PERSISTENCE: UNWRITTEN CONTRACT OF SSDS

Andrea Arpaci-Dusseau

CS 537, Fall 2019

ADMINISTRIVIA

Project 7: xv6 File systems: Improvements + Checker

- Specification Quiz – Worth Project Points
  - 7a due tomorrow, 7b due Thursday
- Topic of tomorrow's Discussion Sections
- Can still request project partner if needed

Final Exam

- Friday, December 13th 7:25-9:25 pm
- Two Rooms: Last name A-H in SOC SCI 5206, I-Z in Humanities 3650
- Slightly cumulative (some T/F from Virtualization and Concurrency – 25%)

Exam Review

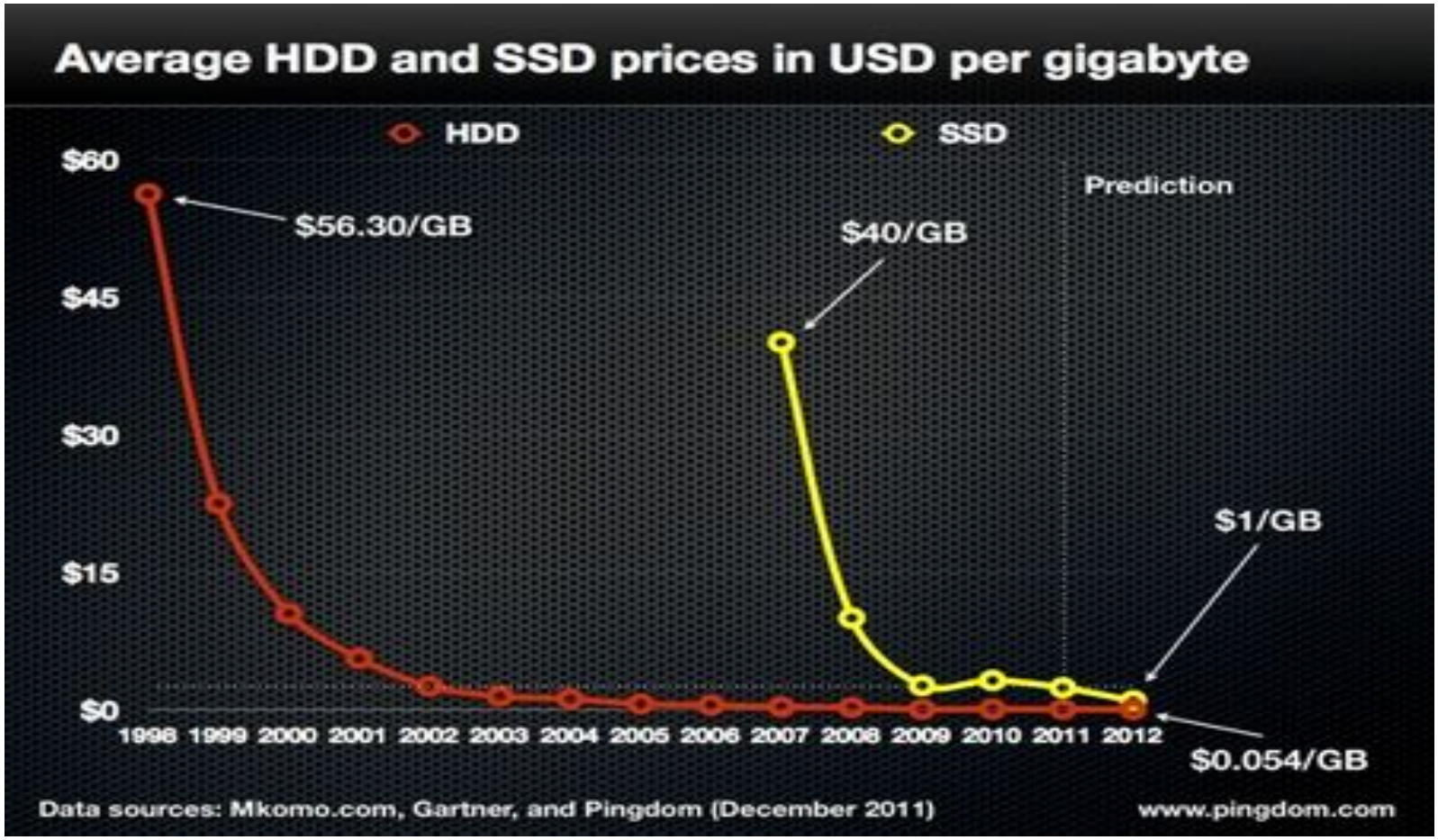
- Next Tuesday: You ask questions to cover by Monday at 5:00pm
- Next Wednesday discussions

AGENDA / LEARNING OUTCOMES

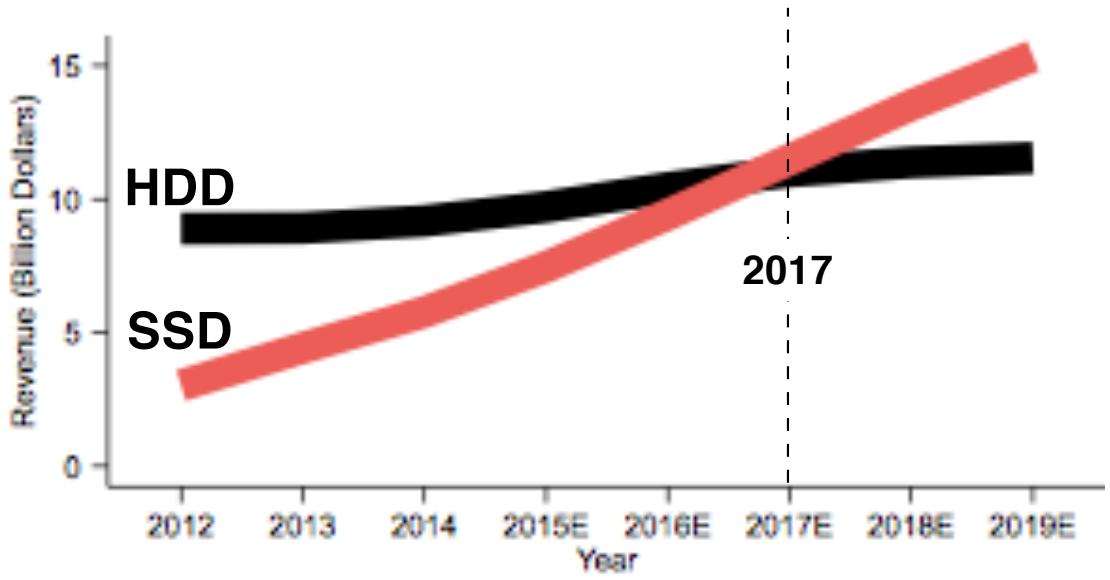
What is the unwritten contract of SSDs?

- How are SSDs structured?
- How should they be used to obtain best performance?

COST: HDD VS. SSD

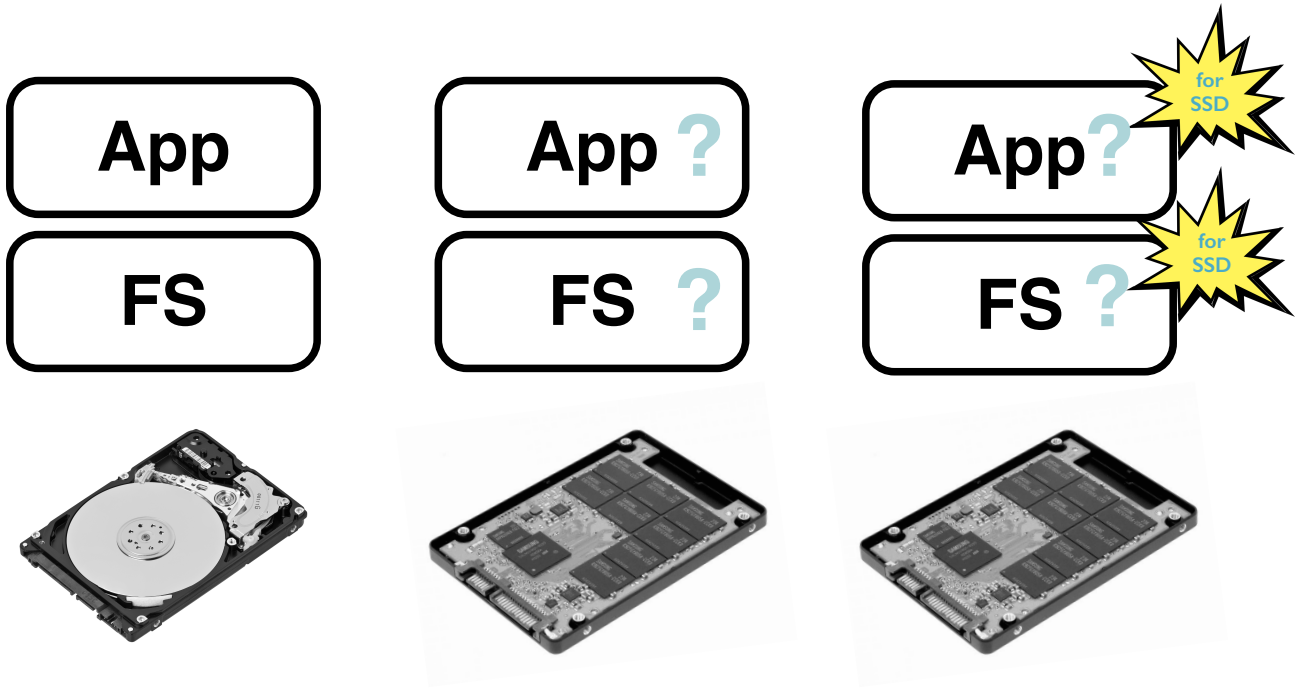


Enterprise SSD revenue is expected to exceed enterprise HDD in 2017



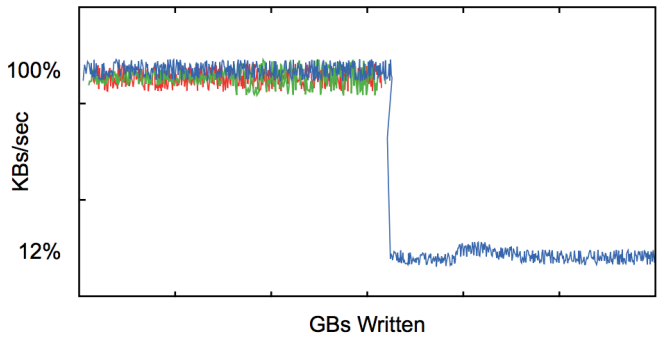
Source: Gartner, Stifel Estimates
https://www.theregister.co.uk/2016/01/07/gartner_enterprise_ssd_hdd_revenue_crossover_in_2017/

PROS AND CONS OF BLOCK-BASED DEVICES

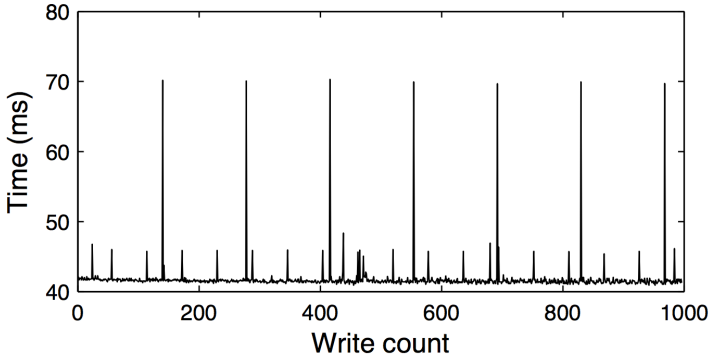


THE CONSEQUENCES OF MISUSING SSDS

Performance degradation



Performance fluctuation



Early end of device life

http://crestingwave.com/sites/default/files/collateral/velobit_whitepaper_ssdperformancetips.pdf

S. Boboila and P. Desnoyers. Write Endurance in Flash Drives: Measurements and Analysis. In Proceedings of the 8th USENIX Symposium on File and Storage Technologies (FAST

RIGHT WAY USE SSDS?

Block Device Interface: *read(), write()*

HDD Unwritten Contract

- **Sequential accesses are best**
- **Nearby accesses are more efficient than farther ones**

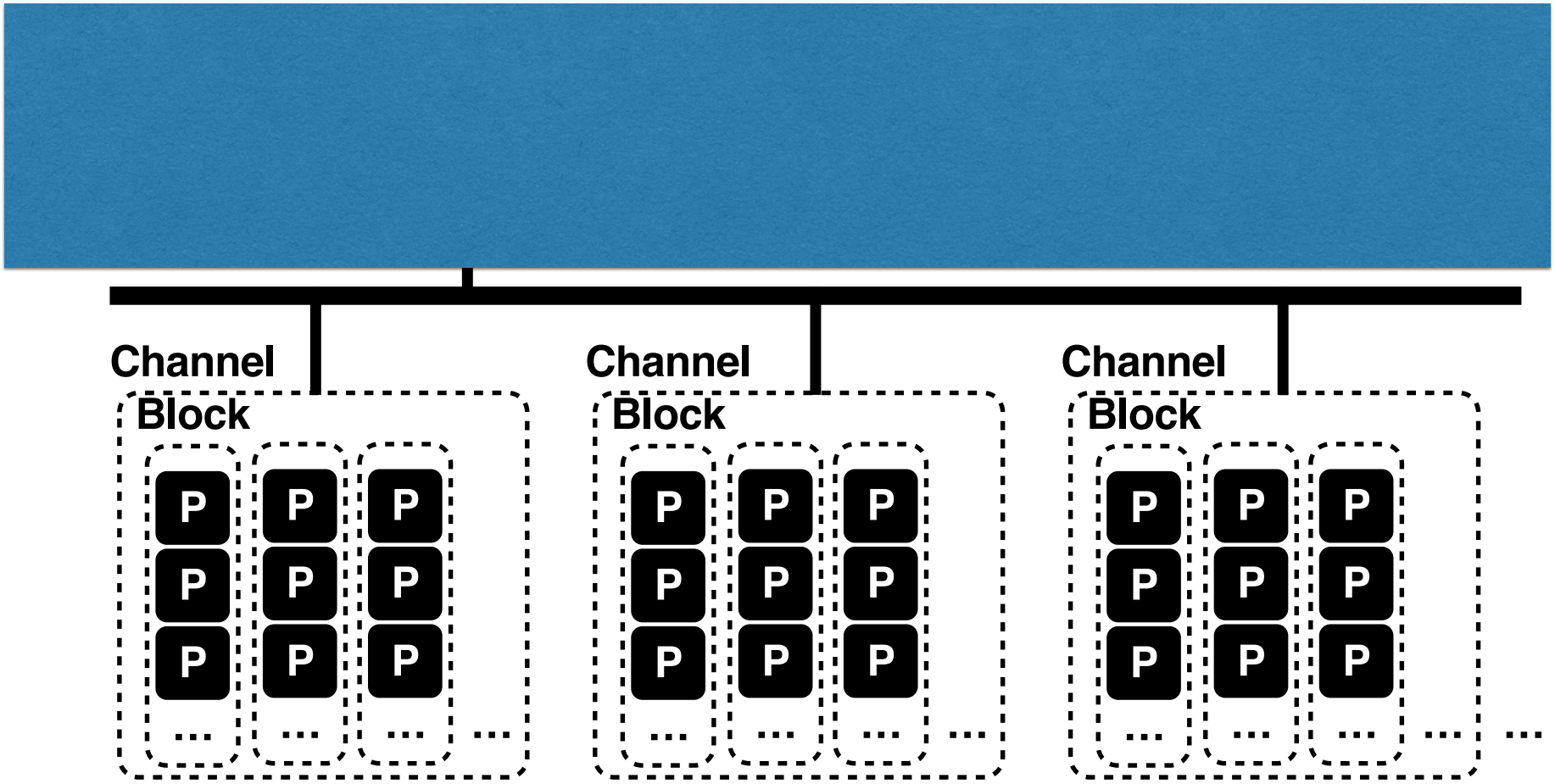
SSD Unwritten Contract

SSD UNWRITTEN CONTRACT

1. **Request Scale**
2. **Locality**
3. **Aligned Sequentiality**
4. **Grouping by Death Time**
5. **Uniform Data Lifetime**

BACKGROUND

SSD Background



FLASH

Hold charge in cells. No moving parts!

