RAID

and beyond

Anwitaman DATTA
SCE, NTU Singapore

⌘ RAID: Redundant Array of Independent Disks

⌘ MDS erasure codes: Fault-tolerant Storage

RAID

Redundant Array of Independent Disks

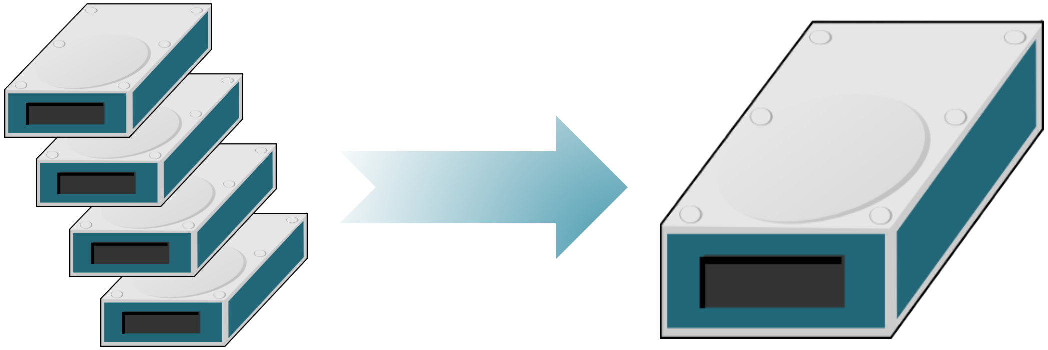
Reference

Operating Systems: Three Easy Pieces (chapter 38)
Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
WWW: **http://pages.cs.wisc.edu/~remzi/OSTEP/**

*Original usage of the term RAID: Redundant Array of ***Inexpensive*** Disks

Redundant Array of Independent Disks

Storage capacity	• A virtual (logical) disk aggregating storage space from multiple disks
I/O speed	• Enhance overall data read/write throughput by parallelizing data access
Fault tolerance	• Lessen the impact of individual disk failures

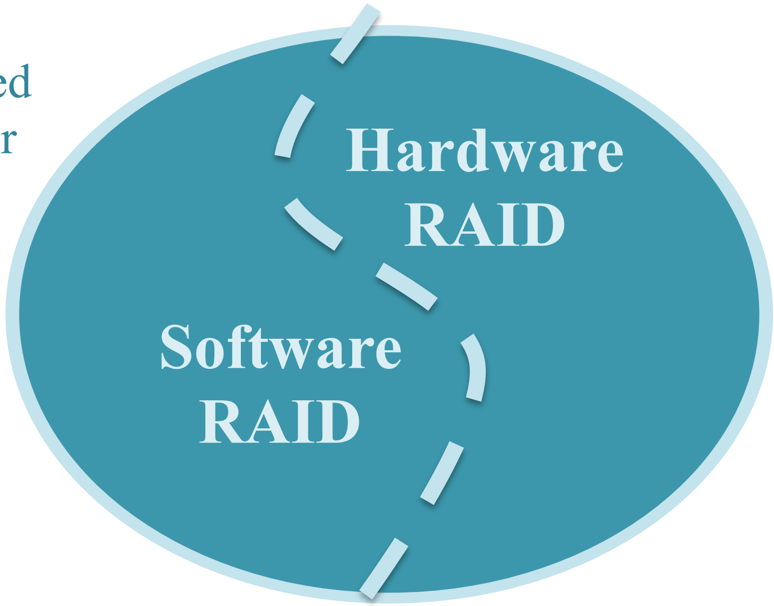


*Original usage of the term RAID: Redundant Array of ***Inexpensive*** Disks

RAID Implementation

S/W RAID

- Functions are performed by the system processor using special software routines
- e.g., Linux: mdadm
- Competes for CPU cycles with other tasks



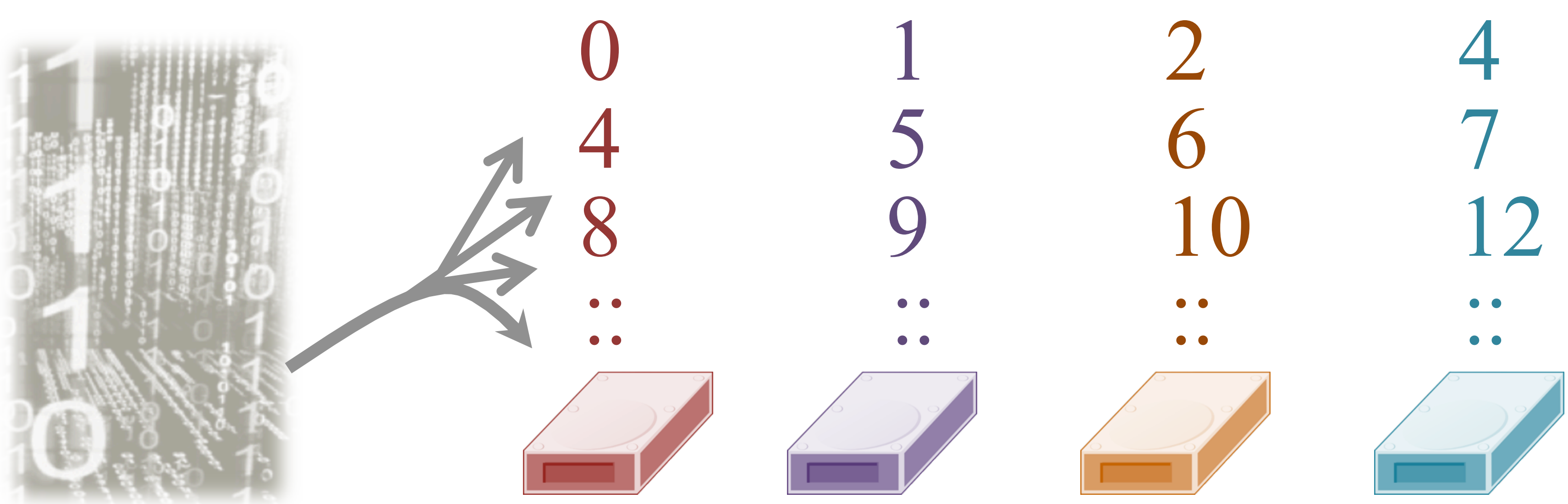
H/W RAID

- Dedicated hardware to control the array
- Transparent to the OS
- Hardware integrated with the computer
- Intelligent, external RAID controller

RAID Level 0: Striping

- ⌘ No redundancy ➔ **No fault-tolerance**
- ⌘ Chunk size **not necessarily same** as file system Block size
- ⌘ Simple striping: Spread chunks across disks in a **round robin** manner

Load across disks is **uniformly distributed** when using **RAID 0**. This is in *contrast* to a **Just a Bunch of Disks (JBOD)** which creates a spanned volume (linear/chain RAID).



Striping Implications

Read/Write throughput ↑

The RAID mapping problem

- Given a logical block to read/write, which physical disk and offset to access?

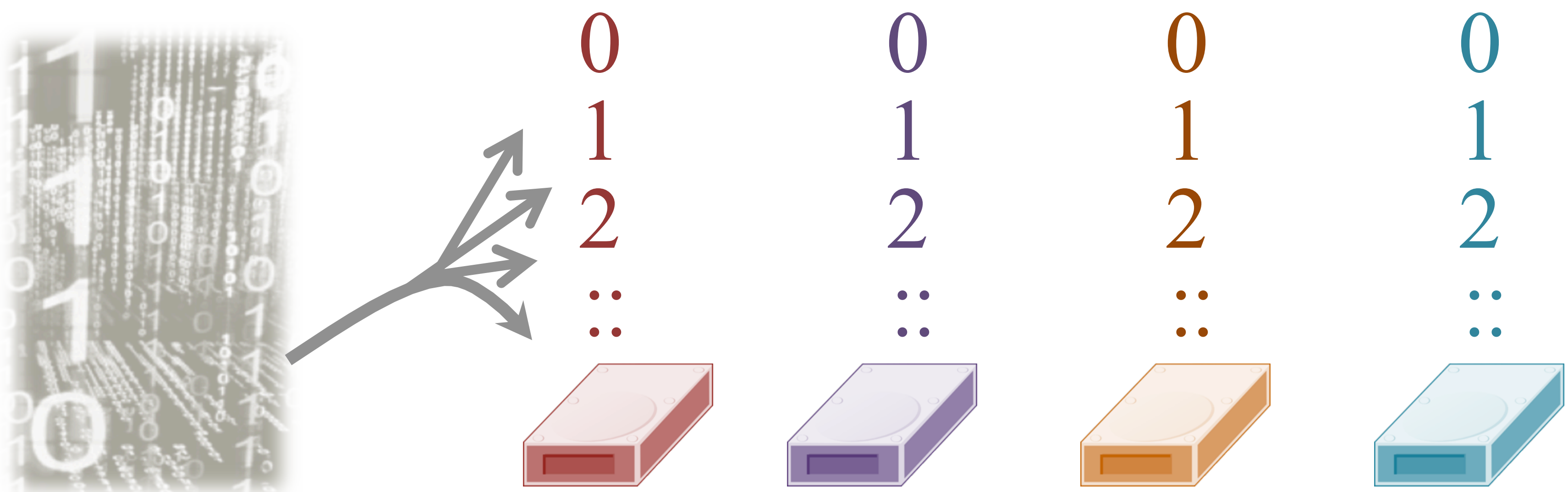
Chunk size implications

- Small chunks → high parallelism for intra-file access
- Need for disk spindle synchronization
- Big chunks → parallelism more likely if concurrent requests

RAID Level 1: Mirroring

- Mirror data over N disks
  ➔ **Tolerate failure of up to N-1 disks**
- Storage inefficient
  (1/N space utilization)

