Exploring
Error Checking and Correction Protocols

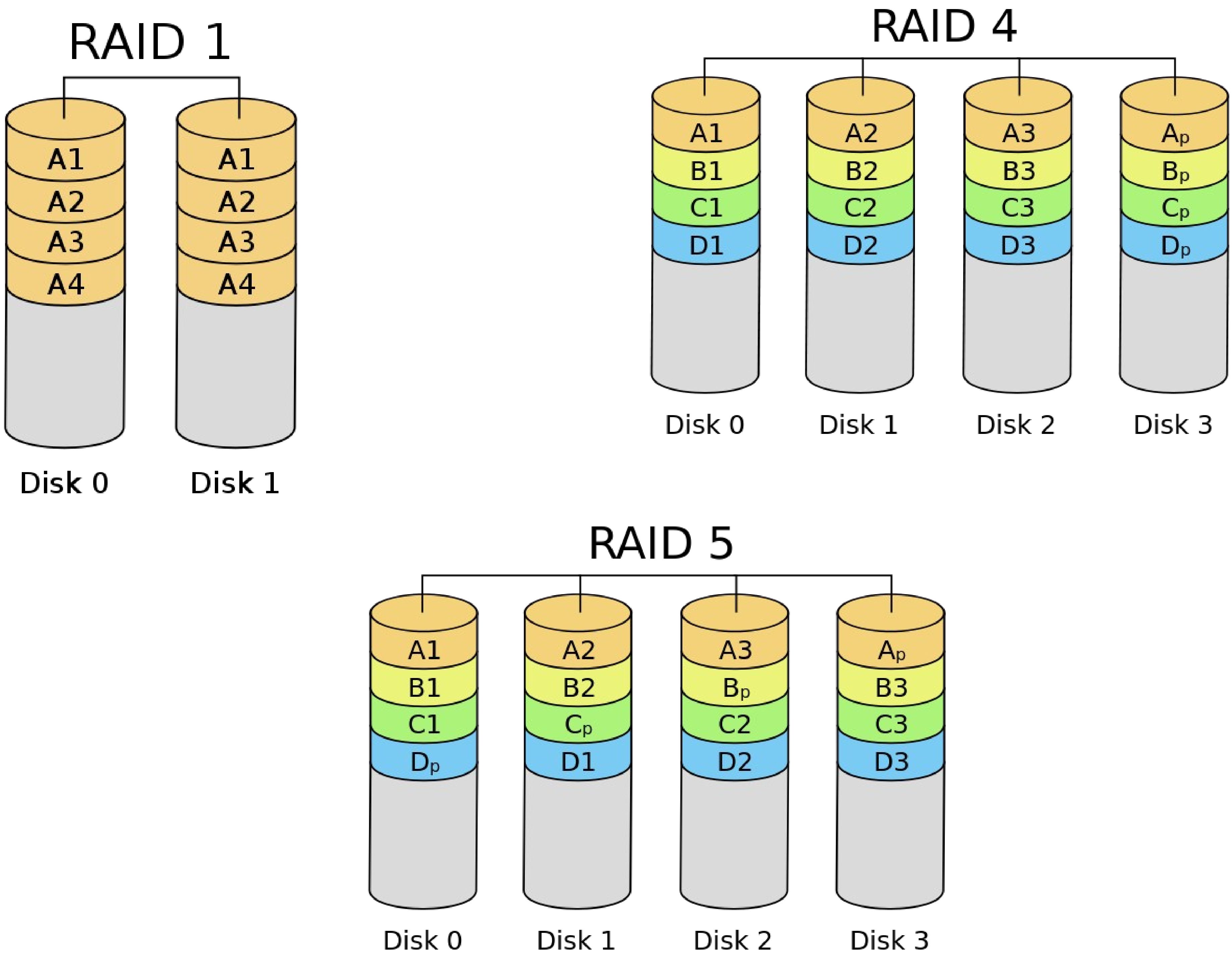
Haoran Yu

Motivation

- Data may become corrupted:
  - Over network communication
  - Over disk failures
- Error correction is necessary:
  - Be able to detect error
  - Be able to correct error efficiently

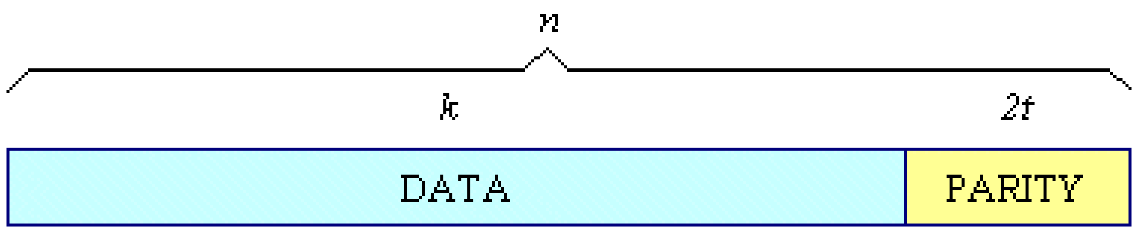
Error correction using RAID

- RAID (redundant array of independent disks) techniques store redundant data in backup disks
  - Upon failure in data disk, restore from backup disk
  - Assume disks fail independently
- Levels
  - RAID 1: mirroring
  - RAID 4: block-level striping with parity block
  - RAID 6: distributed block-level striping
  - Parity data: using the Boolean XOR function to reconstruct the missing data


