Carnegie Mellon University

Filesystems for Exascale Storage

Garth Gibson, garth@cs.cmu.edu

With Qing Zheng, Kai Ren, Lin Xiao, Lei Cao, Chuck Cranor

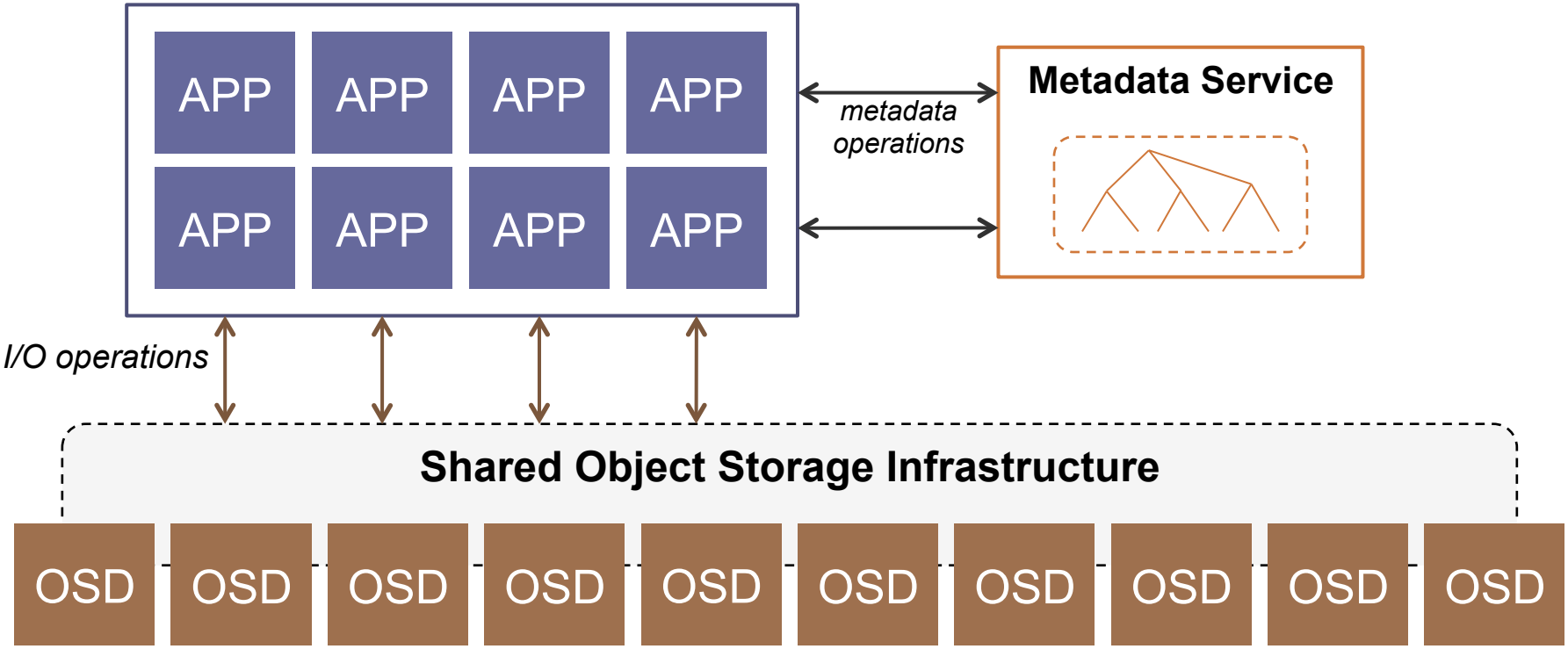
And Gary Grider and team, LANL

IRHPIT INSTITUTE FOR RELIABLE HIGH PERFORMANCE INFORMATION TECHNOLOGY

Intel Science & Technology Center for Cloud Computing

Parallel File System Architecture

Object-based scalable storage circa 2000: Data parallel



Data path is loosely coupled, parallel, simple, scalable but ~POSIX metadata path is not necessarily

Scaling Metadata Performance

INDEXFS

Two orders of magnitude faster than Lustre/PVFS [SC14]

BATCHFS

Scale another order of magnitude [PDSW14]

IRHPIT INSTITUTE FOR RELIABLE HIGH PERFORMANCE INFORMATION TECHNOLOGY

IndexFS:

Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion

SC14

Kai Ren, Qing Zheng, Swapnil Patil,
Garth Gibson

PARALLEL DATA LABORATORY

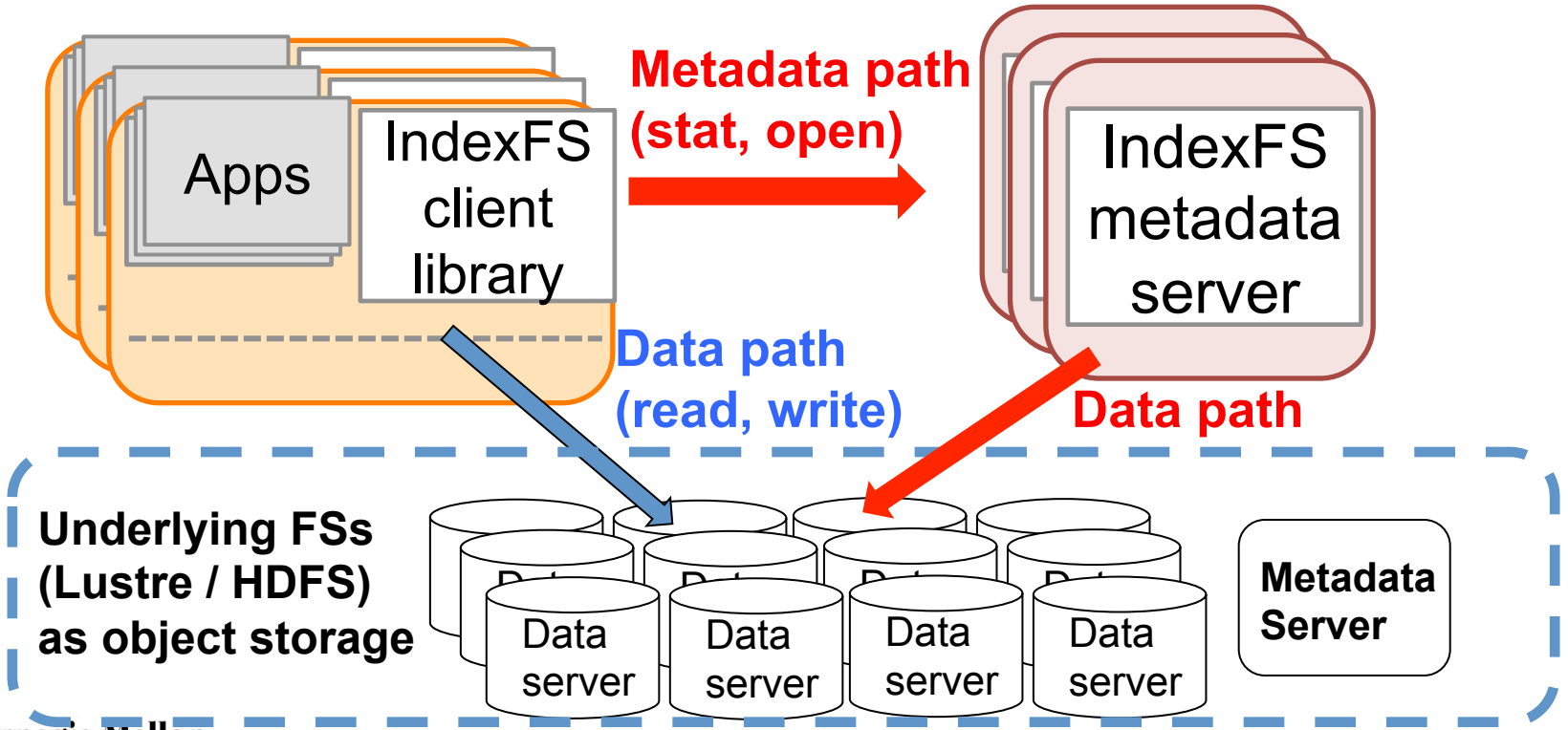
Carnegie Mellon University

Carnegie Mellon
Parallel Data Laboratory

Intel Science & Technology
Center for Cloud Computing

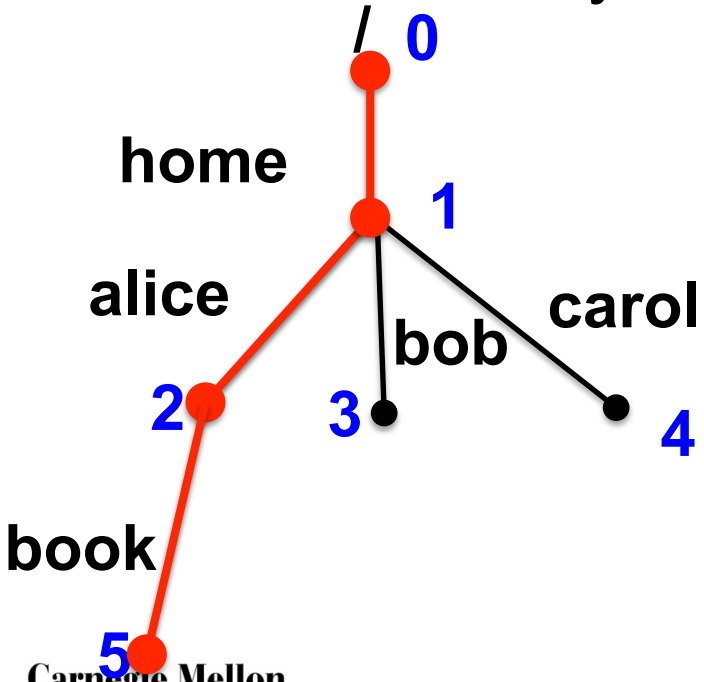
Layered Design

- IndexFS is layered on top of original DFSs
  - Provide a scalable metadata path for existing file systems such as HDFS, PVFS and Lustre



Efficient Per-Node Metadata Table

- Randomly distribute dir partitions over servers
- Key:<parent directory's inode number, filename>
- Value: attributes, file data or file pointer
- Stored in a key value store : LevelDB [Dean11]

