Encontro Nacional de Sistemas Distribuídos

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

20th USENIX Conference in File and Storage Technologies
(USENIX FAST 2022)

Ricardo Macedo
INESC TEC & University of Minho

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

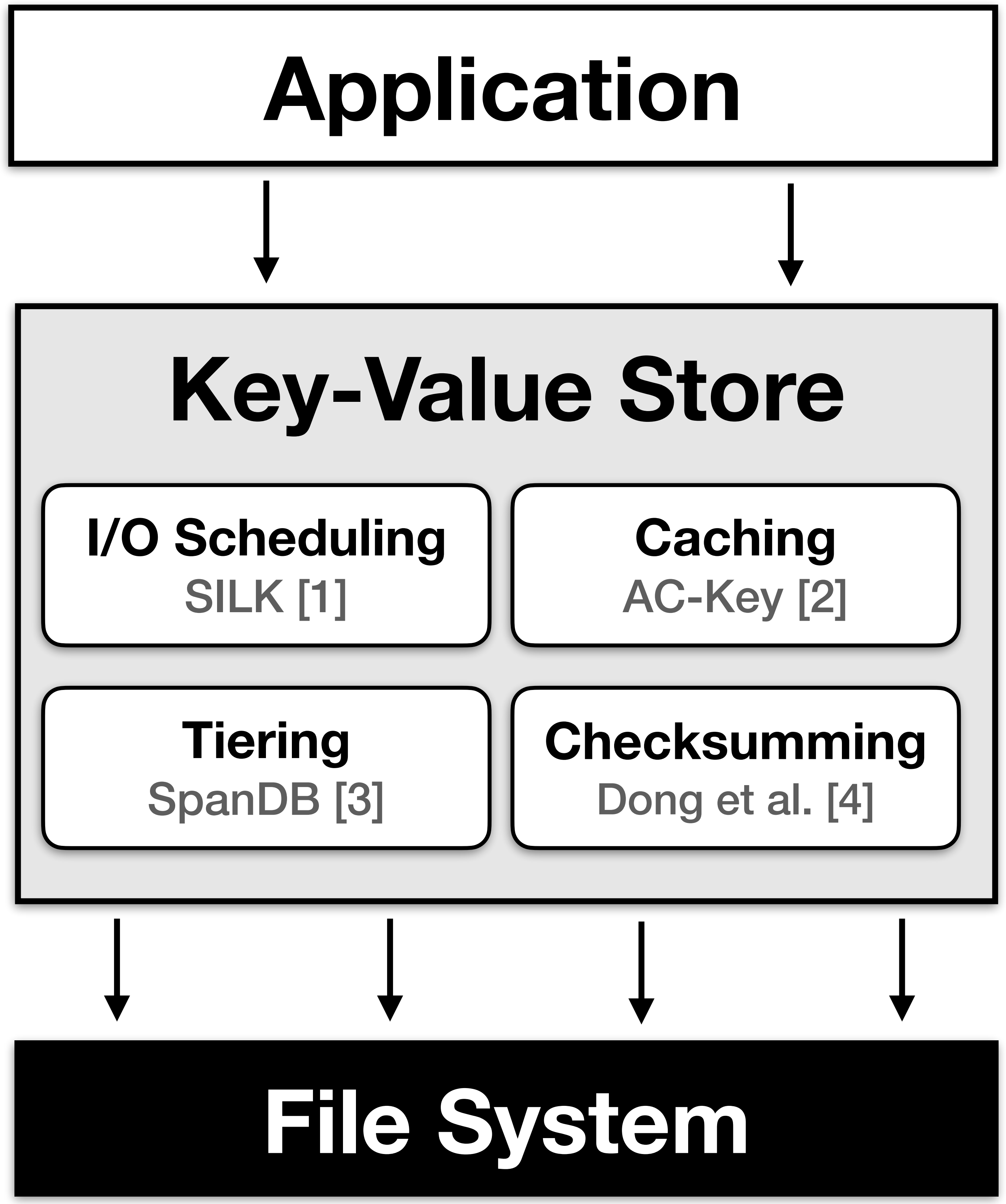
- Data-centric systems have become an integral part of modern I/O stacks
- Good pe
  optimiza
  - **Sche**
- Optimizations are implemented in sub-optimal manner

