IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage

Huaicheng Li[☆✶], Martin L. Putra[☆], Ronald Shi[☆],
Xing Lin[*], Gregory R. Ganger[✶], Haryadi S. Gunawi[☆]

The 28th ACM Symposium on Operating Systems Principles (SOSP'21)

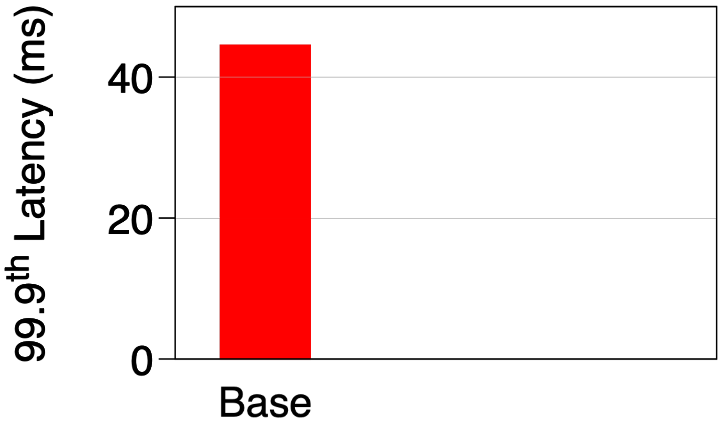
[☆]*University of Chicago,* [✶]*Carnegie Mellon University,* [*]*NetApp*

IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage

IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage

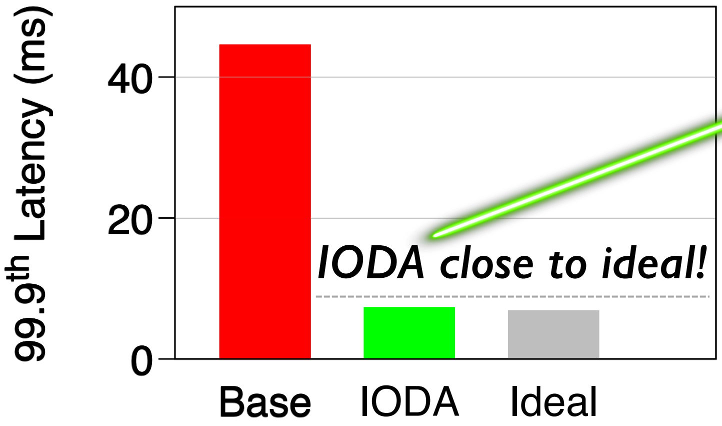
"Small but powerful"

IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage



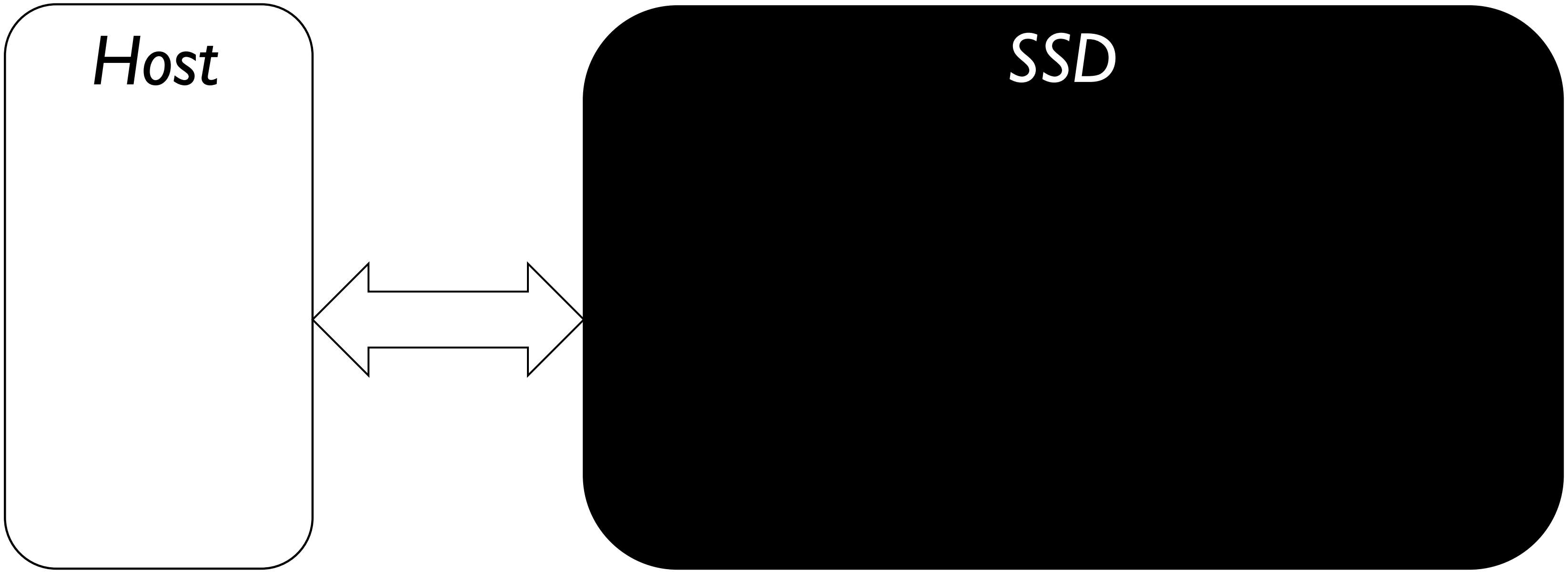
"Small but powerful"

IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage

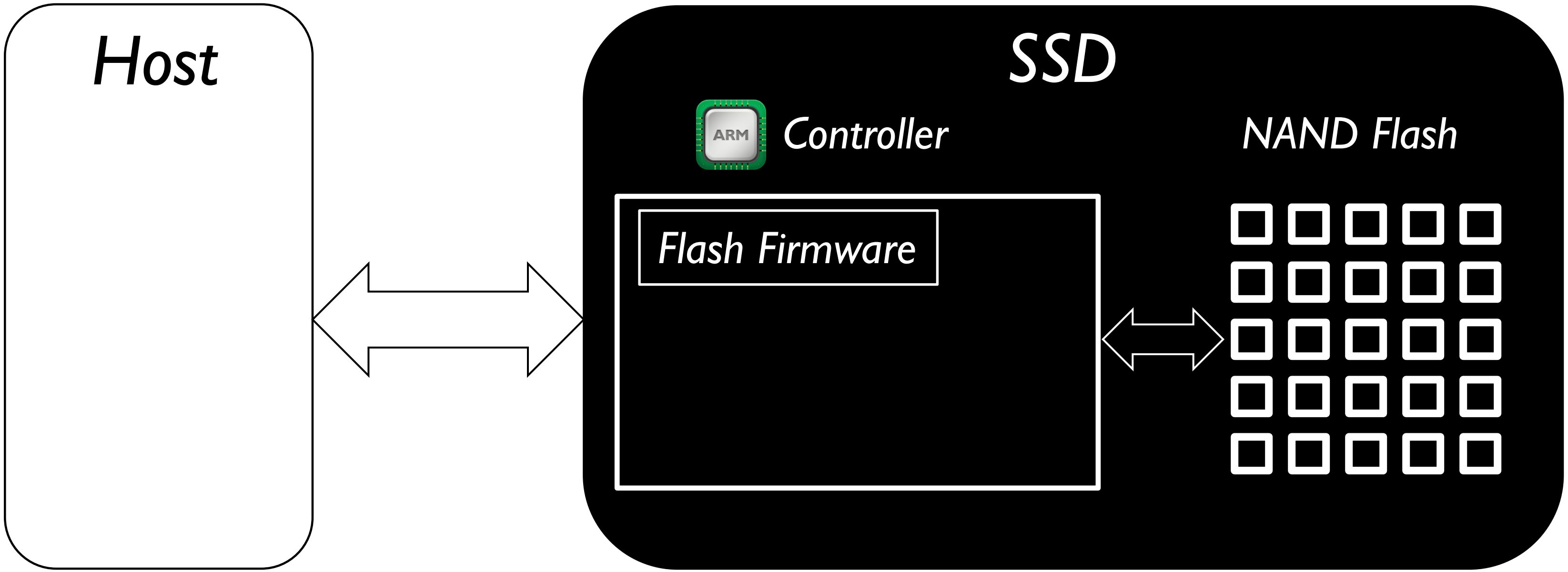


"Small but powerful"

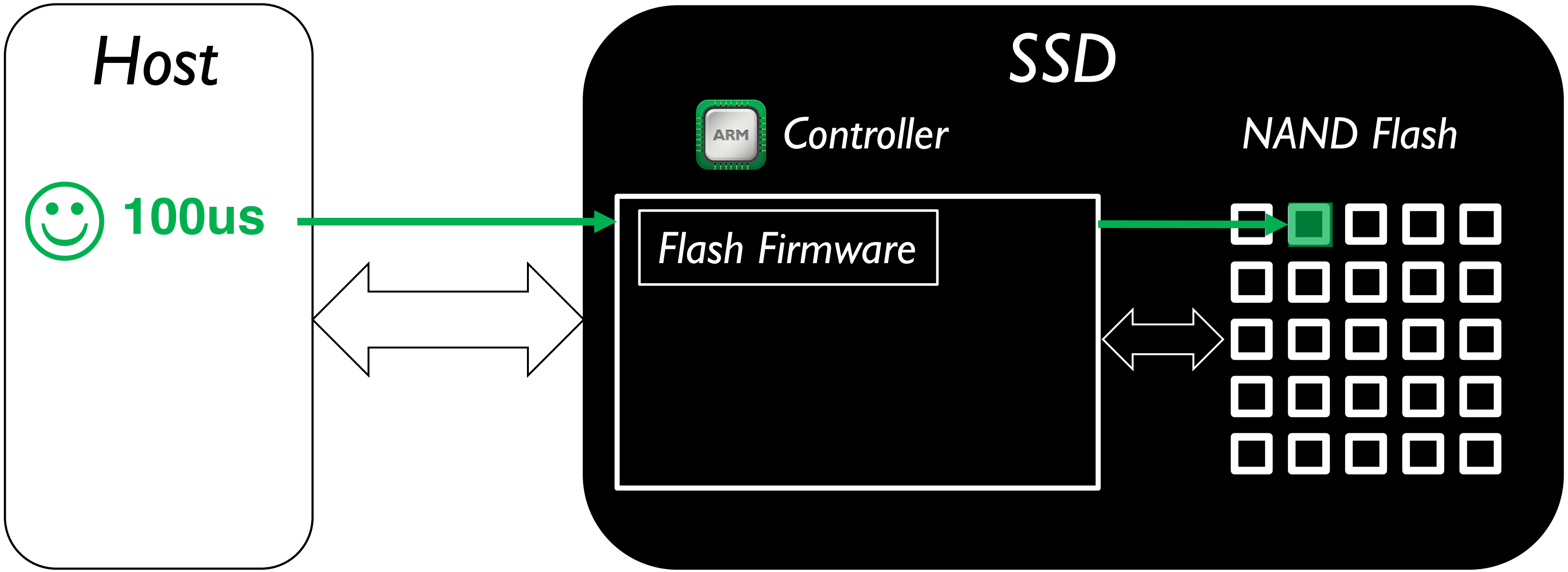
“Attack of GC” – Unpredictability in SSDs



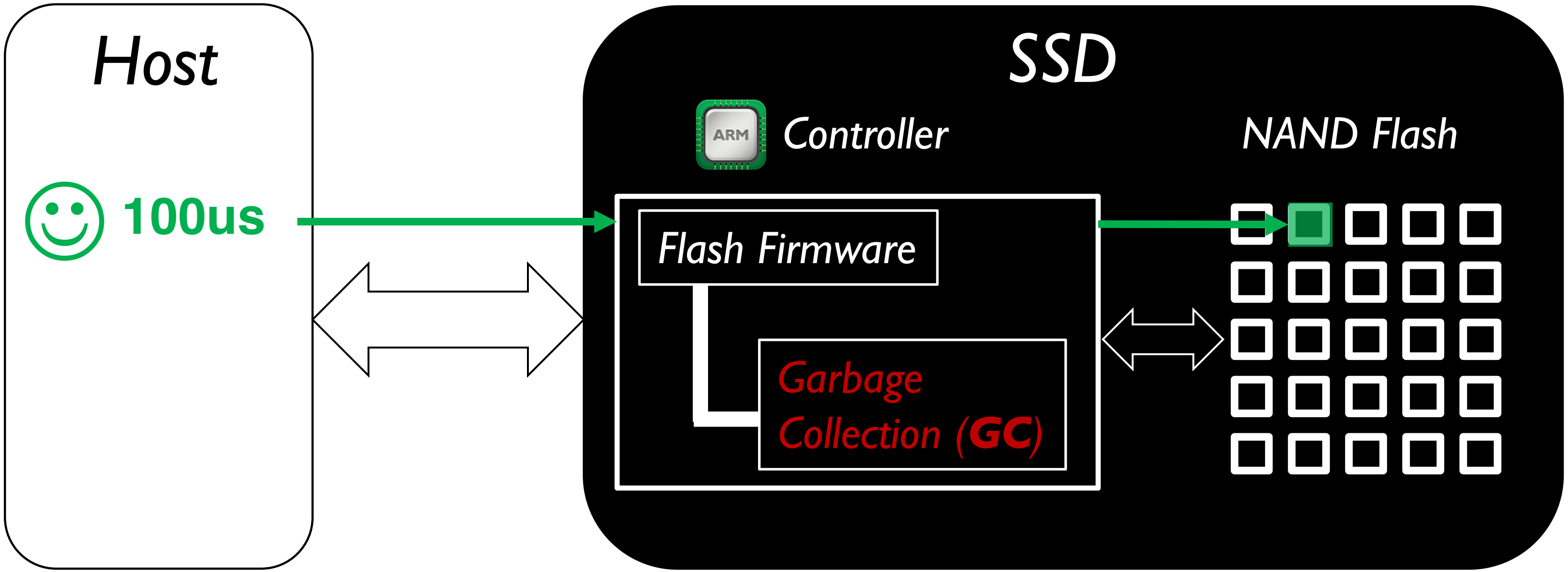
"Attack of GC" – Unpredictability in SSDs



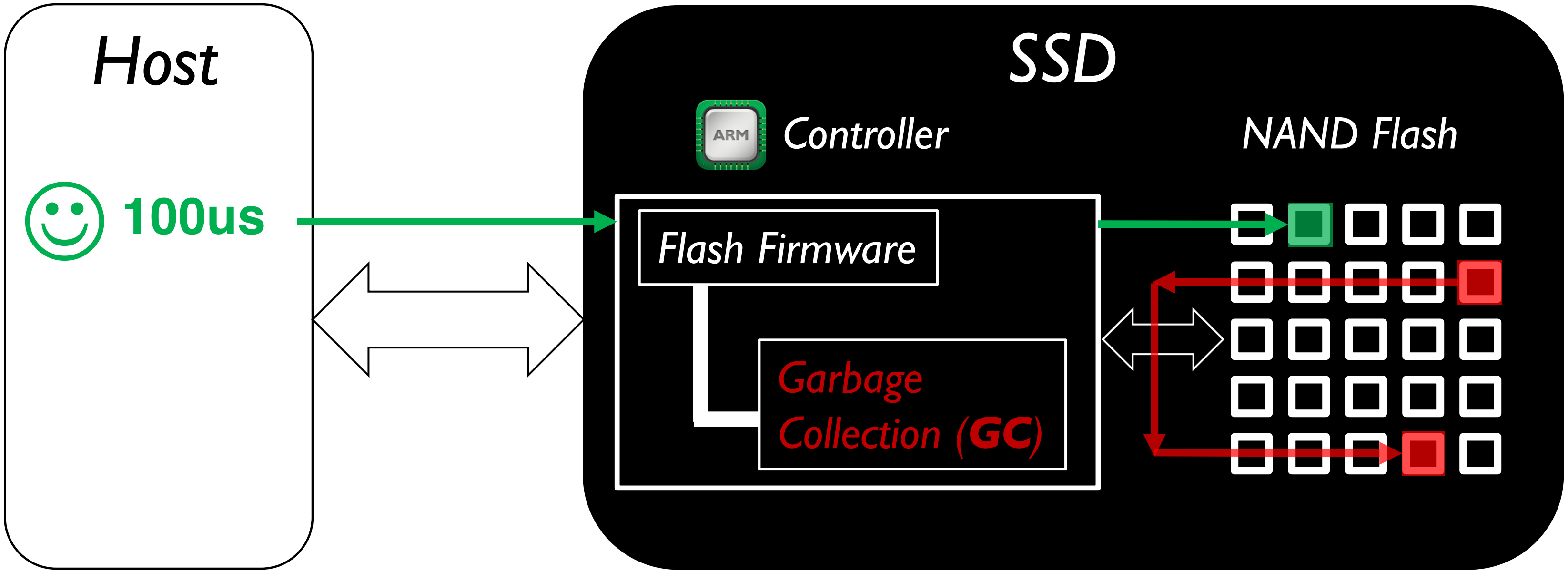
"Attack of GC" – Unpredictability in SSDs



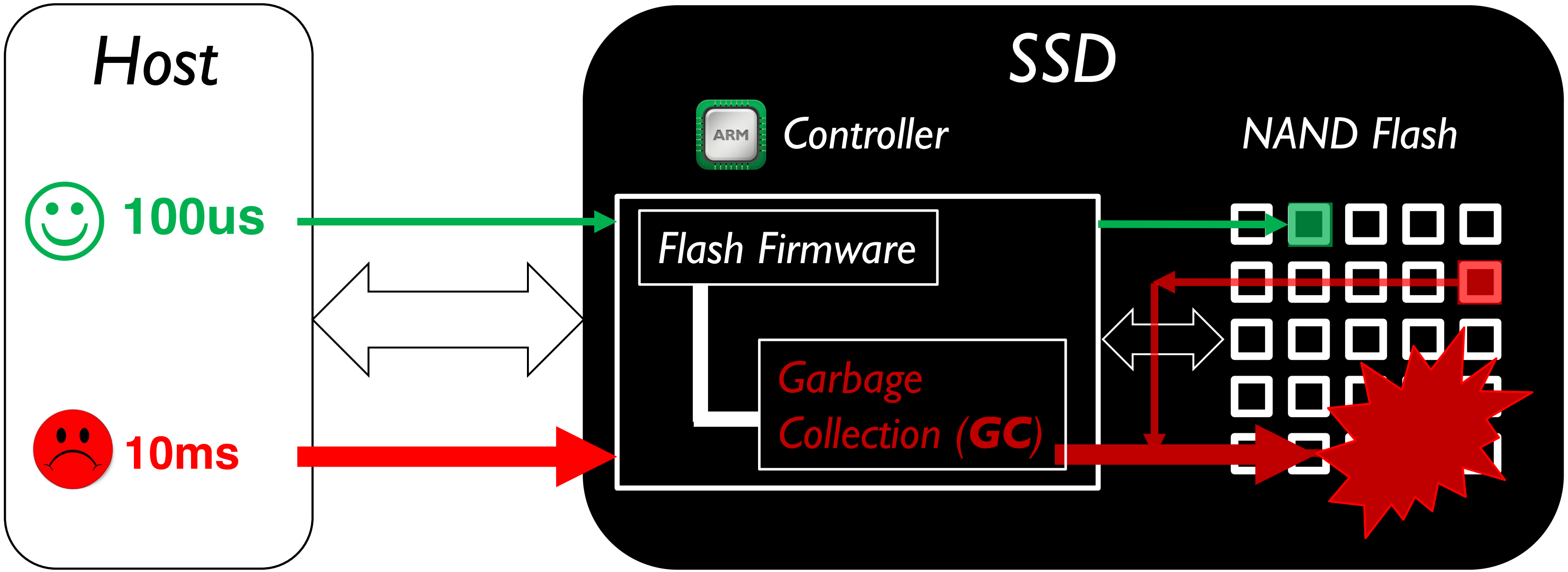
"Attack of GC" – Unpredictability in SSDs



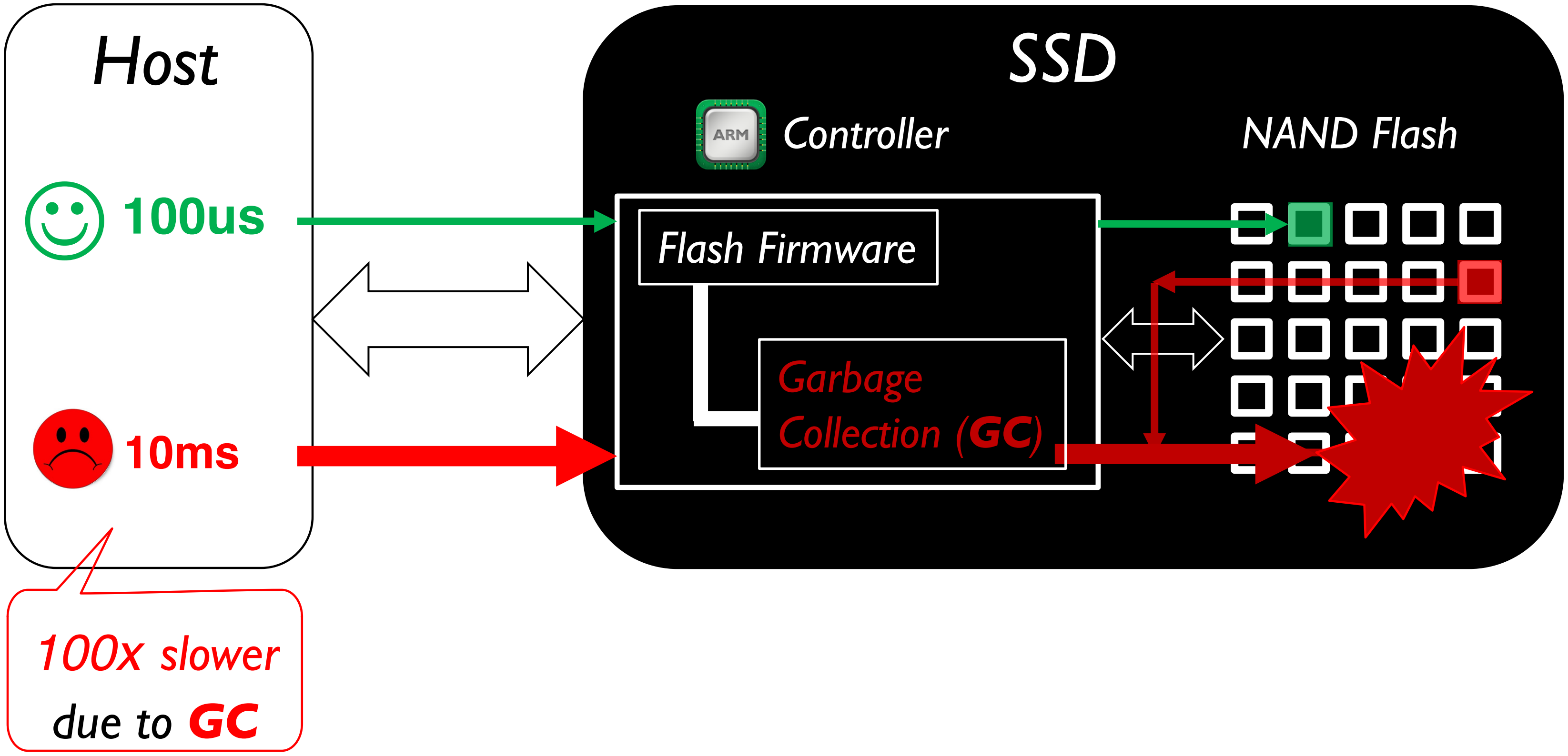
"Attack of GC" – Unpredictability in SSDs



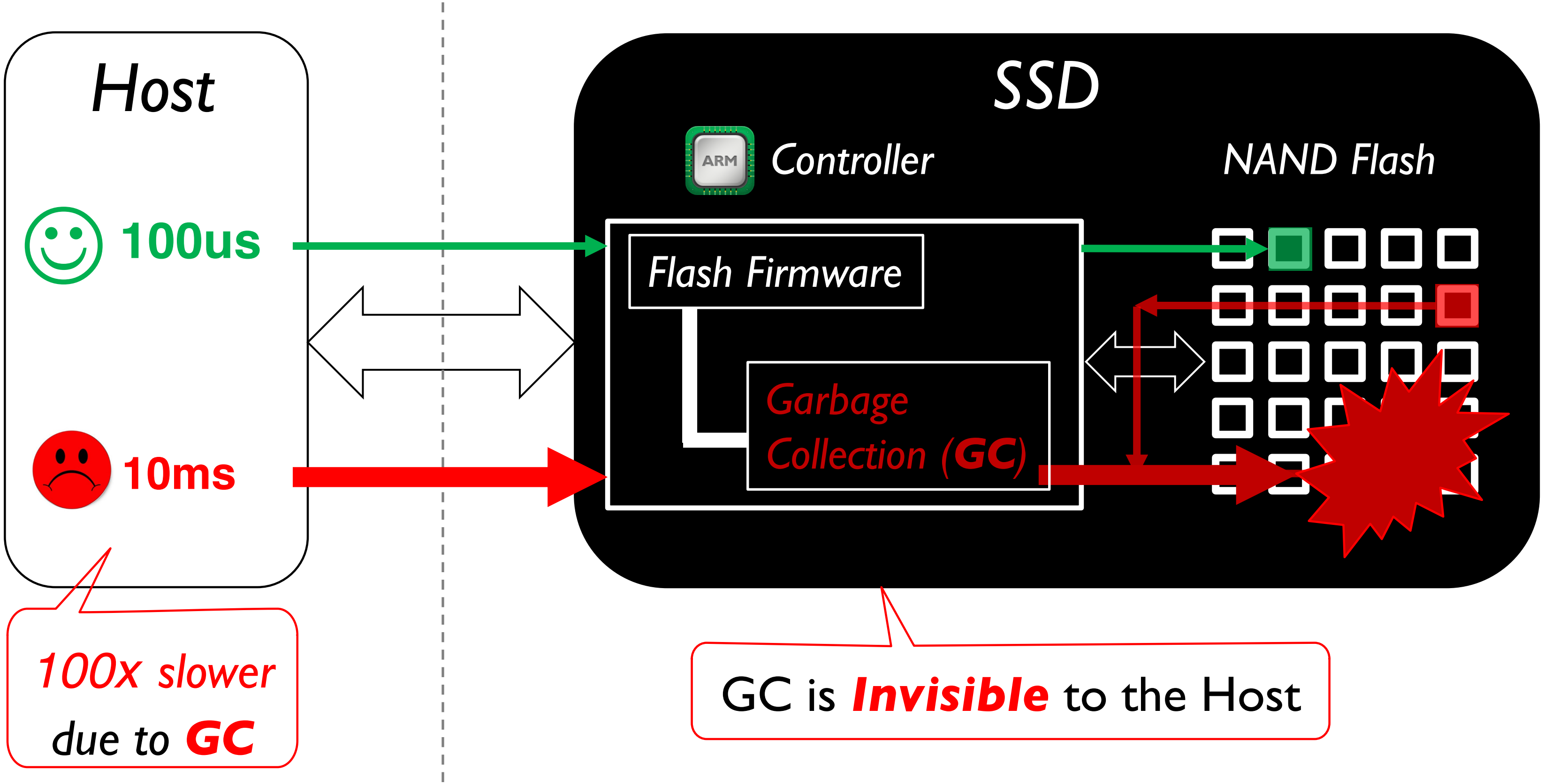
"Attack of GC" – Unpredictability in SSDs



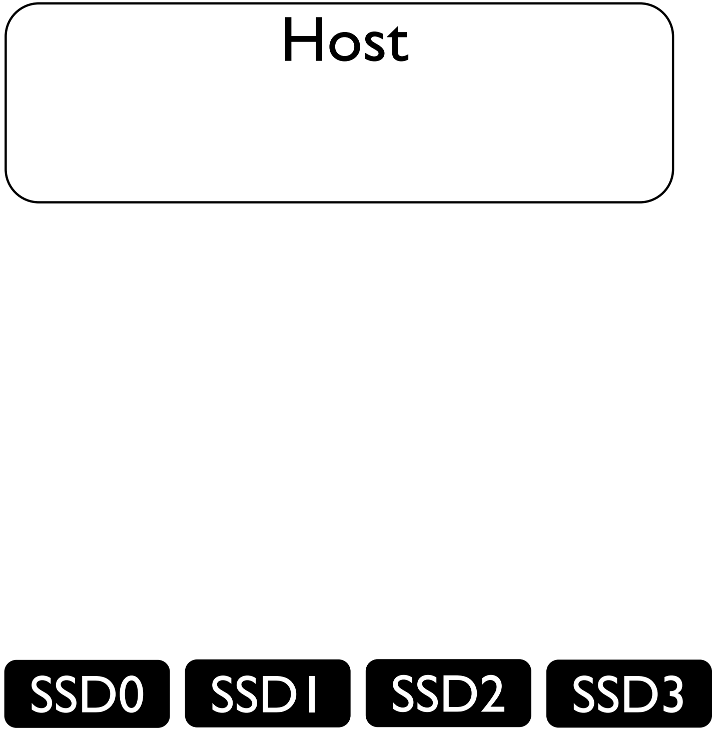
"Attack of GC" – Unpredictability in SSDs



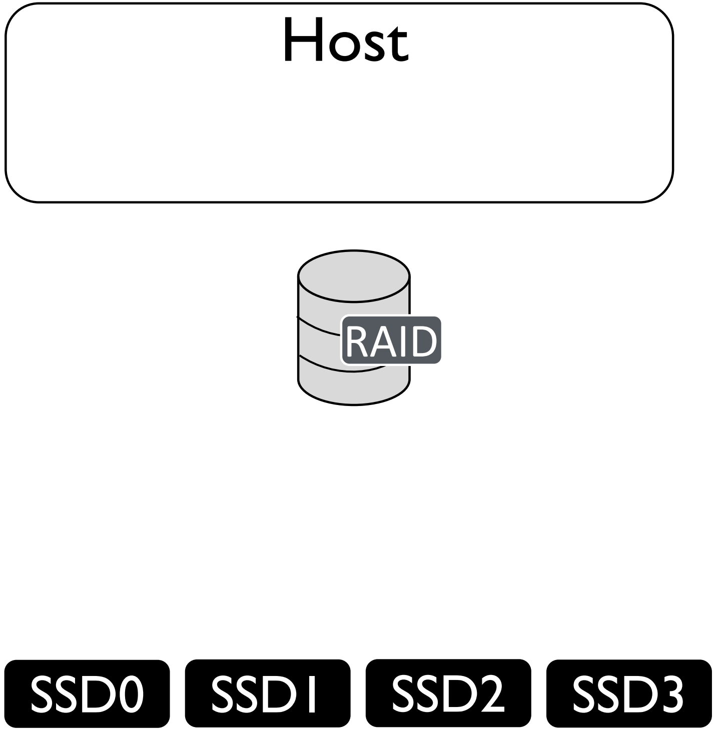
"Attack of GC" – Unpredictability in SSDs



