LUSTRE USER GROUP 2023

SAN DIEGO SUPERCOMPUTER CENTER

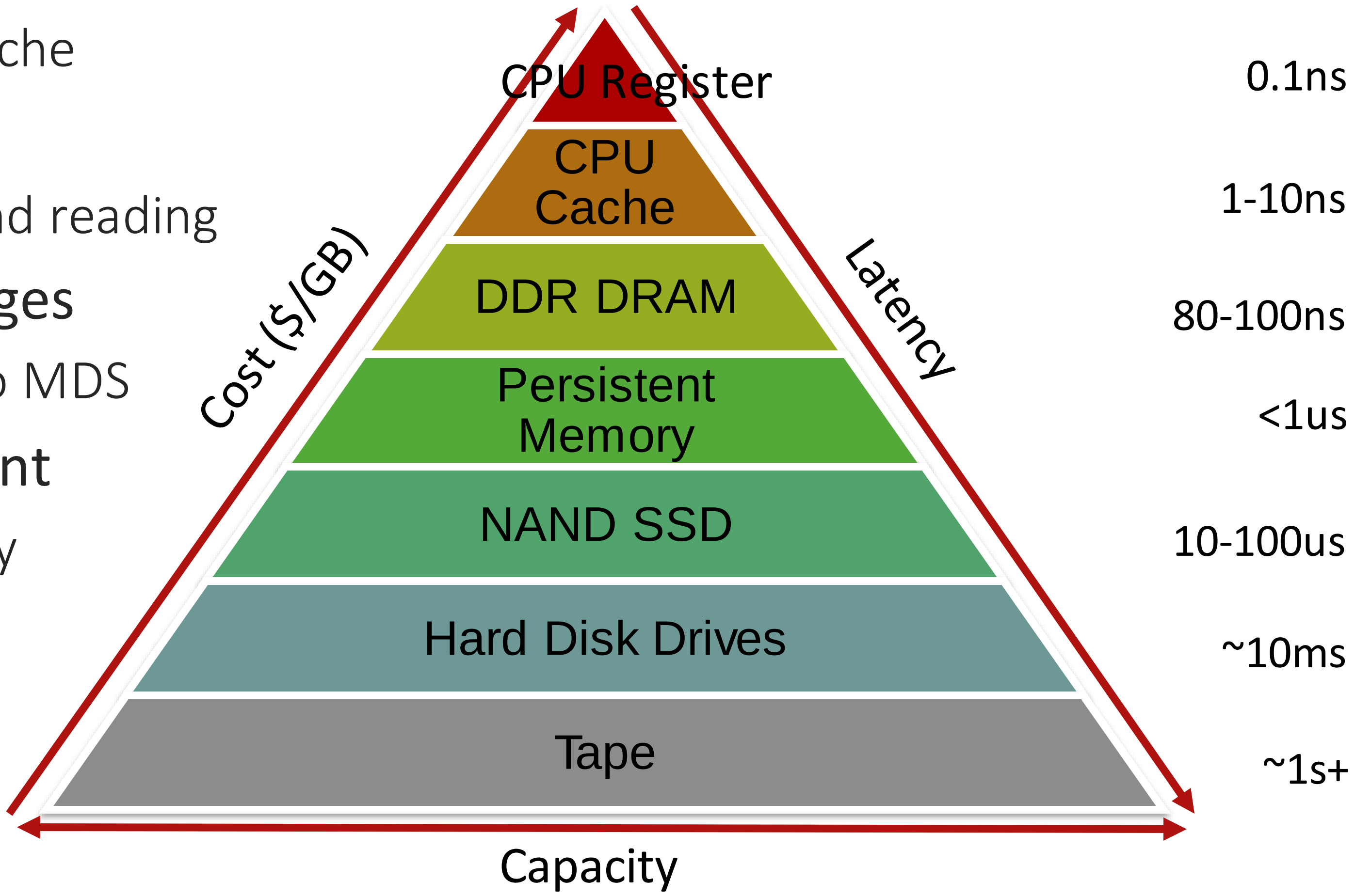
Lustre Metadata Writeback Cache

Design and implementation

Yingjin Qian, Oleg Drokin

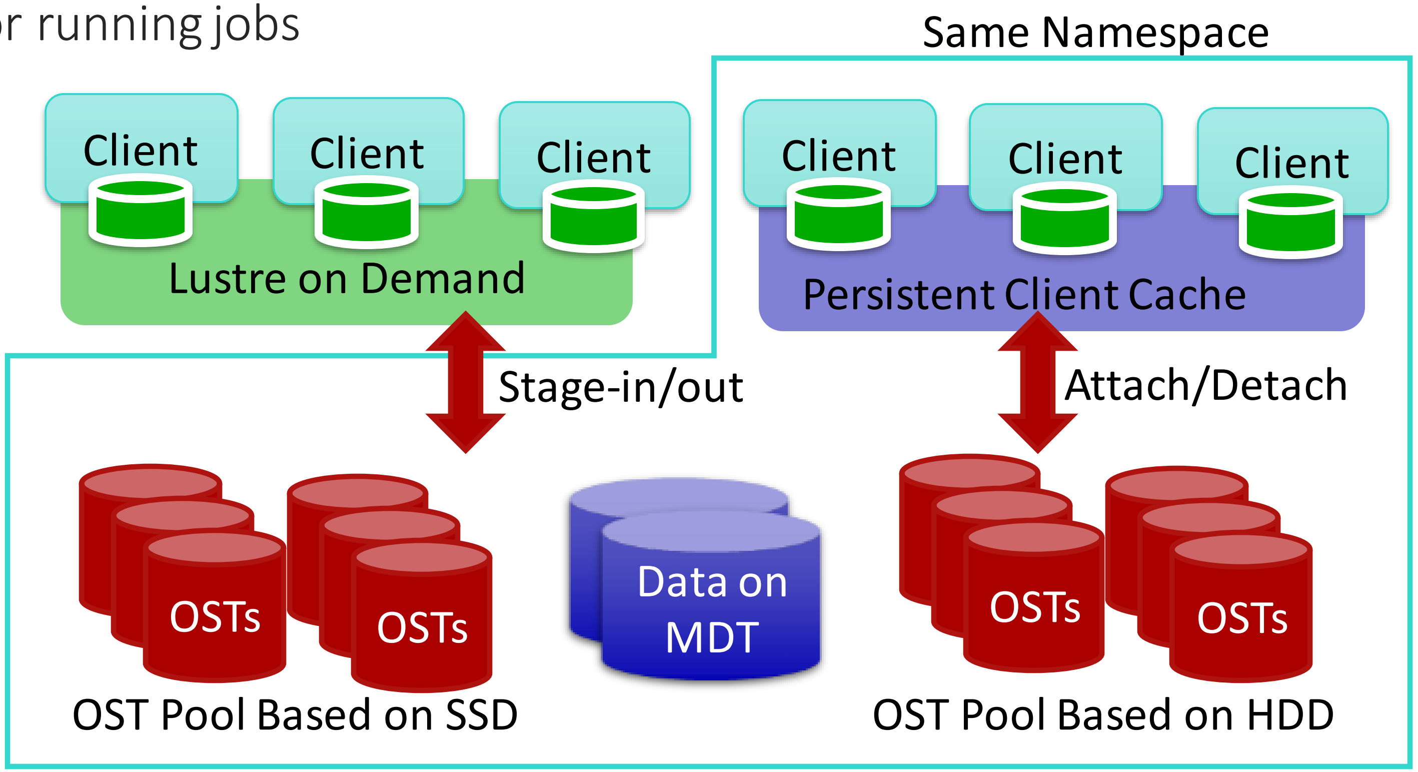
Why (Metadata) Write Back Caching for Lustre?

- 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 client cache for metadata changes
  - Each metadata modification sent to MDS
- Metadata performance is important
  - Applications create many files today
  - Millions of RPCs sent over network



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 read/write are fully cached
  - LDLM lock protects data consistency
  - Page level cache management
- Metadata needs acceleration too!



