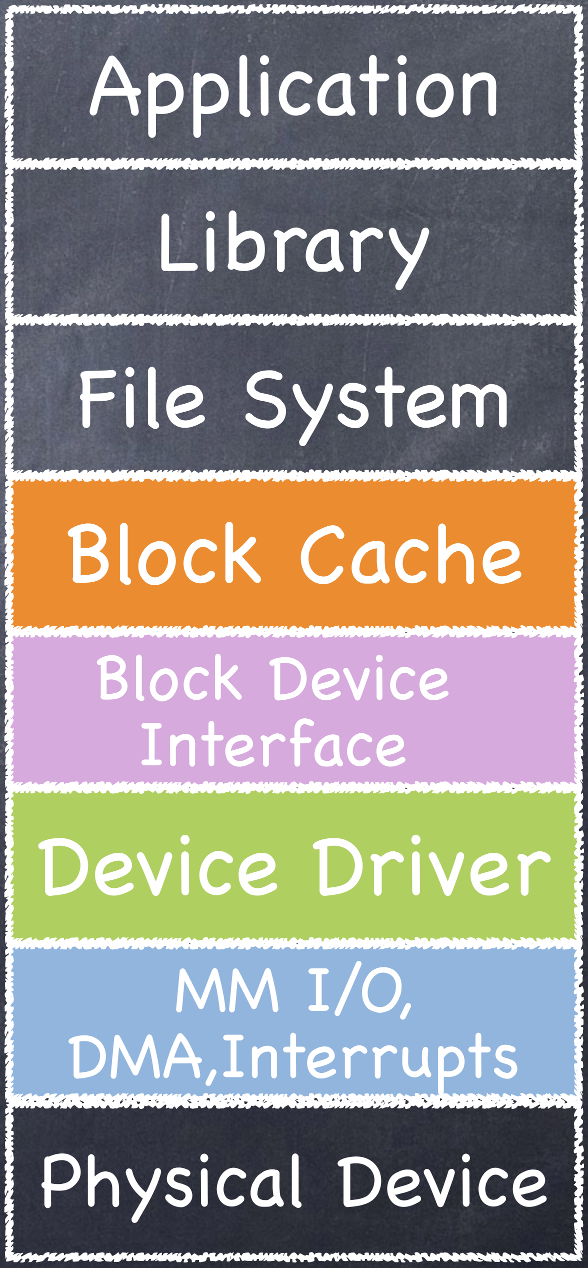
Storage stack: More on RAID

Announcements

- Grading completed
  - All regrade requests completed (HW3 regrade requests due on 11/20)
- HW4
  - Please submit the first 2 questions by 11/20 (File system and RAID)
  - Q3 can be submitted along with HW5, if you prefer
- Prelim 2
  - In-class; open-* (same as Prelim1); infinite time
  - Everything up to today's lecture (cumulative)
  - Those taking makeup exam should have received details over email
- Preparation for Prelim2
  - Practice prelim 2 and solutions released
  - Sunday: review session at 1PM (will be recorded); zoom link on Ed
  - Many extra office hours to help you prepare for Prelim2

Recall: The Storage Stack

- I/O systems are accessed through a series of layered abstractions
  - Caches blocks recently read from disk
  - Buffers recently written blocks
  - Single interface to many devices, allows data to be read/written in fixed sized blocks
  - Translates OS abstractions and hw specific details of I/O devices
  - Control registers, bulk data transfer, OS notifications



What have we discussed in the storage stack so far? [1]

- Structure of the file system
  - Files and directories divided into "blocks"
    - Blocks are allocated on physical storage device
      - in a contiguous allocation, linked structure, or indexed structure
      - Explored different tradeoffs
    - Inodes store "pointers" to physical locations of individual blocks
      - Along with other file metadata
      - Different Inode structure for different block allocation mechanisms
    - Superblock stores the metadata for the file system
      - e.g., where is inode table
  - Files, directories, blocks, inodes, superblocks—all stored on physical device
    - Can be indexed for improved performance

What have we discussed in the storage stack so far? [2]