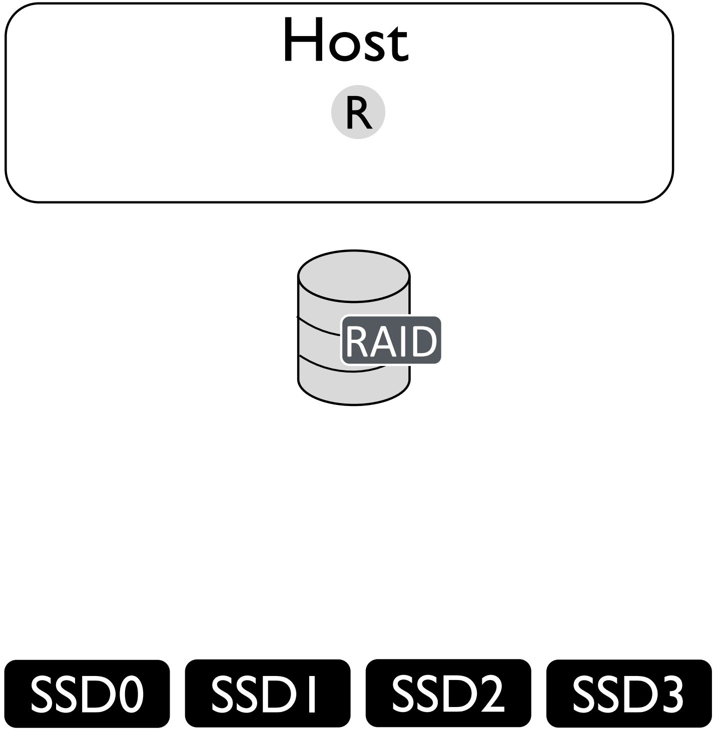
"The Tail Menace" in Flash Arrays



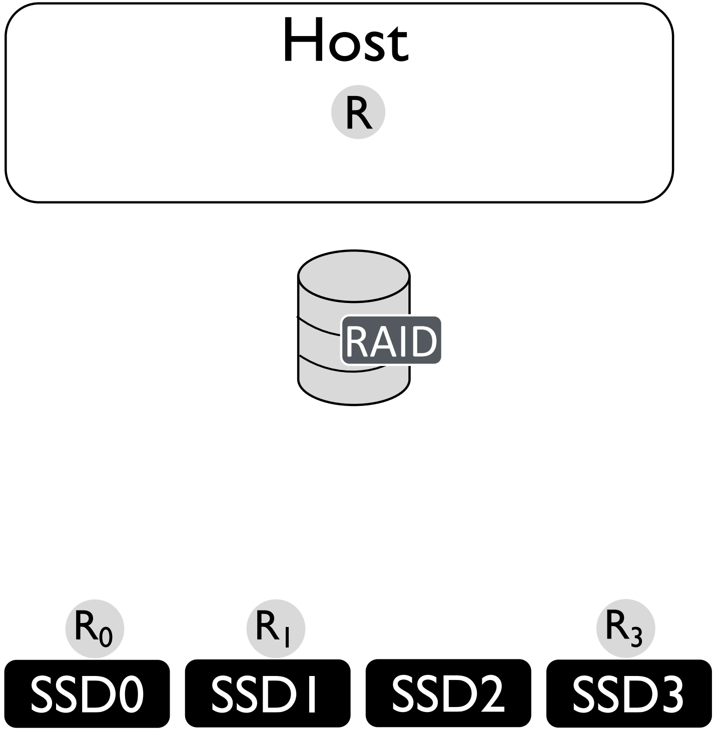
“*The Tail Menace*” in Flash Arrays



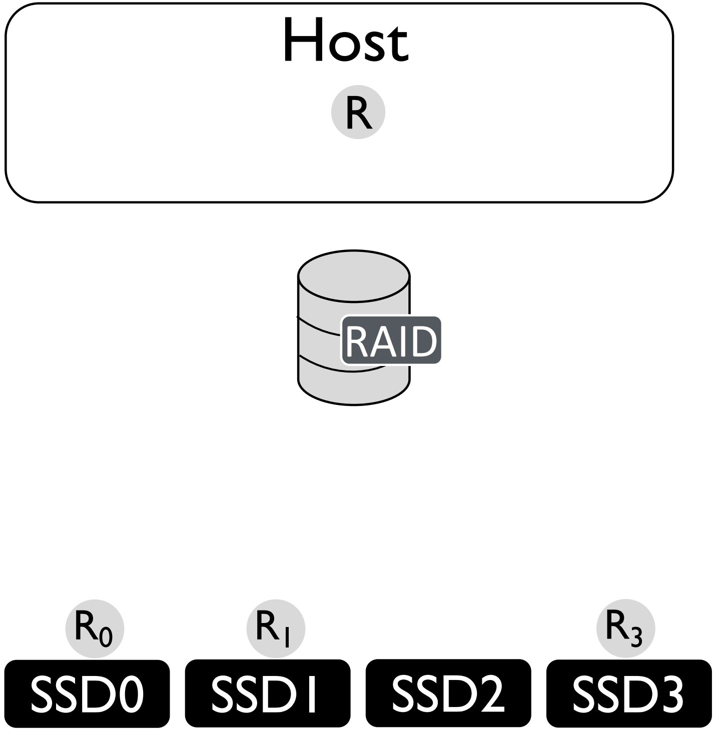
"The Tail Menace" in Flash Arrays



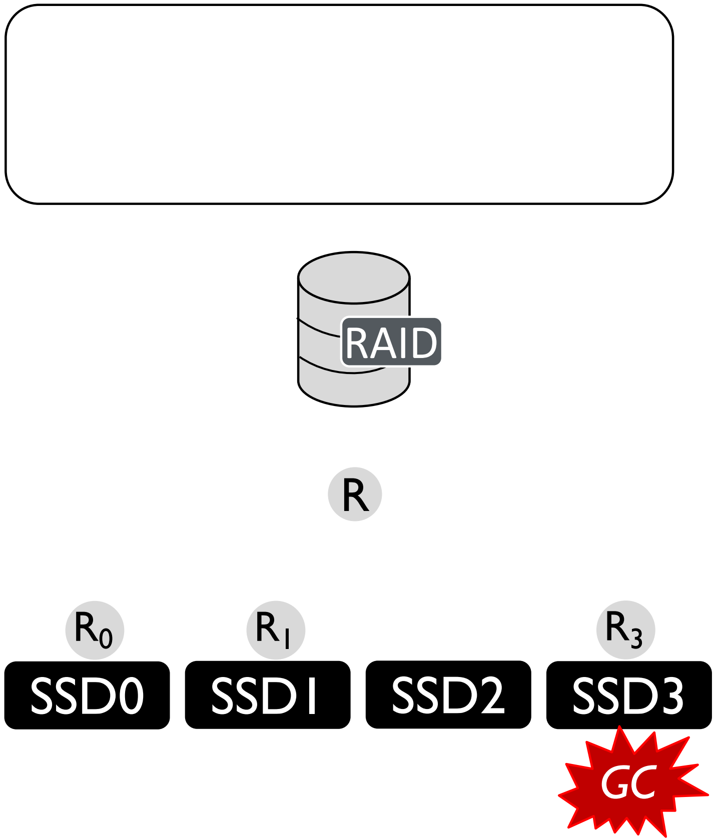
"The Tail Menace" in Flash Arrays

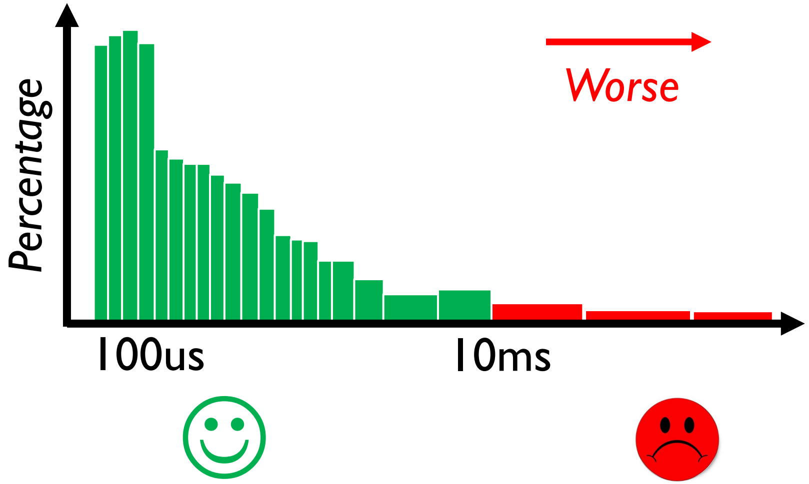


"The Tail Menace" in Flash Arrays

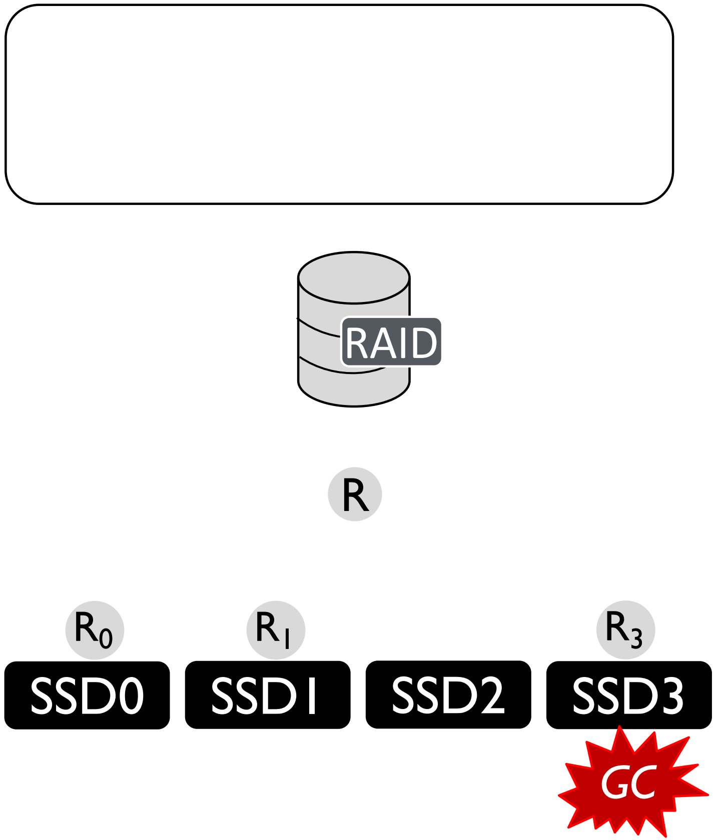


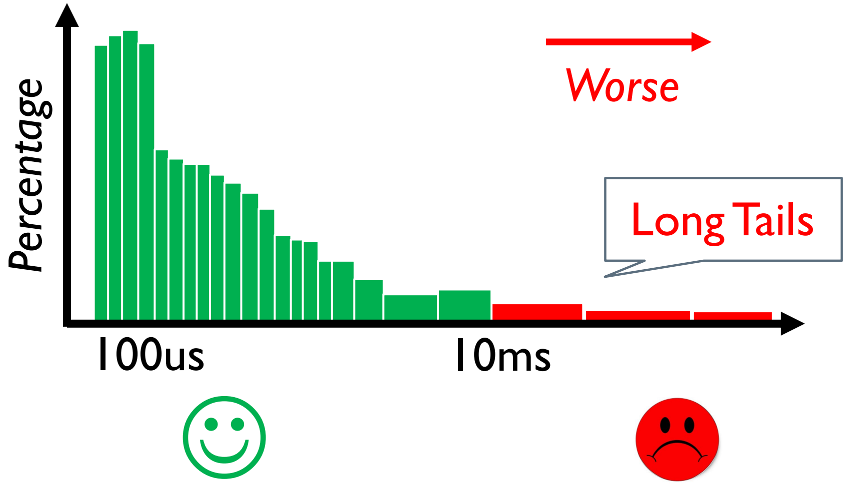
"The Tail Menace" in Flash Arrays



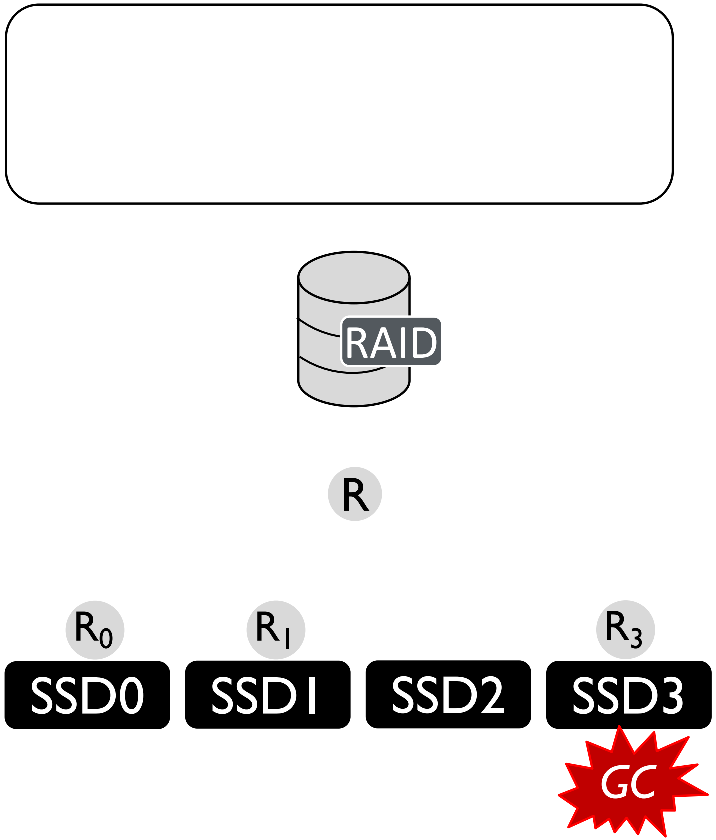


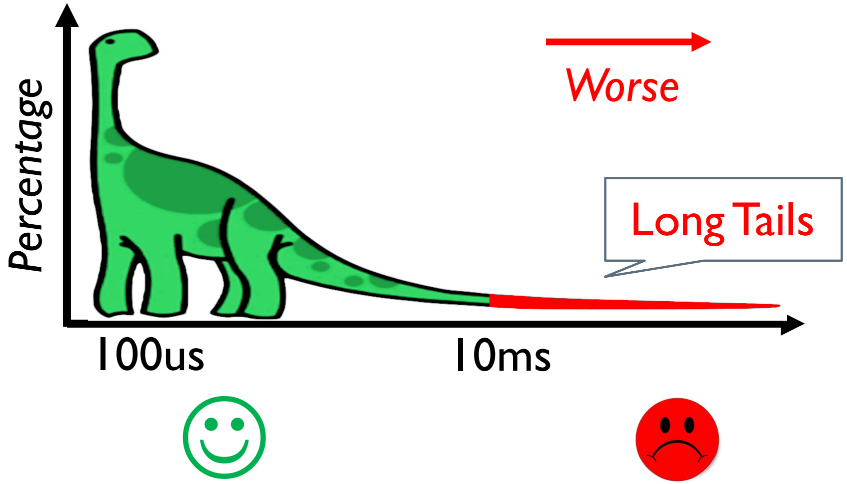
"The Tail Menace" in Flash Arrays



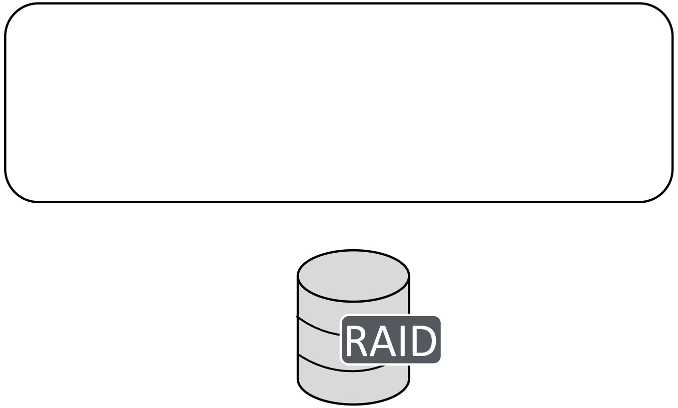


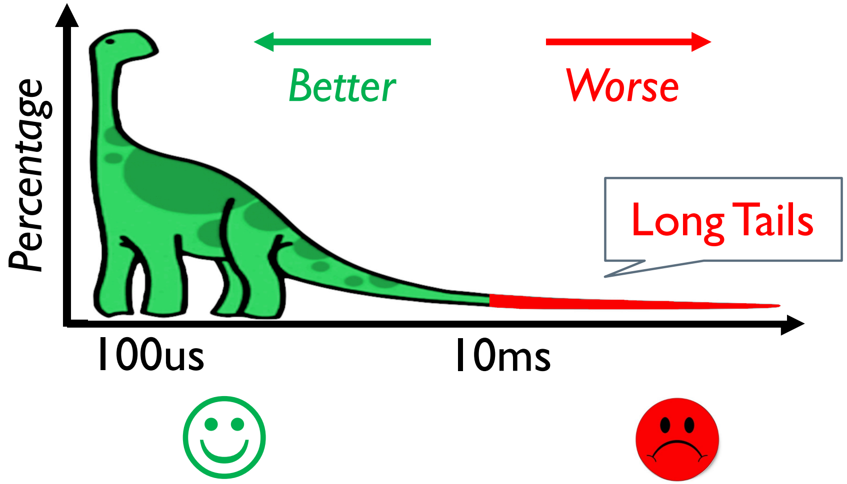
"The Tail Menace" in Flash Arrays





"The Tail Menace" in Flash Arrays





A slow SSD makes the entire flash array slow!

"A New Hope" – NVMe Predictable Latency Mode

NVMe Predictable Latency Mode (***PLM***)

"A New Hope" – NVMe Predictable Latency Mode

NVMe Predictable Latency Mode (***PLM***)

A major leap

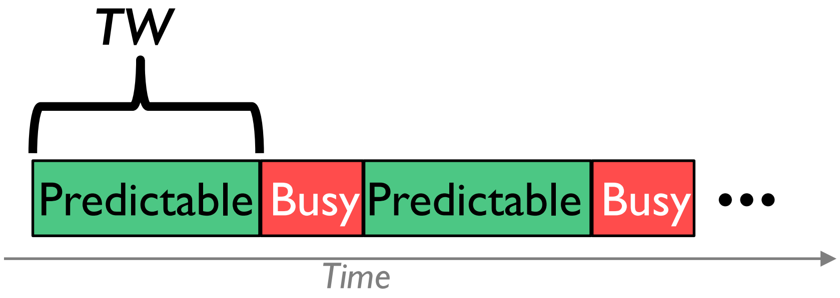
- ⊕ Predictable/Busy *Time Window (TW)*
- ⊕ Device status *query & toggling*

"A New Hope" – NVMe Predictable Latency Mode

NVMe Predictable Latency Mode (***PLM***)

A major leap

- Predictable/Busy *Time Window (TW)*
- Device status *query & toggling*

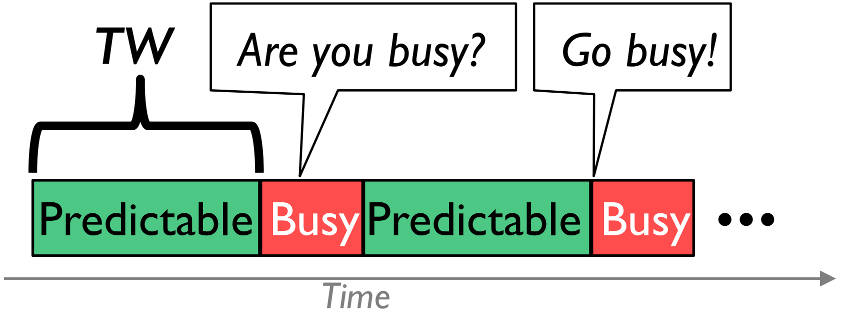


"A New Hope" – NVMe Predictable Latency Mode

NVMe Predictable Latency Mode (***PLM***)

A major leap

- ⊕ Predictable/Busy *Time Window (TW)*
- ⊕ Device status *query & toggling*

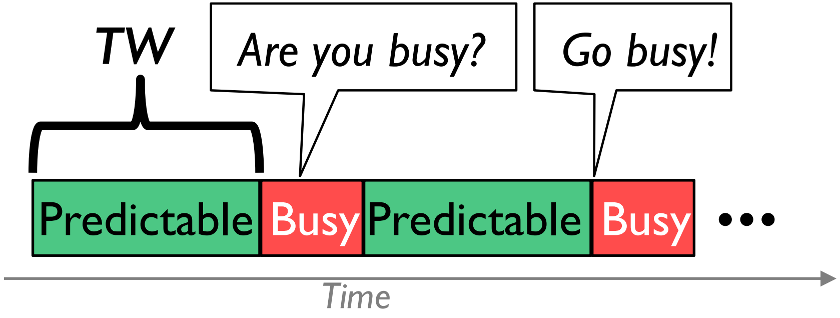


"A New Hope" – NVMe Predictable Latency Mode

NVMe Predictable Latency Mode (***PLM***)

A major leap

- ＋ Predictable/Busy *Time Window (TW)*
- ＋ Device status *query & toggling*



But insufficient

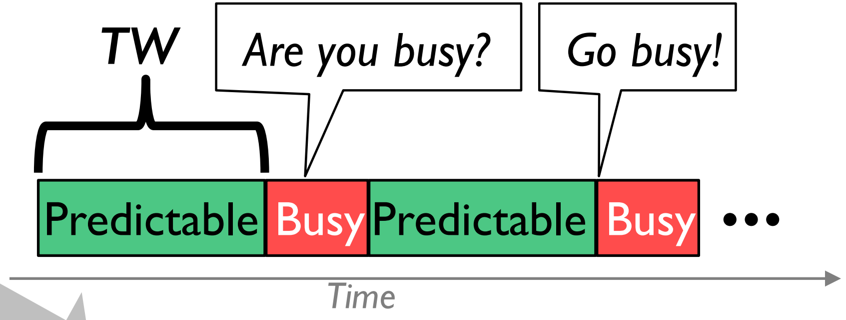
- – *Coarse-grained* device-level predictability
- – *"Soft-contract"* breaking predictability
- – Requiring *complex* status tracking
- – •••

"A New Hope" – NVMe Predictable Latency Mode

NVMe Predictable Latency Mode (***PLM***)

A major leap

- Predictable/Busy *Time Window (TW)*
- Device status *query & toggling*



But insufficient

- *Coarse-grained* device-level predictability
- *"Soft-contract"* breaking predictability
- Requiring *complex* status tracking
- …

How to leverage NVMe PLM and enhance it for predictable latencies?

The IODA Story

- **Goal**: ***Tail-free*** flash array system on top of *slightly-extended* PLM interface

The IODA Story

- **Goal**: ***Tail-free*** flash array system on top of *slightly-extended* PLM interface

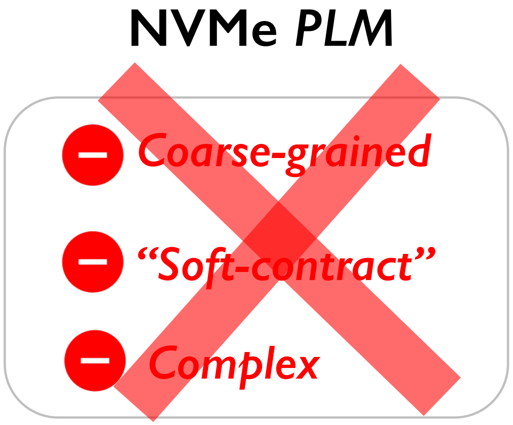
- Design Principles:
  - ***Simple*** policies for efficiency
  - ***Minimal changes*** for easy deployment

The IODA Story

- **Goal**: ***Tail-free*** flash array system on top of *slightly-extended* PLM interface

- Design Principles:
  - ***Simple*** policies for efficiency
  - ***Minimal changes*** for easy deployment

- IODA Approach/Techniques:



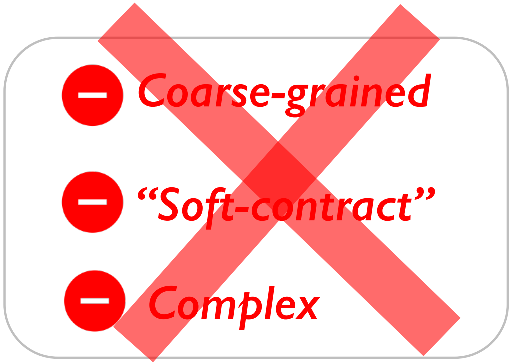
The IODA Story

- ❑ **Goal**: ***Tail-free*** flash array system on top of *slightly-extended* PLM interface

- ❑ Design Principles:
  - ***Simple*** policies for efficiency
  - ***Minimal changes*** for easy deployment

- ❑ IODA Approach/Techniques:
  - ✚ *Per-I/O* latency predictability
  - ✚ Busy Remaining Time (*BRT*) Exposure

NVMe *PLM*



The IODA Story

- **Goal**: ***Tail-free*** flash array system on top of *slightly-extended* PLM interface

- Design Principles:
  - ***Simple*** policies for efficiency
  - ***Minimal changes*** for easy deployment

- IODA Approach/Techniques:
  - + *Per-I/O* latency predictability
  - + Busy Remaining Time (*BRT*) Exposure
  - + Time Window (*TW*) Formulation

NVMe *PLM*

- − *Coarse-grained*
- − *"Soft-contract"*
- − *Complex*

The IODA Story

- Goal: ***Tail-free*** flash array system on top of *slightly-extended* PLM interface

- Design Principles:
  - ***Simple*** policies for efficiency
  - ***Minimal changes*** for easy deployment

- IODA Approach/Techniques:
  - *Per-I/O* latency predictability
  - Busy Remaining Time (*BRT*) Exposure
  - **Time Window (*TW*) Formulation**
  - An end-to-end design exploiting above extensions

NVMe *PLM*

- *Coarse-grained*
- *"Soft-contract"*
- *Complex*

- Background & Motivation
- IODA Overview
- IODA Design
  - Predictable latency flagged I/Os
  - Busy remaining time
  - Time window formulation
  - Relaxed TW for better write amplification
- Evaluation
- Summary

Leverage Redundancy for Performance

An old, effective idea;

Leverage Redundancy for Performance

An old, effective idea;

Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs

Trimming the Tail for Deterministic Read Performance in SSDs

Latency Reduction and Load Balancing in Coded Storage Systems

1

RAIL: Predictable, Low Tail Latency for NVMe Flash

MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface

EC-Cache: Load-balanced, Low-latency Cluster Caching with Online Erasure Coding

K. V. Rashmi[1], Mosharaf Chowdhury[2], Jack Kosaian[2], Ion Stoica[1], Kannan Ramchandran[1]
[1] *UC Berkeley* [2] *University of Michigan*

Abstract

Data-intensive clusters and object stores are increasingly relying on in-memory object caching to meet the I/O performance demands. These systems routinely face the challenges of popularity skew, background load imbalance, and server failures, which result in severe load imbalance across servers and degraded I/O performance. Selective replication is a commonly used technique to tackle these challenges, where the number of cached replicas of an object is proportional to its popularity. In this paper, we explore an alternative approach using erasure coding.

EC-Cache is a load-balanced, low latency cluster cache that uses online erasure coding to overcome the limitations of selective replication. EC-Cache employs erasure coding by: (i) splitting and erasure coding individual objects during writes, and (ii) late binding, wherein obtaining any k out of $(k + r)$ splits of an object are sufficient, during reads. As compared to selective replication, EC-Cache improves load balancing by more than 3× and reduces the median and tail read latencies by more than 2×, while using the same amount of memory. EC-Cache does so using 10% additional bandwidth and a small increase in the amount of stored metadata. The benefits offered by EC-Cache are further amplified in the presence of background network load imbalance

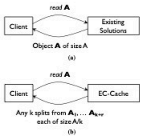


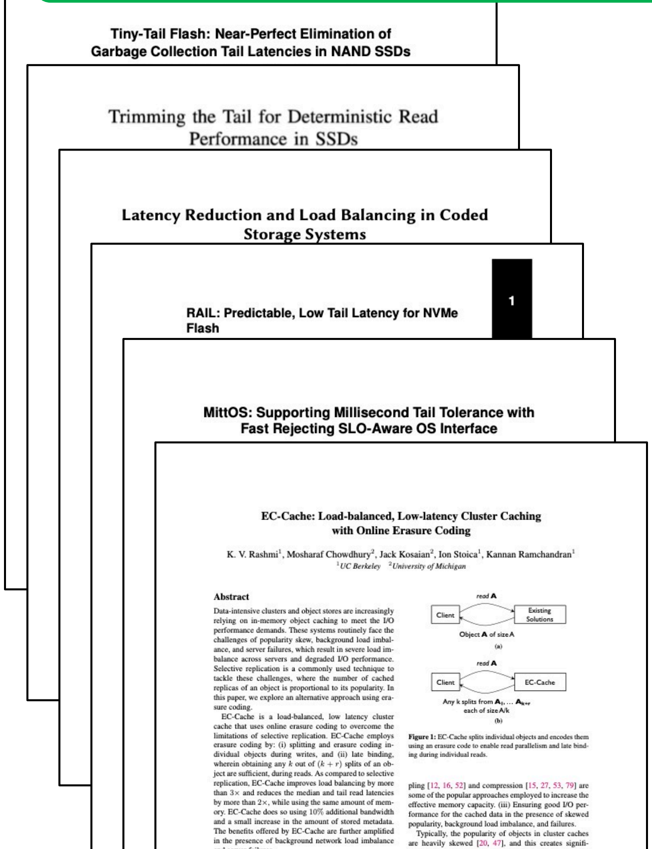
Figure 1: EC-Cache splits individual objects and encodes them using an erasure code to enable read parallelism and late binding during individual reads.

pling [12, 16, 52] and compression [15, 27, 53, 79] are some of the popular approaches employed to increase the effective memory capacity. (iii) Ensuring good I/O performance for the cached data in the presence of skewed popularity, background load imbalance, and failures.

Typically, the popularity of objects in cluster caches are heavily skewed [20, 47], and this creates signifi-

Leverage Redundancy for Performance

An old, effective idea; *Yet, challenging for PLM*



Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs

Trimming the Tail for Deterministic Read Performance in SSDs

Latency Reduction and Load Balancing in Coded Storage Systems

RAIL: Predictable, Low Tail Latency for NVMe Flash

1

MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface

EC-Cache: Load-balanced, Low-latency Cluster Caching with Online Erasure Coding

K. V. Rashmi[1], Mosharaf Chowdhury[2], Jack Kosaian[2], Ion Stoica[1], Kannan Ramchandran[1]
[1]*UC Berkeley* [2]*University of Michigan*

Abstract

Data-intensive clusters and object stores are increasingly relying on in-memory object caching to meet the I/O performance demands. These systems routinely face the challenges of popularity skew, background load imbalance, and server failures, which result in severe load imbalance across servers and degraded I/O performance. Selective replication is a commonly used technique to tackle these challenges, where the number of cached replicas of an object is proportional to its popularity. In this paper, we explore an alternative approach using erasure coding.

EC-Cache is a load-balanced, low latency cluster cache that uses online erasure coding to overcome the limitations of selective replication. EC-Cache employs erasure coding by: (i) splitting and erasure coding individual objects during writes, and (ii) late binding, wherein obtaining any k out of $(k+r)$ splits of an object are sufficient, during reads. As compared to selective replication, EC-Cache improves load balancing by more than 3× and reduces the median and tail read latencies by more than 2×, while using the same amount of memory. EC-Cache does so using 10% additional bandwidth and a small increase in the amount of stored metadata. The benefits offered by EC-Cache are further amplified in the presence of background network load imbalance

Figure 1: EC-Cache splits individual objects and encodes them using an erasure code to enable read parallelism and late binding during individual reads.

