RX-LFS: Rethinking the Bottleneck of LFS and Curbing the Hidden Cost of Slack Space Recycling

INWHI HWANG, JIN YONG HA, HYEONSANG EOM, YONGSEOK SON

NVRAMOS 2026







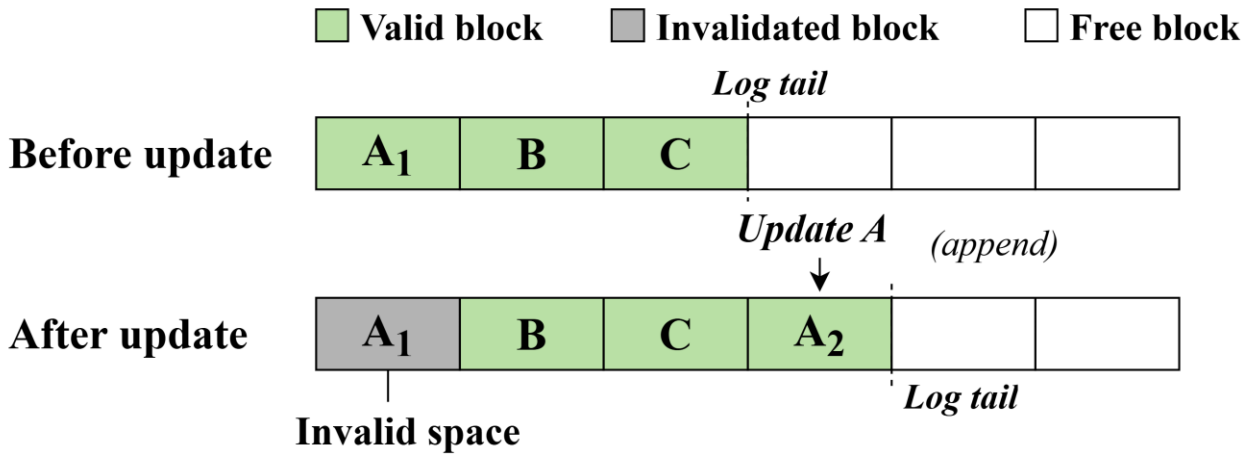
Log-Structured File System (LFS)

Append-only write nature

- Write sequentially (Good for SSDs)
- But requires out-of-place updates

Out-of-place update

- Append new data and invalidate old data
- **Invalid space accumulates**



Recycling Invalid Space

Two ways to recycle invalid space

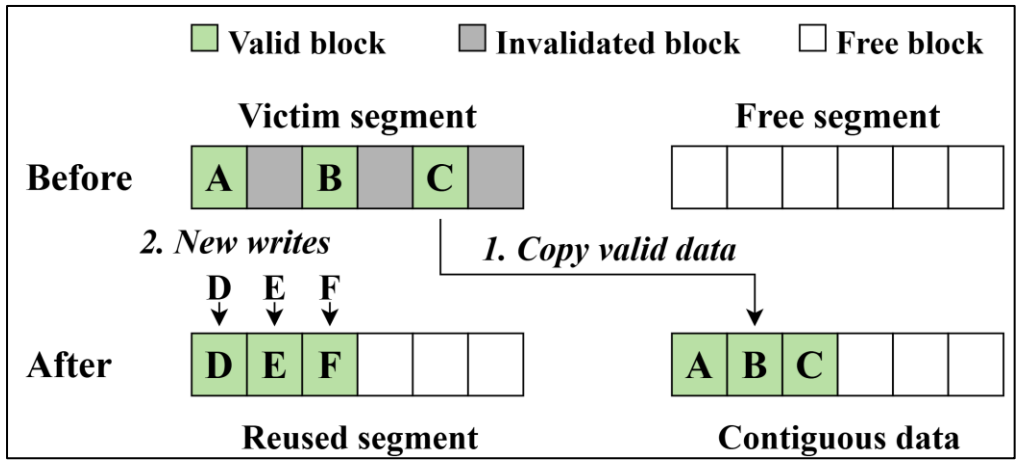
- Garbage collection (GC)
- Slack space recycling (SSR)

Recycling Invalid Space

Two ways to recycle invalid space

- Garbage collection (GC)
- Slack space recycling (SSR)

Space recycling via GC

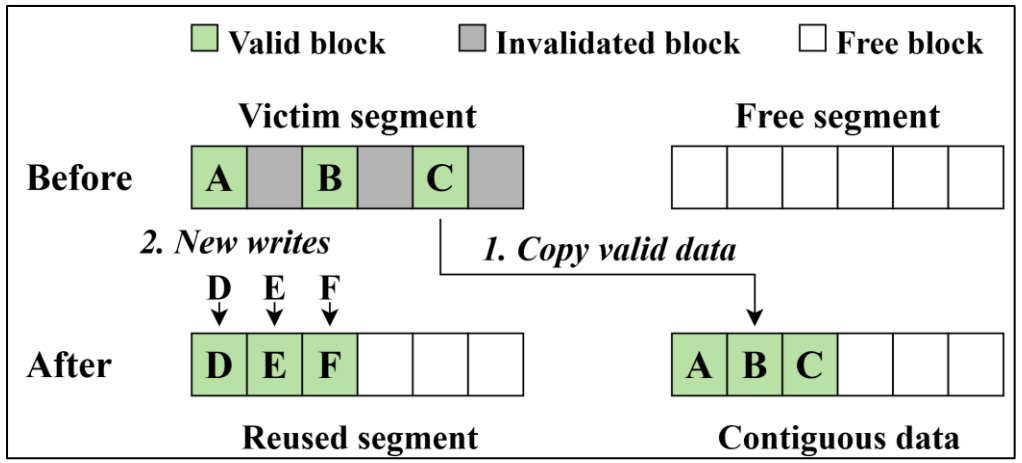


Recycling Invalid Space

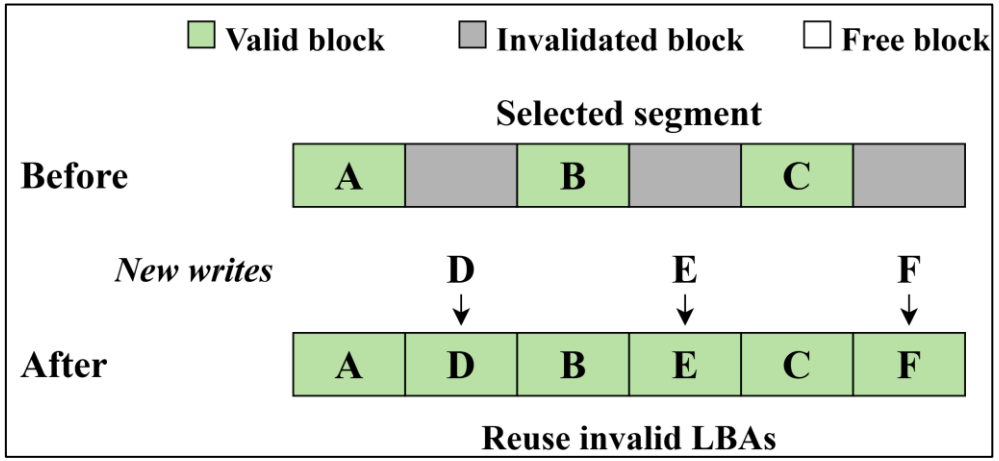
Two ways to recycle invalid space

- Garbage collection (GC)
- Slack space recycling (SSR)

Space recycling via GC



Space recycling via SSR

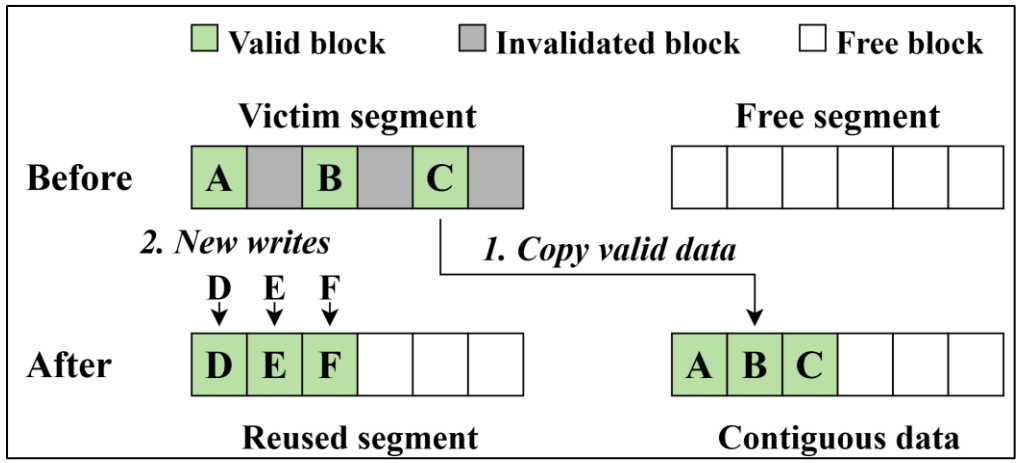


GC or SSR?

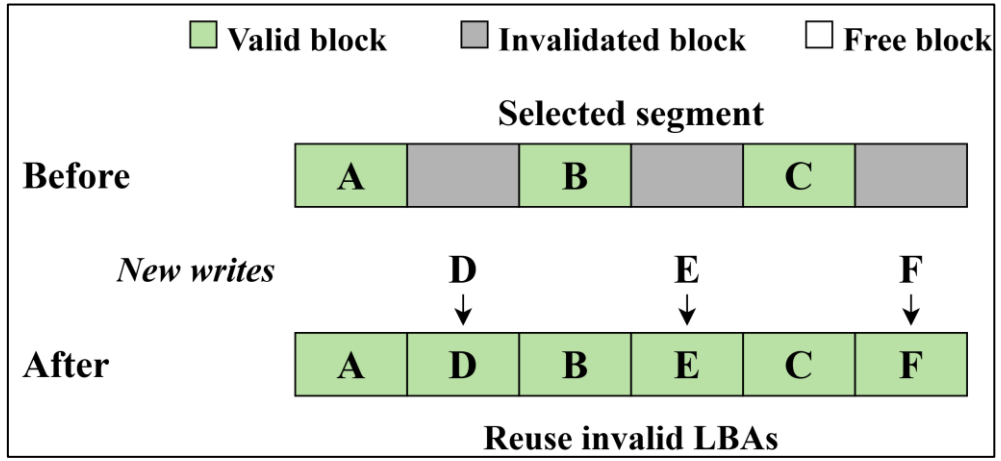
Comparison

GC	Metric	SSR
Large sequential	**Write pattern**	Small random
Small random	**Read pattern**	**No read**
Valid data copy	**File system WAF**	**No file system WAF**

Space recycling via GC



Space recycling via SSR

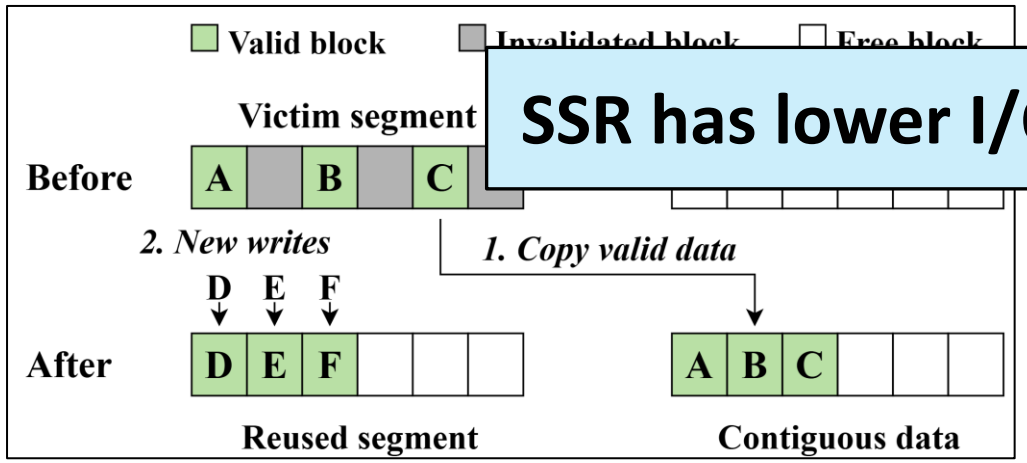


GC or SSR?

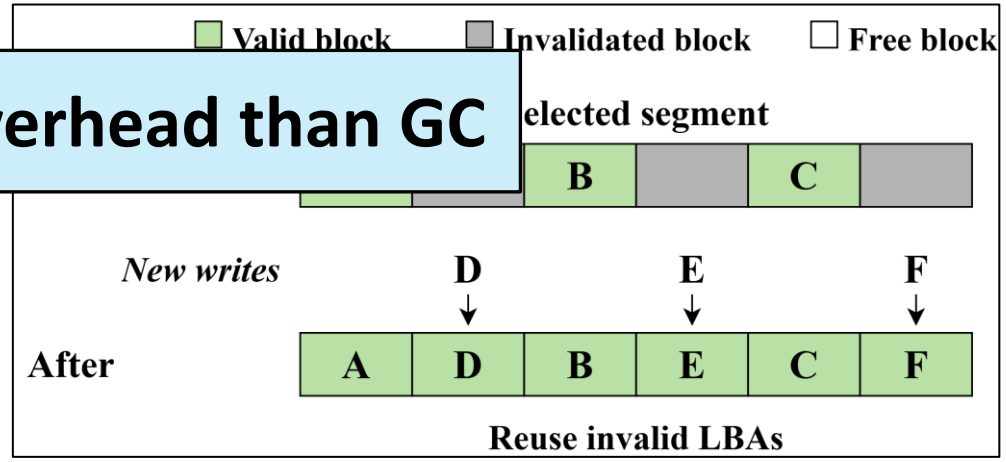
Comparison

GC	Metric	SSR
Large sequential	**Write pattern**	Small random
Small random	**Read pattern**	**No read**
Valid data copy	**File system WAF**	**No file system WAF**

Space recycling via GC



Space recycling via SSR



SSR has lower I/O overhead than GC

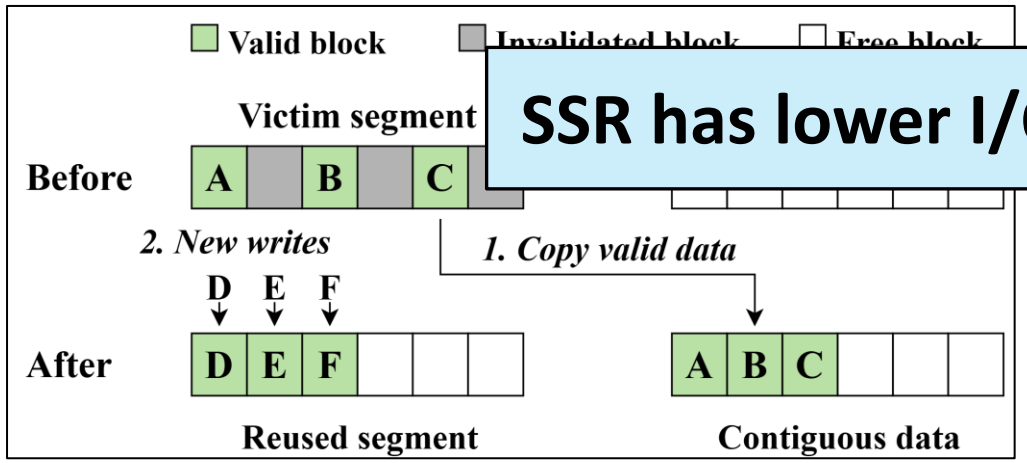
GC or SSR?

