Big Data Processing Technologies

Chentao Wu
Associate Professor
Dept. of Computer Science and Engineering
wuct@cs.sjtu.edu.cn

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Schedule

- lec1: Introduction on big data and cloud computing
- lec2: Introduction on data storage
- lec3: Data reliability (Replication/Archive/EC)
- lec4: Data consistency problem
- lec5: Block level storage and file storage
- lec6: Object-based storage
- lec7: Distributed file system
- lec8: Metadata management

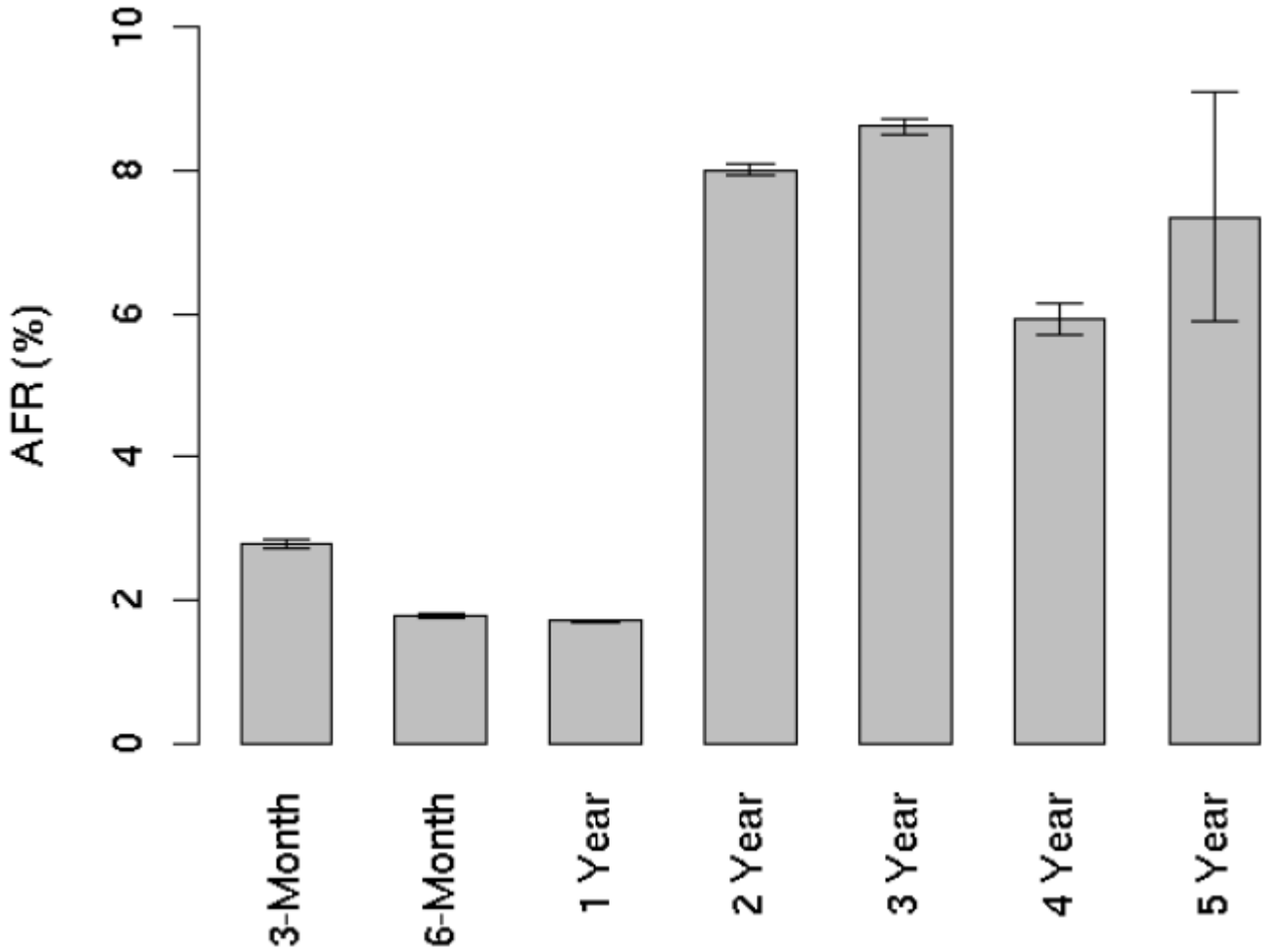
Collaborators

Google 谷歌

阿里云 aliyun.com

DELL EMC

Data Reliability Problem (1)
Google – Disk Annual Failure Rate



Data Reliability Problem (2)
Facebook-- Failure nodes in a 3000 nodes cluster

1 Introduction on Replication

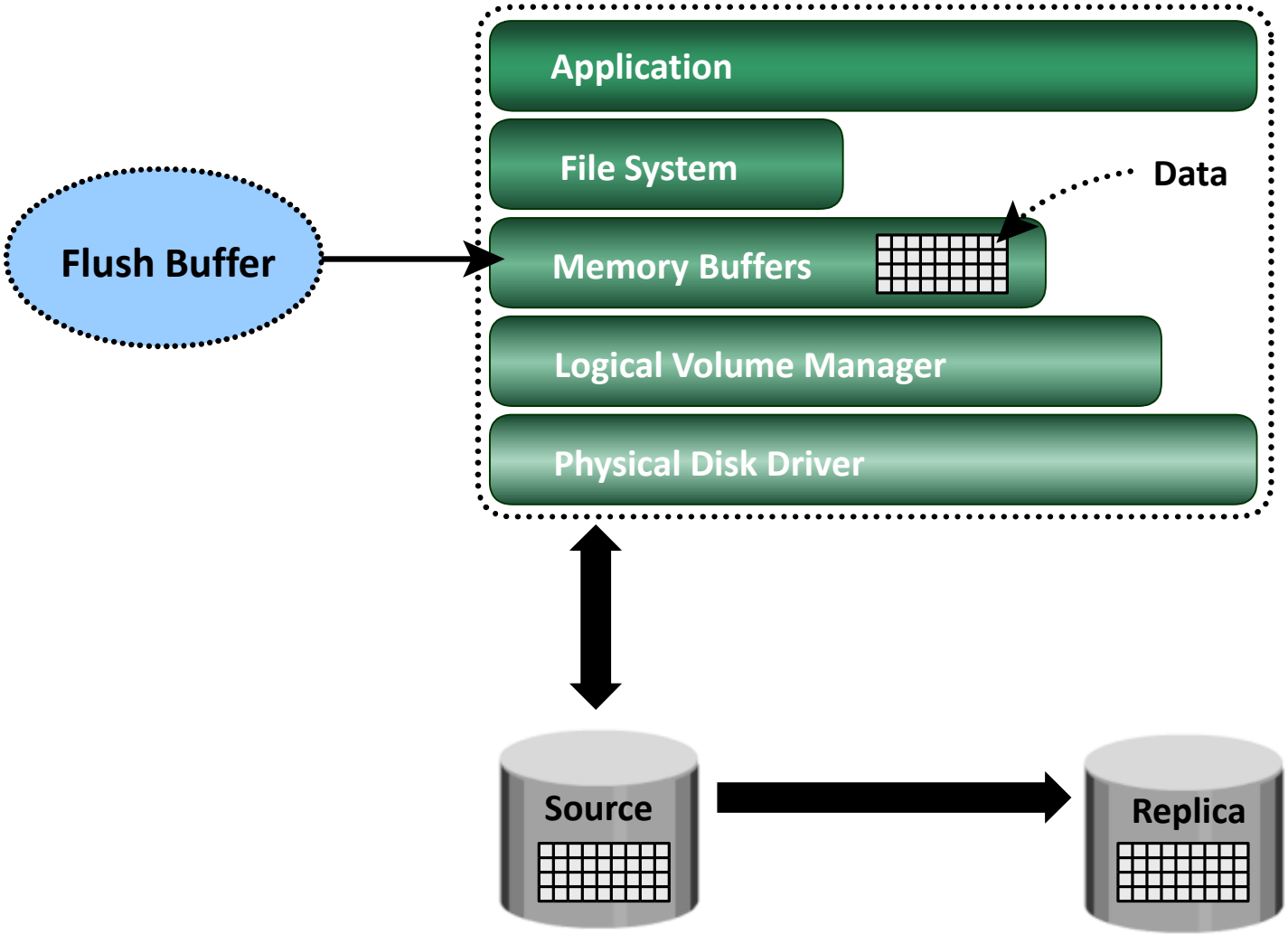
What is Replication?

Replication

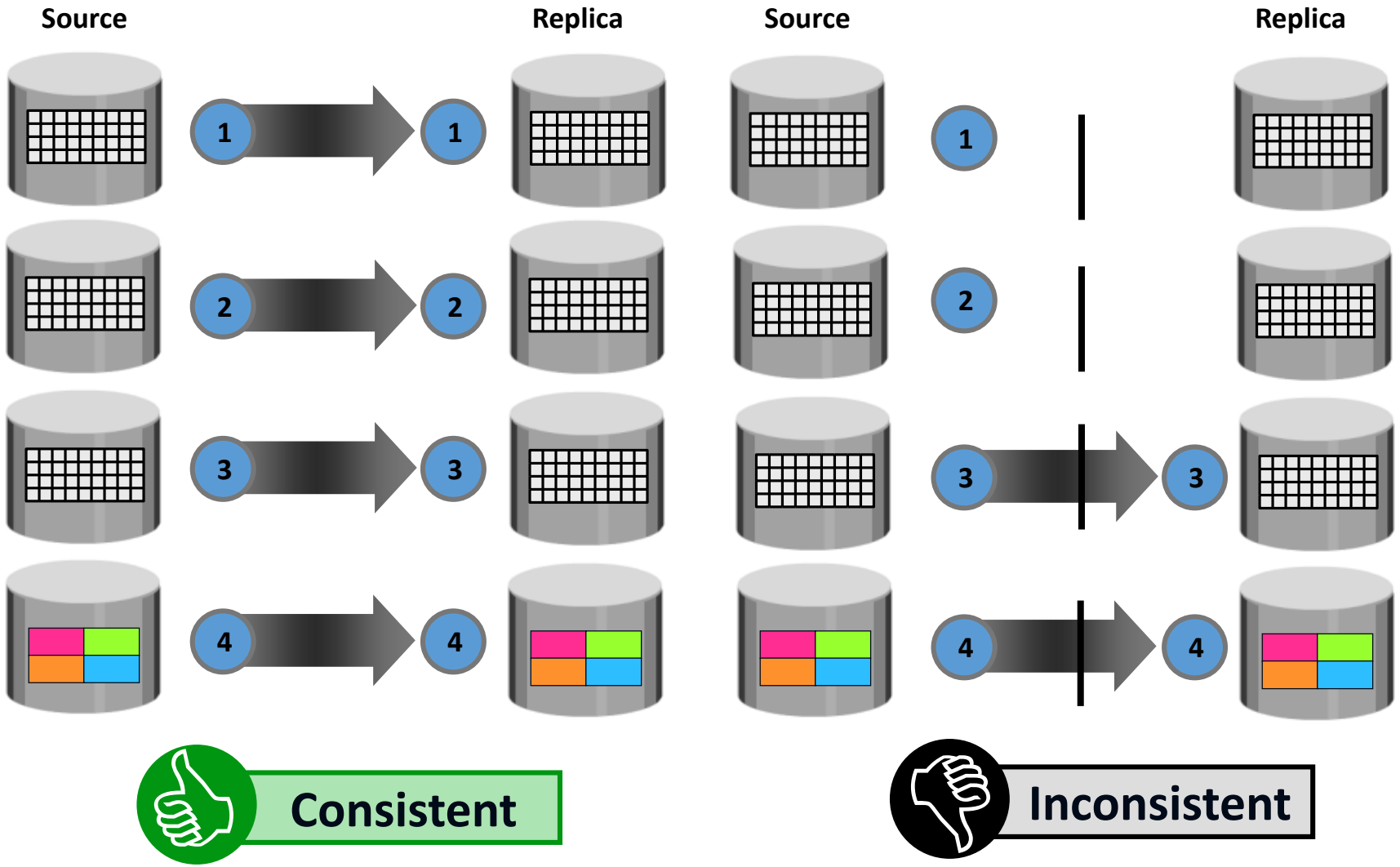
It is a process of creating an exact copy (replica) of data.

- Replication can be classified as
  - Local replication
    - Replicating data within the same array or data center
  - Remote replication
    - Replicating data at remote site

File System Consistency: Flushing Host Buffer

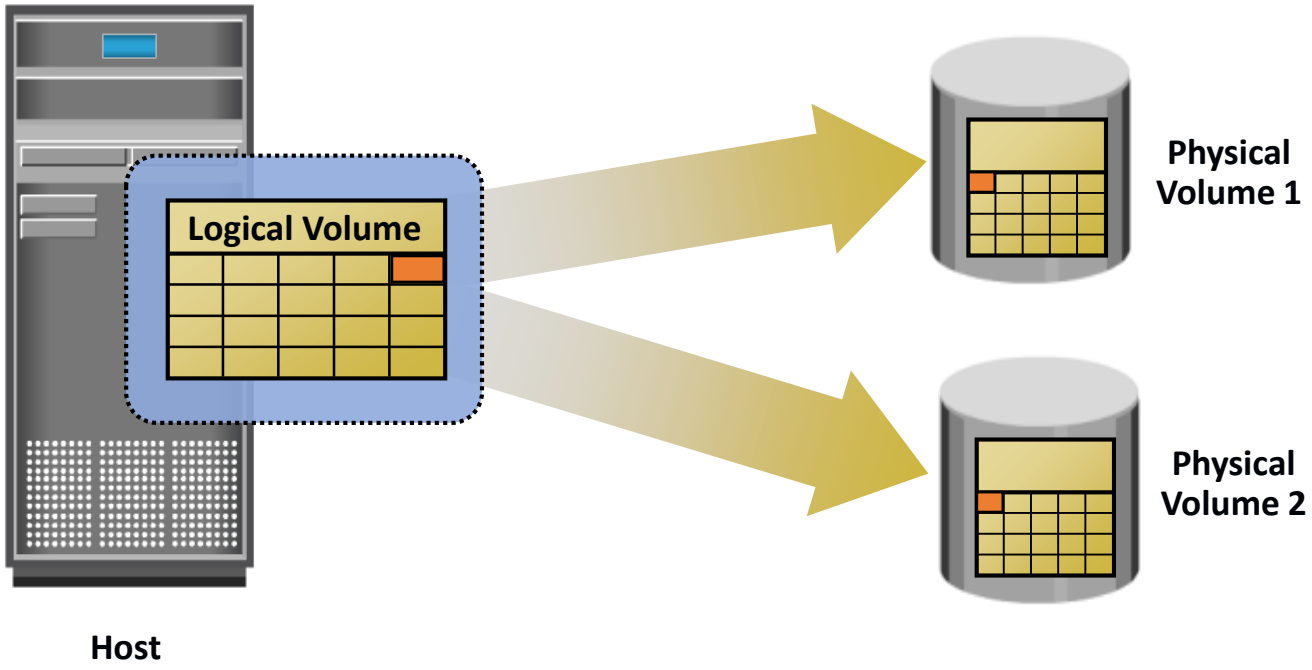


Database Consistency: Dependent Write I/O Principle



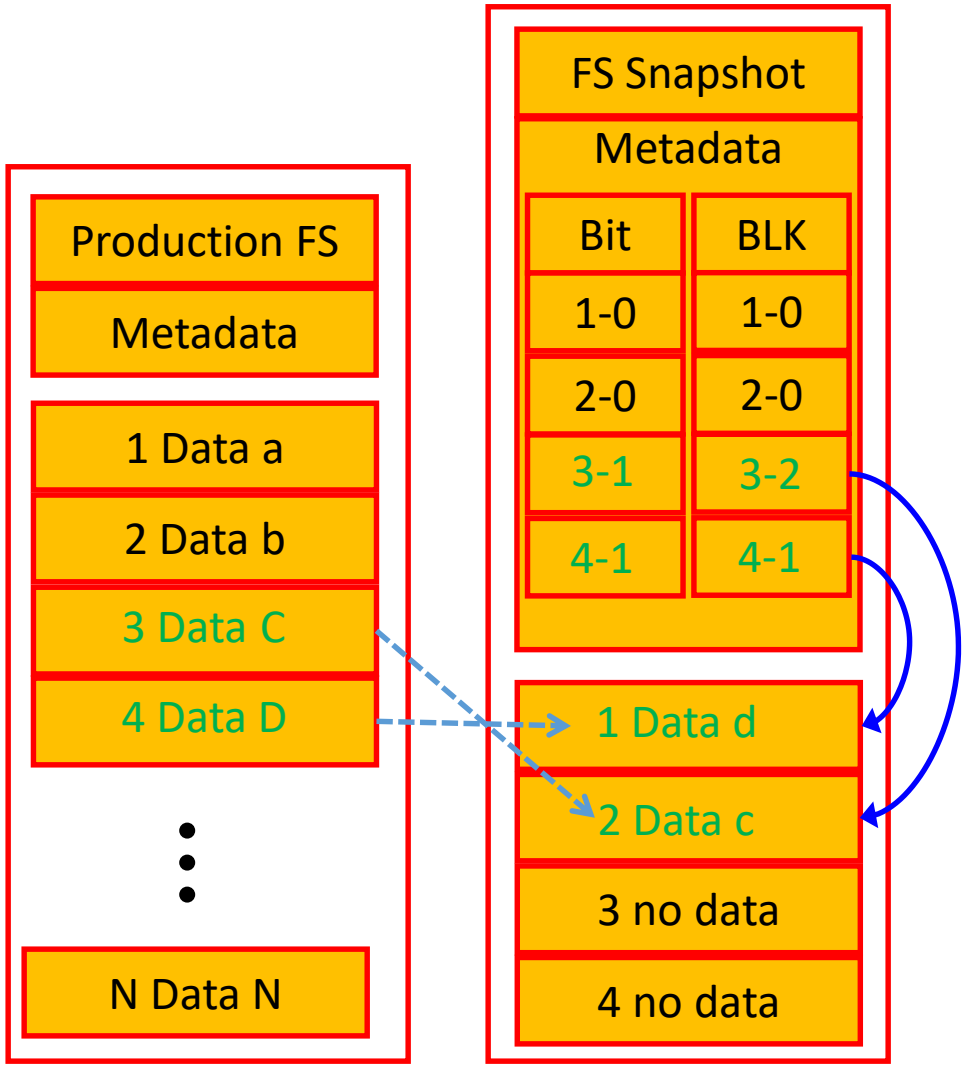
Host-based Replication: LVM-based Mirroring

- LVM: Logical Volume Manager



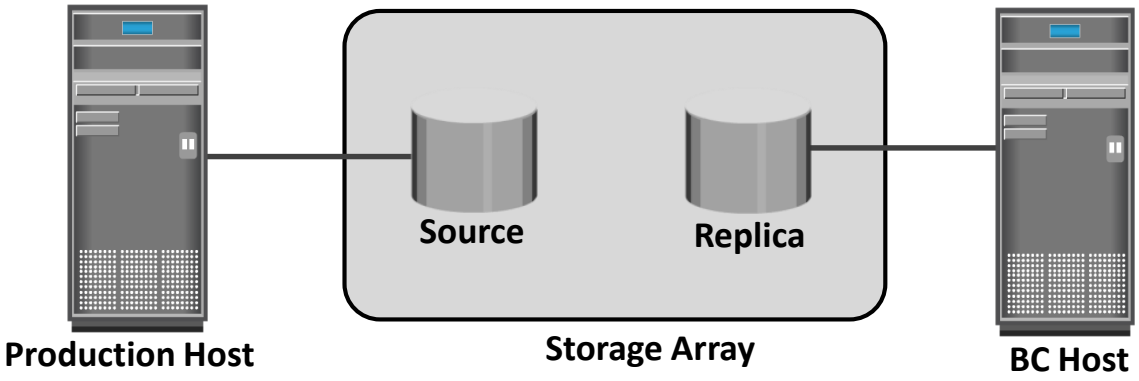
Host-based Replication: File System Snapshot

- Pointer-based replication
- Uses Copy on First Write (CoFW) principle
- Uses bitmap and block map
- Requires a fraction of the space used by the production FS

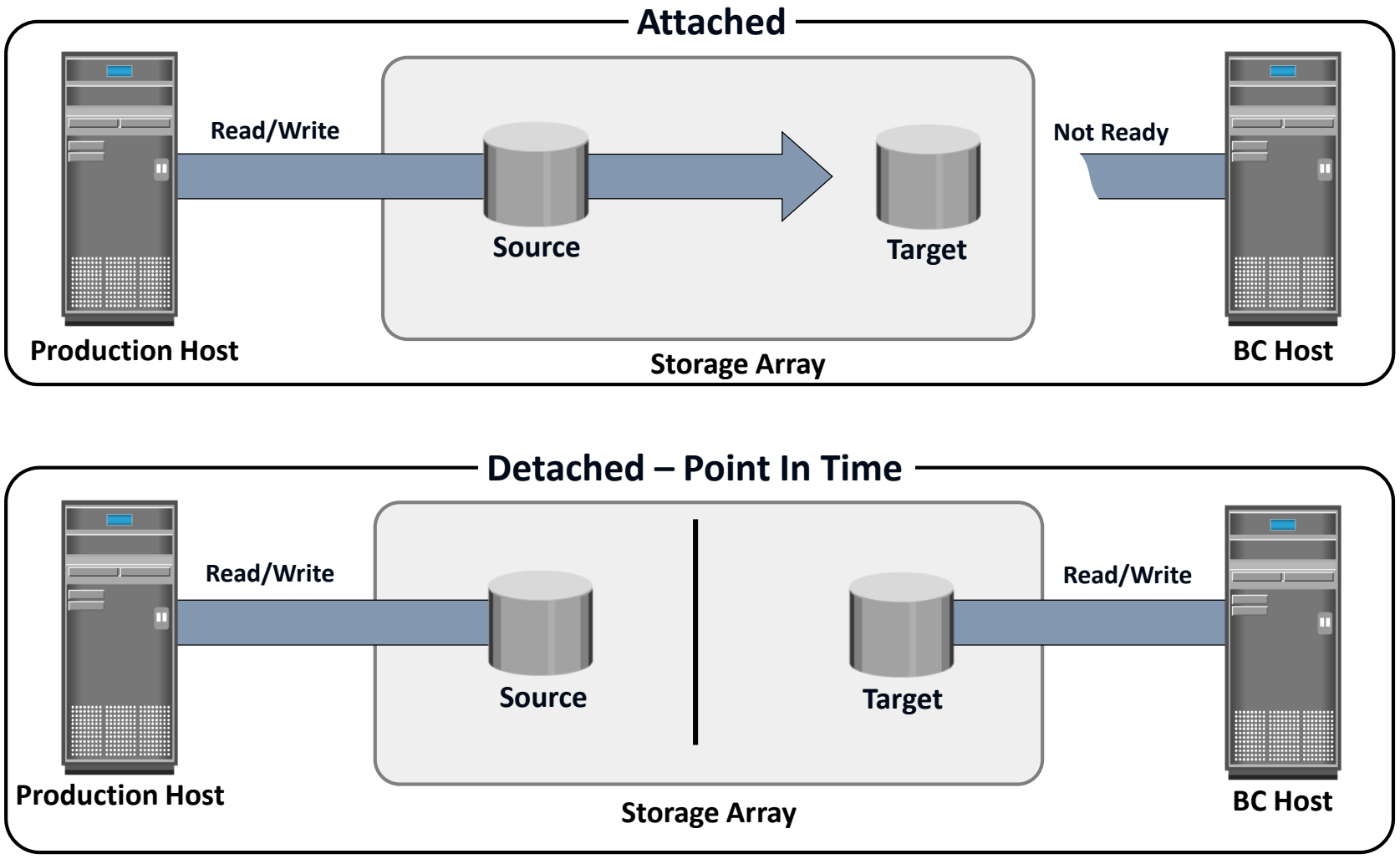


Storage Array-based Local Replication

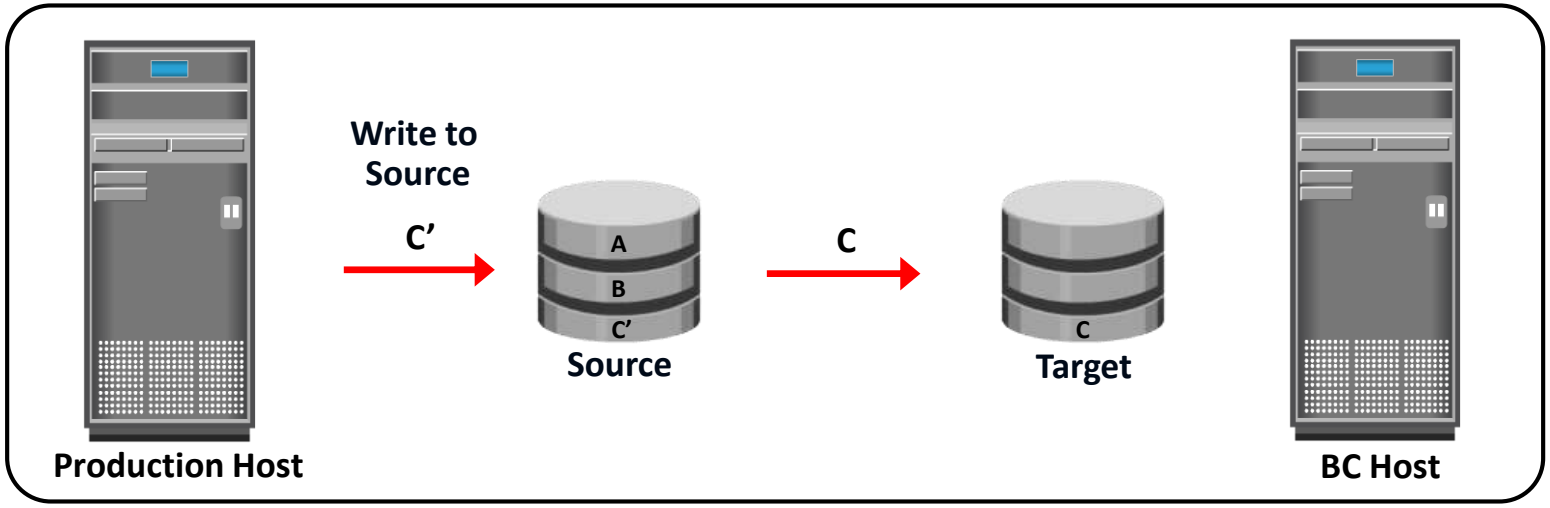
- Replication performed by the array operating environment
- Source and replica are on the same array
- Types of array-based replication
  - Full-volume mirroring
  - Pointer-based full-volume replication
  - Pointer-based virtual replication