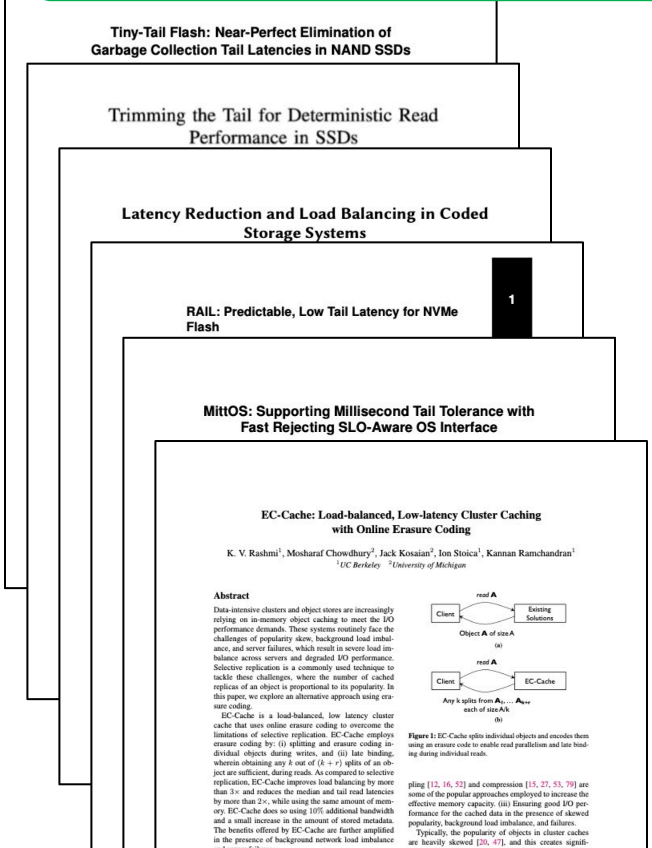
pling [12, 16, 52] and compression [15, 27, 53, 79] are some of the popular approaches employed to increase the effective memory capacity. (iii) Ensuring good I/O performance for the cached data in the presence of skewed popularity, background load imbalance, and failures.

Typically, the popularity of objects in cluster caches are heavily skewed [20, 47], and this creates signifi-

When to issue the parity reads?

Leverage Redundancy for Performance

An old, effective idea; **Yet, challenging for PLM**


Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs

Trimming the Tail for Deterministic Read Performance in SSDs

Latency Reduction and Load Balancing in Coded Storage Systems

RAIL: Predictable, Low Tail Latency for NVMe Flash

MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface

EC-Cache: Load-balanced, Low-latency Cluster Caching with Online Erasure Coding

K. V. Rashmi[1], Mosharaf Chowdhury[2], Jack Kosaian[2], Ion Stoica[1], Kannan Ramchandran[1]
[1]UC Berkeley [2]University of Michigan

Abstract

Data-intensive clusters and object stores are increasingly relying on in-memory object caching to meet the I/O performance demands. These systems routinely face the challenges of popularity skew, background load imbalance, and server failures, which result in severe load imbalance across servers and degraded I/O performance. Selective replication is a commonly used technique to tackle these challenges, where the number of cached replicas of an object is proportional to its popularity. In this paper, we explore an alternative approach using erasure coding.

EC-Cache is a load-balanced, low latency cluster cache that uses online erasure coding to overcome the limitations of selective replication. EC-Cache employs erasure coding by: (i) splitting and erasure coding individual objects during writes, and (ii) late binding, wherein obtaining any k out of $(k+r)$ splits of an object are sufficient, during reads. As compared to selective replication, EC-Cache improves load balancing by more than 3× and reduces the median and tail read latencies by more than 2×, while using the same amount of memory. EC-Cache does so using 10% additional bandwidth and a small increase in the amount of stored metadata. The benefits offered by EC-Cache are further amplified in the presence of background network load imbalance

Figure 1: EC-Cache splits individual objects and encodes them using an erasure code to enable read parallelism and late binding during individual reads.

pling [12, 16, 52] and compression [15, 27, 53, 79] are some of the popular approaches employed to increase the effective memory capacity. (iii) Ensuring good I/O performance for the cached data in the presence of skewed popularity, background load imbalance, and failures.

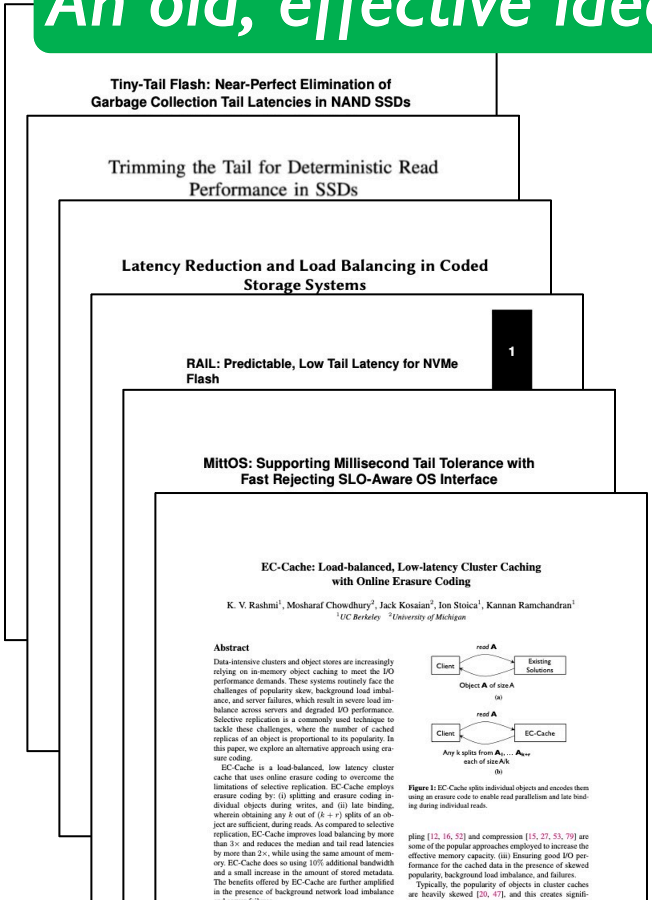
Typically, the popularity of objects in cluster caches are heavily skewed [20, 47], and this creates signifi-

When to issue the parity reads?

(1) *Wait* for timeout

Leverage Redundancy for Performance

An old, effective idea; **Yet, challenging for PLM**


Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs

Trimming the Tail for Deterministic Read Performance in SSDs

Latency Reduction and Load Balancing in Coded Storage Systems

RAIL: Predictable, Low Tail Latency for NVMe Flash

MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface

EC-Cache: Load-balanced, Low-latency Cluster Caching with Online Erasure Coding

K. V. Rashmi[1], Mosharaf Chowdhury[2], Jack Kosaian[2], Ion Stoica[1], Kannan Ramchandran[1]
[1]UC Berkeley [2]University of Michigan

Abstract

Data-intensive clusters and object stores are increasingly relying on in-memory object caching to meet the I/O performance demands. These systems routinely face the challenges of popularity skew, background load imbalance, and server failures, which result in severe load imbalance across servers and degraded I/O performance. Selective replication is a commonly used technique to tackle these challenges, where the number of cached replicas of an object is proportional to its popularity. In this paper, we explore an alternative approach using erasure coding.

EC-Cache is a load-balanced, low latency cluster cache that uses online erasure coding to overcome the limitations of selective replication. EC-Cache employs erasure coding by: (i) splitting and erasure coding individual objects during writes, and (ii) late binding, wherein obtaining any k out of $(k+r)$ splits of an object are sufficient, during reads. As compared to selective replication, EC-Cache improves load balancing by more than 3× and reduces the median and tail read latencies by more than 2×, while using the same amount of memory. EC-Cache does so using 10% additional bandwidth and a small increase in the amount of stored metadata. The benefits offered by EC-Cache are further amplified in the presence of background network load imbalance

Figure 1: EC-Cache splits individual objects and encodes them using an erasure code to enable read parallelism and late binding during individual reads.

pling [12, 16, 52] and compression [15, 27, 53, 79] are some of the popular approaches employed to increase the effective memory capacity. (iii) Ensuring good I/O performance for the cached data in the presence of skewed popularity, background load imbalance, and failures.

Typically, the popularity of objects in cluster caches are heavily skewed [20, 47], and this creates signifi-

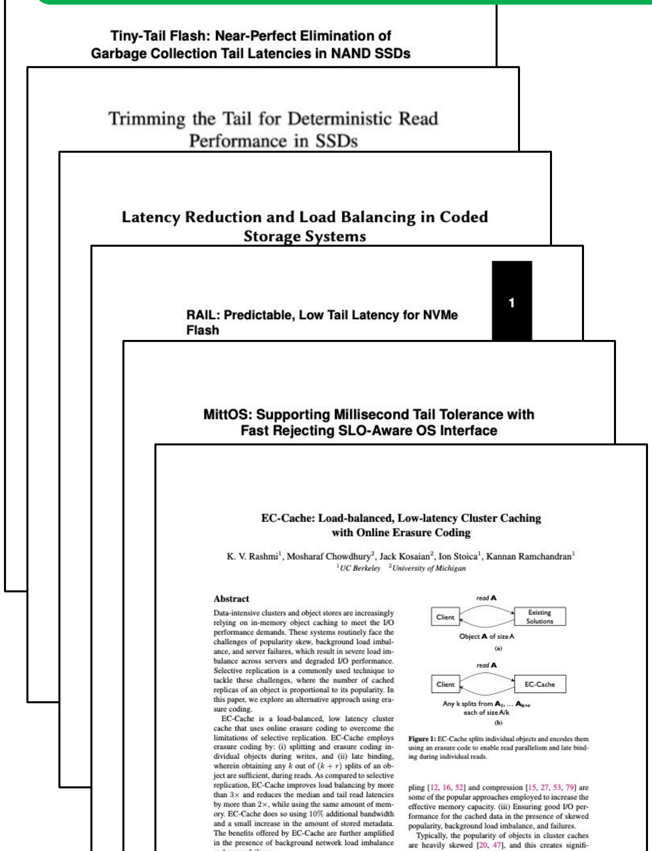
When to issue the parity reads?

(1) *Wait* for timeout

− Best threshold? *Tricky*

Leverage Redundancy for Performance

An old, effective idea; **Yet, challenging for PLM**



When to issue the parity reads?

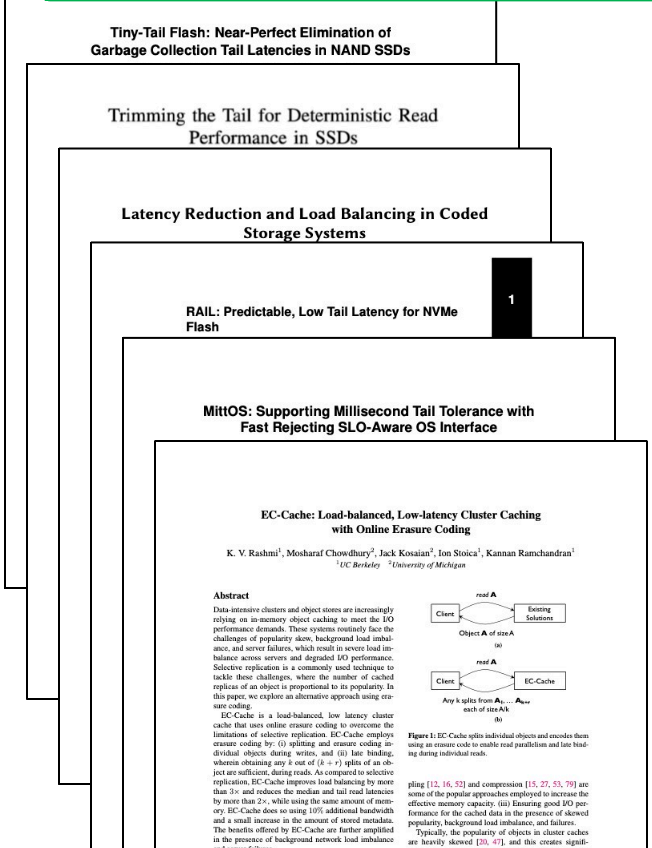
(1) *Wait* for timeout

− Best threshold? *Tricky*

(2) *Always* Proactive (always send full-stripe)

Leverage Redundancy for Performance

An old, effective idea; **Yet, challenging for PLM**



When to issue the parity reads?

(1) *Wait* for timeout

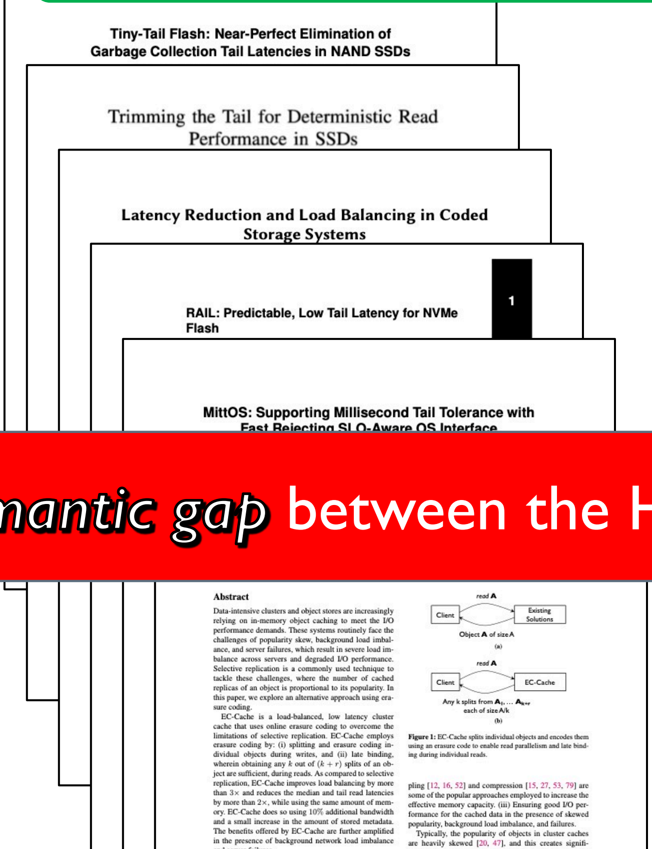
− Best threshold? *Tricky*

(2) *Always* Proactive (always send full-stripe)

− Increased load ⟶ *Inefficient*

Leverage Redundancy for Performance

An old, effective idea; ***Yet, challenging for PLM***



Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs

Trimming the Tail for Deterministic Read Performance in SSDs

Latency Reduction and Load Balancing in Coded Storage Systems

RAIL: Predictable, Low Tail Latency for NVMe Flash

1

MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface

Abstract

Data-intensive clusters and object stores are increasingly relying on in-memory object caching to meet the I/O performance demands. These systems routinely face the challenges of popularity skew, background load imbalance, and server failures, which result in severe load imbalance across servers and degraded I/O performance. Selective replication is a commonly used technique to tackle these challenges, where the number of cached replicas of an object is proportional to its popularity. In this paper, we explore an alternative approach using erasure coding.

EC-Cache is a load-balanced, low latency cluster cache that uses online erasure coding to overcome the limitations of selective replication. EC-Cache employs erasure coding by: (i) splitting and erasure coding individual objects during writes, and (ii) late binding, wherein obtaining any k out of $(k+r)$ splits of an object are sufficient, during reads. As compared to selective replication, EC-Cache improves load balancing by more than 3× and reduces the median and tail read latencies by more than 2×, while using the same amount of memory. EC-Cache does so using 10% additional bandwidth and a small increase in the amount of stored metadata. The benefits offered by EC-Cache are further amplified in the presence of background network load imbalance

Figure 1: EC-Cache splits individual objects and encodes them using an erasure code to enable read parallelism and late binding during individual reads.

pling [12, 16, 52] and compression [15, 27, 53, 79] are some of the popular approaches employed to increase the effective memory capacity. (iii) Ensuring good I/O performance for the cached data in the presence of skewed popularity, background load imbalance, and failures.

Typically, the popularity of objects in cluster caches are heavily skewed [20, 47], and this creates signifi-

When to issue the parity reads?

- (1) *Wait* for timeout
  - Best threshold? *Tricky*
- (2) *Always* Proactive (always send full-stripe)
  - Increased load ⟶ *Inefficient*

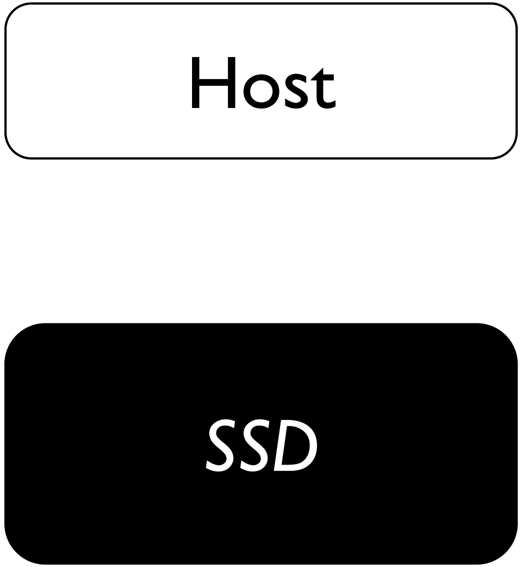
Semantic gap **between the Host and SSD to communicate the "*busyness*"**

IOD_1: Predictable Latency Flagged I/Os

- ***"Fail-if-Slow":*** the SSD should *fast-fail* an I/O if it contends with GC

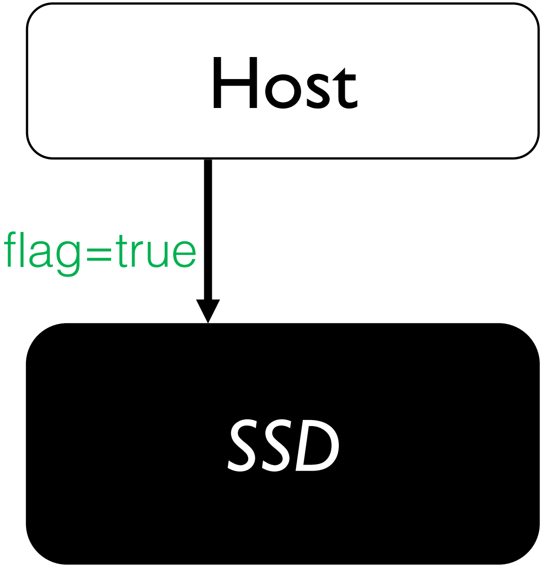
IOD$_1$: Predictable Latency Flagged I/Os

"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC



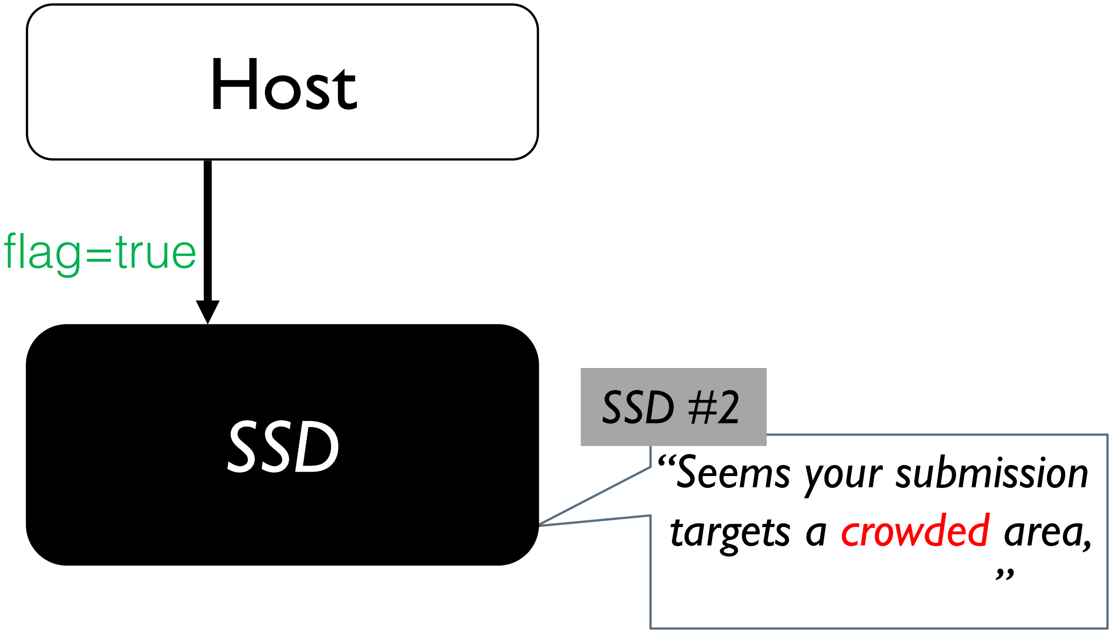
IOD$_1$: Predictable Latency Flagged I/Os

"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC



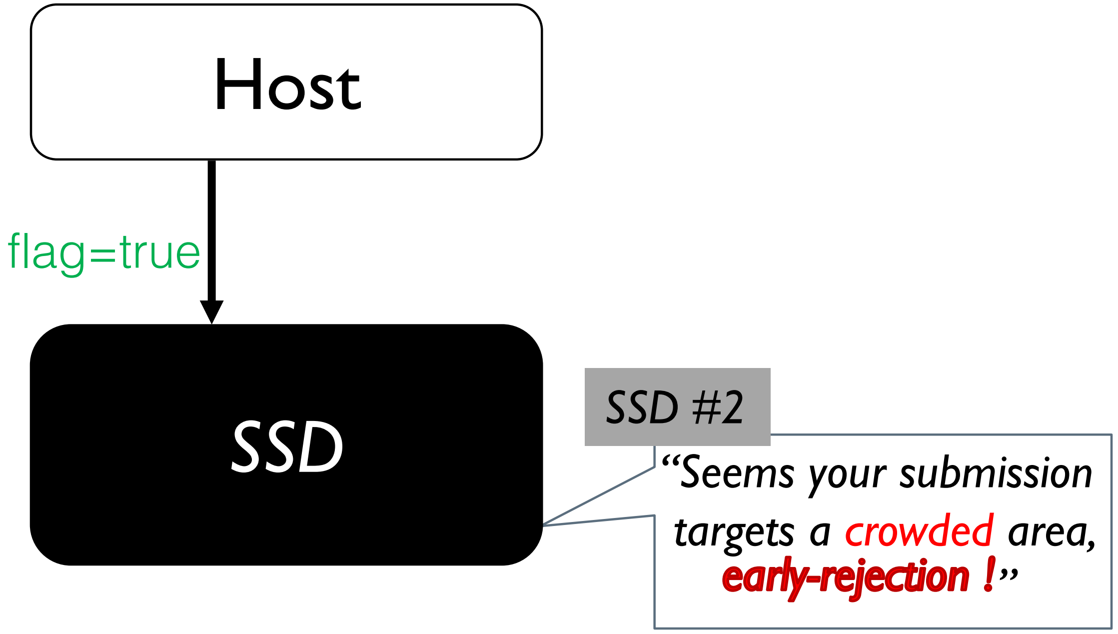
IOD$_1$: Predictable Latency Flagged I/Os

"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC



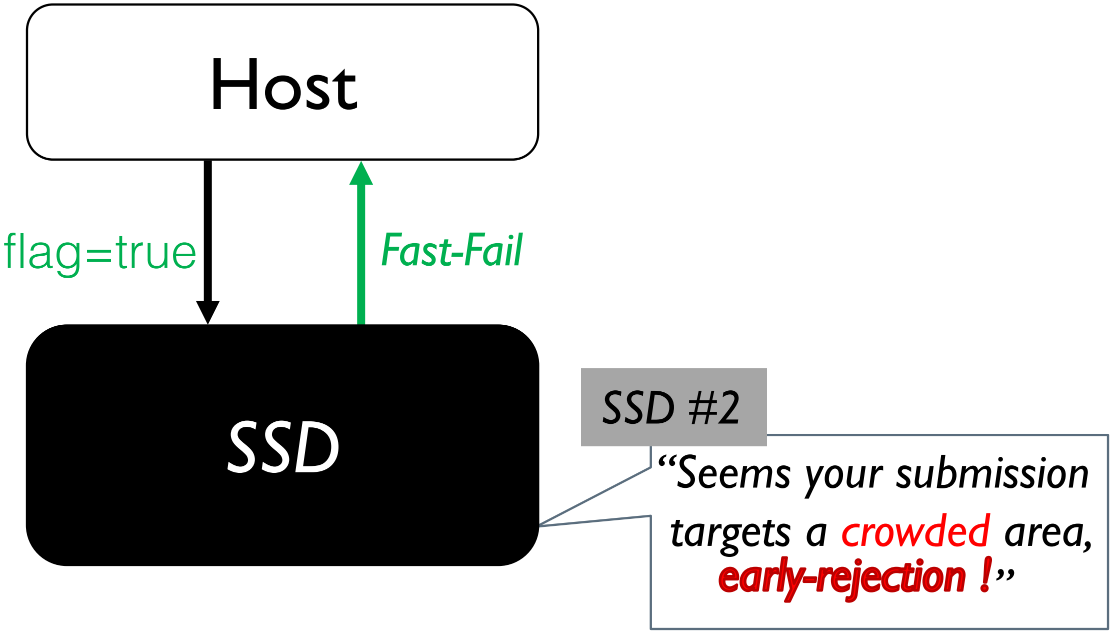
IOD_1: Predictable Latency Flagged I/Os

"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC



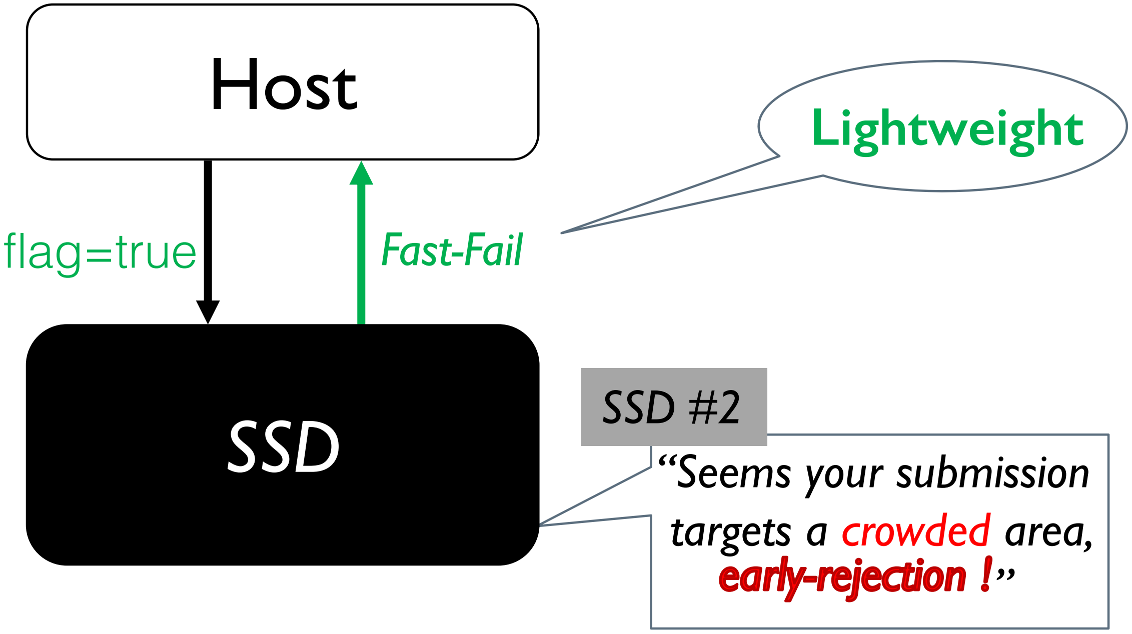
IOD$_1$: Predictable Latency Flagged I/Os

"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC



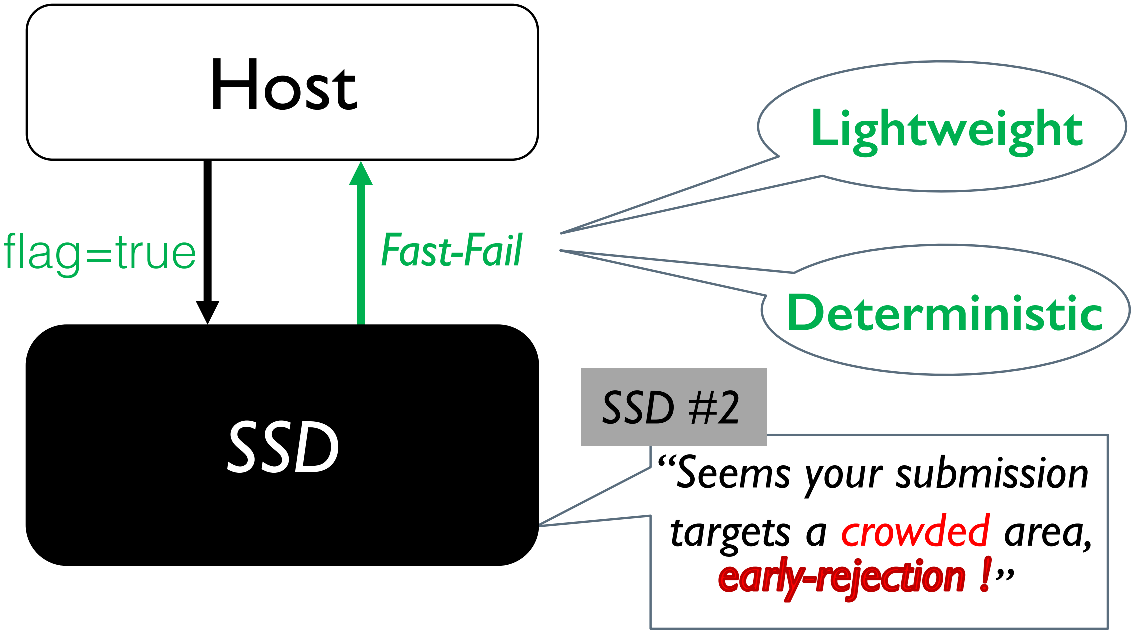
IOD$_1$: Predictable Latency Flagged I/Os

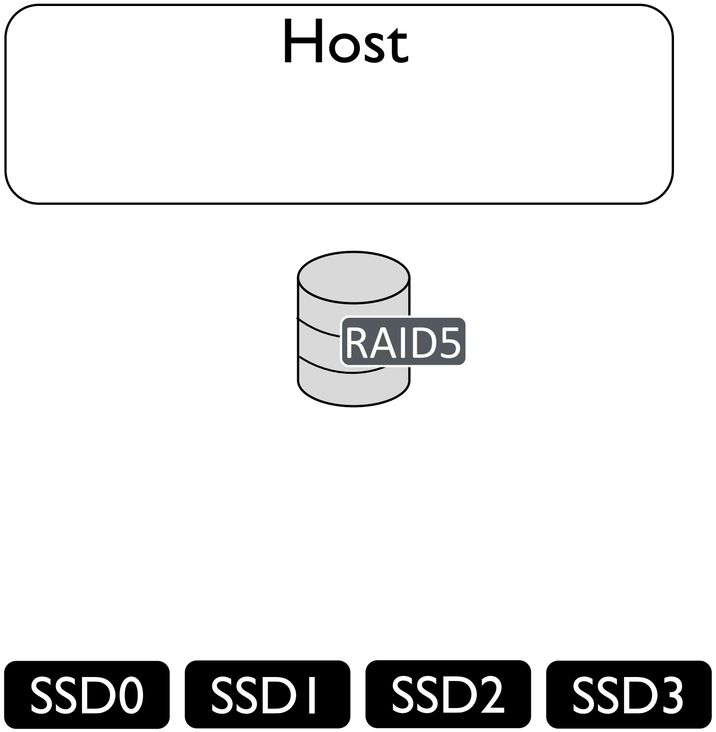
"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC

