Storage Reliability

Juncheng Yang

March 11

Harvard John A. Paulson
School of Engineering
and Applied Sciences

HARVARD UNIVERSITY
MADSYS
Measurement **a**nd **D**esign of **Sys**tems Lab

Update

- Project proposal due next Wednesday
  - discussion (optional) office hour, email or schedule an appointment

Agenda

- RAID
- Reliability calculation
- Erasure coding
- Data placement

Key Questions to Think About After Class

- What are different RAID levels? How do they work?
- How to calculate the MTTDL of different RAID levels?
- What is the guarantee and drawbacks of erasure coding?

RAID

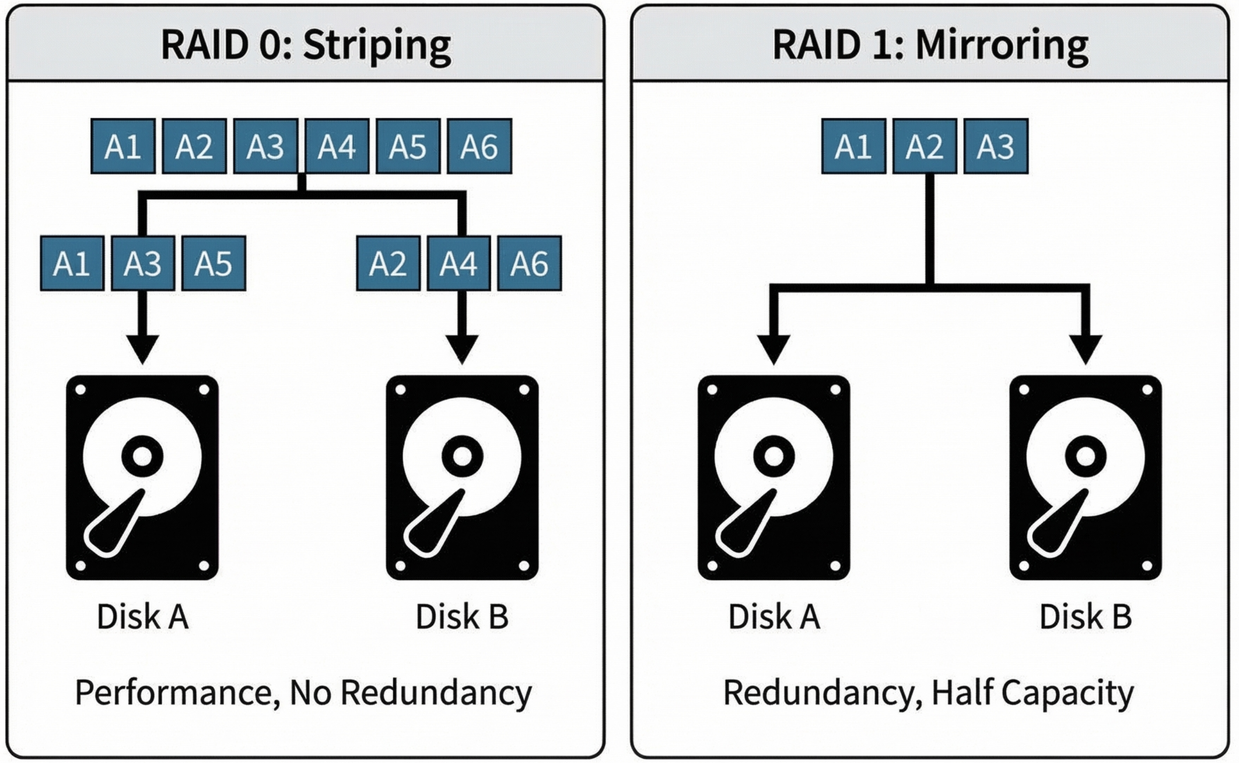
Redundant array of independent disks

