



# FLIN:

## Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives

**Saugata Ghose**

Carnegie Mellon University



Flash Memory Summit

August 7, 2019  
Santa Clara, CA

- Modern solid-state drives (SSDs) use new storage protocols (e.g., NVMe) that **eliminate the OS software stack**
  - I/O requests are now scheduled inside the SSD
  - Enables **high throughput**: millions of IOPS
- OS software stack elimination **removes existing fairness mechanisms**
  - We **experimentally characterize fairness** on four real state-of-the-art SSDs
  - **Highly unfair slowdowns**: large difference across concurrently-running applications
- We find and analyze **four sources of inter-application interference** that lead to slowdowns in state-of-the-art SSDs
- **FLIN**: a new I/O request scheduler for modern SSDs designed to **provide both fairness and high performance**
  - Mitigates all four sources of inter-application interference
  - Implemented fully in the SSD controller firmware, uses < 0.06% of DRAM space
  - FLIN improves **fairness by 70%** and **performance by 47%** compared to a state-of-the-art I/O scheduler

## Background: Modern SSD Design

Unfairness Across Multiple Applications  
in Modern SSDs

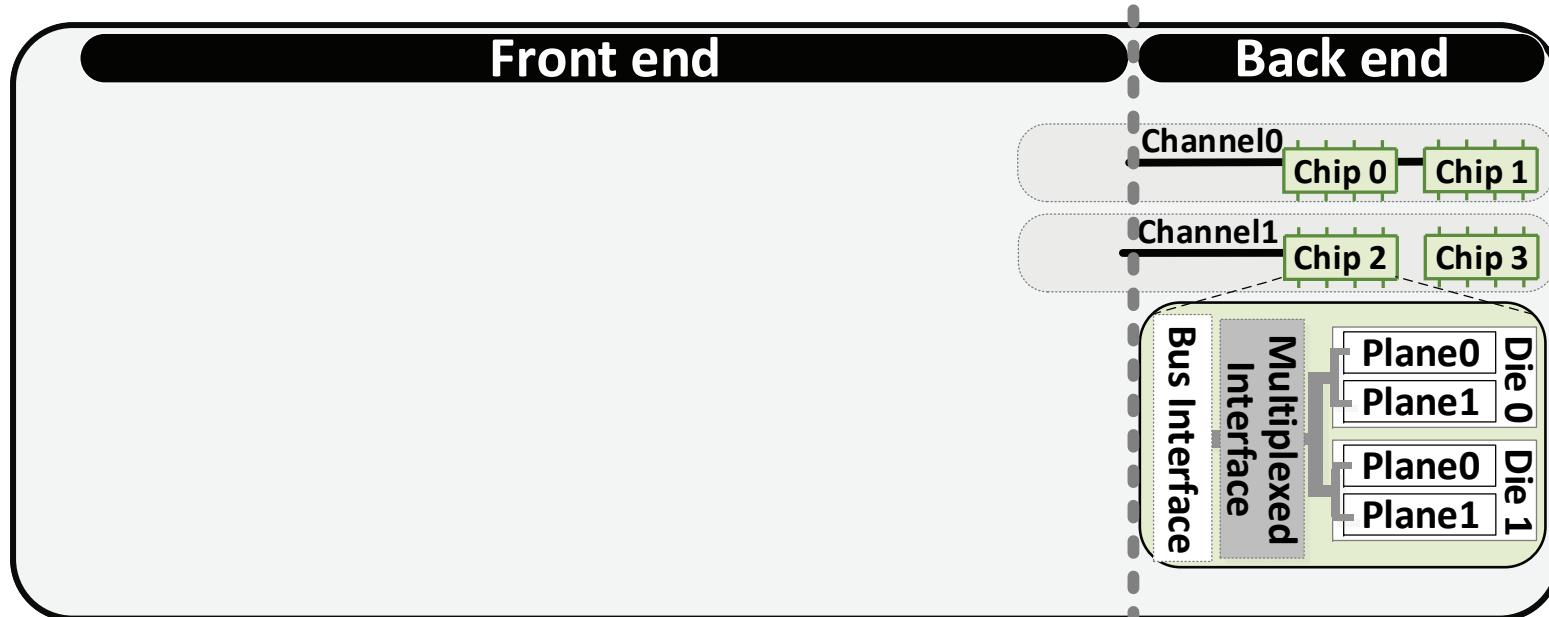
FLIN:  
Flash-Level INterference-aware SSD Scheduler

Experimental Evaluation

Conclusion

# Internal Components of a Modern SSD

SAFARI

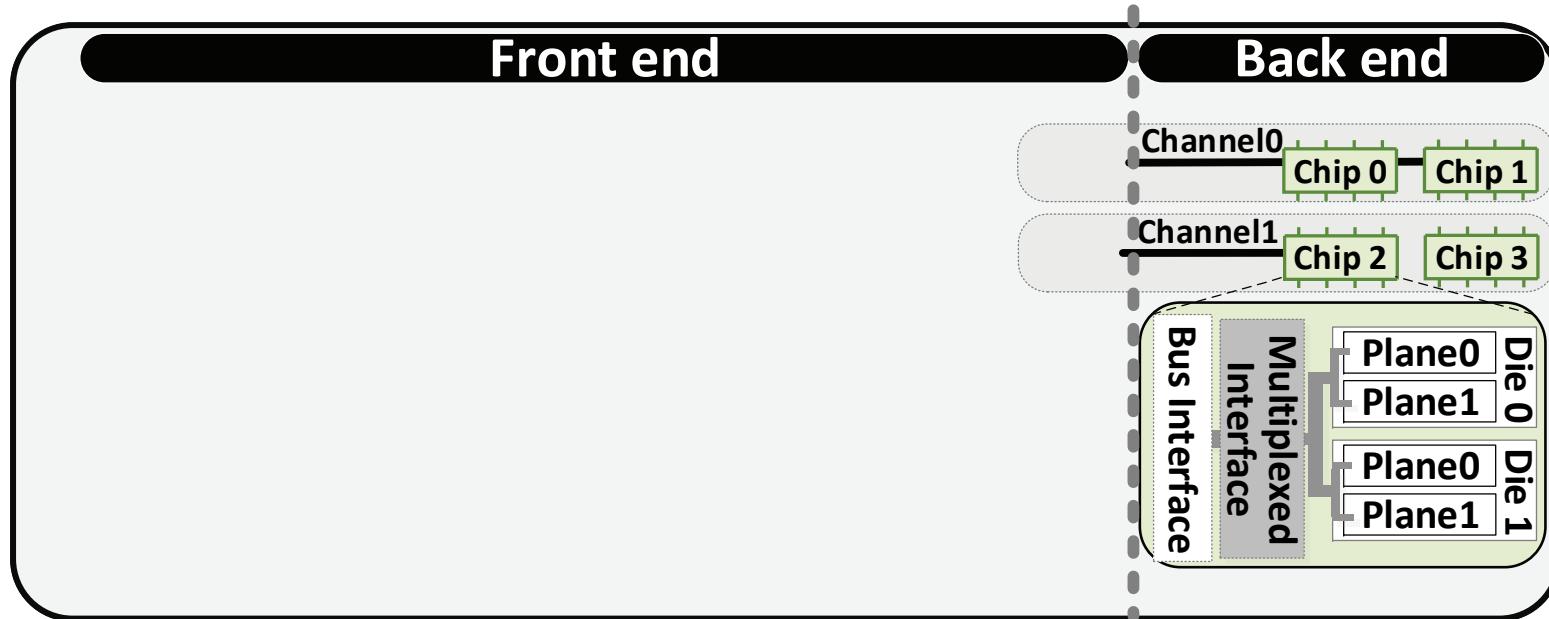


## ■ Back End: data storage

- Memory chips (e.g., NAND flash memory, PCM, MRAM, 3D XPoint)

# Internal Components of a Modern SSD

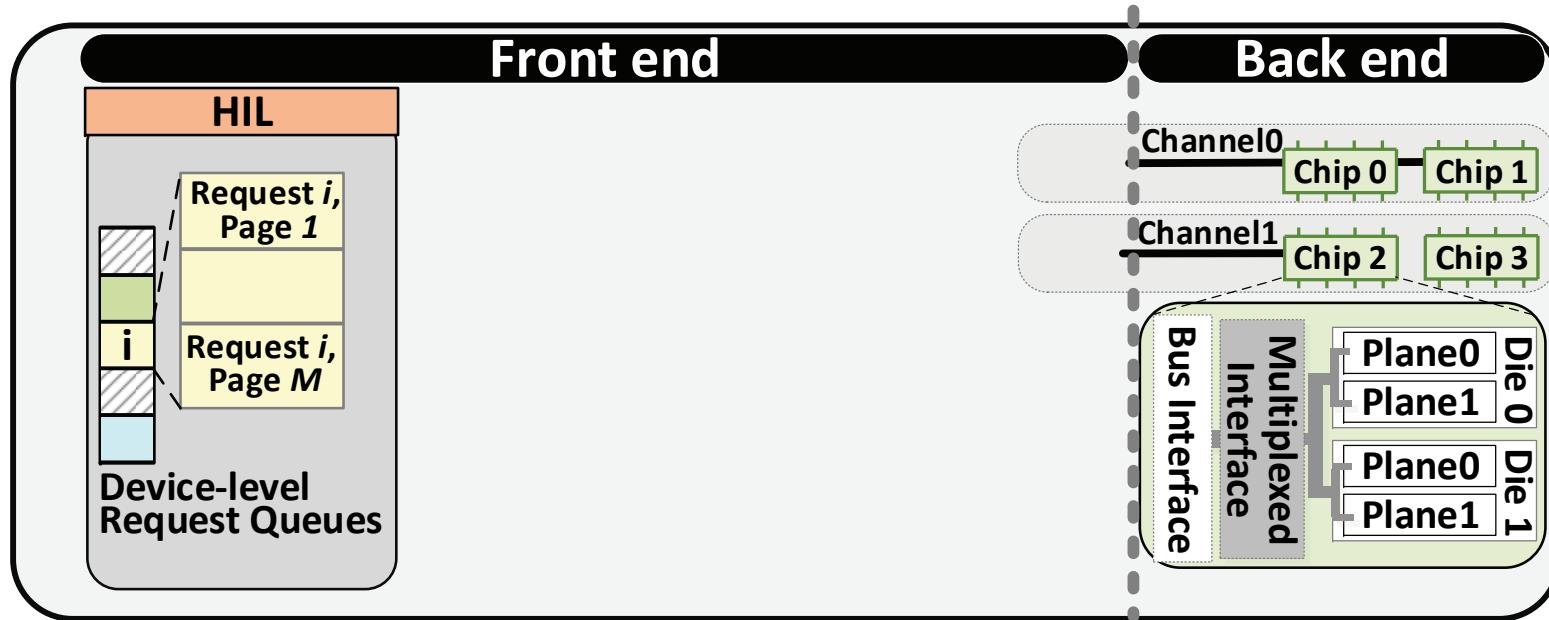
SAFARI



- **Back End:** data storage
  - Memory chips (e.g., NAND flash memory, PCM, MRAM, 3D XPoint)
- **Front End:** management and control units

# Internal Components of a Modern SSD

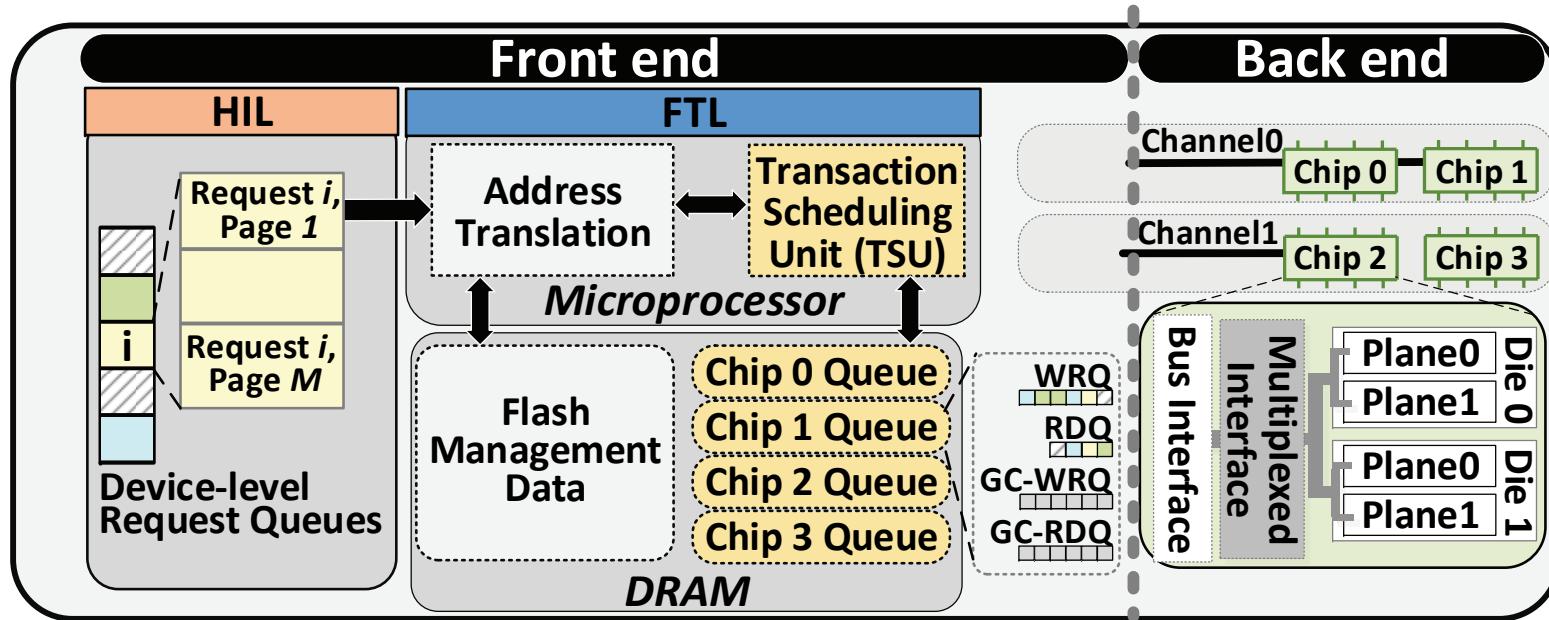
SAFARI



- **Back End:** data storage
  - Memory chips (e.g., NAND flash memory, PCM, MRAM, 3D XPoint)
- **Front End:** management and control units
  - **Host–Interface Logic (HIL):** protocol used to communicate with host

# Internal Components of a Modern SSD

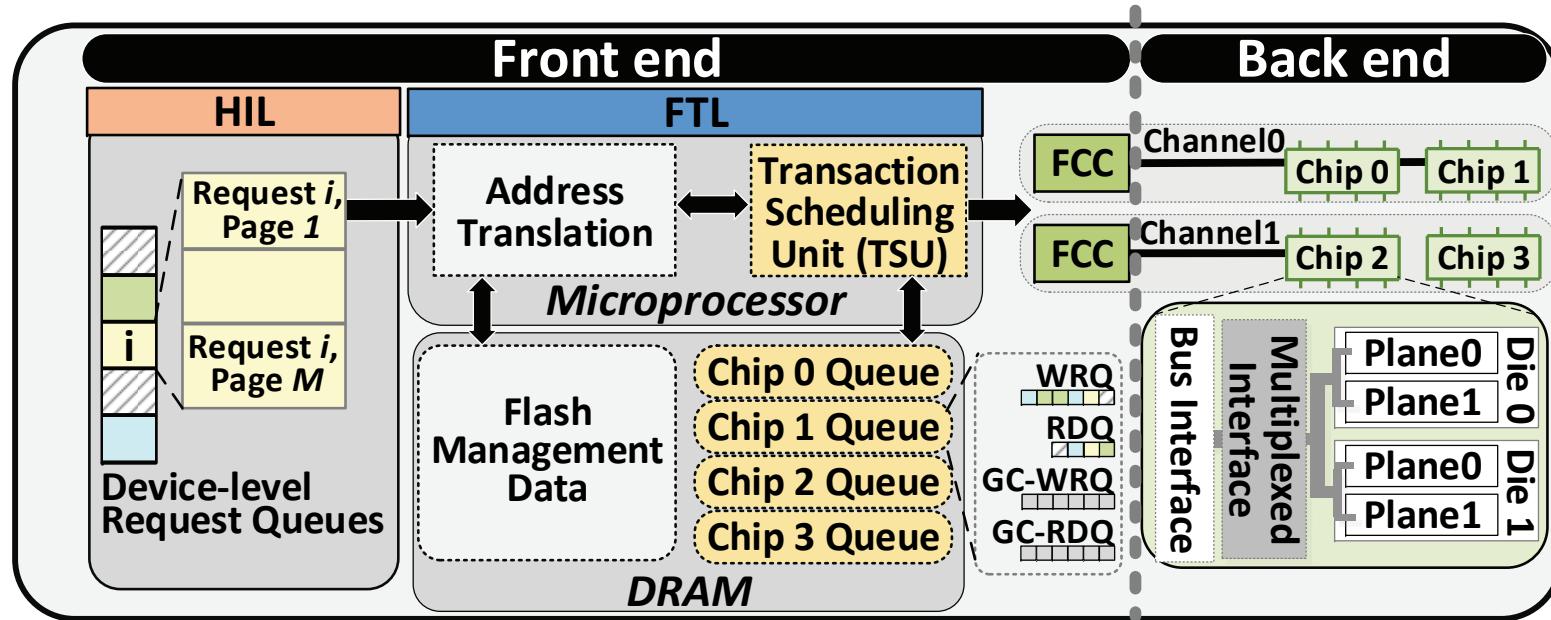
SAFARI



- **Back End:** data storage
  - Memory chips (e.g., NAND flash memory, PCM, MRAM, 3D XPoint)
- **Front End:** management and control units
  - Host-Interface Logic (HIL): protocol used to communicate with host
  - **Flash Translation Layer (FTL):** manages resources, processes I/O requests

