Flash-based SSDs

CSC 343, Operating Systems

Topics covered in this lecture

- Flash-based SSDs

This slide deck covers chapters 44 in OSTEP.

Flash-based SSDs

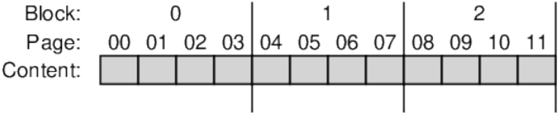
- Solid-state storage devices are built out of transistors, similar to memory and processors
- No mechanical or moving parts
- NAND-based flash is the technology behind Solid State Drives (SSDs) and has unique properties
  - To write a given chunk (flash page) a bigger chunk needs to be erased
  - Writing to a page too often will cause it to wear out

Storing a Single Bit

- Flash chips are designed to store one or more bits in a single transistor
  - Single-level cell (SLC): a single bit is stored
  - Multi-level cell (MLC): two bits are encoded into different levels of charge
  - Triple-level cell (TLC): three bits are encoded

Terminology

- Flash chips are organized into **banks** or **planes** which consist of a large number of cells
- A bank is accessed in two different sized units: **blocks** and **pages**
  - Blocks are typically 128 KB or 256 KB and pages are a few KB in size (for example 4 KB)



- To write to a page within a block, the block must first be erased

Basic Flash Operations

- Read (a page): a client can read any page by specifying the page number; typically fast (microseconds)
- Erase (a block): before writing to a page, the entire block the page is is within must be erased (all bits set to 1); typically slow (milliseconds)
- Program (a page): write data to a page by changing some of the ones to zeros; slower than reads, but faster than erases (100s of microseconds)

Flash States

- Pages start in an INVALID state
- Erasing a block sets all pages within that block to an ERASED state
- Programming a page results in a VALID state for the page
- Example state transitions:

	IIII	Initial: pages in block are invalid
Erase()	EEEE	State of pages in block set to erased
Program(0)	VEEE	Program page 0; state set to valid
Program(0)	error	Cannot reprogram page after programming
Program(1)	VVEE	Program page 1
Erase()	EEEE	Contents erased

Detailed Example

- Initial State

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

- Want to write to page 0; must erase first

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

- Program page 0

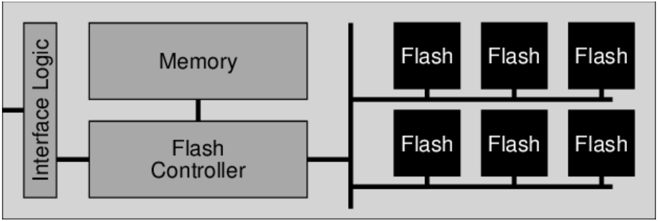
Page 0	Page 1	Page 2	Page 3
00000011	11111111	11111111	11111111
VALID	ERASED	ERASED	ERASED

- Problem: previous contents of Pages 1 to 3 are gone

Flash Performance and Reliability

- Wear out: when a flash block is repeatedly erased and programmed it builds up a little extra charge which makes it difficult to differentiate between 0 and 1.
    - Manufacturers rate MLC-based blocks to have a 10,000 P/E (Program/Erase) cycle lifetime; however, reserch indicates that lifetimes are much longer than expected
- Disturbance: when accessing a page it is possible that some bits get flipped from neighboring pages

Flash-Based SSDs



- A flash-based SSD is composed of
    - a number of flash chips
    - volatile memory (for caching and buffering data, etc.)
    - control logic (flash translation layer (FTL))

Flash Translation Layer

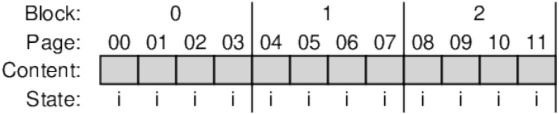
- The FTL takes read and write requests on *logical blocks* and converts them into low-level read, erase, and program commands on the underlying *physical blocks*
- A bad FTL organization approach: direct mapped – a read to logical page N is directly mapped to to a read of physical page N. A write to page N reads all pages in the block, erases the block, then programs the new page and all the old pages.

