POLITECNICO DI MILANO

Computing Infrastructure





SSD - Solid State Disks

The topics of the course: what are we going to see today?

HW Infrastructures:

System-level: Computing Infrastructures and Data Center Architectures, Rack/Structure;

Node-level: Server (computation, HW accelerators), **Storage (Type, technology),** Networking (architecture and technology);

Building-level: Cooling systems, power supply, failure recovery

SW Infrastructures:

Virtualization: Process/System VM, Virtualization Mechanisms (Hypervisor, Para/Full virtualization)

Computing Architectures: Cloud Computing (types, characteristics), Edge/Fog Computing, X-as-a service

Methods:

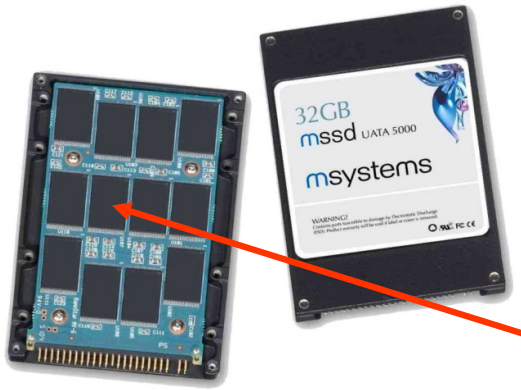
Reliability and availability of datacenters (definition, fundamental laws, RBDs)

Disk performance (Type, Performance, RAID)

Scalability and performance of datacenters (definitions, fundamental laws, queuing network theory)

Overview

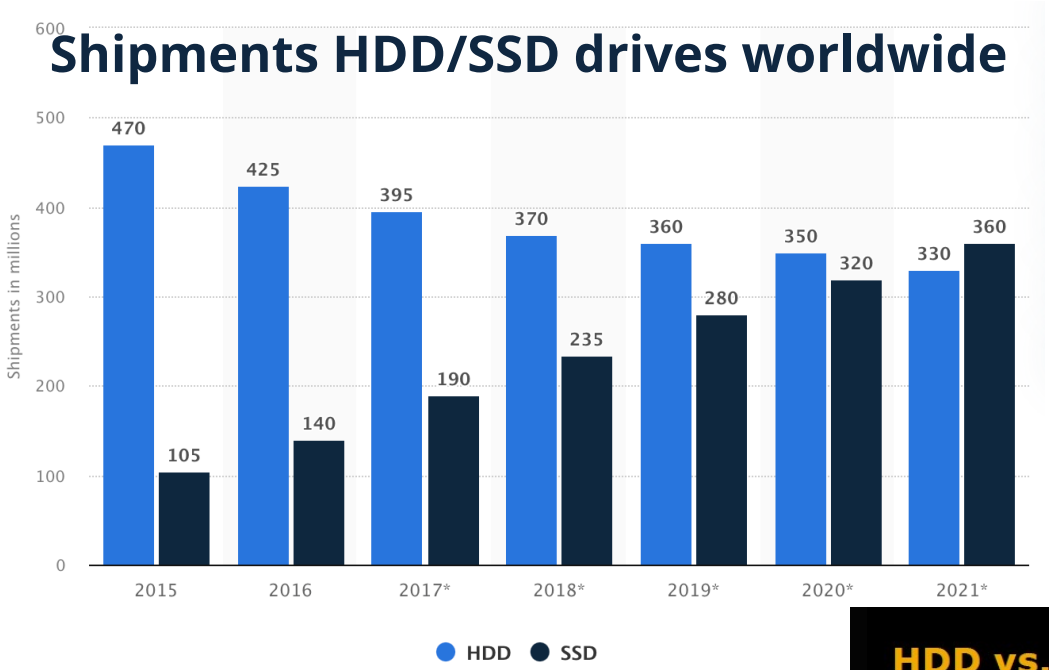
- Solid-state storage device
  - No mechanical or moving parts like HDD
  - Built out of transistors (like memory and processors)
  - Retain information despite power loss unlike typical RAM
  - A controller is included in the device with one or more solid state memory components
  - It uses traditional hard disk drive (HDD) interfaces (protocol and physical connectors) and form factors
  - Higher performances than HDD