# Internal Components of a Modern SSD

SAFARI

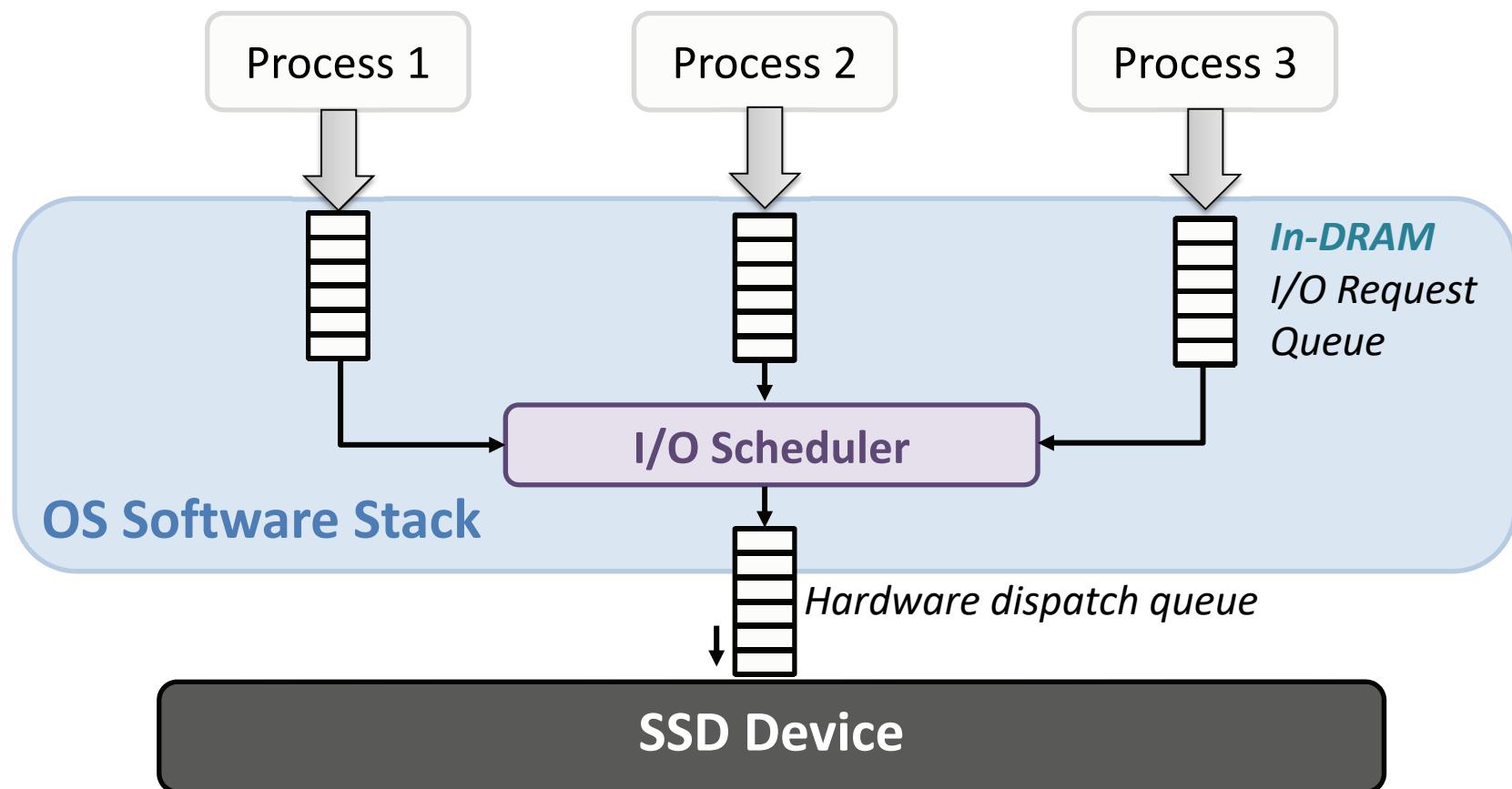


- **Back End:** data storage
  - Memory chips (e.g., NAND flash memory, PCM, MRAM, 3D XPoint)
- **Front End:** management and control units
  - Host-Interface Logic (HIL): protocol used to communicate with host
  - Flash Translation Layer (FTL): manages resources, processes I/O requests
  - **Flash Channel Controllers (FCCs):** sends commands to, transfers data with memory chips in back end

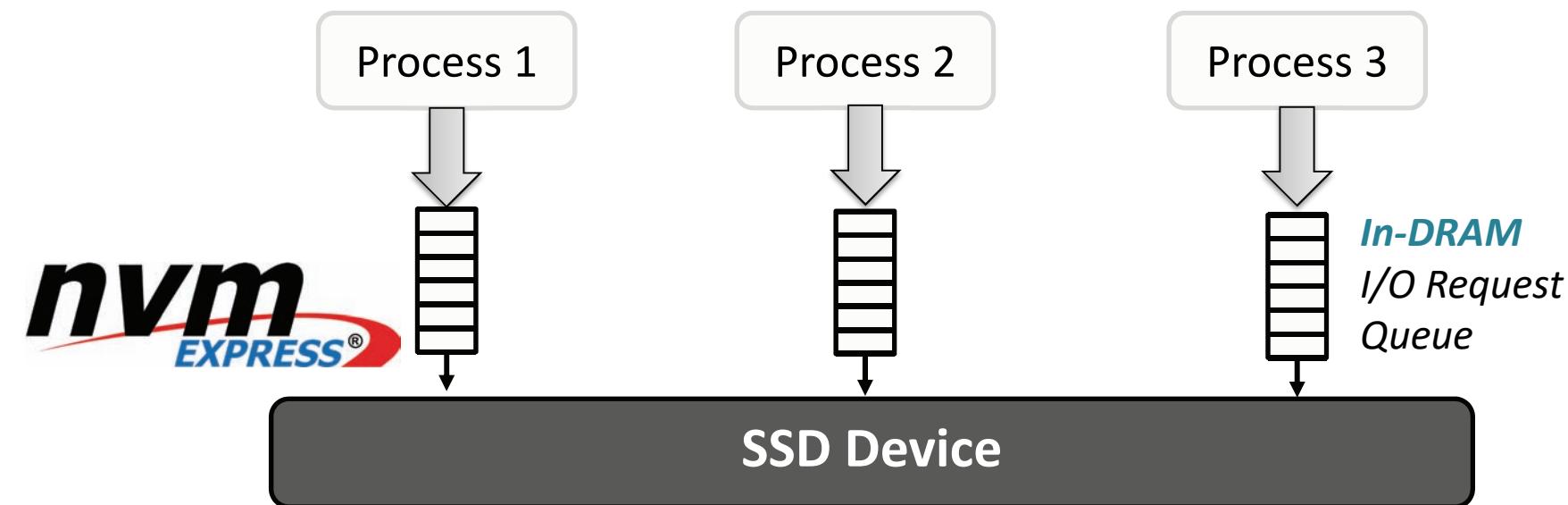
# Conventional Host–Interface Protocols for SSDs

SAFARI

- SSDs initially adopted **conventional** host–interface protocols (e.g., SATA)
  - Designed for magnetic hard disk drives
  - Maximum of only *thousands of IOPS* per device



- Modern SSDs use **high-performance** host–interface protocols (e.g., NVMe)
  - Bypass OS intervention: **SSD must perform scheduling**
  - Take advantage of SSD throughput: enables ***millions of IOPS*** per device



Fairness mechanisms in OS software stack are also eliminated  
Do modern SSDs need to handle fairness control?

## Background: Modern SSD Design

# Unfairness Across Multiple Applications in Modern SSDs

FLIN:

Flash-Level INterference-aware SSD Scheduler

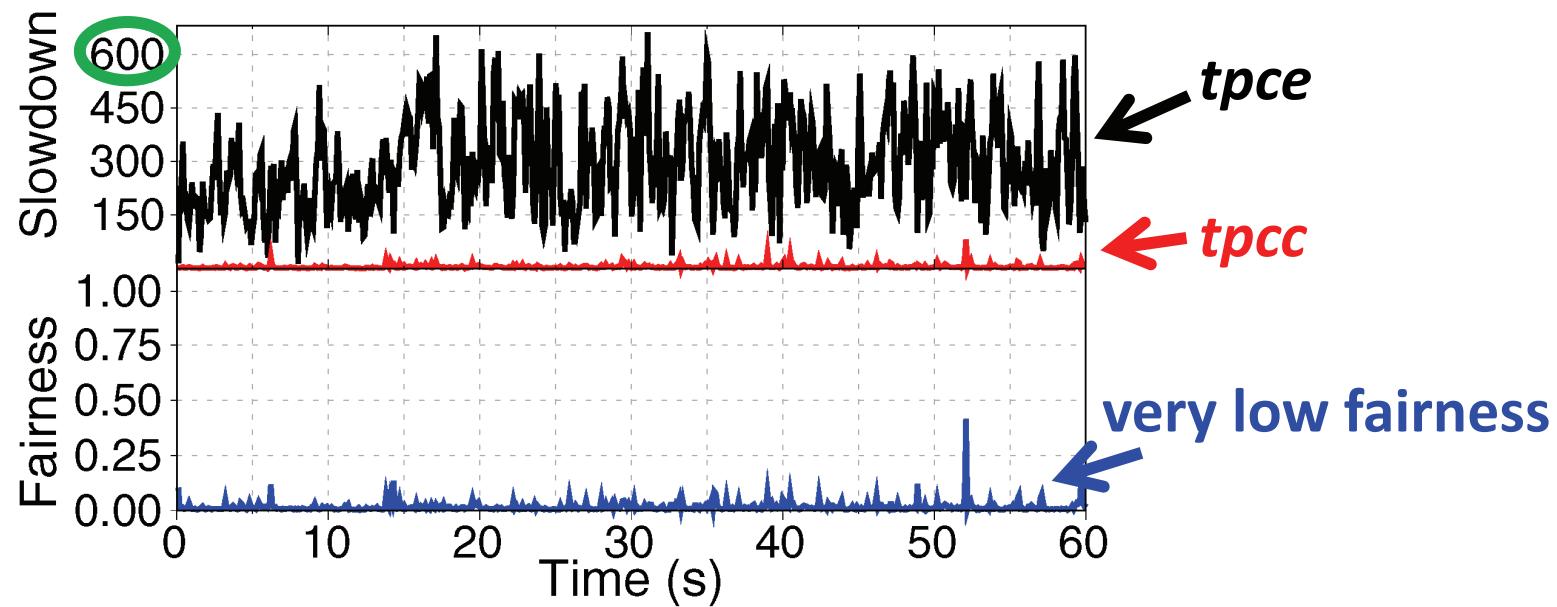
Experimental Evaluation

Conclusion

- We measure fairness using four **real state-of-the-art SSDs**
  - NVMe protocol
  - Designed for datacenters
- **Flow:** a series of I/O requests generated by an application
- **Slowdown** =  $\frac{\text{shared flow response time}}{\text{alone flow response time}}$  (*lower is better*)
- **Unfairness** =  $\frac{\text{max slowdown}}{\text{min slowdown}}$  (*lower is better*)
- **Fairness** =  $\frac{1}{\text{unfairness}}$  (*higher is better*)

# Representative Example: *tpcc* and *tpce*

SAFARI



average slowdown of *tpce*:  
*2x to 106x* across our four real SSDs

SSDs do not provide fairness  
among concurrently-running flows

# What Causes This Unfairness?

SAFARI

- Interference among concurrently-running flows
- We perform a **detailed study of interference**
  - MQSim: detailed, open-source modern SSD simulator [FAST 2018]  
<https://github.com/CMU-SAFARI/MQSim>
  - Run flows that are designed to demonstrate each source of interference
  - Detailed experimental characterization results in the paper
- We uncover four sources of interference among flows

- The I/O intensity of a flow affects the average queue wait time of flash transactions

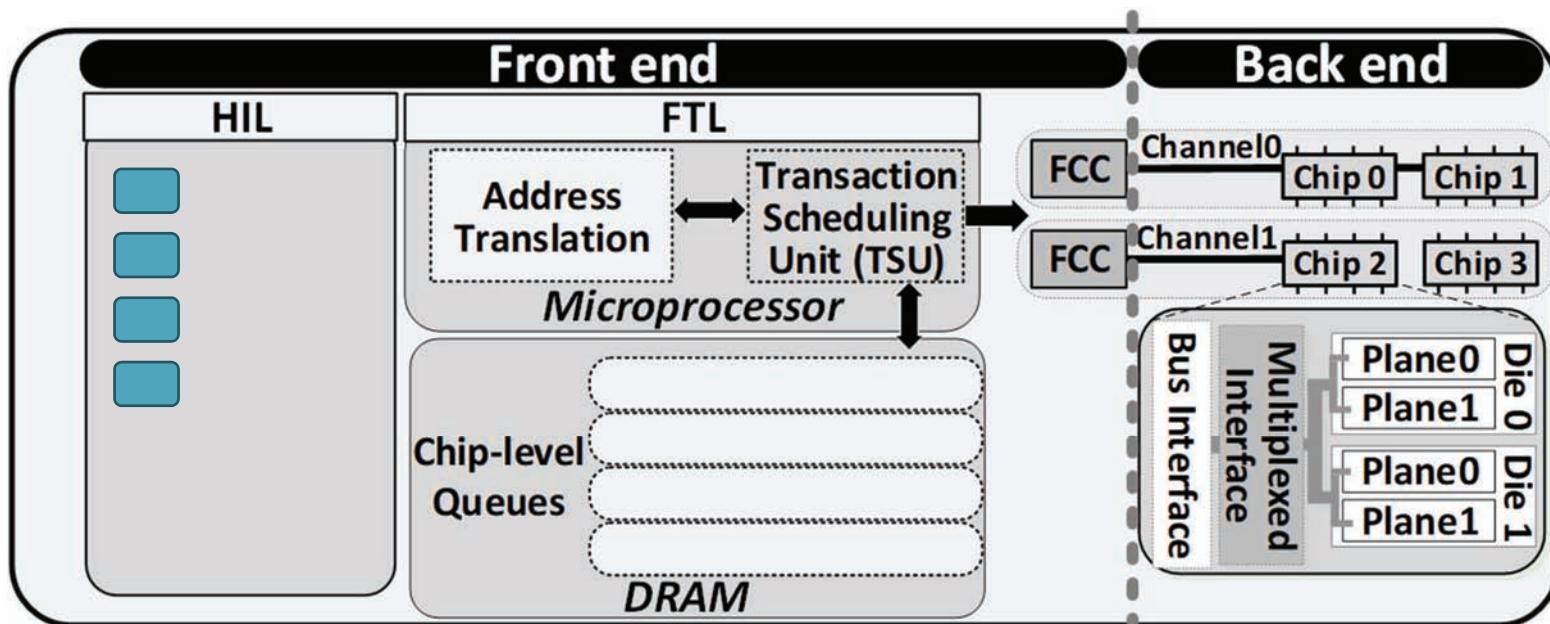
The average response time of a low-intensity flow substantially increases due to interference from a high-intensity flow

- Similar to memory scheduling for bandwidth-sensitive threads vs. latency-sensitive threads

