When F2FS Meets Address Remapping

Yongmyung Lee, Jong-Hyeok Park, Jonggyu Park, Hyunho Gwak,
Dongkun Shin, Young Ik Eom, Sang-Won Lee

Sungkyunkwan University

Hotstorage 2022

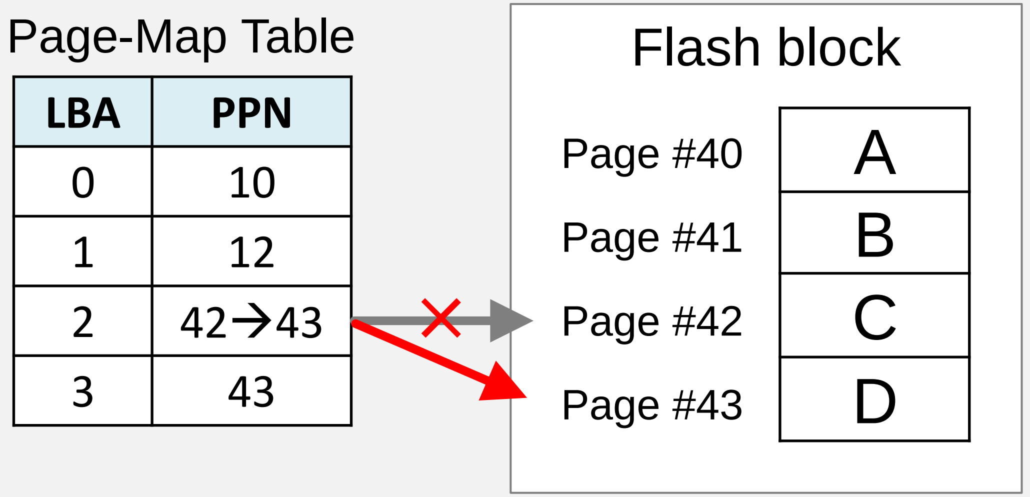
Address-Remapping in Flash Storage

- **Various Address-Remap Command use cases**
  - **Journaling**
    - Remove redundant writes for write-ahead log
  - **Segment Cleaning**
    - Remove the move of valid data in the victim segment

✓ **Remove duplicate writes**

✓ **Flash Lifetime & Performance**

✓ Remap(LBA: 2, LBA: 3)



Page-Map Table

LBA	PPN
0	10
1	12
2	42→43
3	43

Flash block

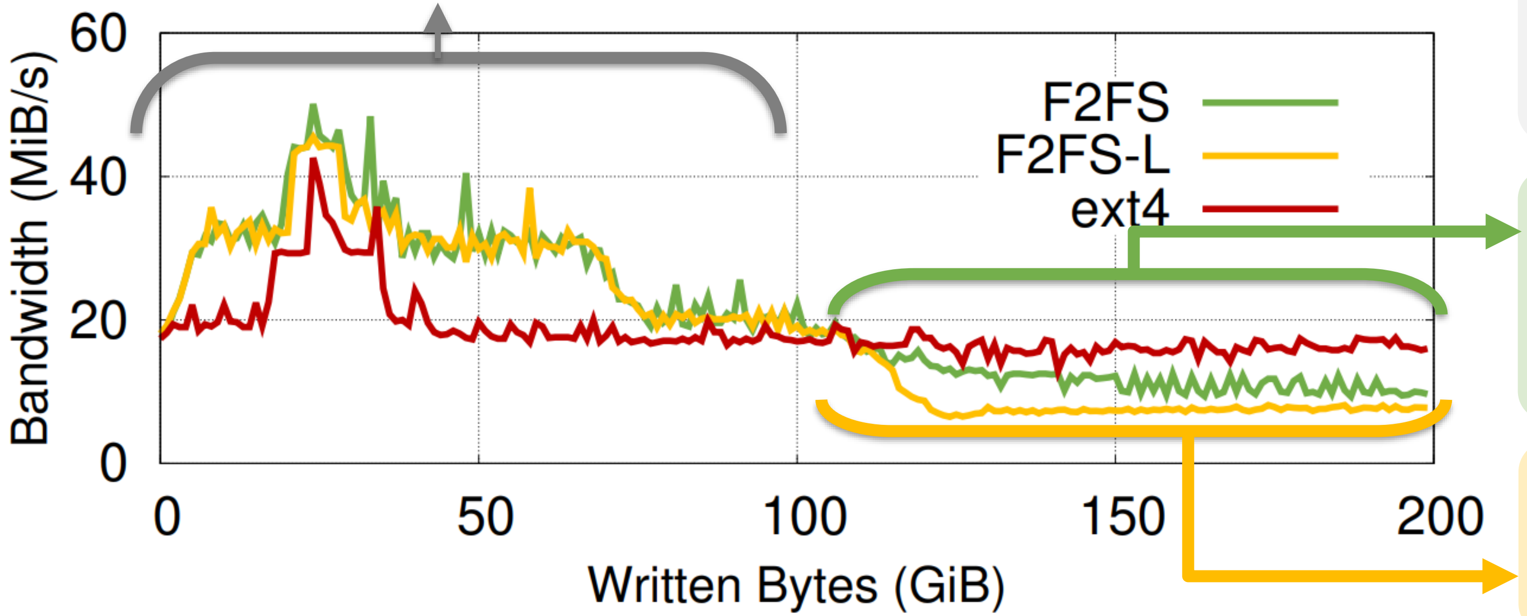
Page #40	A
Page #41	B
Page #42	C
Page #43	D