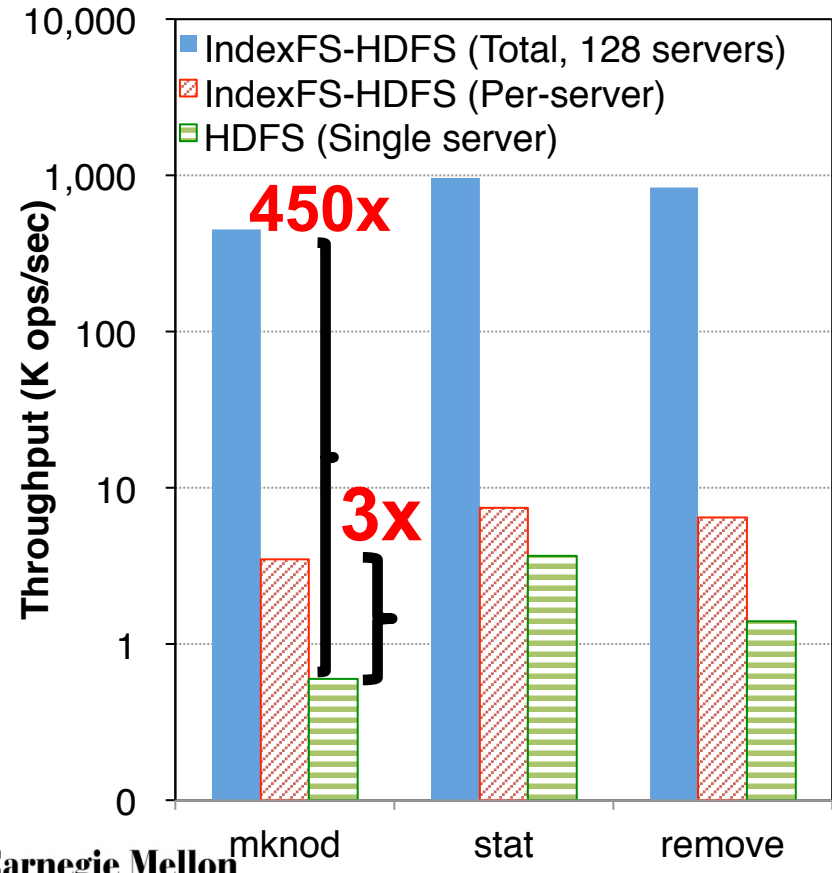


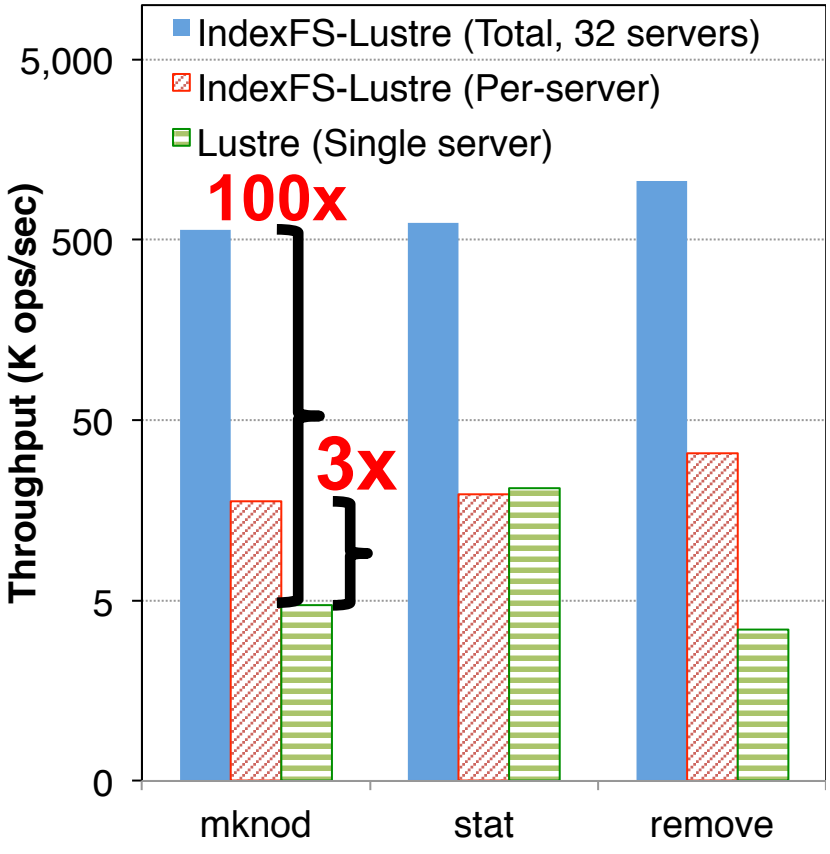
Lexicographic order

Key	Value
<0, *home*>	1, attributes
<1, *alice*>	2, attributes
<1, *bob*>	3, attributes
<1, *carol*>	4, attributes
<2, *book*>	5, attributes, small file data

100-450X Faster For HDFS & Lustre

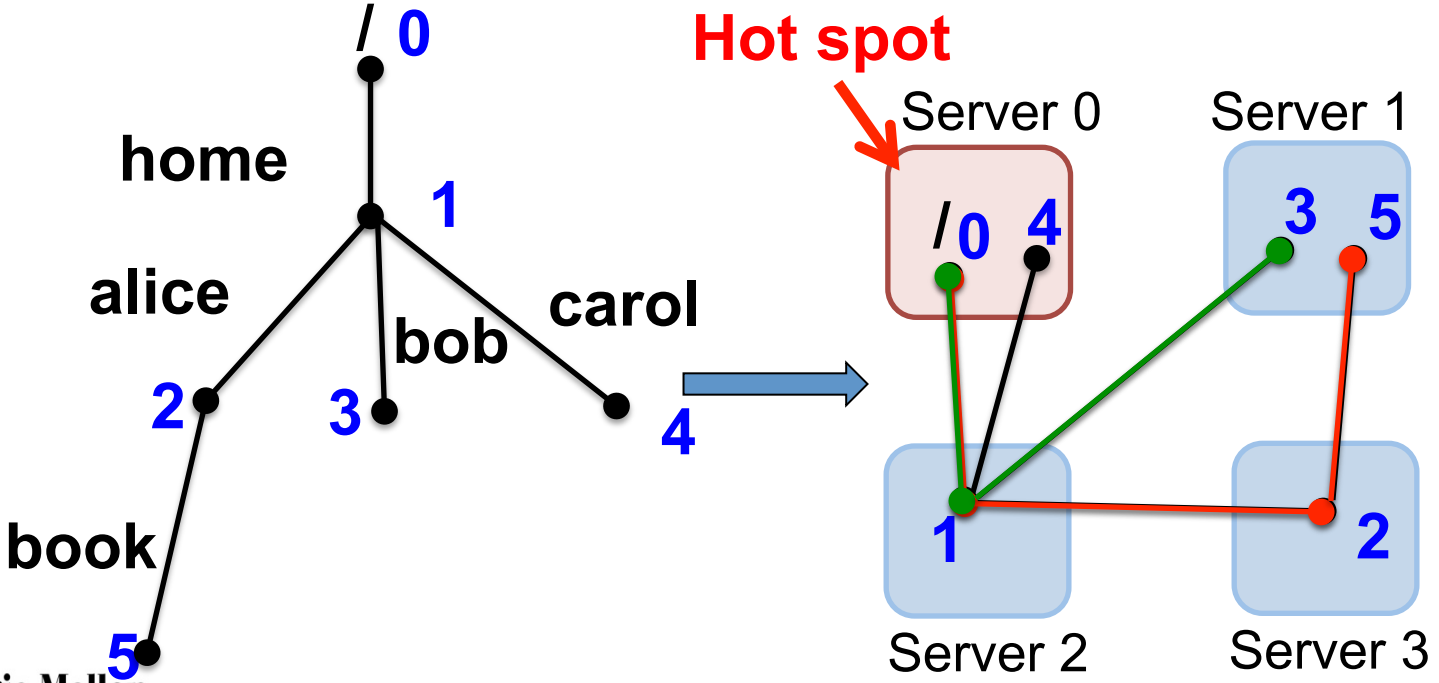
- Run mdtest on IndexFS layered on top of HDFS in PRObE Kodiak and Lustre in LANL Smog clusters





Hot Spot: Server with Root Directories

- Every POSIX lookup starts with root directory
  - Path traversal needs to visit each ancestor
  - Retrieve inode number and access permissions
  - Early IndexFS was defeated by this hot spot



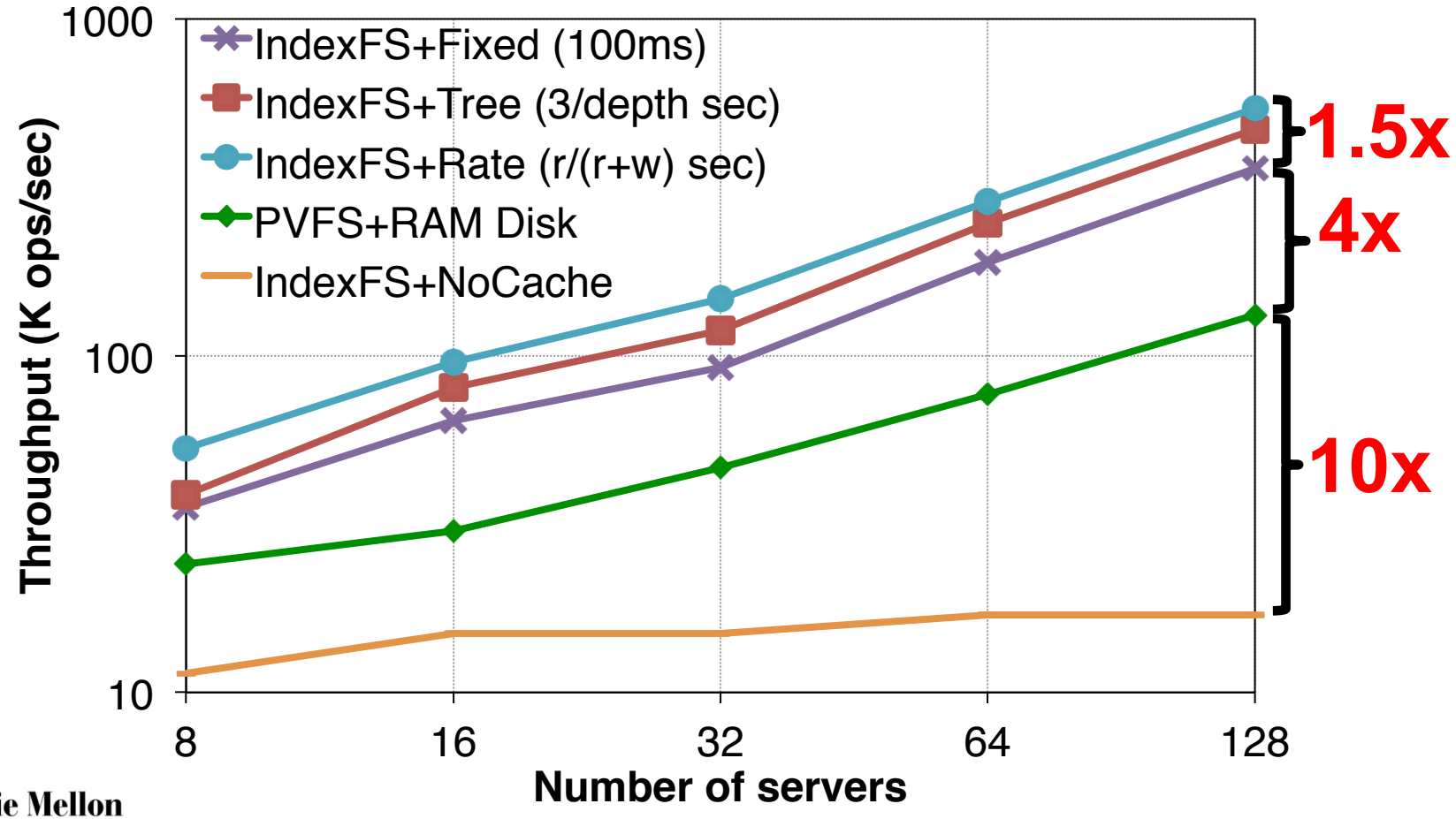
How To Mitigate Hot Spots?