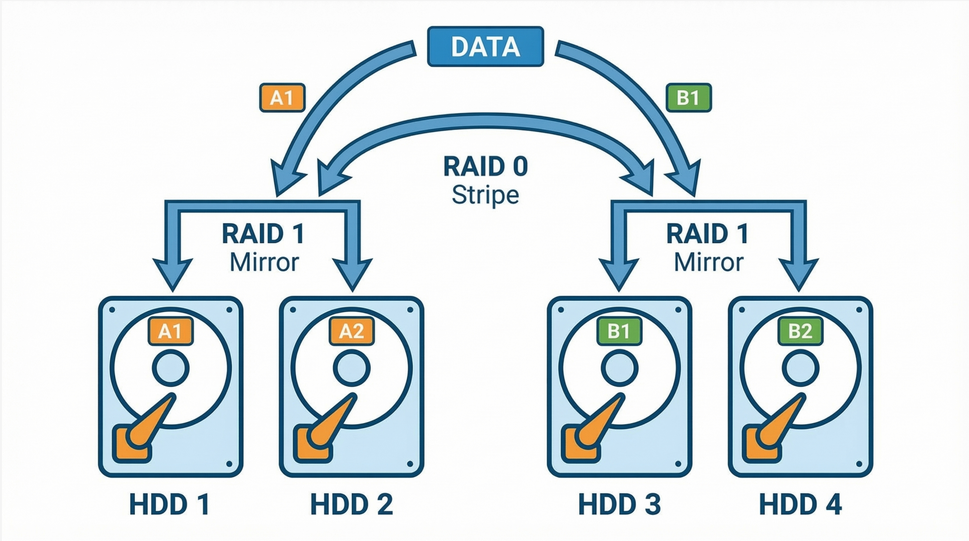
RAID

- What
  - combining multiple *local-attached* disks into one (array)
- Why
  - capacity
  - reliability
  - performance
- Different levels
  - RAID 0: striping
  - RAID 1: mirroring
  - RAID10: striping + mirroring
  - RAID 4
  - RAID 5
  - RAID 6

RAID

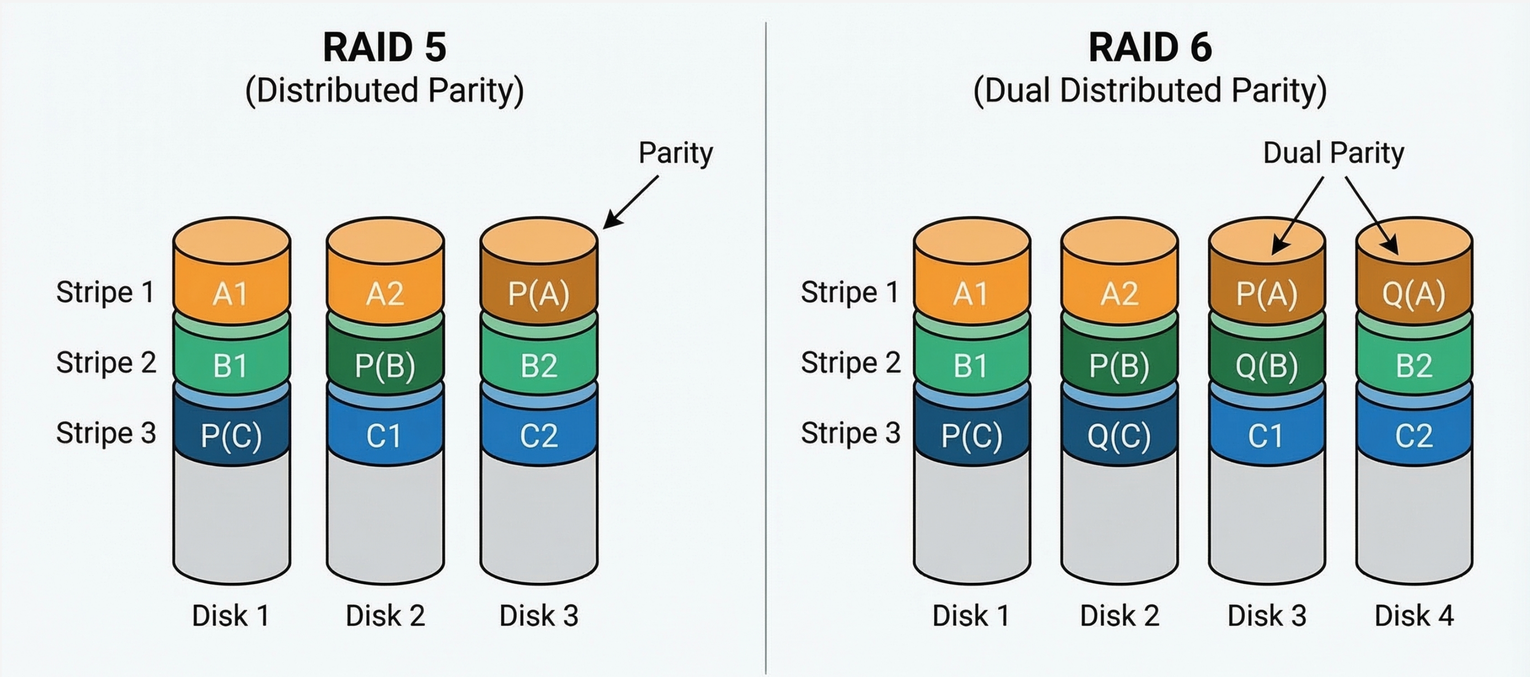
- Striping
  - break up LBN space into fixed-size stripe units
  - distribute stripe units among disks in round-robin fashion
  - key design: stripe size
    - too large: poor load balancing, limited parallelism
    - too small: too many seeks
- RAID 0: striping
- RAID 1: mirroring
- RAID10: striping + mirroring





