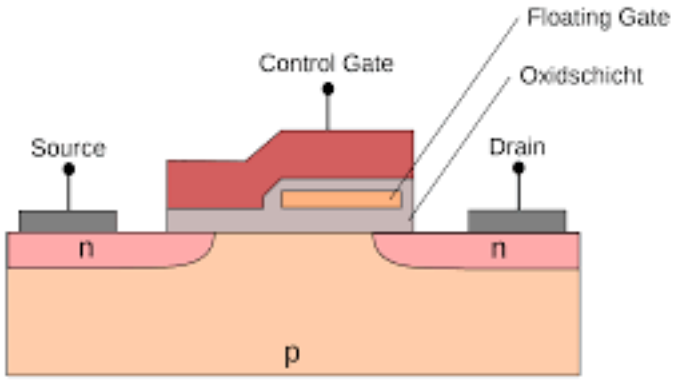
# 44. Flash-Based SSD

## Operating System: Three Easy Pieces

# Non-volatile RAM?

- Solid-State Storage Devices (SSD)
  - Adding persistence to an electron-based device?
  - **Flash** is one of the most successful approaches

- Flash memory
  - Based on hot-electron injection (quantum effect)
  - Most common **NAND**-based Flash (better cost/GB)

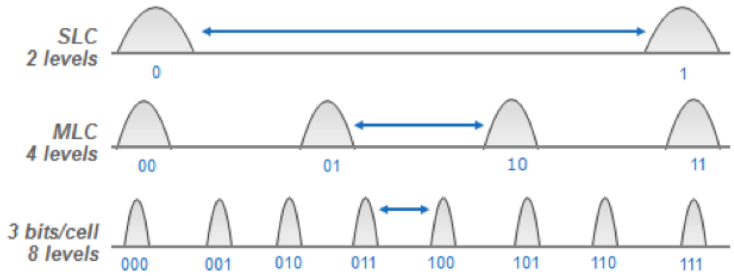


# Some peculiarities about NAND Flash (vs DRAM)

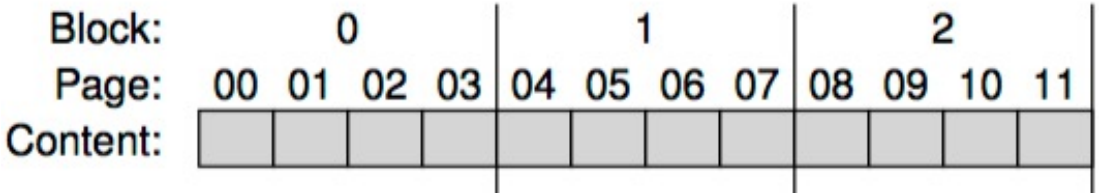
- Information can only be accessed in large "chunks"
- Much more information per mm^2 than DRAM (NAND)
- Electrons are kept trapped in the floating gate (even without power)
  - Non-volatility (~10years)
- Writing is stressful
  - High voltage should be applied (**~12V**) to inject electrons into the floating gate
  - 1000x-10000x write cycles before the cell fails
- Not the only approach (but currently lowest $/GB)
  - Phase change RAM (PCRAM)
  - Filamentary RAM (CBRAM)
  - Memristors (TiOx RAM)
  - Spin-Torque Transfer RAM (STTRAM)