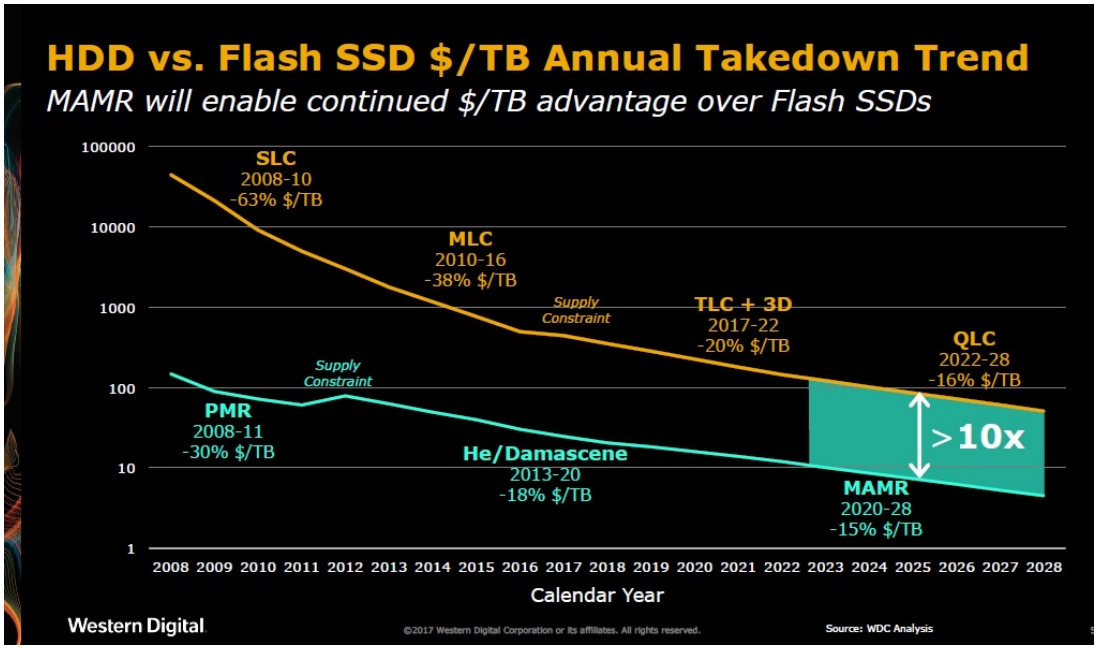


Platters are replaced by NAND-based memories

HDD vs SSD Major trends

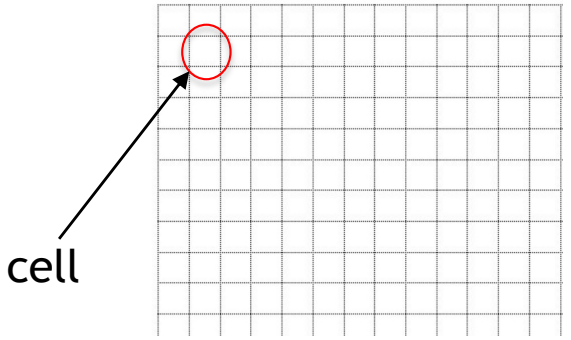
Shipments HDD/SSD drives worldwide





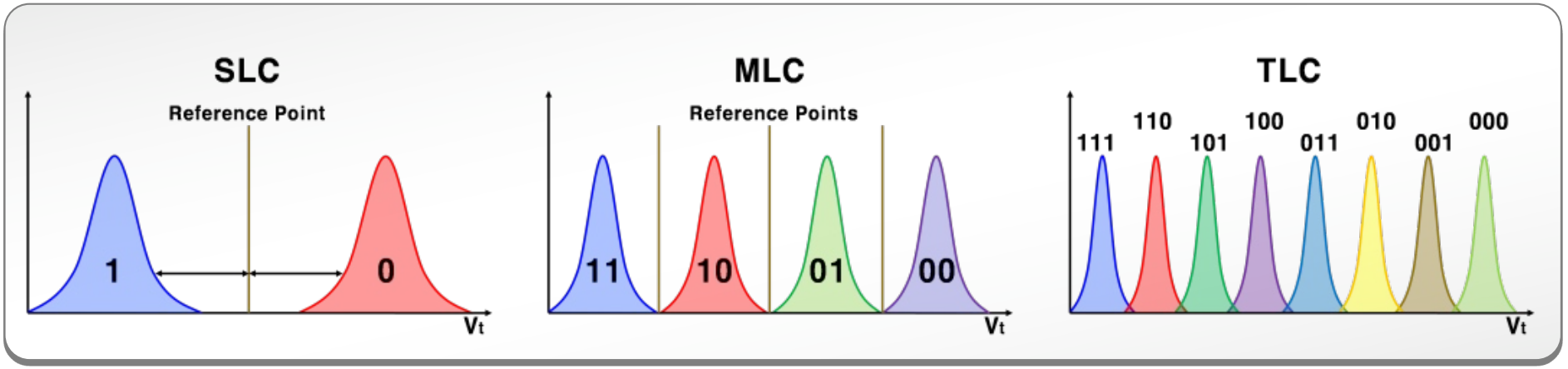
Storing Bit

- Single-level cell (SLC)
  - Single bit per cell
- Multi-level cell (MLC)
  - Two bits per cell
- Triple-level cell (TLC)
  - Three bits per cell
- QLC, PLC...



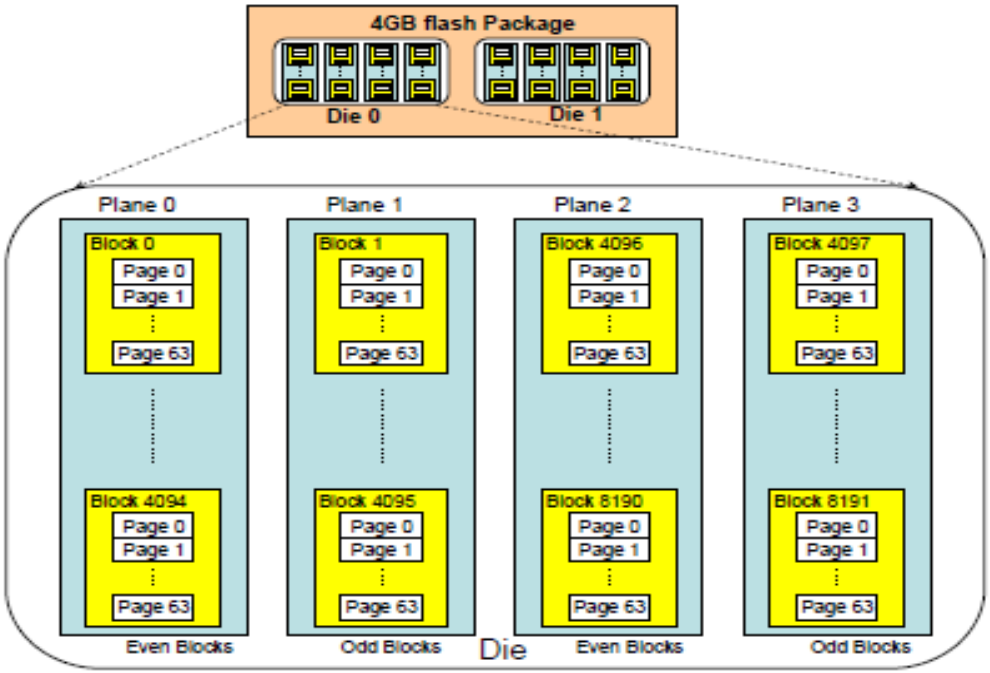
Flash memory

Device	Read (μs)	Program (μs)	Erase (μs)
SLC	25	200-300	1500-2000
MLC	50	600-900	~3000
TLC	~75	~900-1350	~4500



SSD: Internal organization (1)

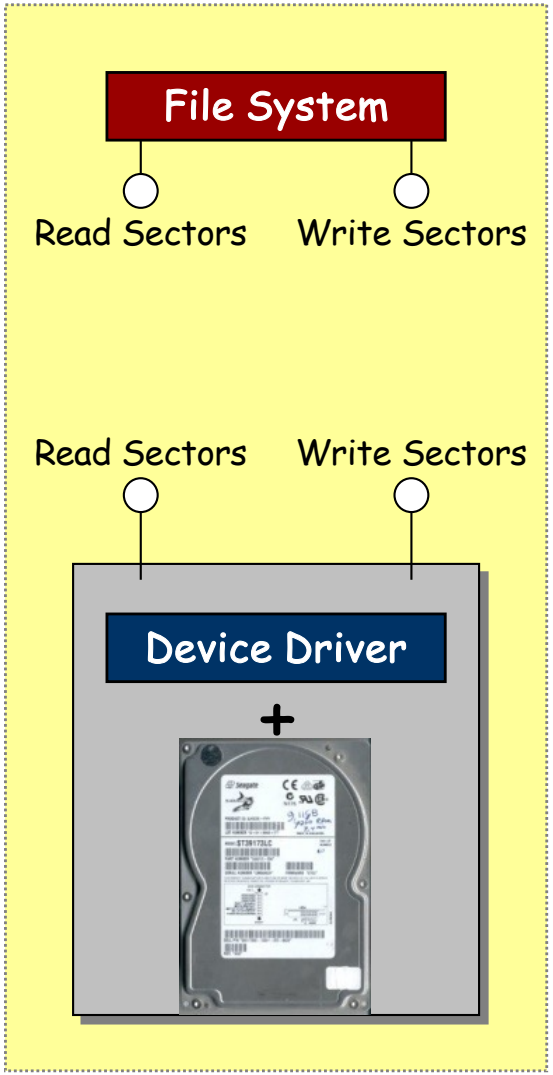
- NAND flash is organized into **Pages** and **Blocks**
- A **page** contains multiple logical block (e.g. 512B-4KB) addresses (LBAs)
- A **block** typically consists of multiple pages (e.g. 64) with a total capacity of around 128-256KB
- **Block/Page** terminolgoy in the SSD context can clash with previous use

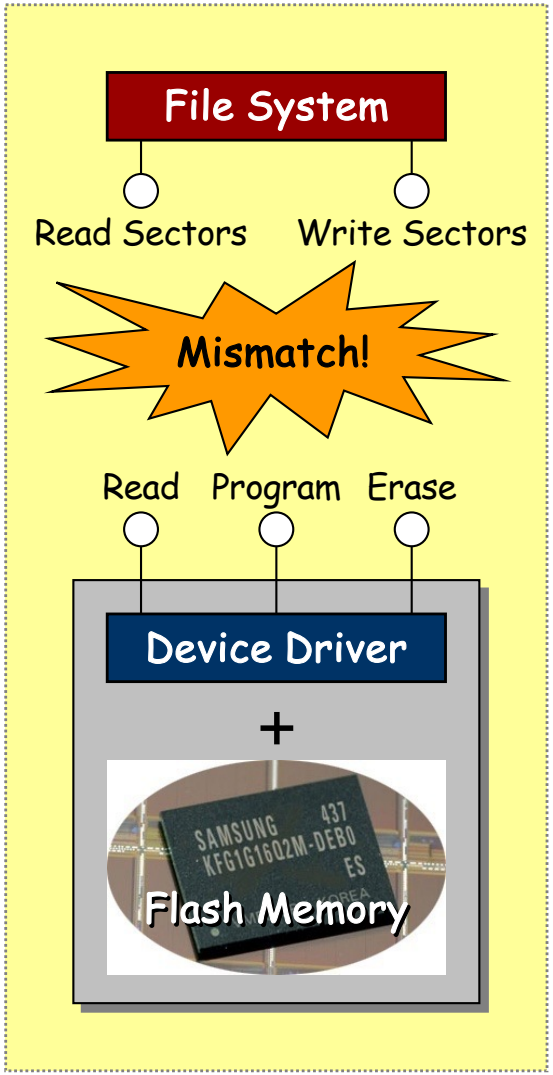


SSD: Internal organization (2)

