Quantifying the Cost of Software-Managed Synchronization on Multi-Host CXL Memory

Peiran Du[1], Weishu Deng[1], Yang Song[1], Chenfei Liao[1]
Yujie Yang[1], Ren Wang[2], Hui Lu[1], Jia Rao[1]

UNIVERSITY OF TEXAS ARLINGTON

intel

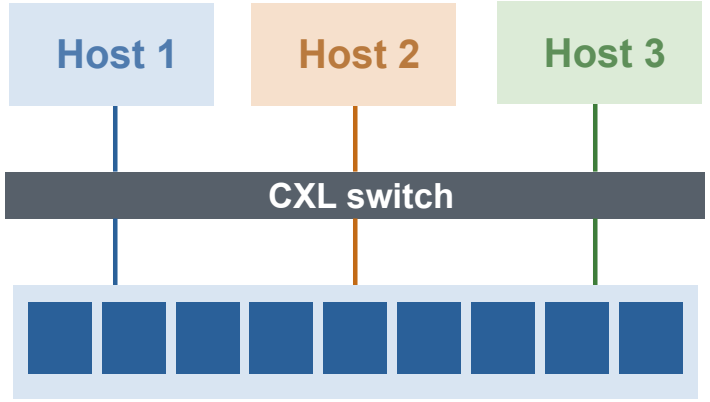


CXL-enabled Distributed Shared Memory (CXL-DSM)

- Compute Express Link (CXL) – a memory-semantic interconnect
  - Enables byte-addressable memory access over a PCIe-based fabric.
  - Allows cross-host memory access via ordinary `load/store` instructions.
- Traditional distributed shared memory (DSM) systems
  - Rely on software runtimes to construct a page-granularity shared address space.

CXL offers *hardware-supported*, *fine-grained* (cacheline-granularity) shared memory access across hosts.

Can CXL-DSM finally deliver the programmability of shared memory with the scalability of distributed systems?



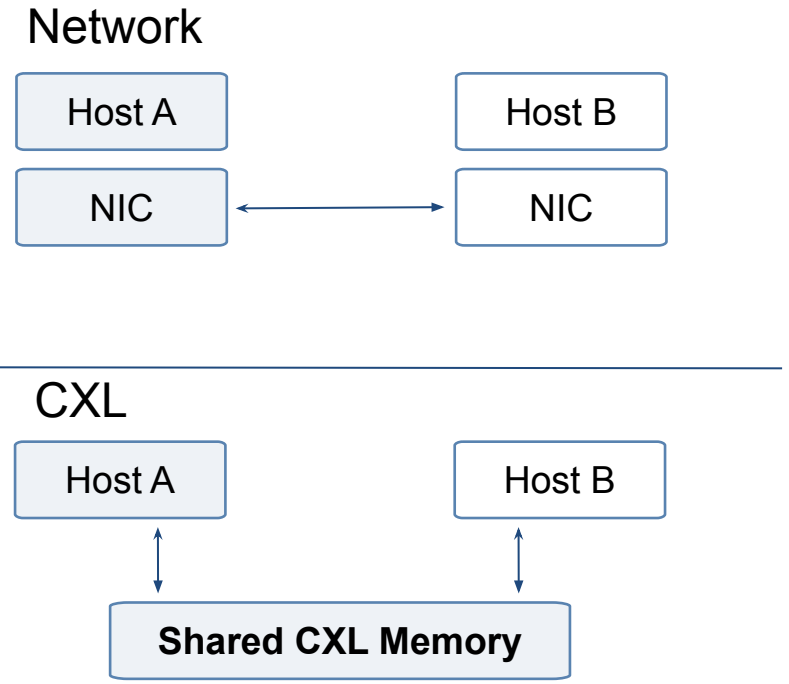
CXL-DSM

Lack of Hardware Cache Coherence in CXL-DSM

- Traditional DSM uses software-managed, page-granularity coherence
  - Tracks page ownership and access permissions through distributed directories.
  - Uses page faults, network messages, and page transfers to coordinate shared access and propagate updates across hosts.
  - **Key limitation:** false sharing, excessive data movement, and high communication overhead for fine-grained sharing.
- Software-Managed Coherence in CXL-DSM
  - **Publish** (writer): Write shared data → `CLWB / CLFLUSHOPT → SFENCE` to make updates visible to other hosts.
  - **Observe** (reader): `CLFLUSHOPT → MFENCE → load` to invalidate stale cached copies and fetch updated data.

Related Work

- Scale out distributed application
  - CXL as a communication channel
  - cMPI [ICS'26]: MPI over CXL as shared buffer
- Build CXL-aware applications
  - PolarCXLMem [SIGMOD'25]: database build on CXL
  - Beluga [SIGMOD'26]: sharing KV-Cache on CXL



We seek to scale traditional shared-memory, multi-threaded programs to multiple hosts via shared CXL memory

Do Shared-memory Programs Scale in CXL-DSM?

- Software-managed cross-host coherence
  Bandwidth / Latency / publish & observe operations

paper §III-C

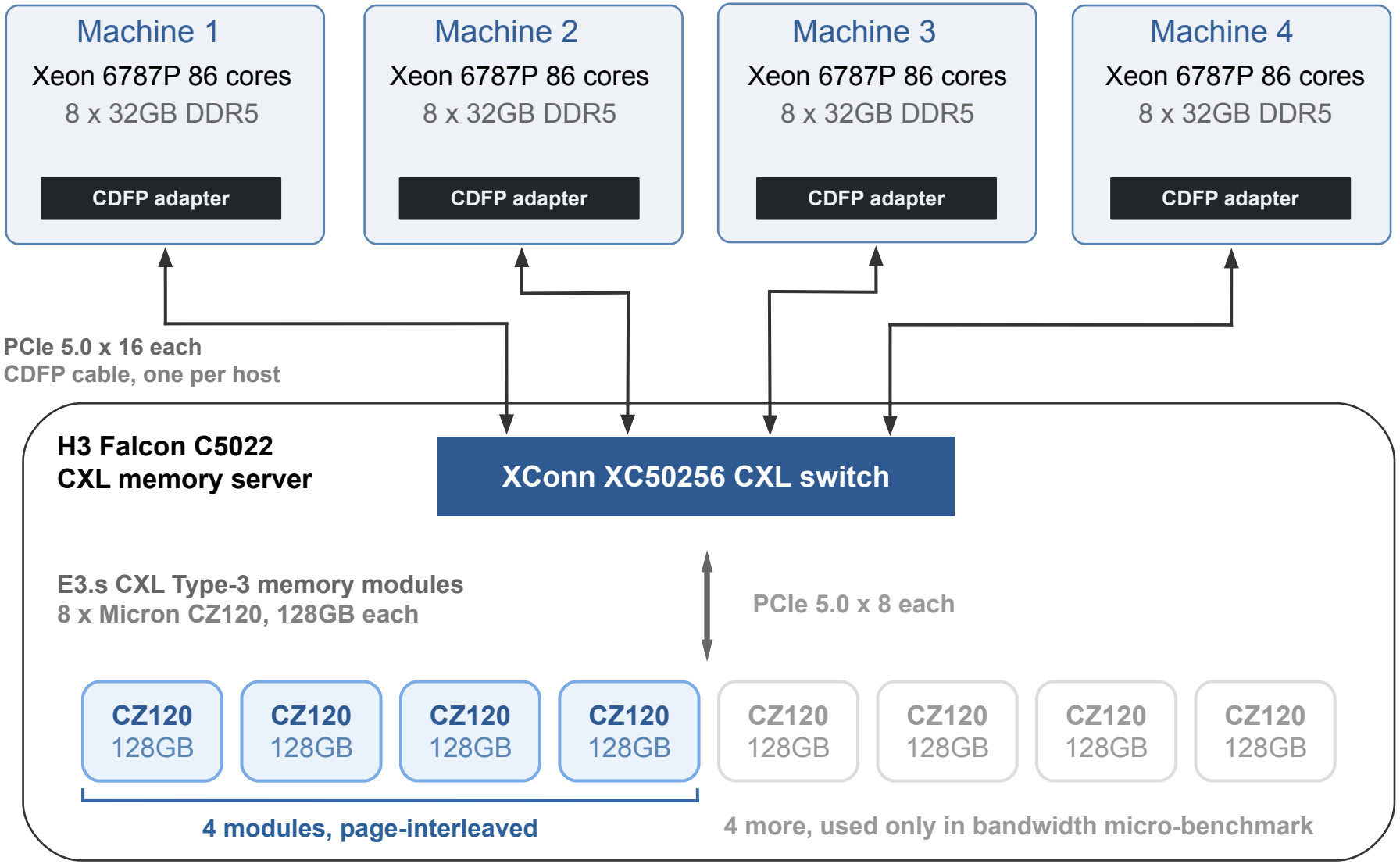
- Cross-host locks: implementation and cost
  Cross-host locks, atomics, barriers

paper §IV

- Applications adapted to shared CXL memory
  BFS, PageRank, triangle counting, and hash table

