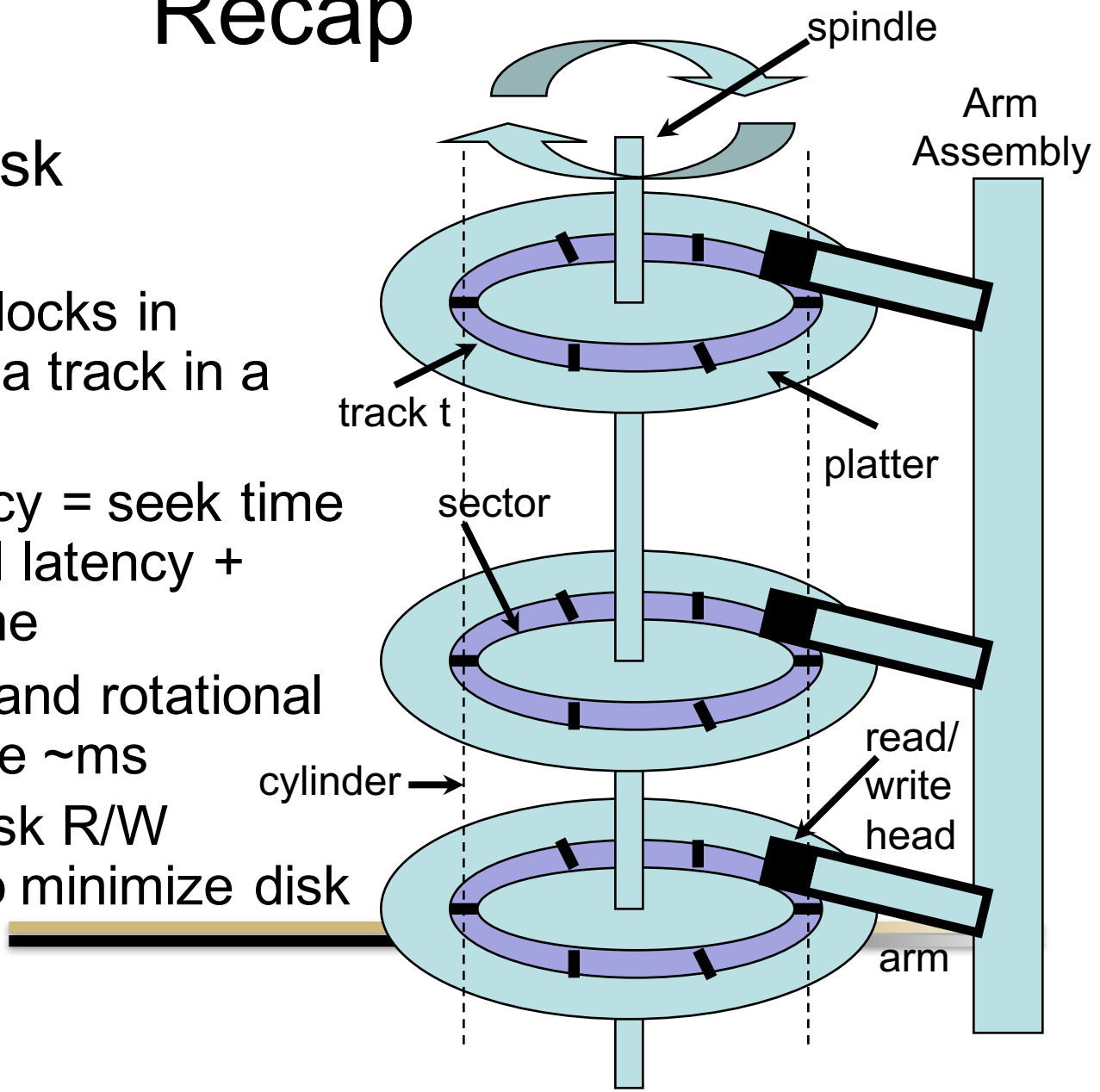
Chapters 10, 11 and 12: Flash File Systems, RAID

CSCI 3753 Operating Systems

Prof. Rick Han

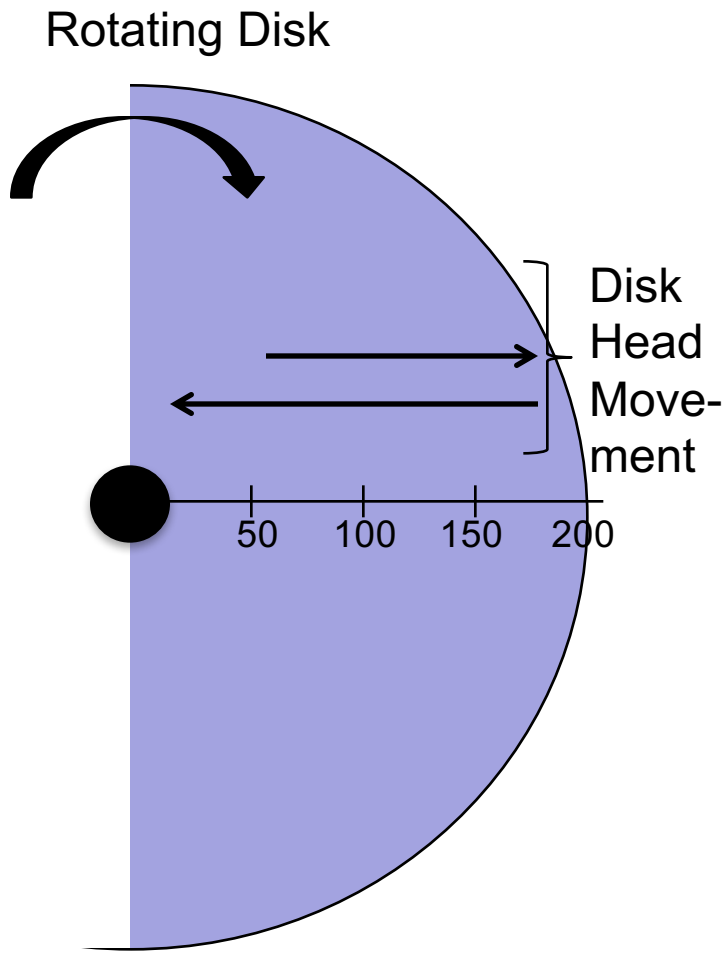
Recap

- Magnetic Disk Scheduling
  - Store file blocks in sectors on a track in a disk platter
  - Total latency = seek time + rotational latency + transfer time
  - Seek time and rotational latency take ~ms
  - Reorder disk R/W requests to minimize disk seek time



Recap

- Magnetic Disk Scheduling
  - FCFS
  - SSTF
  - OPT
  - SCAN approximates OPT
  - LOOK improves SCAN
  - Circular versions of SCAN and LOOK for fairness
- File layout



What is Flash Memory?

