MEGALON: Efficient Data Sharing for Partly Coherent CXL Memory

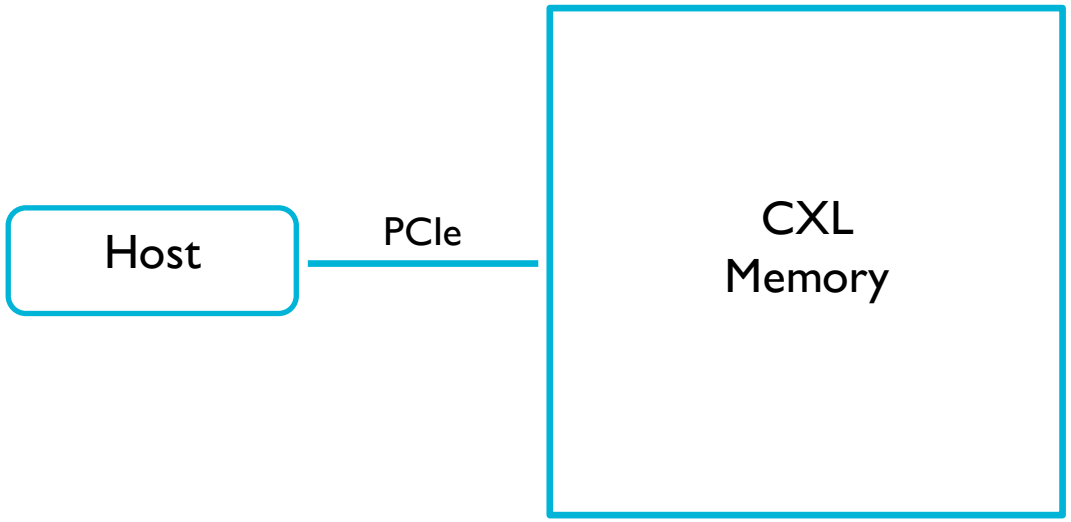
Jiyu Hu, Seokjoo Cho*, Landon Johnson*, Kiran Hombal, Shreesha G. Bhat, Marcos K. Aguilera[1], Ram Alagappan, Aishwarya Ganesan

University of Illinois Urbana-Champaign
[1]NVIDIA

*equal contribution

Compute Express Link (CXL) Background

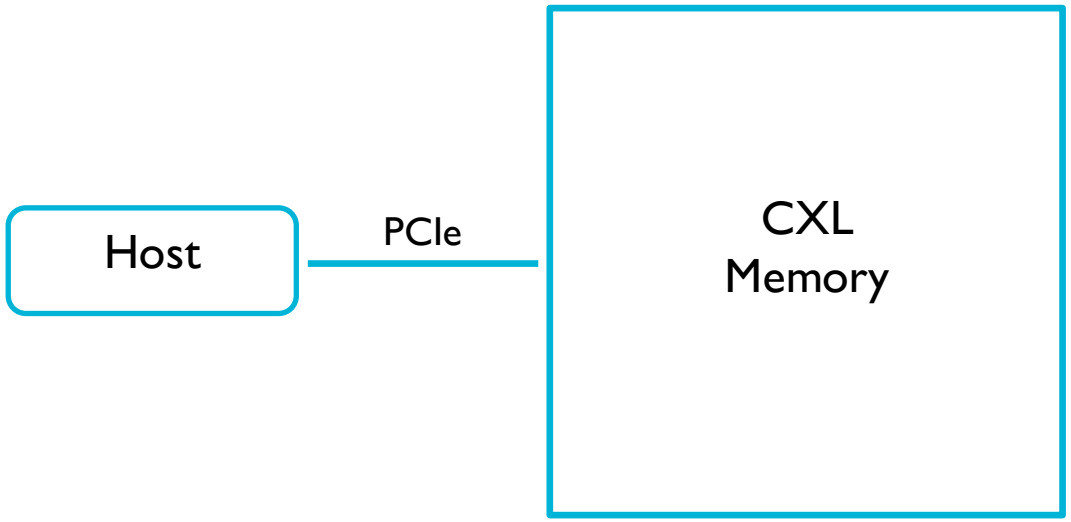
CXL memory: a memory device over CXL attached to compute hosts



Compute Express Link (CXL) Background

CXL memory: a memory device over CXL attached to compute hosts

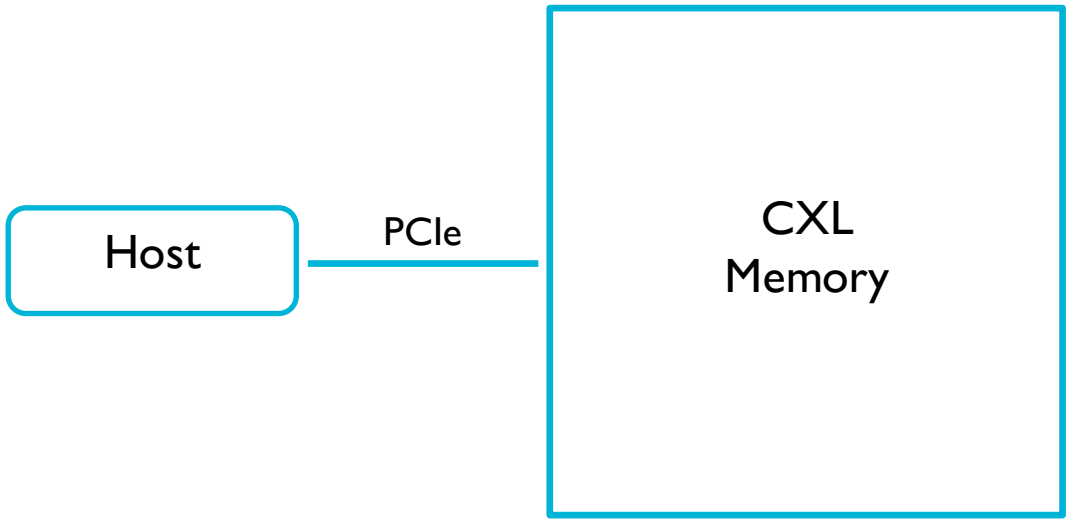
- CPU loads/stores — direct access, zero data copies



Compute Express Link (CXL) Background

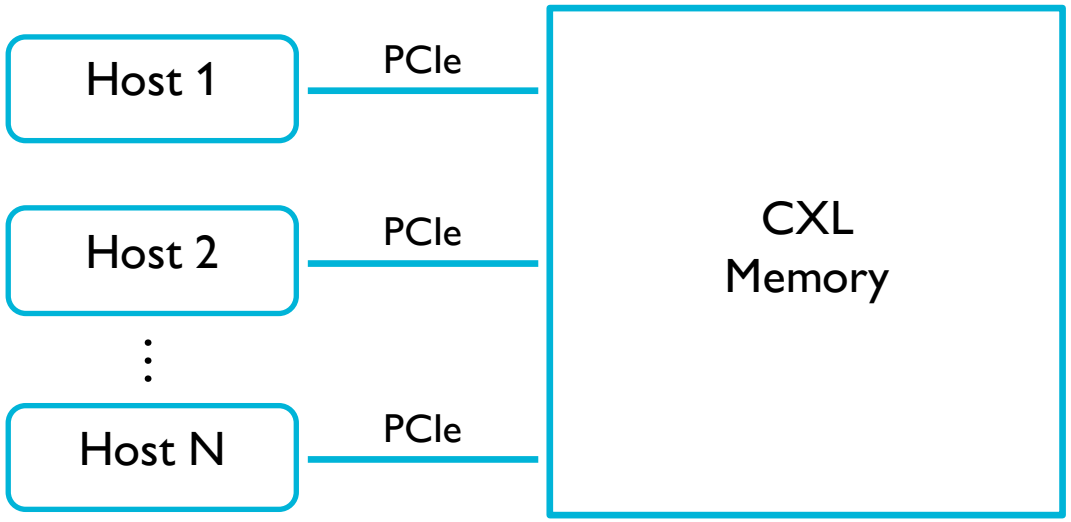
CXL memory: a memory device over CXL attached to compute hosts

- CPU loads/stores — direct access, zero data copies
- Function as memory expander



CXL 3.0: Memory Sharing Across Multiple Hosts

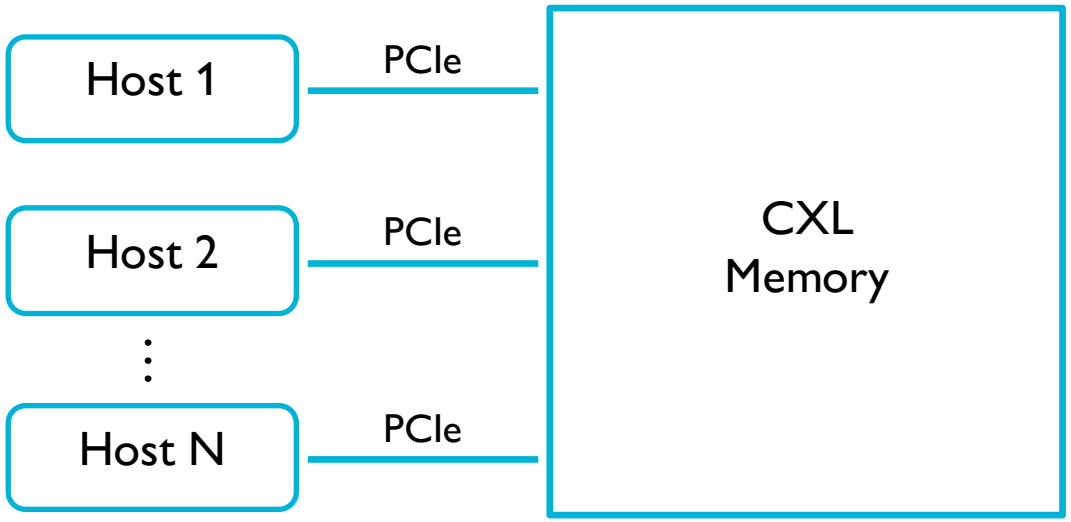
CXL 3.0: TBs of shared memory across multiple hosts



CXL 3.0: Memory Sharing Across Multiple Hosts

CXL 3.0: TBs of shared memory across multiple hosts

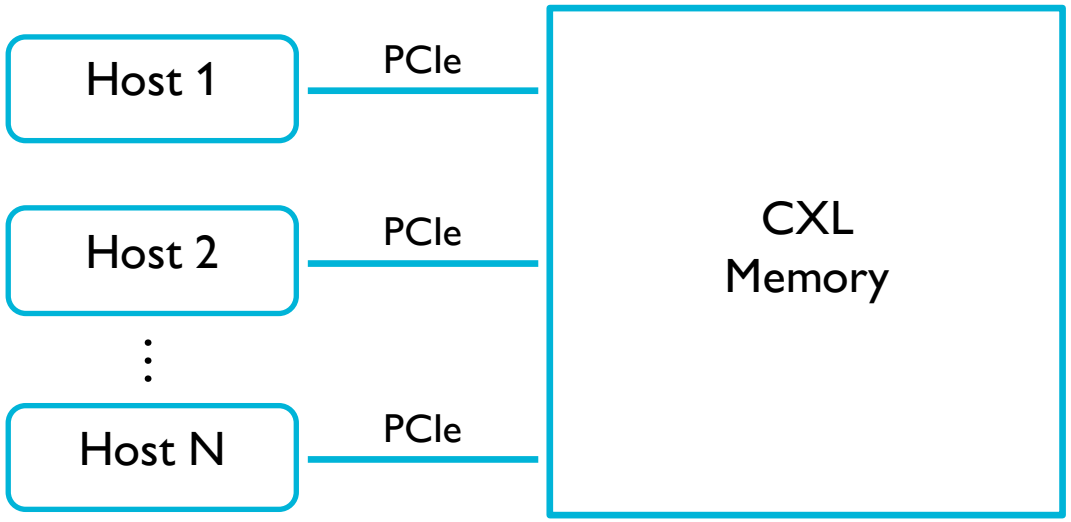
- CPU loads/stores — direct access, zero data copies



CXL 3.0: Memory Sharing Across Multiple Hosts

CXL 3.0: TBs of shared memory across multiple hosts

- CPU loads/stores — direct access, zero data copies
- Efficient multi-host databases, key-value & object stores



The Partly Coherent CXL Model

Hardware coherence covers only a small slice of CXL 3.0 memory

The Partly Coherent CXL Model

Hardware coherence covers only a small slice of CXL 3.0 memory

A few hundred MBs

Hardware coherent

The Partly Coherent CXL Model

Hardware coherence covers only a small slice of CXL 3.0 memory

A few hundred MBs
Hardware coherent

Several TBs
No hardware coherence

The Partly Coherent CXL Model

Hardware coherence covers only a small slice of CXL 3.0 memory

Small Coherent Region

A few hundred MBs

Hardware coherent

Several TBs

No hardware coherence

The Partly Coherent CXL Model

Hardware coherence covers only a small slice of CXL 3.0 memory

Small Coherent Region
A few hundred MBs
Hardware coherent

Large Non-Coherent Region
Several TBs
No hardware coherence

The Partly Coherent CXL Model

Hardware coherence covers only a small slice of CXL 3.0 memory

Small Coherent Region
A few hundred MBs
Hardware coherent

Large Non-Coherent Region
Several TBs
No hardware coherence

Problem:
No cross-host cache coherence for LNR, processor caches may hold stale values

The Partly Coherent CXL Model

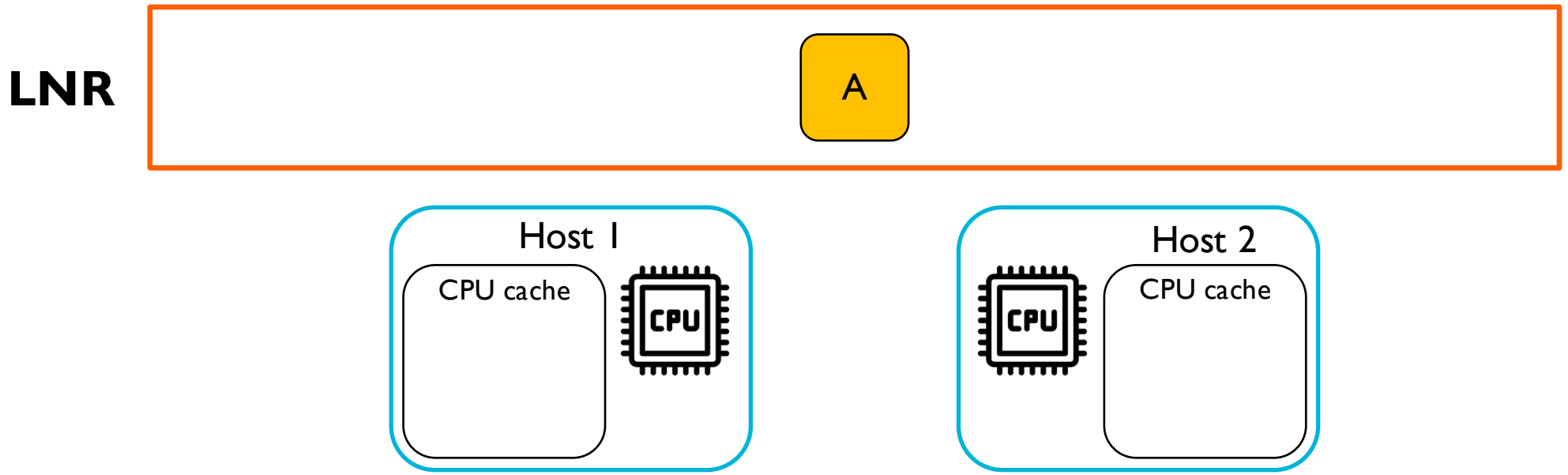
Hardware coherence covers only a small slice of CXL 3.0 memory

Small Coherent Region
A few hundred MBs
Hardware coherent

Large Non-Coherent Region
Several TBs
No hardware coherence

Problem:
No cross-host cache coherence for LNR, processor caches may hold stale values
→ correctness bugs when data shared across hosts

The Partly Coherent CXL Model

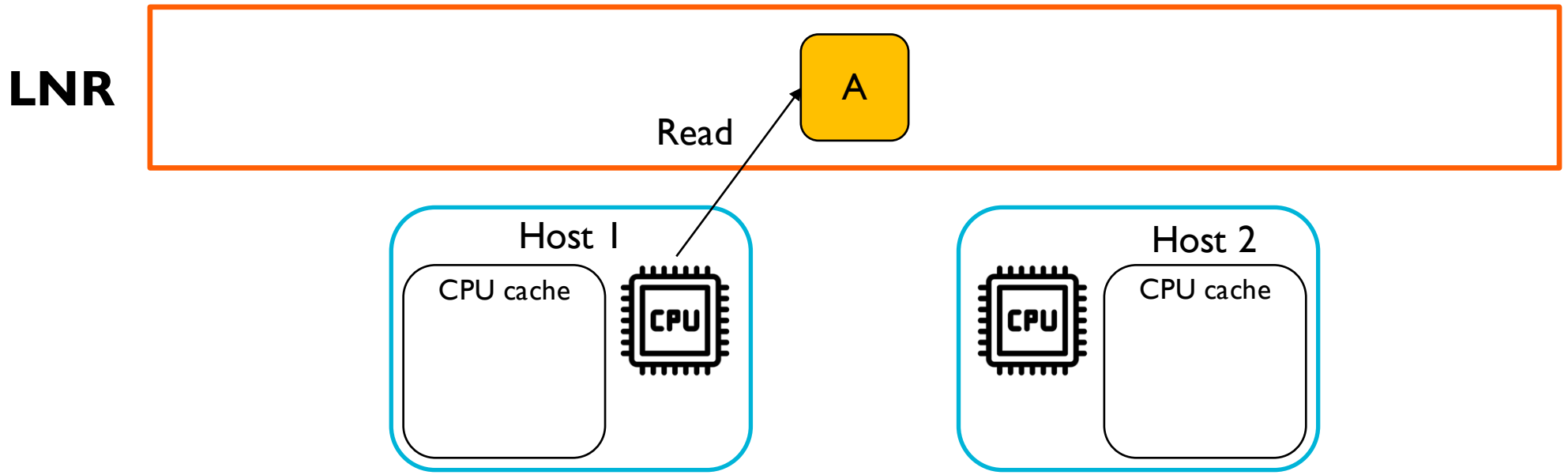


Problem:

No cross-host cache coherence for LNR, processor caches may hold stale values

→ correctness bugs when data shared across hosts

The Partly Coherent CXL Model

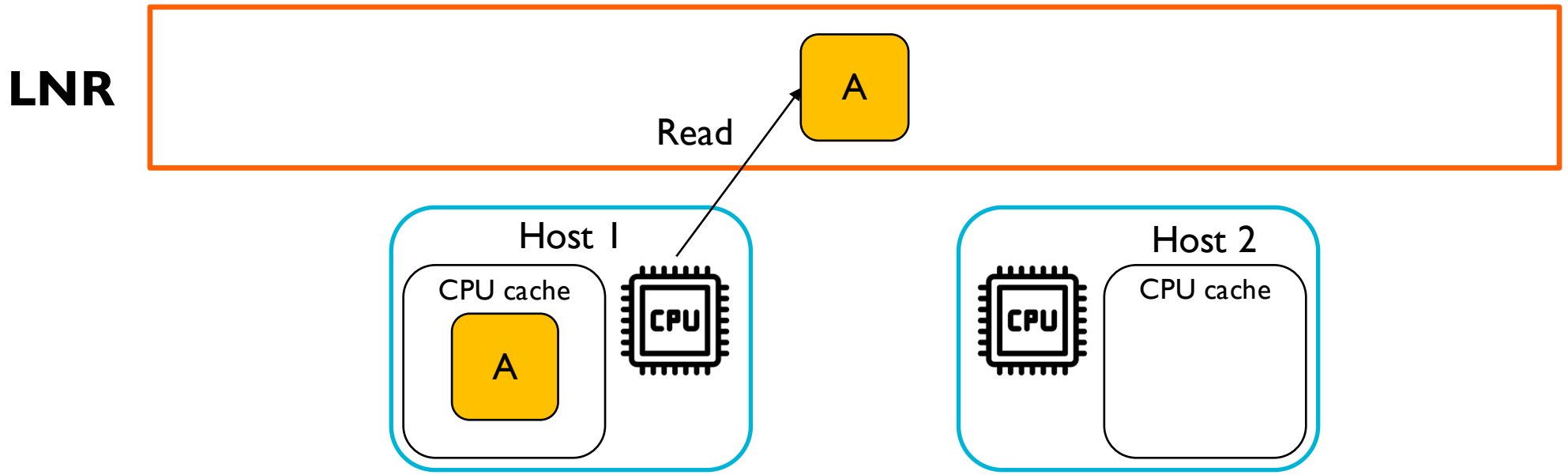


Problem:

No cross-host cache coherence for LNR, processor caches may hold stale values

→ correctness bugs when data shared across hosts

The Partly Coherent CXL Model

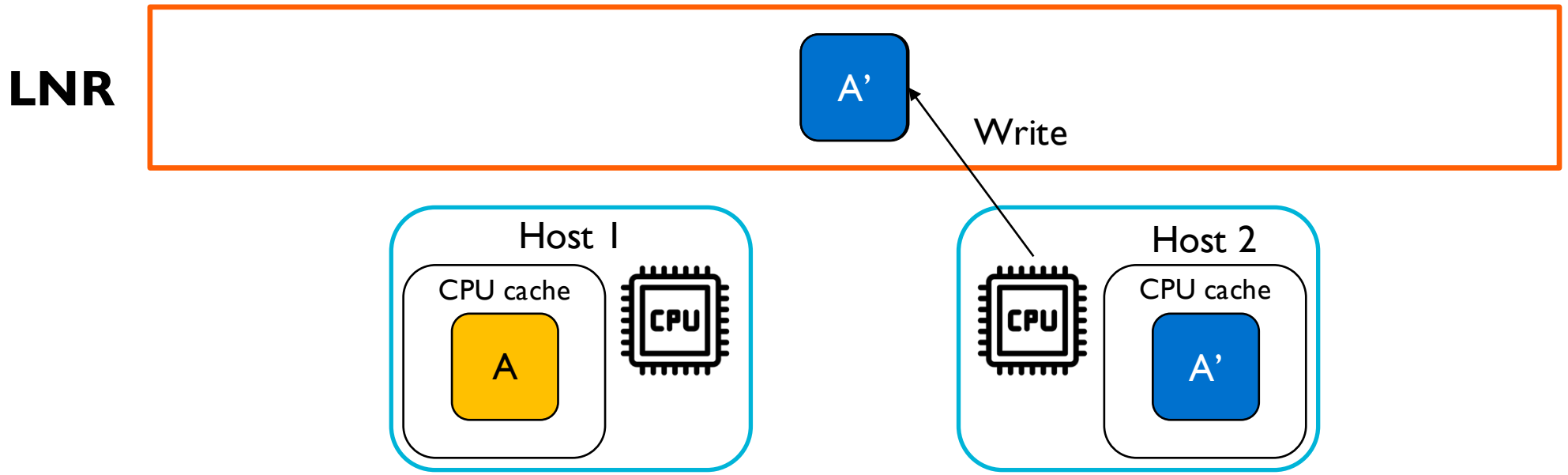


Problem:

No cross-host cache coherence for LNR, processor caches may hold stale values

→ correctness bugs when data shared across hosts

The Partly Coherent CXL Model

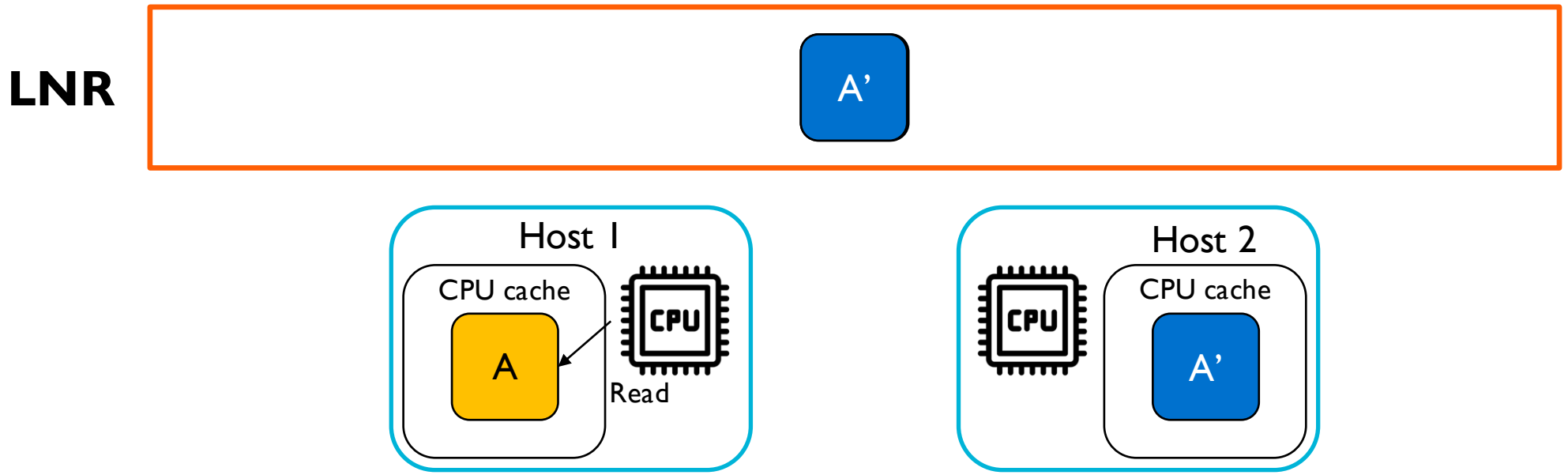


Problem:

No cross-host cache coherence for LNR, processor caches may hold stale values

→ correctness bugs when data shared across hosts

The Partly Coherent CXL Model

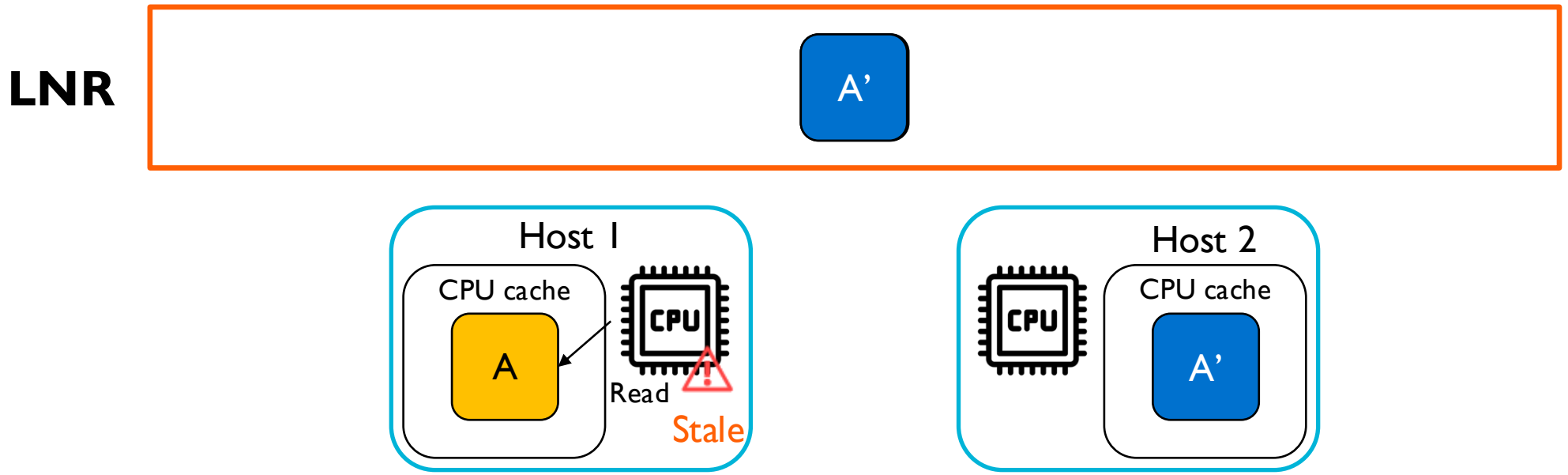


Problem:

No cross-host cache coherence for LNR, processor caches may hold stale values

→ correctness bugs when data shared across hosts

The Partly Coherent CXL Model



Problem:

No cross-host cache coherence for LNR, processor caches may hold stale values

→ correctness bugs when data shared across hosts

How to Coherently Share Data?

Small Coherent Region
A few hundred MBs
Hardware coherent

Large Non-Coherent Region
Several TBs
No hardware coherence

How to Coherently Share Data?

Small Coherent Region
A few hundred MBs
Hardware coherent

Large Non-Coherent Region
Several TBs
No hardware coherence

Restrict data sharing to SCR?

How to Coherently Share Data?