paper §V

Experimental Platform



Experiment Platform - cont'd

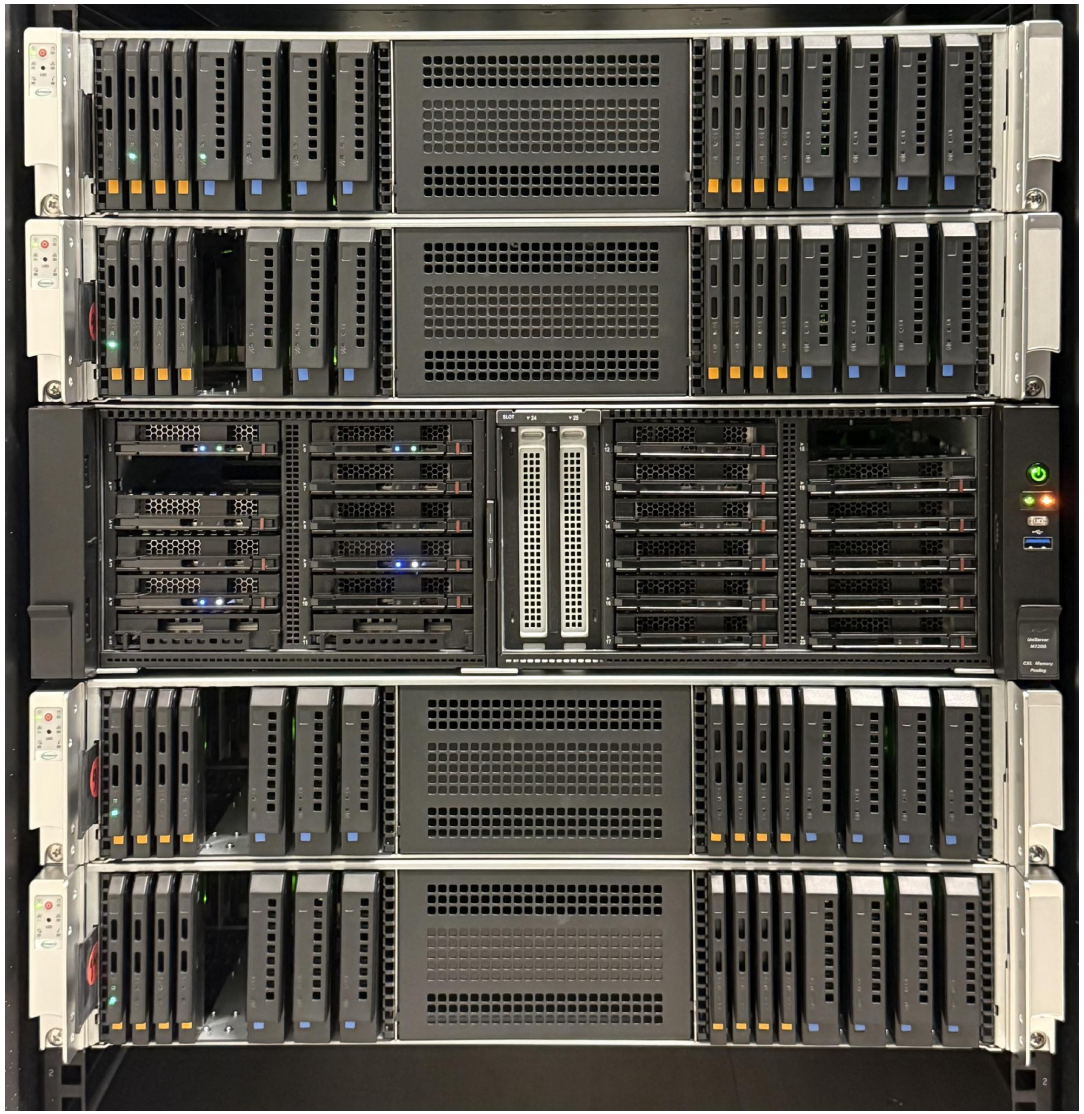
1

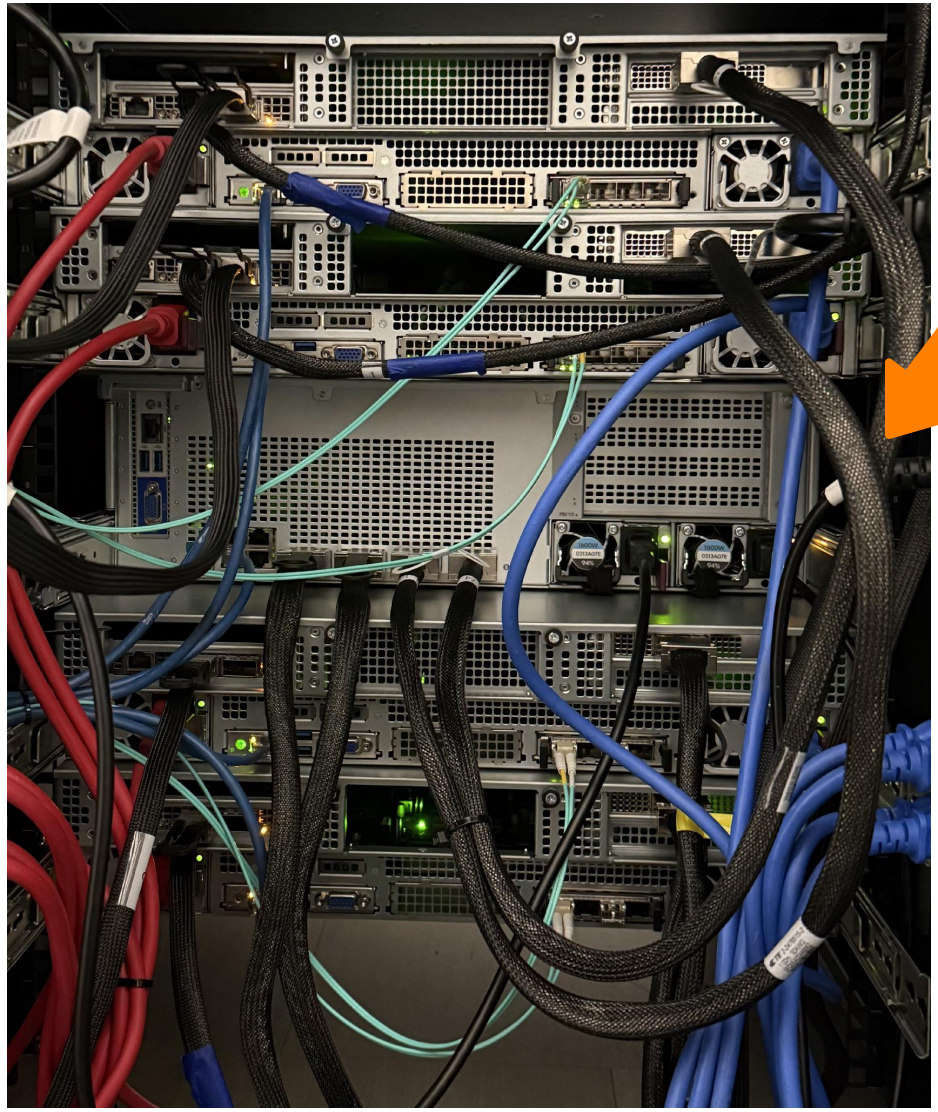
2

CXL

3

4



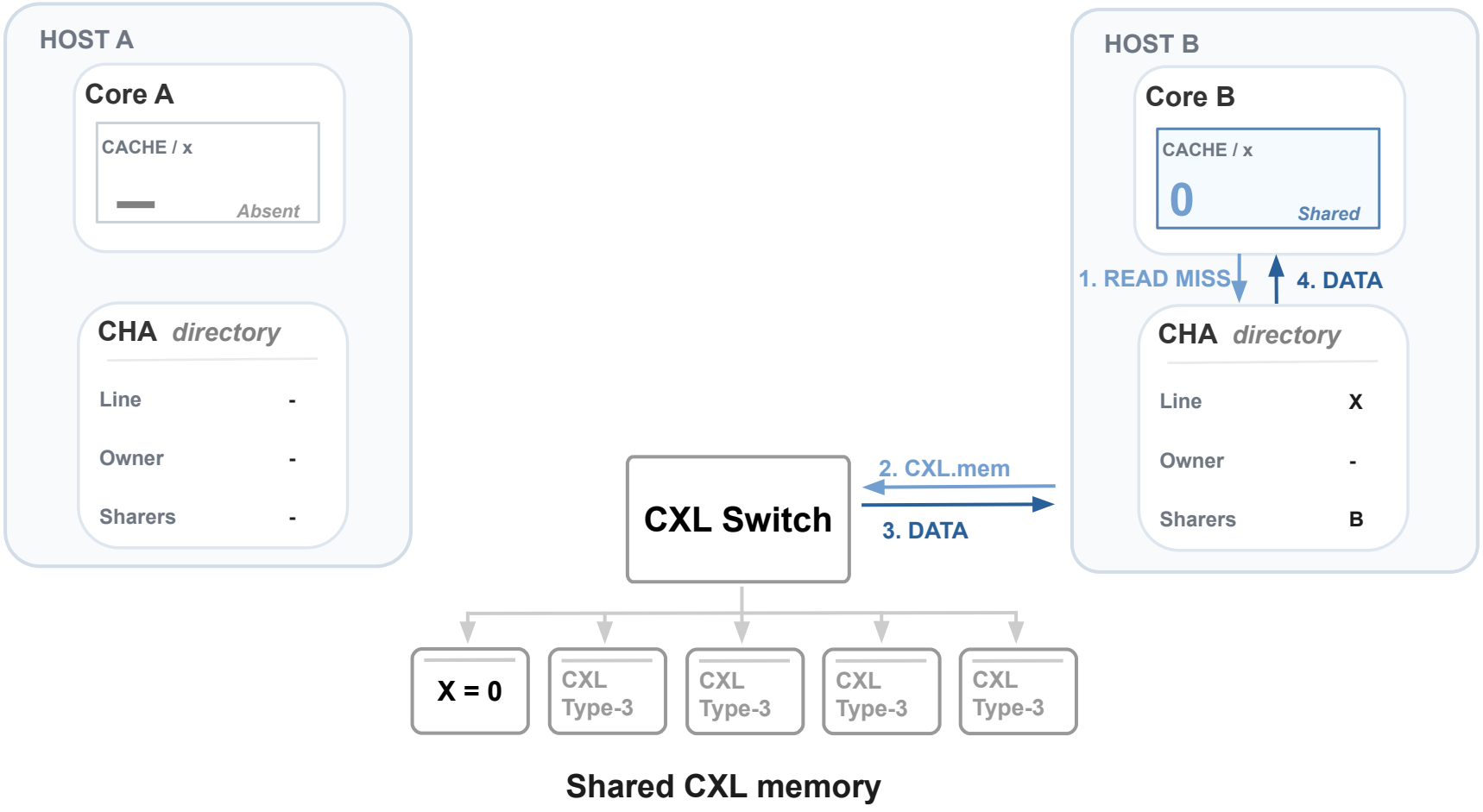


CDFP Cable

No hardware coherence across hosts

Core B loads x | Core A stores x=1 | Core B loads x again

1. B loads x = 0
   - B caches **x = 0**
   - B's CHA tracks B as a sharer

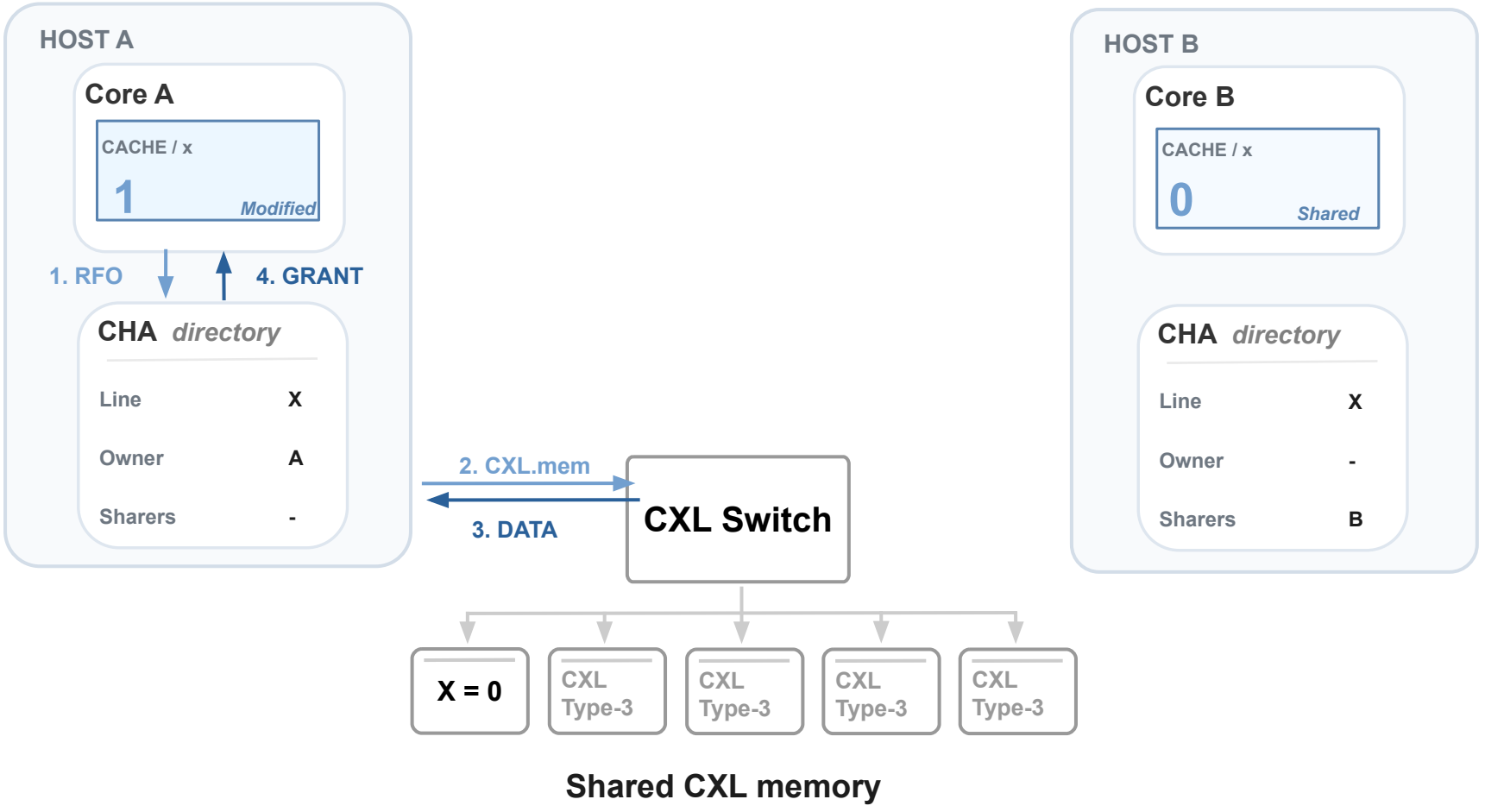


No hardware coherence across hosts

Core B loads x | **Core A stores x=1** | Core B loads x again

1. B loads x = 0
   - B caches x = 0
   - B's CHA tracks B as a sharer

2. A stores x = 1
   - A caches **x = 1** (modified)
   - **No invalidation reaches B**



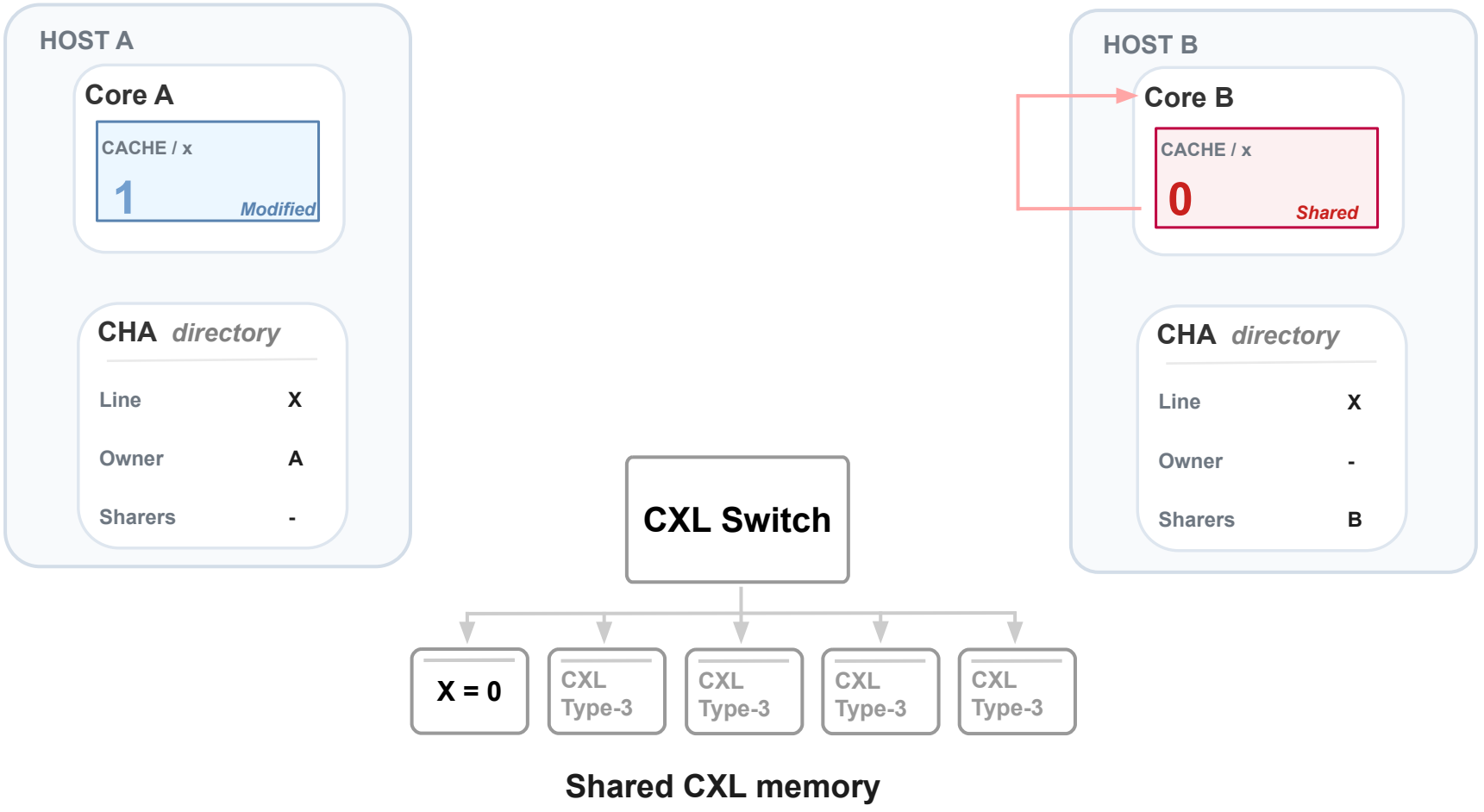
No hardware coherence across hosts

Core B loads x Core A stores x=1 **Core B loads x again**

1. B loads x = 0
