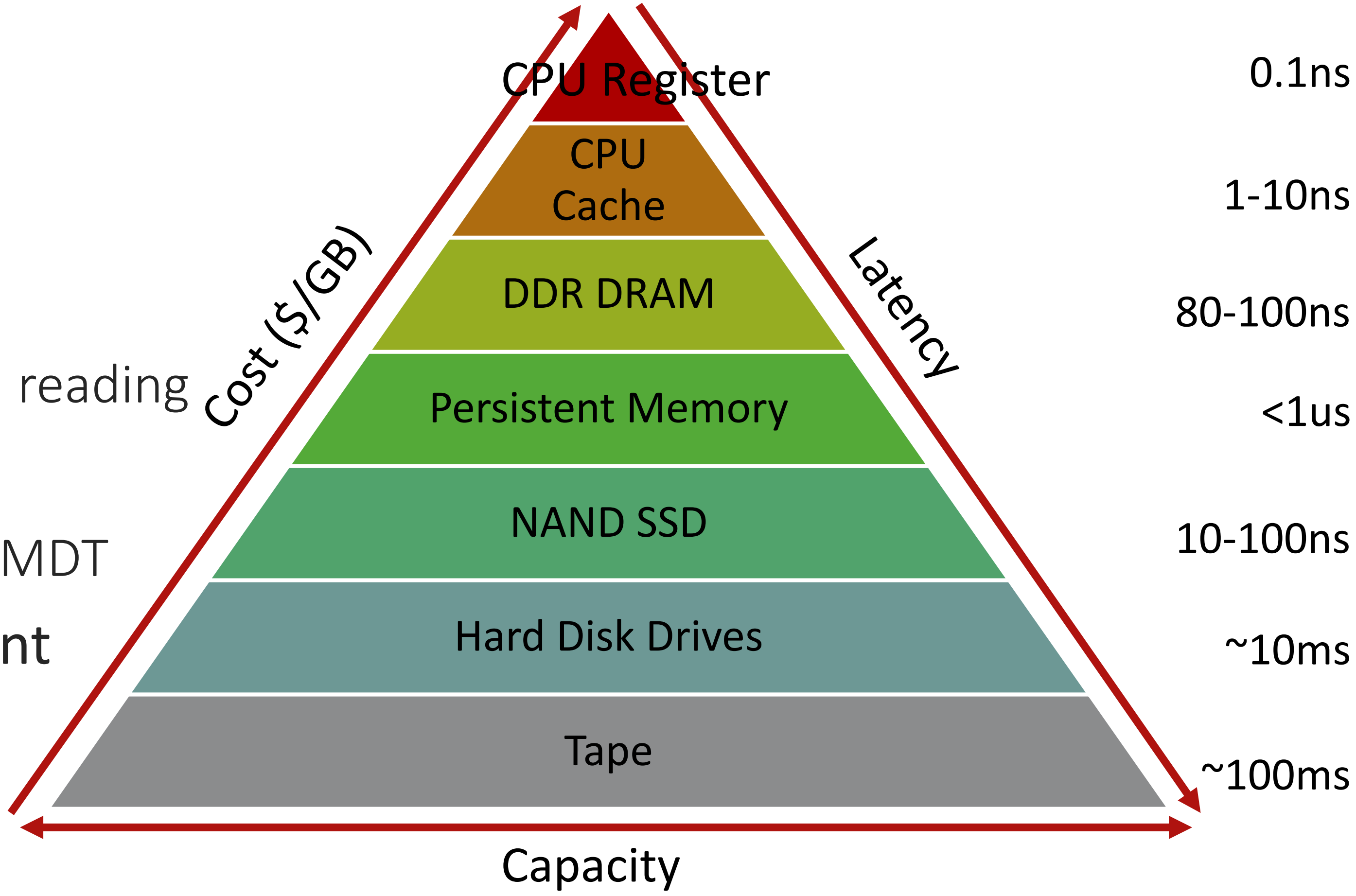
# Lustre Client Metadata Writeback Caching: Design and Implementation