Log-Structured FTL

- FTLs today are log structured
  - A write to logical block N appends the write to the next physical free spot in the currently-being-written to block
  - To enable subsequent reads of block N, the device keeps a mapping table from logical addresses to physical addresses

Log-Structured FTL Example

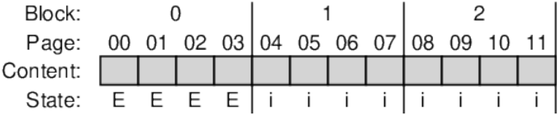
- Assume the following sequence of operations:
  1. Write(100) with contents `a1`
  2. Write(101) with contents `a2`
  3. Write(2000) with contents `b1`
  4. Write(2001) with contents `b2`
- Initial state



Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:												
State:	i	i	i	i	i	i	i	i	i	i	i	i

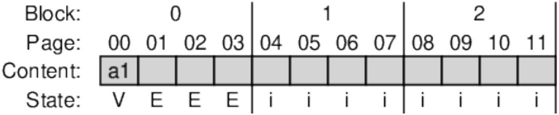
Log-Structured FTL Example

- Write `a1` to logical block 100; we first need to erase



Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:												
State:	E	E	E	E	i	i	i	i	i	i	i	i

- Direct logical block 100 to physical block 0



Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:	a1											
State:	V	E	E	E	i	i	i	i	i	i	i	i

Log-Structured FTL Example

- On a read, we need to find the physical block associated with the logical block; when the FTL writes logical block 100 to physical page 0, it records that fact in an in-memory mapping table

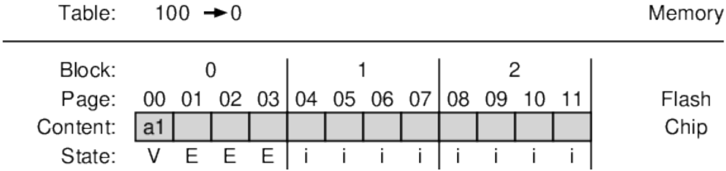


Table: 100 → 0 Memory

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:	a1											
State:	V	E	E	E	i	i	i	i	i	i	i	i

Flash Chip

- The final state

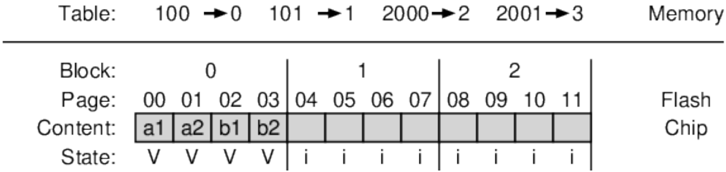


Table: 100 → 0 101 → 1 2000 → 2 2001 → 3 Memory

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

Flash Chip

Garbage

- Problem: log-structured approaches create garbage
- Continuing the previous example, assume that blocks 100 and 101 are written to again with contents c1 and c2:

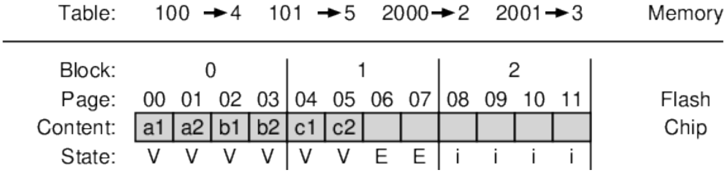


Table: 100 → 4 101 → 5 2000 → 2 2001 → 3 Memory

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

- Now physical pages 0 and 1, although marked valid, contain garbage

Garbage Collection

- Garbage collection is the process of finding garbage blocks and reclaiming them for future use.
- Process:
  1. Find a block that contains one or more garbage pages
  2. Read in the live (non-garbage) pages from that block
  3. Write those pages to the log
  4. Reclaim the entire block for future writes

