23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing

# Taming Metadata-intensive HPC Jobs Through Dynamic, Application-agnostic QoS Control

**Ricardo Macedo**[1], Mariana Miranda[1], Yusuke Tanimura[2], Jason Haga[2], Amit Ruhela[3], Stephen L. Harrell[3], Richard Todd Evans[4], José Pereira[1], João Paulo[1]

[1] INESC TEC & University of Minho, [2] AIST, [3] UTAustin & TACC, [4] Intel

# Large-scale HPC systems

- Modern supercomputers are establishing a new era in HPC
  - Enable running applications at massive scale

- Traditional HPC applications are **compute-bound** and **write-dominated**

- However, **modern** HPC workloads are
  - **Data-intensive** and **read-dominated**
  - Many applications spend 15%-40% of their execution time performing storage I/O
  - Generate massive **bursts of metadata** operations

- Several HPC centers have already observed a **surge of metadata** operations in their clusters, and expect this to become **more severe** over time

# Metadata operations in a production cluster

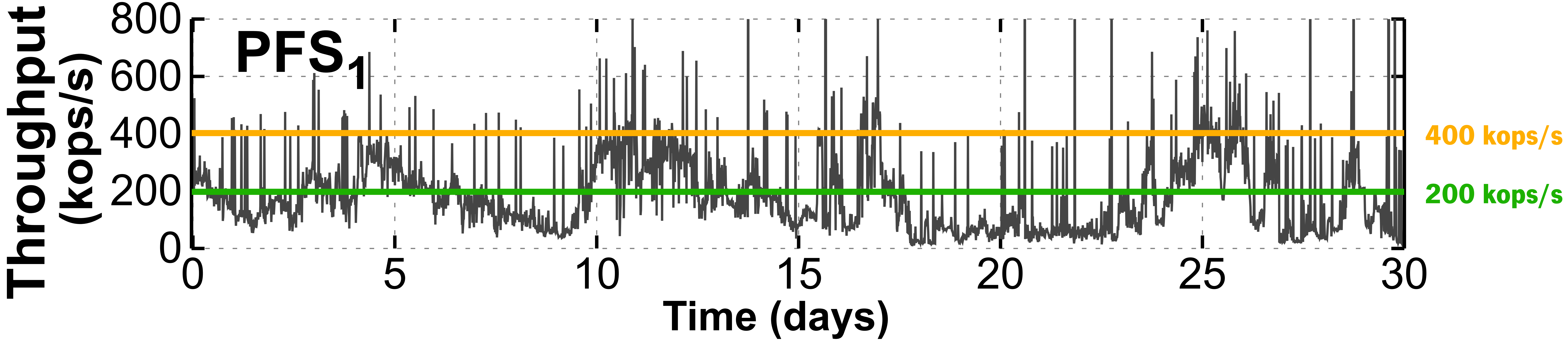
- Analysis of the logs of a **production** Lustre file system from the **ABCI** supercomputer
  - DDN ExaScaler Lustre composed of 2 MDSs, 6 MDTs, and 36 OSTs with 9.5 PiB of capacity

- We monitored I/O activity of the most frequent operations at MDSs/MDTs
  - `open`, `close`, `getattr`, `setattr`, `rename`, `mkdir`, `mknod`, `rmdir`, `statfs`, `sync`, and `unlink`
  - We also monitored `read` and `write` bandwidth observed at OSTs

- Logs report 1-minute samples over a **30-days** observation period

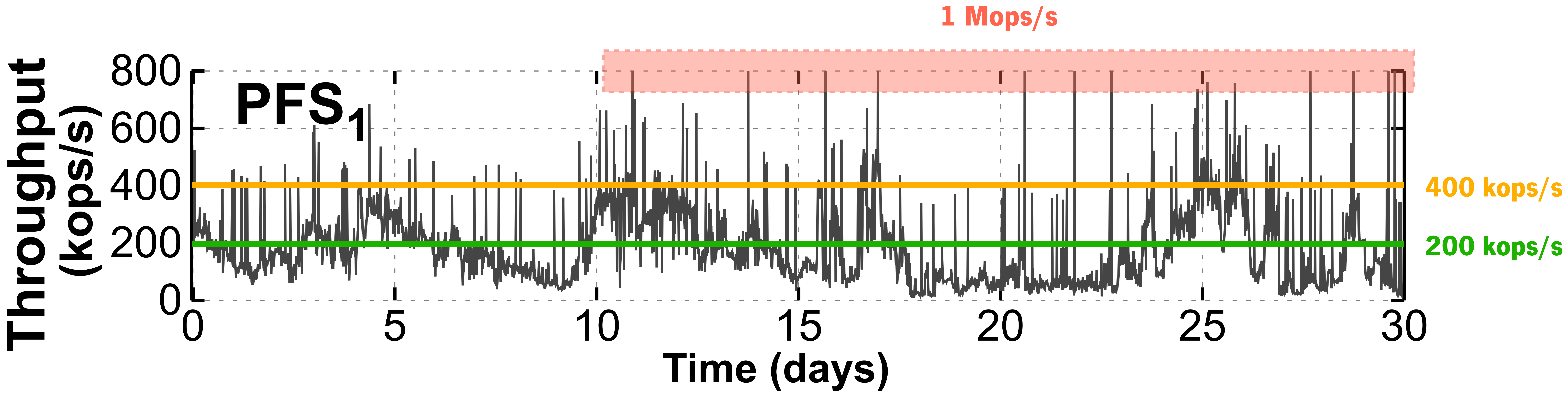
# Metadata operations in a production cluster

## Overall metadata load



# Metadata operations in a production cluster

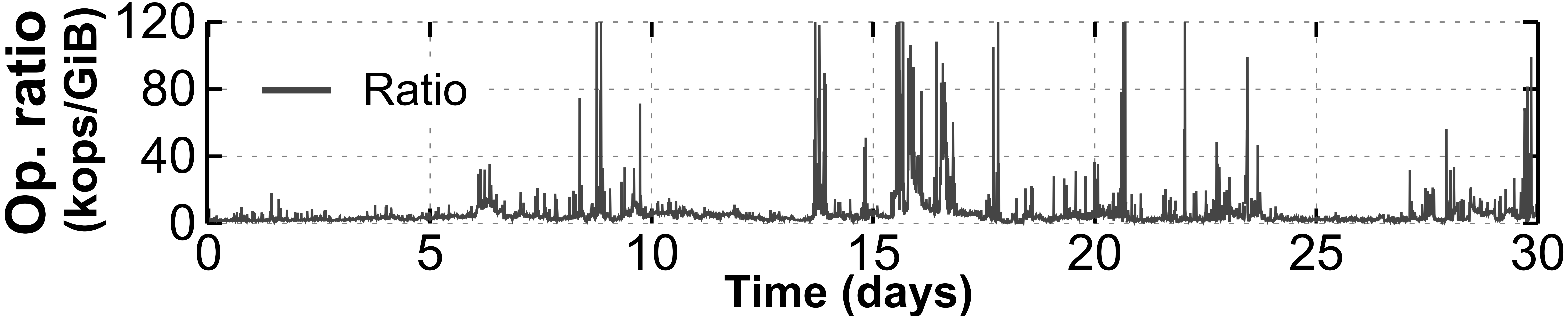
## Overall metadata load




Modern workloads generate **massive** amounts of **metadata** operations with **high throughput rates** and **bursts** that **peak at 1 Mops/s**.

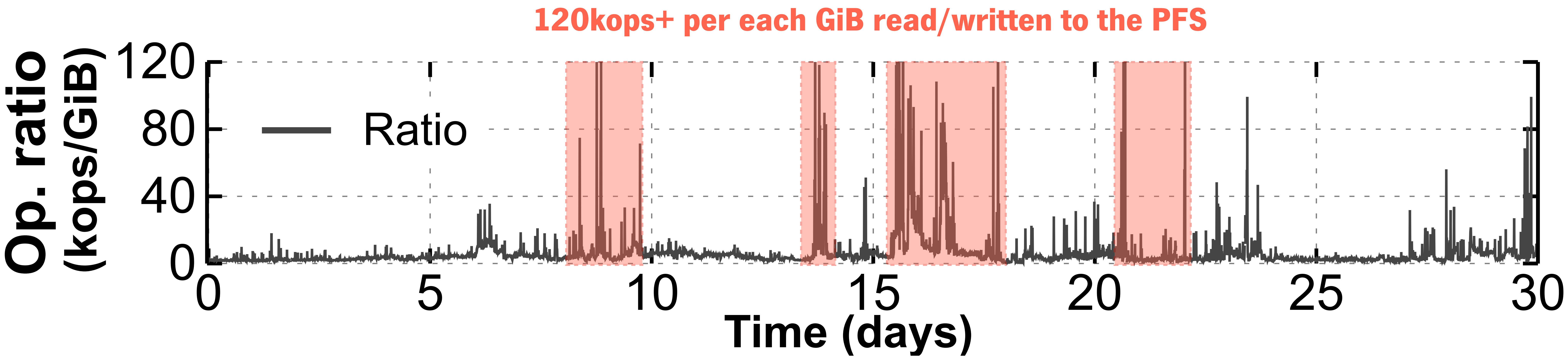
# Metadata operations in a production cluster

