SHRD: Improving Spatial Locality in Flash Storage Accesses by Sequentializing in Host and Randomizing in Device

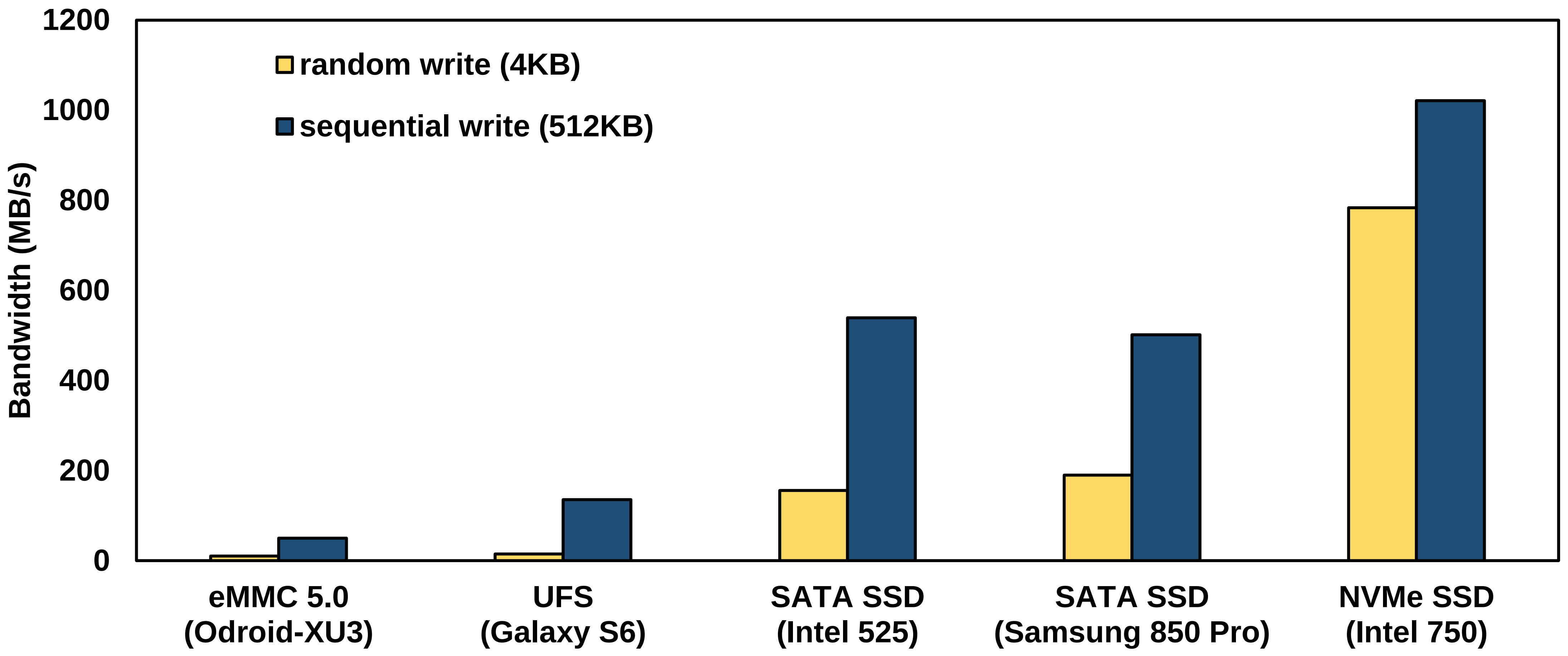
Hyukjoong Kim[1], Dongkun Shin[1], Yun Ho Jeong[2] and Kyung Ho Kim[2]

Sungkyunkwan University[1]

Samsung Electronics[2]

Presented at FAST'17

Random write is still slow at SSD



Why is RW slower than SW?

1. **Request Handling Overhead**

2. **Garbage Collection Overhead**

3. **Mapping Table Handling Overhead**

Why is RW slower than SW?

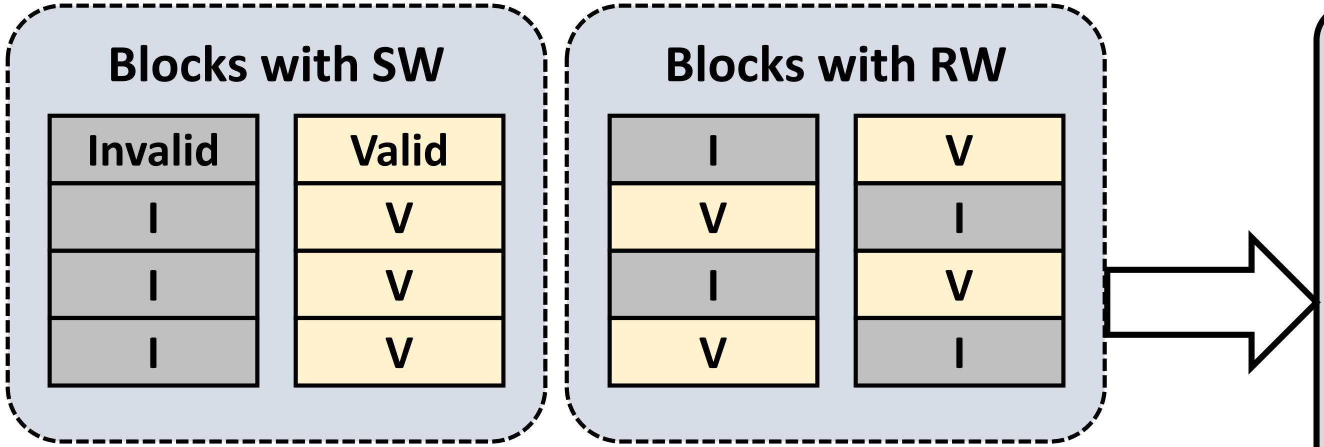
1. Request Handling Overhead

- Sequential write
  → **Large, few requests**
- Random write
  → **Small, many requests**

- **Packed command**
  - e.g. eMMC
- **Interrupt coalescing**
  - e.g. NVMe, SATA NCQ
- **Vectored I/O**
  - e.g. OpenChannel SSD [FAST'17]

Why is RW slower than SW?

2. Garbage Collection Overhead



- RW generates hot/cold-mixed blocks
- Dispersed invalid pages ➔ high GC overhead

- **Hot/cold separation**
  ➔ Stores hot and cold data into different blocks
- **Incremental GC / bgGC**
  ➔ can hide GC latency

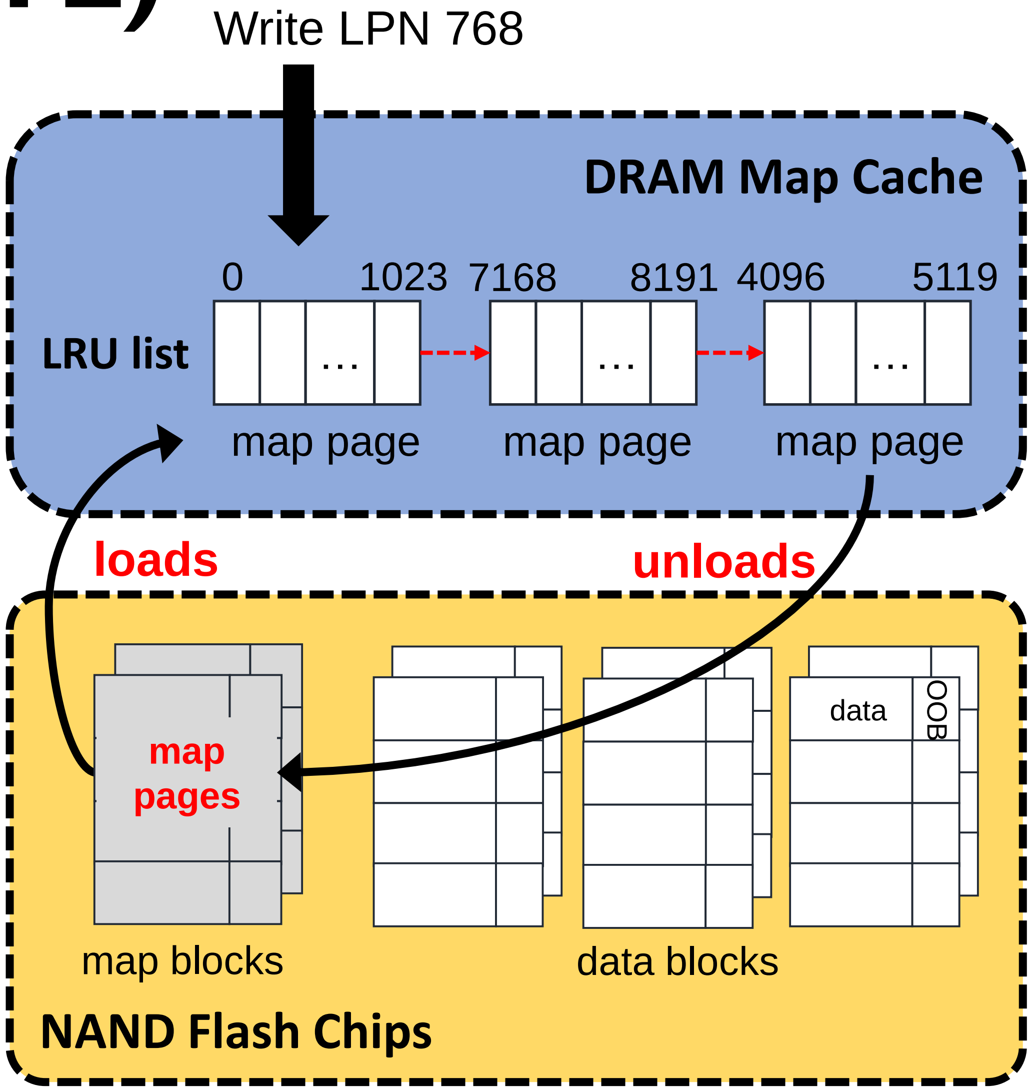
Why is RW slower than SW?