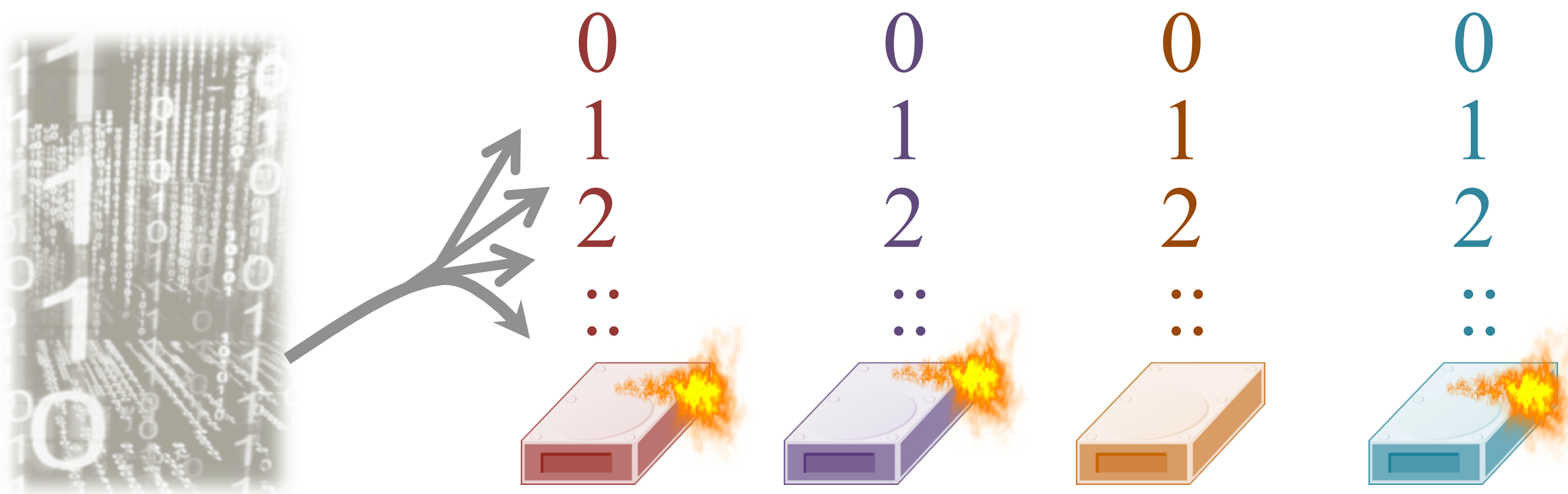
RAID consistency problem:
Arises in all non-trivial RAID configurations.



RAID Level 1: Mirroring

- Mirror data over N disks
  - ➔ **Tolerate failure of up to N-1 disks**
- Storage inefficient (1/N space utilization)

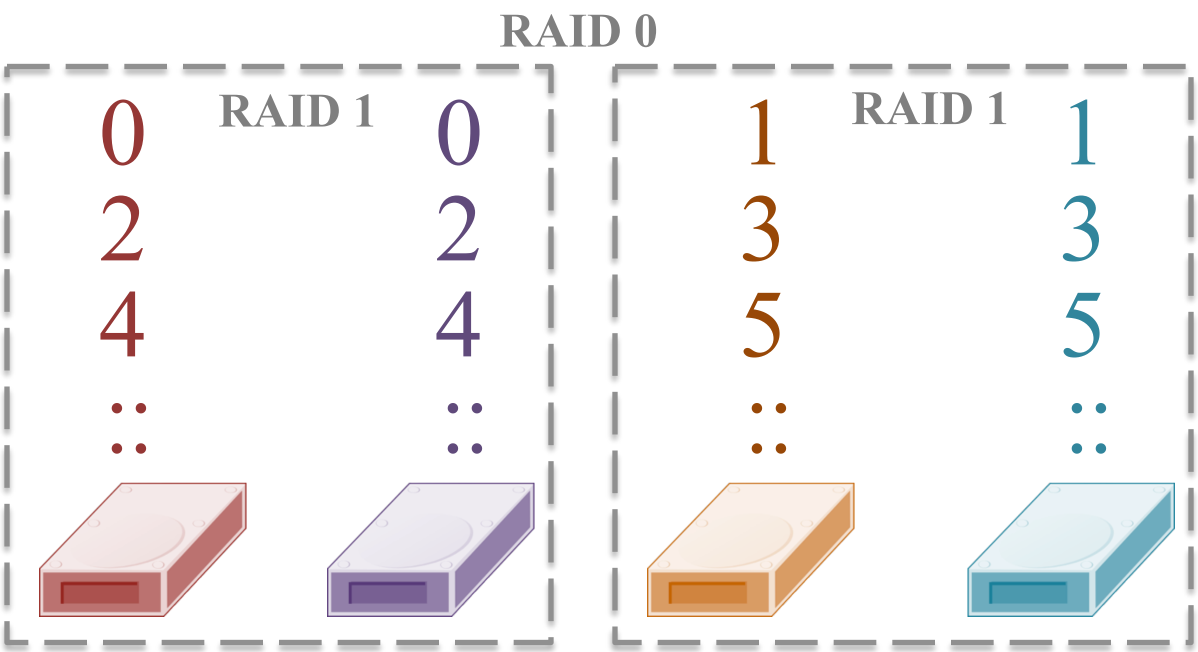
RAID consistency problem:
Arises in all non-trivial RAID configurations.



RAID Level 10: Mirroring + Striping

- ⌘ Tolerate **1 arbitrary disk failure**
- ⌘ Alternative configuration: RAID 01

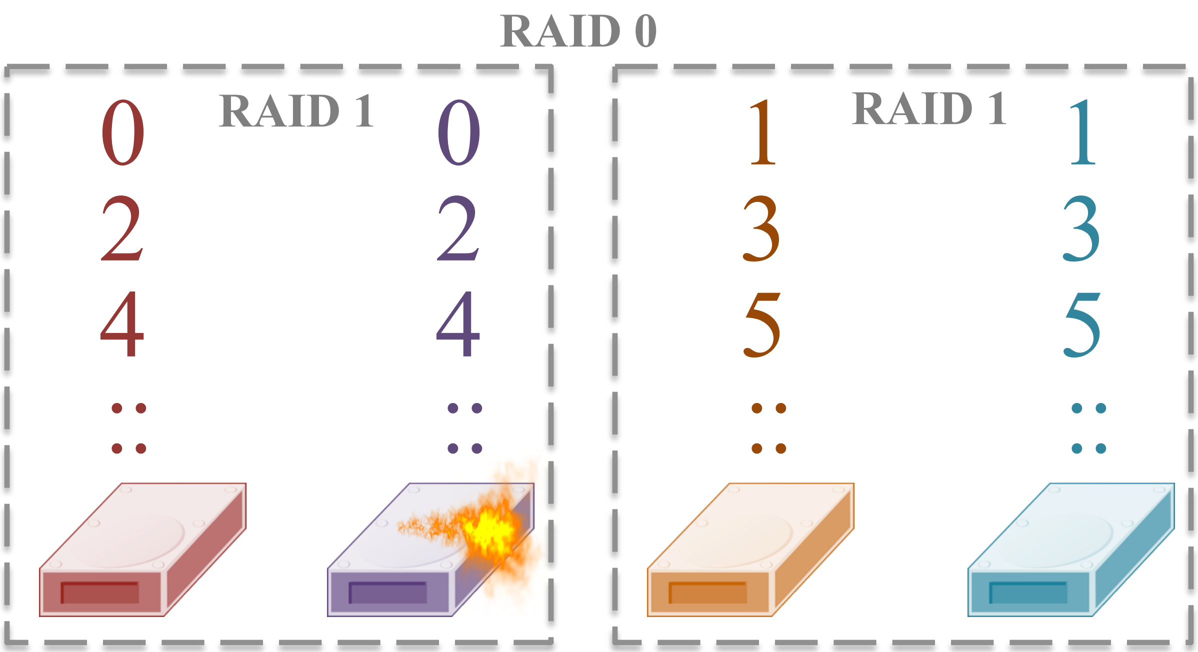
Expensive: With mirroring level of 2, total usable storage is N/2