RAID

- RAID 4-6: use Error Correcting Codes (ECC) to compute a Parity Block from data blocks
- Example
  - XoR: $P(A) = A_1 \oplus A_2$
  - recovery: $A_1 = P(A) \oplus A_2$
- RAID 4: dedicated disk for parity
- RAID 5: rotating disks for parity
- RAID 6: two parities
  - $P(A) = A_1 \oplus A_2$
  - $Q(A) = A_1 \oplus 2A_2$
- Low capacity overhead
- But slow small writes
  - read and double writes



Modes of operation

- Normal mode
  - everything working
  - maximum efficiency
- Degraded mode
  - some disk unavailable
  - must use degraded mode operations
  - read: reconstruct using parity
- Rebuild mode
  - reconstructing lost disk's contents onto spare
  - degraded mode operations plus competition with rebuild

RAID: overhead and performance

RAID Level	Capacity Overhead	Fault Tolerance	Write	Read	Degraded Write	Degraded Read
RAID 0 (Striping)	**0%**	**None**	**Excellent** (parallel writes)	**Excellent** (parallel reads)	**NA**	**NA**
RAID 1 (Mirroring)	**50%**	**Single** (can lose 1 disk)	**Moderate** (write to both)	**Good** (can read from either)	**Good**	**Good**
RAID 5 (Parity)	**1/n** (1 disk)	**Single** (can lose 1 disk)	**Low** (parity calc., read-modify-write)	**Excellent** (striped reads)	**Low**	**Low** (read amp)
RAID 6 (Dual Parity)	**2/n** (2 disks)	**Dual** (can lose 2 disks)	**Low**	**Excellent** (striped reads)	**Low**	**Low**
RAID 10 (1+0)	**50%**	**High** (1 per mirrored pair)	**Good**	**Excellent** (striped and mirrored)	**Good**	**Good**

MTTDL Calculation

Mean-time-to-data-loss

Terms

- MTBF: mean time between failure
  - $\text{MTBF} = \frac{\sum (t_{down} - t_{up})}{\#\text{failure}}$
  - similar term MTTF
    - MTBF assumes a repairable system
    - MTTF assumes a non-repairable system
- Common model
  - disk has a constant failure rate: $\lambda = \frac{1}{\text{MTTF}_{disk}} \approx \frac{1}{\text{MTBF}_{disk}}$
  - repair rate: $\mu = \frac{1}{\text{MTTR}}$
- Example
  - MTBF=1,000,000 hours, $\lambda = 10^{-6}h^{-1}$, MTTR=10 hours, $\mu = 0.1h^{-1}$

