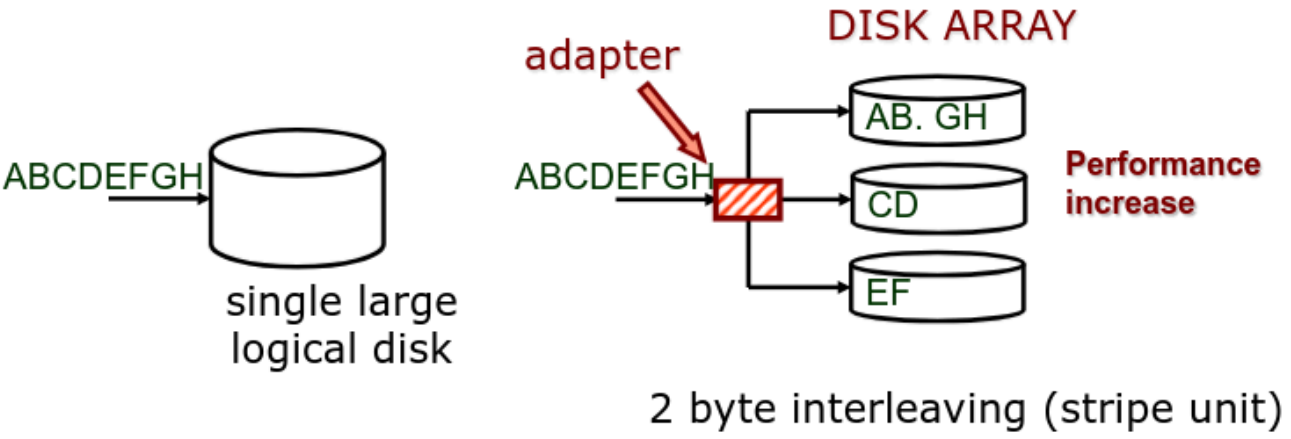
# RAID Disks

Disk can only reach a certain performance in a given time space. Combining several disk together you can go beyond the maximum capabilities of the singular disk. These are single large and fast disks. Starting as JBOD(Just a Bunch Of Disk), they are a way to separate the disks from the computer. RAID combine the disk together to make them larger and faster. RAID does so dividing a single information between all the disks available to use the parallelism of the structure to speed up the transfer rate(Striping method). The decreasing in side is a side effect of the increasing in performance. The parallelism introduce a new way to something goes wrong.
Striping works letting know the computer using a RAID disks as a single disk and not a collection of disks(I/O virtualization).




To put the blocks of a file in a RAID structure I can divide the blocks in the various disks in the structure or I can divide the blocks in smaller blocks and save them in parallel in the RAID structure

- Striping: data are written sequentially (a vector, a file, a table, ...) in units (stripe unit: bit, byte, blocks) on multiple disks according to a cyclic algorithm (round robin)
- Stripe unit: dimension of the unit of data that are written on a single disk
- Stripe width: number of disks considered by the striping algorithm
  1. multiple independent I/O requests will be executed in parallel by several disks decreasing the queue length (and time) of the disks
  2. single multiple-block I/O requests will be executed by multiple disks in parallel increasing of the transfer rate of a single request
  As I increase the number of disks used I increase the risk of failure of my RAID disk so I need to introduce **REDUNDANCY**
  Redundancy: error correcting codes (stored on disks different from the ones with the data) are computed to tolerate loss due to disk failures
  Since write operations must update also the redundant information, their performance is

worse than the one of the traditional writes. To enable all of this I use the usual checksum to assure that I have all the data correctly updated in my disk.