Full-Volume Mirroring

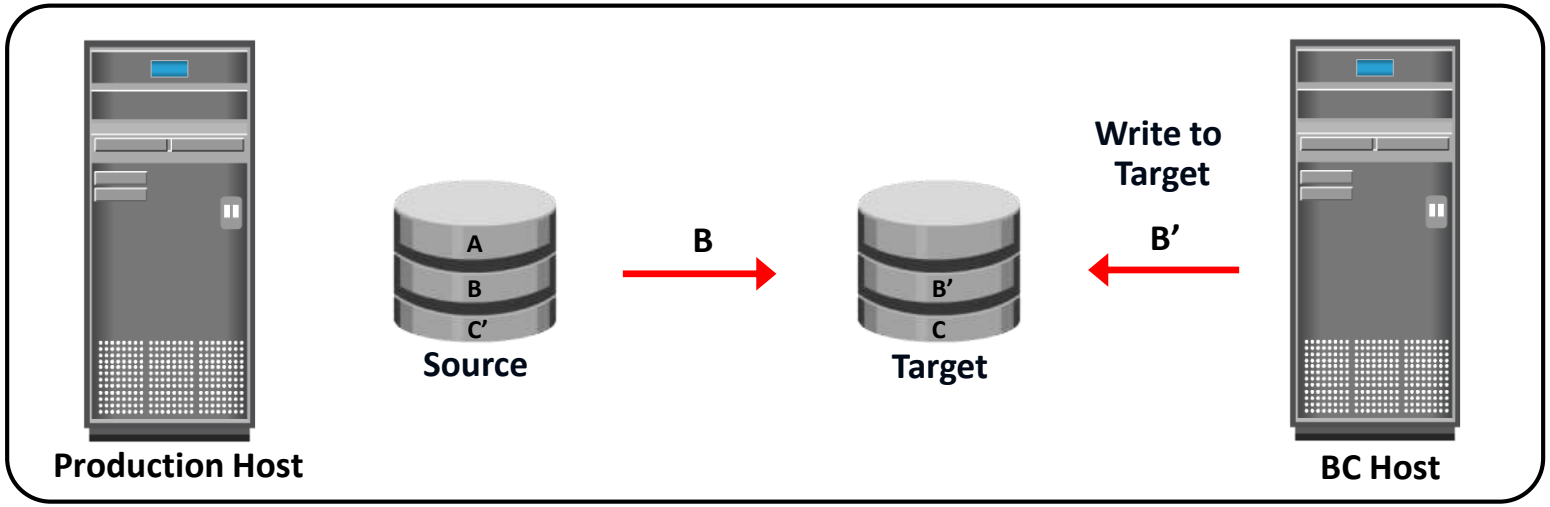


Copy on First Access: Write to the Source



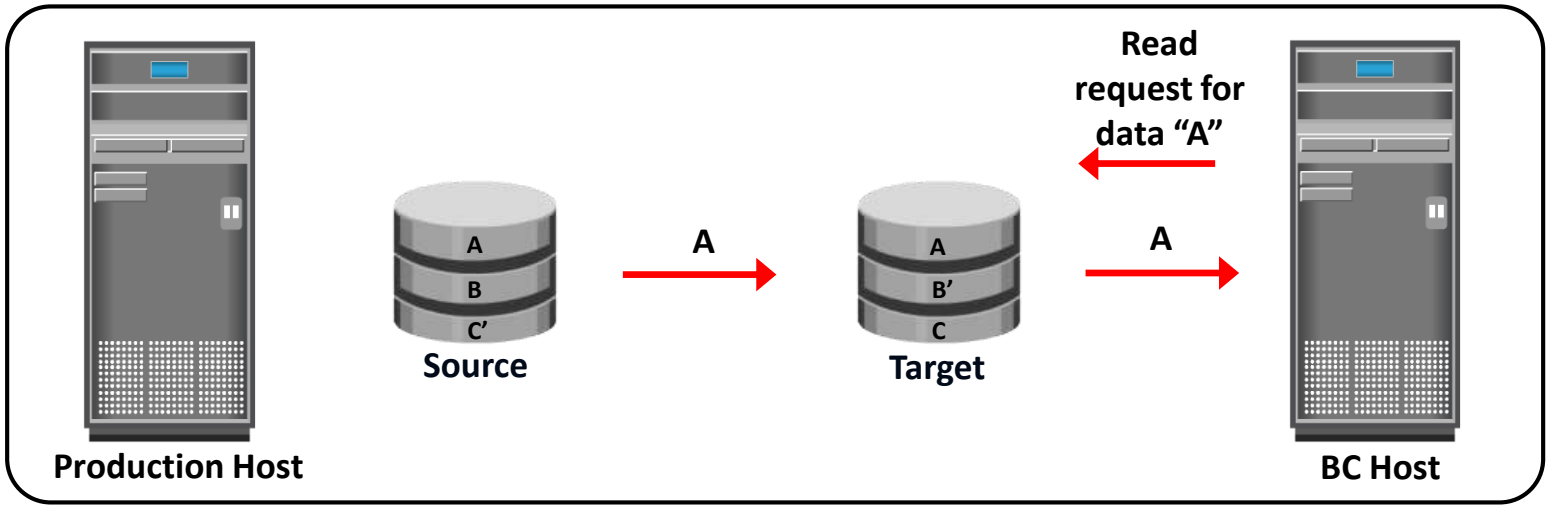
- When a write is issued to the source for the first time after replication session activation:
  - Original data at that address is copied to the target
  - Then the new data is updated on the source
  - This ensures that original data at the point-in-time of activation is preserved on the target

Copy on First Access: Write to the Target



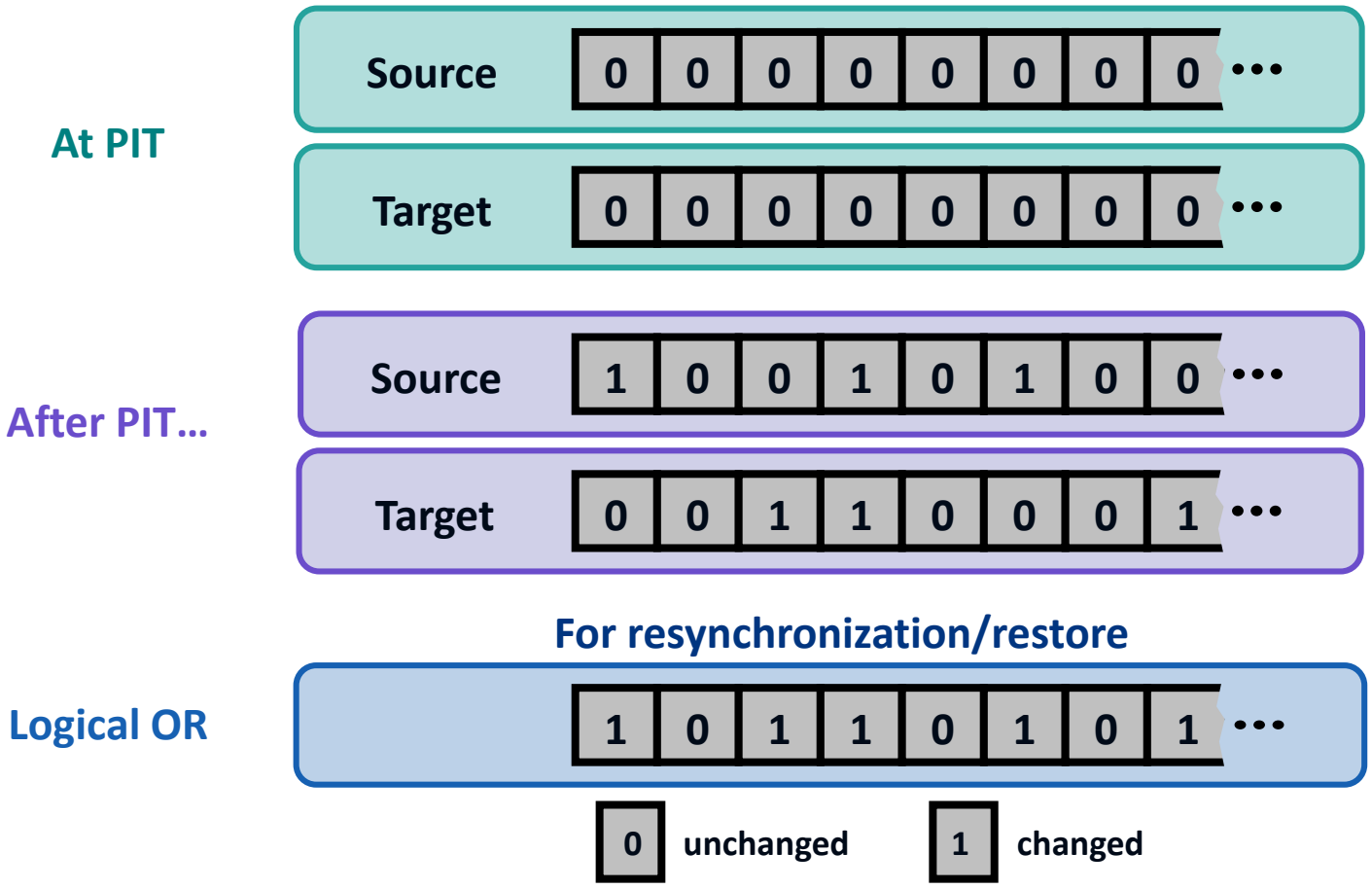
- When a write is issued to the target for the first time after replication session activation:
  - The original data is copied from the source to the target
  - Then the new data is updated on the target

Copy on First Access: Read from Target



- When a read is issued to the target for the first time after replication session activation:
  - The original data is copied from the source to the target and is made available to the BC host

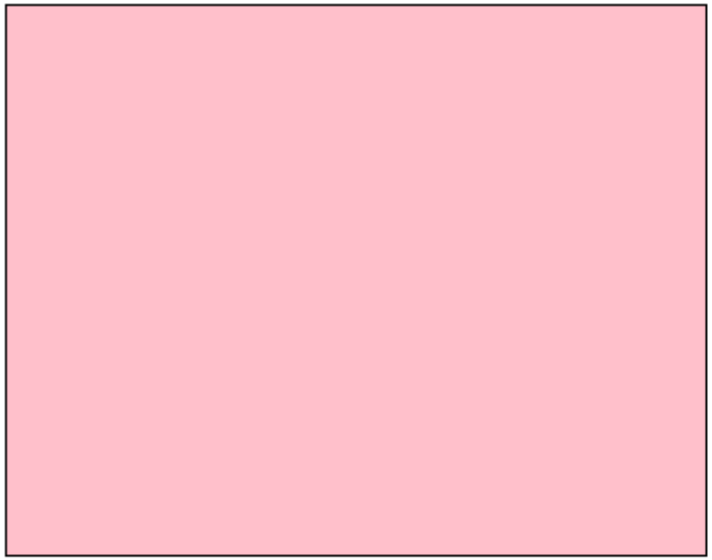
Tracking Changes to Source and Target



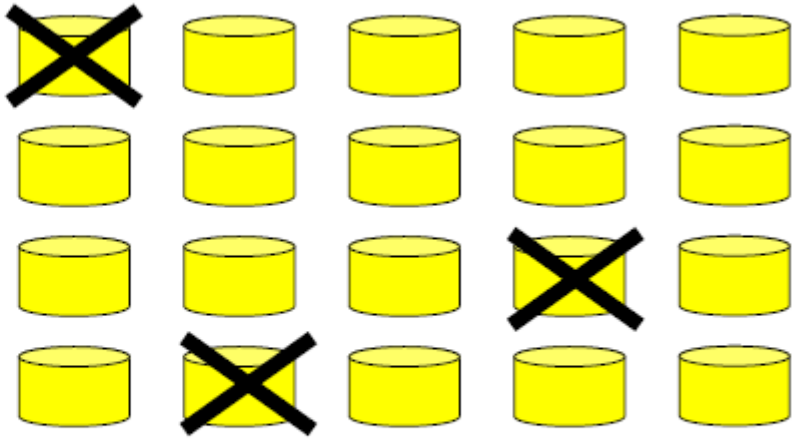
2 Introduction to Erasure Codes

Erasure Coding Basis (1)

- You've got some data



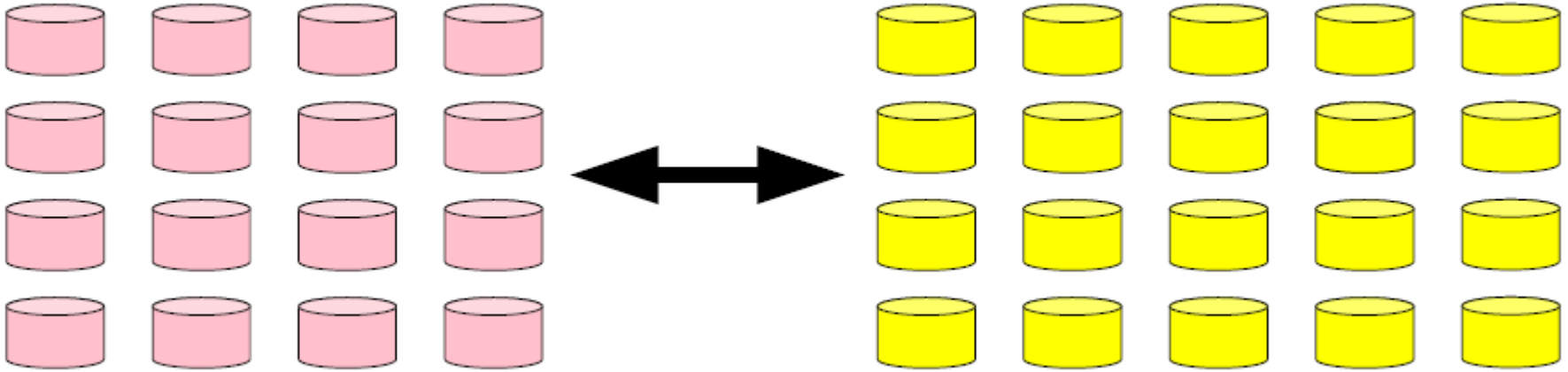
- And a collection of storage nodes.



- And you want to store the data on the storage nodes so that you can get the data back, even when the nodes fail..

Erasure Coding Basis (2)

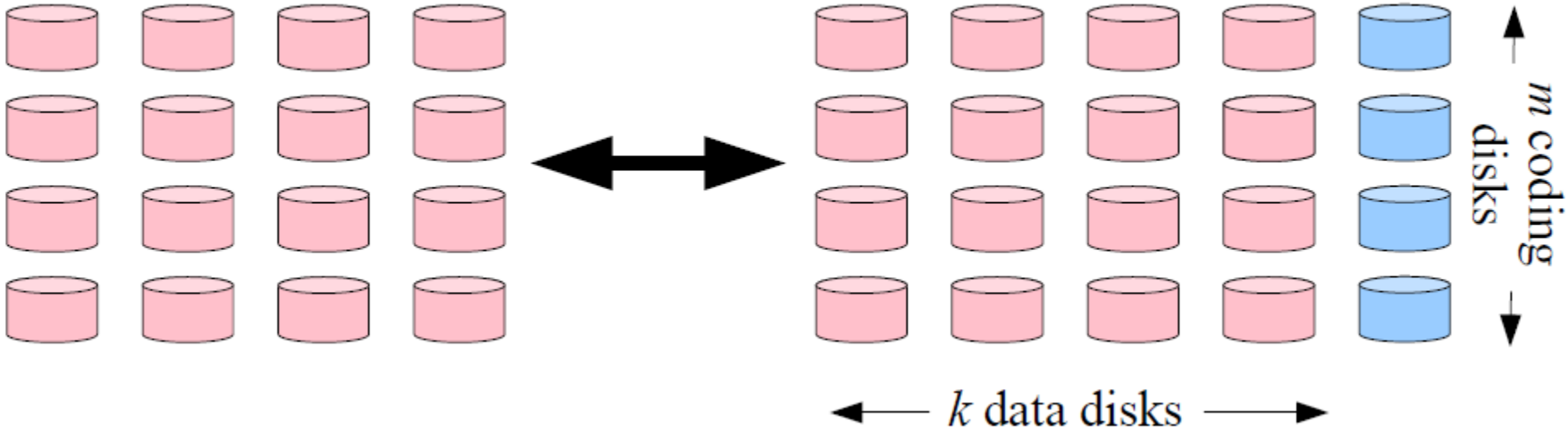
- More concrete: You have *k* disks worth of data
- And *n* total disks.



- The erasure code tells you how to create *n* disks worth of data+coding so that when disks fail, you can still get the data

Erasure Coding Basis (3)

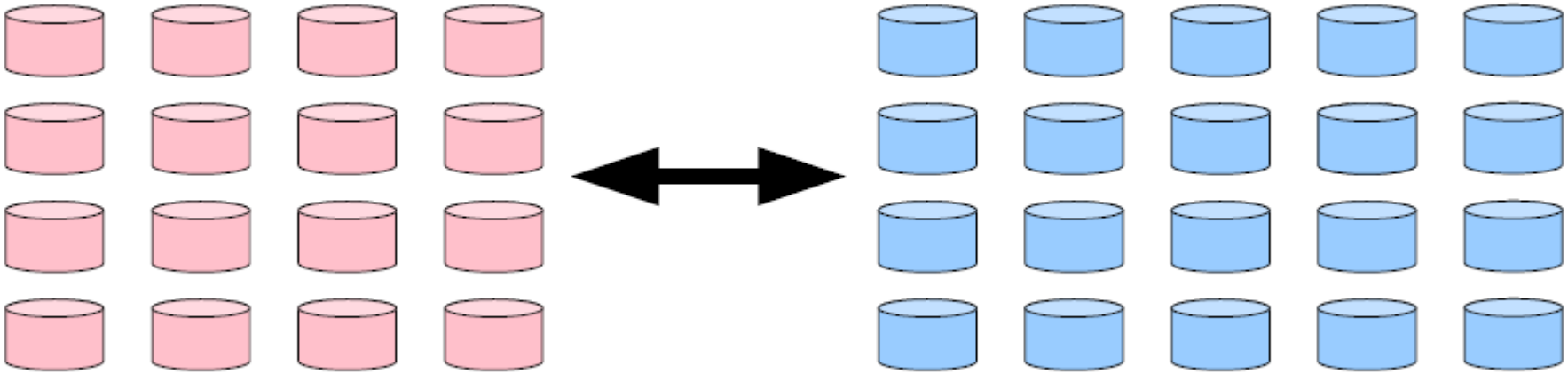
- You have *k* disks worth of data
- And *n* total disks.
- *n = k + m*



- A systematic erasure code stores the data in the clear on *k* of the *n* disks. There are *k* data disks, and *m* coding or "parity" disks. → **Horizontal Code**

Erasure Coding Basis (4)

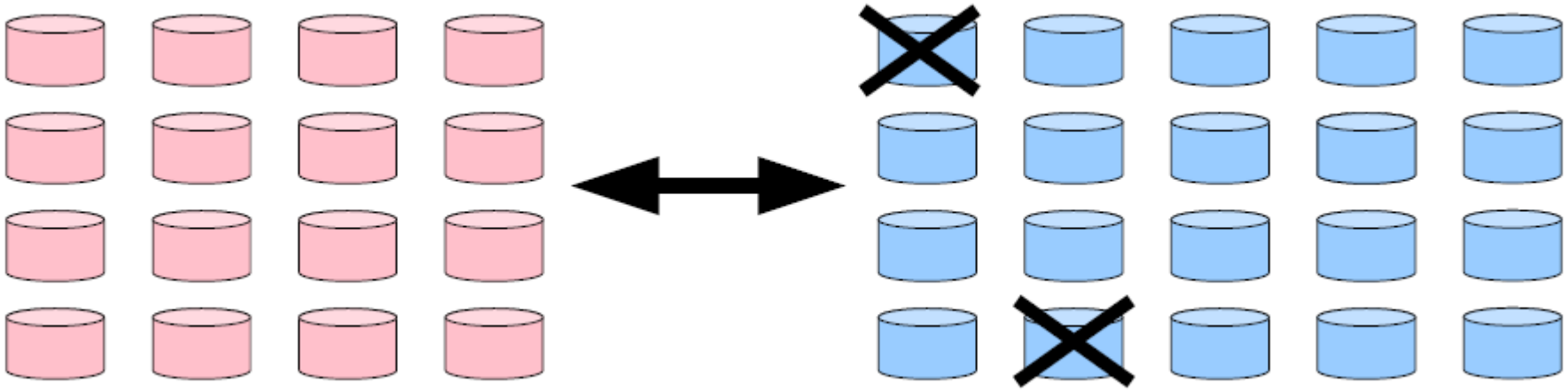
- You have *k* disks worth of data
- And *n* total disks.
- $n = k + m$



- A non-systematic erasure code stores only coding information, but we still use *k*, *m*, and *n* to describe the code. → **Vertical Code**

Erasure Coding Basis (5)

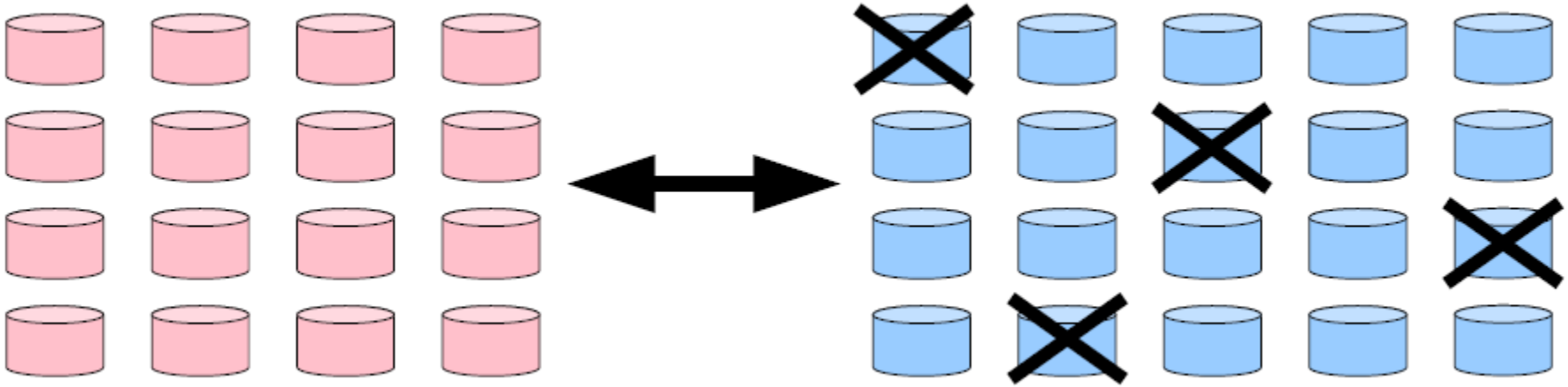
- You have *k* disks worth of data
- And *n* total disks.
- $n = k + m$



- When disks fail, their contents become unusable, and the storage system detects this. This failure mode is called an **erasure**.

Erasure Coding Basis (6)

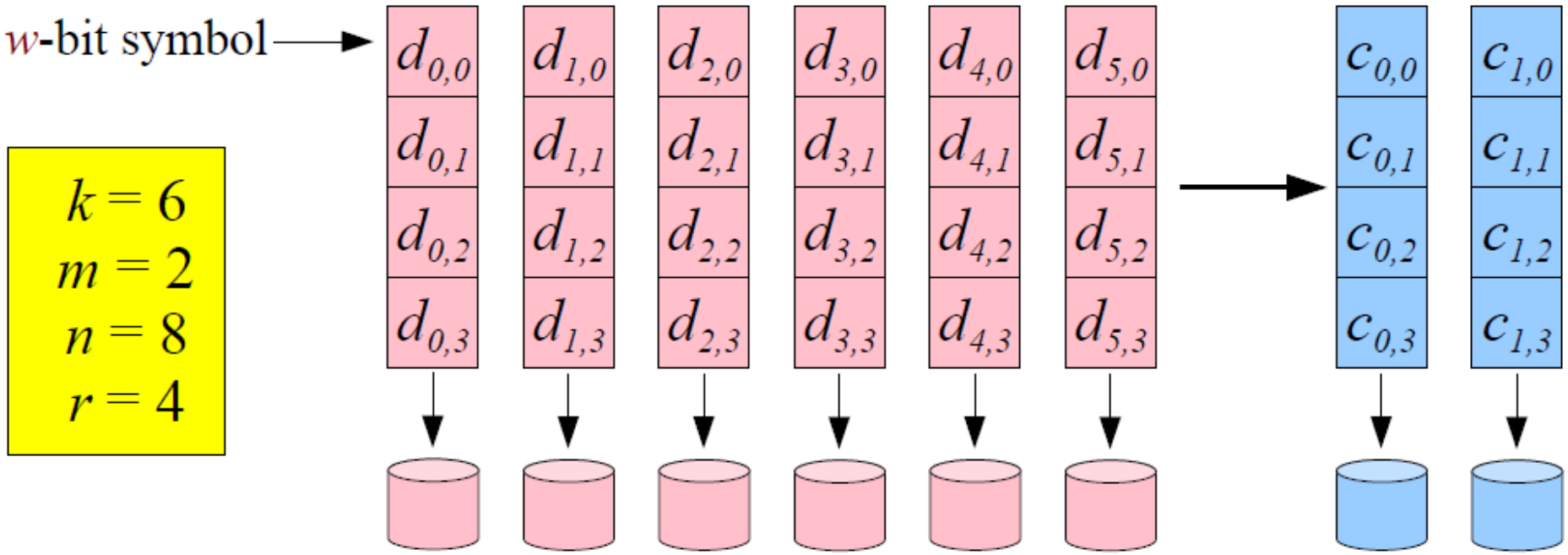
- You have *k* disks worth of data
- And *n* total disks.
- $n = k + m$



- An MDS ("Maximum Distance Separable") code can reconstruct the data from any *m* failures. → Optimal
- Can reconstruct any *f* failures ($f < m$) → non-MDS code

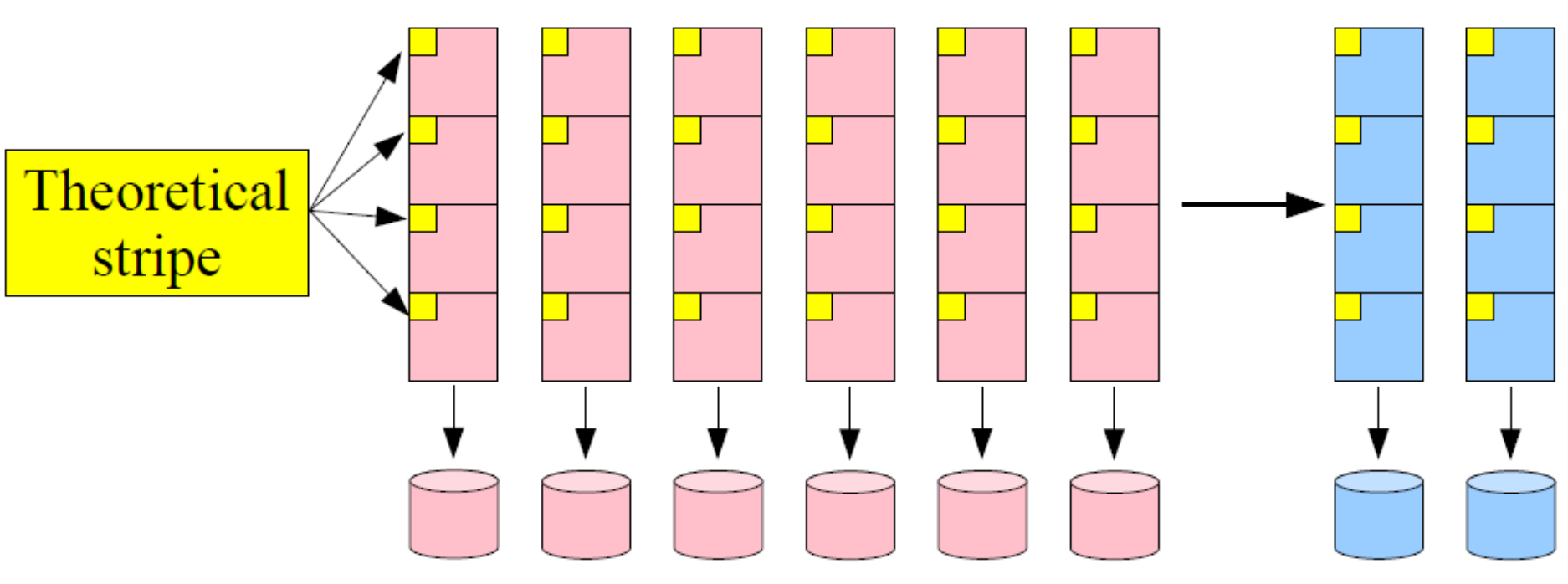
Two Views of a Stripe (1)

- The Theoretical View:
  - The minimum collection of bits that encode and decode together.
  - *r* rows of *w*-bit symbols from each of *n* disks:



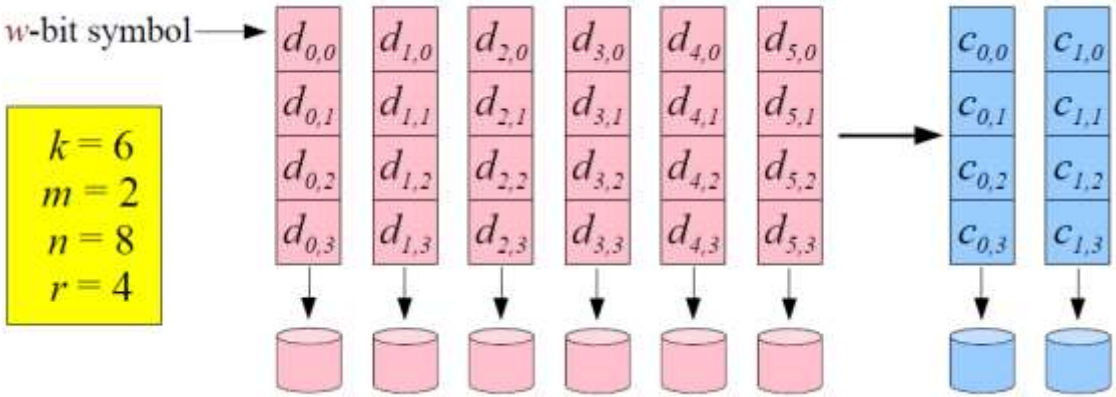
Two Views of a Stripe (2)

- The Systems View:
  - The minimum partition of the system that encodes and decodes together.
  - Groups together theoretical stripes for performance.

