Concord: Rethinking Distributed Coherence for Software Caches in Serverless Environments

HPCA ‘25

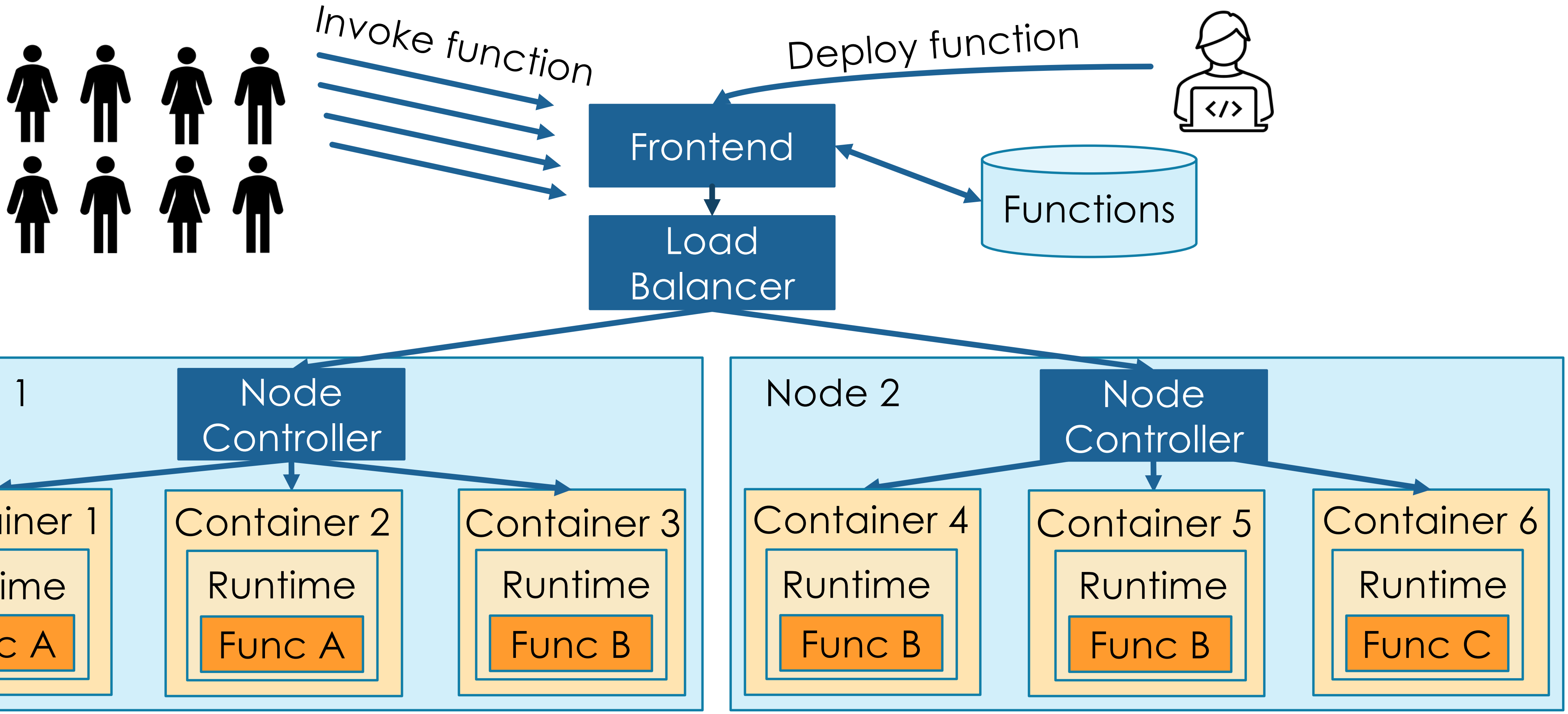
Jovan Stojkovic, Chloe Alverti, Alan Andrade, Nikoleta Iliakopoulou, Tianyin Xu, Hubertus Franke*, Josep Torrellas

University of Illinois at Urbana-Champaign, *IBM Research

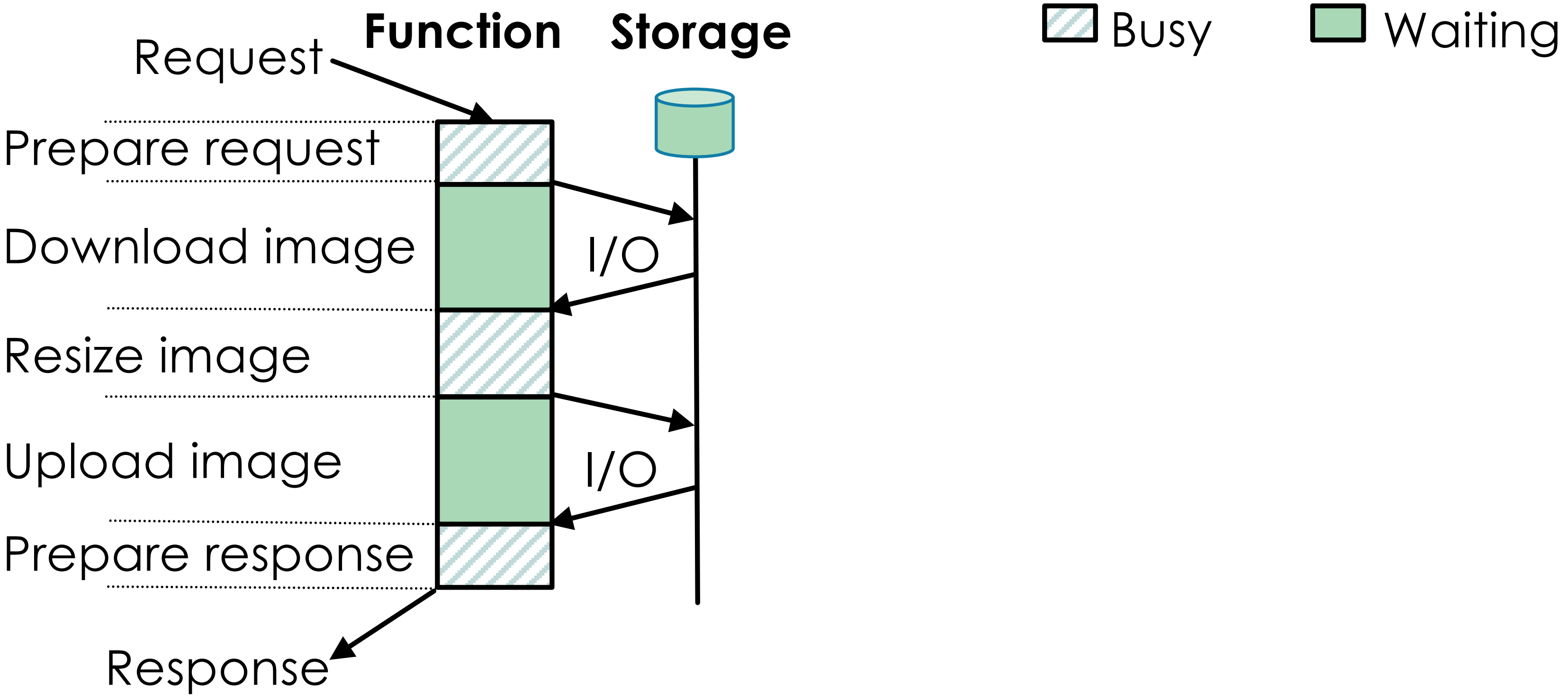
What is Serverless Computing?

- Serverless computing popular cloud paradigm
  - Users deploy apps, providers provision resources
- Many benefits
  - Simple and modular programming
  - Automatic resource scaling
  - Pay-as-you-go model
- AWS Lambda, Microsoft Azure, IBM Cloud

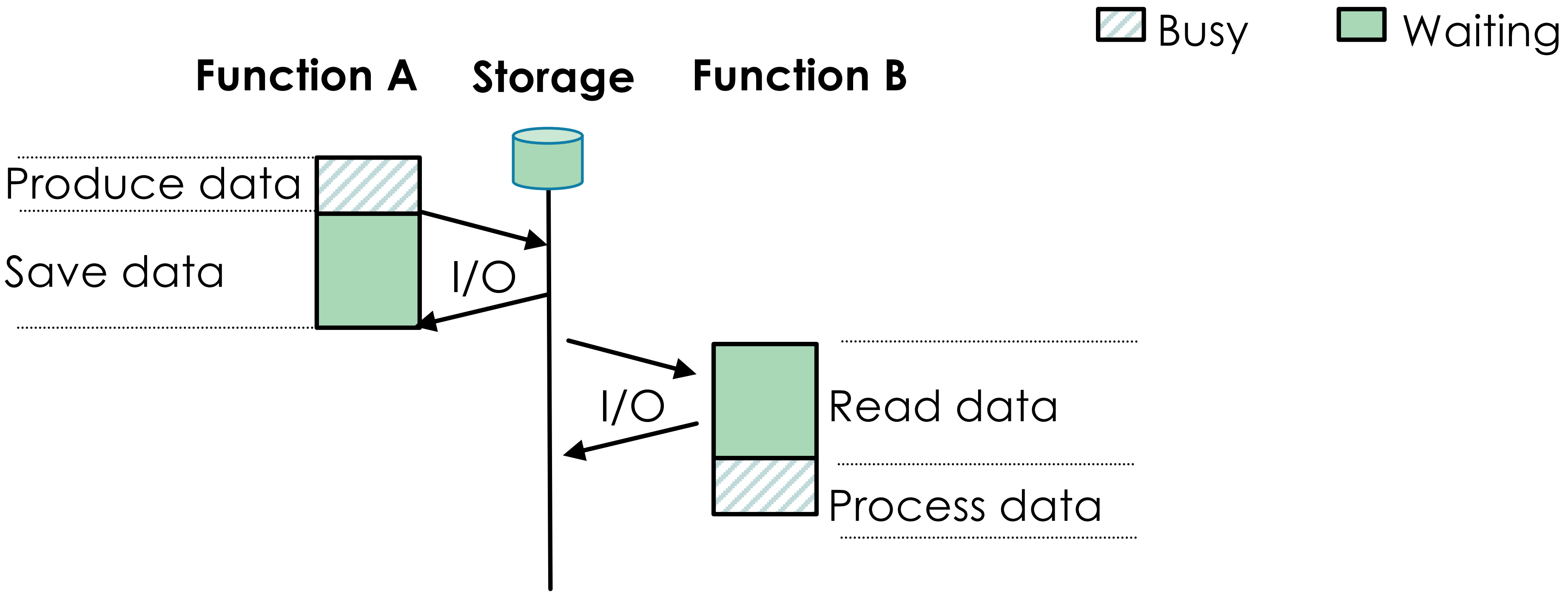
How Does Serverless Computing Work?



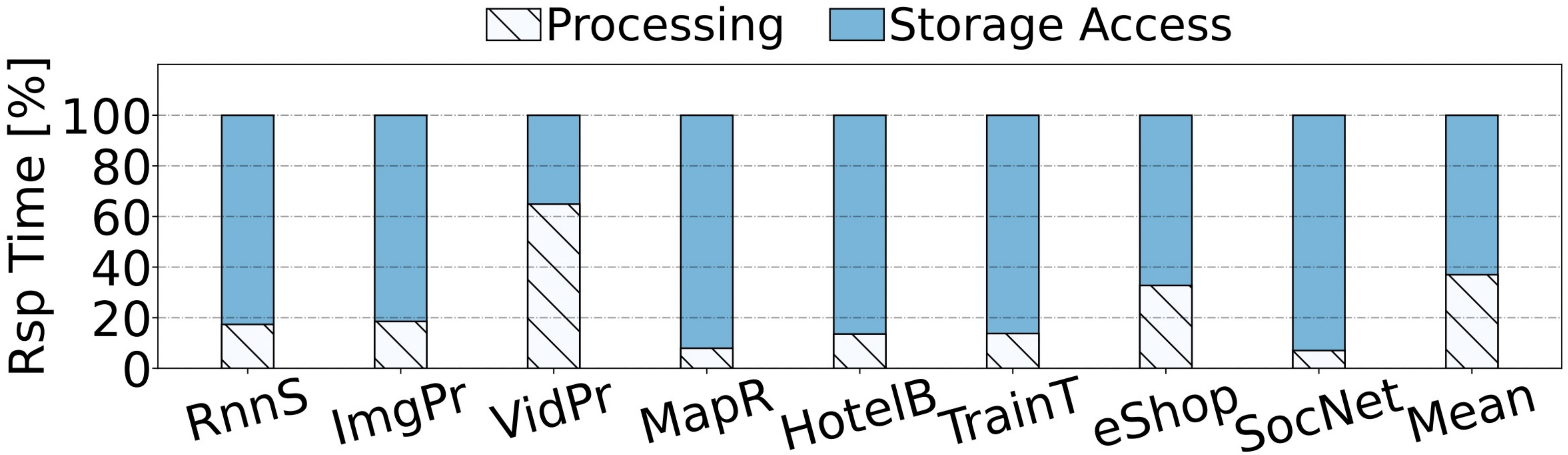
Stateless Functions Save Data in Storage