## Ratio of metadata operations to I/O bandwidth



# Metadata operations in a production cluster

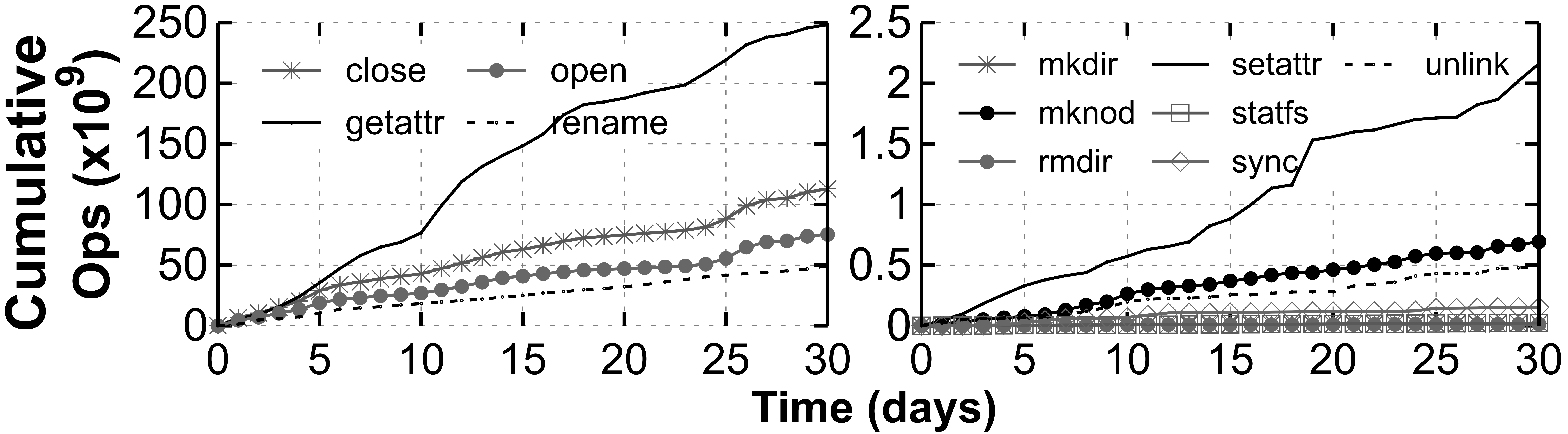
## Ratio of metadata operations to I/O bandwidth




Under several periods, the amount of **metadata** operations **far exceed** the GiBs of **data** read/written from/to the PFS. This means that there is **not a strict dependency** between both operation types.

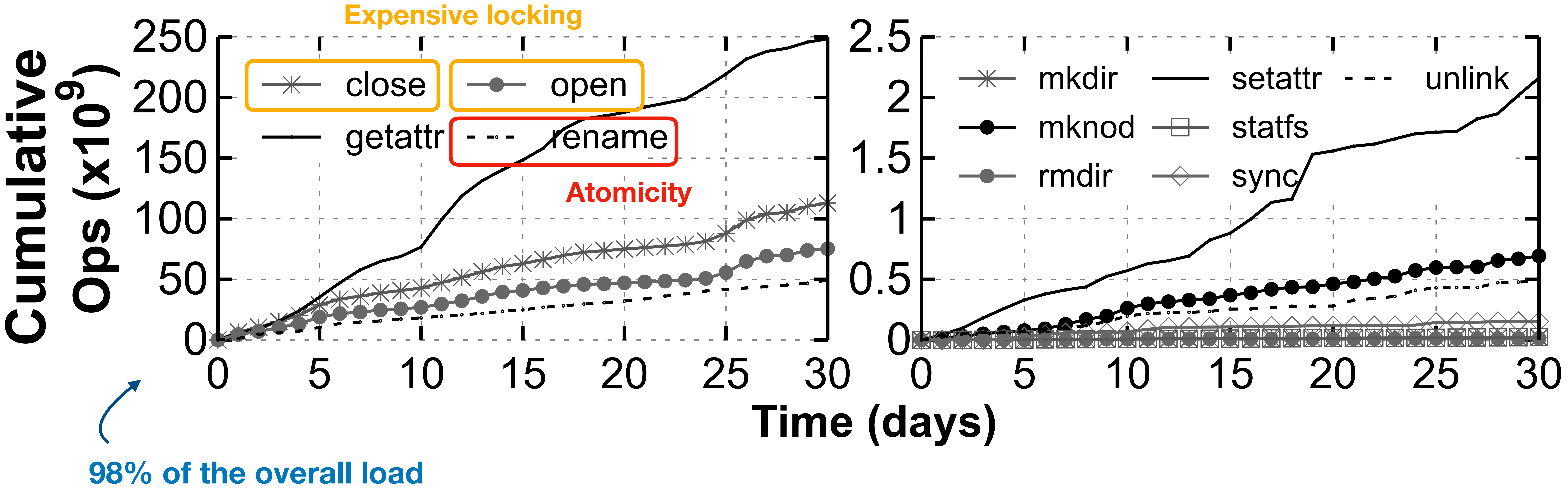
# Metadata operations in a production cluster

## Type and frequency of metadata operations



# Metadata operations in a production cluster

## Type and frequency of metadata operations




**Not all metadata** operations entail the same **cost** and **I/O pressure** over the shared resources, and thus, should be controlled with **fine-granularity**.

# Can HPC storage systems sustain these workloads?

# The metadata challenge
## Parallel file systems

- Lustre-like PFS provide a **centralized metadata** management service

- Multiple **concurrent** jobs compete for shared I/O resources
  - Severe **I/O contention**
  - Overall **performance degradation**

- A single user's I/O operations can saturate Lustre metadata resources

- Existing solutions are suboptimal

# The metadata challenge

## Existing approaches

- **Manual intervention**
  - System administrators stop jobs with aggressive I/O behavior
  - Slow and reactive approach
- **Intrusiveness to I/O layers**
  - Many solutions that ensure QoS control over I/O workflows are tightly coupled to core layers of the HPC I/O stack
  - Profound system refactoring and low portability
- **Partial visibility and I/O control**
  - Few solutions enable QoS control from the application-side, but are agnostic of remainder jobs
  - Isolated and uncoordinated control of metadata resources

# PADLL

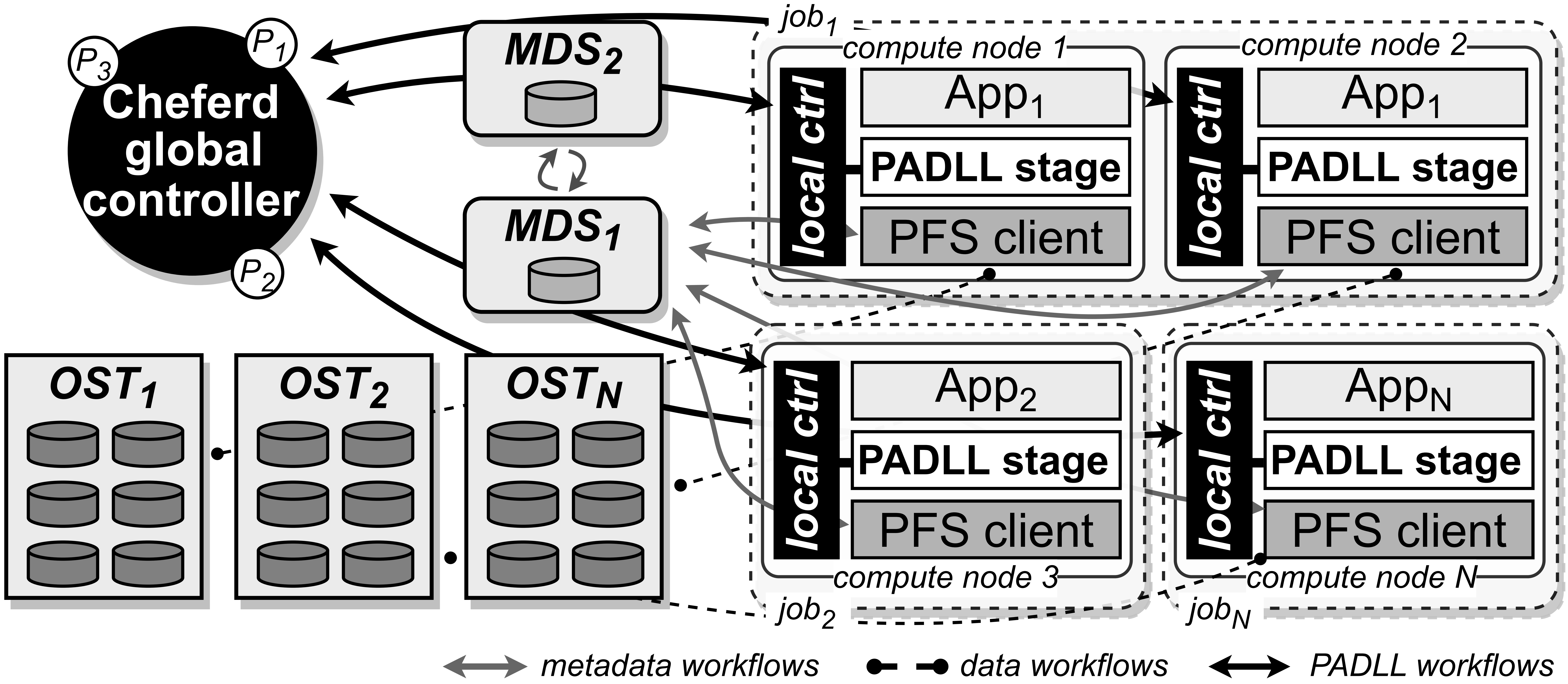
- **Storage middleware** that enables system administrators to **proactively** and **holistically** ensure **QoS over metadata** workflows