Small Coherent Region
A few hundred MBs
Hardware coherent

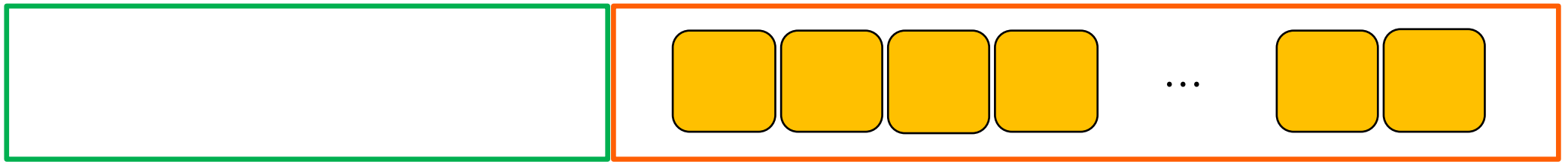
Large Non-Coherent Region
Several TBs
No hardware coherence

Restrict data sharing to SCR?

Too restrictive! Cannot utilize LNR capacity

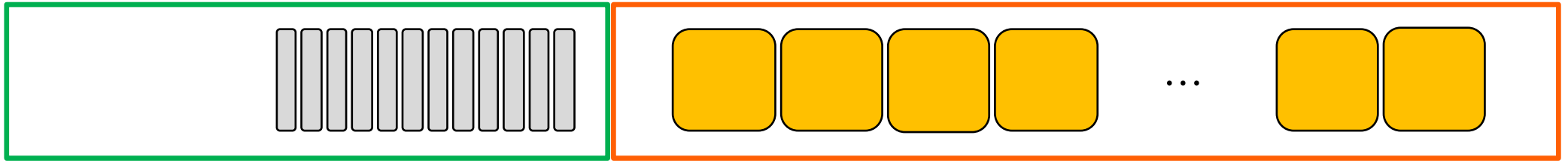
A Natural Approach: Software Coherence

A Natural Approach: Software Coherence



Data objects in LNR
(coarse-grained: e.g. DB rows,
not cacheline-granularity)

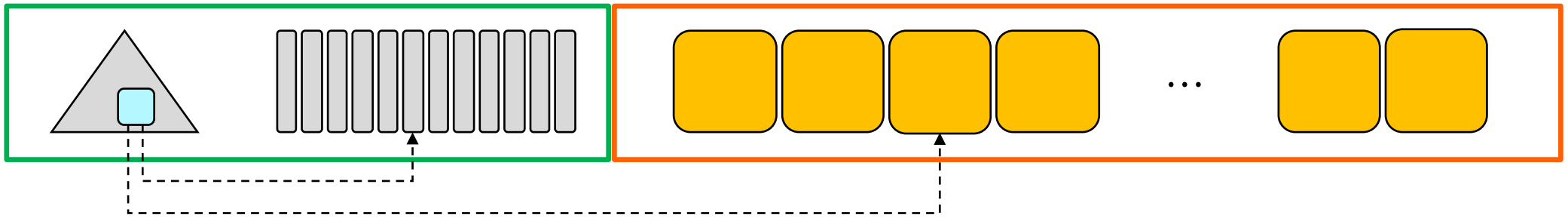
A Natural Approach: Software Coherence



Per-object coherence record in SCR

Data objects in LNR
(coarse-grained: e.g. DB rows,
not cacheline-granularity)

A Natural Approach: Software Coherence

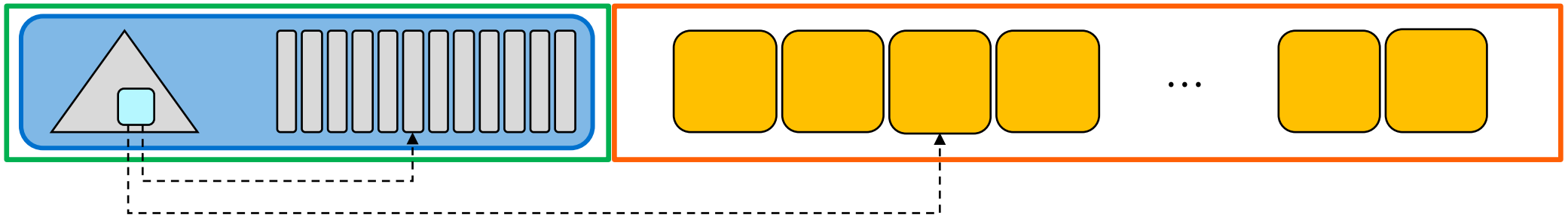


Per-object coherence record in SCR

Shared index in SCR:
obj ID → LNR location
+ coherence record location

Data objects in LNR
(coarse-grained: e.g. DB rows,
not cacheline-granularity)

A Natural Approach: Software Coherence



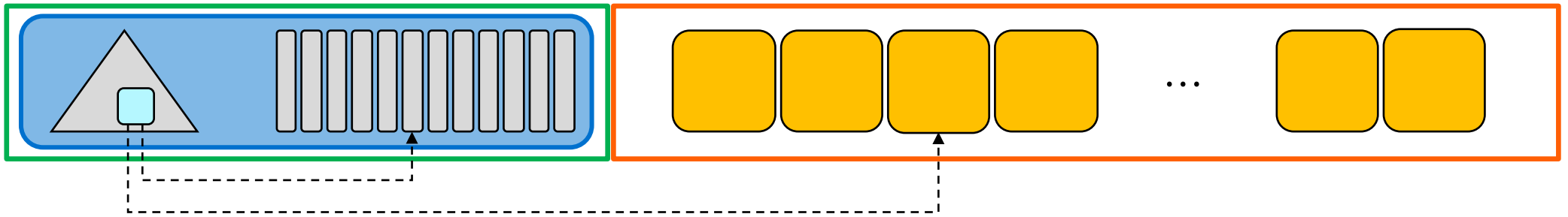
Per-object coherence record in SCR

Shared index in SCR:
obj ID → LNR location
+ coherence record location

Metadata

Data objects in LNR
(coarse-grained: e.g. DB rows,
not cacheline-granularity)

A Natural Approach: Software Coherence



Per-object coherence record in SCR

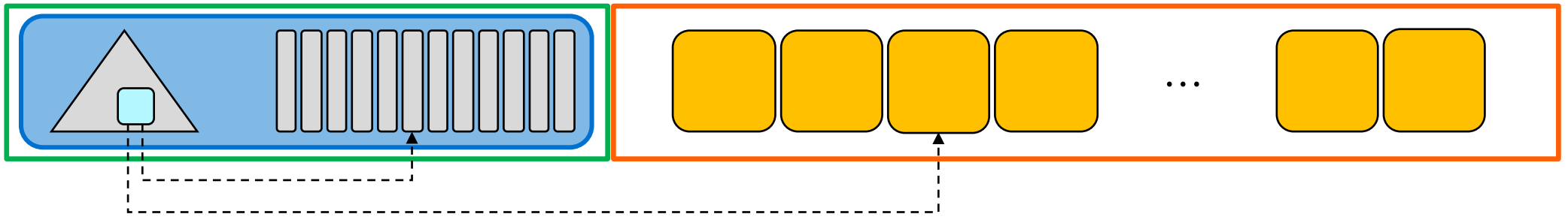
Shared index in SCR:
obj ID → LNR location
+ coherence record location

Metadata

Data objects in LNR
(coarse-grained: e.g. DB rows,
not cacheline-granularity)

HCMeta: Hardware-Coherent Metadata-Based Sharing

A Natural Approach: Software Coherence



Per-object coherence record in SCR

Shared index in SCR:
obj ID → LNR location
+ coherence record location

Metadata

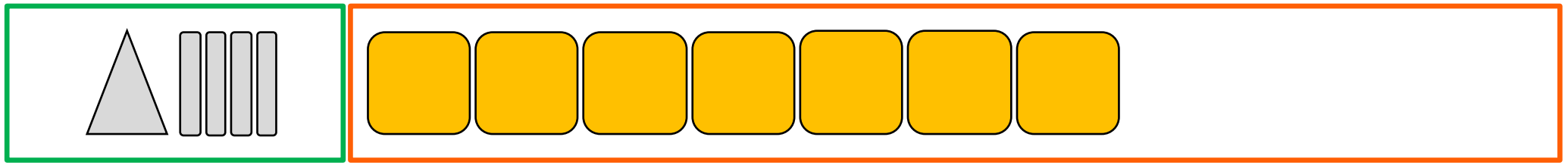
Data objects in LNR
(coarse-grained: e.g. DB rows,
not cacheline-granularity)

HCMeta: Hardware-Coherent Metadata-Based Sharing

Tigon[1], the first database for partly coherent CXL, uses a similar approach

[1] Tigon: A distributed database for a CXL pod. Yibo Huang, Haowei Chen, Newton Ni, Yan Sun, Vijay Chidambaram, Dixin Tang, and Emmett Witchel. OSDI '25

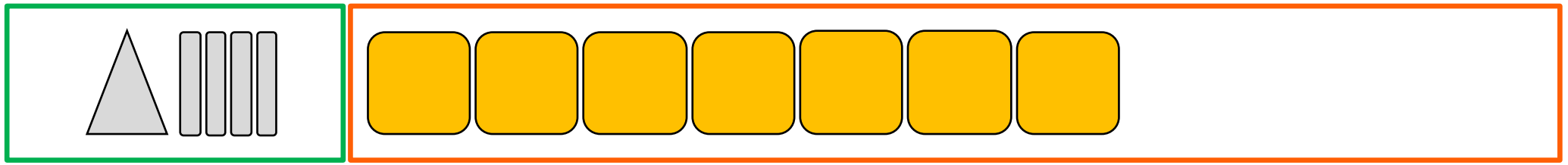
Drawbacks of HCMeta



A few hundred MBs

HCMeta allows sharing data objects beyond the limit of SCR capacity

Drawbacks of HCMeta



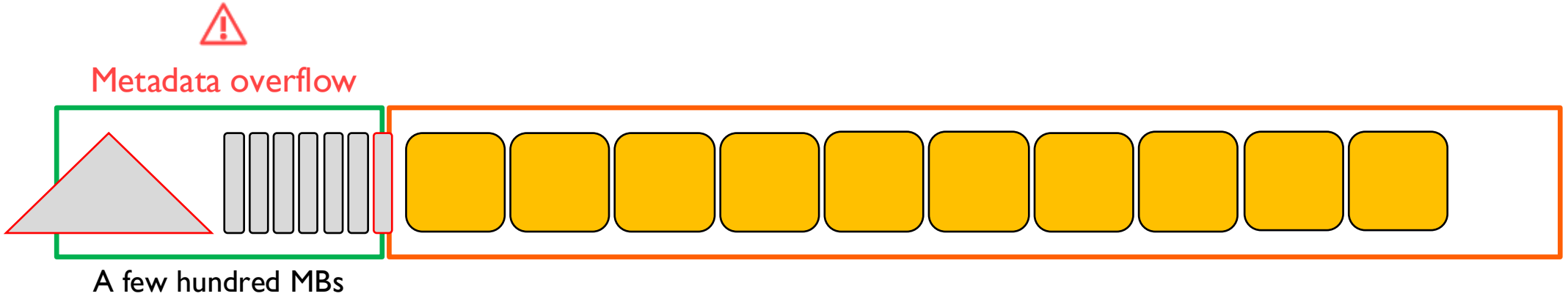
A few hundred MBs

HCMeta allows sharing data objects beyond the limit of SCR capacity

SCR capacity is tiny (a few hundred MBs)

- With a large dataset, metadata is 100x larger than SCR capacity

Drawbacks of HCMeta

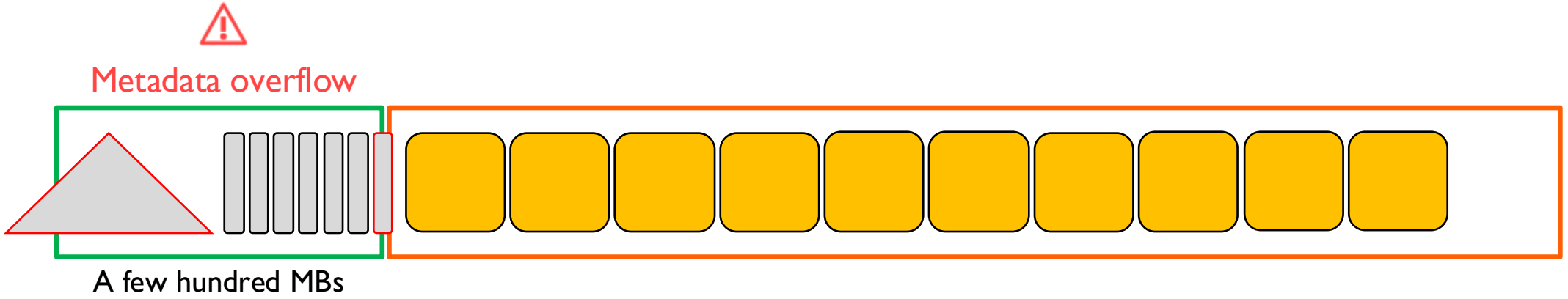


HCMeta allows sharing data objects beyond the limit of SCR capacity

SCR capacity is tiny (a few hundred MBs)

- With a large dataset, metadata is 100x larger than SCR capacity

Drawbacks of HCMeta



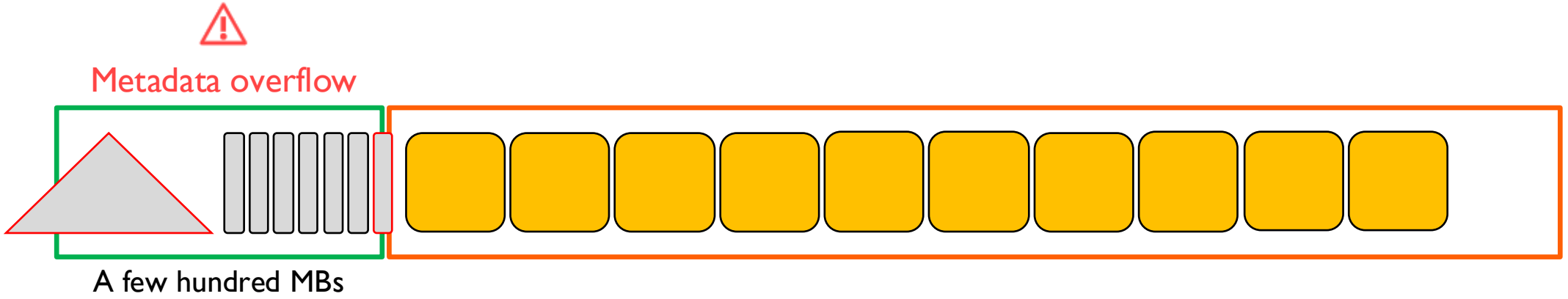
HCMeta allows sharing data objects beyond the limit of SCR capacity

SCR capacity is tiny (a few hundred MBs)

- With a large dataset, metadata is 100x larger than SCR capacity

However, metadata is crucial for correct sharing

Drawbacks of HCMeta



HCMeta allows sharing data objects beyond the limit of SCR capacity

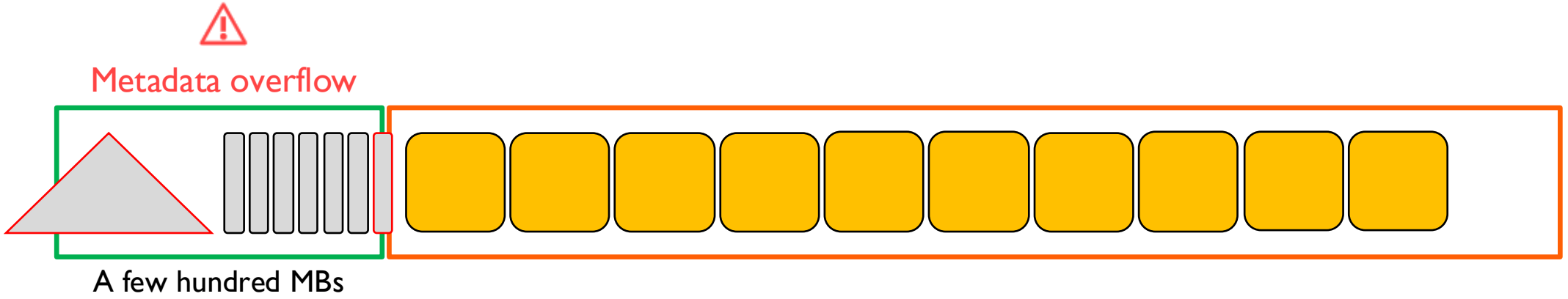
SCR capacity is tiny (a few hundred MBs)

- With a large dataset, metadata is 100x larger than SCR capacity

However, metadata is crucial for correct sharing

→ A limit on the number objects that can be shared

Drawbacks of HCMeta



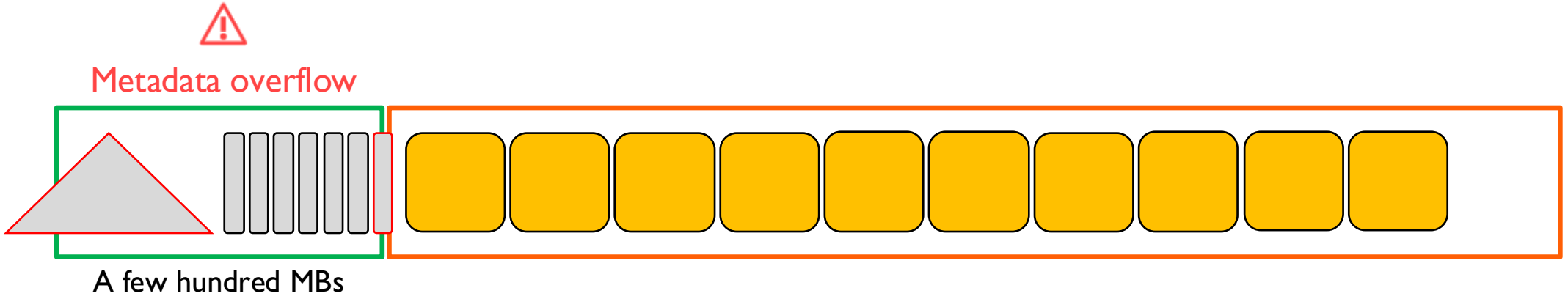
HCMeta allows sharing data objects beyond the limit of SCR capacity

SCR capacity is tiny (a few hundred MBs)
- With a large dataset, metadata is 100x larger than SCR capacity

However, metadata is crucial for correct sharing
→ A limit on the number objects that can be shared

Not an issue for Tigon because data sharing is occasional for cross-partition transactions

Drawbacks of HCMeta



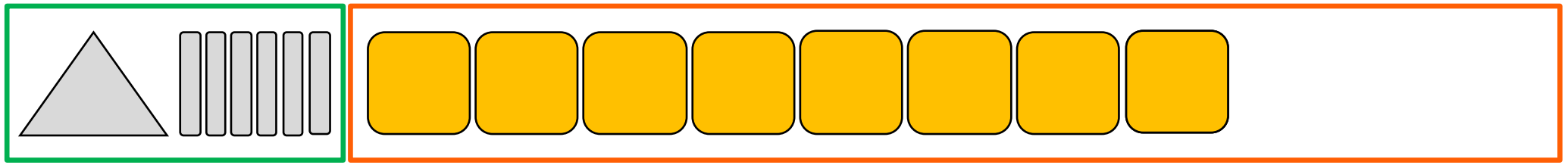
HCMeta allows sharing data objects beyond the limit of SCR capacity

SCR capacity is tiny (a few hundred MBs)
- With a large dataset, metadata is 100x larger than SCR capacity

However, metadata is crucial for correct sharing
→ A limit on the number objects that can be shared

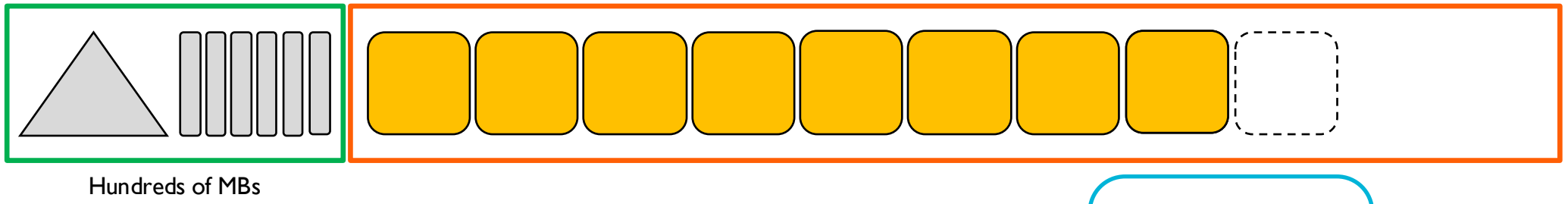
Not an issue for Tigon because data sharing is occasional for cross-partition transactions
But limiting for general sharing of large amount of data

How HCMeta Handles Metadata Overflow?



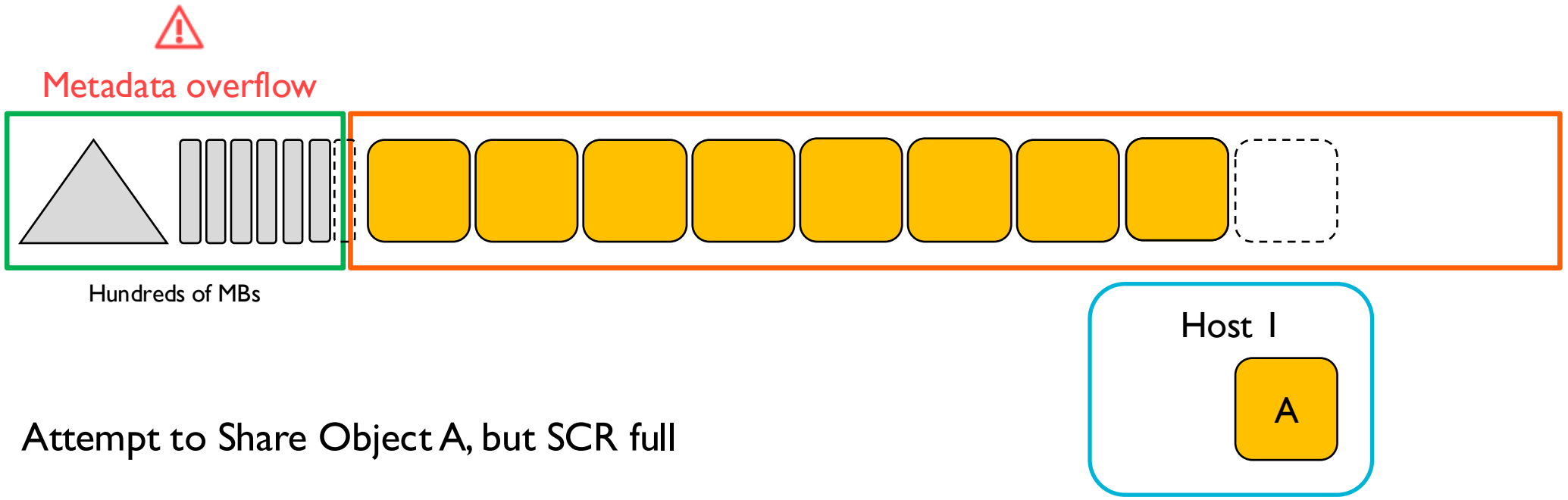
Hundreds of MBs

How HCMeta Handles Metadata Overflow?



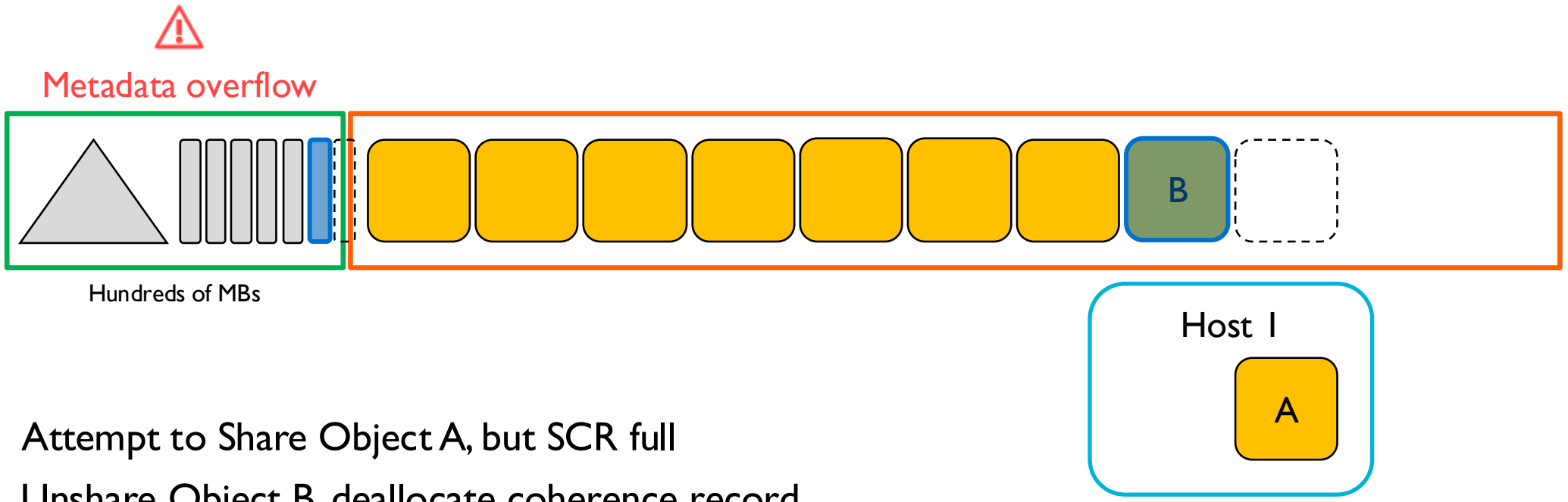
Attempt to Share Object A, but SCR full

How HCMeta Handles Metadata Overflow?



Attempt to Share Object A, but SCR full

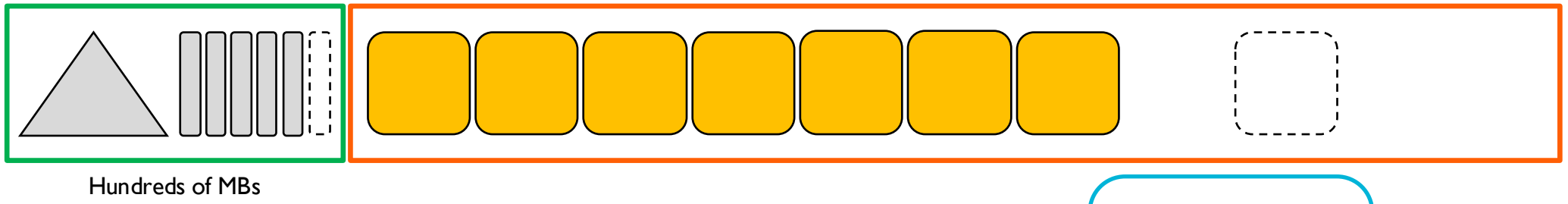
How HCMeta Handles Metadata Overflow?



Attempt to Share Object A, but SCR full

Unshare Object B, deallocate coherence record

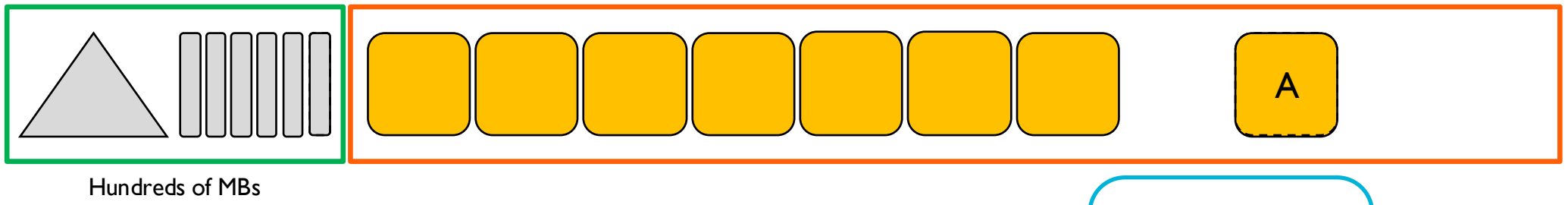
How HCMeta Handles Metadata Overflow?



Attempt to Share Object A, but SCR full

Unshare Object B, deallocate coherence record

How HCMeta Handles Metadata Overflow?