- What semantics/abstractions should file systems provide?
  - Files as a storage mechanism
    - Everything on previous slide
  - Consistent updates
    - Upon one or more writes, file system should be "consistent"
    - If data block written, it should be readable and not be garbage
      - Transactions (worst performance, lowest recovery overheads)
      - Journaling (okay performance, okay recovery overheads)
      - Log-structured file system (best performance, worst recovery overheads)
- What abstractions should the storage system provide?
  - One large, fast, reliable storage system: RAID

RAID

Redundant Array of Inexpensive* Disks

* In industry, "inexpensive" has been replaced by "independent" :-)

High-level idea

- Implement the abstraction of a faster, bigger and more reliable disk using a collection of slower, smaller, and more likely to fail disks
  - different configurations offer different tradeoffs
- Key feature: transparency
  - The Power of Abstraction™
  - to the OS looks like a single, large, highly performant and highly reliable single disk
    - a linear array of blocks
    - mapping needed to get to actual disk
    - cost: one logical I/O may translate into multiple physical I/Os
- In the box:
  - microcontroller, DRAM (to buffer blocks) [sometimes non-volatile memory, parity logic]

Failure Model

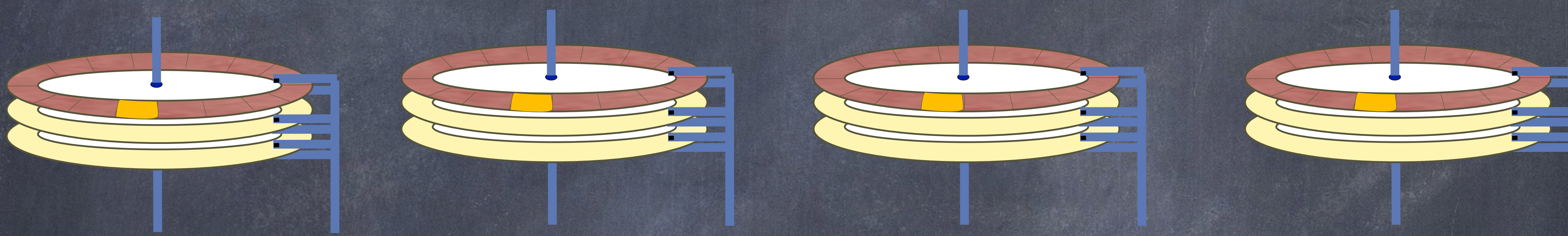
- RAID adopts the strong, somewhat unrealistic Fail-Stop failure model (electronic failure, wear out, head damage)
  - component works correctly until it crashes, permanently
    - Storage device is either working: all blocks can be read and written
    - or has failed: it is permanently lost
  - failure of the component is immediately detected
    - RAID controller can immediately observe a disk has failed and accesses return error codes
- In reality, storage devices can also suffer from isolated failures
  - Permanent: physical malfunction (wear out, scratches, contaminants)
  - Transient: data is corrupted, but new data can be successfully read from/written

How to Evaluate a RAID

- Capacity
  - what fraction of the sum of the storage of its constituent disks does the RAID make available?
- Reliability
  - How many disk faults can a specific RAID configuration tolerate?
- Performance
  - Workload dependent

RAID-0: Striping

Spread blocks across disks using round robin



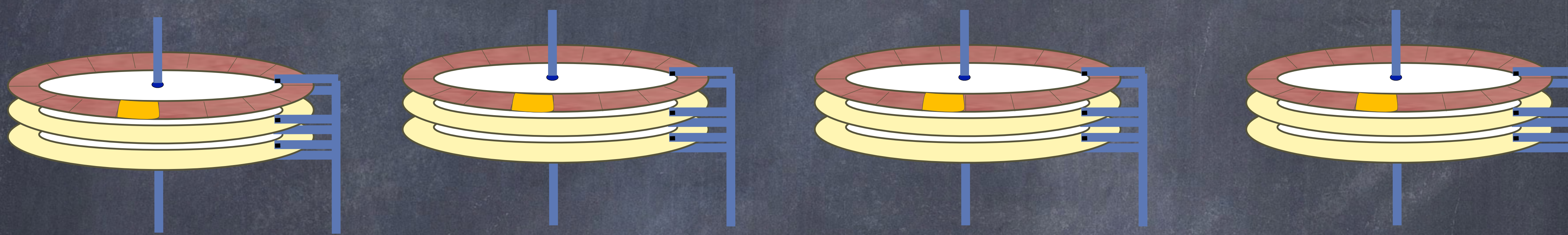
Stripe	0	1	2	3
Stripe	4	5	6	7
Stripe	8	9	10	11
Stripe	12	13	14	15

