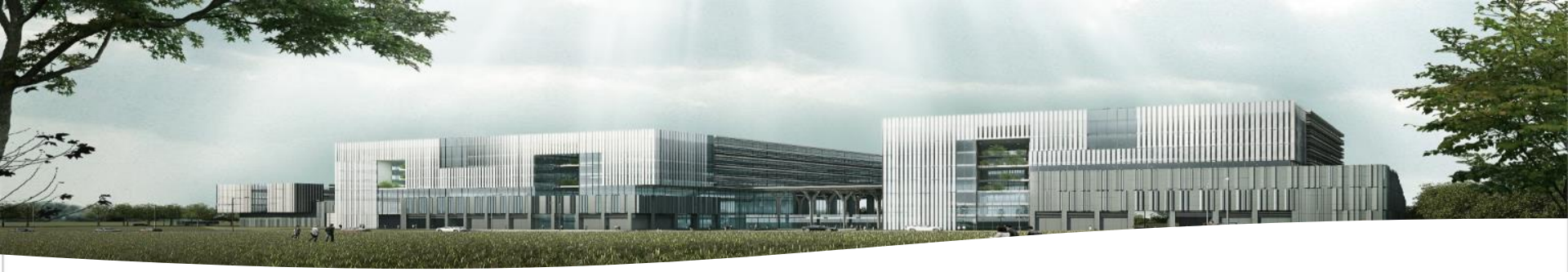


Datacenter Technology (Fall, 2018)

I/O Quality-of-Service for Lustre-based Exascale High Performance Computing

Lingfang Zeng (曾令仿)

Wuhan National Laboratory for Optoelectronics (WNLO)

Huazhong University of Science and Technology (HUST)

https://lingfangzeng.github.io/

With Contributions from




HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

JG|U
JOHANNES GUTENBERG UNIVERSITÄT MAINZ

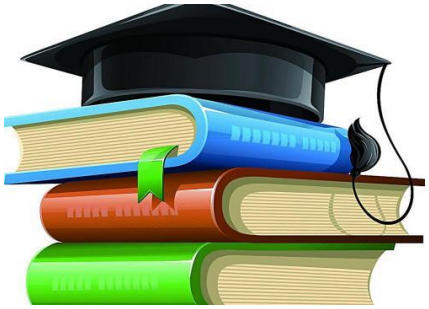
DDN STORAGE

intel

With Contributions from

- Prof. Dan Feng team @ **HUST**
  - **Wen Cheng, Chunyan Li, Fang Wang, et. al.**
- Prof. André Brinkmann team @ **JGU**
  - **Jürgen Kaiser, Tim Süß, et. al.**
- Yingjin Qian, Xi Li, Shuichi Ihara, Carlos Aoki Thomaz, and Shilong Wang @ **DDN**
- Andreas Dilger, and Peter Jones @ **Intel (now DDN Whamcloud)**
- Costin Iancu, and Khaled Ibrahim @ **LBNL**
- Thomas Stibor, and Walter Schön @ **GSI**
- Darrell Long, Frank Howley, Ethan L. Miller, and Yan Li @ **UCSC**
- Michael Kuhn, and Anna Fuchs @ **UHH**
- Julian Kunkel, and Thomas Ludwig @ **DKRZ**
- Florin Isaila @ **UC3M (Universidad Carlos III de Madrid)**

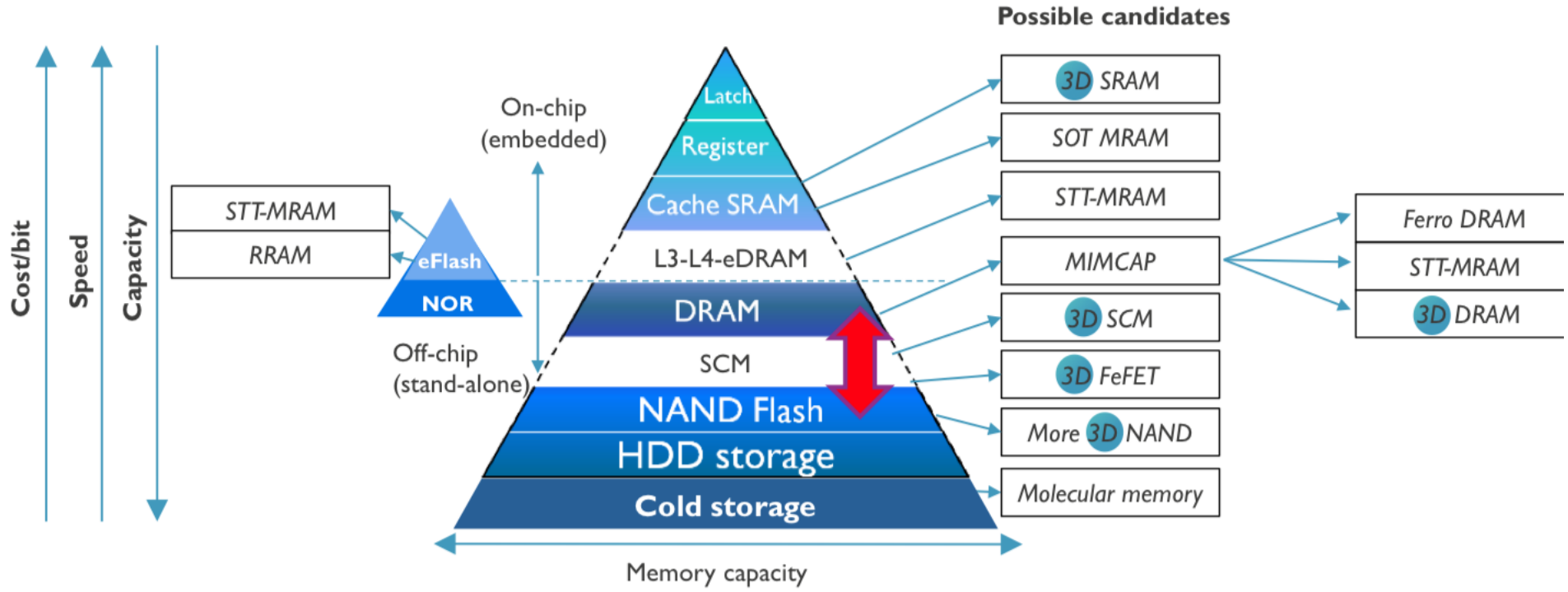
Outline



01 BACKGROUND

PROBLEM & TERMINOLOGY & OBJECTIVES

Hierarchical Storage Management (HSM)

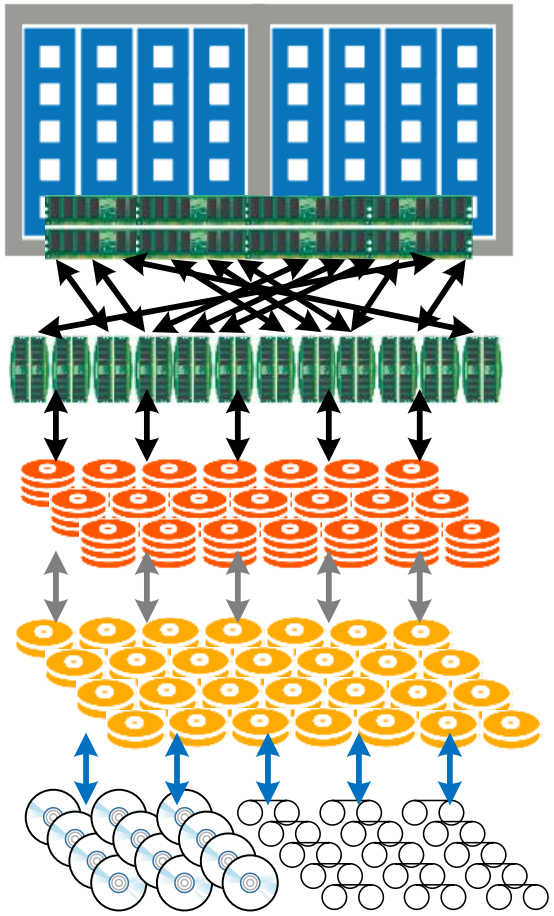


HPC workloads were too big to be stored only on flash

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

Challenge & Motivation

- Simulation, data analytics, deep learning
- Emerging HW (e.g. NVM, RDMA, GPU/TPU)
- Storage systems in HPC centers are usually shared by multiple organizations and various applications
- Various QoS solutions are necessary to satisfy different requirements of performance guarantee
  - Prevent crazy applications that congest the storage
  - Assure workloads of reliable bandwidth
  - Enable use cases outside the mainstream HPC, e.g. cloud
- Intelligent (support more complex access patterns)

Pre-Exascale Supercomputing

Feature	Titan	Summit
Application Performance	Baseline	5-10x Titan
Number of Nodes	18,688	~4,600
Node Performance	1.4 TF	> 40 TF
Memory per Node	32 GB DDR3 + 6 GB GDDR5	512 GB DDR4 + 96 GB HBM2
NV memory per Node	0	1600 GB
Total System Memory	710 TB	>10 PB DDR4 + HBM2 + Non-volatile
System Interconnect (node injection bandwidth)	Gemini (6.4 GB/s)	Dual Rail EDR-IB (25 GB/s)
Interconnect Topology	3D Torus	Non-blocking Fat Tree
Processors	1 AMD Opteron™ 1 NVIDIA Kepler™	2 IBM POWER9™ 6 NVIDIA Volta™
File System	32 PB, 1 TB/s, Lustre®	250 PB, 2.5 TB/s GPFS™
Peak power consumption	9 MW	15 MW

Reference: https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/summit-early-science-program-call-for-proposals/

Lustre File System

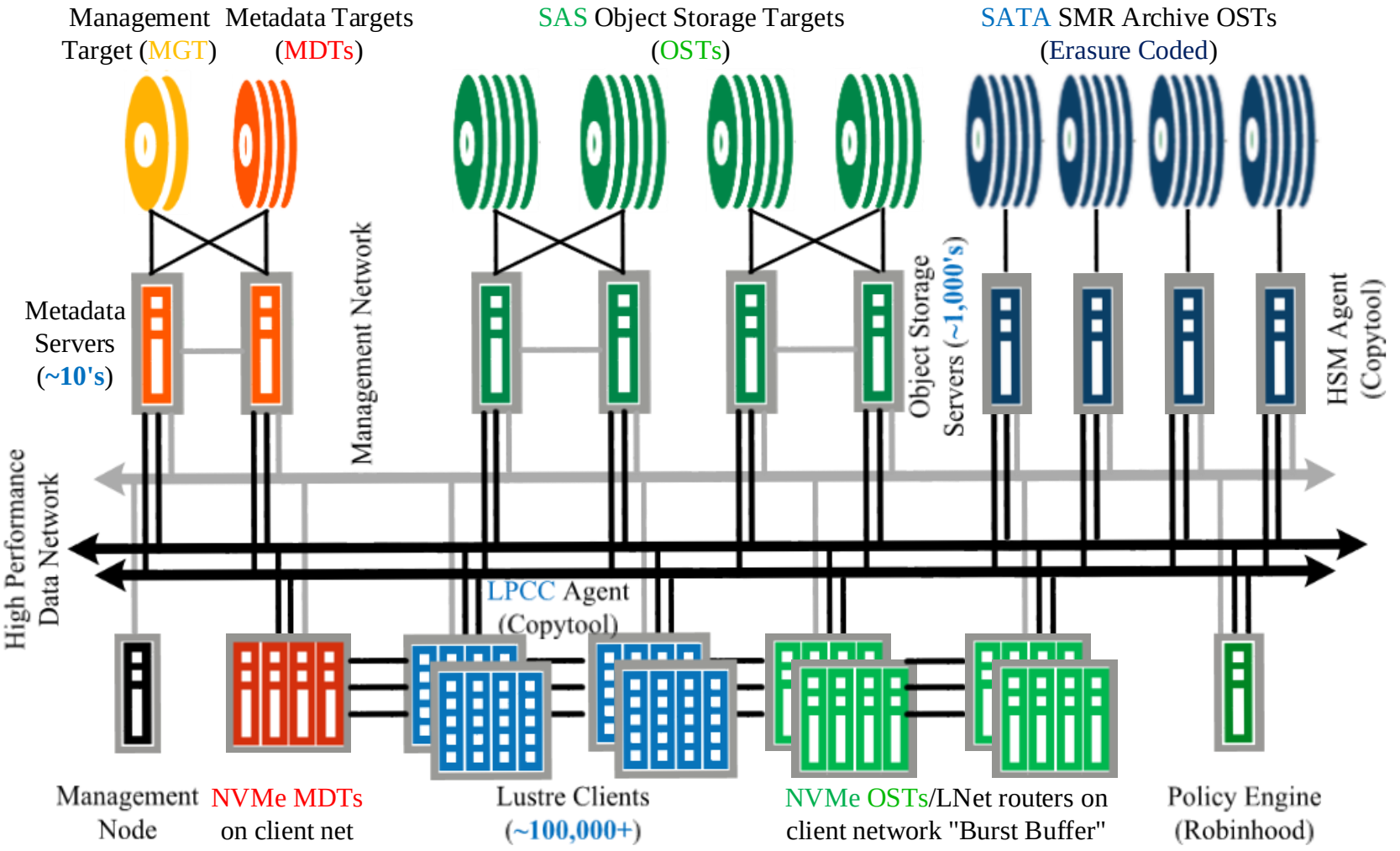
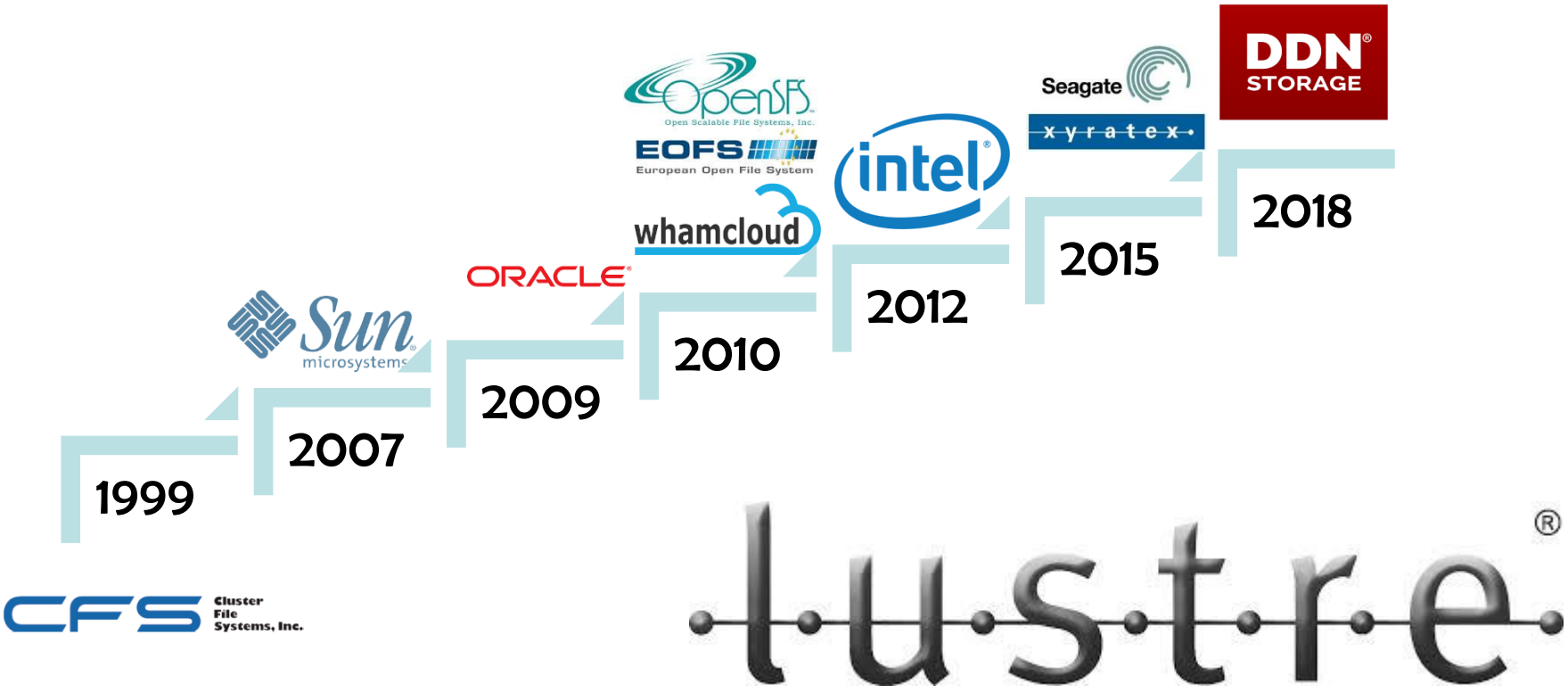


Figure based on Andreas Dilger's Lustre User Group 2018 presentation: Lustre 2.12 and beyond (see http://opensfs.org/lug-2018-agenda/)

About Lustre* Journey



Referece: https://en.wikipedia.org/wiki/Lustre_(file_system)

Solutions for Lustre QoS

- Lustre NRS-TBF
  - SC'17
  - (≤) Lustre 2.10

- QoS Planner
  - Lustre User Group conference 2017 (LUG'17)
  - JGU HPC production system

Reference: http://sc17.supercomputing.org/presenter/?fn=lingfang&ln=zeng&uid=430273
Reference: http://cdn.opensfs.org/wp-content/uploads/2017/06/LUG-2017-new2.pdf