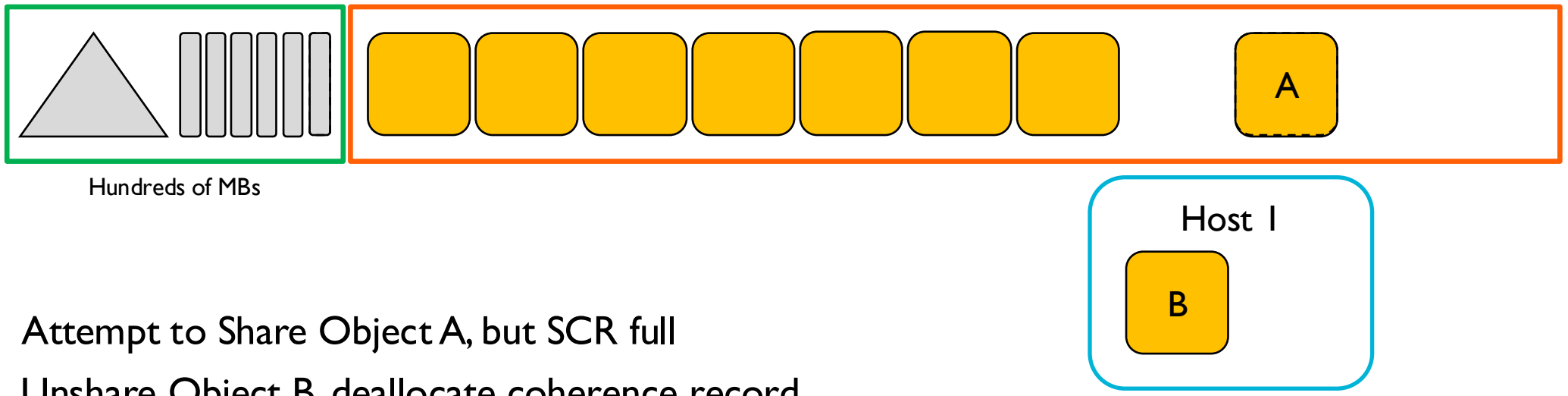


Attempt to Share Object A, but SCR full

Unshare Object B, deallocate coherence record

Use the freed-up space for coherence record for Object A

How HCMeta Handles Metadata Overflow?



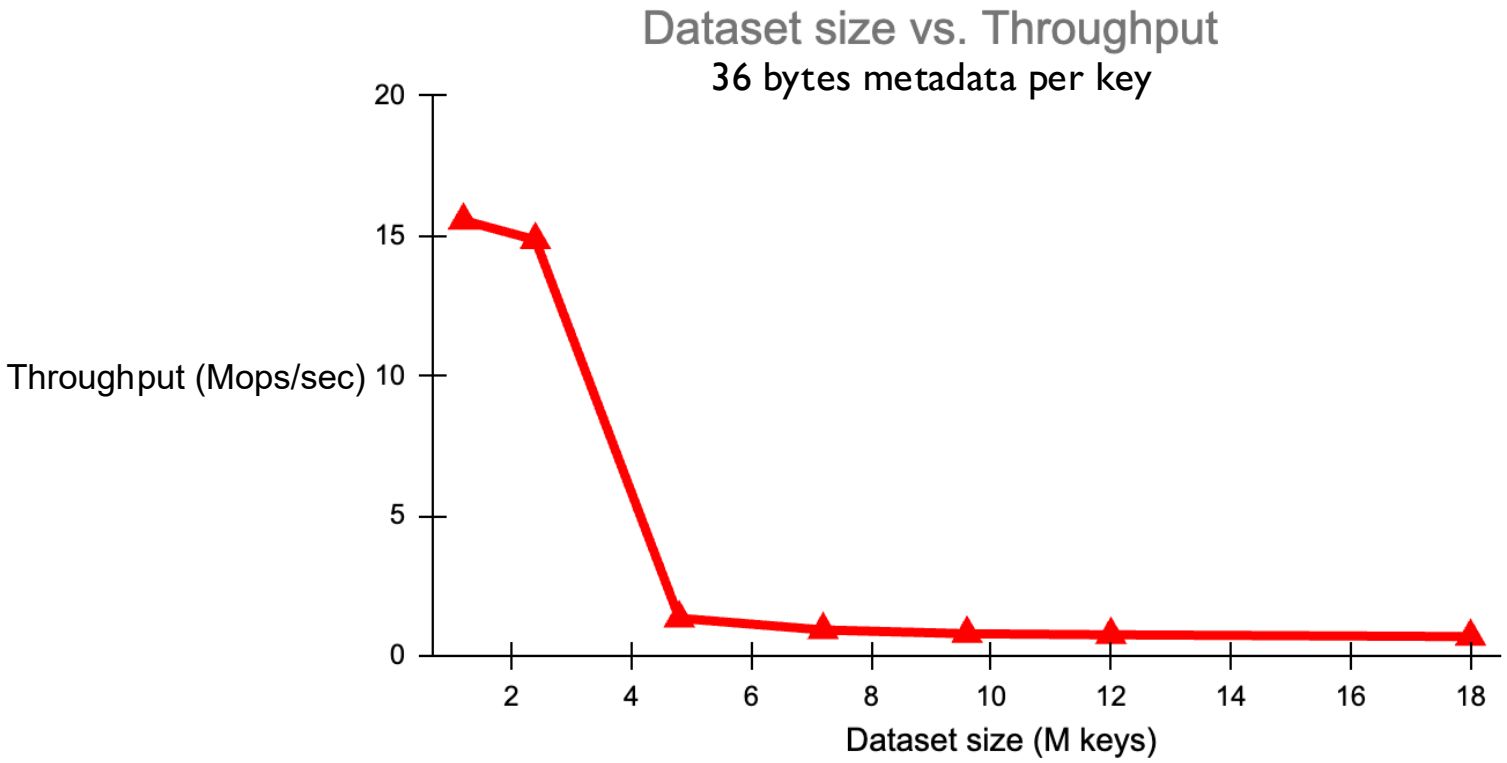
Attempt to Share Object A, but SCR full

Unshare Object B, deallocate coherence record

Use the freed-up space for coherence record for Object A

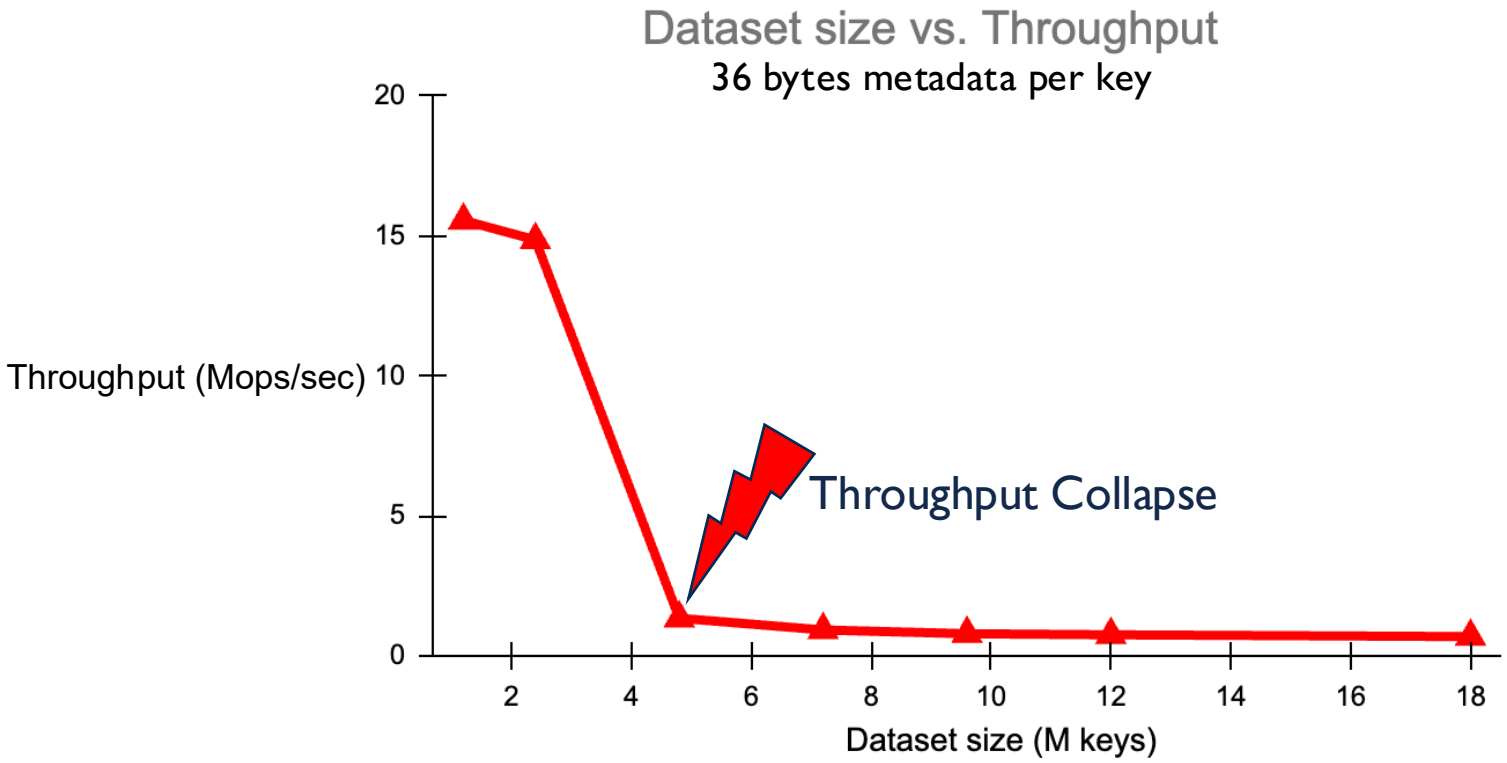
Such repeated sharing and unsharing (churns) will happen with large datasets

Impact on Performance



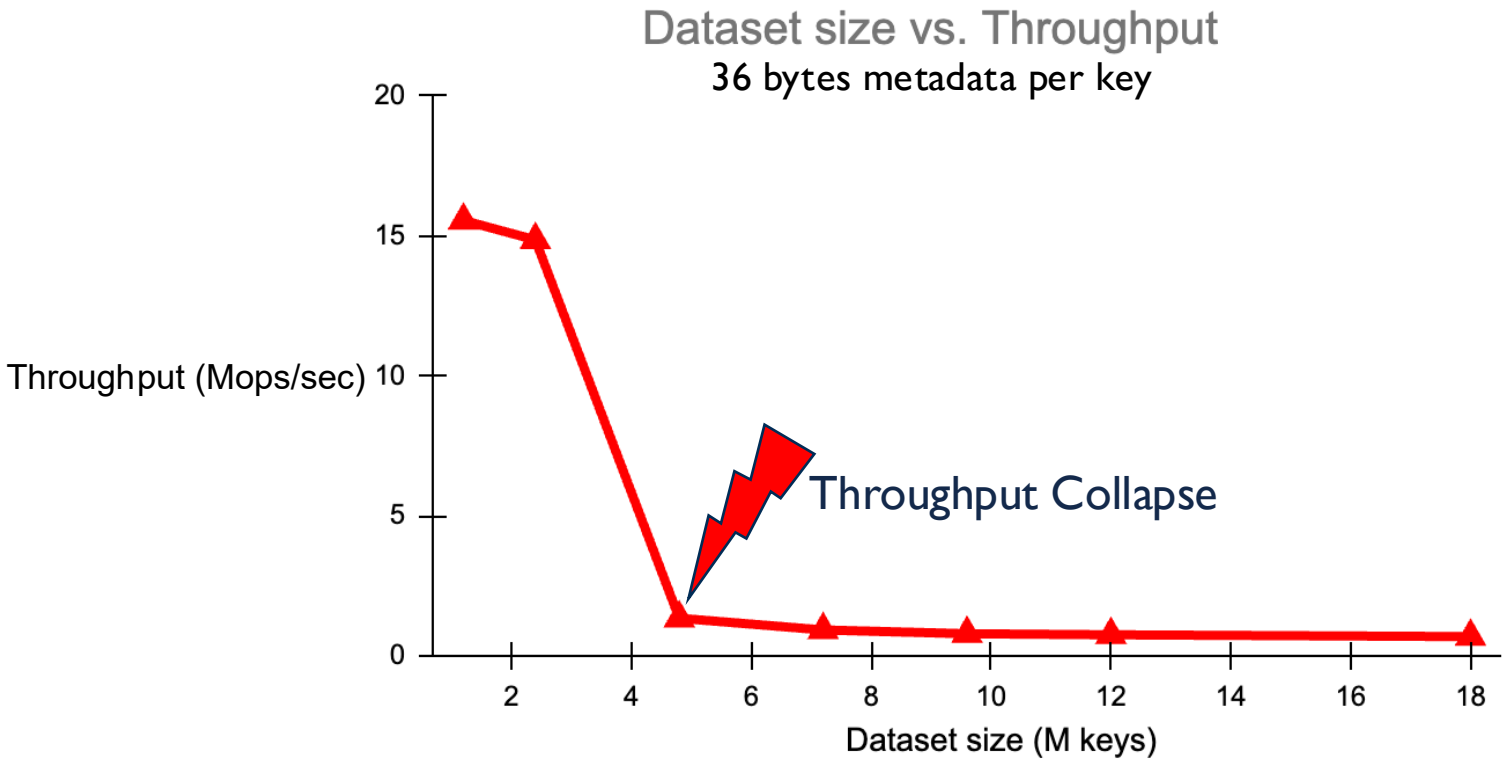
HCMeta: 100 MB SCR capacity

Impact on Performance



HCMeta: 100 MB SCR capacity

Impact on Performance



HCMeta: 100 MB SCR capacity

HCMeta is not performant when a large number of objects need to be shared

How can one efficiently share a large number of objects in CXL when even the metadata is too big to fit in SCR?

How can one efficiently share a large number of objects in CXL when even the metadata is too big to fit in SCR?

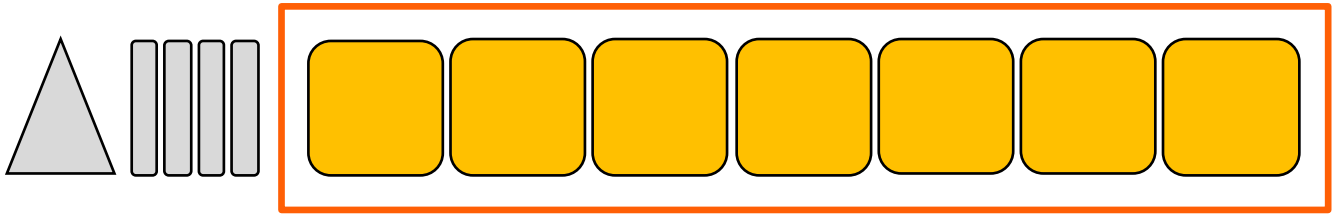
We answer this with:

MEGALON

A new data-sharing approach for partly coherent CXL

MEGALON Overview

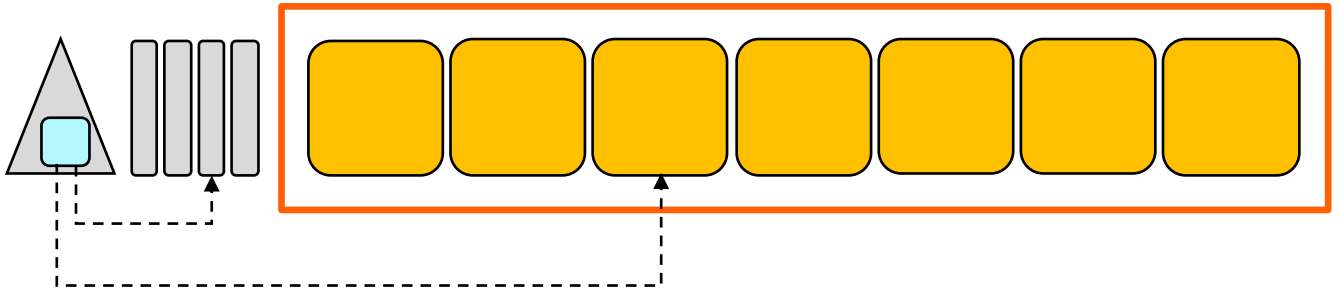
Like HCMeta, share data objects via shared metadata



MEGALON Overview

Like HCMeta, share data objects via shared metadata

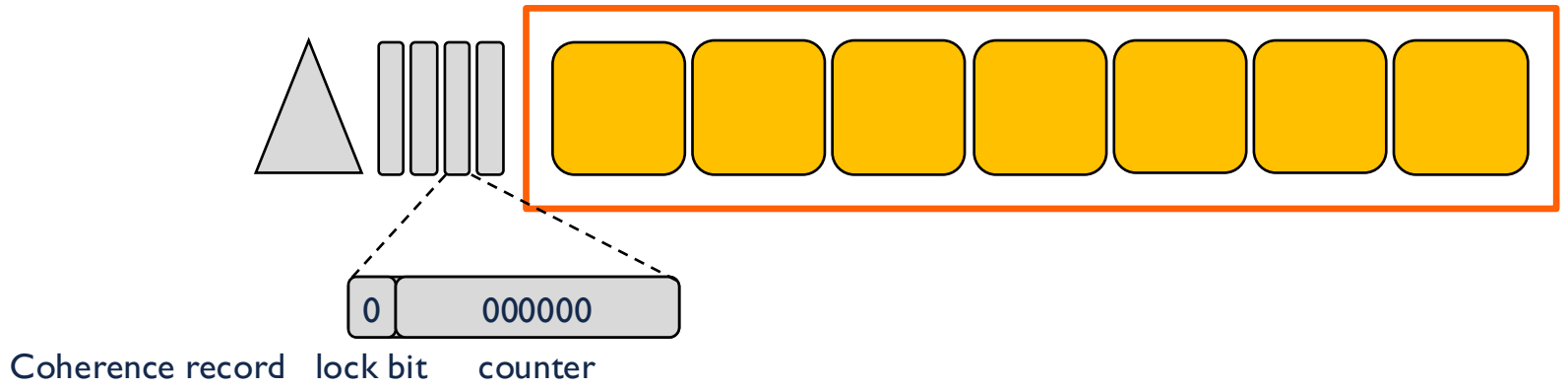
Index: object id → coherence record, object location in LNR



MEGALON Overview

Like HCMeta, share data objects via shared metadata

Index: object id → coherence record, object location in LNR



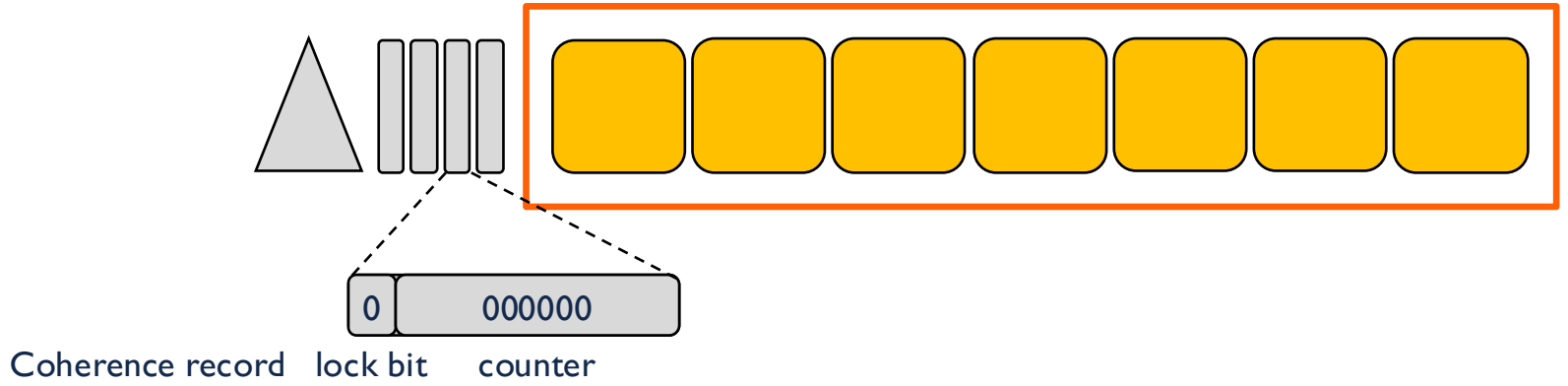
MEGALON Overview

Like HCMeta, share data objects via shared metadata

Index: object id → coherence record, object location in LNR

How software coherence works:

- Lock bit to serialize writes
- Writers increment the counter before and after the write
- Readers use the counter to detect read-write conflicts and stale processor caches (and then flush)



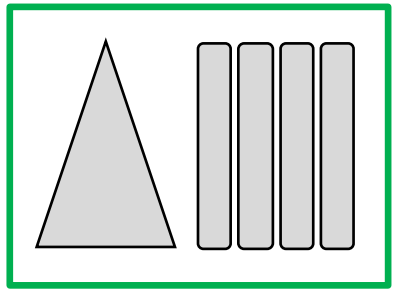
Key Idea

Novelty: How MEGALON shares the metadata

Key Idea

Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata



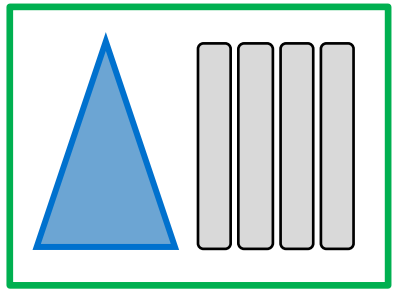
Key Idea

Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Index

Large but updated infrequently



Key Idea

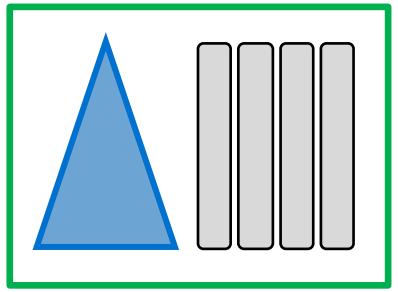
Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Index

Large but updated infrequently

- Contains object ID



Key Idea

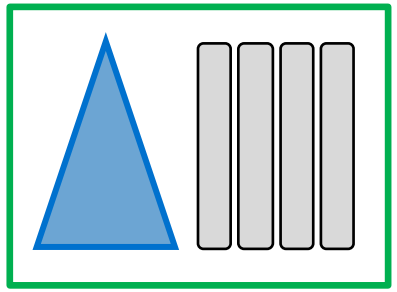
Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete



Key Idea

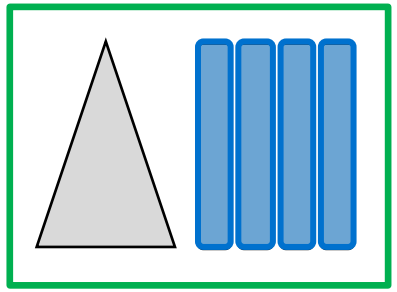
Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete



Coherence Record

Tiny but frequently updated

Key Idea

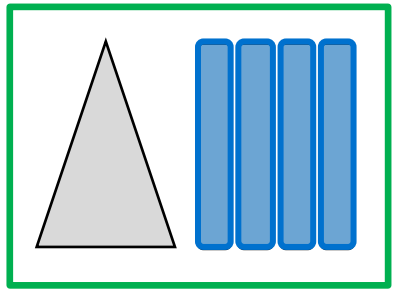
Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete



Coherence Record

Tiny but frequently updated

- Only lock bits and counter

Key Idea

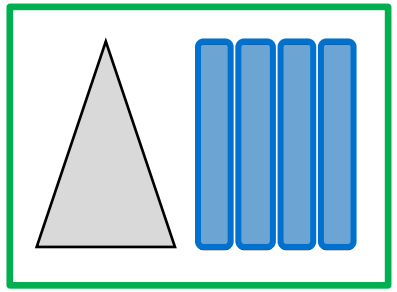
Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete



Coherence Record

Tiny but frequently updated

- Only lock bits and counter
- Updated on every object write

Key Idea

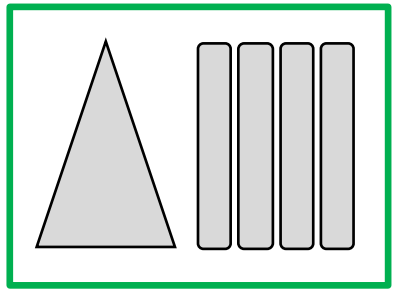
Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata HCMeta: Indistinguishably keeps all metadata in SCR

Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete



Coherence Record

Tiny but frequently updated

- Only lock bits and counter
- Updated on every object write

Key Idea

Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

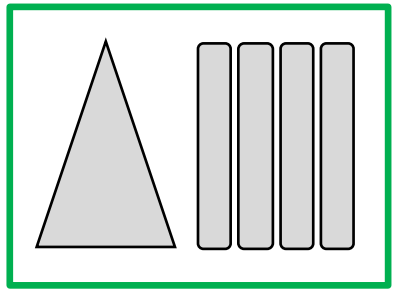
Key Idea: Split Metadata Sharing

HCMeta: Indistinguishably keeps all metadata in SCR

Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete



Coherence Record

Tiny but frequently updated

- Only lock bits and counter
- Updated on every object write

Key Idea

Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Key Idea: Split Metadata Sharing

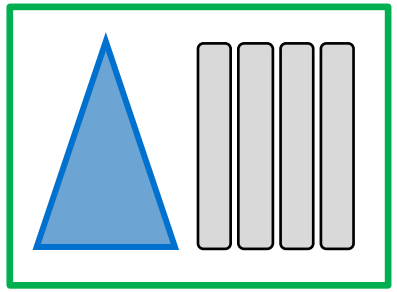
HCMeta: Indistinguishably keeps all metadata in SCR

Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete

Logically share - Replicate



Coherence Record

Tiny but frequently updated

- Only lock bits and counter
- Updated on every object write

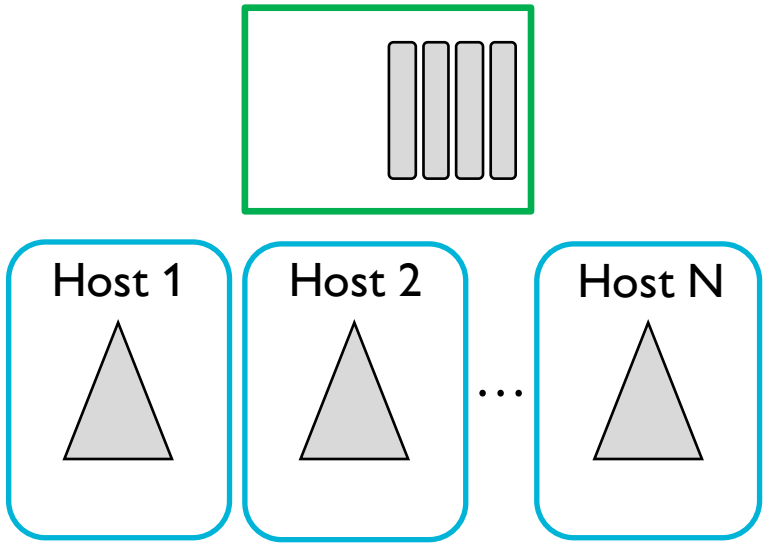
Key Idea

Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Key Idea: Split Metadata Sharing

HCMeta: Indistinguishably keeps all metadata in SCR



Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete

Logically share - Replicate

Coherence Record

Tiny but frequently updated

- Only lock bits and counter
- Updated on every object write

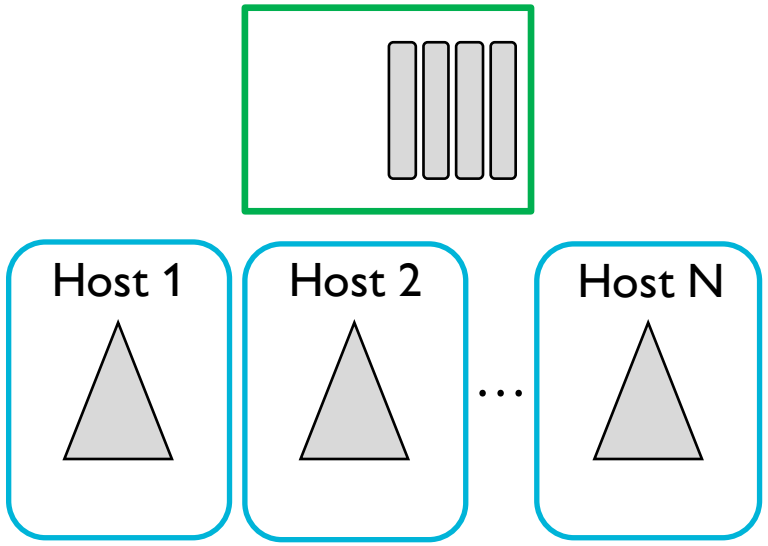
Key Idea

Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Key Idea: Split Metadata Sharing

