FAST'22

# InfiniFS: An Efficient Metadata Service for Large-Scale Distributed Filesystems

Wenhao Lv, Youyou Lu, Yiming Zhang, Peile Duan, Jiwu Shu

*Tsinghua University* *Xiamen University* *Alibaba Group*



# Outline

- Background & Motivation
- Design
- Evaluation
- Conclusion

# Large-Scale Distributed Filesystem

- **Modern datacenters contain a huge number of files**
  - Facebook: billions of files (Tectonic [FAST '21])
  - Alibaba Cloud: tens of billions of files (thousands of Pangu)

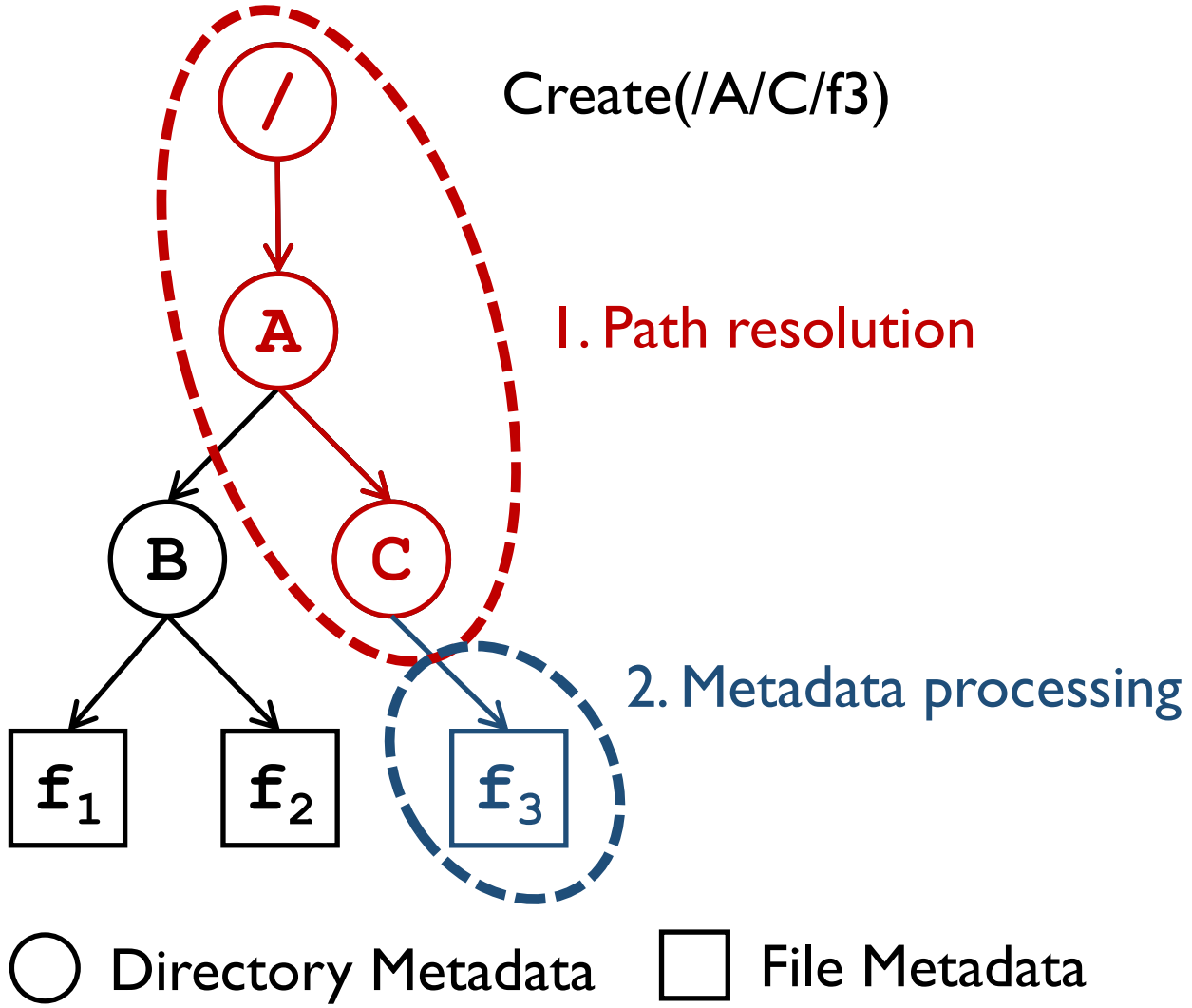
- **One single large-scale filesystem spanning the entire datacenter is desirable**
  - Global data sharing
  - High resource utilization
  - Low operational complexity

- **Managing such huge number of files in one single filesystem brings severe challenges to the metadata service.**

# Filesystem Metadata

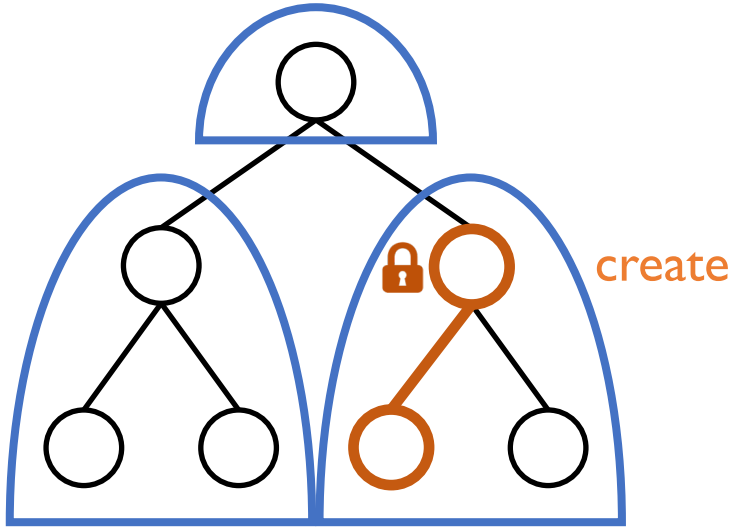
- **Filesystem directory tree**
  - Hierarchical namespace
  - Directory metadata
  - File metadata

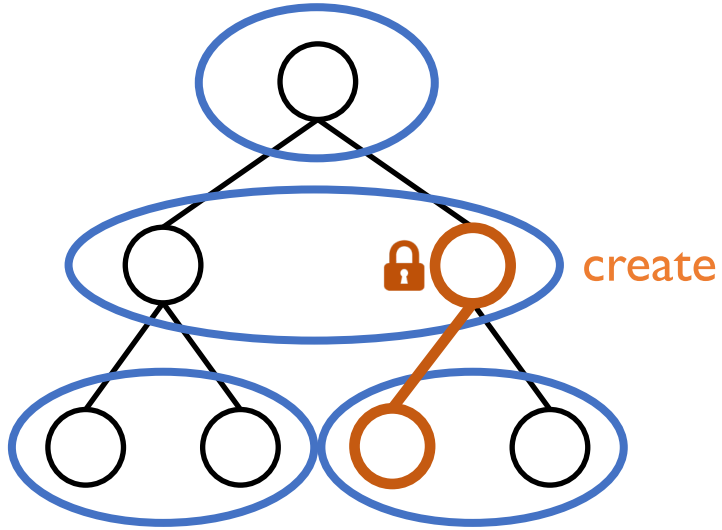
- **Metadata operation**
  1. Path resolution
  2. Metadata processing



# Challenges of efficient Metadata Service

## 1. Partitioning of the directory tree

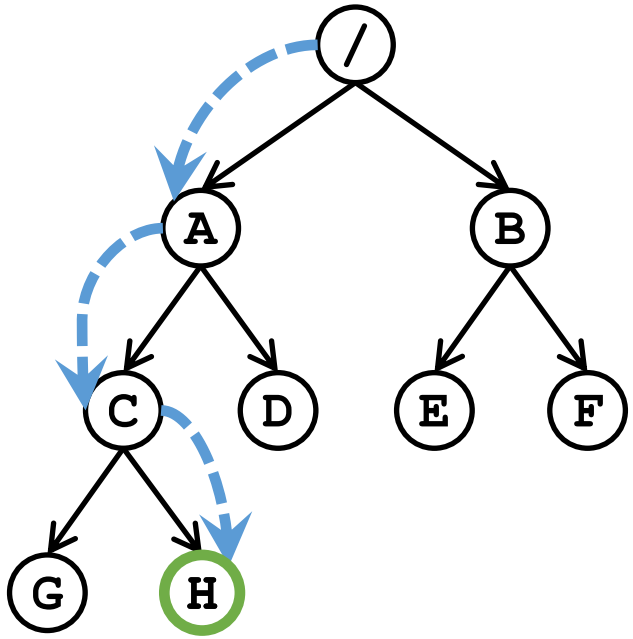


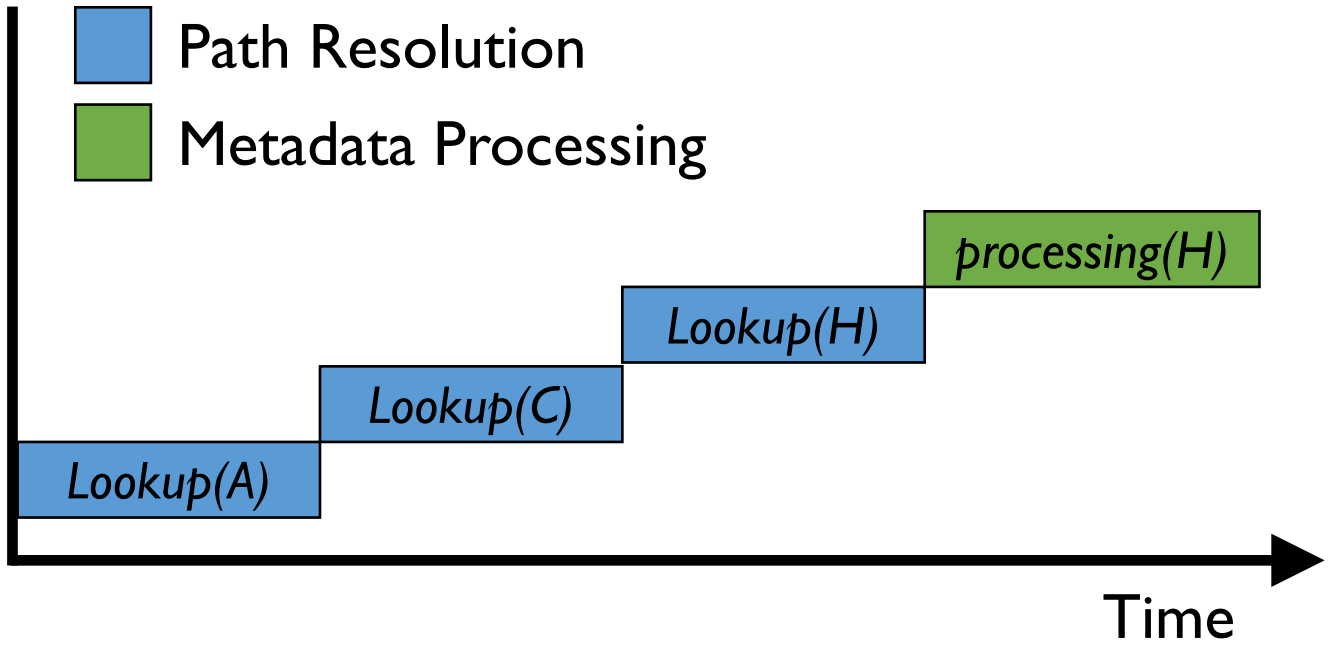




| | Coarse-grained Partitioning | Fine-grained Partitioning |
|---|---|---|
| Metadata Locality | ☺ | ☹ |
| Load Balancing | ☹ | ☺ |

