

# Enlightening the I/O Path: A Holistic Approach for Application Performance

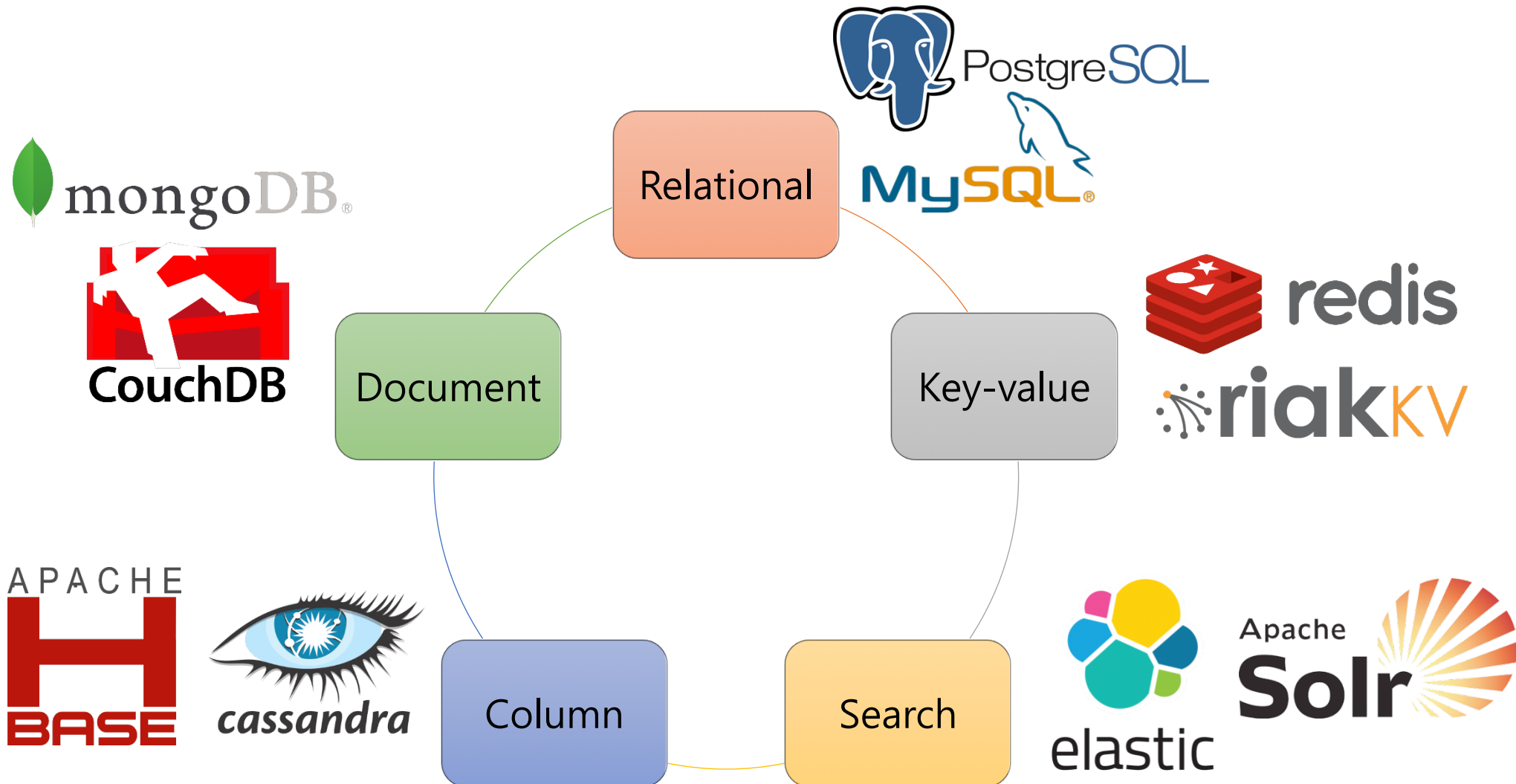
appeared in FAST'17

Jinkyu Jeong

Sungkyunkwan University

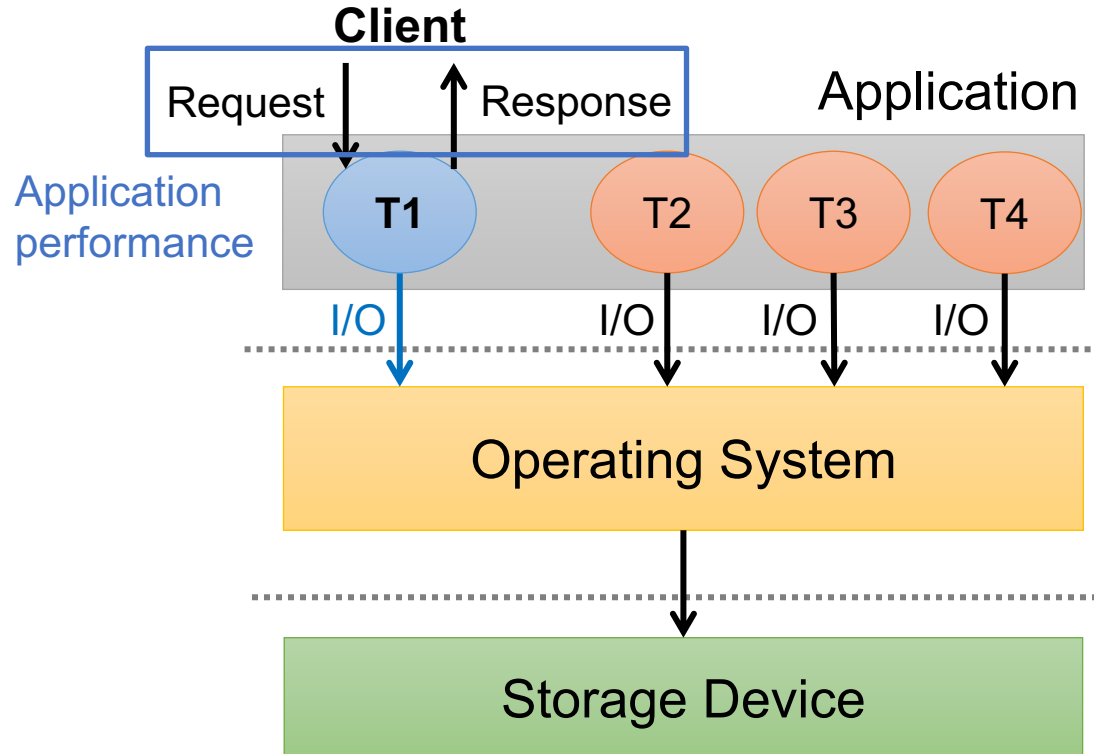


# Data-Intensive Applications



# Data-Intensive Applications

- Common structure



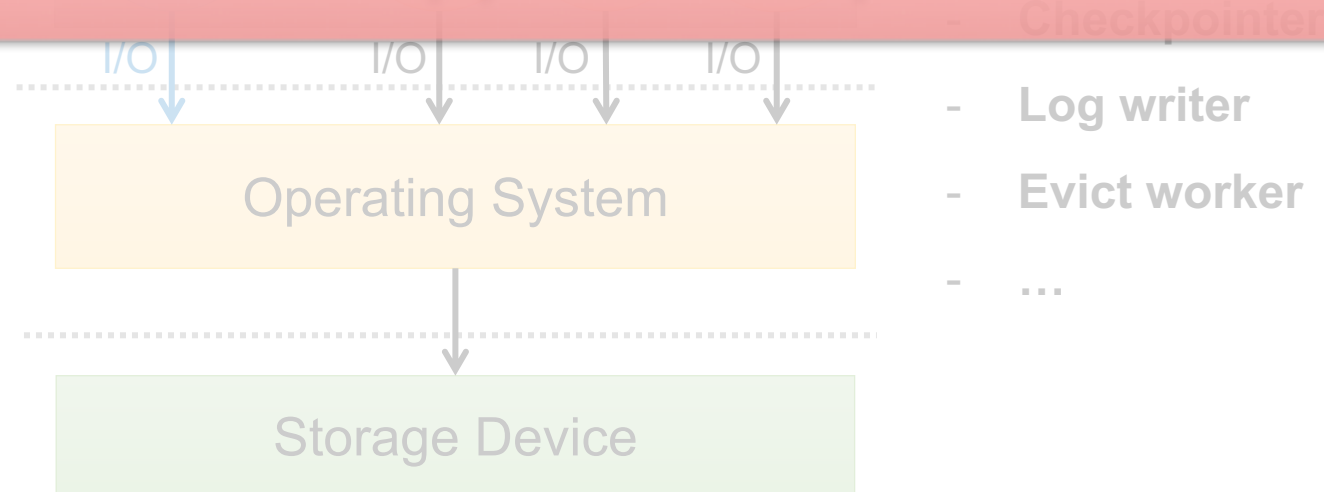
- \* **Example: MongoDB**

- **Client (foreground)**
    - **Checkpoint**
    - **Log writer**
    - **Eviction worker**
    - ...