3. Mapping Table Handling Overhead

- Page-level mapping FTL shows good performance on RW
  - Requires a large DRAM to maintain fine-grained mapping table
  - 4 byte per 4 KB → **8 GB DRAM for 8 TB storage**
- **Demand loading FTL (DFTL [ASPLOS'08])**
  - Uses a **small map cache** with on-demand map loading
  - Random writes invoke frequent map loading/unloading

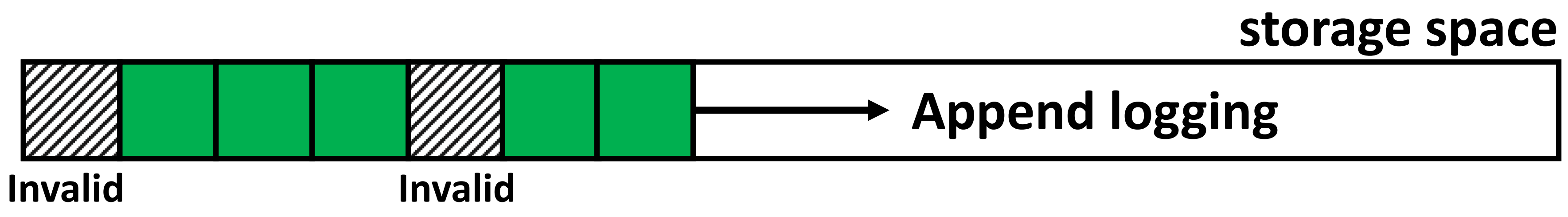
Demand-loading FTL (DFTL)

- Map caching scheme can show good performance by utilizing temporal & spatial locality
  - Page level map load/unload
    - ✓ One map page contains multiple contiguous mapping entries
- Vulnerable to random workload
  - low temporal & spatial locality
  - high map miss rate
  - high map loading overhead

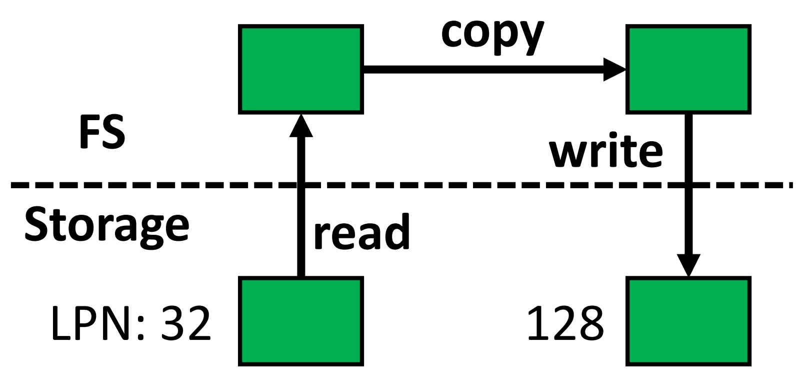


Previous Solution: LFS

- Generate **only sequential writes**
  - out-of-place append-only write scheme



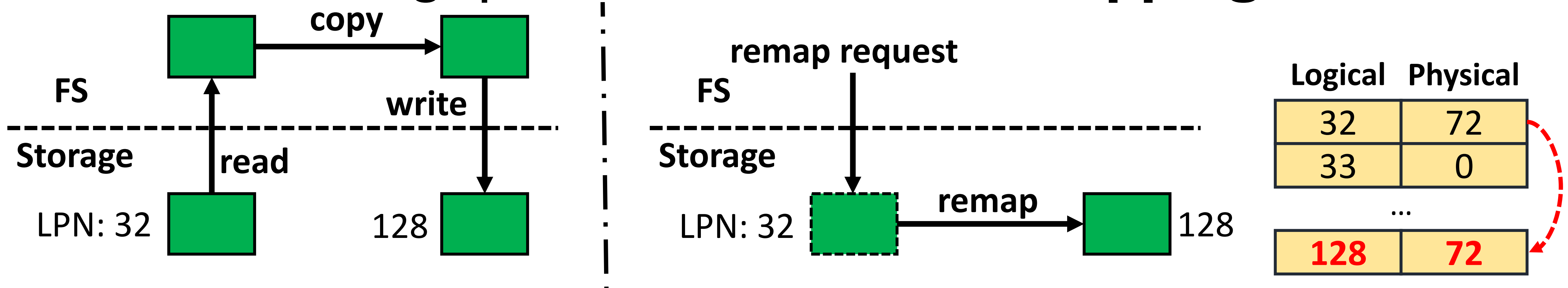
- **Problems**
  - **reclaiming log space** (cleaning overhead)
    - Filesystem needs to copy valid page
      ➔ host-to-device data transfer
  - Large metadata, wandering tree problem
  - Fragmented read operation



Cleaning requires host-to-device data transfer operations

Can we remove copy overhead?

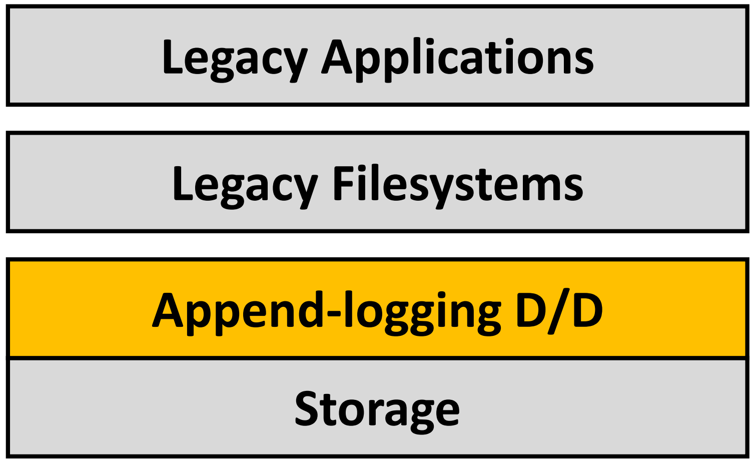
- SSD maintains a page-level mapping table
- Address remapping
  - Can change the logical address of a written data by modifying mapping table
  - AnViL [FAST'15], SHARE [SIGMOD'16]
- Can reclaim log space with **address remapping**



< Copying at filesystem >   < Remapping at storage >

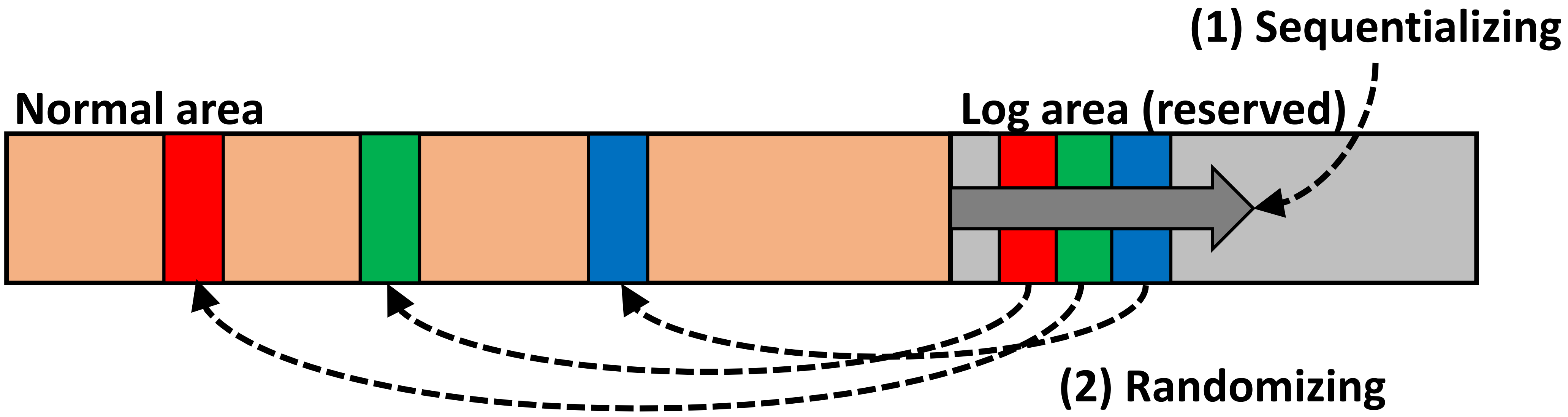
Which layer? File System or Block Layer

- Our solution is Append logging on Block Layer
  - **Append logging** on log area temporarily
  - **Remap** to the original location
  - Can utilize legacy filesystems (e.g. EXT4)
    - Simpler metadata management
    - Faster sequential read performance



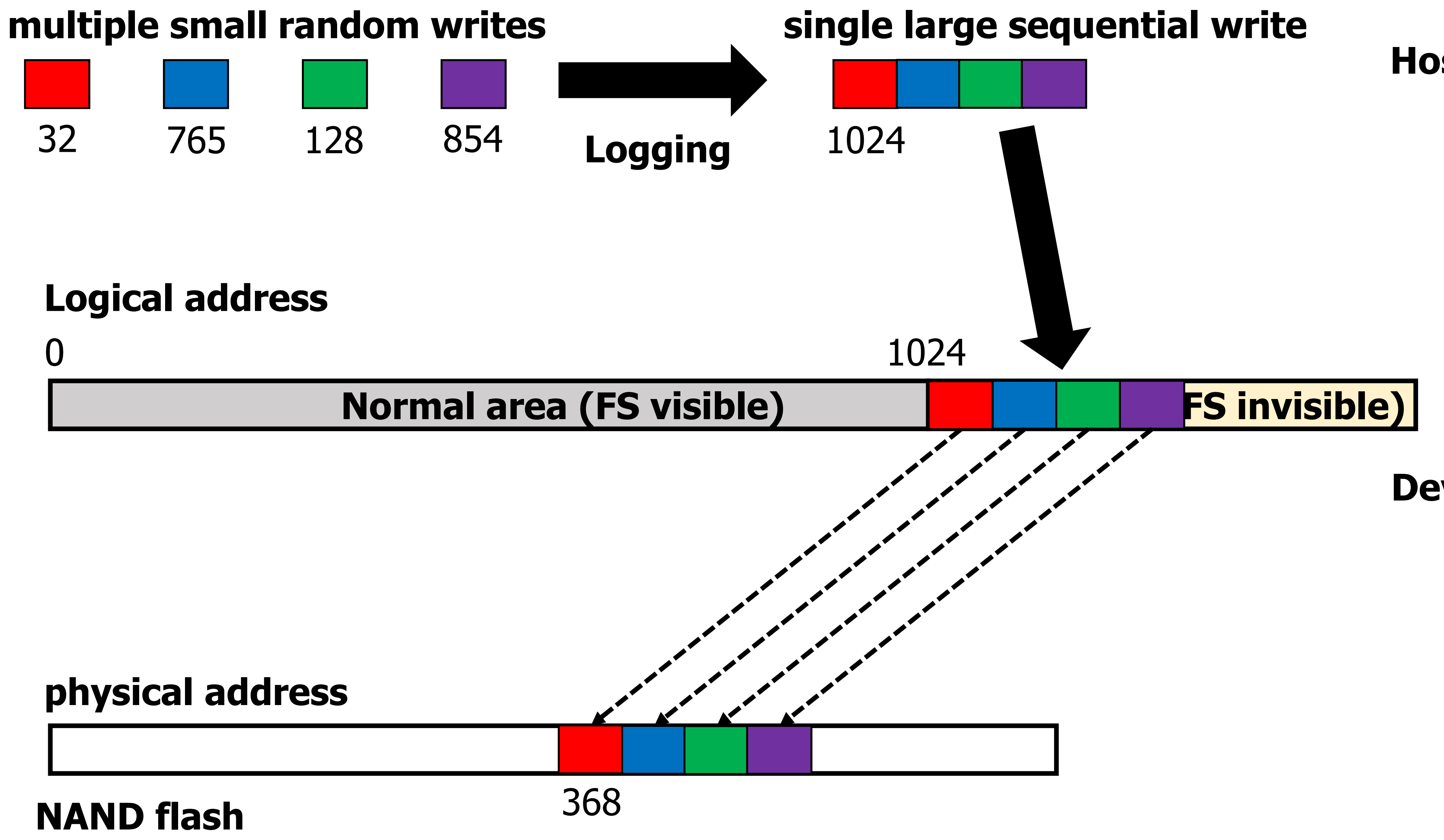
SHRD (Sequentializing in Host, Randomizing in Device)

- **Sequentializing in Host**
  - Host OS writes random requests sequentially at log area
- **Randomizing in Device**
  - SSD modifies the mapping table to change the logical address



