SSDs

Why care?

HDD

- Require seek, rotate, transfer on each I/O
- Not parallel (one head)
- Brittle (moving parts)
- Slow (mechanical)
- Poor random I/O (10s of ms)

SSD

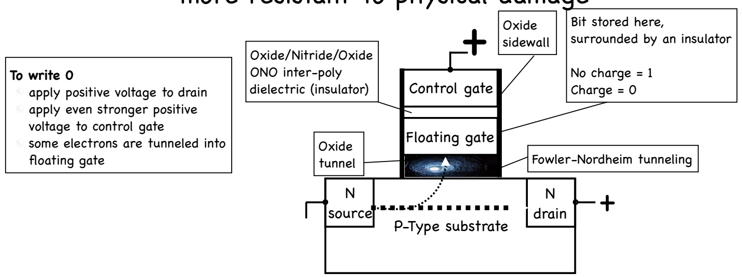
- No seeks
- Parallel
- No moving parts
- Random reads take 10s of μs
- Wears out!

The Cell

- Single-level cells
  - faster, more lasting (50K to 100K program/erase cycles*), more stable
  - 0 means charge; 1 means no charge
- Multi-level cells
  - can store 2, 3, even 4 bits
  - cheaper to manufacture
  - wear out faster (1k to 10K program/erase cycles)
  - more fragile (stored value can be disturbed by accesses to nearby cells)

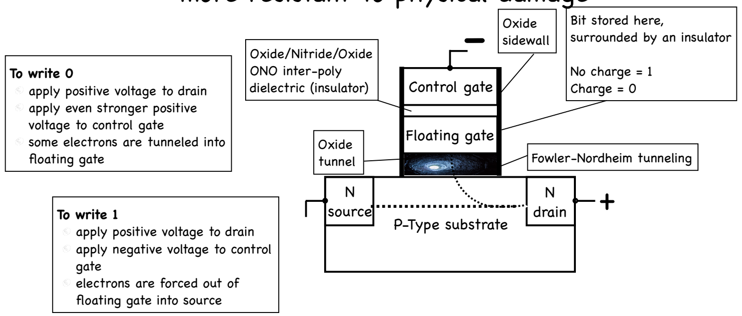
Flash Storage

- No moving parts
  - better random access performance
  - less power
  - more resistant to physical damage



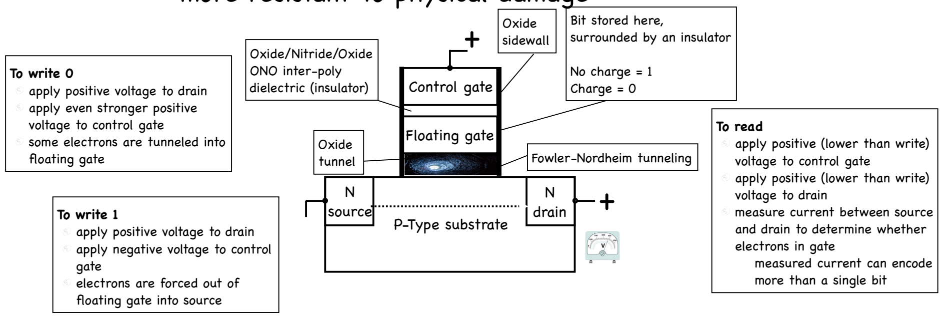
Flash Storage

- No moving parts
  - better random access performance
  - less power
  - more resistant to physical damage



Flash Storage

- No moving parts
  - better random access performance
  - less power
  - more resistant to physical damage



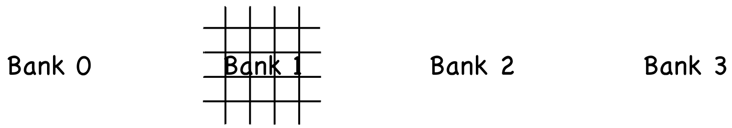
The SSD Storage Hierarchy

Flash Chip	Plane/Bank	Block	Page	Cell
Several banks that can be accessed in parallel	Many blocks (Several Ks)	64 to 256 pages	2 to 8 KB	1 to 4 bits

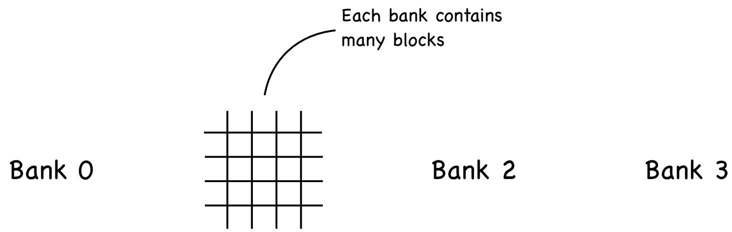
Basic Flash Operations

- Read (a page)
  - 10s of μs, independent of the previously read page
- Erase (a block)
  - sets the entire block (with all its pages) to 1
  - very coarse way to write 1s...
  - 1.5 to 2 ms (on a fast SLC)
- Program (a page)
  - can change some of the bit in a page of an erased block to 0
  - 100s of μs
  - changing a 0 bit back to 1 requires erasing the entire block!

