

# PAIO: General, Portable I/O Optimizations with Minor Application Modifications

Ricardo Macedo<sup>1</sup>, Yusuke Tanimura<sup>2</sup>, Jason Haga<sup>2</sup>, Vijay Chidambaram<sup>3</sup>,  
José Pereira<sup>1</sup>, João Paulo<sup>1</sup>

<sup>1</sup> INESC TEC and University of Minho, <sup>2</sup> AIST, <sup>3</sup> UTAustin and VMware Research



# Data-centric systems

- Data-centric systems have become an integral part of modern I/O stacks
- Good performance for these systems often requires storage optimizations
  - Scheduling, caching, tiering, replication, ...
- Optimizations are implemented in sub-optimal manner



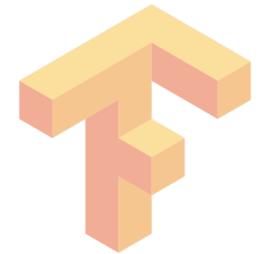
# Data-centric systems

- Data-centric systems have become an integral part of modern I/O stacks
- Good performance optimizations
  - Scheduling
- Optimizations are implemented in sub-optimal manner

**There is a better way to implement I/O optimizations**

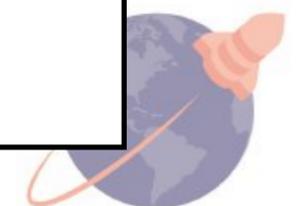


levelDB



mongoDB

Torch



kafka



ceph

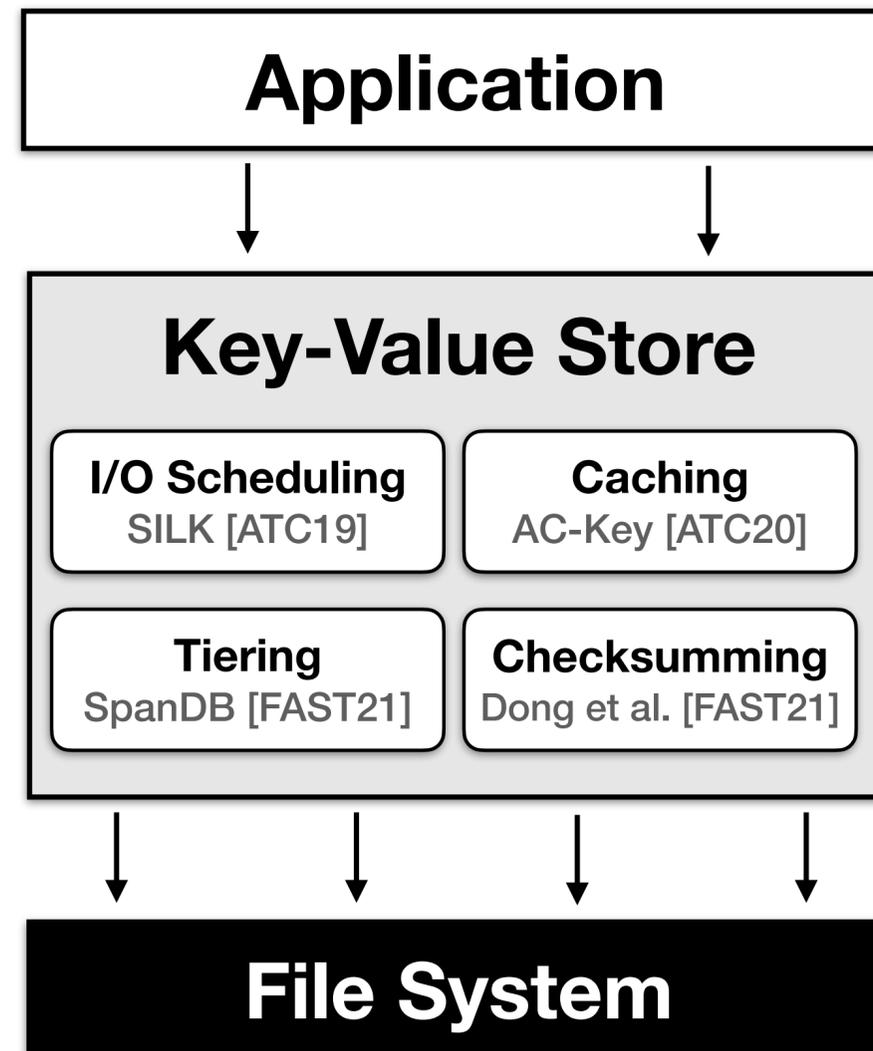


cassandra

# Challenge #1

## ⊗ Tightly coupled optimizations

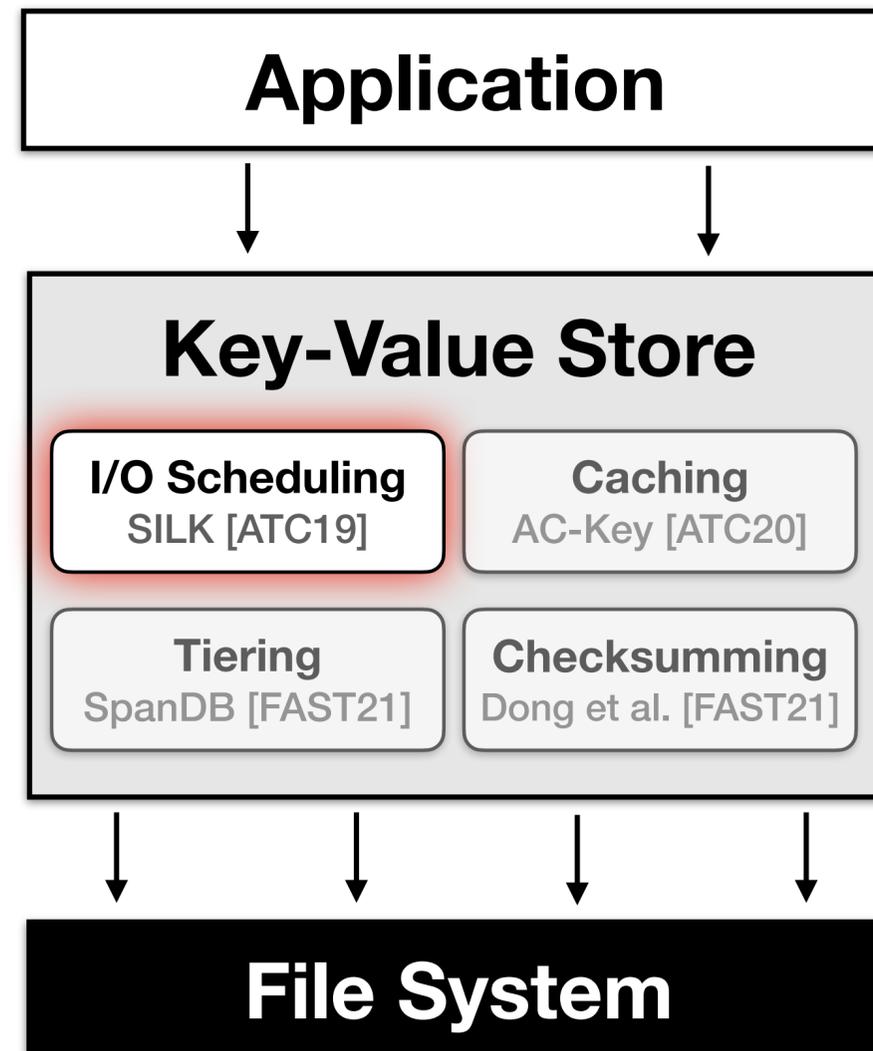
- I/O optimizations are single purposed
- Require deep understanding of the system's internal operation model
- Require profound system refactoring
- Limited portability across systems



# Challenge #1

## ⊗ Tightly coupled optimizations

- I/O optimizations are single purposed
- Require deep understanding of the system's internal operation model
- Require profound system refactoring
- Limited portability across systems



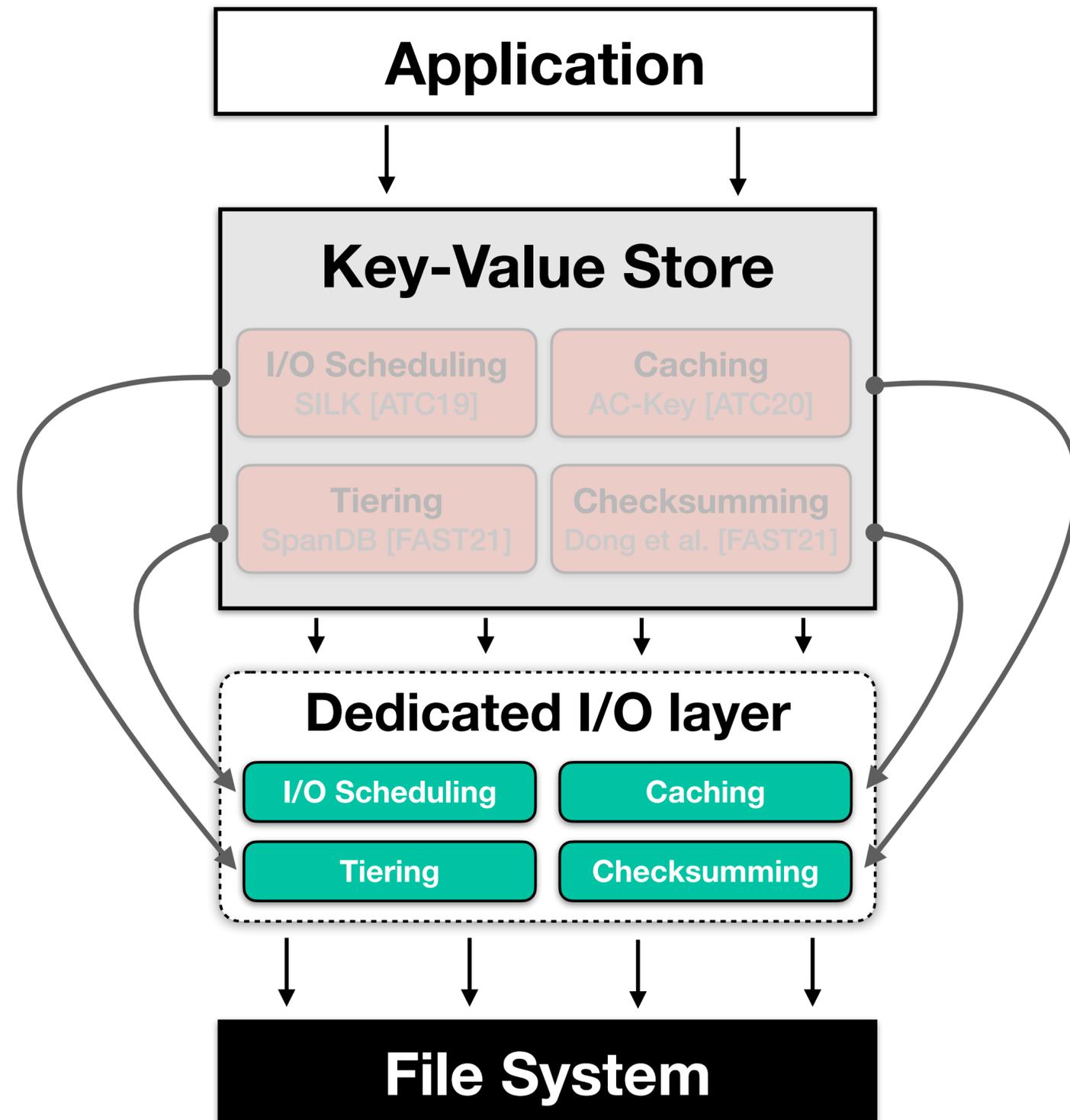
## SILK's I/O Scheduler

- Reduce tail latency spikes in RocksDB
- Controls the interference between foreground and background tasks
- Required changing several modules, such as *background operation handlers*, *internal queuing logic*, and *thread pools*