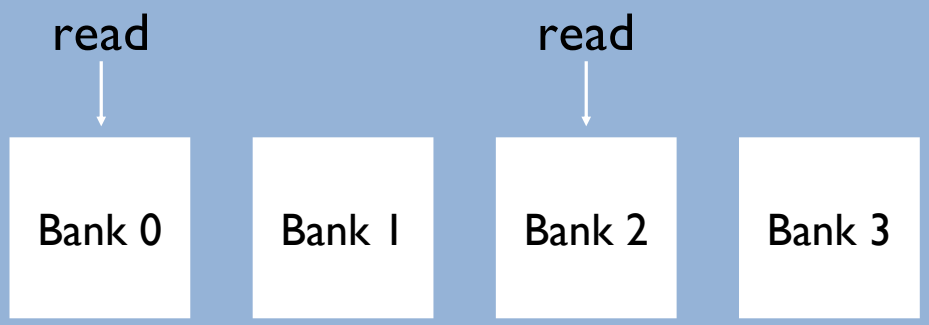
Inherently parallel

No seeks!

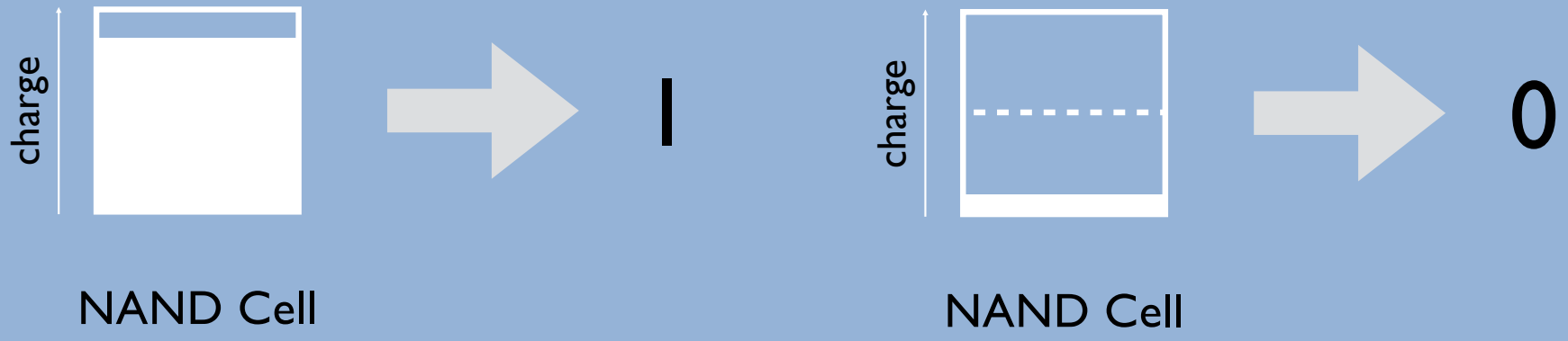
PARALLELISM

Flash devices are divided into banks (aka, planes or channels)

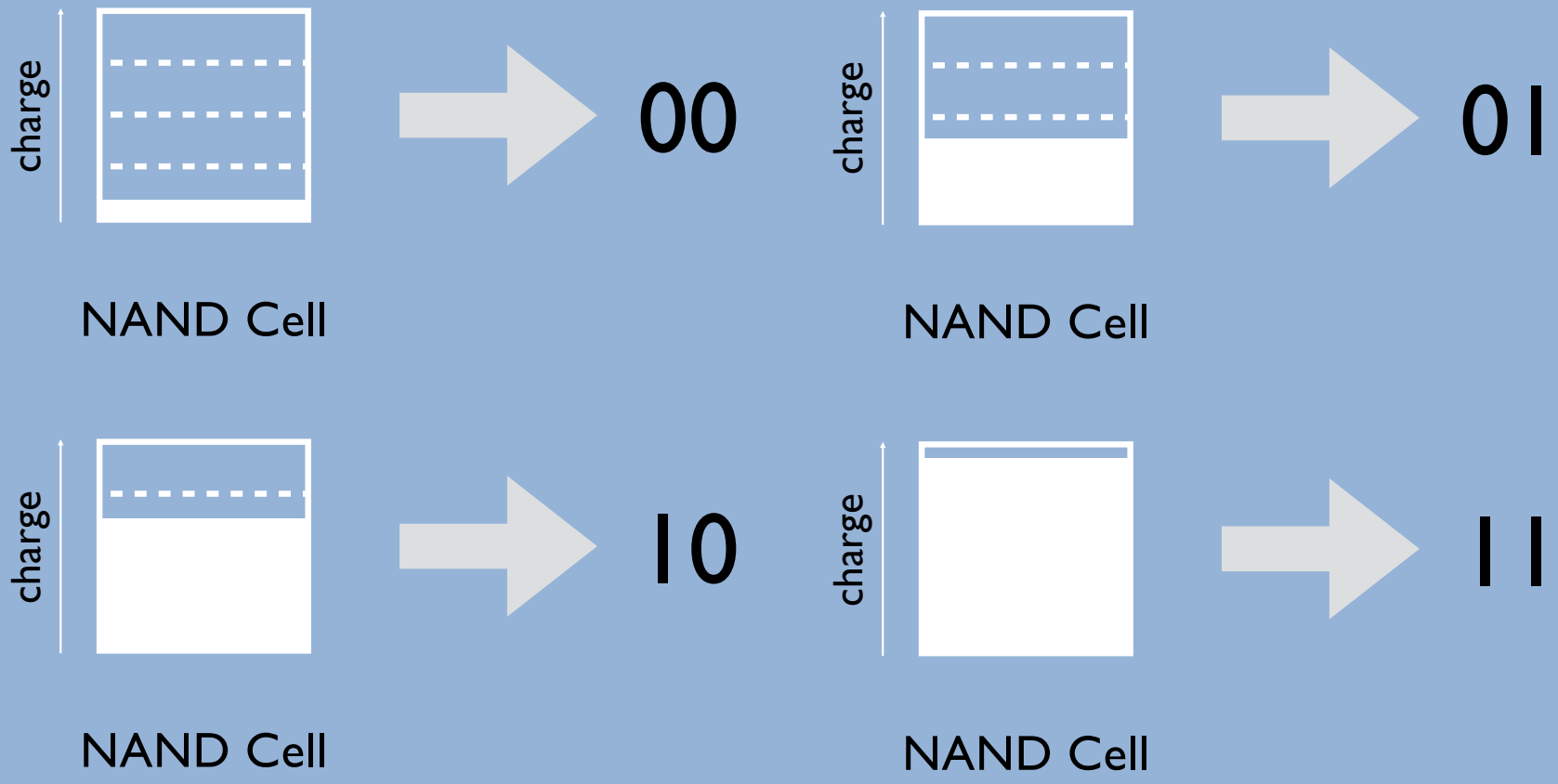
Banks can be accessed in parallel



SLC: SINGLE-LEVEL CELL



MLC: MULTI-LEVEL CELL



SINGLE- VS. MULTI- LEVEL CELL

For the same total capacity, which is the most expensive?

Which is the most robust (fewest errors)?

WEAROUT

Problem: flash cells wear out after being overwritten too many times.

SLC: ~100K times
MLC: ~10K times

Usage strategy: Wear leveling
- Prevents some cells from wearing out while others still fresh
- Need to look at the number of times each has been written and move old data

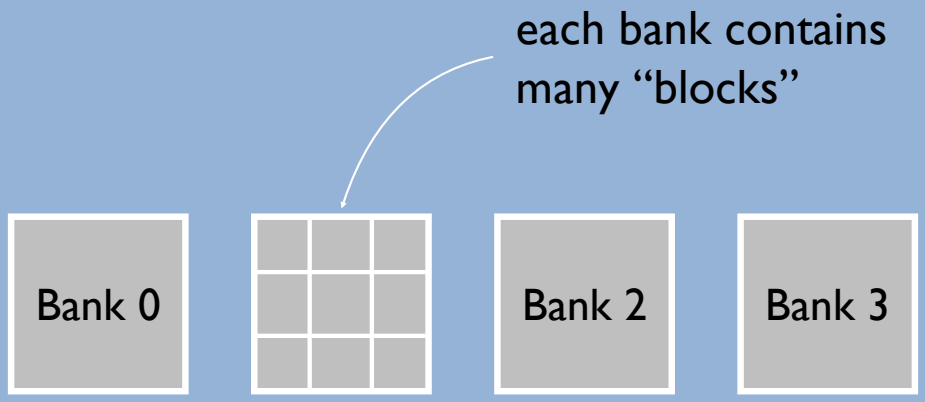
FLASH WRITES

Writing 0's:

- fast, fine-grained [pages]
- called "program"

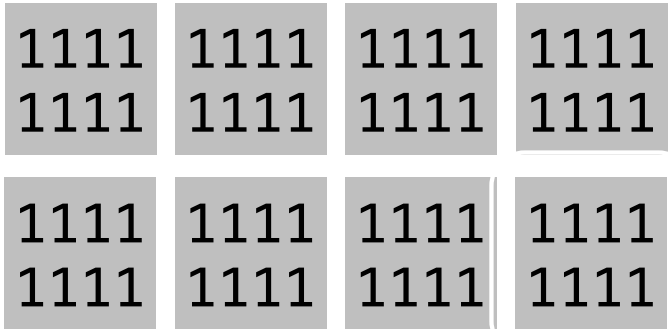
Writing 1's:

- slow, course-grained [blocks]
- called "erase"


each bank contains
many “blocks”
Bank 0
Bank 2
Bank 3

BLOCK

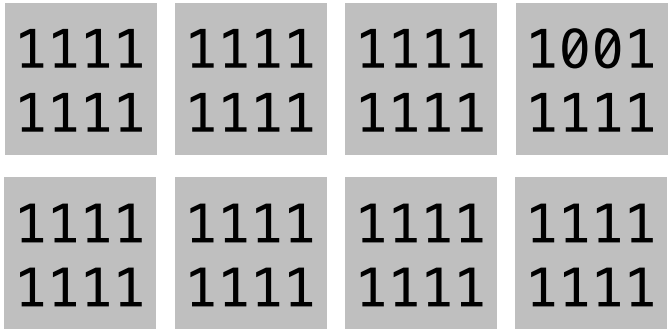
each block contains
many “pages”



one page

BLOCK

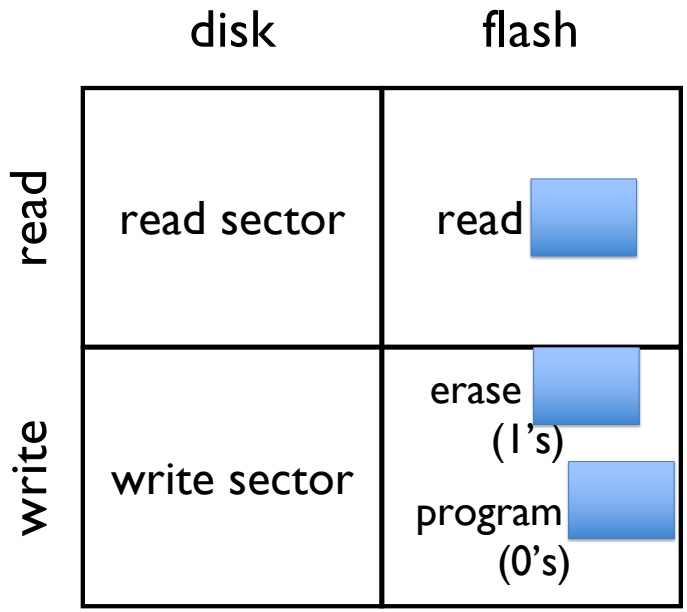
program



BLOCK


erase
1111 1111 1111 1111
1111 1111 1111 1111
1111 1111 1111 1111
1111 1111 1111 1111

API COMPARISON: BLOCKS OR PAGES?



FLASH HIERARCHY

Pick **Plane, Block**, or **Page**:

Accessed in parallel 

64 to 256 pages 

Unit of read and program 

2 to 8 KB 

Unit of erase 

1024 to 4096 blocks 

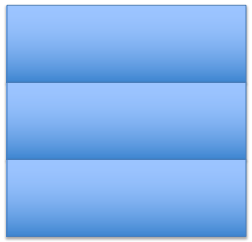
HARD DISK VS. FLASH PERFORMANCE

Throughput:

- disk: ~130 MB/s (sequential only)
- flash: ~200 MB/s

Latency:

- disk: ~10 ms (one op)
- flash



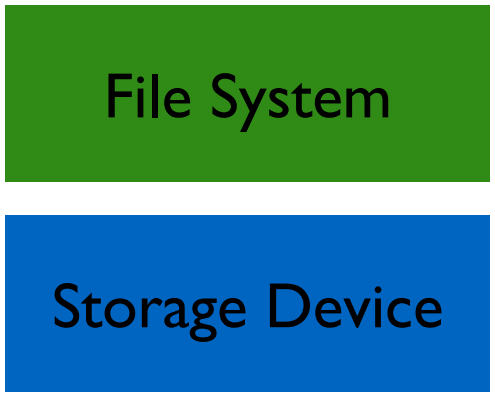
10-50 us

200-500 us

2 ms

IMPLICATIONS:
FILE SYSTEM AND FLASH INTERACTIONS

TRADITIONAL FILE SYSTEMS



Traditional API:
- read sector
- write sector

Not same as flash

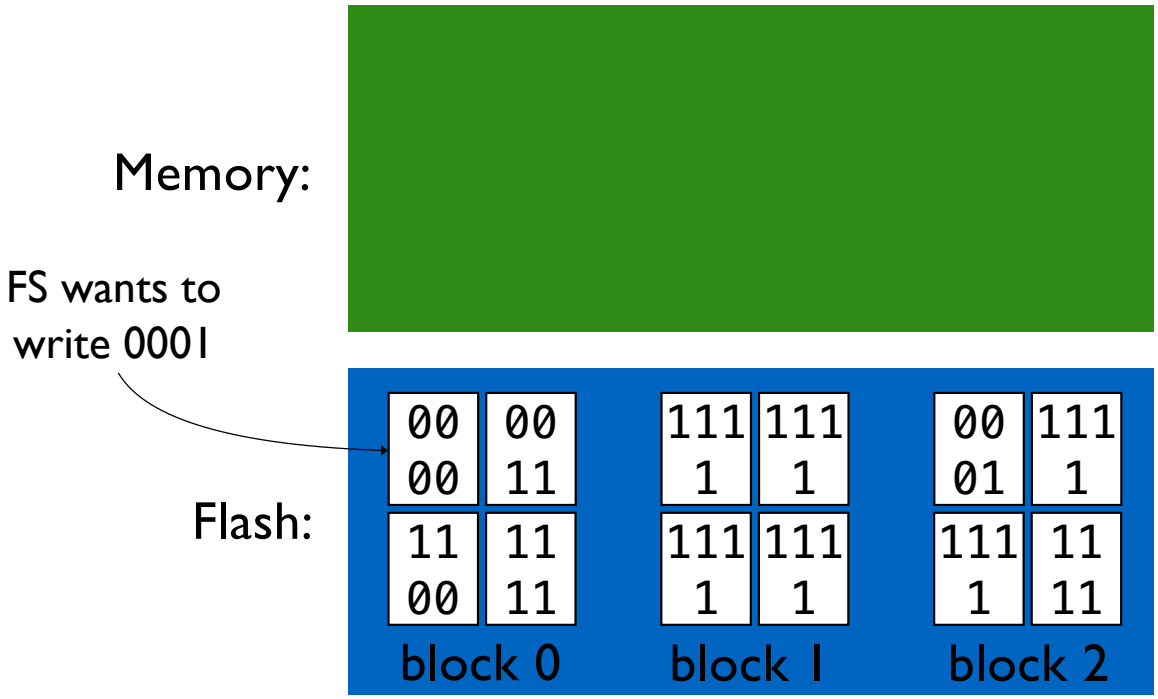
FILE SYSTEM OPTIONS

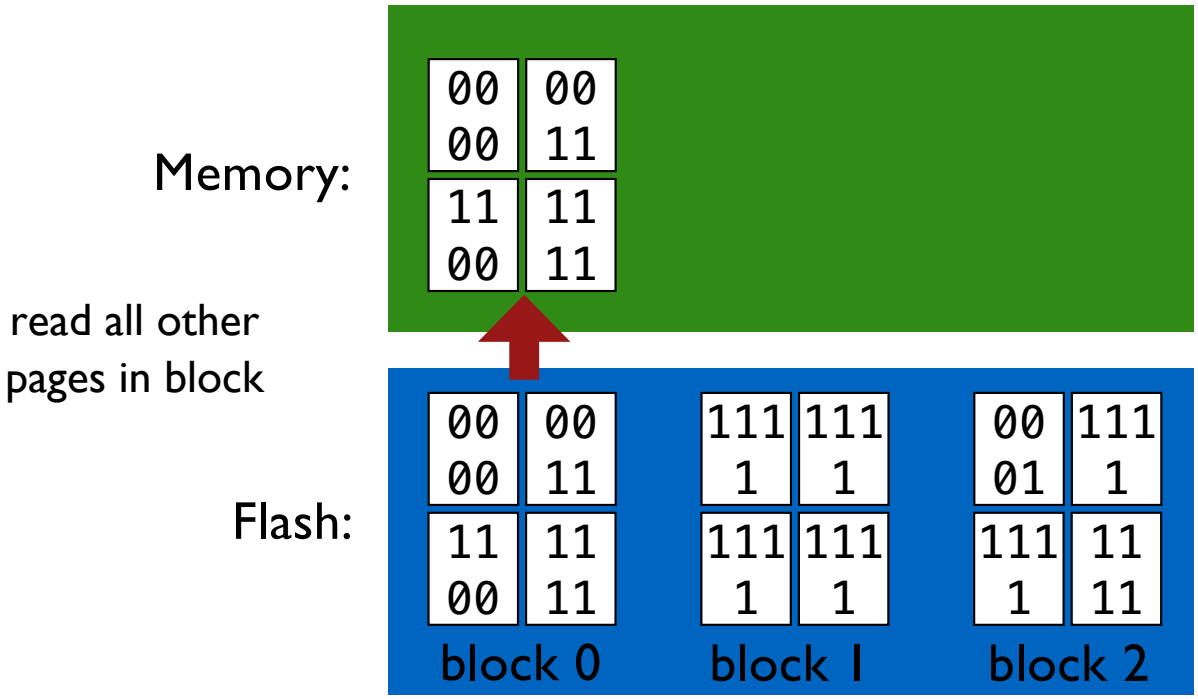
1. Build/use new file systems for flash (using program/erase)
 - JFFS/JFFS2, YAFFS
 - lot of work!

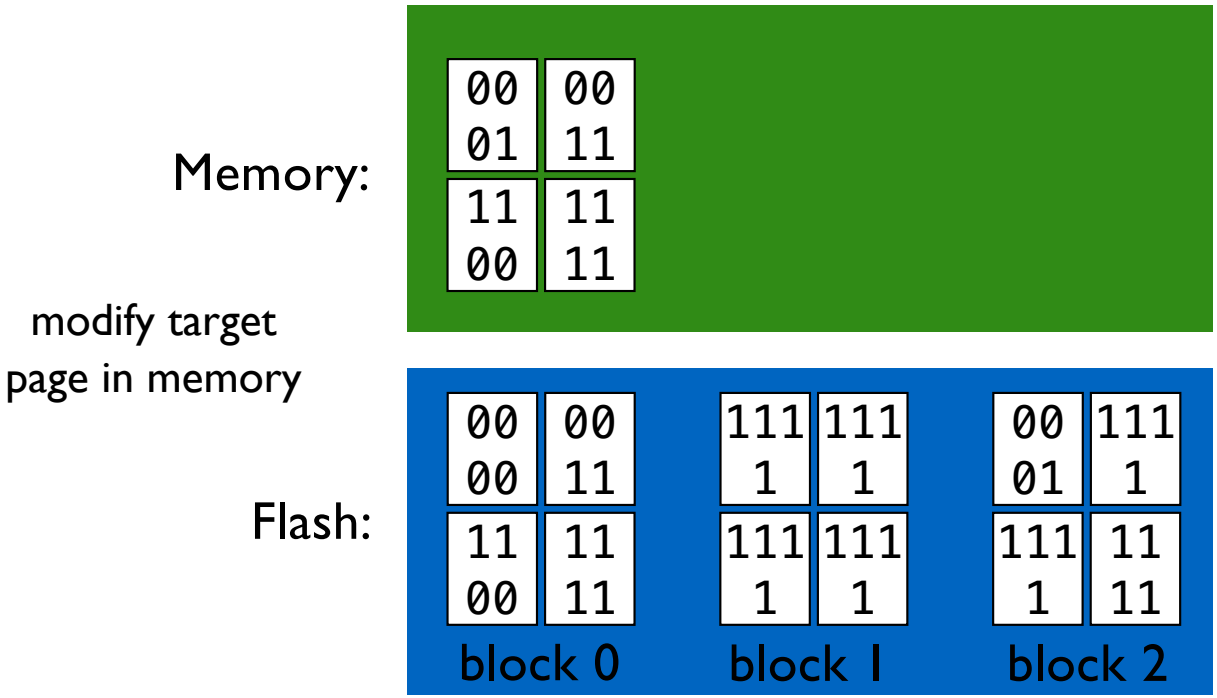
2. Build traditional API over flash API
 - use FFS, LFS, whatever we want

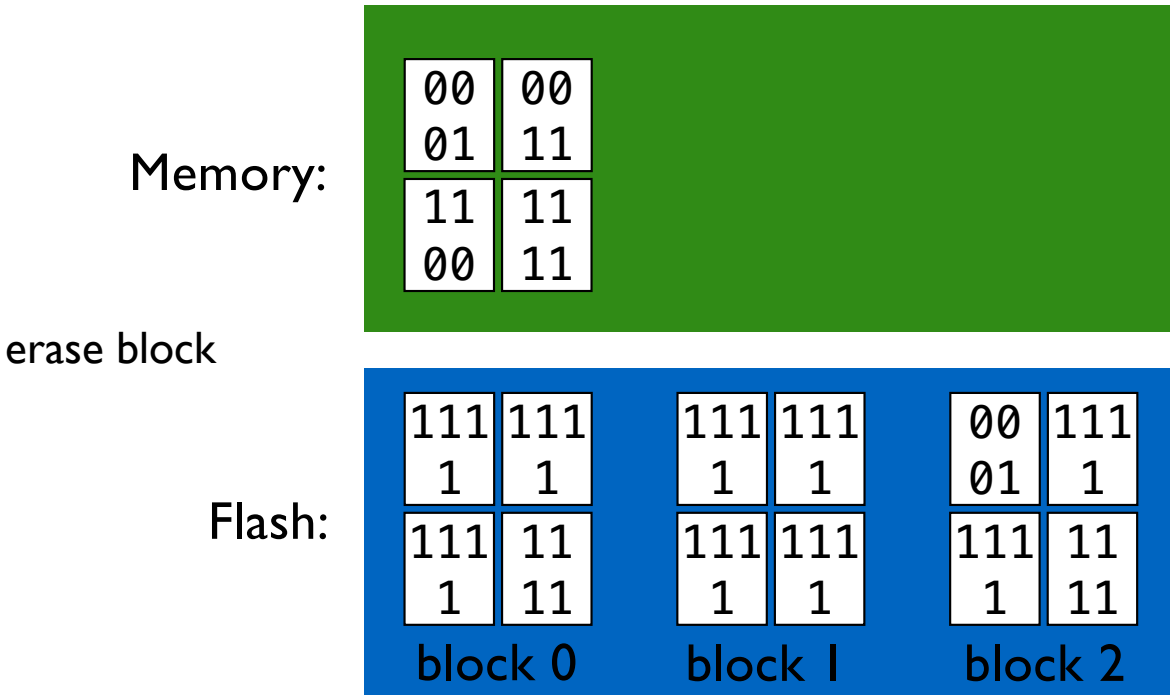
TRADITIONAL API ON FLASH

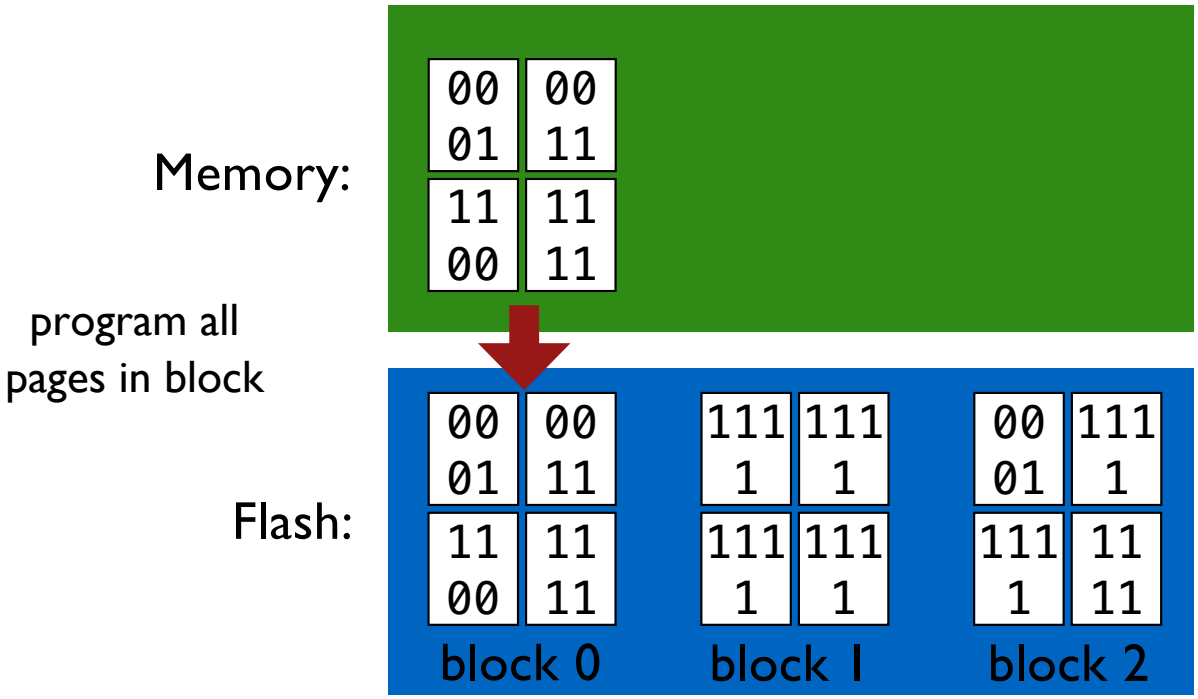
```
read(addr):
    return flash_read(addr)

write(addr, data):
    block_copy = flash_read(block of addr)
    modify block_copy with data
    flash_erase(block of addr)
    flash_program(block of addr, block_copy)
```


Memory:
FS wants to write 0001
Flash:
00 00
00 11
11 00
11 11
111 1
111 1
111 1
111 1
00 01
111 1
111 1
11 11
block 0
block 1
block 2


Memory:
00 00
00 11
11 11
00 11
read all other
pages in block
Flash:
00 00
00 11
11 11
00 11
111 111
1 1
111 111
1 1
00 111
01 1
111 11
1 11
block 0
block 1
block 2


Memory:
00 00
01 11
11 11
00 11
modify target
page in memory
Flash:
00 00
00 11
11 11
00 11
111 111
1 1
111 111
1 1
00 111
01 1
111 11
1 11
block 0
block 1
block 2


Memory:
00 01
00 11
11 00
11 11
erase block
Flash:
1111 1
111 1
1111 1
111 1
00 01
111 1
1111 1
11 11
1111 1
111 1
1111 1
11 11
block 0
block 1
block 2


Memory:
00 00
01 11
11 11
00 11
program all
pages in block
Flash:
00 00
01 11
11 11
00 11
111 111
1 1
111 111
1 1
00 111
01 1
111 11
1 11
block 0
block 1
block 2

WRITE AMPLIFICATION

One write is amplified to many additional writes

Writing one 2KB page may cause:
- `read`, `erase`, and `program` of 256KB block.

Random writes are extremely expensive!

(Sequential could modify all pages in one block at one time…)

Implication for File System:
Would FFS or LFS be better on flash?

COW FS OVER FLASH

Copy-On-Write FS may prevent some expensive random writes

What about wear leveling? LFS won't do this

What if we want to use some other FS?

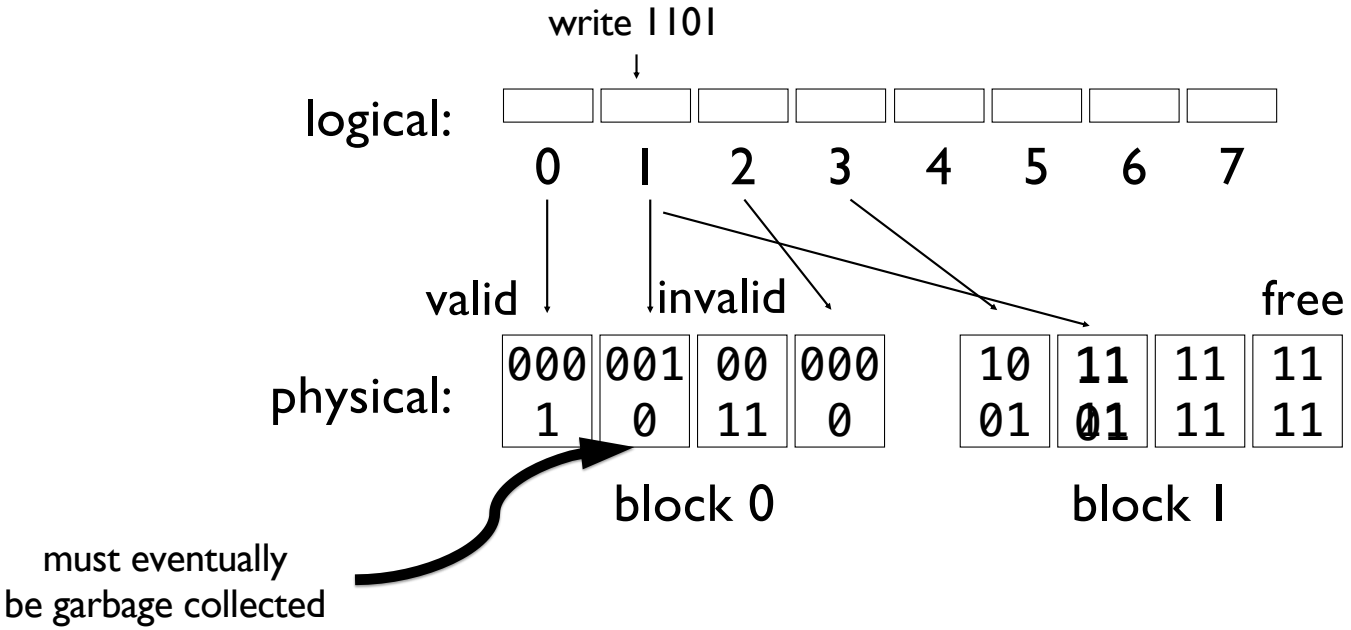
BETTER SOLUTION

Add copy-on-write layer between FS and flash
    Avoids RMW (read-modify-write) cycle

FTL: Flash Translation Layer
    Translate logical device addrs to physical addrs
    Some similarities to RAID translation…

Should FTL use math or data structure?

