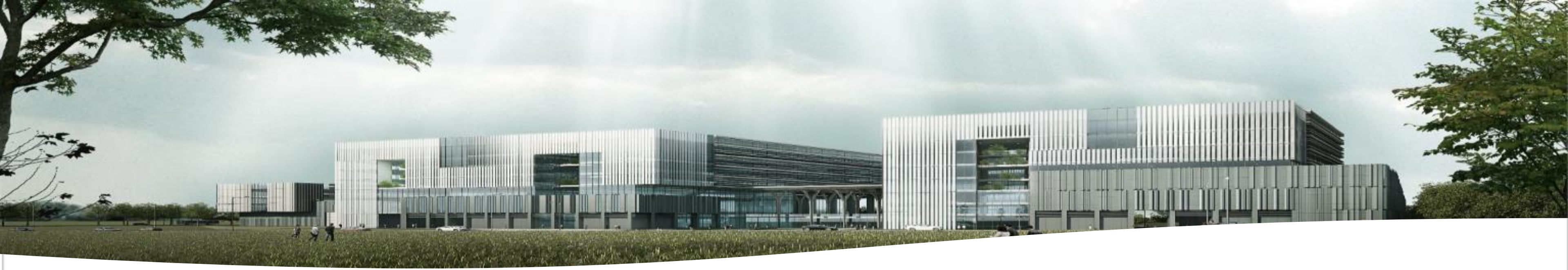


China Lustre User Group 2018 (China LUG), Beijing, October 23, 2018

Providing Persistent Client Caching Services with a Lustre Global Namespace

Lingfang Zeng (曾令仿)

Wuhan National Laboratory for Optoelectronics (WNLO)

Huazhong University of Science and Technology (HUST)

With Contributions from

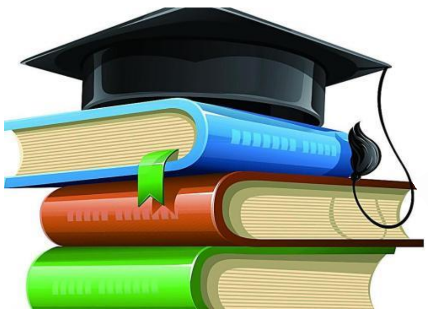
- Yingjin Qian, Xi Li, Shuichi Ihara, Carlos Aoki Thomaz, and Shilong Wang @ **DDN**
- Andreas Dilger @ **DDN Whamcloud**
- Tim Süß, and André Brinkmann @ **JGU**
- Wen Cheng, Chunyan Li, Fang Wang, and Dan Feng @ **HUST**

ACKNOWLEDGMENTS

DDN® STORAGE

JG|U
JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Outline



BACKGROUND

PROBLEM & TERMINOLOGY & OBJECTIVES

METHODS

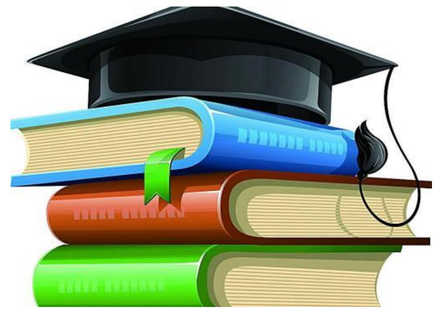
HIERARCHICAL PERSISTENT CLIENT CACHING

IMPLEMENTATION

RW-PCC & RO-PCC & RULE-BASED TRIGGERING & POLICY ENGINE

EVALUATIONS

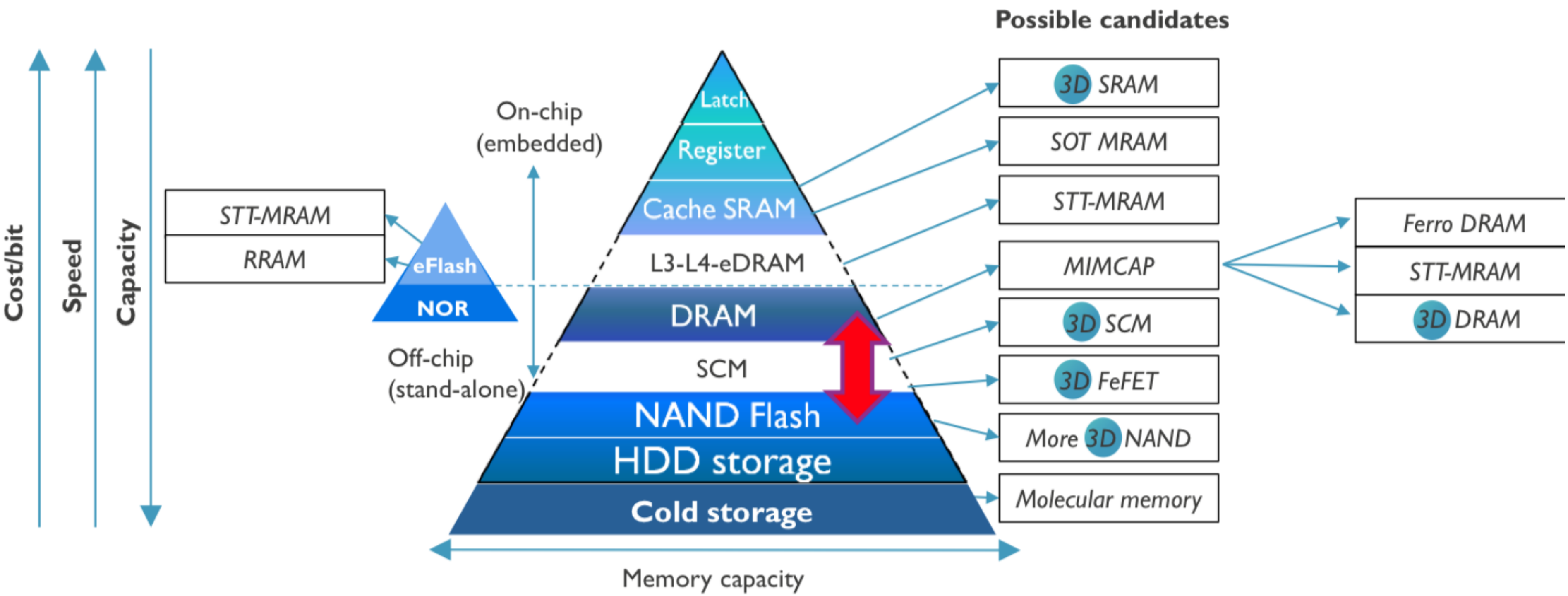
EXPERIMENT & RESULTS



01 BACKGROUND

PROBLEM & TERMINOLOGY & OBJECTIVES

Hierarchical Storage Management (HSM)

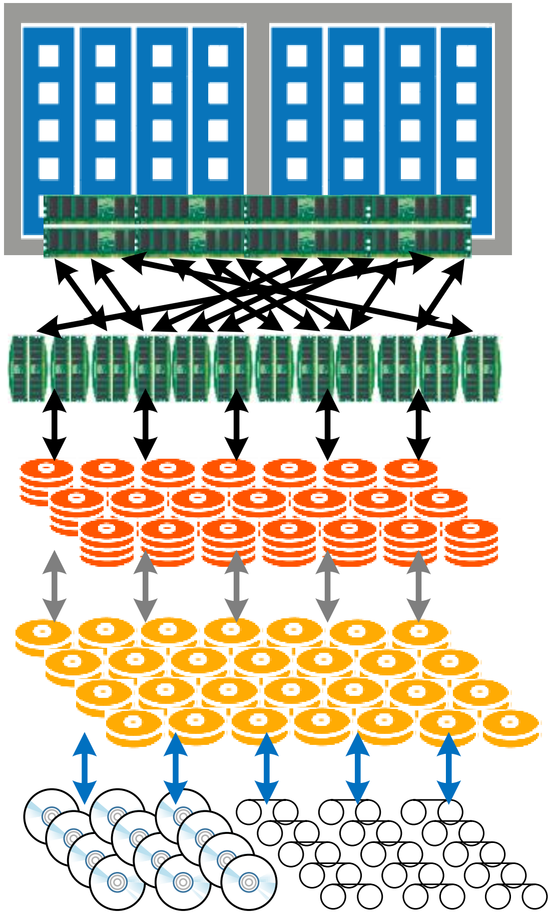


HPC workloads were too big to be stored only on flash

https://semiengineering.com/a-new-memory-contender/
https://storageswiss.com/2018/01/10/enterprise-needs-to-learn-from-hpc-environments/

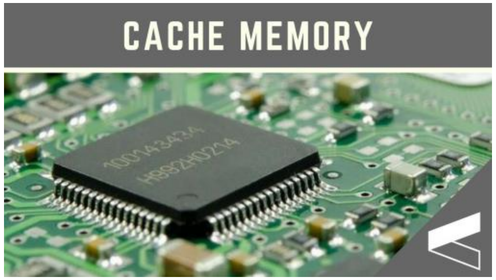
HSM Tier

- Compute servers
  - HBM
  - NVRAM/SCM
- Performance storage
  - DRAM
  - SSD
  - (performance HDD)
- Capacity storage
  - DRAM
  - Capacity HDD



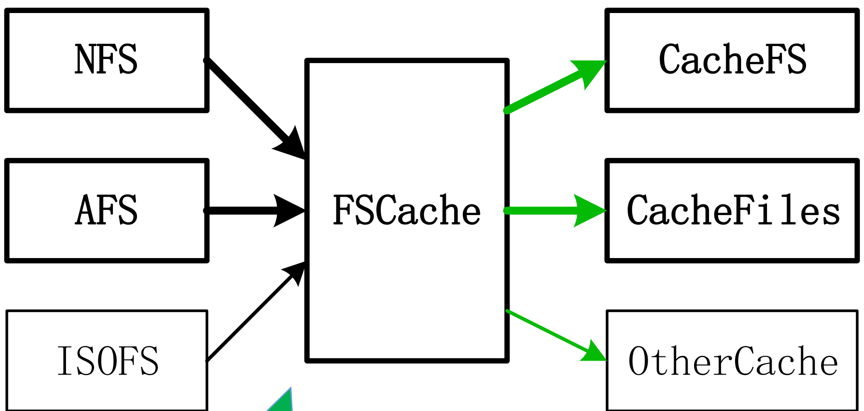
Lang's Law: the more tiers, the more tears

