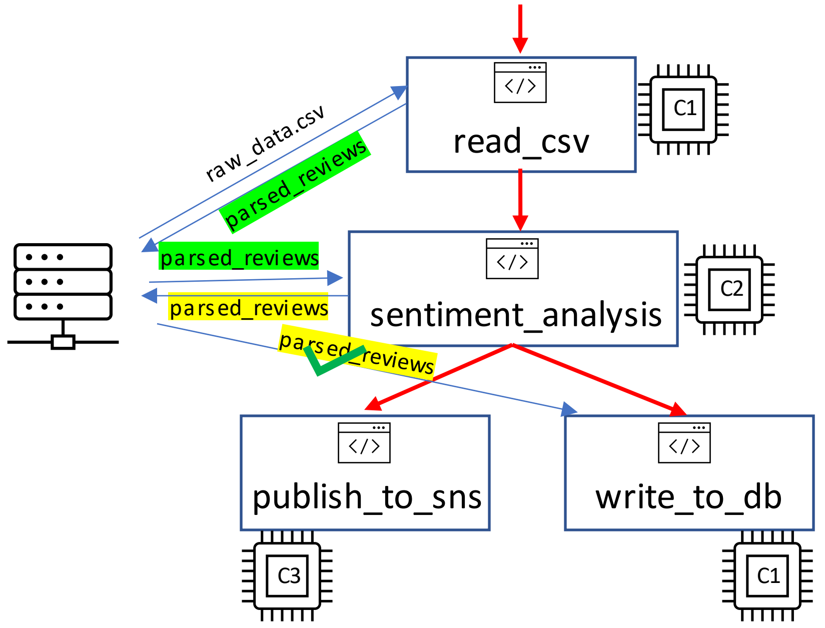


Āpta
Reliable CXL.mem

The CXL.mem coherence

- Enforcing strong consistency in presence of caching
  - Inter-node coherence protocol
  - CXL 3.0 enforce SWMR invariant





The need for fault-tolerant coherence

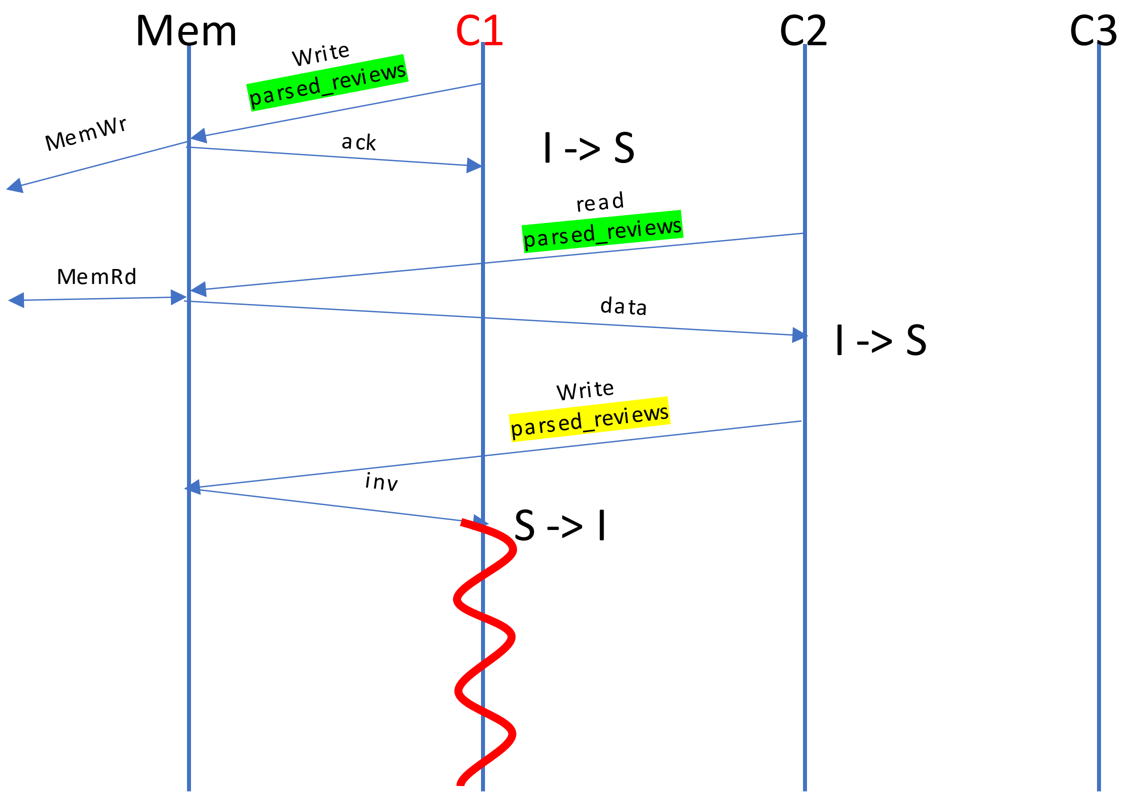
- The fault tolerance problem
  Compute server failures

