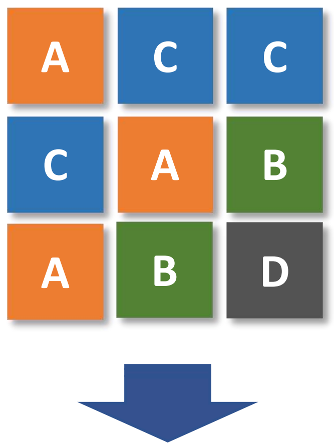
Remap-SSD: Safely and Efficiently Exploiting SSD Address Remapping to Eliminate Duplicate Writes

You Zhou[1], Qiulin Wu[1], Fei Wu[1], Hong Jiang[2],
Jian Zhou[1], and Changsheng Xie[1]

[1] Huazhong University of Science and Technology (HUST), China
[2] University of Texas at Arlington (UTA), USA

FAST'21
19th USENIX Conference on File and Storage Technologies

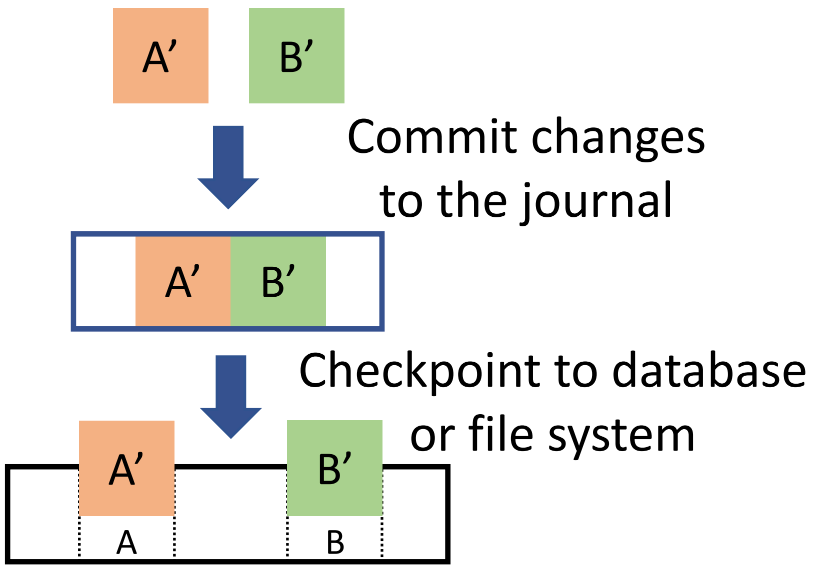
Duplicate Writes are Everywhere



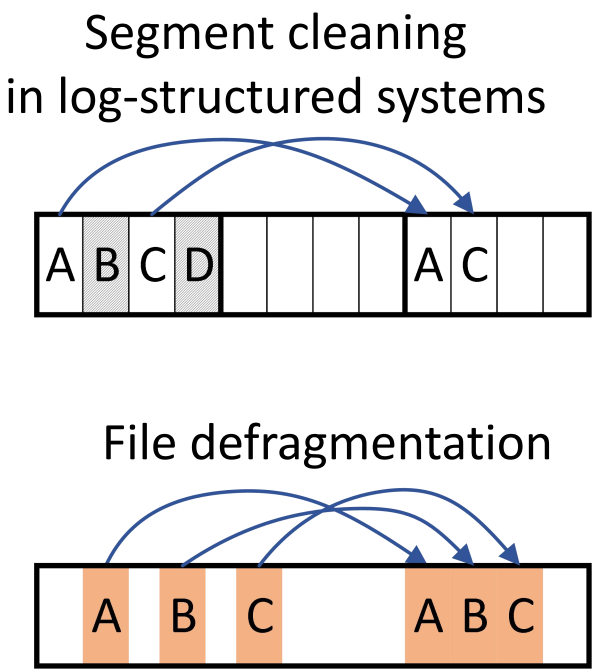
6% ~ 92% duplicate writes

[CAFTL & CASSD (FAST’11), SmartDedup (ATC’19)]

Data Duplication



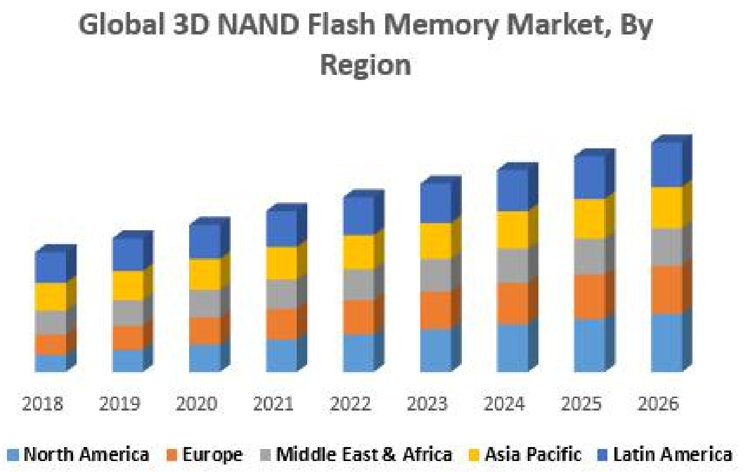
Double-write Journaling



Data Relocations

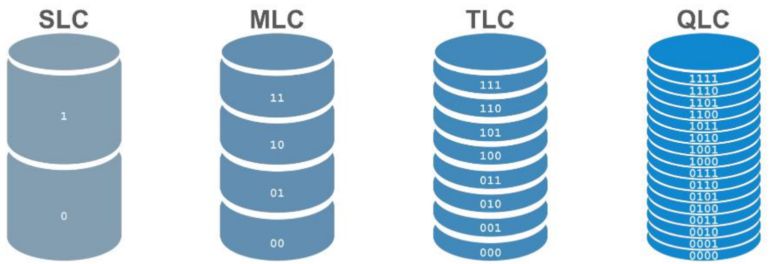
Writes Degrade SSD Performance and Lifetime

Flash SSDs are everywhere

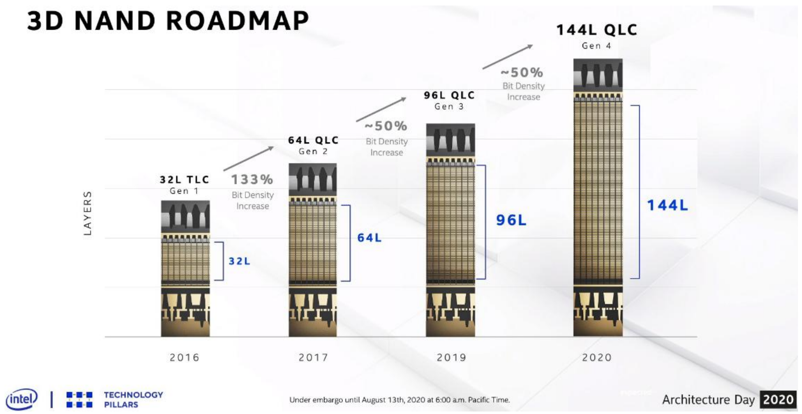


Source: MAXIMIZE MARKET RESEARCH PVT. LTD.