- Adopts ideas from the **Software-Defined Storage** paradigm
  - **Data plane:** application and PFS-agnostic middleware that provides the building blocks for **rate limiting** I/O requests destined towards the shared storage
  - **Control plane:** global coordinator that manages the data plane to ensure storage QoS policies are met at all times

- PADLL **does not require changing** core layers of the HPC I/O stack

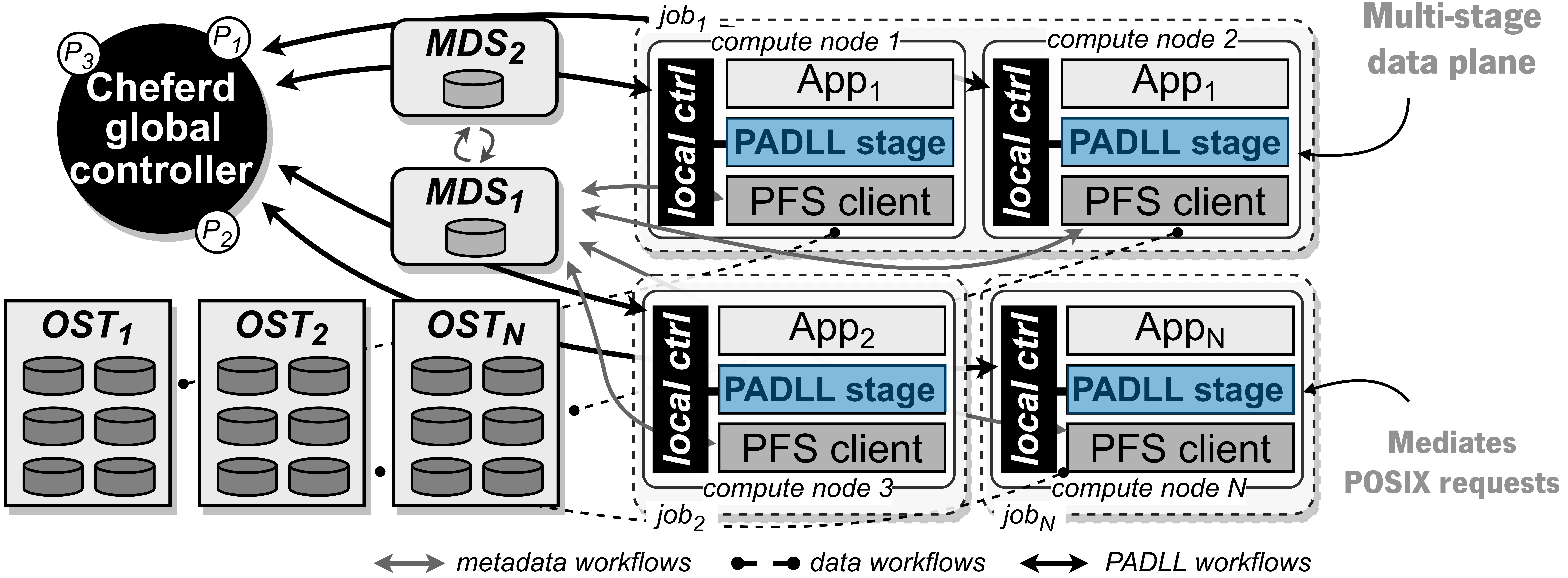
# PADLL

## High-level architecture



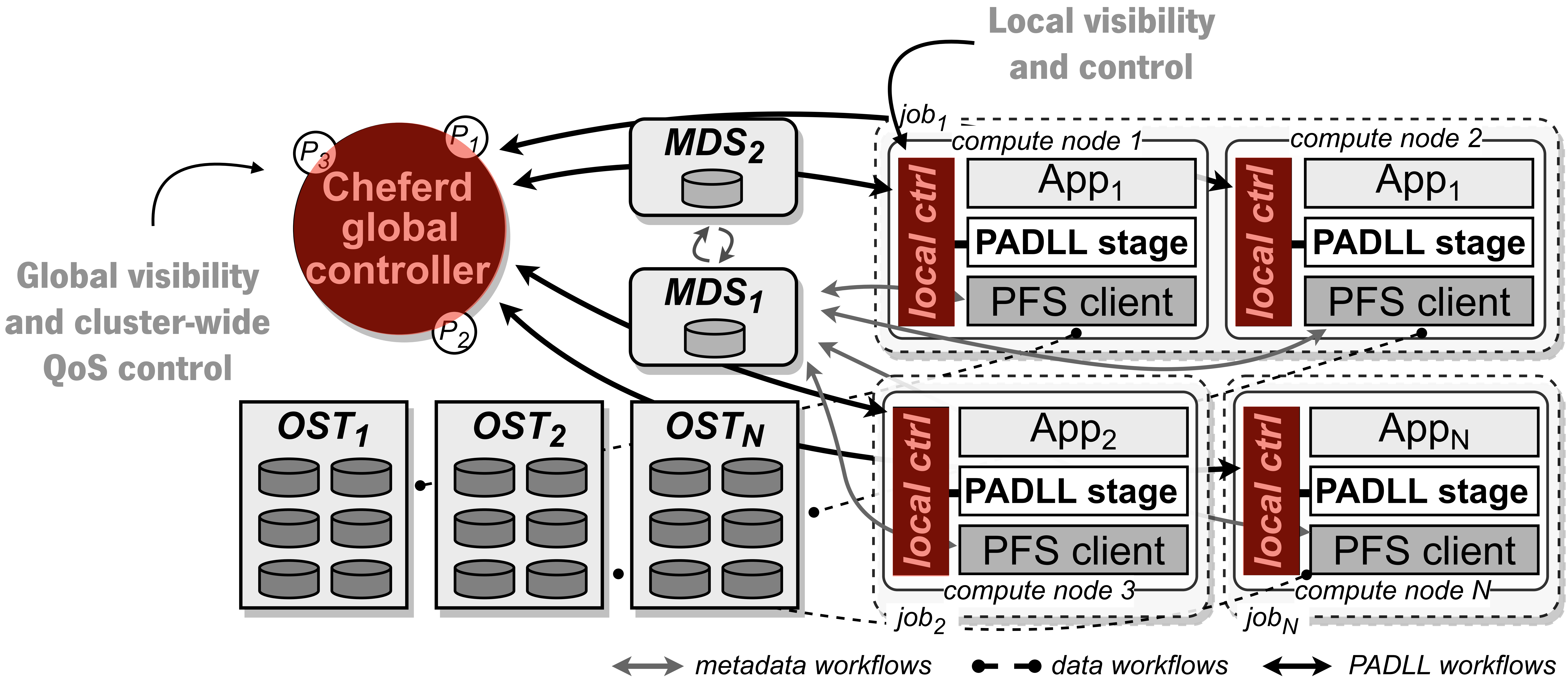
# PADLL

## High-level architecture



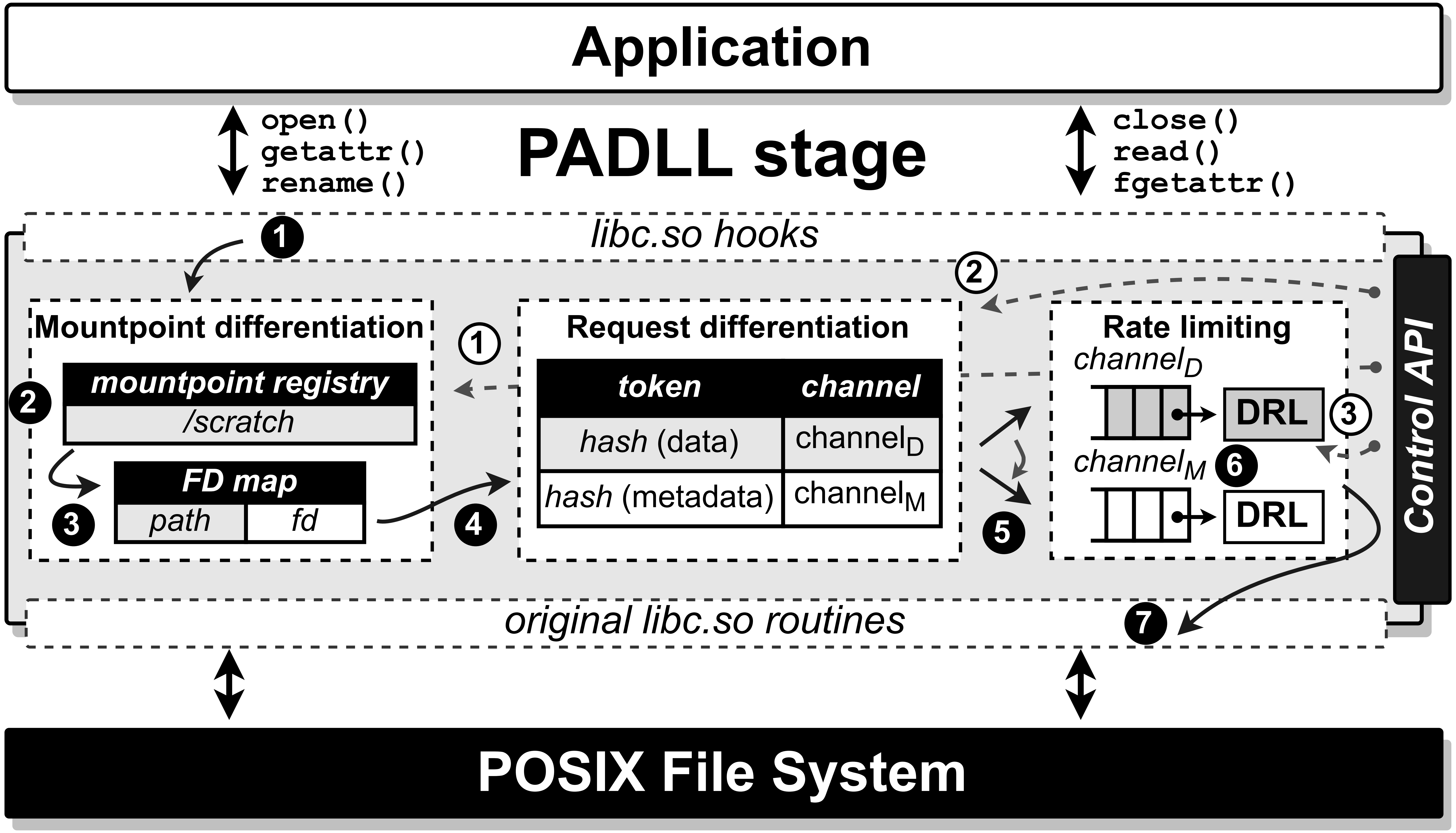
# PADLL

## High-level architecture



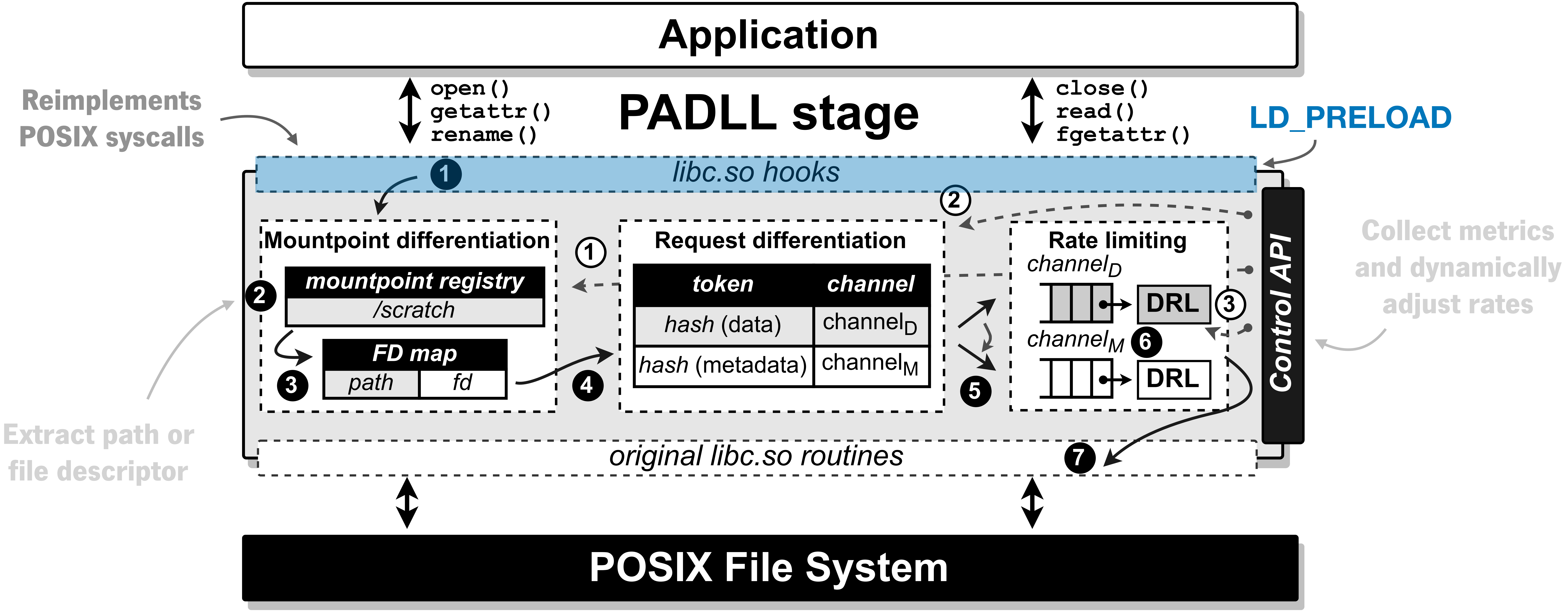
# PADLL

## Data plane



# PADLL

## Data plane: intercepting POSIX calls



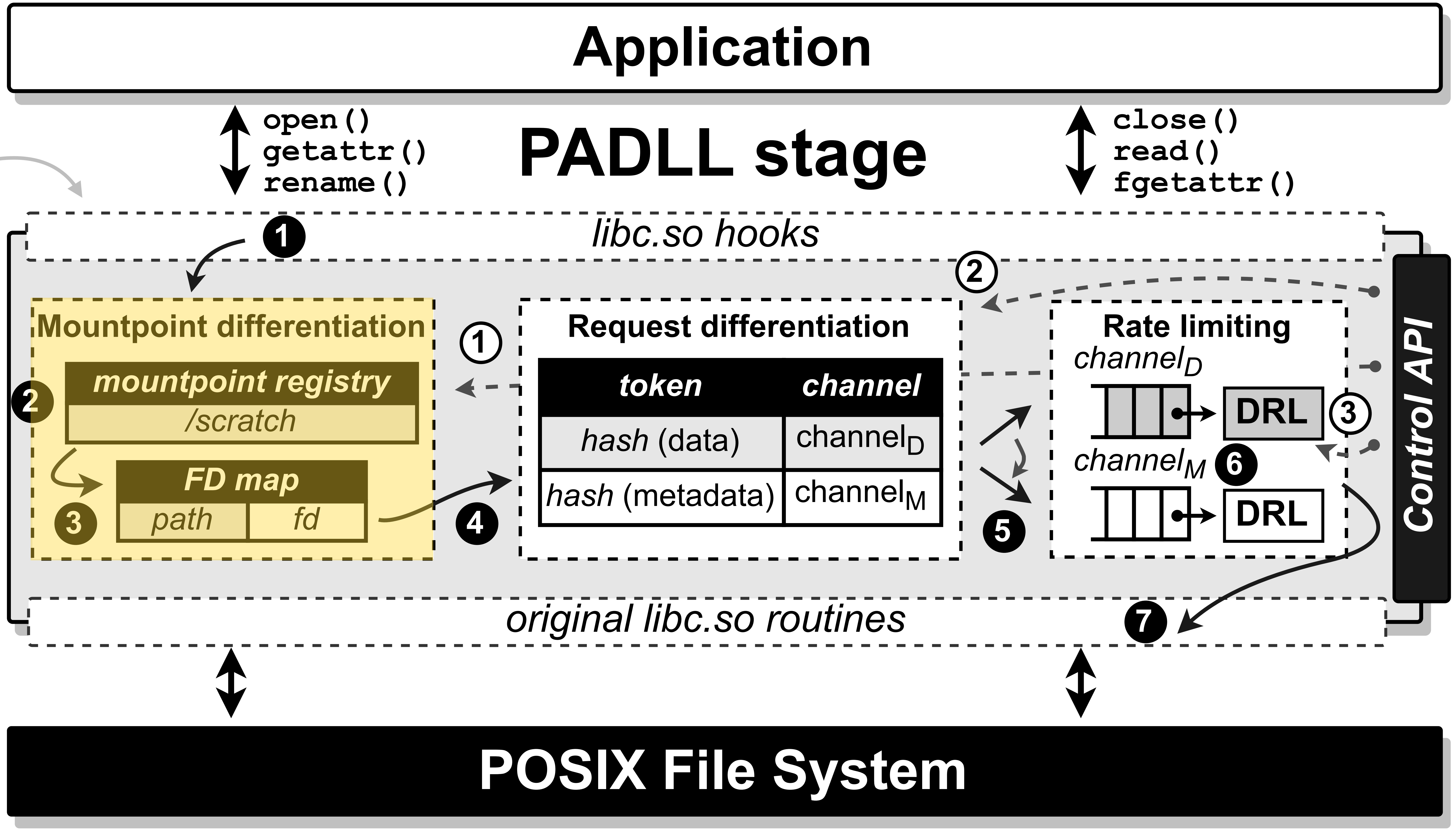
# PADLL

## Data plane: differentiation

Reimplements POSIX syscalls

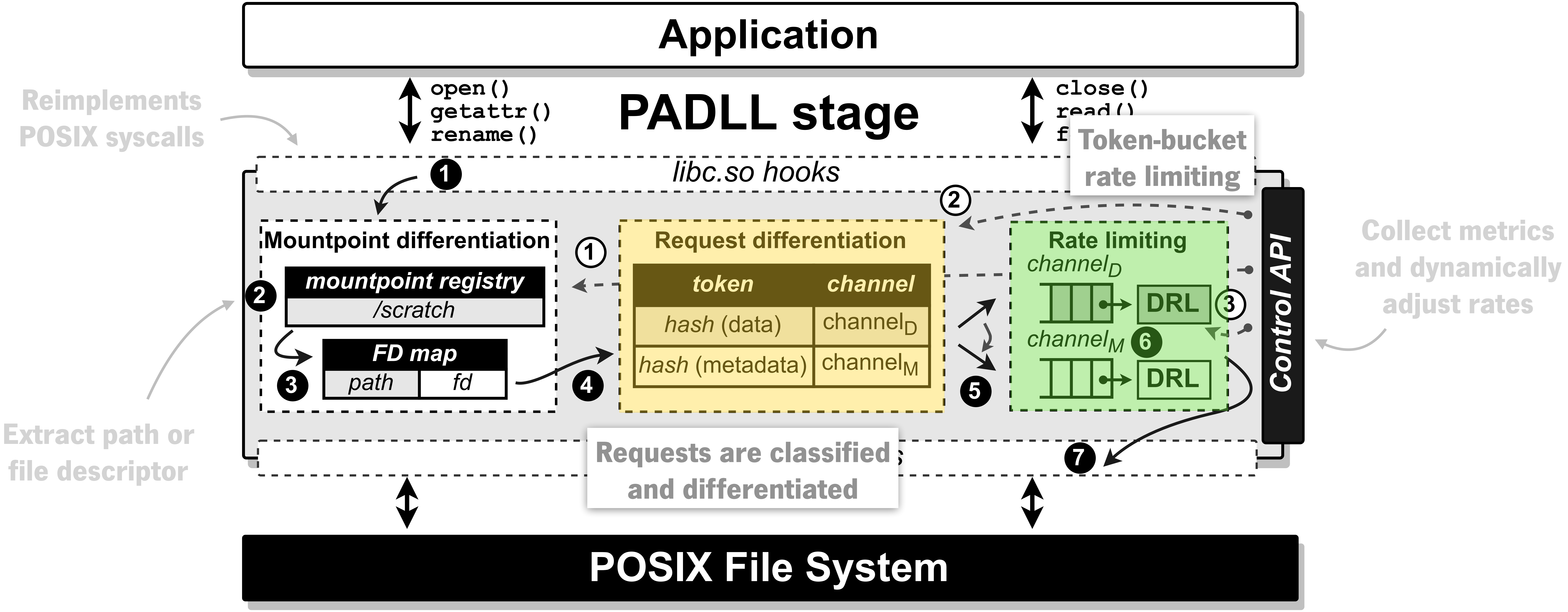
Collect metrics and dynamically adjust rates

Extract path or file descriptor



# PADLL

## Data plane: rate enforcement

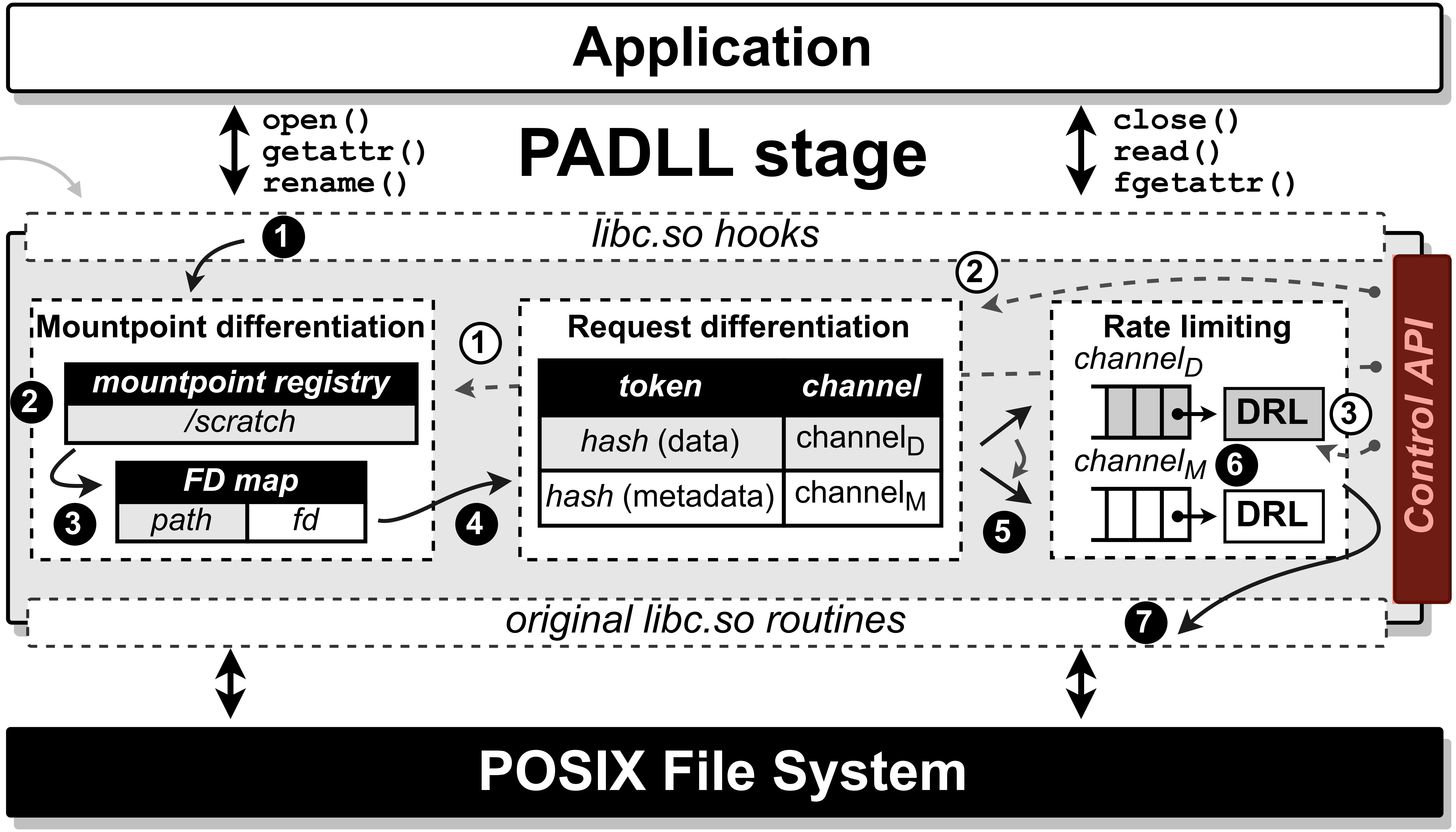