Comparison

But random writes break the SSD-friendly sequential pattern of LFS

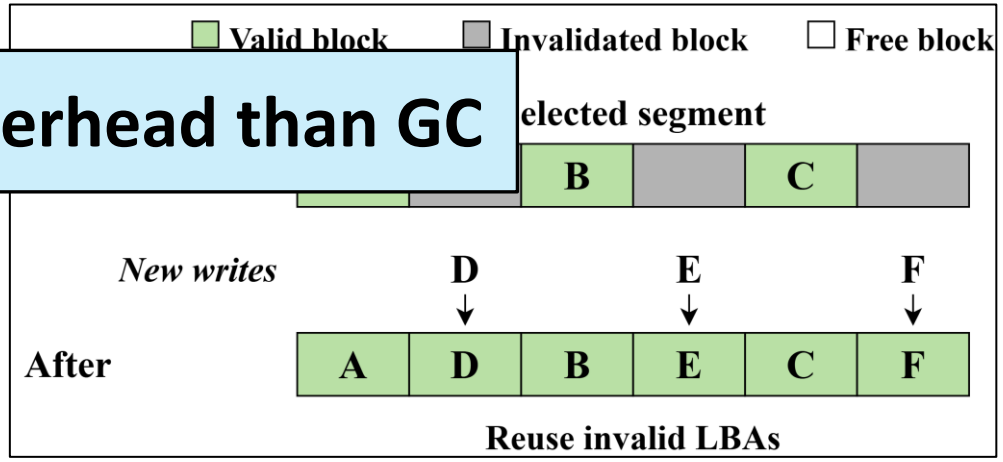
GC	Metric	SSR
Large sequential	Write pattern	Small random
Small random	Read pattern	No read
Valid data copy	File system WAF	No file system WAF

Space recycling via GC



SSR has lower I/O overhead than GC

Space recycling via SSR



GC: The Focus of Most Research

Long-standing problem for sustainable performance

- GC has already been explored in many works
- However, SSR has not been explored enough



Subject	Research
Scalability	ScaleLFS (FAST '25)
Hardware	IPLFS (ATC '22), D2FS (FAST '25)
WAF	MIDAS (FAST '24), DOGI (FAST '26)
Scheduling	ZUFS (FAST '26)

GC: The Focus of Most Research

Long-standing problem for sustainable performance

- GC has already been explored in many works
- However, SSR has not been explored enough



Subject	Research
Scalability	ScaleLFS (FAST '25)
Hardware	IPLFS (ATC '22), D2FS (FAST '25)
WAF	MIDAS (FAST '24), DOGI (FAST '26)
Scheduling	ZUFS (FAST '26)

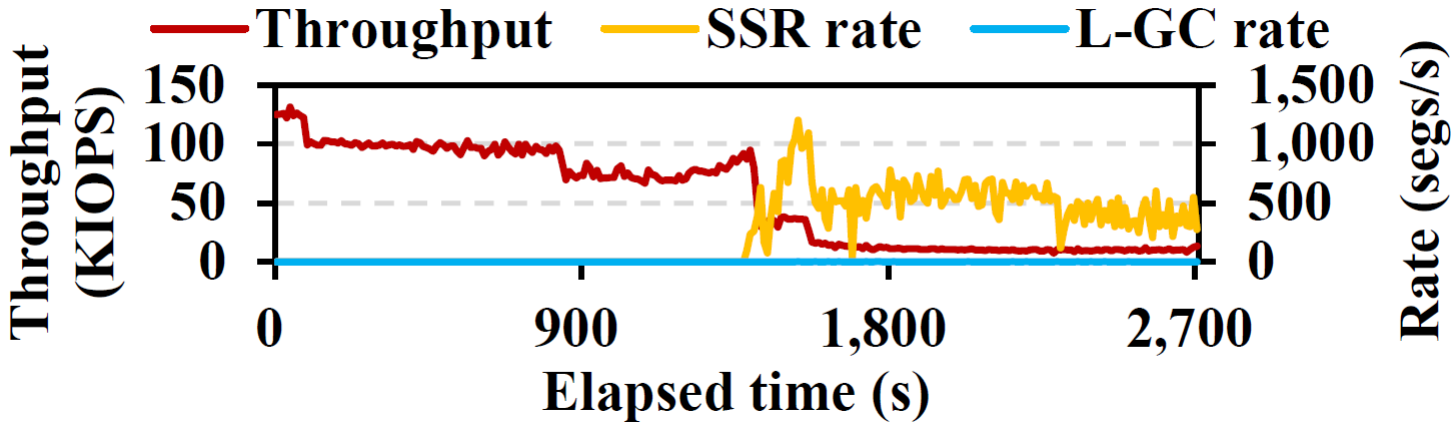
Little attention so far

Real Performance Bottleneck of F2FS*

Space reclamation is mostly SSR, not GC

- When SSR occurs, throughput drops sharply

** F2FS: widely used LFS with both GC and SSR*



Throughput
(4 KB random write)

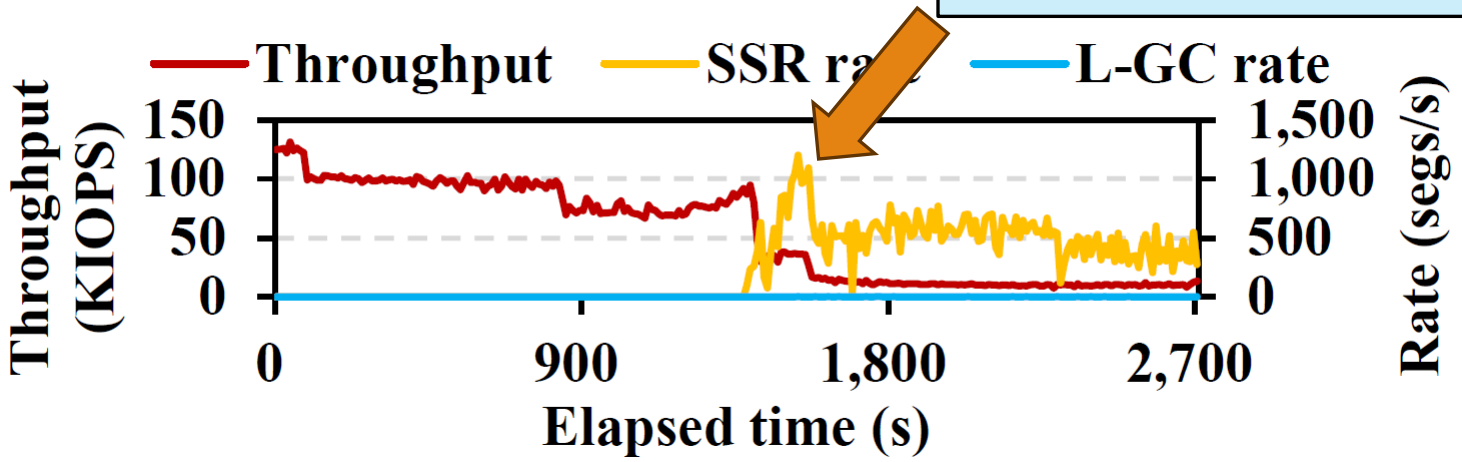
Real Performance Bottleneck of F2FS*

Space reclamation is mostly SSR, not GC

- When SSR occurs, throughput drops sharply

** F2FS: widely used LFS with both GC and SSR*

SSR is the real bottleneck of sustainable performance in F2FS



Throughput (4 KB random write)

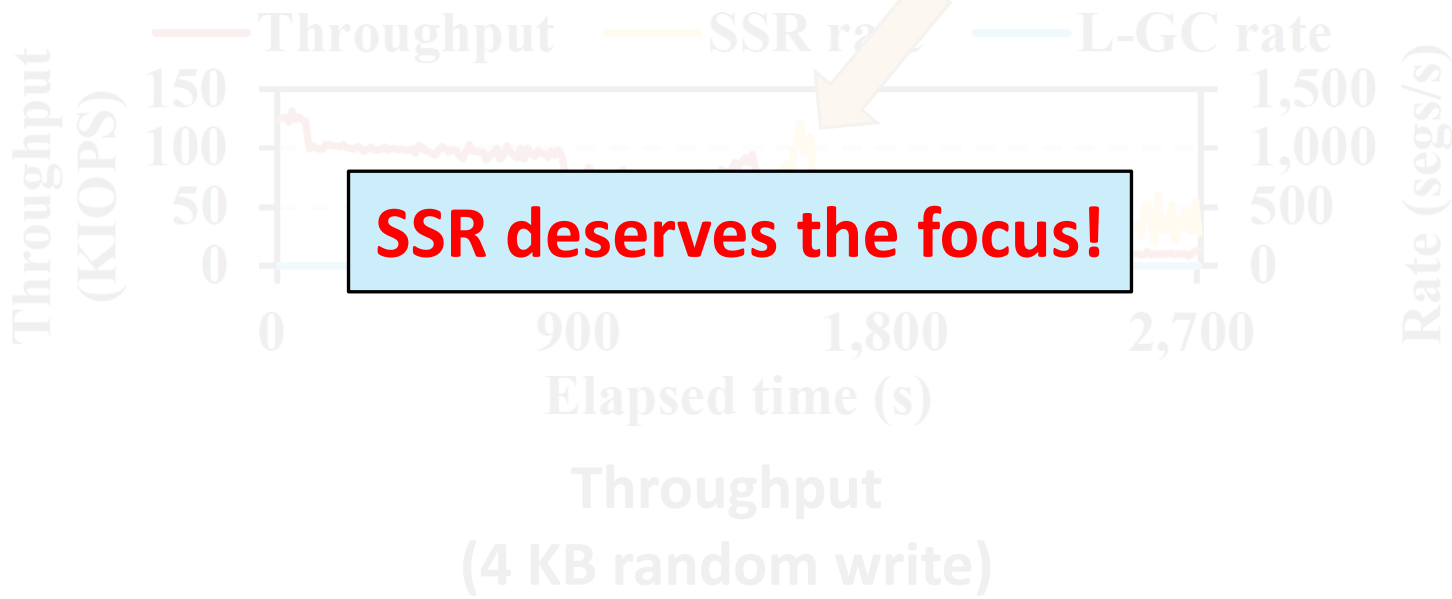
Real Performance Bottleneck of F2FS*

Space reclamation is mostly SSR, not GC

- When SSR occurs, throughput drops sharply

** F2FS: widely used LFS with both GC and SSR*

SSR is the real bottleneck of sustainable performance in F2FS



SSR deserves the focus!

Why Performance Drops during SSR

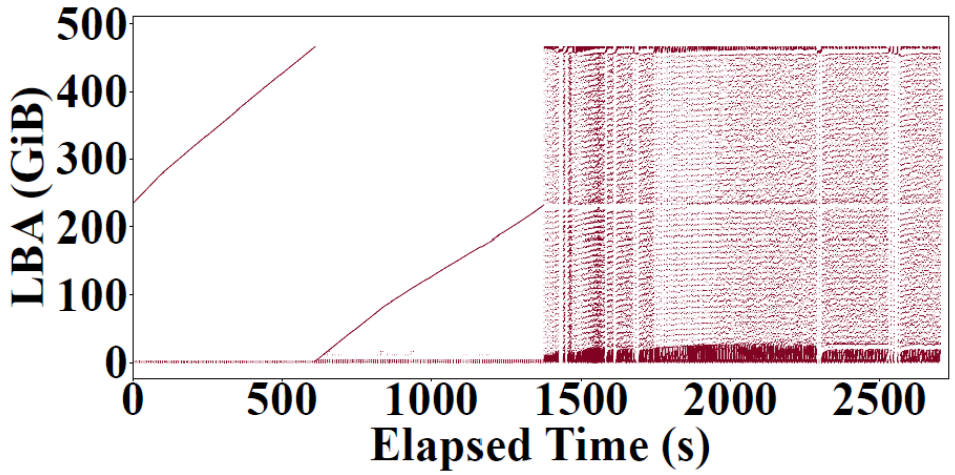
Sequential write pattern

- Unsustainable (Not guaranteed in SSR)

Random write pattern

- SSD-unfriendly (more SSD GC → higher SSD WAF)

Write pattern from F2FS (4KB random write)



Why Performance Drops during SSR

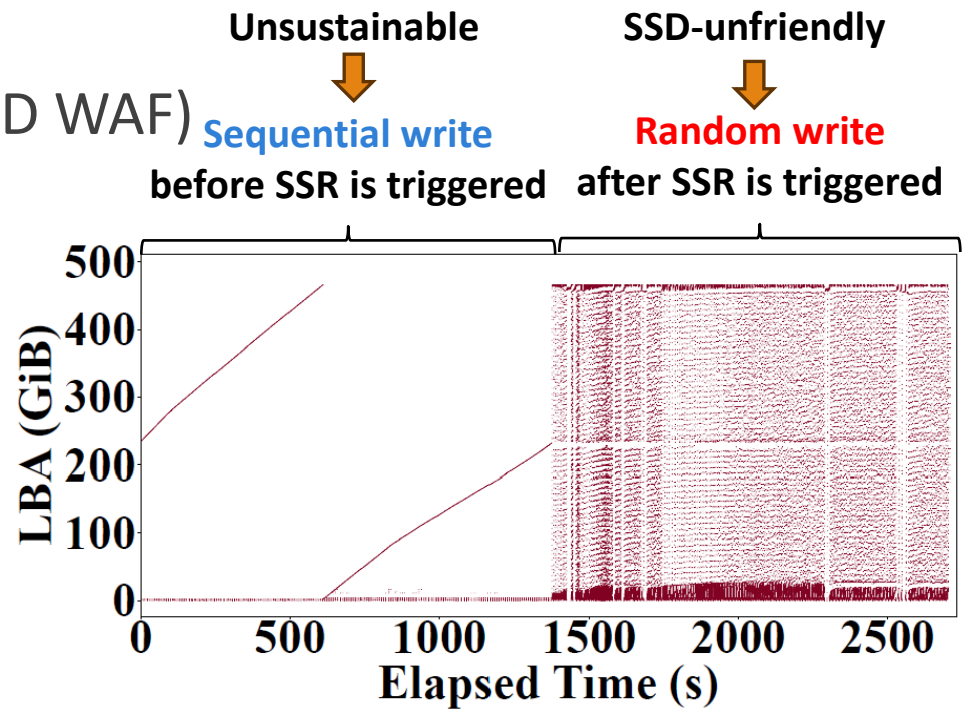
Sequential write pattern

- Unsustainable (Not guaranteed in SSR)

Random write pattern

- SSD-unfriendly (more SSD GC → higher SSD WAF)

Write pattern from F2FS (4KB random write)



Why Performance Drops during SSR

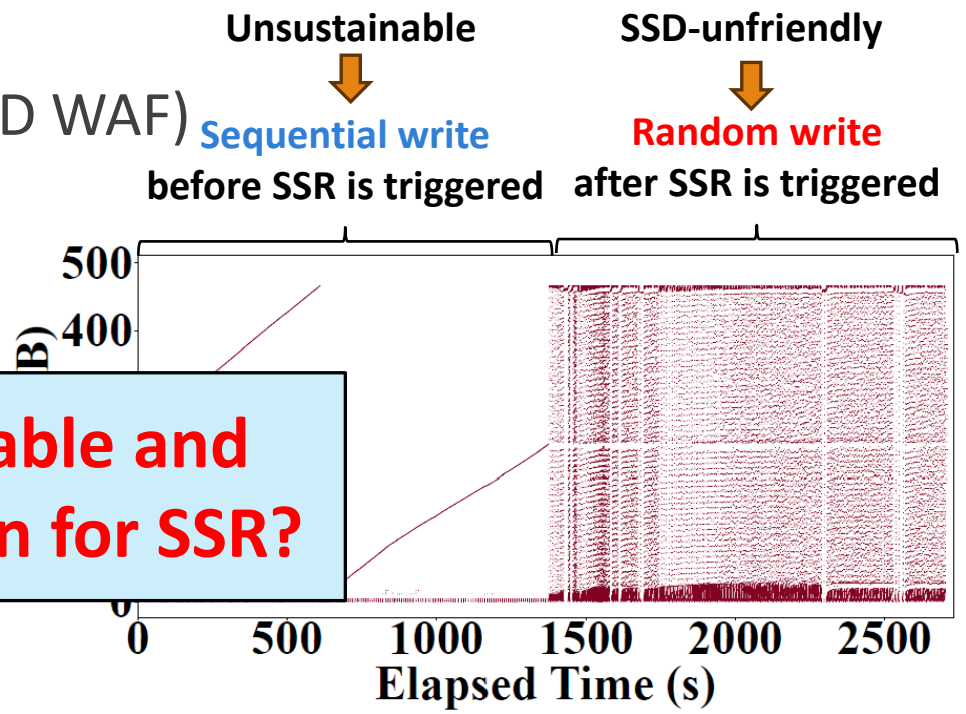
Sequential write pattern

- Unsustainable (Not guaranteed in SSR)

Random write pattern

- SSD-unfriendly (more SSD GC → higher SSD WAF)

Write pattern from F2FS (4KB rand

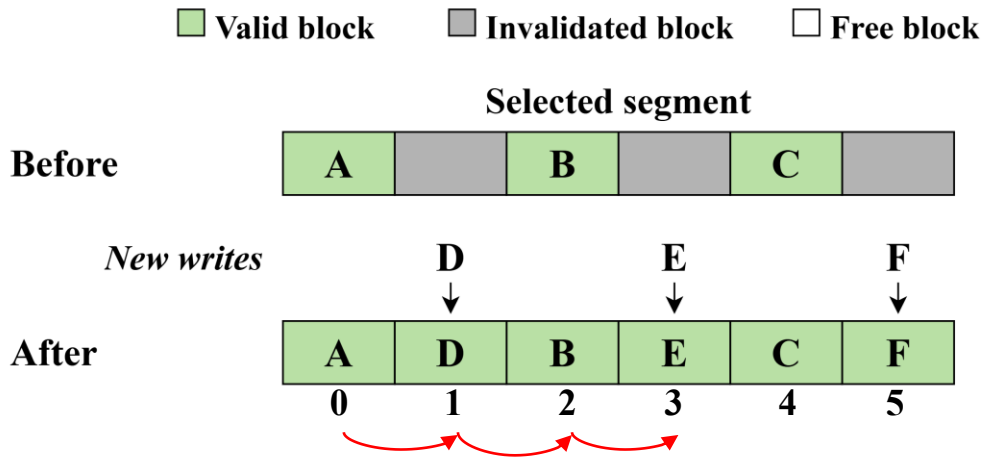


What is a sustainable and SSD-friendly pattern for SSR?