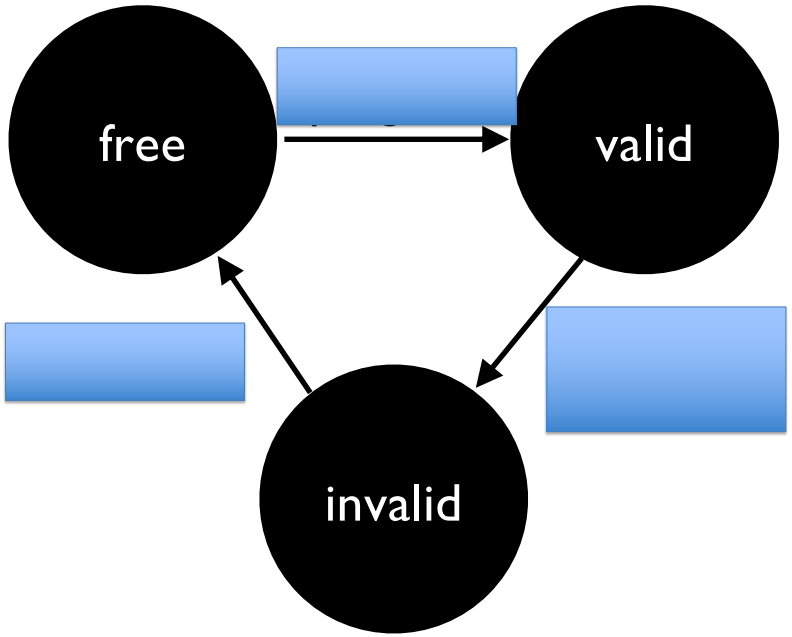
FLASH TRANSLATION LAYER



Physical pages can be in three states: valid, invalid, free

STATES OF PHYSICAL PAGES

Label transitions:
relocate
erase
program



HOW BIG ARE MAPPING TABLES?

Assume 200 GB device, 2 KB pages, 4-byte entries in FTL for each page

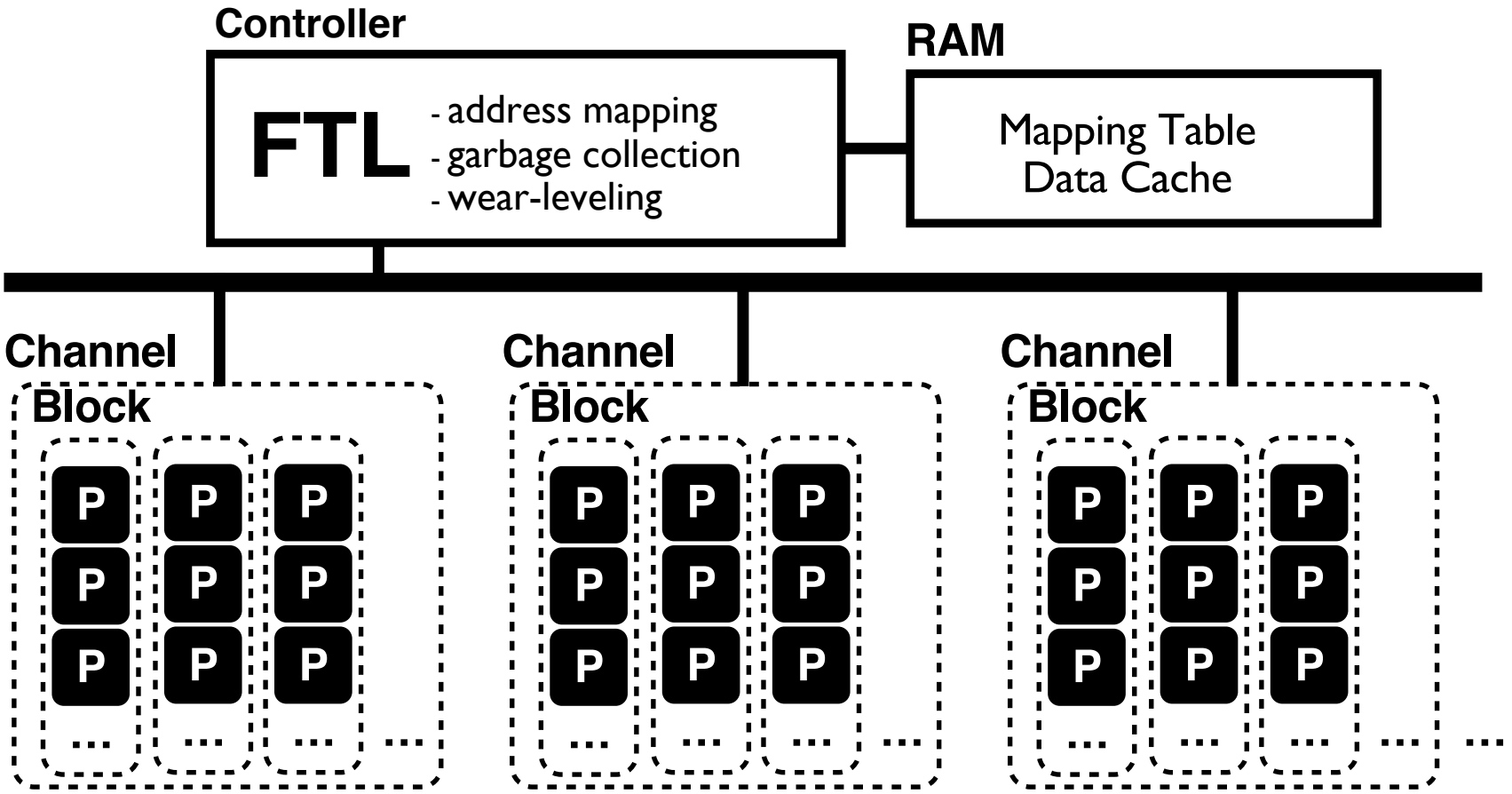
RAM needed: 

Too big, RAM is expensive!

Two solutions:

1. Keep (unused) **portion of Mapping Table in FLASH (→ Locality Rule)**
2. Hybrid-Mapping FTL (→ Aligned Sequentiality Rule)

SSD BACKGROUND



UNWRITTEN CONTRACT FOR SSDS

RULES OF THE UNWRITTEN CONTRACT

#1 Request Scale

#2 Locality

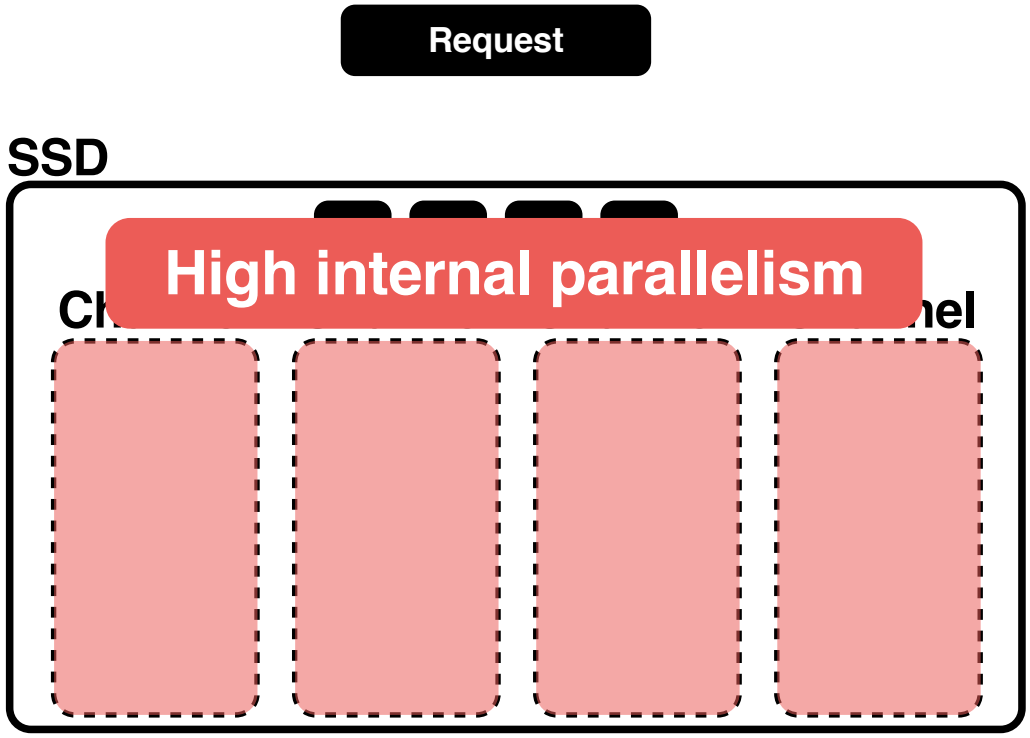
#3 Aligned Sequentiality

#4 Grouping by Death Time

#5 Uniform Data Lifetime

Rule #1: Request Scale

SSD clients should issue **large** data requests or **multiple** outstanding data requests.



Rule #1: Request Scale

Violation

If you violate the rule:

- Low internal parallelism

Performance impact:

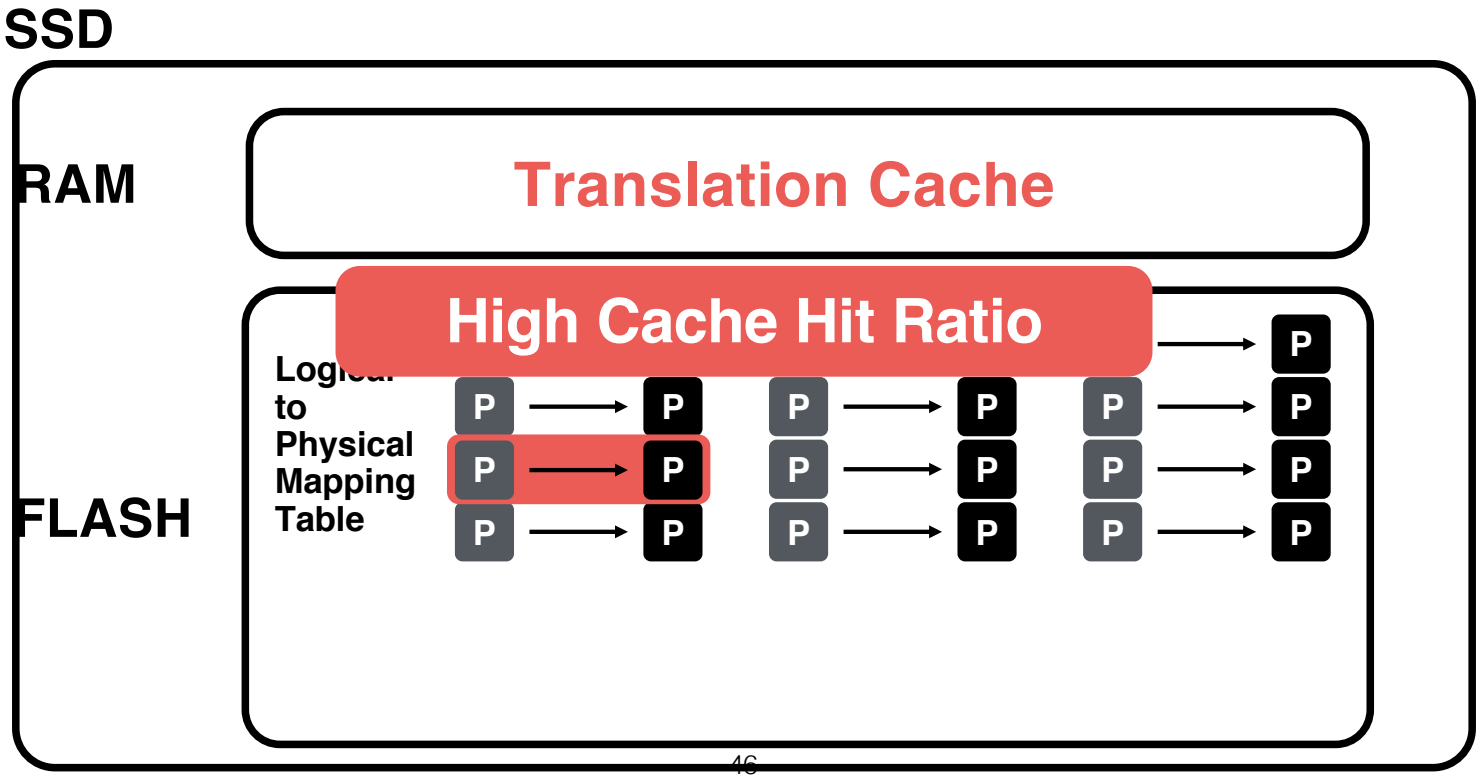
18x read bandwidth

10x write bandwidth

F. Chen, R. Lee, and X. Zhang. Essential Roles of Exploit- ing Internal Parallelism of Flash Memory Based Solid State Drives in High-speed Data Processing. In Proceedings of the 17th International Symposium on High Performance Com- puter Architecture (HPCA-11), pages 266–277, San Antonio, Texas, February 2011.