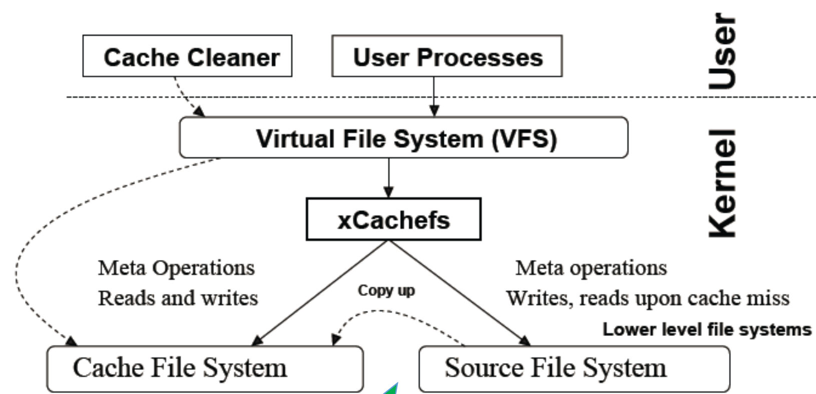
Industry and Academic Solutions



- Andrew File System [TOCS’88, CMU]
- Coda File System [TOCS’88, CMU]
- FS-Cache [Linux Symposium’06, Red Hat]
- BWCC [CLUSTER’12, CAS]
- Nache [FAST’07, RU & IBM]
- Panache [FAST’10, IBM]
- Mercury [MSST’12, NetApp]
- GPFS’ LROC [IBM]
- TRIO [CLUSTER’15, FSU & ORNL & AU]
- BurstFS [SC’16, FSU & LLNL]
- MetaKV [IPDPS’17, FSU & LLNL]
- Dmcache [TOCS’88, CMU]
- Xcachefs [SBU, 2005]
- FlashCache [CASES’06, UM]
- Bcache [LWN, 2010]

Related Work



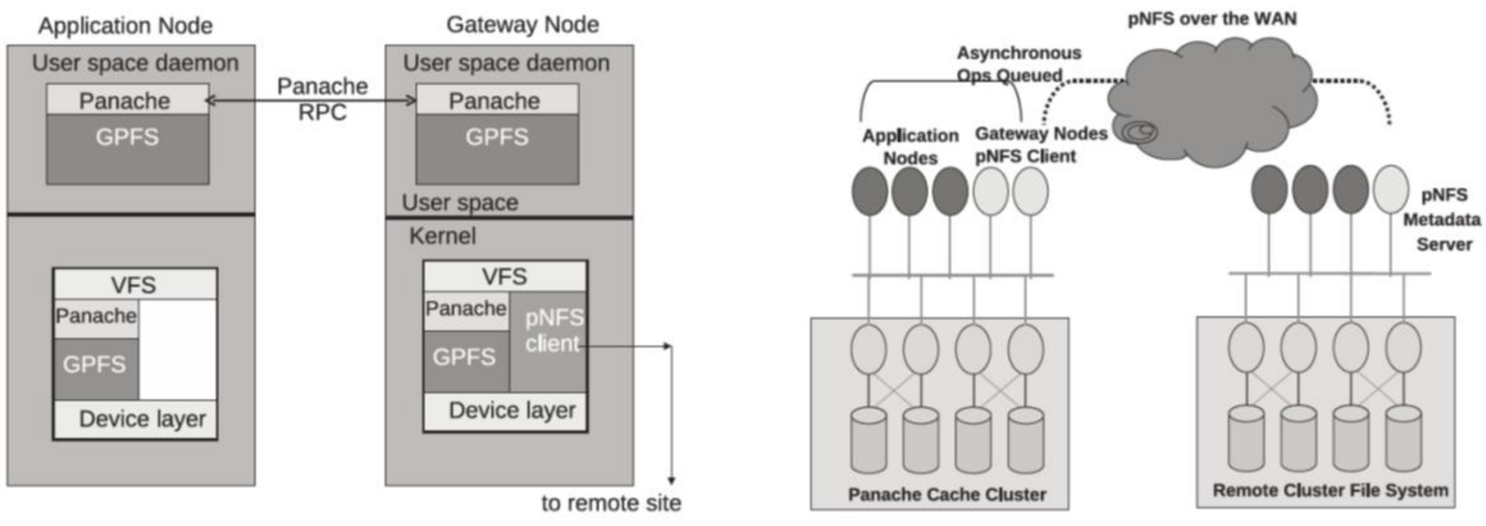


- Read-only cache
- Tolerate I/O failures in cache

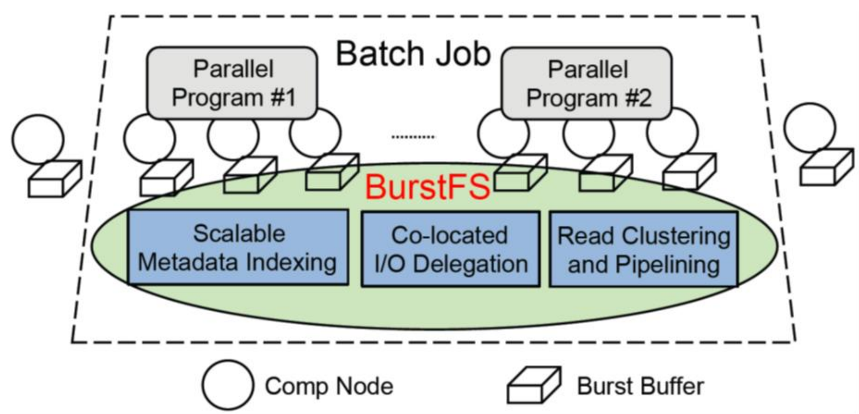
- File system meta-operations (both cache and source)

Reference: Howells, FS-Cache: A Network Filesystem Caching Facility, Red Hat UK Ltd.
Sivathanu+, A Versatile Persistent Caching Framework for File Systems, Stony Brook University, Technical Report FSL-05-05.

Related Work



(a) Node Block Diagram (b) Cache Cluster Architecture



Reference: Eshel+, Panache: A parallel file system cache for global file access, FAST'10
Wang+,An ephemeral burst-buffer file system for scientific applications, SC'16

Lustre File System

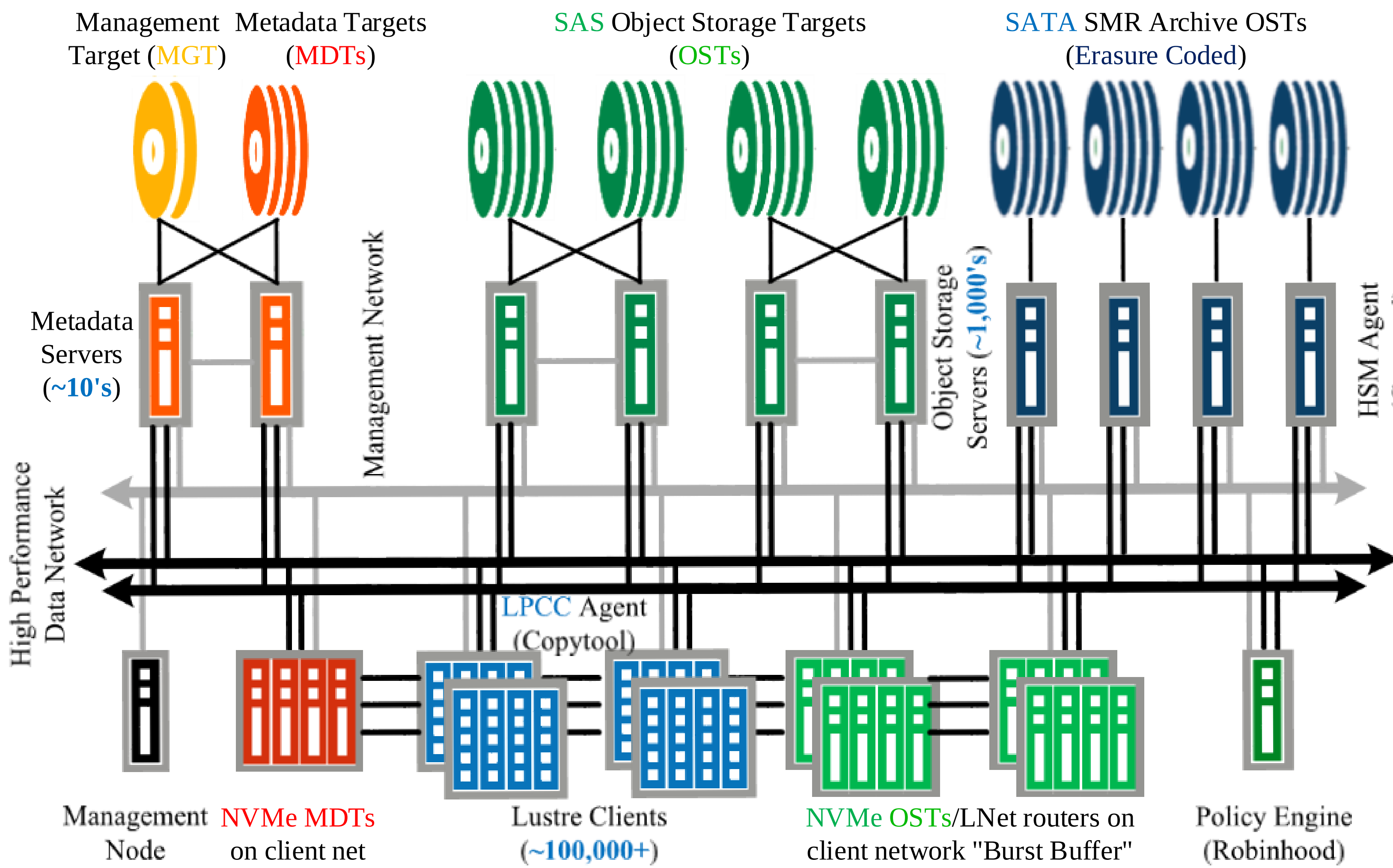
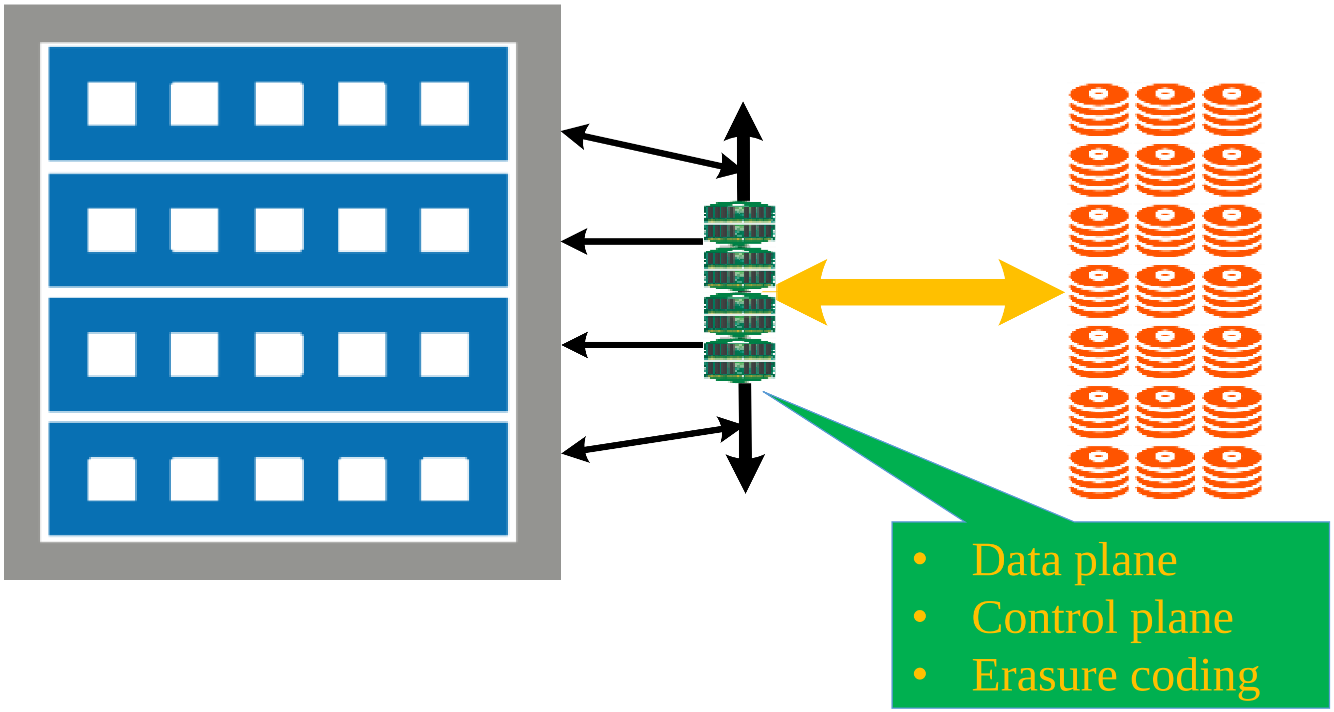