# Challenges of efficient Metadata Service

## 2. High latency of path resolution



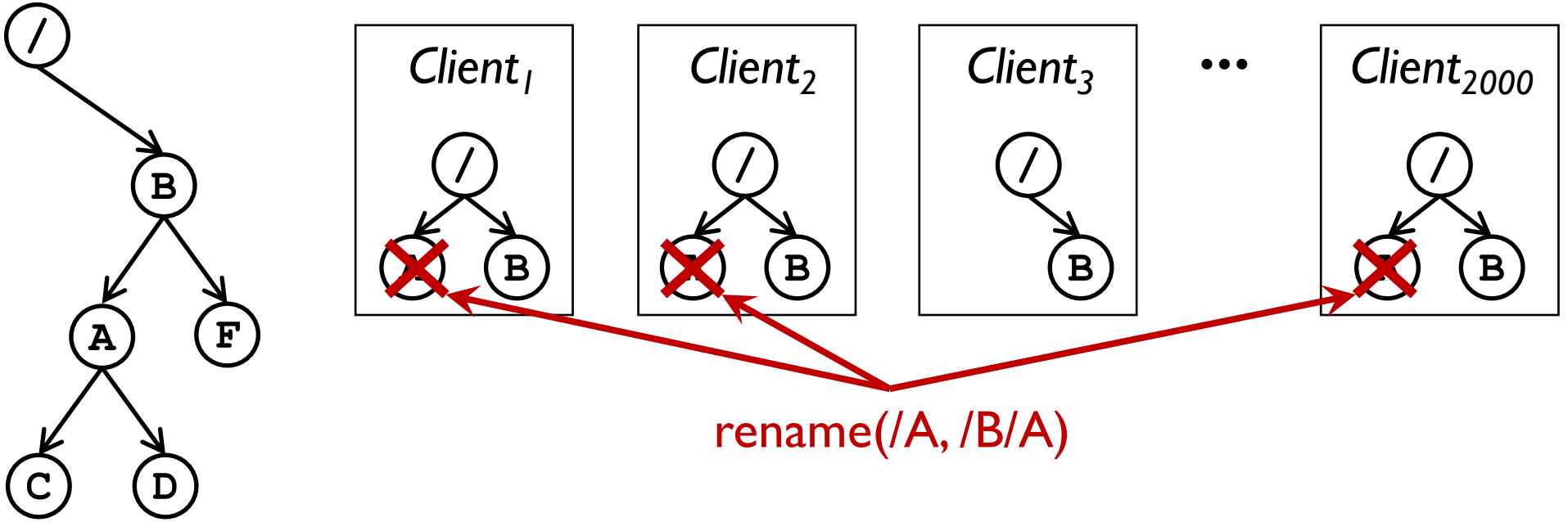

High file depth → High latency

# Challenges of efficient Metadata Service

## 3. High overhead of cache coherence maintenance


Near-root hotspots caused by the path resolution

Cache metadata on the client-side




Huge number of Clients → High coherence overhead

# Outline

- Background & Motivation
- Design
- Evaluation
- Conclusion

# InfiniFS Architecture

## An efficient metadata service

### Clients

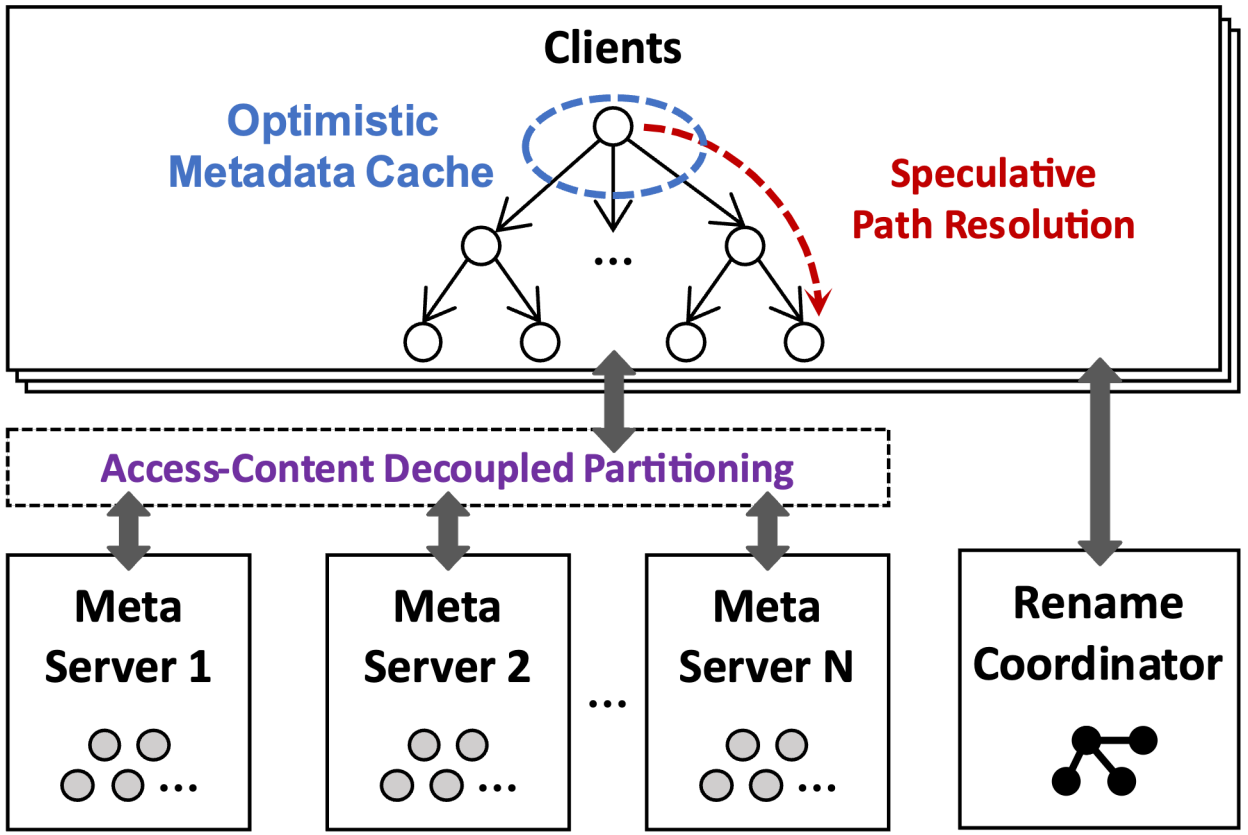
- Speculative path resolution
- Optimistic metadata cache

### Metadata Servers

- Access-content decoupled partitioning

### Rename Coordinator

- Check concurrent directory renames



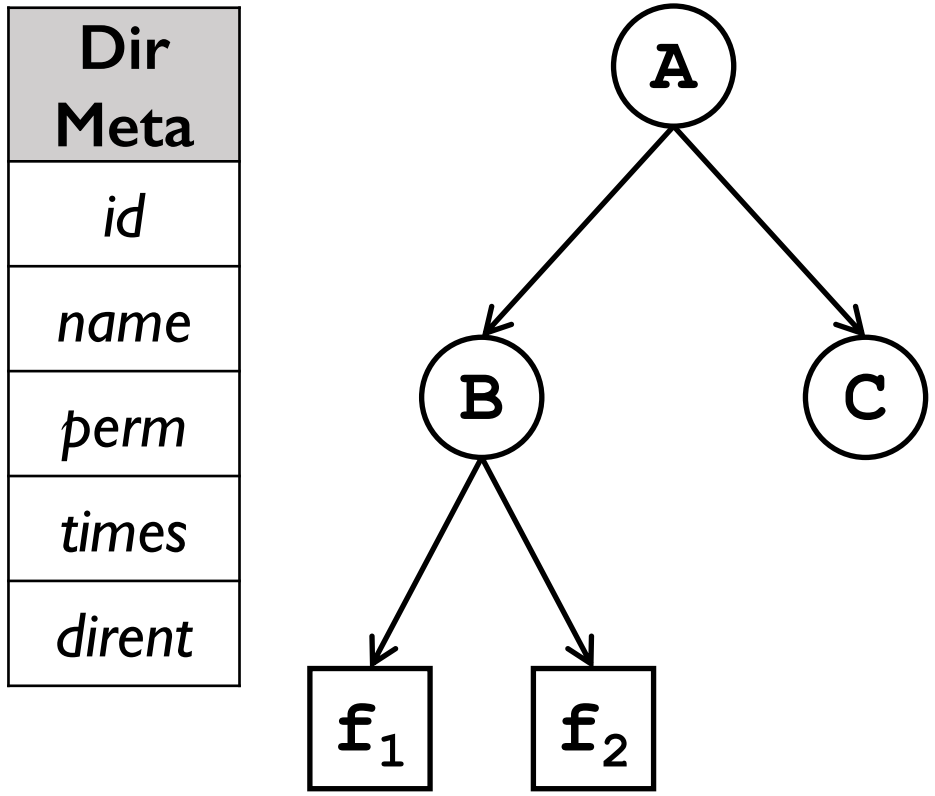
# Key Designs

- Partitioning of the directory tree
  1. Access-Content Decoupled Partitioning

- High latency of path resolution
  2. Speculative Path Resolution

- High overhead of cache coherence maintenance
  3. Optimistic Access Metadata Cache