# Challenge #1

## ✓ Decoupled optimizations

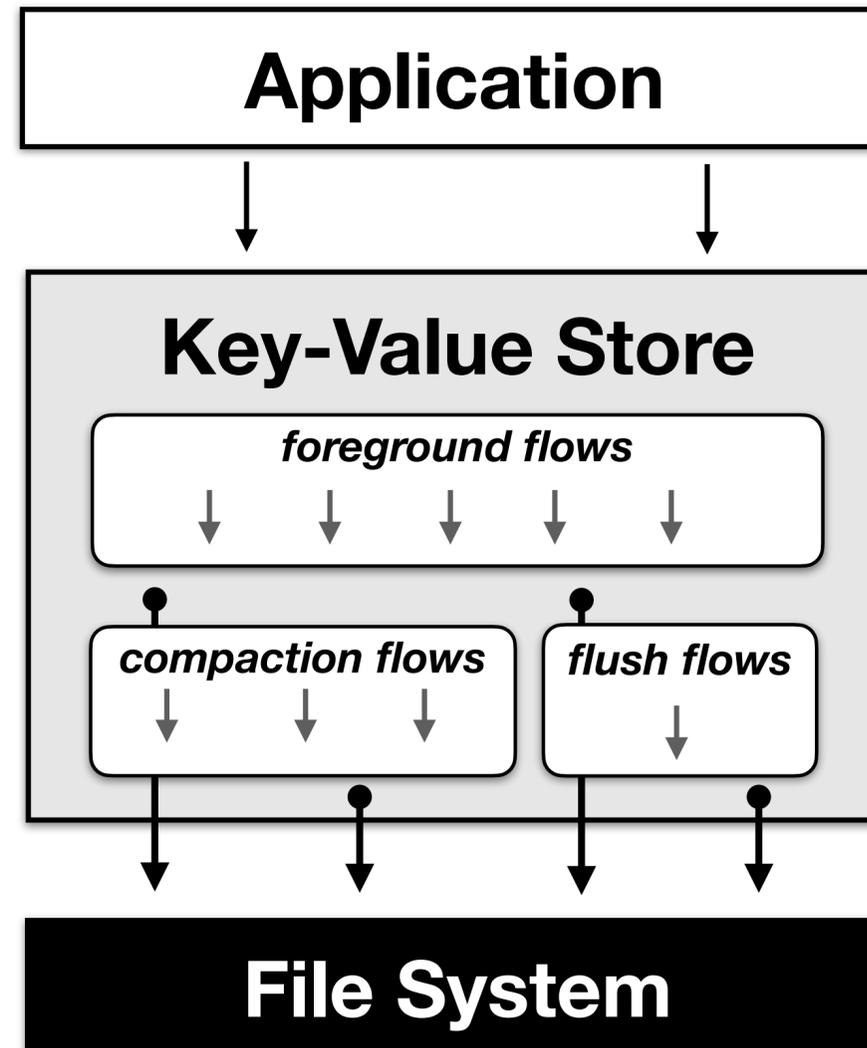
- I/O optimizations should be disaggregated from the internal logic
- Moved to a dedicated I/O layer
- Generally applicable
- Portable across different scenarios



# Challenge #2

## ⊗ Rigid interfaces

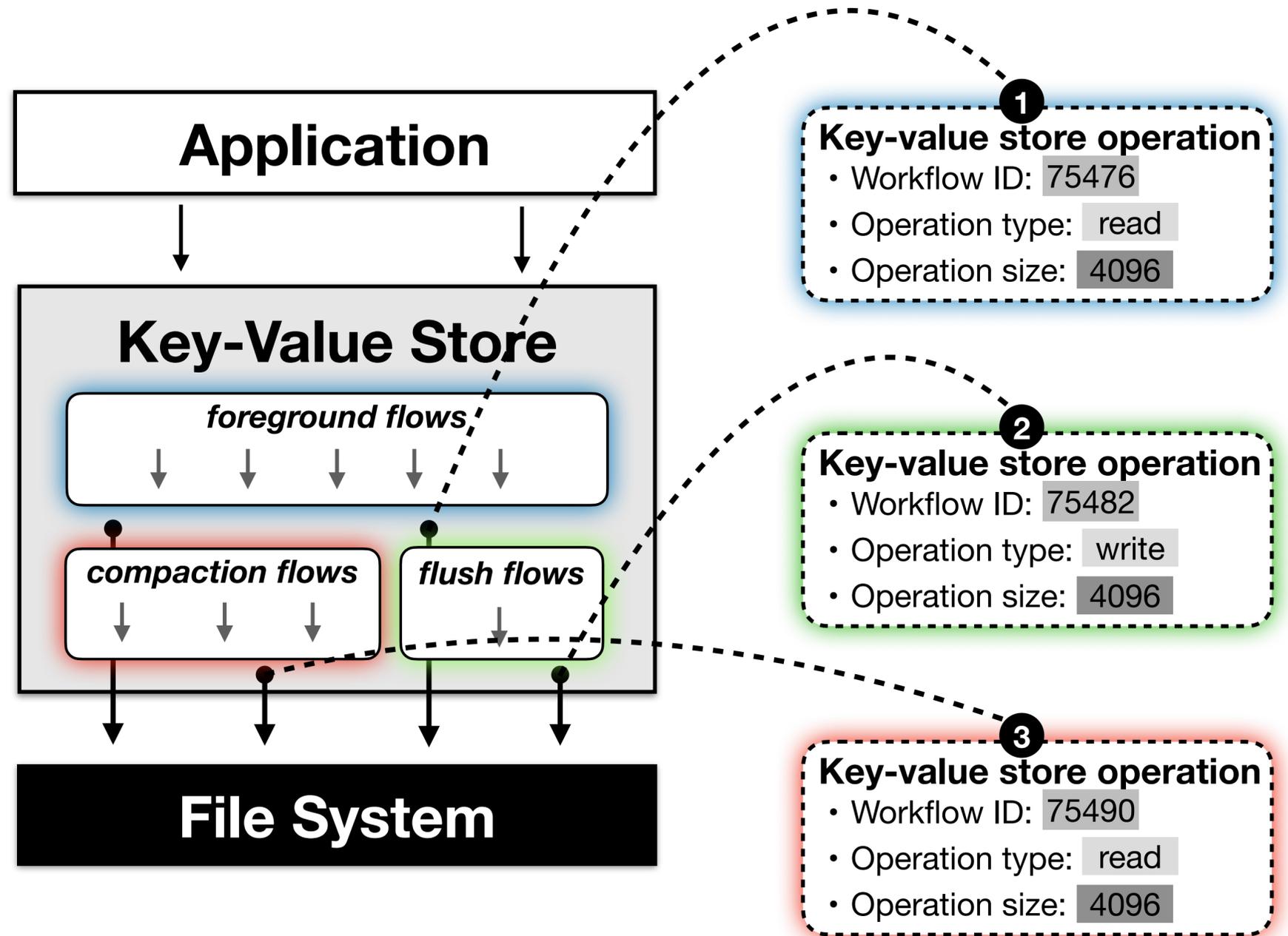
- Decoupled optimizations lose granularity and internal application knowledge
- I/O layers communicate through rigid interfaces
- Discard information that could be used to classify and differentiate requests



# Challenge #2

## ⊗ Rigid interfaces

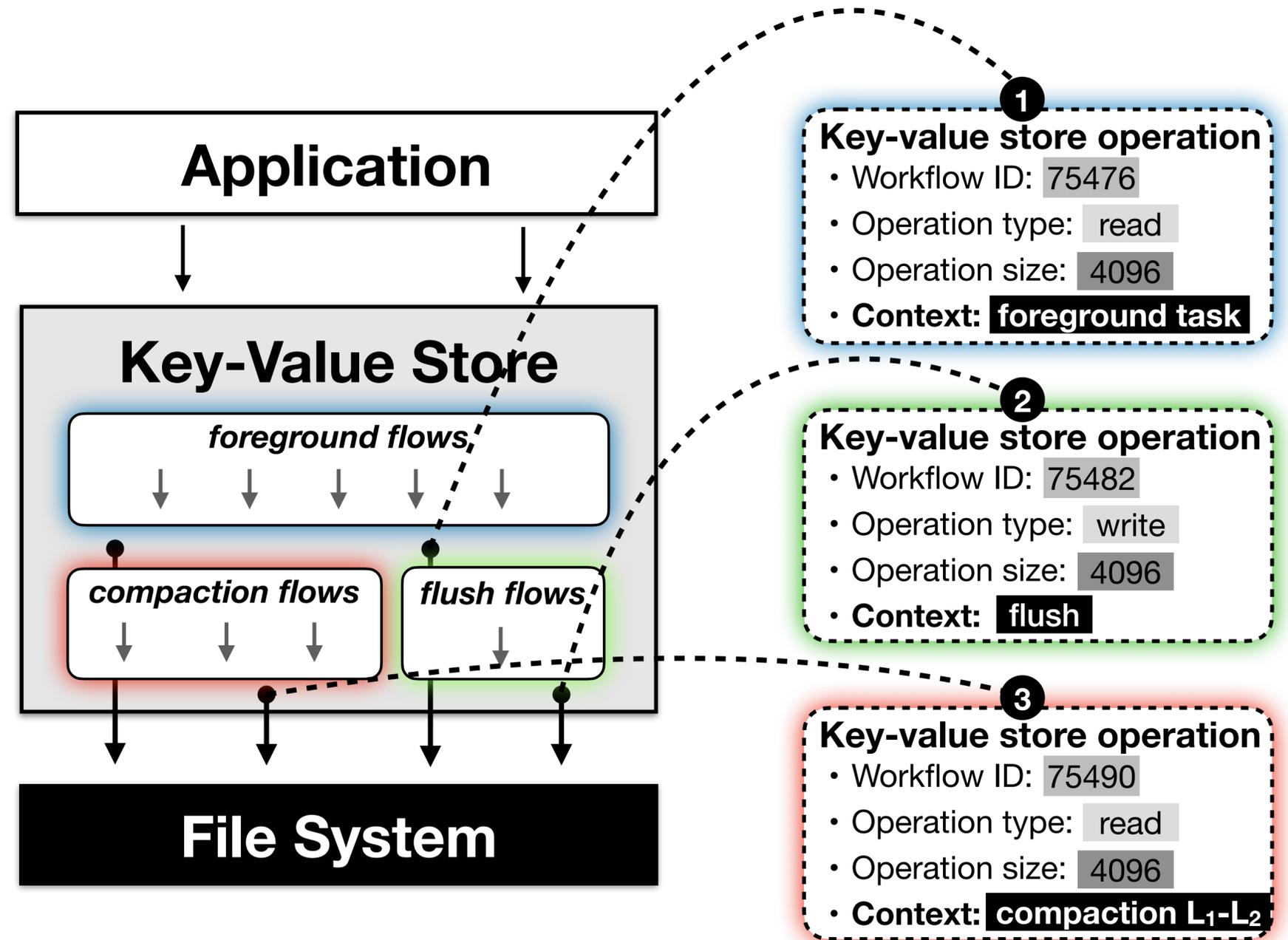
- Decoupled optimizations lose granularity and internal application knowledge
- I/O layers communicate through rigid interfaces
- Discard information that could be used to classify and differentiate requests



# Challenge #2

## ✓ Information propagation

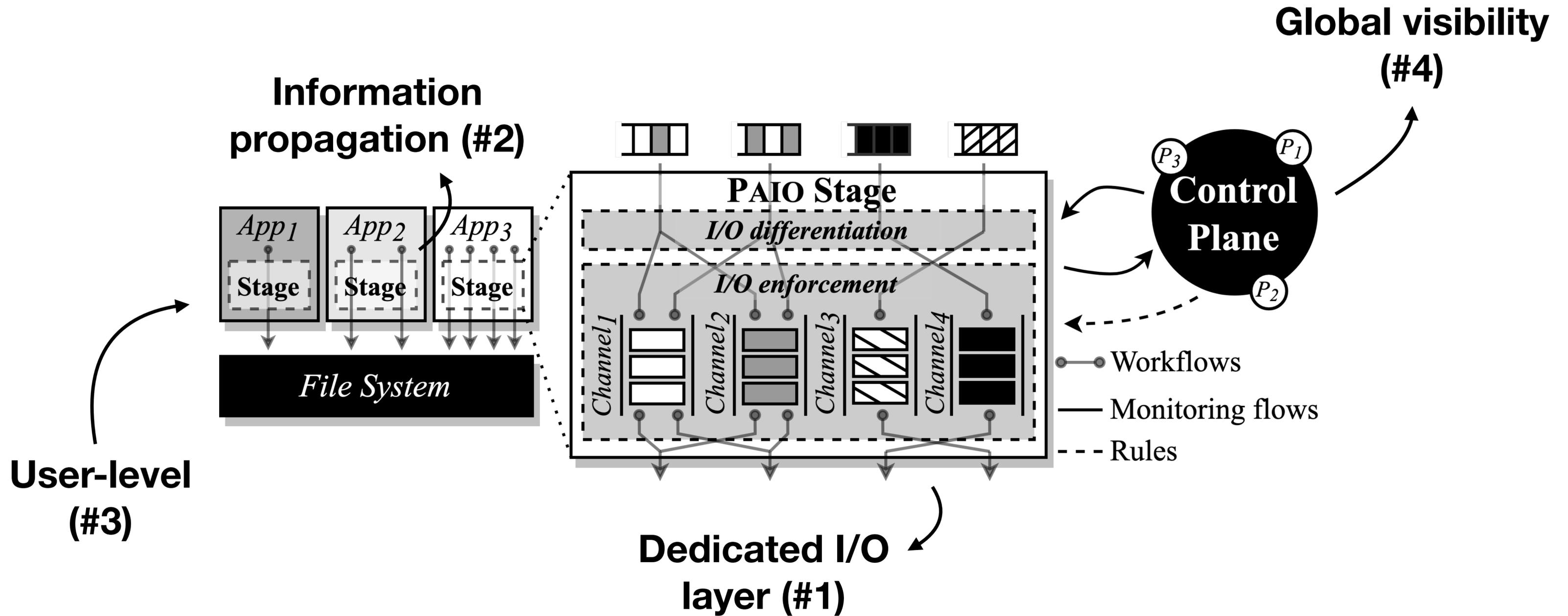
- Application-level information must be propagated throughout layers
- Decoupled optimizations can provide the same level of control and performance