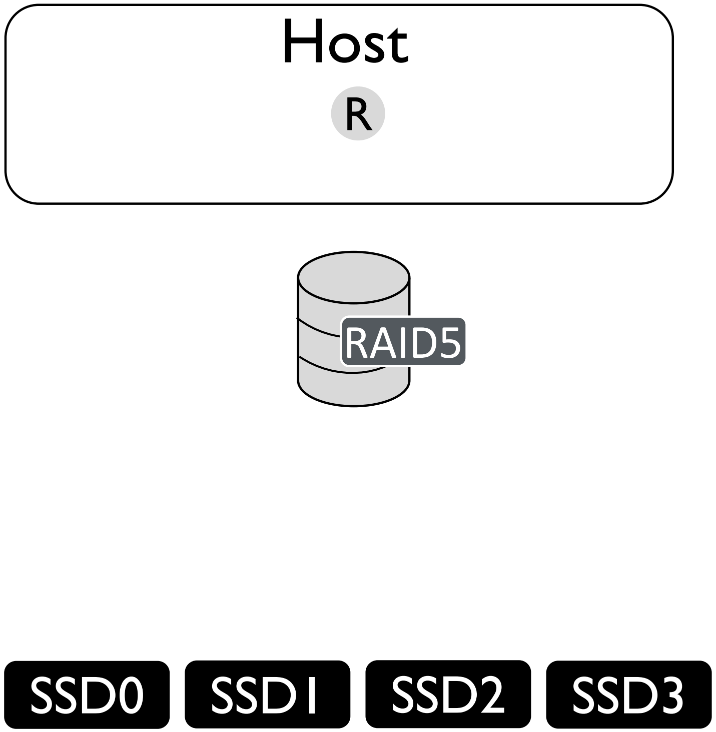


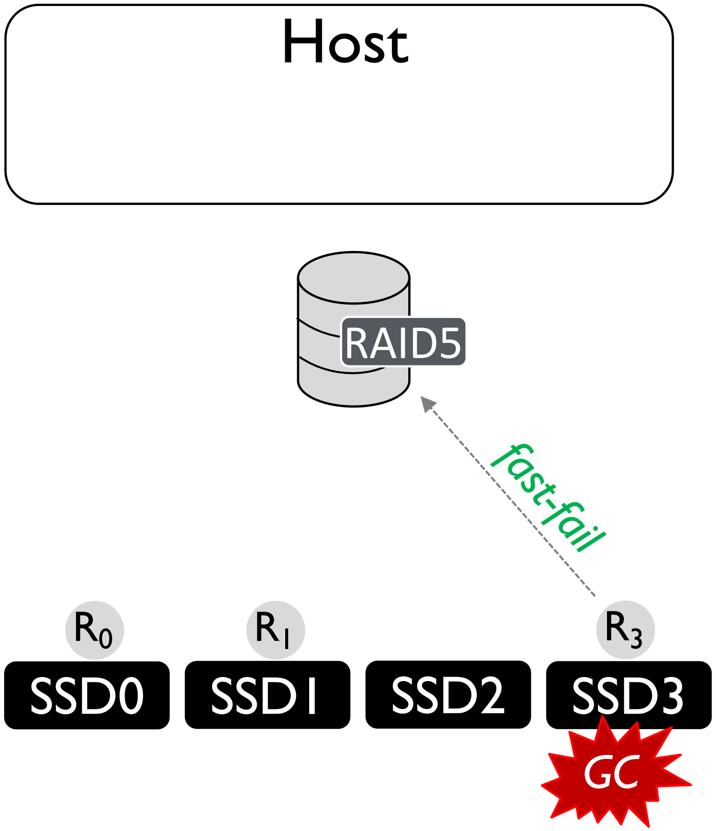
IOD_1: Predictable Latency Flagged I/Os

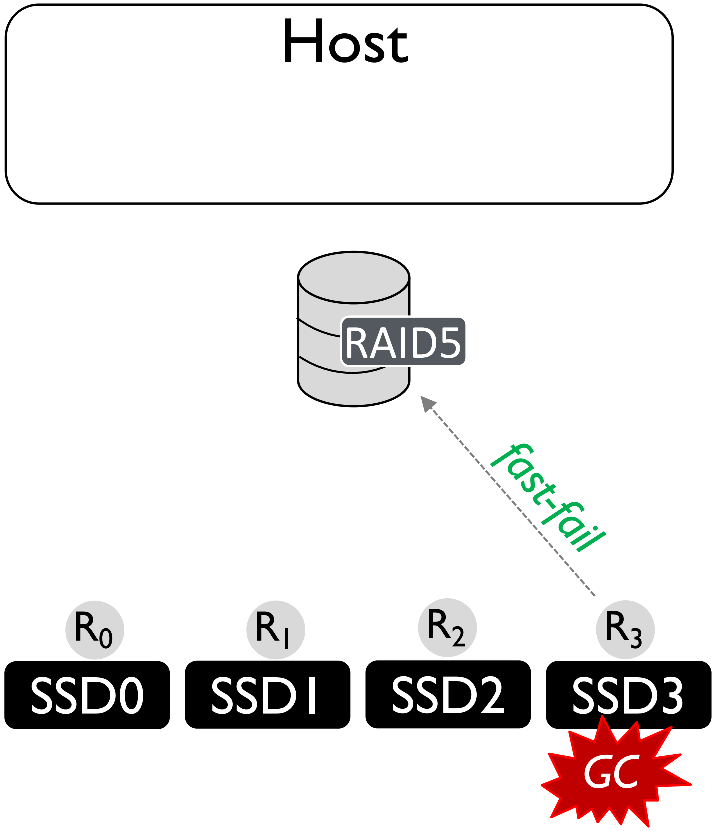
"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC

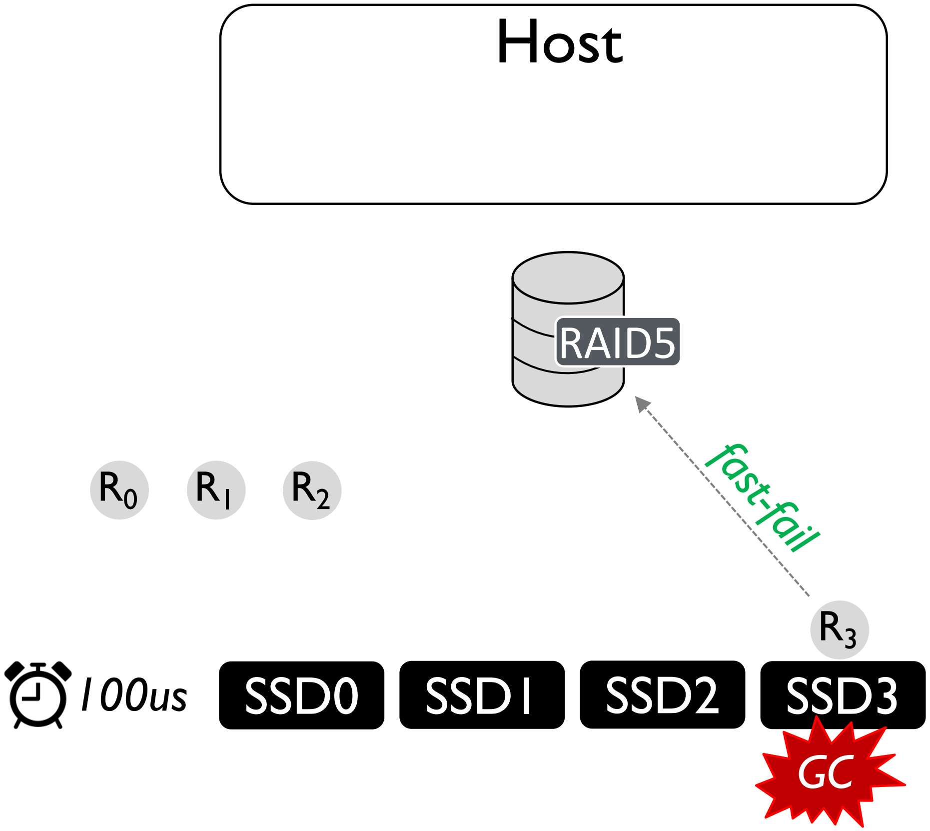


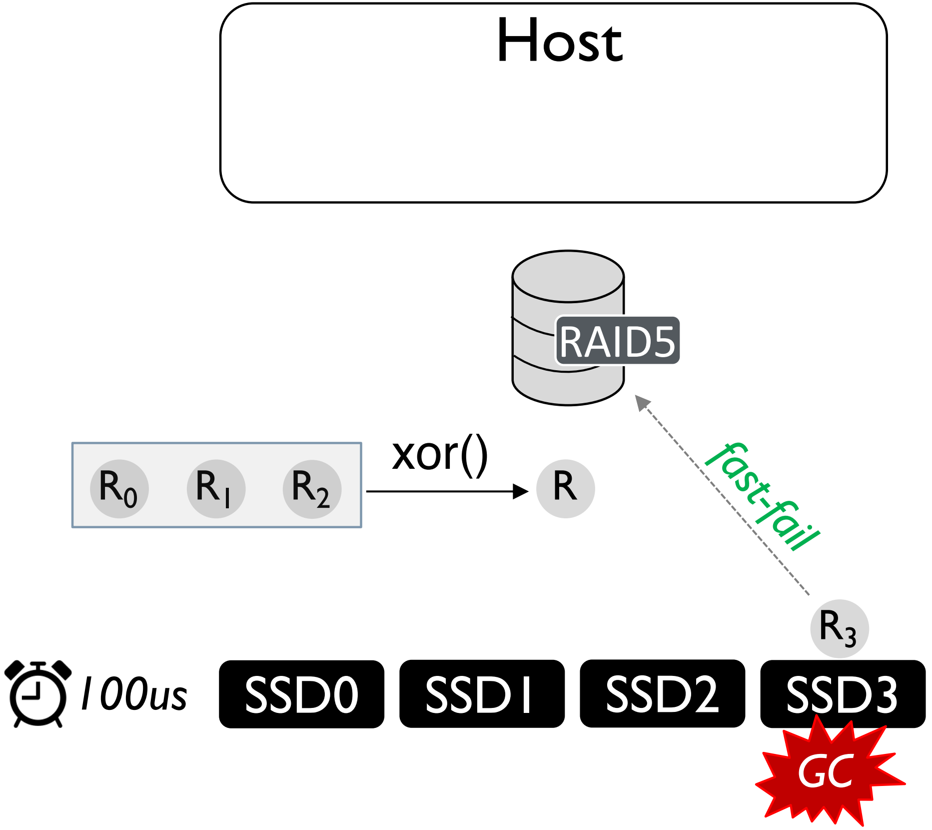

Host
RAID5
SSD0
SSD1
SSD2
SSD3

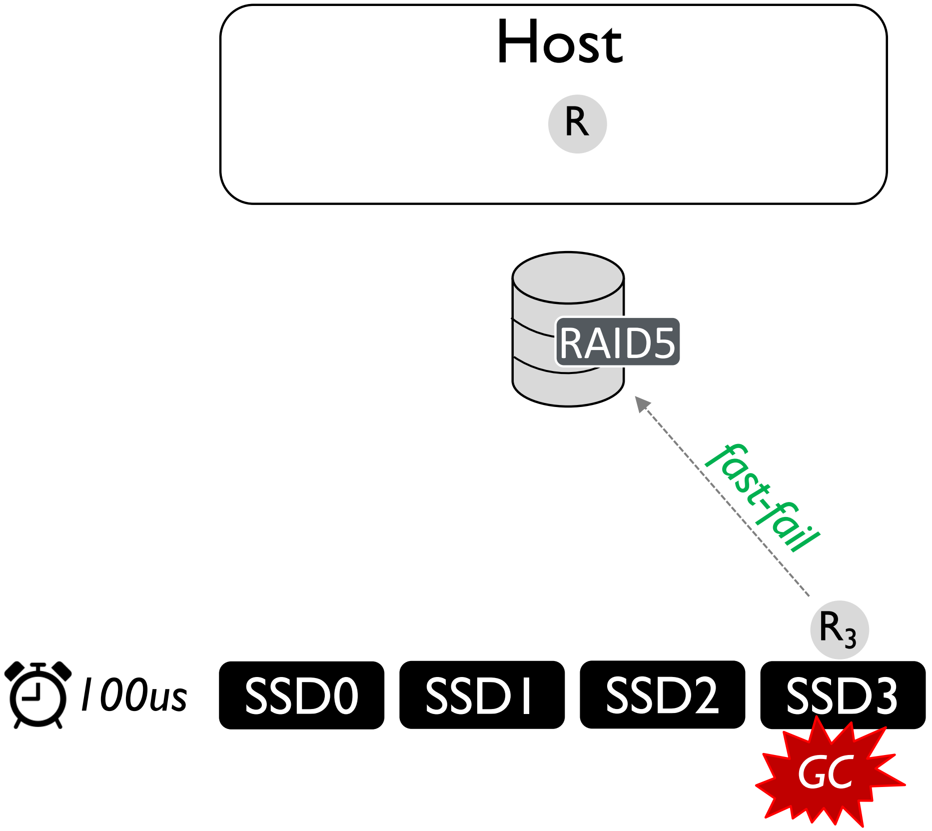

Host
R
RAID5
SSD0
SSD1
SSD2
SSD3

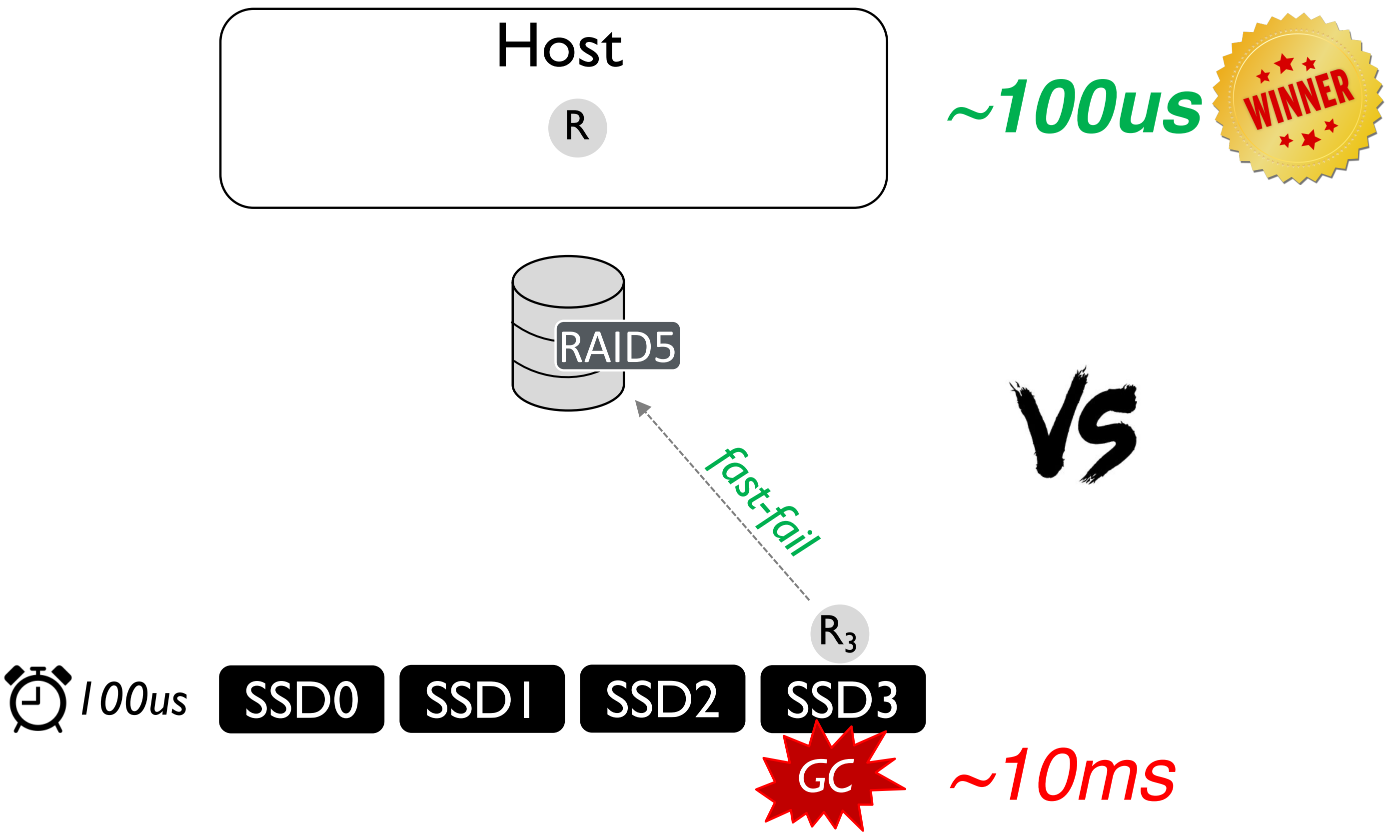

Host
RAID5
fast-fail
R_0
R_1
R_3
SSD0
SSD1
SSD2
SSD3
GC


Host
RAID5
fast-fail
R0
R1
R2
R3
SSD0
SSD1
SSD2
SSD3
GC

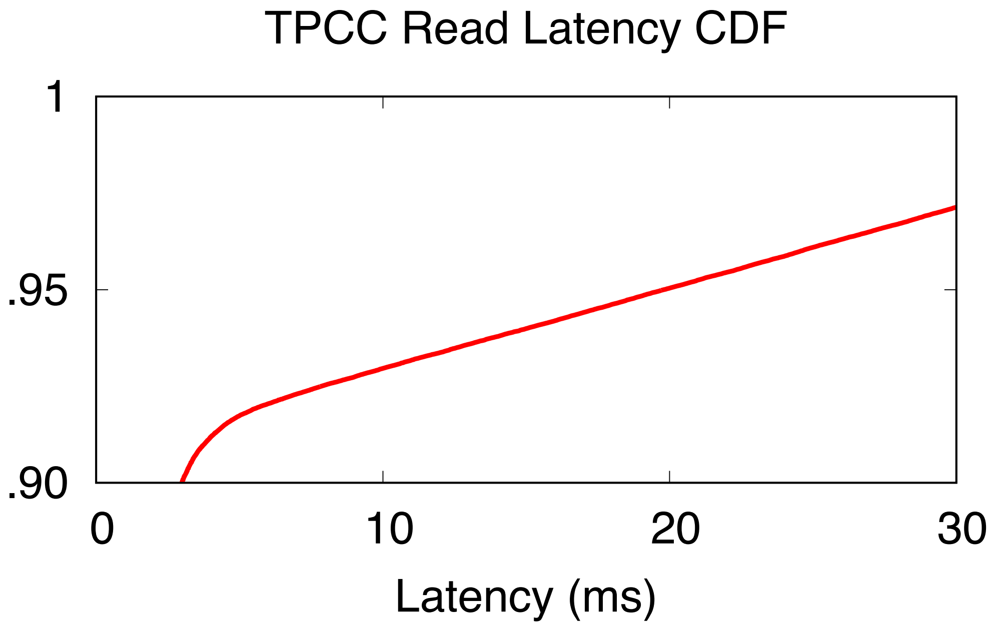

Host
RAID5
fast-fail
R0
R1
R2
R3
100us
SSD0
SSD1
SSD2
SSD3
GC


Host
RAID5
xor()
R0
R1
R2
R
fast-fail
R3
100us
SSD0
SSD1
SSD2
SSD3
GC

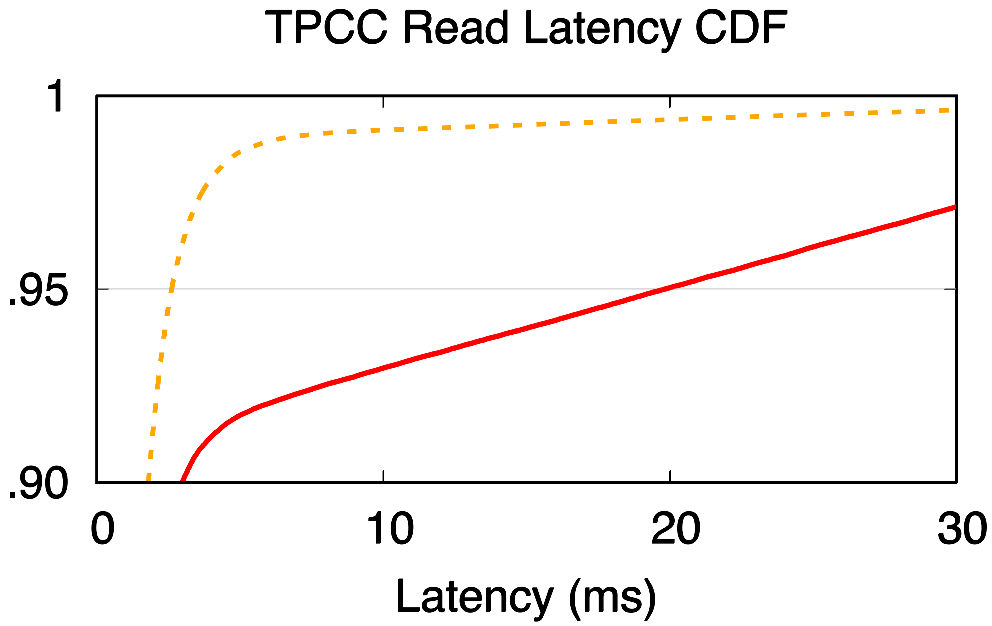

Host
R
RAID5
fast-fail
R3
100us
SSD0
SSD1
SSD2
SSD3
GC


Host
R
~100us
WINNER
RAID5
fast-fail
VS
R3
100us
SSD0
SSD1
SSD2
SSD3
GC
~10ms

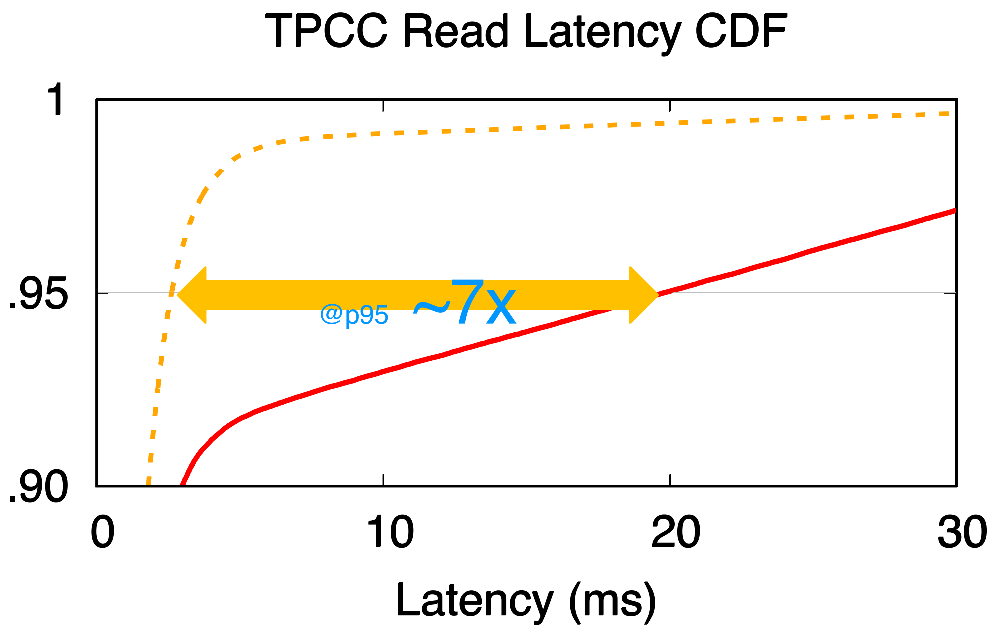
The Effectiveness of "*Fail-if-Slow*" Interface



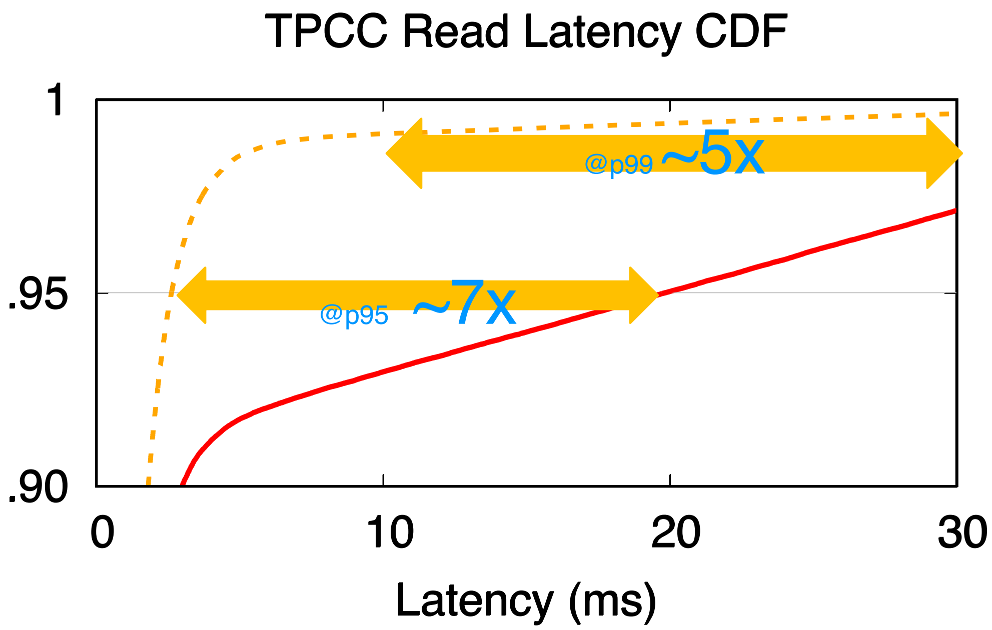
The Effectiveness of "*Fail-if-Slow*" Interface



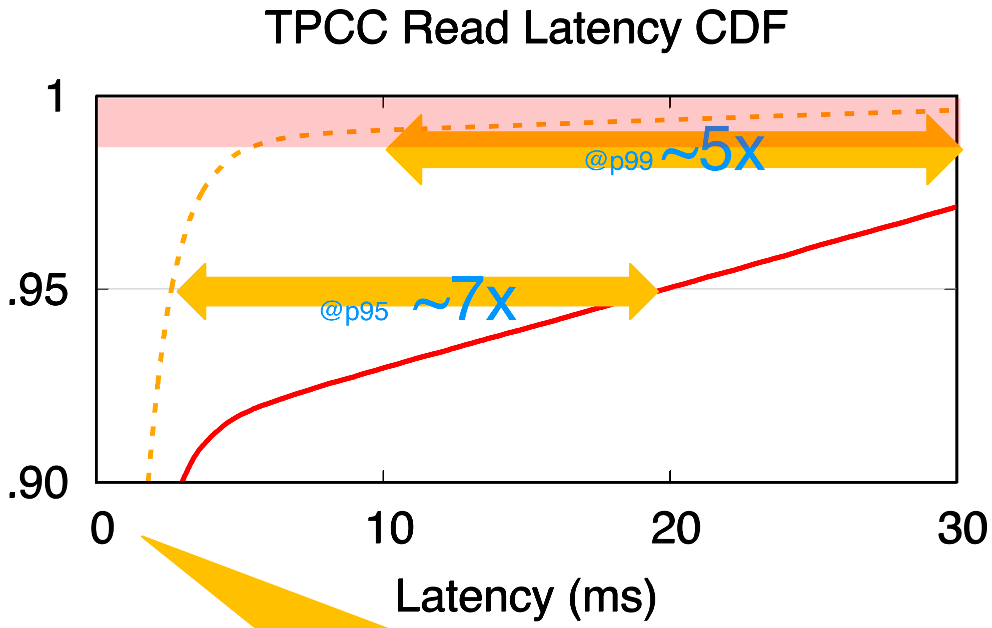
The Effectiveness of "*Fail-if-Slow*" Interface



The Effectiveness of "*Fail-if-Slow*" Interface

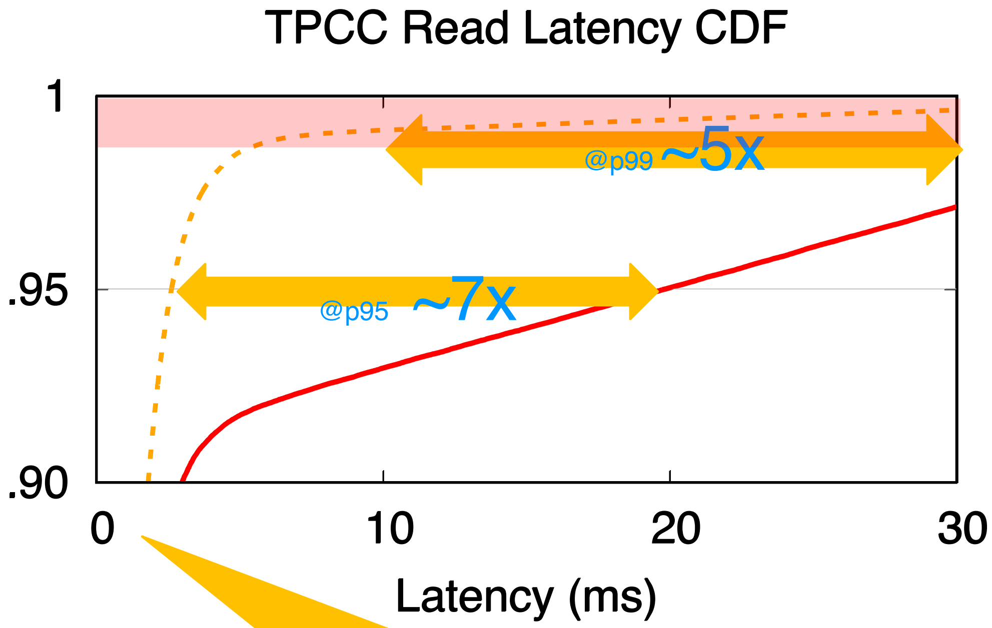


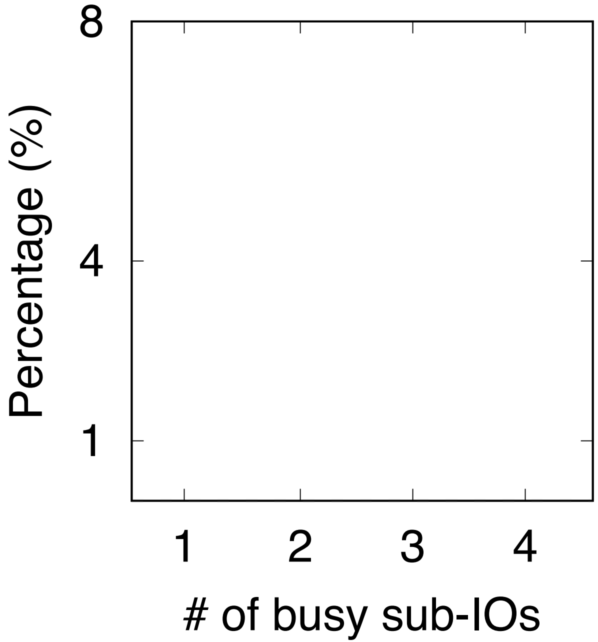
The Effectiveness of "*Fail-if-Slow*" Interface