# From bits to Banks/Planes

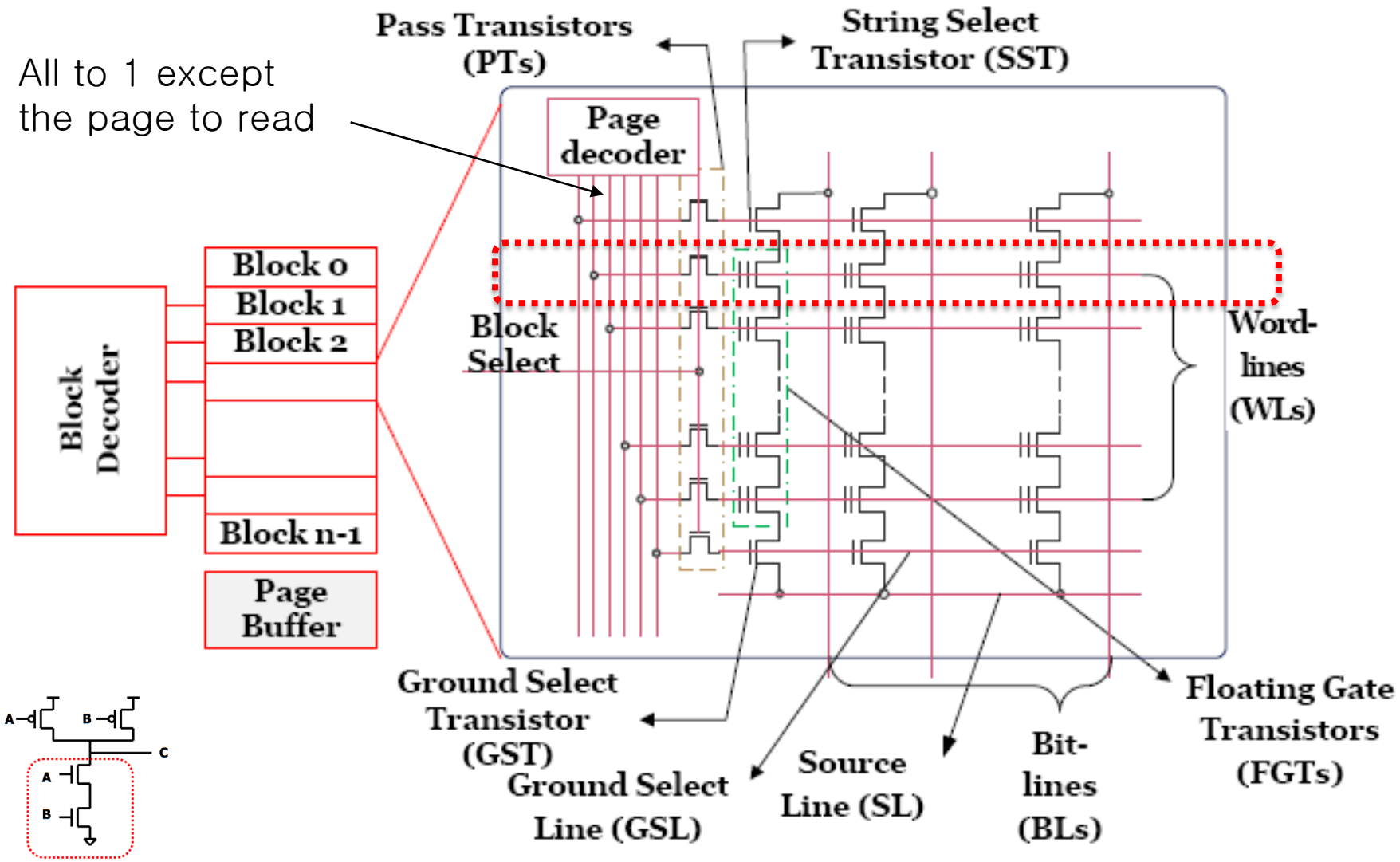
- Each cell can store 1-bit (**SLC**), 2-bits (**MLC**), 3-bits (**TLC**), or 4-bits (**QLC**)