   - B caches x = 0
   - B's CHA tracks B as a sharer

2. A stores x = 1
   - A caches X = 1 (modified)
   - No invalidation reaches B

3. B loads x again
   - load **hit local cache x = 0**
   - B return stale **x = 0**



Without hardware coherence, read could return stale value

Flush after every write, flush before every read

No hardware keeps caches on different hosts in step on this platform, so software does it in two moves · paper §II-C

Publish

writer, after the store



Store → CLWB / CLFLUSHOPT → MFENCE

Observe

reader, before the load

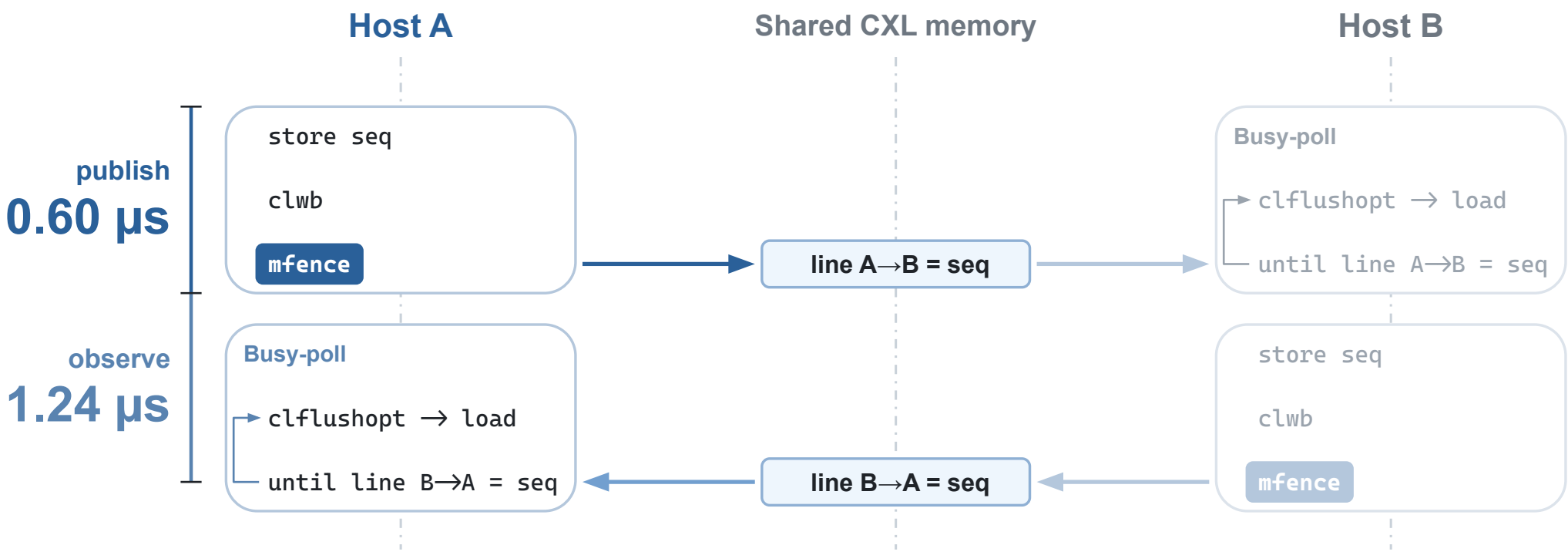


CLFLUSHOPT → MFENCE → Load

Order Intel SDM Vol. 3A 11-7

```
clflush / clflushopt → mfence → clflush / clflushopt
read / write → mfence → read / write
```

RTT Latency



RTT = **1.87 µs** One-way = **0.94 µs**

RDMA write (perftest) one way: 1.91 µs 400Gb/s IB via switch

