Pragh:

Locality-preserving Graph Traversal with Split Live Migration

Xiating Xie, Xingda Wei, Rong Chen, Haibo Chen

Shanghai Jiao Tong University

IPADS
INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

Graphs are Everywhere

Graphs are Everywhere

Graphs are Everywhere

Alipay™

Pay

DBpedia

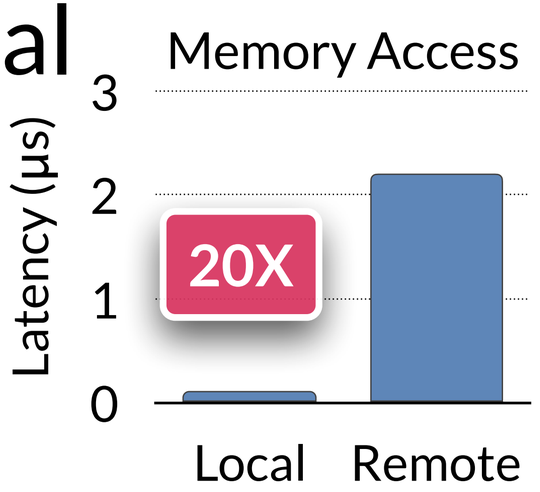
Graph ***traversal queries*** are key operations
to support emerging applications.

Urban Monitoring

User Profiling

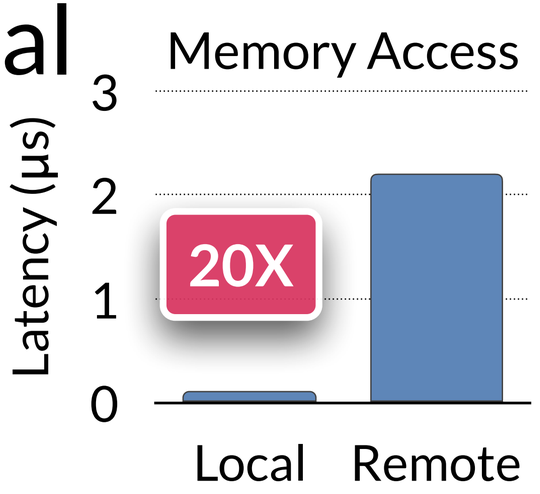
Locality is Important on Graph Traversal

Local access is **faster** than remote access (cross-node)

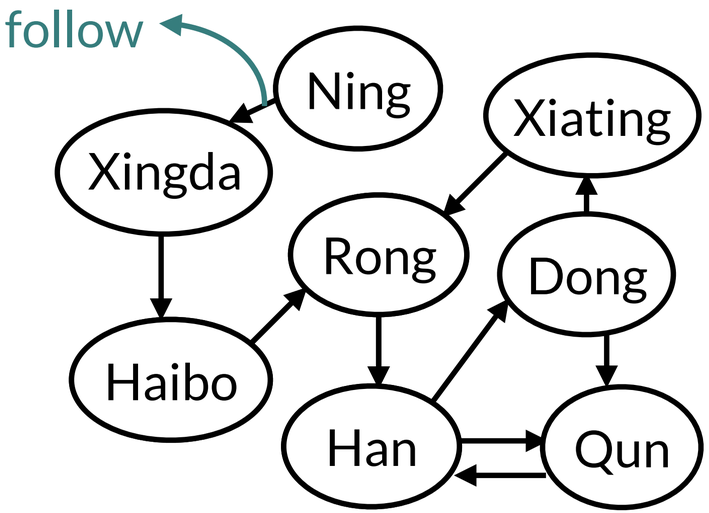


Locality is Important on Graph Traversal

Local access is **faster** than remote access (cross-node)

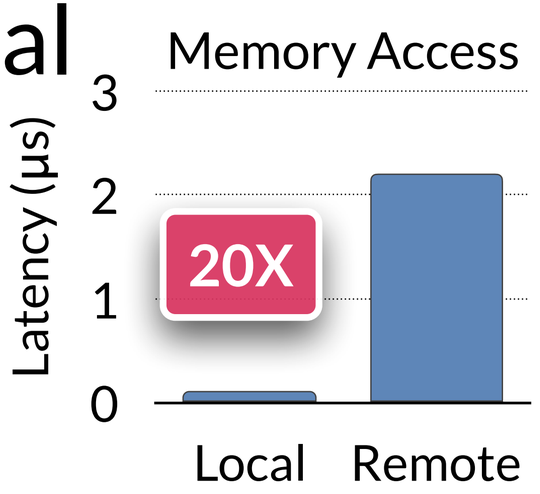


Who is followee of followee of ***Haibo*** ?

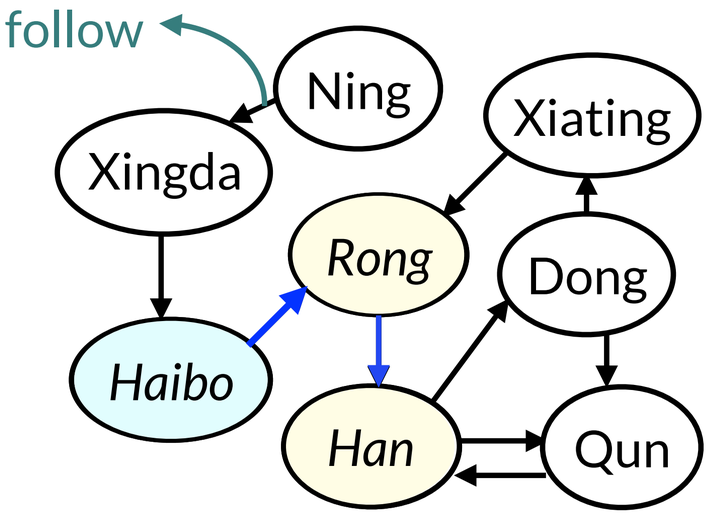


Locality is Important on Graph Traversal

Local access is **faster** than remote access (cross-node)

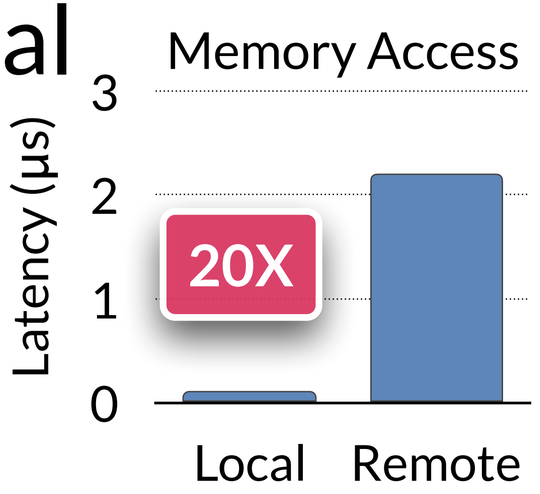


Who is followee of followee of ***Haibo*** ?

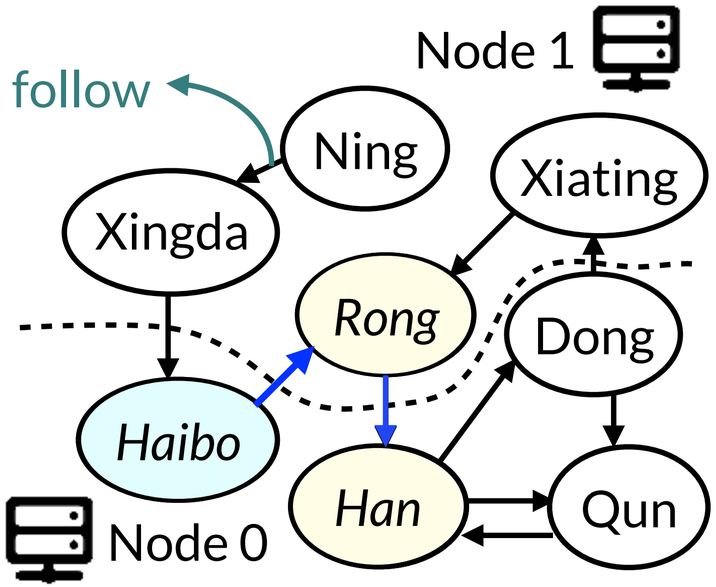


Locality is Important on Graph Traversal

Local access is **faster** than remote access (cross-node)

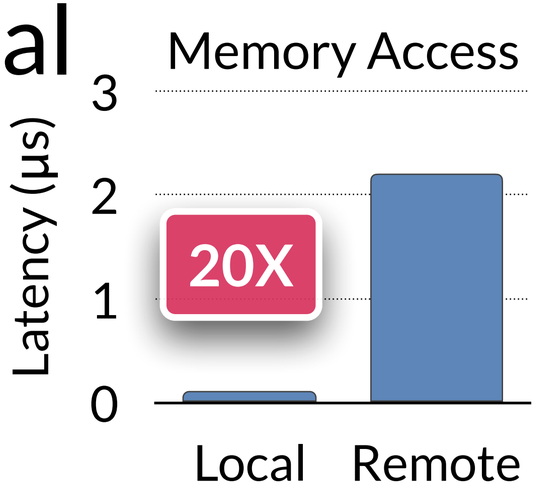


Who is followee of followee of ***Haibo*** ?

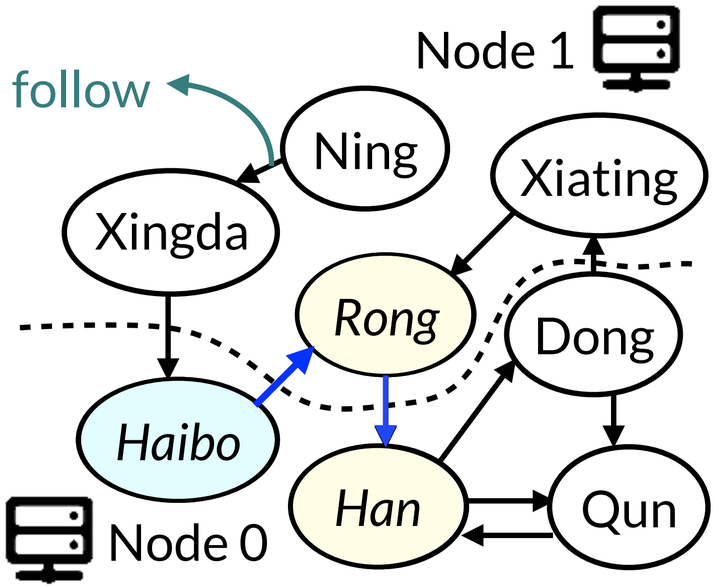


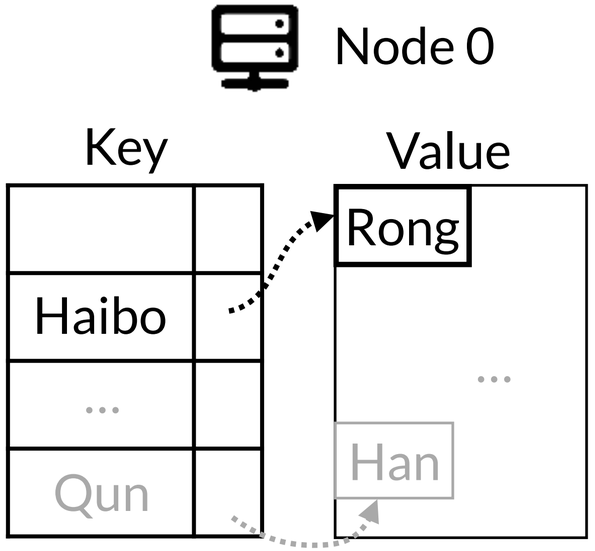
Locality is Important on Graph Traversal

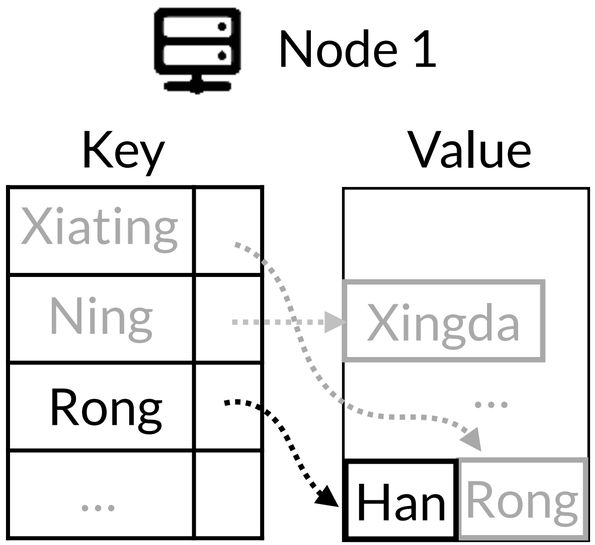
Local access is **faster** than remote access (cross-node)



Who is followee of followee of ***Haibo*** ?

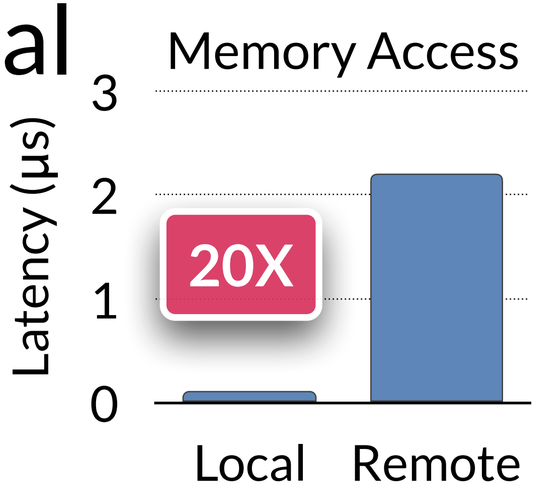




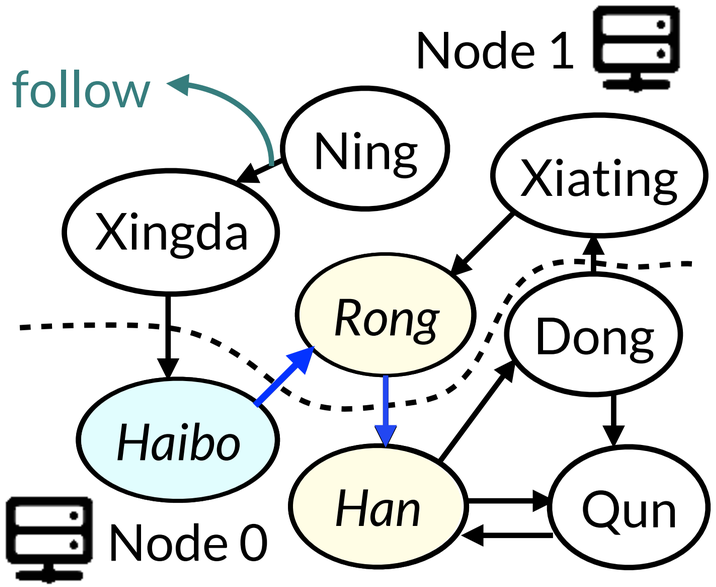


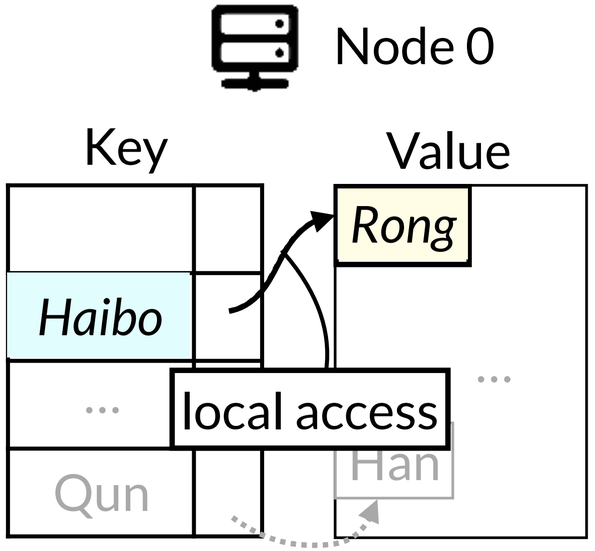
Locality is Important on Graph Traversal

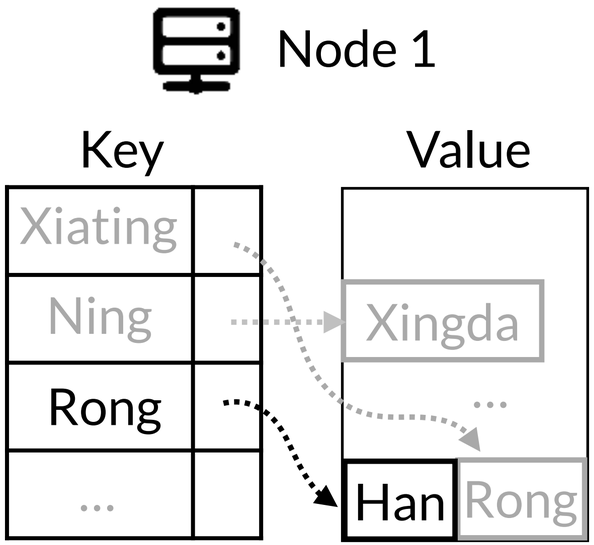
Local access is **faster** than remote access (cross-node)



Who is followee of followee of ***Haibo*** ?

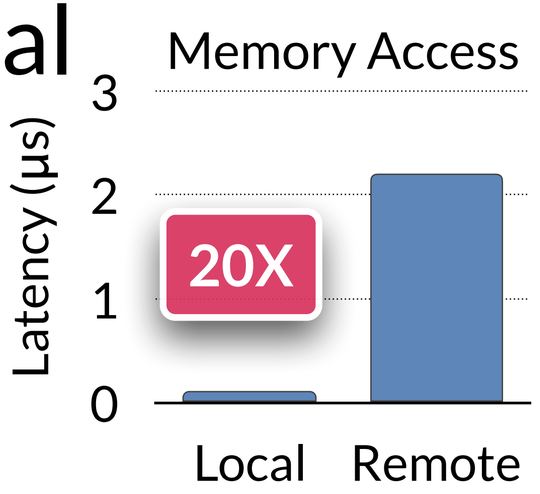




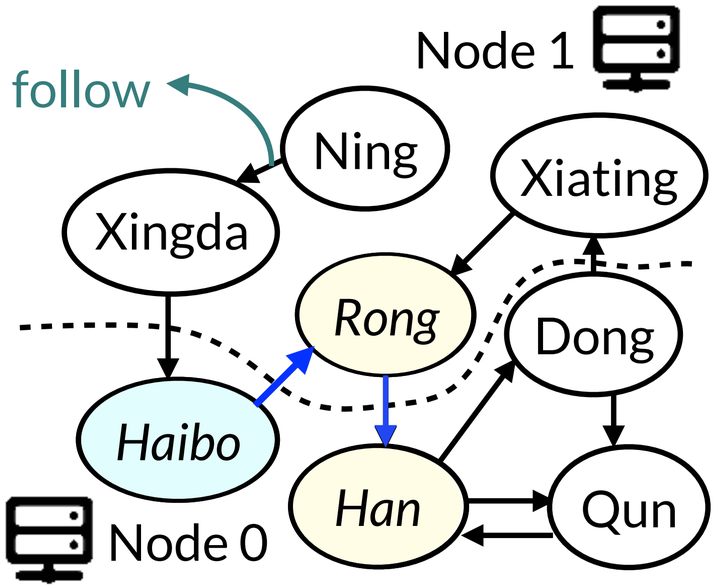


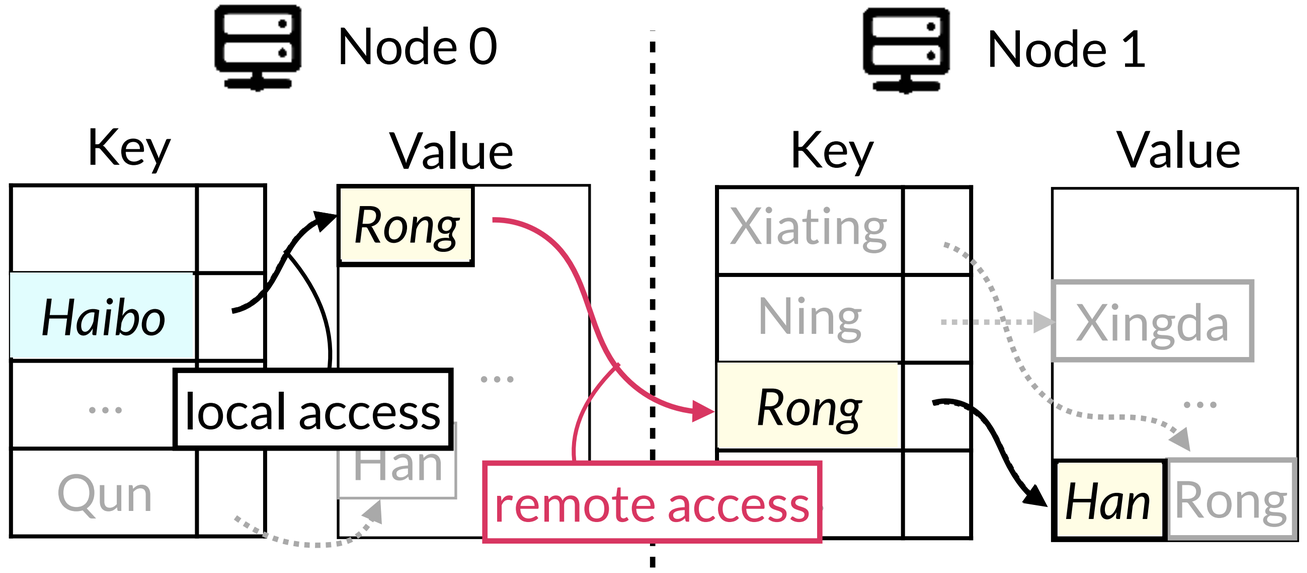
Locality is Important on Graph Traversal

Local access is **faster** than remote access (cross-node)



Who is followee of followee of ***Haibo*** ?





Locality is Challenging on Dynamic Workload

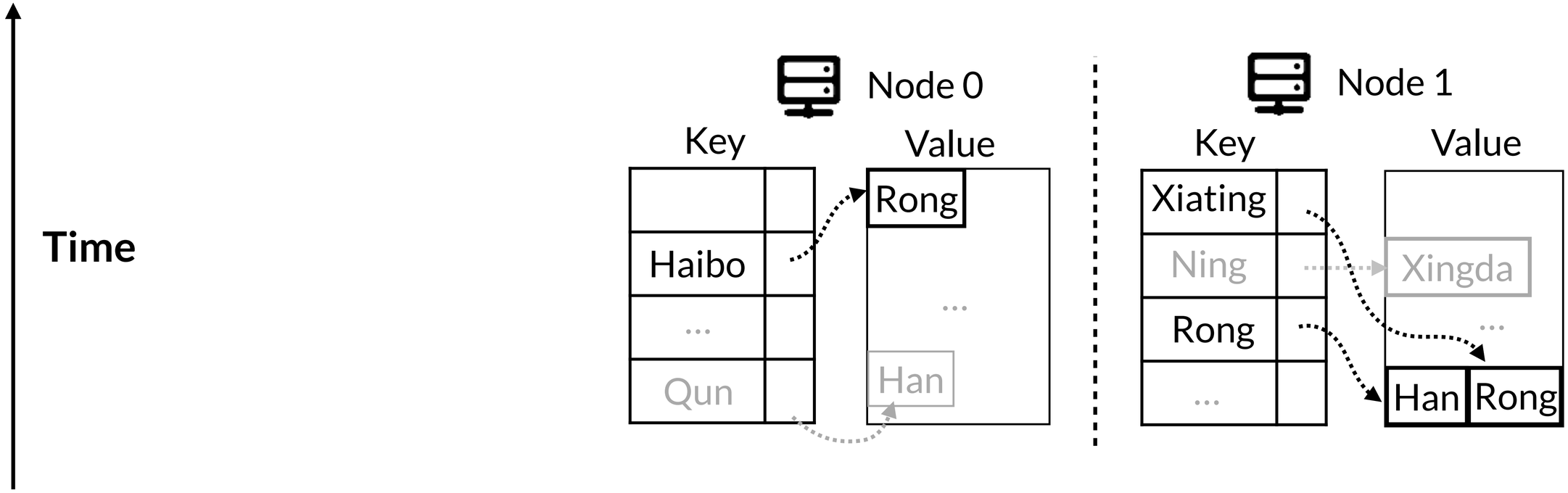
Workload changes over time

- One static partition scheme cannot fit all

Locality is Challenging on Dynamic Workload

Workload changes over time

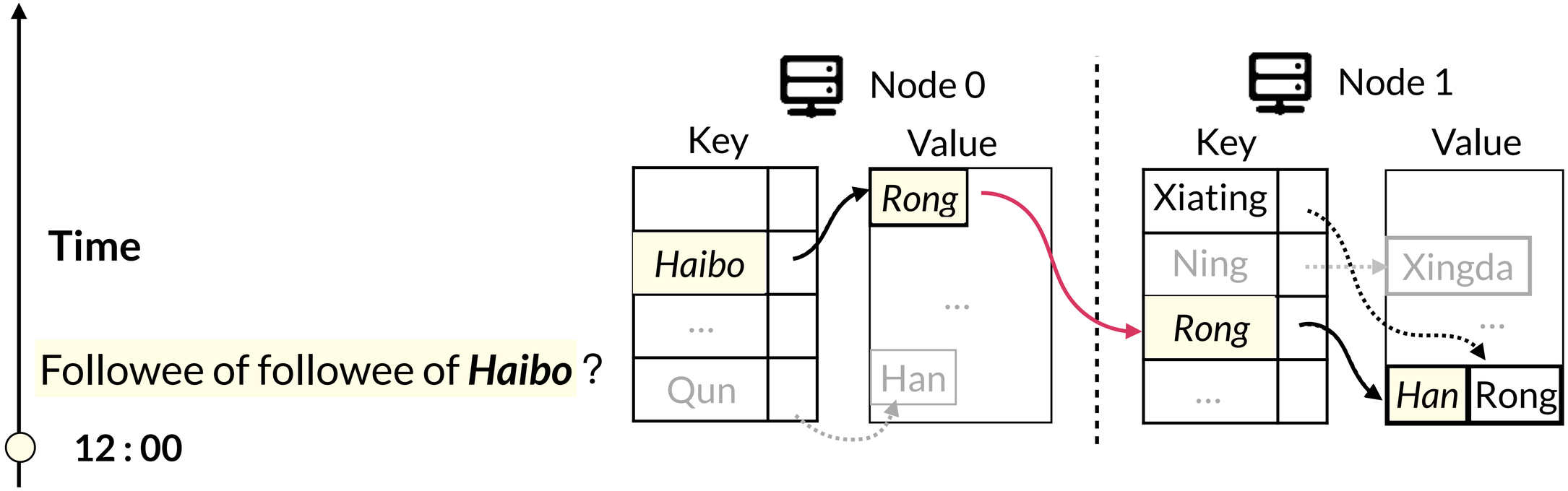
- One static partition scheme cannot fit all



Locality is Challenging on Dynamic Workload

Workload changes over time

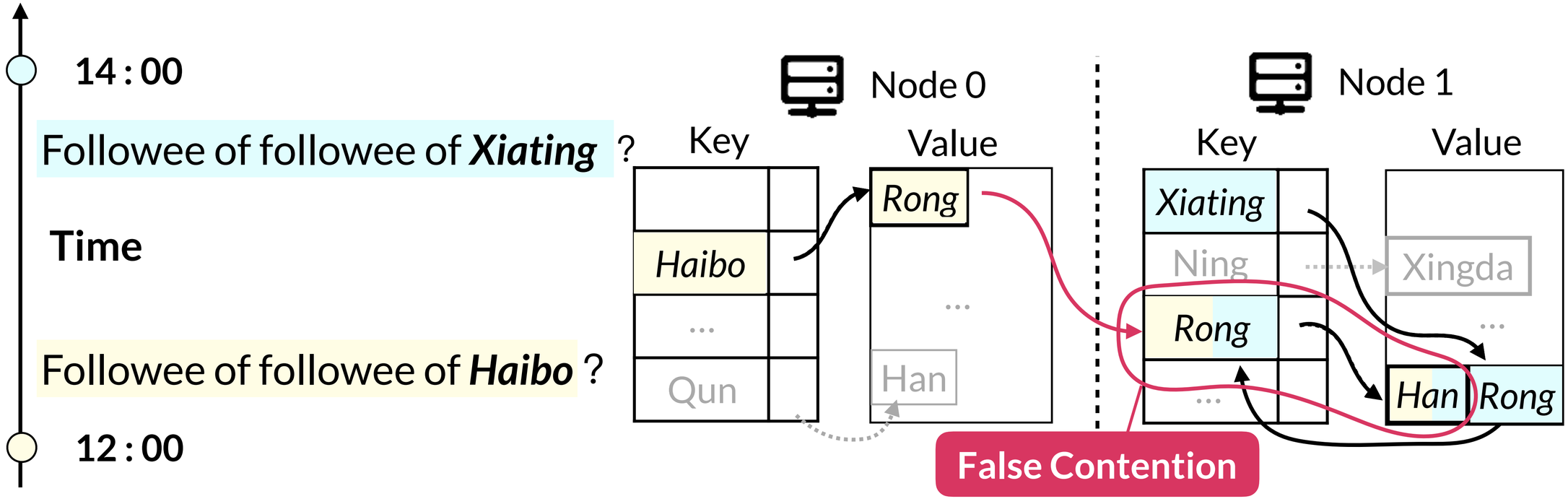
- One static partition scheme cannot fit all



