Advanced Storage

COMS W4118

Prof. Kaustubh R. Joshi

krj@cs.columbia.edu

http://www.cs.columbia.edu/~krj/os

References: Operating Systems Concepts (9e), Linux Kernel Development, previous W4118s

Outline

- Disk reliability and RAID
- Solid state storage

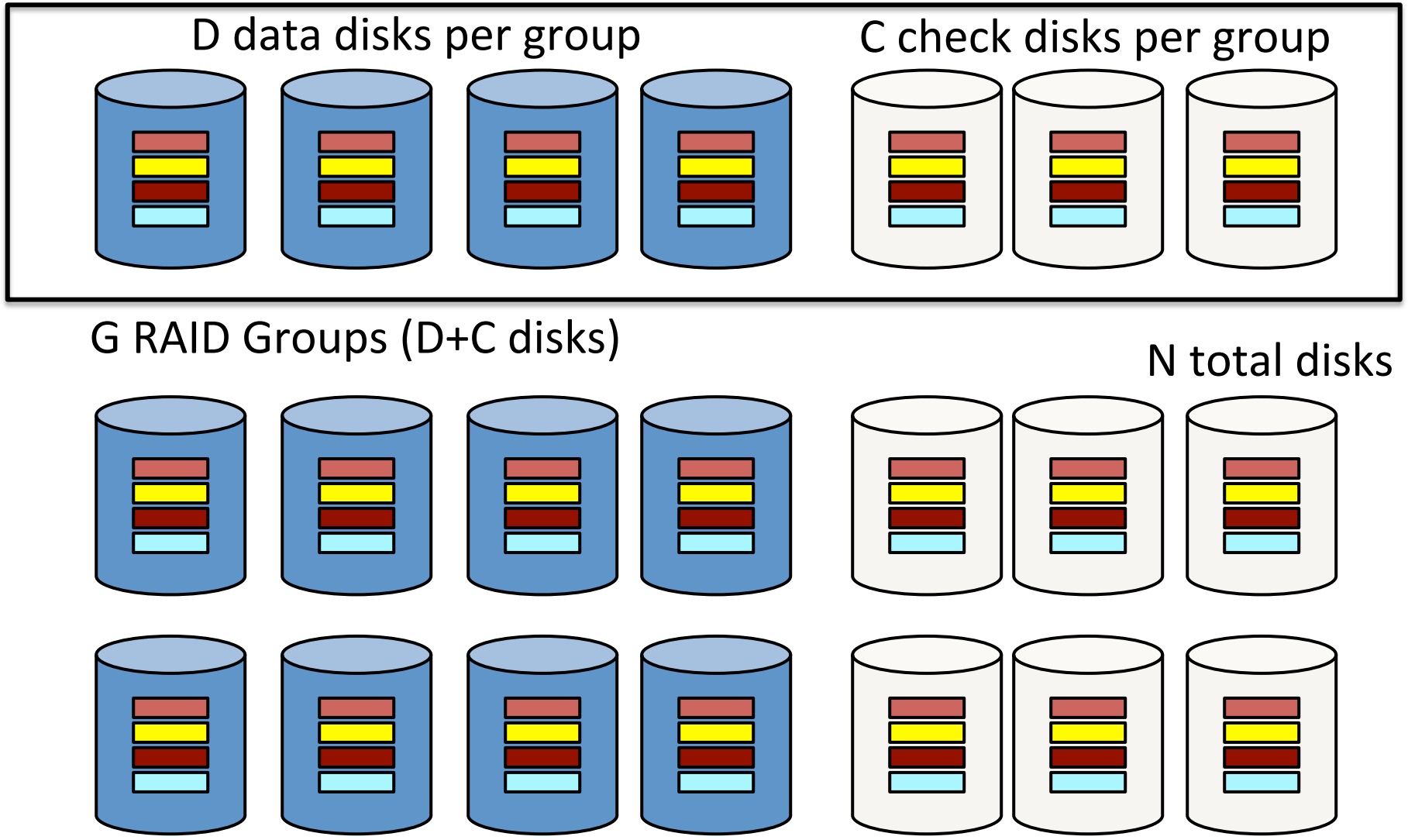
RAID motivation

- Performance
  - Disks are slow compared to CPU
  - Disk speed improves slowly compared to CPU
- Reliability
  - In single disk systems, one disk failure ➔ data loss
- Cost
  - A single fast, reliable disk is expensive

RAID idea

- RAID idea: use redundancy to improve performance and reliability
  - Redundant array of cheap disks as one storage unit
  - Fast: simultaneous read and write disks in the array
  - Reliable: use parity to detect and correct errors
- RAID can have different redundancy levels, achieving different performance and reliability
  - Seven different *RAID levels* (0-6)

RAID Organization



Evaluating RAID

- Cost: check disk capacity / total capacity
  - *Storage utilization*: data capacity / total capacity
- Reliability
  - Tolerance of disk failures
- Performance
  - (Large) sequential read, write, read-modify-write
  - (Small) random read, write, read-modify-write
  - Speedup over a single disk

Computing cost

- D = number of data disks in a RAID group
- C = number of check disks in a RAID group

- Cost = C/(D+C)

Computing Reliability: Assumptions

- Independent Failures
  - Failures don't happen together in time
  - Failures don't happen together in space
  - May be violated by:
    - Disks from same batch may have similar defects
    - Disks with same workload may fail together
    - Disks connected to same power supply, etc.
- Fail Stop behavior
  - Disk knows when there is a failure
  - E.g., can't read sector
  - Violated by:
    - Silent failures and bitrot
    - Read garbage
  - Logical model of behavior.
  - Can be emulated by proper error checking codes

Computing RAID Reliability

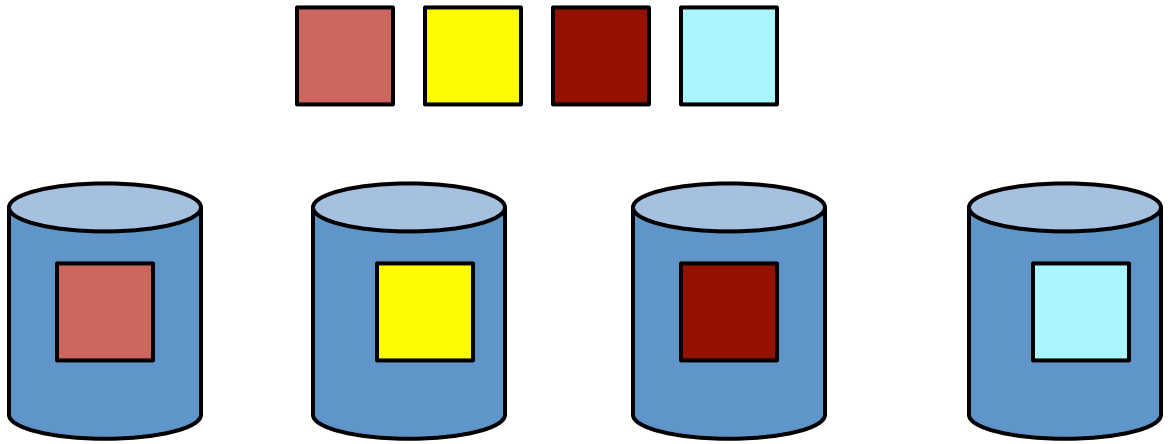
- N = total number of disks
- D = number of data disks in a RAID group
- C = number of check/parity disks in a RAID group
- MTTF(disk) = mean time to failure for a disk
- MTTR = mean time to repair for a failed disk
- MTTF(group) = mean time to two failed disks before first gets repaired in one group
- MTTF(raid) = mean time to failure over entire array
- MTTF(raid) = MTTF(group) / Num. groups
- MTTF(group) = ?

