Flash Memory

Mendel Rosenblum

Flash Memory

Optional readings: Operating Systems: Principles and Practice: Section 12.2

Flash Memory

- Flash memory has replaced magnetic disks in most devices
  - Phones, laptops, etc
  - Packaged into a storage device: **solid state disks (SSDs)**

Flash compared to other memory technologies

Comparison to magnetic disk:

- No moving parts, so more reliable, more shock-resistant
- ~100–1000× lower latency for random access
- Cost/bit 3-10x higher than disk

Comparison to DRAM:

- Nonvolatile: values persist even if device is powered off
- Cost/bit 5-20x lower
- 100-1000x slower

Flash memory cell characteristics

- Reads are relatively fast
  - Typically ~10–100 microseconds
- Flash cell writes are asymmetric:
  - Changing 1 → 0 is relatively fast (~100–1000 microseconds)
  - Changing 0 → 1 is much slower (~1000–10,000 microseconds)
- Memory is accessed in **pages**
  - Typical sizes 4 Kbytes - 16 Kbytes)
  - More like a disk than main memory
- Total chip capacity up to 2 Terabytes

Flash memory weirdness: writes

- Need to erase (write all "1"s) before writing a page
  - Storage divided into **erase units** (typically 1-8 Mbytes, many pages)
  - Slow: 1000 - 10,000 microseconds
- Write: modifies an individual page, can only clear bits to 0
  - Effectively a logical AND operation
  - Can write a page multiple times but only clear more bits
  - Need to erase the "erase unit" containing the page to write a 1
- Erasing wears out the erase unit
  - 100 - 100,000 times before it can't reliably hold information
  - Called **wear out**

Typical flash memory performance

- Read page
  - ~10-100 microseconds latency
  - Throughput depends heavily on parallelism:
    - single NAND chips: hundreds of Mbytes/sec
    - SSD devices: multiple Gigabytes/sec
- Erasure time: around 2 ms
- Write page
  - ~100–1000 microseconds latency
  - Throughput depends heavily on parallelism:
    - single NAND chips: hundreds of Mbytes/sec
    - SSD devices: multiple Gigabytes/sec

Solid State Disks (SSDs)

- Erase unit doesn't work for existing file systems
- Flash memory devices are packaged with a **flash translation layer** (FTL)
  - Software that manages the flash device
  - Typically provides an interface like that for a disk (read and write blocks)
    - Exports linear array of blocks (virtual block number)
    - Stored on a flash memory page (physical block number)
  - Allows existing file system to use SSDs

Flash Translation Layer issues

- Sacrifice performance
- Waste space
- Proprietary implementation

Will present some possible techniques used in FTLs

Direct Mapped FTL

- Map virtual block to a physical page
- Read block N
  - Read page N
- Write block N
  - Read erase unit containing page N
  - Erase erase unit containing page N
  - Re-write erase unit include new page N content

Direct Map Problems

- Writes are super slow: 2ms erase time added to each write
- Repeated writing a block will hit wear limits
- Crash can lose old data

Used on some inexpensive USB memory sticks (with the FAT file system)

Better approach for FTLs

- Separate virtual block number from physical location in flash memory
  - Virtual block can occupy different pages in flash memory over time
- Keep a **block map** that maps from virtual blocks to physical pages
  - For reads:
    - Lookup the physical location in the block map
  - For writes:
    - Find a free and erased page
    - Write virtual block to that page
    - Update map with new physical location
    - Mark previous page for virtual block as free

Block map management

How to manage map (is it stored on the flash device?)

