Reliable Arrays of Inexpensive Disks

-or-

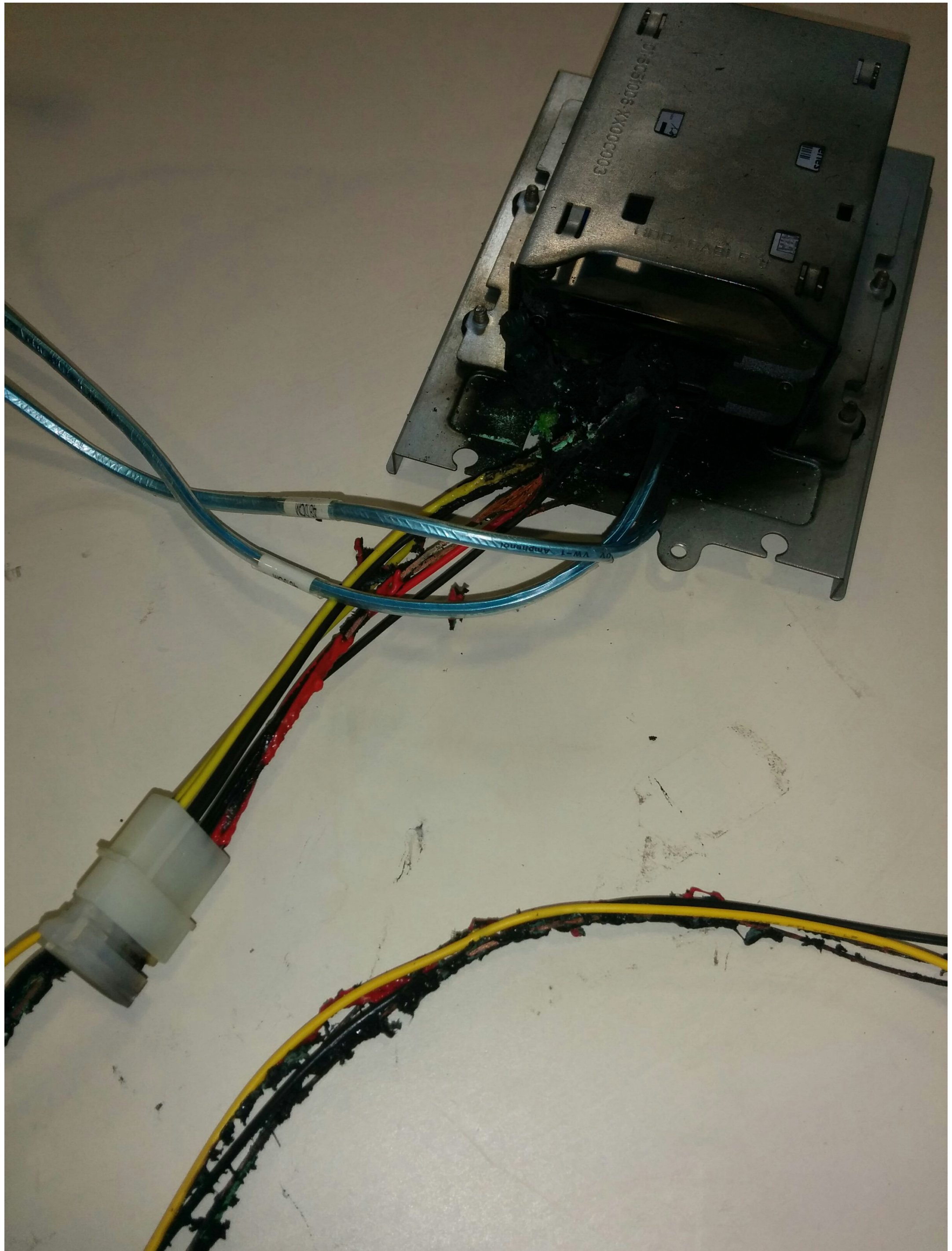
How can we improve performance?

Redundant Arrays of Inexpensive Disk

- How do we get more capacity, performance and reliability on a storage system?
- Key Parameters
  - Capacity
  - Reliability
  - Performance
    - Single request latency
    - Steady state throughput, many sequential or random requests

Causes of Disk Failure

- *Head crash*
  - a head may contact the rotating platter due to mechanical shock or other reason.
- *Bad sectors*
  - some magnetic sectors may become faulty without rendering the whole drive unusable.
- *Stiction*
  - after a time the head may not "take off" when started up as it tends to stick to the platter,
- *Circuit failure*
  - components of the electronic circuitry may fail making the drive inoperable.
- *Bearing and motor failure*
  - electric motors may fail or burn out, and bearings may wear enough to prevent proper operation.
- *Miscellaneous mechanical failures*
  - any mechanism can break or fail, preventing normal operation,



Recall: types of workload

- Sequential
  - Access large # of contiguous sectors (or blocks)
  - E.g., read 1MB starting from sector 12
  - App: searching through large log file for keyword