Li Xi

Yingjin Qian

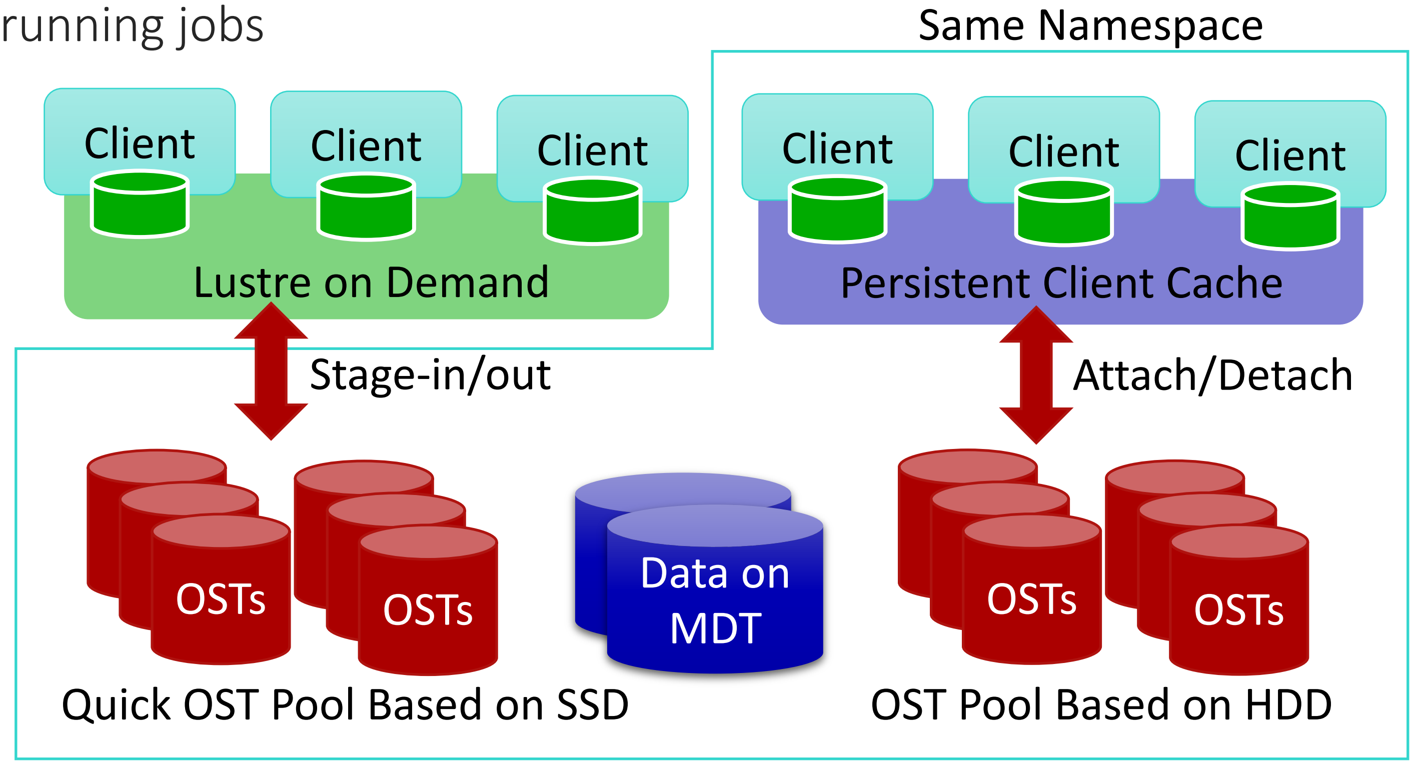
# Why Client Metadata Writeback Caching for Lustre?

- Cache is the key for good performance
  - Page Cache
  - Inode Cache
  - Dentry Cache
- Data is well cached in Lustre
  - Page cache for both data writing and reading
- No cache for changing metadata
  - Each metadata modification goes to MDT
- Metadata performance is important
  - Applications create a lot more files



# Current Data Cache/Acceleration Inside Lustre

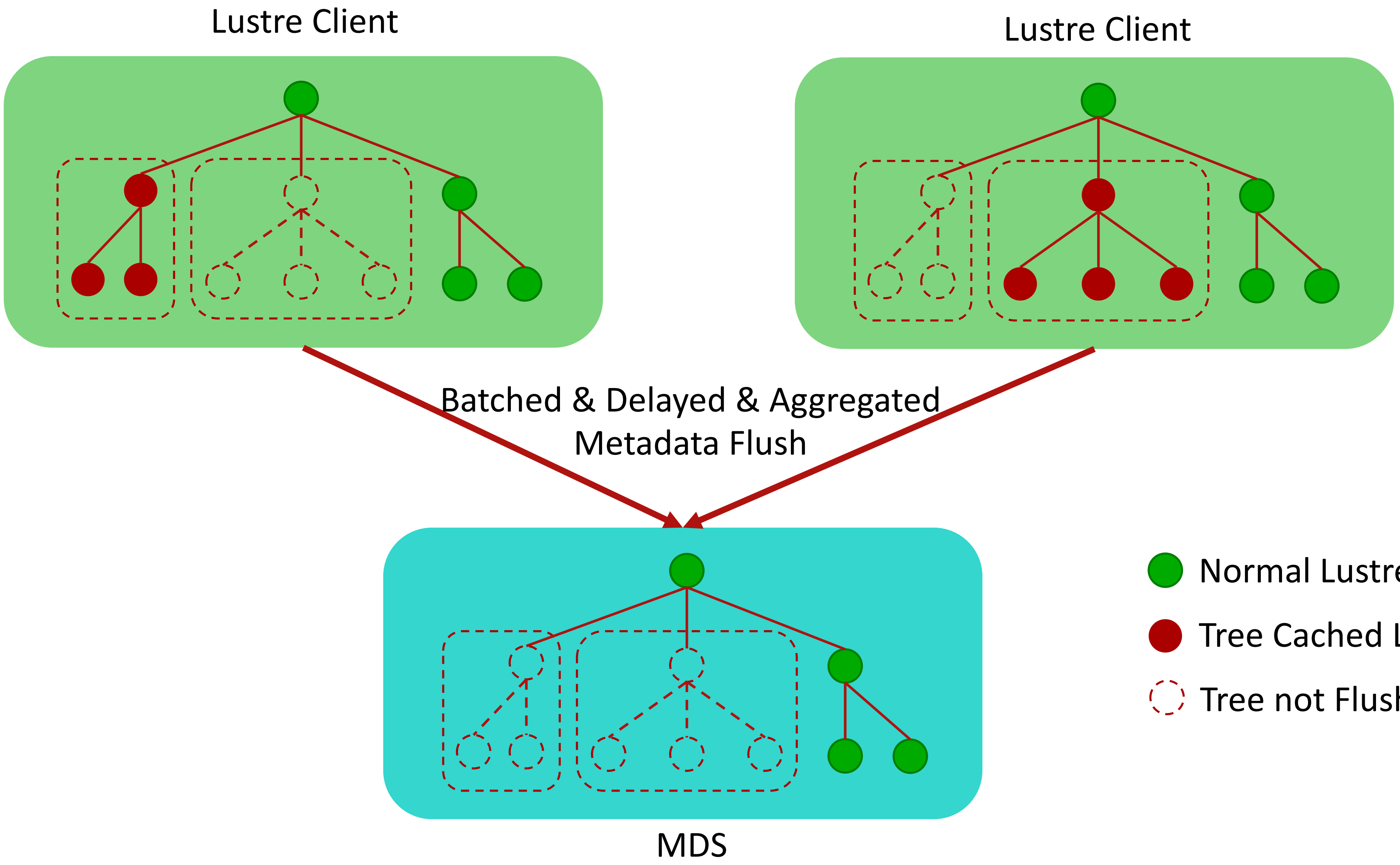
- Persistent Client Cache
  - Local storage on clients for read-only or exclusive files
- Lustre on Demand to cache file sets of jobs
  - Quicker client networks and storage for running jobs
- Data on MDT for data acceleration
  - Less RPC and quick MDT for small files
- OST pool on SSD for cache
  - Quicker OSTs for hot data
- Data reads/writes are fully cached
  - LDLM lock protects data consistency
  - Page level cache management
- Metadata needs acceleration too!