Figure based on Andreas Dilger's Lustre User Group 2018 presentation: Lustre 2.12 and beyond (see http://opensfs.org/lug-2018-agenda/)

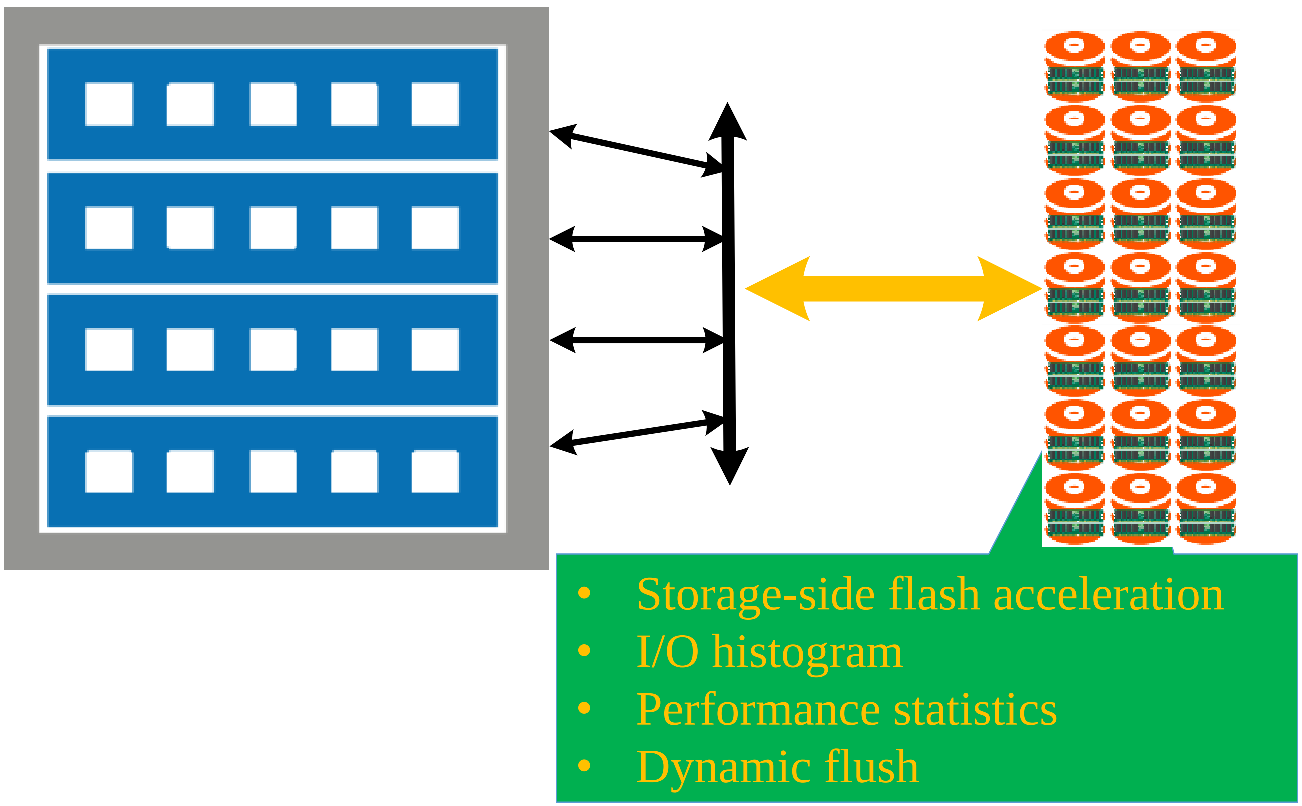
HSM Tier

- Shared
  - DDN IME @ ICHEC
  - Cray Trinity @ LANL



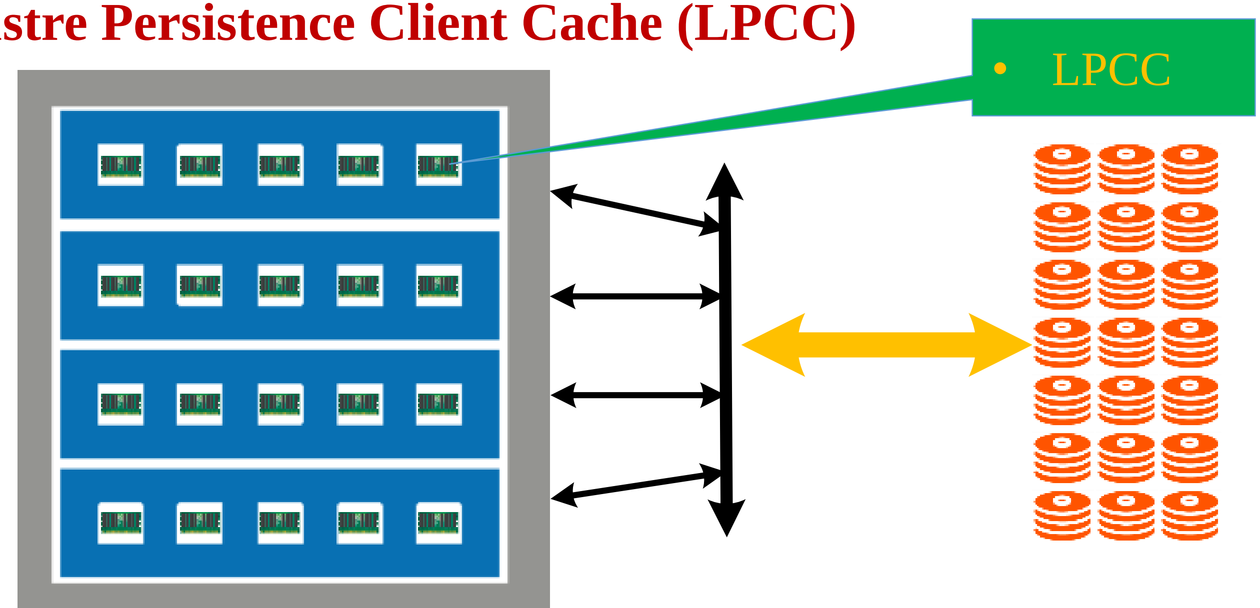
HSM Tier

- Shared
  - DDN IME @ ICHEC
  - Cray Trinity @ LANL
- Server-side
  - Seagate Nytro NXD @ Sanger



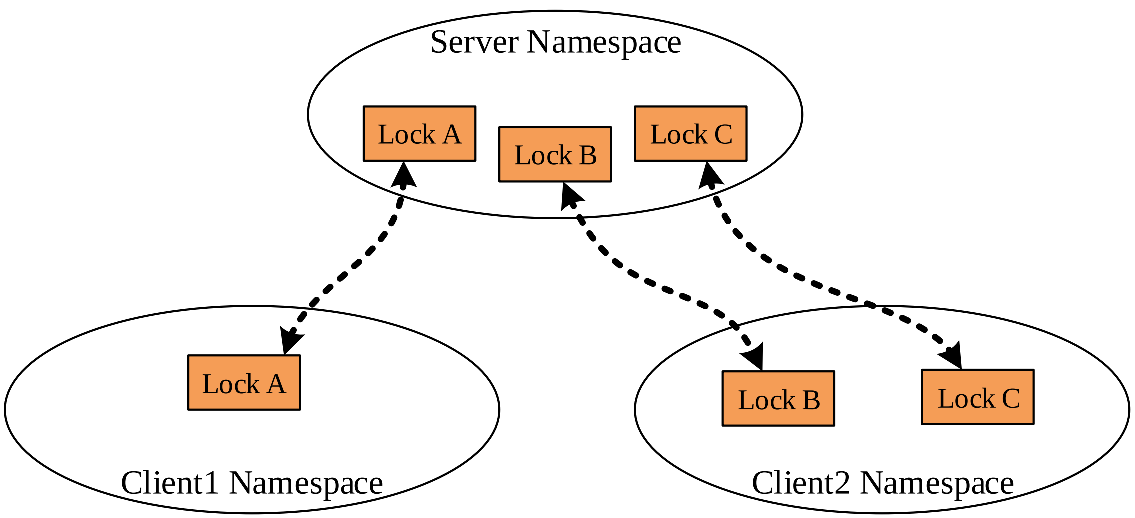
HSM Tier

- Shared
  - DDN IME @ ICHEC
  - Cray Trinity @ LANL
- Server-side
  - Seagate Nytro NXD @ Sanger
- Client-side
  - Intel/Cray Aurora (A21) @ Argonne National Laboratory?
  - **Lustre Persistence Client Cache (LPCC)**