- ➢ LBA #2 can point to D (in PPN#43) without the actual data writes

F2FS Drawbacks

- **Segment Cleaning Overhead**
  - – F2FS uses out-ot-place policy (OPU) for better random update performance
    - OPU generates invalid blocks. To reclaim valid blocks, F2FS performs segment cleaning

- F2FS performs well
  random writes → sequential ones



- ▪ 250GiB commercial SSD
- ▪ Fill up 50% space with test files
- ➢ 4KiB random writes, 4 threads, fsync-interval=10

- F2FS – Threaded Logging
- ✓ Random writes due to SSR (+ metadata)

- F2FS-L – LFS mode
- ✓ Segment cleaning operations

F2FS Drawbacks

Metadata Update Overheads

- OPU generates additional metadata overhead to keep track of the up-to-date data location

▪ Create a 4MiB or 1GiB file
➢ Perform 4KiB Random Writes with fsync-interval=10

	ext4/4MiB	**F2FS/4MiB**	**ext4/1GiB**	**F2FS/1GiB**
Written Bytes	4.50 MiB	4.68 MiB	1.12 GiB	2.02 GiB
Metadata Ovehead	+0.5 MiB	+0.68 MiB	+0.12 GiB	**+1.02 GiB**

- Random writes in big-size file incur **1.8x** metadata overhead

Fragmentation Overheads

▪ Create 8GiB file and issue 2GiB random writes. (Incur file fragmentation by OPU)
➢ Measure sequential read performance

➢ F2FS shows 43% lower performance than ext4

➢ Average I/O size
→ F2FS – 17KiB / ext4 – 733KiB

- F2FS requires **42.5x** more I/O requests than ext4 by file fragmentation

When F2FS Meets Address Remapping

- F2FS can experience severe performance drop under random-update intensive workloads

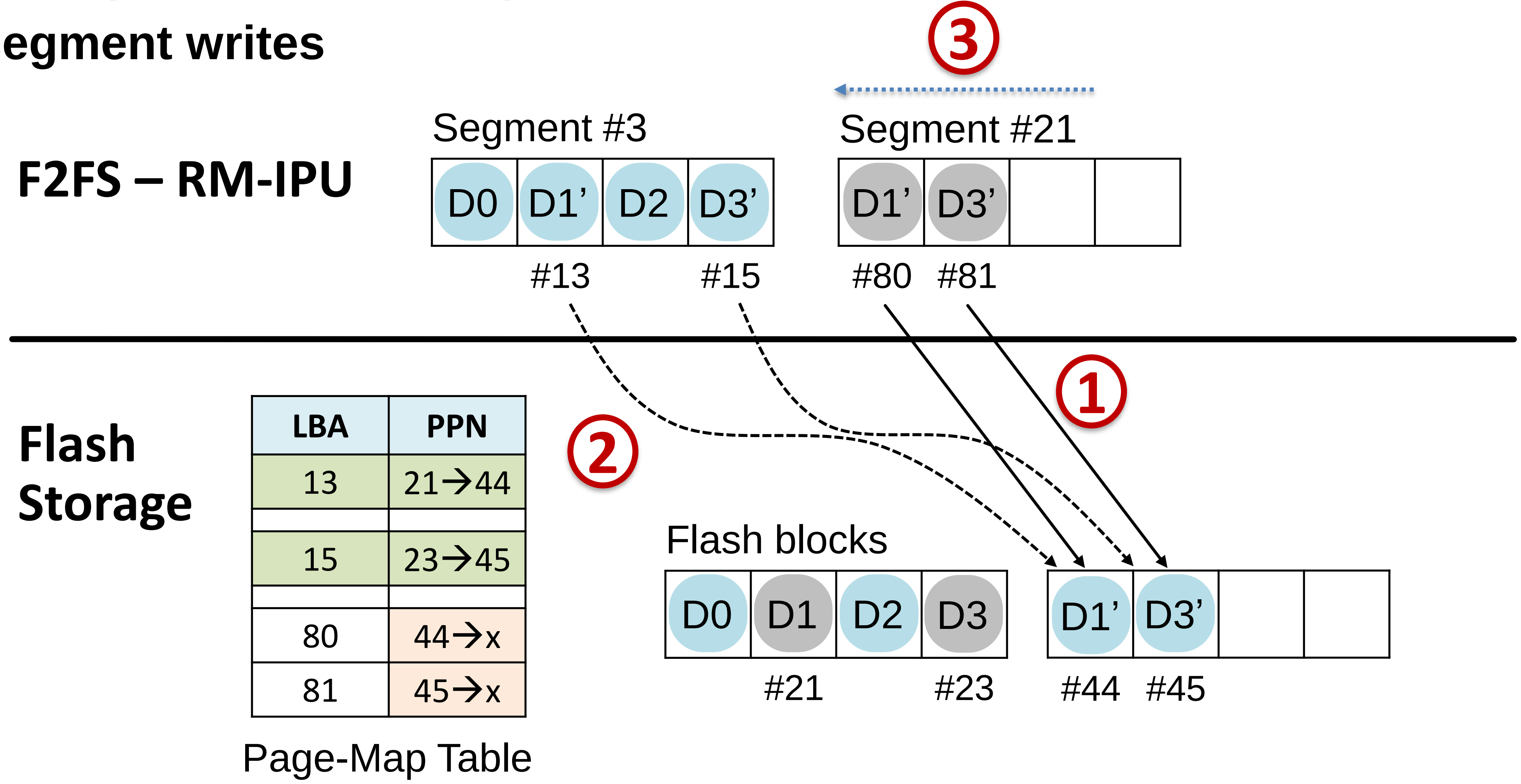
- Suggest **a new Address-remap idea** to mitigate these workloads
  - **How about performing Address-remap immediately after data writes in random-update workloads**
  - Exploit the advantages of both ***OPU (F2FS)*** and ***IPU (ext4)***