Lower write speed & endurance



Write/Erase cycles: ~100,000 → 10,000 → 5,000 → 1,000



Lower cost & higher density

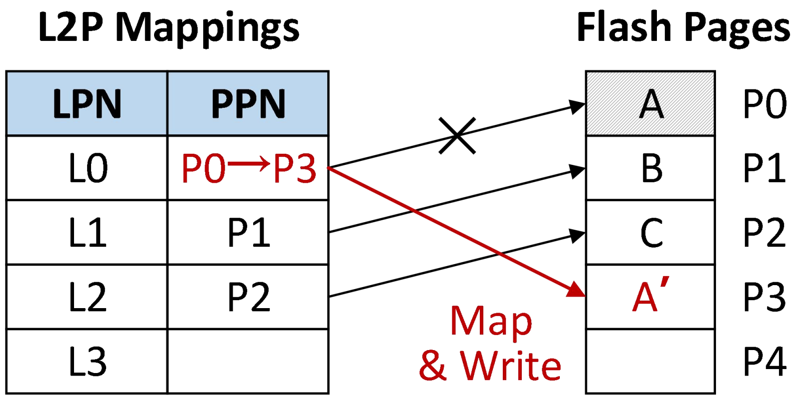
Eliminate duplicate writes on flash

↓

Improve SSD performance & lifetime

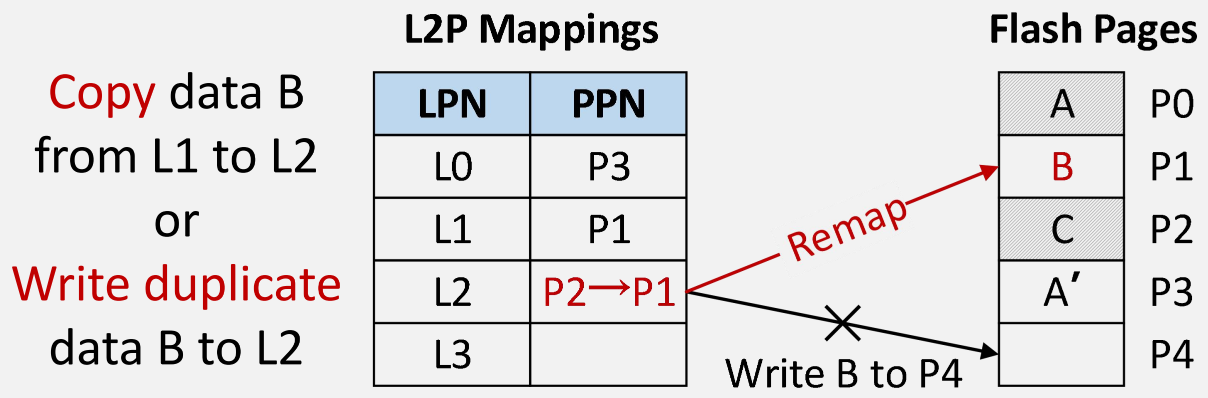
Eliminating Duplicate Writes on Flash

Host logical to flash physical address mapping inside SSD

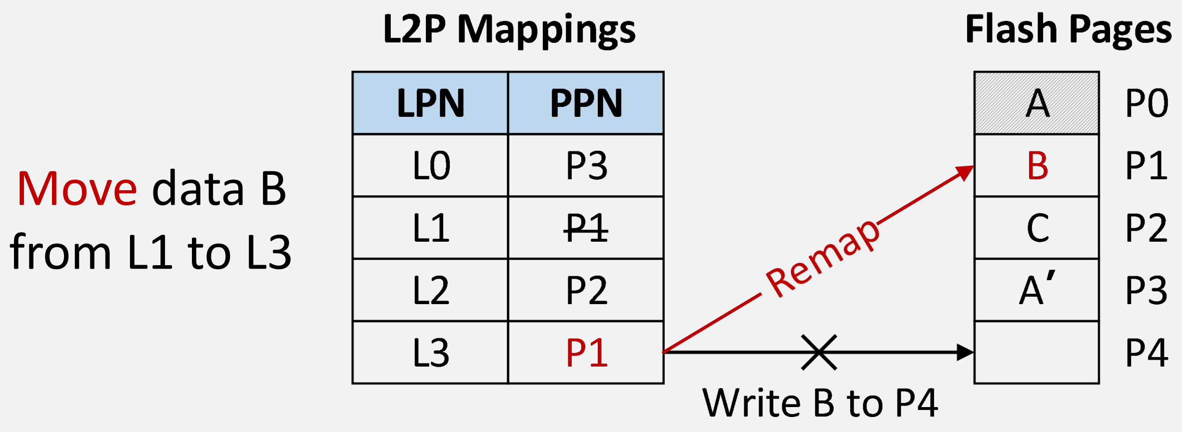


~~Overwrite L0 on flash page P0~~

Update logical page L1 out of place



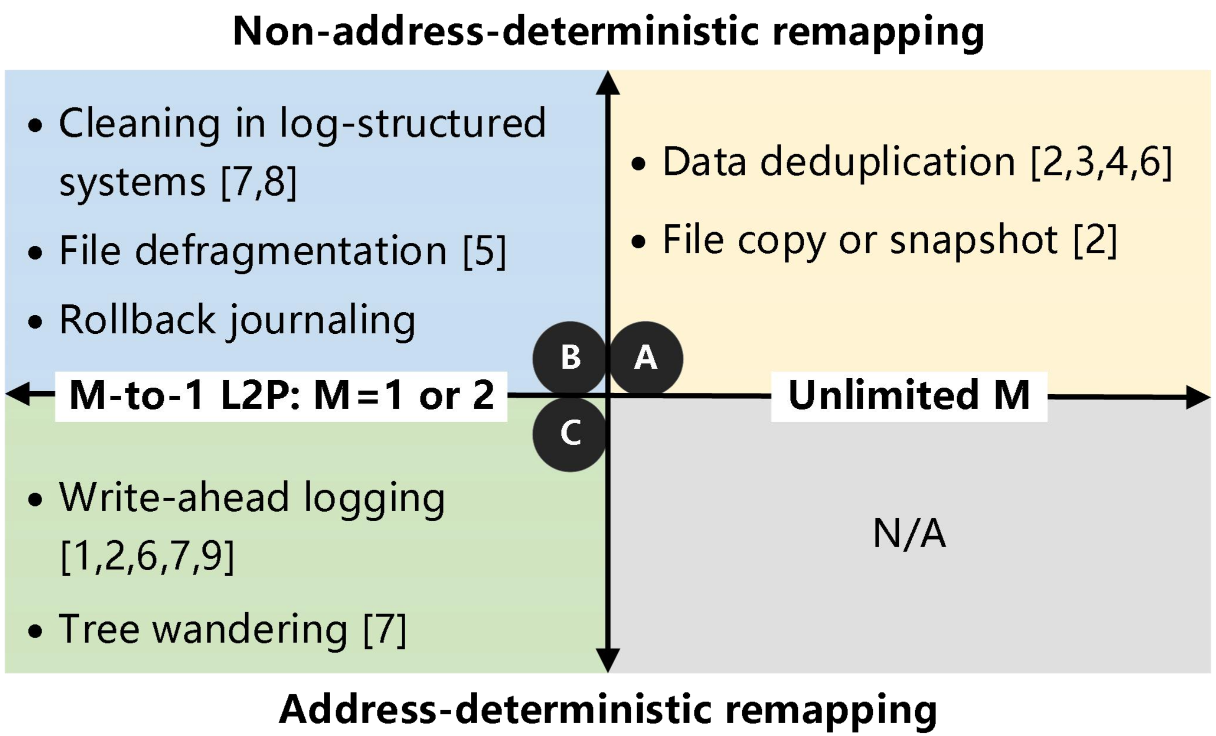
Replace duplicate writes with remappings



LPN: host logical page number
PPN: flash physical page number
L2P: logical-to-physical

Prior Studies Exploiting SSD Address Remapping

- **Various application scenarios of remapping can be classified in two dimensions**
  - M-to-1 L2P mappings: M is limited or unlimited
  - Target addresses of remappings are predetermined or uncertain
    - Write-ahead logging is C type: M = 2, remap data from log to predetermined home locations
    - Data deduplication is A type: M and addresses of remappings depend on workloads



[1] JFTL (TOS 2009)
[2] ANViL (FAST'15)
[3] CAFTL (FAST'11)
[4] CA-SSD (FAST'11)
[5] Janusd (ATC'17)
[6] Copyless copy (HPCC'19)
[7] SHARE (SIGMOD'16)
[8] PebbleSSD (MEMSYS'17)