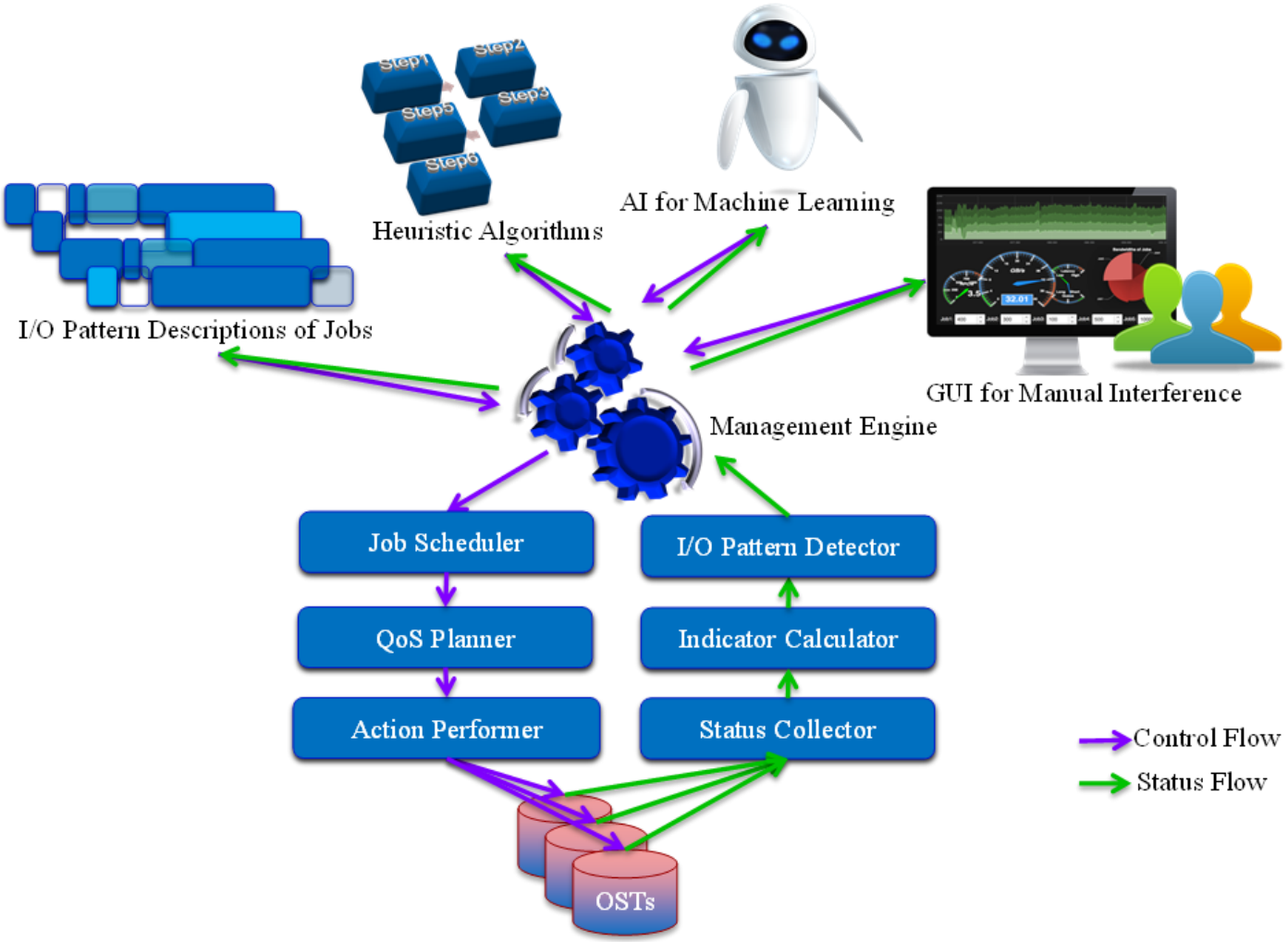
Solutions for Lustre QoS

- Lustre Intelligent Management Engine (LIME)
  - Lustre Administrator and Developer workshop 2018 (LAD'18)

- Lustre Persistent Client Caching (LPCC)
  - LUG China 2018 (keynote)
  - Lustre 2.13 or 2.14

Reference: https://www.eofs.eu/events/lad18
Reference : http://lustrefs.cn/clug2018/

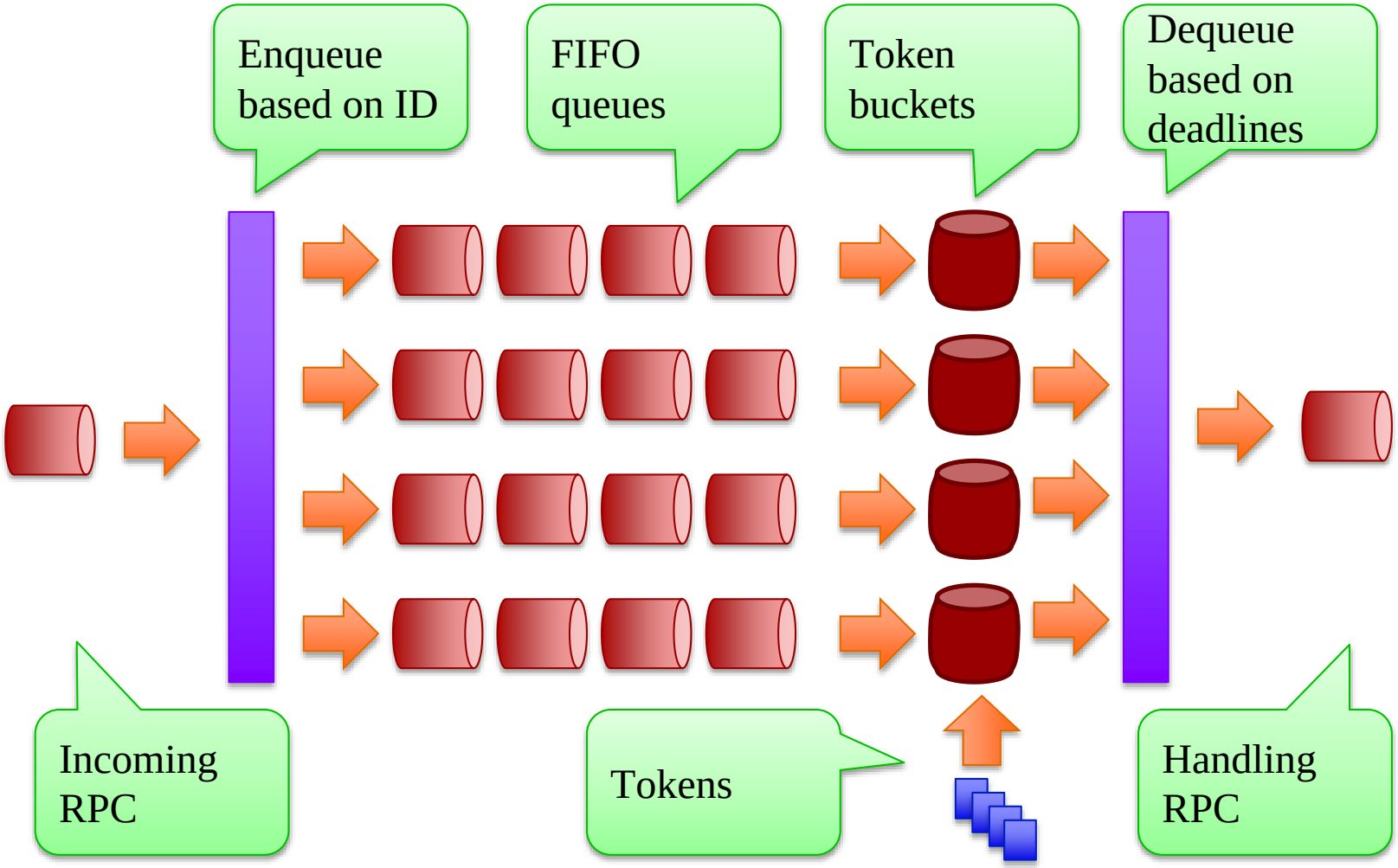
Framework


Step1
Step2
Step3
Step5
Step6
Heuristic Algorithms
AI for Machine Learning
I/O Pattern Descriptions of Jobs
GUI for Manual Interference
Management Engine
Job Scheduler
I/O Pattern Detector
QoS Planner
Indicator Calculator
Action Performer
Status Collector
OSTs
Control Flow
Status Flow

02 NRS-TBF

CLASSFUL TOKEN BUCKET FILTER

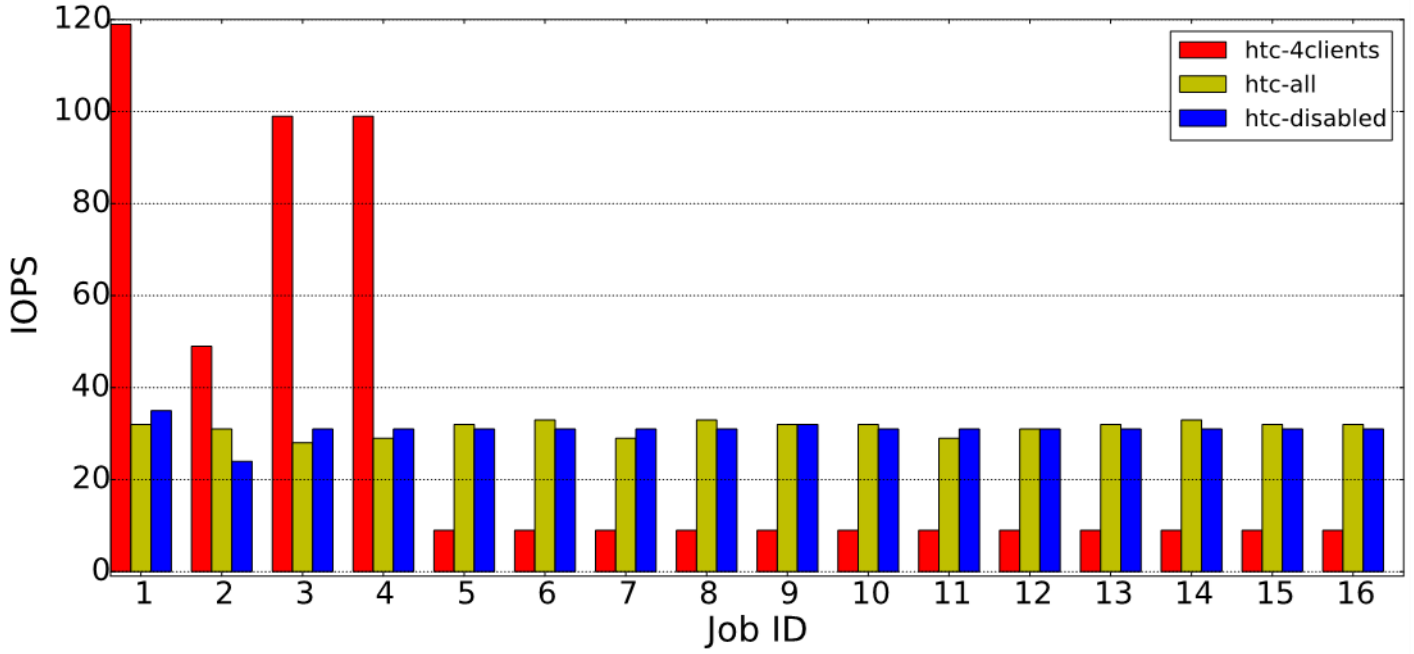
Classful Token Bucket Filter



Classful Token Bucket Filter (cont.)

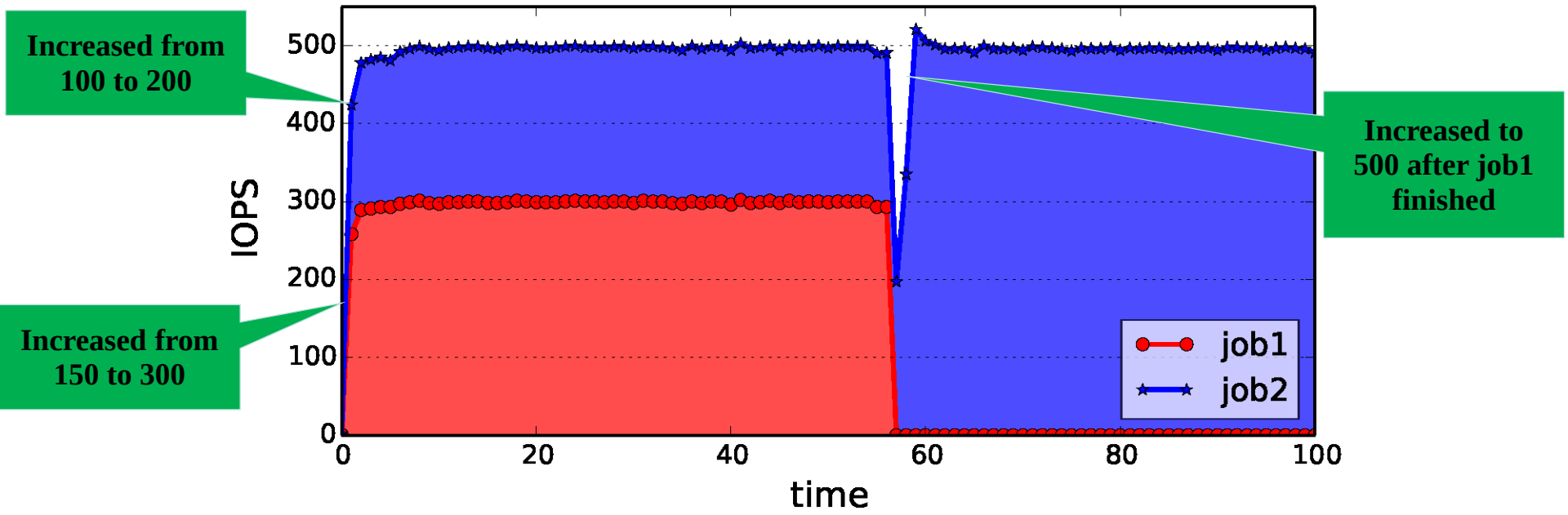
- Hard Token Compensation (HTC)
- Proportional Sharing Spare Bandwidth (PSSB)
- Dependent Rules
- Global Rate Limiting (GRL)
  - Enforce system-wide RPC rate limitations

HTC Evaluation



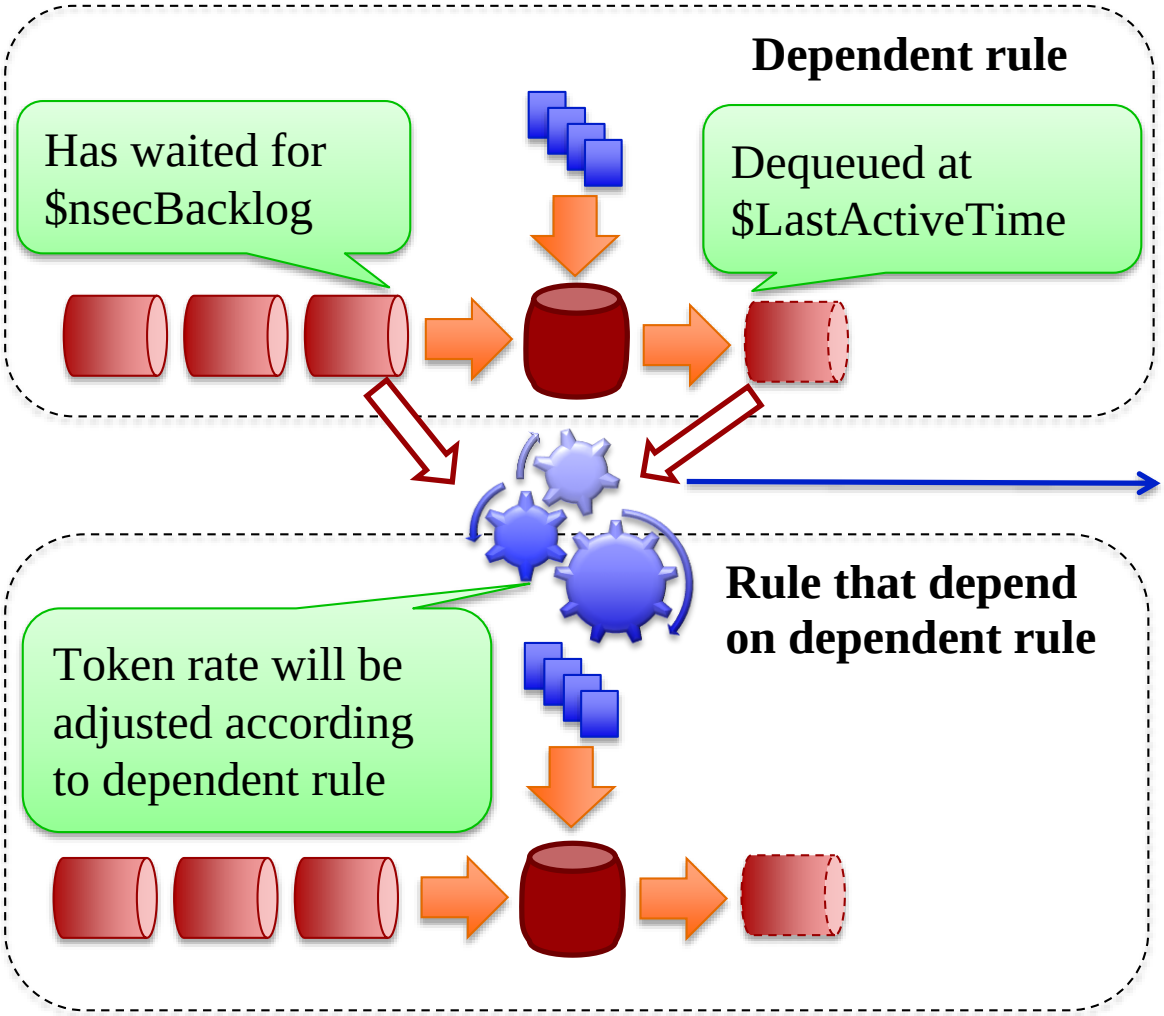
- 16 clients used, each running a different job with a rate limit of 100, except the first two clients' jobs with limits of 120 and 50
- Total bandwidth requirement is larger than the system capacity of around 500 (buffered) IOPS

PSSB Evaluation



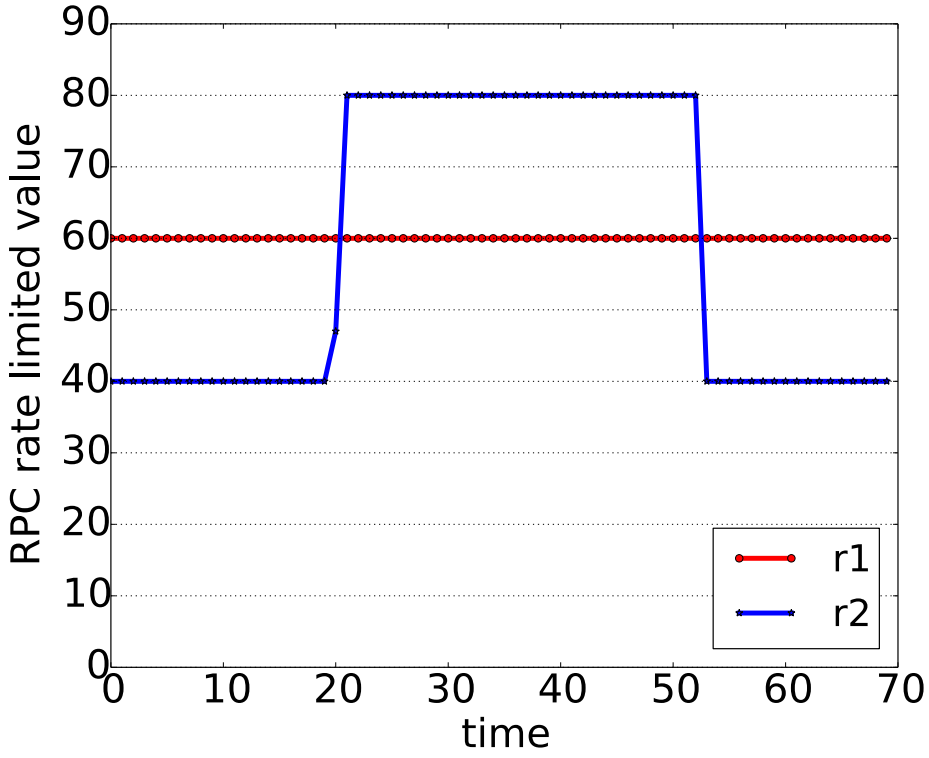
- 16 clients working in parallel on job1; each client writes 1GB data at an initial rate setting of 150
- Additional 16 clients run job2, each writing 2GB data at a rate setting of 100