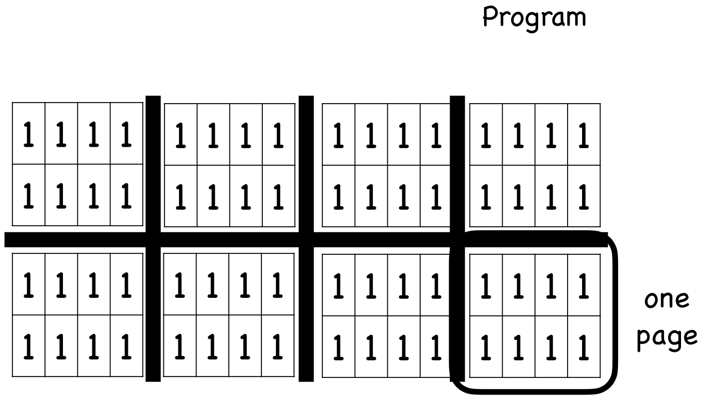
Banks



Banks

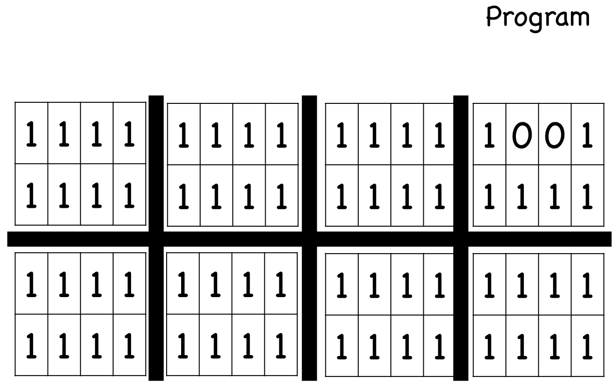


Block



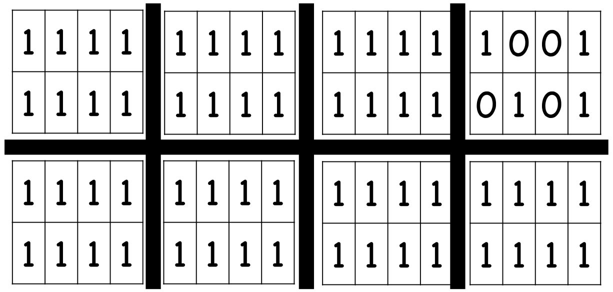
After an Erase, all cells are discharged (i.e., store 1s)

Block



Block

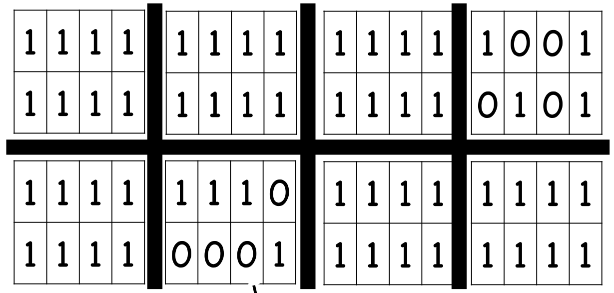
Program



Program

Block

Erase (!)

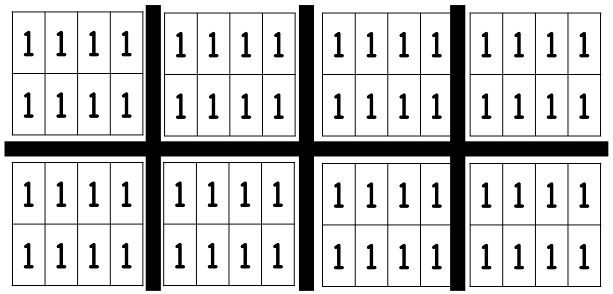


If now we want to set this bit to 1, we need to erase the entire block!

Modified pages must be copied elsewhere, or lost!

Block

Erase



Wear Out

Every erase/program cycle adds some charge to a block; over time, hard to distinguish 1 from 0!