- **Blocks** (or Erase Block): smallest unit that can be erased
  - It consists of multiple pages and can be cleaned using the **ERASE** command
- **Pages**: smallest unit that can be read/written.
  - It is a sub-unit of an erase block and consists of the number of bytes which can be read/written in a single operations through the ***READ*** or ***PROGRAM*** commands
- Pages can be in *three states*:
  - Empty (or *EREASED*):
    - they do not contain data.
  - Dirty (or *INVALID*):
    - they contain data, but this data is no longer in use (or never used).
  - In use (or *VALID*):
    - The page contains data that can be actually read

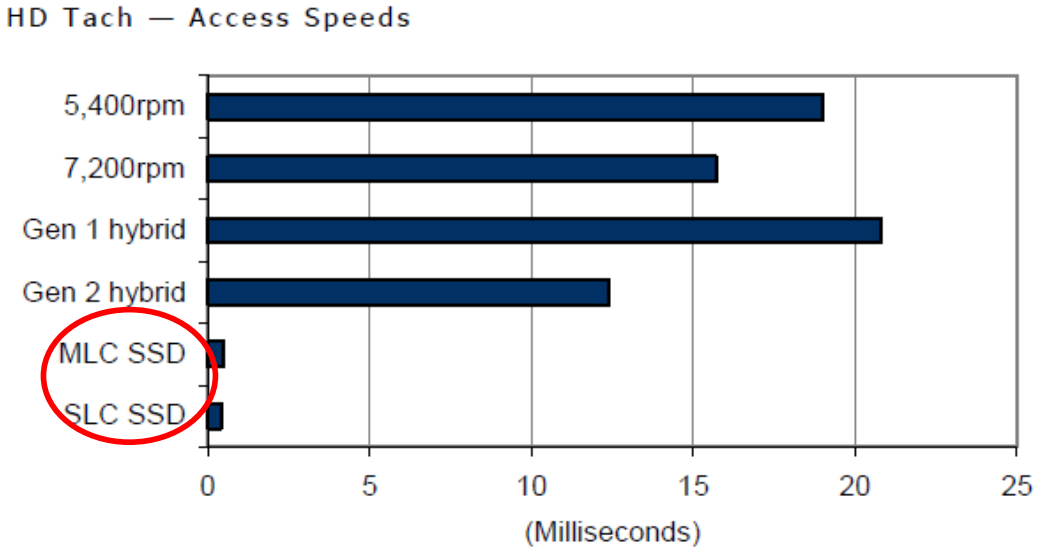
Hard Disk and Flash SSD





This mismatch is one of the cause for the **WRITE AMPLIFICATION** problem!

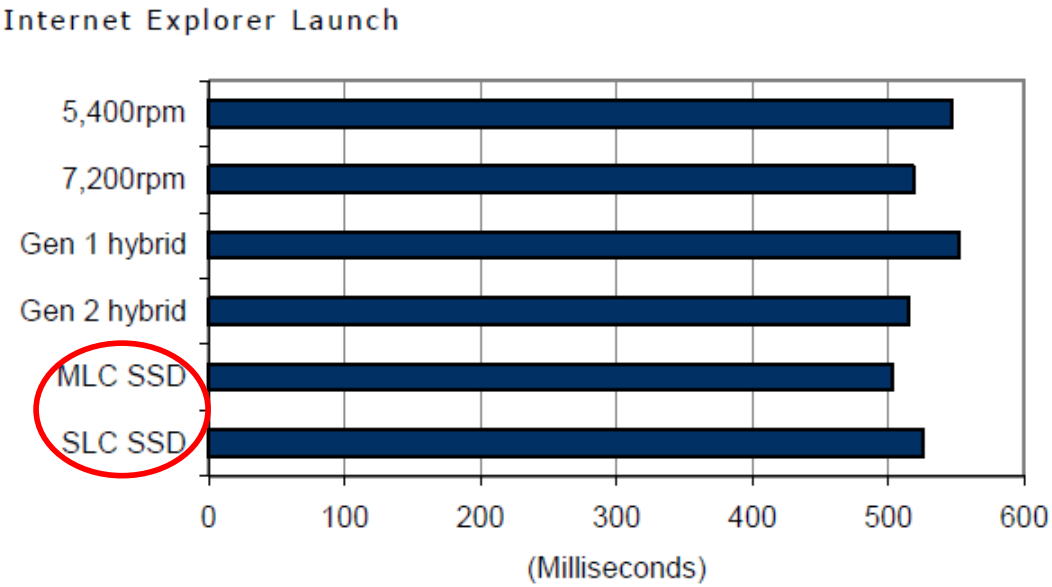
SSD: Performances: from theory...to practice



Access speed
measured at device level in a controlled laboratory environment
HDD vs SSD
~ 15-20x improvement

Same device integrated into a notebook PC with a real application test
~ same performace !!!

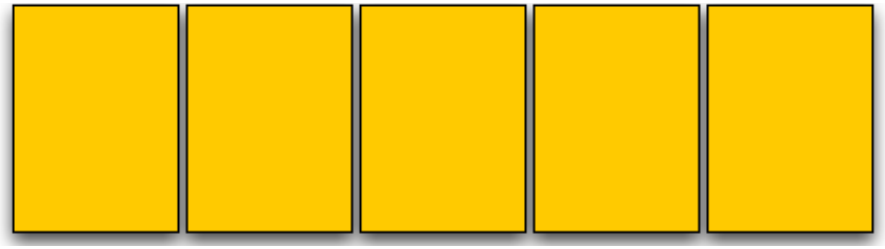
Why such behavior ?



Source: Exposing the Strengths and Weaknesses of HDDs, SSDs, IDC and Hybrids, IDC, 2008

Example:

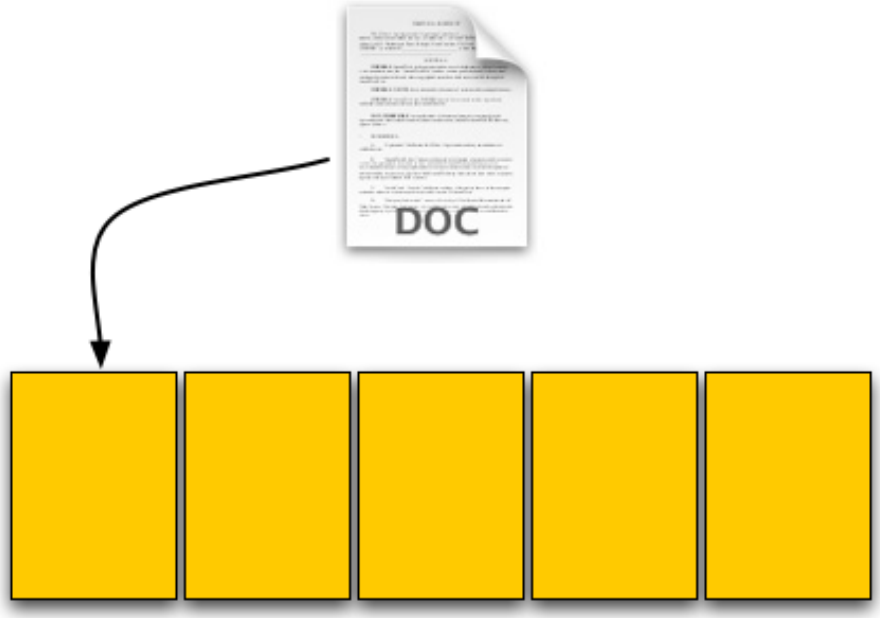
- Hypothetical SSD:
  - Page Size: 4KB
  - Block Size: 5 Pages
  - Drive Size: 1 Block
  - Read Speed: 2 KB/s
  - Write Speed: 1 KB/s



Example:

- Lets write a 4Kb text file to the brand new SSD
- Overall Writing Time:
  - 4 seconds!