- Random
  - Small requests, each on random location on disk
  - E.g., first read 4KB at sector 10, then at 255, next at 150, …
  - App: transaction management

Workload

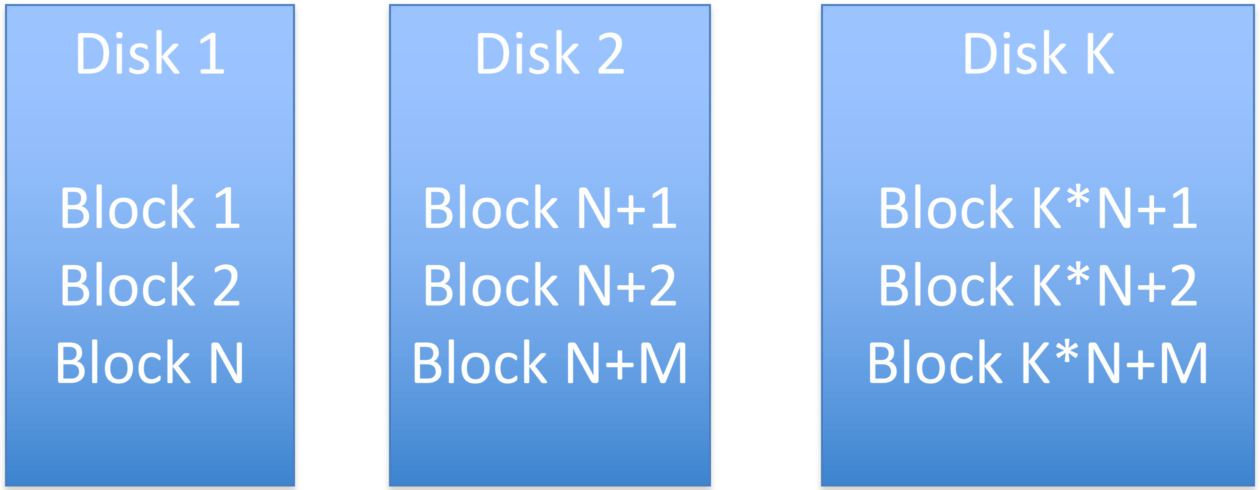
- Sequential: actual bandwidth S (MB/sec)
  - Seek and rotates once
- Random: actual bandwidth denoted as R
  - Seek and rotates for every request
- Typically S >> R

Throughput: sequential vs. random

- Consider disk with 7ms avg seek, 10,000 RPM platter speed and 50 MB/sec transfer rate, 4KB/block
- Sequential access of 10 MB
  - S = 10 MB / (7 + 3 + 10/50*1000) = 10MB/210ms = 47.62 MB/Sec
- Random access of 10 MB (2,500 blocks)
  - R = 10MB / (2500*(7 + 3 + 4/50)) = 10MB / 25.2sec = .397 MB/Sec

How do we get more capacity?

- 6 TB is a lot….
- But some data sets are bigger
  - Sizes of petabytes not uncommon
- Use additional disks (Just a bunch of disks – JBOD)



JBOD Performance Characteristics

- Capacity is the sum of the capacity of each disk
- Bandwidth and latency is that of each individual disk
- Reliability is worse then single disk system

JBOD Reliability

- Measured in mean time between failure (MTBF)
  - Sum of operational time/ # of failures
- MTBF for single disk is ~800,000-1,200,000 hours (~136 years)
- However, MTBF for JBOD is mean time to first disk failure:
  - $MTBF_{JBOD}$= $MTBF_{DISK}$/ number of disks
  - So 200 disk JBOD will fail in .45-.65 years

Getting more performance

- Latency we are stuck with
- But we can get better bandwidth by having read/write be serviced by more then one disk
  - We can do this by distributing data in chunks across multiple disks
  - If read or write request is larger then chunk size, then we will get better bandwidth

Disk Striping (RAID 0)