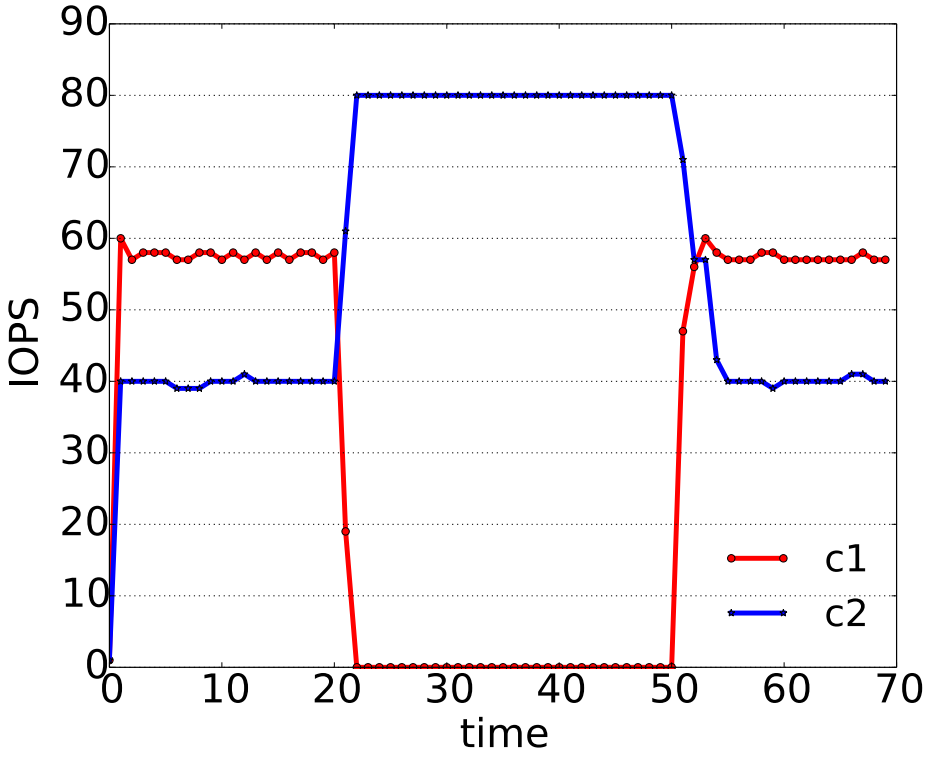
Dependent Rules



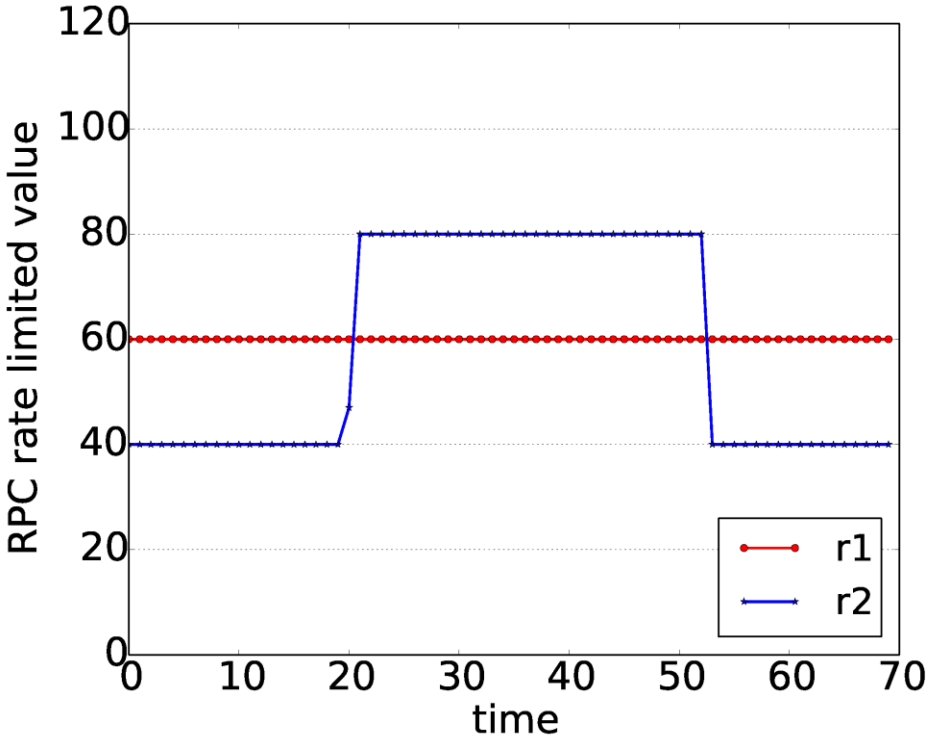

```
1  procedure UpdateRuleRate (rule, deprule)
2      passed = now - deprule.lastActiveTime;
3      if passed > λ* deprule.nsecs &
       deprule.nsecsBacklog < β * deprule.nsecs then
4   # increase the RPC rate
5          rule.speedup <<= 1
6      else
7          rule.speedup >>= 1
8      end if
9      rule.rate = rule.speedup + rule.lowerrate
10     cap rule.rate at rule.upperrate
11 end procedure
```

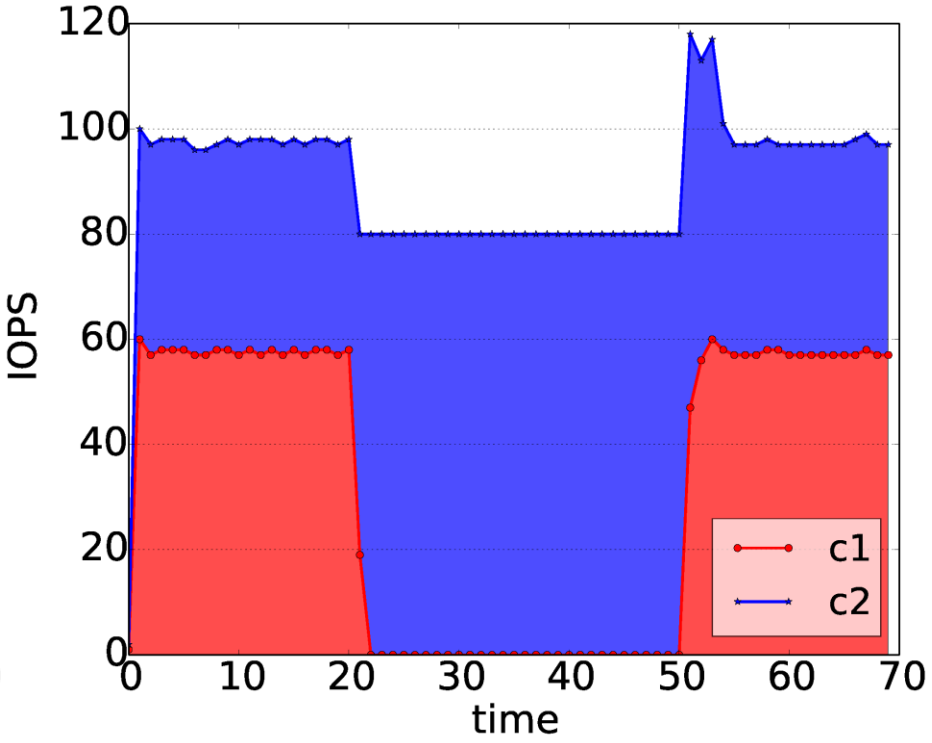
Evaluation of Dependent Rules





Evaluation of Dependent Rules



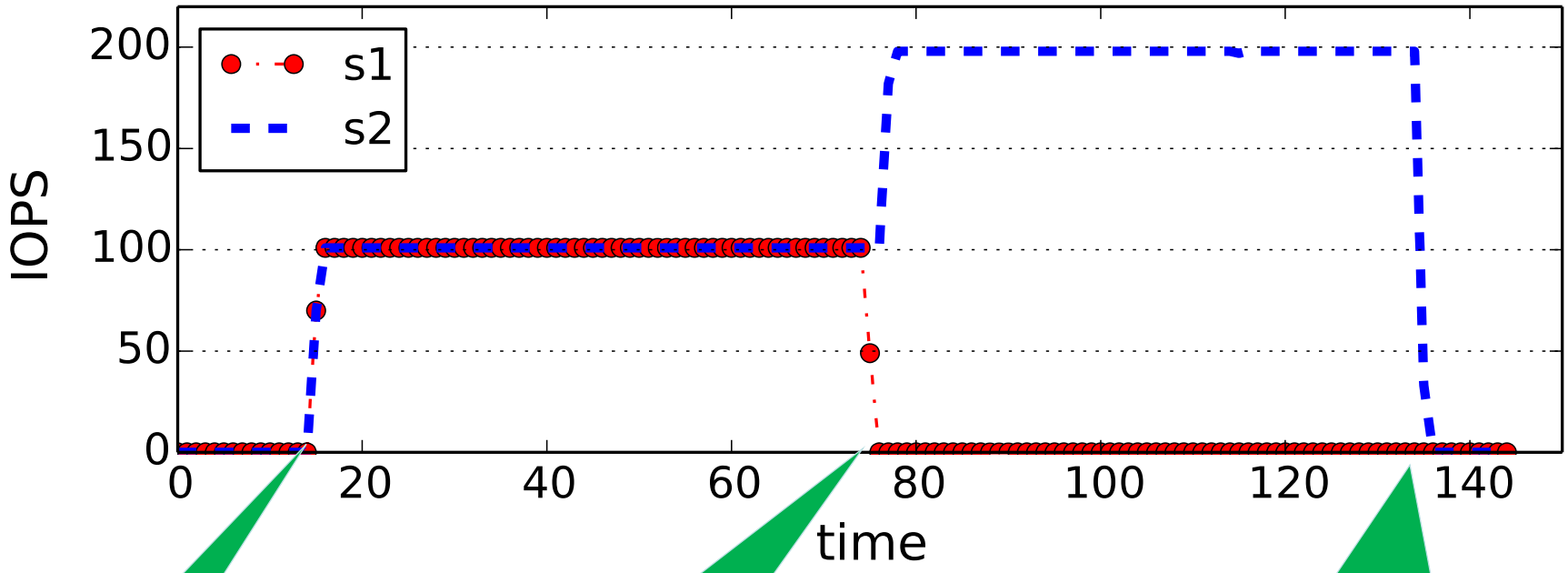


Global Rate Limiting (GRL)

Algorithm 2 Global Rate Limiting algorithm

1: // N : number of storage servers
2: // C_i^j : class corresponding to C_i on j^{th} server
3: // R_i : global rate limit for C_i
4: // R_i^j : the setting of the rate limit for C_i^j
5: // M_i^j : the measured rate for C_i^j
6: // Q_i^j : current request queue depth for class C_i^j
7: // σ : bandwidth assigned to I/O inactive classes
8: // C_i is the set of C_i^j sorted in descending order of M_i^j
9: let A_i be the set C_i with $M_i^j > 0$ or $Q_i^j > 0$
10: let K be the number of elements in A_i
11: **for** $j \in \{0, \ldots, K-1\}$ **do**
12: $\quad R_i^j = (R_i - \sigma)/K$
13: **end for**
14: **for** $j \in \{K, \ldots, N-1\}$ **do**
15: $\quad R_i^j = \sigma/(N-K)$
16: **end for**

GRL Evaluation



Both threads started at 15s

At 75s, t1 terminated, rate limitation of two servers changed to 1 and 199

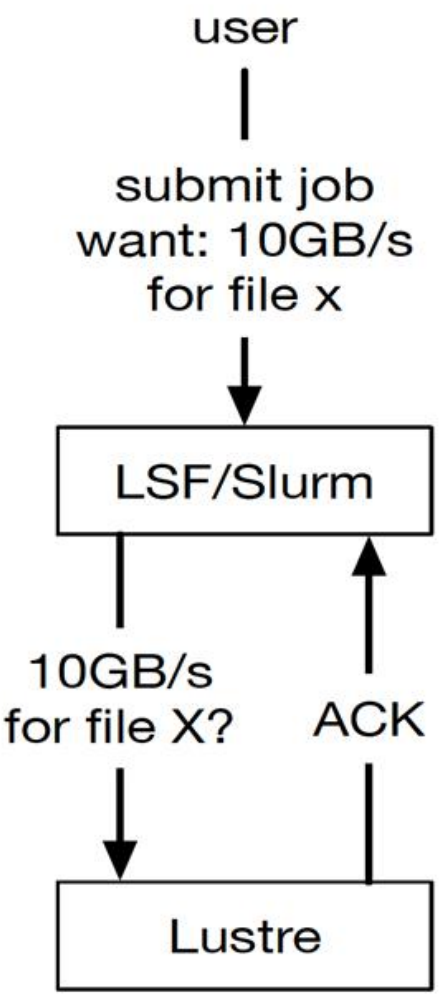
At 135s, t2 terminated, rate limitation of two servers both changed to 100

03 QOSPLANNER

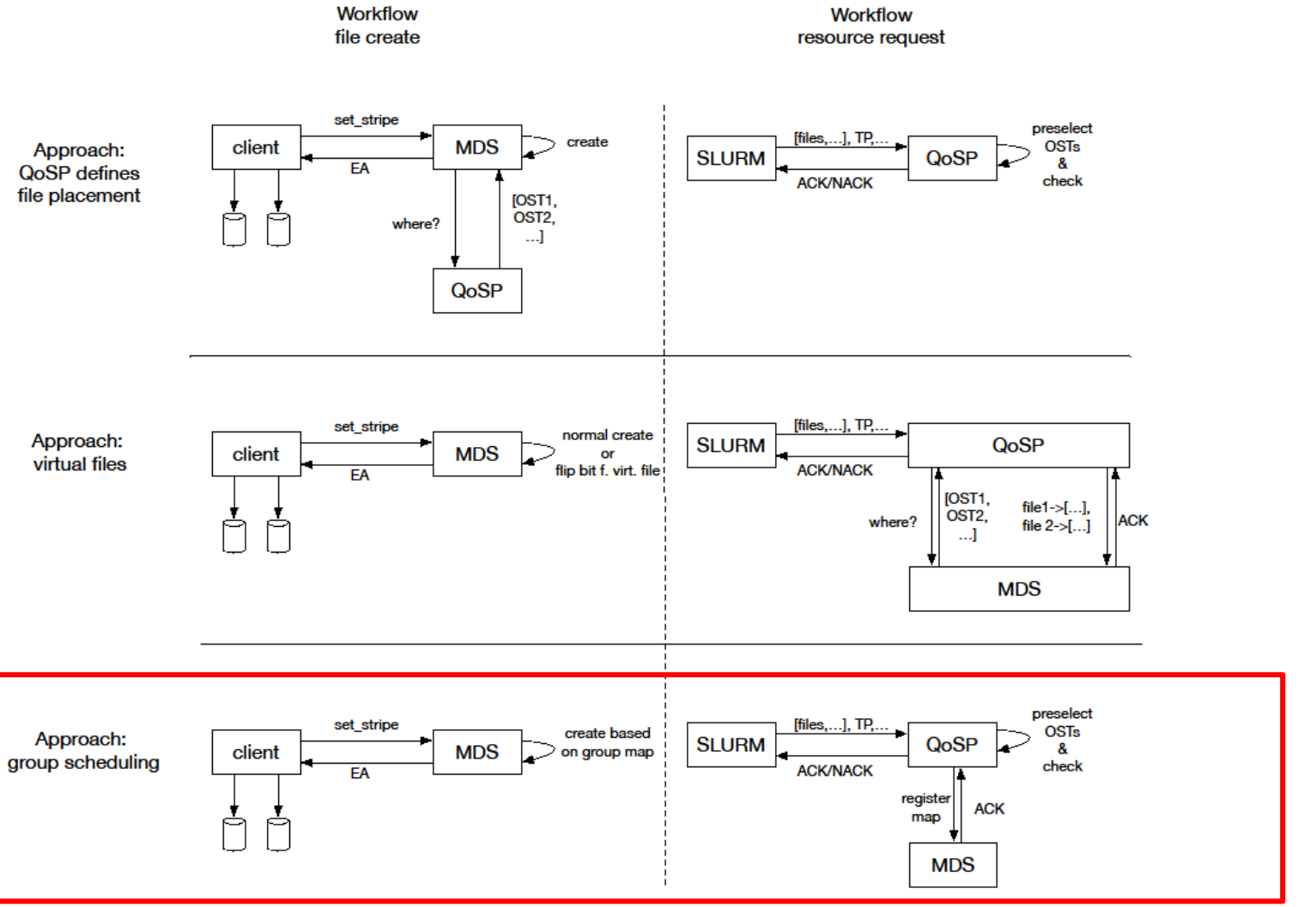
PROVIDING QOS-MECHANISMS FOR LUSTRE

QoS Planning in Lustre

- QoS Planning for storage resources
  - Guarantee x GB/s read throughput
  - Guarantee y GB/s write throughput
  - For specific files?
- Architecture includes
  - Batch System
  - Client and/or server component in Lustre enforcing QoS