# PADLL

## Data plane: control

Reimplements POSIX syscalls

Extract path or file descriptor




Collect metrics and dynamically adjust rates

# PADLL
## Control logic

- Storage QoS policies are specified through **control algorithms**
  - **Static:** fixed I/O limits for accessing shared storage
    - Example: limit `open` operations to X ops/s
  - **Dynamic:** assign resource shares (i.e., bandwidth, IOPS) that change over time
    - Adaptable to workload and system variations
    - Example: limit `metadata` operations to at least K ops/s

- Control algorithms are implemented in a **feedback control loop**
  - **Collect** I/O metrics from data plane stages
  - Verify if QoS limits are being respected and **computes** new rules for uncompliant stages
  - **Enforce** new rules to the corresponding stages

# Implementation

- Data and control plane implemented in 16k and 6k lines of C++ code

- Support of 42 POSIX calls from different operation classes
  - Including data, metadata, extended attributes, and directory management

- Data plane was built using the **PAIO**[1] data plane framework
  - Request differentiation and rate limiting

- Communication between components
  - Local controllers and data plane stages communicate through **UNIX Domain Sockets**
  - Controllers communicate through **RPC**

**Macedo** *et al. "PAIO: General, Portable I/O Optimizations With Minor Application Modifications"*. USENIX FAST, 2022.

# Evaluation

- **Can PADLL control I/O workflows at different granularities?**
  - Per-operation **type** rate limiting (section V.A)
  - Per-operation **class** rate limiting (section V.B)
- **Can PADLL enforce QoS policies over concurrent jobs?**
  - **Per-job** rate limiting and **QoS control** (section V.C)
- **What is the performance of PADLL control and data plane?**
  - Performance, resource usage, and overhead (section V.D)

# Evaluation

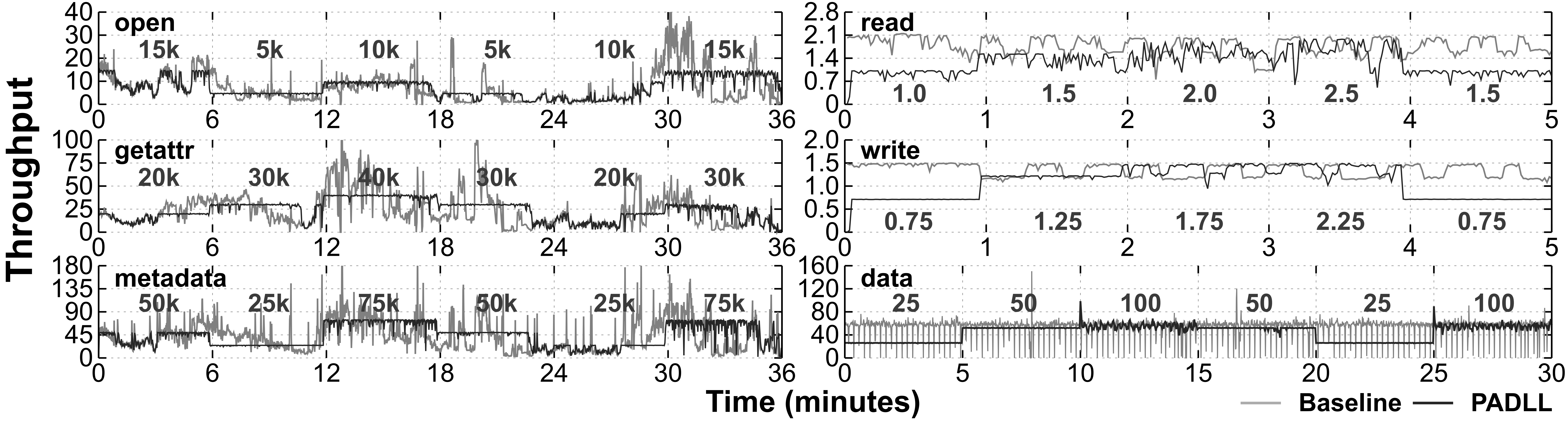
- **Experimental testbed (configuration A)**
  - Compute nodes of the ABCI supercomputer
  - Two 20-core Intel Xeon, 384 GiB RAM, and an InfiniBand EDR network card
  - CentoOS 7.5 with Linux kernel v3.10
  - **Dedicated** Lustre file system composed of 2 MDS/MDTs and 24 OSTs with 359 TiB

