Recap

Physical Journaling Protocols

- Journal Write -> Journal Commit -> Checkpoint -> Free Transaction

Metadata Journaling Protocol

- (application) Data Write || Journal Metadata Write -> Journal Commit -> Checkpoint -> Free Transaction
- For deleting file/directory: never reuse data block before the deleted transaction is checkpointed; revoke the transactions of the deleted file/directory

Flash-based Solid State Drives

(Based on Ch. 44)

Revised from
Matthew Tancreti
Iowa State University

SSD

Solid-State Drive (SSD) is made from **flash memory**, a silicon transistor technology

Unlike HDD there are no moving parts – no spinning platters or mechanical arm

After decades, SSD becoming more popular that HDD, what are the consequences on file systems?

NAND-based Flash

A transistor forms a **cell** which stores a bit in a single-level cell or up to three bits in a trip-level cell

To write a value, the cell must first be erased, setting all bits to 1

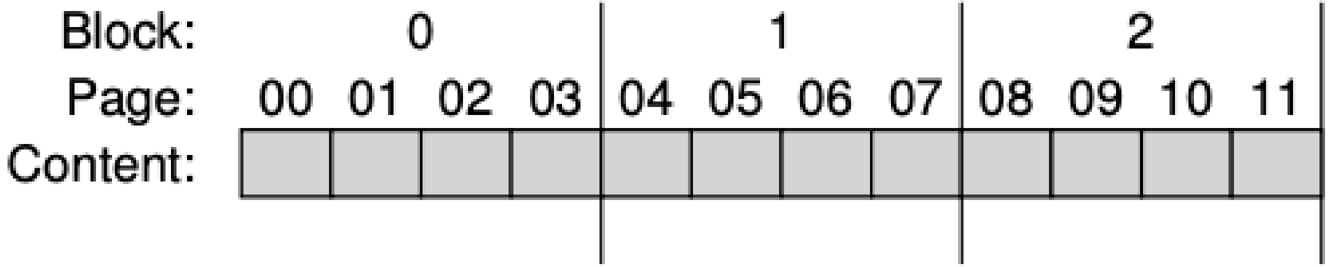
Only after a bit is set to 1 can it be set to 0

The need to **erase before write** is important to understanding SSD performance characteristics.

Data Organization

Hierarchically organized

- Individual bits stored in **cells**
- **Page** is typically 4KB
- **Block** is typically 128KB and consists of multiple pages
- **Bank** consists of multiple blocks



Warning: the words page and block have a different meaning for SSD than they have in other contexts, such as virtual memory or HDD

Basic Flash Operations

Flash operations are performed at the page/block level

Read (page) – client provides page number to read; relatively fast (e.g., 25μs); does not depend on location of page or previous page (**random access** device)

Erase (block) – before programming the block must be set to all 1; orders of magnitude slower than read (e.g., 1.5ms)

Program (page) – writes the page by setting 1's to 0's where needed; time is somewhere between a read and erase (e.g., 200μs)

HDD vs SSD Latencies

Rotational delay – time for sector to rotate under the disk head, 66μs for one rotation on fastest drives

Seek time – time for disk arm to change position to the correct track, 4ms typical

Read (page) – fast and random access, 25μs

Erase (block) – 1.5ms

Program (page) – 200μs

These are only latencies, must also consider transfer times

Example of Basic Flash Operations

Assume we want to write to page 0.

Starting state

Page 0	Page 1	Page 2	Page 3
00011000	11001110	00000001	00111111
VALID	VALID	VALID	VALID

Need to figure out what to do with pages 1 to 3, e.g., move them to another block.

Erase block

Page 0	Page 1	Page 2	Page 3
11111111	11111111	11111111	11111111
ERASED	ERASED	ERASED	ERASED

Program page 0

Page 0	Page 1	Page 2	Page 3
00000011	11111111	11111111	11111111
VALID	ERASED	ERASED	ERASED

Wear Out

Wear out is the issue that a flash block has a limited number of times (100,000) it can be erased before it becomes unusable

Suppose a program writes 100,000 times to a file, seems like a show stopper

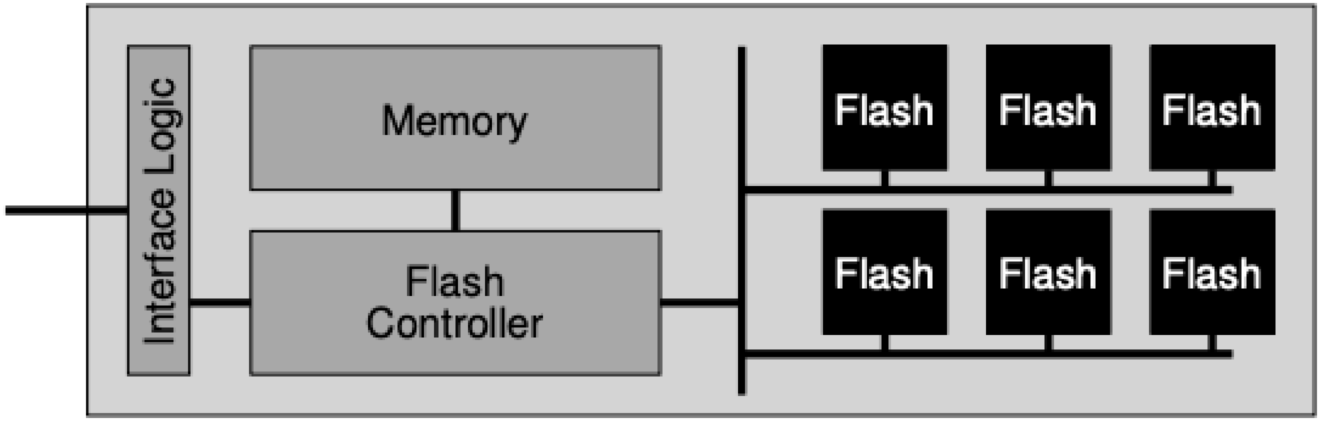
The SSD mitigates this problem with **wear leveling** – let blocks wear at the same pace