- Flash memory is a type of solid state storage
  - bits are stored in an electronic chip, not on a magnetic disk
  - Programmed and erased electrically – no mechanical parts
  - Non-volatile storage, i.e. "permanent" with caveats

- Primarily used in solid state drives (SSDs), memory cards, USB flash drives, mobile smartphones, digital cameras, etc.

What is Flash Memory?

- Flash memory is a form of EEPROM
  - Electrically Erasable Programmable Read Only Memory
  - bits can be rewritten again and again using ordinary electrical signals, non-volatile

- Compare to:

RAM (Random Access Memory) – main memory, volatile

ROM (Read Only Memory) – code is burned in at the factory and can only be read thereafter, no writes

PROM (Programmable ROM) – code can be written once using a special machine

EPROM (Erasable PROM) – code can be written multiple times with a special machine

What is Flash Memory?

- Advantages vs. disk:
  - much faster access (lower latency),
  - more resistant to kinetic shock (& intense pressure, water immersion etc.),
  - more compact,
  - lighter,
  - lower power (typically 1/3-1/2 of disks), …

What is Flash Memory?

- Disadvantages vs. disk:
  - more costly per byte,
  - limited lifetime,
  - erases are costly, …
- Flash's name is due to its electronic circuit design, which can inexpensively erase a large amount of solid state memory quickly, like a "flash" of light

NAND Flash

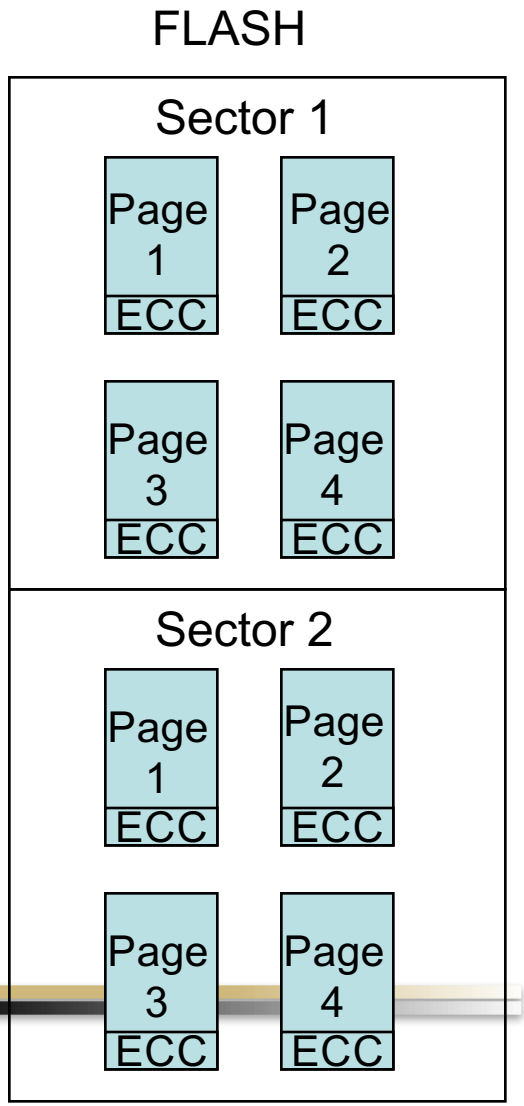
- Smaller memory cell area (less expensive)
- Word accessible (large granularity, 100-1000 bits/word, good for secondary storage where files don't mind being read out in large chunks/words)
- Slower random byte access (have to read a word)
- Short erasing (~2 ms) and programming times (~5 MB/sec sustained write speed)
- Longer write lifetime

NOR Flash

- Larger memory cell area
- Byte accessible (good for ROM-like program storage, where instructions need to be read out on a byte-size granularity)
- Faster random byte access
- Longer erasing (~750 ms) and programming times (~.2 MB/sec sustained write speed)
- Shorter write lifetime
- iPhone uses both multi-GB NAND flash for multimedia file storage and multi-MB NOR flash for code/app storage

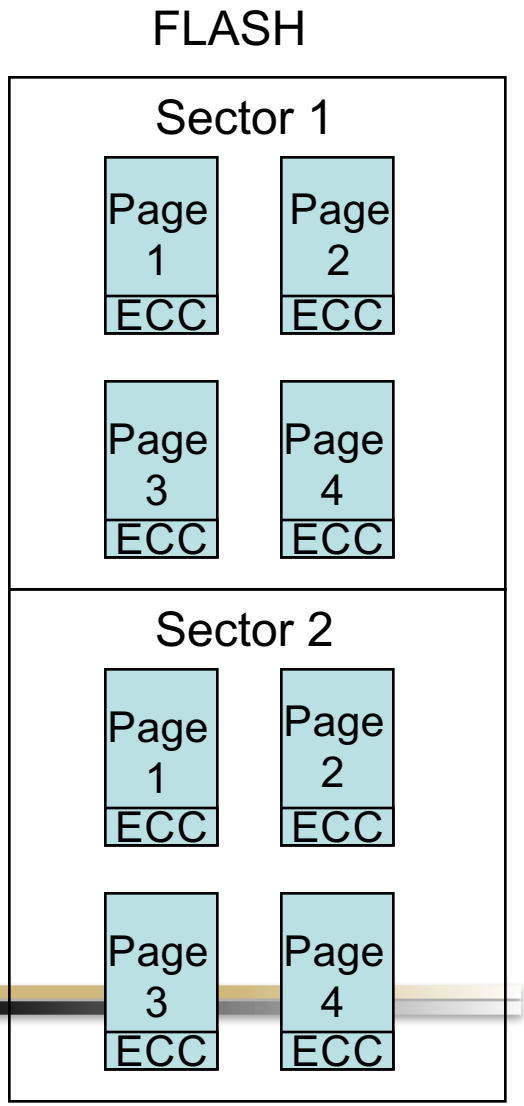
Organizing Flash Memory

- Flash memory is typically divided into blocks or *sectors*
  - There can be many *pages* per block/sector
    - e.g. 64 pages per block, 16 blocks per flash, and 4 KB per block => 4 MB of flash
  - The last 12-16 bytes of each page is reserved for error detection/correction (ECC) and bookkeeping
    - the flash file system can put information in this space