HCMeta: Indistinguishably keeps all metadata in SCR



Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete

Logically share - Replicate

Hosts have ample DRAM

Coherence Record

Tiny but frequently updated

- Only lock bits and counter
- Updated on every object write

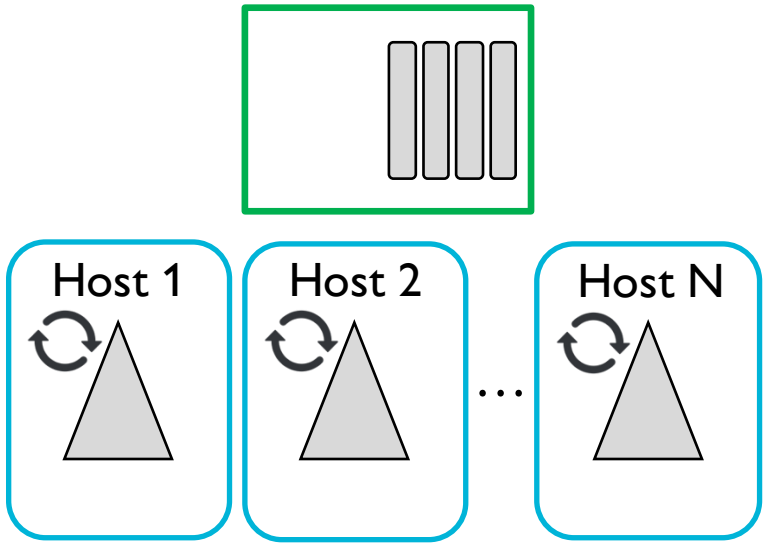
Key Idea

Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Key Idea: Split Metadata Sharing

HCMeta: Indistinguishably keeps all metadata in SCR



Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete

Logically share - Replicate

Hosts have ample DRAM

Cheap synchronization

Coherence Record

Tiny but frequently updated

- Only lock bits and counter
- Updated on every object write

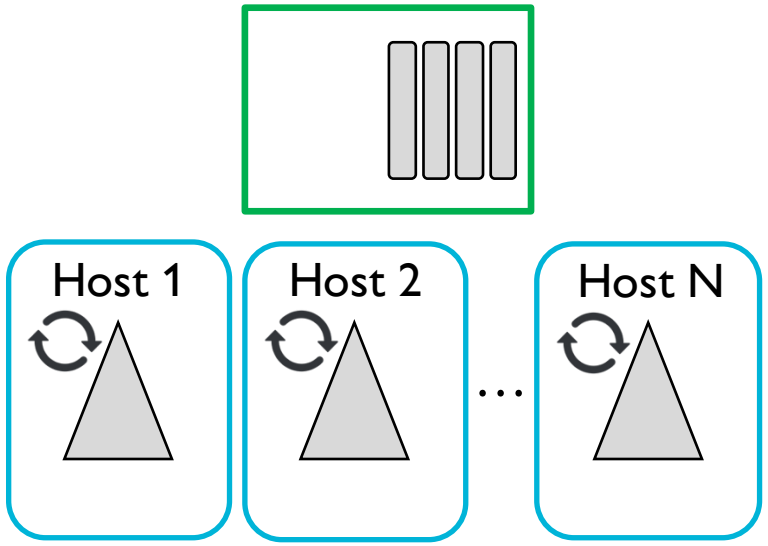
Key Idea

Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Key Idea: Split Metadata Sharing

HCMeta: Indistinguishably keeps all metadata in SCR



Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete

Logically share - Replicate

Hosts have ample DRAM

Cheap synchronization

Coherence Record

Tiny but frequently updated

- Only lock bits and counter
- Updated on every object write

Physically shared via SCR

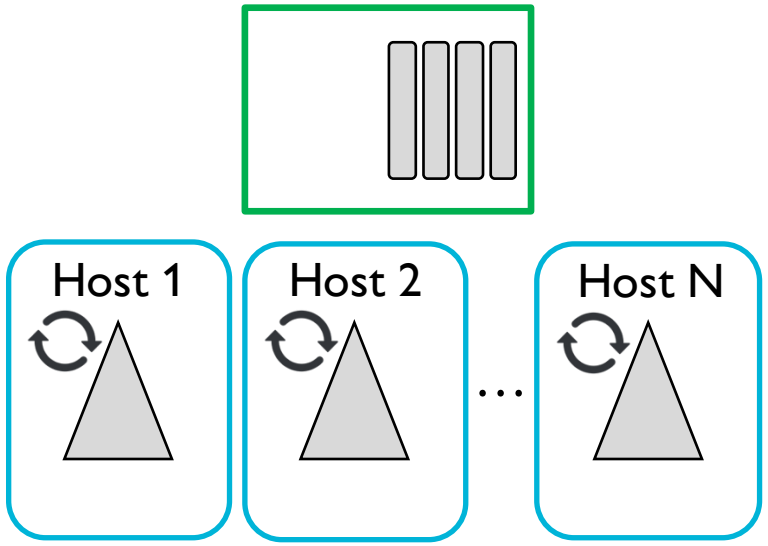
Key Idea

Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Key Idea: Split Metadata Sharing

HCMeta: Indistinguishably keeps all metadata in SCR



Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete

Logically share - Replicate

Hosts have ample DRAM

Cheap synchronization

Coherence Record

Tiny but frequently updated

- Only lock bits and counter
- Updated on every object write

Physically shared via SCR

Benefit from hardware coherence

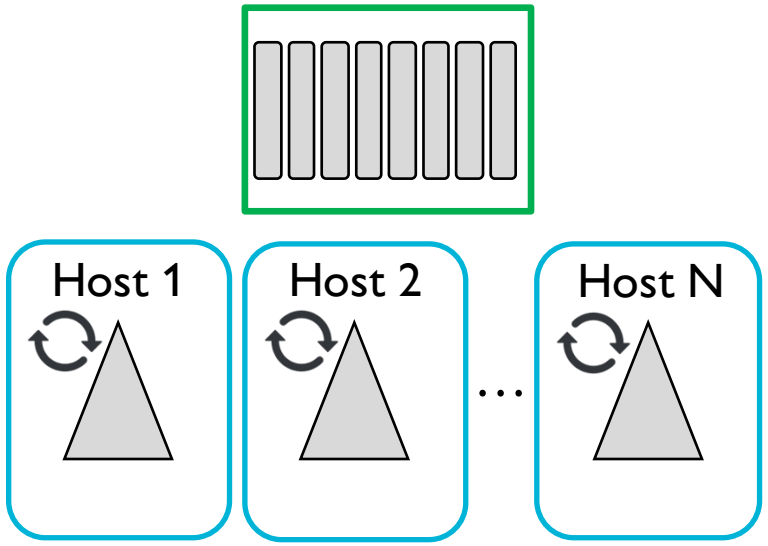
Key Idea

Novelty: How MEGALON shares the metadata

Observation: Two kinds of metadata

Key Idea: Split Metadata Sharing

HCMeta: Indistinguishably keeps all metadata in SCR



Index

Large but updated infrequently

- Contains object ID
- Only updated on object insert/delete

Logically share - Replicate

Hosts have ample DRAM

Cheap synchronization

Coherence Record

Tiny but frequently updated

- Only lock bits and counter
- Updated on every object write

Physically shared via SCR

Benefit from hardware coherence

Result: Share significantly more objects than HCMeta

Challenges

Challenges

C1. How to keep the index replicas consistent?

Challenges

C1. How to keep the index replicas consistent?

C2. What to do when SCR for coherence records runs out?

Challenges

C1. How to keep the index replicas consistent?

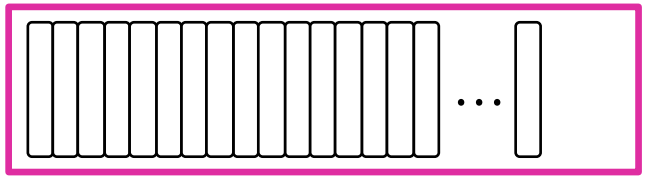
C2. What to do when SCR for coherence records runs out?

MEGALON's solution:

A Shared Log in CXL

What is a Shared Log?

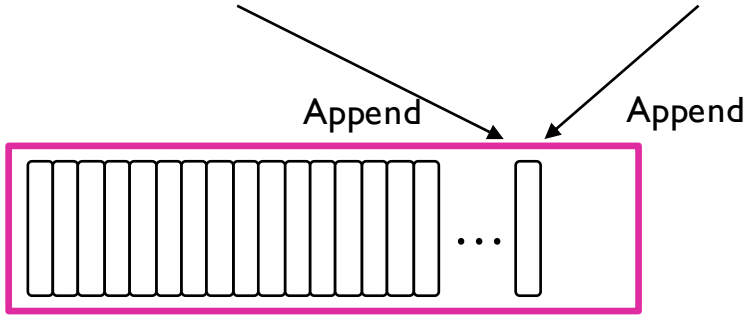
Shared log: linearizably ordered sequence of records



What is a Shared Log?

Shared log: linearizably ordered sequence of records

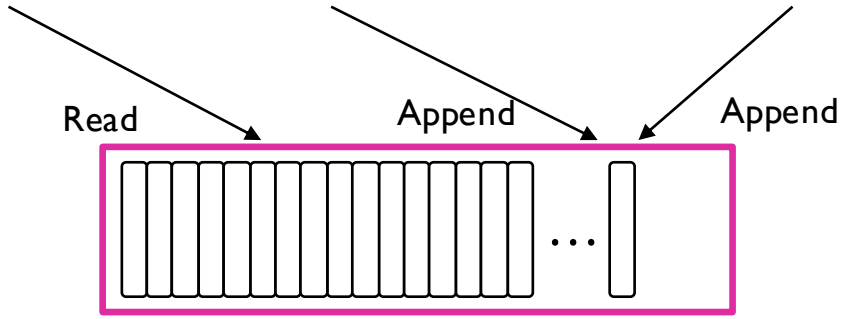
- Append to the tail



What is a Shared Log?

Shared log: linearizably ordered sequence of records

- Append to the tail
- Read from anywhere

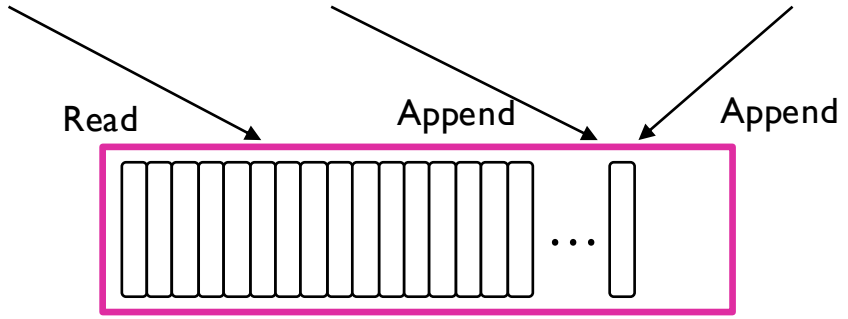


What is a Shared Log?

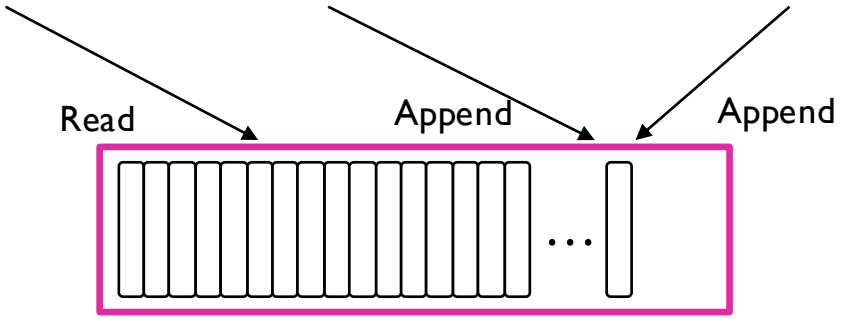
Shared log: linearizably ordered sequence of records

- Append to the tail
- Read from anywhere

State machine replication can be built on top of shared log

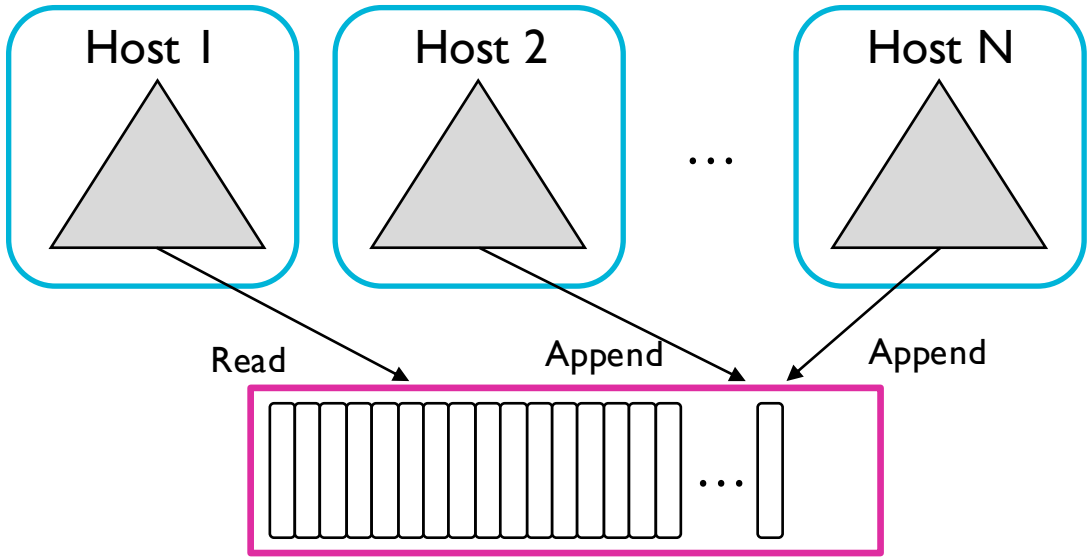


C1. Index Consistency on top of Shared Log



C1. Index Consistency on top of Shared Log

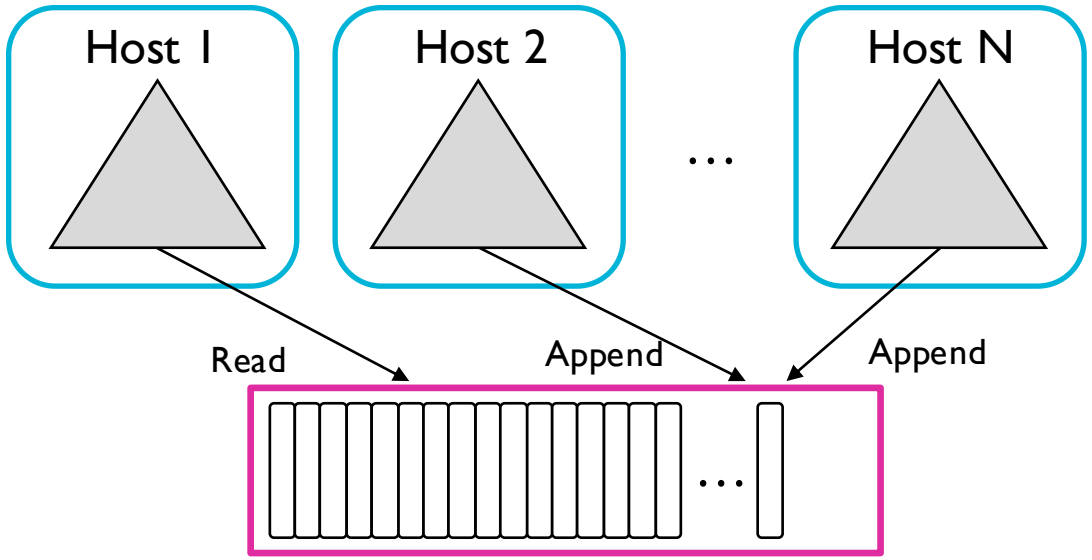
Use shared log to synchronize logical index replicas through state machine replication



C1. Index Consistency on top of Shared Log

Use shared log to synchronize logical index replicas through state machine replication

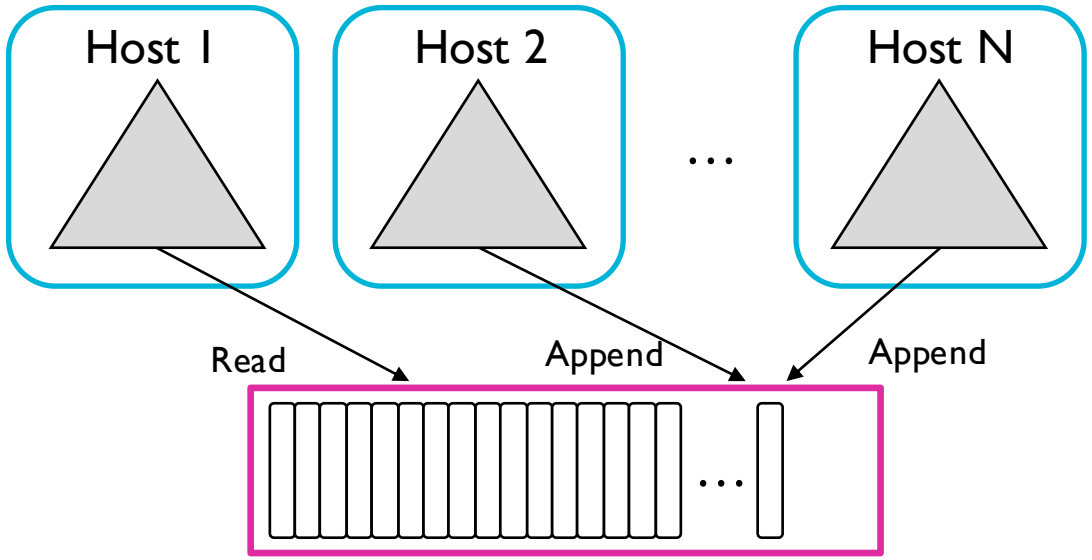
- All index operations ordered through shared log



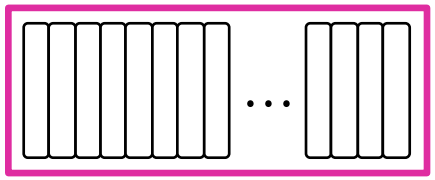
C1. Index Consistency on top of Shared Log

Use shared log to synchronize logical index replicas through state machine replication

- All index operations ordered through shared log
- Synchronize by replaying log



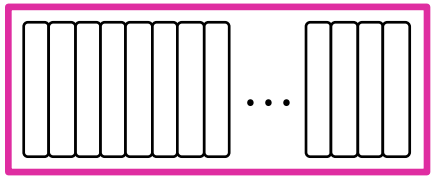
How does MEGALON implement the Shared Log efficiently?



Shared log

How does MEGALON implement the Shared Log efficiently?

Normally, shared log synchronized through distributed protocols



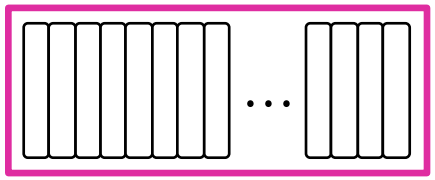
Shared log

How does MEGALON implement the Shared Log efficiently?

Normally, shared log synchronized through distributed protocols

Inspired by Node Replication[2]

- Build data structures in single-machine NUMA architecture through in-memory shared log



Shared log

[2] Black-box Concurrent Data Structures for NUMA Architectures. Irina Calciu, Siddhartha Sen, Mahesh Balakrishnan, and Marcos K. Aguilera. ASPLOS'17

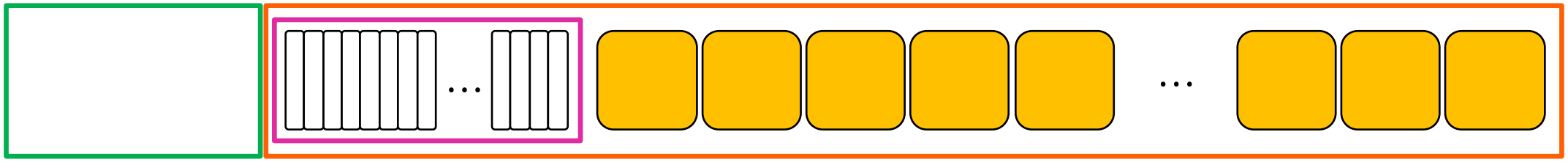
How does MEGALON implement the Shared Log efficiently?

Normally, shared log synchronized through distributed protocols

Inspired by Node Replication[2]

- Build data structures in single-machine NUMA architecture through in-memory shared log

Put shared log in CXL, multiple hosts can access



Shared log

[2] Black-box Concurrent Data Structures for NUMA Architectures. Irina Calciu, Siddhartha Sen, Mahesh Balakrishnan, and Marcos K. Aguilera. ASPLOS'17

How does MEGALON implement the Shared Log efficiently?

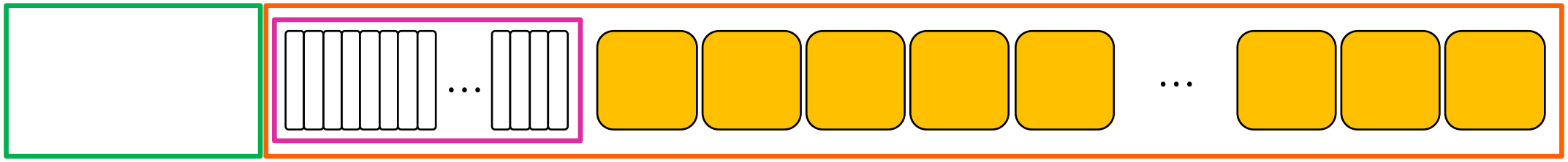
Normally, shared log synchronized through distributed protocols

Inspired by Node Replication[2]

- Build data structures in single-machine NUMA architecture through in-memory shared log

Put shared log in CXL, multiple hosts can access

- Faster than distributed protocols



Shared log

[2] Black-box Concurrent Data Structures for NUMA Architectures. Irina Calciu, Siddhartha Sen, Mahesh Balakrishnan, and Marcos K. Aguilera. ASPLOS'17

How does MEGALON implement the Shared Log efficiently?

Normally, shared log synchronized through distributed protocols

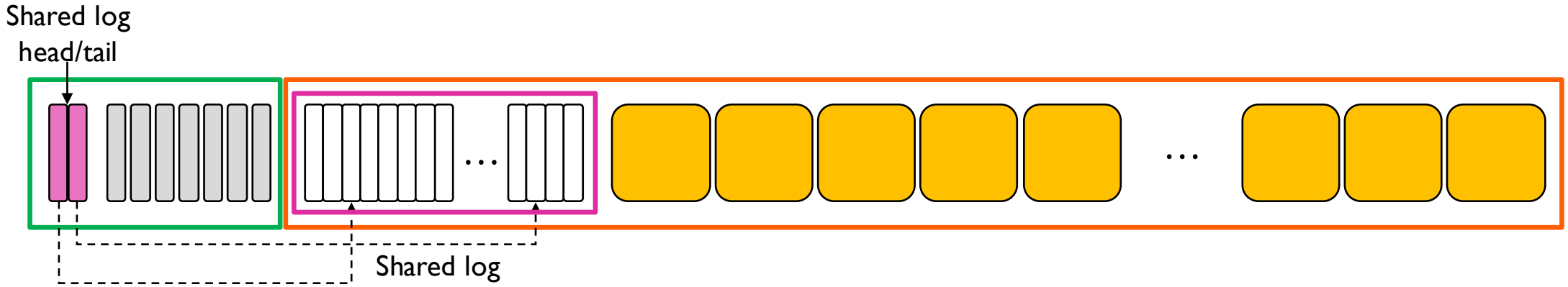
Inspired by Node Replication[2]

- Build data structures in single-machine NUMA architecture through in-memory shared log

Put shared log in CXL, multiple hosts can access

- Faster than distributed protocols

Only the head/tail in SCR



[2] Black-box Concurrent Data Structures for NUMA Architectures. Irina Calciu, Siddhartha Sen, Mahesh Balakrishnan, and Marcos K. Aguilera. ASPLOS'17

How does MEGALON implement the Shared Log efficiently?

Normally, shared log synchronized through distributed protocols

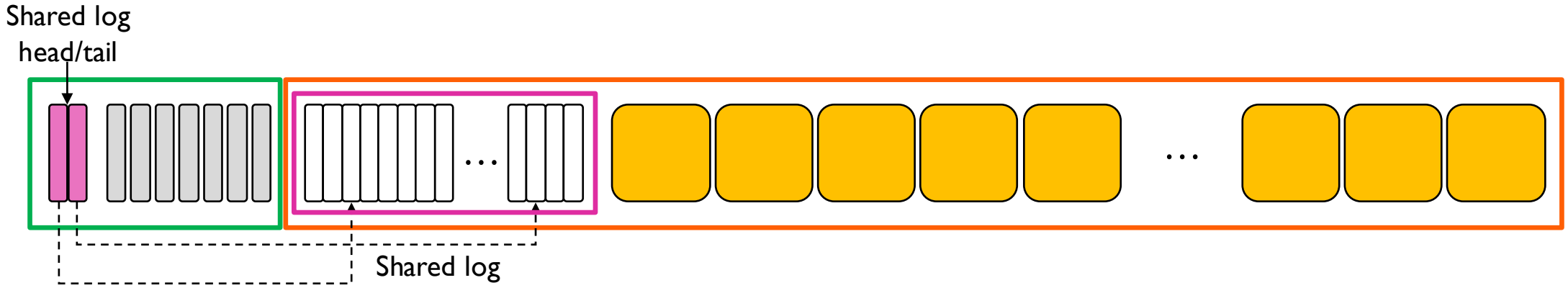
Inspired by Node Replication[2]

- Build data structures in single-machine NUMA architecture through in-memory shared log

Put shared log in CXL, multiple hosts can access

- Faster than distributed protocols

Only the head/tail in SCR → Hosts can detect index changes efficiently by checking the tail



[2] Black-box Concurrent Data Structures for NUMA Architectures. Irina Calciu, Siddhartha Sen, Mahesh Balakrishnan, and Marcos K. Aguilera. ASPLOS'17

How does MEGALON implement the Shared Log efficiently?

Normally, shared log synchronized through distributed protocols

Inspired by Node Replication[2]

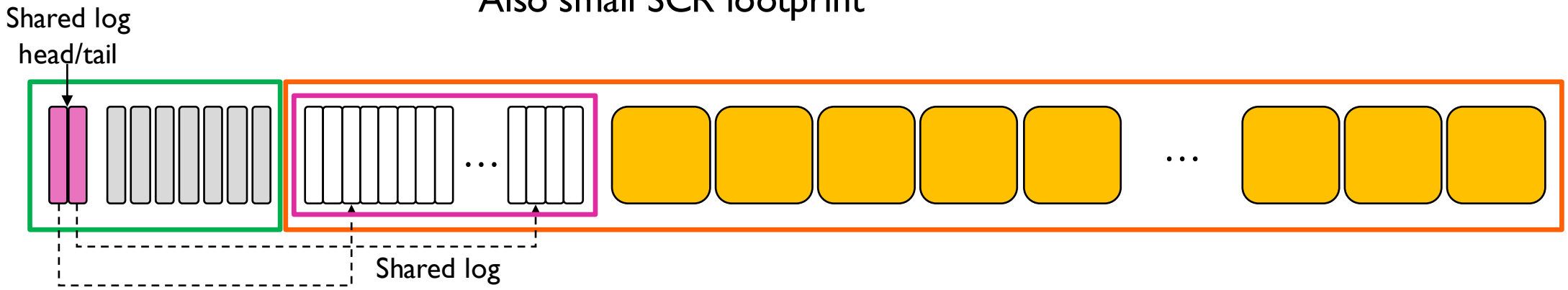
- Build data structures in single-machine NUMA architecture through in-memory shared log

Put shared log in CXL, multiple hosts can access

- Faster than distributed protocols

Only the head/tail in SCR → Hosts can detect index changes efficiently by checking the tail

Also small SCR footprint



[2] Black-box Concurrent Data Structures for NUMA Architectures. Irina Calciu, Siddhartha Sen, Mahesh Balakrishnan, and Marcos K. Aguilera. ASPLOS'17

Challenges

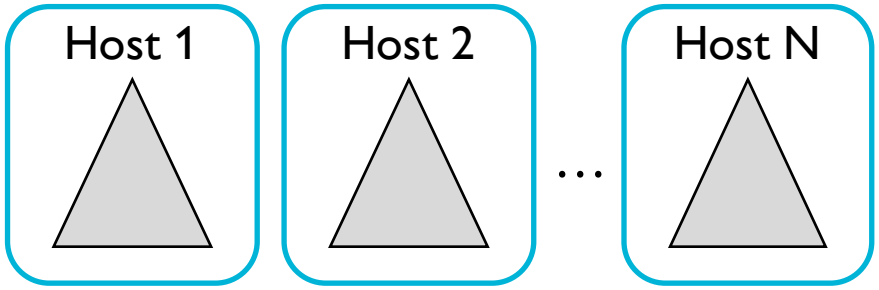
C1. How to keep the index replicas consistent?

C2. What to do when SCR for coherence records runs out?

