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

Ricardo Macedo[1], Yusuke Tanimura[2], Jason Haga[2], Vijay Chidambaram[3], José Pereira[1], João Paulo[1]

[1] INESC TEC and University of Minho, [2] AIST, [3] UTAustin and VMware Research

INESCTEC Universidade do Minho AIST TEXAS The University of Texas at Austin vmware

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

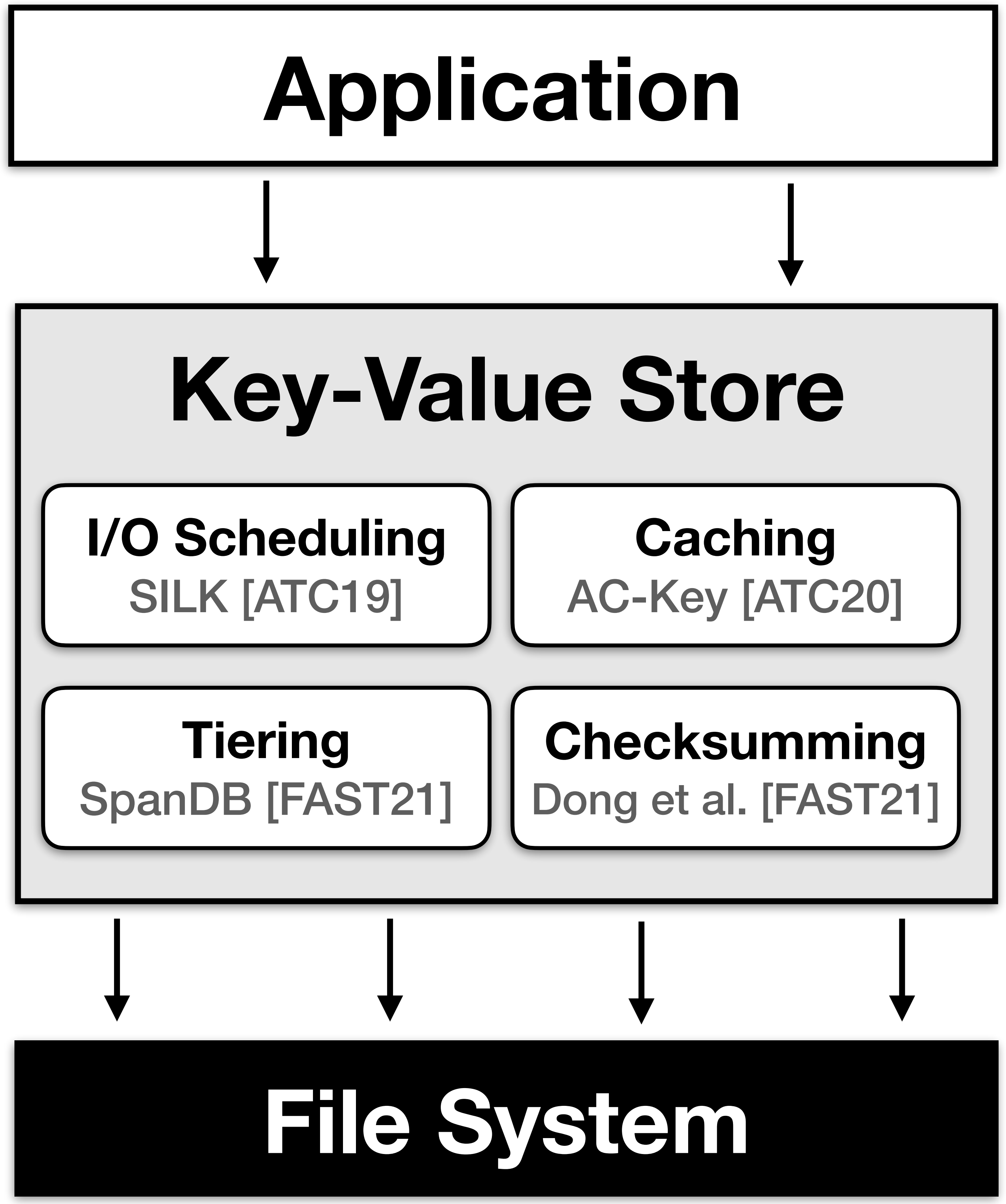
- Data-centric systems have become an integral part of modern I/O stacks
- Good pe
  optimiza
  - Sched
- Optimizations are implemented in sub-optimal manner

There is a better way to implement I/O optimizations

Challenge #1

Tightly coupled optimizations

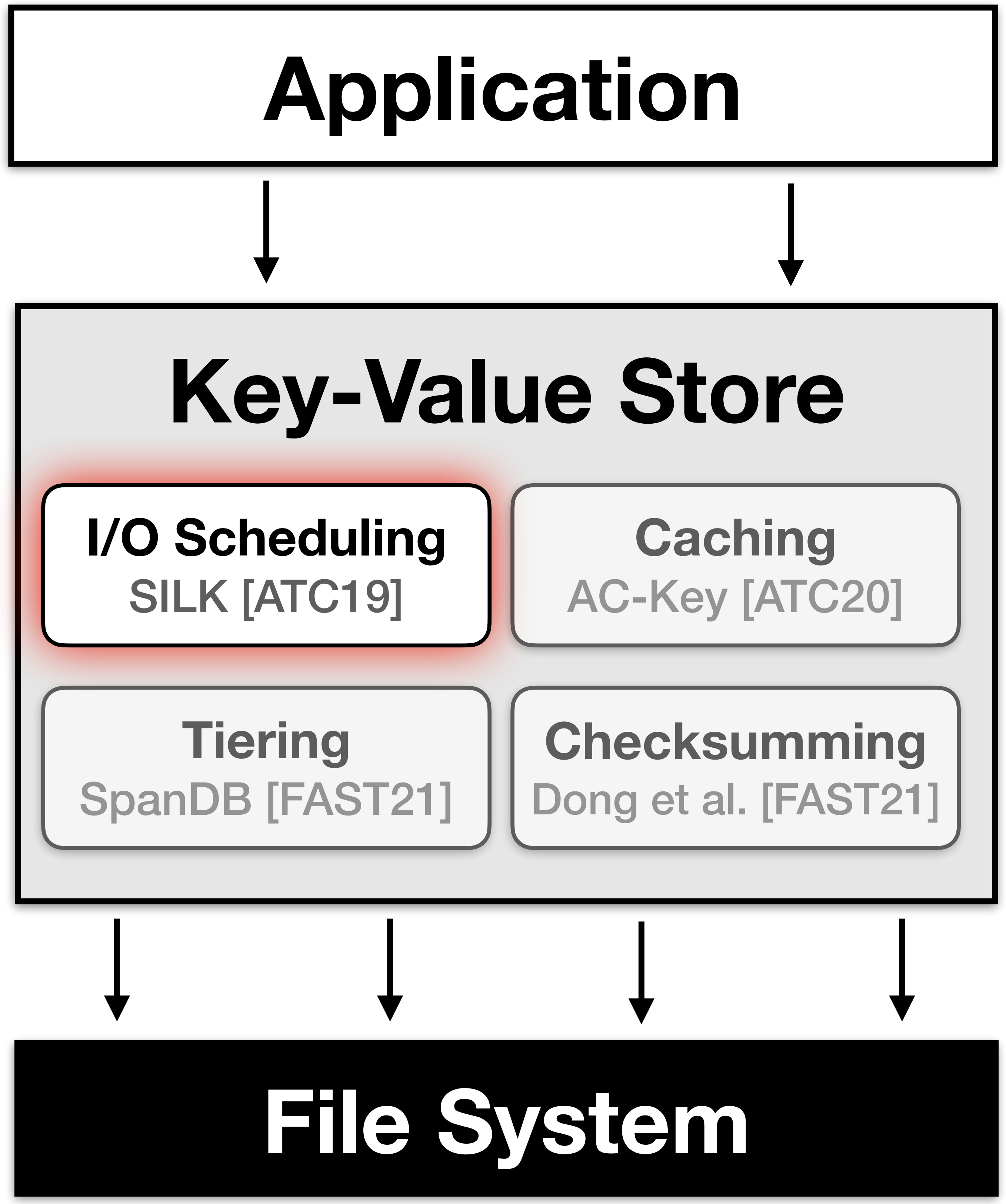
- I/O optimizations are single purposed
- Require deep understanding of the system's internal operation model
- Require profound system refactoring
- Limited portability across systems



