ECE590-03
Enterprise Storage Architecture

Fall 2016

RAID

Tyler Bletsch
Duke University

Slides include material from Vince Freeh (NCSU)

A case for redundant arrays of inexpensive disks

- Circa late 80s..
- MIPS = $2^{year-1984}$ Joy's Law
- There seems to be plenty of main-memory available (multi mega-bytes per machine).
- To achieve a balanced system

  Secondary storage system has to match the above developments.
- Caches
  - provide a bridge between memory levels
- SLED (Single Large Expensive Disk) had shown modest improvement…
  - Seek times improved from 20ms in 1980 to 10ms in 1994
  - Rotational speeds increased from 3600/minute in 1980 to 7200 in 1994

Core of the proposal

- Build I/O systems as ARRAYS of inexpensive disks.
  - Stripe data across multiple disks and access them in parallel to achieve both higher data transfer rates on large data accesses and…
  - higher I/O rates on small data accesses

- Idea not entirely new…
  - Prior very similar proposals [Kim 86, Livny et al, 87, Salem & Garcia-Molina 87]
- 75 inexpensive disks versus one IBM 3380
  - Potentially 12 times the I/O bandwidth
  - Lower power consumption
  - Lower cost

Original Motivation

- Replacing large and expensive mainframe hard drives (IBM 3310) by several cheaper Winchester disk drives
- Will work but introduce a data reliability problem:
  - Assume MTTF of a disk drive is 30,000 hours
  - MTTF for a set of *n* drives is 30,000/*n*
    - n = 10 means MTTF of 3,000 hours

Data sheet

- Comparison of two disk of the era
  - Large differences in capacity & cost
  - Small differences in I/O's & BW
- Today

  Difference are different ;)

IBM 3380	Conner CP 3100
14" in diameter	3.5" in diameter
7,500 Megabytes	100 Megabytes
$135,000	$1,000
120-200 IO's/sec	20-30 IO's/sec
3 MB/sec	1MB/sec
24 cube feet	.03 cube feet

Reliabilty

- MTTF: mean time to failure
- MTTF for a single disk unit is long..
  - For IBM 3380 is estimated to be 30,000 hours (> 3 years)
  - For CP 3100 is around 30,000 hours as well..
- For an array of 100 CP3100 disk the…

  MTTF = MTTF_for_single_disk / Number_of_disk_in_the_Array

  I.e., 30,000 / 100 = 30 hours!!! (or once a day!)
- That means that we are going to have failures very frequently

A better solution

- Idea: make use of extra disks for reliability!
- Core contribution of paper (in comparison with prior work):
  - Provide a full taxonomy (RAID-levels)
  - Qualitatively outlines the workloads that are "good" for every classification
  - RAID ideas are applicable to both hardware and software implementations

Basis for RAID

- Two RAID aspects taken into consideration:
  - **Data striping** : leads to enhanced bandwidth
  - **Data redundancy** : leads to enhanced reliability
    - Mirroring, parity, or other encodings

Data striping

- Data striping:
  - Distributes data transparently over multiple disks
  - Appears as a single fast large disk
  - Allows multiple I/Os to happen in parallel.
- Granularity of data interleaving
  - Fine grained (byte or bit interleaved)
    - Relatively small units; High transfer rates
    - I/O requests access all of disks in the disk array.
    - Only one logical I/O request at a time
    - All disks must waste time positioning for each request: bad!
  - Coarse grained (block-interleaved)
    - Relatively large units
    - Small I/O requests only need a small number of disks
    - Large requests can access all disks in the array