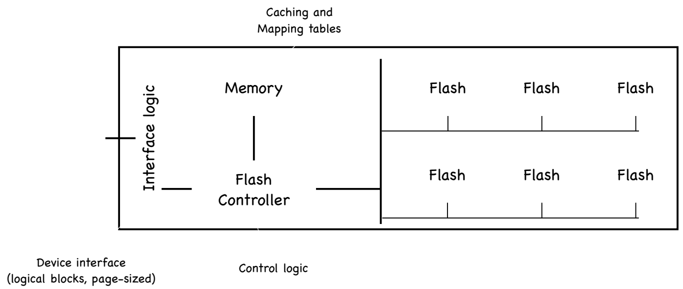
APIs

	HDD	Flash
read	read sector	read page
write	write sector	program page (0's) erase block (1's)

Performance

	HDD	Flash
Throughput	≈ 130MB/s (sequential)	≈200MB/s
Latency	≈ 10ms	read 25μs program 200-300μs erase 1.5-2 ms

From Flash to SSD



- Flash Translation Layer
  - maps read/write operations on logical blocks into read, erase, and program operations
  - tries to minimize
    - write amplification: $\left[\frac{\text{write traffic (bytes) to flash chips}}{\text{write traffic (bytes) to SSD}}\right]$
    - wear out: practice wear leveling
    - disturbance: write pages in a block in order, low to high

FTL through Direct Mapping

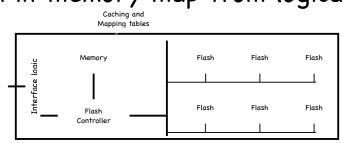
- Just map logical disk block to physical page
  - reads are fine
  - write to logical block involves
    - reading the (physical) block where physical page lives
    - erasing the block
    - programming old pages as well as new page
- Severe write amplification
  - writes are slow!
- Poor wear leveling
  - page corresponding to "hot" logical block experiences disproportionate number of erase/program cycles

FTL through Direct Mapping

- Just map logical disk block to physical page
  - reads are fine
  - write to logical block involves
    - reading the (physical) block where physical page lives
    - erasing the block
    - programming old pages as well as new page
- Severe write amplification
  - writes are slow!
- Poor wear leveling
  - page corresponding to "hot" logical block experiences disproportionate number of erase/program cycles

Log Structured FTL

- Think of flash storage as implementing a log
- On a write, program next available page of physical block being currently written
  - i.e., "append" the write to your log
- On a read, find in the log the page storing the logical block
  - don't want to scan the whole log...
  - keep an in-memory map from logical blocks to pages!



Example

- SSD's clients read/write 4KB logical blocks
- Many physical blocks; each holds 4 pages, each 4KB
  - A logical block maps to a physical page

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content													Flash Chip
State	i	i	i	i	i	i	i	i	i	i	i	i	

- Client operations

Write (a1, 100)

1) Erase(00)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical blocks; each holds 4 pages, each 4KB
  - A logical block maps to a physical page

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content													Flash Chip
State	E	E	E	E	i	i	i	i	i	i	i	i	

- Client operations

Write (a1, 100)

2) Program(00)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical blocks; each holds 4 pages, each 4KB
  - A logical block maps to a physical page

Table 100 → 00 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1												Flash Chip
State	V	E	E	E	i	i	i	i	i	i	i	i	

- Client operations

Write (a1, 100)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical blocks; each holds 4 pages, each 4KB
  - A logical block maps to a physical page

Table 100 → 00 Memory

Block	00				01				02				
Page	00	01	02	03	04	05	06	07	08	09	10	11	
Content	a1												Flash Chip
State	V	E	E	E	i	i	i	i	i	i	i	i	

- Client operations

Write (a1, 100)
Write (a2, 101)

3) Program(01)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical blocks; each holds 4 pages, each 4KB
  - A logical block maps to a physical page

Table 100 → 00 101 → 01 Memory

Block	00				01				02			
Page	00	01	02	03	04	05	06	07	08	09	10	11
Content	a1	a2										
State	V	V	E	E	i	i	i	i	i	i	i	i

Flash Chip

- Client operations

Write (a1, 100)
Write (a2, 101)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical blocks; each holds 4 pages, each 4KB
  - A logical block maps to a physical page

Table 100 → 00 101 → 01 2000 → 02 2001 → 03 Memory

Block	00				01				02			
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

- Client operations

Write (a1, 100)
Write (a2, 101)
Write (b1, 2000)
Write (b2, 2001)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical blocks; each holds 4 pages, each 4KB
  - A logical block maps to a physical page