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



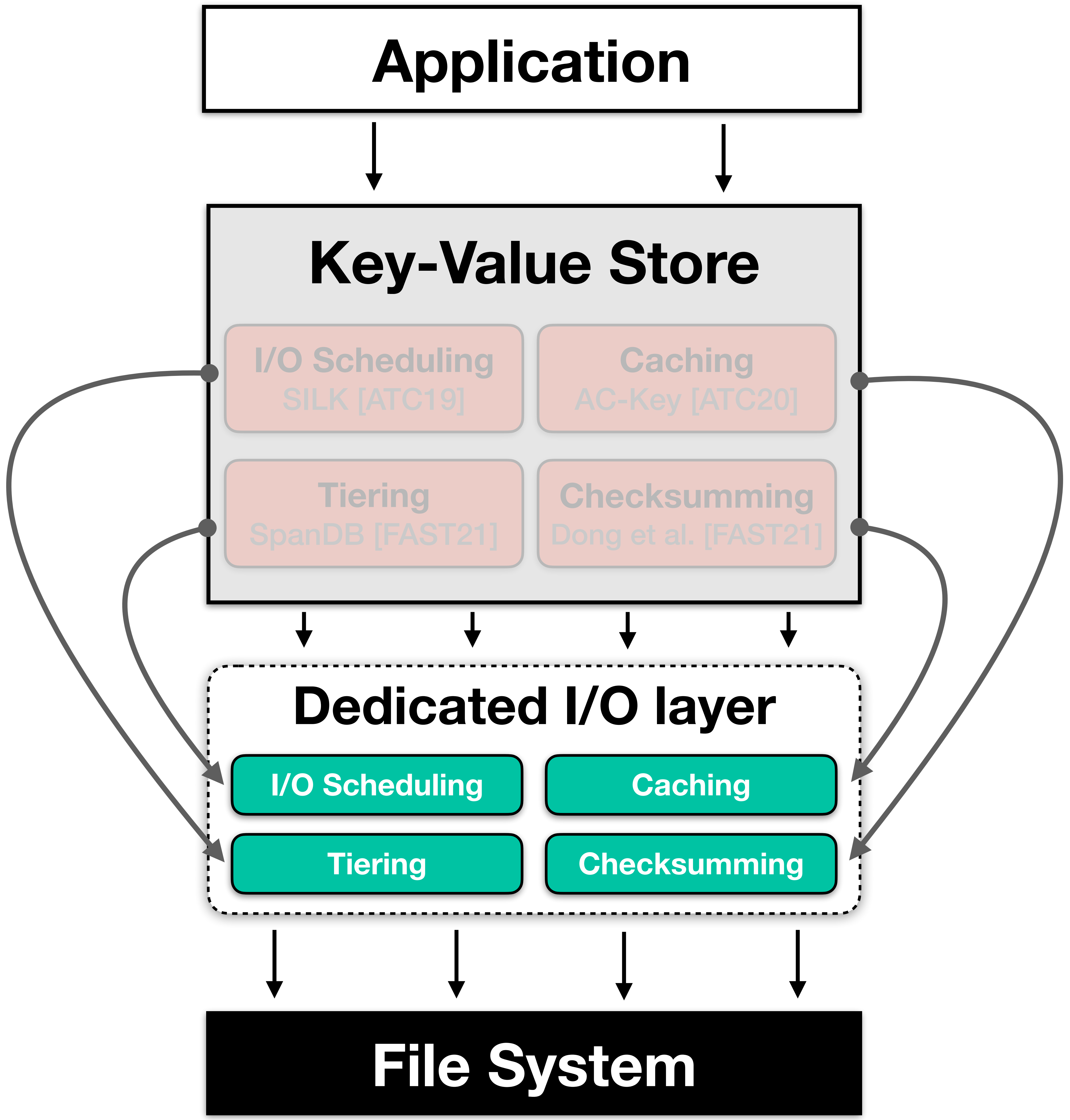
SILK's I/O Scheduler

- 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

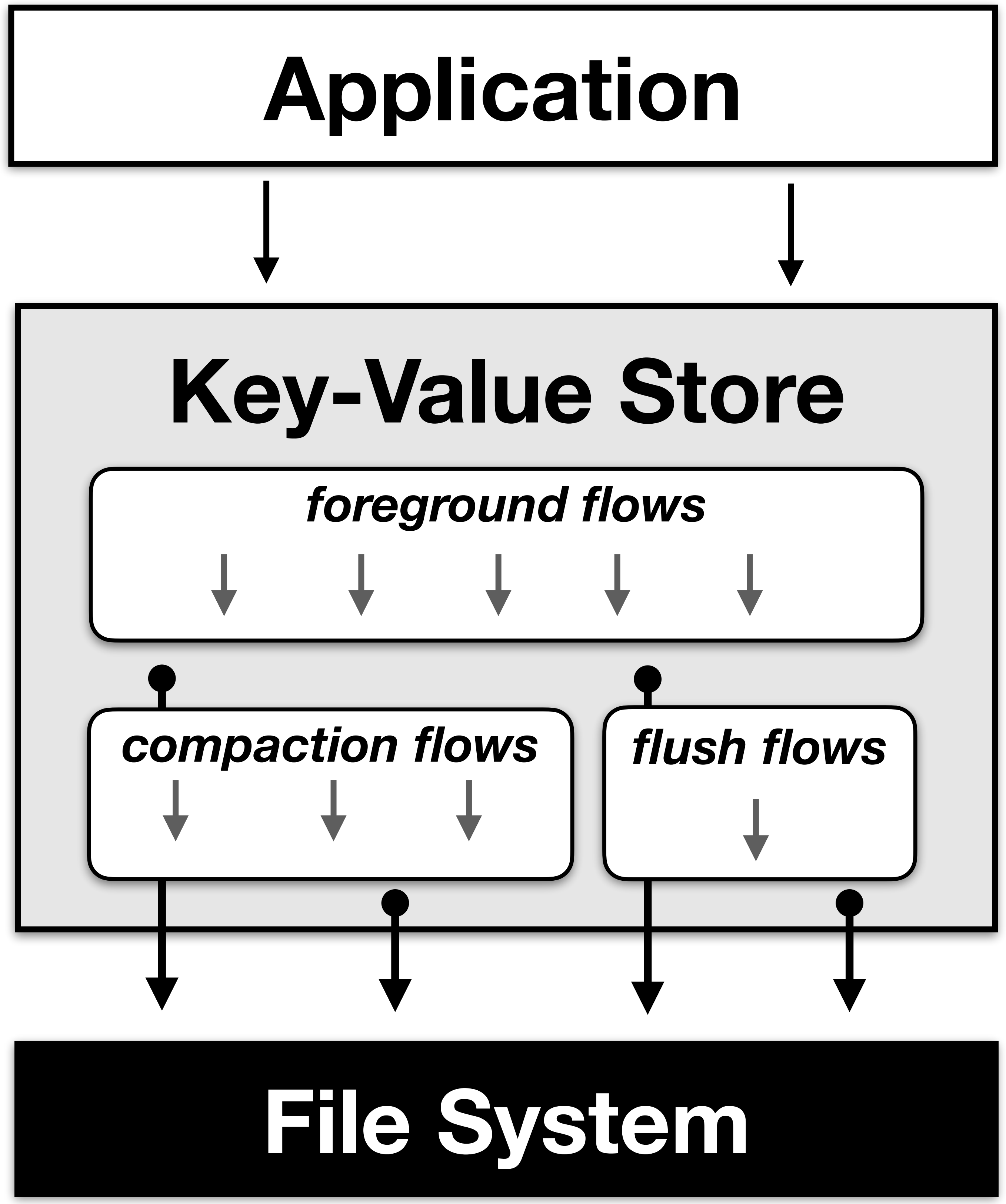
- I/O optimizations should be disaggregated from the internal logic
- Moved to a dedicated I/O layer
- Generally applicable
- Portable across different scenarios