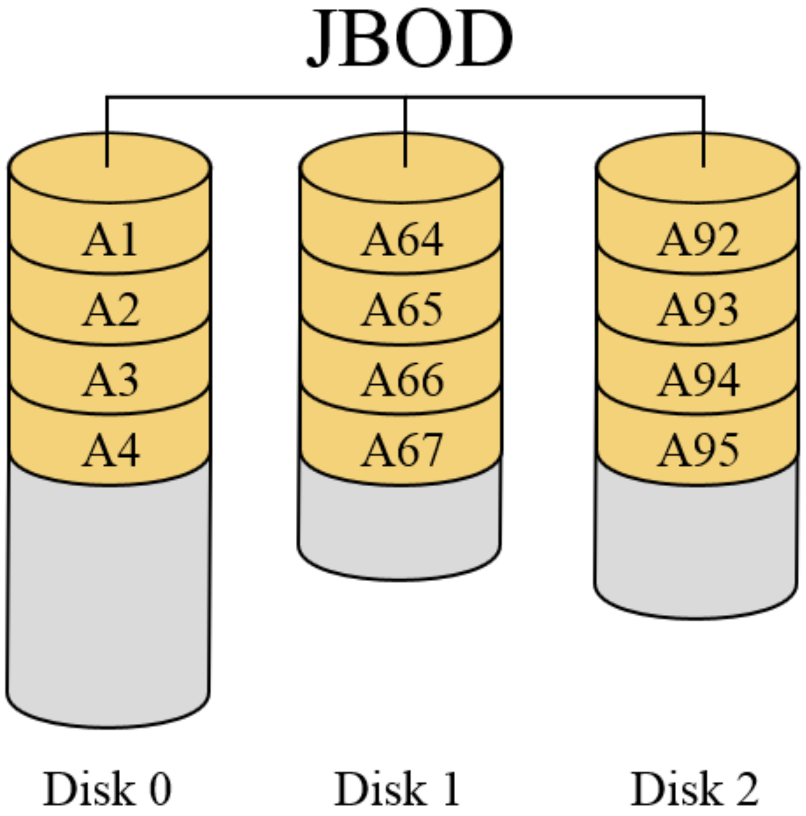
Disk 0	Disk 1	Disk 2	Disk 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

A row is called a “stripe”.

Collection of data (“chunk”) can be more then one disk block

JBOD

- Disks are concatenated together
  - Some may have more blocks than others

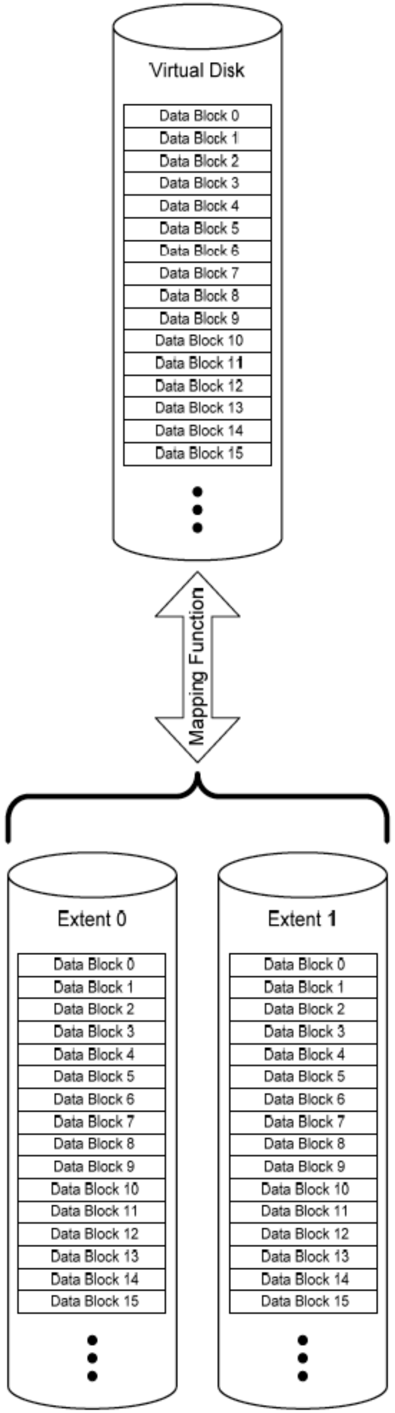


RAID 0 Performance

- Capacity and reliability is the same as JBOD
- Performance will depend on access patterns and chunk size
- Consider
  - Sequential workload: request single large file
  - Random workload: lots of requests for small amounts of data, all different
- In general
  - Latency will be that of a single disk
  - Throughput will be N*single disk bandwidth (N*S or N*R, depending on sequential or random workload)

How can we improve reliability?

- Fail/stop performance model
  - On failure, entire disk just stops working
- We can just make a copy of all the data on a second disk
  - Called “Mirroring” or RAID 1
- 100% overhead in cost
  - No capacity advantage from 2nd disk
- Reads can go to either disk
- Writes must go to both disks



Virtual Disk
Data Block 0
Data Block 1
Data Block 2
Data Block 3
Data Block 4
Data Block 5
Data Block 6
Data Block 7
Data Block 8
Data Block 9
Data Block 10
Data Block 11
Data Block 12
Data Block 13
Data Block 14
Data Block 15
Mapping Function
Extent 0
Data Block 0
Data Block 1
Data Block 2
Data Block 3
Data Block 4
Data Block 5
Data Block 6
Data Block 7
Data Block 8
Data Block 9
Data Block 10
Data Block 11
Data Block 12
Data Block 13
Data Block 14
Data Block 15
Extent 1
Data Block 0
Data Block 1
Data Block 2
Data Block 3
Data Block 4
Data Block 5
Data Block 6
Data Block 7
Data Block 8
Data Block 9
Data Block 10
Data Block 11
Data Block 12
Data Block 13
Data Block 14
Data Block 15

Disk Mirroring (RAID 1)

Disk 0	Disk 1
0	0
1	1
2	2
3	3

RAID 1 Performance Latency

- Read request latency that of single disk
- Write request latency will be worst case of two disks
  - Both writes must complete for operation to complete

RAID 1 Performance Throughput

- Sequential writes are performance N/2*S
  - Must write out to both disks
- Sequential reads are performance of N/2*S!
  - Simple interleaving of sequential requests across two disks will yield 50% bandwidth per disk
- Random reads can see bandwidth of both disks
  - Requests distributed across both disks so R*N Mb/s
- Random writes see ½ bandwidth of single disk
  - Need to write to both mirrors.