# RAID

Redundant Array of Inexpensive Disks: use multiple disks to create the illusion of a large, faster, more reliable disk. Externally, RAID looks like a single disk(it is Trasparent), Data blocks are read/written as usual, No need for software to explicitly manage multiple disks or perform error checking/recovery. Internally, RAID is a complex computer system: Disks managed by a dedicated CPU + software, RAM and non-volatile memory, Many different configuration options(RAID levels)

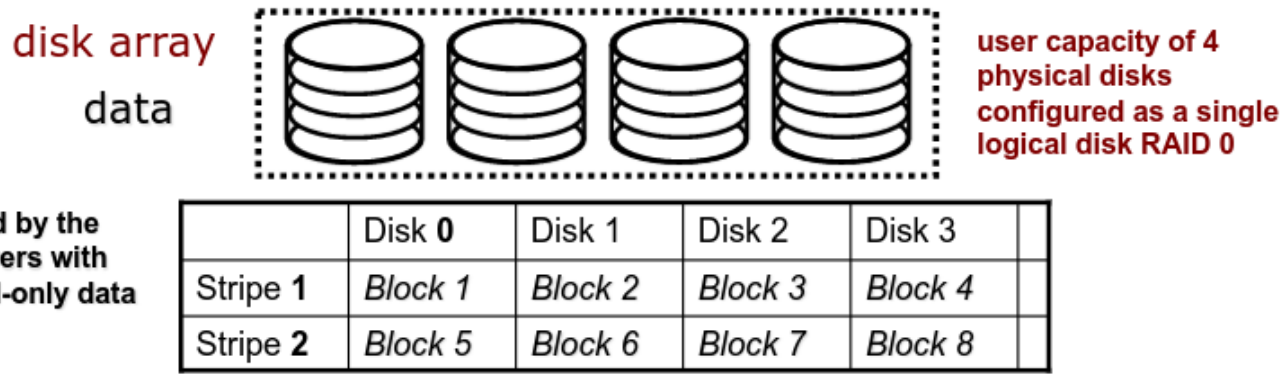
## RAID Levels

- RAID 0 striping only
- RAID 1 mirroring only
  - RAID 0+1 (nested levels)
  - RAID 1+0 (nested levels)
- RAID 2 bit interleaving (not used)
- RAID 3 byte interleaving - redundancy (parity disk)
- RAID 4 block interleaving - redundancy (parity disk)
- RAID 5 block interleaving - redundancy (parity block distributed)
- RAID 6 greater redundancy (2 failed disks are tolerated)

### RAID level 0: striping, no redundancy

Data are written on a single logical disk and splitted in several blocks distributed across the disks according to a striping algorithm used where performance and capacity, rather than reliability, are the primary concerns, minimum two drives required:

- lowest cost because it does not employ redundancy (no error-correcting codes are computed and stored)
- best write performance (it does not need to update redundant data and it is parallelized)
- single disk failure will result in data loss



Since I am cutting data and using the disks in parallel I have the maximum performance I can. My speed can increase up to the number of disks I am using. This is also the lowest cost as I am exploiting as much as possible my data structure.
Key idea: present an array of disks as a single large disk
Maximize parallelism by striping data cross all N disks
How do you access specific data blocks?

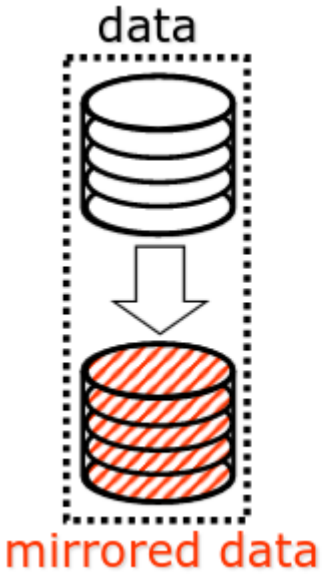
- Disk = logical_block_number % number_of_disks
- Offset = logical_block_number / number_of_disks
  Chunk size impacts array performance:
- Smaller chunks → greater parallelism
- Big chunks → reduced seek times
  Typical arrays use 64KB chunks
  As usual, we focus on sequential and random workloads. Assume disks in the array have random transfer rate R. Doing a single operation is different to do millions of small operation.
  Capacity: Nx Lap One DIsk(All space on all drives can be filled with data)
  Reliability: 0(If any drive fails, data is permanently lost, MTTDL = MTTF/N)
  Sequential read and write: N *S(Full parallelization across drives)*
  *Random read and write: N* R(Full parallelization across all drives)

## RAID level 1: mirroring

whenever data is written to a disk it is also duplicated (mirrored) to a second disk (there are always two copies of the data), minimum 2 disk drives
Positive:

- high reliability: when a disk fails the second copy is used
- read of a data: it can be retrieved from the disk with the shorter queueing, seek, and latency delays
- fast writes (no error correcting code should be computed) – but still slower than standard disks (due to duplication)
  Negative:

- high costs (50% of the capacity is used)



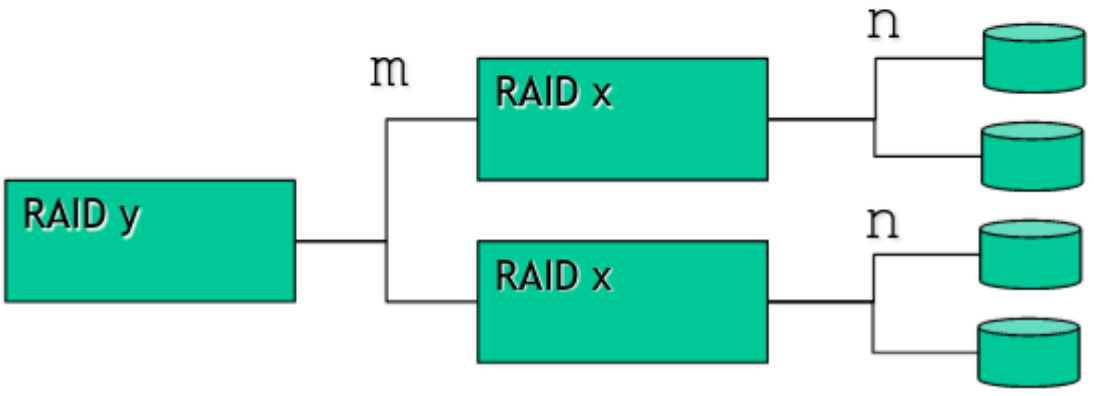

| Disk **0** | Disk 1 |
|---|---|
| *Block 1* | *Block 1* |
| *Block 2* | *Block 2* |
| *Block 3* | *Block 3* |

In principle, a RAID 1 can mirror the content over more than one disk. This give resiliency to errors even if more than one disk breaks. It allows with a voting mechanism to identify errors not reported by the disk controller. In practice this is never used, because the overhead and costs are too high.
The idea is that RAID 0 offers high performance, but zero error recovery so the key idea is to make two copies of all data. But if several disks are available(in even number) disks could be coupled and the total capacity is halved as each disk has a mirror. I need to combine the RAID level 0 and 1. So usually whenever you have 20 disks you can have 10 disks of storage and 10 disks for recovery(You can dispose the disks in RAID 0+1 or RAID 1+0).

RAID levels can be combined

RAID *x* + *y* (or RAID *xy*) =>

- *n* x *m* disks in total
- Consider *m* groups of *n* disks
- Apply RAID *x* to each group of *n* disks
- Apply RAID *y* considering the *m* groups as single disks

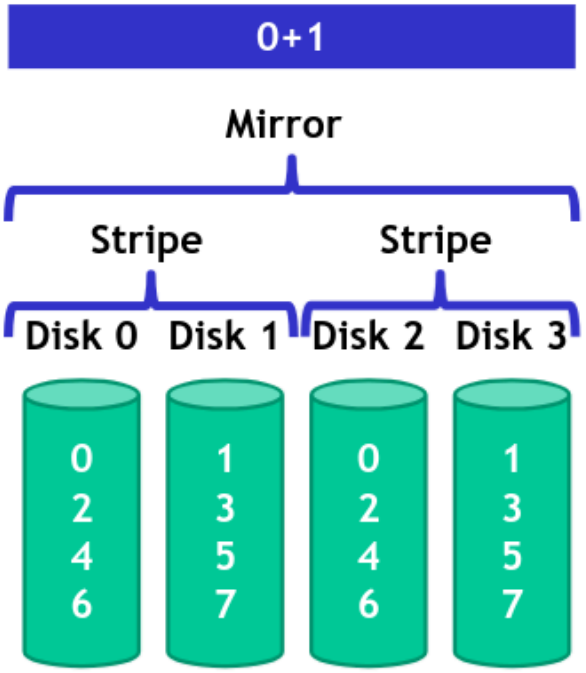



RAID 10 means you are combining RAID 1 and RAID 0.