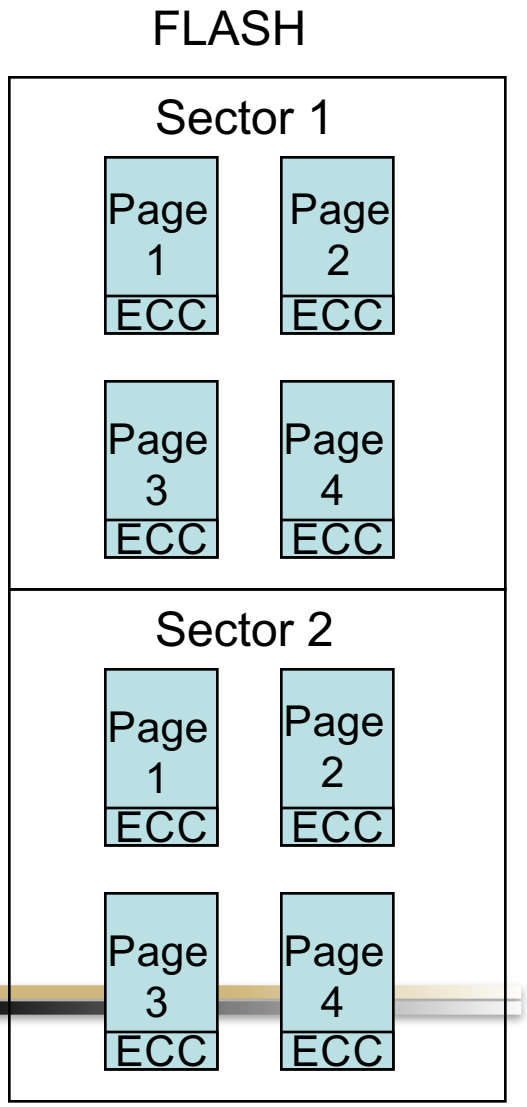
Operations on Flash Memory

- Reads and writes occur on a sub-page level granularity
  - Can read a byte (NOR) or word (NAND) by giving page + offset
- However, writes are tricky
  - Writes of a byte (NOR) or word (NAND) only proceed immediately if that memory has been cleanly reset/erased and not yet written to
  - "Rewrites" to memory that has already been written to *require an erase first before the write*



Rewrites Are Costly

- Unfortunately, erasing in flash can only occur on a sector granularity!
  - This is the price we pay for cheap "flash" technology
- Hence, to write just one byte into a memory location that has already been written to
  - First you have to erase the entire sector containing that byte address
  - Then write the byte!



Rewrites Are Costly

- Rewrites happen all the time.
  - Example: deleting a file
    - De-allocate the flash pages containing the file data
    - If we want to allocate these free flash pages to other files, we have to first erase the entire sectors containing these pages
- In comparison, the cost of rewrites is the same as writes for magnetic disk:
  - Can simply have magnetic head reset of the orientation of the bits on magnetic media
  - No need to erase before writing

Rewrites Are Costly

- To rewrite over a page of flash memory *while preserving* the other pages in the same sector:
  - Either:
    1. Copy the entire sector into RAM
    2. In RAM, write the changes to the page that you want to modify
    3. Erase the sector on flash
    4. Write all the sector's pages currently in RAM to the newly erased sector, including the modified page
  - Or:
    - Copy the entire sector into a new clean sector, except for the page to be modified
    - Write the one page to the correct location in the new sector (page is clean)

Limited Lifetime of Flash Memory

- Memory wear - Flash memory has a limited write lifetime
  - NOR flash memory can withstand 100,000 write-erase cycles before becoming unreliable
  - NAND flash memory can withstand 1,000,000 write-erase cycles
  - The strong electric field needed to set and reset bits eventually breaks down the silicon transistor
- Eventually, your mobile devices can no longer save data!
  - Fortunately, you'll probably upgrade your mobile device long before the lifetime limit is reached

Wear Leveling Solution

- To extend the life of flash memory chips, write and erase operations are spread out over the entire flash memory
  - Keep a count in 12-16 byte trailer of each page of how many writes have been made to that page, and choose the free page with the lowest write count
  - Randomly scatter data over the entire flash memory span, since it doesn't take any longer to extract data from nearby pages as distant pages
  - Many of today's flash chips implement wear leveling in the hardware's device controller

Problems Applying Standard Disk File Systems to Flash Memory

1. A typical disk file system stores data/metadata in fixed locations
   - the directory structure, file headers, allocation structures (e.g. FAT) and free space structures (e.g. bitmap) and even file data are in relatively fixed locations
   - Repeated modifications to these structures in static locations on flash would result in rapidly exhausting the write lifetimes of those locations, hence all of flash

Problems Applying Standard Disk File Systems to Flash Memory

2. Disk file systems are not optimized for rewrites
   - Disk file system rewrites are viewed essentially the same as writes, and repeated rapid rewrites can be quickly done because there is little disk seek time
   - Flash rewrites incur a much greater penalty, are much longer than writes to clean flash pages, and repeated rapid rewrites in flash would cause disastrously long latency, not to mention reduced lifetime

Flash File Systems

- Must be designed to deal with 2 major problems of flash memory:
  - Rewrites requires erases
  - Limited lifetime
- Log-structured file systems are well-suited to flash memory!
  - Writes to a file are added/appended to the end of the log, as are rewrites. The log is the file system.
  - The log grows sequentially and doesn't rewrite over the same location
  - Hence less erasing & also longer lifetime

Flash File Systems

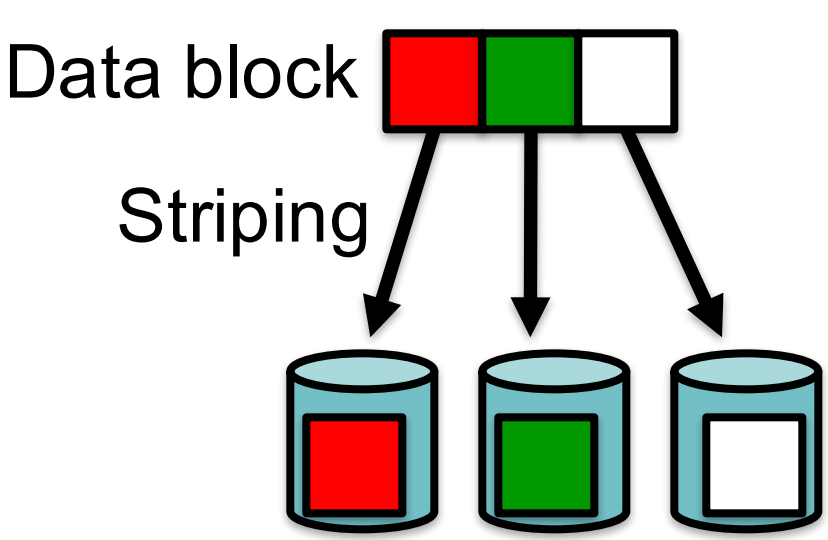
- Journaling Flash File System (JFFS) and JFFS2
  - Is a Log-structured file system similar to log-based recovery seen earlier for file system reliability
  - The file system is the log, and is written sequentially over flash pages,
    - The entries in the log contain all the data we need to reconstruct the file (recall the X & Y old and new values)
    - The log may be circular, eventually wrapping around once all free flash pages have been consumed.
      - Free space = distance between head and tail of log
      - As free space decreases, garbage collection is performed to free up some more space.