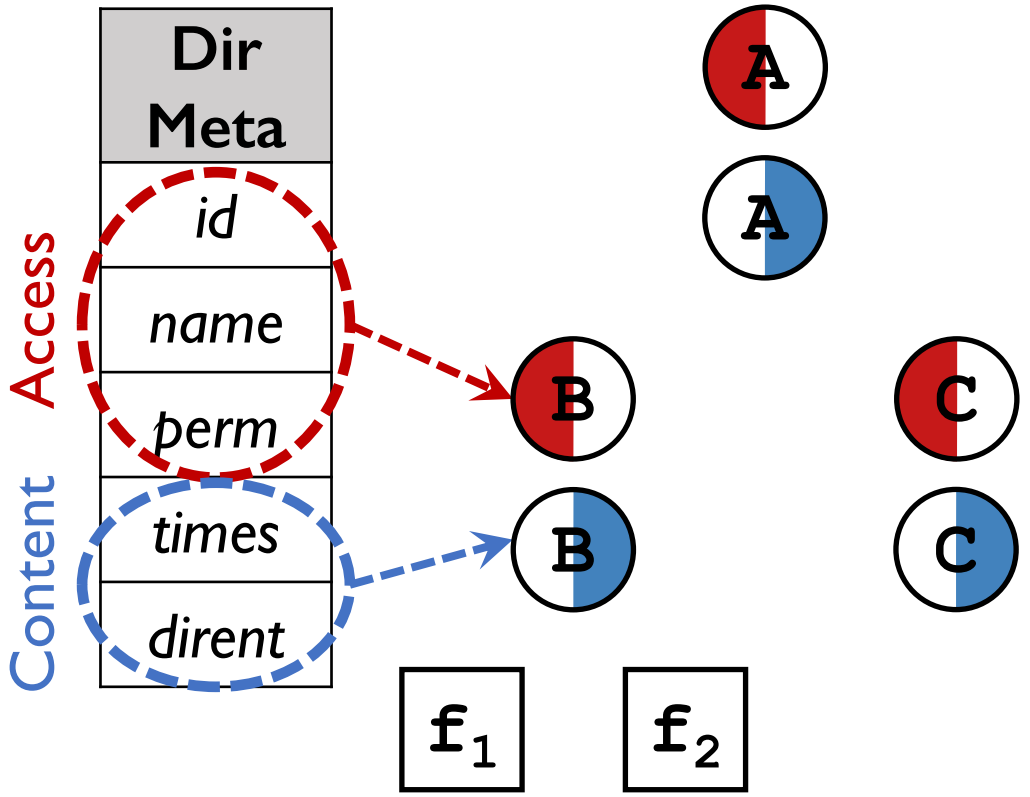
# 1. Access-Content Decoupled Partitioning



# 1. Access-Content Decoupled Partitioning

## Decoupling directory metadata

- Access metadata
  - Related to directory tree accessing
- Content metadata
  - Related to the children



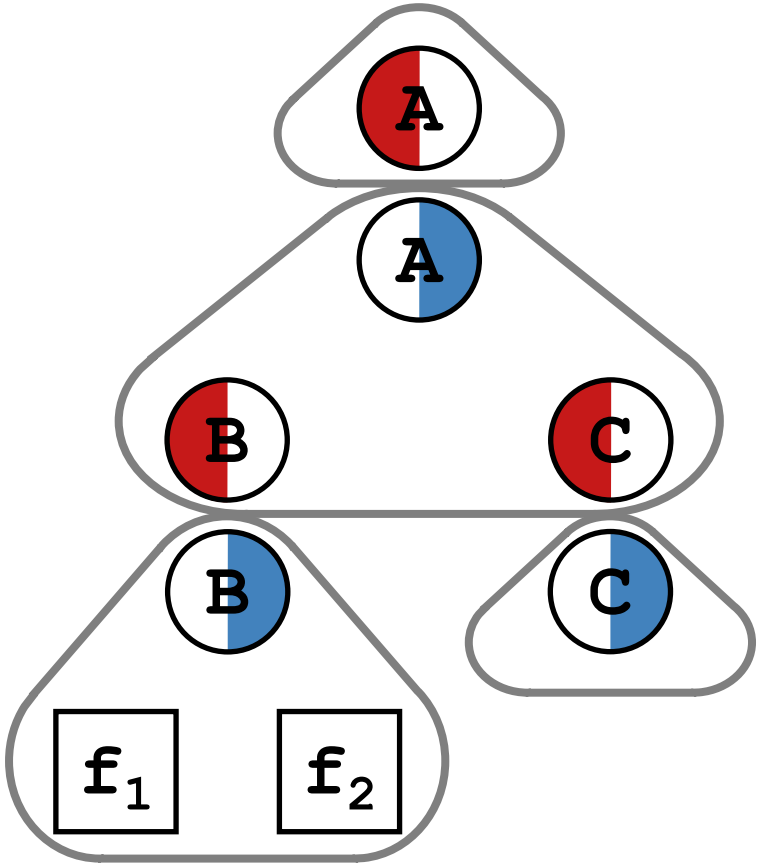
# 1. Access-Content Decoupled Partitioning

## Decoupling directory metadata

- Access metadata
  - Related to directory tree accessing
- Content metadata
  - Related to the children

## Grouping related metadata for locality

- Access metadata with the parent
- Content metadata with the children



# 1. Access-Content Decoupled Partitioning

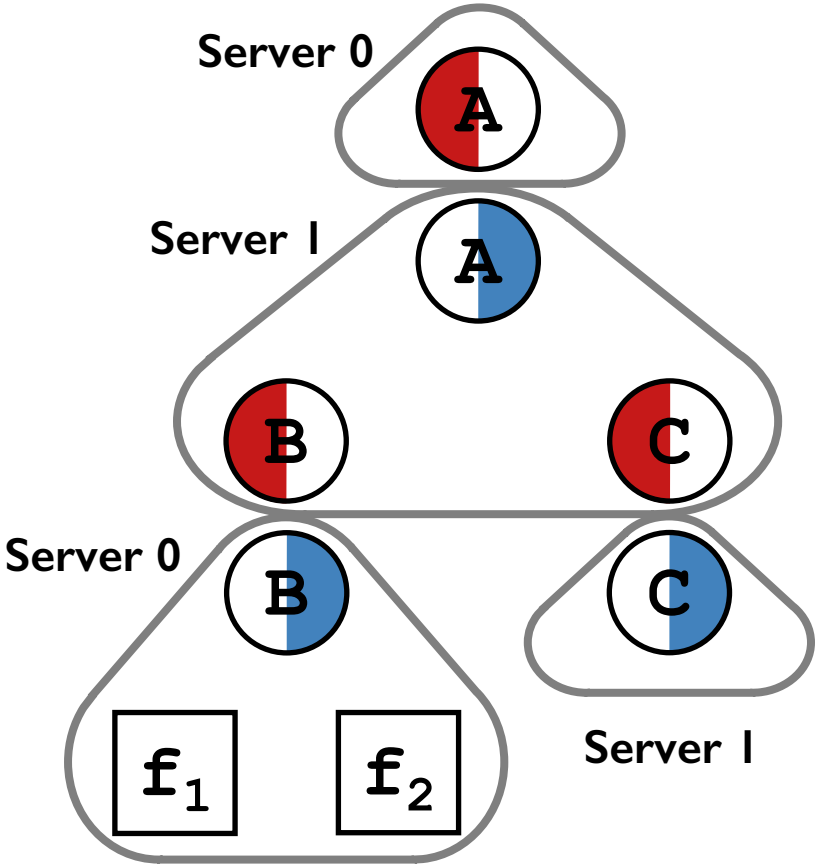
## Decoupling directory metadata

- Access metadata
  - Related to directory tree accessing
- Content metadata
  - Related to the children

## Grouping related metadata for locality

- Access metadata with the parent
- Content metadata with the children

## Hash Partitioning for load balancing



# 1. Access-Content Decoupled Partitioning

Good load balancing and high metadata locality for common operations like file create, delete, and directory readdir.

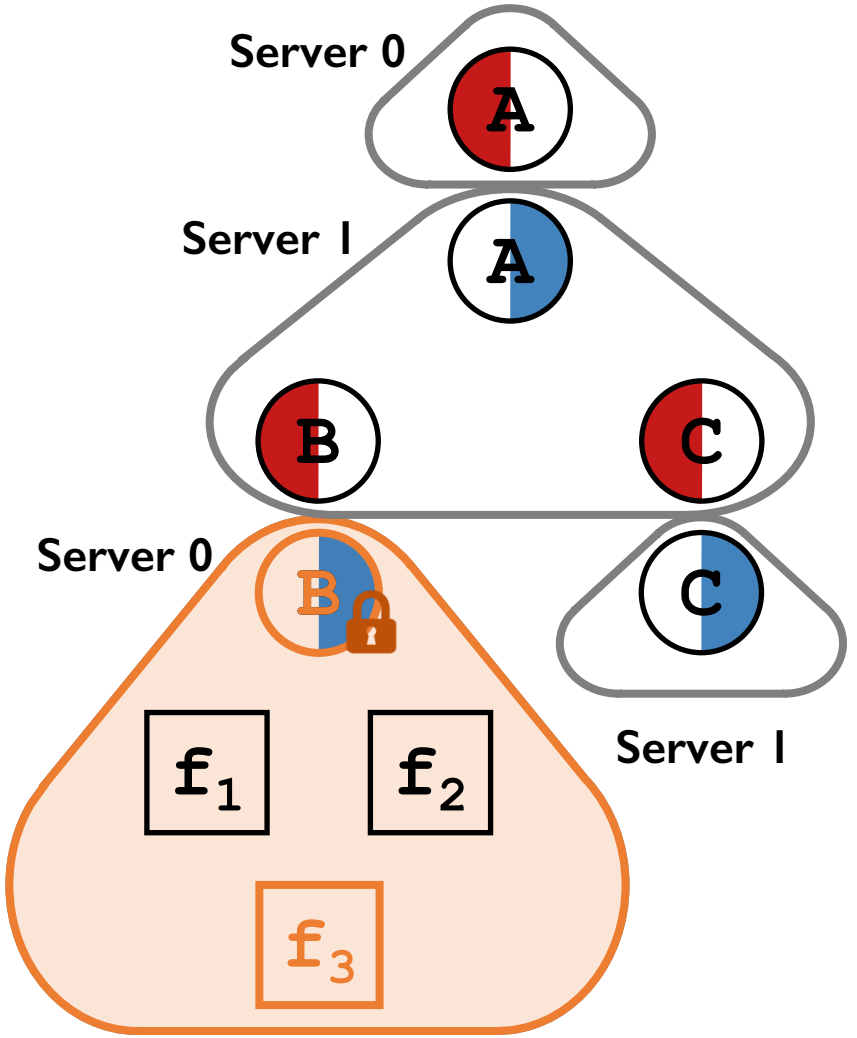
E.g., create(/A/B/$f_3$)