# Data-Intensive Applications

- Common structure

**Background tasks are problematic for application performance**

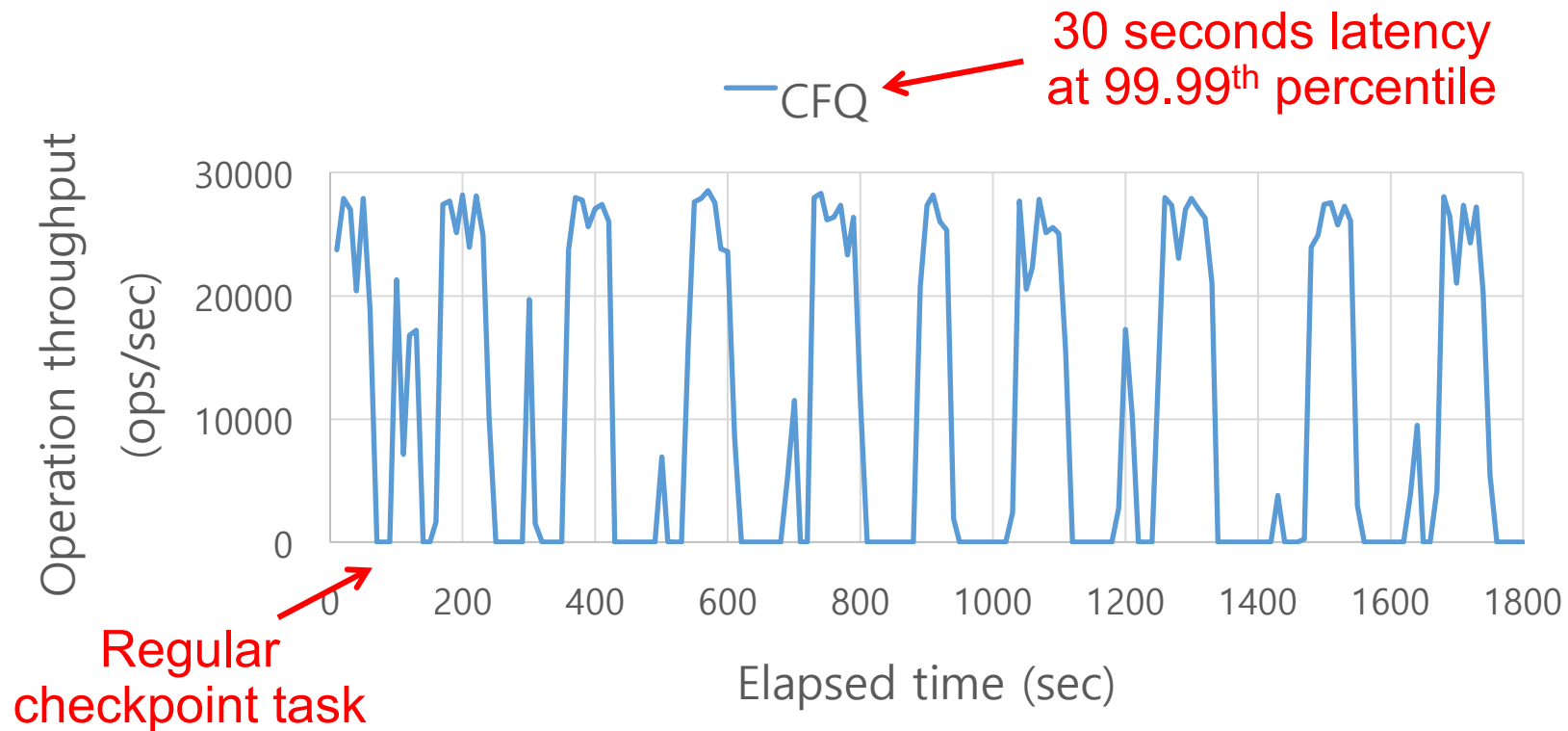


# Application Impact

- Illustrative experiment
  - YCSB update-heavy workload against MongoDB

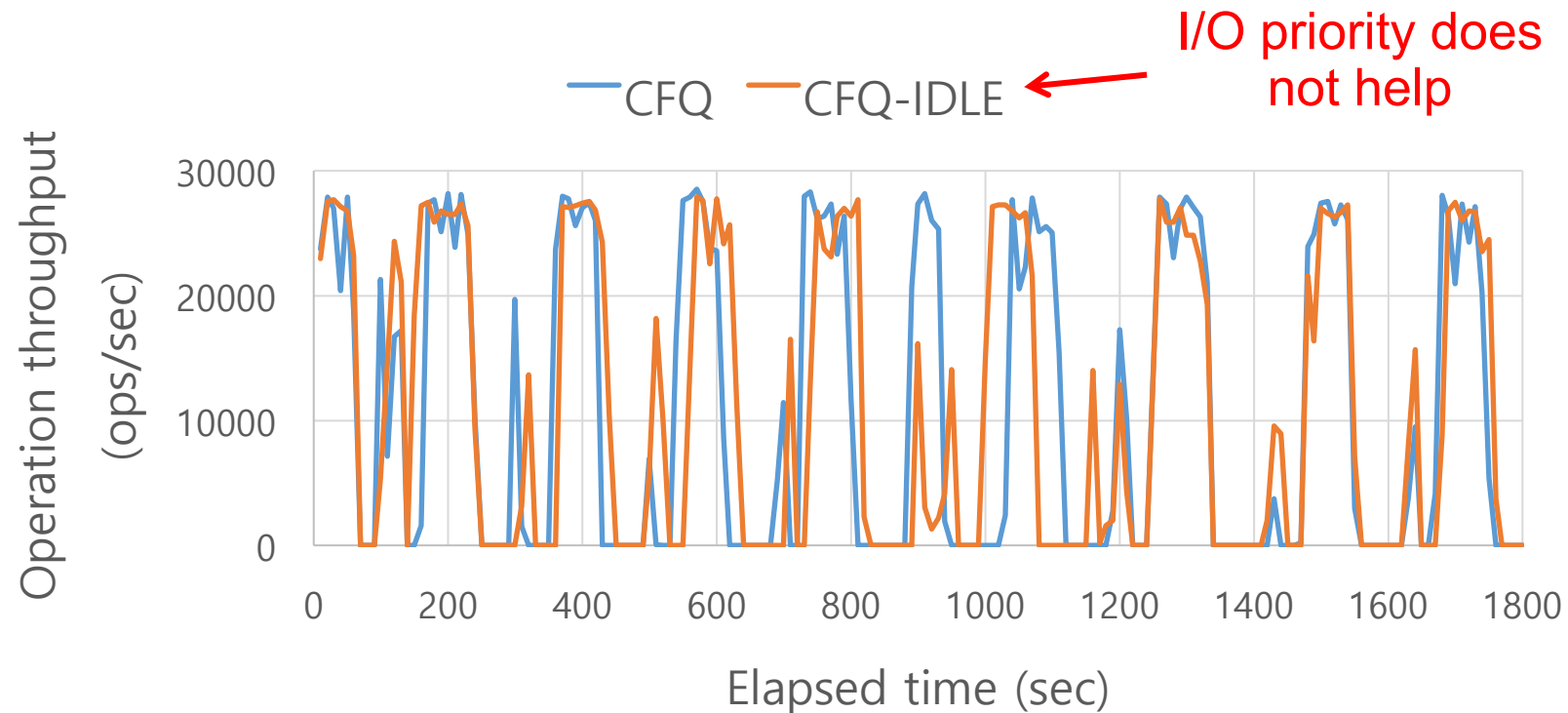
# Application Impact

- Illustrative experiment
  - YCSB update-heavy workload against MongoDB



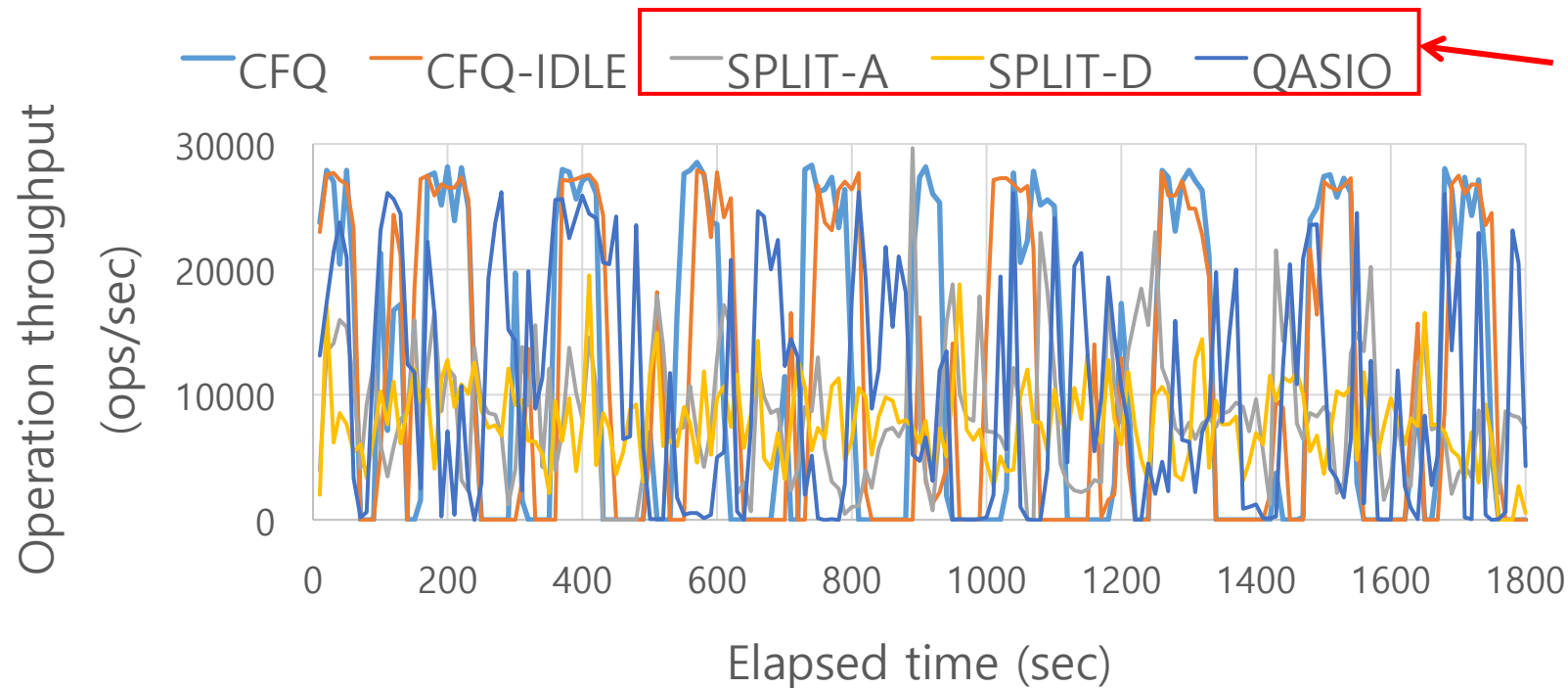
# Application Impact

- Illustrative experiment