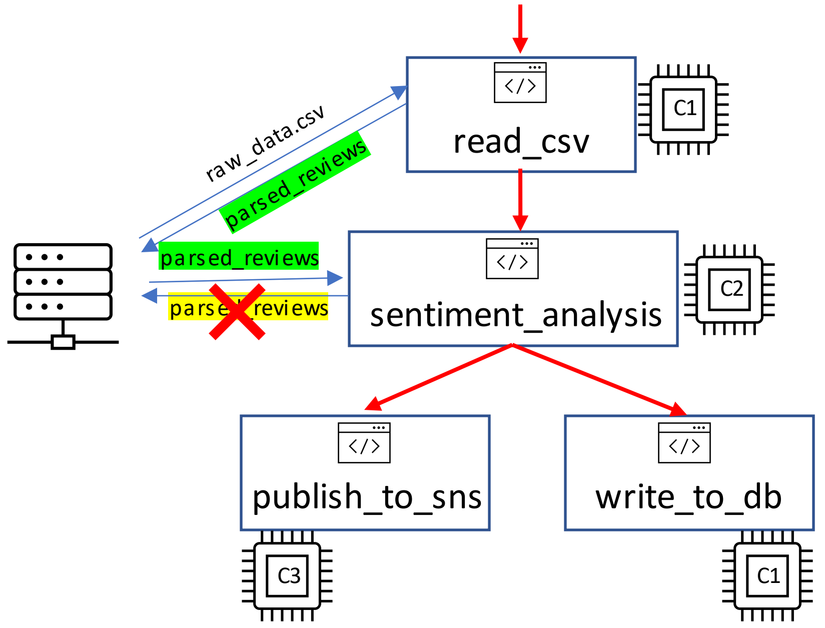
Āpta Reliable CXL.mem

The need for fault-tolerant coherence

- The fault tolerance problem

  Compute server failures



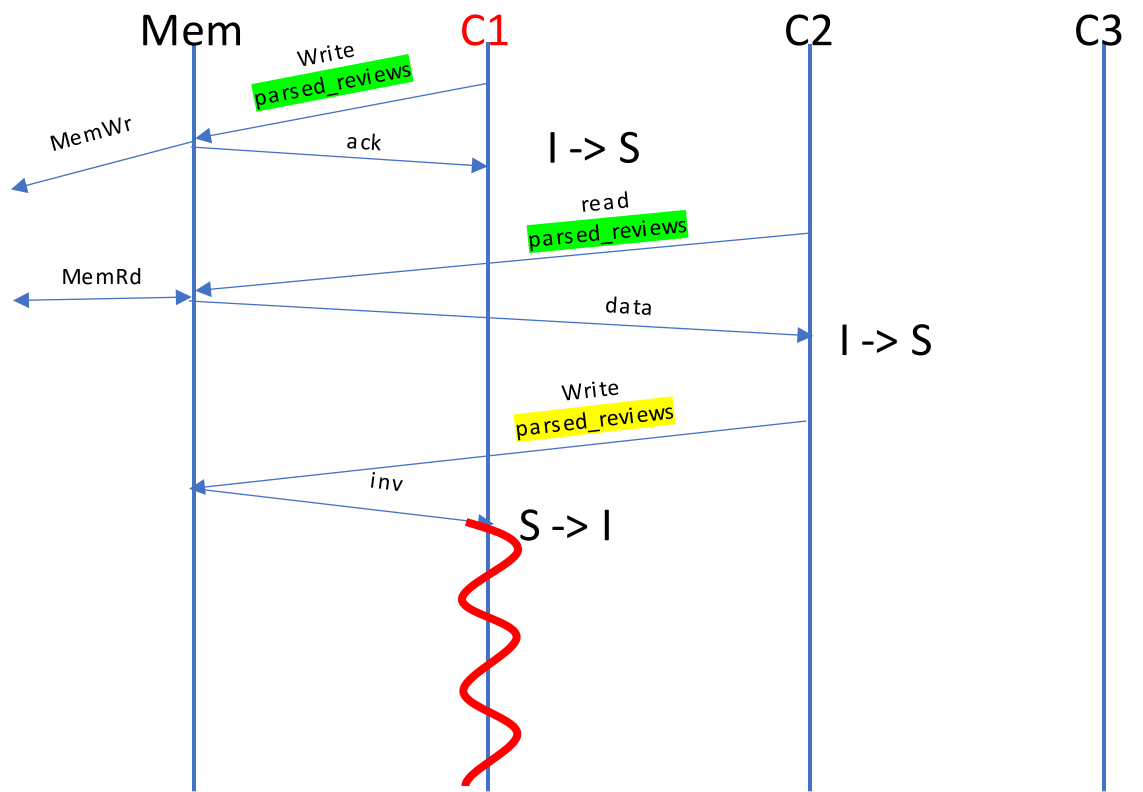