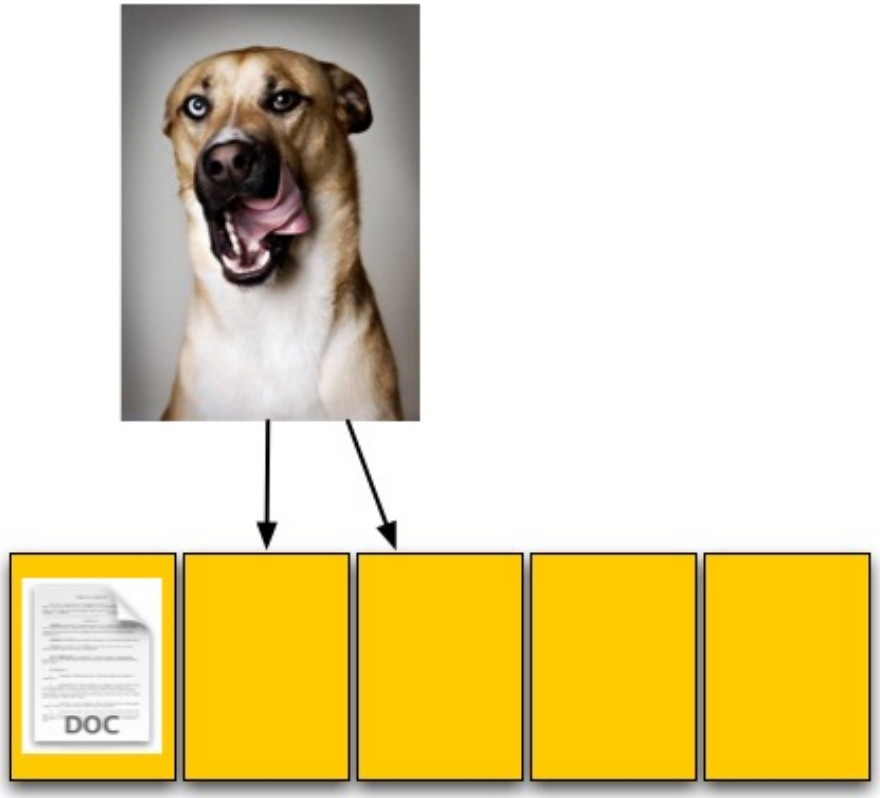
- Hypothetical SSD:
  - Page Size: 4KB
  - Block Size: 5 Pages
  - Drive Size: 1 Block
  - Read Speed: 2 KB/s
  - Write Speed: 1 KB/s



Example:

- Now lets write a 8Kb pic file to the almost brand new SSD, thankfully there's space
- Overall Writing Time:
  - 8 seconds!

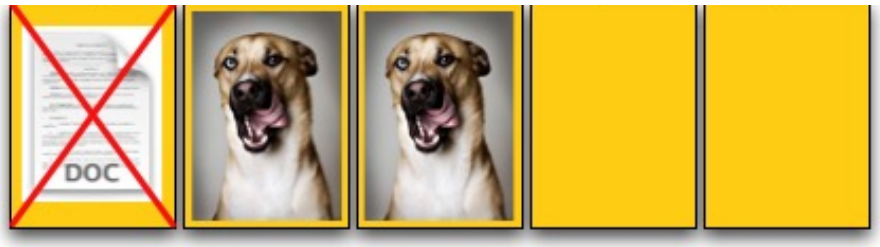
- Hypothetical SSD:
  - Page Size: 4KB
  - Block Size: 5 Pages
  - Drive Size: 1 Block
  - Read Speed: 2 KB/s
  - Write Speed: 1 KB/s



Example Cont.

- Let's now consider that the txt file in the first page is not more needed

- Hypothetical SSD:
  - Page Size: 4KB
  - Block Size: 5 Pages
  - Drive Size: 1 Block
  - Read Speed: 2 KB/s
  - Write Speed: 1 KB/s



Example Cont.

- Finally, let's write a 12kb pic to the SSD.
- How long should it take? 12s?

- Hypothetical SSD:
  - Page Size: 4KB
  - Block Size: 5 Pages
  - Drive Size: 1 Block
  - Read Speed: 2 KB/s
  - Write Speed: 1 KB/s

Example Cont.

- Finally, let's write a 12kb pic to the SSD.
- How long should it take? 12s?
  - **NO!!!!! 24 SECONDS!!!!!**

- Hypothetical SSD:
  - Page Size: 4KB
  - Block Size: 5 Pages
  - Drive Size: 1 Block
  - Read Speed: 2 KB/s
  - Write Speed: 1 KB/s

For the OS there are 3 free pages on the SSD when there are only 2 "empty".

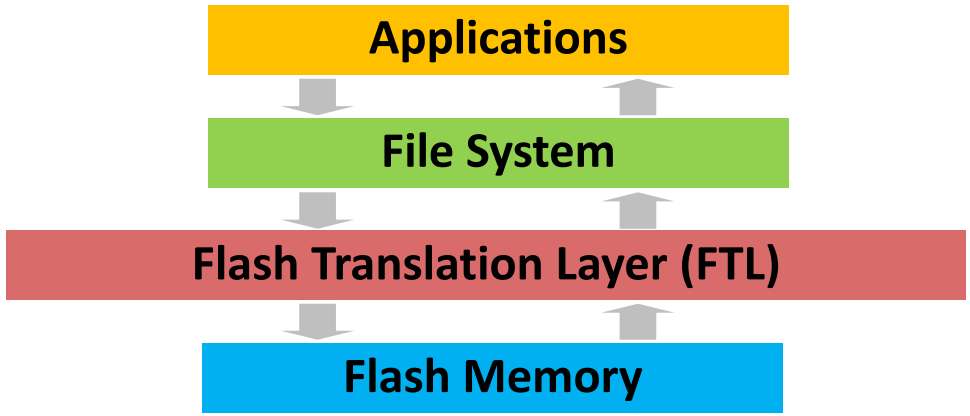
Example Cont.

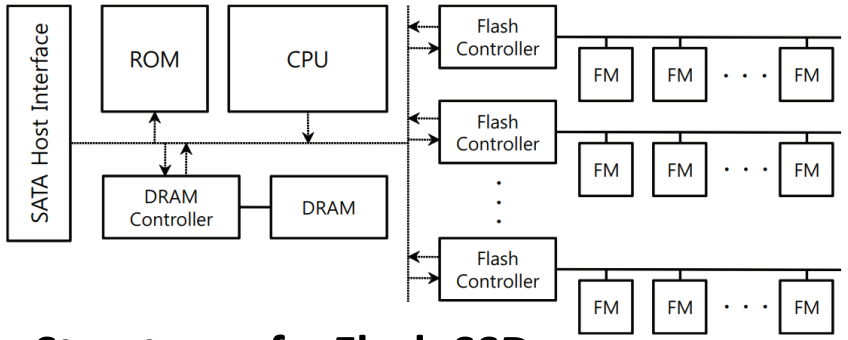
- Step 1: Read block into cache
- *(Step 2: Delete page from cache)*
- Step 3: Write new pic into cache
- Step 4: ERASE the old block on SSD
- Step 5: Write cache to SSD

- The OS only thought it was writing 12 KBs of data when in reality the SSD had to read 8 KBs (2 KB/s) and then write 20KBs (1 KB/s), the entire block.
- The writing should have taken 12 secs but actually took 4 + 20 = 24 seconds, resulting in a write speed of 0.5KB/s not 1KB/s

Flash Translation Layer

- Direct mapping between Logical to Physical pages is not feasible
- FTL is an SSD component that make the SSD “look as HDD”
  - Data Allocation and Address translation
    - Efficient to reduce Write Amplification effects
    - Program pages within an erased block in order (from low to high pages)
      - **Log-Structured FTL**
  - Garbage collection
    - Reuse of pages with old data (Dirty/Invalid)
  - Wear leveling
    - FTL should try to spread writes across the blocks of the flash ensuring that all of the blocks of the device wear out at roughly the same time.