RAID Level 10: Mirroring + Striping

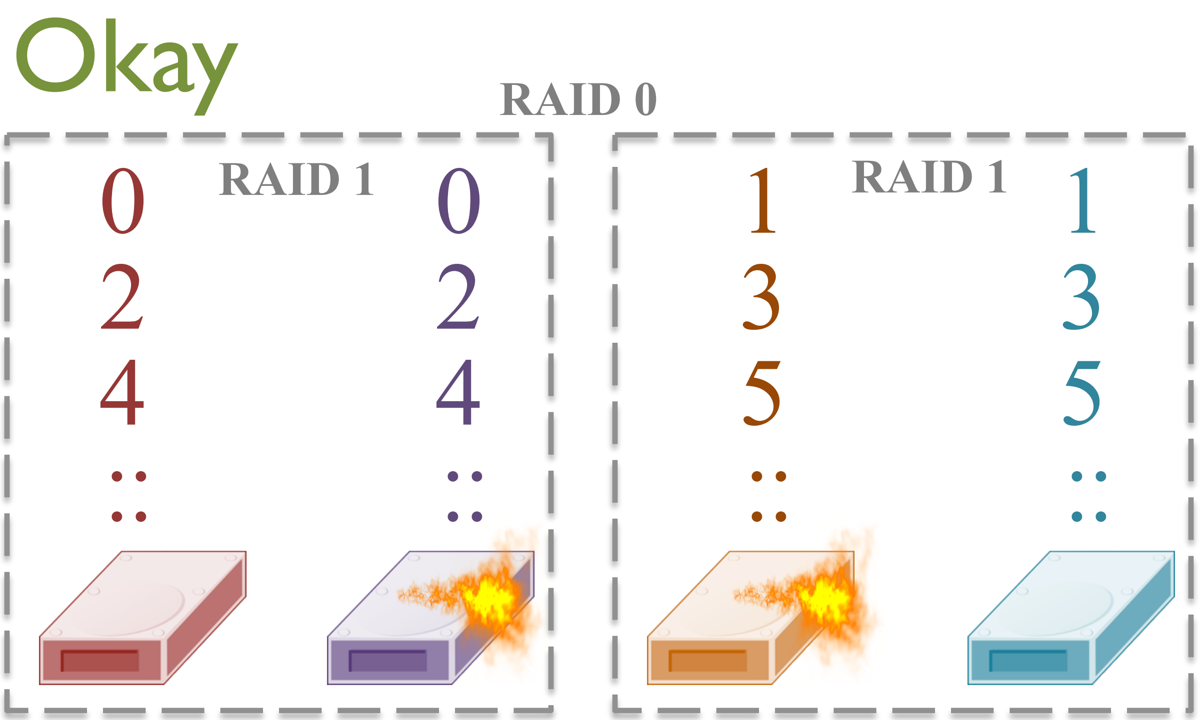
- Tolerate **1 arbitrary disk failure**

Expensive: With mirroring level of 2, total usable storage is N/2



RAID Level 10: Mirroring + Striping

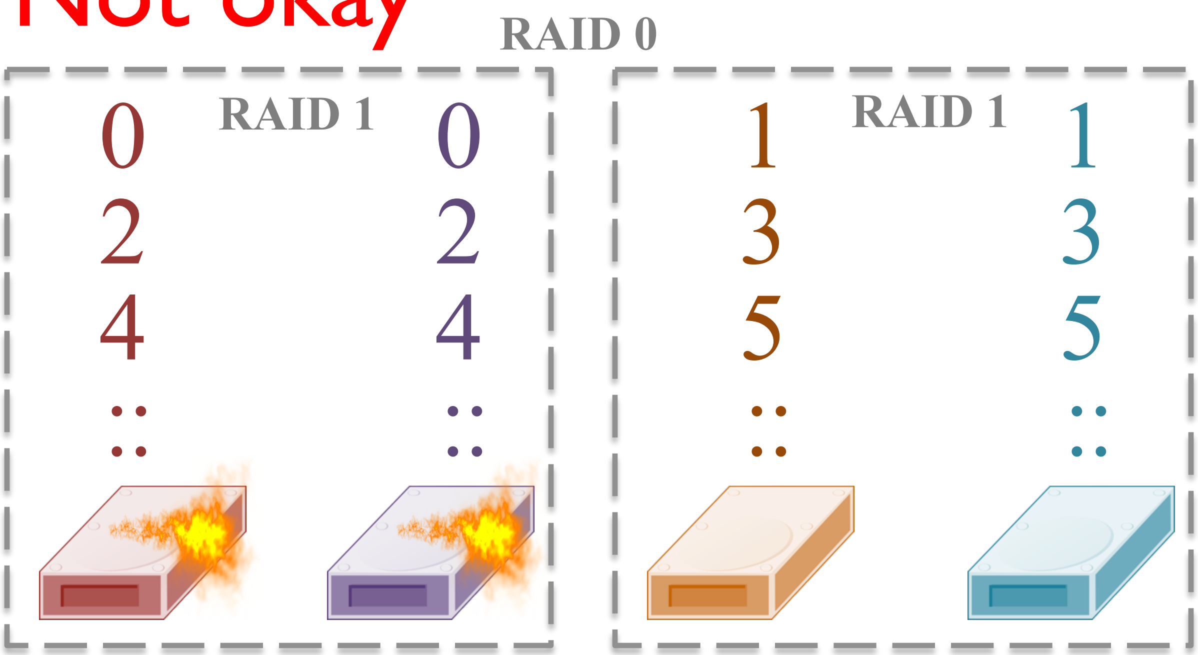
Some instances of **double disk failures** may be tolerated



RAID Level 10: Mirroring + Striping

Some instances of **double disk failures** can **NOT** be tolerated

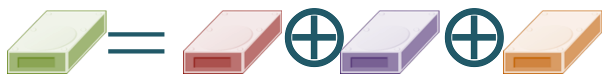
Not okay



Tolerating (single) disk failure

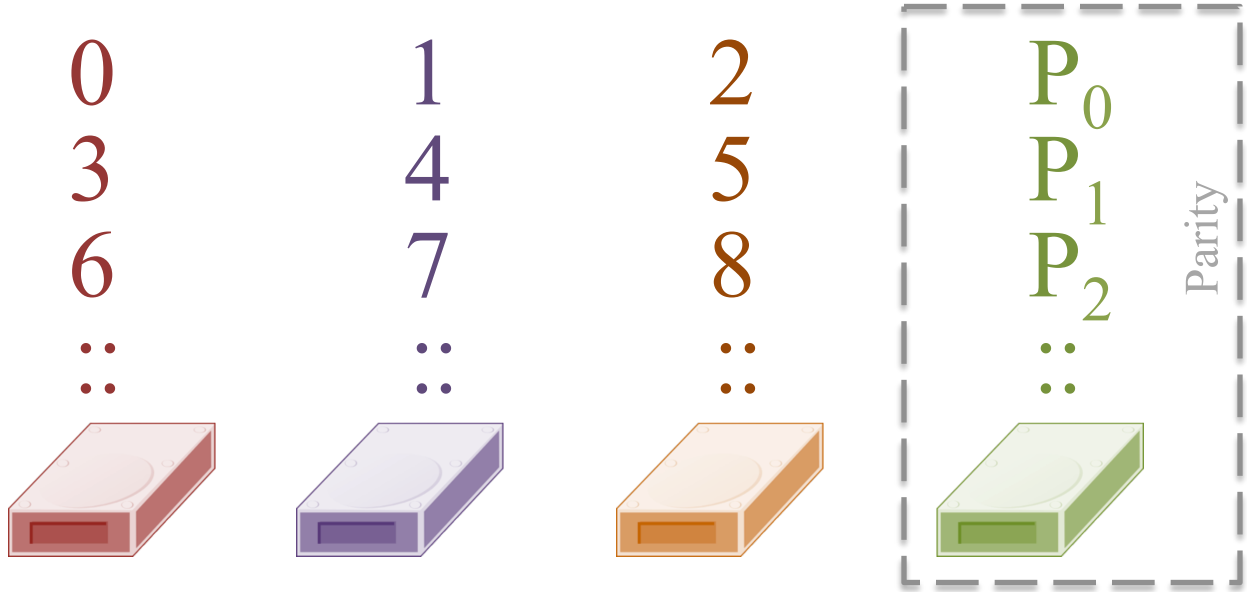
⌘ What is the best possible strategy?
(w.r.to. storage efficiency)

RAID 4: Using parity

- RAID 4: Store data stripes in N-1 disks, **parity** in N^{th} disk
- Parity: 

Improves space utilization, trading it off against performance

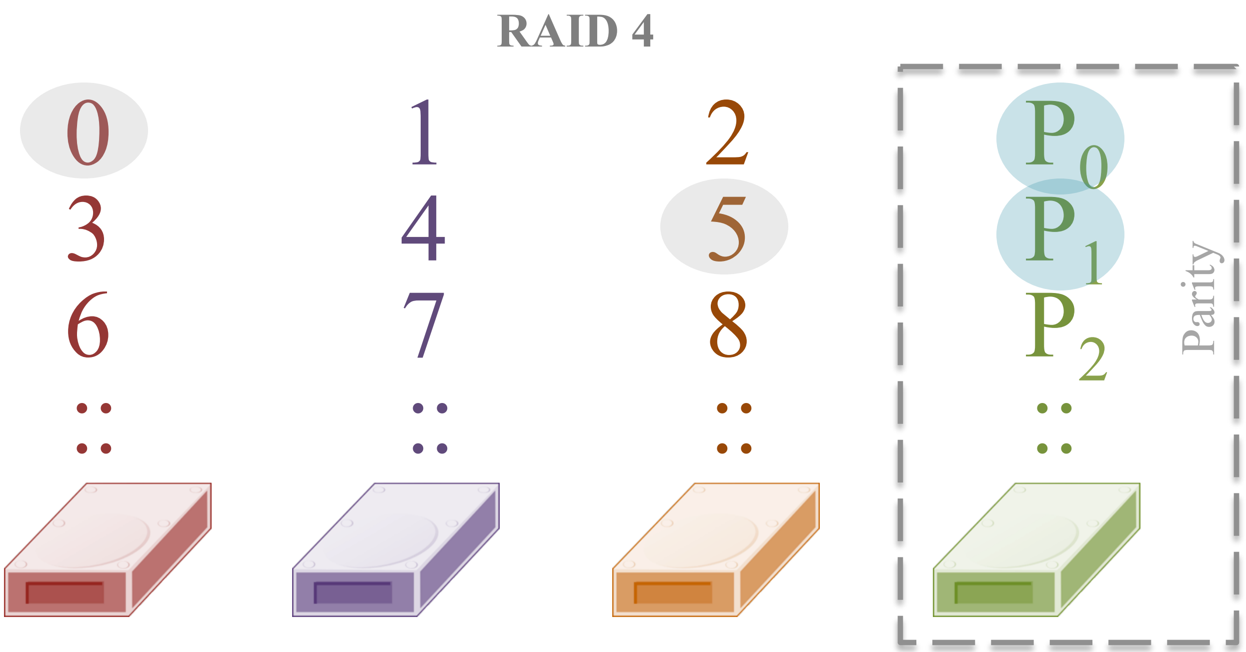
RAID 4



* RAID 2 & 3 are obsolete, and we won't discuss them

Small write problem

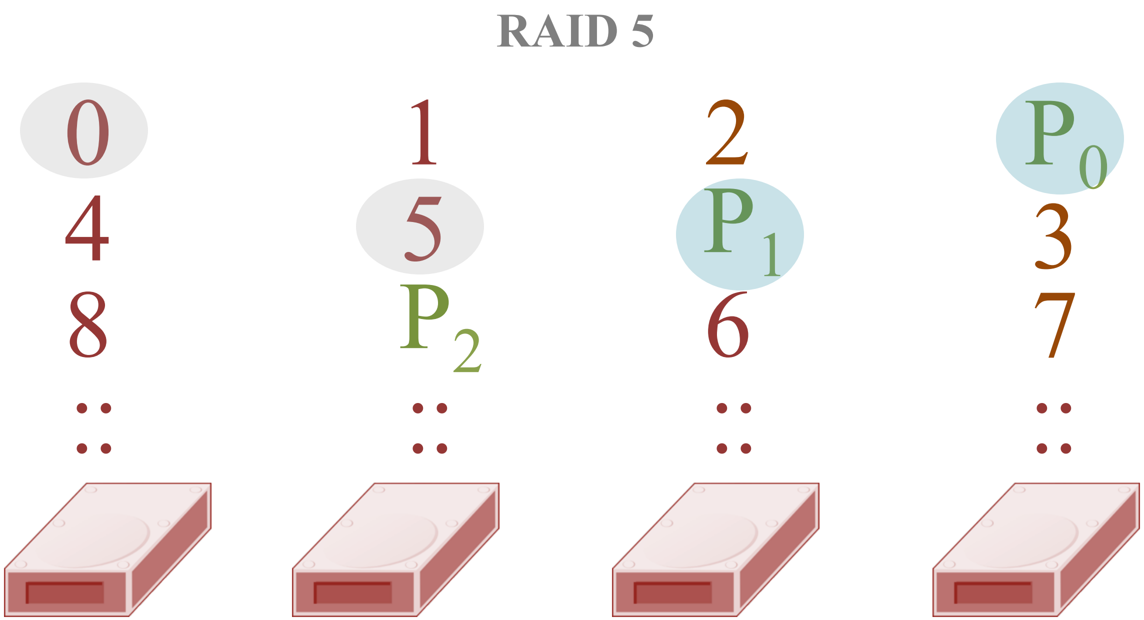
- ⌘ The parity disk becomes an I/O bottleneck