# PAIO

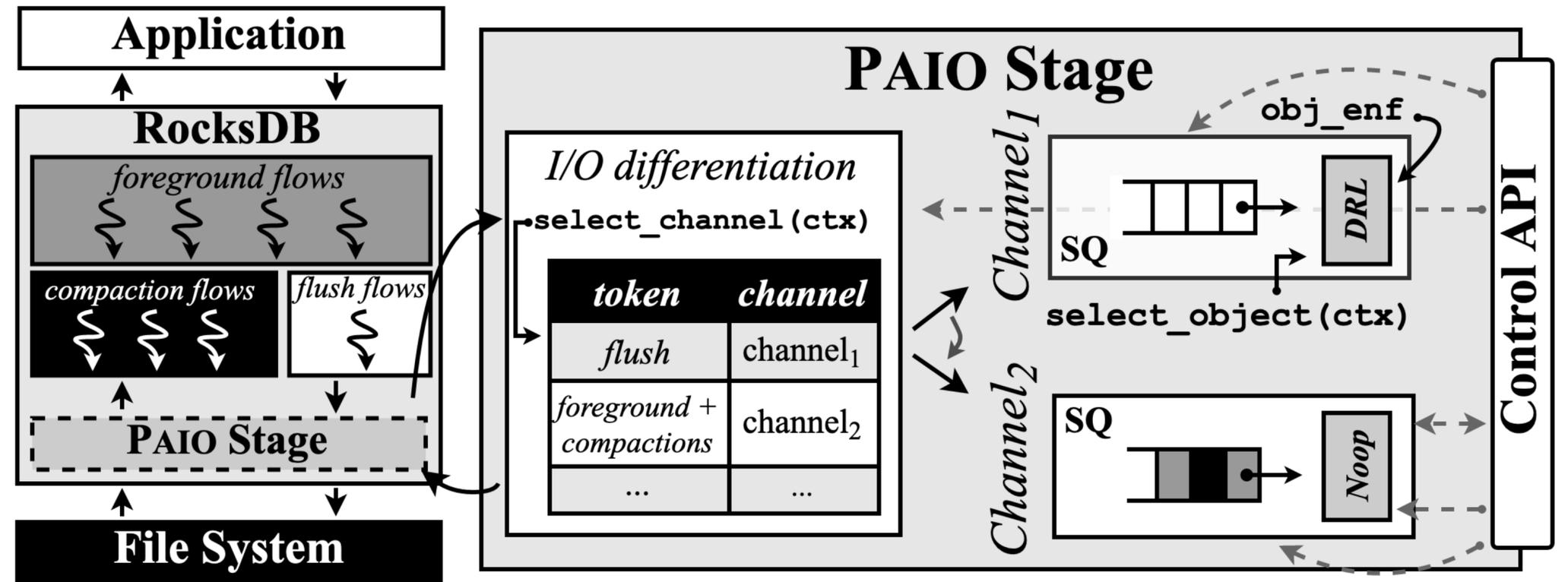
- **User-level** framework for building **portable** and **generally applicable** optimizations
- Adopts ideas from **Software-Defined Storage**
  - I/O optimizations are implemented *outside* applications as **data plane stages**
  - **Stages** are controlled through a **control plane** for coordinated access to resources
- Enables the propagation of application-level information through **context propagation**
- Porting I/O layers to use PAIO requires **none to minor** code changes

# PAIO design



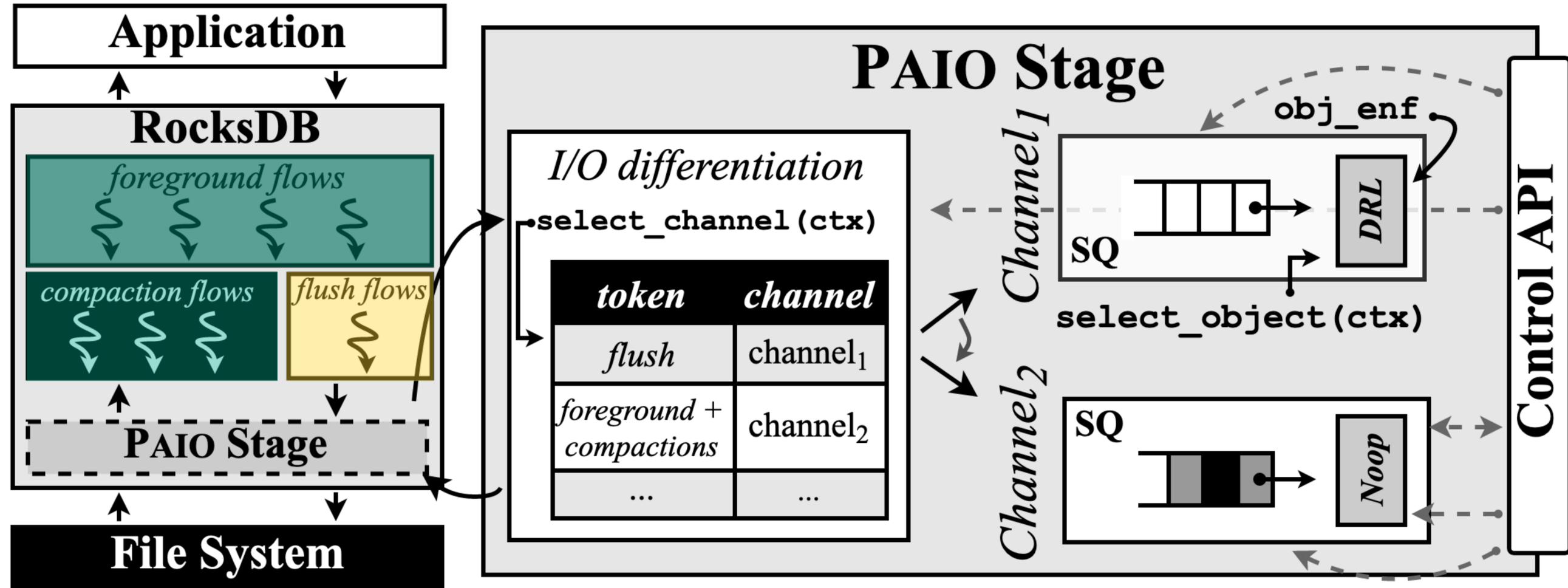
# PAIO design

- I/O differentiation
- I/O enforcement
- Control plane interaction



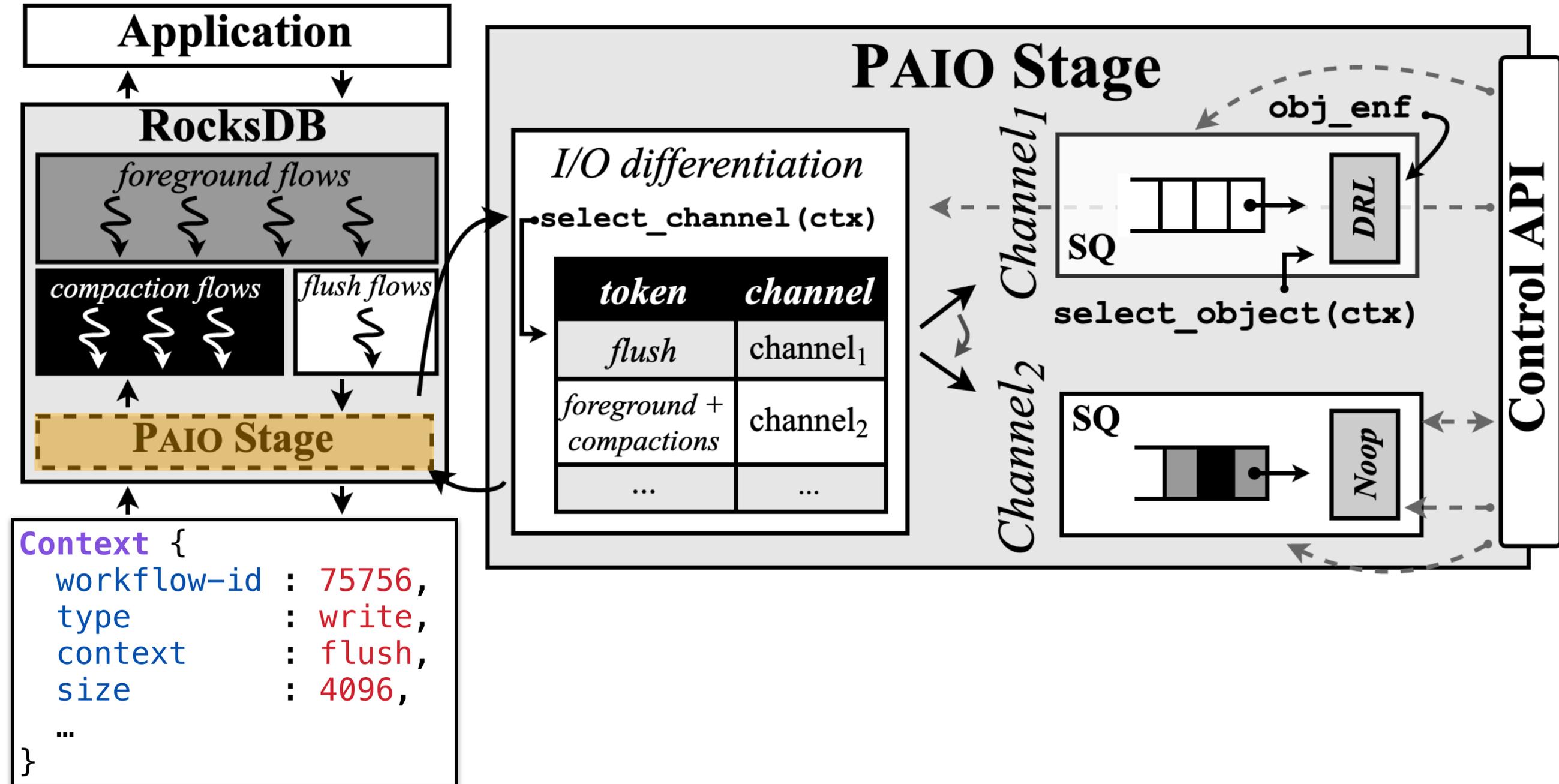
**Policy:** limit the rate of RocksDB's flush operations to  $X$  MiB/s

# I/O differentiation

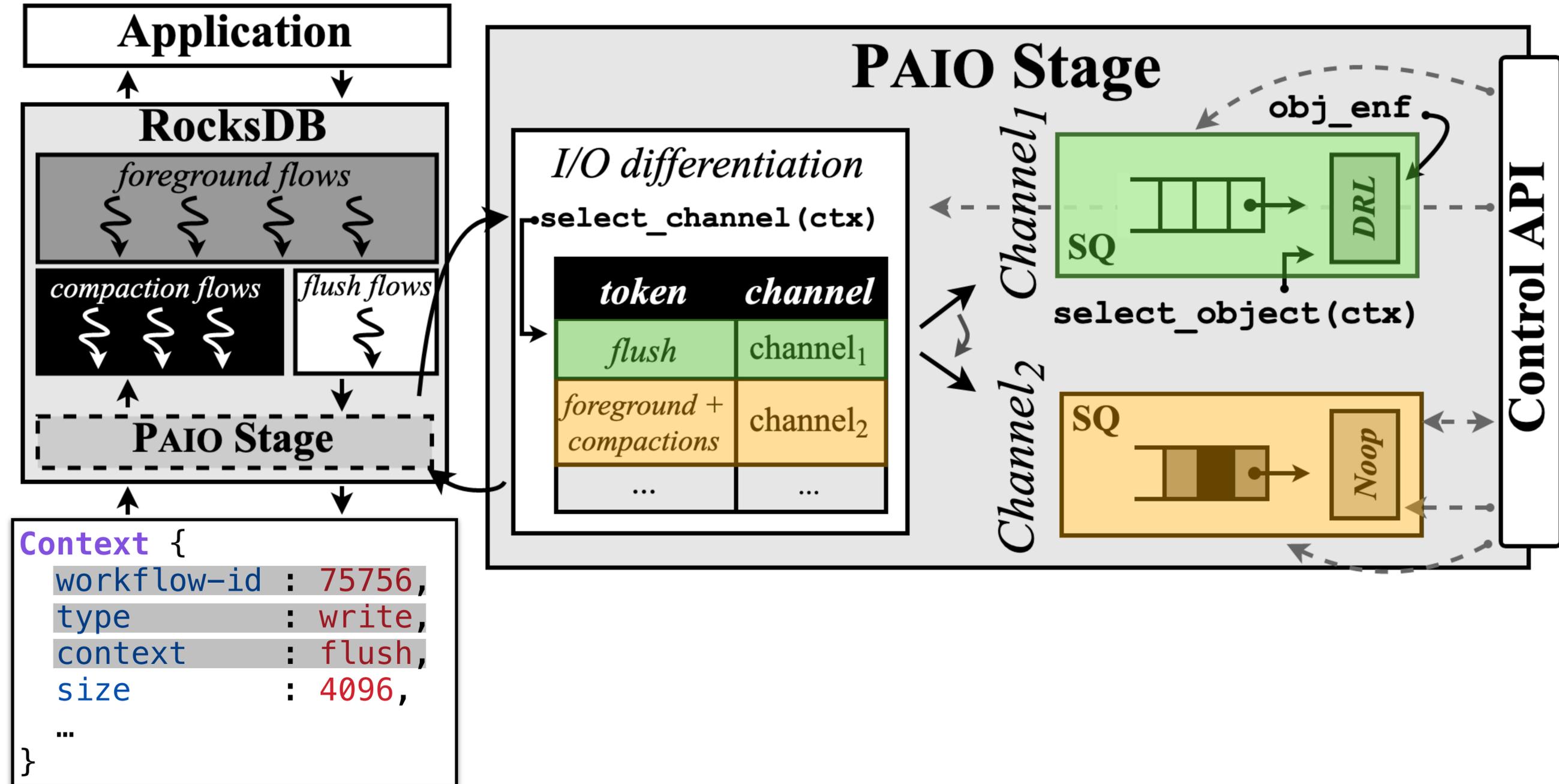


Identify the origin of POSIX operations (i.e., foreground, compaction, or flush operations)