Relaxed Sequential Write Pattern (RX-S)

Typical sequential pattern

- All writes are **strictly consecutive**
- Not usable for SSR



No gaps

Addresses in typical sequential: 0 → 1 → 2

Not usable

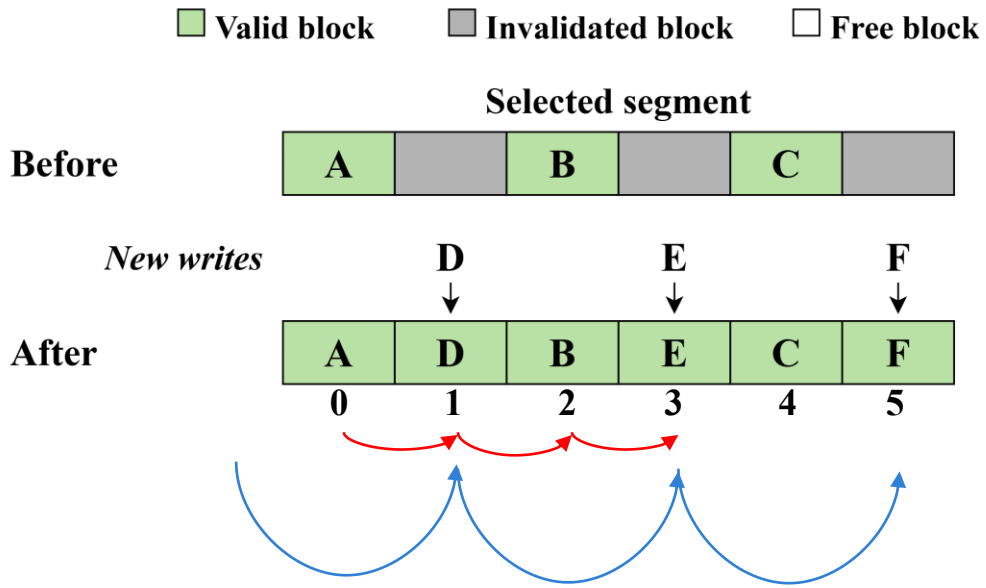
Relaxed Sequential Write Pattern (RX-S)

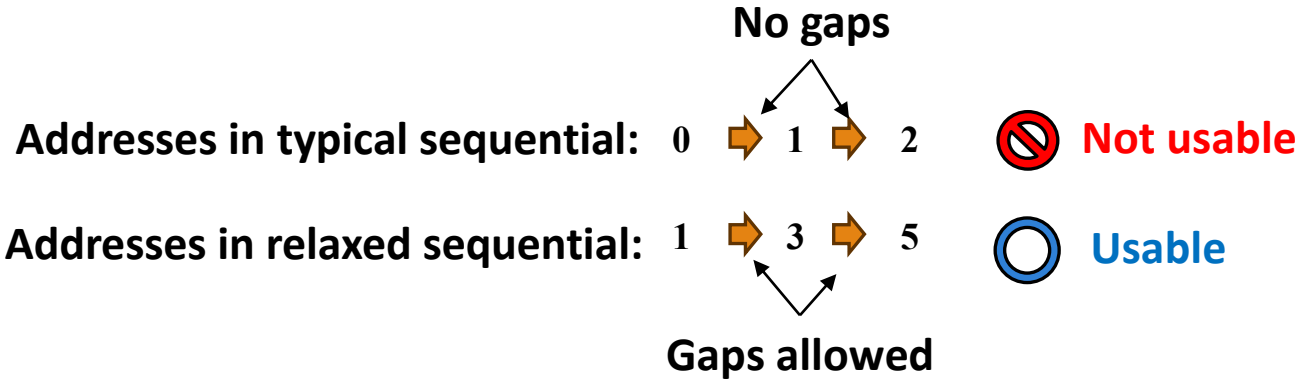
Typical sequential pattern

- All writes are **strictly consecutive**
- Not usable for SSR

Relaxed sequential pattern

- Writes in increasing LBA order, **gaps allowed**
- Usable for SSR





Effect of RX-S

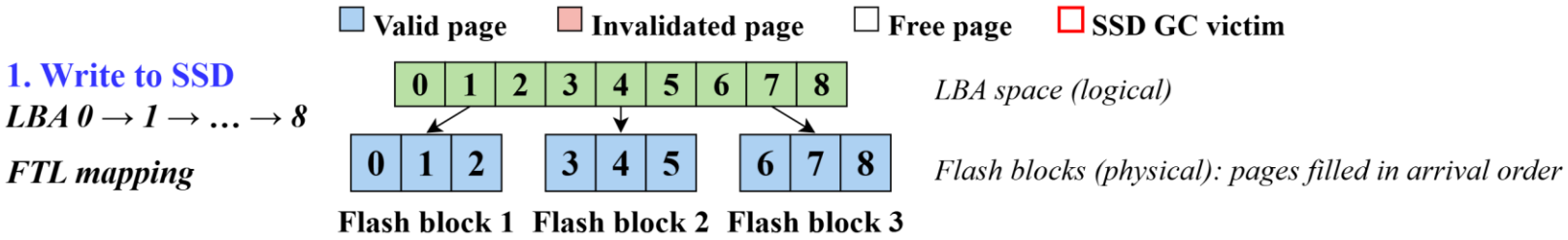
Reducing SSD WAF

- FTL puts data with nearby LBAs into consecutive flash pages
- Next RX-S pass overwrites them in the same order
  - Invalidated around the same time

Effect of RX-S

Reducing SSD WAF

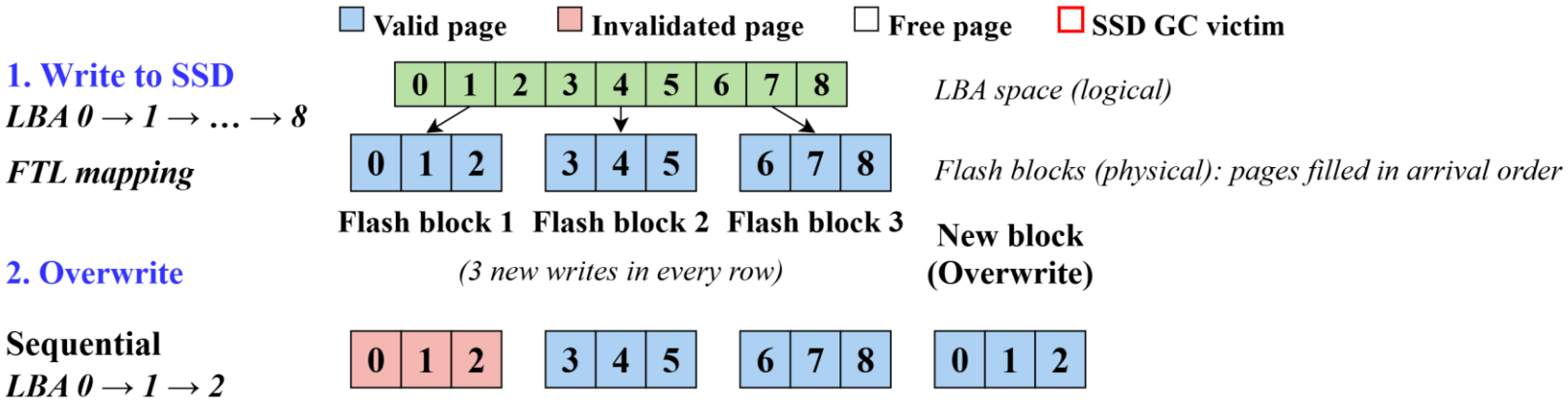
- FTL puts data with nearby LBAs into consecutive flash pages
- Next RX-S pass overwrites them in the same order
  - Invalidated around the same time



Effect of RX-S

Reducing SSD WAF

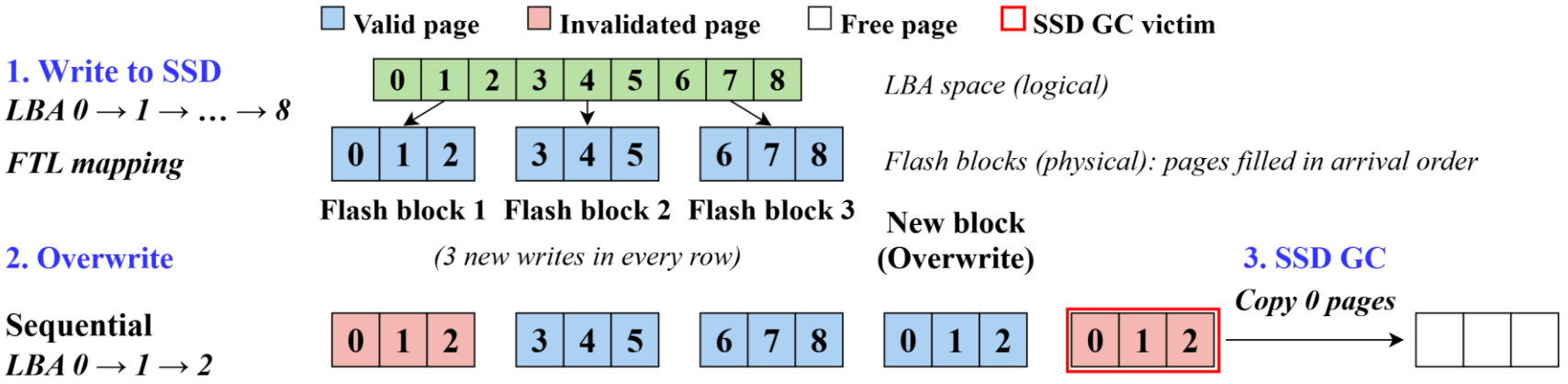
- FTL puts data with nearby LBAs into consecutive flash pages
- Next RX-S pass overwrites them in the same order
  - Invalidated around the same time



Effect of RX-S

Reducing SSD WAF

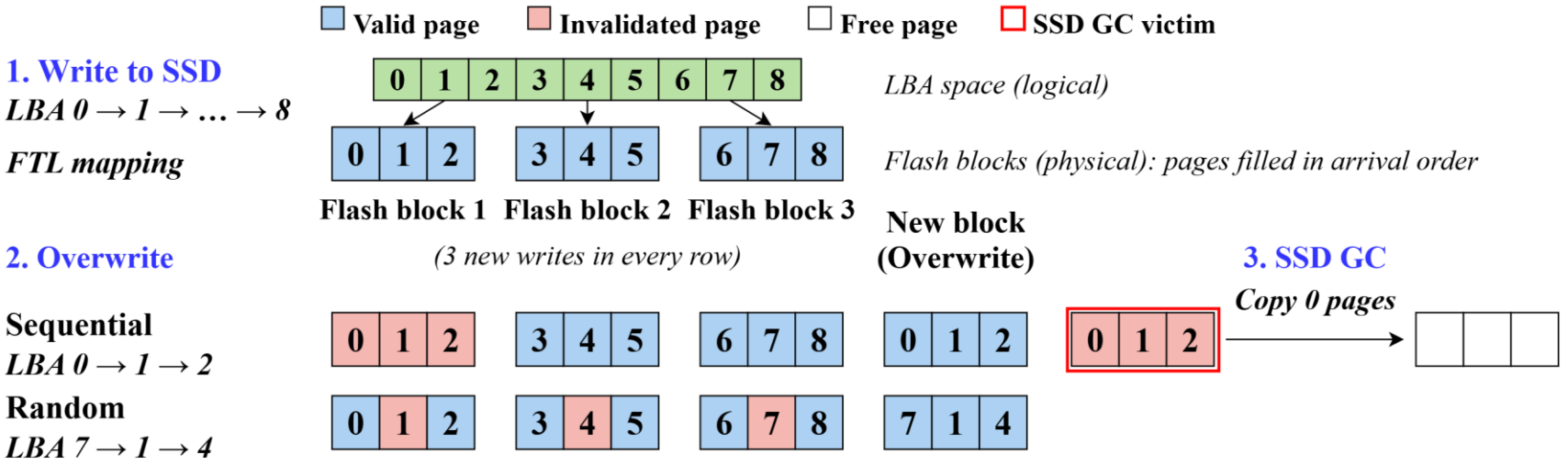
- FTL puts data with nearby LBAs into consecutive flash pages
- Next RX-S pass overwrites them in the same order
  - Invalidated around the same time



Effect of RX-S

Reducing SSD WAF

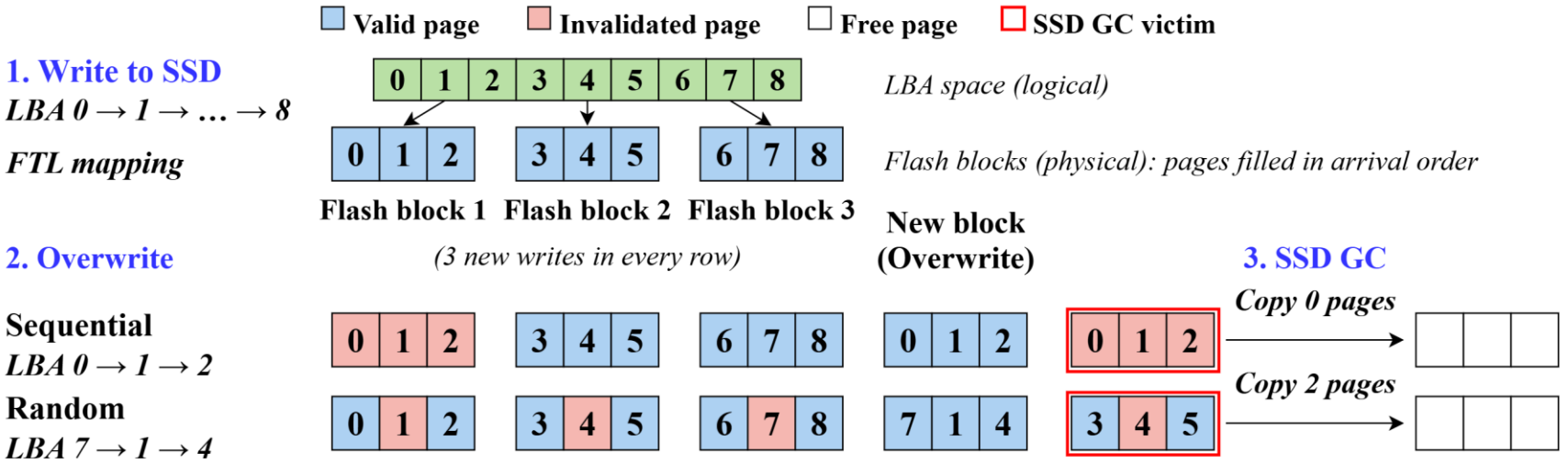
- FTL puts data with nearby LBAs into consecutive flash pages
- Next RX-S pass overwrites them in the same order
  - Invalidated around the same time