Cut tails up to ~99th percentile

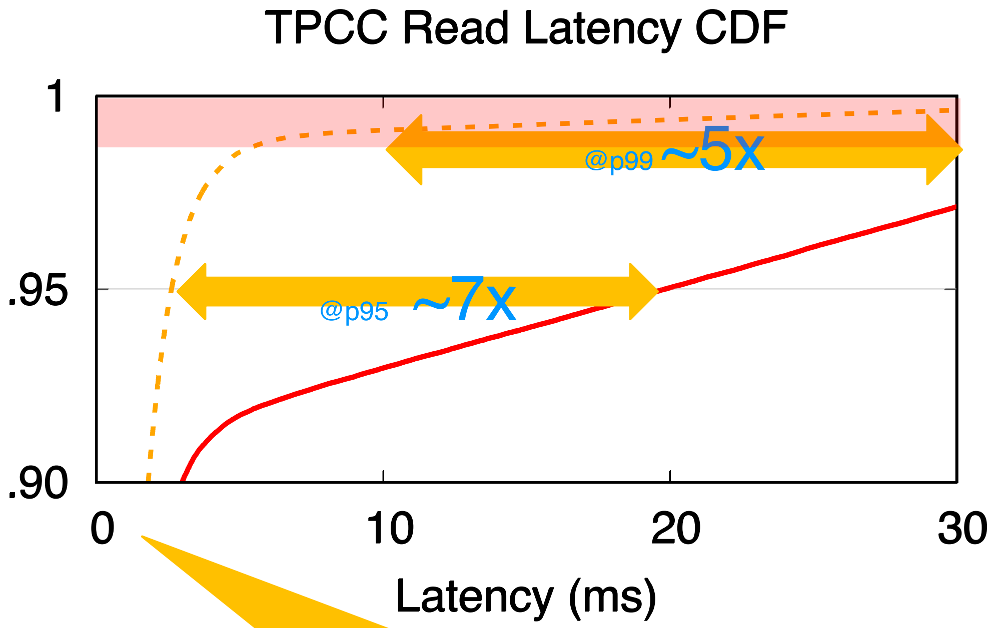
The Effectiveness of “*Fail-if-Slow*” Interface

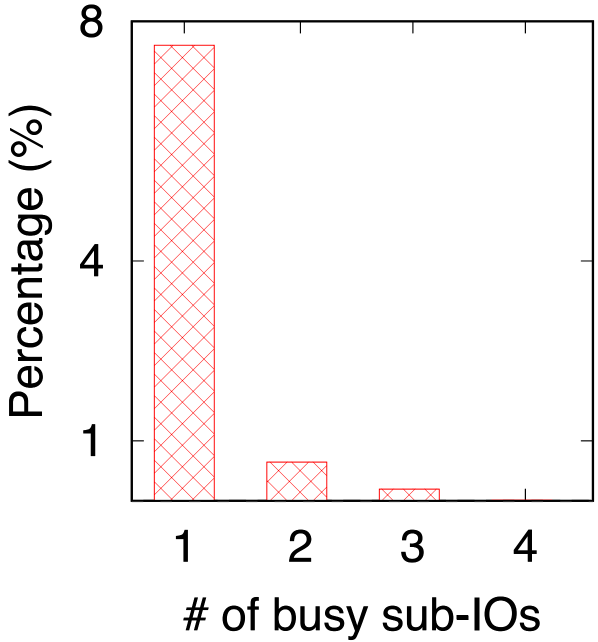




Cut tails up to ~99th percentile

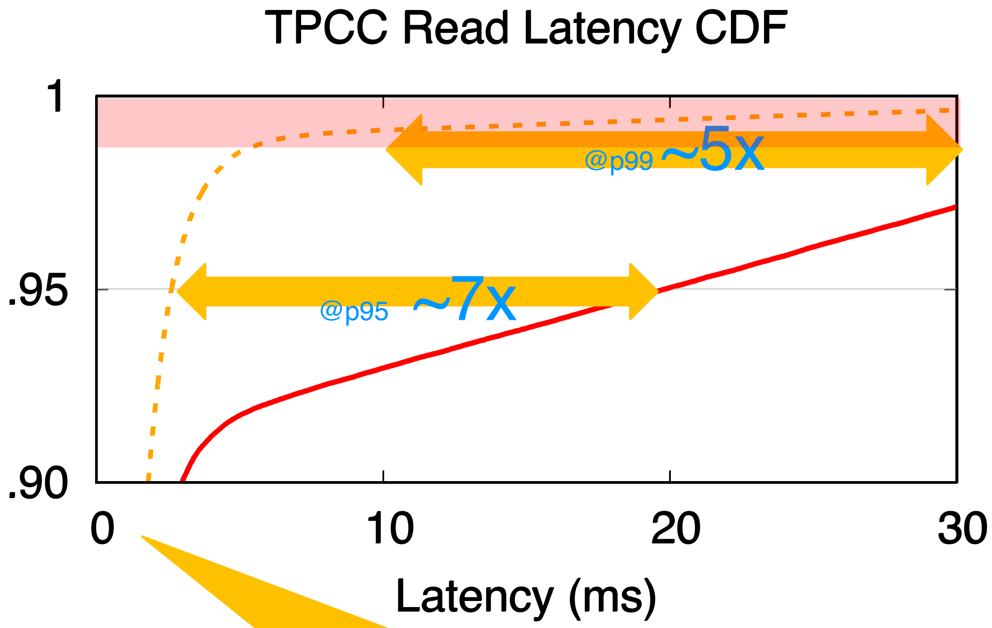
The Effectiveness of "*Fail-if-Slow*" Interface

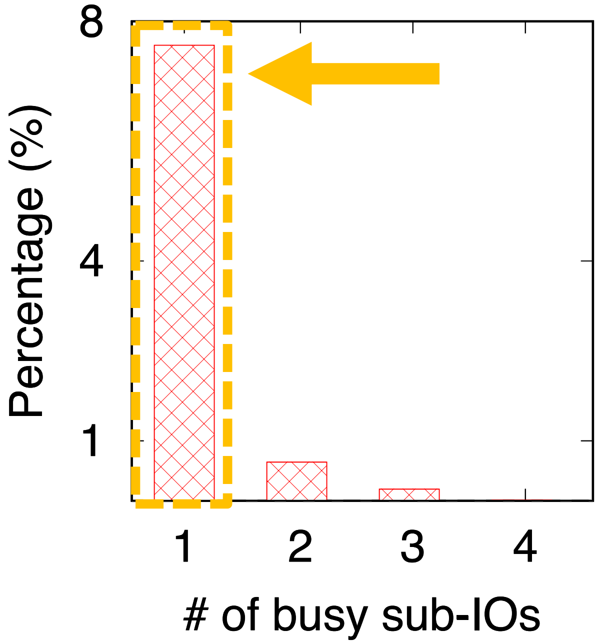




Cut tails up to ~99th percentile

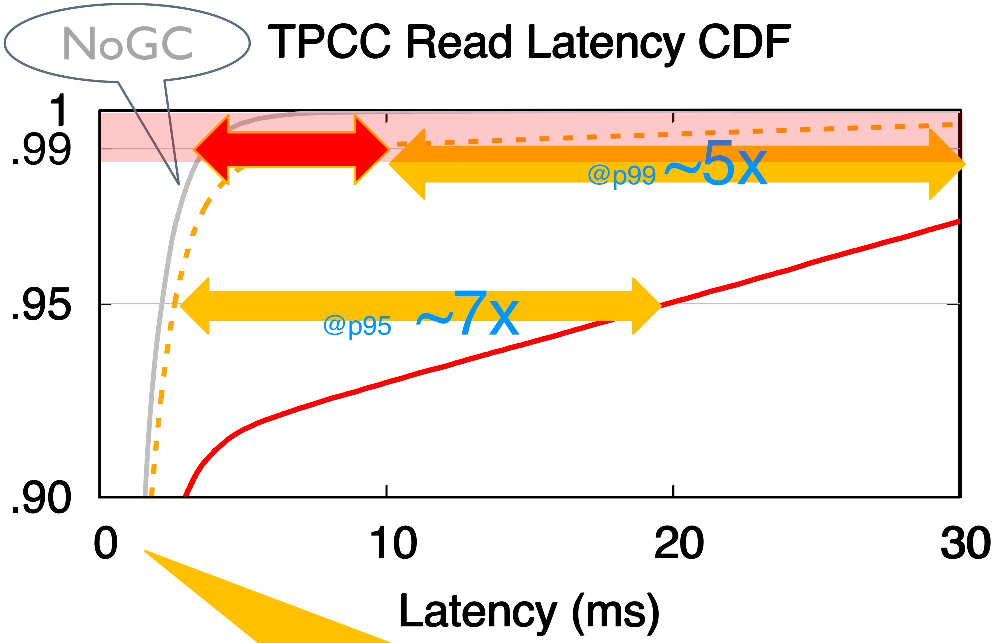
The Effectiveness of "*Fail-if-Slow*" Interface

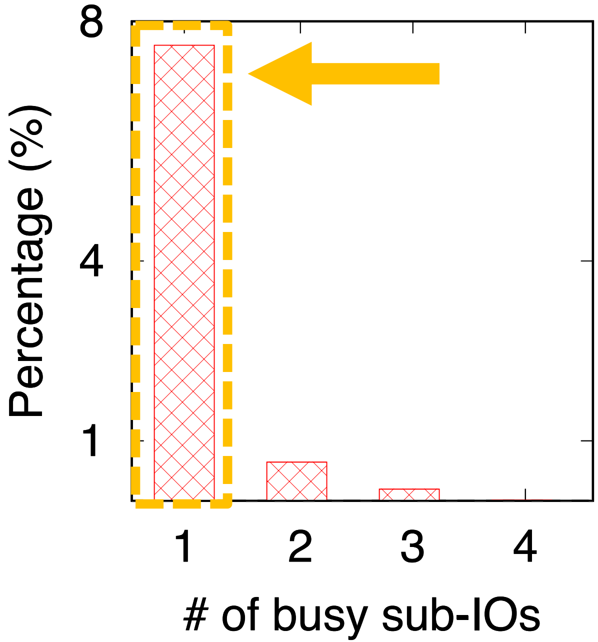




Cut tails up to ~99th percentile

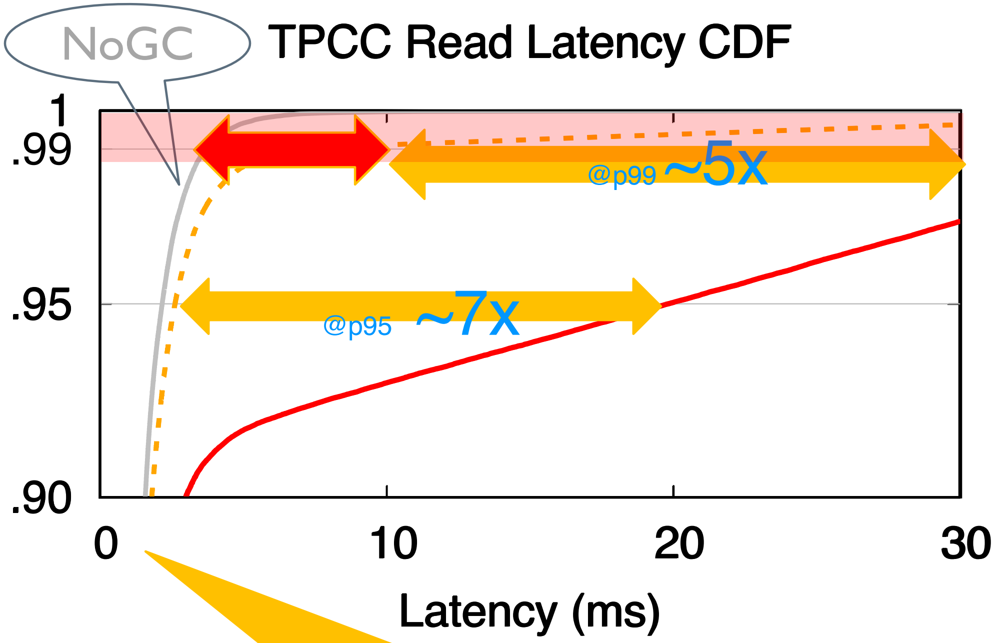
The Effectiveness of "*Fail-if-Slow*" Interface

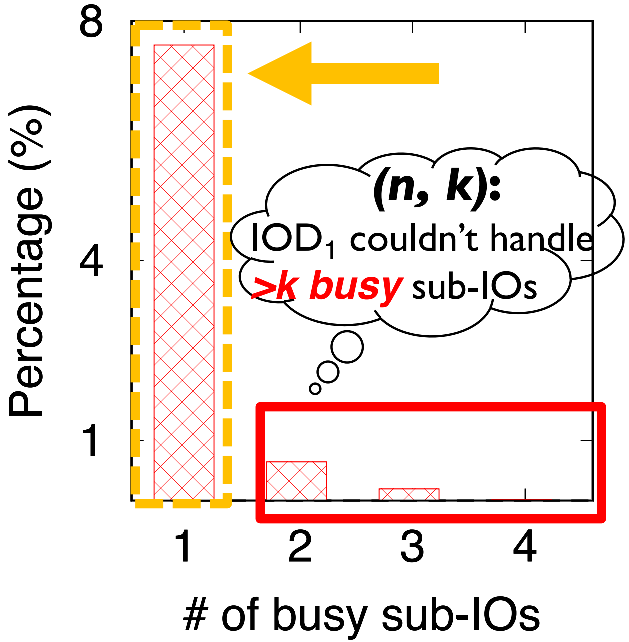




Cut tails up to ~99th percentile

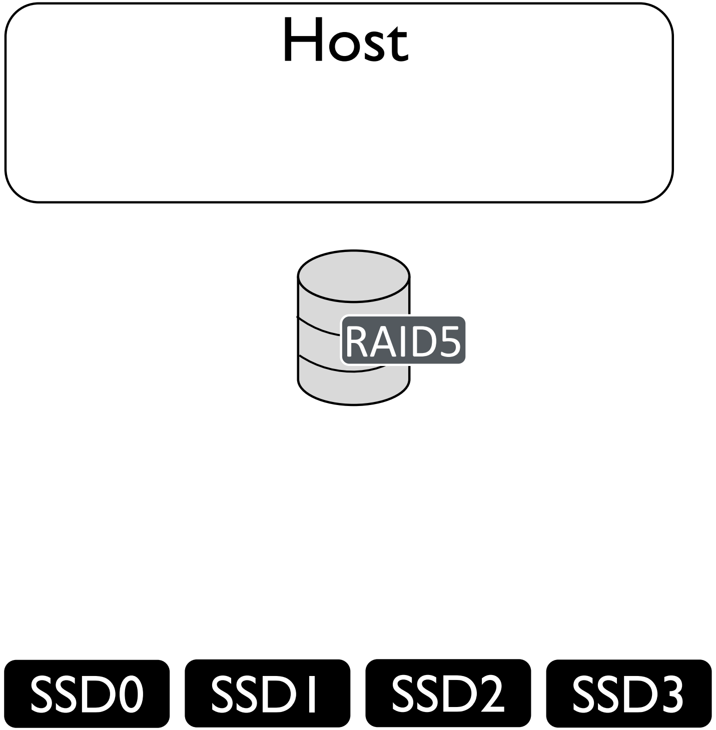
The Effectiveness of "*Fail-if-Slow*" Interface



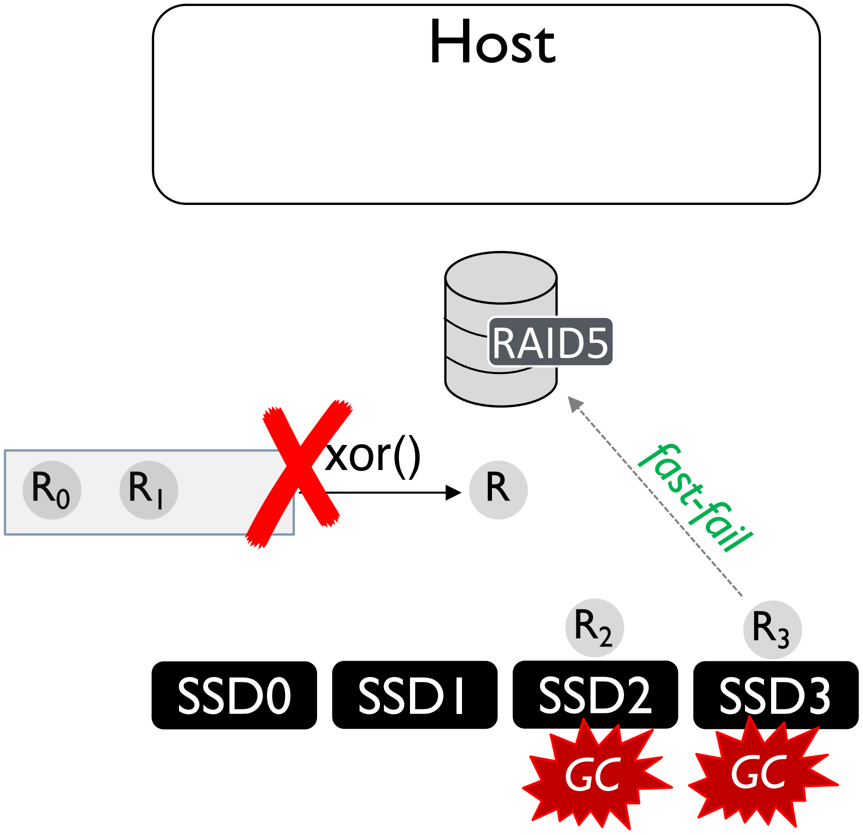


Cut tails up to ~99th percentile

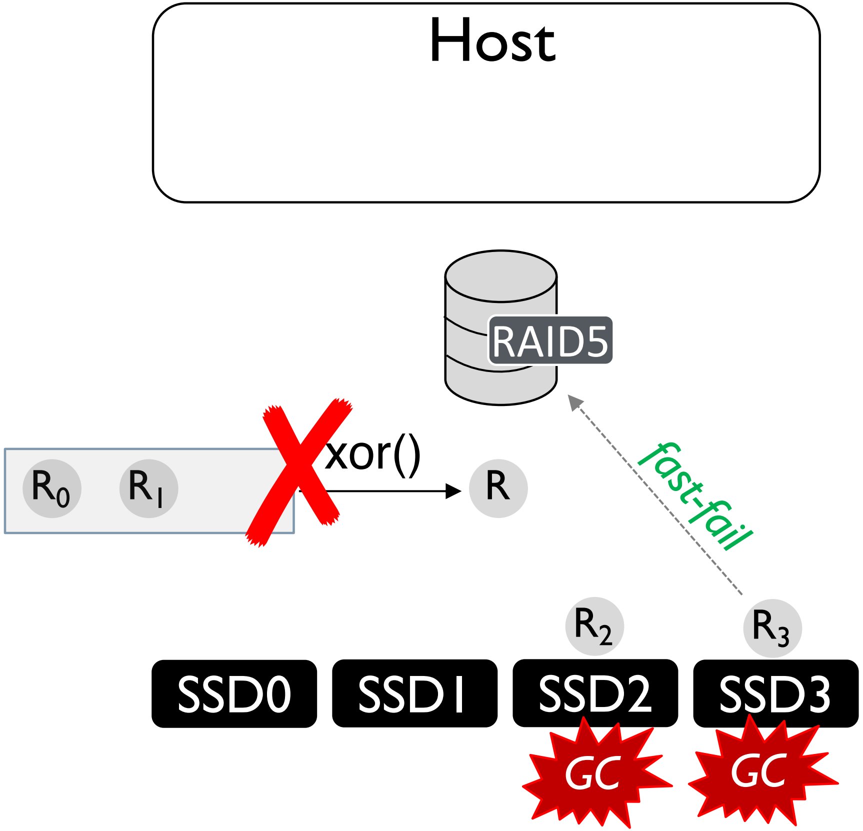
A Case Against Proactive Reconstruction



A Case Against Proactive Reconstruction

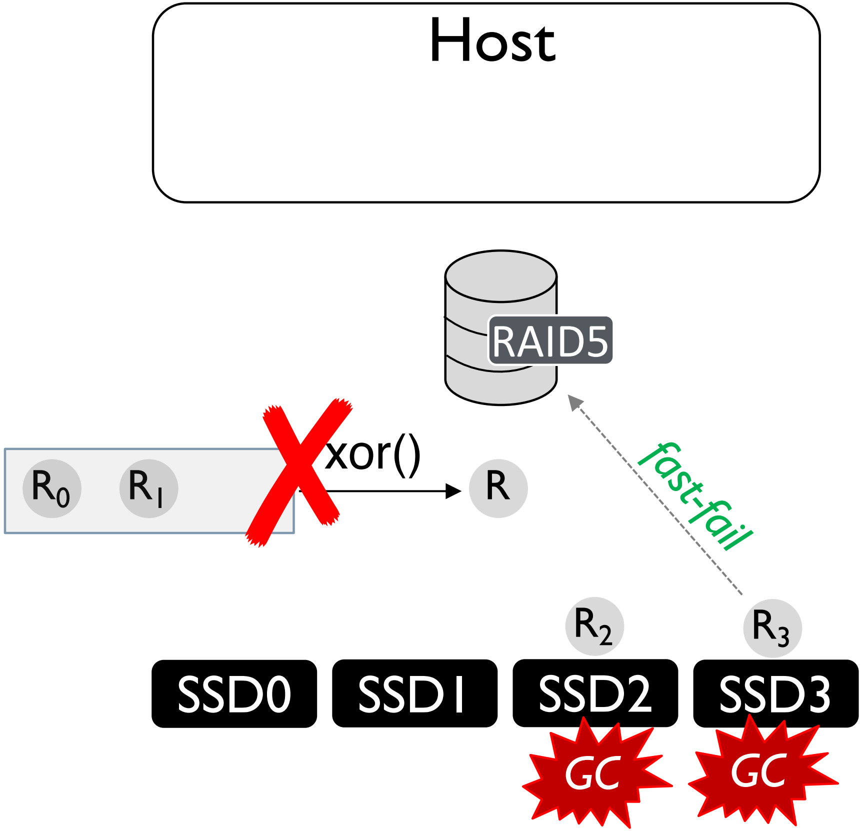


A Case Against Proactive Reconstruction



Semantic Gap: the host doesn't know how long SSD "busyness" will last

A Case Against Proactive Reconstruction



Semantic Gap: the host doesn't know how long SSD "busyness" will last

End up waiting for the busiest SSD

Busy Remaining Time (BRT) Exposure

"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC

+

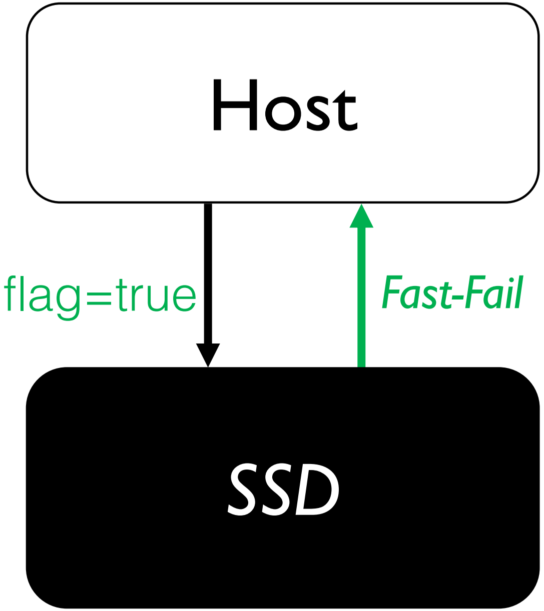
Piggybacking ***BRT*** to reconstruct data from less busy SSDs