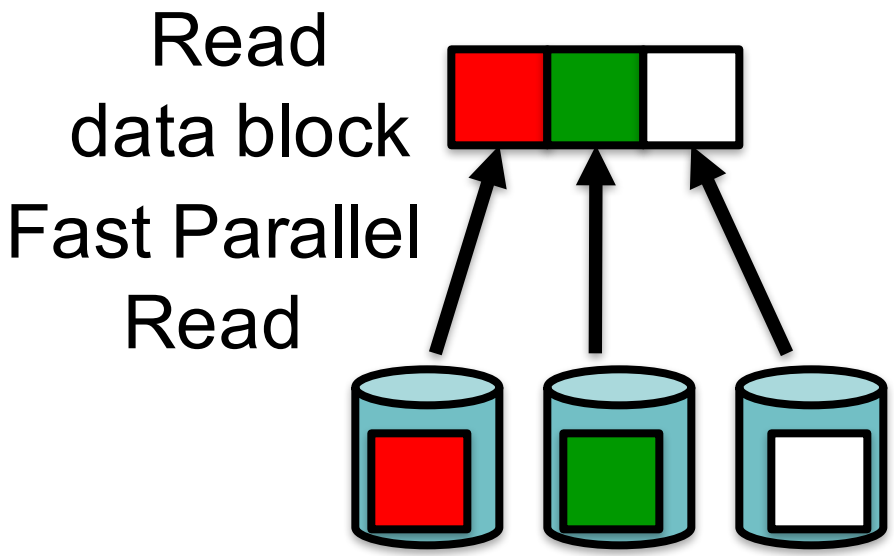
Redundant Arrays of Inexpensive Disks (RAID)

- Magnetic disks are cheap these days.
- Attaching an array of magnetic disks to a computer brings several advantages compared to a single disk:
  - Faster read/write access to data by having multiple reads/writes in parallel.
    - Data is striped across different disks, e.g. each byte of an 8-byte word is striped onto a different disk
  - Better fault tolerance/reliability
    - if one disk fails, a copy of the data could be stored on another disk – redundancy to the rescue!

RAID0

- RAID has different levels with increasing redundancy
  - RAID0 = data striping with no redundancy