Locality is Challenging on Dynamic Workload

Workload changes over time

- One static partition scheme cannot fit all



How to preserve Locality? Live Migration

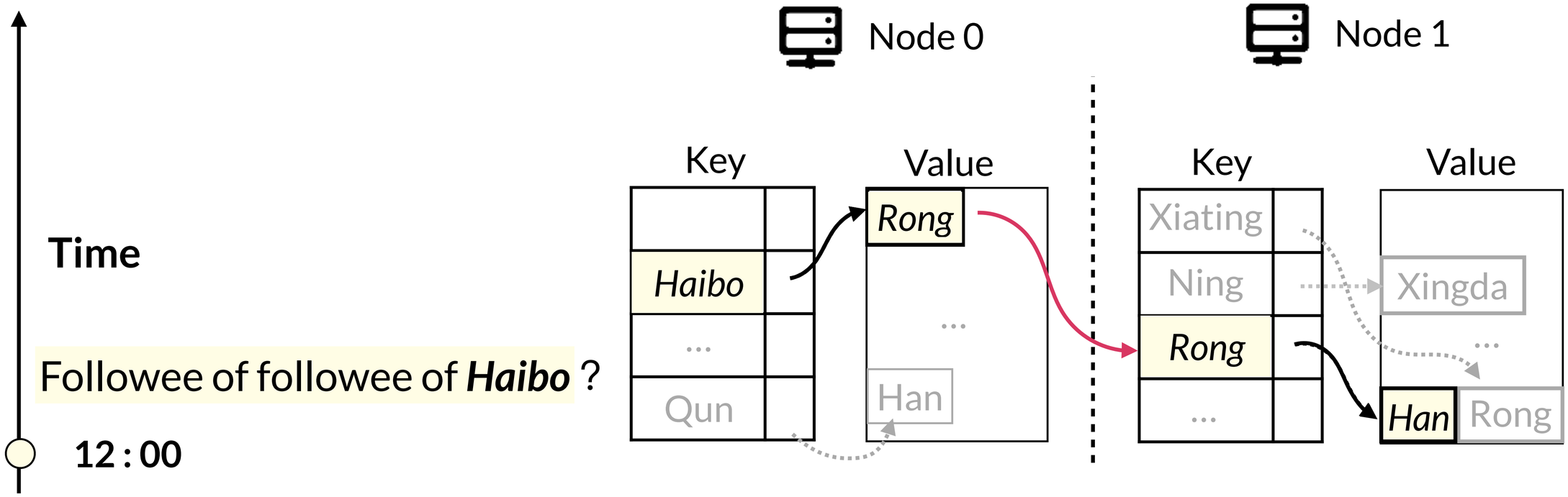
Workload is specific **in a period**

- Workload Change ⟶ Migration

How to preserve Locality? Live Migration

Workload is specific **in a period**

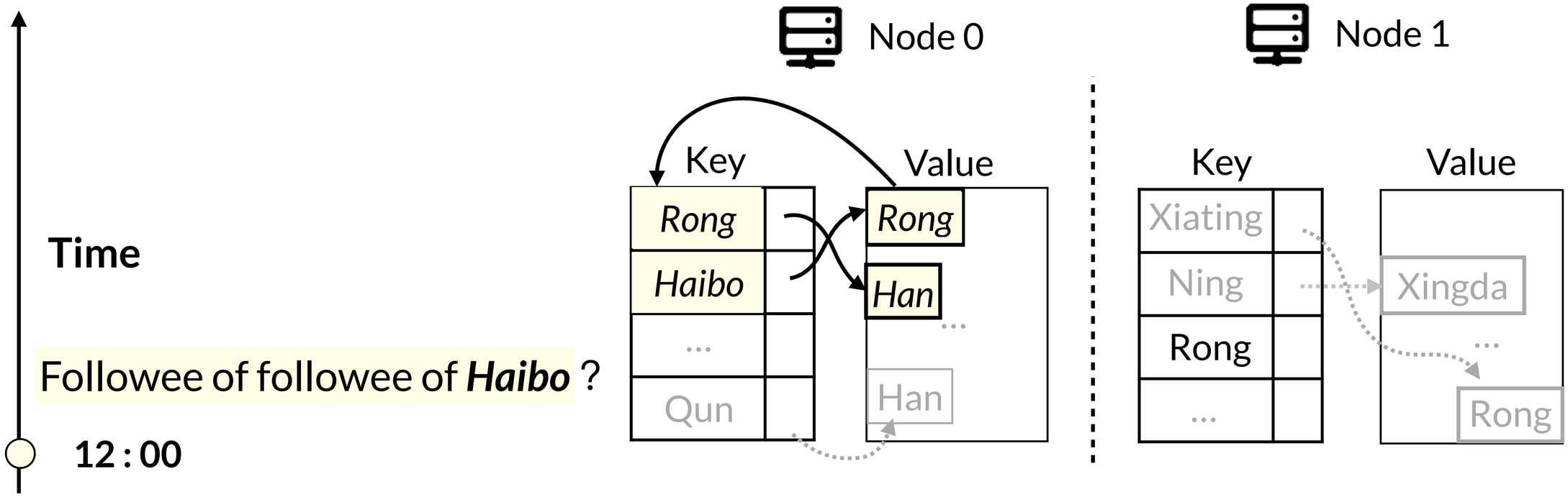
- Workload Change → Migration



How to preserve Locality? Live Migration

Workload is specific **in a period**

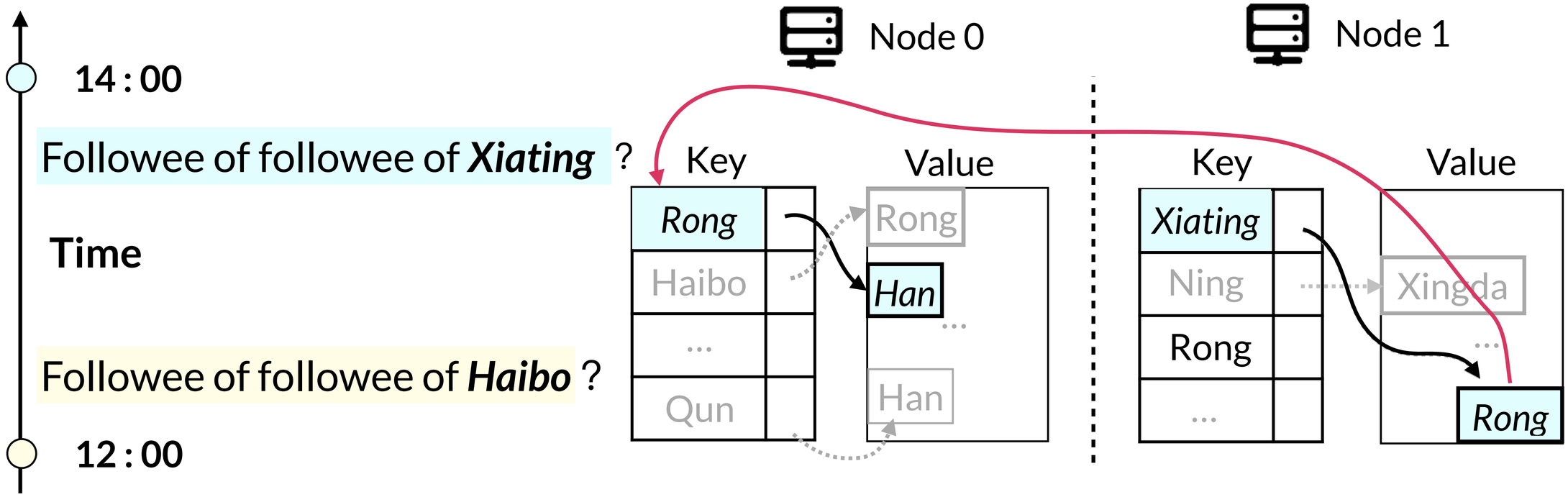
- Workload Change ⟶ Migration



How to preserve Locality? Live Migration

Workload is specific **in a period**

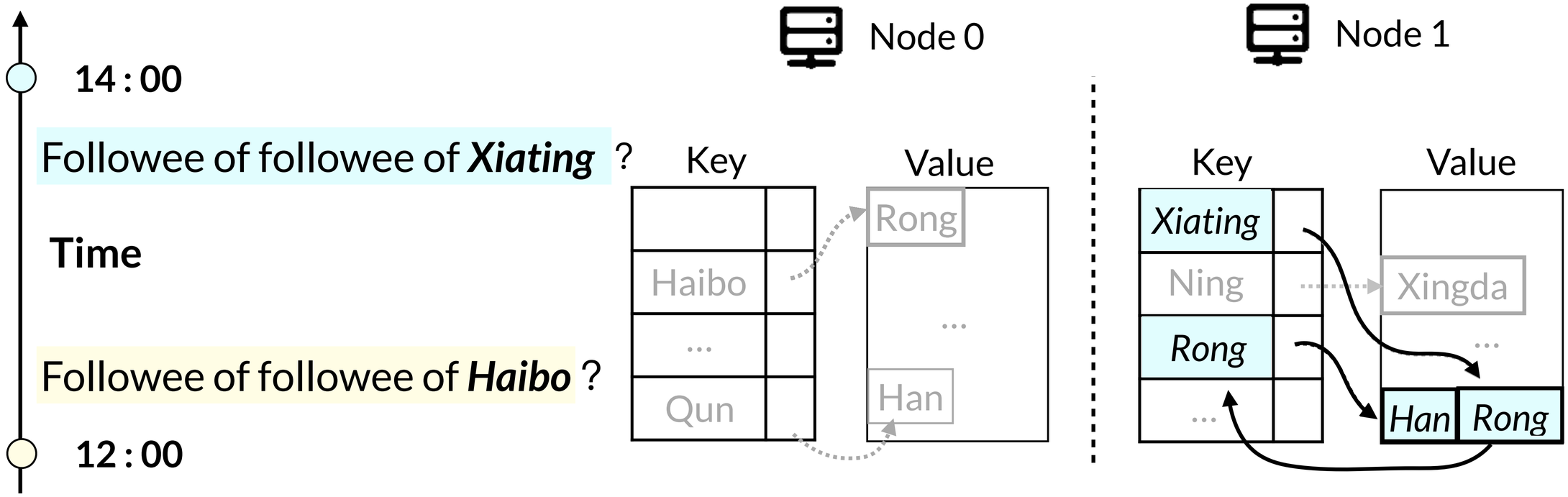
- Workload Change ⟶ Migration



How to preserve Locality? Live Migration

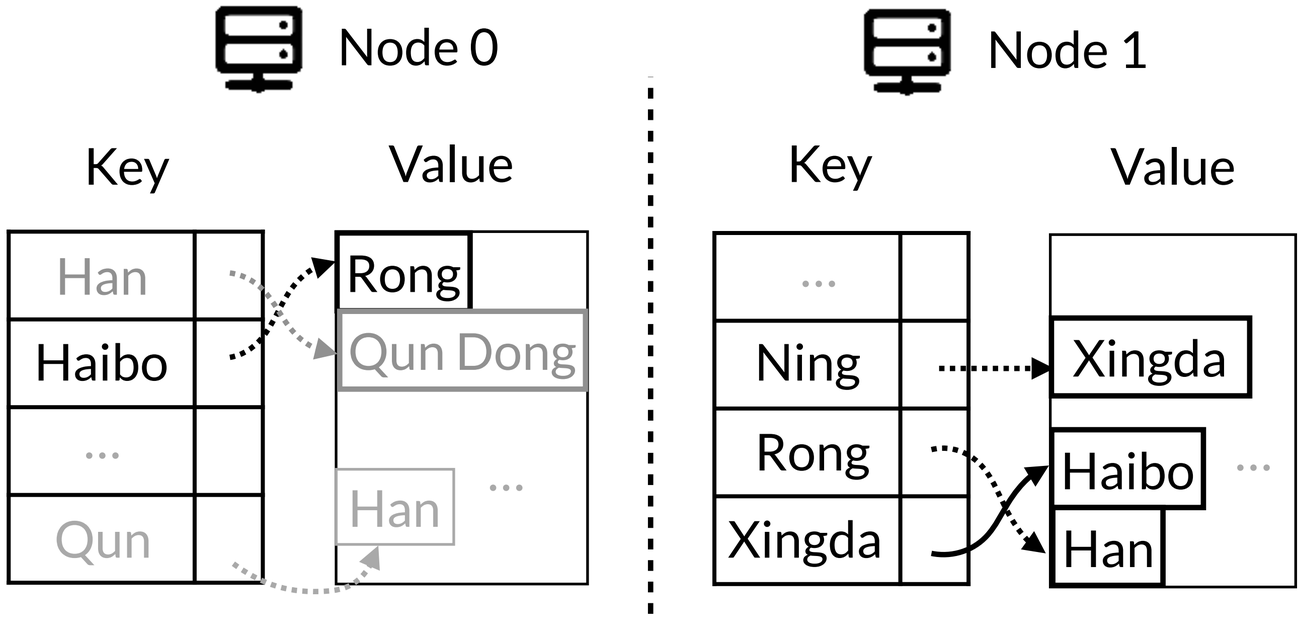
Workload is specific **in a period**

- Workload Change ⟶ Migration



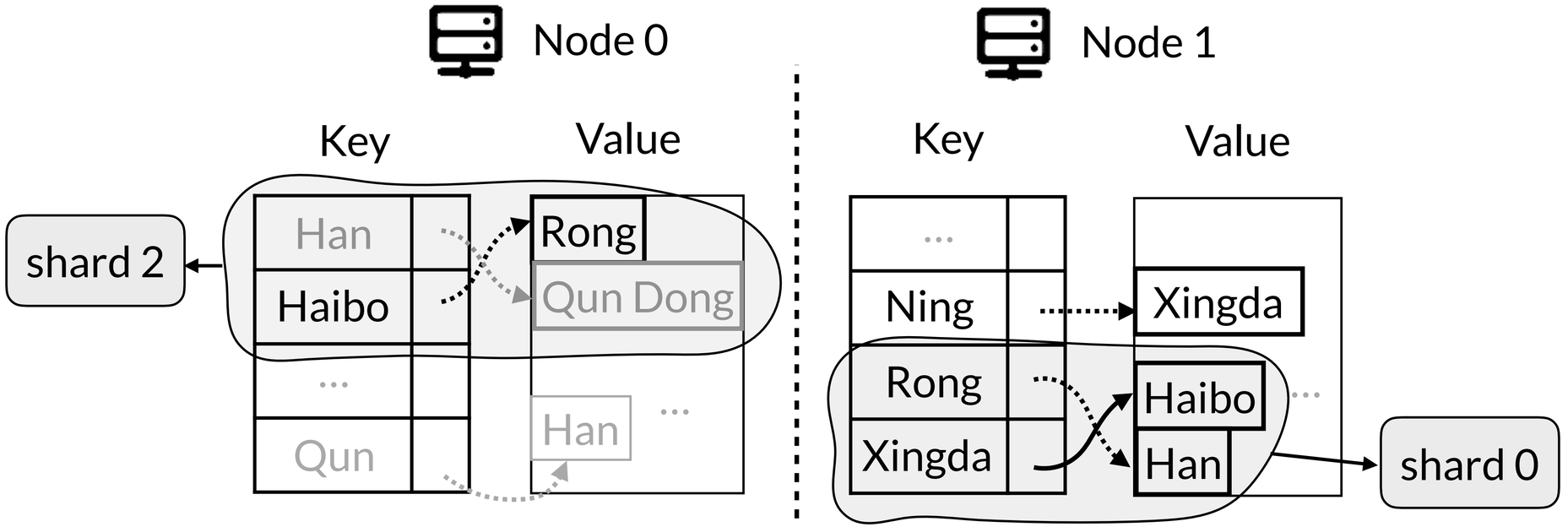
Traditional: Shard-based Migration Not Fits Graph

Graph: **scheme-less**, challenging to partition



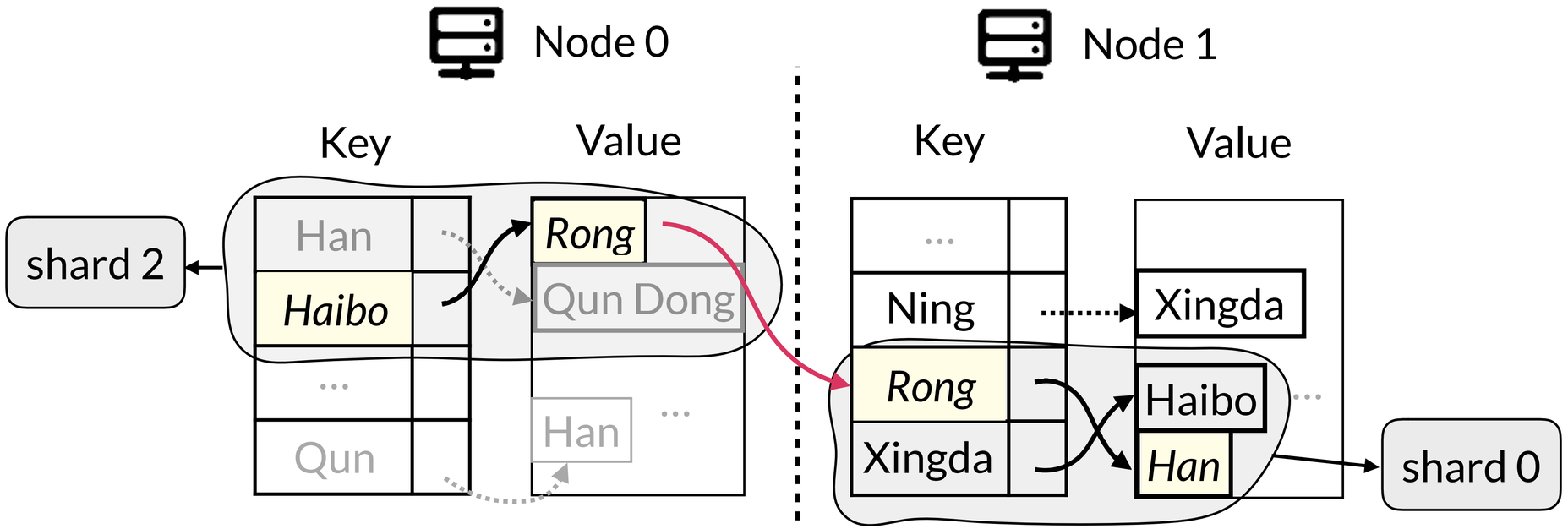
Traditional: Shard-based Migration Not Fits Graph

Graph: **scheme-less**, challenging to partition



Traditional: Shard-based Migration Not Fits Graph

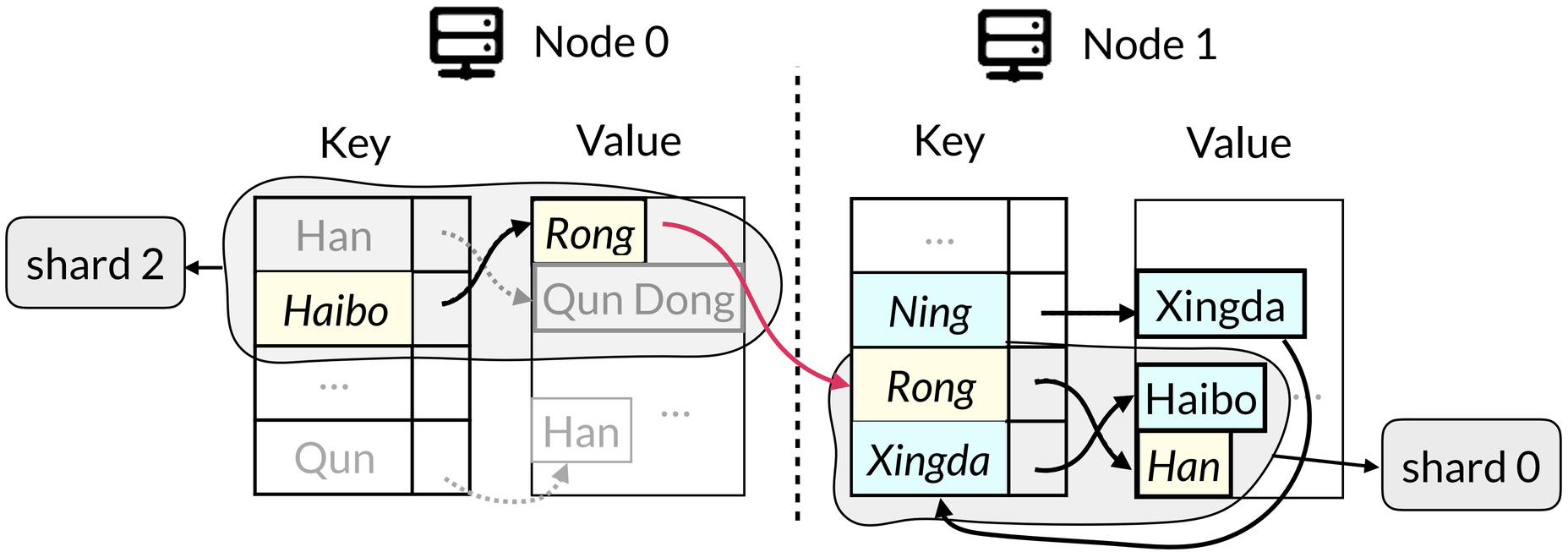
Graph: **scheme-less**, challenging to partition



Followee of followee of ***Haibo*** ?

Traditional: Shard-based Migration Not Fits Graph

Graph: **scheme-less**, challenging to partition

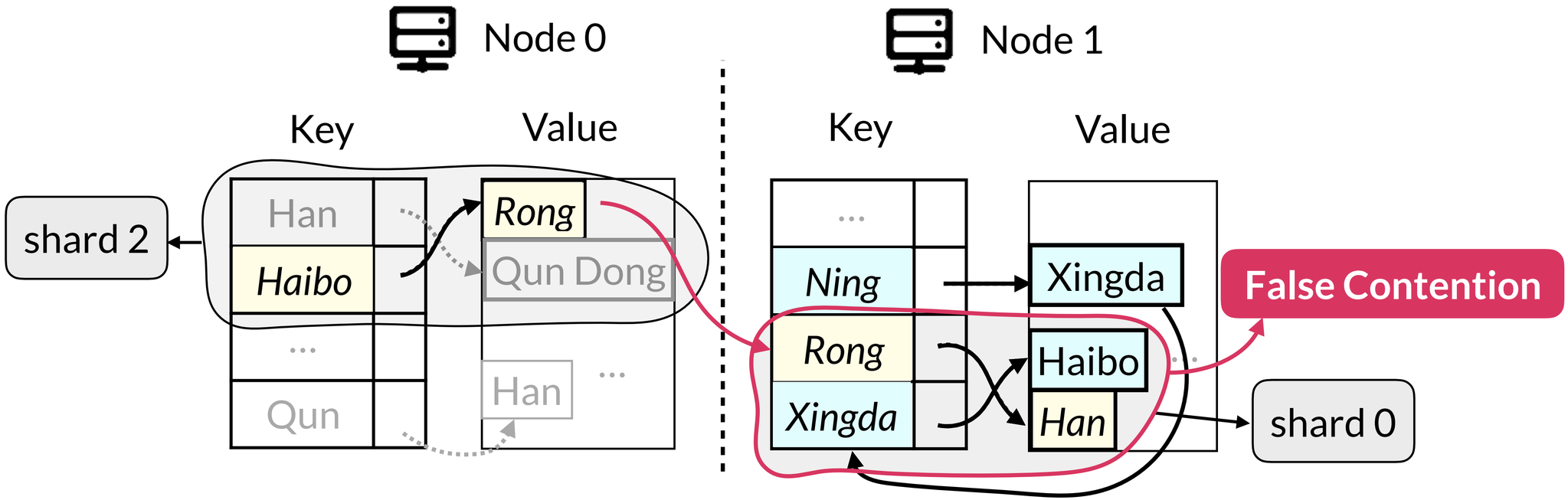


Followee of followee of ***Haibo*** ?

Followee of followee of ***Ning*** ?

Traditional: Shard-based Migration Not Fits Graph

Graph: **scheme-less**, challenging to partition

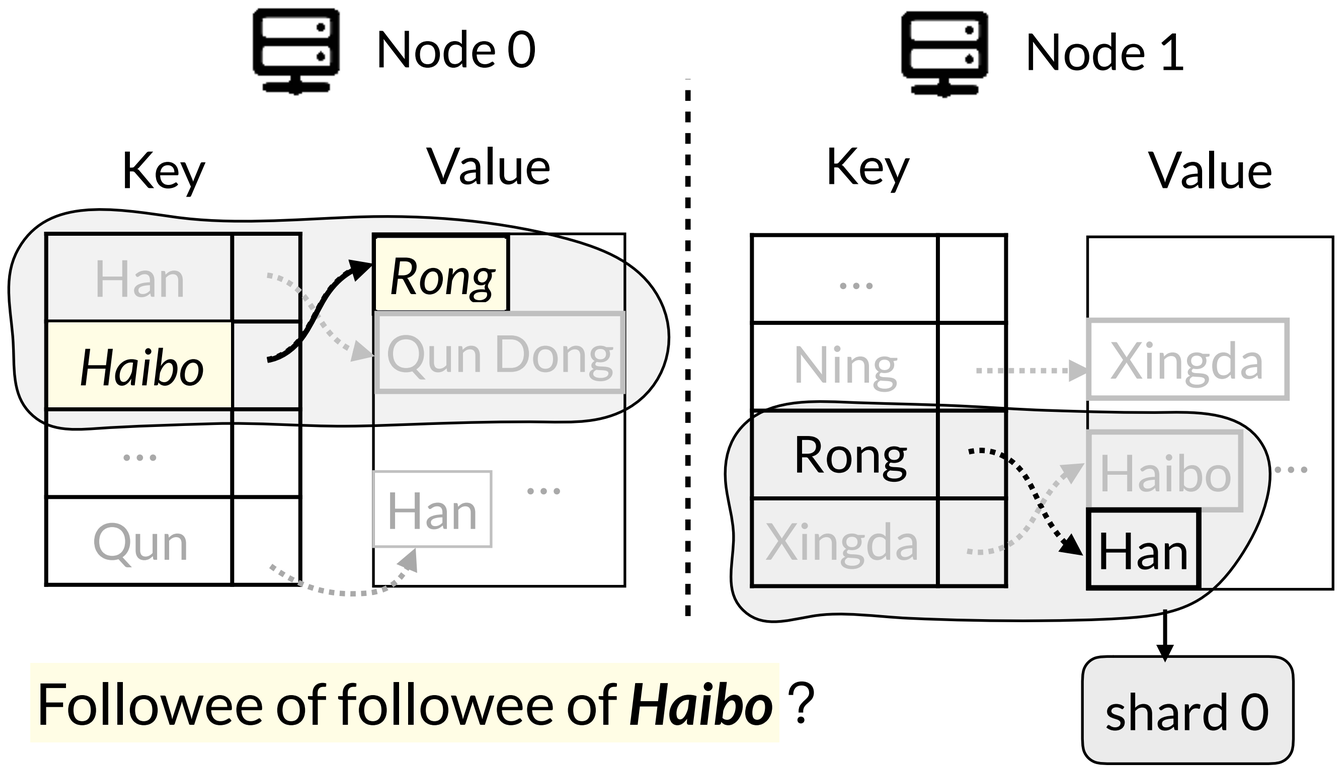


Followee of followee of ***Haibo*** ?

Followee of followee of ***Ning*** ?

Traditional: Shard-based Migration Not Fits Graph

Graph: **scheme-less**, challenging to partition



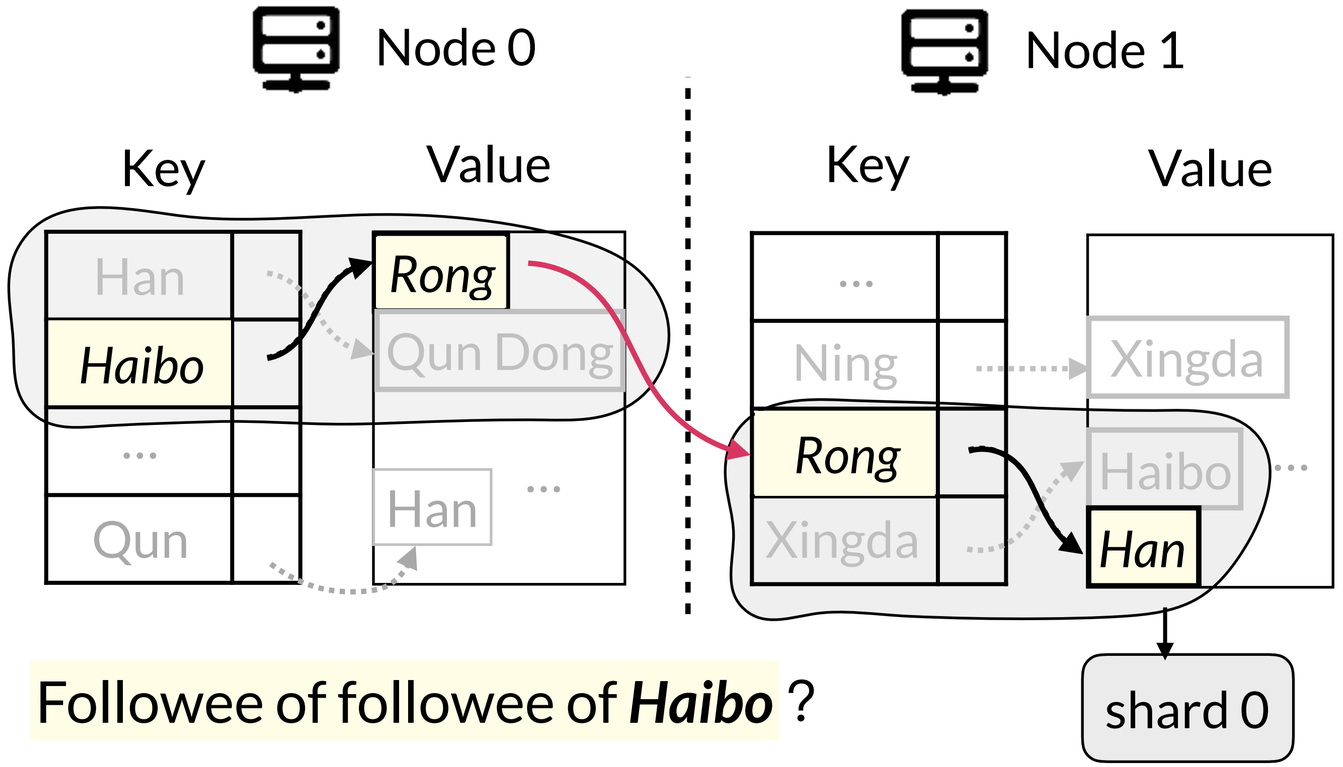
Followee of followee of ***Haibo*** ?

Metadata

Shard	Node
0	1
...	...

Traditional: Shard-based Migration Not Fits Graph

Graph: **scheme-less**, challenging to partition



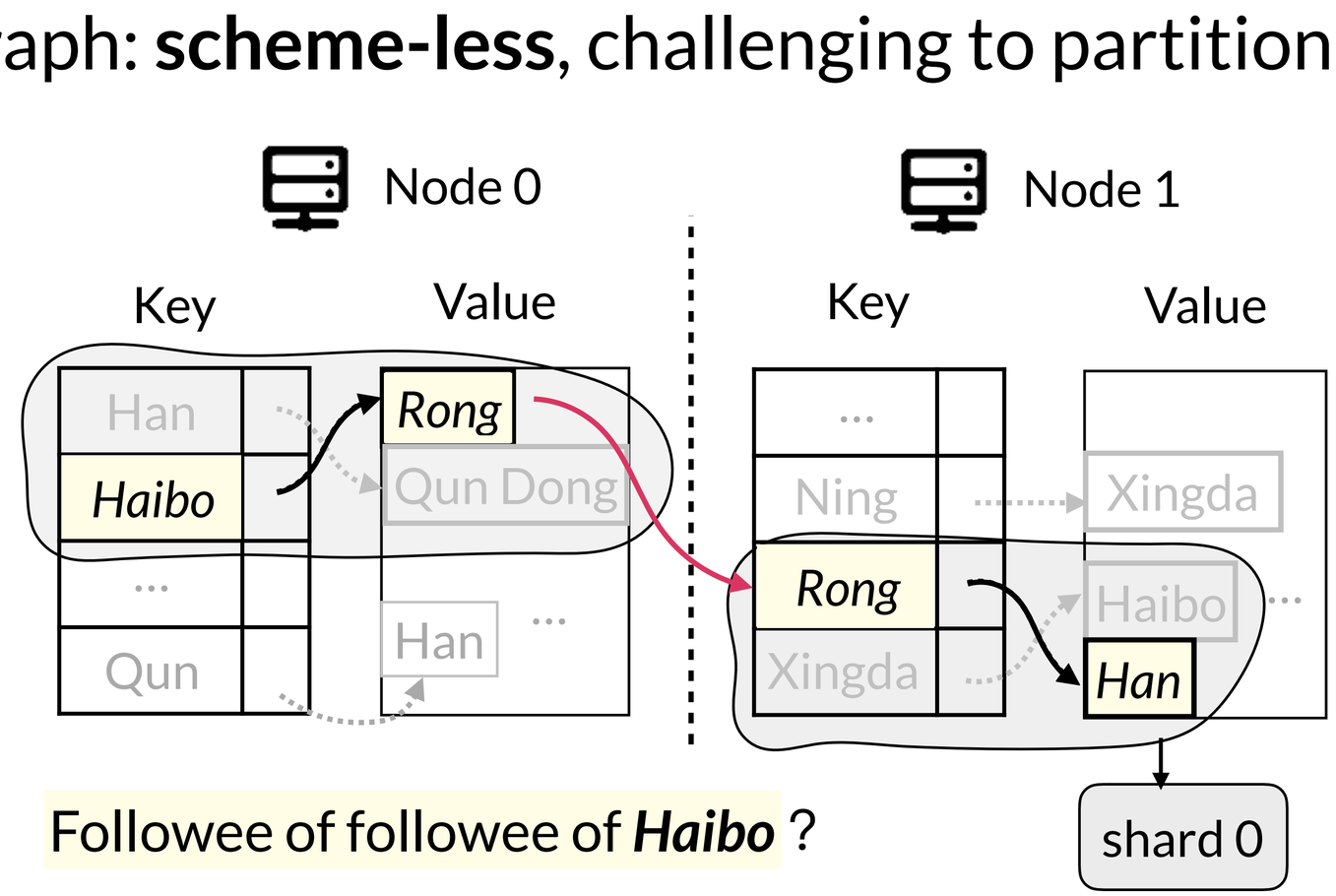
Followee of followee of ***Haibo*** ?

Metadata

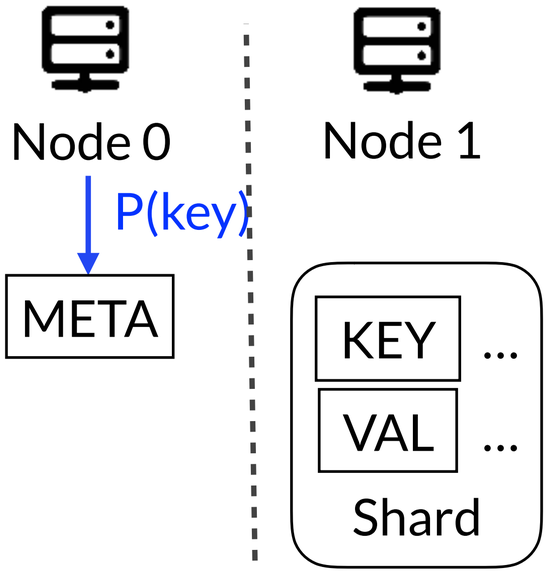
Shard	Node
0	1
...	...

Traditional: Shard-based Migration Not Fits Graph

Graph: **scheme-less**, challenging to partition



Followee of followee of ***Haibo*** ?

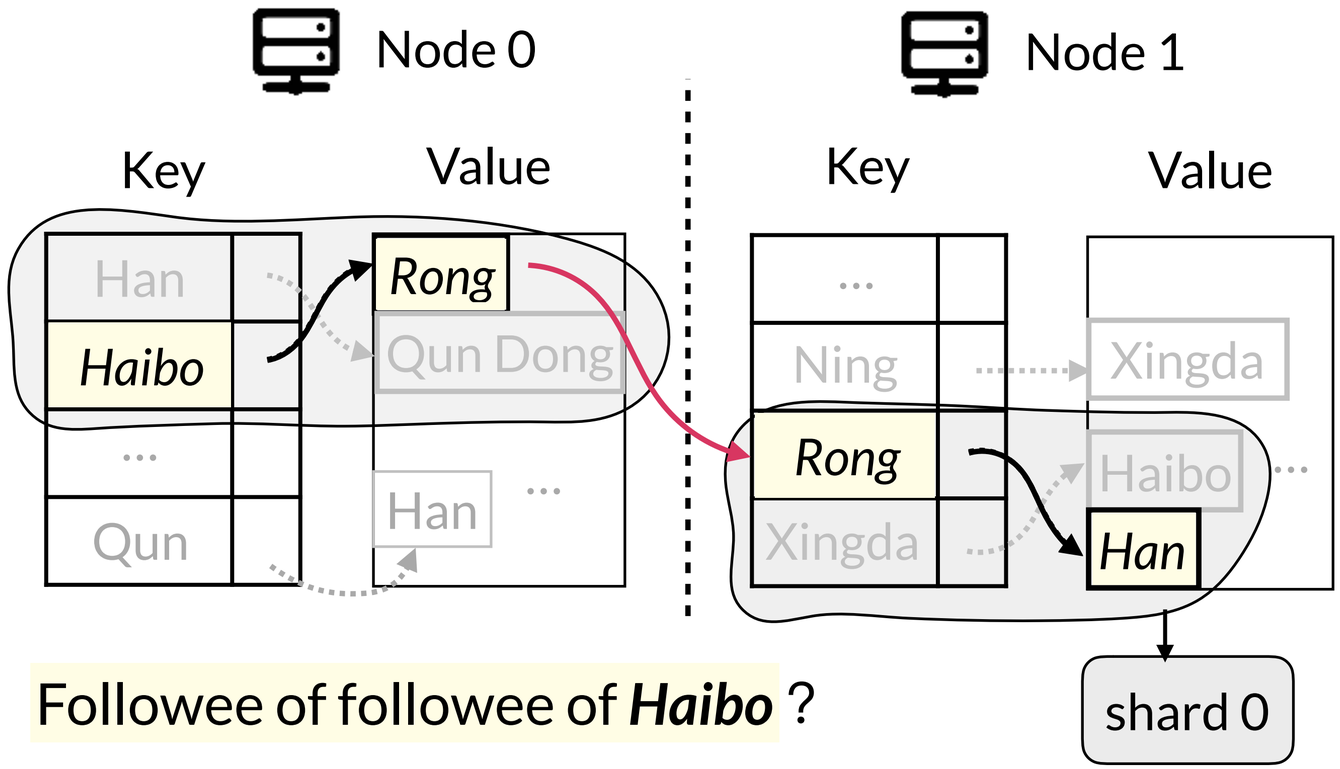


Metadata

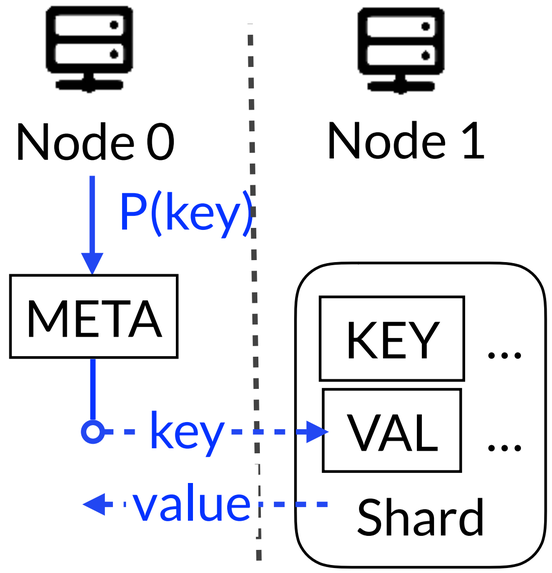
Shard	Node
0	1
...	...

Traditional: Shard-based Migration Not Fits Graph

Graph: **scheme-less**, challenging to partition



Followee of followee of ***Haibo*** ?

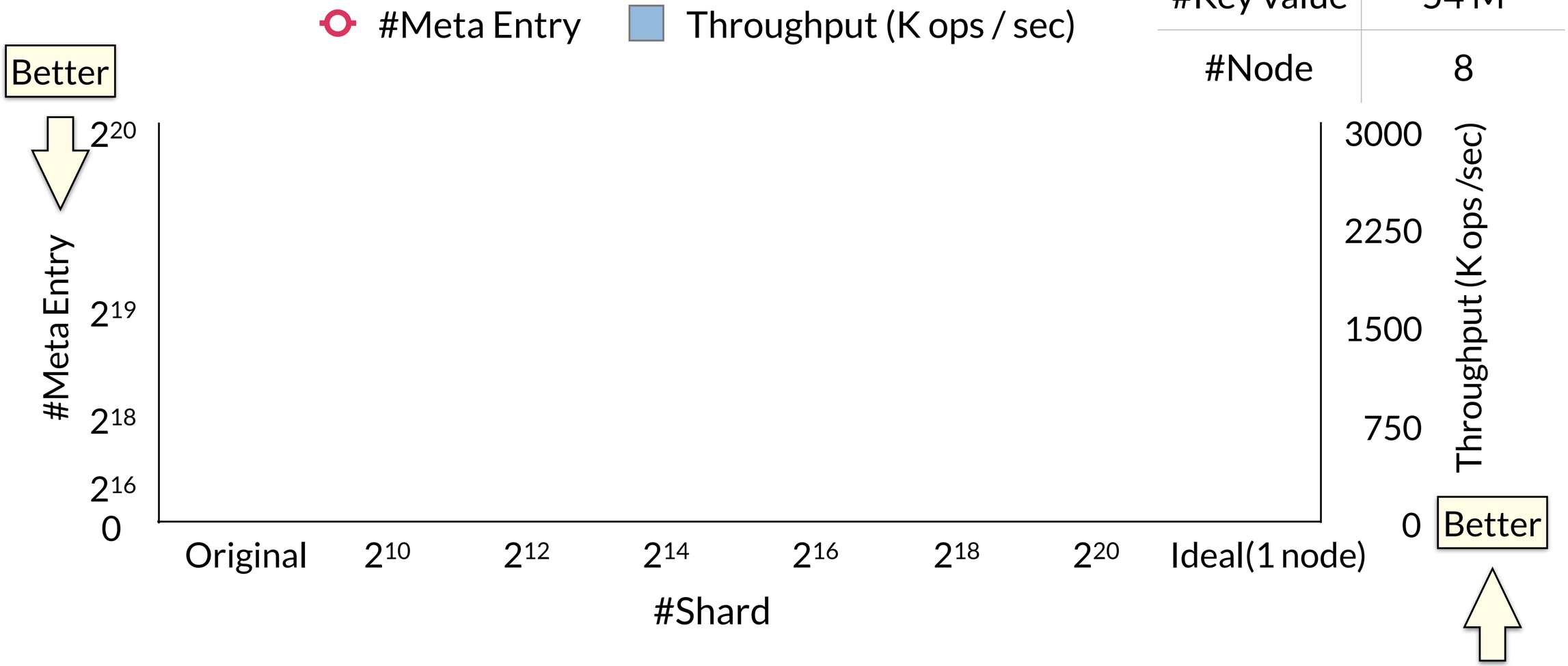


Metadata

Shard	Node
0	1
...	...