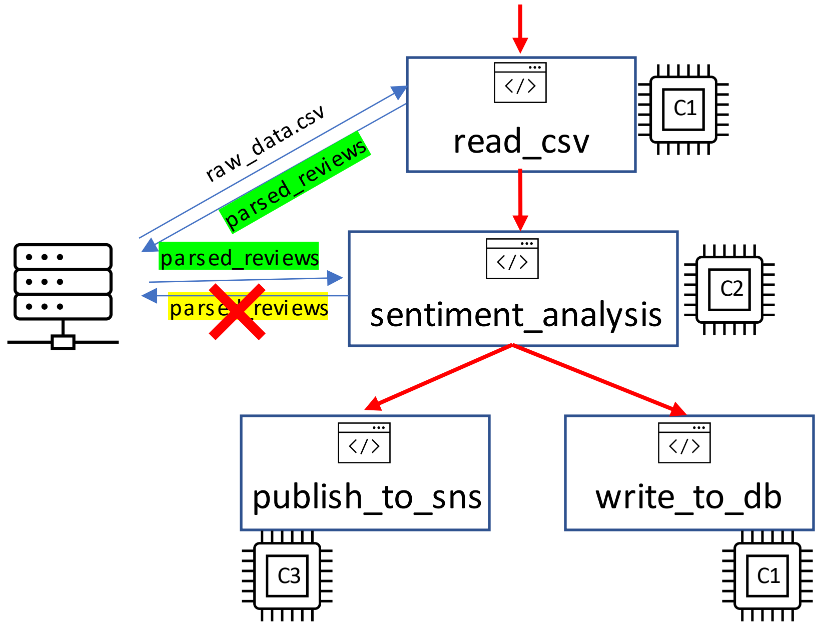
Āpta Reliable CXL.mem

The need for fault-tolerant coherence

- The fault tolerance problem

  Compute server failures – blocking

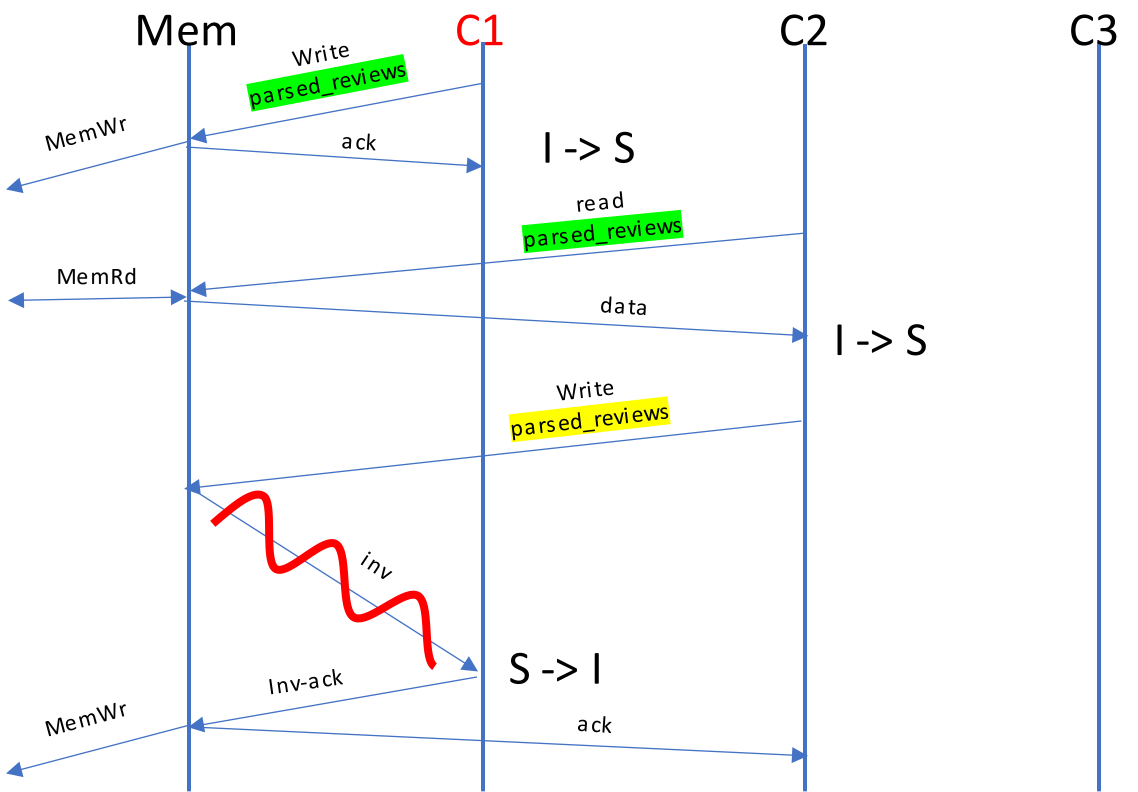