REMAP-BASED IN-PLACE-UPDATE

① **Write Data blocks to Current Segments**

② **Issue Remap command for Update-Blocks**

③ **Undo segment writes**



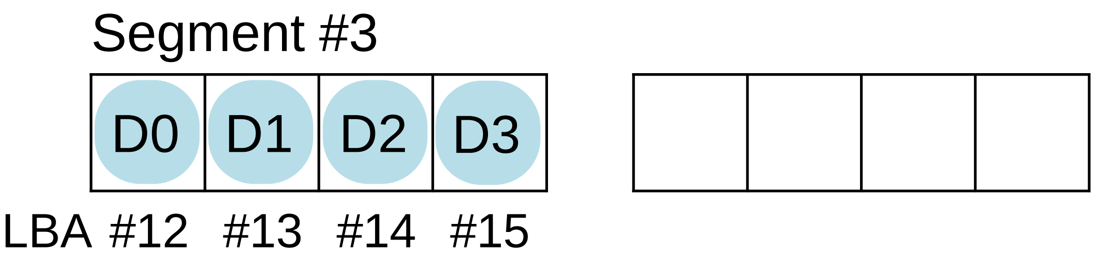
LBA	PPN
13	21→44
15	23→45
80	44→x
81	45→x

Page-Map Table

REMAP-BASED IN-PLACE-UPDATE

- **File has four blocks, D0, D1, D2, D3**

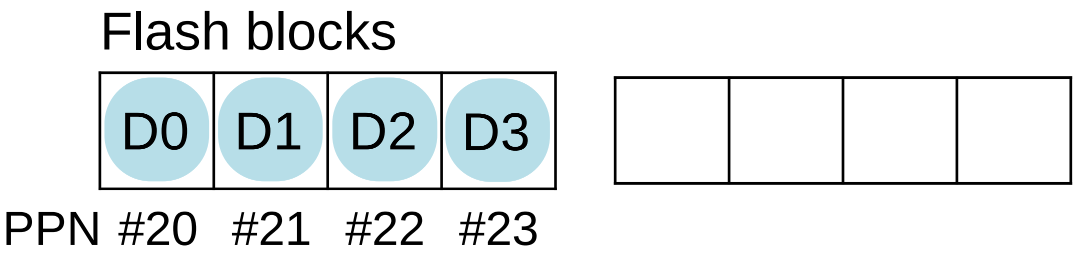
F2FS – RM-IPU



Flash Storage