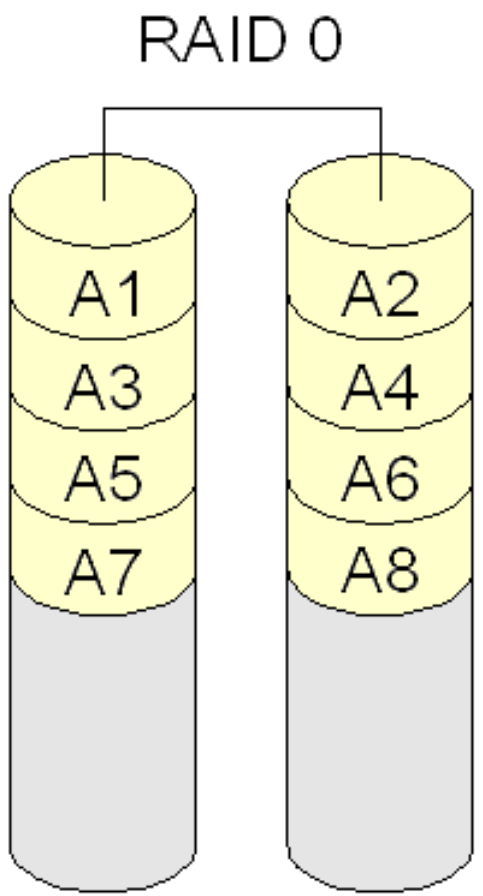
Data redundancy

- Method for computing redundant information
  - Parity (3,4,5), Hamming (2) or Reed-Solomon (6) codes
- Method for distributing redundant information
  - Concentrate on small number of disks vs. distribute uniformly across all disks
  - Uniform distribution avoids hot spots and other load balancing issues.

- Variables I'll use:
  - N = total number of drives in array
  - D = number of data drives in array
  - C = number of "check" drives in array (overhead)
  - N = D+C
  - Overhead = C/N
    ("how many more drives do we need for the redundancy?")

RAID 0

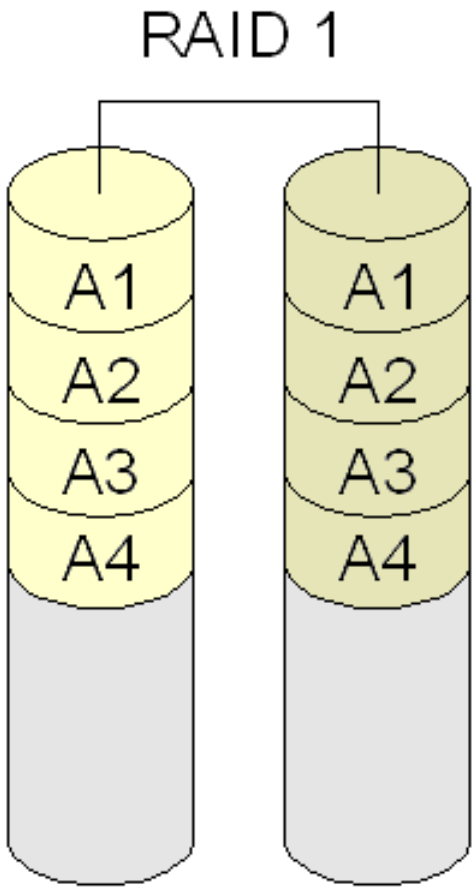
- Non-redundant
  - Stripe across multiple disks
  - Increases throughput
- Advantages
  - High transfer
  - Cost
- Disadvantage
  - No redundancy
  - Higher failure rate



RAID 0 ("Striping")
Disks: N≥2, typ. N in {2..4}. C=0.
SeqRead: N
SeqWrite: N
RandRead: N
RandWrite: N
Max fails w/o loss: 0
Overhead: 0

RAID 1

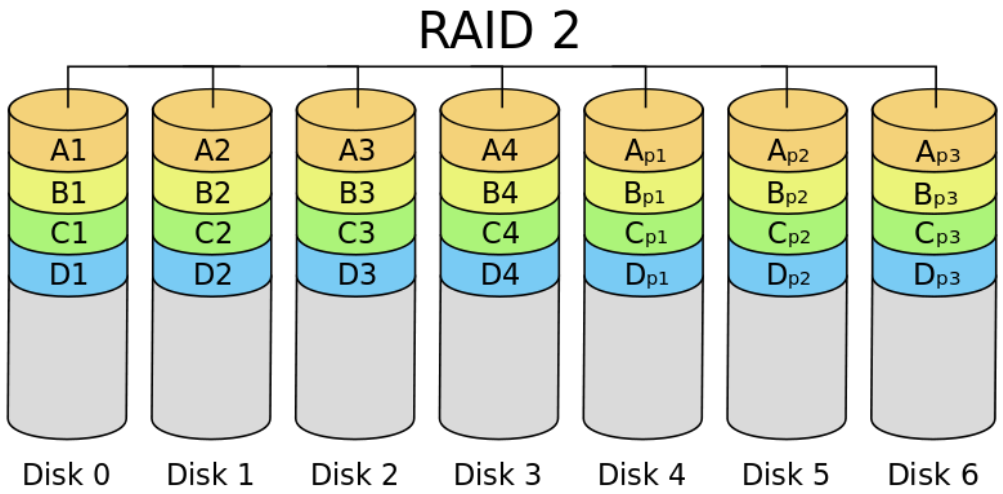
- Mirroring
  - Two copies of each disk block
- Advantage
  - Simple to implement
  - Fault-tolerant
- Disadvantage
  - Requires twice the disk capacity