Garbage Collection Example

- Initial state:

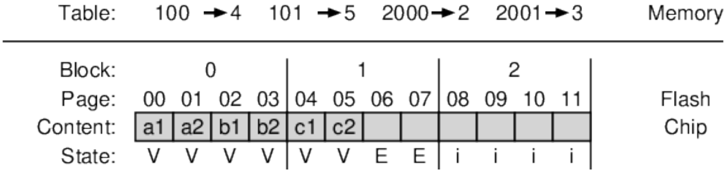


Table: 100 → 4 101 → 5 2000 → 2 2001 → 3 Memory

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	

- After collection:

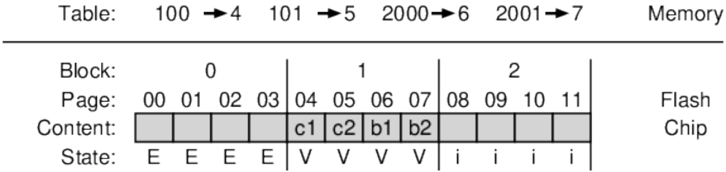


Table: 100 → 4 101 → 5 2000 → 6 2001 → 7 Memory

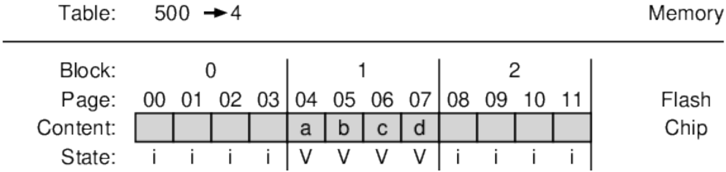
Block:	0				1				2				
Page:	00	01	02	03	04	05	06	07	08	09	10	11	Flash
Content:					c1	c2	b1	b2					Chip
State:	E	E	E	E	V	V	V	V	i	i	i	i	

Block-Based Mapping

- Another cost of the log-structured approach is the potential for extremely large mapping tables
- Block-level FTL: an approach to reduce costs of mapping by keeping a pointer per block of the device instead of per page
- Block based mapping reduces the amount of mapping information by a factor of $\frac{\text{block size}}{\text{page size}}$

Block-Based Mapping Example

- The logical block address consists of two portions: a chunk number and an offset (similar to virtual memory). Here assume logical blocks 2000, 2001, 2002, and 2003 have the same chunk number (500).



Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:					a	b	c	d				
State:	i	i	i	i	V	V	V	V	i	i	i	i

Block-Based Mapping Example

- A write to logical block 2002 with content `c'` requires reading in 2000, 2001, and 2003 and writing out all four logical blocks in a new location.

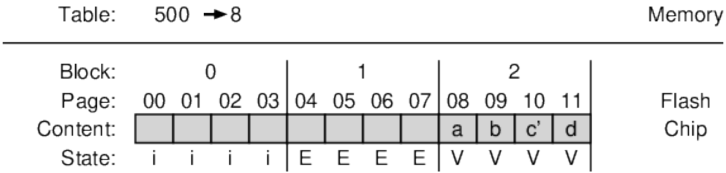


Table: 500 → 8

Memory

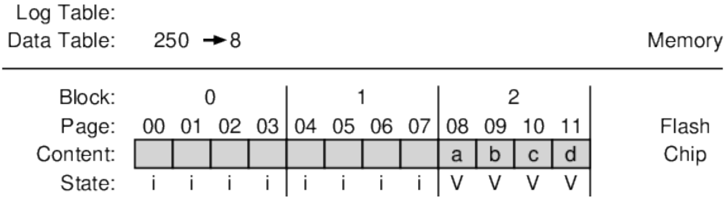
Block:	0				1				2				
Page:	00	01	02	03	04	05	06	07	08	09	10	11	Flash
Content:									a	b	c'	d	Chip
State:	i	i	i	i	E	E	E	E	V	V	V	V	

Hybrid Mapping

- A block level mapping greatly reduces the amount of memory needed for translations, however it causes significant performance problems when a write is smaller than the physical block size.
- A hybrid mapping approach keeps a few blocks erased and directs all writes to them; these are called log blocks.
- The FTL keeps a per-page mapping for the log blocks.
- Goal: keep the number of log blocks small (to keep the per-page mapping small) and switch them into blocks that can be pointed to by a single block pointer.