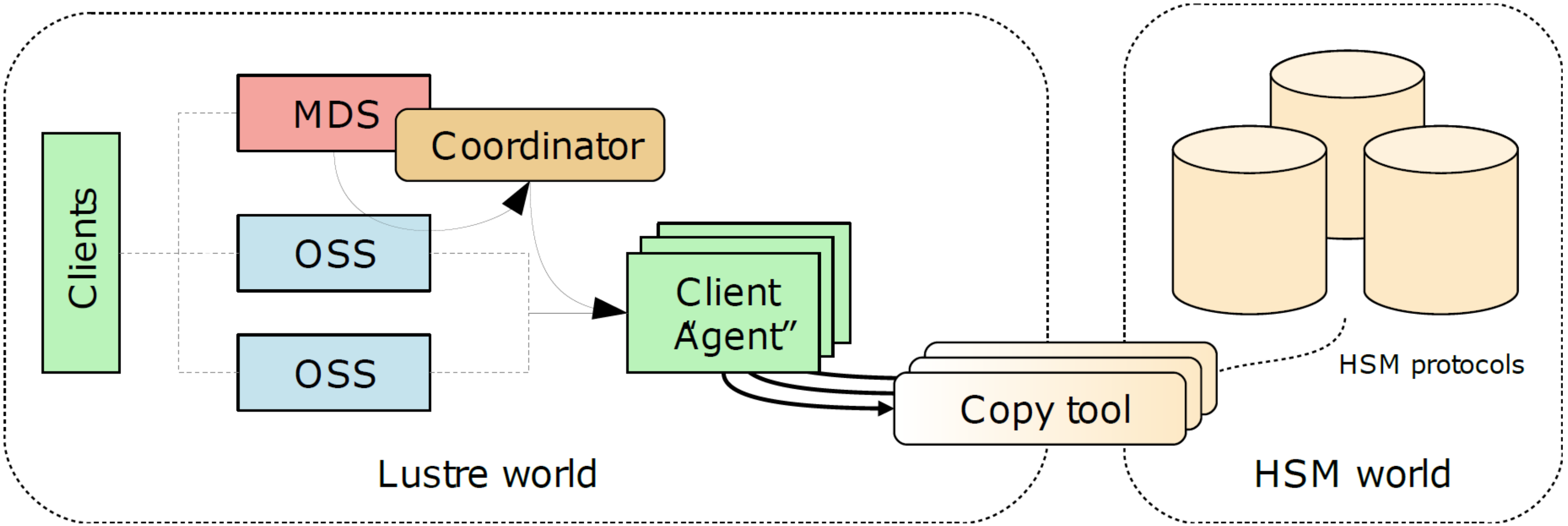
Lustre's DLM and Layout Lock

- Distributed lock manager (DLM)
  - Data and metadata consistency
  - A separate namespace
- Excusive mode (EX) lock
- Concurrent read mode (CR) lock
- *L.Gen* field



Lustre HSM

- Agents – Lustre file system clients running Copytool
- Coordinator – Act as an interface between the policy engine, the metadata server(MDS) and the Copytool



Refererence: Lustre manual. http://doc.lustre.org/lustre_manual.pdf

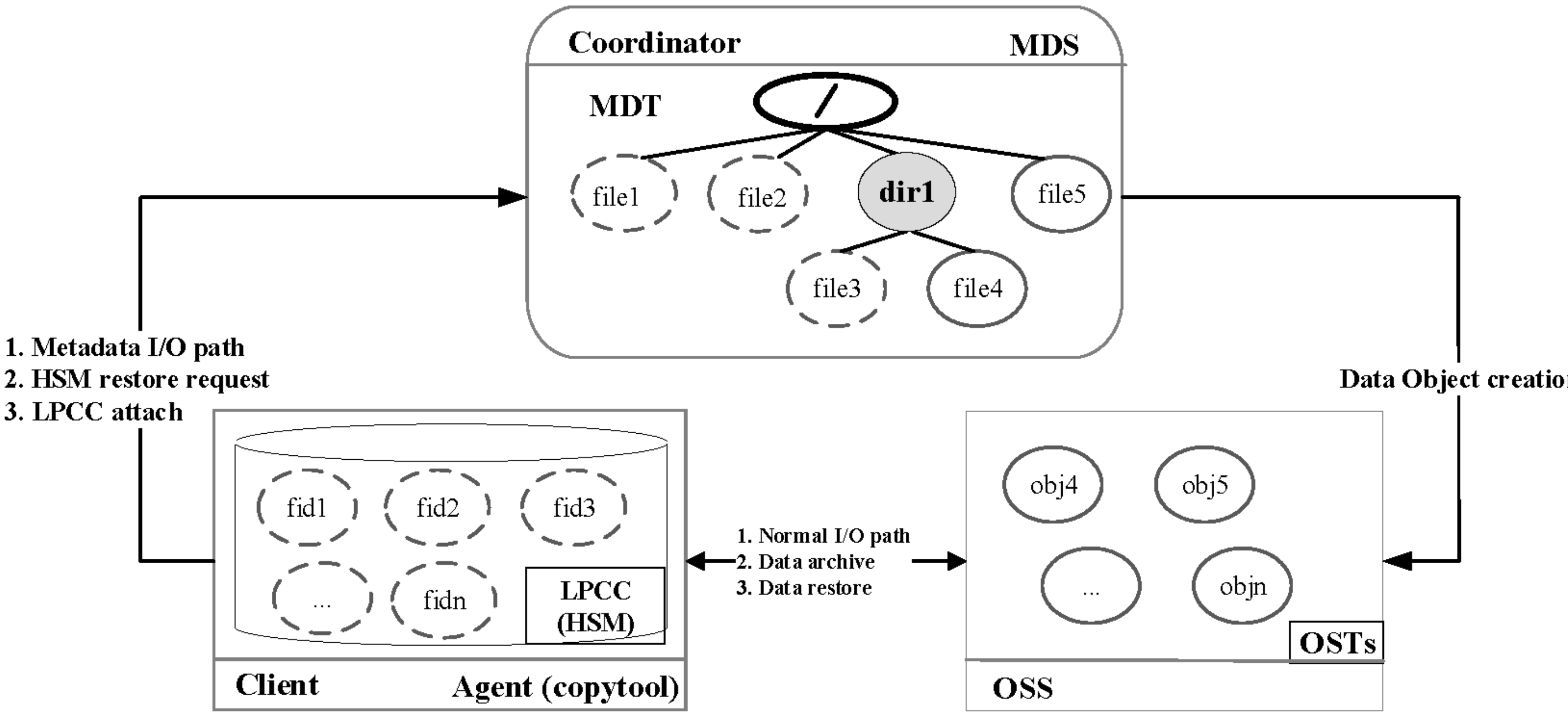
Key Idea

- Logical two-tier (with physical multitier)
  - Simple and efficient architecture (memory vs. disk)
- A global namespace
  - Space efficient
- Latencies and lock conflicts can be significantly reduced
- Caching reduces the pressure on (OSTs)
  - small or random I/Os can be regularized to big sequential I/Os and temporary files do not even need to be flushed to OSTs.