RAID Reliability (Cont'd)

- Assume single-error tolerance in one group
  - If another error comes before repair, group fails)
- MTTF(group) = MTTF(1 disk) / Prob[Another failure occurs before MTTR]
  - If Prob[…] ≈ 1, MTTF(group) same as MTTF(1 disk). No benefit of RAID
  - If Prob[…] ≈ 0, MTTF(group) approaches ∞.
- MTTF(1 disk) = MTTF(disk)/(D+C)
- MTTF(another disk) = MTTF(disk)/(D+C-1)
- Prob[Another failure occurs before MTTR] = MTTR/(MTTF(disk)/(D+C-1))
- MTTF(group) = MTTF(1 disk)/Prob[Another failure occurs before MTTR] = $(MTTF(disk))^2/((D+C)*(D+C-1)*MTTR)$
- Num. groups G = N / (D + C)
- MTTF(raid) = MTTF(group) / G = MTTF(group)/(N/(D+C)) = $(MTTF(disk))^2$/ (N*(D+C-1)*MTTR)

RAID 0: non-redundant striping

- Structure
  - Data striped across all disks in an array
  - No parity (C=0)
- Advantages:
  - Good performance: with N disks, roughly N times speedup
- Disadvantages:
  - Poor reliability: one disk failure ➔ data loss.
  - MTTF(raid)=MTTF(disk)/N



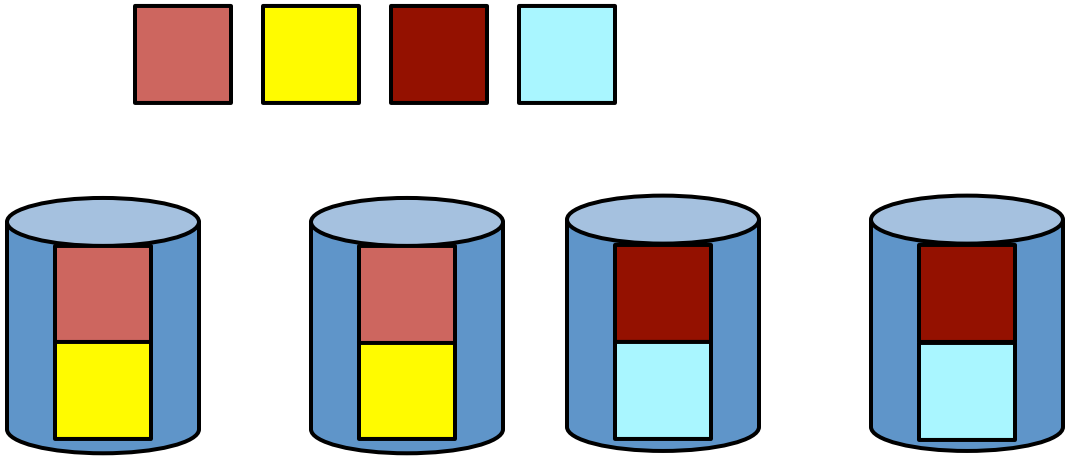
RAID 0 performance

- Large read of 100 blocks.
  - One disk: 100 * t,
  - Raid0: 100/N * t * S
  - S: slowdown. Need to wait for slowest disk to complete before return.

- Performance:
  - Large read: N/S
  - Large write: N/S
  - Large R-M-W: N/S
  - Small read: N
  - Small write: N
  - Small R-M-W: N

RAID 1: mirroring

- Structure
  - Keep a *mirrored* (shadow) copy of data (D=1, C=1)
- Advantages
  - Good reliability: one disk failure within each mirrored disk group OK
  - Good read performance
- Disadvantage
  - High cost: one data disk requires one parity disk



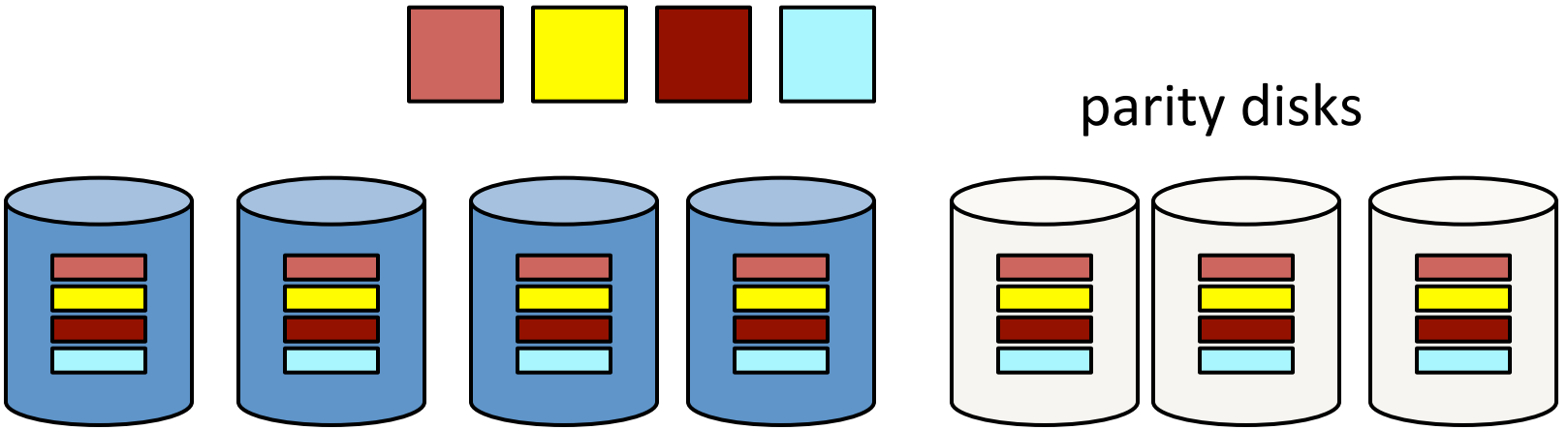
RAID 1 performance

- Cost = C/(D+C) = 1/(1+1) = 50%
- MTTF(raid) = MTTF(disk)^2/(N*MTTR)