# Source 2: Different Access Patterns

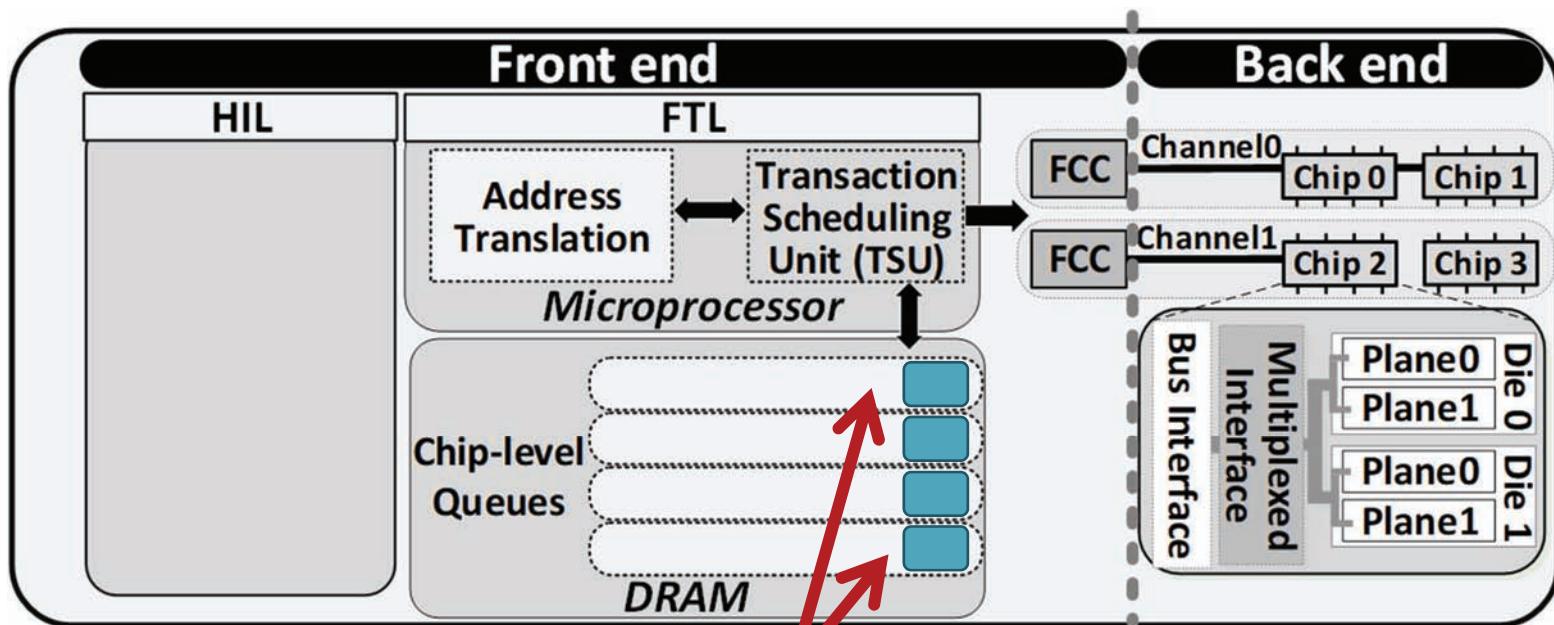
SAFARI

- Some flows take advantage of **chip-level parallelism** in back end



# Source 2: Different Access Patterns

- Some flows take advantage of **chip-level parallelism** in back end



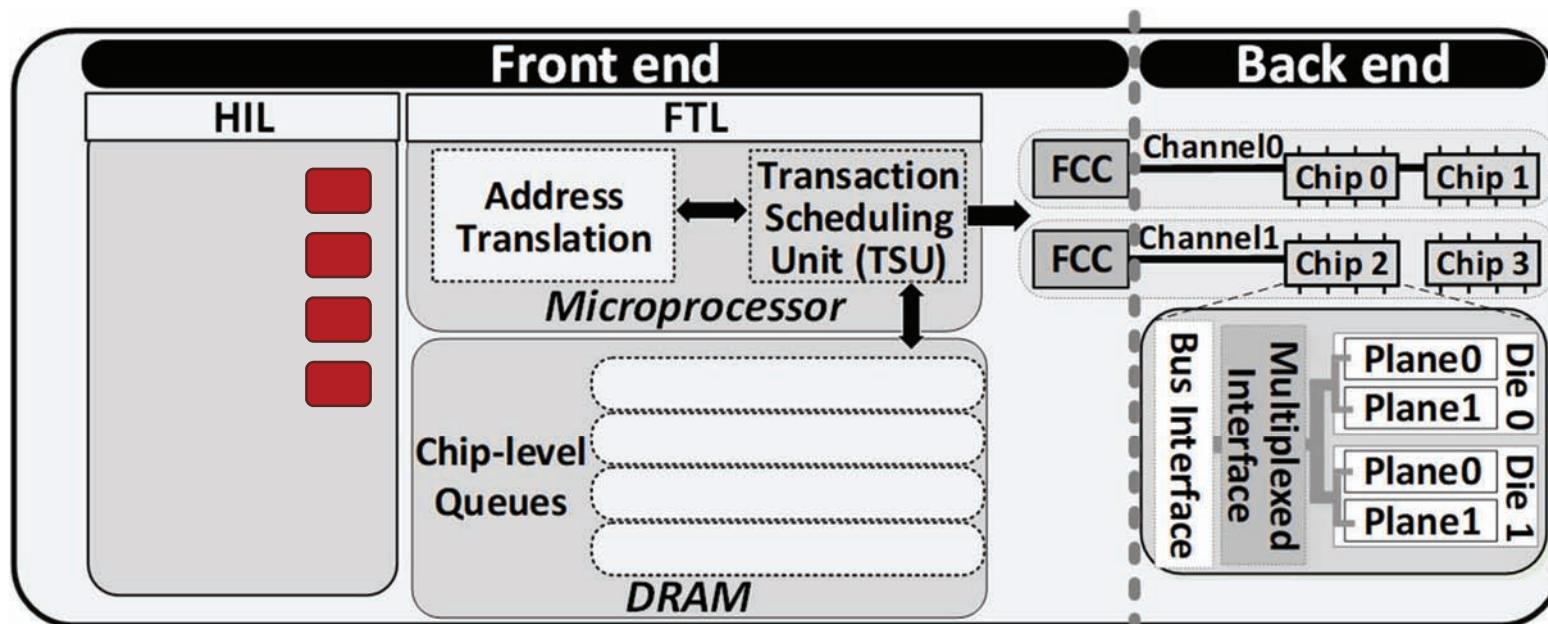
***Even distribution of transactions in chip-level queues***

- Leads to a **low queue wait time**

# Source 2: Different Request Access Patterns

SAFARI

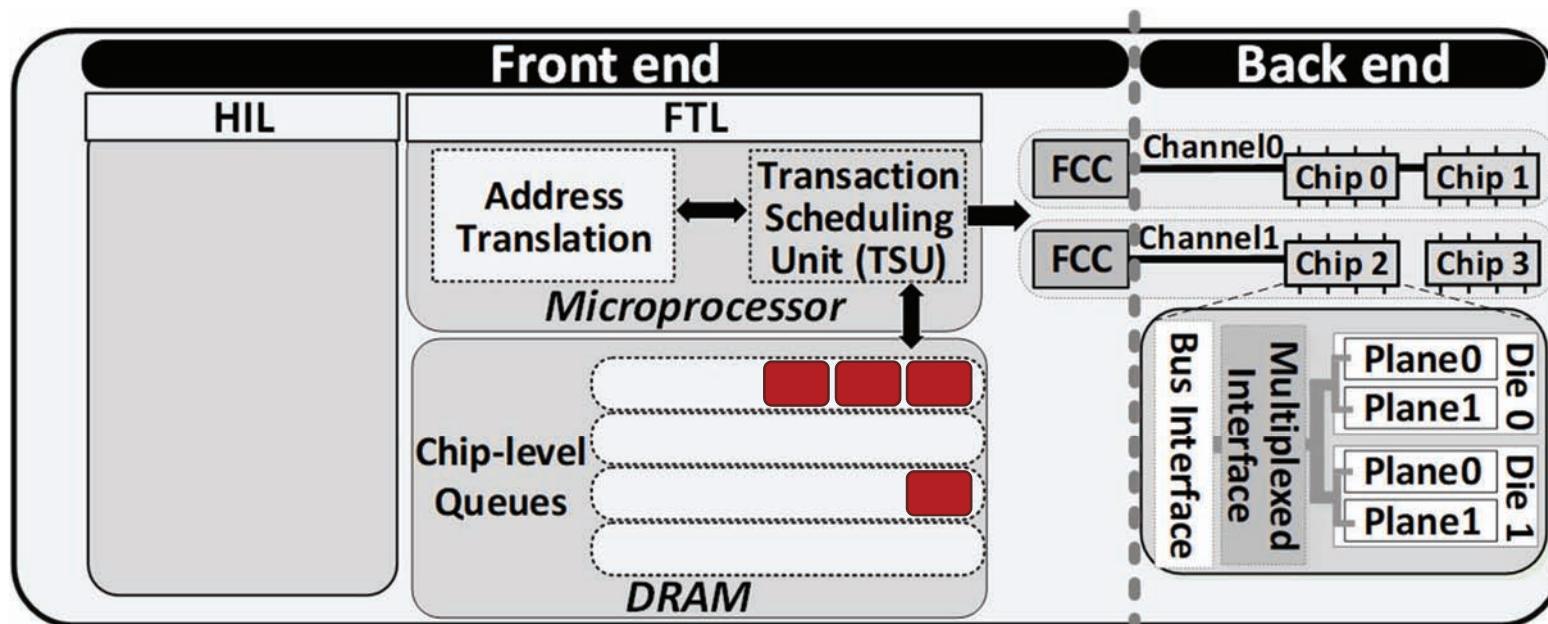
- Other flows have access patterns that **do not exploit parallelism**



# Source 2: Different Request Access Patterns

SAFARI

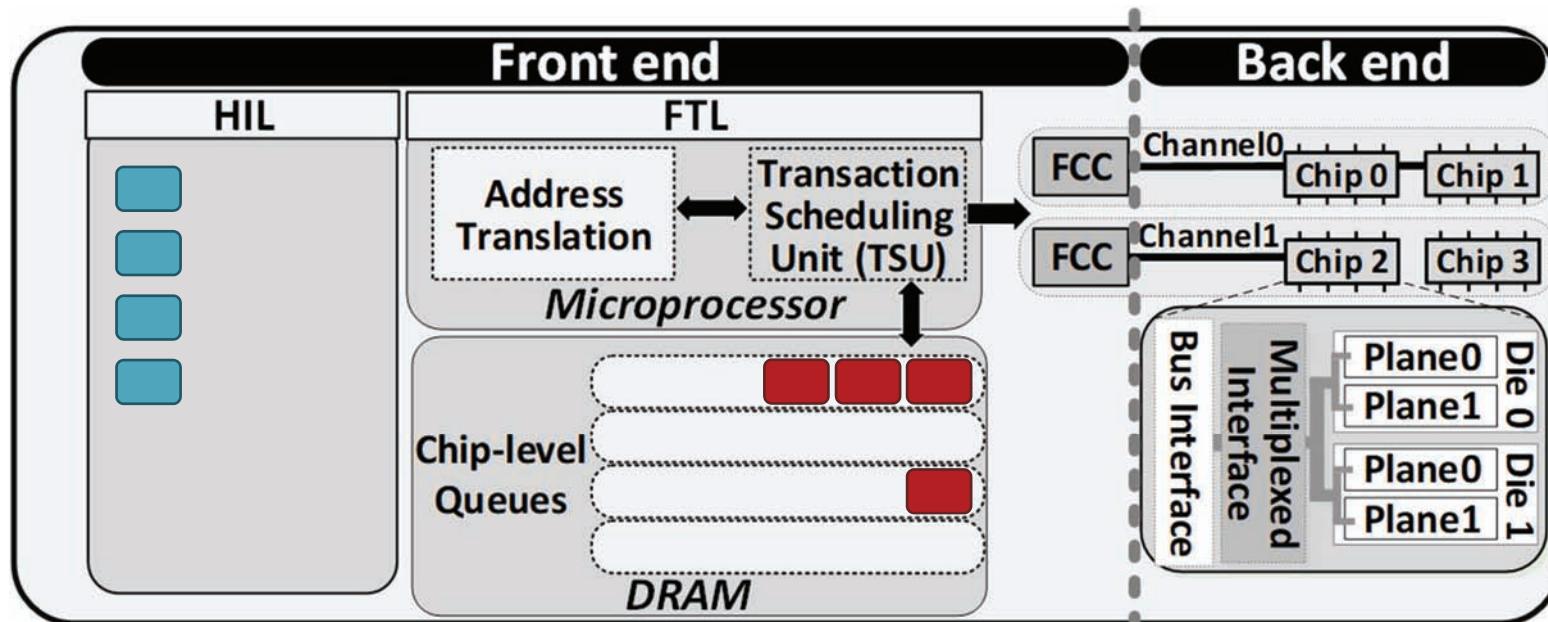
- Other flows have access patterns that **do not exploit parallelism**



# Source 2: Different Request Access Patterns

SAFARI

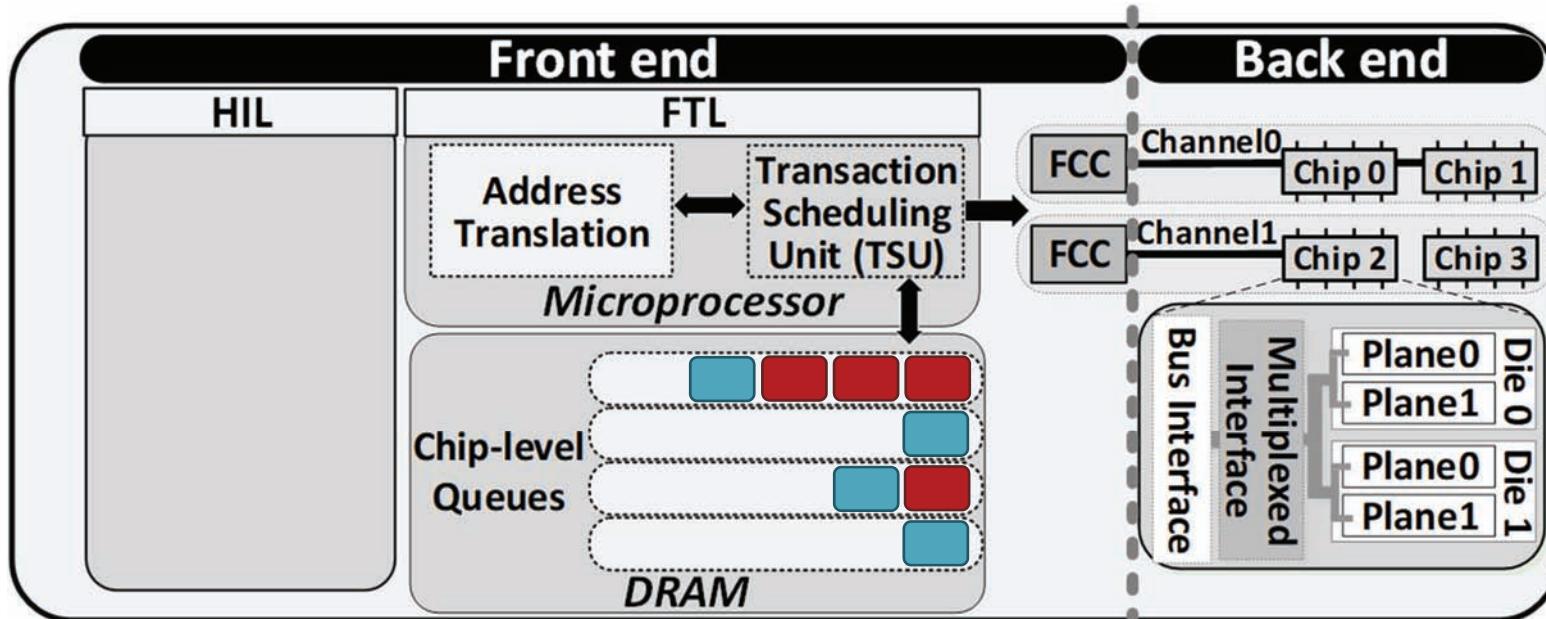
- Other flows have access patterns that **do not exploit parallelism**



# Source 2: Different Request Access Patterns

SAFARI

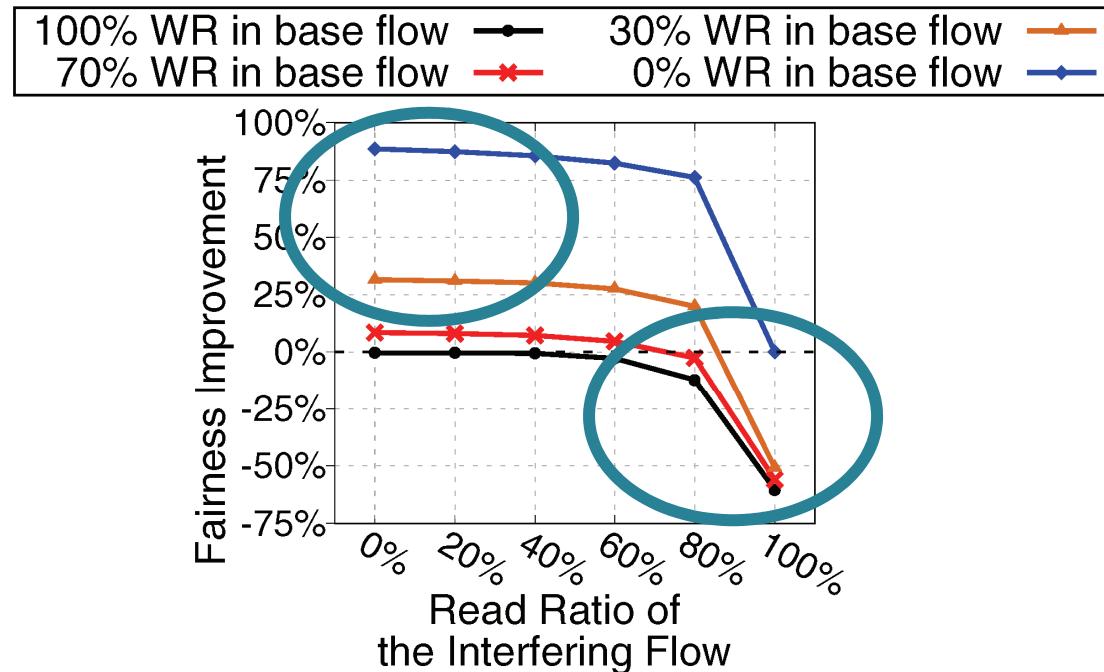
- Other flows have access patterns that **do not exploit parallelism**



Flows with **parallelism-friendly access patterns**  
are **susceptible to interference** from  
flows whose access patterns do not exploit parallelism

# Source 3: Different Read/Write Ratios

- State-of-the-art SSD I/O schedulers **prioritize reads over writes**
- Effect of read prioritization on fairness (vs. first-come, first-serve)



When flows have **different read/write ratios**,  
existing schedulers do not effectively provide fairness

# Source 4: Different Garbage Collection Demands

SAFARI

- NAND flash memory performs **writes out of place**
  - Erases can only happen on an entire **flash block** (hundreds of flash pages)
  - Pages marked invalid during write
- **Garbage collection (GC)**
  - Selects a block with mostly-invalid pages
  - Moves any remaining valid pages
  - Erases blocks with mostly-invalid pages
- **High-GC flow:** flows with a higher write intensity induce more garbage collection activities