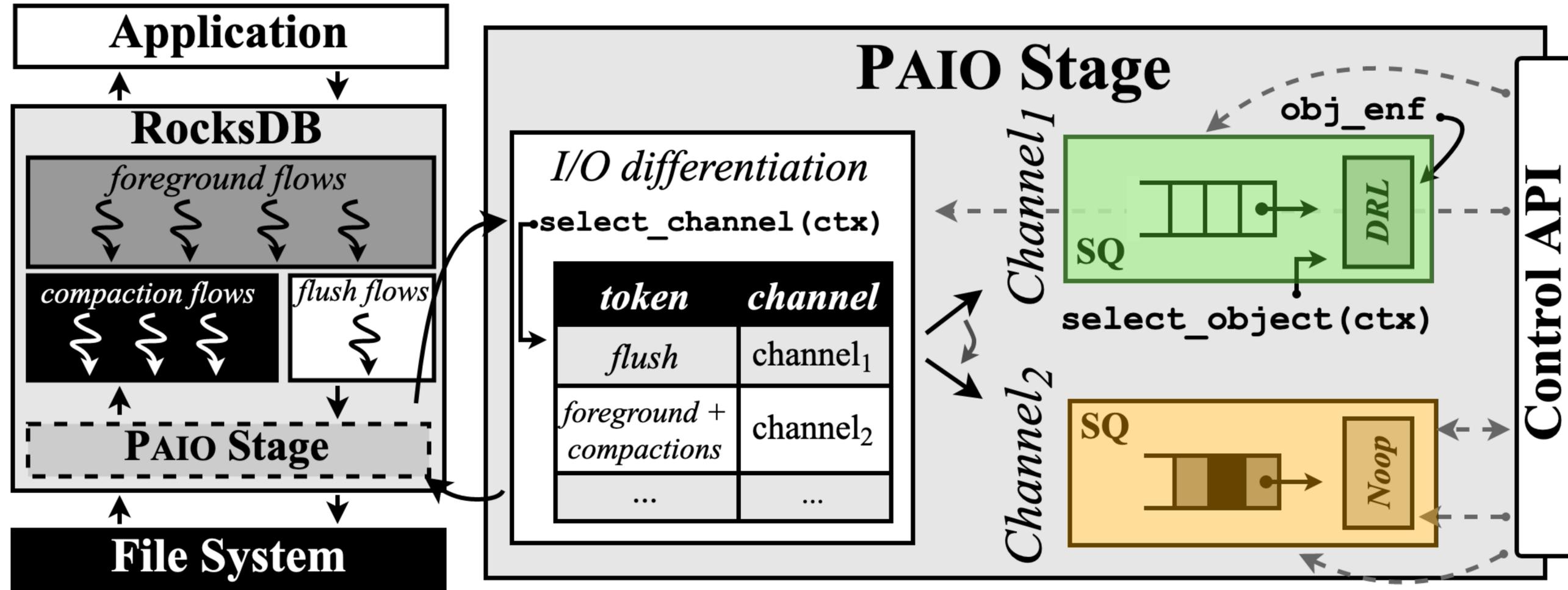
# I/O differentiation



# I/O differentiation

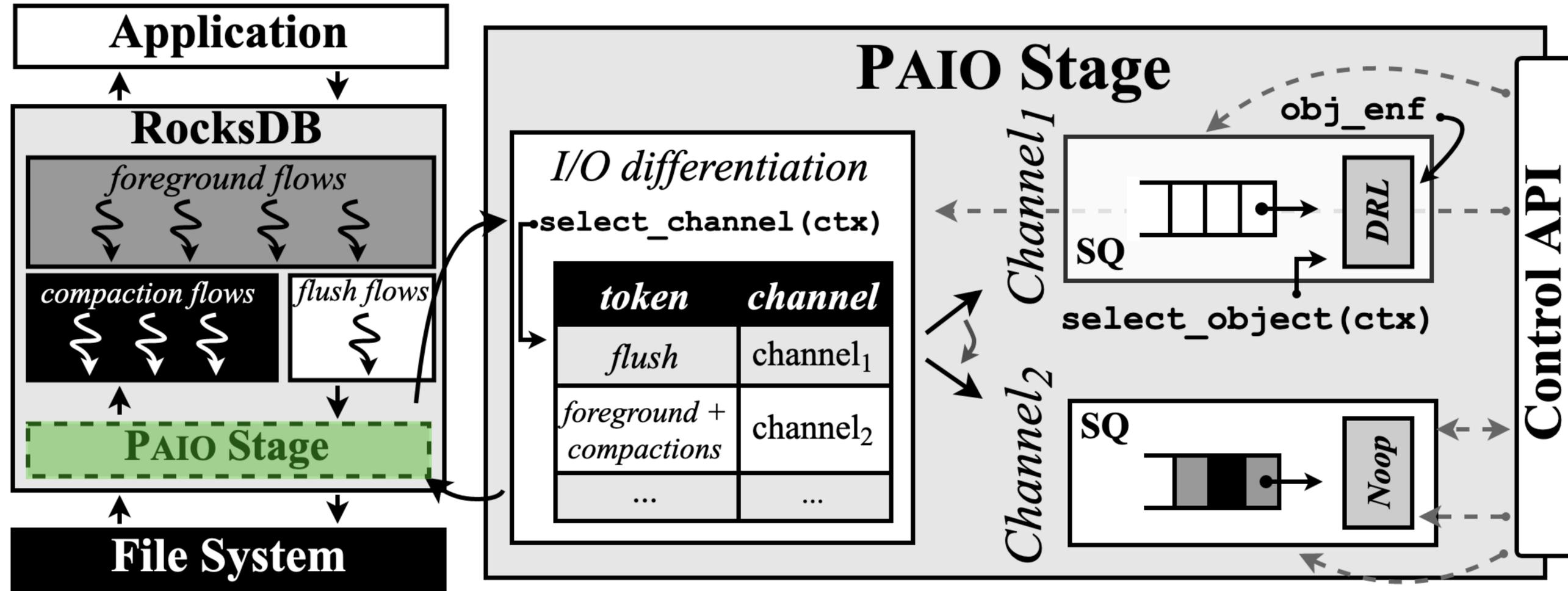


# I/O enforcement



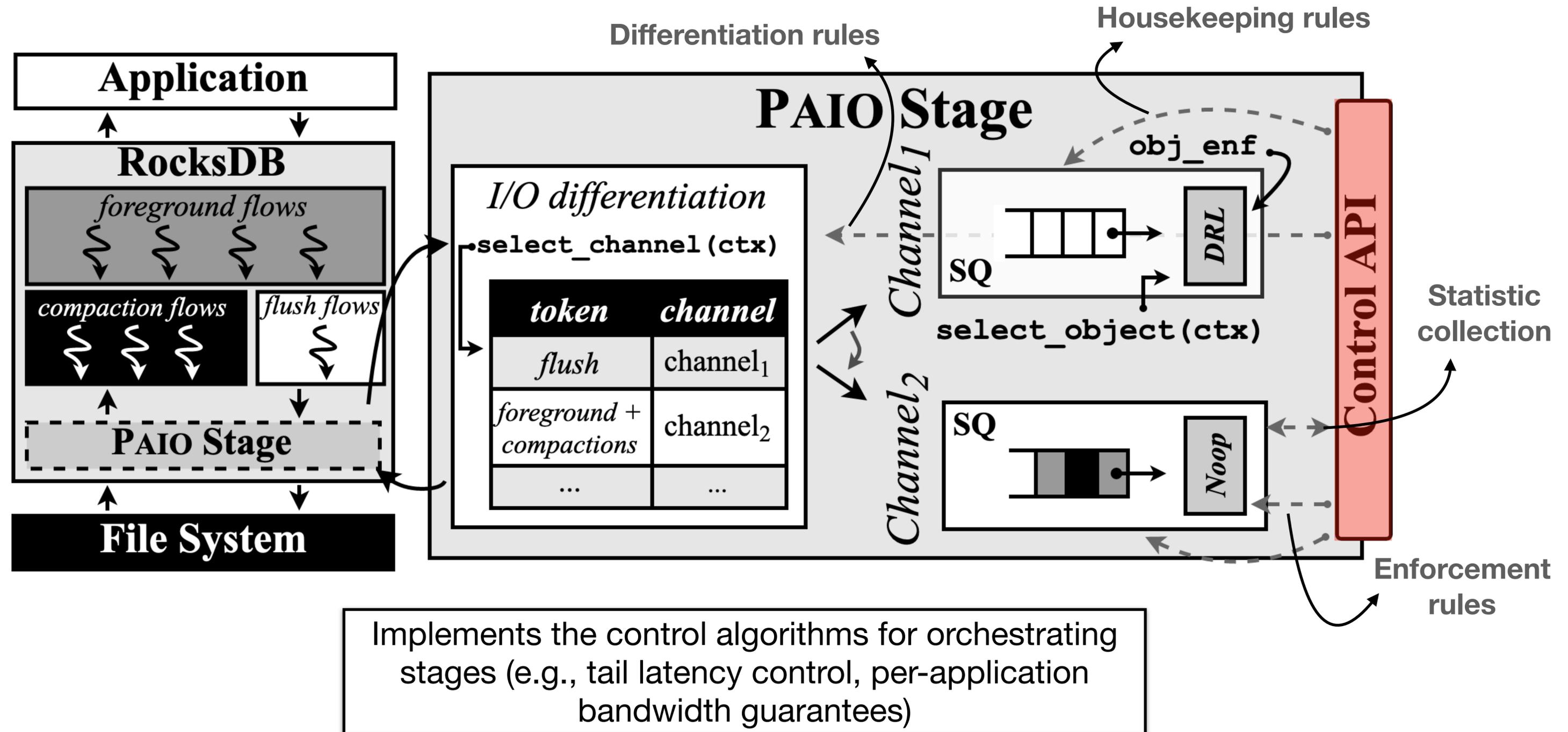
PAIO currently supports **Noop** and **DRL** enforcement objects

# I/O enforcement



Requests return to their original I/O path

# Control plane interaction



# Tail Latency Control in LSM-based Key-Value Stores

## RocksDB

- Interference between foreground and background tasks generates high latency spikes
- Latency spikes occur due to  $L_0$ - $L_1$  compactions and flushes being slow or on hold

## SILK

- I/O scheduler
  - Allocates bandwidth for internal operations when client load is low
  - Prioritizes flushes and low level compactions
  - Preempts high level compactions with low level ones
- Required changing several core modules made of thousands of LoC

## PAIO

- Stage provides the I/O mechanisms for prioritizing and rate limiting background flows
  - Integrating PAIO in RocksDB only required adding 85 LoC
- Control plane provides a SILK-based I/O scheduling algorithm

# Tail Latency Control in LSM-based Key-Value Stores

## RocksDB

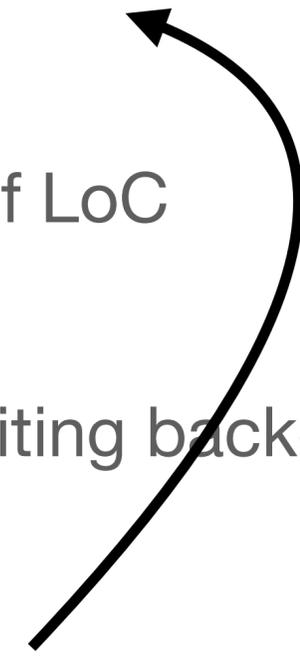
- Interference between foreground and background tasks generates high latency spikes
- Latency spikes occur due to  $L_0$ - $L_1$  compactions and flushes being slow or on hold

## SILK

- I/O scheduler
  - Allocates bandwidth for internal operations when client load is low
  - Prioritizes flushes and low level compactions
  - ~~Preempts high level compactions with low level ones~~
- Required changing several core modules made of thousands of LoC

## PAIO

- Stage provides the I/O mechanisms for prioritizing and rate limiting background flows
  - Integrating PAIO in RocksDB only required adding 85 LoC
- Control plane provides a SILK-based I/O scheduling algorithm



# Tail Latency Control in LSM-based Key-Value Stores

## RocksDB

- Interference between foreground and background tasks generates high latency spikes
- Latency spikes occur due to  $L_0$ - $L_1$  compactions and flushes being slow or on hold

## SILK

- I/O scheduler

**By propagating application-level information to the stage, PAIO can enable similar control and performance as system-specific optimizations**

- Required changing several core modules made of thousands of LoC

## PAIO

- Stage provides the I/O mechanisms for prioritizing and rate limiting background flows
  - Integrating PAIO in RocksDB only required adding 85 LoC
- Control plane provides a SILK-based I/O scheduling algorithm