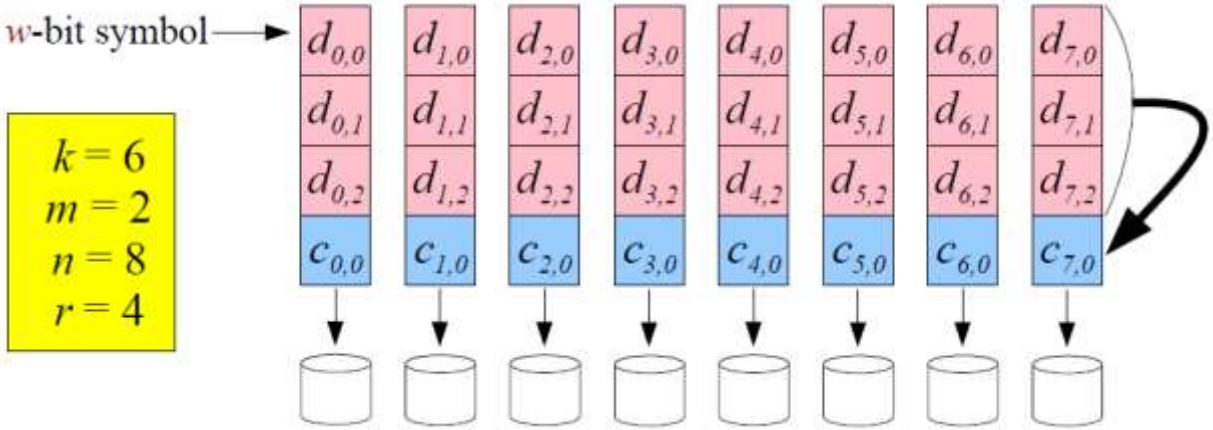


Horizontal & Vertical Codes

- Horizontal Code

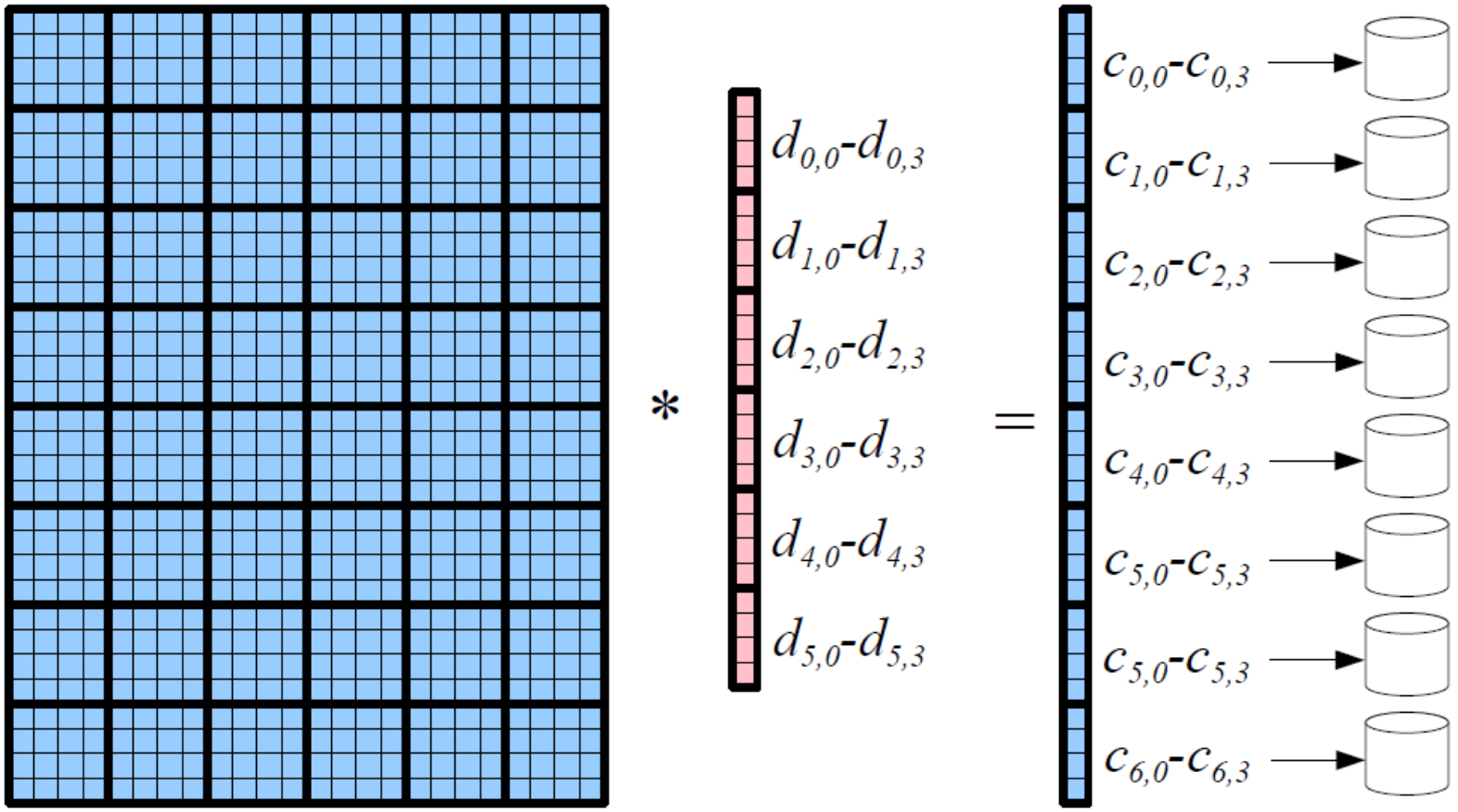


- Vertical Code



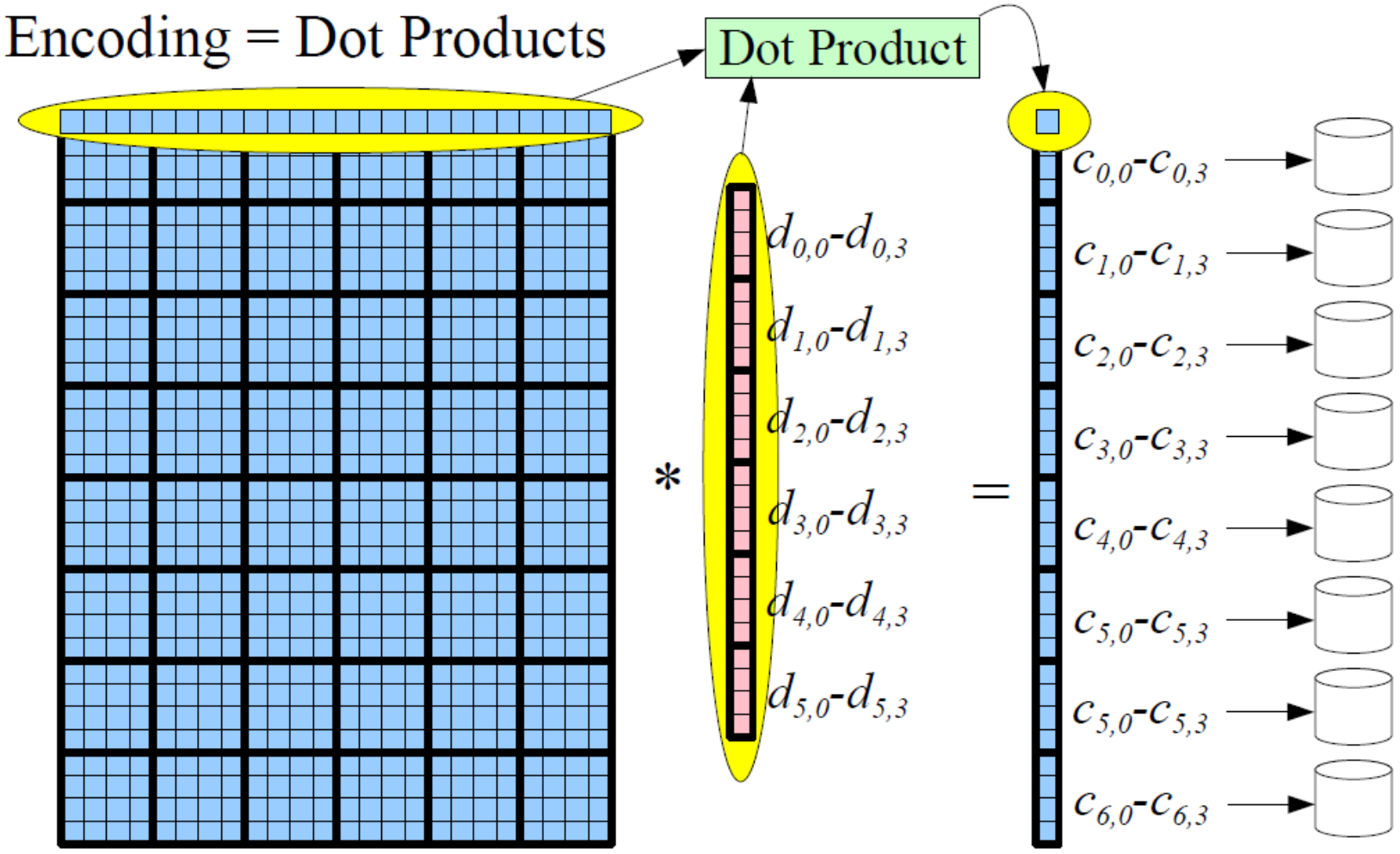
Expressing Code with Generator Matrix (1)

Non-systematic code with k=6, n=8, r=4.



Generator Matrix (G^T): nr X kr w-bit symbols

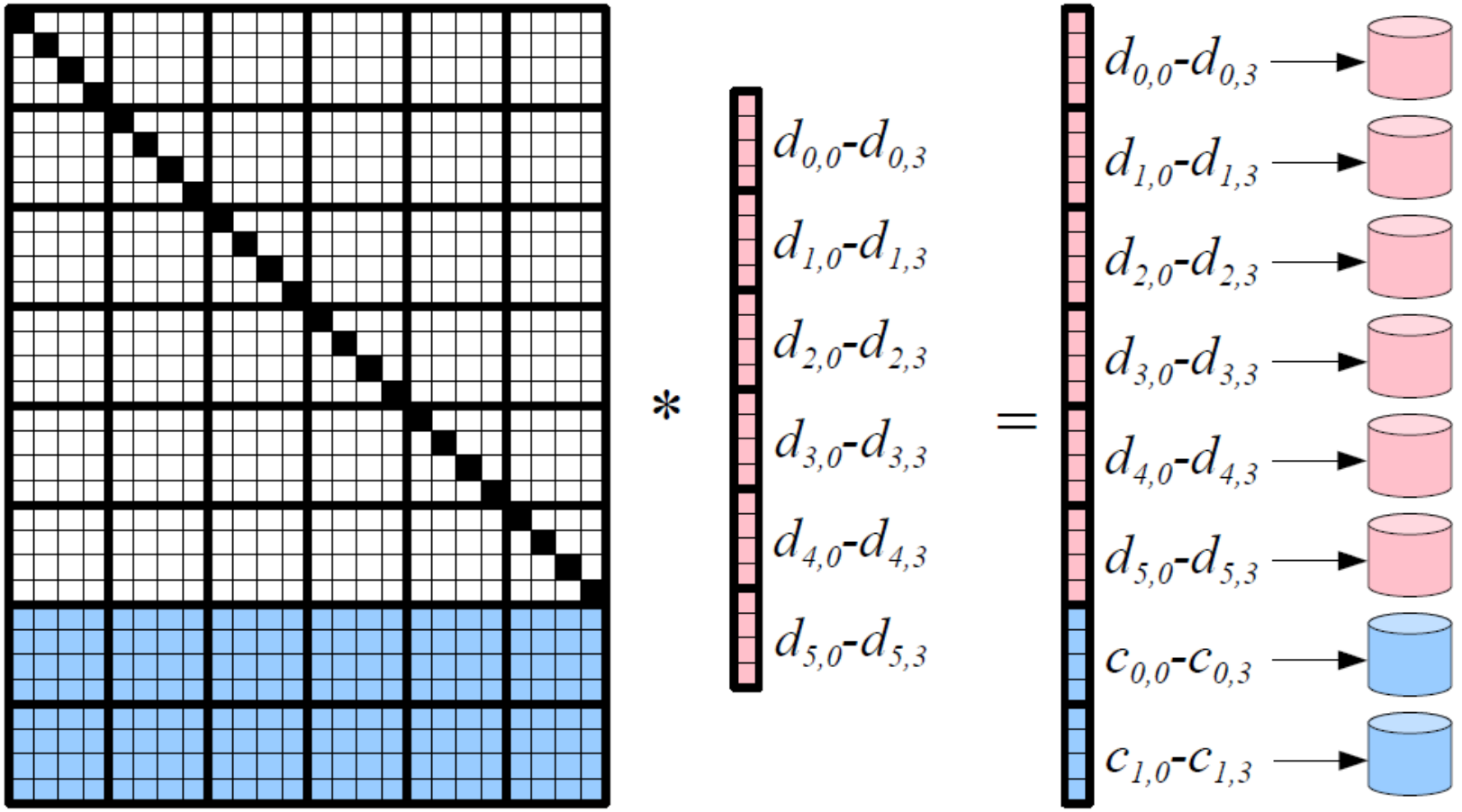
Expressing Code with Generator Matrix (2)



Generator Matrix (G^T): nr X kr w-bit symbols

Expressing Code with Generator Matrix (3)

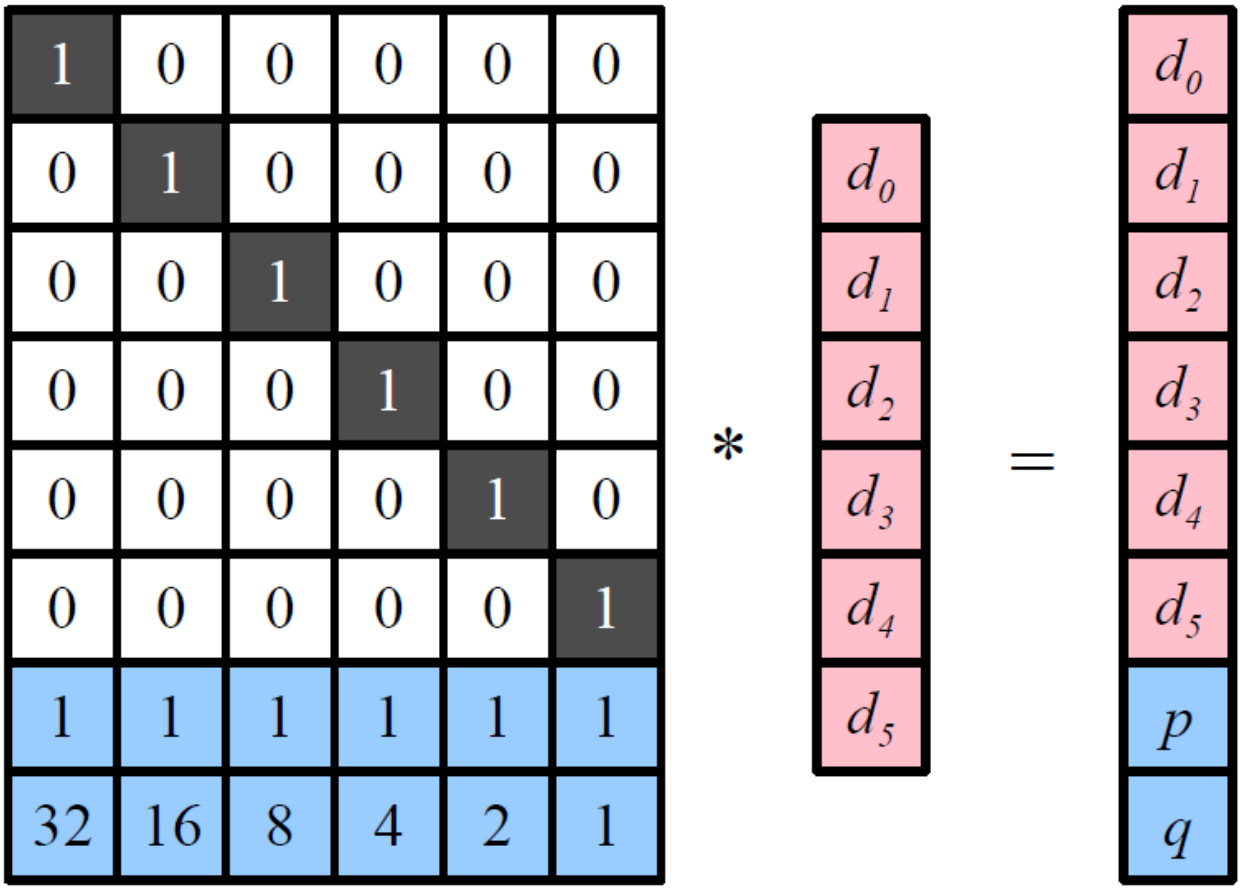
Systematic code with k=6, n=8, r=4.



Generator Matrix (G^T): nr X kr w-bit symbols

Encoding— Linux RAID-6 (1)

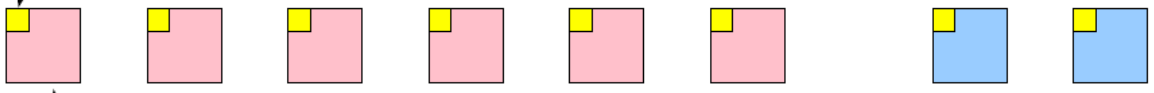
Systematic code with k=6, n=8, r=1, w=8 (Linux RAID-6)



Generator Matrix (G^T)

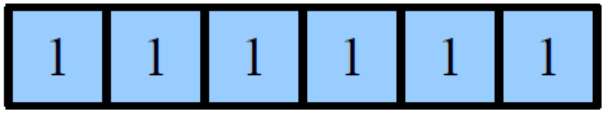
Encoding— Linux RAID-6 (2)

One byte (an 8-bit word, since $w = 8$)

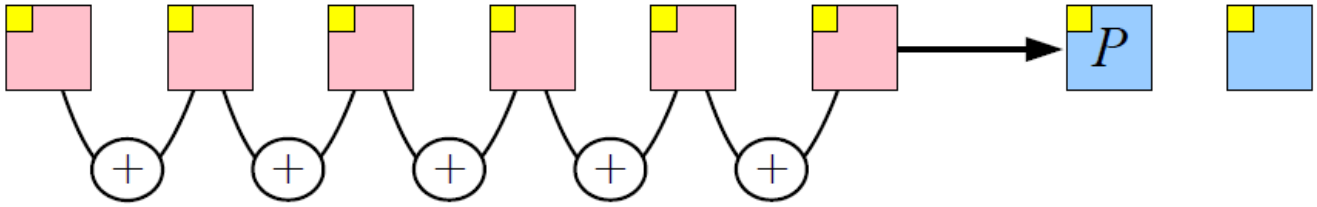


A bigger block (e.g. 1K or 1 M)

And calculate the first coding block (P):



P's row of the generator matrix



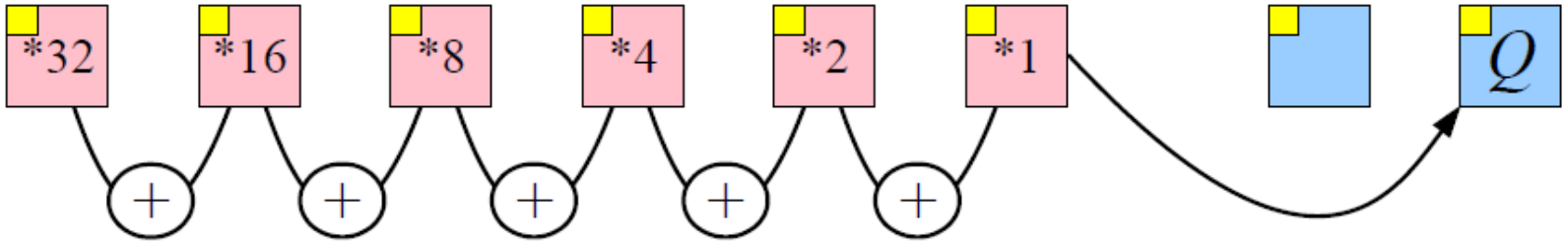
Addition = XOR. Can do that 64/128 bits at a time.

Encoding— Linux RAID-6 (3)

And calculate the second coding block (Q):

32	16	8	4	2	1

Q's row of the generator matrix



Addition still equals XOR.

Multiplication = "Galois Field Multiplication" $GF(2^w)$.

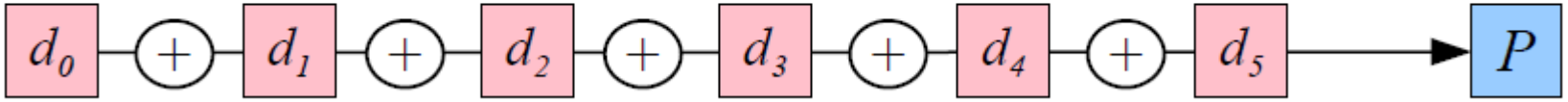
Complex, but still pretty fast (more on that later)

Accelerate Encoding— Linux RAID-6

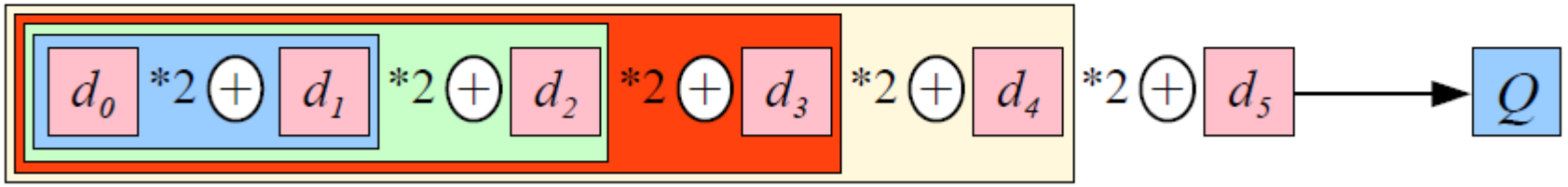
1	1	1	1	1	1
32	16	8	4	2	1

Why is encoding fast?

First, the *P* drive is simply parity.

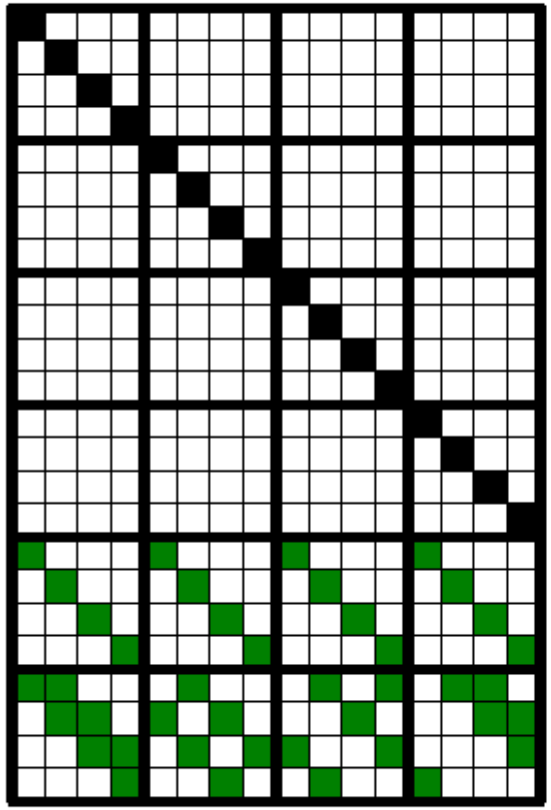


Second, the *Q* drive leverages fast multiplication by two.



Encoding— RDP (1)

Systematic code with k=4, n=6, r=4, w=1 (RDP RAID-6)

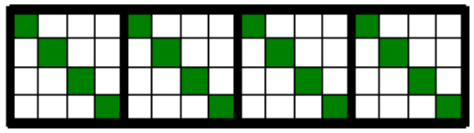


$$*\begin{bmatrix} d_{0,0}\text{-}d_{0,3} \\ d_{1,0}\text{-}d_{1,3} \\ d_{2,0}\text{-}d_{2,3} \\ d_{3,0}\text{-}d_{3,3} \end{bmatrix} = \begin{bmatrix} d_{0,0}\text{-}d_{0,3} \\ d_{1,0}\text{-}d_{1,3} \\ d_{2,0}\text{-}d_{2,3} \\ d_{3,0}\text{-}d_{3,3} \\ p_0\text{-}p_3 \\ q_0\text{-}q_3 \end{bmatrix}$$

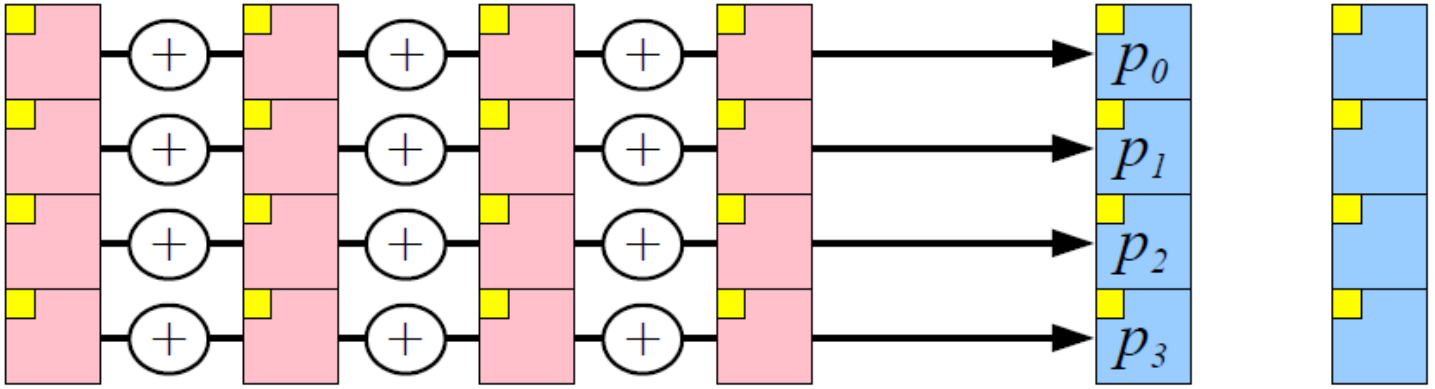
Generator Matrix (G^T):

Encoding— RDP (2)

Calculating the first coding block (P):



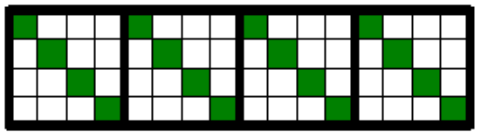
$p_0 - p_3$'s rows of the generator matrix



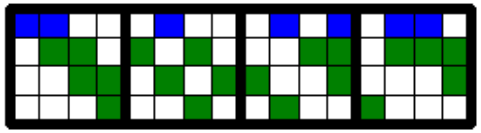
Again, we get to XOR blocks of bytes instead of single bits.