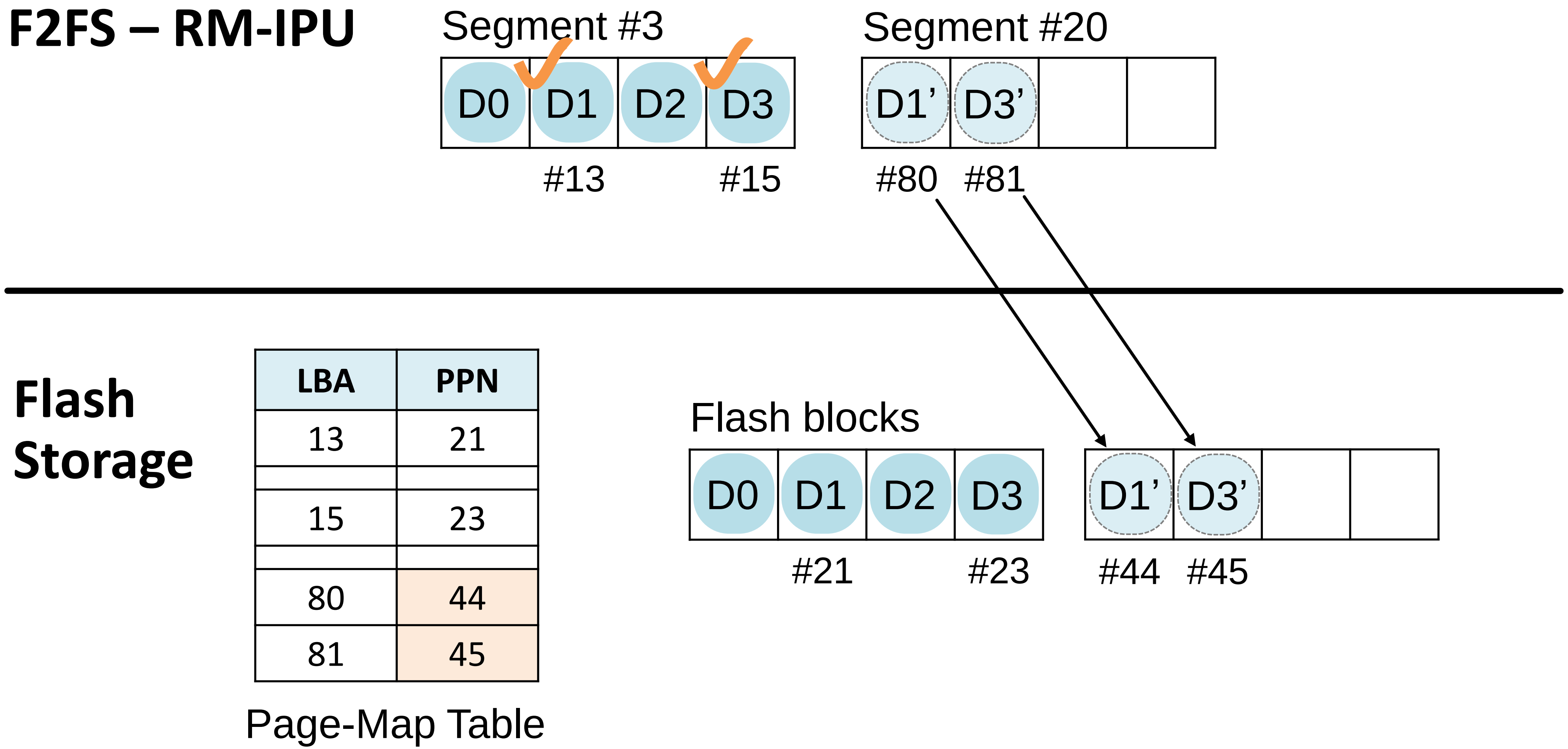
LBA	PPN
12	20
13	21
14	22
15	23

Page-Map Table



REMAP-BASED IN-PLACE-UPDATE

- **1. Write Data blocks to Current Segments**

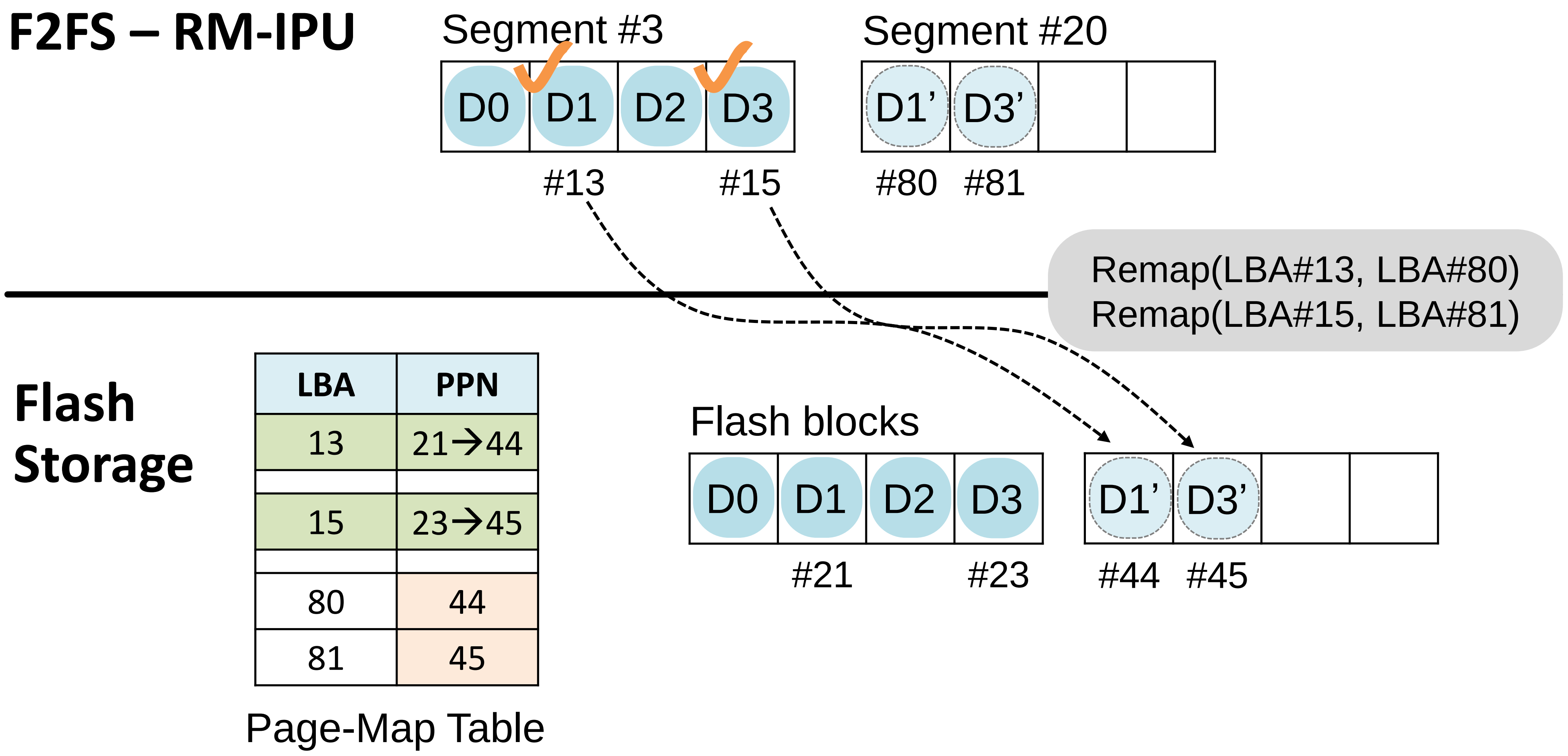


LBA	PPN
13	21
15	23
80	44
81	45

Page-Map Table

REMAP-BASED IN-PLACE-UPDATE

- **2. Issue Remap command for Update-Blocks**



Remap(LBA#13, LBA#80)
Remap(LBA#15, LBA#81)