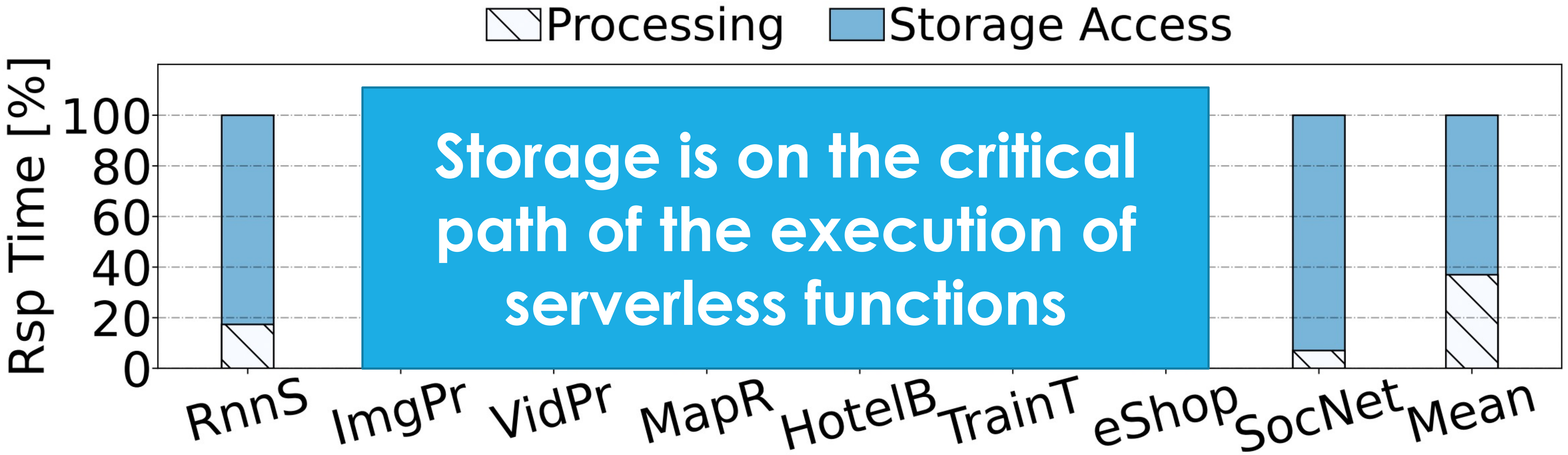
Functions Communicate via Storage



Lots of Time Spent on Storage Access

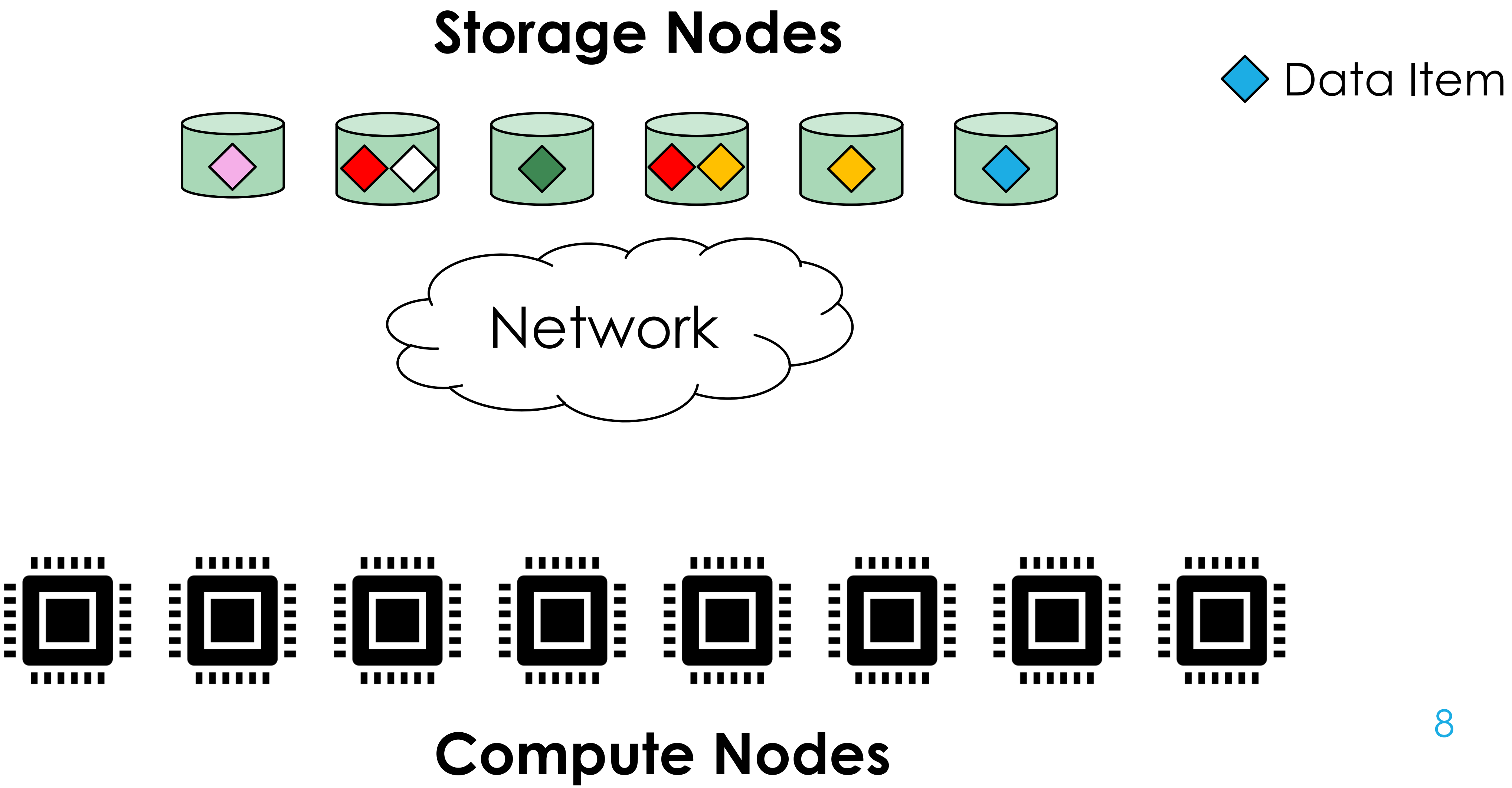


Lots of Time Spent on Storage Access



Storage is on the critical path of the execution of serverless functions

Bring the Data Closer to Functions → SW Caches



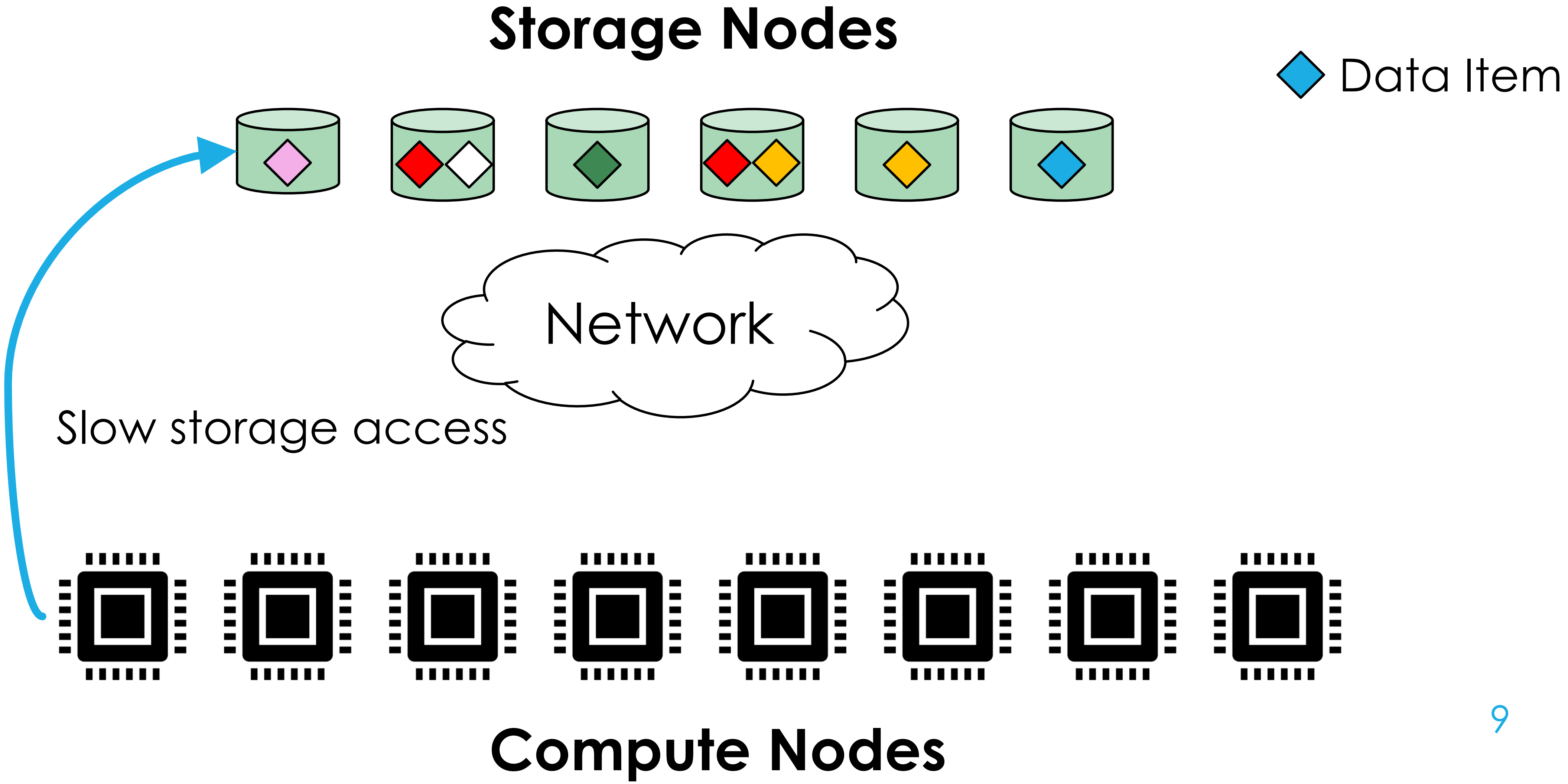
Storage Nodes

◆ Data Item

Network

Compute Nodes

Bring the Data Closer to Functions → SW Caches



Storage Nodes

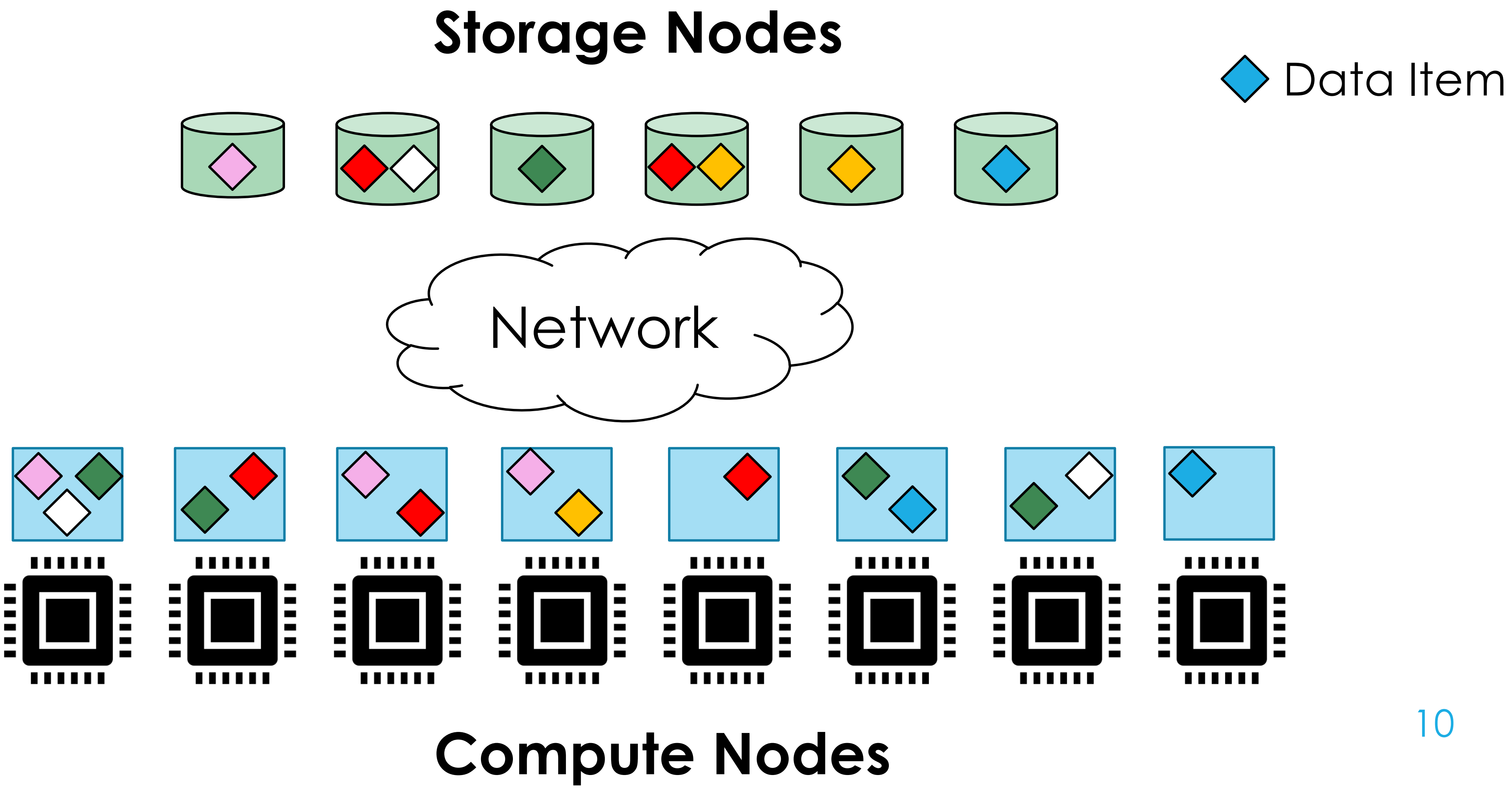
◆ Data Item

Network

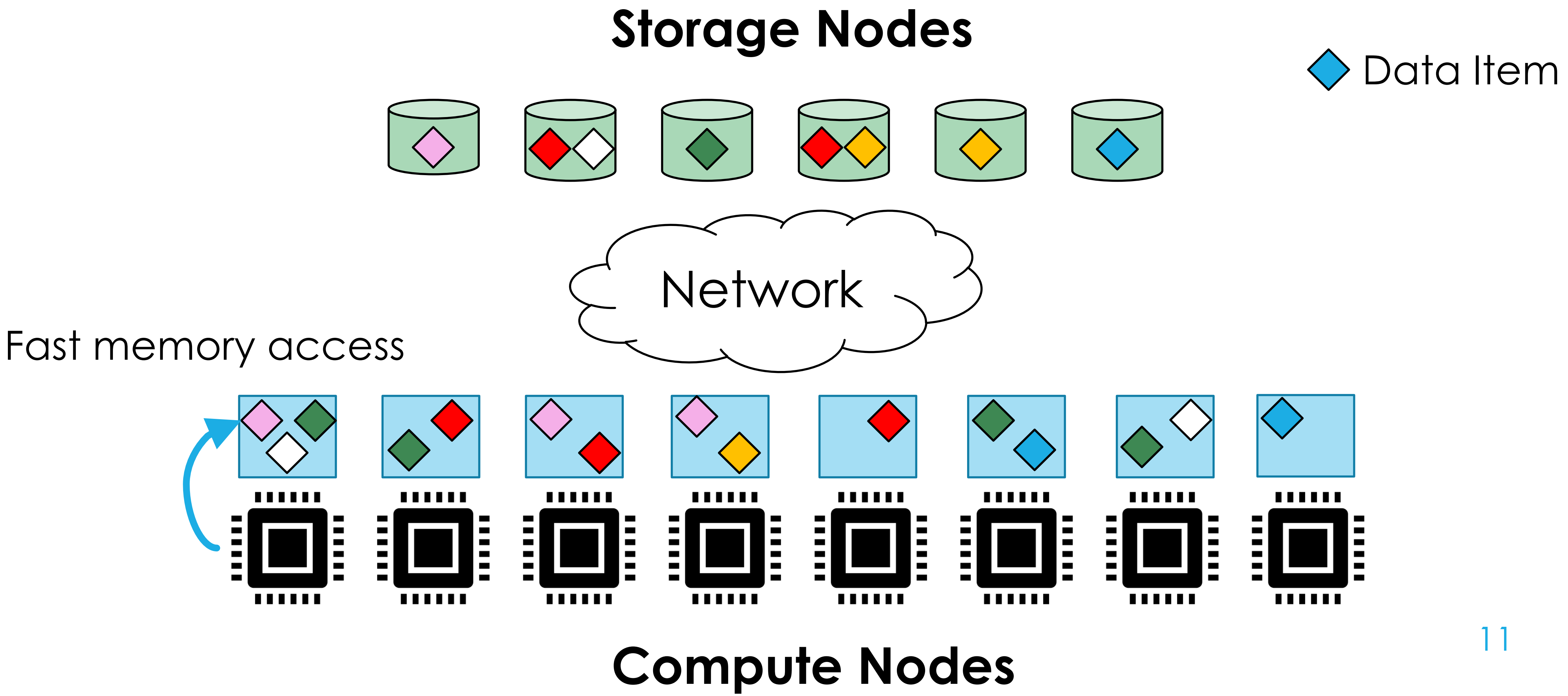
Slow storage access

Compute Nodes

Bring the Data Closer to Functions → SW Caches



Bring the Data Closer to Functions → SW Caches



Storage Nodes

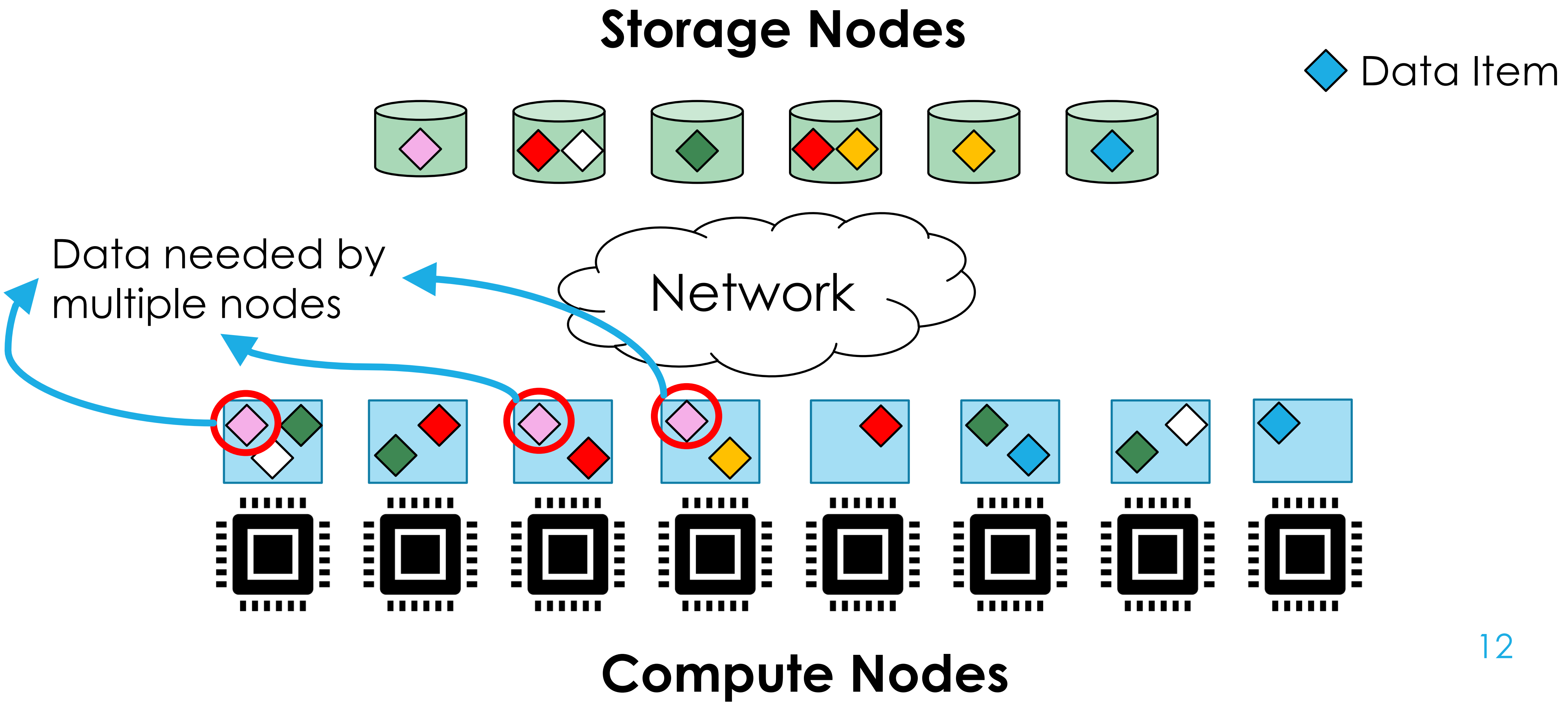
◆ Data Item

Network

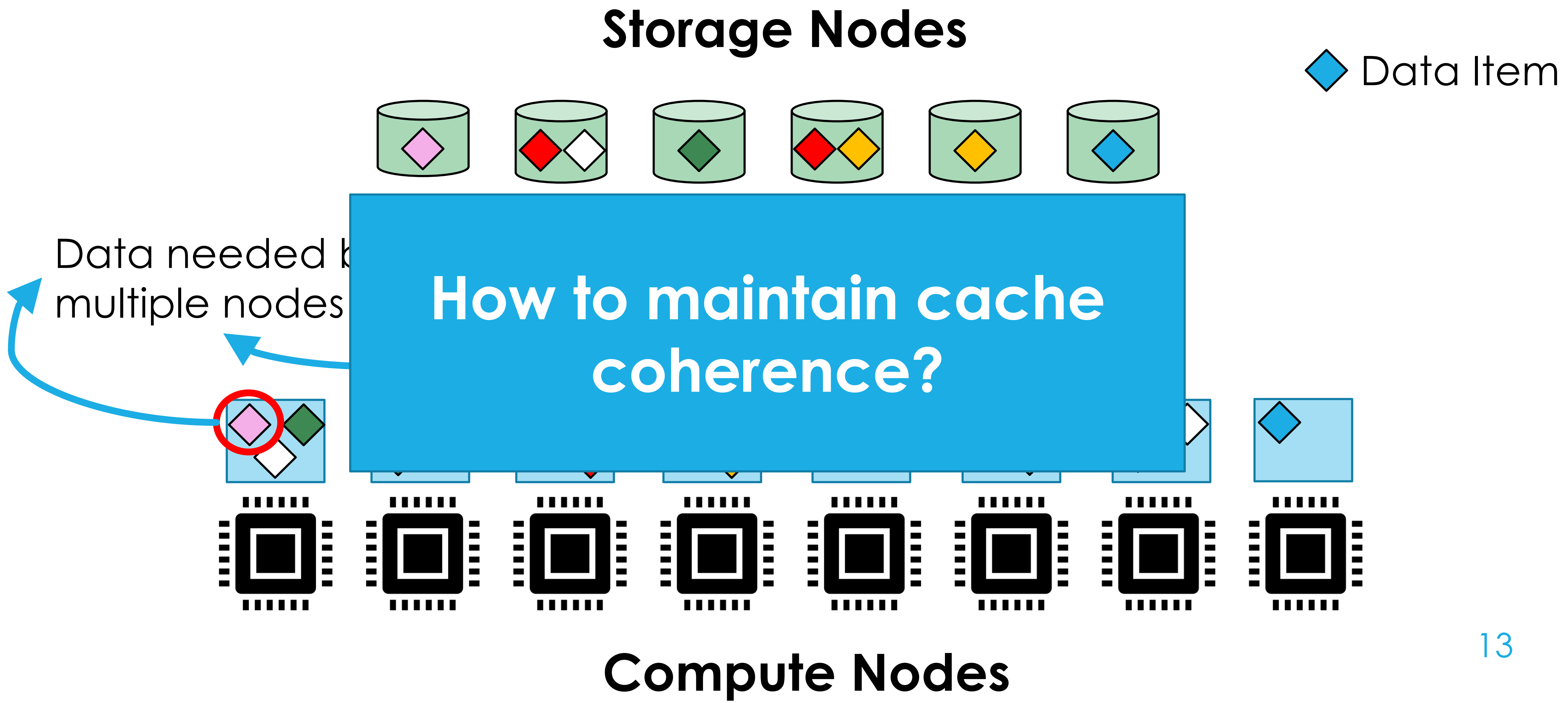
Fast memory access

Compute Nodes

Bring the Data Closer to Functions → SW Caches



Bring the Data Closer to Functions → SW Caches

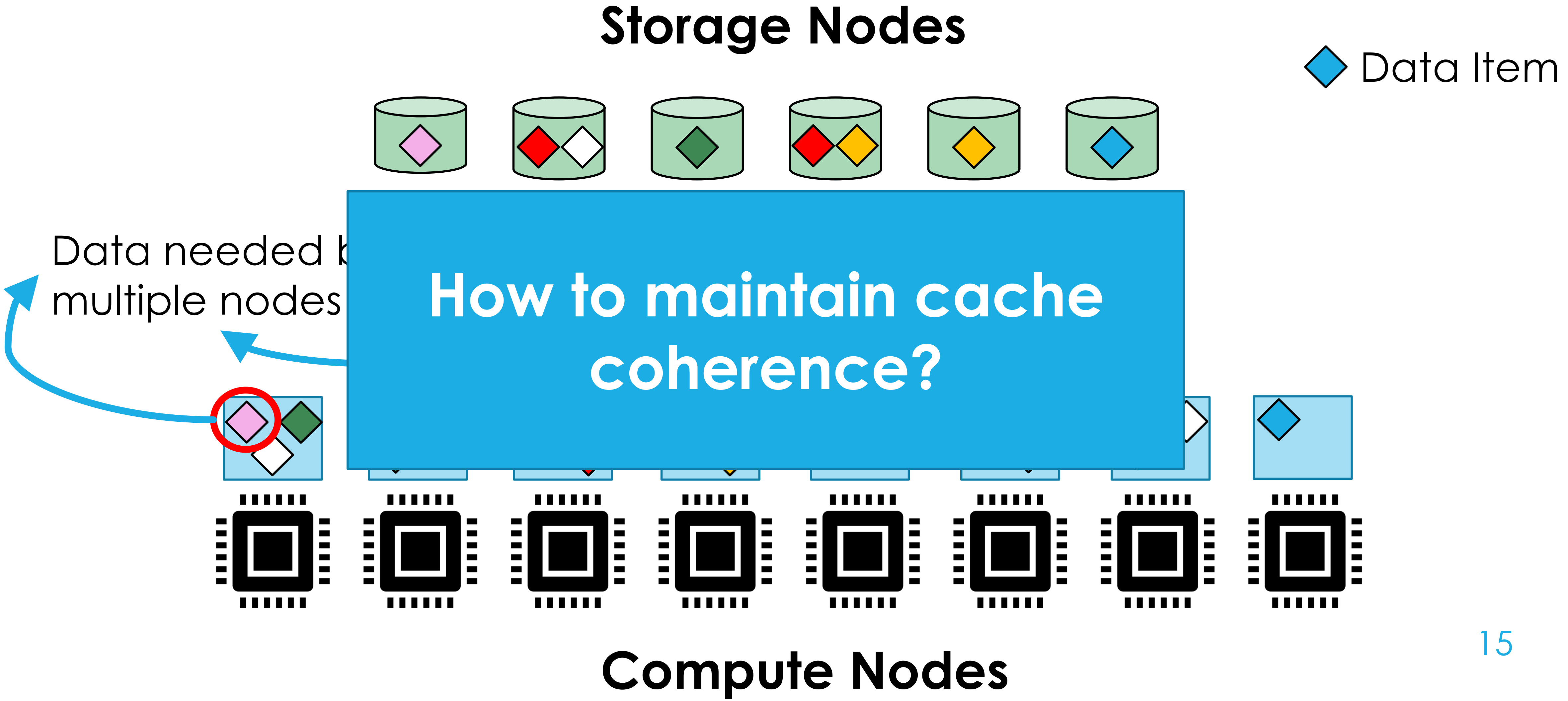


How to maintain cache coherence?

Contributions

- Characterize distributed serverless software cache designs and the design space of coherence protocols for them
- Propose **Concord**
  - High-performance and fault-tolerant distributed directory-based coherence protocol for software caches
- Achieves speedup of 2.4x and higher throughput by 1.7x over state-of-the art serverless schemes

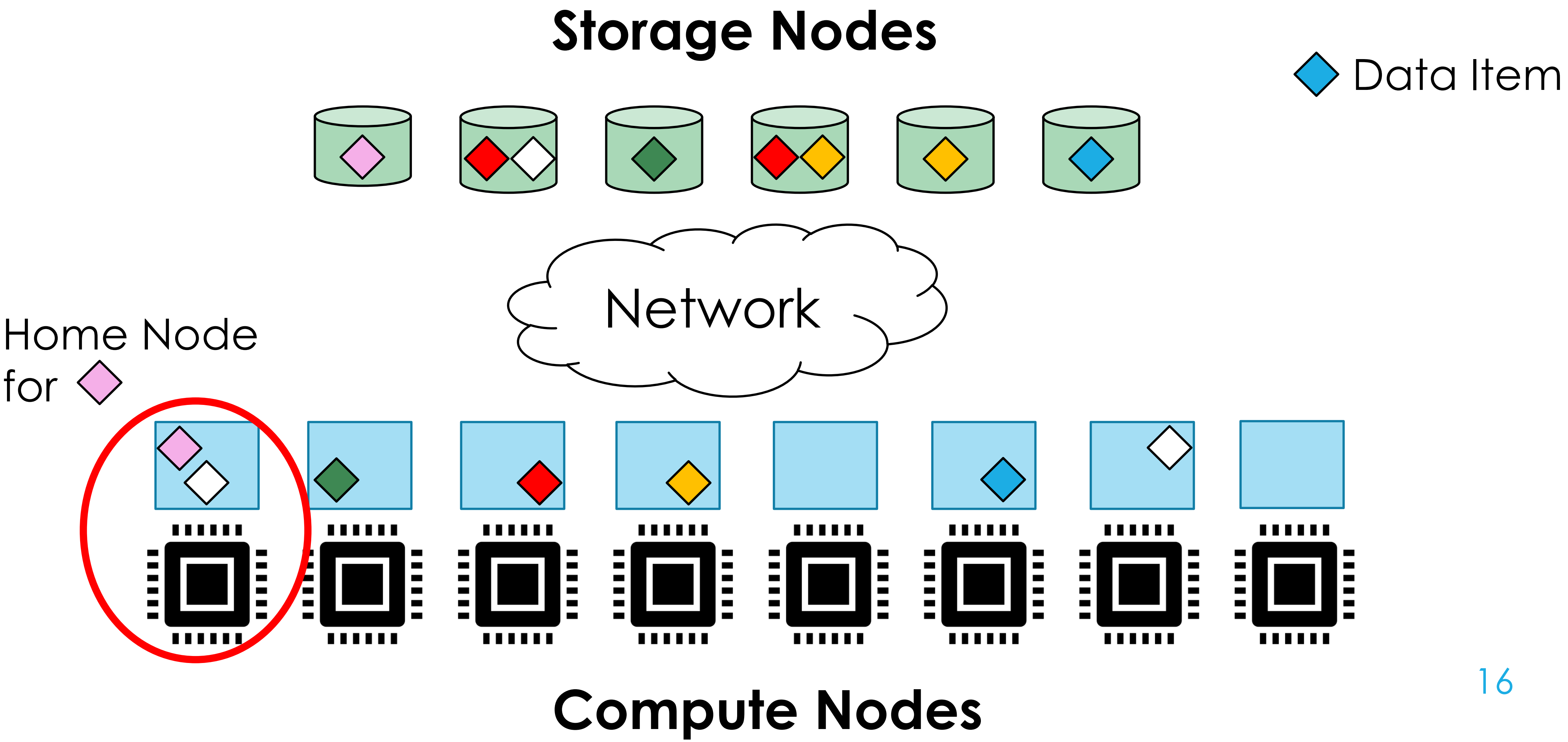
Bring the Data Closer to Functions → SW Caches



How to maintain cache coherence?