*"Group of striped disks (RAID 0) that are then mirrored (RAID 1)"*

striping first (RAID 0)
then mirroring (RAID 1)

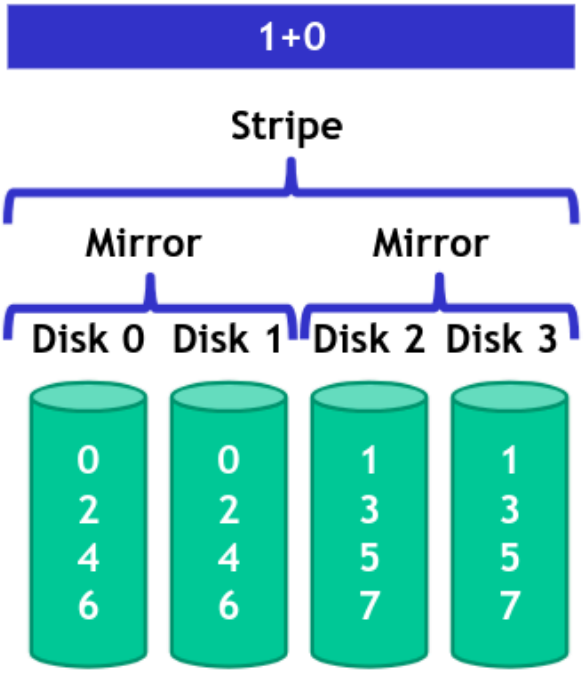
- minimum 4 drives
- after the first failure the model becomes as a RAID 0




*"Group of mirrored disks (RAID 1) that are then striped (RAID 0)"*

mirroring first (RAID 1)
then striping (RAID 0)

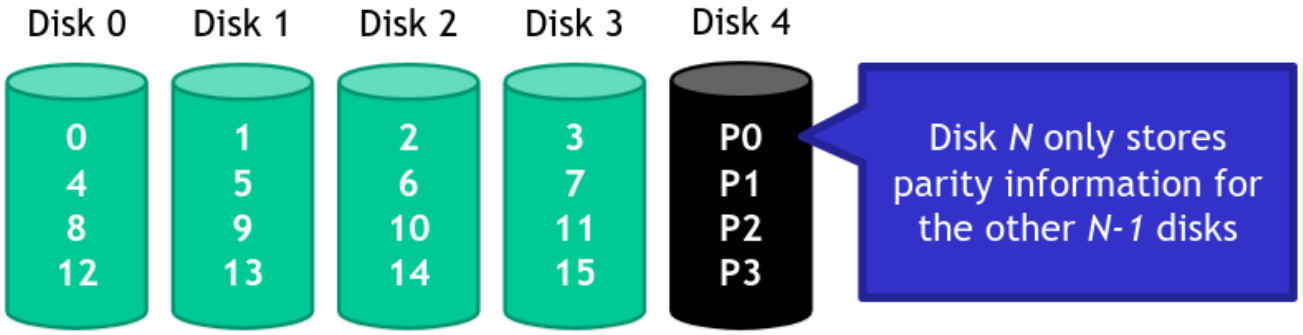
- minimum 4 drives are required
- used in databases with very high workload (fast writes)




The main difference is the fault tolerance level:
On most implementations of RAID controllers RAID 0+1 fault tolerance is less and RAID 1+0 fault tolerance is larger.
Capacity: N / 2(Two copies of all data, thus half capacity)
Reliability: 1 drive can fail, sometime more(If you are lucky, N / 2 drives can fail without data loss)
Sequential write: (N / 2) *S(Two copies of all data, thus half throughput)*
*Sequential read: (N / 2)* S(Half of the read blocks are wasted, thus halving throughput)
Random read: N *R(Best case scenario for RAID 1, Reads can parallelize across all disks)*
*Random write: (N / 2)* R(Two copies of all data, thus half throughput)
Mirrored writes should be atomic

