香港中文大學
The Chinese University of Hong Kong

CSCI5550 Advanced File and Storage Systems

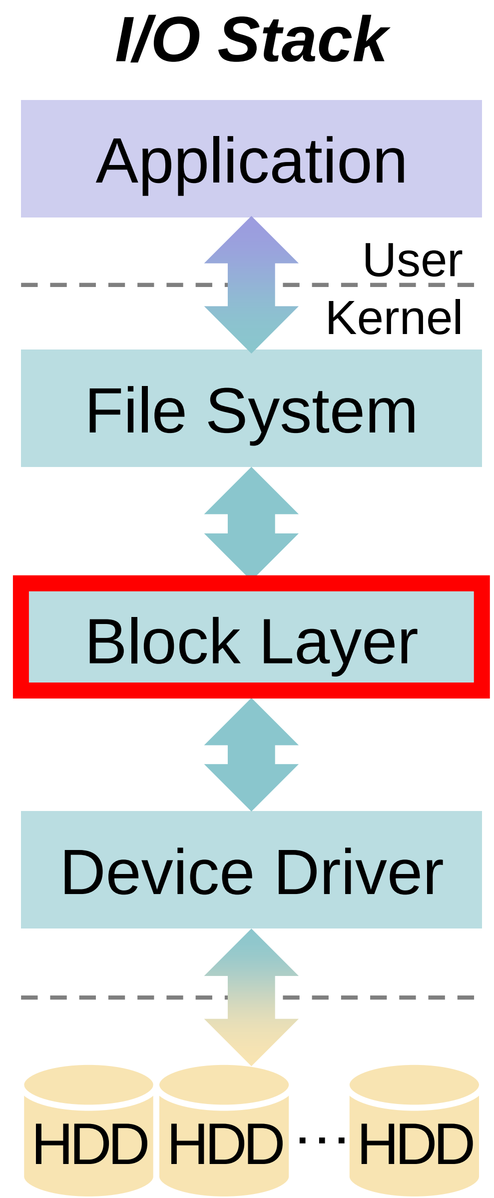
Lecture 02: RAID and Data Integrity

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

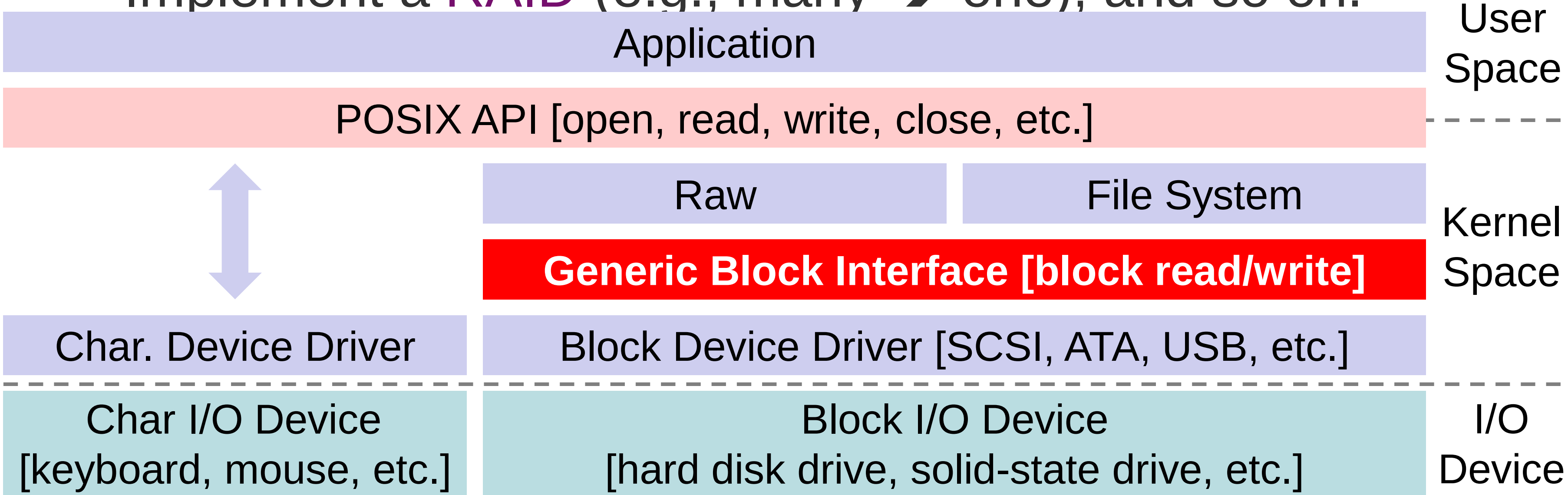
Outline

- Generic Block Layer
- Redundant Arrays of Inexpensive Disk
  - RAID Interface and Internals
  - Fault Model: Fail-Stop
  - RAID Levels and Analysis
    - Capacity, Reliability, and Performance
  - RAID Reconstruction
- Data Integrity
  - Other Disk Failure Modes and Handling
    - Latent Sector Error
    - Corruption
    - Lost Writes
    - Scrubbing



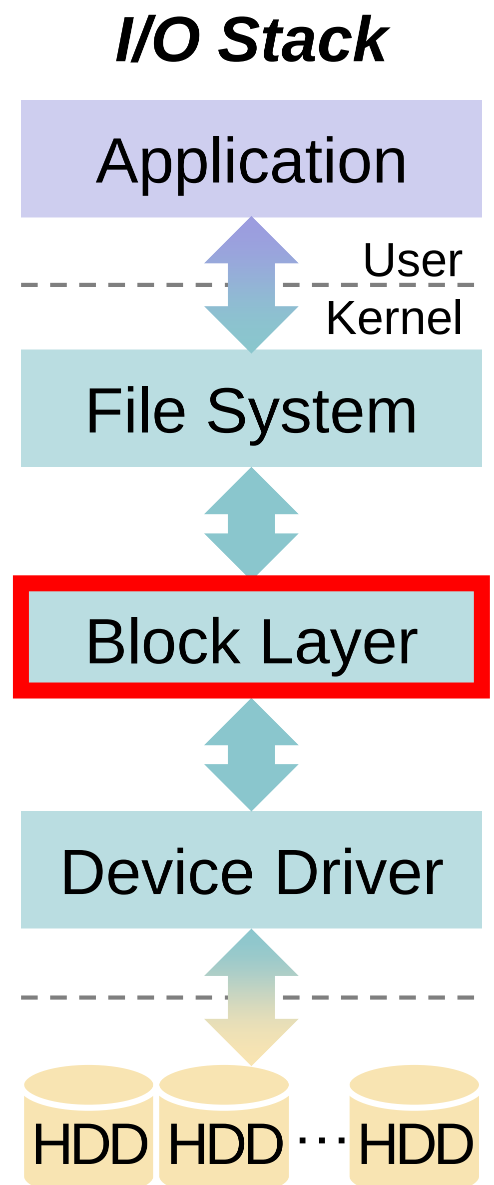
Generic Block Layer

- **Generic Block Layer**: A kernel component that handles the requests for all block devices in blocks.
- Thanks to this abstraction, the kernel may easily:
  - Schedule I/O requests for I/O devices (e.g., HDD);
  - Implement data buffer to keep data blocks to optimize I/O;
  - Manage logical volumes (e.g., one disk → many volumes);
  - Implement a RAID (e.g., many → one), and so on.



Outline

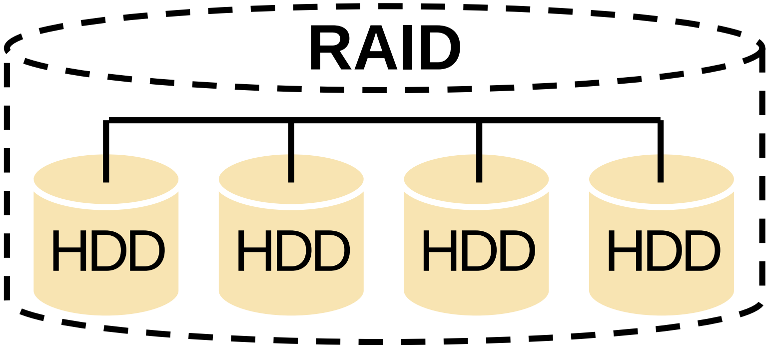
- Generic Block Layer
- Redundant Arrays of Inexpensive Disk
  - RAID Interface and Internals
  - Fault Model: Fail-Stop
  - RAID Levels and Analysis
    - Capacity, Reliability, and Performance
  - RAID Reconstruction
- Data Integrity
  - Other Disk Failure Modes and Handling
    - Latent Sector Error
    - Corruption
    - Lost Writes
    - Scrubbing