When Metadata is Nothing Special

- Shared block filesystems have it easy – metadata is just data
  - Client locks the block(s), reads and interprets the contents
  - Perfect cache for both reads and modifications, just like a local filesystem
  - Crumbles under contention as lock-read-modify-write-unlock cycles get expensive fast
- To address the contention various tricks are played
  - Various libraries embed subdirectory trees inside specially formatted files are common
  - Minimize roundtrips by trying to send updates directly between clients (GPFS)
  - Complicate matters by reducing lockable block size

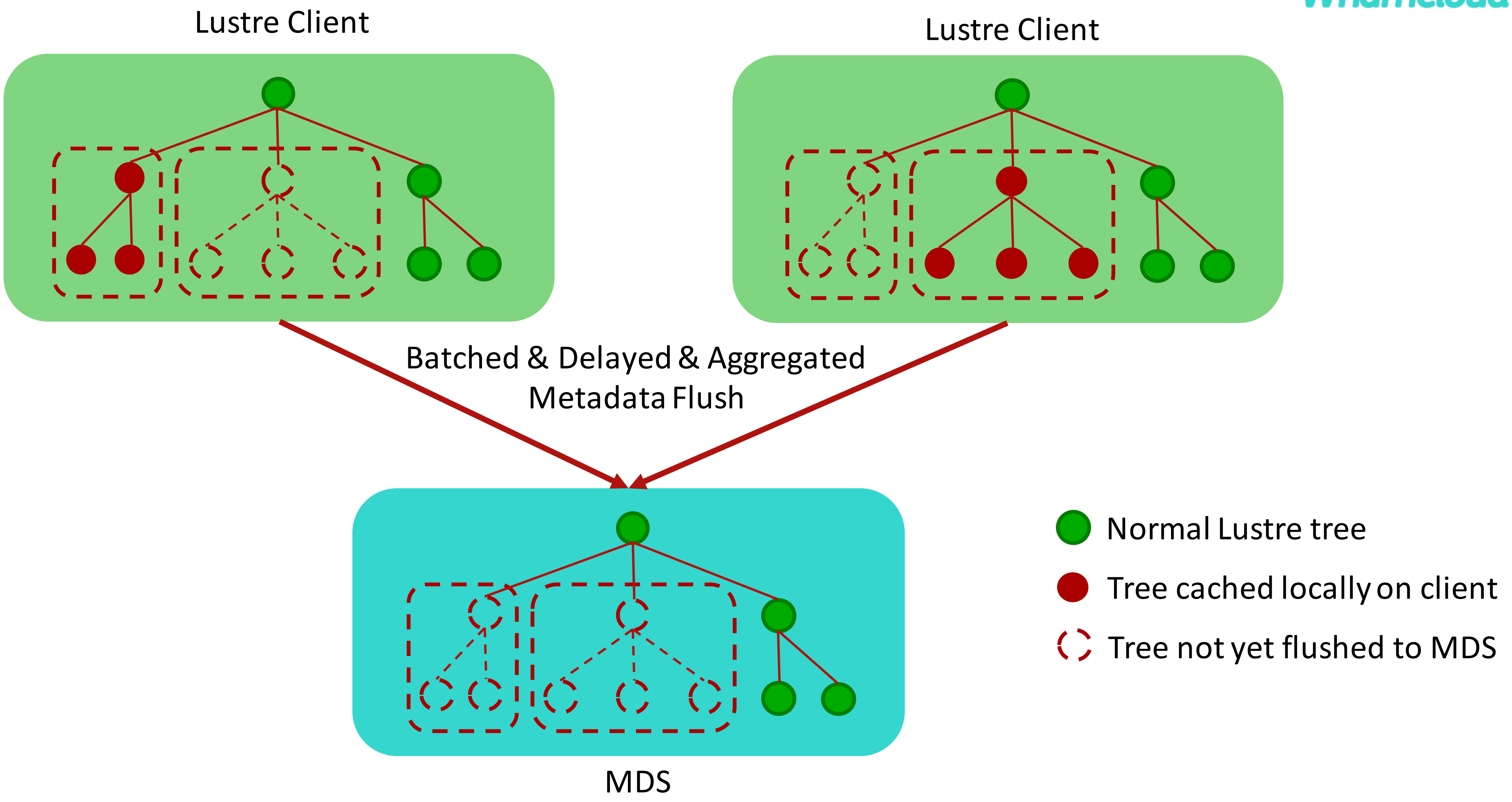
When Metadata is Unique

- Lustre is not a shared block filesystem
  - Metadata is interpreted on the server
  - Client receives piecemeal bits, that allows each one to be locked/cached separately
  - Changes are sent piecemeal, no need to read entire directory to create a new name
  - This gets expensive when there is no contention
- To address uncontended cases some tricks are played
  - Block-based images of filesystems for “filesets” that are separately mounted (CCI)

When Metadata is Just the VFS

- Exclusive lock at the root of subtree
  - The subtree could be populated by new creates (common)
  - Or reading data from the server
- All the operations then become node local
  - It’s like a shared block filesystem without a block underneath
  - Super low latency
- Granularity of this lock is “whole subtree underneath”

General Idea of Lustre WBC



Batching of Metadata Modifying Operations

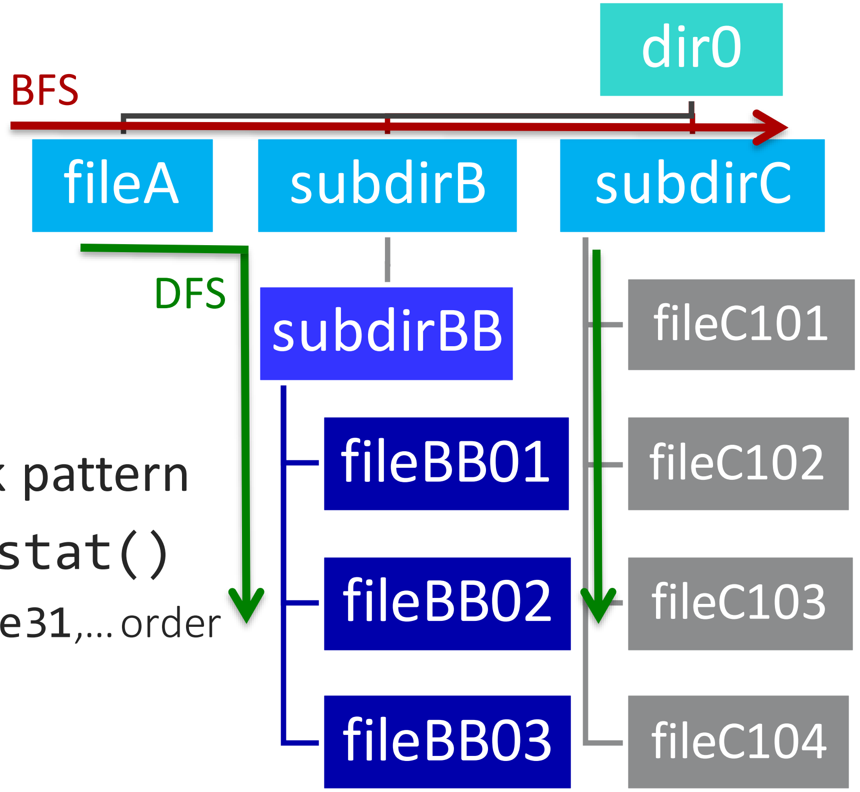
- Now with full cache of modifications we can also batch updates
- One RPC brings along many changes to the server
- Some updates could be coalesced locally and even reduced to nothing