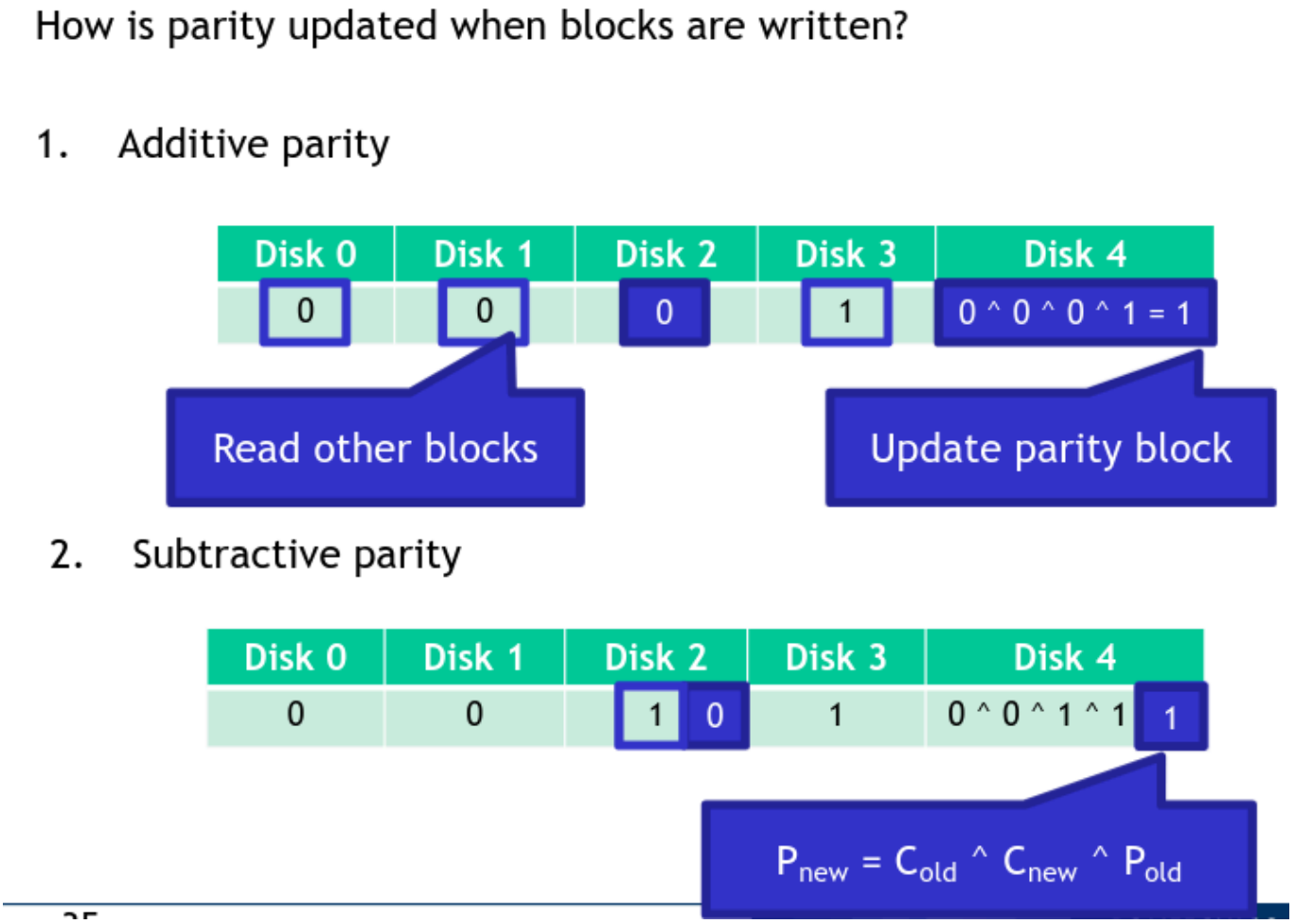
- All copies are written, or none are written
  However, this is difficult to guarantee
  Example: power failure
  Many RAID controllers include a write-ahead log, Battery backed, non-volatile storage of pending writes ,A recovery procedure ensures to recover the out-of-sync mirrored copies
  Decreasing the Cost of Reliability
- RAID 1 offers highly reliable data storage
- But, it uses N / 2 of the array capacity
- Can we achieve the same level of reliability without wasting so much capacity?
- Yes!
- Use information coding techniques to build light-weight error recovery mechanisms
  We need to Decrease the cost of Reliability.