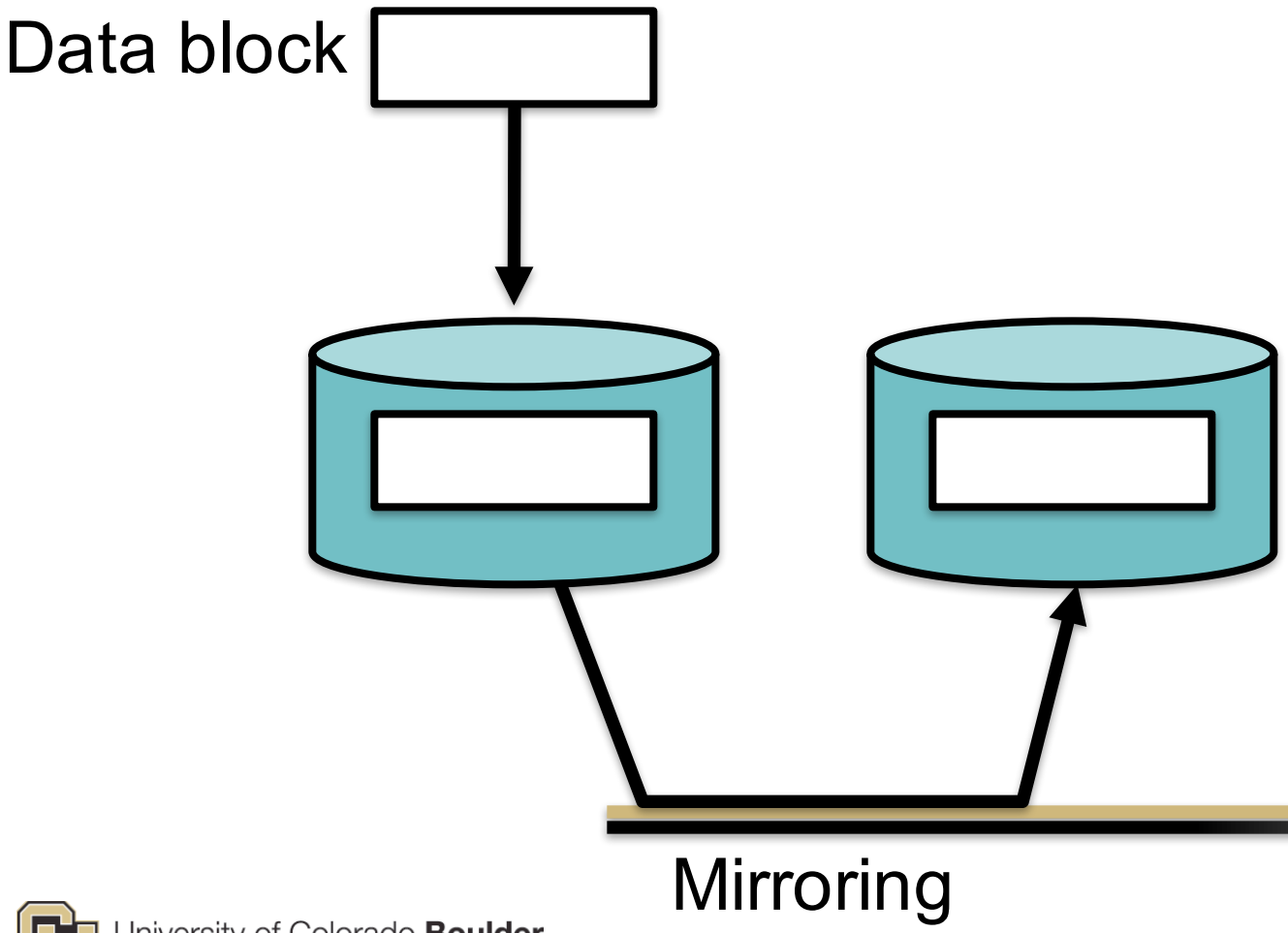




Also fast parallel writes

RAID1

– RAID1 = mirror each disk



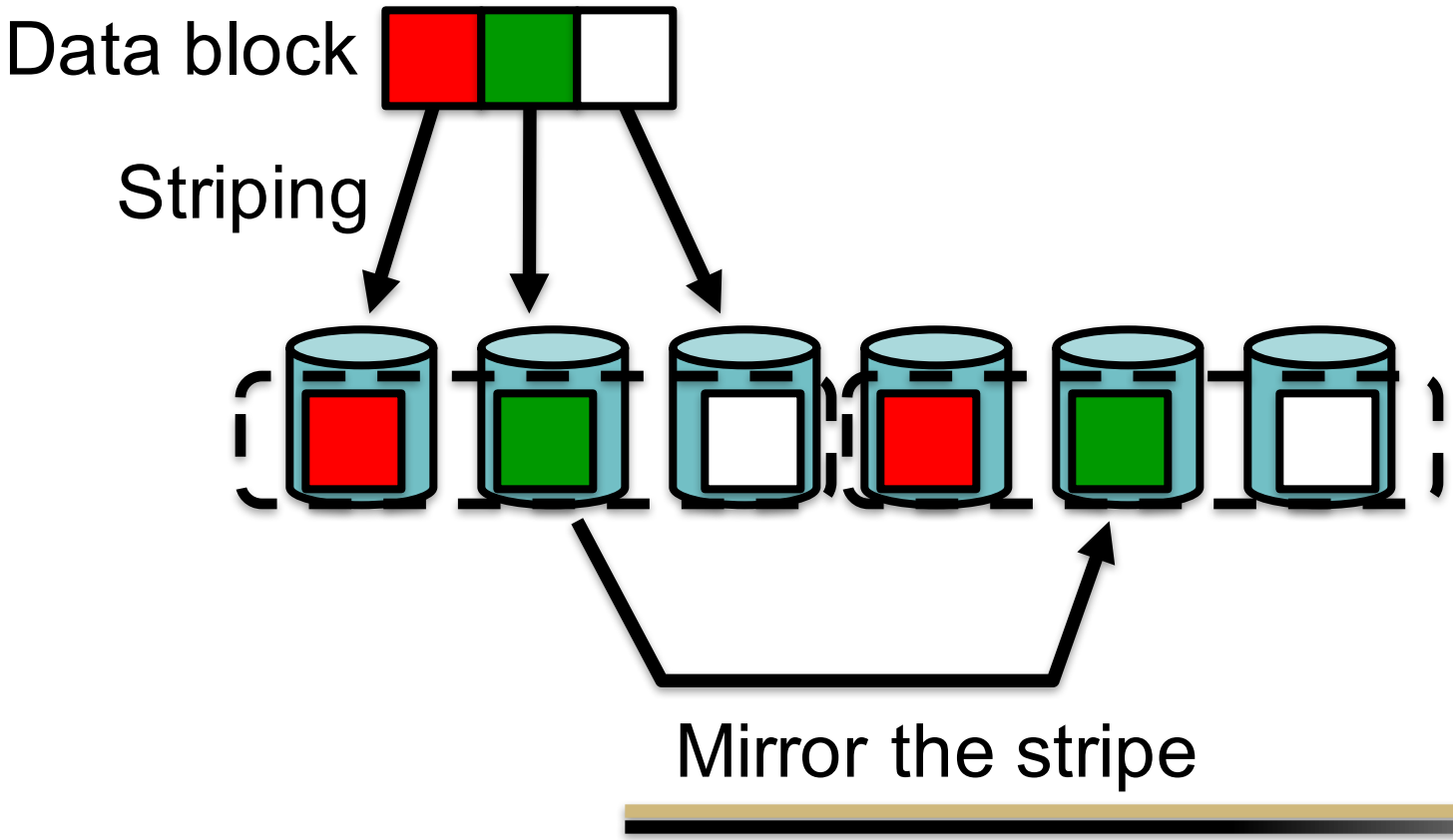
Get redundancy, but not parallel performance speedup of RAID0. Hence, combine RAID0 with RAID1 (next slide)

Can get limited read speedup by submitting the read to all mirrors, and taking the 1st data result