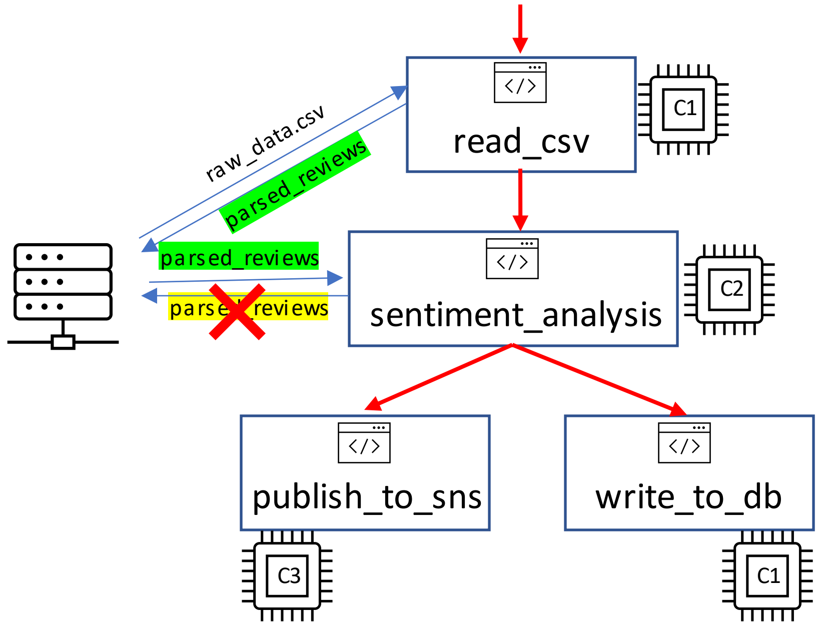


The need for fault-tolerant coherence

- The fault tolerance problem

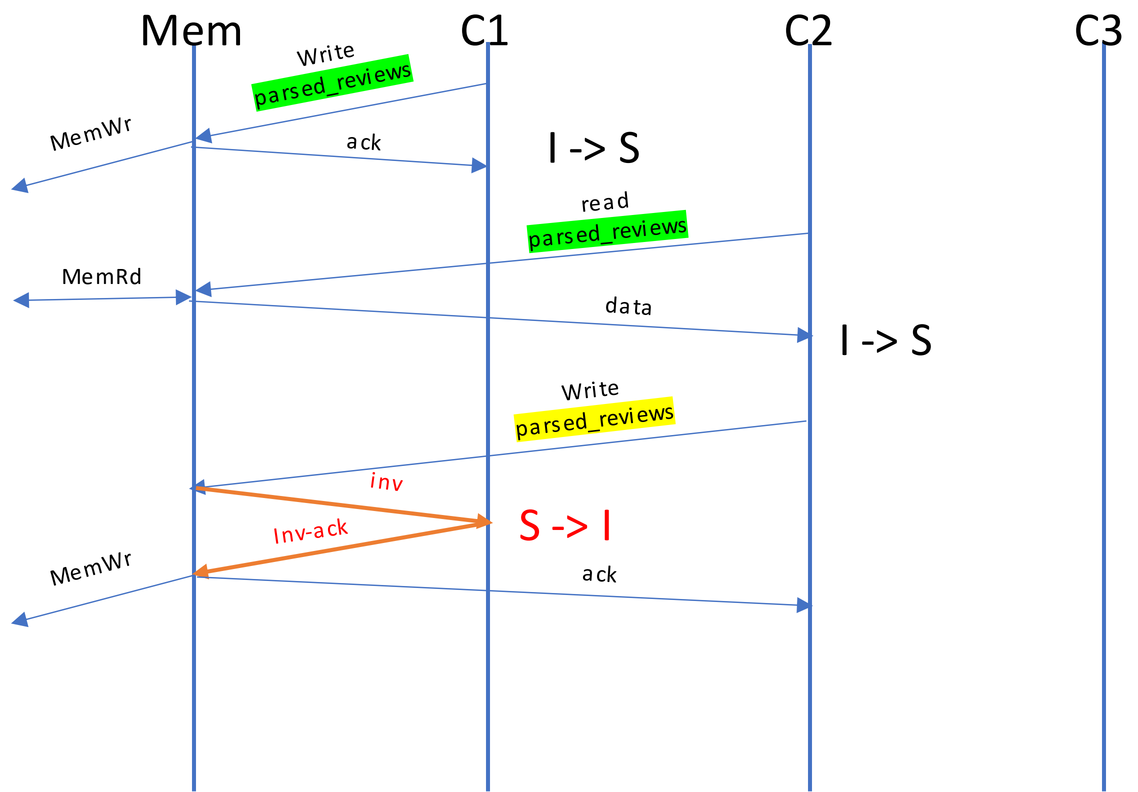
  Compute server failures – blocking
  Network congestions – high tail latency