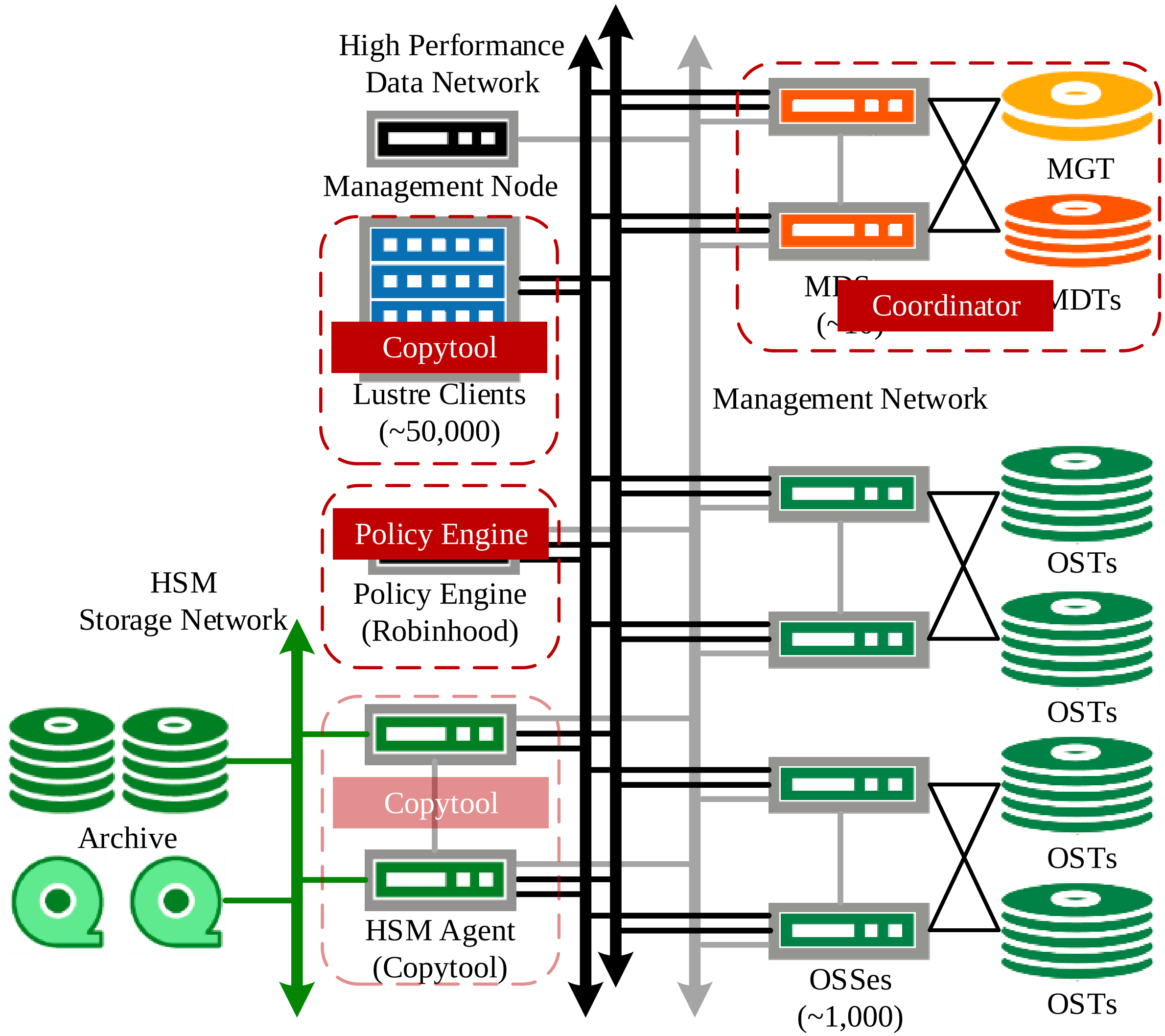
02 METHODS

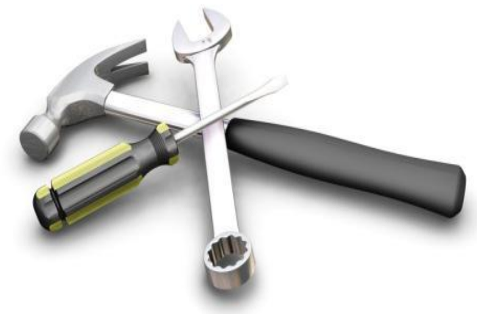
HIERARCHICAL PERSISTENT CLIENT CACHING

Overview of LPCC Architecture



Overview of LPCC Architecture

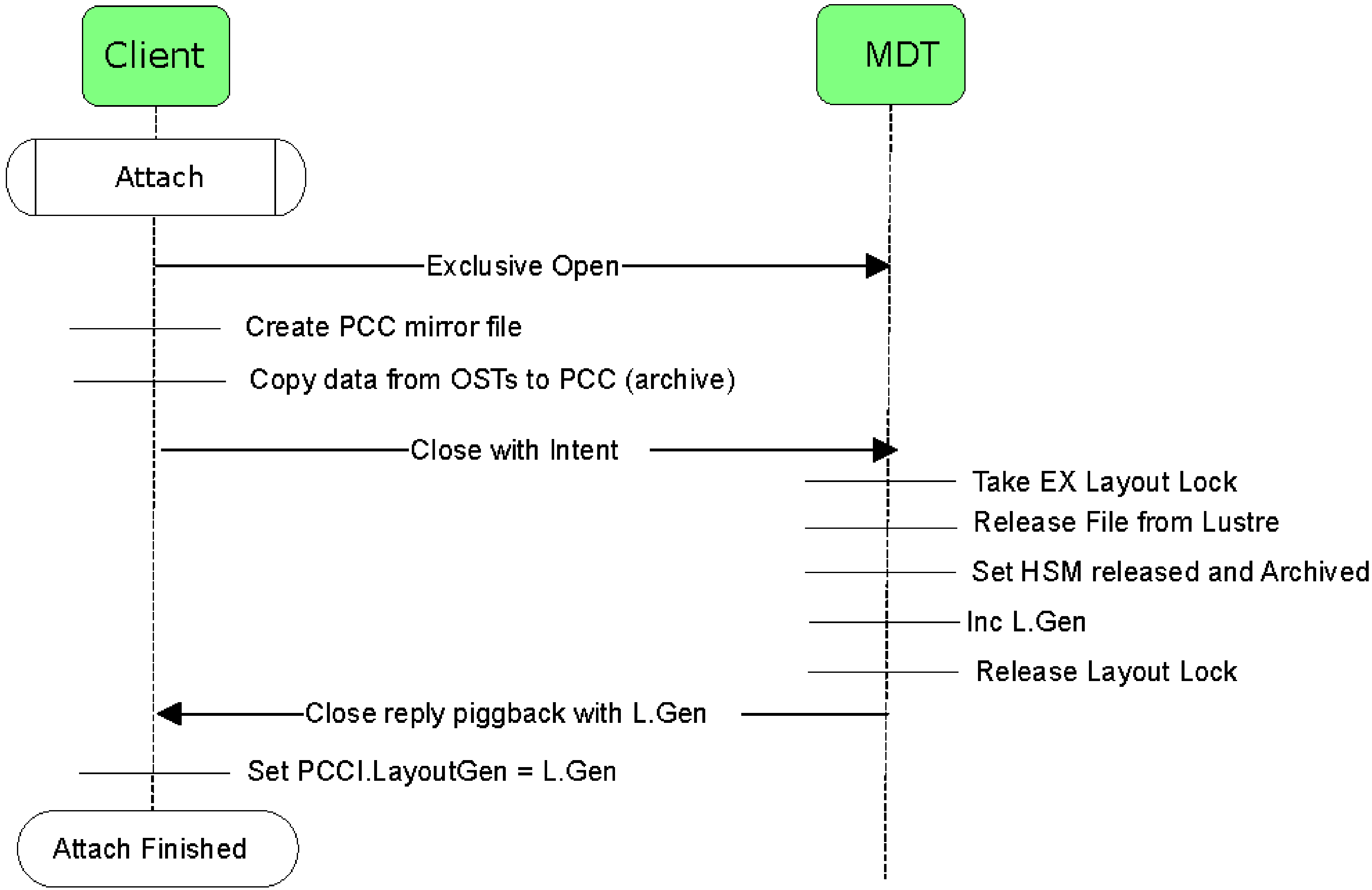




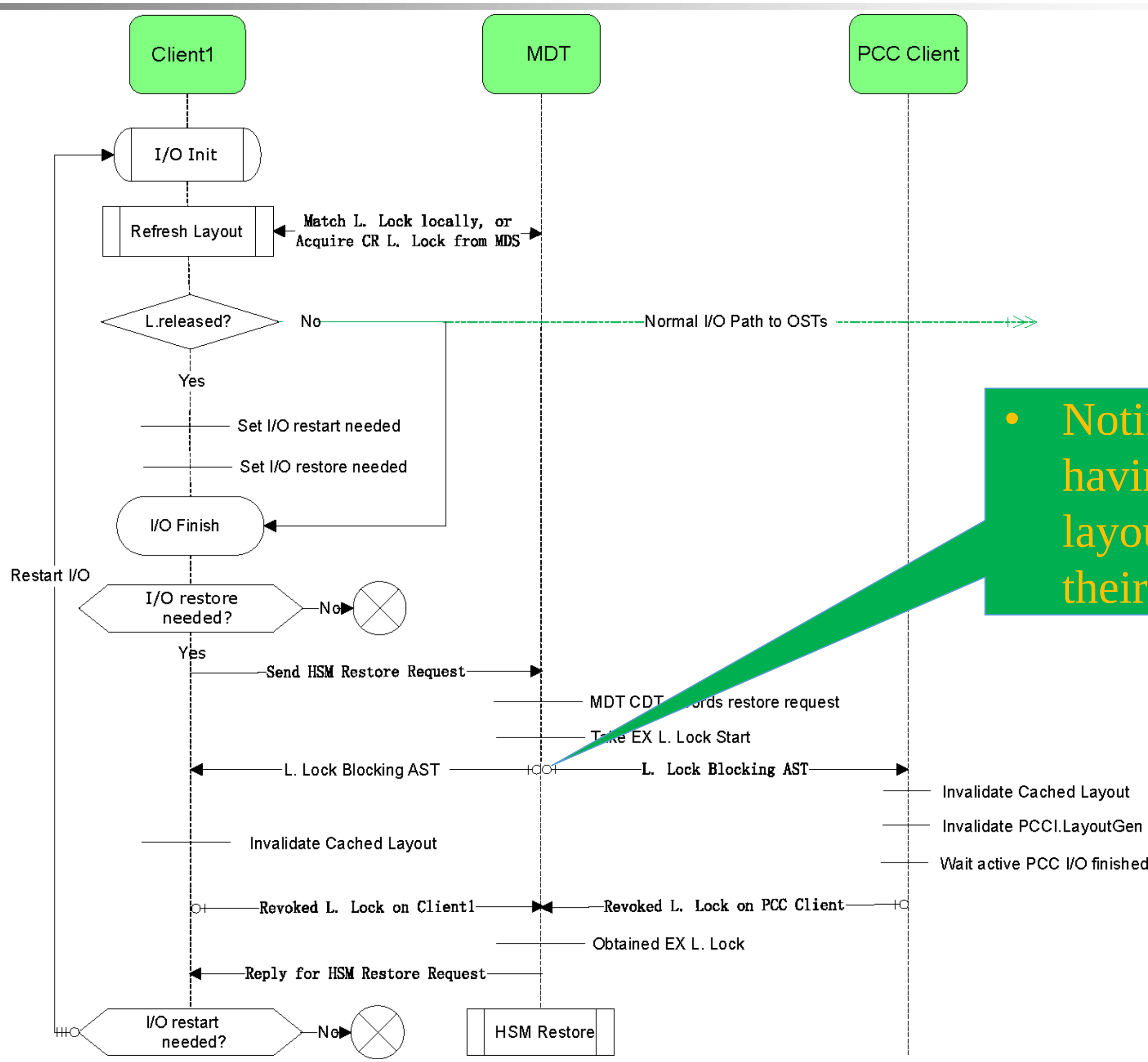
03 IMPLEMENTATION

RW-PCC & RO-PCC & RULE-BASED TRIGGERING & POLICY ENGINE

Lustre Read-Write PCC Caching (attach)

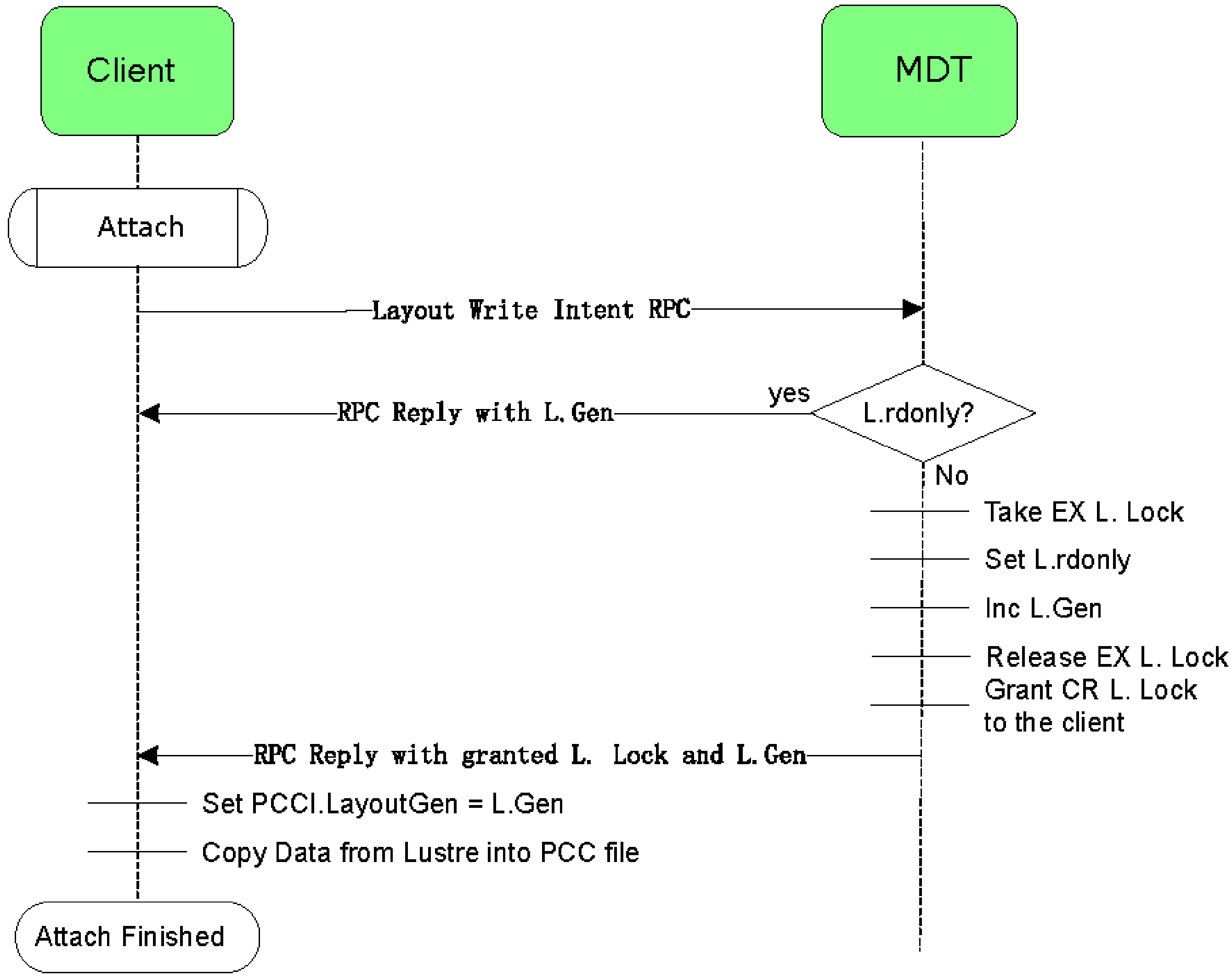


Lustre Read-Write PCC Caching (restore)