Twice the delay on writes, but OS can mask this by delaying writes

RAID0+1

– RAID0+1

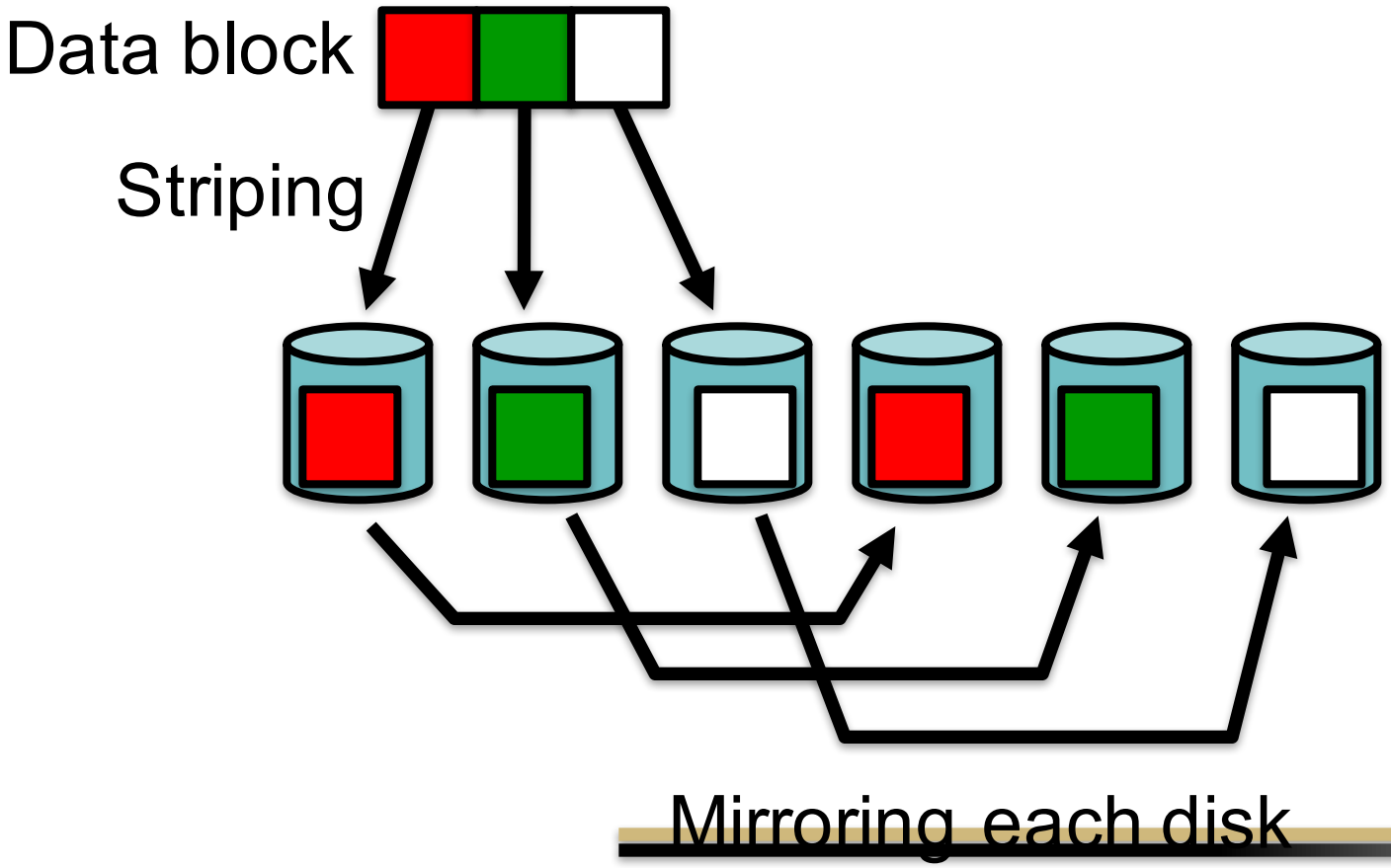


RAID 0+1 = RAID 0 data striping for performance + RAID 1 mirroring of stripes for redundancy

Note: if any one disk fails in the primary stripe, and any one disk fails in the second stripe, then the entire data block is lost

RAID1+0

– RAID1+0, aka “RAID 10”



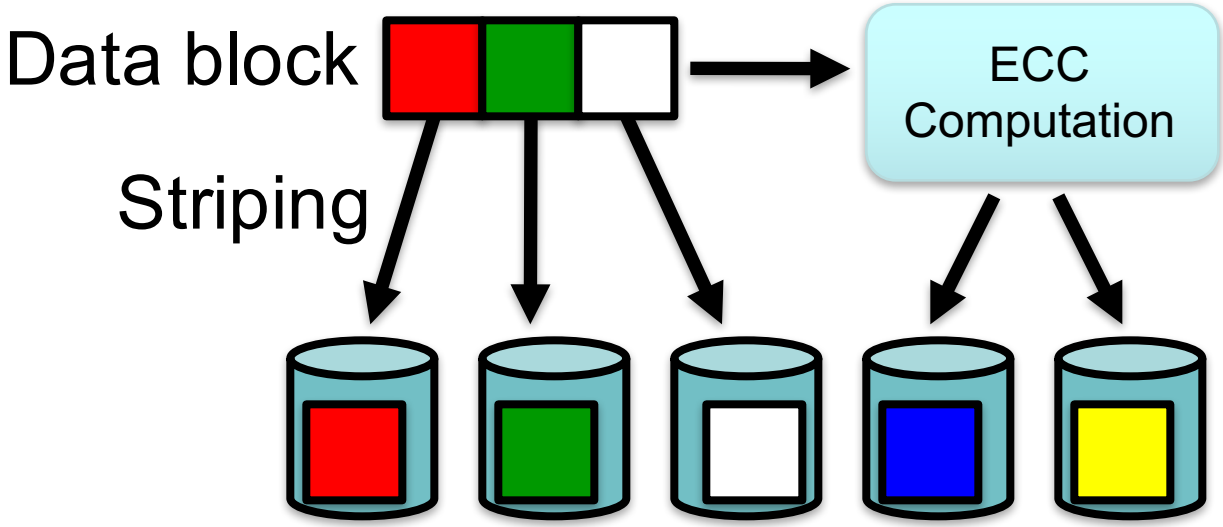
RAID 1+0 = mirror each disk then stripe.

Any two disks may fail, and the data can still be retrieved, unless the two disks mirror the same data.

Thus, RAID 1+0 is more robust than RAID 0+1.

RAID2

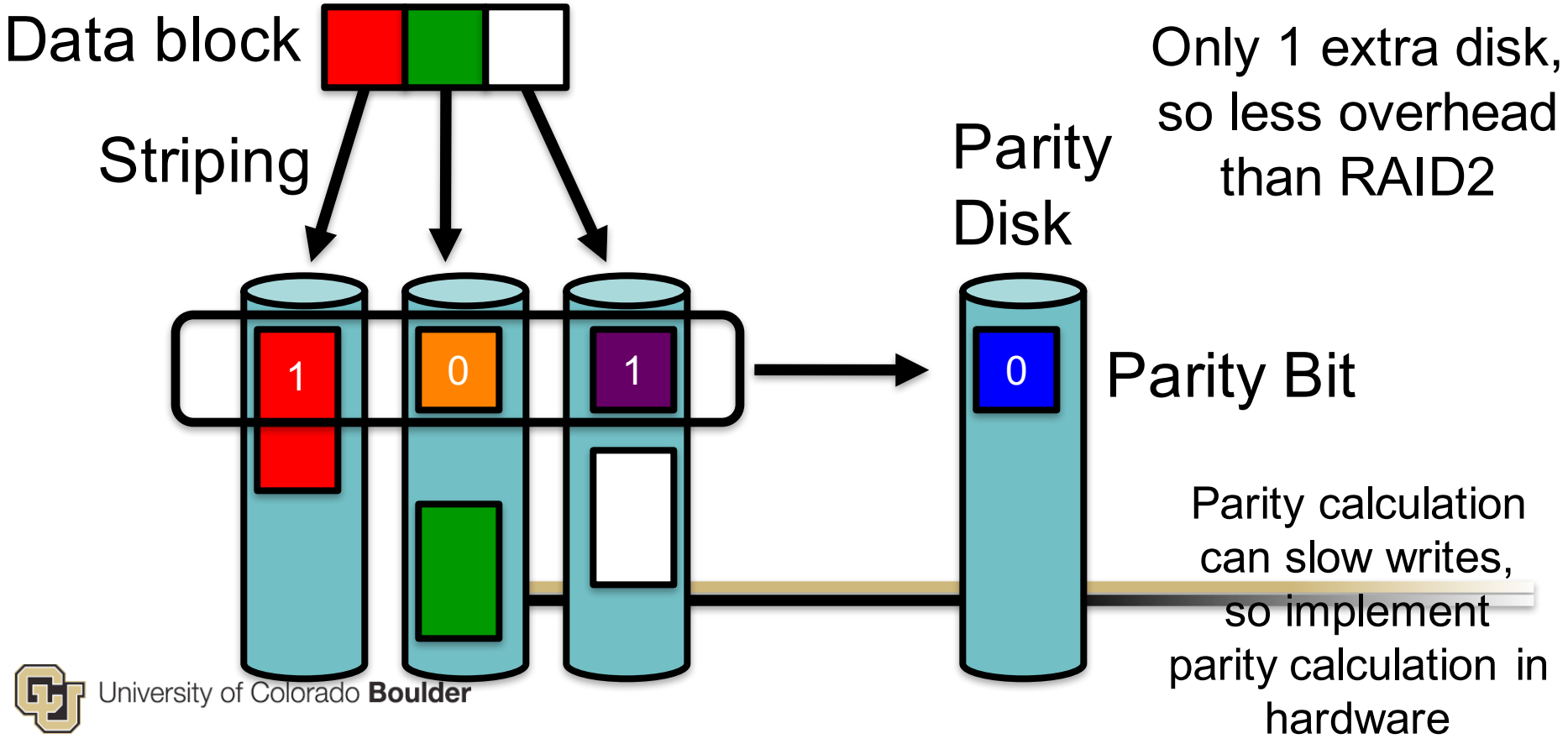
– RAID2 = put Error Correction Code (ECC) bits on other disks