What happens when a disk fails

- Operate in “degraded” mode
  - Reads and writes only go to healthy disk
- Notify operator of failure
  - Swap out bad disk (often hot-swap is done)
- Rebuild mode, copy data from good disk to bad
  - May occur concurrent with continued operation at performance cost
- Continue in normal mode

Combining Mirroring With Striping RAID 1+0 or RAID 10

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

Now sequential operations can utilize 2 disks, but still only 50% total bandwidth

Can stripe across more then two disks

RAID 10: Mirror (1) + Striping (0)

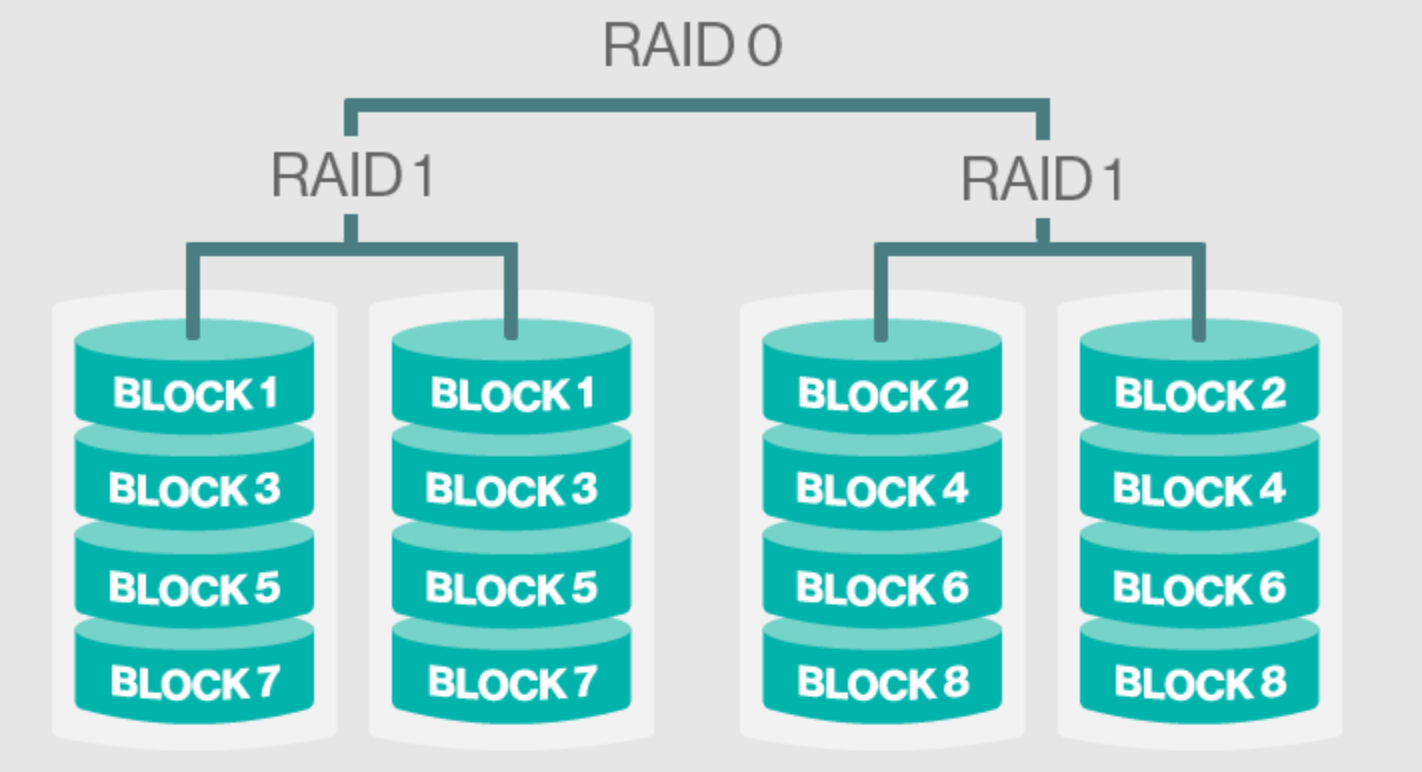


Image from techtarget.com

Can we make it more efficient?