MTBF calculation without repair

- RAID 0 (striping)
  - $\text{MTBF}_{\text{array}} = \text{MTBF}_{\text{drive}}/N$
- RAID 1 (mirror)
  - $\text{MTBF}_{\text{array}} = \text{MTBF}_{\text{drive}}/2 + \text{MTBF}_{\text{drive}} = 1.5 \times \text{MTBF}_{\text{drive}}$
  - when both disks are healthy, failure rate 2λ
- RAID 5 (one parity)
  - $\text{MTBF}_{\text{array}} = \text{MTBF}_{\text{drive}}/5 + \text{MTBF}_{\text{drive}}/4 = 0.45 \times \text{MTBF}_{\text{drive}}$

MTTDL calculation with repair

- Repair: rebuild data on the failed disk on a new disk
- MTTR: mean time to rebuild
- MTTDL: mean time to data loss
- Calculation (intuitive approach):

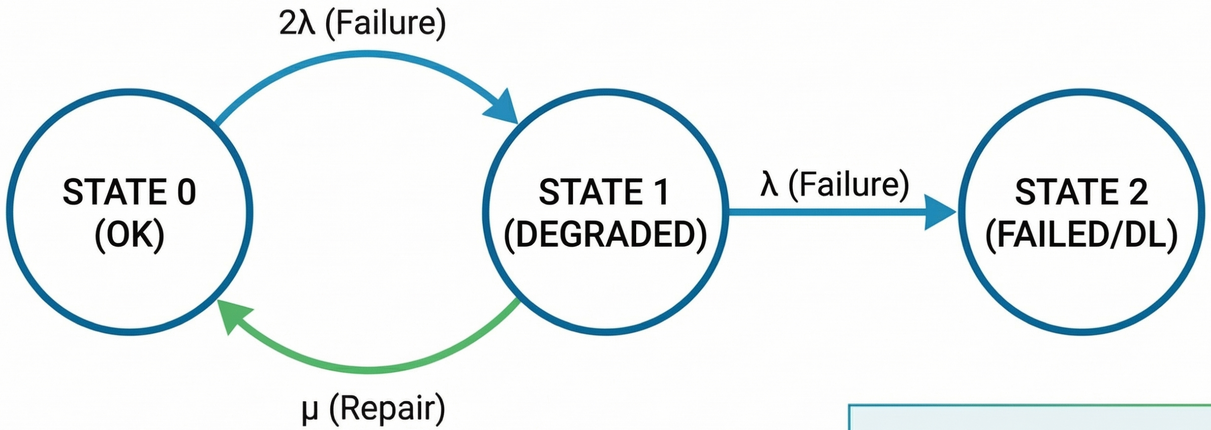
canonical way: solve Markov Chain

  - $\text{MTTDL}_{\text{array}} = \text{MTBF}_{\text{first drive}} / Pr(\text{2nd failure before repair}) = \dfrac{\text{MTBF}_{\text{first drive}}}{\text{MTTR}/\text{MTBF}_{\text{second drive}}}$
  - second term: average numbers of times until a second failure happens before repair finishes
  - RAID 1 (mirror): $\text{MTTDL}_{\text{array}} = \dfrac{\text{MTBF}_{\text{drive}}/2}{\text{MTTR}/\text{MTBF}_{\text{drive}}} = \dfrac{\text{MTBF}_{\text{drive}}^2}{2 \times \text{MTTR}}$
  - RAID 5: $\text{MTTDL}_{\text{array}} = \dfrac{\text{MTBF}_{\text{drive}}/5}{\text{MTTR}/\text{MTBF}_{\text{drive}}/4} = \dfrac{\text{MTBF}_{\text{drive}}^2}{20 \times \text{MTTR}}$

MTTDL calculation with repair

- Continuous-time Markov chain
  - time spend in a state before jump is exponentially distributed with rate equal to the total exit rate out of that state
- From State 0
  - only one possible jump (to State 1), with exit rate 2λ
  - mean time spent in State 0: $\frac{1}{2\lambda}$
  - so $T_0 = \frac{1}{2\lambda} + T_1$
- From State 1
  - exit rate is $\mu + \lambda$, so mean time is $\frac{1}{\mu + \lambda}$
  - then:
    - with probability $\frac{\mu}{\mu + \lambda}$ go to State 0
    - with probability $\frac{\lambda}{\mu + \lambda}$ go to State 2 (data loss)
  - So $T_1 = \frac{1}{\mu + \lambda} + \frac{\mu}{\mu + \lambda} T_0 + \frac{\lambda}{\mu + \lambda} T_2$