RAID 1 ("Mirroring")
Disks: N≥2, typ. N=2. C=1.
SeqRead: N
SeqWrite: 1
RandRead: N
RandWrite: 1
Max fails w/o loss: N-1
Overhead: (N-1)/N *(typ. 50%)*

RAID 2

- Instead of duplicating the data blocks we use an **error correction** code (derived from ECC RAM)
- Need 3 check disks, bad performance with scale.



RAID 2 ("Bit-level ECC")
Disks: N≥3
SeqRead: depends
SeqWrite: depends
RandRead: depends
RandWrite: depends
Max fails w/o loss: 1
Overhead: ~ 3/N (actually more complex)

Safe to ignore

XOR parity demo

- Given four 4-bit numbers: [0011, 0100, 1001, 0101]

XOR them

~~0011~~
0100
1001
⊕ 0101
1011

Lose one and XOR what's left

1011
0100
1001
⊕ 0101
0011

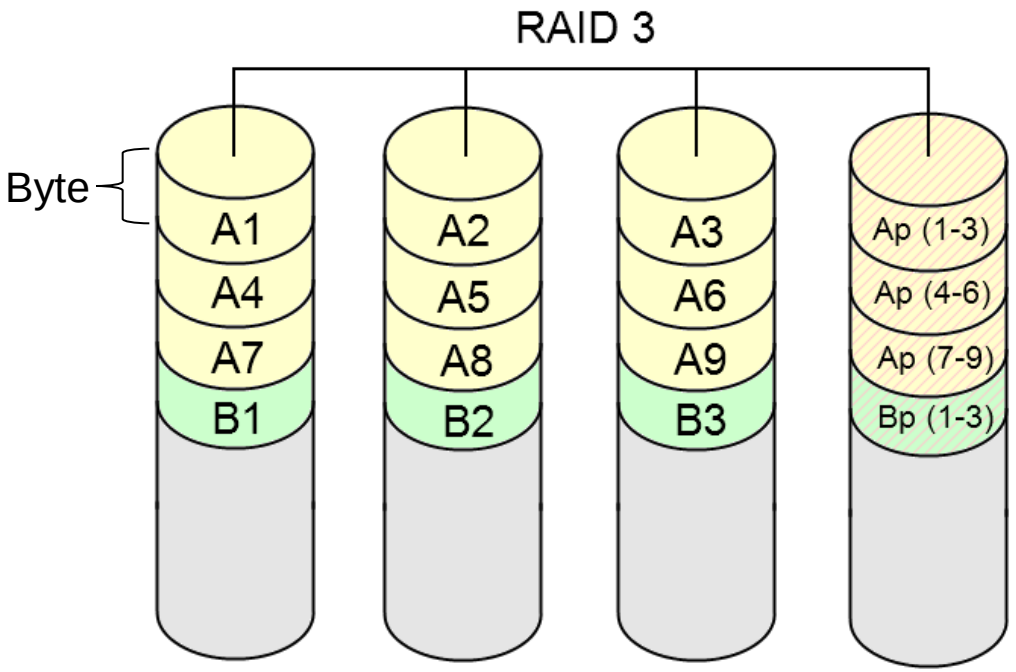
Recovered!

