Flash-based SSDs

CMPU 334 – Operating Systems
Jason Waterman

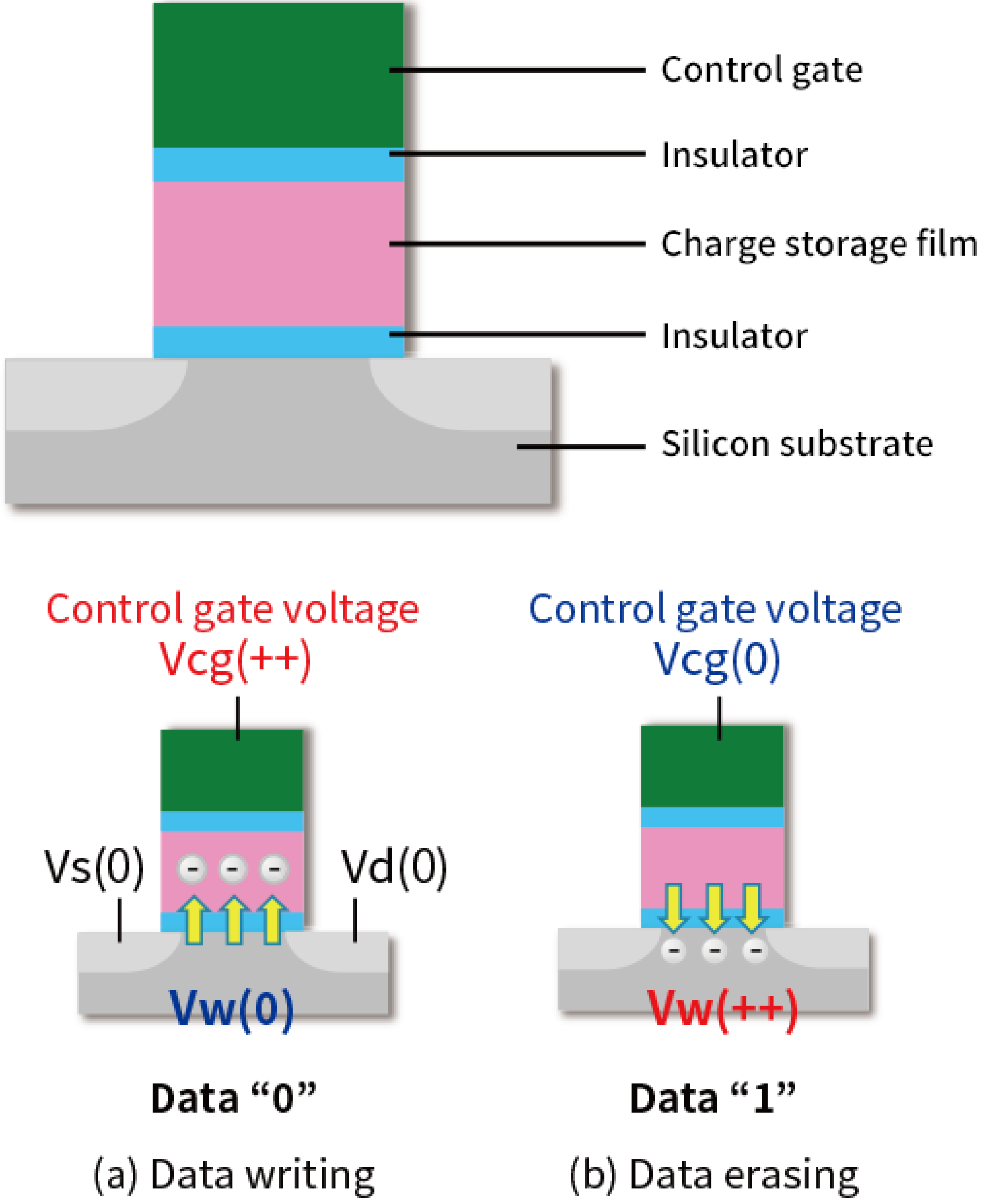
New form of persistent storage

- Solid-state storage
  - No mechanical or moving parts
  - Built out of transistors
  - Retains information despite power loss

- NAND-based flash
  - Created in the 1980s
  - Before writing a flash page (small chunk of data):
    - First must erase the flash block (large chunk of data) where the page lives
    - This takes a relatively long time
  - Writing a page too often will cause it to wear out
    - After about 100,000 writes to a page, it is no longer usable