Challenge #2

Rigid interfaces

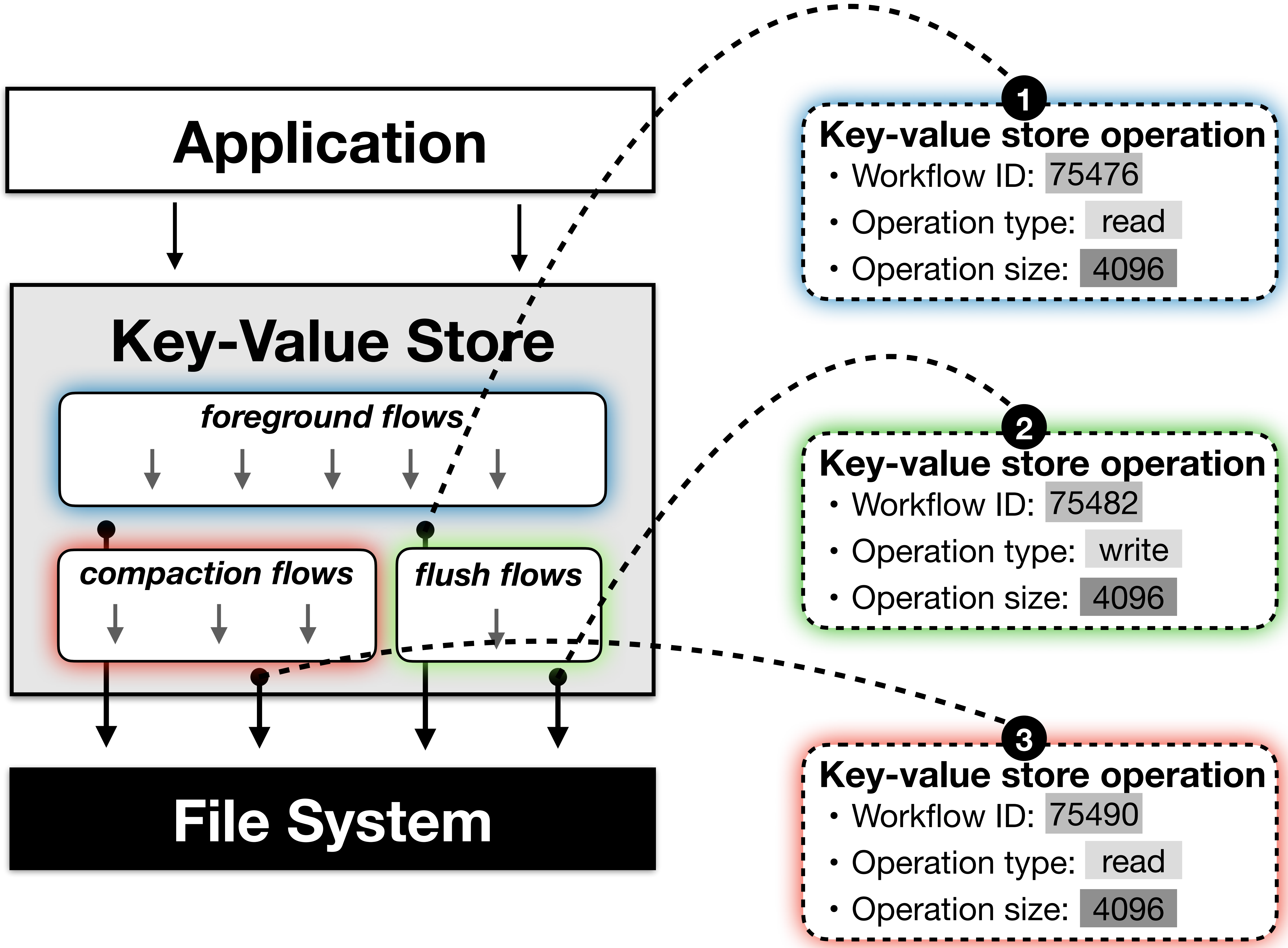
- Decoupled optimizations lose granularity and internal application knowledge
- I/O layers communicate through rigid interfaces
- Discard information that could be used to classify and differentiate requests



Challenge #2

❌ Rigid interfaces

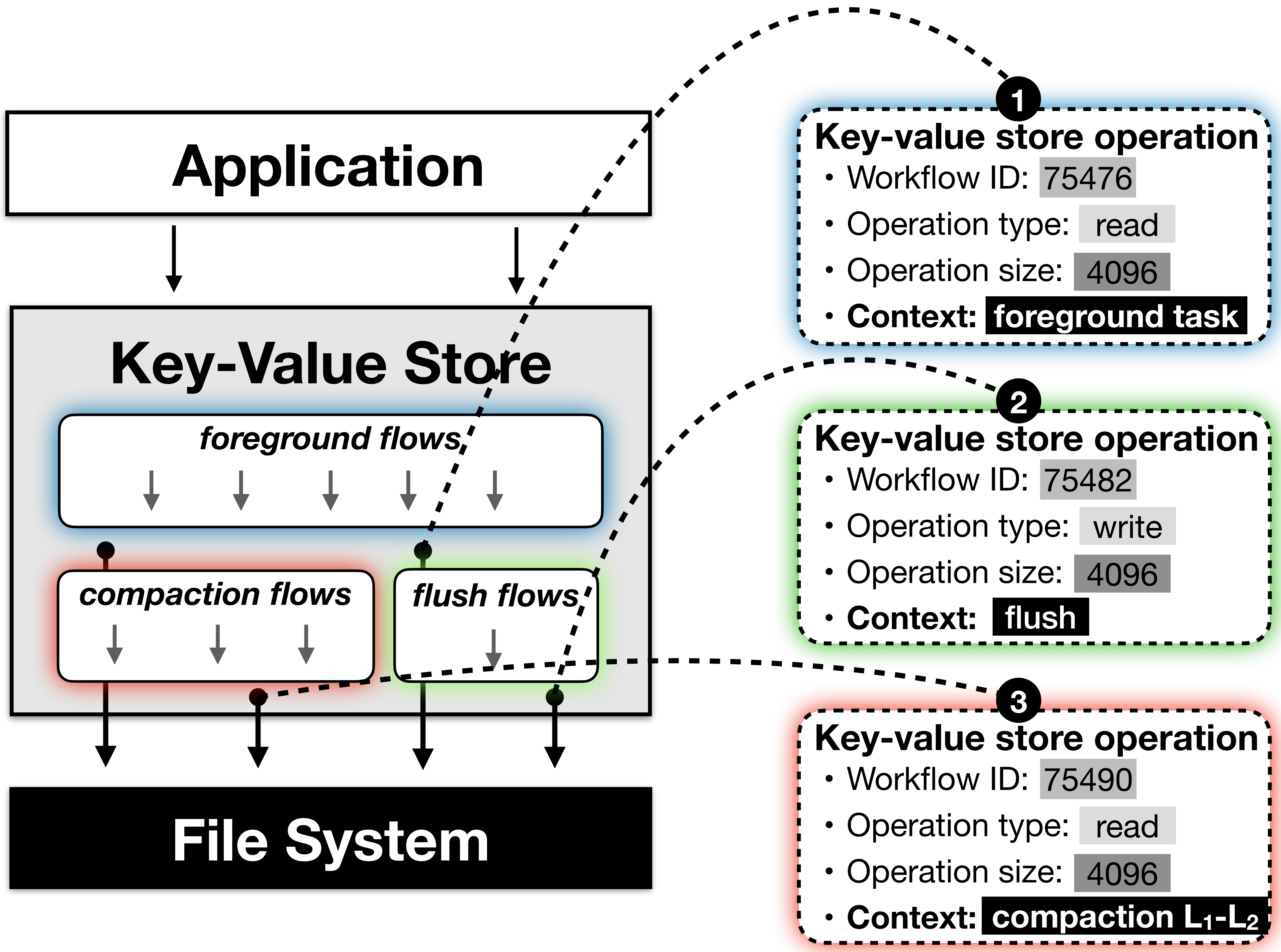
- Decoupled optimizations lose granularity and internal application knowledge
- I/O layers communicate through rigid interfaces
- Discard information that could be used to classify and differentiate requests