- T_0 = expected time to data loss starting in State 0
- T_1 = expected time to data loss starting in State 1
- T_2=0

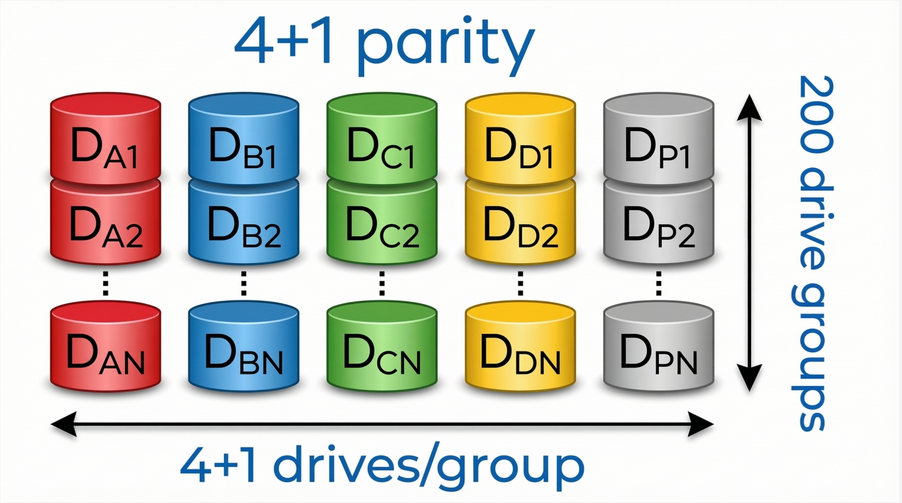


$$\text{MTTDL}_{RAID1} = \frac{\mu + 3\lambda}{2\lambda^2}$$

λ: Single drive failure rate
μ: Drive repair rate
DL: Data Loss

Array reliability with/without repair

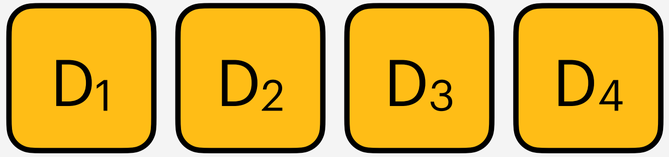
- Assume 1000 disks using RAID5 (5-disks) + striping
  - without repair
    - $\text{MTBF}_{\text{sripe}} = \text{MTBF}_{\text{drive}}/5 + \text{MTBF}_{\text{drive}}/4 = 0.45 \times \text{MTBF}_{\text{drive}}$
    - $\text{MTBF}_{\text{array}} = \text{MTBF}_{\text{sripe}}/200$
    - **93 days!**
  - with repair
    - $\text{MTTDL}_{\text{stripe}} = \dfrac{\text{MTBF}_{\text{drive}}/5}{\text{MTTR}/\text{MTBF}_{\text{drive}}/4} = \dfrac{\text{MTBF}^2_{\text{drive}}}{20 \times \text{MTTR}}$
    - $\text{MTTDL}_{\text{array}} = \text{MTTDL}_{\text{sripe}}/200$
    - **594 years**