The GC activities of a **high-GC** flow can unfairly block flash transactions of a **low-GC** flow

- Four major sources of unfairness in modern SSDs

1. I/O intensity
2. Request access patterns
3. Read/write ratio
4. Garbage collection demands

## OUR GOAL

Design an I/O request scheduler for SSDs that  
(1) provides fairness among flows  
by mitigating all four sources of interference, and  
(2) maximizes performance and throughput

## Background: Modern SSD Design

## Unfairness Across Multiple Applications in Modern SSDs

**FLIN:**

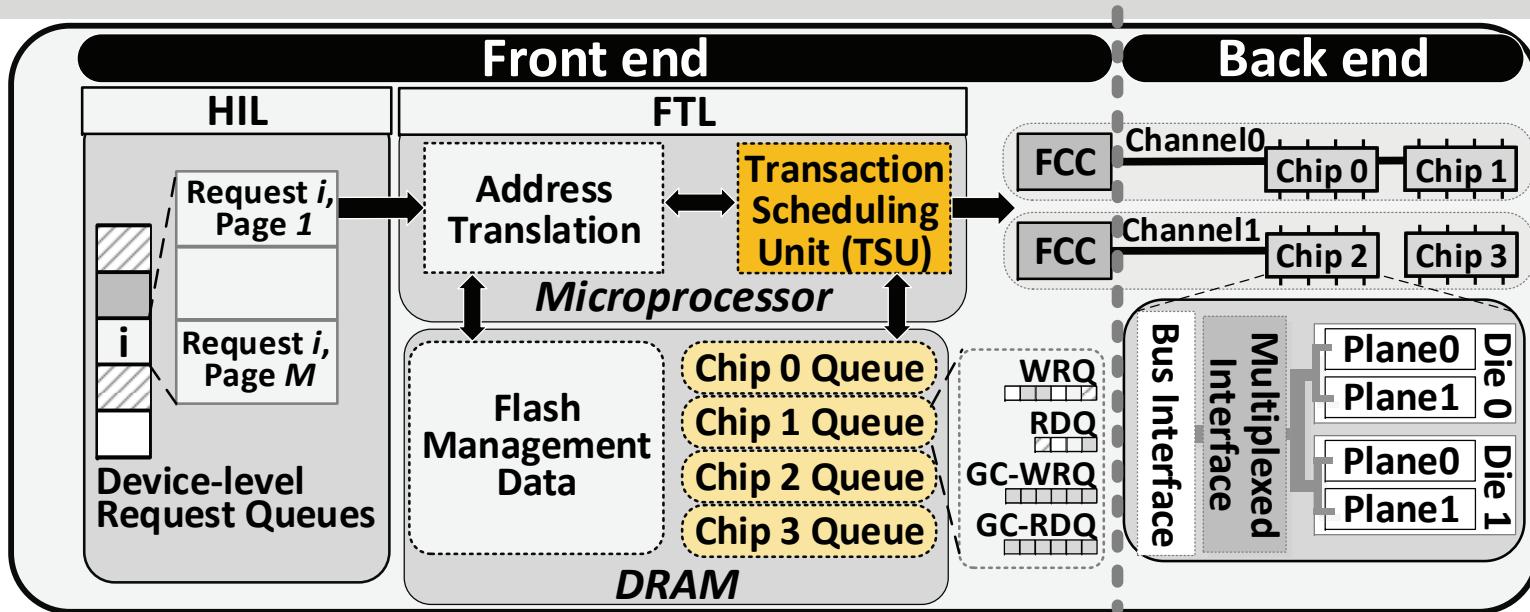
**Flash-Level INterference-aware SSD Scheduler**

Experimental Evaluation

Conclusion

# FLIN: Flash-Level INterference-aware Scheduler

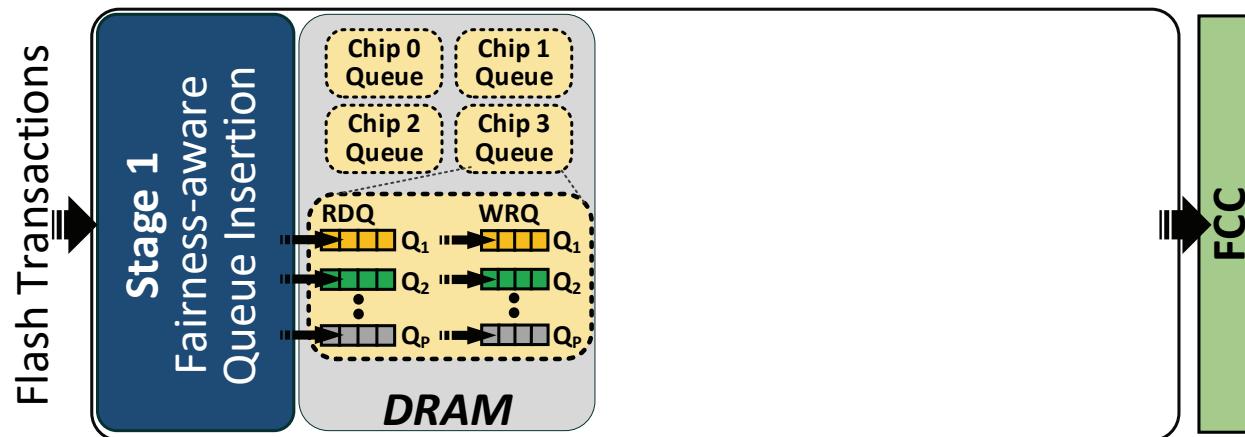
SAFARI



- FLIN is a three-stage I/O request scheduler
  - Replaces existing transaction scheduling unit
  - Takes in flash transactions, reorders them, sends them to flash channel
- Identical throughput to state-of-the-art schedulers
- Fully implemented in the SSD controller firmware
  - No hardware modifications
  - Requires < 0.06% of the DRAM available within the SSD

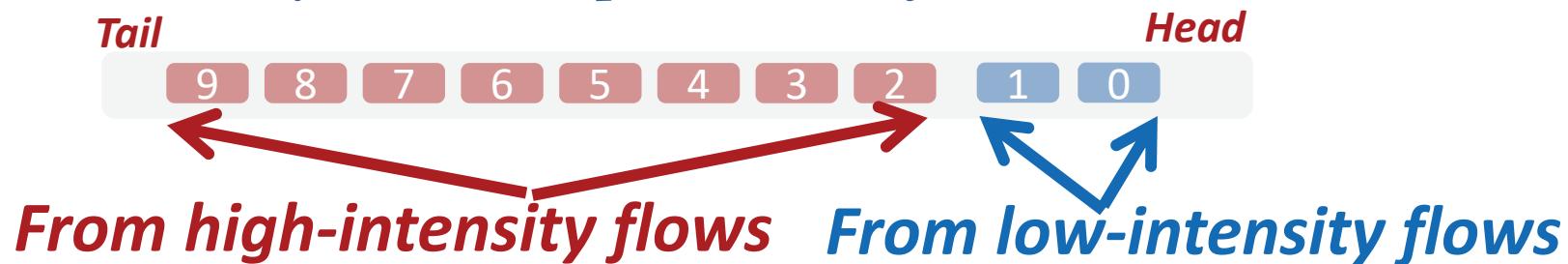
# Three Stages of FLIN

SAFARI



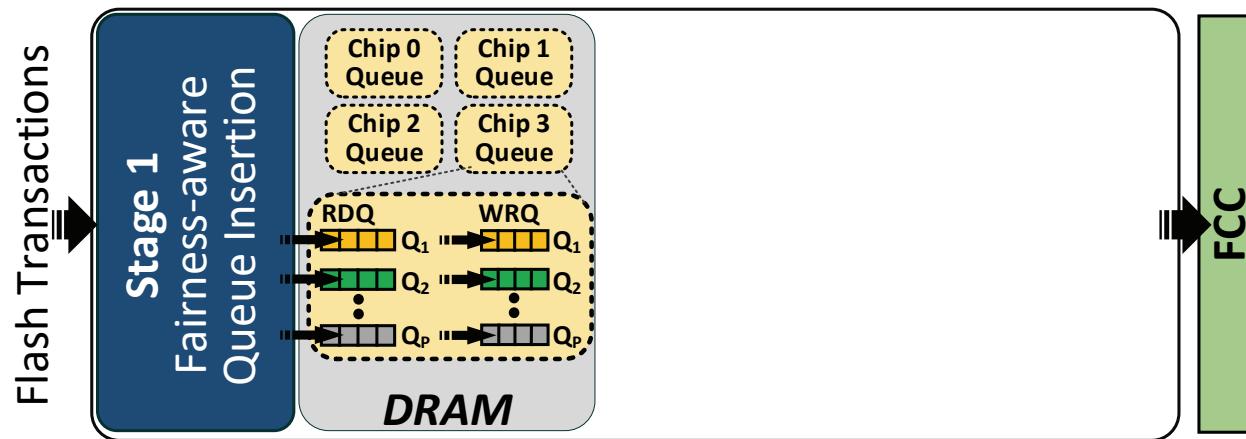
## ■ Stage 1: Fairness-aware Queue Insertion

*relieves I/O intensity and access pattern interference*



# Three Stages of FLIN

SAFARI

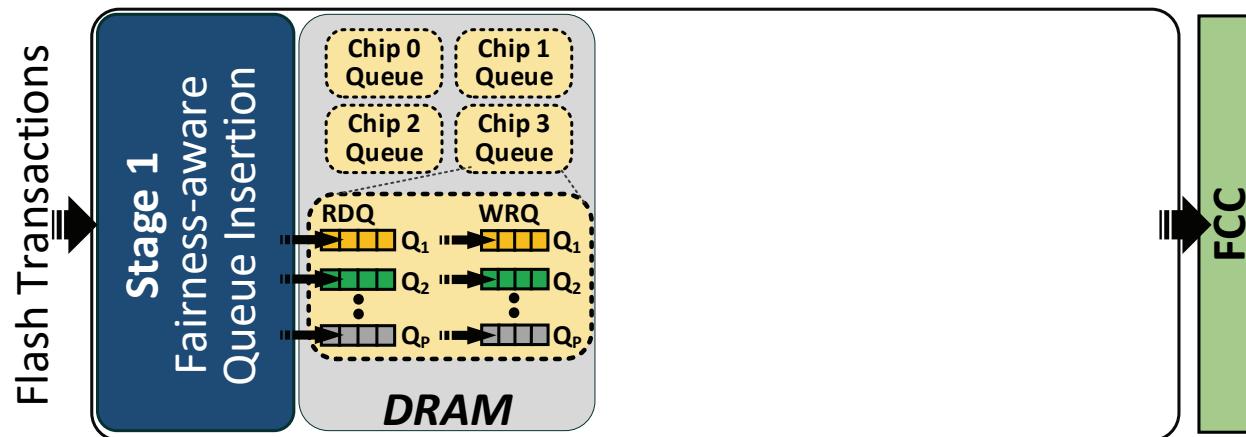


- Stage 1: Fairness-aware Queue Insertion  
*relieves I/O intensity and access pattern interference*



# Three Stages of FLIN

SAFARI

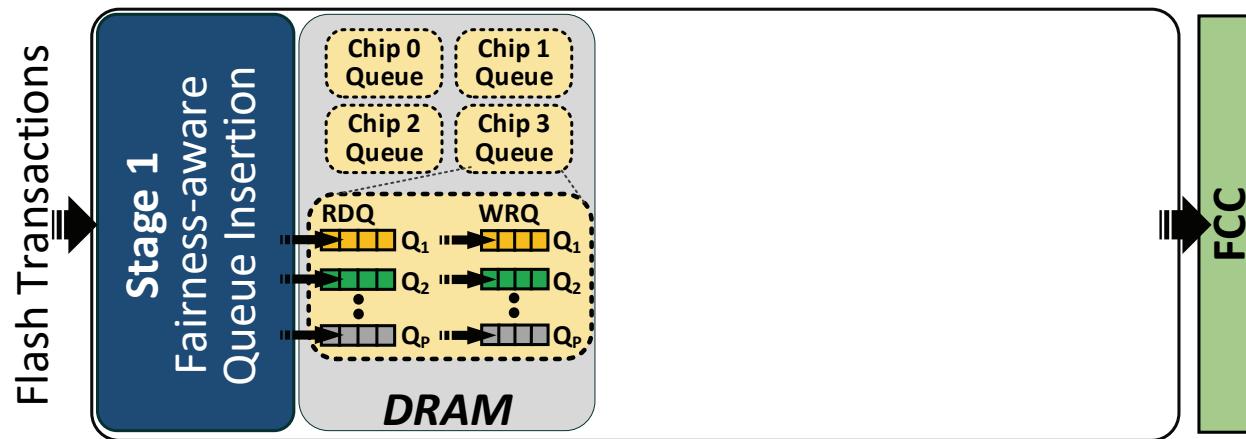


- Stage 1: Fairness-aware Queue Insertion  
*relieves I/O intensity and access pattern interference*



# Three Stages of FLIN

SAFARI

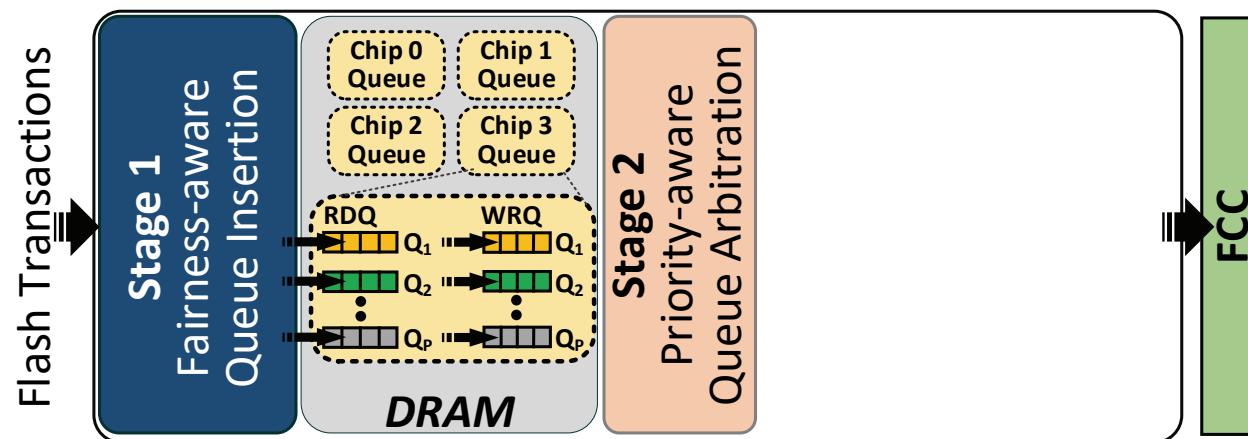


- Stage 1: Fairness-aware Queue Insertion  
*relieves I/O intensity and access pattern interference*



# Three Stages of FLIN

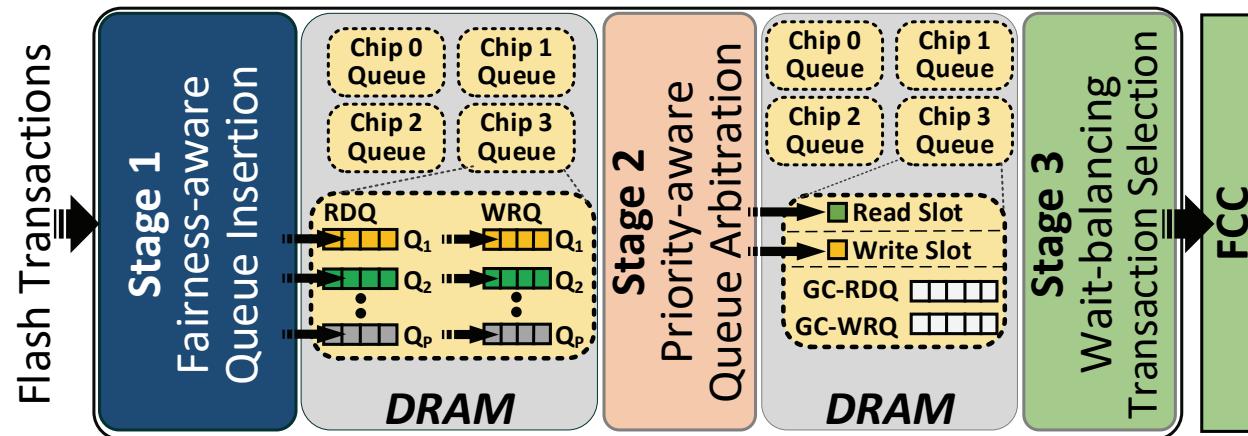
SAFARI



- Stage 1: Fairness-aware Queue Insertion  
*relieves I/O intensity and access pattern interference*
- Stage 2: Priority-aware Queue Arbitration  
*enforces priority levels that are assigned to each flow by the host*

# Three Stages of FLIN

SAFARI



- Stage 1: Fairness-aware Queue Insertion  
*relieves I/O intensity and access pattern interference*
- Stage 2: Priority-aware Queue Arbitration  
*enforces priority levels that are assigned to each flow by the host*
- Stage 3: Wait-balancing Transaction Selection  
*relieves read/write ratio and garbage collection demand interference*

## Background: Modern SSD Design

## Unfairness Across Multiple Applications in Modern SSDs

FLIN:

Flash-Level INterference-aware SSD Scheduler

## Experimental Evaluation

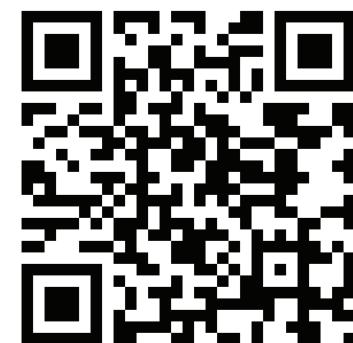
Conclusion

- **Detailed SSD Simulator: MQSim [FAST 2018]**

- Protocol: NVMe 1.2 over PCIe
- User capacity: 480GB
- Organization: 8 channels, 2 planes per die, 4096 blocks per plane, 256 pages per block, 8kB page size

*Download the Simulator and FAST 2018 Paper at*

**<http://github.com/CMU-SAFARI/MQSim>**



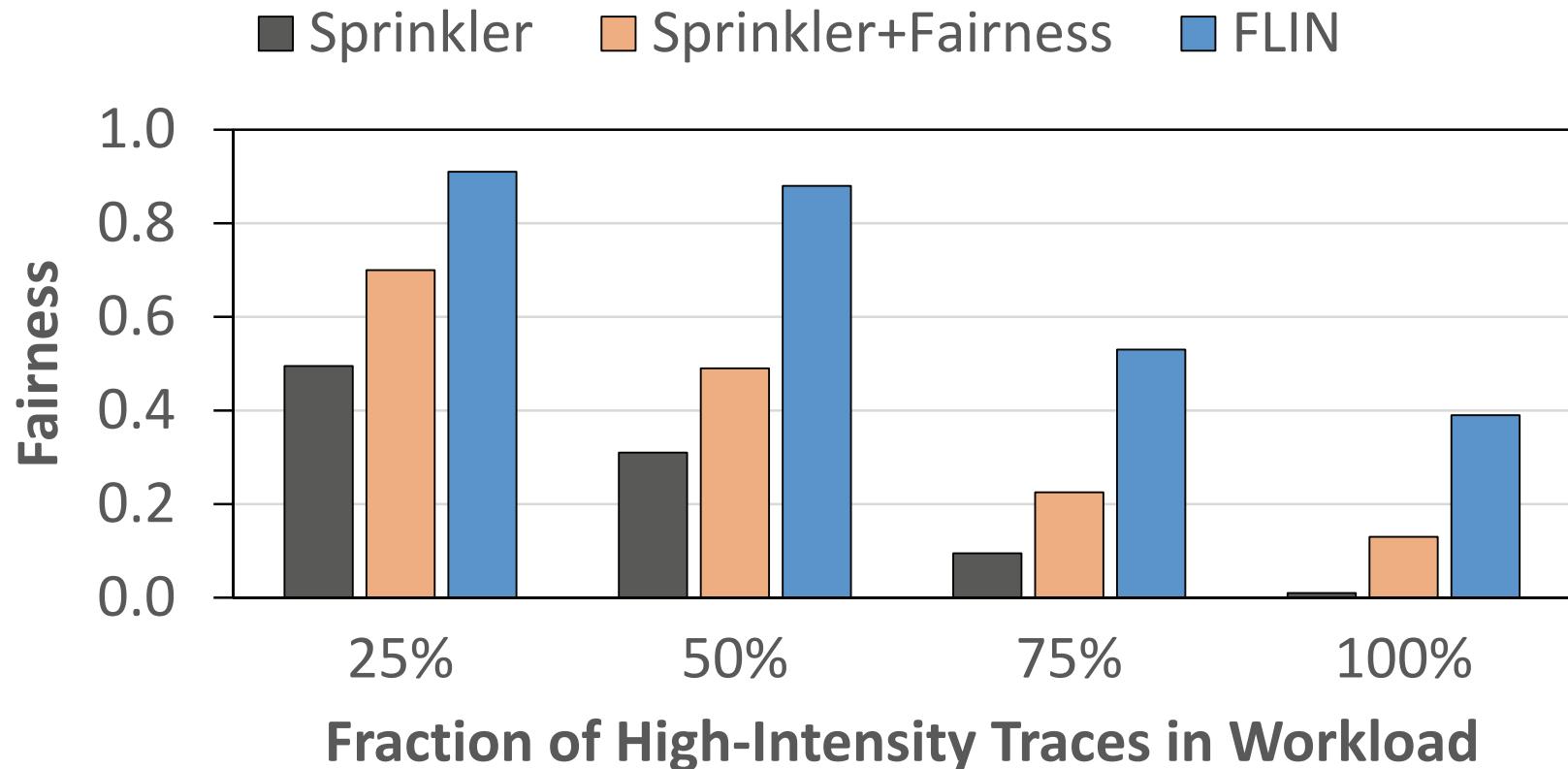
- **40 workloads containing four randomly-selected storage traces**

- Each storage trace is collected from real enterprise/datacenter applications: UMass, Microsoft production/enterprise
- Each application classified as low-interference or high-interference

- **Sprinkler** [Jung+ HPCA 2014]  
a state-of-the-art device-level high-performance scheduler
- **Sprinkler+Fairness** [Jung+ HPCA 2014, Jun+ NVMSA 2015]  
we add a state-of-the-art fairness mechanism to Sprinkler  
that was previously proposed for OS-level I/O scheduling
  - Does not have direct information about the internal resources and mechanisms of the SSD
  - Does not mitigate all four sources of interference

# FLIN Improves Fairness Over the Baselines

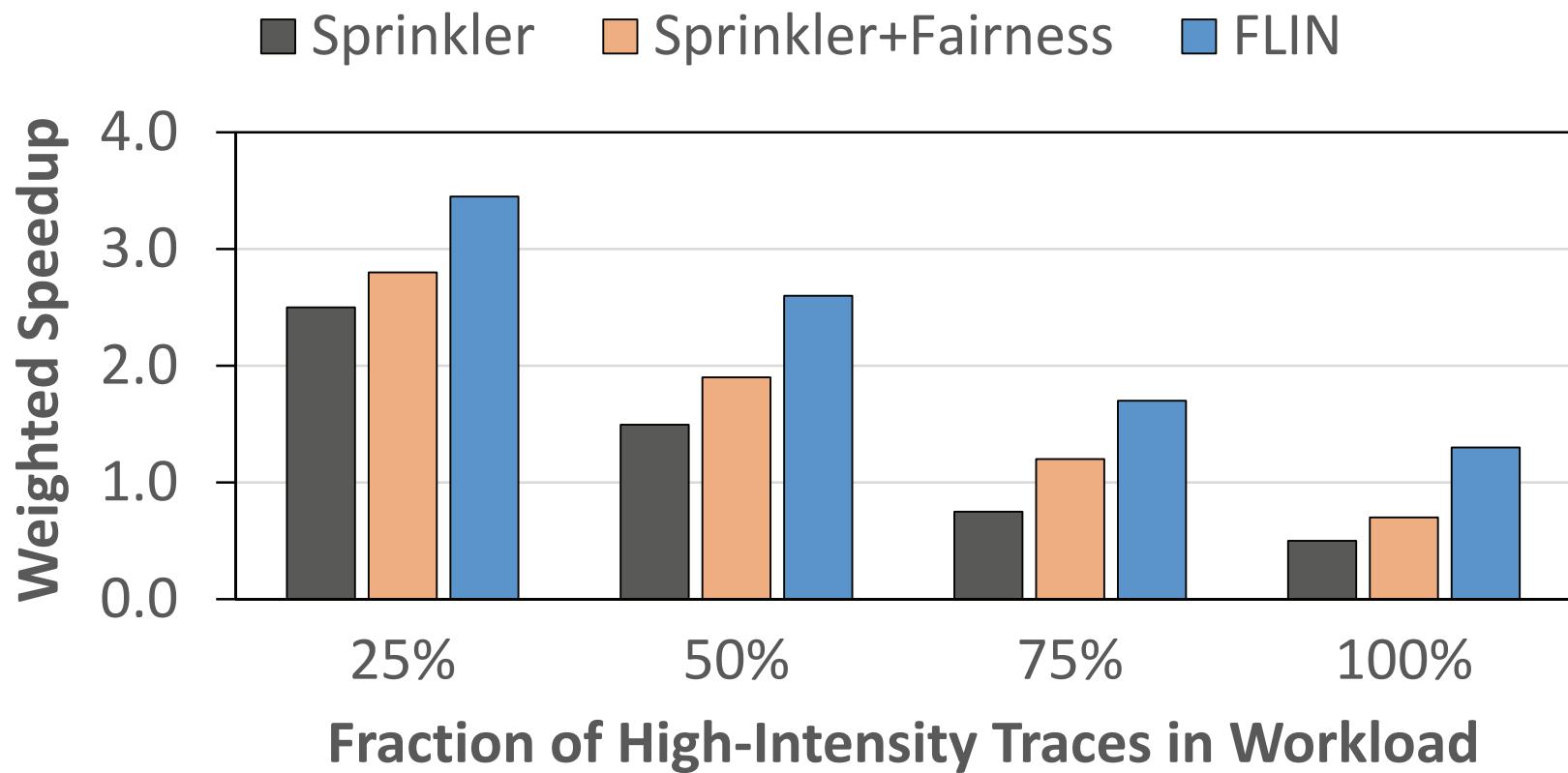
SAFARI



FLIN improves fairness by an average of 70%,  
by mitigating *all* four major sources of interference

# FLIN Improves Performance Over the Baselines

SAFARI



FLIN improves performance by an average of 47%,  
by making use of idle resources in the SSD and  
improving the performance of low-interference flows

- Fairness and weighted speedup for each workload
  - FLIN improves fairness and performance for *all* workloads
- Maximum slowdown
  - Sprinkler/Sprinkler+Fairness: several applications with maximum slowdown over 500x
  - FLIN: no flow with a maximum slowdown over 80x
- Effect of each stage of FLIN on fairness and performance
- Sensitivity study to FLIN and SSD parameters
- Effect of write caching

## Background: Modern SSD Design

## Unfairness Across Multiple Applications in Modern SSDs

FLIN:

Flash-Level INterference-aware SSD Scheduler

Experimental Evaluation

Conclusion

- Modern solid-state drives (SSDs) use new storage protocols (e.g., NVMe) that **eliminate the OS software stack**
  - Enables **high throughput**: millions of IOPS
  - OS software stack elimination **removes existing fairness mechanisms**
  - **Highly unfair slowdowns** on real state-of-the-art SSDs
- **FLIN**: a new I/O request scheduler for modern SSDs designed to **provide both fairness and high performance**
  - Mitigates all four sources of inter-application interference
    - » Different I/O intensities
    - » Different request access patterns
    - » Different read/write ratios
    - » Different garbage collection demands
  - Implemented fully in the SSD controller firmware, uses < 0.06% of DRAM
  - FLIN improves **fairness by 70%** and **performance by 47%** compared to a state-of-the-art I/O scheduler (Sprinkler+Fairness)



# FLIN:

## Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives

Saugata Ghose

Carnegie Mellon University

*Download our ISCA 2018 Paper at*

[http://ece.cmu.edu/~saugatag/papers/18isca\\_flin.pdf](http://ece.cmu.edu/~saugatag/papers/18isca_flin.pdf)



# References to Papers and Talks

# Our FMS Talks and Posters

SAFARI

- FMS 2019
  - Saugata Ghose, Modeling and Mitigating Early Retention Loss and Process Variation in 3D Flash
  - Saugata Ghose, Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives
- FMS 2018
  - Yixin Luo, HeatWatch: Exploiting 3D NAND Self-Recovery and Temperature Effects
  - Saugata Ghose, Enabling Realistic Studies of Modern Multi-Queue SSD Devices
- FMS 2017
  - Aya Fukami, Improving Chip-Off Forensic Analysis for NAND Flash
  - Saugata Ghose, Vulnerabilities in MLC NAND Flash Memory Programming
- FMS 2016
  - Onur Mutlu, [ThyNVM: Software-Transparent Crash Consistency for Persistent Memory](#)
  - Onur Mutlu, [Large-Scale Study of In-the-Field Flash Failures](#)
  - Yixin Luo, [Practical Threshold Voltage Distribution Modeling](#)
  - Saugata Ghose, [Write-hotness Aware Retention Management](#)
- FMS 2015
  - Onur Mutlu, [Read Disturb Errors in MLC NAND Flash Memory](#)
  - Yixin Luo, [Data Retention in MLC NAND Flash Memory](#)
- FMS 2014
  - Onur Mutlu, [Error Analysis and Management for MLC NAND Flash Memory](#)

- Summary of our work in NAND flash memory
  - Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu, Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid-State Drives, *Proceedings of the IEEE*, Sept. 2017.
- Overall flash error analysis
  - Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis, DATE 2012.
  - Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai, Error Analysis and Retention-Aware Error Management for NAND Flash Memory, ITJ 2013.
  - Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu, Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory, *IEEE JSAC*, Sept. 2016.

## ■ 3D NAND flash memory error analysis

- Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu, Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation, SIGMETRICS 2018.
- Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu, HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature-Awareness, HPCA 2018.

## ■ Multi-queue SSDs

- Arash Tavakkol, Juan Gomez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu, MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices, FAST 2018.
- Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan G. Luna and Onur Mutlu, FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives, ISCA 2018.

- Flash-based SSD prototyping and testing platform
  - Yu Cai, Erich F. Haratsh, Mark McCartney, Ken Mai, FPGA-based solid-state drive prototyping platform, FCCM 2011.
- Retention noise study and management
  - Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman Unsal, and Ken Mai, Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime, ICCD 2012.
  - Yu Cai, Yixin Luo, Erich F. Haratsch, Ken Mai, and Onur Mutlu, Data Retention in MLC NAND Flash Memory: Characterization, Optimization and Recovery, HPCA 2015.
  - Yixin Luo, Yu Cai, Saugata Ghose, Jongmoo Choi, and Onur Mutlu, WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management, MSST 2015.
  - Aya Fukami, Saugata Ghose, Yixin Luo, Yu Cai, and Onur Mutlu, Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices, *Digital Investigation*, Mar. 2017.

## ■ Program and erase noise study

- Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai, Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis and Modeling, DATE 2013.
- Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques, HPCA 2017.

## ■ Cell-to-cell interference characterization and tolerance

- Yu Cai, Onur Mutlu, Erich F. Haratsch, and Ken Mai, Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation, ICCD 2013.
- Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Osman Unsal, Adrian Cristal, and Ken Mai, Neighbor-Cell Assisted Error Correction for MLC NAND Flash Memories, SIGMETRICS 2014.

## ■ Read disturb noise study

- Yu Cai, Yixin Luo, Saugata Ghose, Erich F. Haratsch, Ken Mai, and Onur Mutlu, Read Disturb Errors in MLC NAND Flash Memory: Characterization and Mitigation, DSN 2015.

## ■ Flash errors in the field

- Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu, A Large-Scale Study of Flash Memory Errors in the Field, SIGMETRICS 2015.

## ■ Persistent memory

- Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu, and Onur Mutlu, ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems, MICRO 2015.

- All are available at
  - <https://safari.ethz.ch/publications/>
  - <https://www.ece.cmu.edu/~safari/talks.html>
- And, many other previous works on
  - Challenges and opportunities in memory
  - NAND flash memory errors and management
  - Phase change memory as DRAM replacement
  - STT-MRAM as DRAM replacement
  - Taking advantage of persistence in memory
  - Hybrid DRAM + NVM systems
  - NVM design and architecture

# Backup Slides

# Enabling Higher SSD Performance and Capacity

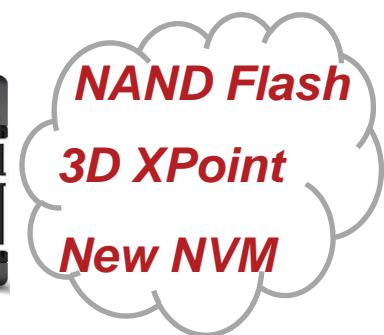
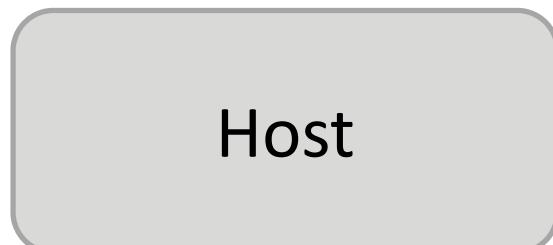
SAFARI

- Solid-state drives (SSDs) are widely used in today's computer systems

- Data centers
- Enterprise servers
- Consumer devices



- I/O demand of both enterprise and consumer applications continues to grow
- SSDs are rapidly evolving to deliver improved performance