Redundant Arrays of Inexpensive Disks

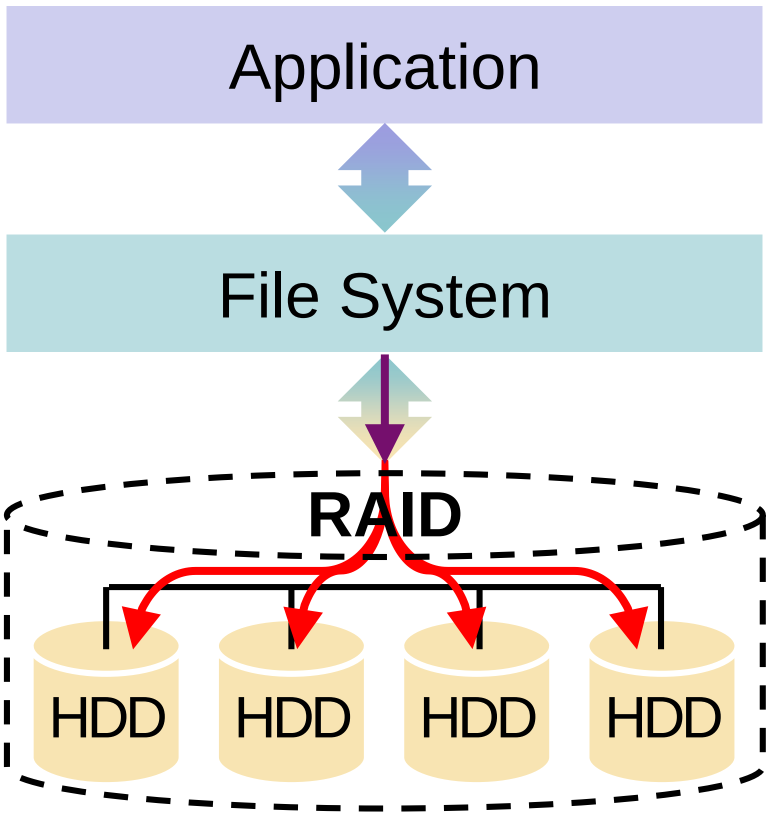
- **RAID**: Redundant Arrays of Inexpensive Disks
  - Aggregates multiple physical disks as one logical (and bigger) one.
  - Developed by researchers at Berkeley in late 80s.

- RAID offers the following advantages transparently with the same interface as a single disk.
  - **Capacity**: more disks
  - **Reliability**: fault tolerance by maintaining redundancy
  - **Performance**: parallelism



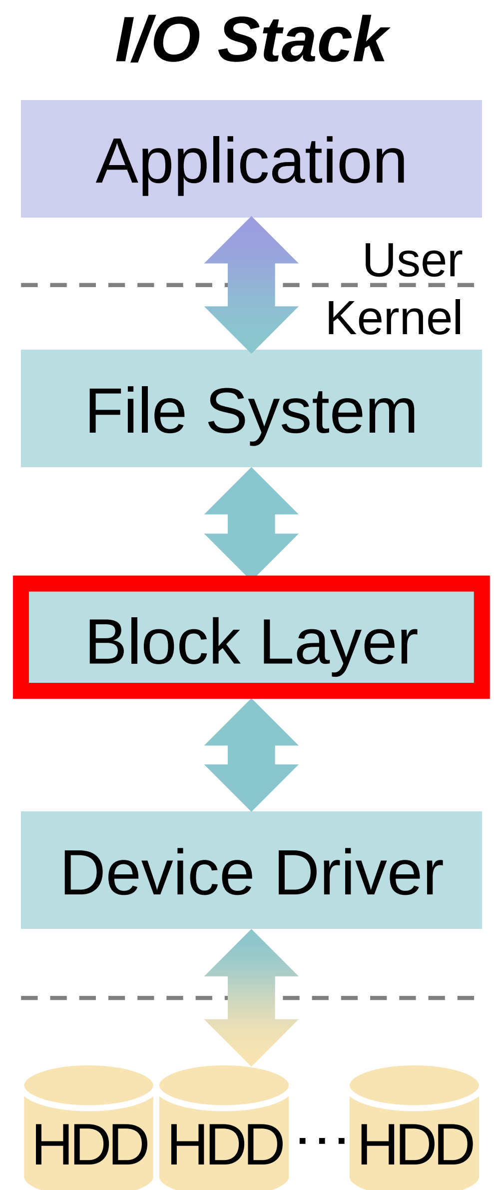
RAID Interface and Internals

- RAID converts a logical I/O from the file system into one or multiple physical I/Os to disks(s).
- RAID is often built as a separate hardware box, with a standard bus to a host.
  - A microcontroller to run firmware for RAID operations.
  - Some memory to buffer data blocks as they are read/written.
  - Specialized logic to perform parity (redundancy) calculation.
- RAID can be also built by software (e.g., `mdadm`).



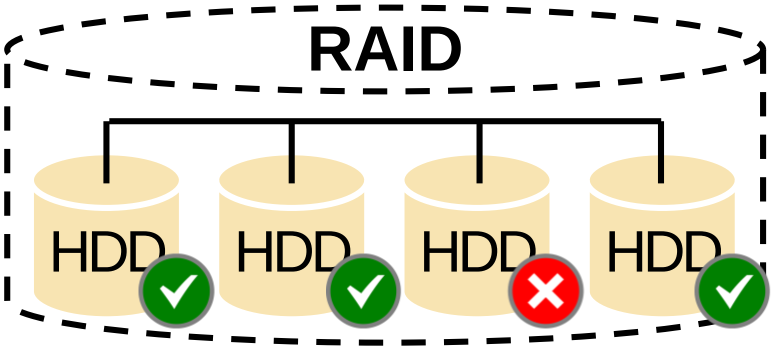
Outline

- Generic Block Layer
- **R**edundant **A**rrays of **I**nexpensive **D**isk
  - RAID Interface and Internals
  - Fault Model: Fail-Stop
  - RAID Levels and Analysis
    - Capacity, Reliability, and Performance
  - RAID Reconstruction
- Data Integrity
  - Other Disk Failure Modes and Handling
    - Latent Sector Error
    - Corruption
    - Lost Writes
    - Scrubbing



Fault Model

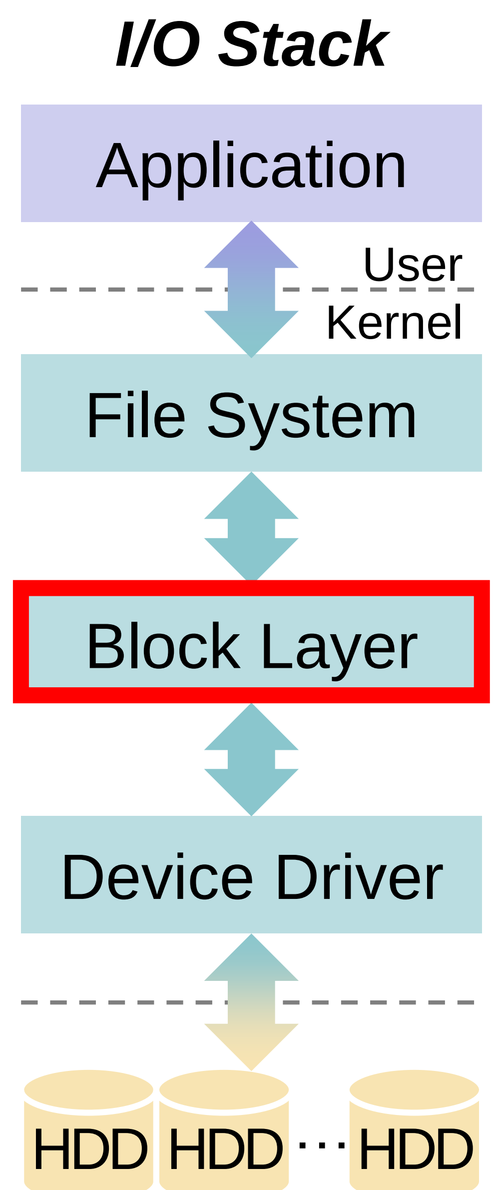
- RAID is designed to detect and recover from certain kinds of disk faults.
- Let's begin with the simplest **fail-stop** fault model:
  - If a disk is working, all its data can be read/written.
  - If a disk fails, all its data are permanently lost.
    - RAID controller can immediately detect if a disk fails.



- In practice, disk failures can be more complex (e.g., bad sectors in a working disk or "silent" failures).