`touch file; chmod o-r file; mv file file2; rm file2` => no RPCs

- Some audit folks might not be happy about this though

Batching of Read Operations

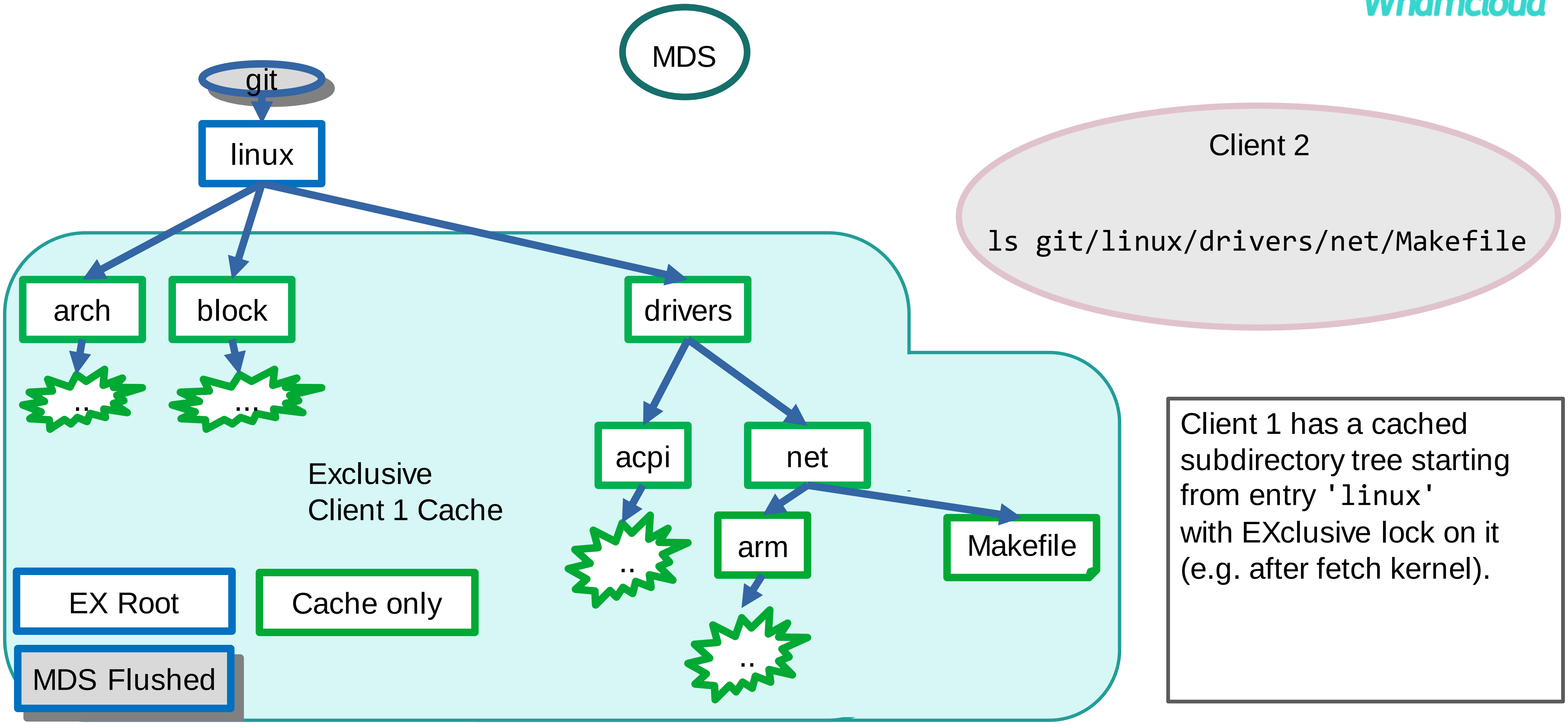
- Lustre already has a "statahead" metadata readahead
- Makes a good natural first step to showcase batched RPC functionality
- Will plug into the batched RPC mechanism nicely
- Coming in Lustre 2.16
- Will aggregate getattr RPCs for statahead
- Detects breadth-first (**BFS**) or Depth-first (**DFS**)
  - Direct statahead to next file/subdirectory based on tree walk pattern
  - Detect strided pattern for alphanumeric ordered traversal + `stat()`
    - e.g. `file00001`,`file001001`,`file002001`... or `file1`,`file17`,`file31`,... order
- IO500 `mdtest-{easy/hard}-stat` performance improved 77%/95%



Handling Contention in a Cached Directory Tree

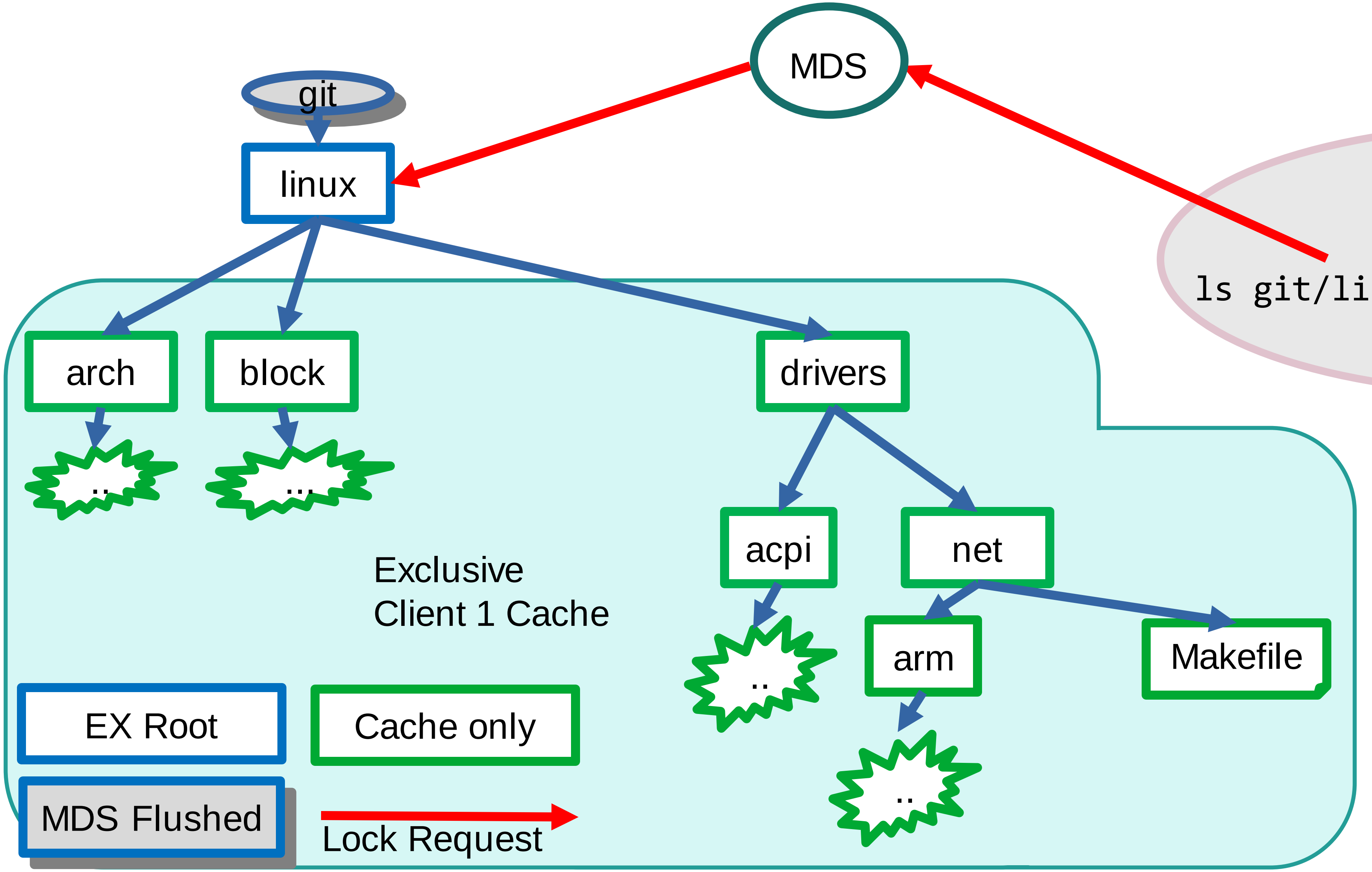
- When second client tries to access files in the cached directory tree
- Bump into EXclusive lock at top level of the tree
- Lock holder is asked to release the lock
- Flushes top level of entries, obtaining EX locks on new subdirectories
- Another client can now see and descend into next subdirectory level
- Repeat as needed for second client to access subdirectory treee
- No need to flush entire subtree at once to have global visibility