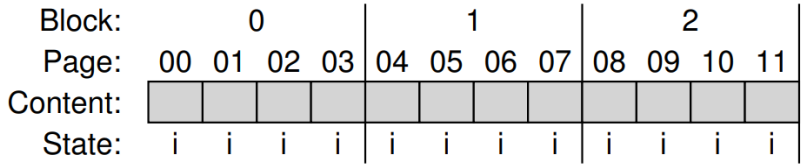




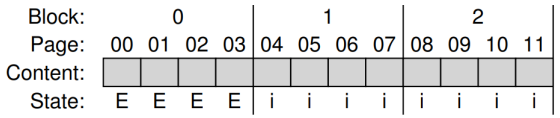
Structure of a Flash SSD

Example: Log-Structured FTL

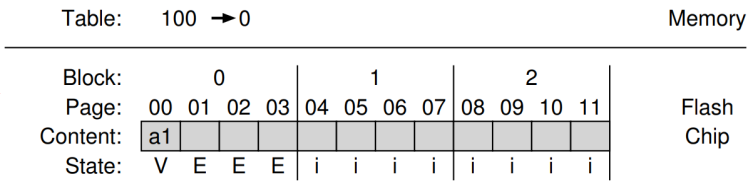
- Assume that a page size is 4 Kbyte and a block consists of four pages.
- Write(pageNumber, value)
  - Write(100, a1)
  - Write(101, a2)
  - Write(2000, b1)
  - Write(2001, b2)
  - Write(100, c1)
  - Write(101, c2)
- The initial state is with all pages marked `INVALID(i)`



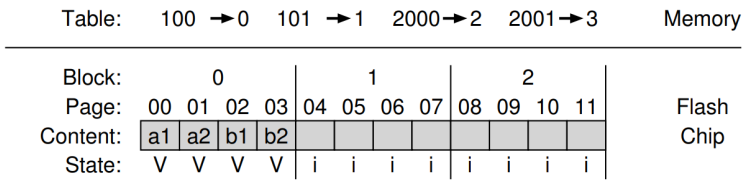
- Erase block 0.



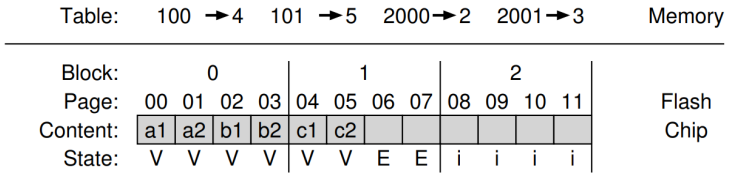
- Program pages in order and update mapping information.



- After performing four writes

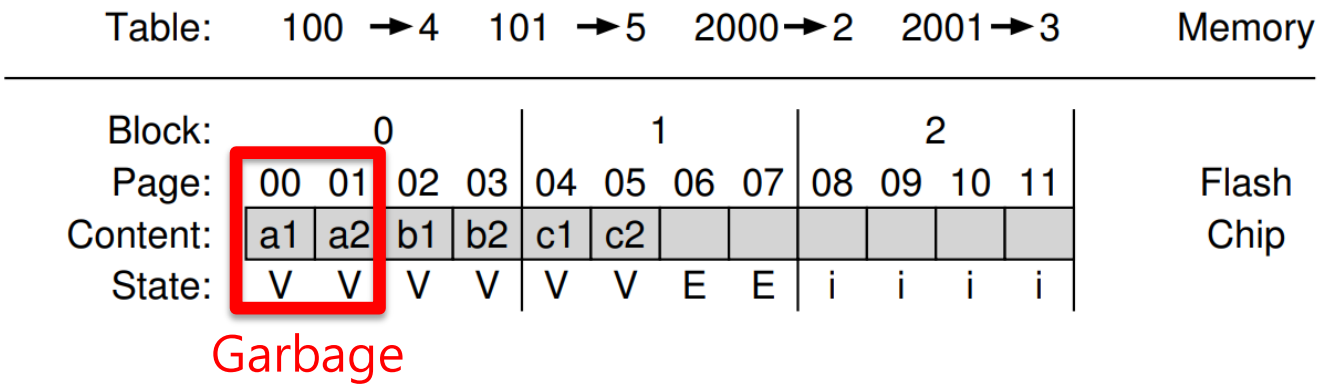


- After udating 100 and 101



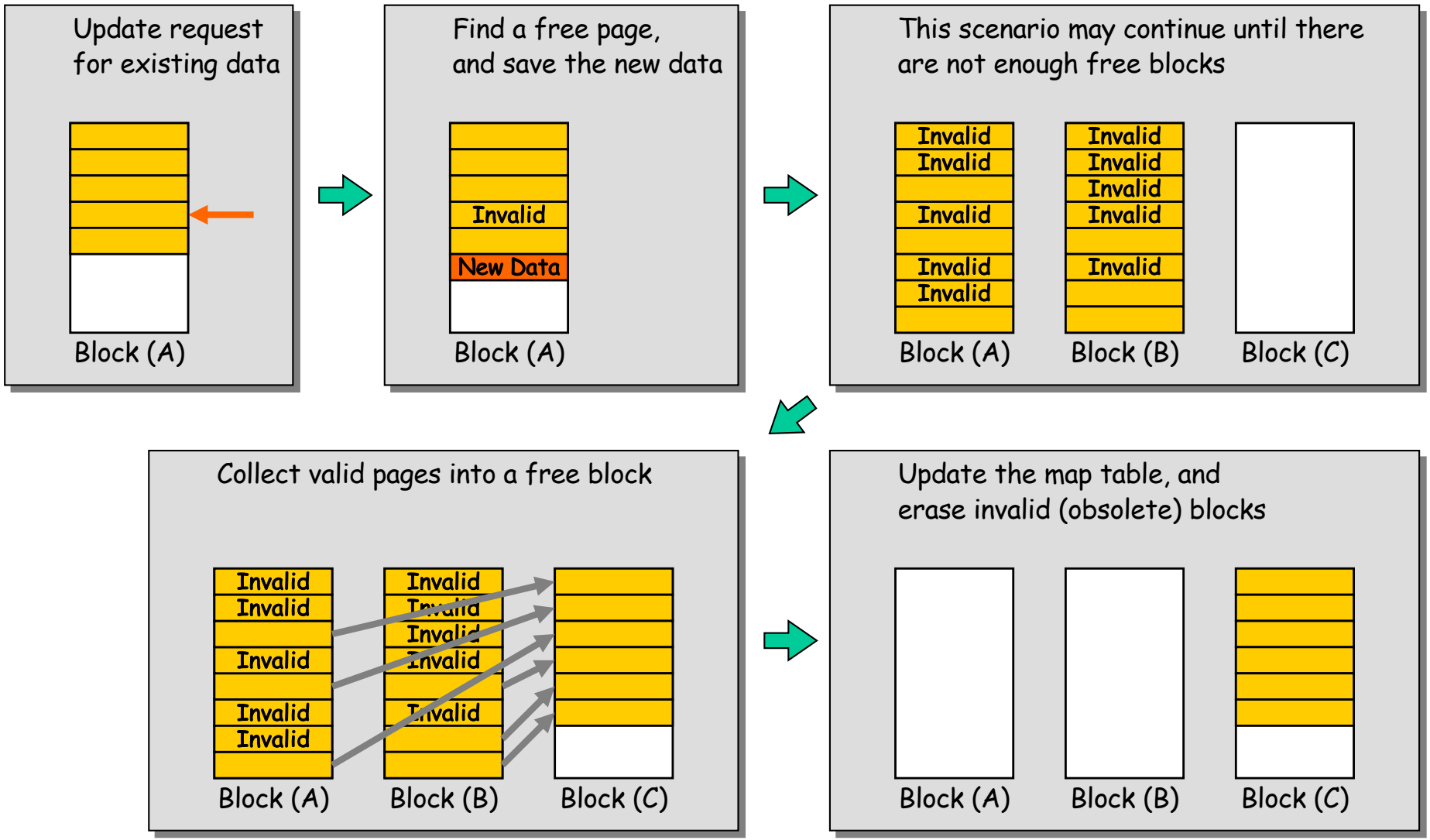
Garbage collection

- When an existing page is updated → old data becomes obsolete!



