Reliable Storage with RAID

Principles and Operating Systems Considerations

Max Kopinsky

School of Computer Science
McGill University

25 November, 2024

Assumed Prerequisites

- Concepts of individual drives
  - Logical Block Addressing (LBA)
  - HDD and SSD
  - single-request vs steady-state drive performance
  - sequential vs random drive workloads & bandwidths
- Basics of IO scheduling in operating systems
- Mentioned briefly: OS boot sequence

Mass Storage with Drives

Data servers have to store huge amounts of data – exabytes!

Dead simple approach:

- Throw a bunch of disks together
- *Concatenate* their blocks into a huge logical drive

Mass Storage with Drives

Data servers have to store huge amounts of data – exabytes!

Dead simple approach:

- Throw a bunch of disks together
- *Concatenate* their blocks into a huge logical drive

But this is riddled with issues:

- Performance: idling disks
- Reliability: drive failures are irrecoverable
- Need for reliable drives $\Longrightarrow$ expensive!

Can We Do Better?

Mass storage has a few ideals:

Can We Do Better?

Mass storage has a few ideals:

- Keep disks as busy as possible

Can We Do Better?

Mass storage has a few ideals:

- Keep disks as busy as possible
- Ability to survive ("tolerate") disk failures

Can We Do Better?

Mass storage has a few ideals:

- Keep disks as busy as possible
- Ability to survive ("tolerate") disk failures
- Avoid wasted space

Can We Do Better?

Mass storage has a few ideals:

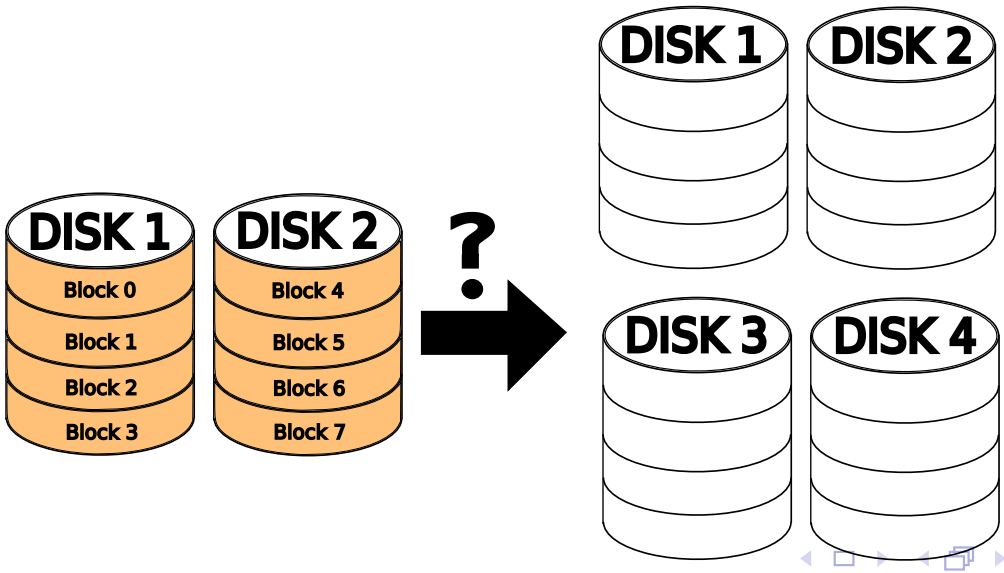
- Keep disks as busy as possible
- Ability to survive ("tolerate") disk failures
- Avoid wasted space
- Should be **cheap**!

Can We Do Better?

Mass storage has a few ideals:

- Keep disks as busy as possible
- Ability to survive ("tolerate") disk failures
- Avoid wasted space
- Should be **cheap**!

Discussion: how can we achieve this?