Storing a single bit

- Single-level cell (SLC) flash
  - Single bit stored within a transistor
  - Floating gate stores charge
  - Best performing, more expensive
- Multi-level cell (MLC) flash
  - Two bits are encoded into 4 levels of charge
- Triple-level cell (TLC) flash
  - Encodes 3 bits per cell
  - Cheaper but not as good performance

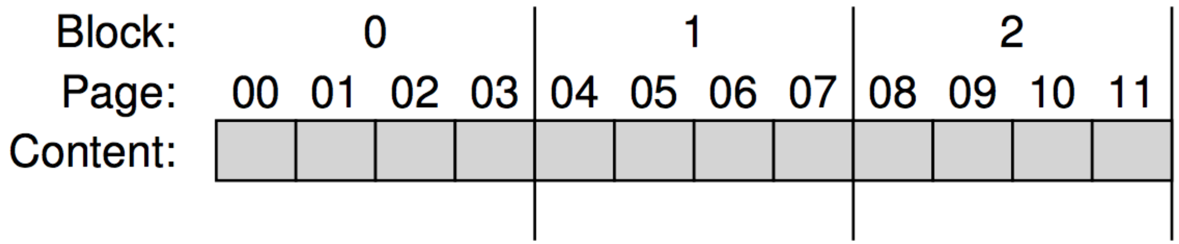


Flash organization

- Flash chips are organized into banks
  - Each bank is accessed as **erase blocks** or **pages**

- Erase blocks
  - Typically, 128 KB or 256 KB
  - Contains many pages
  - When a single page needs to be overwritten, the entire block must be erased first!

- Pages
  - Fundamental unit for a flash
  - Typical size: 4 KB



Flash Operations

- Read a page
  - Can read any page by specifying the read command and a page number
  - Fast operation (10s of microseconds)
  - Regardless of the location of previous request (random access device)
- Erase a block
  - Before writing to a page within a block, you need to erase the entire block
  - Destroys the contents of the block by setting all bits to the value ‘1’
  - Slow operation (a few milliseconds)
- Program a page
  - Writes data to an erased page by changing some of the ones within a page to zeros
  - Less expensive than erasing a block, but more expensive than reading a page
  - 100s of microseconds

Flash example

- Four 8-bit pages within a 4-page block (unrealistically small)

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

- Like to write Page 0 – must move other pages before erasing entire block

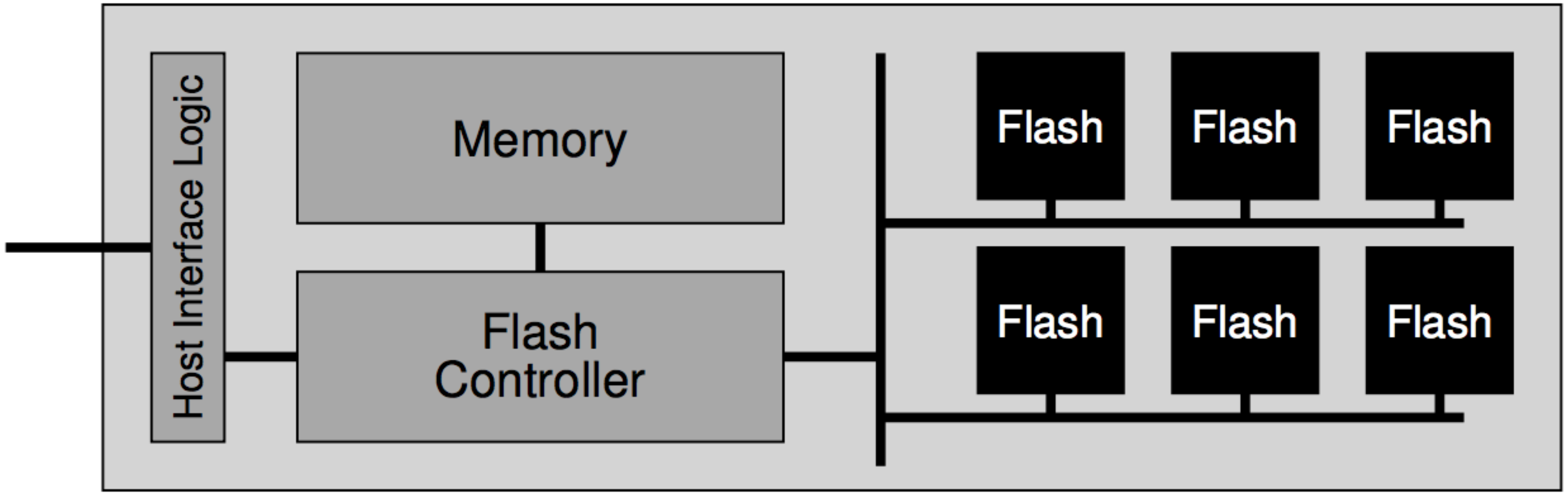
Page 0	Page 1	Page 2	Page 3
11111111	11111111	11111111	11111111
ERASED	ERASED	ERASED	ERASED