- Client caches directory entry in many FSs
- Problem with invalidation callbacks at scale

- Idea: let the "clock" do the invalidation
  - Clients use ***read-only*** cache for directory entries
  - Servers remember only ***expiration deadline***
  - Assume ***clock synchronization*** within data center
  - Directory mutations wait for expiration
    - *rmdir*, *rename*, *chmod* can be slow

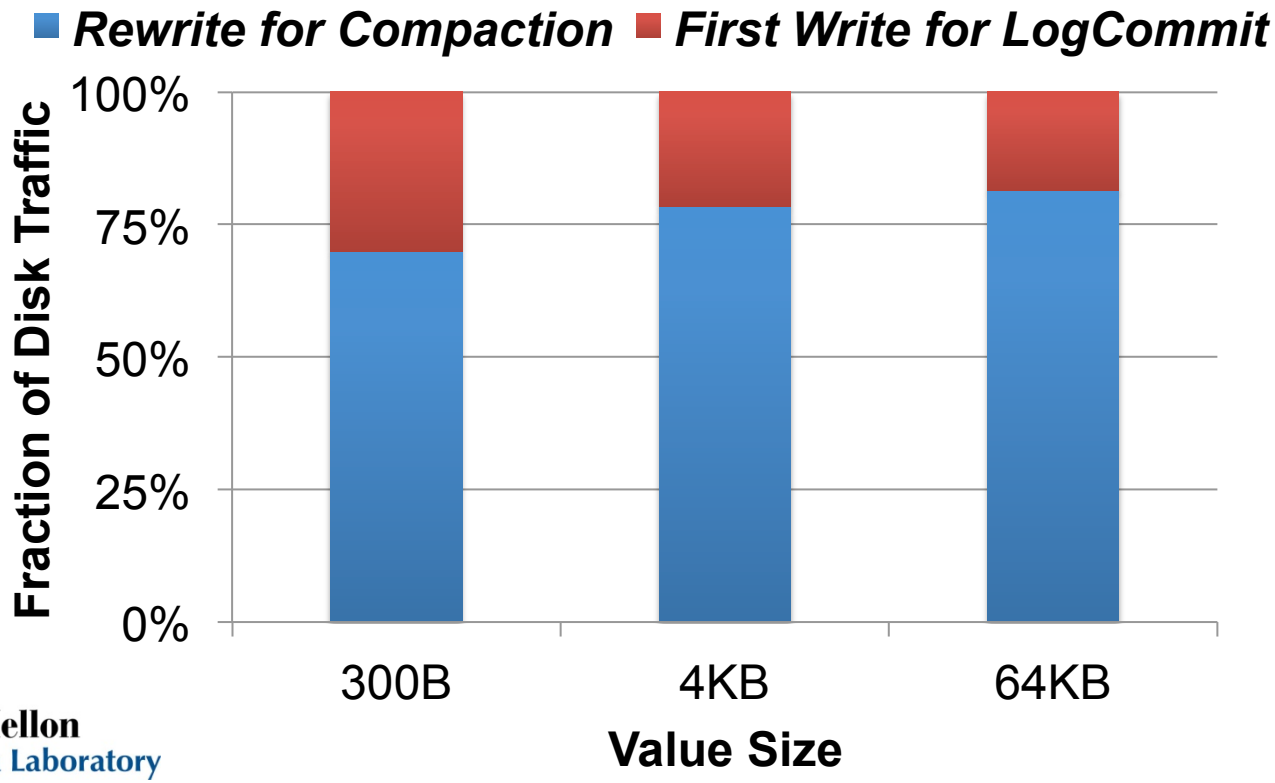
Replay Result: Throughput

- Directory entry cache mitigates hot spots



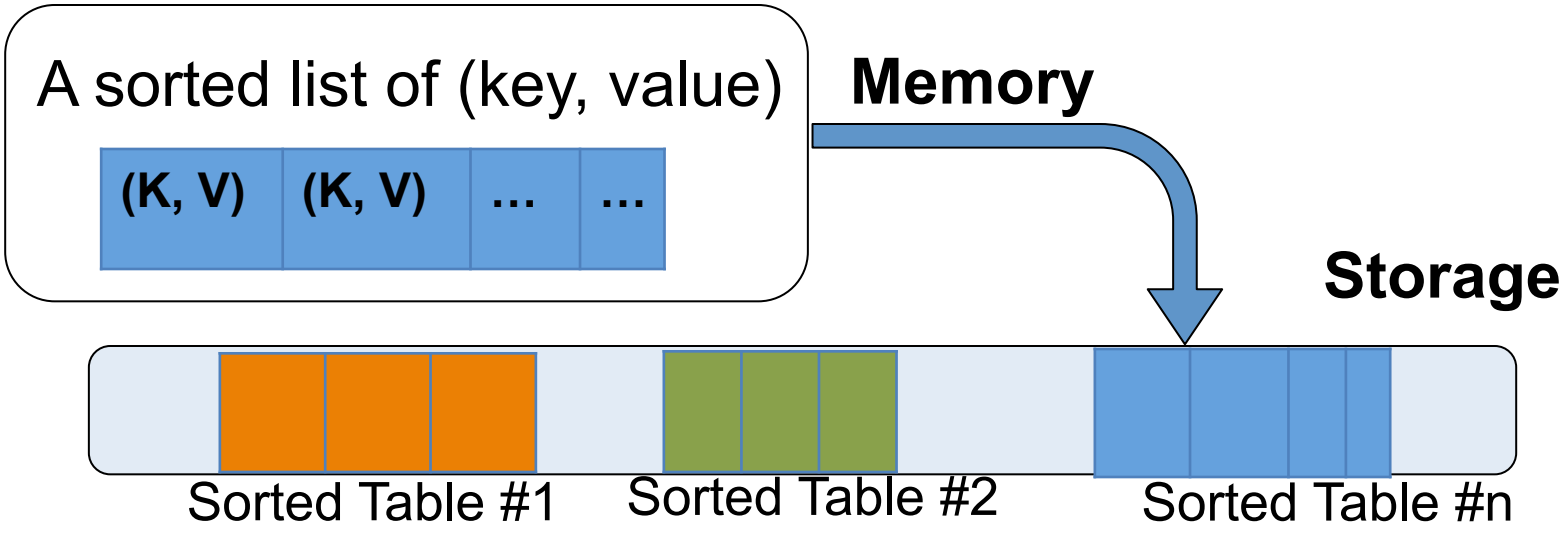
2nd Bottleneck: Compaction in LevelDB

- Slows insertion of user metadata/small files
  - 75% of insertion disk traffic caused by compaction
  - Compaction of larger values wastes more bandwidth



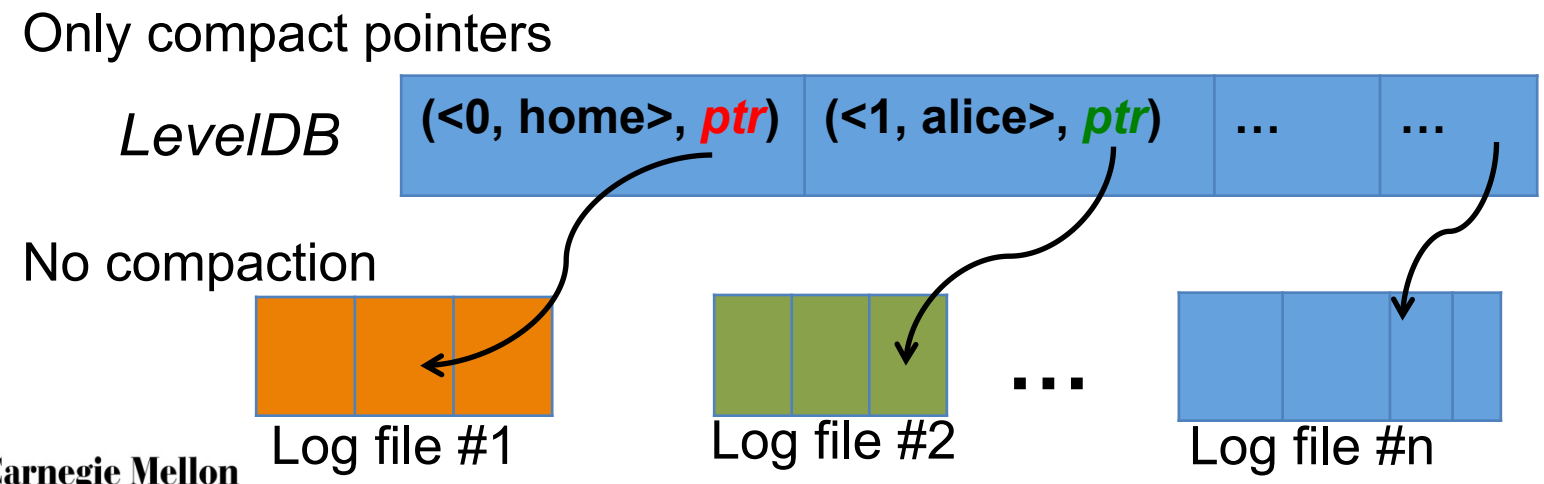
Background: LevelDB Internal

- LevelDB (LSM Tree): Insert and update ops:
  - Buffer and sort recent inserts/updates in memory
  - Flush buffer to generate immutable sorted tables
  - Perform compaction to reduce #tables for searching
- Want to avoid rewrites while having fast lookup