# Defining Slowdown and Fairness for I/O Flows

SAFARI

- $RT_{f_i}$ : response time of Flow  $f_i$

- $S_{f_i}$ : slowdown of Flow  $f_i$

$$S_{f_i} = \overline{RT_{f_i}^{shared}} / RT_{f_i}^{alone}$$

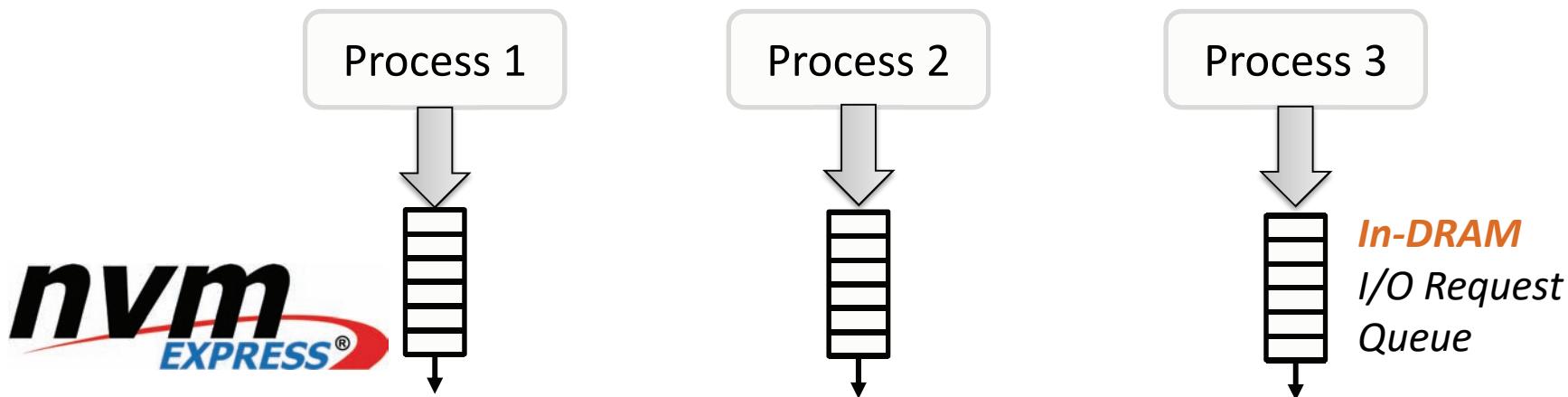
- $F$ : fairness of slowdowns across multiple flows

$$F = \frac{\min_i\{S_{f_i}\}}{\max_i\{S_{f_i}\}}$$

- $0 < F < 1$
  - Higher  $F$  means that system is *more* fair
- $WS$ : weighted speedup

$$WS = \sum_i \frac{RT_i^{alone}}{RT_i^{shared}}$$

- Modern SSDs use **high-performance** host–interface protocols (e.g., NVMe)
  - Take advantage of SSD throughput: enables *millions of IOPS* per device
  - Bypass OS intervention: **SSD must perform scheduling, ensure fairness**

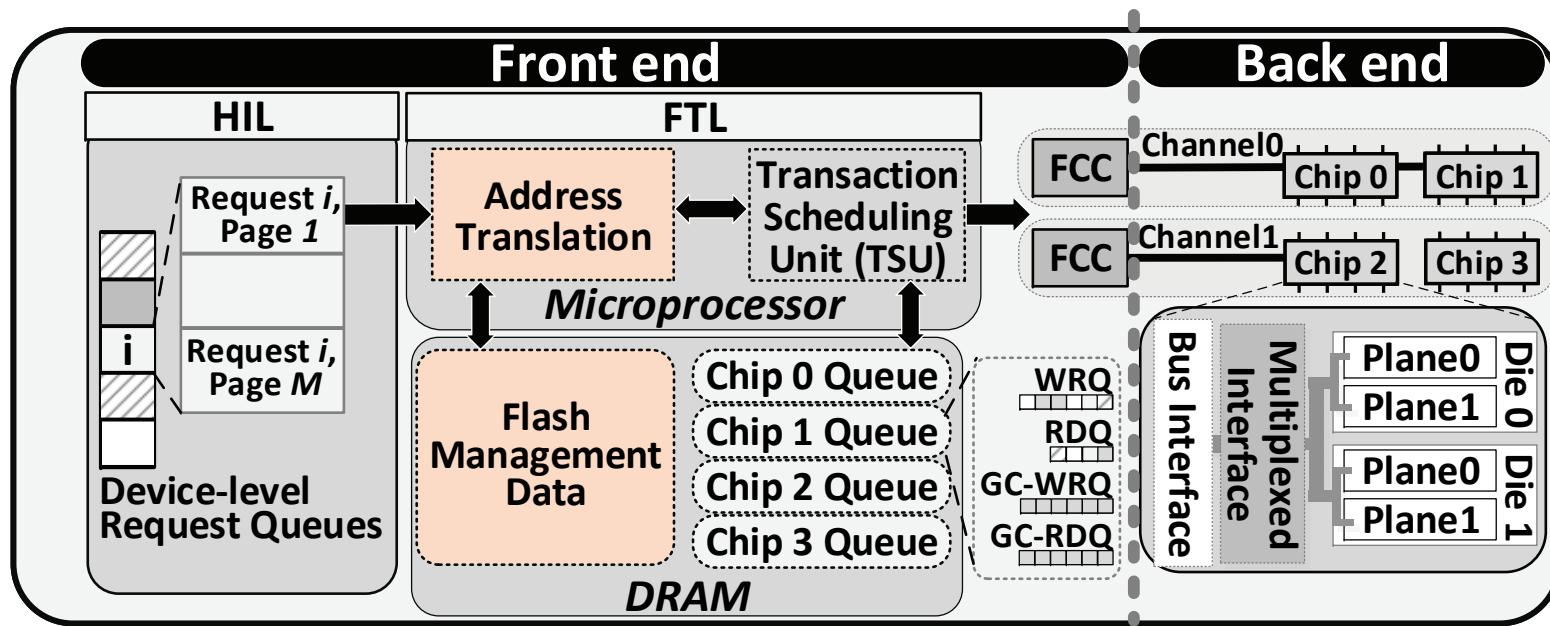


Fairness should be provided by the SSD itself.  
Do modern SSDs provide fairness?

# FTL: Managing the SSD's Resources

SAFARI

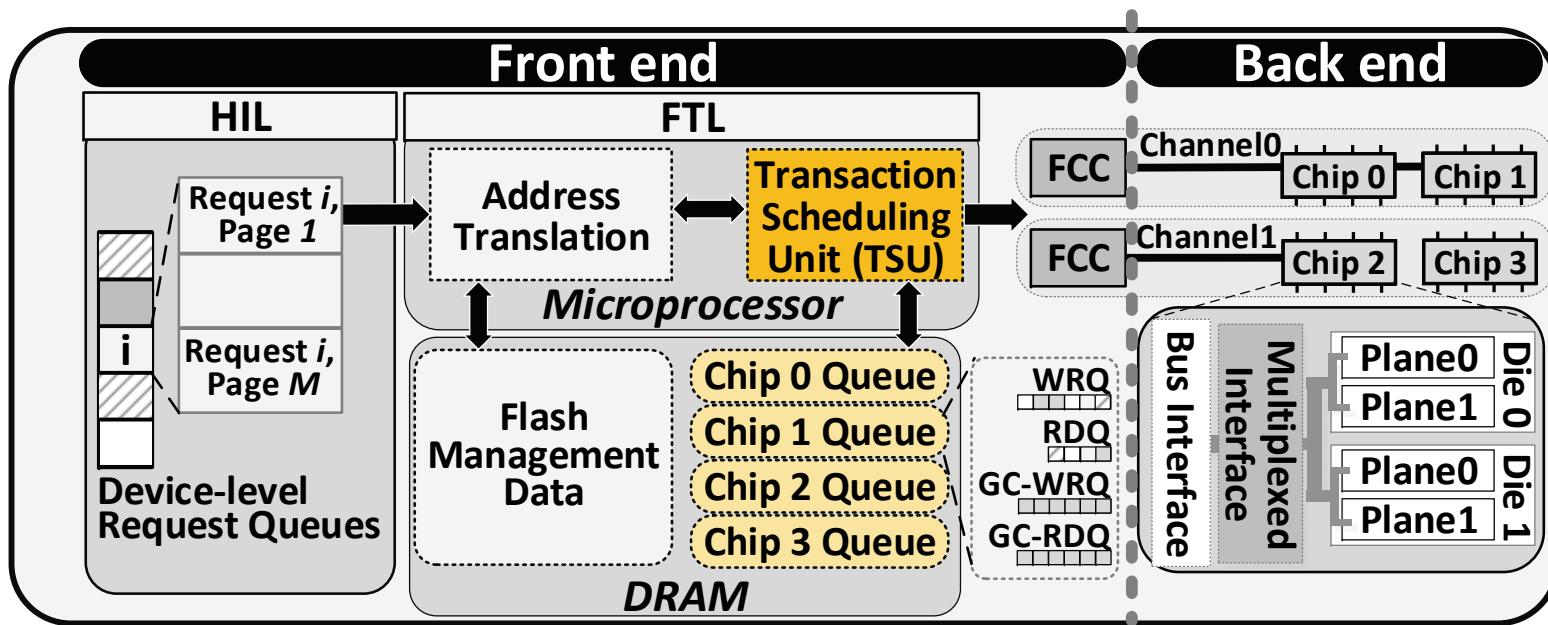
- Flash writes can take place only to pages that are erased
  - Perform **out-of-place updates** (i.e., write data to a different, free page), mark **old page** as invalid
  - **Update logical-to-physical mapping** (makes use of *cached mapping table*)
  - Some time later: **garbage collection** reclaims invalid physical pages off the critical path of latency



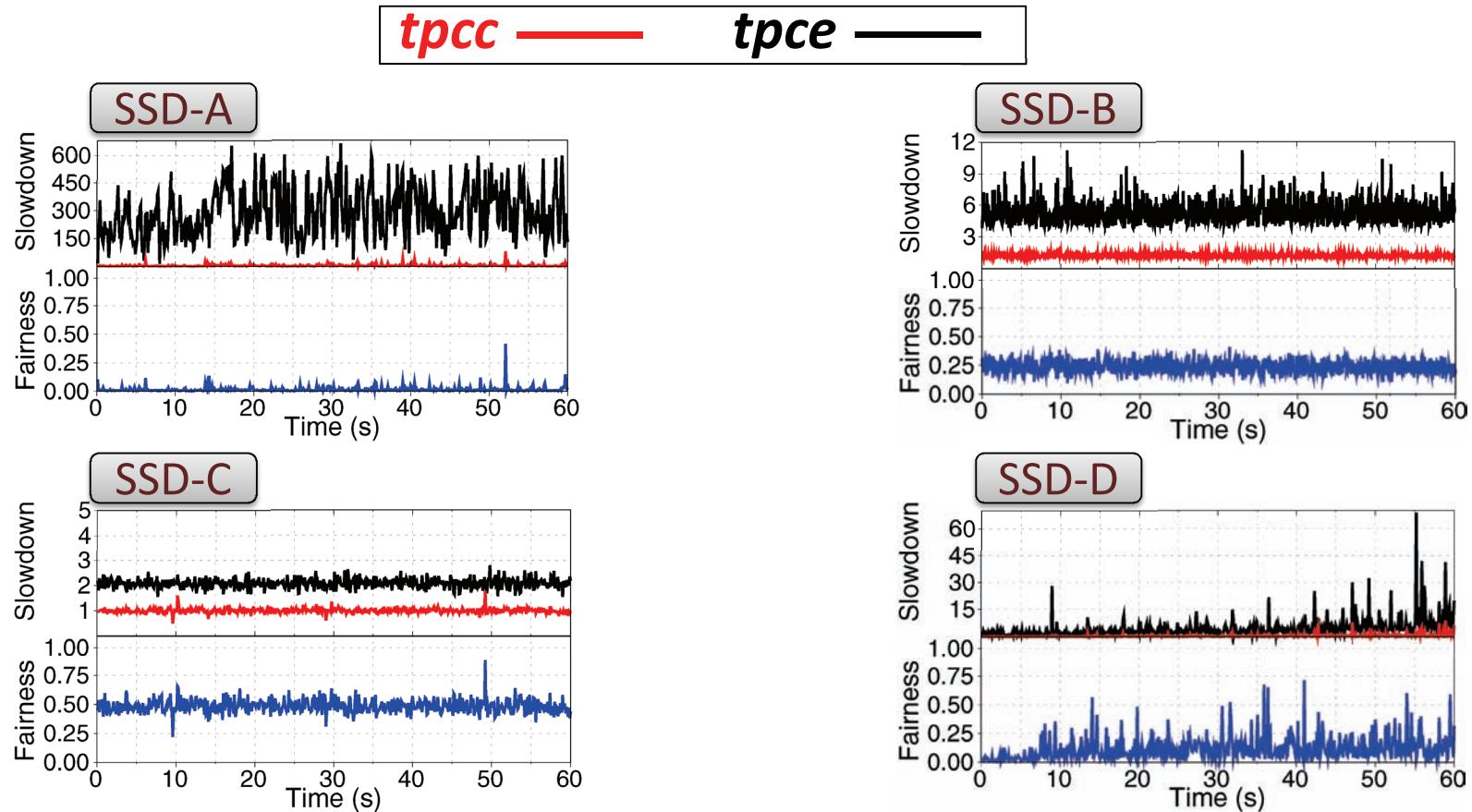
# FTL: Managing the SSD's Resources

SAFARI

- Flash writes can take place only to pages that are erased
  - Perform **out-of-place updates** (i.e., write data to a different, free page), mark **old page** as invalid
  - **Update logical-to-physical mapping** (makes use of *cached mapping table*)
  - Some time later: **garbage collection** reclaims invalid physical pages off the critical path of latency
- **Transaction Scheduling Unit:** resolves resource contention



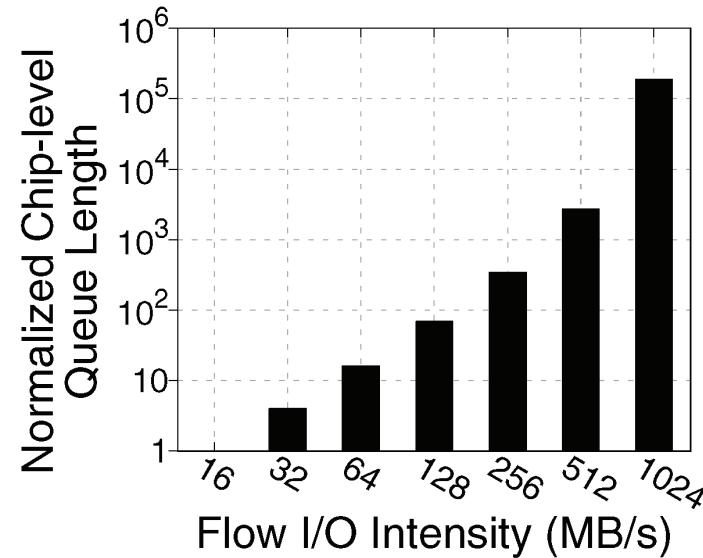
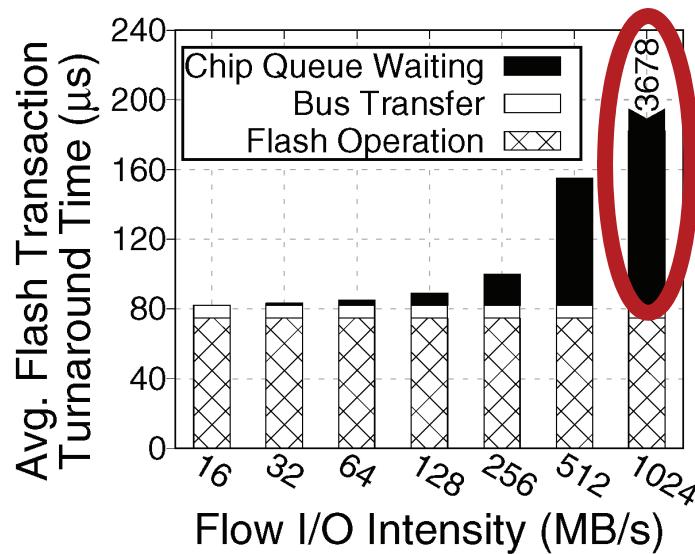
- The study experimental results on our four SSDs
  - An example of two datacenter workloads running concurrently



***tpce on average experiences 2x to 106x higher slowdown compared to tpcc***

# Reason 1: Difference in the I/O Intensities

- The I/O intensity of a flow affects the average queue wait time of flash transactions

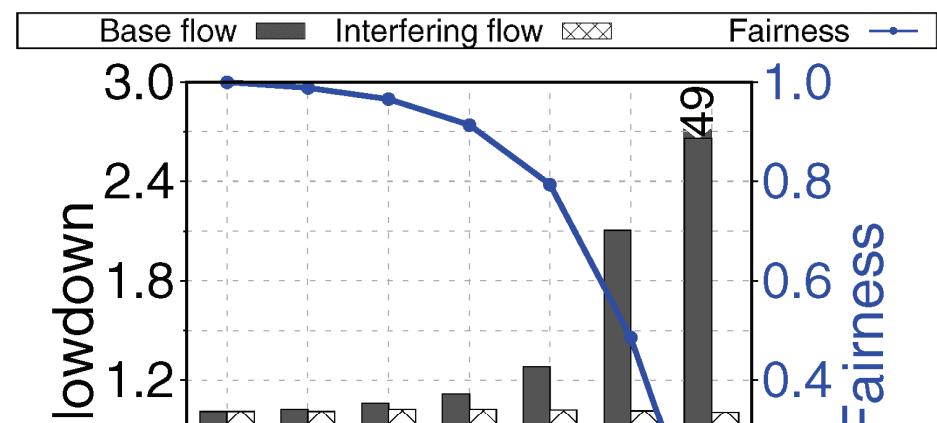


The queue wait time  
highly increases with I/O intensity

# Reason 1: Difference in the I/O Intensities

- An experiment to analyze the effect of concurrently executing two flows with **different I/O intensities** on fairness
  - Base flow: low intensity (16 MB/s) and **low** average chip-level queue length
  - Interfering flow: varying I/O intensities from **low** to very high