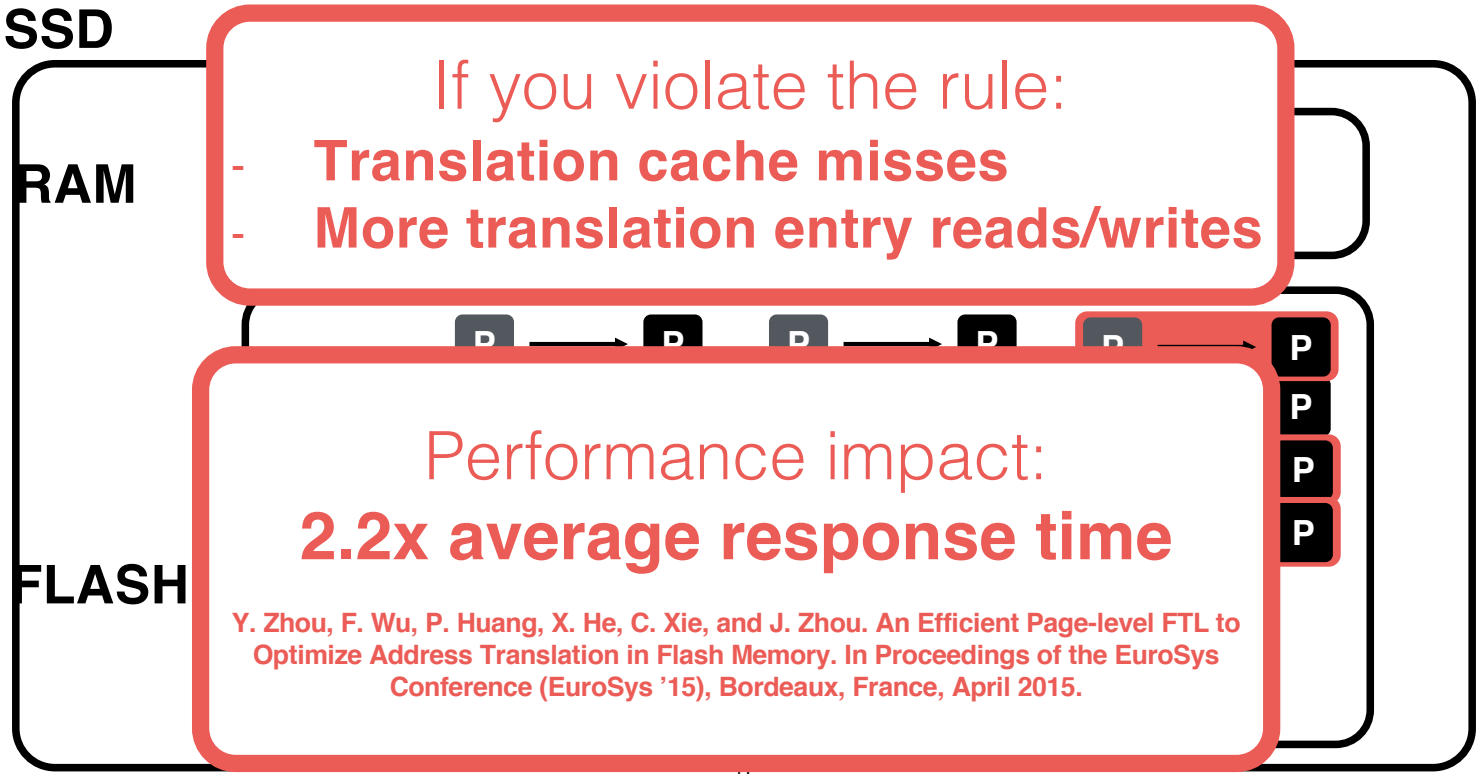
Rule 2: Locality

SSD clients should access with locality



Rule 2: Locality Violation

SSD clients should access with locality

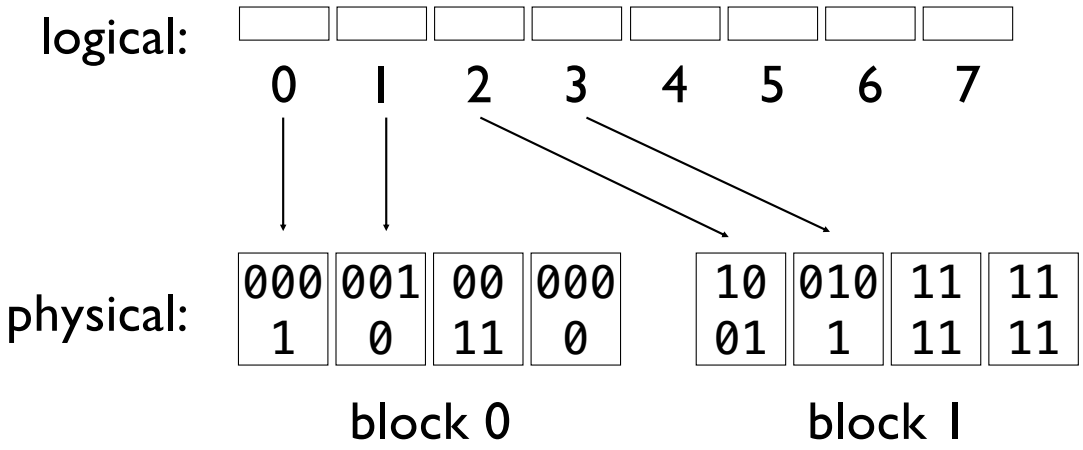


RULE 3: ALIGNED SEQUENTIALITY

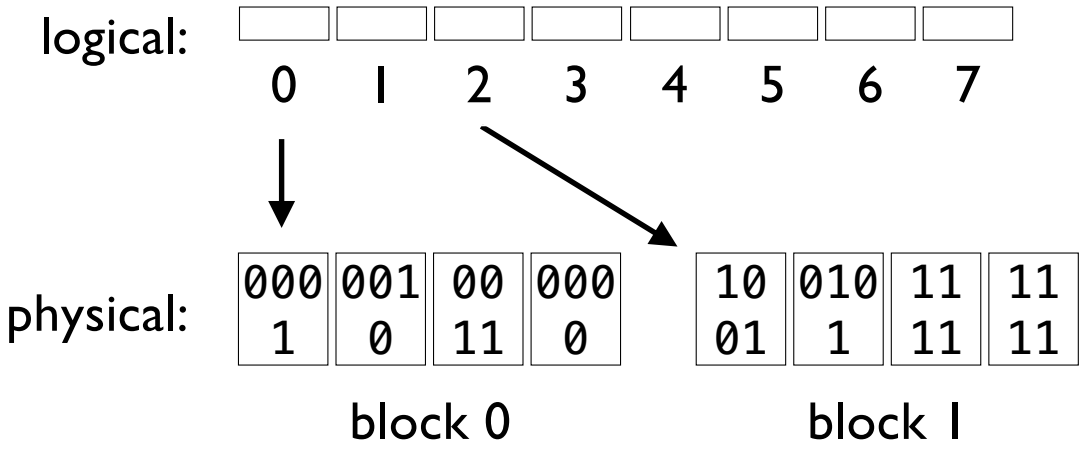
Hybrid mapping FTL:
To save space, compromise between page and block-level mappings

Aligned Sequentiality:
Perform best if write to entire block, starting from 1st page of block

PER-PAGE TRANSLATIONS

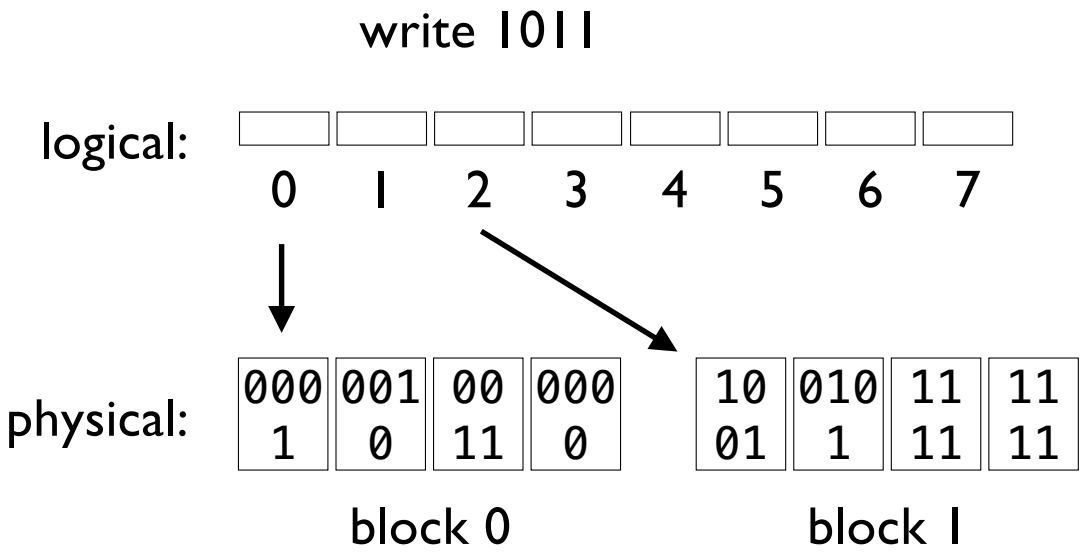


2-PAGE TRANSLATIONS (GENERALIZE...)

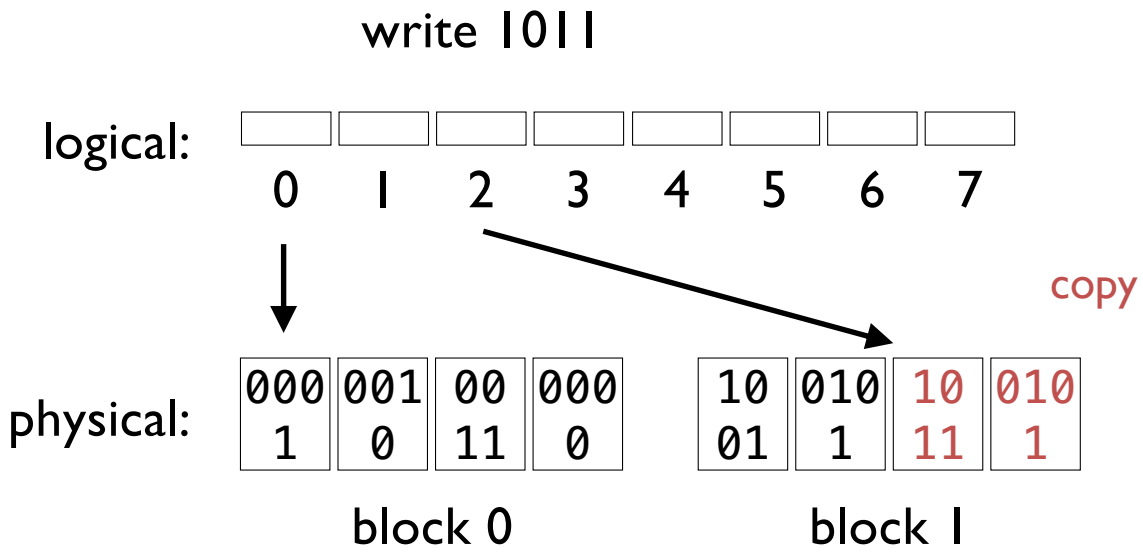


Advantage: Reduce number of translations!
Disadvantage?

2-PAGE TRANSLATIONS



2-PAGE TRANSLATIONS



Disadvantages?

- more read-modify-write updates
- more garbage
- less flexibility for placement

HYBRID FTL

Compromise between page and block-level mappings

Get advantages of each approach

Use course-grained mapping for most (e.g., 95%) of data

- Map at block level
- Most data → Most space savings

Use fine-grained mapping for recent data

- Map at page level
- Changing rapidly → Needs performance efficiency

LOG BLOCKS

Write changed pages to designated log blocks in Flash

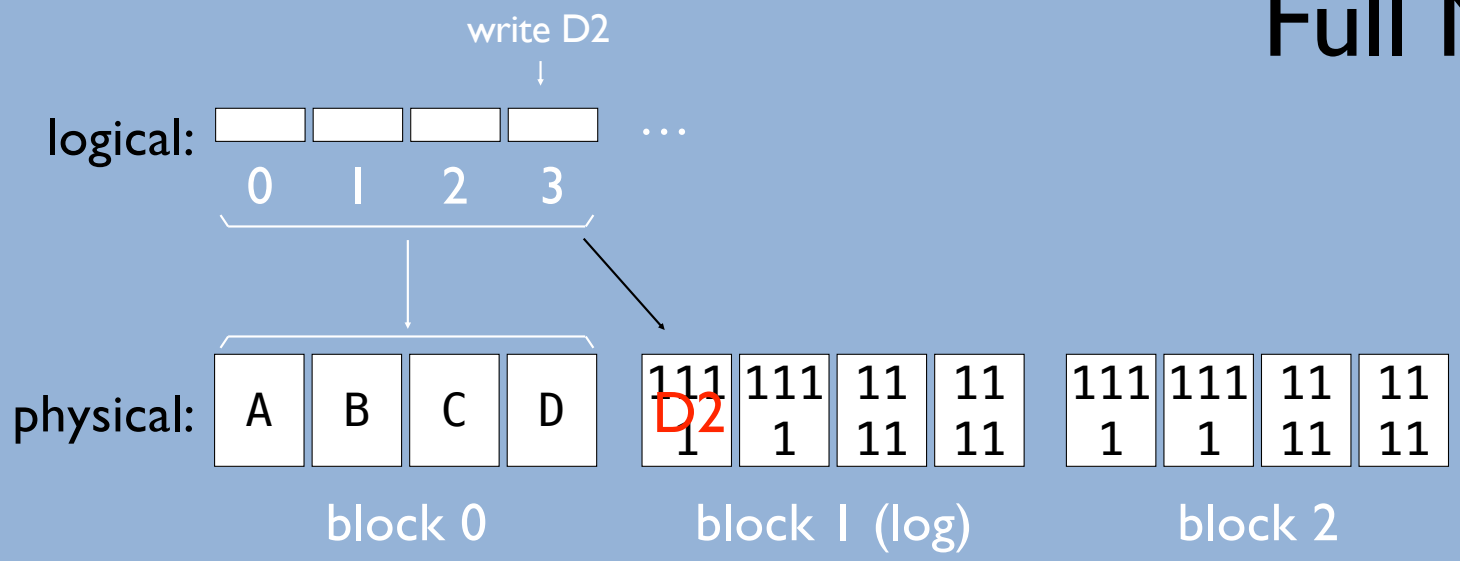
After log blocks become full, merge changes with old data

Type of merge depends on pattern of written data

- full merge (worst performance)
- partial merge
- switch merge (best performance)