Contention and Global Visibility



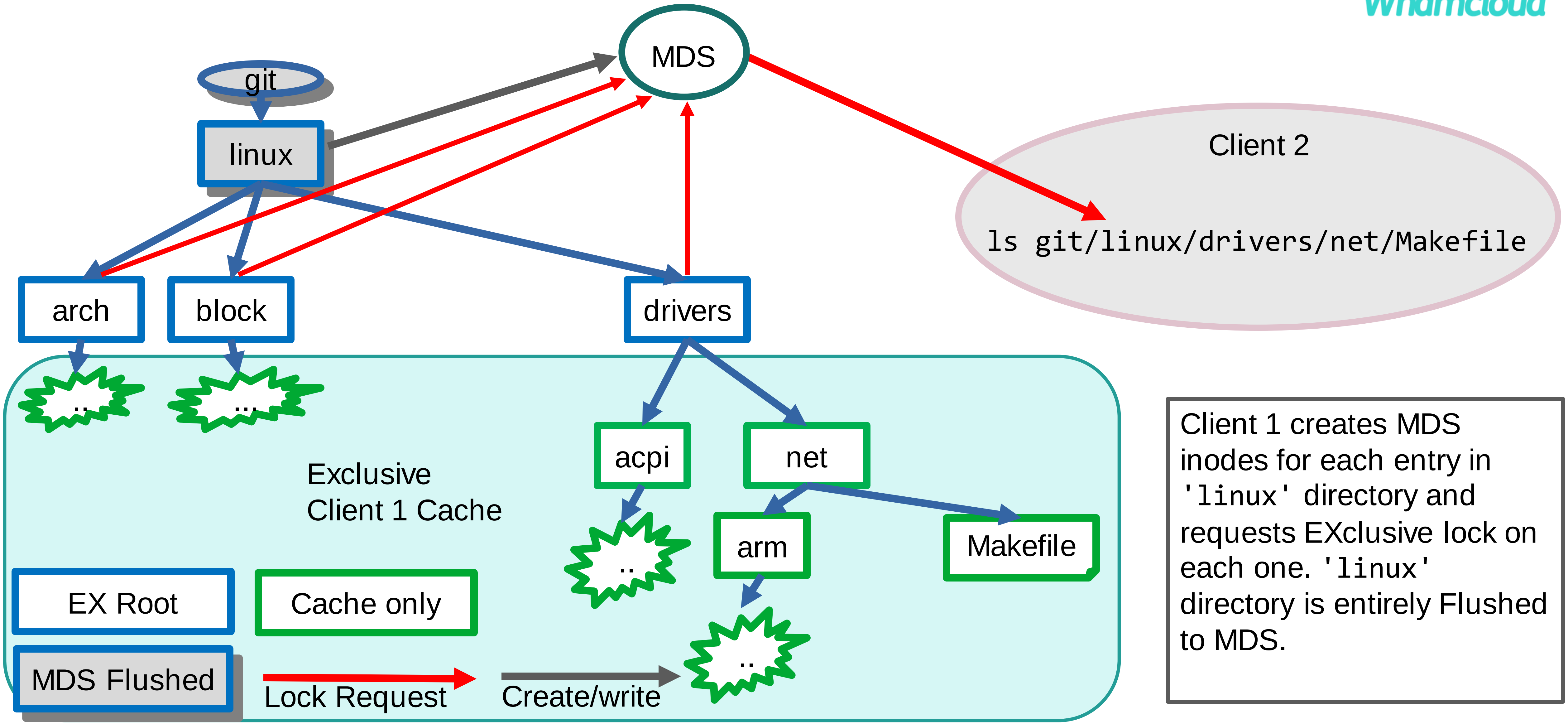
Client 1 has a cached subdirectory tree starting from entry `'linux'` with EXclusive lock on it (e.g. after fetch kernel).

Contention and Global Visibility



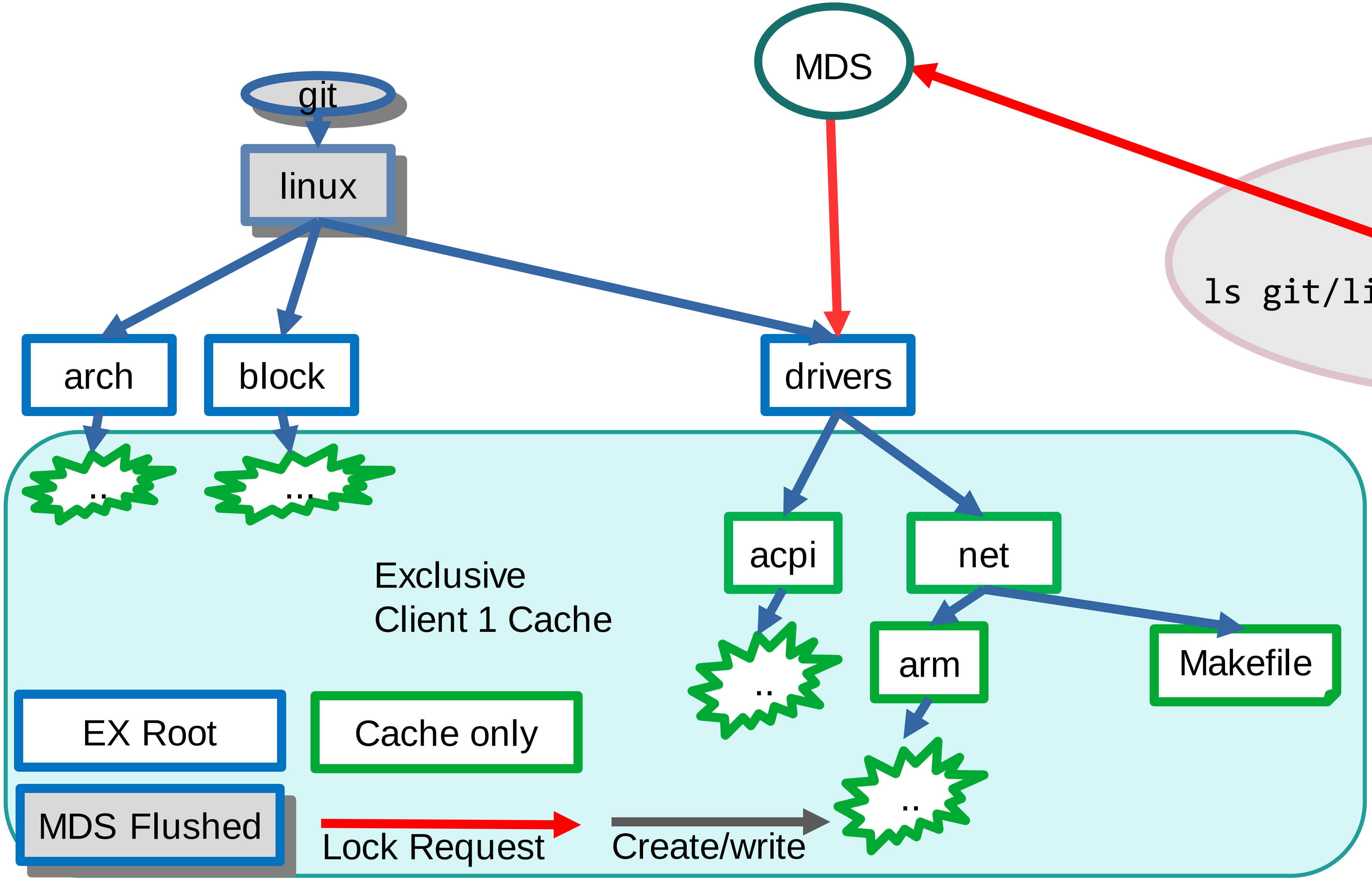
Client 2 requests access to pathname in Client 1's cached subdirectory tree starting from topmost uncached entry `'linux'`, MDS passes request on to Client 1.