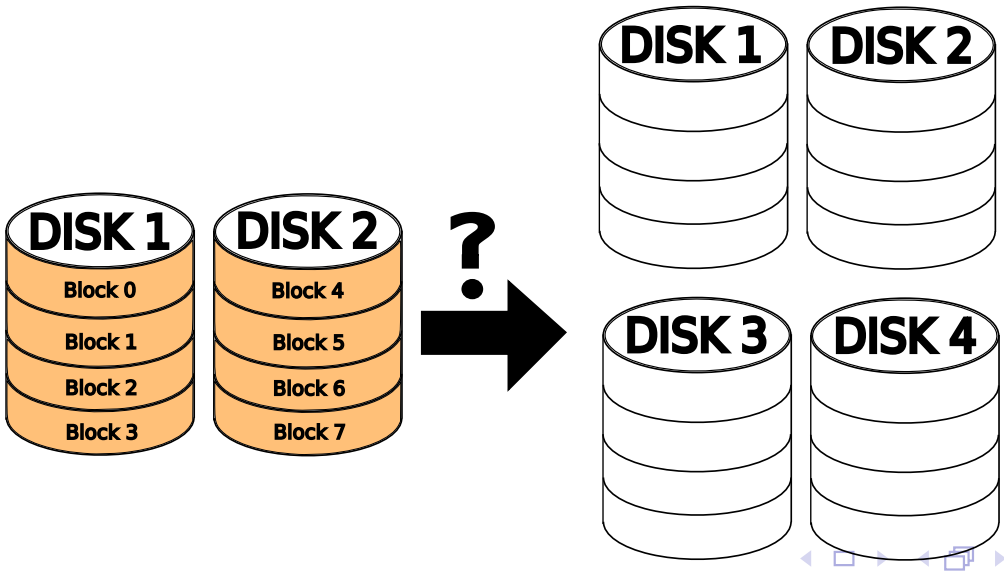


Can We Do Better?

Mass storage has a few ideals:

- Keep disks as busy as possible – spread out data
- Ability to survive ("tolerate") disk failures – redundancy
- Avoid wasted space – clever tricks!
- Should be **cheap**! – work with cheap HDDs

Discussion: how can we achieve this?



RAID

Redundant Array of Inexepensive Disks

RAID

Redundant Array of ~~inexpensive~~ Independent Disks

RAID

Redundant Array of ~~inexpensive~~ Independent Disks

- **Family** of methods to improve reliability and/or performance
- Variations called "RAID Levels"

RAID

Redundant Array of ~~inexpensive~~ Independent Disks

- **Family** of methods to improve reliability and/or performance
- Variations called "RAID Levels"
- All combine many drives into a "RAID array"
- Whole array presents as single logical drive

RAID

Redundant Array of ~~inexpensive~~ Independent Disks

- **Family** of methods to improve reliability and/or performance
- Variations called "RAID Levels"
- All combine many drives into a "RAID array"
- Whole array presents as single logical drive

- We will assume disks are HDDs
- RAID arrays can be controlled by hardware or by software

RAID Levels

Raid levels represent different compromises between

- Capacity
- Reliability
- Performance

RAID Levels

Raid levels represent different compromises between

- Capacity
- Reliability
- Performance

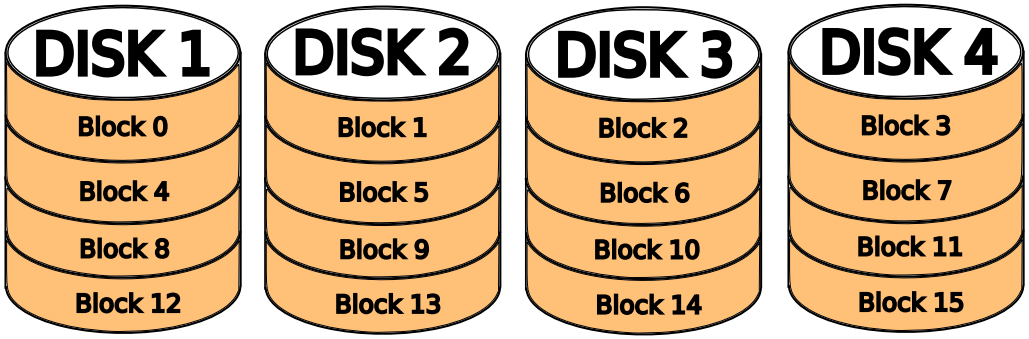
Common (standard[1]) levels:

- RAID 0: "striping"
- RAID 1: "mirroring"
- RAID 4: "parity"
- RAID 5: "distributed parity"
- RAID 6: "dual parity"

[1] By SNIA, Storage Networking Industry Association

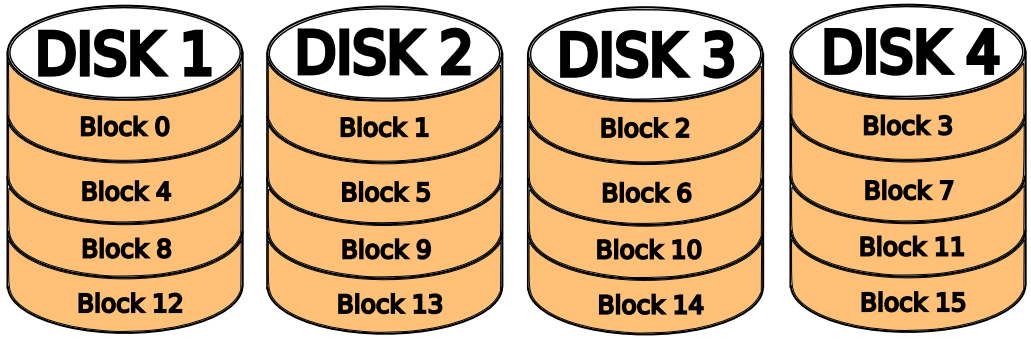
Level 0: Striping

Idea: increase disk utilization by spreading blocks around

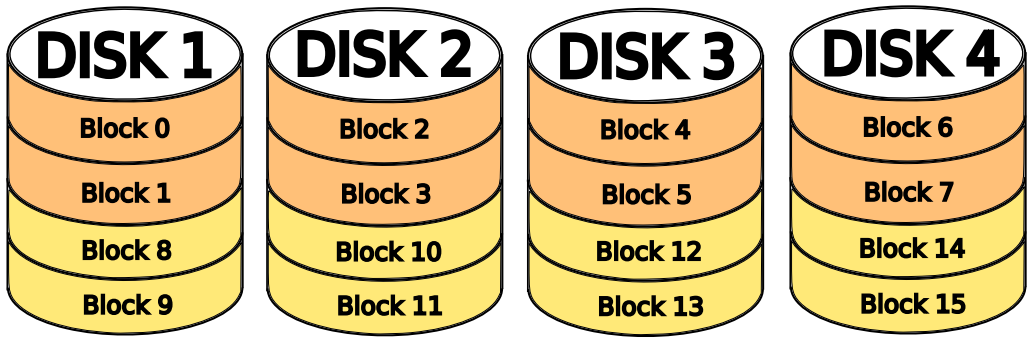


Level 0: Striping

Idea: increase disk utilization by spreading blocks around



Variations possible, like grouping blocks:



Level 0: Striping Analysis

- Capacity: no wasted space!

Level 0: Striping Analysis

- Capacity: no wasted space!
- Recall how we analyze drive performance
  - "single request" or "steady state" performance
  - Sequential workload (S B/s) vs Random workload (R B/s)
  - new: "N" is the number of disks

Level 0: Striping Analysis

- Capacity: no wasted space!
- Recall how we analyze drive performance
  - "single request" or "steady state" performance
  - Sequential workload (S B/s) vs Random workload (R B/s)
  - new: "N" is the number of disks
- Throughput:
  - Sequential throughput: $N * S$
  - Random throughput: $N * R$