Encoding— RDP (3)

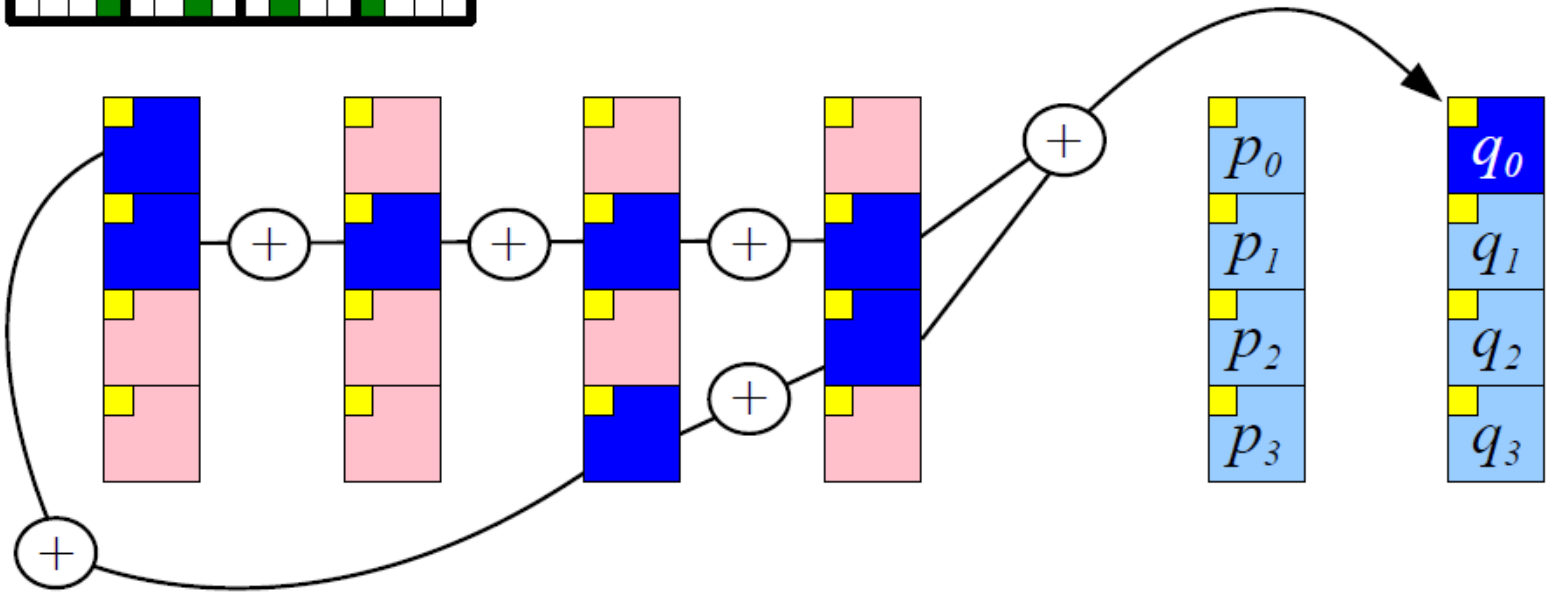
Calculating the second coding block (Q): - Just q_0 for now.



$p_0 - p_3$'s rows of the generator matrix

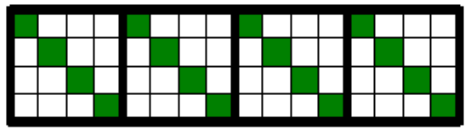


$q_0 - q_3$'s rows of the generator matrix

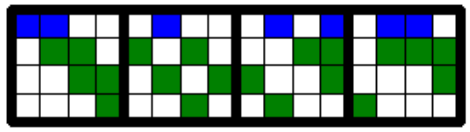


Encoding— RDP (4)

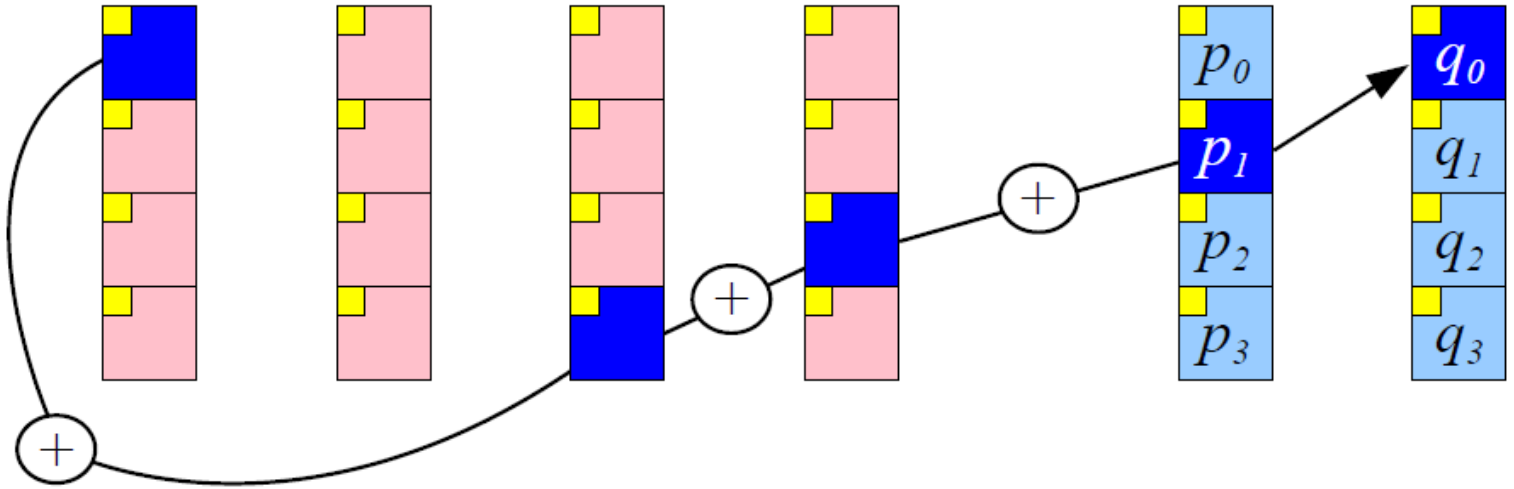
Calculating the second coding block (Q): - Just q_0 for now.



$p_0 - p_3$'s rows of the generator matrix

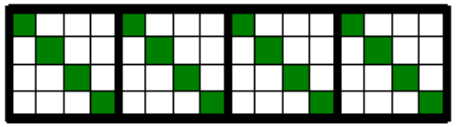


$q_0 - q_3$'s rows of the generator matrix

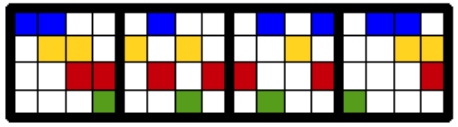


Encoding— RDP (5)

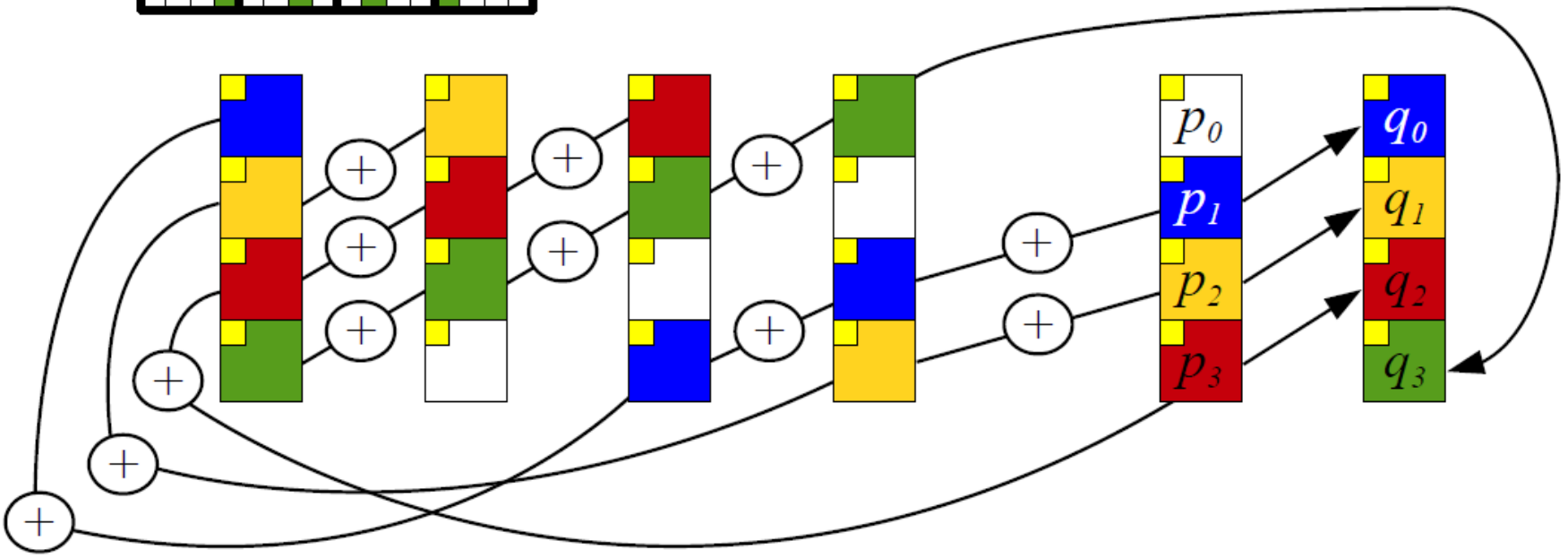
Calculating the second coding block (Q):



$p_0 - p_3$'s rows of the generator matrix

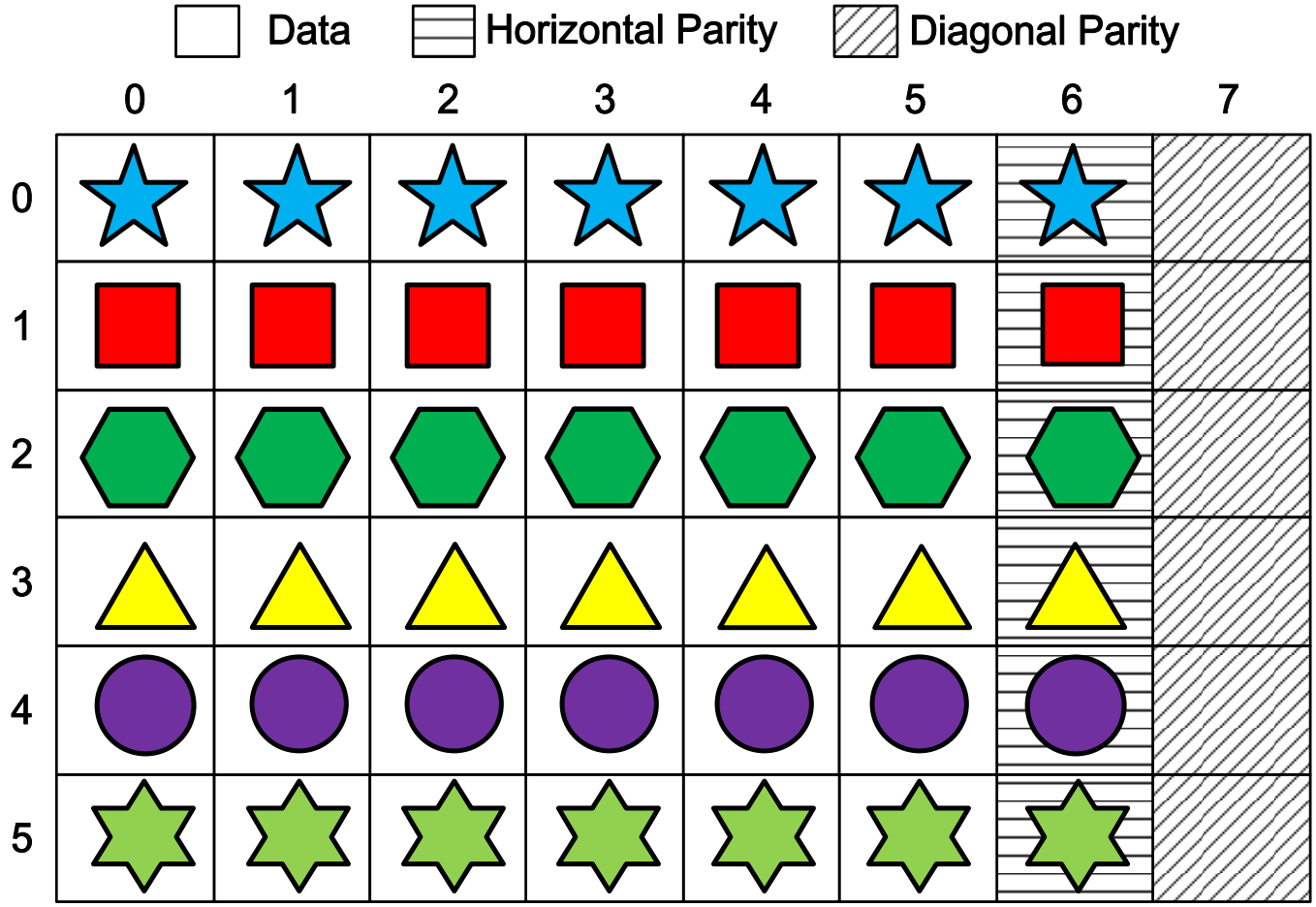


$q_0 - q_3$'s rows of the generator matrix



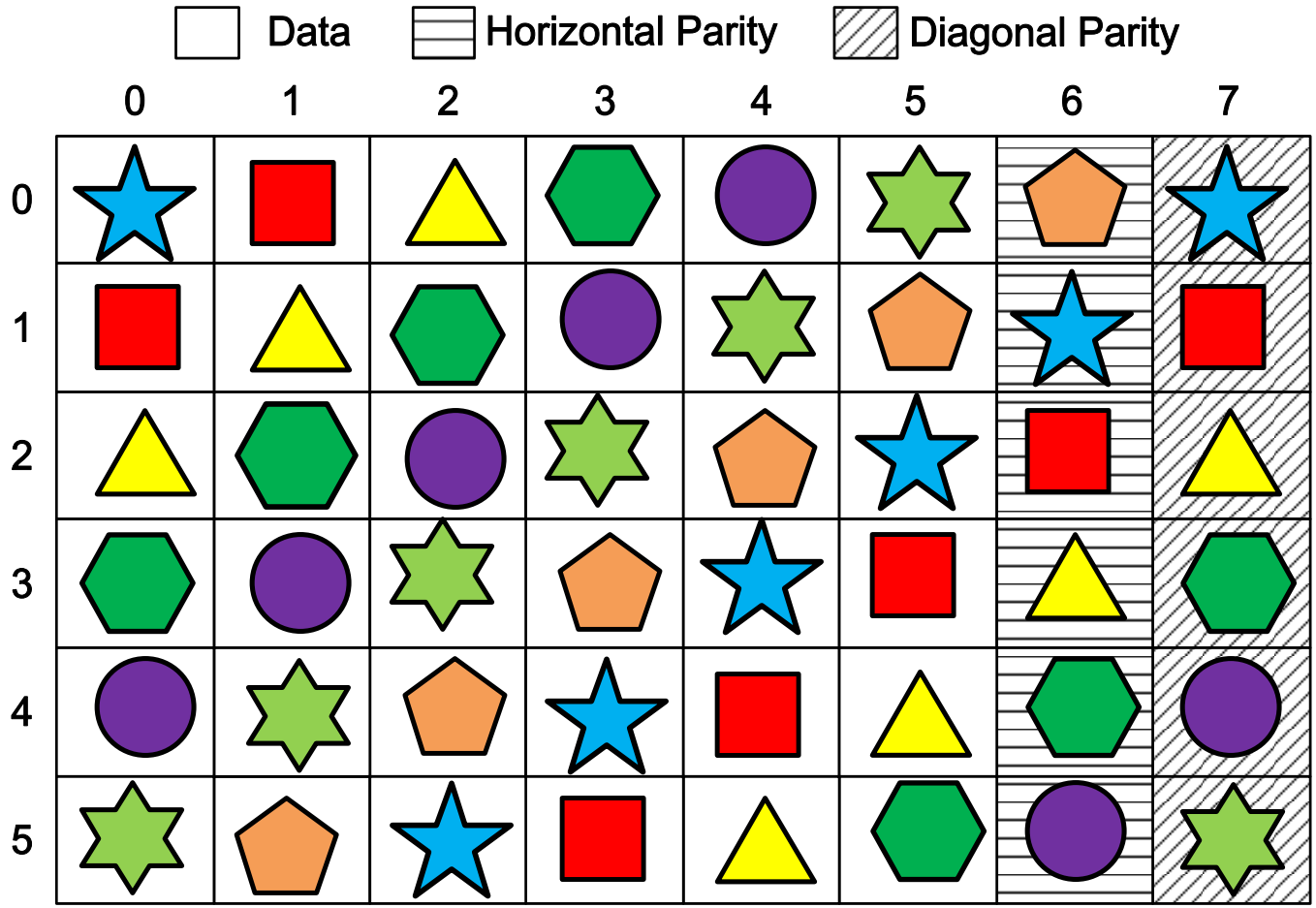
Encoding— RDP (6)

- Horizontal parity layout (p=7, n=8)



Encoding— RDP (7)

- Diagonal parity layout (p=7, n=8)



Arithmetic for Erasure Codes

- When *w = 1*: XOR's only.
- Otherwise, Galois Field Arithmetic *GF(2w)*
  - *w* is 2, 4, 8, 16, 32, 64, 128 so that words fit evenly into computer words.
  - Addition is equal to XOR.

    Nice because addition equals subtraction.
  - Multiplication is more complicated:

    Gets more expensive as *w* grows.

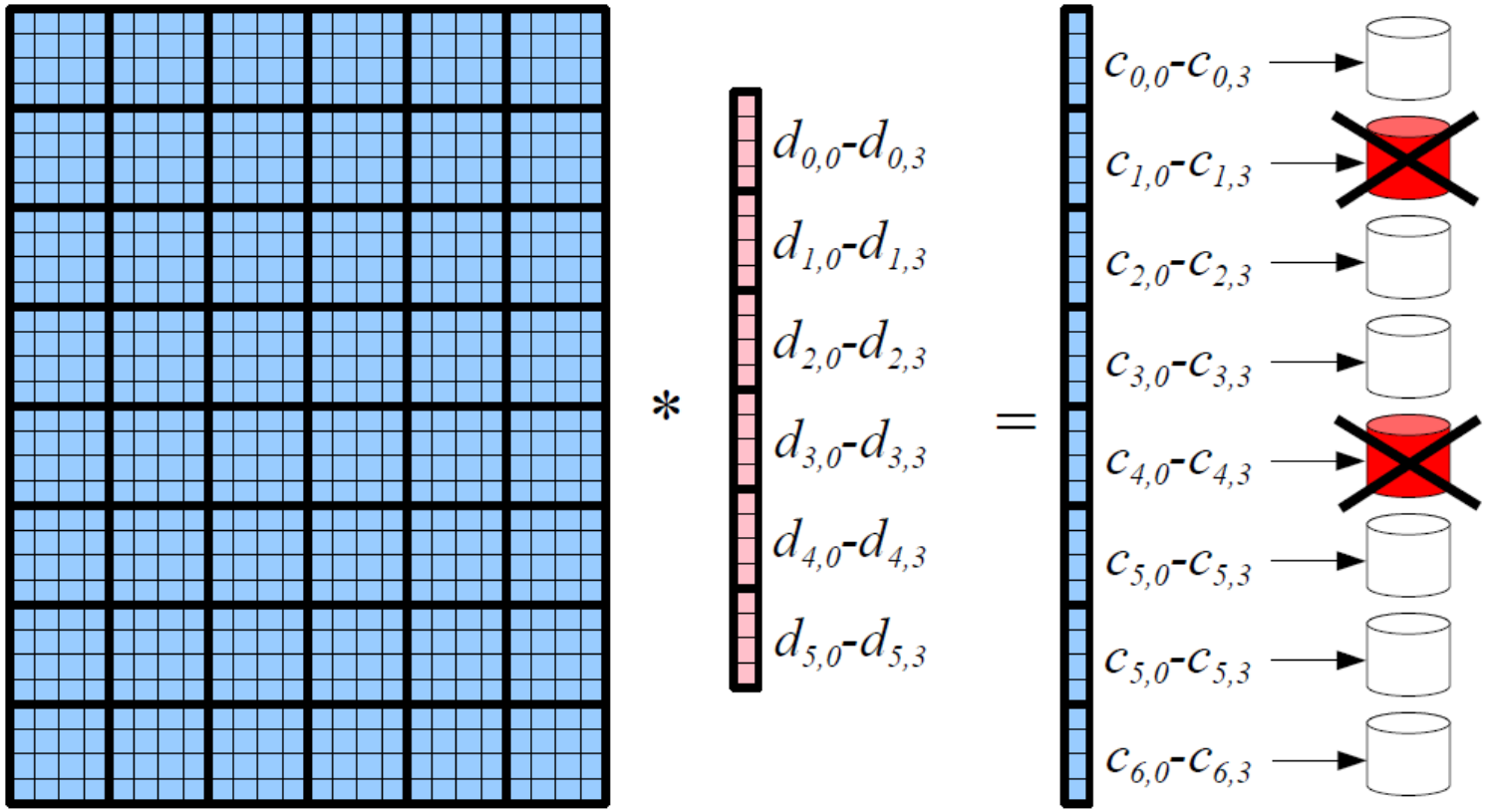
    Buffer-constant different from *a * b*.

    Buffer * 2 can be done really fast.

    Open source library support.

Decoding with Generator Matrices (1)

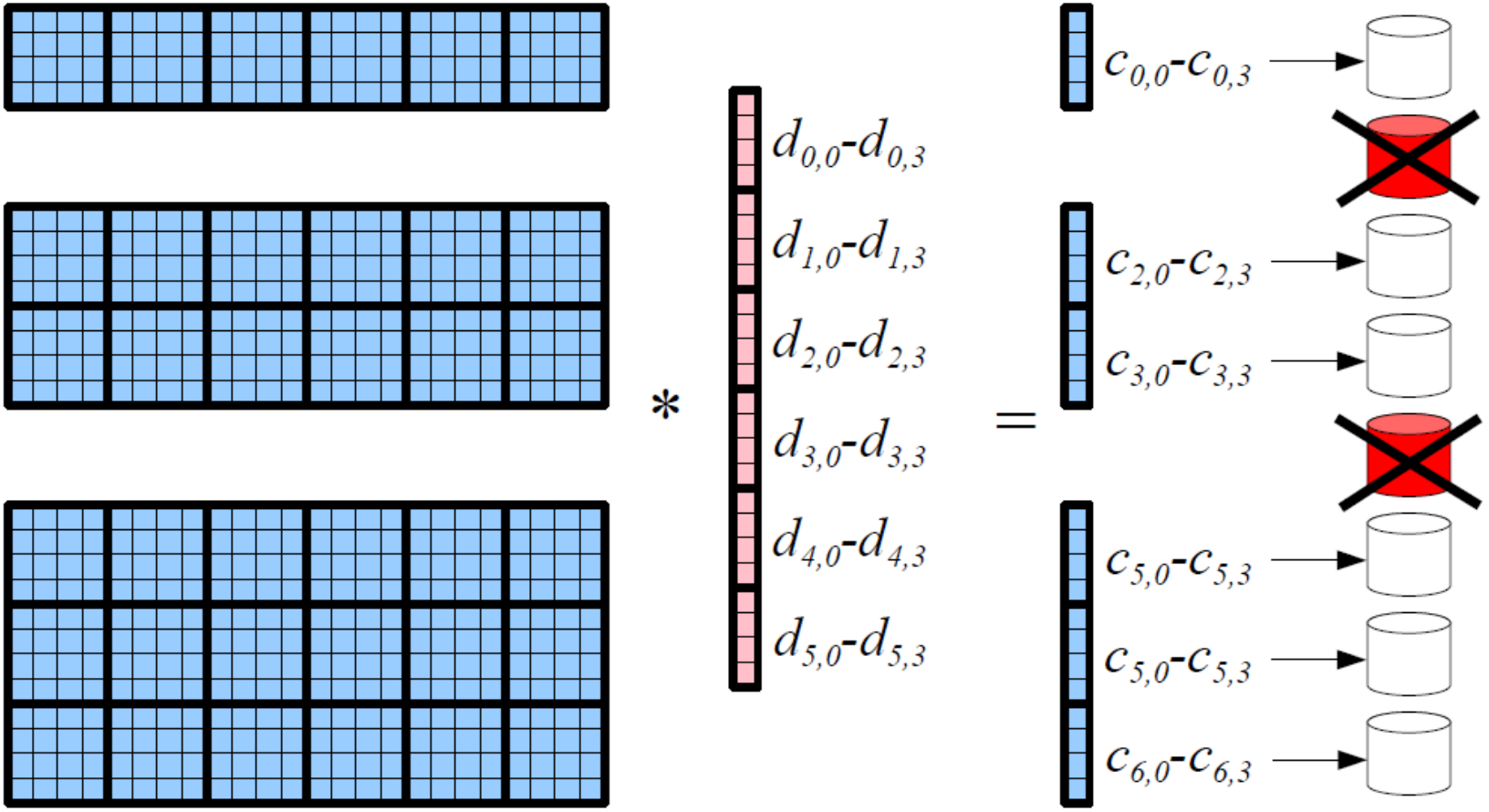
(Non-systematic code with k=6, n=8, r=4.)



Generator Matrix (G^T)

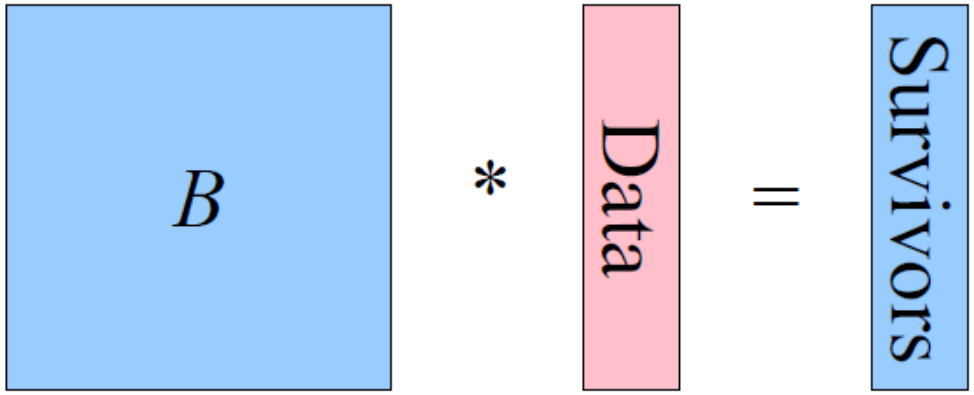
Decoding with Generator Matrices (2)

Step #1: Delete rows that correspond to failed disks.