  - YCSB update-heavy workload against MongoDB



# Application Impact

- Illustrative experiment
  - YCSB update-heavy workload against MongoDB





# What's the Problem?

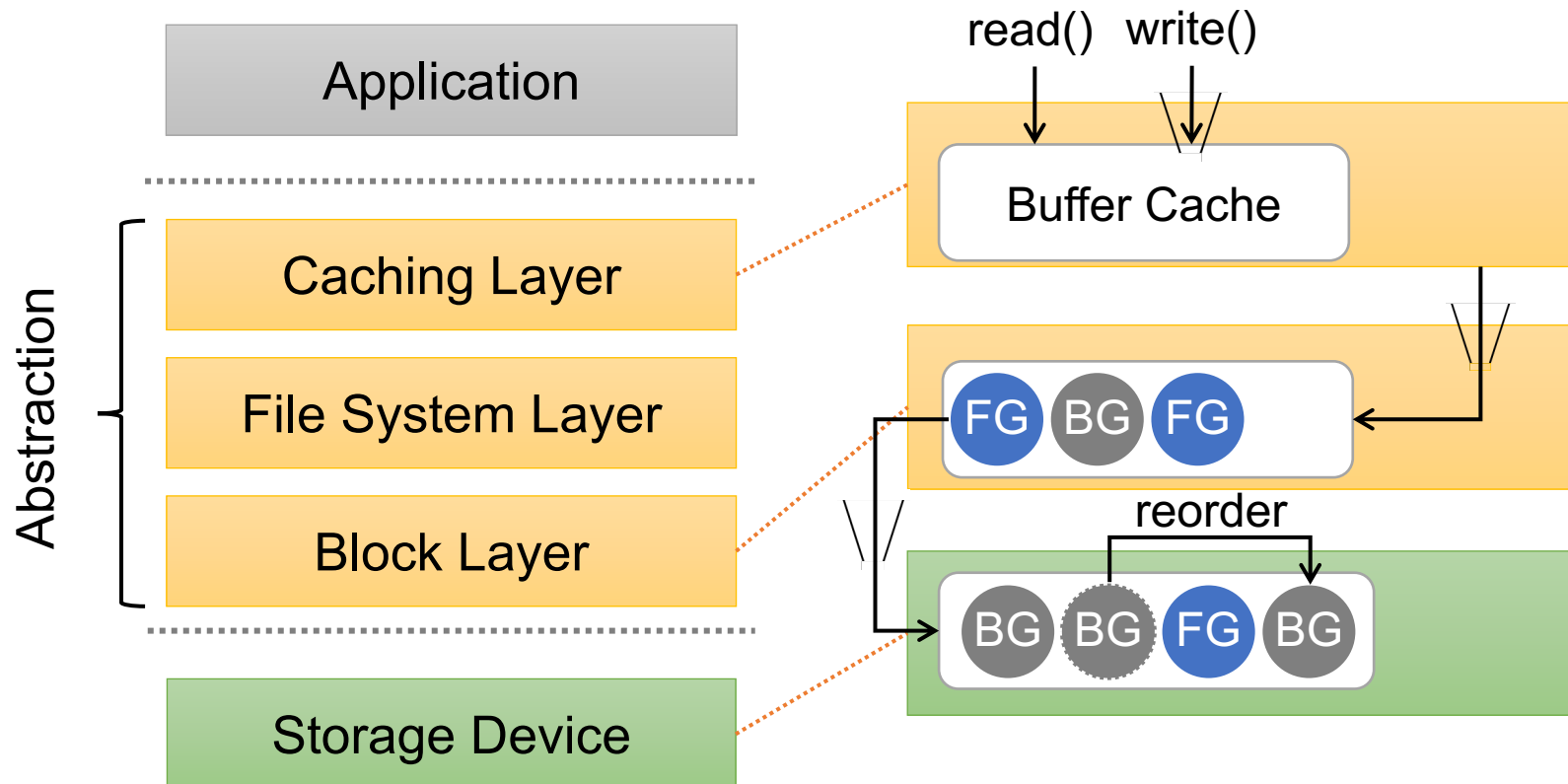
- Independent policies in multiple layers
  - Each layer processes I/Os w/ limited information
- I/O priority inversion
  - Background I/Os can arbitrarily delay foreground tasks

# What's the Problem?

- **Independent policies in multiple layers**