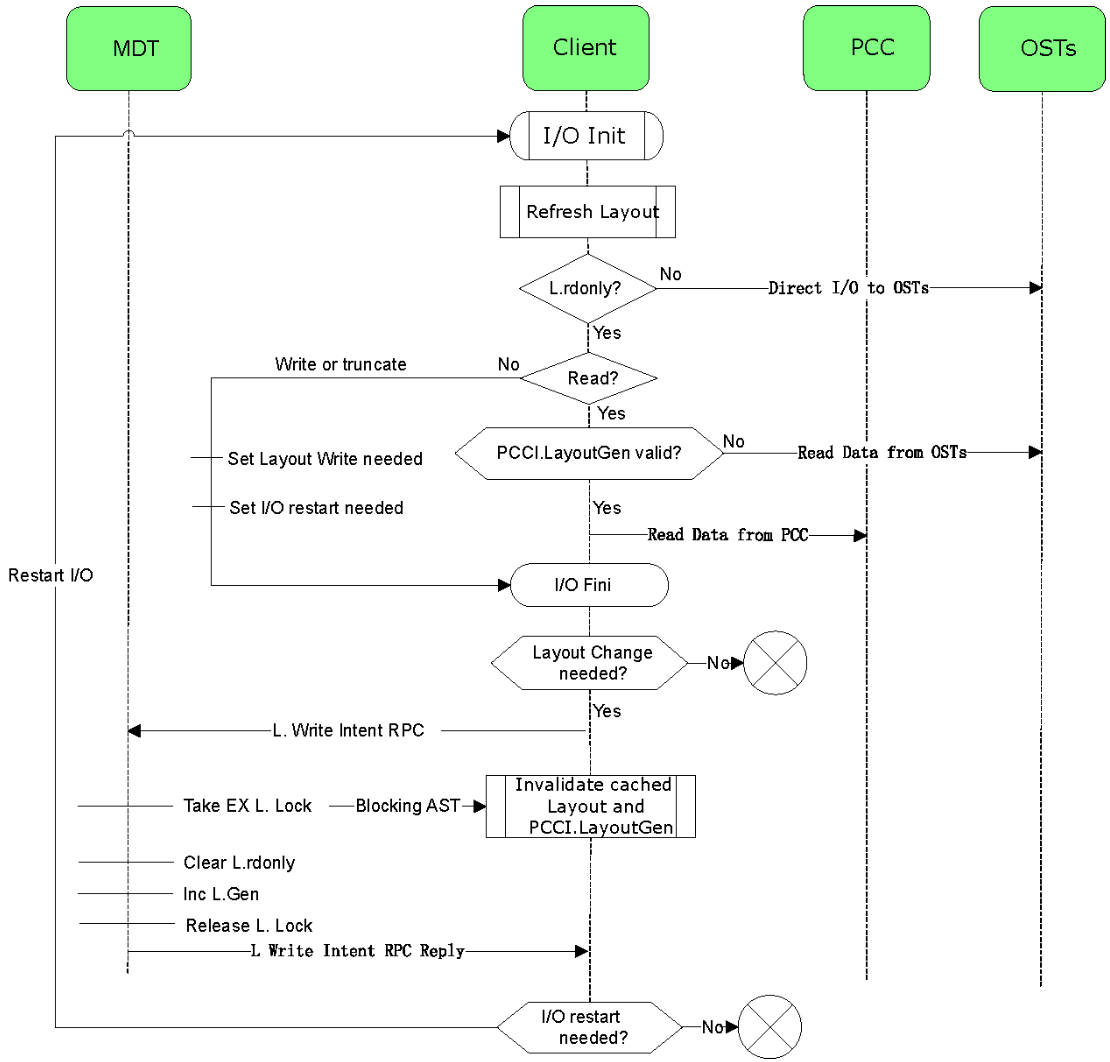


- Notify all clients having cached the layout to invalidate their layouts

Lustre Read-only PCC Caching (attach)



Lustre Read-only PCC Caching (I/O flow)



Rule-based Persistent Client Caching

- Different user, groups, and projects or filenames
  - E.g. (projid={500,1000} & fname=*.h5),(uid=1001)
- Quota limitation
  - Cache isolation
- Auto LPCC caching mechanism

Cache Prefetching and Replacement

- Policy engine
  - Manage data movement
- Lustre changelogs
  - Periodic prefetching decision
- LRU and SIZE

04 EVALUATIONS

EXPERIMENT & RESULTS

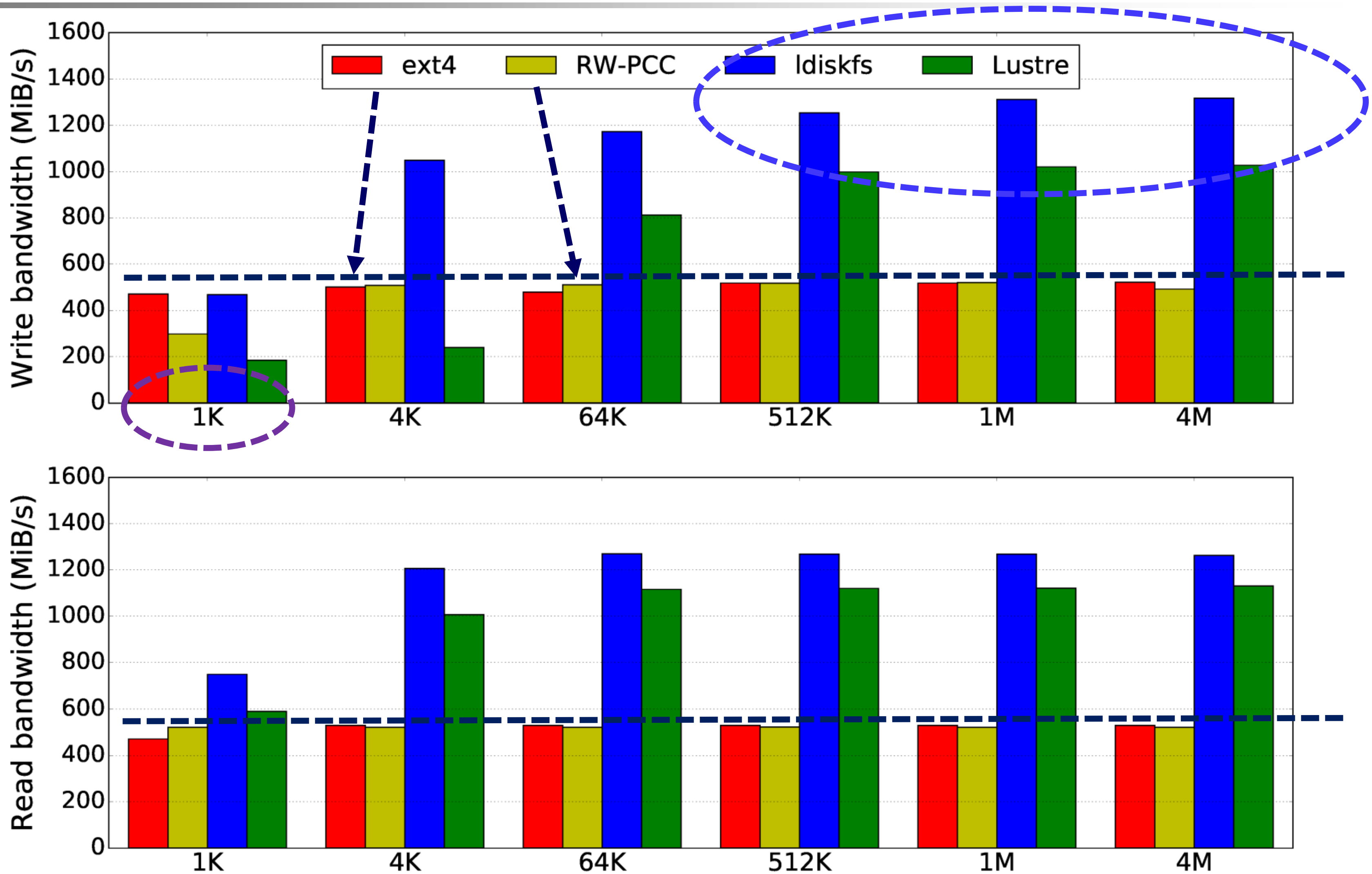
Evaluation Setup

- CentOS 7 Linux (3.10.0) and Lustre (2.11.53)
- All client nodes included
  - An Intel Xeon E5-2650 processors with 128GB of memory
  - 512GB Samsung 840 PRO series SSD as LPCC cache (ext4-based LPCC)
- Lustre OSS DDN SFA14KXE with 10 OSTs (ext4-based ldiskfs)
- MDS Toshiba 200GB SSD (ext4-based ldiskfs)
- “stripe=n” means file data is striped over *n* OSTs
- Lustre Data on MDT (DoM)
  - To improve small file performance by allowing the data on the MDT
- FS-Cache mechanism