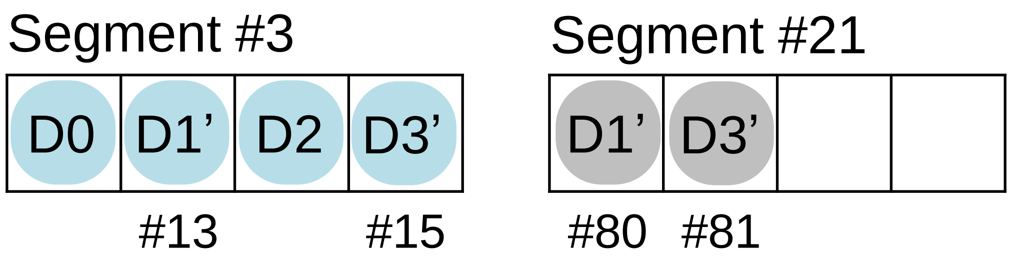
LBA	PPN
13	21→44
15	23→45
80	44
81	45

Page-Map Table

REMAP-BASED IN-PLACE-UPDATE

- **3. Undo segment writes**

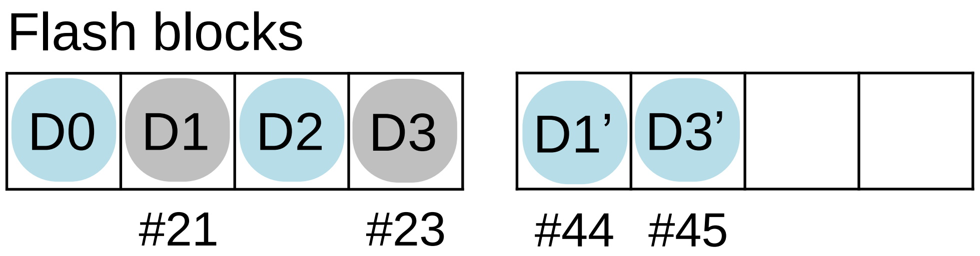
F2FS – RM-IPU



Flash Storage

LBA	PPN
13	21→44
15	23→45
80	44→x
81	45→x

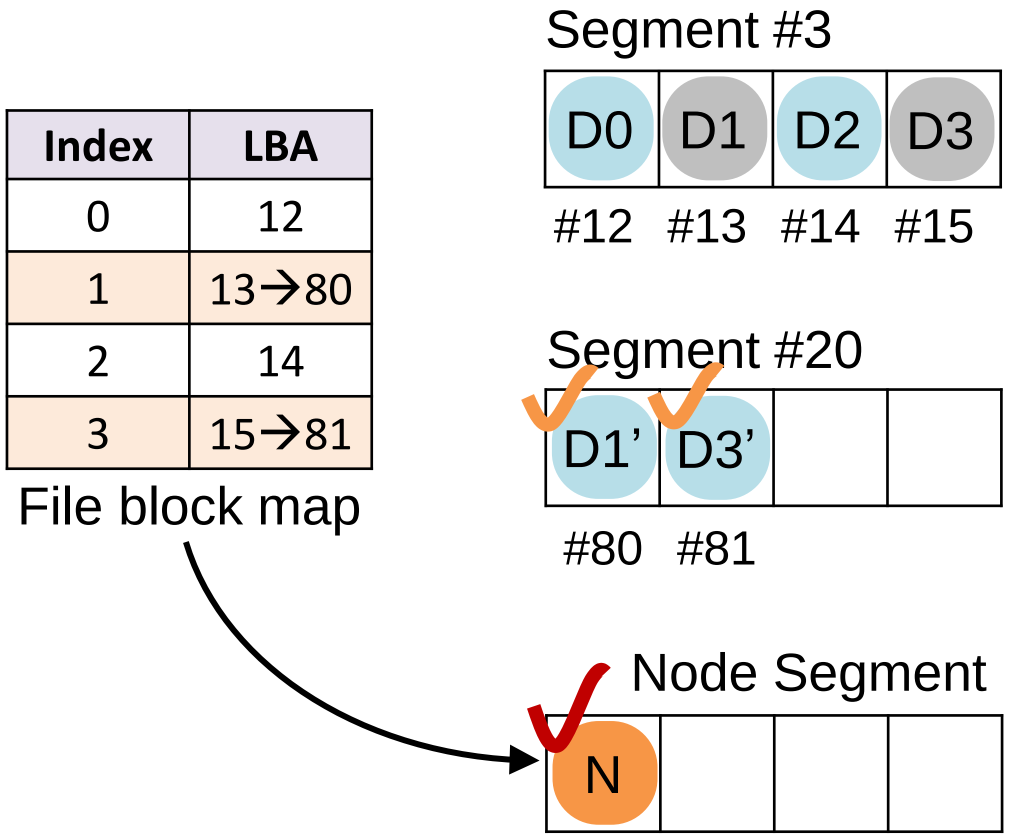
Page-Map Table



REMAP-BASED IN-PLACE-UPDATE