Contention and Global Visibility



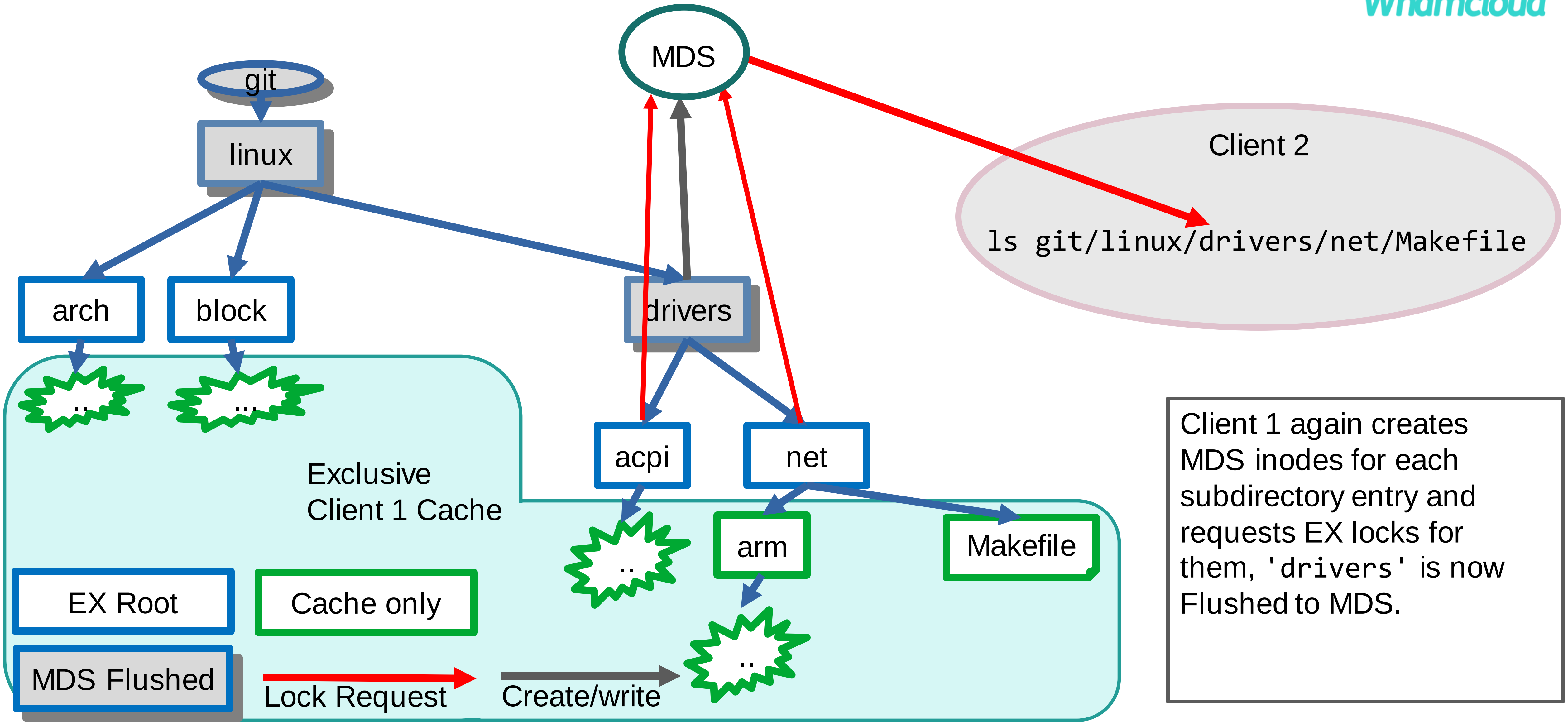
Client 1 creates MDS inodes for each entry in `'linux'` directory and requests EXclusive lock on each one. `'linux'` directory is entirely Flushed to MDS.

Contention and Global Visibility



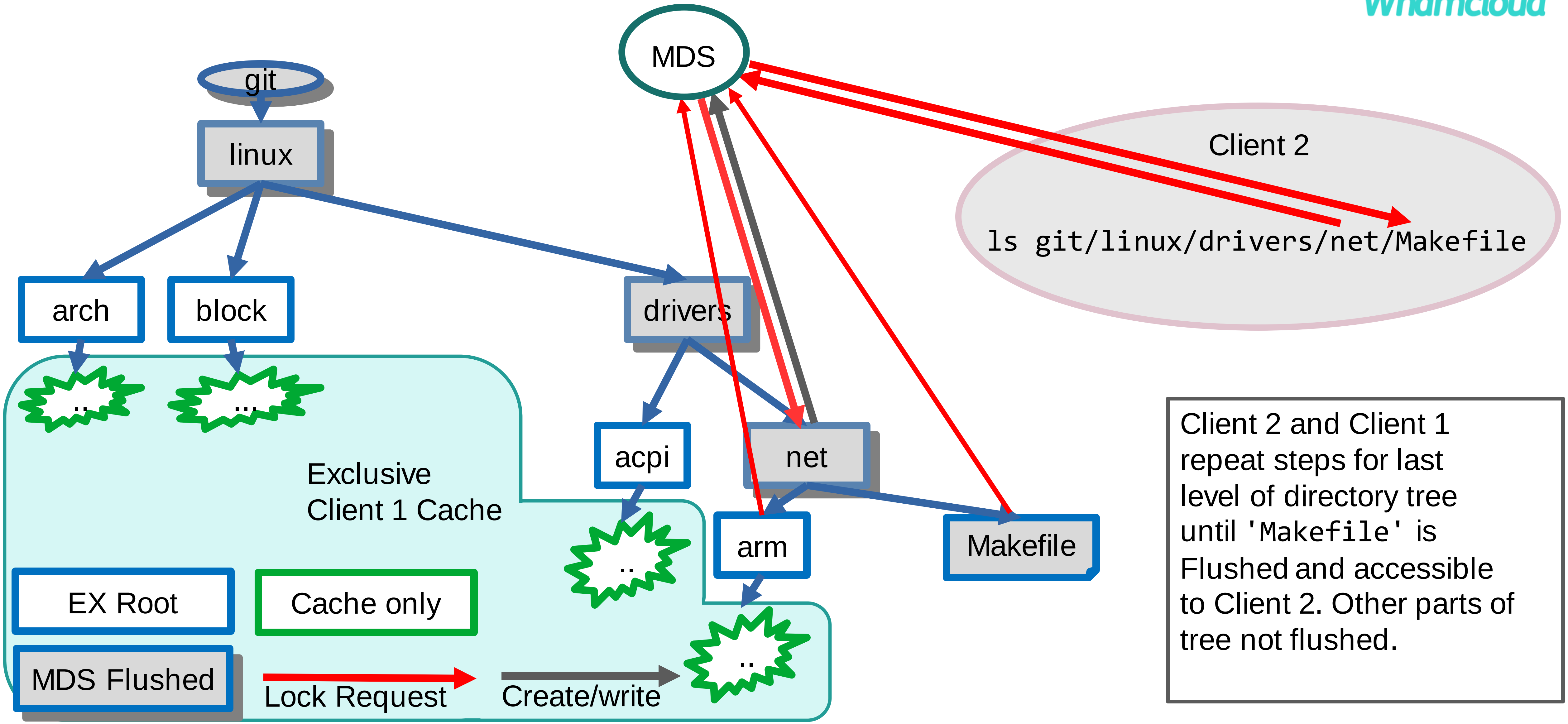
Client 2 next requests 'drivers' entry in subdirectory from MDS, passed on to Client 1. Other entries in 'linux' remain in Client 1 cache undisturbed.

Contention and Global Visibility



Client 1 again creates MDS inodes for each subdirectory entry and requests EX locks for them, `'drivers'` is now Flushed to MDS.

Contention and Global Visibility



Client 2 and Client 1 repeat steps for last level of directory tree until 'Makefile' is Flushed and accessible to Client 2. Other parts of tree not flushed.