SHRD Example: write

oLPN: original LPN
tLPN: temporal LPN



Host redirection table

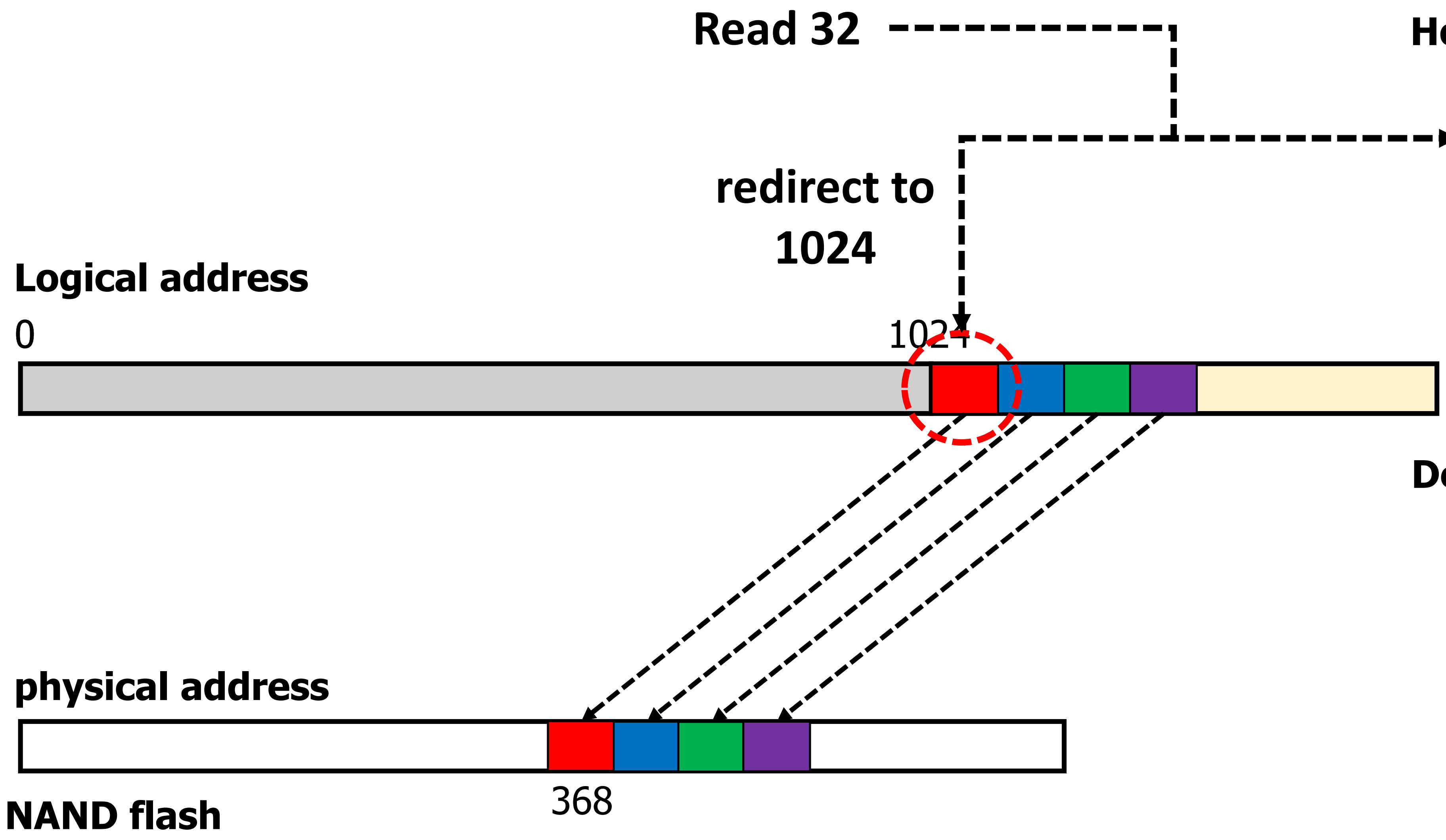
oLPN	tLPN
32	1024
128	1026
765	1025
854	1027

Device mapping table

LPN	PPN
1024	368
1025	369
1026	370
1027	371

SHRD Example: read redirection

oLPN: original LPN
tLPN: temporal LPN



Host redirection table

oLPN	tLPN
32	1024
128	1026
765	1025
854	1027

Device mapping table

LPN	PPN
1024	368
1025	369
1026	370
1027	371

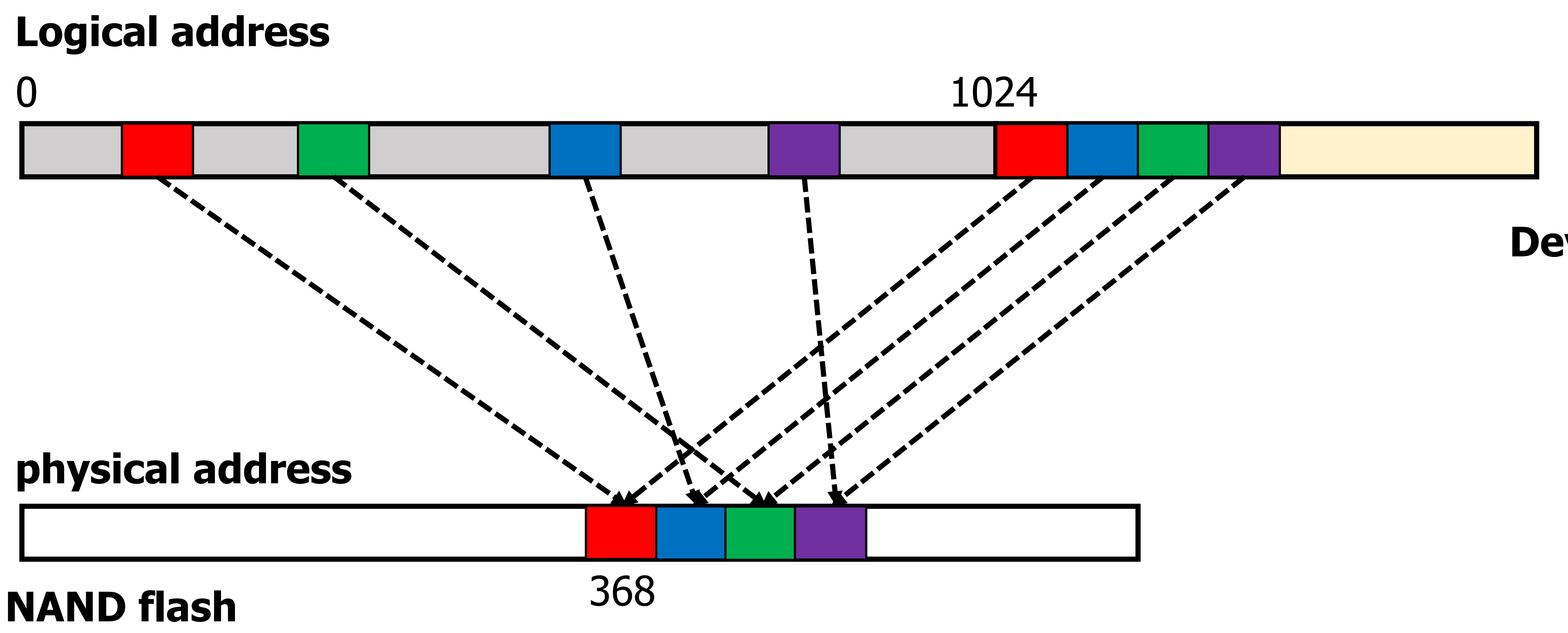
SHRD Example: remap

oLPN: original LPN
tLPN: temporal LPN

remap 1024-1027

Host redirection table

oLPN	tLPN

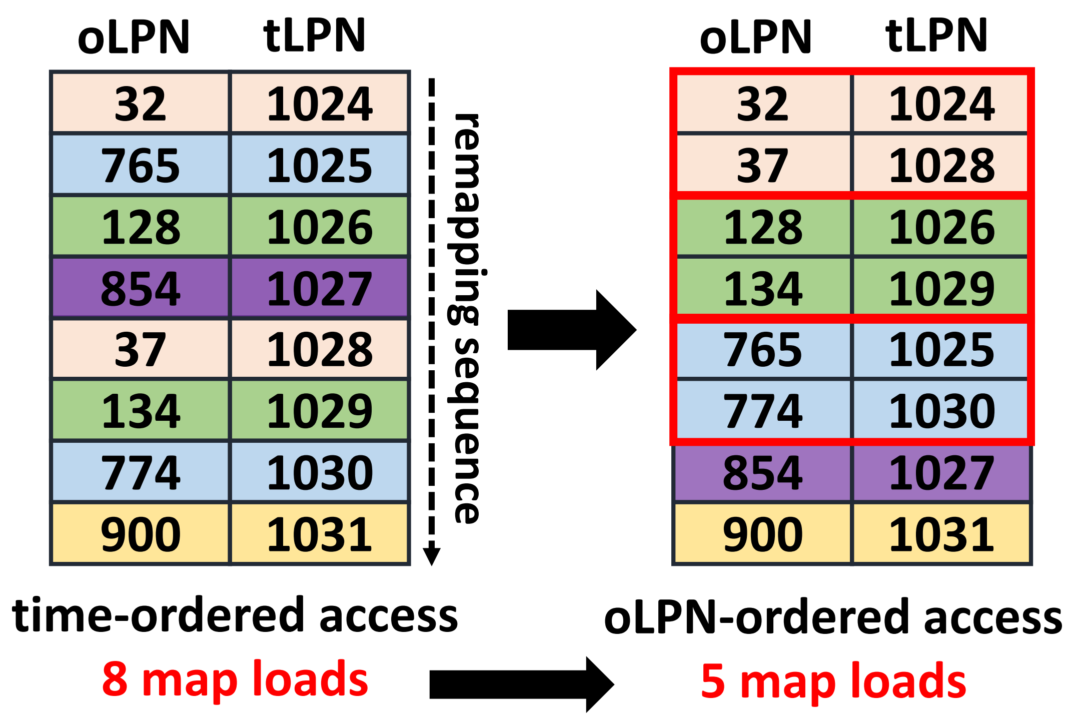


Device mapping table

LPN	PPN
32	368
765	369
128	370
854	371

Can we really reduce map loading overhead?