Option #1: Data Cached Only in Home

[OFC EuroSys'21]



Storage Nodes

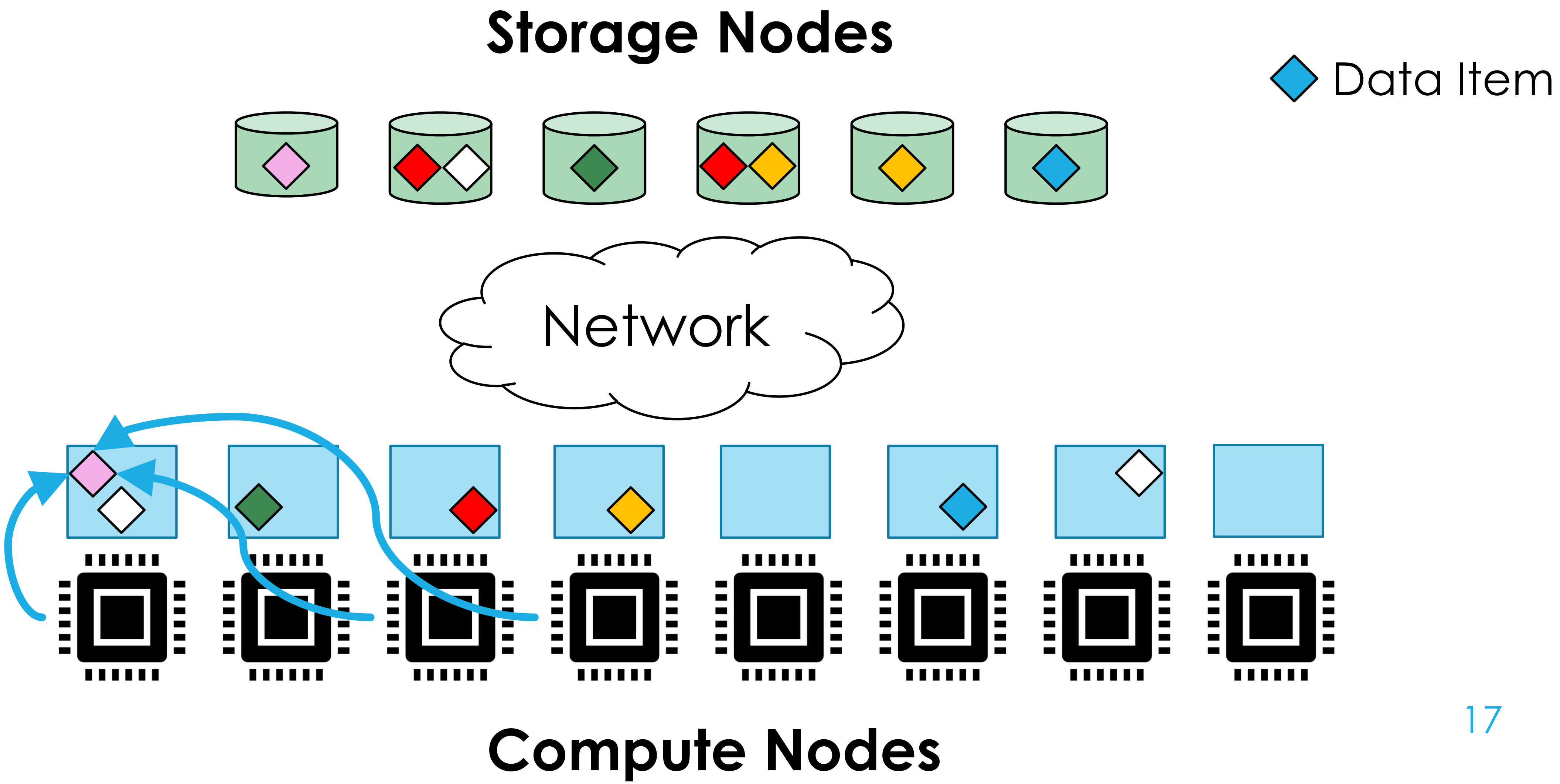
◆ Data Item

Network

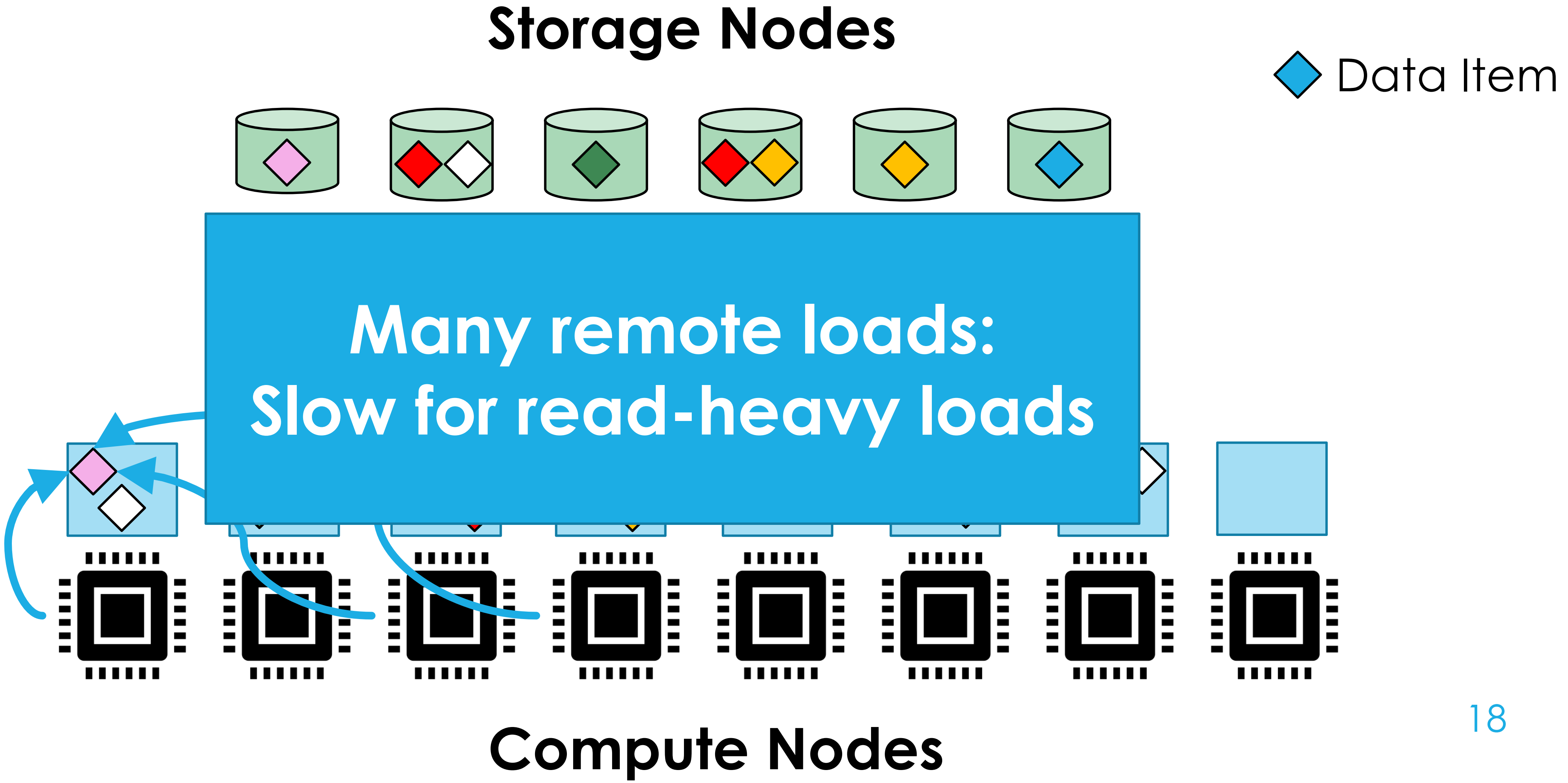
Home Node for ◇

Compute Nodes

Option #1: Data Cached Only in Home



Option #1: Data Cached Only in Home



Storage Nodes

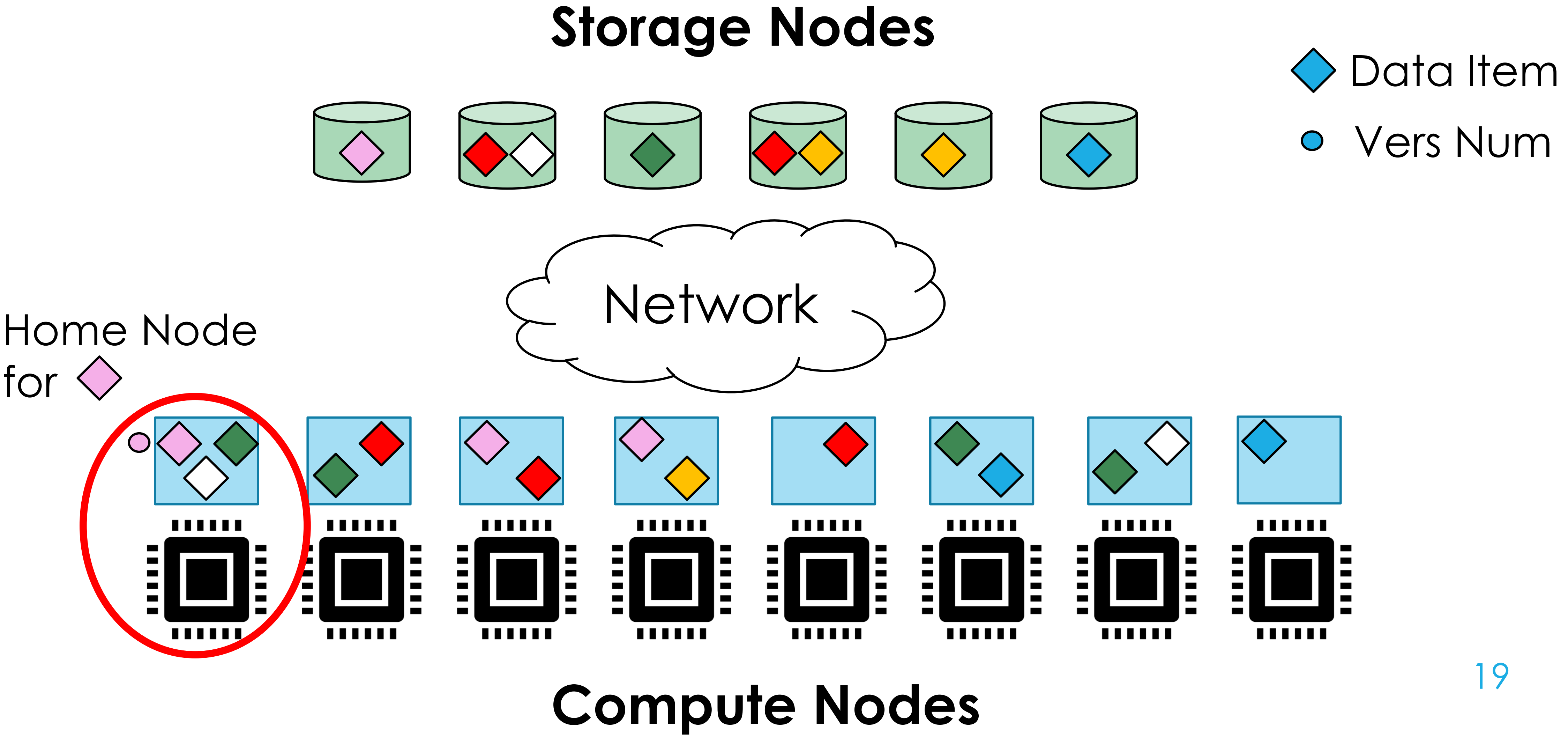
◆ Data Item

Many remote loads:
Slow for read-heavy loads

Compute Nodes

Option #2: Version Numbers

[Faa$T SoCC'21]



