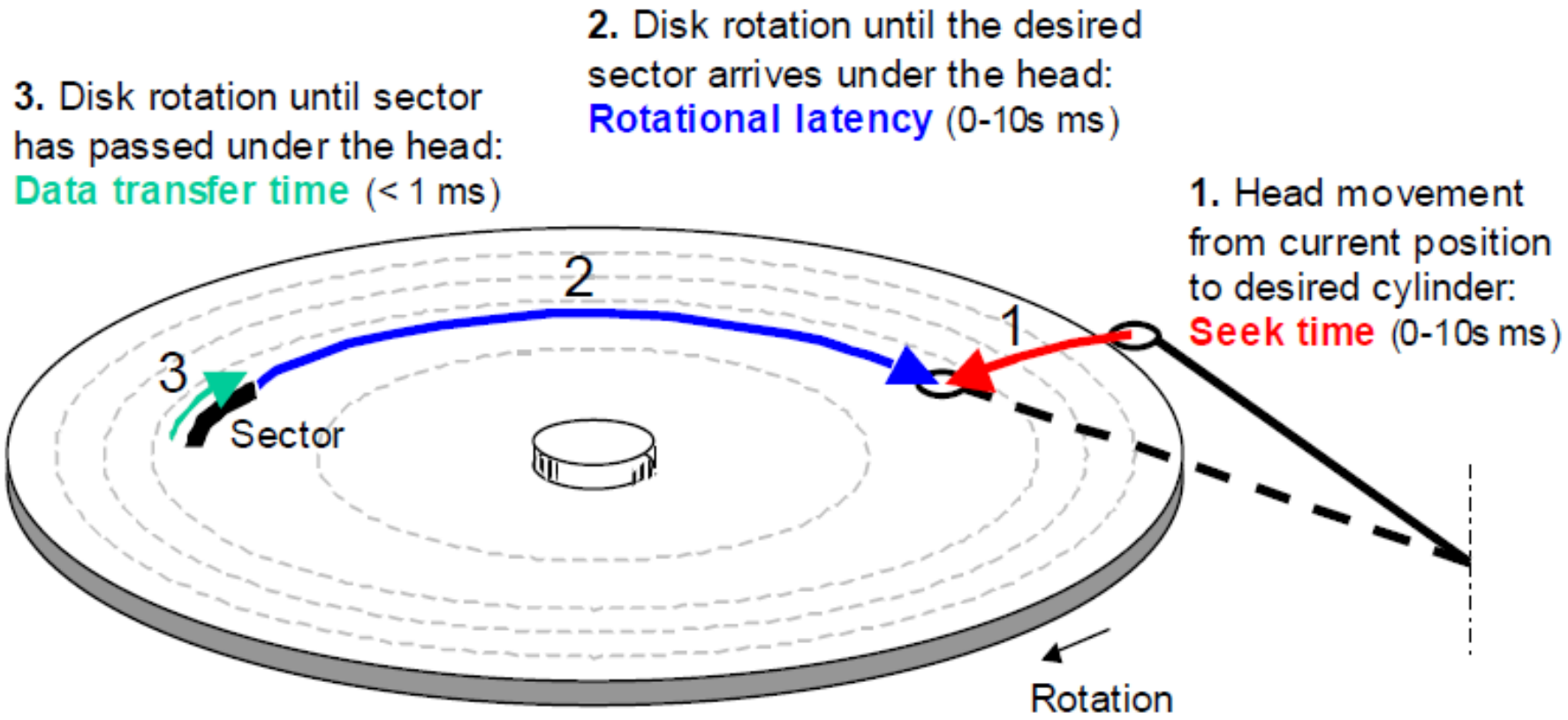
IE 411: Operating Systems

FFS, RAID

Very simple file system (vsfs)

- aka old UNIX file system
- We use part of the disk for holding data, and other portions for metadata