Table 100 → 00 101 → 01 2000 → 02 2001 → 03 Memory

Block	00				01				02			
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

- Client operations

Write (c1, 100)

Erase(01)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical blocks; each holds 4 pages, each 4KB
  - A logical block maps to a physical page

Table 100 → 00 101 → 01 2000 → 02 2001 → 03 Memory

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

Flash Chip

- Client operations

Write (c1, 100)

Program(04)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 00 101 → 01 2000 → 02 2001 → 03 Memory

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

- Client operations

Write (c1, 100)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 04 101 → 01 2000 → 02 2001 → 03 Memory

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

- Client operations

Write (c1, 100)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 04 101 → 05 2000 → 02 2001 → 03 Memory

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

- Client operations

Write (c1, 100)
Write (c2, 101)
Write (b1, 2000)
Write (b2, 2001)

Example

- SSD's clients read/write 4KB logical blocks
- Many physical blocks; each holds 4 pages, each 4KB

A logical block maps to a physical page

Table 100 → 04 101 → 05 2000 → 02 2001 → 03 Memory

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

- Client operations

Write (c1, 100)
Write (c2, 101)

Garbage Collection

- Reclaim dead blocks
  - find a block with garbage pages
  - copy elsewhere the block's live pages
    - store somewhere in block mapping from page to logical block (the "reverse mapping")
    - use Mapping Table to distinguish live pages from dead
  - make block available for writing again

Table: 100 → 04, 101 → 05, 2000 → 02, 2001 → 03 (Memory)

Block	00				01				02				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	

Table: 100 → 04, 101 → 05, 2000 → 06, 2001 → 07 (Memory)

Block	00				01				02				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	

Shrinking the Mapping Table

- Per-page mapping is memory hungry
  - 1TB SSD, 4KB pages, 4B MTEs: 1GB Mapping Table!

Shrinking the Mapping Table

- Per-page mapping is memory hungry
  - 1TB SSD, 4KB pages, 4B MTEs: 1GB Mapping Table!
- Per-block mapping?
  - think of logical block address as [chunk number (size of physical block)] … [page offset]
  - decreases MT size by factor $\left[\frac{\text{block size}}{\text{page size}}\right]$

Table: 2000 → 04, 2001 → 05, 2002 → 06, 2003 → 07 (Memory)

Block	00				01				02				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	

maps virtual chunk number to physical block

Table: 500 → 04 (Memory)

Block	00				01				02				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	

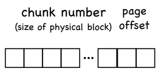
Shrinking the Mapping Table

- Per-page mapping is memory hungry
  - 1TB SSD, 4KB pages, 4B MTEs: 1GB Mapping Table!
- Per-block mapping?
  - think of logical block address as [chunk number (size of physical block)] … [page offset]
  - decreases MT size by factor $\left[\frac{\text{block size}}{\text{page size}}\right]$
    - reading is easy

Table: 500 → 04 (Memory)

Block	00				01				02				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	

Shrinking the Mapping Table

- Per-page mapping is memory hungry
  - 1TB SSD, 4KB pages, 4B MTEs: 1GB Mapping Table!
- Per-block mapping?
  - think of logical block address as 

  - decreases MT size by factor $\left[\frac{\text{block size}}{\text{page size}}\right]$
    - reading is easy
    - but writes smaller than a block require a erase/program cycle!

Hybrid Mapping

- Log Table: a small number of per-page mappings
- Data Table: a large number of per-block mappings
- On read
  - search for block in Log Table; then go to Data Table
- Periodically, "do the switch"
  - turn Log Table blocks with freshest values into Data Table blocks
  - turn Data Table blocks with dead values into Log Blocks
- For wear leveling, periodically read and copy elsewhere long-lived, live data

Caching

- Keep page-mapped FTL, but only keep in memory the active part of the Mapping Table
  - same idea as demand paging
- On a miss, must perform another flash read
  - to bring in the mapping
- If cache is full, must evict a mapping
  - if mapping not on flash yet, need an additional write!

Performance

- Huge difference between SSD and HDD for random I/O
- Not so much for sequential I/O
- On SSDs
  - sequential still better than random
    - FS design tradeoffs for HDD still apply
  - sequential reads perform better than writes
    - sometimes you have to erase
  - random writes perform much better than random reads
    - log transform random into sequential

	Random		Sequential	
Device	Reads (MB/s)	Writes (MB/s)	Reads (MB/s)	Writes (MB/s)
Samsung 840Pro 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