Storage Nodes

◆ Data Item

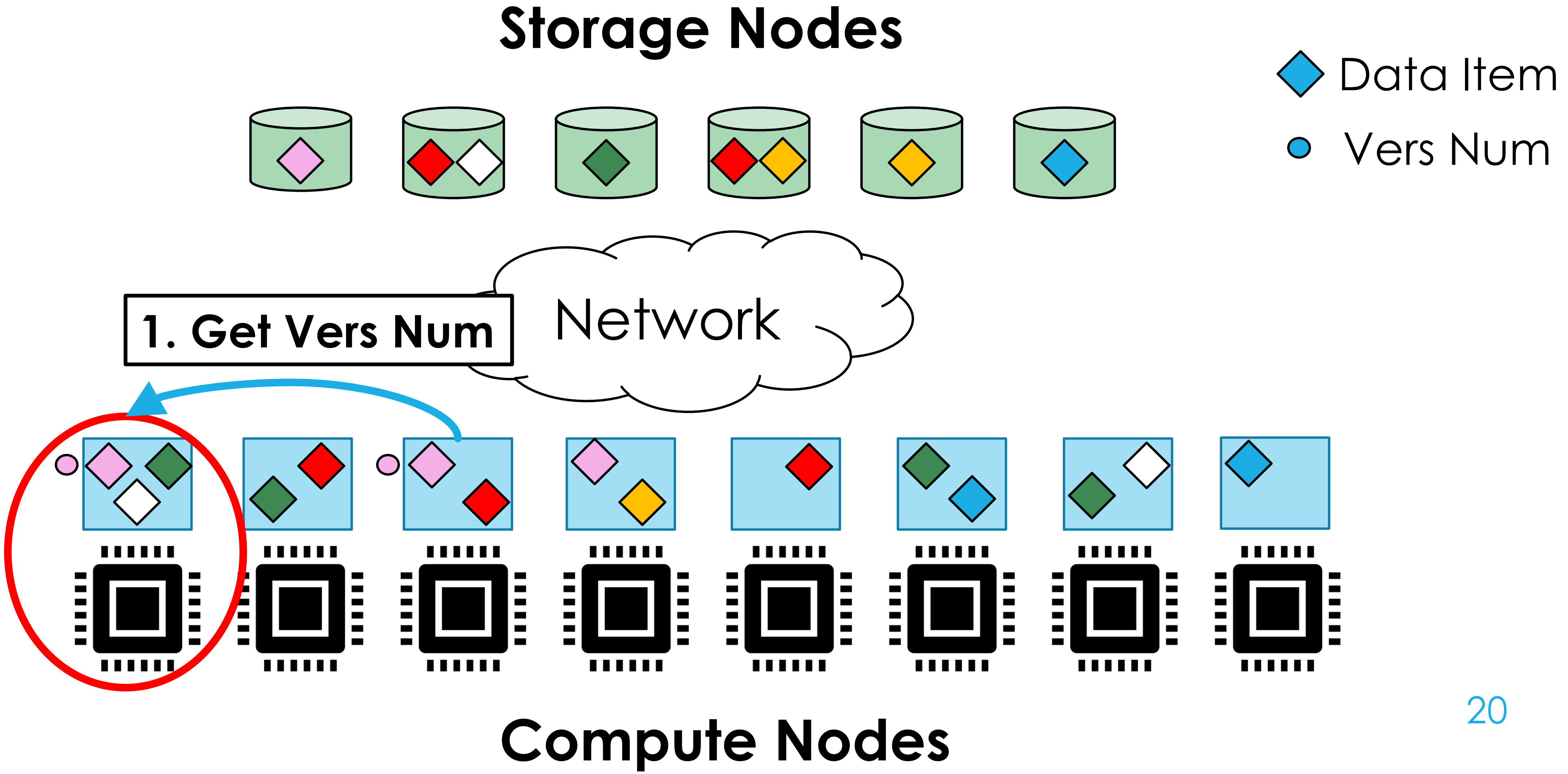
● Vers Num

Network

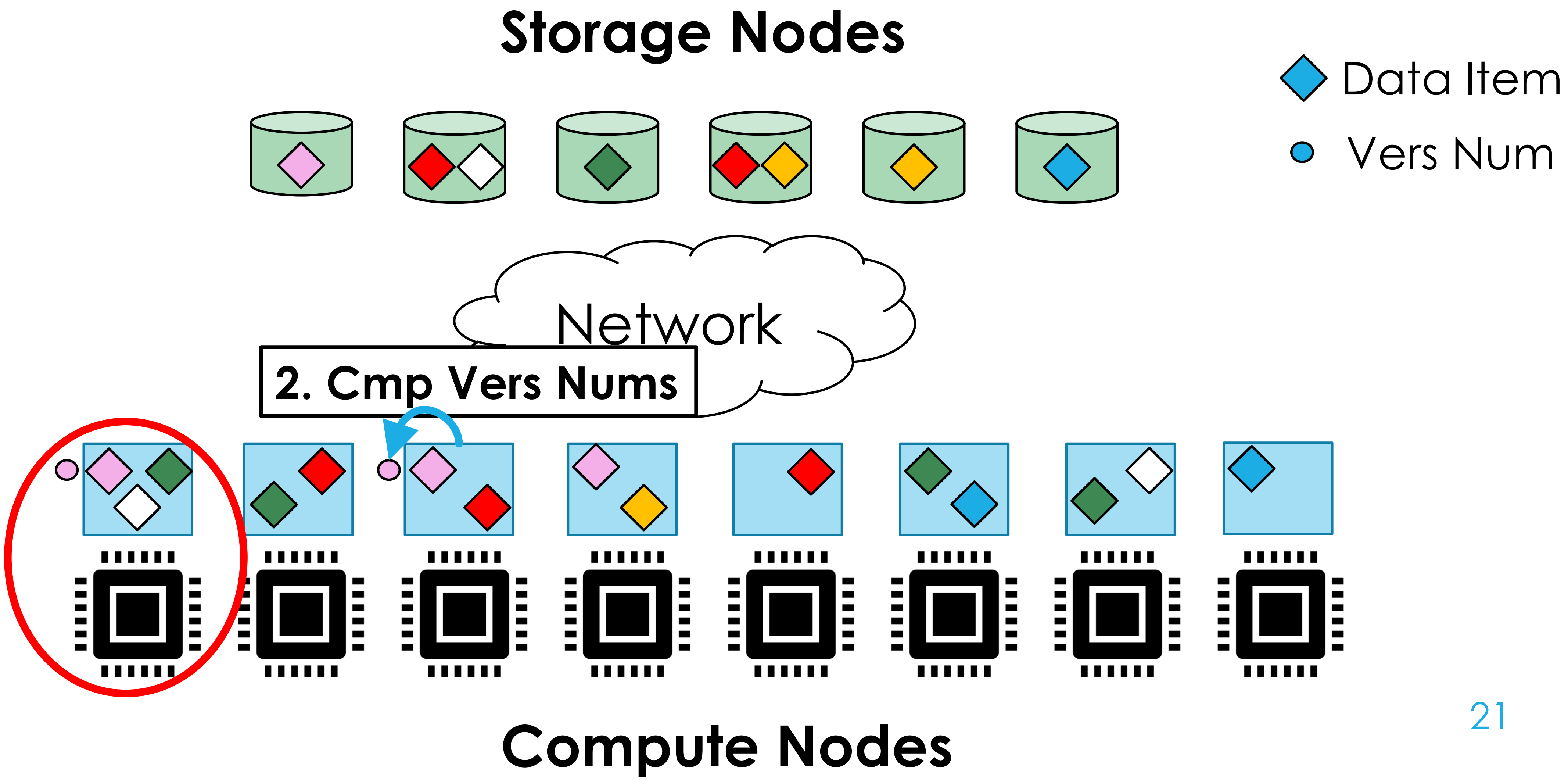
Home Node for ◇

Compute Nodes

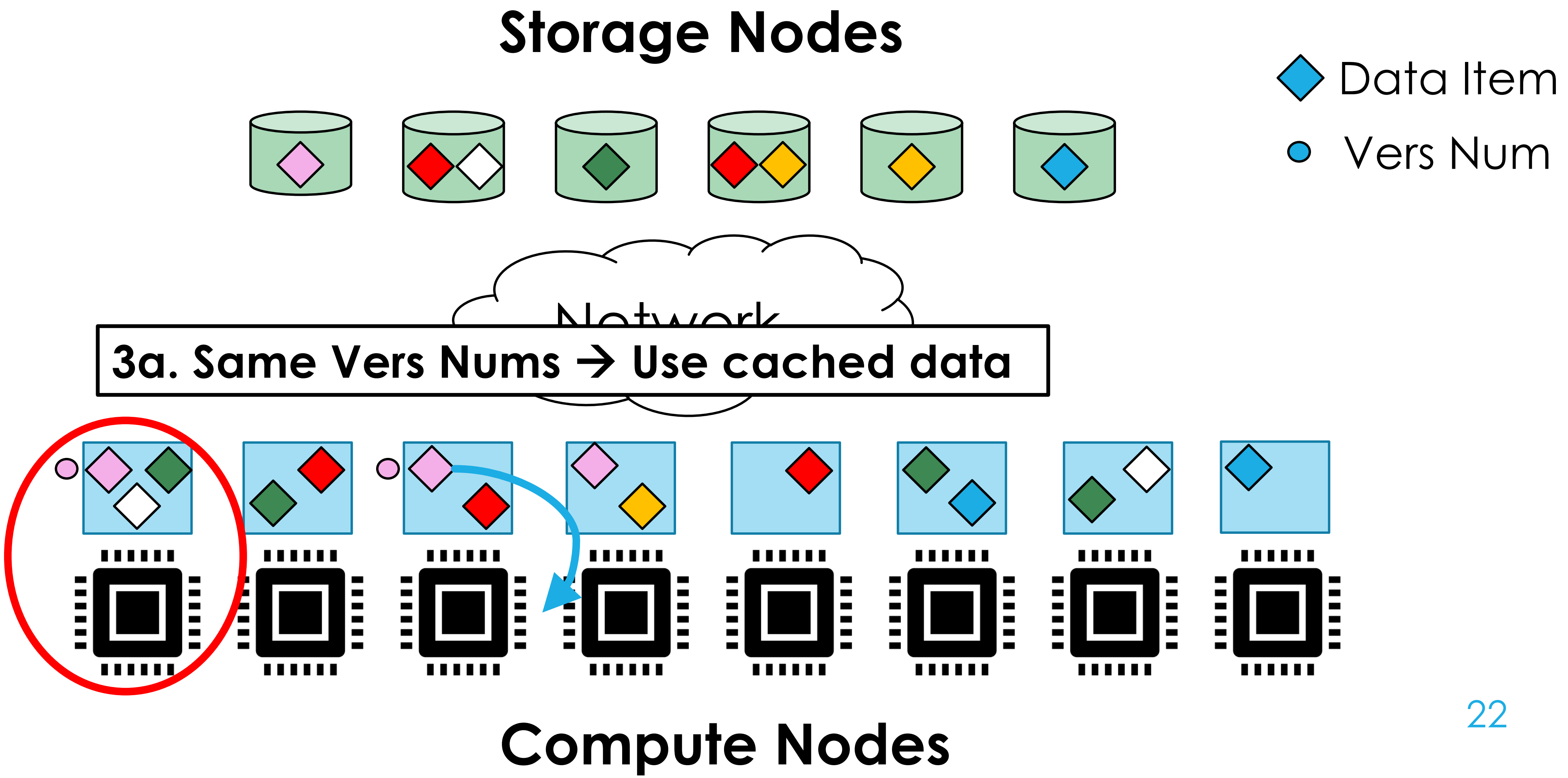
Option #2: Version Numbers



Option #2: Version Numbers

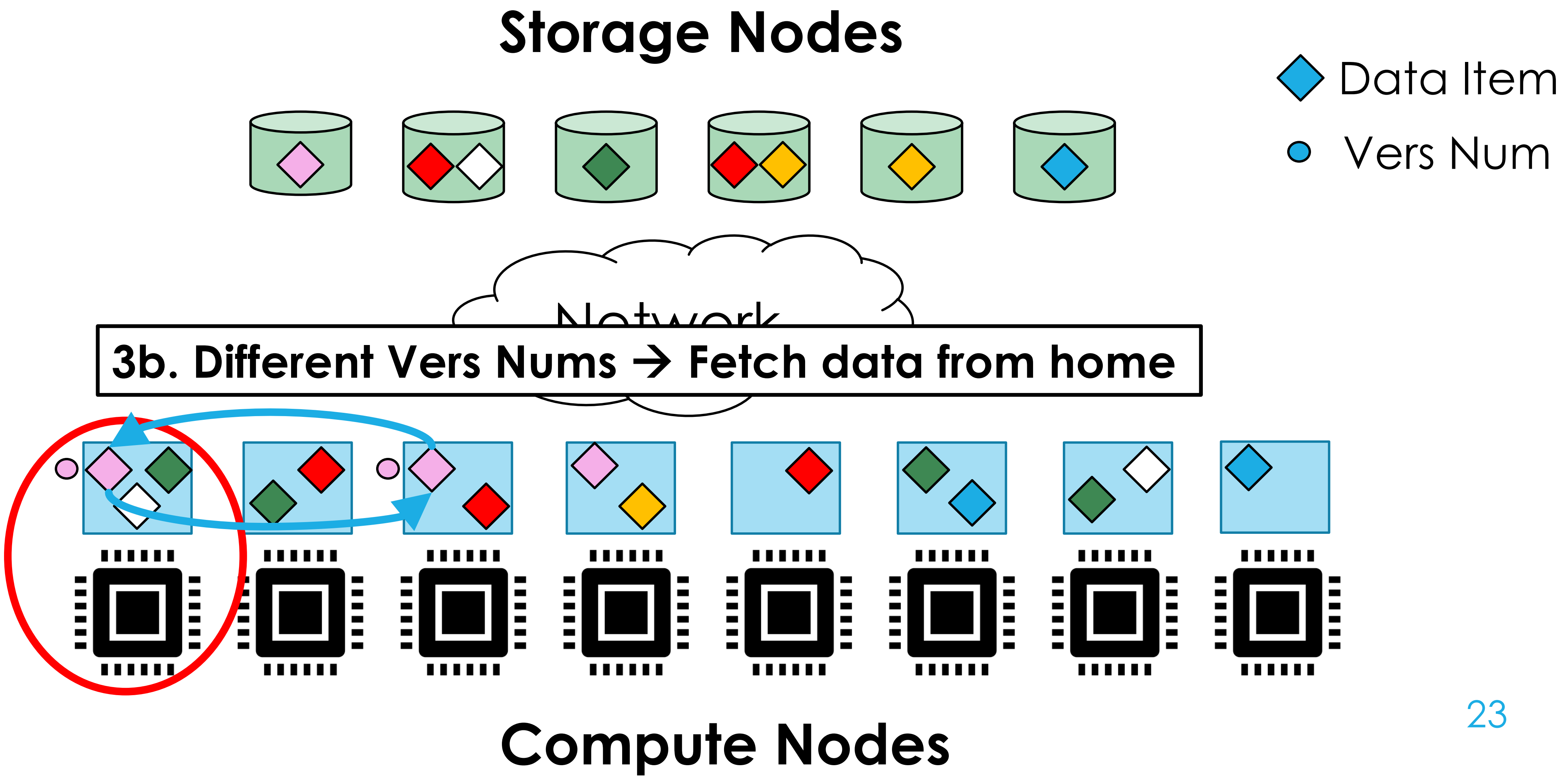


Option #2: Version Numbers



3a. Same Vers Nums → Use cached data

Option #2: Version Numbers



Storage Nodes

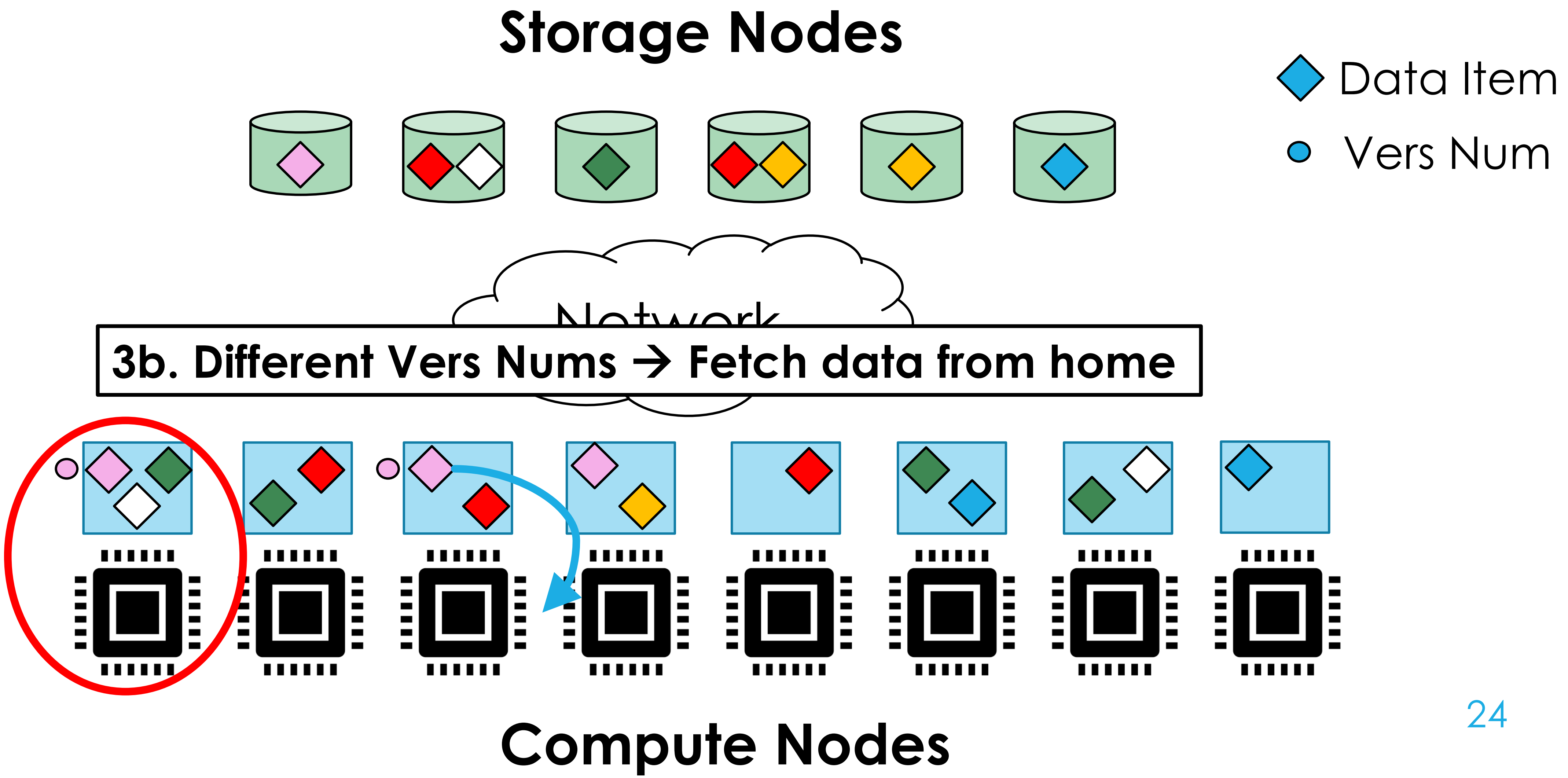
Data Item

Vers Num

3b. Different Vers Nums → Fetch data from home

Compute Nodes

Option #2: Version Numbers



Storage Nodes

◆ Data Item

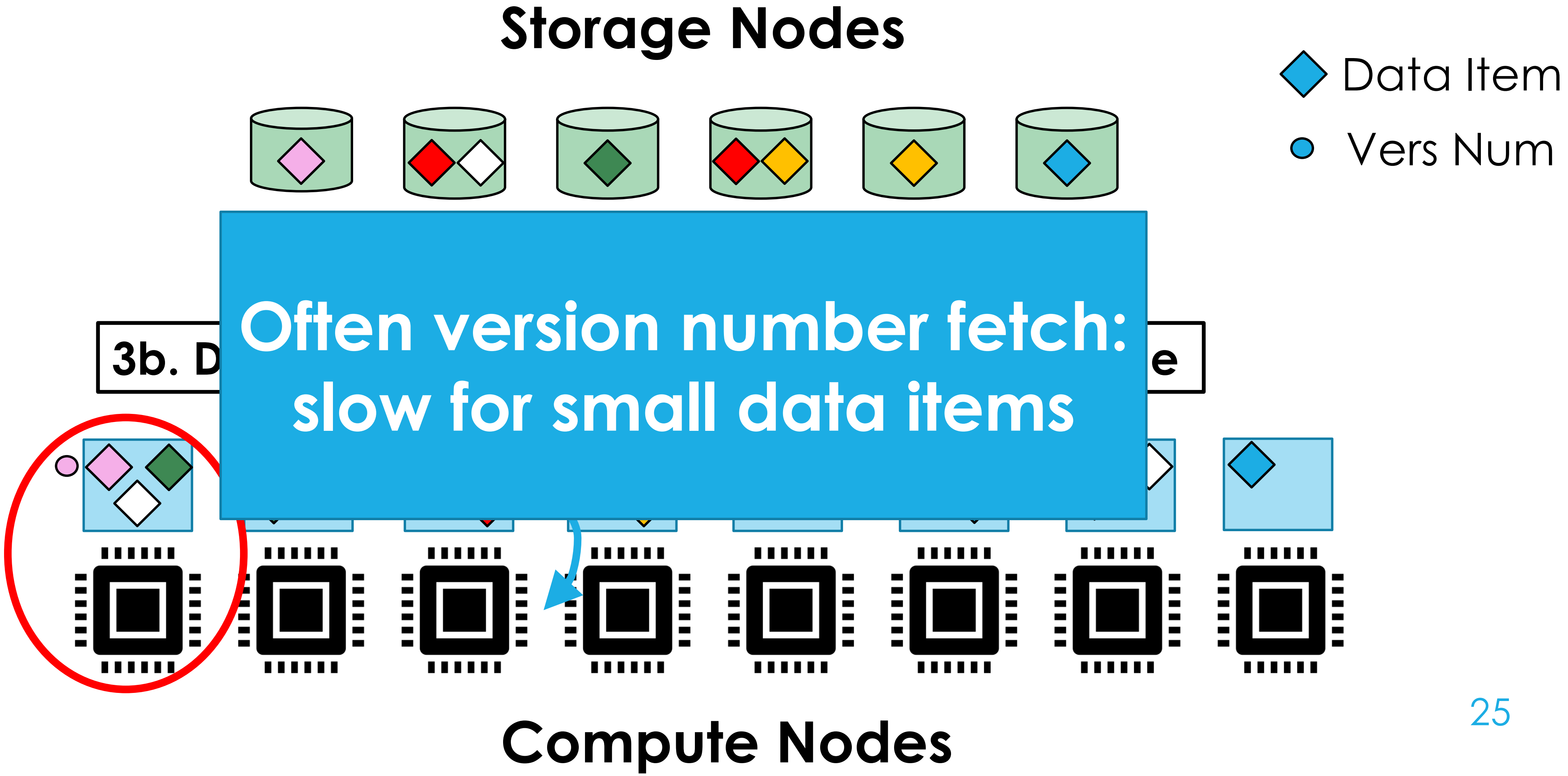
● Vers Num

Network

3b. Different Vers Nums → Fetch data from home

Compute Nodes

Option #2: Version Numbers



Often version number fetch: slow for small data items

Existing Protocols → High Overheads

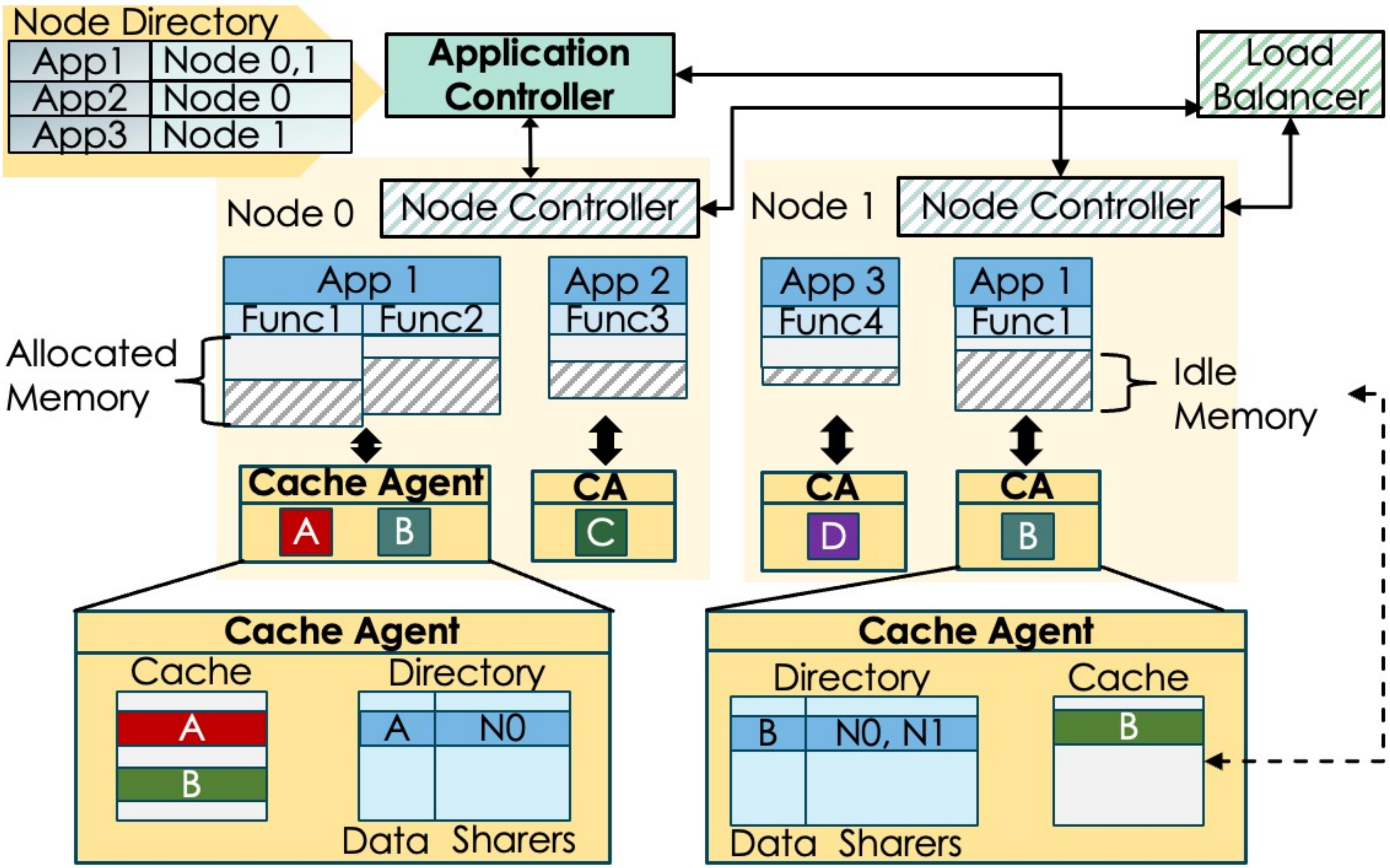