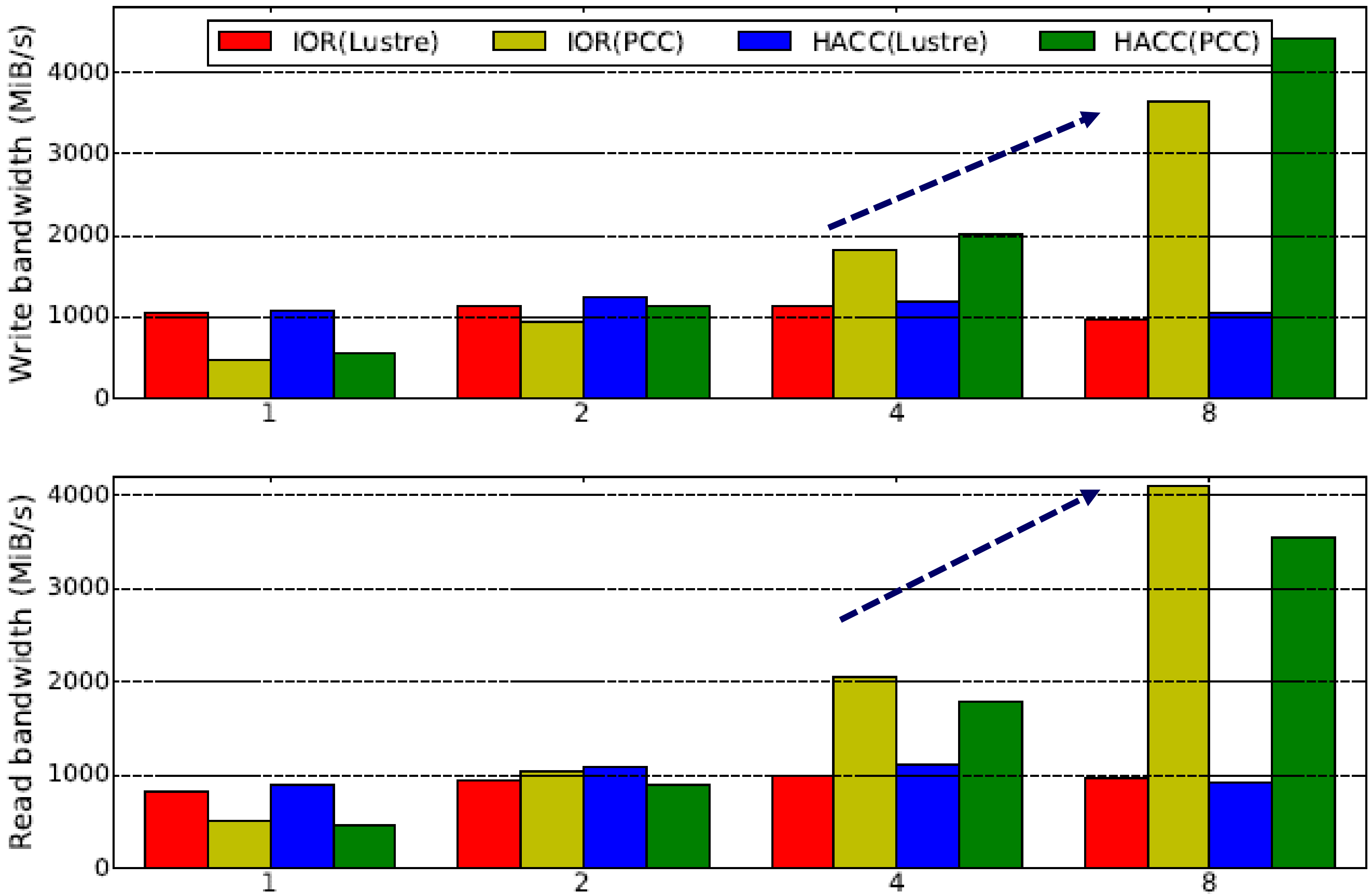
Benchmark Tools

- fio
- IOR (file-per-processor (FPP))
- mdtest
- filebench
- HACC I/O
  - HPC application simulation in FPP mode
- Compliebench
  - Simulate kernel compiles with target to metadata and small file operations