- Given N values and one parity, can recover the loss of *any* of the values

RAID 3

- N-1 drives contain data, 1 contains parity data
- Last drive contains the parity of the corresponding **bytes** of the other drives.
- Parity: XOR them all together

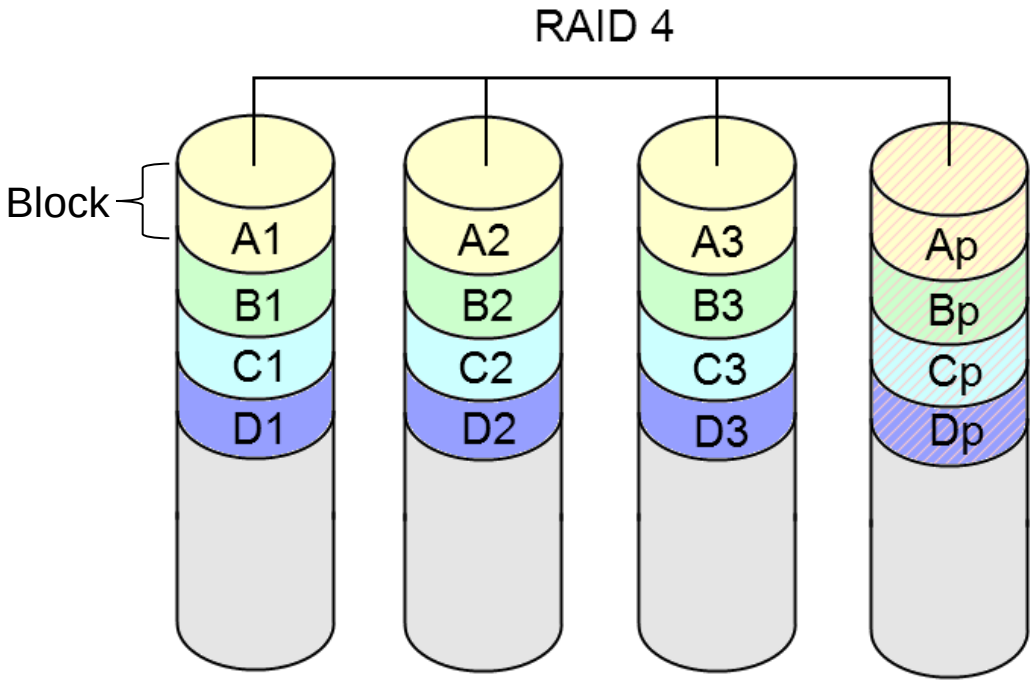
$$p[k] = b[k,1] \oplus b[k,2] \oplus \ldots \oplus b[k,N]$$



RAID 3 ("Byte-level parity")
Disks: N≥3, C=1
SeqRead: N
SeqWrite: N
RandRead: 1
RandWrite: 1
Max fails w/o loss: 1
Overhead: 1/N

RAID 4

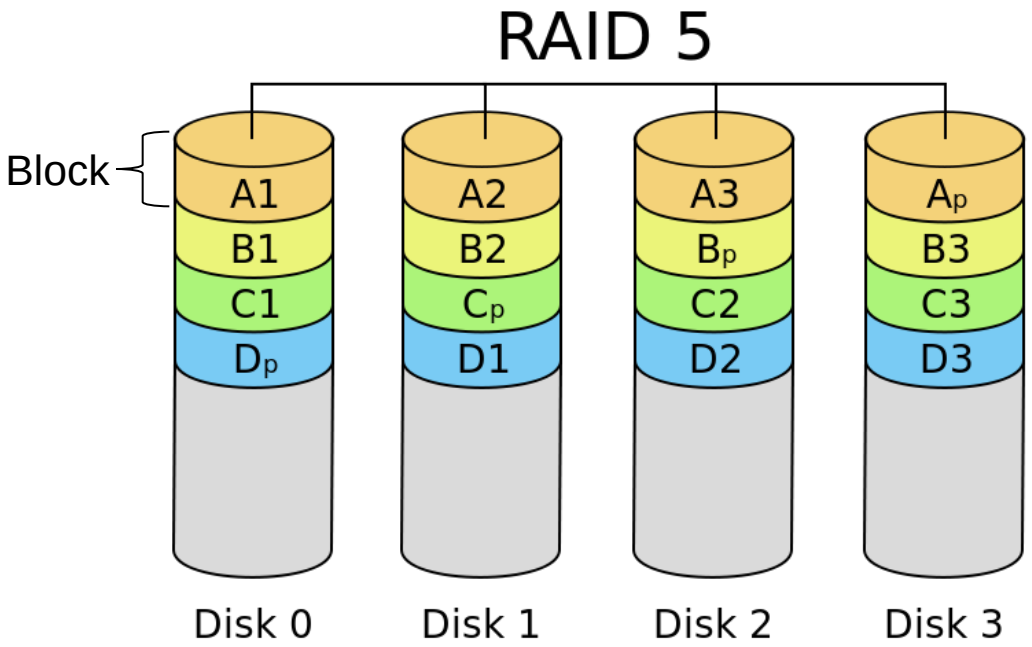
- N-1 drives contain data , 1 contains parity data
- Last drive contains the parity of the corresponding **blocks** of the other drives.
- Why is this different? Now we don't need to engage ALL the drives to do a single small read!
  - Drive independence improves small I/O performance
- Problem: Must hit parity disk on every write



