ScaleLFS: A Log-Structured File System with Scalable Garbage Collection for Commodity SSDs

Jin Yong Ha[1], Sangjin Lee[2], Hyeonsang Eom[1], Yongseok Son[2]

[1] Seoul National University
[2] Chung-Ang University

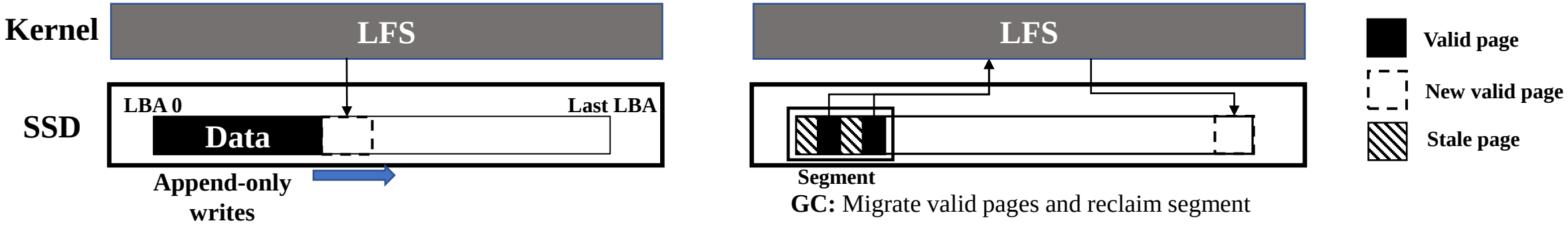
Log-structured File System (LFS)

❖Append-only manner

- Data are placed to consecutive logical block address (LBA)
- Leverage high **sequential write performance** of block device
- **Stale pages induced by out-of-place update**
- F2FS, JFFS, NILFS, Etc.

❖Garbage collection (GC)

- **Reclaim spaces** occupied by stale pages in segment unit
- Migrate dispersed valid pages (non-stale pages) to consecutive space



GC: Migrate valid pages and reclaim segment

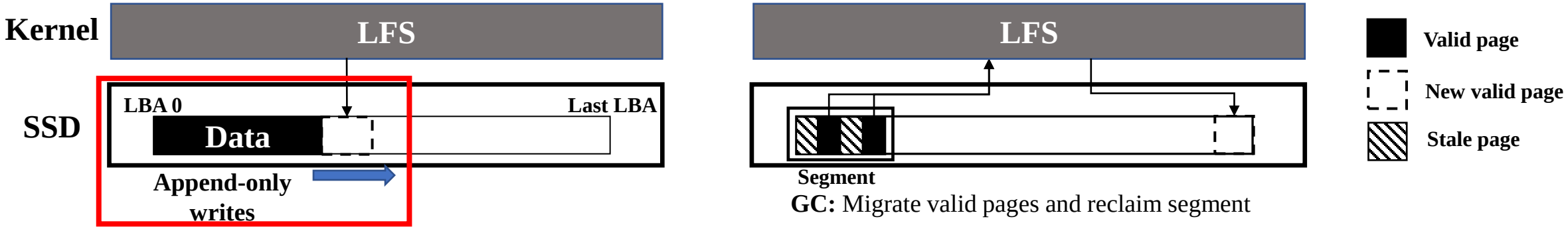
Log-structured File System (LFS)

❖Append-only manner

- Data are placed to consecutive logical block address (LBA)
- Leverage high **sequential write performance** of block device
- **Stale pages induced by out-of-place update**
- F2FS, JFFS, NILFS, Etc.

❖Garbage collection (GC)

- **Reclaim spaces** occupied by stale pages in segment unit
- Migrate dispersed valid pages (non-stale pages) to consecutive space



GC: Migrate valid pages and reclaim segment

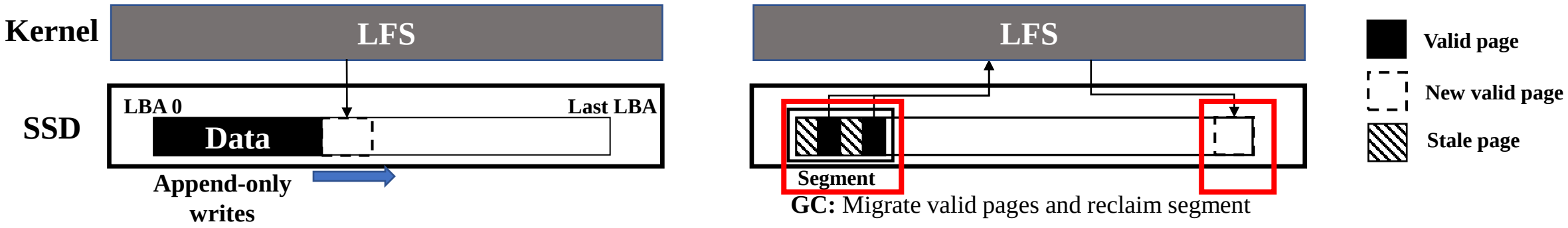
Log-structured File System (LFS)

❖Append-only manner

- Data are placed to consecutive logical block address (LBA)
- Leverage high **sequential write performance** of block device
- **Stale pages induced by out-of-place update**
- F2FS, JFFS, NILFS, Etc.

❖Garbage collection (GC)

- **Reclaim spaces** occupied by stale pages in segment unit
- Migrate dispersed valid pages (non-stale pages) to consecutive space

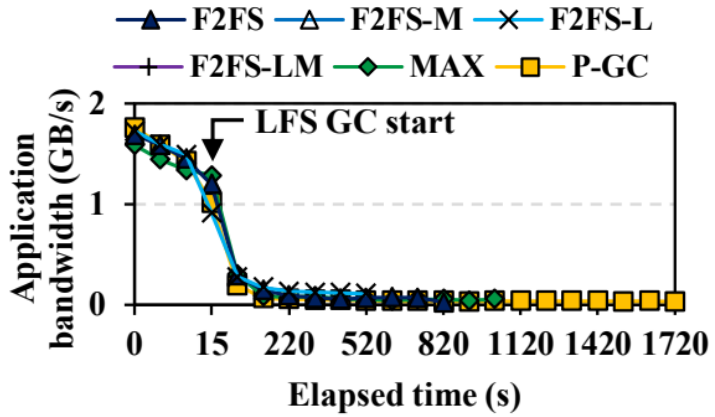


GC: Migrate valid pages and reclaim segment

Sustained Performance of LFS

❖Overhead of GC

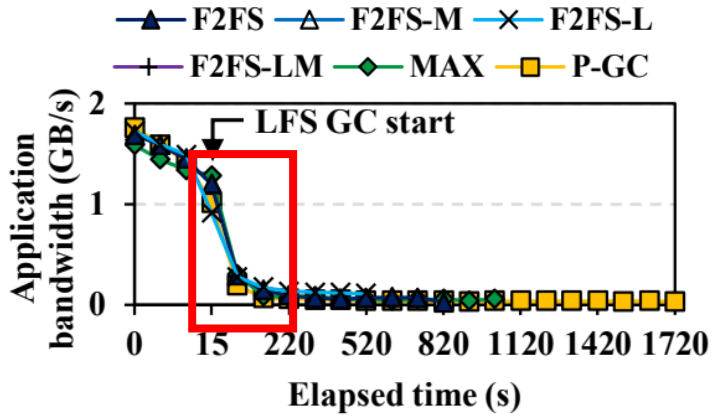
- Bottleneck induced by **single threaded GC**
- Application bandwidth decreases by up to **68 times**



Sustained Performance of LFS

❖Overhead of GC

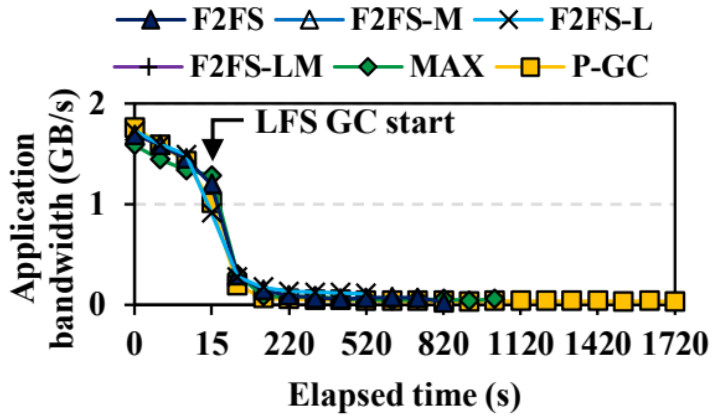
- Bottleneck induced by **single threaded GC**
- Application bandwidth decreases by up to **68 times**



Sustained Performance of LFS

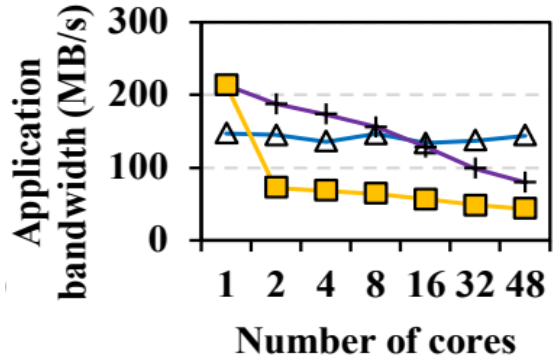
❖Overhead of GC

- Bottleneck induced by **single threaded GC**
- Application bandwidth decreases by up to **68 times**



❖Naïvely scaling out threads

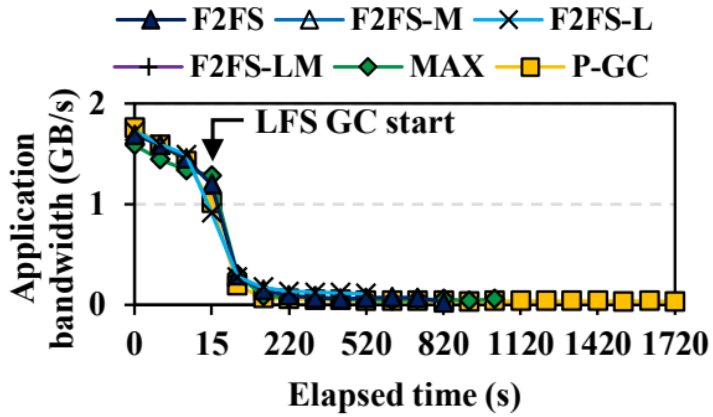
- No performance gain
- Even decreased performance due to **increased lock contention**



Sustained Performance of LFS

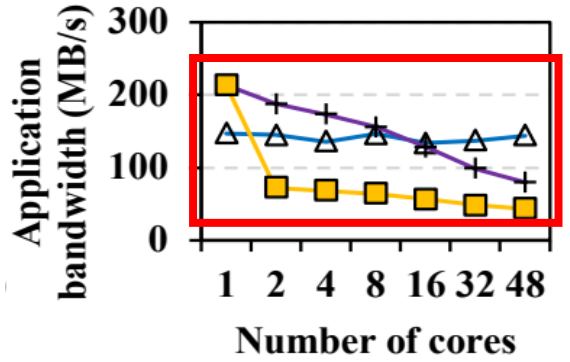
❖Overhead of GC

- Bottleneck induced by **single threaded GC**
- Application bandwidth decreases by up to **68 times**



❖Naïvely scaling out threads

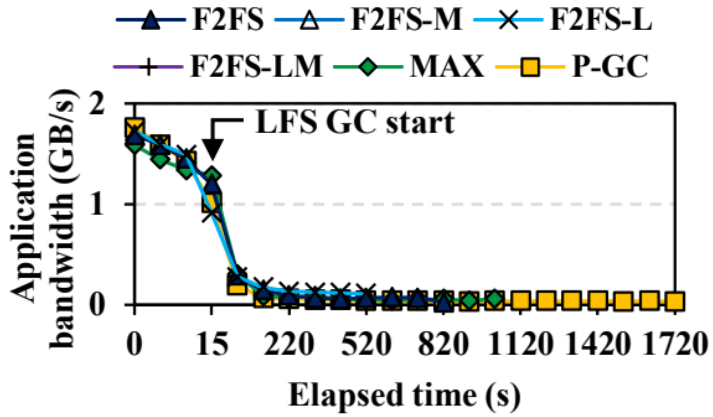
- No performance gain
- Even decreased performance due to **increased lock contention**



Sustained Performance of LFS

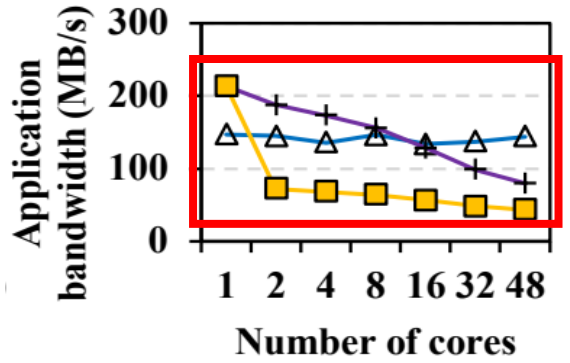
❖Overhead of GC

- Bottleneck induced by **single threaded GC**
- Application bandwidth decreases by up to **68 times**



❖Naïvely scaling out threads

- No performance gain
- Even decreased performance due to **increased lock contention**



Concurrent and scalable techniques are required

Obstacles for Scaling GC procedure

Obstacles for Scaling GC procedure

❖**Shared resource**
- GC destination segment
- Page cache for GC

→ **Resource dedication required**

Obstacles for Scaling GC procedure

❖**Shared resource**
- GC destination segment
- Page cache for GC

→ **Resource dedication required**

❖**Excessive contention**
- Victim selection protected exclusively

→ **Concurrent victim selection required**

Obstacles for Scaling GC procedure

❖**Shared resource**
- GC destination segment
- Page cache for GC

→ **Resource dedication required**

❖**Excessive contention**
- Victim selection protected exclusively

→ **Concurrent victim selection required**

❖**Over-strict synchronization**
- A single lock for segment metadata (valid bitmap and count)

→ **Concurrent access/update on segment metadata**

Obstacles for Scaling GC procedure

❖**Shared resource**
- GC destination segment
- Page cache for GC

→ **Resource dedication required**

❖**Excessive contention**
- Victim selection protected exclusively

→ **Concurrent victim selection required**

❖**Over-strict synchronization**
- A single lock for segment metadata (valid bitmap and count)

→ **Concurrent access/update on segment metadata**

❖**Coarse-grained data protection**
- File level data consistency management between GC and I/O threads

→ **Page-level data consistency management**

ScaleLFS

❖Dedicated garbage collector (DGC)