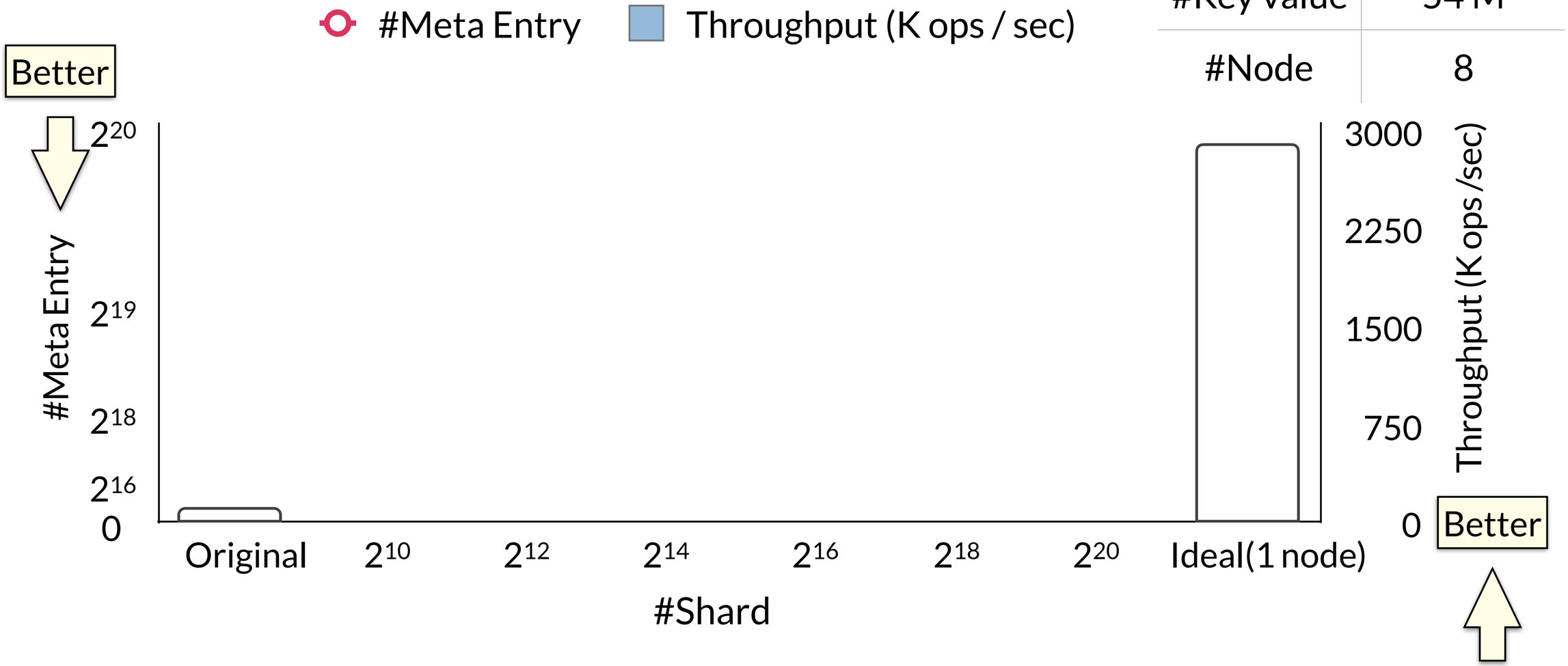
Shard-based Migration: Dilemma

Graph	RMAT_26
#Key value	54 M
#Node	8



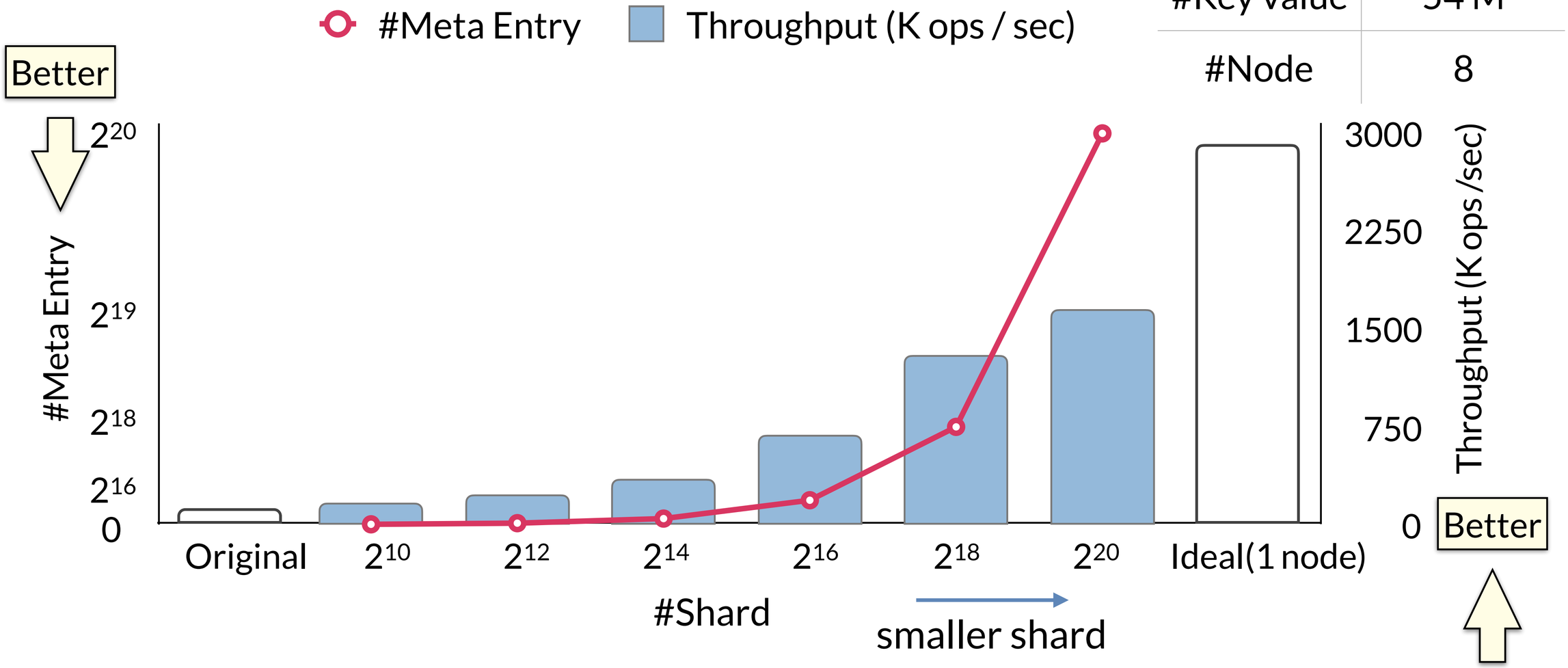
Shard-based Migration: Dilemma

Graph	RMAT_26
#Key value	54 M
#Node	8



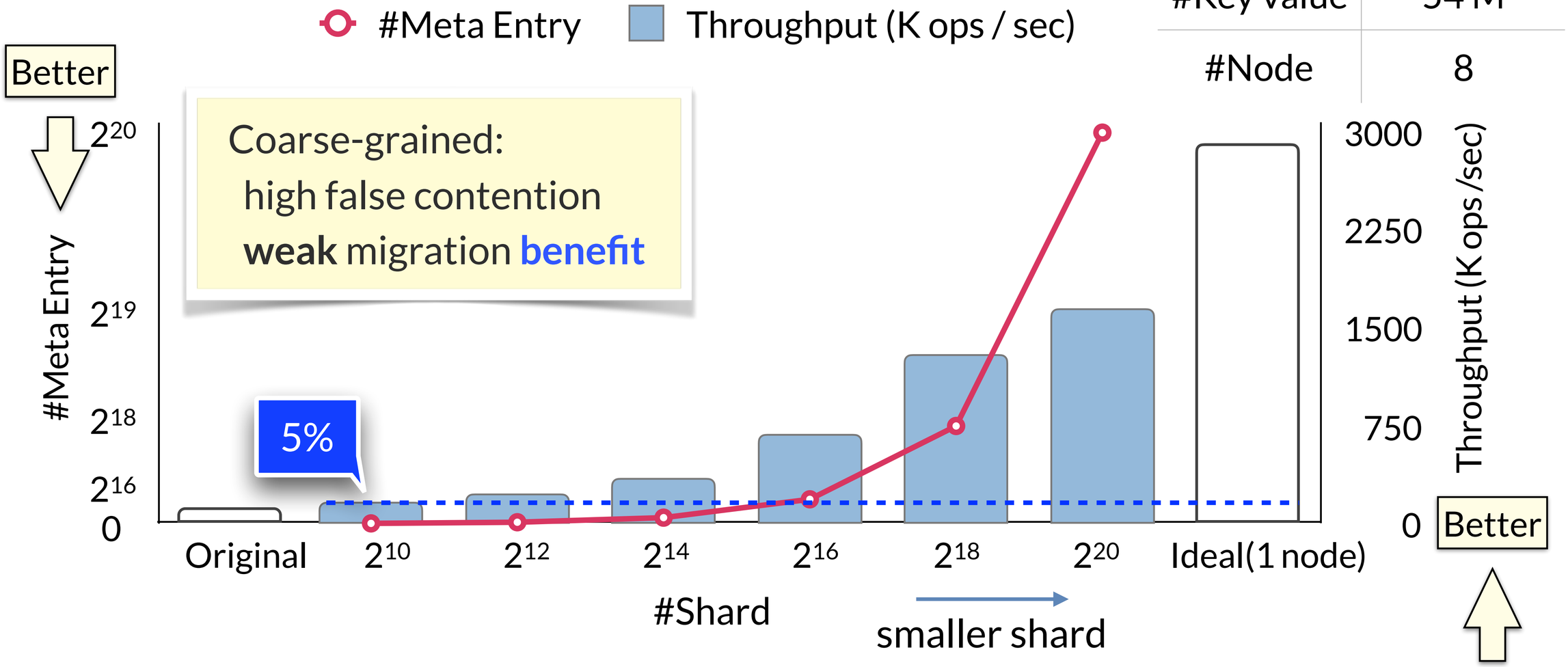
Shard-based Migration: Dilemma

Graph	RMAT_26
#Key value	54 M
#Node	8



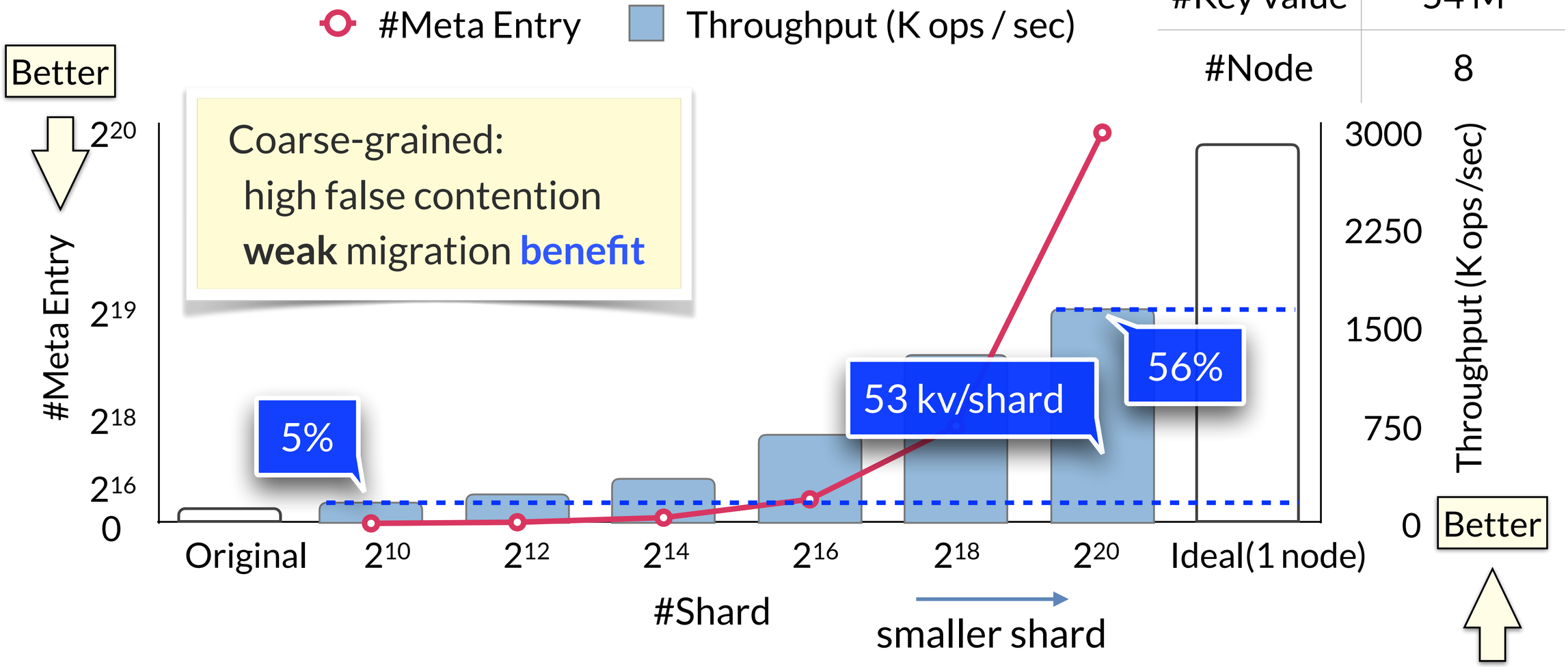
Shard-based Migration: Dilemma

Graph	RMAT_26
#Key value	54 M
#Node	8



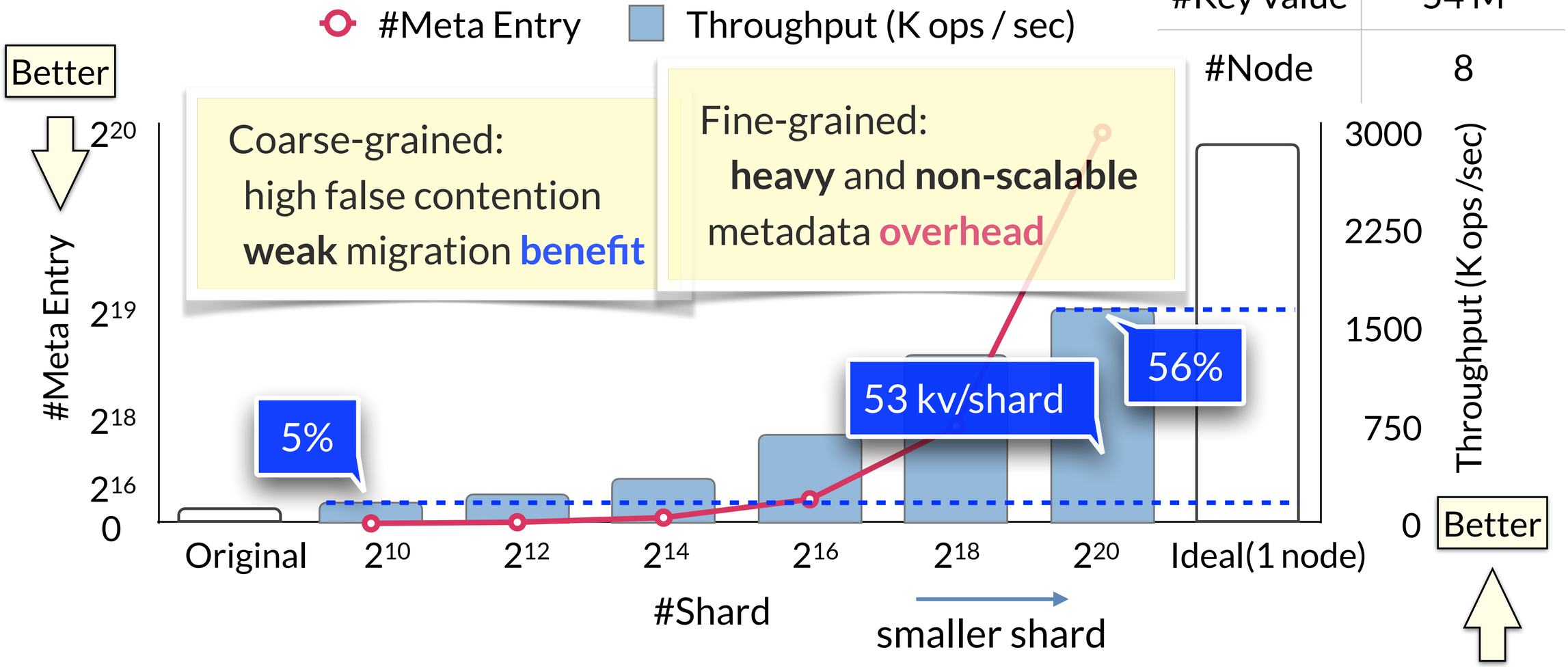
Shard-based Migration: Dilemma

Graph	RMAT_26
#Key value	54 M
#Node	8



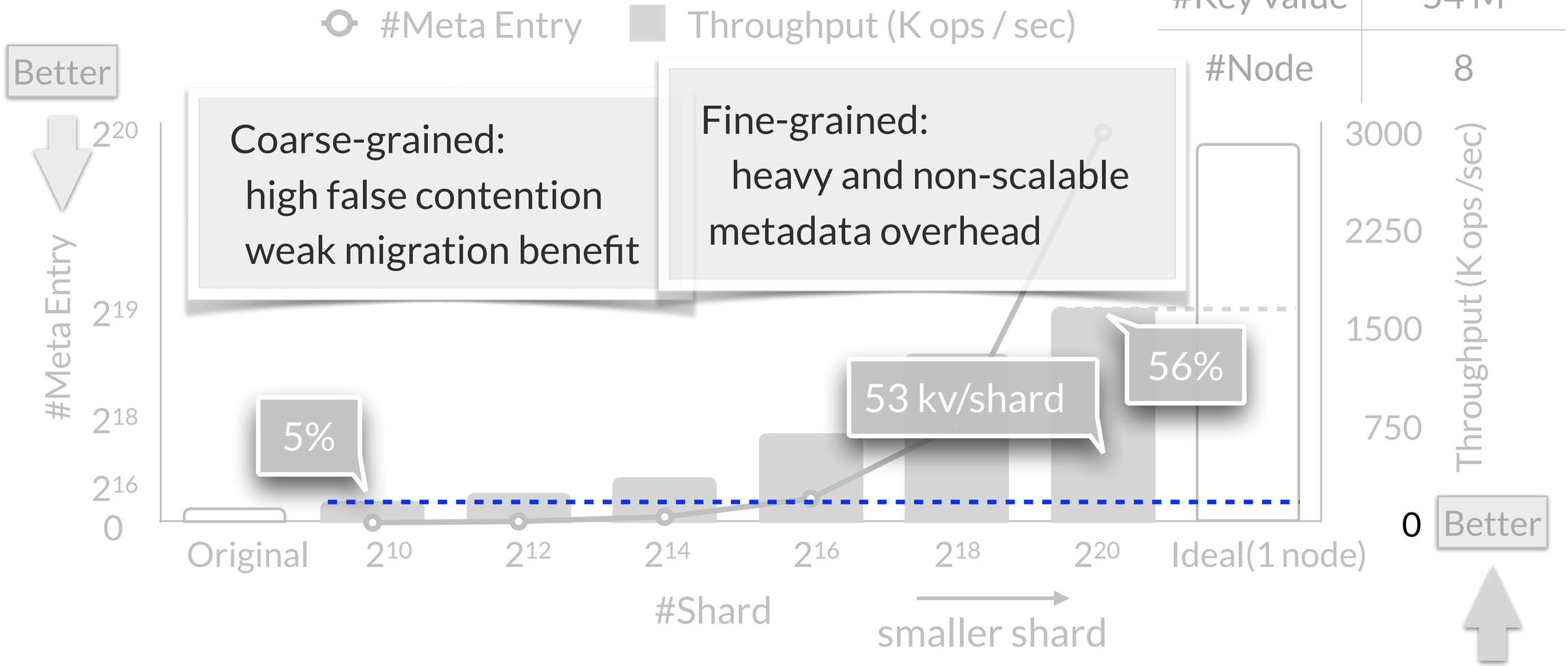
Shard-based Migration: Dilemma

Graph	RMAT_26
#Key value	54 M
#Node	8



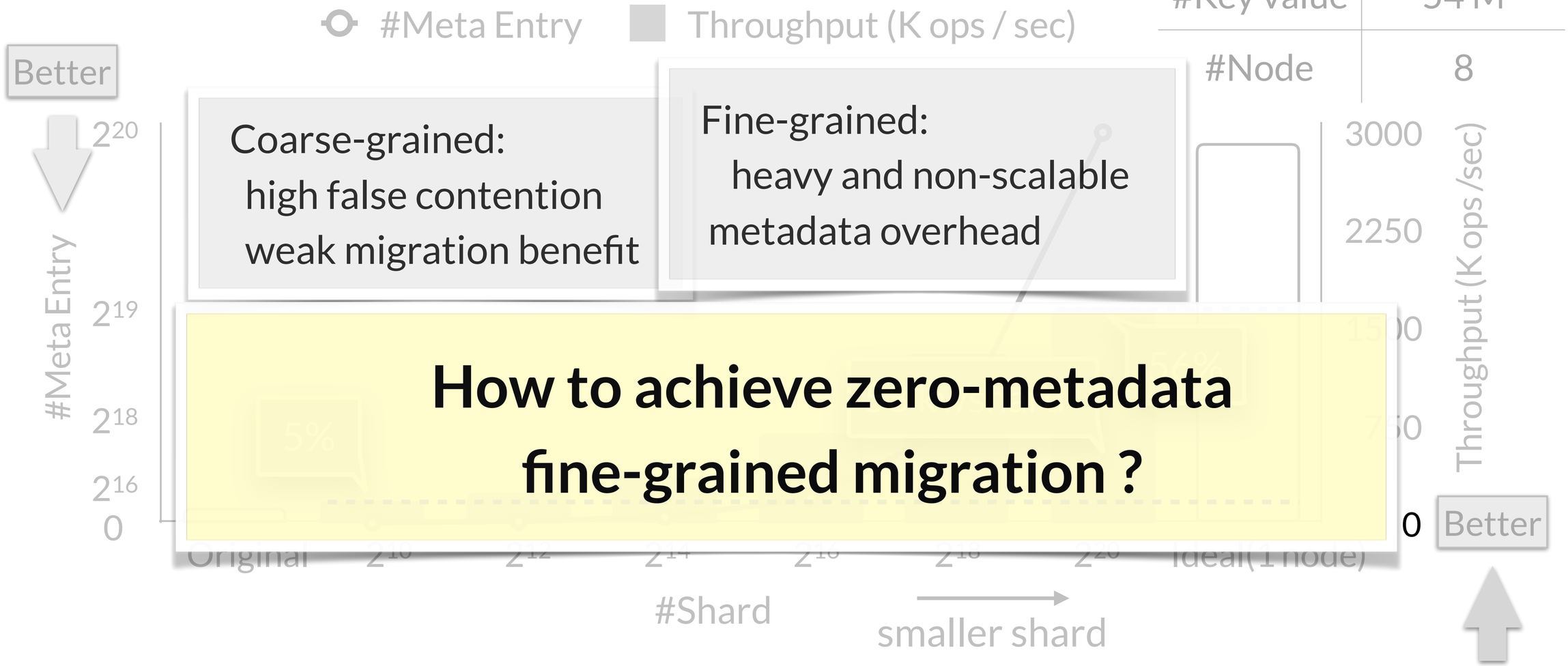
Shard-based Migration: Dilemma

Graph	RMAT_26
#Key value	54 M
#Node	8



Shard-based Migration: Dilemma

Graph	RMAT_26
#Key value	54 M
#Node	8



Opportunity: Remote Direct Memory Access (RDMA)

Provide cross-node accesses:

- High speed, low latency, kernel bypassing

One-sided RDMA primitive (Read, Write, CAS[1])

- Direct access **bypassing CPU**
- **Split accesses** to keys & values

[1] Atomic **c**ompare **a**nd **s**wap

Opportunity: Remote Direct Memory Access (RDMA)

Provide cross-node accesses:

- High speed, low latency, kernel bypassing

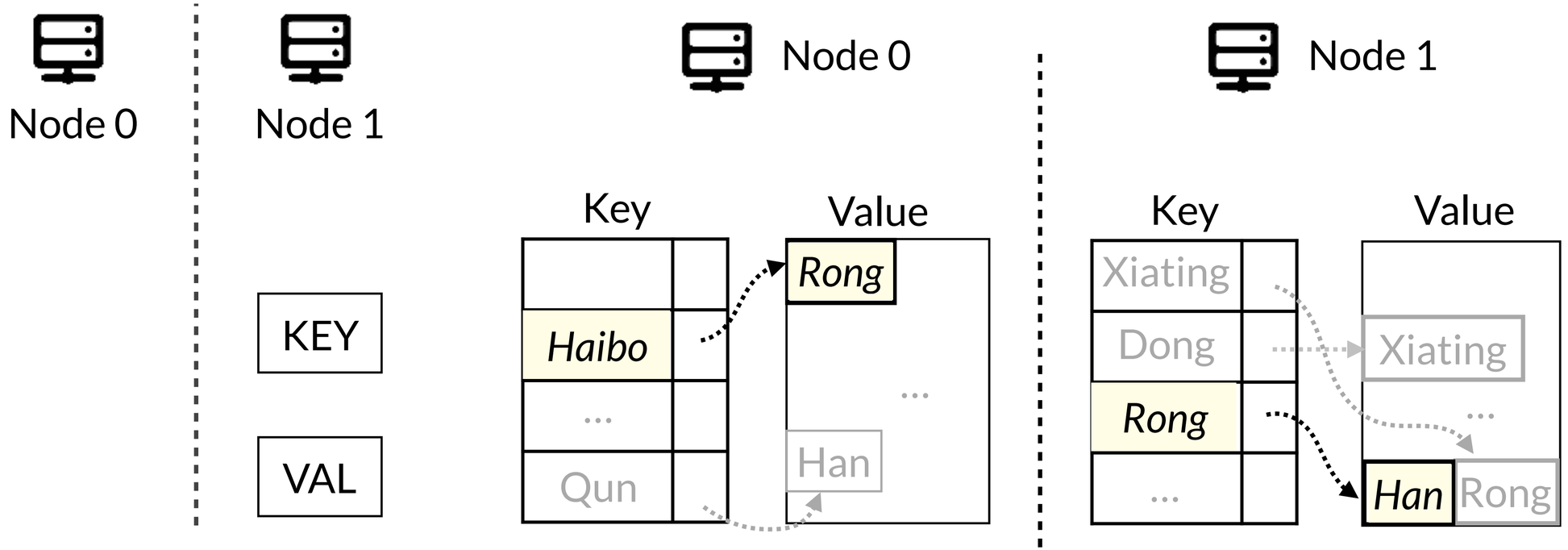
One-sided RDMA primitive (Read, Write, CAS[1])

- Direct access **bypassing CPU**
- **Split accesses** to keys & values

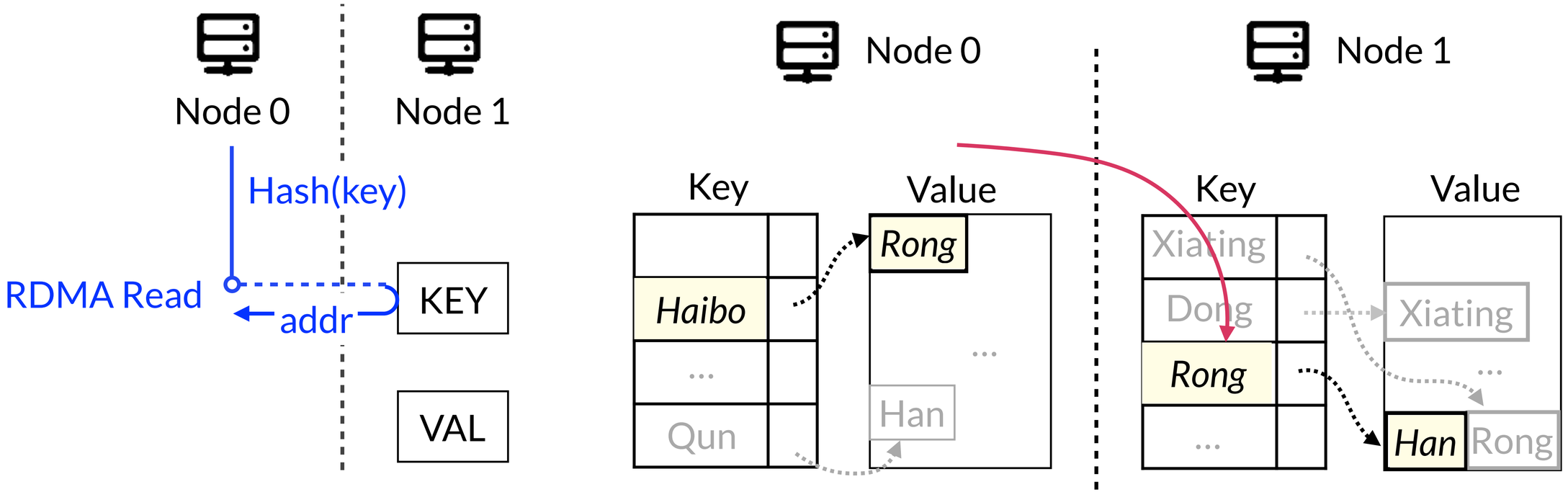
Split Access → **Opportunity** → Zero-metadata Fine-grained Migration

[1] Atomic **c**ompare **a**nd **s**wap

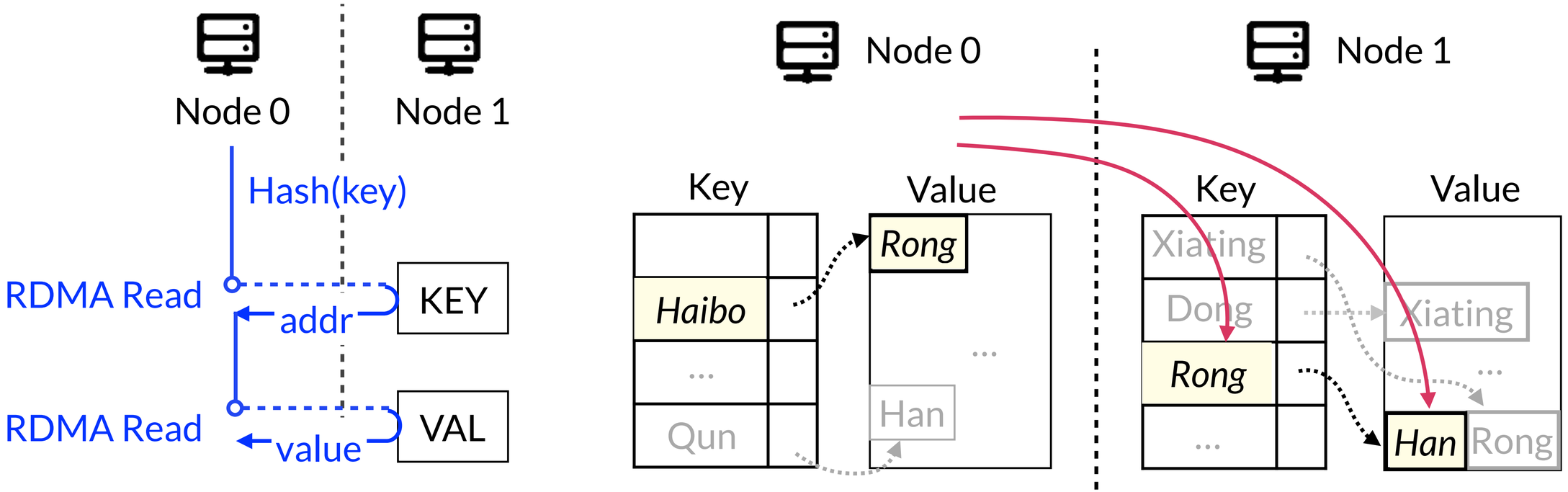
RDMA Splits Accesses to Keys & Values



RDMA Splits Accesses to Keys & Values

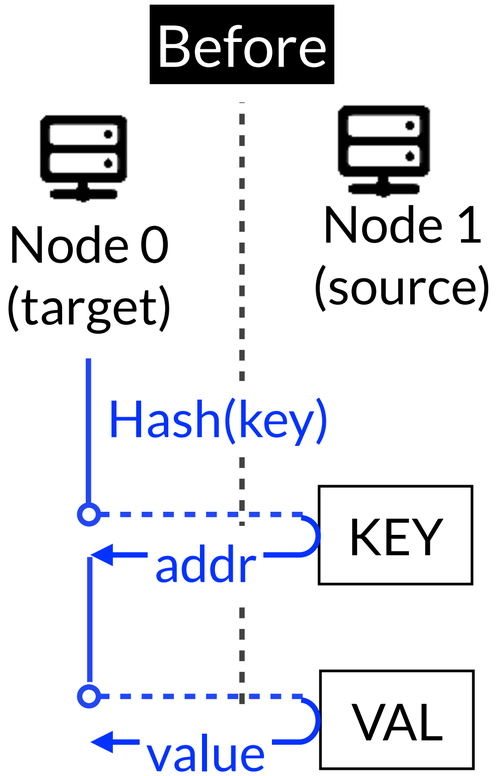


RDMA Splits Accesses to Keys & Values



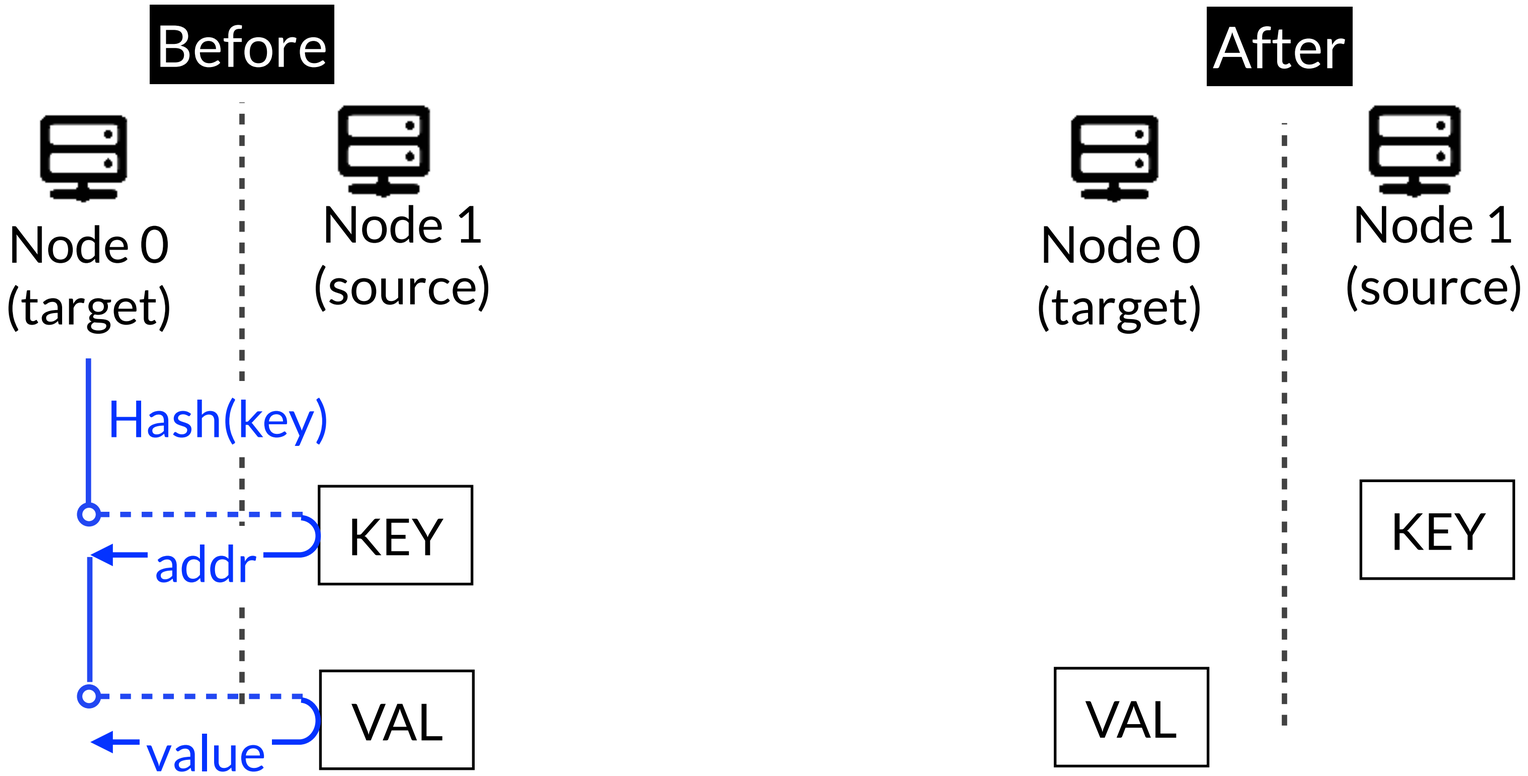
Split Access Makes Fine-grained Migration Easier

Easy, efficient to **split keys & values** in physical



Split Access Makes Fine-grained Migration Easier

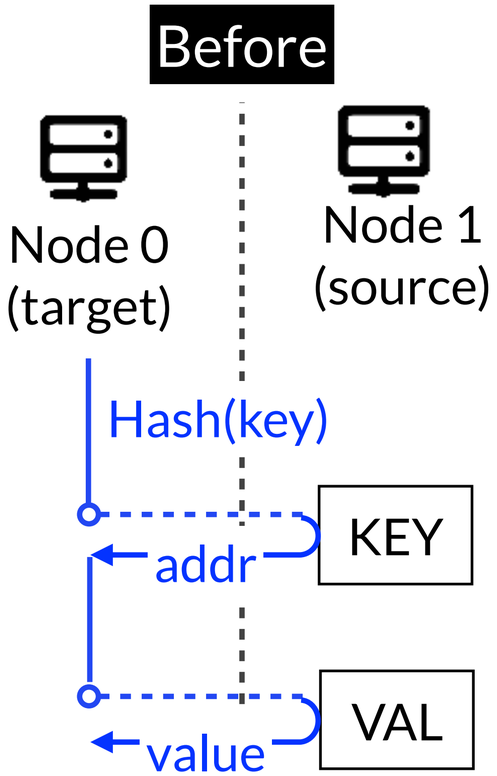
Easy, efficient to **split keys & values** in physical