# Main Targets of Lustre WBC

- Client-side cache instead of server-side
  - Pros: higher acceleration caused by metadata locality
  - Cons: complex mechanisms to keep consistency
- Delayed and grouped metadata flush instead of immediate RPC to MDS
  - Pros: much less MDS intervention for better performance
  - Cons: complex mechanisms of batched flush and space/inode reservation
- Cache in volatile memory instead of persistent storage
  - Pros: quickest storage type
  - Cons: complex mechanisms to reduce risk of data loss
- Keeping strong POSIX semantics instead of loosening semantics
  - Pros: transparent acceleration for all applications
  - Cons: complex LDLM lock protection

# General Idea of Lustre WBC



🟢 Normal Lustre Tree

🔴 Tree Cached Locally

◌ Tree not Flushed to MDS yet

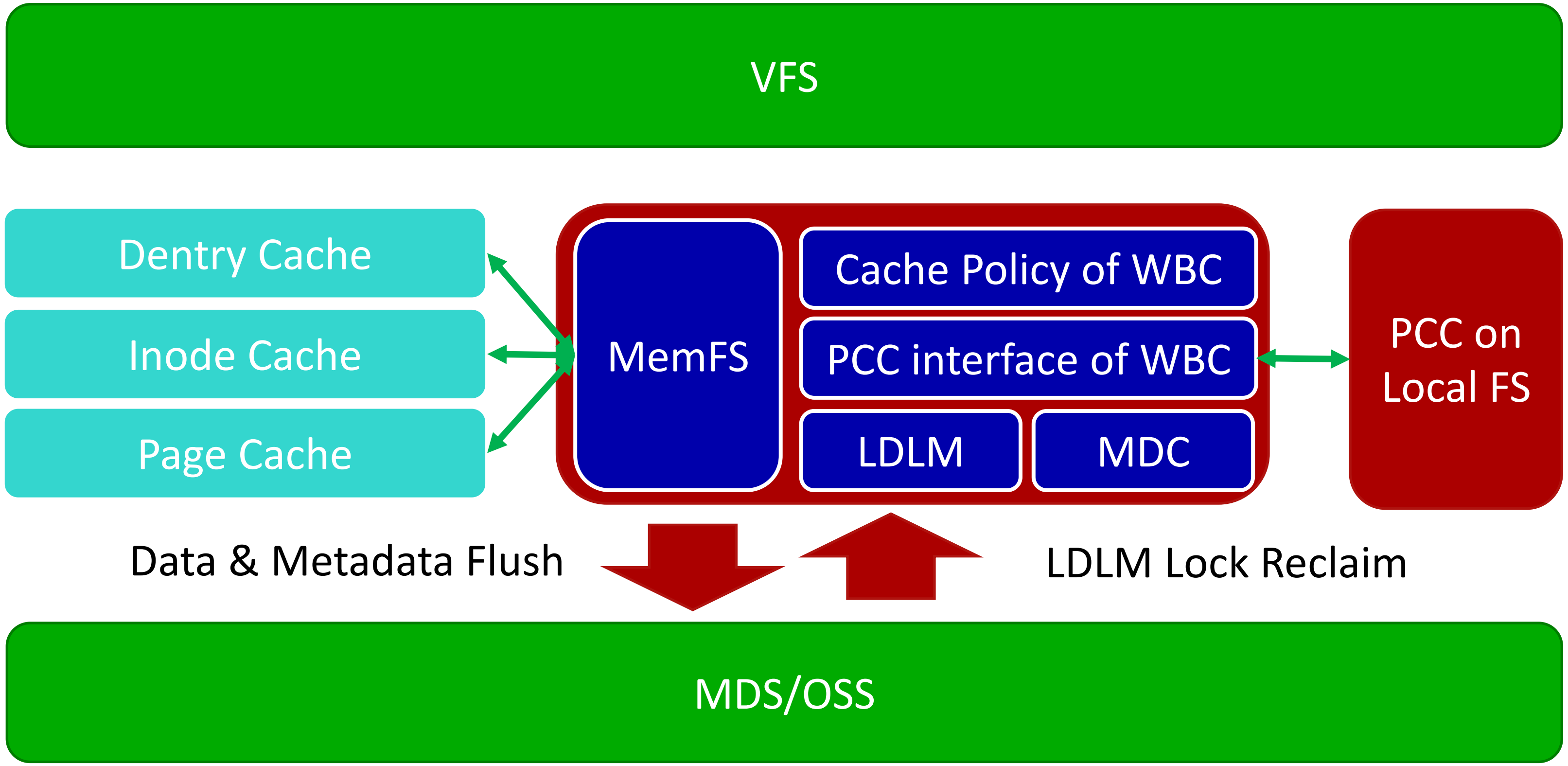
# Design of Lustre WBC (1)