- Remap modifies the mapping entries of sequentialized pages
  - A time-ordered access scheme inherits the original random pattern
    - low spatial locality
- **oLPN-ordered map access**
  - The mapping table is oLPN-indexed
  - Can increase spatial locality



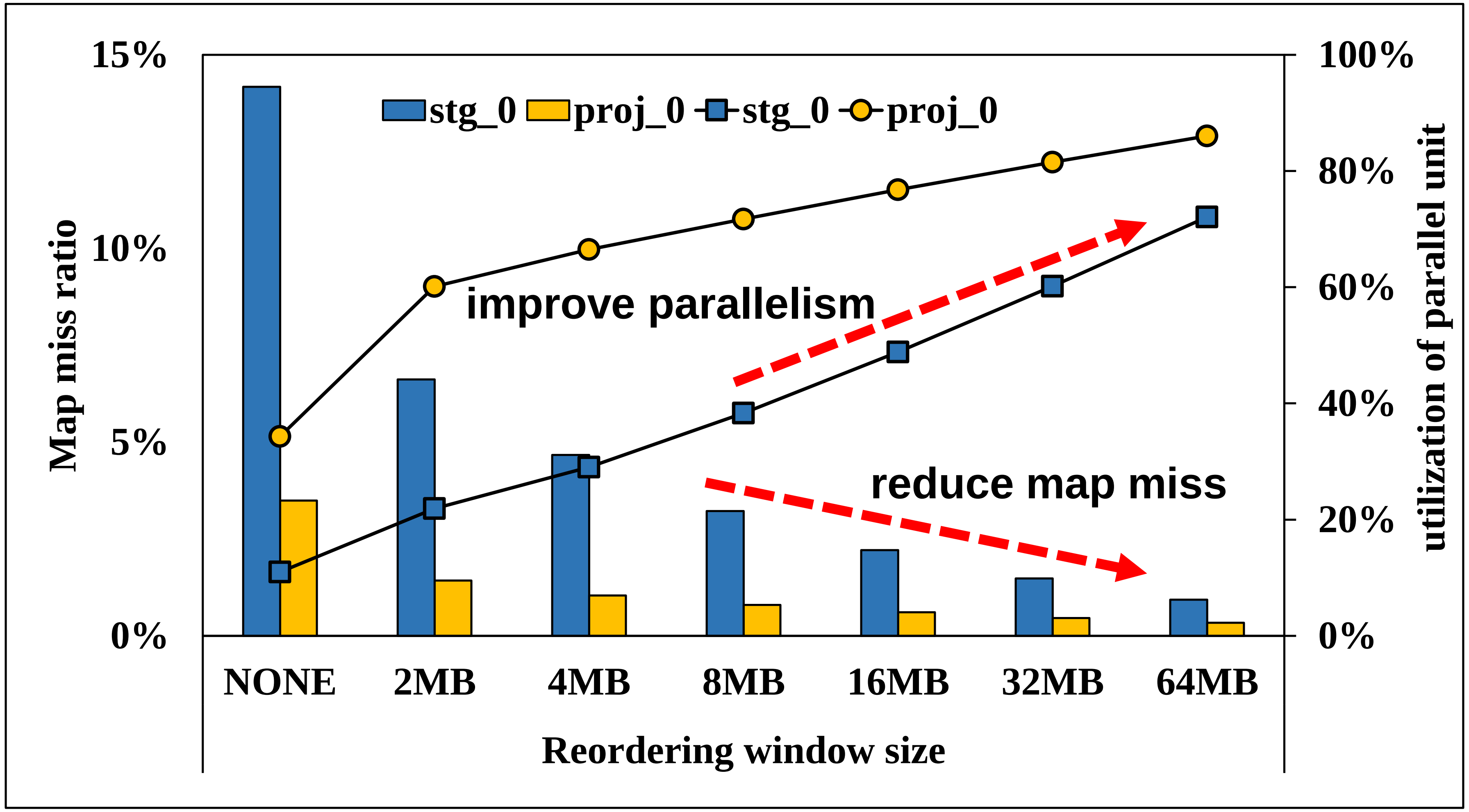
oLPN	tLPN
32	1024
765	1025
128	1026
854	1027
37	1028
134	1029
774	1030
900	1031

time-ordered access
8 map loads

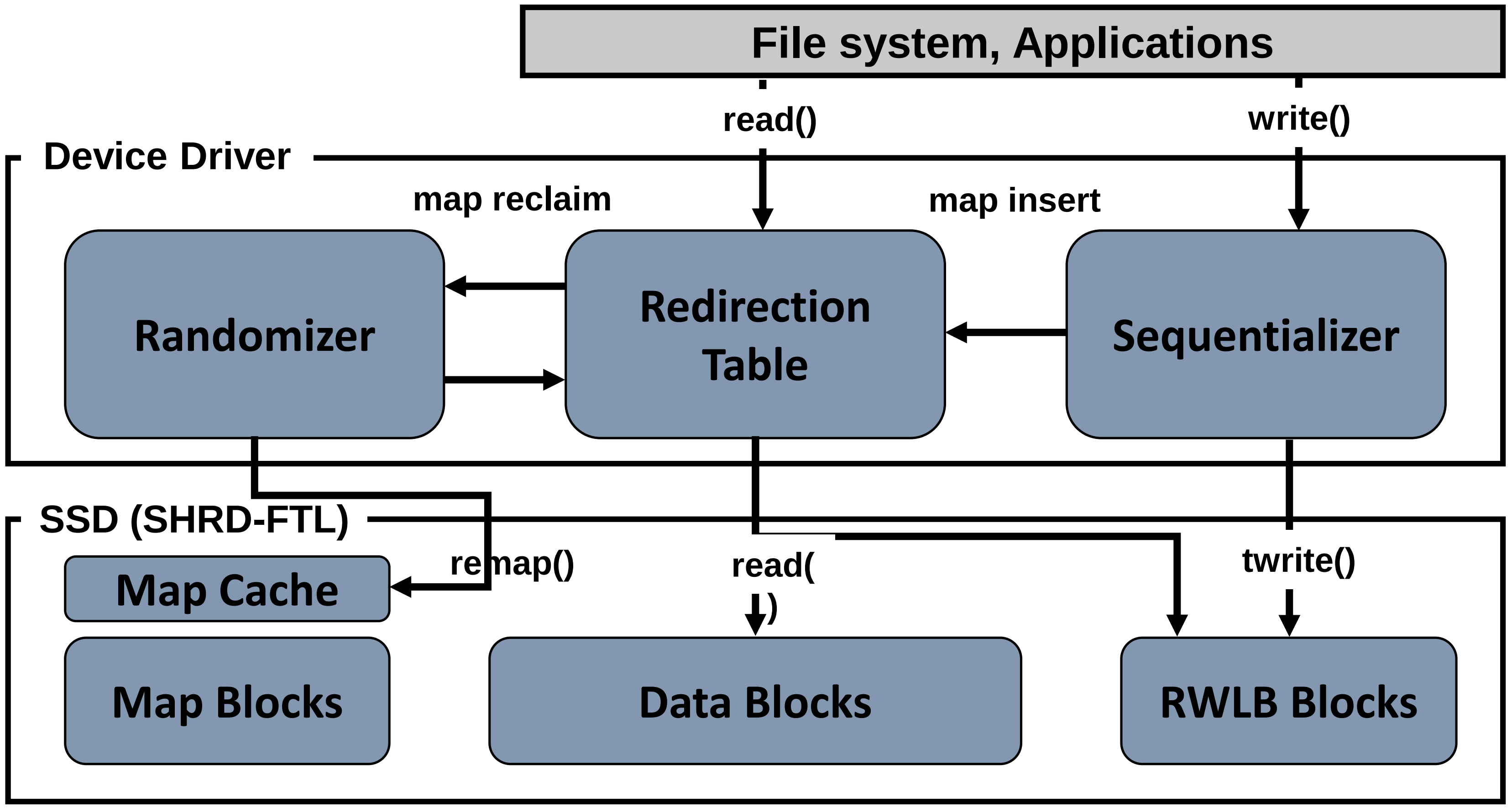
oLPN	tLPN
32	1024
37	1028
128	1026
134	1029
765	1025
774	1030
854	1027
900	1031

oLPN-ordered access
5 map loads

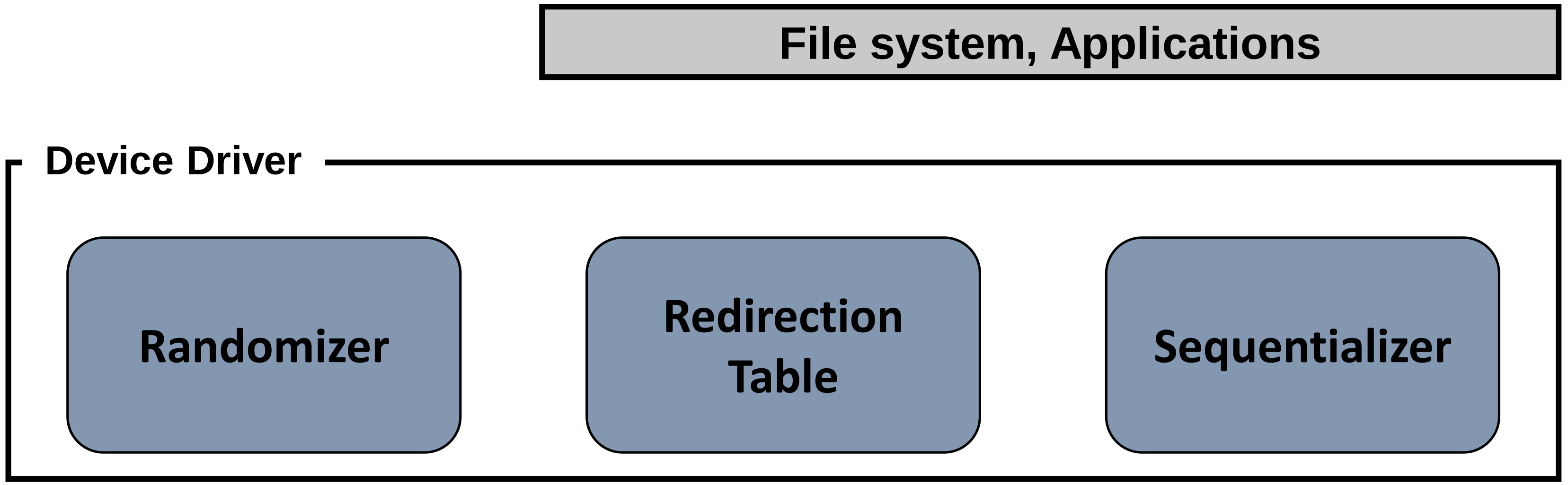
The effect of request reordering



SHRD (Sequentializing in Host, Randomizing in Device)



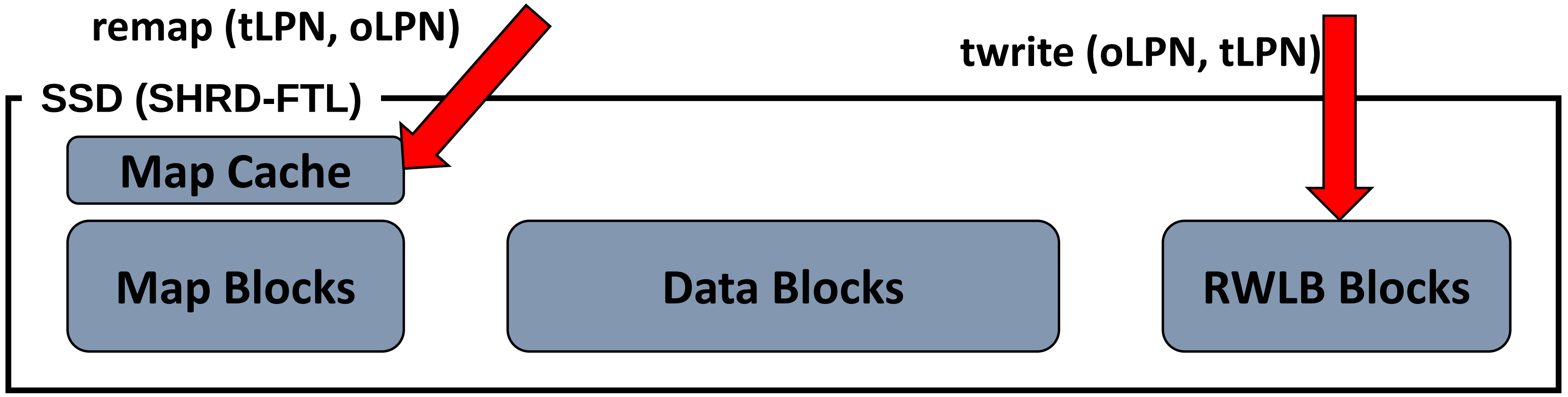
SHRD (Sequentializing in Host, Randomizing in Device)



- **Sequentializer**
  - Gathers random write requests, sequentially logs into temporal location
- **Redirection table**
  - Maintains redirection table between temporal address and original address
- **Randomizer**
  - Sends *remap* command to storage device and reclaims temporal location