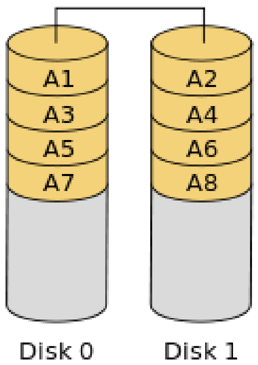
Outline

- Generic Block Layer
- <u>R</u>edundant <u>A</u>rrays of <u>I</u>nexpensive <u>D</u>isk
  - RAID Interface and Internals
  - Fault Model: Fail-Stop
  - RAID Levels and Analysis
    - Capacity, Reliability, and Performance
  - RAID Reconstruction
- Data Integrity
  - Other Disk Failure Modes and Handling
    - Latent Sector Error
    - Corruption
    - Lost Writes
    - Scrubbing

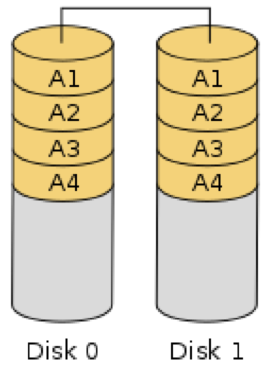


Basic RAID Levels: A Glance

RAID-0
Striping

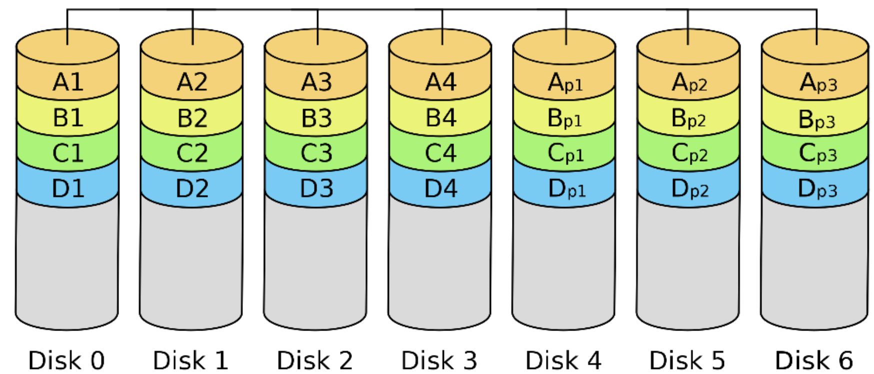


RAID-1
Mirroring



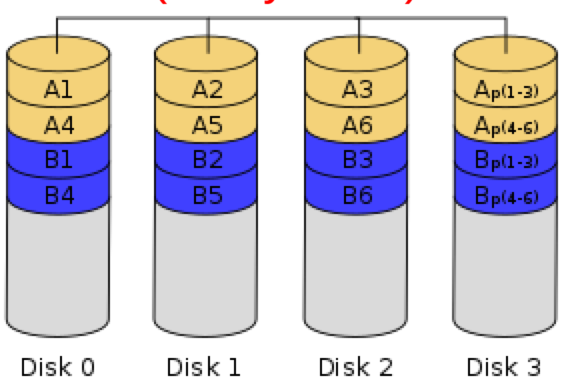
~~RAID-2~~
~~Striping at bit-level~~

~~(rarely used)~~

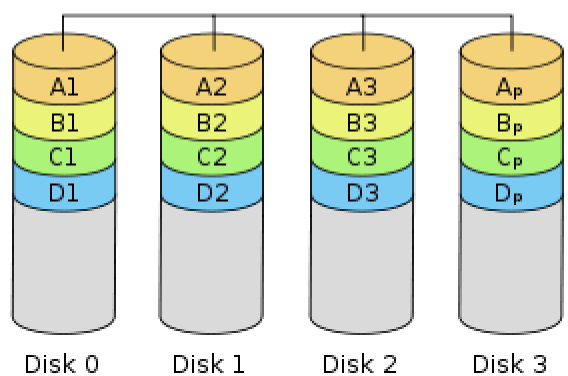


~~RAID-3~~
~~Striping at byte-level~~

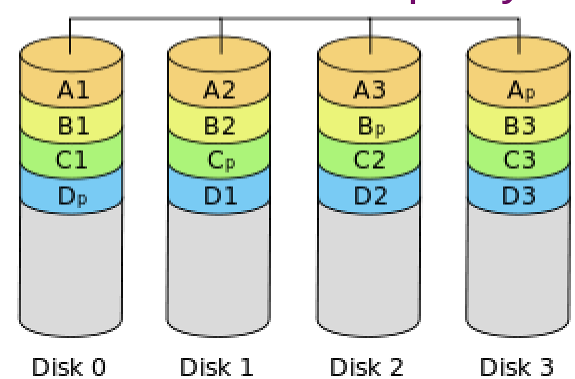
~~(rarely used)~~



RAID-4
Striping at block-level



RAID-5
Striping at block-level with distributed parity



https://en.wikipedia.org/wiki/Standard_RAID_levels

RAID Analysis: Three Aspects

- **Capacity**
  - The effective storage size in number of blocks
    - Let **N** be the total number of disks in RAID.
    - Let **B** be the total number of blocks in a single disk.
- **Reliability**
  - The number of tolerable disk failures.
- **Performance**
  - **Single-Request Latency:** $T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$
    - Let **T** be the latency that a request to a single disk would take.
  - **Steady-State Throughput:** $Rate_{I/O} = Size_{Transfer} \div T_{I/O}$
    - Let **S** and **R** be the single-disk transfer bandwidths (or rates) under sequential and random workloads, respectively (S >> R).

Terminologies

- Disks organize data in **blocks** (e.g., 4KB).
- RAID usually distributes data across disks in units of **chunks**, which is composed of one or more **blocks**.
  - Chunk size mostly affects performance of RAID.
    - Small chunk size increases parallelism of reads/writes.
    - Large chunk size reduces positioning time of disks.
  - Let the chunk size be the block size in our analysis.
- A **stripe** refers to the same row of chunks.

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

Stripe **Block** **Chunk**

chunk size: 2 blocks

RAID-0 (Striping)

- **RAID-0** distributes data blocks across disks in a round-robin fashion (without any redundancy!).
  - **Capacity**: $N * B$ (the upper bound)
  - **Reliability**: 0 (no fault tolerance)
  - **Performance**:
    - Read/Write Latency: T (the same as in a single disk)
      - The I/O request is simply redirected to one of the disks.
    - Sequential Read/Write Throughput: $N * S$ (full bandwidth)
    - Random Read/Write Throughput: $N * R$ (full bandwidth)

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

Discussion

- Question: How to do address mapping from a logical block address to a physical block address in RAID?

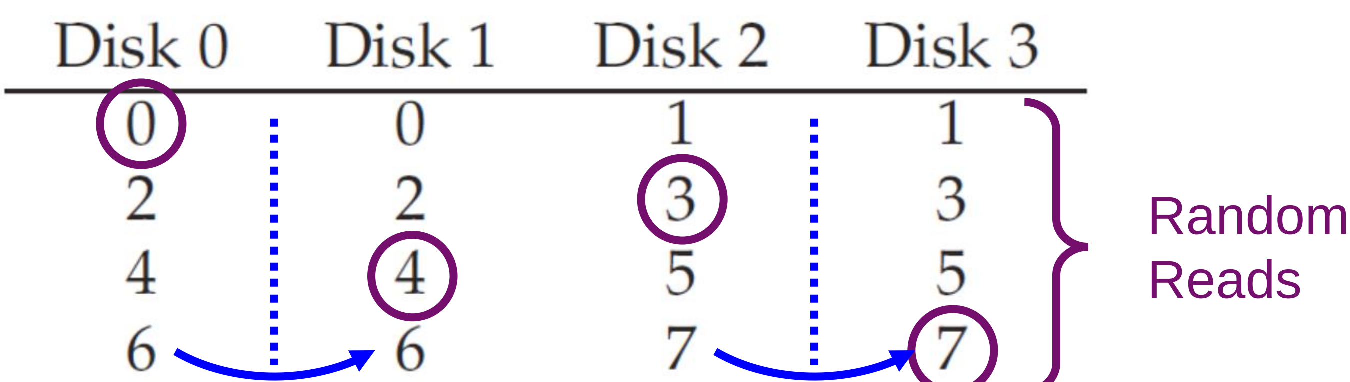
- **Answer**:
  - Let the chunk size be the block size.
  - Let LBA be the logical block address.

$$\boxed{\begin{aligned} Disk &= LBA \,\%\, number\_of\_disks \\ Offset &= LBA \,/\, number\_of\_disks \end{aligned}}$$

- Bonus: What about a general chunk size (i.e., a chunk is of multiple blocks)?