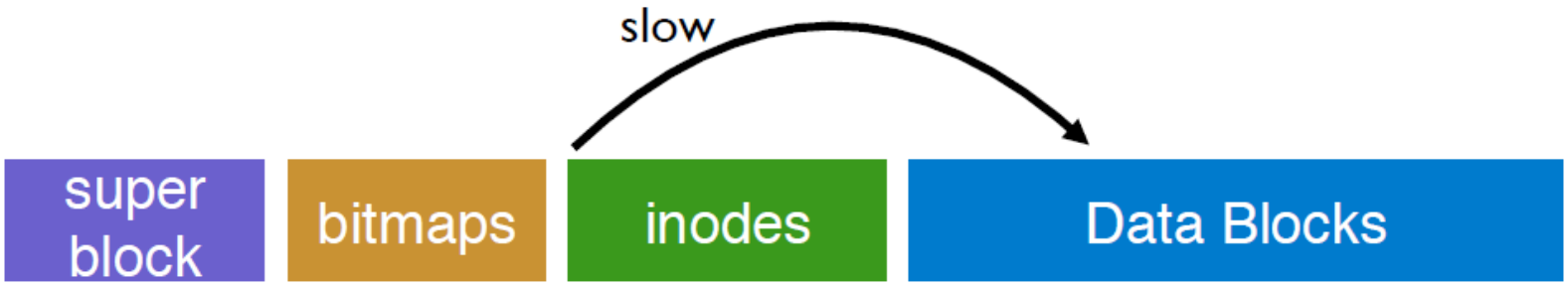
Traditional hard disk drive



Video: `https://www.youtube.com/watch?v=L0nbo1VOF4M`

Poor performance

- Recall: when traversing a file path, at each level we need to access the inode first then access the data blocks
- In the old UNIX file system, data blocks of a file were often very far from its inode, which causes expensive seek operations



Poor performance

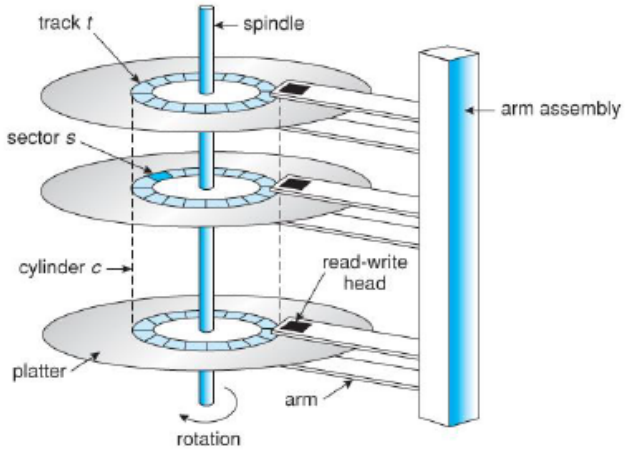
- One other problem: the file system would end up getting quite fragmented
  - consecutive file blocks not close together, pay seek cost for even sequential access

FFS: Fast File System

- Designed by McKusick, Joy, Leffler, and Fabry (1980s)
- Tries to keep the blocks that are likely to be accessed together (temporal) close to each other (spatial) on the disk

Cylinder groups

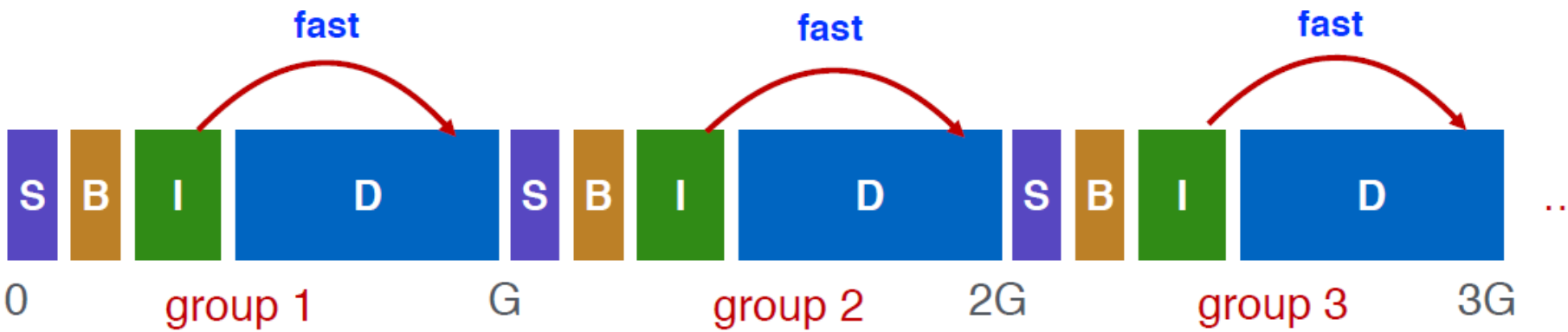
- Cylinder: set of tracks on different surfaces of a hard drive that are at the same distance from the center of the drive



- We can access any block in a cylinder without performing a seek. Next fastest place is adjacent cylinder