SHRD (Sequentializing in Host, Randomizing in Device)

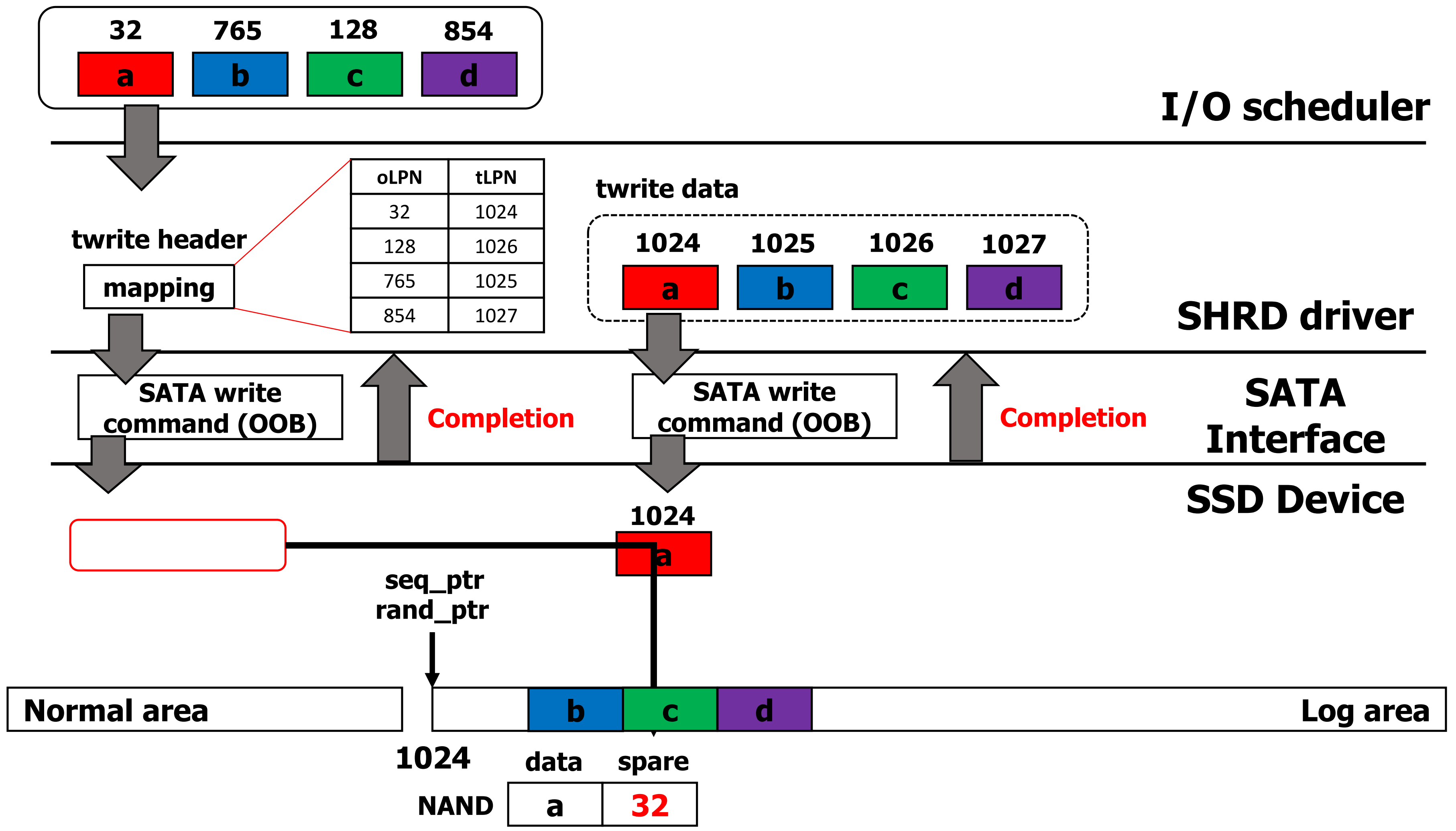
- **SHRD-FTL**
  - Receives *twrite* and *remap* command from host OS
- **twrite**
  - Write command with two addresses, **temporal/original** address
  - The data must be stored into separate physical blocks called **RWLB**
- **remap**
  - Restores the data written at temporal location into original address
  - Changes mapping table from temporal address to original address
  - Corresponding RWLB blocks will be transferred into data blocks



Special commands: twrite & remap

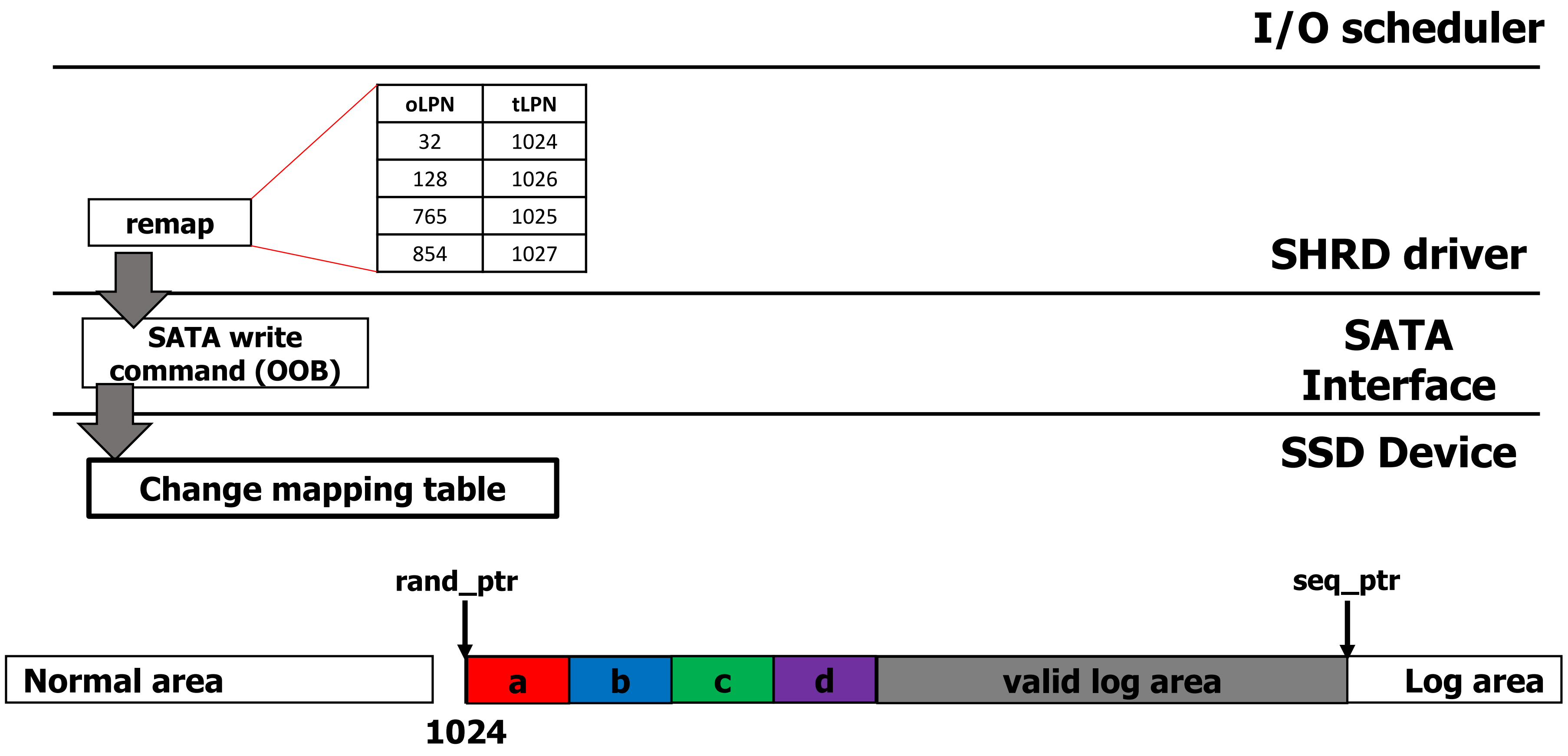
- **twrite (oLPN[n], tLPN_start, n, data)**
  - Write command sends two addresses, (**tLPN**, **oLPN**)
  - oLPN is stored at the OOB area of physical page
    - used for power-off-recovery / GC
  - Packed command with multiple RW requests
- **remap (oLPN[m], tLPN[m], m)**
  - m = # of remapping entries per remap command
    - oLPN-sorted entries ➔ Improving spatial locality
  - Changes mapping table from tLPN to oLPN
    - tLPN : PPN ➔ oLPN : PPN

Command Sequence: Sequentializing in Host



oLPN	tLPN
32	1024
128	1026
765	1025
854	1027

Command Sequence: Randomizing in Device

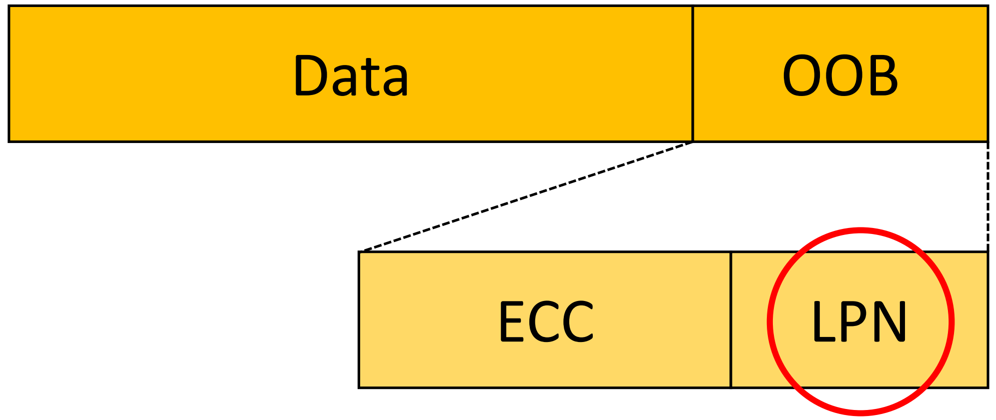


oLPN	tLPN
32	1024
128	1026
765	1025
854	1027

GC & Power Off Recovery (POR)