# Experimental setup

## System configuration

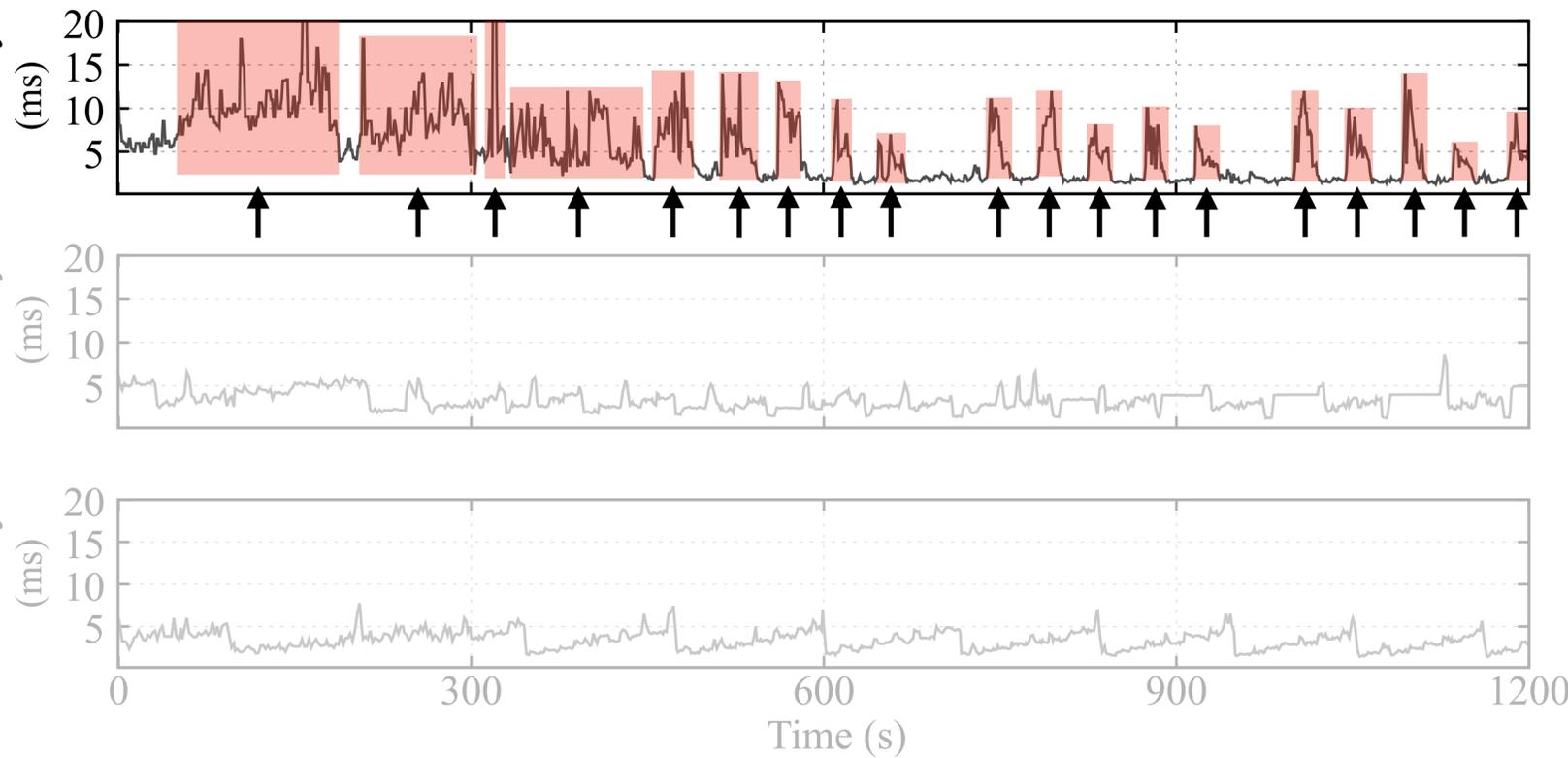
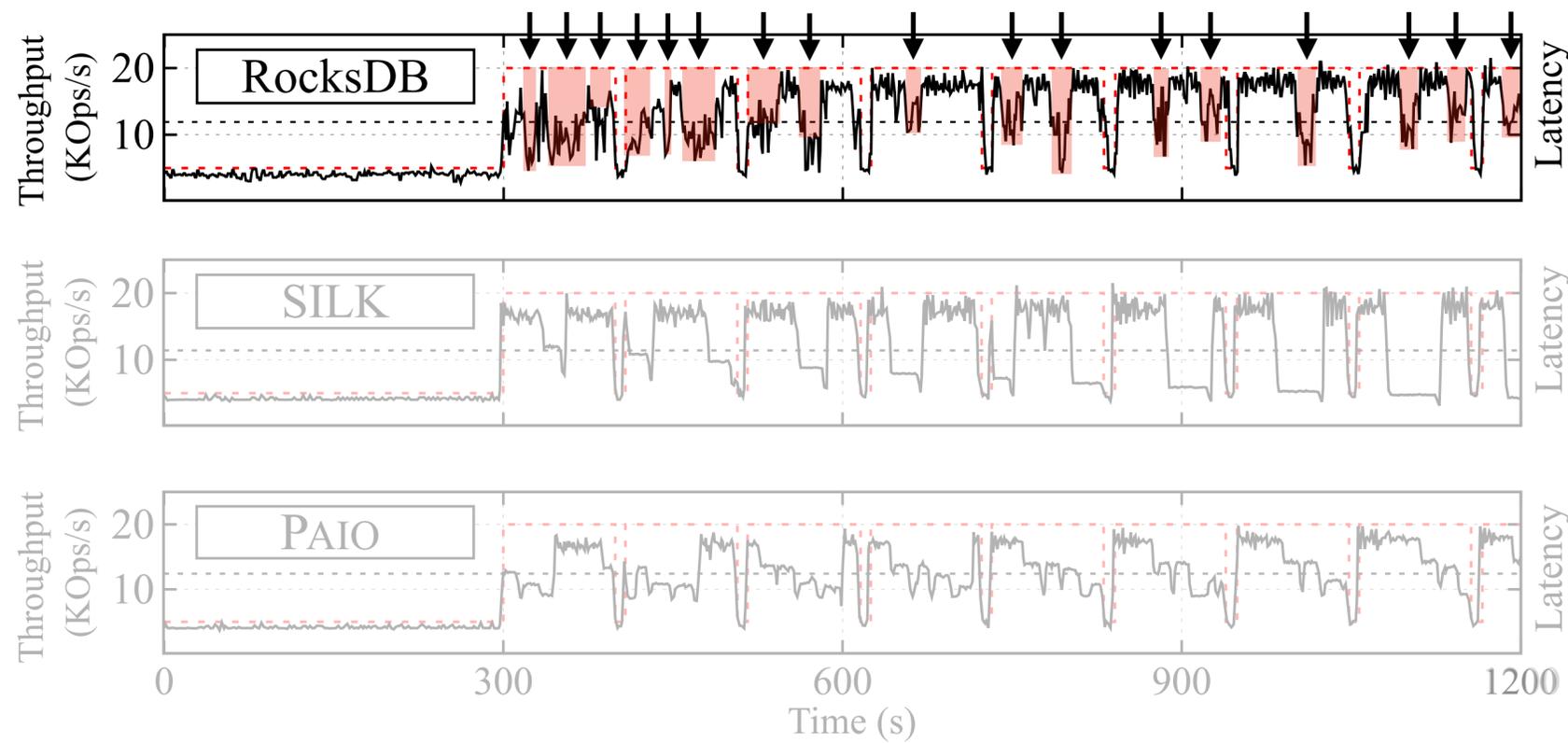
- RocksDB, SILK, and PAIO
- 8 client threads
- 8 background threads: 1 flush and 7 compaction threads
- Memory usage limited to 1GB and I/O bandwidth to 200MB/s

## Workloads

- Bursty clients (peaks and valleys)
- Initial valley of 300s at 5 KOps/s
- 100s peaks at 20 KOps/s and 10s valleys at 5 KOps/s
- Mixture, read-heavy, and write-heavy workloads

# Mixture workload

50% read 50% write

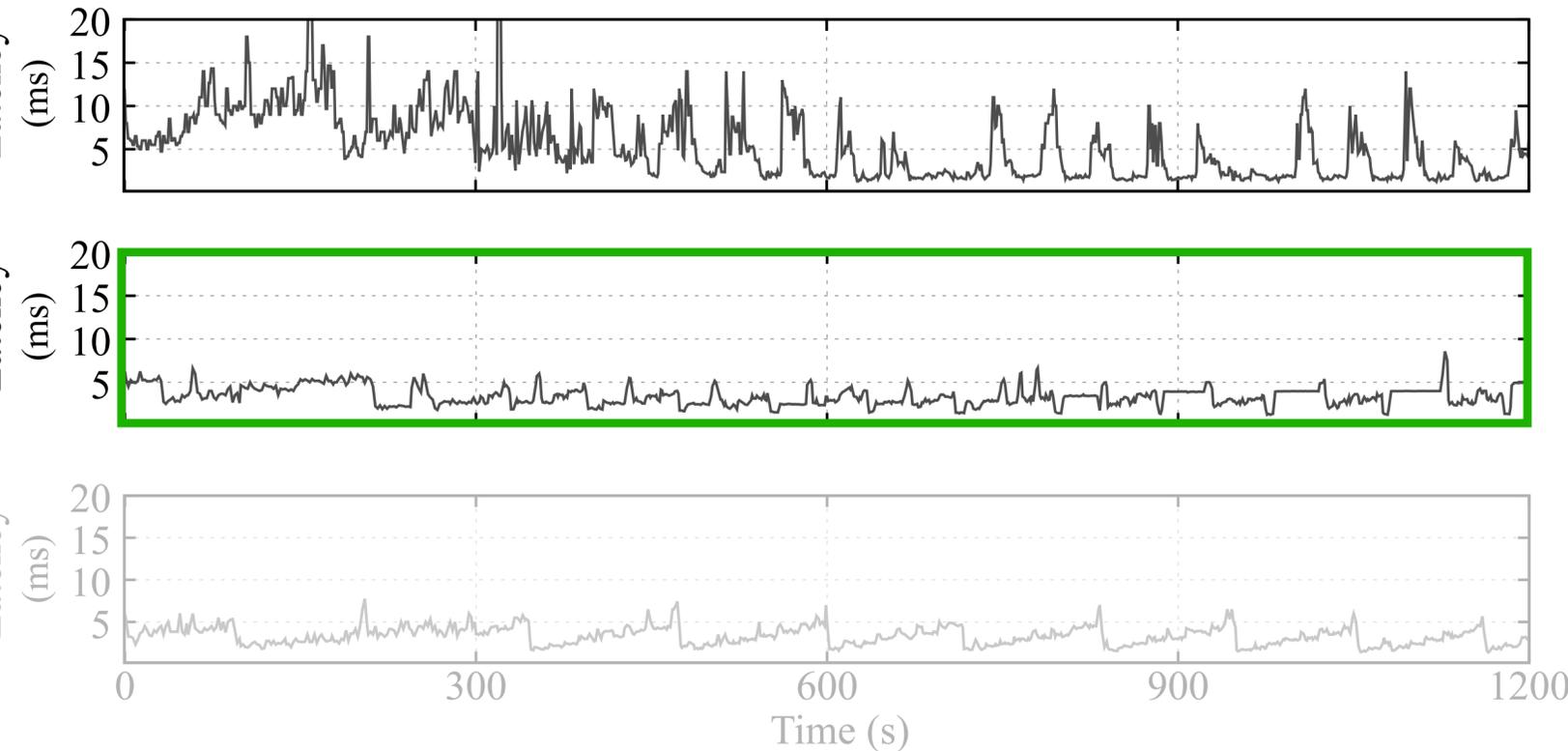
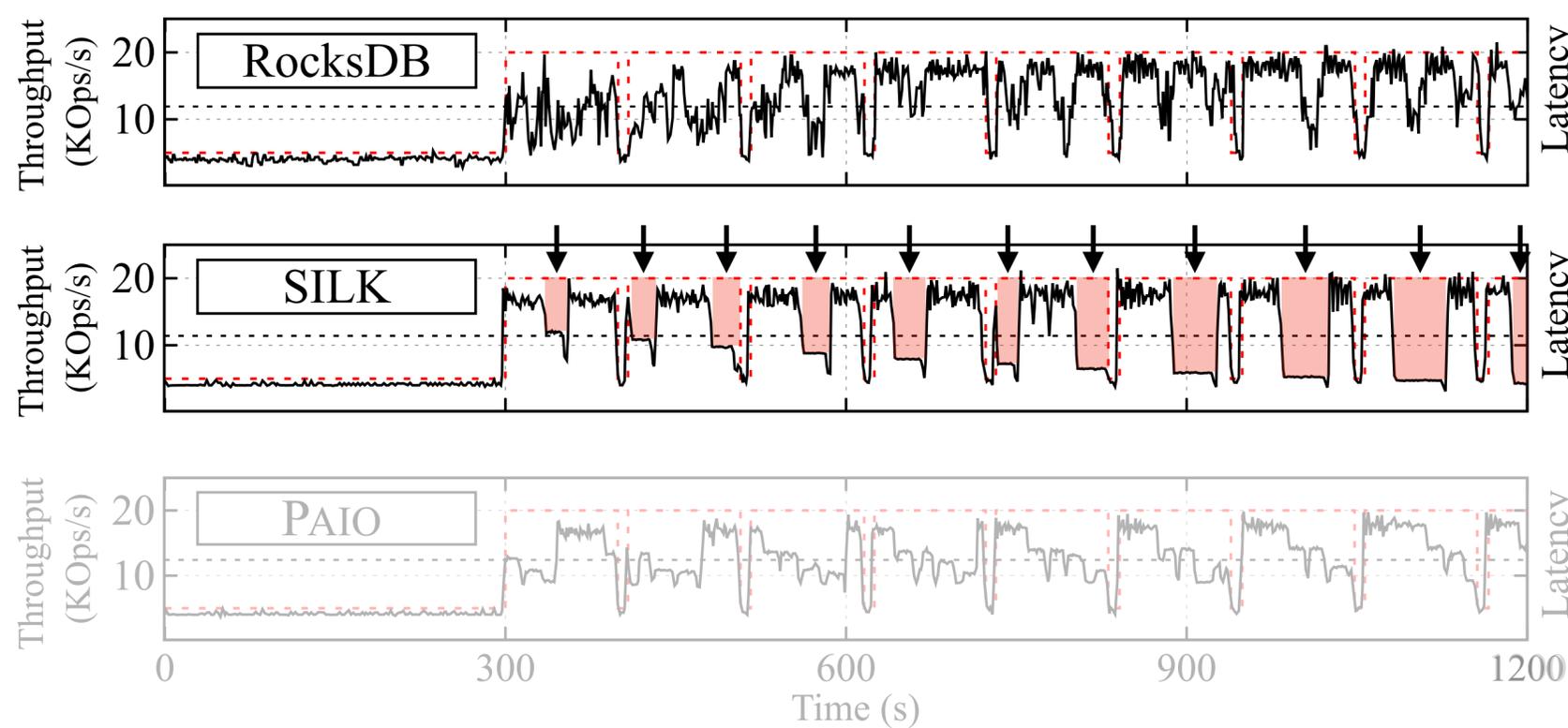