[9] WAL-SSD (TC 2020)

Outline

- Introduction
- Motivation
- Design of Remap-SSD
- Case Studies and Evaluation
- Conclusion

Mapping Inconsistency Problem with Remapping

L2P and P2L mappings in SSD

L2P Mappings in DRAM

LPN	PPN
L0	P3
L1	P1
L2	P2
L3	

Flash Pages w/ P2L Mappings

Data	OOB	
A	L0, t0	P0
B	L1, t1	P1
C	L2, t2	P2
A′	L0, t3	P3
		P4

OOB: out-of-band area

P2L mappings are necessary for

- data migrations between flash pages for **garbage collection (GC)** and wear leveling
  - locate and modify relevant L2P mappings
- power-off recovery (POR): restore L2P mappings

Remap L2 to P1 for copying data B from L1 to L2

L2P Mappings in DRAM

LPN	PPN
L0	P3
L1	P1
L2	P2→P1
L3	

Remap→

Flash Pages w/ P2L Mappings

Data	OOB	
A	L0, t0	P0
B	L1, t1	P1
C	L2, t2	P2
A′	L0, t3	P3

{L1, L2} → P1
vs.
P1 → L1

Inconsistent L2P and P2L mappings

Corrupted L2P mappings after GC/POR

GC migrates data B to P1′

LPN	PPN
L1	P1→P1′
L2	P1 (erased)

Recover L2P via P2L

LPN	PPN
L1	P1
L2	P2 (invalid)

Prior Schemes for Mapping Consistency

L2P Mappings in DRAM

LPN	PPN
L0	P3
L1	P1
L2	P2→P1
L3	

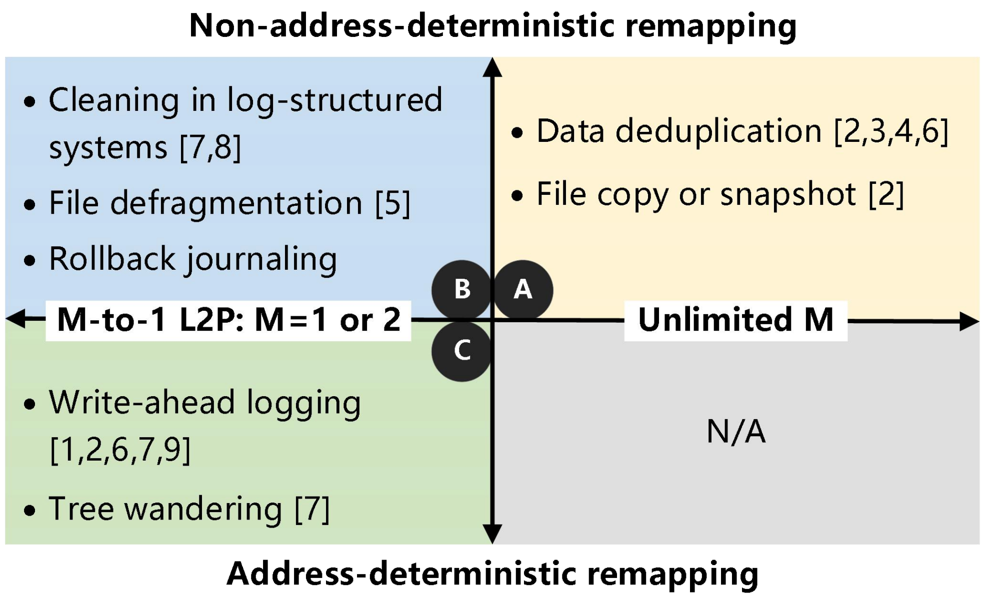
Remap

Flash Pages w/ P2L Mappings

Data	OOB	
A	L0, t0	P0
B	L1, t1	P1
C	L2, t2	P2
A'	L0, t3	P3

Inconsistent mappings after remapping L2 to L1 (P1)