Main Usage Targets for Lustre WBC

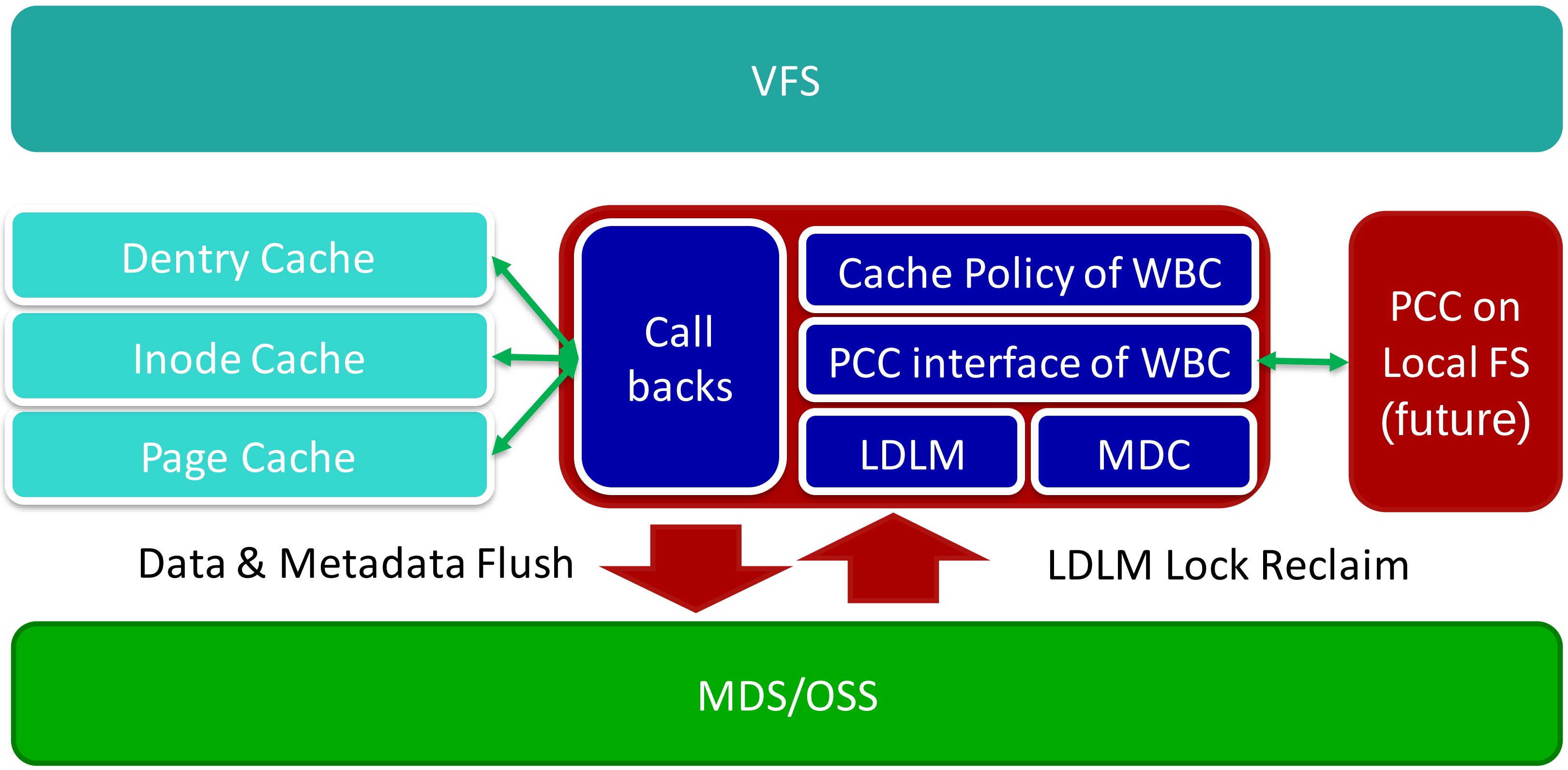
Whamcloud

- Client-side metadata writeback cache instead of server-side
  - Pros: higher acceleration allowed by metadata locality
  - Cons: more complex mechanisms to keep consistency
- Delayed and grouped metadata flush instead of immediate RPC to MDS
  - Pros: many fewer MDS interactions for better performance
  - Cons: mechanism needed for batched flush and space/inode reservation
- Cache in volatile memory instead of persistent storage
  - Pros: quickest storage type
  - Cons: need to flush frequently to reduce risk of data loss
- Keep strong POSIX semantics instead of loosening semantics
  - Pros: transparent and standard behavior for applications on multiple clients
  - Cons: complex LDLM lock protection to maintain consistency

Flushing and Memory Control

- Data and metadata flush happens when:
  - Access of the directory tree from remote clients
  - Memory pressure on local host
  - Periodic aging of cache
- Quick flush from client cache to MDTs
  - Metadata flushing will use bulk RPC for batched flush
  - Only flush or degrade part of the directory tree rather than entire tree

Components in Lustre WBC



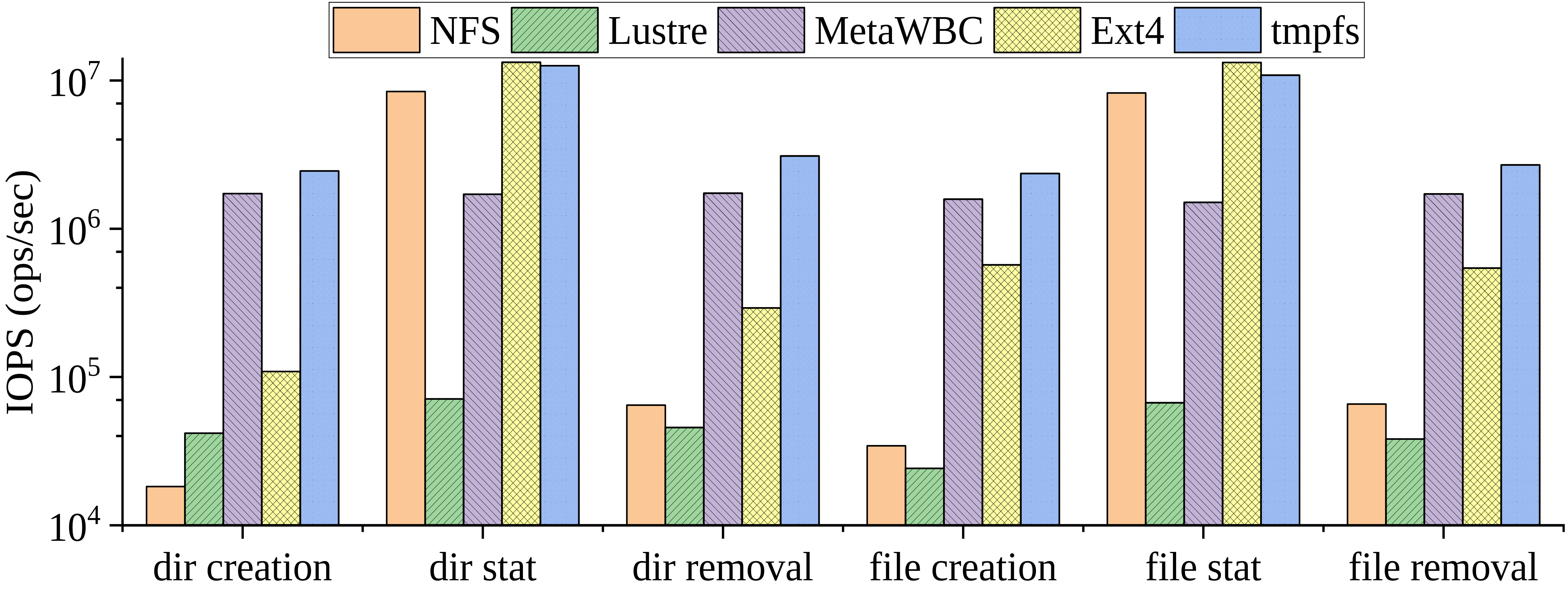
Assimilation of File Data in WBC

- WBC manages both cached metadata and data
- What is WBC-Assimilation of data?
  - Move page cache from being managed by WBC to being managed by Lustre client
  - Data is still in page cache of Lustre client, not flushed to OSS yet
- When to WBC-Assimilate data?
  - Before flushing data to OSS, a WBC-cached file need to be WBC-Assimilated
- How to WBC-Assimilate data?
  - Metadata of the file and its ancestors need to be flushed first
  - File layout created on MDT
  - Put all page cache of the file under the management of main Lustre
  - Now file data could be flushed to storage servers too