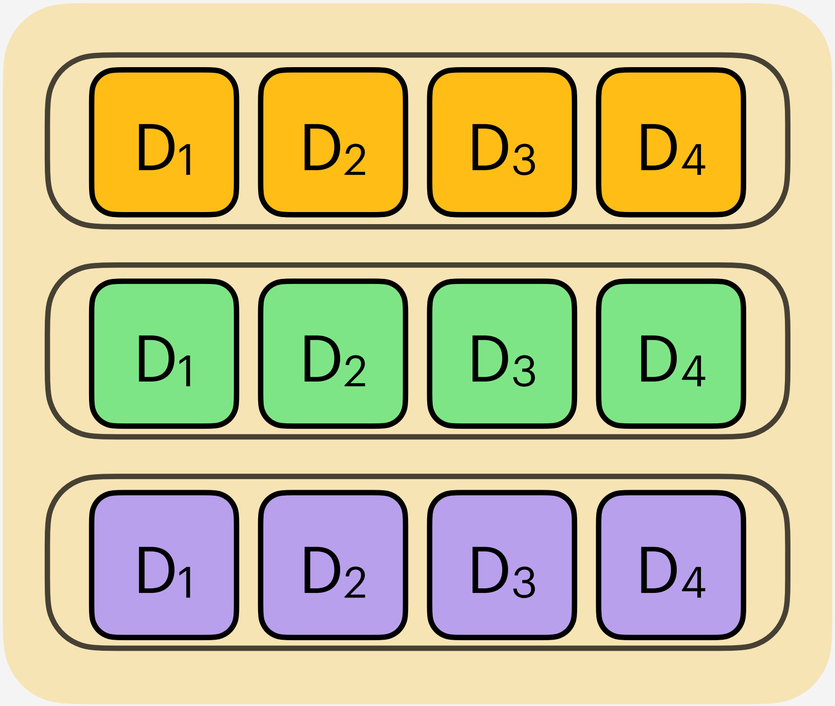
Common value
- $\text{MTBF}_{\text{drive}}$ = 1,000,000 hours
- MTTR = 2 days

Erasure Coding

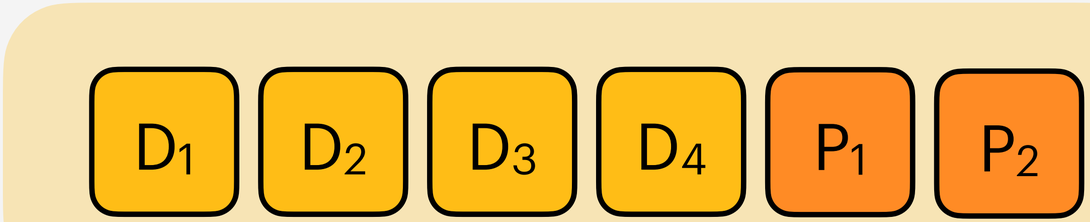
From replication to simple parity to erasure coding



tolerating two failures



replication: capacity overhead 200%



RAID 6: capacity overhead 50%

Can we have arbitrary number of data and parity chunks?

Erasure coding

each chunk stored on one server

any *K* chunks can recover the original data

Terminology:

- ***k***: # data shards
- ***p***: # parity shards
- ***n***=*k*+*p*: # total shards per stripe
- **stripe**: the unit EC operates on
- **systematic codes**: k data shards stored as-is
- **MDS codes**: maximum distance separable, any *k* of *n* shards can reconstruct data (minimum capacity overhead)
- example notation: EC(6, 3) uses 3 data chunks and 3 parity chunks

simplest idea: single-parity XOR

Erasure coding

- Treat each shard as a vector of bytes, parity are computed as linear combinations of data shards
  - decoding: solve the linear system
- Flaw with standard arithmetic
  - overflow, e.g., the sum of two uint8 200 and 100 cannot fit in uint8
- Reed-Solomon codes operate over a finite field (Galois Fields) $GF(2^w)$
  - usually w=8: 256 elements
  - the result of an arithmetic operation on two elements is always another element in the field

Reed-Solomon code

- RS codes are typically MDS
  - any k can reconstruct the original data
- Most storage systems use RS via a **generator matrix** (G) of size $(n \times k)$
  - $$\begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_{n-1} \end{bmatrix} = G \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{k-1} \end{bmatrix}$$
  - for systematic RS codes
    - $G = \begin{bmatrix} I_k \\ A \end{bmatrix}$
      - top part (I_k): the data shards are unchanged
      - bottom part (A): parity equations