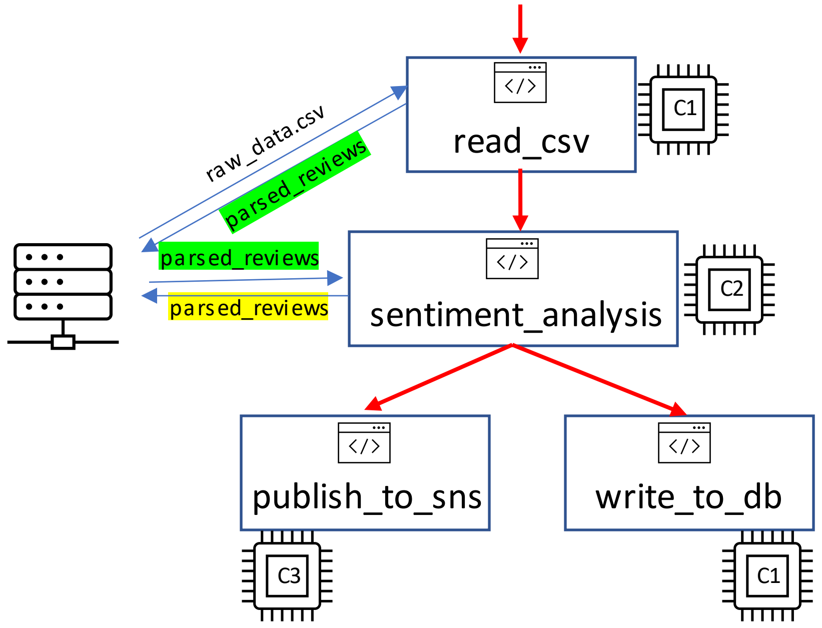




Key Problem

- Invalidations are in the critical path of a write



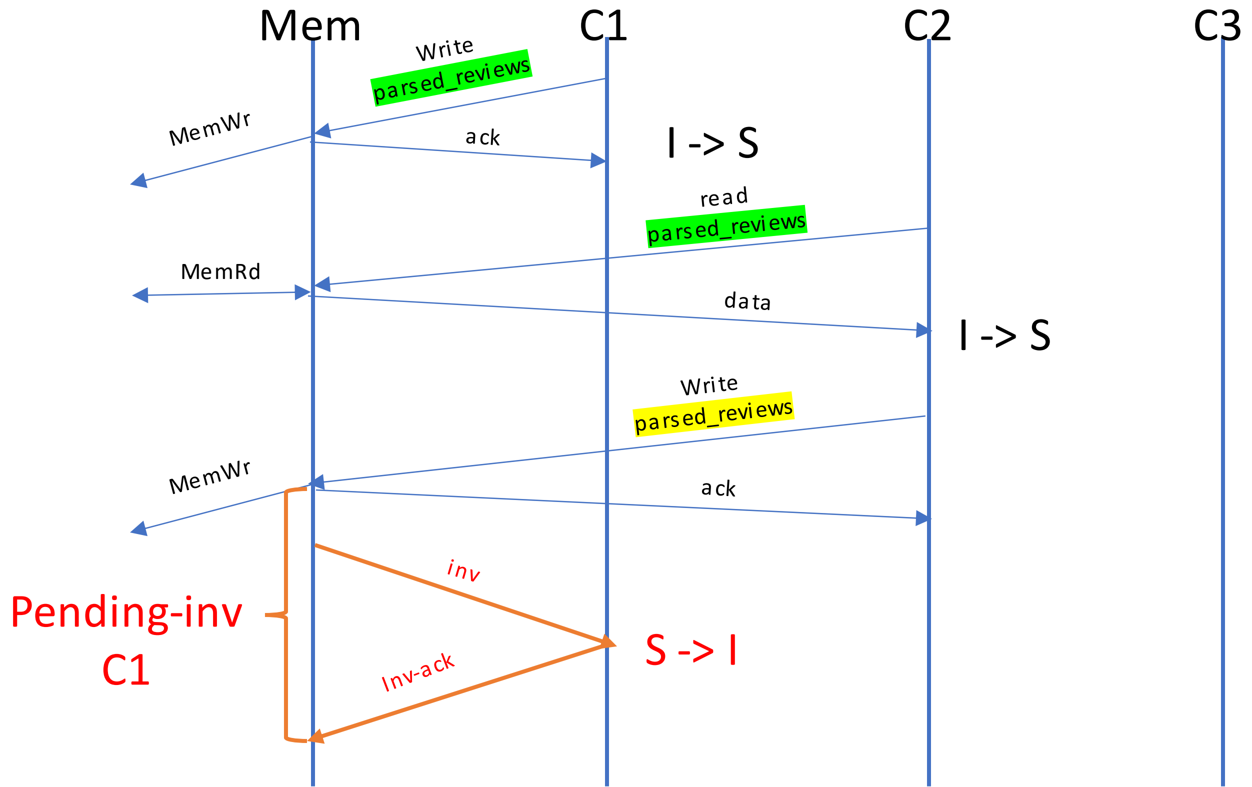


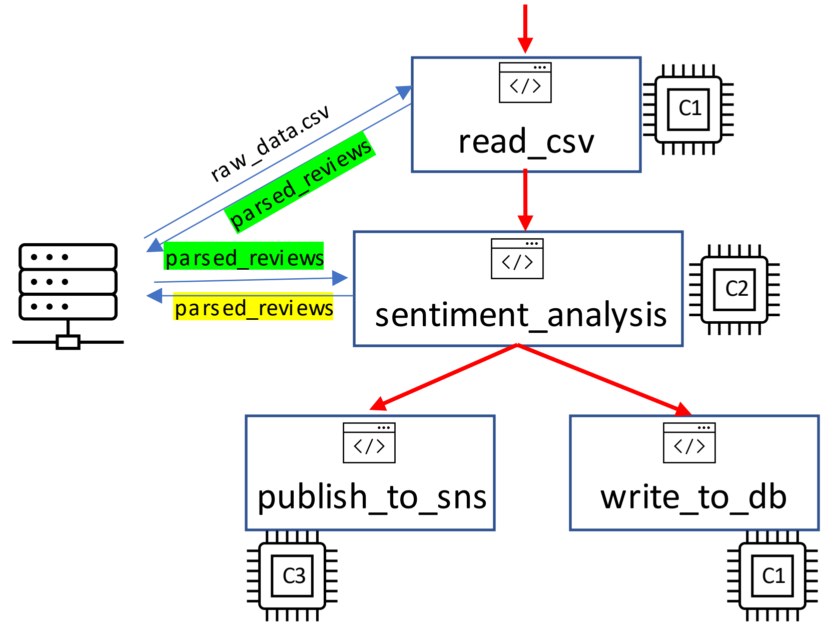
Āpta: Fault-tolerant Coherence Protocol

i. Lazy invalidation policy

Write is acknowledged immediately