  - Each layer processes I/Os w/ limited information
- I/O priority inversion
  - Background I/Os can arbitrarily delay foreground tasks

# Multiple Independent Layers

- Independent I/O processing

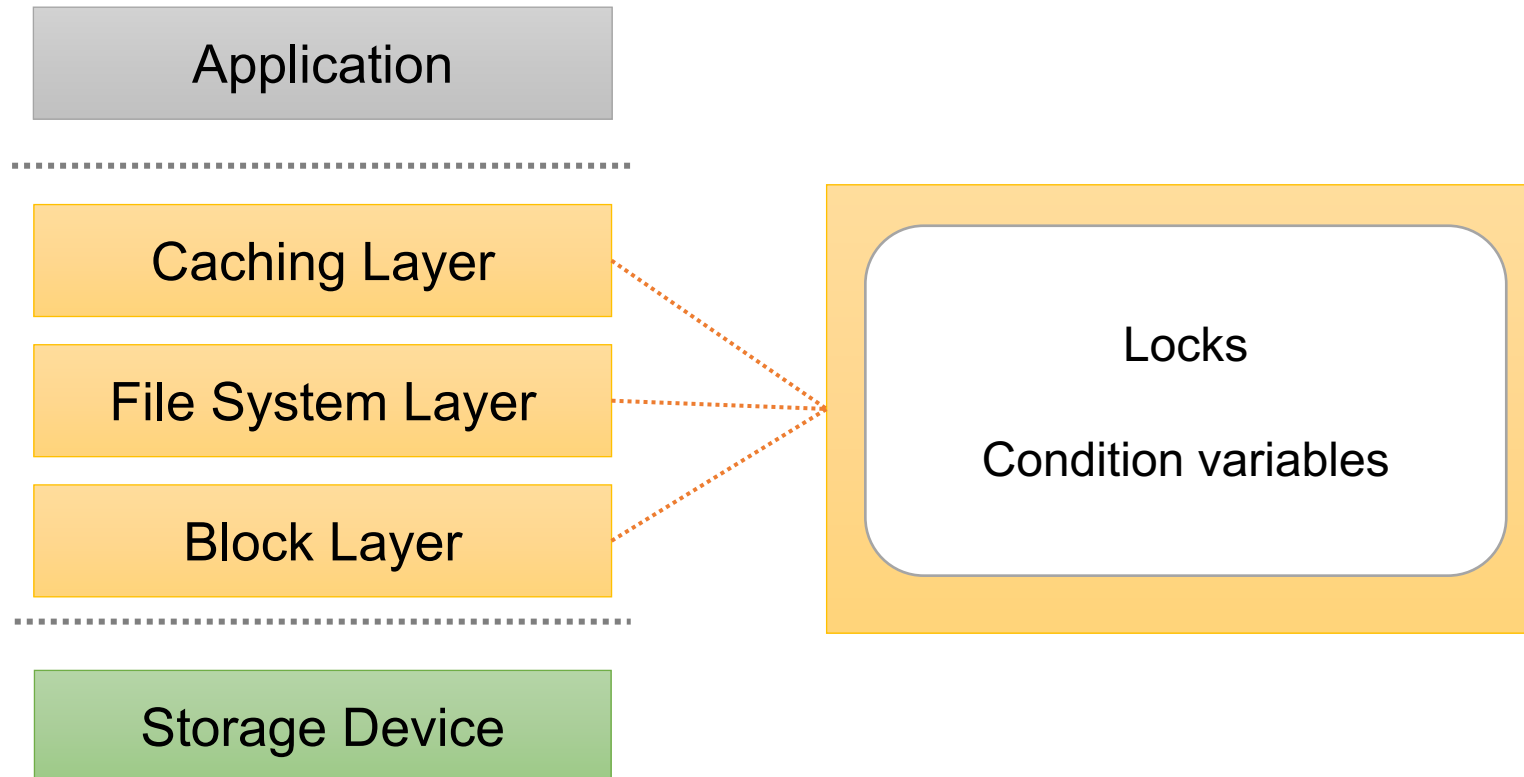


# What's the Problem?

- Independent policies in multiple layers
  - Each layer processes I/Os w/ limited information
- **I/O priority inversion**
  - Background I/Os can arbitrarily delay foreground tasks