- Data items small → 80% in production workloads less than 12KB
- Read operations dominate → 77% in production workloads are reads

Existing Protocols → High Overheads

- Data items small → 80% in production workloads less than 12KB
- Read operations dominate → 77% in production workloads are reads

→ **Existing cache coherence protocols inefficient**

Concord: Distributed Caching Scheme for FaaS



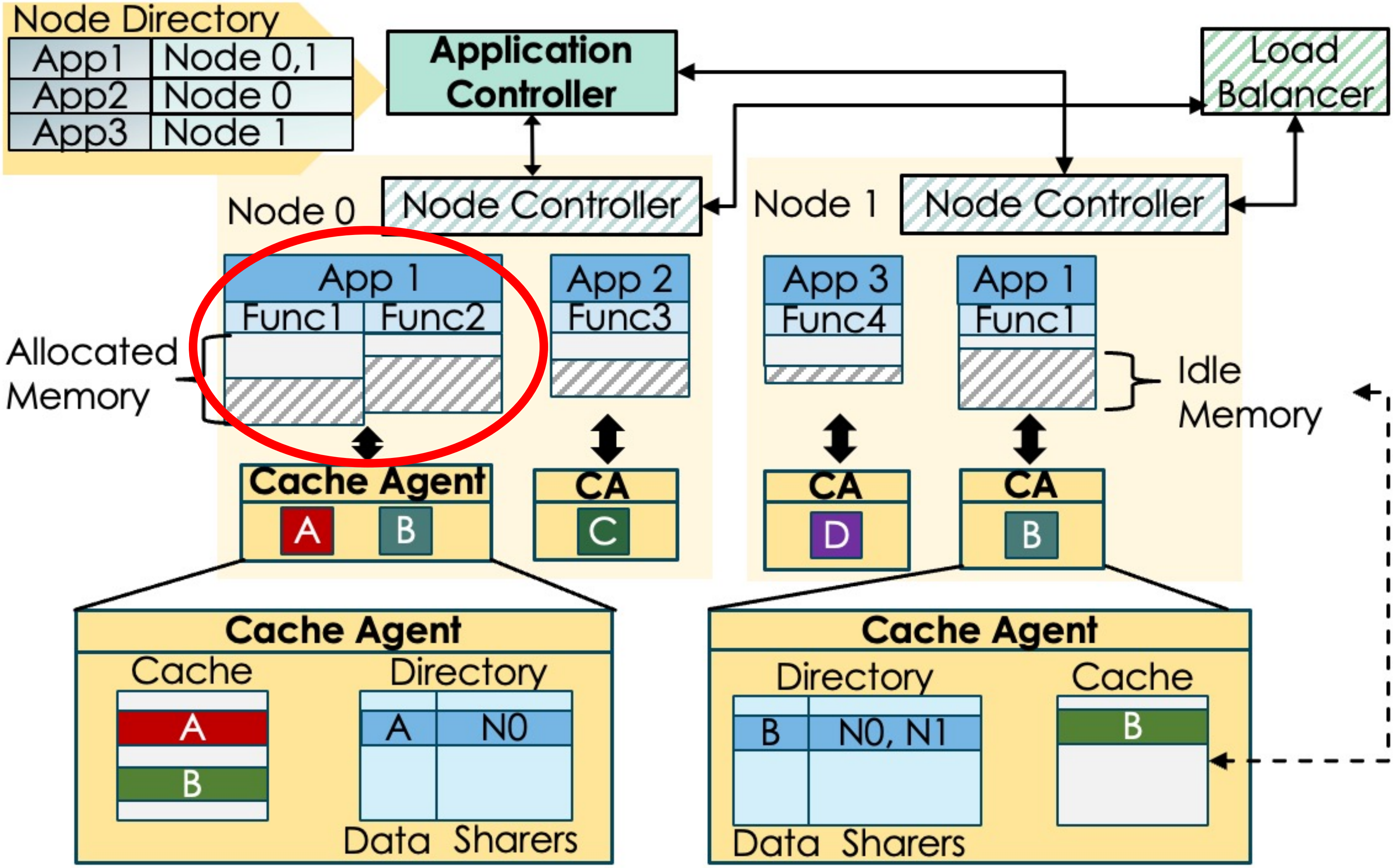
Node Directory

App1	Node 0,1
App2	Node 0
App3	Node 1

Concord Key Ideas

1. **Allocated but unused per allocation memory → app's cache**

Re-purpose Idle Memory into Cache



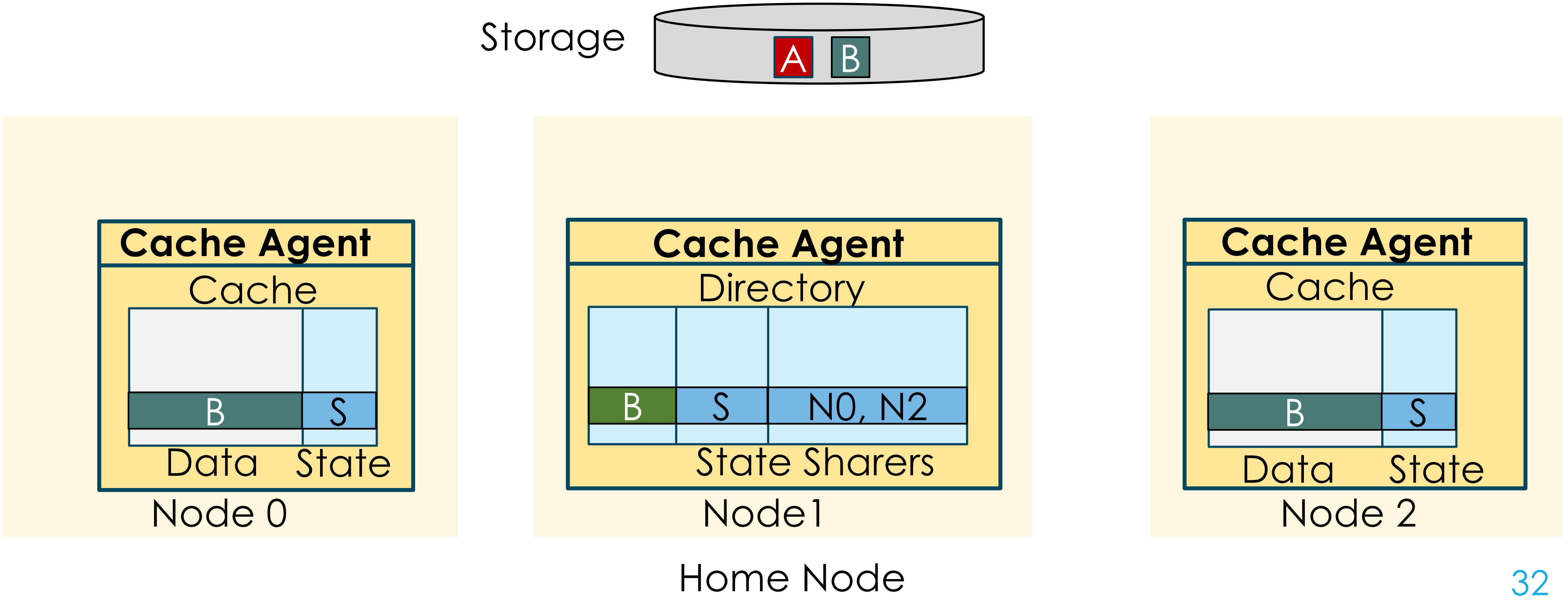
Node Directory

App1	Node 0,1
App2	Node 0
App3	Node 1

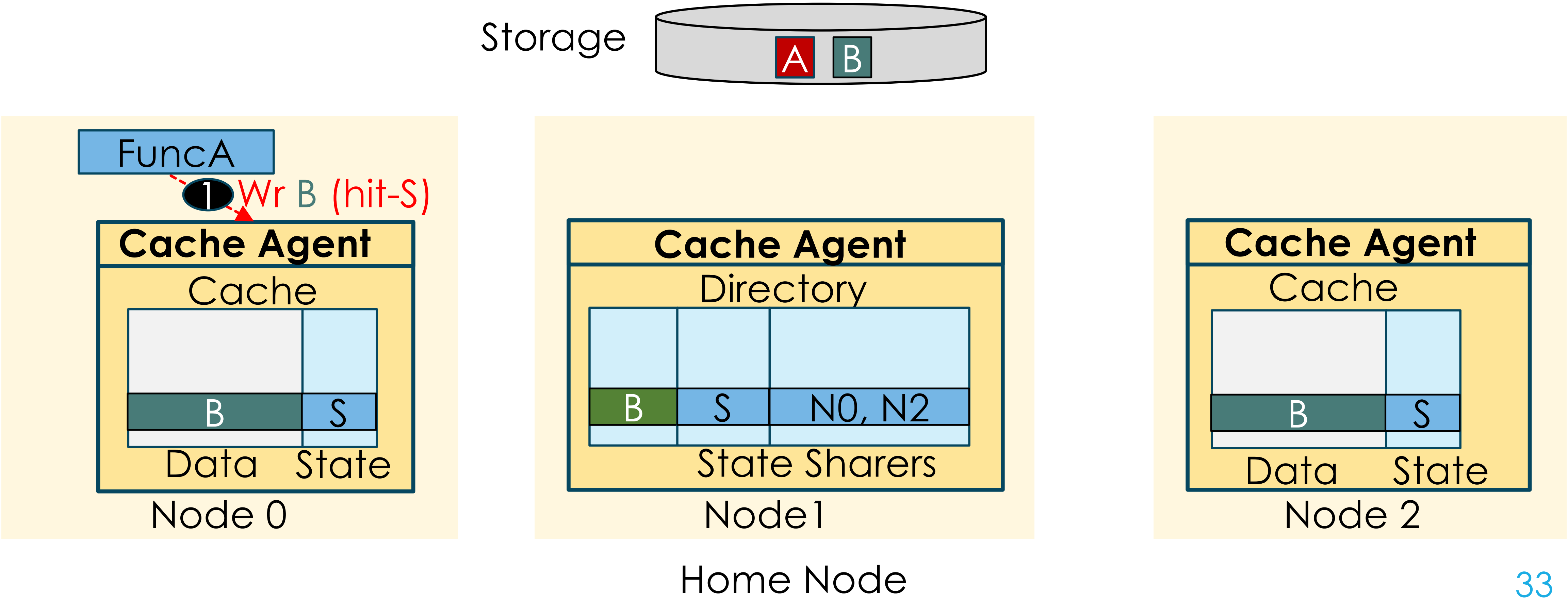
Concord Key Ideas

1. Allocated but unused per allocation memory → app's cache
2. **Directory-based invalidation protocol for cache coherence**