Invalidation messages are sent asynchronously and tracked

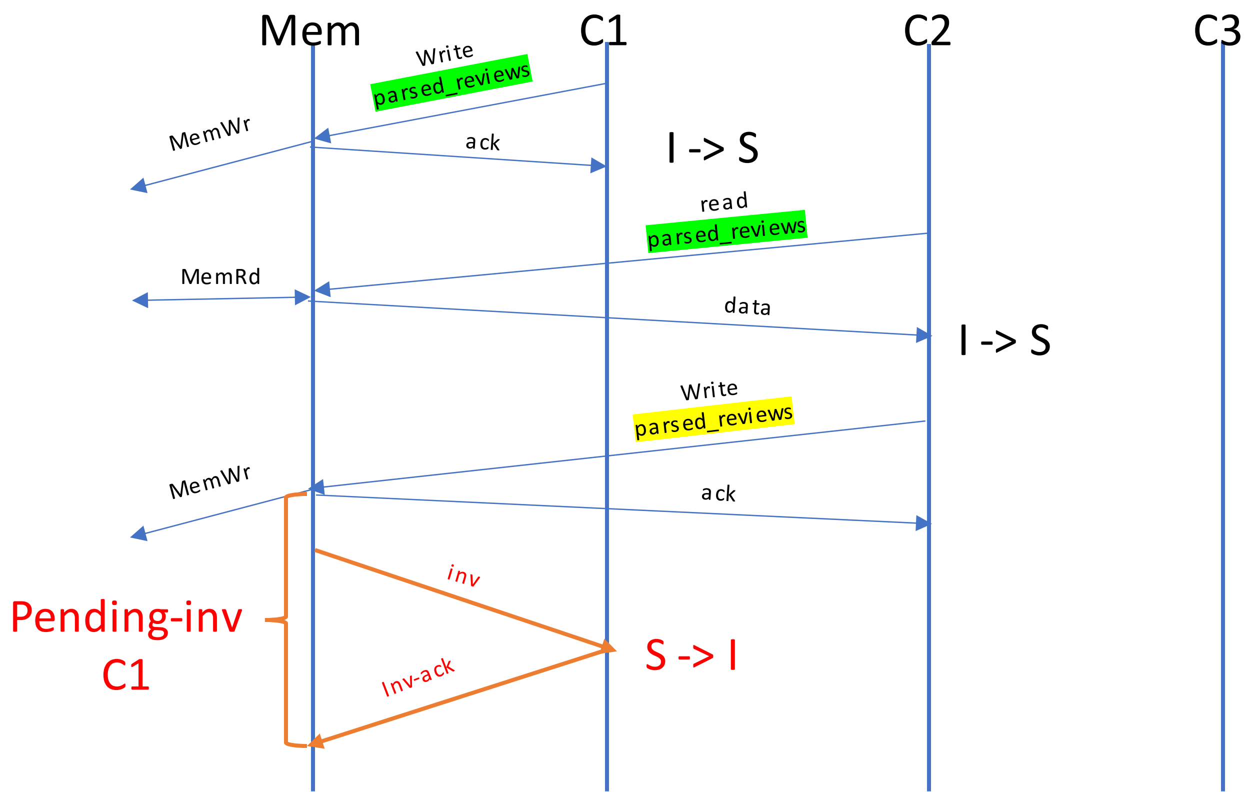


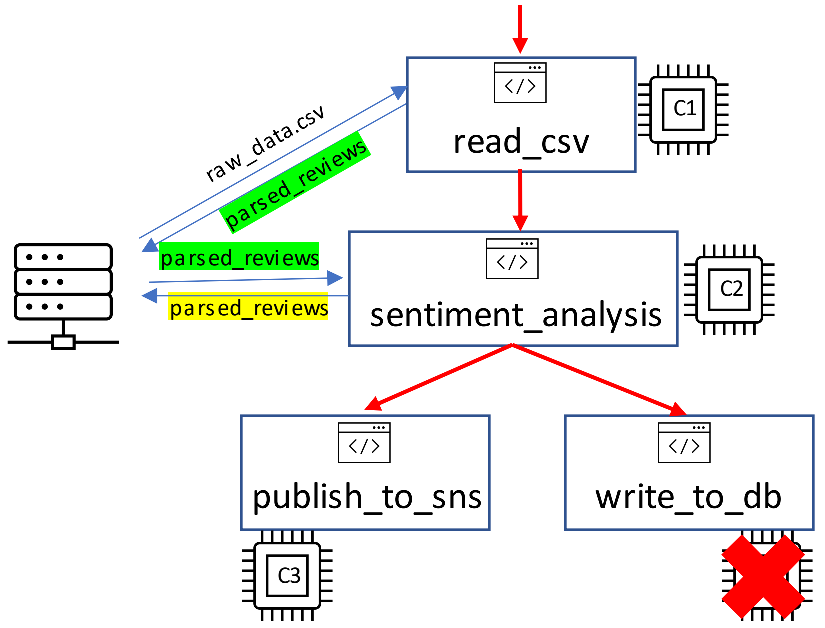


Āpta: Fault-tolerant Coherence Protocol

ii. Coherence-aware function scheduling

Never schedules function invocations on servers with pending invalidation-acknowlegements