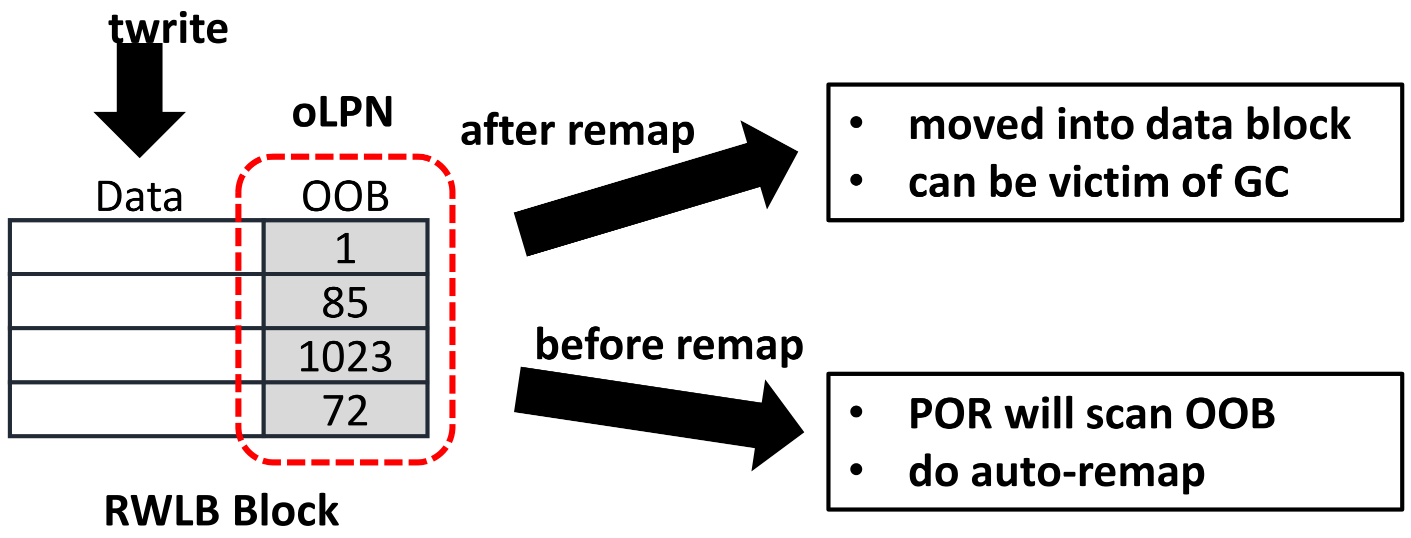
- Reverse map in out-of-band (OOB) area
  - SSD stores corresponding LPN in OOB area
  - Reverse map is used for GC & recovery
    - GC: change the mapping table of victim valid page
    - Recovery: recover the mapping table of active blocks

Physical page layout



GC & Power Off Recovery (POR)

- Store oLPN at the OOB area of RWLB
  - RWLB blocks must be excluded from choosing victim
    - until entire data stored in the blocks are remapped
  - Non-remapped data will be **auto-remapped** at POR
    - by scanning the OOB area of RWLB blocks

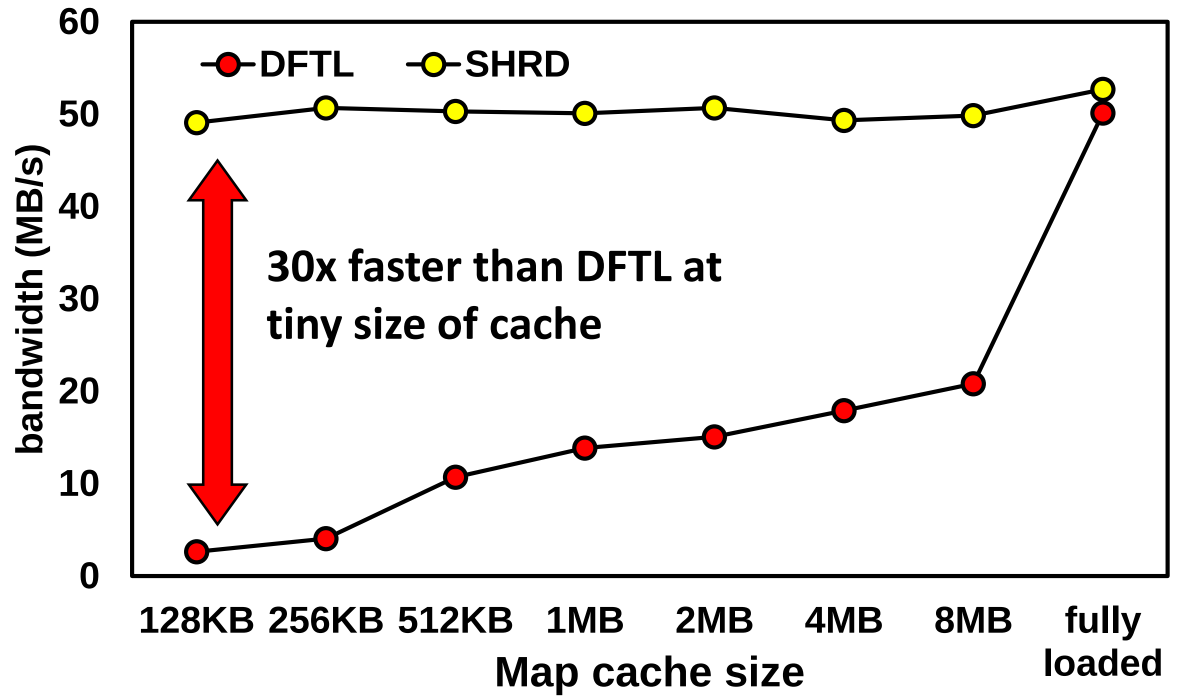


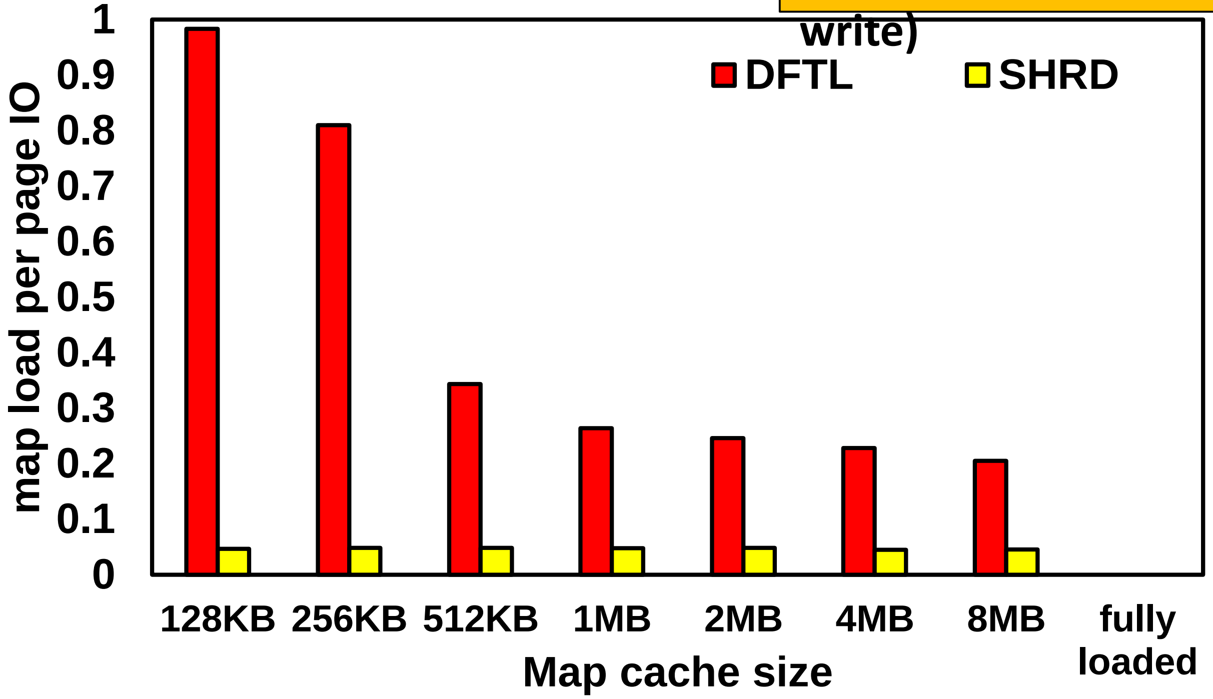
Implementation

- SHRD D/D is implemented in Linux kernel 3. 17.4
  - Additional kernel module at SCSI D/D layer
  - Host redirection table: about 1 MB for 64 MB log area
- Prototype SSD device
  - Modified the firmware of a commercial SATA3 SSD device (Samsung 843)
  - DFTL & SHRD-FTL are implemented
  - Map cache size is configurable

RW Performance According to cache

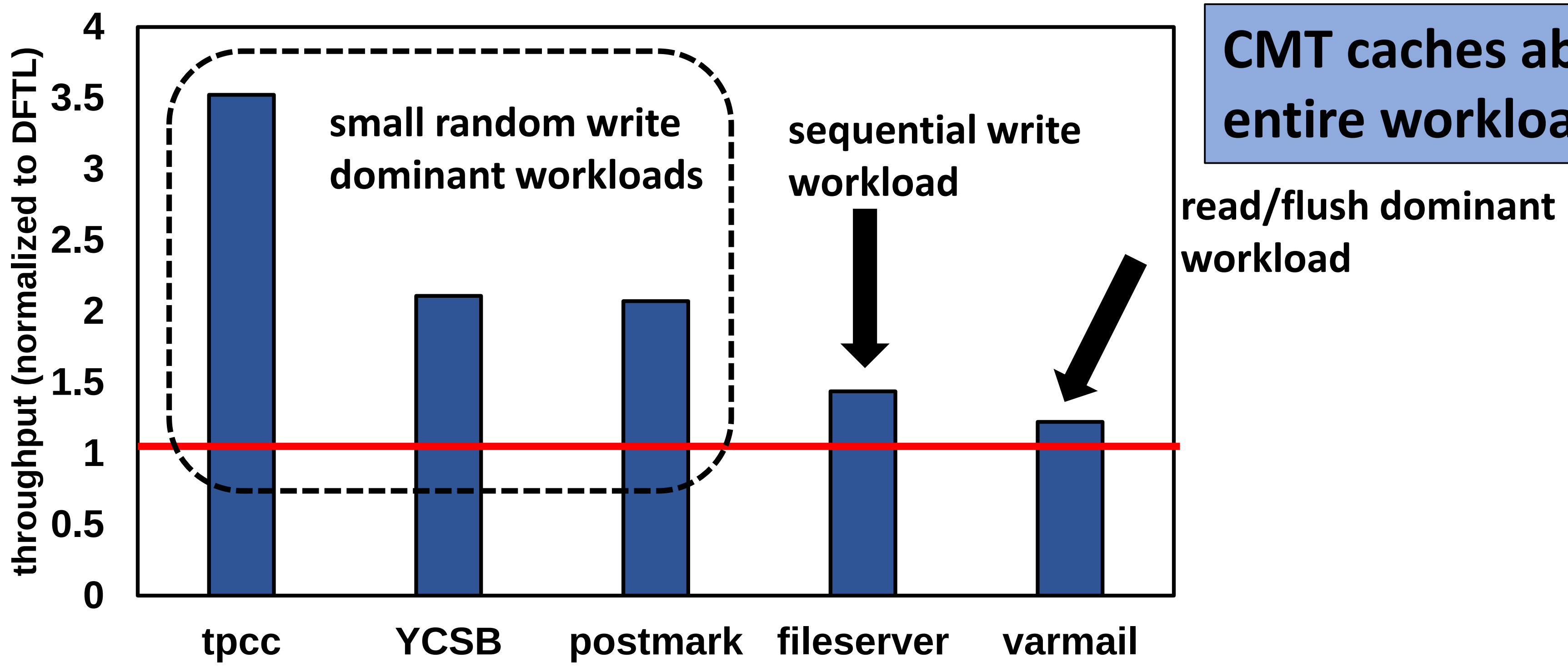
fio random write test (32GB space, 4KB write)





- Better performance than DFTL
  - By reducing **map loading/unloading overhead**
- SHRD shows steady performance regardless of cache size