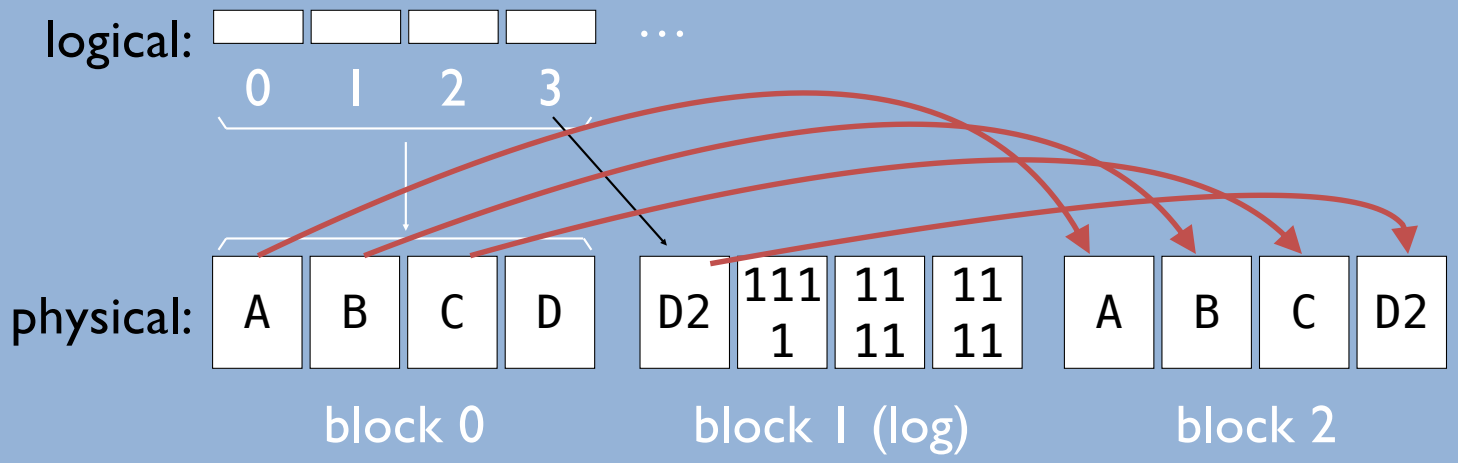
Eventually garbage collect old pages

Full Merge

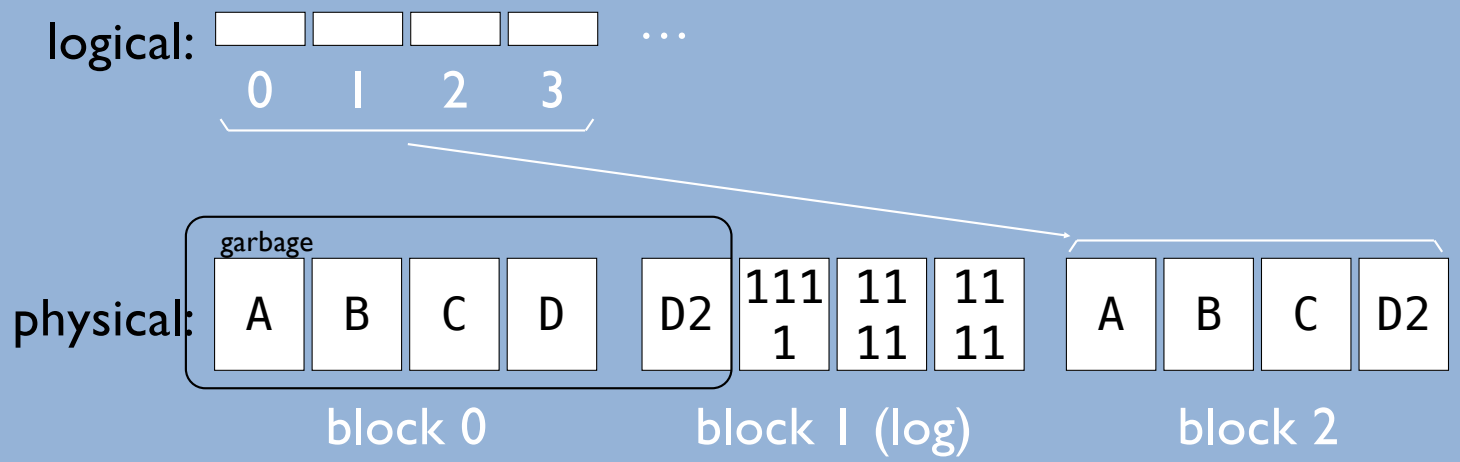


Eventually, need to get rid of page mappings because they are expensive

Full Merge


logical:
0
1
2
3
physical:
A
B
C
D
D2
111
1
11
11
11
11
A
B
C
D2
block 0
block 1 (log)
block 2

Full Merge

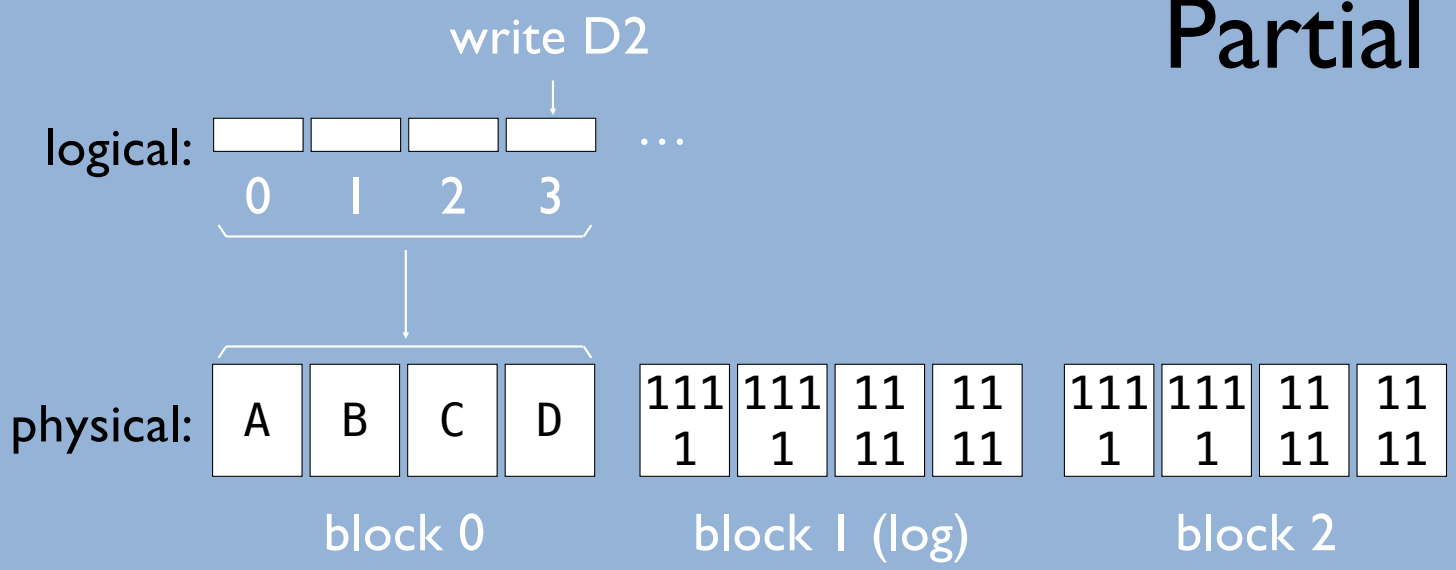

logical:
0
1
2
3
...
garbage
physical:
A
B
C
D
D2
111
1
11
11
11
11
A
B
C
D2
block 0
block 1 (log)
block 2

MERGING

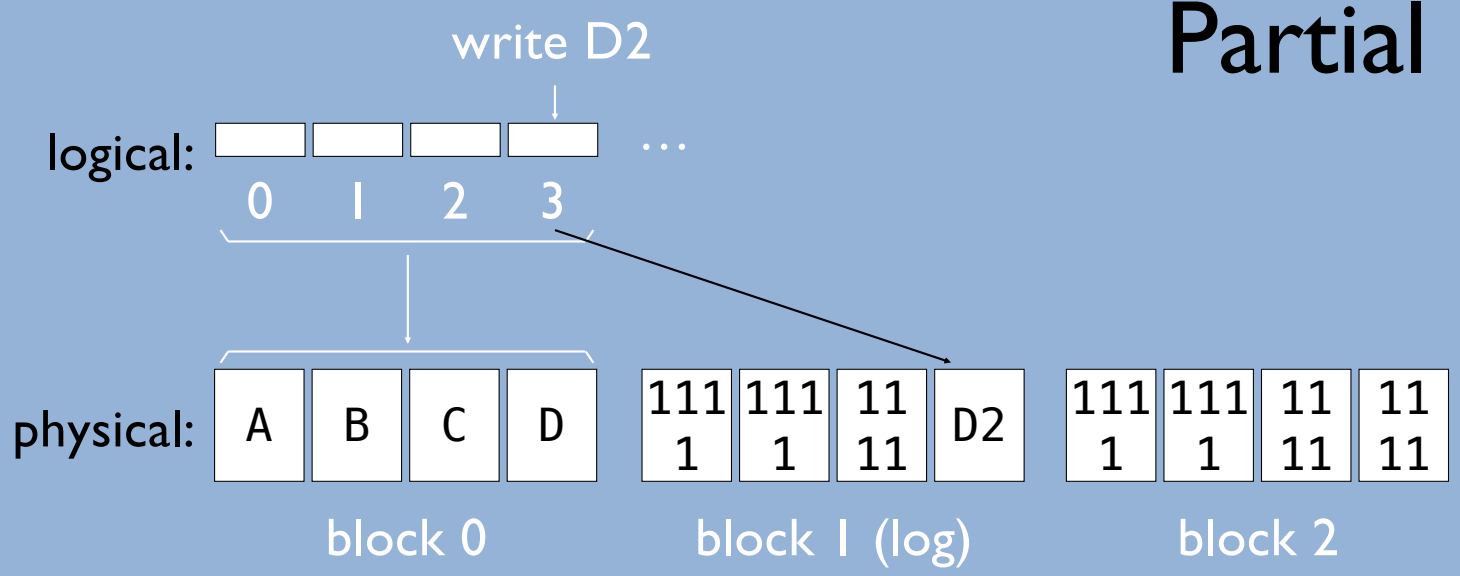
Three merge types:

- full merge (worst performance)
- partial merge
- switch merge (best performance)