- Logic page address is independent of physical page on flash
- Every time a page is written, a new physical page is used to store the logic page
- In this way a logic page can be written to 100,000 times but every write is to a different physical location

Logical Diagram of SSD

A flash controller virtualizes the flash (i.e., mapping logical address to virtual address) in the **Flash Translation Layer (FTL)**

On chip memory is required for caching and buffering of blocks and pages



Log-structured File System (LFS): Brief Introduction

Motivation: to make writing efficient!

Key idea: by accumulating writes into large chunk and writing the chunk to contiguous space in disk sequentially (avoid random access whenever possible!)

How:

- Allocate a large segment in memory
- When requested to write to disk, LFS first buffers all updates (including metada) in a memory segment
- When the segment is full, the updates buffered there is written to disk in one long, sequential transfer to an unused part of the disk. (Appending only, NOT overwriting!)
- Old content (which should be updated) left in the disk becomes garbage and should be removed via garbage collection
- Need to carefully track the locations of data and metadata of files/directories, as they may change after updates

Log-Structured FTL

Motivations

- Large time cost of erasing a block before pages can be written
- Want to have wear leveling (-> independence between virtual and physical addresses)

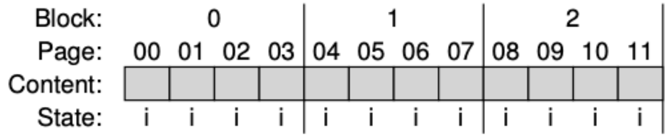
In **log-structured FTL** an in-memory table is used to map virtual to physical pages

On every write the page is moved to a different physical location

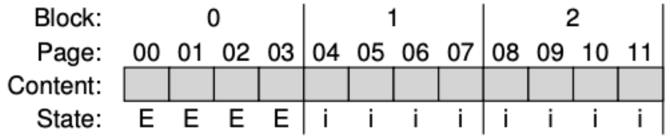
Log-Structured Example

Write(100) with contents a1
Write(101) with contents a2
Write(2000) with contents b1
Write(2001) with contents b2

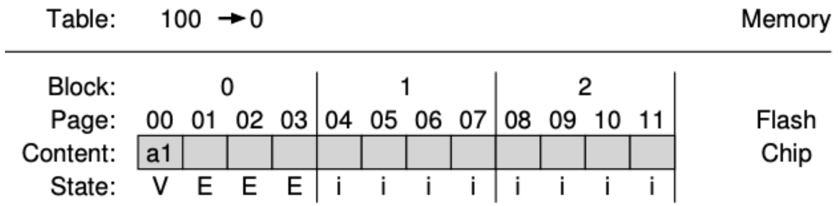
Block starts with all invalid



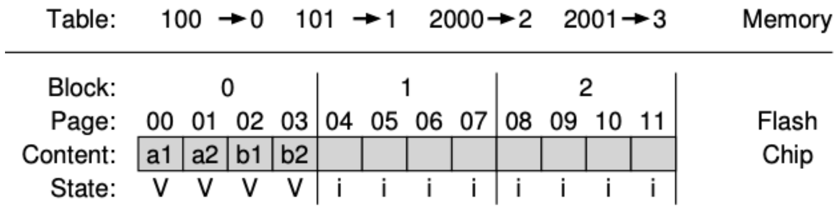
Erase block



Write a1 to first free physical
Page and log mapping



Write other pages to free
Physical pages and log
mappings



Garbage Collection

If the same logical page is written multiple times, the old versions of the page will remain in physical memory as garbage (unusable)

Table: 100 → 4 101 → 5 2000 → 2 2001 → 3 Memory

Block:	0				1				2				
Page:	00	01	02	03	04	05	06	07	08	09	10	11	Flash
Content:	a1	a2	b1	b2	c1	c2							Chip
State:	V	V	V	V	V	V	E	E	i	i	i	i	

Garbage collection is the reclamation of dead blocks

In example above, b1 and b2 can be moved to physical pages 6 and 7, then block 0 can be erased for reuse

Mapping Table Size

Mapping table can be very large

Assume 1TB SSD and 4 byte entry for each 4KB page, then map is 1GB

A **hybrid mapping** approach can map at either the page or block level, far fewer blocks so less mapping required

Example Performance of HDD vs SSD

Device	Random Reads (MB/s)	Random Writes (MB/s)	Sequential Reads (MB/s)	Sequential Writes (MB/s)
Samsung 840 Pro SSD	103	287	421	384
Seagate 600 SSD	84	252	424	374
Intel SSD 335 SSD	39	222	344	354
Seagate Savvio 15K.3 HDD	2	2	223	223

- For random I/O, SSD significantly outperforms HDD
- For sequential I/O, SSD still outperforms HDD
- SSD random write outperforms SSD random read; why?
- Sequential access always outperforms random access