RAID 1
A1
A2
A3
A4
A1
A2
A3
A4
Disk 0
Disk 1
RAID 4
A1
B1
C1
D1
A2
B2
C2
D2
A3
B3
C3
D3
Ap
Bp
Cp
Dp
Disk 0
Disk 1
Disk 2
Disk 3
RAID 5
A1
B1
C1
Dp
A2
B2
Cp
D1
A3
Bp
C2
D2
Ap
B3
C3
D3
Disk 0
Disk 1
Disk 2
Disk 3

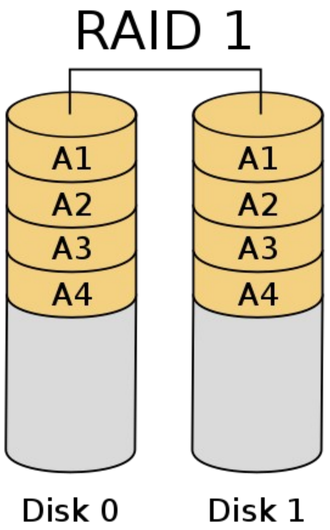
Erasure code error correction

- transforms a message of k symbols into:
  - a longer message (code word) with n symbols such that
  - the original message can be recovered from a subset of the n symbols
- Optimal erasure codes
  - Parity: used in RAID storage systems
  - Reed–Solomon codes: parameterized by symbol size, set of symbols is interpreted as the finite field

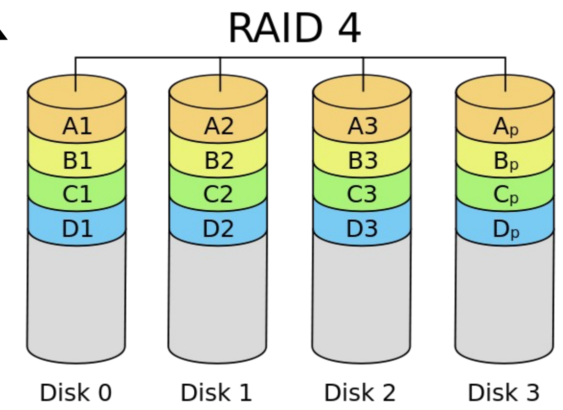


Demo: Mirroring Data

- One data disk and one backup disk
- Procedure:
  - [OS] Add additional system calls to allow specification of which disk to read/write to.
  - Write data to disk 1, write the same data to disk 2 as a backup.
  - Something goes wrong with the data on disk 1.
  - Recover data from disk 2.
- Tool: Xv6 Operating System from MIT
- Emulator: QEMU (use virtual disks)



Demo: Parity Disk

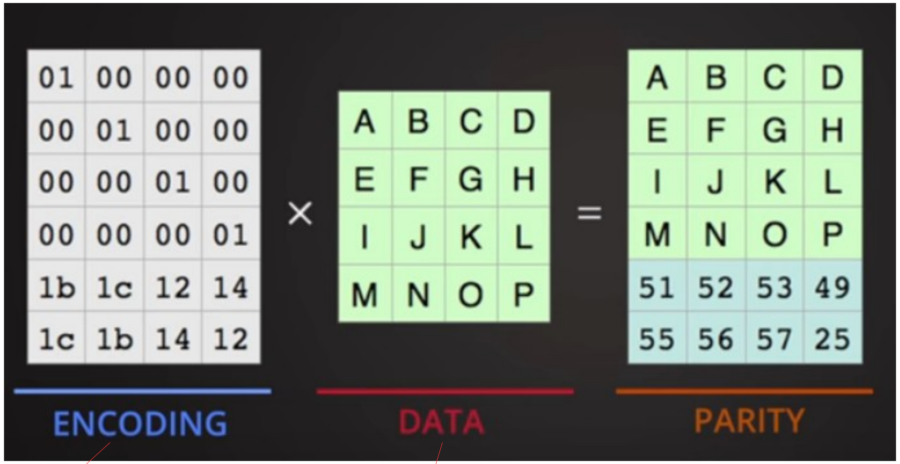


- Two data disks and one parity disk
- Procedure:
  - Break a file into blocks (size 512 bytes)
  - Place odd numbered block on data disk 1
  - Place even numbered block on data disk 2
  - [OS] Initialize parity disk, in case data disks are unknown
  - [OS] Perform XOR between the two adjacent blocks, store the result on parity disk
  - Parity check: something goes wrong with data on one data disk
  - Restore data by XOR between the other data disk and the parity disk.

$$\begin{array}{rr} & \mathbf{01101101} \\ \text{XOR} & \mathbf{11010100} \\ \hline & \mathbf{10111001} \end{array}$$

Reed Solomon

- Break data into units of size of symbol
- Multiply data by encoding generator polynomial
- Append the result
  - This is the code word
- Error checking:
  - Divide the code word by the decoding generator polynomial
  - If data is correct: result is 0
  - If data is corrupted: result is not 0
- Error correction:
  - Very complex
  - One approach: since each error is unique, find the solution in a lookup table



$$G(x) = x^2 + 6x + 3$$

$a_5\,x^4$	$a_5\,x^3$	$a_2\,x^2$	$a_3\,x^1$	$a_0\,x^0$
111	111	100	011	010
7	7	4	3	2

Demo: Circular Redundancy Check

- Adding Reed Solomon to Xv6 is too difficult
- Implemented Circular Redundancy Check(CRC) instead
  - There is some conceptual overlap between CRC and RS
  - Both perform polynomial divisions between code word and generator polynomial
  - Disadvantage of CRC: only performed check, did not perform error correction
- Procedure:
  - Store data on disk
  - At later point, request data from disk
  - [OS] Filesystem computes codeword
  - Application performs CRC, finds error
  - Request data from disk again to correct