**Throughput:** high variability due to constant flushes and compactions

**99<sup>th</sup> latency:** high tail latency with peaks with an average range between 3 and 15 ms

# Mixture workload

50% read 50% write

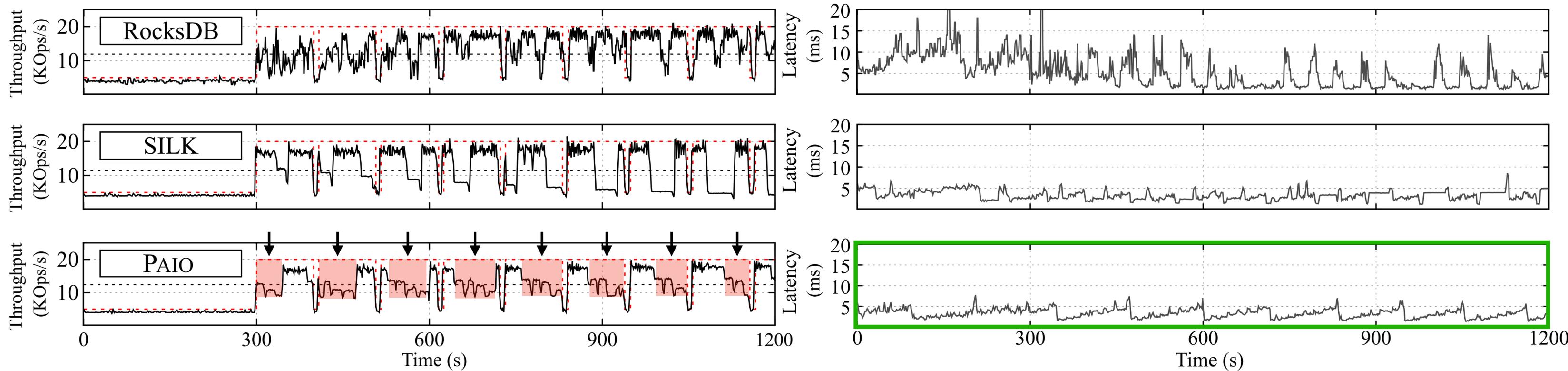


**Throughput:** suffers periodic throughput drops due to accumulated backlog

**99<sup>th</sup> latency:** low and sustained tail latency

# Mixture workload

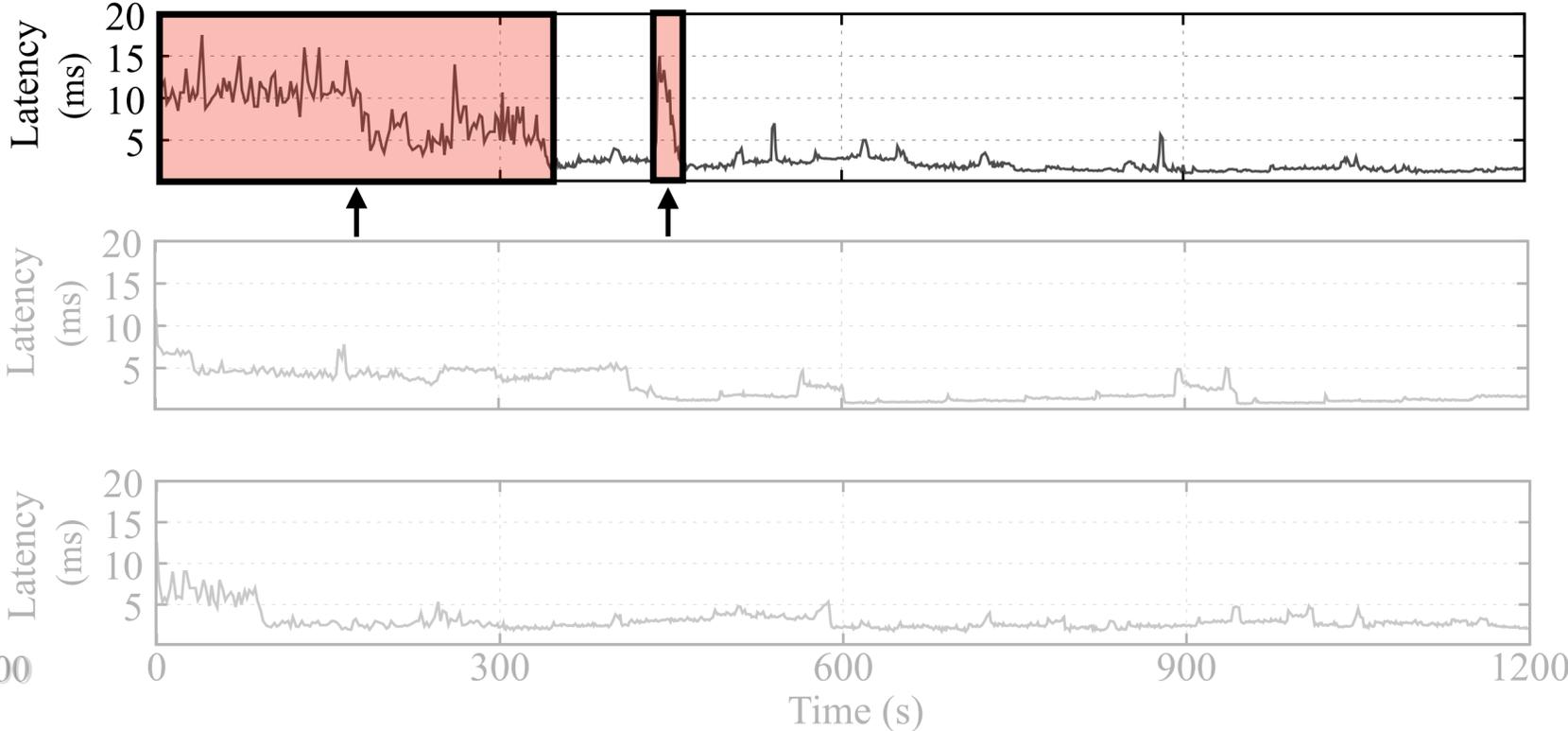
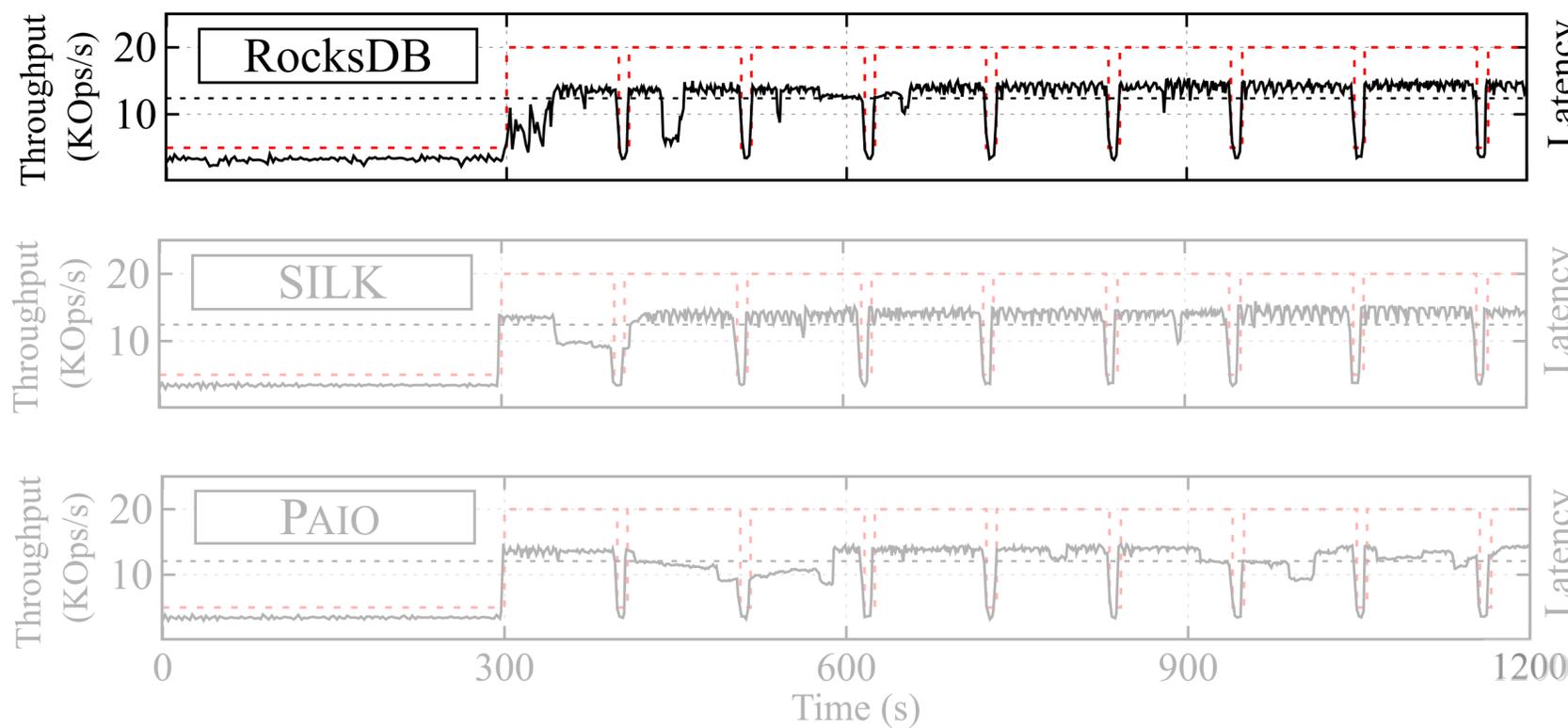
50% read 50% write