Challenge #2

✅ Information propagation

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



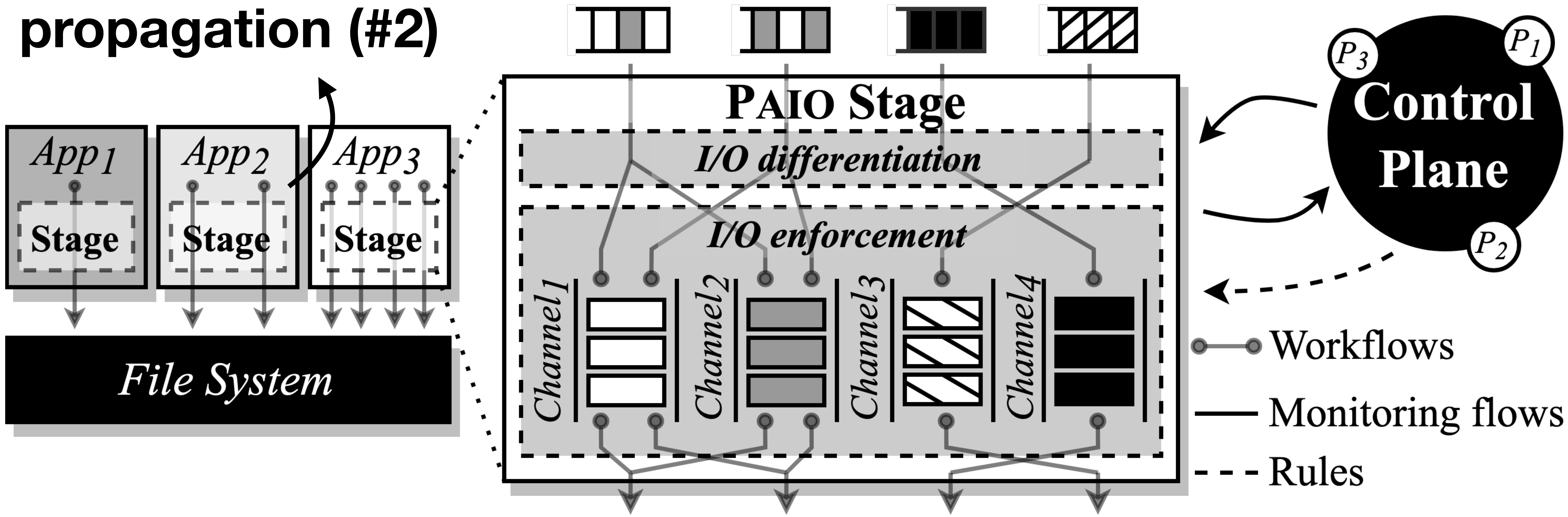
PAIO

- **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

Global visibility (#4)

Information propagation (#2)

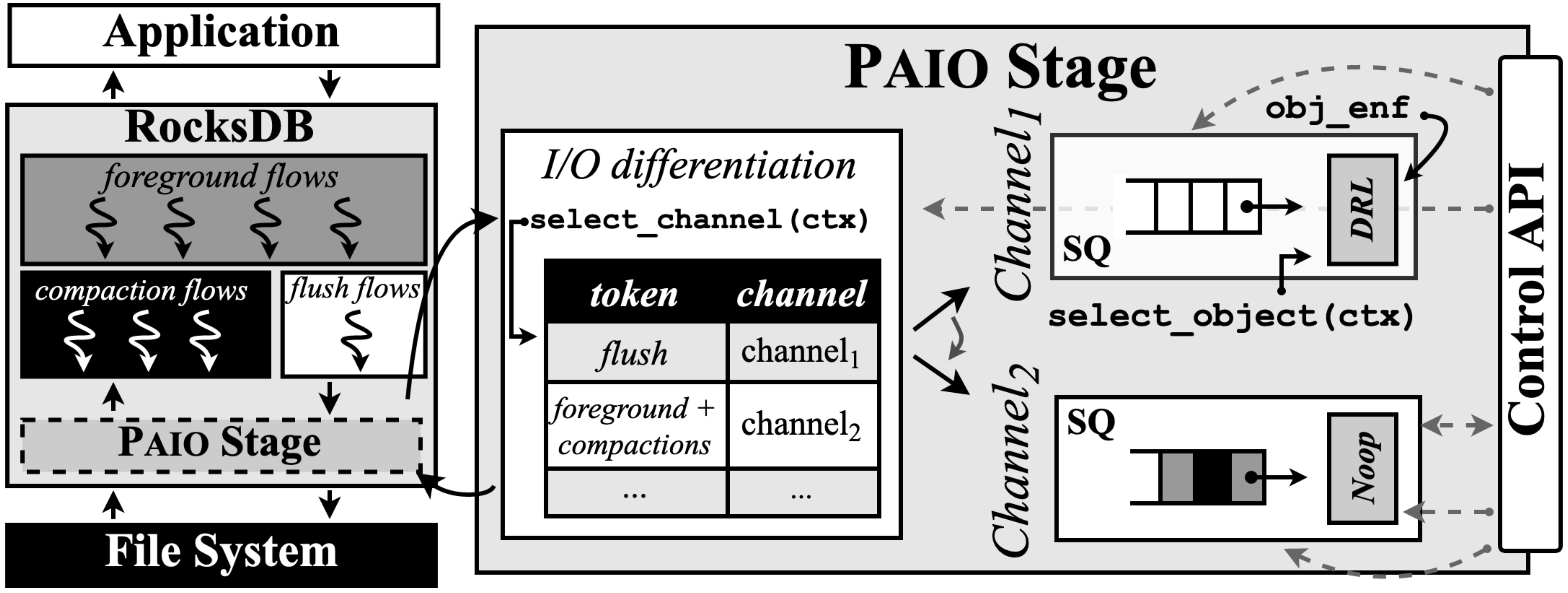


User-level (#3)

Dedicated I/O layer (#1)