- Mirroring is expensive solution due to complete replication of the data
  - Doubles system cost with limited performance improvement (random reads).
- Other, more efficient means of creating data redundancy exist.
  - Erasure correcting codes
    - Recover missing data if you know a failure (erasure) has occured

Parity

- Count up the number of 1s in a string
  - Parity bit is 1 if odd number
  - Parity bit is 0 if even number
  - “Even parity” (string + parity bit is even number of ones)
- Example
  - 0x54 == 0101 0100_2 (Three 1’s --> parity bit is set to “1”)"
  - Store 9 bits: 0101 0100 1

Filling in Missing Data

- If one bit is missing, and we know which one, we can figure out its value:
  - If parity bit is missing, don't worry!
  - Otherwise, count the number of 1s in the remaining bits
    - If sum is even, missing bit is 0
    - If sum is odd, missing bit is 1
  - Example?
    - 010X 0100 1 → 3 ones, so missing a one
    - 0101 010X 1 → 4 ones so missing a zero.

Exclusive Or (XOR) computes parity

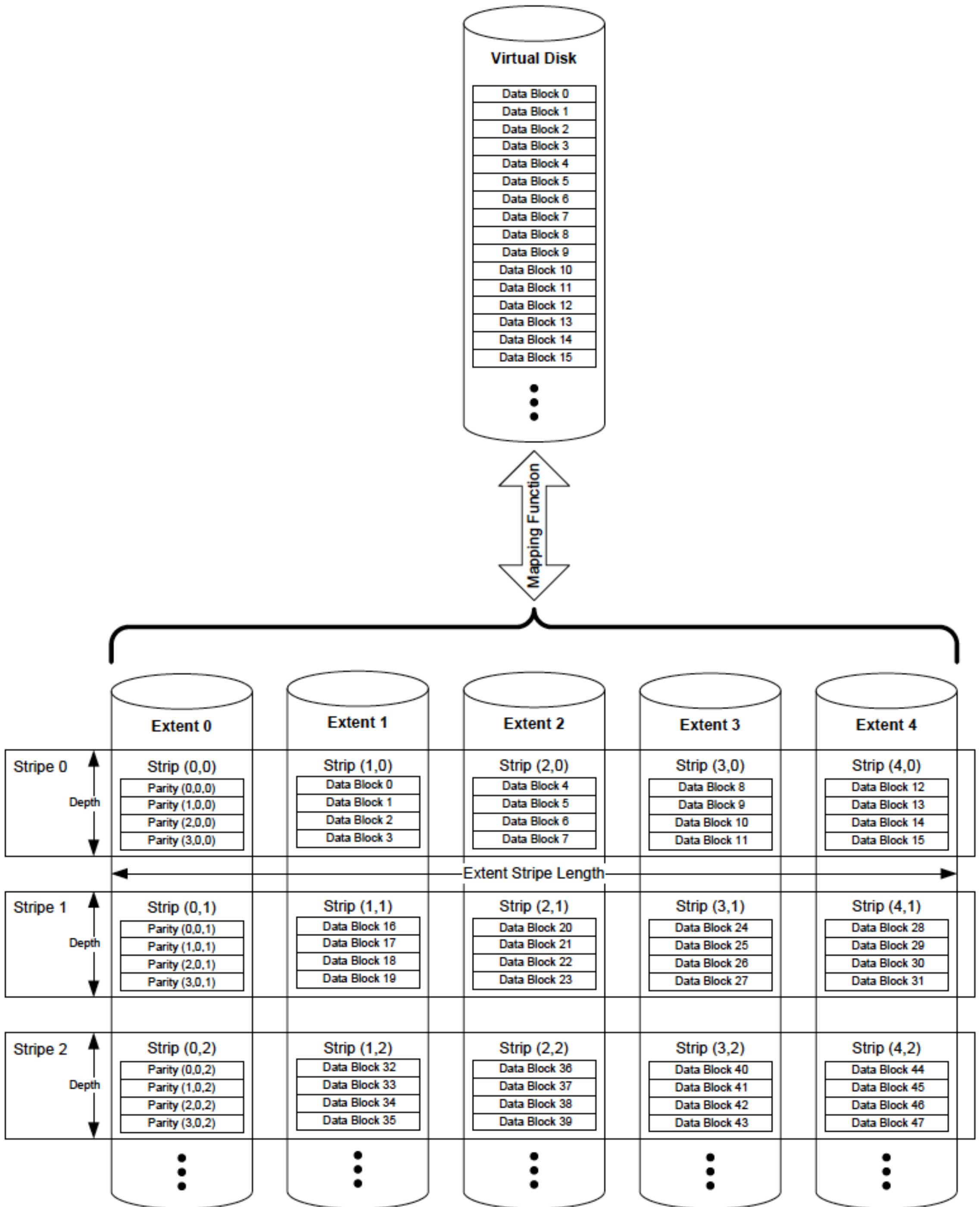
- Function definition
  – XOR (0, 1) = 1
  – XOR (1, 0) = 1
  – XOR (0,0) = 0
  – XOR (1,1) = 0
- Generalizing to more then 2 bits
  – XOR(0,0,1,1) = 0
  – XOR(0,1,0,0) = 1
  – XOR(1,0,1,1) = 1

Parity Over Redundancy

- With mirroring we had 100% overhead
- If we used parity instead, wasted capacity would decrease to 1/N.