**PAIO and SILK observe a 4x decrease in absolute tail latency**

# Read-heavy workload

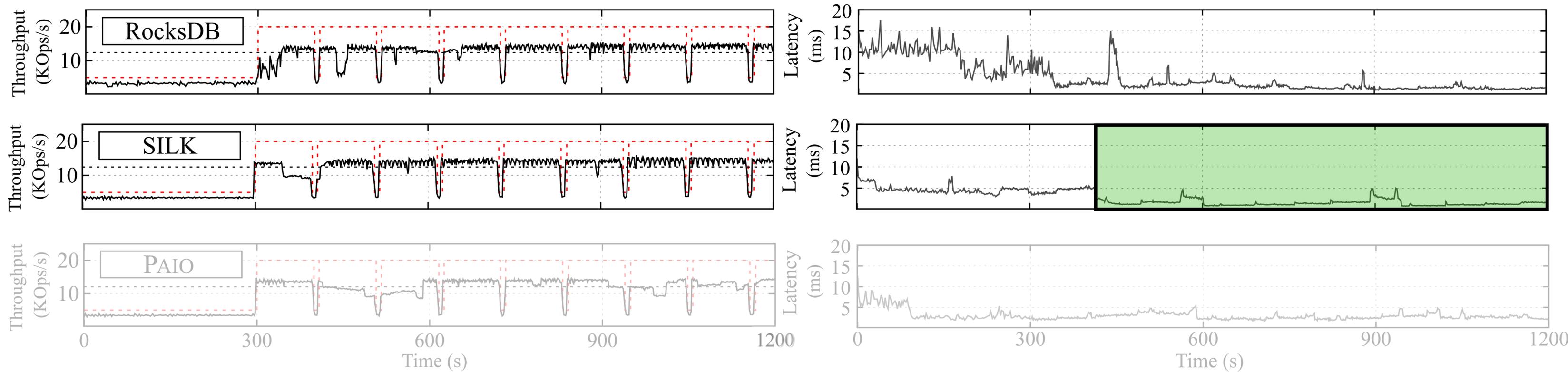
90% read 10% write



**99<sup>th</sup> latency:** temporary performance degradation due to accumulated backlog

# Read-heavy workload

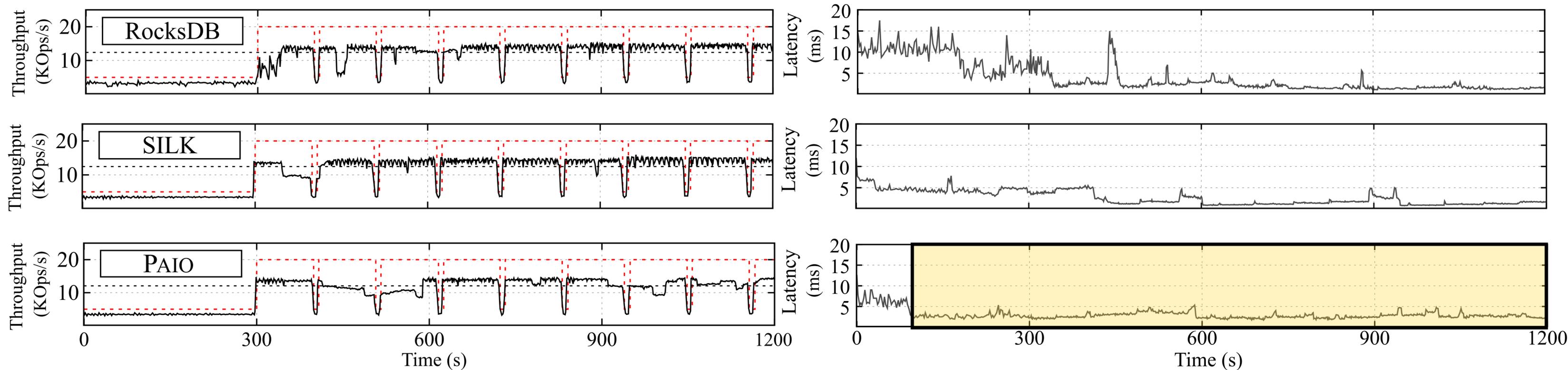
90% read 10% write



**99<sup>th</sup> latency:** after 400s, SILK preempts high level compactions, achieving a tail latency between 1-2ms

# Read-heavy workload

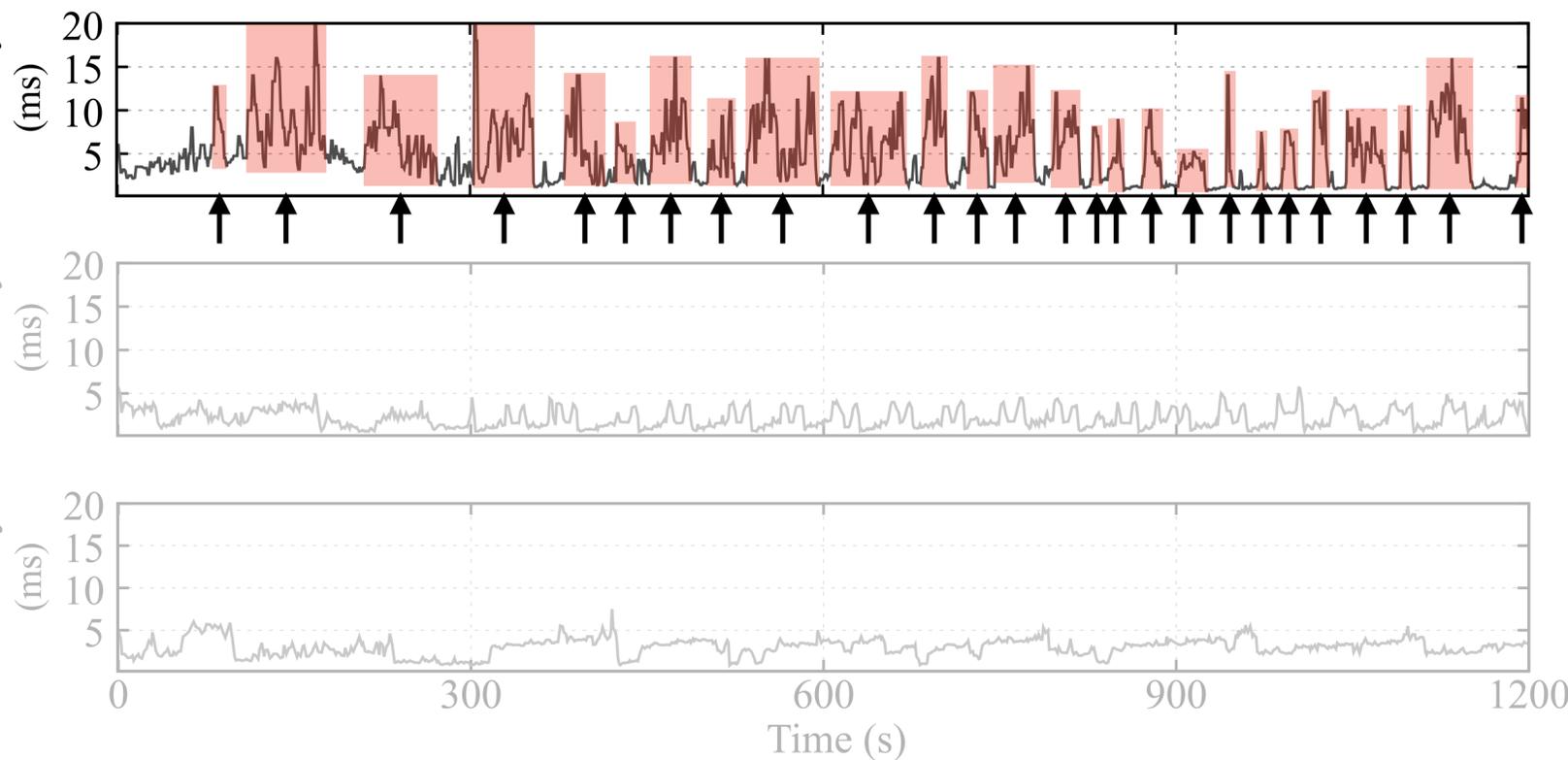
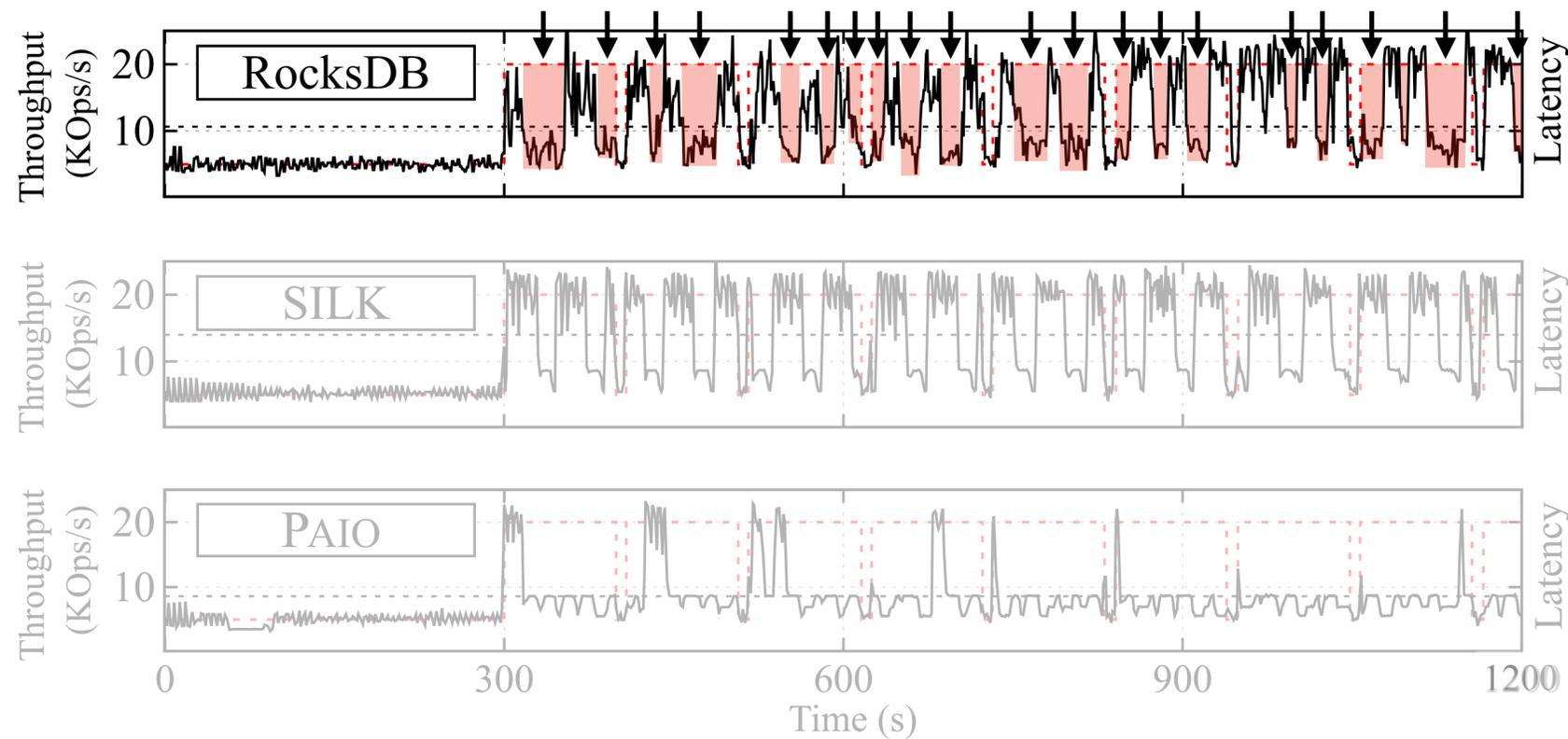
90% read 10% write