- Directory tree will decide whether to be cached in WBC based on policy when being created
  - User defined rules based on UID/GID/ProjID/fname and their combinations
  - projid={100 200}&gid={1000},uid={500}
  - fname={*.local_dir}
  - Protect the client exclusive access to the entire directory subtree
- Exclusive LDLM lock will be held for root inode of cached directory tree
  - Data/Metada can be then cached safely
- All local modification in the directory tree will be cached
  - Data will be cached in page cache
  - Metadata (inodes/dentries) will be cached in memory too
  - No RPC to MDS/OSS at all

# Design of Lustre WBC (2)

- WBC uses data structure with name of MemFS for cache management
  - Works like Ramfs/Tmpfs but managed by Lustre
  - MemFS manages cached data & metadata
  - MemFS uses inode/dentry/page cache in VFS
- Data and metadata flush happens when:
  - Access of the directory tree from remote clients
  - Memory pressure on local host
  - Periodic auto-flush
- Quick flush from MemFS to MDTs
  - Metadata flushing will use bulk RPC for batched flush
  - Only flush or degrade part of the directory tree rather than whole of it

# Components in Lustre WBC



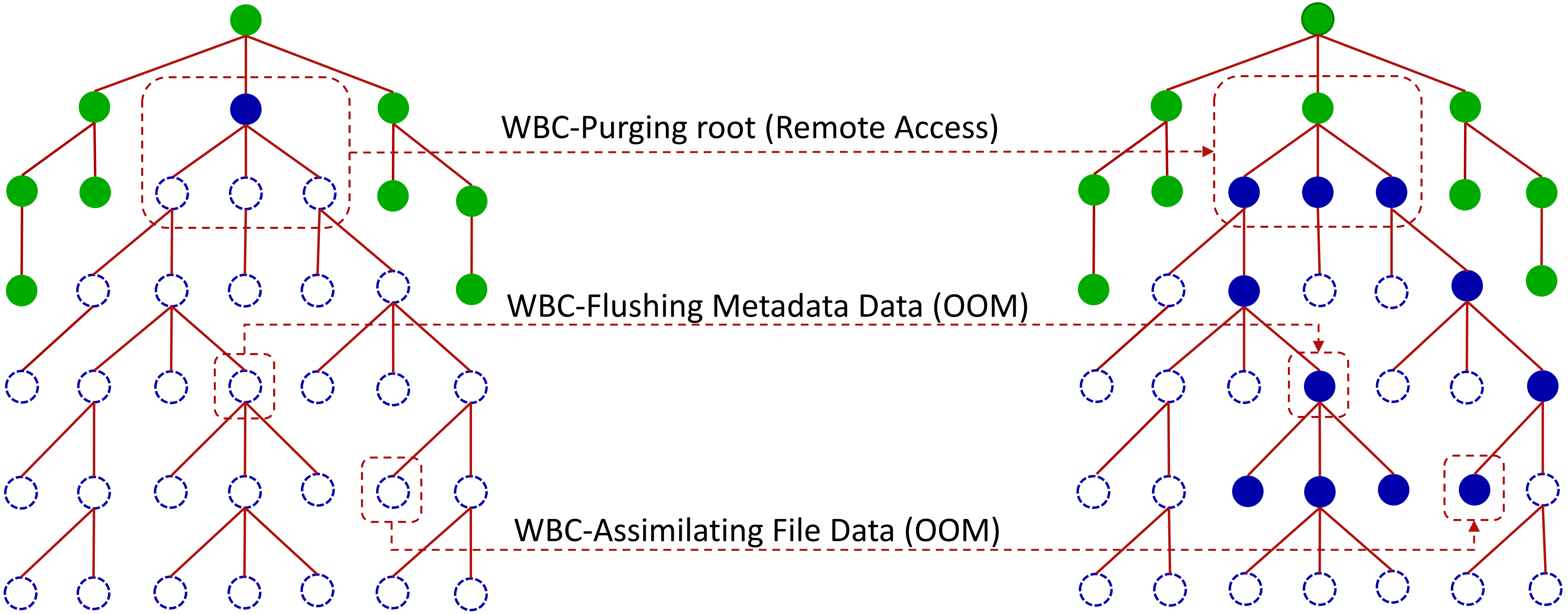
# State Flags of Cached Files/Directories in WBC

- **WBC-Root**: Root of the cached directory tree
  - The exclusive LDLM lock of the tree is being held for this directory
- **WBC-Protected**: File is protected by an exclusive LDLM lock (directly or indirectly)
  - WBC-Root directory is always WBC-Protected
  - Files under WBC-Root directory are WBC-Protected indirectly
- **WBC-Cached**: The children under this directory are fully cached in MemFS
  - Controls whether the metadata operations of the file/dir go to MemFS or MDS
- **WBC-Flushed**: Metadata has been flushed from MemFS to MDS
  - WBC-Root directory is always WBC-Flushed
- **WBC-Assimilated:** Page cache of the file has been assimilated from MemFS to Lustre OSC
- **WBC-None**: None of the above flags is set for normal Lustre files

# Operations to Change WBC States

- **WBC-Purge**: purge the WBC-Root from the WBC
  - Happens when remote client access the WBC-Root
  - WBC-Root flushes metadata, releases exclusive LDLM lock and becomes normal Lustre directory
  - The child directories get exclusive LDLM locks and becomes WBC-Roots
- **WBC-Assimilate:** assimilate the data from WBC to normal page cache of Lustre
  - Happens when need to release memory from cache
  - Metadata of the file and its ancestors need to be flushed first
  - Data is still in page cache of Lustre client, not flushed to OSS yes
- **WBC-Flush:** flush the directory from WBC to MDS and not fully cached any more
  - Happens when need to create a file under the directory but do not have more memory to cache
  - Renaming or creating hardlinks will also trigger WBC-Flush to simplify implementation
  - This directory and its children needs to be flushed back to MDS and remove the WBC-Cached flags