RAID 5

- RAID 5: Distribute the parity over disks

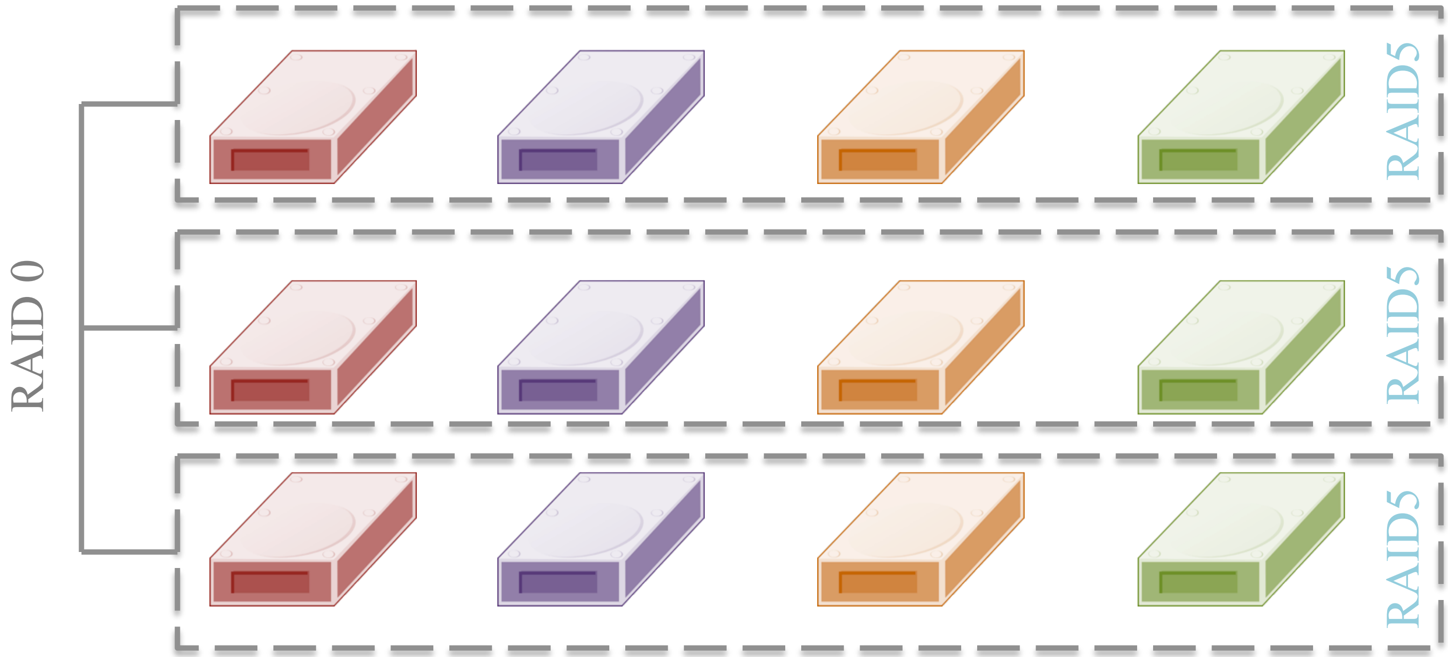
Partly addresses the **small-write problem** of RAID 4



RAID Levels 4 & 5: Using parity

- Another very popular deployment model is RAID 50

Smaller group of disks affected by a single failure. Better degraded performance & recovery

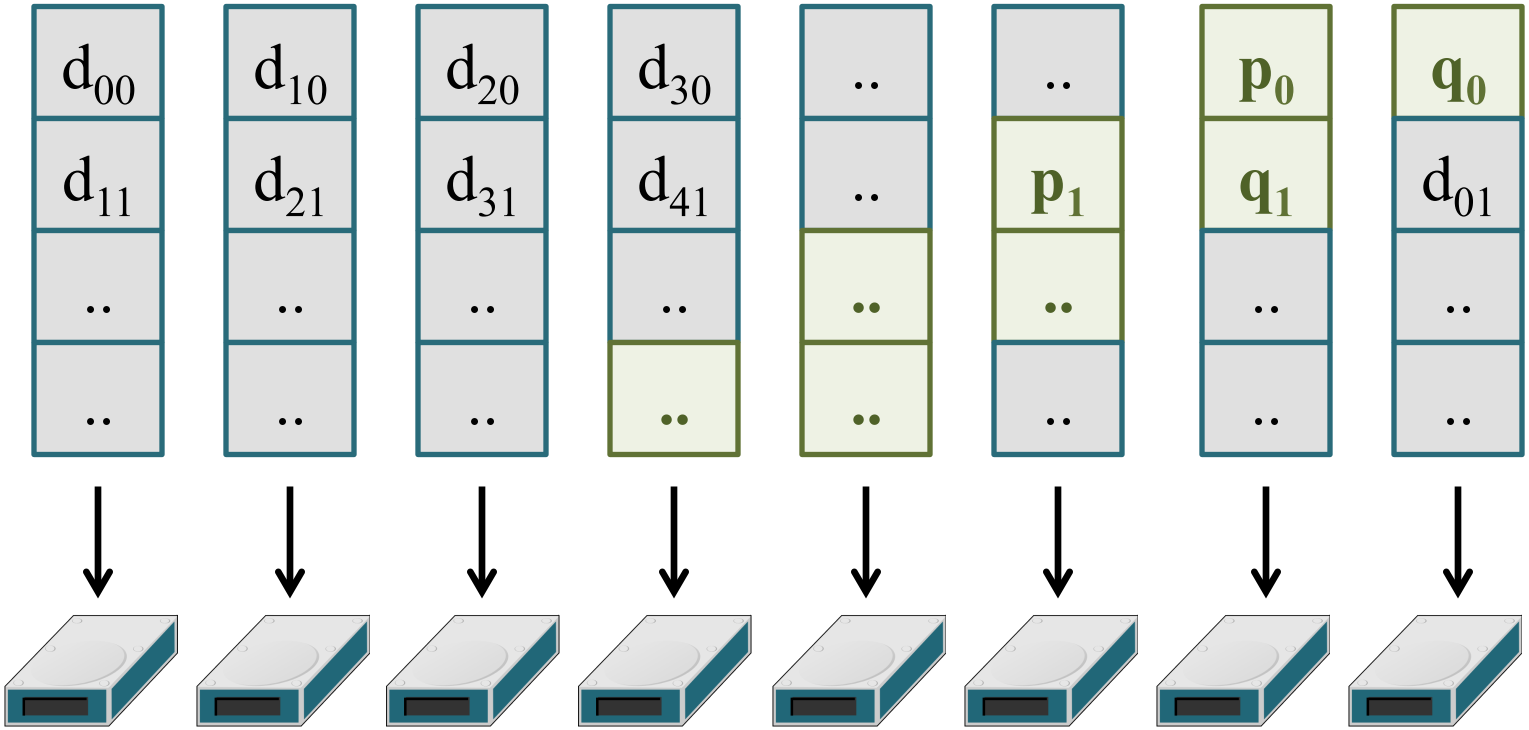


Single parity systems are fragile

- Larger capacity disks lead to longer rebuild time
- What happens if another disk fails before disk rebuild is completed?

RAID6: Using two parities

- RAID 6: **Parities p & q**
  (typically) distributed over disks



How do we compute two parities?

- Generic mechanism is to utilize MDS **erasure codes**
  e.g., **Cauchy (Reed-Solomon) codes**

There are several schemes specifically optimized for RAID-6, e.g.