RAID 4 ("Block-level parity")
Disks: N≥3, C=1
SeqRead: N
SeqWrite: N
RandRead: **N**
RandWrite: 1
Max fails w/o loss: 1
Overhead: 1/N

RAID 5

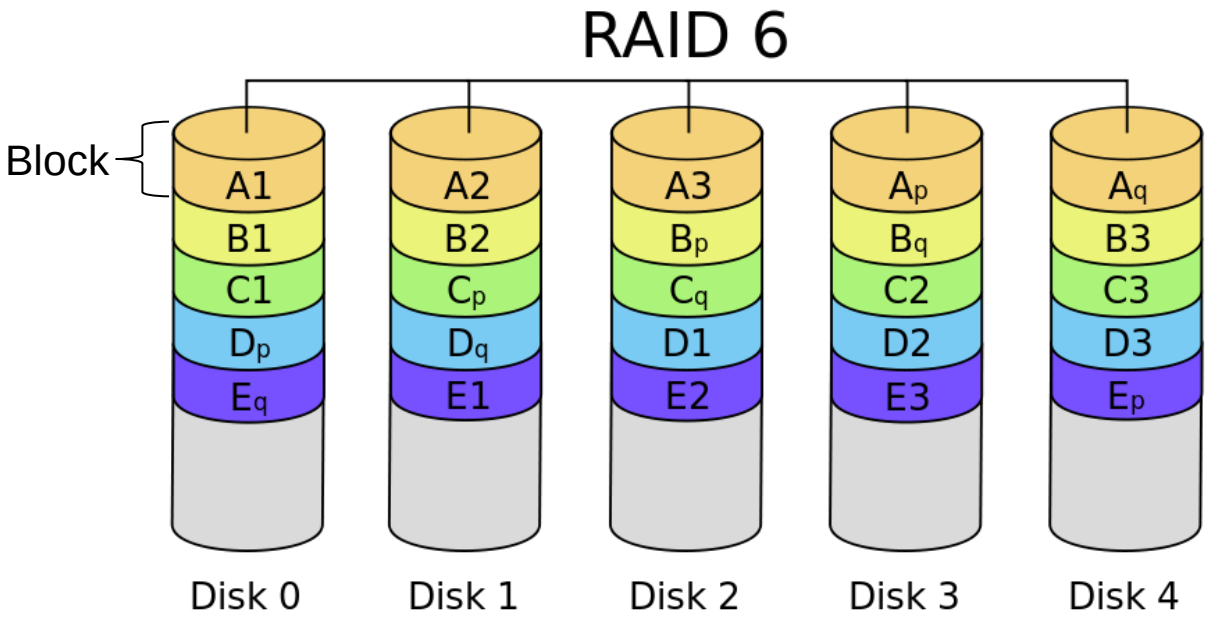
- Distribute the parity:
  Every drive has (N-1)/N data and 1/N parity
- Now two independent writes will often engage two separate sets of disks.
  - Drive independence improves small I/O performance, again



RAID 5 ("Distributed parity")
Disks: N≥3, C=1
SeqRead: N
SeqWrite: N
RandRead: N
RandWrite: **N**
Max fails w/o loss: 1
Overhead: 1/N