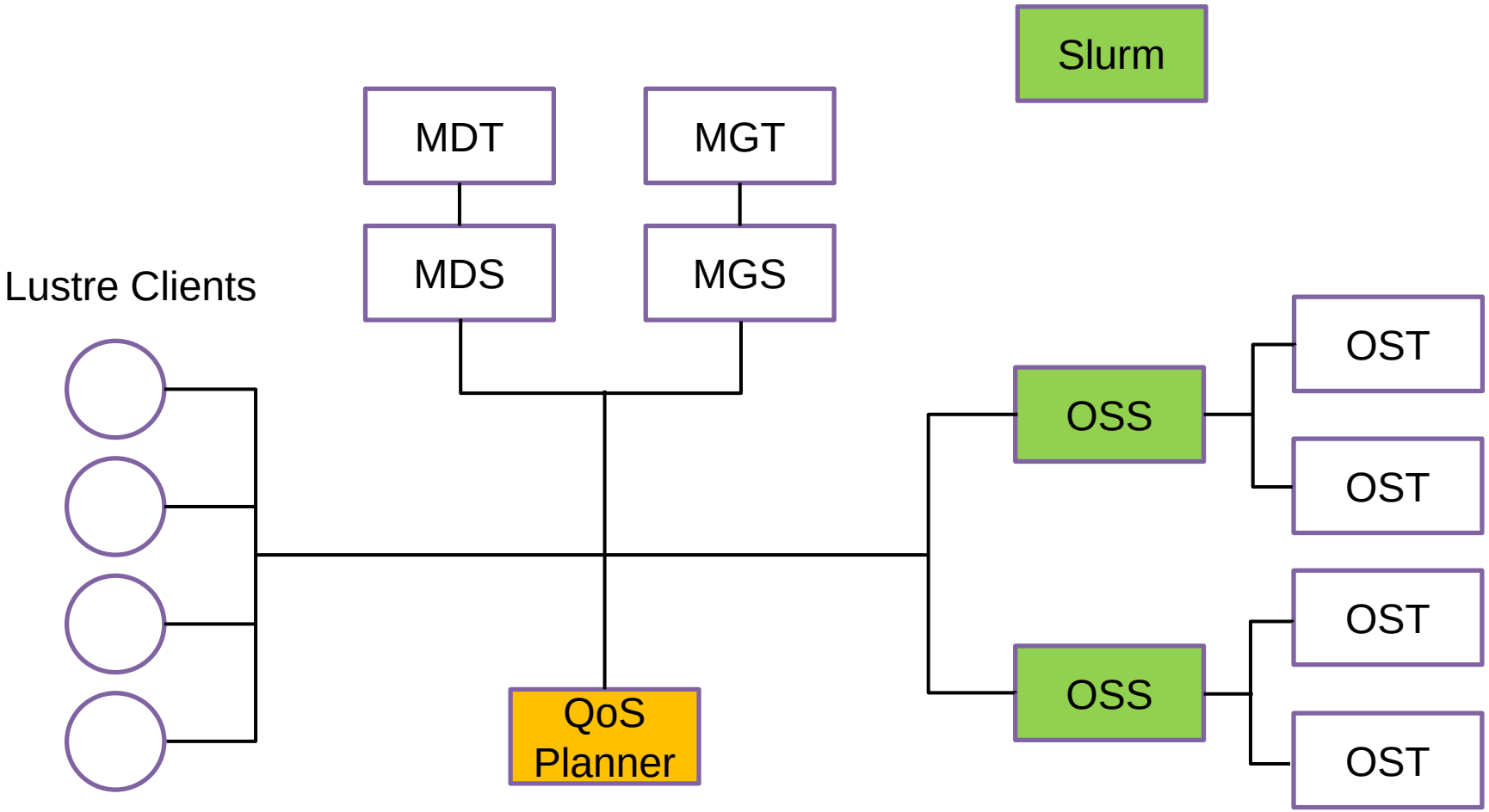
Write Throughput Handling



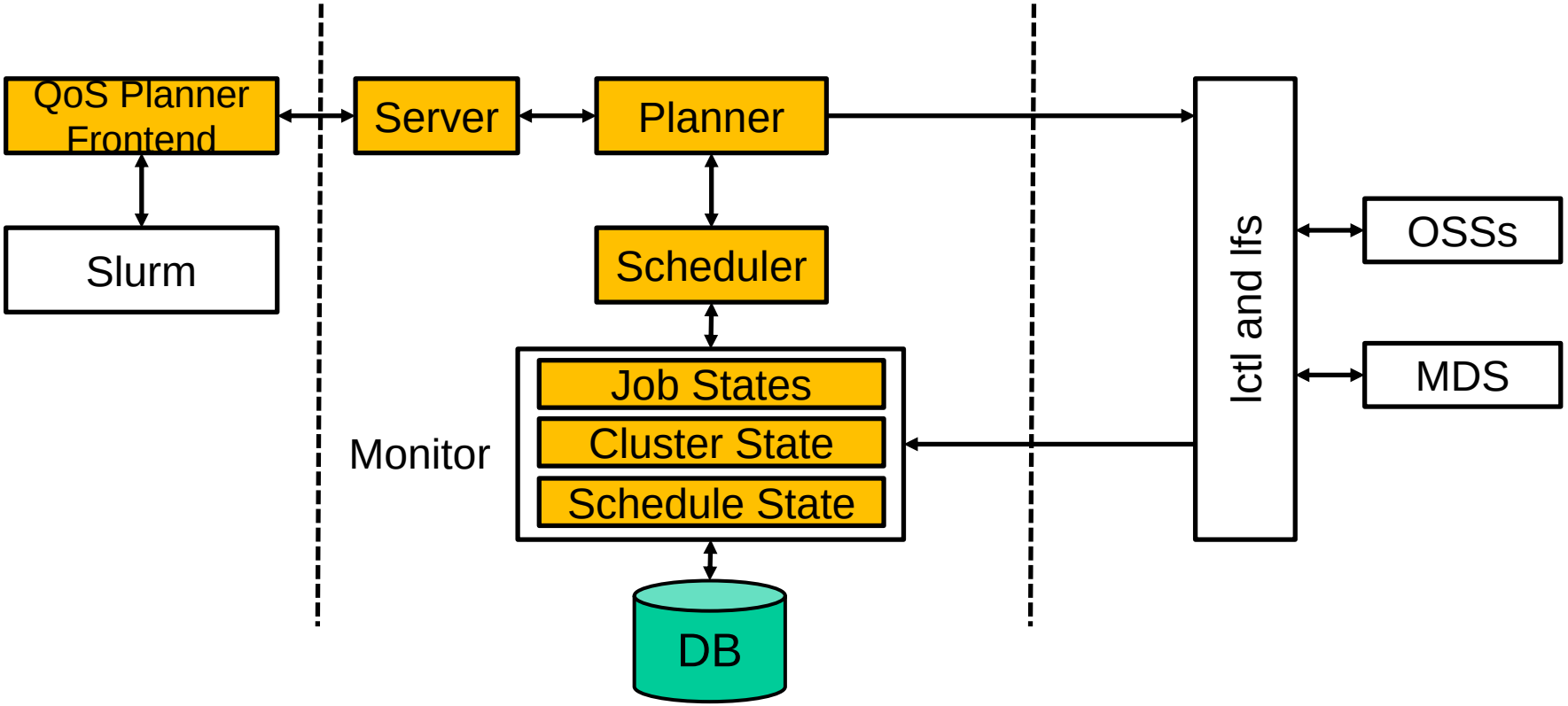
Phased I/O

- Simple
  - QoSP must maintain a schedule of all I/O phases of all jobs and readjust the NRS settings when the next I/O phase change is due
- Negotiate at submission time
  - However, only few job schedulers implement a planning-based scheduling
- Negotiate at runtime
  - Jobs only reserve a basic I/O throughput (which covers the steady state operations) and then negotiate the start and end time of their I/O phases with the QoSP at runtime

Architecture including QoS-Planner



QoSPlanner Current Approach



QoSPlanner & Slurm Integration

- Bandwidth is defined as a global and as a local resource
- Slurm plug-in controls:
  - Globally available bandwidth - treated as *license* (one license/MB)
  - Local bandwidth - treated as *generic* resource
- Job gets rejected if one resource is not available
- Example:
  - `srun -N1 -p bigmem -A system --gres=qoslustre:100M -L lustreqos:100 sleep 5`

QoSPlanner: A priori Reservation

- A client application for reserving bandwidth has been developed for Slurm
  - `# qosp reserve -throughput 100 -duration 100 \`
    `-filenames /path/to/folder -id=slurm_job_id`
- Command reserves a **throughput** of 100 RPCs for 100 seconds
- OSTs are identified via **filenames** respectively paths
- Available shares can be identified via **id**

QoSPlanner: A priori Reservation

- Slurm-plugin uses `qosp` command for reserving bandwidth Throughput is taken from global and local resource
- Further integrations are possible:
  - Coupling users or groups with QoS manager
  - Credit bandwidth of reservations that terminate earlier

QoSPlanner: Spontaneous Reservation

- Many programs require high I/O bandwidth only for a short time period
  - Loading input data during initialization
  - Checkpointing
  - Storing final results
- We provide a C++ API for spontaneous I/O accesses
  - Reserve bandwidth for a certain time span
  - Test if reservation is available
  - Remove reservation after I/O is done

QoSPlanner: Spontaneous Reservation

- Most important API functions:
  - // none-blocking reservation
    string addReservationAsync(int tp, int sec, string fs);
  - // blocking reservation
    string addReservationSync(int tp, int sec, string fs);
  - // delete a specific reseravtion
    bool removeReservation(string id);
  - // test the status of a reservation
    // (UNDEFINED, SCHEDULED, ACTIVE)
    // required for asynchronous reservation
    int testReservation(string id);

Spontaneous Reservation

- QoS scheduler currently uses backfilling, thus a reservation start time may change during waiting period
- Asynchronous functions supports this behavior
  - // none-blocking reservation
    string addReservationAsync(int tp, int sec, string fs);
  - // test the status of a reservation
    int testReservation(string id);
- Programs like ESPResSo++ or tools like SCR can use these features to request bandwidth for asynchronous checkpoints

ESPResSo++

- After every simulation step the checkpointing function DumpXYZQoS::dump() is called
  - void DumpXYZQoS::dump() {
    if(!qos_waiting){
    qosId = qosp.addReservationAsync(1000, 10, filename());
    qos_waiting = true;
    conf.gather()
    }
    if(qosp.testReservation(qosId) != ACTIVE) return;
    qos_waiting = false;
    ... // write checkpoint
    }

04 LIME

LUSTRE GLOBAL QOS MANAGEMENT

Basic Mechanisms inside Lustre for LIME

- Implemented: NRS TBF policy
  - QoS policies for object allocation that can be controlled by external tools
  - Client side QoS based on jobid
- A global performance monitoring system
  - Analysis of I/O patterns
  - Summarize statistics
- A centralized management framework
  - Configure global TBF rules on all OSS/MDS
  - Make decisions according to statistics
  - Enforce consistent policies across the whole file system
- Collaboration from users of the file system
  - Users should have enough motivation to optimize their application
  - Penalty will be enforced for bad behaviors
  - High-priority users/application have higher I/O rates

Decay policy of LIME

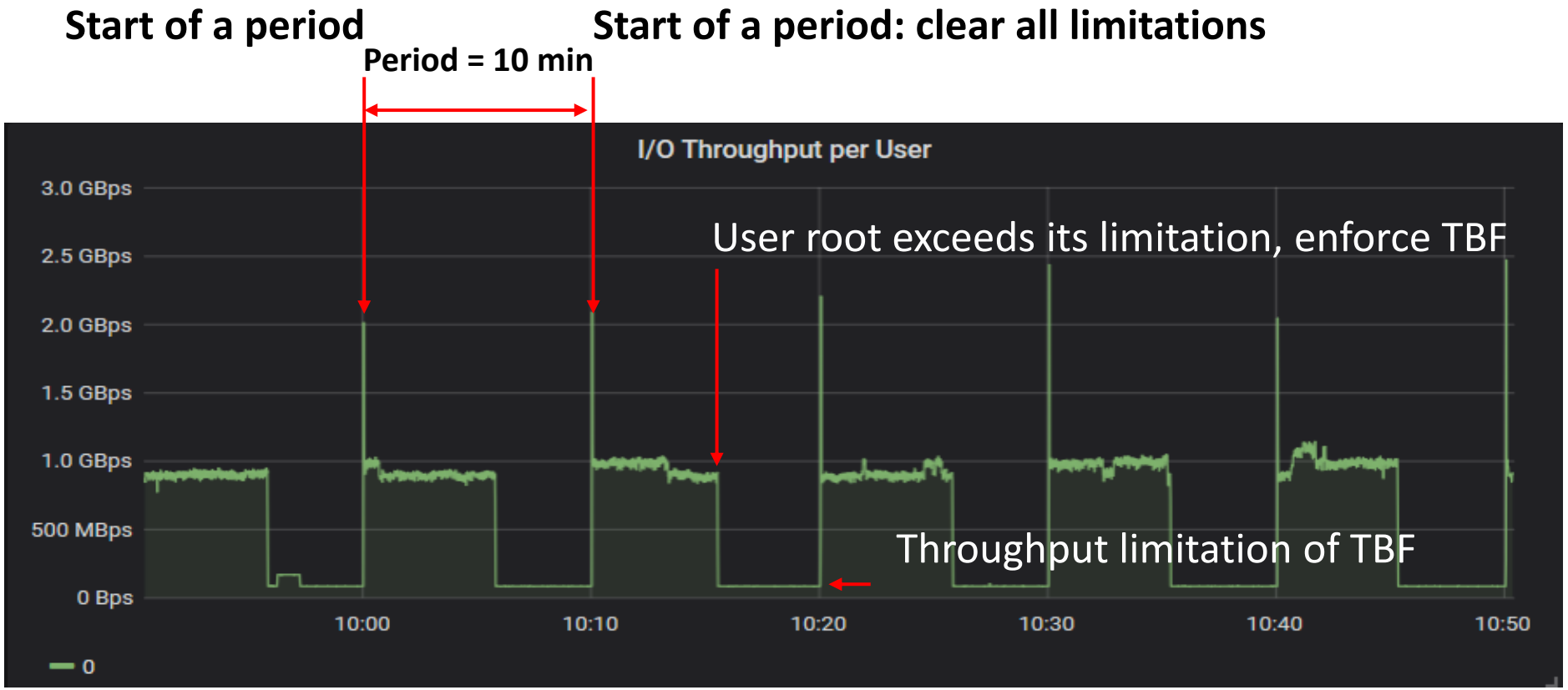
- Time period of 24 hours
- Throughput/IOPS will be recorded for all users
- Use QoS warning to notify users when throttling their I/O rate

LIME: Lustre Intelligent Management Engine

- Lustre statistics collector based on Collectd
  - Supports different Lustre versions: 1.8/2.5/2.7/2.10/…
  - Collects all kind of statistics from Lustre /proc or /sys entires
- Time-series database based on Influxdb
  - LIME can query the database for statistics during a time period
- Monitoring GUI based on Grafana
- System management framework
  - The control center can SSH to a cluster of nodes and execute commands
- Different QoS Policies for different purposes
  - “Decay” Policy to enforce a throughput/IOPS quota

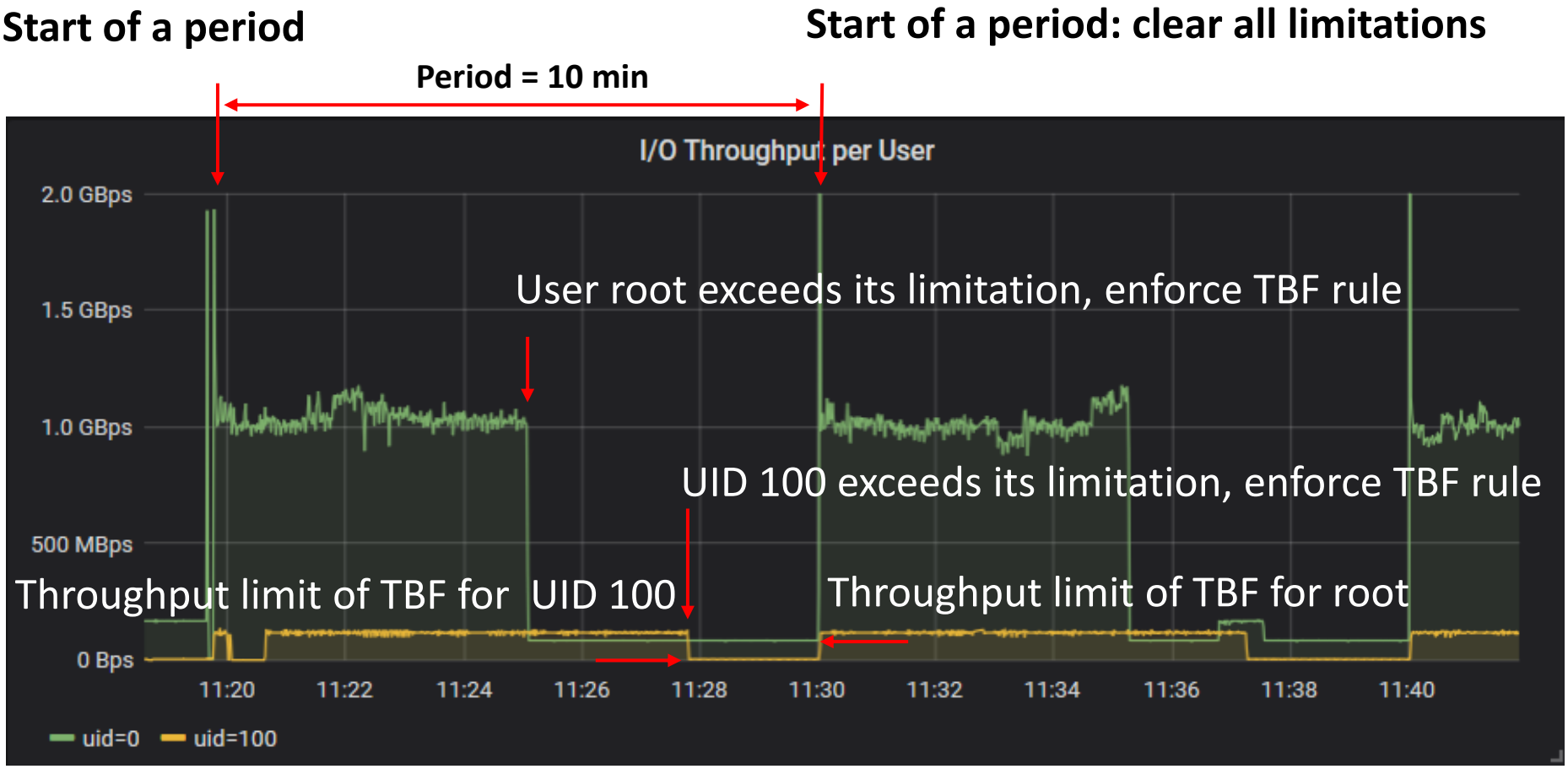
Test Result of Decay Policy – I/O throughput(1)

- I/O pattern: dd if=/dev/zero of=/lustre/file bs=1048576



Test Result of Decay Policy – I/O throughput(2)

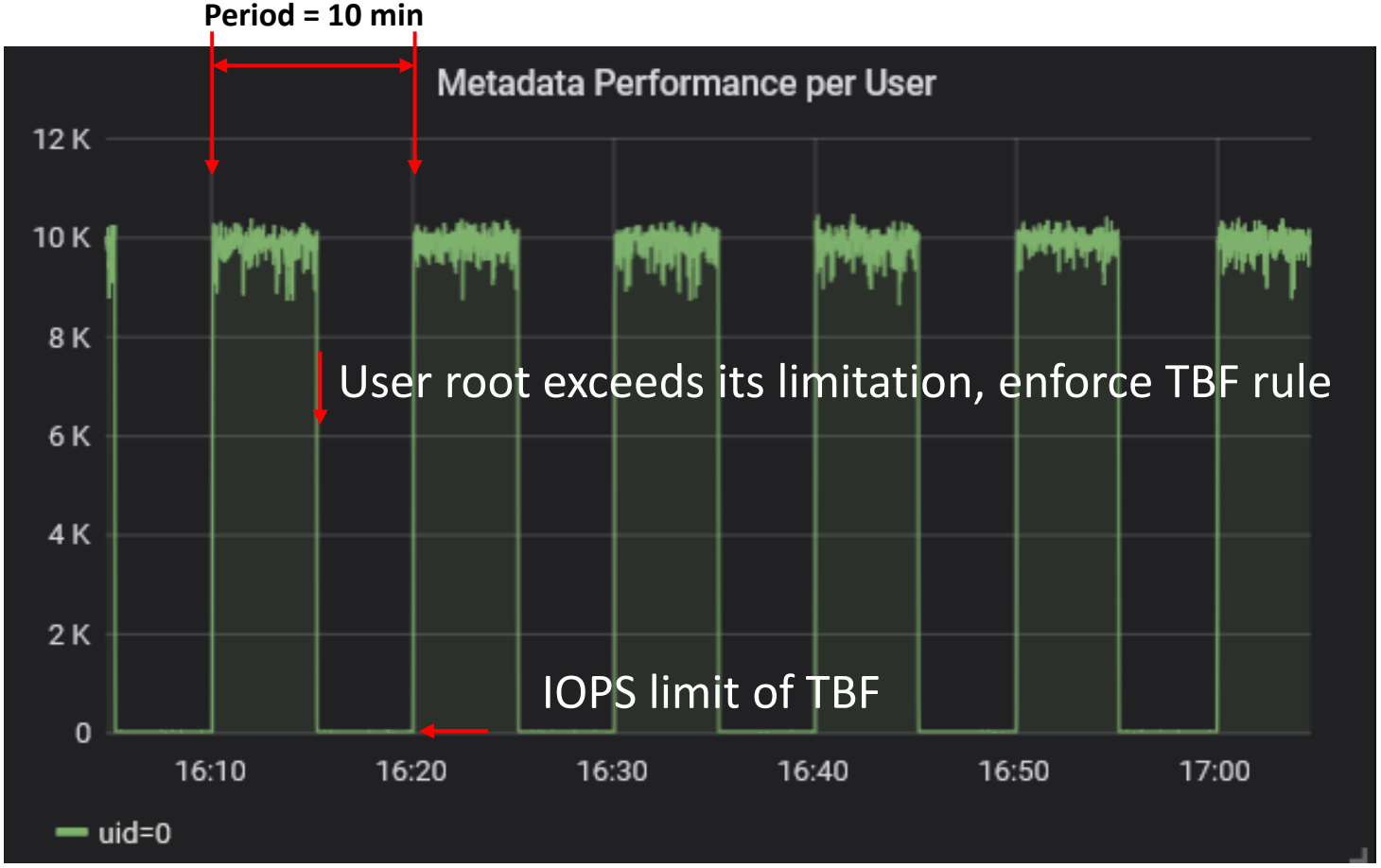
- I/O pattern: dd if=/dev/zero of=/lustre/file bs=1048576



