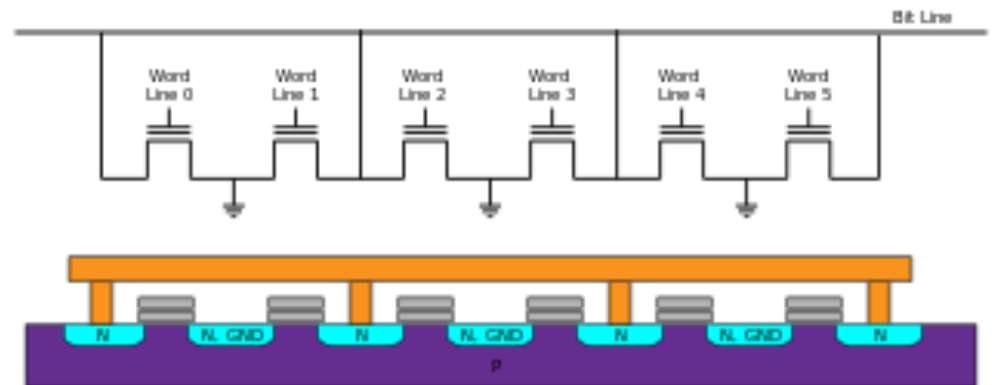
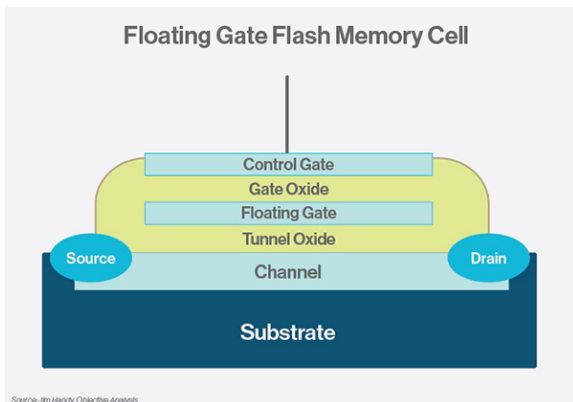


44. Flash-Based SSD

Operating System: Three Easy Pieces

Non-volatile RAM?

- ❑ Solid-State Storage Devices (SSD)
 - ◆ Add persistence to a electron based device?
 - ◆ Flash is one the most successfully approaches
- ❑ Flash memory
 - ◆ Based on hot-electron injection (quantum effect)
 - ◆ Most common NAND Flash (better cost/GB)

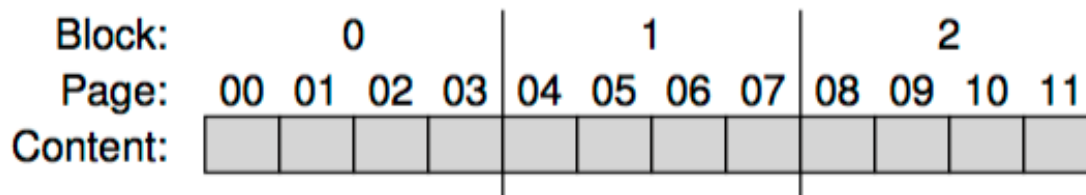


Some peculiarities about NAND Flash based SSD (vs DRAM)

- ▣ Information should be accessed in large “chunks”
- ▣ Better information per mm² than DRAM
- ▣ Electrons are kept trapped in the floating gate (even with no power)
 - ◆ Non-volatility
- ▣ Writing is a stressful operation
 - ◆ Large tension should be applied (>12V) to inject electrons
 - ◆ 1000x-10000x write cycles before break down the cell
- ▣ Not the only approach
 - ◆ Phase change RAM
 - ◆ Filamentary RAM
 - ◆ Ion RAM

From bits to Banks/Planes

- ▣ Each cell can store 1-bit (SLC), 2-bits (MLC) or 3-bits (TLC)
 - ◆ Take a look at [j10] if want to understand more beyond our scope
- ▣ Flash chips are structured in “planes” (or so called banks)
 - ◆ Banks are divided in **blocks** (or so called erase blocks):
 - Typically 128-64KB
 - ◆ Each block is divided in **pages**
 - Typically 4KB



FLASH Operations

- ▣ Read (a page)
 - ◆ Address the content with bit line -> Get the content through the word lines
 - ◆ 10s of microseconds and independent of the address of las access

- ▣ Erase (a block)
 - ◆ Before to write a page we need to erase all the block where belongs
 - ◆ "Save" the pages that we want to keep before to erase
 - ◆ Quite expensive process (few milliseconds)

- ▣ Program (a page)
 - ◆ 100s of microseconds

- ▣ Each page has a state associated: (i)nvalid, (e)rased, (v)alid
 - ◆ Reads don't change those states
 - ◆ Writes has to follow some rules (iiii→eeee→vvee and vvee→eeee→vvve)

Detailed Example

- 4-page blocks of 8-bits with initial state

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

- Want to write page 0 with new content. Need to erase the block.

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

- Before to erase we need to handle P1, P2 and P3 content (move it somewhere in the flash)

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

- Lots of writing and write "wear out" the cells! How deal with it?