RAID 6

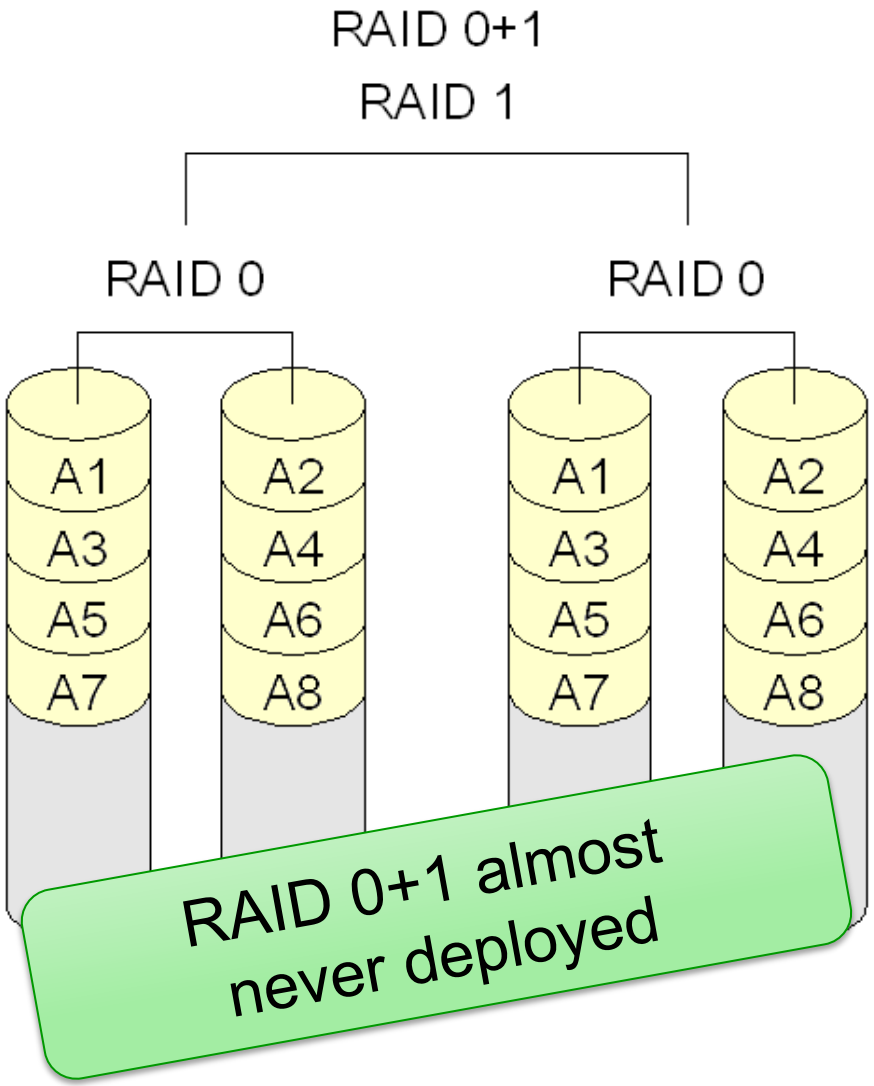
- Distribute *more* parity:
  Every drive has (N-2)/N data and 2/N parity
- Second parity not the same; not a simple XOR. Various possibilities (Reed-Solomon, diagonal parity, etc.)
- Allowing two failures without loss has huge effect on MTTF
  - Essential as drive capacities increase – the bigger the drive, the longer RAID recovery takes, exposing a longer window for a second failure to kill you



RAID 6 ("Dual parity")
Disks: N≥4, C=2
SeqRead: N
SeqWrite: N
RandRead: N
RandWrite: N
Max fails w/o loss: **2**
Overhead: **2/N**