Test Result of Decay Policy – Metadata Performance(1)

- I/O pattern: repeatedly create and remove files

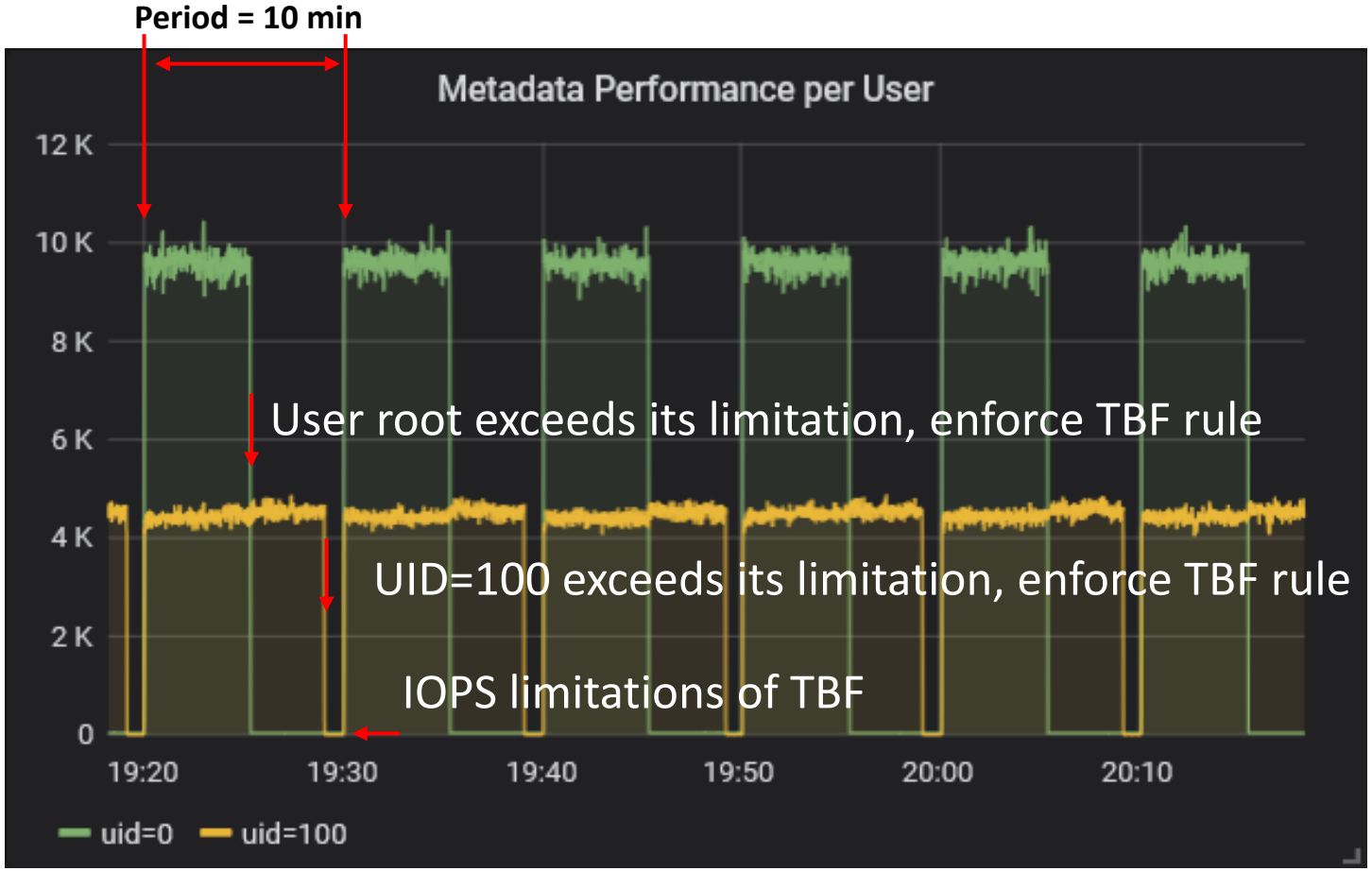
Start of a period **Start of a period: clear all limitations**



Test Result of Decay Policy – Metadata Performance(2)

- I/O pattern: repeatedly create and remove files

Start of a period **Start of a period: clear all limitations**

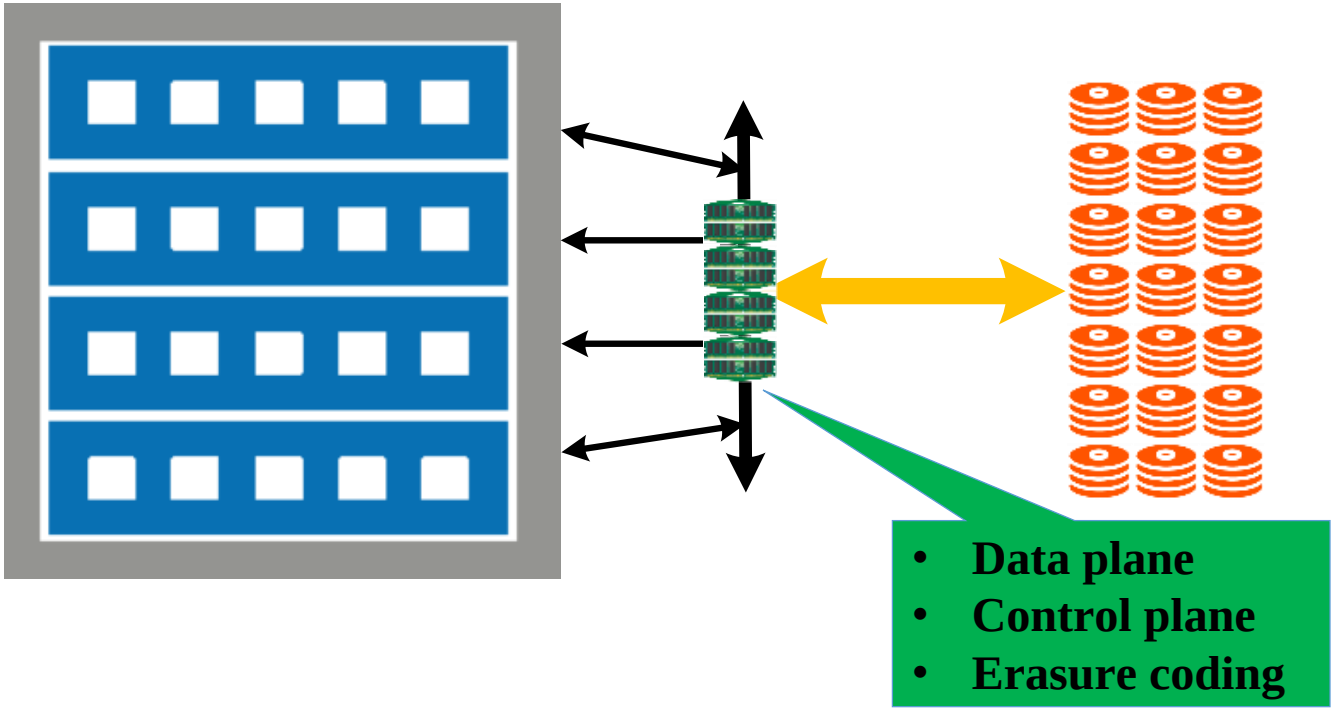


05 LPCC

HIERARCHICAL PERSISTENT CLIENT CACHING

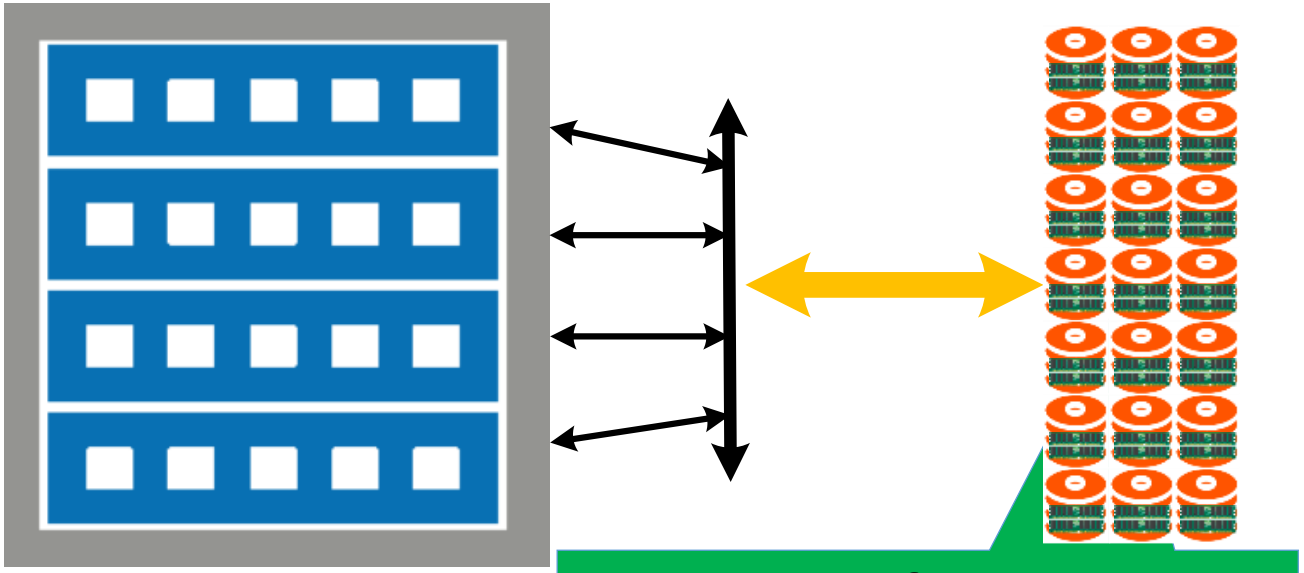
HSM Tier

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



HSM Tier

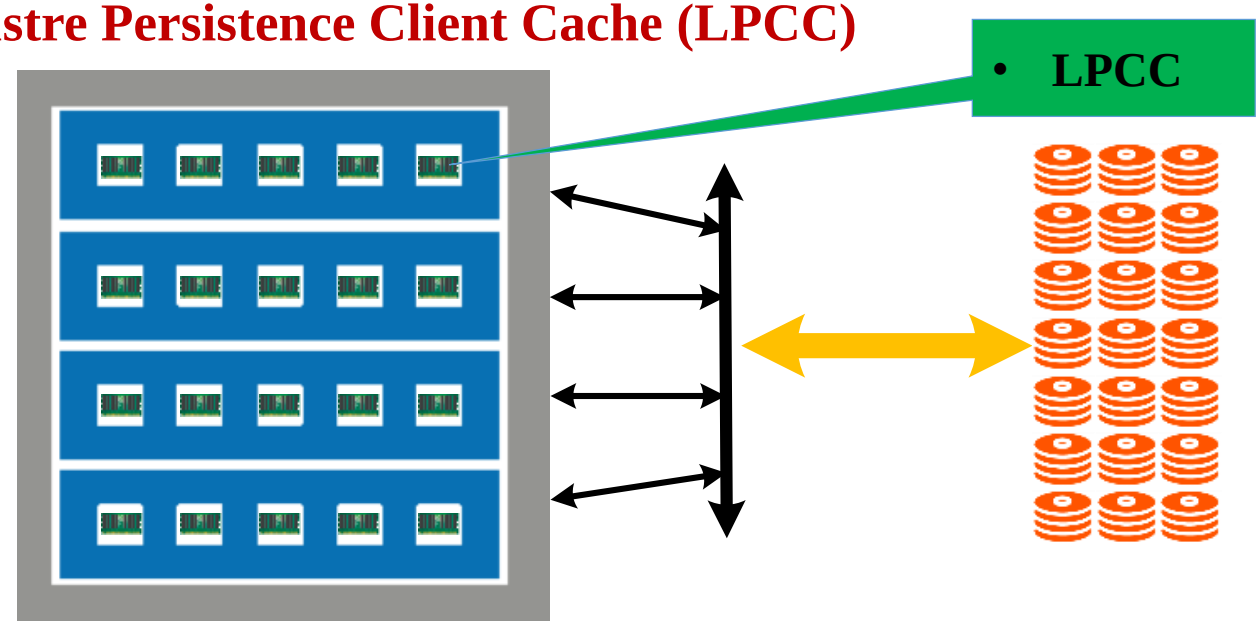
- Shared
  - DDN IME @ ICHEC
  - Cray Trinity @ LANL
- Server-side
  - Seagate Nytro NXD @ Sanger



- **Storage-side flash acceleration**
- **I/O histogram**
- **Performance statistics**
- **Dynamic flush**