{L1, L2} → P1
vs.
P1 → L1



1. Persist a device-wide log [3,4,5,6,7]

P1 → L2, t4

Remapping metadata entries

P, L, t	P, L, t	P, L, t	P, L, t		…

High lookup overheads & poor scalability

- Log size grows: ~1GB for 300GB remap data
  - Log scanning **in every GC** takes seconds
- Or limit the log size by disabling remapping

2. NVRAM OOB [8]

Flash Page P1

Data	NVRAM OOB
B	(L1, t1), **(L2, t4)**

Only fit in B & C-type remapping scenarios

3. Prewrite L2 to P1 OOB [9]

L2P Mappings

LPN	PPN
L1	P1
L2	P2→P1

Remap

Flash Pages

Data	OOB	
B	L1, t1, **L2**	P1
C	L2, t2	P2

Only fit in C-type scenarios
(remap L2 to P1: known in advance)

Motivation

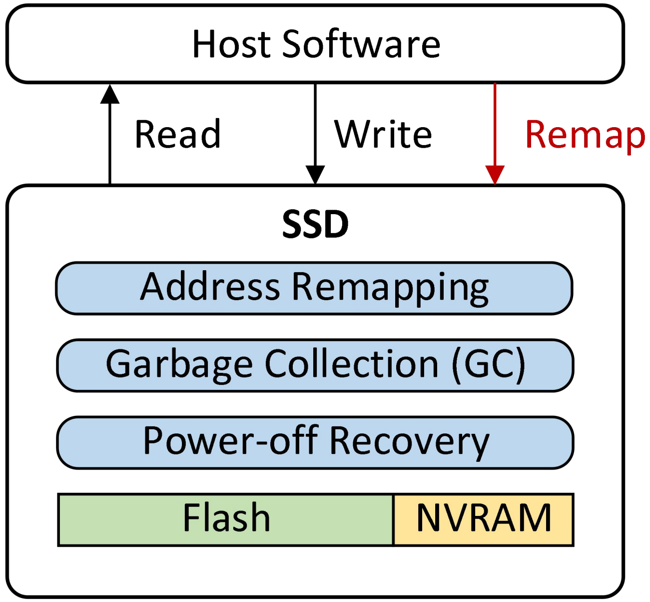
Since the mapping inconsistency problem cannot be addressed properly, existing schemes severely limit the usage of SSD address remapping.

Outline

- Introduction
- Motivation
- Design of Remap-SSD
- Case Studies and Evaluation
- Conclusion

Design Overview of Remap-SSD

Goal: full potentials of remapping



Data move (MV)

Move data from L1 to L2 at time t1

Remap(L2, L1, MV)

LPN	PPN
L1	P1 → null
L2	null → P1

P1, L2, L1, t1

- Modify L2P mappings in DRAM
- Persist remapping metadata entries

Data copy or Dedup (CP)

Write L4 with the same data as L3 at time t2

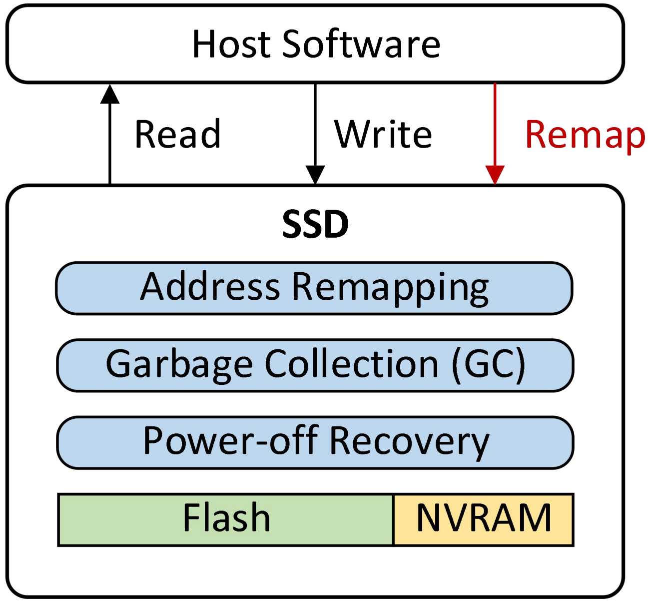
Remap(L4, L3, CP)