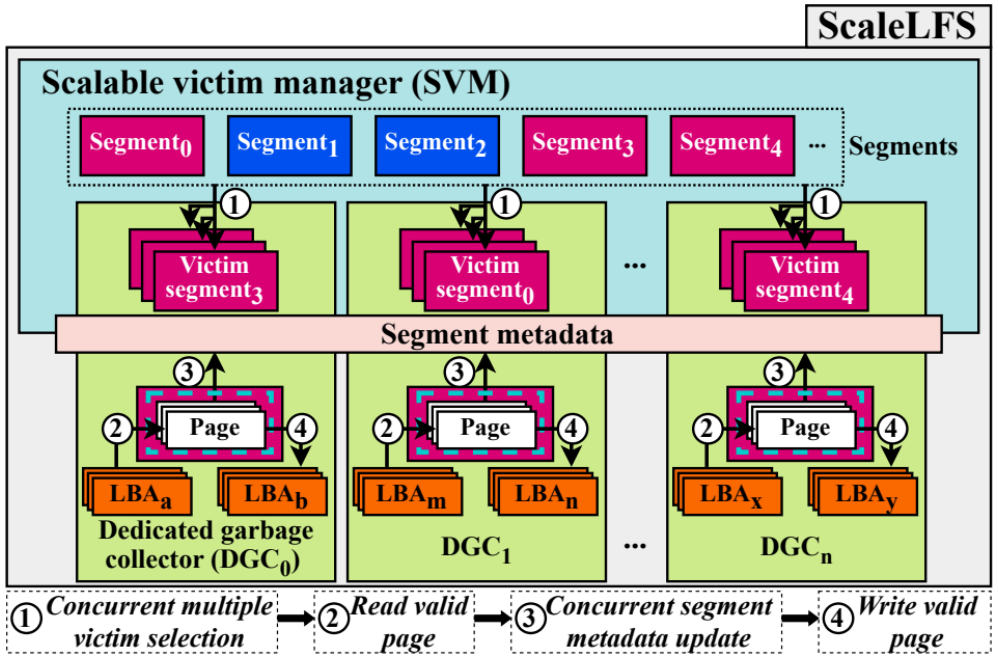
- GC thread with dedication strategies
  - **Dedicated** page buffer
  - **Dedicated** destination segment (write stream)

❖Scalable victim manager (SVM)

- Victim selection
  - Atomic victim bitmap
  - **Concurrent victim selection**
- Segment metadata management
  - Atomic valid page bitmap
  - Atomic valid page count
  - **Loose-synchronization update**

❖Scalable victim protector (SVP)

- Page-level conflict management
  - Concurrent hash table per file



ScaleLFS

❖Dedicated garbage collector (DGC)

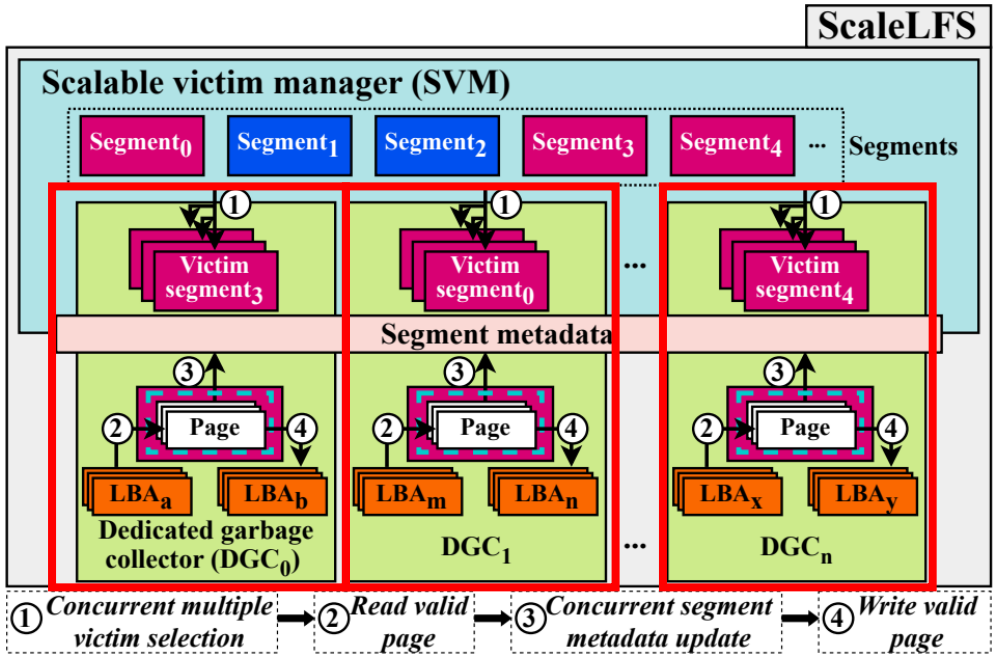
- GC thread with dedication strategies
  - **Dedicated** page buffer
  - **Dedicated** destination segment (write stream)

❖Scalable victim manager (SVM)

- Victim selection
  - Atomic victim bitmap
  - **Concurrent victim selection**
- Segment metadata management
  - Atomic valid page bitmap
  - Atomic valid page count
  - **Loose-synchronization update**

❖Scalable victim protector (SVP)

- Page-level conflict management
  - Concurrent hash table per file



ScaleLFS

❖Dedicated garbage collector (DGC)

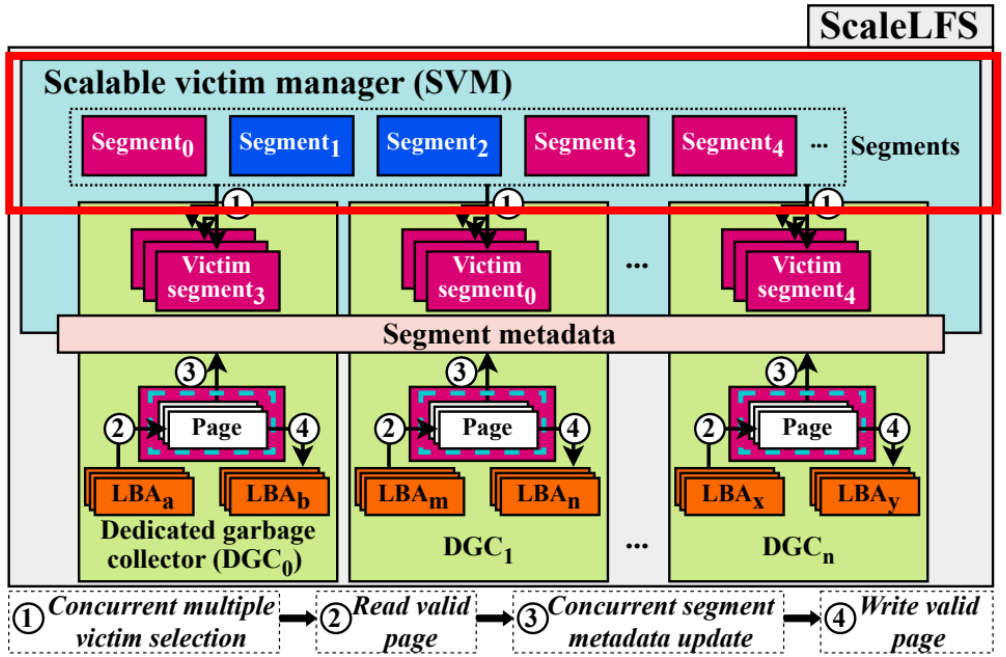
- GC thread with dedication strategies
  - **Dedicated** page buffer
  - **Dedicated** destination segment (write stream)

❖Scalable victim manager (SVM)

- Victim selection
  - Atomic victim bitmap
  - **Concurrent victim selection**
- Segment metadata management
  - Atomic valid page bitmap
  - Atomic valid page count
  - **Loose-synchronization update**

❖Scalable victim protector (SVP)

- Page-level conflict management
  - Concurrent hash table per file



ScaleLFS

❖Dedicated garbage collector (DGC)

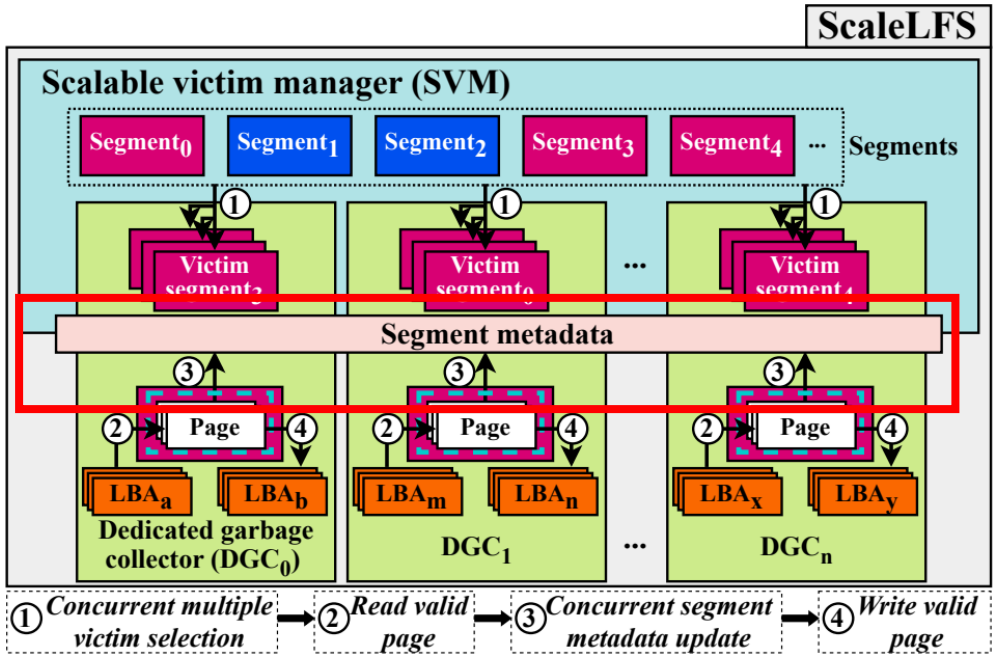
- GC thread with dedication strategies
  - **Dedicated** page buffer
  - **Dedicated** destination segment (write stream)

❖Scalable victim manager (SVM)

- Victim selection
  - Atomic victim bitmap
  - **Concurrent victim selection**
- Segment metadata management
  - Atomic valid page bitmap
  - Atomic valid page count
  - **Loose-synchronization update**

❖Scalable victim protector (SVP)

- Page-level conflict management
  - Concurrent hash table per file



ScaleLFS

❖Dedicated garbage collector (DGC)

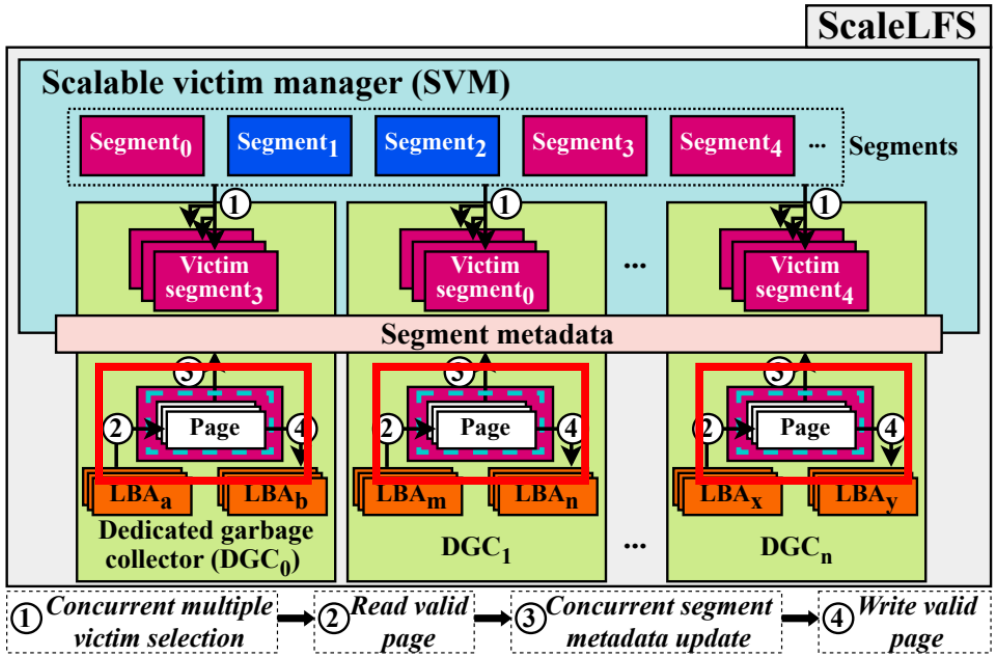
- GC thread with dedication strategies
  - **Dedicated** page buffer
  - **Dedicated** destination segment (write stream)

❖Scalable victim manager (SVM)

- Victim selection
  - Atomic victim bitmap
  - **Concurrent victim selection**
- Segment metadata management
  - Atomic valid page bitmap
  - Atomic valid page count
  - **Loose-synchronization update**

❖Scalable victim protector (SVP)

- Page-level conflict management
  - Concurrent hash table per file



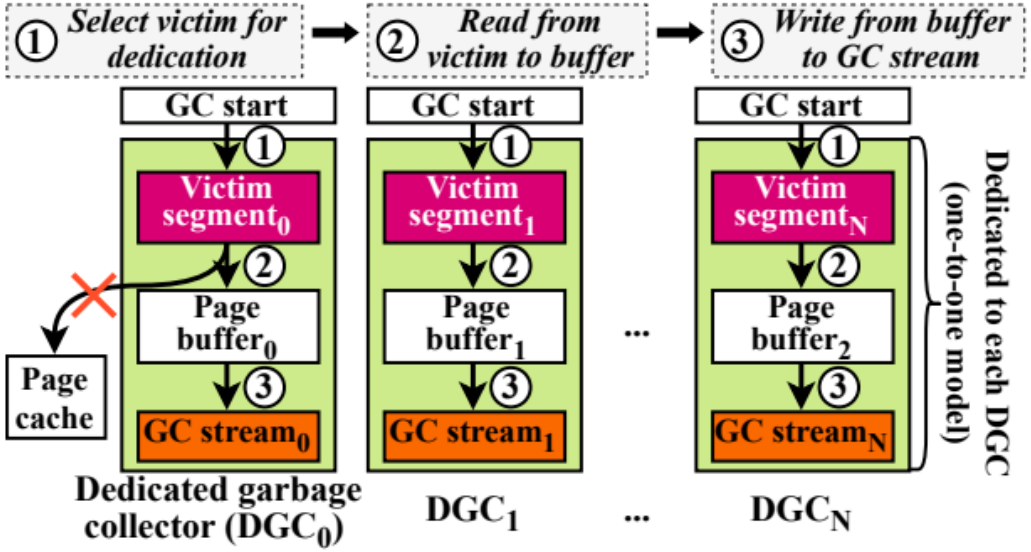
Dedicated Garbage Collector (DGC)

❖Dedicated page buffer (DPB)

- Per-GC-thread GC data buffer
- **Avoid expensive page cache overhead**

❖Dedicated write stream (DWS)

- Per-GC-thread write stream
- **Lock-free**
  - Allocating new block address
  - Updating sequential write curser in segment
  - Merging consecutive write requests