How to manage free space (e.g. wear leveling)

Approach #1

- Keep block map in memory, rebuild on startup
  - Don't store block map on flash device
- Each page on flash contains an additional header:
  - Virtual block number
  - Allocated bit
    - 1 => free, 0 => allocated
  - Written bit
    - 1 => not yet written (all ones), 0 => written
  - Garbage bit
    - 0 => no longer in use, must be erased before reusing page

A-W-G bit track page lifecycle

	A	W	G
Just erased - Not allocated, not Written, not Garbage	1	1	1
Allocated but not yet written	0	1	1
Block successfully written	0	0	1
Block deleted (new copy written elsewhere)	0	0	0

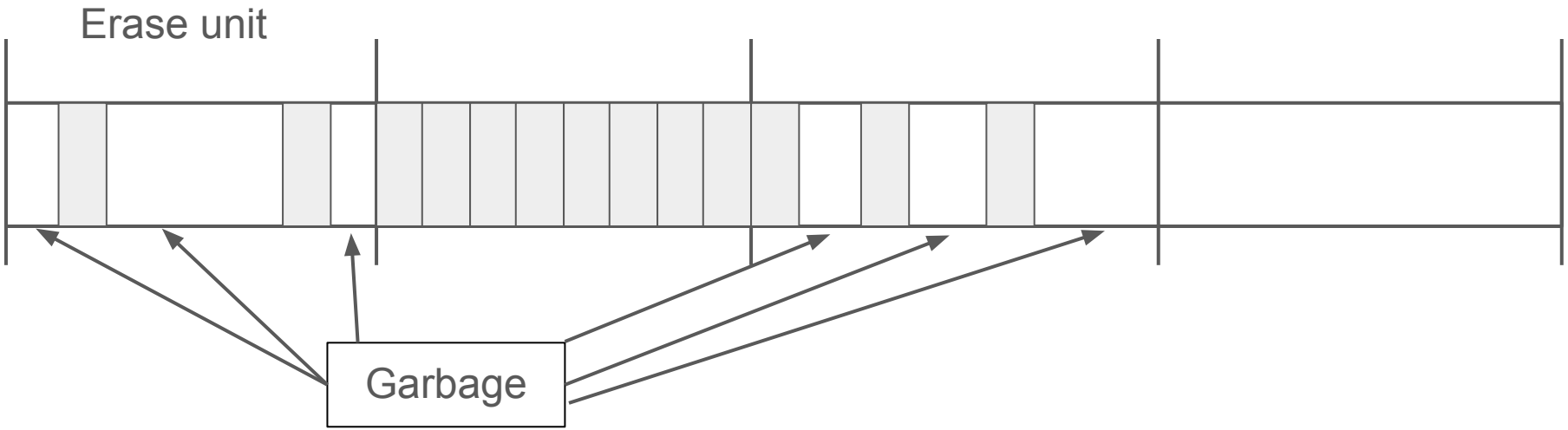
Why is 0-1-1 state needed?

Detects crashes that occur while writing a block

On crash: Scan all headers, rebuild map and free map

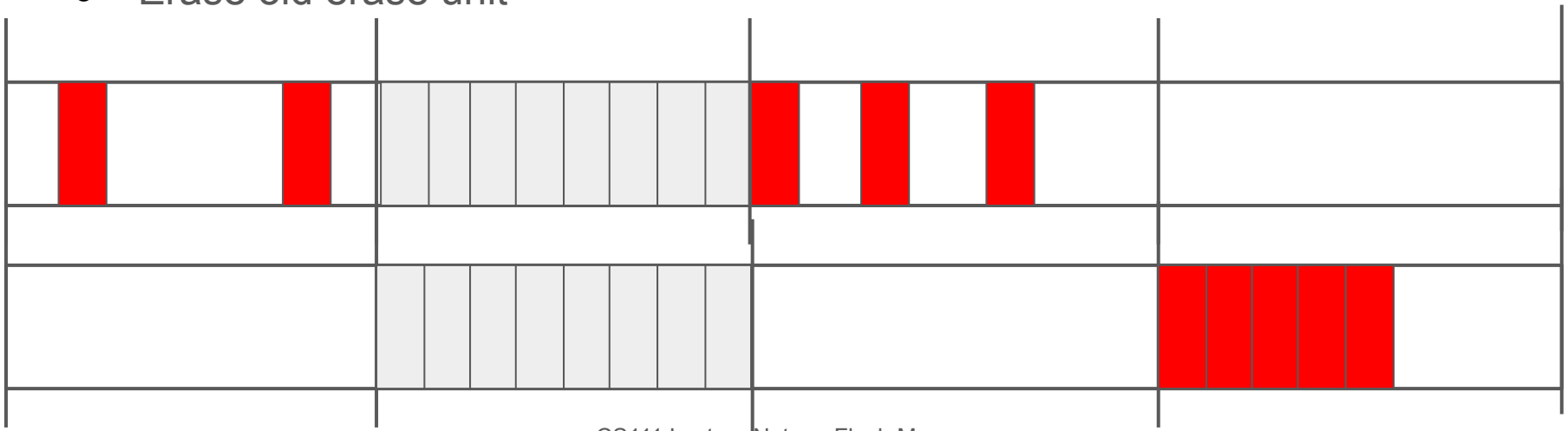
Problem: garbage pages accumulate in erase units