- Now can write Page 0

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

Flash translation layer (FTL)

- Turns system reads and writes into internal flash operations
  - Logical blocks → low-level read, erase, and program the physical blocks and pages
- Flash chips (persistent storage)
- SRAM (caching and buffering data, mapping tables)
- Control logic for device operation



Performance goals

- Speed
  - Use multiple flash chips in parallel to obtain higher performance
- Reduce write amplification
  - Write traffic in bytes issued by the FTL divided by write traffic issued to the flash
- Wear leveling
  - Spread out writes across blocks of the flash as evenly as possible
- Program disturbance
  - Writing a page can flip bits of neighboring pages
  - Write pages from low page to high page to minimize this

Log-structured FTL

- For reliability and performance FTLs are log structured
- Given a write to logical block N:
  - Device appends the write to the next free spot in the currently being written block
- To find logical block N:
  - Device keeps a **mapping table** (both in memory and persistent storage)
  - Keeps the physical address of each logical block in the system

Log-structured FTL example

- Write logical block 100

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

- Logical write of 101, 2000, 2001

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

Persisting the FTL mapping

- Map is stored in memory on the device for performance
- How does the mapping survive a power loss?

- Record some mapping information with each page
  - Out-of-band (OOB) area
  - Mapping can be reconstructed from this information
  - Scanning a large SSD to find mappings is slow

- Higher-end devices use logging and checkpointing

Garbage Collection

- Assume blocks 100 and 101 are written 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	

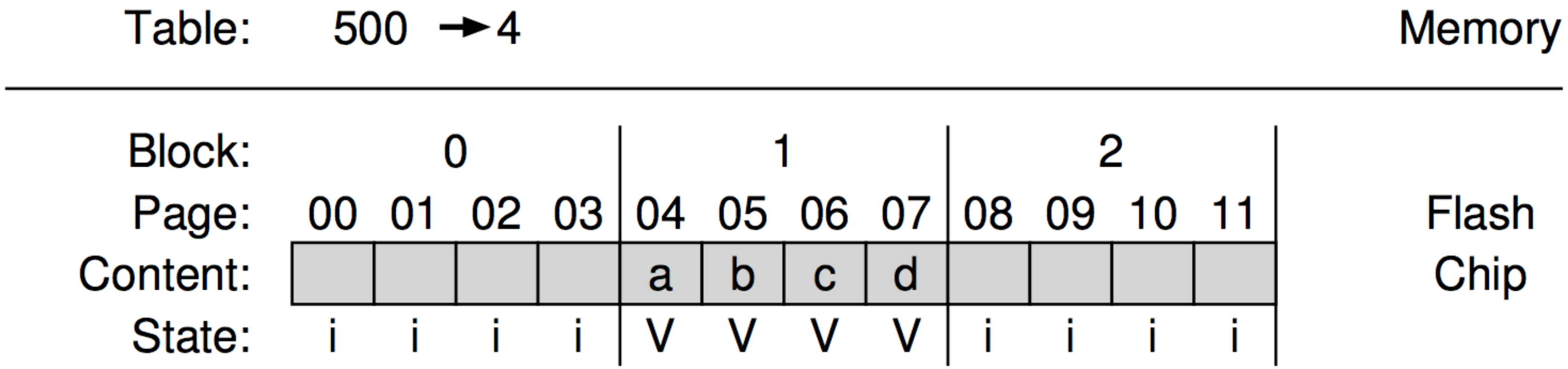
- Garbage collection (reclaiming dead blocks)
  - Find a block with one or more garbage pages
  - Read the live (non-garbage) pages from the block
  - Write out live pages to the log
  - Reclaim block for use in writing