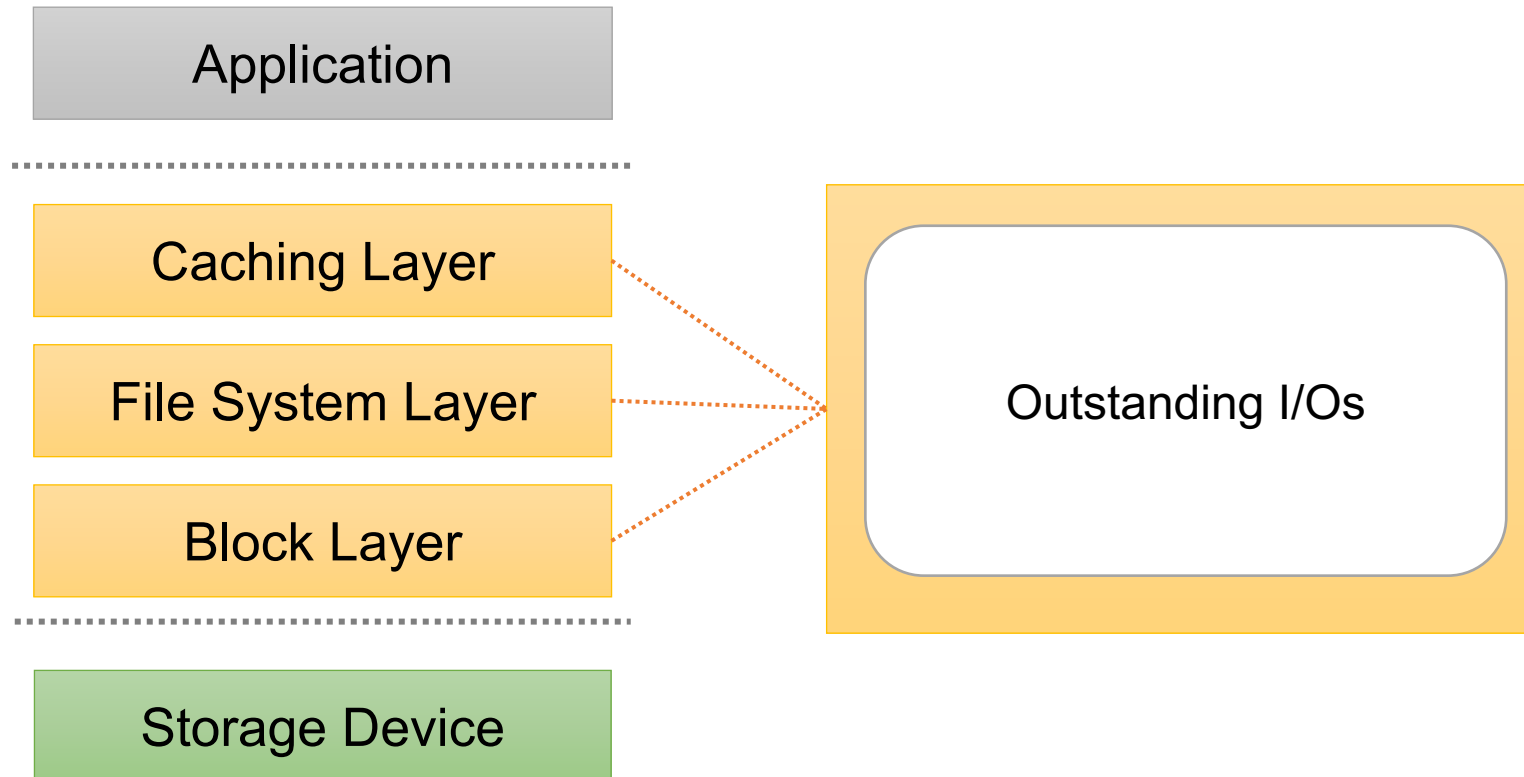
# I/O Priority Inversion

- Task dependency



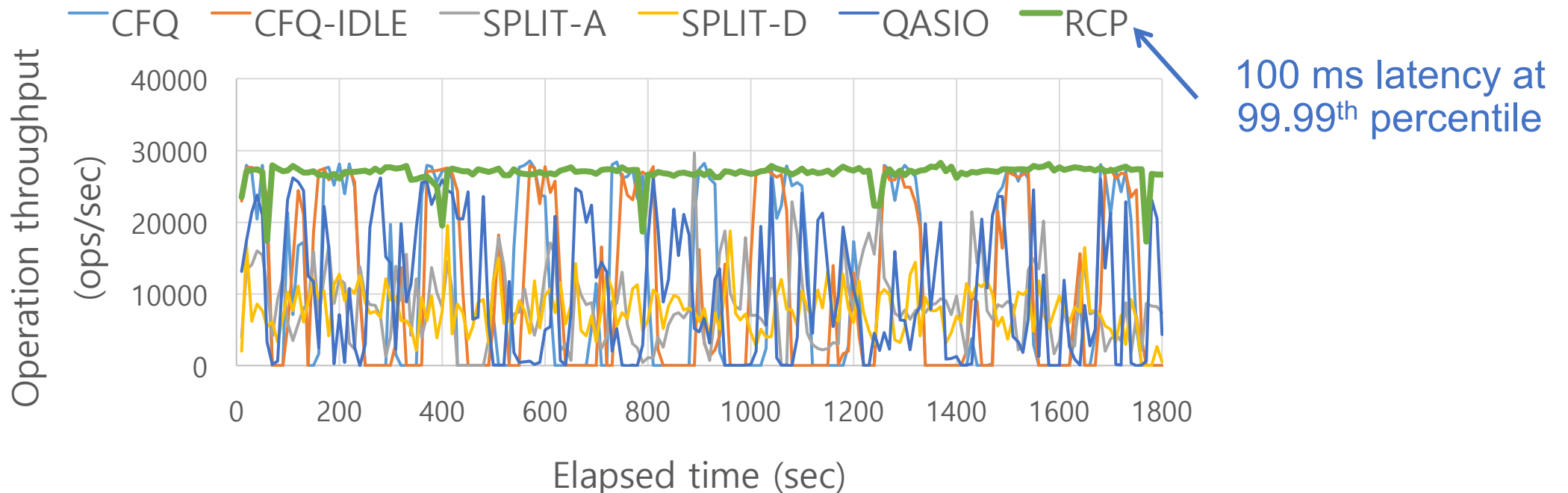
# I/O Priority Inversion

- I/O dependency



# Our Approach

- Request-centric I/O prioritization (RCP)
  - Critical I/O: I/O in the critical path of request handling
  - Policy: holistically prioritizes critical I/Os along the I/O path



# Challenges

- **How to accurately identify I/O criticality**
- How to effectively enforce I/O criticality



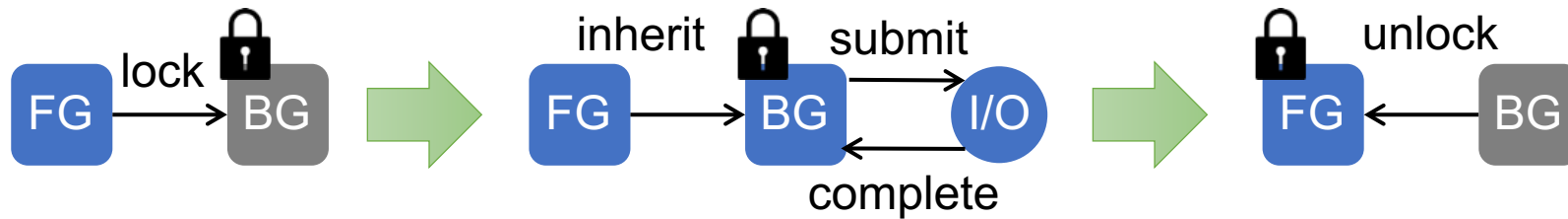
# Critical I/O Detection

- Enlightenment API
  - Interface for tagging foreground tasks
- I/O priority inheritance
  - Handling task dependency
  - Handling I/O dependency