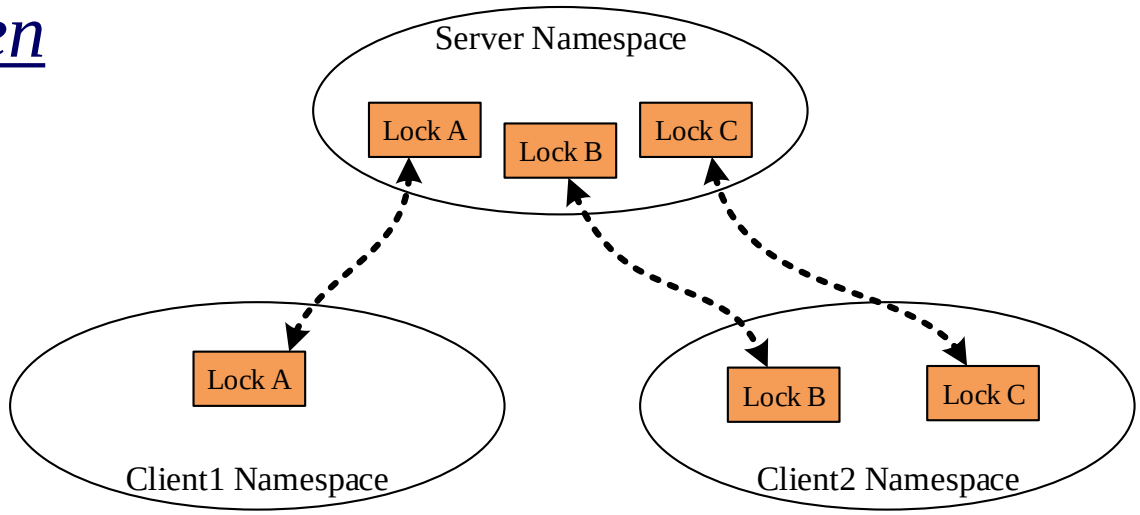
HSM Tier

- 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)**



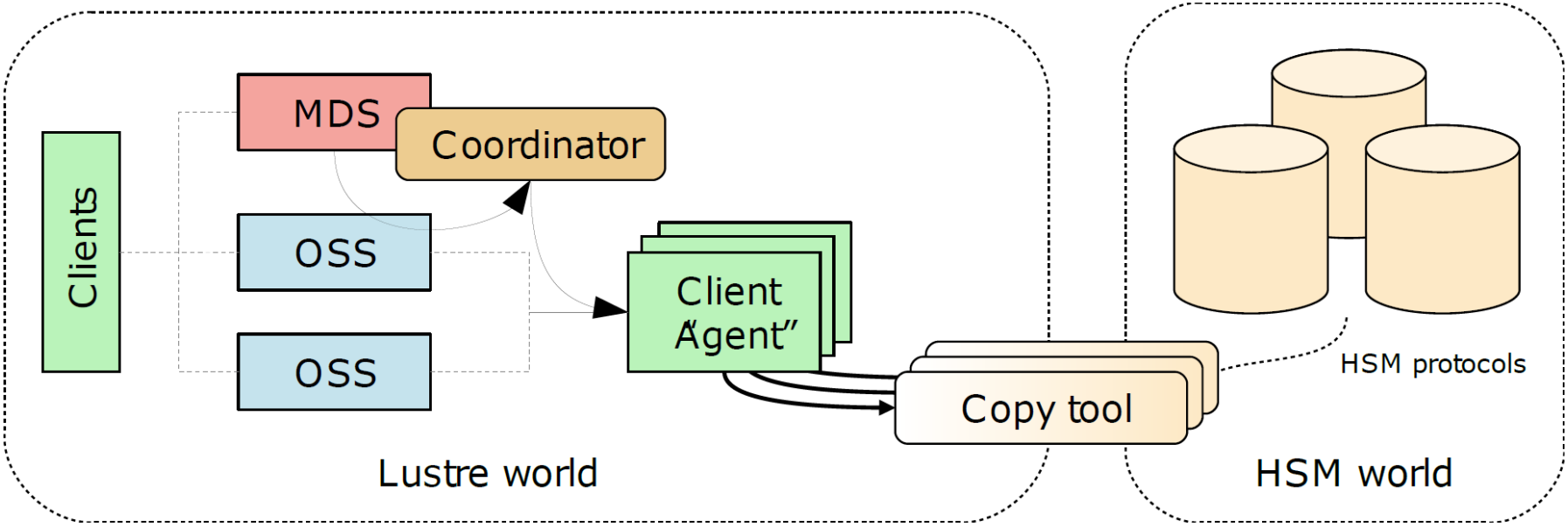
Lustre's DLM and Layout Lock

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



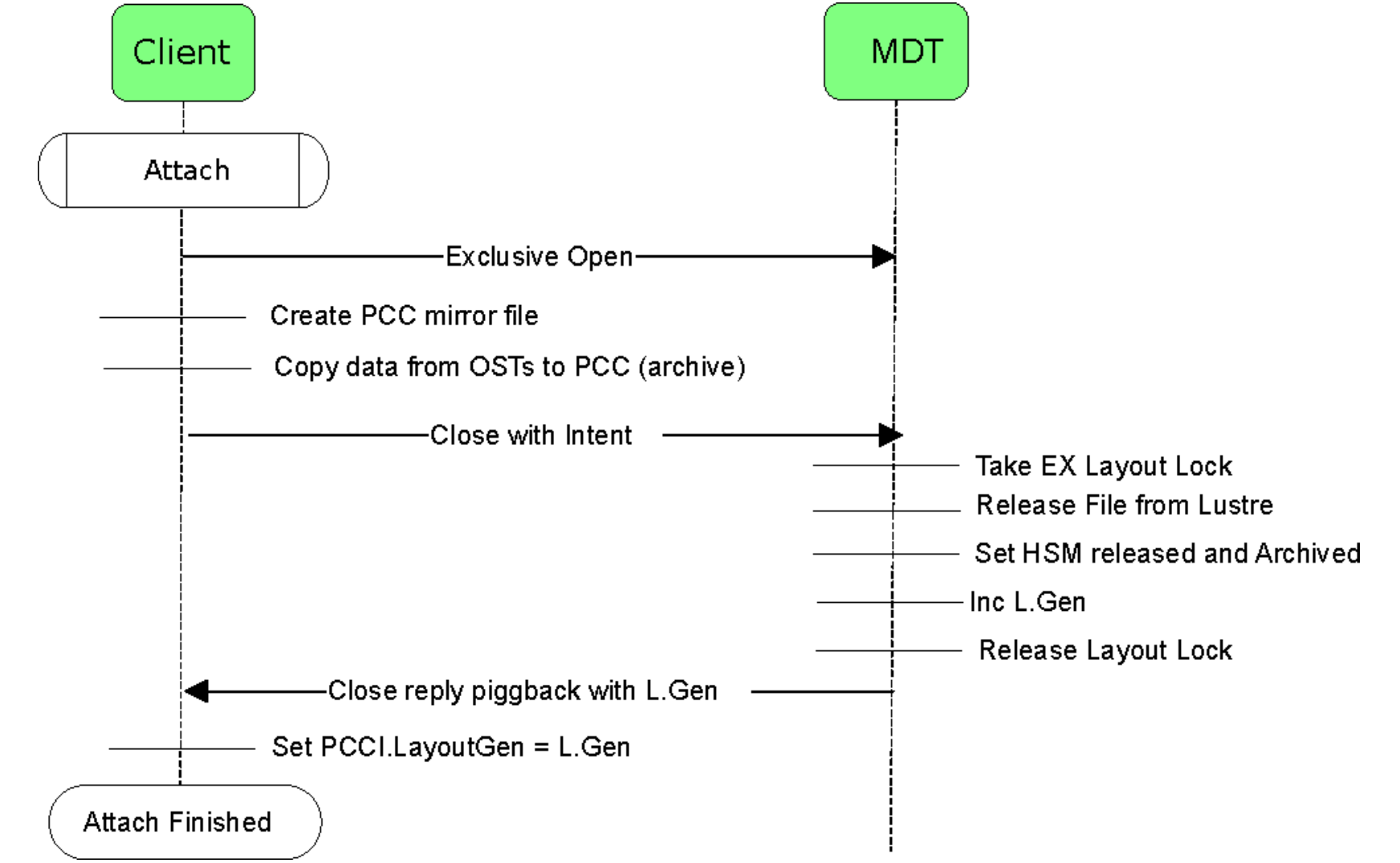
Lustre HSM

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

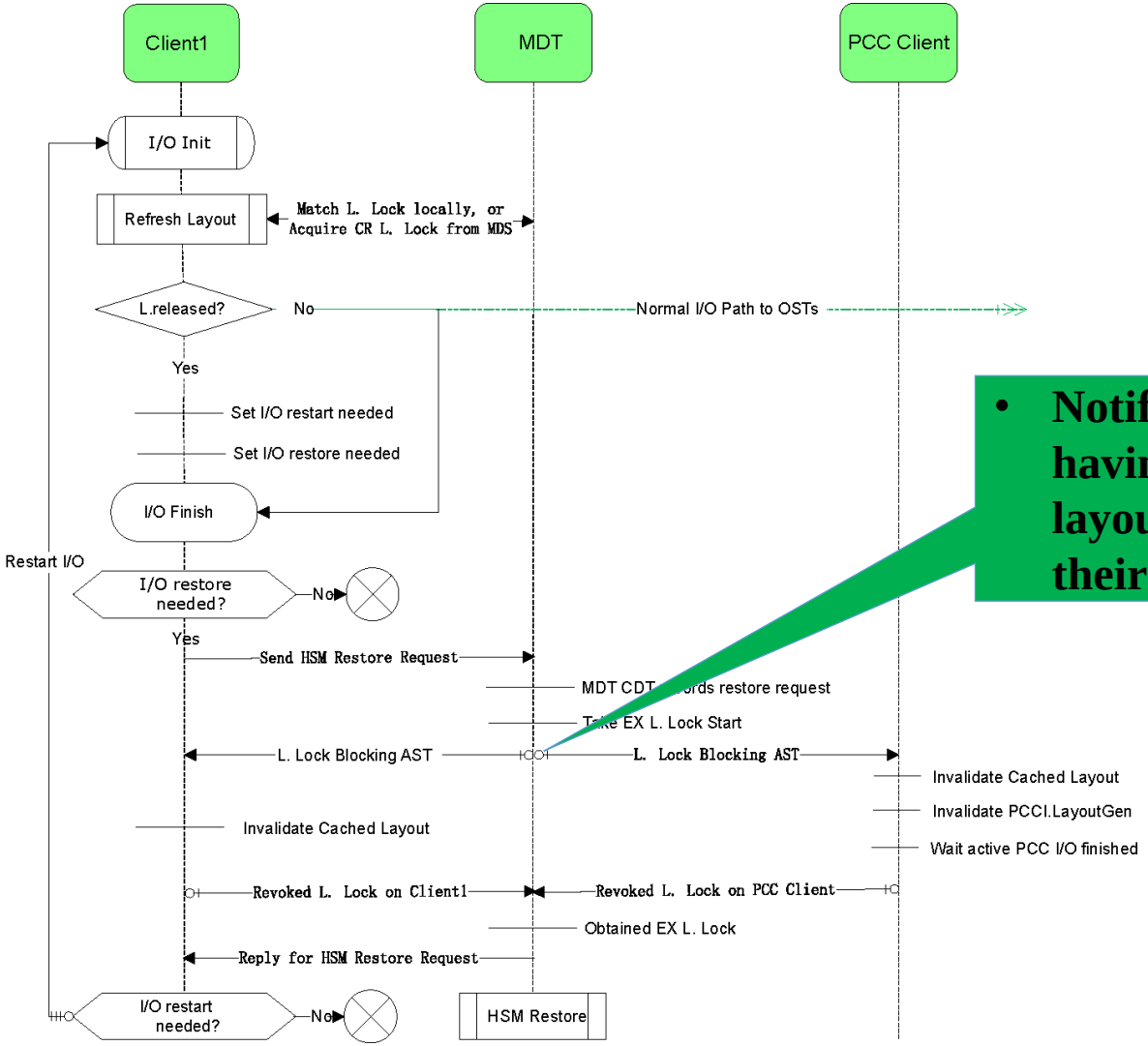


Refererence: Lustre manual. http://doc.lustre.org/lustre_manual.pdf

Lustre Read-Write PCC Caching (attach)

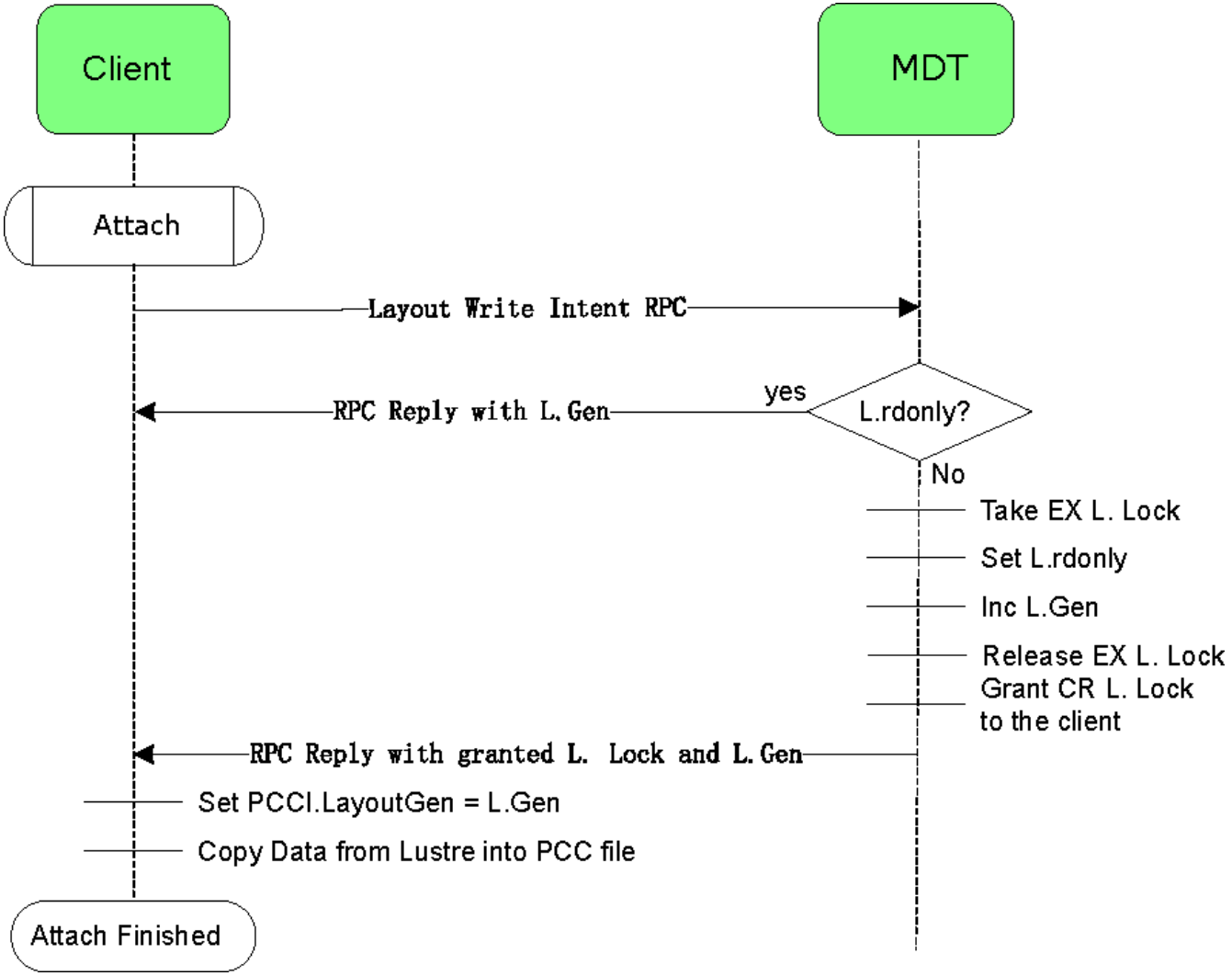


Lustre Read-Write PCC Caching (restore)

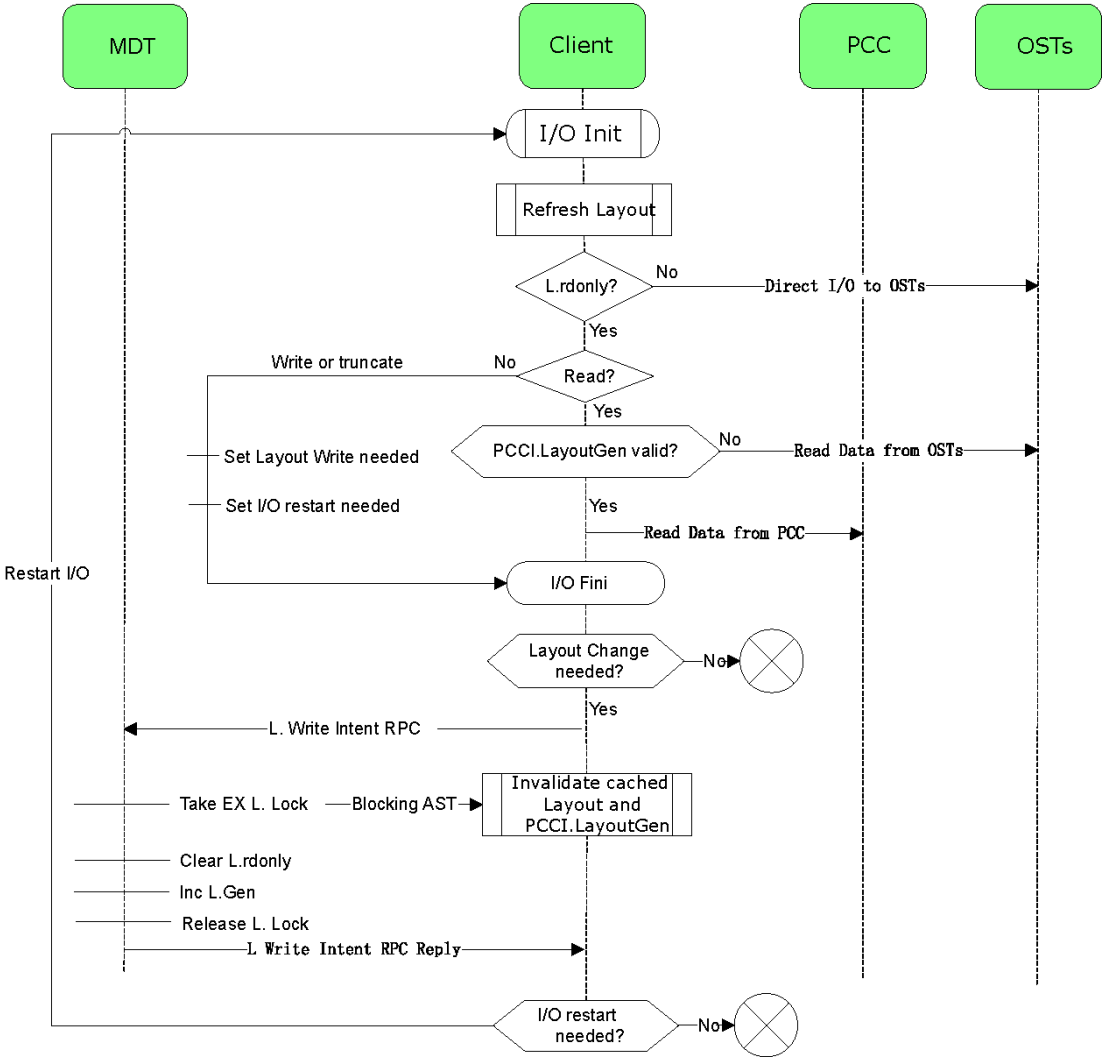


- **Notify all clients having cached the layout to invalidate their layouts**

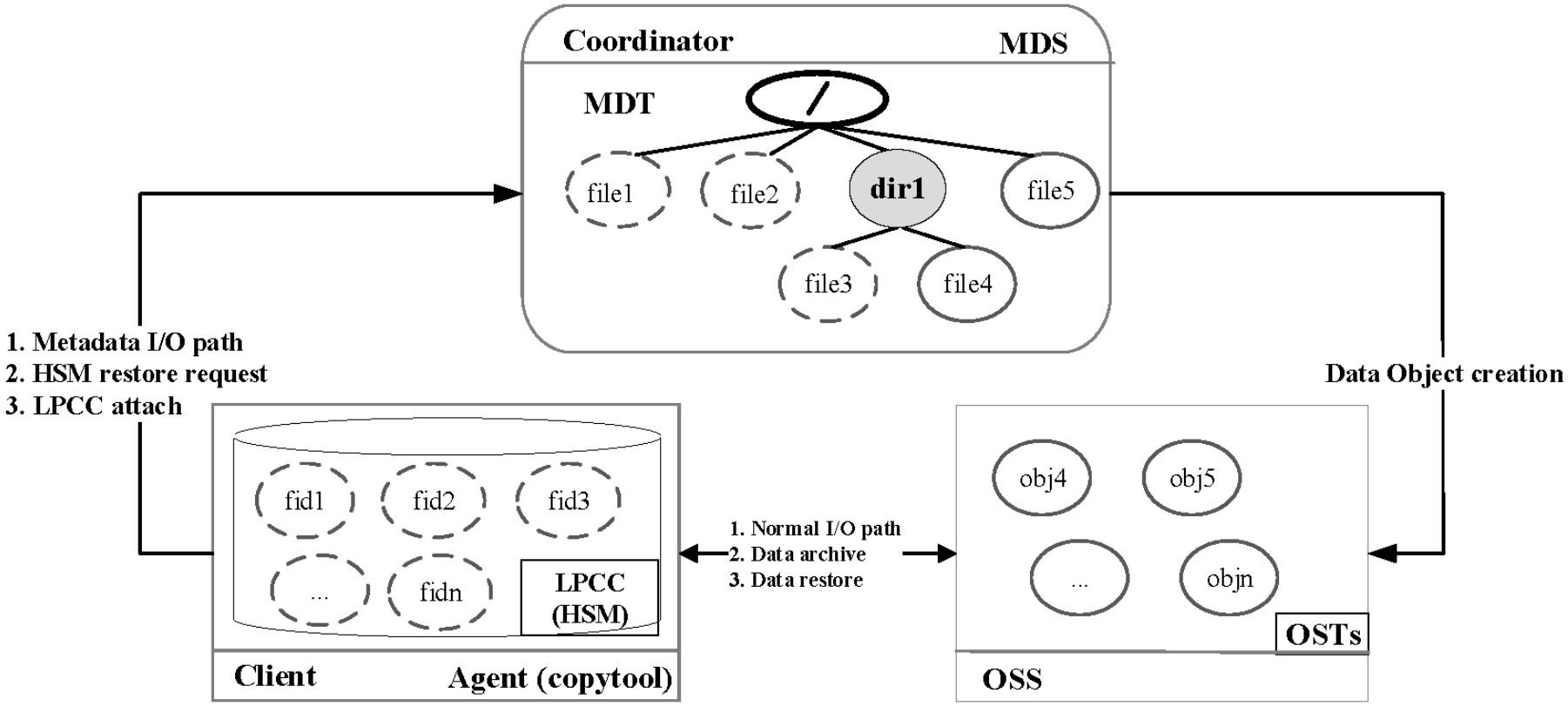
Lustre Read-only PCC Caching (attach)