*Step 1.* lock the directory

*Step 2.* insert new file's metadata

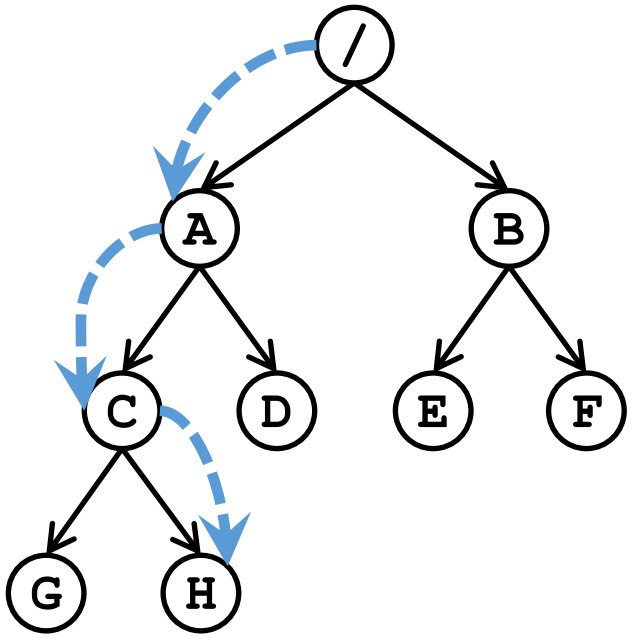
*Step 3.* update directory's *dirent* and *timestamps*

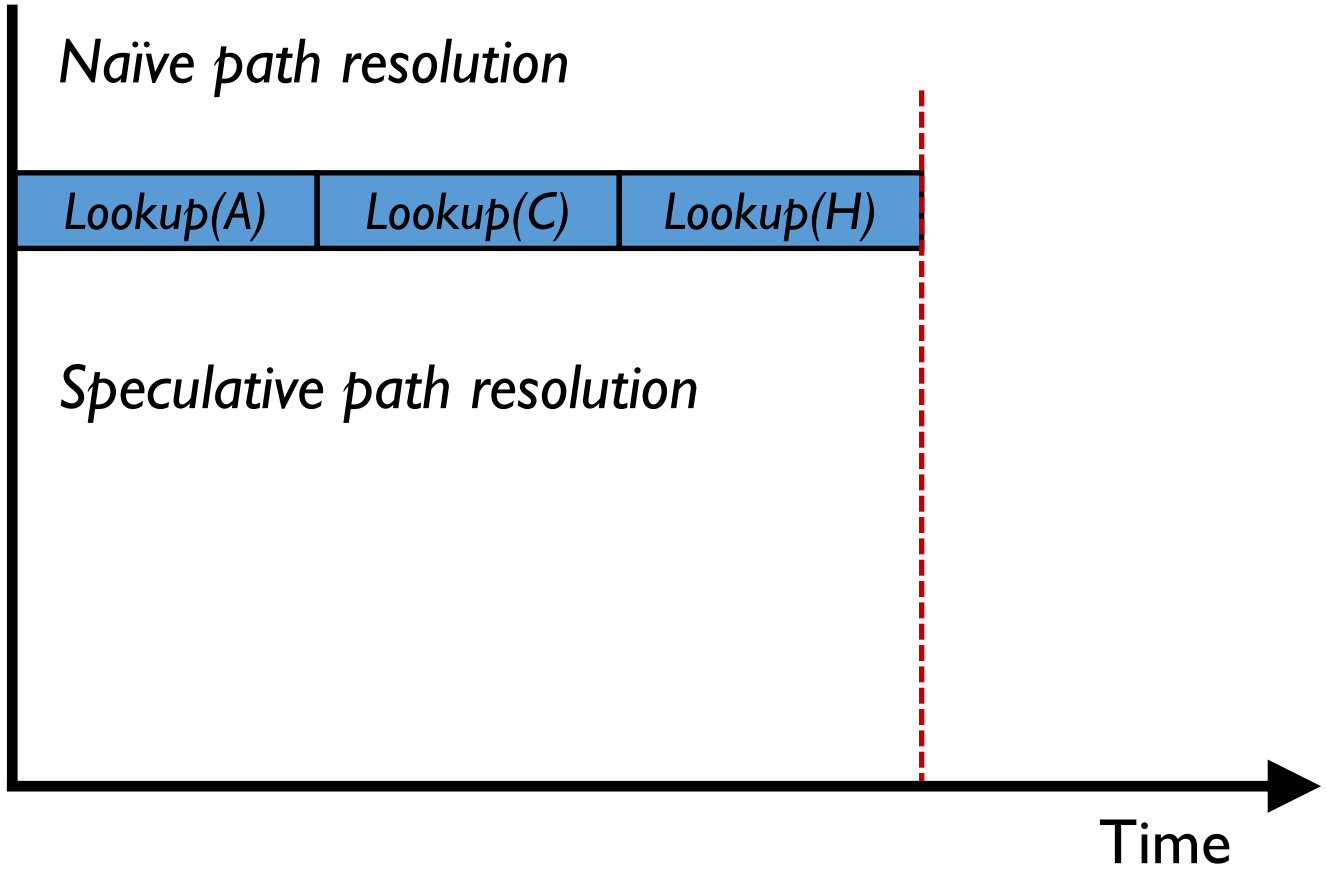
Only involve one single metadata server



# 2. Speculative Path Resolution

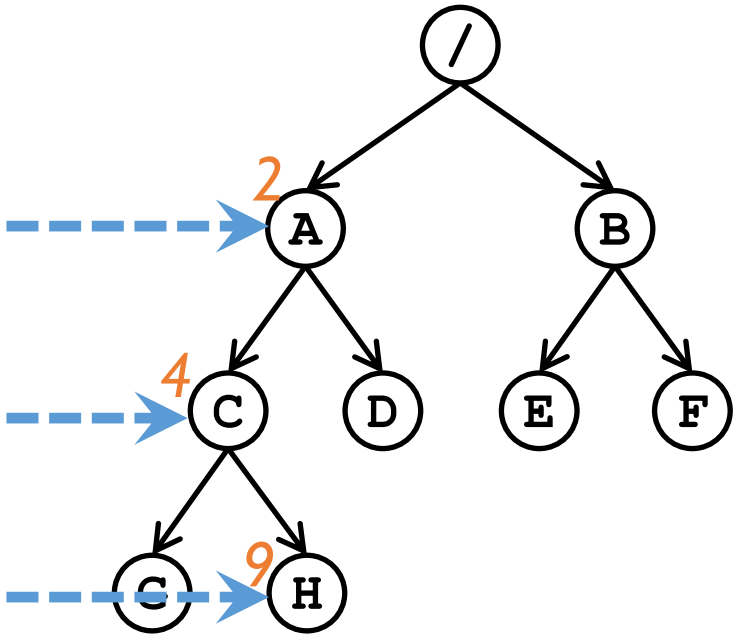
**Predict directory IDs and parallelize lookup requests**