Level 0: Striping Analysis

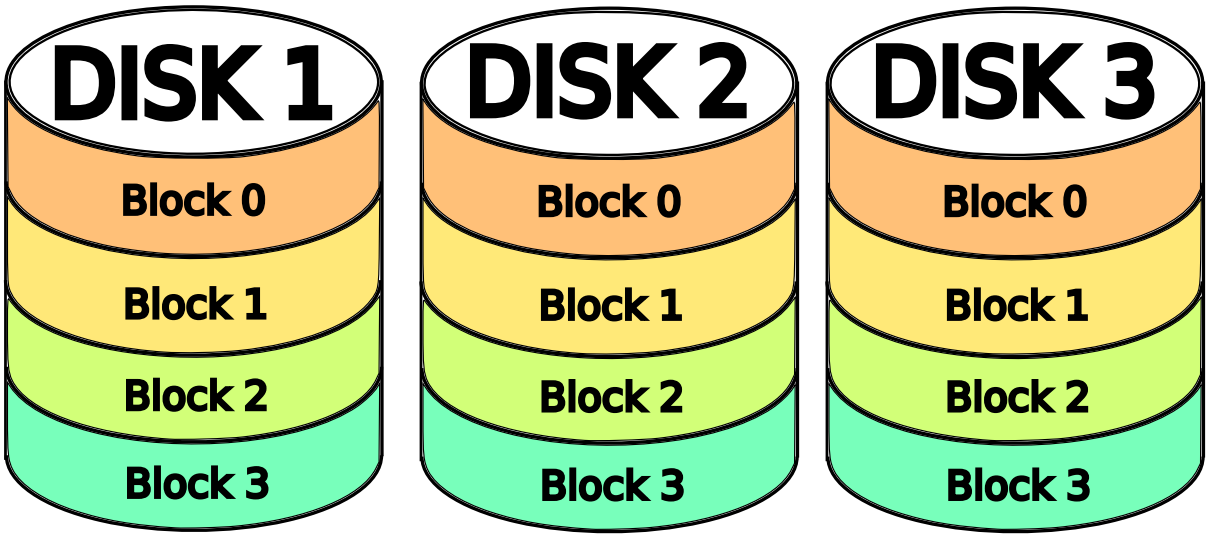
- Capacity: no wasted space!
- Recall how we analyze drive performance
  - "single request" or "steady state" performance
  - Sequential workload (S B/s) vs Random workload (R B/s)
  - new: "N" is the number of disks
- Throughput:
  - Sequential throughput: $N * S$
  - Random throughput: $N * R$
- Fault Tolerance:
  - Important characteristic: resilience to drive failure

Level 0: Striping Analysis

- Capacity: no wasted space!
- Recall how we analyze drive performance
  - "single request" or "steady state" performance
  - Sequential workload (S B/s) vs Random workload (R B/s)
  - new: "N" is the number of disks
- Throughput:
  - Sequential throughput: $N * S$
  - Random throughput: $N * R$
- Fault Tolerance:
  - Important characteristic: resilience to drive failure
  - **None at all!** Any drive failure causes loss of data.