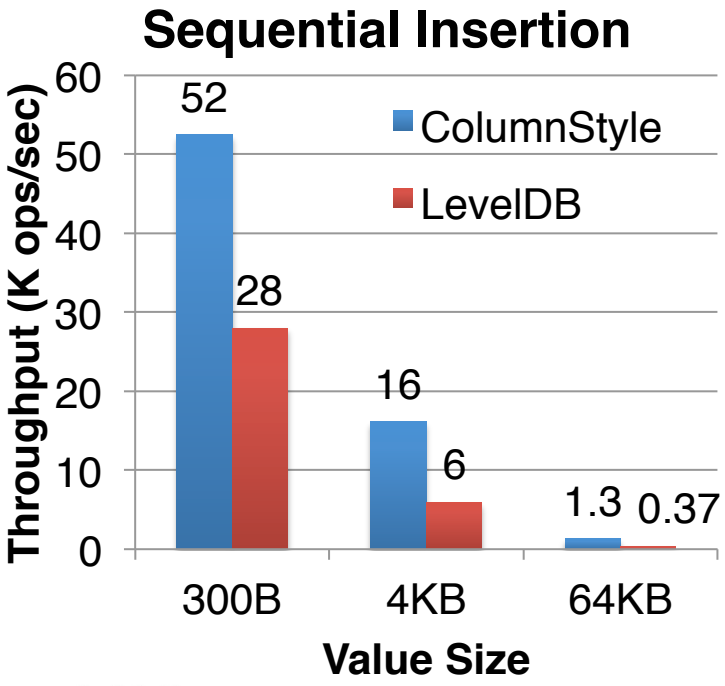
Column-Style Metadata Schema

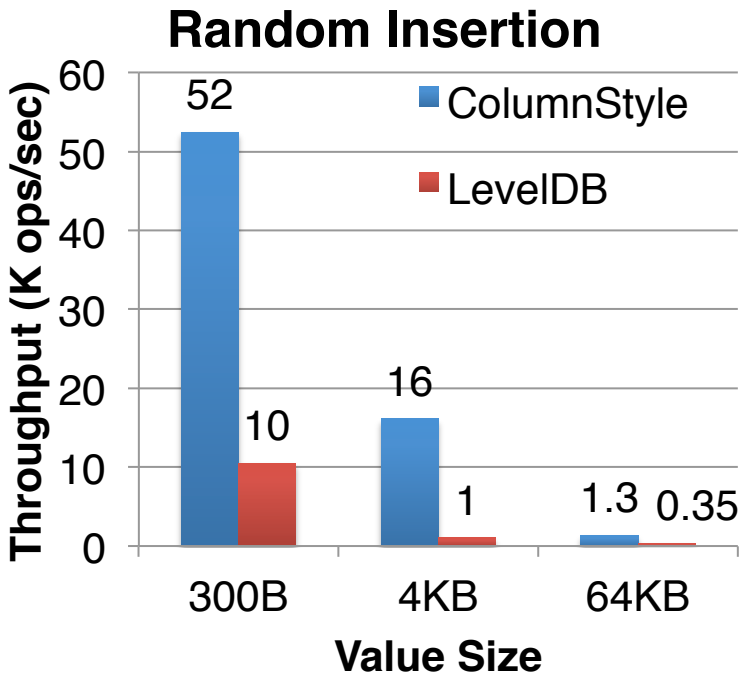
- Column-style approach:
  - Metadata/small files appended to non-LevelDB log files
  - LevelDB stores only pointers to metadata
  - Don't compact the metadata
    - Delay space reclamation for deleted/over-written values since metadata are small



Speed Up Ingestion Rate

- Insert 30M entries sequentially or randomly
  - Limit memory to 350 MB, using single SATA disk
- About 2 to 15 times faster insertion





IndexFS Summary

- IndexFS: a **middleware** approach to scale metadata path, portable to HDFS, Lustre, PVFS
- Scale out: **partition tree on directory basis**
  - File system distribution into server local LSM trees
  - Read-only consistent caching without per-client state
- Scale up: **optimize log-structured merge tree**
  - Column-style schema reduces compaction overhead
- Code available: http://www.pdl.cmu.edu/indexfs

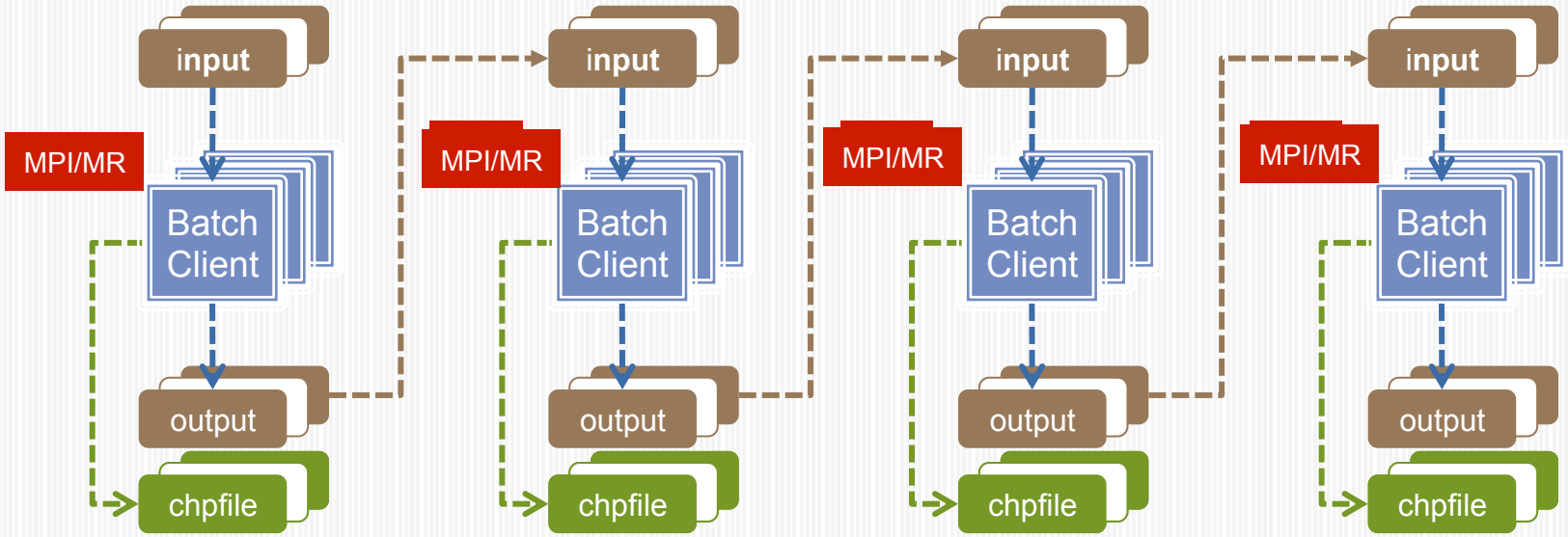
BATCHFS

Scaling the File System Control Plane with Client-Funded Metadata Servers

Qing Zheng, Kai Ren, Garth Gibson
Carnegie Mellon University

9th Parallel Data Storage Workshop (PDSW) 2014

Batch Applications



Batch apps are self-coordinated by framework & workflow engines

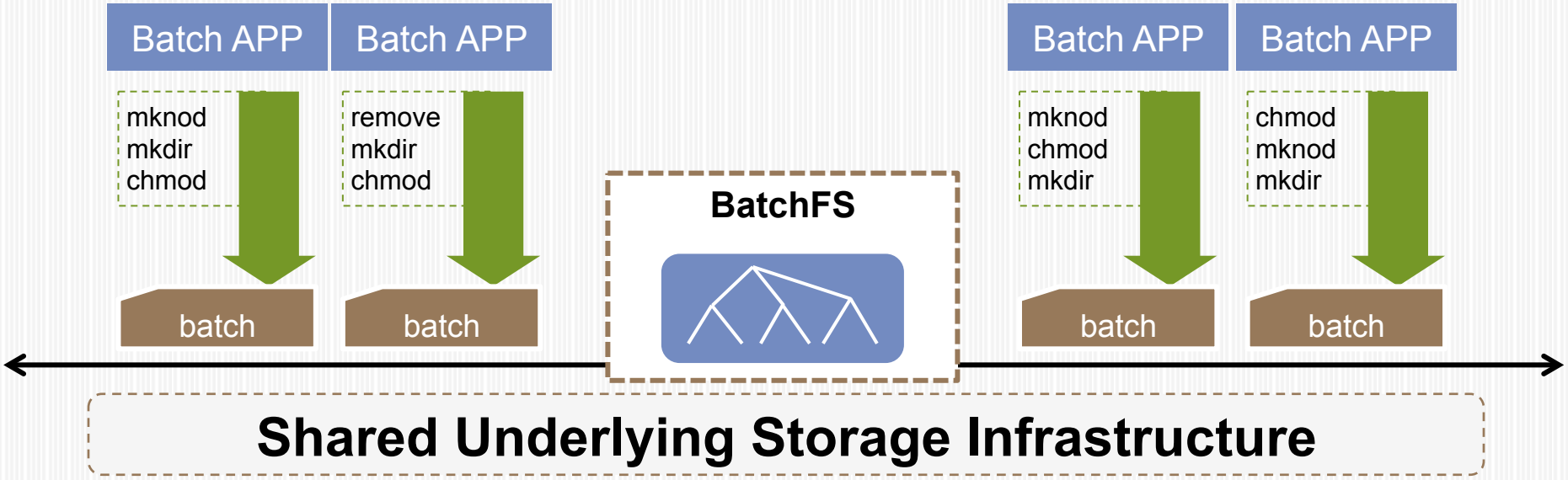
Key Observation

Batch apps **DON'T** need FS to communicate

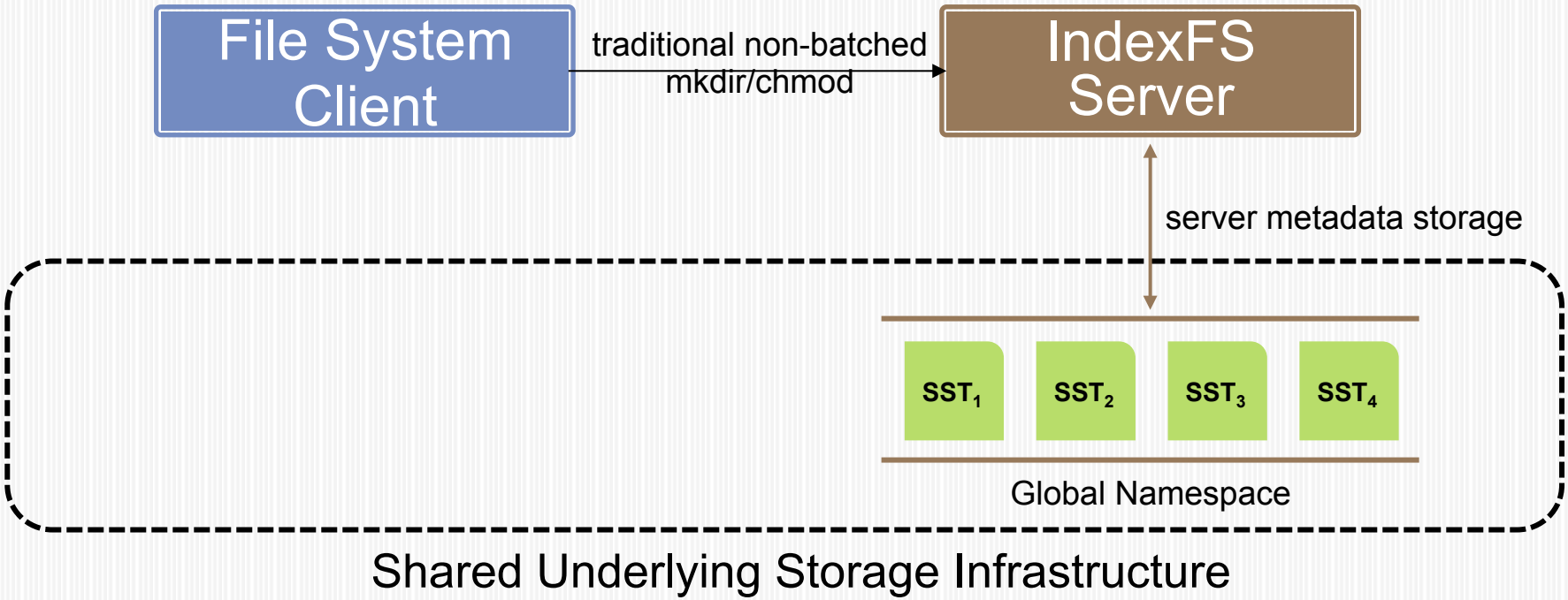
SYNCHRONOUS and SERIALIZED metadata management is **OVERKILL** for batch apps