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

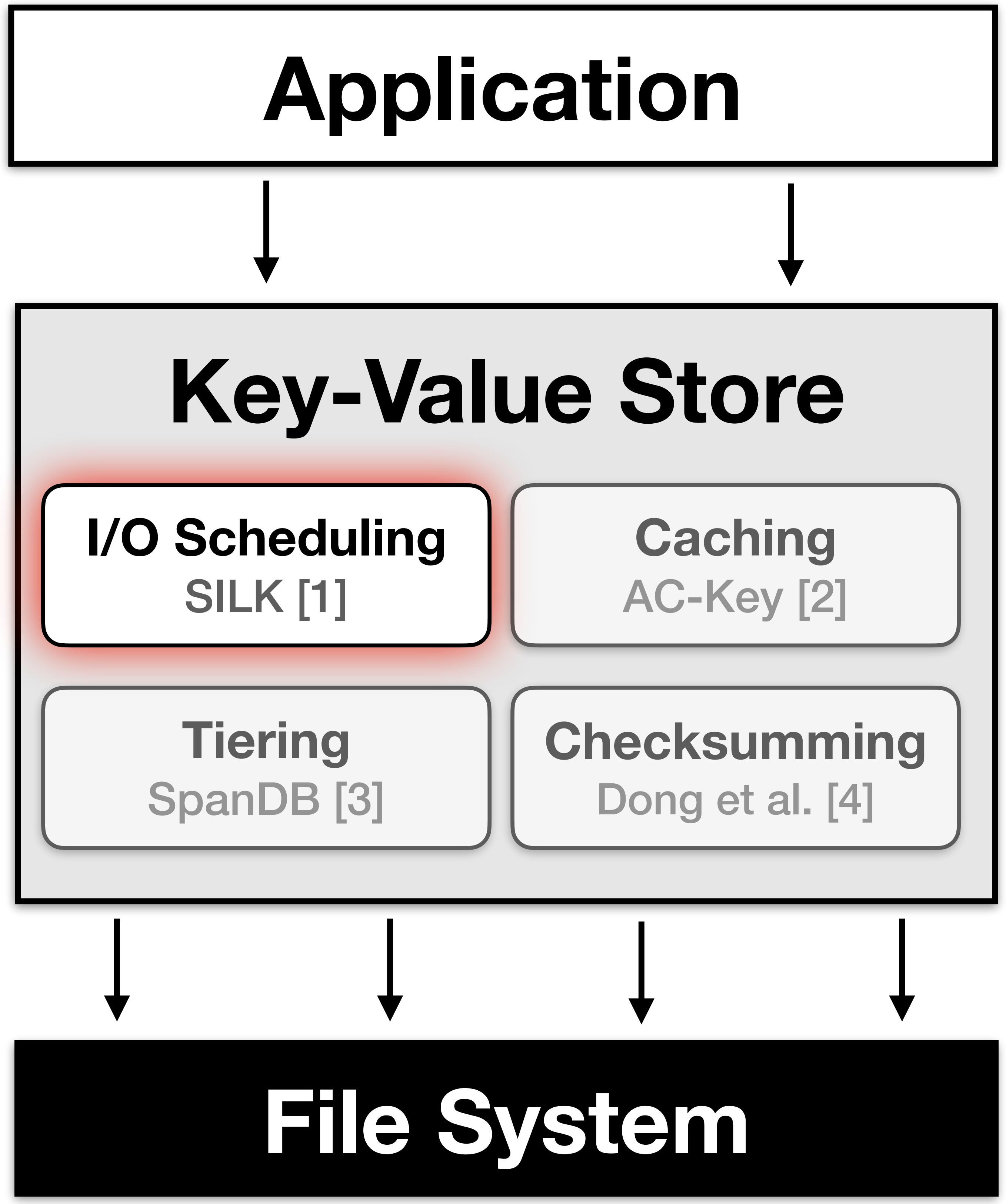


[1] *"SILK: Preventing Latency Spikes in Log-Structured Merge Key-Value Stores"*. Balmau et al. USENIX ATC 2019.
[2] *"AC-Key: Adaptive Caching for LSM-based Key-Value Stores"*. Wu et al. USENIX ATC 2020.
[3] *"SpanDB: A Fast, Cost-Effective LSM-tree Based KV Store on Hybrid Storage"*. Chen et al. USENIX FAST 2021.
[4] *"Evolution of Development Priorities in Key-Value Stores Serving Large-scale Applications: The RocksDB Experience"*. Dong et al. USENIX FAST 2021.