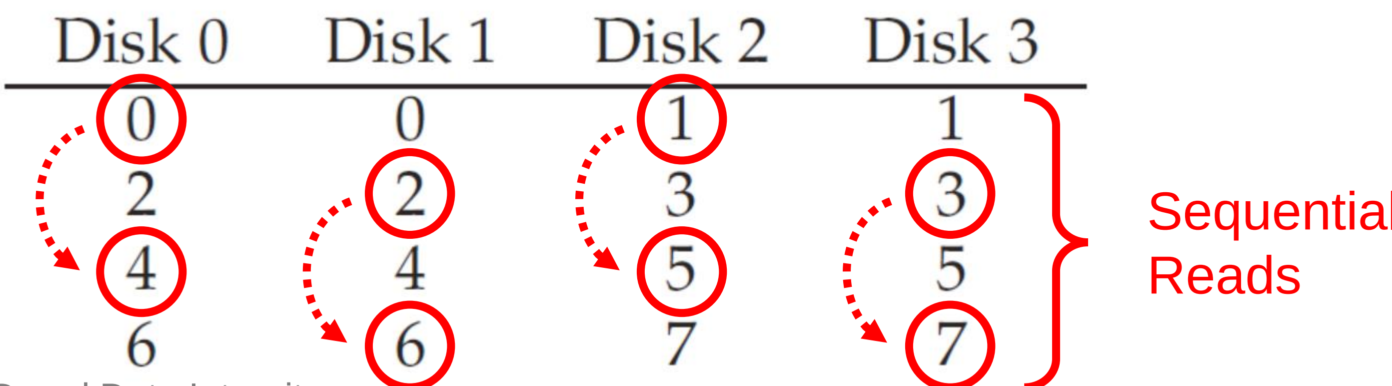
RAID-1 (Mirroring) (1/2)

- **RAID-1** keeps **two** physical copies for every block.
  - **Capacity**: $\underline{N * B/2}$ (very expensive!)
  - **Reliability**: $\underline{1}$ (any one for certain); $\underline{up\ to\ N/2}$ (if lucky!)
  - **Performance**:
    - Read/Write Latency: $\underline{T}$ (the same as in a single disk)
      - Read from one hard copy; Write to two hard copies in parallel.
    - Random Write Throughput: $\underline{N * R/2}$ (all in use, but half effective)
    - Random Read Throughput: $\underline{N * R}$ (**possible** to reach full bandwidth)
      - E.g., randomly read blocks 0, 3, 4, and 7



RAID-1 (Mirroring) (2/2)

- **RAID-1** keeps **two** physical copies for every block.
  - **Capacity**: $N * B/2$ (very expensive!)
  - **Reliability**: 1 (any one for certain); $up\ to\ N/2$ (if lucky!)
  - **Performance (cont'd)**:
    - Sequential Write Throughput: $N * S/2$ (all in use, but half effective)
    - Sequential Read Throughput: $N * S/2$
      - Why not $N * S$ (similar to random read throughput)?
      - Answer: Each disk receive a request for **every other block**. While it is rotating over the skipped block, it is not delivering effective bandwidth.



Discussion

- In RAID-1, updates (i.e., writes) to both copies of each logical block must be **consistent** (or **atomic**, i.e., both copies are updated or neither is updated).

- Question: How to guarantee the consistency when a power loss (or system crash) occurs?

- **Answer**: Write-Ahead Log (will be discussed later)
  - Do the log before updating two disks.
  - Use a small amount of non-volatile, battery-backed RAM for better logging performance.
  - Replay the log if a crash occurs.

RAID-4 (1/5)

- **RAID-4** adding redundancy (known as **parity**) to a disk for each stripe.
  - 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 must withstand the loss of any one block in a stripe.
    - Parity can be computed via **bitwise XOR**.
    - Recovery? Block0 = Block1 XOR Block2 XOR Block3 XOR Parity

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

RAID-4 (2/5)

- **RAID-4** adding redundancy (known as **parity**) to a disk for each stripe.
  - **Capacity**: $N - 1$ (one dedicated parity disk)
  - **Reliability**: 1 (any one for certain and no more)
  - **Performance:**
    - Random Read Throughput: $(N - 1) \ast R$ (parity disk has no effect!)
    - Sequential Read Throughput: $(N - 1) \ast S$ (parity disk has no effect!)

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

RAID-4 (3/5)

- RAID-4 adding redundancy (known as **parity**) to a disk for each stripe.
  - **Performance:**
    - Sequential Write Throughput: $(N-1) * S$ (parity disk has no effect!)
    - How to do full-stripe write under RAID-4?
      ① Buffer all data blocks of a stripe
      ② Compute the parity block
      ③ Write all data and parity blocks in parallel

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

RAID-4 (4/5)

- **RAID-4** adding redundancy (known as **parity**) to a disk for each stripe.
  - **Performance:**
    - Random writes need to update both data and parity blocks.

      Approach 1) Additive Parity (as known as reconstruct-writes)

      ① Read all other data blocks in a stripe in parallel

      ② XOR those with the new block to form a new parity block

      ③ Write the new data block and new parity block to disks

      Approach 2) Subtractive Parity (as known as read-modify-writes)

      ① Read only the old data block to be updated and old parity block

      ② Compute the new parity block: $P_{new} = (D_{new} \wedge D_{old}) \wedge P_{old}$

      ③ Write the new data block and new parity block to disks
    - Random Write Throughput: $R/2$ (using subtractive parity)
      - Each random write triggers two reads and two writes.
        - » The reads can happen in parallel, as can the writes.

Discussion

- Question: What is the tradeoff between additive parity and subtractive parity?

- **Answer**: Additive parity incurs more I/Os if the number of disks is large; vice versa for subtractive parity.

- Bonus: What is the cross-over point?
  - That is, how many disks would need so that the additive method performs fewer I/Os than subtractive method.

RAID-4 (5/5)

- **RAID-4** adding redundancy (known as **parity**) to a disk for each stripe.
  - **Performance:**
    - Read Latency: T (the same as in a single disk)
      - A single read is just redirected to a single disk.
    - Write Latency: $T * 2$ (twice in a single disk)
      - A single write needs two reads and two writes (subtractive parity).
        - » The reads can happen in parallel, as can the writes.

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

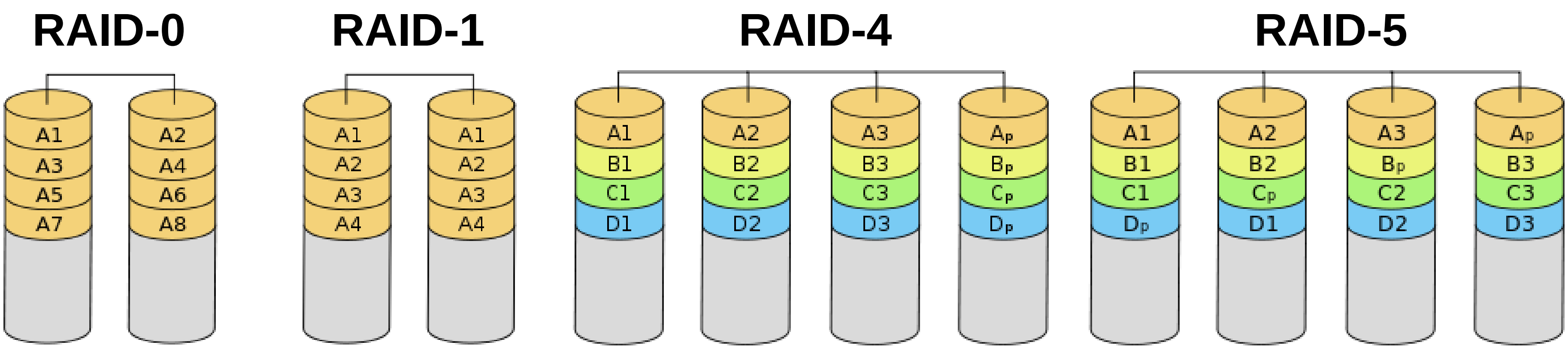
RAID-5

- **RAID-5** rotates parity blocks across stripes.
  - Other operations remain the same as RAID-4.

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 the following: capacity, reliability, read/write latency, and sequential r/w throughput.
    - Random Read Throughput: $N * R$ (**possible** to reach full bandwidth).
    - Random Write Throughput: $N/4 * R$ (improved greatly over RAID-4).
      - Assume a large number of random writes keeps all disks evenly busy.
      - The factor of four loss: Each RAID-5 write still need four I/O operations.

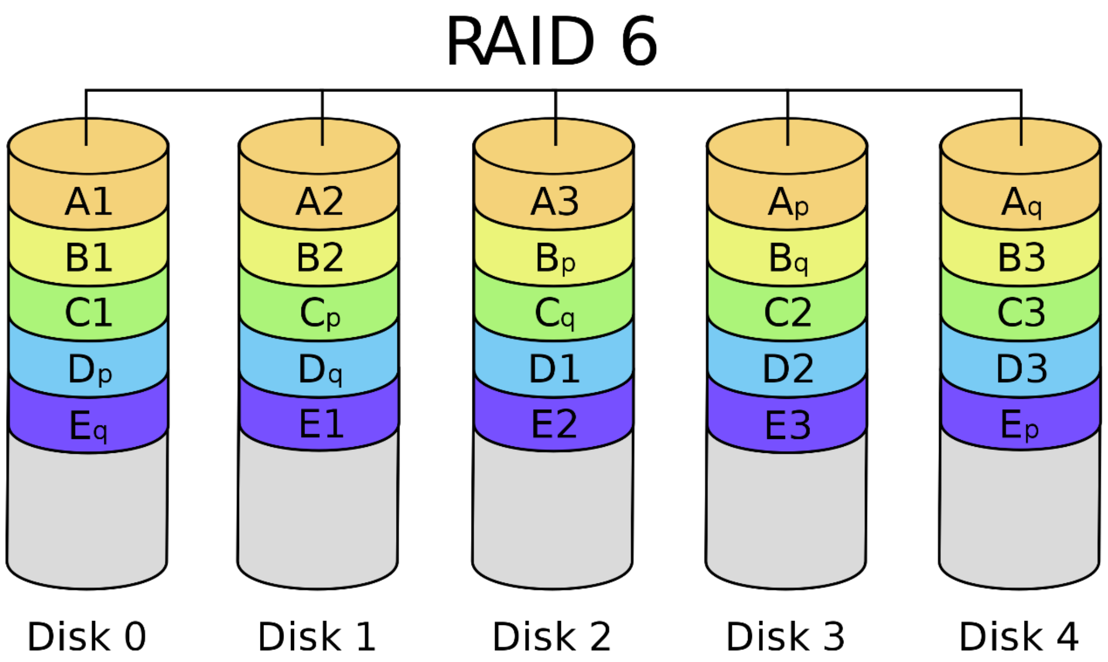
RAID Comparison: A Summary



	RAID-0	RAID-1	RAID-4	RAID-5
Capacity	$N \cdot B$	$(N \cdot B)/2$	$(N-1) \cdot B$	$(N-1) \cdot B$
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	$N \cdot R$	$(N/2) \cdot R$	$\boxed{\frac{1}{2} \cdot R}$	$\frac{N}{4} R$
Latency				
Read	T	T	T	T
Write	T	T	$2T$	$2T$