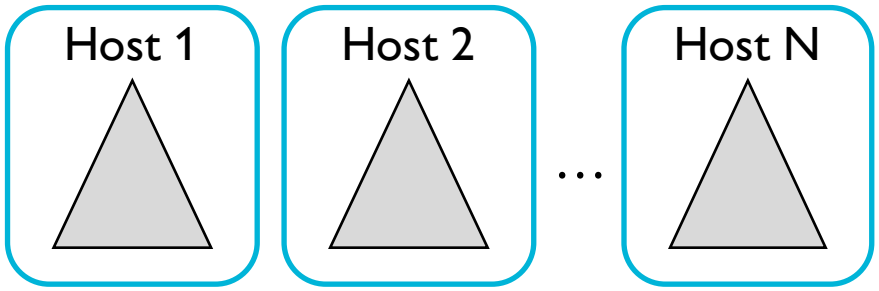
MEGALON's solution:

A Shared Log in CXL

Step 1: Dynamic Coherence Records

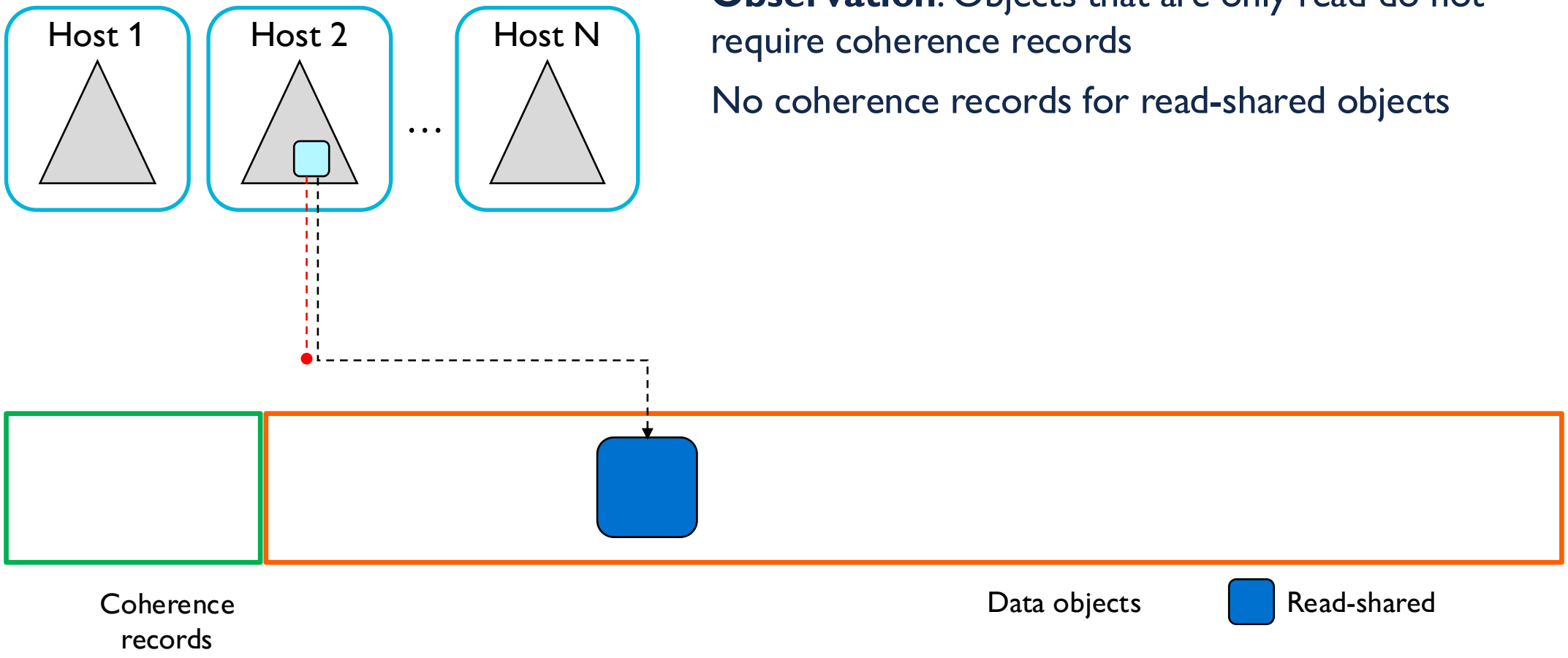


Step 1: Dynamic Coherence Records



Observation: Objects that are only read do not require coherence records

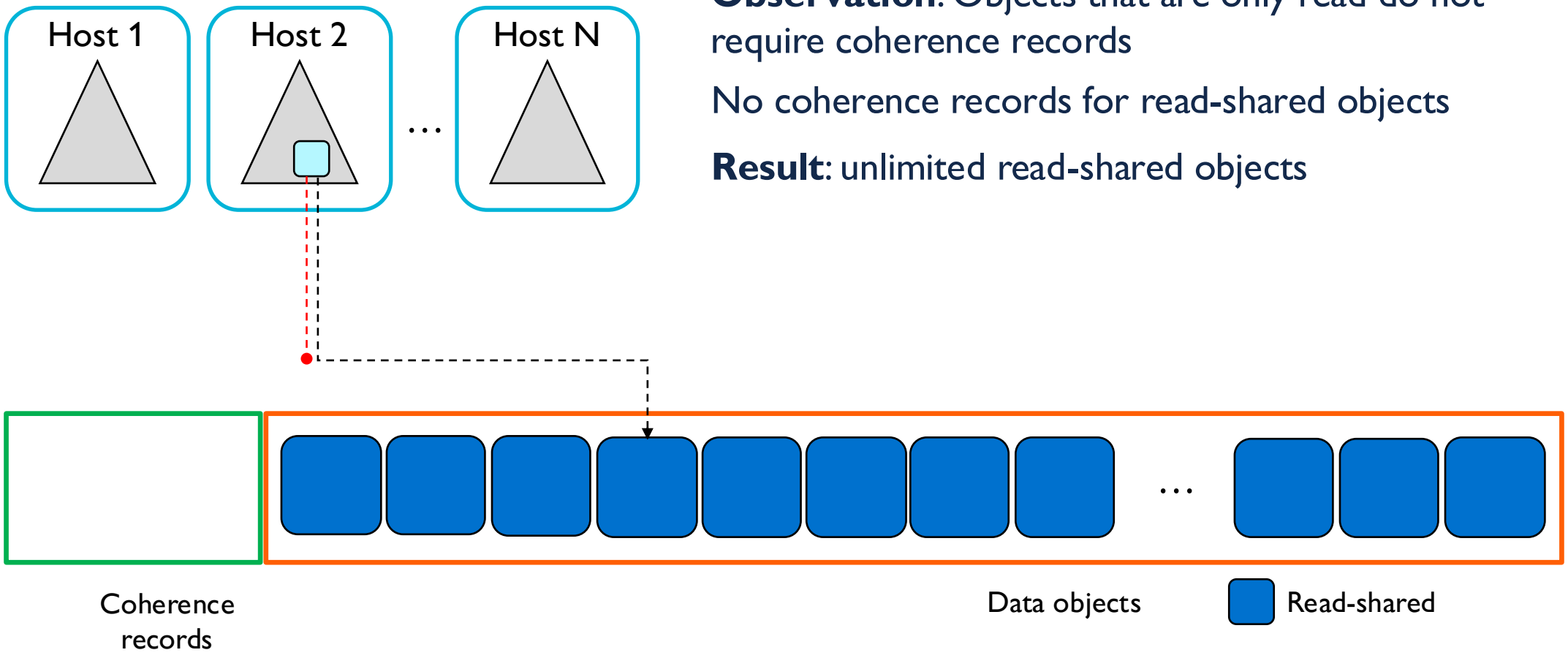
Step 1: Dynamic Coherence Records



Observation: Objects that are only read do not require coherence records

No coherence records for read-shared objects

Step 1: Dynamic Coherence Records

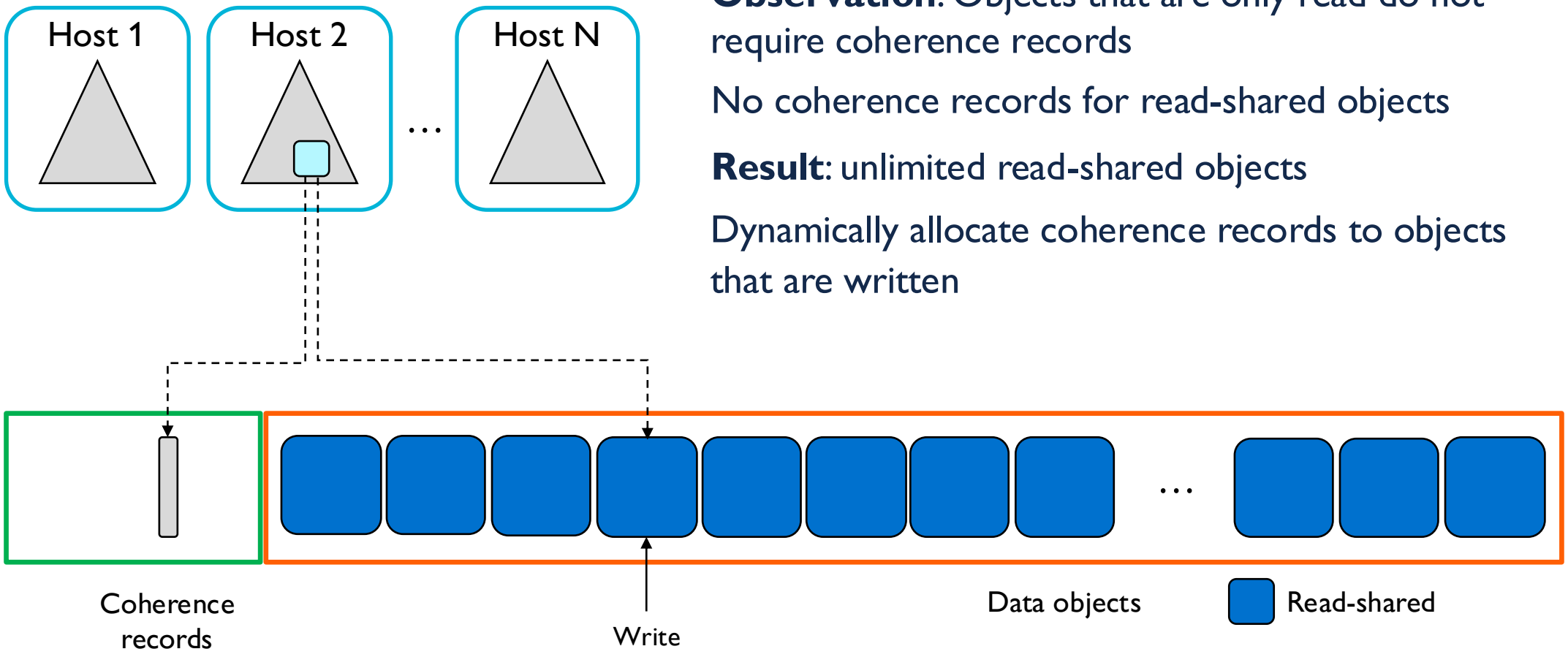


Observation: Objects that are only read do not require coherence records

No coherence records for read-shared objects

Result: unlimited read-shared objects

Step 1: Dynamic Coherence Records



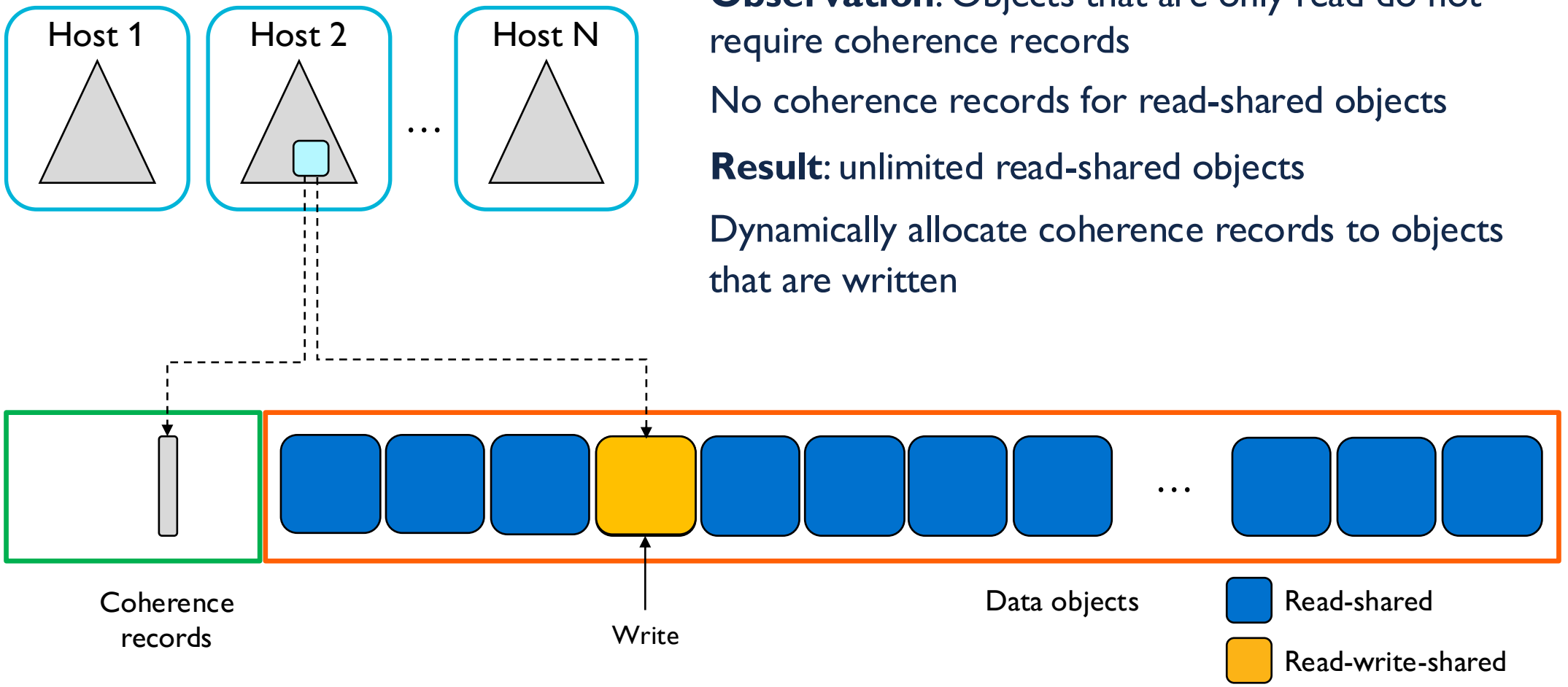
Observation: Objects that are only read do not require coherence records

No coherence records for read-shared objects

Result: unlimited read-shared objects

Dynamically allocate coherence records to objects that are written

Step 1: Dynamic Coherence Records



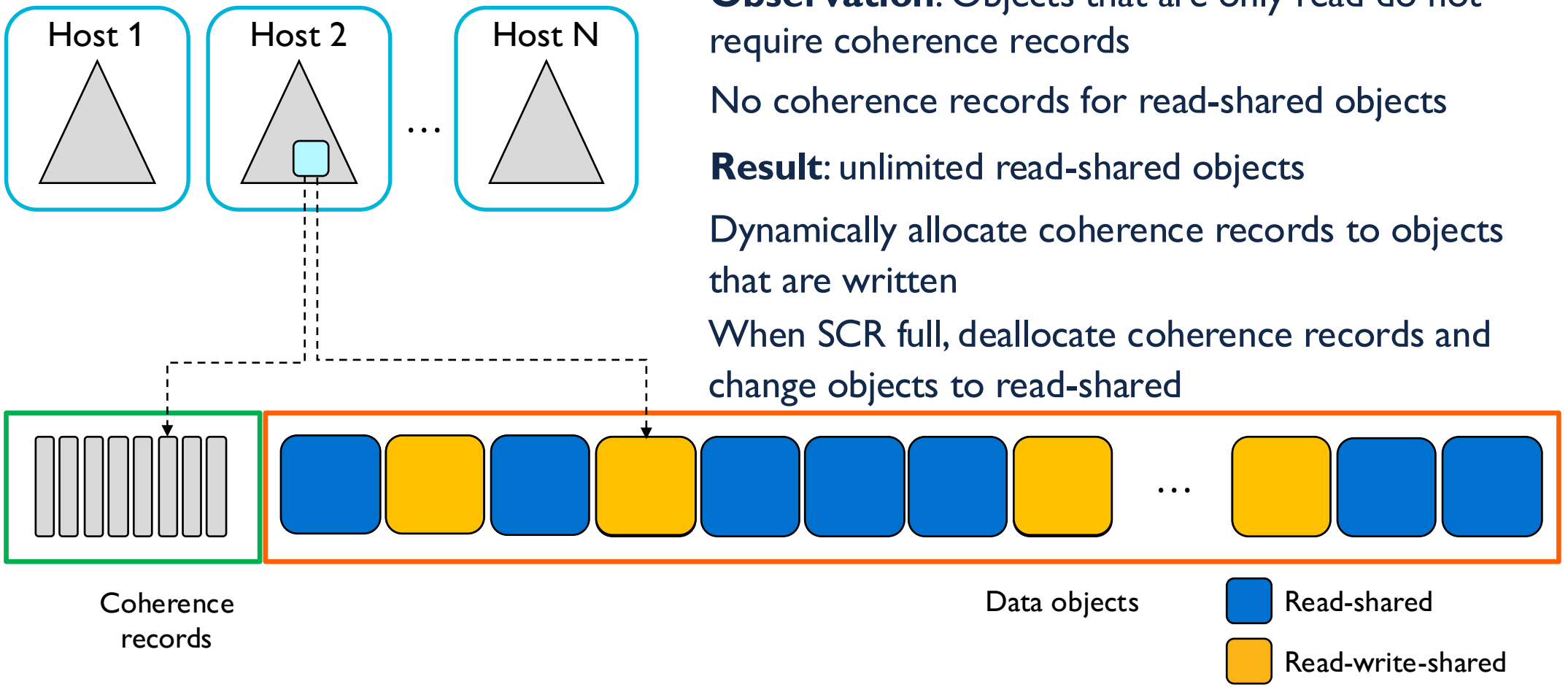
Observation: Objects that are only read do not require coherence records

No coherence records for read-shared objects

Result: unlimited read-shared objects

Dynamically allocate coherence records to objects that are written

Step 1: Dynamic Coherence Records



Observation: Objects that are only read do not require coherence records

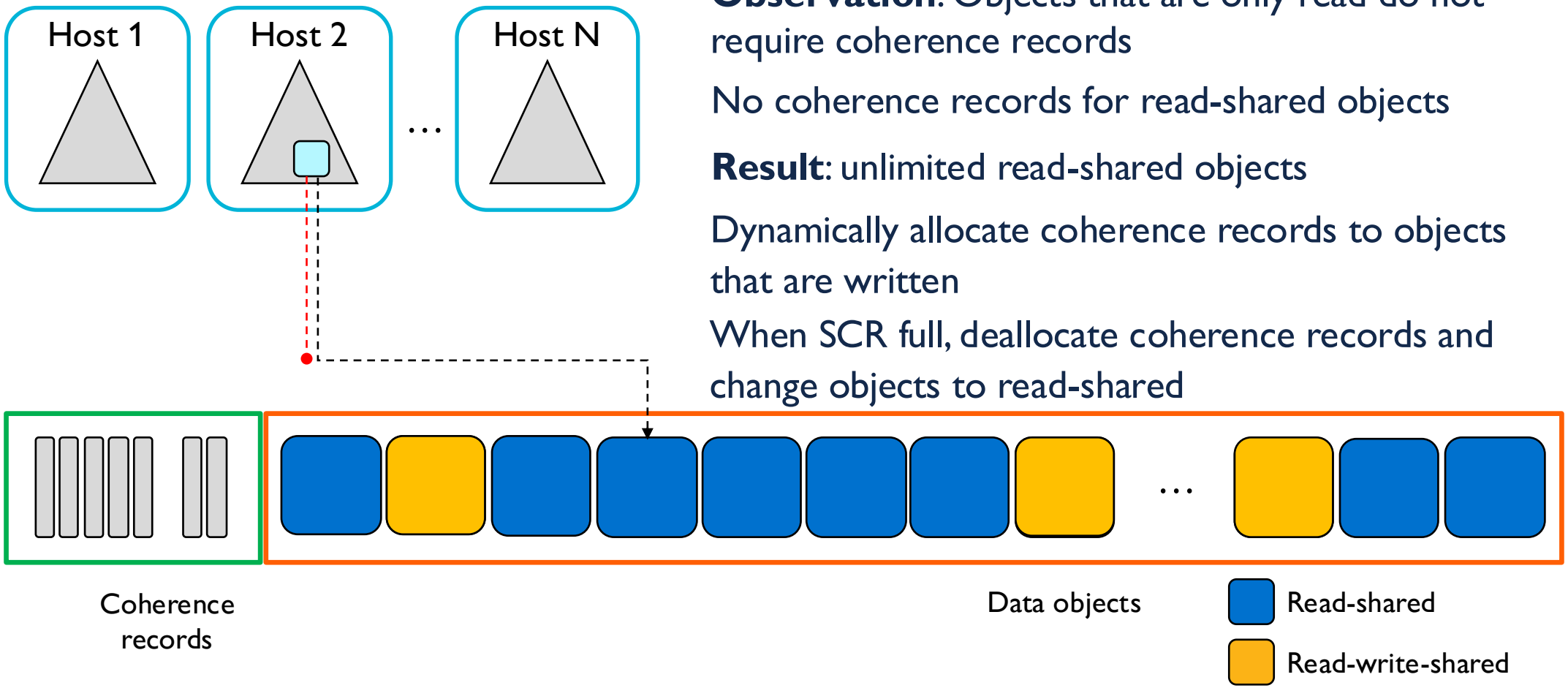
No coherence records for read-shared objects

Result: unlimited read-shared objects

Dynamically allocate coherence records to objects that are written

When SCR full, deallocate coherence records and change objects to read-shared

Step 1: Dynamic Coherence Records



Observation: Objects that are only read do not require coherence records

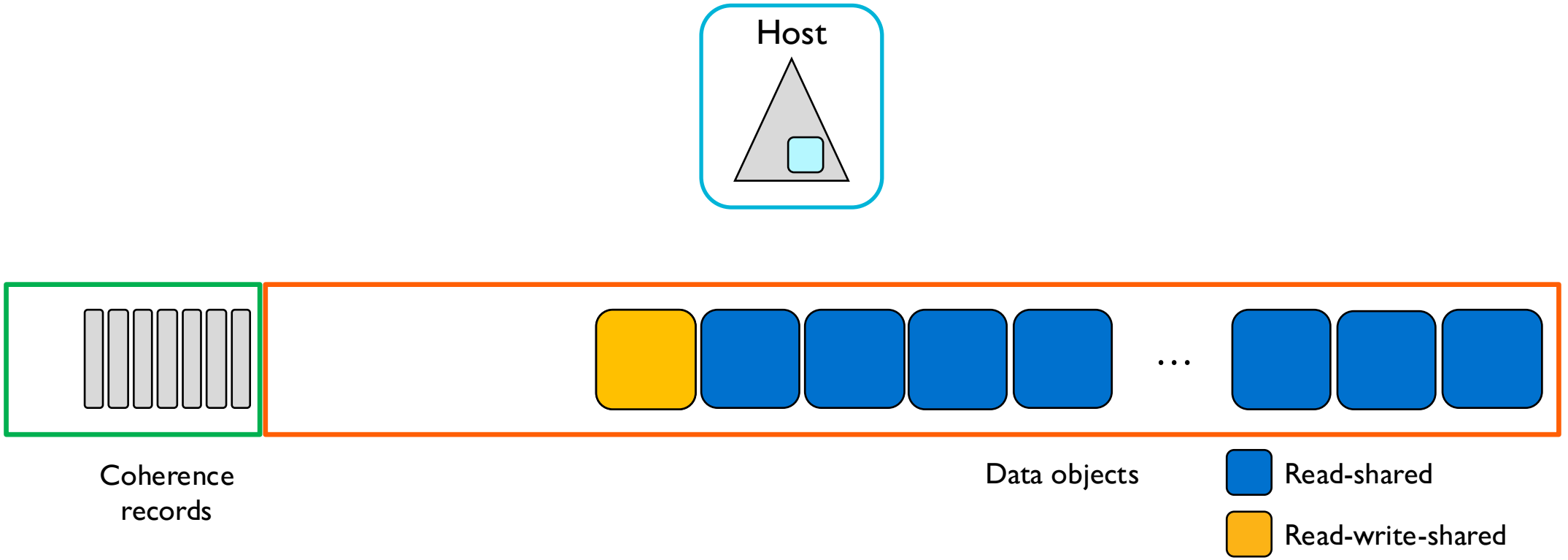
No coherence records for read-shared objects

Result: unlimited read-shared objects

Dynamically allocate coherence records to objects that are written

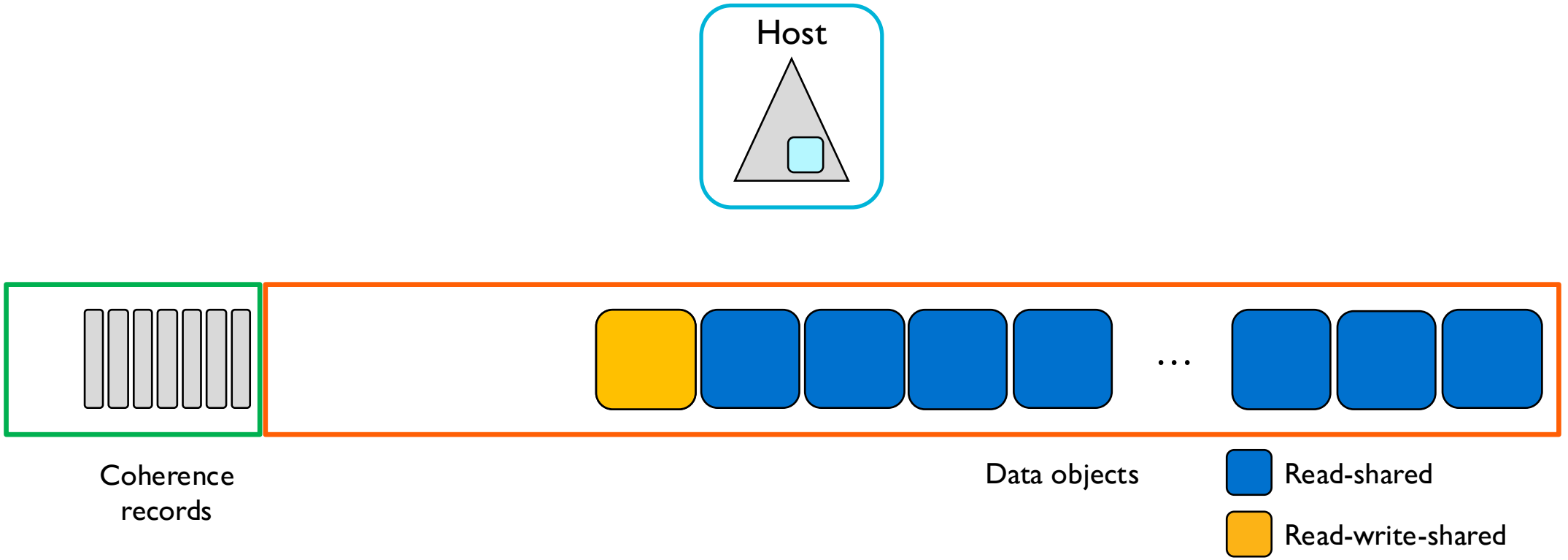
When SCR full, deallocate coherence records and change objects to read-shared