Level 1: Mirroring

Opposite approach: maximize redundancy



Level 1: Mirroring Analysis

- Fault Tolerance:
  - Tolerates **all but one** drive failing

Level 1: Mirroring Analysis

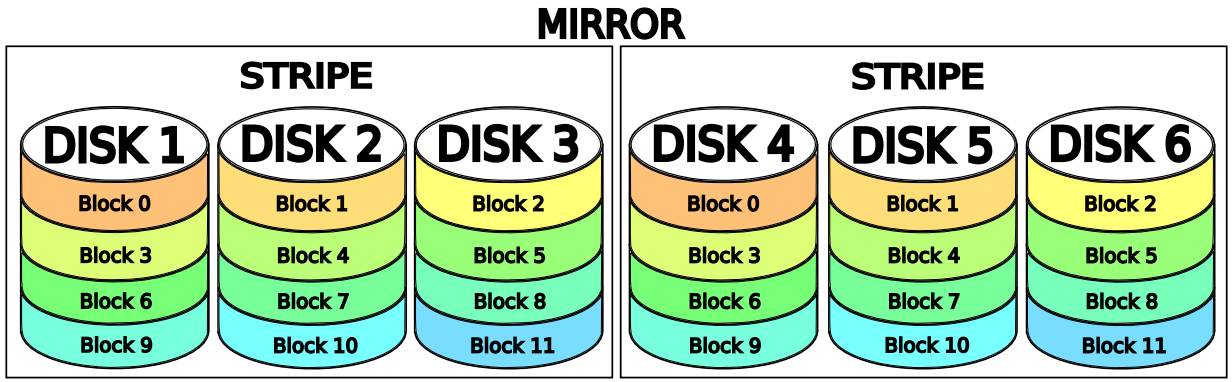
- Fault Tolerance:
  - Tolerates **all but one** drive failing
- Capacity: only one disk

Level 1: Mirroring Analysis

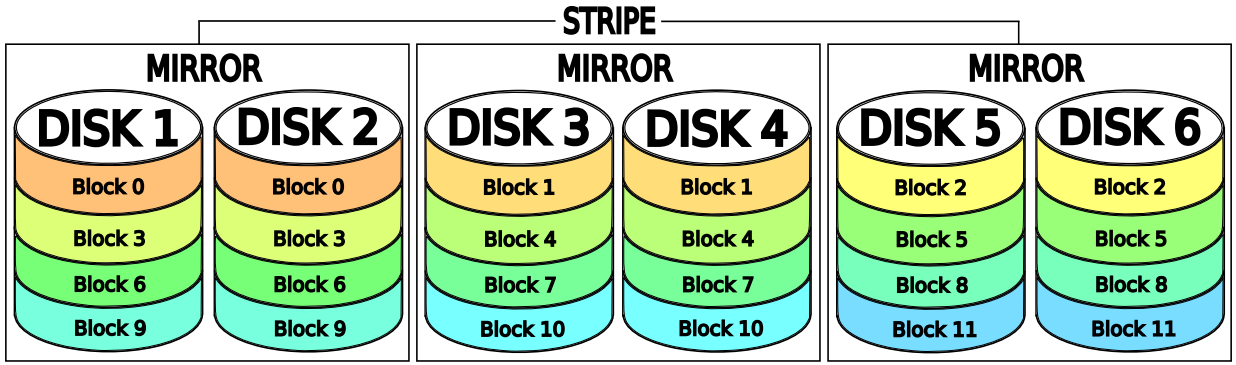
- Fault Tolerance:
  - Tolerates **all but one** drive failing
- Capacity: only one disk
- Throughput:
  - Sequential Reads: $N * S$ (well-scheduled)
  - Sequential Writes: S
  - Random Reads: $N * R$
  - Random Writes: R
- **Very bad** write performance. Doesn't scale!

Nesting Levels 0 and 1

- 0+1: "Mirror of Stripes"