# State Transitions in Different Cases



WBC-Purging root (Remote Access)

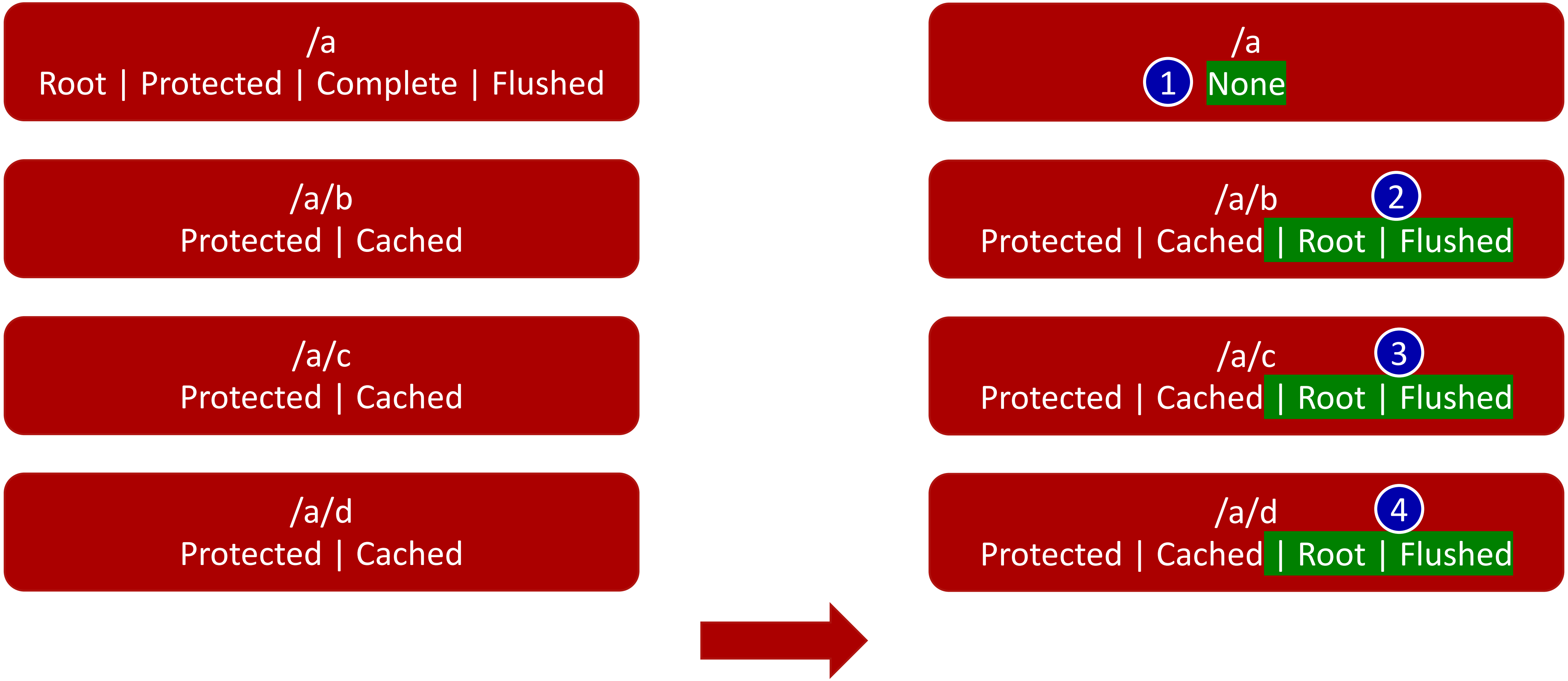
WBC-Flushing Metadata Data (OOM)

WBC-Assimilating File Data (OOM)

🟢 Normal Lustre Tree 🔵 Tree in WBC & Flushed to MDS ◌ Tree in WBC & Not Flushed to MDS