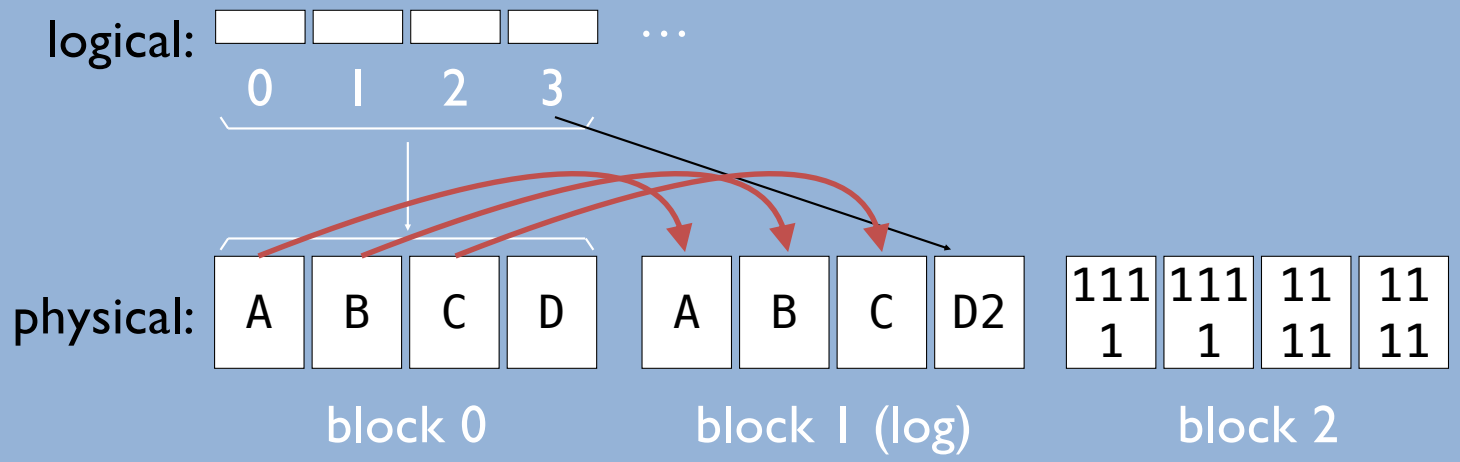
Partial Merge



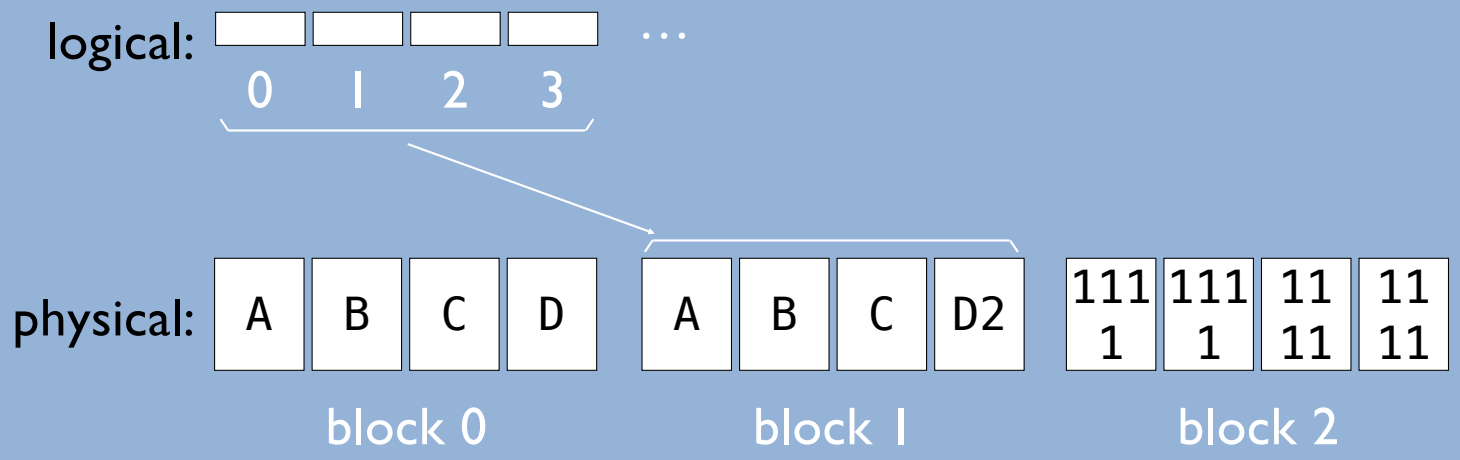
Partial Merge



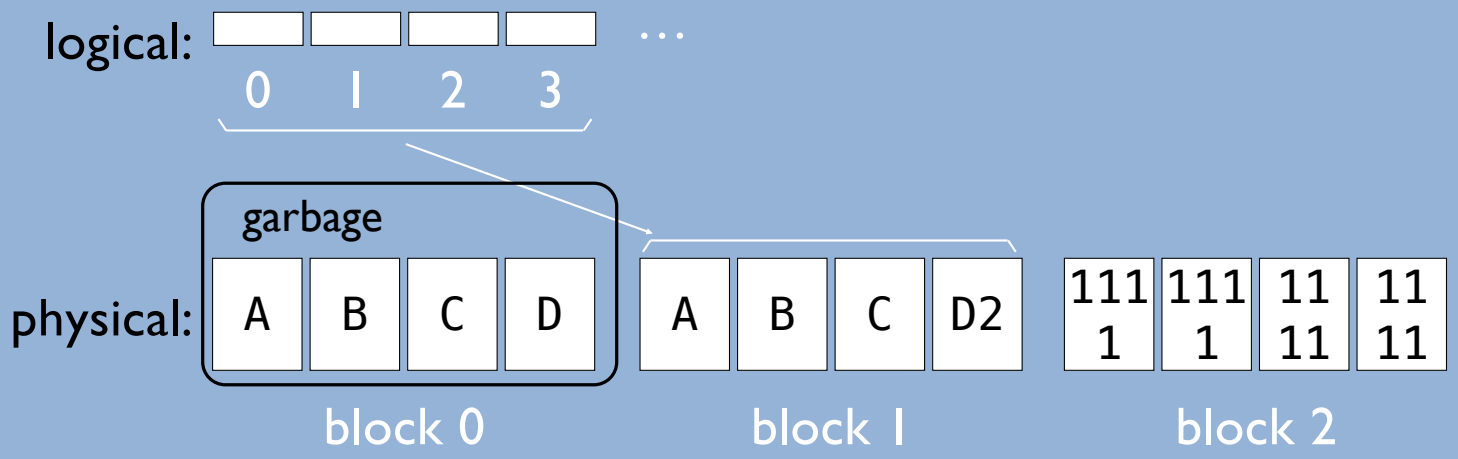
Partial Merge



Partial Merge



Partial Merge

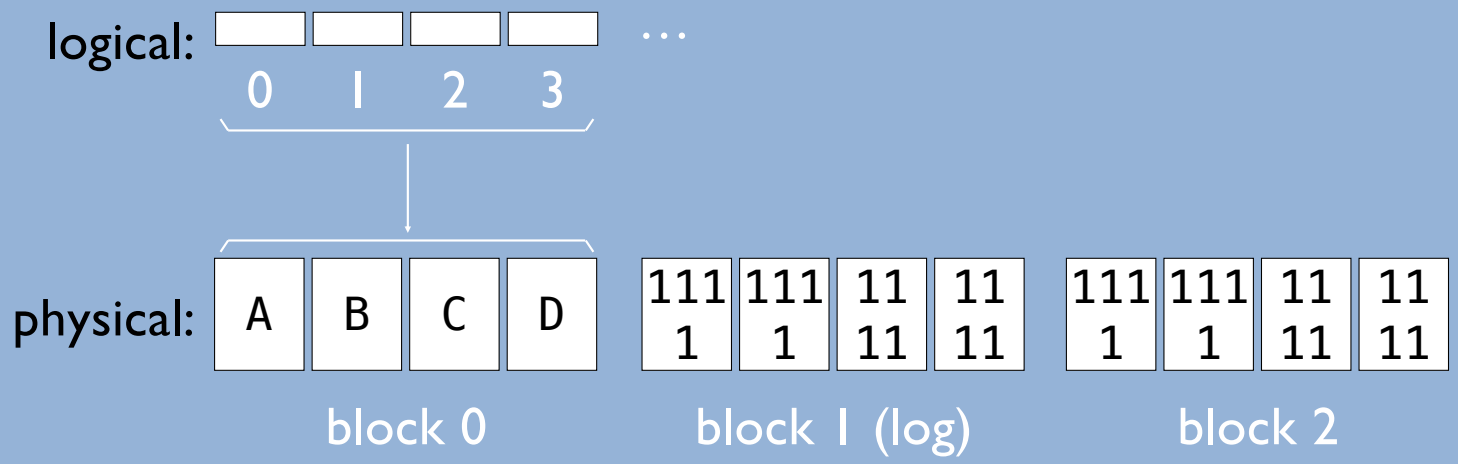


MERGING

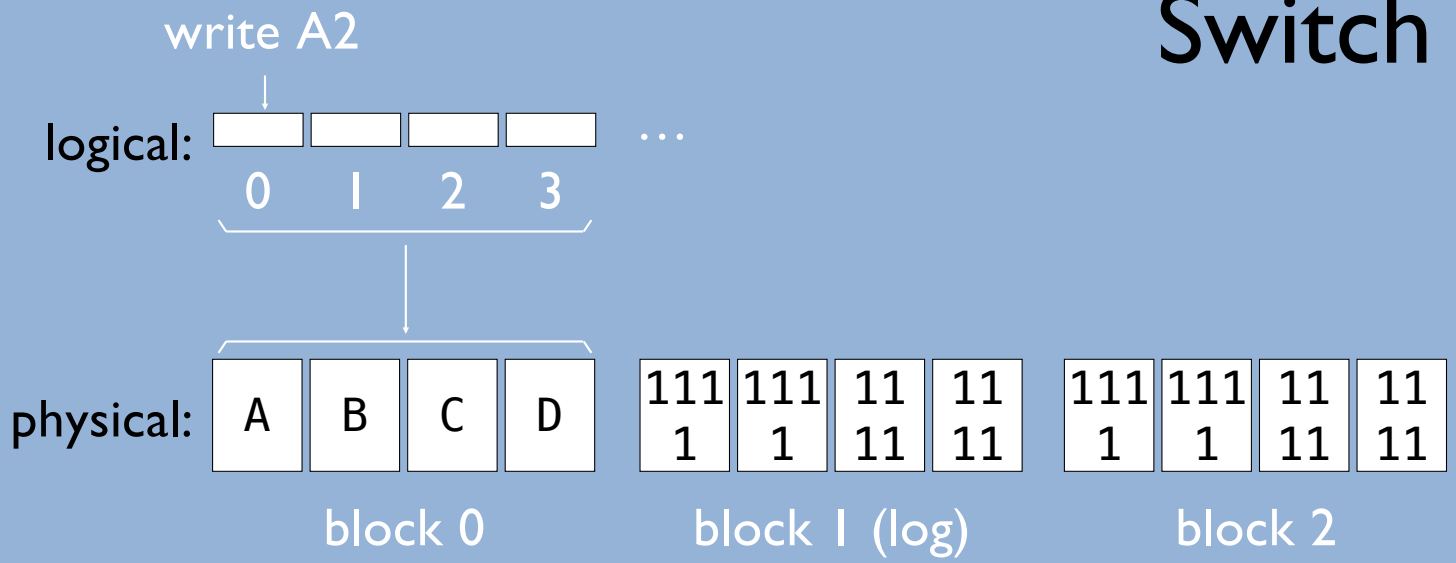
Three merge types:

- full merge (worst performance)
- partial merge
- switch merge (best performance)