- **Benchmarks and workloads**
  - **Metadata:** traces collected from ABCI's production Lustre file system
    - Trace replayer that replicates the original traces at different scales
  - **Data:** IOR and TensorFlow

- **Methodology**
  - Global controller executes at a dedicated compute node
  - Local controller runs co-located with each job instance and respective data plane stages

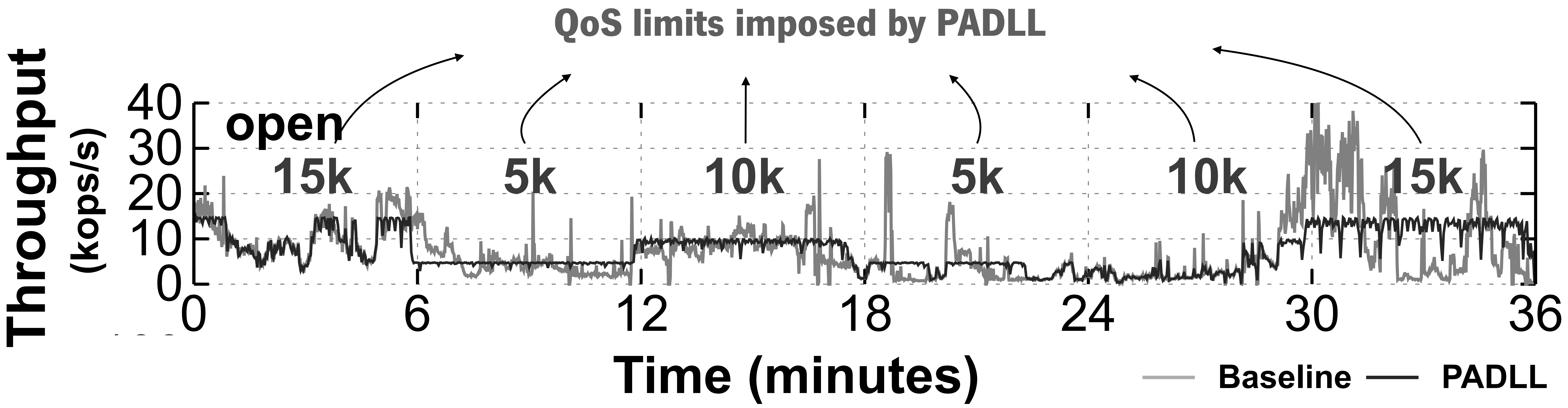
# Functional Evaluation

## Per-operation type and class rate limiting



# Functional Evaluation

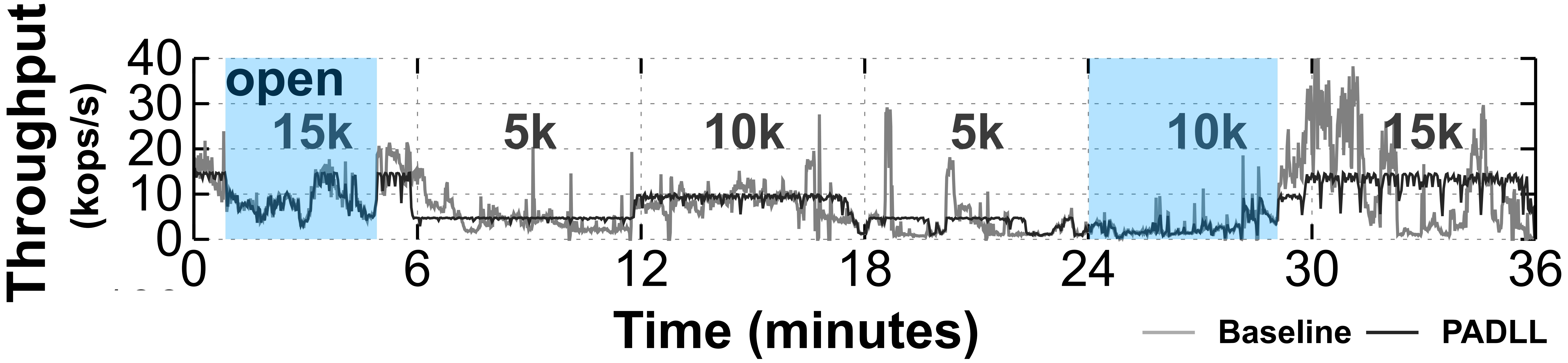
## Per-operation type rate limiting



# Functional Evaluation

## Per-operation type rate limiting

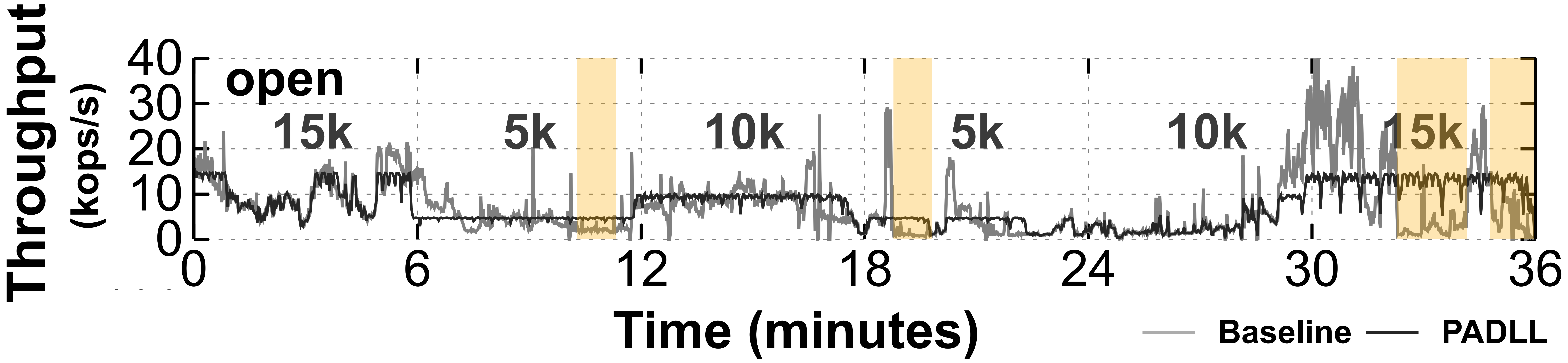
**Load is lower than imposed QoS limit**



# Functional Evaluation

## Per-operation type rate limiting

Backlog of operations due to aggressive rate limiting




**We draw similar conclusions for the remainder functional evaluation scenarios**

# Evaluation

## Per-job QoS control

- **Objective**
  - Limit overall metadata load in the PFS, while assigning different I/O priorities to jobs
- **Experimental environment**
  - Multi-job QoS control in the ABCI supercomputer
- **Four jobs** replaying metadata operations of the ABCI cluster
  - **Overall load:** Job1 - 15% , Job2 - 20% , Job3 - 20% , Job4 - 45%
- **Setups and control algorithms**
  - Baseline, uniform, priority, proportional sharing, and proportional sharing without false allocation

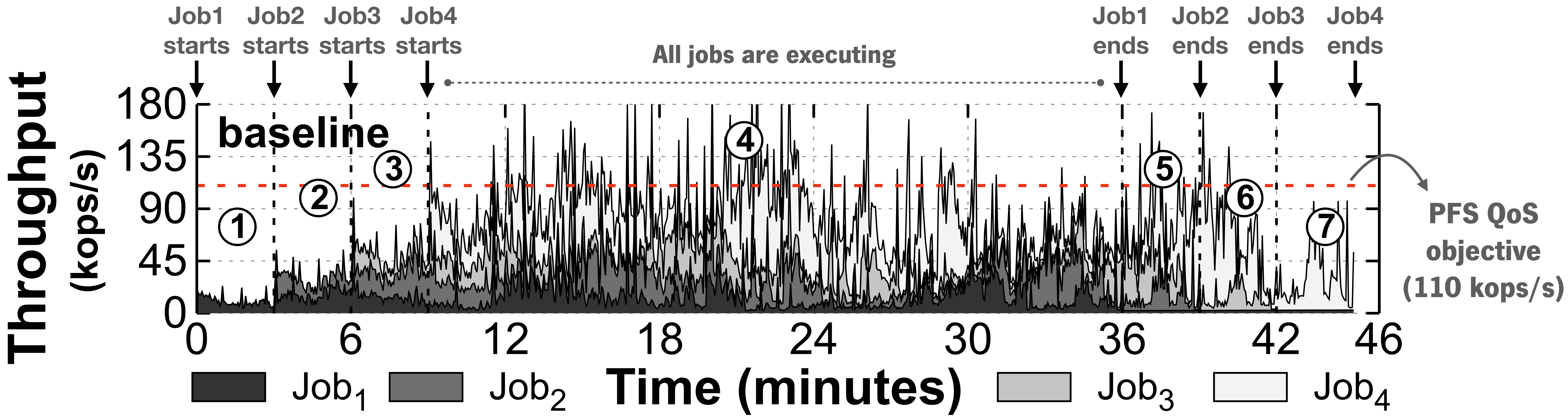
# Evaluation

## Per-job QoS control

**System configuration and workload**

- Maximum metadata rate is set to **110 kops/s**
- New job is added every 3 minutes
- Baseline execution time is 36 minutes (per job)
- Jobs execute with different loads {15%,20%,20%,45%}