Features and Advantages of WBC

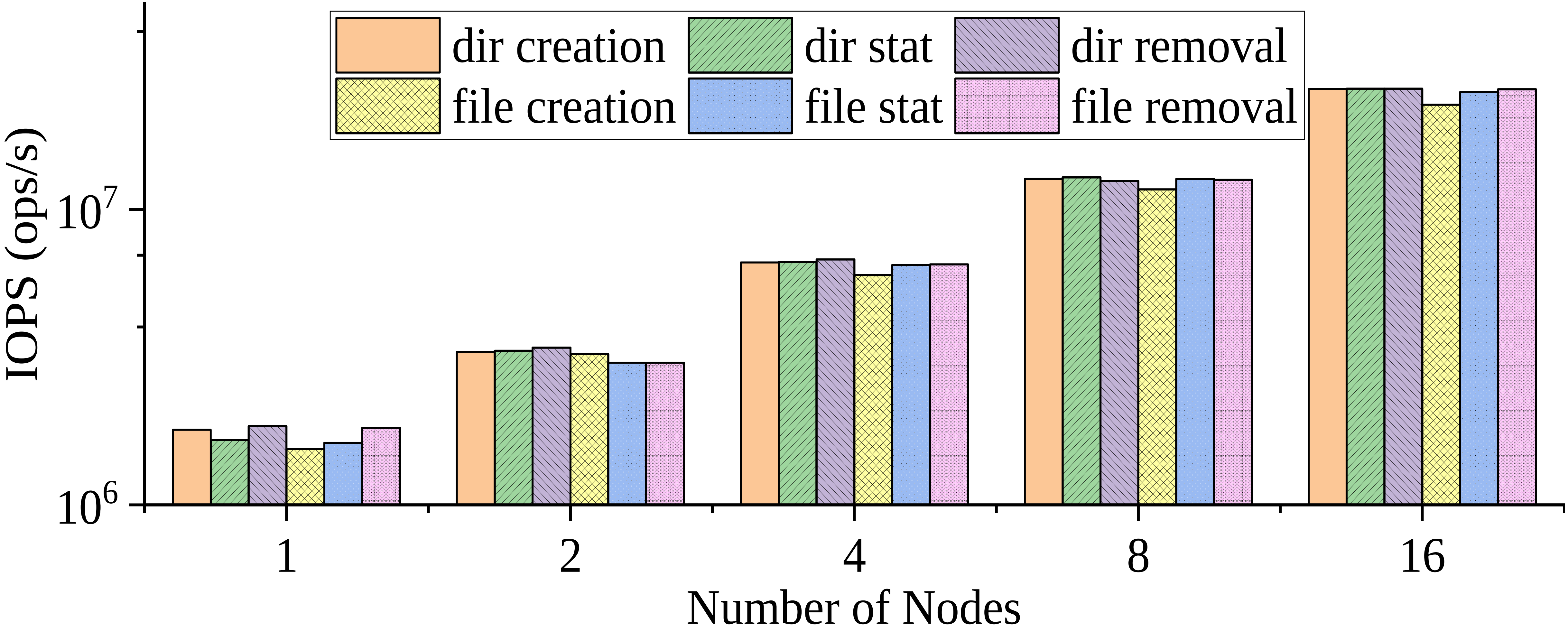
- WBC flushes metadata of file in batch
  - > 1000 file updates in a single bulk RPC
- Batch operations of metadata can be used to delete a whole directory
  - Accelerates "`rm -rf`" a lot
- WBC aggregates metadata updates within a short interval
  - Only the final state of metadata will be flushed to MDS
  - Multiple operations aggregated into a single RPC
- WBC can be integrated with Persistent Client Cache (PCC)
  - Data will still be cached in PCC after WBC-Assimilation
  - Keep more data local to client, more RAM for metadata caching
- Possible offline/disconnected operations on Lustre client in the future

Evaluation: Single Node Metadata Performance



Single node `mdtest` metadata performance (16 processes @ 100K files and directories)

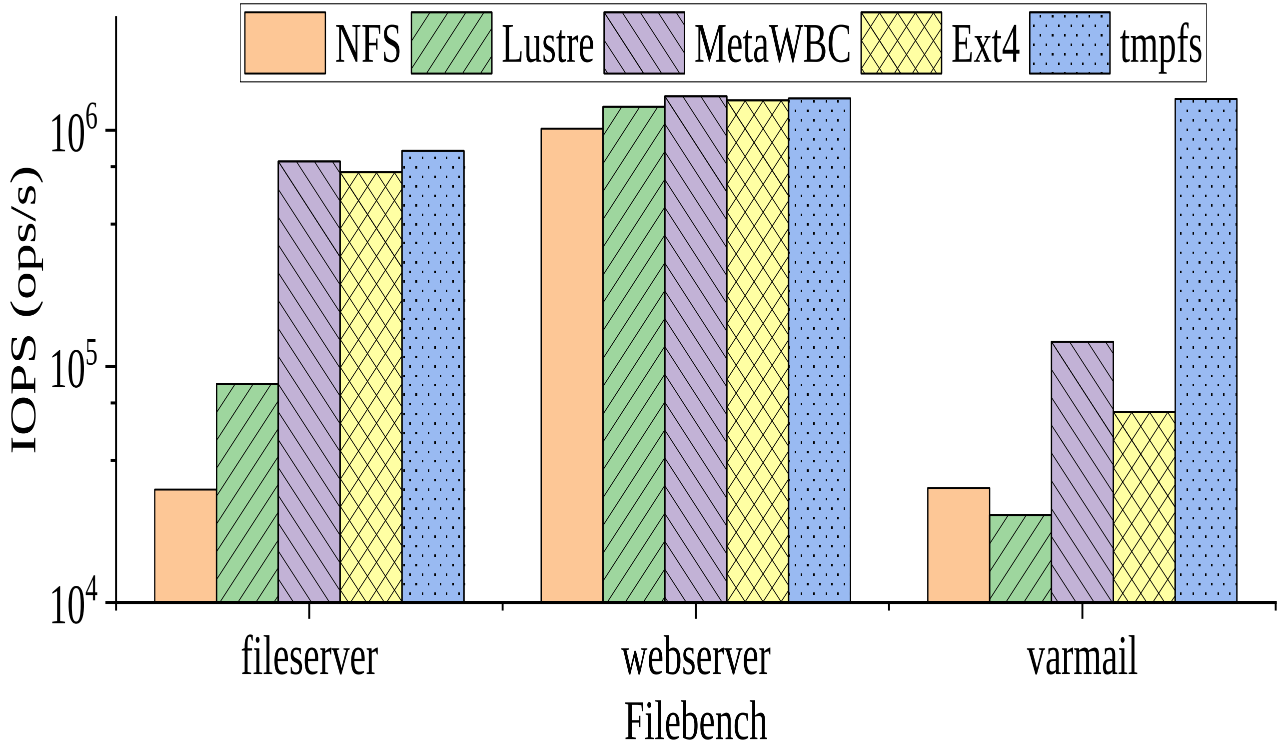
Evaluation: Multiple Node Metadata Performance



MetaWBC `mdtest` metadata performance scaling

Evaluation: Real-world Workloads

- Compare various workloads to other file systems on a single node
  - `filebench` default workloads (1 minute runtime)
  - Common command line applications operating on the Linux kernel source code



Filebench workloads' throughput

App.	NFS	Lustre	MetaWBC	Ext4	tmpfs
cp -rf	64.00	48.00	5.10	1.24	0.75
find -uid	2.89	2.48	0.10	0.71	0.09
du -s	2.84	2.34	0.10	0.70	0.08
ls -lRU	3.40	1.57	0.28	0.93	0.29
grep -r	15.59	16.58	1.08	15.82	0.46

Command line applications' runtime (in seconds)

Evaluation: Pathological Workload

- Investigate pathological workload for write-back caching
  - `mpiFileUtils dtar`: Parallel extraction of eight Linux Kernel source code trees
  - EX locks are granted when creating a directory or during de-rooting
  - EX locks immediately revoked due to conflicting access from remote clients
  - Constant flush-back of cached files and transition to write-through mode