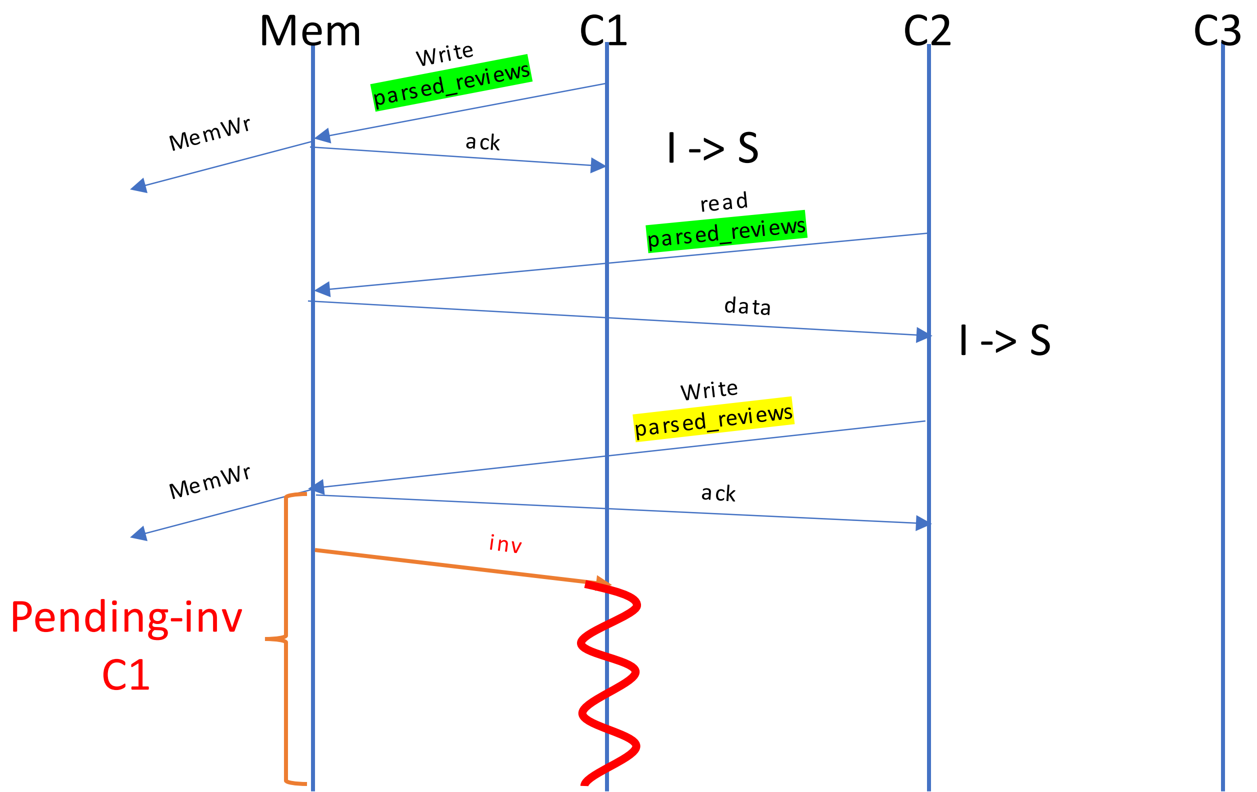
Āpta: Fault-tolerant Coherence Protocol

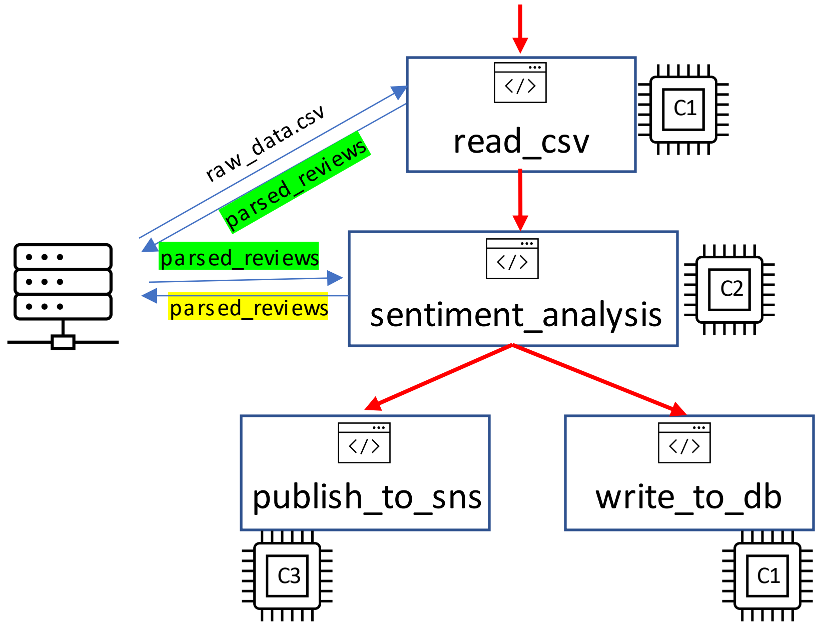
Lazy linearizability (Lazy invalidation policy + Coherence-aware scheduling)

Āpta: Fault-tolerant Coherence Protocol

Lazy linearizability (Lazy invalidation policy + Coherence-aware scheduling)