NVIDIA BlueField-3 NIC NVIDIA Quantum-2 MQM9700 NDR IB switch

Locks without cross-host atomics

A Fast Mutual Exclusion Algorithm Lamport, 1987

```
// Lamport, 1987
lock(i):
  b[i] = 1                          // write
  x = i                             // write
  if y ≠ 0:                         // read
    b[i] = 0; wait y == 0; retry
  y = i                             // write
  if x ≠ i:                         // read
    b[i] = 0; wait all b[j] == 0
    if y ≠ i: wait y == 0; retry
  critical section

unlock(i):
  y = 0;  b[i] = 0
```

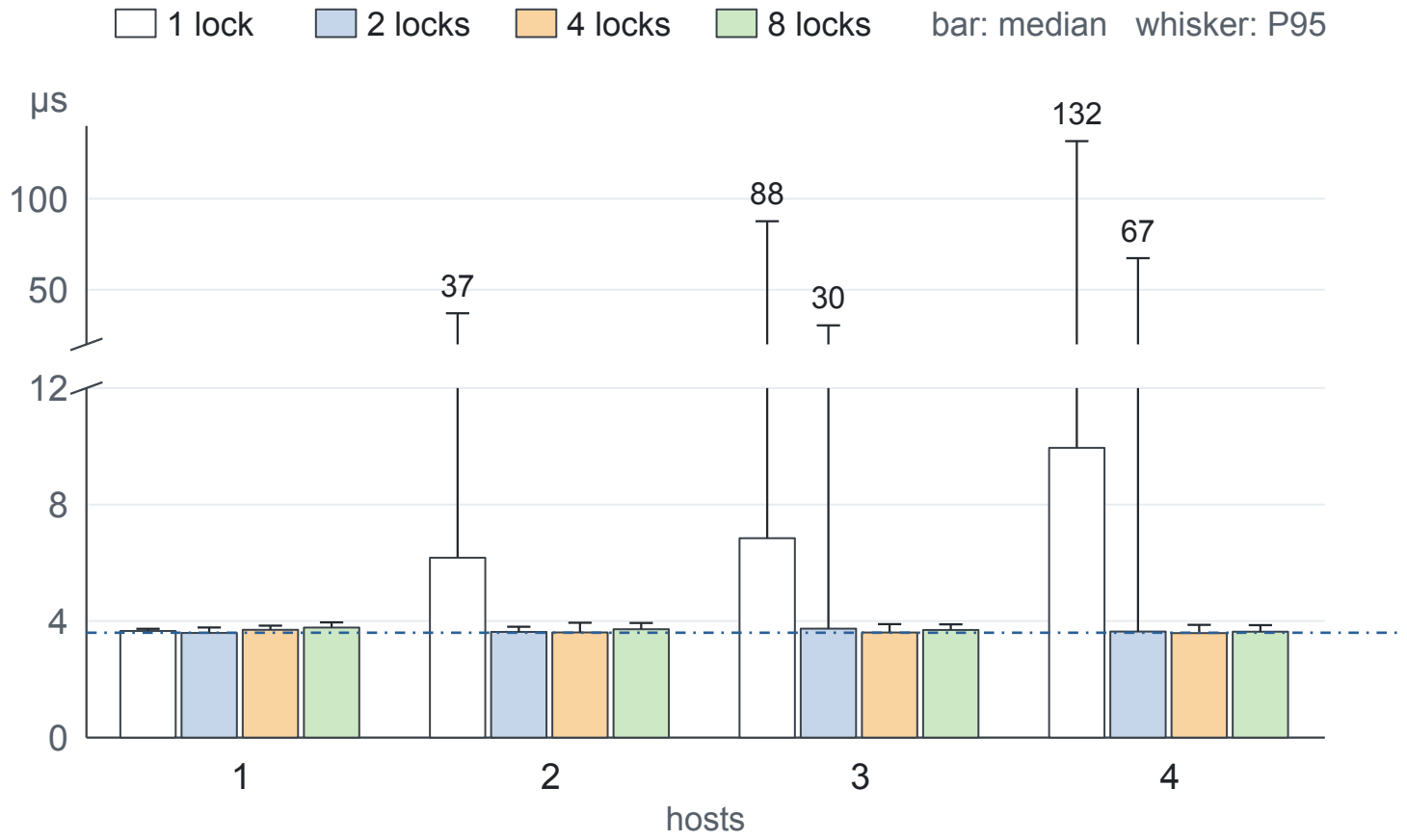
Lamport's proof assumes

atomic reads and writes
=> each 8 B variable in its own cache line

one order of all accesses
=> every write flushed, every read invalidated

Alone, a lock takes 3 writes and 2 reads; under contention it polls and retries.