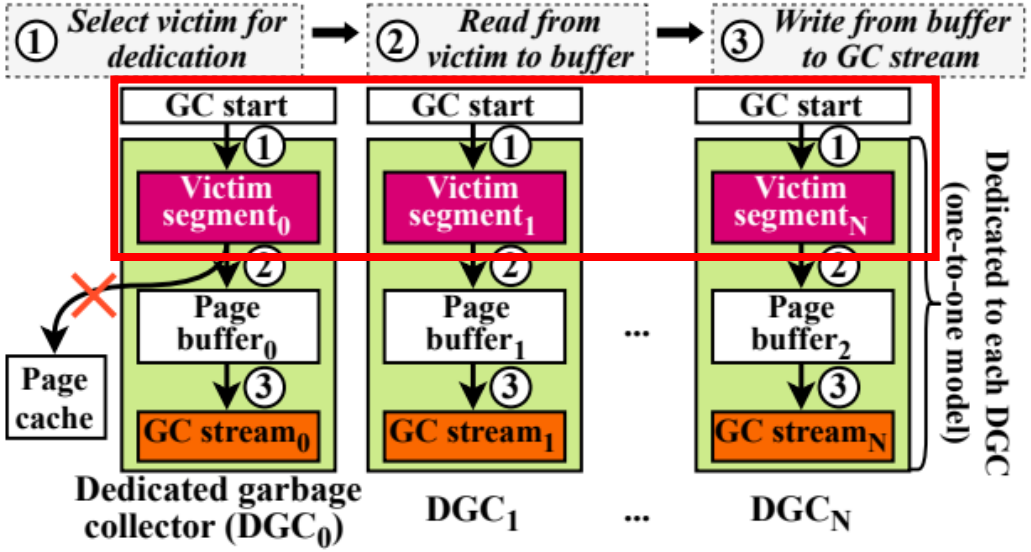
Dedicated Garbage Collector (DGC)

❖Dedicated page buffer (DPB)

- Per-GC-thread GC data buffer
- **Avoid expensive page cache overhead**

❖Dedicated write stream (DWS)

- Per-GC-thread write stream
- **Lock-free**
  - Allocating new block address
  - Updating sequential write curser in segment
  - Merging consecutive write requests



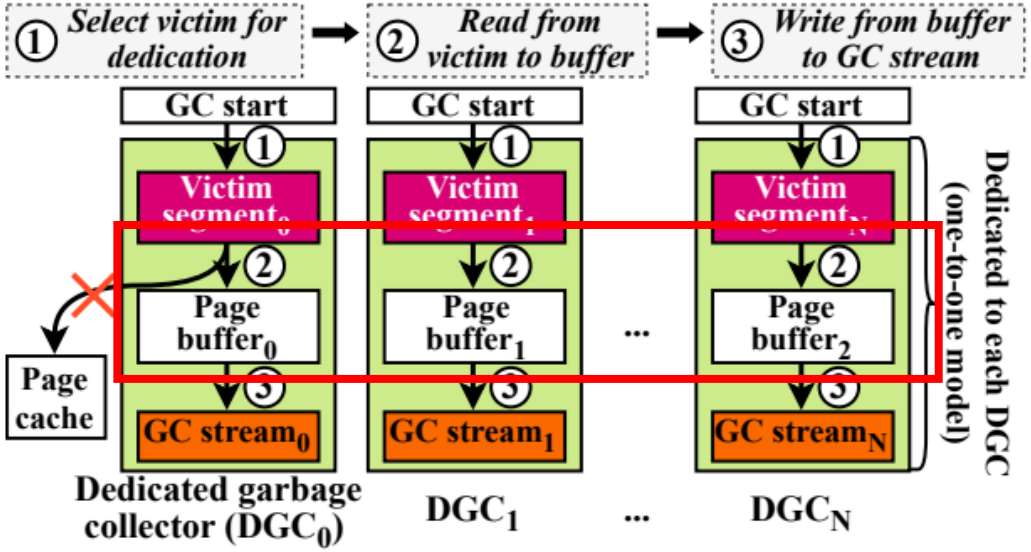
Dedicated Garbage Collector (DGC)

❖Dedicated page buffer (DPB)

- Per-GC-thread GC data buffer
- **Avoid expensive page cache overhead**

❖Dedicated write stream (DWS)

- Per-GC-thread write stream
- **Lock-free**
  - Allocating new block address
  - Updating sequential write curser in segment
  - Merging consecutive write requests



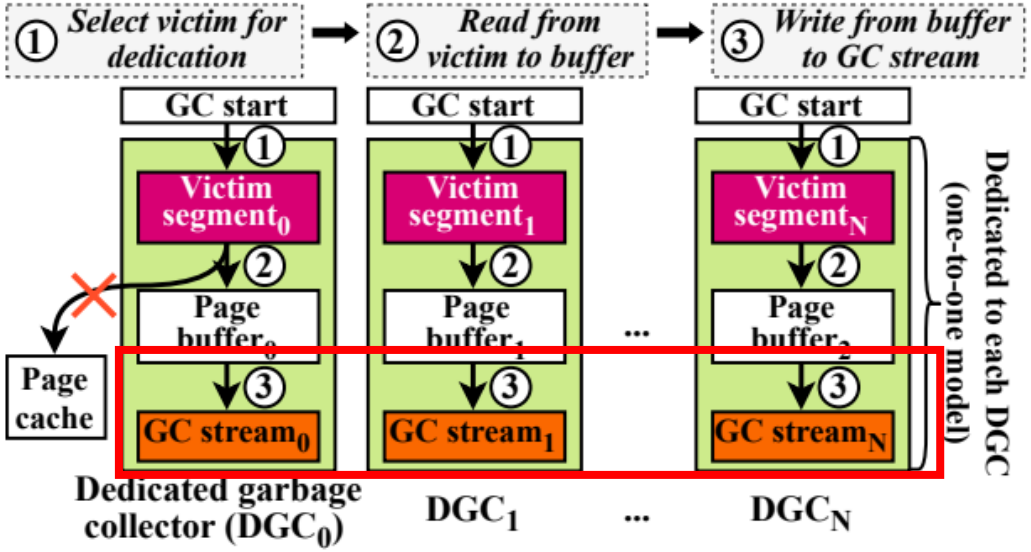
Dedicated Garbage Collector (DGC)

❖Dedicated page buffer (DPB)

- Per-GC-thread GC data buffer
- **Avoid expensive page cache overhead**

❖Dedicated write stream (DWS)

- Per-GC-thread write stream
- **Lock-free**
  - Allocating new block address
  - Updating sequential write curser in segment
  - Merging consecutive write requests

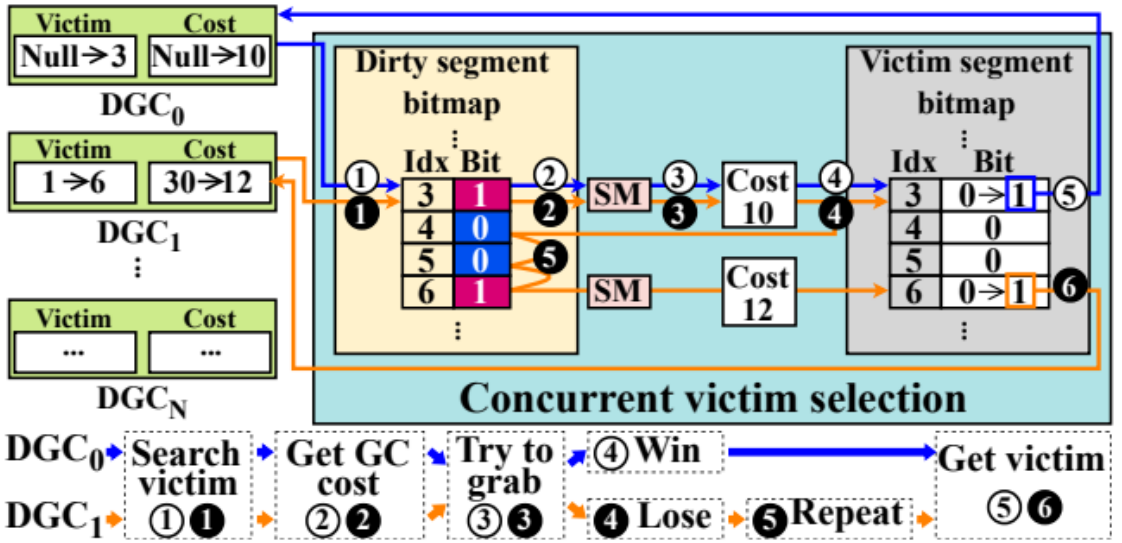


Scalable Victim Manager (1)

❖Concurrent victim selection (CVS)

- Two atomic bitmaps
  - Dirty segment bitmap (GC victim candidates)
  - Victim segment bitmap (Selected GC victim)

- Lock free victim selection procedure
  - **Concurrently scan** dirty bitmap
  - Select segment having least valid pages
  - **Concurrently mark** selected segment to victim bitmap

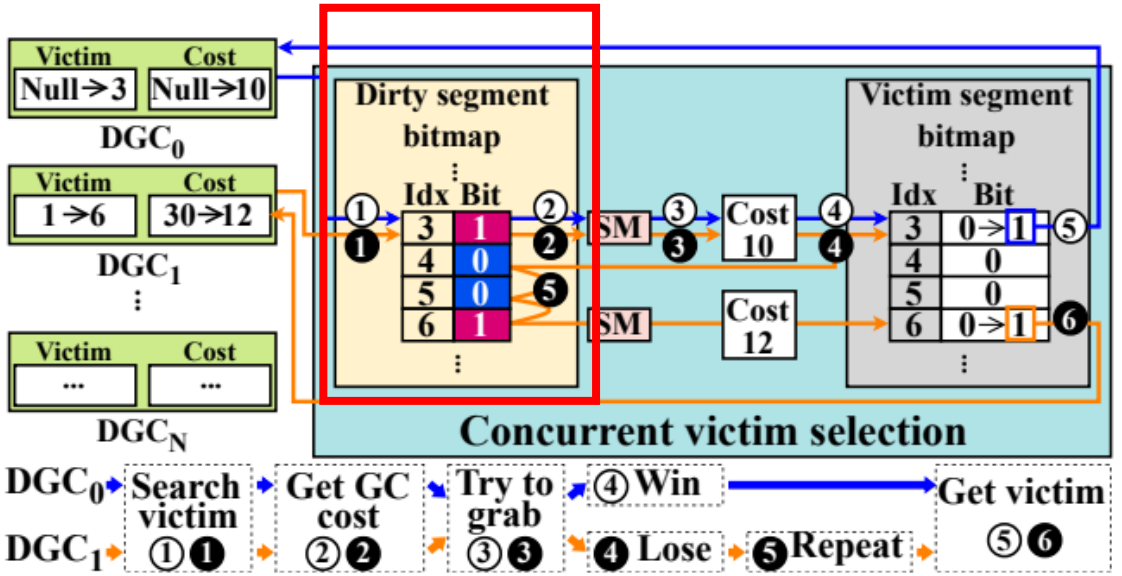


Scalable Victim Manager (1)

❖Concurrent victim selection (CVS)

- Two atomic bitmaps
  - Dirty segment bitmap (GC victim candidates)
  - Victim segment bitmap (Selected GC victim)

- Lock free victim selection procedure
  - **Concurrently scan** dirty bitmap
  - Select segment having least valid pages
  - **Concurrently mark** selected segment to victim bitmap

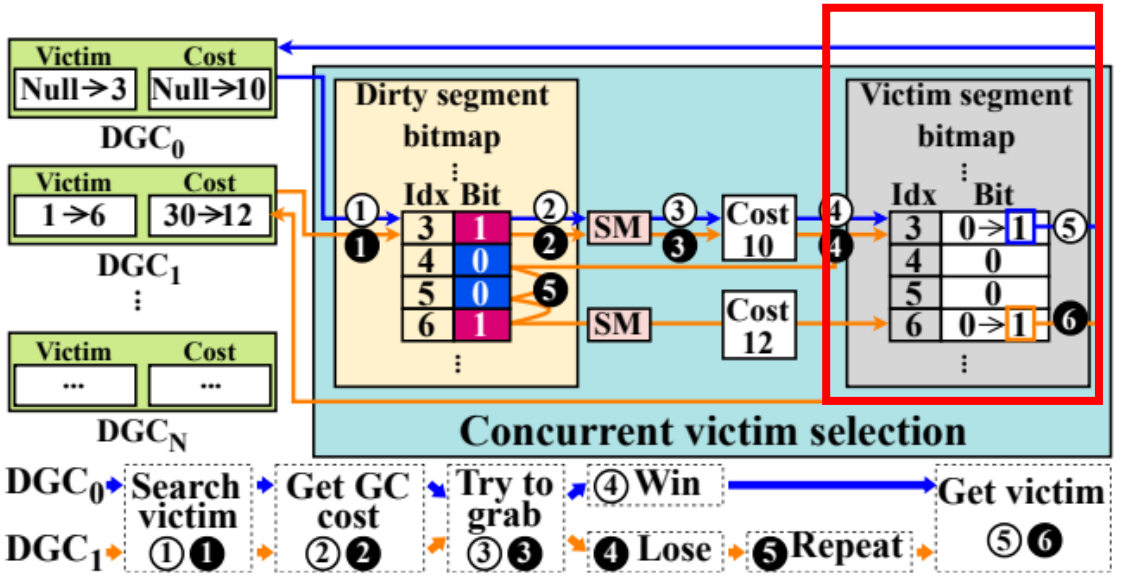


Scalable Victim Manager (1)

❖Concurrent victim selection (CVS)

- Two atomic bitmaps
  - Dirty segment bitmap (GC victim candidates)
  - Victim segment bitmap (Selected GC victim)

- Lock free victim selection procedure
  - **Concurrently scan** dirty bitmap
  - Select segment having least valid pages
  - **Concurrently mark** selected segment to victim bitmap

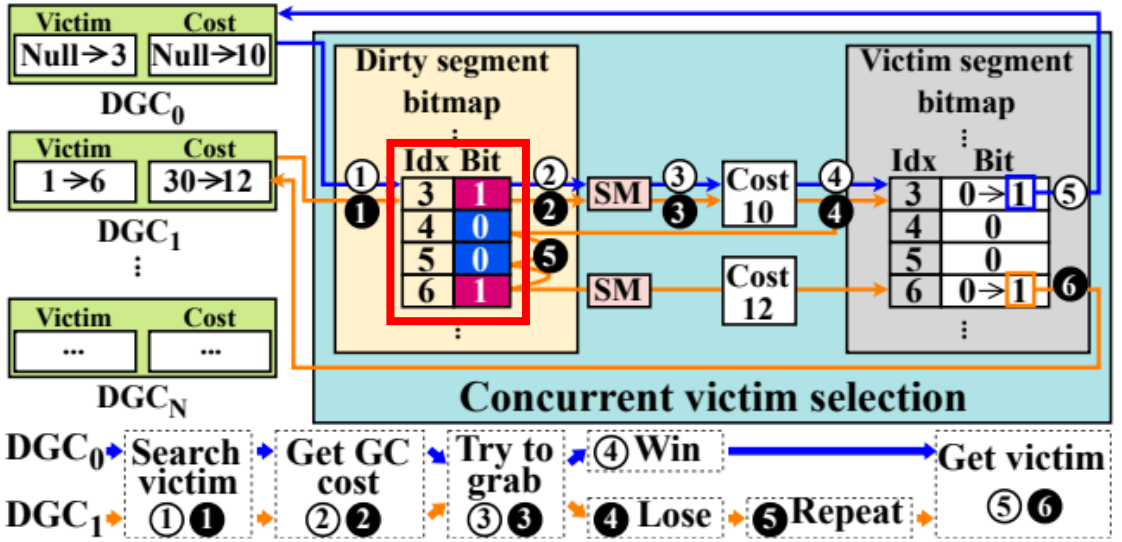


Scalable Victim Manager (1)