Error correction codes are more compact than just copying the data, and provide strong statistical guarantees against error, e.g. a crashed disk's lost data can be corrected with the redundant data stored in the ECC disks

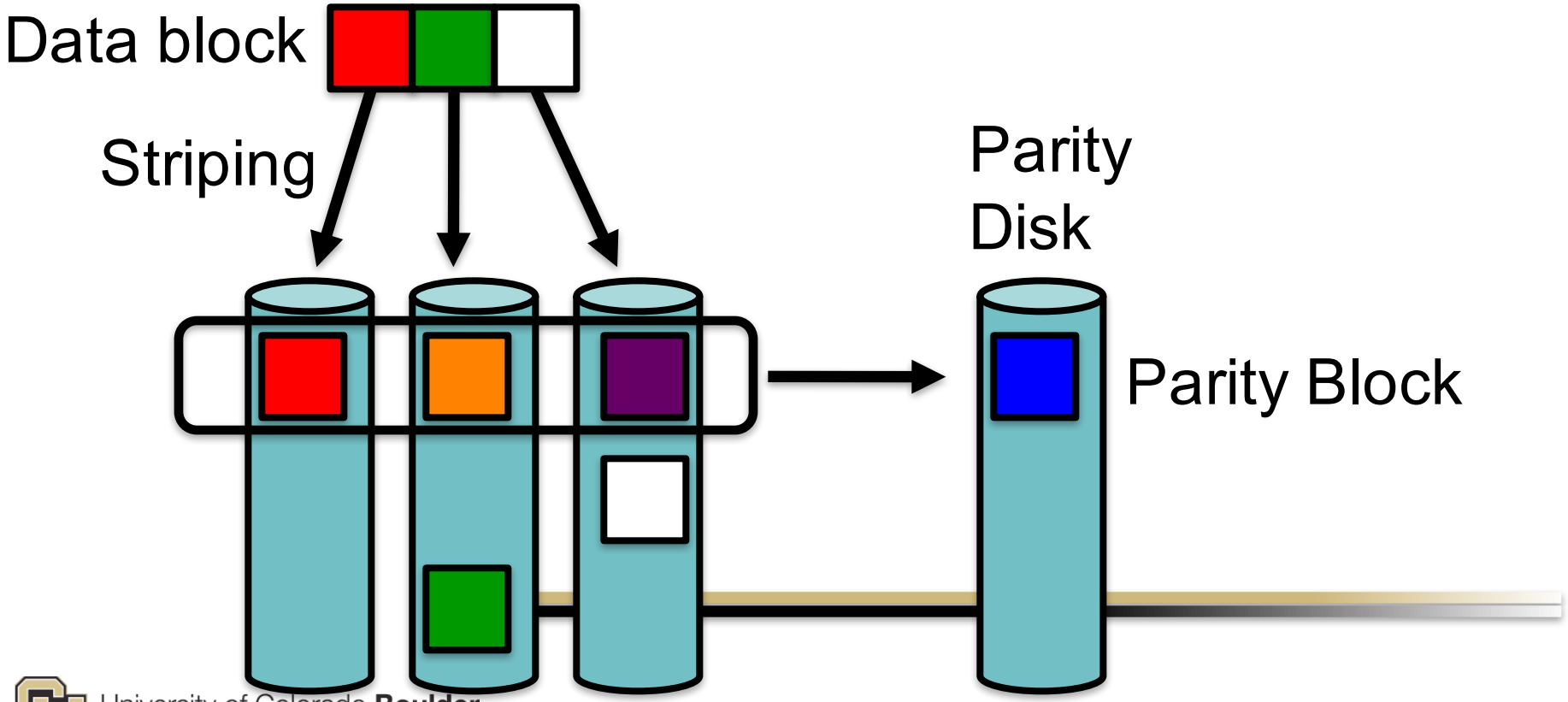
RAID3

- RAID3 = bit-interleaved parity: for each bit at the same location on different disks, compute a parity bit, that is stored on a parity disk