# State Transition when WBC-Purging the WBC-Root

Flags: Newly Added Flags



/a
Root | Protected | Complete | Flushed

/a/b
Protected | Cached

/a/c
Protected | Cached

/a/d
Protected | Cached

/a
1 None

/a/b 2
Protected | Cached | Root | Flushed

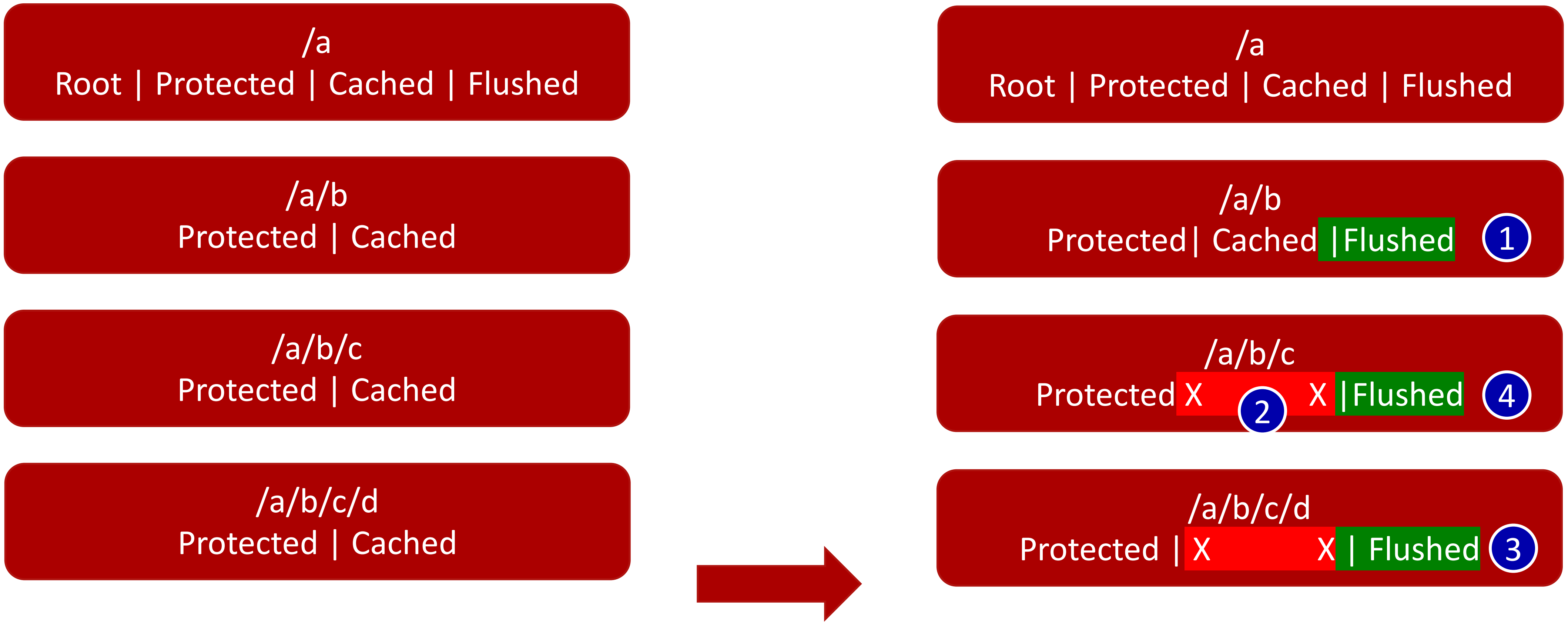
/a/c 3
Protected | Cached | Root | Flushed