❖Concurrent victim selection (CVS)

- Two atomic bitmaps
  - Dirty segment bitmap (GC victim candidates)
  - Victim segment bitmap (Selected GC victim)

- Lock free victim selection procedure
  - **Concurrently scan** dirty bitmap
  - Select segment having least valid pages
  - **Concurrently mark** selected segment to victim bitmap

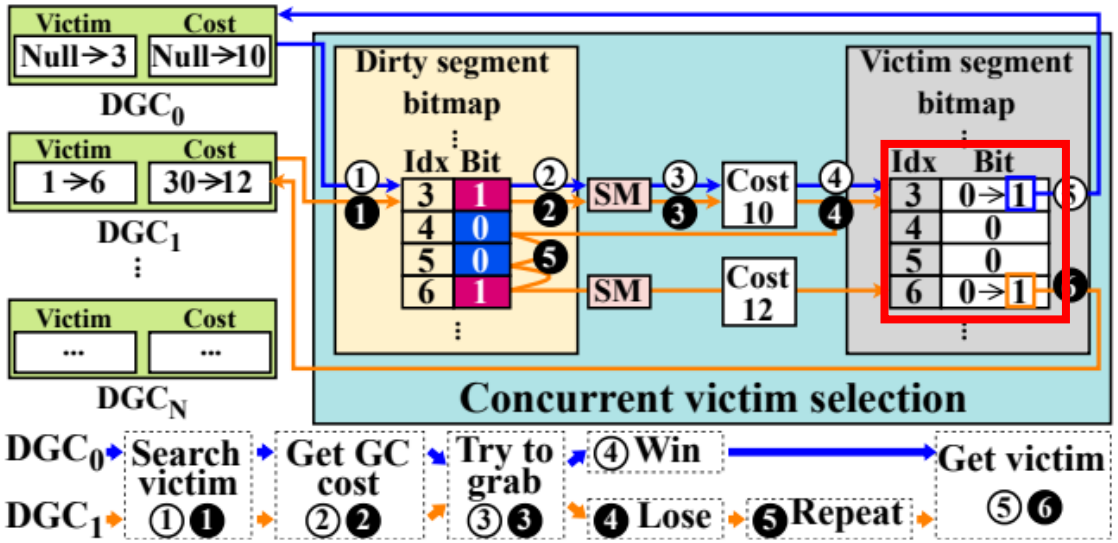


Scalable Victim Manager (1)

❖Concurrent victim selection (CVS)

- Two atomic bitmaps
  - Dirty segment bitmap (GC victim candidates)
  - Victim segment bitmap (Selected GC victim)

- Lock free victim selection procedure
  - **Concurrently scan** dirty bitmap
  - Select segment having least valid pages
  - **Concurrently mark** selected segment to victim bitmap



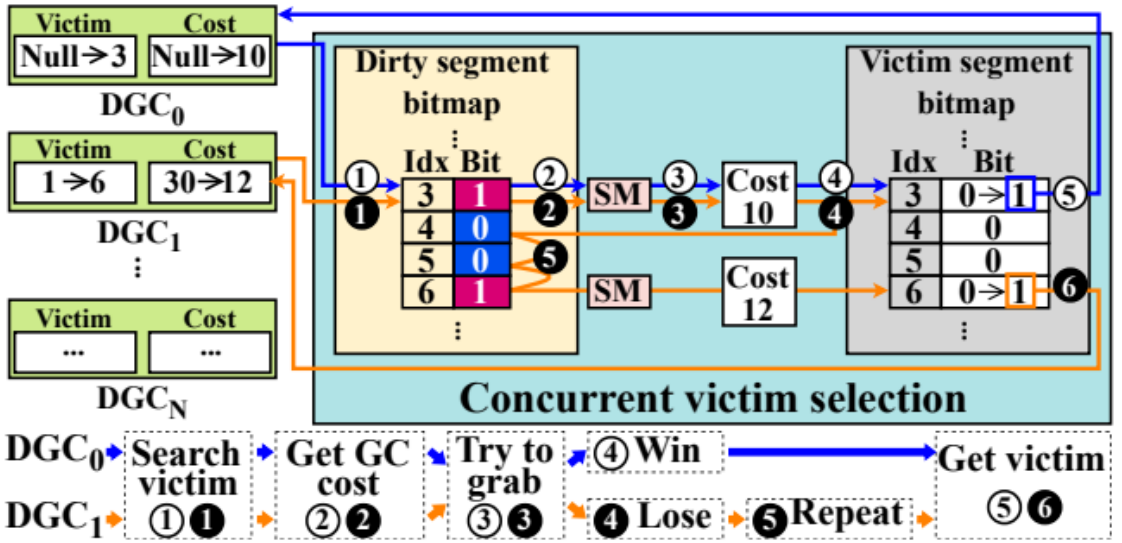
Scalable Victim Manager (1)

❖Concurrent victim selection (CVS)

- Two atomic bitmaps
  - Dirty segment bitmap (GC victim candidates)
  - Victim segment bitmap (Selected GC victim)
- Lock free victim selection procedure
  - **Concurrently scan** dirty bitmap
  - Select segment having least valid pages
  - **Concurrently mark** selected segment to victim bitmap

❖Data race between DGCs

- Multiple DGCs try to select the same victim segment
- Utilize atomic ***test_and_set_bit()***
  - Try update same offset of victim segment bitmap
  - **Only one DGC can update bit to 1 (win)**
  - DGC that lost try to select another segment



Scalable Victim Manager (2)

❖**Loose-synchronization Update (LSU)**

- Manage per-segment metadata
  - Atomic valid page bitmap (VPB)
  - Atomic valid page count (VPC)
- **Lock-free update and access** to valid bitmap and valid count

Scalable Victim Manager (2)

❖Loose-synchronization Update (LSU)

- Manage per-segment metadata
  - Atomic valid page bitmap (VPB)
  - Atomic valid page count (VPC)
- **Lock-free update and access** to valid bitmap and valid count

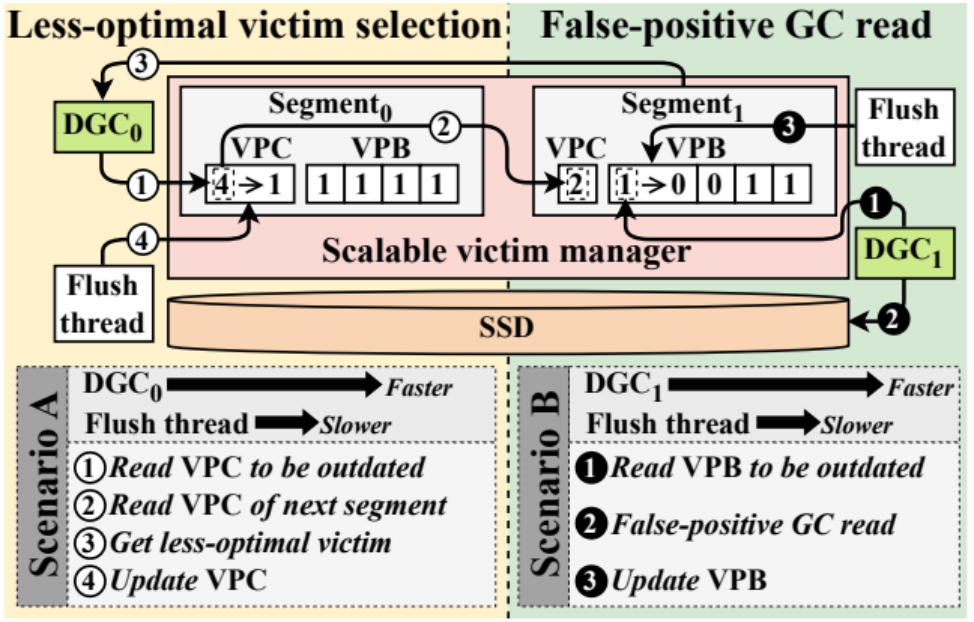
❖Two side effects due to separately updated VPB and VPC

- Less-optimal victim selection
  - Selecting best victim segment is not guaranteed
- False-positive GC read
  - Unnecessary GC read can be submitted

Scalable Victim Manager (3)

❖Less-optimal victim selection

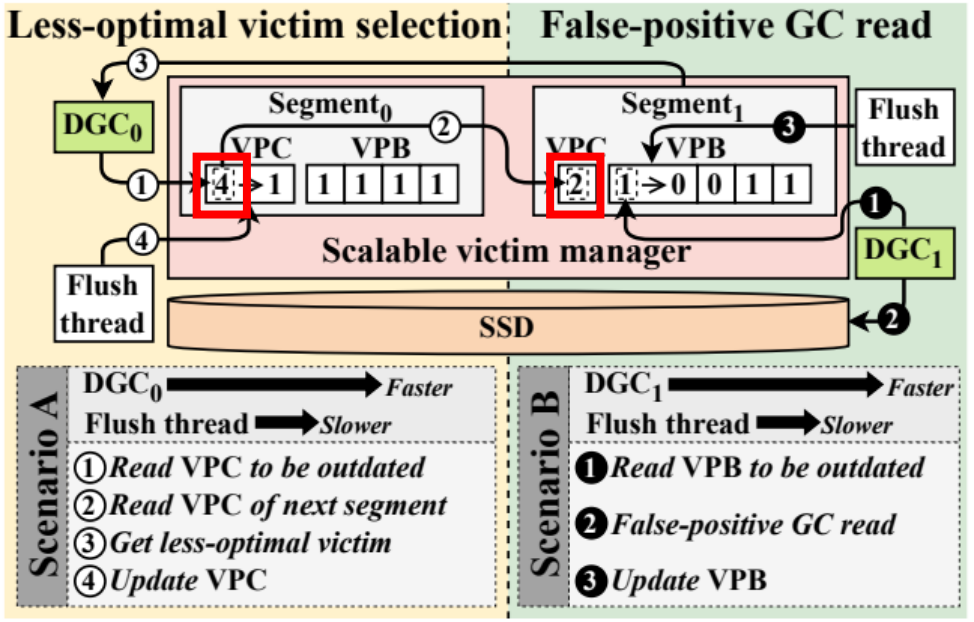
- Scenario
  - VPC decreases during victim selection
  - Update is not aware to DGC (**outdated VPC**)
  - Less-optimal victim selection caused by outdated VPC
- **Negligible write amplification** observed



Scalable Victim Manager (3)

❖Less-optimal victim selection

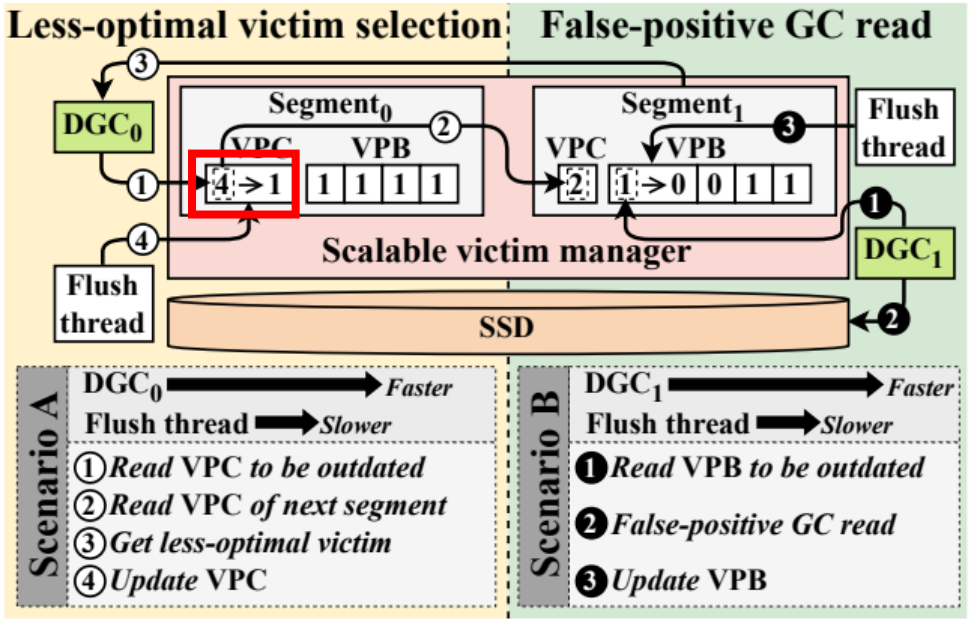
- Scenario
  - VPC decreases during victim selection
  - Update is not aware to DGC (**outdated VPC**)
  - Less-optimal victim selection caused by outdated VPC
- **Negligible write amplification** observed



Scalable Victim Manager (3)

❖Less-optimal victim selection

- Scenario
  - VPC decreases during victim selection
  - Update is not aware to DGC (**outdated VPC**)
  - Less-optimal victim selection caused by outdated VPC
- **Negligible write amplification** observed



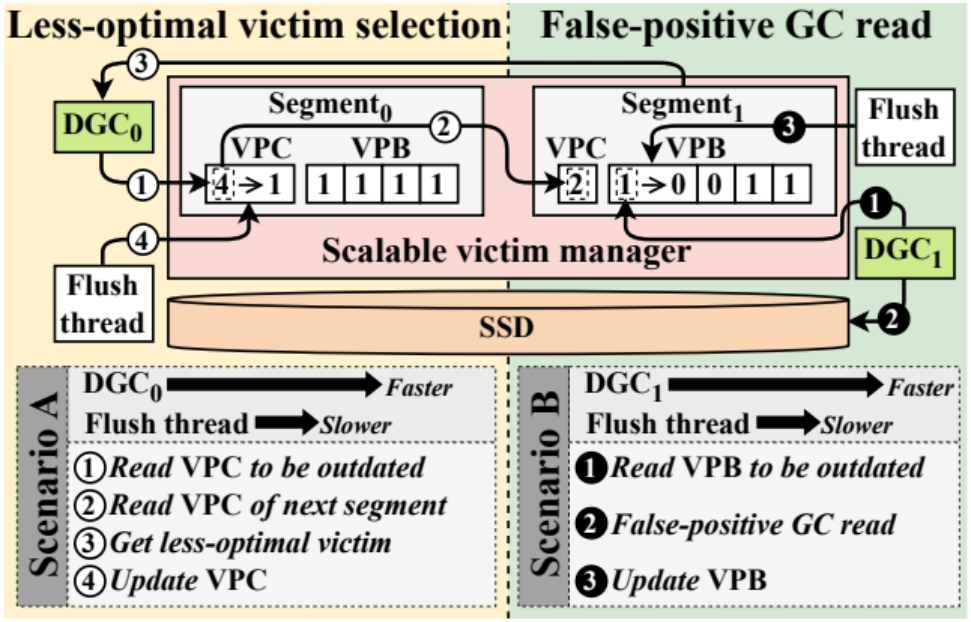
Scalable Victim Manager (3)