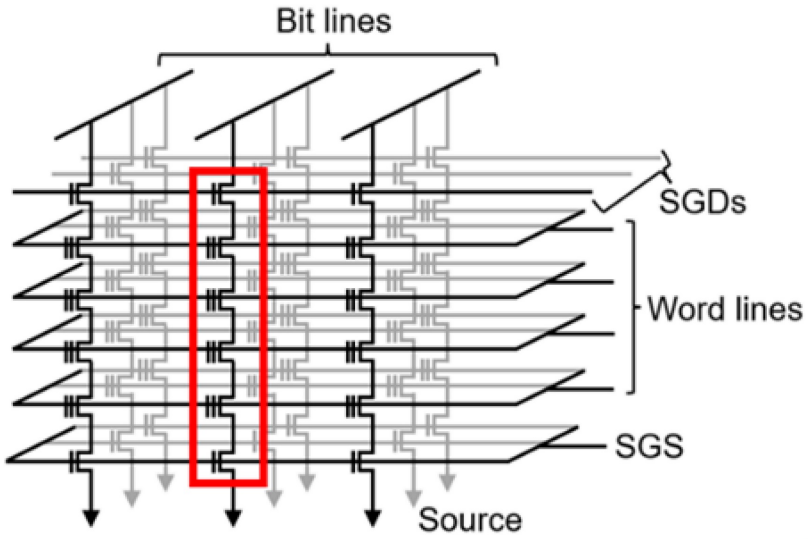

- Flash chips are organized into "planes" (or so-called banks)
  - Banks are divided into **blocks** (or so-called erase blocks):
    - Typically, **64KB-512KB**
  - Each block is divided into **pages**
    - Typically, **4KB**

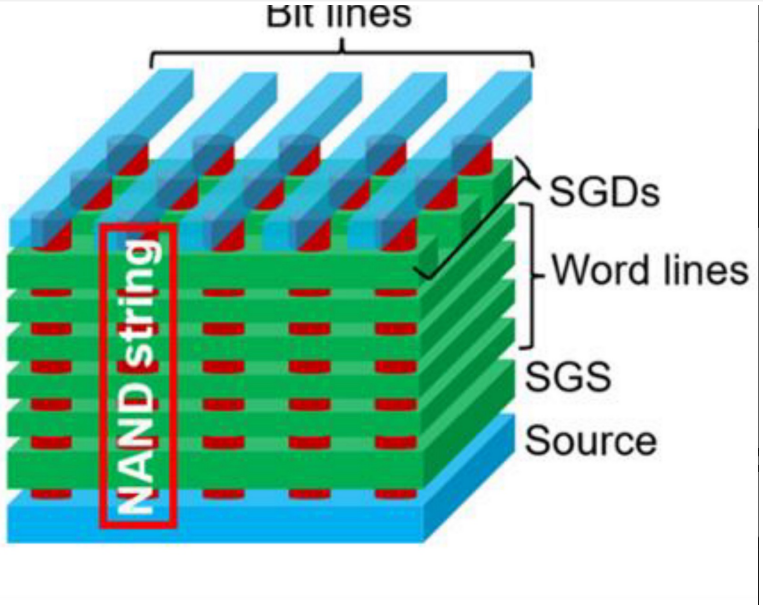


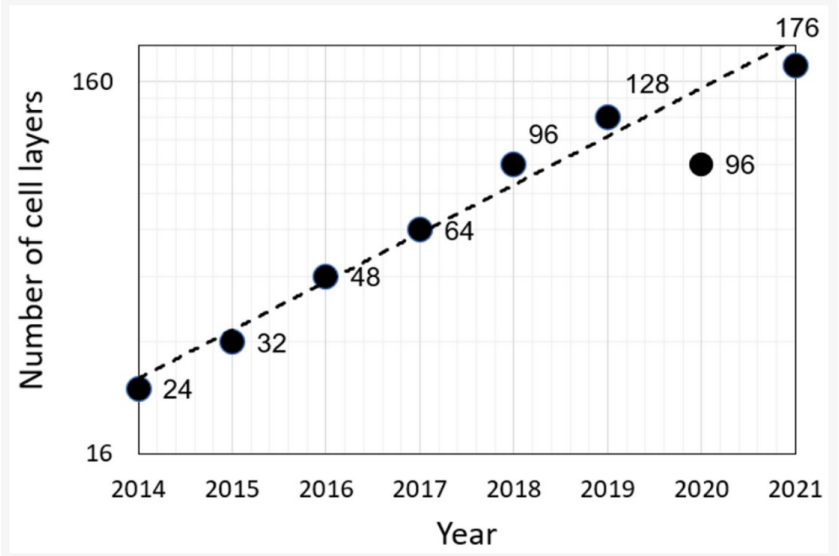
# Aside: More detail about blocks and pages (NAND)