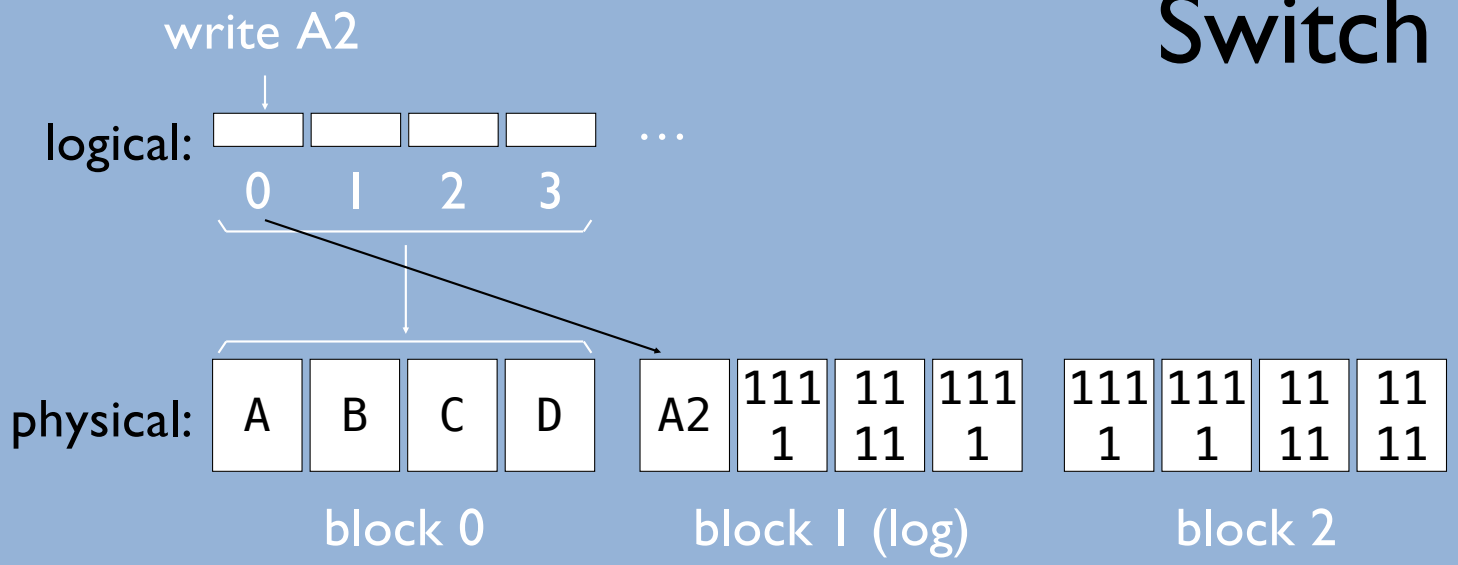
Switch Merge



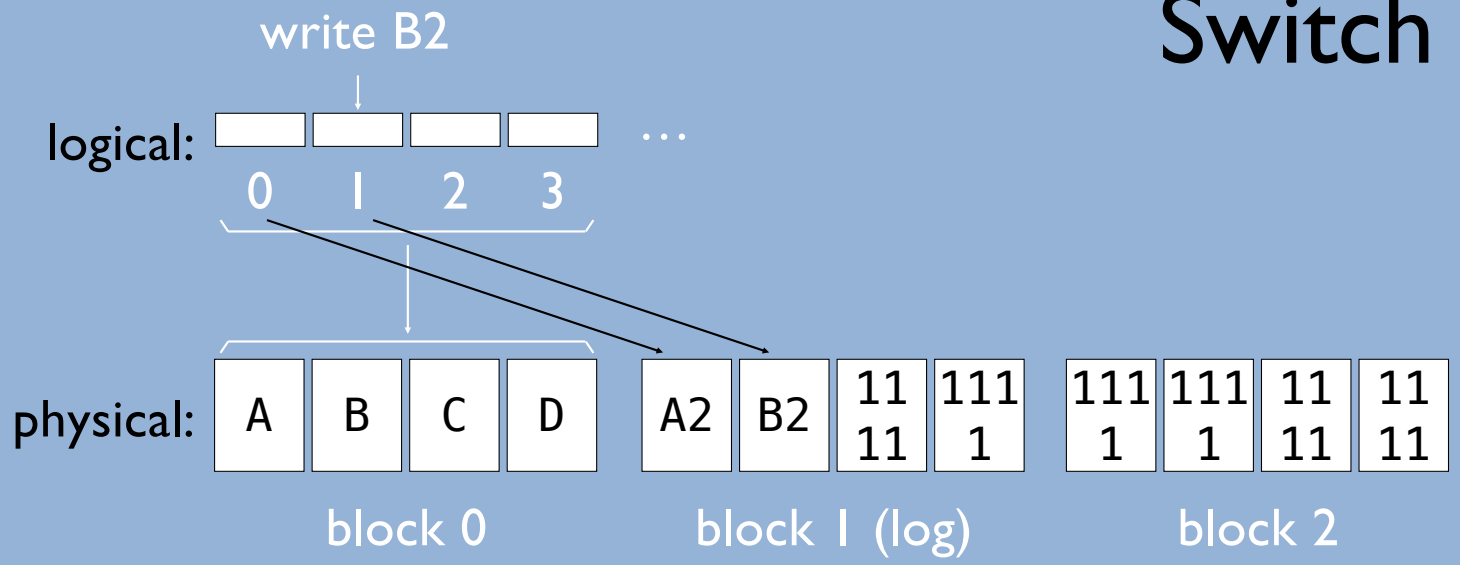
Switch Merge



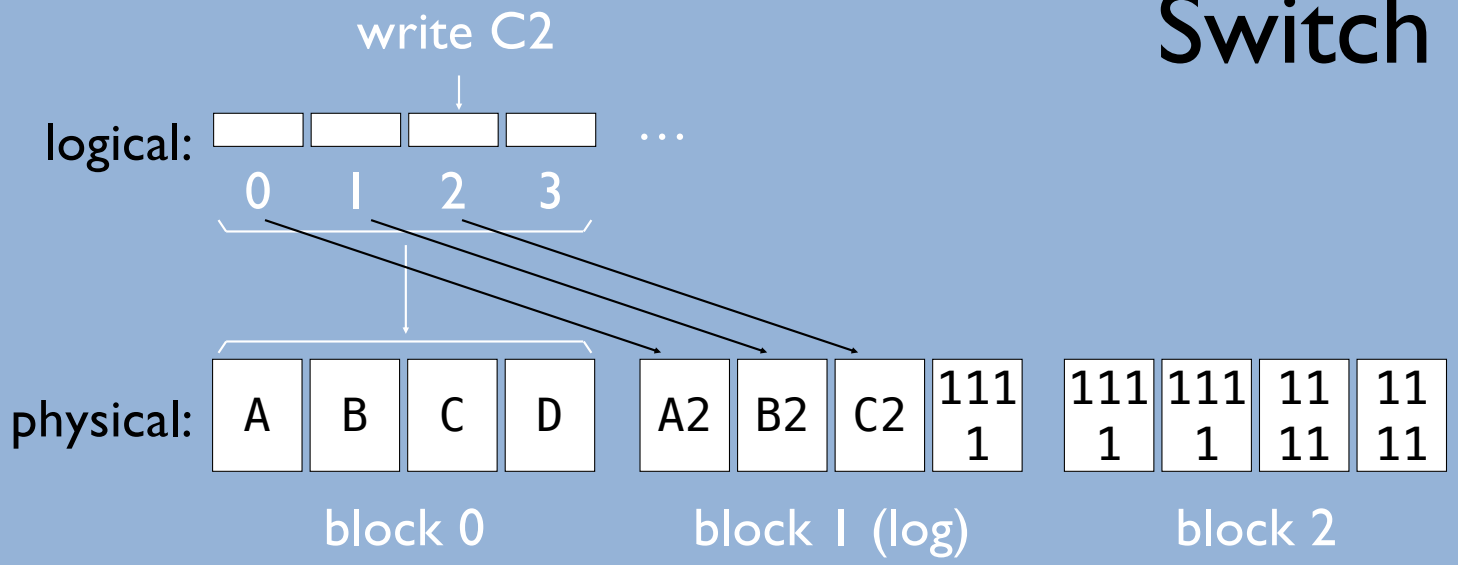
Switch Merge



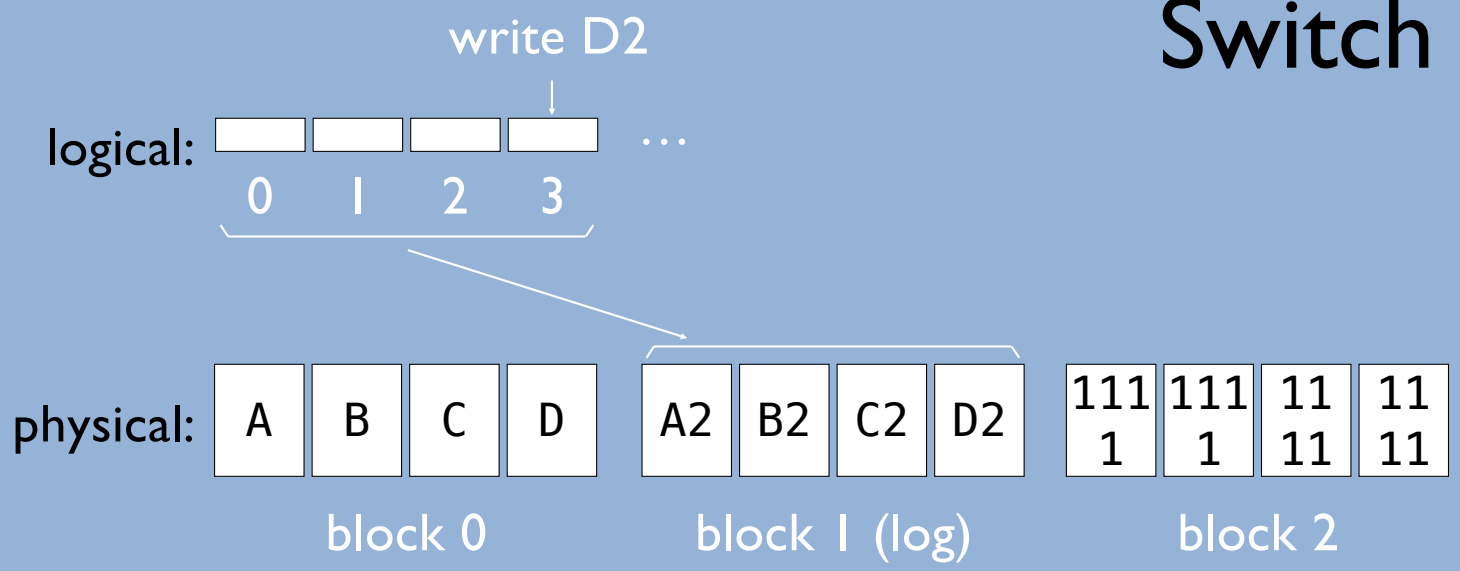
Switch Merge



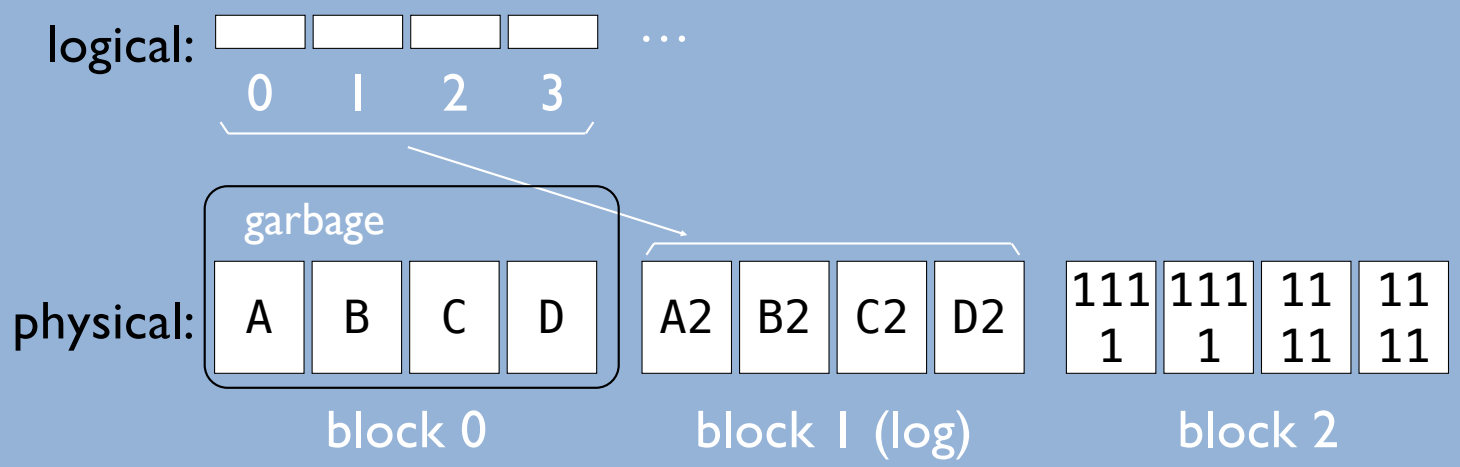
Switch Merge



Switch Merge



Switch Merge



Rule 3: Aligned Sequentiality

Hybrid mapping FTL:
Compromise between page and block-level mappings

Perform best if write to entire block, starting from 1st page of block

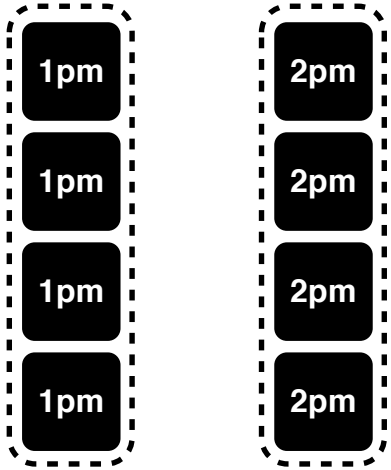
Does not need to copy pages from one block to the next

Rule 4: Grouping By Death Time

Data with similar death times should be placed in the same block.

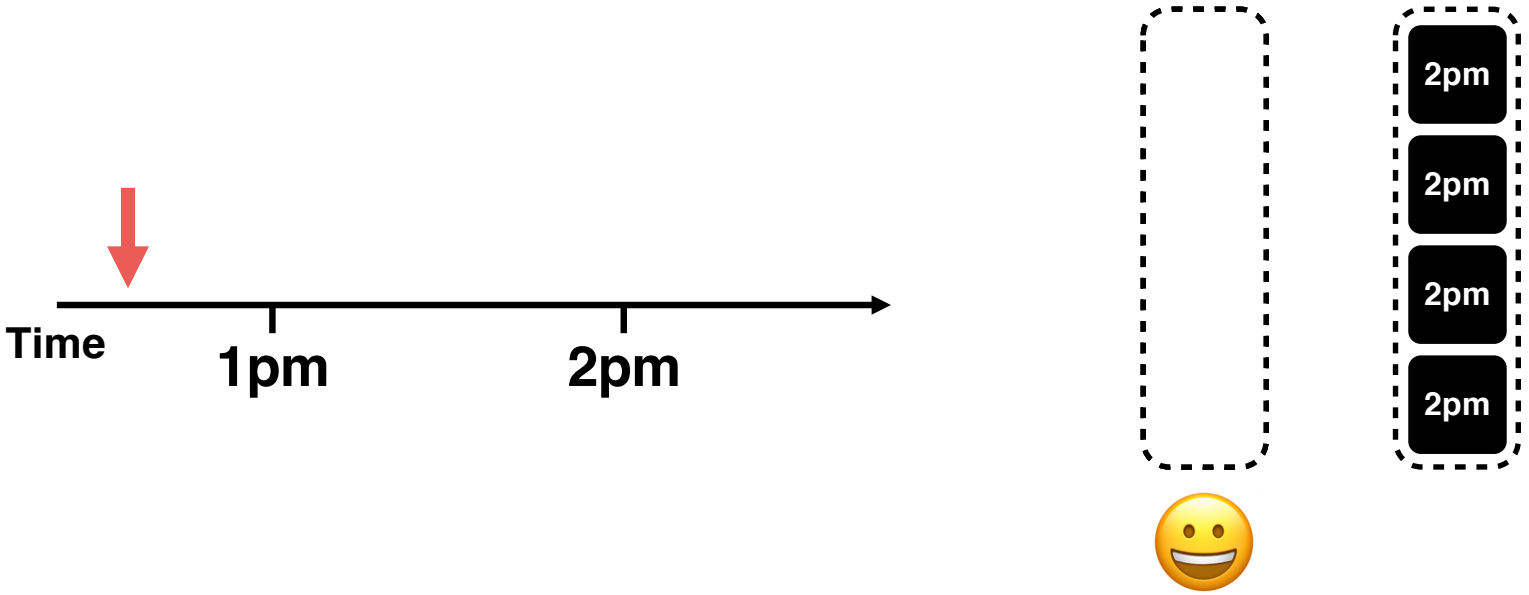
Rule 4: Grouping By Death Time

Data with similar death times should be placed in the same block.



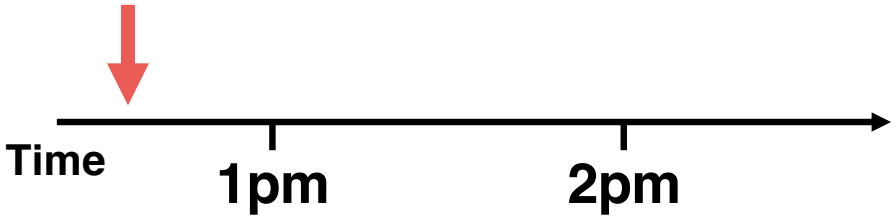
Rule 4: Grouping By Death Time

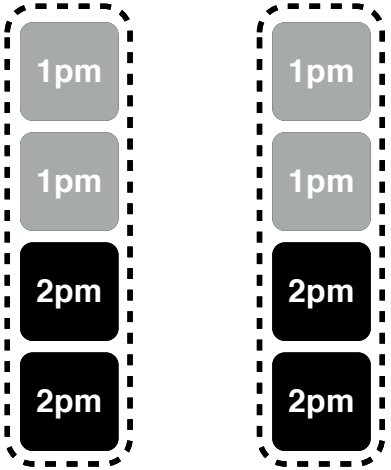
Data with similar death times should be placed in the same block.



Rule 4: Grouping By Death Time

Violation



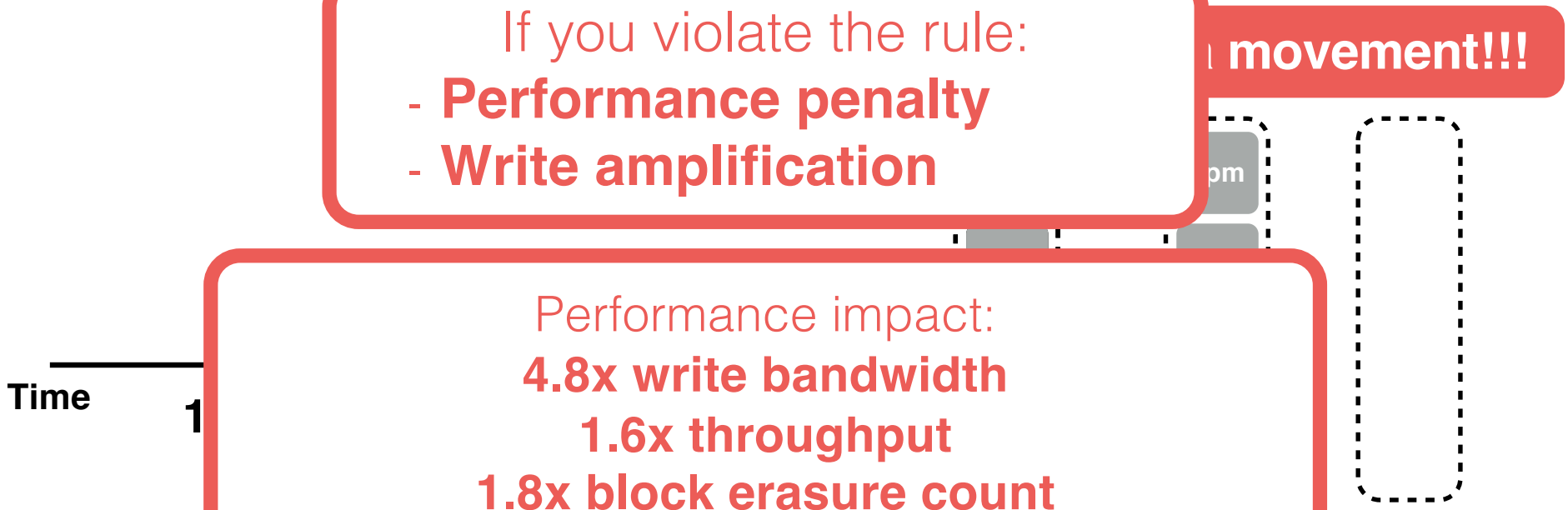


Rule 4: Grouping By Death Time

Violation

If you violate the rule:

- **Performance penalty**
- **Write amplification**



Performance impact:

4.8x write bandwidth

1.6x throughput

1.8x block erasure count

C. Lee, D. Sim, J.-Y. Hwang, and S. Cho. F2FS: A New File System for Flash Storage. In Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST '15), Santa Clara, California, February 2015.

J.-U. Kang, J. Hyun, H. Maeng, and S. Cho. The Multi- streamed Solid-State Drive. In 6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '14), Philadelphia, PA, June 2014.

. Y. Cheng, F. Douglis, P. Shilane, G. Wallace, P. Desnoyers, and K. Li. Erasing Belady's Limitations: In Search of Flash Cache Offline Optimality. In 2016 USENIX Annual Technical Conference (USENIX ATC 16), pages 379–392, Denver, CO, 2016. USENIX Association.

Rule 5: Uniform Data Lifetime

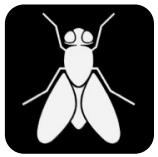
Clients of SSDs should create data with similar lifetimes

Lifetime

1 Day

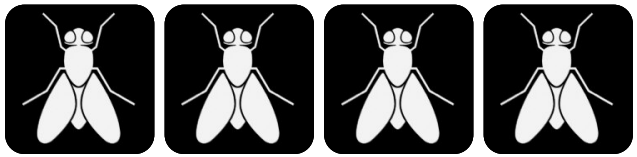
Rule 5: Uniform Data Lifetime

Clients of SSDs should create data with similar lifetimes



Lifetime

1 Day



Per Block

No wear-leveling needed

SSD

Usage Count:

Rule 5: Uniform Data Lifetime

Violation

Lifetime

1 Day

1000 Year

If you violate the rule:

- **Performance penalty**
- **Write amplification**

Per Block

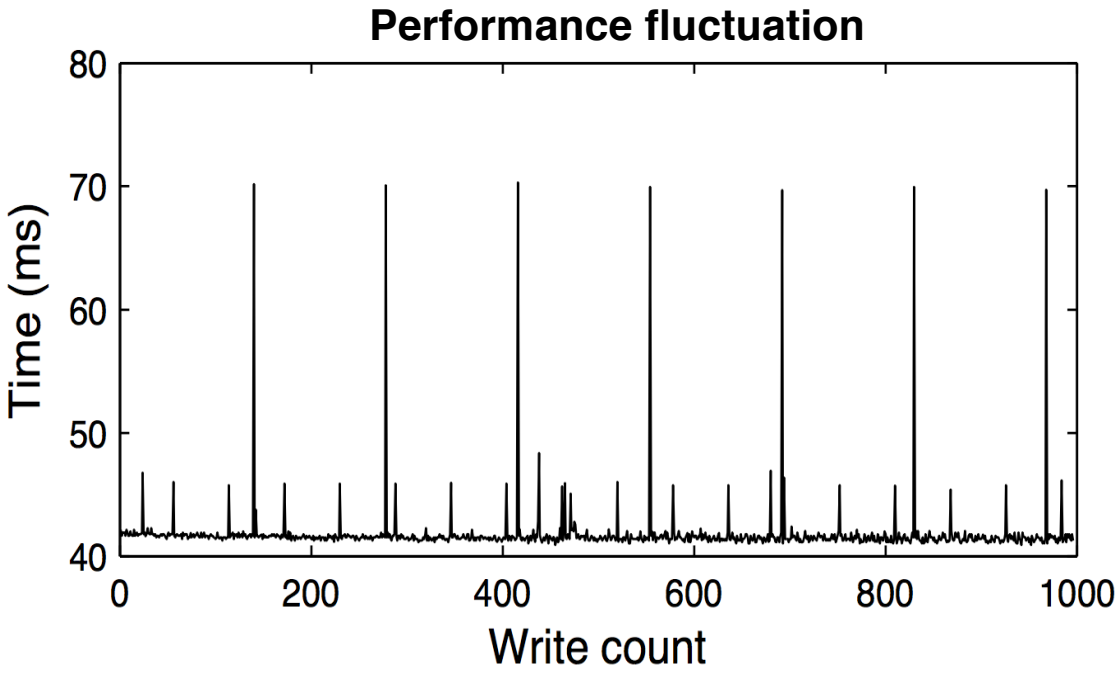
Performance impact:

1.6x write latency

S. Boboila and P. Desnoyers. Write Endurance in Flash Drives: Measurements and Analysis. In Proceedings of the 8th USENIX Symposium on File and Storage Technologies (FAST '10), San Jose, California, February 2010.

Usage Count:

Impact of Garbage Collection + Wear Leveling



Summary

Flash is much faster than disk, but…

More expensive

Not a drop-in replacement without a complex layer (FTL) for emulating hard disk API

Applications and file systems must carefully interact with Flash to obtain best performance

#1 Request Scale

#2 Locality

#3 Aligned Sequentiality

#4 Grouping by Death Time

#5 Uniform Data Lifetime