Effect of RX-S

Reducing SSD WAF

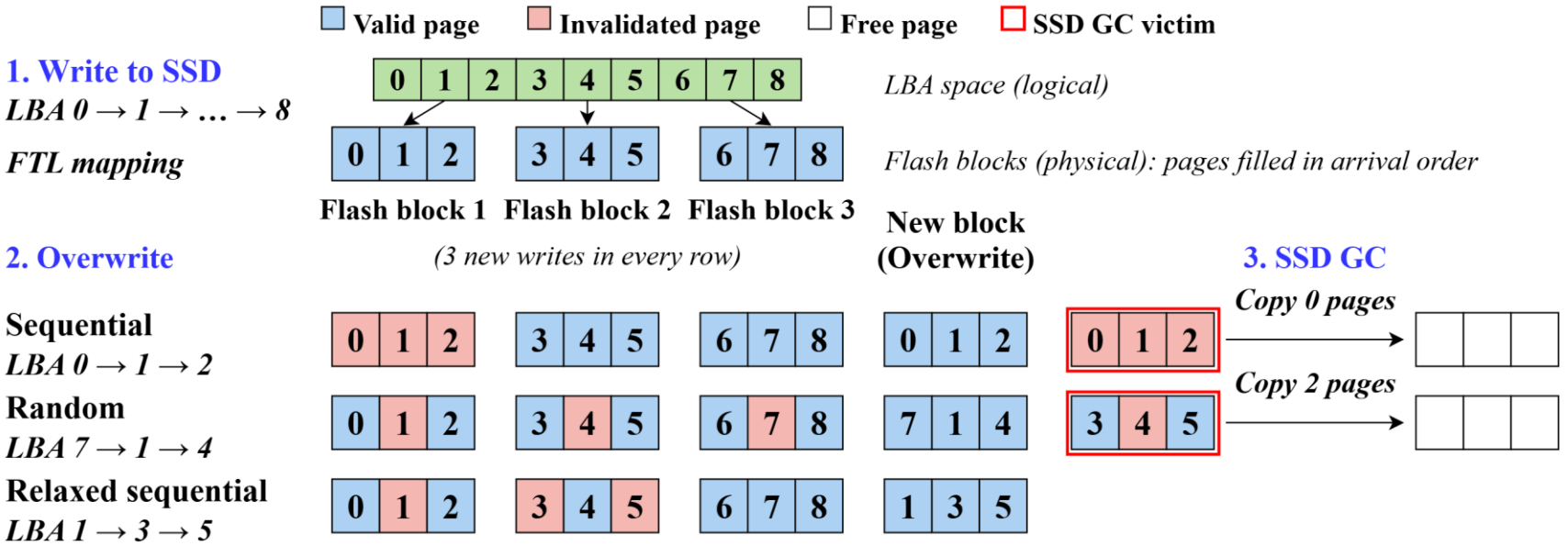
- FTL puts data with nearby LBAs into consecutive flash pages
- Next RX-S pass overwrites them in the same order
  - Invalidated around the same time



Effect of RX-S

Reducing SSD WAF

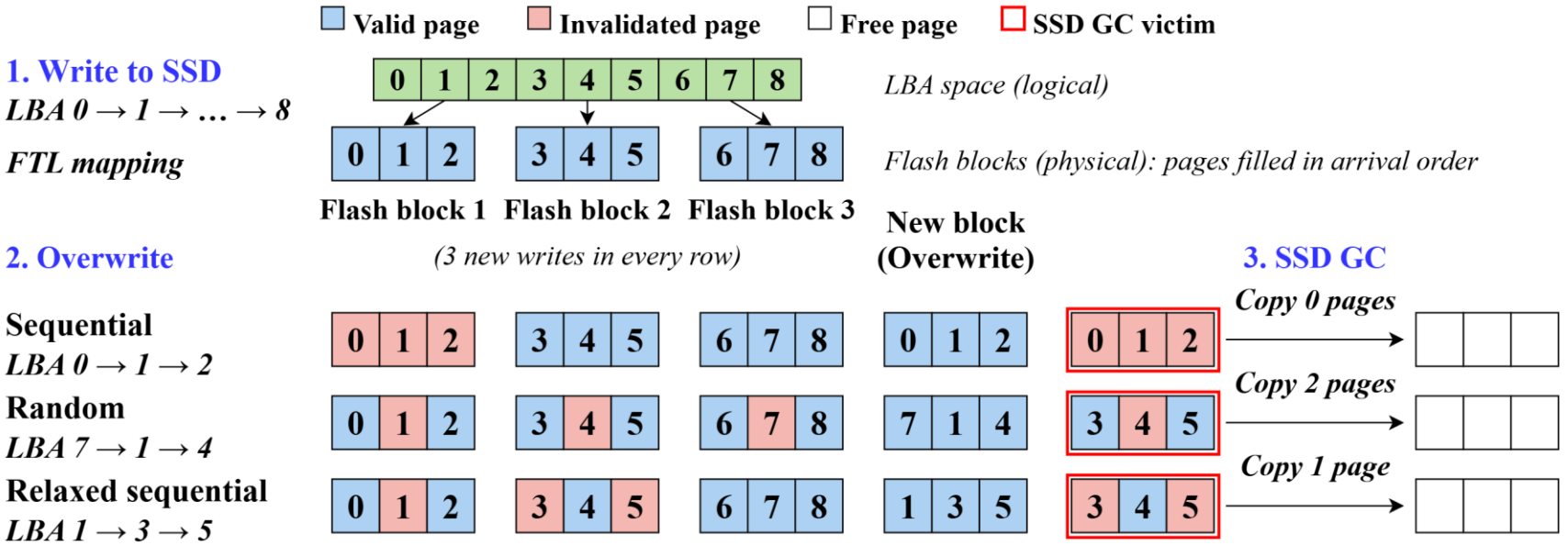
- FTL puts data with nearby LBAs into consecutive flash pages
- Next RX-S pass overwrites them in the same order
  - Invalidated around the same time



Effect of RX-S

Reducing SSD WAF

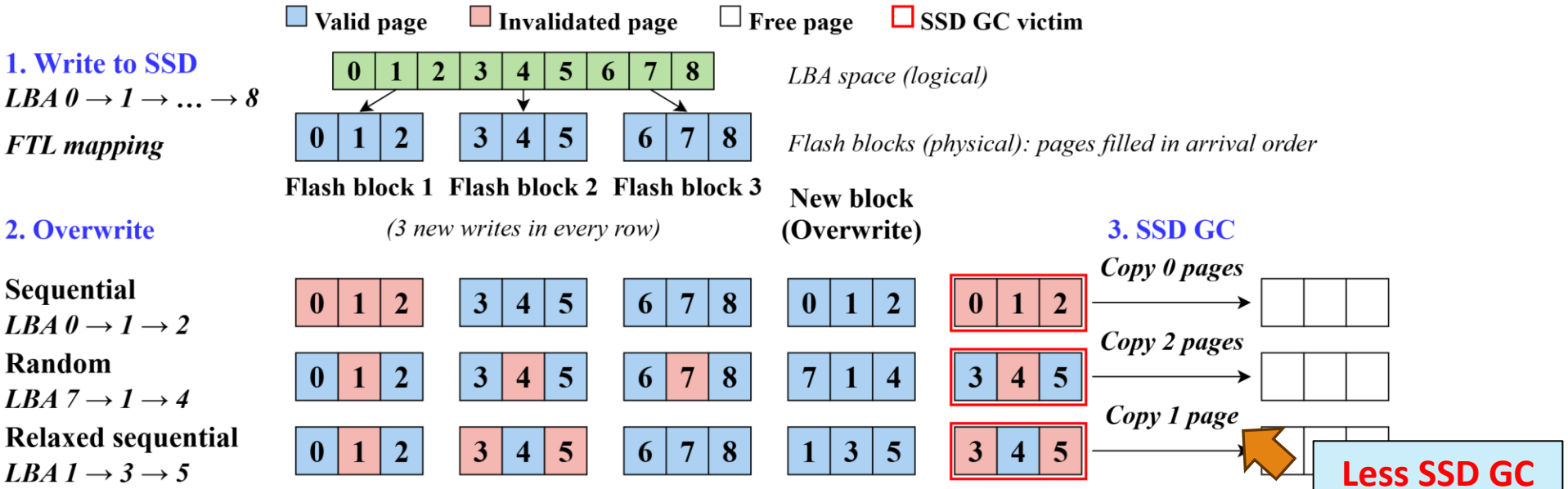
- FTL puts data with nearby LBAs into consecutive flash pages
- Next RX-S pass overwrites them in the same order
  - Invalidated around the same time



Effect of RX-S

Reducing SSD WAF

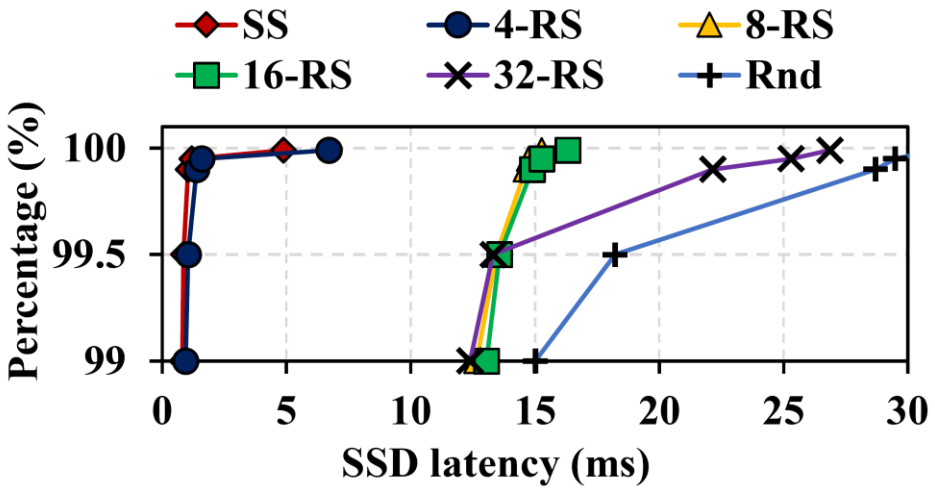
- FTL puts data with nearby LBAs into consecutive flash pages
- Next RX-S pass overwrites them in the same order
  - Invalidated around the same time



Measurement of RX-S

Preliminary experiment (Tail latency)

- Emulate relaxed sequential patterns with **stride patterns**
  - X-RS: Stride patterns with X KB gaps
  - SS: Strict sequential patterns (Typical sequential)
  - Rnd: Random patterns