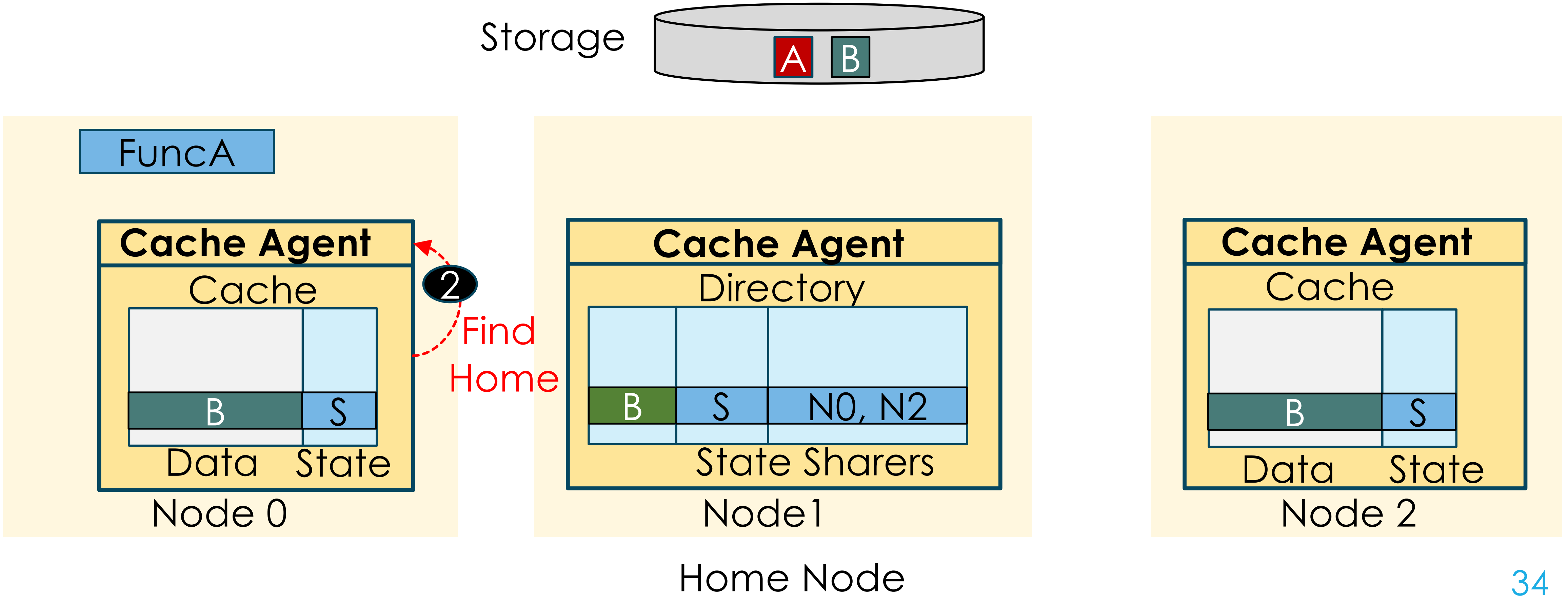
Directory-Based Invalidation Protocol



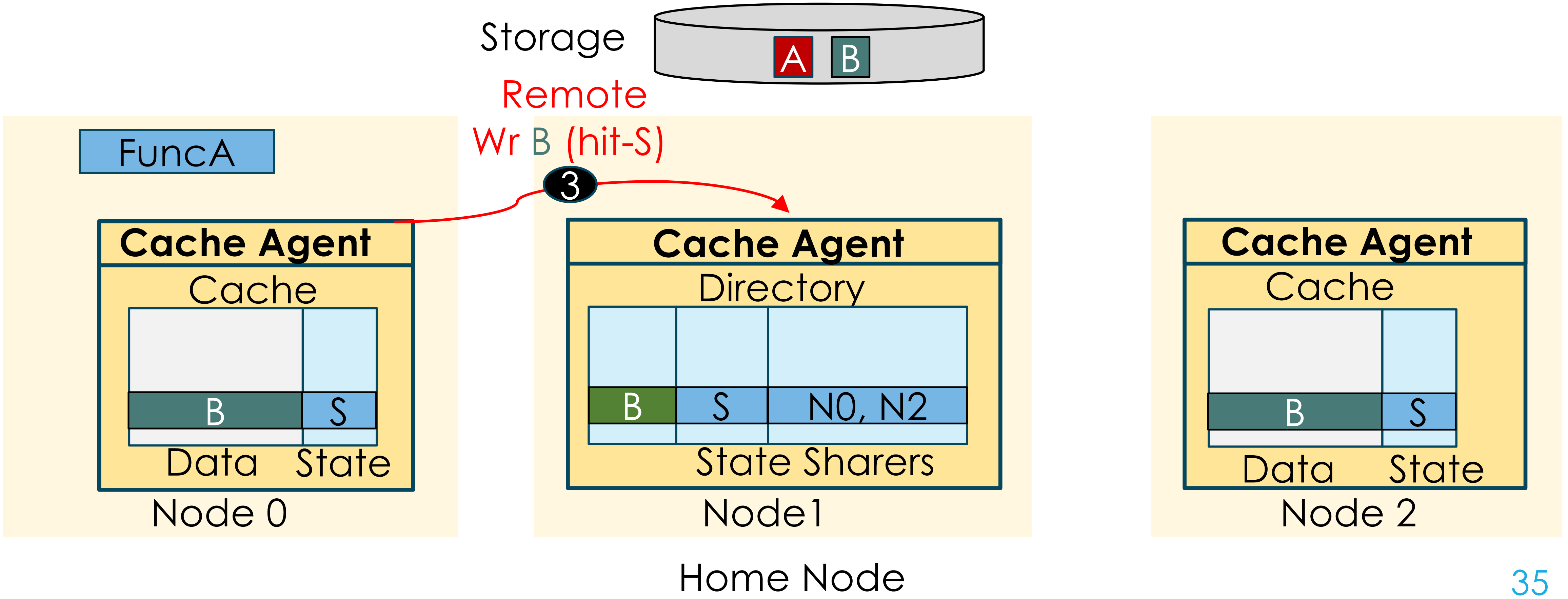
Directory-Based Invalidation Protocol



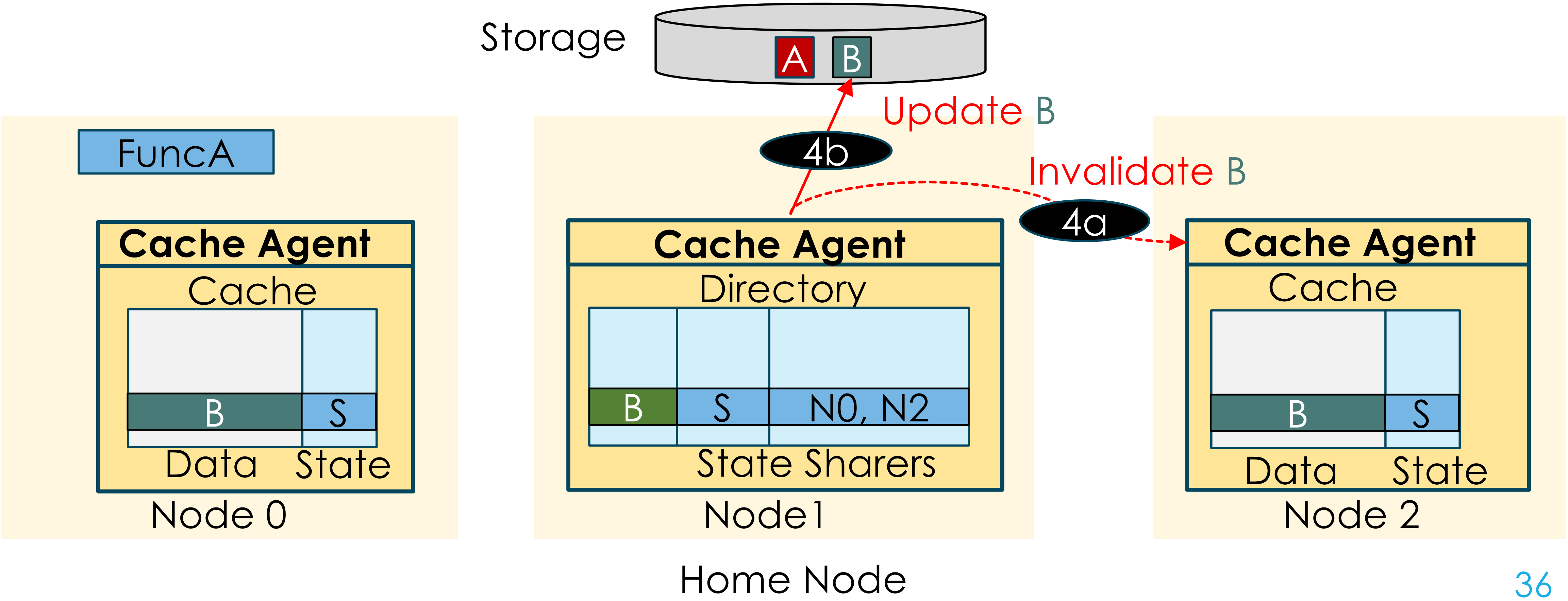
Directory-Based Invalidation Protocol



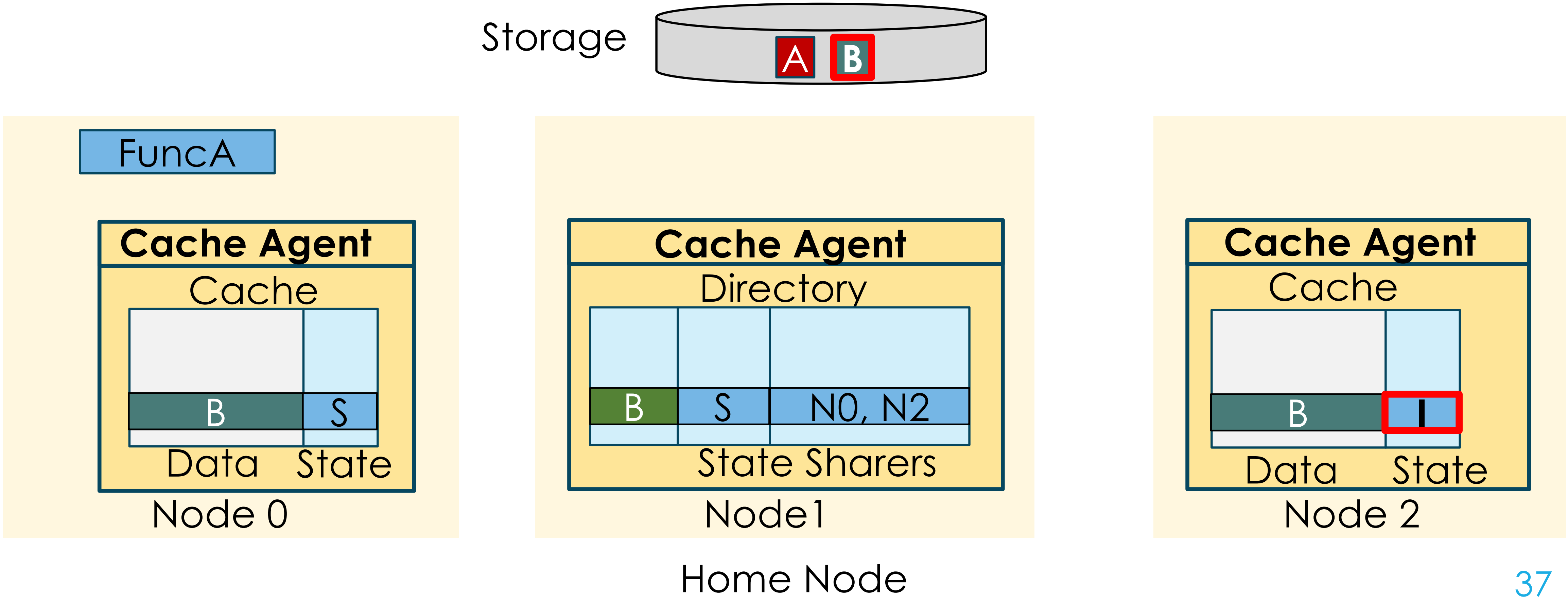
Directory-Based Invalidation Protocol



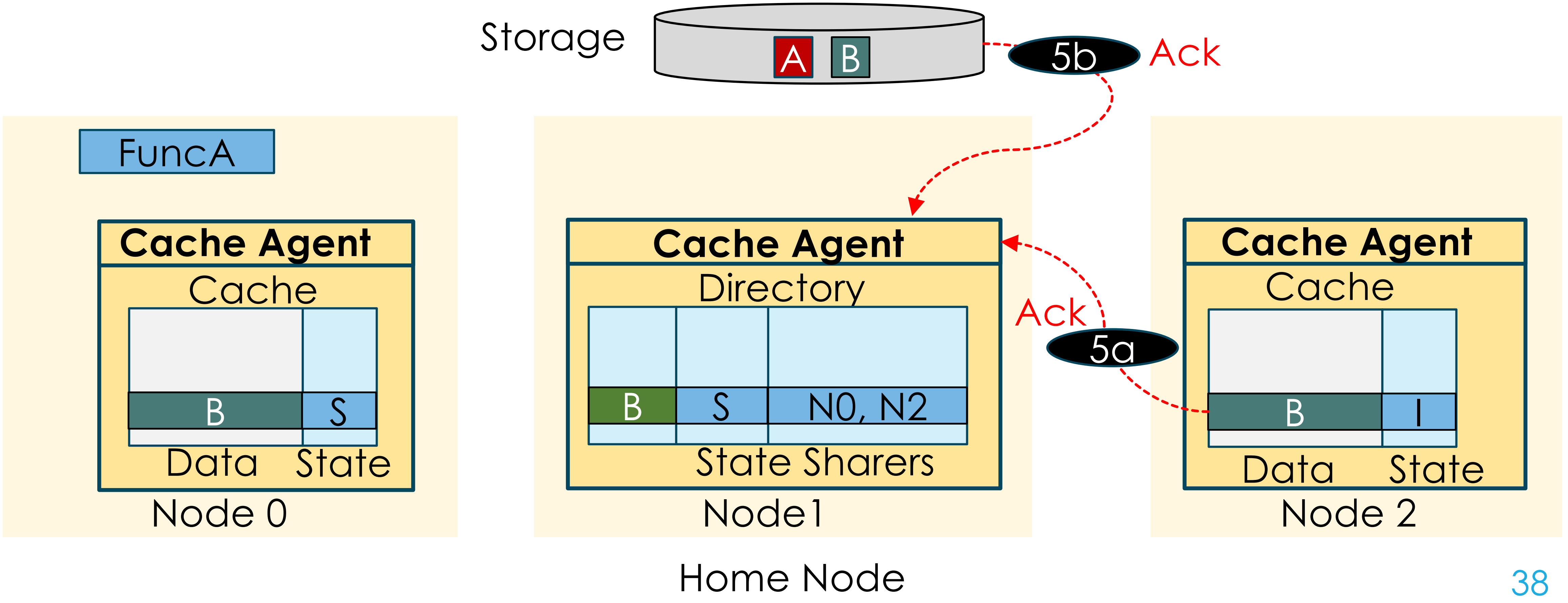
Directory-Based Invalidation Protocol



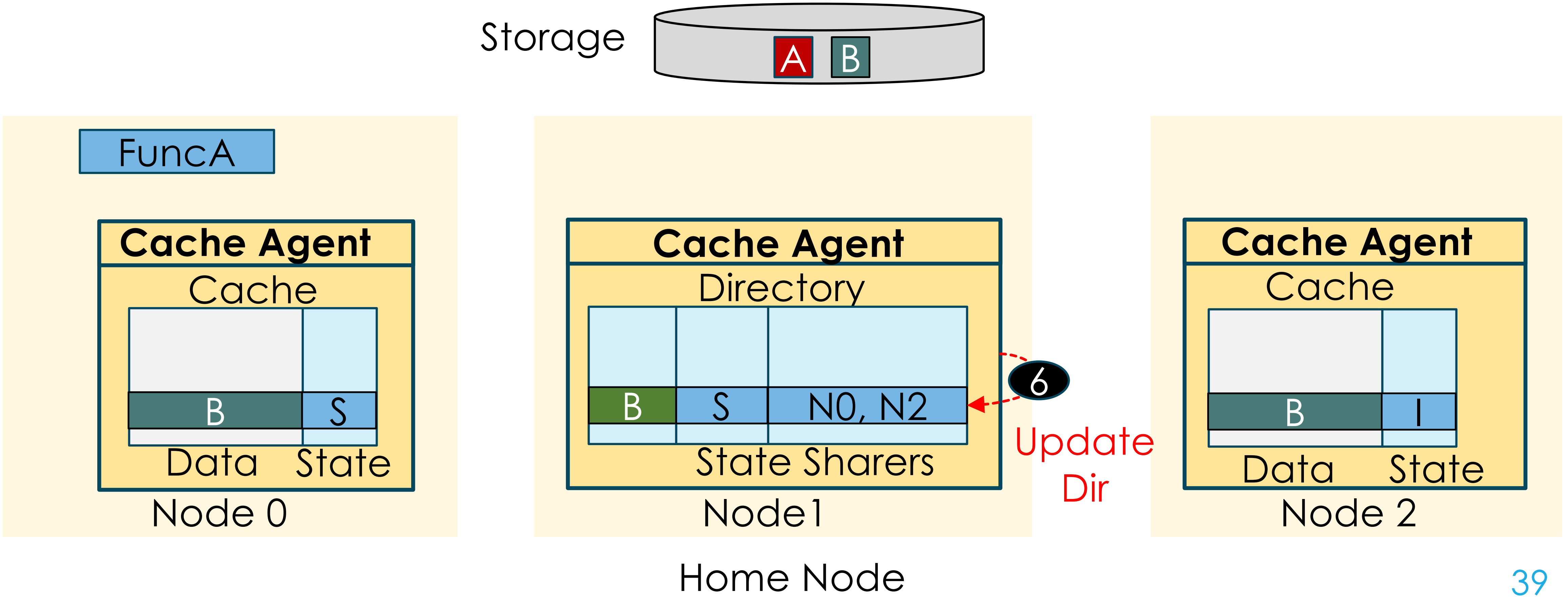
Directory-Based Invalidation Protocol



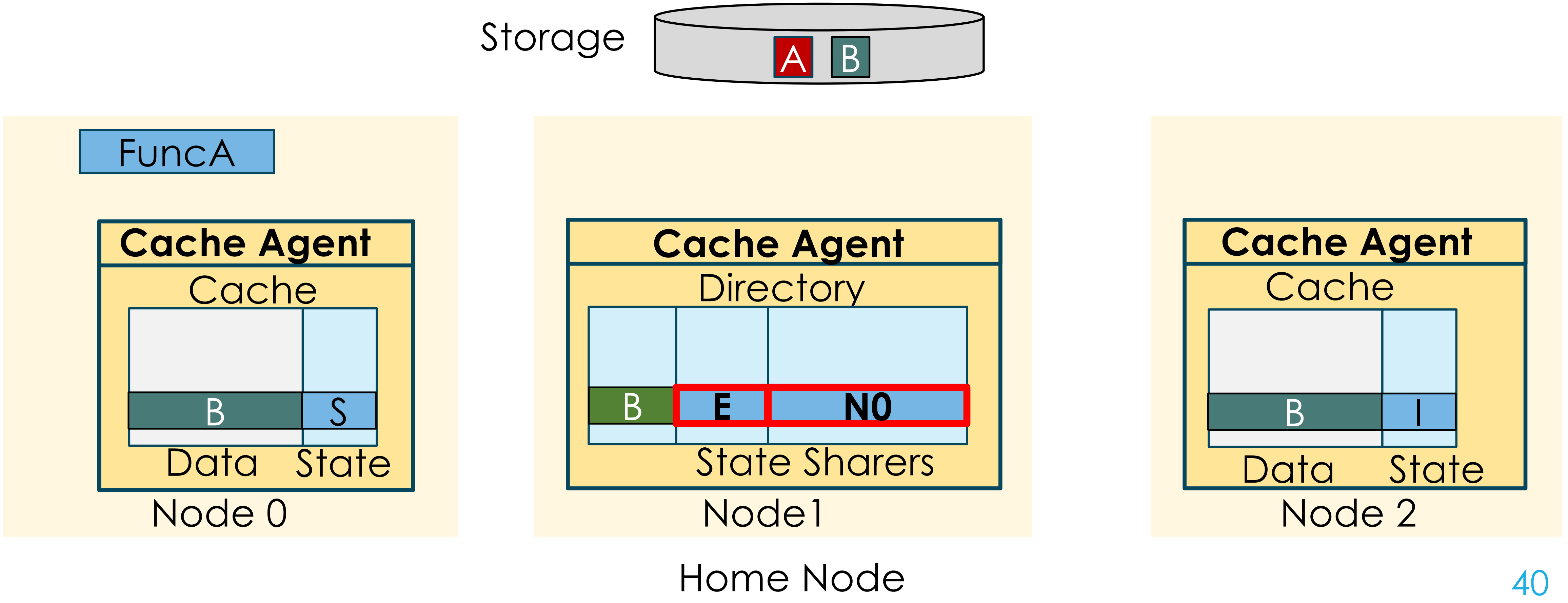
Directory-Based Invalidation Protocol



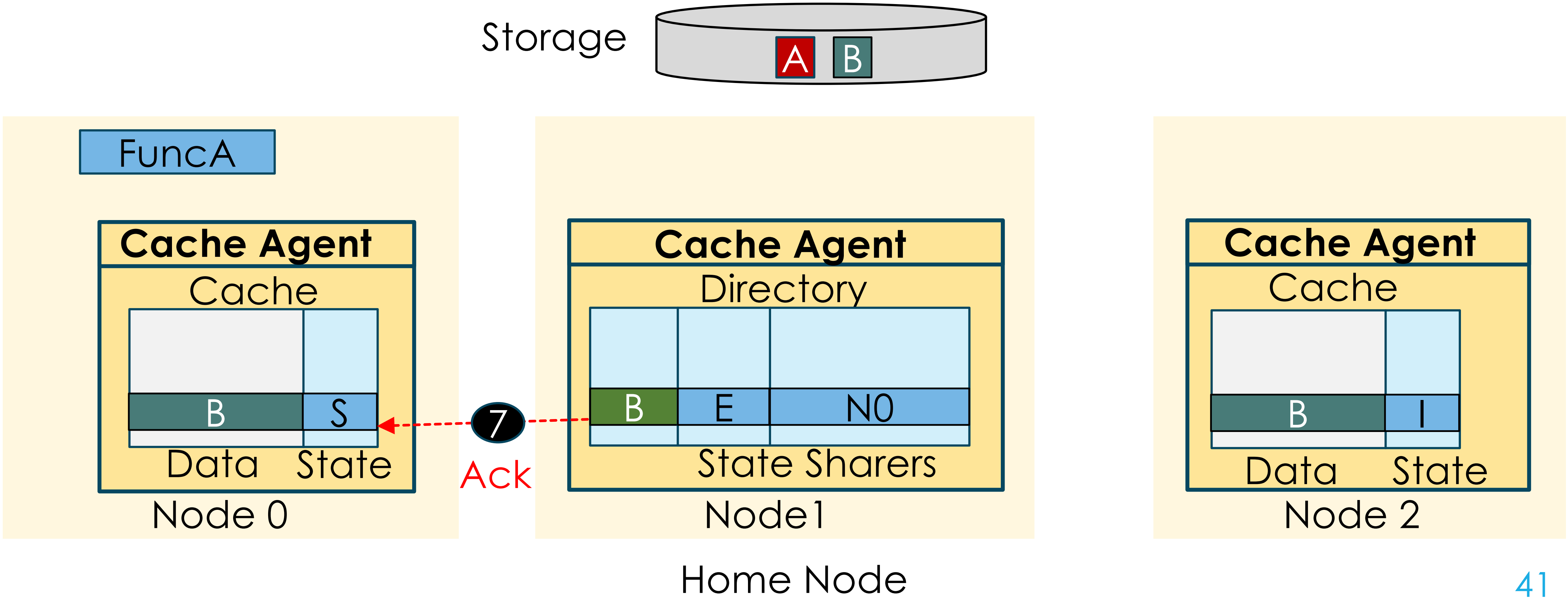
Directory-Based Invalidation Protocol



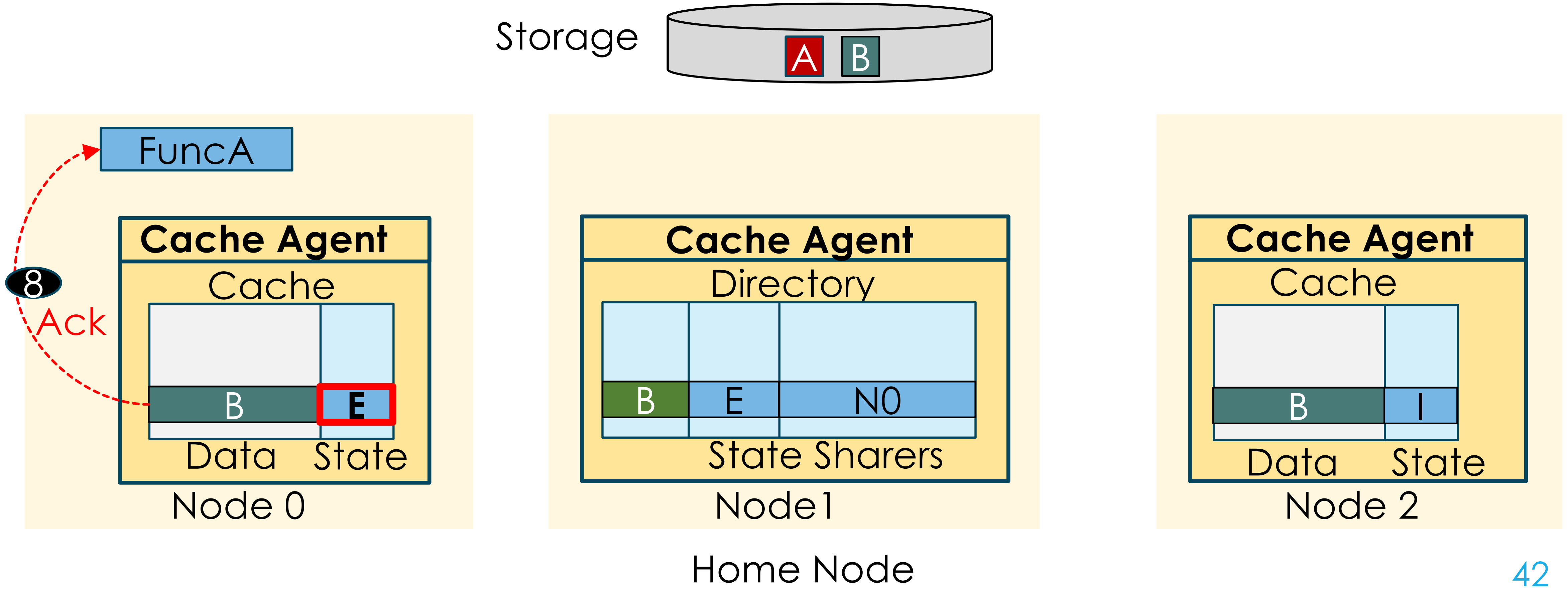
Directory-Based Invalidation Protocol



Directory-Based Invalidation Protocol



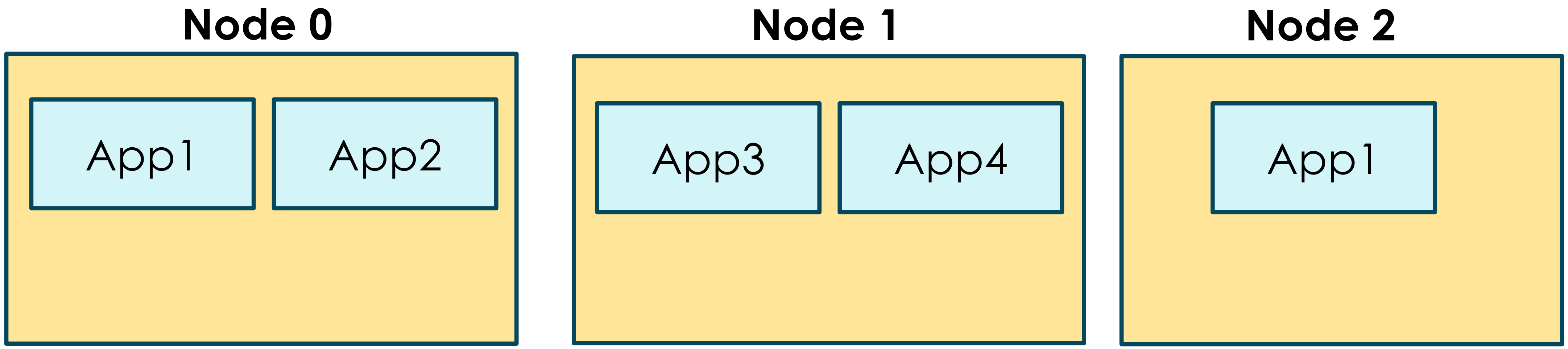
Directory-Based Invalidation Protocol



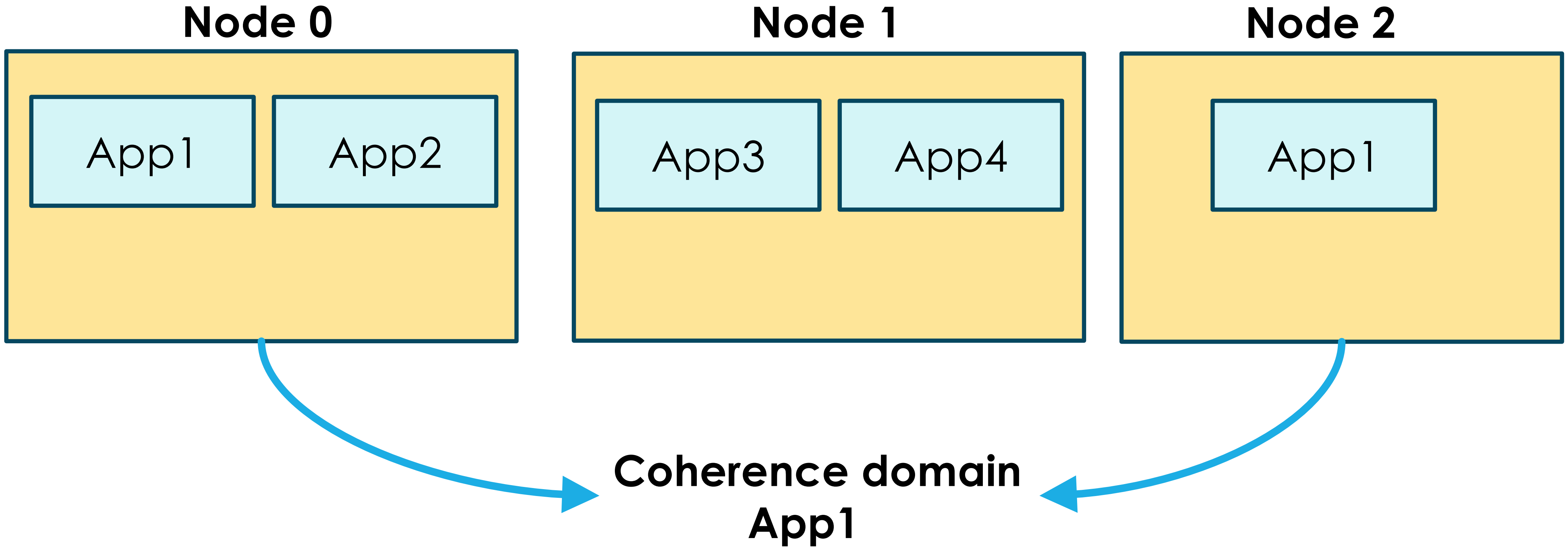
Concord Key Ideas

1. Allocated but unused per allocation memory → app’s cache
2. Directory-based invalidation protocol for cache coherence
3. **Dynamic cache coherence domains**