Other Basic RAID Levels

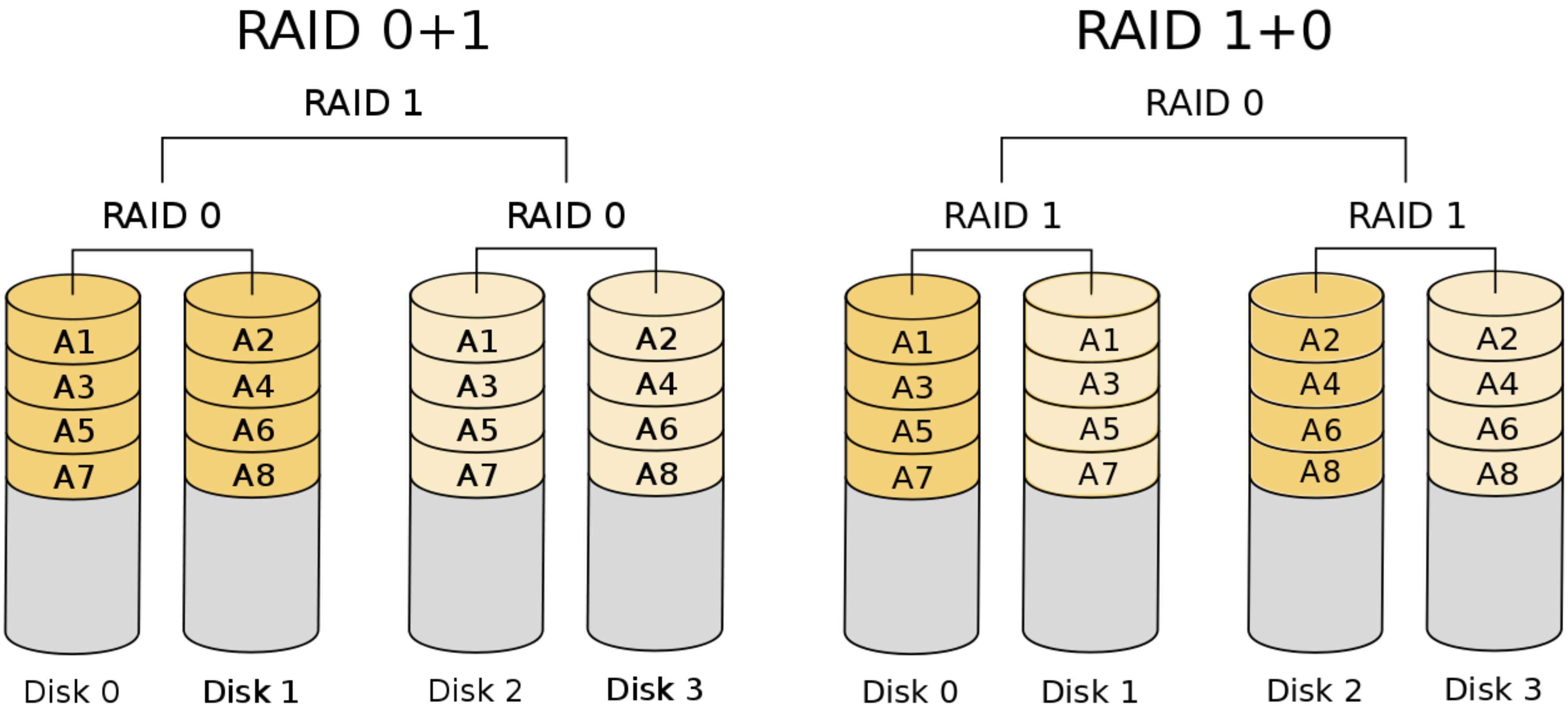
- **RAID-6** can tolerate multiple disk faults by
  – Introducing more redundancy (i.e., parity blocks);
  – Using more powerful error correction code (e.g., Reed-Solomon code).



https://en.wikipedia.org/wiki/Standard_RAID_levels

Advanced RAID Levels

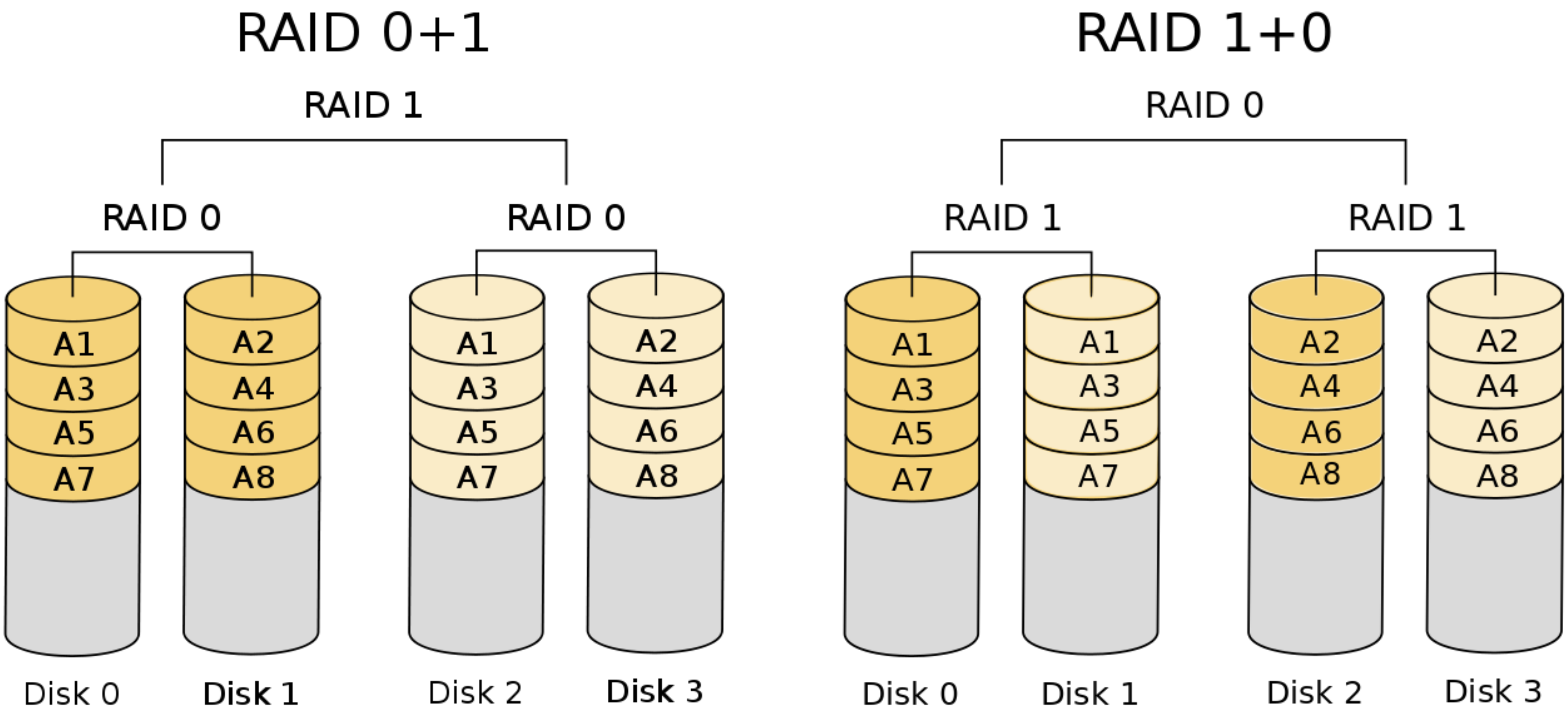
- **Nested RAID** (or **Hybrid RAID**): Combines two or more of basic RAID levels (i.e., RAID-0~RAID-6).
  - To gain performance, additional redundancy or both, as a result of combining properties of different RAID layouts.