- Performance
  - Large read: N/S
  - Large write: N/2S
  - Large R-M-W: 2N/3S
    - X sectors, 2X events (X reads, X writes)
    - Speedup (w.r.t. to 1 disk) = 2X / (X/(N/S) + 2*X/(N/S)) = 2N/3S
  - Small read: N (no S here)
  - Small write: N/2
  - Small R-M-W: 2N/3

RAID 2: error-correction parity

- Structure
  - A data sector striped across data disks
  - Compute *error-correcting parity* and store in parity disks
- Advantages
  - Good reliability with higher storage utilization than mirroring
- Disadvantages
  - Unnecessary cost: disk can already detect failure
  - Poor random performance

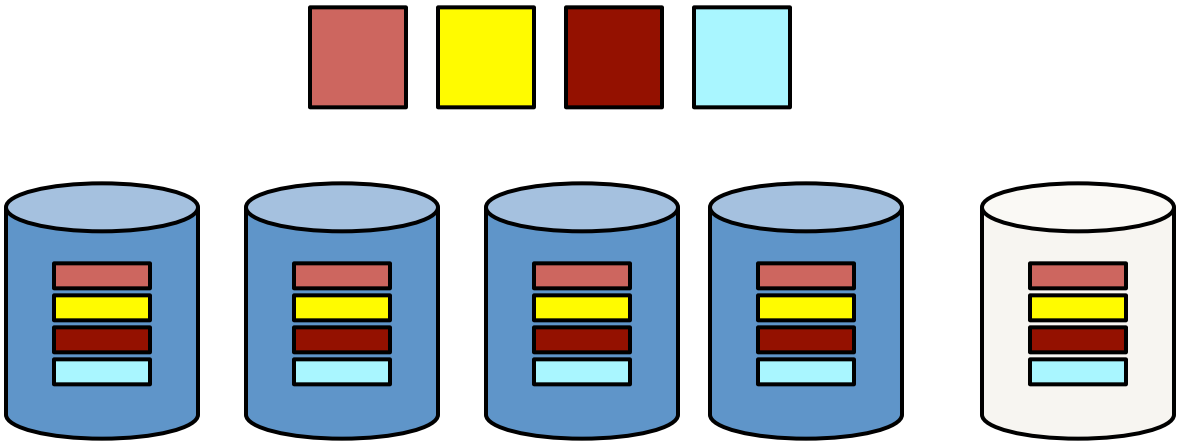


RAID 2

- Error correction parity used in memory or networking. Can't detect which bit has error in packet or memory bank.
  - E.g. in simple parity code, 0010 (parity 1) -> 1010 (parity 1). Parity mismatch, but which bit is bad?
  - Hamming code: 0010 (011) -> 1010 (011). Identifies bit 0 as error. 1
- Hamming codes
  - http://en.wikipedia.org/wiki/Hamming(7,4)
  - Cost: 14 disks needs 4 check disks
- Reliability: MTTR(raid) = MTTF(disk)^2/(N*(D+C-1)*MTTR)
- Suffers from unnecessary cost
  - Is a hamming code really needed? Can we tell which sector is bad?
  - Small operation: E.g., to read a data sector, must read a sector from each disk. Because disks are accessed by sectors, not by bits as memory

RAID 3: bit-interleaved parity

- Structure
  - Single parity disk (XOR of each stripe of a data sector) (C=1)
- Advantages
  - Same reliability with one disk failure as RAID2 since disk controller can determine what disk fails
  - Higher storage utilization
- Disadvantages
  - Poor random performance

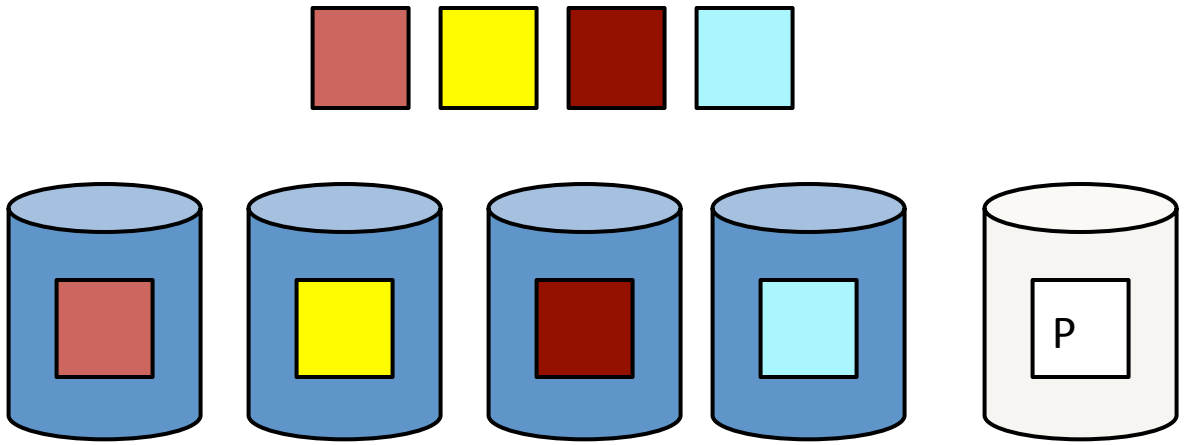


RAID 3

- Cost: 1/(D+1)
- Reliability: MTTF(raid) = MTTF(disk)^2/ (N*D*MTTR)
- Still the same problem: small access touches all disks

RAID 4: block-interleaved parity

- Structure
  - A set of data sectors (*parity group*) striped across data disks (C=1)
- Advantages
  - Same reliability as RAID3
  - Good random read performance
- Disadvantages
  - Poor random write and read-modify-write performance