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



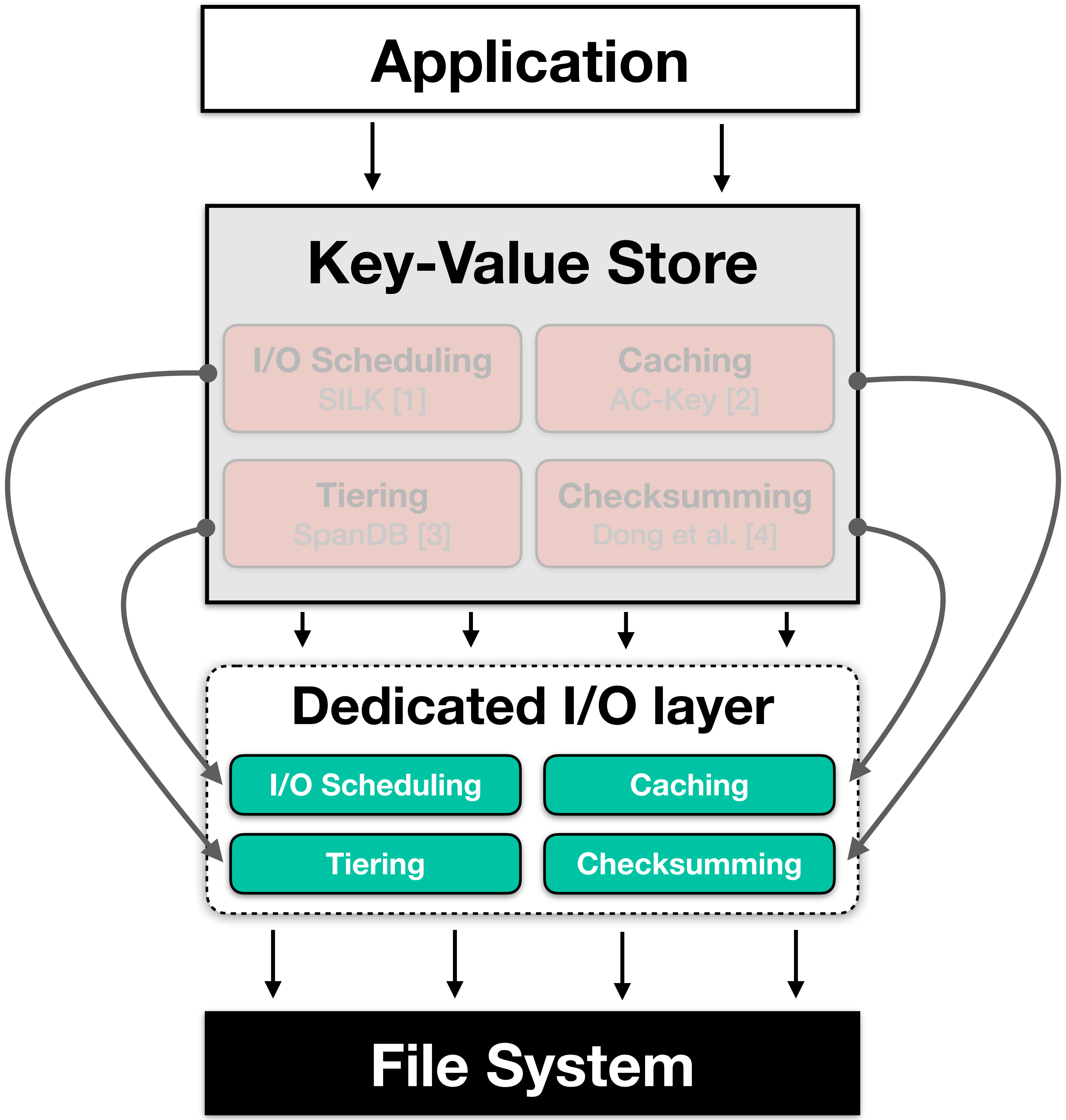
SILK's I/O Scheduler

- 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

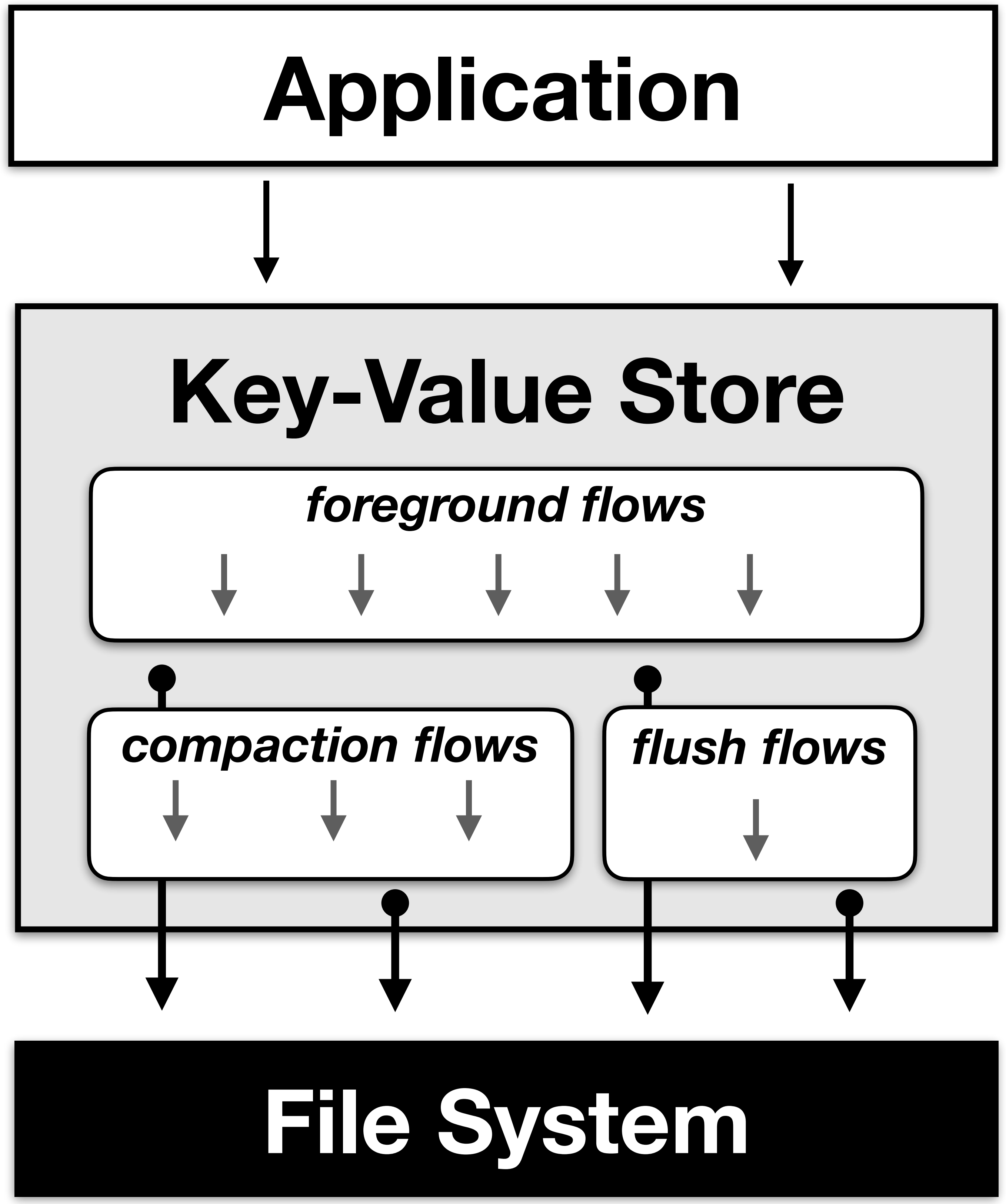
- I/O optimizations should be disaggregated from the internal logic
- Moved to a dedicated I/O layer
- Generally applicable
- Portable across different scenarios



Challenge #2

Rigid interfaces

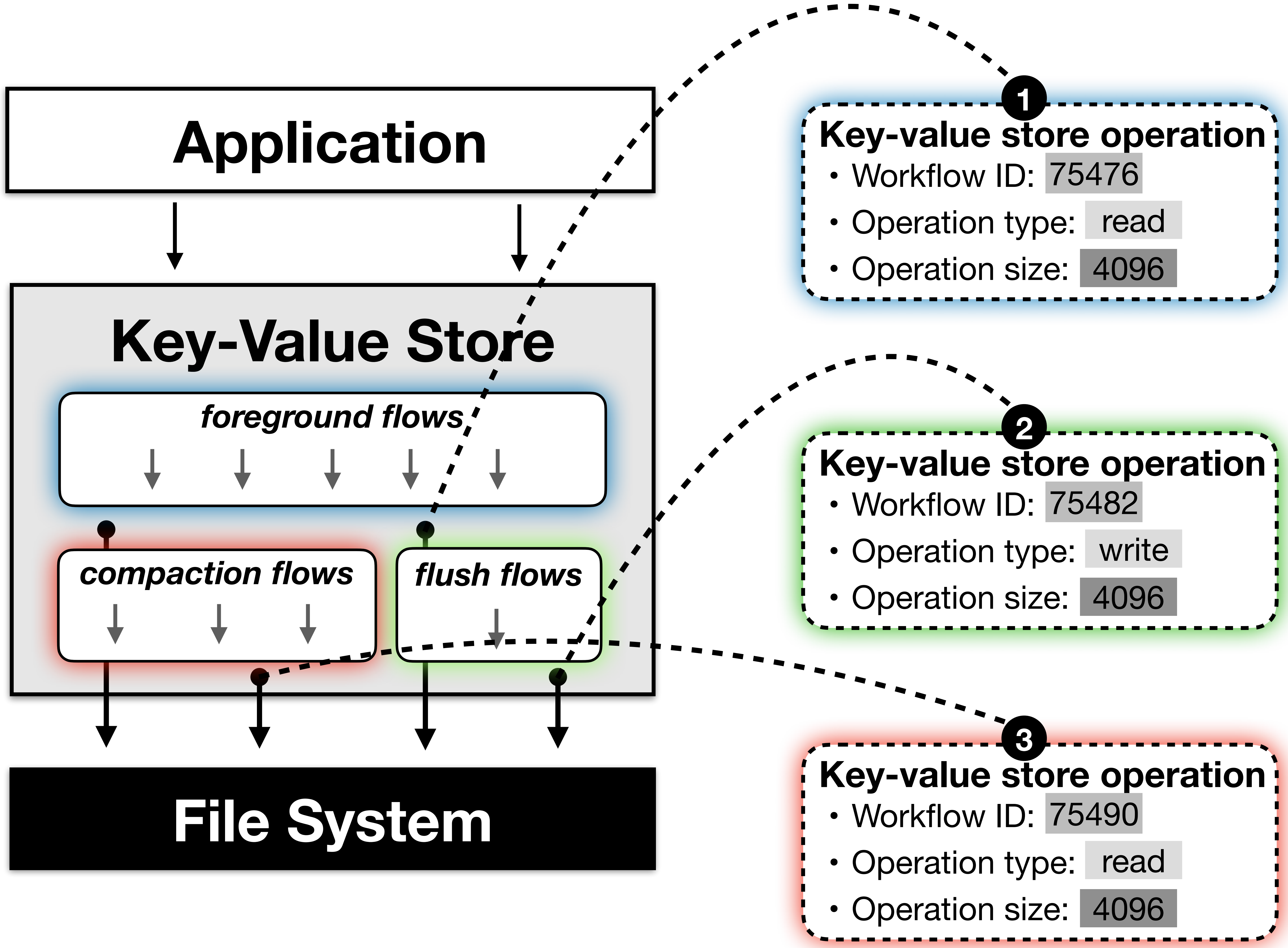
- Decoupled optimizations lose granularity and internal application knowledge
- I/O layers communicate through rigid interfaces
- Discard information that could be used to classify and differentiate requests