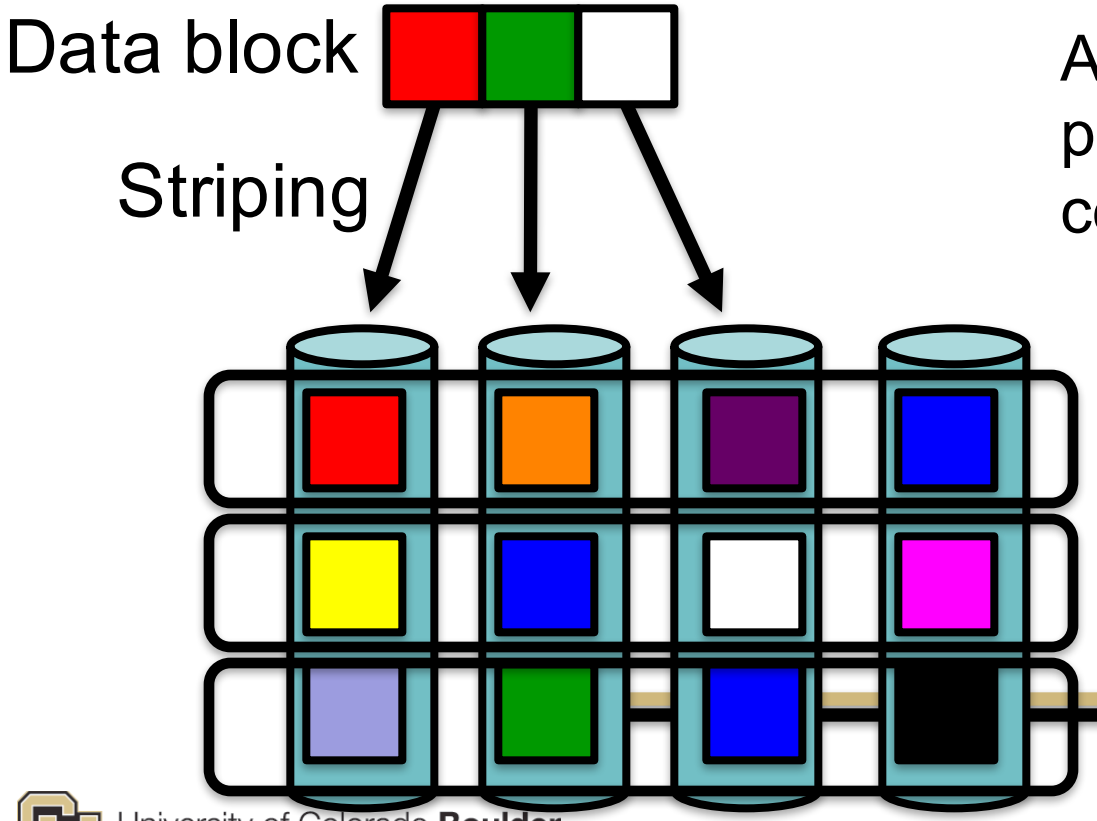
RAID4

- RAID4 = block-interleaved parity: for each block at the same location on different disks, compute a parity block, that is stored on a parity disk



RAID5

– RAID5 = block-interleaved *distributed* parity: spread the parity blocks to different disks, so every disk has both data and parity blocks



Avoids overuse of RAID4's parity disk, and is the most common parity RAID system

= Parity Blocks

RAID6 replaces parity blocks with stronger Reed-Solomon ECC for multiple disk failures

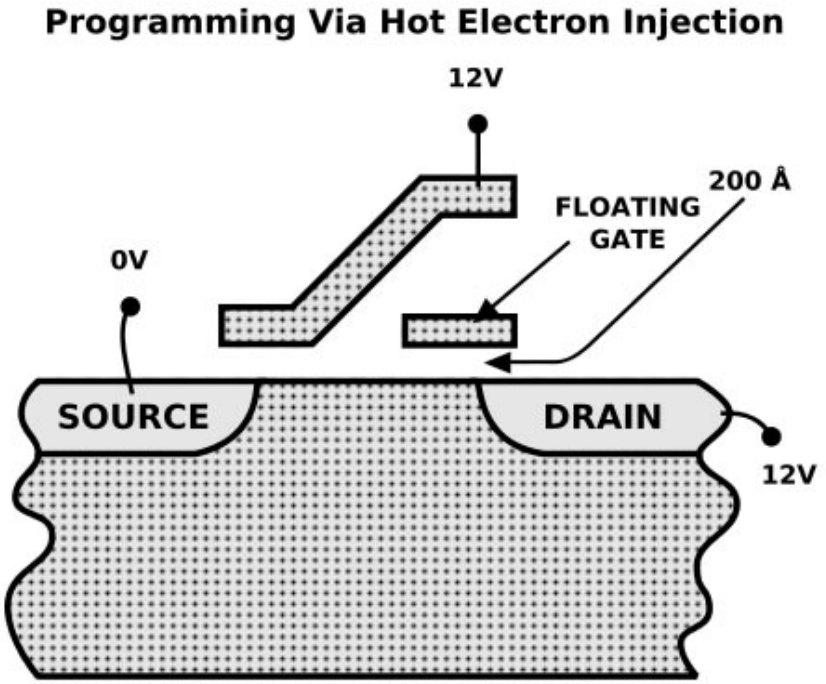
RAID

- Implementations of RAID
  1. Plug in disks to your computer bus and implement RAID management in software in OS kernel
     - In Linux, use the mdadm package to manage RAID arrays
  2. Plug in disks to your computer's I/O bus, and an intelligent hardware RAID controller recognizes them and integrates them into a RAID system automatically
  3. Plug in disks to a RAID array, which is a stand-alone hardware unit with a RAID controller that looks like one physical device, e.g. SCSI, to your computer bus
     - File system doesn't have to know about RAID to use and benefit from this RAID disk array

Supplementary Slides

Characteristics of Flash Memory

- Information is stored in floating-gate transistors
- Floating-gate transistors are insulated with 2 layers of silicone dioxide
- Above which is a control gate, below a transistor base
- Maintaining a link from floating gate to control gate, results in a value of 1 being stored
- Applying voltage to the floating gate allows the electrons to drain to the ground, changing the stored value to 0.
- Erasing 0 bits back to 1's requires an electric field to be applied to the entire chip or a sector of the chip.
  - Erasing is the only way to set a 0 back to a 1 in flash
- Thus it's easy to target a specific bit and write a '0' to it, but you can't target just that bit to rewrite a '0' back into a '1'. Instead, you have to erase an entire sector in order to rewrite that single '0' bit back into a '1'!



Flash File Systems

- Example: JFFS2
  - Changes to files and directories are logged in nodes:
    - Inodes = header with file metadata followed by payload
    - Dirent = directory entries
  - Nodes start out valid, and become obsolete when a newer version is written to the log
  - Not circular like JFFS. Instead, flash memory is divided into blocks = erase sector.
    - Blocks are filled by nodes, one at a time, in sequential log fashion
    - A clean block contains only valid nodes. A dirty block contains at least one obsolete node.
    - A free block contains no nodes.
  - A garbage collector runs in the background, and turns dirty blocks into free blocks.
    - Copies valid nodes to a clean block, and erases dirty block, making it into a clean block. This essentially frees the space occupied by obsolete nodes.
    - Occasionally, a garbage collector will consume clean blocks to improve wear leveling

Flash File Systems

- Other Flash File systems
  - YAFFS (Yet Another Flash File System) designed specifically for NAND flash
  - LogFS designed for large flash devices
- Hopefully, one day new solid state technology will replace Flash…
  - With better rewrite latency
  - And better/indefinite write lifetime