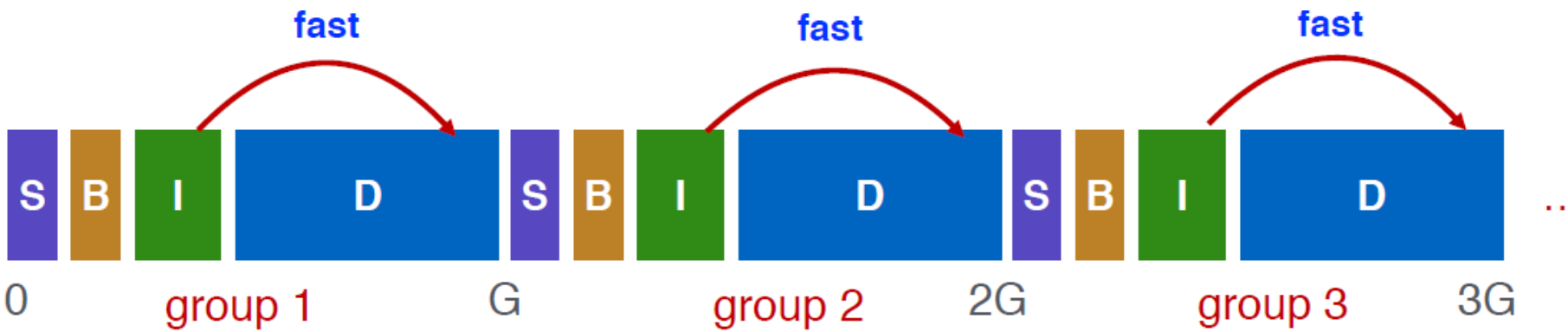
FFS disk layout

- FFS collects adjacent cylinders into a cylinder group
- Tries to keep related stuff together – within the same cylinder group



FFS disk layout

- FFS collects adjacent cylinders into a cylinder group
- Tries to keep related stuff together – within the same cylinder group



- Where should new inodes and data blocks go?

Placement heuristics

- Where should new inodes go?
  - File inode: allocate in same group as the directory inode
  - Dir inode: allocate in new group

Placement heuristics

- Where should new inodes go?
  - File inode: allocate in same group as the directory inode
  - Dir inode: allocate in new group
- Where should new data blocks go?
  - First data blocks: allocate near inode
  - Other data blocks: allocate near previous block