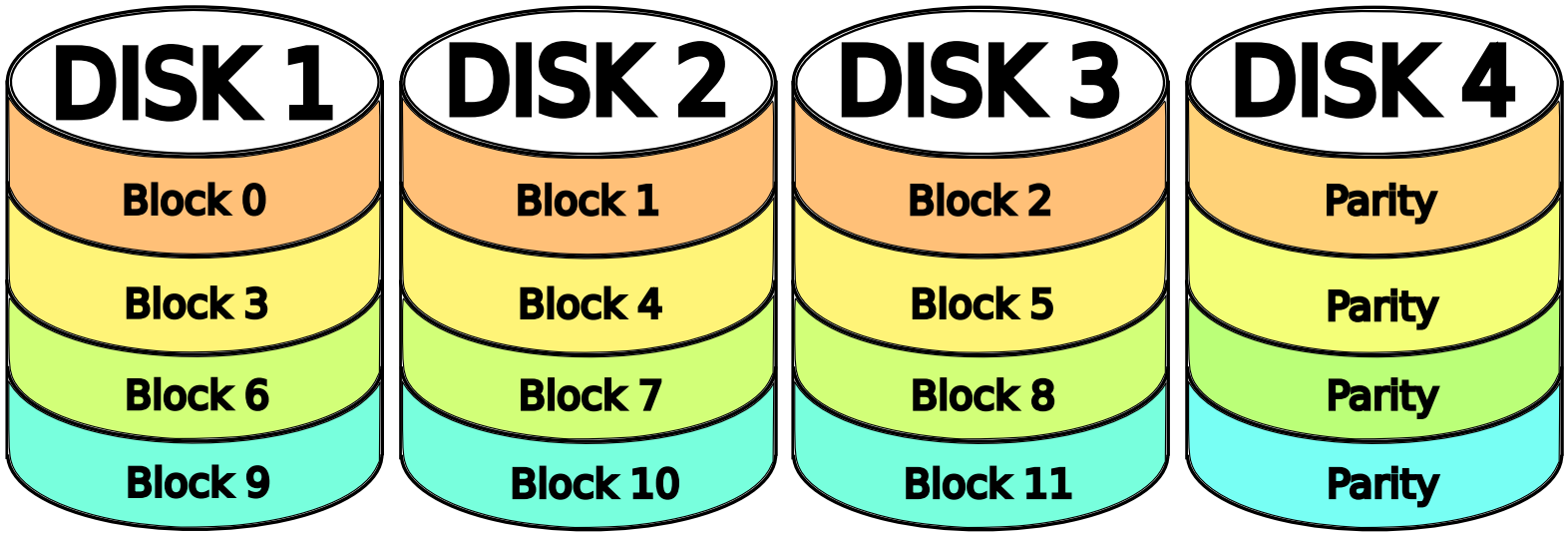


- 1+0: "Stripe of Mirrors"



Raid 4: Parity

Stepping stone to good middle grounds.



$$\text{Parity } 0 = \text{Block } 0 \oplus \text{Block } 1 \oplus \text{Block } 2$$

Raid 4: Parity Analysis

- Fault Tolerance
  - Any single disk
  - Including the parity disk!

Raid 4: Parity Analysis

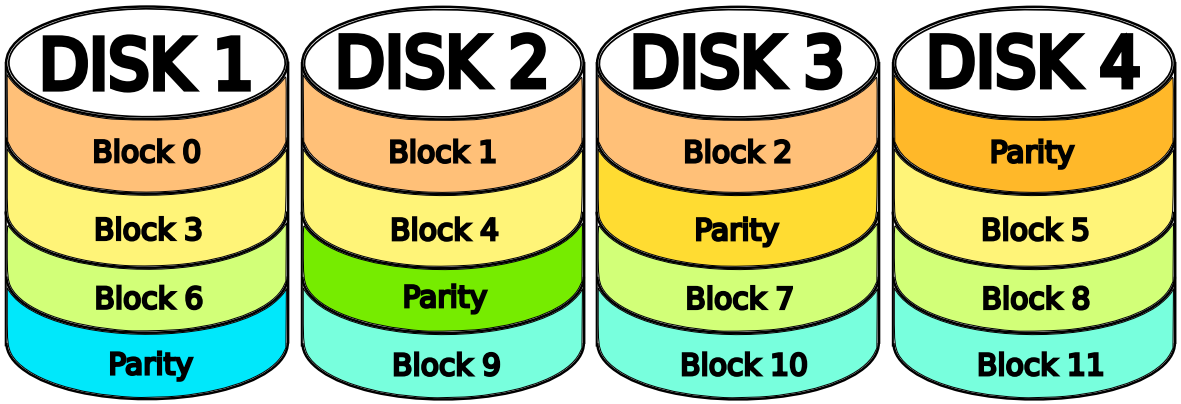
- Fault Tolerance
  - Any single disk
  - Including the parity disk!
- Capacity: All but one disk