❖Less-optimal victim selection

- Scenario
  - VPC decreases during victim selection
  - Update is not aware to DGC (**outdated VPC**)
  - Less-optimal victim selection caused by outdated VPC
- **Negligible write amplification** observed

❖False-positive GC read

- Scenario
  - VPB is cleared during GC read
  - Update is not aware to DGC (**outdated VPB**)
  - GC read to stale page is submitted
- **Data consistency can be violated**



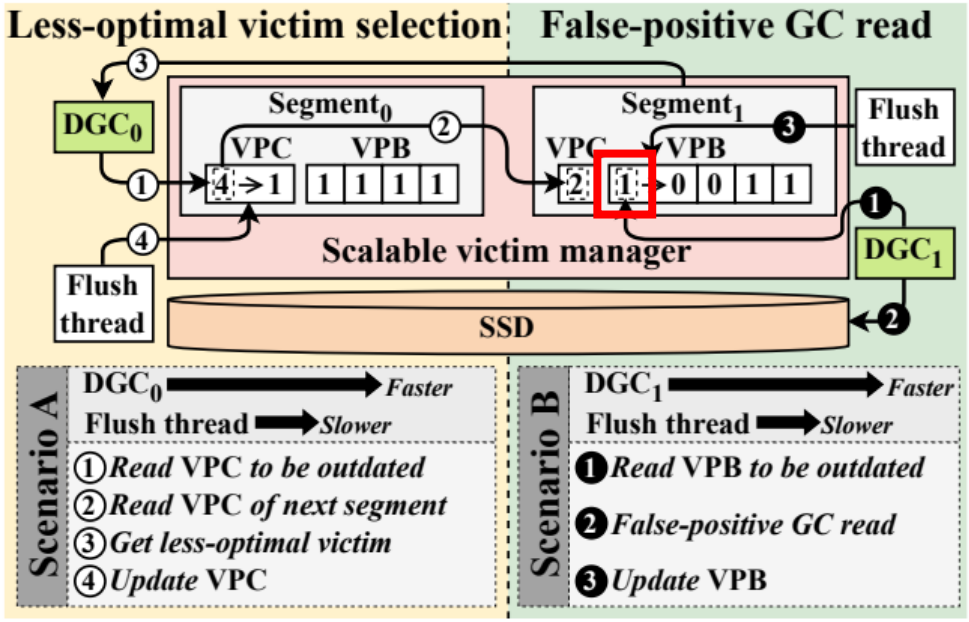
Scalable Victim Manager (3)

❖Less-optimal victim selection

- Scenario
  - VPC decreases during victim selection
  - Update is not aware to DGC (**outdated VPC**)
  - Less-optimal victim selection caused by outdated VPC
- **Negligible write amplification** observed

❖False-positive GC read

- Scenario
  - VPB is cleared during GC read
  - Update is not aware to DGC (**outdated VPB**)
  - GC read to stale page is submitted
- **Data consistency can be violated**



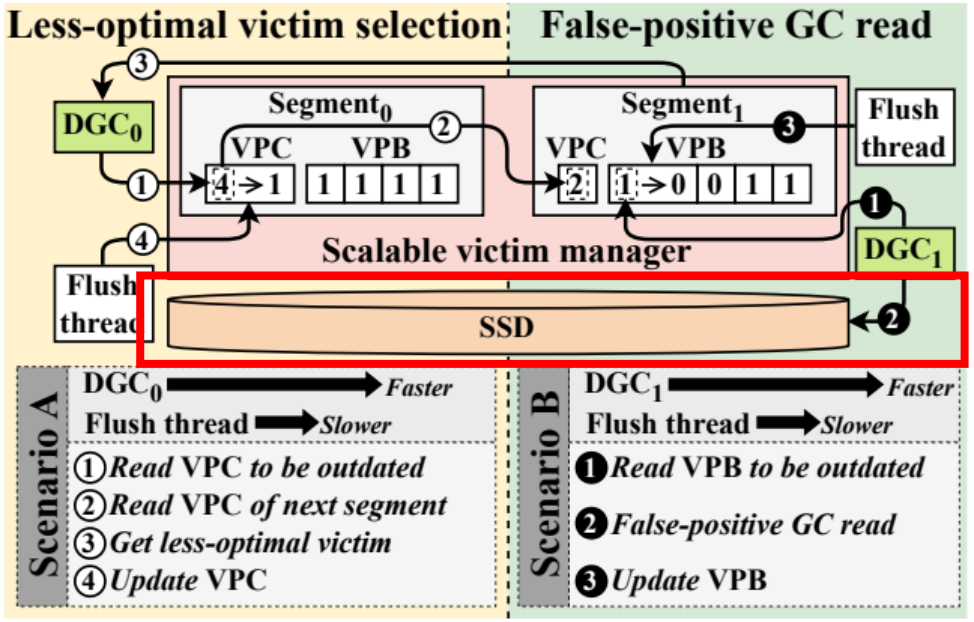
Scalable Victim Manager (3)

❖Less-optimal victim selection

- Scenario
  - VPC decreases during victim selection
  - Update is not aware to DGC (**outdated VPC**)
  - Less-optimal victim selection caused by outdated VPC
- **Negligible write amplification** observed

❖False-positive GC read

- Scenario
  - VPB is cleared during GC read
  - Update is not aware to DGC (**outdated VPB**)
  - GC read to stale page is submitted
- **Data consistency can be violated**



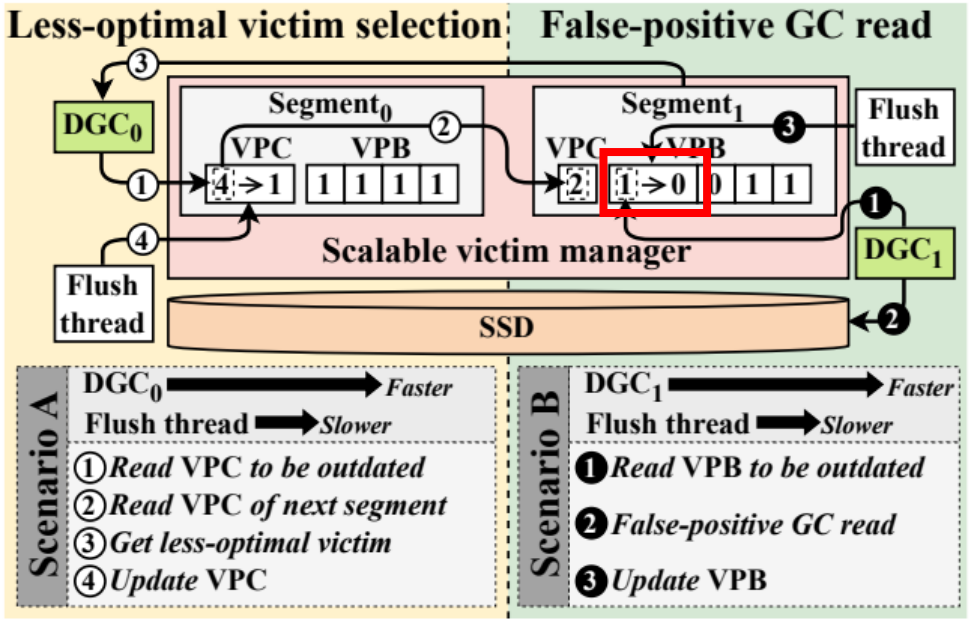
Scalable Victim Manager (3)

❖Less-optimal victim selection

- Scenario
  - VPC decreases during victim selection
  - Update is not aware to DGC (**outdated VPC**)
  - Less-optimal victim selection caused by outdated VPC
- **Negligible write amplification** observed

❖False-positive GC read

- Scenario
  - VPB is cleared during GC read
  - Update is not aware to DGC (**outdated VPB**)
  - GC read to stale page is submitted
- **Data consistency can be violated**



Data Consistency with LSU

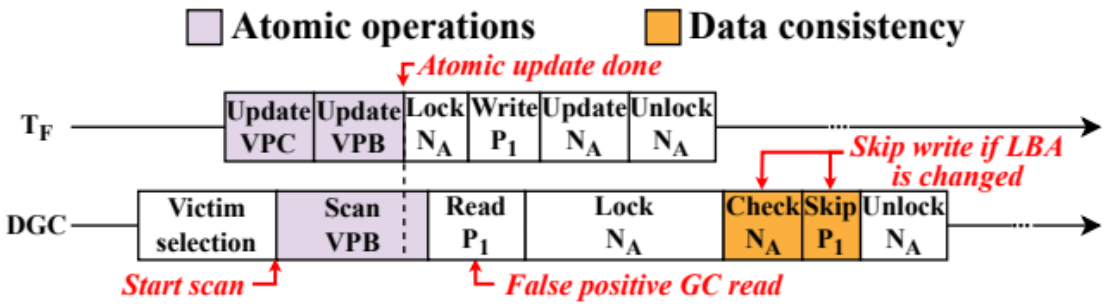
- **Data from stale page (false-positive GC read)**
  - Can induce overwriting to valid page

Data Consistency with LSU

❖Data from stale page (false-positive GC read)

- Can induce overwriting to valid page
- Solution
  - Utilize the protection from **existing node-level lock**
  - Write GC pages only if inode includes same block address with GC read address
  - Skip writing GC pages, otherwise

- No additional overheads for data consistency
  - **Naturally protected by existing node-level lock**

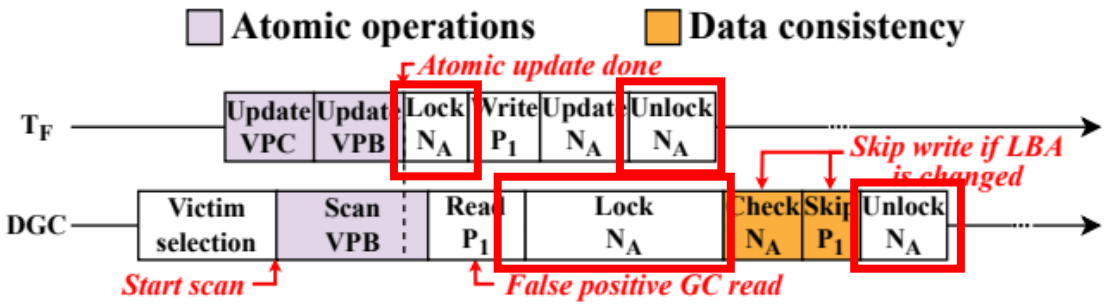


Data Consistency with LSU

❖Data from stale page (false-positive GC read)

- Can induce overwriting to valid page
- Solution
  - Utilize the protection from **existing node-level lock**
  - Write GC pages only if inode includes same block address with GC read address
  - Skip writing GC pages, otherwise

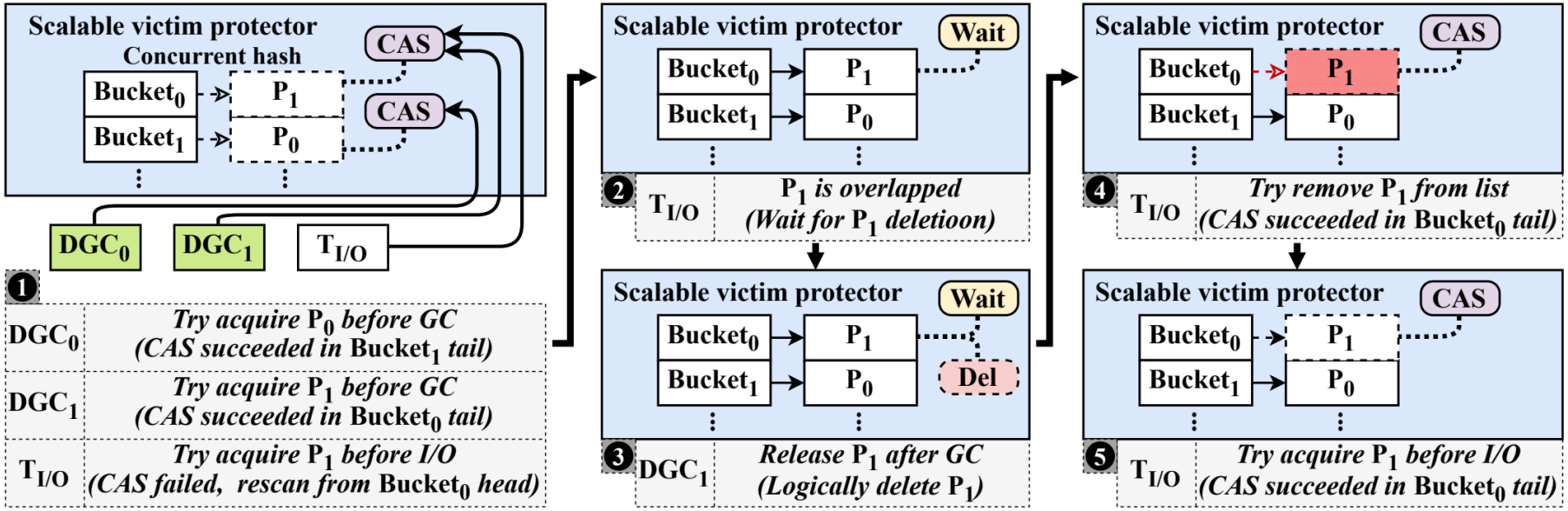
- No additional overheads for data consistency
  - **Naturally protected by existing node-level lock**



Scalable victim protector (SVP)