Example

- Two dirs

  `/a`

  `/b`

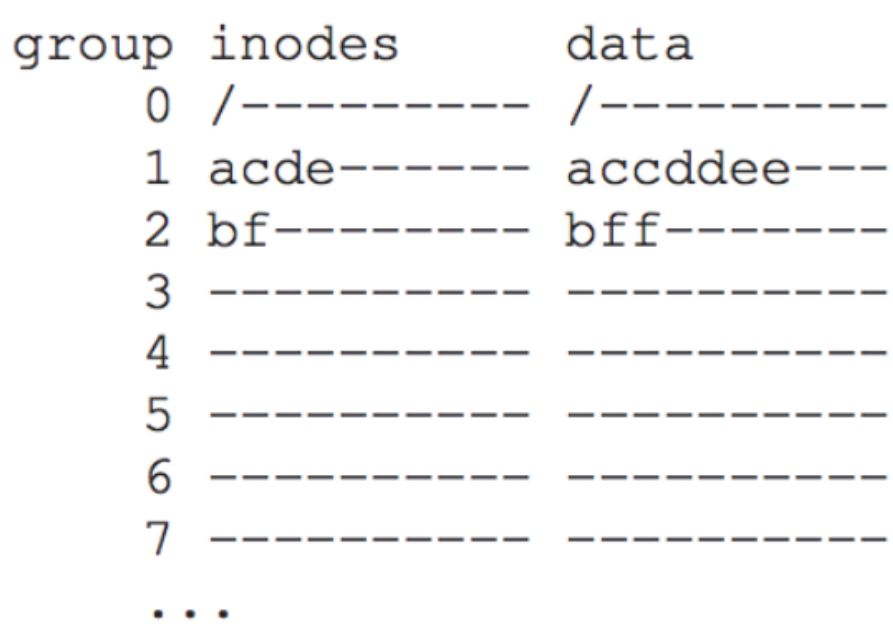
- Four files

  `/a/c`

  `/a/d`

  `/a/e`

  `/b/f`



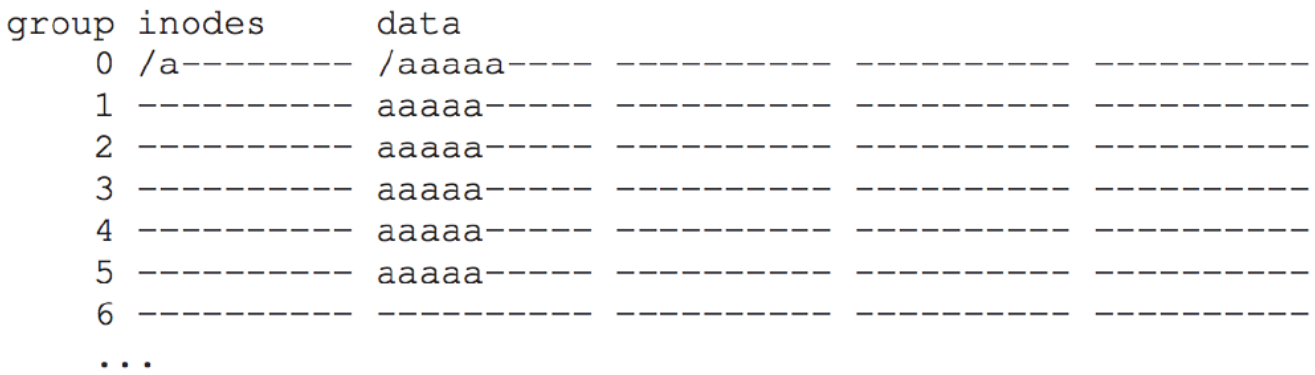
Large file exception

- A large file could take up all the space in a group
- This means we might not be able to store other files in the same directory within the same group

```
group inodes     data
    0 /a-------- /aaaaaaaaa aaaaaaaaaa aaaaaaaaaa a---------
    1 ---------- ---------- ---------- ---------- ----------
    2 ---------- ---------- ---------- ---------- ----------
    ...
```

Large file exception

- The first 12 direct blocks are allocated into the first block group
- Each subsequent indirect block, and all the blocks it pointed to are allocated in another block group



RAID

- Hardware with its own memory, processor, and firmware dedicated to the task of managing a collection of hard disks
- Better performance

  We can perform parallel I/O operations across multiple disks
- More capacity
  - We can use the disks together to increase our overall capacity, but treat it like one disk within the OS
- Reliability
  - Spreading the data over multiple disks can help avoid data loss (especially with some form of redundancy)

Measures to evaluate RAID

- Capacity
  - While we have multiple disks, some raid configurations do not give us access to the complete storage space
- Reliability
  - How many disk faults/failures can we tolerate before we cannot recover?
- Performance
  - With more disks comes the opportunity for parallelism
  - However, some configurations have more overhead as well

RAID-0: Striping

- Distributes data blocks across the disks in a round-robin fashion

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

Each row is called a stripe

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

Another way of striping. Difference?

- Not a "true" RAID because it is not fault-tolerant

Evaluating RAID-0

- N: total number of disks
- B: total number of blocks in a disk
- Capacity: $N \star B$ (the upper bound)

Evaluating RAID-0

- N: total number of disks
- B: total number of blocks in a disk
- Capacity: $N \star B$ (the upper bound)
- Reliability: 0
  - data is lost whenever any disk fails

How to map?

- Given logical address A:
  - Disk $= \ldots$
  - Offset $= \ldots$

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

How to map?

- Given logical address A:
  - Disk = A % disk_count
  - Offset = A / disk_count

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

Mapping Example: Find Block 13

- Given logical address 13:
  - Disk $= 13 \% 4 = 1$
  - Offset $= 13 / 4 = 3$

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

RAID-1: Mirroring

- Keeps two physical copies of each block
- To read block 5, can choose either Disk 2 or Disk 3
- To write block 5, write to both disks 2 and 3
  - note: the disks can be accessed in parallel

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

Evaluating RAID-1

- Capacity: $N \star B/2$
- Reliability
  - 1 (no data loss if any one disk fails)

Evaluating RAID-1

- Capacity: $N \star B/2$
- Reliability
  - 1 (no data loss if any one disk fails)
  - up to $N/2$, depending on which disks fail

RAID-4: parity

- Use less space to achieve redundancy, compared to mirroring
- One disk is dedicated as the parity disk