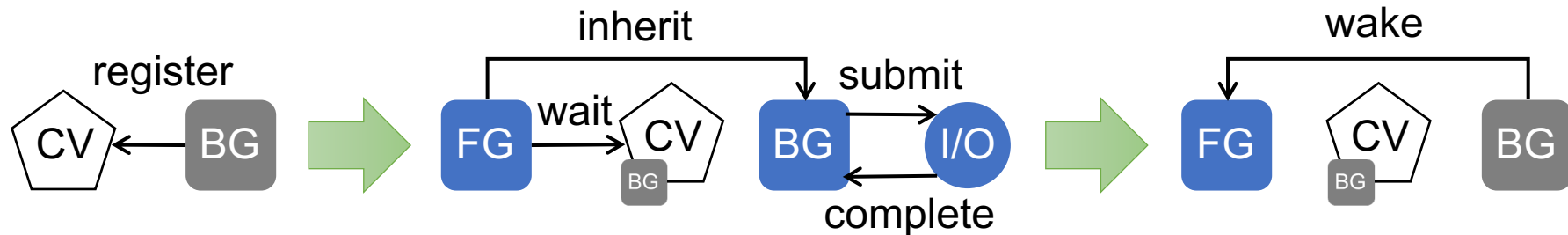
# I/O Priority Inheritance

- Handling task dependency

- Locks

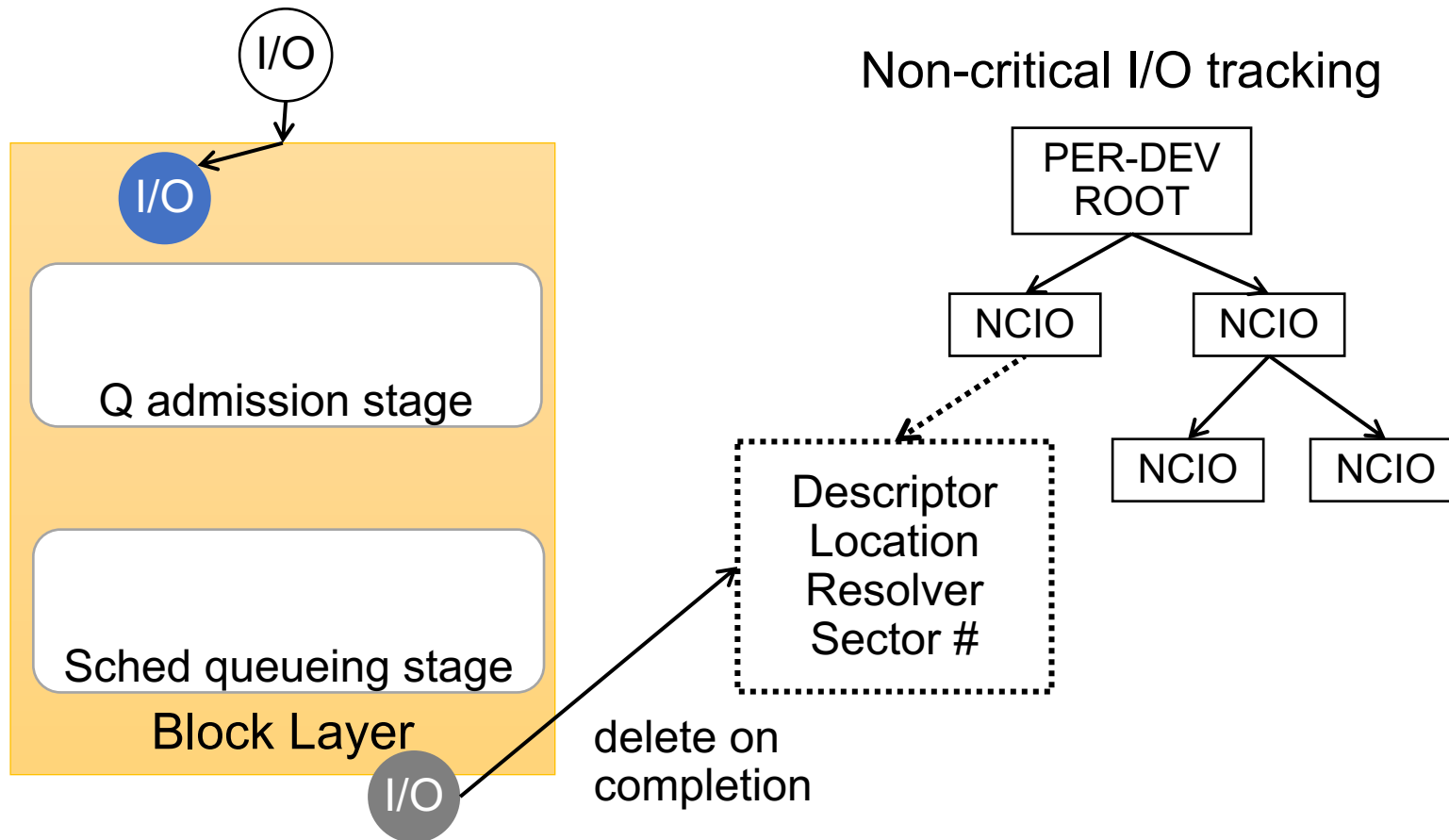


- Condition variables



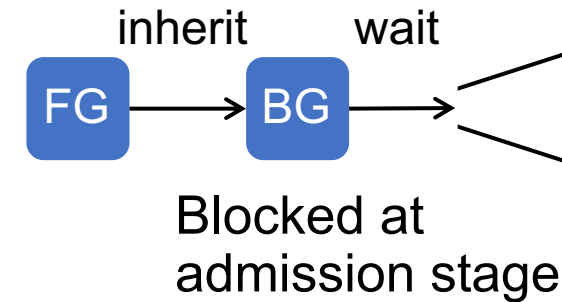
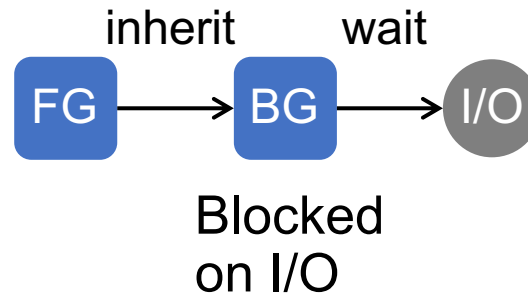
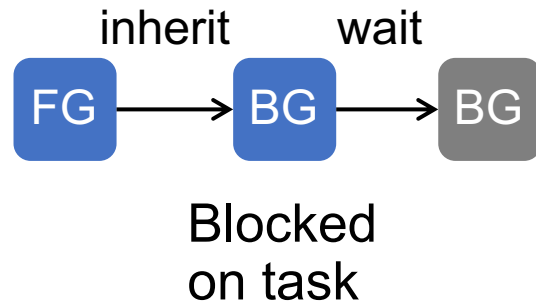
# I/O Priority Inheritance