- EvenOdd
- Liberation/Liber8tion
- Row-Diagonal Parity
- etc

Erasure codes for storage

Reference

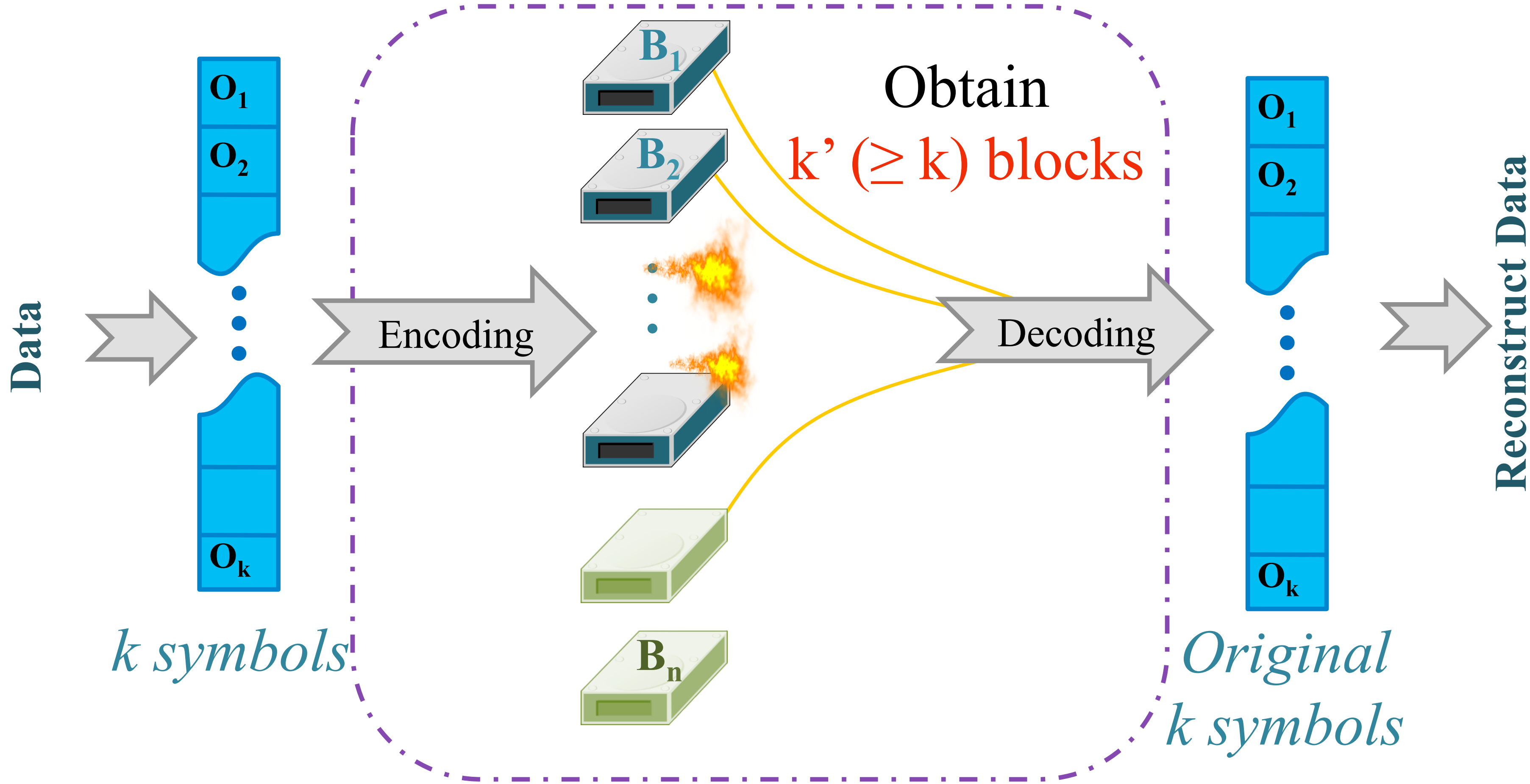
Tutorial on Erasure Coding for Storage Applications (part 1)
James S. Plank, USENIX FAST 2013
http://web.eecs.utk.edu/~plank/plank/papers/FAST-2013-Tutorial.html

Reference

Coding Techniques for Repairability in Networked Distributed Storage Systems (chapter 3)
Frédérique Oggier, Anwitaman Datta
NOW Publishers FnT Communications & Information Theory Survey
http://pdcc.ntu.edu.sg/sands/CodingForNetworkedStorage/pdf/longsurvey.pdf

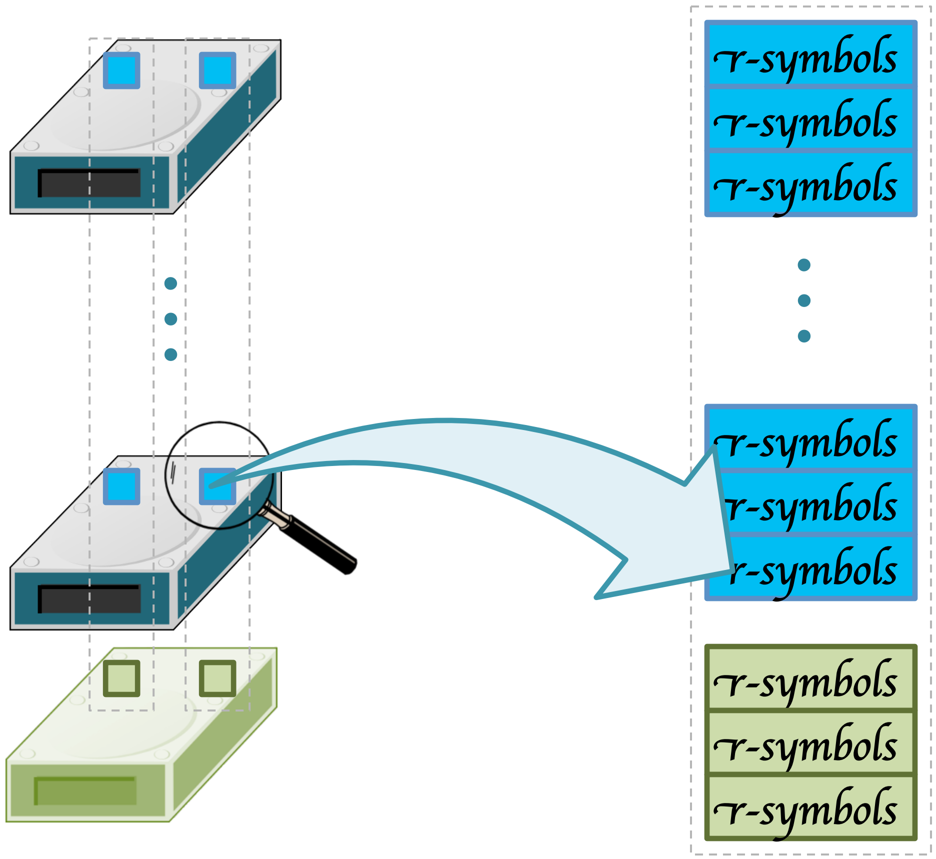
Acknowledgement: Parts of the following content & visualization are based on Prof. Plank's tutorial.

Erasure codes (EC) for storage



Systems perspective

- ⌘ Conceptually, computations for coding are carried out using ***w*-bit symbols**
- ⌘ The implementation groups multiple (r) such ***w*-bit** symbols together
- ⌘ The stripes stored in the disks are of yet another size
- ⌘ Parity stripes may further be distributed across disks



Erasure codes

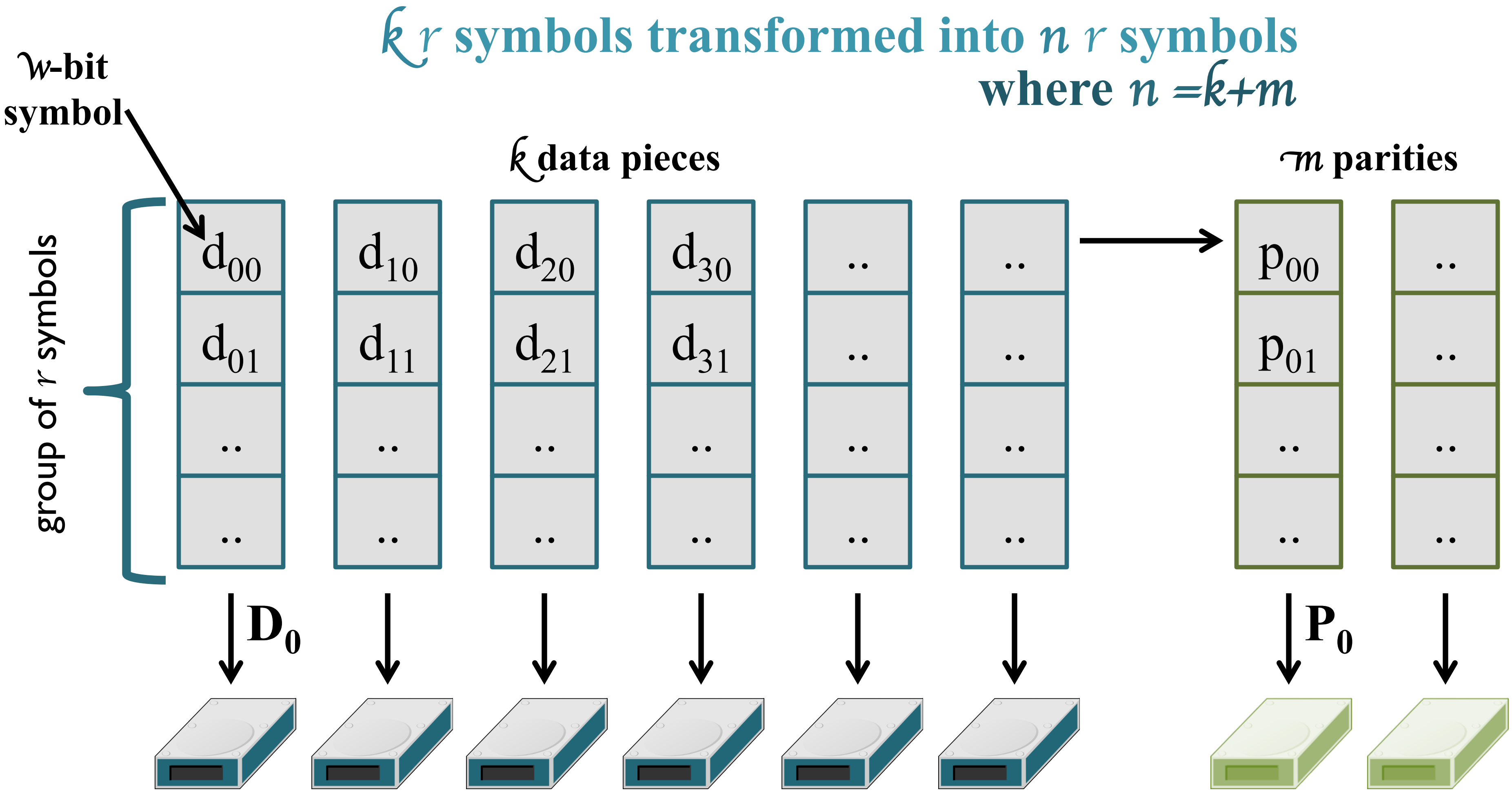
MDS codes

- A **maximum distance separable** code will allow reconstruction of the original k symbols using any subset of k-out-of-n distinct symbols.

Systematic codes

- If all the **original k symbols are present** in the resulting n symbols after the coding process, we call the resulting code as systematic, otherwise, we call it non-systematic.