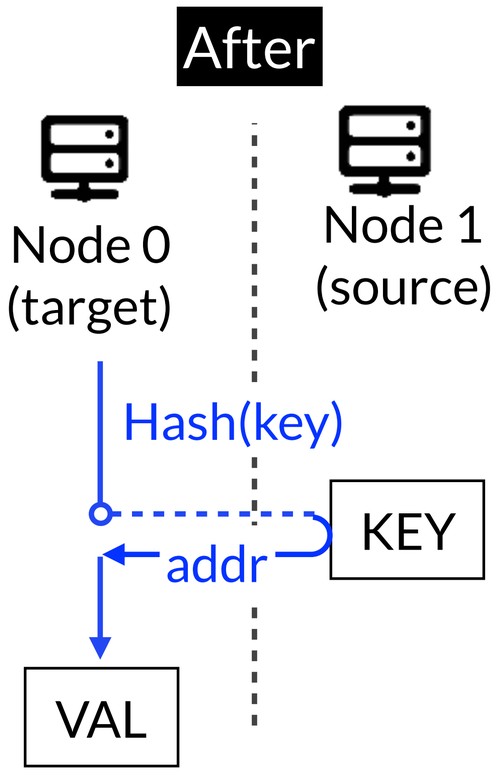


migrates **values**, not migrate **keys**

Split Access Makes Fine-grained Migration Easier

Easy, efficient to **split keys & values** in physical

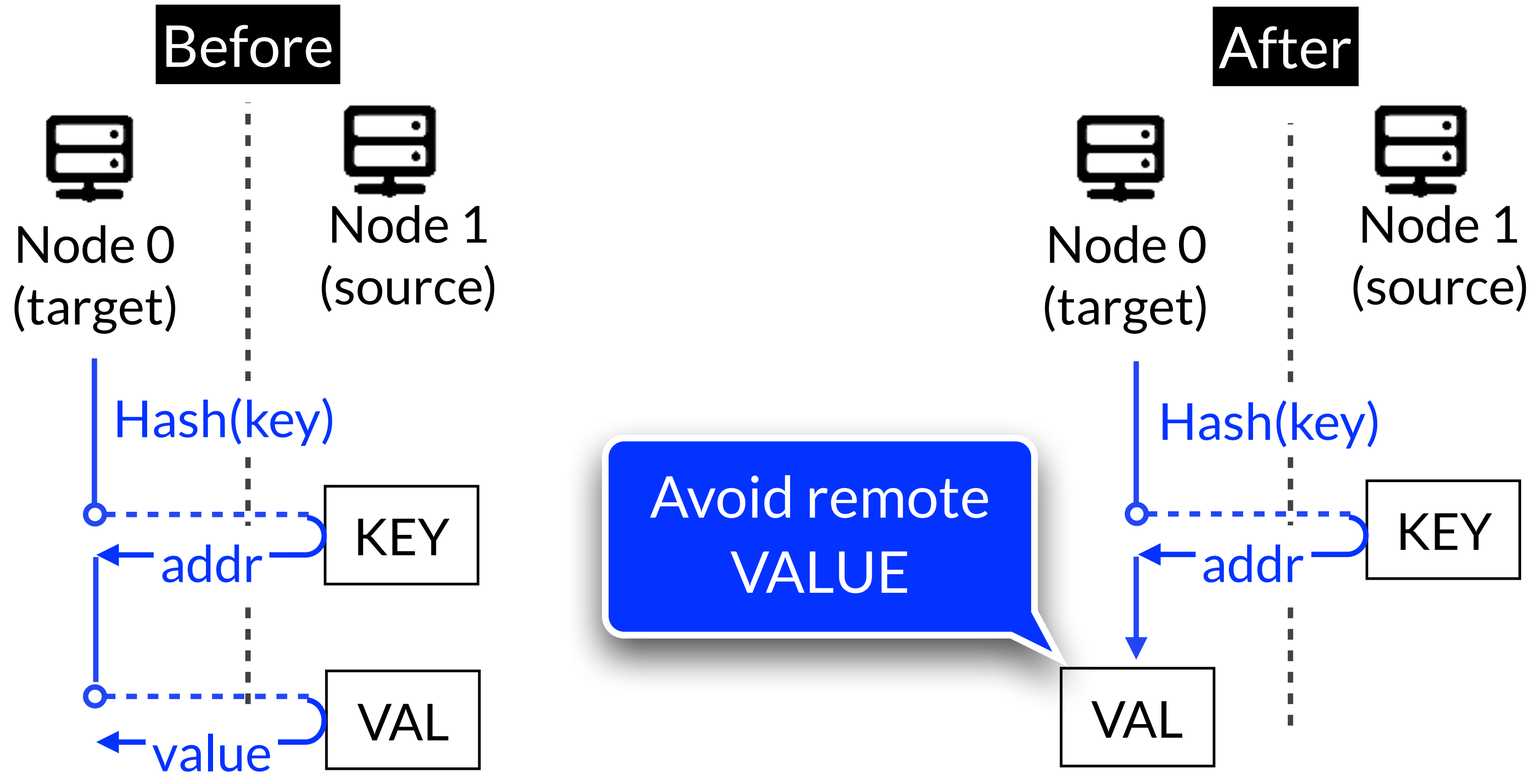




migrates **values**, not migrate **keys**

Split Access Makes Fine-grained Migration Easier

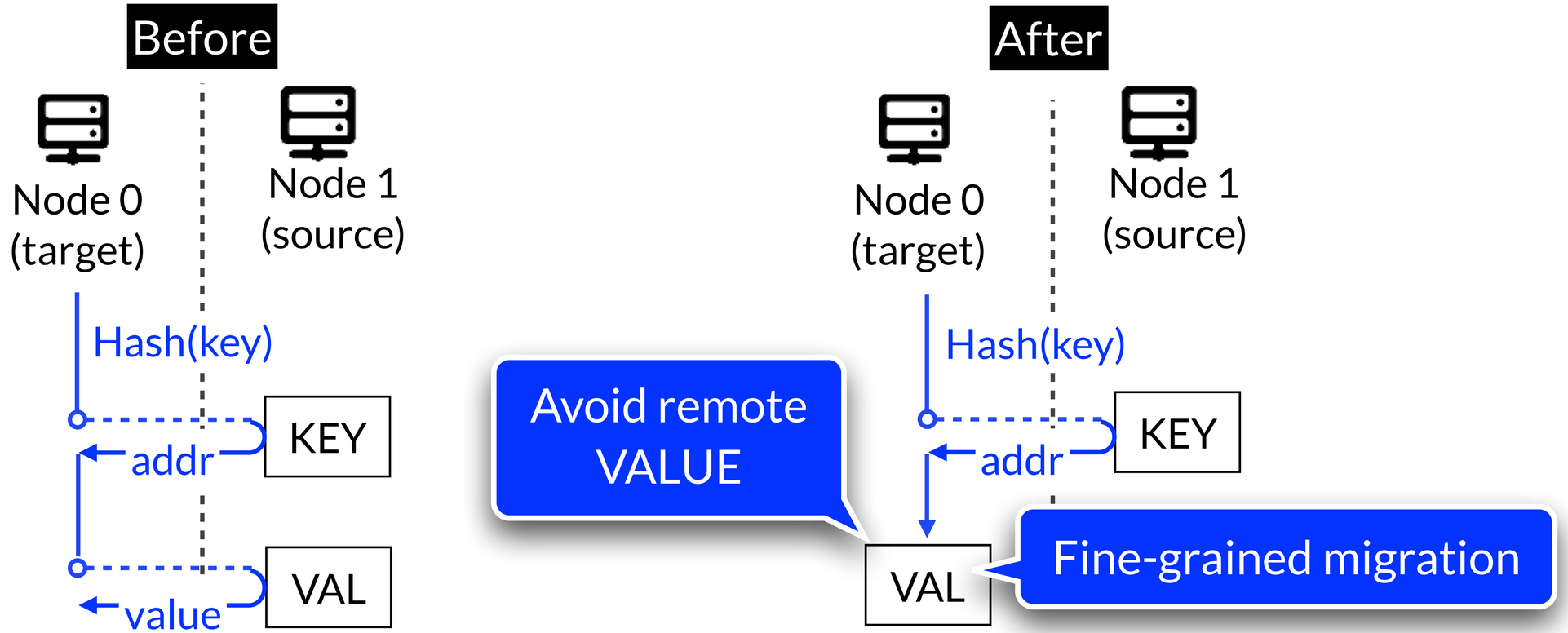
Easy, efficient to **split keys & values** in physical



migrates **values**, not migrate **keys**

Split Access Makes Fine-grained Migration Easier

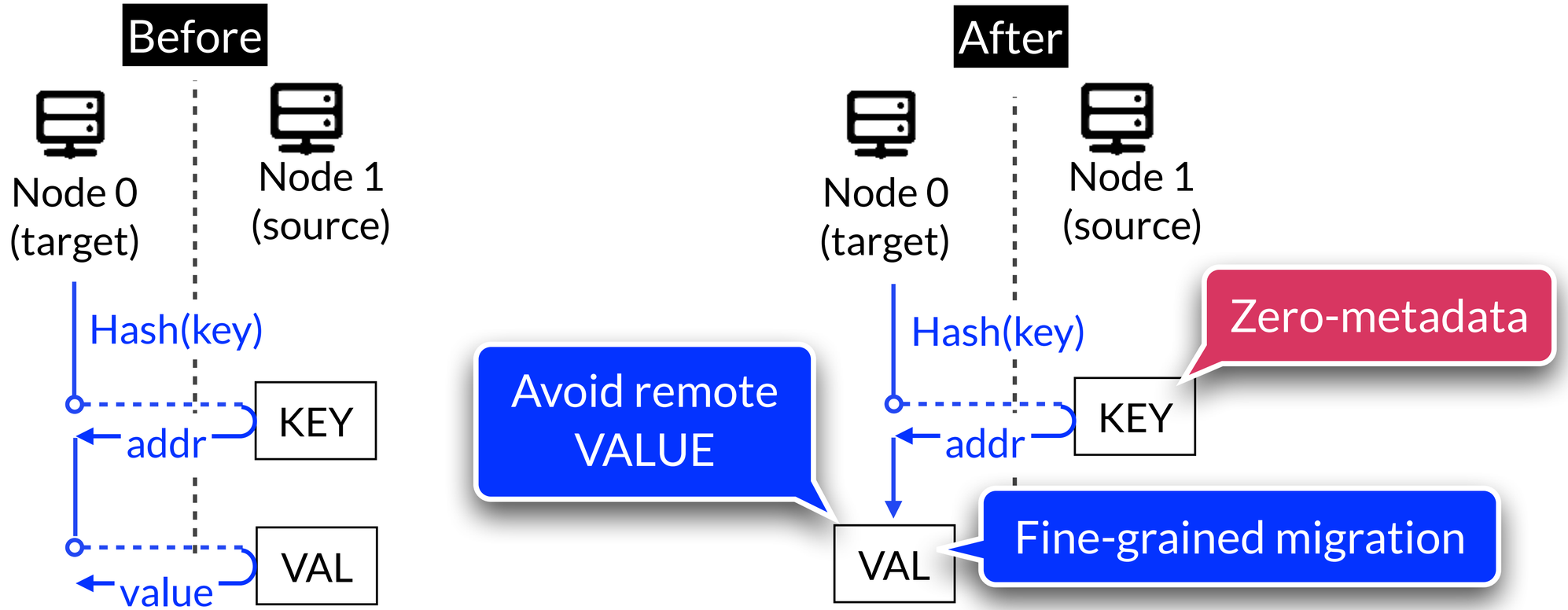
Easy, efficient to **split keys & values** in physical



migrates **values**, not migrate **keys**

Split Access Makes Fine-grained Migration Easier

Easy, efficient to **split keys & values** in physical



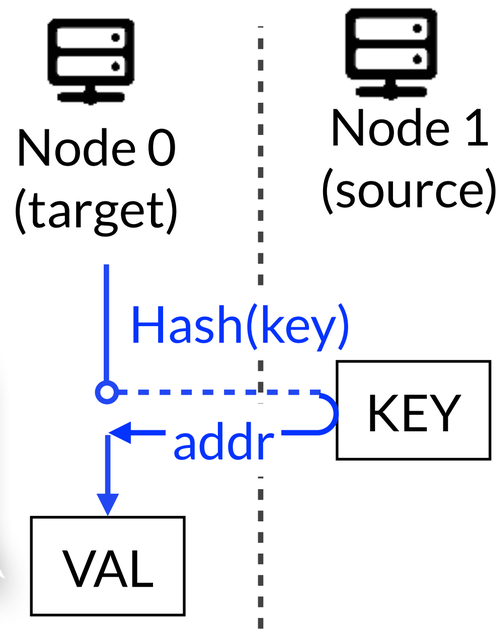
migrates **values**, not migrate **keys**

Pragh: split live migration

Split live migration: migrates **values**, not migrate **keys**

Fine-grained migration

Zero-metadata



Avoid remote VALUE

Micro-benchmark: **19X**

Wukong(OSDI'16) port: **2.53X**

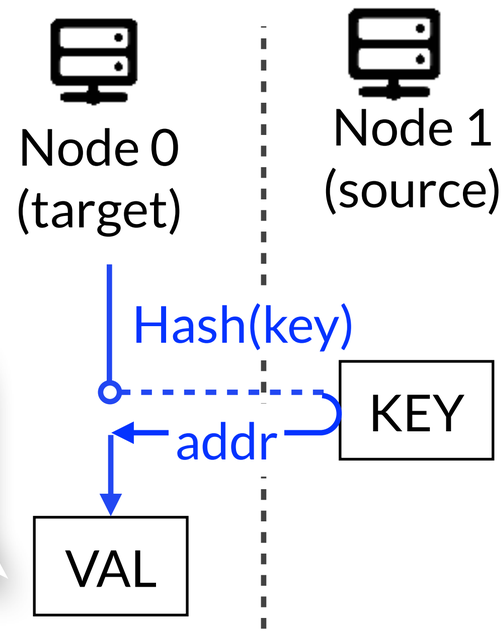
Pragh: split live migration

Split live migration: migrates **values**, not migrate **keys**

- Source bypassing migration
- Fully-localized migration
- Lightweight fine-grained monitor
- Supporting evolving graph (dynamic graph update)

Fine-grained migration

Zero-metadata

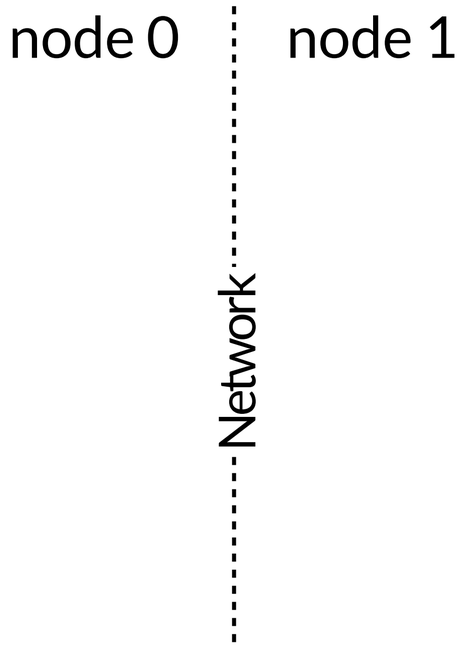


Avoid remote VALUE

Micro-benchmark: **19X**

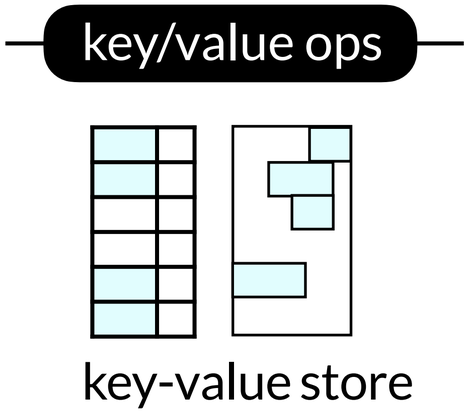
Wukong(OSDI'16) port: **2.53X**

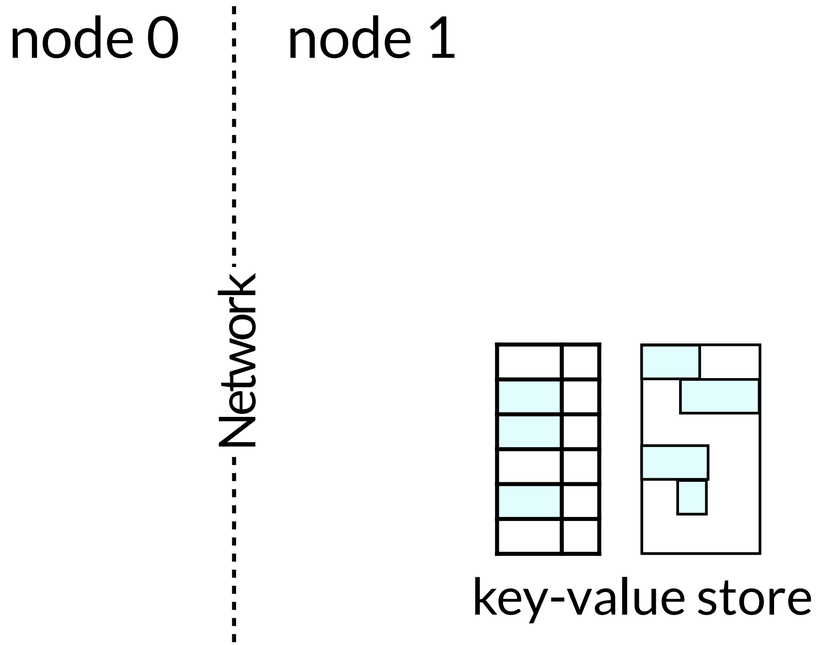
System Architecture



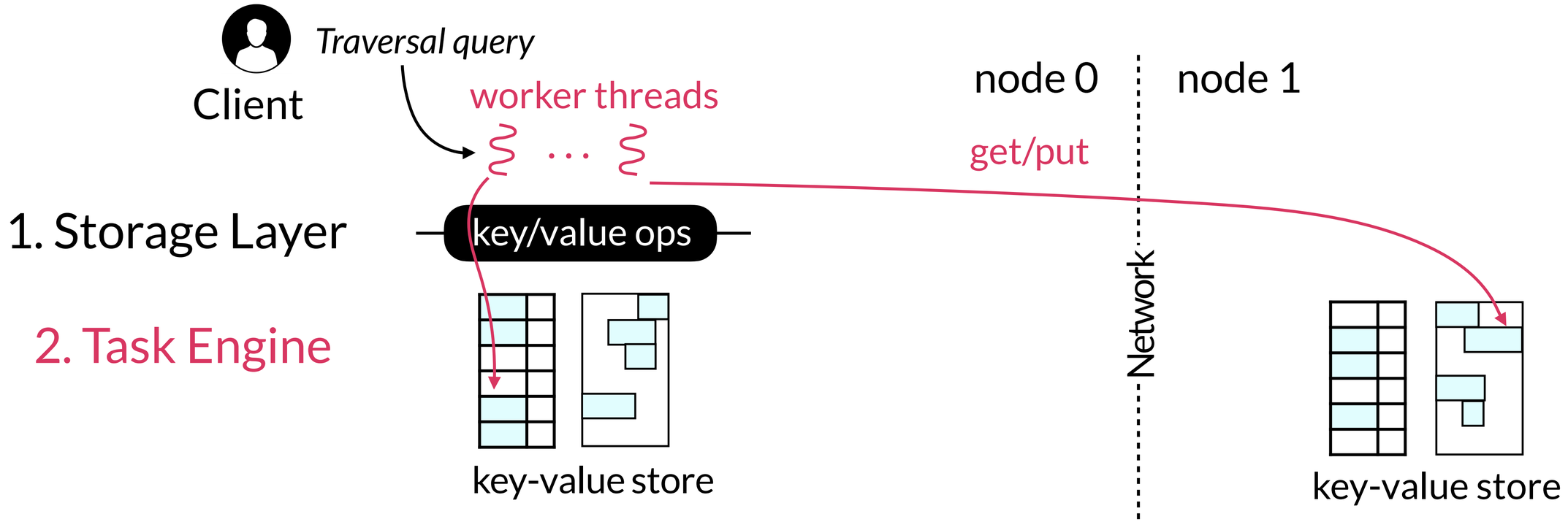
System Architecture

1. Storage Layer

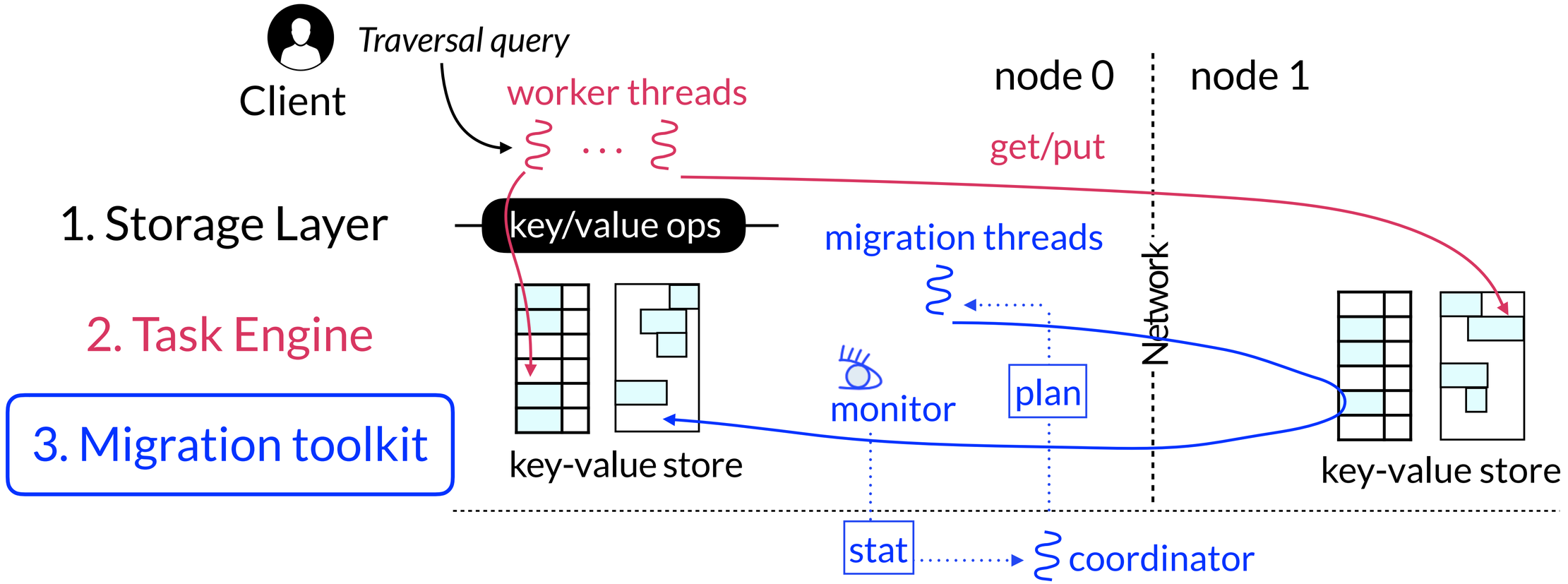




System Architecture



System Architecture



Agenda

Unilateral Migration Protocol

- Efficient Source Bypassing Migration

Integration with Location Cache

- Fully-localized Split Migration

Separate Monitoring

- Lightweight Fine-grained Migration

Agenda

Unilateral Migration Protocol

- Efficient Source Bypassing Migration

Integration with Location Cache

- Fully-localized Split Migration

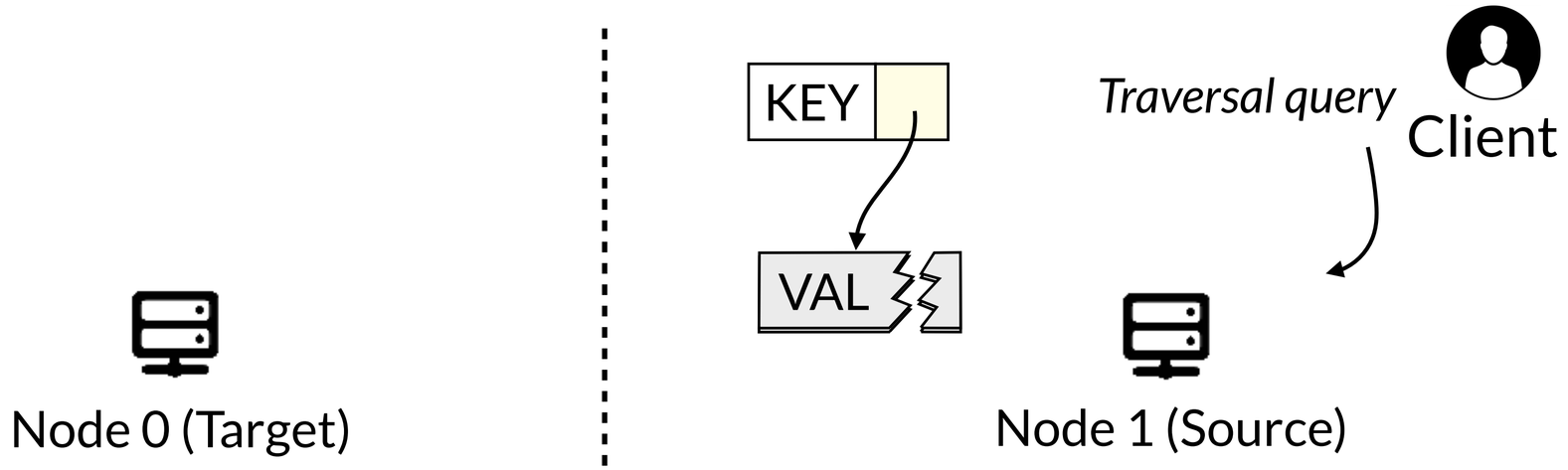
Separate Monitoring

- Lightweight Fine-grained Migration

Unilateral Migration Protocol

Migration is done by the target alone (bypassing source CPU)

- **No interruption** to the source
- Instant migration



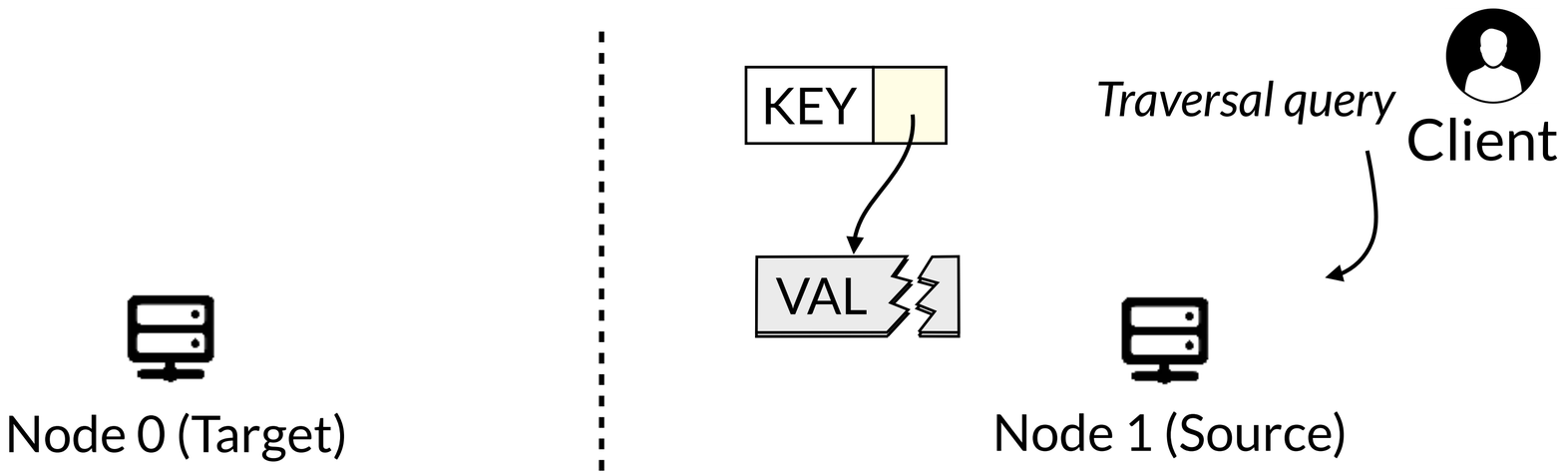
Unilateral Migration Protocol

Migration is done by the target alone (bypassing source CPU)

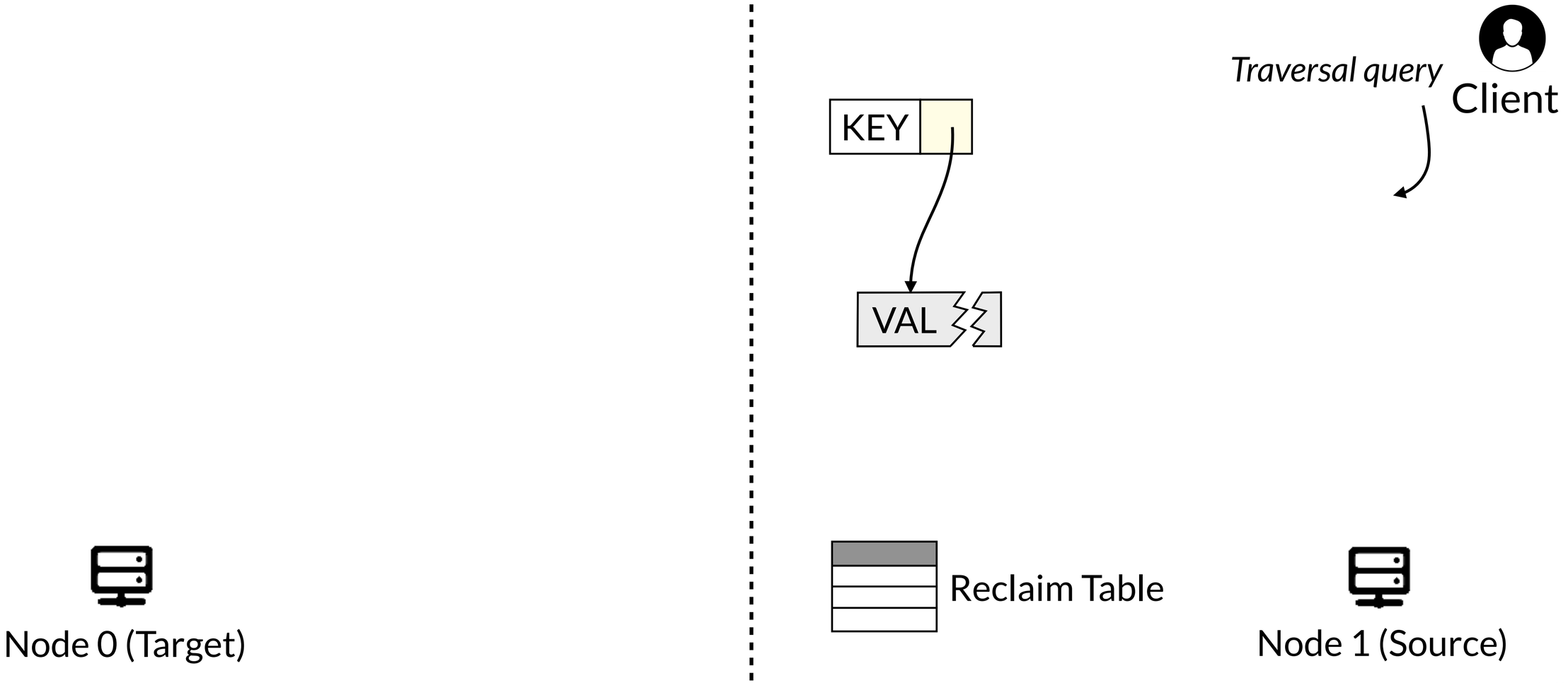
- **No interruption** to the source
- Instant migration

Leverage RDMA one-sided primitives (bypassing CPU)

- Read, Write, CAS
- Weak semantics

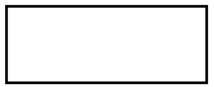


Unilateral Migration Protocol

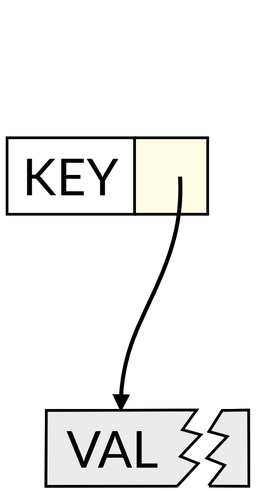


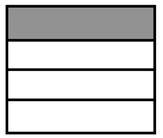
Unilateral Migration Protocol

1. Allocate memory



Node 0 (Target)





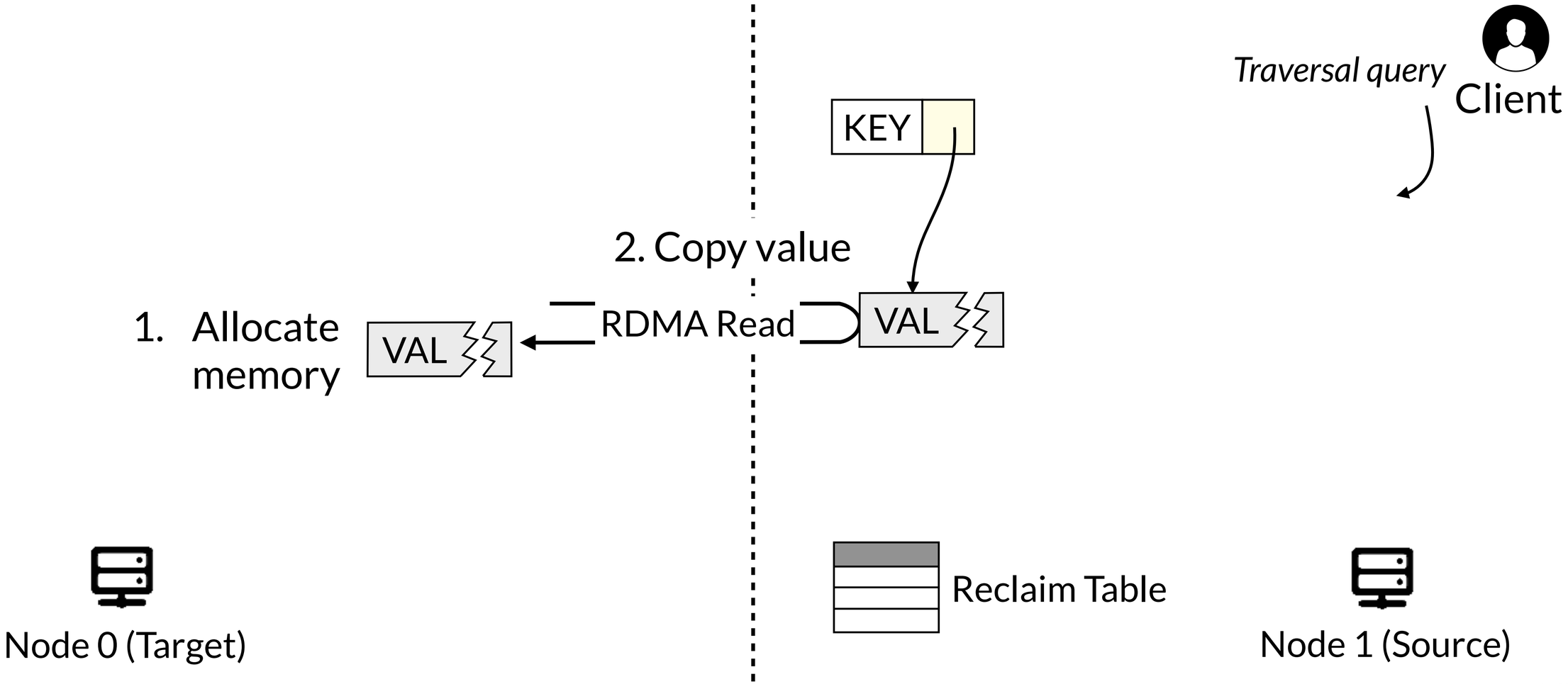
Reclaim Table

Traversal query

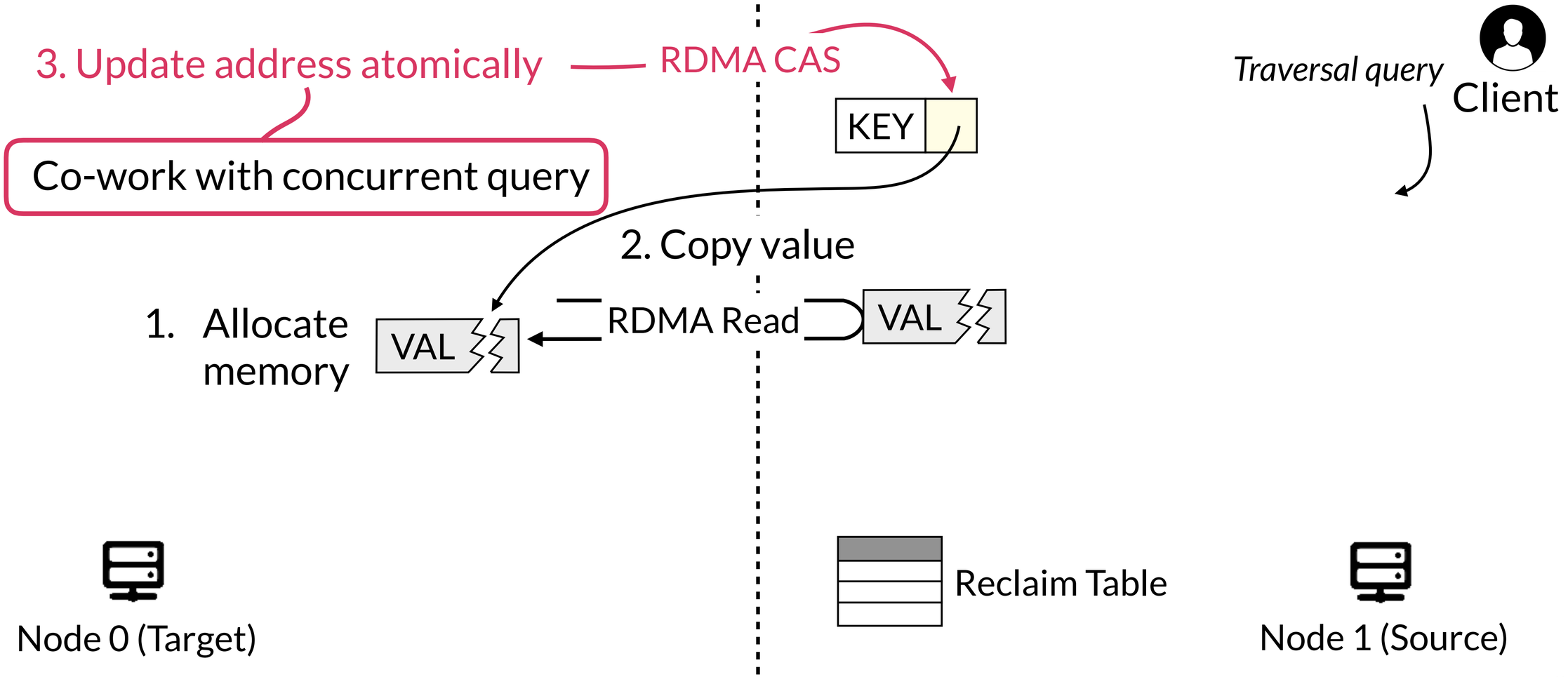
Client

Node 1 (Source)