RAID 4:
Add a Parity Disk

- Strip data like in RAID 1
- Compute parity on each block
- Store block parity in corresponding location on parity disk


Virtual Disk
Data Block 0
Data Block 1
Data Block 2
Data Block 3
Data Block 4
Data Block 5
Data Block 6
Data Block 7
Data Block 8
Data Block 9
Data Block 10
Data Block 11
Data Block 12
Data Block 13
Data Block 14
Data Block 15
Mapping Function
Extent 0
Extent 1
Extent 2
Extent 3
Extent 4
Stripe 0
Depth
Strip (0,0)
Parity (0,0,0)
Parity (1,0,0)
Parity (2,0,0)
Parity (3,0,0)
Strip (1,0)
Data Block 0
Data Block 1
Data Block 2
Data Block 3
Strip (2,0)
Data Block 4
Data Block 5
Data Block 6
Data Block 7
Strip (3,0)
Data Block 8
Data Block 9
Data Block 10
Data Block 11
Strip (4,0)
Data Block 12
Data Block 13
Data Block 14
Data Block 15
Extent Stripe Length
Stripe 1
Depth
Strip (0,1)
Parity (0,0,1)
Parity (1,0,1)
Parity (2,0,1)
Parity (3,0,1)
Strip (1,1)
Data Block 16
Data Block 17
Data Block 18
Data Block 19
Strip (2,1)
Data Block 20
Data Block 21
Data Block 22
Data Block 23
Strip (3,1)
Data Block 24
Data Block 25
Data Block 26
Data Block 27
Strip (4,1)
Data Block 28
Data Block 29
Data Block 30
Data Block 31
Stripe 2
Depth
Strip (0,2)
Parity (0,0,2)
Parity (1,0,2)
Parity (2,0,2)
Parity (3,0,2)
Strip (1,2)
Data Block 32
Data Block 33
Data Block 34
Data Block 35
Strip (2,2)
Data Block 36
Data Block 37
Data Block 38
Data Block 39
Strip (3,2)
Data Block 40
Data Block 41
Data Block 42
Data Block 43
Strip (4,2)
Data Block 44
Data Block 45
Data Block 46
Data Block 47

RAID 4

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
P0	0	1	2	3
P1	4	5	6	7
P2	8	9	10	11
P3	12	13	14	15

Calculating Block Parity

Block 0	Block 1	Block 2	Block 3	Parity
00	10	11	10	11
10	01	00	01	10

Computing Parity on Update

- Additive parity: Read out all data other blocks in stripe except one being updated and XOR together with new block
  - May require a lot of reads
- Subtractive parity: Compare new block with block being updated, flip parity bit of old and new differ.

RAID-4 Performance

- Capacity
  - Available storage is N-1 times disk size
  - Much better then RAID 1
- Reliability
  - Tolerates single disk failure, just as good as RAID 1

Computing parity on update

- Subtractive method: Compare new block with block being updated, flip parity bit when old and new differ
- Example: update block 0: B0 => B0'
  - P' = XOR(B0, B0', P)
  - How many read/write?

Block 0	Block 1	Block 2	Block 3	Parity
00	10	11	10	11
10	01	00	01	10

Single request latency

- Read
  - single disk latency D

Disk 0	Disk 1	Disk 2	Disk 3	Parity
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

- Write (assume subtractive method)
  - ?

Write latency

- Write requires two reads and then two writes
  - Need two reads to calculate new parity, followed by updates to data block and parity
  - Good news, reads can go in parallel, as can writes
  - So latency is twice that of a single disk (need one for read + one for write)

=>2D

Throughput

- Sequential reads
  - use all disks other then parity disk, so (N-1)* S
- Sequential writes (assume full-stripe write)
  - XOR all the blocks, write stripe across all disks including parity disk
  - Bandwidth of parity disk not available to data, so write throughput is (N-1)*S

Disk 0	Disk 1	Disk 2	Disk 3	Parity
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

Performance: throughput

- Random reads
  - Can spread across all disks except parity disk, so (N-1) * R (random read bandwidth)
- Random writes
  - ?