- **Reduce metadata update overheads**
  - No need to update node block

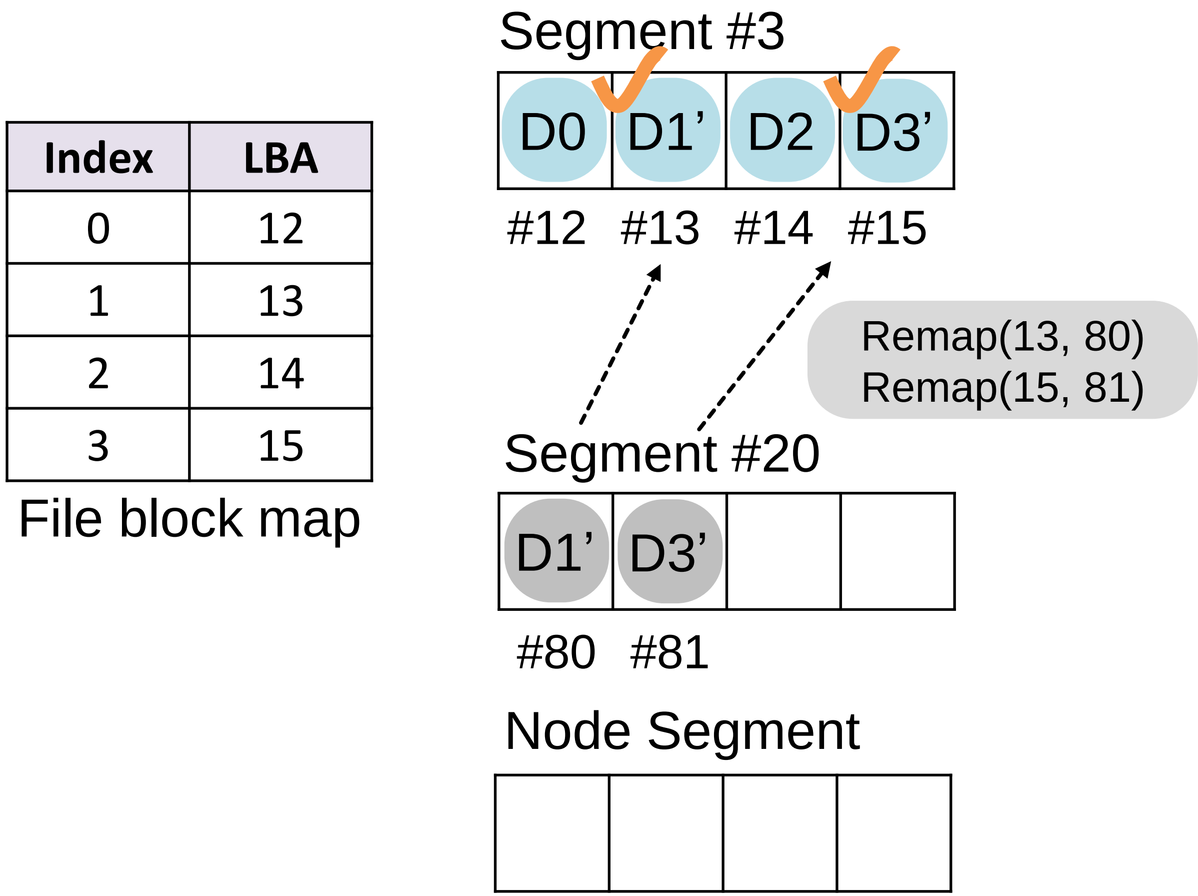
F2FS – OPU



Index	LBA
0	12
1	13→80
2	14
3	15→81

File block map

F2FS – RM-IPU



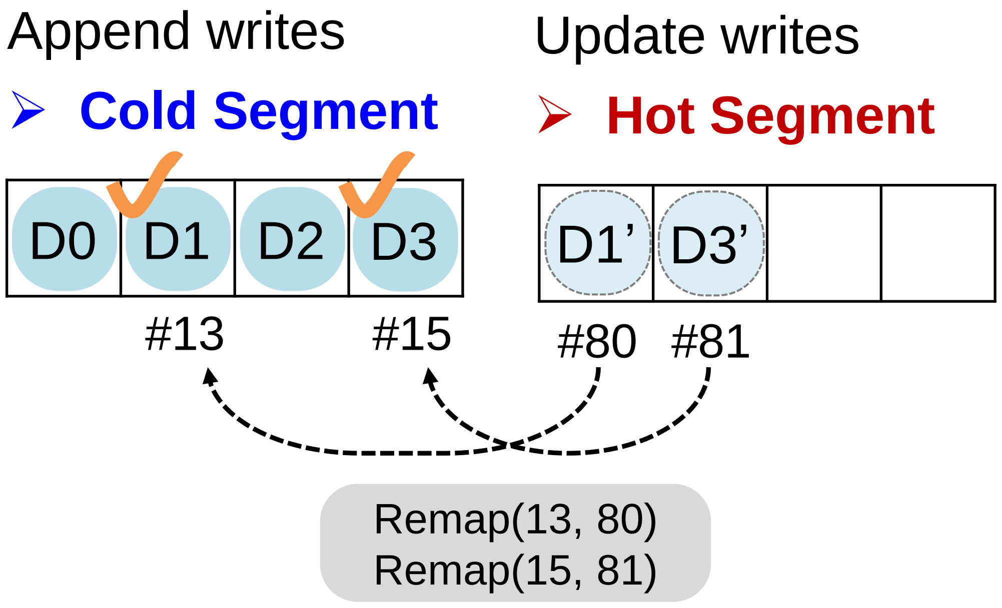
Index	LBA
0	12
1	13
2	14
3	15

File block map

Remap(13, 80)
Remap(15, 81)

REMAP-BASED IN-PLACE-UPDATE

- **Minimize segment cleaning overheads**