- Old version of data are called garbage and (sooner or later) garbage pages must be reclaimed for new writes to take place
- **Garbage Collection** is the process of finding garbage blocks and reclaiming them
  - Simple process for fully garbage blocks
  - More complex for partial cases. I.e. Basic steps:
    - Find a suitable partial block
    - Copy non-garbage pages
    - Reclaim (erase) the entire block for writing

Garbage collection (Example)

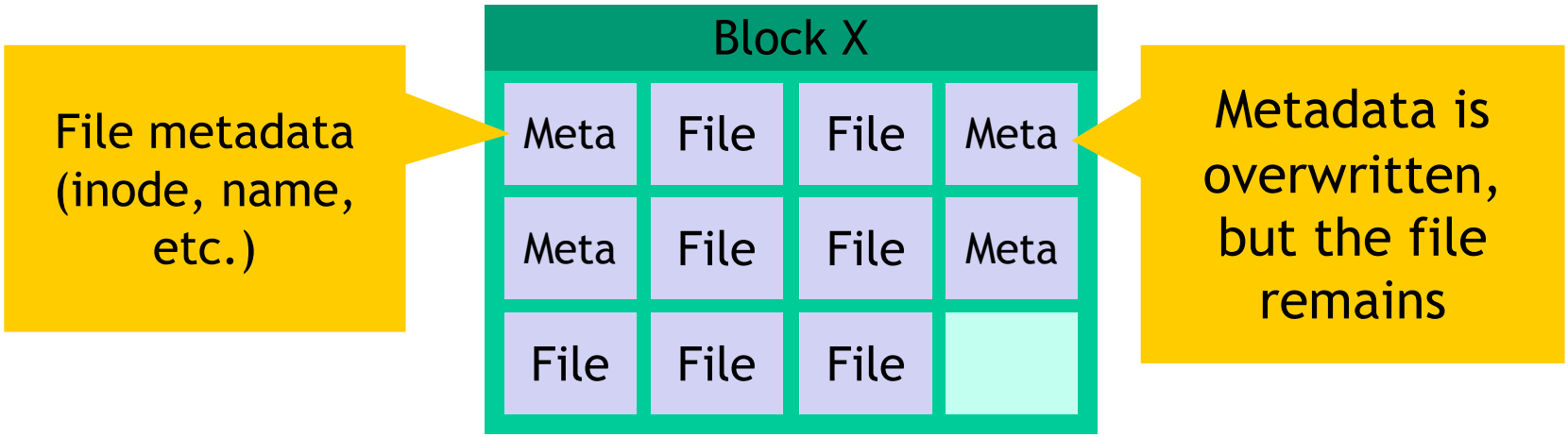


Garbage Collection Cost

- Garbage collection is expensive
  - Require reading and rewriting of live data.
  - Ideal garbage collection is reclamation of a block that consists of only dead pages
- The cost of Garbage Collection depends on the amount of data blocks that have to be migrated
- Solutions to alleviate the problem:
  - Overprovision the device by adding extra flash capacity
    - Cleaning can be delayed
  - Run the Garbage Collection in the background using less busy periods for the disk

Garbage Collection: The Ambiguity of Delete

- When performing background Garbage Collection the SSD assumes to know which pages are invalid
- Problem: Most file systems don't actually delete data
  - E.g. On Linux, the "delete" function is unlink() and it removes the file meta-data, but not the file itself

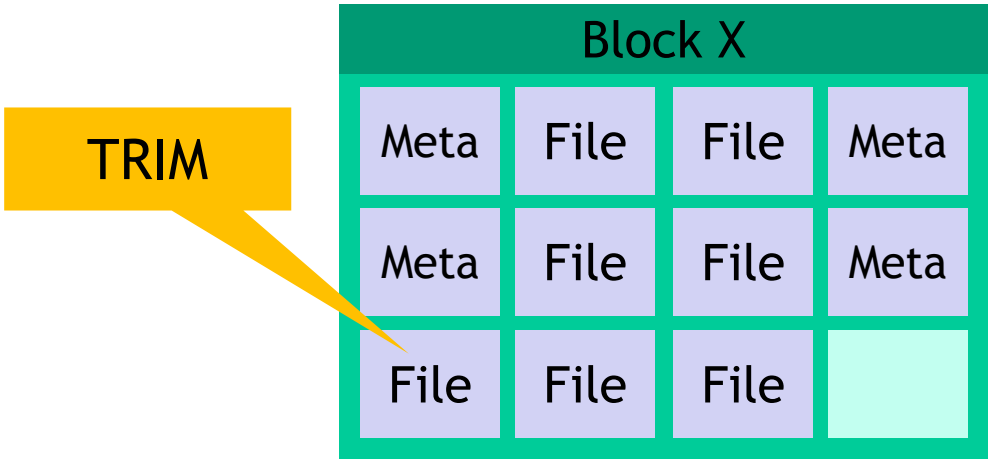


1. File is written to SSD
2. File is deleted
3. The GC executes
   - 9 pages look valid to the SSD
   - The OS knows only 2 pages are valid

- Lack of explicit delete means the GC wastes effort copying useless pages
- Hard drives are not GCed, so this was never a problem

Garbage Collection: The Ambiguity of Delete

- When performing background GC the SSD assumes to knows which pages are invalid
- Problem: most file systems don't actually delete data
  - E.g. On Linux, the "delete" function is unlink() and it removes the file meta-data, but not the file itself
- New SATA command TRIM (SCSI – UNMAP)
  - The OS tells the SSD that specific LBAs are invalid and may be GCed
  - OS support for TRIM
    - Win 7, OSX Snow Leopard, Linux 2.6.33, Android 4.3