Step 2: Dual-Path Coherence



Step 2: Dual-Path Coherence

Coherence is simple when coherence records allocation does not change

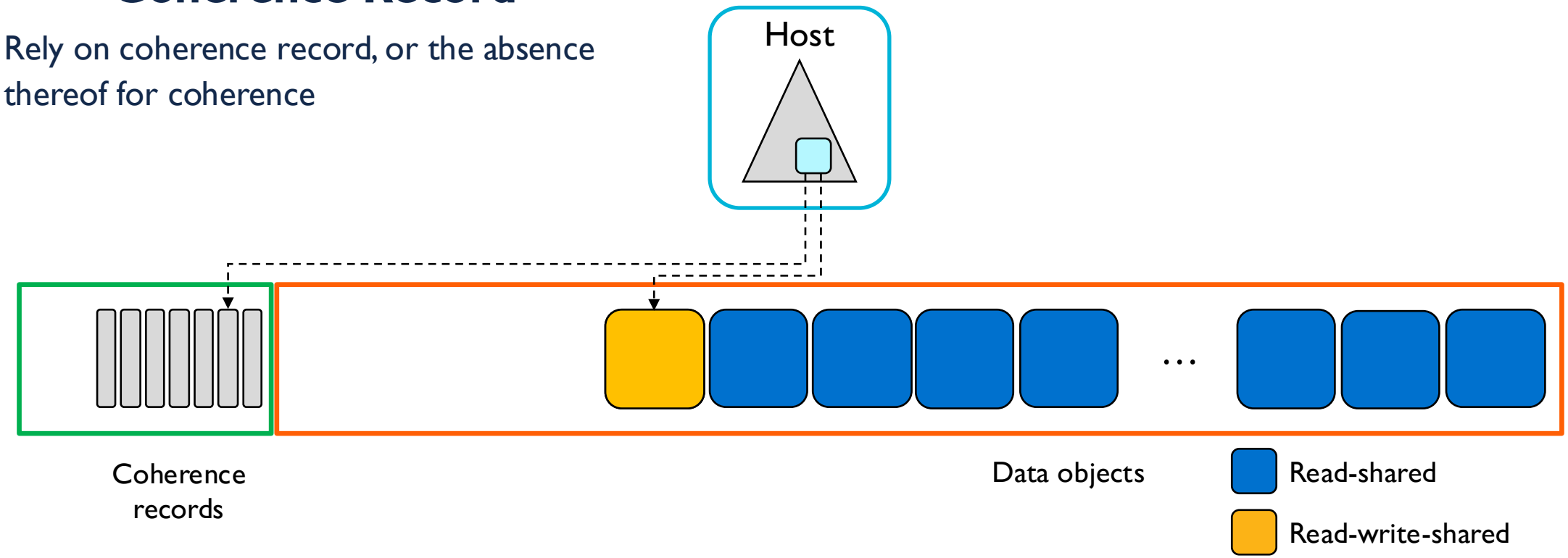


Step 2: Dual-Path Coherence

Coherence is simple when coherence records allocation does not change

Coherence Record

Rely on coherence record, or the absence thereof for coherence

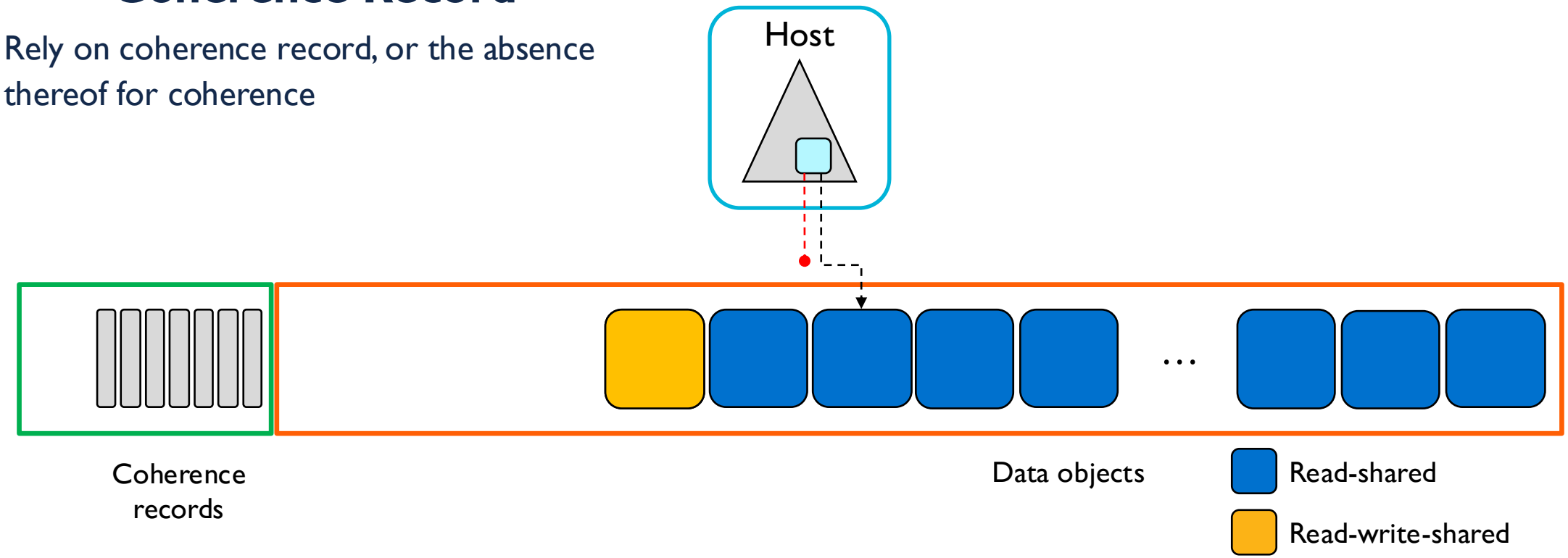


Step 2: Dual-Path Coherence

Coherence is simple when coherence records allocation does not change

Coherence Record

Rely on coherence record, or the absence thereof for coherence

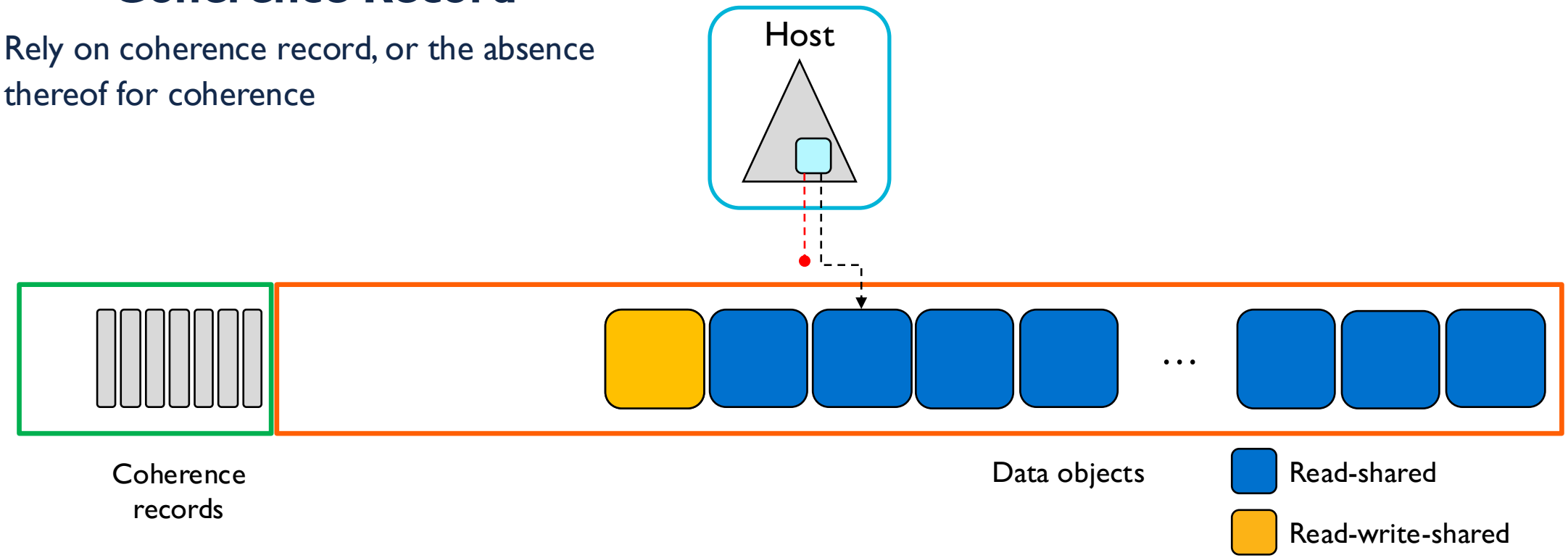


Step 2: Dual-Path Coherence

Coherence is simple when coherence records allocation does not change, but it changes dynamically

Coherence Record

Rely on coherence record, or the absence thereof for coherence



Step 2: Dual-Path Coherence

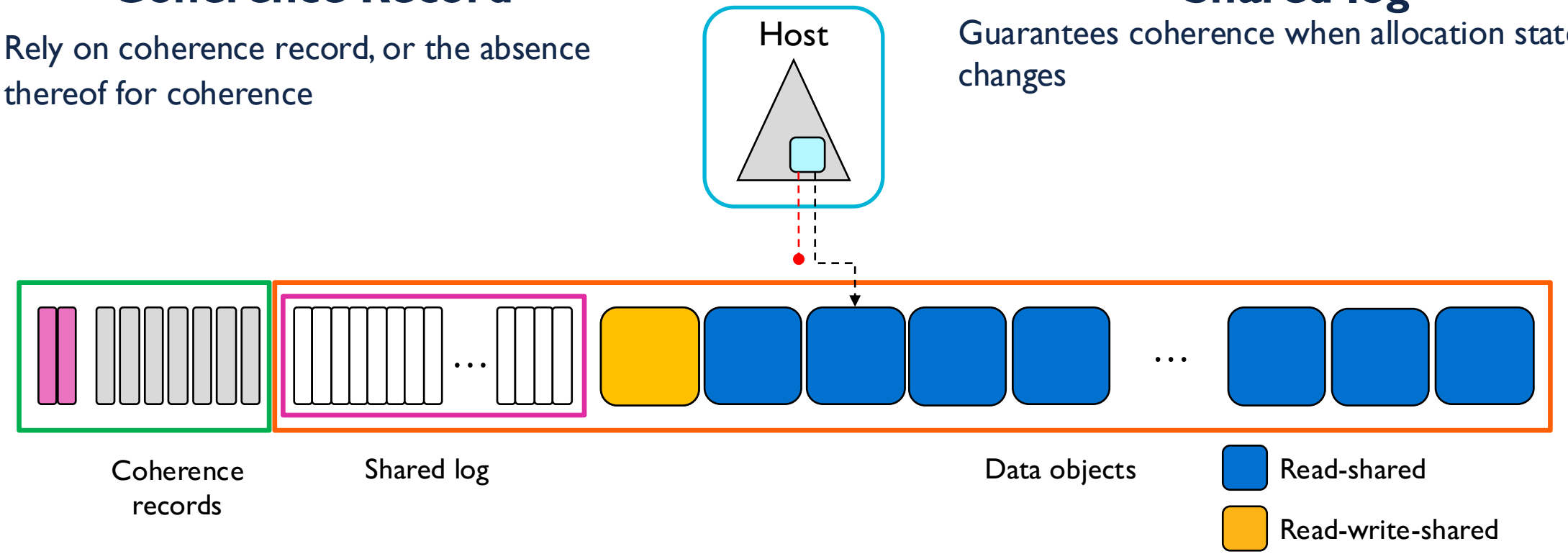
Coherence is simple when coherence records allocation does not change, but it changes dynamically

Coherence Record

Rely on coherence record, or the absence thereof for coherence

Shared log

Guarantees coherence when allocation state changes



Step 2: Dual-Path Coherence

Coherence is simple when coherence records allocation does not change, but it changes dynamically

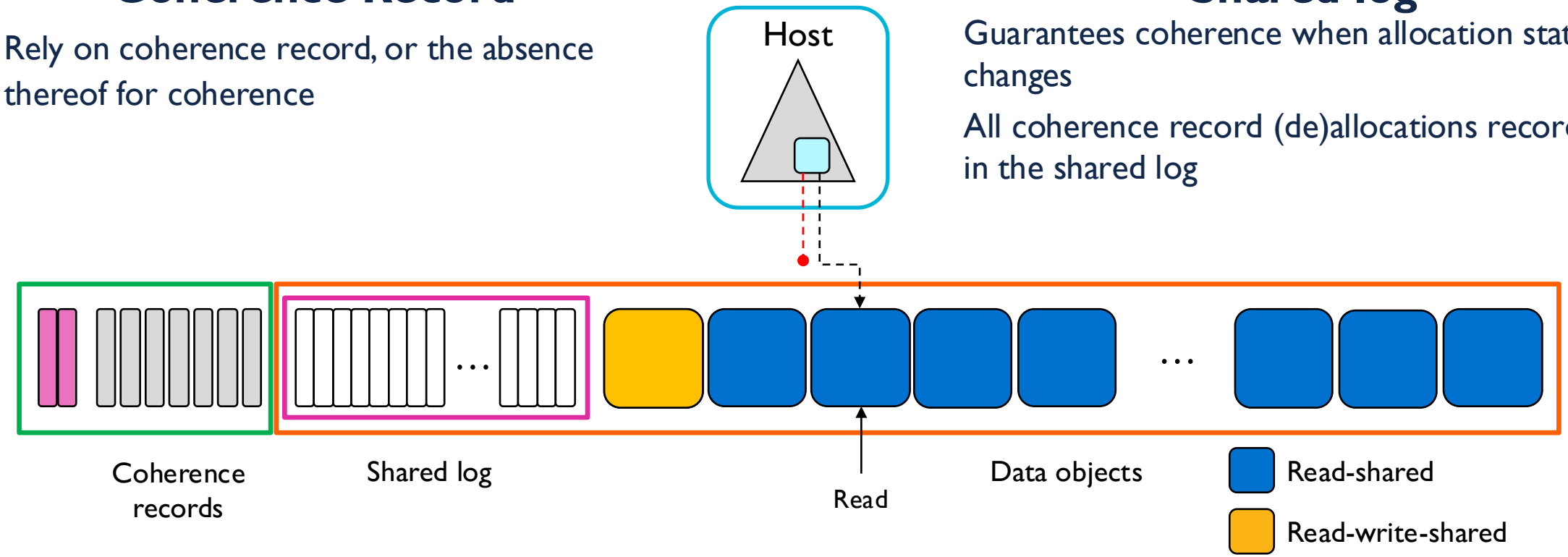
Coherence Record

Rely on coherence record, or the absence thereof for coherence

Shared log

Guarantees coherence when allocation state changes

All coherence record (de)allocations recorded in the shared log



Step 2: Dual-Path Coherence

Coherence is simple when coherence records allocation does not change, but it changes dynamically

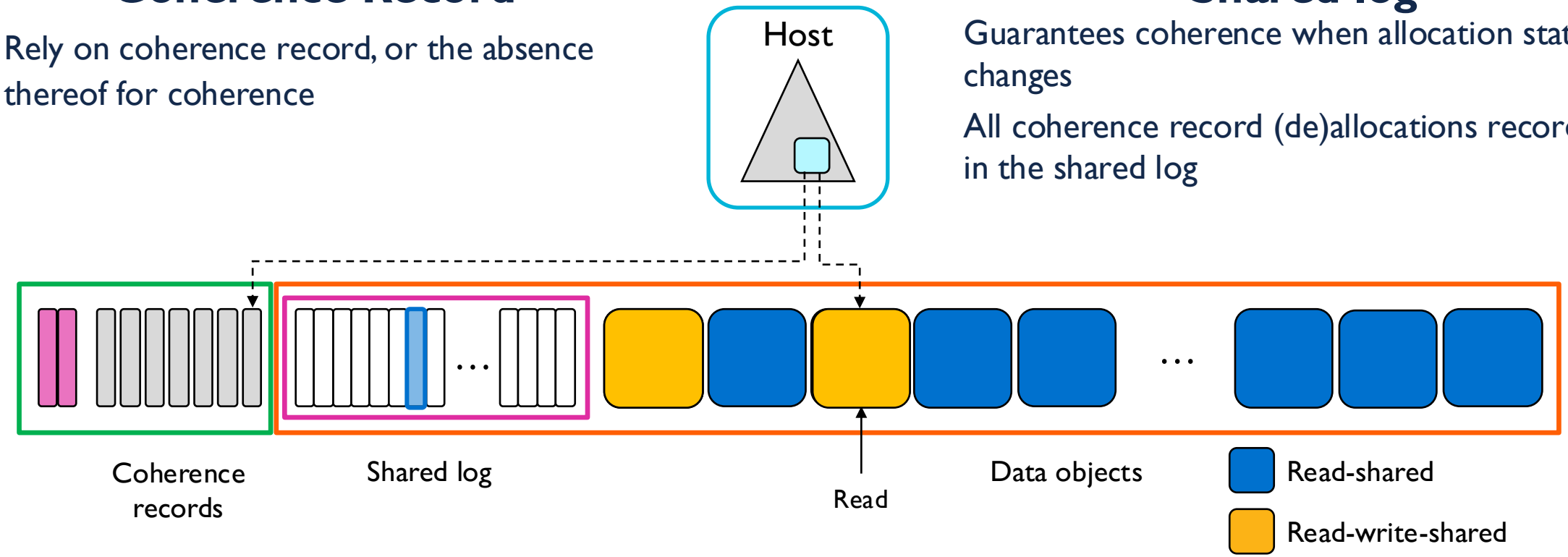
Coherence Record

Rely on coherence record, or the absence thereof for coherence

Shared log

Guarantees coherence when allocation state changes

All coherence record (de)allocations recorded in the shared log



Step 2: Dual-Path Coherence

Coherence is simple when coherence records allocation does not change, but it changes dynamically

Coherence Record

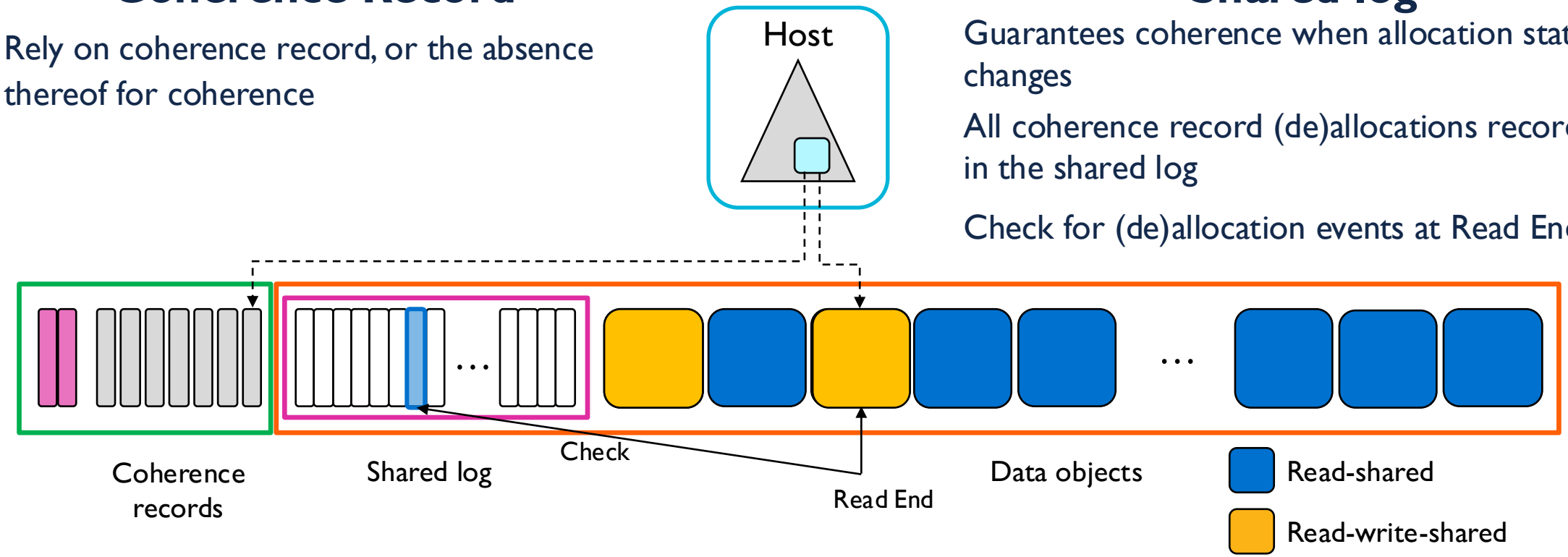
Rely on coherence record, or the absence thereof for coherence

Shared log

Guarantees coherence when allocation state changes

All coherence record (de)allocations recorded in the shared log

Check for (de)allocation events at Read End



Step 2: Dual-Path Coherence

Coherence is simple when coherence records allocation does not change, but it changes dynamically

1st Path: Coherence Record

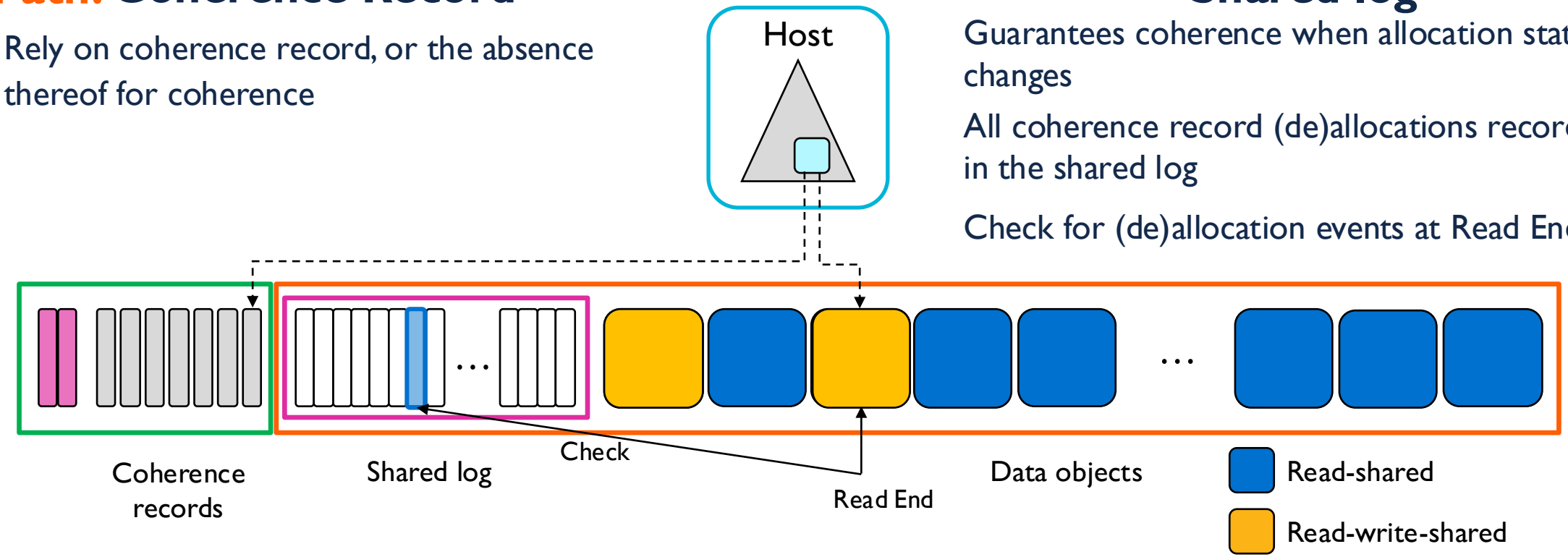
Rely on coherence record, or the absence thereof for coherence

Shared log

Guarantees coherence when allocation state changes

All coherence record (de)allocations recorded in the shared log

Check for (de)allocation events at Read End



Step 2: Dual-Path Coherence

Coherence is simple when coherence records allocation does not change, but it changes dynamically

1st Path: Coherence Record

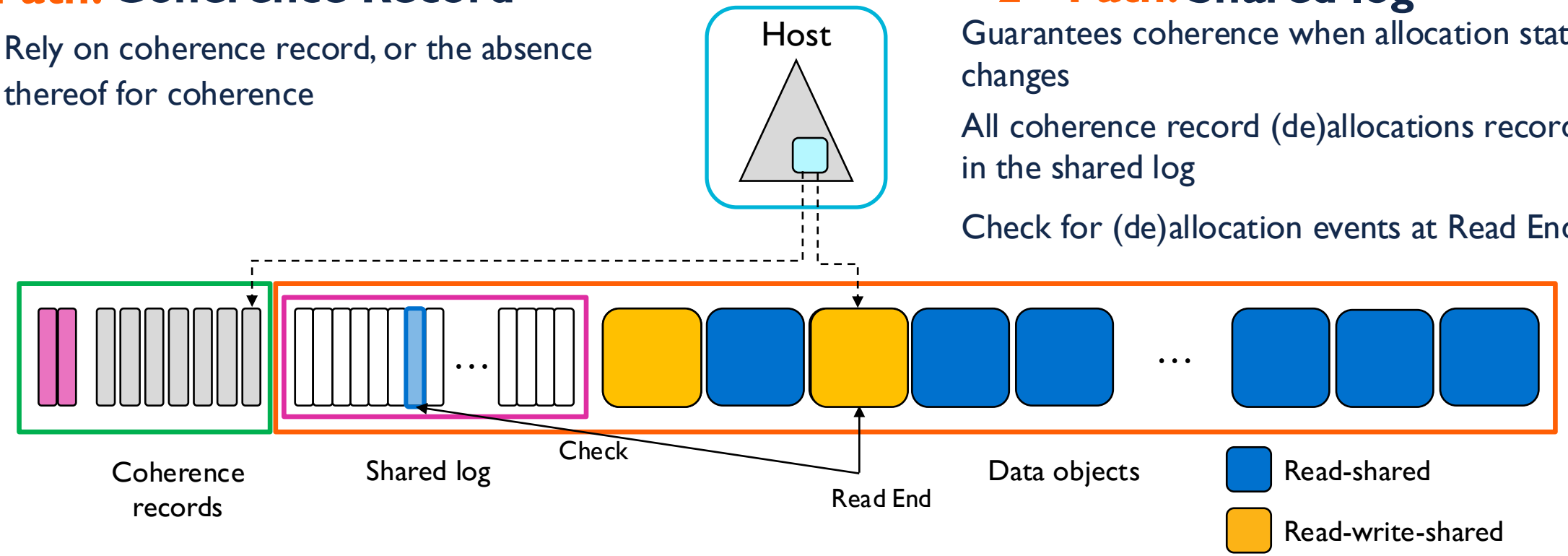
Rely on coherence record, or the absence thereof for coherence

2nd Path: Shared log

Guarantees coherence when allocation state changes

All coherence record (de)allocations recorded in the shared log

Check for (de)allocation events at Read End



MEGALON's solution to CXL's high latency

MEGALON supports data replication and tiering to host-local DRAM

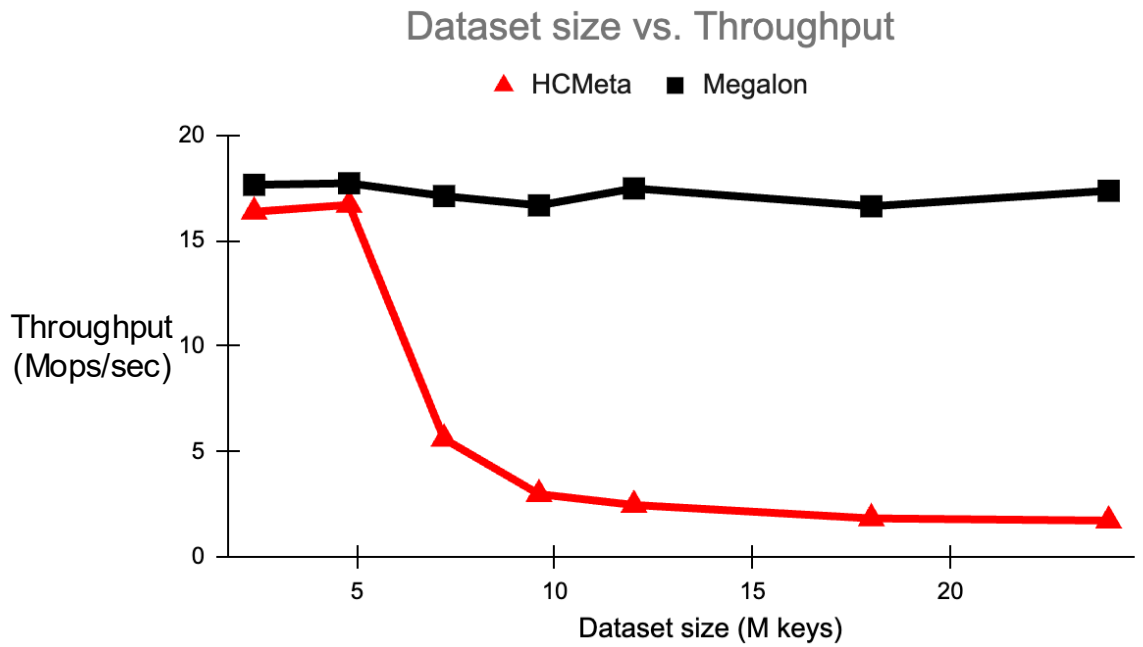
Details in the paper

Performance Evaluation

- How does MEGALON perform in read-write workload?
- How does MEGALON perform when coherence records do not fit in SCR?

How does MEGALON perform in read-write workload?