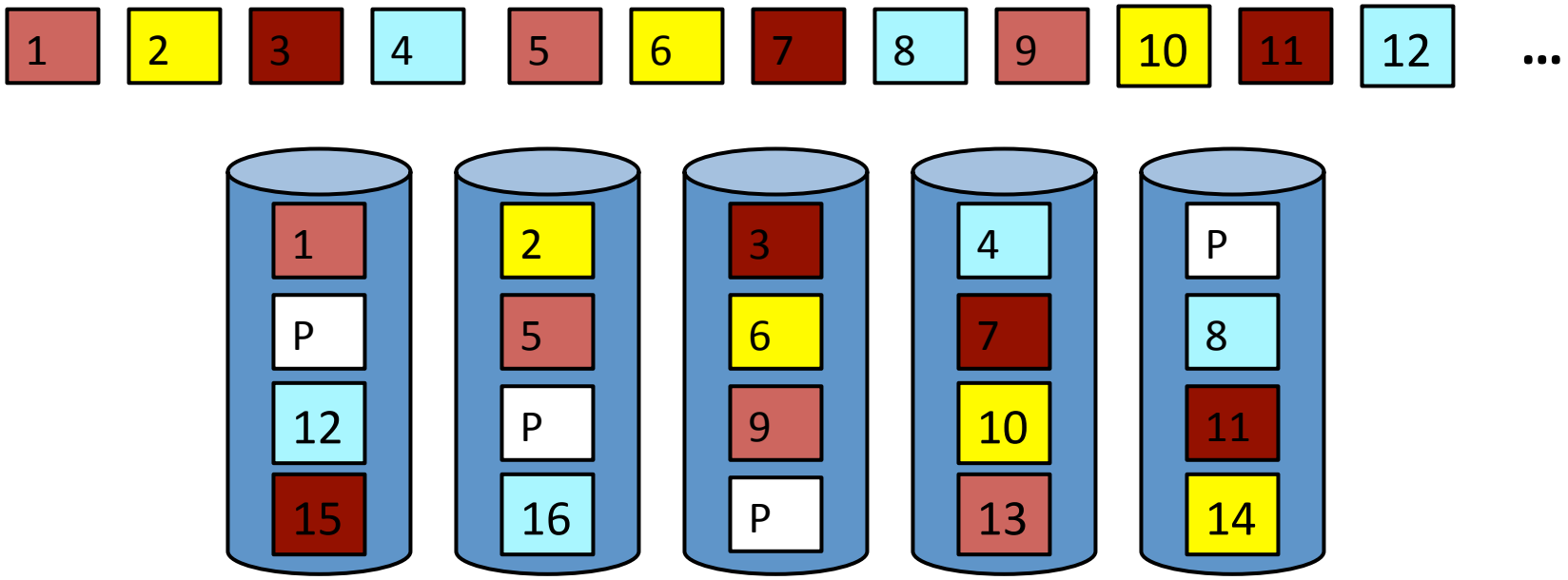
RAID 4 performance

- One parity disk (XOR of data sectors)
  - Write data disk + parity disk
  - To update parity, don't have to read all disk sectors, Parity = oldParity xor (changed bits) = oldParity xor newData xor oldData
- Cost: 1/(D+1), Reliability: MTTF(raid) = MTTF(disk)^2/(N*D*MTTR)
- Number of groups G = N/(D+1) = number of check disks
- Large read of 100 blocks. One disk: 100 * t, Raid4: 100/(N-N/(D+1)) * t * S
- Large read: (N-G)/S
- Large write: (N-G)/S
- Large R-M-W: (N-G)/S
- Small read: N-G
- Small write: ½*G (for each block, need a read and a write to parity disk)
  - RAID: X sectors. X/((X/1) + (X/1)) = ½
- Small R-M-W: 1*G
  - RAID: X sectors. 2X/((X/1) + (X/1)) = 1

RAID 5: block-interleaved distributed parity

- Structure
  - Relieves parity disk bottleneck
  - Parity sectors distributed across all disks (C=1)

- Advantages
  - Good performance



RAID 5 performance

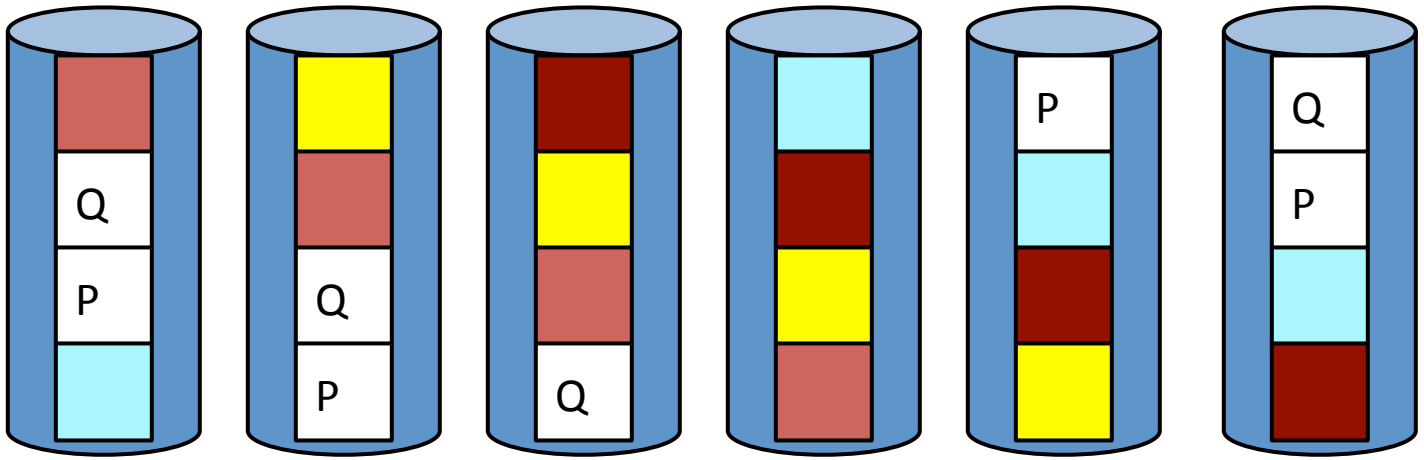
- Same as RAID4 except no single parity disk
  - Good small write and read-modify-write performance

- Large read of 100 blocks. one disk: 100 * t, Raid5 100/(N-G) * t * S

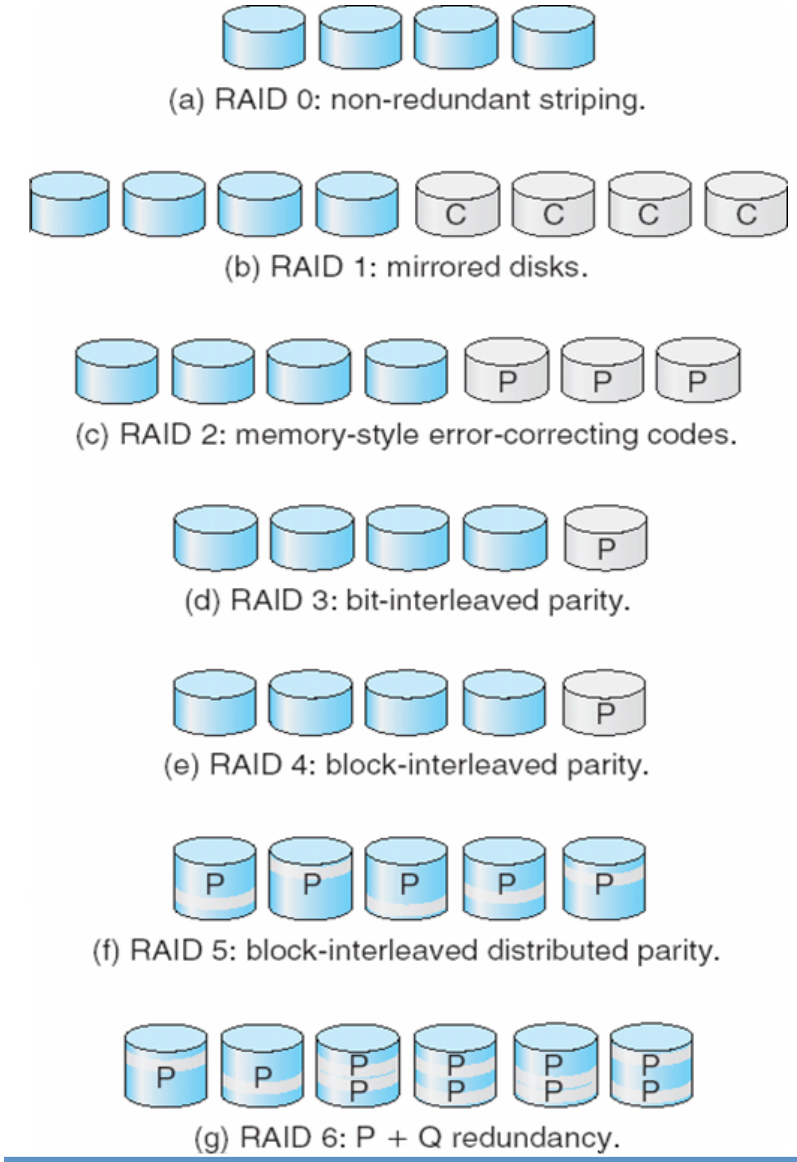
- Large read: ? If 5 disks, block 1-4 are on different disks, block 5 may be on disk 1-4 as disk 5 stores parity. So speedup is not N/S
- Large write: ?
- Large R-M-W: ?
- Small read: ?
- Small write: ?
- Small R-M-W: ?