Decoding with Generator Matrices (3)

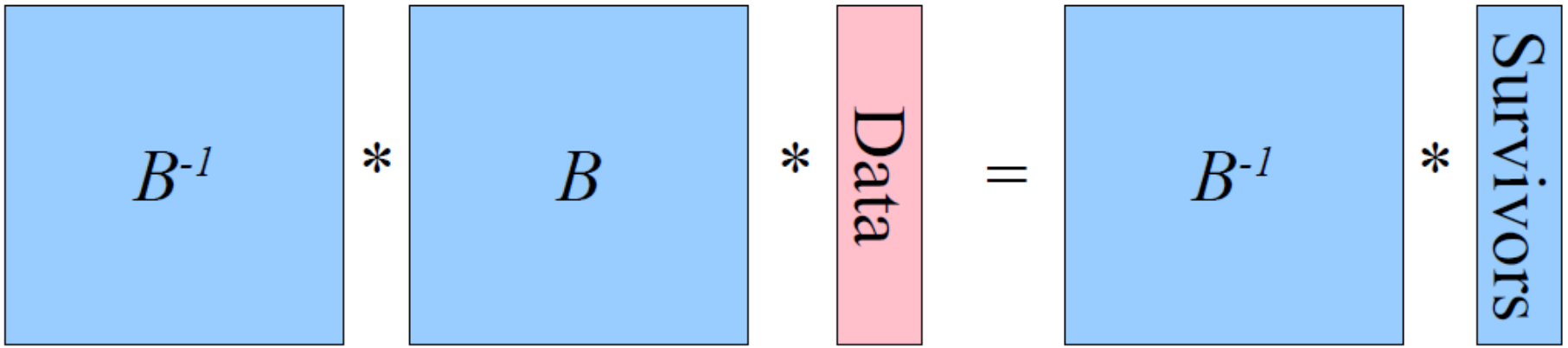
Step #2: Let's rewrite this equation so that looks like math.



Generator Matrix (G^T)
with deleted rows

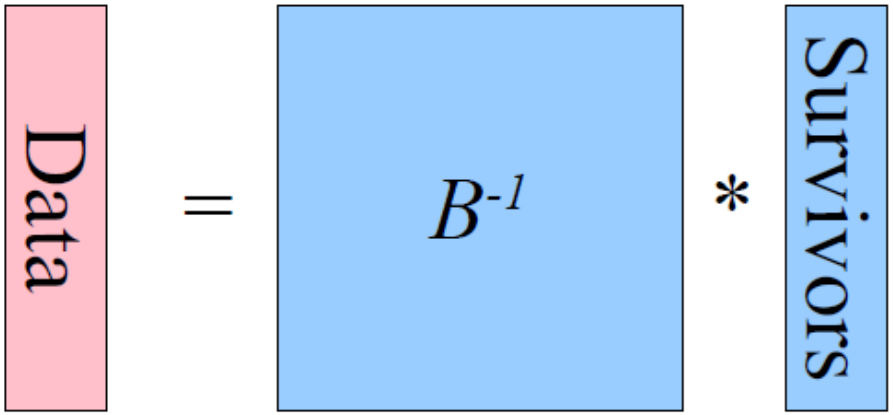
Decoding with Generator Matrices (4)

Step #3: Invert B and multiply it by both sides of the equation:



Decoding with Generator Matrices (5)

Voila – you now know how to decode the data!



Erasure Codes — Reed Solomon (1)

- Given in **1960**.
- MDS Erasure codes for any *n* and *k*.
  - That means any *m = (n-k)* failures can be tolerated without data loss.
- *r* = 1

  (Theoretical): One word per disk per stripe.
- *w* constrained so that $n \leq 2w$.
- Systematic and non-systematic forms.

Erasure Codes —Reed Solomon (2)
Systematic RS -- Cauchy generator matrix

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
$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	71	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$

Example with
$k = 6$ and $n = 9$.

- Create two sets X and Y.
- X has m elements: x_0 to x_{m-1}.
- Y has k elements: y_0 to y_{k-1}.
- Elements in $(X \cup Y)$ are distinct.
- $a_{i,j} = 1/(x_i + y_j)$ in $GF(2^w)$.

$X = \{ 1, 2, 3 \}$

$Y = \{ 4, 5, 6, 7, 8, 9 \}$

$a_{1,2} = 1/(2+6)$ in $GF(2^8)$.
$= 1/4 = 71$.

(Using a Cauchy generator matrix)

Erasure Codes —Reed Solomon (3)
Non-Systematic RS -- Vandermonde generator matrix

1^0	1^1	1^2	1^3	1^4	1^5
2^0	2^1	2^2	2^3	2^4	2^5
3^0	3^1	3^2	3^3	3^4	3^5
4^0	4^1	4^2	4^3	4^4	4^5
5^0	5^1	5^2	5^3	5^4	5^5
6^0	6^1	6^2	6^3	6^4	6^5
7^0	7^1	7^2	7^3	7^4	7^5
8^0	8^1	8^2	8^3	8^4	8^5
9^0	9^1	9^2	9^3	9^4	9^5

Example with
$k = 6$ and $n = 9$.

This is MDS,
so long as $n < 2^w$

Erasure Codes —Reed Solomon (4)
Non-Systematic RS -- Vandermonde generator matrix

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
1^0	1^1	1^2	1^3	1^4	1^5
2^0	2^1	2^2	2^3	2^4	2^5
3^0	3^1	3^2	3^3	3^4	3^5

Example with
$k = 6$ and $n = 9$.

The Vandermonde construction *only* applies to non-systematic codes.

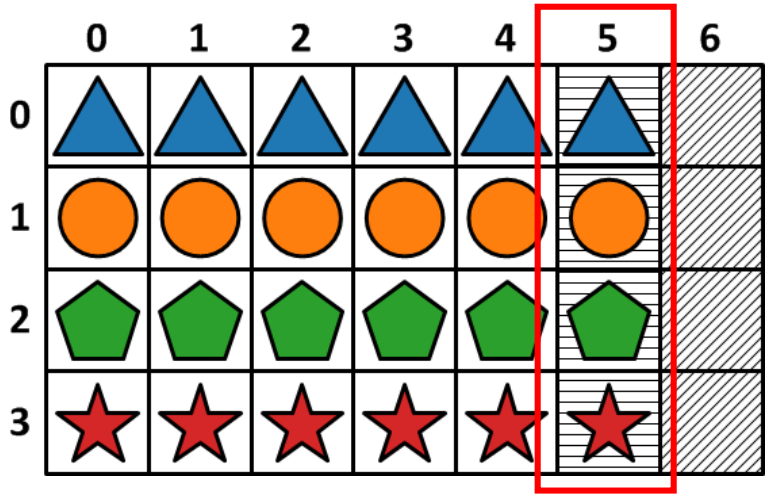
This generator matrix is *not guaranteed to be MDS.*

So don't use it!

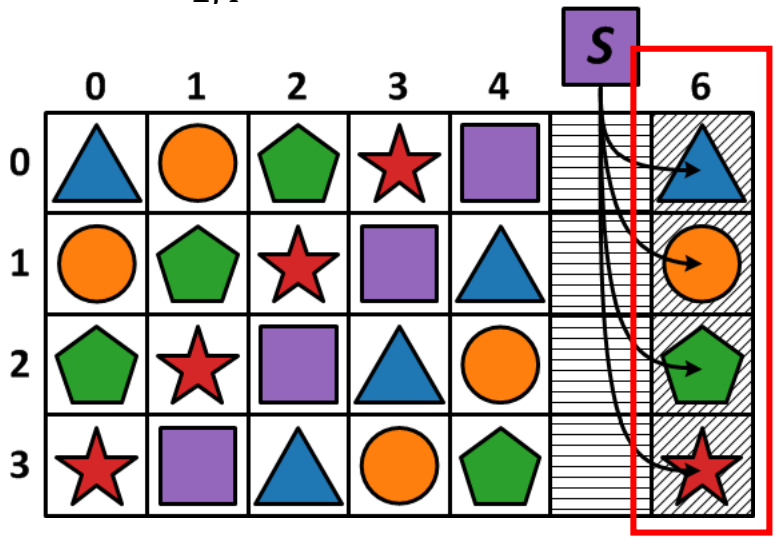
(You can convert an MDS non-systematic generator to a systematic one, though, if you care to.)

Erasure Codes —EVENODD 1995 (7 disks, tolerating 2 disk failures)

- Horizontal Parity Coding
- Calculated by the data elements in the same row
- E.g. $C_{0,5} = C_{0,0} \oplus C_{0,1} \oplus C_{0,2} \oplus C_{0,3} \oplus C_{0,4}$

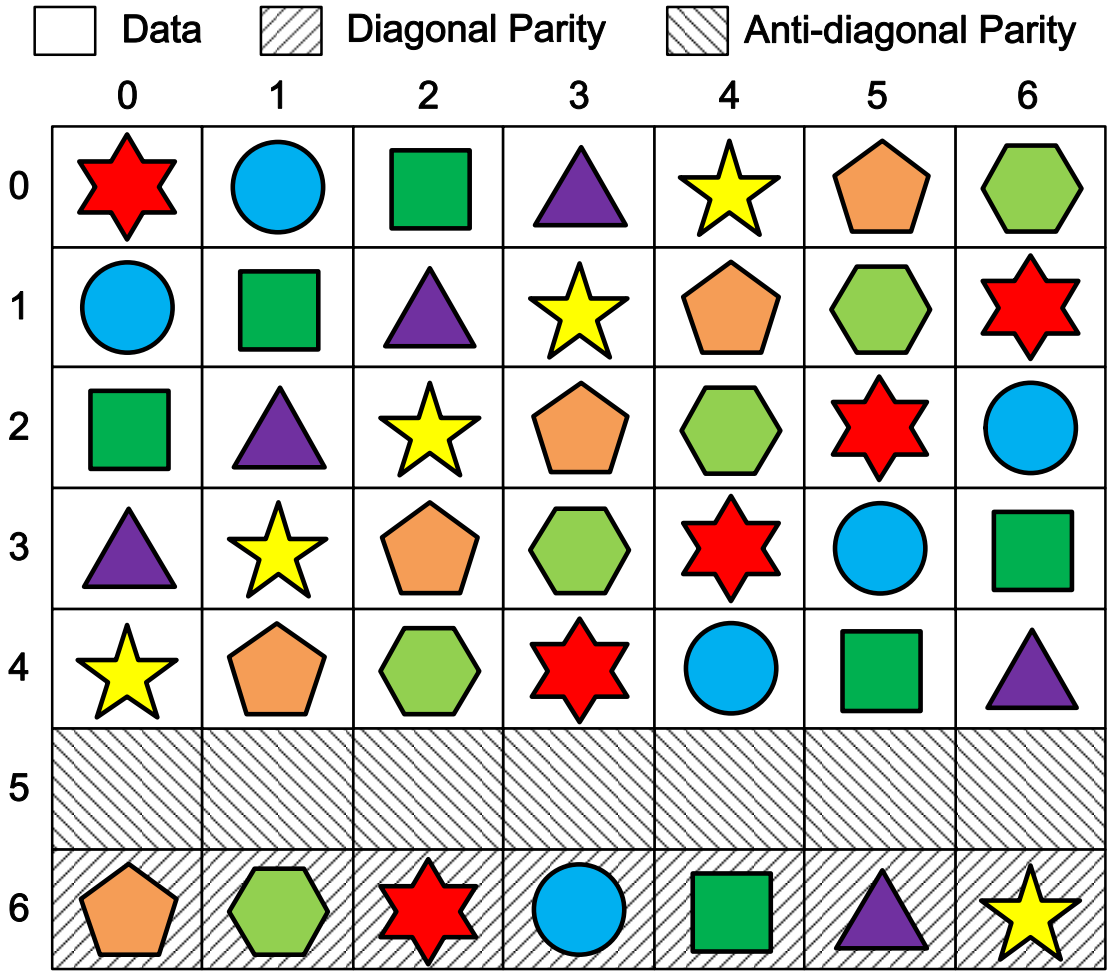


- Diagonal Parity Coding
- Calculated by the data elements and S
- E.g. $C_{0,6} = C_{0,0} \oplus C_{3,2} \oplus C_{2,3} \oplus C_{1,4} \oplus S$



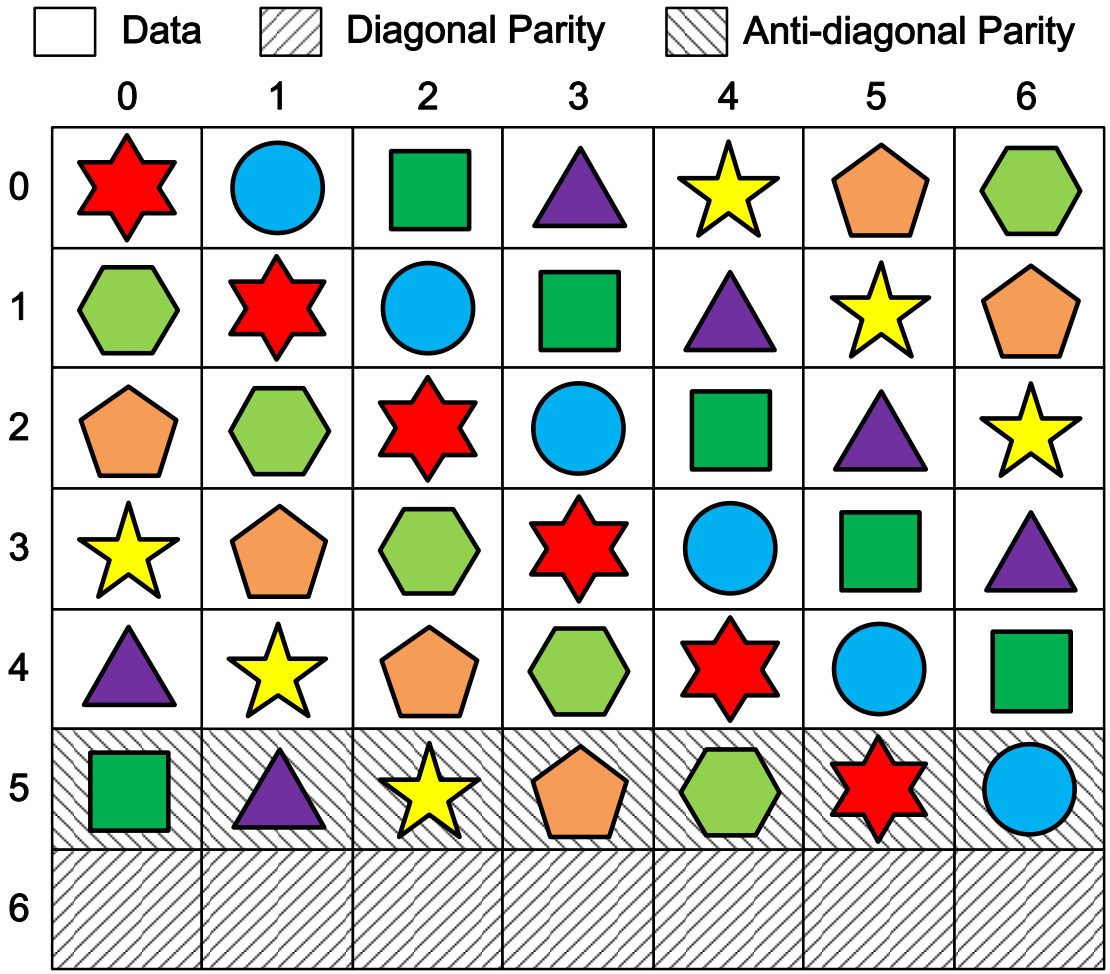
Erasure Codes —X-Code 1999 (1)

- Diagonal parity layout (p=7, n=7)



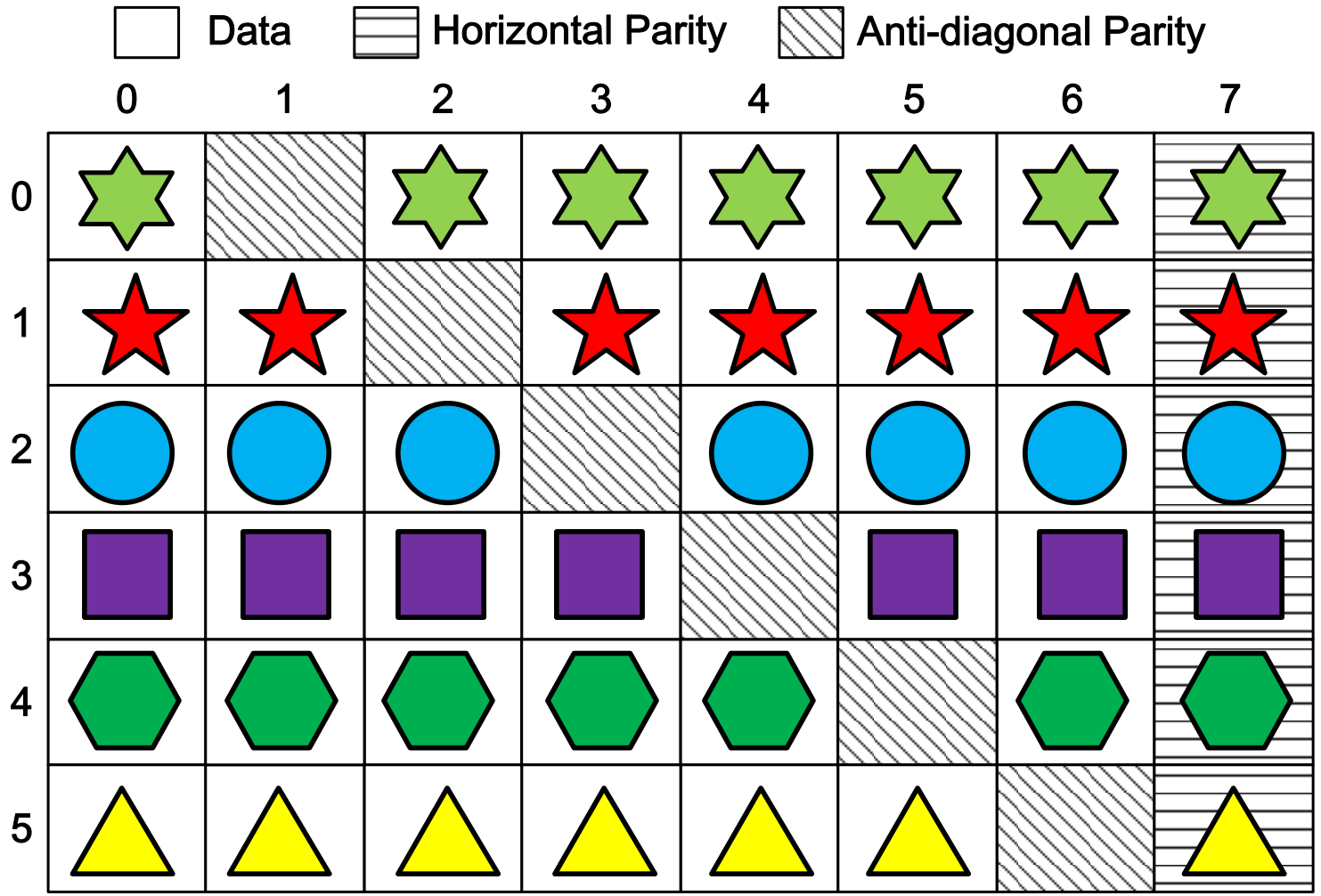
Erasure Codes —X-Code 1999 (2)

- Anti-diagonal parity layout (p=7, n=7)



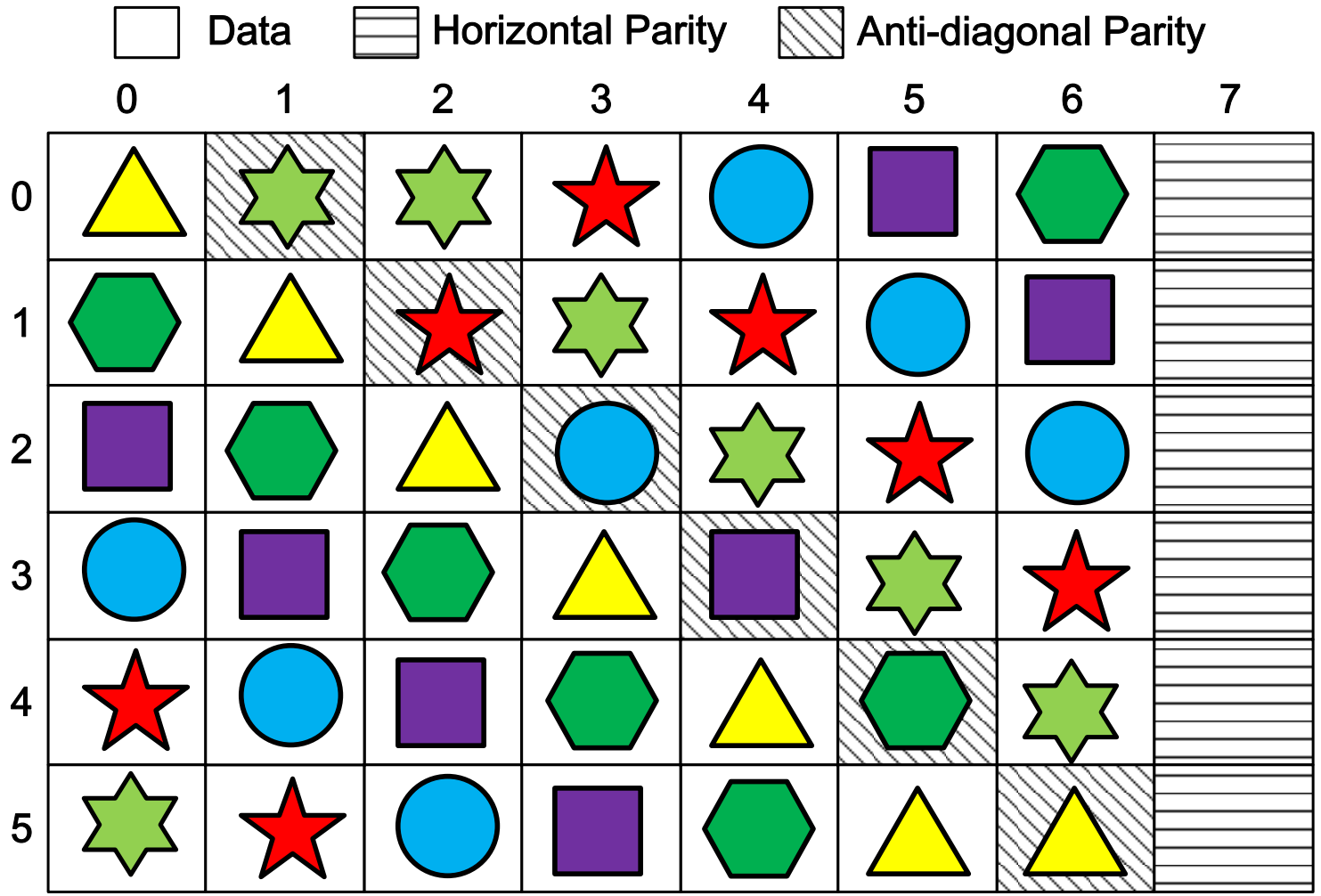
Erasure Codes —H-Code (1)

- Horizontal parity layout (p=7, n=8)



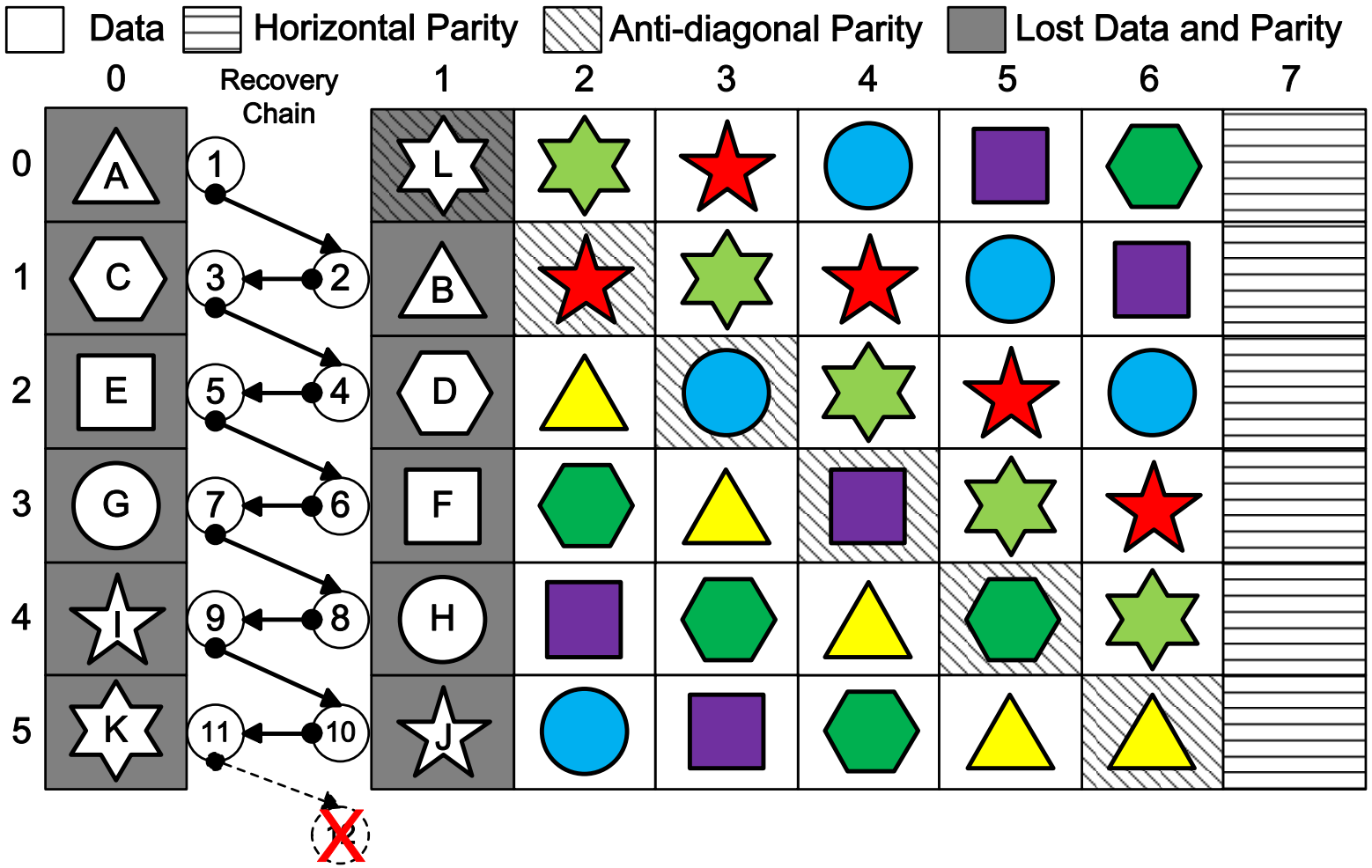
Erasure Codes —H-Code (2)

- Anti-diagonal parity layout (p=7, n=8)