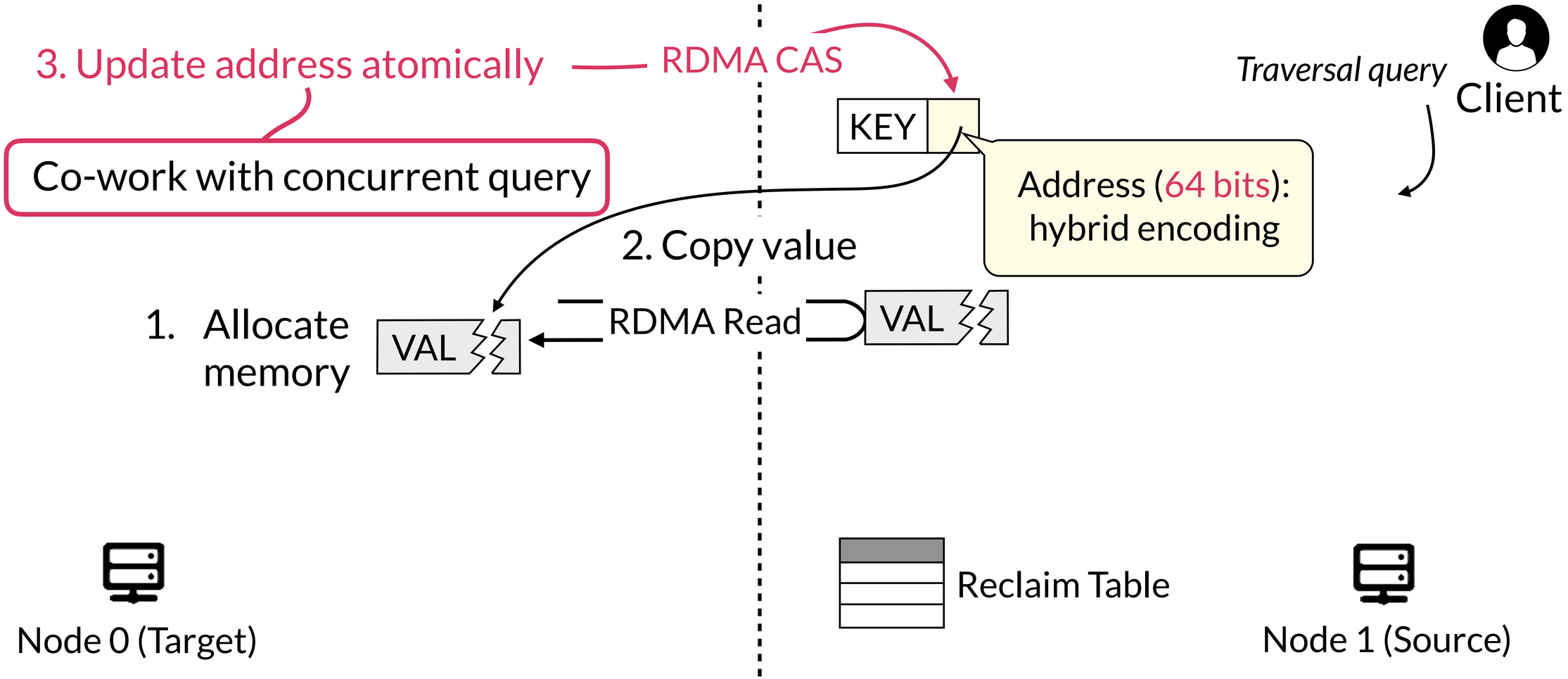
Unilateral Migration Protocol



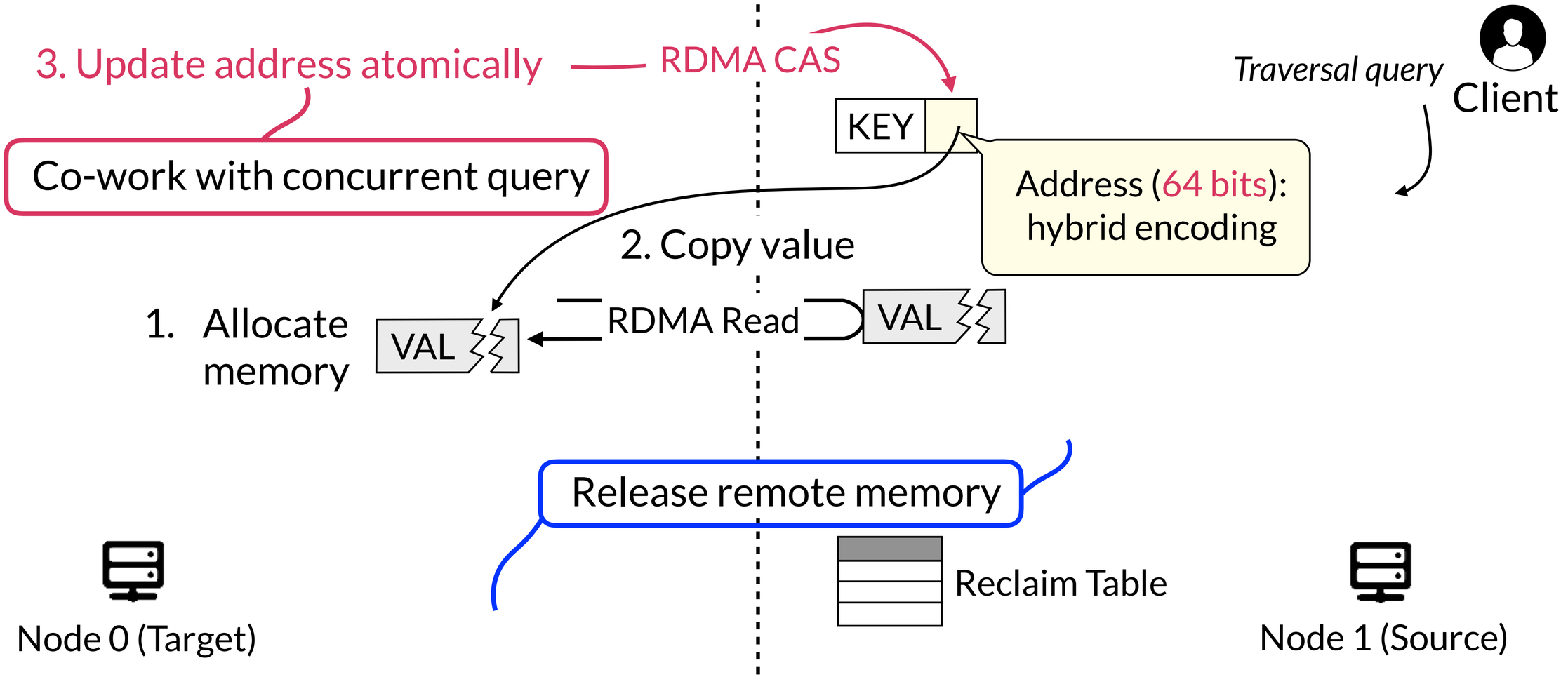
Unilateral Migration Protocol



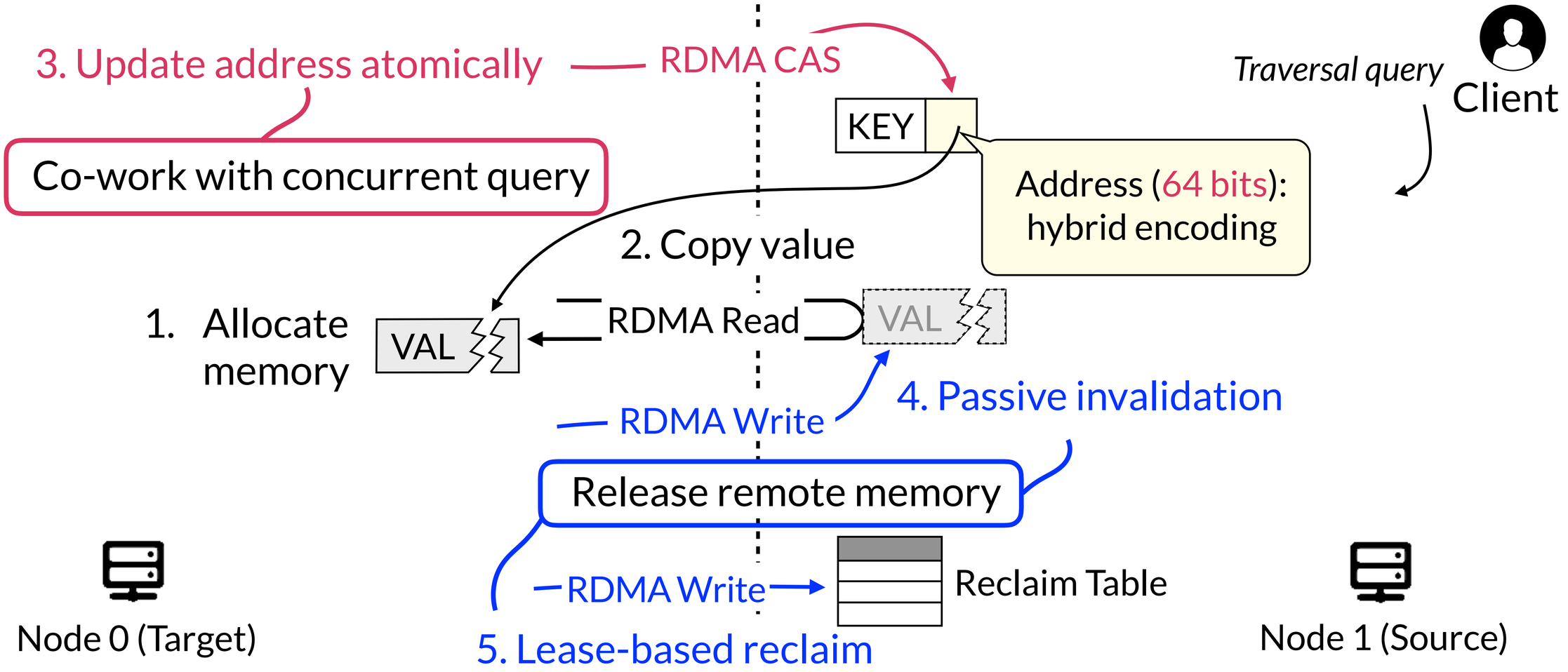
Unilateral Migration Protocol



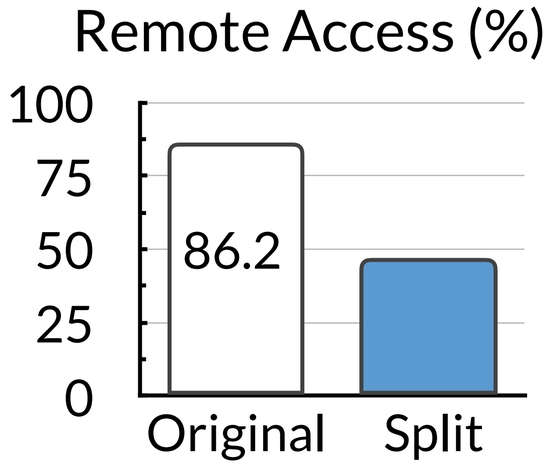
Unilateral Migration Protocol



Unilateral Migration Protocol

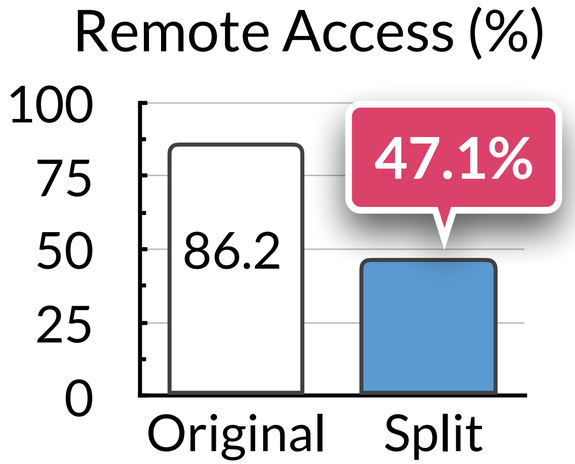


Towards Fully-localized Migration



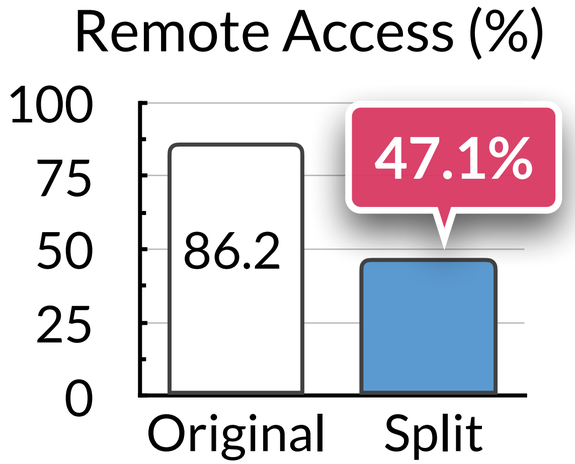
[1] DrTM (SOSP'15)

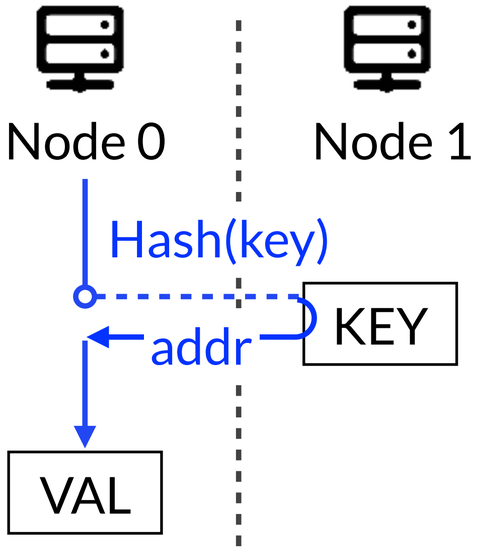
Towards Fully-localized Migration



[1] DrTM (SOSP'15)

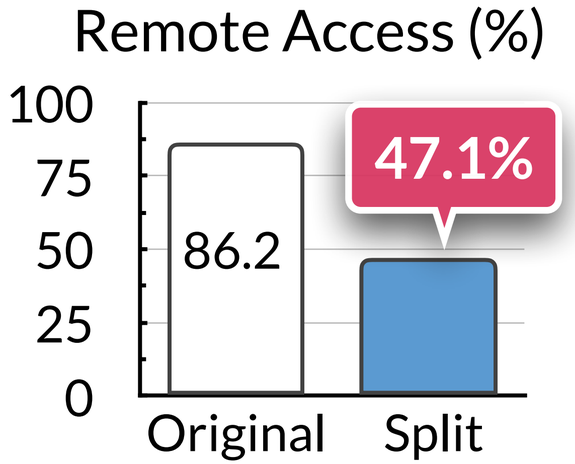
Towards Fully-localized Migration

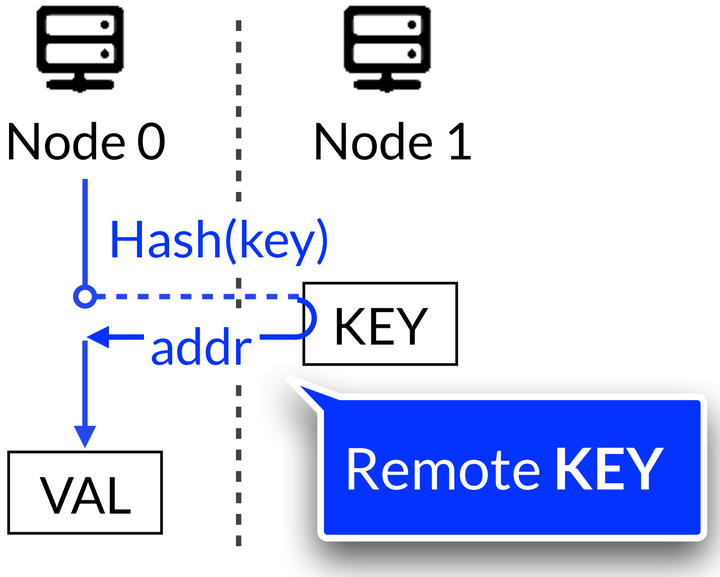




[1] DrTM (SOSP'15)

Towards Fully-localized Migration



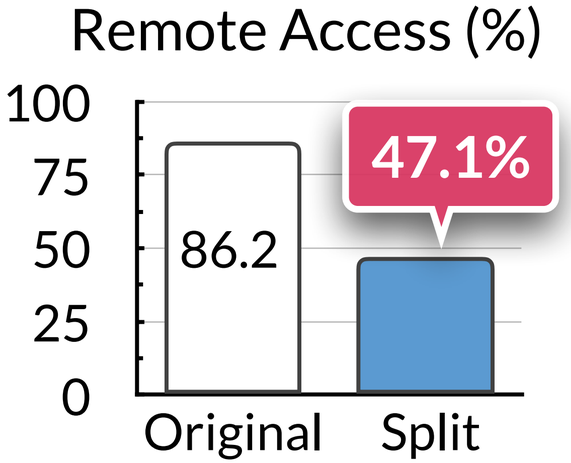


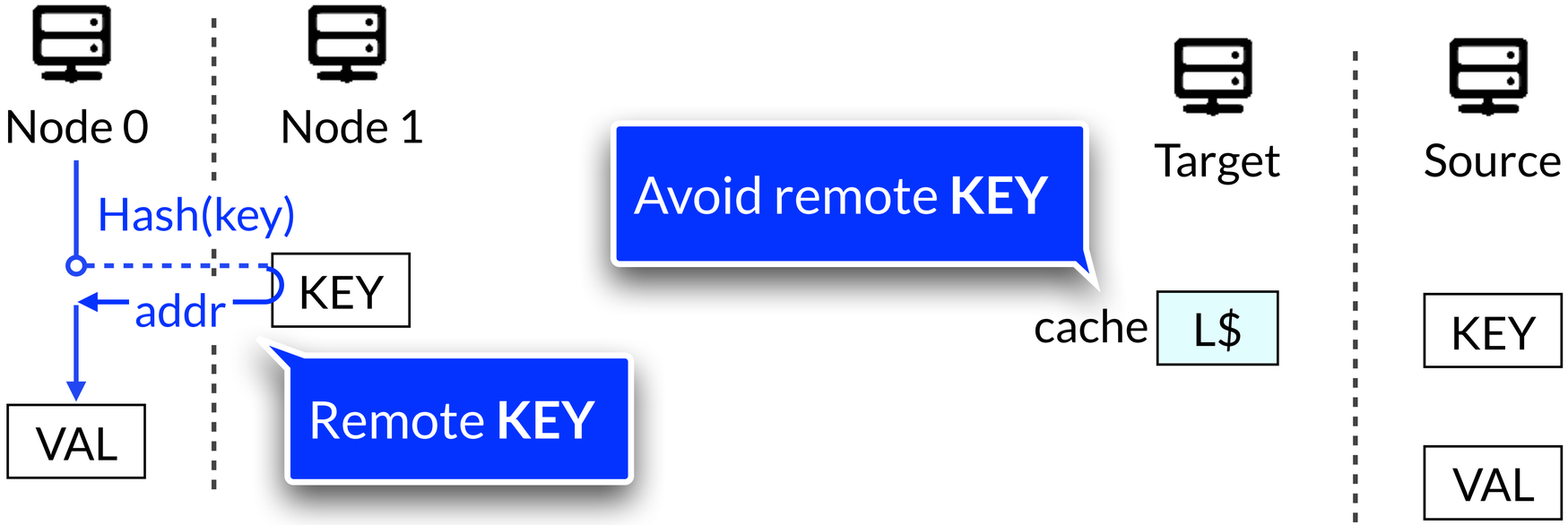
[1] DrTM (SOSP'15)

Towards Fully-localized Migration

Location cache[1]: perfect match for split migration

- Avoid half of remote access



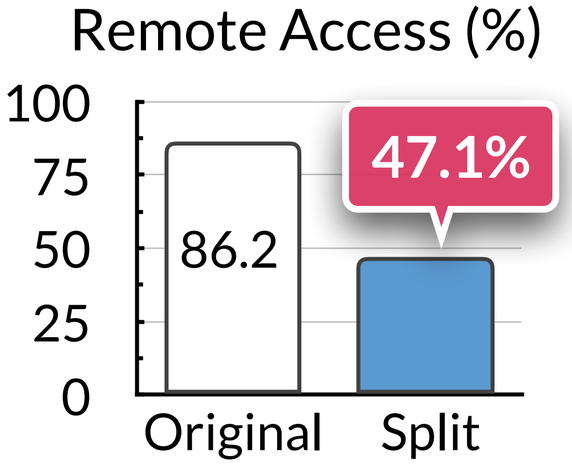


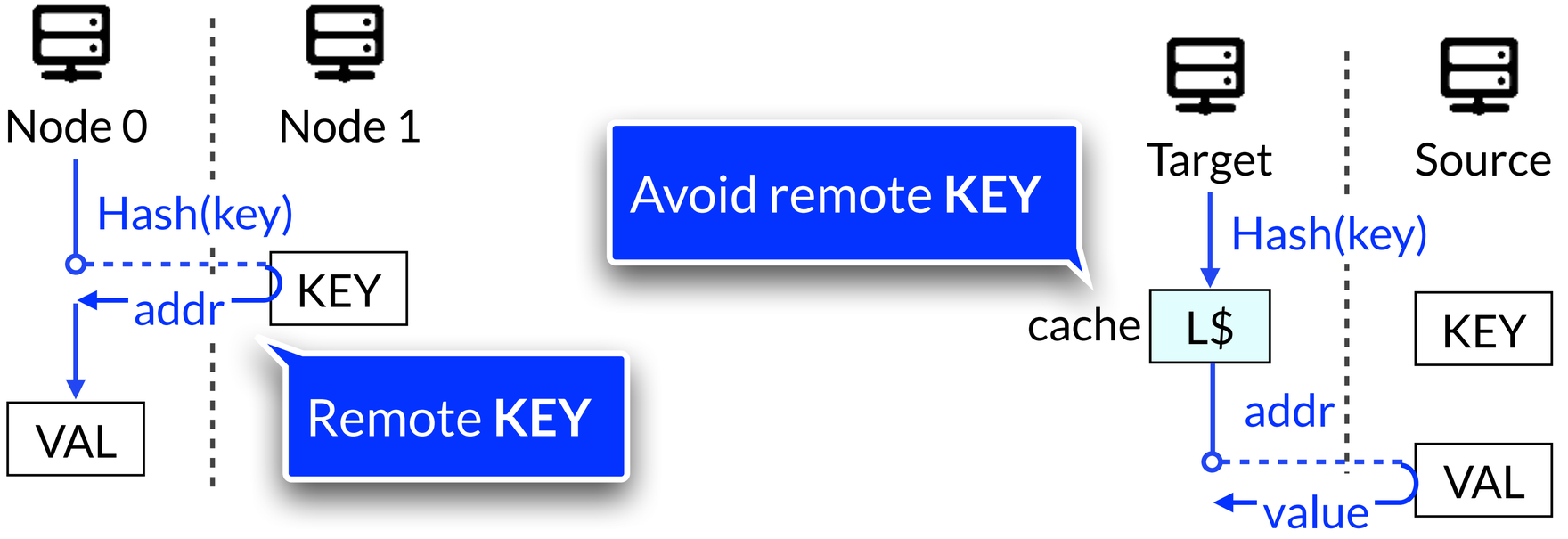
[1] DrTM (SOSP'15)

Towards Fully-localized Migration

Location cache[1]: perfect match for split migration

- Avoid half of remote access



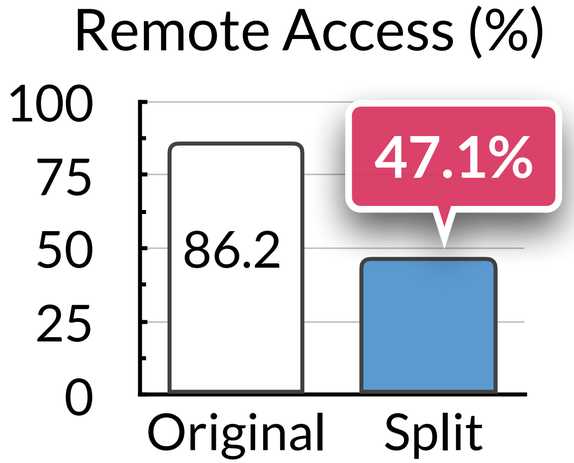


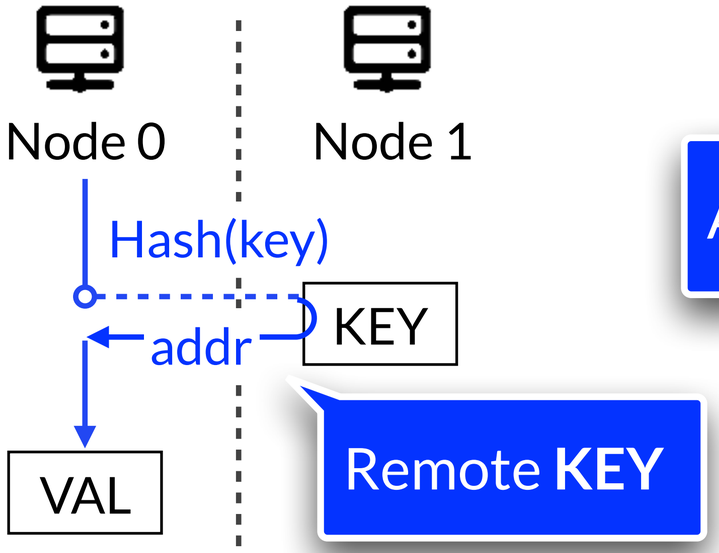
[1] DrTM (SOSP'15)

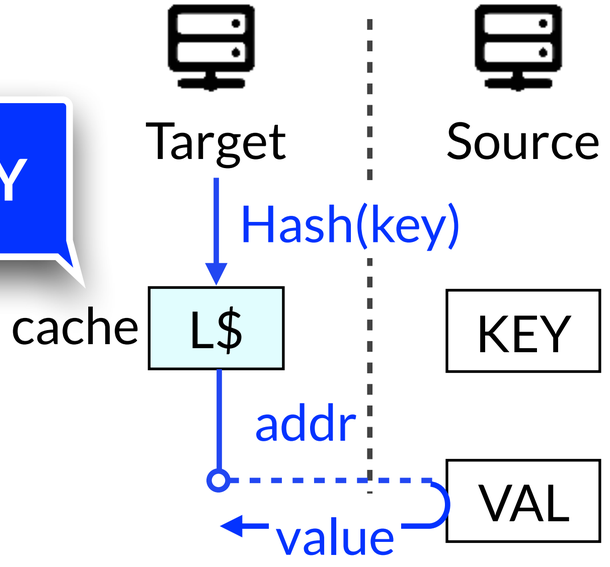
Towards Fully-localized Migration

Location cache[1]: perfect match for split migration

- Avoid half of remote access
- Candidates: remote data accessed frequently

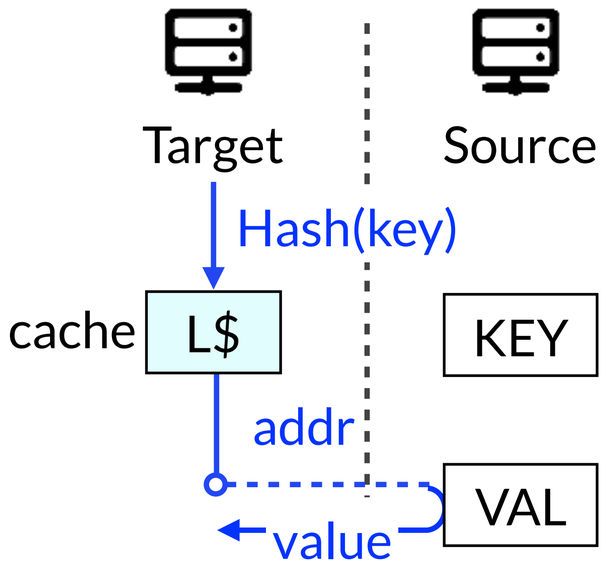






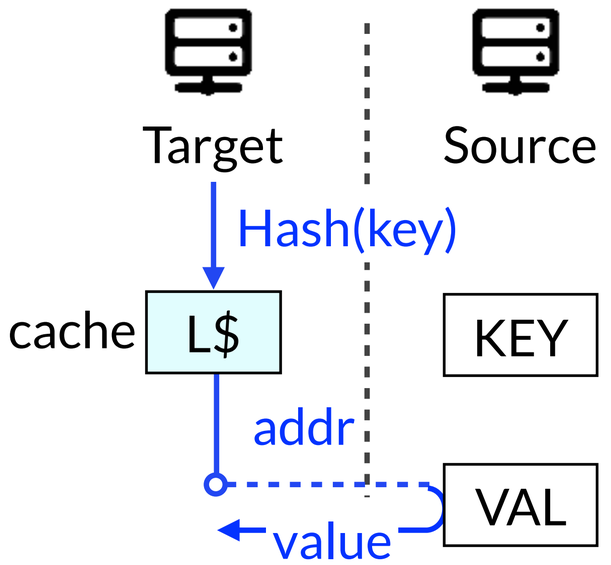
[1] DrTM (SOSP'15)

Integration with Location Cache



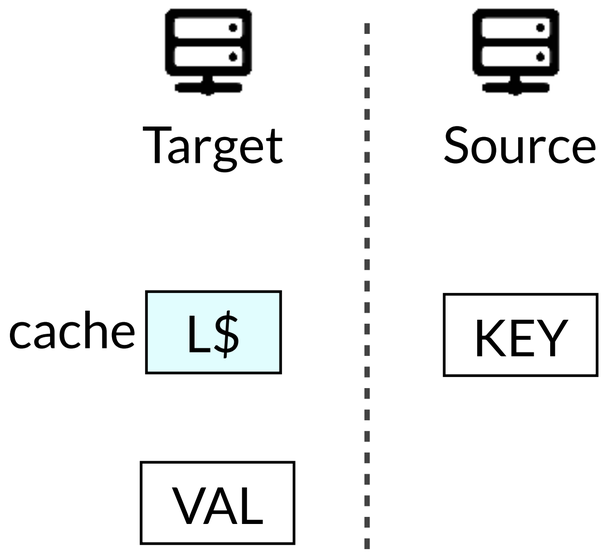
Integration with Location Cache

Avoid both remote **keys** and **values**



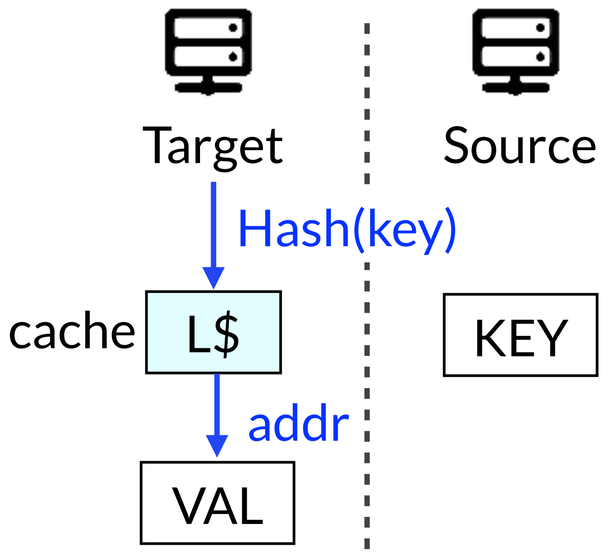
Integration with Location Cache

Avoid both remote **keys** and **values**



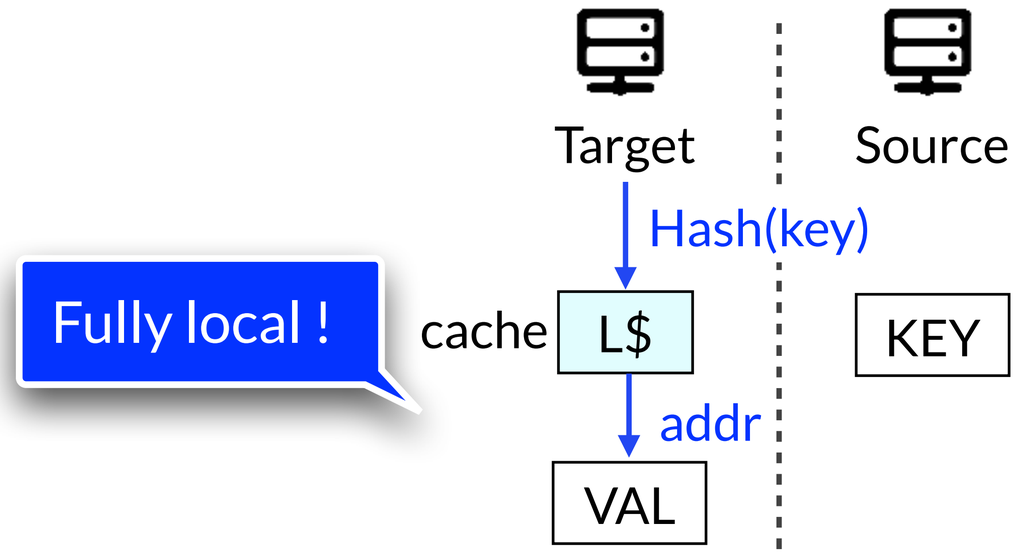
Integration with Location Cache

Avoid both remote **keys** and **values**



Integration with Location Cache

Avoid both remote **keys** and **values**

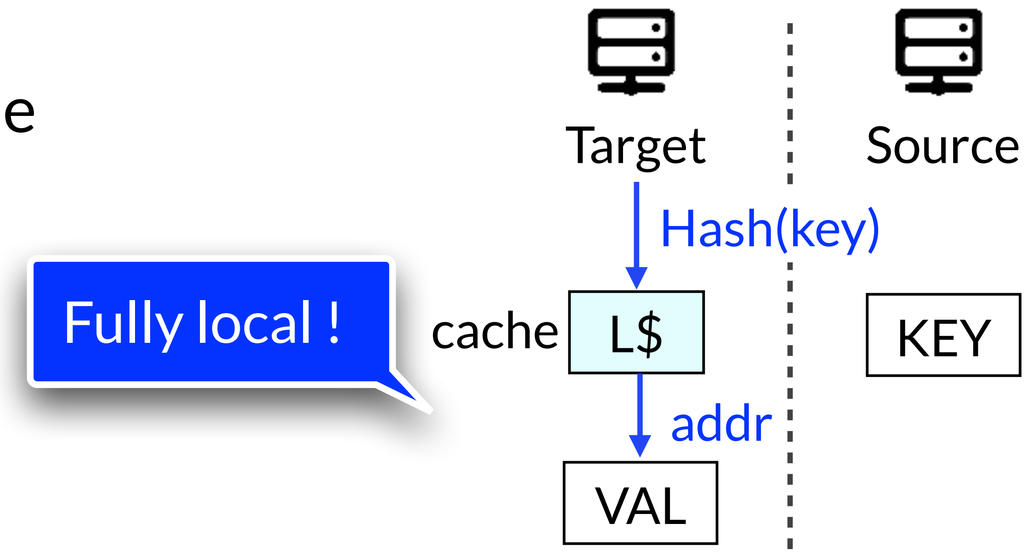


Integration with Location Cache

Avoid both remote **keys** and **values**

Consistency of key

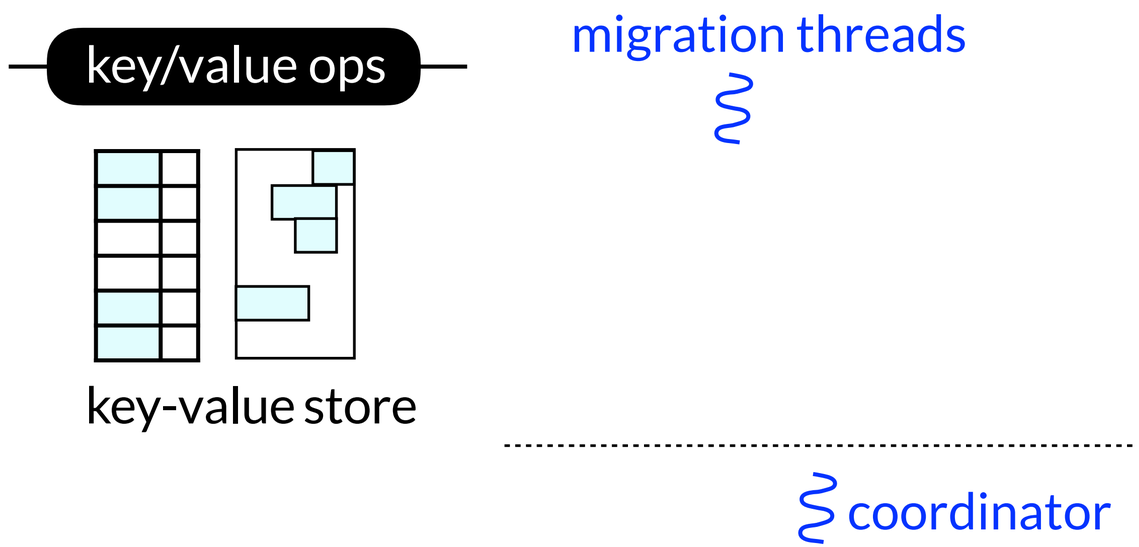
- Reuse passive invalidation of cache
- Lease-based invalidation