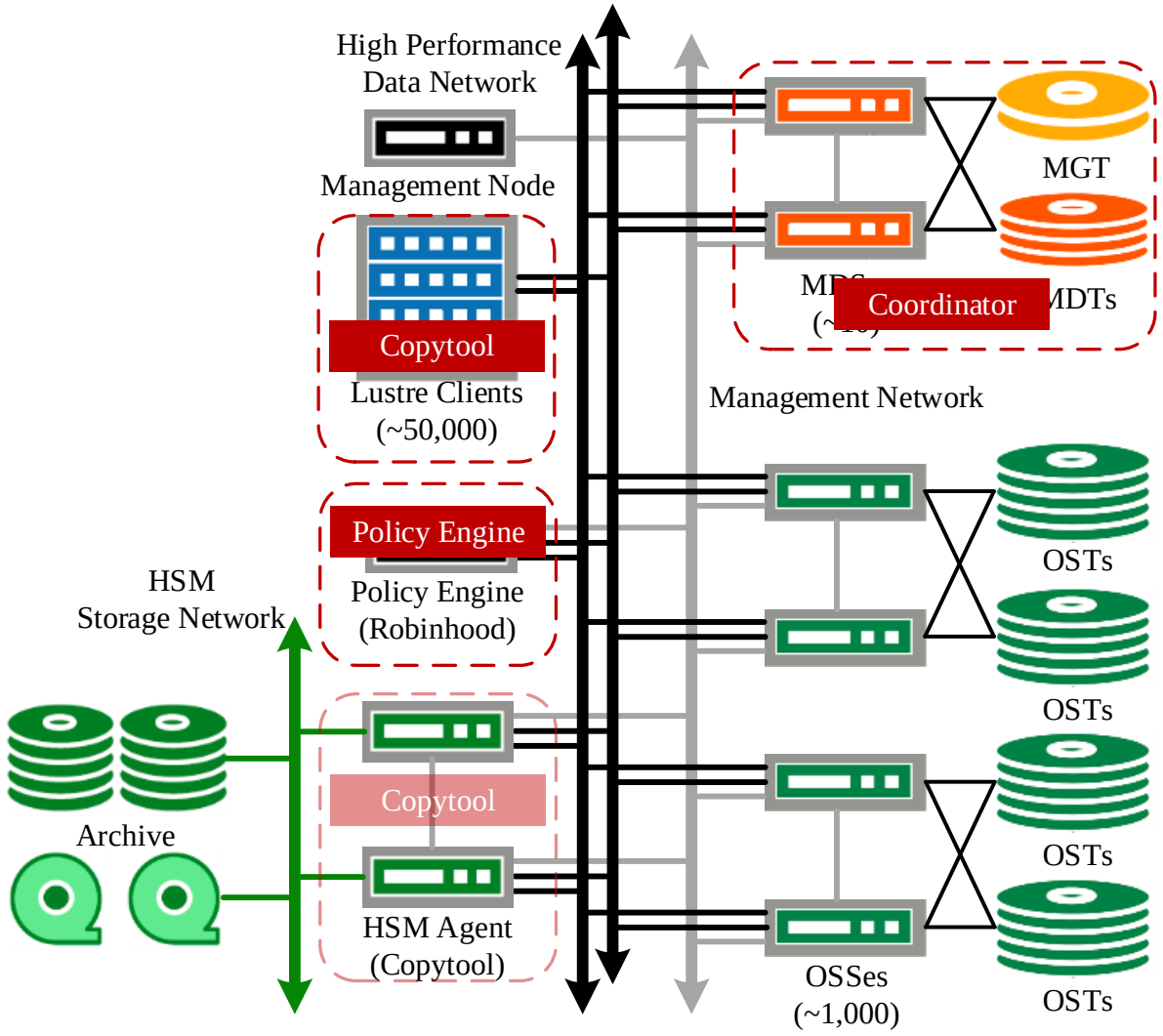
Lustre Read-only PCC Caching (I/O flow)