Lamport Fast Mutex latency



1 lock, 4 hosts

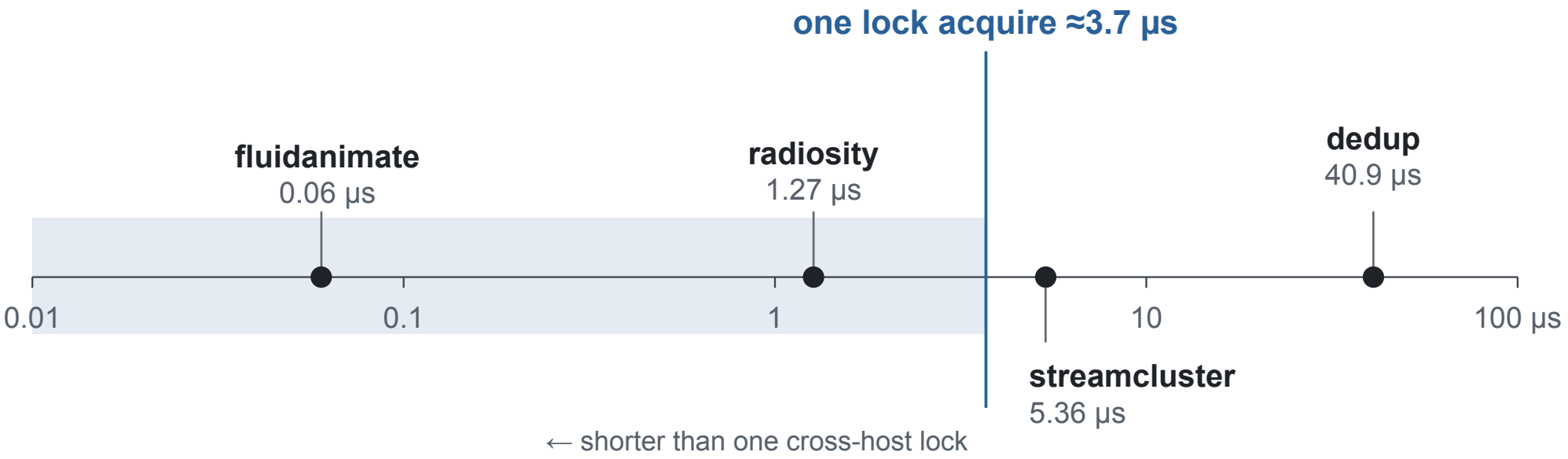
P95 132 µs

locks ≥ hosts

≈3.6 µs

Lamport Fast Mutex kept ≈3.6 µs and a short tail only with at least as many locks as hosts

Critical sections vs. one cross-host lock



Ported as is, some critical sections would be shorter than the lock around them

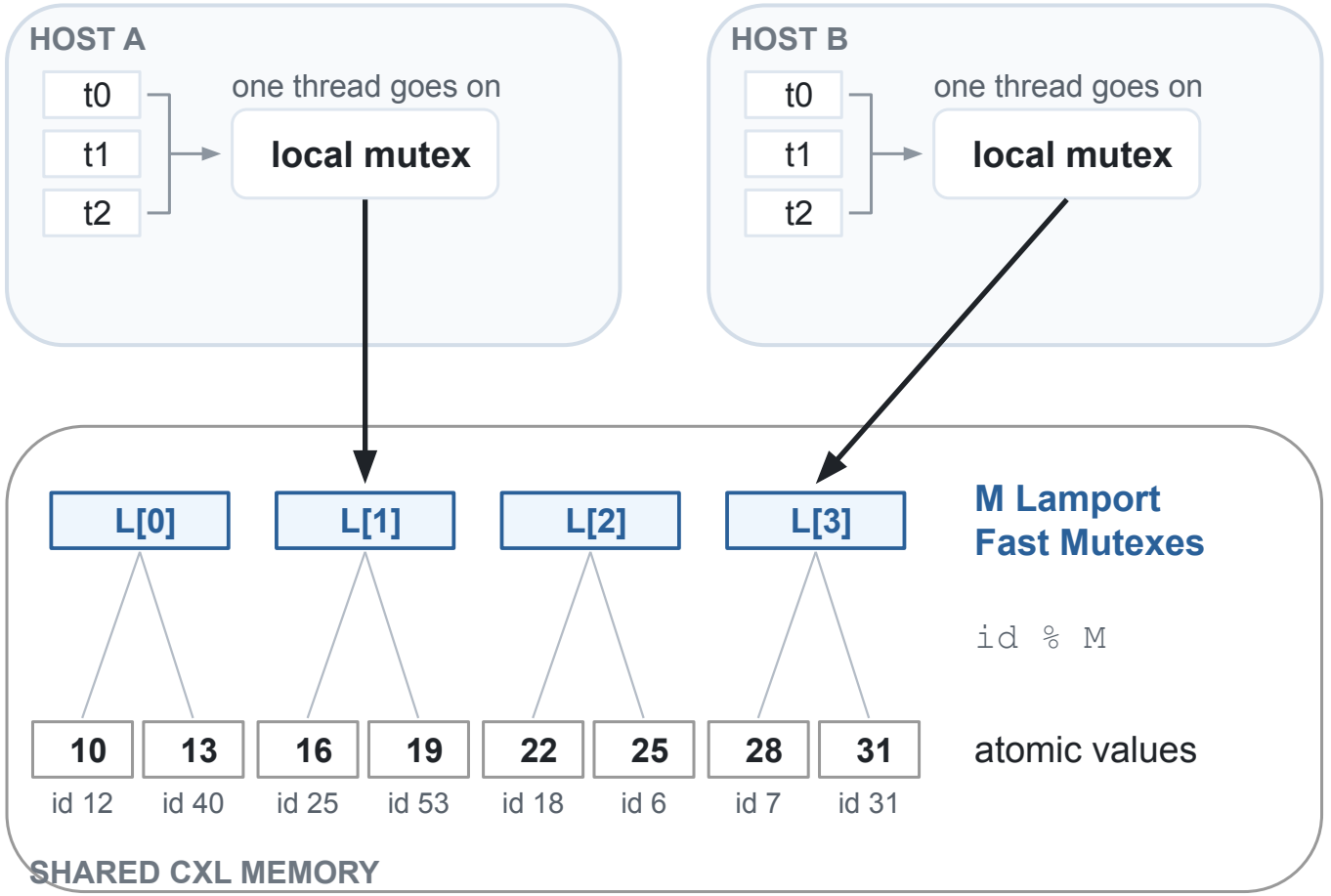
Cross-host atomics on Lamport Fast Mutexes

```
lock(L[id % M])
x += 1
unlock(L[id % M])
```



value, 8 B

Lamport Fast Mutex, 960 B

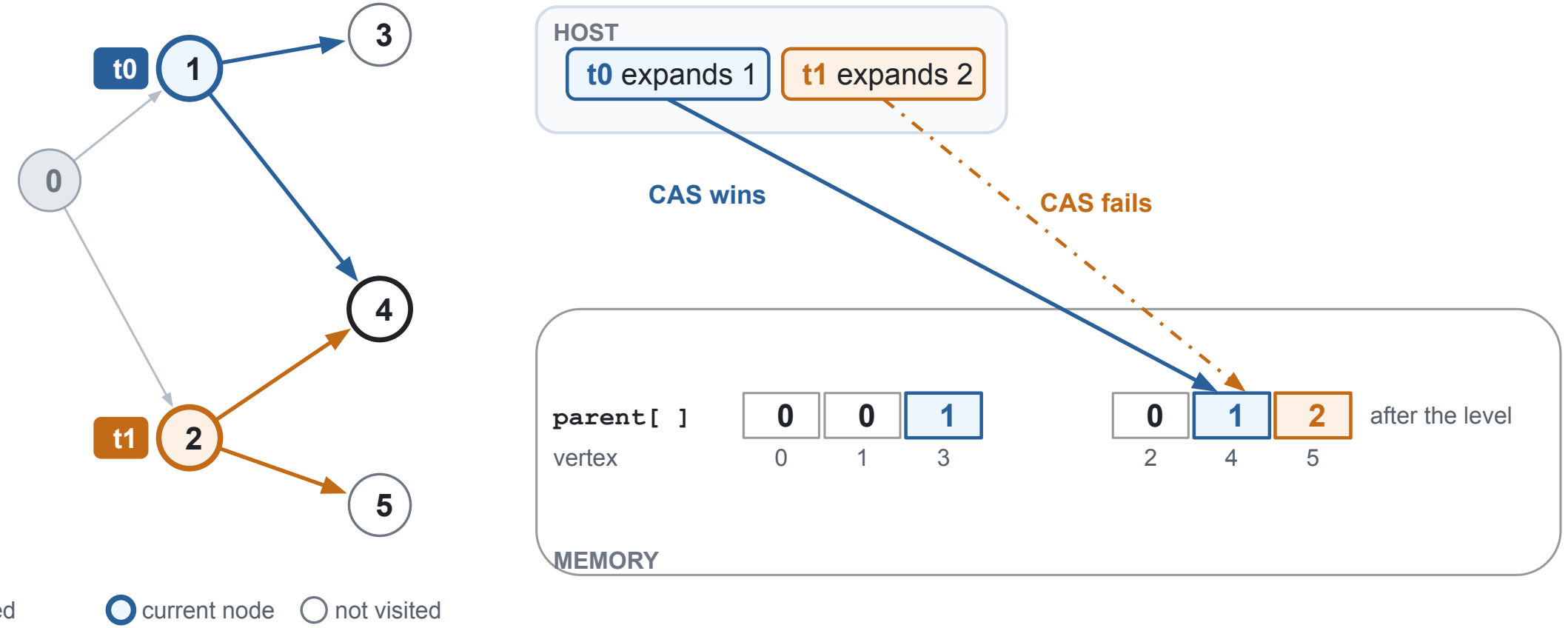


Fewer locks than atomic values, each rarely contended

Atomics share Lamport Fast Mutexes by id: little space, and with many locks, hosts rarely meet