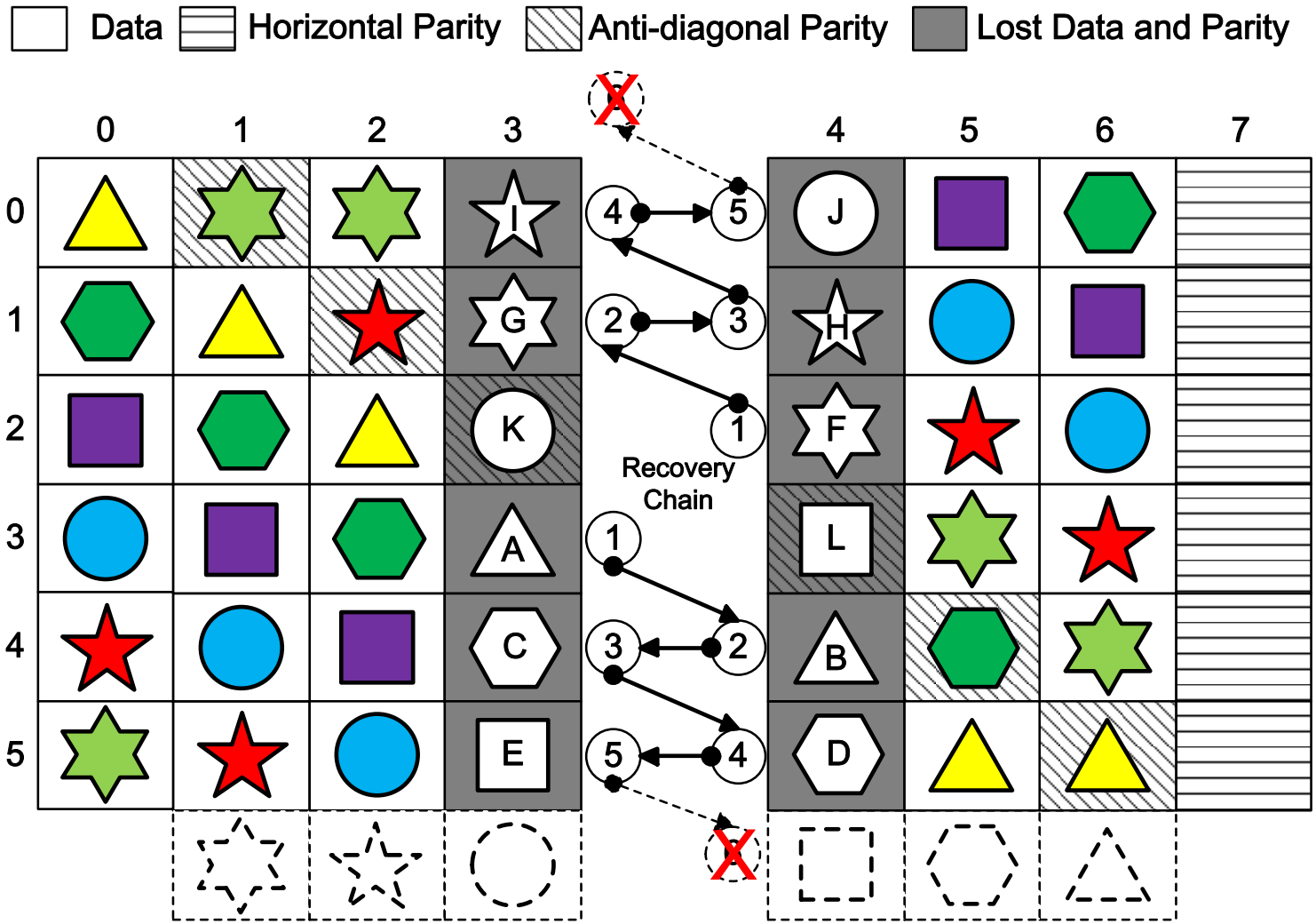
Erasure Codes —H-Code (3)

- Recover double disk failure by single recovery chain



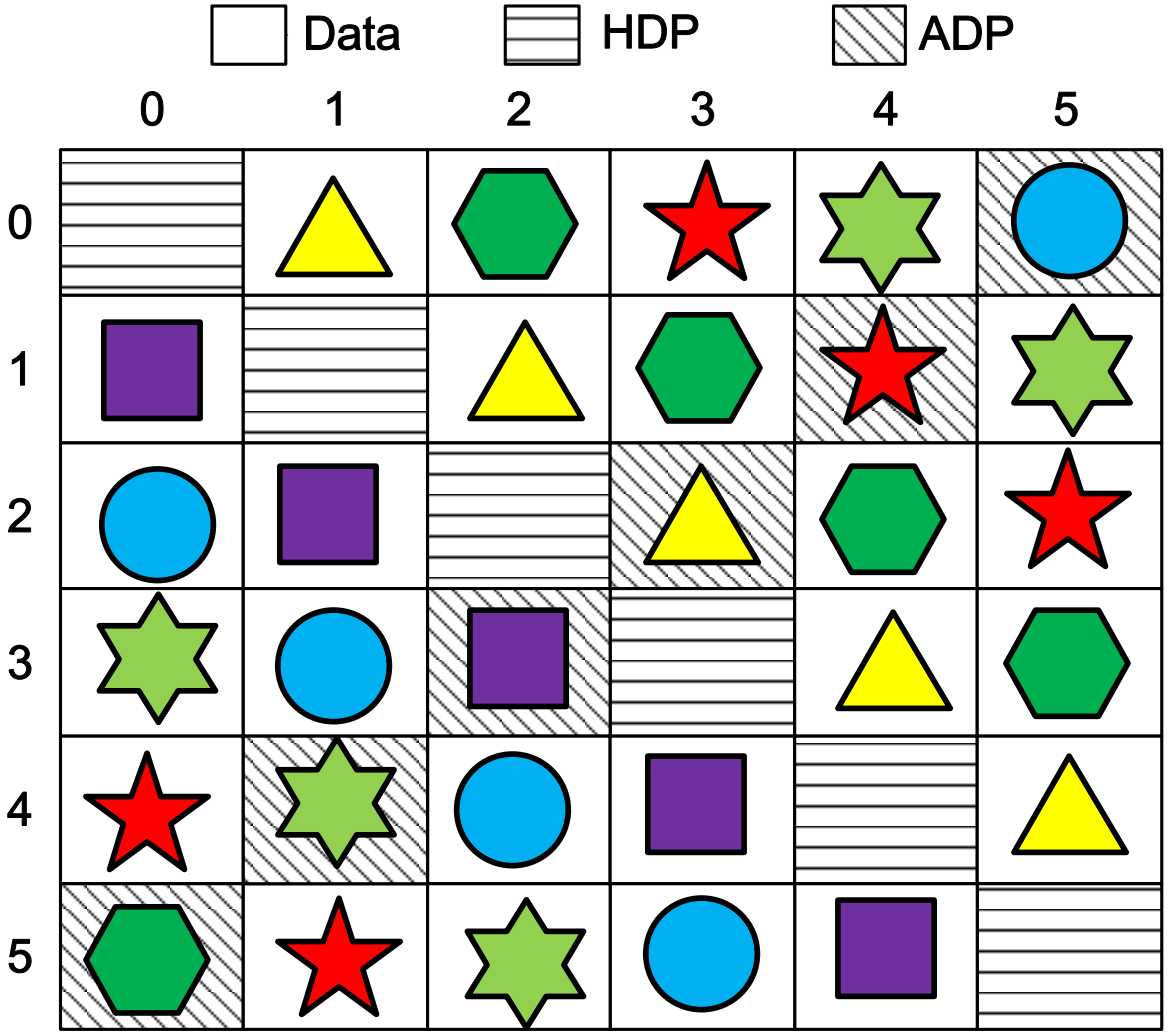
Erasure Codes —H-Code (4)

- Recover double disk failure by two recovery chains



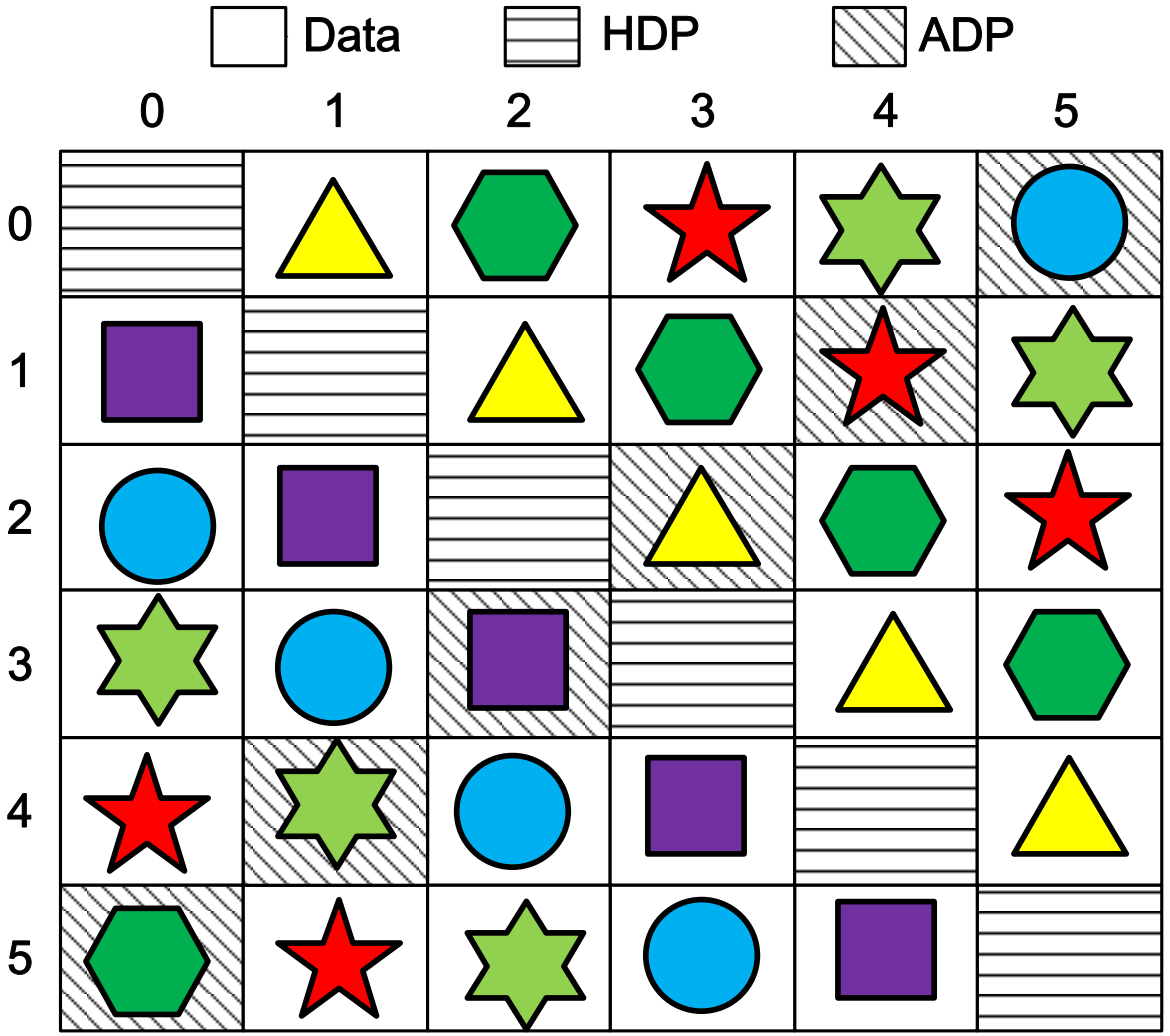
Erasure Codes —HDP Code (1)

- Diagonal parity layout (p=7, n=6)



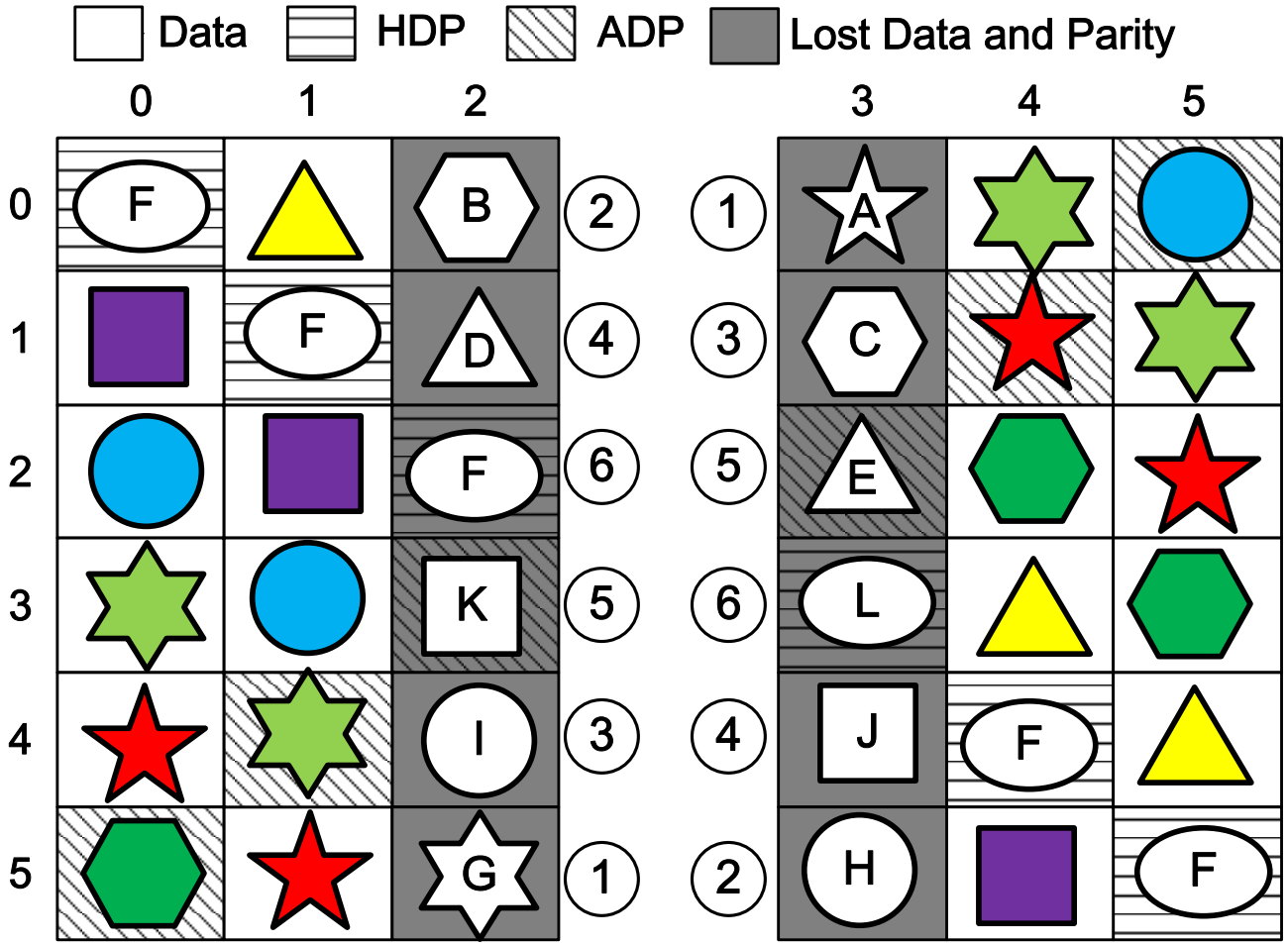
Erasure Codes —HDP Code (2)

- Diagonal parity layout (p=7, n=6)



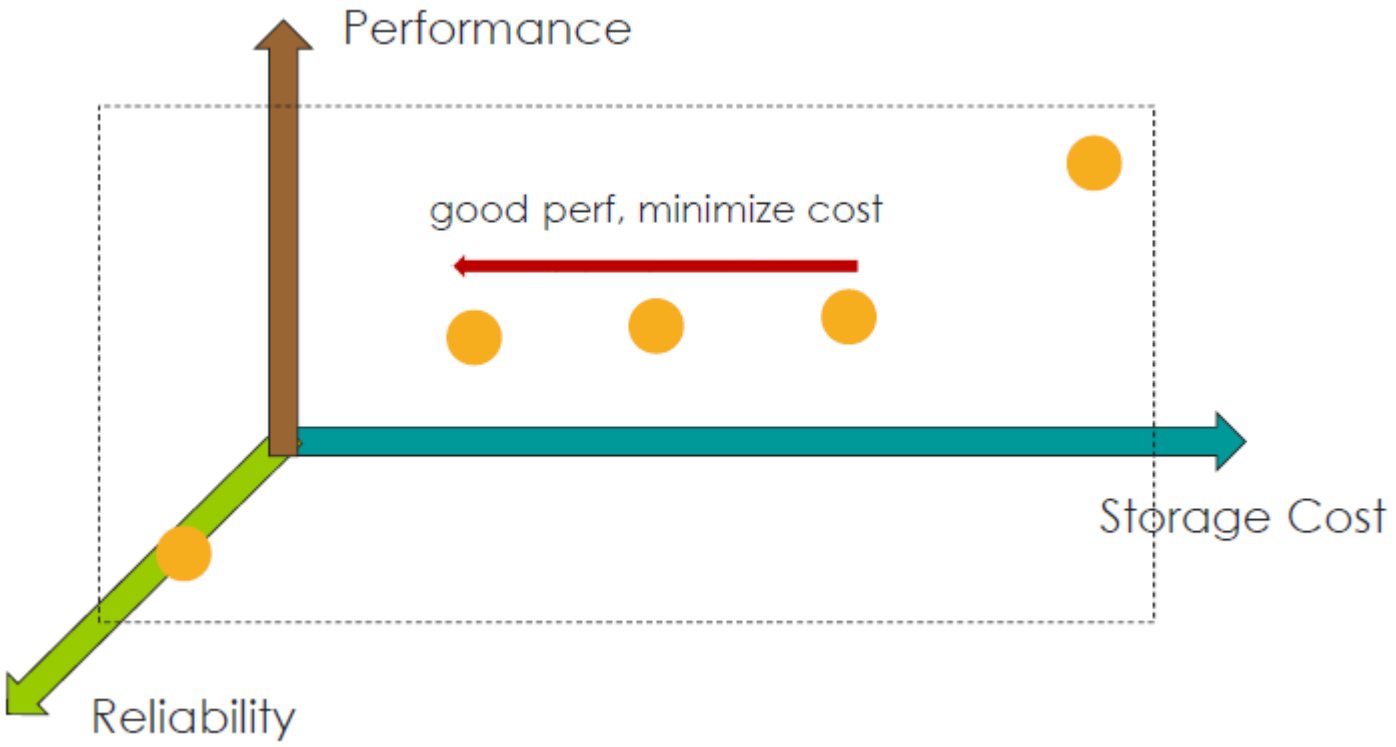
Erasure Codes —HDP Code (3)

- HDP reduces more than 30% average recovery time.

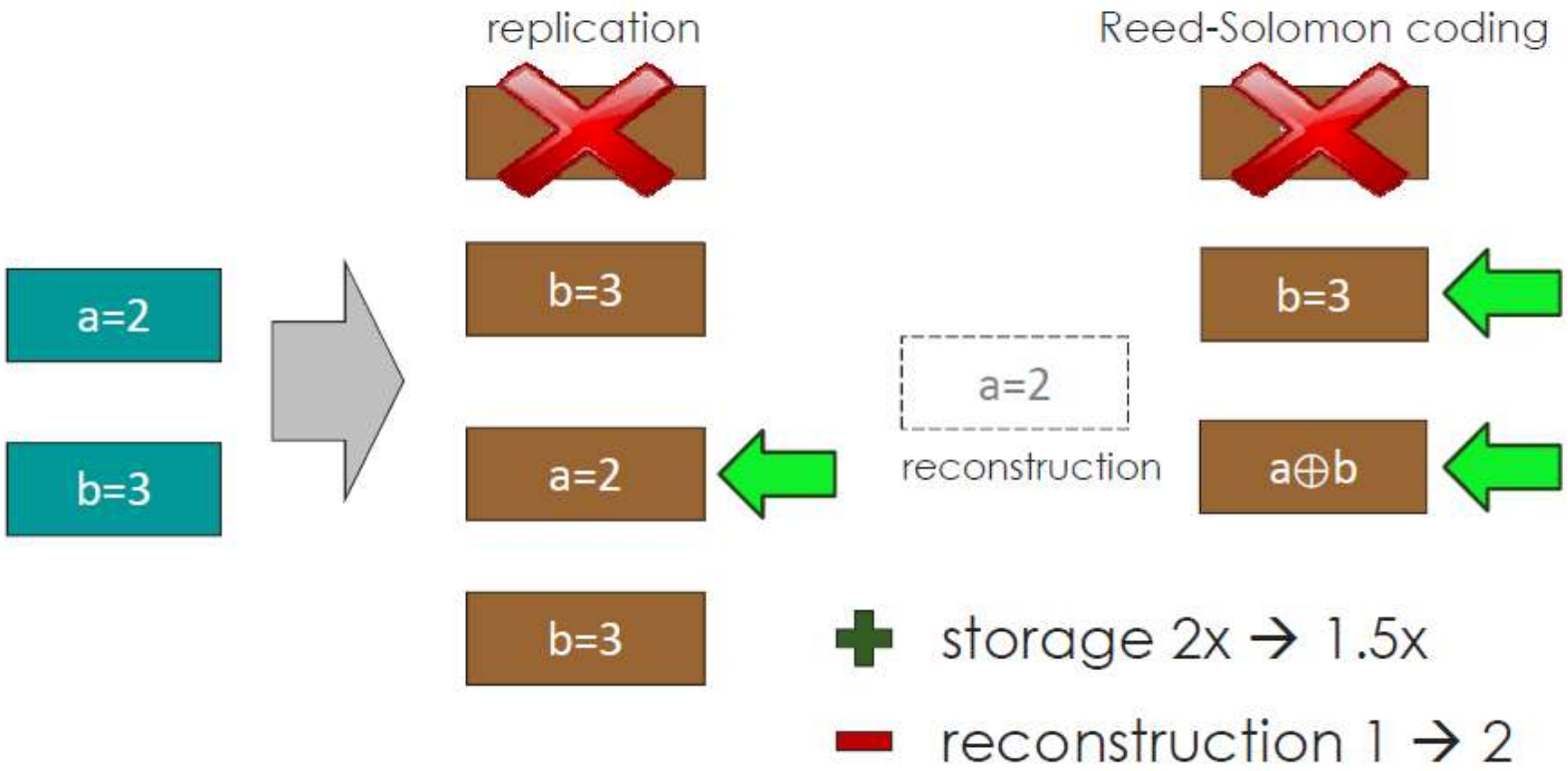


3 Replication and EC in Cloud

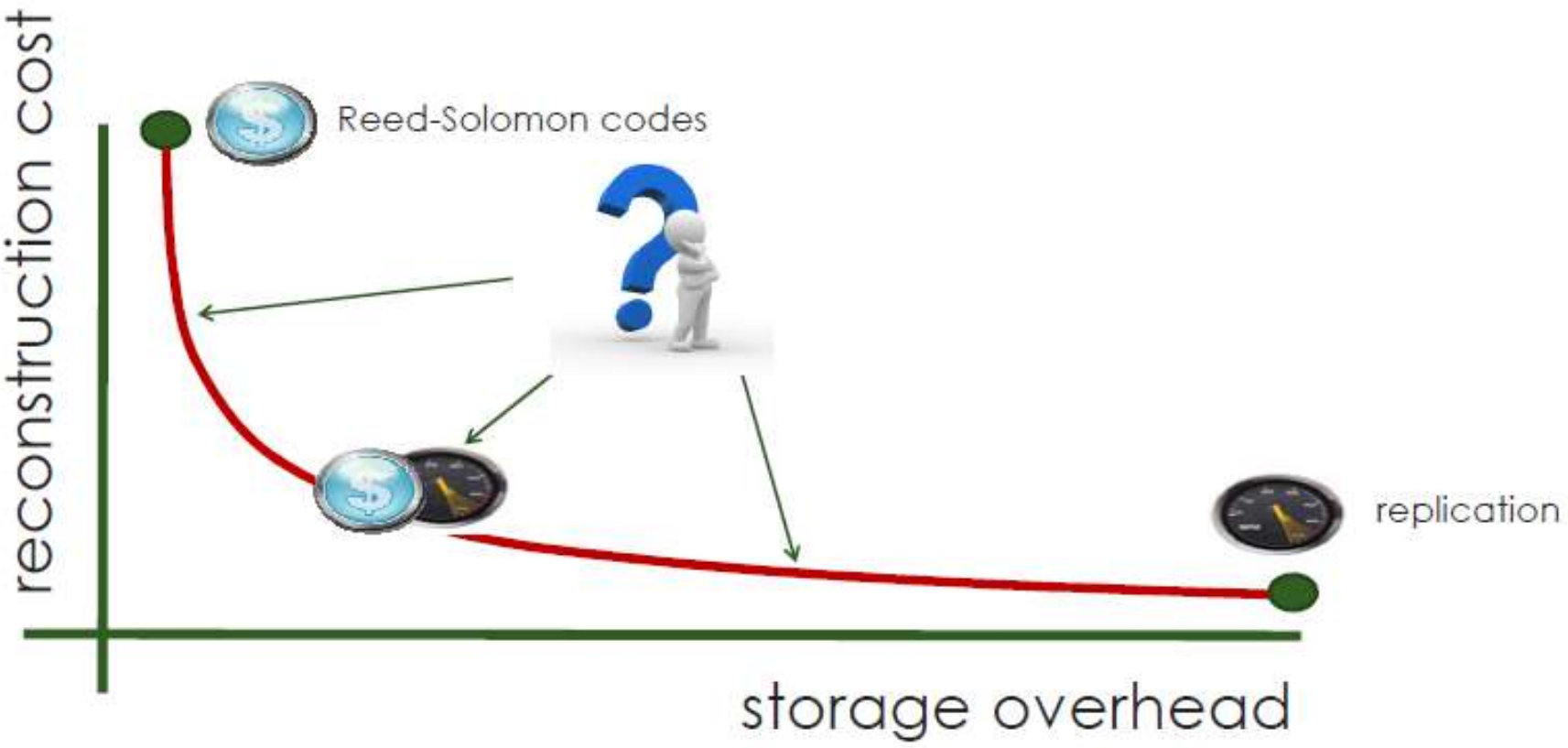
Three Dimensions in Cloud Storage



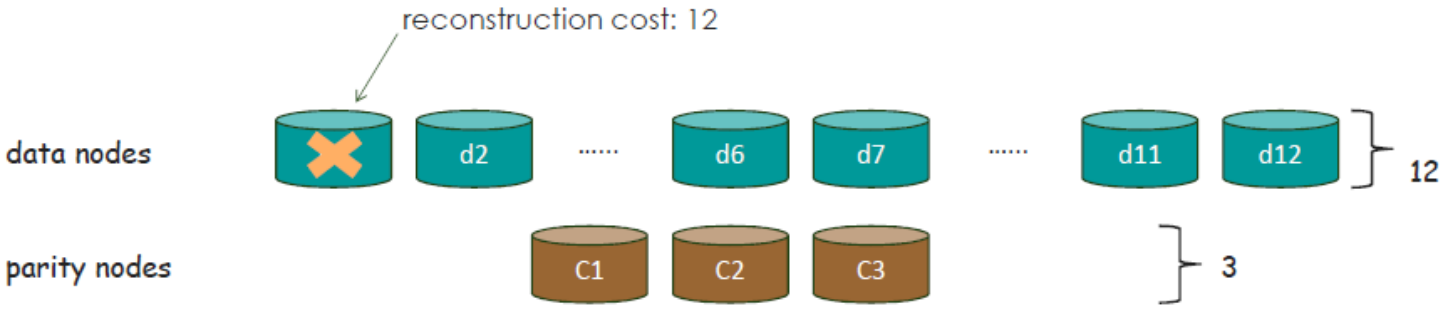
Replication vs Erasure Coding (RS)



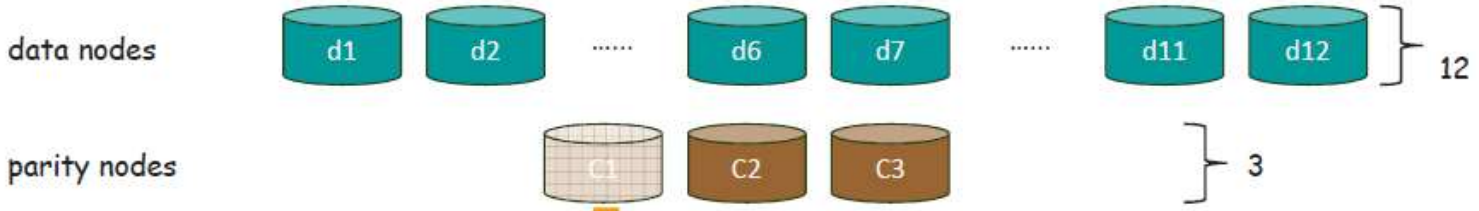
Fundamental Tradeoff



Pyramid Codes (1)