Challenge #2

❌ Rigid interfaces

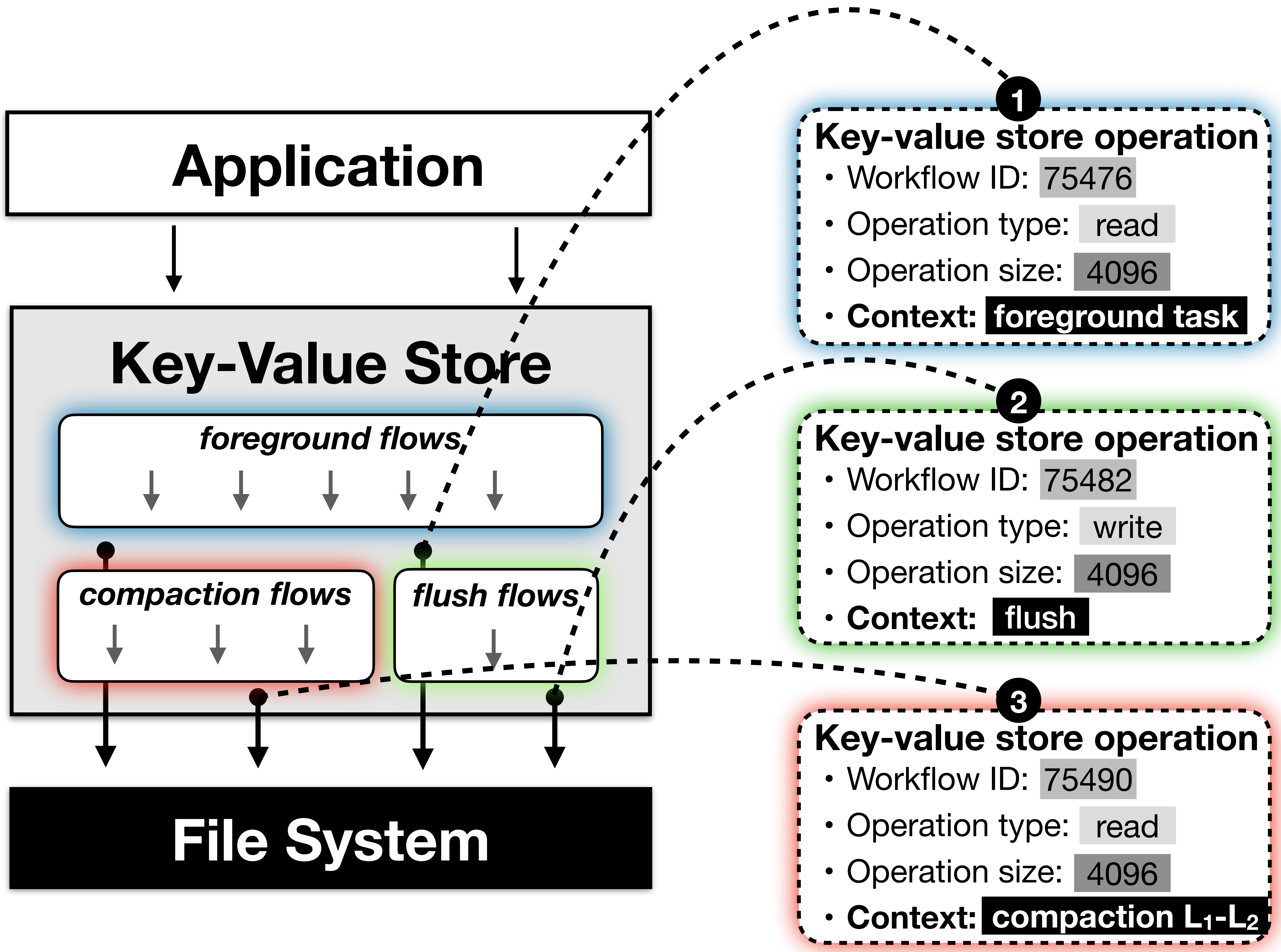
- Decoupled optimizations lose granularity and internal application knowledge
- I/O layers communicate through rigid interfaces
- Discard information that could be used to classify and differentiate requests