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	

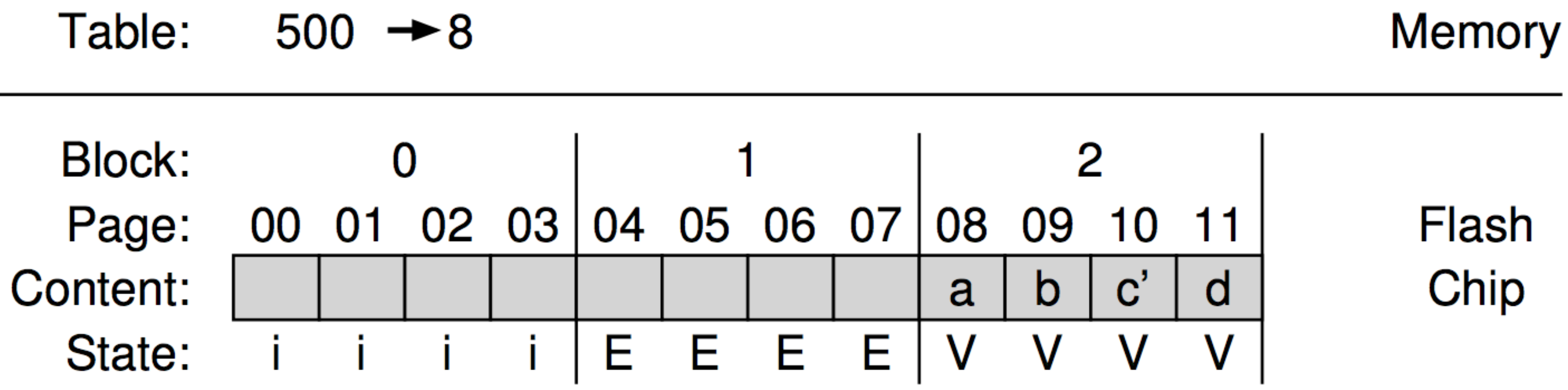
Mapping table size

- 1-TB SSD, 4-KB page size, 4-byte map entry
  - 1 GB of memory needed for just the mappings
  - Page-level FTL mapping is impractical
- Block-Based Mapping
  - One pointer per *block* of the device instead of per page
  - Logical address divided into block sized chunks
    - Logical addresses consists of two portions: chunk number and offset
  - Poor performance for “small writes” (less than a block) – must copy all pages in the block
- Example: logical blocks 2000, 2001, 2002, and 2003 all have the same chunk number (500) and have offsets (0, 1, 2, 3)



Block-based Mapping Writes

- Writing to logical block 2002 (with contents c’)
  - Read in 2000, 2001, 2003 and write out all four logical blocks in a new location
  - Update mapping table
  - Small writes (less than a physical block) hurt performance
  - Increase write amplification
  - With block sizes of 256KB or larger, small writes can happen often