/a/d 4
Protected | Cached | Root | Flushed

Remote access of /a ---> WBC-Purge /a/b/c

# State Transition when WBC-Flushing a Directory

Flag: Removed Flags



/a
Root | Protected | Cached | Flushed

/a/b
Protected | Cached

/a/b/c
Protected | Cached

/a/b/c/d
Protected | Cached

/a
Root | Protected | Cached | Flushed

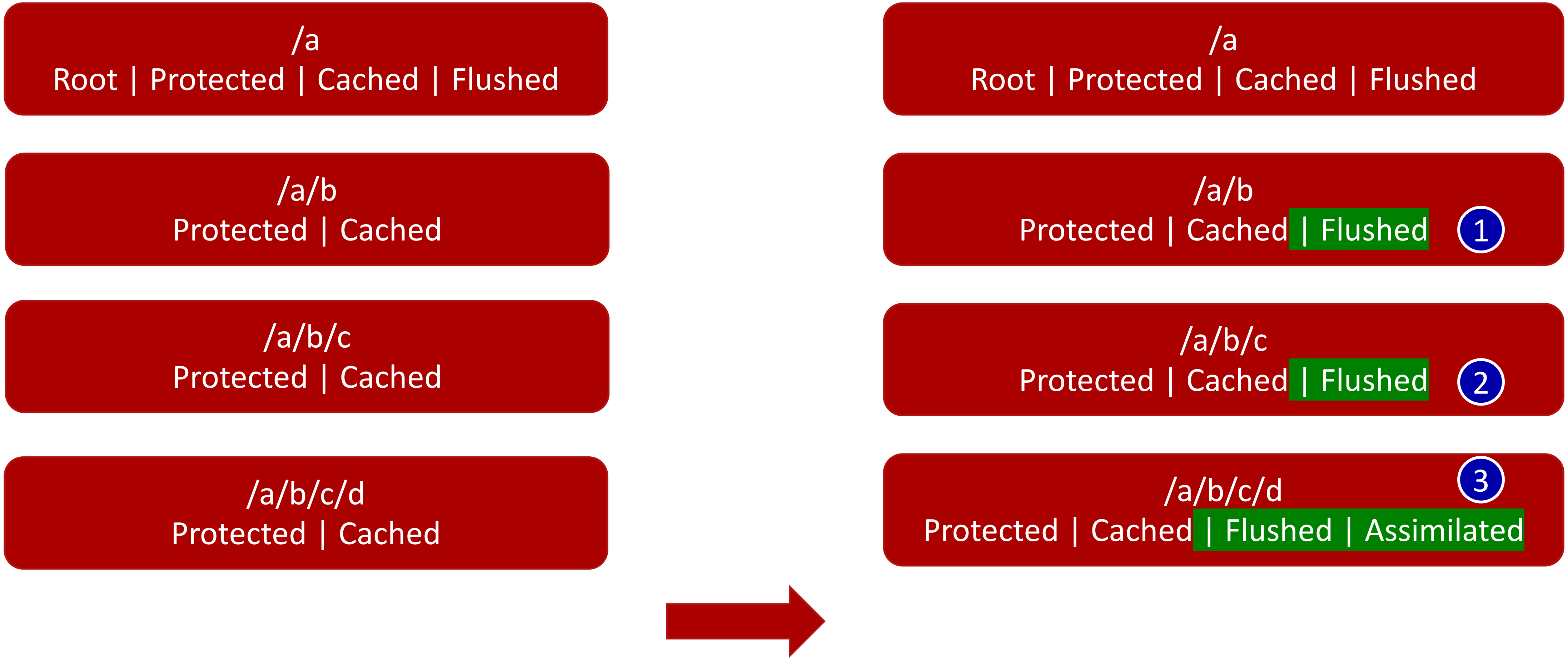
/a/b
Protected| Cached |Flushed 1

/a/b/c
Protected X 2 X |Flushed 4

/a/b/c/d
Protected | X X | Flushed 3

OOM when creating /a/b/c/e on MemFS ---> WBC-Flush /a/b/c

# State Transition when WBC-Assimilating File Data



/a
Root | Protected | Cached | Flushed

/a/b
Protected | Cached

/a/b/c
Protected | Cached

/a/b/c/d
Protected | Cached

/a
Root | Protected | Cached | Flushed

/a/b
Protected | Cached | Flushed 1

/a/b/c
Protected | Cached | Flushed 2

3
/a/b/c/d
Protected | Cached | Flushed | Assimilated

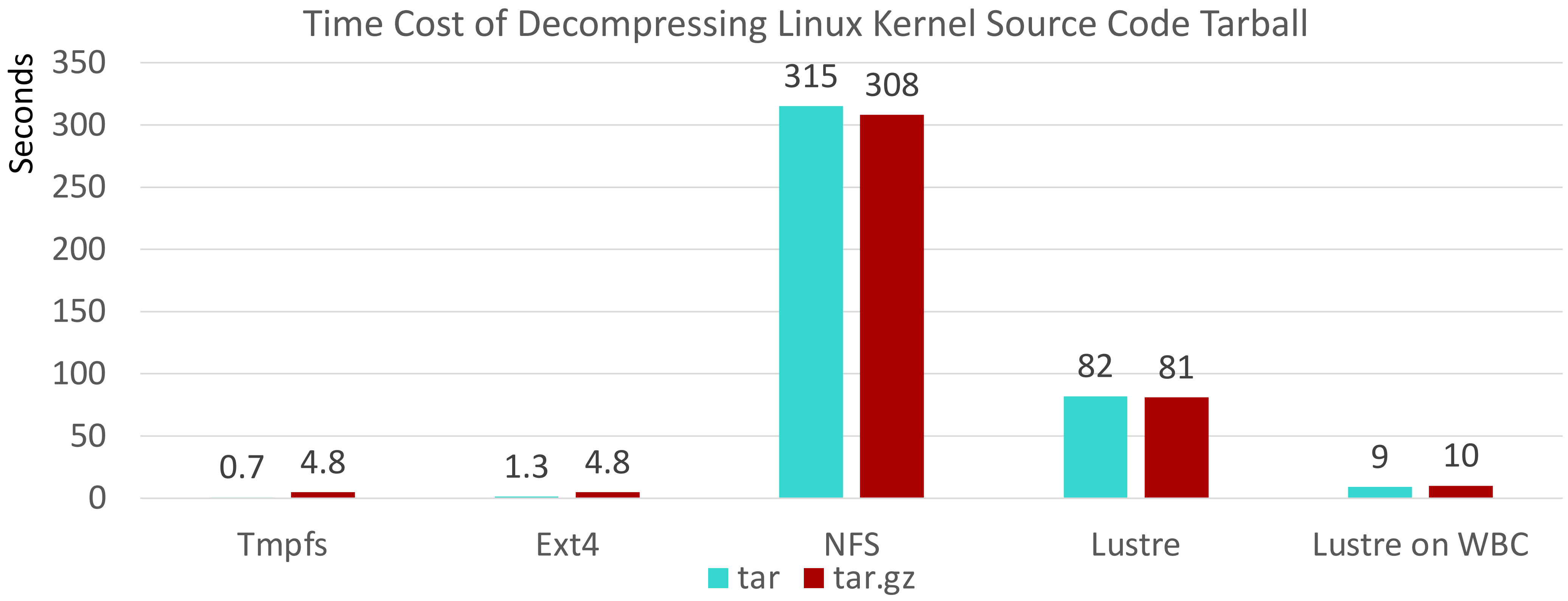
Assimilate Data of /a/b/c/d

# Features and Advantages of WBC

- WBC flushes metadata of files in batch
  - > 1000 updates on files in a single bulk RPC
- Batch operations of metadata can be used to delete a whole directory
  - Accelerates “rm -fr” a lot
- WBC aggregates metadata updates
  - Only the final state of metadata will be flushed to MDS
  - create() + chattr() + chmod() + unlink() = No RPC to MDS
- WBC can be integrated with PCC
  - Data will still be cached in PCC after WBC-Assimilation
  - Cache more data on client
  - More memory for metadata caching
- Possible offline/disconnected operations on Lustre client