Busy Remaining Time (BRT) Exposure

"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC

+

Piggybacking ***BRT*** to reconstruct data from less busy SSDs

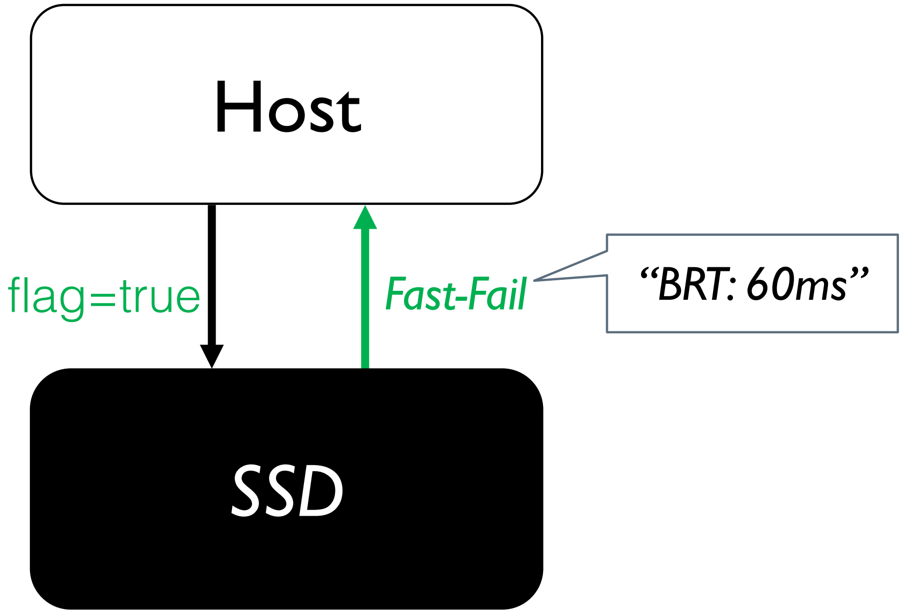


Busy Remaining Time (BRT) Exposure

“Fail-if-Slow”: the SSD should *fast-fail* an I/O if it contends with GC

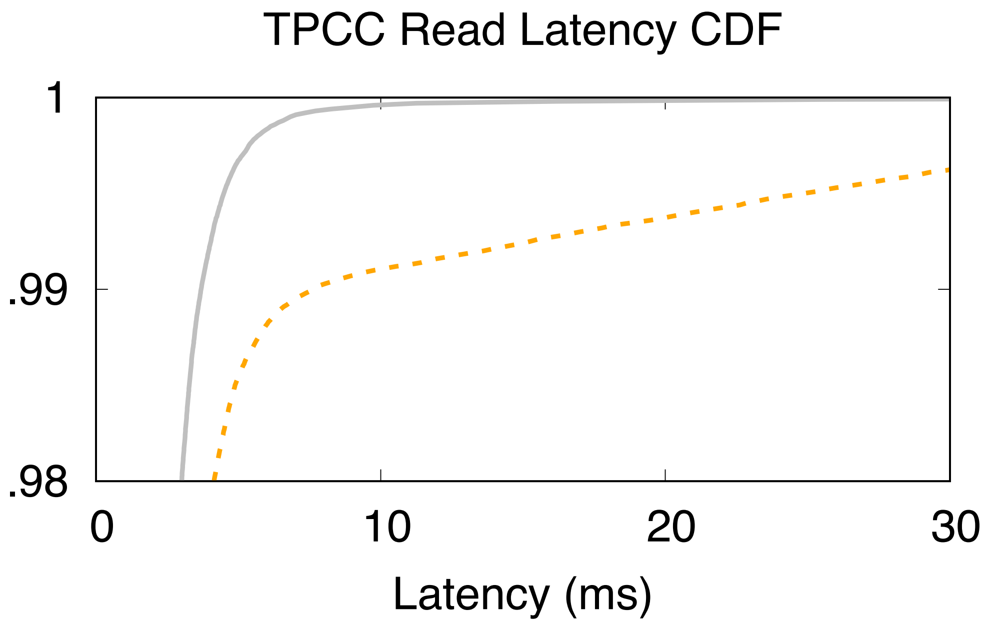
+

Piggybacking ***BRT*** to reconstruct data from less busy SSDs

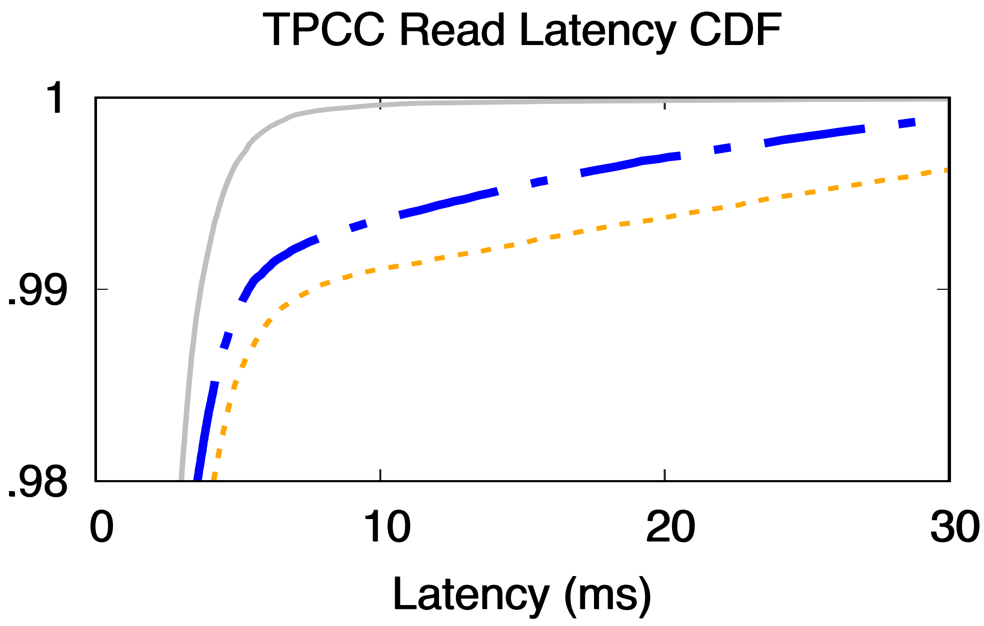


The Effectiveness of "*BRT*" Interface

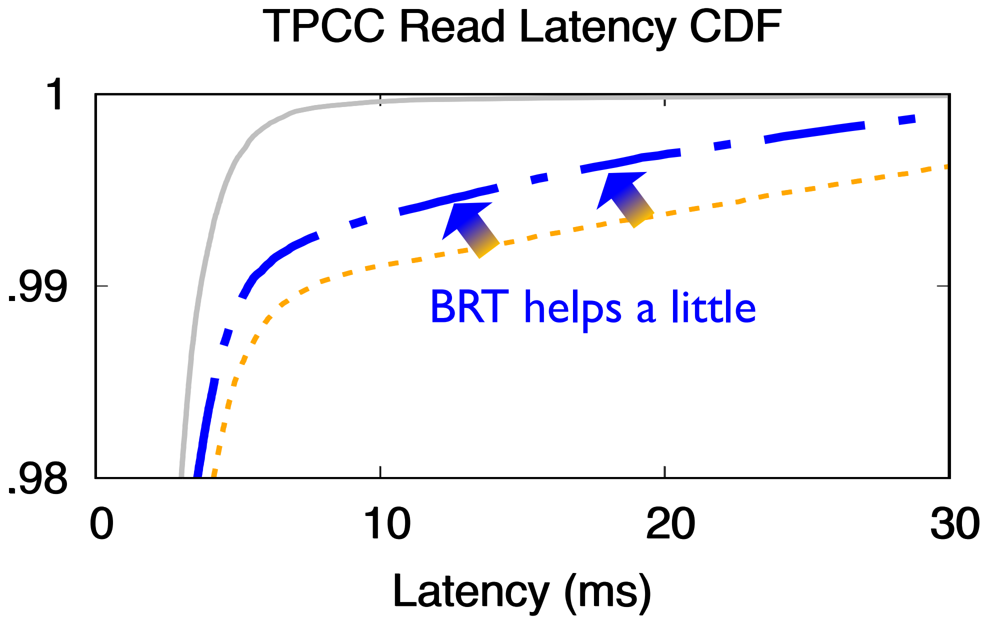
The Effectiveness of "*BRT*" Interface



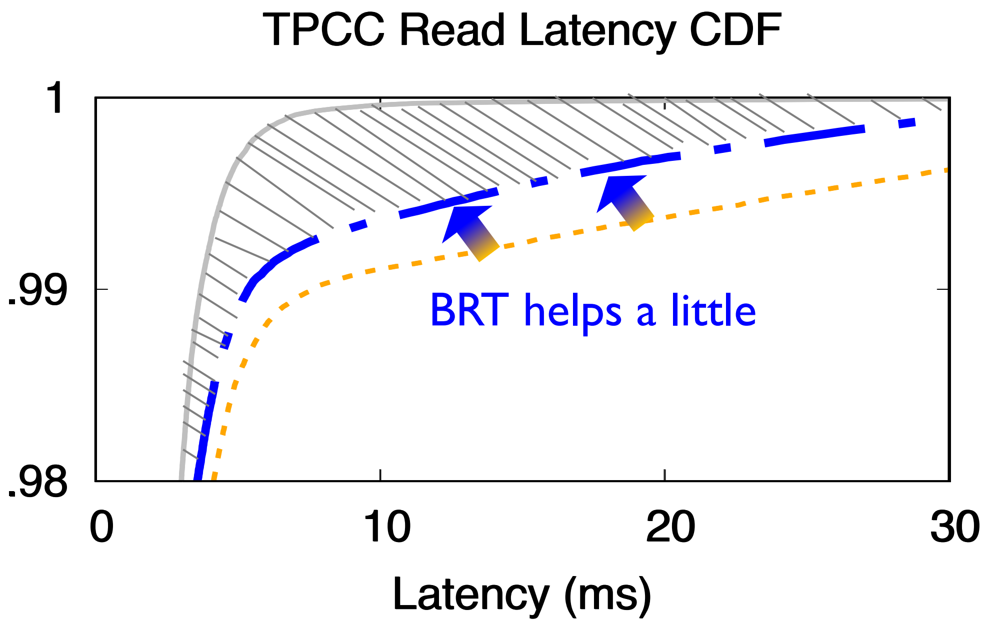
The Effectiveness of "*BRT*" Interface



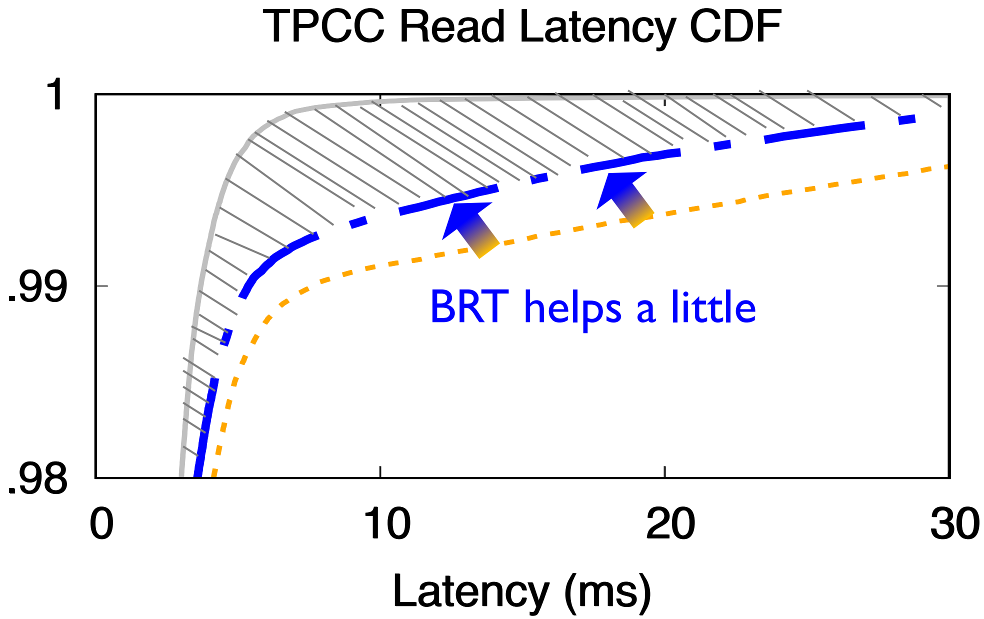
The Effectiveness of "*BRT*" Interface



The Effectiveness of "*BRT*" Interface



The Effectiveness of "*BRT*" Interface



Can we do better?

IODA Busy Latency Windows

"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC

\+

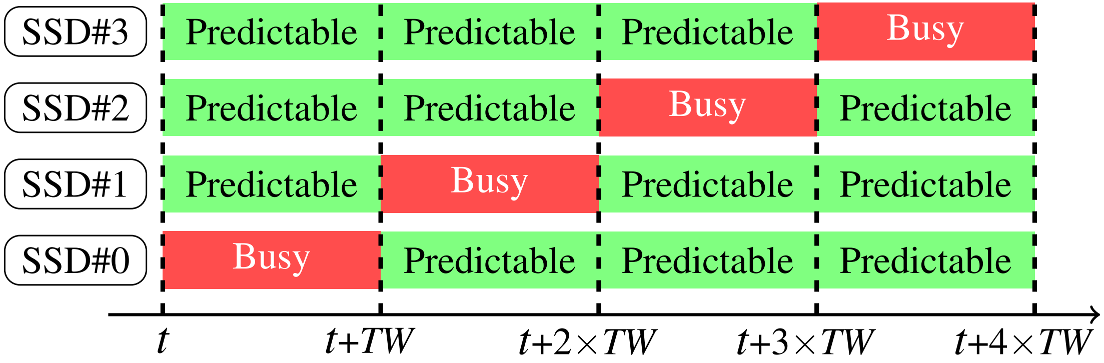
TW Coordination: SSDs take turns to perform GCs

IODA Busy Latency Windows

"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC

+

TW Coordination: SSDs take turns to perform GCs



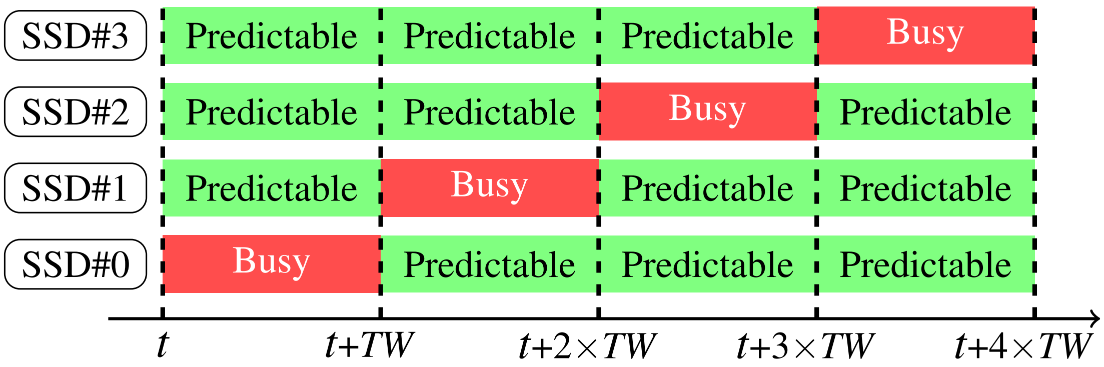
IODA Busy Latency Windows

"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC

+

TW Coordination: SSDs take turns to perform GCs

IODA: Always Predictable Latencies!



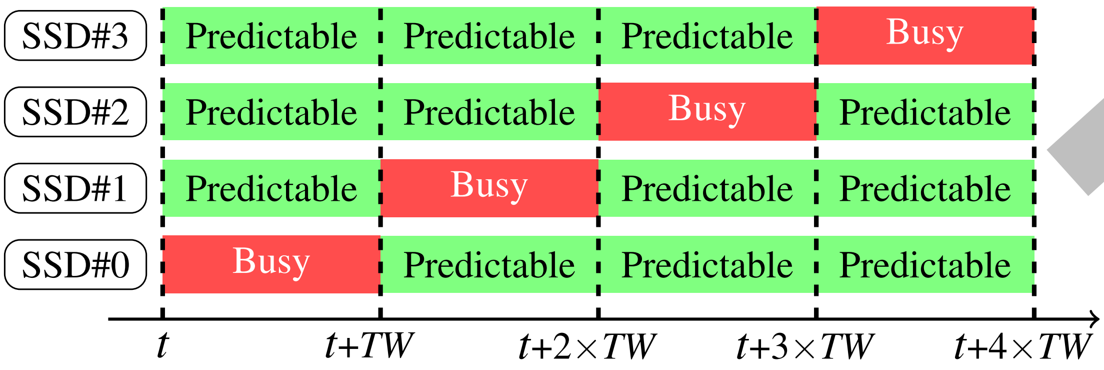
IODA Busy Latency Windows

"Fail-if-Slow": the SSD should *fast-fail* an I/O if it contends with GC

\+

TW Coordination: SSDs take turns to perform GCs

IODA: Always Predictable Latencies!



How long should *TW* be?

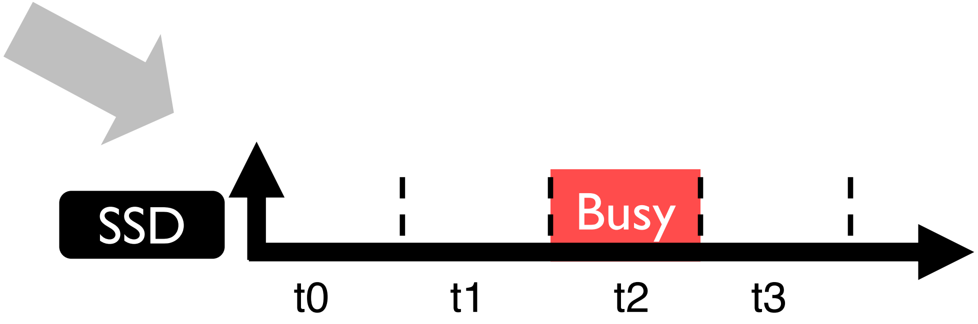
IODA Time Window (*TW*) Formulation

IODA Time Window (*TW*) Formulation

SSD free space >= *User load*

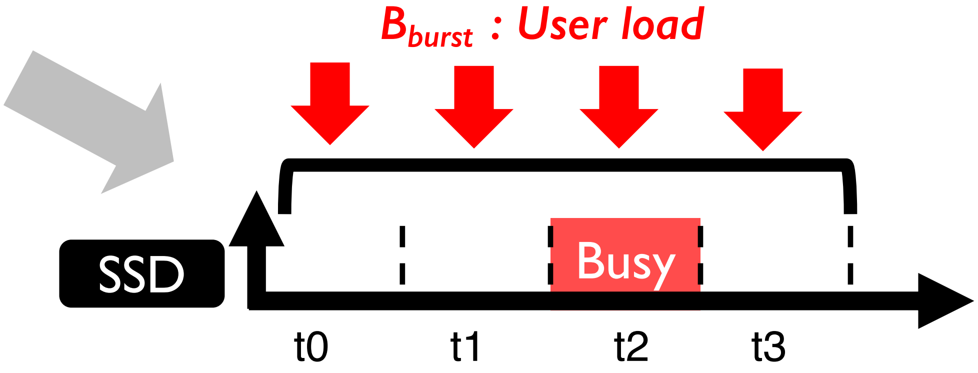
IODA Time Window (*TW*) Formulation

SSD free space >= *User load*



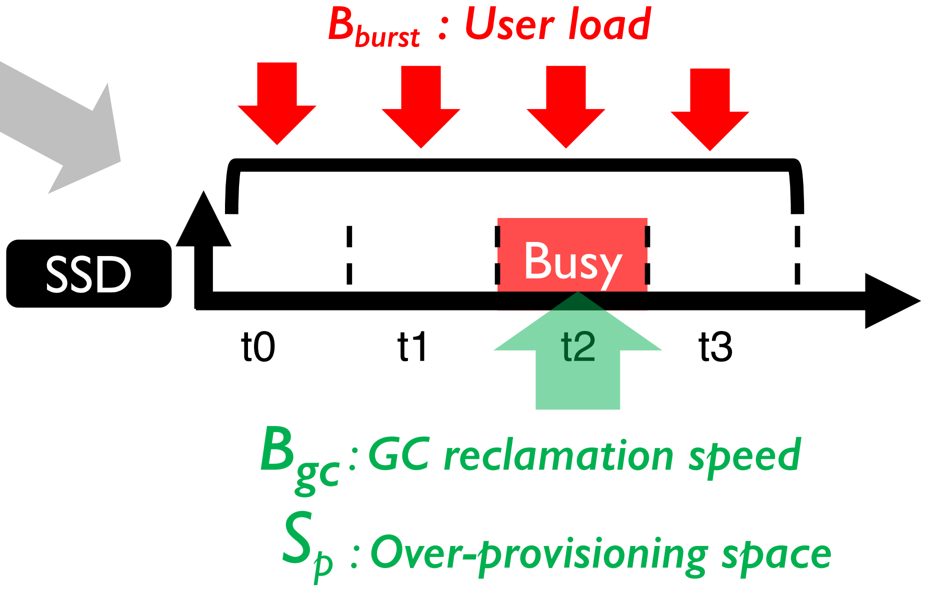
IODA Time Window (*TW*) Formulation

SSD free space >= *User load*



IODA Time Window (*TW*) Formulation

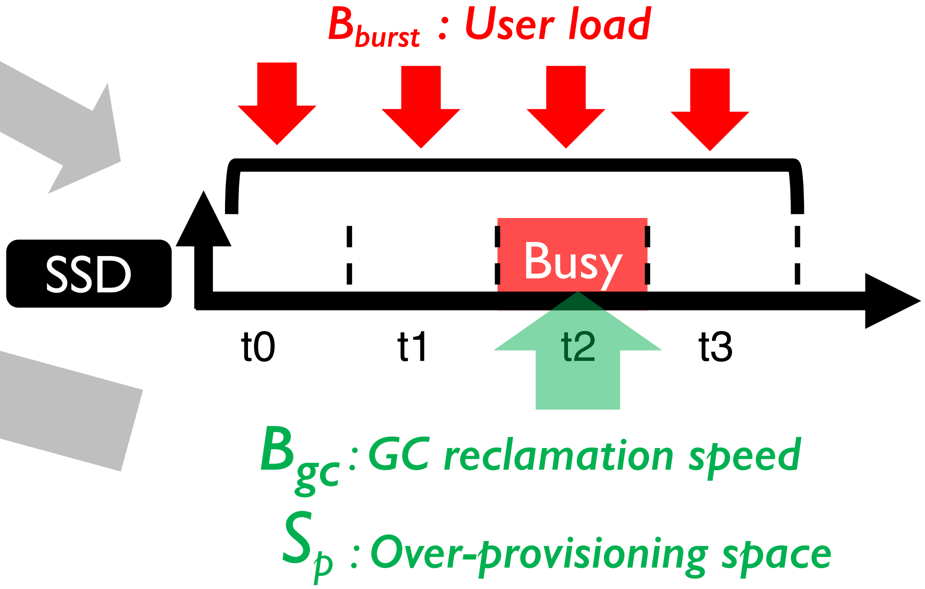
SSD free space >= *User load*



IODA Time Window (*TW*) Formulation

SSD free space >= *User load*

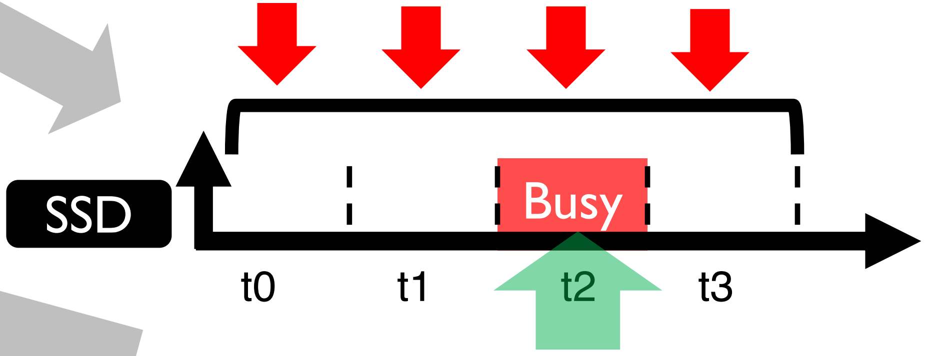
$$TW \leq S_p \,/\, ((N_{ssd} \times B_{burst}) - B_{gc})$$



IODA Time Window (*TW*) Formulation

SSD free space >= *User load*

B_{burst} : *User load*



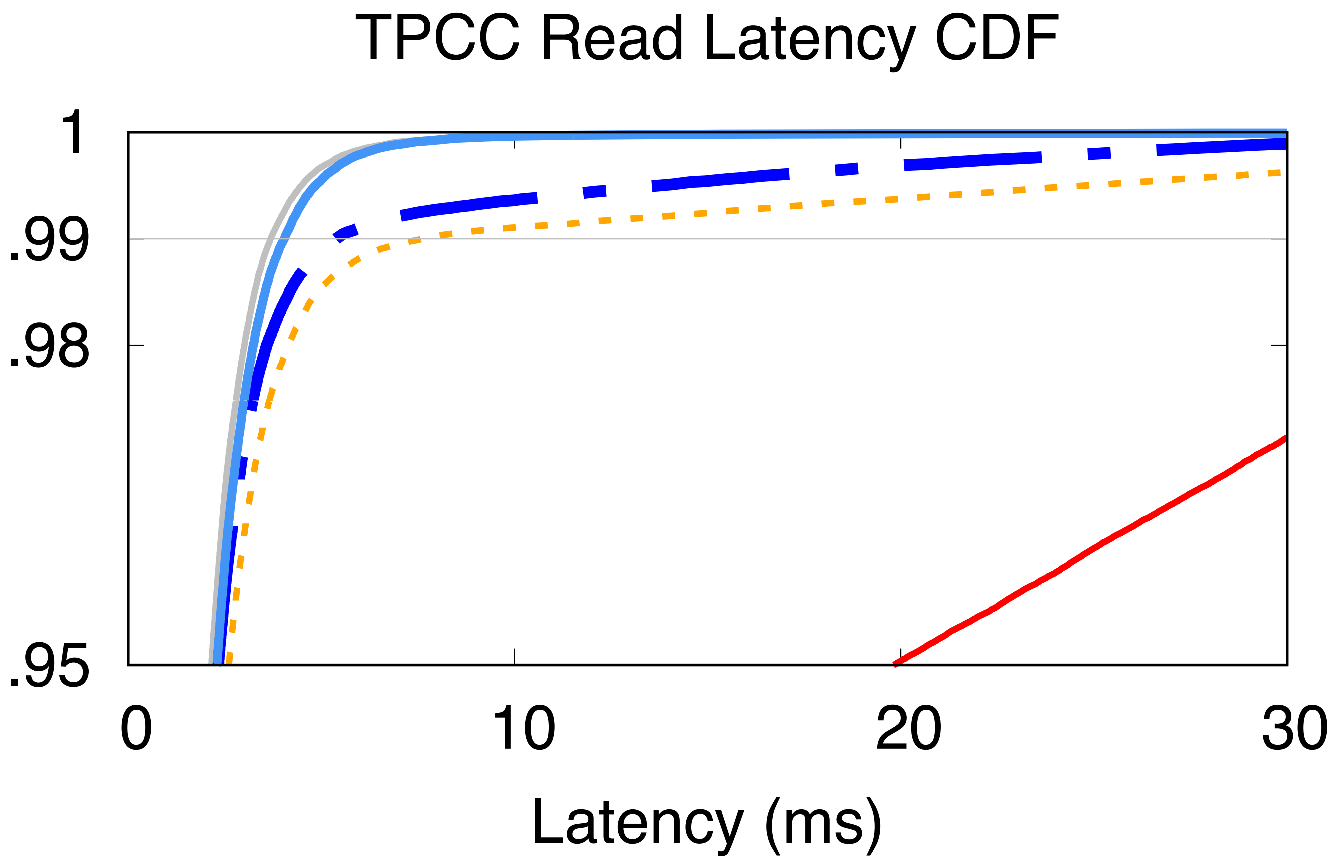
B_{gc} : *GC reclamation speed*

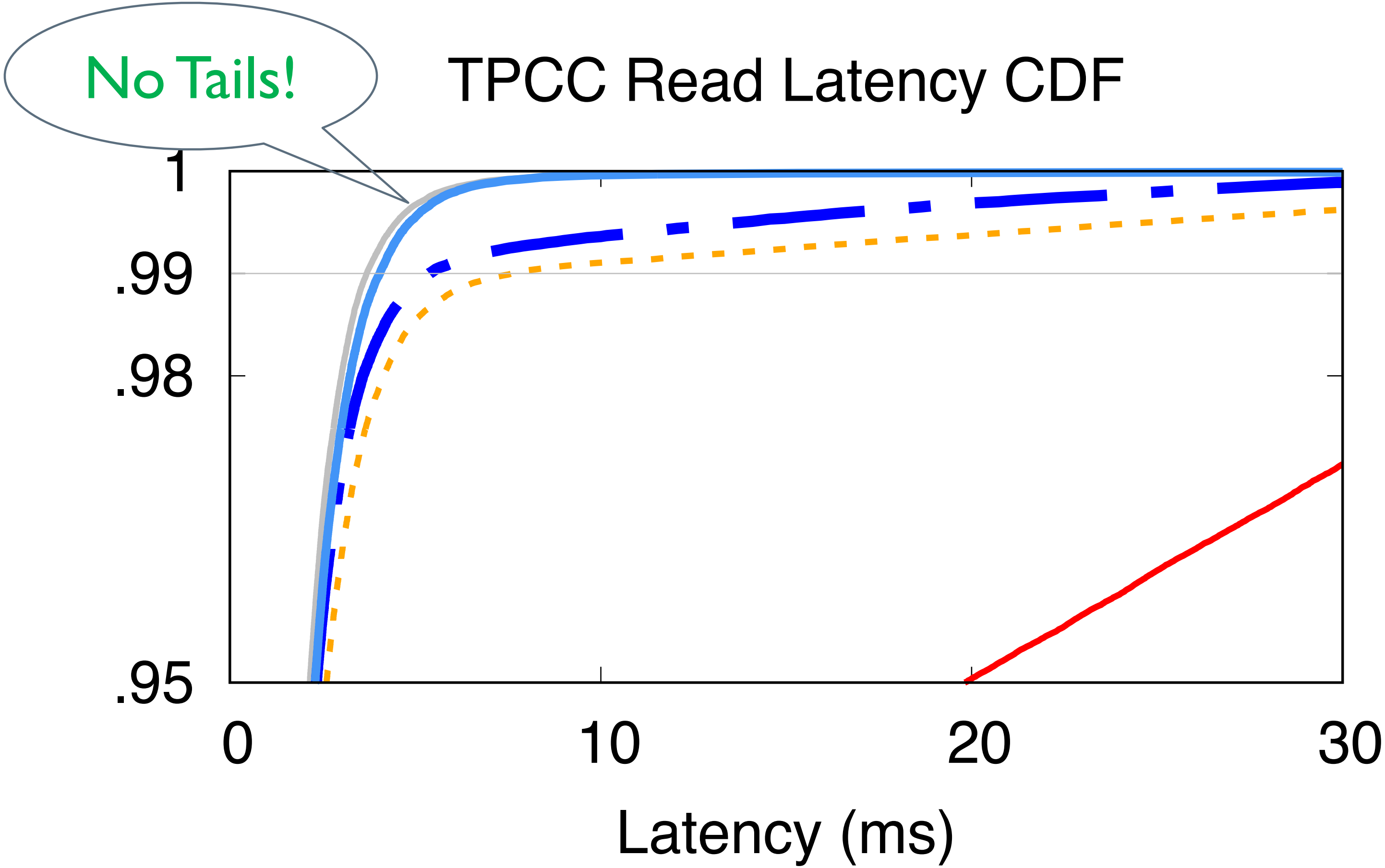
S_p : *Over-provisioning space*

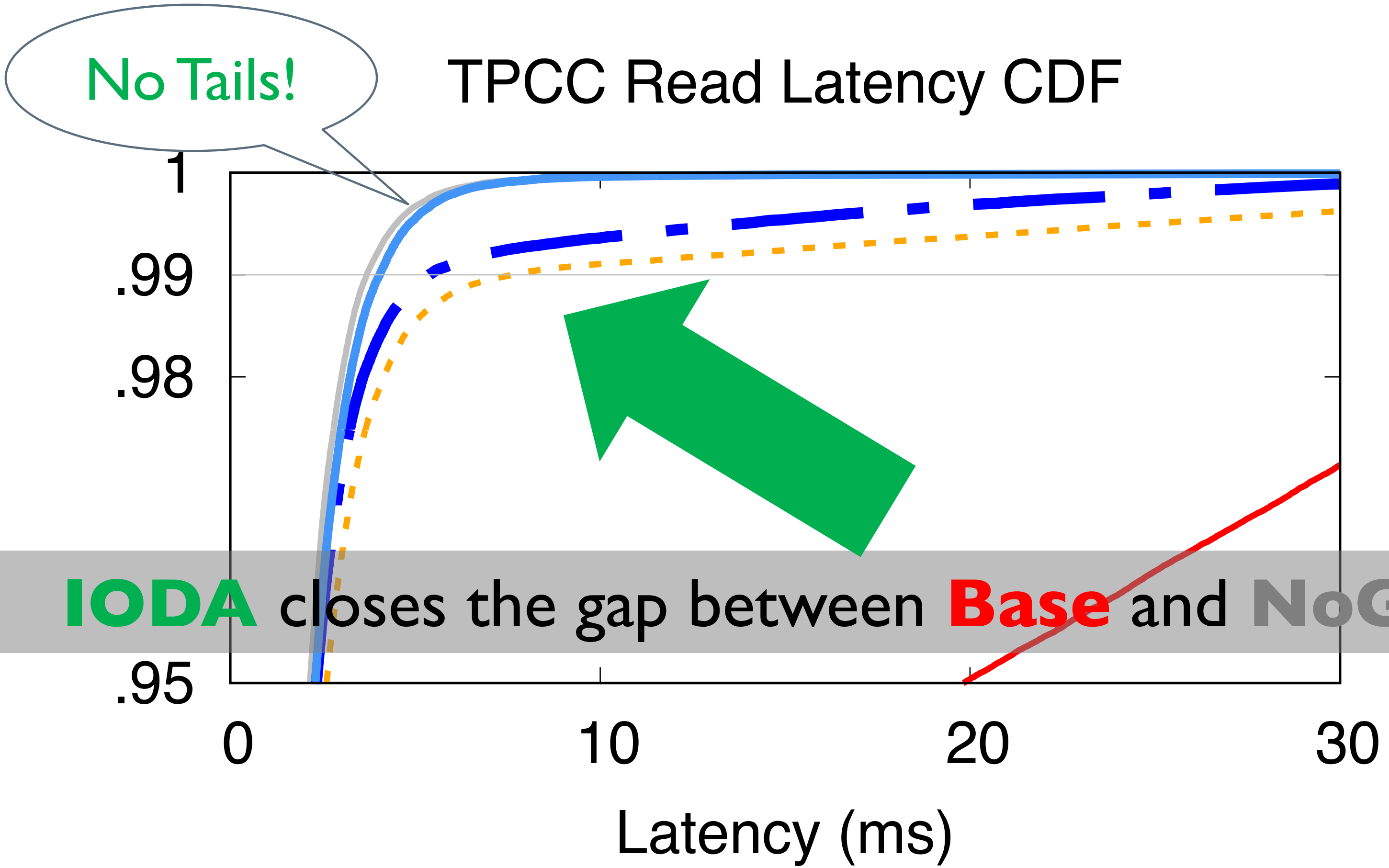
$$TW \leq S_p \;/\; ((N_{ssd} \times B_{burst}) - B_{gc})$$

$$TW \leq \frac{R_p \times S_t}{(N_{ssd} \times Min(B_{pcie}, Max(\frac{N_{dwpd} \times (1 - R_p) \times S_t}{8hours/day}))) - (\frac{(1 - R_v) \times N_{ch} \times S_{pg} \times N_{pg}}{(t_r + t_w + 2 \times t_{cpt}) \times R_v \times N_{pg} + t_e})}$$

TW Upper Bound


TPCC Read Latency CDF
1
.99
.98
.95
0
10
20
30
Latency (ms)

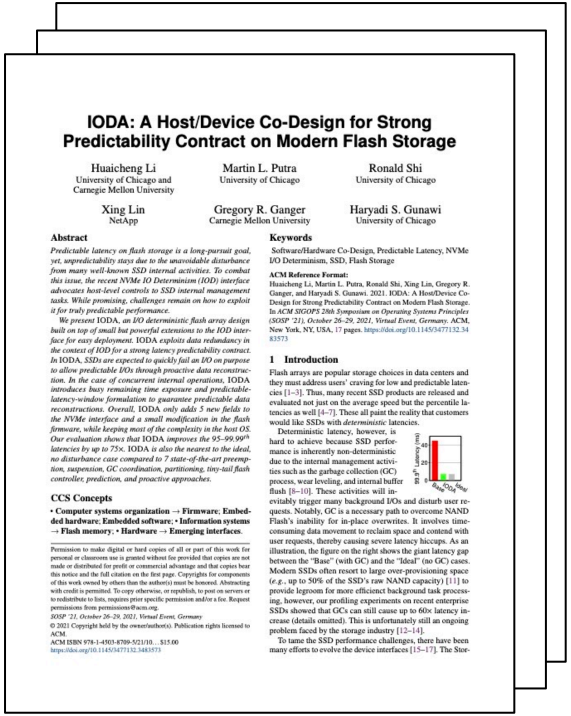

No Tails!
TPCC Read Latency CDF
1
.99
.98
.95
0
10
20
30
Latency (ms)



IODA closes the gap between **Base** and **NoGC**

More in the paper!

- IODA *TW* analysis
  - ***6*** SSD models
  - Relaxed *TW*
  - *TW vs. WAF* tradeoffs
- Implementation
  - Platforms: FEMU + OpenChannel-SSD
  - Kernel: Linux Software-RAID + NVMe
- More evaluation results
  - ***9*** datacenter block traces + ***21*** real applications
  - IODA vs. ***7*** State-of-the-art approaches
  - IODA on OpenChannel-SSD
  - IODA throughput and write latency
  - …



IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage

Huaicheng Li, University of Chicago and Carnegie Mellon University
Martin L. Putra, University of Chicago
Ronald Shi, University of Chicago
Xing Lin, NetApp
Gregory R. Ganger, Carnegie Mellon University
Haryadi S. Gunawi, University of Chicago

Abstract

Predictable latency on flash storage is a long-pursuit goal, yet, unpredictability stays due to the unavoidable disturbance from many well-known SSD internal activities. To combat this issue, the recent NVMe IO Determinism (IOD) interface advocates host-level controls to SSD internal management tasks. While promising, challenges remain on how to exploit it for truly predictable performance.