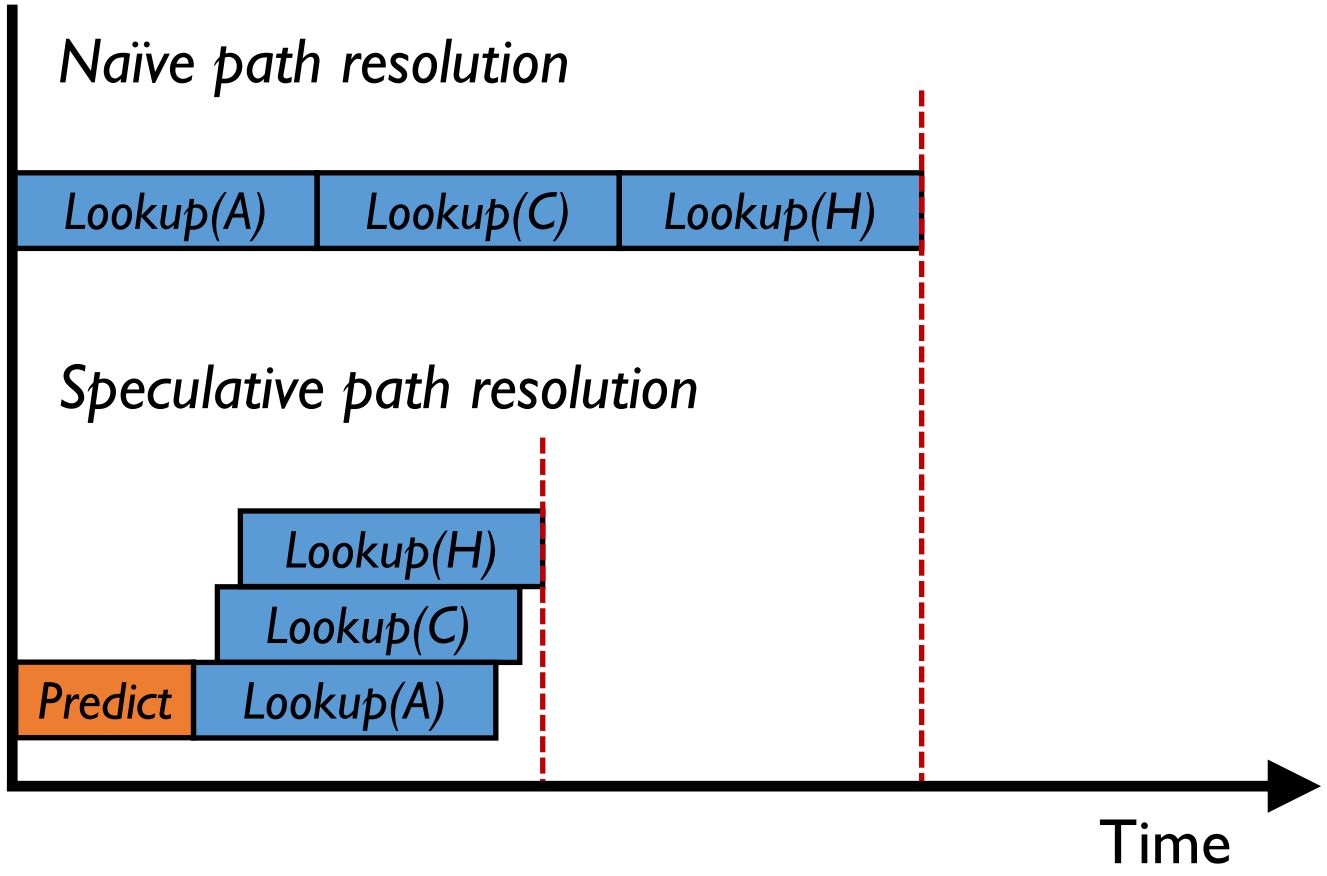


# 2. Speculative Path Resolution

## Predict directory IDs and parallelize lookup requests

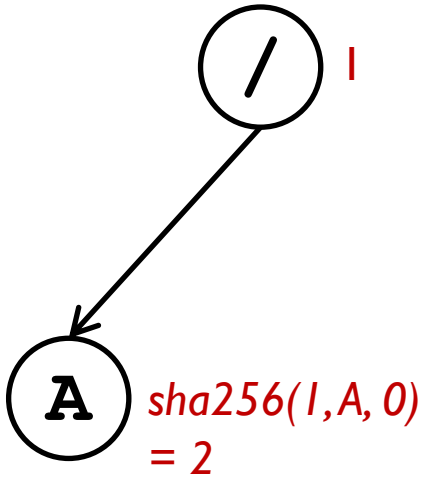


# 2. Speculative Path Resolution

## Predictable Directory ID

- SHA256(parent ID, name, version)
- Version is 0 by default, unless the ID collision is detected



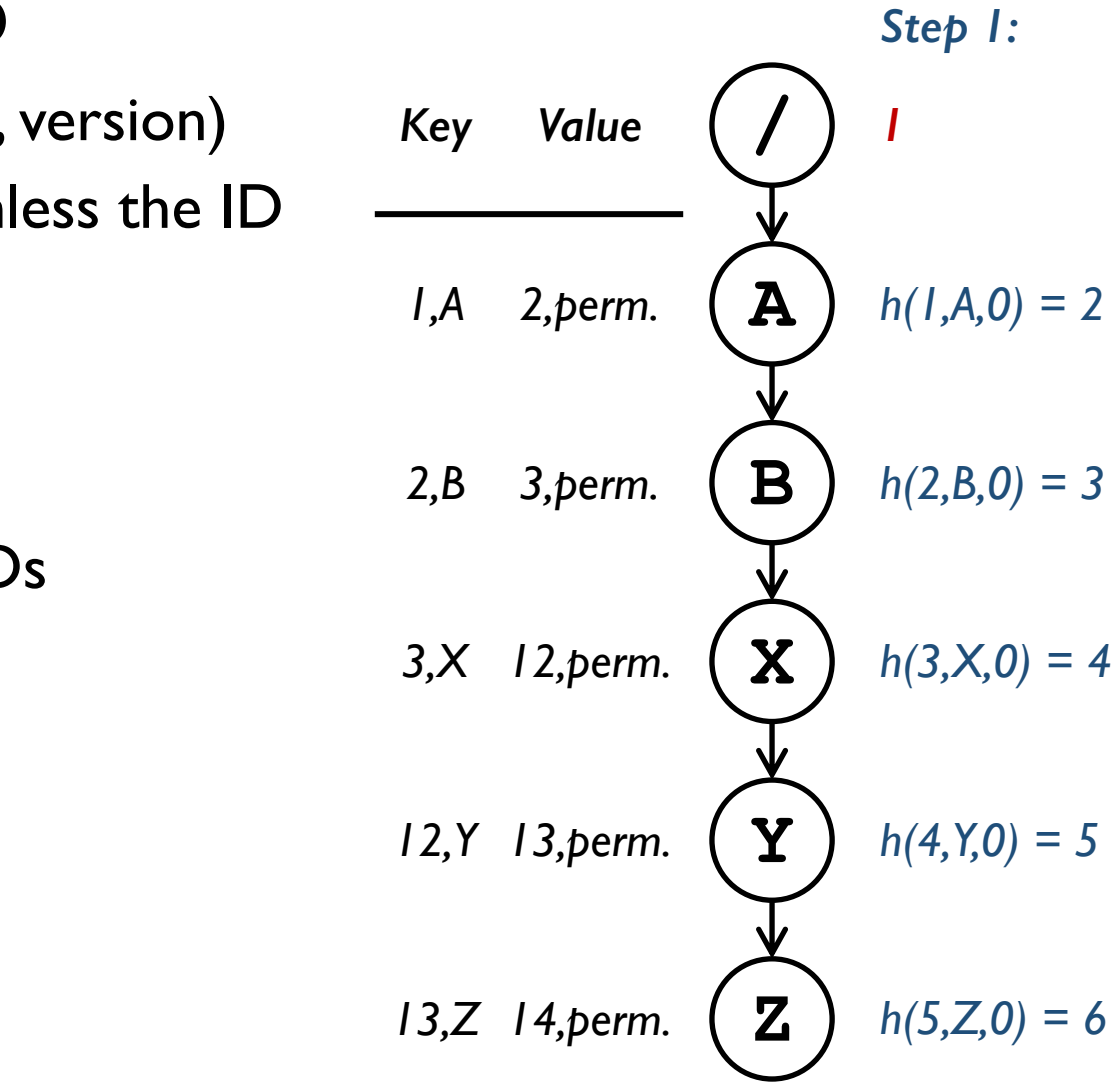
# 2. Speculative Path Resolution

## Predictable Directory ID

- SHA256(parent ID, name, version)
- Version is 0 by default, unless the ID collision is detected

## Parallel Path Resolution

*Step 1.* predict directory IDs



# 2. Speculative Path Resolution

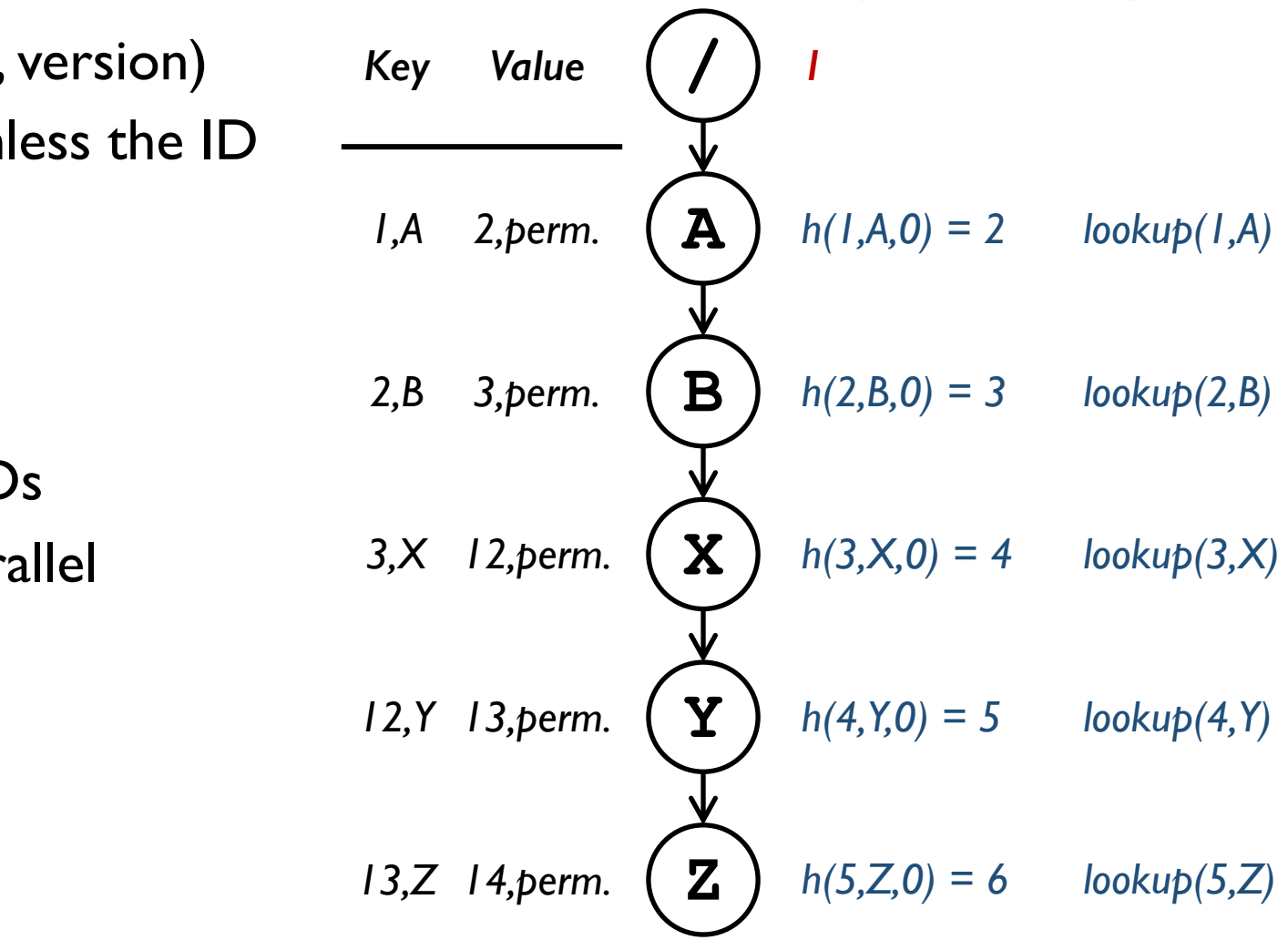
## Predictable Directory ID

- SHA256(parent ID, name, version)
- Version is 0 by default, unless the ID collision is detected