Challenge #2

Information propagation

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

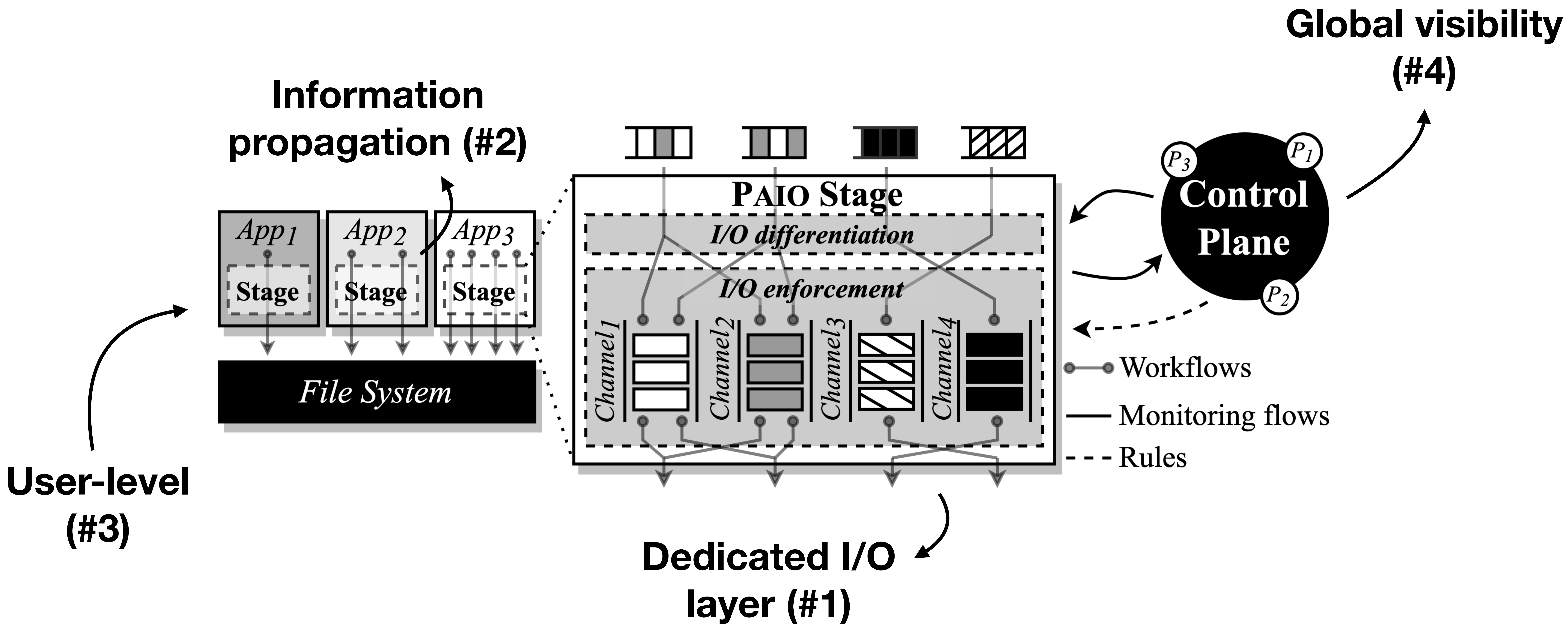


PAIO

- **User-level** framework for building **portable** and **generally applicable** optimizations
- Adopts ideas from **Software-Defined Storage** [6]
    - 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

[5] *"PAIO: General, Portable I/O Optimizations with Minor Application Modifications"*. Macedo et al. USENIX FAST 2022.
[6] *"A Survey and Classification of Software-Defined Storage Systems"*. Macedo et al. ACM CSUR 2020.

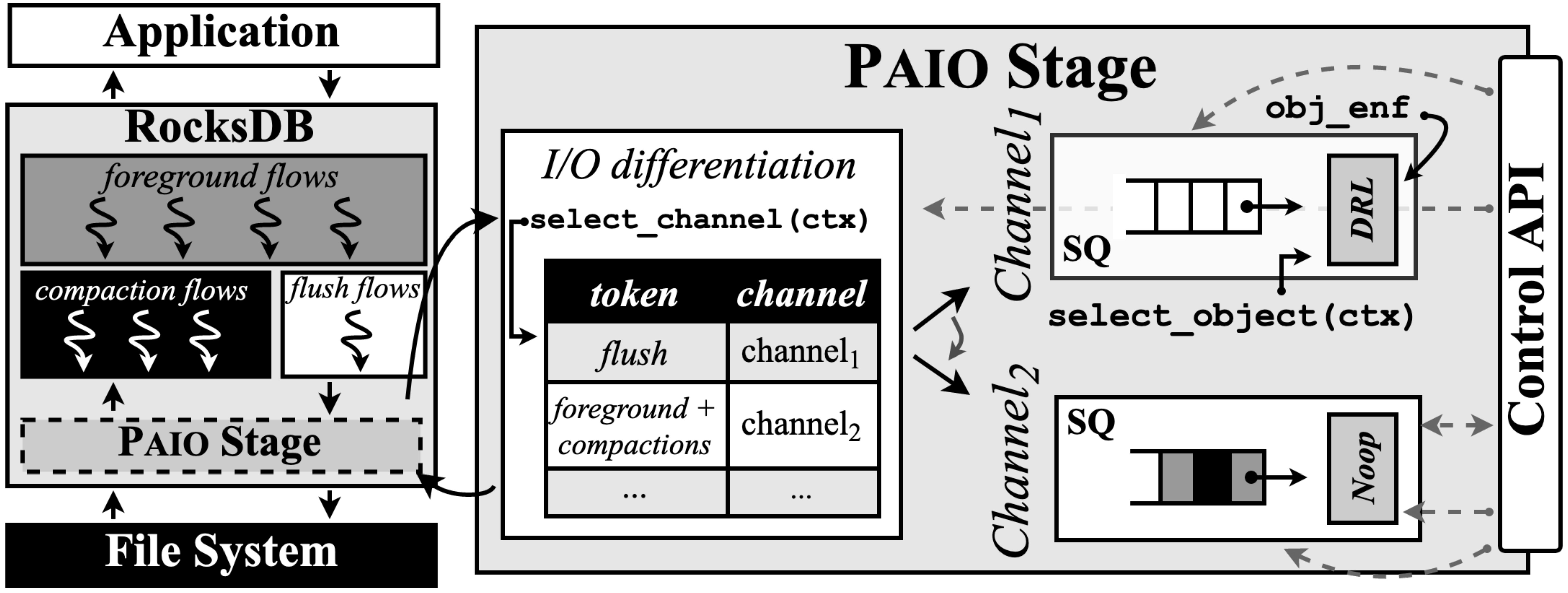
PAIO design



Global visibility (#4)
Information propagation (#2)
App1
App2
App3
Stage
Stage
Stage
File System
PAIO Stage
I/O differentiation
I/O enforcement
Channel1
Channel2
Channel3
Channel4
Control Plane
P1
P2
P3
Workflows
Monitoring flows
Rules
User-level (#3)
Dedicated I/O layer (#1)