We present IODA, *an I/O deterministic flash array design built on top of small but powerful extensions to the IOD interface for easy deployment.* IODA *exploits data redundancy in the context of IOD for a strong latency predictability contract. In* IODA, *SSDs are expected to quickly fail an I/O on purpose to allow predictable I/Os through proactive data reconstruction. In the case of concurrent internal operations,* IODA *introduces busy remaining time exposure and predictable-latency-window formulation to guarantee predictable data reconstructions. Overall,* IODA *only adds 5 new fields to the NVMe interface and a small modification in the flash firmware, while keeping most of the complexity in the host OS. Our evaluation shows that* IODA *improves the 95–99.99th latencies by up to 75×.* IODA *is also the nearest to the ideal, no disturbance case compared to 7 state-of-the-art preemption, suspension, GC coordination, partitioning, tiny-tail flash controller, prediction, and proactive approaches.*

CCS Concepts

• Computer systems organization → Firmware; Embedded hardware; Embedded software; • Information systems → Flash memory; • Hardware → Emerging interfaces.

SOSP '21, October 26–29, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-8709-5/21/10…$15.00
https://doi.org/10.1145/3477132.3483573

Keywords

Software/Hardware Co-Design, Predictable Latency, NVMe I/O Determinism, SSD, Flash Storage

ACM Reference Format:

Huaicheng Li, Martin L. Putra, Ronald Shi, Xing Lin, Gregory R. Ganger, and Haryadi S. Gunawi. 2021. IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage. In *ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21), October 26–29, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3477132.3483573

1 Introduction

Flash arrays are popular storage choices in data centers and they must address users' craving for low and predictable latencies [1–3]. Thus, many recent SSD products are released and evaluated not just on the average speed but the percentile latencies as well [4–7]. These all paint the reality that customers would like SSDs with *deterministic* latencies.

Deterministic latency, however, is hard to achieve because SSD performance is inherently non-deterministic due to the internal management activities such as the garbage collection (GC) process, wear leveling, and internal buffer flush [8–10]. These activities will inevitably trigger many background I/Os and disturb user requests. Notably, GC is a necessary path to overcome NAND Flash's inability for in-place overwrites. It involves time-consuming data movement to reclaim space and contend with user requests, thereby causing severe latency hiccups. As an illustration, the figure on the right shows the giant latency gap between the "Base" (with GC) and the "Ideal" (no GC) cases. Modern SSDs often resort to large over-provisioning space (*e.g.*, up to 50% of the SSD's raw NAND capacity) [11] to provide legroom for more efficienct background task processing, however, our profiling experiments on recent enterprise SSDs showed that GCs can still cause up to 60× latency increase (details omitted). This is unfortunately still an ongoing problem faced by the storage industry [12–14].

To tame the SSD performance challenges, there have been many efforts to evolve the device interfaces [15–17]. The Stor-