## Parallel Path Resolution

*Step 1.* predict directory IDs

*Step 2.* send lookups in parallel



# 2. Speculative Path Resolution

## Predictable Directory ID

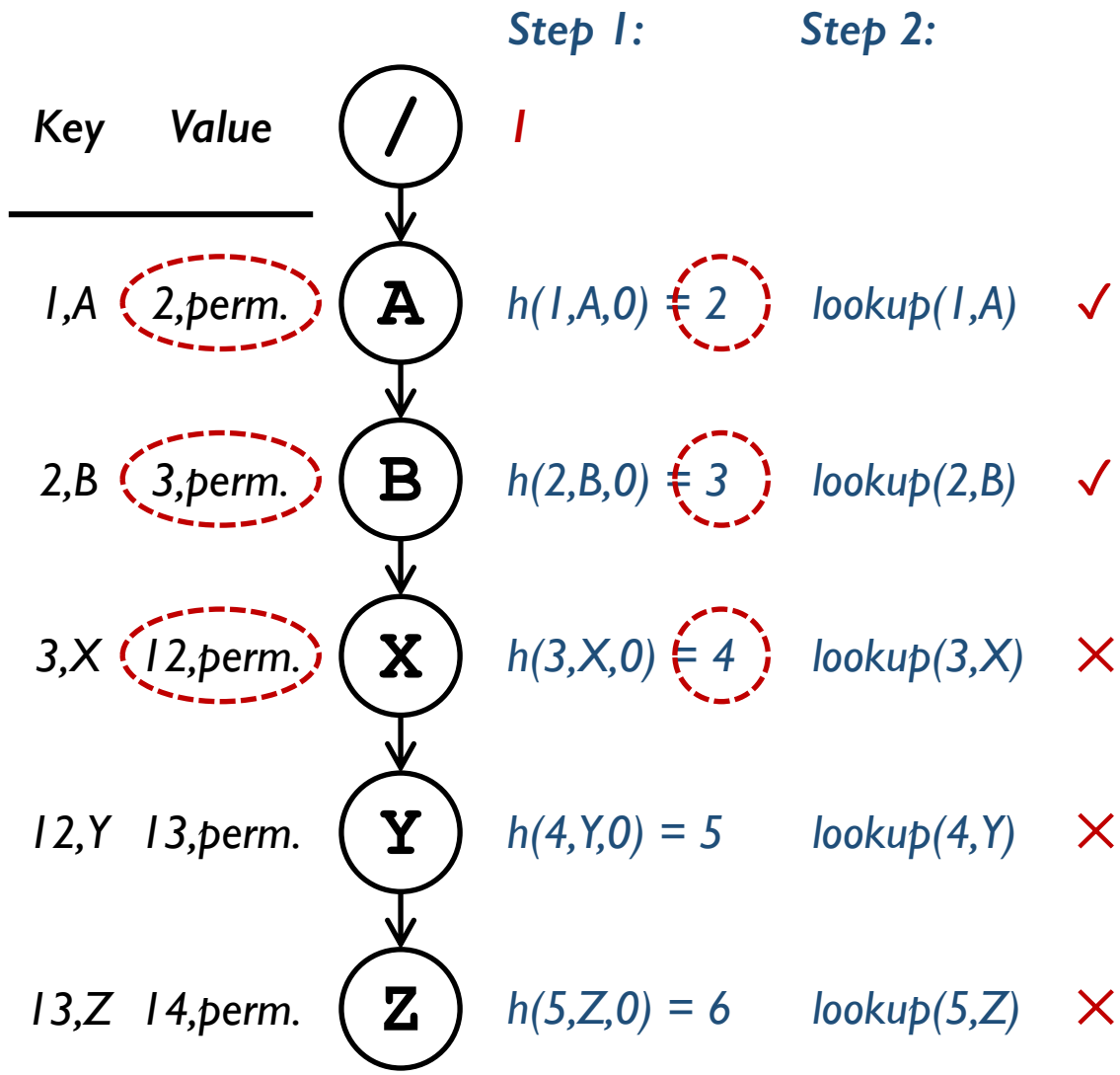
- SHA256(parent ID, name, version)
- Version is 0 by default, unless the ID collision is detected

## Parallel Path Resolution

*Step 1.* predict directory IDs

*Step 2.* send lookups in parallel

- check permissions
- verify predicted IDs



# 2. Speculative Path Resolution

## Predictable Directory ID

- SHA256(parent ID, name, version)
- Version is 0 by default, unless the ID collision is detected