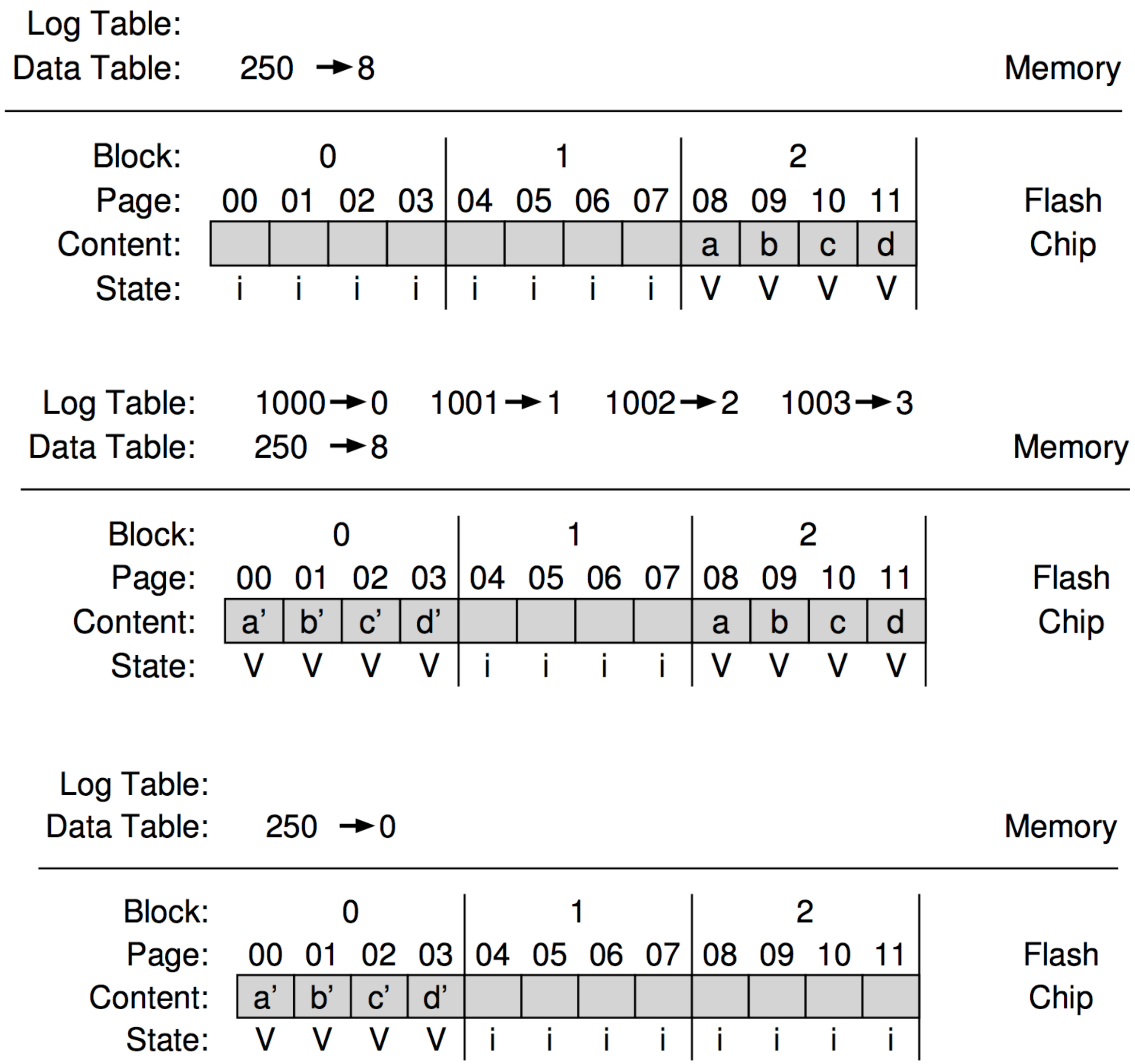


Hybrid Mapping

- FTL keeps a few blocks erased and directs all writes to them
  - Called log blocks
  - Keep per-page mappings for these log blocks
- Keeps two types of of tables in memory
  - Log table (per-page mappings)
  - Data table (per-block mappings)
- When looking for a logical block
  - First look in log table
  - Then check data table
- Must keep number of log blocks small
  - Periodically examine log blocks and switch them to data blocks when possible
  - Done with three main techniques, based on the contents of the block

Switch Merge

- Logical pages 1000, 1001, 1002, and 1003 were written and placed in block 2
- Each of these blocks are overwritten in the exact same order (a’, b’, c’, d’)
- FTL can perform a switch merge
  - Log block 0 becomes the storage location
  - Block 2 is erased and used as a log block
- Best case for hybrid FTL



Partial Merge

- What happens in the case of a partial write?

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	E	E	i	i	i	i	V	V	V	V

Flash Chip

- FTL performs a **partial merge**
  - Logical blocks 1002 and 1003 are read from physical block 2 and are appended to the log (in pages 2 and 3)
  - Then we can do a switch merge like before

Full Merge

- FTL must pull together pages from many other blocks to perform cleaning
- Example: logical blocks 0, 4, 8, and 12 are written to a log block
  - To switch to a block-mapped page the FTL must:
    - Create a data block containing logical blocks 0, 1, 2, and 3
    - Read 1, 2, and 3 from elsewhere and write out 0-4 together
    - Must do the same for logical blocks 4, 8, and 12 as well
  - Then log block can be freed
- Frequent full merges can harm performance and should be avoided when possible

Page Mapping Plus Caching

- Given the complexity of the hybrid approach a simpler solution would be to cache only the active parts of the page mappings in memory
  - Reduces the memory needed
- With a workload that accesses a small set of pages, this approach works well
- With a working set of pages larger than cache memory
  - Each access will require an extra flash read to bring in the missing mapping
  - FTL will have to evict an old mapping
    - If that mapping is dirty (has been changed from the copy in flash) it will have to be written out to flash
- Many workloads will have locality, so caching can reduce memory overheads and keep performance high

Wear leveling

- Multiple erase/program cycles will wear out a flash block
  - Try to spread that work across all the blocks of the device evenly
- Log-structured approach does a good job of spreading out write log
  - Garbage collection helps as well
- What about long-lived data that does not get over-written?
  - Garbage collection will never reclaim the block
- Periodically read all the live data out of those blocks and re-write it elsewhere
  - Helps with wear-leveling
  - Increases write amplification of the SSD
  - Decreases performance

SSD performance and cost

- Performance
  - Great random access compared to HDD

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

- Cost
  - About 10x more expensive than HDD