Latency results

SS < 4-RS < 8-RS < 16-RS < 32-RS < Rnd

The narrower the gap, the closer to sequential-write latency

Sustainability of RX-S

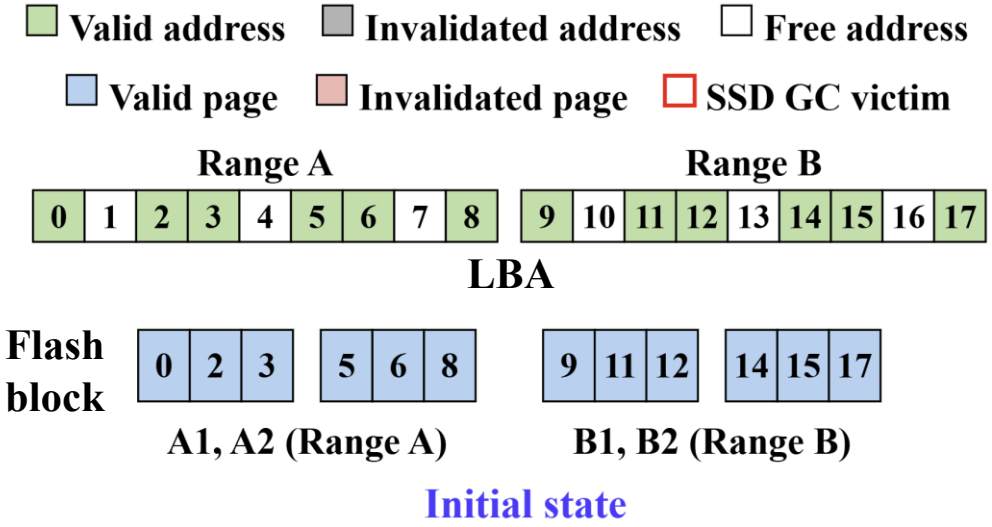
To sustain the effect of RX-S,

- Narrow gap between neighboring writes

Sustainability of RX-S

To sustain the effect of RX-S,

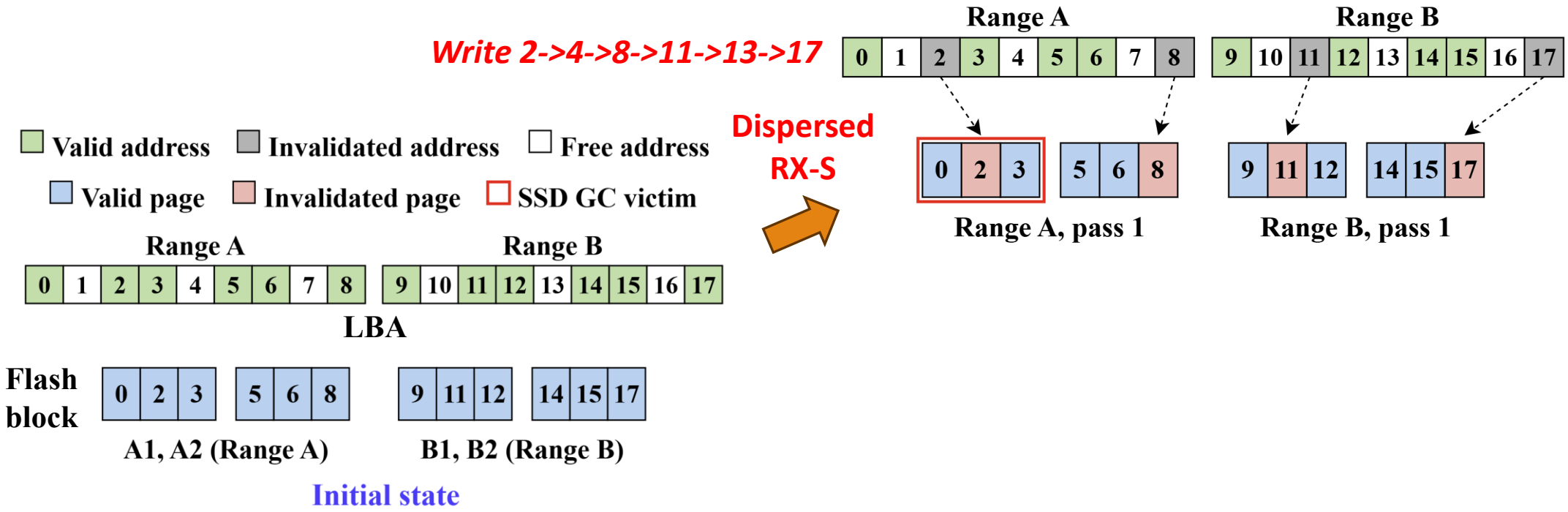
- Narrow gap between neighboring writes



Sustainability of RX-S

To sustain the effect of RX-S,

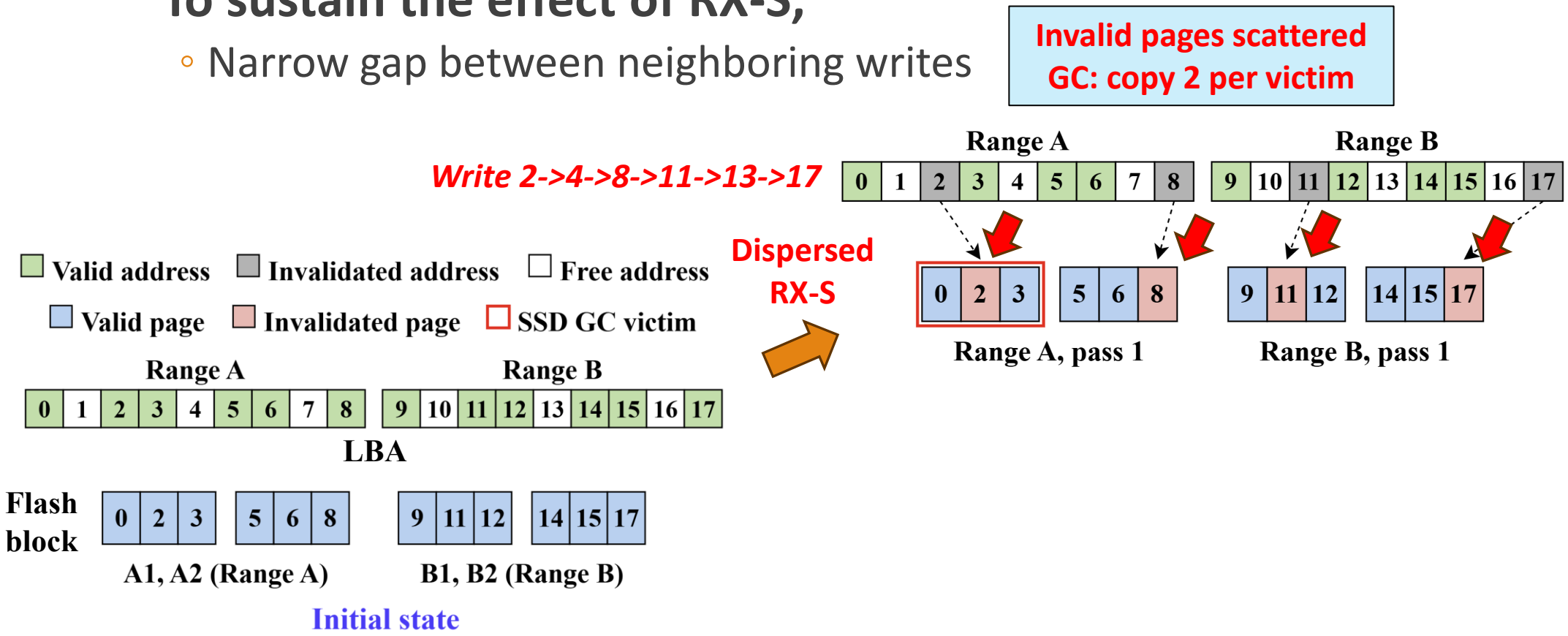
- Narrow gap between neighboring writes



Sustainability of RX-S

To sustain the effect of RX-S,

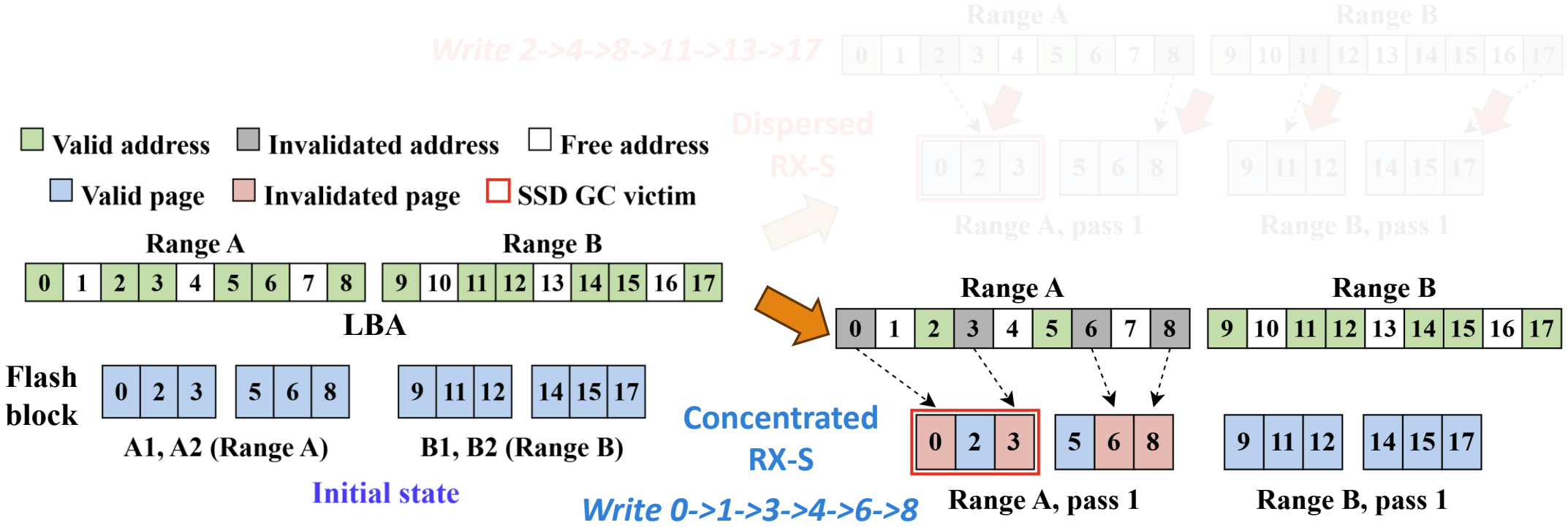
- Narrow gap between neighboring writes



Sustainability of RX-S

To sustain the effect of RX-S,

- Narrow gap between neighboring writes



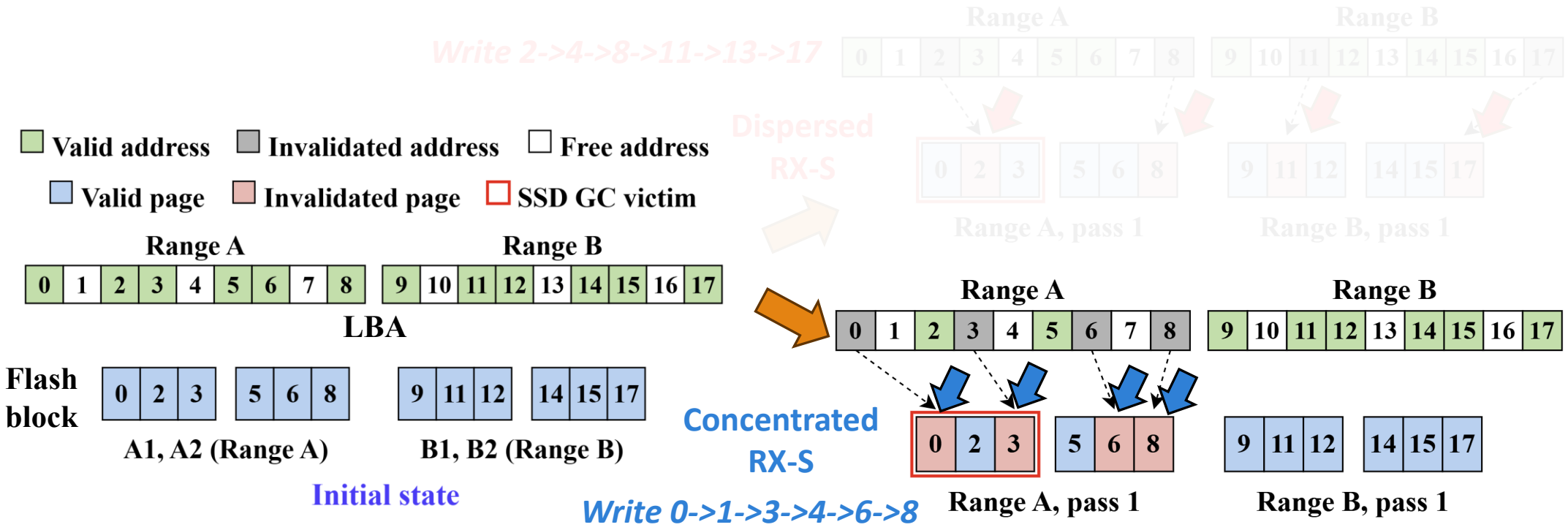
Sustainability of RX-S

To sustain the effect of RX-S,

- Narrow gap between neighboring writes

Invalid pages scattered
GC: copy 2 per victim

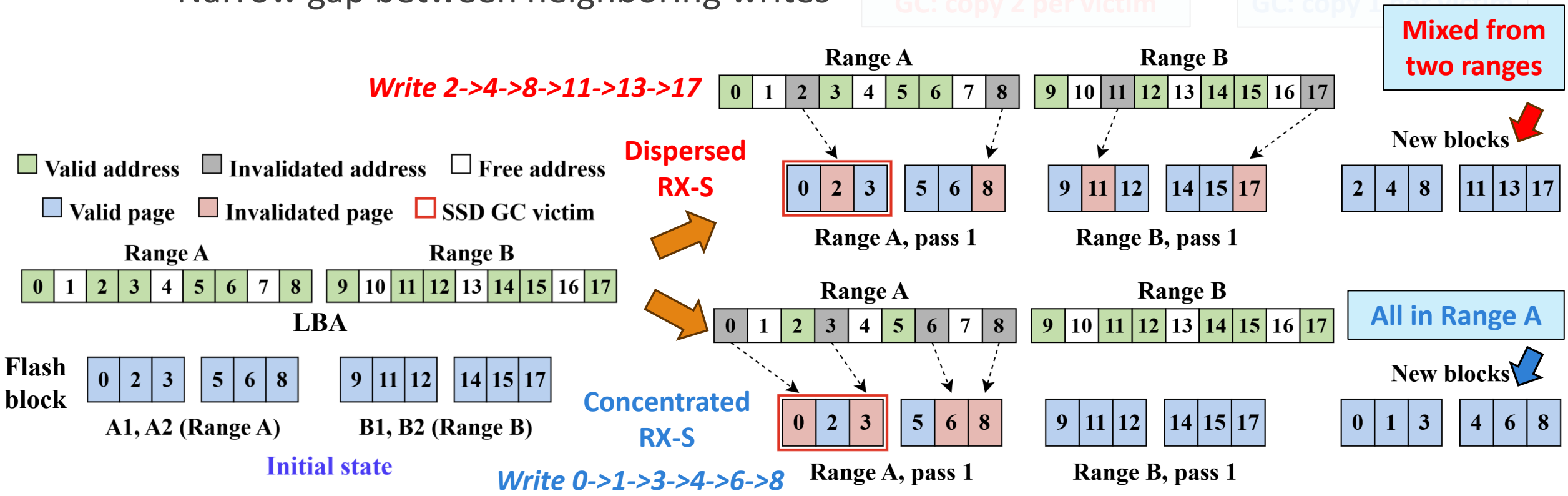
Invalid pages packed
GC: copy 1 per victim



Sustainability of RX-S

To sustain the effect of RX-S,

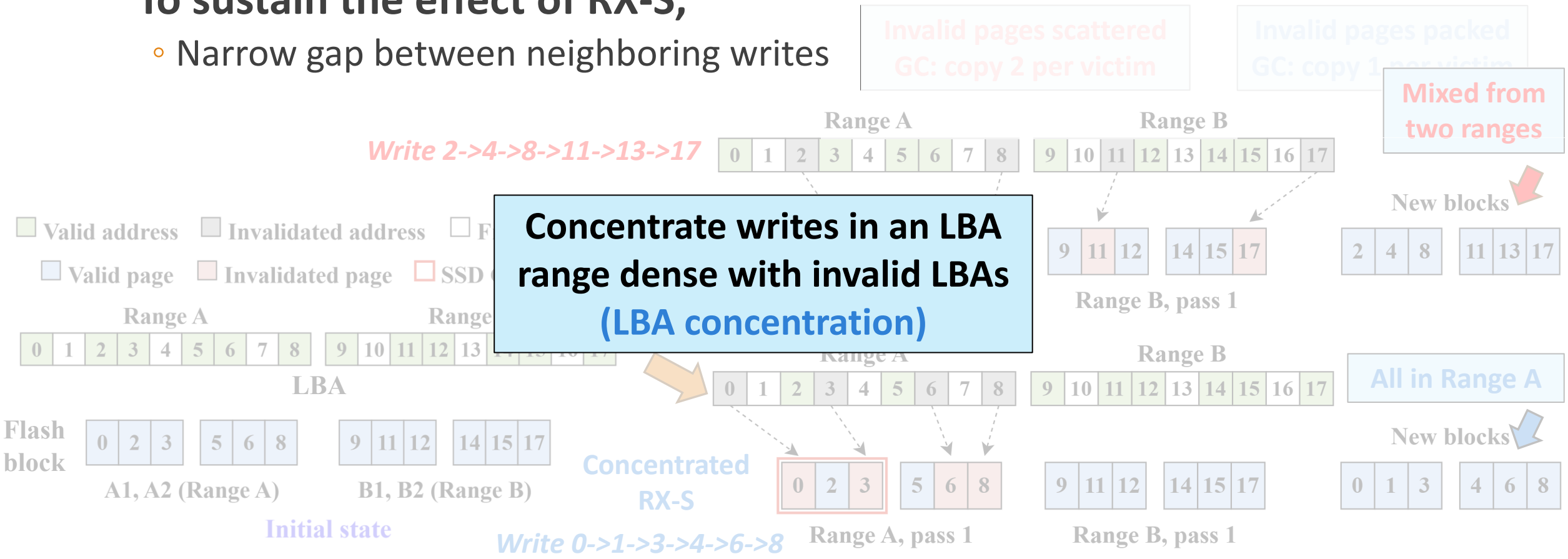
- Narrow gap between neighboring writes



Sustainability of RX-S

To sustain the effect of RX-S,

- Narrow gap between neighboring writes

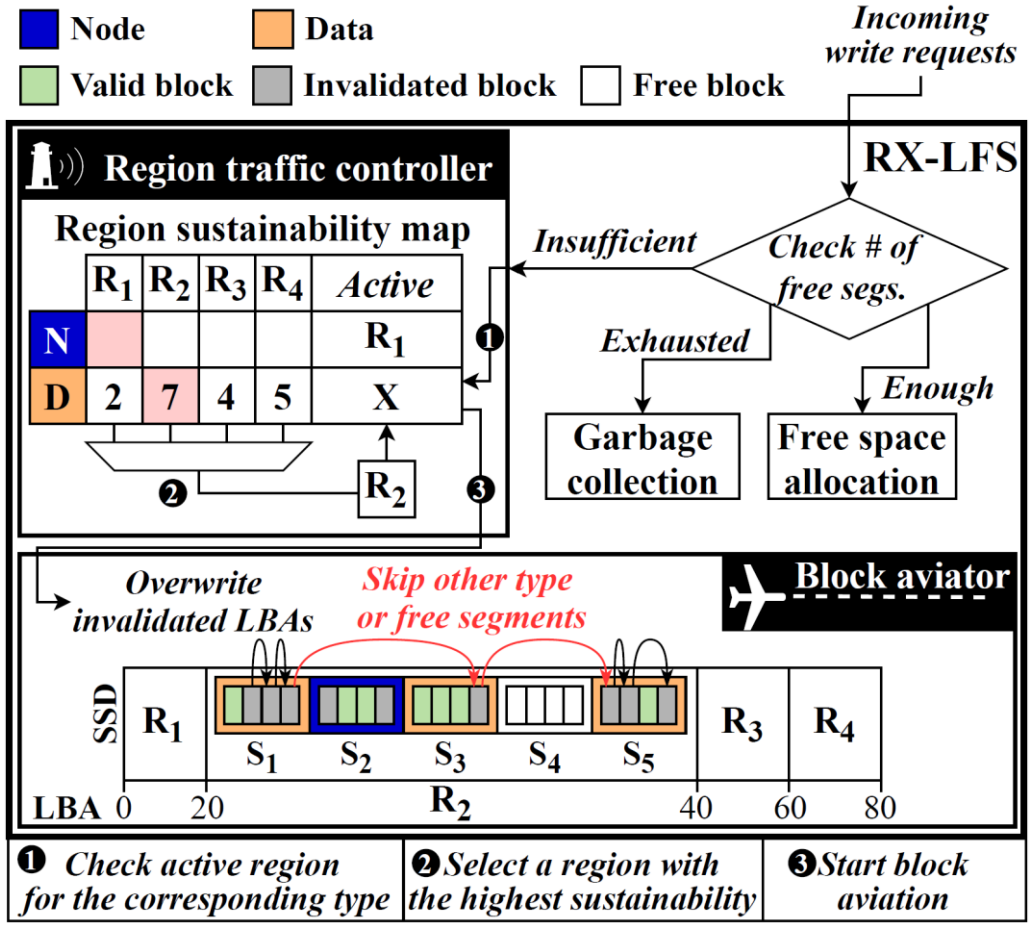


Proposed Design

RX-LFS with two components

- **Region Traffic Controller (RTC)**
  - Concentrate writes in one target region
    - Region: fixed-size LBA range
- **Block Aviator (BA)**
  - Place writes in the RX-S manner