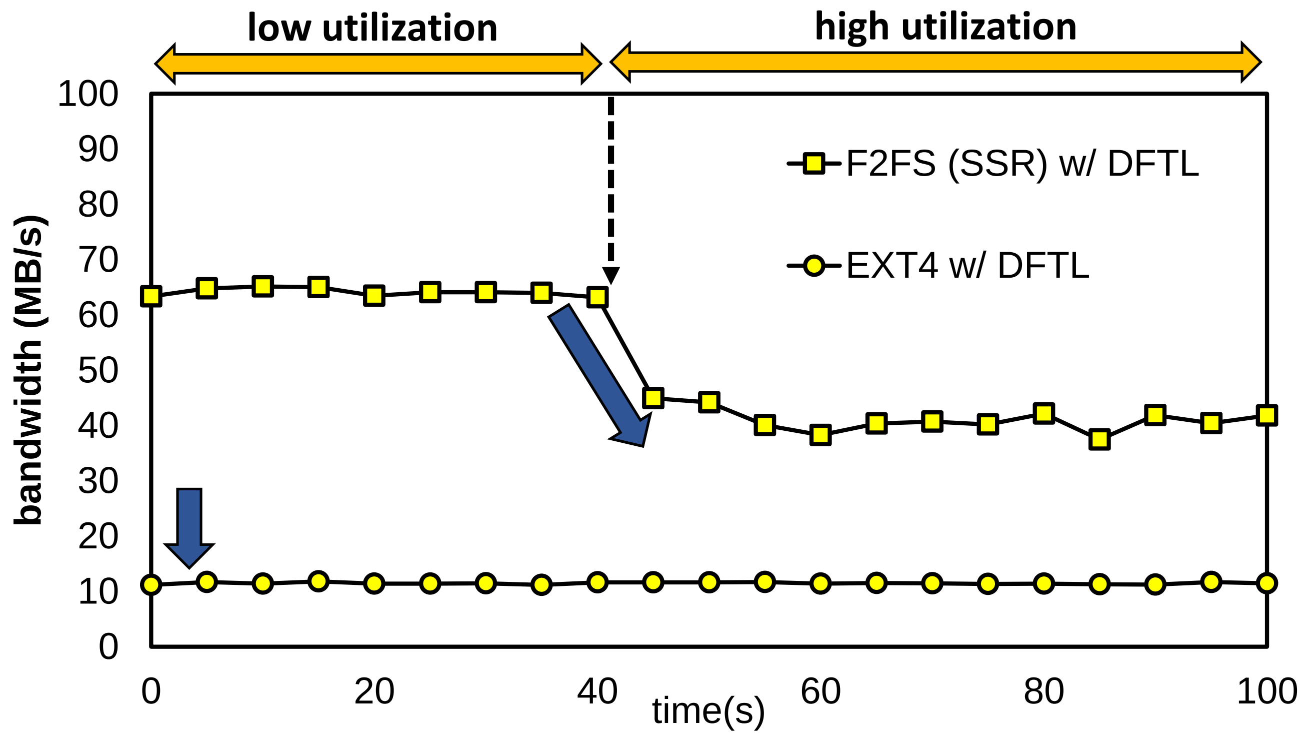
Performance on Real Benchmarks



CMT caches about 5% of entire workload space

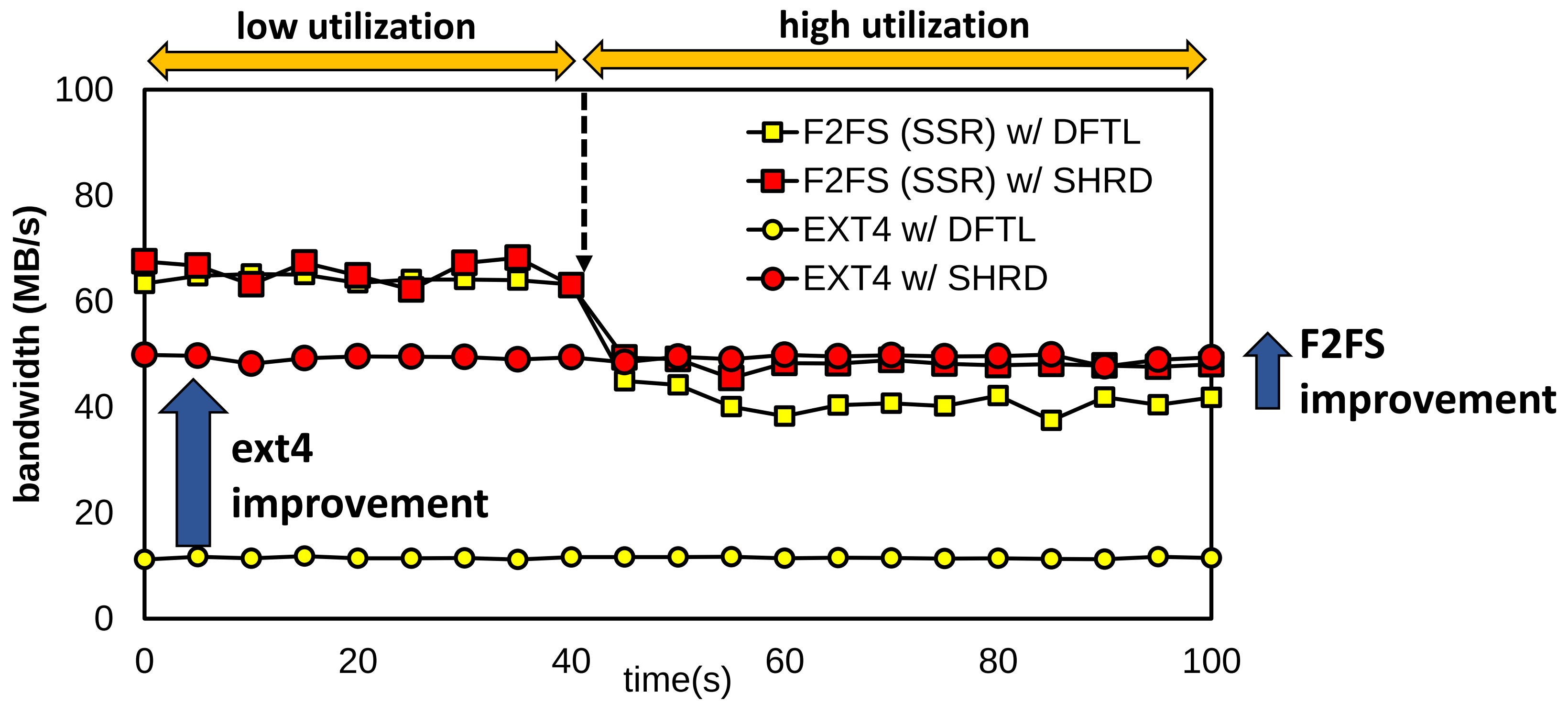
- Better performance at all workloads
- Small gains at sequential or read dominant workload
    - still better than DFTL

SHRD gains at EXT4 vs. F2FS



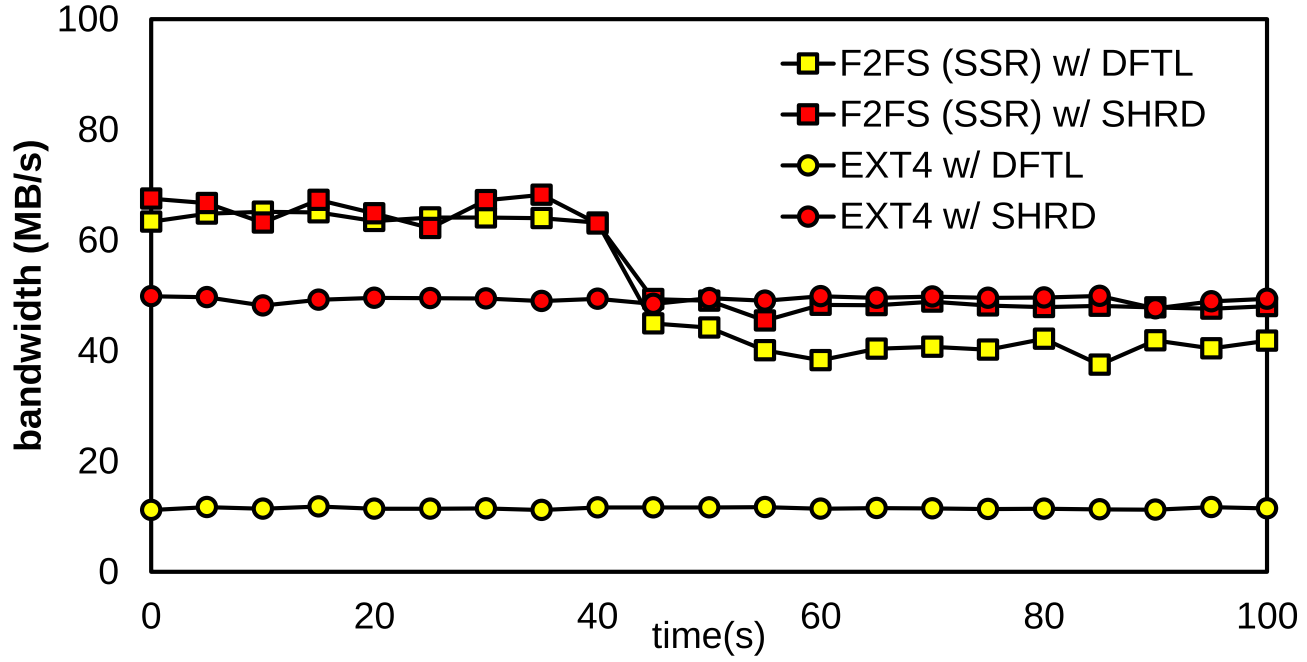
- EXT4 shows bad performance on random write
- Performance of F2FS decreases due to SSR at high utilization

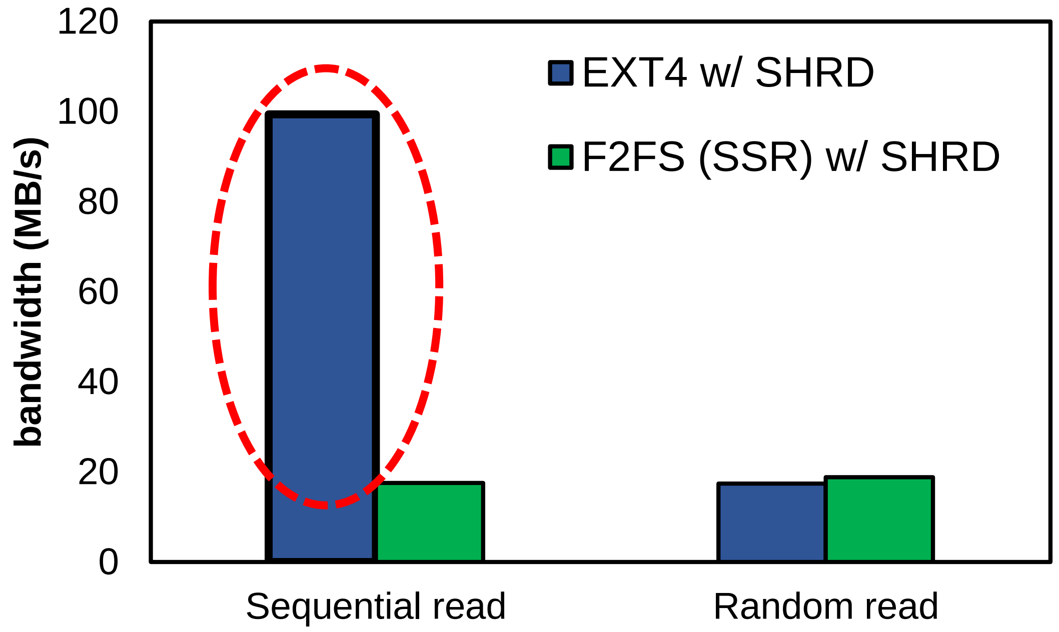
SHRD gains at EXT4 vs. F2FS



- SHRD improves both EXT4 and F2FS
  - SHRD improves the bandwidth of aged F2FS
  - EXT4 shows similar performance as F2FS by using SHRD