Reed-Solomon 12 + 3

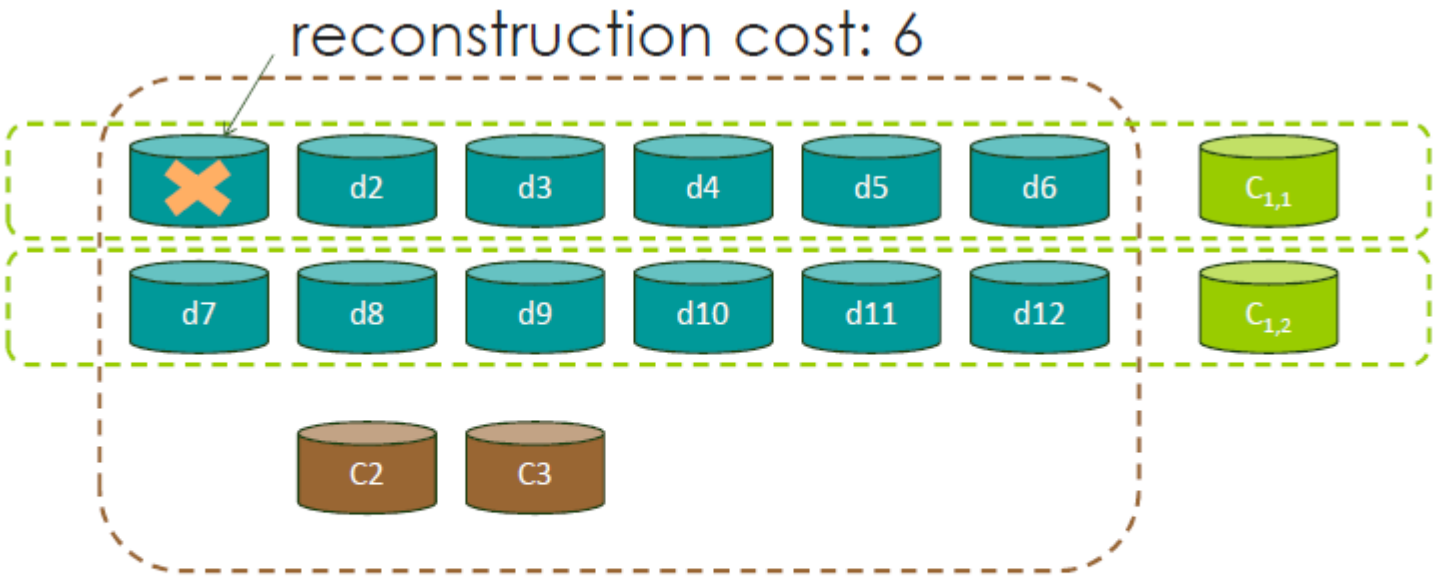


Pyramid Codes Construction:

- take an arbitrary Reed-Solomon (RS) code
- split one RS parity into multiple local parities
- 12 + 3 RS → 12 + 4 Pyramid

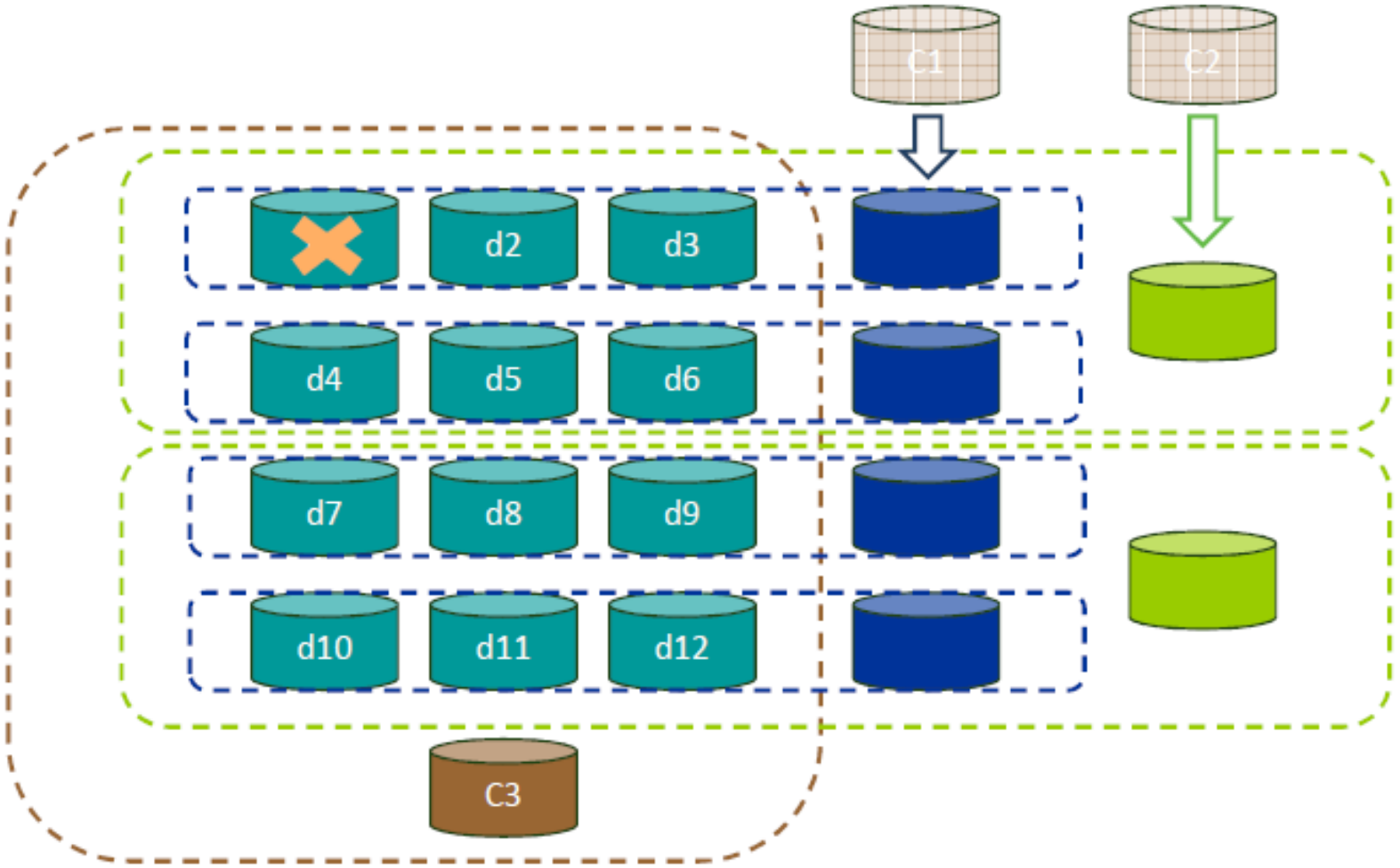
- $C_{1,1}$: {d1, d2, ..., d6}
- $C_{1,2}$: {d7, d8, ..., d12}

Pyramid Codes (2)



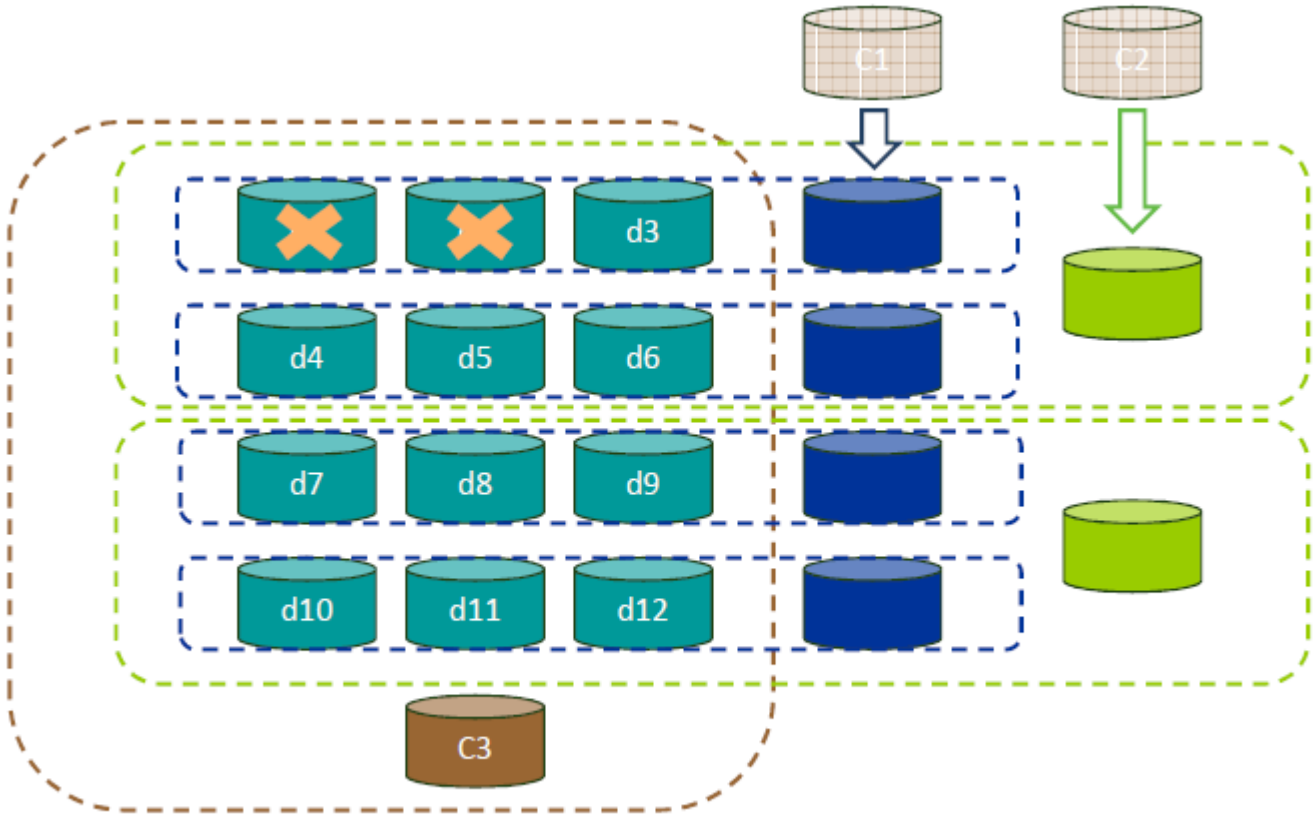
- storage overhead: 1.33x
- tolerate arbitrary 3 failures

Pyramid Codes (3) Multiple Hierachies



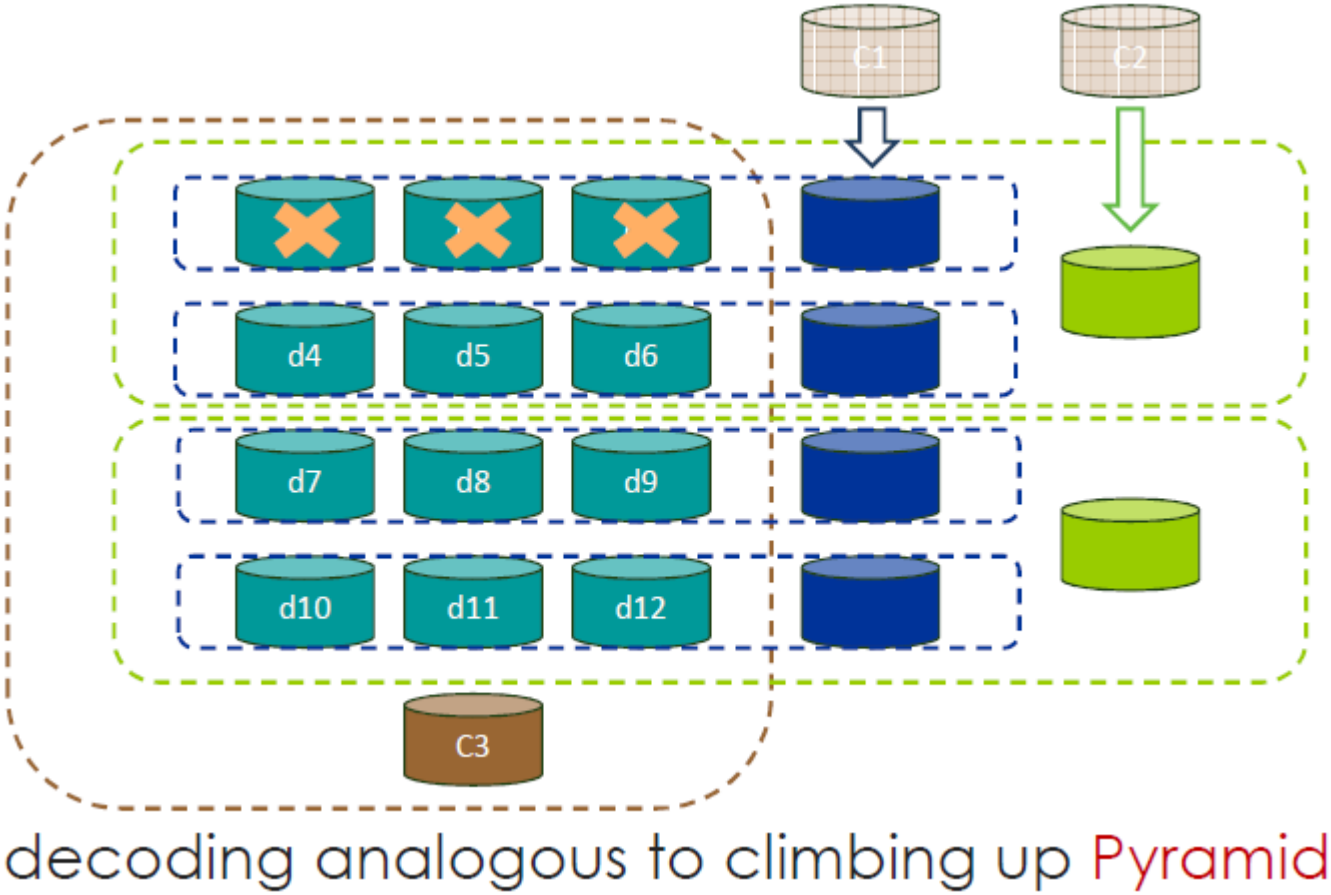
reconstruction cost of d_1 → 3

Pyramid Codes (4) Multiple Hierachies



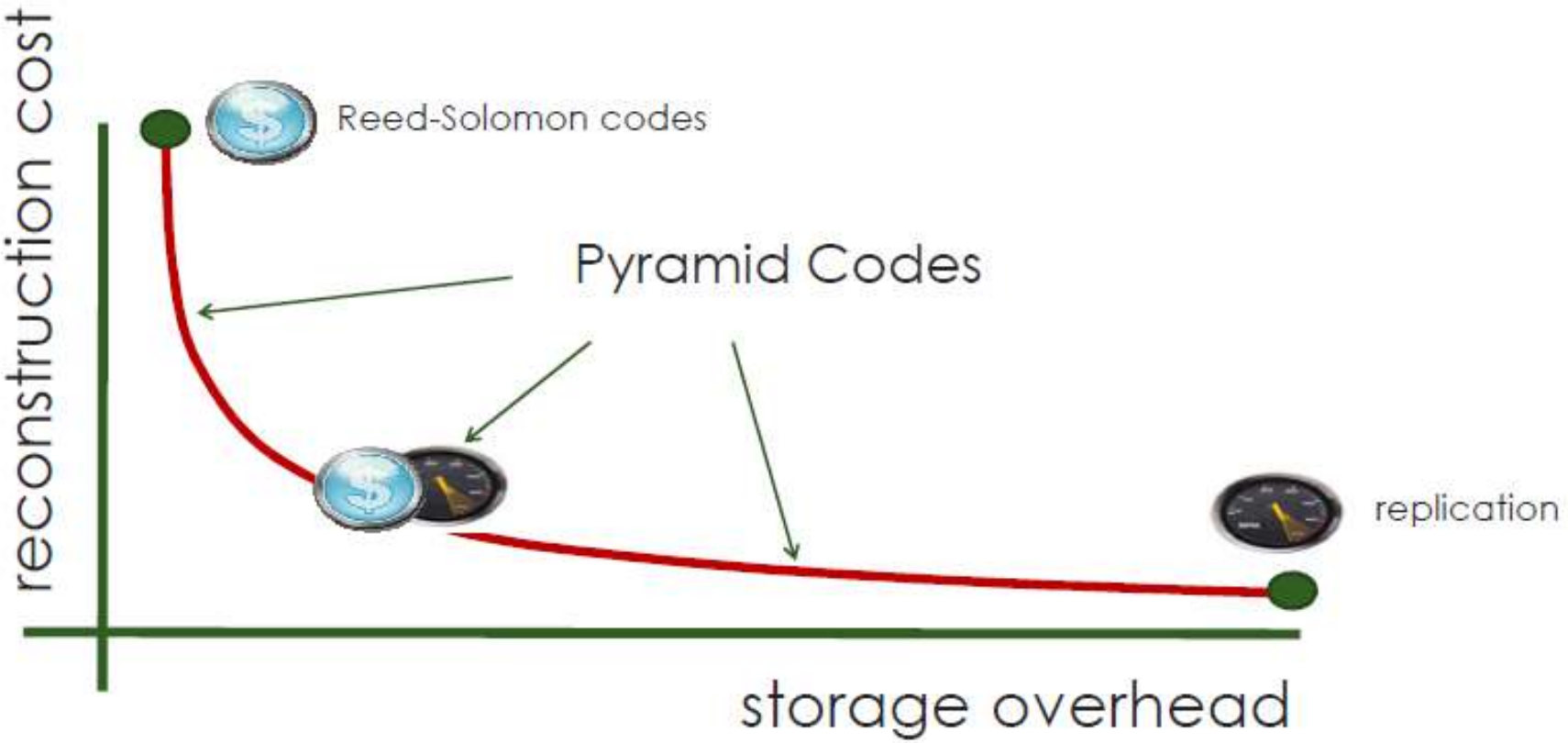
reconstruction cost of d_1 and d_2 → 6

Pyramid Codes (5) Multiple Hierachies

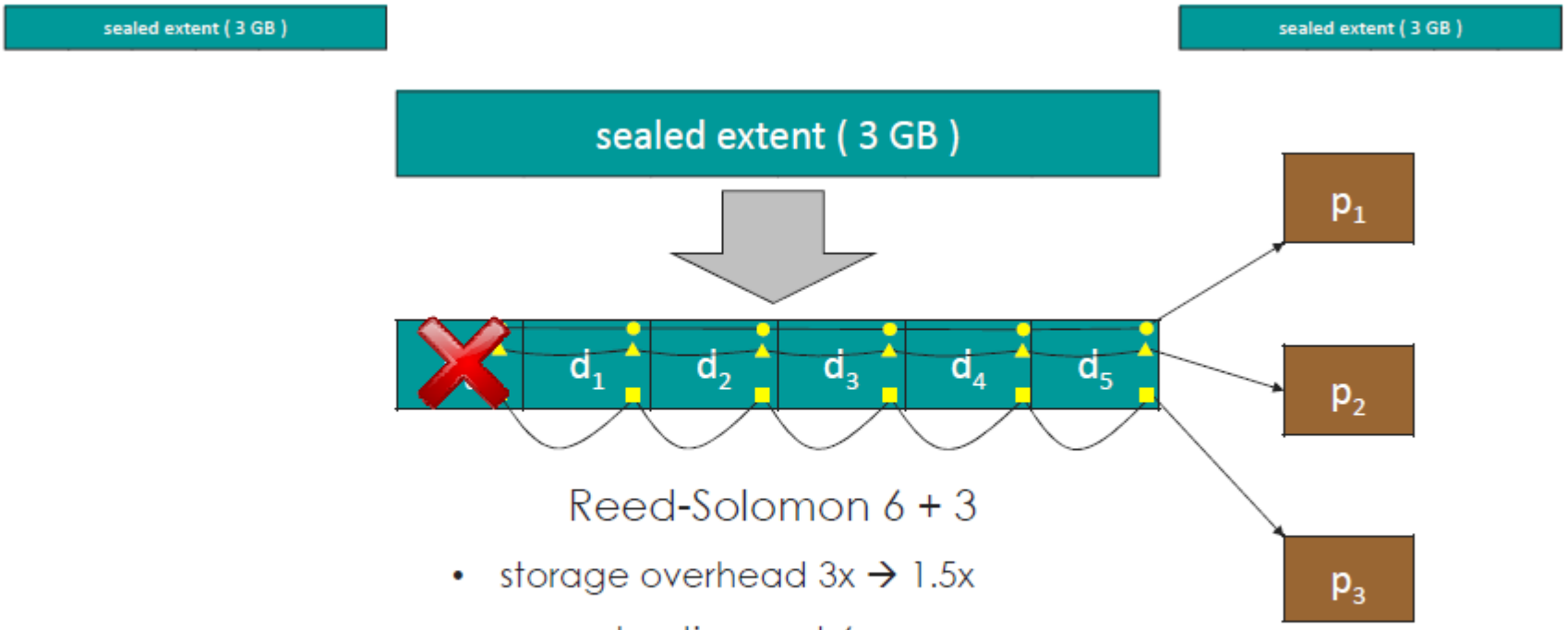


decoding analogous to climbing up Pyramid

Pyramid Codes (6)



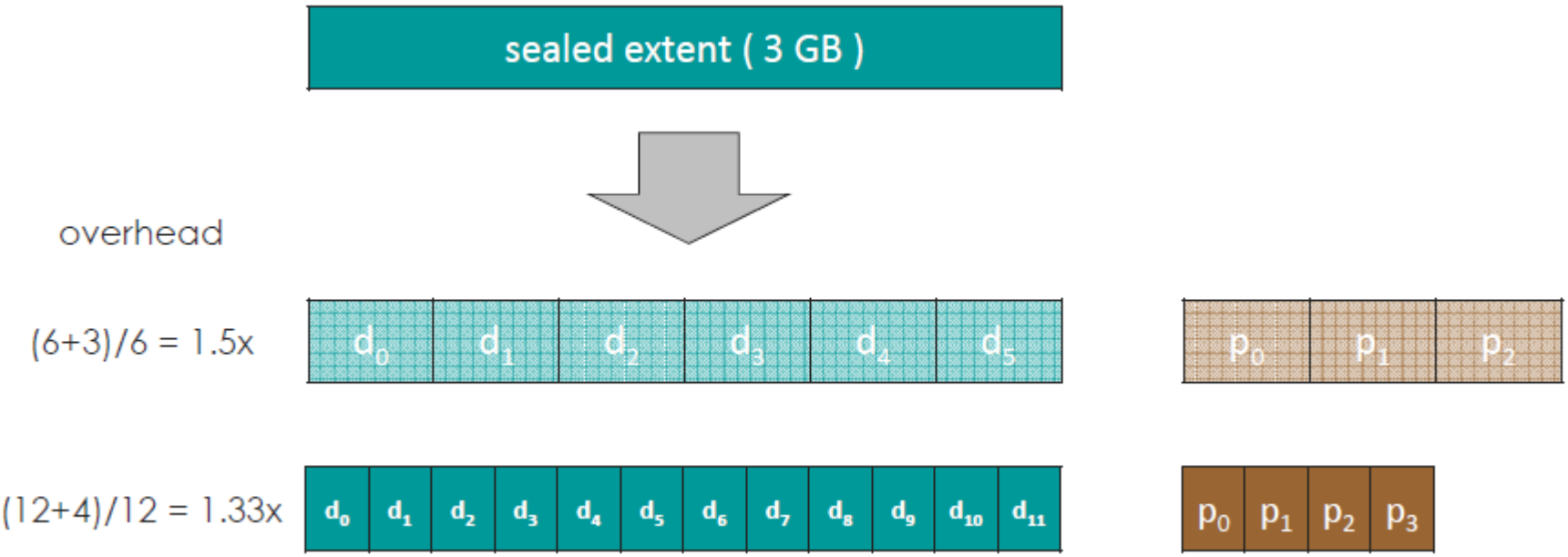
Google GFS II – Based on RS



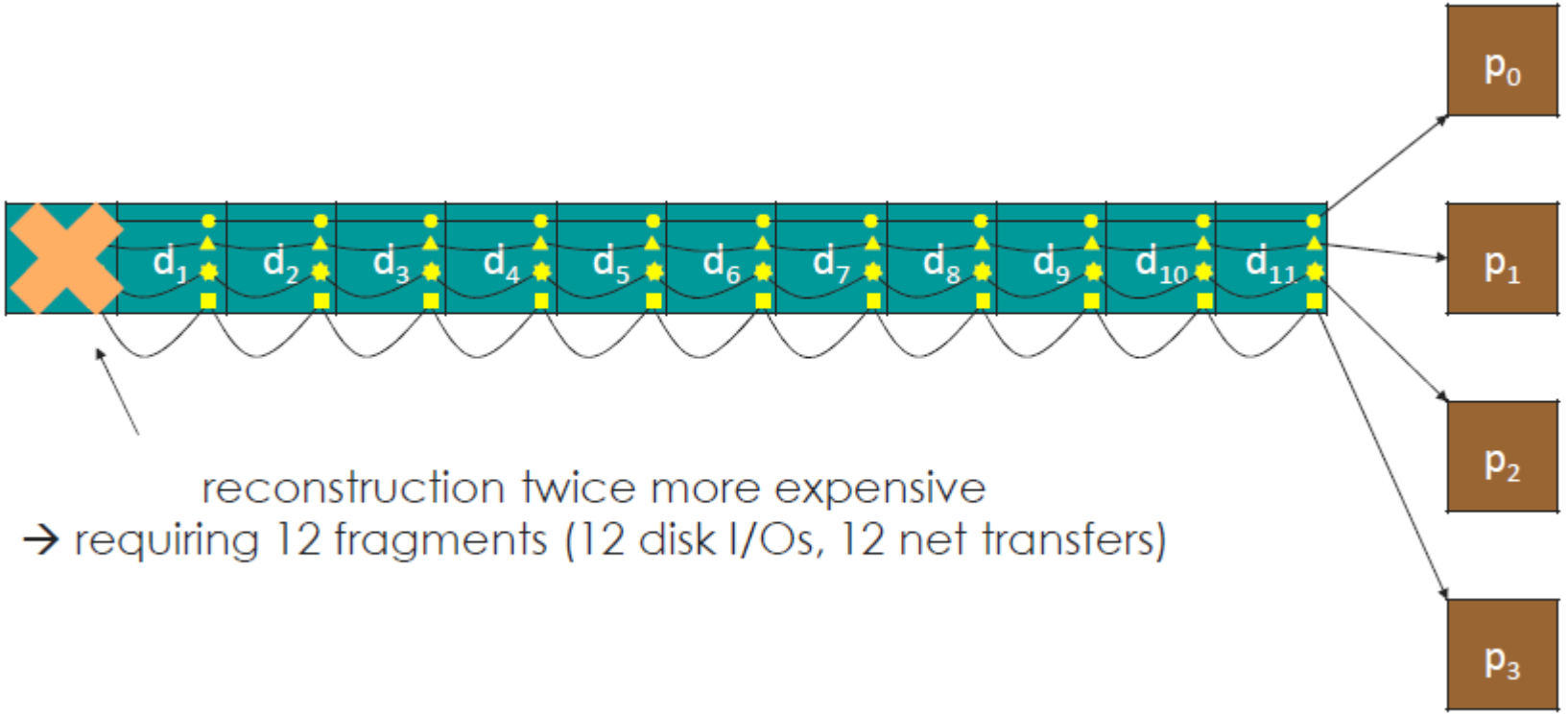
Reed-Solomon 6 + 3

- storage overhead 3x → 1.5x
- reconstruction cost 6
- used in Google GFS II (as of 2012)

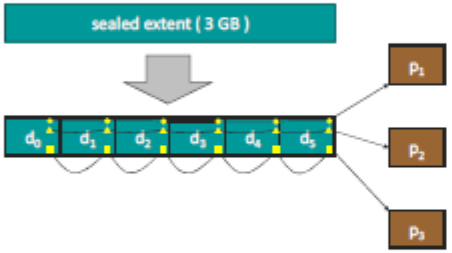
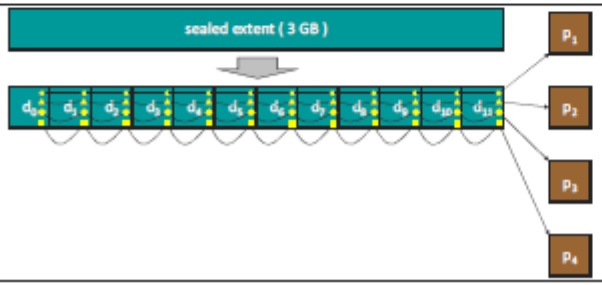
Microsoft Azure (1)
How to Reduce Cost?