**Baseline:** all jobs execute without being rate limited




✗ Volatile and bursty workload
✗ Peaks reaching over 600 kops/s

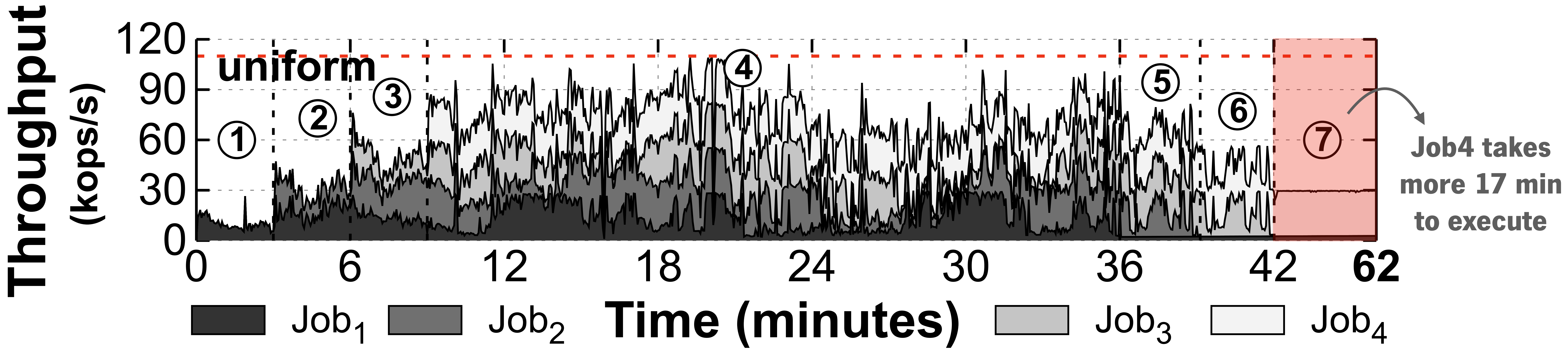
# Evaluation

## Per-job QoS control

**System configuration and workload**

- Maximum metadata rate is set to **110 kops/s**
- New job is added every 3 minutes
- Baseline execution time is 36 minutes (per job)
- Jobs execute with different loads {15%,20%,20%,45%}

**Uniform:** each job is rate limited with the same priority (27.5 kops/s)




✓ Maximum metadata limit is respected, eliminating burstiness
✓ Each reservation of metadata is respected
✗ Does not enable priorities
✗ Jobs are aggressively rate limited

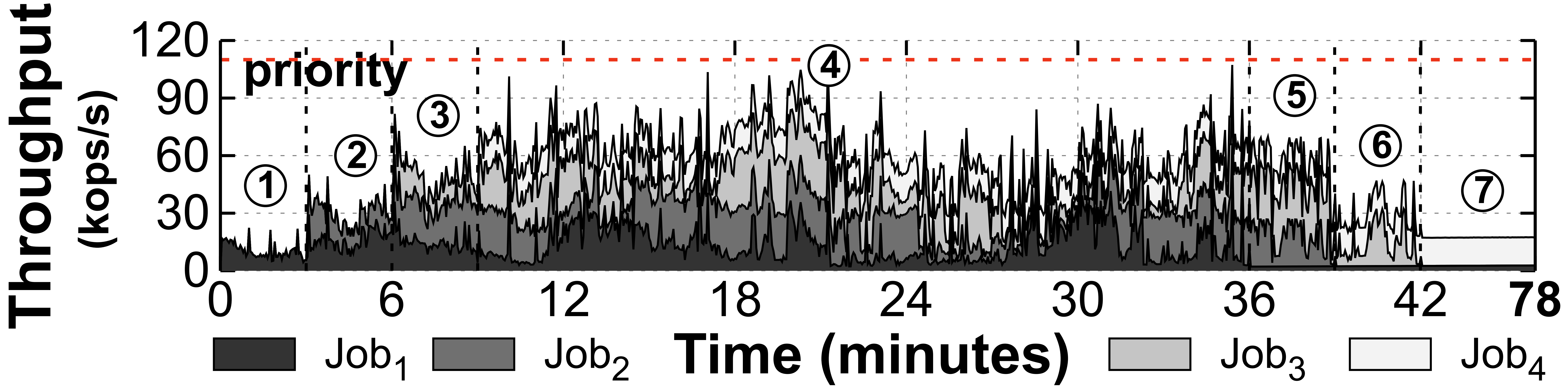
# Evaluation

## Per-job QoS control

**System configuration and workload**

- Maximum metadata rate is set to **110 kops/s**
- New job is added every 3 minutes
- Baseline execution time is 36 minutes (per job)
- Jobs execute with different loads {15%,20%,20%,45%}

**Priority:** each job is rate limited with a different priority (40, 25, 30, 15 kops/s)




✓ Maximum metadata limit is respected, eliminating burstiness
✓ Enables priorities between jobs

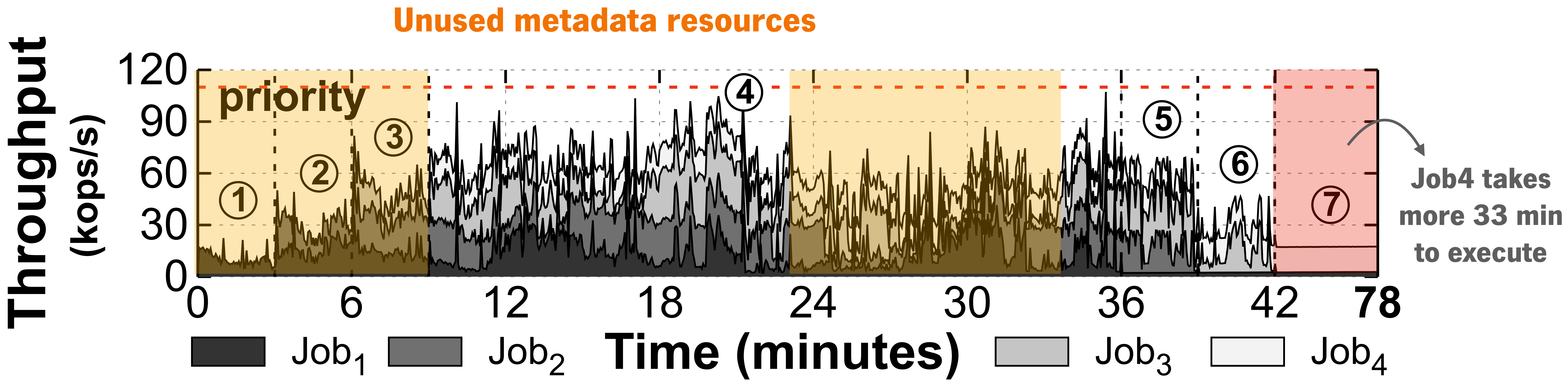
# Evaluation

## Per-job QoS control

**System configuration and workload**

- Maximum metadata rate is set to **110 kops/s**
- New job is added every 3 minutes
- Baseline execution time is 36 minutes (per job)
- Jobs execute with different loads {15%,20%,20%,45%}

**Priority:** each job is rate limited with a different priority (40, 25, 30, 15 kops/s)




✓ Maximum metadata limit is respected, eliminating burstiness
✓ Enables priorities between jobs
✗ Unable to use leftover metadata rate

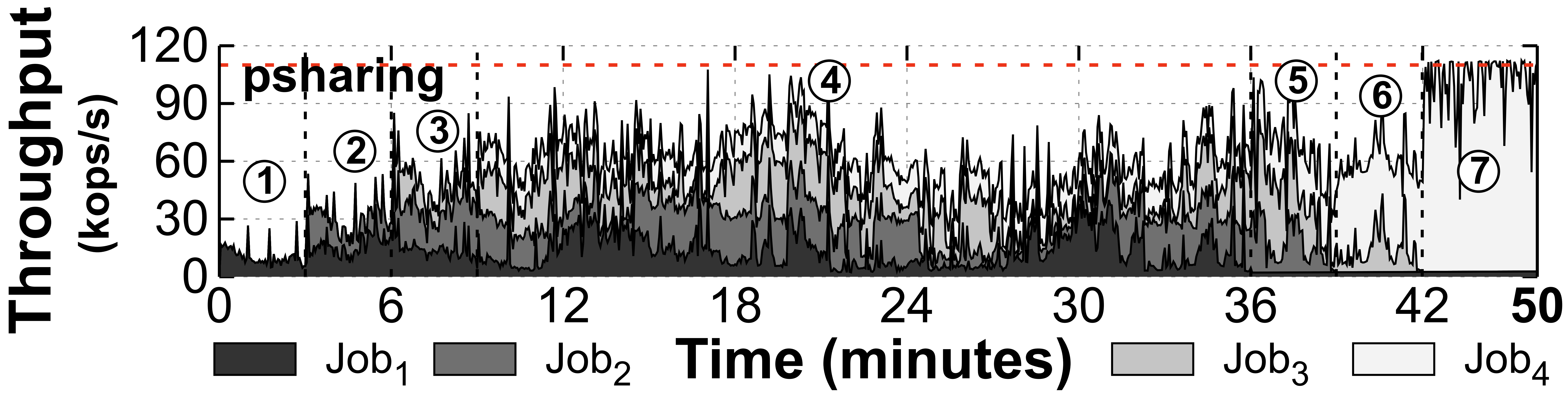
# Evaluation

## Per-job QoS control

**System configuration and workload**

- Maximum metadata rate is set to **110 kops/s**
- New job is added every 3 minutes
- Baseline execution time is 36 minutes (per job)
- Jobs execute with different loads {15%,20%,20%,45%}