+ Excellent parallelism
    - can read/write from multiple disks

– Worst-case latency
    - wait for largest latency across all ops

RAID-0: Striping (Big Chunk Edition)

Spread blocks across disks using round robin



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

+ improve sequential throughput

— decrease parallelism

RAID-0: Evaluation

- Capacity
  - Excellent: N disks, each holding B blocks support the abstraction of a single disk with NxB blocks
- Reliability
  - Poor: Striping reduces reliability
    - Any disk failure causes data loss
- Performance
  - Workload dependent, of course
  - We'll consider two workloads
    - Sequential: single disk transfers S MB/s
    - Random: single disk transfer R MB/s
    - S >> R

RAID-0: Performance

- Single-block read/write throughput
  - about the same as accessing a single disk
- Latency
  - Read: T ms (latency of one I/O op to disk)
  - Write: T ms
- Steady-state read/write throughput
  - Sequential: N x S MB/s
  - Random: N x R MB/s

RAID-1: Mirroring

Each block is replicated twice

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

Read from any

Write to both

RAID-1: Evaluation

- Capacity
  - Poor: N disks of B blocks yield (N x B)/2 blocks
- Reliability
  - Good: Can tolerate the loss (not corruption!) of any one disk
- Performance
  - Fine for reads: can choose any disk
  - Poor for writes: every logical write requires writing to both disks
    - suffers worst-case delay of the two writes

RAID-1: Performance

- Steady-state throughput
  - Sequential Writes: N/2 x S MB/s
    - Each logical Write involves two physical Writes
  - Sequential Reads: N x S MB/s

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

Suppose we want to read

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

RAID-1: Performance

- Steady-state throughput
  - Sequential Writes: N/2 x S MB/s
    - Each logical Write involves two physical Writes
  - Sequential Reads: N x S MB/s

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

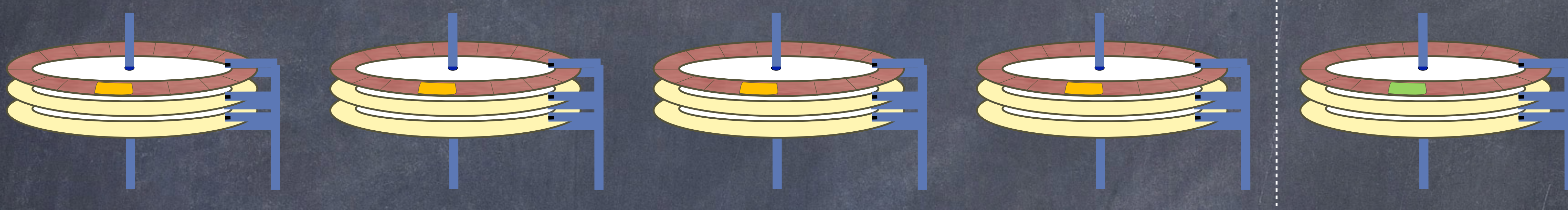
Suppose we want to read

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

  - Random Writes: N/2 x R MB/s
    - Each logical Write involves two physical Writes
  - Random Reads: N x R MB/s
    - Reads can be distributed across all disks
- Latency for Reads and Writes: T ms