## RAID 4




| Disk 0 | Disk 1 | Disk 2 | Disk 3 | Disk 4 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 ^ 0 ^ 1 ^ 1 = 0 |
| 0 | 1 | 0 | 0 | 0 ^ 1 ^ 0 ^ 0 = 1 |
| 1 | 1 | 1 | 1 | 1 ^ 1 ^ 1 ^ 1 = 0 |
| 0 | 1 | 1 | 1 | 0 ^ 1 ^ 1 ^ 1 = 1 |

Parity calculated using XOR



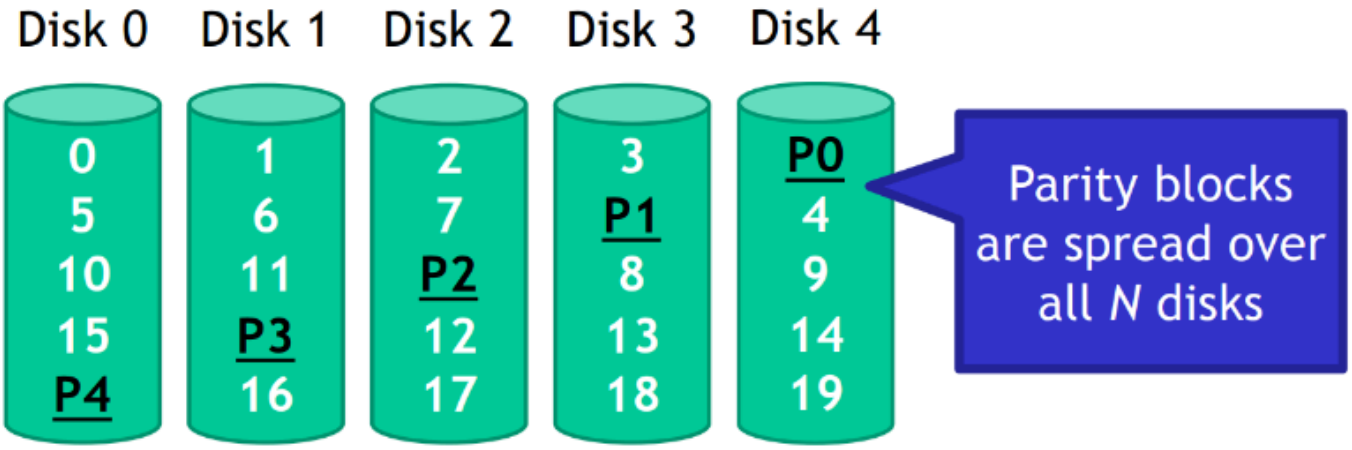

The idea is to not have an exact copy but store the important information and recompose the information from them in case of failure. We use a disk to store the parity(aggregation of the information coming from the other disks).
The bottleneck is created from the controller and the parity disk as every time I need to update the data I need to read and write on the parity disk. This bottleneck is particularly relevant if I need to do random writing(a lot of small files unrelated from each others) as this create serialization

**Analysis of RAID 4**

- Capacity:
  - N-1
  - Space on the parity drive is lost
- Reliability
  - 1 drive can fail
  - Massive performance degradation during partial outage
- Sequential Read and write: (N – 1) * S
  - Parallelization across all non-parity blocks in the stripe
- Random Read: (N – 1) * R
  - Reads parallelize over all but the parity drive
- Random Write: R / 2
  - Writes serialize due to the parity drive

- Each write requires 1 read and 1 write of the parity drive, thus R / 2

## RAID 5




| Disk 0 | Disk 1 | Disk 2 | Disk 3 | Disk 4 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 ^ 0 ^ 1 ^ 1 = 0 |
| 1 | 0 | 0 | 0 ^ 1 ^ 0 ^ 0 = 1 | 0 |
| 1 | 1 | 1 ^ 1 ^ 1 ^ 1 = 0 | 1 | 1 |
| 1 | 0 ^ 1 ^ 1 ^ 1 = 1 | 0 | 1 | 1 |

Writes are spread roughly evenly across all drives.
This creates an overhead in all the disks but we are distributing the parity diminushing the gravity of the bottleneck(We improve from RAID 4). The difference is that accessing the parity doesn't mean accessing the same disk.