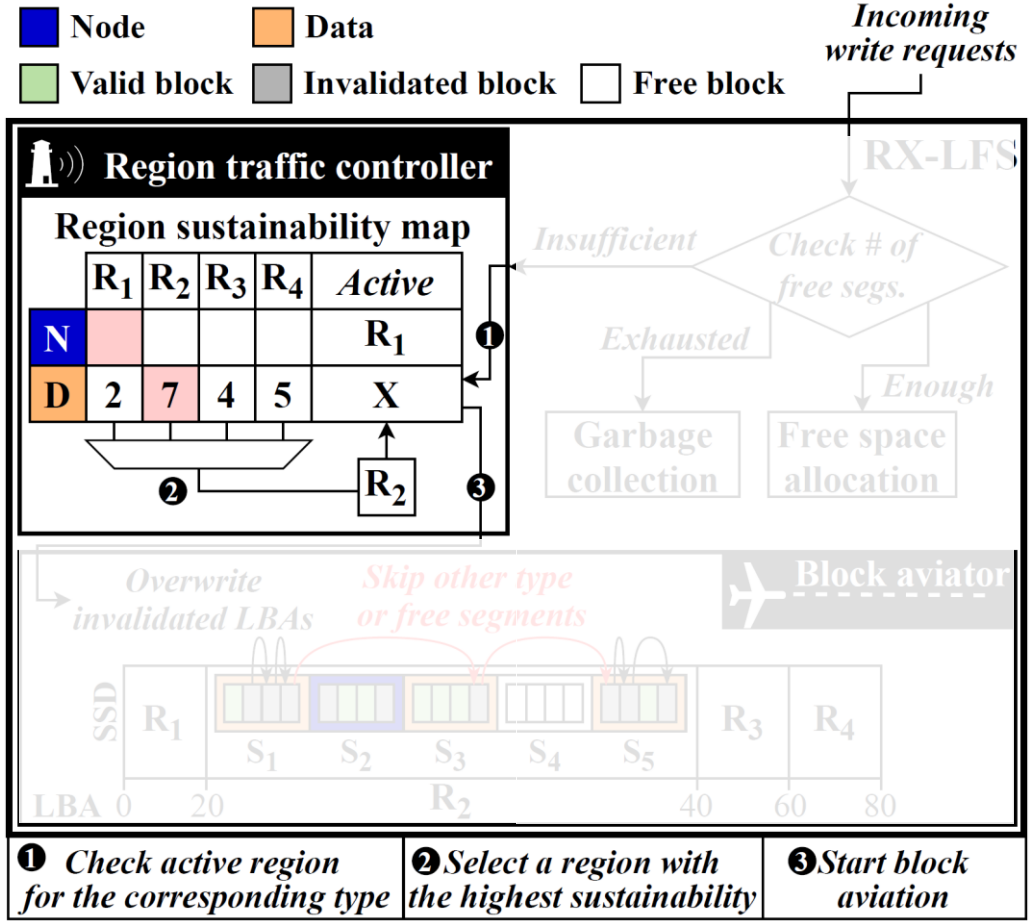
Both components coexist with GC



Region Traffic Controller (RTC)

LBA concentration

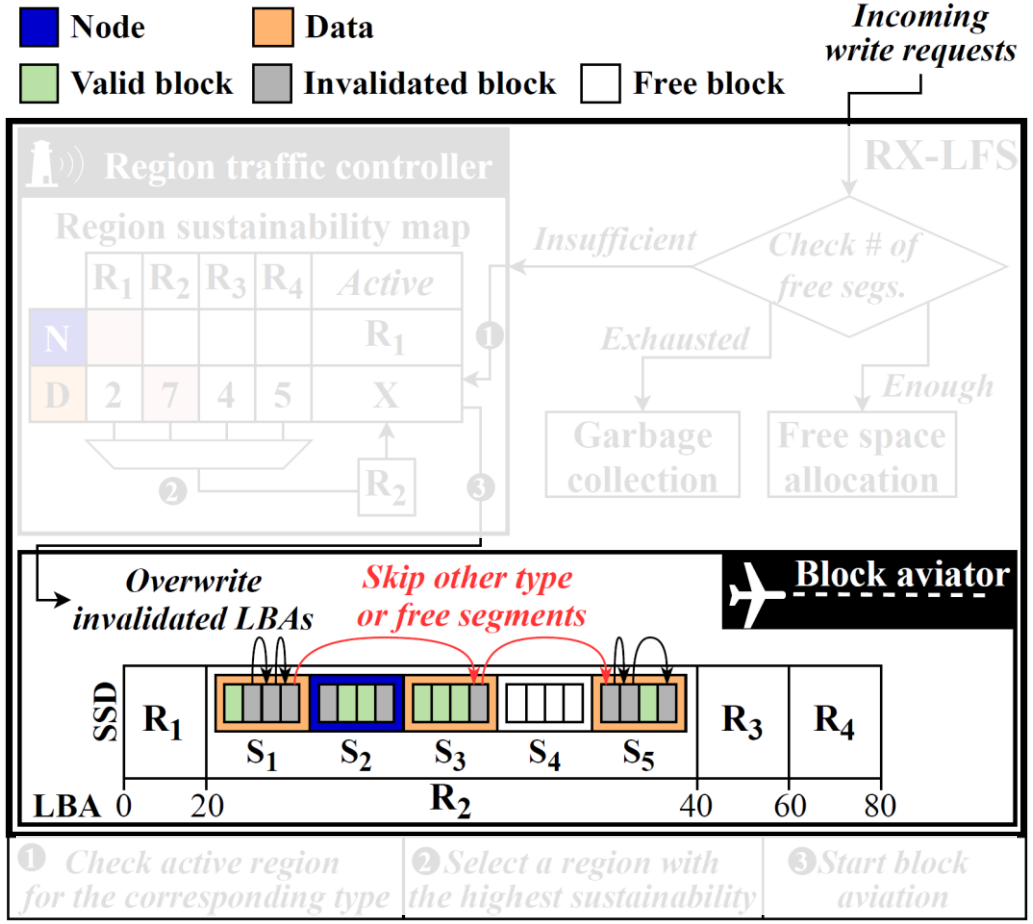
- Target region selection
  - Based on sustainability score (Invalid LBA count)
- Region sustainability map (RSM)
  - Maintain per-region sustainability score
- Region re-selection
  - When the target region runs out of invalid LBAs



Block Aviator (BA)

Relaxed sequential writes

- Traverse the target region
  - From the first to the last address in the target region
- New write placement
  - Place onto invalidated addresses in the RX-S manner
- Coexistence with existing mechanisms
  - Skip segments of other types or free
  - Preserve hot/cold separation or LFS GC



Experimental Setup

Baselines & our design

- F2FS
  - Existing LFS using both SSR and GC
- F2FS-L
  - F2FS with SSR disabled (GC only)
- ScaleLFS
  - State-of-the-art LFS with scalable GC
- **RX-LFS**
  - Proposed design on top of F2FS

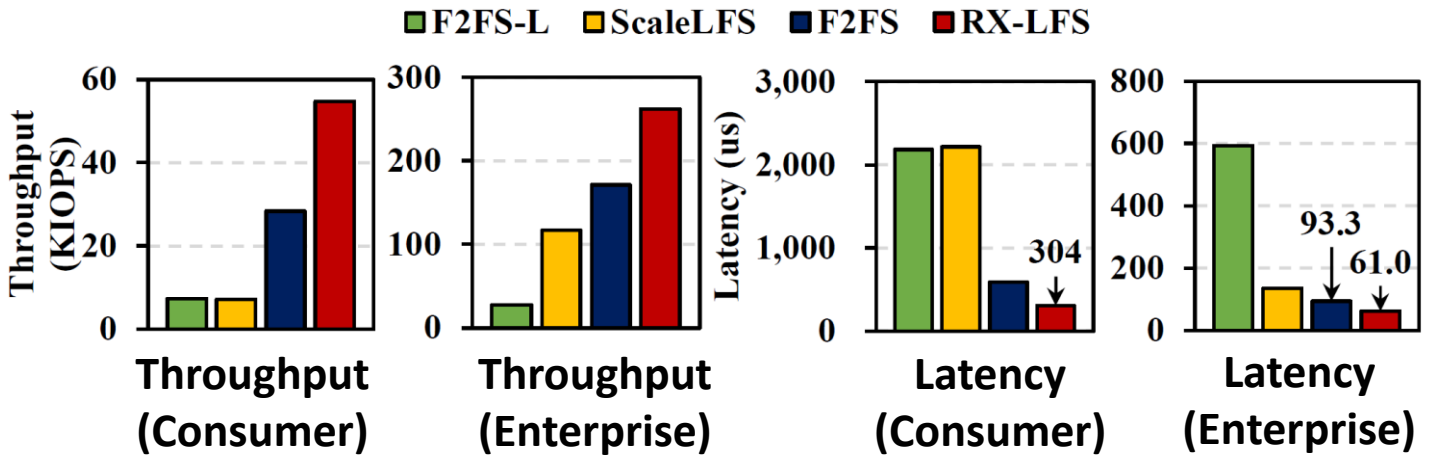
Environment

- Linux kernel 6.6.52
- Ubuntu 24.04
- Intel i7-13700K CPU
- SSD
  - Samsung 980 500 GB SSD (Consumer)
  - Samsung PM9A3 7.8 TB SSD (Enterprise)
- Memory
  - 16 GB DRAM (For consumer SSD)
  - 64 GB DRAM (For enterprise SSD)

Micro-Benchmark Results

FIO benchmark

- 16-thread 4 KB random-write workload (Zipfian θ = 0.5)
- Write amount
  - 300% (Consumer) and 150% (Enterprise) of device capacities

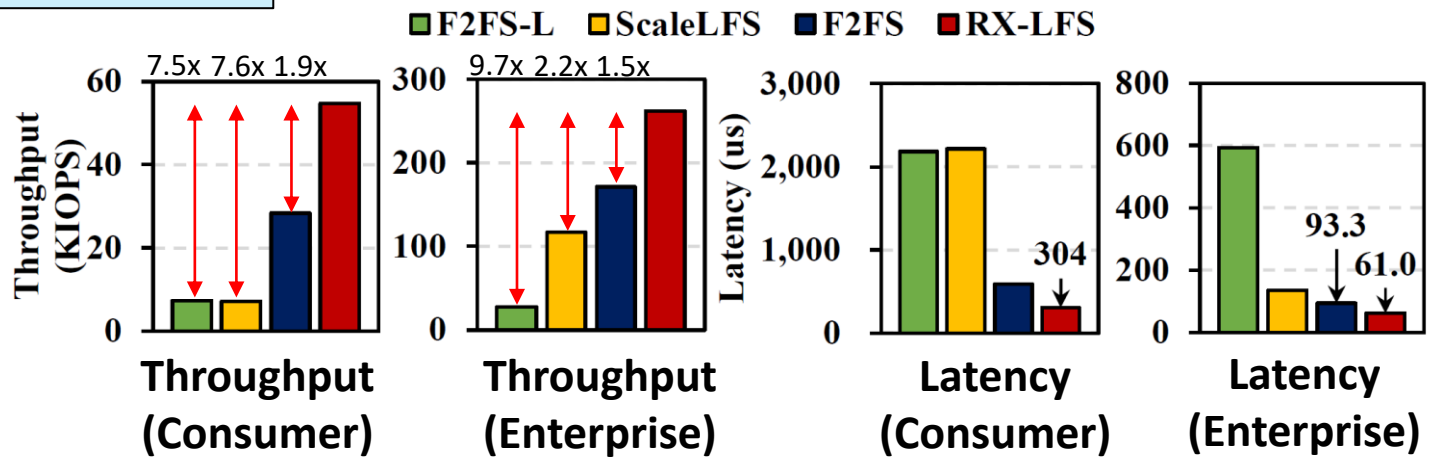


Micro-Benchmark Results

FIO benchmark

- 16-thread 4 KB random-write workload (Zipfian θ = 0.5)
- Write amount
  - 300% (Consumer) and 150% (Enterprise) of device capacities

Improve throughput by up to 9.7x



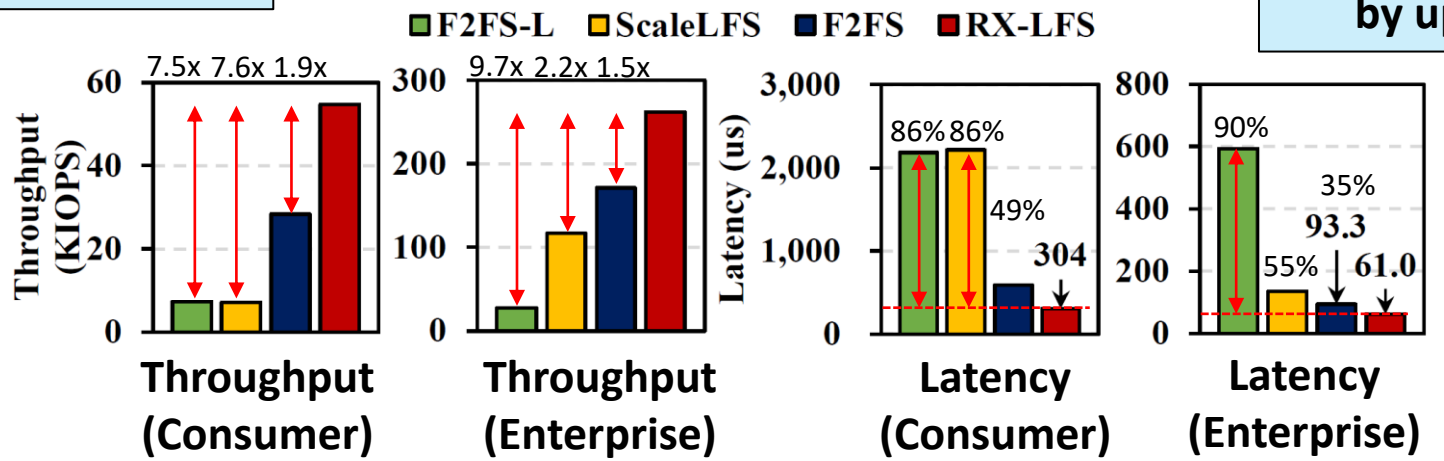
Micro-Benchmark Results

FIO benchmark

- 16-thread 4 KB random-write workload (Zipfian θ = 0.5)
- Write amount
  - 300% (Consumer) and 150% (Enterprise) of device capacities