Flash Performance and Reliability

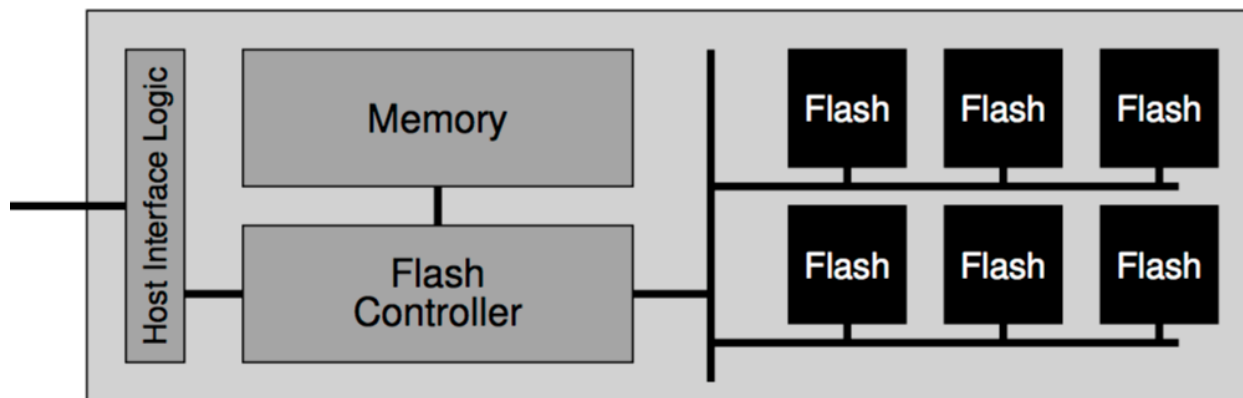
- Not all low level devices are equal: higher density => lower speed, lower reliability (endurance)

Device	Read (us)	Program(us)	Erase(us)	P/E cycles
SLC	25	200–300	1500–2000	$\sim 10^5$
MLC	50	600–900	$\sim 3000'$	$\sim 10^4$
TLC	~ 75	~ 900 –1350	~ 4500	$\sim 10^3$

- Reliability has two stand-points:
 - Permanent cell wear out
 - Read disturbs or program disturbs
- Higher density means usually lower reliability
 - 3D NAND ($\sim 10^3$ or lower)
- Really vulnerable to “malicious” attacks

From Flash to SSD

- SSD has a variable number of Flash chips, some SRAM and logic



- ◆ Usually some spare capacity and flash
- Flash Translation layer (FTL) is performed by Flash controller
 - ◆ Balance wear out of the cells (then the reliability is amortized by the disk size and sparing). **Wear leveling.**
 - ◆ Reduce **write amplification**, i.e. minimize the write traffic to the flash chips

FTL Organization: a bad approach

- ▣ Direct mapped
 - ◆ Each logical page N is mapped in a particular physical page N across all the lifetime of the device

- ▣ Problems:
 - ◆ Performance problems: amplification write
 - The block has to be erased and written in the same chip (sequentially), making it slower than mechanical disks
 - ◆ Reliability: premature wear-out
 - To write repeatedly the same data (v.gr. Metadata) will physically damaging the cells
 - Since the mapping is exposed to the client, malicious code can damage the disk

A Log-Structured FTL

- For different reasons to mechanical disks, LFS is also a good idea here
 - ◆ Note that FTL is not FS!
- Lets use a example:
 - ◆ From client perspective 512-B sectors, writes and reads 4-KB chunks
 - ◆ SSD uses (unrealistically small) 16-KB blocks, 4-KB pages
 - ◆ Four OPs: Over different logical addresses (Internally the device choses a phicial page for each logical adress
 - Write (100) with a1. Write(101) with a2, Write(2000) with b1, Write (2001) with b2

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

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

A Log-Structured FTL (cont.)

Update translation table

Table:	100 → 0												Memory
Block:	0				1				2				Flash Chip
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	

After all operations done (as a single write in the device)

Table:	100 → 0	101 → 1	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				
State:	V V V V	i i i i	i i i i		

- Successive writes (to the same logical pages) will be spread across the whole device
 - Some natural form of wear-levelling

Garbage collection

- Lets assume we want to write c1 and c2 to 100 and 101

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	

- Some pages has garbage (0,1). We we want to reclaim such capacity we need to reclaim the whole block
- Ex) System needs to reclaim page 0 and 1

Table:	100	→	4	101	→	5	2000	→	6	2001	→	7	Memory
Block:	0				1				2				Flash Chip
Page:	00	01	02	03	04	05	06	07	08	09	10	11	
Content:					c1	c2	b1	b2					
State:	E	E	E	E	V	V	V	V	i	i	i	i	

Garbage Collection

- ▣ Many reads and writes
 - ◆ Might be expensive
- ▣ To ease the effect many disk are overprovisioned (10-20%)
 - ◆ Allows to perform GC when the system is less used (in **background**)
- ▣ Adding more capacity increases internal bandwidth (more flash chips), that can be used to perform GC with no harm in client side
- ▣ If FS is also LFS, garbage collector can be coordinated

Mapping Table Size

■ Block-Based Mapping

- ◆ Only keep a pointer for the block
- ◆ Block-level FTL is akin to having bigger page sizes (less for VPN) and larger offset
- ◆ BL-FTL don't work very well with LFS-FTL: "small write" problem, which Increase write amplification
 - Ex) Client wrote logical blocks 2000,2001,2002, and 2003 logical pages with a,b,c,d on physical pages 4,5,6,7
 - Per-page Translation table should record all mappings (2000->4,..., 2003->7)
 - Block translation table should only include 1 entry:

Table: 500 → 4		Memory											
Block:	0				1				2				Flash Chip
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 (cont.)