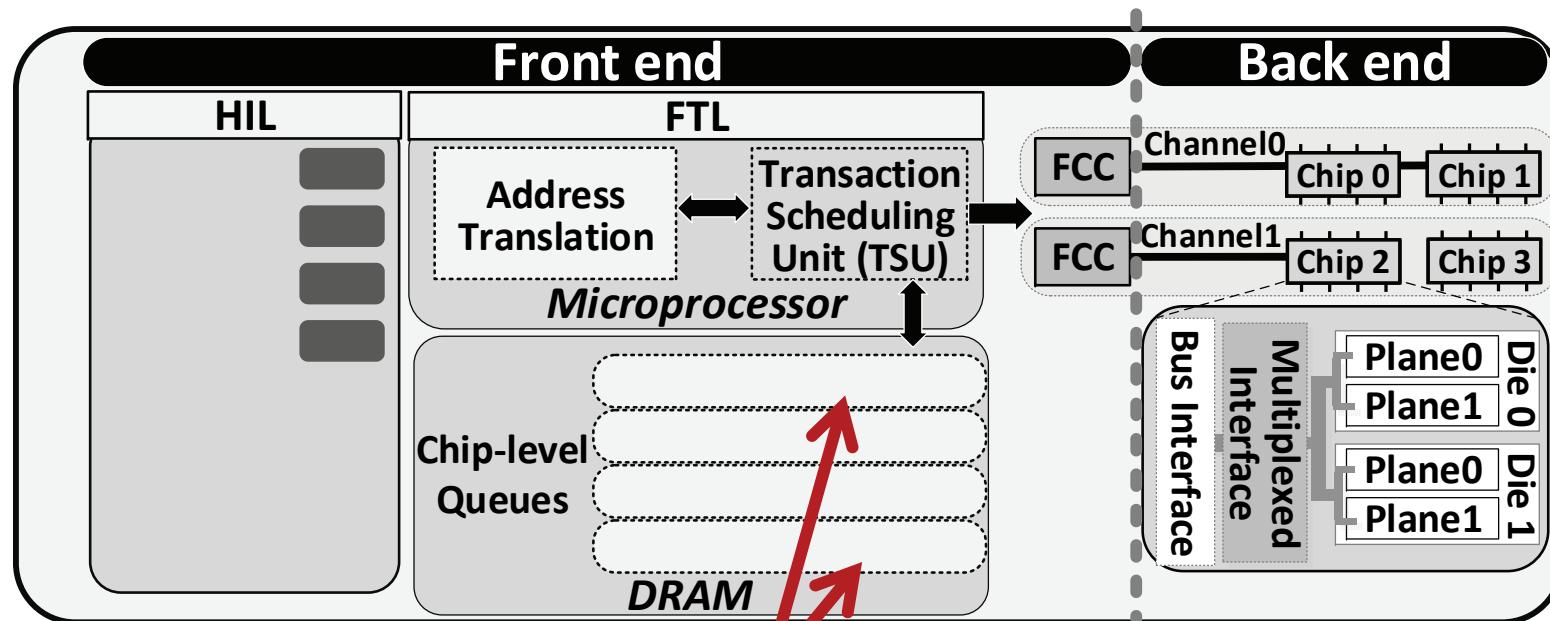
*Base flow experiences a  
drastic increase  
in the average length of  
the chip-level queues*



*The average response time of a low-intensity flow  
substantially increases due to  
interference from a high-intensity flow*

# Reason 2: Difference in the Access Pattern

- The access pattern of a flow determines how its transactions are distributed across the chip-level queues

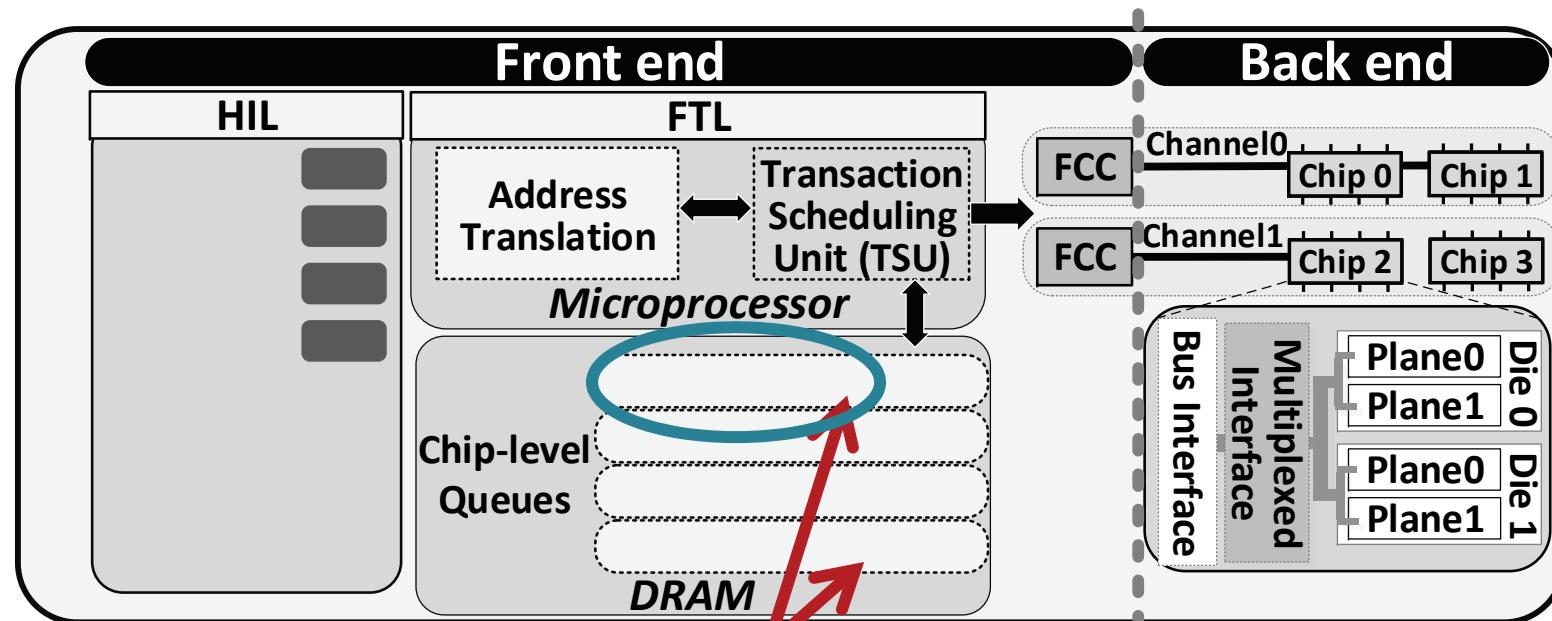


*Even distribution of transactions in chip-level queues*

- The running flow benefits from **parallelism** in the back end
- Leads to a **low** transaction queue wait time

# Reason 2: Difference in the Access Pattern

- The access pattern of a flow determines how its transactions are distributed across the chip-level queues

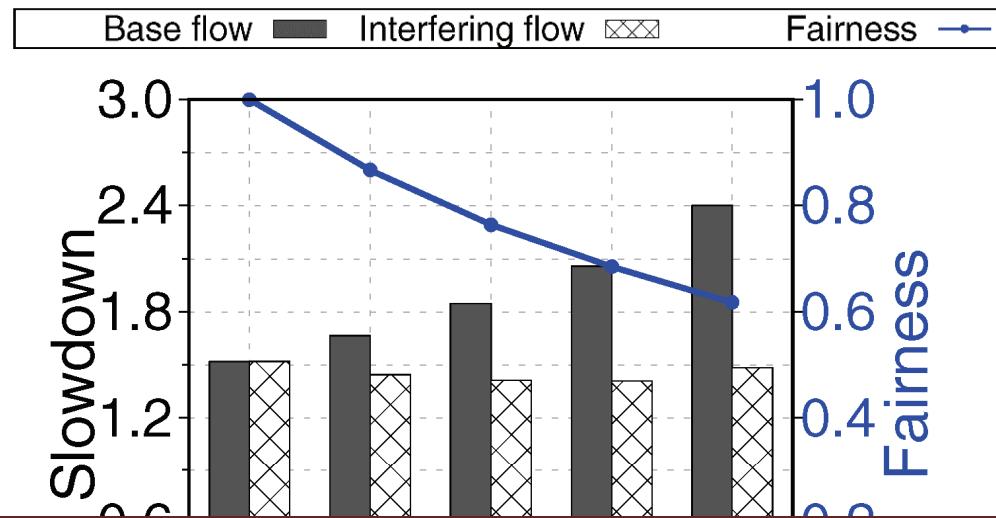


*Uneven distribution of flash transactions*

- Higher transaction wait time in the chip-level queues

# Reason 2: Difference in the Access Pattern

- An experiment to analyze the interference between concurrent flows with **different access patterns**
  - Base flow: **streaming** access pattern (parallelism friendly)
  - Interfering flow: **mixed** streaming and random access pattern

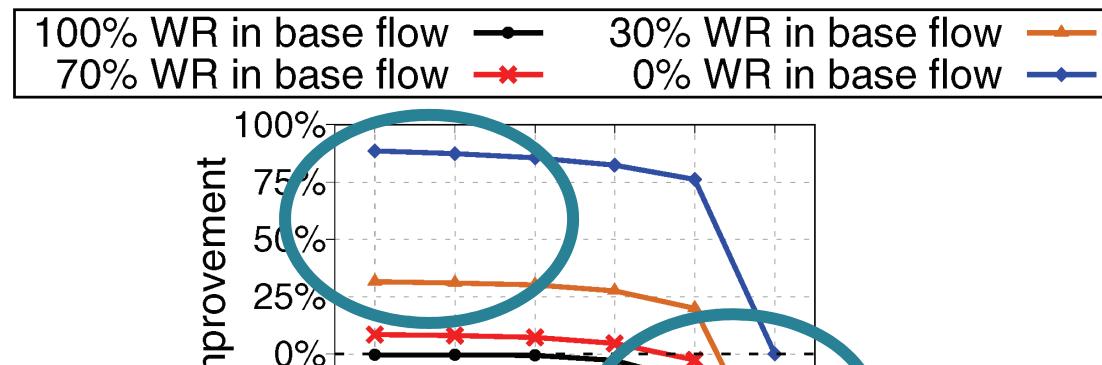


Flows with parallelism-friendly access patterns are susceptible to interference from flows with access patterns that do not exploit parallelism

# Reason 3: Difference in the Read/Write Ratios

SAFARI

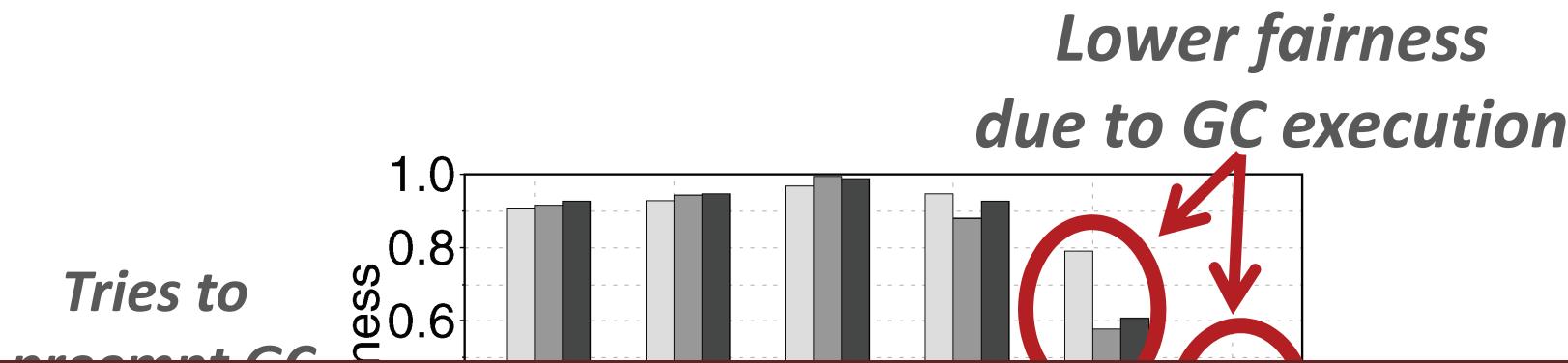
- State-of-the-art SSD I/O schedulers tend to prioritize reads over writes
  - Reads are 10-40x faster than writes
  - Reads are more likely to fall on the critical path of program execution
- The effect of read prioritization on fairness
  - Compare a first-come first-serve scheduler with a read-prioritized scheduler



Existing scheduling policies are not effective  
at providing fairness, when  
concurrent flows have different read/write ratios

# Reason 4: Difference in the GC Demands

- **Garbage collection** may block user I/O requests
  - Primarily depends on the **write intensity** of the workload
- An experiment with two 100%-write flows with different intensities
  - Base flow: low intensity and **moderate** GC demand
  - Interfering flow: different write intensities from **low-GC** to **high-GC**



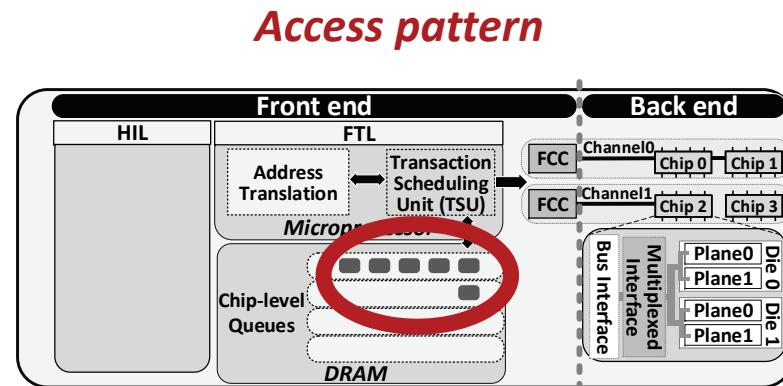
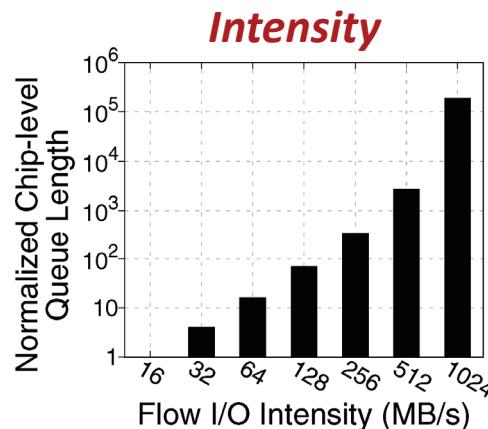
The GC activities of a **high-GC** flow can unfairly block flash transactions of a **low-GC** flow

Intensity of the interfering Flow (MB/S)

# Stage 1: Fairness-Aware Queue Insertion

SAFARI

- Relieves the interference that occurs due to the **intensity** and **access pattern** of concurrently-running flows



- In **concurrent** execution of two flows
  - Flash transactions of one flow experience a higher increase in the chip-level queue wait time
- Stage 1 performs **reordering** of transactions within the chip-level queues to reduce the queue wait

# Stage 1: Fairness-Aware Queue Insertion

SAFARI



If source of the new transaction is **low-intensity**



2a. Estimate slowdown of each transaction and reorder transactions to improve fairness in **low-intensity part**



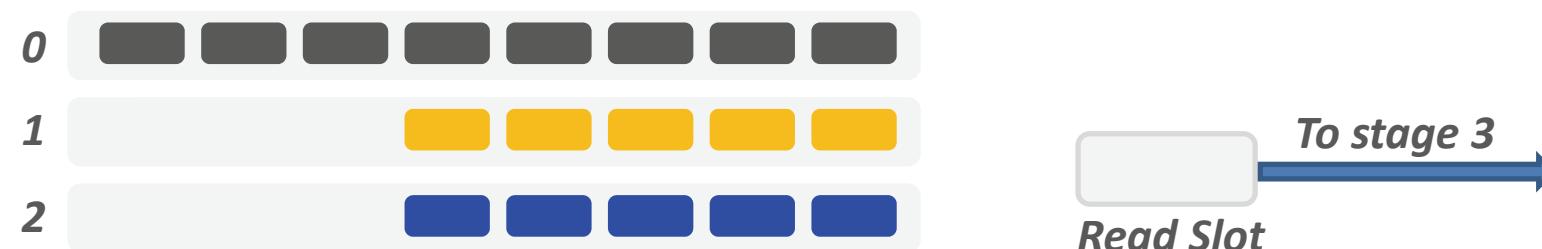
2b. Estimate slowdown of each transaction and reorder transactions to improve fairness in **high-intensity part**



# Stage 2: Priority-Aware Queue Arbitration

SAFARI

- Many host–interface protocols, such as NVMe, allow the host to assign different **priority levels** to each flow
- FLIN maintains a read and a write queue for **each priority** level at Stage 1
  - Totally  $2 \times P$  read and write queues in DRAM for  $P$  priority classes
- **Stage 2**
  - Selects one ready read/write transaction from the transactions at the head of the  $P$  read/write queues and moves it to Stage 3
  - It uses a **weighted round-robin** policy
- **An example**



# Stage 3: Wait-Balancing Transaction Selection

SAFARI

- Minimizes interference resulted from the **read/write ratios** and **garbage collection demands** of concurrently-running flows
- Attempts to **distribute** stall times evenly across read and write transactions
- Stage 3 considers **proportional wait** time of the transactions