- Garbage pages reduces effective capacity



Solution: garbage collection

- Find erase unit with many garbage pages
- Copy live pages to clean erase unit (update block map)
  - Use rest of erase unit for new pages
- Erase old erase unit



Garbage collection write problem

- Every page write creates one garbage page on average
- Garbage collection reads garbage pages and writes them again
- How much write overhead is there?
  - Depends on how much garbage we have to collect
- Hard to achieve good performance and good utilization at the same time

Write amplification

- Garbage collection
  - Read old, live pages from fragmented erase units
    - Assume an average utilization of U ($0 \leq U \leq 1$)
    - Read U pages of old data
  - Rewrite old pages
    - Write U pages of old data
  - Free up space for 1 - U pages of new data
- For each erase unit we write, we write 1/(1 - U) units
  - Called **write amplification**

Write amplification is bad

U	1/(1 - U)
.5	2
.9	10
.99	100

- Frequent garbage collection also wears out the device faster
- Lower utilization makes writes cheaper, but wastes space

Write amplification formula too high for real systems

- A 50% filled flash memory has an average U = .5
  - Garbage collector doesn't have to target **average** U erase units
- Ideal situation:
  - Memory is packed with nearly filled erase units (average U near 1)
  - Collector finds nearly empty erase units (U near 0)
- There are ways to encourage such a bimodal distribution

Encouraging bimodal U distribution

- Exploit **locality of reference**
- Put disk blocks that are updated at the same time into erase unit pages
- Segregate erase units by "temperature" of the blocks
  - **Hot** - Likely to be overwritten in the near future
  - **Cold** - Unlikely to be changed in the near future
- Garbage collector targets Hot erase units with small U values

Wear-leveling

- Problem: Hot erase units get erased constantly and wear out
  - Ideally we spread erase cycles across all the erase units
- Occasionally garbage collect a cold erase unit
  - Won't get much space back
  - Will get an unworn erase unit to be used for new data
- Called **wear-leveling**
- Can effectively spread erase across entire flash memory

Flash memory as a disk with FTL is inefficient

- Duplication
  - File block -> logical disk block -> flash page
  - Why not in the FS: File block -> flash page?
    - Inexpensive flash storage had FTL so flash-optimized file systems lost
- Lack of information
  - FTL doesn't know when OS has freed a block; only when overwrite happens
  - Much FTL garbage collection overhead copying deleted file blocks
  - Fix: Have file system tell flash memory when blocked is freed
    - Flash memory added a TRIM command
- Current approach: File systems that work well with FTLs
  - Let FTLs handle wear-levelling, bad blocks, erase scheduling, etc.

Looking forward

- Flash memory encourages file systems that don't update in place
- Disks discouraged file systems that don't update in place
  - Messes up layout for reads
- Current file systems for SSDs try not to update in place