Nested RAID

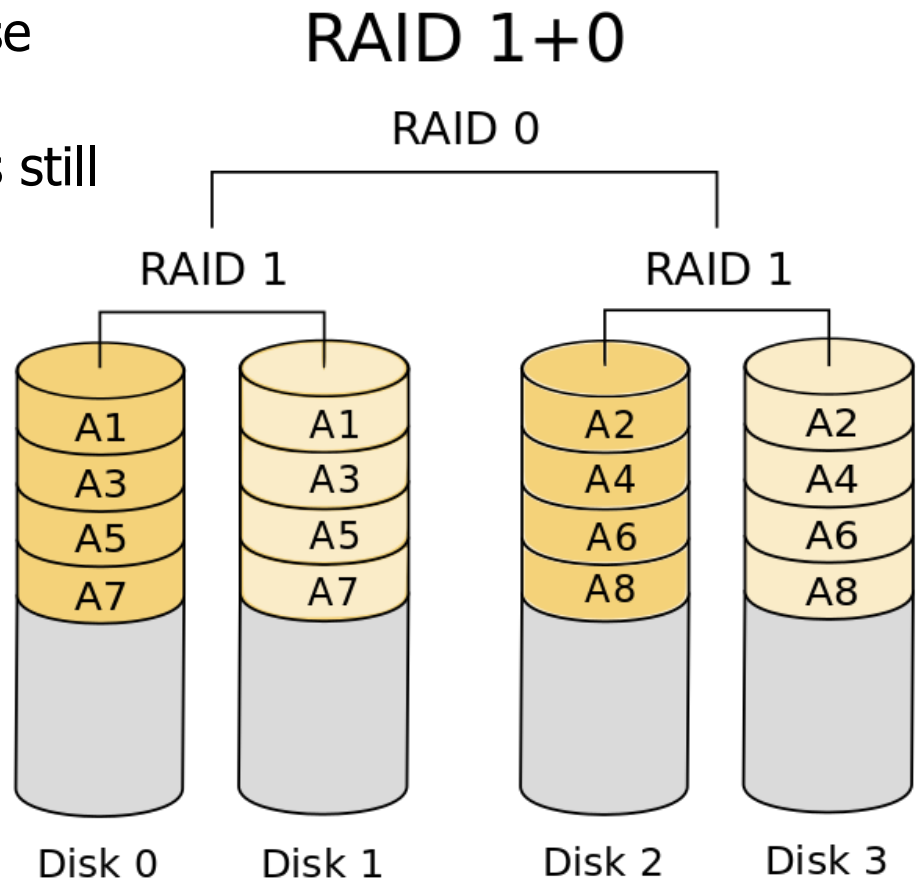
- Deploy hierarchy of RAID
- Example shown: RAID 0+1



RAID 0+1 ("mirror of stripes")
Disks: N>4, typ. N_1=2
SeqRead: N_0*N_1
SeqWrite: N_0
RandRead: N_0*N_1
RandWrite: N_0
Max fails w/o loss: N_0*(N_1-1) (unlikely)
Mins fails w/ possible loss: N_1
Overhead: N_1

RAID 1+0

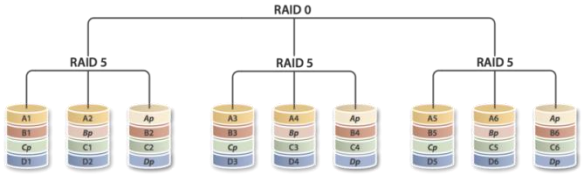
- RAID 1+0 is commonly deployed.
- Why better than RAID 0+1?
  - When RAID 0+1 is degraded, lose striping (major performance hit)
  - When RAID 1+0 is degraded, it's still striped



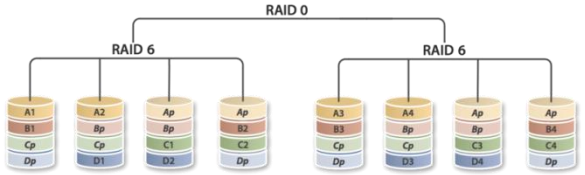
RAID 1+0 ("RAID 10", "Striped mirrors")
Disks: N>4, typ. N_1=2
SeqRead: N_0*N_1
SeqWrite: N_0
RandRead: N_0*N_1
RandWrite: N_0
Max fails w/o loss: N_0*(N_1-1) (unlikely)
Mins fails w/ possible loss: N_1
Overhead: N_1/N

Other nested RAID

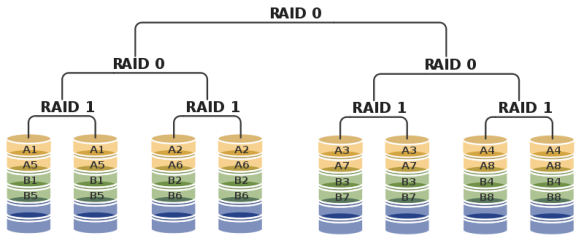
- RAID 50 or 5+0
  - Stripe across 2 or more block-parity RAIDs