Small write problem

Disk 0	Disk 1	Disk 2	Disk 3	Parity
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

Update block 4 and 13:
- Will need to read and write parity disk for both updates

Parity disk will become bottleneck
- All updates need to update parity disk
- Have to be done in sequence, no parallelization

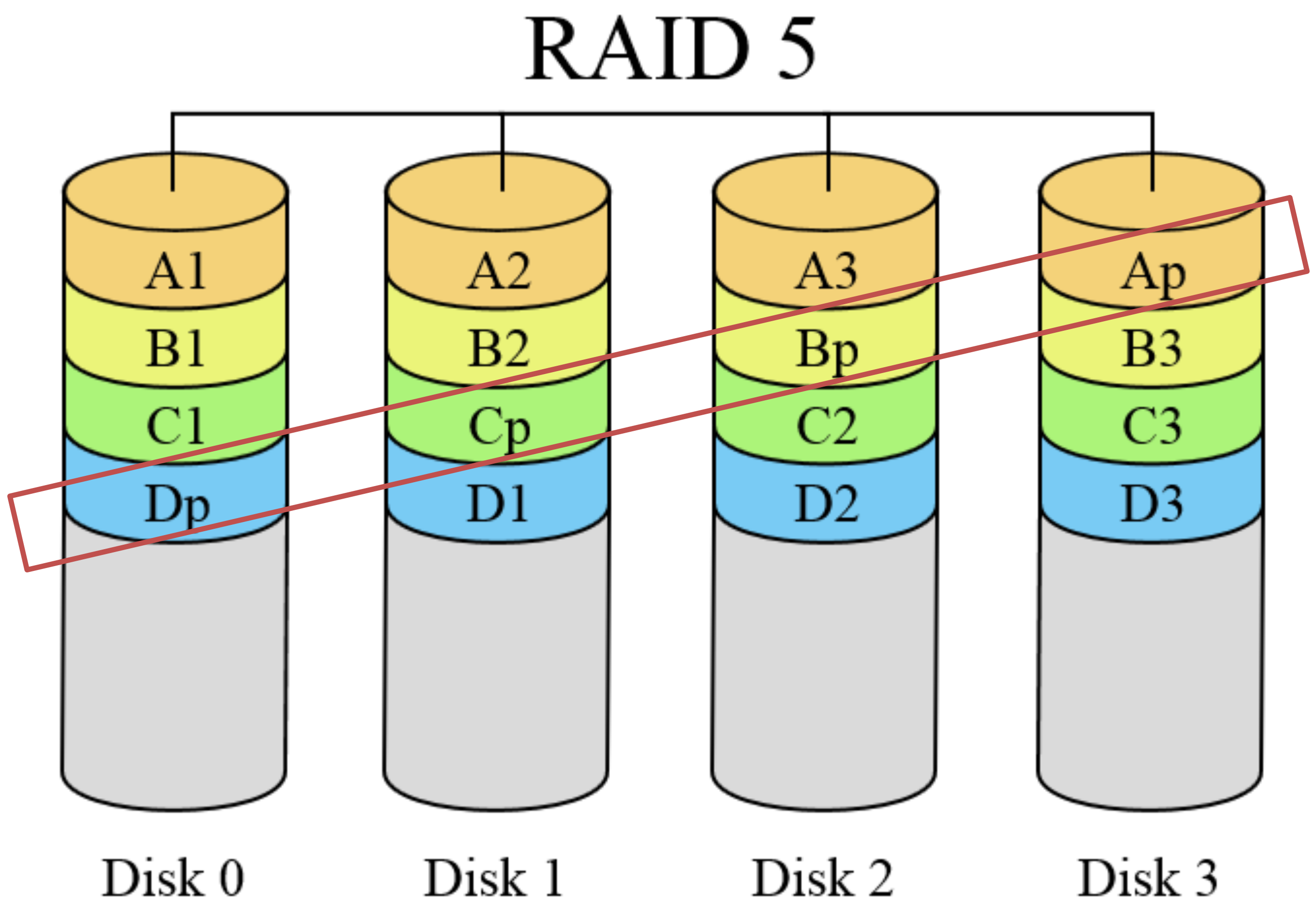
Since, two parity I/O's per update (read followed by write), so throughput on random writes will be (R/2) MB/s

Random write throughput

- Concurrency on random writes will be limited by access to parity disk
  - Small-write problem
  - So R/2

Solving the small write problem

- Caused by contention for parity disk
- So….
  - Why not stripe parity blocks across other disks rather then put into a single disk.
- RAID 5
  - Distributed parity


RAID 5
A1
B1
C1
Dp
A2
B2
Cp
D1
A3
Bp
C2
D2
Ap
B3
C3
D3
Disk 0
Disk 1
Disk 2
Disk 3

RAID 5

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

Random write:
-Write block 1: read 1, p0; write 1, p0 => disk 1, 4
-Write block 10: read 10, p2; write 10, p2 => disk 0, 2

- No more bottlenecks
- Since 4 operations/each, bandwidth = N/4 * R

Random write: RAID 5 vs. RAID 4

- Both require two read and two write
  - E.g., XOR(B0, B0', P0) = P0'
  - Read: B0, P0
  - Write: B0', P0'
- RAID 4
  - Two reads can be done in parallel
  - But two writes can only be done in sequence!
- RAID 5
  - Both reads and writes can be done in parallel