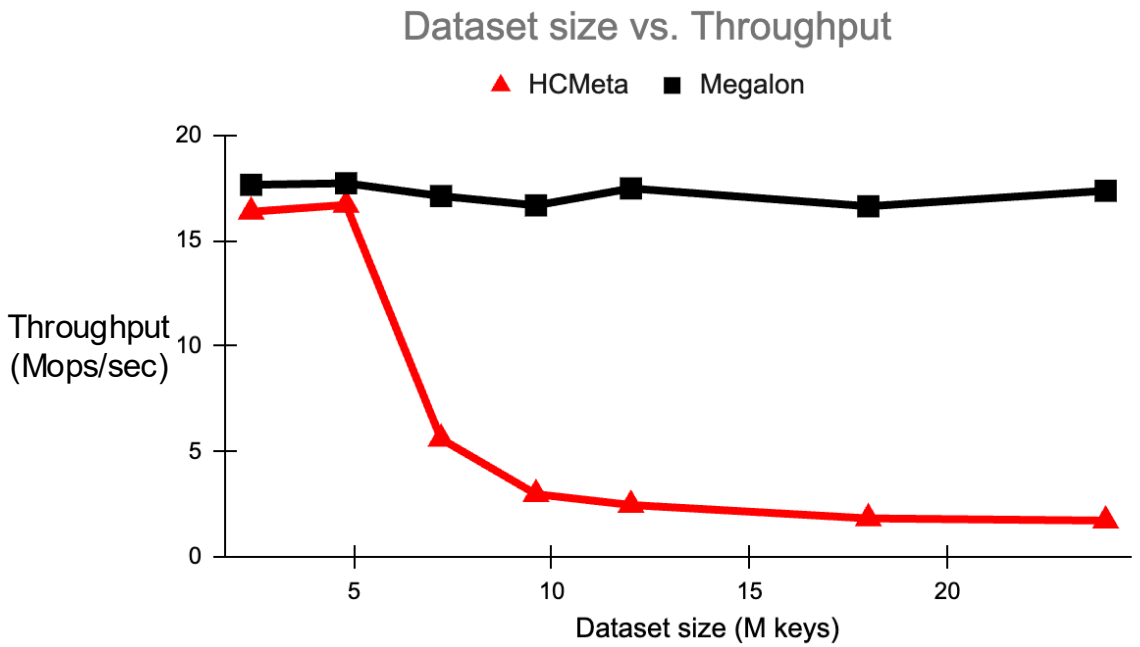
How does MEGALON perform in read-write workload?



5% write, Zipfian, 200 MB SCR

How does MEGALON perform in read-write workload?

MEGALON allocates coherence records for read-write-shared objects

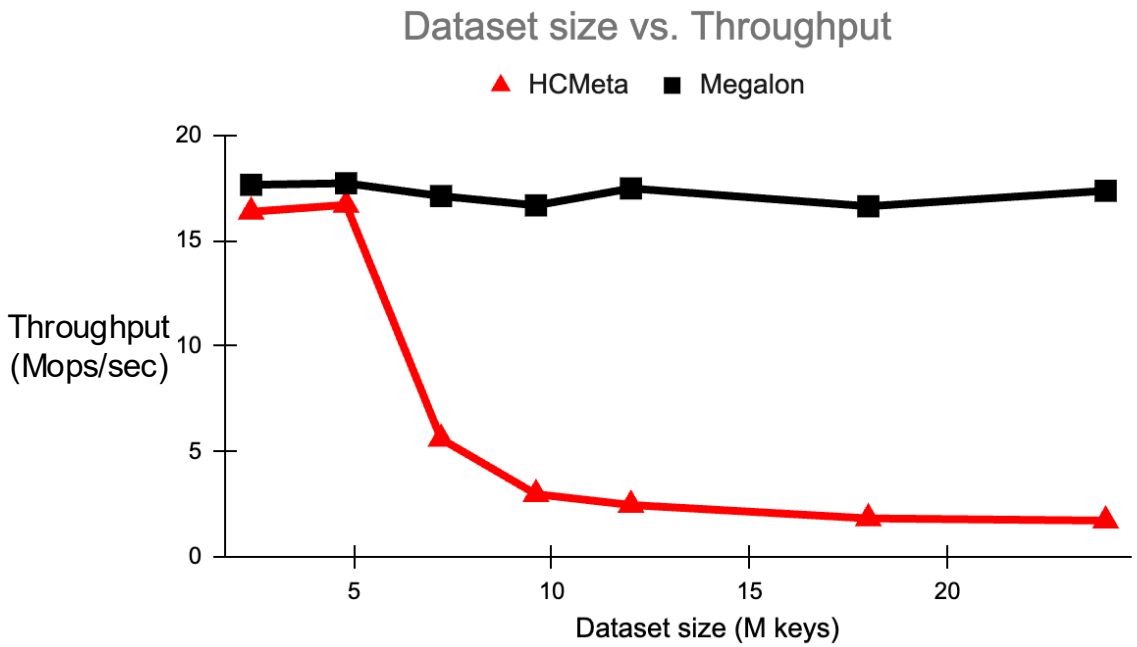


5% write, Zipfian, 200 MB SCR

How does MEGALON perform in read-write workload?

MEGALON allocates coherence records for read-write-shared objects

24M coherence records does not overflow SCR for MEGALON



5% write, Zipfian, 200 MB SCR

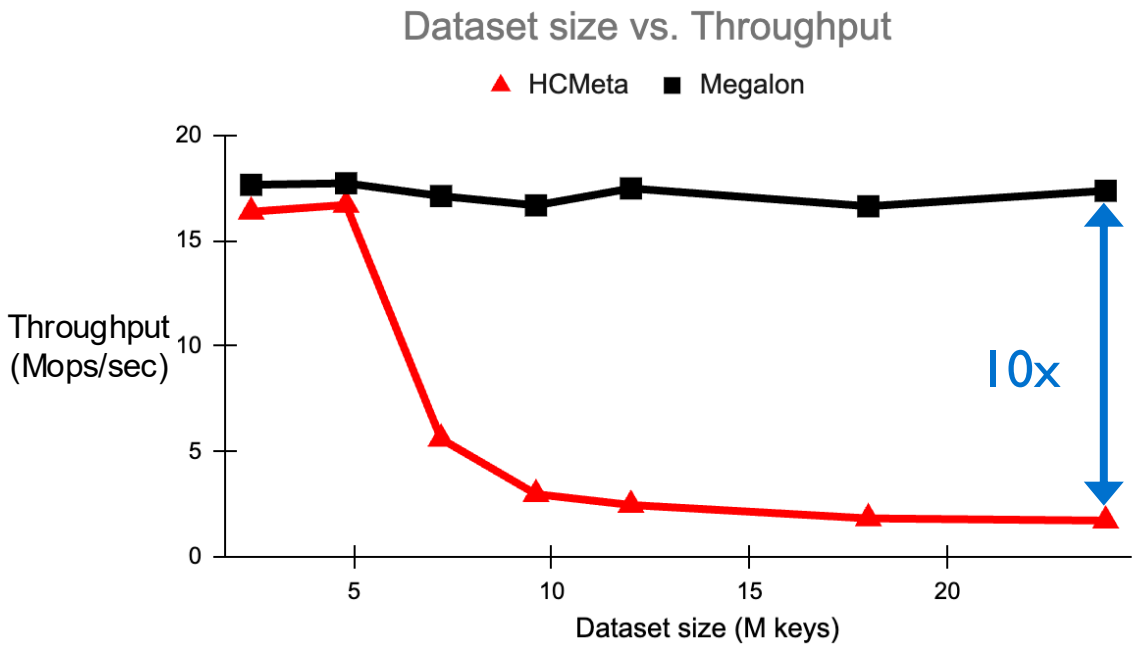
How does MEGALON perform in read-write workload?

MEGALON allocates coherence records for read-write-shared objects

24M coherence records does not overflow SCR for MEGALON

MEGALON increases system throughput

- By 10x compared to HCMeta when metadata does not fit in SCR



5% write, Zipfian, 200 MB SCR

How does MEGALON perform in read-write workload?

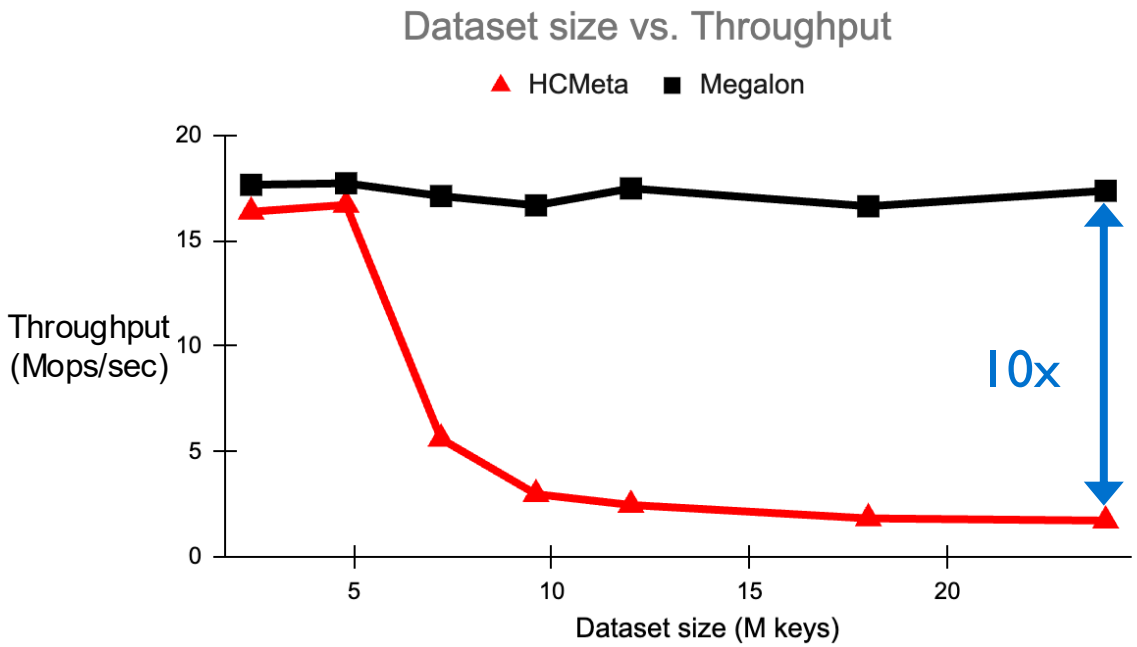
MEGALON allocates coherence records for read-write-shared objects

24M coherence records does not overflow SCR for MEGALON

MEGALON increases system throughput

- By 10x compared to HCMeta when metadata does not fit in SCR

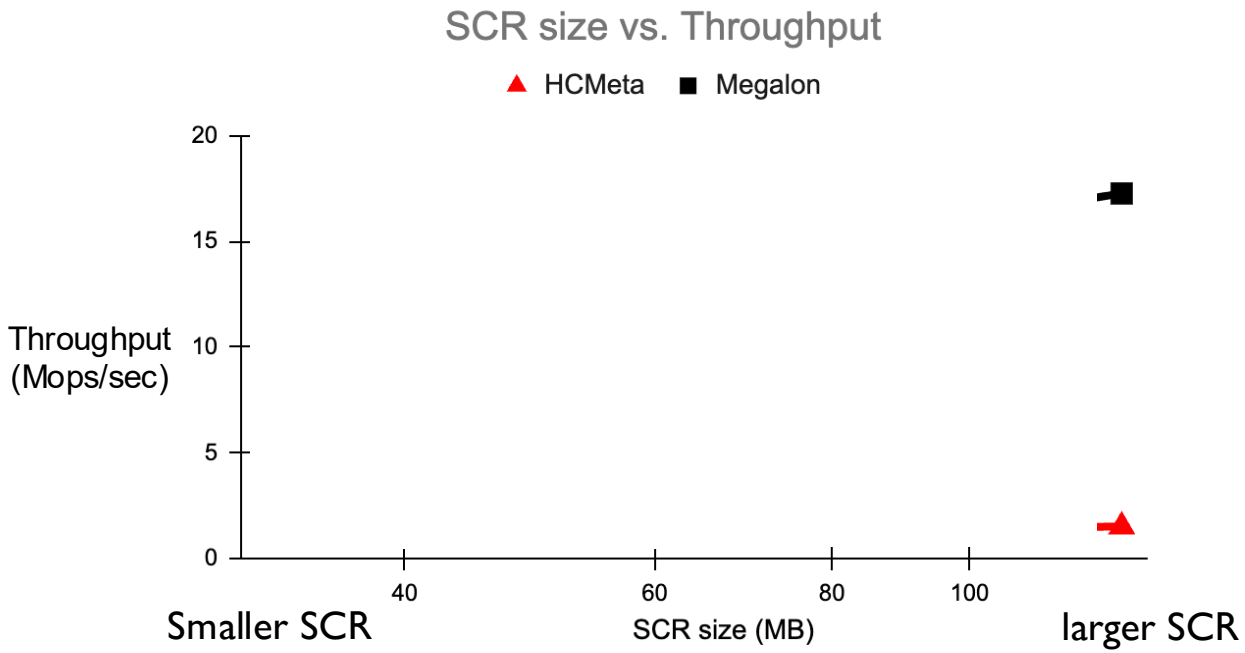
Takeaway: MEGALON significantly increases the number of coherence records that can be shared with Split Sharing compared to HCMeta



5% write, Zipfian, 200 MB SCR

How does MEGALON perform when coherence records do not fit in SCR?

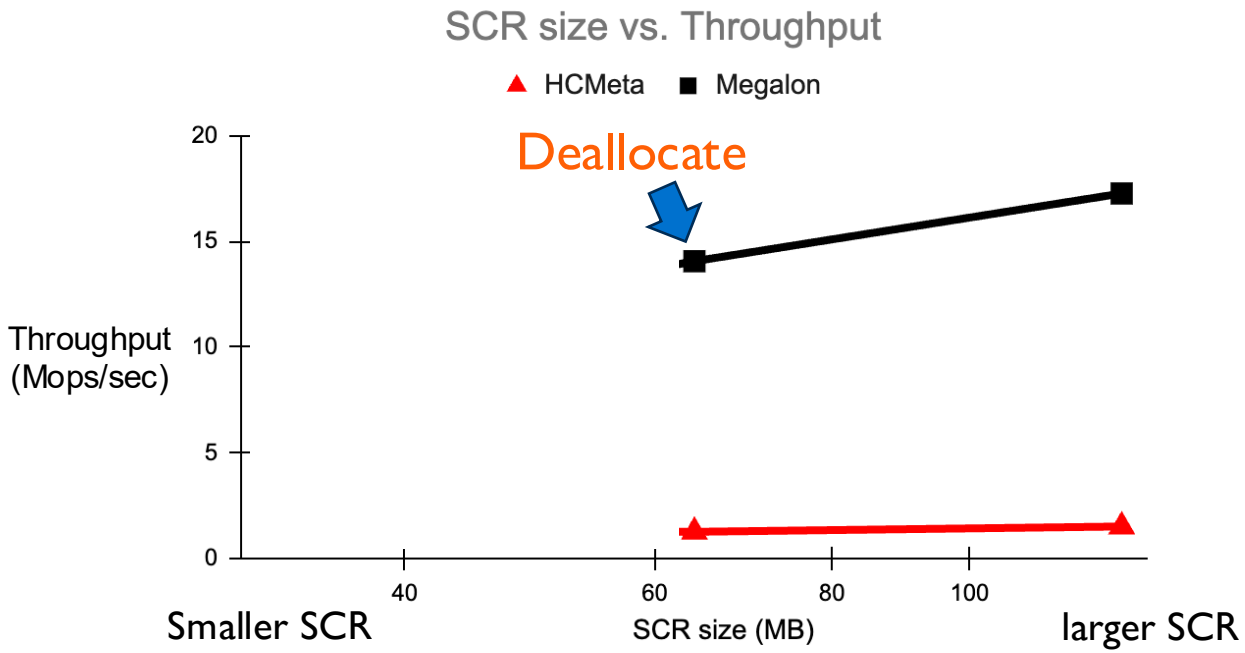
How does MEGALON perform when coherence records do not fit in SCR?



5% write, Zipfian, 18M objects

How does MEGALON perform when coherence records do not fit in SCR?

At 64 MB, MEGALON starts dynamically deallocate coherence records



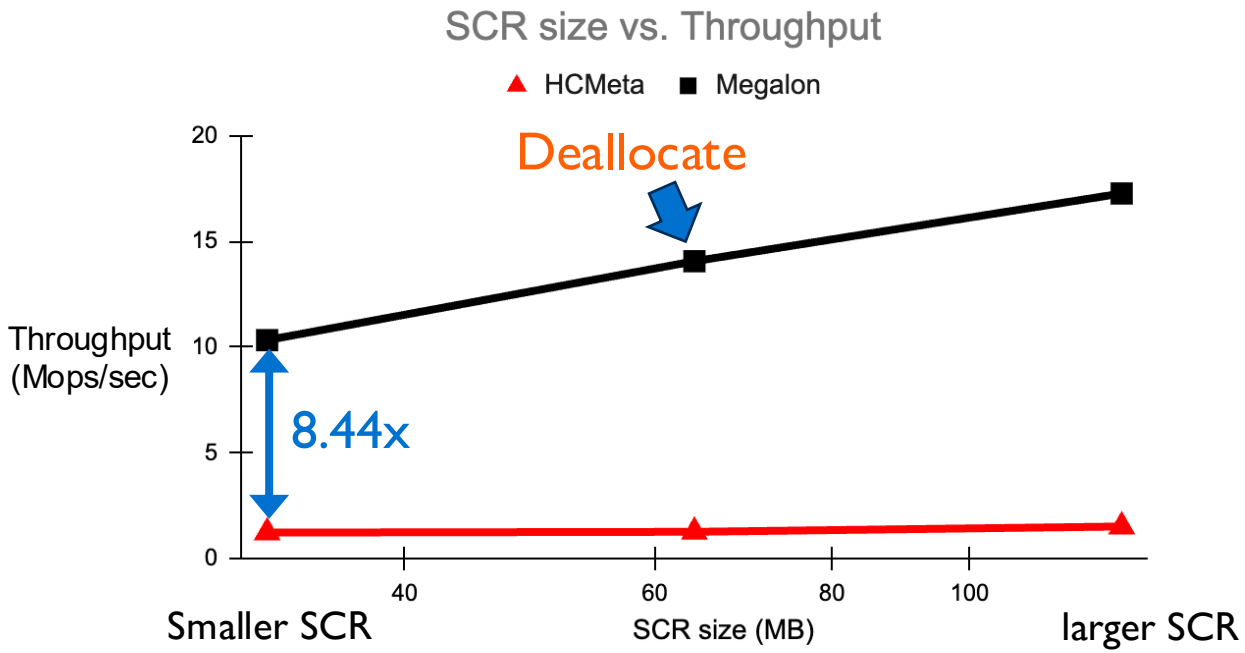
5% write, Zipfian, 18M objects

How does MEGALON perform when coherence records do not fit in SCR?

At 64 MB, MEGALON starts dynamically deallocate coherence records

At 32 MB, MEGALON still increases system throughput

- By 8.4x compared to HCMeta



5% write, Zipfian, 18M objects

How does MEGALON perform when coherence records do not fit in SCR?

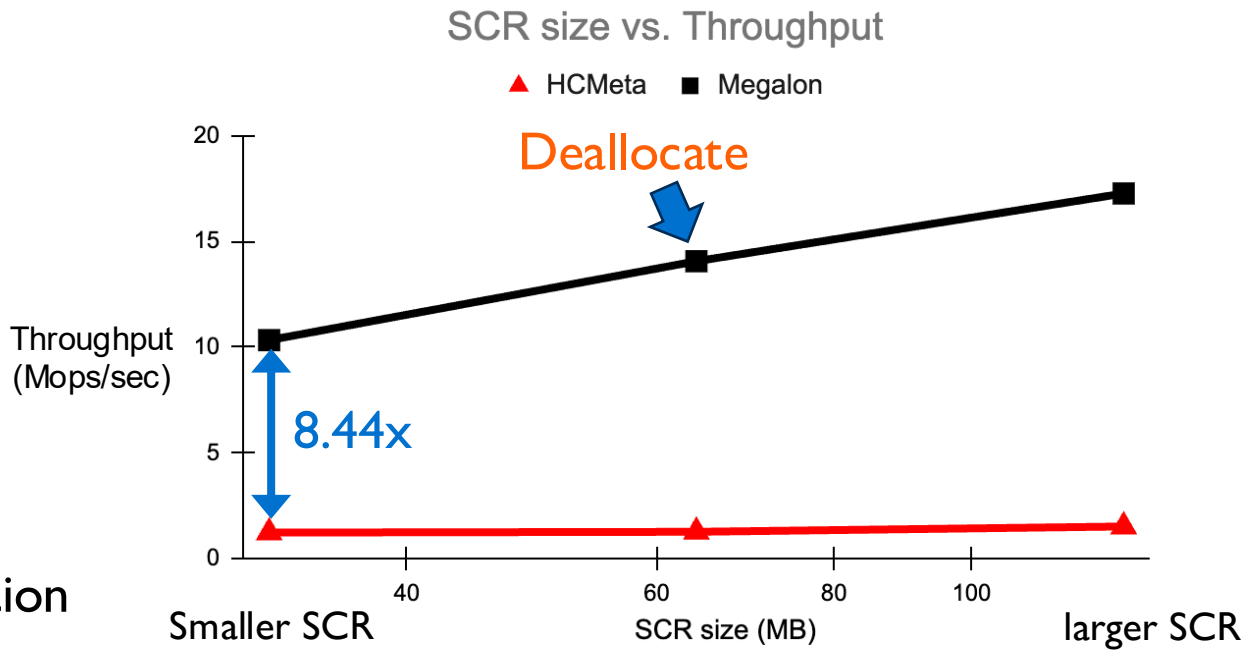
At 64 MB, MEGALON starts dynamically deallocate coherence records

At 32 MB, MEGALON still increases system throughput

- By 8.4x compared to HCMeta

Because:

- MEGALON can fit more coherence records
- Shared log has lower overhead for (de)allocation



5% write, Zipfian, 18M objects

How does MEGALON perform when coherence records do not fit in SCR?

At 64 MB, MEGALON starts dynamically deallocate coherence records

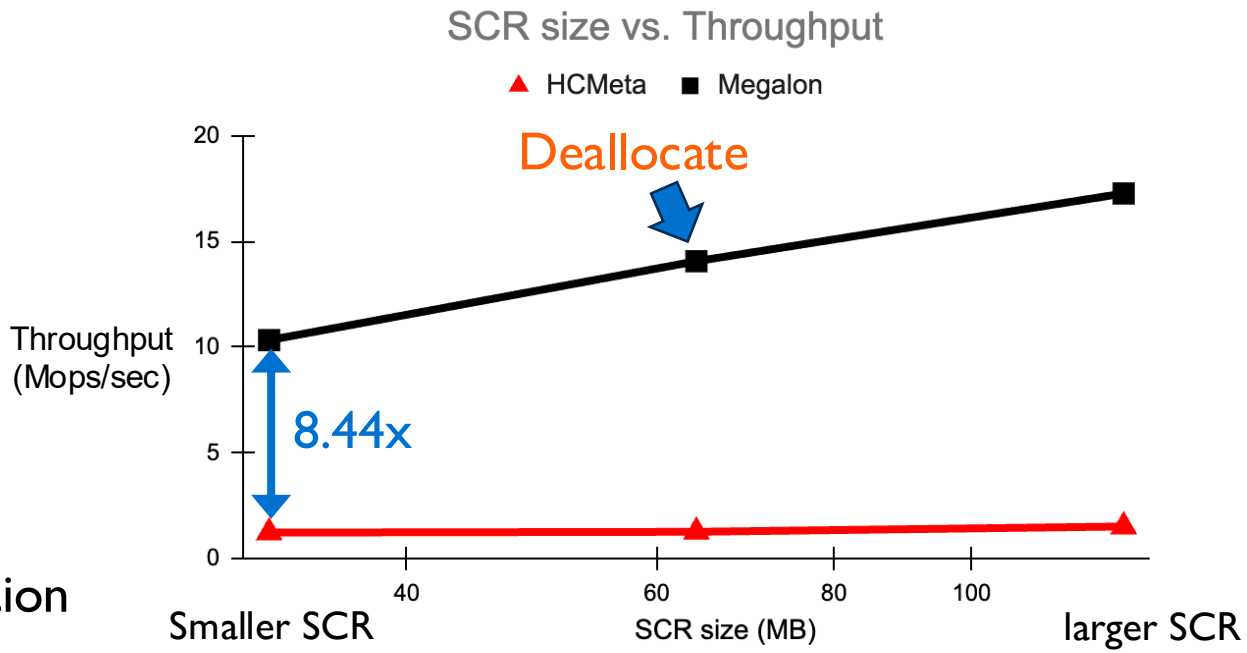
At 32 MB, MEGALON still increases system throughput

- By 8.4x compared to HCMeta

Because:

- MEGALON can fit more coherence records
- Shared log has lower overhead for (de)allocation

Takeaway: even when coherence records do not fit in SCR, MEGALON still show performance benefit over HCMeta



5% write, Zipfian, 18M objects

Summary

Summary

- Small coherence region of CXL 3.0 significantly limits its memory sharing capability

Summary

- Small coherence region of CXL 3.0 significantly limits its memory sharing capability
- MEGALON – a new data sharing approach for partly coherent CXL

Summary

- Small coherence region of CXL 3.0 significantly limits its memory sharing capability
- MEGALON – a new data sharing approach for partly coherent CXL
- Supports sharing a large number of objects through split metadata sharing approach

Summary

- Small coherence region of CXL 3.0 significantly limits its memory sharing capability
- MEGALON – a new data sharing approach for partly coherent CXL
- Supports sharing a large number of objects through split metadata sharing approach
- A CXL shared log enables index consistency and dual-path coherence

Summary

- Small coherence region of CXL 3.0 significantly limits its memory sharing capability
- MEGALON – a new data sharing approach for partly coherent CXL
- Supports sharing a large number of objects through split metadata sharing approach
- A CXL shared log enables index consistency and dual-path coherence
- MEGALON delivers significant performance benefit over the current approach

Summary

- Small coherence region of CXL 3.0 significantly limits its memory sharing capability
- MEGALON – a new data sharing approach for partly coherent CXL
- Supports sharing a large number of objects through split metadata sharing approach
- A CXL shared log enables index consistency and dual-path coherence
- MEGALON delivers significant performance benefit over the current approach

Artifact








Backup

The number of objects that can be shared before churn

The number of objects that can be shared before churn

24 Byte key, 200 MB SCR:

- HCMeta: 36 bytes metadata SCR footprint per object; ~5.5 M objects
- Megalon: 4 bytes (32-bit coherence record) SCR footprint per object; ~50M read-write-shared objects & unlimited-read shared objects

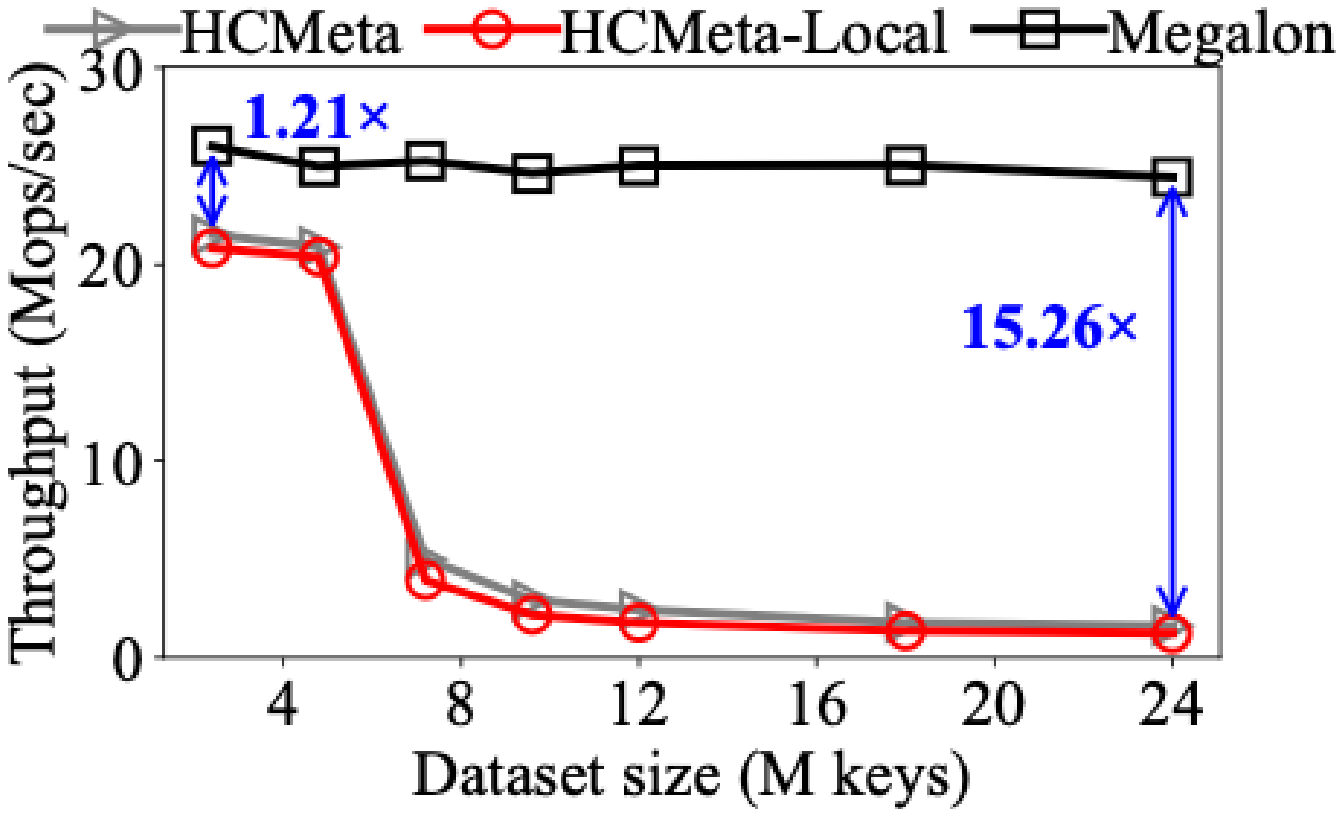
Memory overhead

Memory overhead

24 Byte key, 1KB data object, 24M object, 200 MB SCR:

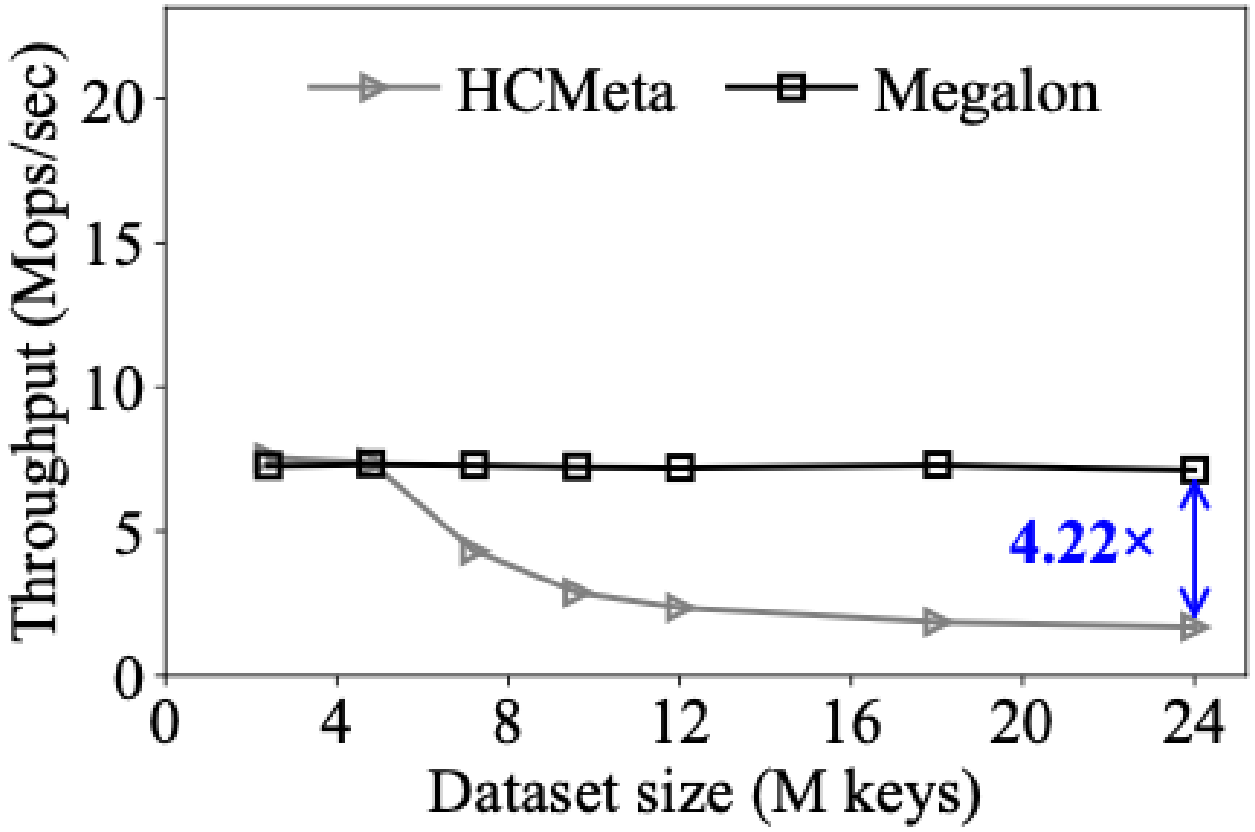
- HCMeta: 25.76 GB total memory footprint, 1.6 MOps/s
- Megalon: 27.71GB total memory footprint, 24.4 Mops/

How does MEGALON perform in read-only workload?