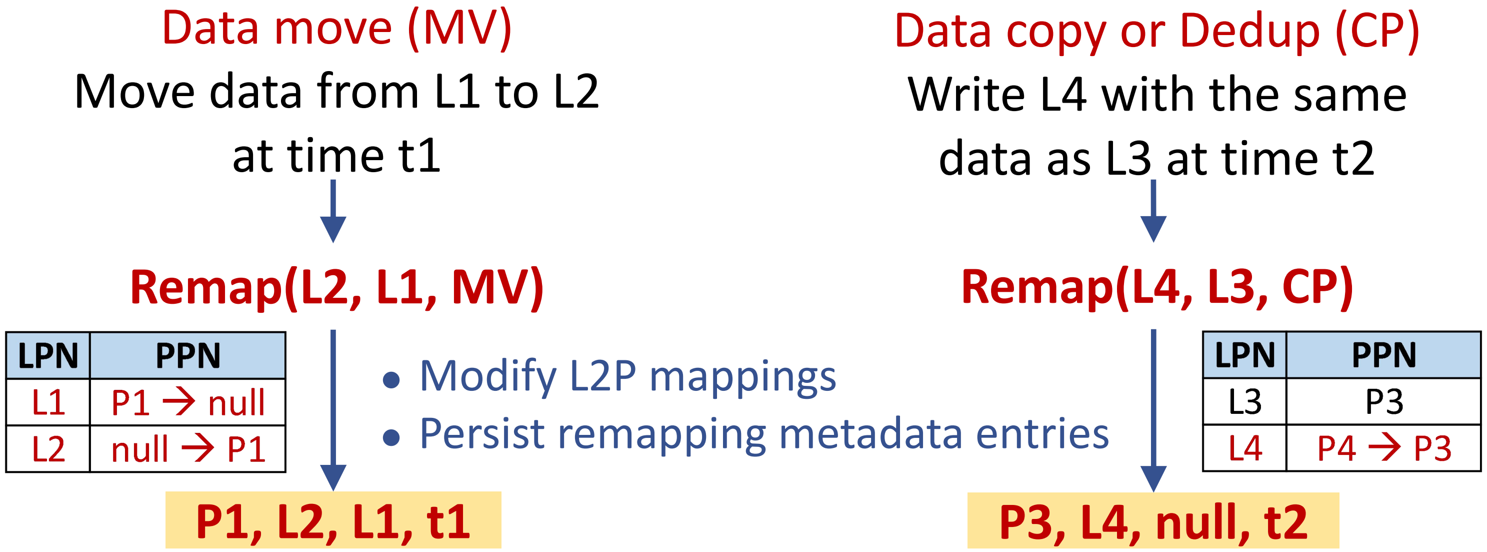
LPN	PPN
L3	P3
L4	P4 → P3

P3, L4, null, t2

Design Overview of Remap-SSD

Goal: full potentials of remapping

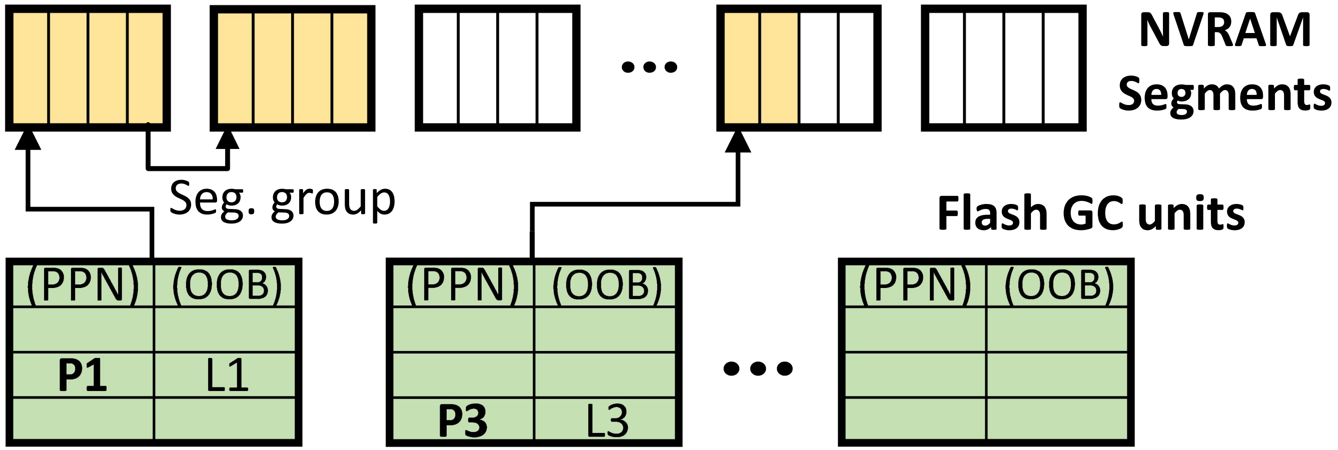




Lookup pattern of P2L mappings:

in a batch of flash pages in a GC unit

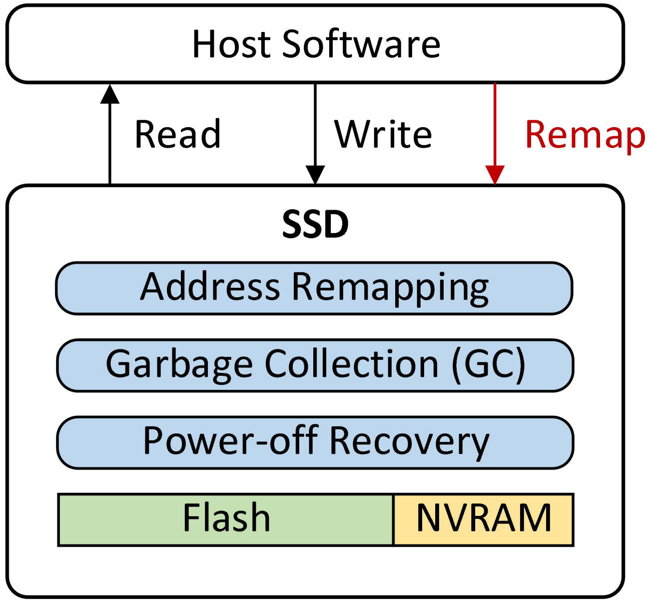
(e.g., flash block/superblock*)



*Superblock: a group of flash blocks with the same offset across flash dies.