❖Per-file concurrent hash table

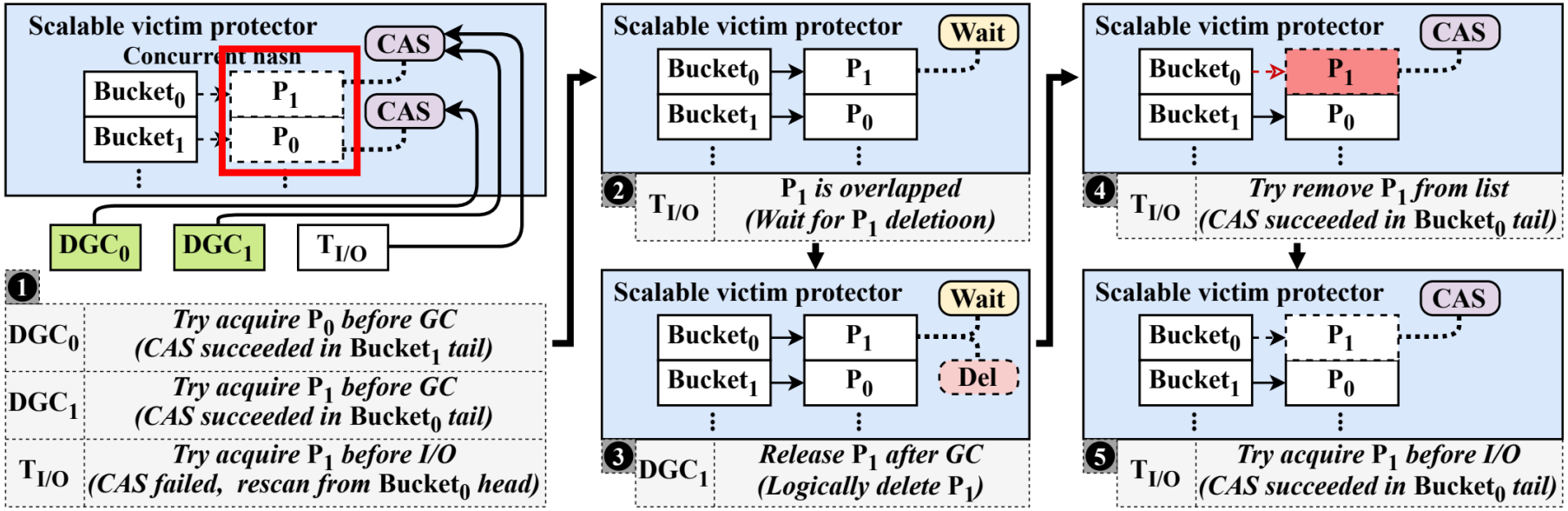
- **Concurrently insert/remove** page before/after access
  - Wait for removal of conflict page before insert
- Page offset-based hash bucket selection to reduce contention



Scalable victim protector (SVP)

❖Per-file concurrent hash table

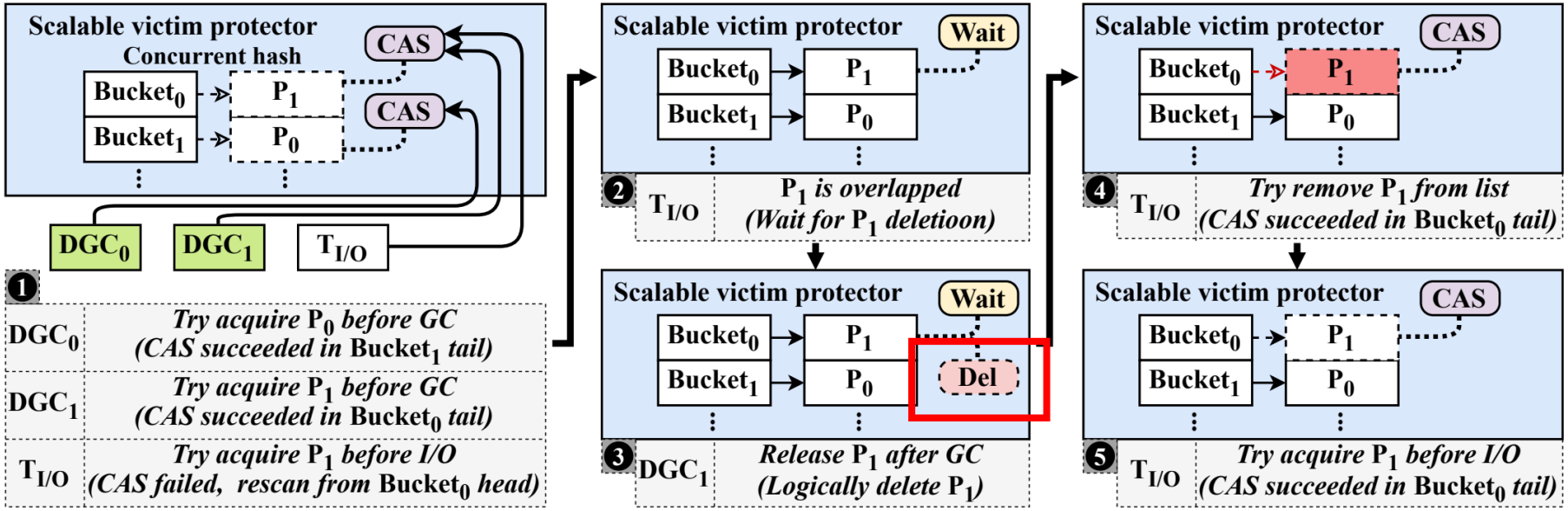
- **Concurrently insert/remove** page before/after access
  - Wait for removal of conflict page before insert
- Page offset-based hash bucket selection to reduce contention



Scalable victim protector (SVP)

❖Per-file concurrent hash table

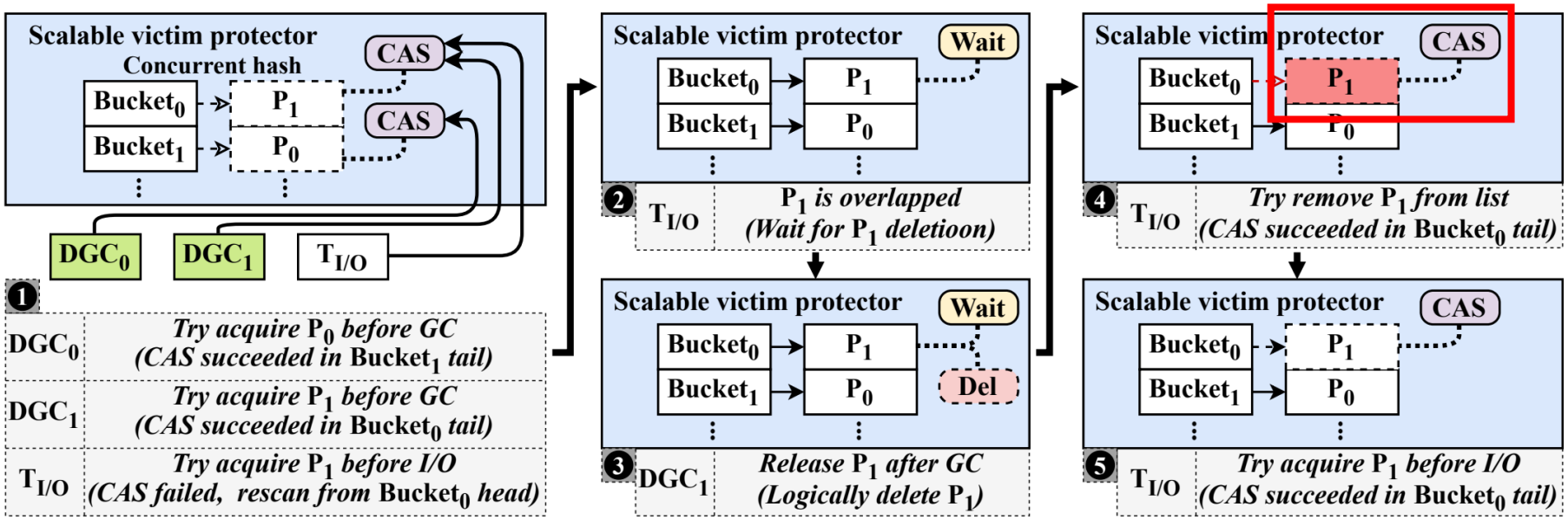
- **Concurrently insert/remove** page before/after access
  - Wait for removal of conflict page before insert
- Page offset-based hash bucket selection to reduce contention



Scalable victim protector (SVP)

❖Per-file concurrent hash table

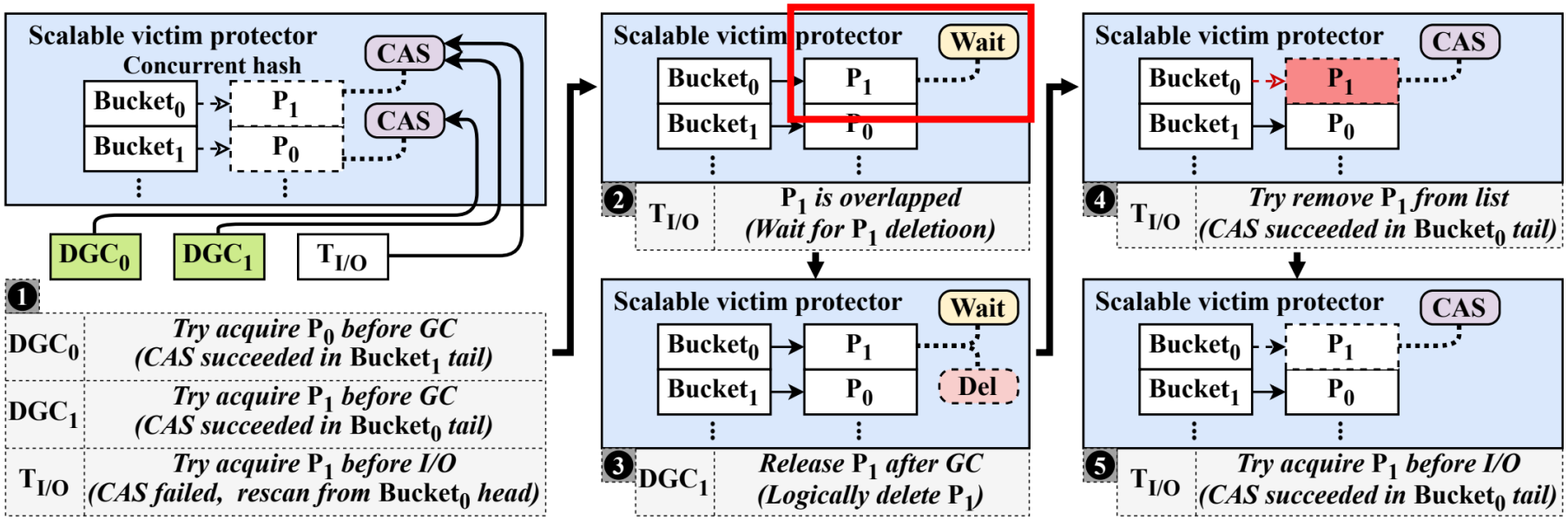
- **Concurrently insert/remove** page before/after access
  - Wait for removal of conflict page before insert
- Page offset-based hash bucket selection to reduce contention



Scalable victim protector (SVP)

❖Per-file concurrent hash table

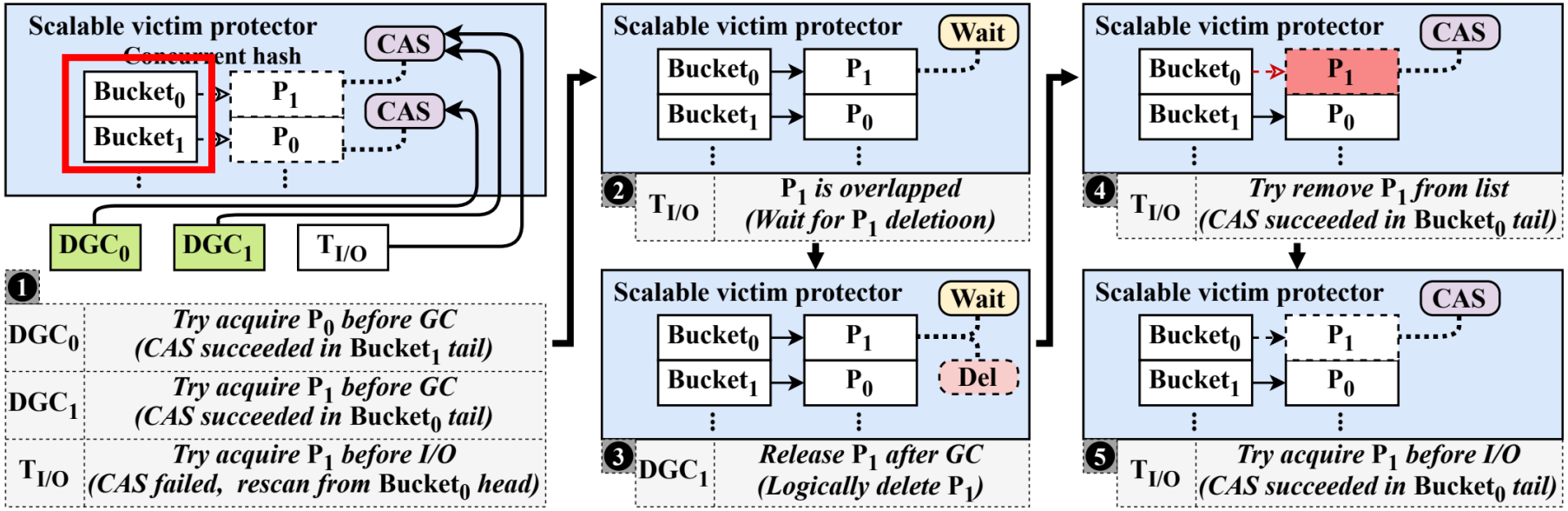
- **Concurrently insert/remove** page before/after access
  - Wait for removal of conflict page before insert
- Page offset-based hash bucket selection to reduce contention



Scalable victim protector (SVP)

❖Per-file concurrent hash table

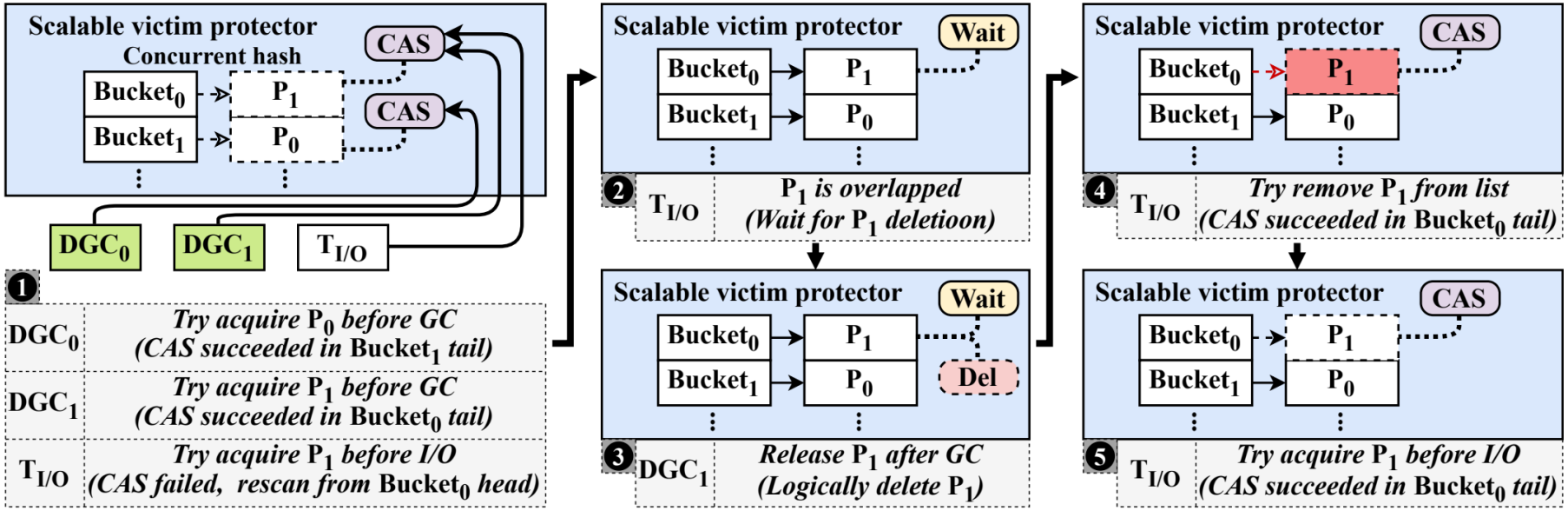
- **Concurrently insert/remove** page before/after access
  - Wait for removal of conflict page before insert
- Page offset-based hash bucket selection to reduce contention