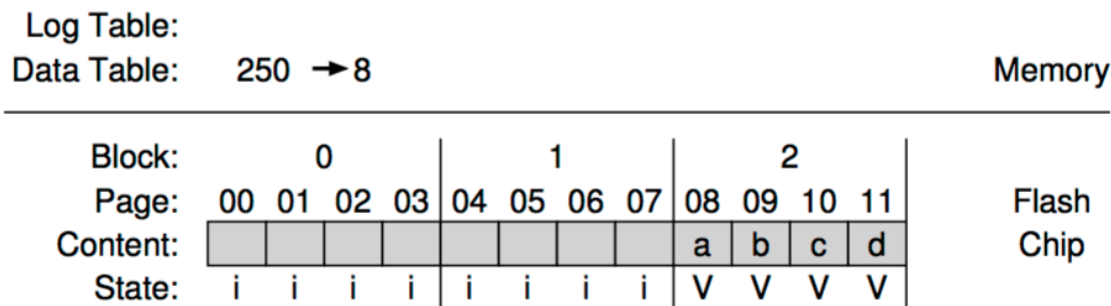
- ▣ Reading is easy
- ▣ Writing not so easy (unmodified data should be replicated in the new block)

Table: 500 → 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	E	E	E	E	V	V	V	V

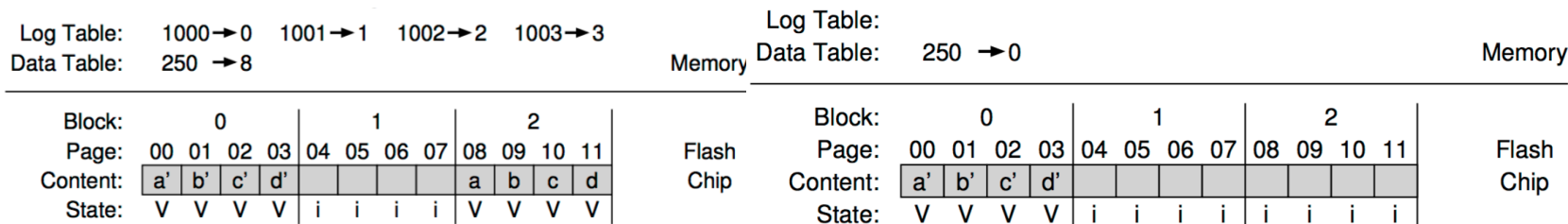
Flash Chip

Hybrid approaches

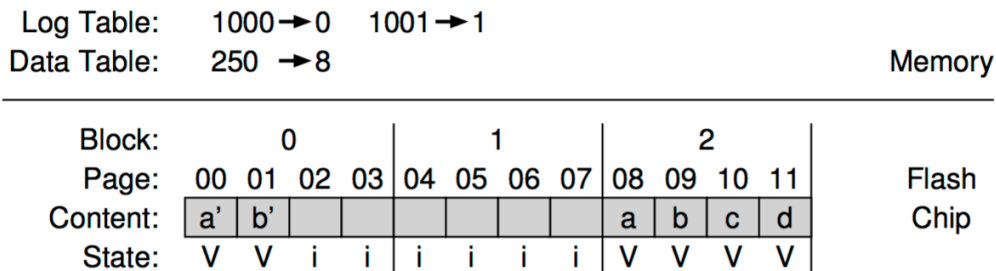
- Keep translation to partially modified blocks in the **Log Blocks**



- Client writes a', b', c', d'



- Client writes a', b'



Wear levelling

- Although LFS does a good job levelling the writes over flash cells, some times a block is never overwritten: the cells does not recibe a fair share of "load"
- FTL should periodically identify such blocks, a write them somewhere else
 - ◆ The cells are available for other write
- Increases the write amplification
- Bigger disks -> less load per cell -> less I/O effect of wear levelling

Performance SSD vs HDD

▣ Sequential accesses vs Random Accesses

Device	Random		Sequential	
	Reads (MB/s)	Writes (MB/s)	Reads (MB/s)	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

Cost SSD vs HDD

- ▣ SSD are .20-.50 cents/GB
- ▣ HDD



- Disclaimer: This lecture slide set was initially developed for Operating System course in Computer Science Dept. at Hanyang University. This lecture slide set is for OSTEP book written by Remzi and Andrea at University of Wisconsin.