Top-down BFS across hosts

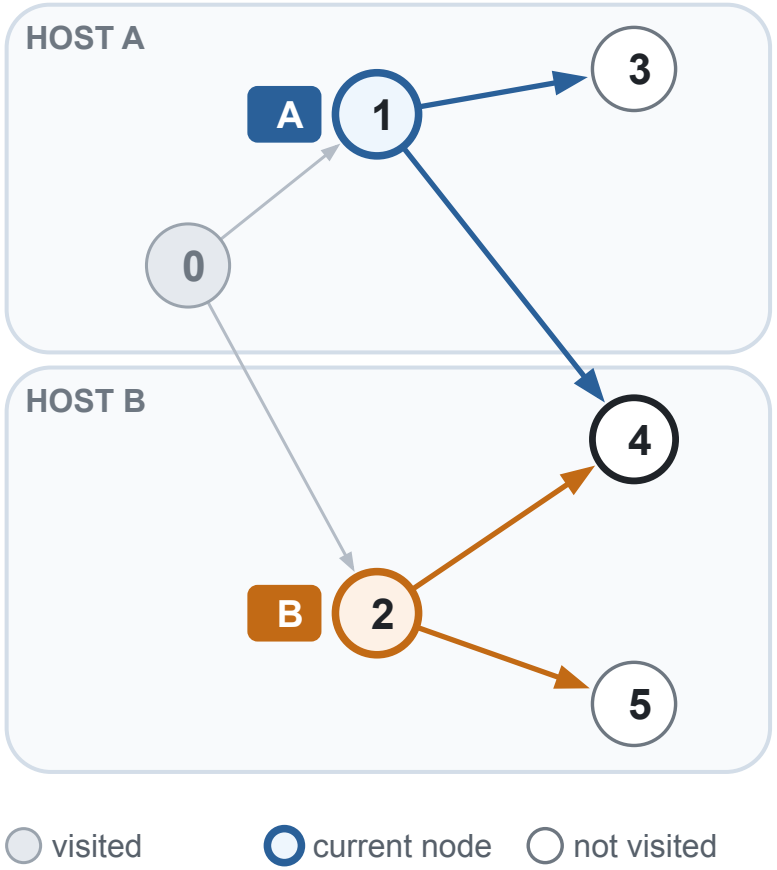
BFS on one host | Across hosts: cross-host lock | Our port



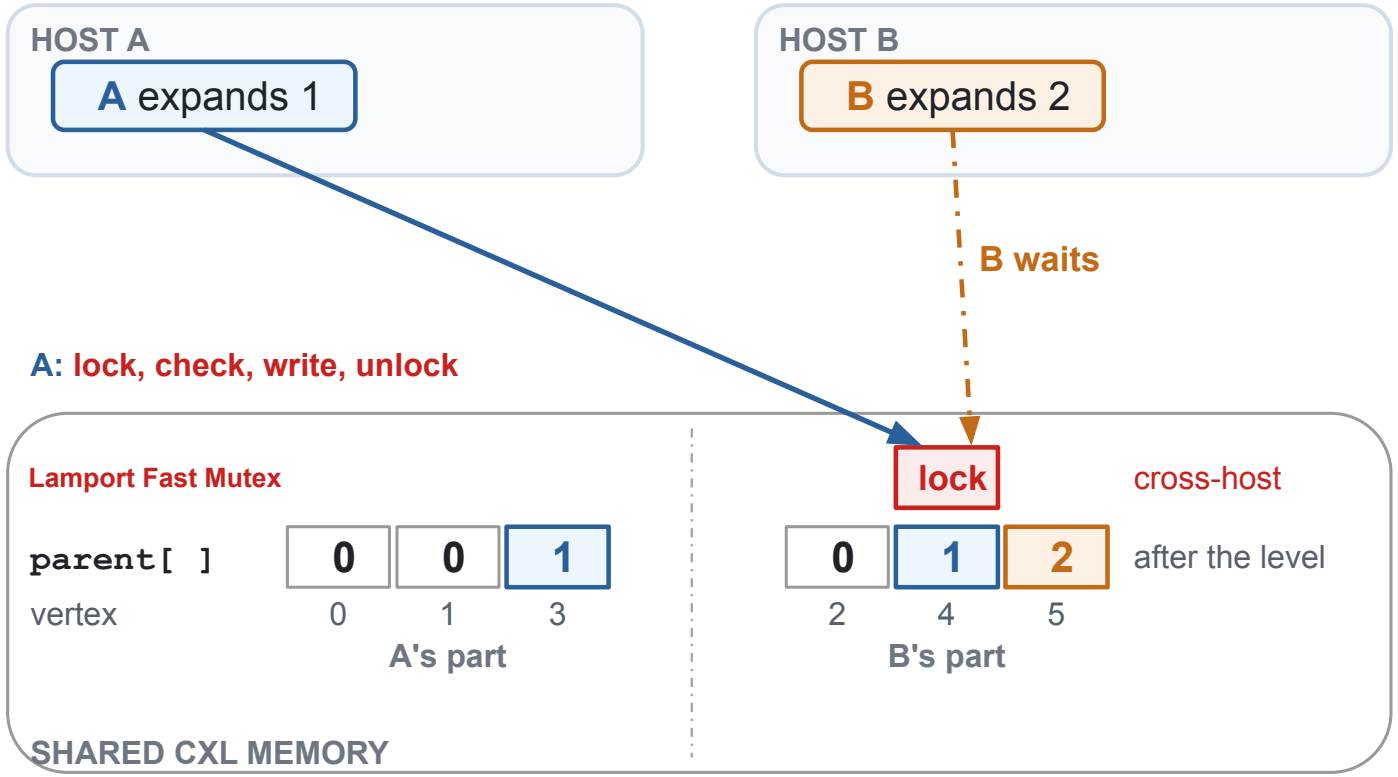
On one host, a hardware CAS lets exactly one thread claim vertex 4