  - RM-IPU uses the multi-head logging strategy
  - Update blocks are stored in the hot segment.
    After address-remap,
    it will be invalidated immediately
    and the segment will be a free segment



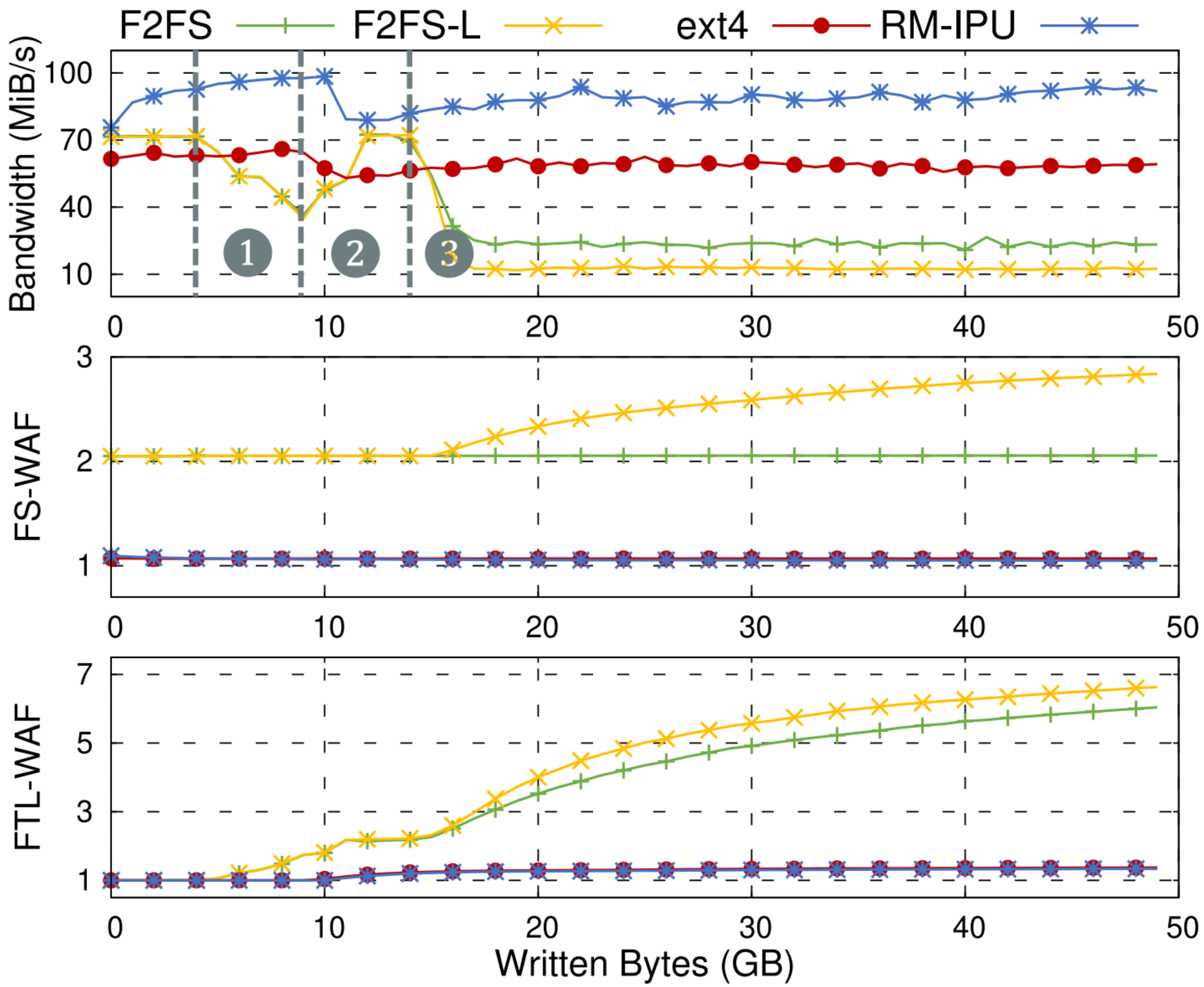
- **Eliminate fragmentation overheads**
  - Update data blocks are relocated original location by Address-remap