Raid 4: Parity Analysis

- Fault Tolerance
  - Any single disk
  - Including the parity disk!
- Capacity: All but one disk
- Throughput:
  - Sequential: (N - 1) $*$ S
  - Random **Read**: (N - 1) $*$ R
  - Random **Write**: R / 2
    - "small-write" problem

Briefly, RAIDS 5: Distributed Parity

Solves the "small-write" problem.



Summary

	Striping	Mirroring	MoS	SoM	Parity	Dist. Parity
Capacity	$N * B$	B	$N * B/M$	$N * B/M$	$(N-1) * B$	$(N-1) * B$
S Reads	$N * S$	$N * S$	$N * S$	$N * S$	$(N-1) * S$	$(N-1) * S$
S Writes	$N * S$	S	$N * S/M$	$N * S/M$	$(N-1) * S$	$(N-1) * S$
R Reads	$N * R$	$N * R$	$N * R$	$N * R$	$(N-1) * R$	$N * R$
R Writes	$N * R$	R	$N * R/M$	$N * R/M$	$R/2$	$N * R/4$
Tolerance	0	$N-1$	1	M	1	1

Variables:

- N: number of disks
- B: blocks per disk
- M: mirroring factor
- S: steady-state sequential throughput
- R: steady-state random throughput

RAID Faults and How to Tolerate

There are two major kinds of faults:

- Drive failures
  - Lost drives must be "rebuilt" which takes time.
  - Think: chance of second failure during rebuild?

RAID Faults and How to Tolerate

There are two major kinds of faults:

- Drive failures
  - Lost drives must be "rebuilt" which takes time.
  - Think: chance of second failure during rebuild?
- Consistency faults, or "Consistent Update Problem"
  - Catastrophy (e.g. power outage) during update
  - Result: volume in inconsistent state
  - Solution: transactions
  - Usually implemented with journaling

Journaling: Hardware vs Software

Hardware:

- Journal stored in nonvolatile onboard cache
- Unfinished events "replayed" during startup

Software:

- Journal stored somewhere special on-disk
- Must be careful: if in RAID volume, could lose it!
- But if not duplicated, susceptible to failure

Designing an OS for Hardware RAID

- Hardware controller presents as one huge drive!
- Typical scheduling for LBA drives works well
- On-board scheduling will improve our work for specifics
- Still critical to do our own scheduling!
- At the FS/OS level, we have more ability to coalesce
- Research supports "deadline" algorithms in general[2]
  - (but other algorithms can be better for known access patterns)

[2] https://trustworthy.systems/publications/theses_public/08/Carroll%3Abe.pdf

Designing for Software RAID

- Not generally done - rather, "RAID utilities" for OS
  - For Linux, `mdadm`
- But if you do, important considerations:
  - How to handle consistent update problem?
  - Scheduling IO across disks – many algorithms
    - Layered approach good for SoC:
    - Schedule as if for one disk first
    - Then separate layer schedules for RAID array

Booting with Software RAID

Non-obvious problem with software RAID:

- Complicates boot sequence!
- Need to load OS from the RAID array
- But can't access the RAID array without the OS!
- Typically solved by boot-only filesystem image
  - Linux "initramfs"
- If volume fails, easy to recreate

Recap

- RAID Principles:
  - improves performance and/or reliability
  - Pick 1+0 (stripe of mirrors) for performance
  - Pick 6 (dual parity) for reliability + capacity
- RAID Array Faults:
  - Disk failures
  - Consistent Update Problem
- OS Considerations
  - Hardware controller preferable if possible
  - Software RAID has many decision points:
    - Controlled by OS or utility?
    - Handling consistent update problem?
    - Boot sequence?
- Next time: end-of-term Q&A