Separate Monitoring: Lightweight Monitor

Fine-grained migration: fine-grained monitoring

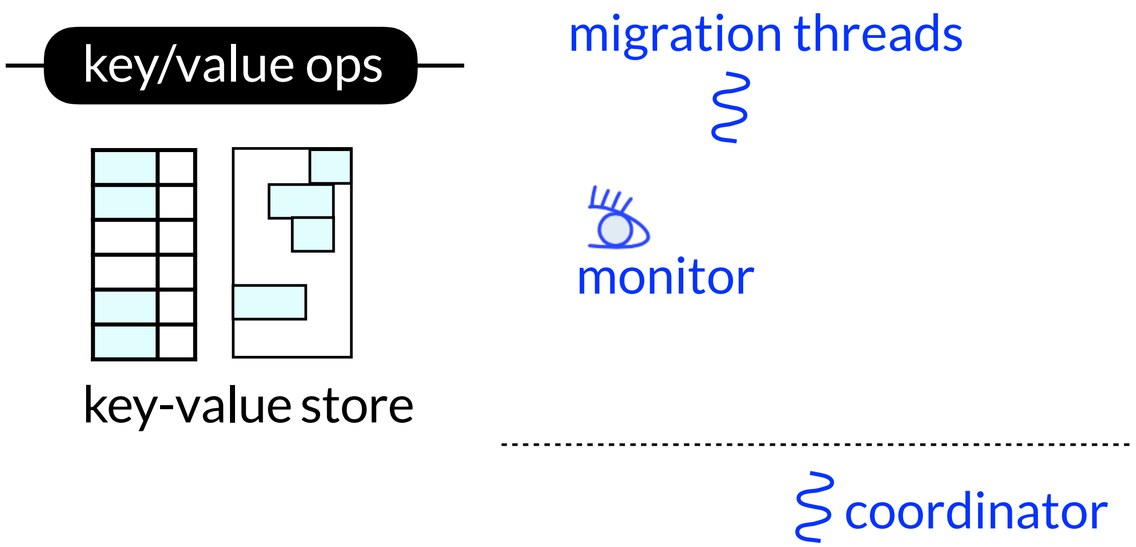
- Non-trivial overhead



Separate Monitoring: Lightweight Monitor

Fine-grained migration: fine-grained monitoring

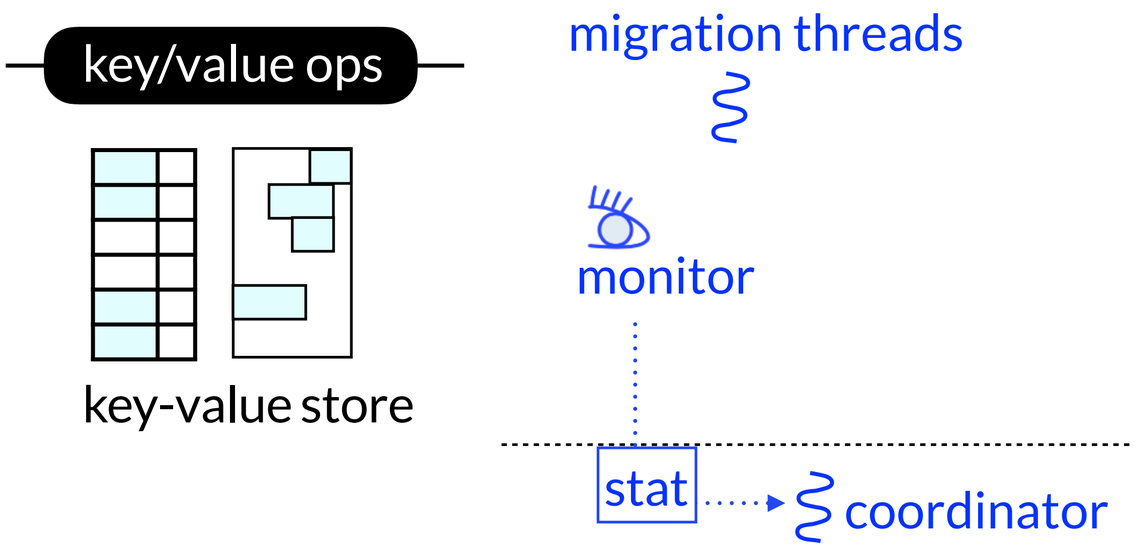
- Non-trivial overhead



Separate Monitoring: Lightweight Monitor

Fine-grained migration: fine-grained monitoring

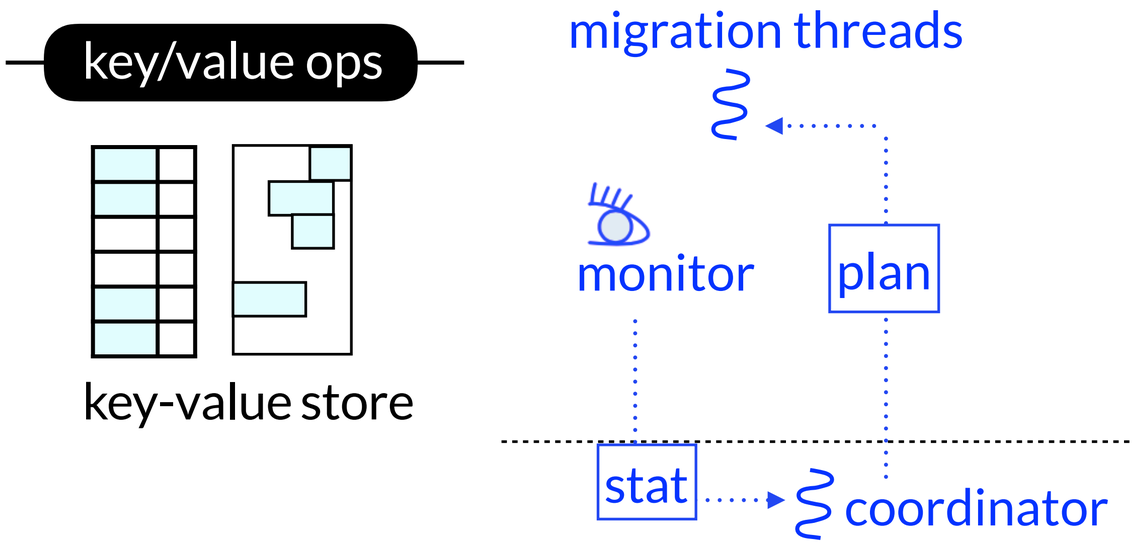
- Non-trivial overhead



Separate Monitoring: Lightweight Monitor

Fine-grained migration: fine-grained monitoring

- Non-trivial overhead



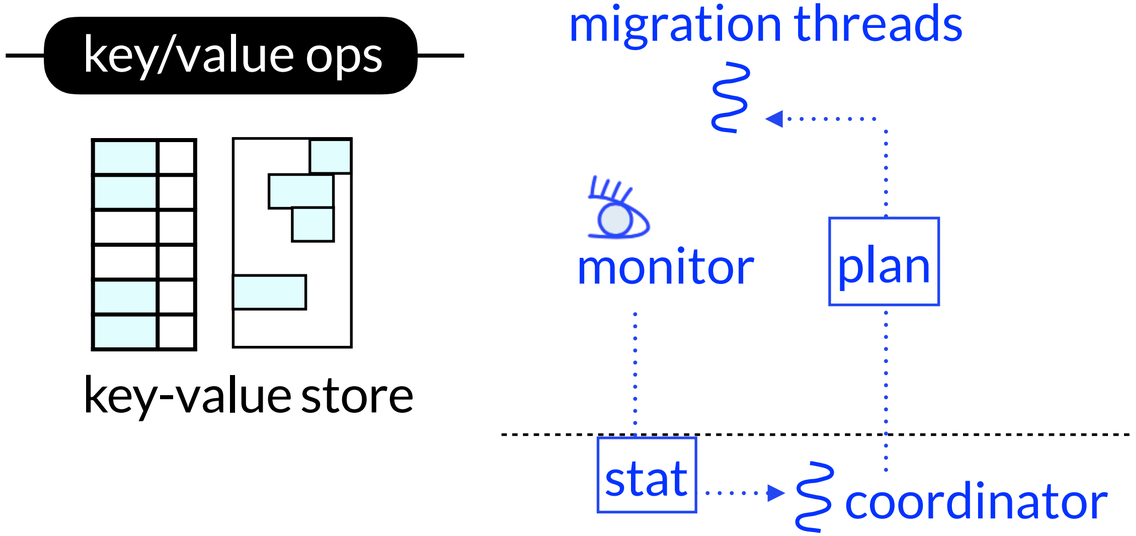
Separate Monitoring: Lightweight Monitor

Fine-grained migration: fine-grained monitoring

- Non-trivial overhead

Remote access:

- **Reuse location cache**



Separate Monitoring: Lightweight Monitor

Fine-grained migration: fine-grained monitoring

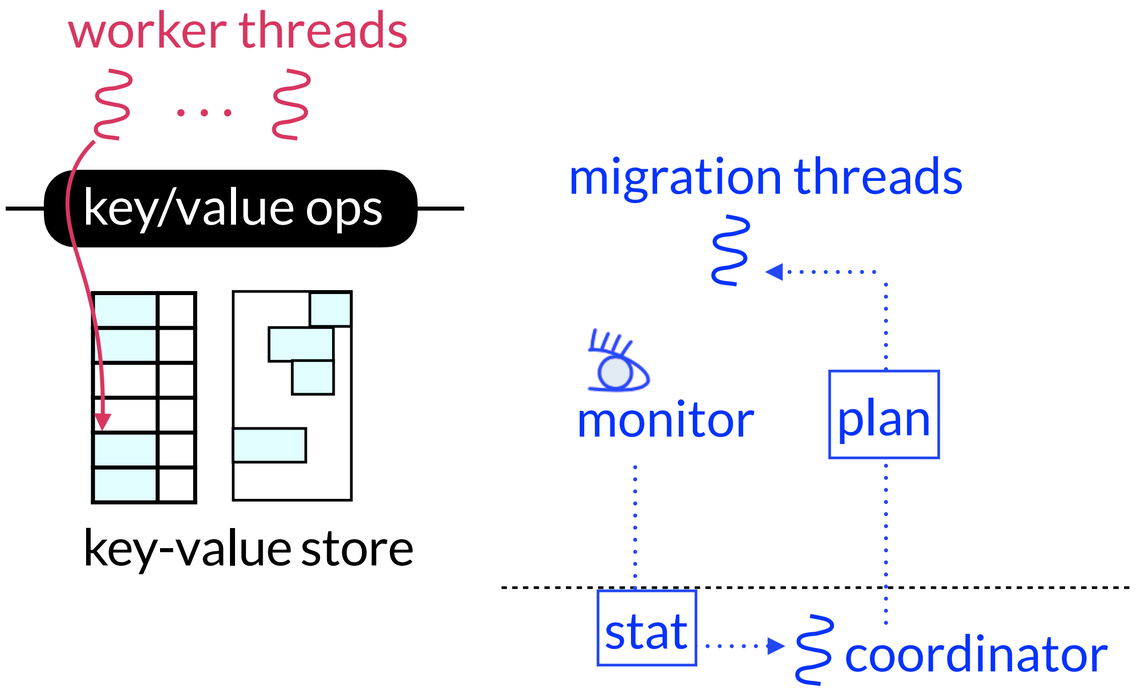
- Non-trivial overhead

Remote access:

- **Reuse location cache**

Local access:

- On-demand tracking



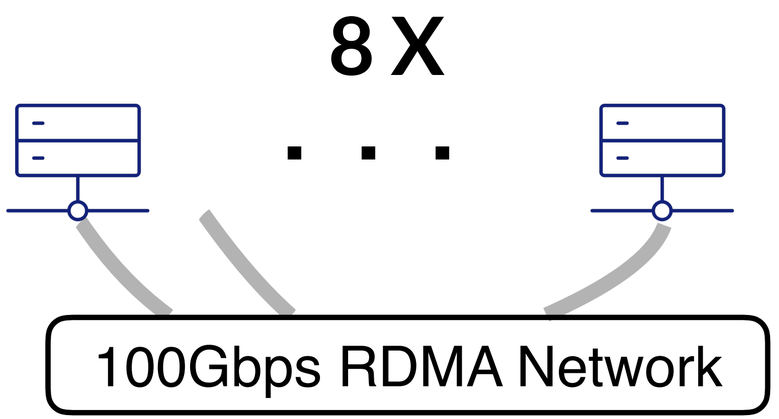
Others

- Check-and-forward mechanism: support evolving graph
- Memory reclamation
- Failure handling
- Pipelined migration: speedup unilateral migration

Micro: YCSB-B Like Graph Traversal Benchmark

RMAT-26 dataset

95% traversal, 5% *PUT* (like YCSB-B)

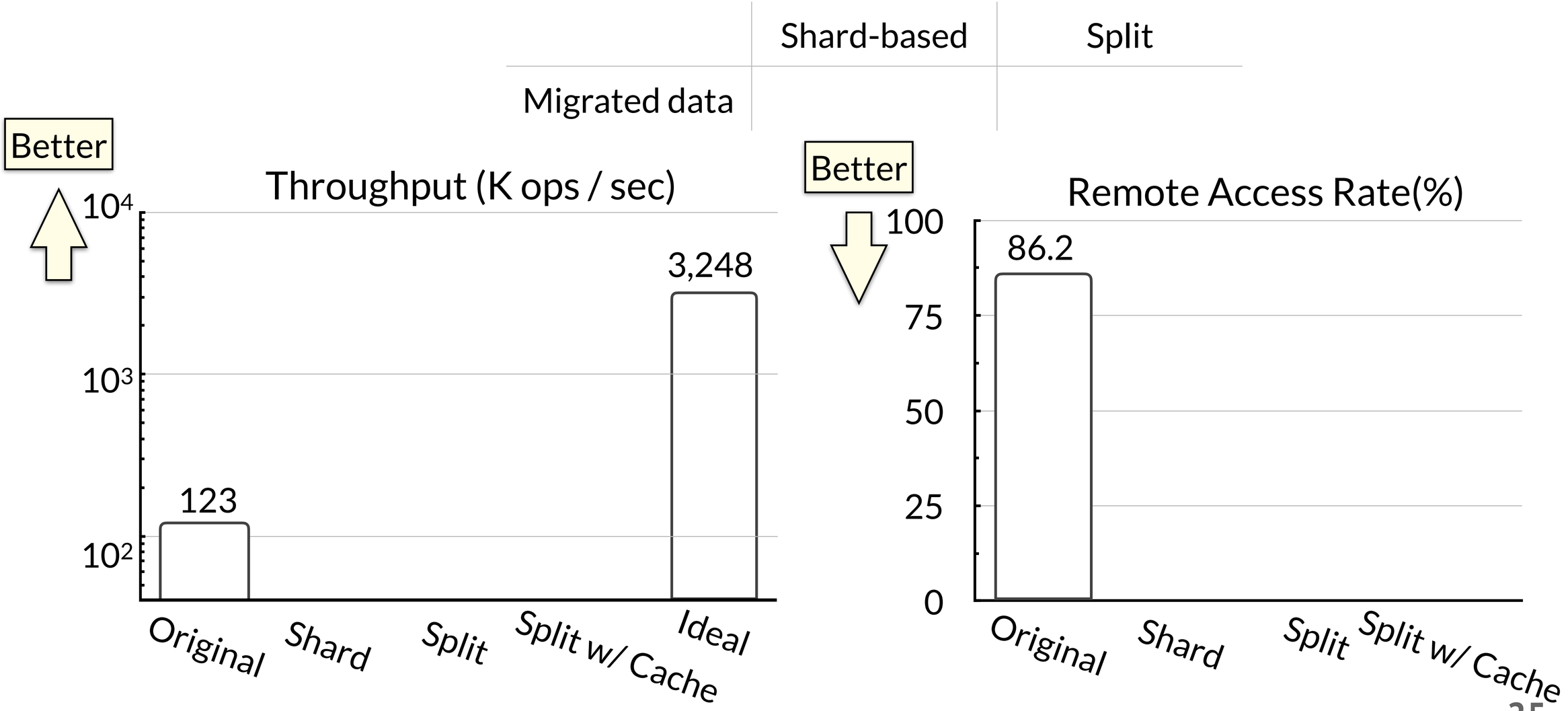


Baseline

- Ideal: (throughput on **1 node**) * **8**
- Shard-based: 800 shards

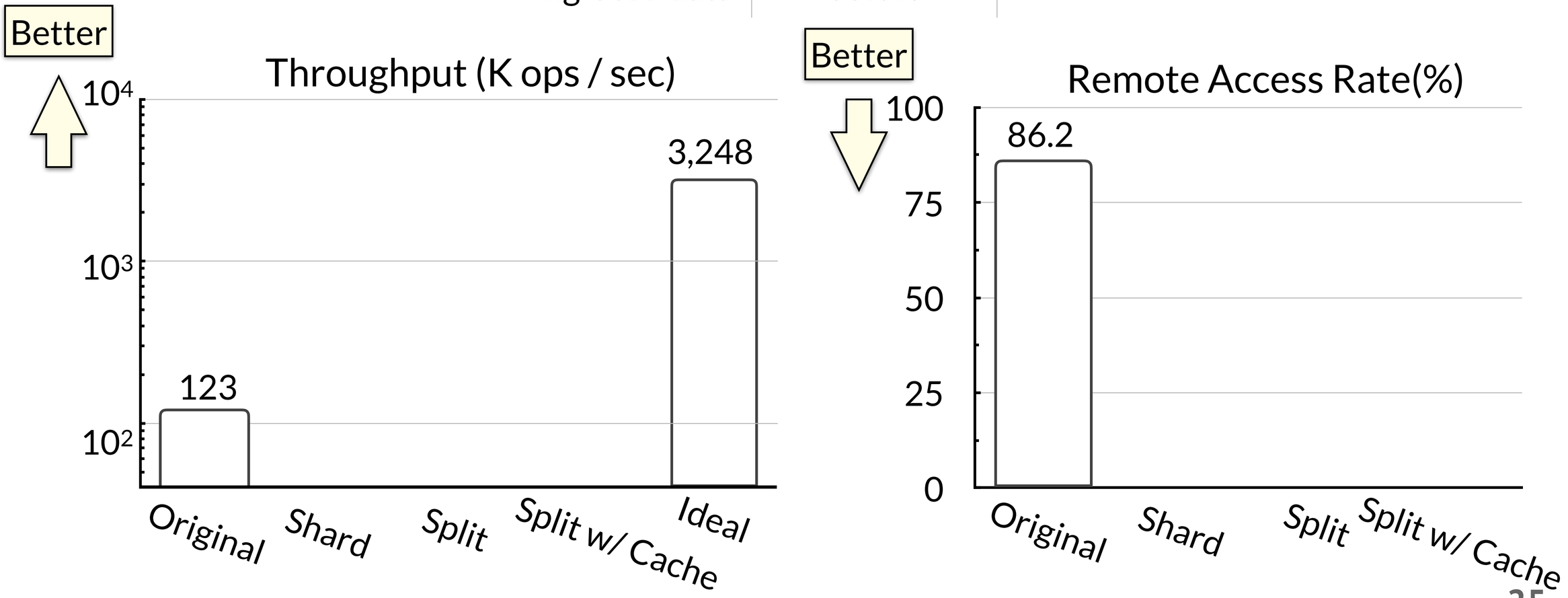
CPU	NIC	Memory
24 cores	2*CX4	128GB

Performance on Micro-benchmark



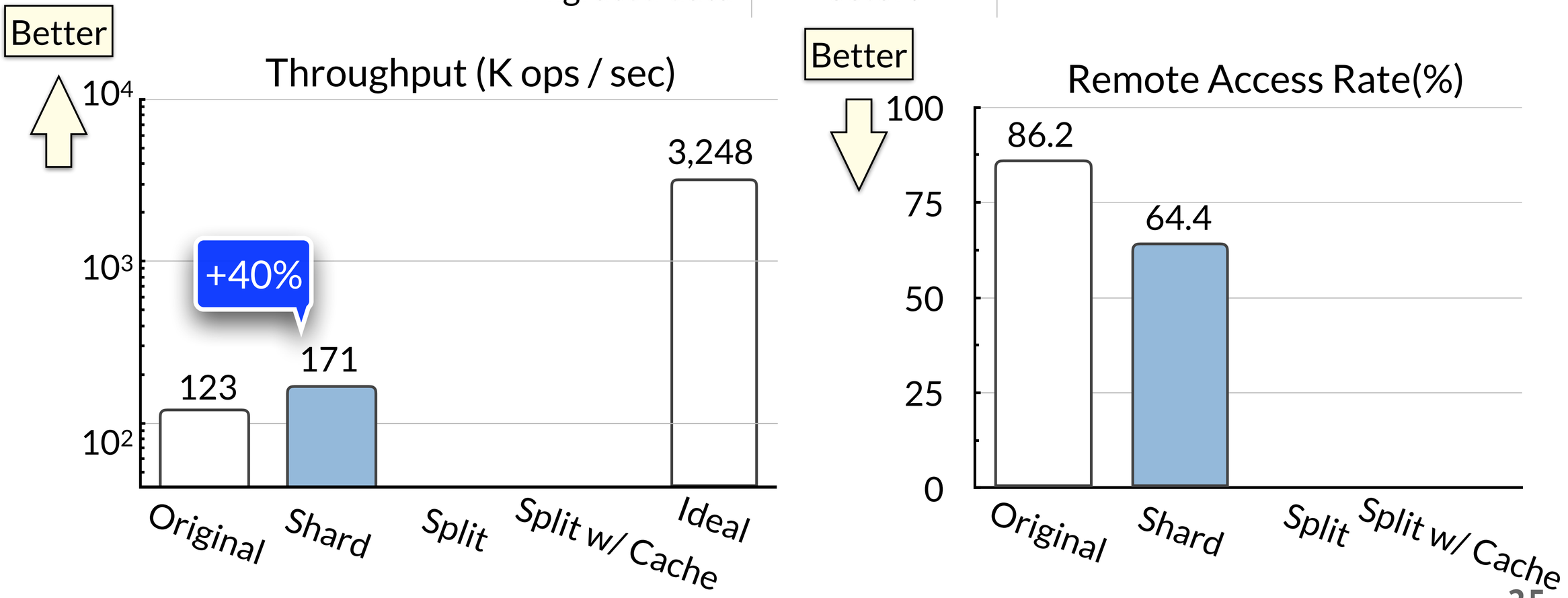
Performance on Micro-benchmark

	Shard-based	Split
Migrated data	**85.6%**	



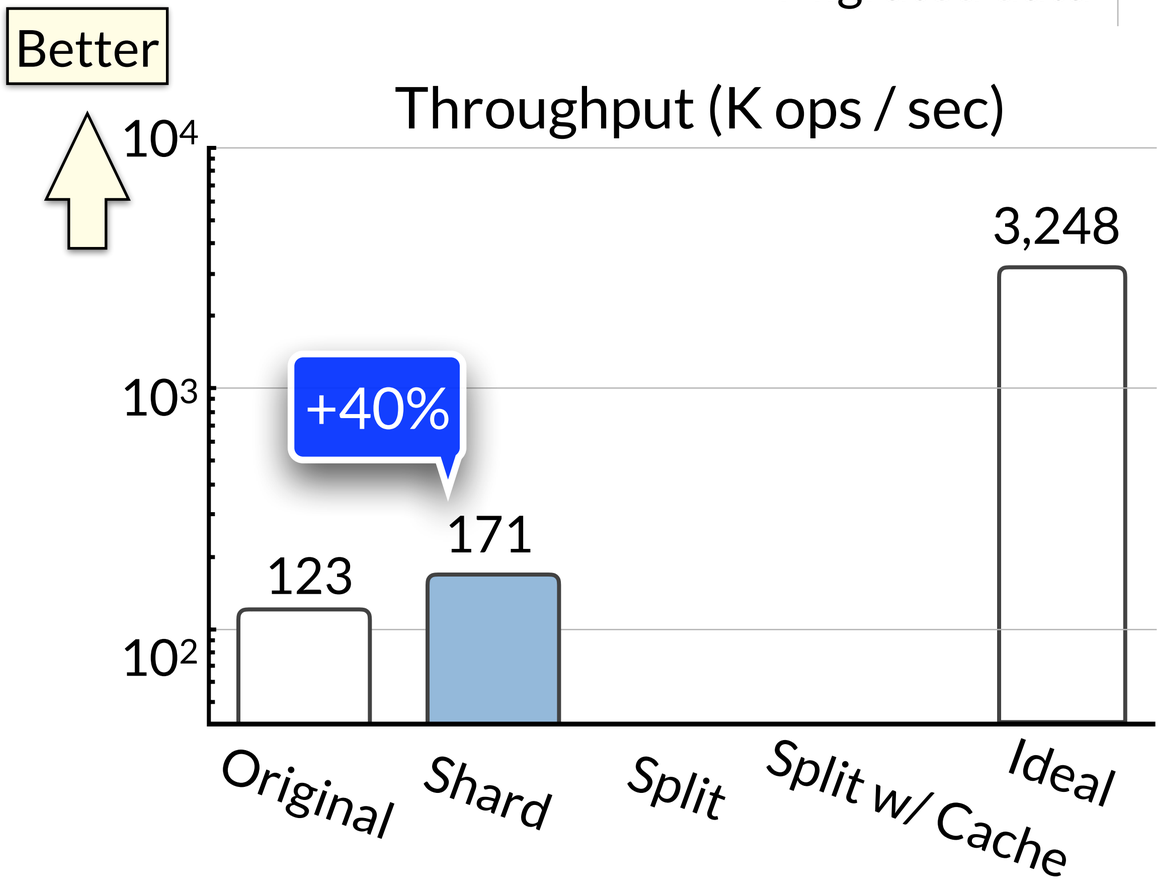
Performance on Micro-benchmark

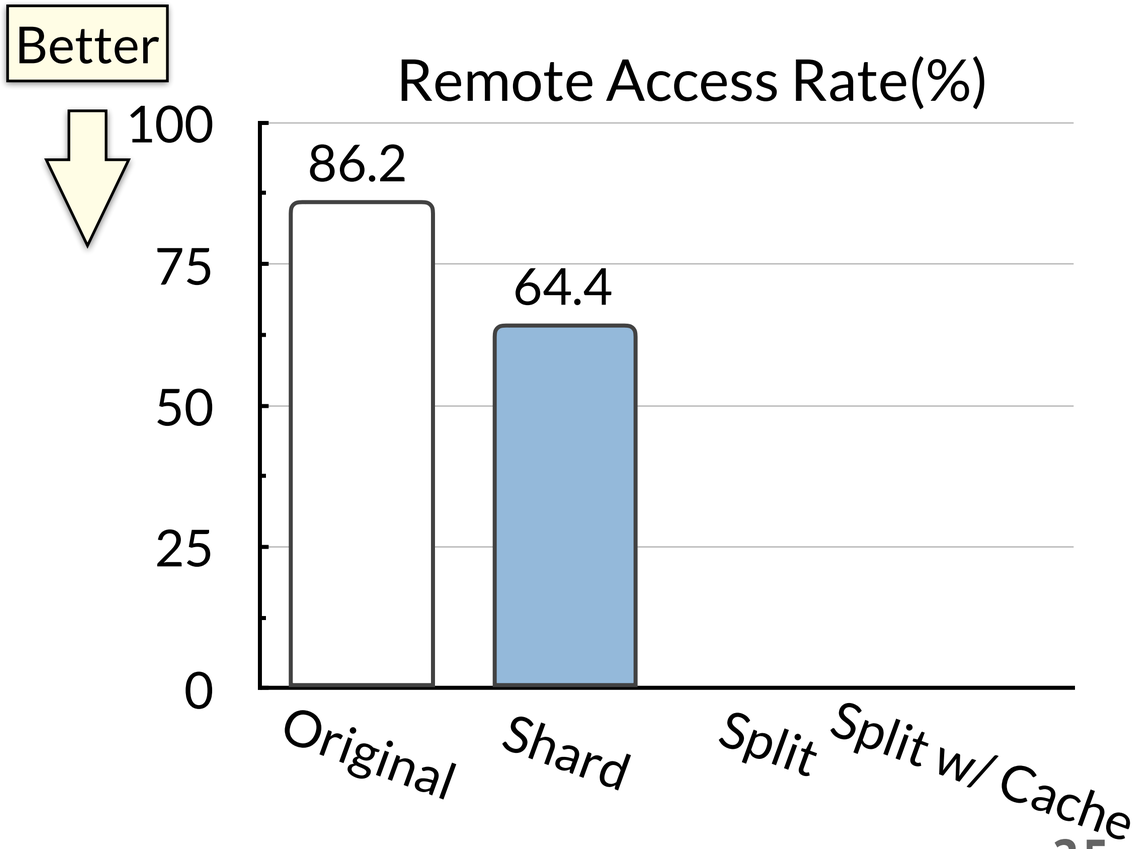
	Shard-based	Split
Migrated data	**85.6%**	



Performance on Micro-benchmark

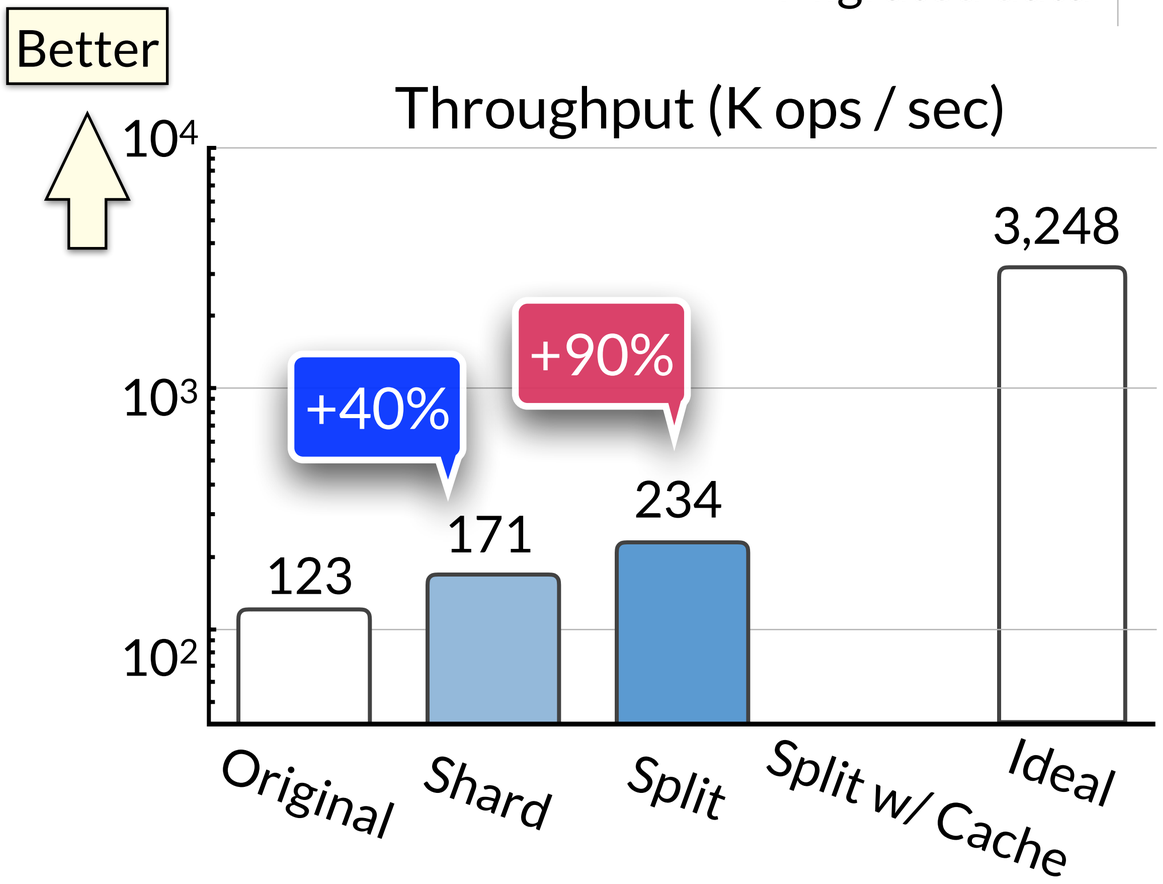
	Shard-based	Split
Migrated data	**85.6%**	**0.13%**

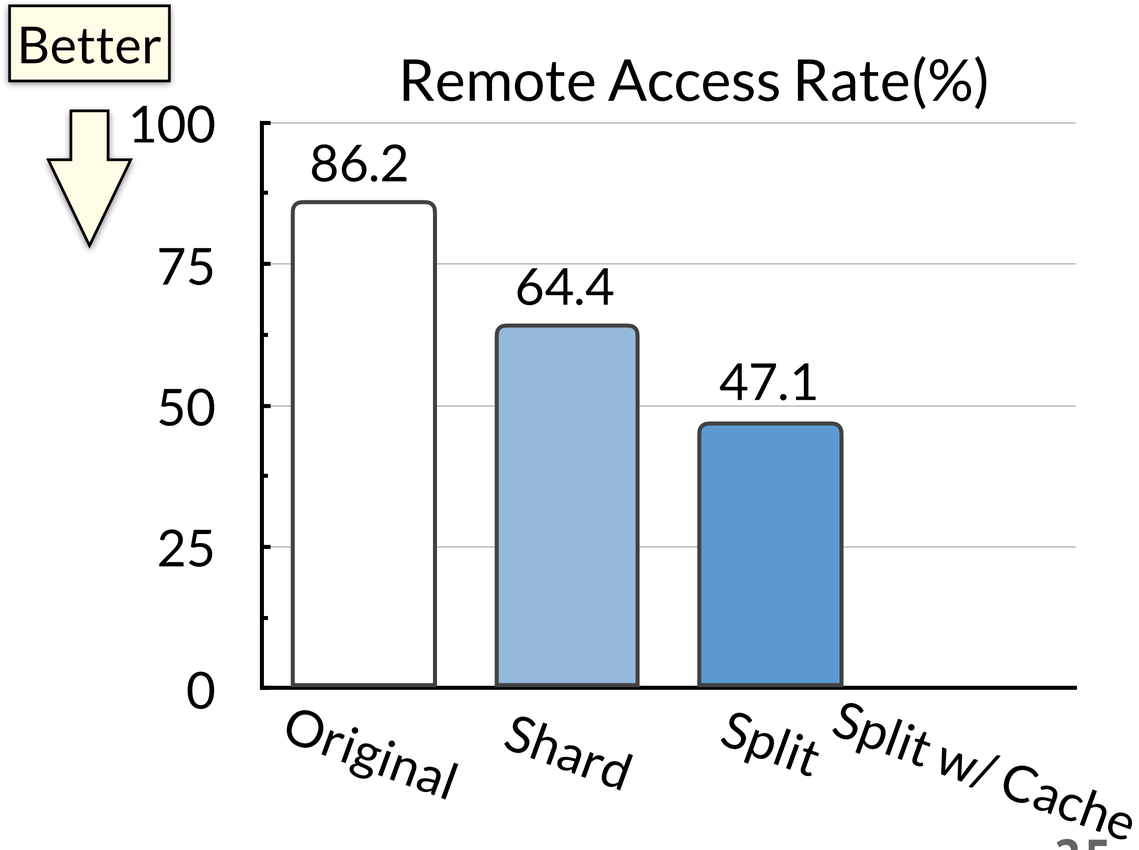




Performance on Micro-benchmark

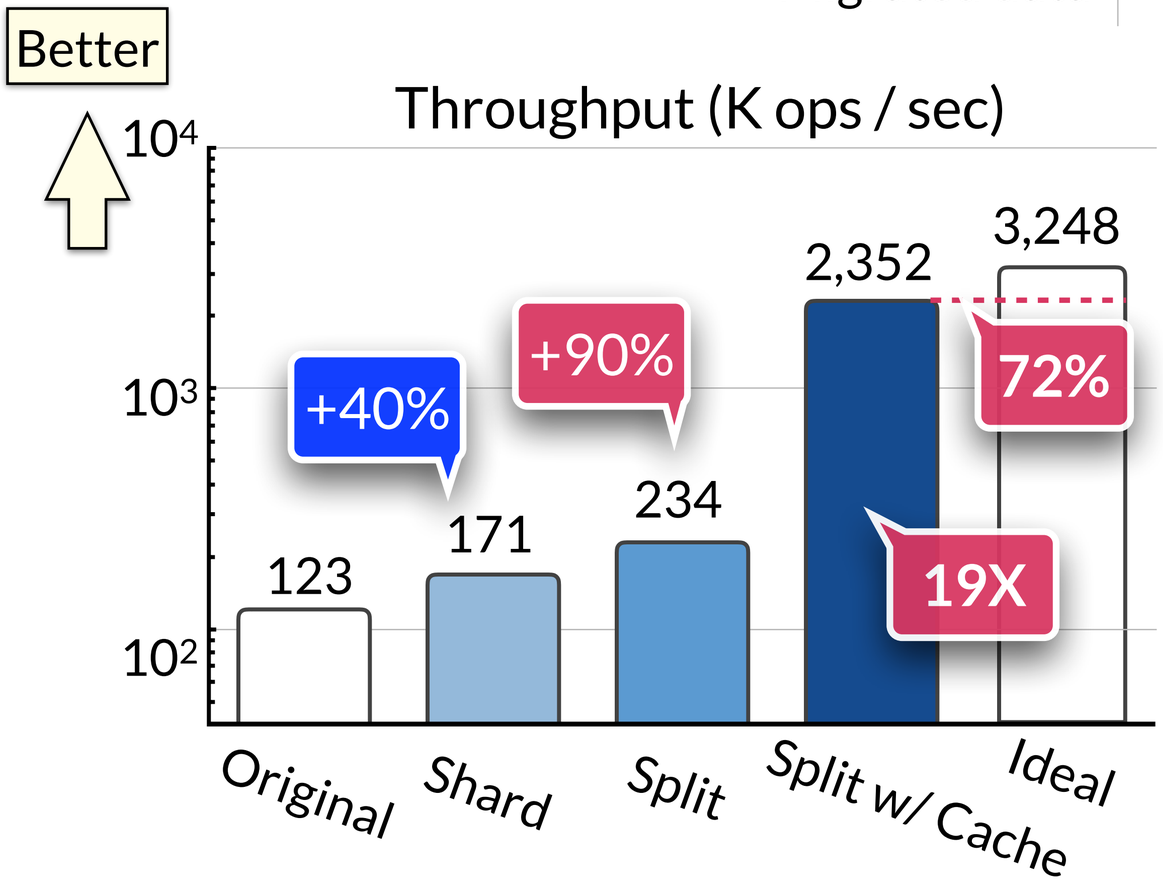
	Shard-based	Split
Migrated data	**85.6%**	**0.13%**