Introducing BatchFS

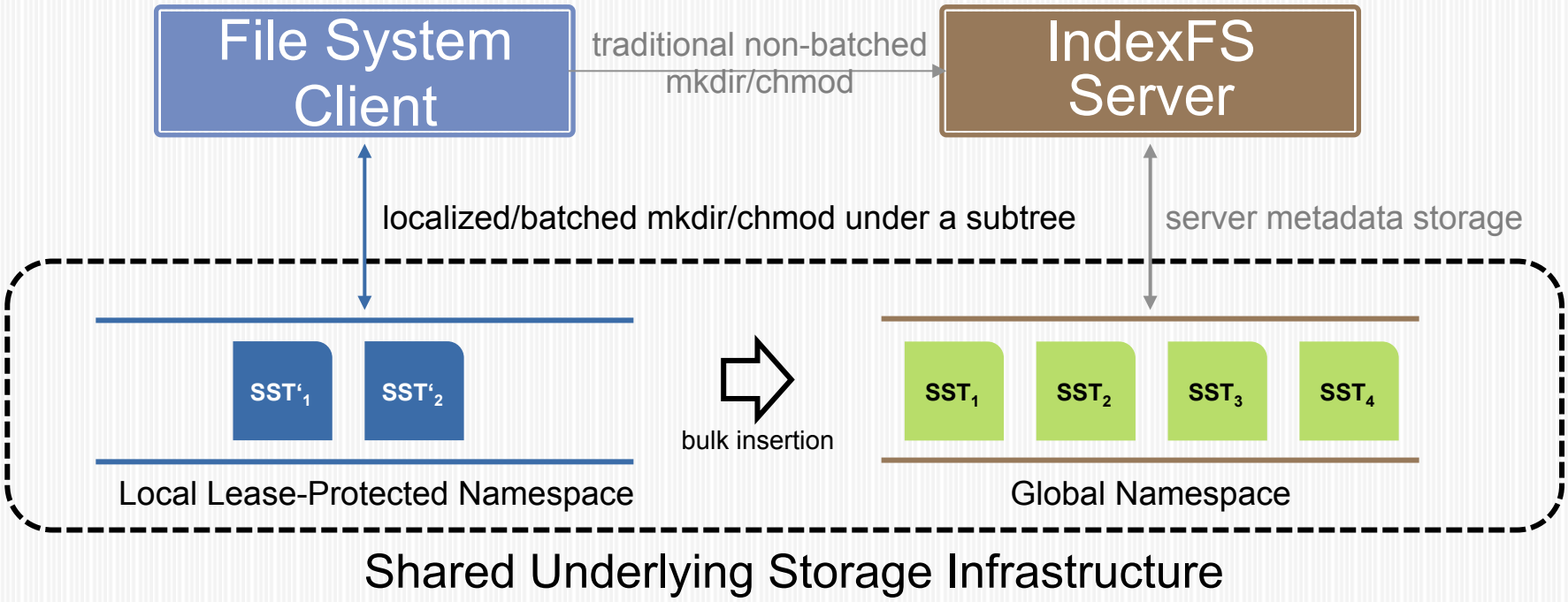
Deep batching for high throughput



Client-Server Interaction



Metadata Bulk Insertion

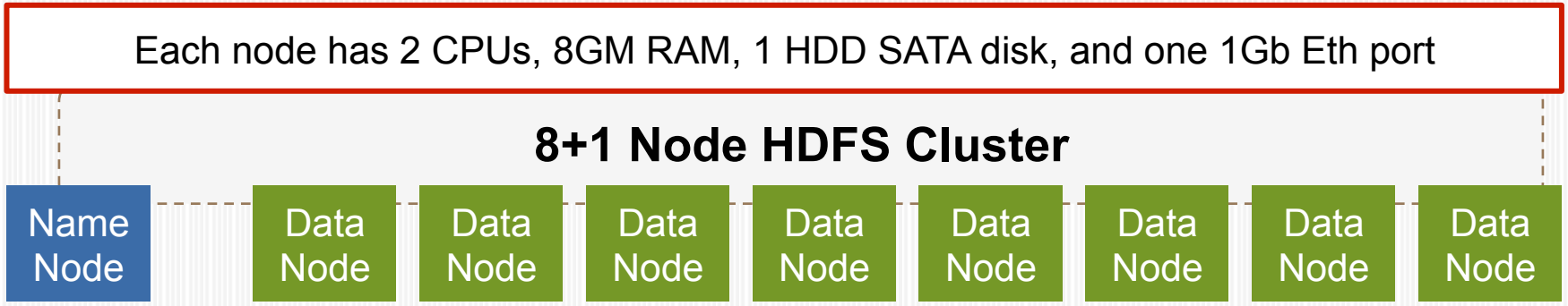


EXPERIMENT PLATFORM

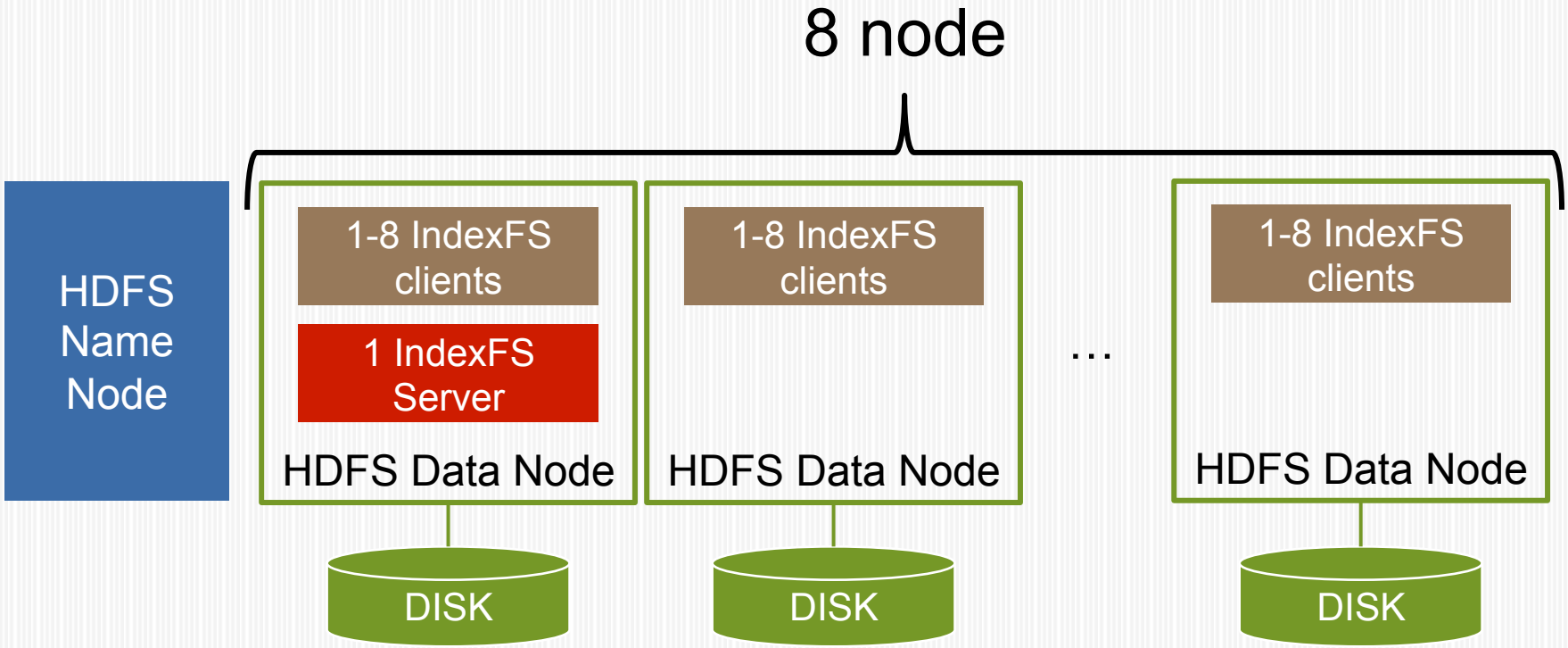
A prototype of BatchFS as an IndexFS feature

metadata bulk insertion (batching)

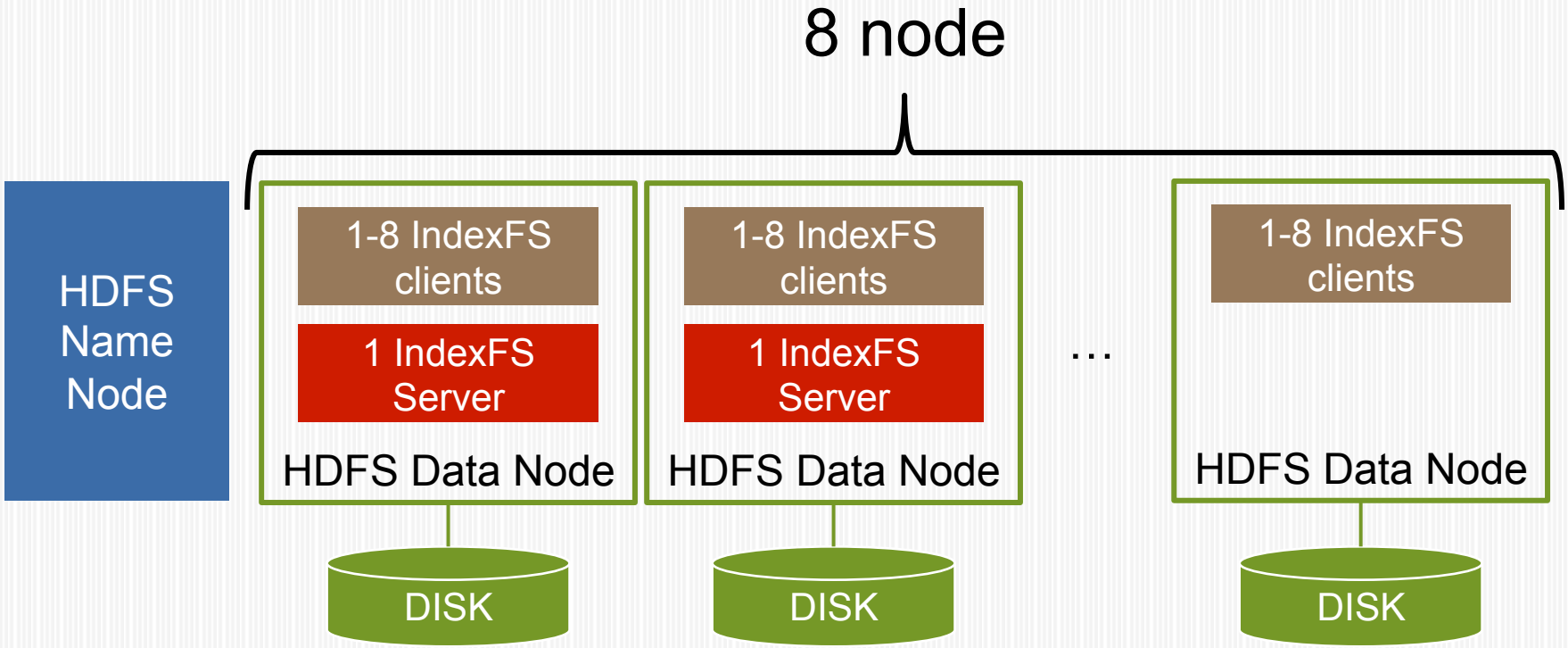
Clients create private directories & insert empty files