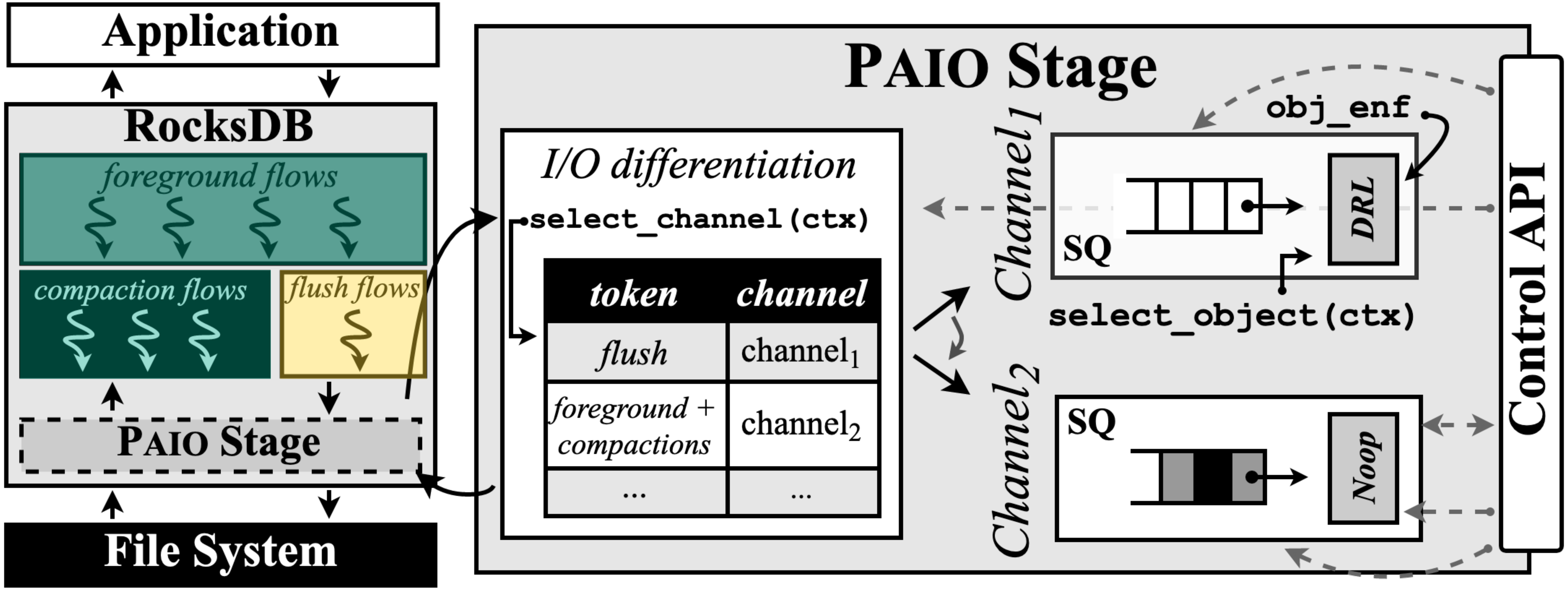
PAIO design

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



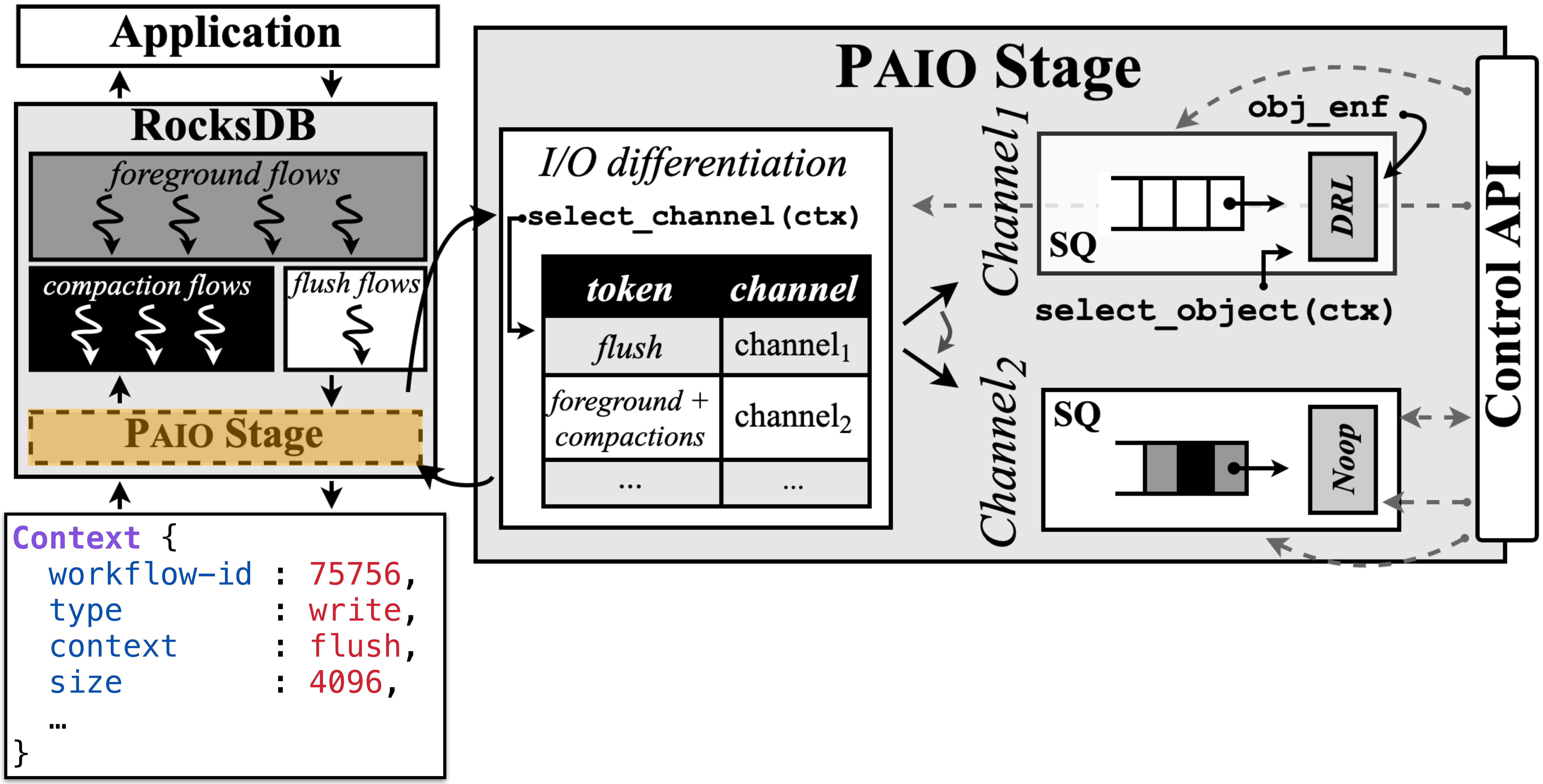
Policy: *limit the rate of RocksDB's flush operations to X MiB/s*