RAID6: P+Q redundancy

- Structure
  - Same as RAID 5 except using two parity sectors per parity group
- Advantages
  - Can tolerate two disk failures



Summary of RAID Levels



- Practically used: RAID 0, RAID 1, RAID 5, RAID 6
- RAID0: best performance, no cost, no reliability
- RAID1: good performance, better small write performance than RAID5, high cost, better reliability than RAID5
- RAID5: good performance, better large write performance than RAID1, low cost, good reliability
- RAID6: good performance, low cost, better reliability than RAID5

Outline

- Disk reliability and RAID

- Solid state storage

Flash and SSDs

- Solid state storage
  - Use silicon transistors to store data rather than spinning magnetic platters
  - Fundamentally different characteristics than disks
  - Increasing popularity in mobile devices, large server farms
- Pros
  - No moving parts - robust to mechanical failure
  - No mechanical limitations: high throughput, random access
  - Less energy use, less heat
  - High density
- Cons
  - Expensive
  - Unfavorable reliability characteristics over time (bit rot)
  - Limitations on read-modify-write cycles
  - Complex to use

Basic Idea

- Use silicon devices based on MOSFETs
  - Metal Oxide Field Effect Transistor
  - Also used in DRAM
  - Each cell contains a single MOSFET with an additional "floating gate"

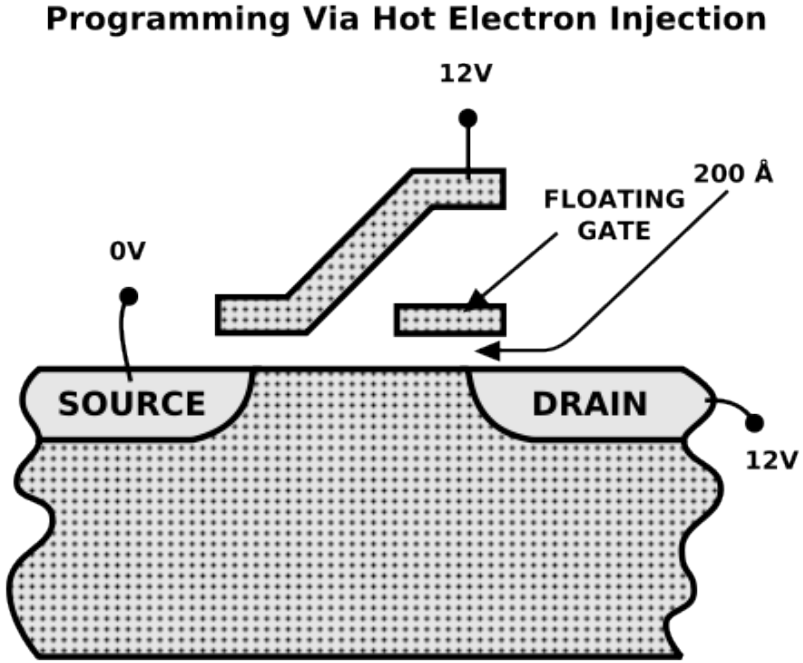
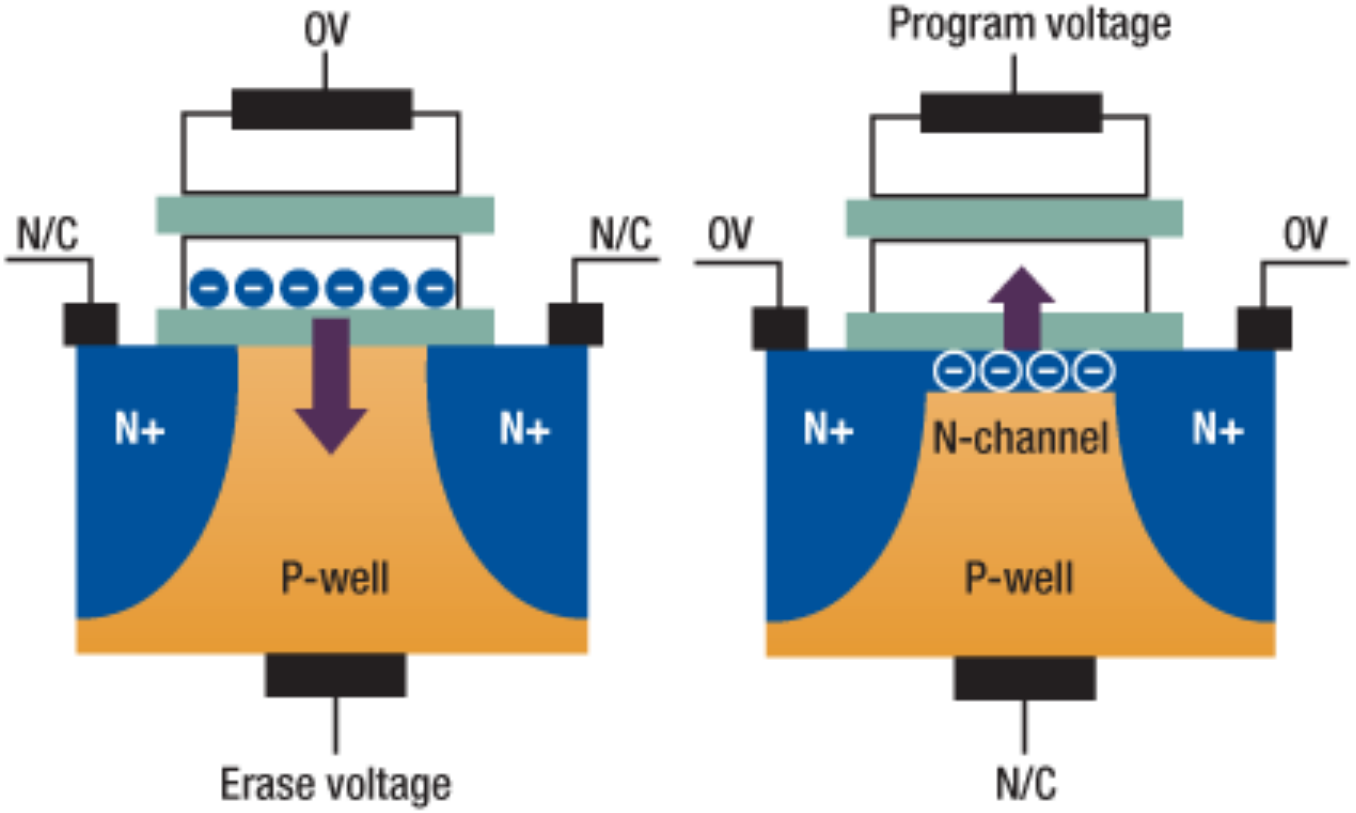