Scalable victim protector (SVP)

❖Per-file concurrent hash table

- **Concurrently insert/remove** page before/after access
  - Wait for removal of conflict page before insert
- Page offset-based hash bucket selection to reduce contention



Experimental Setup

- **Test bed**
  - Intel Xeon E5-2650 CPU with 48 logical cores
  - 160 GB DRAM
    - Limited to 8 GB for evaluation in 30 GB partition
  - Samsung 9A3 SSD 7.68 TB
  - Ubuntu 22.04 LTS with kernel 6.0.0

Experimental Setup

❖Test bed

- Intel Xeon E5-2650 CPU with 48 logical cores
- 160 GB DRAM
  - Limited to 8 GB for evaluation in 30 GB partition
- Samsung 9A3 SSD 7.68 TB
- Ubuntu 22.04 LTS with kernel 6.0.0

❖Workloads

- Micro-benchmark (FIO)
  - 4 KB random writes
- Macro-benchmark (Filebench)
  - Fileserver, varmail, OLTP
- Real Application (MySQL with YCSB)
  - YCSB A, B

Experimental Setup

❖Test bed

- Intel Xeon E5-2650 CPU with 48 logical cores
- 160 GB DRAM
  - Limited to 8 GB for evaluation in 30 GB partition
- Samsung 9A3 SSD 7.68 TB
- Ubuntu 22.04 LTS with kernel 6.0.0

❖Workloads

- Micro-benchmark (FIO)
  - 4 KB random writes
- Macro-benchmark (Filebench)
  - Fileserver, varmail, OLTP
- Real Application (MySQL with YCSB)
  - YCSB A, B

❖Comparison

- F2FS
  - LFS in Linux kernel
- F2FS-L
  - F2FS in pure LFS mode (no SSR, in-place update)
- MAX
  - State-of-the-art scalable LFS
- P-GC
  - LFS with parallel GC (LFS mode)
- **ScaleLFS**
  - Proposed idea implemented based on F2FS (LFS mode)

Micro-benchmark (1)

❖Evaluation on small partition

- Using 30 GB partition
  - **Exclude SSD-level GC** and show **pure file system performance**
  - 120GB of 4KB random writes

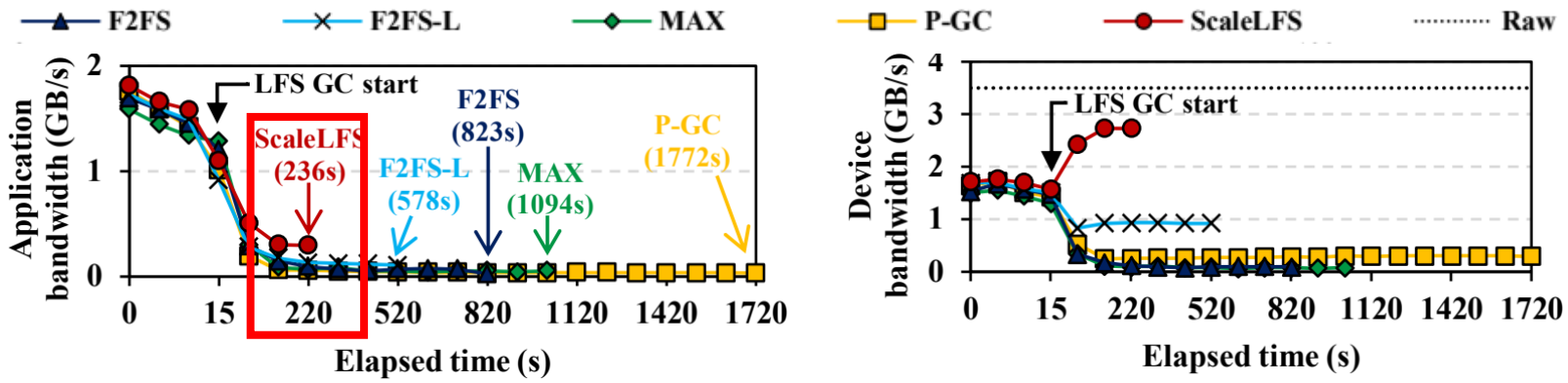
Micro-benchmark (1)

❖Evaluation on small partition

- Using 30 GB partition
  - **Exclude SSD-level GC** and show **pure file system performance**
  - 120GB of 4KB random writes

❖Improved execution time

- Outperform existing LFSs by up to **7.0x**



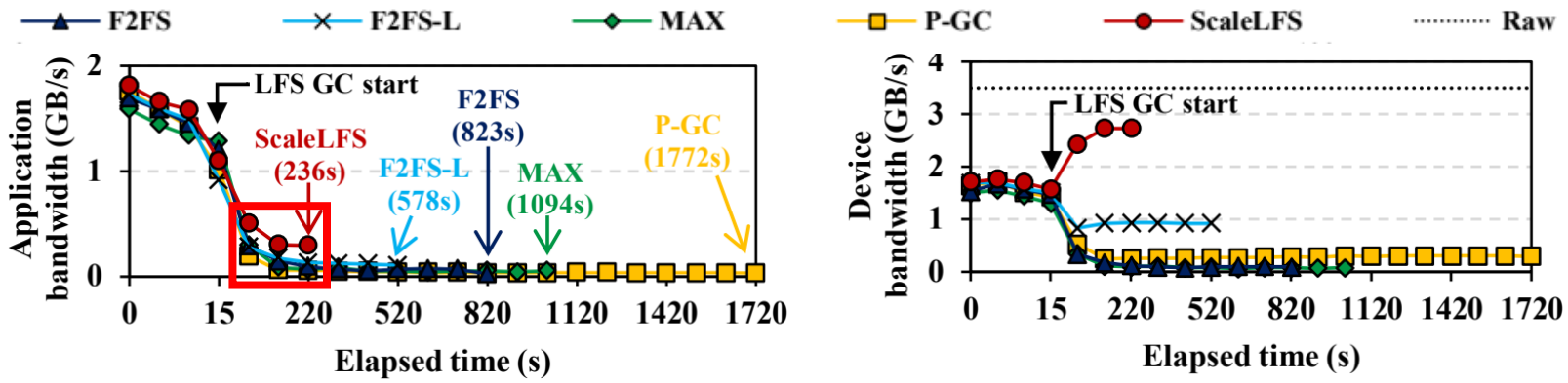
Micro-benchmark (1)

❖Evaluation on small partition

- Using 30 GB partition
  - **Exclude SSD-level GC** and show **pure file system performance**
  - 120GB of 4KB random writes

❖Improved execution time

- Outperform existing LFSs by up to **7.0x**



Micro-benchmark (1)

❖Evaluation on small partition

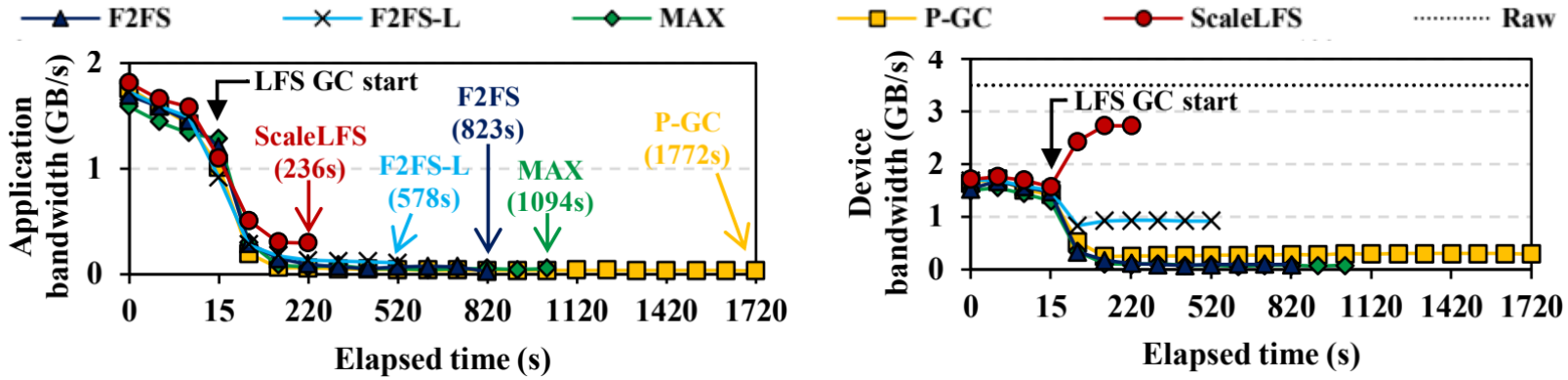
- Using 30 GB partition
  - **Exclude SSD-level GC** and show **pure file system performance**
  - 120GB of 4KB random writes

❖Improved execution time

- Outperform existing LFSs by up to **7.0x**

❖Efficiently utilized SSD bandwidth

- Utilize up to **19.6x** SSD bandwidth compared to existing LFSs
- Reach **near raw SSD bandwidth limit** (2.9GB/s and 3.4 GB/s)



Micro-benchmark (1)

❖Evaluation on small partition

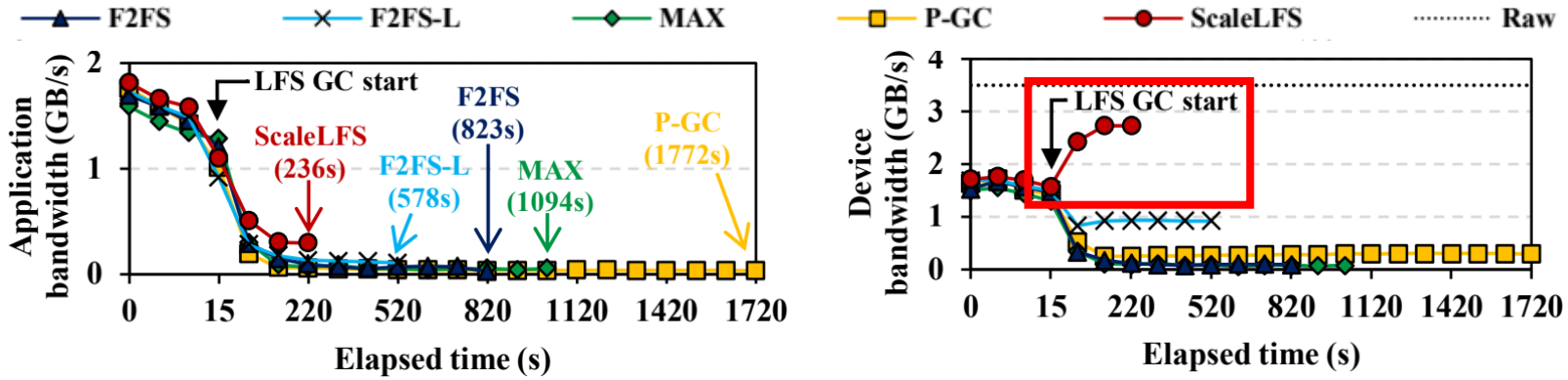
- Using 30 GB partition
  - **Exclude SSD-level GC** and show **pure file system performance**
  - 120GB of 4KB random writes

❖Improved execution time

- Outperform existing LFSs by up to **7.0x**

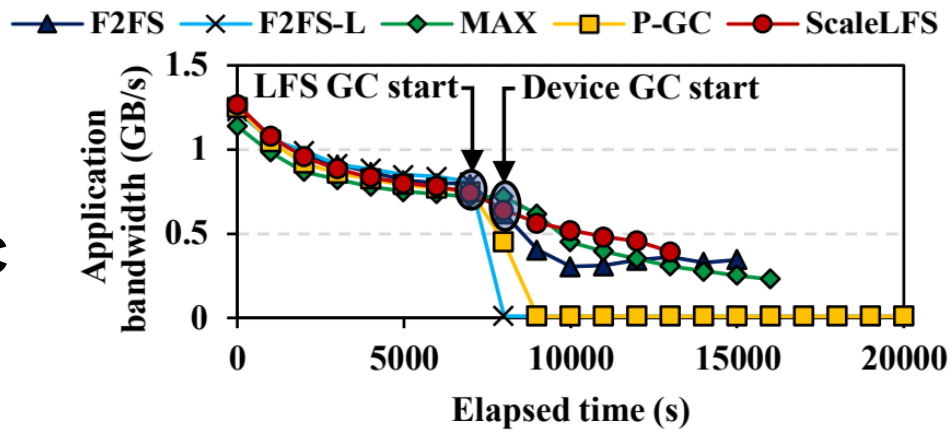
❖Efficiently utilized SSD bandwidth

- Utilize up to **19.6x** SSD bandwidth compared to existing LFSs
- Reach **near raw SSD bandwidth limit** (2.9GB/s and 3.4 GB/s)



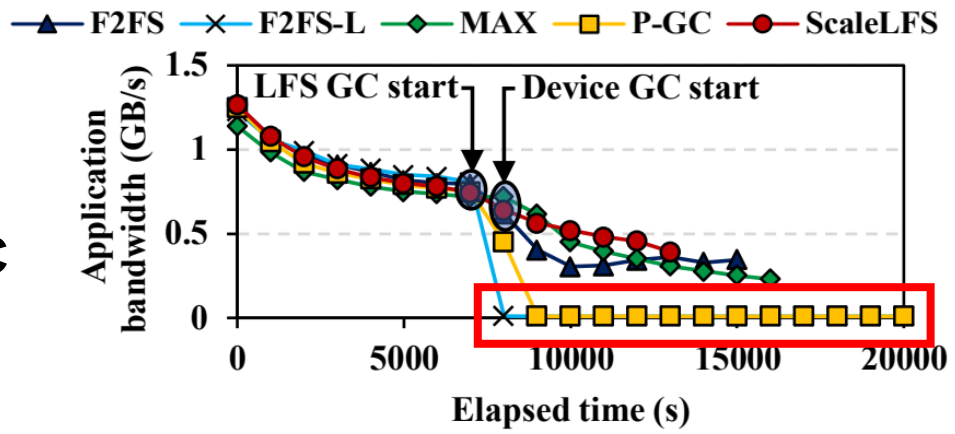
Micro-benchmark (2)