- Handling I/O dependency



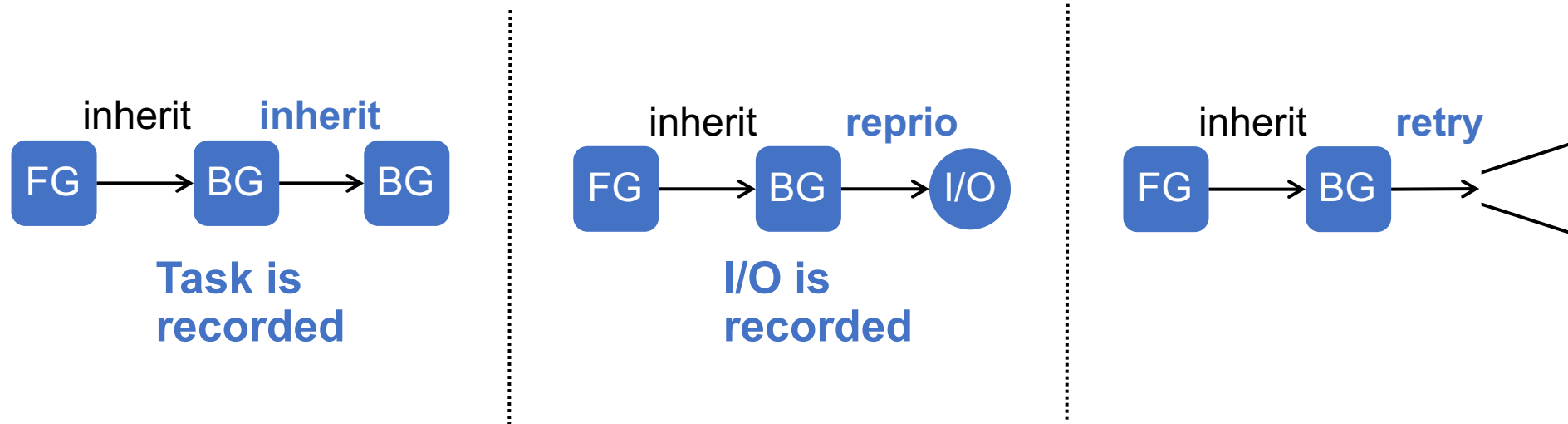
# Handling Transitive Dependency

- Possible states of dependent task



# Handling Transitive Dependency

- Recording blocking status



# Challenges

- How to accurately identify I/O criticality
  - Enlightenment API
  - I/O priority inheritance
  - Recording blocking status
- **How to effectively enforce I/O criticality**

# Criticality-Aware I/O Prioritization

- Caching layer
  - Apply low dirty ratio for non-critical writes (1% by default)
- Block layer
  - Isolate allocation of block queue slots
  - Maintain 2 FIFO queues
  - Schedule critical I/O first
  - Limit # of outstanding non-critical I/Os (1 by default)
  - Support queue promotion to resolve I/O dependency

# Evaluation

- Implementation on Linux 3.13 w/ ext4
- Application studies
  - PostgreSQL relational database
    - Backend processes as foreground tasks
    - I/O priority inheritance on LWLocks (semop)
  - MongoDB document store
    - Client threads as foreground tasks
    - I/O priority inheritance on Pthread mutex and condition vars (futex)
  - Redis key-value store
    - Master process as foreground task

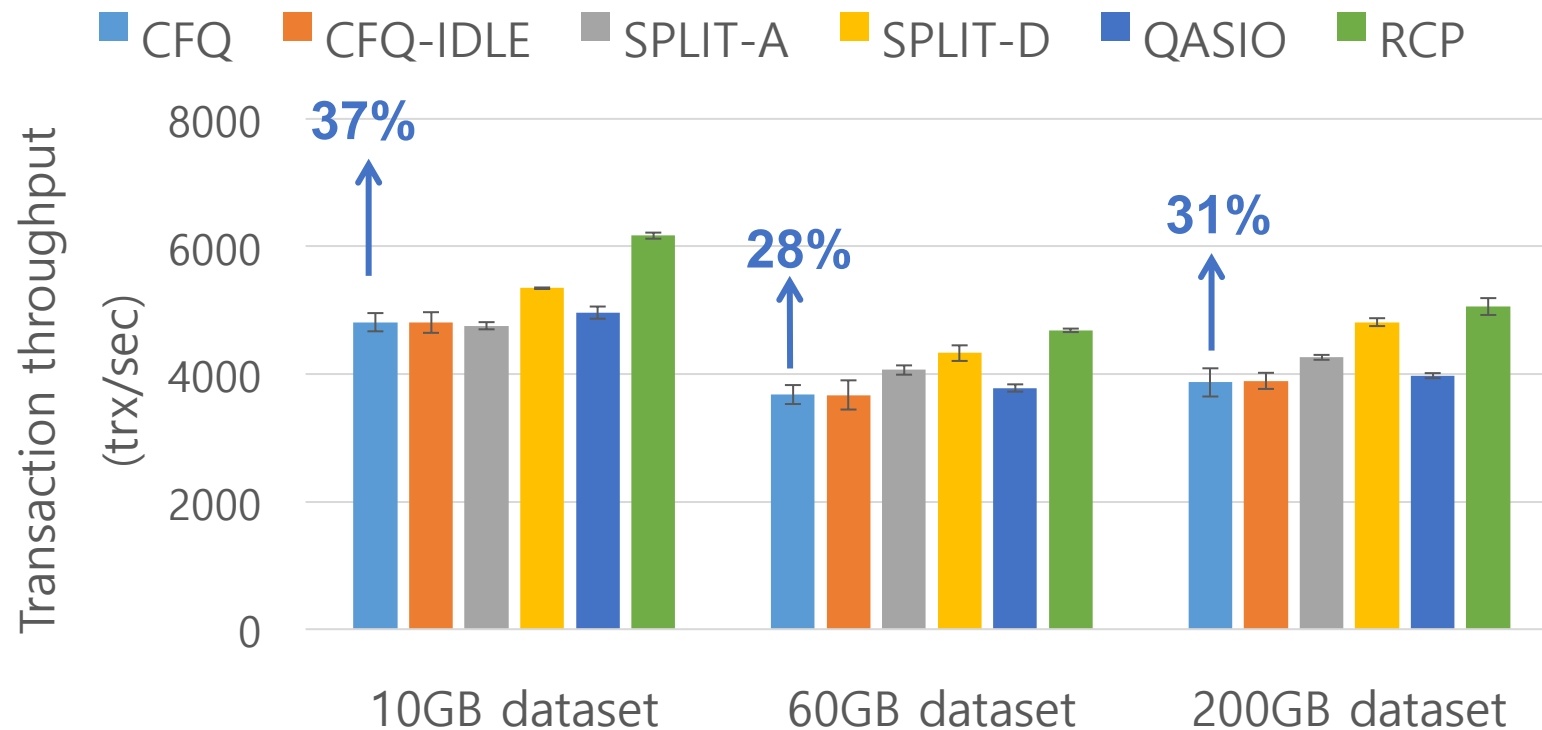


# Evaluation

- Experimental setup
  - 2 Dell PowerEdge R530 (server & client)
  - 1TB Micron MX200 SSD
- I/O prioritization schemes
  - CFQ (default), CFQ-IDLE
  - SPLIT-A (priority), SPLIT-D (deadline) [SOSP'15]
  - QASIO [FAST'15]
  - **RCP**