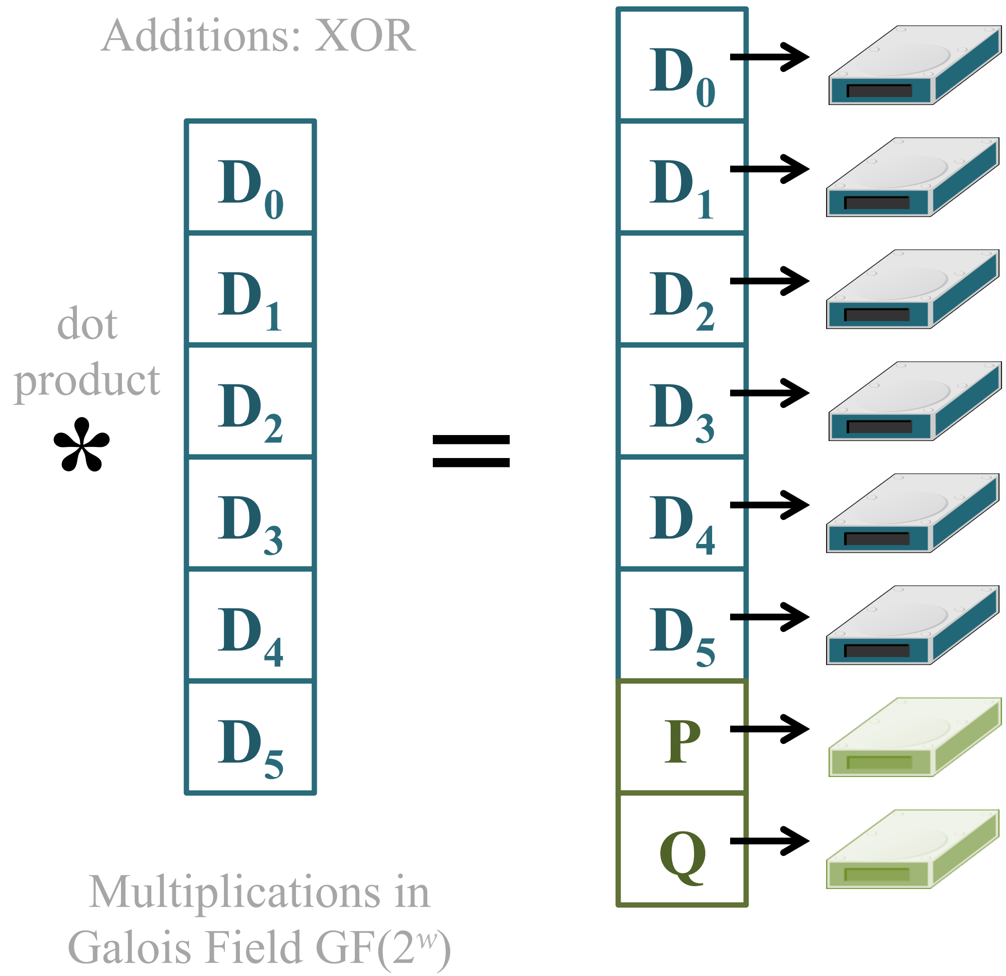
Systematic erasure code



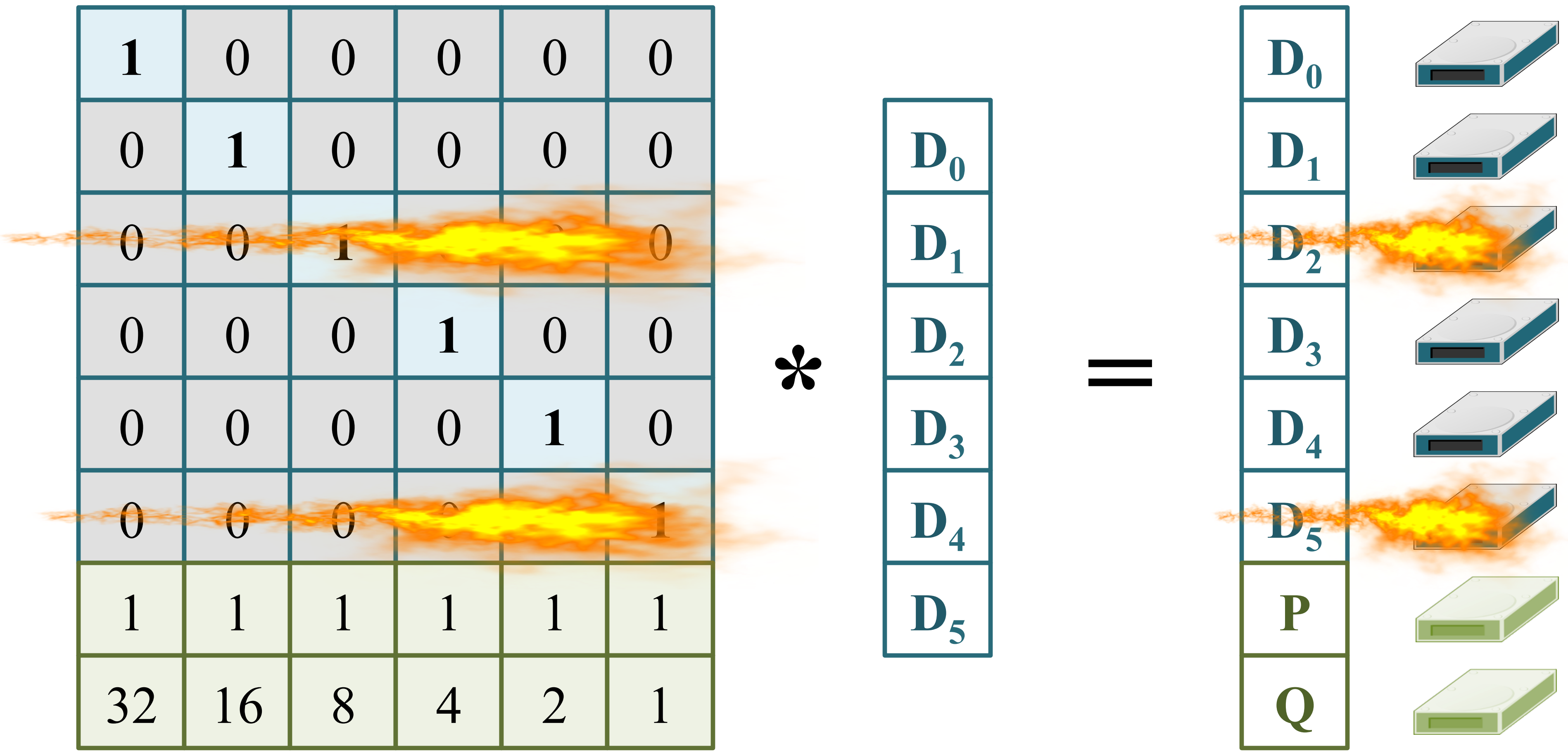
Linux RAID6 Example: k=6, n=8, r=1, w=8

1	0	0	0	0	0
0	**1**	0	0	0	0
0	0	**1**	0	0	0
0	0	0	**1**	0	0
0	0	0	0	**1**	0
0	0	0	0	0	**1**
1	1	1	1	1	1
32	16	8	4	2	1

Generator Matrix (G^T)



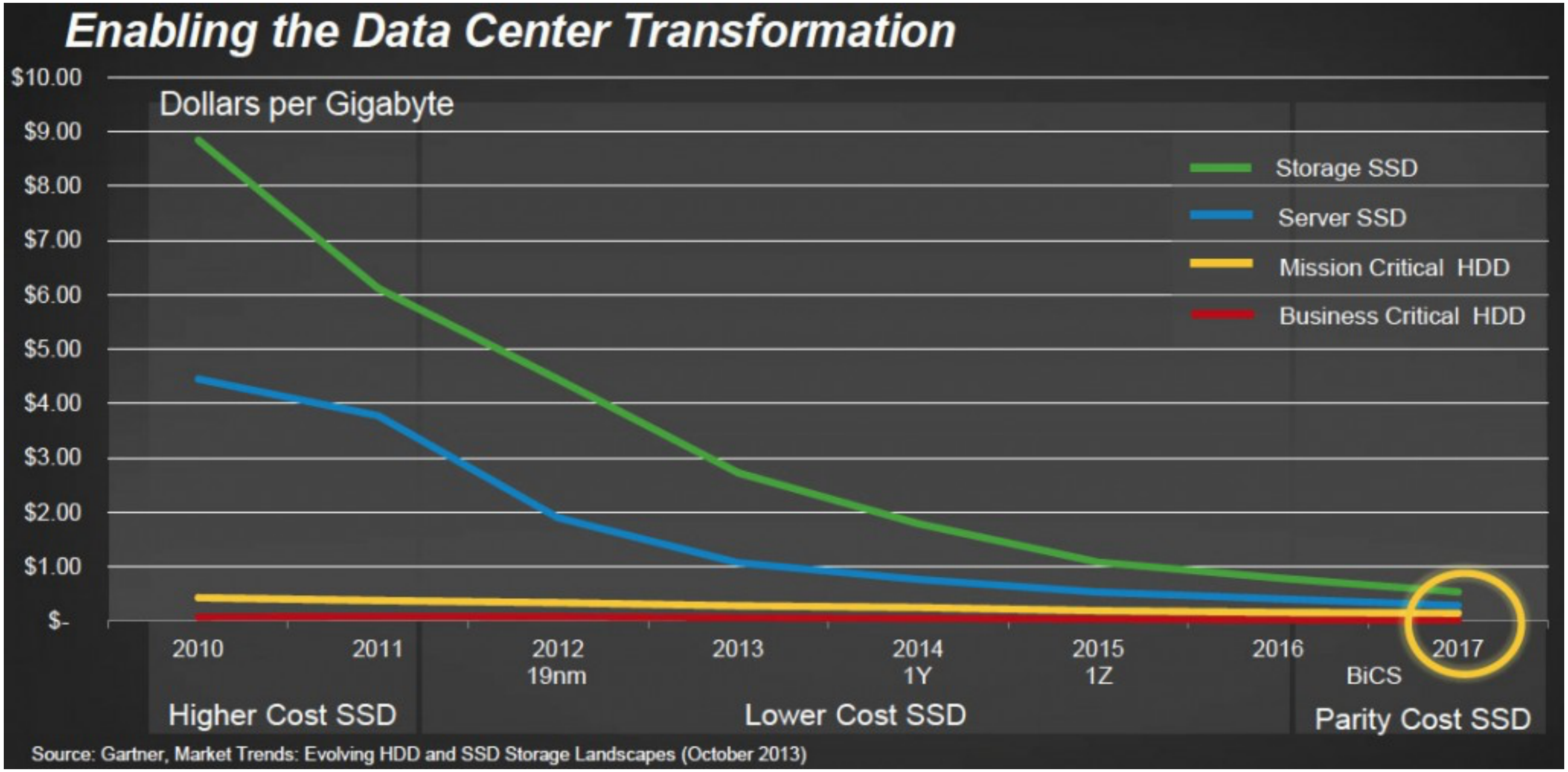
Decoding



Decoding: Solve the remaining linear equations (e.g., using Matrix inversion)