Dynamic Cache Coherence Domains



Dynamic Cache Coherence Domains



Node 0

App1

App2

Node 1

App3

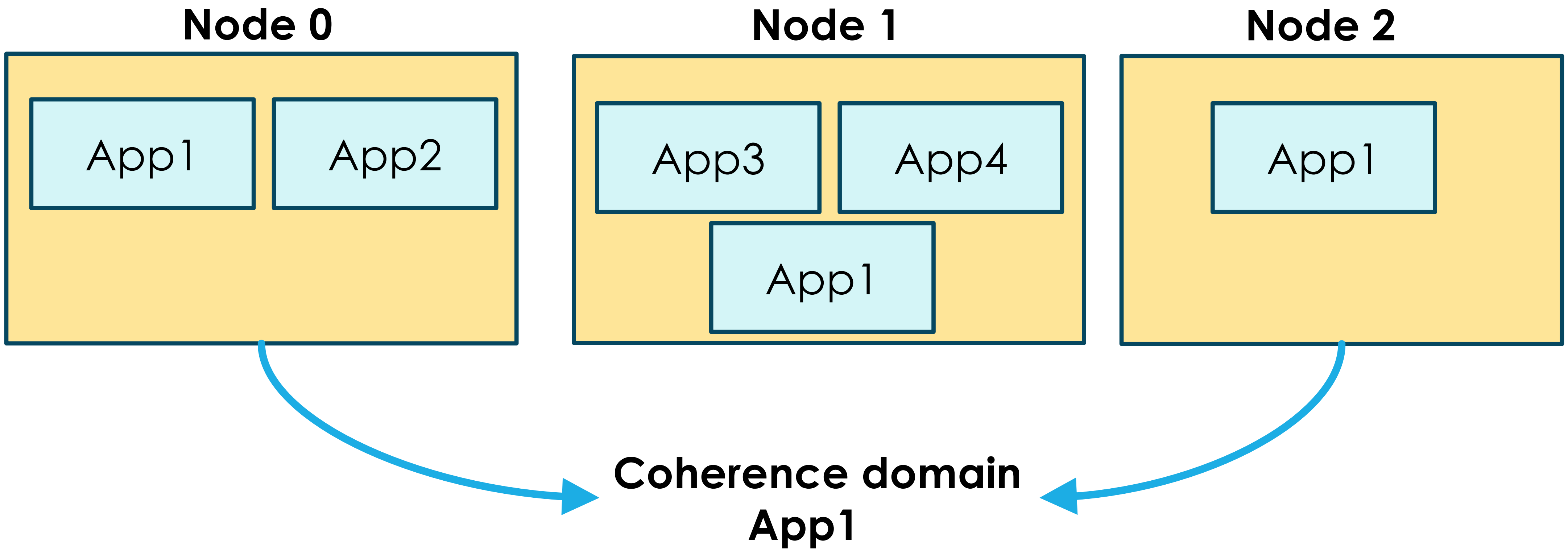
App4

Node 2

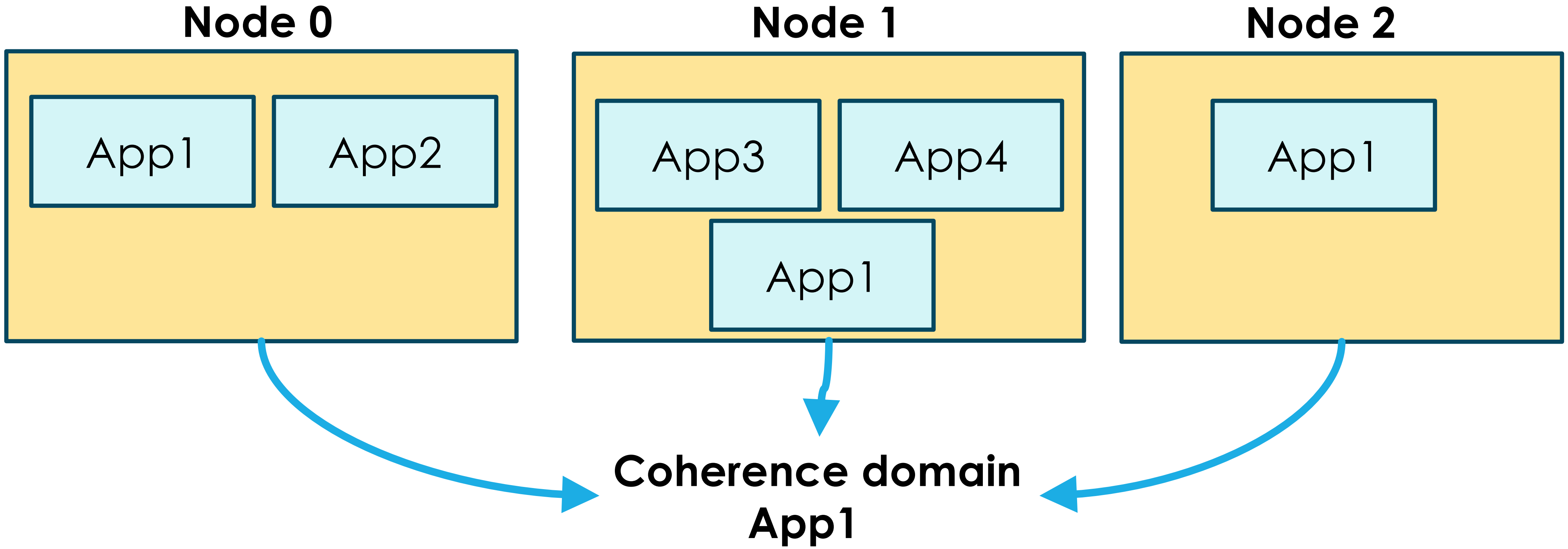
App1

Coherence domain App1

Dynamic Cache Coherence Domains



Dynamic Cache Coherence Domains



Node 0

App1

App2

Node 1

App3

App4

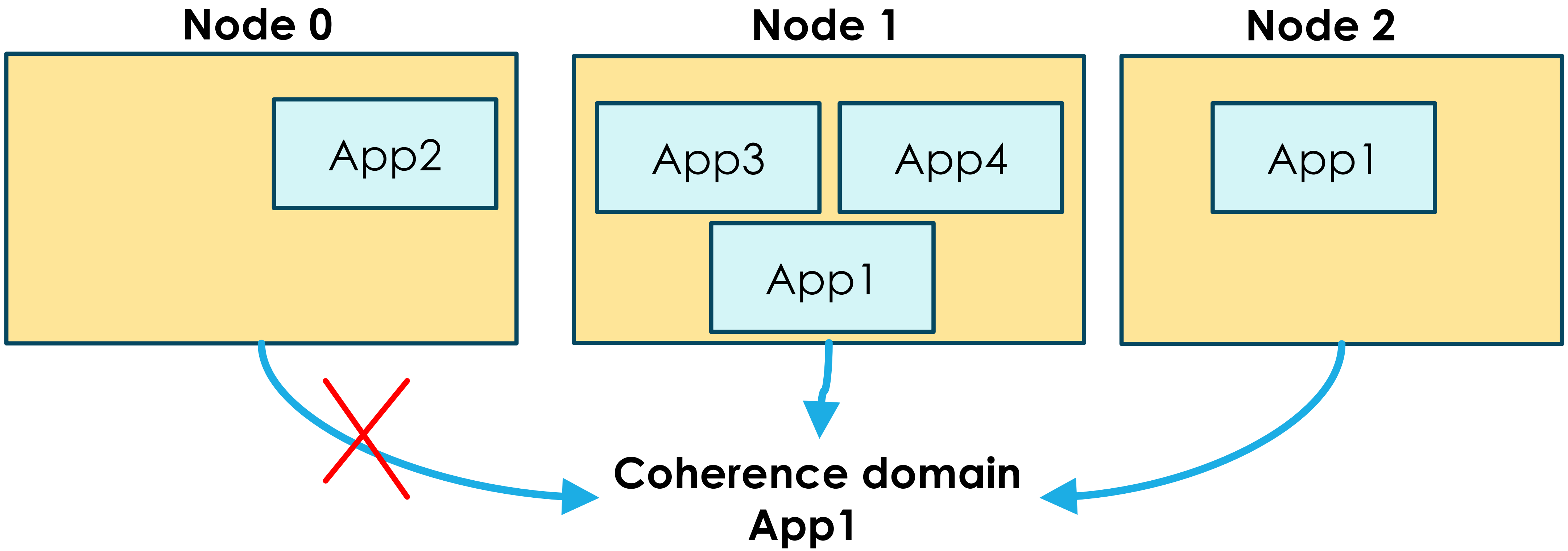
App1

Node 2

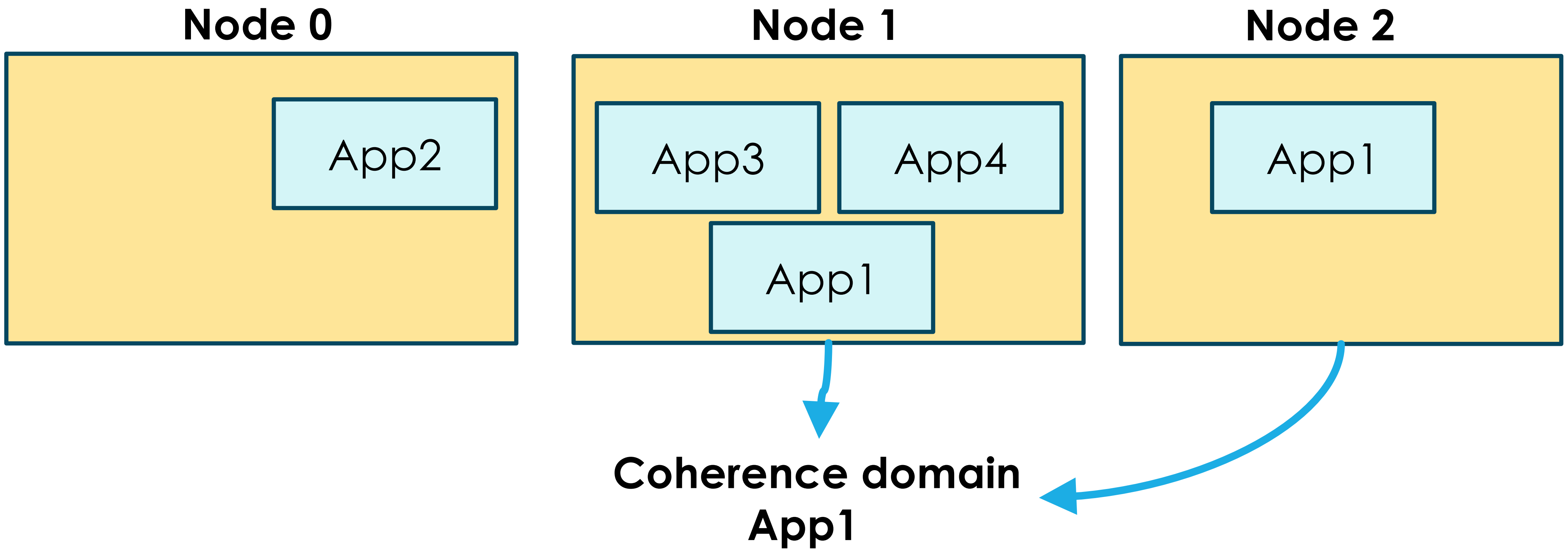
App1

Coherence domain
App1

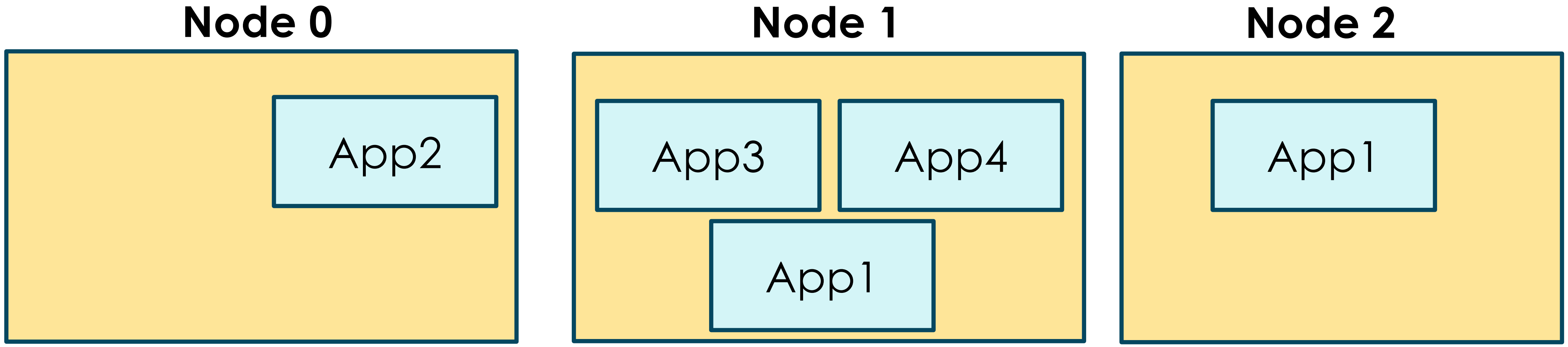
Dynamic Cache Coherence Domains

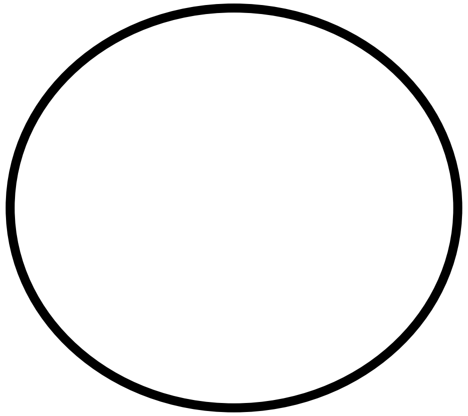


Dynamic Cache Coherence Domains



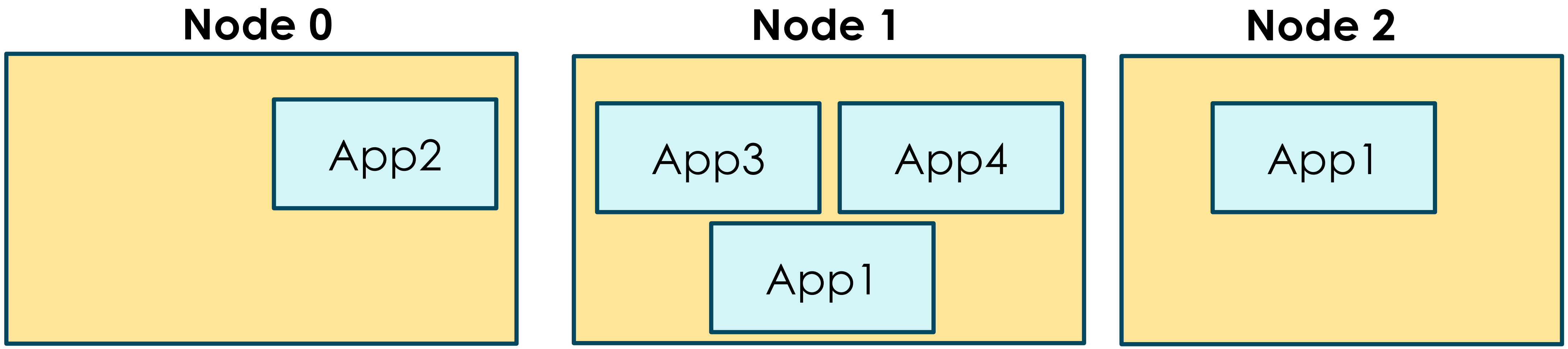
Dynamic Cache Coherence Domains





Consistent Hashing

Dynamic Cache Coherence Domains



Node 0

App2

Node 1

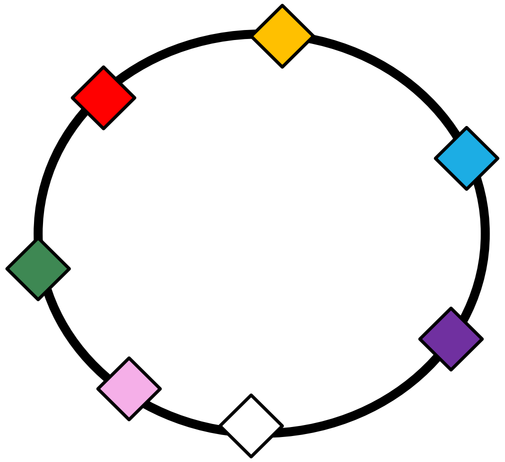
App3

App4

App1

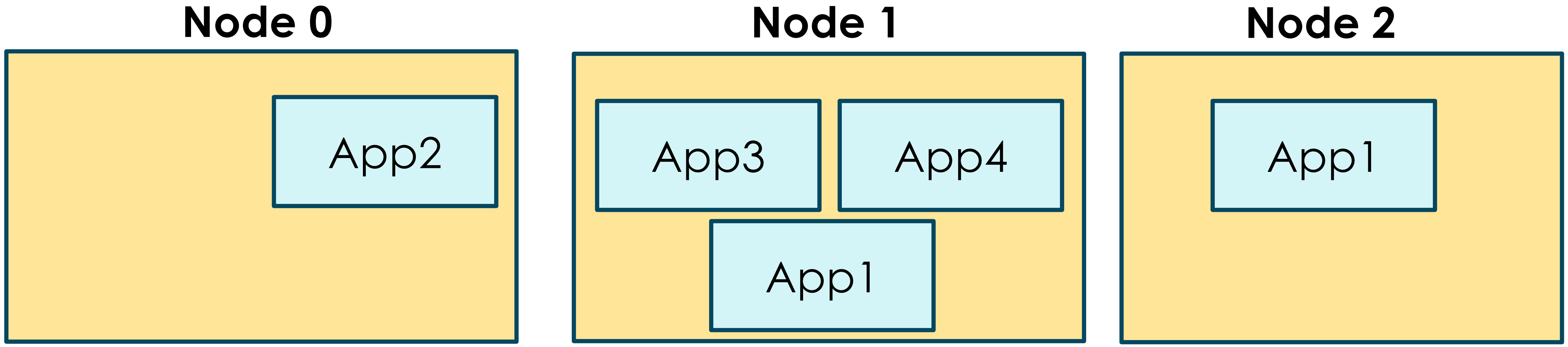
Node 2

App1



Consistent Hashing

Dynamic Cache Coherence Domains



Node 0

App2

Node 1

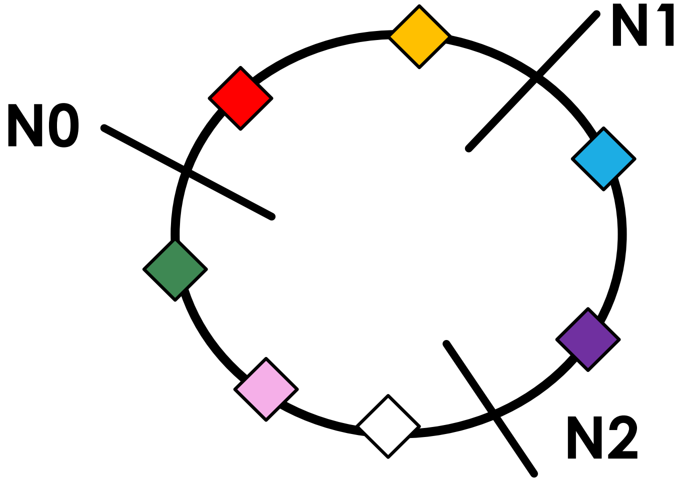
App3

App4

App1

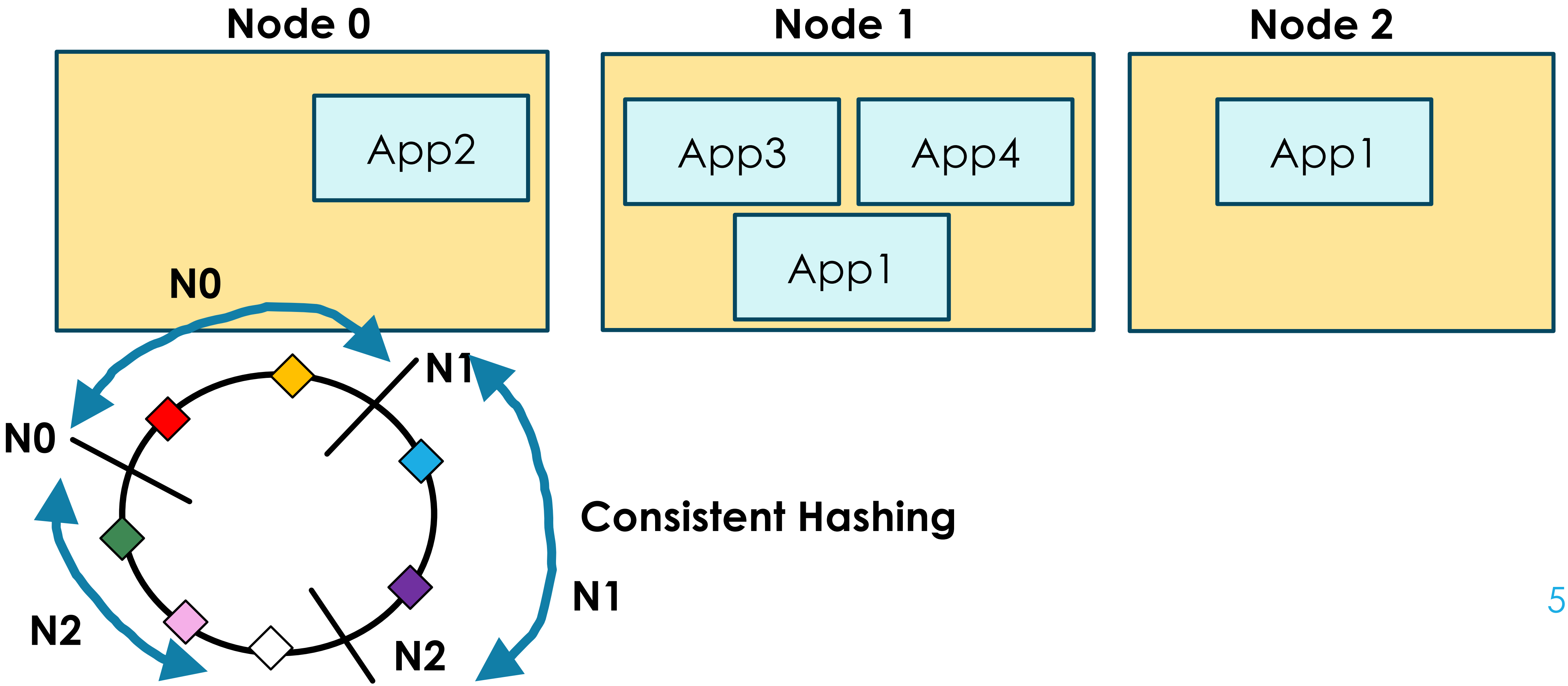
Node 2

App1



Consistent Hashing

Dynamic Cache Coherence Domains



Node 0

App2

N0

Node 1

App3

App4

App1

Node 2

App1

N1

N0

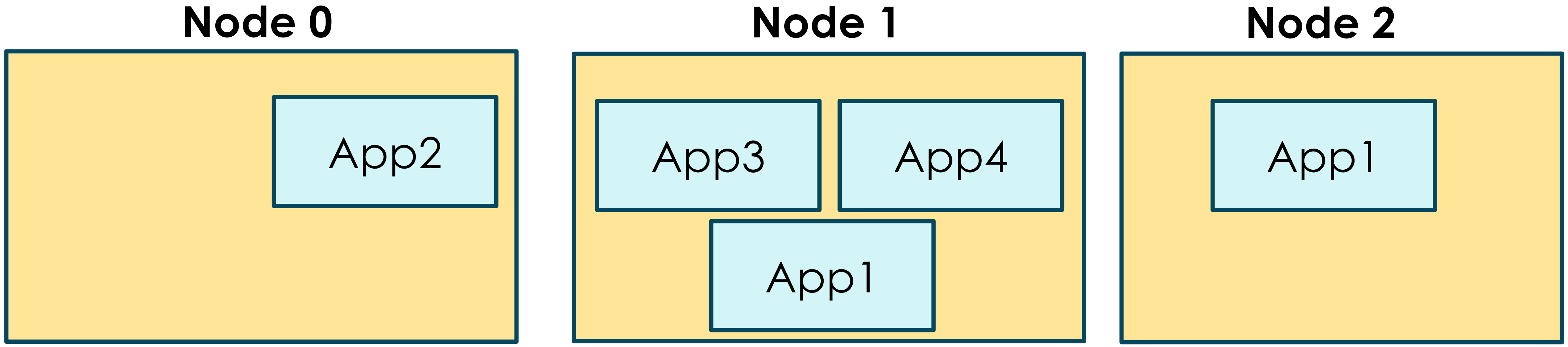
N2

N2

N1

Consistent Hashing

Dynamic Cache Coherence Domains



Node 0

App2

Node 1

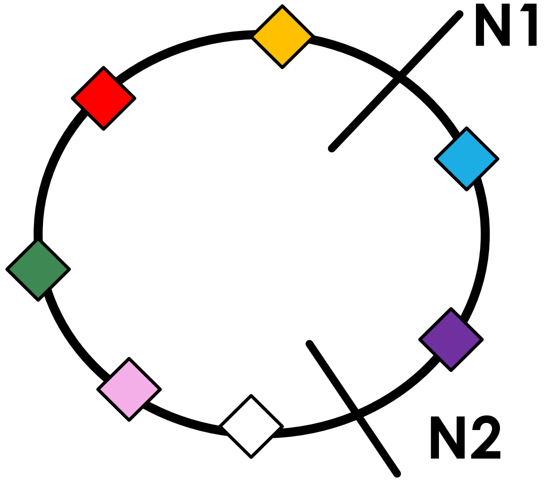
App3

App4

App1

Node 2

App1

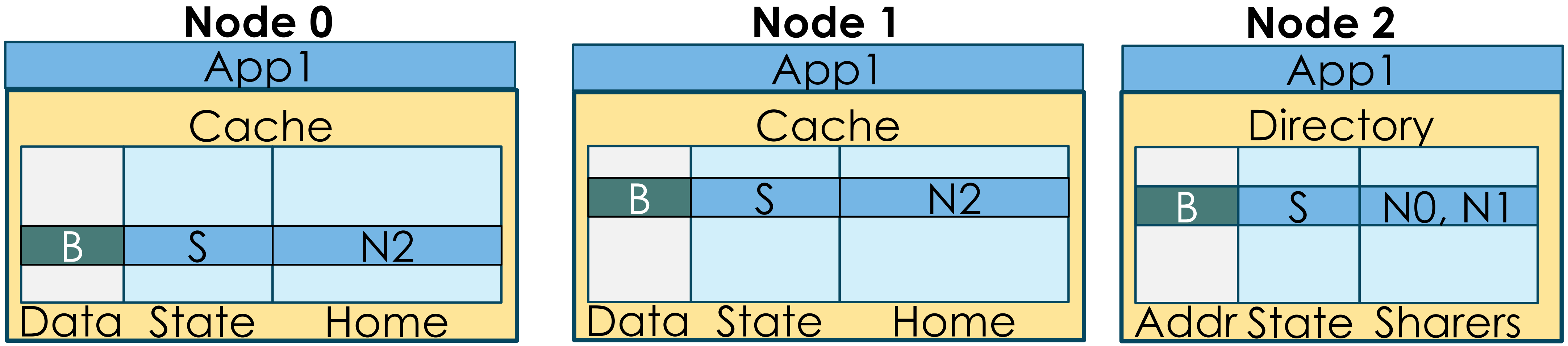


Consistent Hashing

Concord Key Ideas

1. Allocated but unused per allocation memory → app's cache
2. Directory-based invalidation protocol for cache coherence
3. Dynamic cache coherence domains
4. **Fault-tolerant directory-based coherence protocol**

Fault-Tolerant Cache Coherence Protocol



Node 0

App1

Cache

Data	State	Home
B	S	N2

Node 1

App1

Cache

Data	State	Home
B	S	N2

Node 2

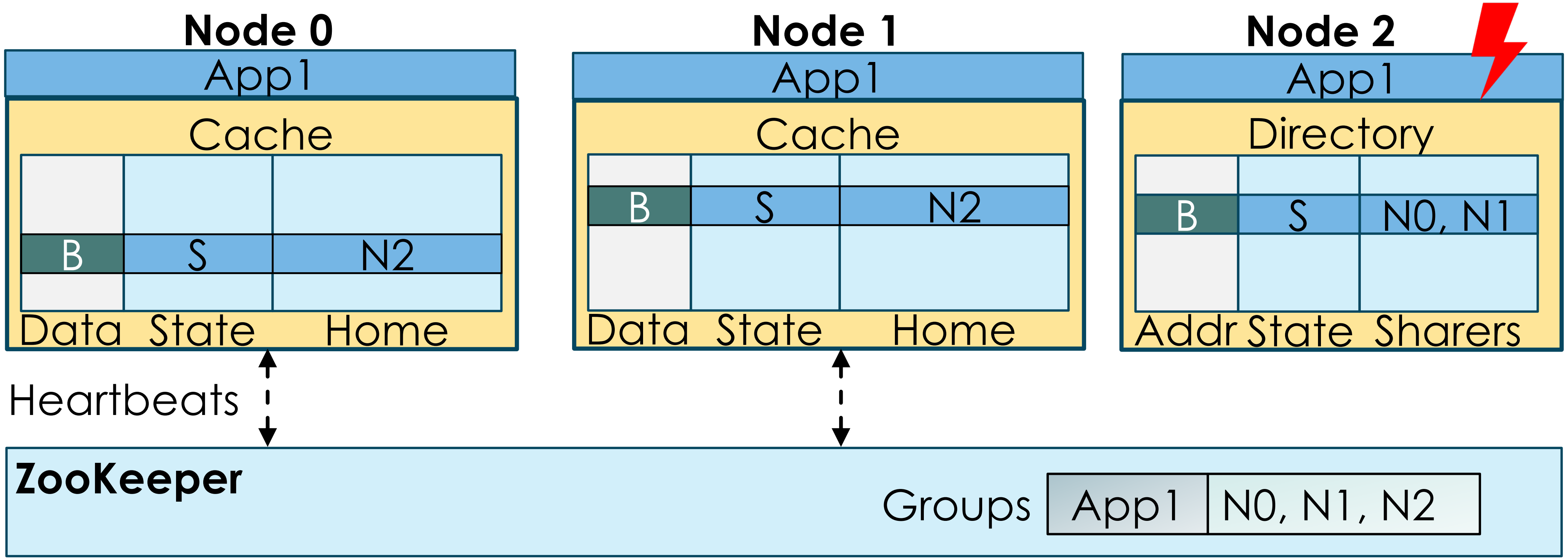
App1

Directory

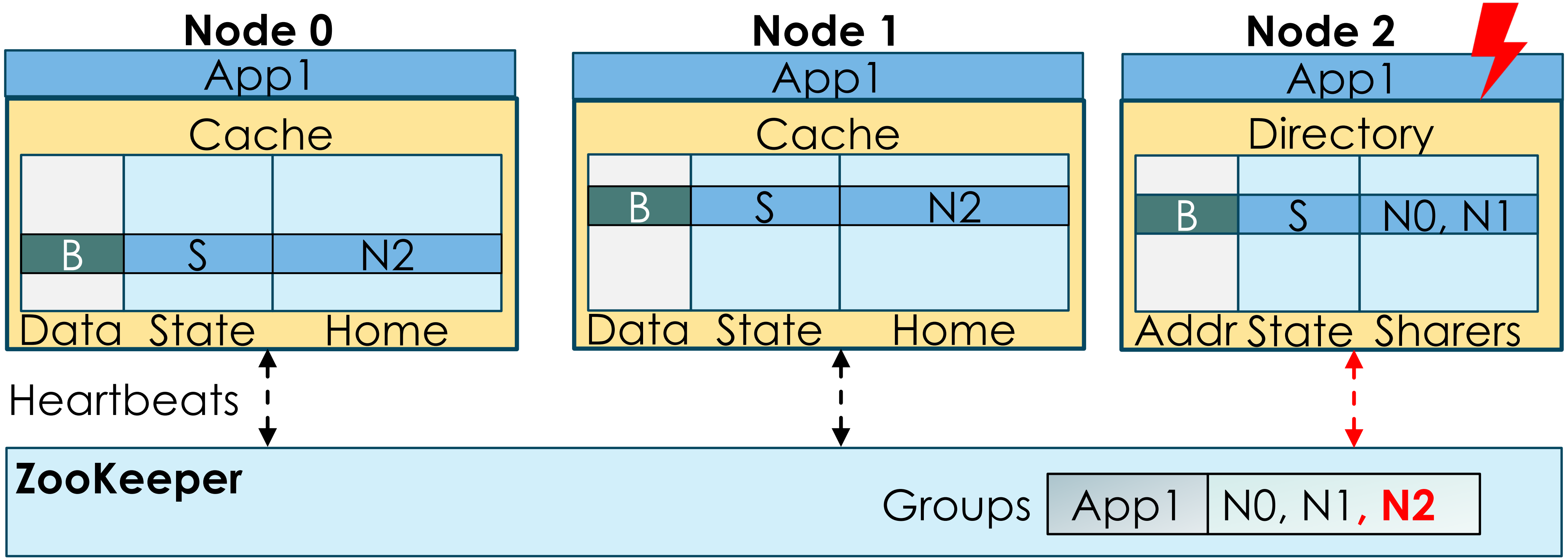
Addr	State	Sharers
B	S	N0, N1

1. Data in storage is always up-to-date → write-through caches
2. Failed node detected via heartbeats from configuration manager