IODA Stack and Evaluation Setup

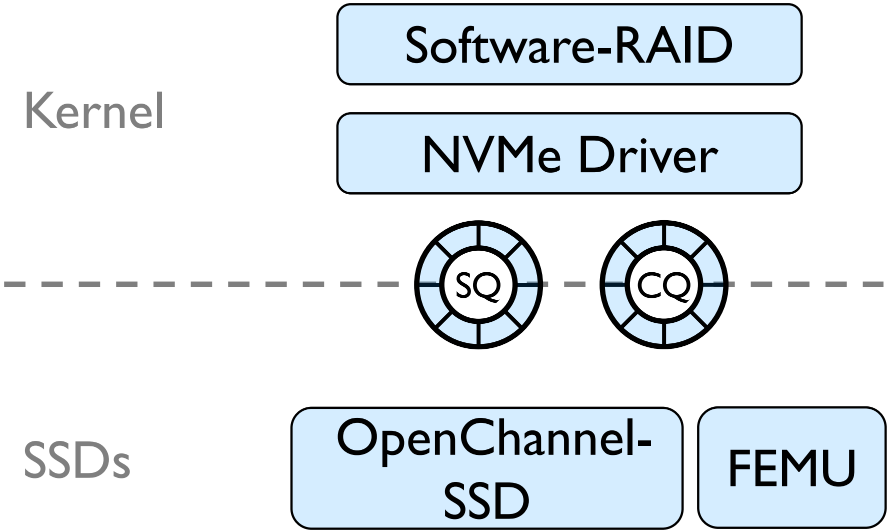
Kernel

SSDs

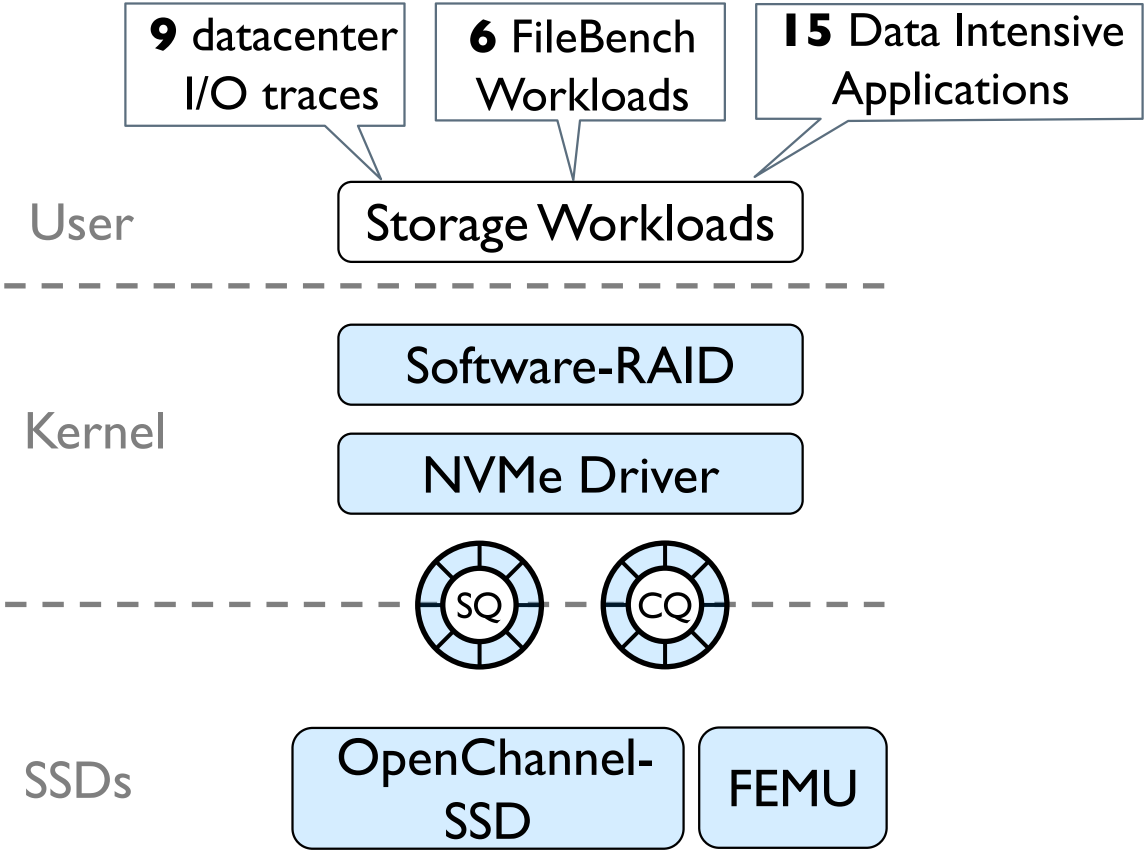
OpenChannel-SSD

FEMU

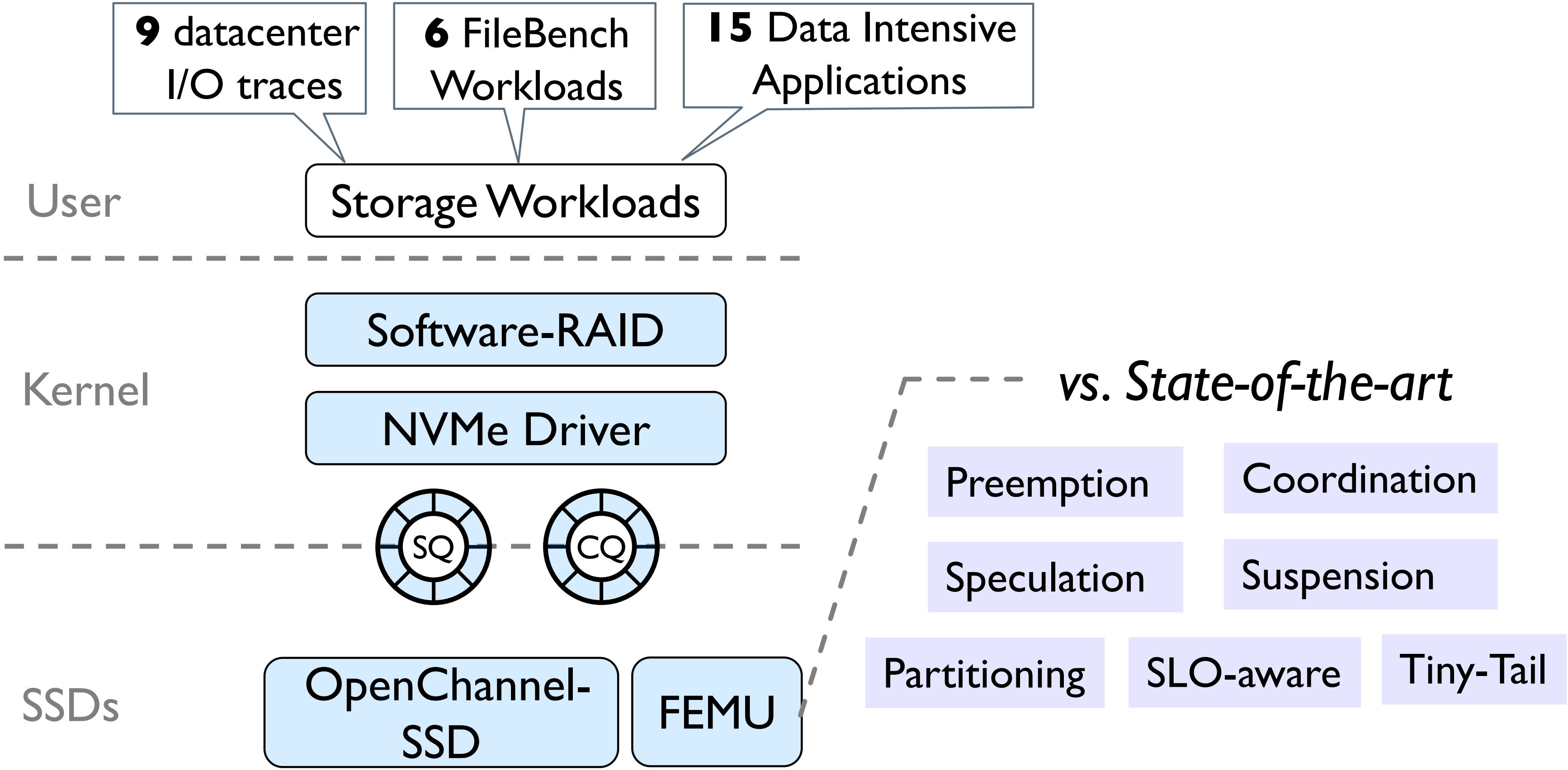
IODA Stack and Evaluation Setup



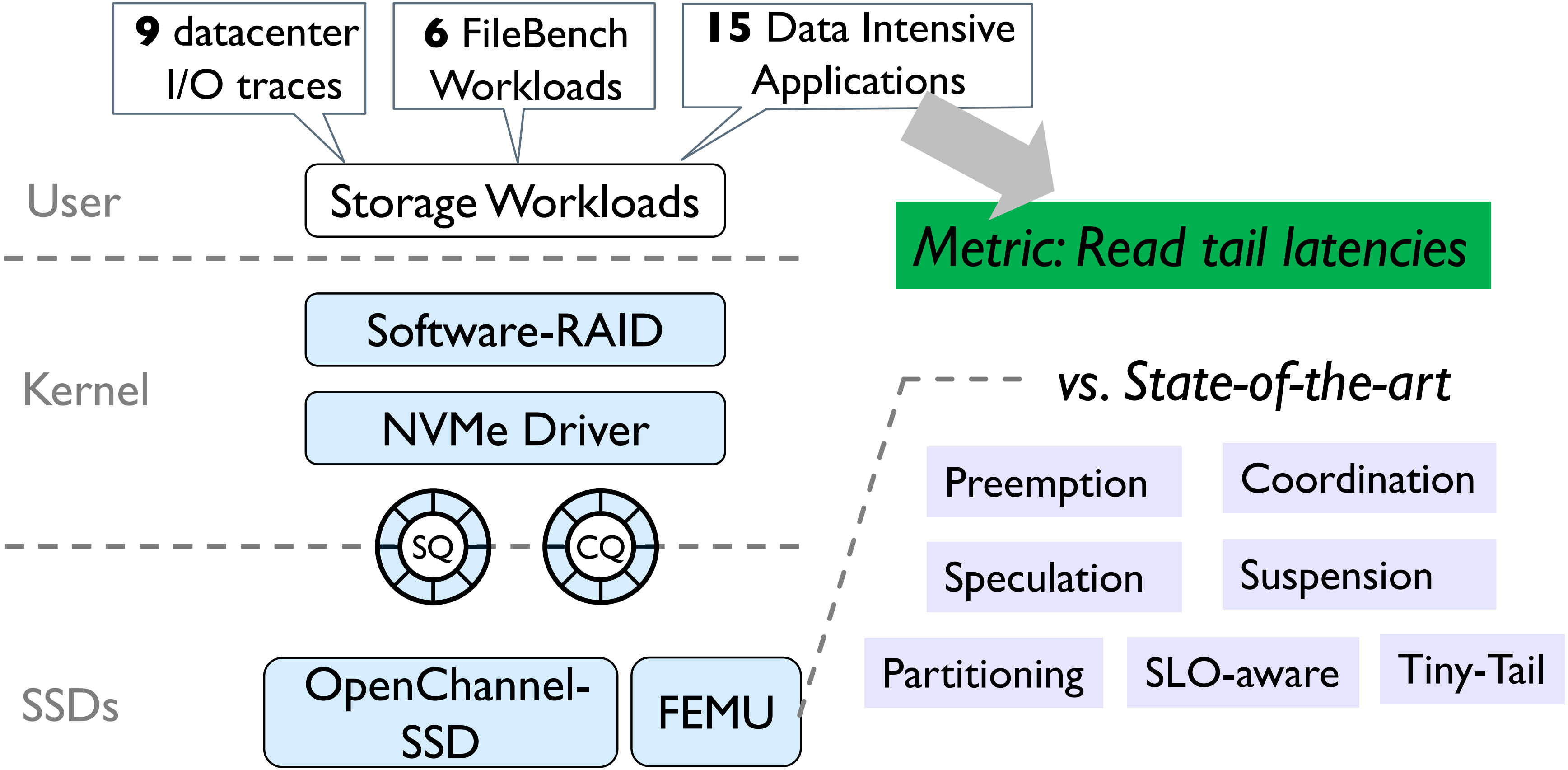
IODA Stack and Evaluation Setup



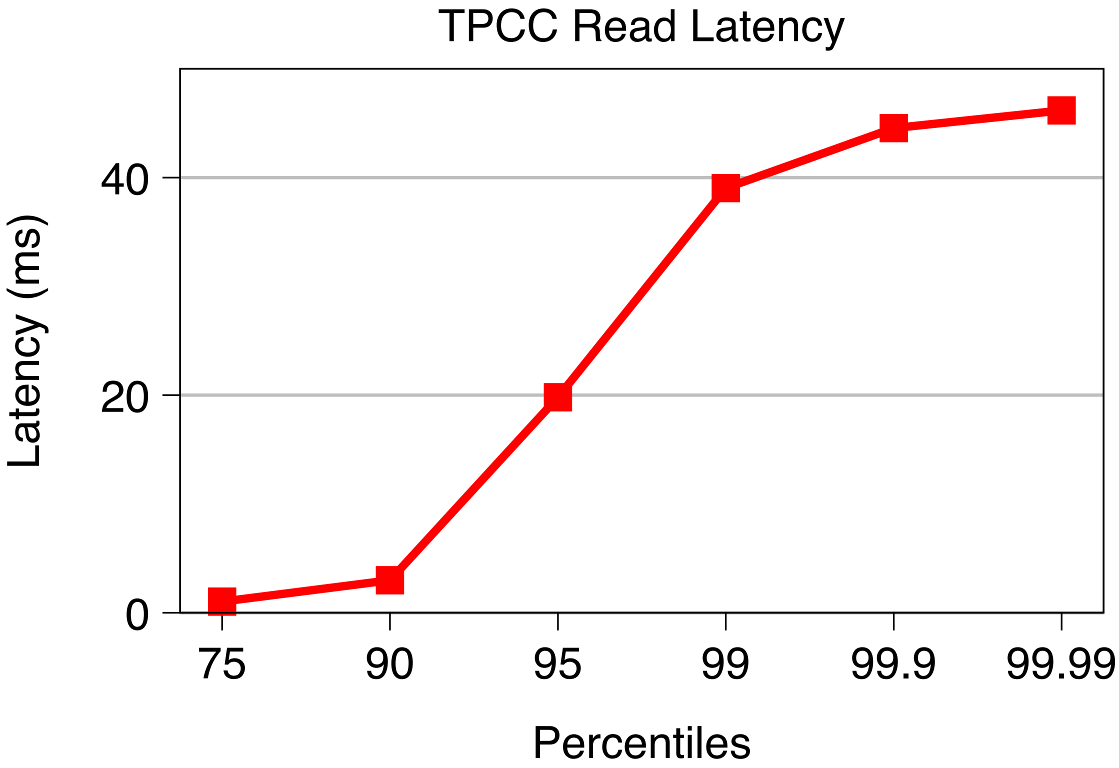
IODA Stack and Evaluation Setup



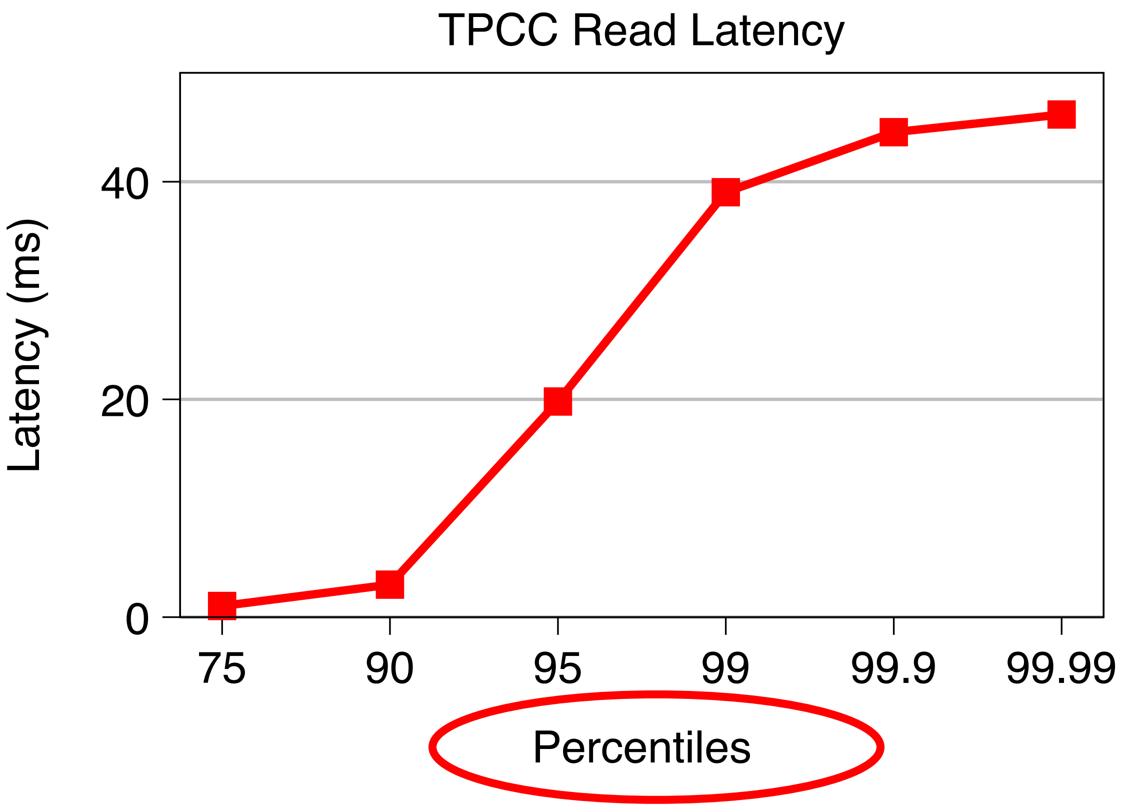
IODA Stack and Evaluation Setup



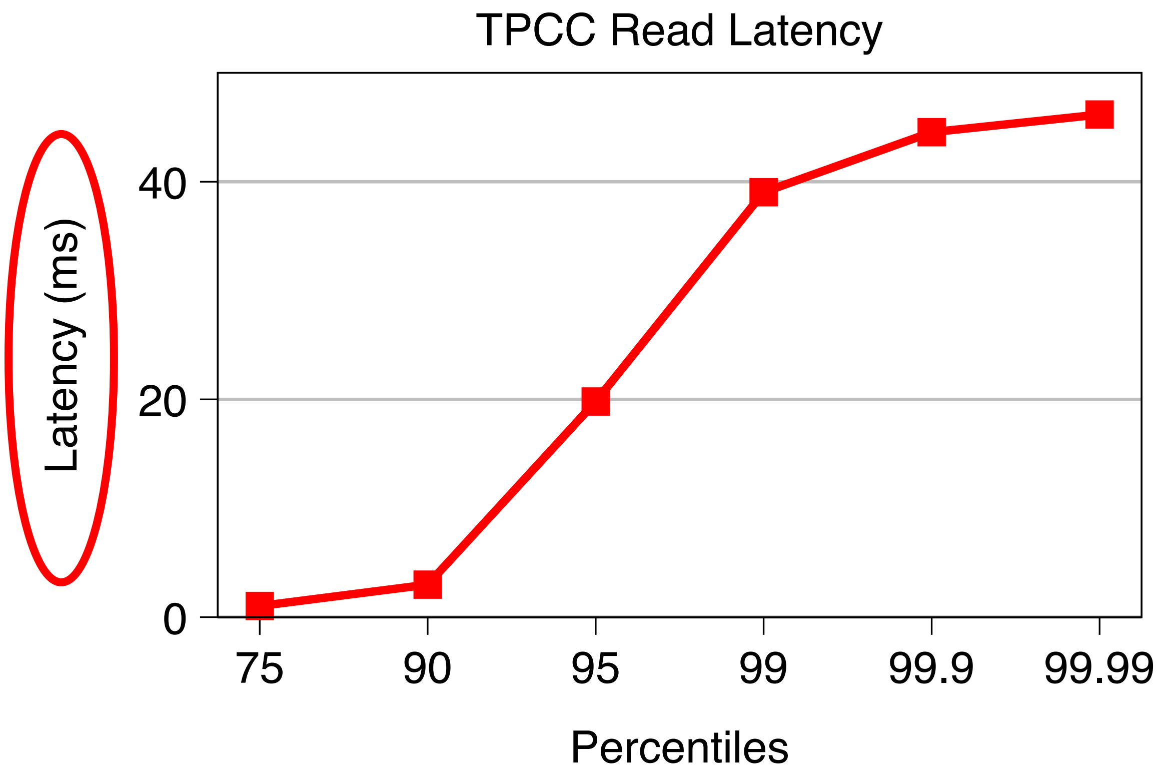
IODA Evaluation



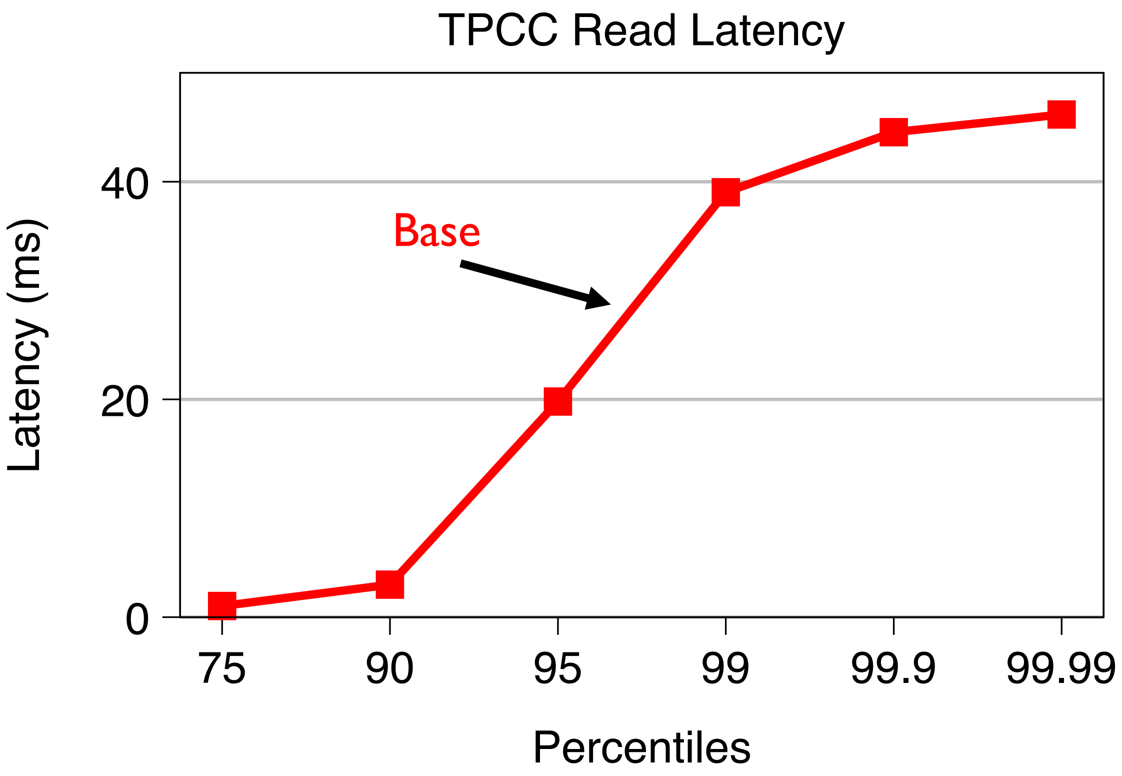
IODA Evaluation



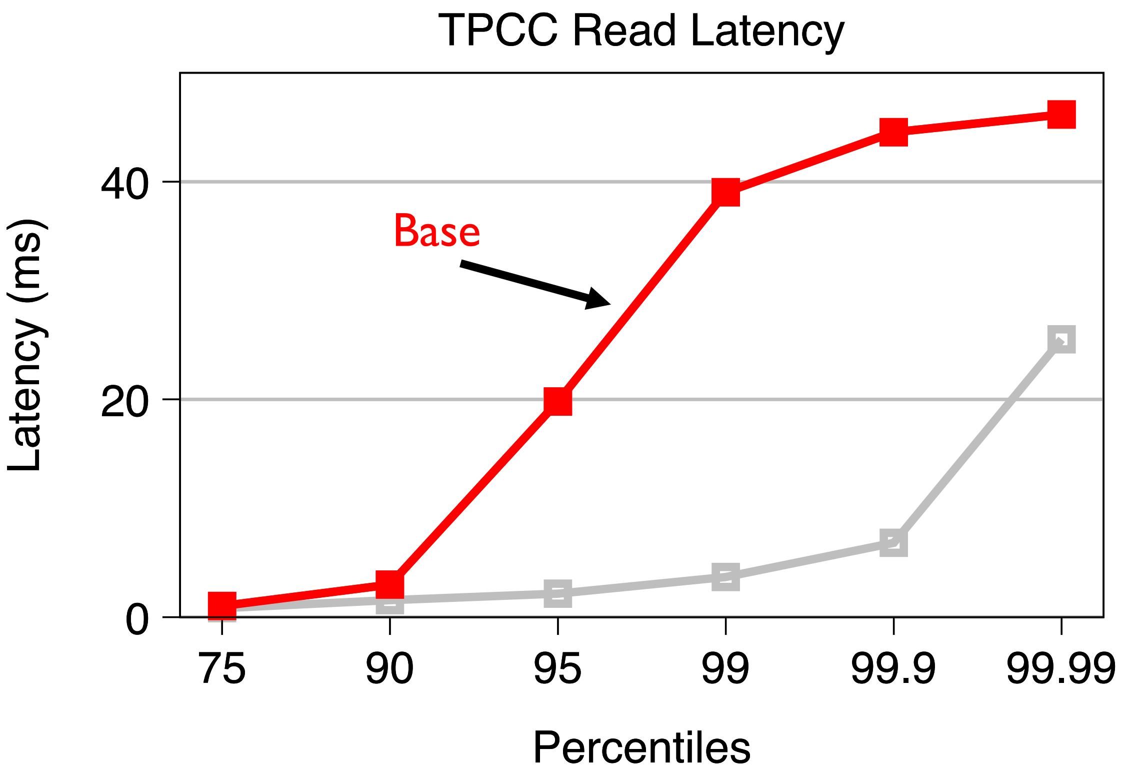
IODA Evaluation



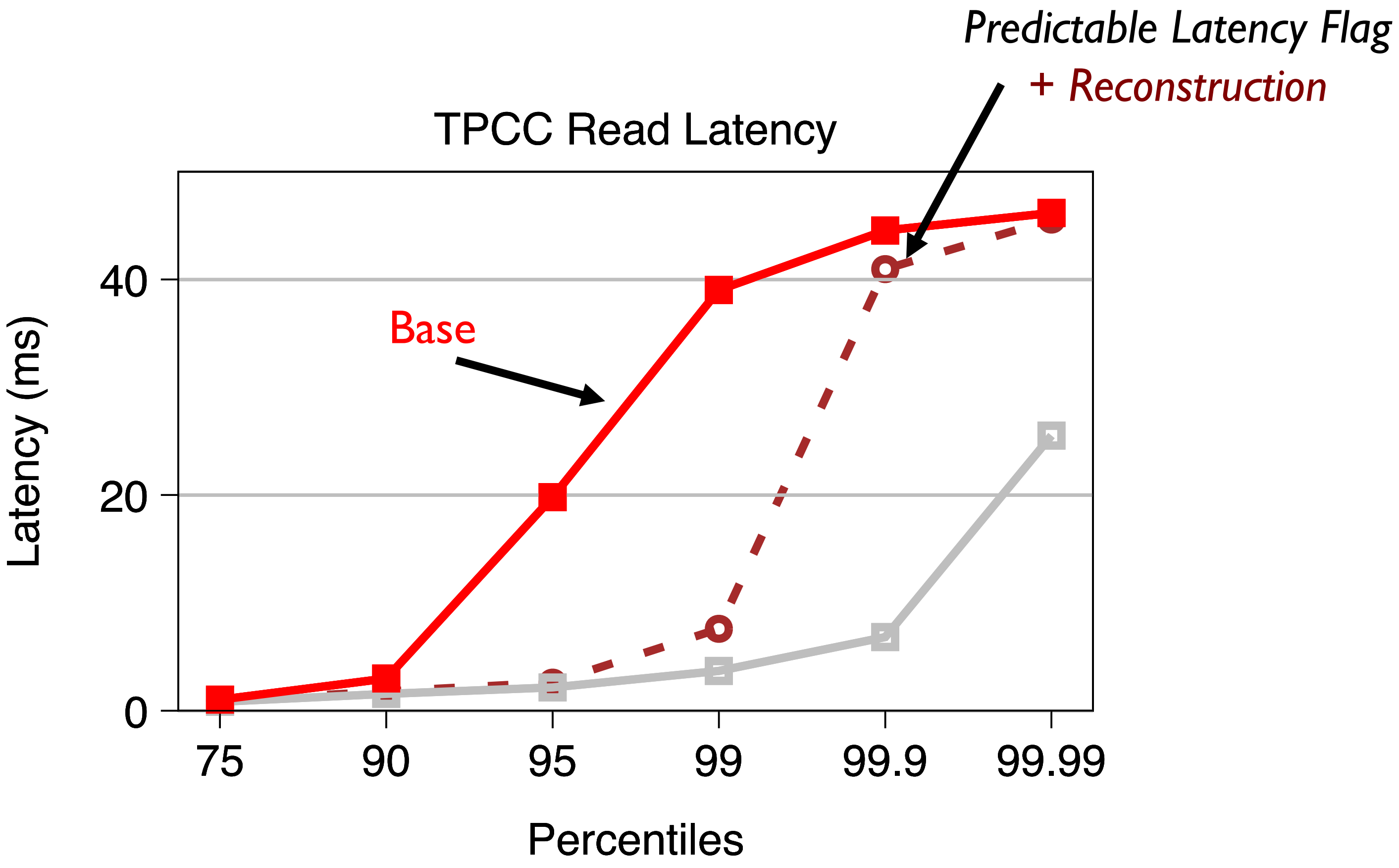
IODA Evaluation



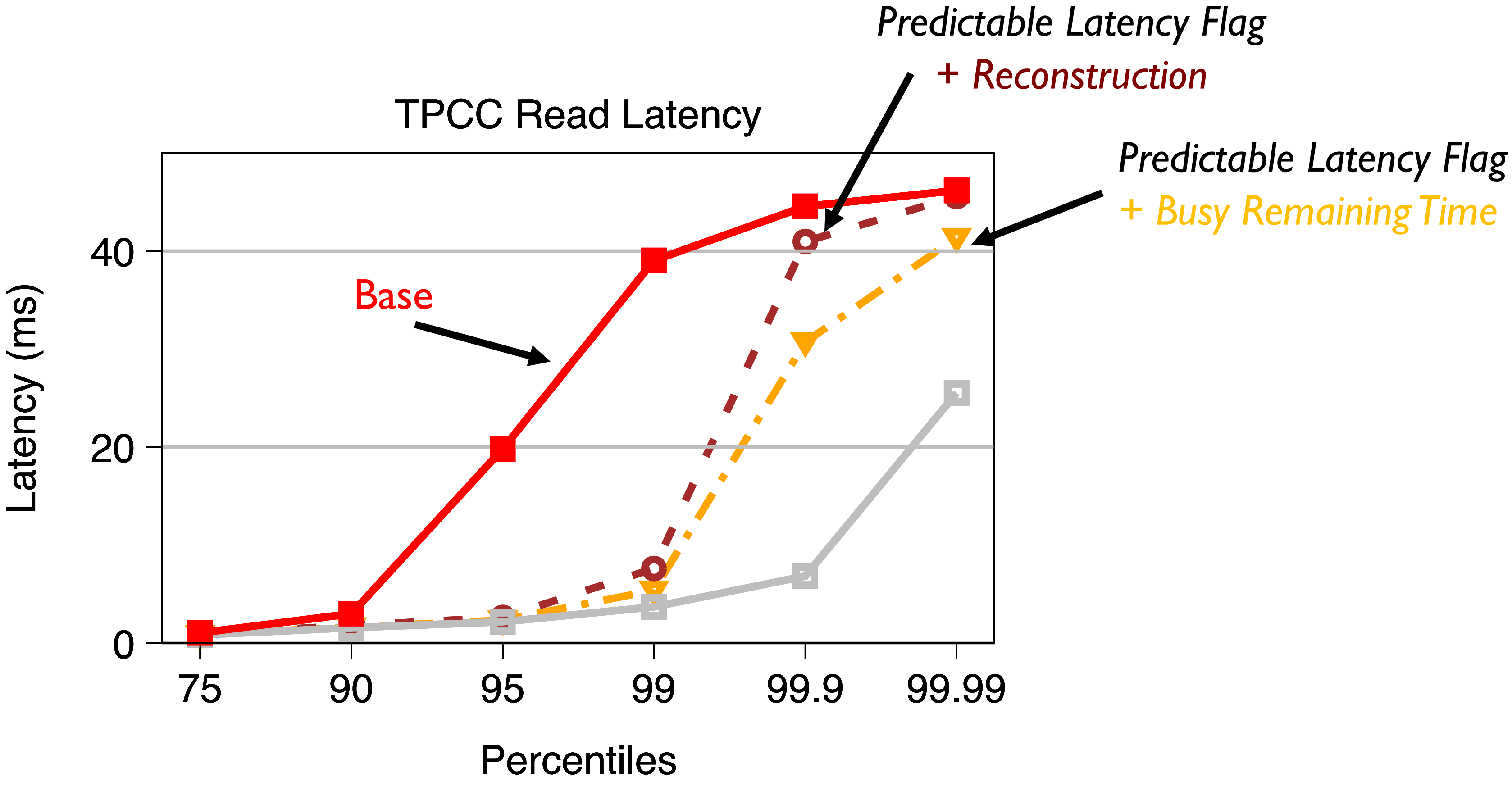
IODA Evaluation



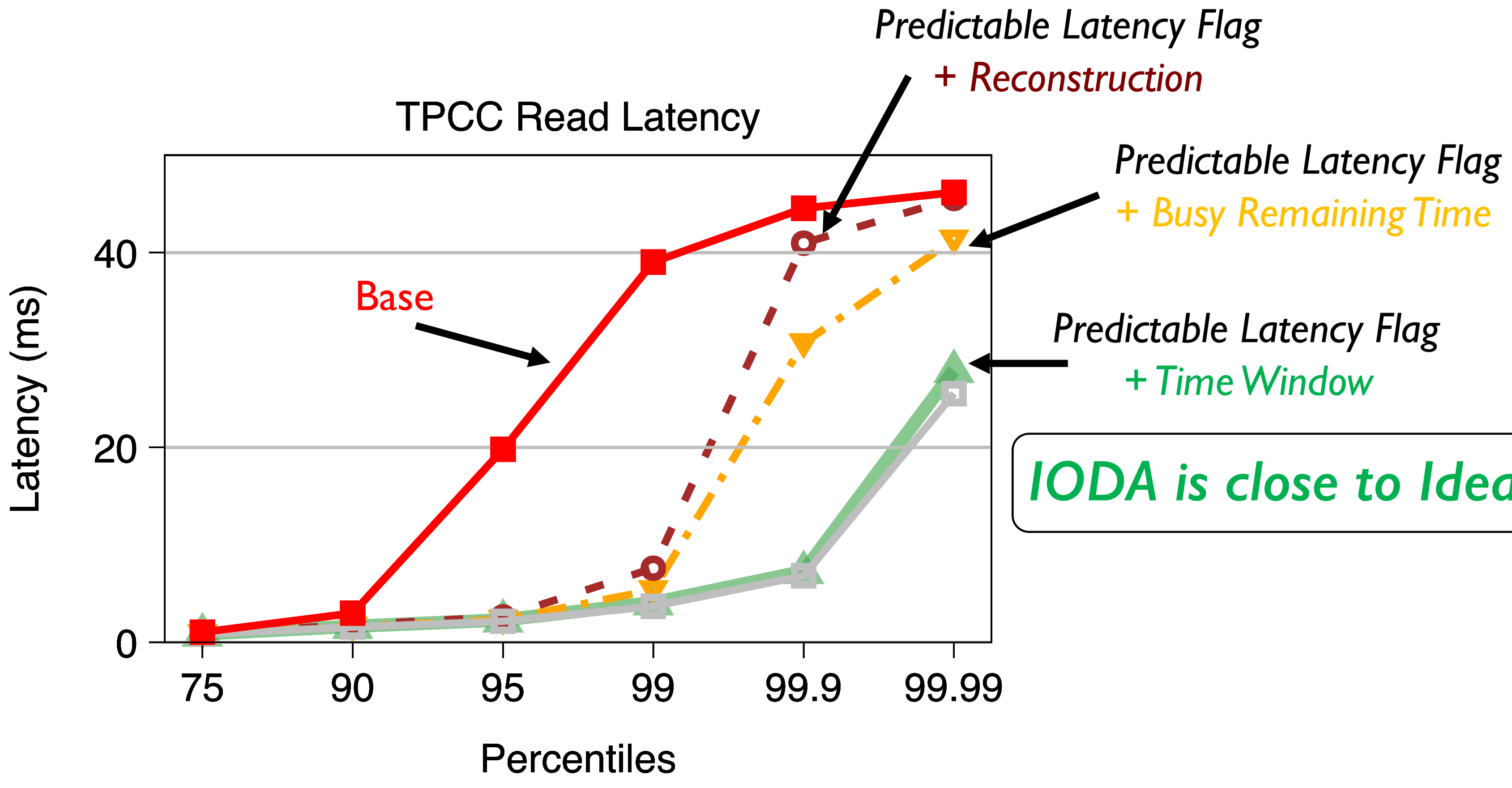
IODA Evaluation



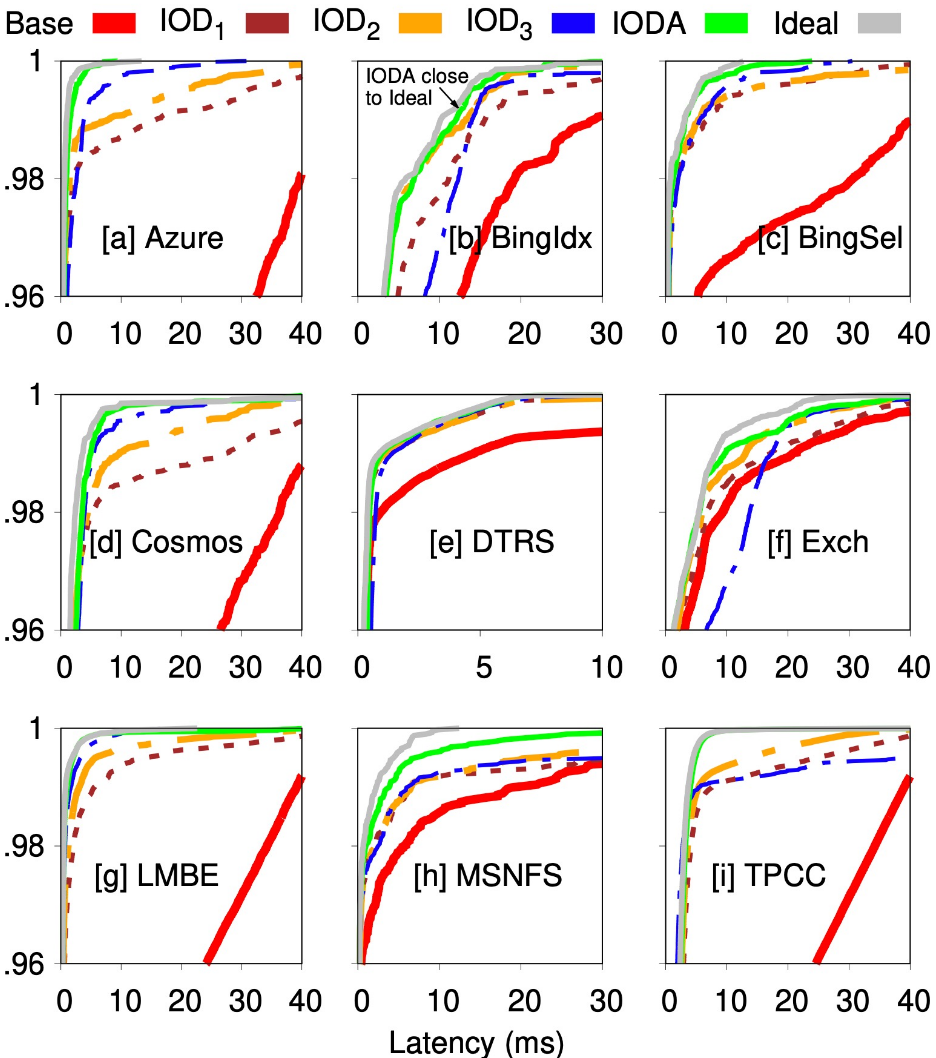
IODA Evaluation

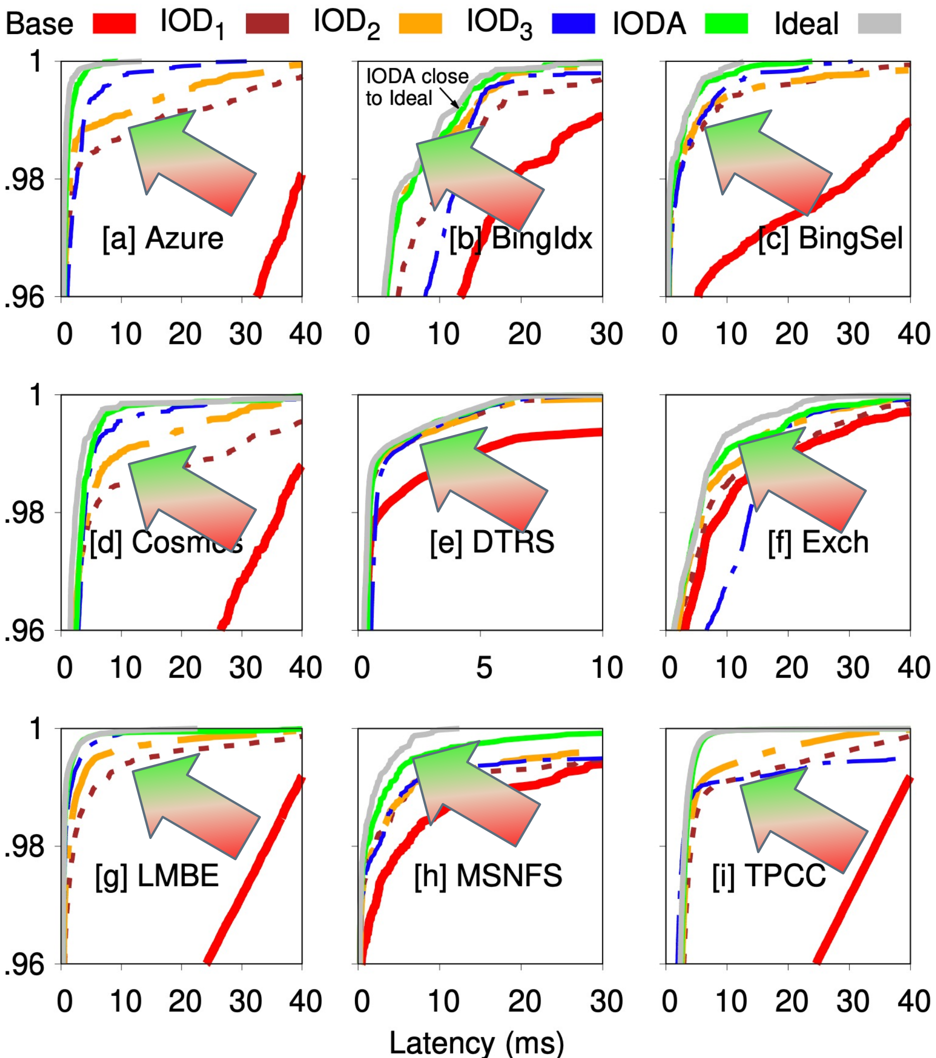


IODA Evaluation



IODA is close to Ideal!

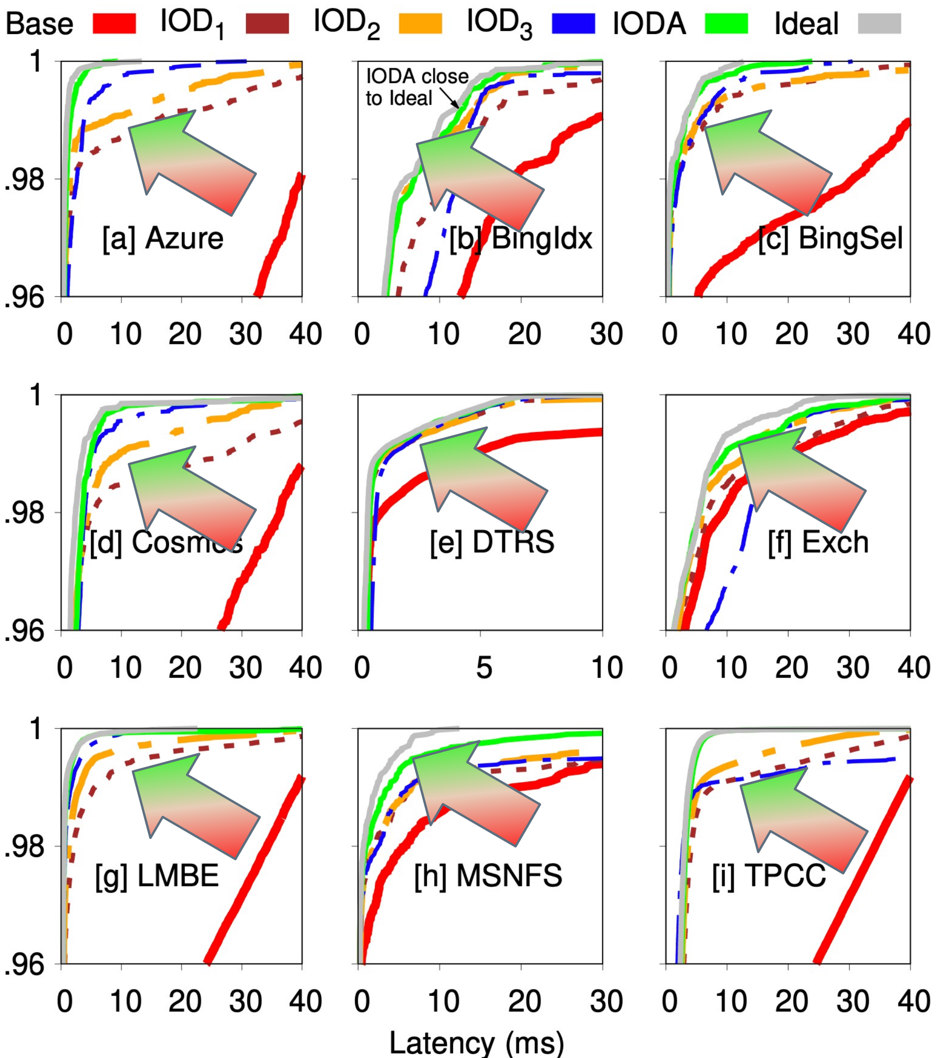

Base
IOD_1
IOD_2
IOD_3
IODA
Ideal
IODA close to Ideal
[a] Azure
[b] BingIdx
[c] BingSel
[d] Cosmos
[e] DTRS
[f] Exch
[g] LMBE
[h] MSNFS
[i] TPCC
Latency (ms)

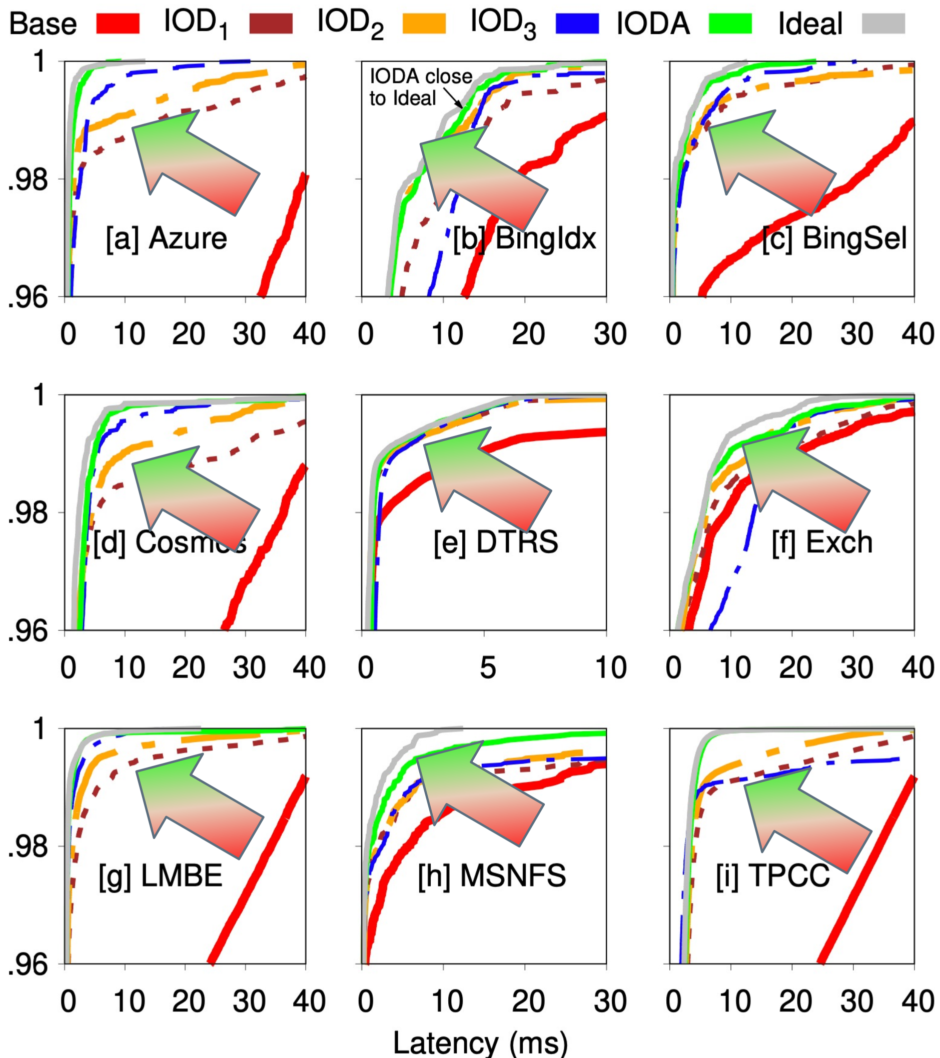


Base
IOD1
IOD2
IOD3
IODA
Ideal
IODA close to Ideal
[a] Azure
[b] BingIdx
[c] BingSel
[d] Cosmos
[e] DTRS
[f] Exch
[g] LMBE
[h] MSNFS
[i] TPCC
1
.98
.96
0 10 20 30 40
0 10 20 30
0 5 10
Latency (ms)

IODA Results: *(95th – 99.99th)*

Up to 75x improvement *over Base*





IODA Results: *(95th – 99.99th)*

Up to 75x improvement *over Base*

vs.

Preemption

Coordination

Speculation

Suspension

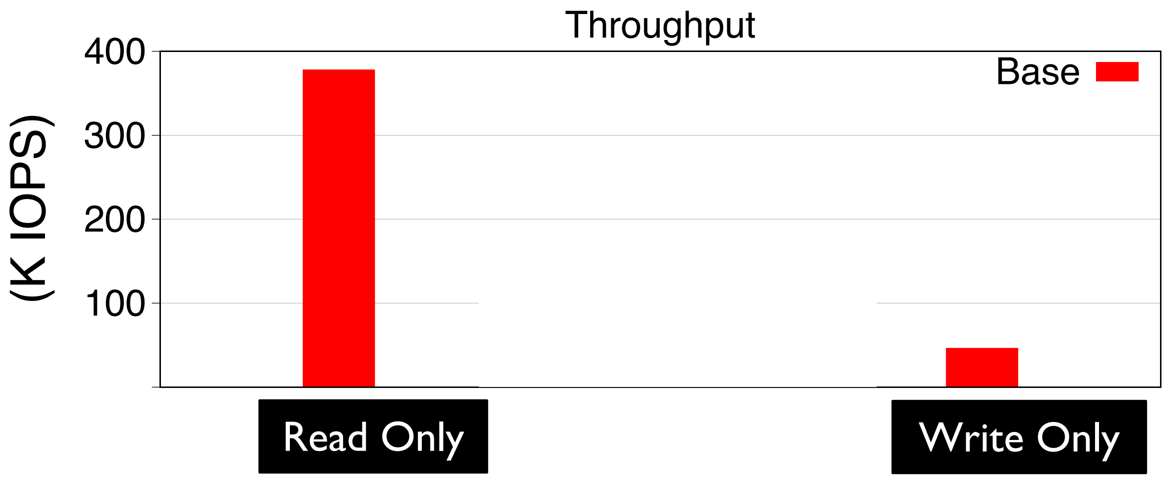
Partitioning

SLO-aware

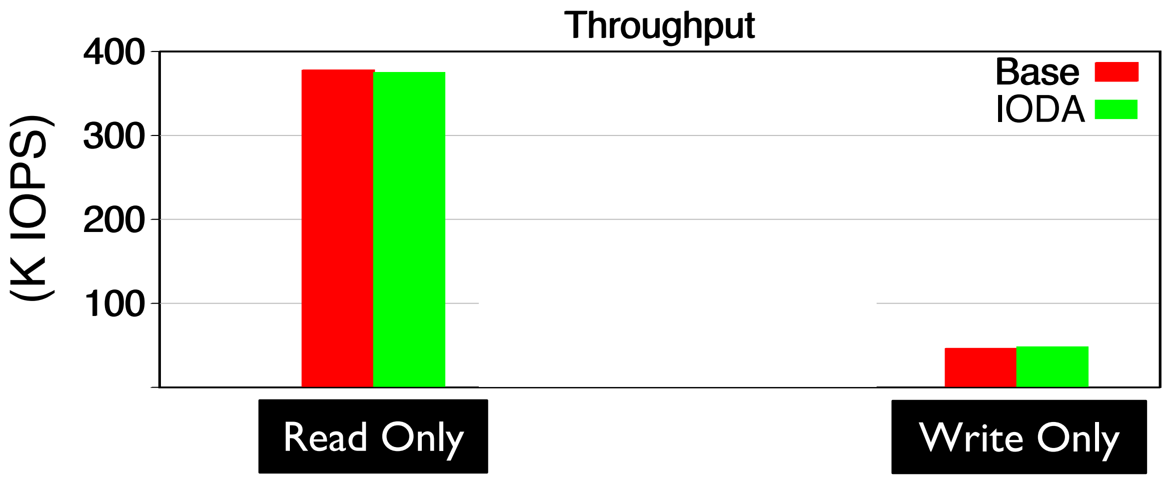
Tiny-Tail

IODA is more deterministic and efficient in cutting tail latencies!

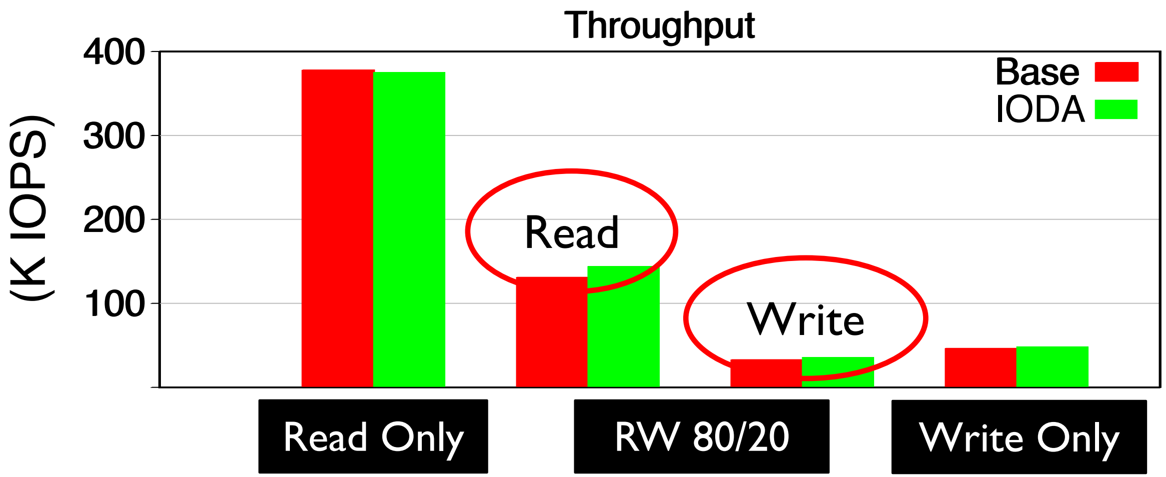
IODA Throughput



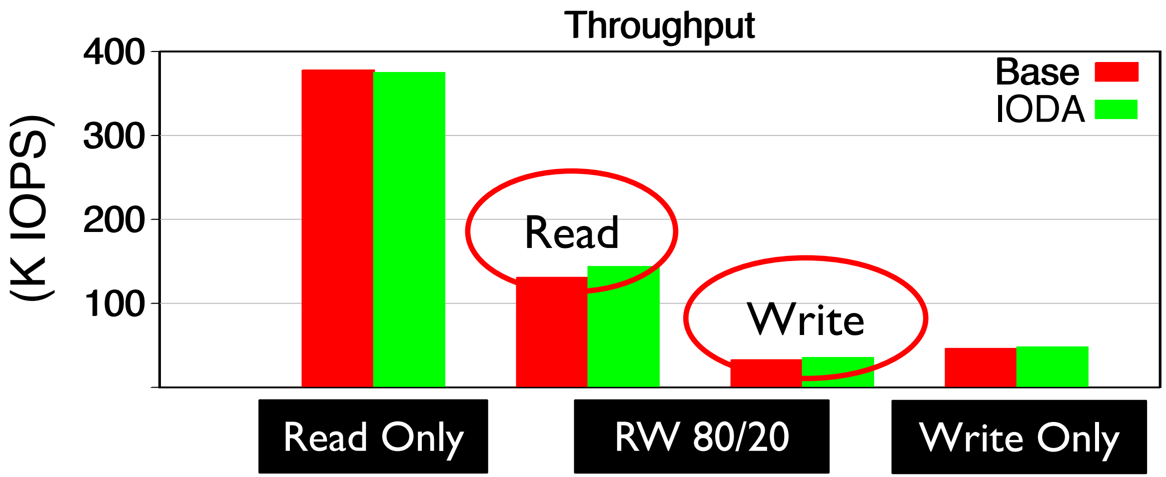
IODA Throughput



IODA Throughput



IODA Throughput



IODA doesn't sacrifice the array's aggregate bandwidth

IODA Takeaways

- A *Co-Design* Approach for Performance Predictability
  - Proactive *reconstruction* via *fast-fail* interface
  - *BRT* for improved latencies
  - *TW* formulation to program the window length
  - Cross-device synchronization

I'm on the job market.

IODA: https://github.com/huaicheng/IODA

IODA Takeaways

- A *Co-Design* Approach for Performance Predictability
  - Proactive *reconstruction* via *fast-fail* interface
  - *BRT* for improved latencies
  - *TW* formulation to program the window length
  - Cross-device synchronization

Thank you!

I'm on the job market.

IODA: https://github.com/huaicheng/IODA