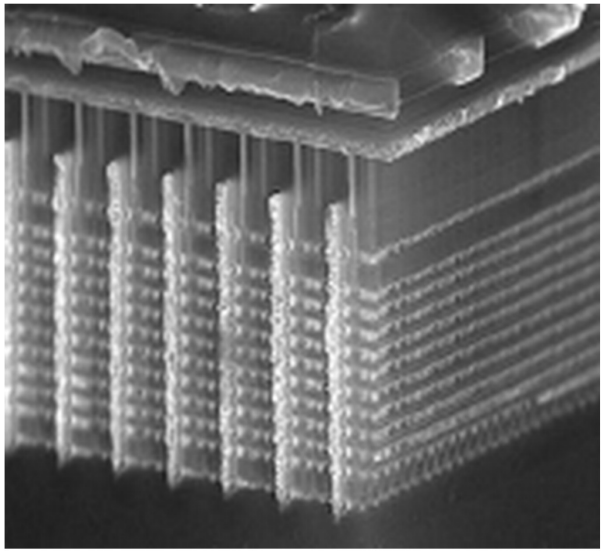


# Aside: 3D NAND Flash

# FLASH Operations

- **Read** (one page)
  - Address the page with a word-line -> Get the content via the bit-lines
  - 10s of microseconds and independent of the address of the last access

- **Erase** (one block)
  - Before to write a page, we must erase all the block where belongs
  - "Save" the pages that we want to keep before erasing
  - Quite expensive process (few milliseconds)
  - We can't erase a page due (too much voltage, cross talk can modify neighbors values)

- **Program** (one page)
  - 100s of microseconds (pushing electrons in the "right" floating gate is "slow")

- Each page has an associate state: (**i**)nvalid, (**e**)rased, (**v**)alid
  - Reads don't change these states
  - Writes must 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. Must 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 writes "**wear out**" the cells!

# Flash Performance and Reliability

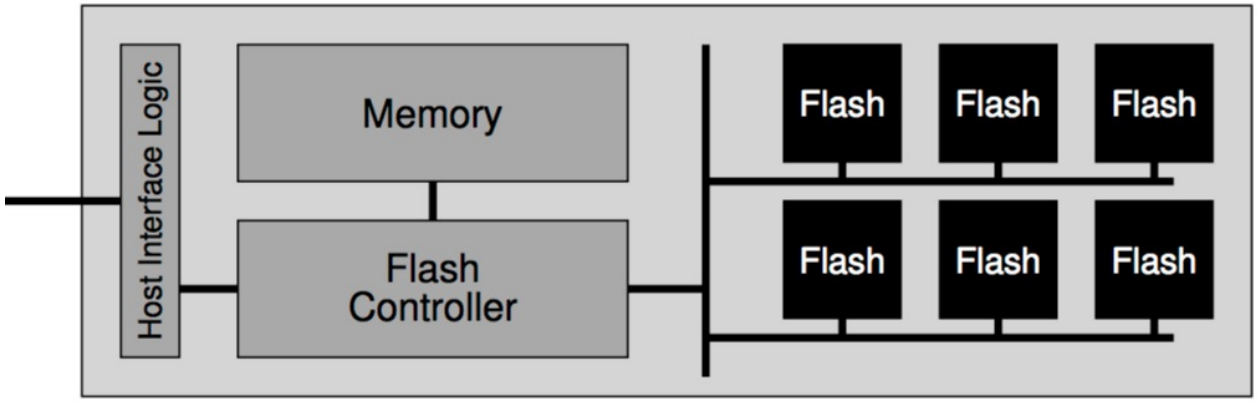
- Not all low-level devices are equal: higher density => lower speed, lower reliability (endurance)

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

- Reliability has two stand-points:
  - Permanent cell wear out
  - Read disturbs or program disturbs
- Higher density and bits per cell means usually means lower reliability
  - 3D NAND (~10^3 or lower ??)
- Really vulnerable to “malicious” attacks (or errors, e.g., Tesla Model S)