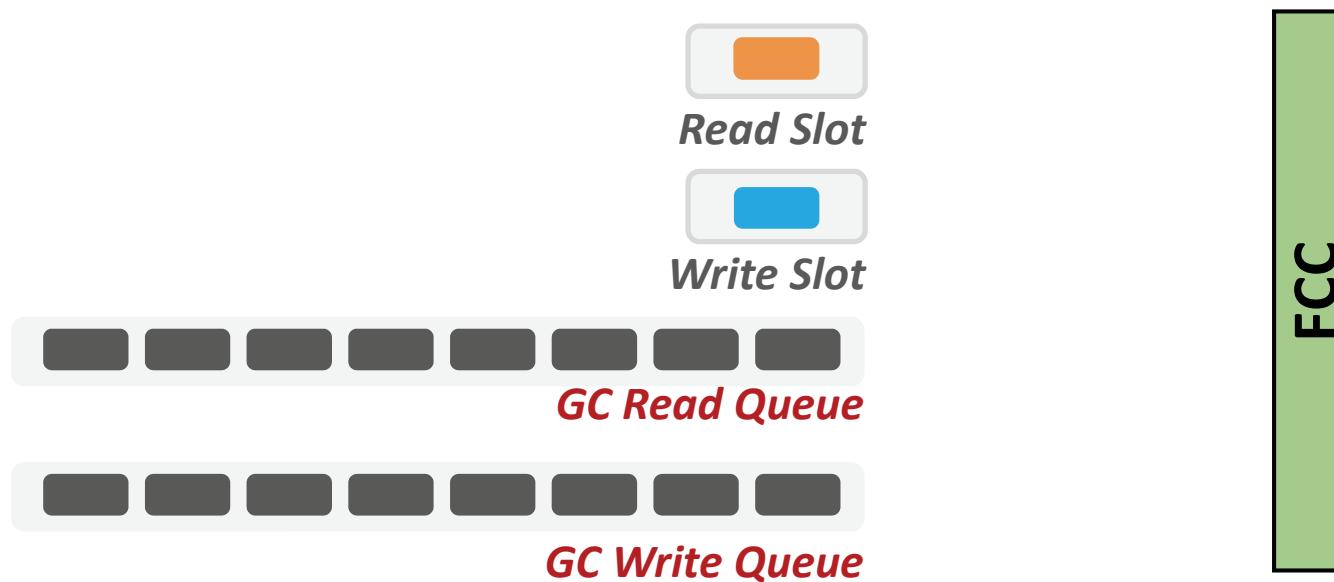
$$PW = \frac{T_{wait}}{T_{memory} + T_{transfer}}$$

*Smaller for reads* ← *Waiting time before the transaction is dispatched to the flash controller* →

- Reads are still **prioritized** over writes
- Reads are only prioritized when their **proportional wait time** is greater than write transaction's proportional wait time

# Stage 3: Wait-Balancing Transaction Selection

SAFARI



1. Estimate proportional wait times for the transactions in the **read slot** and **write slot**
2. If the **read-slot** transaction has a higher proportional wait time, then dispatch it to channel

The number of GC activities is **estimated** based on 1) relative write intensity, and 2) relative usage of the storage space

- FLIN can be implemented in the **firmware** of a modern SSD, and does **not require** specialized hardware
- FLIN has to keep track of
  - flow **intensities** to classify flows into and low-intensity categories,
  - **slowdowns** of individual flash transactions in the queues,
  - the average **slowdown** of each **flow**, and
  - the GC cost estimation data
- Our worst-case estimation shows that the DRAM overhead of FLIN would be **very modest** (< 0.06%)
- The **maximum throughput** of FLIN is identical to the baseline
  - All the processings are performed **off the critical path** of transaction processing

- **MQSim**, an open-source, accurate modern SSD simulator:  
<https://github.com/CMU-SAFARI/MQSim> [FAST'18]

**Table 1: Configuration of the simulated SSD.**

<b>SSD Organization</b>	Host interface: PCIe 3.0 (NVMe 1.2) User capacity: 480 GB 8 channels, 4 dies-per-channel
<b>Flash Communication Interface</b>	ONFI 3.1 (NV-DDR2) Width: 8 bit, Rate: 333 MT/s
<b>Flash Microarchitecture</b>	8 KiB page, 448 B metadata-per-page, 256 pages-per-block, 4096 blocks-per-plane, 2 planes-per-die
<b>Flash Latencies</b> [63]	Read latency: 75 $\mu$ s, Program latency: 1300 $\mu$ s, Erase latency: 3.8 ms

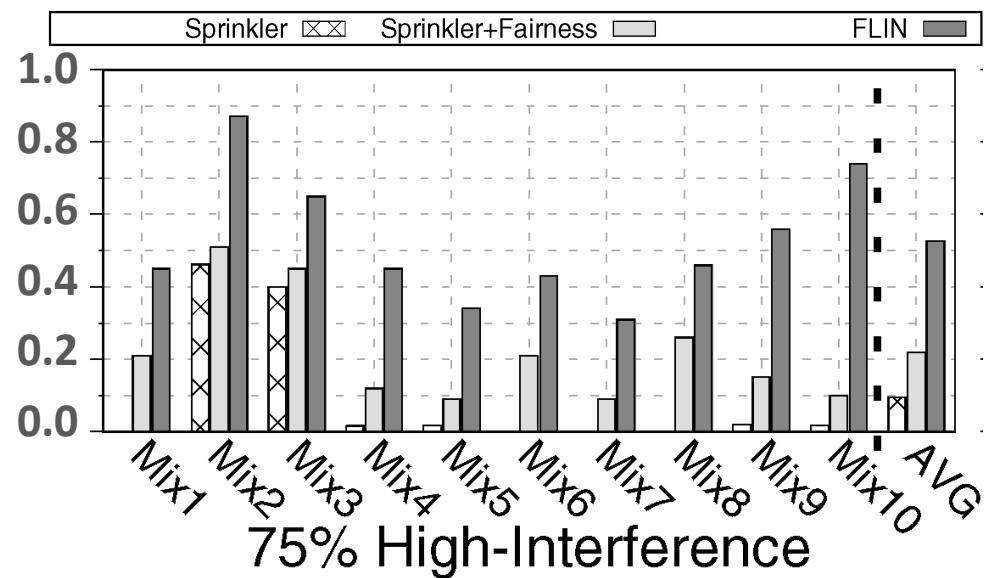


# Experimental Results: Fairness

SAFARI

- For workload mixes 25%, 50%, 75%, and 100%, FLIN improves average fairness by
  - 1.8x, 2.5x, 5.6x, and 54x over Sprinkler, and
  - 1.3x, 1.6x, 2.4x, and 3.2x over Sprinkler+Fairness

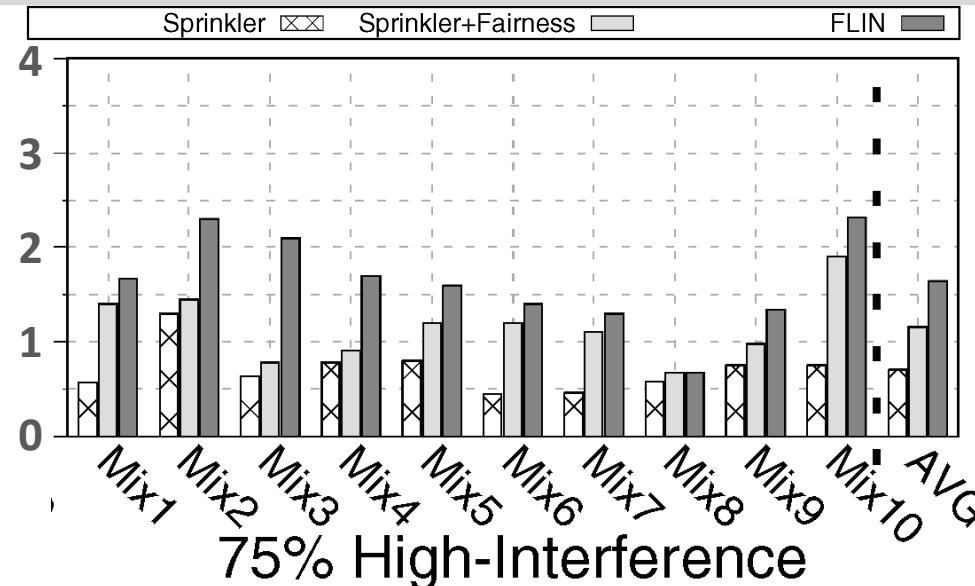
- Sprinkler+Fairness improves fairness over Sprinkler
  - Due to its inclusion of fairness control



- Sprinkler+Fairness does not consider all sources of interference, and therefore has a **much lower fairness** than FLIN

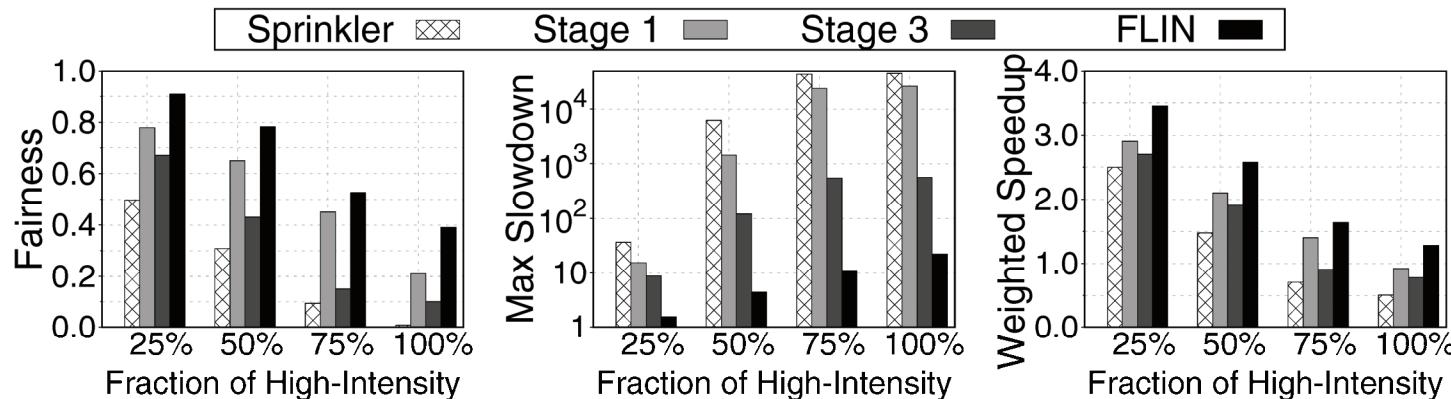
# Experimental Results: Weighted Speedup

SAFARI



- Across the four workload categories, FLIN on average improves the weighted speedup by
  - 38%, 74%, 132%, 156% over Sprinkler, and
  - 21%, 32%, 41%, 76% over Sprinkler+Fairness
- FLIN's fairness control mechanism improves the performance of low-interference flows
- Weighted-speedup remains low for Sprinkler+Fairness as its throughput control mechanism leaves many resources idle

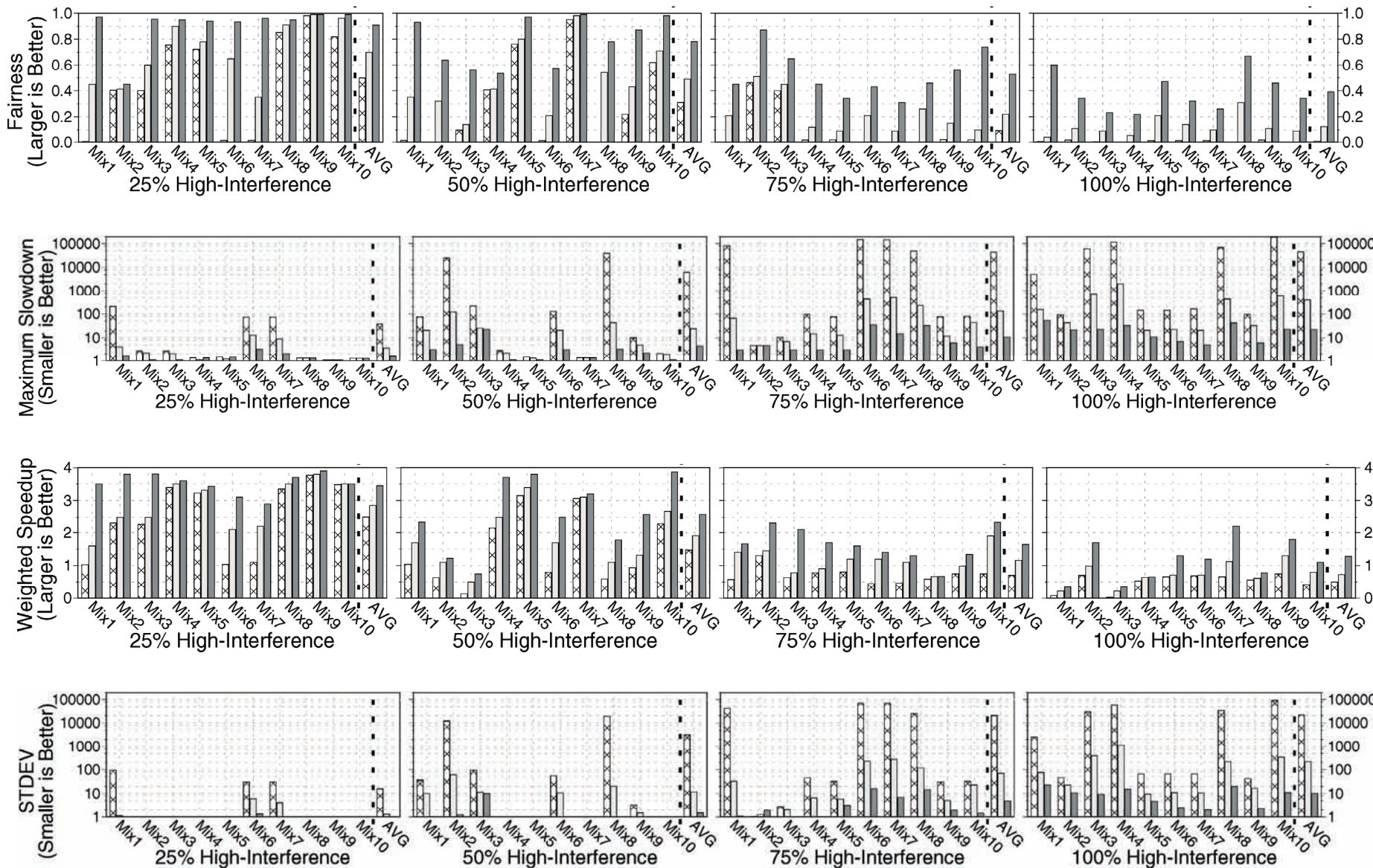
# Effect of Different FLIN Stages



- The individual stages of FLIN improve both fairness and performance over Sprinkler, as each stage works to reduce some sources of interference
- The fairness and performance improvements of Stage 1 are much higher than those of Stage 3
  - I/O intensity is the most dominant source of interference
- Stage 3 reduces the maximum slowdown by a greater amount than Stage 1
  - GC operations can significantly increase the stall time of transactions

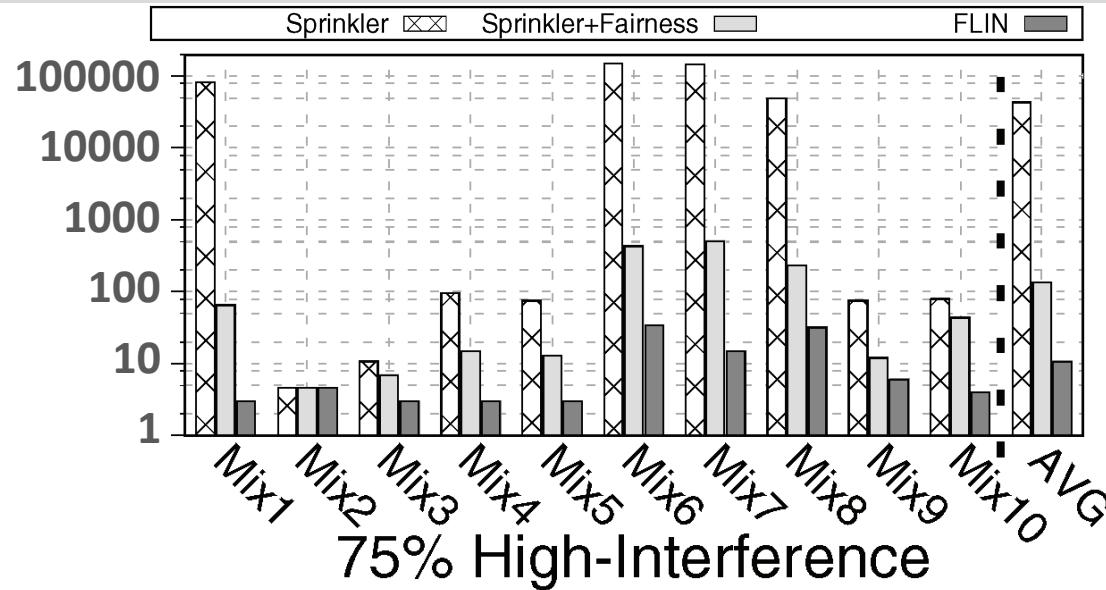
# Fairness and Performance of FLIN

SAFARI



# Experimental Results: Maximum Slowdown

SAFARI



- Across the four workload categories, FLIN reduces the average maximum slowdown by
  - 24x, 1400x, 3231x, and 1597x over Sprinkler, and
  - 2.3x, 5.5x, 12x, and 18x over Sprinkler+Fairness
- Across all of the workloads, **no flow** has a maximum slowdown greater than 80x under FLIN
- There are **several flows** that have maximum slowdowns over 500x with Sprinkler and Sprinkler+Fairness

- FLIN is a lightweight transaction scheduler for modern multi-queue SSDs (MQ-SSDs), which provides fairness among concurrently-running flows
- FLIN uses a three-stage design to protect against all four major sources of interference that exist in real MQ-SSDs
- FLIN effectively improves both fairness and system performance compared to state-of-the-art device-level schedulers
- FLIN is implemented fully within the SSD firmware with a very modest DRAM overhead (<0.06%)
- Future Work
  - Coordinated OS/FLIN mechanisms