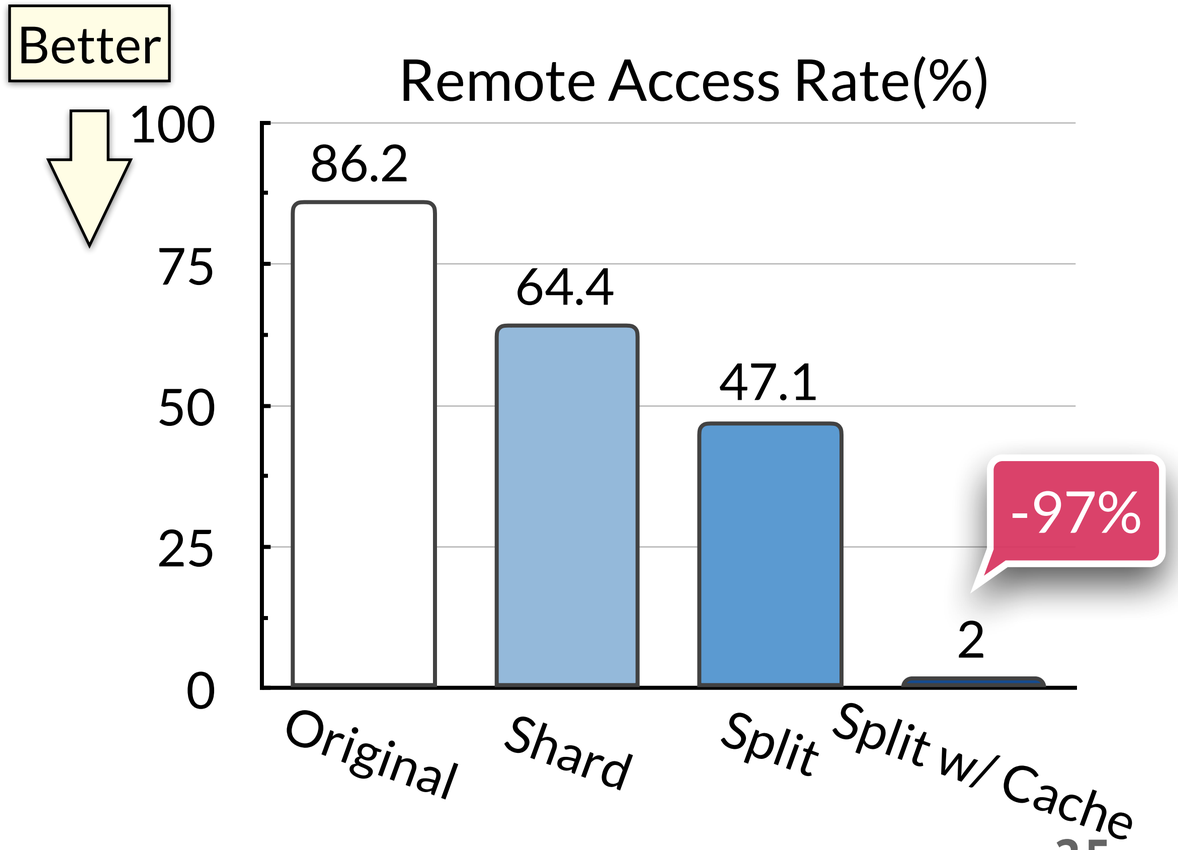




Performance on Micro-benchmark

	Shard-based	Split
Migrated data	**85.6%**	**0.13%**

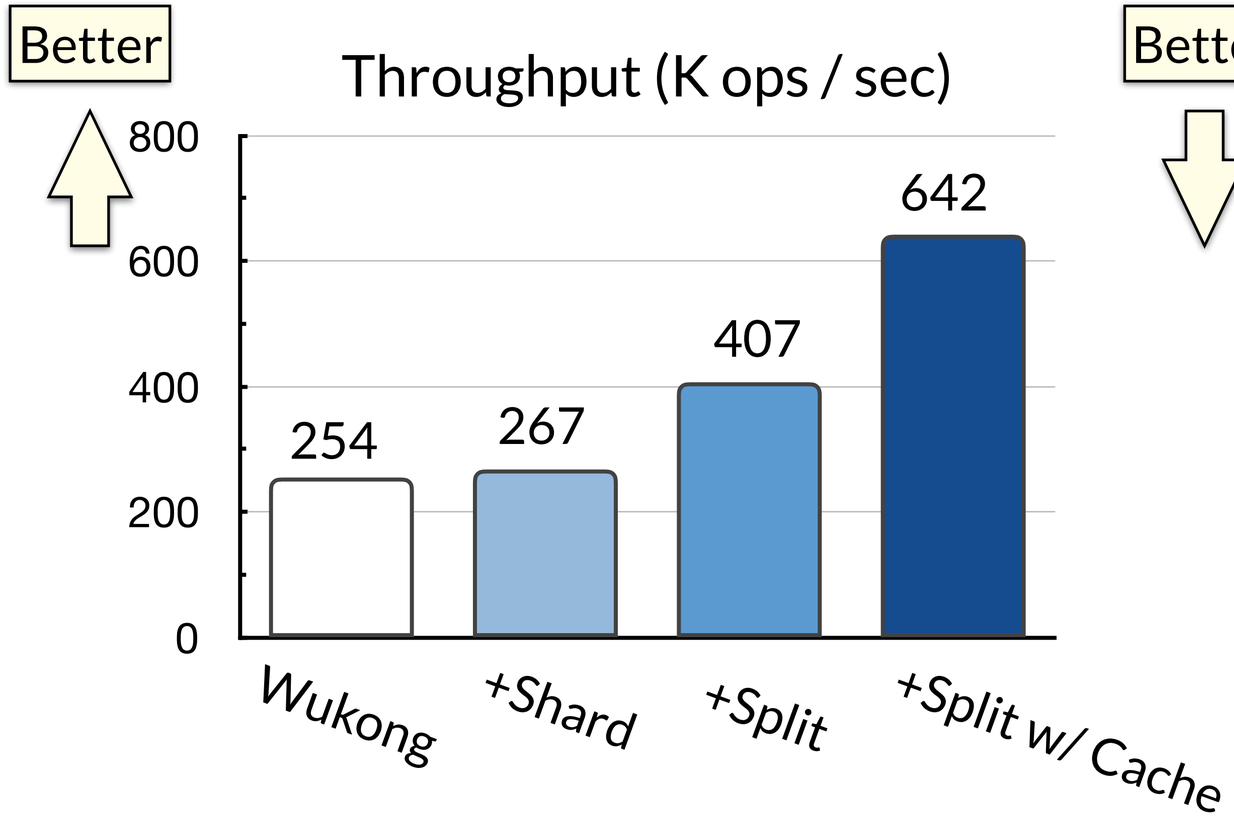


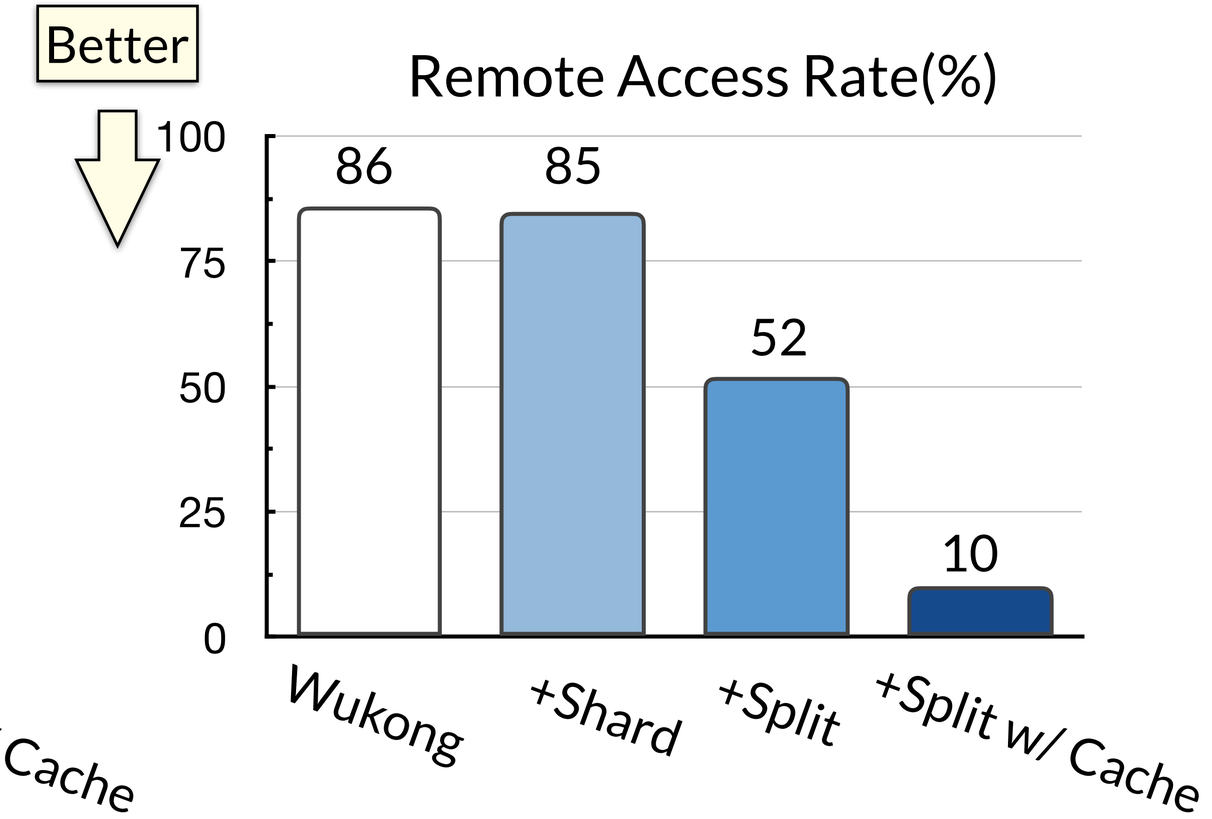


Application: Wukong(OSDI'16)

A graph store processing SQL-like queries

Dataset	#kv
LUBM-10240	1.6 B

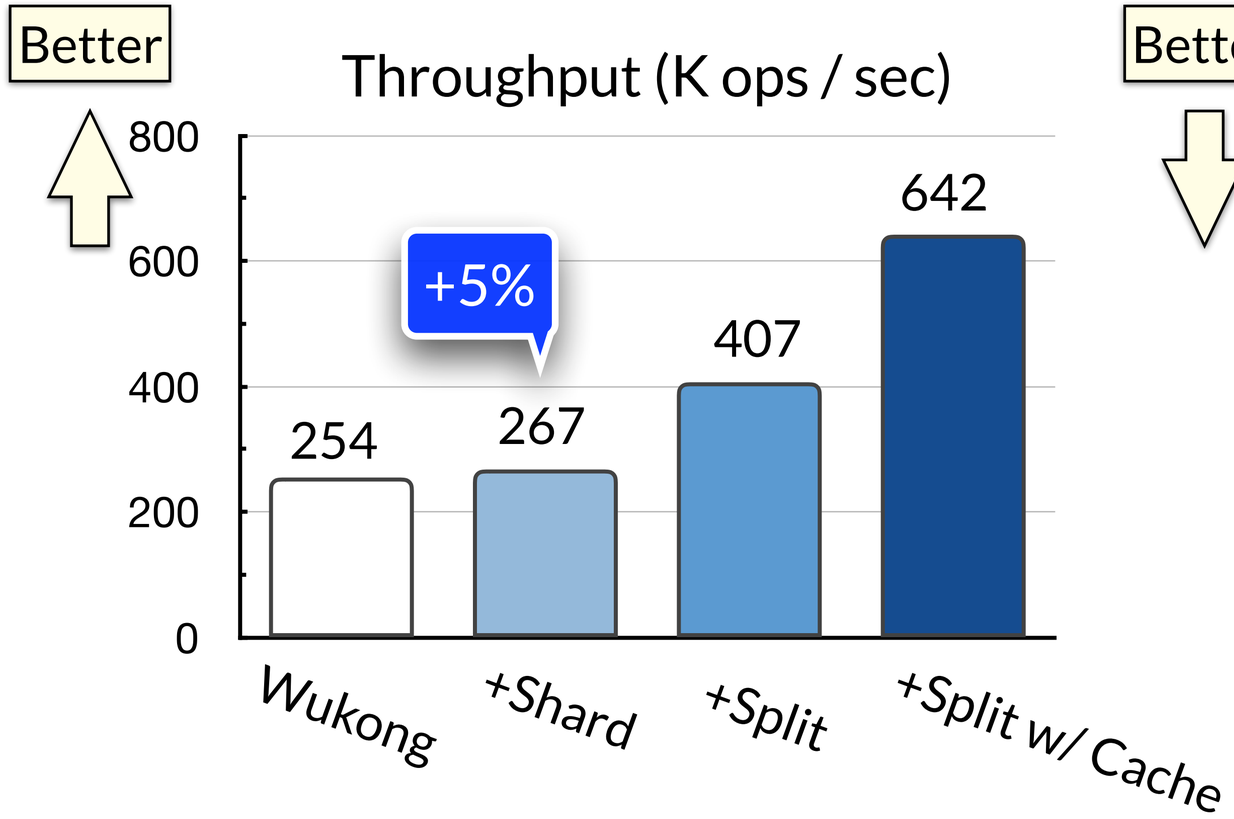


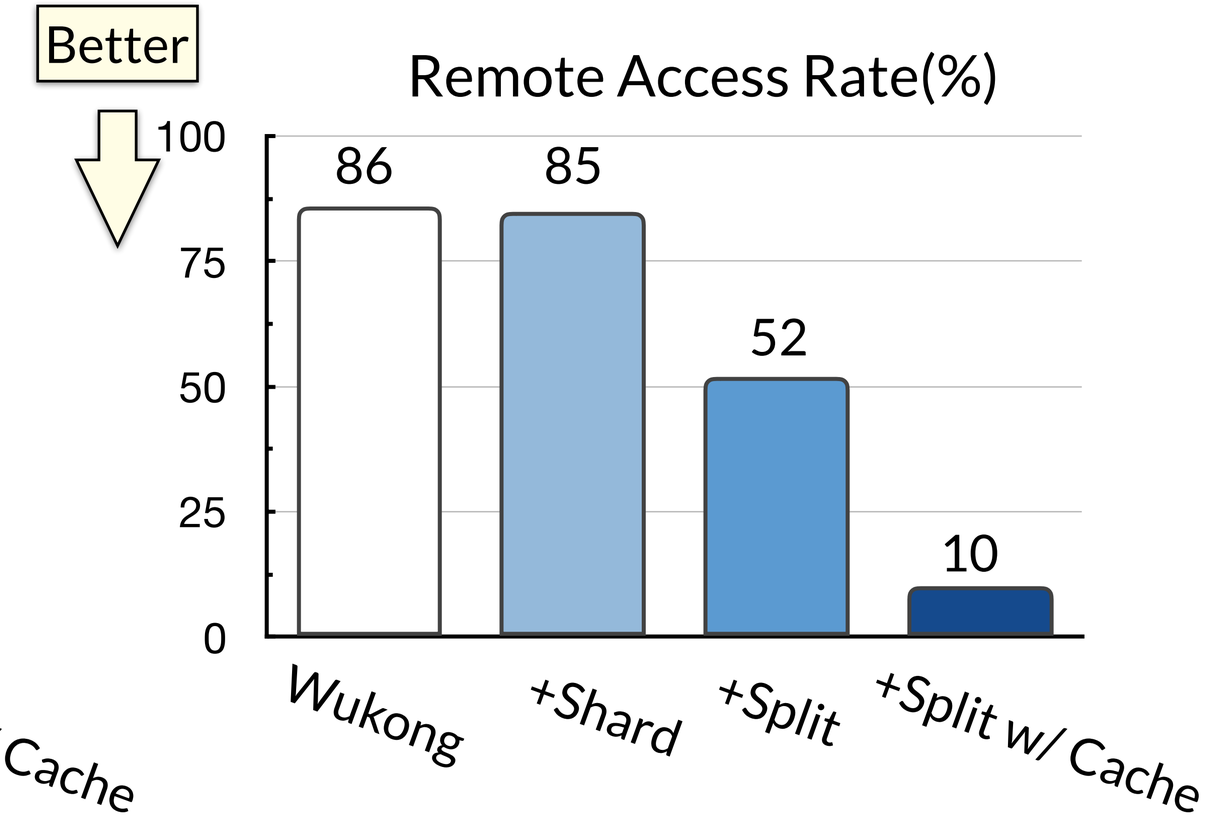


Application: Wukong(OSDI'16)

A graph store processing SQL-like queries

Dataset	#kv
LUBM-10240	1.6 B

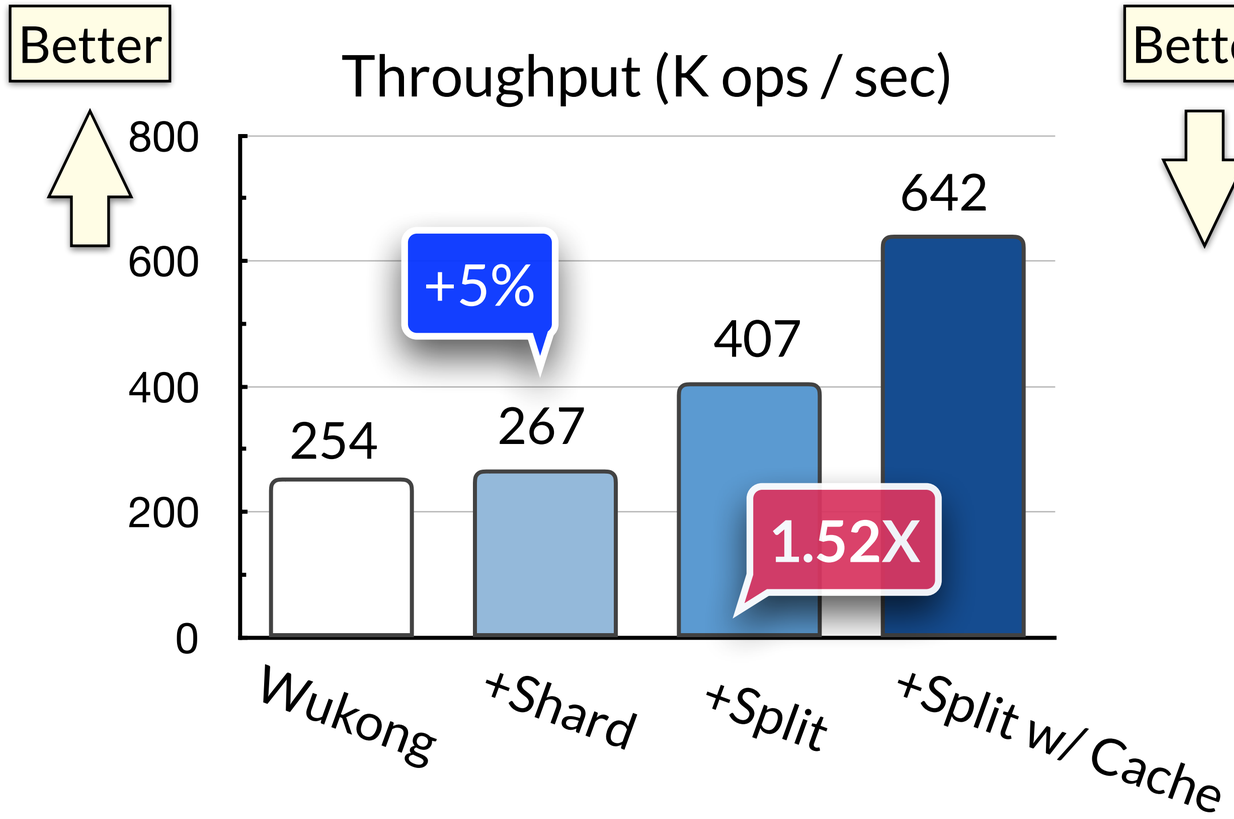


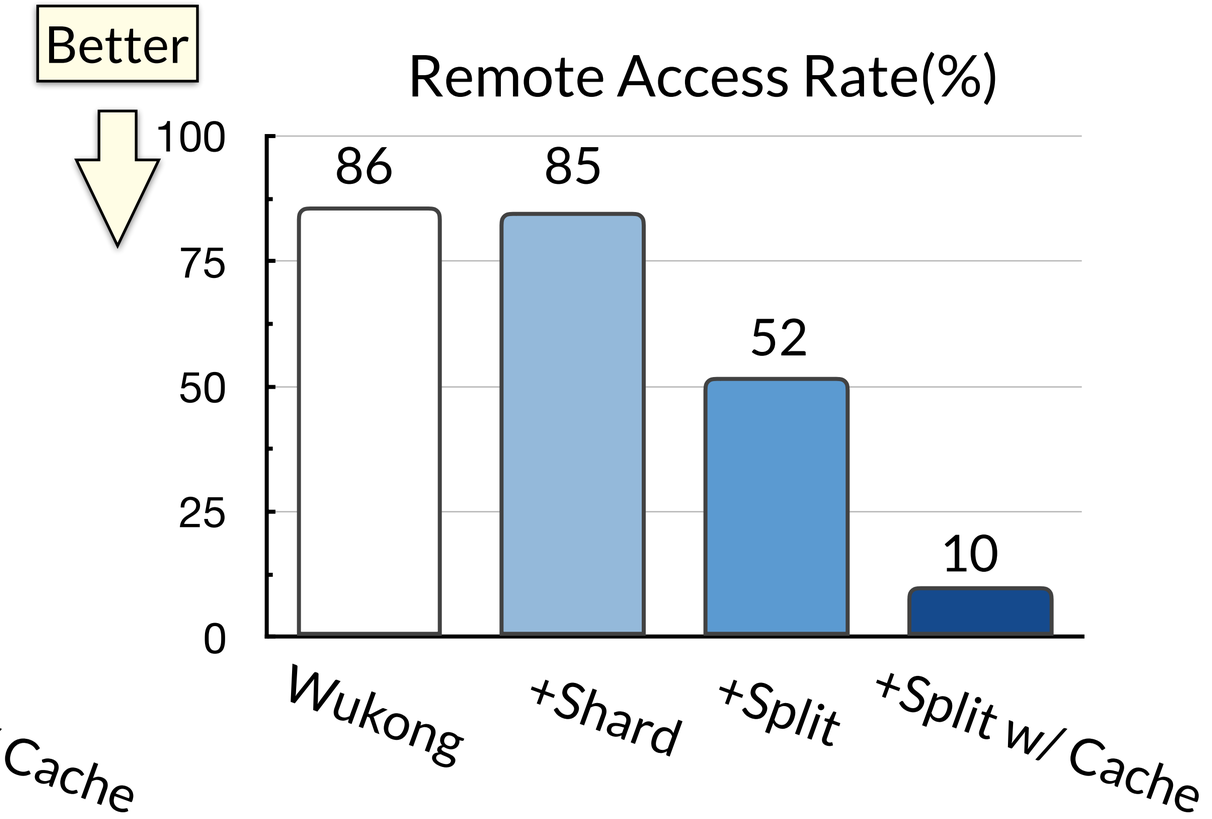


Application: Wukong(OSDI'16)

A graph store processing SQL-like queries

Dataset	#kv
LUBM-10240	1.6 B

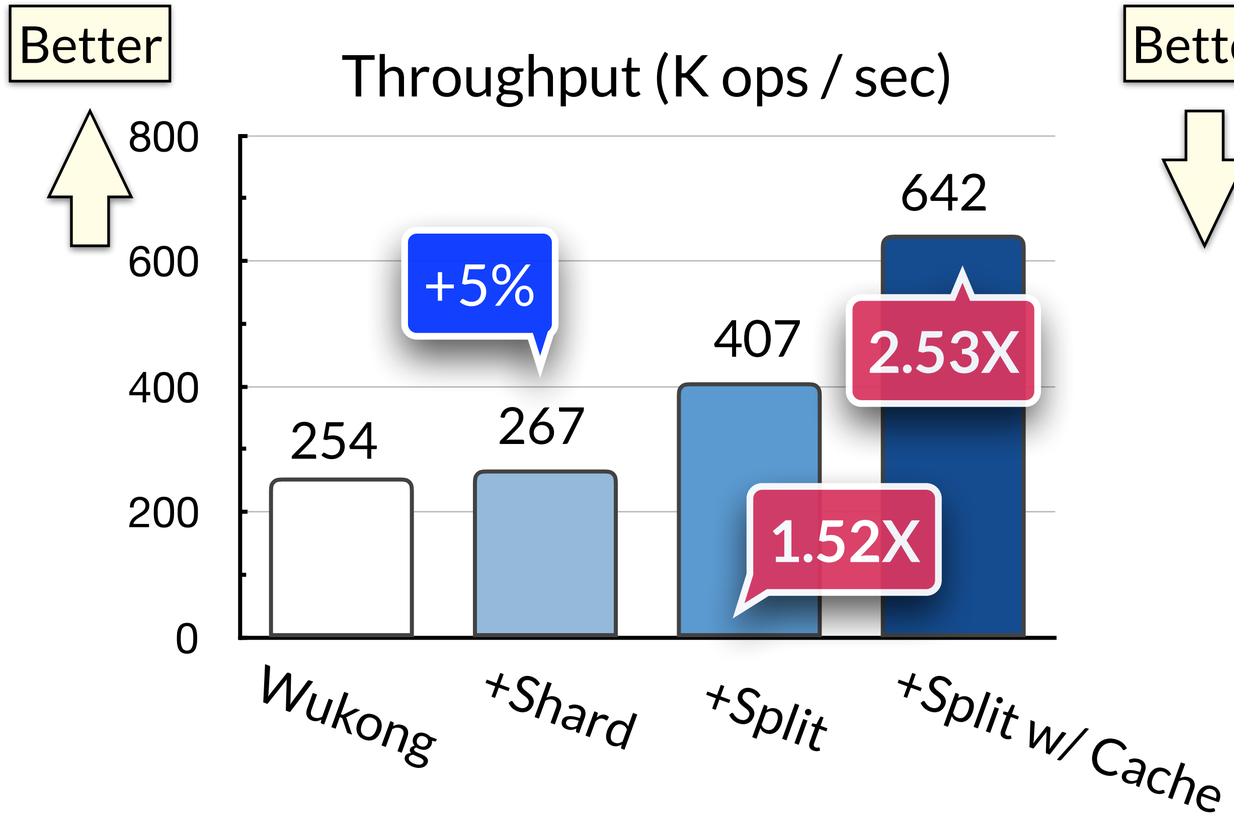


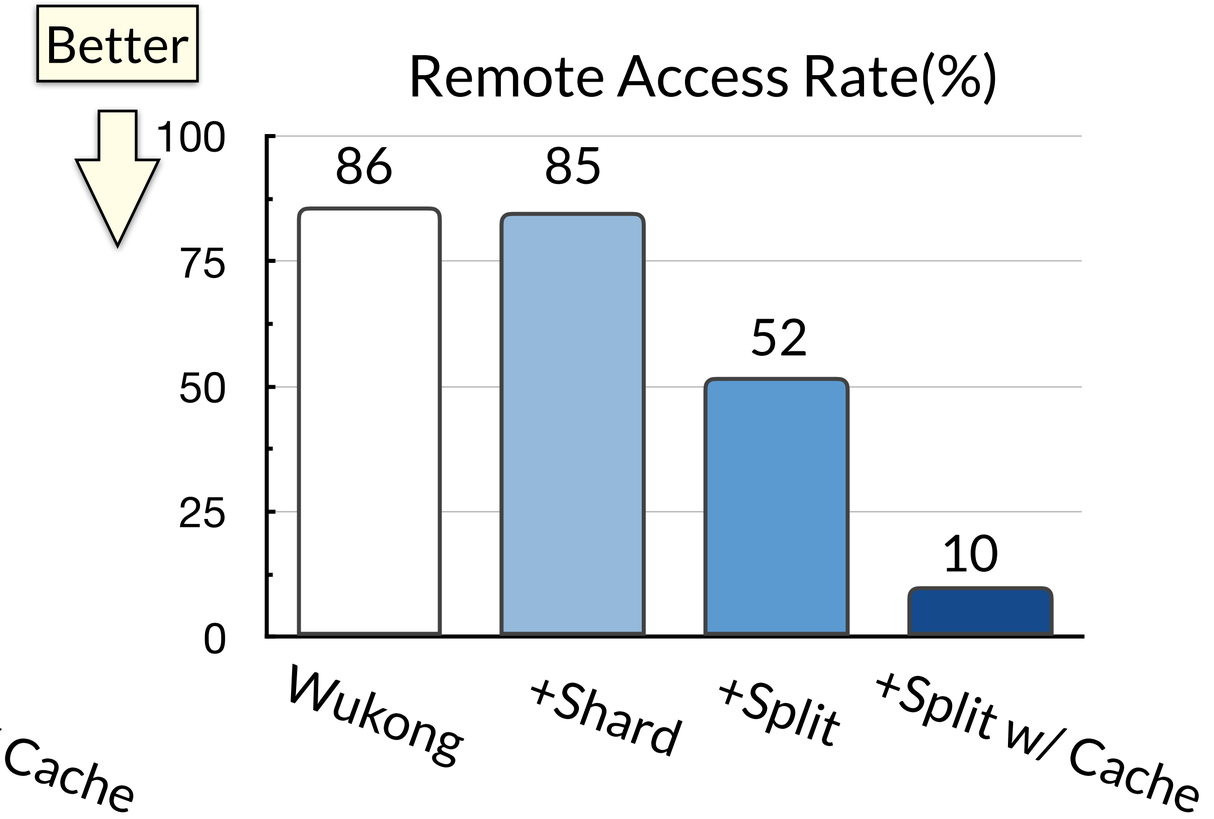


Application: Wukong(OSDI'16)

A graph store processing SQL-like queries

Dataset	#kv
LUBM-10240	1.6 B

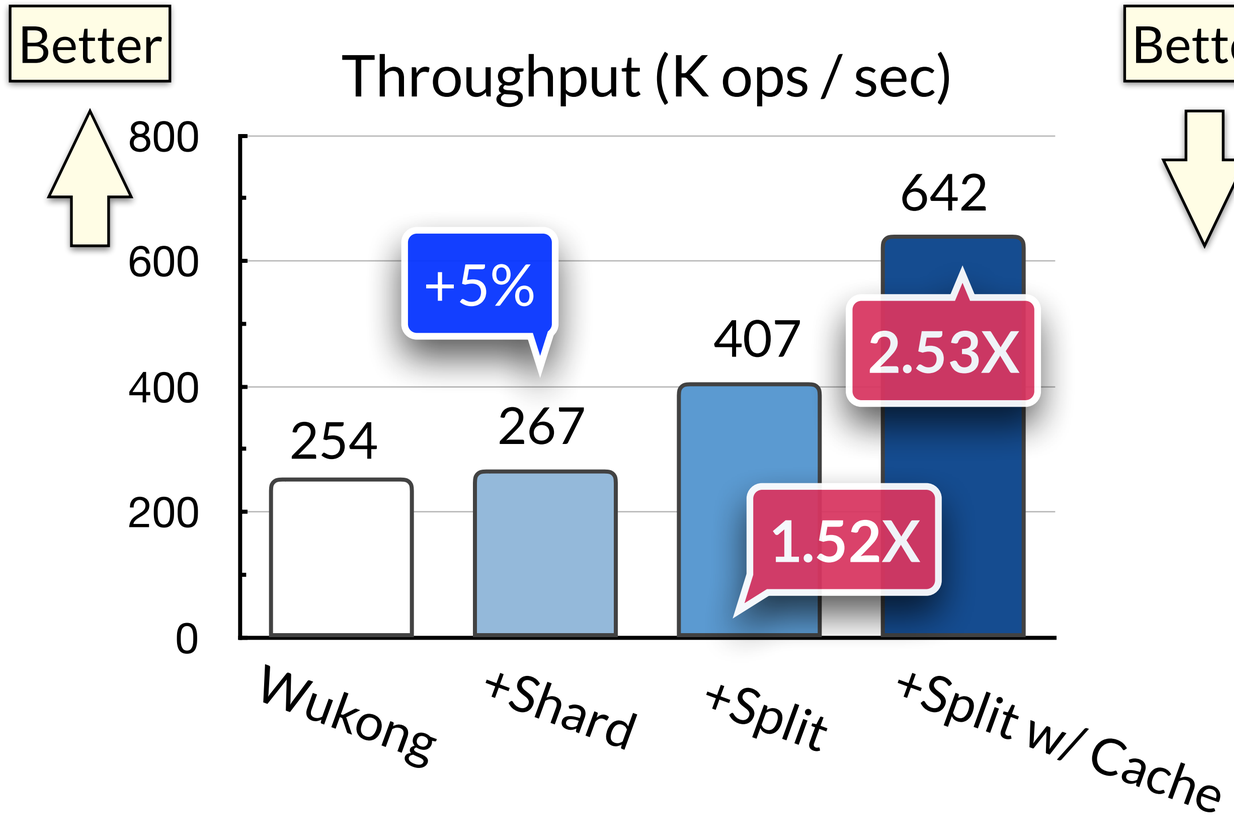


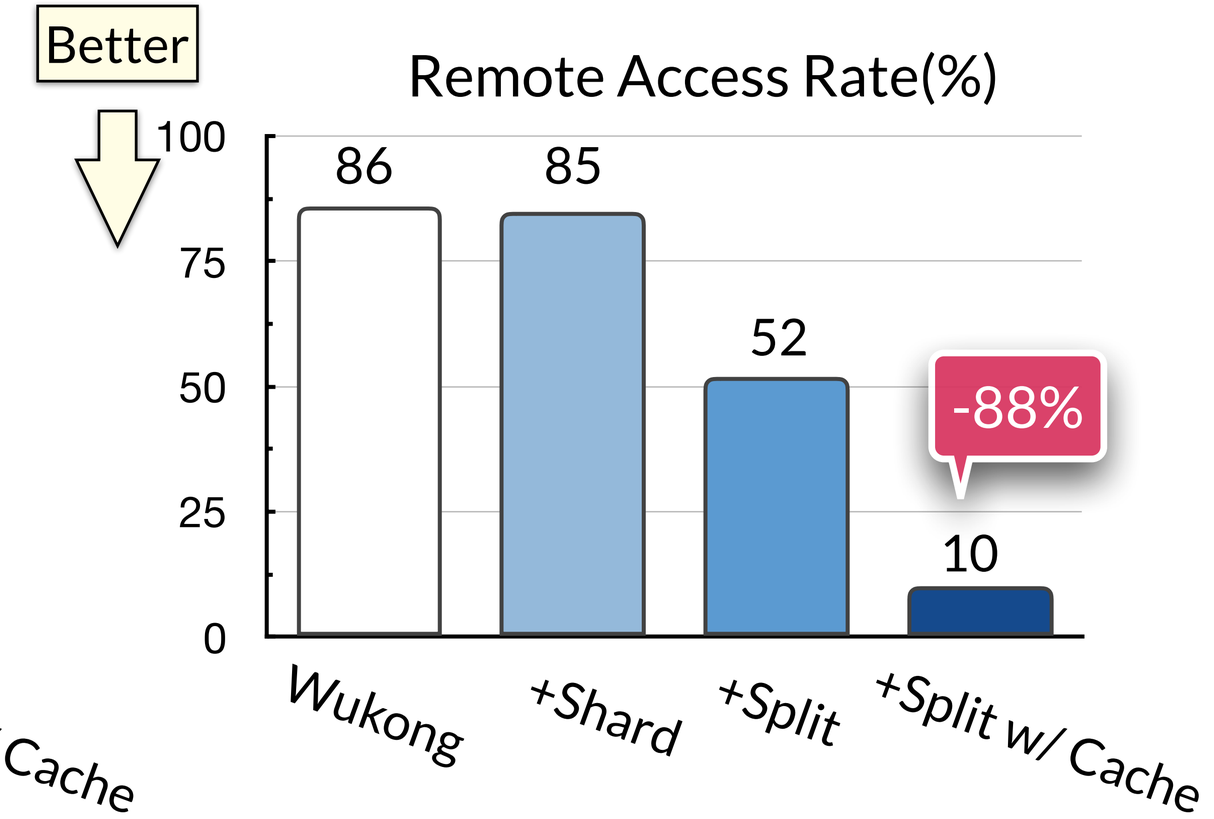


Application: Wukong(OSDI'16)