# From Flash to SSD

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




  - Usually, some spare capacity and flash
- Flash Translation layer (FTL) is performed by Flash controller
  - Wear balancing 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
  - Minimize **write disturbance**, by writing in order the pages in the block

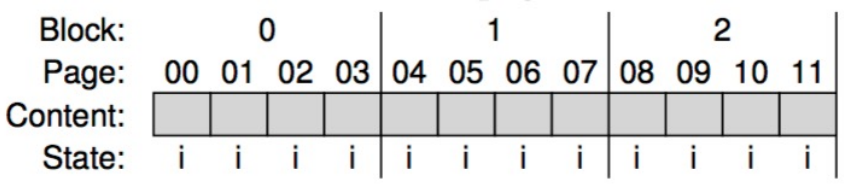
# FTL Organization: A Bad Approach

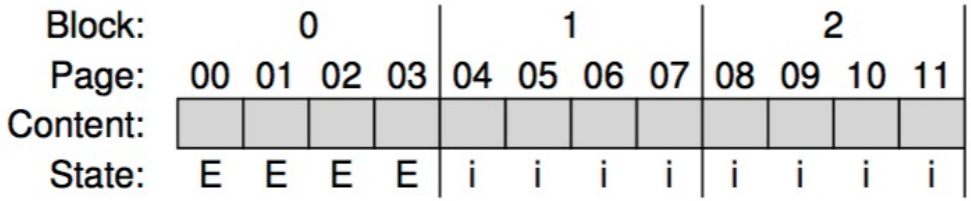
- **Direct mapped**
  - Each logical page N is mapped in a particular physical page N across all the lifetime of the device

- Problems:
  - **Performance**: write amplification
    - The block has to be erased and written in the same chip (sequentially), making it slower than mechanical disks
  - **Reliability**: premature wear-out
    - Writing repeatedly the same data (e.g. Metadata) will physically damage the cells
    - Since the mapping is exposed to the client, malicious programs can physically damage the disk

# A Log-Structured FTL

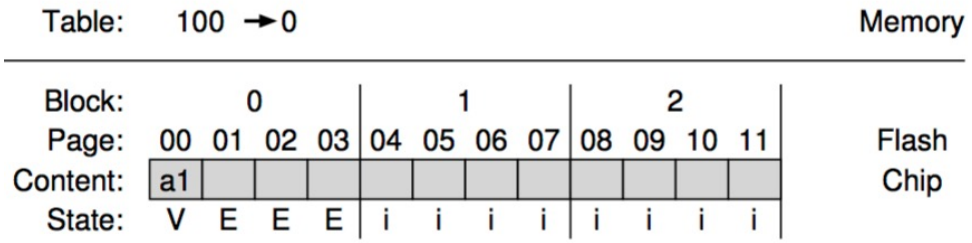
- For different reasons than mechanical disks, the LFS idea is also a good idea here
  - Note that FTL is not a file system!
- Let's use an example:
  - From client perspective **512B sectors**, writes and reads **4-KB chunks**
  - SSD uses (unrealistically small here) **16-KB blocks**, **4-KB pages**
  - Four OPs: Over different logical addresses (Internally the device choses one physical page for each logical address
    - Write (100) with **a1**. Write(101) with **a2**, Write(2000) with **b1**, Write (2001) with **b2**