**Proportional sharing:** enforce per-job metadata rate reservations, while assigning leftover rate when available




✓ Maximum metadata limit is respected, eliminating burstiness
✓ Leftover rate is assigned to jobs whenever available
✓ Prevents under-provisioning

# Evaluation

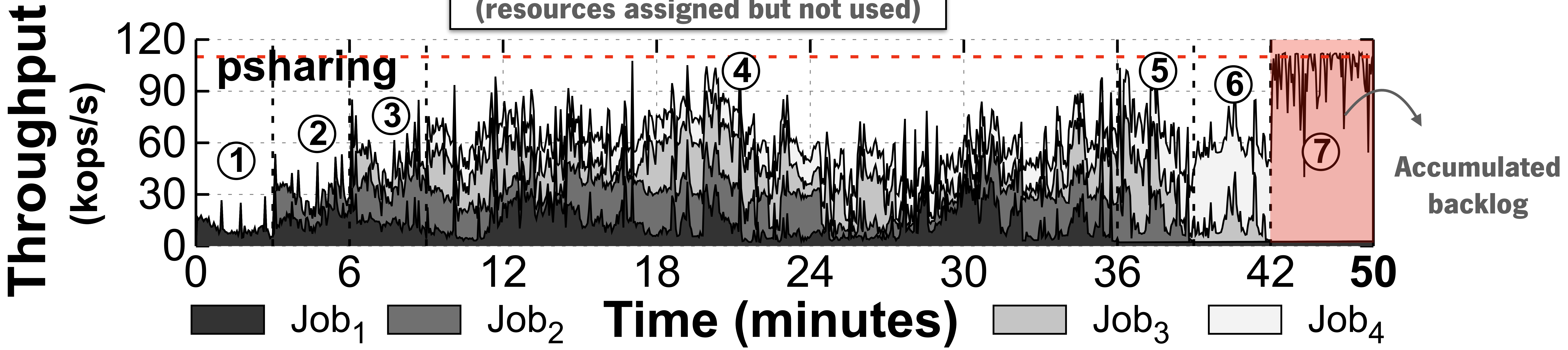
## Per-job QoS control

**System configuration and workload**

- Maximum metadata rate is set to **110 kops/s**
- New job is added every 3 minutes
- Baseline execution time is 36 minutes (per job)
- Jobs execute with different loads {15%,20%,20%,45%}

**Proportional sharing:** enforce per-job metadata rate reservations, while assigning leftover rate when available

**Long periods of over-provisioning (resources assigned but not used)**




- ✓ Maximum metadata limit is respected, eliminating burstiness
- ✓ Leftover rate is assigned to jobs whenever available
- ✓ Prevents under-provisioning
- ✗ Executes 5 minutes longer than Baseline

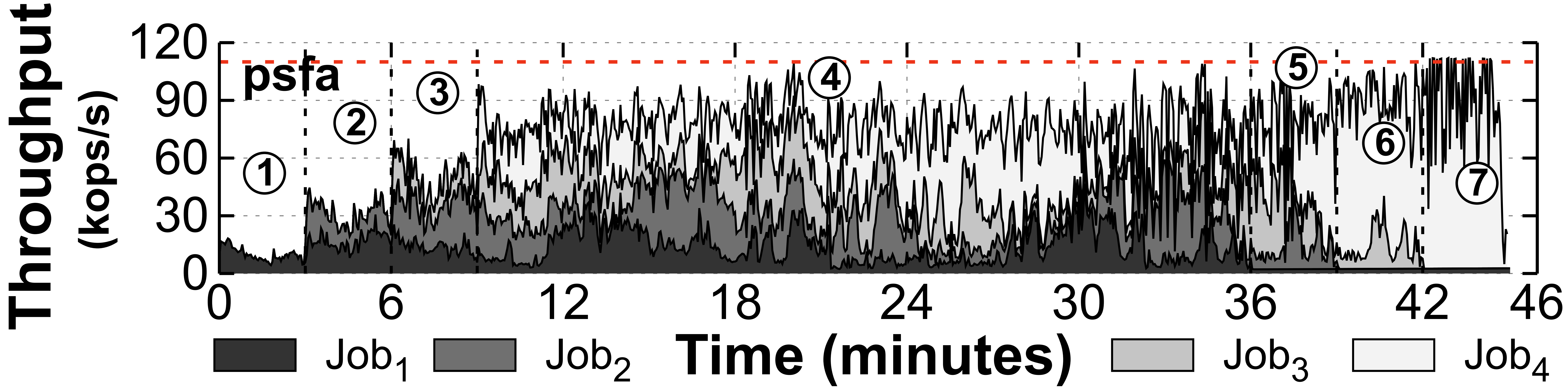
# Evaluation

## Per-job QoS control

**System configuration and workload**

- Maximum metadata rate is set to **110 kops/s**
- New job is added every 3 minutes
- Baseline execution time is 36 minutes (per job)
- Jobs execute with different loads {15%,20%,20%,45%}

**Proportional sharing w/o false allocation:** enforce per-job metadata rate reservations based on the actual I/O usage




✓ Maximum metadata limit is respected, eliminating burstiness
✓ Each reservation of metadata is respected

# Evaluation

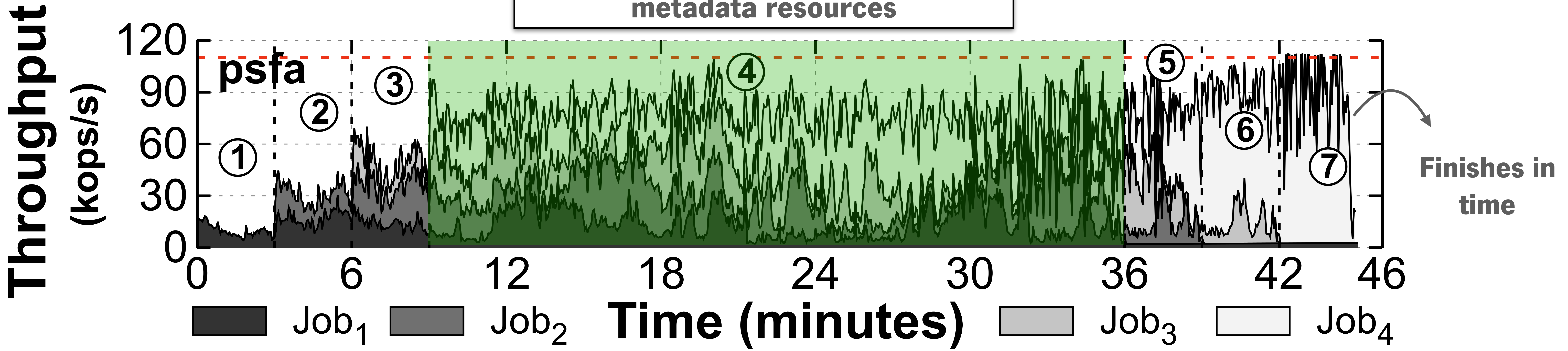
## Per-job QoS control

**System configuration and workload**

- Maximum metadata rate is set to **110 kops/s**
- New job is added every 3 minutes
- Baseline execution time is 36 minutes (per job)
- Jobs execute with different loads {15%,20%,20%,45%}

**Proportional sharing w/o false allocation:** enforce per-job metadata rate reservations based on the actual I/O usage

**Redistributes leftover and unused metadata resources**




✓ Maximum metadata limit is respected, eliminating burstiness
✓ Each reservation of metadata is respected
✓ Unused I/O resources are reassigned, preventing over-provisioning
✓ All jobs finish under the same time as Baseline

# Summary

- **PADLL** is an application and PFS-agnostic **storage middleware** that enables enforcing **QoS policies over metadata workflows** in HPC clusters
- Enables system administrators to proactively and holistically control the I/O rate of all running jobs
- New **max-min fair share** algorithm enables differentiated QoS control, while preventing resource over-provisioning under volatile workloads
- More details of the design, implementation, algorithm, and results in the paper
- All artifacts are publicly available
  - **Zenodo: 10.5281/zenodo.7627949**
  - **GitHub: dsrhaslab/padll** and **dsrhaslab/cheferd**

23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing

# Taming Metadata-intensive HPC Jobs Through Dynamic, Application-agnostic QoS Control

**Ricardo Macedo**[1], Mariana Miranda[1], Yusuke Tanimura[2], Jason Haga[2], Amit Ruhela[3], Stephen L. Harrell[3], Richard Todd Evans[4], José Pereira[1], João Paulo[1]

[1] INESC TEC & University of Minho, [2] AIST, [3] UTAustin & TACC, [4] Intel