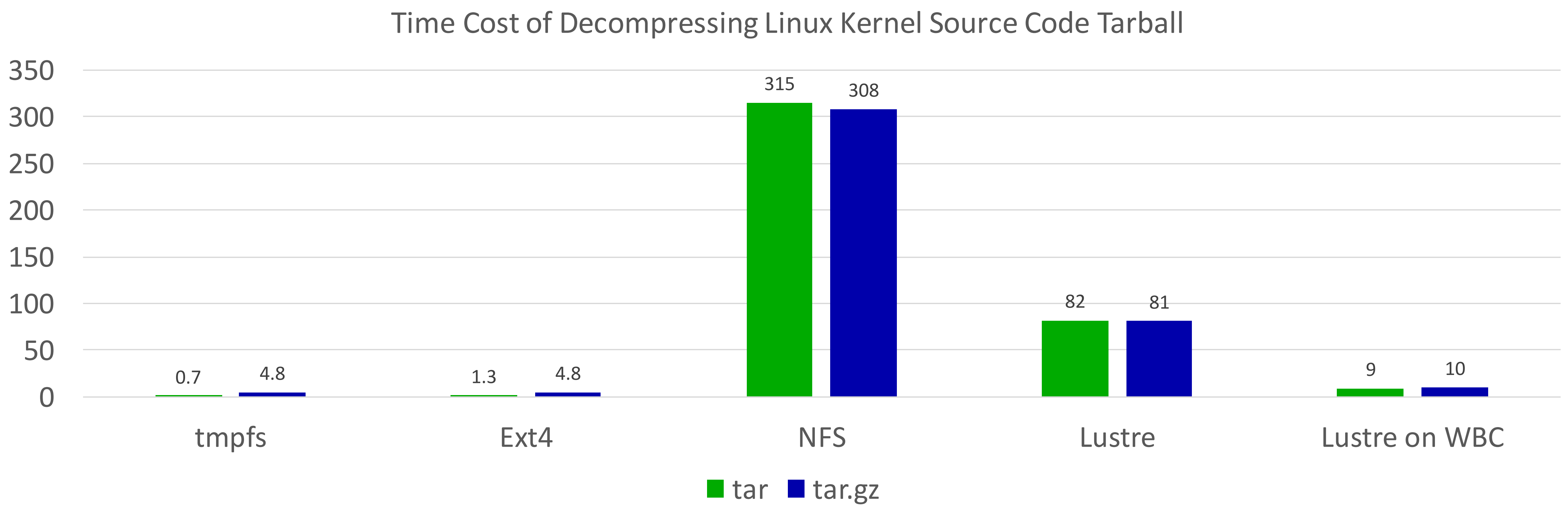
Time phase	Create tree	Extract data	Update attr	Total
CephFS	87	180	59	326
Ceph_async	89	170	62	321
Lustre	13	76	28	197
MetaWBC	1	56	1	136

Time (in seconds) for mpiFileUtils dtar phases

Even under worst-case workloads for writeback caching, WBC improves Lustre metadata performance

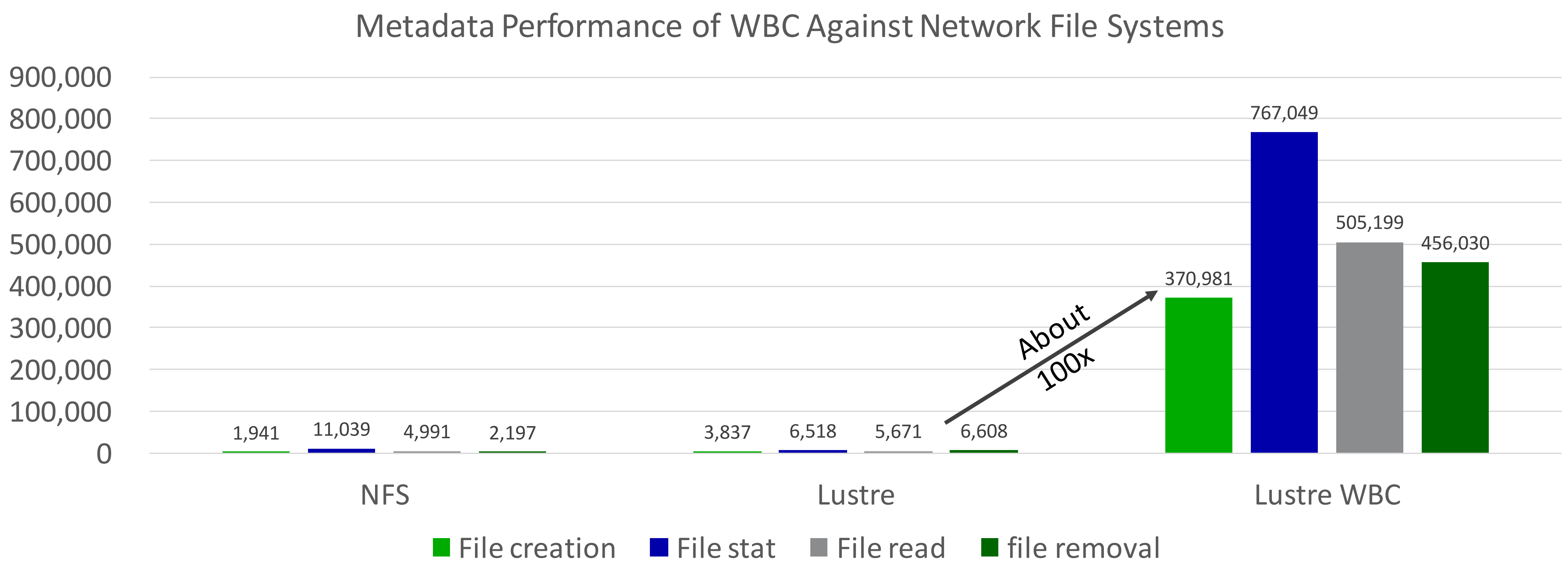
Evaluation: Untar of WBC Against Other File Systems

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



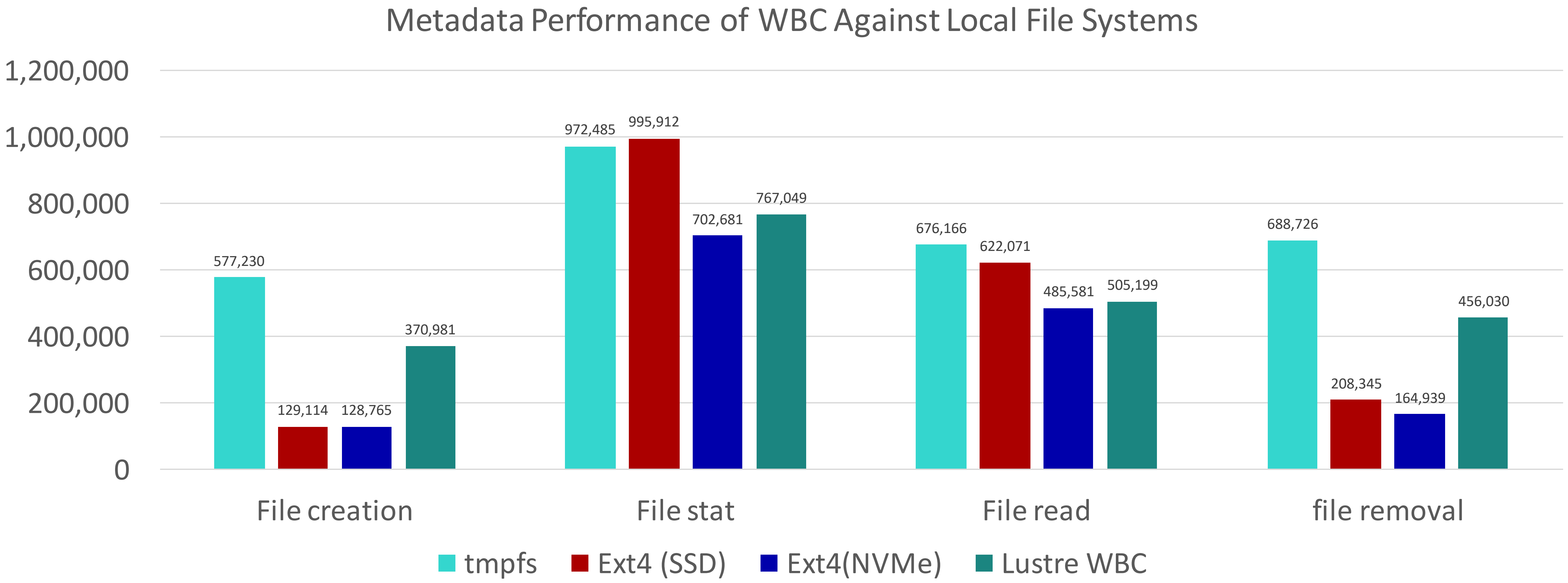
Metadata Performance of WBC Against Network FS

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



Metadata Performance of WBC Against Local FS

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



Summary

- Metadata Writeback Cache will accelerate metadata intensive workloads
- Batched RPC support and improved statahead coming in Lustre 2.16
- Complete WBC feature targeted for Lustre 2.17

Whamcloud

Thank you!

ddn