PAIO design

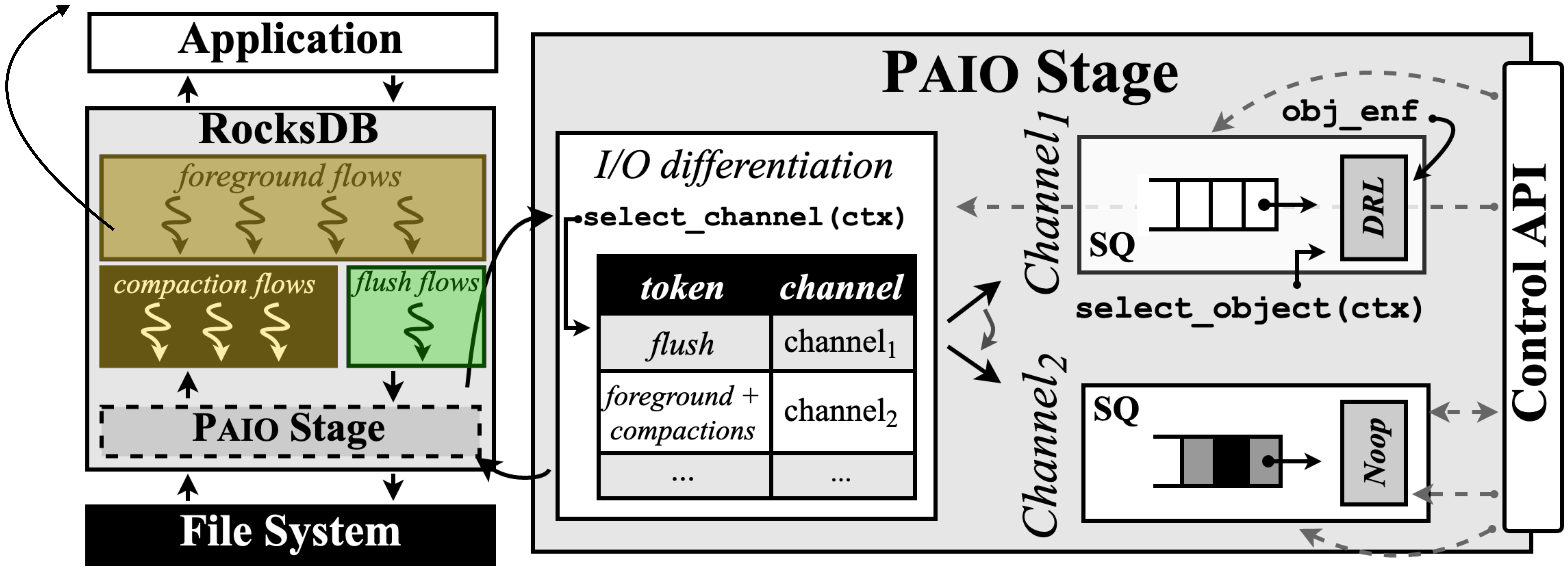
- I/O differentiation
- I/O enforcement
- Control plane interaction



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

I/O differentiation

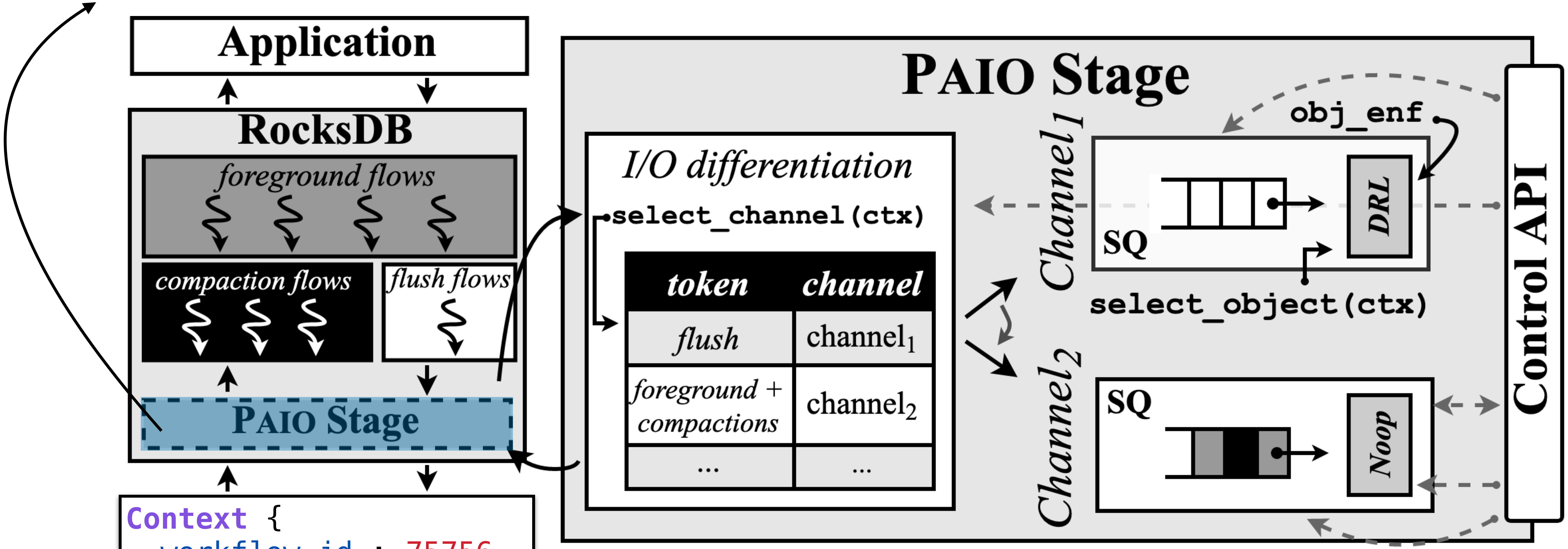
Context propagation:
instrumentation + propagation phases