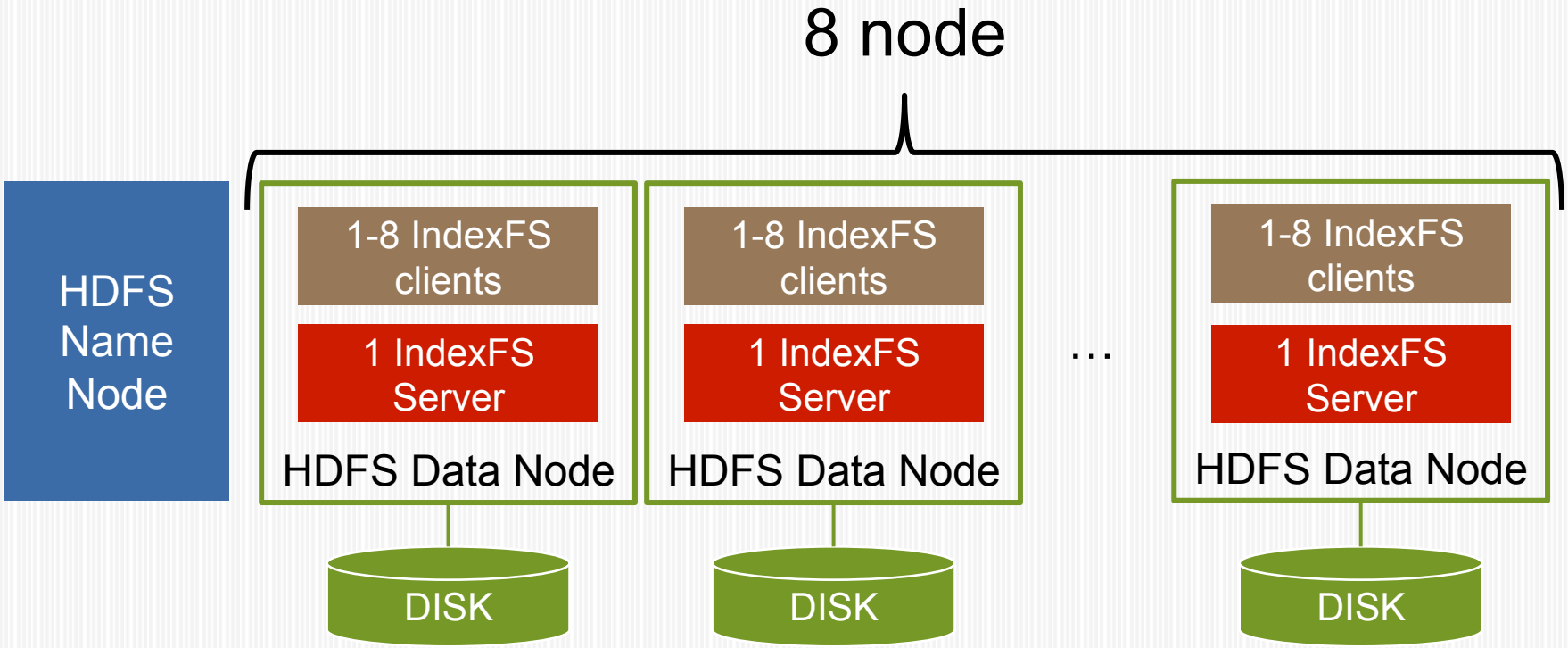
Experiment Setup #1



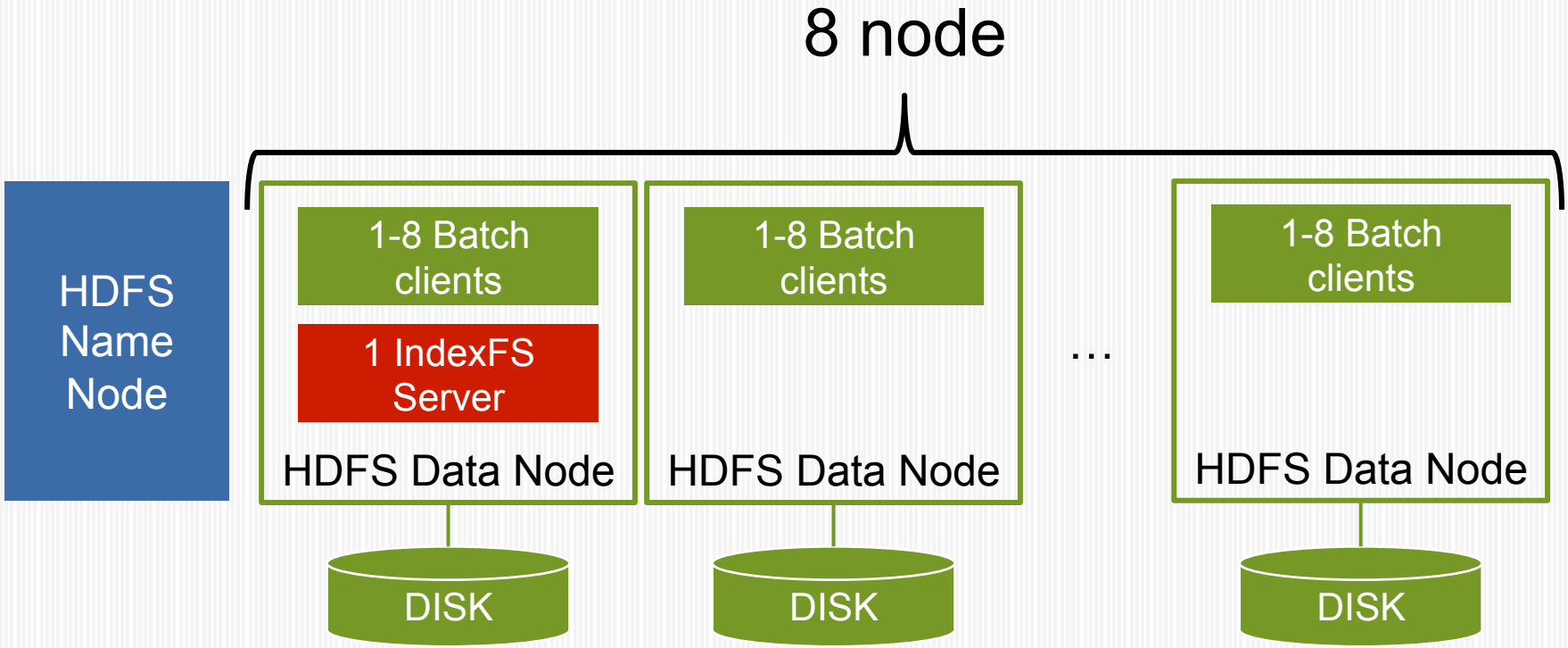
Experiment Setup #2



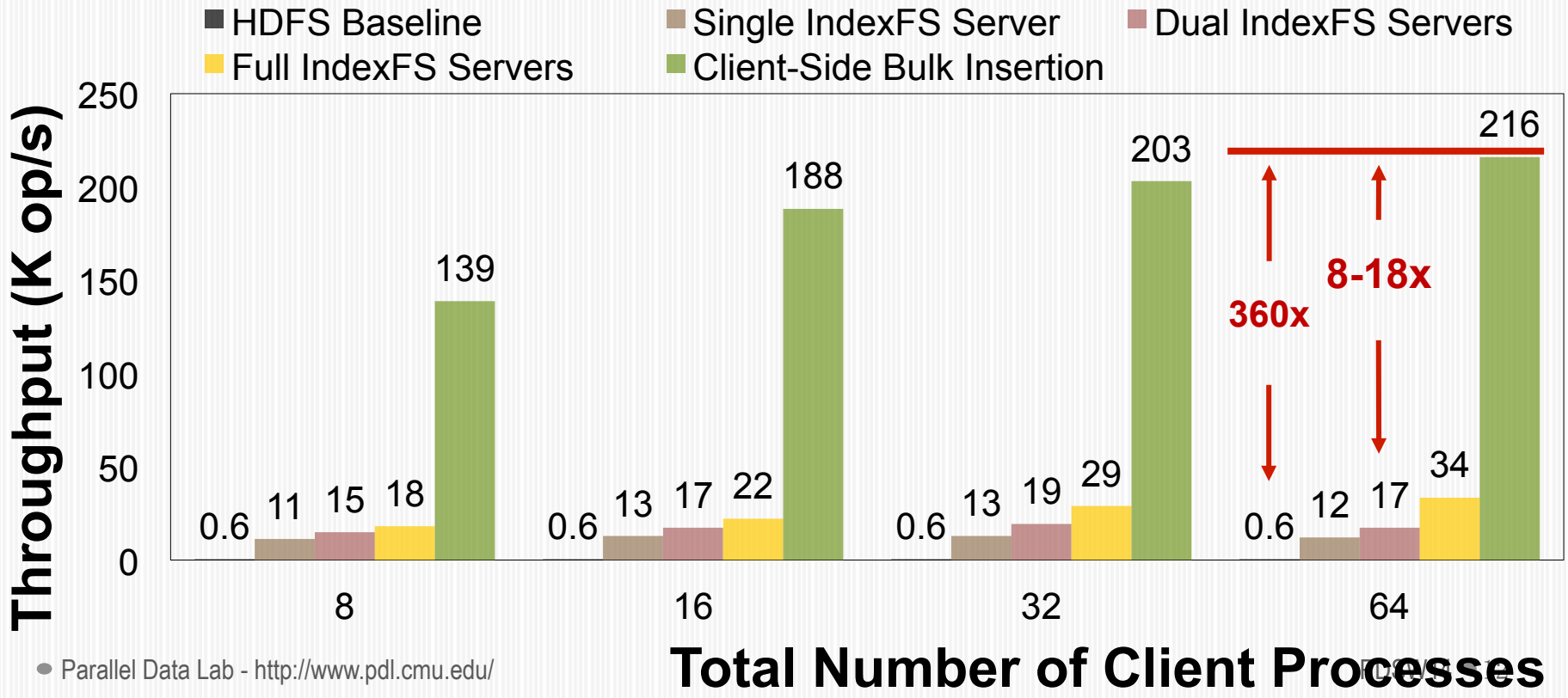
Experiment Setup #3



Experiment Setup #4



8x-360x Perf. Improvements

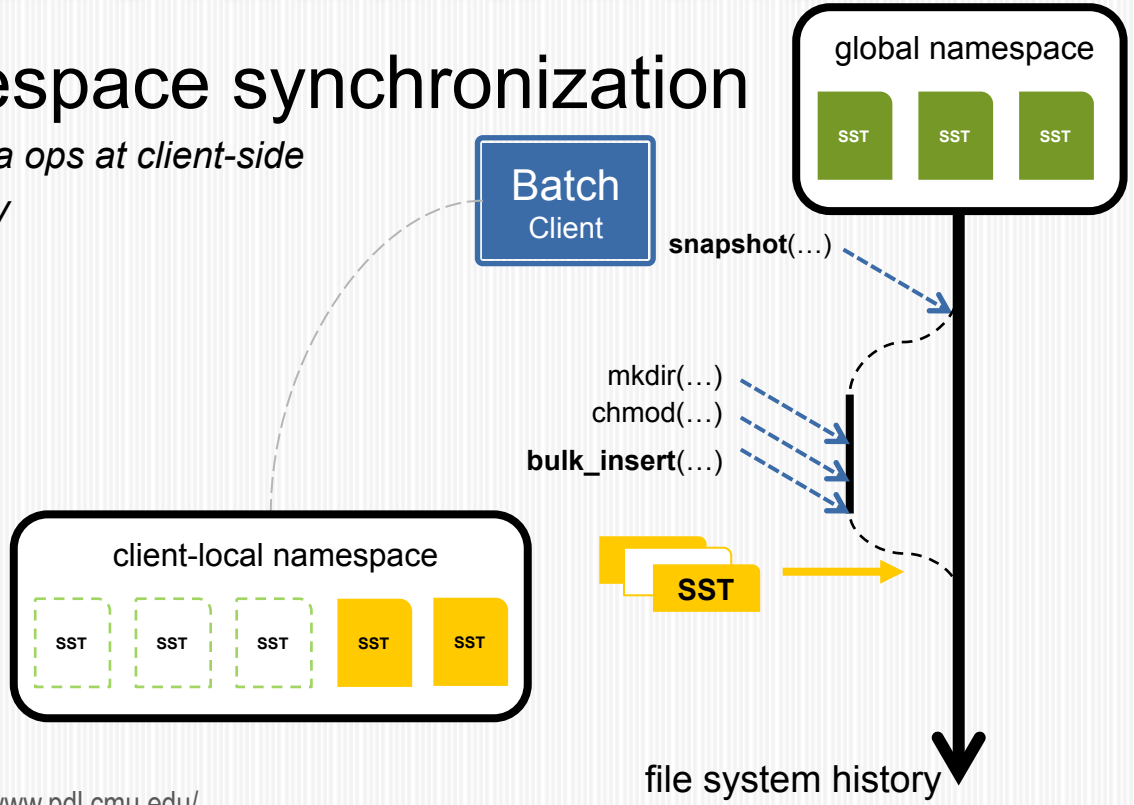


Deep Metadata Batch

Lazy namespace synchronization

Pre-execute metadata ops at client-side

Snapshot consistency



Deep Metadata Batch

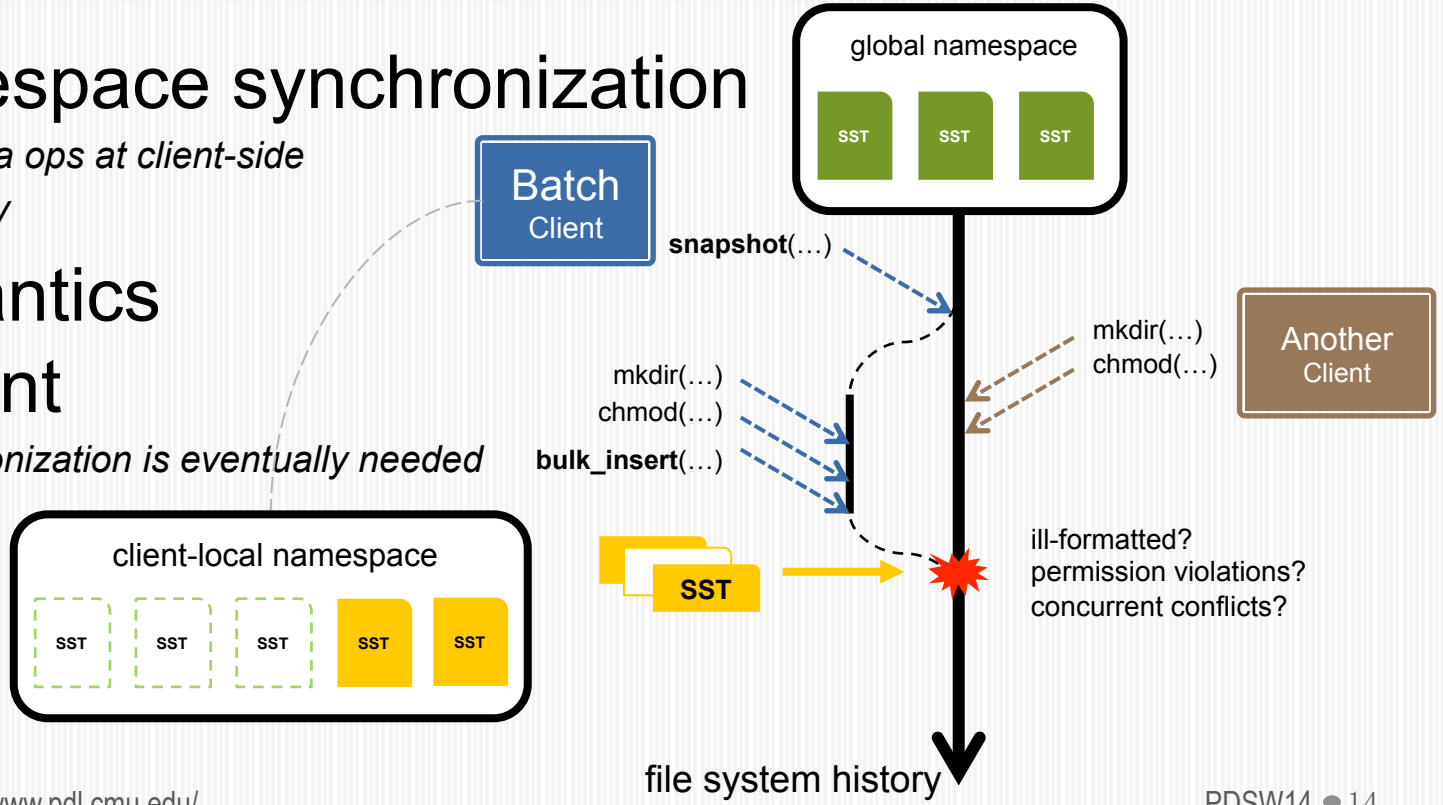
Lazy namespace synchronization

Pre-execute metadata ops at client-side

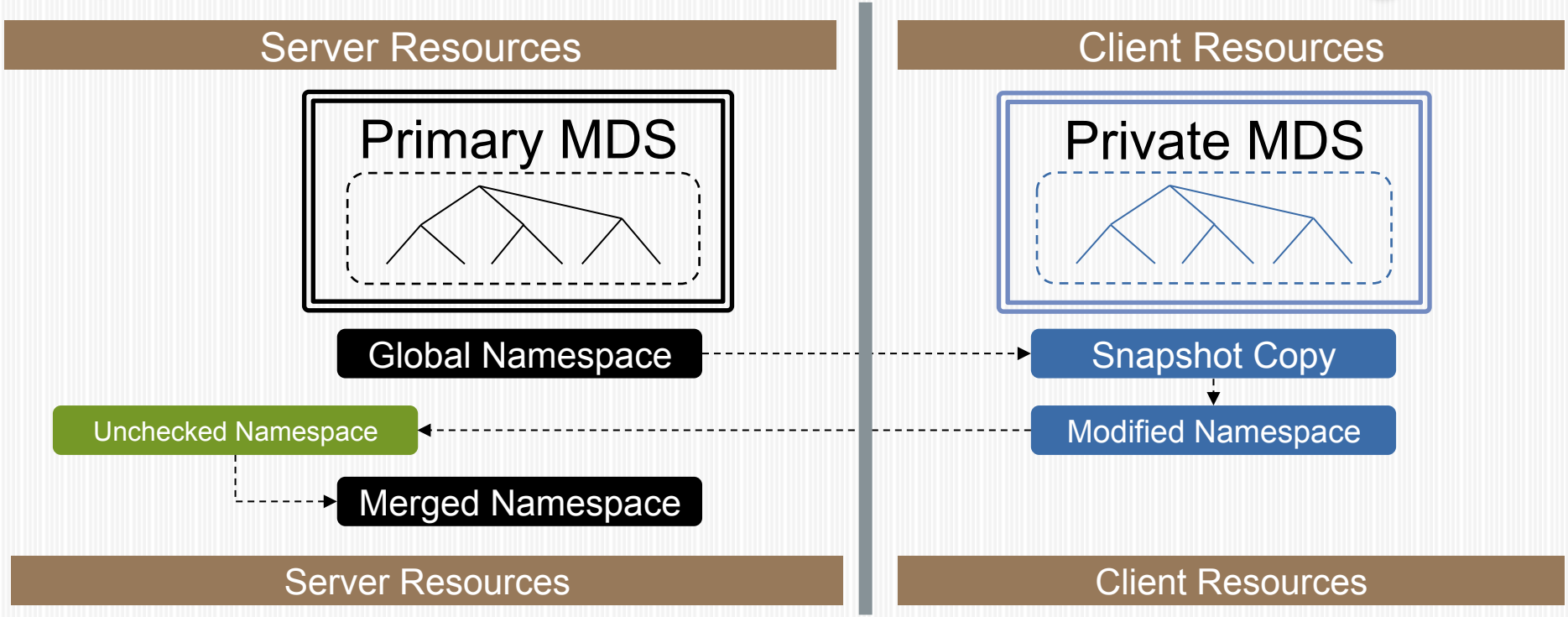
Snapshot consistency

Lazy semantics enforcement

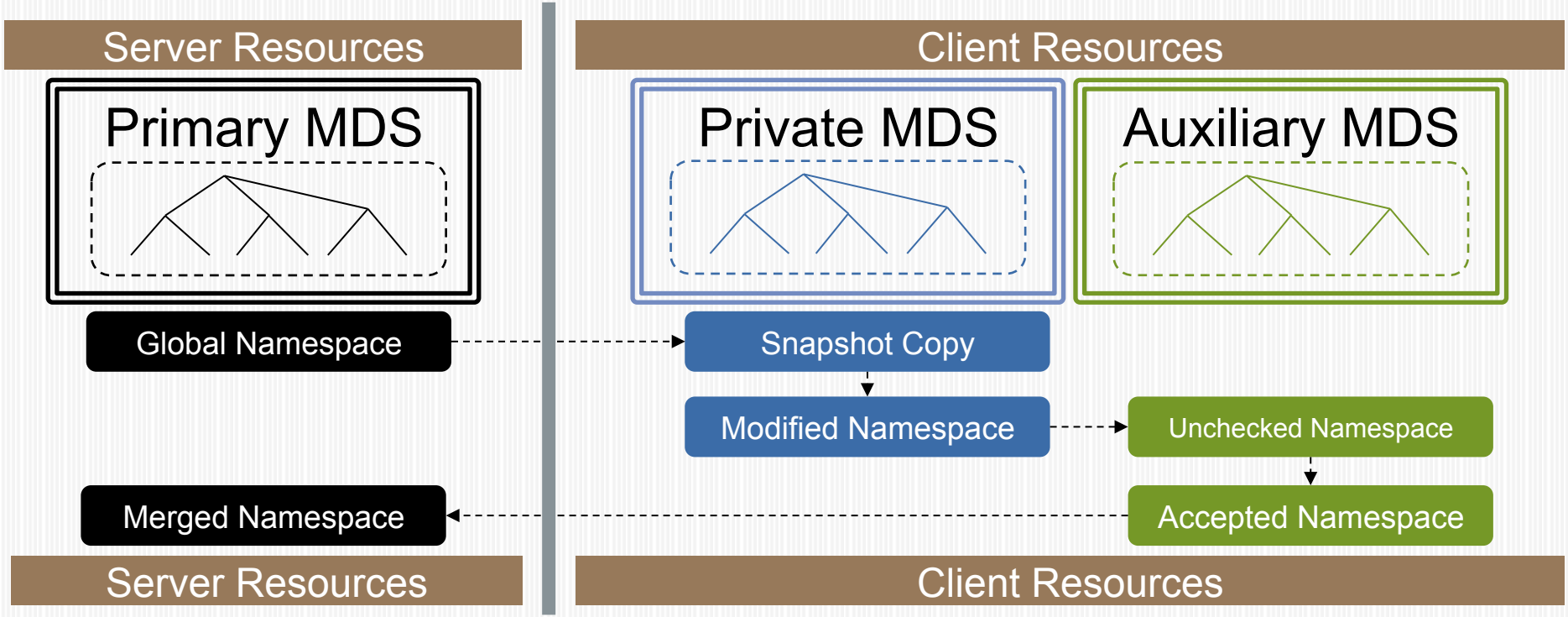
Delayed until synchronization is eventually needed



Client-Funded Metadata Processing



Client-Funded Metadata Verification

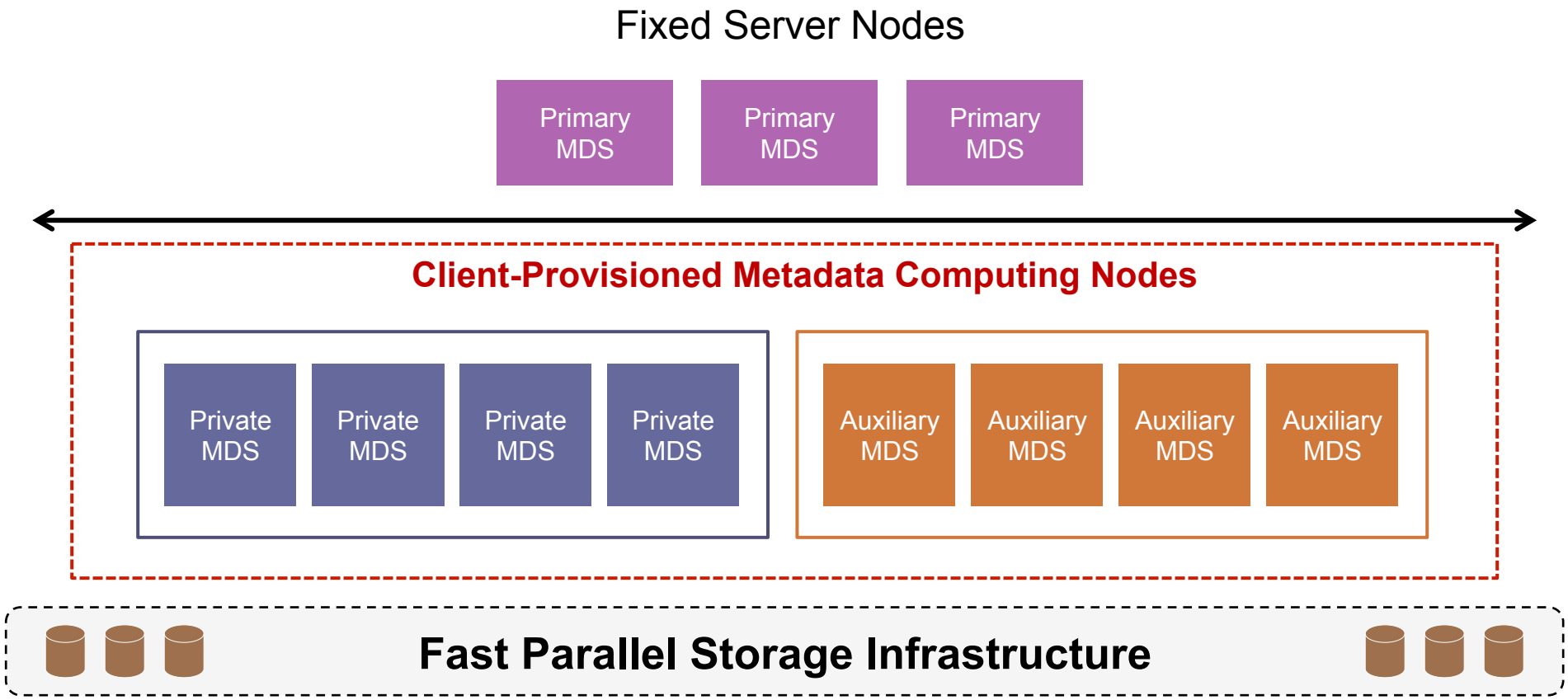


SUMMARY

BATCHFS

At least one RPC per operation	✖
Inefficient metadata representation	✖