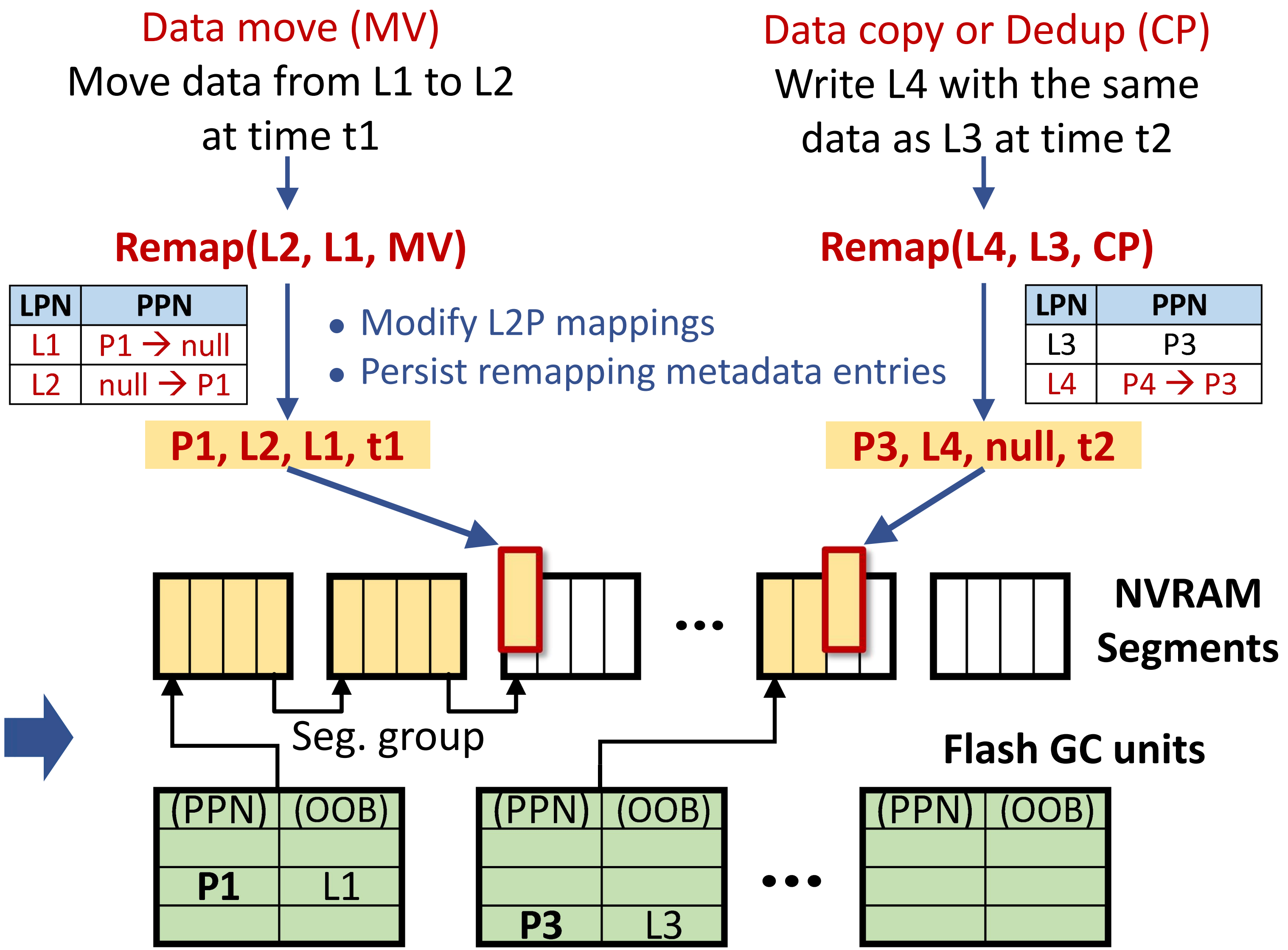
Design Overview of Remap-SSD

Goal: full potentials of remapping



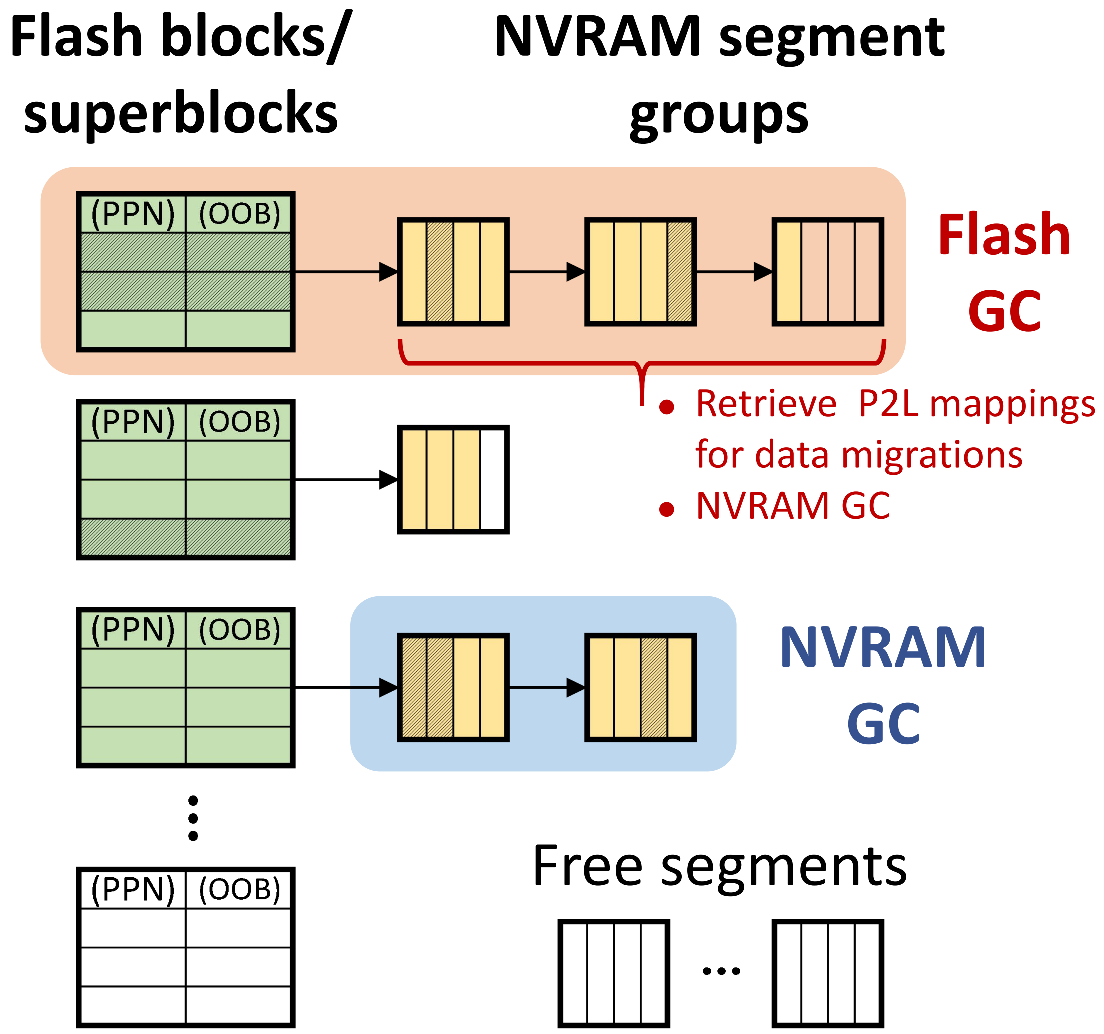
Lookup pattern of P2L mappings:
in a batch of flash pages in a GC unit
(e.g., flash block/superblock*)

*Superblock: a group of flash blocks with the same offset across flash dies.