Top-down BFS across hosts

BFS on one host | **Across hosts: cross-host lock** | Our port



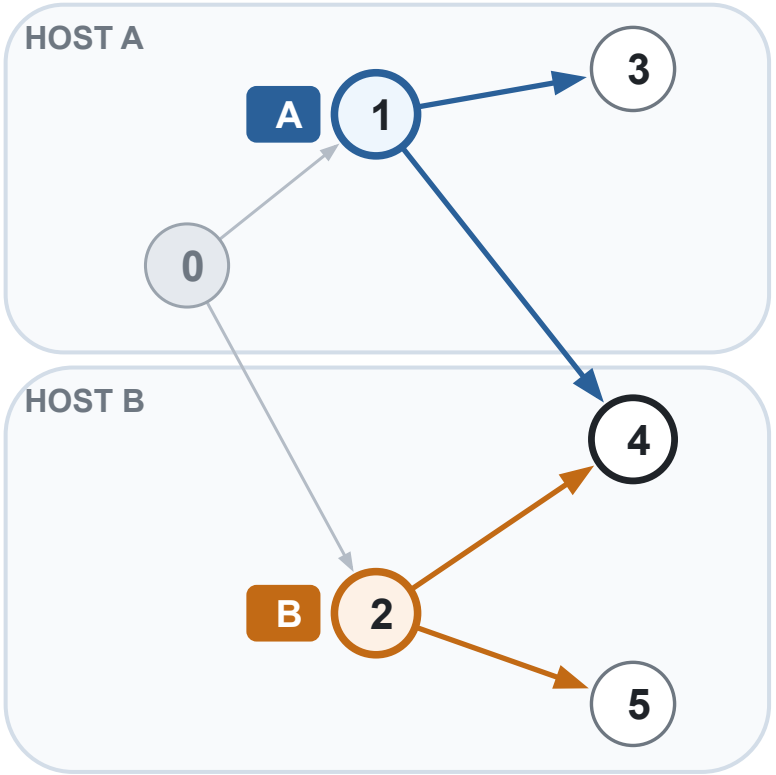
visited | current node | not visited



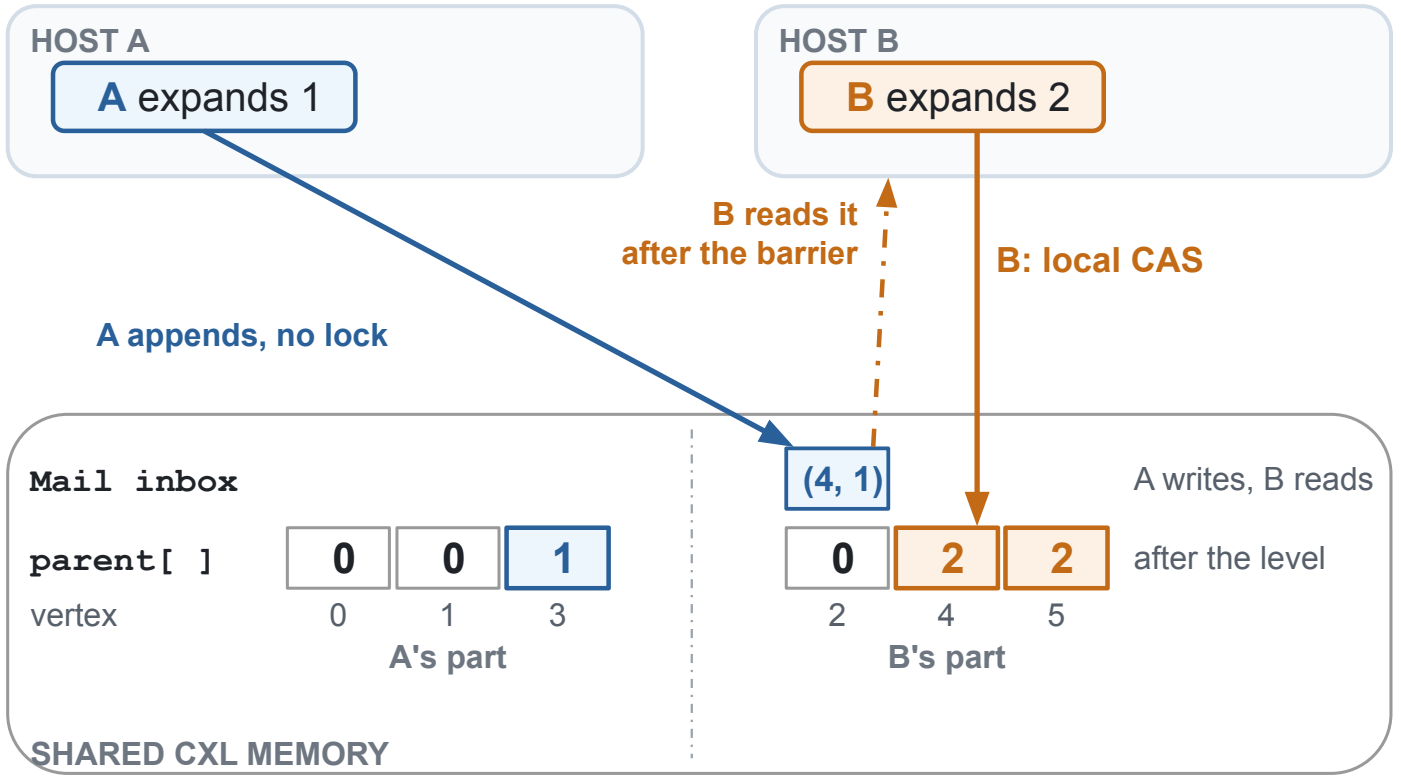
Across hosts, a CAS is not atomic: A and B meet on a Lamport Fast Mutex, ≈3.7 µs even uncontended

Top-down BFS across hosts

BFS on one host | Across hosts: cross-host lock | **Our port**



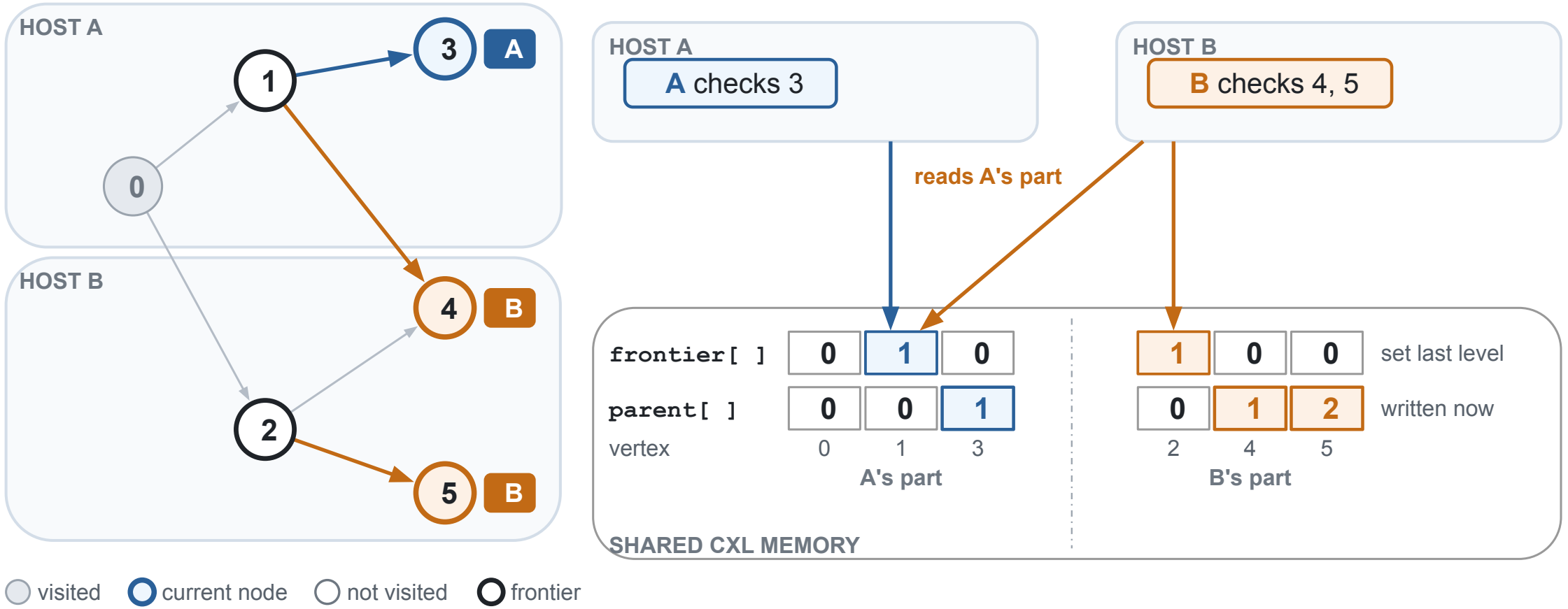
visited | current node | not visited



shared across hosts: only the inbox data and the level barrier

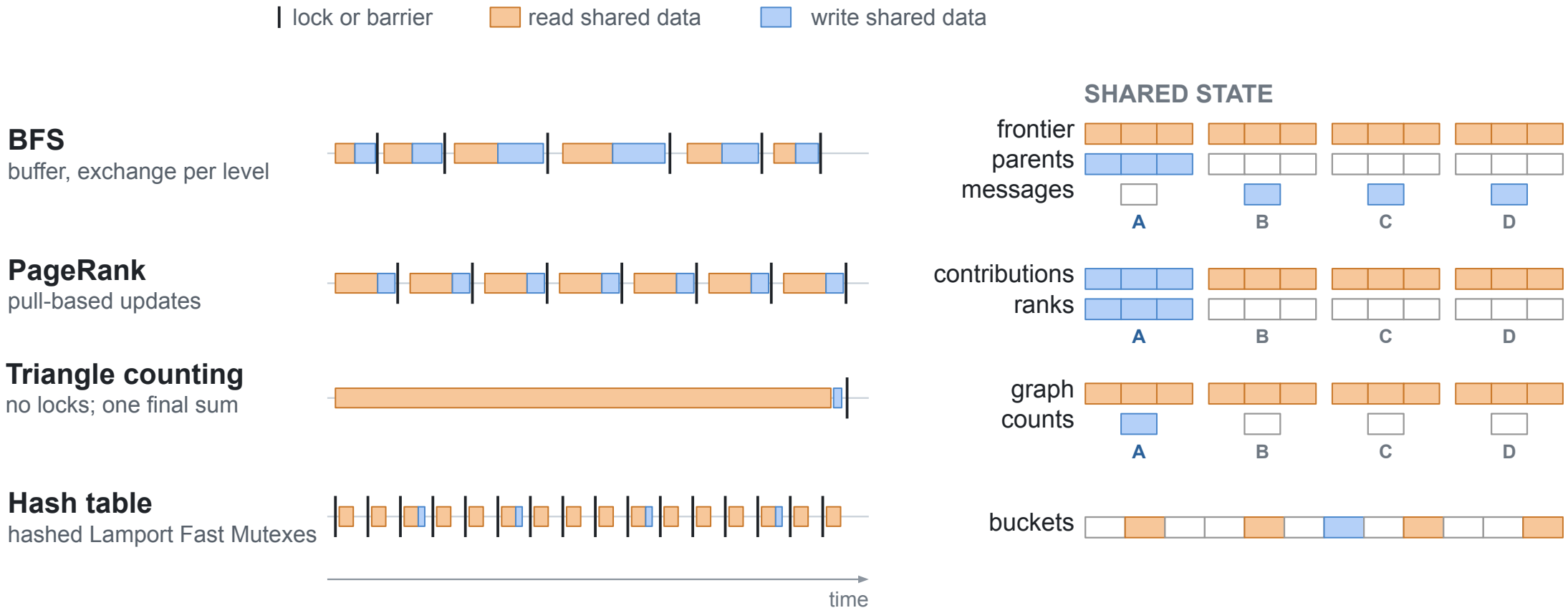
Our port: no cross-host lock; A's claim travels as data, B claims vertex 4 with its own CAS