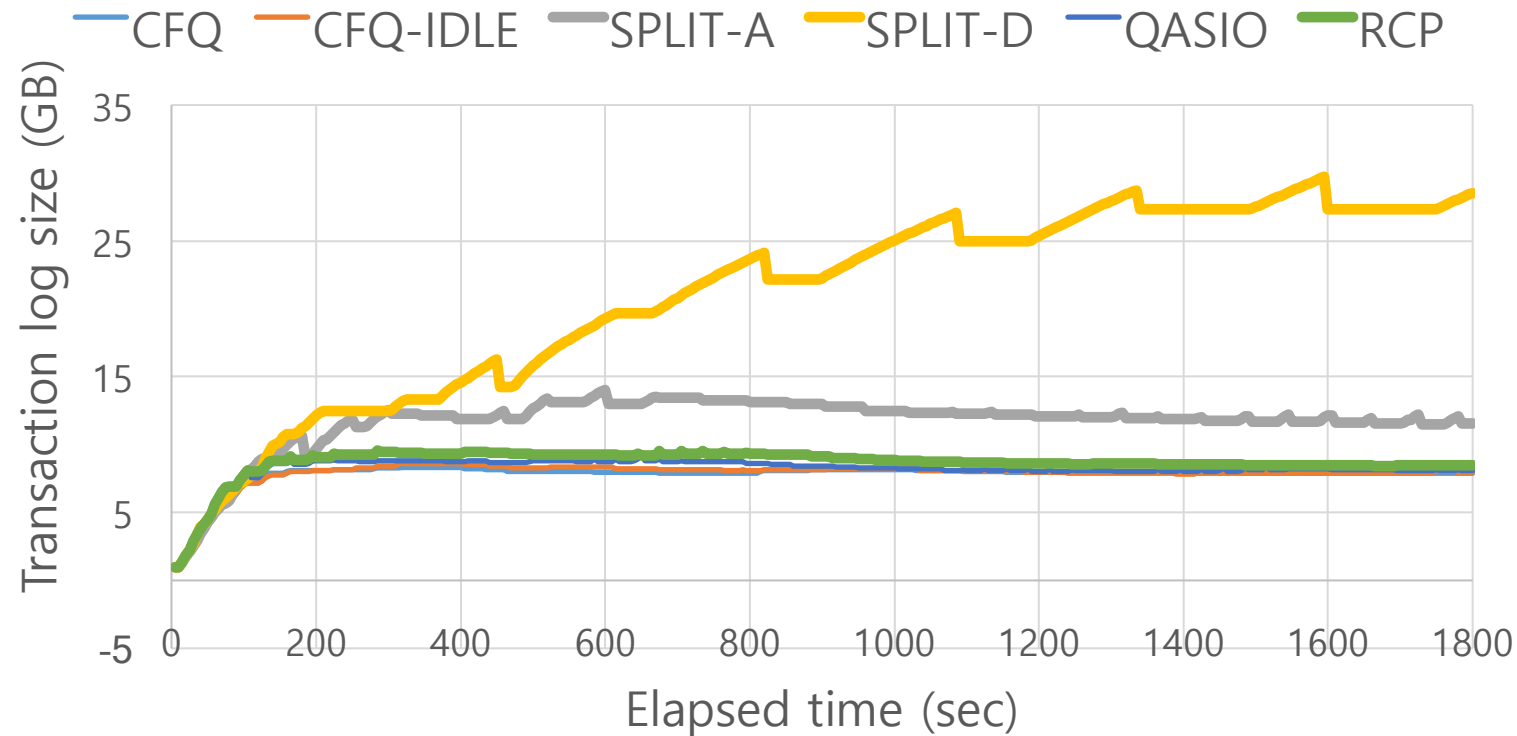
# Application Throughput

- PostgreSQL w/ TPC-C workload



# Application Throughput

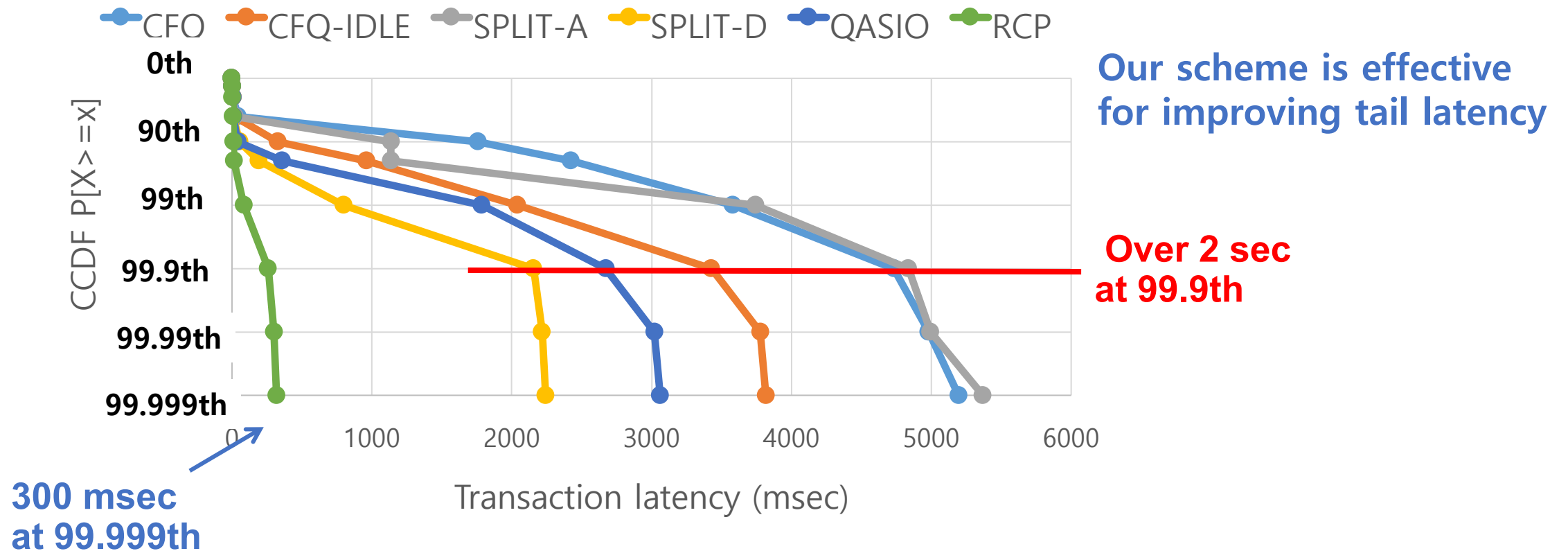
- Impact on background task



**Our scheme improves application throughput w/o penalizing background tasks**

# Application Latency

- PostgreSQL w/ TPC-C workload



# Summary of Other Results

- Performance results
  - MongoDB: 12%-201% throughput, 5x-20x latency at 99.9<sup>th</sup>
  - Redis: 7%-49% throughput, 2x-20x latency at 99.9<sup>th</sup>
- Analysis results
  - System latency analysis using LatencyTOP
  - System throughput vs. Application latency
  - Need for holistic approach

# Conclusions

- Key observation
  - All the layers in the I/O path should be considered as a whole with I/O priority inversion in mind for effective I/O prioritization
- Request-centric I/O prioritization
  - Enlightens the I/O path solely for application performance
  - Improves throughput and latency of real applications
- Ongoing work
  - Practicalizing implementation
  - Applying RCP to database cluster with multiple replicas

# Thank You!

- Questions and comments