Hybrid Mapping Example

- Assume logical pages 1000, 10001, 10002, and 1003 are mapped to physical block 2



Log Table:

Data Table: 250 → 8

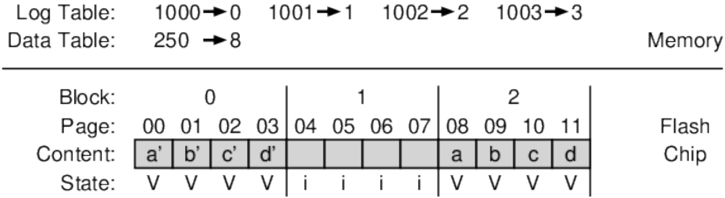
Memory

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:									a	b	c	d
State:	i	i	i	i	i	i	i	i	V	V	V	V

Flash Chip

Hybrid Mapping Example

- Now, assume the client overwrites each of these blocks (with data `a'`, `b'`, `c'`, and `d'`) in the exact same order:



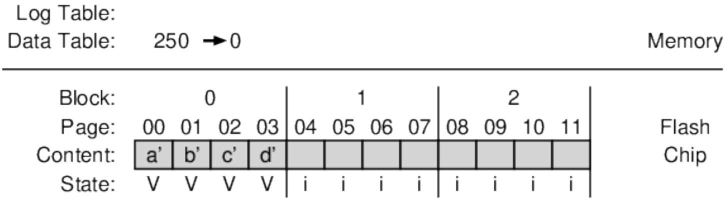
Log Table: 1000 → 0 1001 → 1 1002 → 2 1003 → 3
Data Table: 250 → 8

Memory

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:	a’	b’	c’	d’					a	b	c	d
State:	V	V	V	V	i	i	i	i	V	V	V	V

Flash Chip

- The blocks have been written in order, so the FTL can perform a **switch merge**



Log Table:
Data Table: 250 → 0

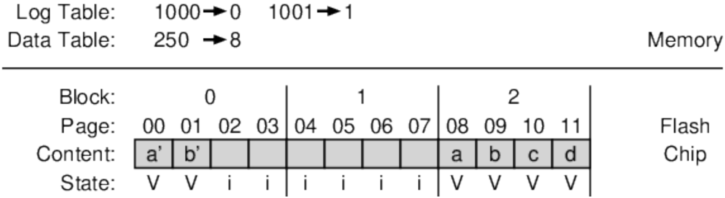
Memory

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:	a’	b’	c’	d’								
State:	V	V	V	V	i	i	i	i	i	i	i	i

Flash Chip

Hybrid Mapping Example

- Now, assume the client only overwrites logical blocks 1000, and 1001 from the initial state:



Log Table: 1000 → 0 1001 → 1
Data Table: 250 → 8
Memory

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:	a’	b’							a	b	c	d
State:	V	V	i	i	i	i	i	i	V	V	V	V

Flash Chip

Wear Leveling

- Idea: because multiple erase/program cycles wear out a flash block, spread the work across all the blocks evenly
- Log-structuring does a good initial job of spreading out write load; garbage collection helps as well
- Problem: sometimes a block will be filled with long-lived data that does not get overwritten; the garbage collector will never reclaim the block so it does not receive a fair share of the write load
- Solution: periodically read all the live data out of such blocks and rewrite it elsewhere making the block available for future writes