- **Evaluation on full device partition**
  - Using 7.68 TB partition
    - 10.3 TB of random writes
    - Show **practical performance with SSD level GC**
- **Improved execution time**
  - Improve **37 minutes** compared to F2FS



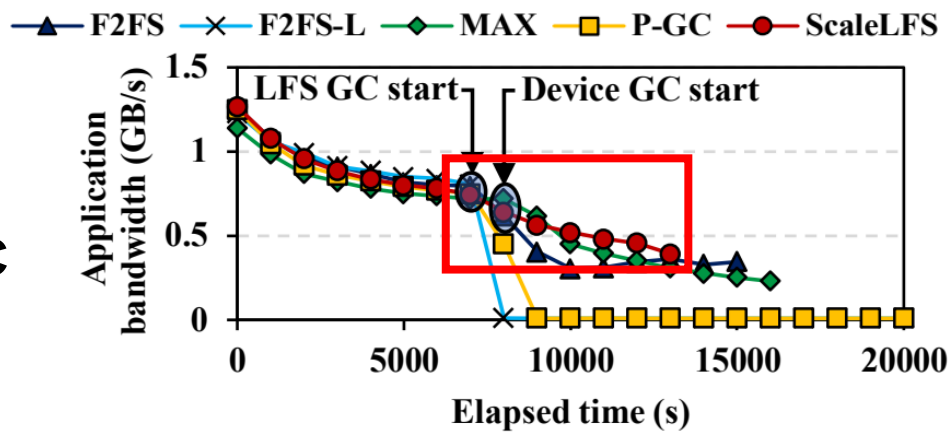
Micro-benchmark (2)

- **Evaluation on full device partition**
  - Using 7.68 TB partition
    - 10.3 TB of random writes
    - Show **practical performance with SSD level GC**
- **Improved execution time**
  - Improve **37 minutes** compared to F2FS
- **Performance of pure LFS mode (F2FS-L, P-GC, and ScaleLFS)**
  - F2FS-L and P-GC show **11 MB/s** after both LFS GC and SSD GC have started
  - ScaleLFS shows the **highest performance** without significant performance degradation



Micro-benchmark (2)

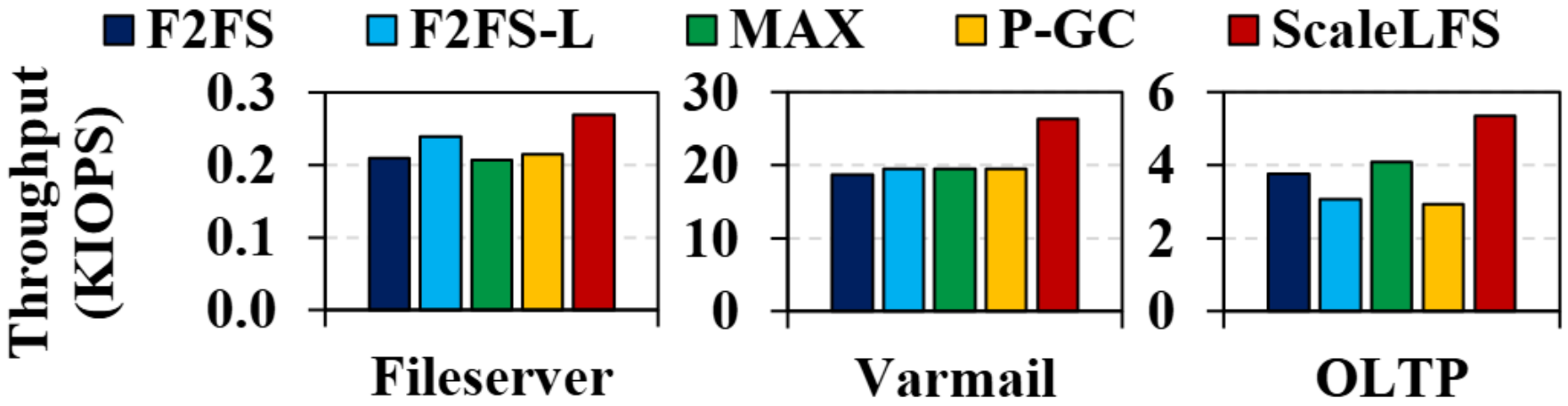
- **Evaluation on full device partition**
  - Using 7.68 TB partition
    - 10.3 TB of random writes
    - Show **practical performance with SSD level GC**
- **Improved execution time**
  - Improve **37 minutes** compared to F2FS
- **Performance of pure LFS mode (F2FS-L, P-GC, and ScaleLFS)**
  - F2FS-L and P-GC show **11 MB/s** after both LFS GC and SSD GC have started
  - ScaleLFS shows the **highest performance** without significant performance degradation



Macro-benchmark

❖**Evaluation on three write-intensive workloads**

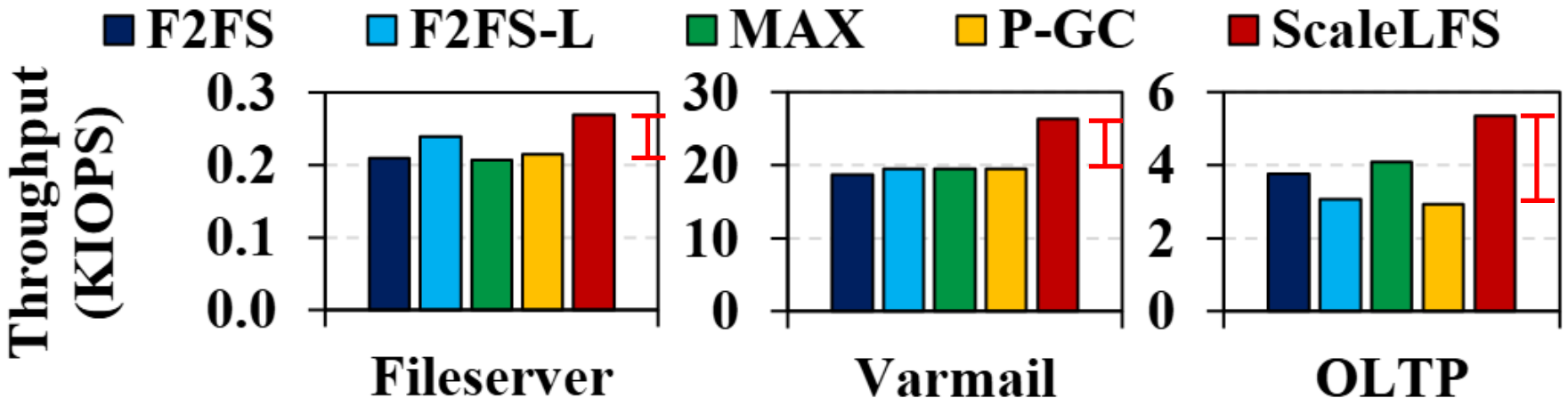
- Improve throughput **up to 83.1%**
- Less improvement than FIO
  - **Frequent deletion, resulting in infrequent GC**
  - **Contention among I/O threads, resulting in less stress to LFS**



Macro-benchmark

❖**Evaluation on three write-intensive workloads**

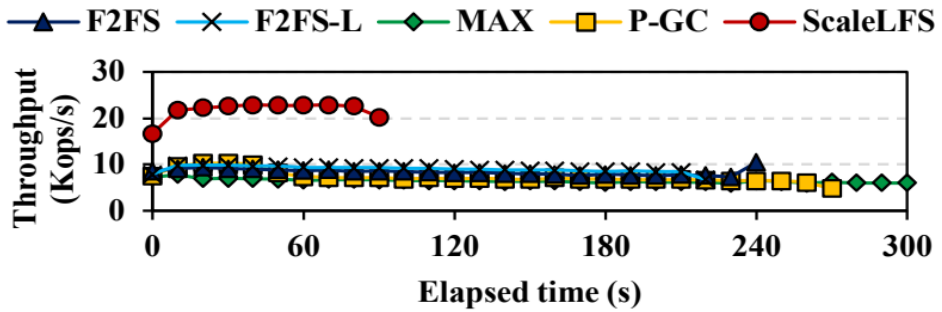
- Improve throughput **up to 83.1%**
- Less improvement than FIO
  - **Frequent deletion, resulting in infrequent GC**
  - **Contention among I/O threads, resulting in less stress to LFS**



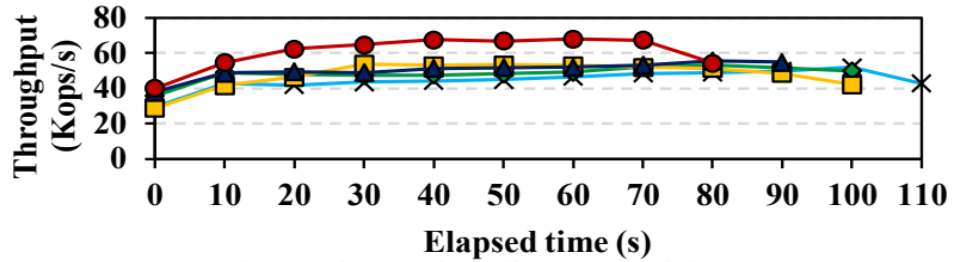
Real Application

❖MySQL with YCSB benchmark

- Initially 6.5 million records are inserted
  - 70 % of total capacity occupied
- Workload A (Write intensive)
  - Update 50%, read 50%
- Workload B (Read intensive)
  - Update 5%, read 95%



Throughput timeline (Workload A).



Throughput timeline (Workload B).

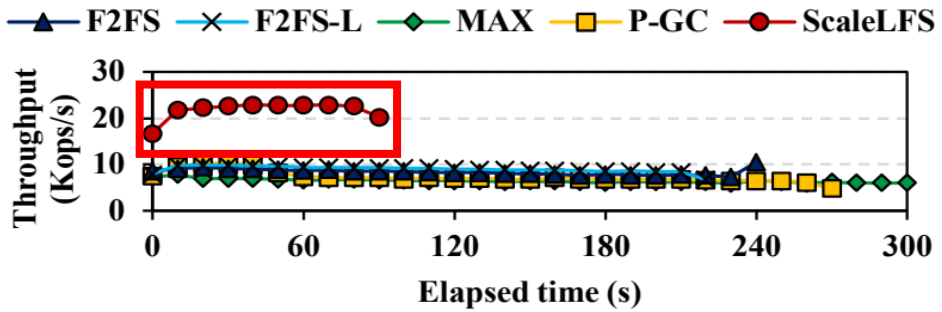
❖Execution time

- Decrease by up to **70.4%** in Workload A
- Decrease by up to **27.3%** in Workload B
- ScaleLFS improves performance even under the read-intensive workload

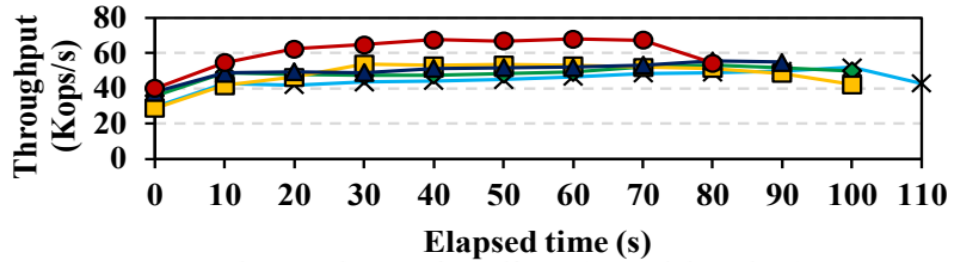
Real Application

❖MySQL with YCSB benchmark

- Initially 6.5 million records are inserted
  - 70 % of total capacity occupied
- Workload A (Write intensive)
  - Update 50%, read 50%
- Workload B (Read intensive)
  - Update 5%, read 95%



Throughput timeline (Workload A).



Throughput timeline (Workload B).

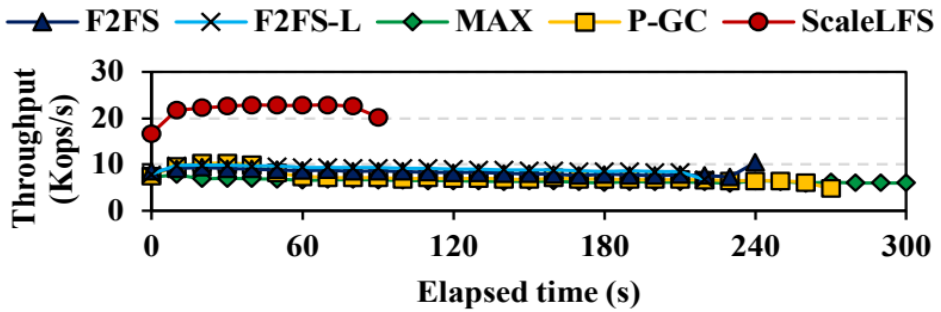
❖Execution time

- Decrease by up to **70.4%** in Workload A
- Decrease by up to **27.3%** in Workload B
- ScaleLFS improves performance even under the read-intensive workload

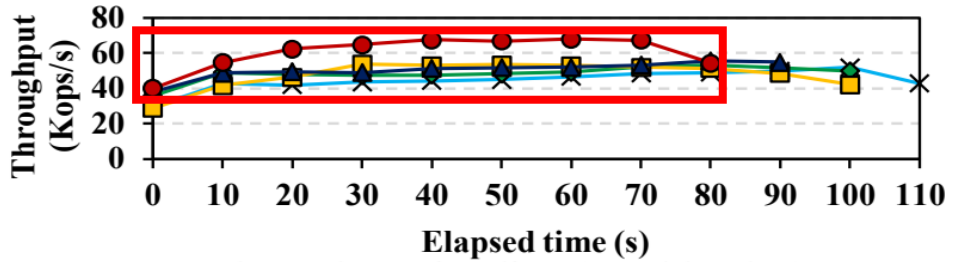
Real Application

❖MySQL with YCSB benchmark

- Initially 6.5 million records are inserted
  - 70 % of total capacity occupied
- Workload A (Write intensive)
  - Update 50%, read 50%
- Workload B (Read intensive)
  - Update 5%, read 95%



Throughput timeline (Workload A).



Throughput timeline (Workload B).

❖Execution time

- Decrease by up to **70.4%** in Workload A
- Decrease by up to **27.3%** in Workload B
- ScaleLFS improves performance even under the read-intensive workload

Conclusion

- **ScaleLFS accelerate LFS-level GC with multi-thread friendly designs**
  - Utilize multiple GC threads with **dedicated strategy** on GC segments and page buffers
  - Increase concurrency of GC with **concurrent data structures**, **lock-free victim management** policies and **loose-synchronization** for GC metadata
  - Enable a **page-level GC procedure** with scalable while resolving the conflict with victim pages
  - ScaleLFS increases LFS performance by up to **7.0x** compared to existing LFSs

Thank you