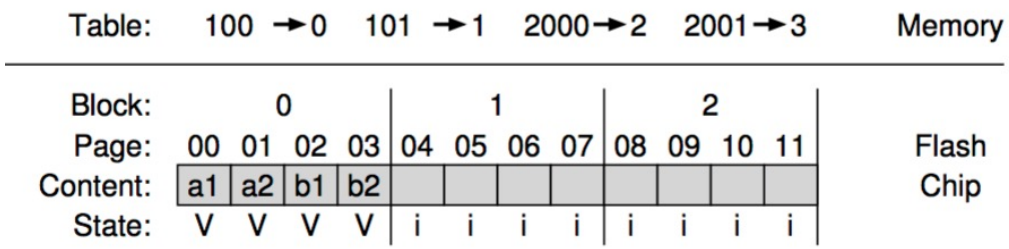



# A Log-Structured FTL (cont.)

- Update translation table



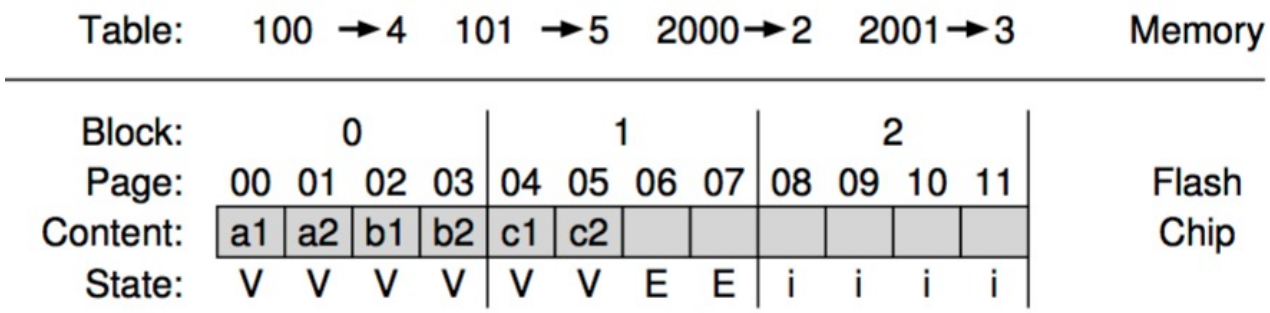

- After all operations done (as a single write in the device)



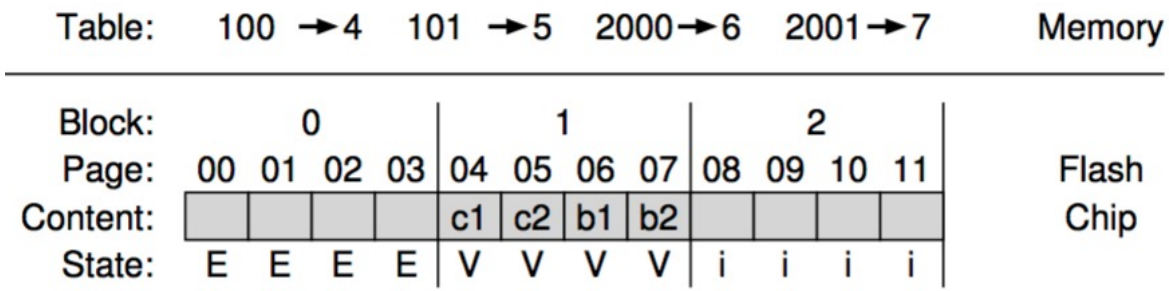

- Successive writes (to the same logical pages) will be spread across the whole device
  - A natural form of wear-levelling

# Garbage collection

- Let's say we want to write c1 and c2 to 100 and 101




- Some pages have 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



# Garbage Collection

- Many reads and writes
  - Can be expensive
- To mitigate the effect, many disk are overprovisioned (10-20%)
  - Allows GC to be to performed when the system is less busy (inthe **background**)
- Adding more capacity increases internal bandwidth (more flash chips), which can be used to perform GC without harming the client side
- If FS is also LFS, garbage collector can be coordinated. FS can help letting know to the FTL that the blocks are not in use (**TRIM** support)

# Aside: FTL Mapping Information Persistence

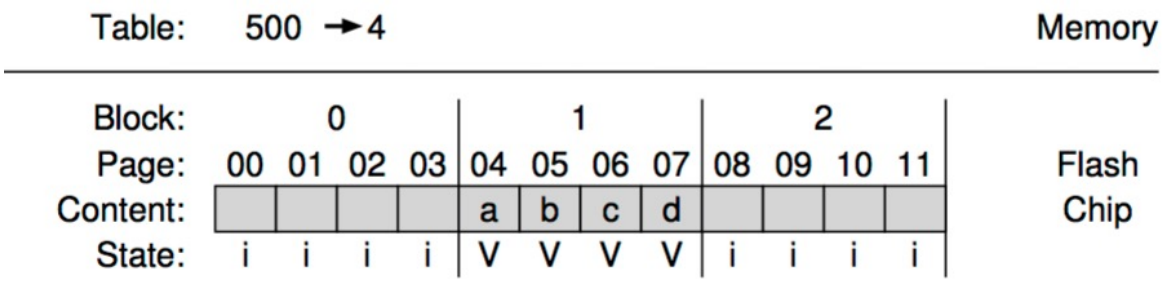
- Translation table **will be lost** in power losses!