A graph store processing SQL-like queries

Dataset	#kv
LUBM-10240	1.6 B





Conclusion

Locality is important & challenging for graph traversal

Traditional live migration: Shard-based **does not fit** graph

Pragh uses ***split live migration***

- Zero-metadata, fine-grained migration
- Micro-benchmark: **19X**, Application: **2.53X**
- Included in Wukong

https://ipads.se.sjtu.edu.cn/projects/wukong

Conclusion

Locality is important & challenging for graph traversal

Traditional live migration: Shard-based **does not fit** graph

Pragh uses ***split live migration***

- Zero-metadata, fine-grained migration
- Micro-benchmark: **19X**, Application: **2.53X**
- Included in Wukong

Thanks & Questions?

https://ipads.se.sjtu.edu.cn/projects/wukong

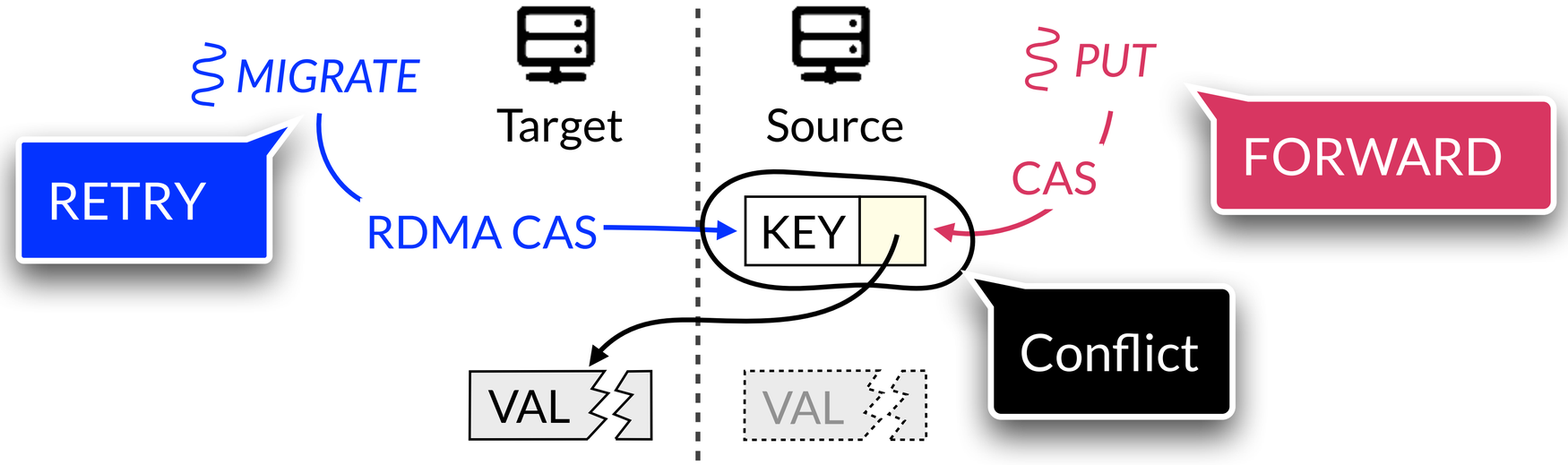
Backup

Check-and-Forward Mechanism: Evolving Graph

Vertex: not migrated ✓

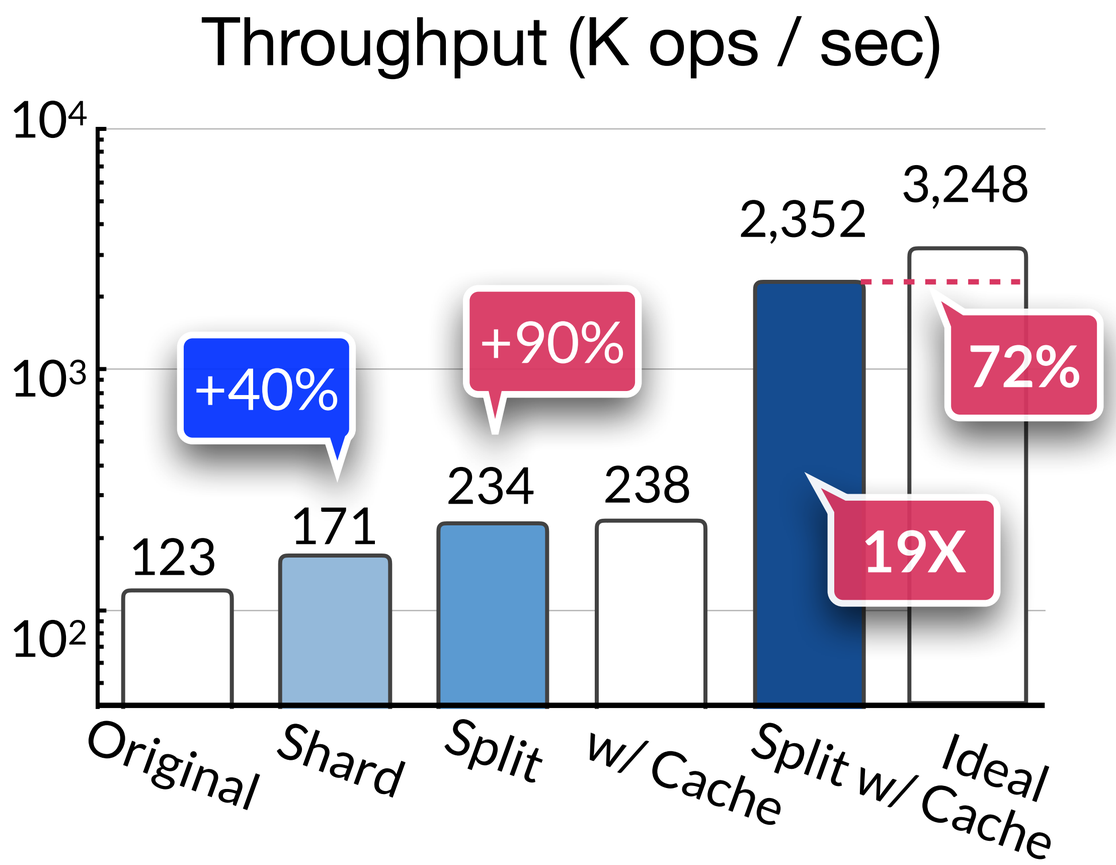
Edge: conflicts among **multi** nodes

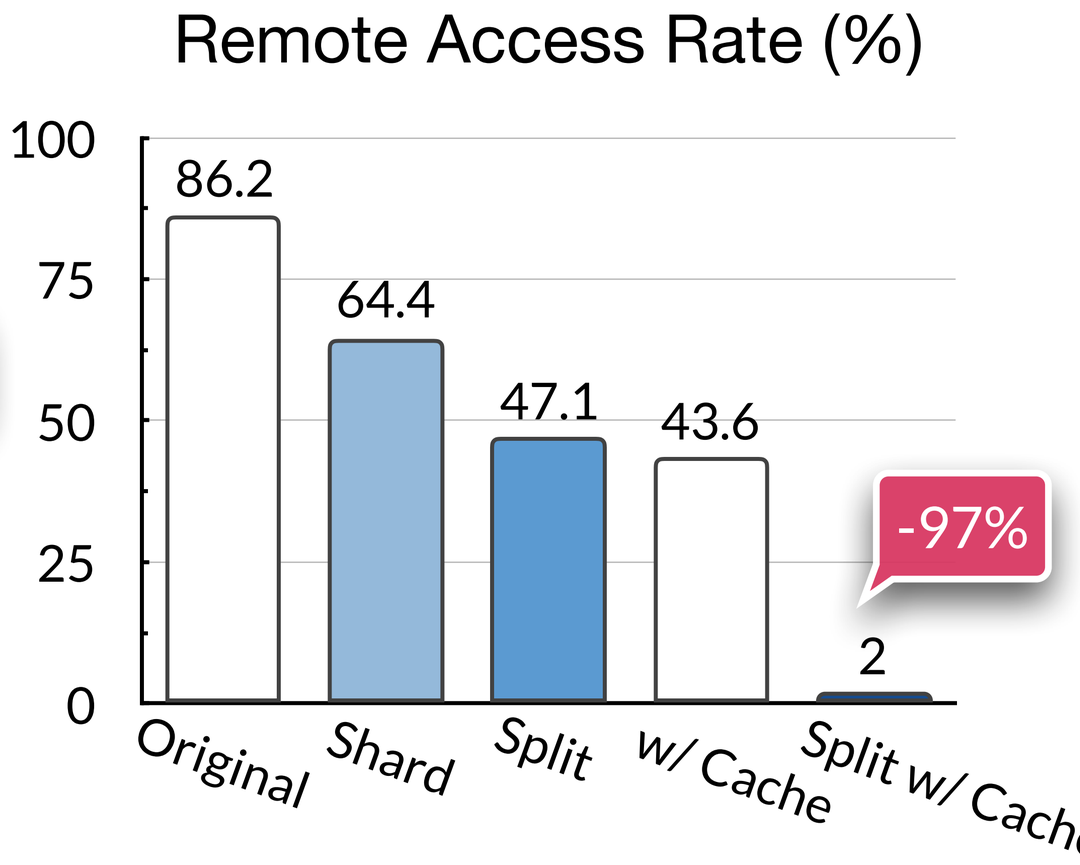
- Detect conflicts in **keys**



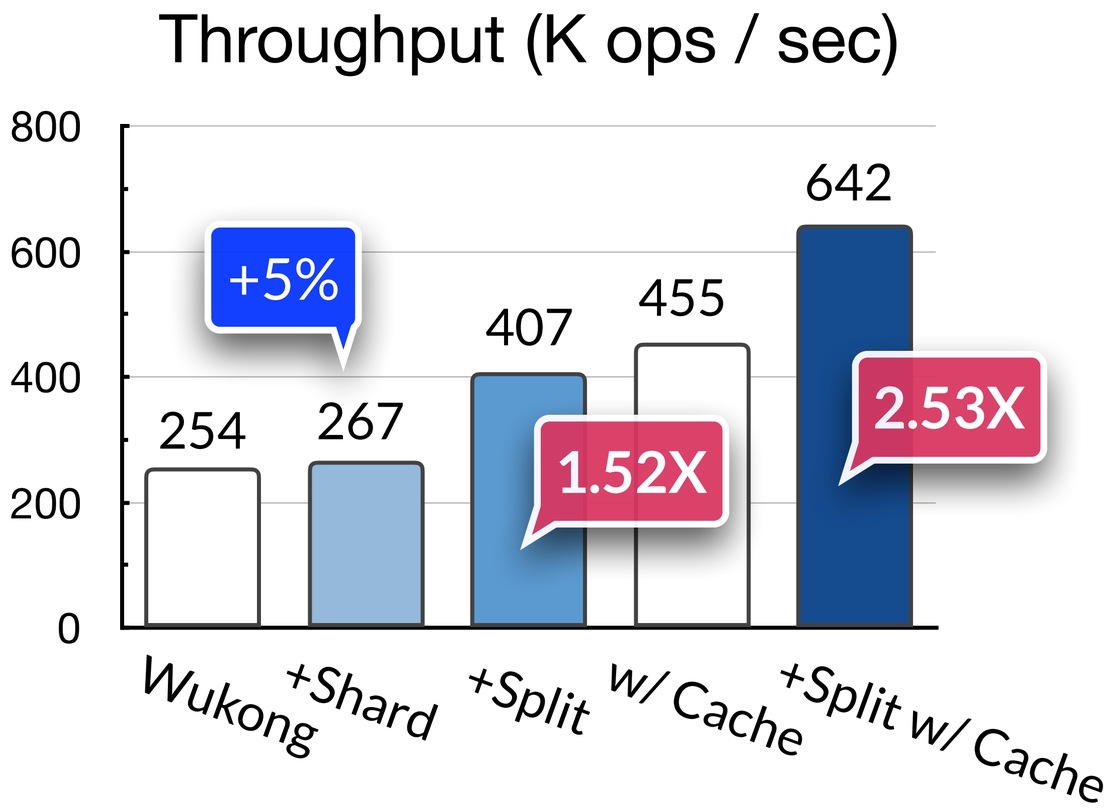
Micro-benchmark

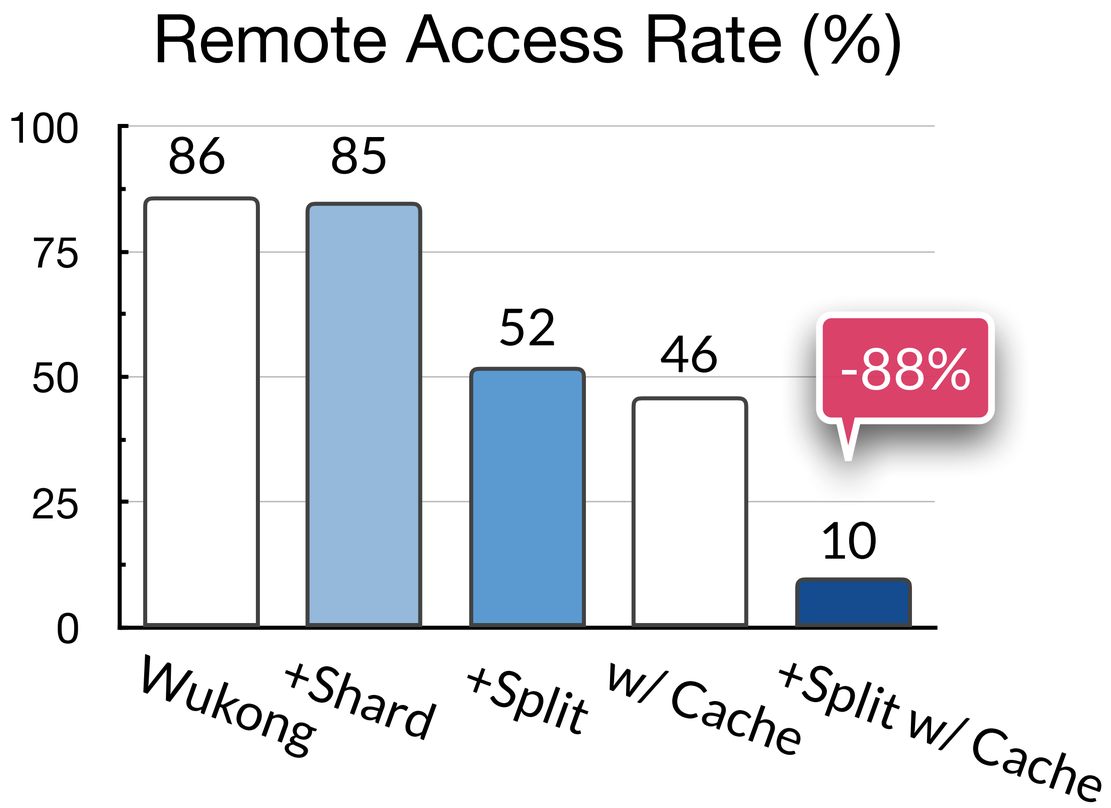
	Shard-based	Split
Migrated data	**85.6%**	**0.13%**



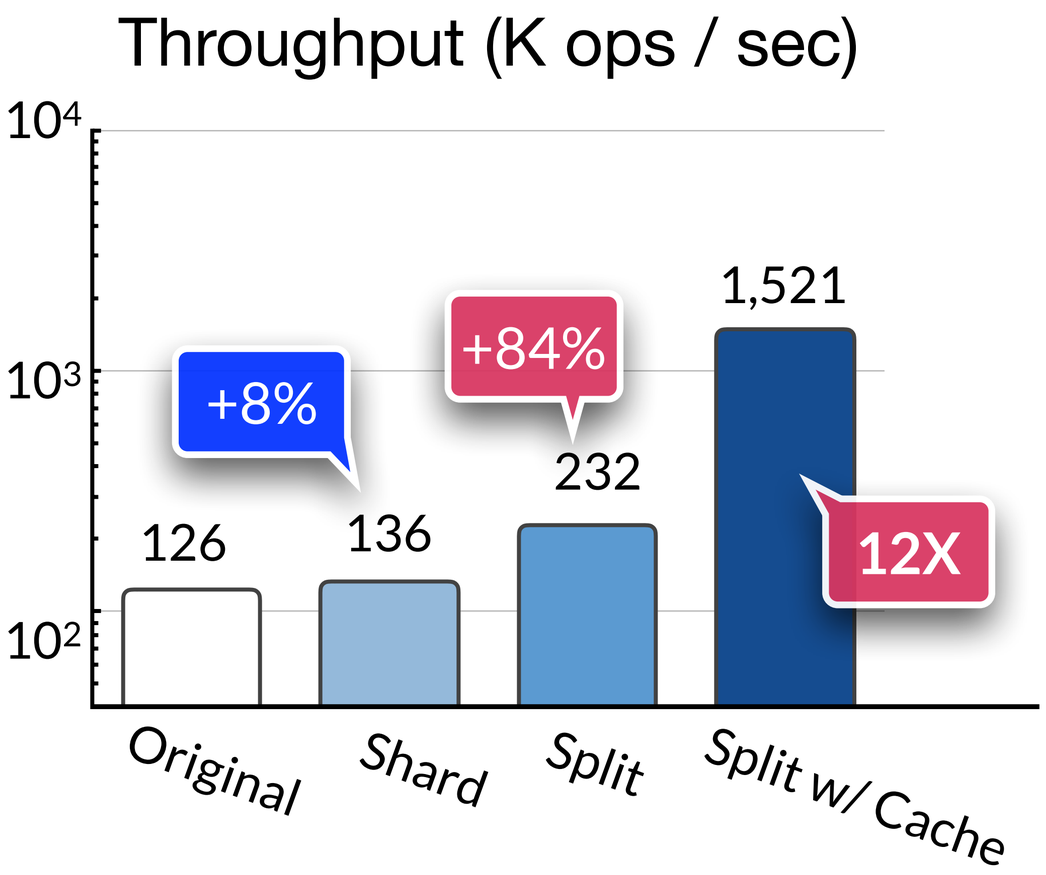


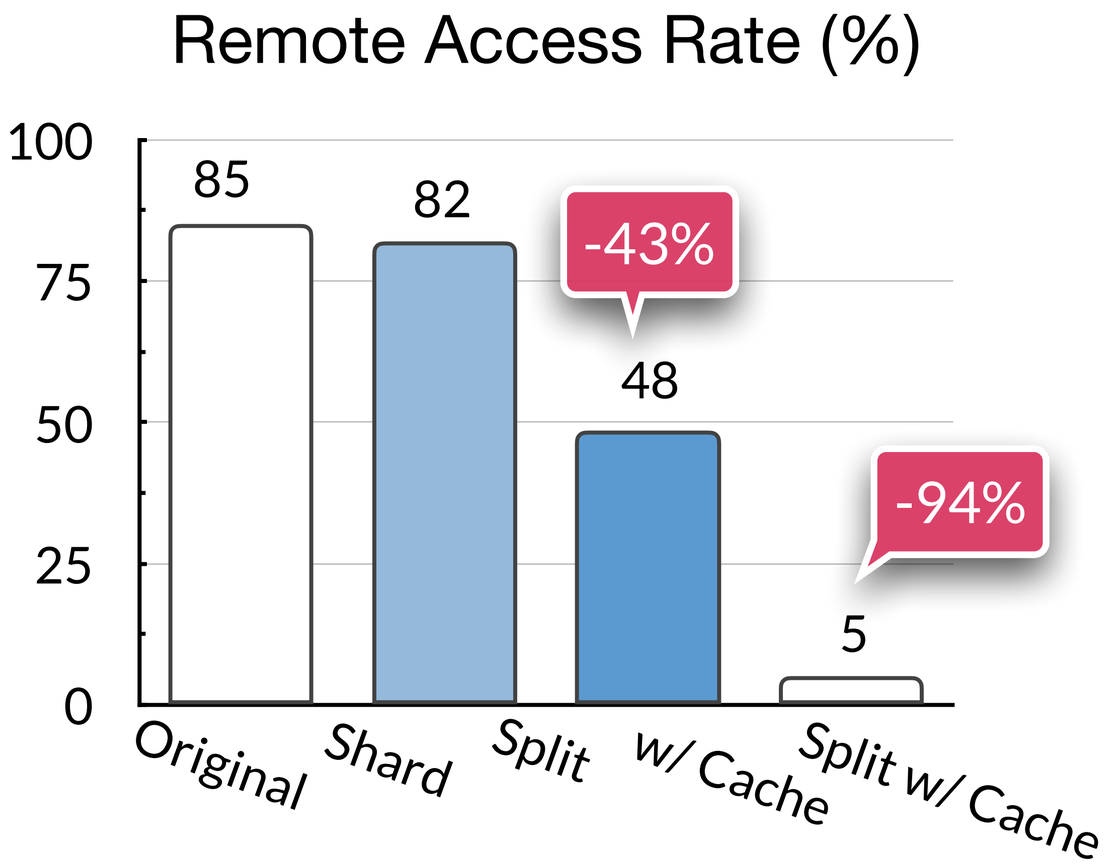
Application: Wukong



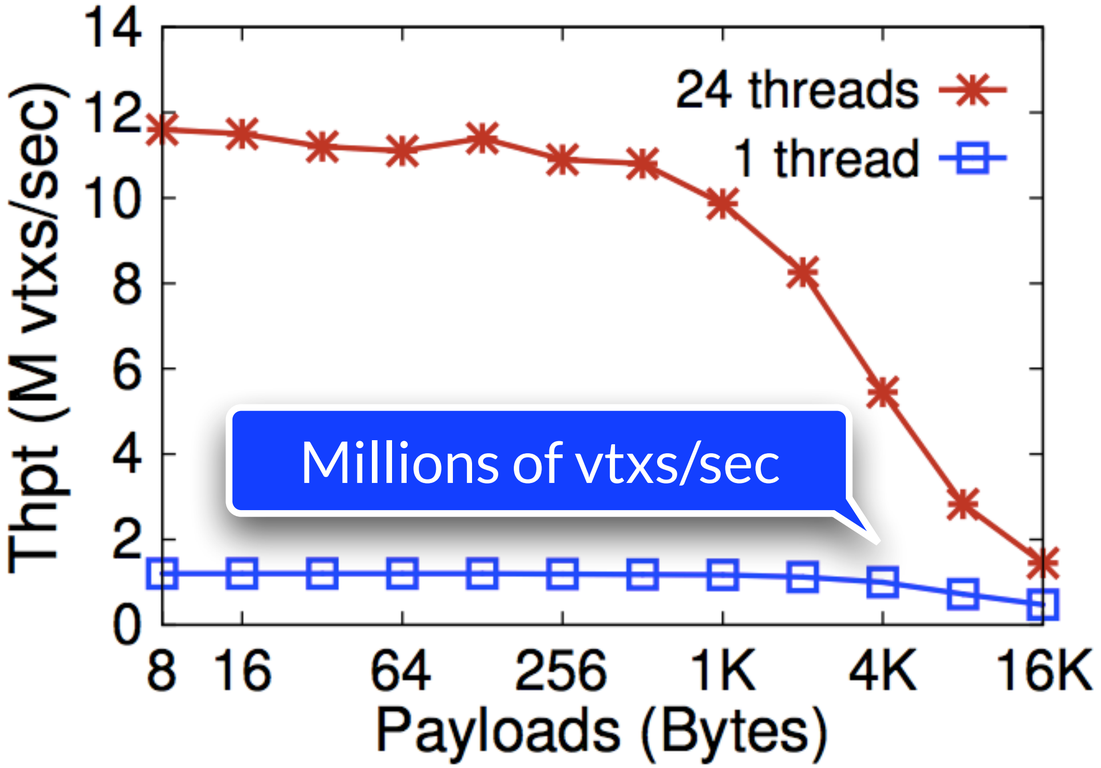


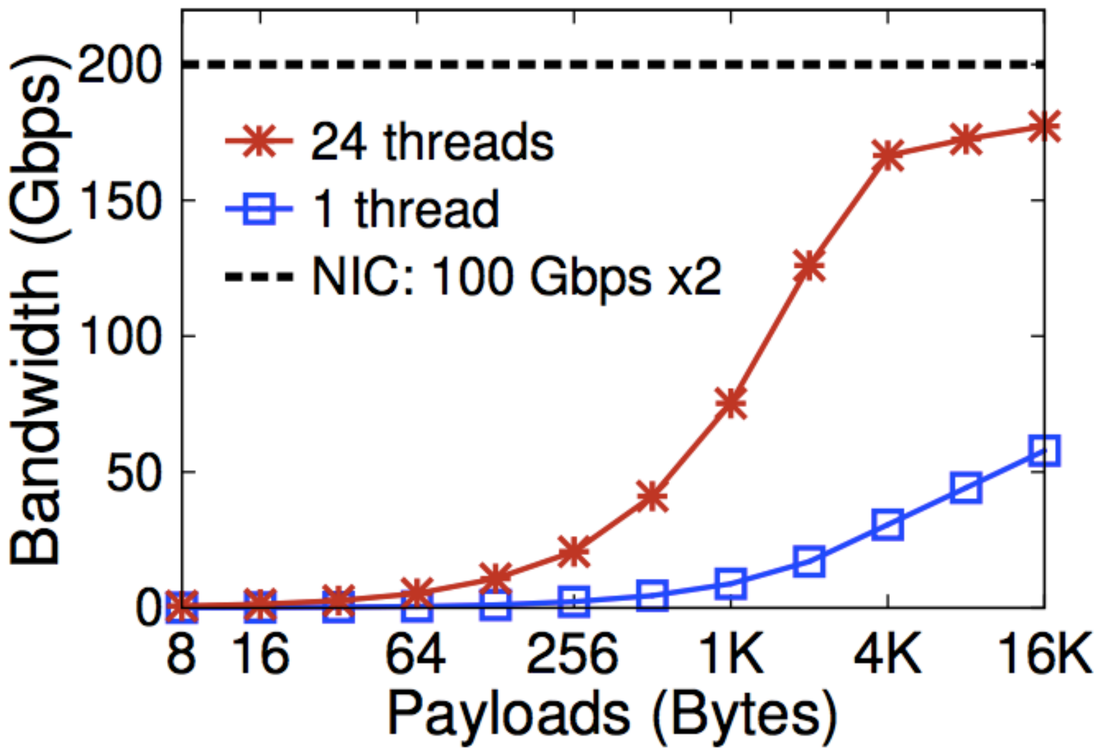
Performance on Uniform Workload



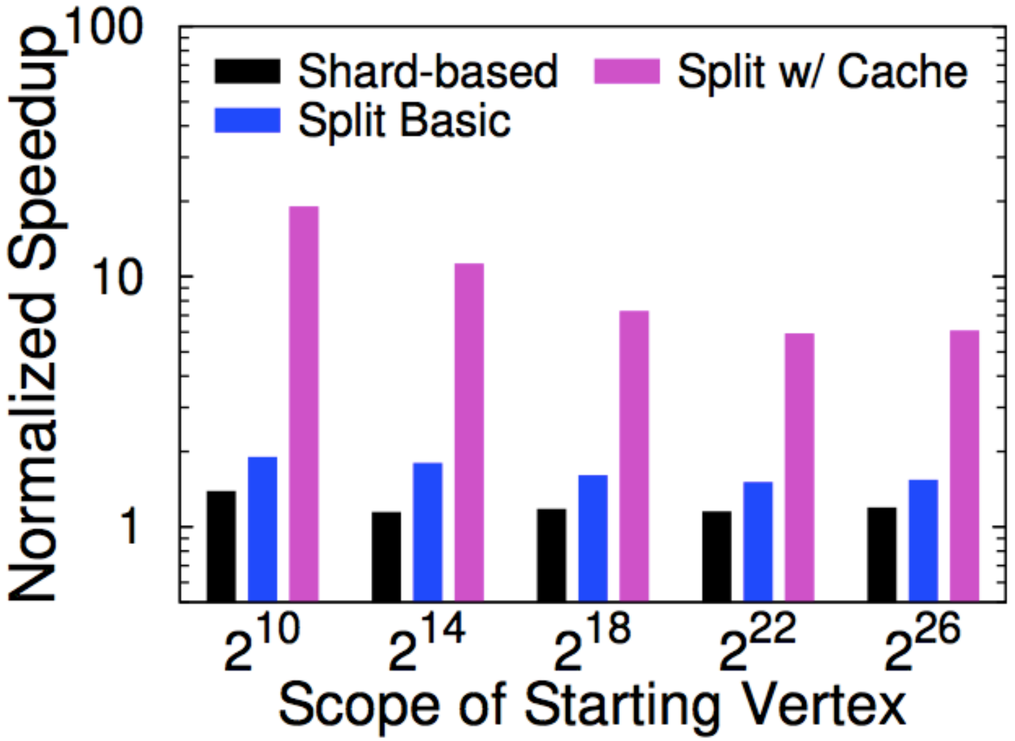


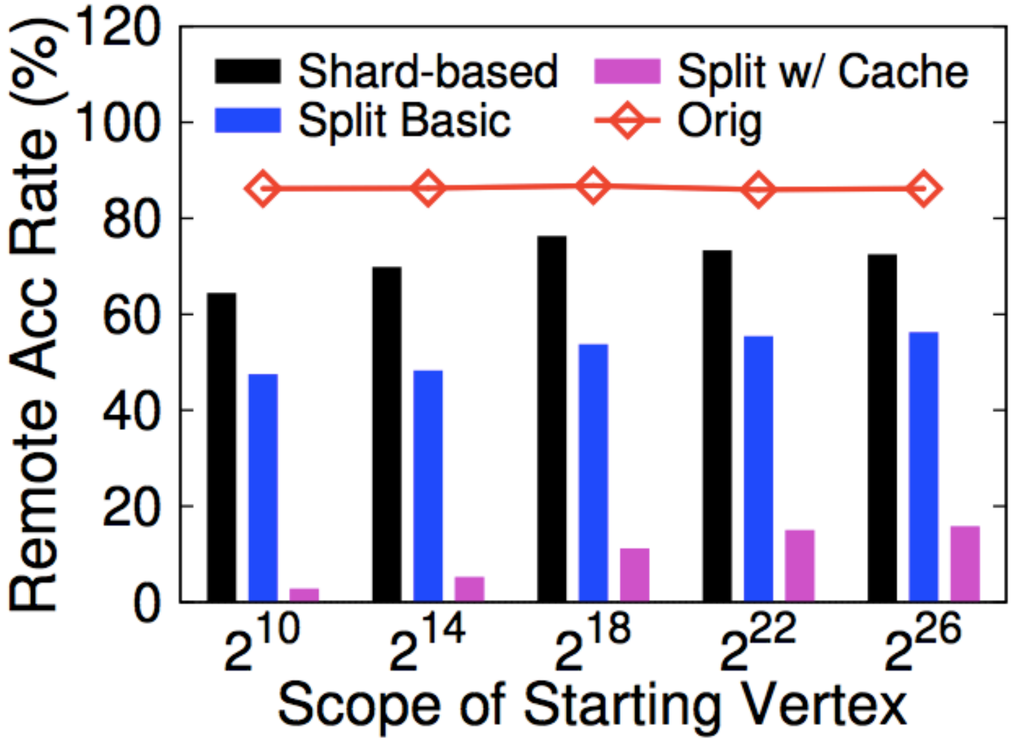
Migration Speed





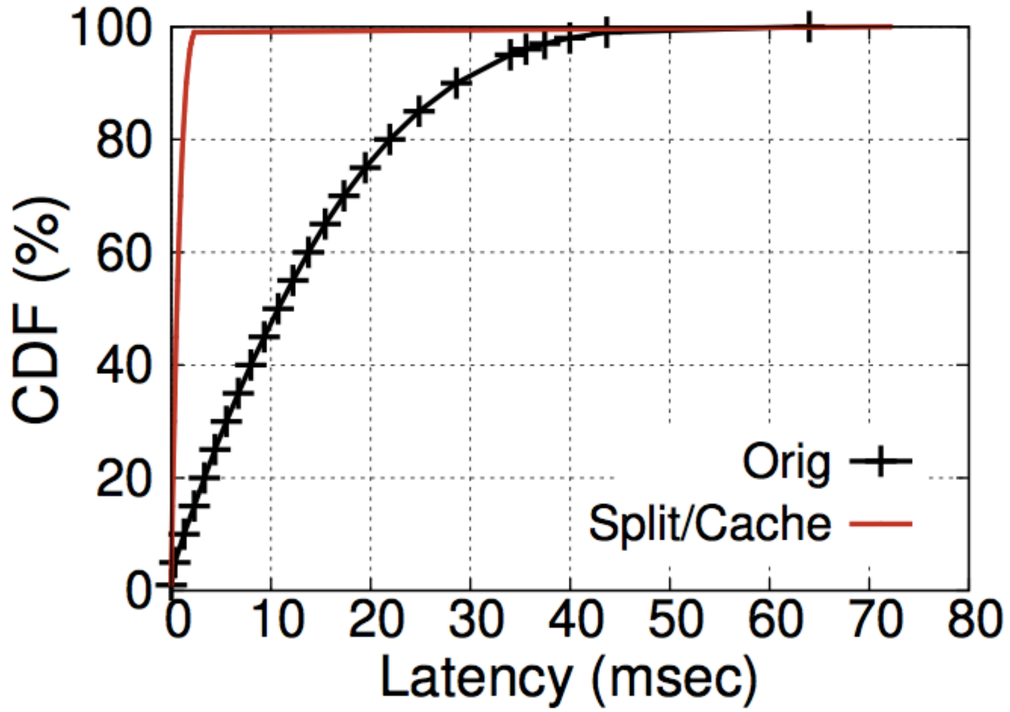
Scope of Starting Vertex

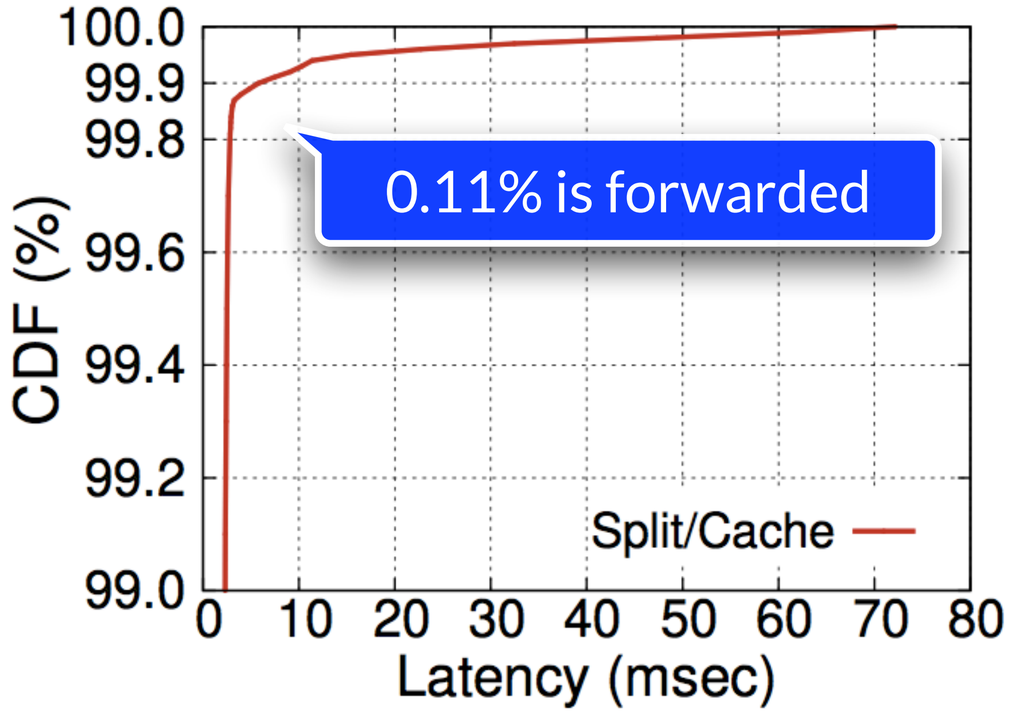




Impact on ***PUT*** operations

50% two-hop queries, 50% edge update/insert





Performance on Dynamic Workload