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

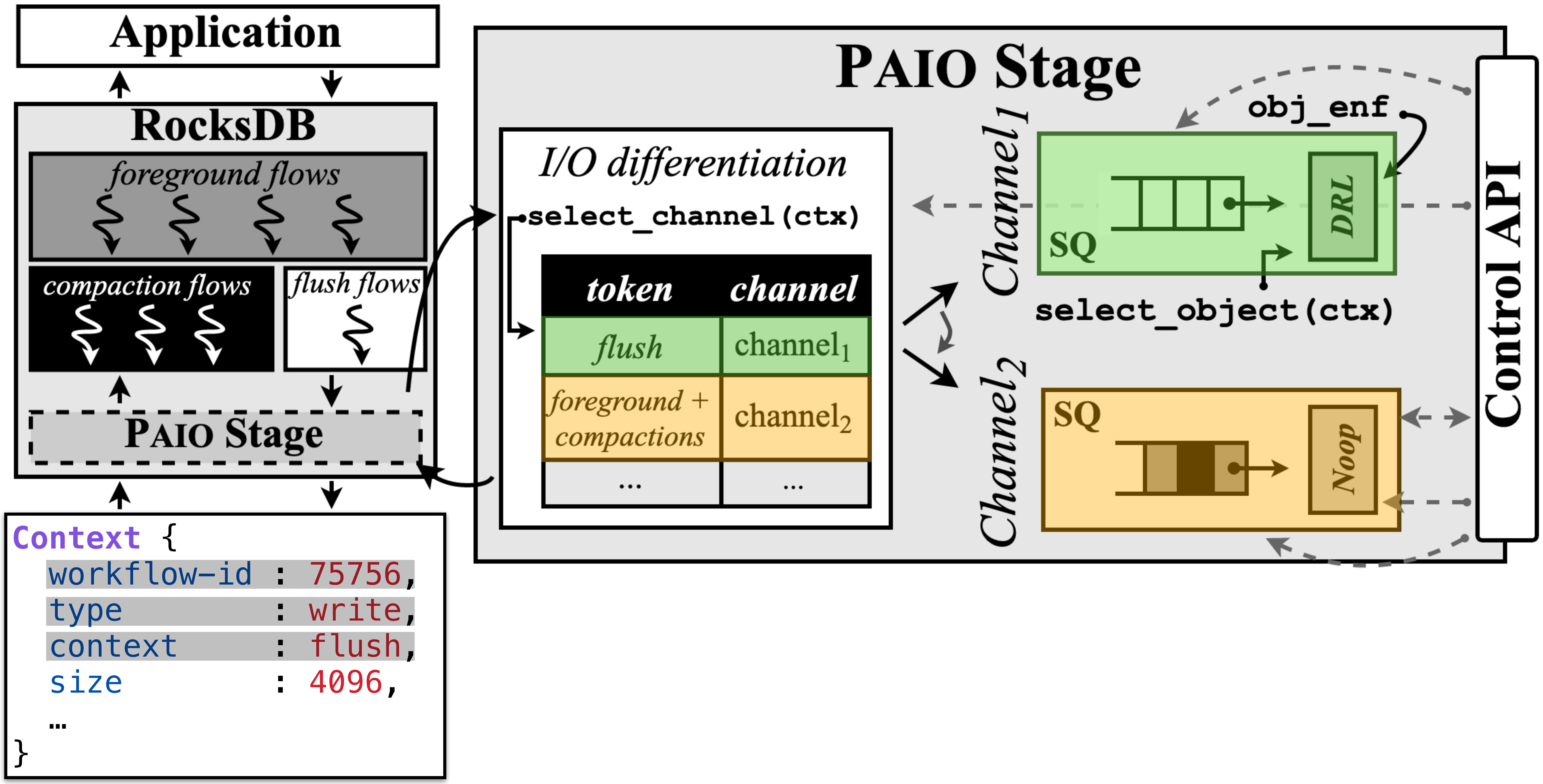
I/O differentiation

Context propagation:
propagation + classification phases

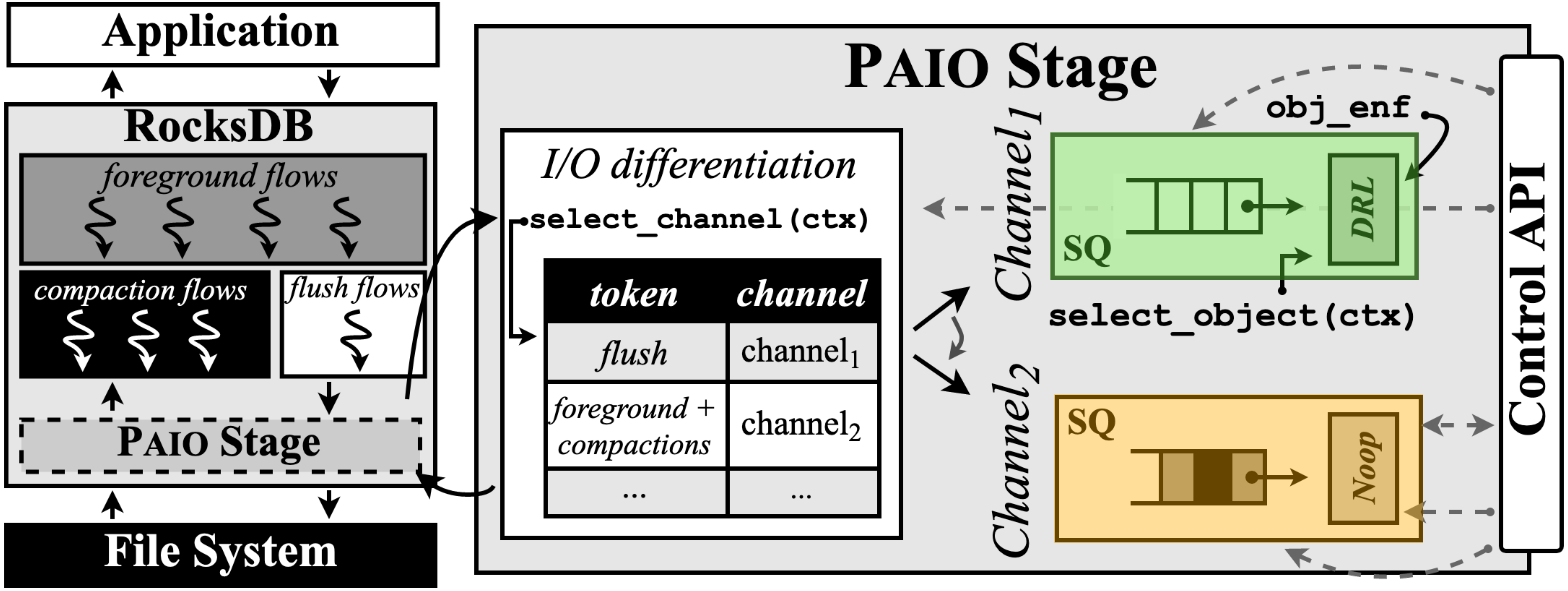



```
Context {
  workflow-id : 75756,
  type        : write,
  context     : flush,
  size        : 4096,
  …
}
```