https://en.wikipedia.org/wiki/Nested_RAID_levels

Discussion

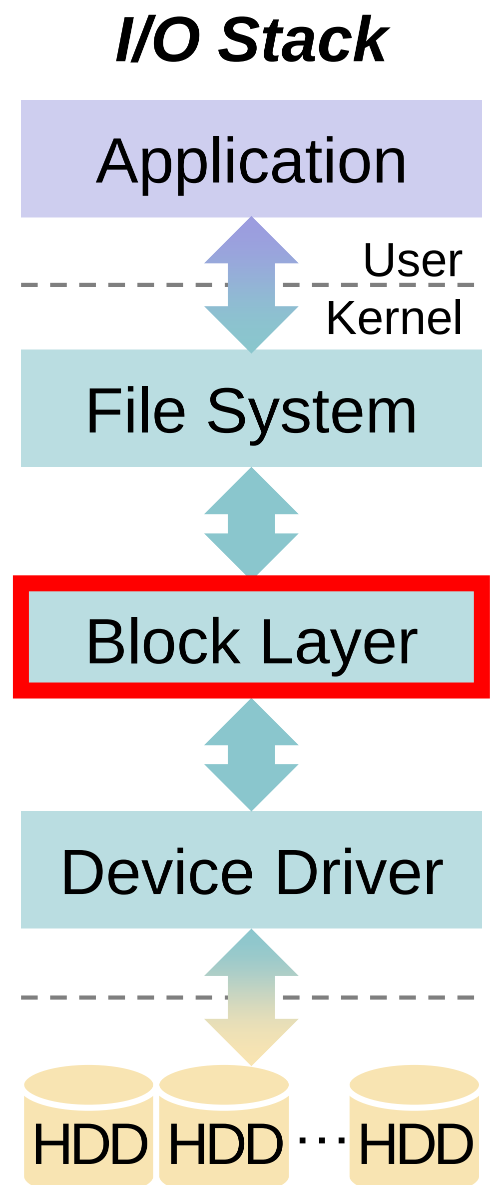
- Question: Which one is better? RAID-01 or RAID-10?



- **Answer**: The reliability of RAID-10 is better than RAID-01 in more failure scenarios.

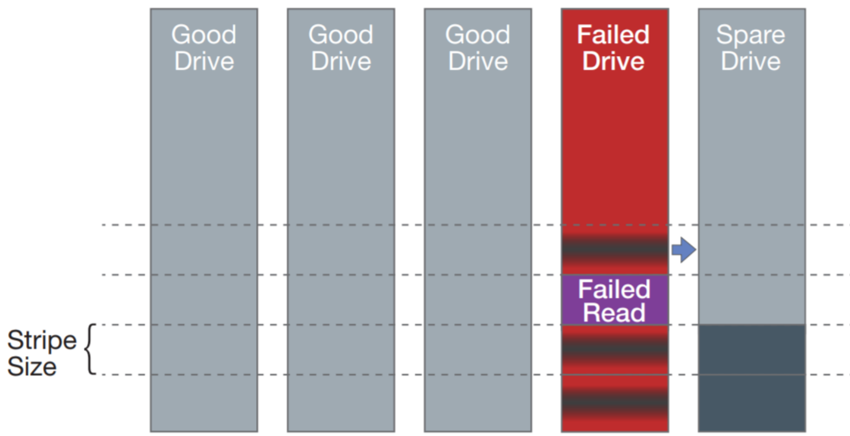
Outline

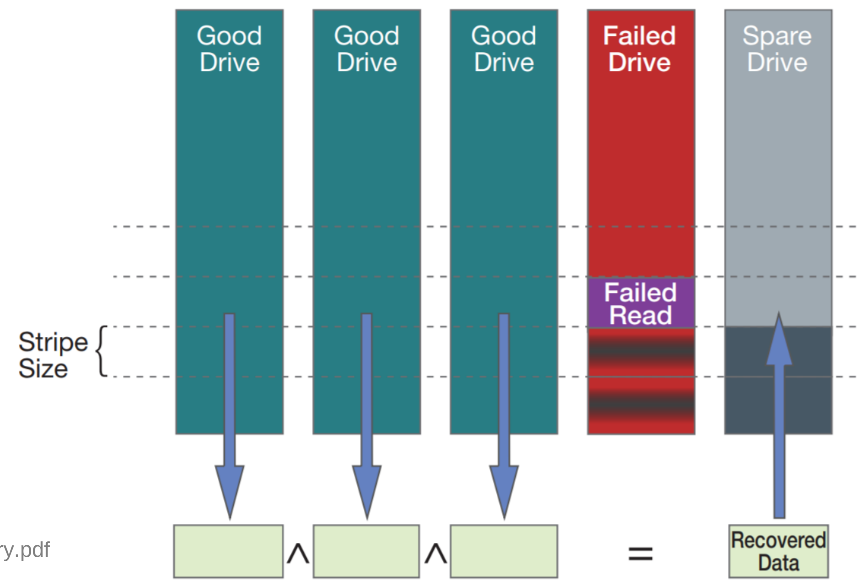
- Generic Block Layer
- **R**edundant **A**rrays of **I**nexpensive **D**isk
  - RAID Interface and Internals
  - Fault Model: Fail-Stop
  - RAID Levels and Analysis
    - Capacity, Reliability, and Performance
  - RAID Reconstruction
- Data Integrity
  - Other Disk Failure Modes and Handling
    - Latent Sector Error
    - Corruption
    - Lost Writes
    - Scrubbing



RAID Reconstruction

- The RAID system needs to be **reconstructed** when a disk fails.
  - The failed disk needs to be replaced by a spare one.
    - **Hot Spare**: enables a RAID system to automatically failover.
    - **Cold Spare**: resides in the RAID but requires manual intervention.
  - The entire spare disk needs to be rebuilt by using either failed disk or other healthy disks in the RAID system.

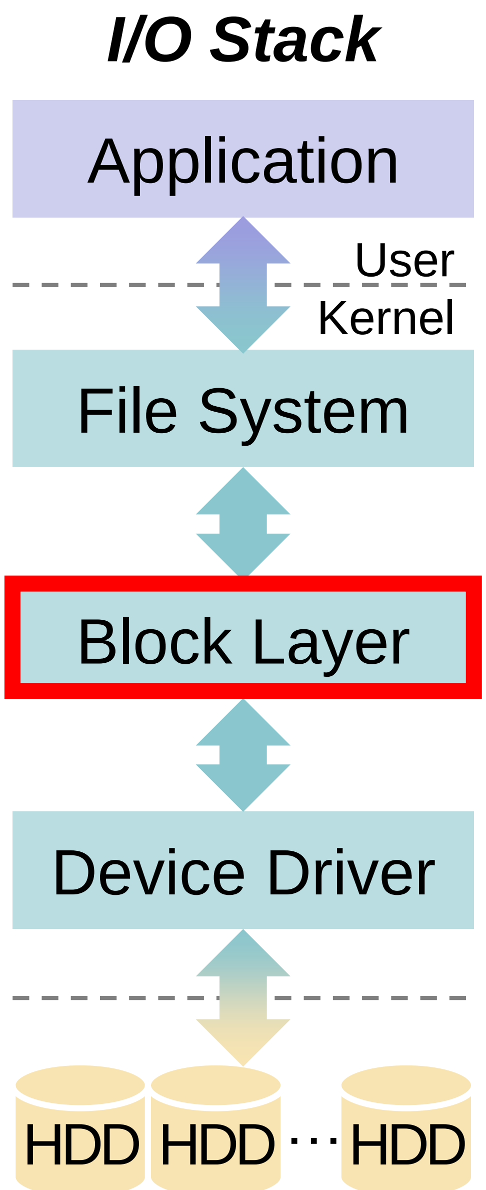




https://www.seagate.com/files/staticfiles/docs/pdf/whitepaper/tp620-1-1110us-reducing-raid-recovery.pdf

Outline

- Generic Block Layer
- Redundant Arrays of Inexpensive Disk
  - RAID Interface and Internals
  - Fault Model: Fail-Stop
  - RAID Levels and Analysis
    - Capacity, Reliability, and Performance
  - RAID Reconstruction
- Data Integrity
  - Other Disk Failure Modes and Handling
    - Latent Sector Error
    - Corruption
    - Lost Writes
    - Scrubbing



Data Integrity and Other Failure Modes

- So far, we only consider the **fail-stop** fault model.
  - Either the entire disk is working, or it fails completely.
- The **data integrity** should be further ensured.
  - The data put into the system must be the same as returned.
- The **fail-partial** fault model is more practical.
  - Disks seem working, but some blocks can't be used.
- Two common types of single-block failures.
  - **Latent Sector Errors**: Blocks are inaccessible or damaged.
  - **(Silent) Corruptions**: Blocks hold wrong content.