GC and Power-off Recovery

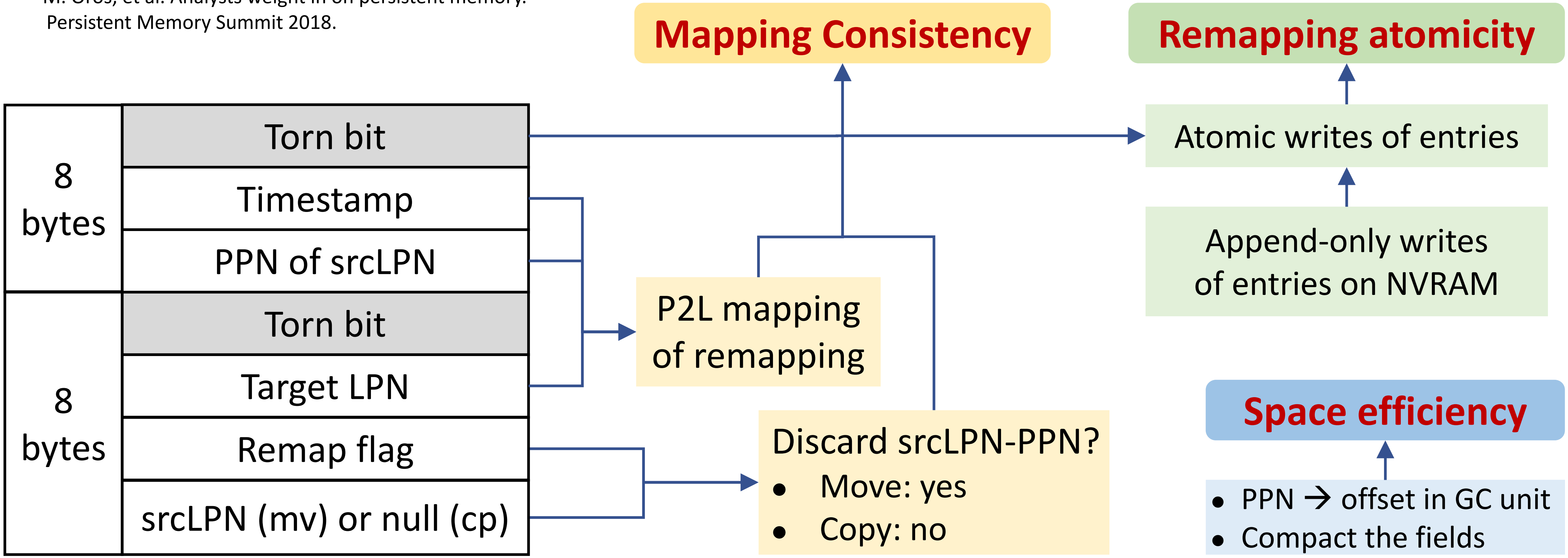
- **Flash GC for free blocks/superblocks**
  - Victim: block/superblock with the most invalid pages and its NVRAM seg. group
  - Fast lookups of P2L mappings in *a small segment group* (not a large device-wide log)
- **NVRAM GC for free segments**
  - Victim: NVRAM segment group with the most invalid remapping metadata entries
- **Power-off recovery**
  - Restore the latest L2P mappings from persistent P2L mappings and timestamps
    - in flash OOB (P2L from writes)
    - in NVRAM segments (P2L from remappings)



Remapping Metadata Entry

- Remap operation: `target LPN`, `source LPN(srcLPN)`, `remap flag`
- NVRAM supports 8-byte atomic writes → Entry size: 2 x 8 bytes
  - Cost benefit of NVRAM: 16B entry on PCM vs. 4KB duplicate data on flash ≈ 1x $ vs. 50x $*

*M. Oros, et al. Analysts weight in on persistent memory. Persistent Memory Summit 2018.



Outline

- Introduction
- Motivation
- Design of Remap-SSD
- Case Studies and Evaluation
- Conclusion

Experimental Setup

- **Experimental platforms**
  - *FEMU SSD* emulator[1]: 32GB capacity
    - filebench, fio, YCSB on RocksDB/MongoDB, db_bench
  - *SSDsim* simulator[2]: 256GB capacity
    - Real-world dedup traces from FIU[3]

Flash page read	50µs / 4KB
Flash page write	500µs / 4KB
Flash block erase	5ms / 1MB
NVRAM read	50ns / 64B
NVRAM write	500ns / 64B
Segment size	1KB by default

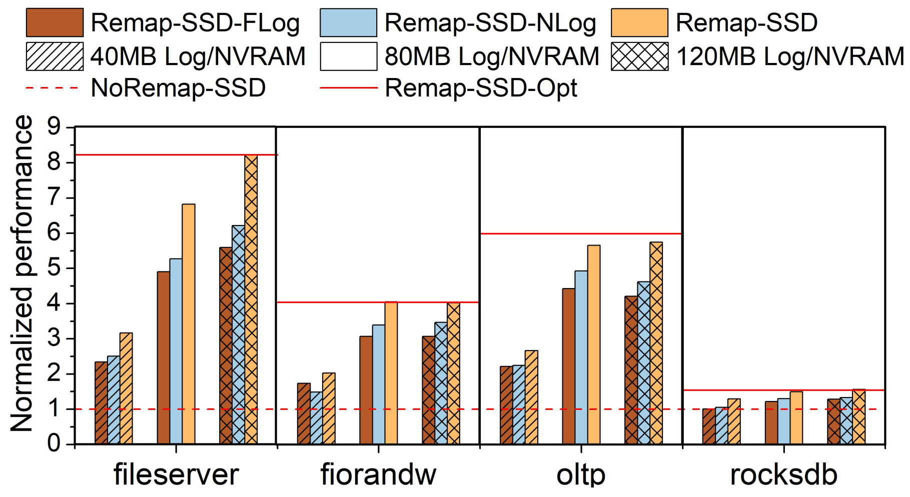
- **Four schemes for comparison**
  - NoRemap-SSD: **baseline** that does NOT exploit the address mapping utility
  - Remap-SSD-FLog: **existing scheme** with a device-wide log on flash memory
  - Remap-SSD-Nlog: an enhancement by storing the device-wide log on NVRAM
  - Remap-SSD-Opt: **optimal case** with no limits on remapping and no P2L lookup overheads
- **Three case studies**
  - Data deduplication: remap for writes of duplicate data
  - Write-ahead logging in SQLite: remap for checkpointing writes
  - Cleaning/GC in log-structured F2FS: remap for data relocations

1. H. Li, et al. The CASE of FEMU: Cheap, Accurate, Scalable and Extensible Flash Emulator. FAST'18.
2. Y. Hu, et al. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. ICS'11.
3. A. Gupta, et al. Leveraging value locality in optimizing NAND flash-based SSDs. FAST'11.