Image source: wikipedia

Programming a Flash Cell

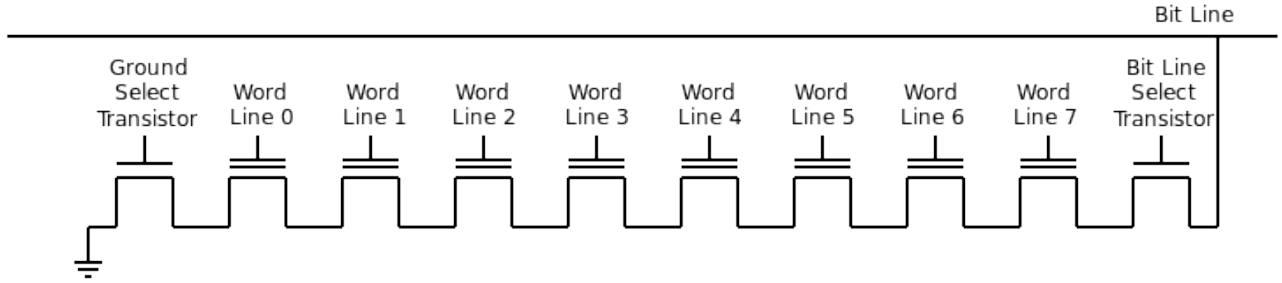
- Two basic operations: erase (reset) and program
  - Erase clears charge on floating gate. Allows channel to conduct, setting bit to "1"
  - Program forces charge onto floating gate (via tunneling/hot electron injection), blocking the channel, and setting bit to "0"



Source: http://www.electroiq.com/articles/sst/2011/05/solid-state-drives.html

NAND Flash

- Two basic types
  - Differ in how cells are connected and accessed
  - NOR: bit level addressability, lower density, expensive
  - NAND: "block" level addressability, higher density, cheap



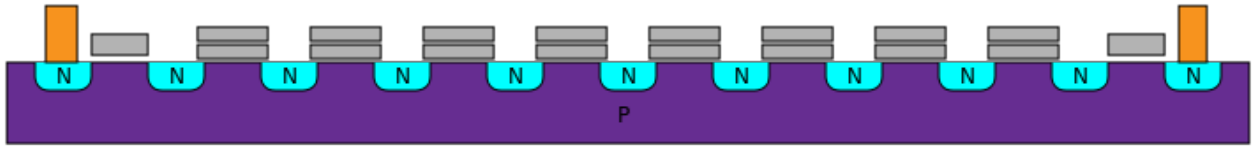
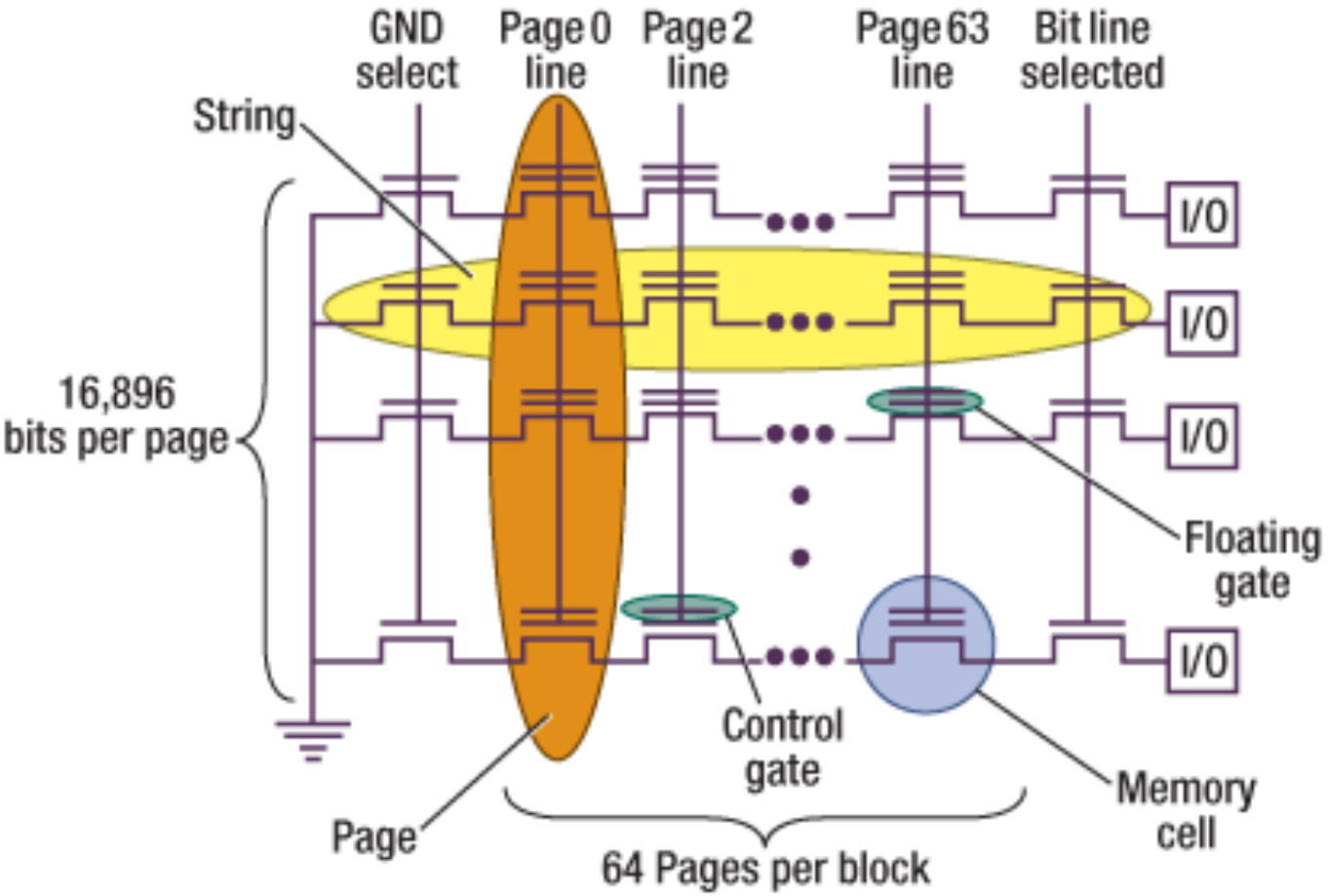