The rise of SSD

⌘ Lot of original RAID design issues may be irrelevant/need to be revisited



BEYOND

RAID

- ⌘ RAIN: Redundant Array of Independent Nodes
- ⌘ Non-MDS codes: Repairable Storage Codes

Erasure codes for storage

Reference

Coding Techniques for Repairability in Networked Distributed Storage Systems (chapters 2, 7)
Frédérique Oggier, Anwitaman Datta
NOW Publishers FnT Communications & Information Theory Survey
http://pdcc.ntu.edu.sg/sands/CodingForNetworkedStorage/pdf/longsurvey.pdf

Cloud and RAIN

⌘ **Half a trillion photographs** uploaded to the web **in a year** [2015]

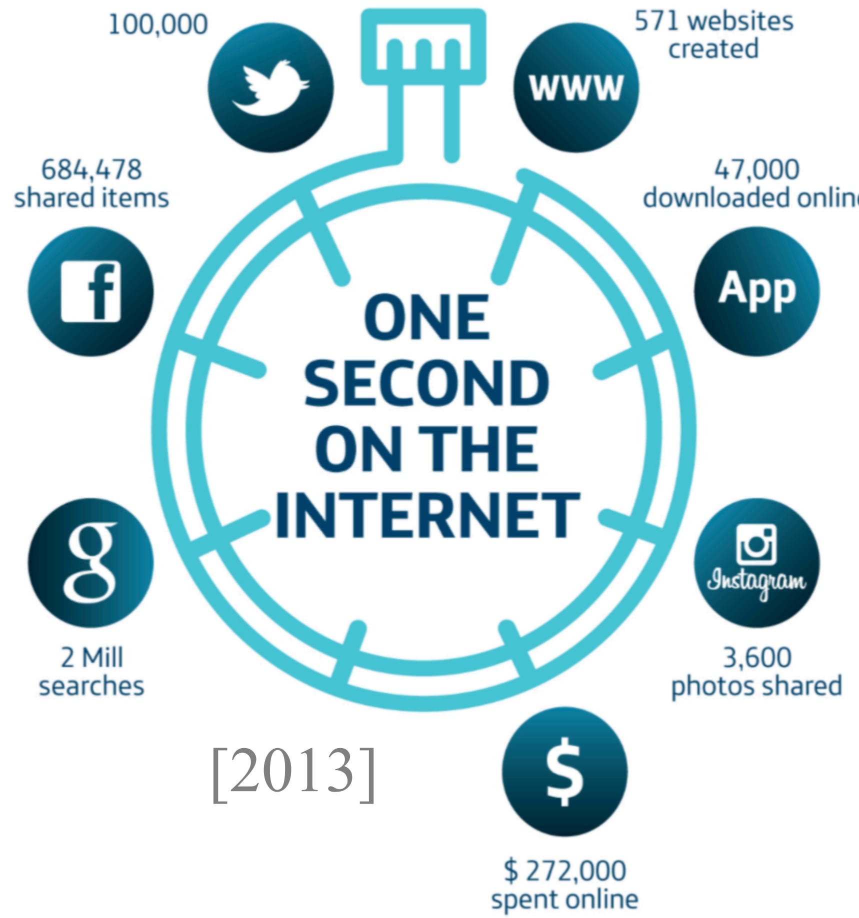
⌘ **2.5 billion gigabytes (GB) of data** was **generated every day** in 2012 [IBM]

⌘ **Three hundred hours of video** uploaded to YouTube **every minute** [Dec 2014]

Storage solutions at unprecedented

BIG DATA

scales

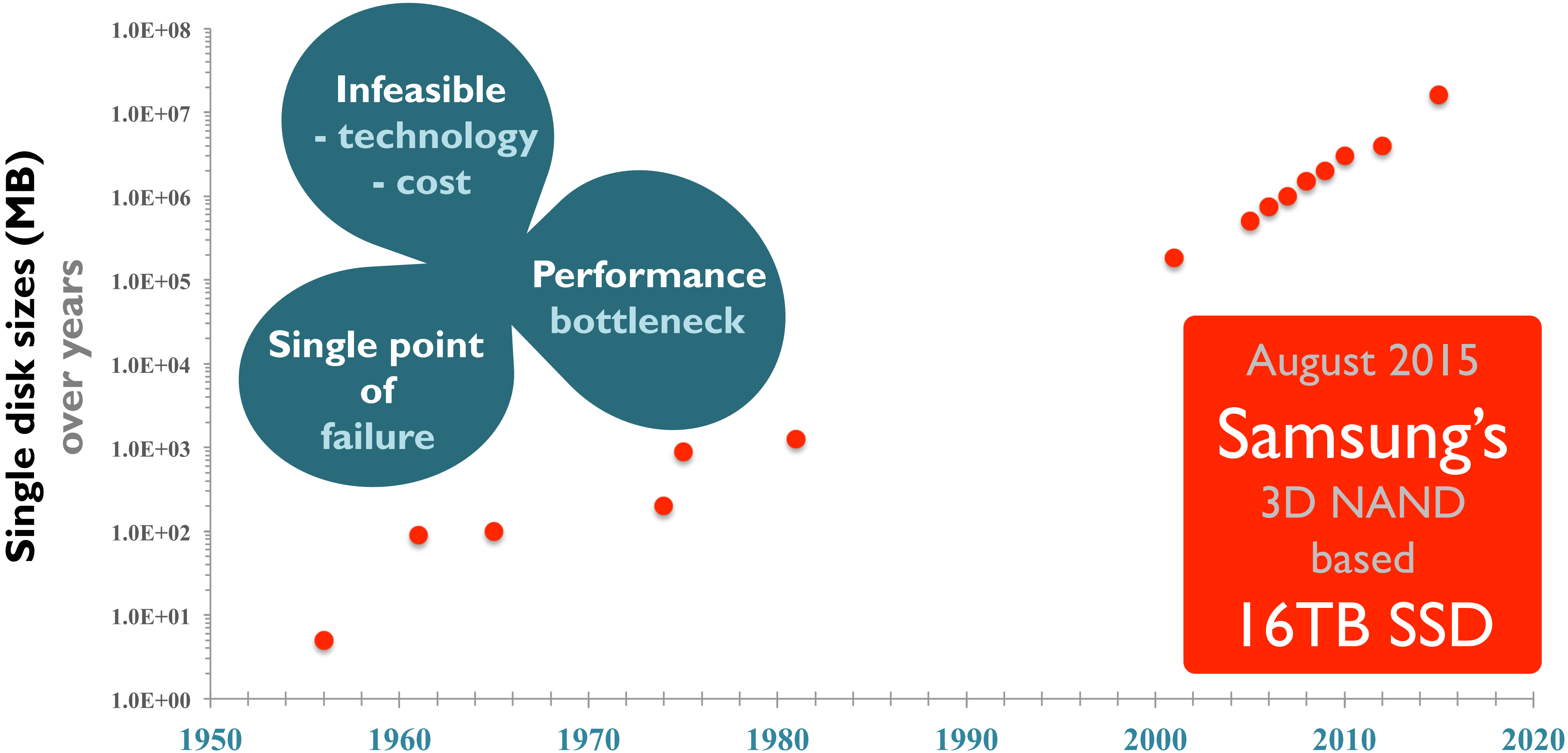


[2013]

Source: Telefónica analysis based on Social and Digital Media Revolution Statistics 2013 from MistMediaGroup (http://youtube.com/watch?v=Slb5x5fixk4).

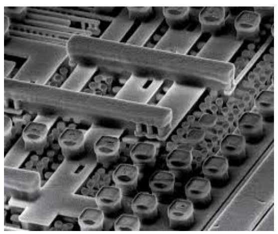
Scale up

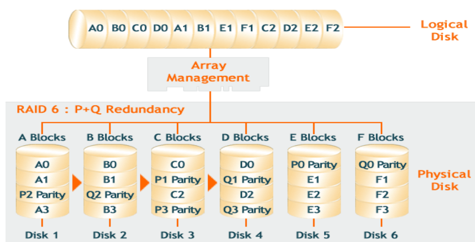
⌘ **Scale up** (vertically): Add resources to a single node in a system

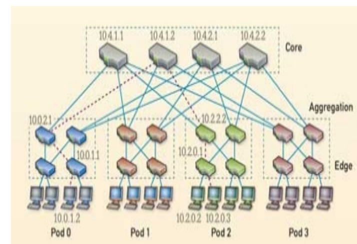


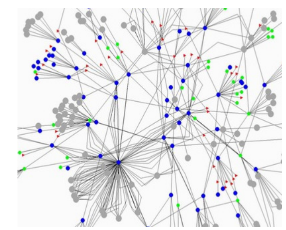
Scale out

⌘ **Scale out** (horizontally): Add more nodes to a system running distributed applications









Storage drives — RAID — RAI"N" — P2P/edge/fog

Granularity of distribution

Not distributing is not an option

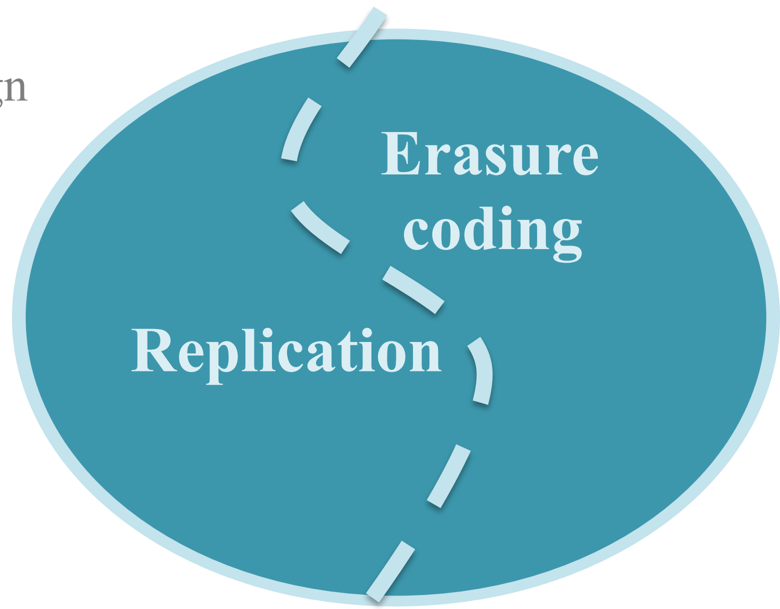
- ⌘ Added complexities and vulnerabilities
  Latency, network partitions, faults, …
- ⌘ Consistency, Availability and Partition tolerance
  CAP theorem – choose any two?
- ⌘ RAID like solution needed for fault-tolerance, but across nodes
  RAIN: Redundant Array of Independent Nodes
  Each node may apply some RAID configuration within