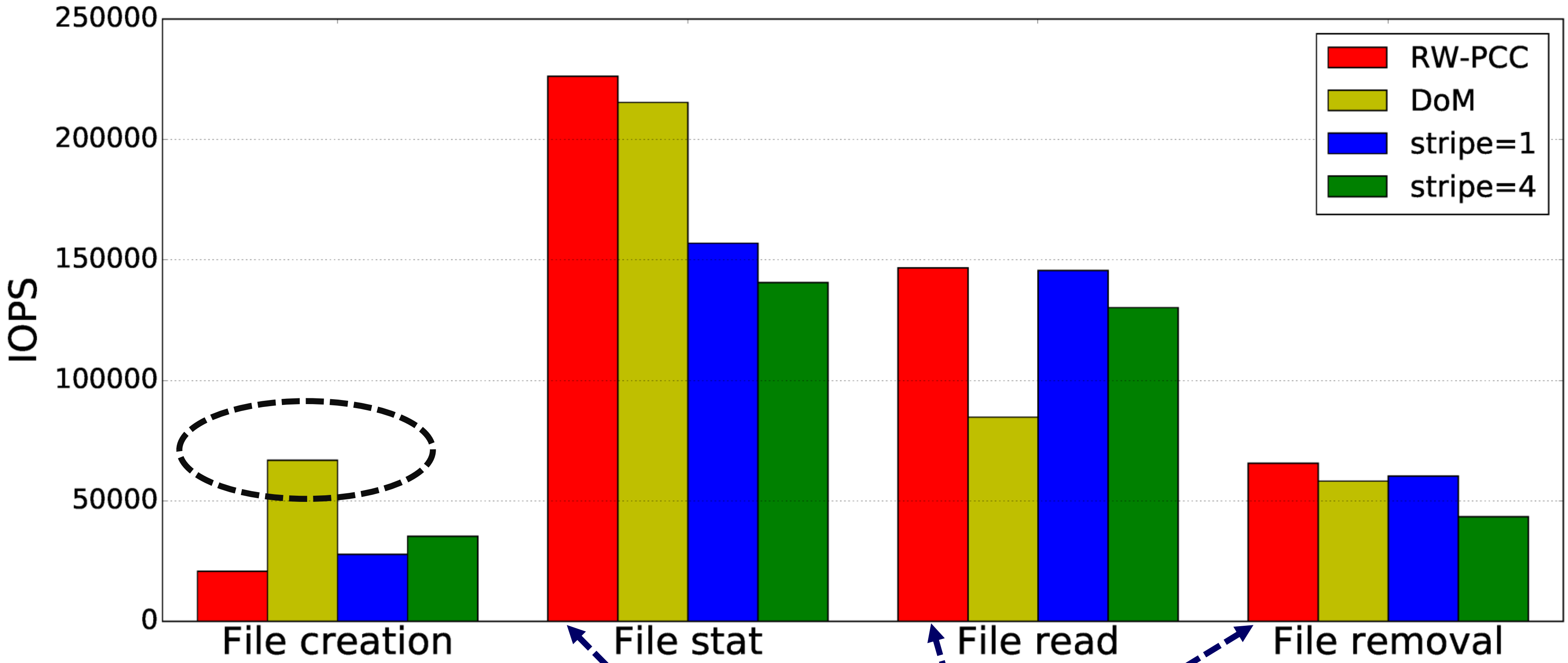
Single Thread Performance



RW-PCC Scalability Evaluation

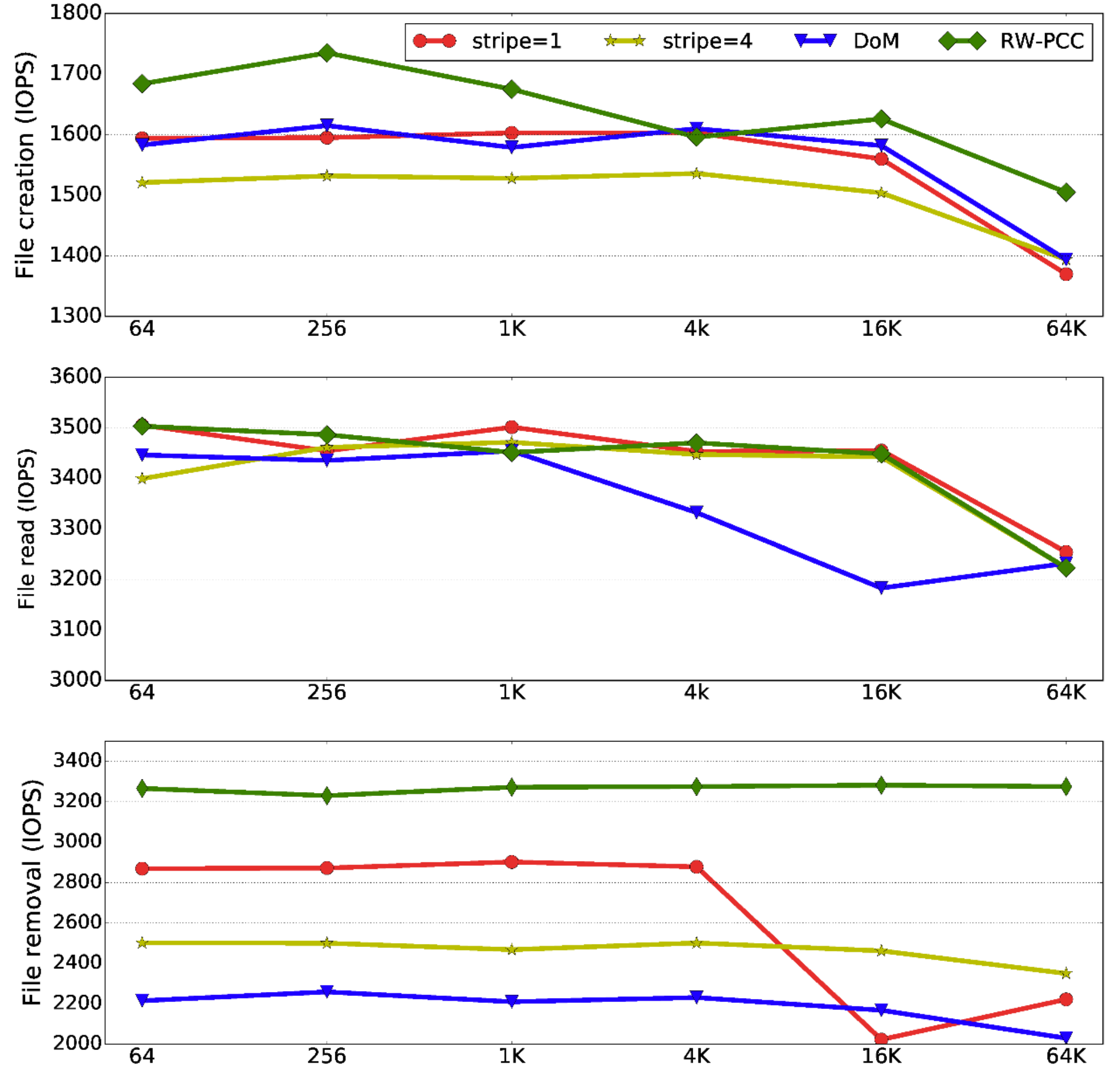


Metadata Performance



RW-PCC achieves the best

Small Files with Various Size

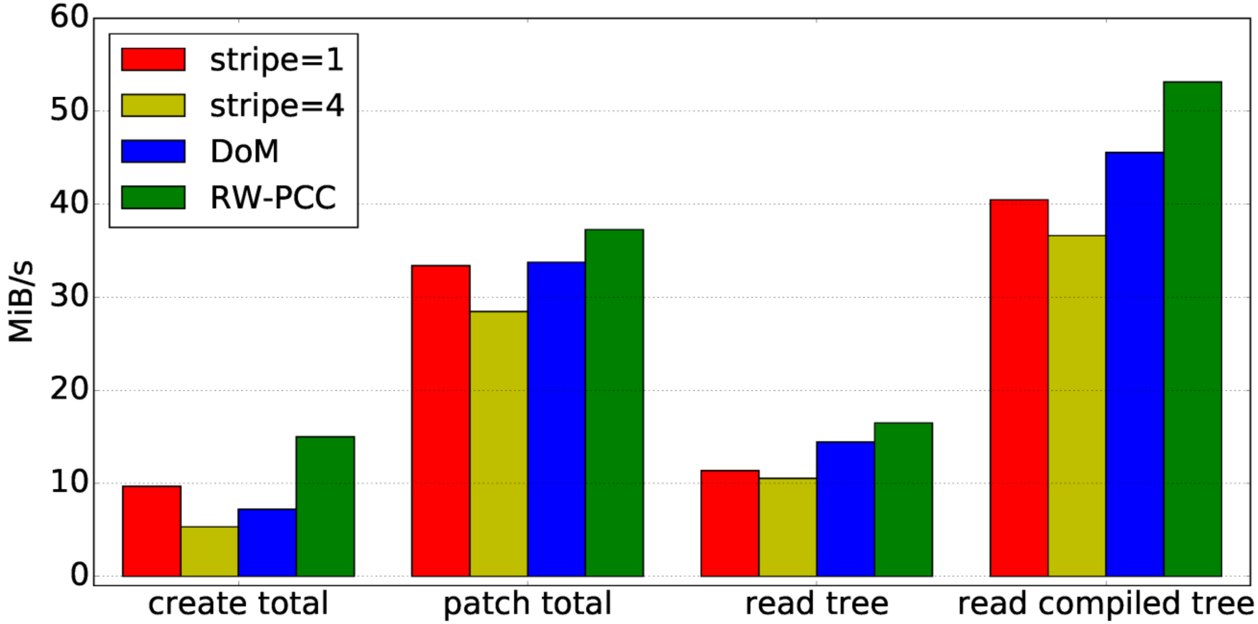


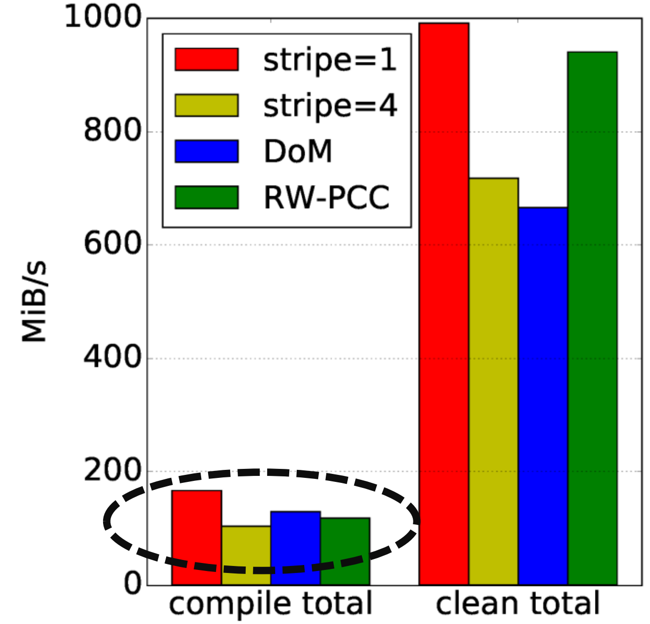
Metadata Performance Write Size =0

- File create with no data is slow
- But, the better small write performance on local SSDs compensated for this and allowed a speedup compared to DoM and standard lustre

Operation	File creation	File read	File removal
stripe=1	2776	3614	2974
stripe=4	2610	3552	2573
DoM	2793	3478	3126
RW-PCC	1714	3591	3271

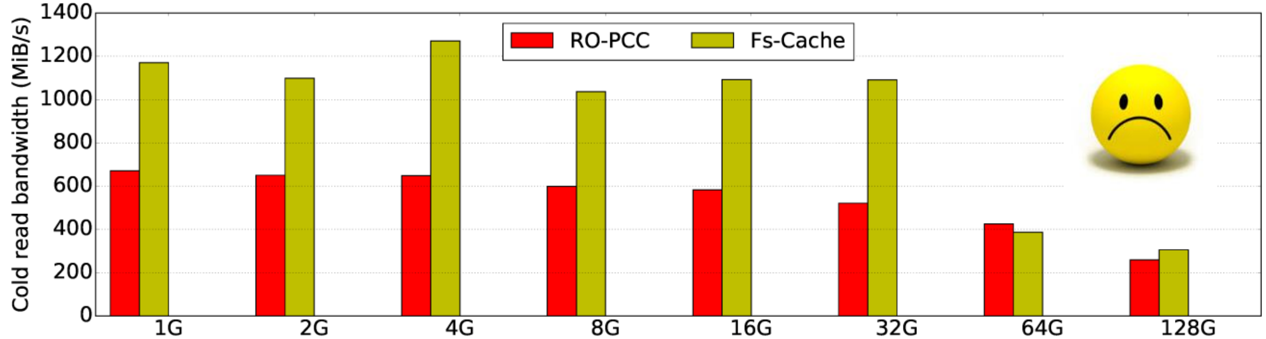
Small File for *Compilebench*



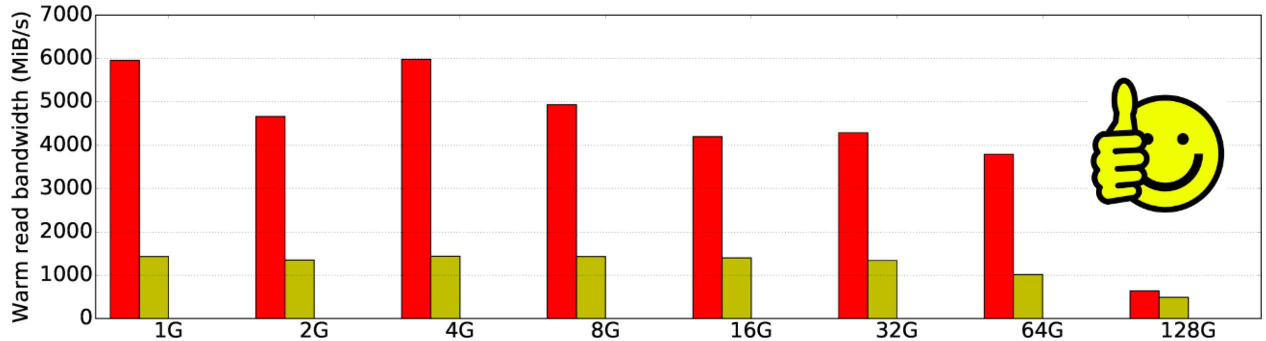


Read Performance

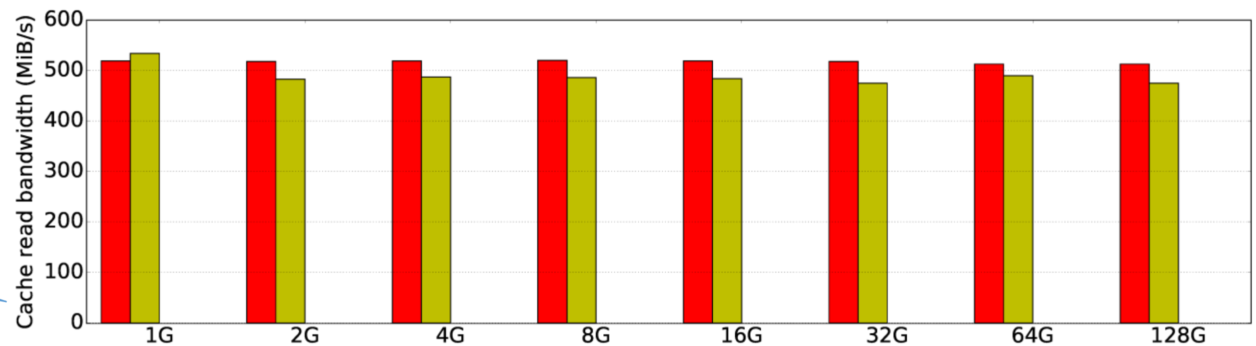
- File data is read for the first time and loaded into cache



- By repeating the test immediately after the “Cold” one



- Directly from the persistent cache after cleaning all page caches

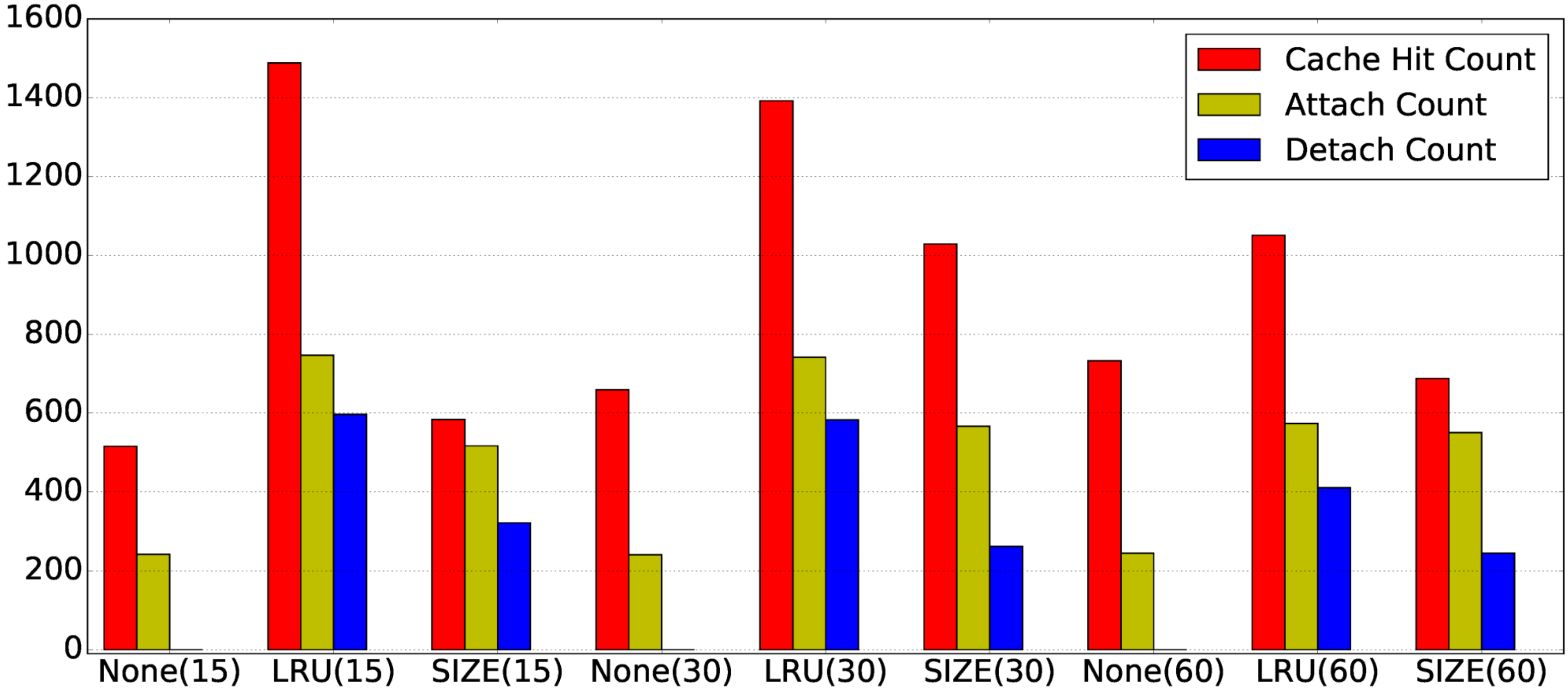


RO-PCC Scalability Evaluation

- RO-PCC performance in "Warm" and "Cache" state
- Scale nearly linearly with the increasing client number

Client count	1	2	4	8
Cold (MiB/s)	478	688	718	1389
Warm (MiB/s)	4374	8746	17520	34943
Cache (MiB/s)	521	1042	2074	4029

Metrics Statistic



"SIZE" evicts the least number of cached file

File Hit Rate

Summary

- A global namespace
  - Space efficient
  - Simple and transparent
- Less overhead, and network latencies and lock conflicts significantly reduced
- Simpler I/O stack: no interference with I/Os from other clients
- Small requirements on the HW inside the client nodes (SSD/HDD)
- LPCC reduces the pressure on the OSTs

Thanks for your attention!