Image source: wikipedia

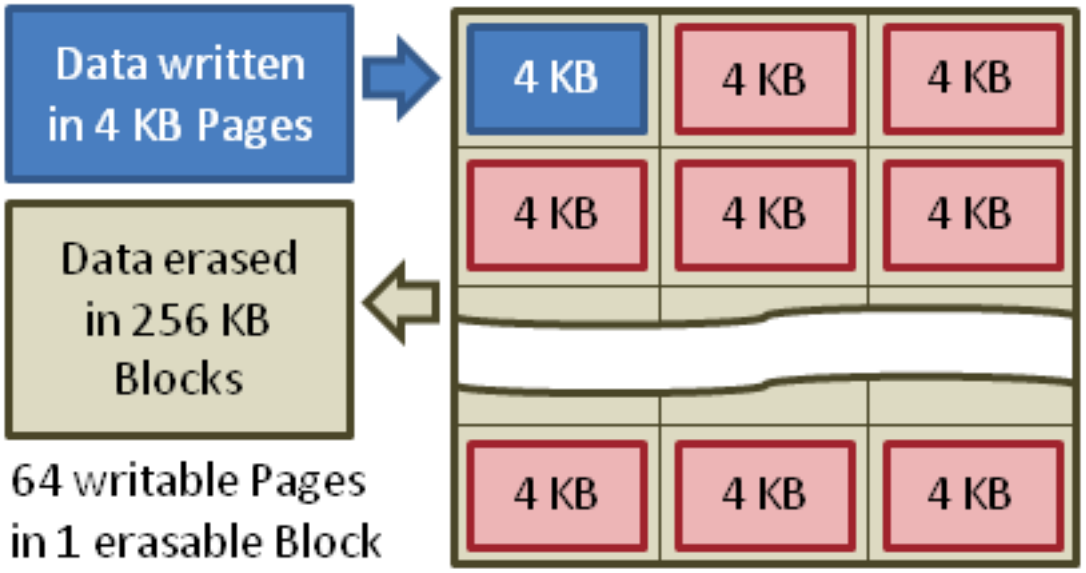
- NAND Flash
  - All flash cells in a "row" accessed through the one sense line
  - No bit addressability read out a page at a time

NAND Flash Structure



Source: http://www.electroiq.com/articles/sst/2011/05/solid-state-drives.html

NAND Flash Programming Model



Typical NAND Flash Pages and Blocks

- Can read data in page level units. Fast: 10 microsec.
- Can program data in page level units. Fast: 10-100 microsec
- Can only erase entire block. Slow 1-10 msec

Reliability Characteristics

- The process of reading/writing from a cell impacts it ability to retain data
- P/E Cycles
  - High voltage, charge moves into/out of floating gate
  - Some charge gets stuck in oxide layer
  - Over time, cell gets "stuck" and cant be programmed
- Read/write disturb
  - Occurs because multiple cells are connected in series
  - Read/program voltages on a cell can cause leakage in other cells, causing their values to "flip"
  - Can result in "bitrot"

Flash Reliability

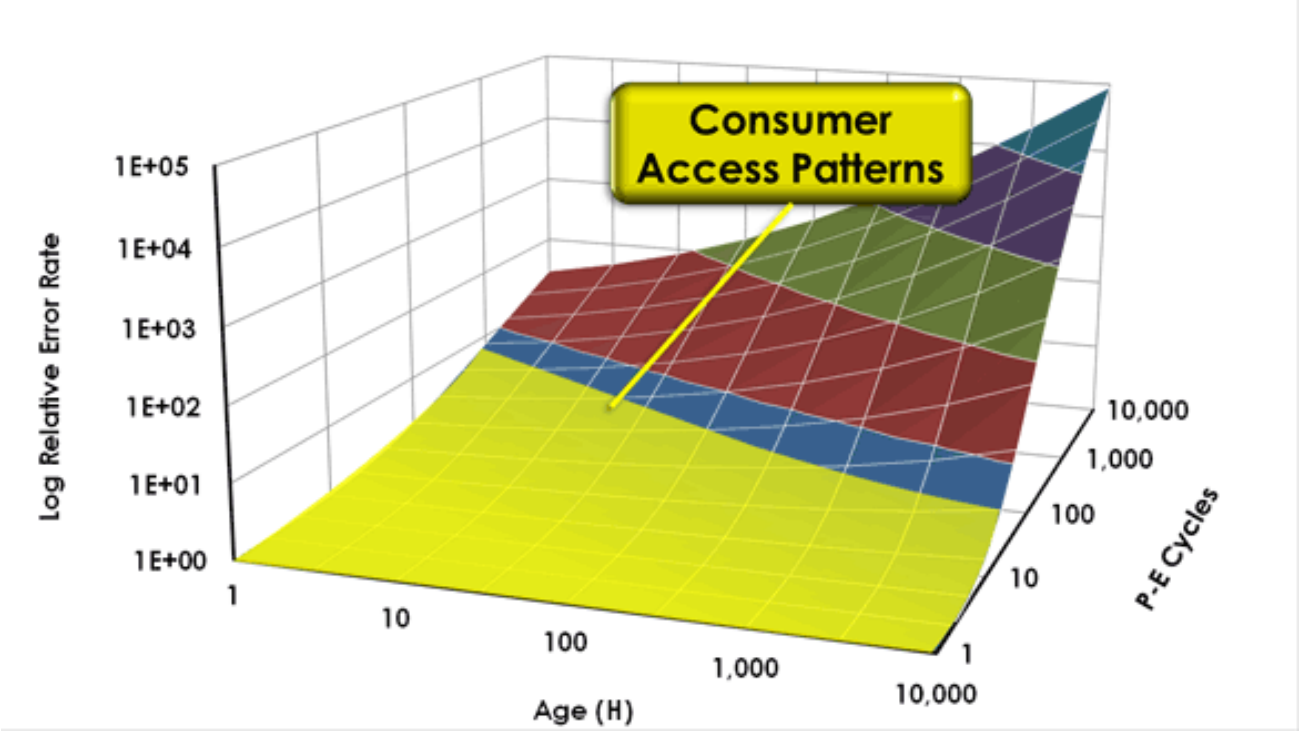


Figure is for illustrative purposes only

- BER: bit error rate
- RBER: raw bit error rate (can be reduced through error checking codes)
- UBER: uncorrected bit error rate
- P/E cycles: number of program/erase cycles a cell is subjected to
- Typical SLC 100k P/E cycles, MLC < 10k P/E cycles for HDD-like error rates