Performance Evaluation

- **Evaluation Setup**
  - FEMU (QEMU based NVMe SSD Simulator)
    - Capacity – 32GiB (+4GiB Overprovisioning space)
    - Page size – 4KiB
    - Flash Page Read/Write, Block Erase latency – 50us, 500us, 5ms
    - One map entry (in Remap Command) access latency – 0.1 us (NVRAM access latency)
- **Workloads**
  - 1) Fio Benchmark
    - Random write 4KiB, 4 Threads - 4GB Files, fsync-interval 10
  - 2) TPC-C benchmark (MySQL)
    - Update-intensive OLTP Workload
    - 16 clients, 200 Warehouses (Disk Usage 60~70%)

Performance Evaluation

- **Fio Benchmark**



1
- **F2FS/F2FS-L**
  Free flash blocks run out → Performance drop
  (Due to metadata overhead)

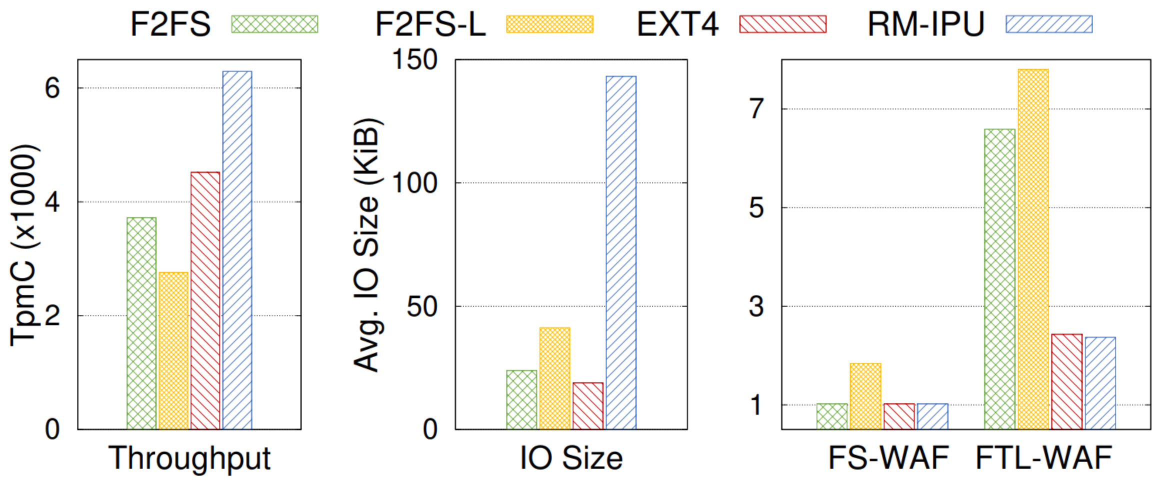
2
- Securing the free flash blocks,
  F2FS recovers its performance

3
- Show throughput drop again,
  due to consuming all valid segments,
  - F2FS → random writes (SSR)
  - F2FS-L → segment cleaning operation

- **RM-IPU** can sustain better performance than F2FS and ext4 due to sequential writes and small metadata overhead

Performance Evaluation

- **TPC-C – MySQL**



➢ RM-IPU shows better performance due to exploiting sequential writes manners (→ better IO size) and eliminating segment cleaning overhead (→ good for FS-WAF & FTL & WAF)

- **Sequential Continuity**

	F2FS	ext4	RM-IPU
Mean request size	17 KiB	740 KiB	737 KiB
Request count	498K	11K	11K

- Create 8GiB file and issue 2GiB random writes, measure sequential read performance

➢ RM-IPU retains a similar size and number of request with ext4's

Conclusion & Future Directions

- **RM-IPU takes advantages of Log-structured manner and Address-remap**
  - RM-IPU reduces the segment cleaning overhead and prevents the file fragmentation problem under the random write-intensive workloads, thus it resolves the drawbacks of F2FS
- **Future Directions**
  - What about other workloads? – Sequential & Large block size workloads
  - F2FS Atomic-write + RM-IPU
  - Adaptive write mode selection in F2FS – LFS mode / Adaptive mode / RM-IPU mode
- **Question and Answer**