Ensures compute server fault-tolerant operation





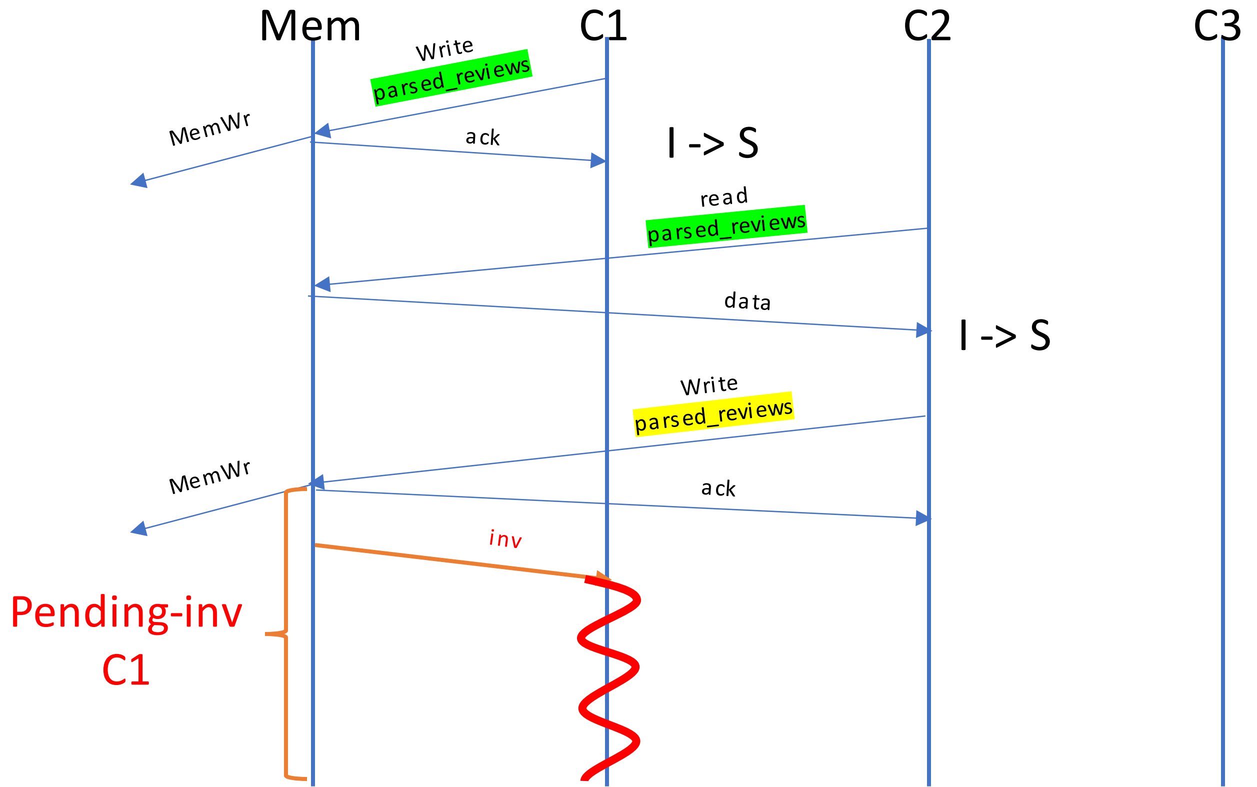
Āpta Reliable CXL.mem

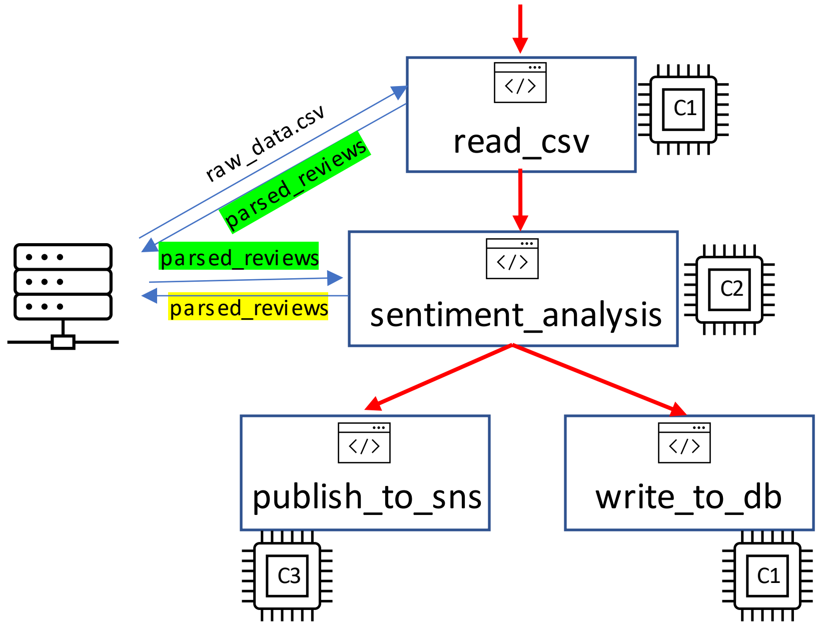
Āpta: Fault-tolerant Coherence Protocol

Lazy linearizability (Lazy invalidation policy + Coherence-aware scheduling)

Ensures compute server fault-tolerant operation

Provides line-rate coherence





Āpta Summary

Accelerating function-as-a-service

Improves performance by
40% - 142% over RDMA
21% – 90% over RDMA + caching
15% - 42% over un-cached CXL

Fault-tolerant coherence protocol

Protocol verified in Murϕ model checker
32% lower standard deviation of exec time

Object-granular Disaggregated Memory

CXL-based shared memory IPC
Bulk cache-line loads
Transaction atomic durability

Artifacts available

https://github.com/adarshpatil/apta
https://adar.sh/apta

Summary: Thesis contributions

Dvé
Memory Replication

- Unique design point in the reliability design space
- Explored a novel extrapolation of two-tier approach
- Introduced flexible / on-demand reliability

Āpta
Reliable CXL.mem

- Showcases a use case for shared disaggregated memory
- Proposes a lightweight fault tolerance solution
- Consistency & availability via fault-tolerant coherence

Summary: Retrospective contemplation

Critical analysis
Software complexity: OS, scheduler
Problems of scale: throughput, co-location
Performance corner cases: worst-case scenarios

Lessons learnt
Mental model of correctness during development
Think and reason from first principles

Takeaways
Robust reliability is key for next gen memory • technology agnostic, demand reliability (DDR, LPDDR, GDDR) • hardware disaggregated memory (new fault models)
Application ***driven*** architecture • Hardware fault-tolerance must match application evolution • Good understanding of application characteristics
Revisit design decisions in-step with advances in technology • shared memory systems today are more closely resembling traditional distributed systems
End-to-end argument to system design [Saltzer, 1984]
Tame complexity through modularization

Future Research Directions

- Value-added disaggregated memory

  Reliability, Availability, Security, Compression….

- Redesigning distributed datacenter co-ordination services for modern hardware

  Kubernetes (scheduler), Chubby (locks), Kafka (configuration)….

- Efficient shared disaggregated memory

  Heterogenous compute, consistency-directed coherence mechanisms

Future Research Directions

- Value-added disaggregated memory

  Reliability, Availability, Security, Compression....

- Redesigning distributed datacenter co-ordination services for modern hardware

  Kubernetes (scheduler), Chubby (locks), Kafka (configuration)....

- Efficient shared disaggregated memory

  Heterogenous compute, consistency-directed coherence mechanisms

What's next?

#OpenToWork – Industry Research positions
Current: Post-doc at University of Edinburgh

What I...

Enjoyed... the journey

- Going with a hunch, high-level problem– usually after discussions with Vijay
- Reading related work critically
- Designing experiments to demonstrate the problem (motivation)
- Inception of a workable solution
- Proof of viability – pen/paper, creative descriptions
- Designing experiments to demonstrate the solution
- Refining the idea & solution
- Putting it all together: Presenting, writing the problem, solution, pros/cons

Disliked...

- Convincing reviewers
- The journey of solitude – the imposter syndrome, the large gaps

What I...

Would do differently (with benefit of hindsight)....

- Better evaluation techniques
  - e.g., Learn HDL, try FPGA-based prototype
- Better paper positioning for maximum impact
  - e.g., Apta is an intersection of KV store + FaaS performance + CXL protocol

- Time-management: better context switching between projects
- Dealing with rejection (still not mastered this)
- Prioritize mental wellbeing: self-reassurance, self-belief, avoid comparing