# Untar Performance of WBC Against Other File Systems

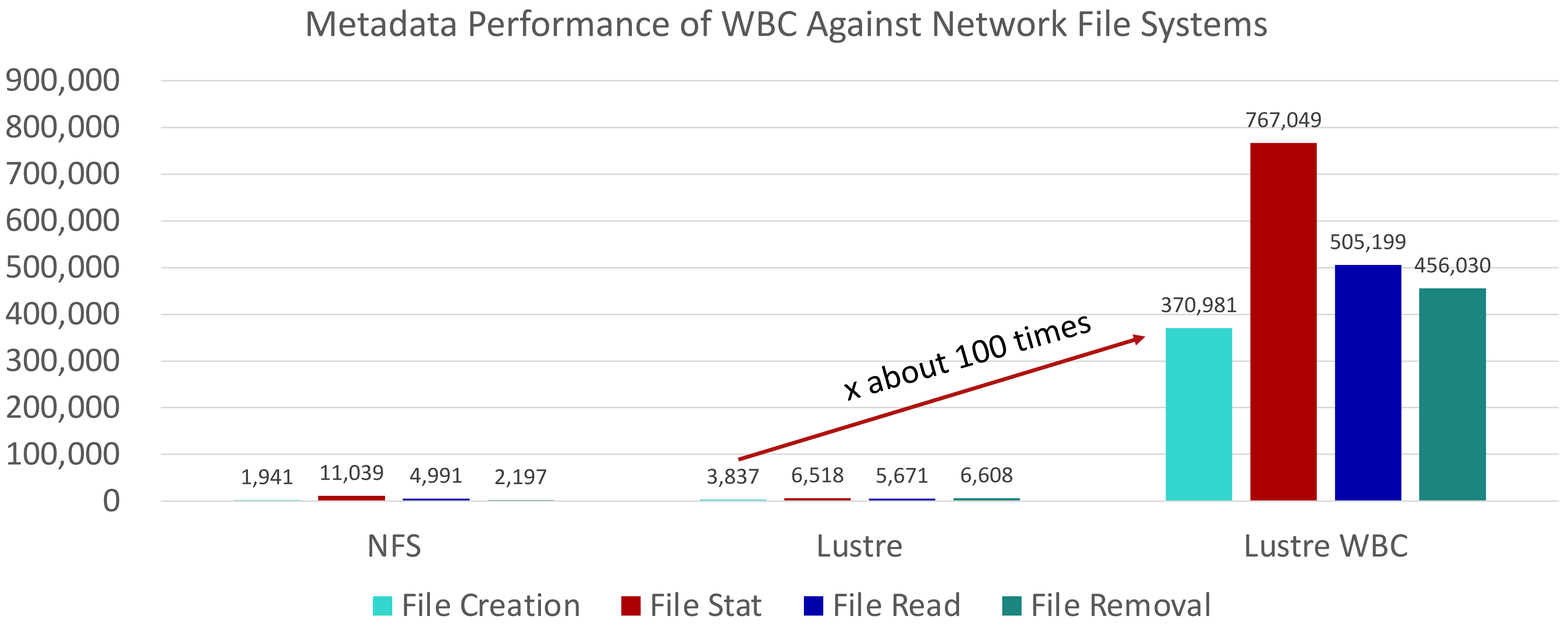
Lustre: DDN AI400X Appliance (20 X SAMSUNG 3.84TB NVMe, 4X IB-HDR100)
Lustre client: Intel Gold 5218 processor, 96 GB DDR4 RAM, CentOS 8.1
Local File System on SSD: Intel SSDSC2KB240G8



# Metadata Performance of WBC Against Network File Systems

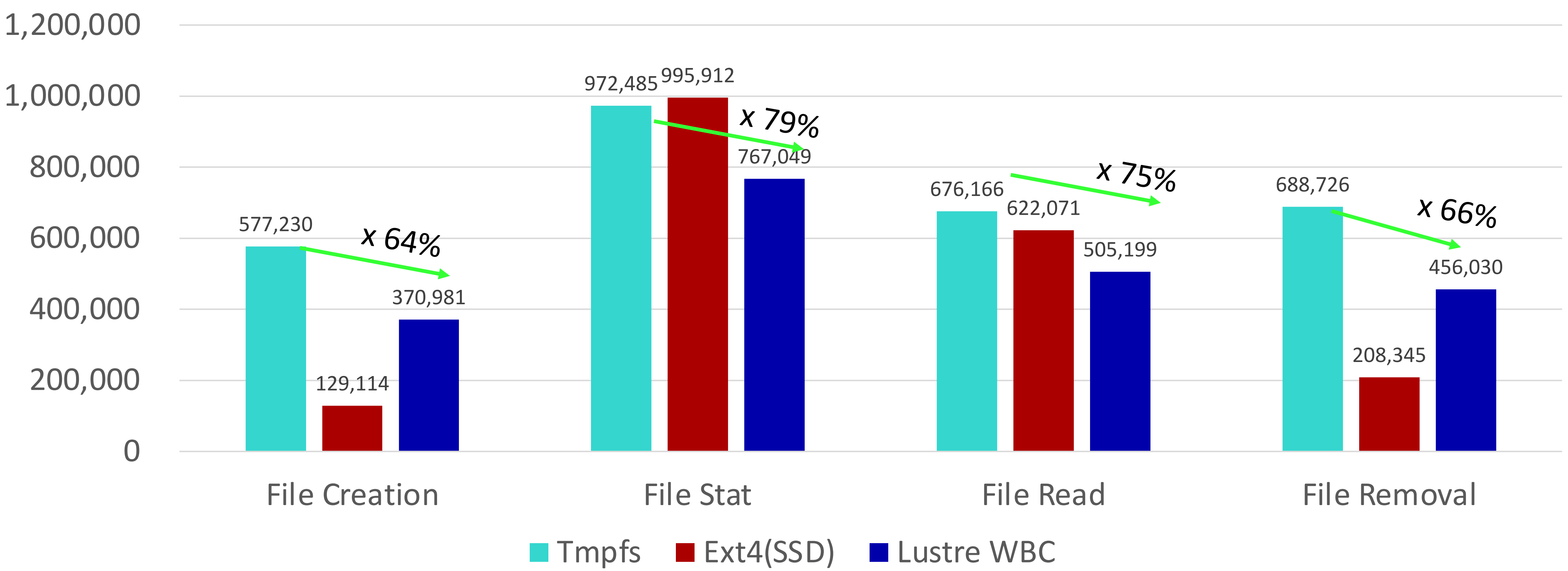
Whamcloud

Lustre: DDN AI400X Appliance (20 X SAMSUNG 3.84TB NVMe, 4X IB-HDR100)
Lustre client: Intel Gold 5218 processor, 96 GB DDR4 RAM, CentOS 8.1
Local File System on SSD: Intel SSDSC2KB240G8
Benchmark Commands: mdtest -n 200000 -d $DIR



# Metadata Performance of WBC Against Local File Systems

Lustre: DDN AI400X Appliance (20 X SAMSUNG 3.84TB NVMe, 4X IB-HDR100)
Lustre client: Intel Gold 5218 processor, 96 GB DDR4 RAM, CentOS 8.1
Local File System on SSD: Intel SSDSC2KB240G8
Benchmark Commands: mdtest -n 200000 -d $DIR



Metadata Performance of WBC Against Local File Systems

Whamcloud

Thank you!

DDN®
STORAGE