Figure source: http://drhetzler.com/smorgastor/2012/09/03/series-sdd-5-the-error-rate-surface-for-mlc-nand-flash/

Implications of Flash Storage

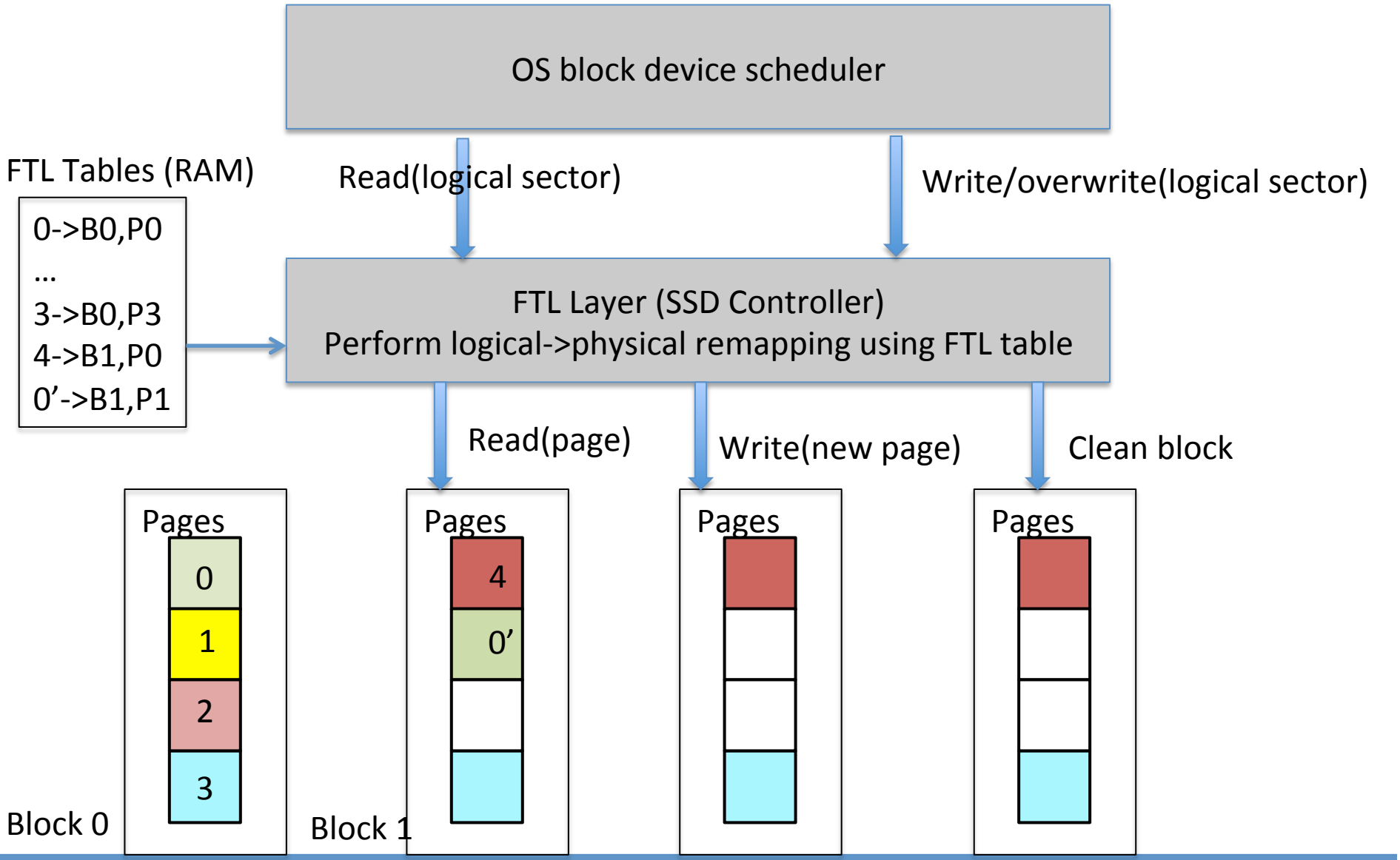
- Block level erase
  - Erasing takes more time than reading/writing
  - Can only do block at a time
- Wear leveling
  - Cell reliability degrades with P/E cycles
  - Distribute P/E cycles equally between cells
- Random access
  - No concept of seeks
  - No need for scheduling

Who deals with Flash quirks?

- OS Filesystem
  - Log structured handles block level erase
  - Implement wear leveling through log cleaning
  - E.g., Linux JFFS/JFFS2, YAFFS (2002) for NAND flash, Android YAFFS2, Samsung F2FS (2012)

- On disk controller
  - Block level erase handled through FTL (flash translation layer)
  - FTL maps logical block (LBA) to physical block
  - Modify cycle allocates new phy block and changes FTL mapping
  - Garbage collection pass erases partially used blocks
  - More common for high end SSD drives
  - Normal block device interface exported to OS

FTL Layer


OS block device scheduler
FTL Tables (RAM)
0->B0,P0
…
3->B0,P3
4->B1,P0
0'->B1,P1
Read(logical sector)
Write/overwrite(logical sector)
FTL Layer (SSD Controller)
Perform logical->physical remapping using FTL table
Read(page)
Write(new page)
Clean block
Pages
0
1
2
3
Block 0
Pages
4
0'
Block 1
Pages
Pages

Wear Leveling

- No wear leveling
- Dynamic wear leveling
  - Always write to new page
  - Garbage collect old blocks (compare to LFS)
  - Infrequently changing blocks left untouched
- Static wear leveling
  - Similar to dynamic wear leveling, but
  - Also periodically move unmodified blocks
  - More overhead, but better leveling

Write Amplification

- Write amplification = Data written to flash/Data written by OS
- Factors that impact write amplification
  - Garbage collection (increases WA during cleaning)
  - Over-provisioning (less cleaning, decrease WA)
  - TRIM (less cleaning, decrease WA)
  - Free user space (less cleaning, decrease WA)
  - Wear leveling (more rewrites, increase WA)
  - Separating static and dynamic data (decrease WA)
  - Sequential writes (low WA)
  - Random writes (more cleaning, more WA)

TRIM

- SSD cleaning overhead can be significantly reduced if unused blocks can be used for read-modify-write cycles
- But which data unused?
  - Deleting a file from a FS doesn't clear block. Only marks inode as unused
  - Eventually, SSD controller thinks whole disk is full, and every write needs a corresponding cleaning operation
  - Excessive overhead
- TRIM command
  - OS informs SSD that a particular block not being used
  - Relatively recent (e.g., OS X supports since 2011)
  - Still fairly expensive (hundreds of msec)
  - Active debate on how OS should use