- The simplest approach is to record **some** information of the TT in the disk (called **out-of-band** area)

- Can be **extremely slow** to reconstruct map information from OOB on **large disks**.
  - Higher-end disks usually provide additional hardware to do more complex **logging** and **checkpointing**

# Mapping Table Size

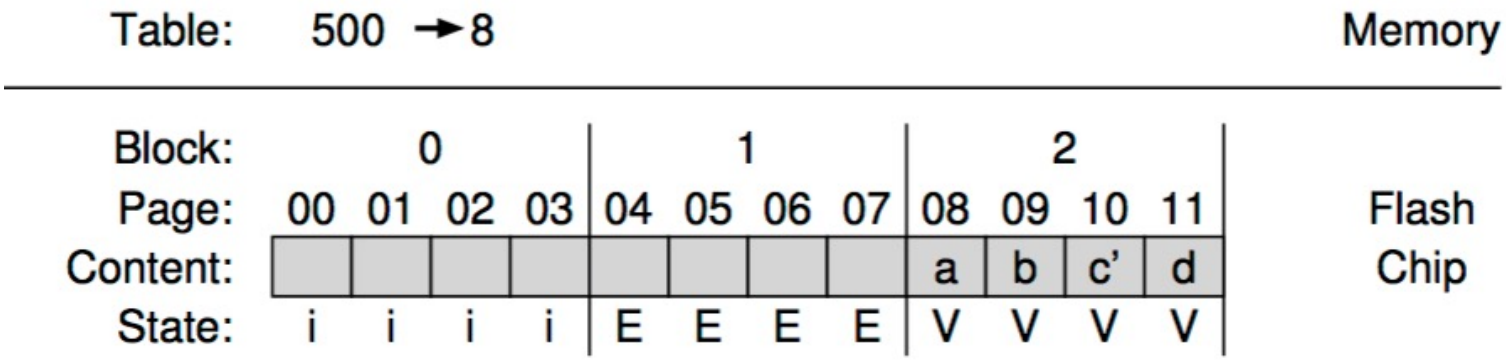
- Block-Based Mapping
  - **Keep only one pointer for the block**
  - Block-level Map is like having larger page sizes (less for VPN) and larger offset
  - Block-level Map don't work very well with LFS-FTL: "**small write**" problem, that increases write amplification
    - Ex) Client wrote logical blocks 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 have only include 1 entry (all logical blocks have 500):



# Block based mapping (cont.)

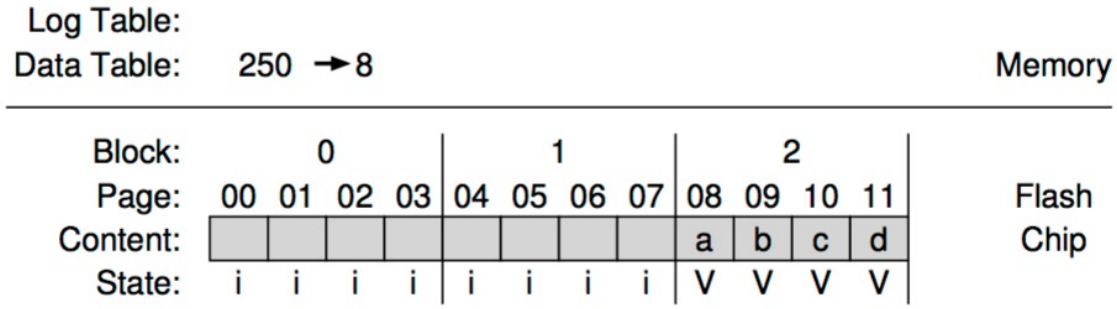
- Reading is easy

- Writing not so easy (unmodified data should be replicated in the new block): after writing c' in 2002 and d' in 2003 (pages << blocks!)