Improve throughput by up to 9.7x

Reduce avg. latency by up to 90%

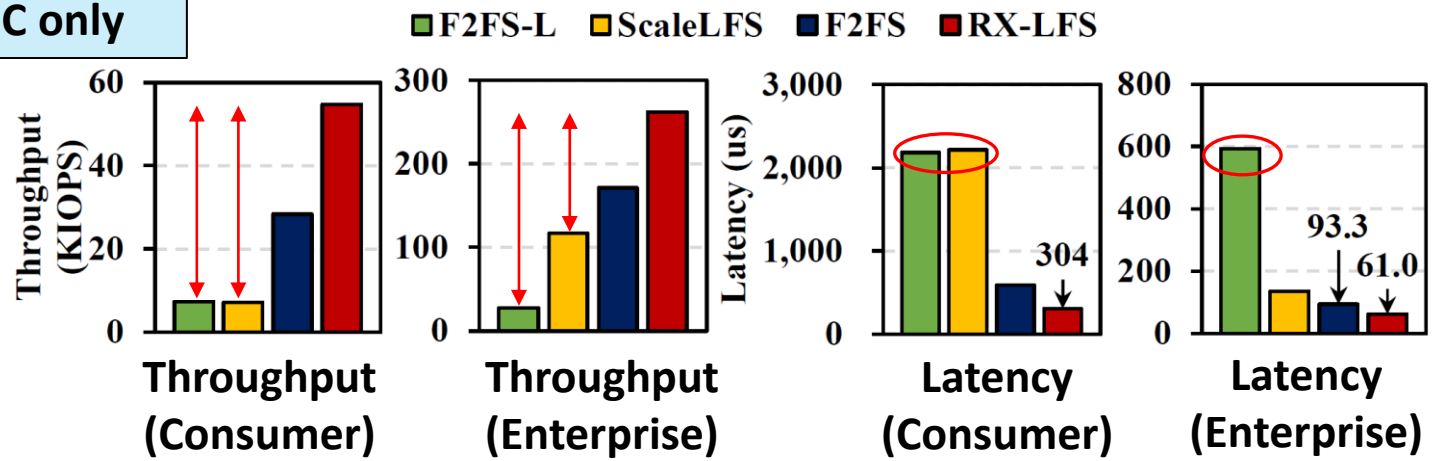


Micro-Benchmark Results

FIO benchmark

- 16-thread 4 KB random-write workload (Zipfian θ = 0.5)
- Write amount
  - 300% (Consumer) and 150% (Enterprise) of device capacities

Low performance on LFSs with GC only



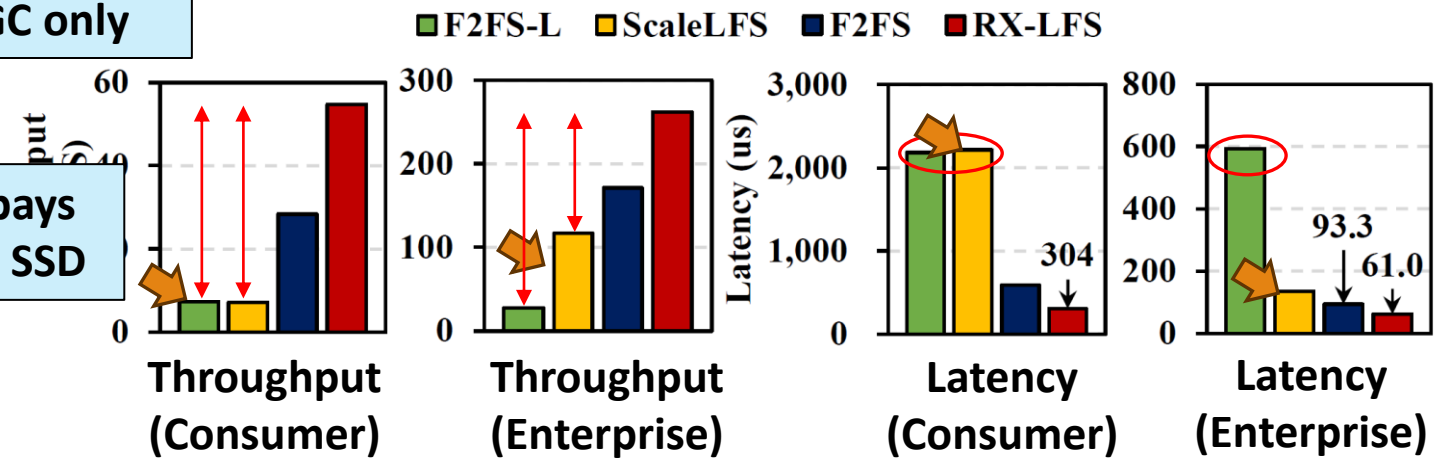
Micro-Benchmark Results

FIO benchmark

- 16-thread 4 KB random-write workload (Zipfian θ = 0.5)
- Write amount
  - 300% (Consumer) and 150% (Enterprise) of device capacities

Low performance on LFSs with GC only

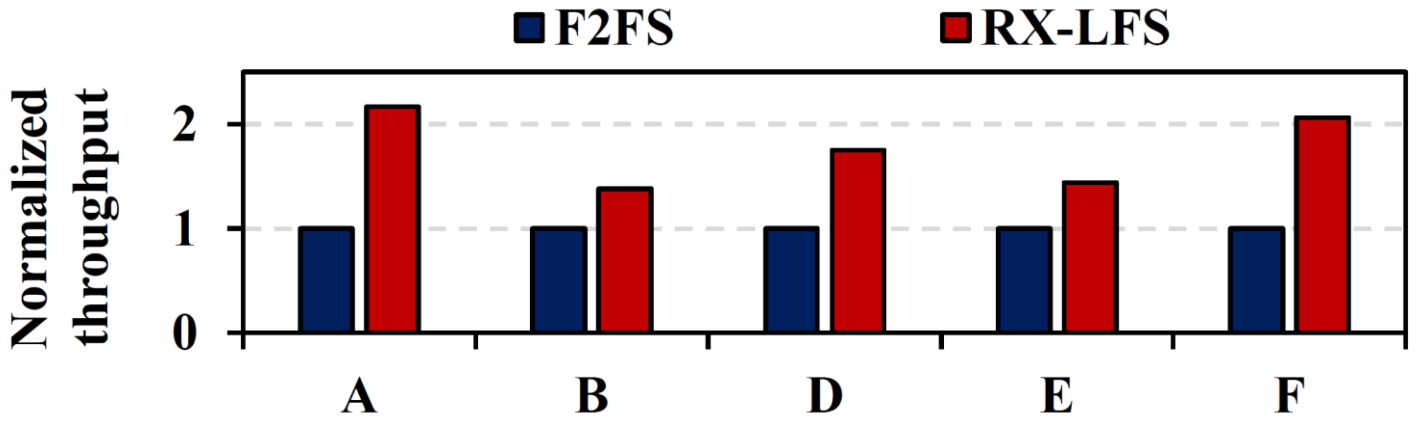
GC scalability (ScaleLFS) pays off only on the enterprise SSD



Real-World Application (MySQL)

YCSB benchmark (Workloads A, B, D, E and F)

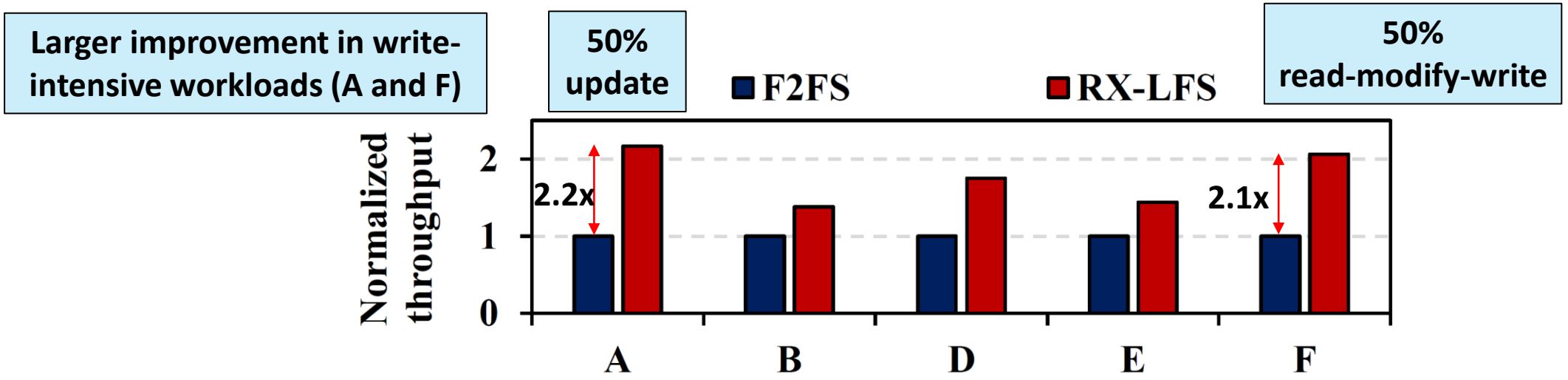
- Diverse insert/read/update ratios
- Workload C is excluded since it is read-only
- Results on the consumer SSD



Real-World Application (MySQL)

YCSB benchmark (Workloads A, B, D, E and F)

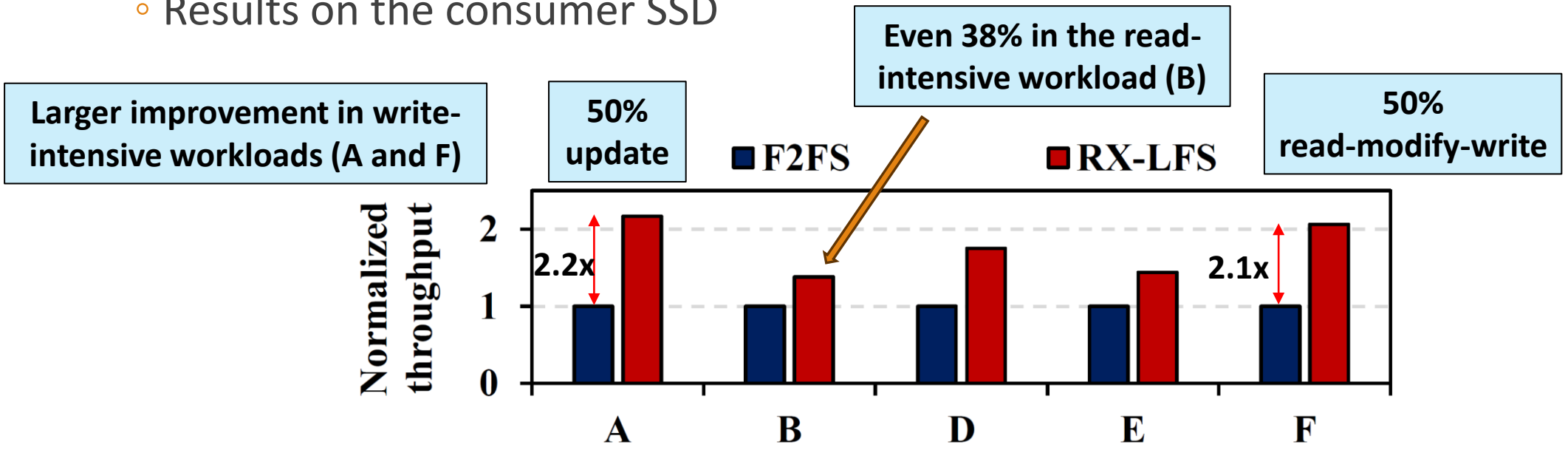
- Diverse insert/read/update ratios
- Workload C is excluded since it is read-only
- Results on the consumer SSD



Real-World Application (MySQL)

YCSB benchmark (Workloads A, B, D, E and F)

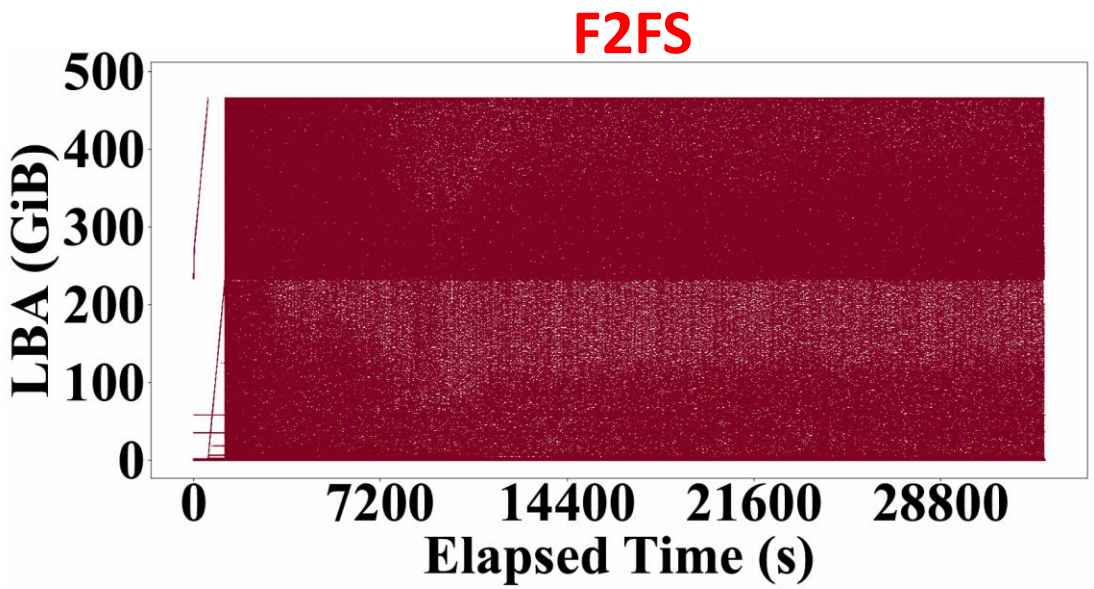
- Diverse insert/read/update ratios
- Workload C is excluded since it is read-only
- Results on the consumer SSD

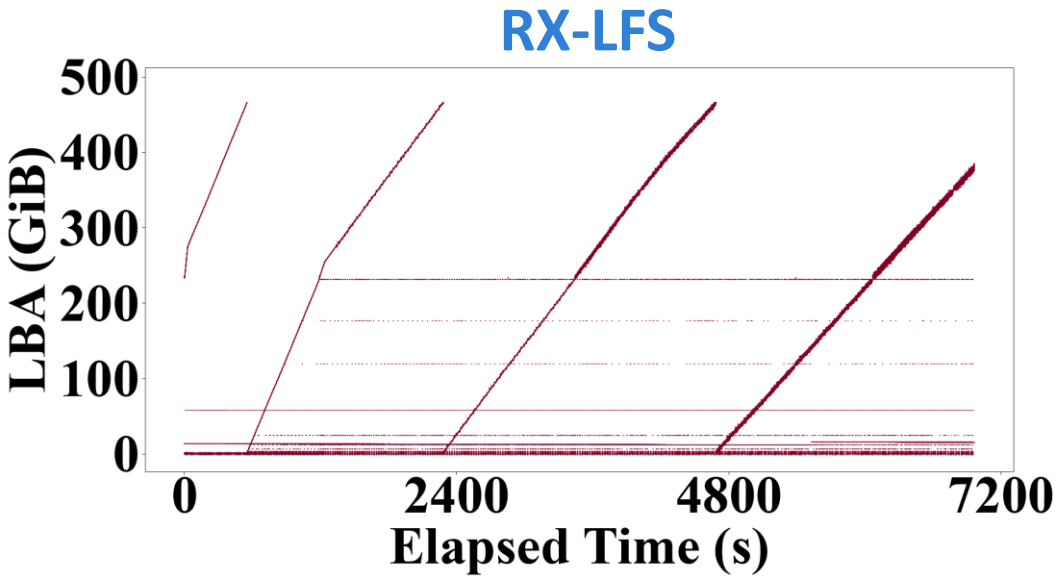


Sequentialized Write Pattern

Write patterns (Under FIO workload)