I/O differentiation

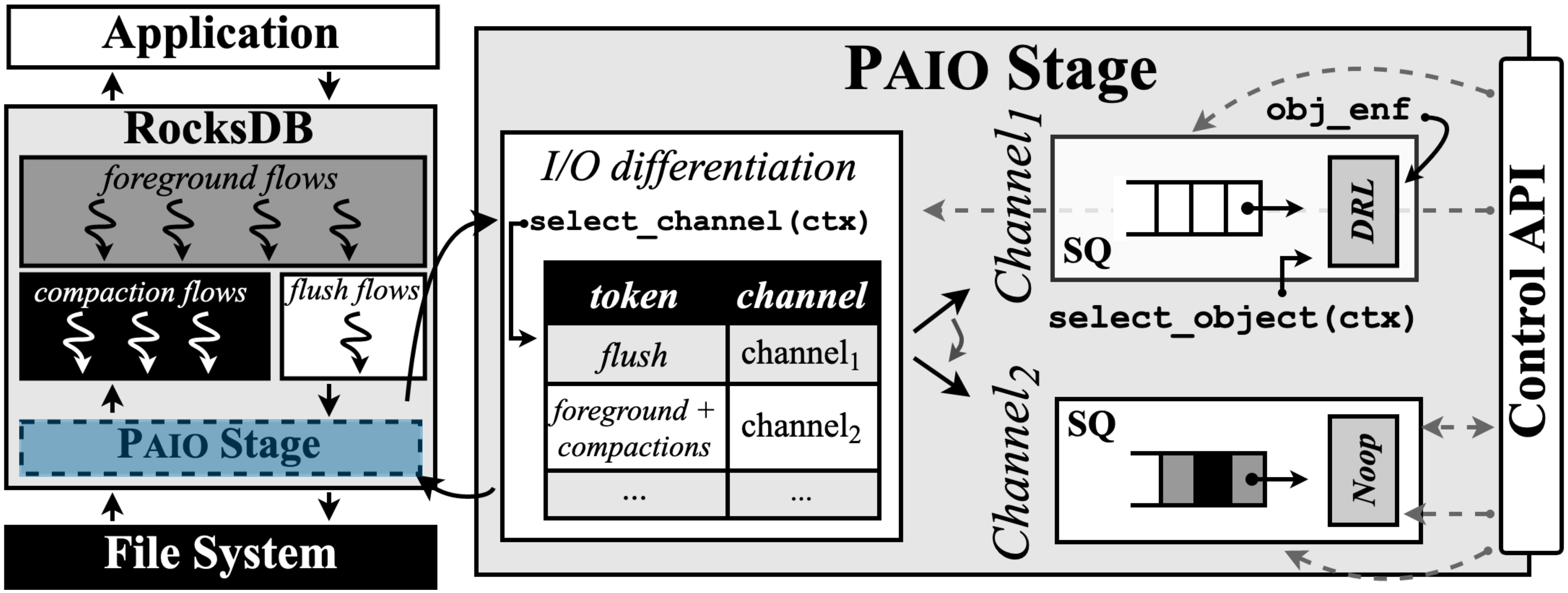


I/O enforcement



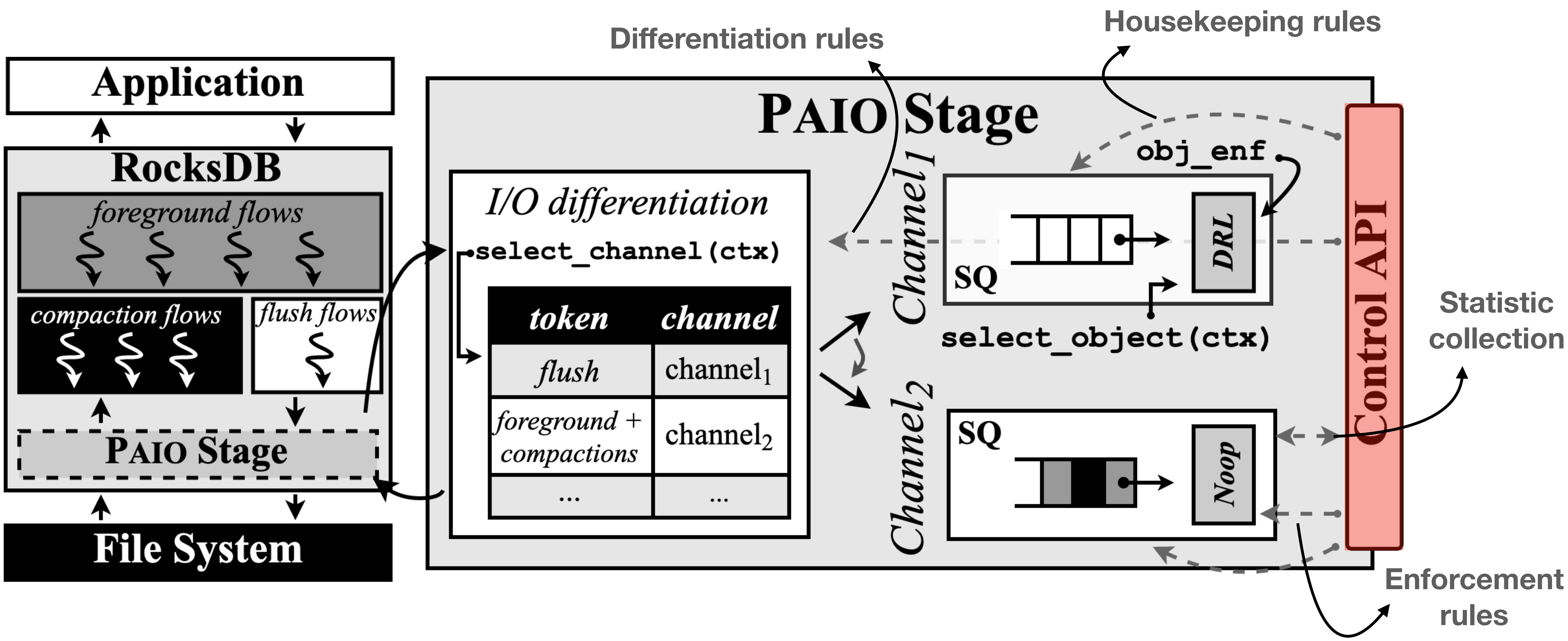
PAIO currently supports **Noop** *(passthrough)* and **DRL** *(token-bucket)* enforcement objects

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 KVS

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 KVS

RocksDB

- Interference between foreground and background t
- Latency spikes occur due to L_0-L_1 compactions an

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

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

Mixture workload

50% read 50% write

System configuration and workload

- 8 client threads and 8 background threads
- Memory limited to 1GB and I/O BW to 200MB/s
- Bursty workload with peaks and valleys

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

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

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

Encontro Nacional de Sistemas Distribuídos

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

Ricardo Macedo

INESC TEC & University of Minho

ricardo.g.macedo@inesctec.pt

github.com/dsrhaslab

dsr-haslab.github.io