Need to tolerate more than two failures

- Many more nodes in the system
- More sources of disruptions: power, network switches, …

Replication

- **Simpler** system design
- Not computation intensive
- Storage inefficient



Erasure coding

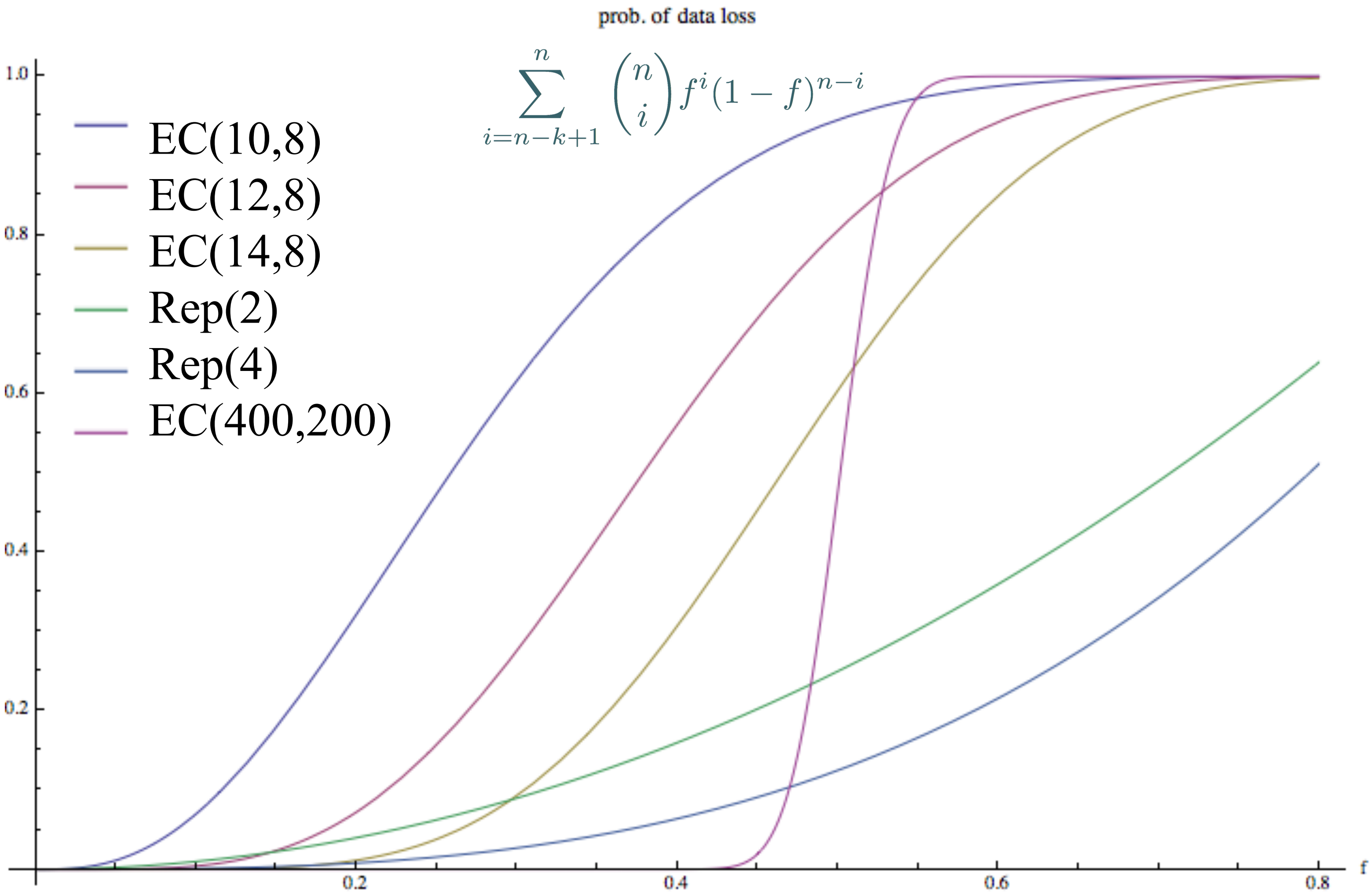
- **Expensive** data access & modification
- Distributed over a larger number of nodes
- System **complexity**
- Storage **efficient**

Storage efficiency of erasure codes

- ECs provide better fault-tolerance versus storage trade-off
- Static resilience analysis of **MDS ECs** with parameters **(n,k)**
  Replication is a special case EC with k=1
- Probability of losing data, if any node fails *iid* with probability f

$$\sum_{i=n-k+1}^{n} \binom{n}{i} f^i (1-f)^{n-i}$$

MDS erasure codes vs replication



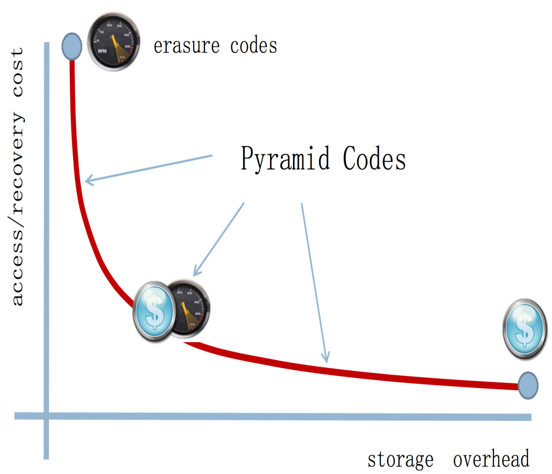
Erasure codes in data centers?

Does erasure coding have a role to play in my data center? [MSR]
- 2010

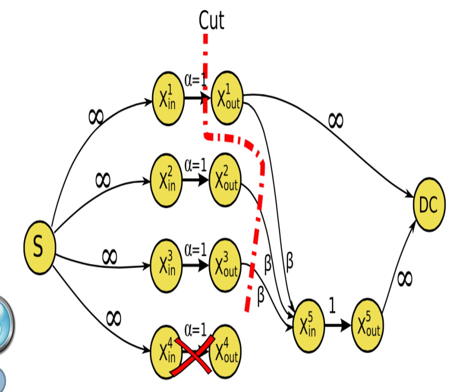
HDFS-RAID
Windows Azure
Google Collosus
- 2011/12

Facebook F4
- 2014

EC as black box
Fault tolerance
- 1999



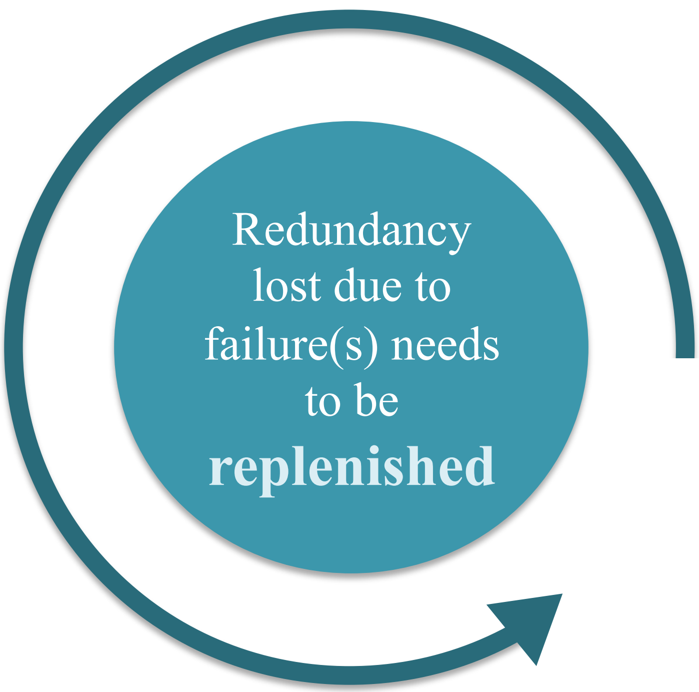
Non-MDS ECs
Improved degraded reads
- 2007



MDS EC + NetCod
B/W efficient repair
- 2007

The repair problem of erasure codes

- What happens when a storage node fails? Can tolerate up to n-k failures
- Initial redundancy provides fault-tolerance, but



Basic repair approach:
Requires data worth **k-symbols** to recreate one lost symbol

Network coding techniques have been proposed to minimize bandwidth usage for repairs (*regenerating codes*), but they suffer from several practicality issues.

Locally reconstructable/repairable codes

Pyramid codes
Huang et al
NCA 2007

All of these are **non-MDS**, i.e., there are localized dependencies among (some) codeword symbols

Self-repairing codes
Oggier & Datta
Infocom 2011, ITW 2011

Local reconstruction codes
Huang et al
USENIX ATC 2012

Used in Windows Azure system

Pyramid code

- ⌘ Underlying principle: **Local & Global parities**
  Created by **composing a MDS** code
- ⌘ Example: Consider a **MDS (11,8) code**

$$[u_1, \ldots, u_8]G = [u_1, \ldots, u_8, p_1, p_2, p_3]$$

Pyramid code (contd.)

- Underlying principle: **Local & Global parities**
  Created by **composing a MDS** code

- Example: Consider a **MDS (11,8) code**

$$[u_1, \ldots, u_8]G = [u_1, \ldots, u_8, g_1, g_2, g_3]$$

- We can create a **(12,8) Pyramid code** using the above MDS code:

$$[u_1, \ldots, u_8]G' = [u_1, \ldots, u_8, \underline{l_{1,1}, l_{1,2}}, g_2, g_3]$$

Where G' is such that

$$l_{1,1} = [u_1, \ldots, u_4, \mathbf{0}]G$$
$$l_{1,2} = [\mathbf{0}, u_5, \ldots, u_8]G$$

$$l_{1,1} + l_{1,2} = g_1$$

Concluding remarks