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

Parity function: XOR example

C0	C1	C2	C3	P
0	0	1	1	XOR(0,0,1,1) = 0
0	1	0	0	XOR(0,1,0,0) = 1

Parity function: XOR example

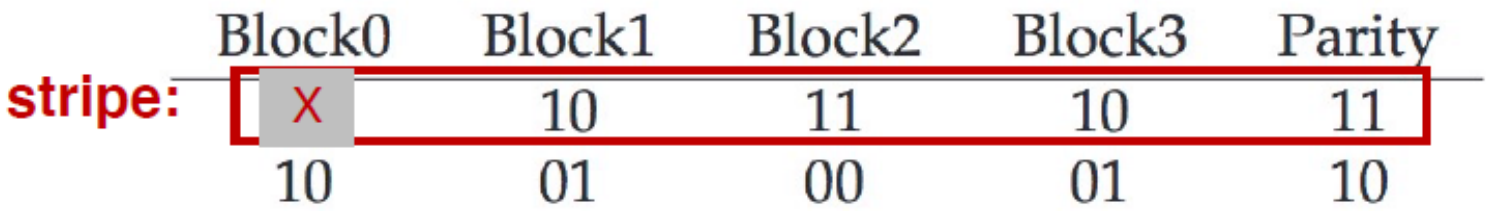
C0	C1	C2	C3	P
0	0	1	1	XOR(0,0,1,1) = 0
0	1	0	0	XOR(0,1,0,0) = 1

- XOR function:
  - $P = 0$: The number of 1s in a stripe must be an even number
  - $P = 1$: The number of 1s in a stripe must be an odd number

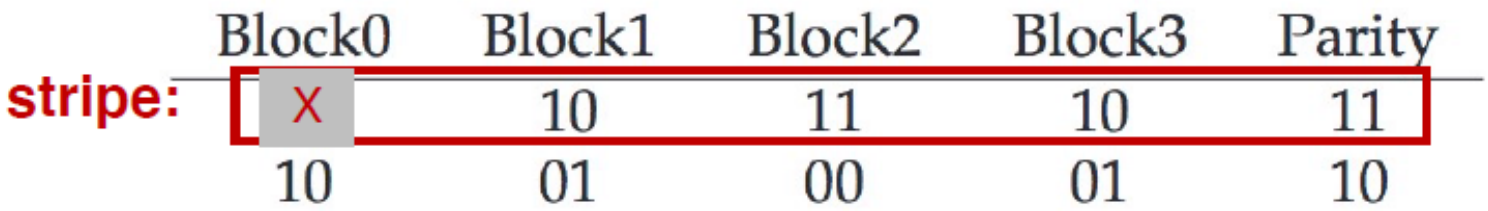
Parity function: XOR example

	Block0	Block1	Block2	Block3	Parity
stripe:	00	10	11	10	11
	10	01	00	01	10

Parity function: XOR example



Parity function: XOR example



Block0 = XOR(10, 11, 10, 11) = 00

Parity function: XOR example

	Block0	Block1	Block2	Block3	Parity
stripe:	00	10	11	10	11
	10	01	00	01	10

Block0 = XOR(10,11,10,11) = **00**

RAID-4: write operation

- For write, how to calculate the new parity block?

RAID-4: write operation

- For write, how to calculate the new parity block?
- Method 1: additive parity
  - Read all other data blocks in a stripe in parallel
  - XOR these with the new block to form a new parity block
  - Write the new data block and new parity block

RAID-4: write operation

- Method 2: subtractive parity
  - Read only the old data block D_{old} and the old parity block P_{old} to be updated
  - Compute the new parity block: $P_{new} = (D_{new} \oplus D_{old}) \oplus P_{old}$
  - Write the new data block and new parity block to disks

Evaluating RAID-4

- Capacity: $(N-1) \star B$
- Reliability: 1 (single disk failure)

RAID-4: a problem

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

- Suppose we are writing to both block 4 and block 13 simultaneously, since they are on different disks we expect good performance due to parallelism

RAID-4: a problem

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

- Suppose we are writing to both block 4 and block 13 simultaneously, since they are on different disks we expect good performance due to parallelism
- N.B. both writes cause writes on the parity blocks on Disk 4, which spoiled the parallelism

RAID-4: a problem

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

- Suppose we are writing to both block 4 and block 13 simultaneously, since they are on different disks we expect good performance due to parallelism
- N.B. both writes cause writes on the parity blocks on Disk 4, which spoiled the parallelism
- the disk with parity blocks is so frequently written that it becomes the bottleneck

RAID-5: Rotating Parity

- Distributes the parity blocks to different disks

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

- Identical to RAID-4 in capacity and reliability