Case Study: Data Deduplication

- **Performance results with different log/NVRAM sizes**
  - Small log/NVRAM would limit the usage of remapping
  - Large log/NVRAM would cause high P2L lookup overheads

- **Remap-SSD: near-optimal performance & scalability**
  - Fast lookups of P2L mappings during GC even with large NVRAM and large-scale remappings

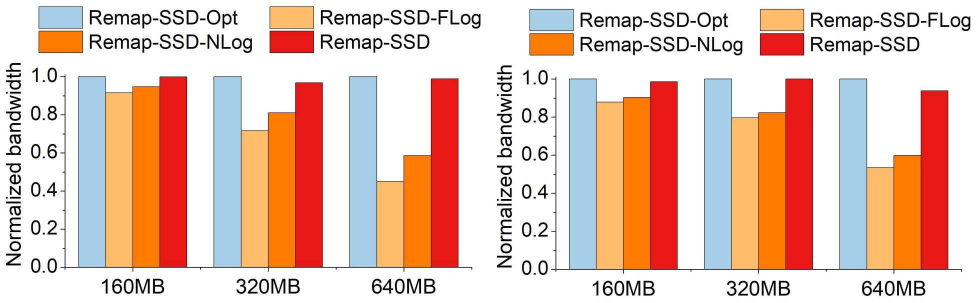


32GB SSD, 40MB-80MB-120MB log/NVRAM
(simulated content locality, 30% duplicate data)

Avg. performance: Remap-SSD vs. others

	40MB	80MB	120MB	160MB	320MB	640MB
FLog	20%	39%	44%	11%	32%	97%
NLog	17%	24%	27%	7%	22%	63%
Opt	-46%	-6%	< -1%	< -1%	< -1%	-4%

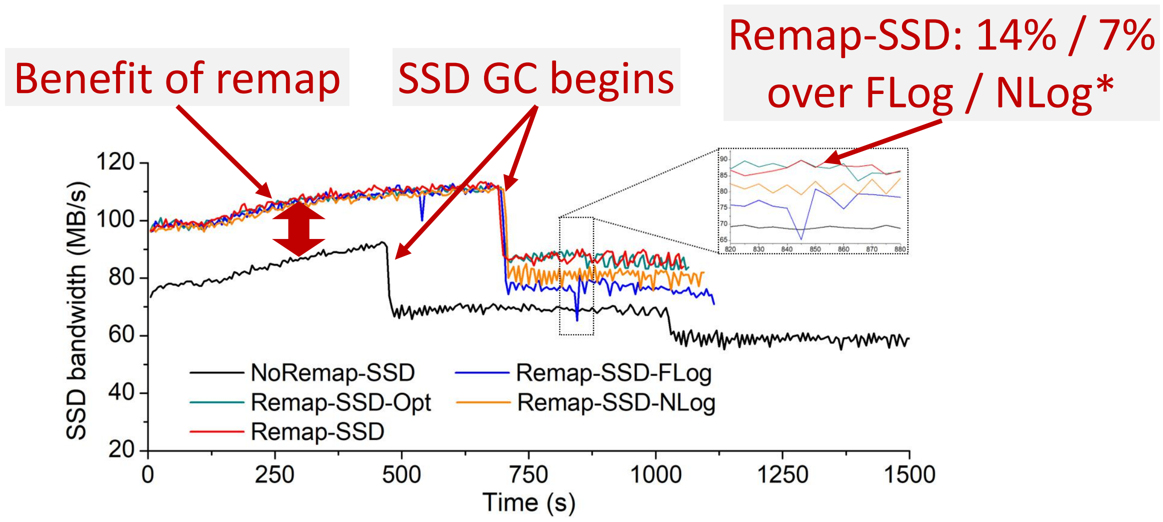
Small NVRAM limits the maximum number of remappings



256GB SSD, 160MB-320MB-640MB log/NVRAM
(real-world dedup traces *homes* and *mail*)

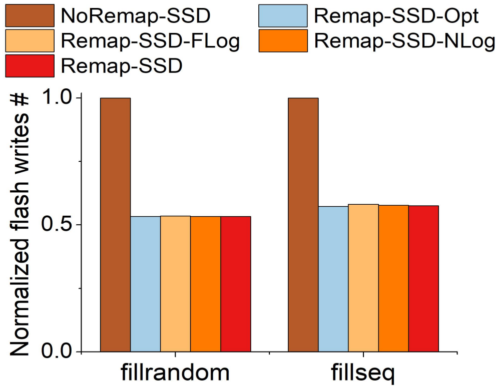
Case Studies: SQLite WAL and F2FS Cleaning

SQLite write-ahead logging



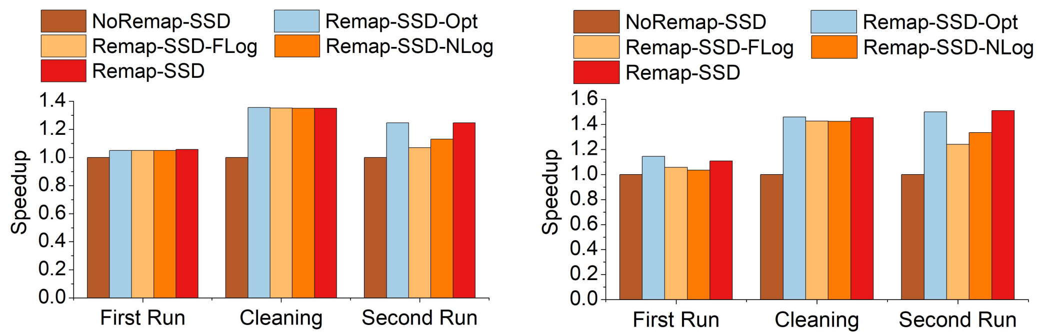
SSD bandwidth in fill-random (db_bench)

Remapping enables
single-write WAL:
45% fewer flash writes



* 32GB SSD, 80MB Log/NVRAM.

Cleaning in log-structured F2FS



Speedups in fileserver (filebench), YCSB on MongoDB

1. **First run of workload to age F2FS**
   - Similar perf.: few clean/remap operations
2. **Cleaning F2FS until all invalid data are reclaimed**
   - Accumulate remapping metadata entries
   - Remapping accelerates cleaning by 28%
3. **Second run of workload to show performance**
   - Remap-SSD improves perf. by 19% over FLog and 12% over NLog*.

Outline

- Introduction
- Motivation
- Design of Remap-SSD
- Case Studies and Evaluation
- Conclusion

Conclusion

- **Duplicate writes are common but harmful to SSD performance and lifetime.**
- **SSD address remapping can eliminate duplicate writes but its usage is severely limited due to the L2P and P2L mapping inconsistency problem.**
  - A device-wide log for P2L mappings: high lookup overheads limit remappings
  - Other solutions: only specific remapping scenarios
- **We propose Remap-SSD to exploit full potentials of SSD address remapping.**
  - Expressive Remap primitive: logical writes of duplicate data
  - Well-designed remapping metadata: mapping consistency, remapping atomicity
  - Novel metadata management: fast lookups and persistence of P2L mappings
    - Maintain small local logs in segmented NVRAM for flash GC units on demand
- **Three case studies show Remap-SSD can achieve near-optimal performance and good scalability in all types of remapping scenarios.**
  - Data deduplication, SQLite write-ahead logging, F2FS cleaning

Thank You!

Q&A

Email: zhouyou2@hust.edu.cn

FAST'21
19th USENIX Conference on
File and Storage Technologies