I/O differentiation

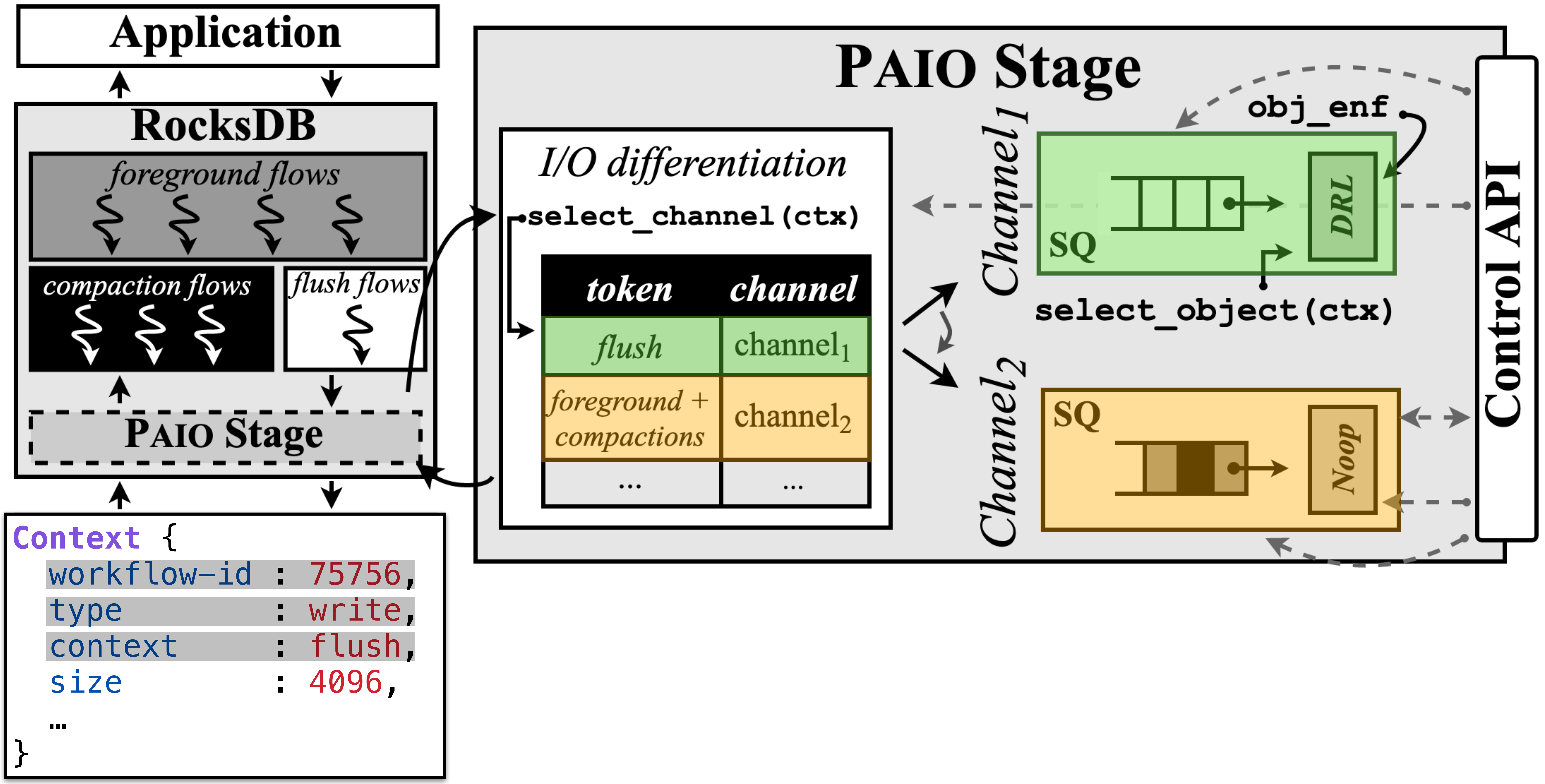


Identify the origin of POSIX operations (i.e., **foreground**, **compaction**, or **flush** operations)