Overview of LPCC Architecture



Overview of LPCC Architecture



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

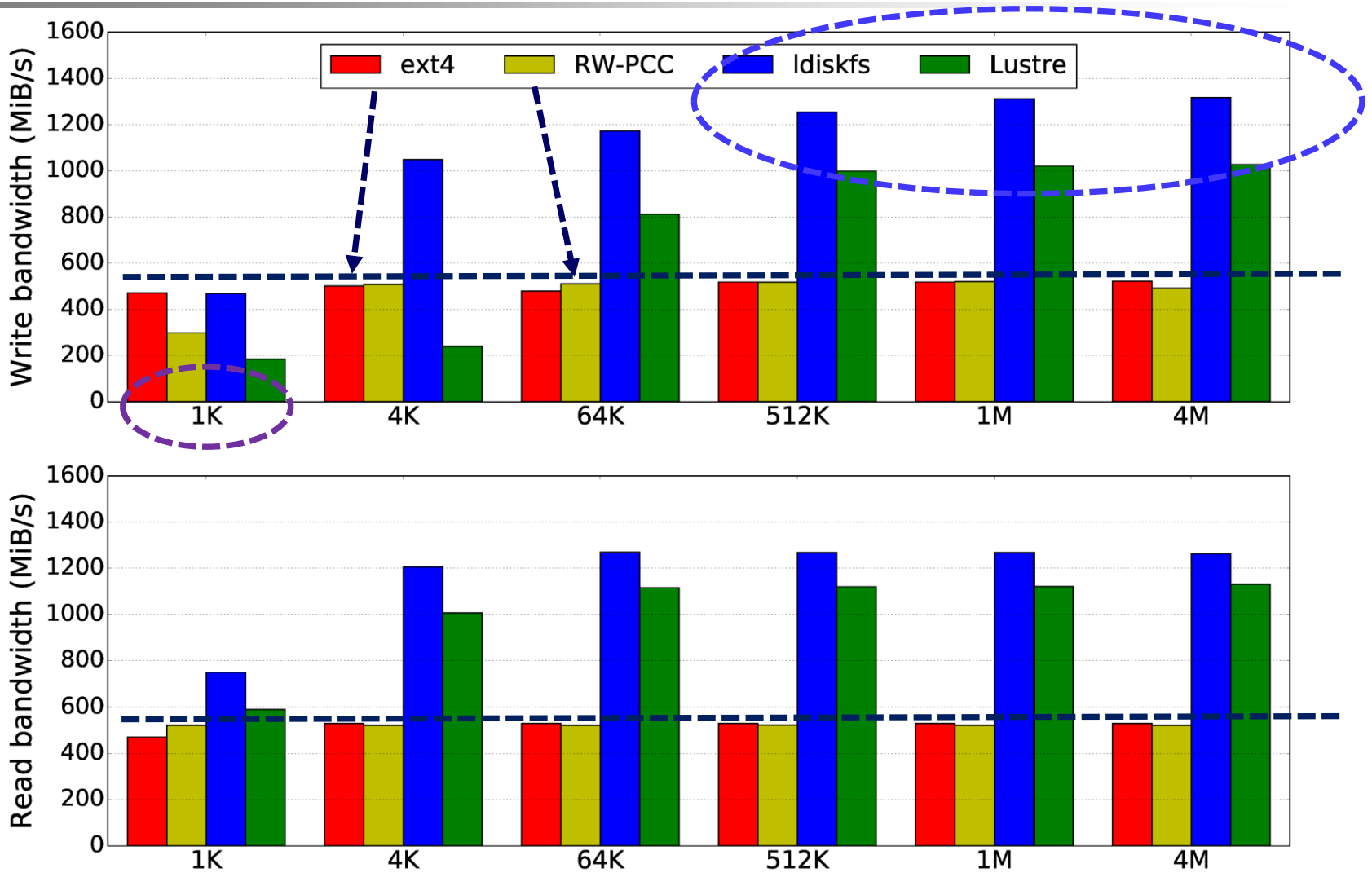
Evaluation Setup

- 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

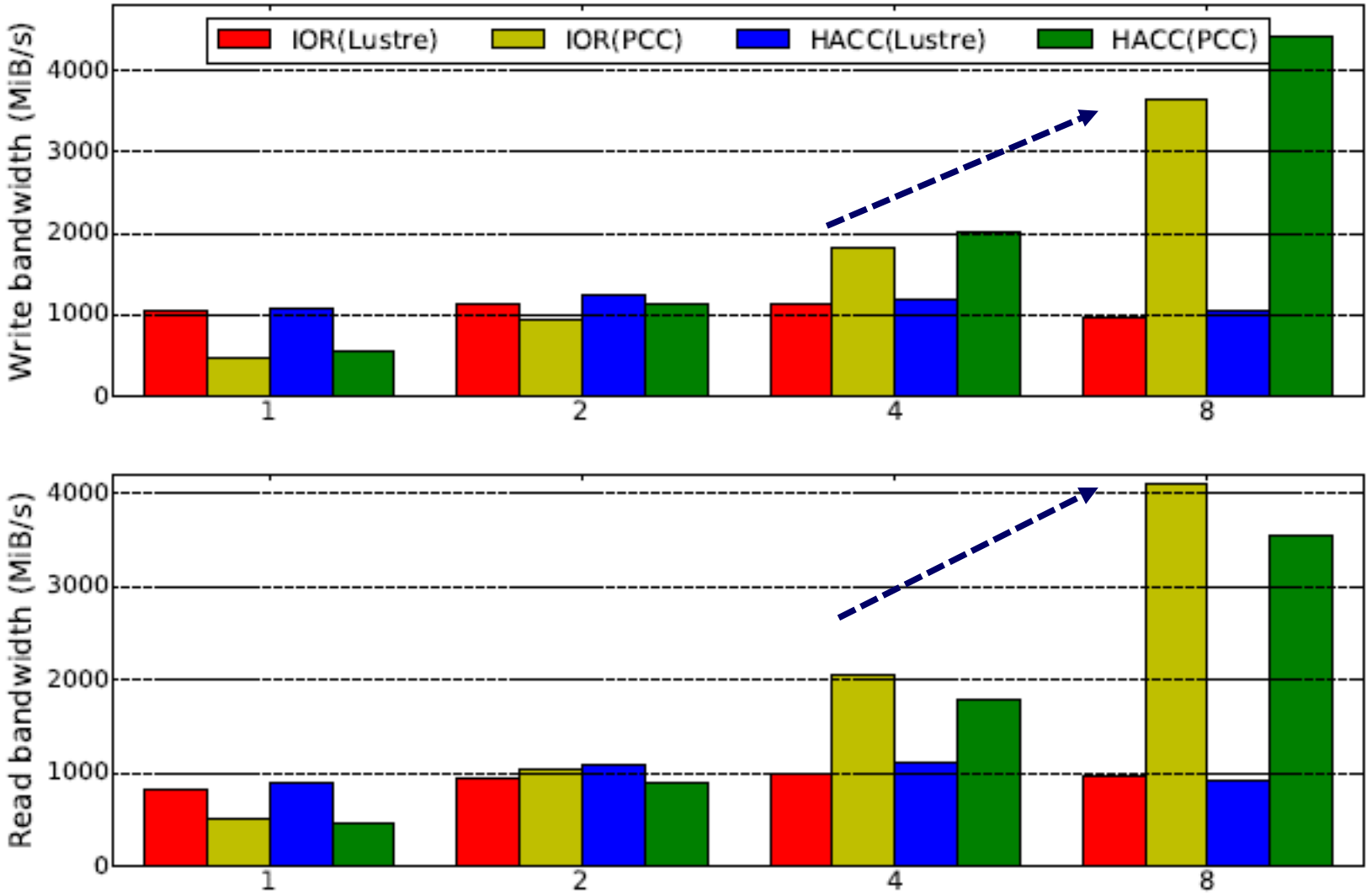
Benchmark Tools

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

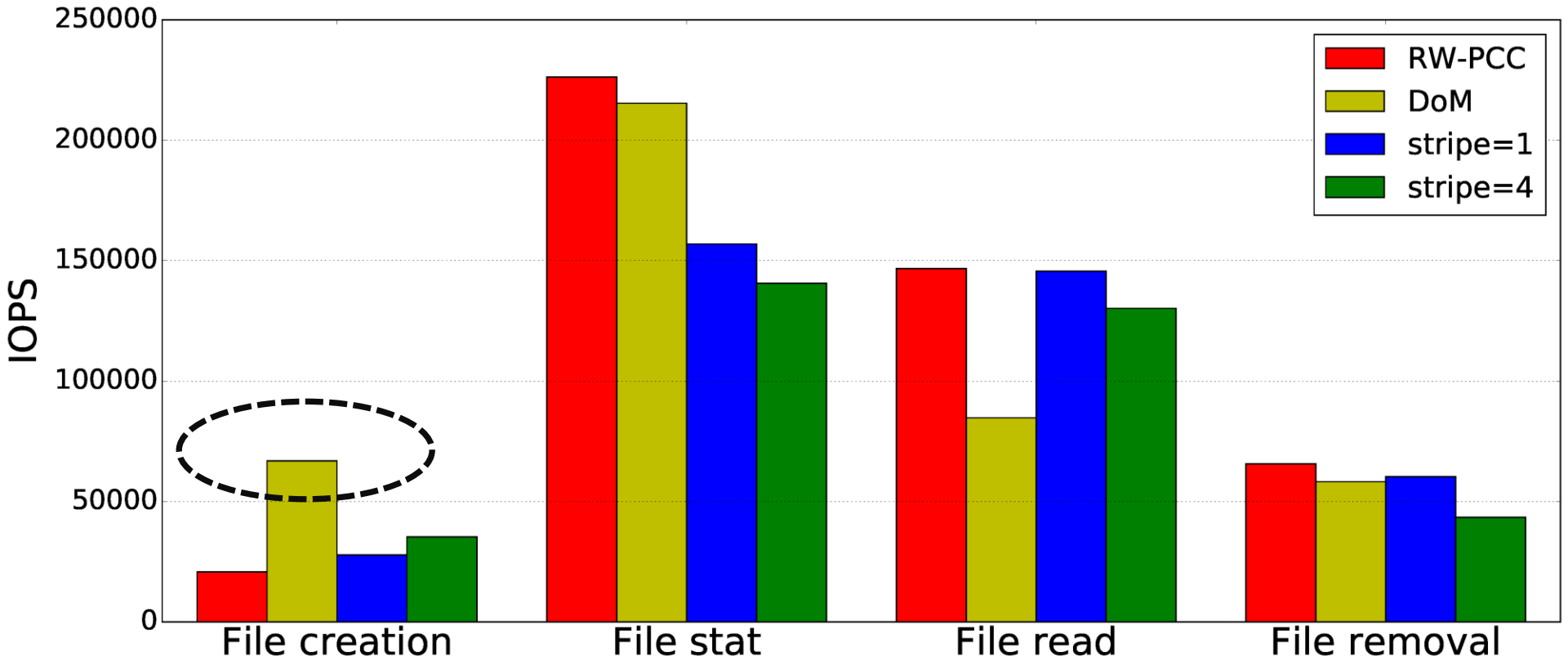
Single Thread Performance



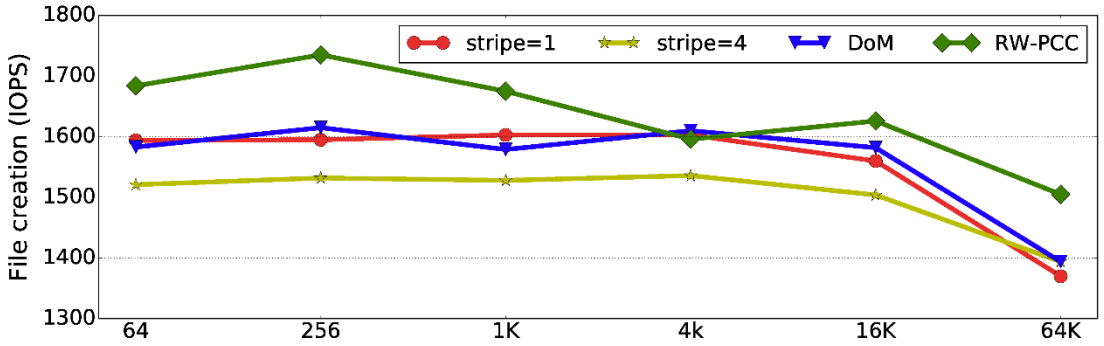
RW-PCC Scalability Evaluation

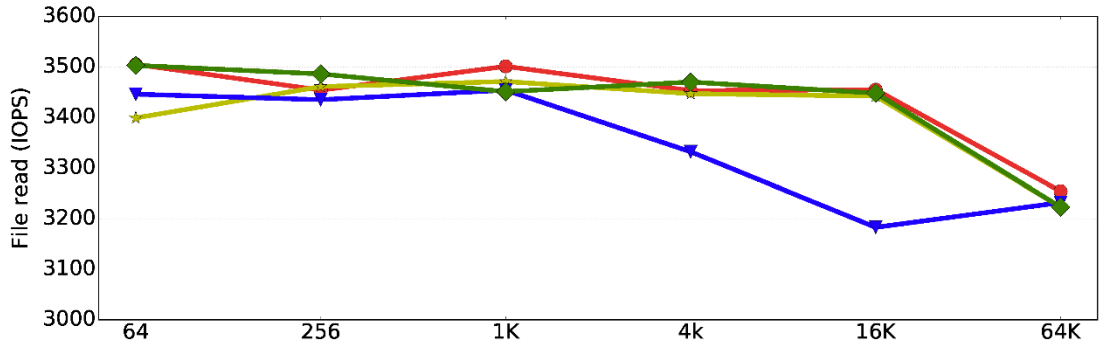


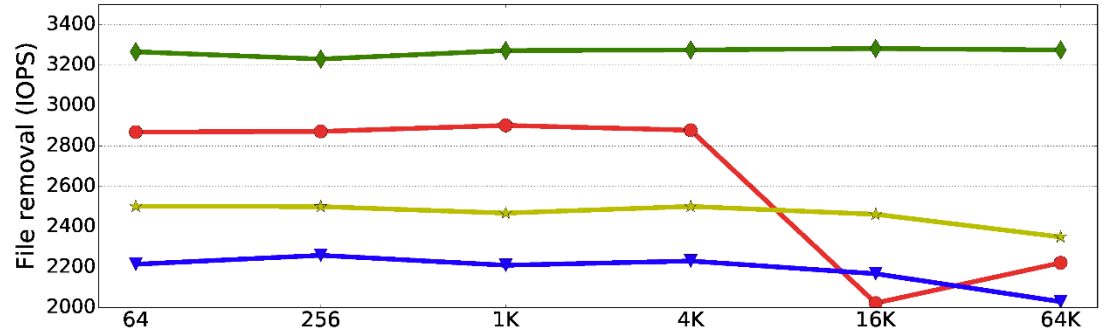
Metadata Performance



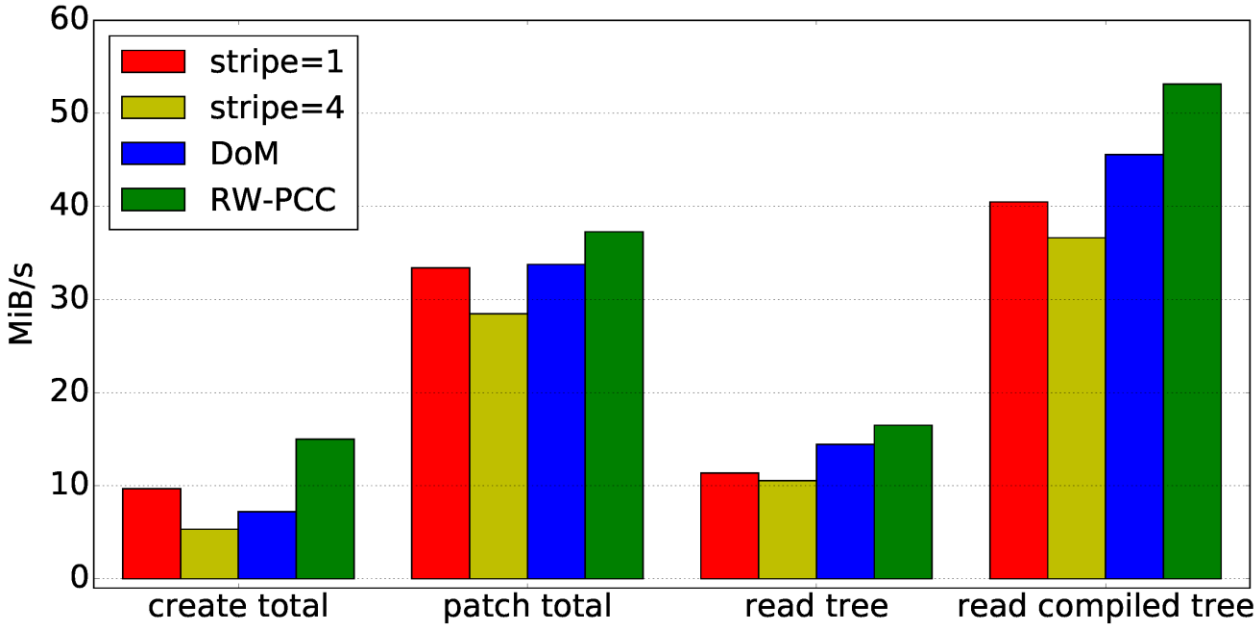
Small Files with Various Size

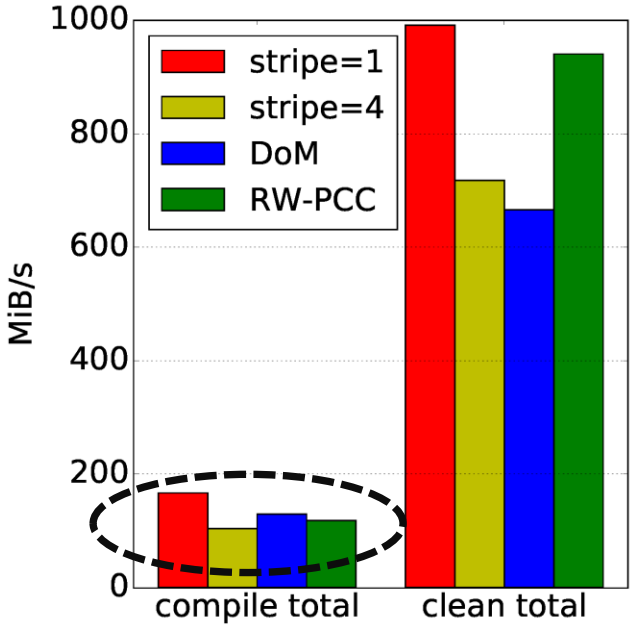






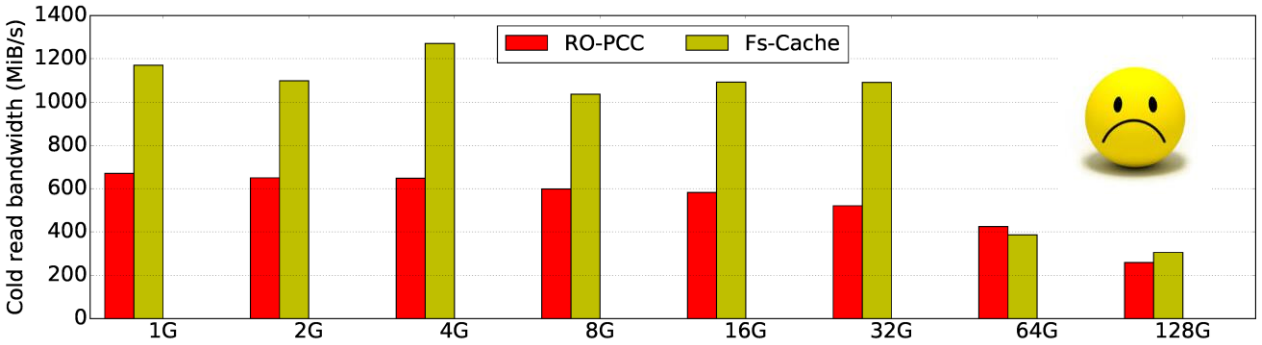
Small File for *Compilebench*



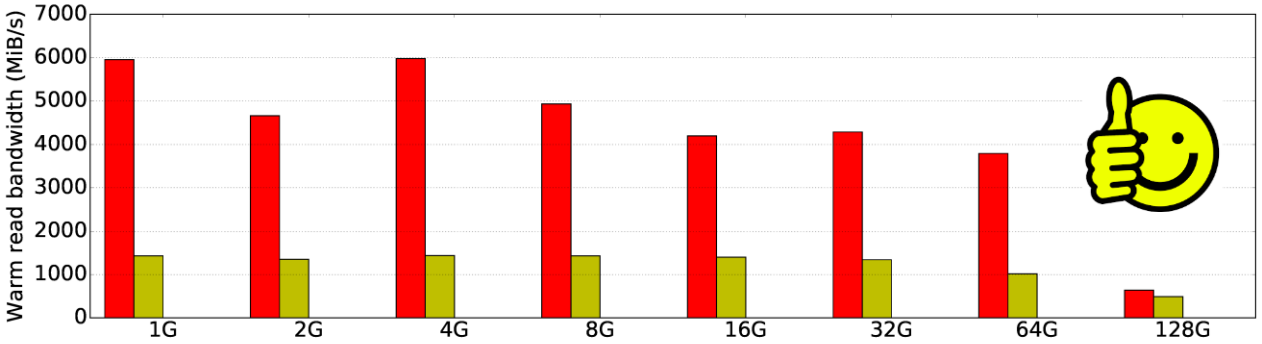


Read Performance

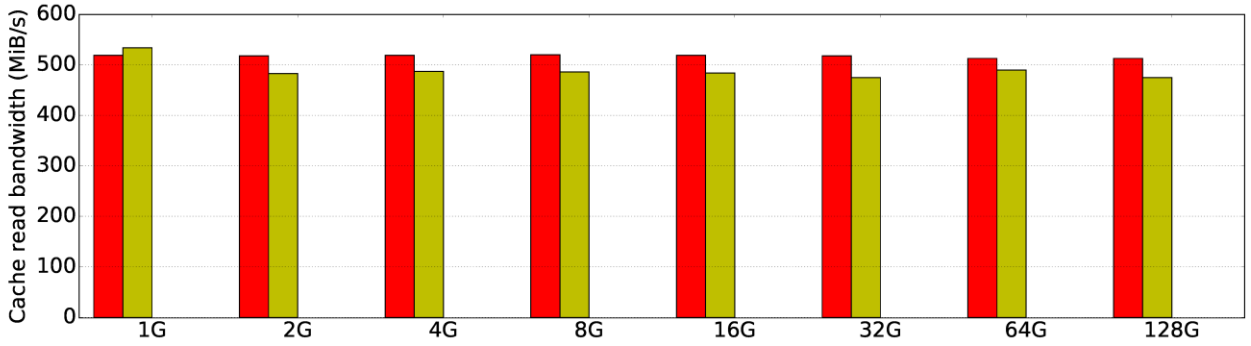
- **File data is read for the first time and loaded into cache**



- **By repeating the test immediately after the "Cold" one**



- **Directly from the persistent cache after cleaning all page caches**

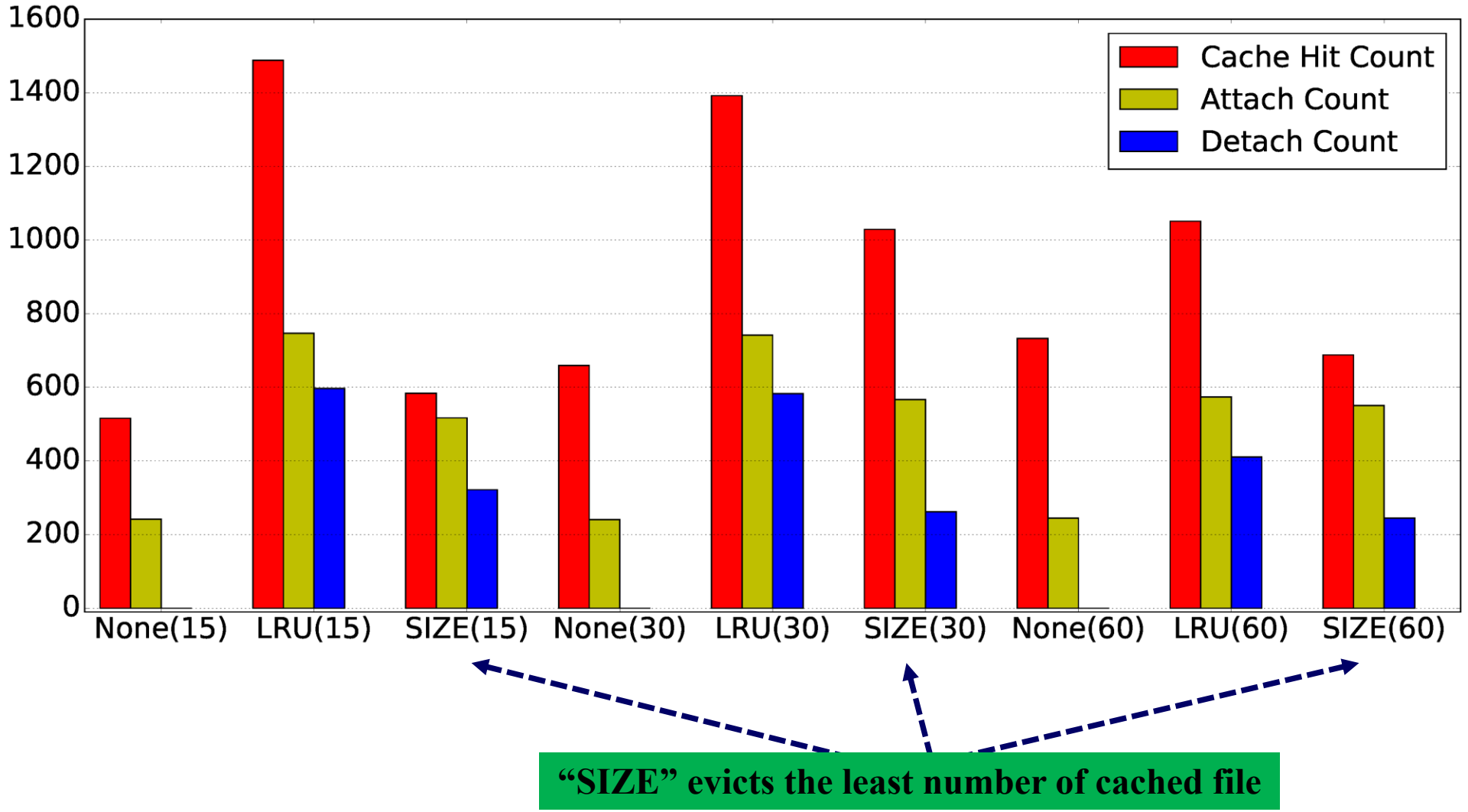


RO-PCC Scalability Evaluation

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

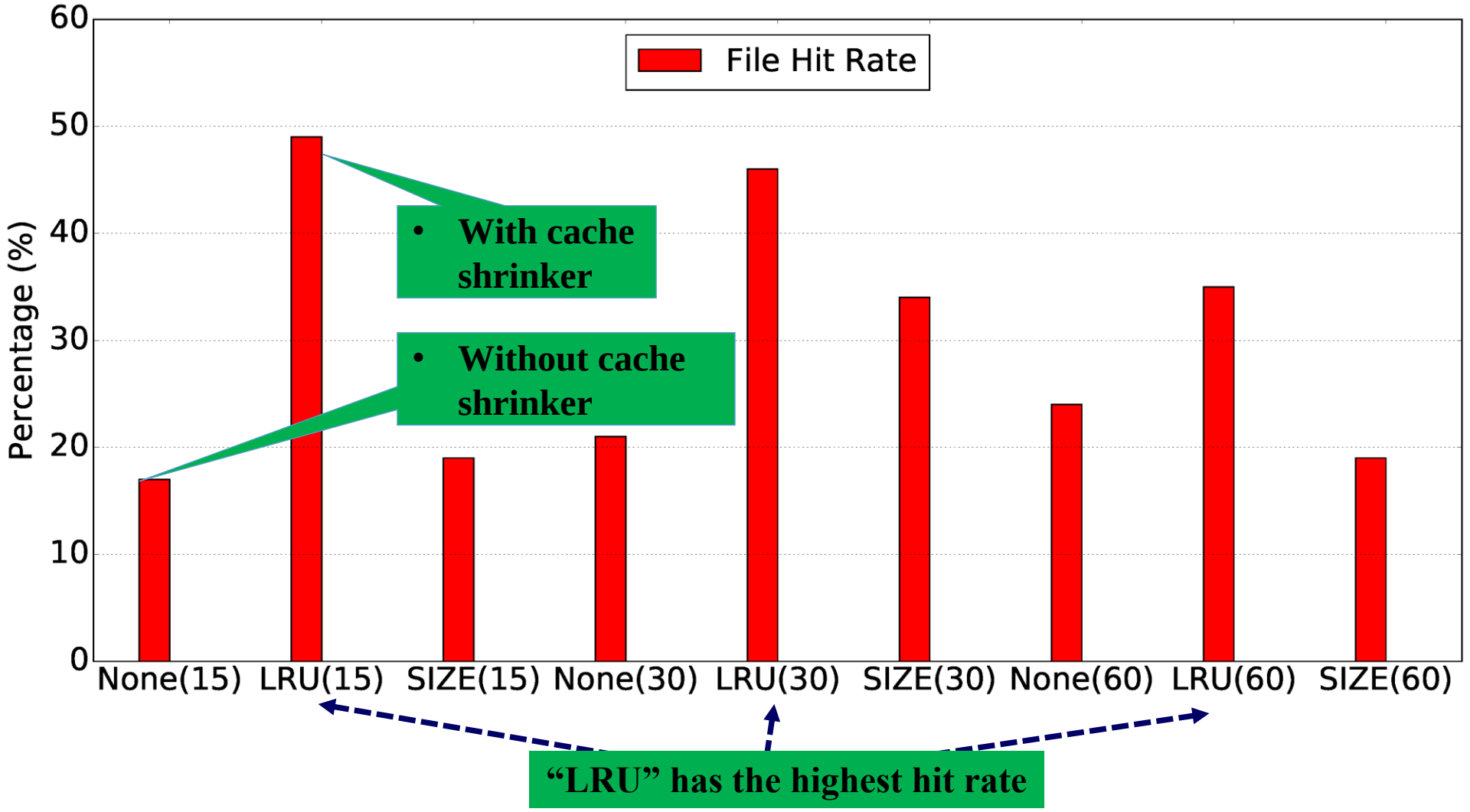
Client count	1	2	4	8
Cold (MiB/s)	478	688	718	1389
Warm (MiB/s)	4374	8746	17520	34943
Cache (MiB/s)	521	1042	2074	4029

Metrics Statistic



“SIZE” evicts the least number of cached file

File Hit Rate



06 CONCLUSION

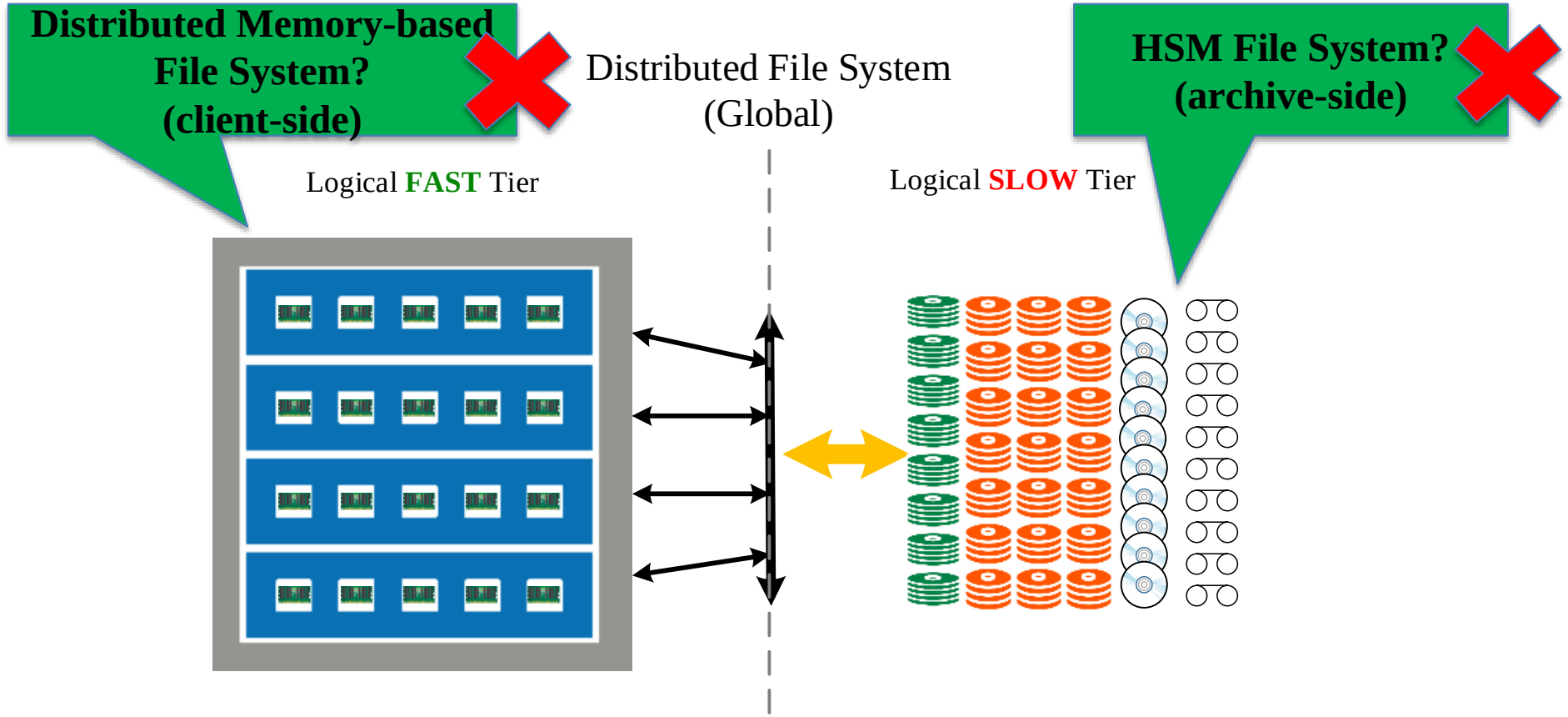
REMARKS AND ON-GOING WORK

Remarks and On-going Work (Summary)

- Lustre NRS Token Bucket Filter (TBF)
- QoS Planner
- Lustre Intelligent Management Engine (LIME)
- Lustre Persistent Client Caching (LPCC)

LPCC's Future (?)

- Logical two-tier (with physical multitier)
- NVRAM-based LPCC



Related libraries/tools

Related libraries/tools

google-gflags

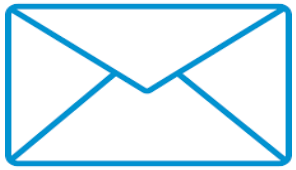
googletest
Google C++ Testing Framework

protobuf
Protocol Buffers

google-glog

Related libraries/tools

Thanks!



lfzeng@hust.edu.cn