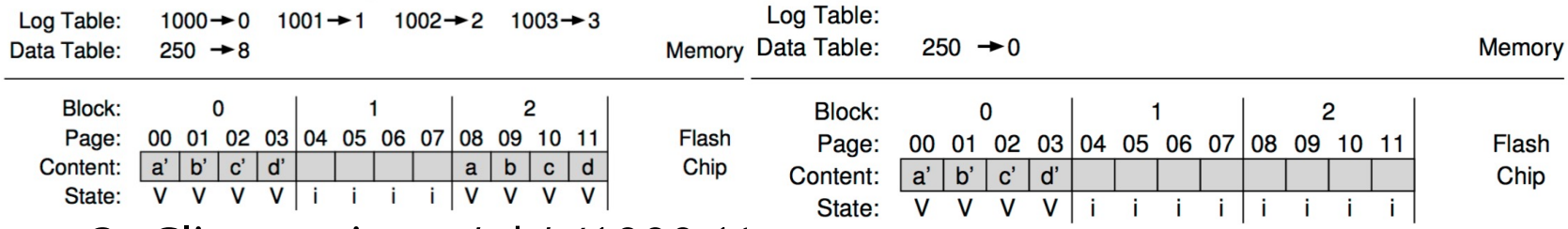
# Hybrid approaches (blocks and pages)

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

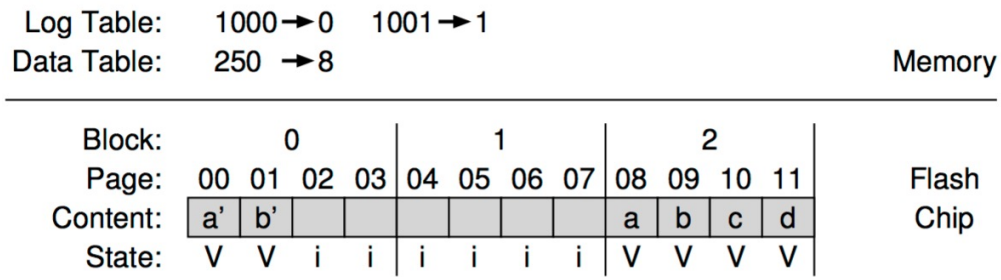



Write
1000→a, 1001→b,
1002→c, 1003→d

- **Case 1:** Client writes a', b', c', d' (1000, …, 1003)




- **Case 2:** Client writes a', b' (1000,1001)



# Wear levelling

- Although LFS does a good gob levelling the writes across flash cells, sometimes a block is never overwritten: the cells **does not geat a fair share of "wear"**

- FTL should periodically identify such blocks, and write them elsewhere
  - The cells are available for another write

- Increases the write amplification

- Bigger disks -> less load per cell -> less I/O effect of wear leveling

- Endurance is expressed in whole drive-writes per Day (DWPD) (for years of warranty) or Terabyte/Petabyte Writes (TBW/PBW)

# Performance SSD vs HDD and Endurance

- 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 |

- Endurance

| Model | DWPD (TPBW) | Cost |
|---|---|---|
| Samsung 870 QVO 1TB (QLC) | 0.14 (260 TBW) | ~$100 |
| Samsung 870 QVO 8TB (QLC) | 0.13 (1.8 PT) | ~$600 |
| Samsung 980ZET 480 GB (SLC) | 8.5 (7.4PB) | ~$1000 |
| Samsung 980ZET 960 GB (SLC) | 10 (17.5PB) | ~$2000 |

# Cost SSD vs HDD

- Non QLC SSD are .20-.50 cents/GB
- QLC SSD are bellow 0.1 cents/GB (and poor endurance <0.1 DWPD)
- HDD

- 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.