**Analysis of Raid 5**

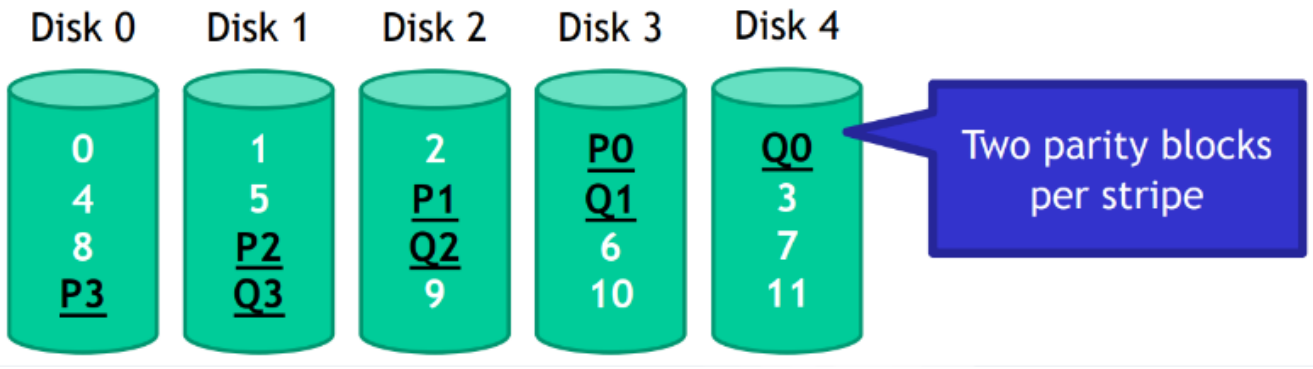
- Capacity: same as RAID 4
  - N – 1
- Reliability: same as RAID 4
  - 1 drive can fail
  - Massive performance degradation during partial outage
- Sequential Read and write:
  - (N – 1) * S same
  - Parallelization across all non-parity blocks
- Random Read:
  - N *R vs. (N – 1)* R
  - Unlike RAID 4, reads parallelize over all drives
- Random Write:
  - N / 4 * R vs. R / 2 for RAID 4
  - Unlike RAID 4, writes parallelize over all drives
  - Each write requires 2 reads and 2 write, hence N / 4

In RAID 5 I can access all the disk in parallel and I don't need to use a disk for the storing of parity.

## RAID 6

Meant to survive two failures of disk. I will run and manage a bit more slower but I can survive two failure. I use two parity disk extending the survivability. The problem is that i don't really want two phisical parity disk, they are vistual and are distributed along all the disks. The cost is that I reduce my storing capacity to N-2(N+2 disks required).
Uses Solomon-Reeds codes with two redundancy schemes

- (P+Q) distributed and independent
  High overhead for writes (computation of parities)
- each write require 6 disk accesses due to the need to update both the P and Q parity blocks (slow writes)
  Minimum set of 4 data disks





| RAID level | Capacity | Reliability | R/W performance | Rebuild performance | Suggested applications |
|---|---|---|---|---|---|
| 0 | 100% | N/A | Very good | Good | Non critical data |
| 1 | 50% | Excellent | Very good / good | good | Critical information |
| 5 | (n-1)/n | Good | Good/ fair | Poor | Database, transaction based applications |
| 6 | (n-2)/n | Excellent | Very good/ poor | Poor | Critical information, w/minimal |
| 1+0 | 50% | Excellent | Very good/ good | Good | Critical information, w/better performance |

## Other considerations

Many RAID systems include a hot spare

- An idle, unused disk installed in the system
- If a drive fails, the array is immediately rebuilt using the hot spare
  RAID can be implemented in hardware or software
- Hardware is faster and more reliable…
- But, migrating a hardware RAID array to a different hardware controller almost never works
- Software arrays are simpler to migrate and cheaper, but have worse performance and weaker reliability
- Due to the consistent update problem