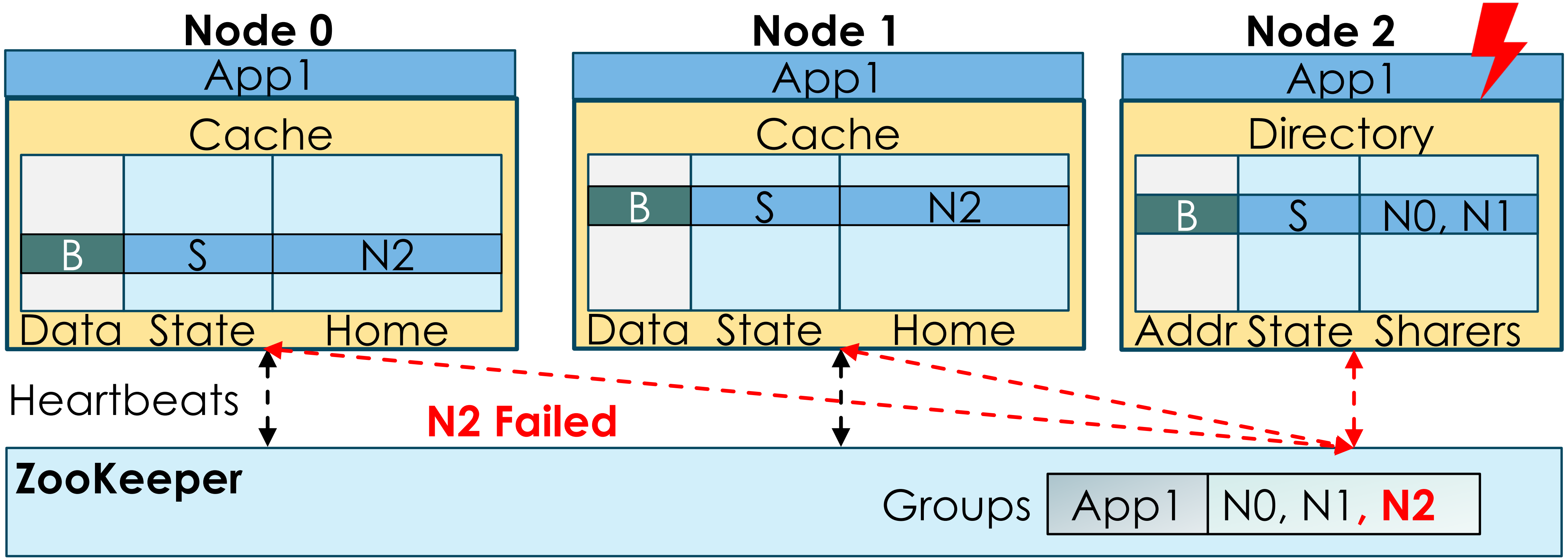
Fault-Tolerant Cache Coherence Protocol



Fault-Tolerant Cache Coherence Protocol



Fault-Tolerant Cache Coherence Protocol



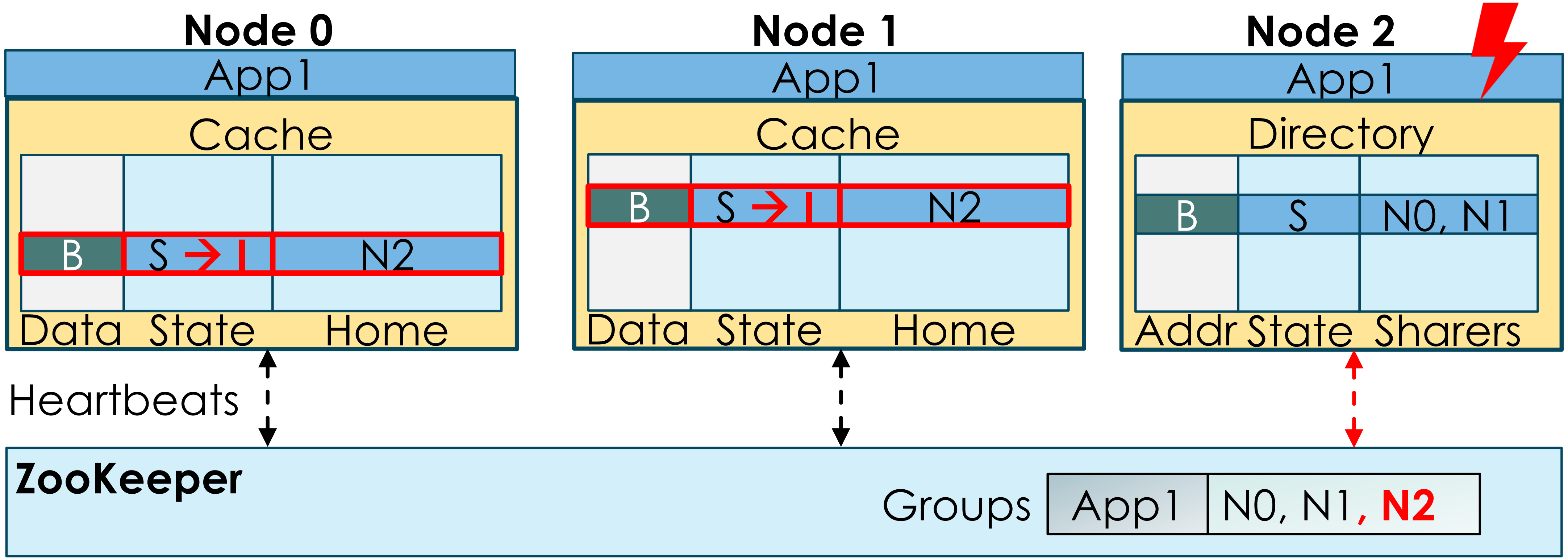
Data	State	Home
B	S	N2

Data	State	Home
B	S	N2

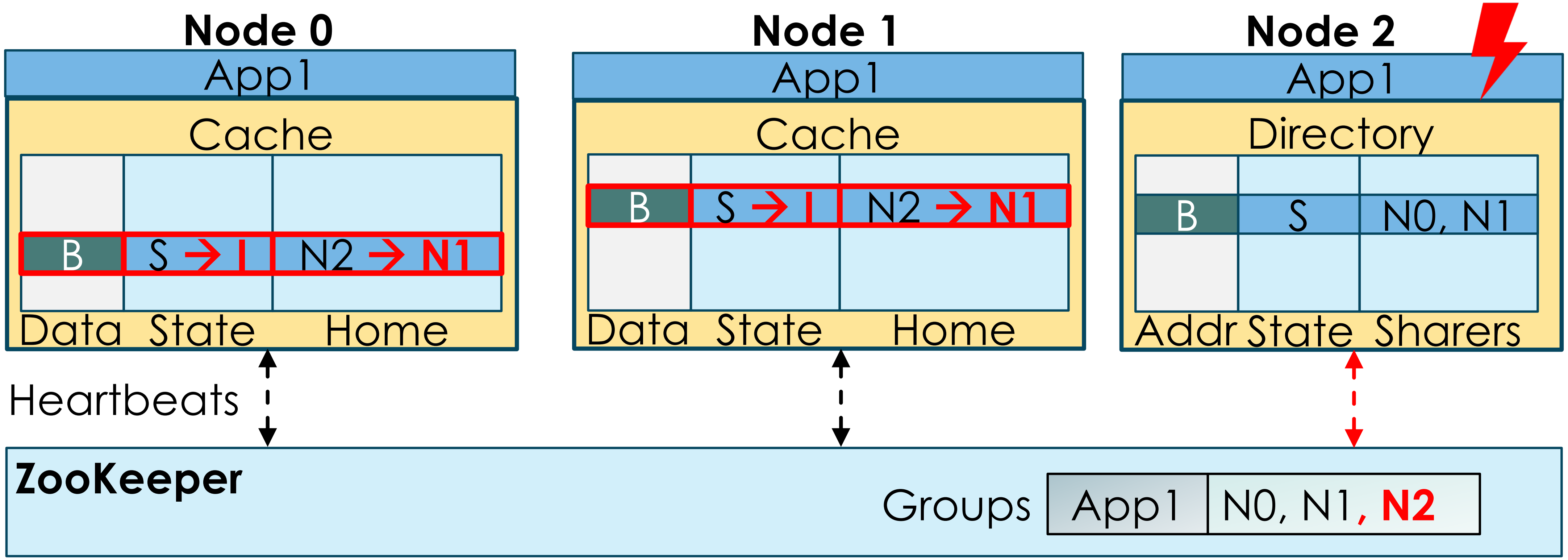
Addr	State	Sharers
B	S	N0, N1

Groups		
	App1	N0, N1, N2

Fault-Tolerant Cache Coherence Protocol



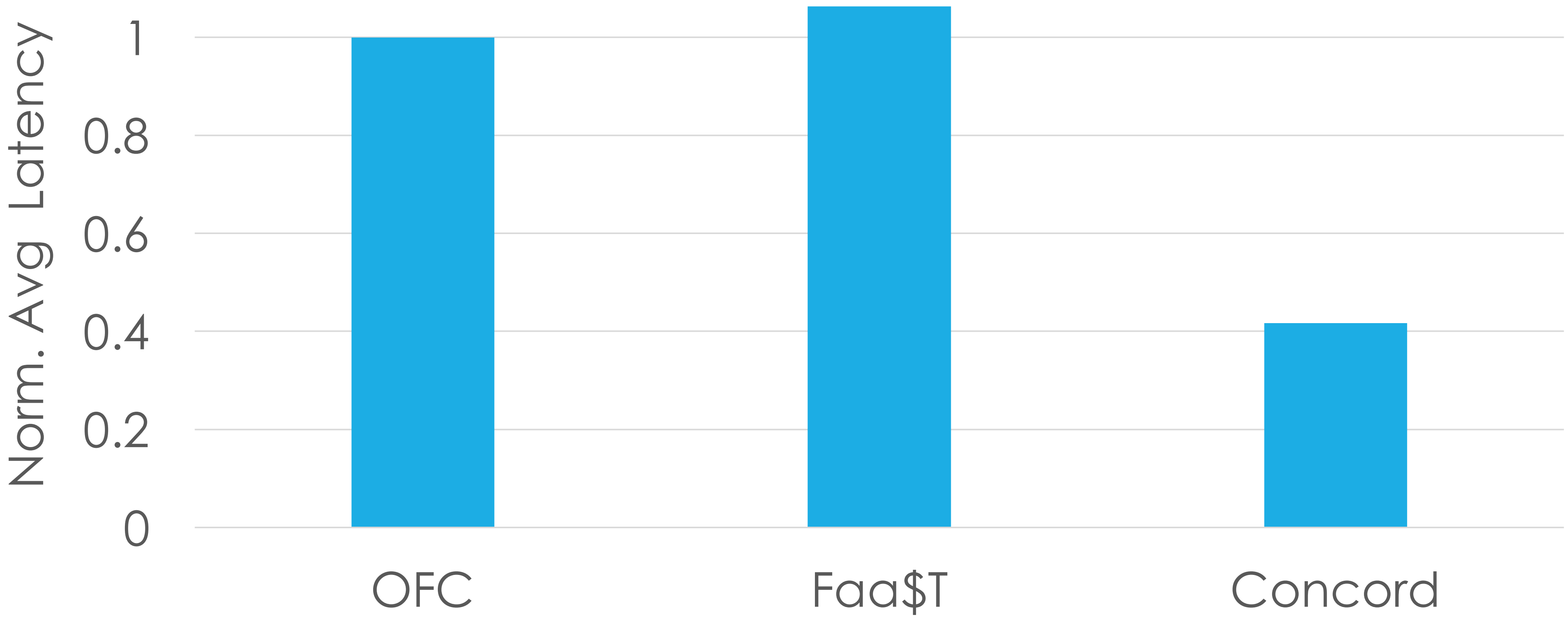
Fault-Tolerant Cache Coherence Protocol



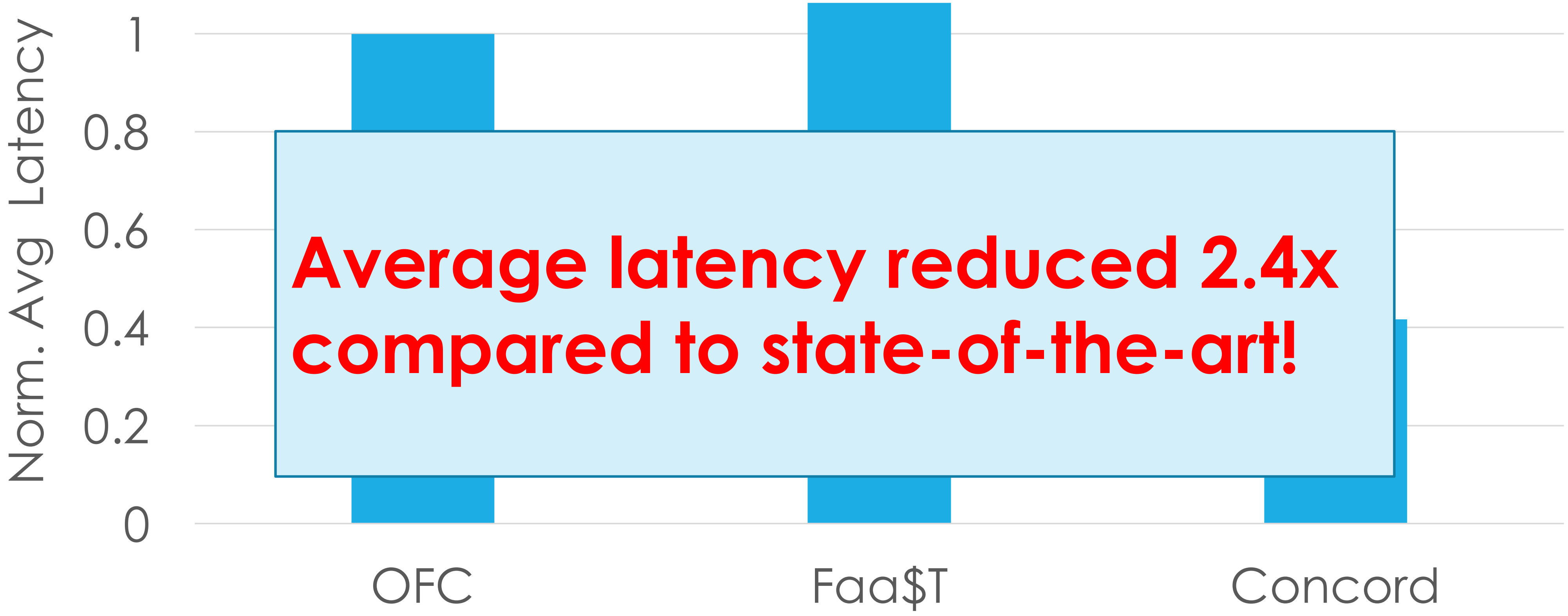
Evaluation Methodology

- Cluster with 16 Intel Xeon servers (20 cores, each)
- Platform: optimized OpenWhisk (using MXFaaS ISCA'23)
- Systems evaluated
  - **OFC (EuroSys'21):** data cached only in the home
  - **Faa$T (SoCC'21):** data in multiple caches + versioning coherence protocol
  - **Concord:** our proposal

Concord Reduces Average Latency

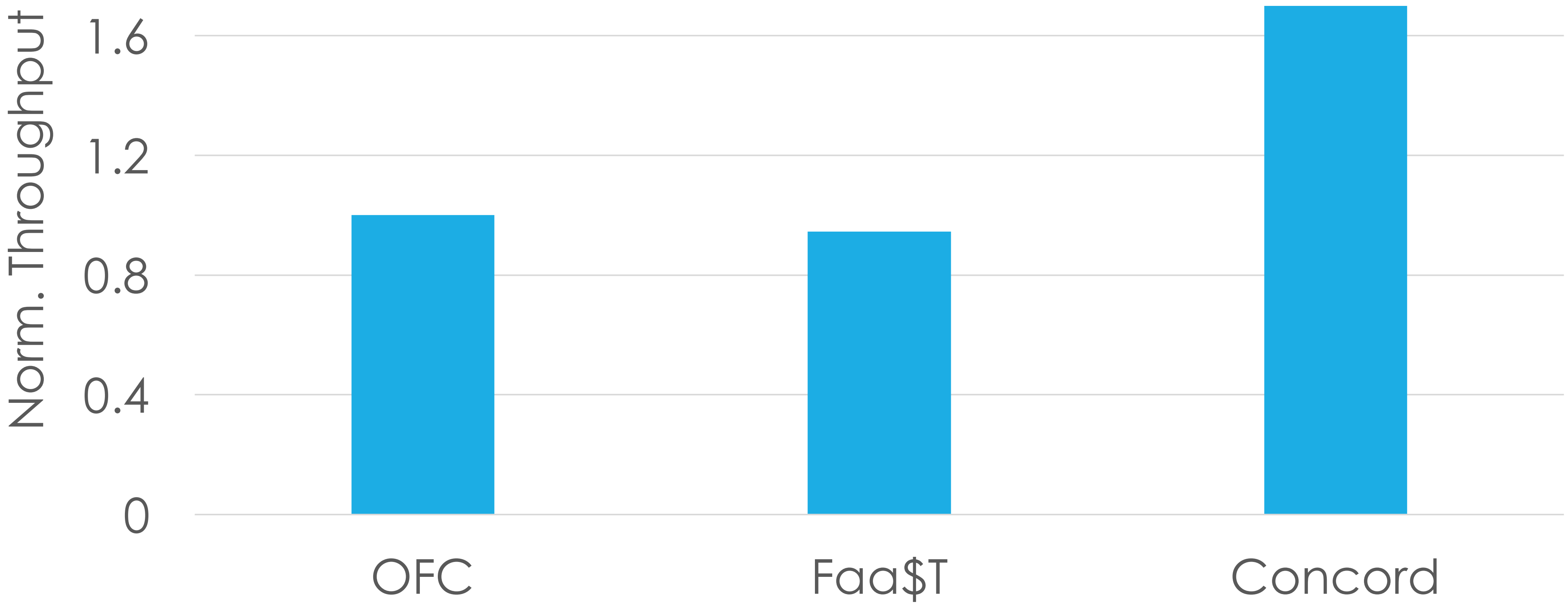


Concord Reduces Average Latency

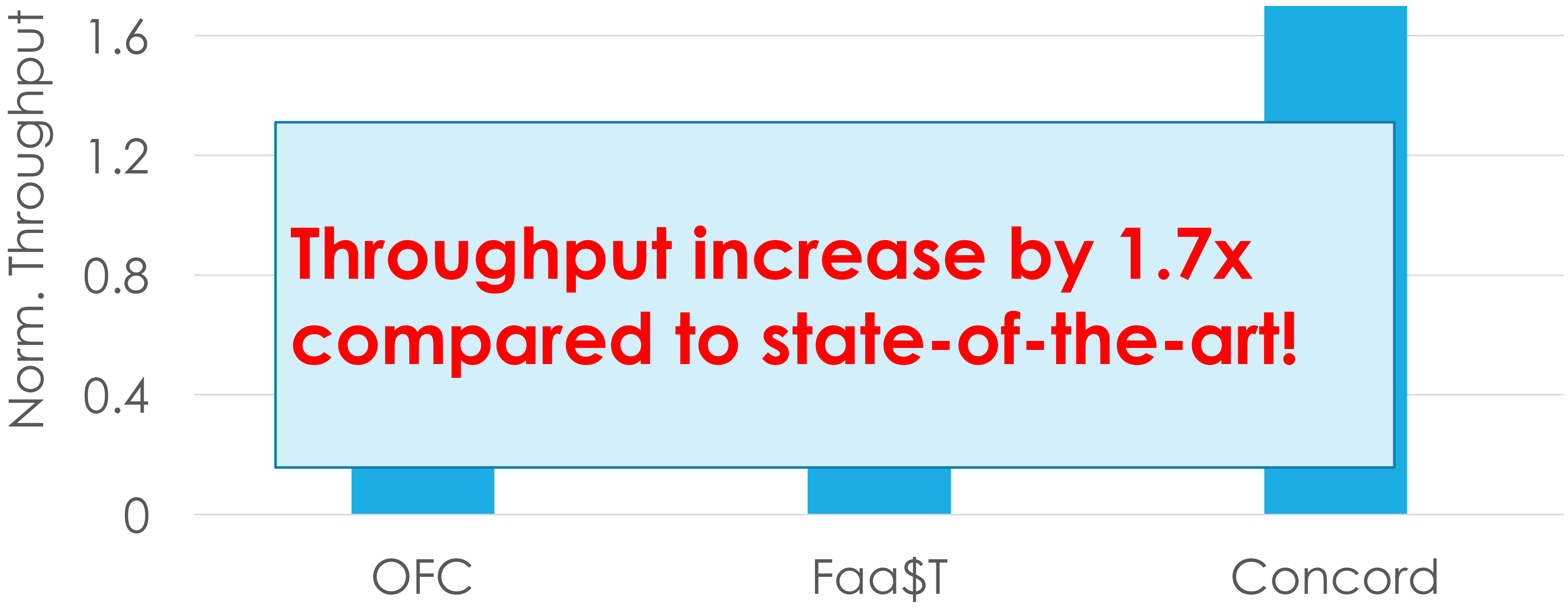


Average latency reduced 2.4x compared to state-of-the-art!

Concord Improves Throughput



Concord Improves Throughput



Throughput increase by 1.7x compared to state-of-the-art!

Conclusions

- Storage accesses expensive in serverless environments
- How to design efficient software caching scheme?
- Concord: high-performance and fault-tolerant distributed directory-based coherence protocol for software caches
  - 2.4x lower average latency and 1.7x higher throughput

Concord: Rethinking Distributed Coherence for Software Caches in Serverless Environments

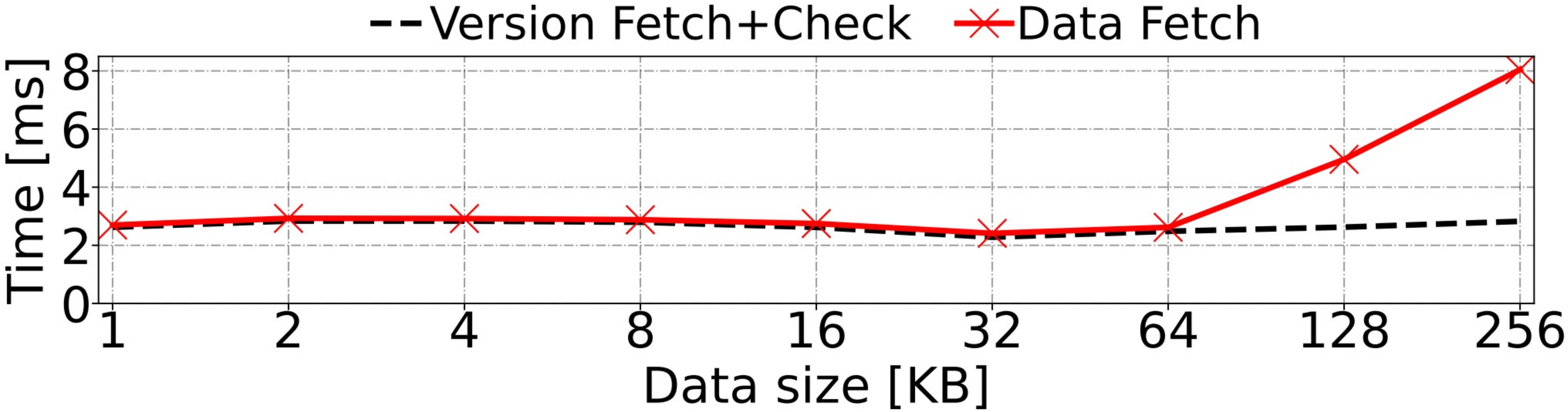
HPCA ‘25

Jovan Stojkovic, Chloe Alverti, Alan Andrade, Nikoleta Iliakopoulou, Tianyin Xu, Hubertus Franke*, Josep Torrellas

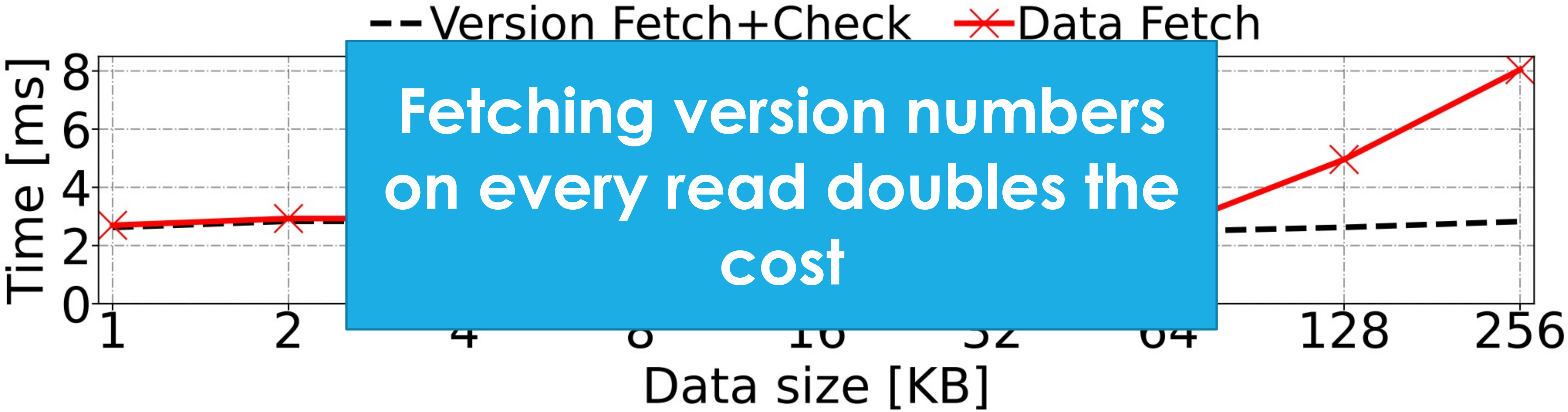
University of Illinois at Urbana-Champaign, *IBM Research

Questions?

Version Fetch + Check → High Overheads

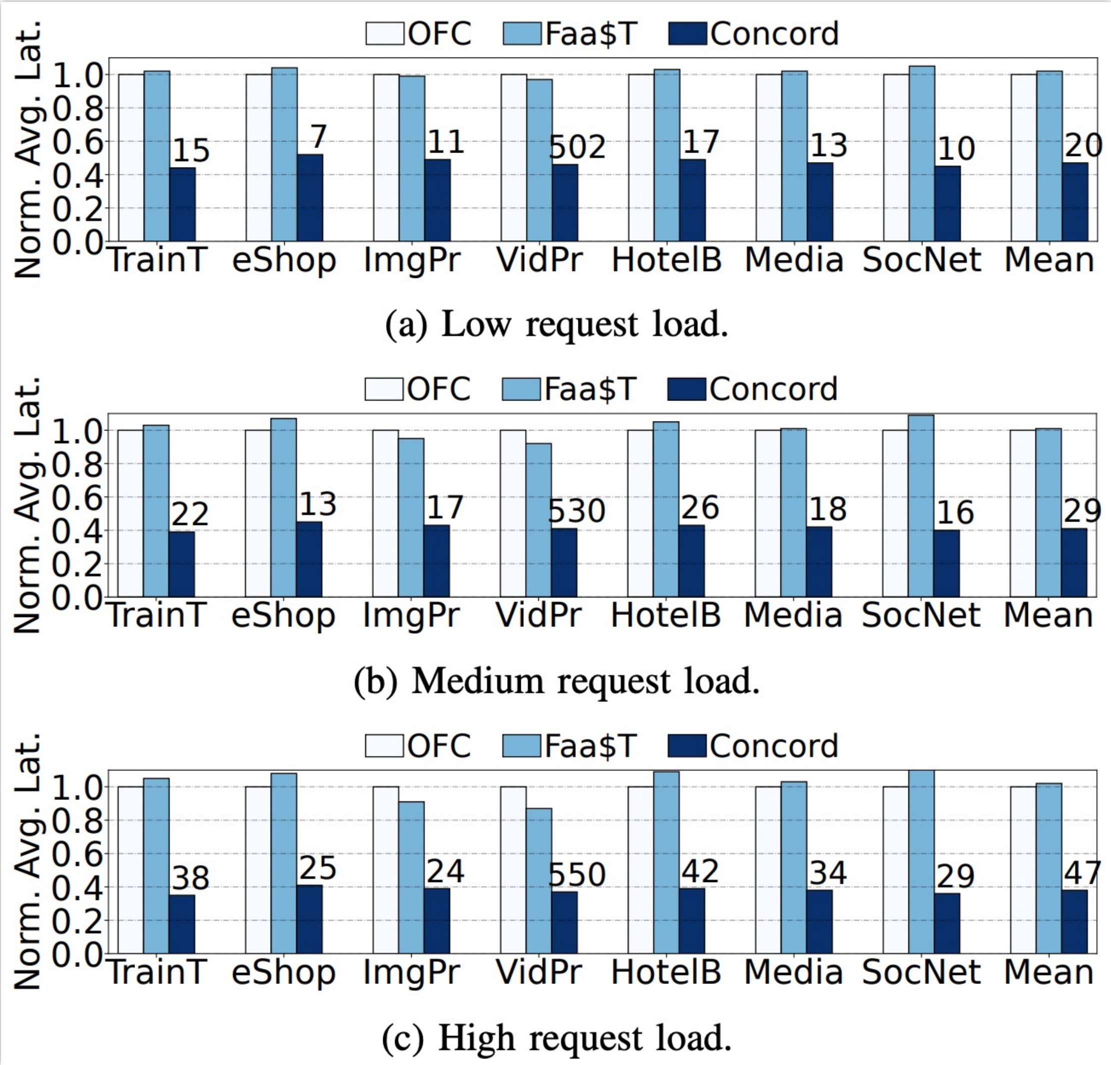


Version Fetch + Check → High Overheads



Fetching version numbers on every read doubles the cost

Evaluation – Loads



(a) Low request load.

(b) Medium request load.

(c) High request load.

What are Properties of the Workload?

- Data items small → 80% in production workloads less than 12KB
- Read operations dominate → 77% in production workloads are reads
- Serverless functions span ~10s of nodes + designed to be fault tolerant

→ **Time to rethink invalidation-based directory protocols!**

Unlocking New Capabilities with Concord

1. **Support for FaaS transactions**
   - Mark all data accesses during transactions as "speculative"
   - Use ***coherence messages*** to detect transaction violation
   - If violated, squash the transaction and rollback
   - Otherwise, commit the transaction

Unlocking New Capabilities with Concord

2. **Communication-aware function placement**
   - Use ***coherence messages*** to detect common function pairs
   - Build producer-consumer table → paired functions
   - When placing a function check if there is an already scheduled ”pair” function and collocate such functions