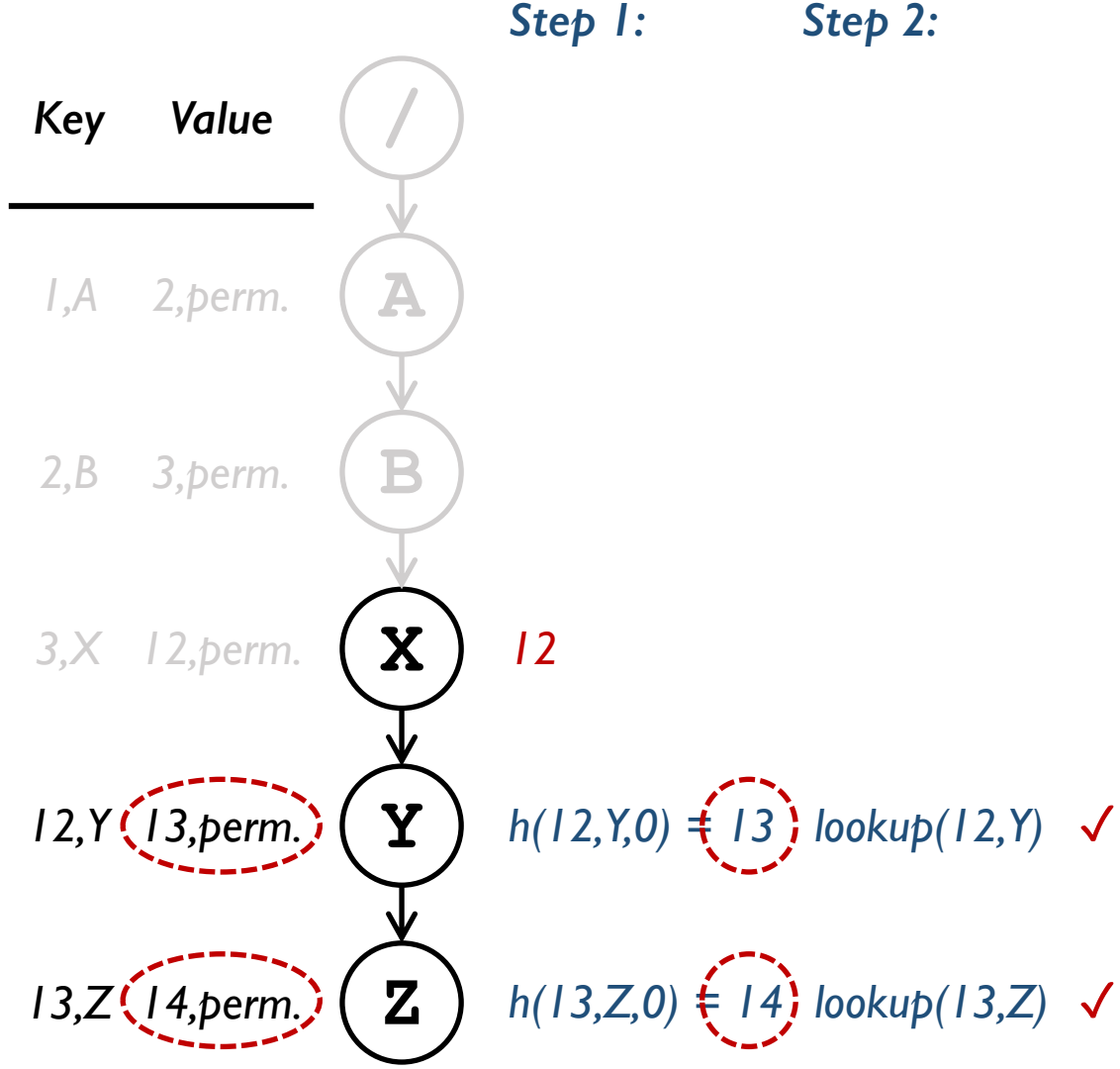
## Parallel Path Resolution

*Step 1.* predict directory IDs

*Step 2.* send lookups in parallel

- check permissions
- verify predicted IDs

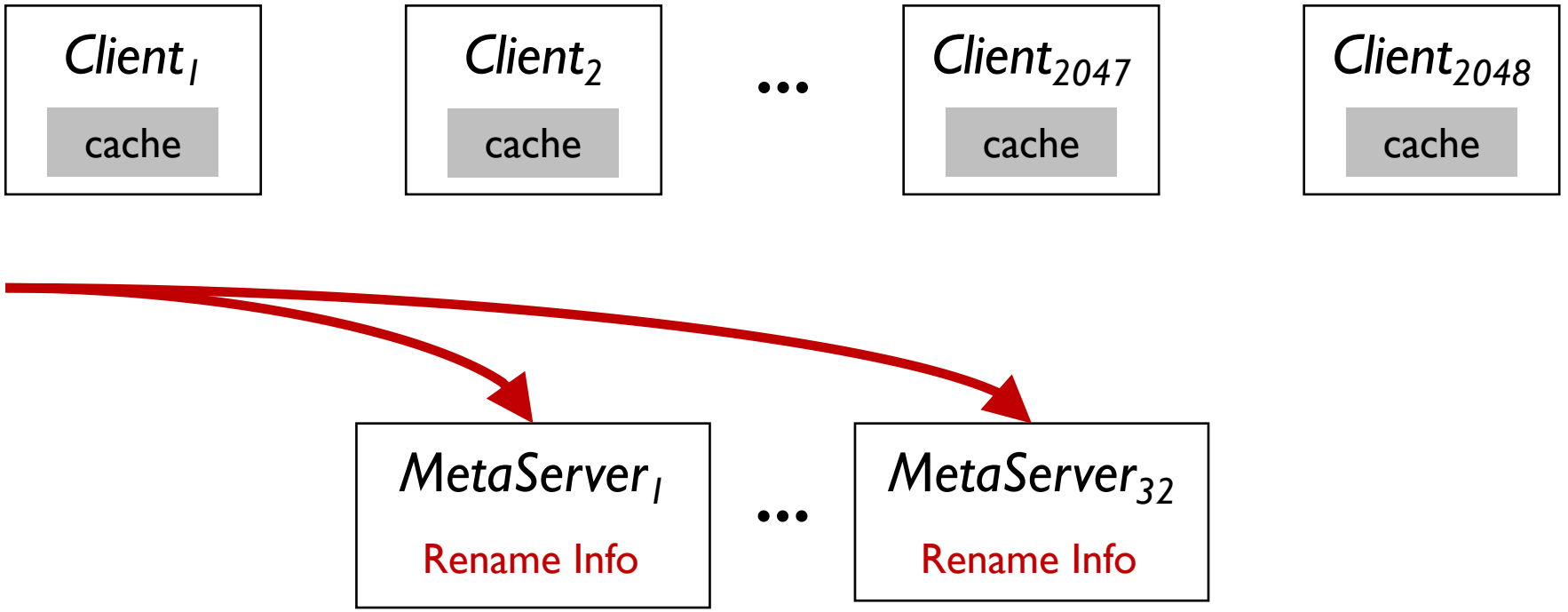
*Step 3.* repeat until finished



# 3. Optimistic Access Metadata Cache

High overhead of cache coherence due to the huge number of clients

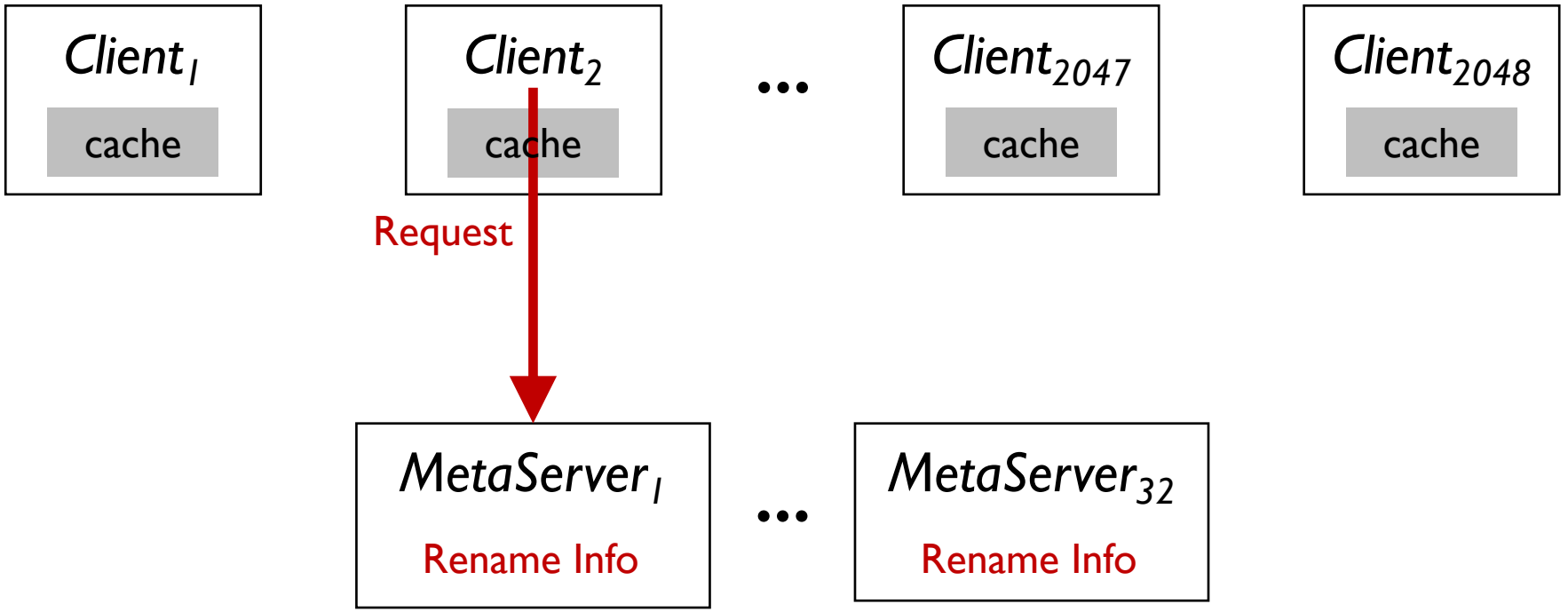
Validate cache staleness lazily on metadata servers



# 3. Optimistic Access Metadata Cache

High overhead of cache coherence due to the huge number of clients

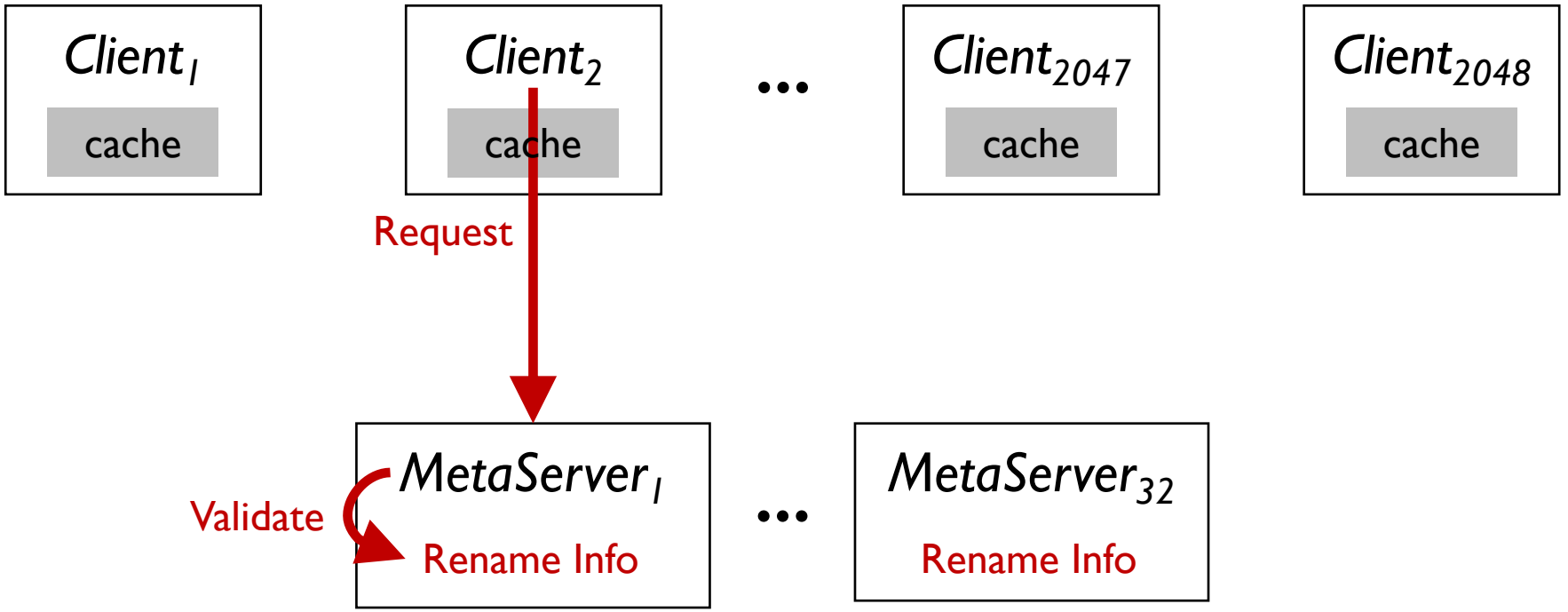
Validate cache staleness lazily on metadata servers



# 3. Optimistic Access Metadata Cache

High overhead of cache coherence due to the huge number of clients

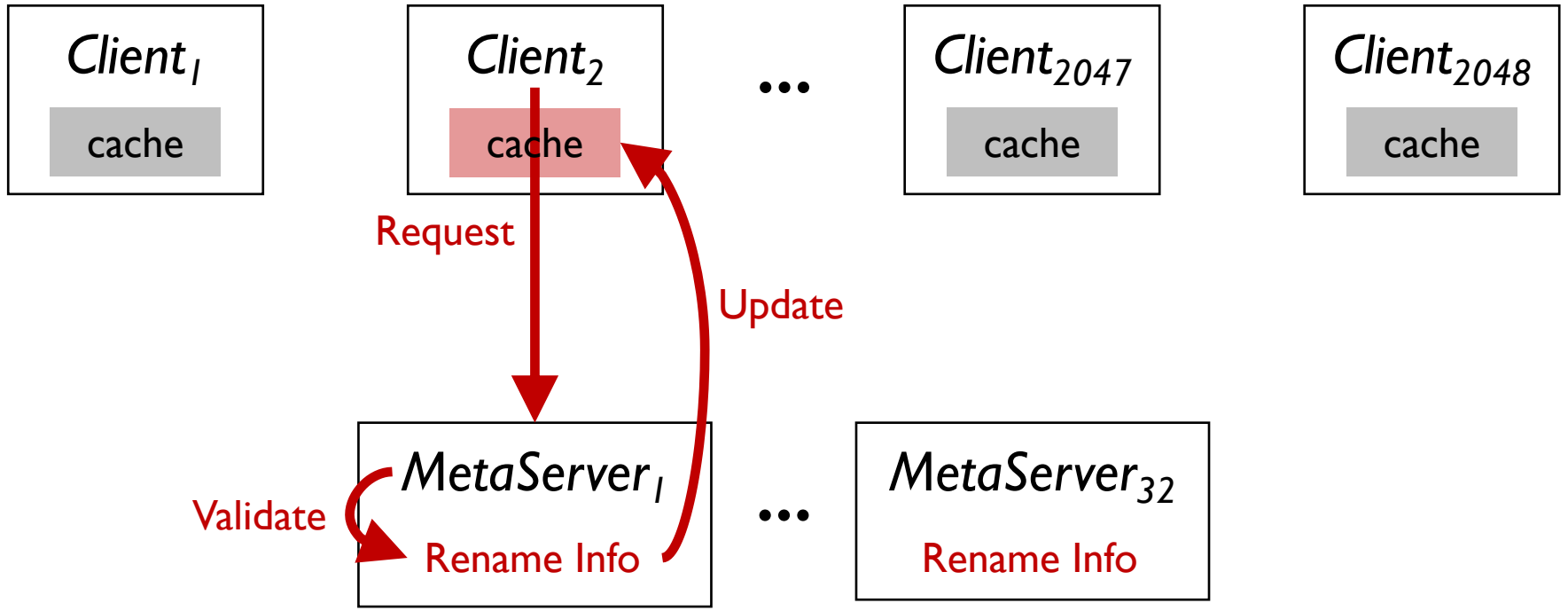
Validate cache staleness lazily on metadata servers