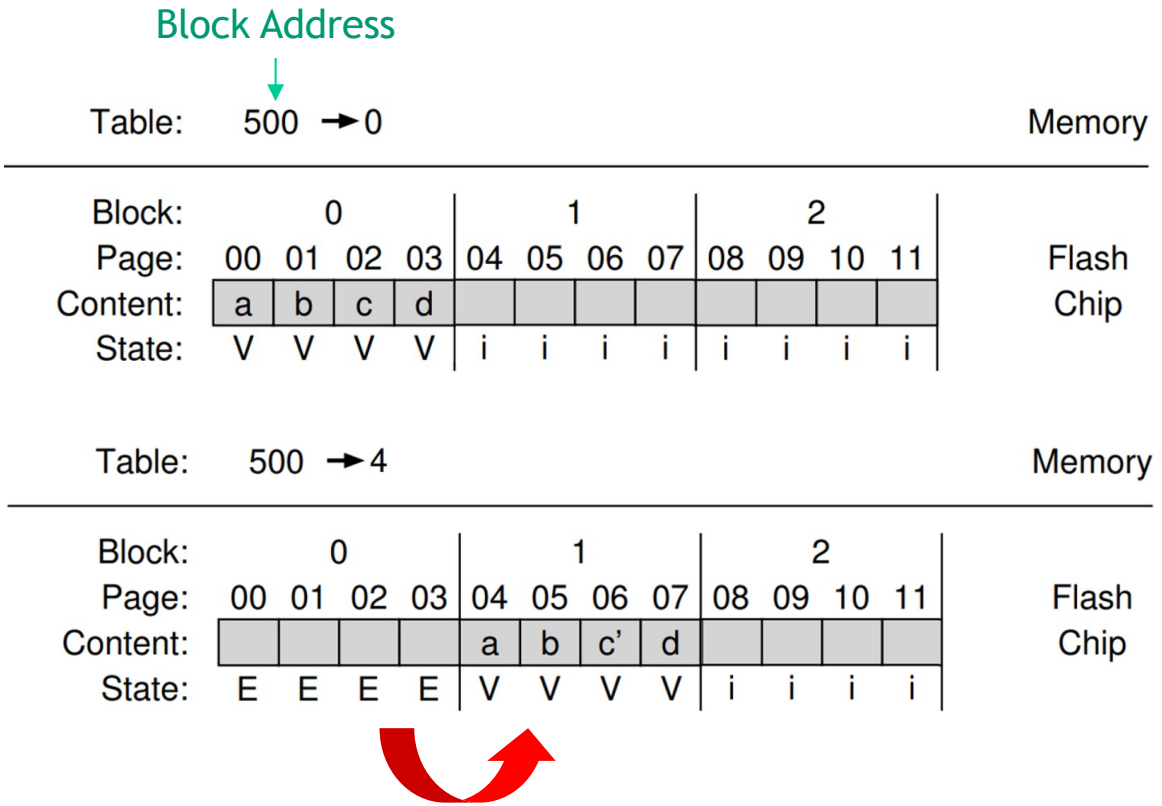


Mapping Table Size

- The size of page-level mapping table is too large
  - With a 1TB SSD with a 4byte entry per 4KB page, 1GB of DRAM is needed for mapping.
- Some approaches to reduce the costs of mapping
  - Block-based mapping
    - Coarser grain approach
  - Hybrid mapping
    - Multiple tables
  - Page mapping plus caching
    - Exploiting Data Locality

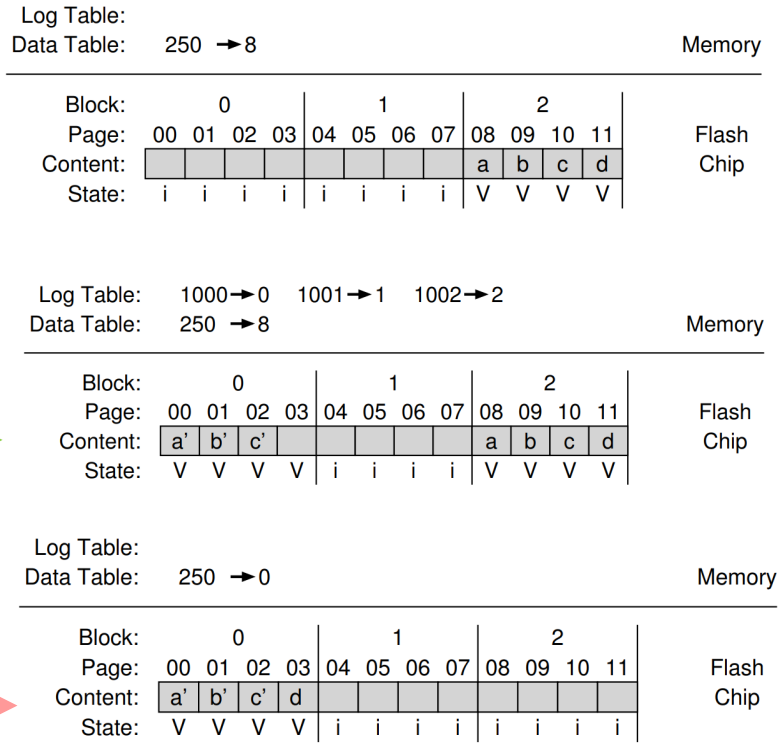
Block Mapping

- Mapping at block granularity
  - To reduce the size of a mapping table
- **Small write problem:**
  - The FTL must read a large amount of live data from the old block and copy them into a new one.
- **Example:**
  - Write(2000, a)
  - Write(2001, b)
  - Write(2002, c)
  - Write(2003, d)
  - Write(2002, c')



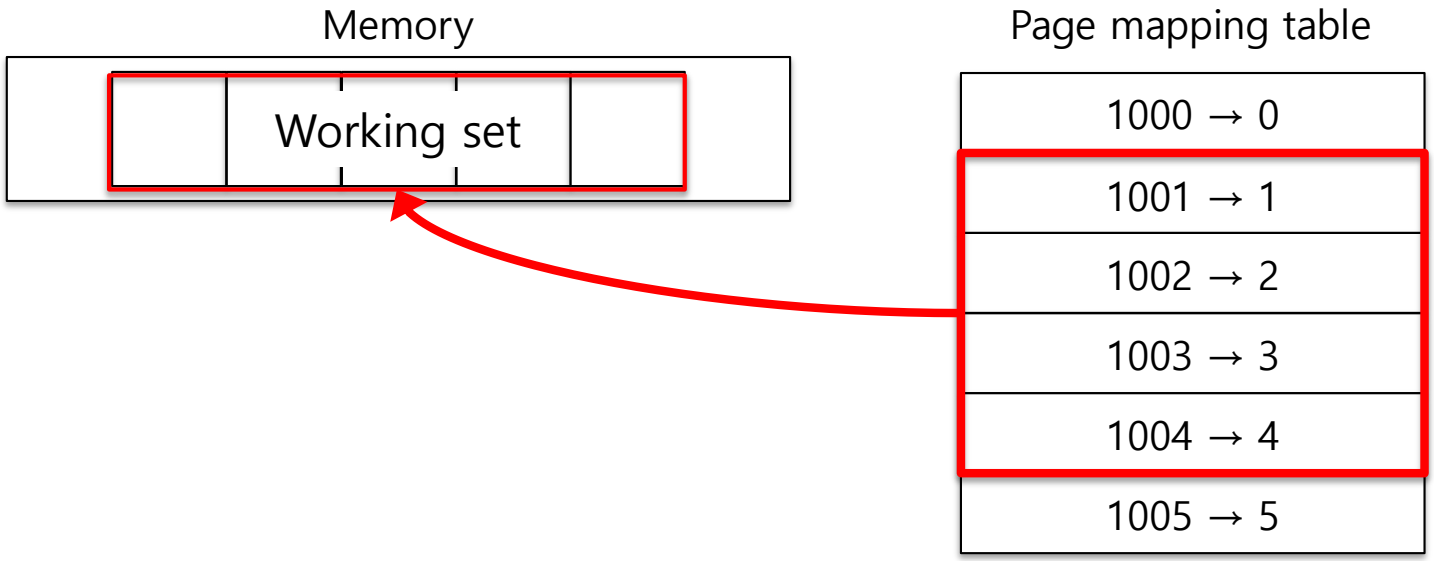
Hybrid mapping

- FTL maintains two tables:
  - Log blocks: page mapped
  - Data blocks: block-mapped
- When looking for a particular logical block, the FTL will consult the page mapping table and block mapping table in order
- Example:
  - Let's suppose the past sequence
    - Write(1000, a)
    - Write(1001, b)
    - Write(1002, c)
    - Write(1003, d)
  - Let's update some pages
    - Write(1000, a')
    - Write(1001, b')
    - Write(1002, c')
    - FTL updates only the page mapping information
  - When needed FTL can perform MERGE operations