**Sustained tail latency but higher than SILK, due to not preempting compactions**

# Write-heavy workload

10% read 90% write

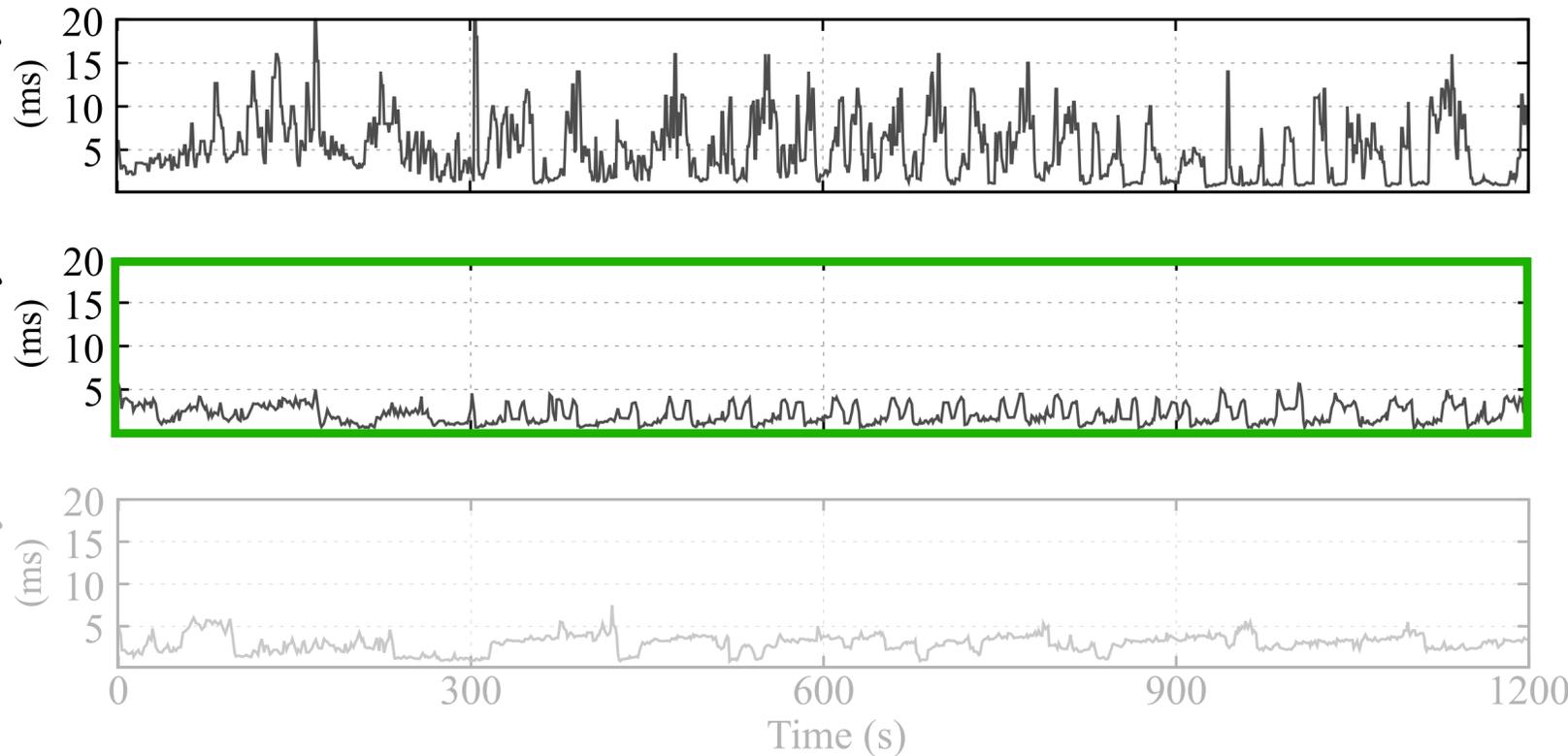
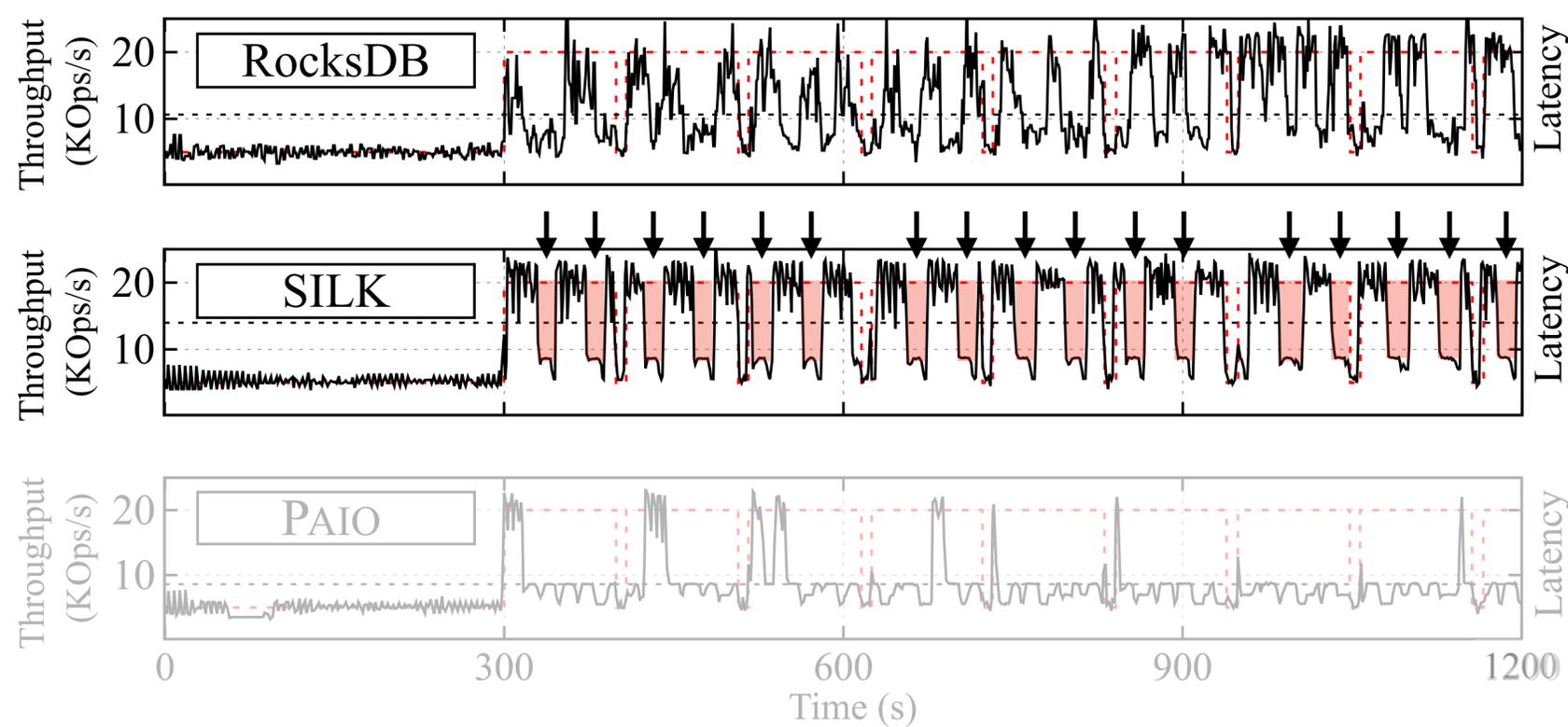


**Throughput:** large backlog of background tasks leads to high throughput variability

**99<sup>th</sup> latency:** high latency spikes throughout the entire execution

# Write-heavy workload

10% read 90% write

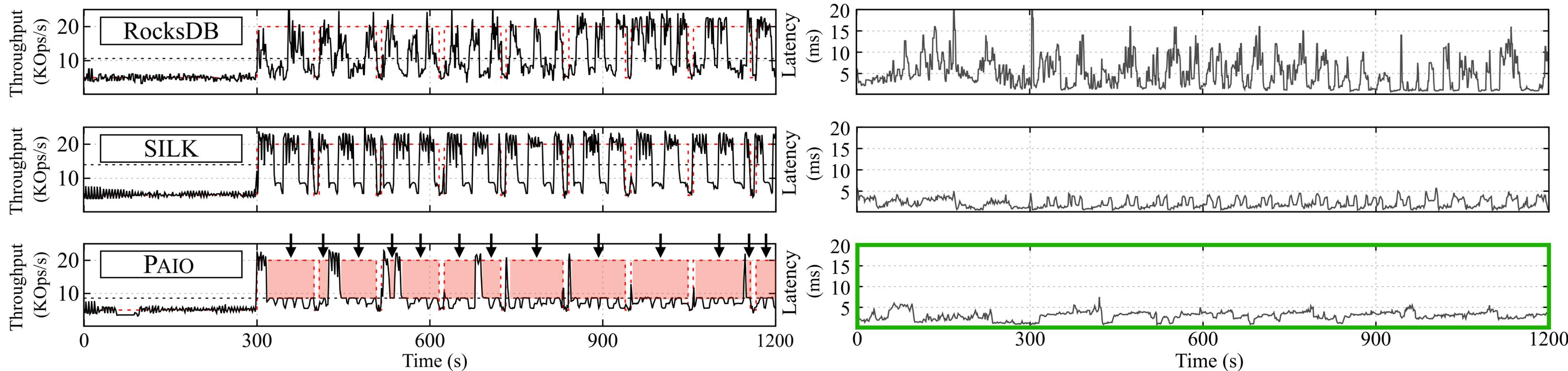


**Throughput:** suffers periodic throughput drops due to constant flushes

**99<sup>th</sup> latency:** SILK pauses high level compactions and only serves high priority operations

# Write-heavy workload

10% read 90% write



**Since flushes occur more frequently, PAIO slows down high level compactions more aggressively, temporarily halting low level ones**

# Summary

**PAIO**, a **user-level** framework that enables system designers to build *custom-made data plane stages*

- Combines ideas from **Software-Defined Storage** and **context propagation**

Decouples system-specific optimizations to dedicated **I/O layers**

## Data plane stages

- Tail latency control in LSM-based KVS (RocksDB)
- Per-application bandwidth control in shared storage settings (TensorFlow)

Enables similar **control** and **I/O performance** as system-specific optimizations

# Paper

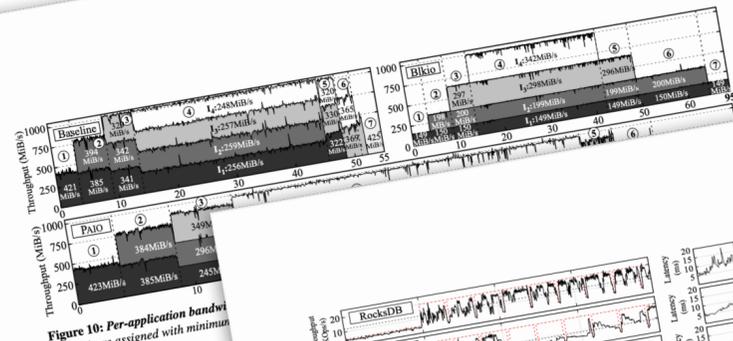
## Data plane stages built with PAIO

- Tail latency control in key-value stores ([RocksDB](#))
- Per-application bandwidth control ([TensorFlow](#))
- You can build your's too!

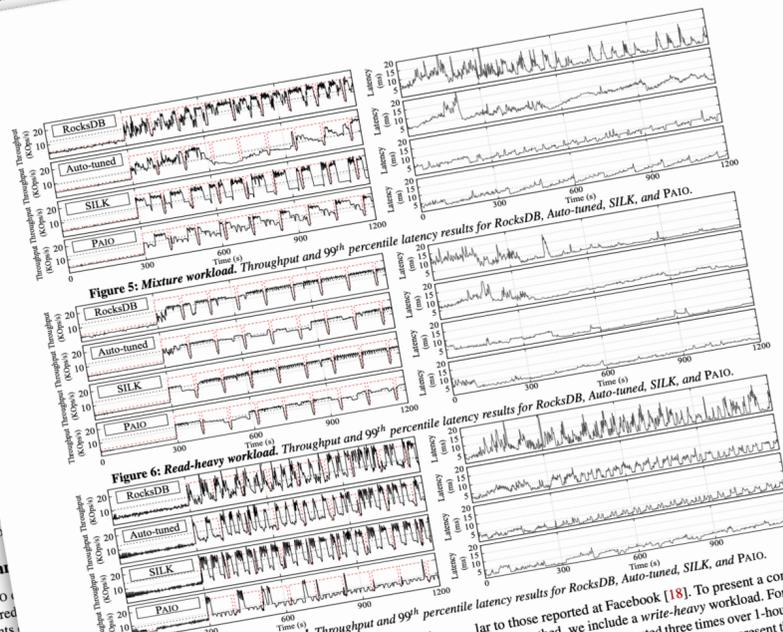
## Experiments

- Performance and scalability
- Profiling
- Mixture workload without rate limiting
- Per-application bandwidth results

PAIO is publicly available at [dsrhaslab/paio](https://github.com/dsrhaslab/paio)



**Figure 10: Per-application bandwidth control.** The results of the experiment show that PAIO can control the bandwidth of individual applications. The graph shows that PAIO can maintain the throughput of each application within a target range, even when the total system throughput is high. This is achieved by dynamically adjusting the bandwidth of each application based on its current usage and the overall system load.



**Figure 5: Mixture workload. Throughput and 99th percentile latency results for RocksDB, Auto-tuned, SILK, and PAIO.** The graph shows that PAIO achieves higher throughput and lower latency than the other systems. This is due to its ability to dynamically adjust the bandwidth of each application based on its current usage and the overall system load. PAIO also provides better tail latency control, which is important for key-value stores like RocksDB.

### 9.3 Per-Application Bandwidth Control

We now demonstrate how PAIO can guarantee a minimum bandwidth for each application under a shared bandwidth. The minimum bandwidth threshold for each application is set to 10MB/s. To simplify the results, internal operations like commit logging are turned off. All experiments are conducted using the `db_bench` benchmark [3]. As used in the SILK testbed [16], we limit memory usage to 1GiB and I/O bandwidth to 200MiB/s (unless stated otherwise).

**Workloads.** We focus on workloads made of bursty clients, to better simulate existing services in production [16, 18]. Client requests are issued in a closed loop through 300 seconds submits peaks and valleys. An initial valley of 300 seconds of KVS operations at 5kops/s, and is used for executing the KVS internal backlog. Peaks are issued at a rate of 20kops/s. All 100 seconds, followed by 10 seconds valleys at 5kops/s. Using datastores were preloaded with 100M key-value pairs, using a uniform key-distribution, 8B keys and 1024B values. We use three workloads with different read:write ratios: *mixture* (50:50), *read-heavy* (90:10), and *write-heavy* (10:90). *Mixture* represents a commonly used YCSB workload (workload A) and provides a similar ratio as Nutanix production workloads [16]. *Read-heavy* provides an operation ratio similar to those reported at Facebook [18].

To present a comprehensive testbed, we include a *write-heavy* workload. For each system, workloads were executed three times over 1-hour with 20 minutes of a single run. Similar performance curves were observed for the rest of the execution. Fig. 5-9 depict throughput and 99th percentile latency of all systems and workloads. Theoretical client load is presented as a red dashed line. Mean throughput is shown as an horizontal dashed line.

**Mixture workload (Fig. 5).** Due to accumulated backlog of the loading phase, the throughput achieved in all systems does not match the theoretical client load. RocksDB presents high tail latency spikes due to constant flushes and low level compactions. Auto-tuned presents less latency spikes but degrades overall throughput. This is due to the rate limiter being agnostic of background tasks' priority, and because it increases its rate when there is more backlog, contending for disk bandwidth. SILK achieves low tail latency but suffers periodic drops in throughput due to accumulated backlog. Compared to RocksDB (11.9 kops/s), PAIO provides similar

# PAIO: General, Portable I/O Optimizations with Minor Application Modifications

Ricardo Macedo<sup>1</sup>, Yusuke Tanimura<sup>2</sup>, Jason Haga<sup>2</sup>, Vijay Chidambaram<sup>3</sup>,  
José Pereira<sup>1</sup>, João Paulo<sup>1</sup>

<sup>1</sup> INESC TEC and University of Minho, <sup>2</sup> AIST, <sup>3</sup> UTAustin and VMware Research