	Cheap (e.g., SATA)	**Costly** (e.g., SCIS)
Latent Sector Errors	9.40%	1.40%
Corruptions	0.50%	0.05%

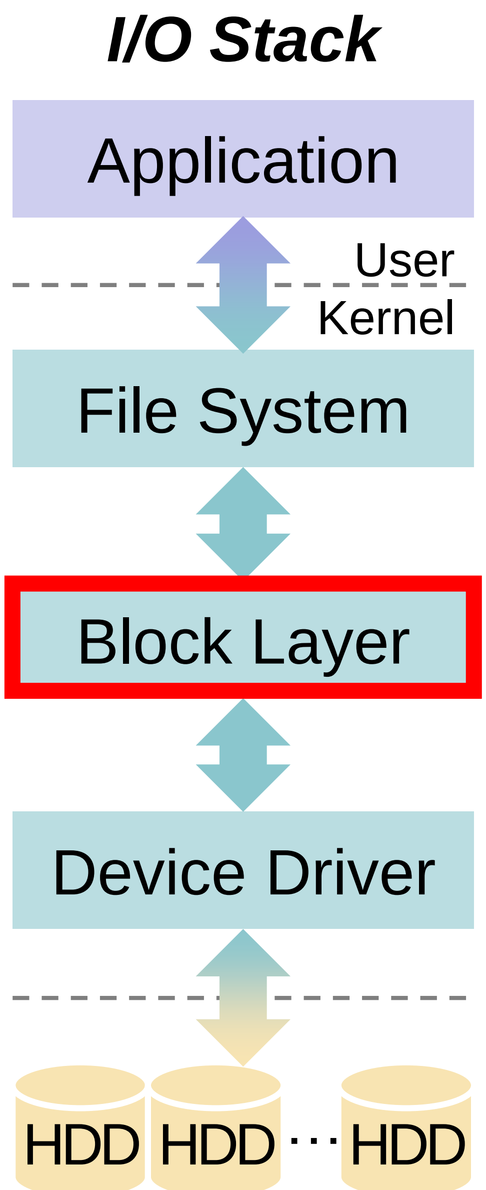
Failure percentages of 1.5 million drives over a 3-year span.

Latent Sector Errors

- **LSEs** arise when a disk sector (or group of sectors) has been damaged in some way such as:
  - Head crash damages disk surface, making bits unreadable.
  - Cosmic rays flip bits, leading to incorrect contents.
- LSEs can be easily detected when accessing a block.
  - If a block cannot be accessed, the disk returns an error.
  - If a block can be accessed but the in-disk **error correcting codes (ECC)** cannot fixed LSEs, the disk returns an error.
    - ECCs associate the data with some redundancy for **detecting** and **recovering** (usually a fixed number of) error bits.
    - Classical ECCs include Golay, BCH, Hamming codes, etc.
- Most RAID levels (*except RAID-0*) can recover LSEs by leveraging the redundancy.

Outline

- Generic Block Layer
- Redundant Arrays of Inexpensive Disk
  - RAID Interface and Internals
  - Fault Model: Fail-Stop
  - RAID Levels and Analysis
    - Capacity, Reliability, and Performance
  - RAID Reconstruction
- Data Integrity
  - Other Disk Failure Modes and Handling
    - Latent Sector Error
    - Corruption
    - Lost Writes
    - Scrubbing

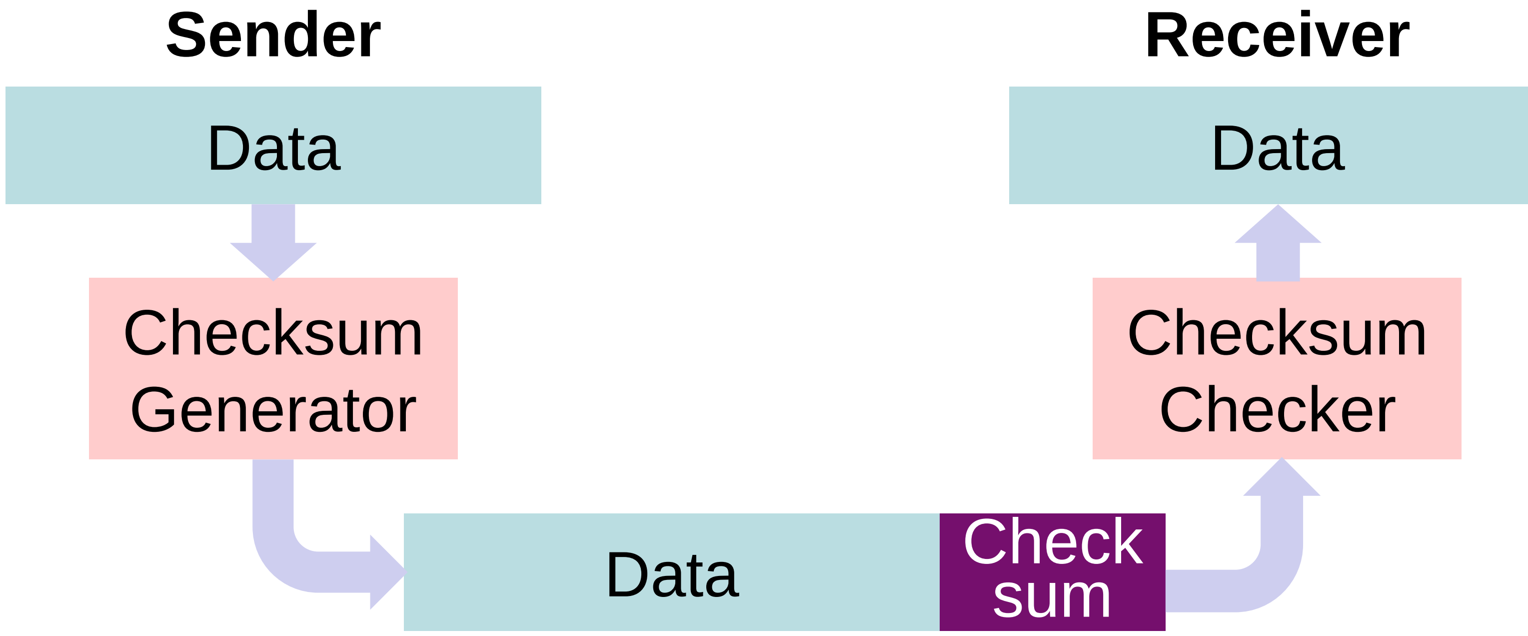


Corruptions

- **Corruptions** refer to the cases where a block becomes corrupt but not detectable such as:
  - A heathy block may get corrupted when it is transferred between the host and the disk across a faulty bus.
  - Buggy disk firmware writes a block to the wrong location.
    - In such a case, the in-disk ECC indicates the block contents are fine, but the wrong block is returned to users.
- These types of faults are particularly insidious because they are silent faults.
  - The disk itself has no idea when returning the faulty data.
- Once it is known that a particular block is corrupt, recovery is the same as before.
  - We need a way to **detect** corruptions.

Detecting Corruption: The Checksum

- **Checksum**: ensures **data integrity** despite corruption.
  - It is simply a small summary of the data contents (e.g., 4-8 bytes), computed from a chunk of data (e.g., 4KB).
  - The corruption can be detected only if the checksum does NOT match the data contents.
    - Why? Beyond the checksum capability or checksum corrupted.



The Simplest Checksum: XOR

- One simple way: **exclusive or (XOR)**.
- Considering a 4-byte checksum over a data block of 16 bytes (lined up in groups of 4 bytes per row):
  - The checksum is computed by XOR'ing over each column.

```
         00110110 01011110 11000100 11001101
         10111010 00010100 10001010 10010010
         11101100 11101111 00101100 00111010
    XOR) 01000000 10111110 11110110 01100110
    ----------------------------------------
Checksum 00100000 00011011 10010100 00000011
```

- **Collision**: Different blocks have the same checksum.
- **Limitation**: Even number of error bits in a column.
- **Bonus**: Can you do "error correction" with XOR?

Fletcher and CRC Checksum(s)

- **Fletcher Checksum**: iteratively computes two check bytes, namely s1 and s2, as follows:
  - Assume a block `D` consists of bytes `d1`, `d2`, …, `dn`.

```
for (i=1; i<=n; ++i) {
    s1 = (s1 + di) mod 255;
    s2 = (s2 + s1) mod 255;
}
```

- **Cyclic Redundancy Check (CRC)**: divides the data block by an agreed upon value (k) and takes the remainder of this division as the checksum.
  - It is one of the most commonly-used checksums today.
- Both are good at detecting single-bit, double-bit, and even a large portion of burst errors (*think about why*).