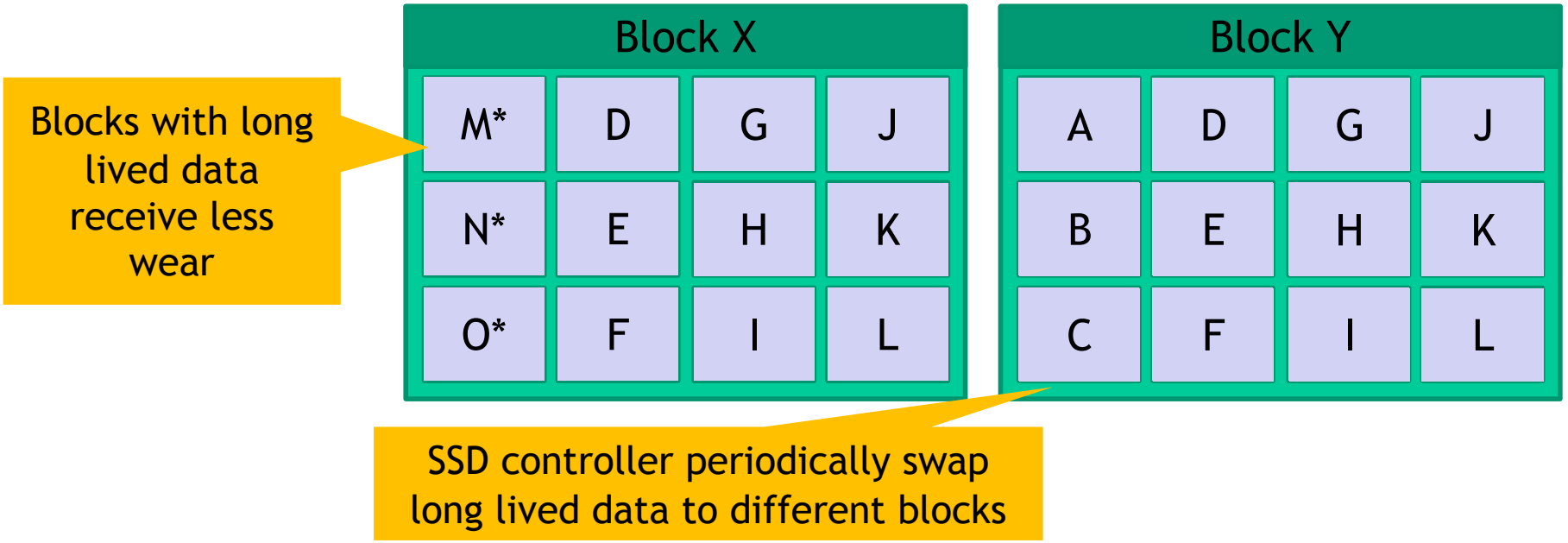
Page mapping plus caching

- Basica idea is to cache the active part of the page-mapped FTL
  - If a given workload only accesses a small set of pages, the translations of those pages will be stored in the FTL memory
- High performance without high memory cost if the cache can contain the necessary working set
- Cache miss overhead exists



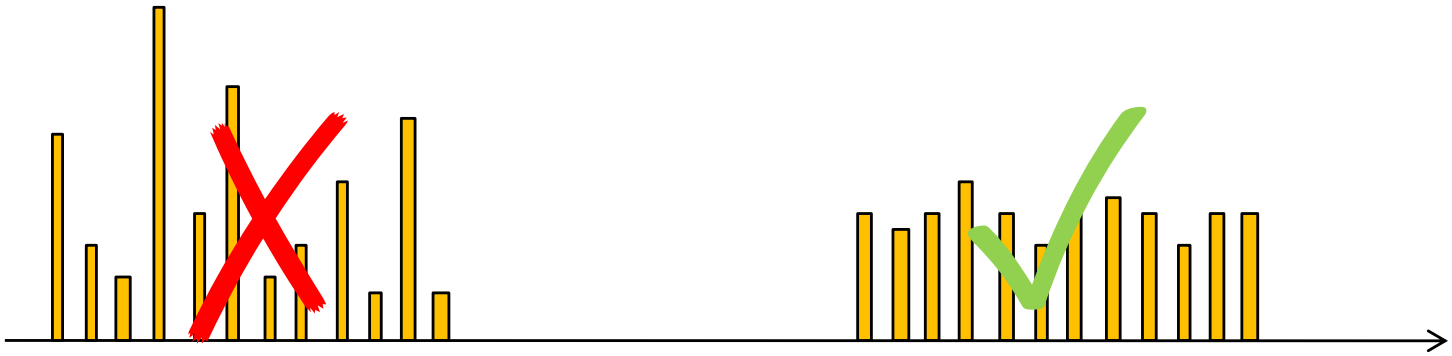
Wear Leveling

- Erase/Write cycle is limited in Flash memory
  - Skewness in the EW cycles shortens the life of the SSD
  - All blocks should wear out at roughly the same time
- Log-Structured approach and garbage collection helps in spreading writes. However, a block may consist of cold data
  - The FTL must periodically read all the live data out of such blocks and re-write it elsewhere



Wear Leveling

- Erase/Write cycle is limited in Flash memory
  - Skewness in the EW cycles shortens the life of the SSD
  - All blocks should wear out at roughly the same time
- Log-Structured approach and garbage collection helps in spreading writes. However, a block may consist of cold data
  - The FTL must periodically read all the live data out of such blocks and re-write it elsewhere
  - Wear leveling increases the write amplification of the SSD and decreases performance
    - Simple Policy: Each Flash Block has EW cycle counter
      - Maintain |Max(EW cycle) – Min(EW cycle)| < e

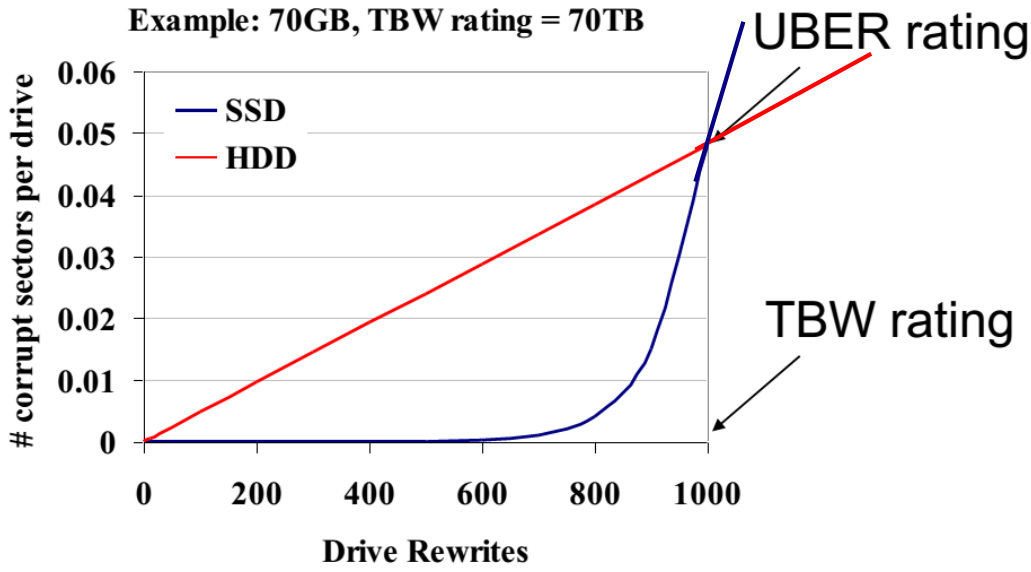


Summary - Solid State Disks (SSD)

- Flash-based SSDs are becoming a common presence in …
  - laptops, desktops, and servers inside the datacenters.
- They cost more than the conventional HDD
- Flash memory can be written only a limited number of times
- Different read/write speed
  - Write amplification
- SSD are not affected by data-locality and must not be defragmented (*defragmentation reduces the disk lifetime*)
- Flash Translation Layer is one of the key components
  - Data Allocation
  - Address Translation
  - Garbage Collection
  - Wear Leveling

HDD vs SSD

- **Unrecoverable Bit Error Ratio (UBER):** A metric for the rate of occurrence of data errors, equal to the number of data errors per bits read
- **Endurance rating: Terabytes Written (TBW)** is the total amount of data that can be written into an SSD before it is likely to fail). The number of terabytes that may be written to the SSD while still meeting the requirements.



Capacity and TBW

- **Memory cells can accept data recording between 3,000 and 100,000 during its lifetime**
  - Once the limit value is exceeded, the cell "forgets" any new data
- **A typical TBW for a 250 GB SSD is between 60 and 150 Terabytes of data written to the drive.**
  - This means that in order to overcome, for example, a TBW of 70 Terabytes, a user should write 190 GB every day for a year or fill his SSD on a daily basis for two thirds with new files for a whole year
- **It is difficult to comment on the duration of SSDs.**
  - Dell, in an old study (2011), spoke of an estimated duration between three months and ten years explaining, however, that there are so many factors (temperature and workload) that may depend on the life of an SSD that is very difficult to make predictions.