Pessimistic concurrency control	✖
Synchronous metadata interface	✖
Dedicated authorization service	✖

BatchFS Architecture Revisited



BatchFS scales with the number of client nodes

Next Challenge: Get rid of Primary MDS servers

Ephemeral File Systems

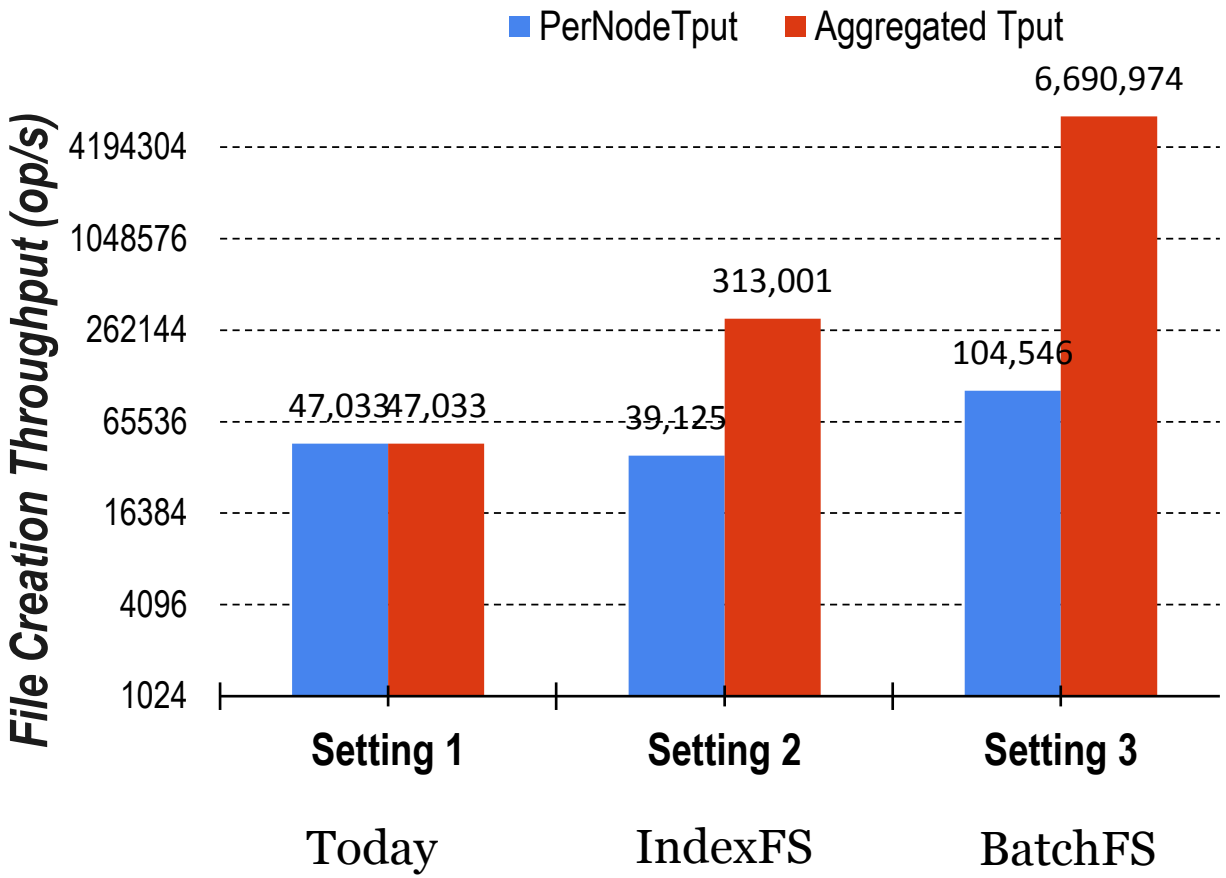
- We can scale object storage
  - Partition into pools tied to storage
- IndexFS/BatchFS built on non-scalable PFS
  - Escape dependence on presence of a PFS
  - Port IndexFS/BatchFS to raw object storage
    - Ceph's RADOS today
    - Replace FS directory with list-of-objects object
- Branching namespace consistency model
  - Parallel job starts with snapshot, builds a branch
  - Workflow/job scheduler passes branch directly
  - Completed workflow publishes branch
  - Curated trunk periodically merges branches

Preliminary RADOS-on-Flash Experiments

	Setting 1	Setting 2	Setting 3
Total Files Created	8,000,000	64,000,000	512,000,000
RADOS: 2-way replicated, built on top of 8x 16GB tmpfs (in total 128GB sto			
1x Head Node	1x ceph-mon 8x indexfs-mds	1x ceph-mon	1x ceph-mon 8x indexfs-mds (primary mds)
8x Data Nodes	32x ceph-osd	32x ceph-osd 64x indexfs-mds	32x ceph-osd
64x Compute Nodes	256x indexfs-clients	1024x indexfs-clients	512x indexfs-bulk-clients (private mds)

Two Orders of Magnitude FASTER

Parallel Metadata-Intensive Job Generating a Namespace Branch



Conclusions and Directions

- Object storage scales data throughput
- Metadata service partitioned over many servers
- Batches/logs/LSM objects encapsulate metadata
- Namespaces as collections of metadata objects
- Decreased frequency of namespace synch
- Middleware execution of namespace mgmt
- Fund scalable metadata from client resources

- Exascale parallel filesystem
  without parallel file systems servers!
- Collaborating with ANL, LANL for Exascale HPC