Reed-Solomon Codes

- Encoding
  - split input bytes into k equal-sized chunks (pad if needed)
  - compute p parity chunks
  - store the $k+p$ shards on distinct nodes/racks/AZs
- Decoding
  - pick any k available chunks
  - build a $k \times k$ decoding matrix from the corresponding rows of (G)
  - invert that matrix over the field
  - multiply inverse by the available shard symbols to recover original data chunks
- Most widely used codes
  - example RS(6,3), RS(12,4)

Hitchhiker

- Problem with RS codes
  - repair bandwidth: read *k* chunks to reconstruct one
- Hitchhiker: piggybacking parity on data
  - two independent systems
    - system A: data units $a_0, a_1, \ldots, a_{k-1}$, $\text{parity}_A = \sum a_i$
    - system B: data units $b_0, b_1, \ldots, b_{k-1}$, $\text{parity}_B = \sum b_i + \sum f_i(a_i)$
  - if node 0 containing (a_0, b_0) fails
    - download $(a_i, b_i + f_i(a_i))$ from surviving nodes
    - recover a_0 as in standard RS
    - recover b_0 after a_0 is recovered

Hitchhiker example

- EC(10, 4)
  - data node *i* stores (a_i, b_i)
  - parity node 11: $(f_1(a), f_1(b))$
  - parity node 12: $(f_2(a), f_2(b) \oplus a_1 \oplus a_2 \oplus a_3)$
  - parity node 13: $(f_3(a), f_3(b) \oplus a_4 \oplus a_5 \oplus a_6)$
  - parity node 14: $(f_4(a), f_4(b) \oplus a_7 \oplus a_8 \oplus a_9 \oplus a_{10})$
- Recover data node 1
  - download 13 bytes
    - node 2 and 3: a_2, a_3, b_2, b_3
    - node 4-10: $b_4 \ldots b_{10}$
    - node 11: $f_1(b)$
    - node 12: $f_2(b) \oplus a_1 \oplus a_2 \oplus a_3)$
  - decode
    - decode b_1 using $b_2..b_{10}$ and $f_1(b)$
    - decode a_1 using a_2, a_3 and $f_2(b) \oplus a_1 \oplus a_2 \oplus a_3)$

LRC (locally repairable codes)

- Trade storage for repair bandwidth
  - usually not MDS codes (not storage optimal)
- Motivation
  - hierarchical data center: higher bandwidth between disks in the same server/rack/AZ
- Core idea: local parity
  - repair a chunk in RS: read *k* chunks
  - repair a chunk in LRC: read neighbor chunks
- Notation (k, l, r) ***l***: local groups (1 parity per group), ***r***: global parity
  - Azure (12, 2, 2)
    - two groups of RS(6,1) and two global parity
    - tolerate three unavailabilities

Regenerating code

- Reed-Solomon codes
  - optimal for fault tolerance
  - expensive in repair: repair one chunk requires fetching *k* chunks ($k \times$ traffic amplification)
- MSR codes (Minimum Storage Regenerating)
  - minimum storage overhead (same as MDS)
  - minimum repair bandwidth
  - core parameters (n, k, d)
    - ***n***: # total node, ***k***: # node needed to reconstruct the whole file, ***d***: # helper node in repair
    - each node stores α, repair contacts *d* helpers and download β from each, so total repair bandwidth is $d\beta$
  - $\alpha = \frac{B}{k}, \beta = \frac{\alpha}{d-k+1}, d\beta = \frac{d}{d-k+1}\alpha$
  - stripe size 4MB, *n*=6, *k*=4, *d*=5, repair bandwidth 5×0.5 MB=2.5MB

Clay codes

- Historically, many MSR codes: elegant theoretically but practically painful
  - sub-packetization (split each node's data into *many* tiny chunks)
  - complex repair logic or uneven repair behavior between data and parity nodes
- Clay (Coupled LAYer) codes: base MDS code + structured "coupling" method across stacked layers
  - take a familiar base MDS code (e.g., Reed–Solomon-like) for an (n,k) stripe
  - make multiple layers of that code (each layer is a full coded stripe by itself)
  - apply a coupling (linear transformation) between sub-chunks across layers