- RX-LFS sequentializes the write pattern as intended

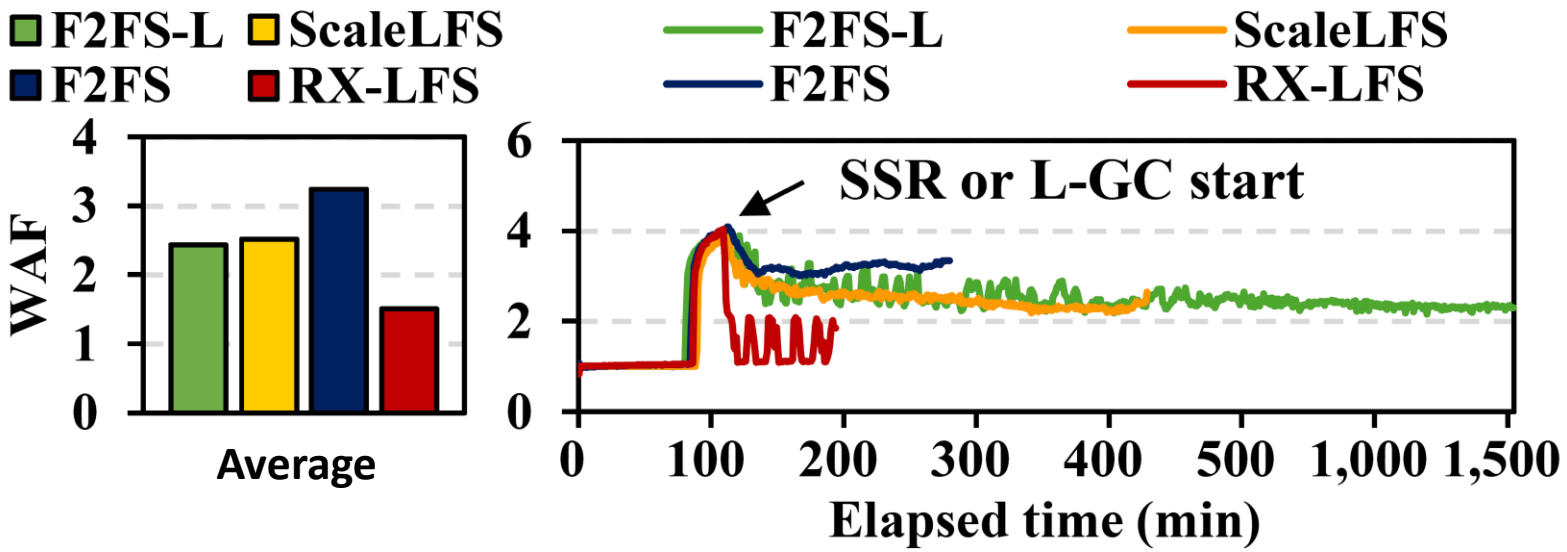




Where the Gain Comes From

SSD WAF over time (Under FIO workload)

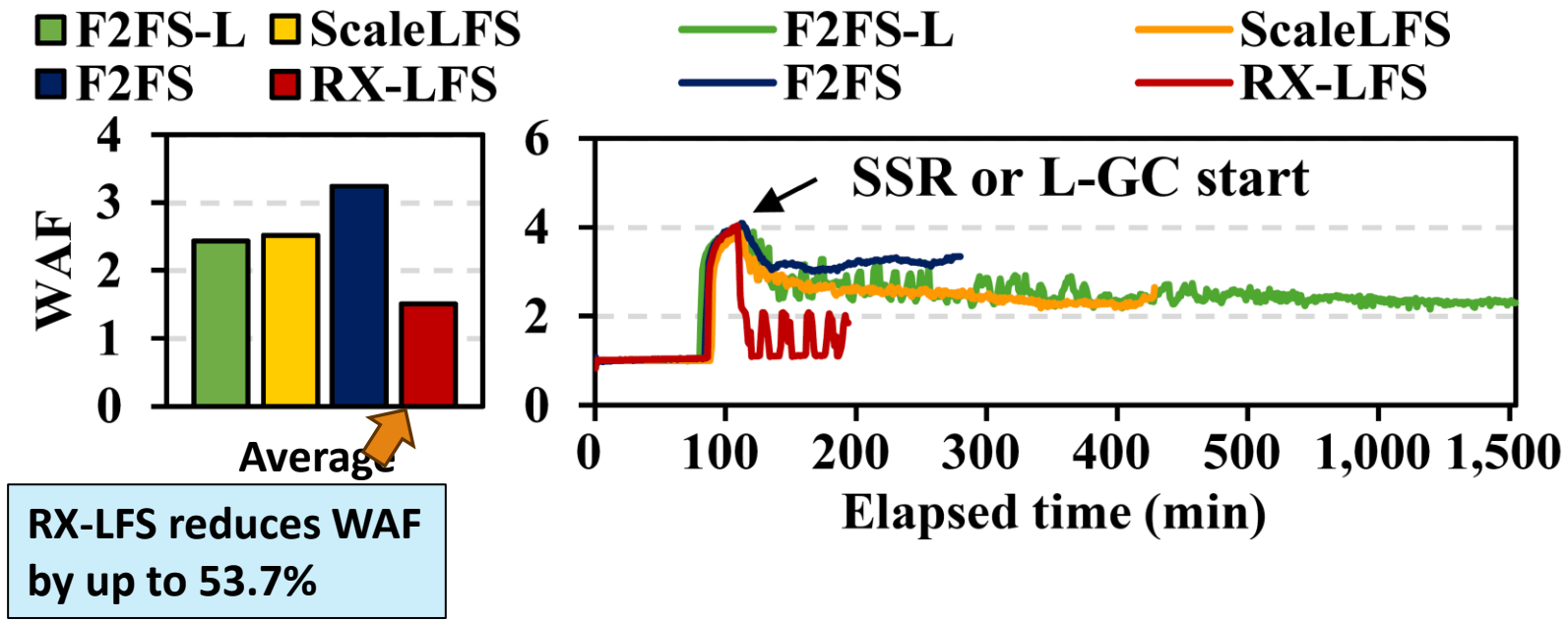
- Lower WAF, higher SSD performance



Where the Gain Comes From

SSD WAF over time (Under FIO workload)

- Lower WAF, higher SSD performance

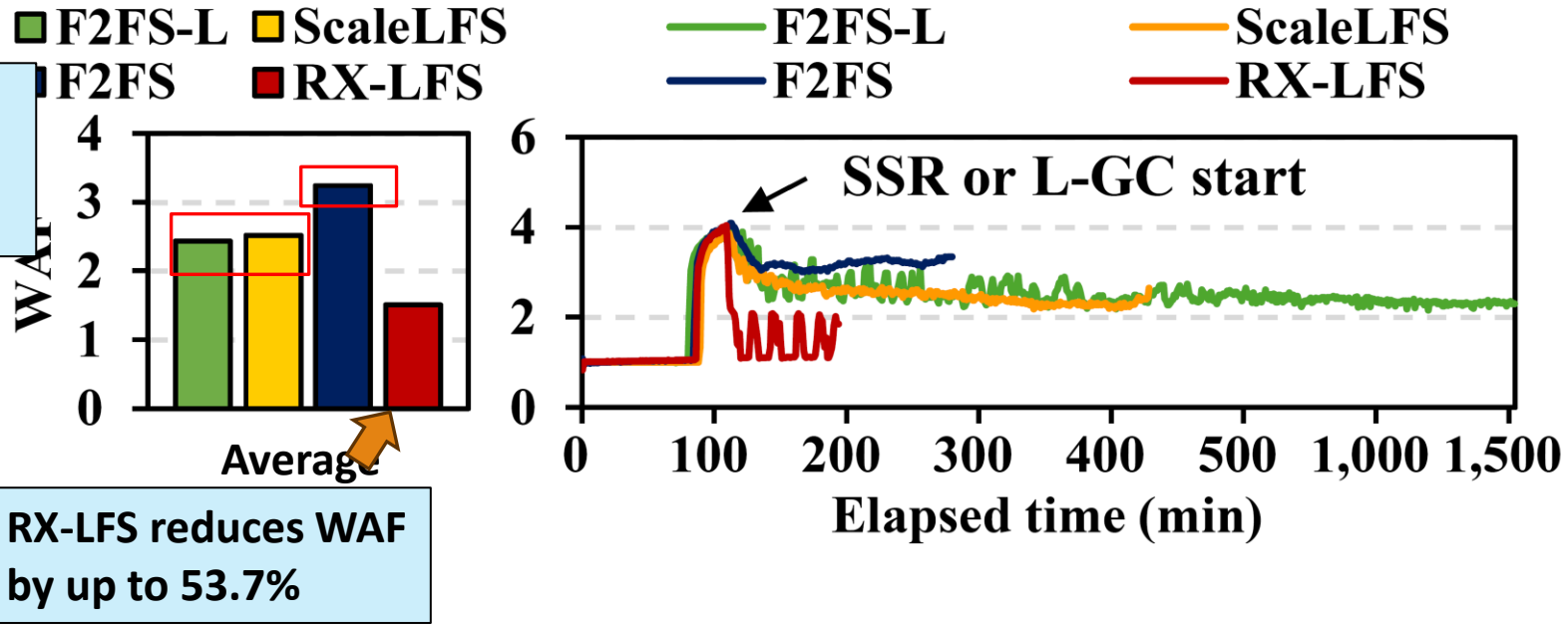


Where the Gain Comes From

SSD WAF over time (Under FIO workload)

- Lower WAF, higher SSD performance

Existing LFSs
Lower WAF with LFS GC,
Higher WAF with SSR



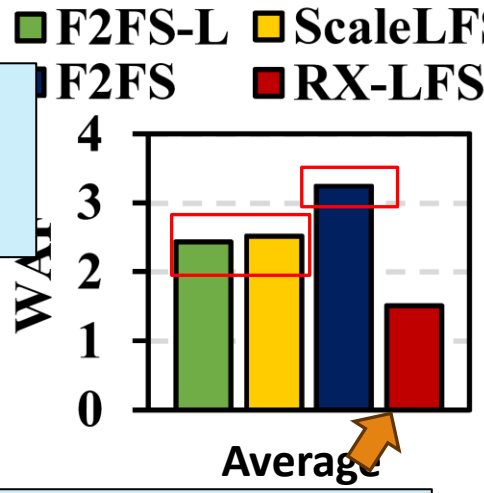
RX-LFS reduces WAF by up to 53.7%

Where the Gain Comes From

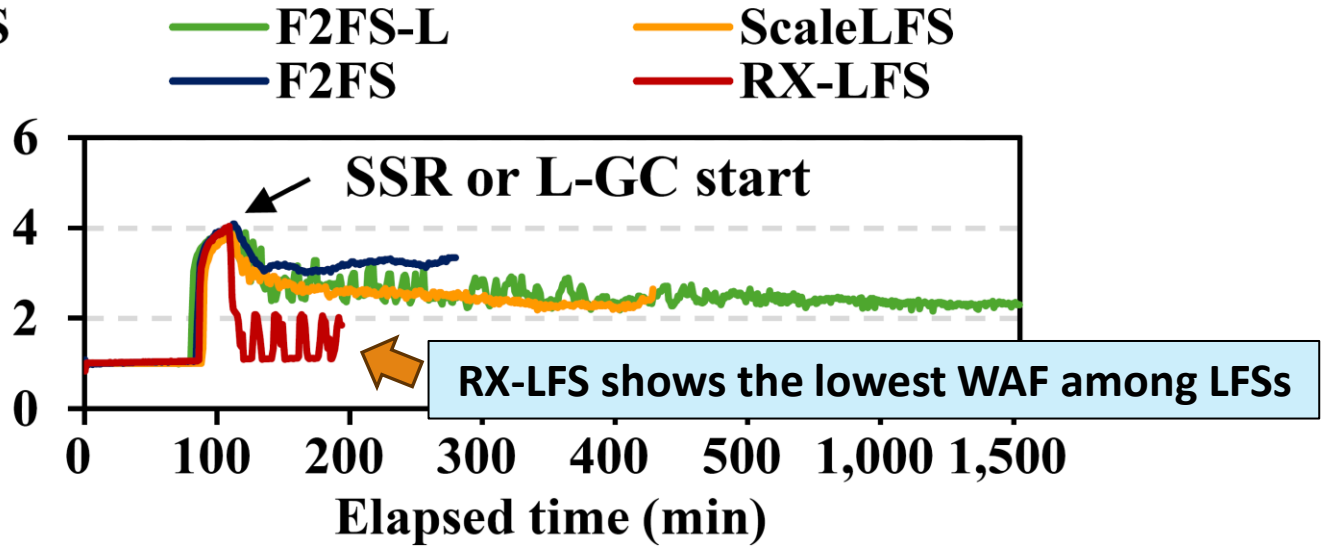
SSD WAF over time (Under FIO workload)

- Lower WAF, higher SSD performance

Existing LFSs
Lower WAF with LFS GC,
Higher WAF with SSR



RX-LFS reduces WAF by up to 53.7%



RX-LFS shows the lowest WAF among LFSs

Summary

RX-LFS improves the sustainable performance of SSR-based LFS

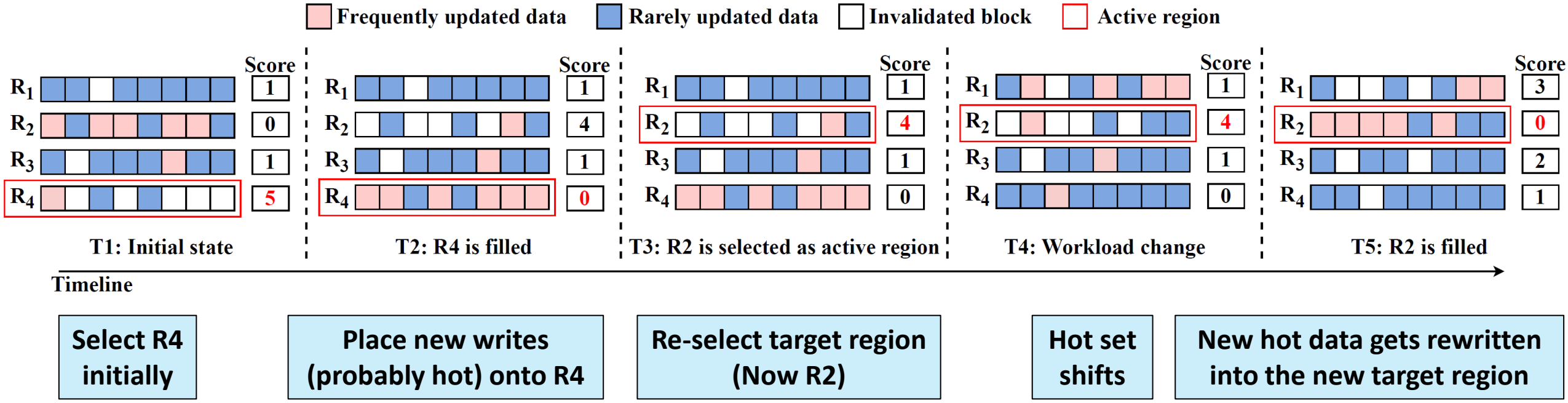
- Relaxed sequential write pattern
  - Sequential writes with gaps
  - Reduce SSD WAF (SSD GC cost)
- LBA concentration
  - Enhance the effectiveness of the RX-S pattern
  - Maintain a narrow gap between writes
- Up to 9.7x throughput improvement while achieving up to 53.7% WAF reduction

Thank you!

Operational Walkthrough

Robustness to workload changes

- Even when hotness changes, hot data naturally gathers in the target region



Select R4 initially

Place new writes (probably hot) onto R4

Re-select target region (Now R2)

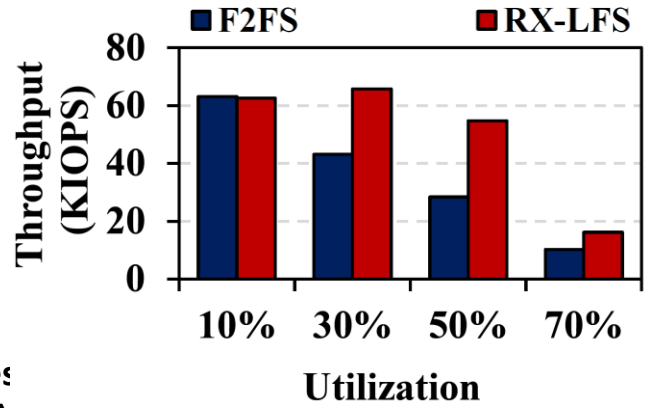
Hot set shifts

New hot data gets rewritten into the new target region

Performance under Diverse Utilization

Throughput according to file system utilization

- No performance gain at low utilization
- But high gain at 30% or higher utilization



Robustness under fluctuating utilization