SHRD gains at EXT4 vs. F2FS

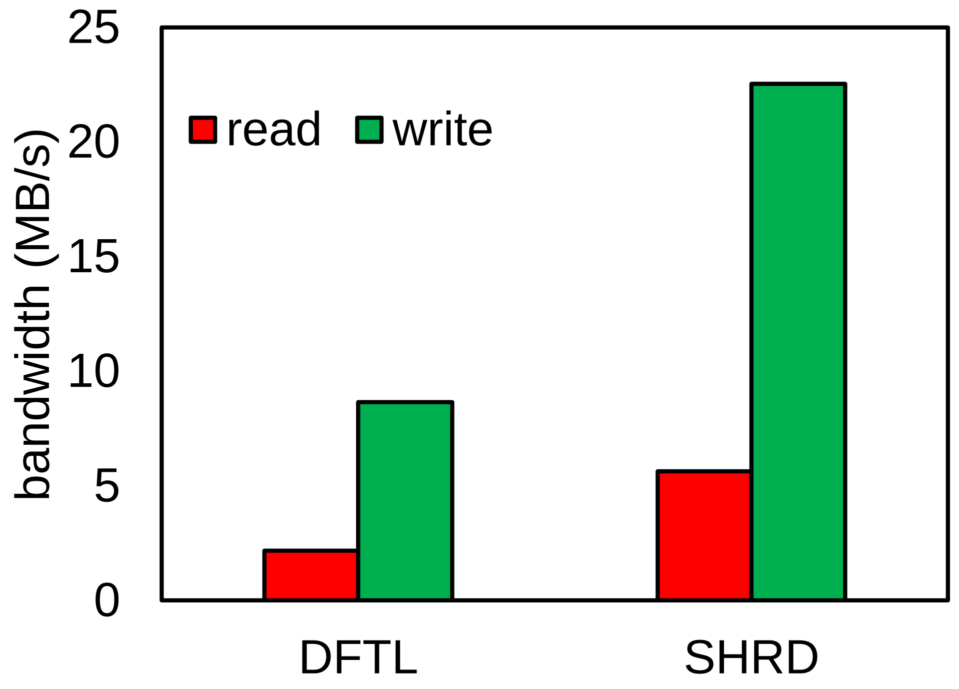


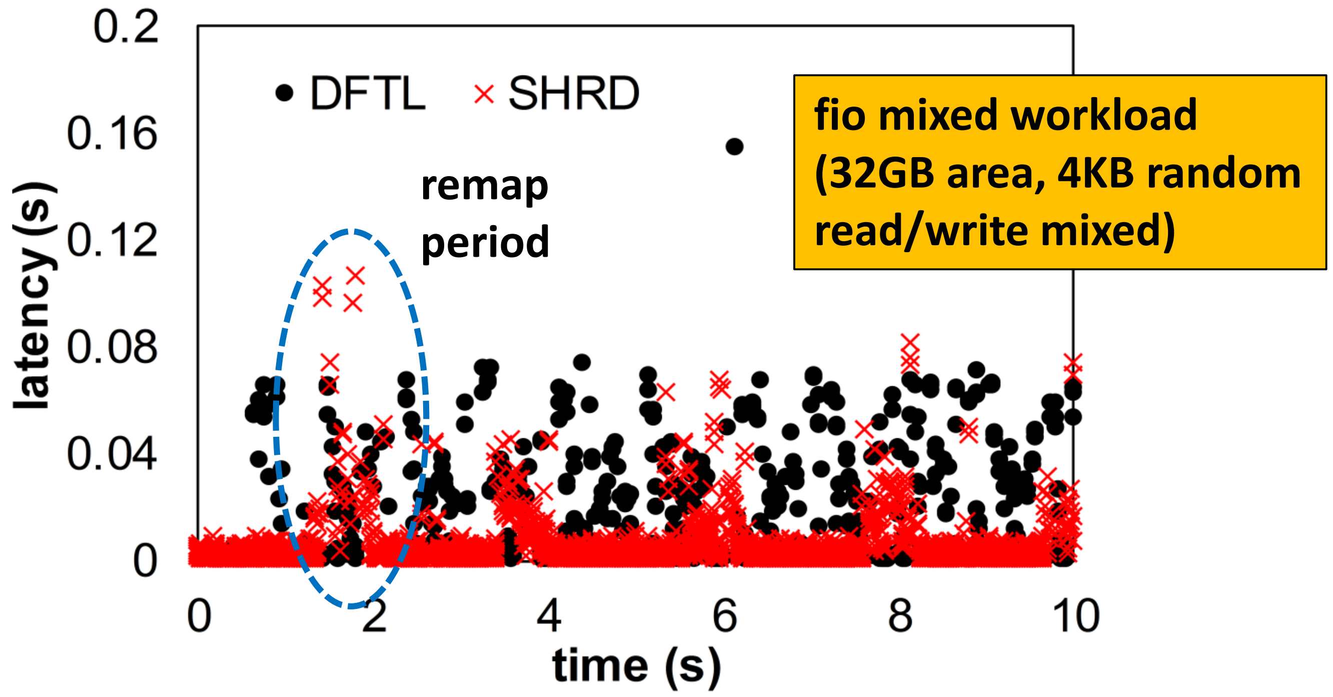


- **Sequential read performance of EXT4 is much better**
  - The out-of-place scheme of F2FS scatters the data blocks of a file

Latency Comparison

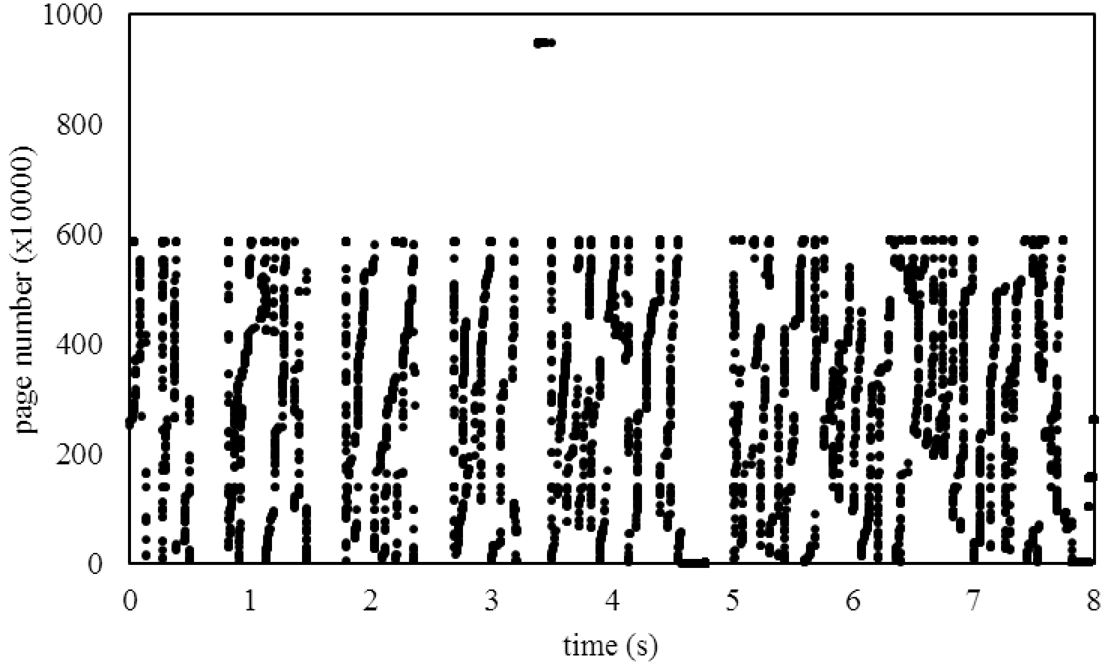
- Remap command can delay read operations
  - Several remapping entries are batched into a single command
  - # of remapping entries per command can control the maximum latency of following I/O operations



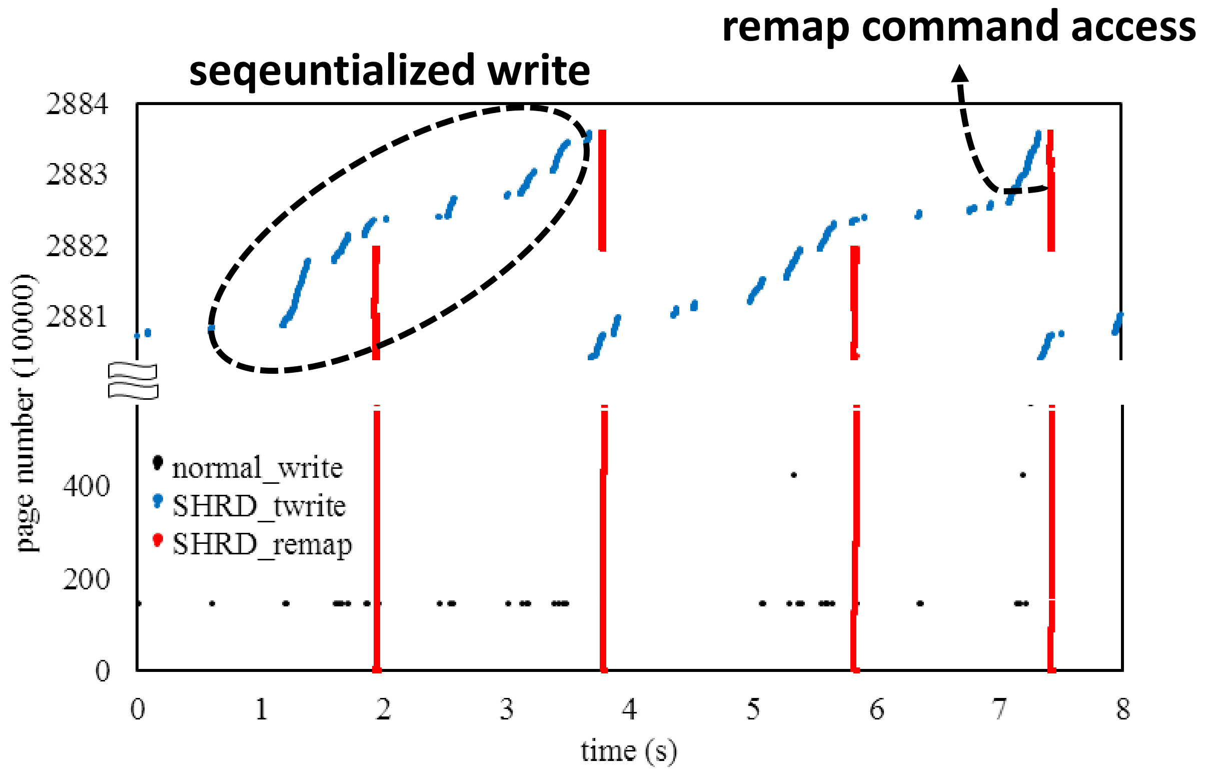


fio mixed workload (32GB area, 4KB random read/write mixed)

Visualizing address accessing pattern: postmark



< without SHRD >



< with SHRD >

Conclusion

- SHRD is an address reshaping technique
  - transforms RW into SW at the block D/D
  - restores the original addresses without copy operations
  - Solves POR / GC issues of address remapping

- SHRD improves 30x better performance at a small map cache
  - reduces DRAM drastically

Thank you.