Bottom-up BFS across hosts



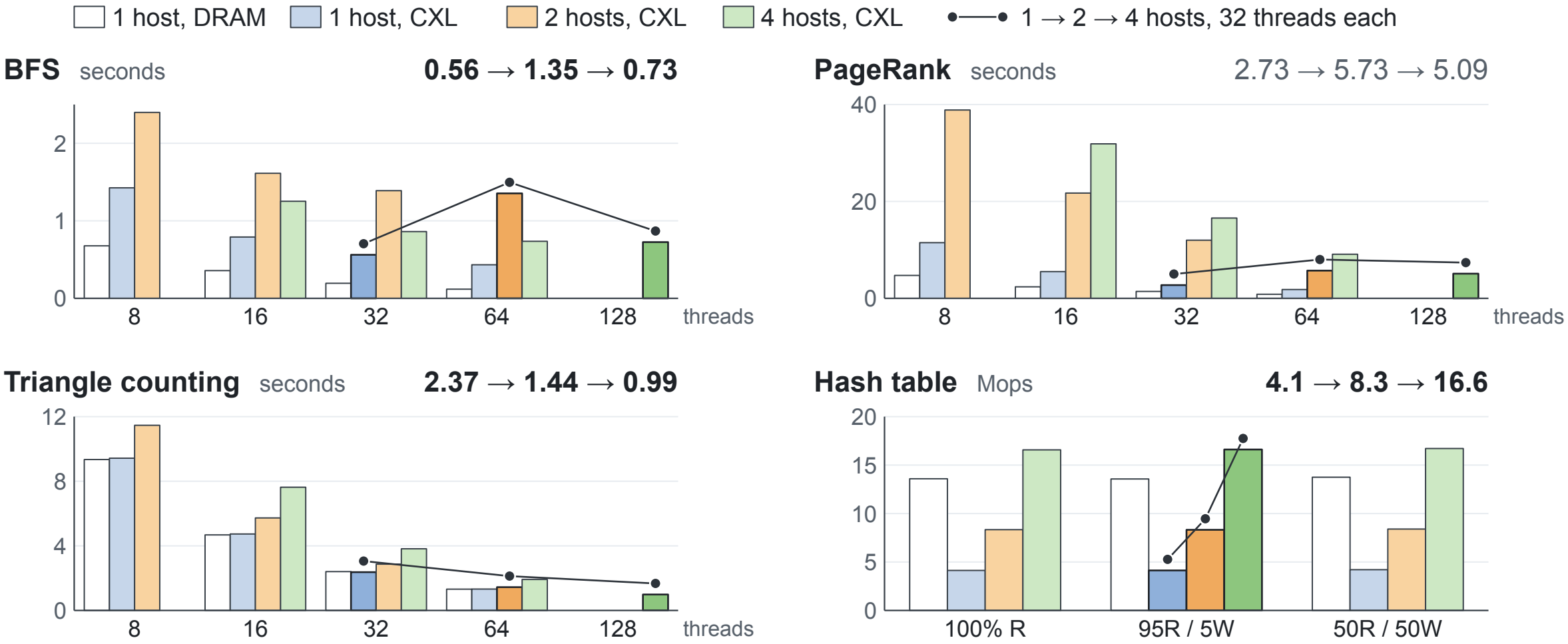
Bottom-up needs no lock, but every level each host re-reads the frontier the others wrote

Benchmarks with different sharing patterns

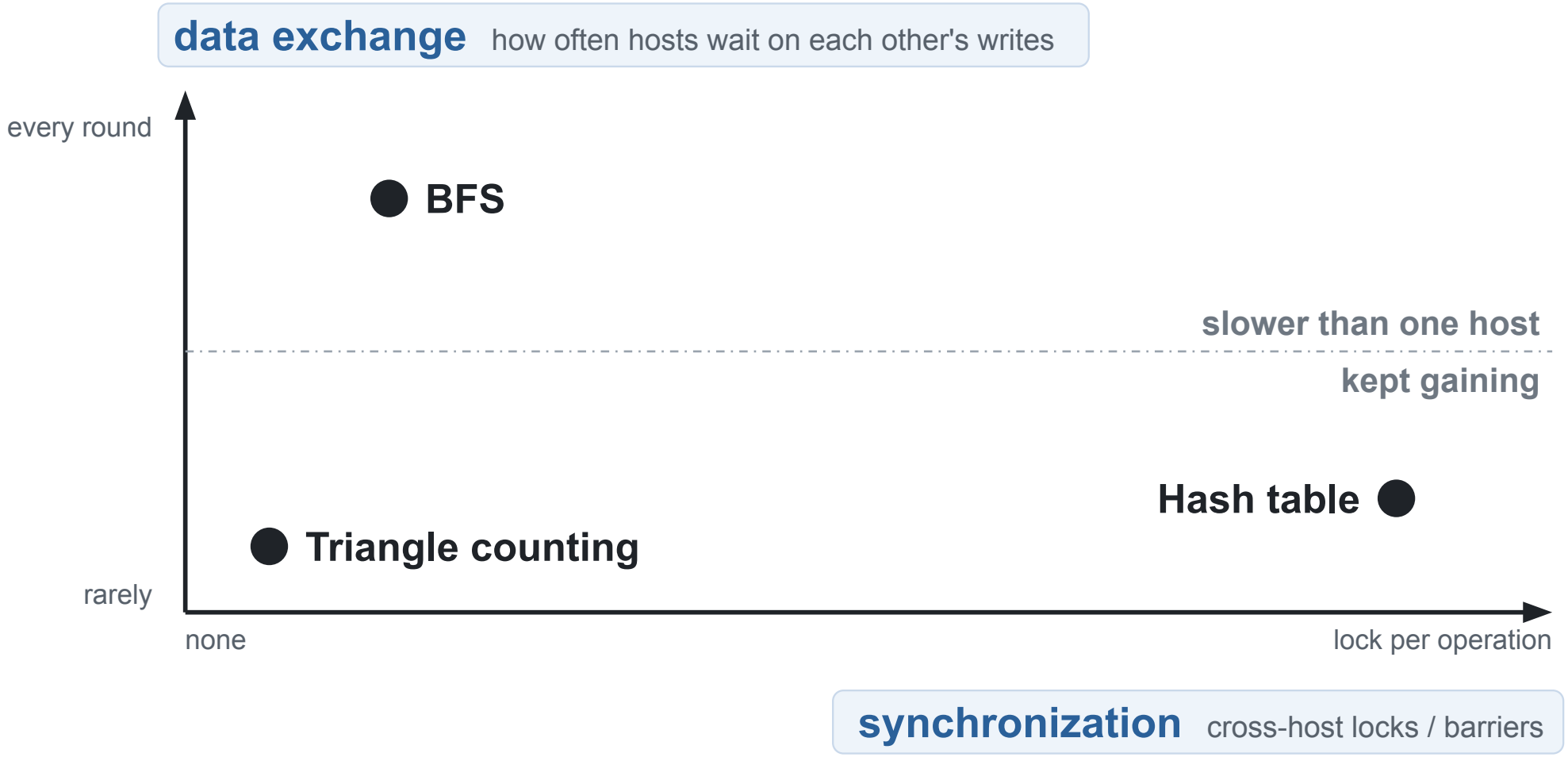


BFS and PageRank wait on all hosts every round; triangle counting only at the end; the hash table rarely

Application performance



Where did more hosts help?



More hosts helped where hosts rarely waited on each other, even with a lock on every operation

Summary

Software-managed coherence across hosts

0.94 µs one way, **1.87 µs** round trip

Cross-host locks: implementation and cost

Lamport Fast Mutex **3.7 µs** alone; up to **132 µs** P95 under contention

Applications adapted to shared CXL memory

Two axes: **synchronization** and **data exchange**



Paper, Artifact, Data

dupeiran001.github.io/blog/papers/sc26-cxl-shm-bench

github.com/dupeiran001/cxl-shm-bench
doi.org/10.5281/zenodo.21753320