- RAID 60 or 6+0
  - Stripe across 2 or more dual-parity RAIDs



- RAID 10+0
  - Three-levels
  - Stripe across 2 or more RAID 10 sets
  - Equivalent to RAID 10
  - Exists because hardware controllers can't address that many drives, so you do RAID-10s in hardware, then a RAID-0 of those in software



The small write problem

- Specific to block level striping
- Happens when we want to update a single block
  - Block belongs to a stripe
  - How can we compute the new value of the parity block

First solution

- Read values of N-1 other blocks in stripe
- Recompute

  $p[k] = b[k] \oplus b[k+1] \oplus \ldots \oplus b[k+N-1]$

- Solution requires
  - N-1 reads
  - 2 writes (new block and parity block)

Second solution

- Assume we want to update block b[m]
- Read old values of b[m] and parity block p[k]
- Compute

  p[k] = new_b[m] ⊕ **old**_b[m] ⊕ **old**_p[k]

- Solution requires
  - 2 reads (old values of block and parity block)
  - 2 writes (new block and parity block)

Picking a RAID configuration

- Just need raw throughput, don't care about data loss?
  (e.g., scratch disk for graphics/video work)
  - RAID 0
- Small deployment? Need simplicity?
  (e.g., Local boot drives for servers)
  - RAID 1, n=2
- Small deployment but need low overhead?
  (e.g., Home media storage)
  - RAID 5, n=4..6
  - Danger: big drives with large RAID-5's increase risk of double failure during repair
- Need simplicity and big throughput?
  - RAID 1+0
- Large capacity?
  - RAID 6 or RAID 6+0, n=15..30
- Simplicity when workload never has small writes?
  - RAID 4, n=4..6

High availability vs. resiliency

- Main purpose of RAID is to build fault-tolerant file systems for **high availability**
- However,

RAID DOES NOT REPLACE BACKUPS

What RAID can't do

- RAID does not protect against:
  - Human error (e.g. accidental deletion)
  - Malware
  - Non-drive hardware failure (I/O card, motherboard, CPU, RAM, etc.)
  - Undetected read errors from disk
    - Unless you're reading all disks and checking against parity every time...
      - But that's performance-prohibitive.
      - Even then you wouldn't know which drive's data was bad.
  - Data corruption due to power outage
    - In fact, RAID makes it worse...what if you lose power when only *some* of the drives in a stripe have been updated? The "write hole"
  - Catastrophic destruction of the system, rack, building, city, continent, or planet

Recovering from failure

- When a disk fails in an array, the array becomes **degraded**
- While array is degraded, it is at risk of additional disk failures!
  - Remember, for RAID 1/4/5, double disk failure = death!
- When the disk is replaced, the degraded array can be **rebuilt**
  - For RAID-1, re-copy data. For RAID-4/5/6, reconstruct from parity.
- **Hot spares**: Disks that don't participate in the array
  - On failure, system immediately disabled bad disk, promotes a spare, and begins rebuilding.
  - Reduces time spent in degraded state.
  - Administrator can remove and replace bad disk at leisure (no urgency).

Issues

- What happens when new disks are added into the system?
  - Usually have to change layout, rearrange data
  - (More advanced techniques can avoid/minimize this)
- How to "grow" the array by replacement with bigger disks?
  - Must replace every disk in turn, rebuilding between each
  - Only a consideration for small deployments – large deployments just add whole shelves of disks at a time

Optimizations in the Array Controller

- Access Coalescing
  - Determine whether several disk I/Os on same disk are coalesced into a single disk I/O.
- Load Balancing
  - How the disk controller distributes the load between a disk and its mirror.
    - E.g. read from 3 disks or submit requests to 6 (3+ mirrors).
    - Advantage: Reduced transfer time
    - Disadvantage: Queue length longer at all disks. (Consider 2 3s vs. 2 6s).

More Array Controller Optimizations

- Adaptive Prefetching
  - Based on automatic detection of sequential I/O streams.
- Write-back Caching Policy
  - When are dirty data written from cache to disk
    - Parameter: max number of dirty blocks that can be held in cache without triggering disk writes.