RAID-4: Block Striped, with Parity

Data disks   Parity disk



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

1	1	0
0	1	0
0	0	1

1	0	0
1	1	0
0	1	1

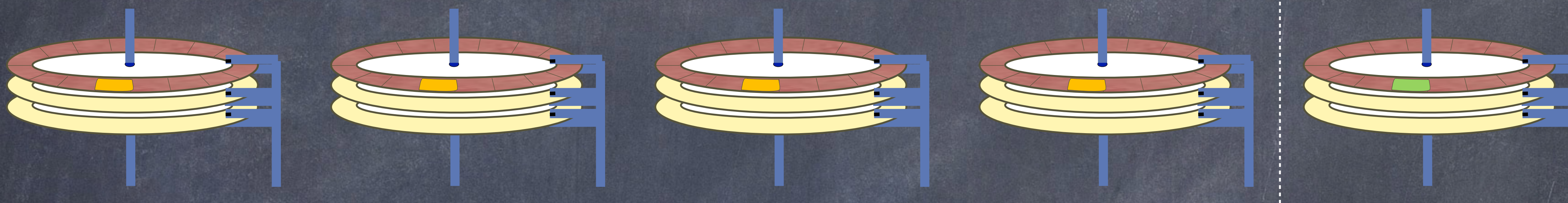
1	0	0
0	1	0
1	0	1

1	1	0
1	1	1
0	0	1

0	0	0
0	0	1
1	1	0

RAID-4: Block Striped, with Parity

Data disks | Parity disk



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

1	1	0	1	0	0	1	0	0	1	1	0	0	0	0
0	1	0	1	1	0	0	1	0	1	1	1	0	0	1
0	0	1	0	1	1	1	0	1	0	0	1	1	1	0

Disk controller can identify faulty disk

- single parity disk can detect and correct errors

RAID-4: Evaluation

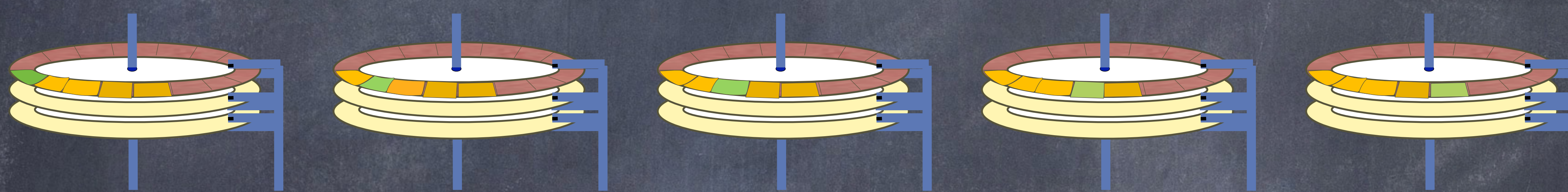
- Capacity
  - N disks of B blocks yield (N-1) x B blocks
- Reliability
  - Tolerates the failure of any one disk
- Performance
  - Fine for sequential read/write accesses and random reads
  - Random writes are a problem!

RAID-4: Performance

- Sequential Reads: (N-1) x S MB/s
- Sequential Writes: (N-1) x S MB/s
  - compute & write parity block once for the full stripe
- Random Read: (N-1) x R MB/s
- Random Writes: R/2 MB/s (N is gone! Yikes!)
  - need to read block from disk and parity block
  - Compute $P_{new} = (B_{old} \text{ XOR } B_{new}) \text{ XOR } P_{old}$
  - Write back B_{new} and P_{new}
  - Every write must go through parity disk, eliminating any chance of parallelism
  - Every logical I/O requires two physical I/Os at parity disk: can at most achieve 1/2 of its random transfer rate (i.e. R/2)
- Latency: Reads: T ms; Writes: 2T ms

RAID-5: Rotating Parity (avoids the bottleneck)

Parity and Data distributed across all disks



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

RAID-5: Evaluation

- Capacity & Reliability
  - As in Raid-4
- Performance
  - Sequential read/write accesses as in RAID-4
    - (N-1) x S MB/s
  - Random Reads are slightly better
    - N x R MB/s (instead of (N-1) x R MB/s)
  - Random Writes much better than RAID-4: R/2 x N/2
    - as in RAID-4 writes involve two operations at every disk: each disk can achieve at most R/2
    - but, without a bottleneck parity disk, we can issue up to N/2 writes in parallel (each involving 2 disks)