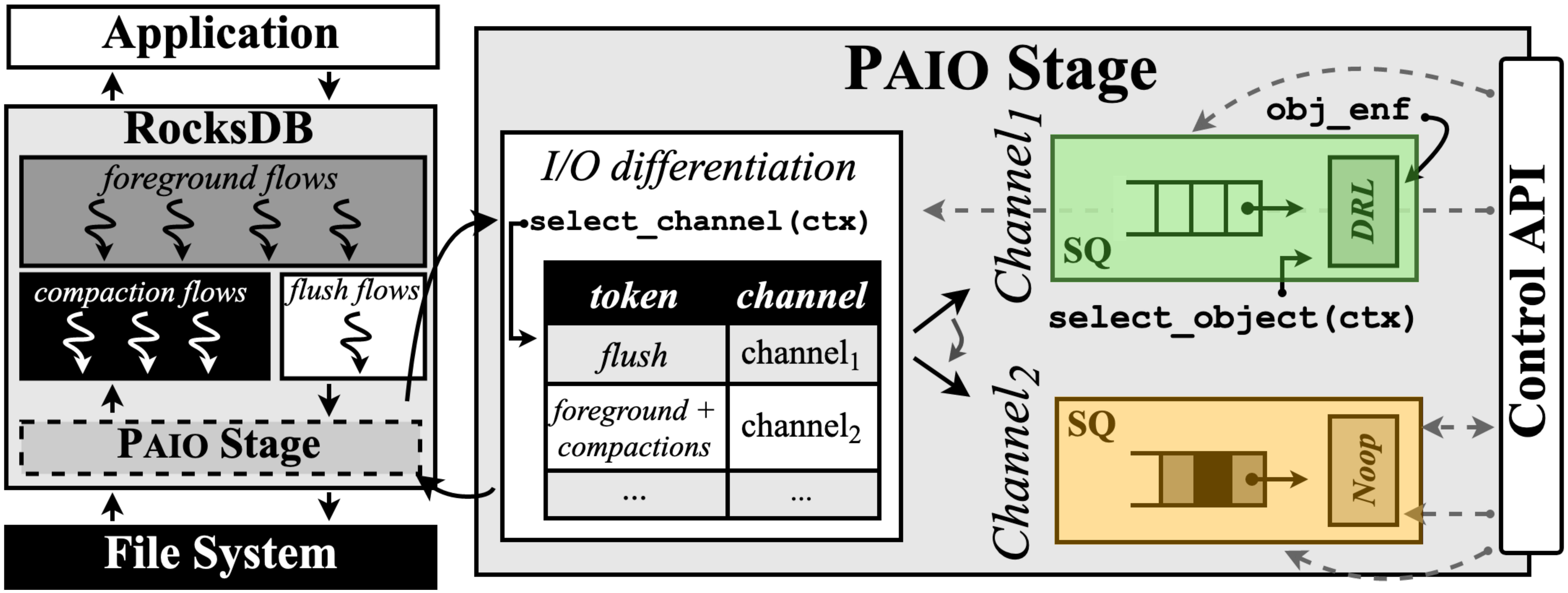
I/O differentiation



I/O differentiation

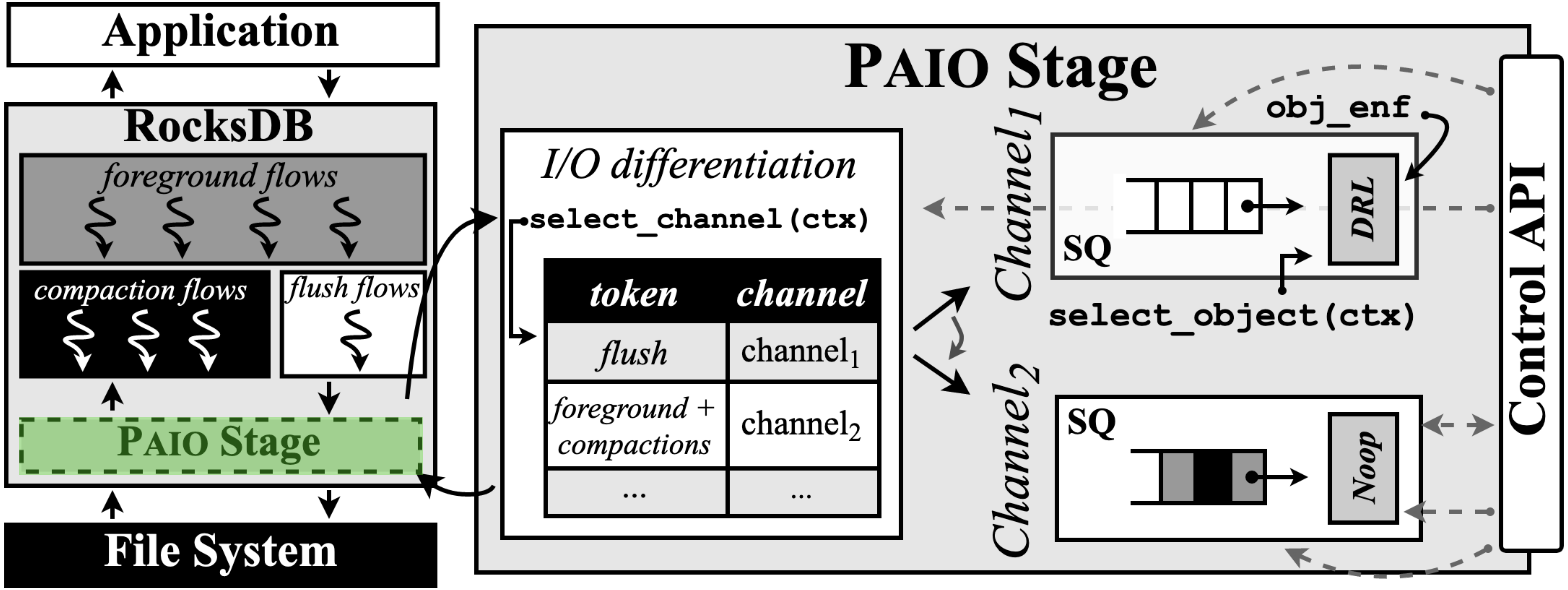


I/O enforcement



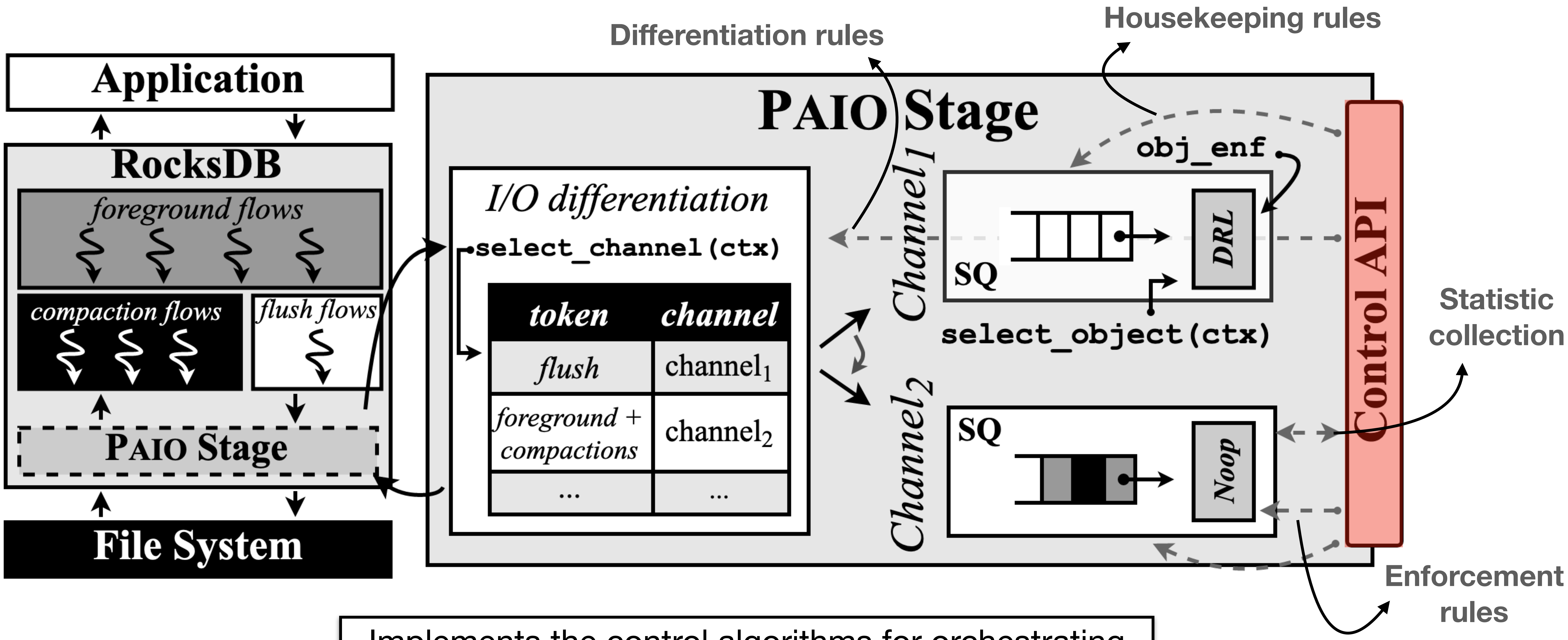
PAIO currently supports **Noop** and **DRL** enforcement objects

I/O enforcement



Requests return to their original I/O path

Control plane interaction



Implements the control algorithms for orchestrating stages (e.g., tail latency control, per-application bandwidth guarantees)