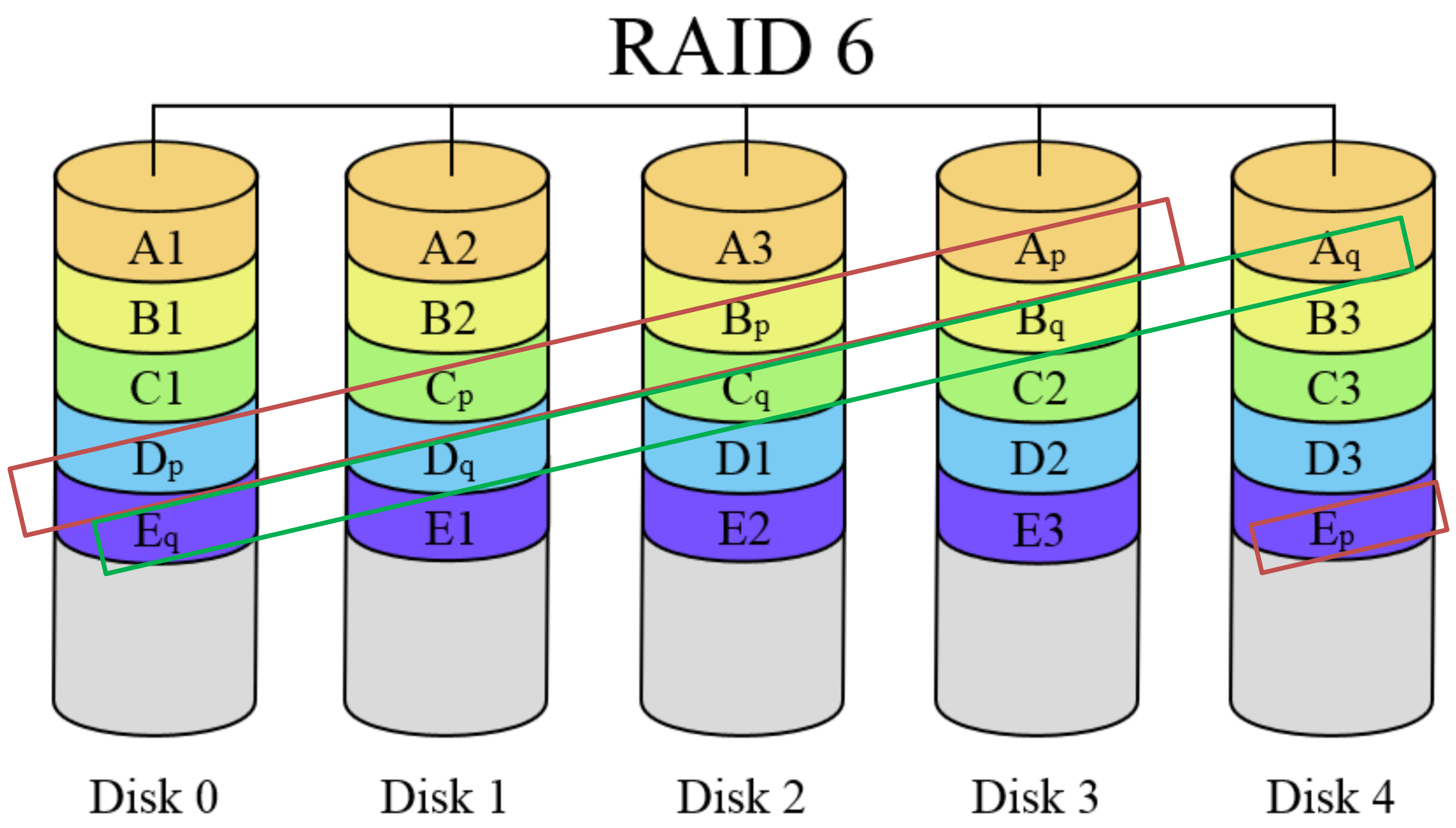
RAID 5 throughput

- Only differs on random workload
  – Sequential read/write: (N-1) * S
- Random read throughput is N * R (why?)
  – Get back bandwidth from parity disk
- Random write no longer bottlenecked by parity disk, although still need four operations
  – N/4 * R

Single request latency

- Same as RAID-4
  - Read: D
  - Write: 2D

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19



Two parity blocks per stripe:
p = normal parity code
q = a different coding scheme

Techniques

- JBOD (concatenating blocks)
- RAID 0 (striping or horizontal layout)
- RAID 1 (mirroring)
  – RAID 10/01 (mirroring + striping)
- RAID 4 (striping + block-level parity)
- RAID 5 (striping + distributed parity)
- RAID 6 (recover from two-disk failure)

Summary

	RAID-0	RAID-1	RAID-4	RAID-5
Capacity	N	$N/2$	$N-1$	$N-1$
Reliability	0	1 (for sure) $\frac{N}{2}$ (if lucky)	1	1
Throughput				
Sequential Read	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Sequential Write	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
Random Read	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$	$N \cdot R$
Random Write	$N \cdot R$	$(N/2) \cdot R$	$\frac{1}{2} \cdot R$	$\frac{N}{4} R$
Latency				
Read	D	D	D	D
Write	D	D	$2D$	$2D$