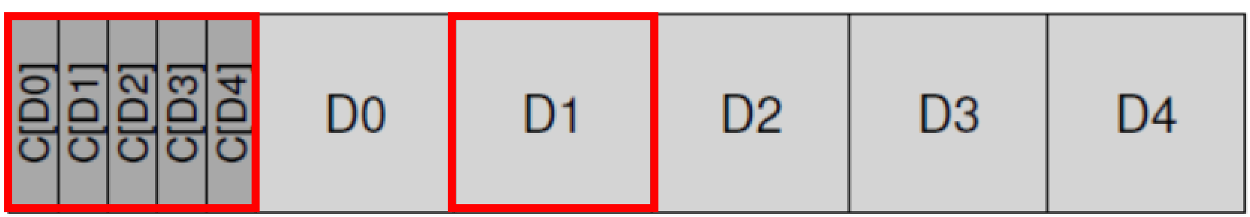
Checksum Layout

- How should checksums be stored on disk?
- Given five data blocks D0, D1, …, D4, let's call the checksum of Di as C(Di).

1) The checksum can be added next to each block:

   – Requiring disks be formatted with non-512-byte sector.

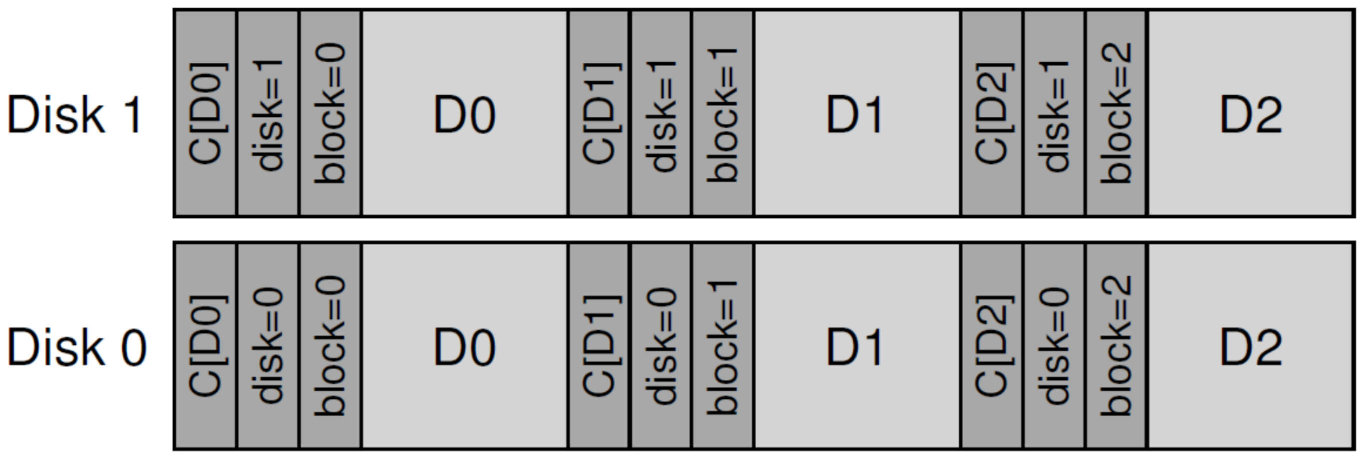
2) The checksums can be also packed into a block:



   – Working on all disks but less efficient.
      - Two writes for checksum block and the data block.

Misdirected Writes

- Recall: A corruption will occur when buggy disk firmware writes a block to the wrong location.
  - This failure mode is called a **misdirected write**.
- Solution: Adding a little more information to checksum.
  - The **physical identifier (ID)** can be used to verify whether the data chuck resides within a “correct” location.

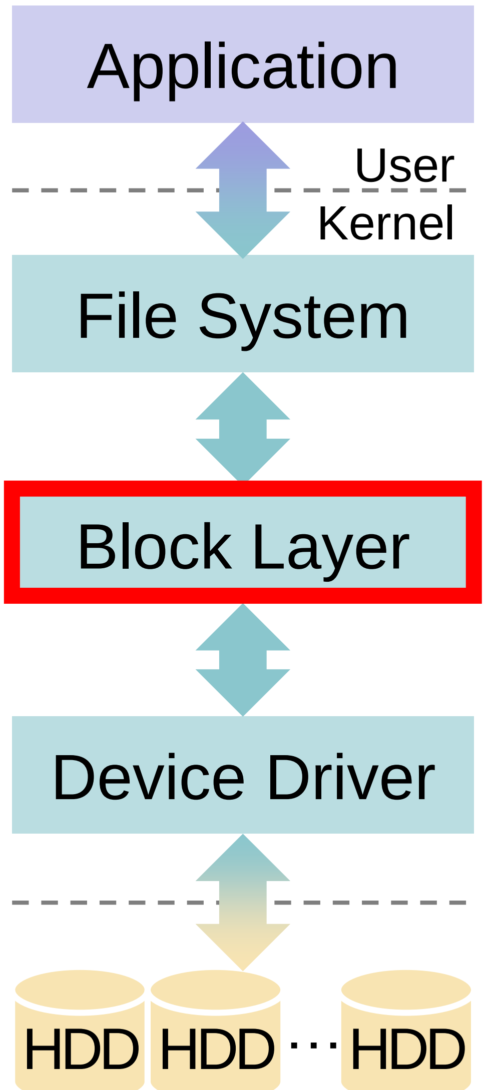


- Redundancy is always the key for both error detection (in this case) and recovery (in others such as RAID).

Outline

- Generic Block Layer
- Redundant Arrays of Inexpensive Disk
  - RAID Interface and Internals
  - Fault Model: Fail-Stop
  - RAID Levels and Analysis
    - Capacity, Reliability, and Performance
  - RAID Reconstruction
- Data Integrity
  - Other Disk Failure Modes and Handling
    - Latent Sector Error
    - Corruption
    - Lost Writes
    - Scrubbing

I/O Stack



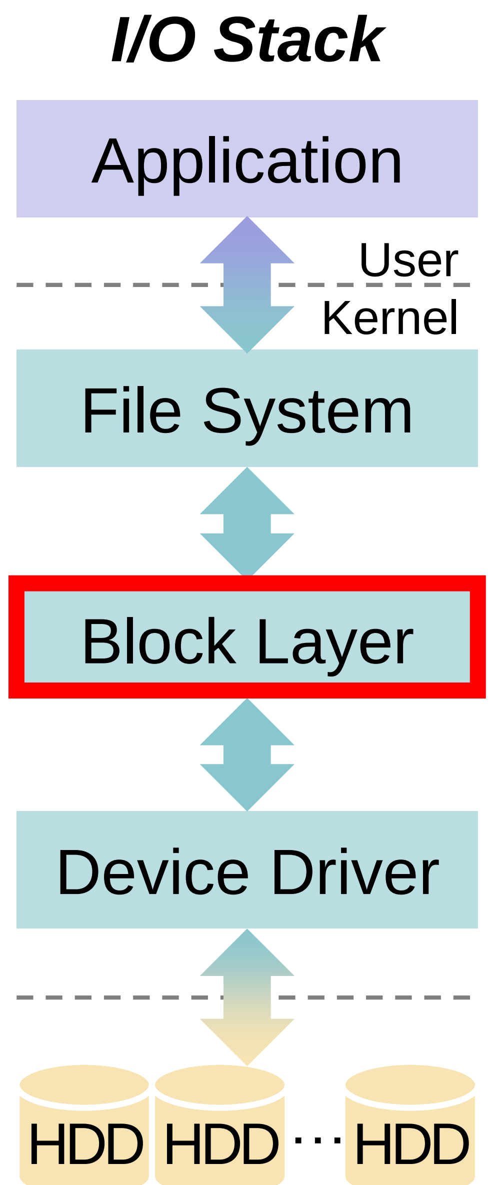
Lost Writes

- **Lost Write**: The device informs the upper layer that a write has completed but it’s not persisted.
  - Basic checksum with physical identity does NOT help.
  - The old block likely has a matching checksum, and the physical ID used above will also be correct.

- Possible Solutions
  - Read-after-Write
    - It may double the I/O.
  - Maintain additional checksum elsewhere in the system
    - It still can’t solve the problem if both writes are lost.

Outline

- Generic Block Layer
- Redundant Arrays of Inexpensive Disk
  - RAID Interface and Internals
  - Fault Model: Fail-Stop
  - RAID Levels and Analysis
    - Capacity, Reliability, and Performance
  - RAID Reconstruction
- Data Integrity
  - Other Disk Failure Modes and Handling
    - Latent Sector Error
    - Corruption
    - Lost Writes
    - Scrubbing



Scrubbing

- Unchecked data are problematic for a reliable system.
  - Bit rot could accumulate and eventually become unrecoverable anymore.

- **Disk scrubbing** is a periodic process that:
  - Reads through every block;
  - Checks whether checksums are still valid;
  - Repairs the problem if needed;
  - Scheduled on a nightly or weekly basis.

Summary

- Generic Block Layer
- **R**edundant **A**rrays of **I**nexpensive **D**isk
  - RAID Interface and Internals
  - Fault Model: Fail-Stop
  - RAID Levels and Analysis
    - Capacity, Reliability, and Performance
  - RAID Reconstruction
- Data Integrity
  - Other Disk Failure Modes and Handling
    - Latent Sector Error
    - Corruption
    - Lost Writes
    - Scrubbing

I/O Stack