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

99th 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

99th 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

99th latency: temporary performance degradation due to accumulated backlog

Read-heavy workload

90% read 10% write

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

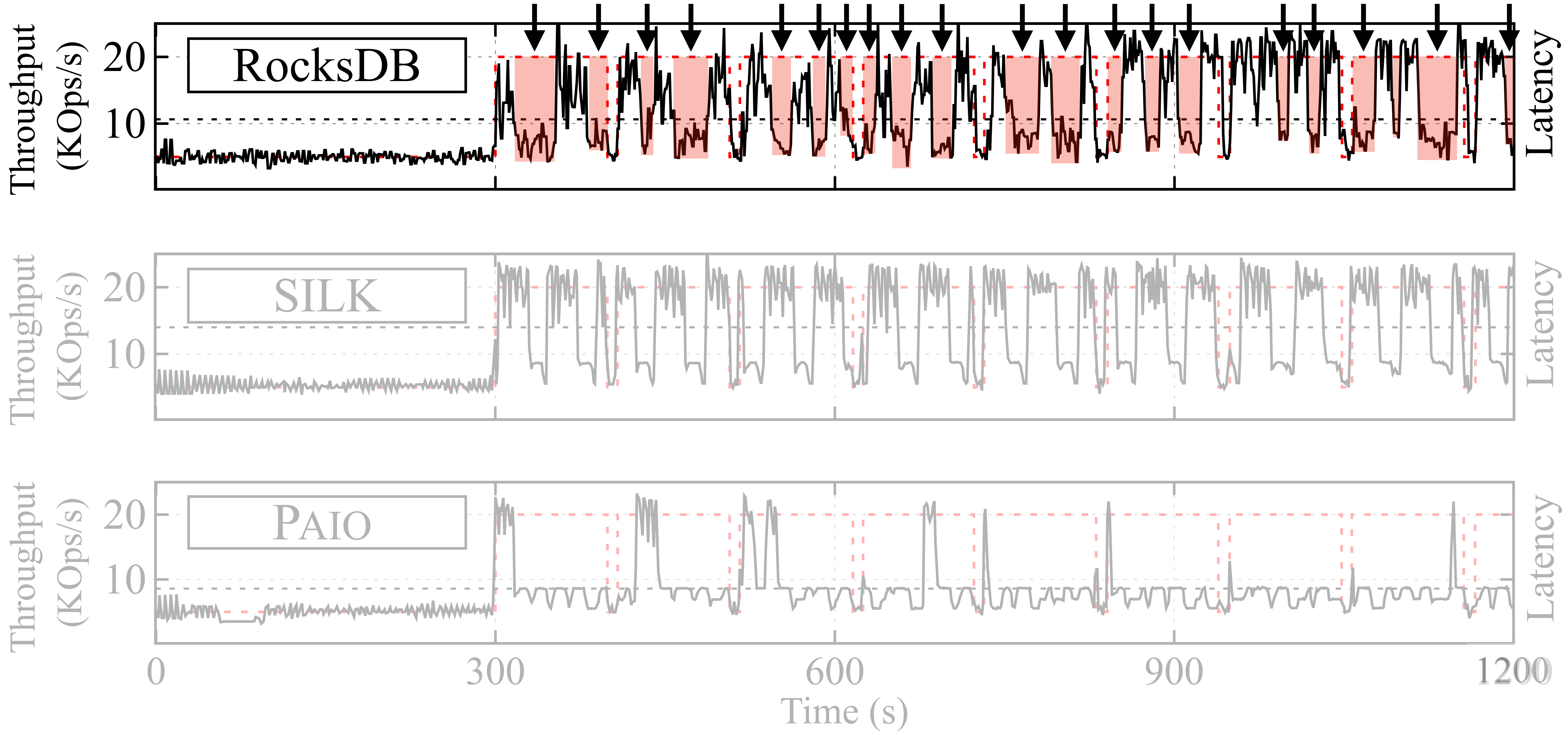
Read-heavy workload

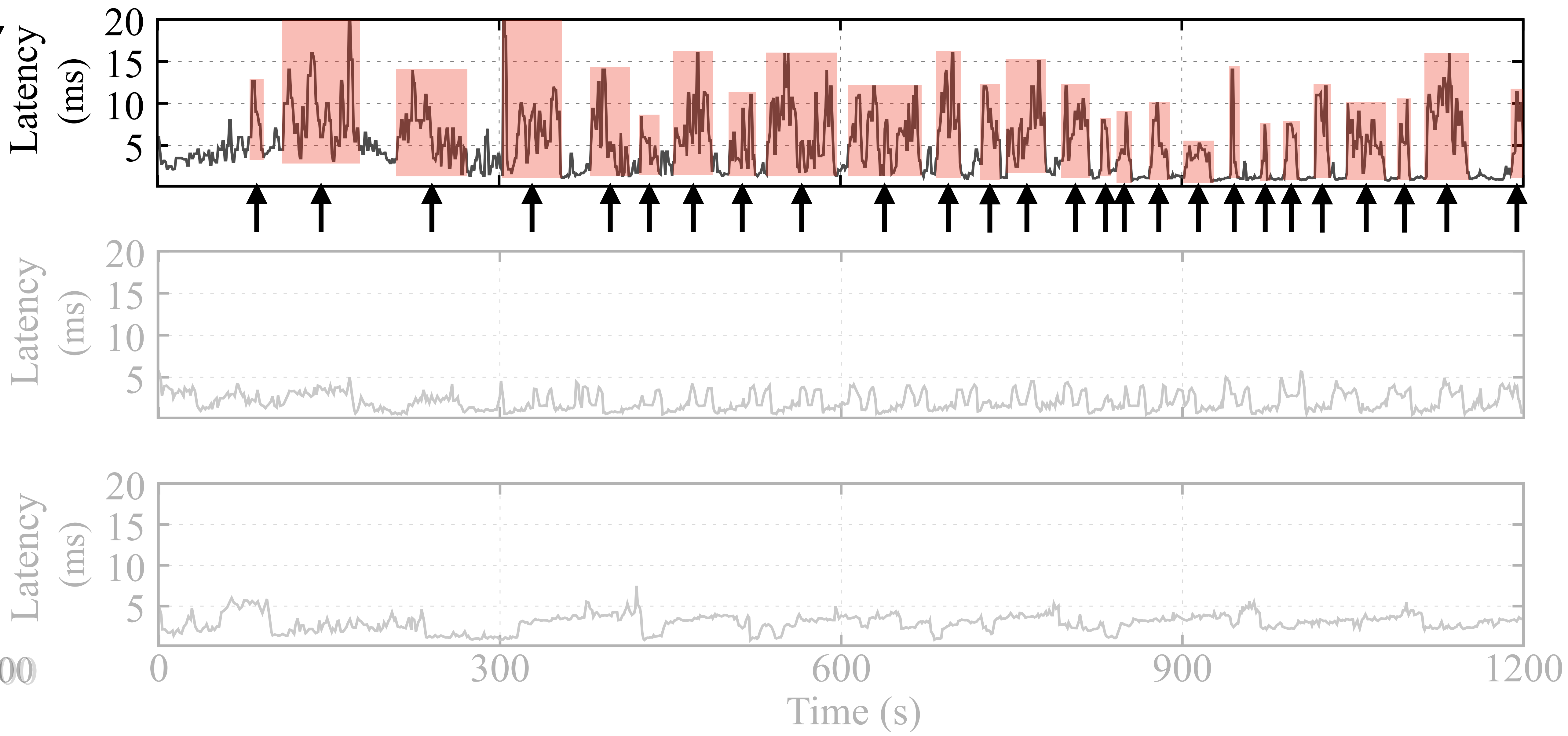
90% read 10% write

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

99th latency: high latency spikes throughout the entire execution

Write-heavy workload
10% read 90% write

Throughput: suffers periodic throughput drops due to constant flushes

99th 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**

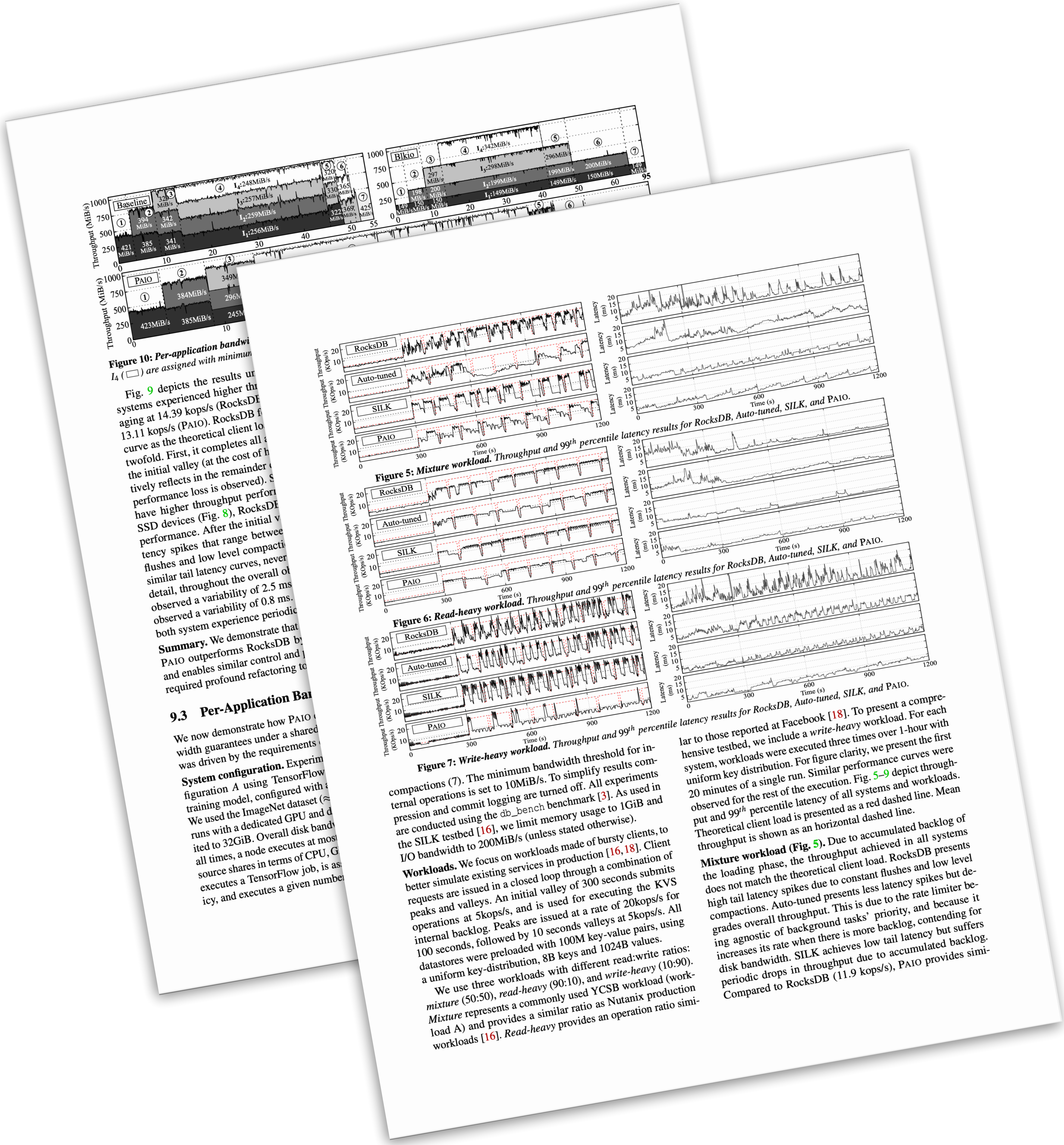
Decouples system-specific optimizations to dedicated **I/O layers**

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

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

Ricardo Macedo[1], Yusuke Tanimura[2], Jason Haga[2], Vijay Chidambaram[3], José Pereira[1], João Paulo[1]

[1] INESC TEC and University of Minho, [2] AIST, [3] UTAustin and VMware Research