# 3. Optimistic Access Metadata Cache

High overhead of cache coherence due to the huge number of clients

Validate cache staleness lazily on metadata servers



# Outline

- Background & Motivation
- Design
- Evaluation
- Conclusion

# Experimental Setup

## Hardware Platform

- 32 server nodes; 32 client nodes; up to 100 billion files

| | |
|---|---|
| CPU | Intel Xeon Platinum 2.50GHz, 96 cores |
| Memory | Micron DDR4 2666MHz 32GB × 16 |
| Storage | RAMdisk |
| Network | ConnectX-4 Lx Dual-port 25Gbps |

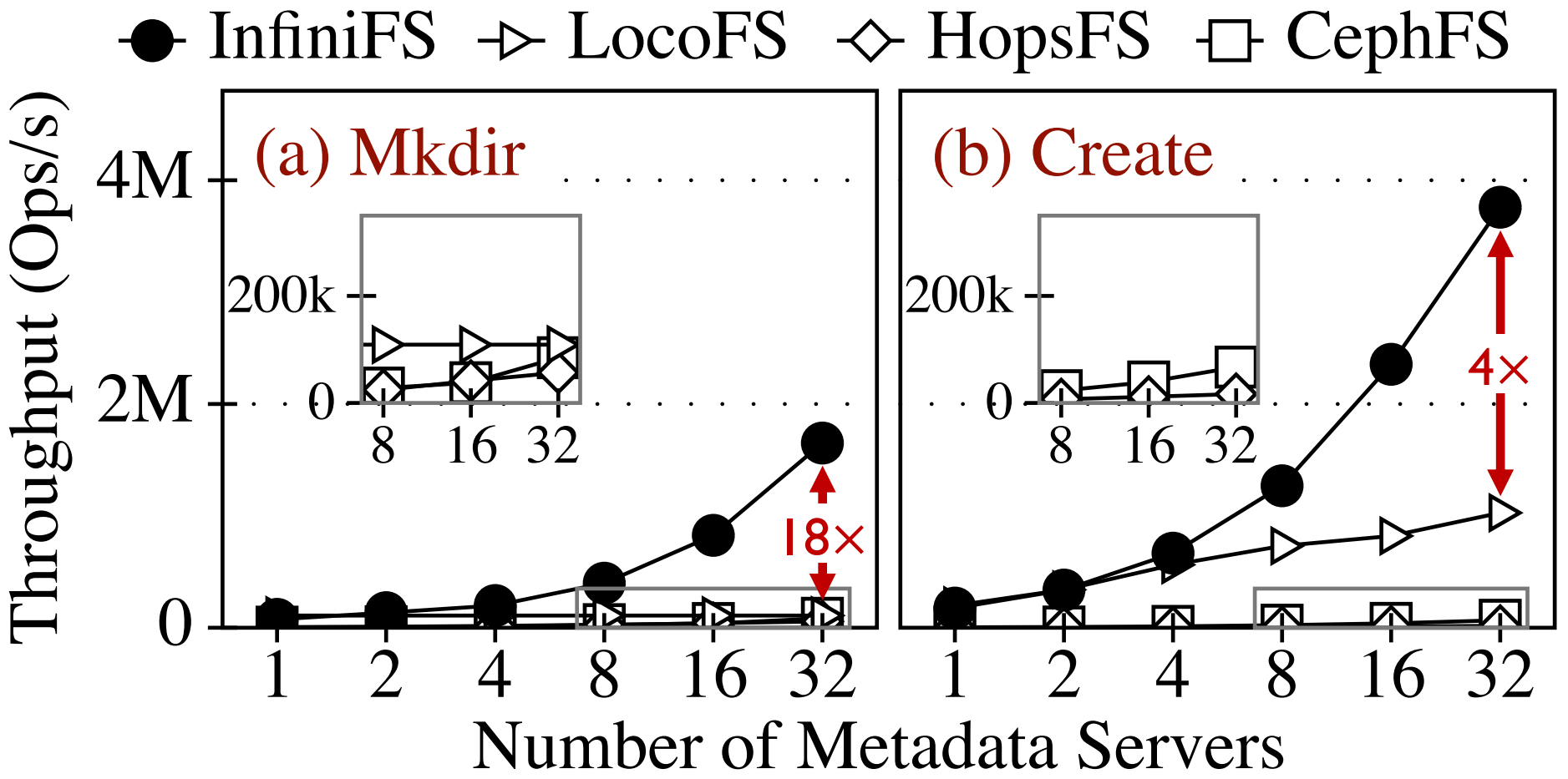
## Compared System

- LocoFS [SC '17], HopsFS [FAST '17] , IndexFS [SC '14], CephFS [OSDI '06]

## Benchmark

- The *mdtest* benchmark
- All tests create files of zero length

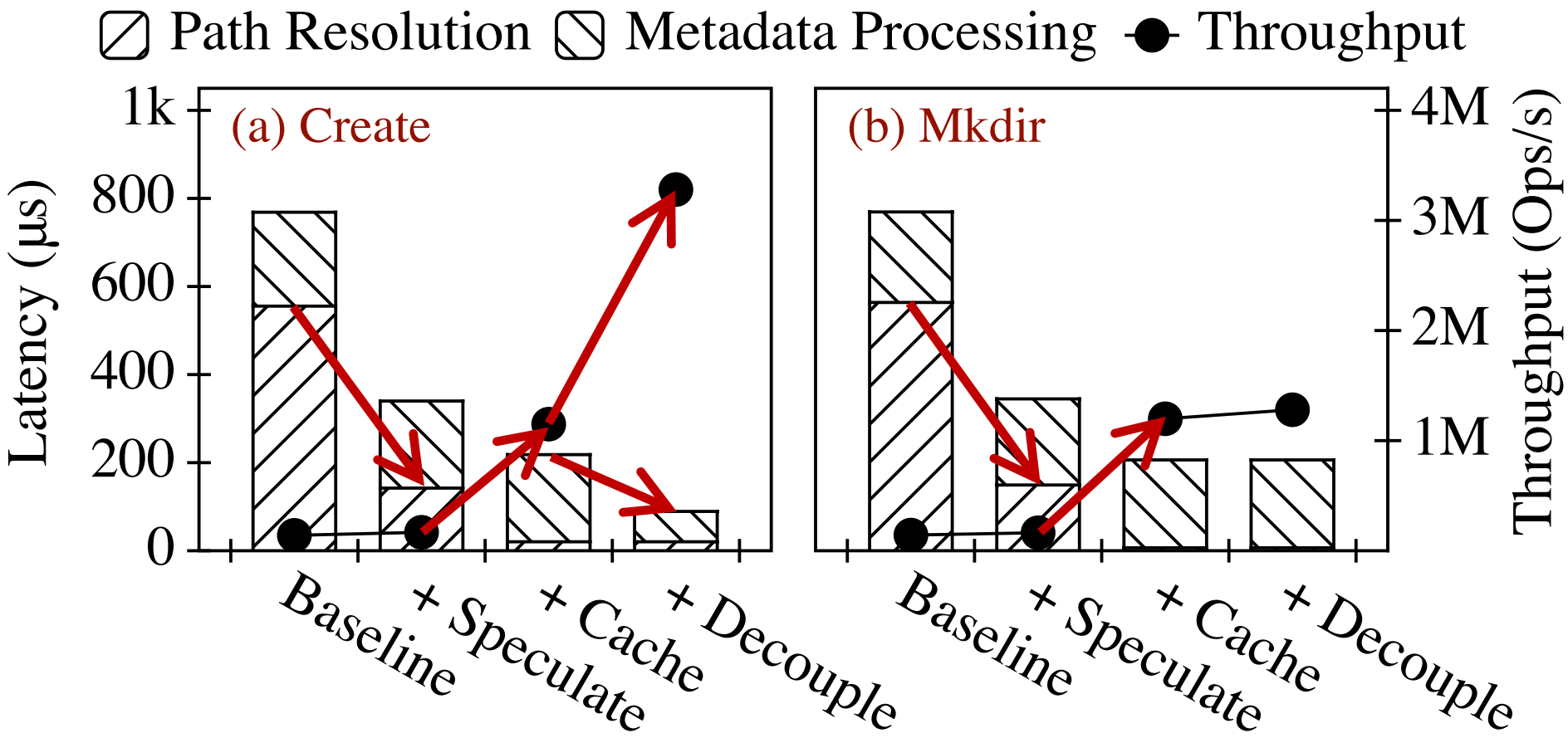
# Scalability




- 2048 clients
- 500 million files
- Depth of 10

InfiniFS outperforms the state-of-the-art systems

# Breakdown




- 1024 clients
- 250 million files
- Depth of 10

Speculate: Speculative Path Resolution
Cache: Optimistic Access Metadata Cache
Decouple: Access-Content Decoupled Partitioning

Designs of InfiniFS effectively improve the latency and throughput

# Outline

- Background & Motivation
- Design
- Evaluation
- Conclusion

# Conclusion

- Problem
  - Metadata service is inefficient for large-scale distributed filesystems
- Key Techniques of InfiniFS
  - Access-content decoupled partitioning
  - Speculative path resolution
  - Optimistic access metadata cache
- Results
  - InfiniFS outperforms the state-of-the-art systems
  - Designs of InfiniFS effectively improve the latency and throughput

# Thanks & QA

**InfiniFS: An Efficient Metadata Service for Large-Scale Distributed Filesystems**




Contact Information: lvwh19@mails.tsinghua.edu.cn