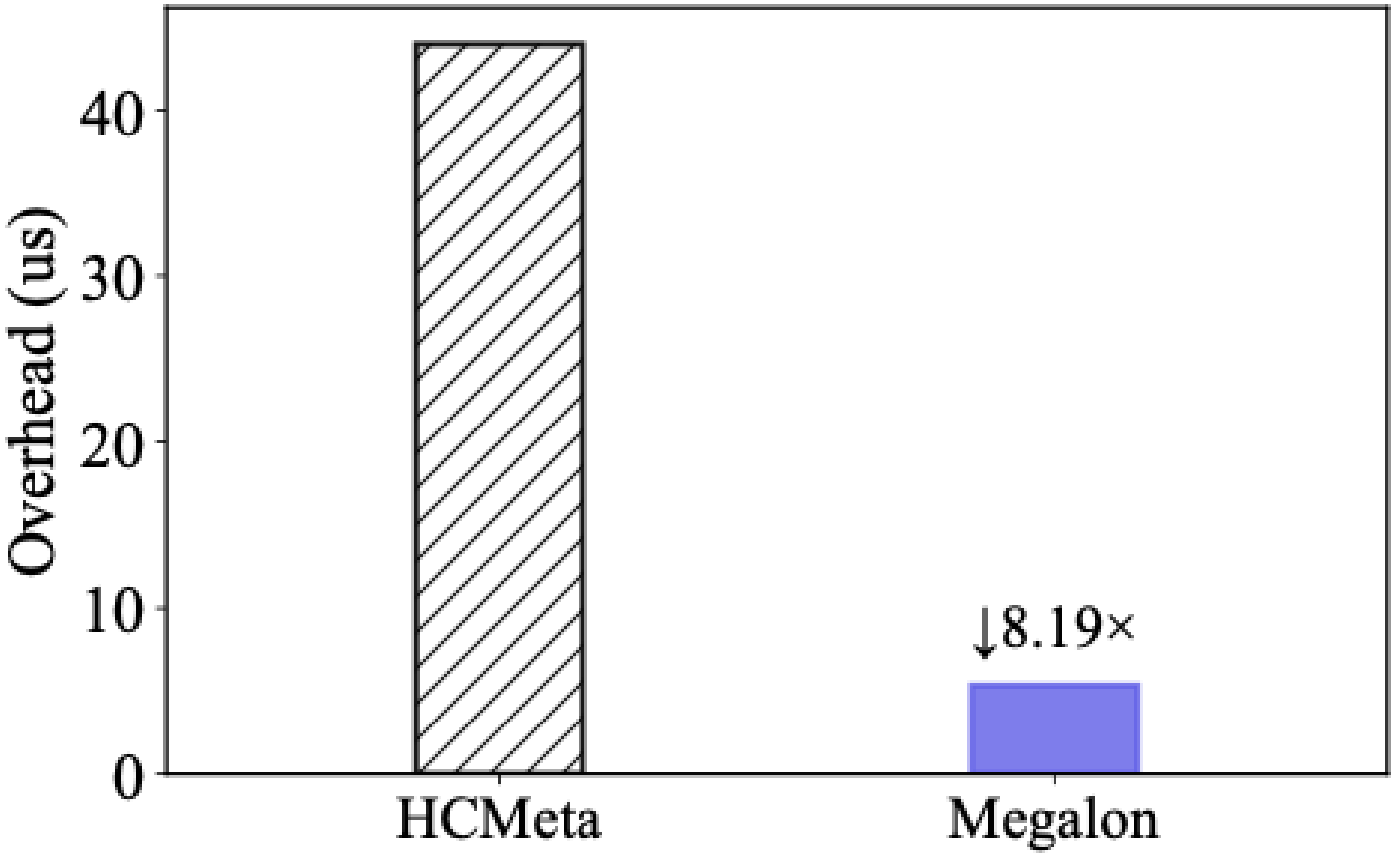
Read-only, Zipfian, 200 MB SCR

How does MEGALON perform in 50%-write workload?



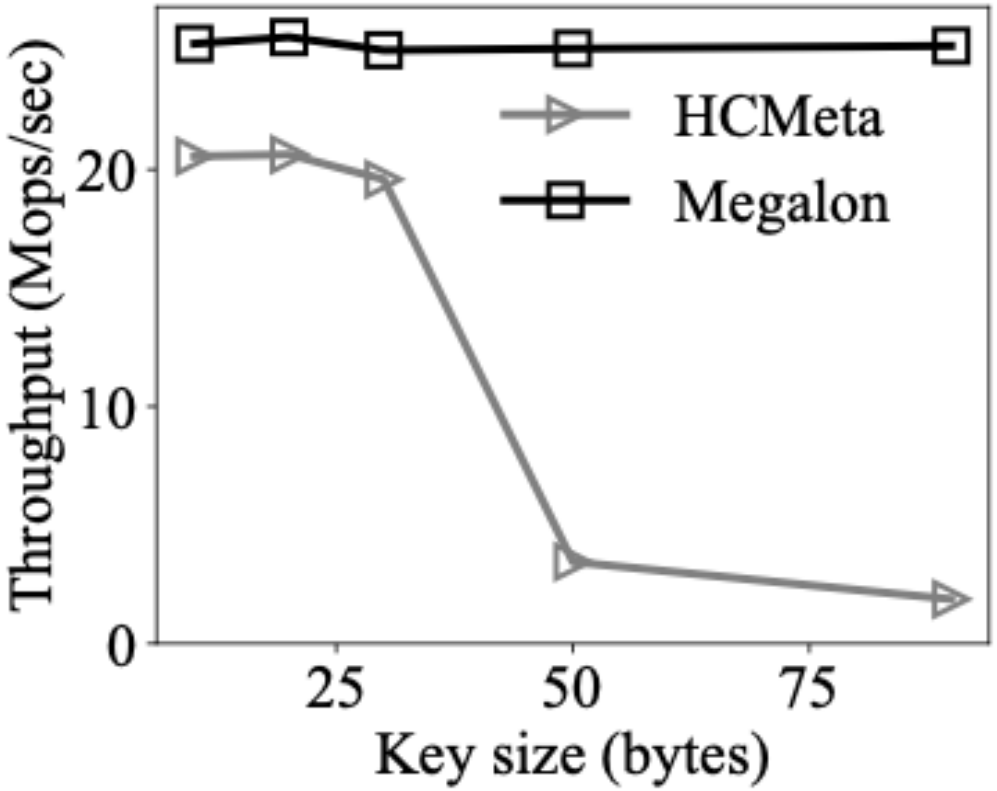
50% write, Zipfian, 200 MB SCR

Overhead of Churns



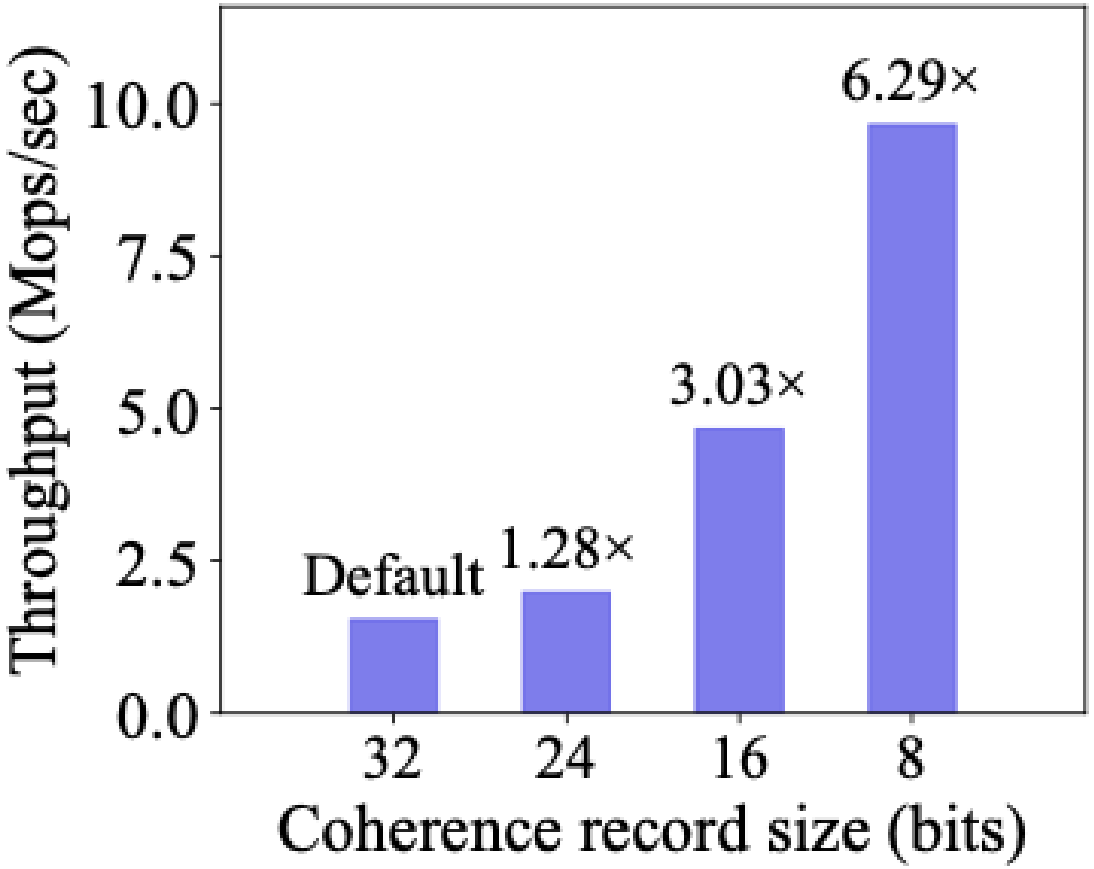
Data object (un)sharing vs. coherence records (de)allocation

Effect of Different Key Sizes



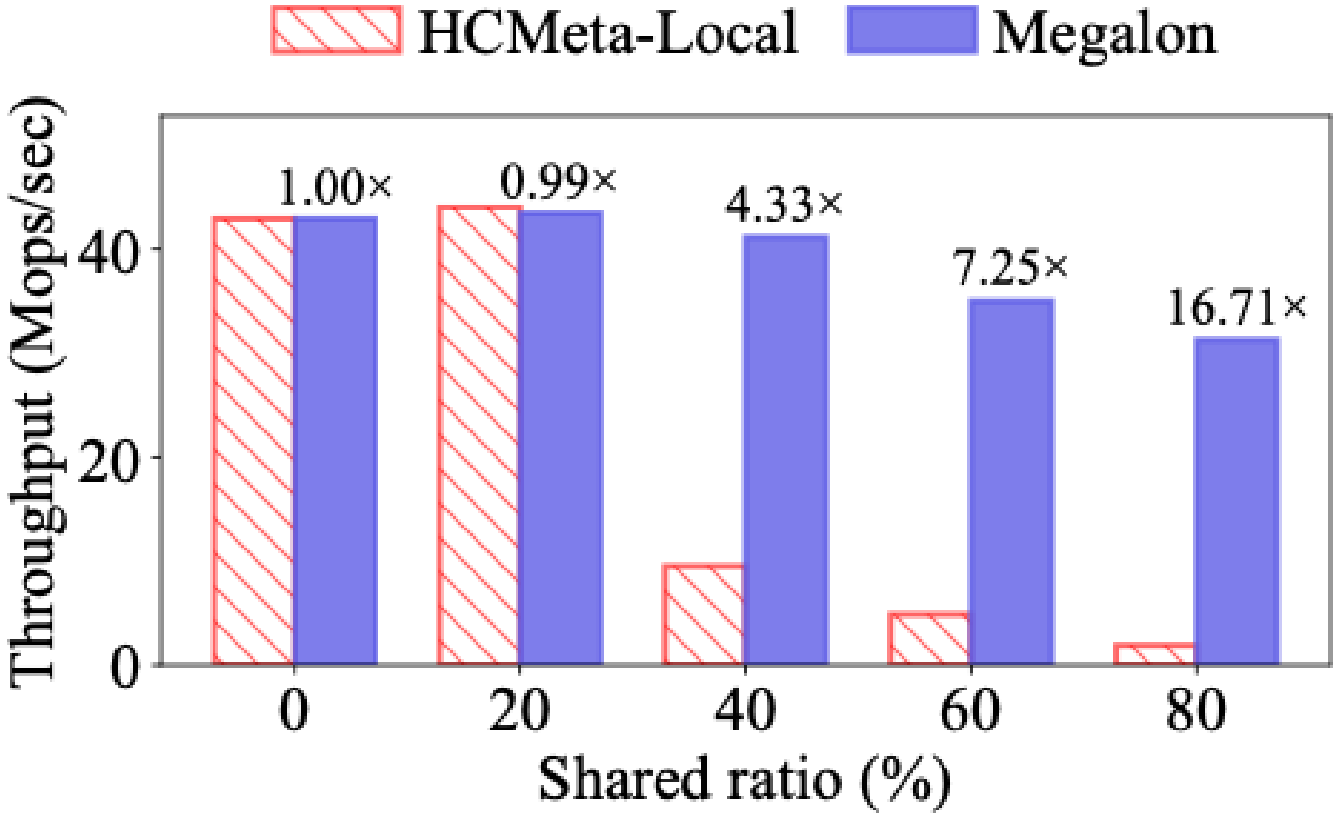
Read only , Zipfian, 200MB SCR, 4.8M objects

Effect of Different Coherence Record Sizes



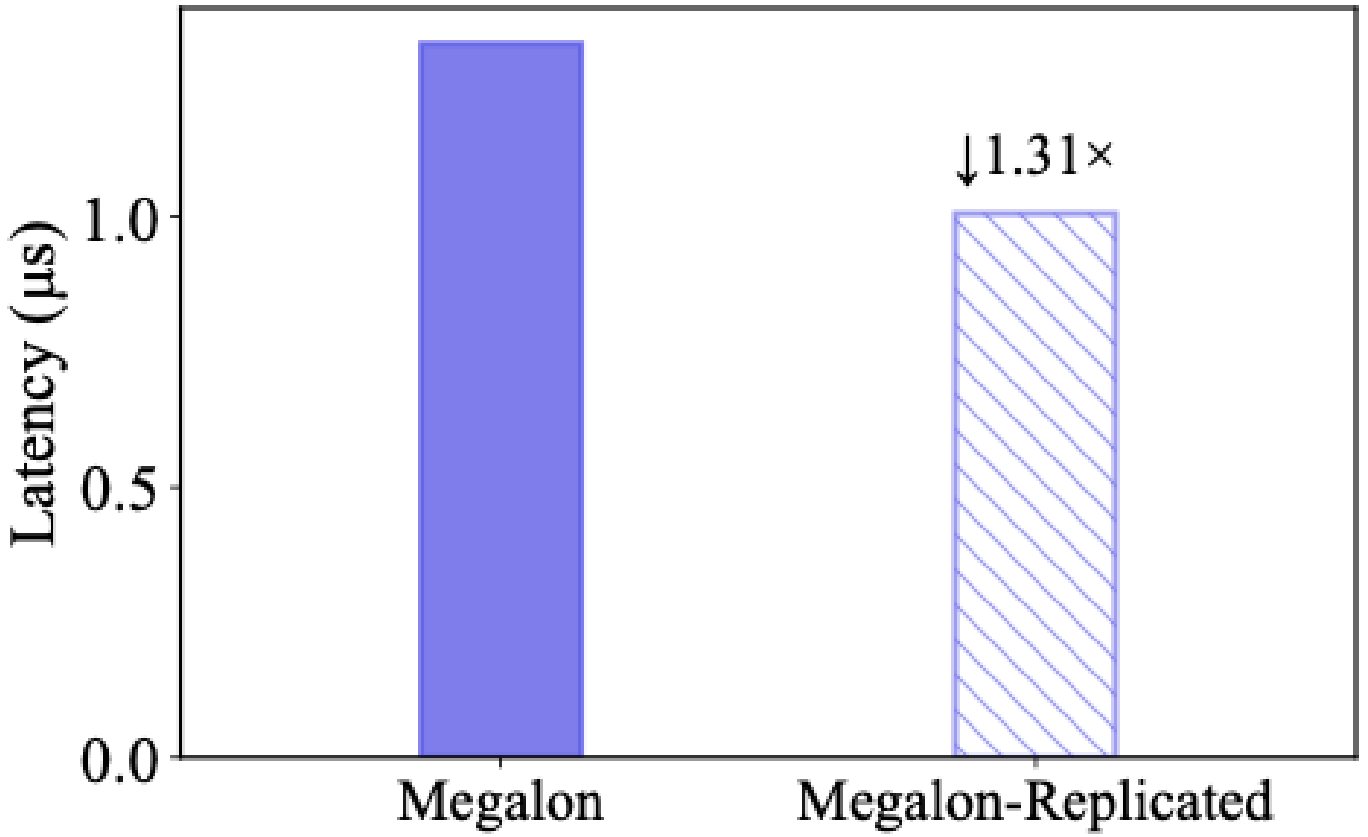
50% write, Zipfian (less skew), 32MB SCR, 18M objects

Local Partition



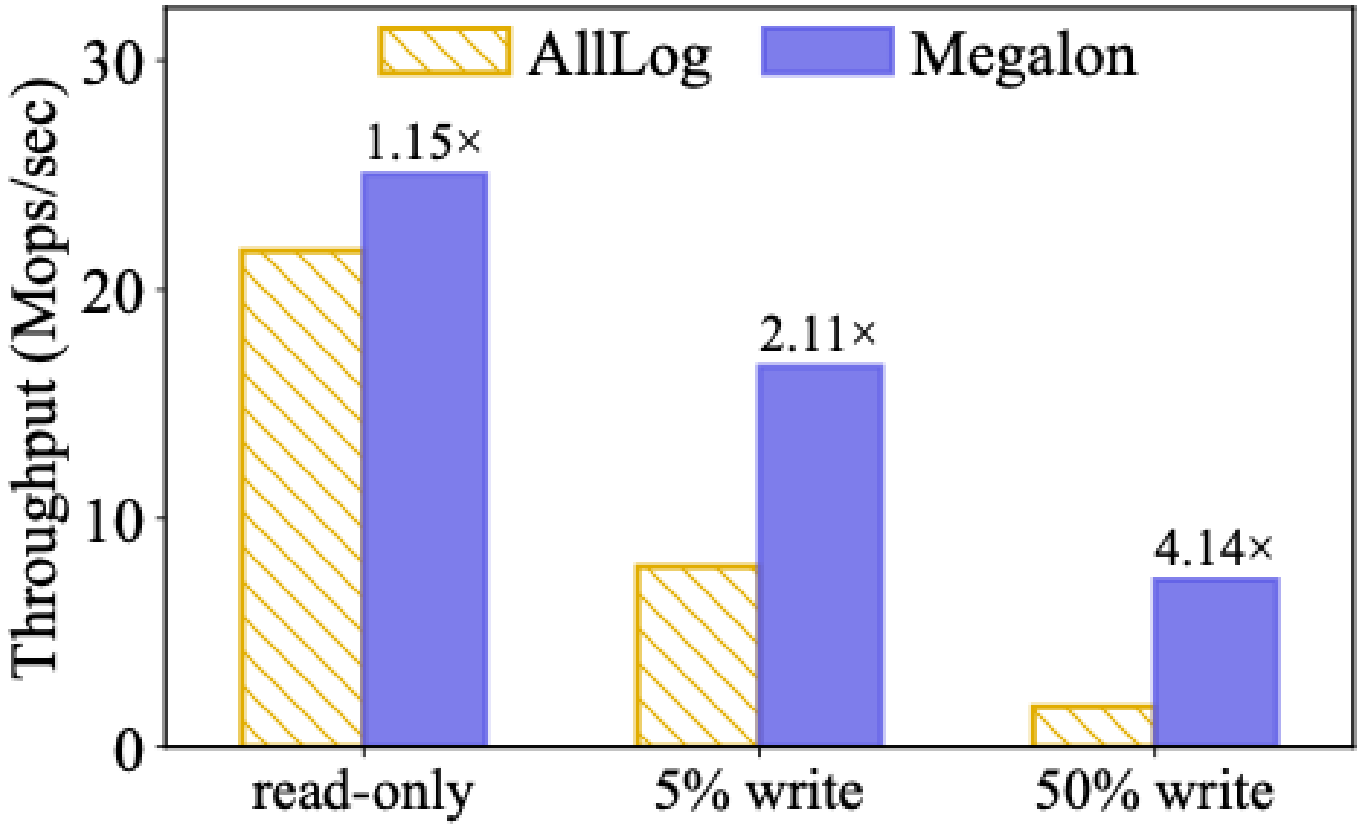
Read only, Zipfian, 200MB SCR, 18M objects

Local Replication



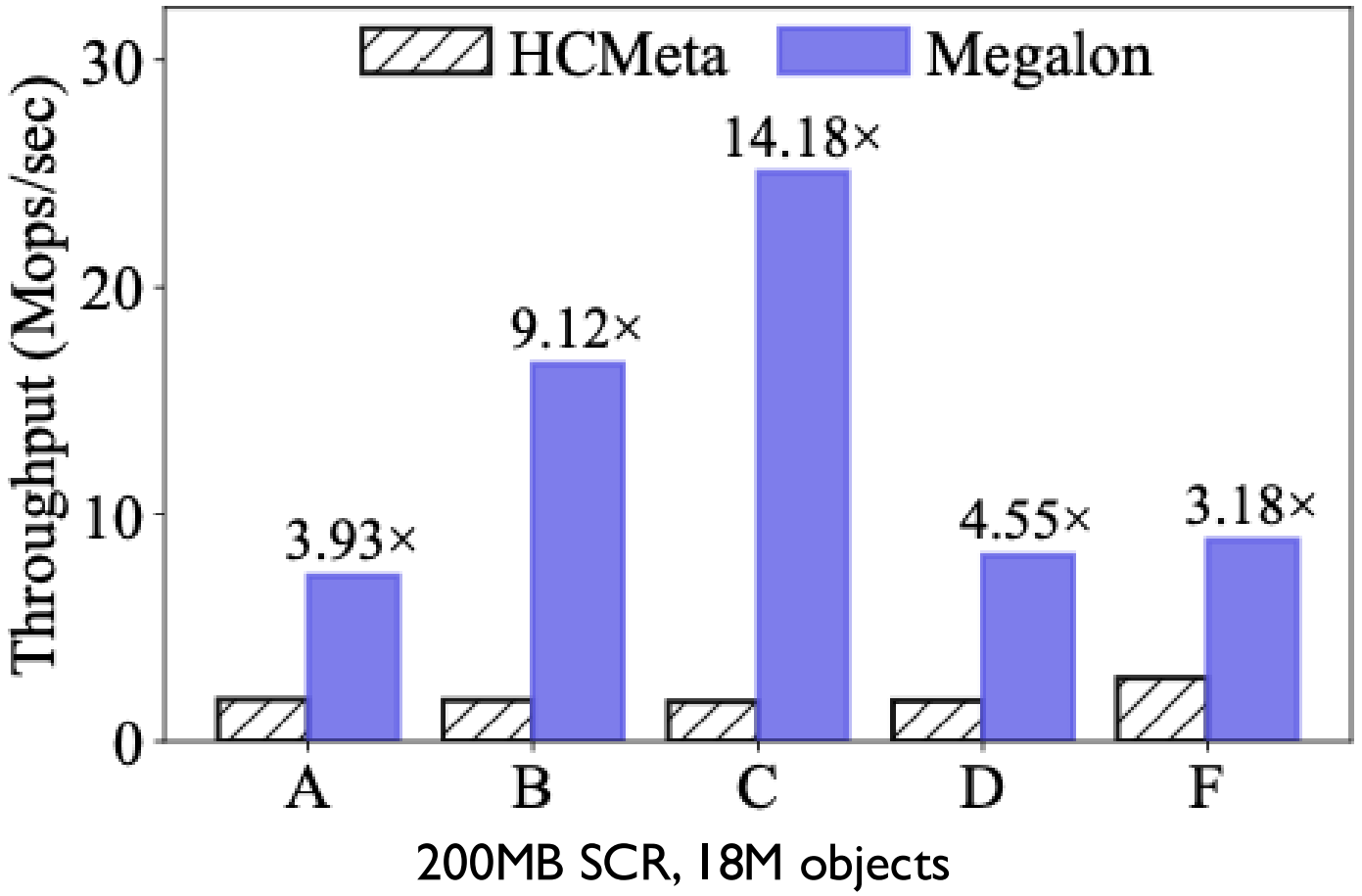
Read only, Zipfian, 200MB SCR, 18M objects

What if Coherence Record is also Logically Replicated?



Zipfian, 200MB SCR, 18M objects

YCSB Workloads



200MB SCR, 18M objects

How does MEGALON perform in page-cache application?

Workload: 48 GB dataset size, 4KB page size, Zipfian

MEGALON increases system throughput

- By 5.7x compared to HCMeta in read-only workload
- By up to 4.5x in read-write workload