Configuration	Sub-packetization (α)	RS Repair Bandwidth	Clay Repair Bandwidth	Bandwidth Savings
k=4, m=2, d=5	8	4S	2.5S	**37.5%**
k=8, m=4, d=11	64	8S	2.75S	**65.6%**
k=10, m=4, d=13	256	10S	3.25S	**67.5%**

Data Placement and Failure Domain

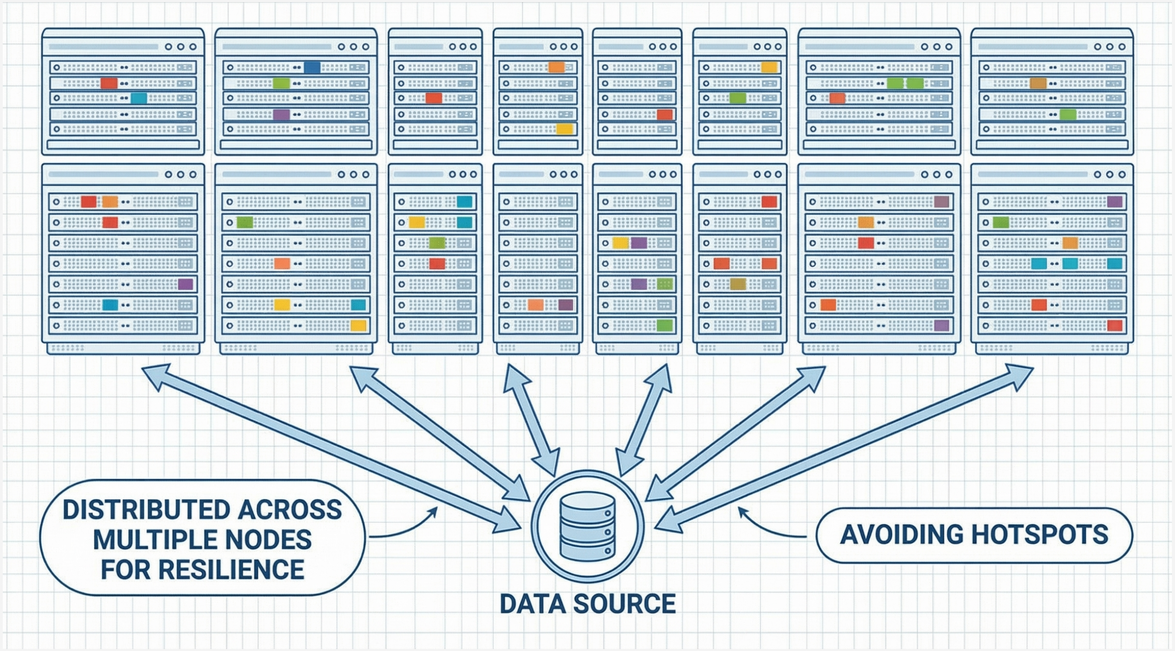
The hierarchy of failure

- **Failure Domain**
  - any set of resources that share a single point of failure, they form a hierarchy
  - placement strategy must "climb" this hierarchy depending on reliability goals

Failure Domain	Shared Component (The Risk)	Survival Goal
Disk	The mechanical platter / controller	Survive a drive failure
Server	Motherboard, RAM, OS kernel	Survive a crash or kernel panic
Rack	Top-of-Rack switch, PDU	Survive a blown fuse or switch reboot
Row / Zone	Cooling, power line, fire suppression	Survive a localized fire or AC failure
Region	Geography	Survive an earthquake or flood

Modern data placement

- Server awareness
- Rack-awareness
  - HDFS: calculate distance between nodes
    - Replica 1: local node (rack)
    - Replica 2: different rack
    - Replica 3: same rack as Replica 2 (bandwidth optimization)
  - Ceph: CRUSH algorithm
    - no central server: hash(id)→PG, CRUSH(PG)→placement
    - similar to consistent hashing, but with hierarchy and weight
- Zone-awareness
  - cloud, e.g., S3
- Declustered placement
  - each stripe is assigned a different set of disks
  - balanced distribution and faster recovery



Summary

- RAID
- Reliability calculation
- Erasure coding
- Data placement

Next class

- Data security and protection