Microsoft Azure (2)
Recovery becomes expensive

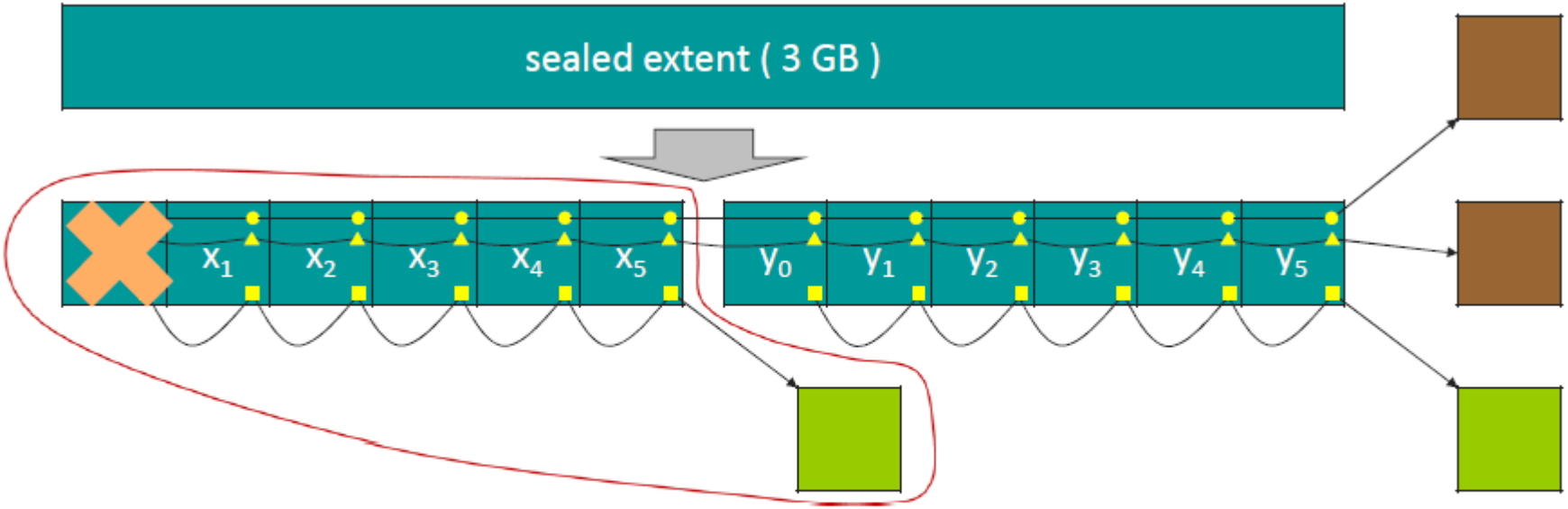


Microsoft Azure (3)
Best of both worlds?

Conventional Reed-Solomon Coding	Storage Overhead	Reconstruction Cost
	1.5x	6 reads
	1.33x	12 reads

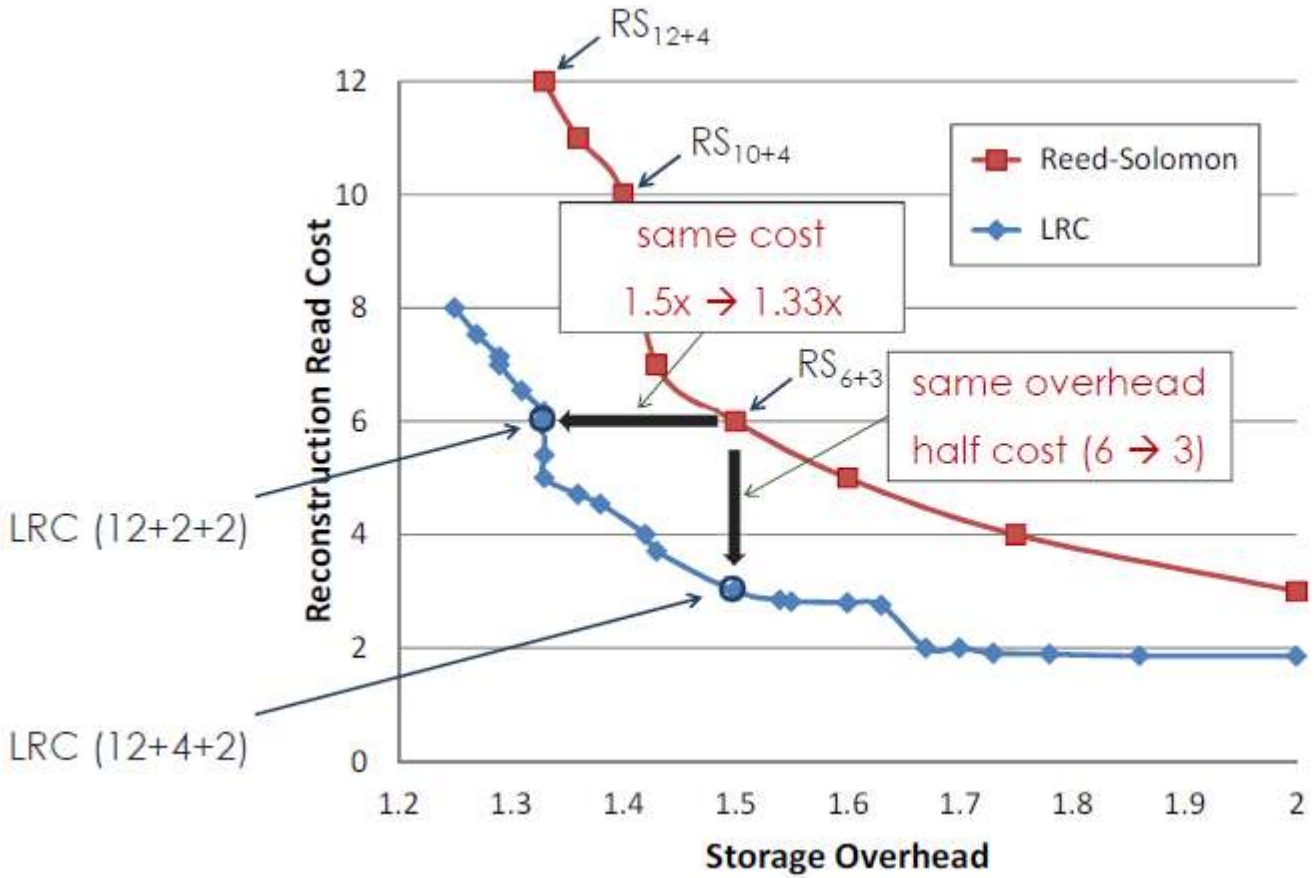
LRC

Microsoft Azure (4)
Local Reconstruction Code (LRC)



- LRC_{12+2+2}: **12** data fragments, **2** local parities and **2** global parities
  - storage overhead: (12 + 2 + 2) / 12 = 1.33x
- Local parity: reconstruction requires only 6 fragments

Microsoft Azure (5)
Analysis LRC vs RS

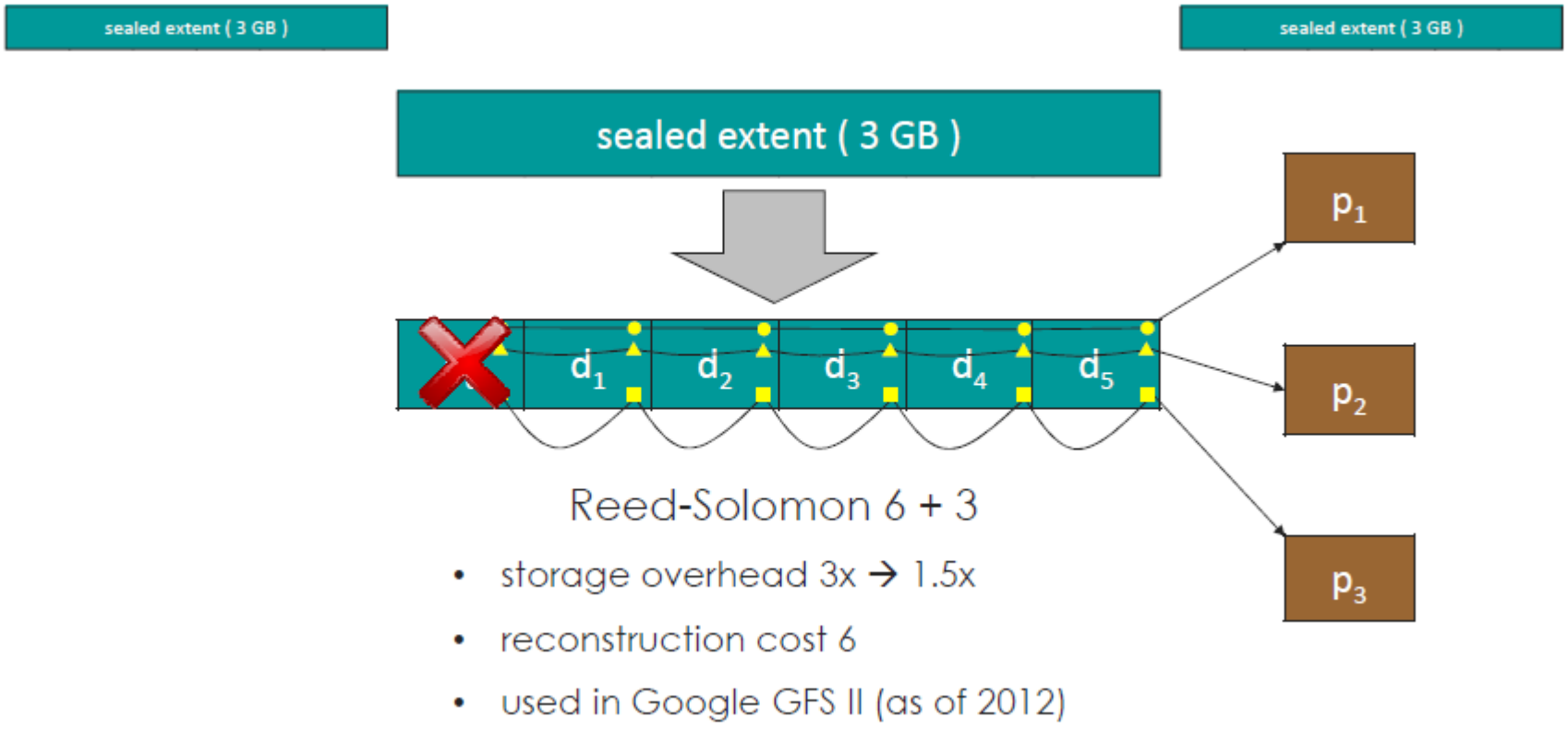


- RS_{10+4}: HDFS-RAID at Facebook
- RS_{6+3}: GFS II (Colossus) at Google

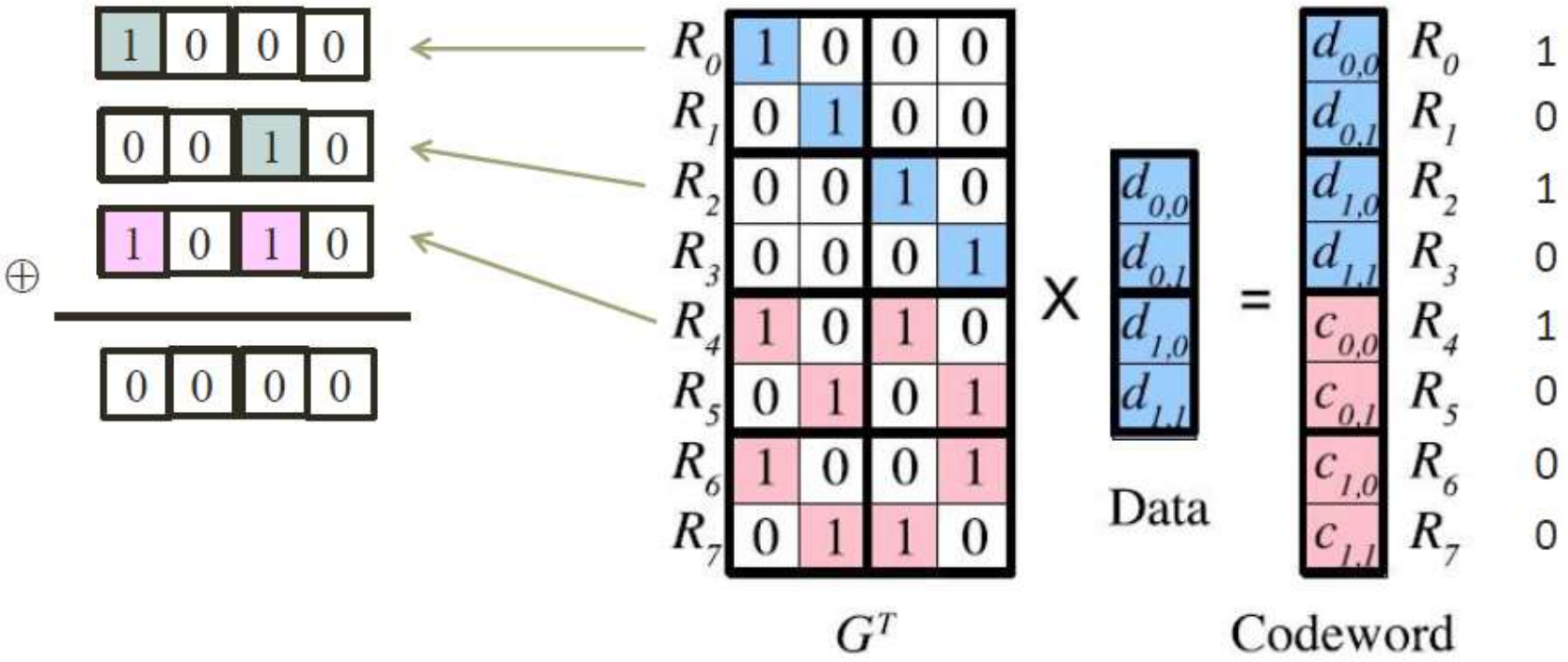
Microsoft Azure (6)
Analysis LRC vs RS

Recovery problem in Cloud

- Recovery I/Os from 6 disks (high network bandwidth)



Optimizing Recovery Network I/O (1)

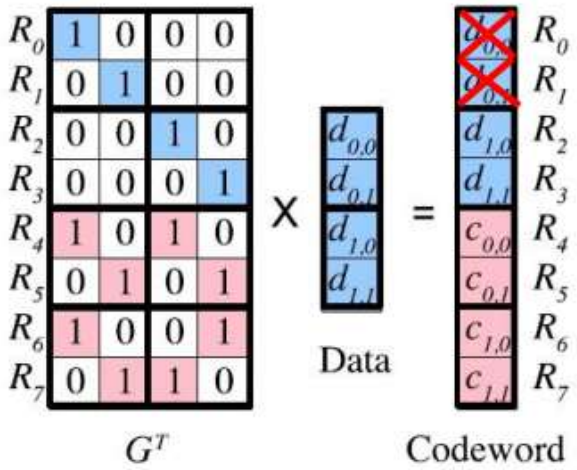


{R_0, R_2, R_4} is a **decoding equation**

And it can be represented by 10101000

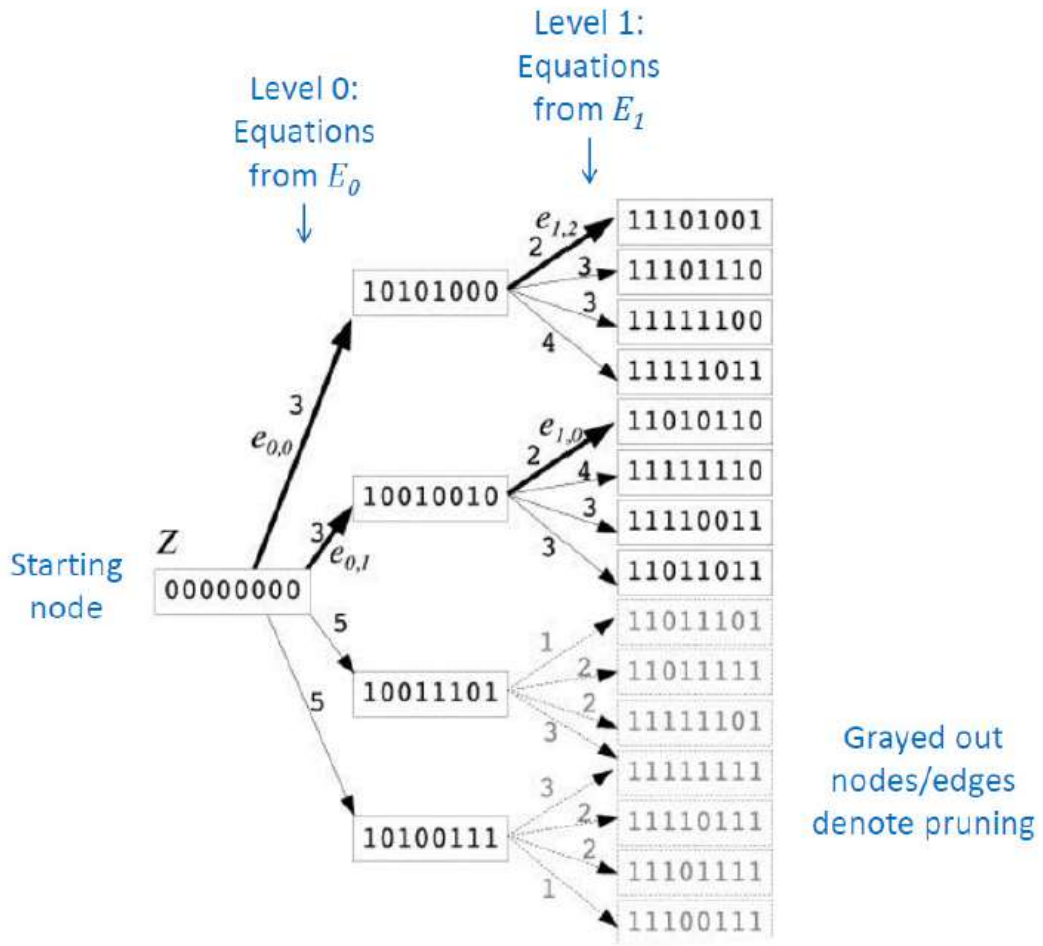
Optimizing Recovery Network I/O (1)

- Establish recovery relationships among disks



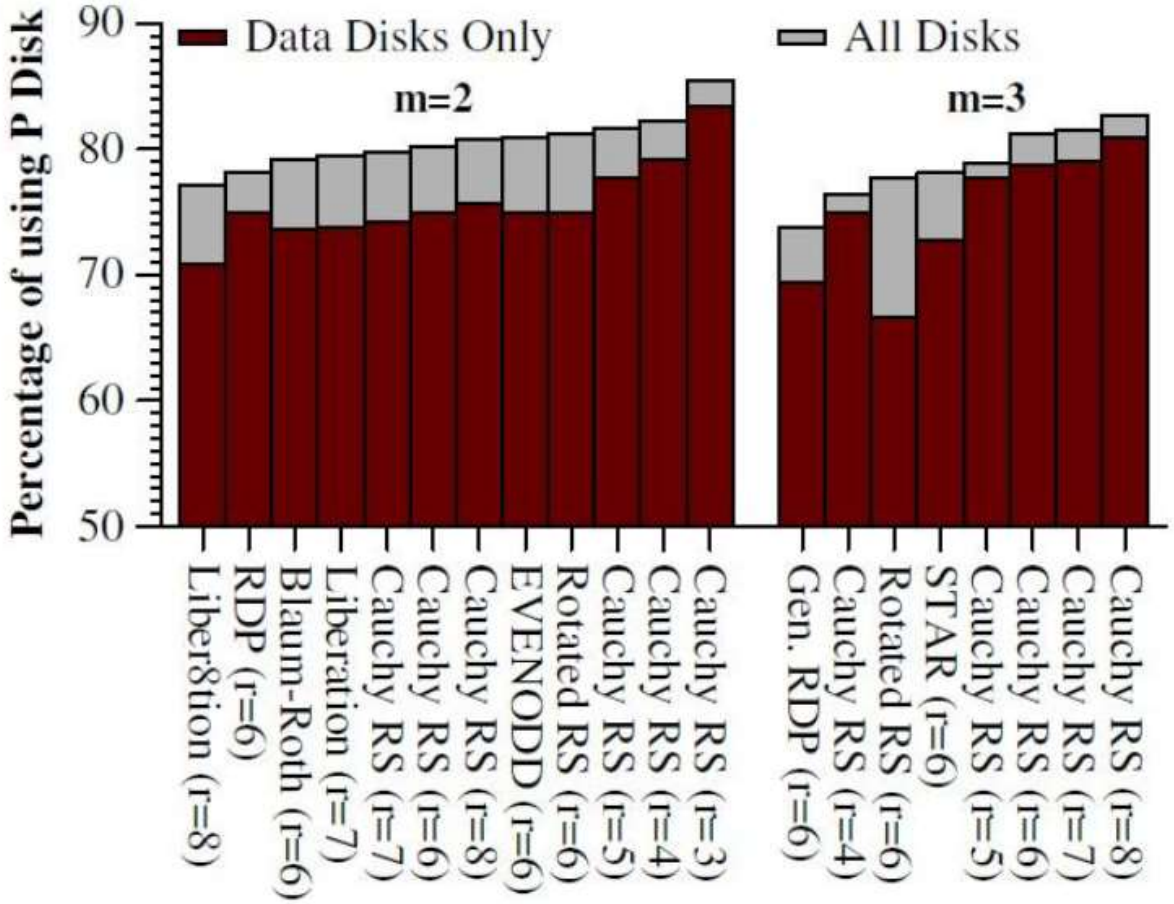
E_0	E_1
$e_{0,0}$ = 10101000	$e_{1,0}$ = 01010100
$e_{0,1}$ = 10010010	$e_{1,1}$ = 01101110
$e_{0,2}$ = 10011101	$e_{1,2}$ = 01100001
$e_{0,3}$ = 10100111	$e_{1,3}$ = 01011011

Recovery options for R_0 | Recovery options for R_1



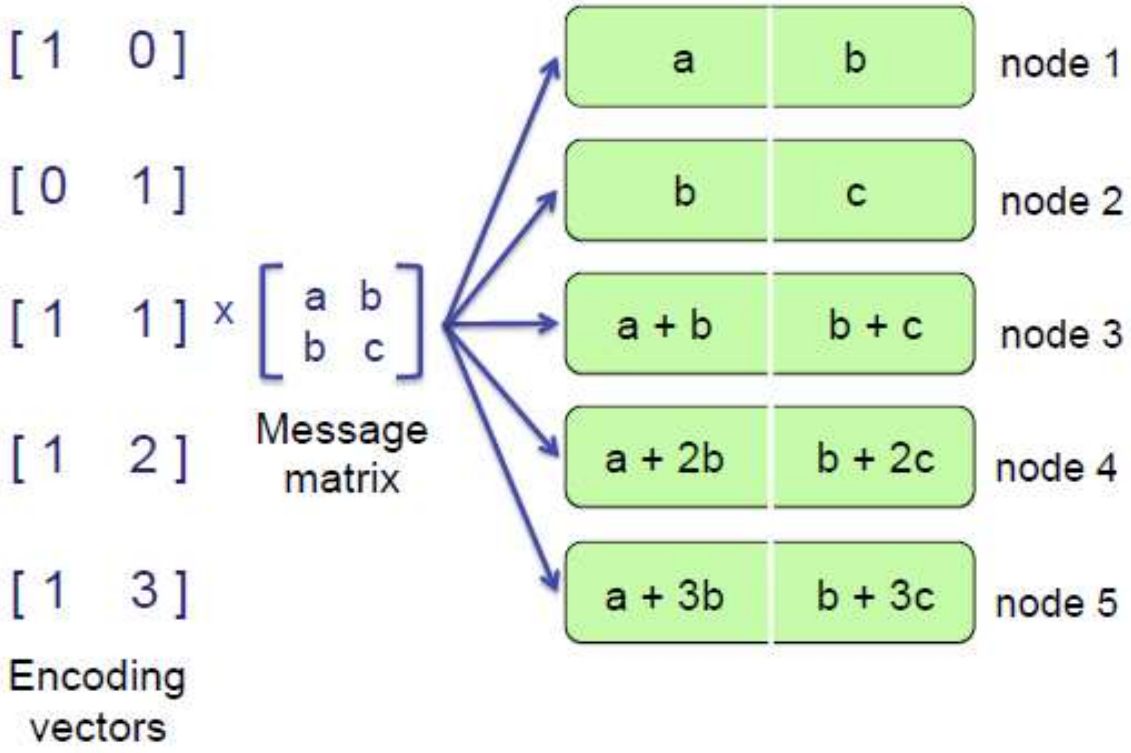
Optimizing Recovery I/O (3)

- ~20+% savings in general



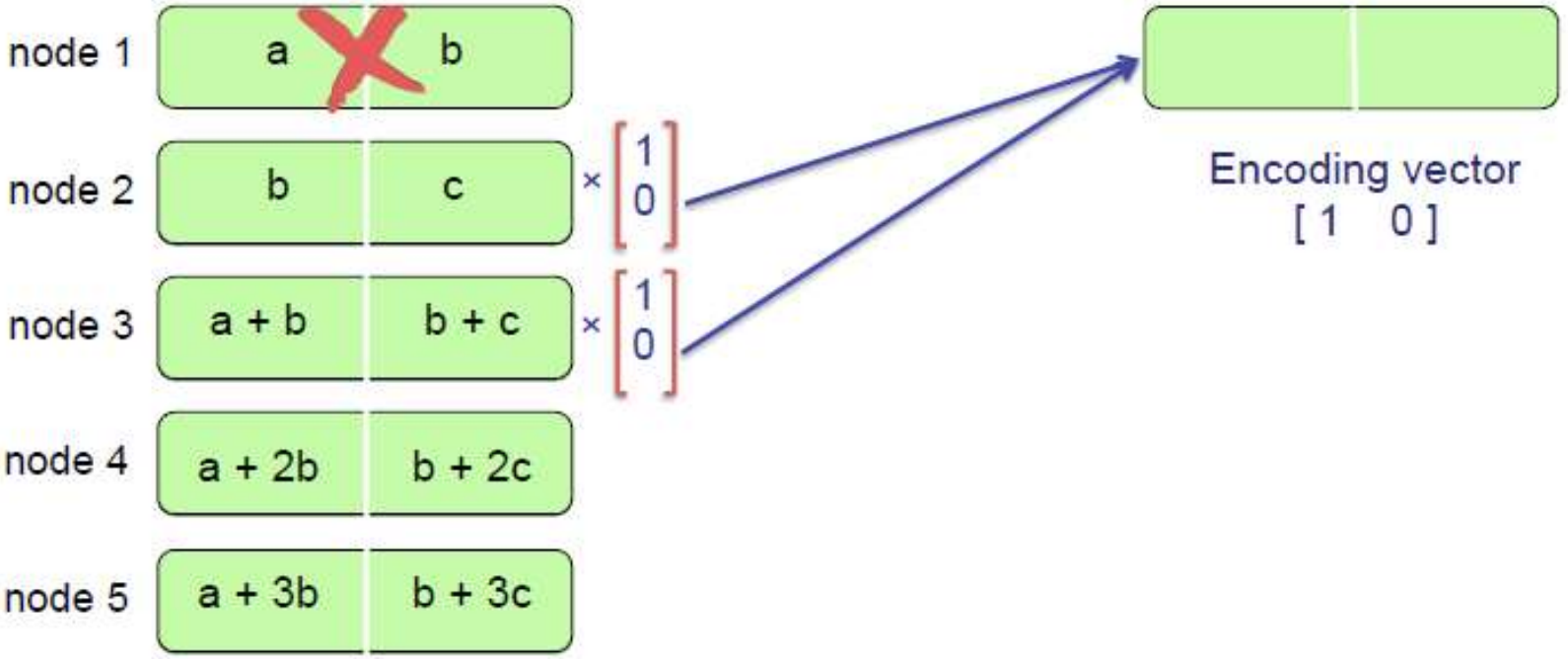
Regenerating Codes (1)

- Data = {a,b,c}



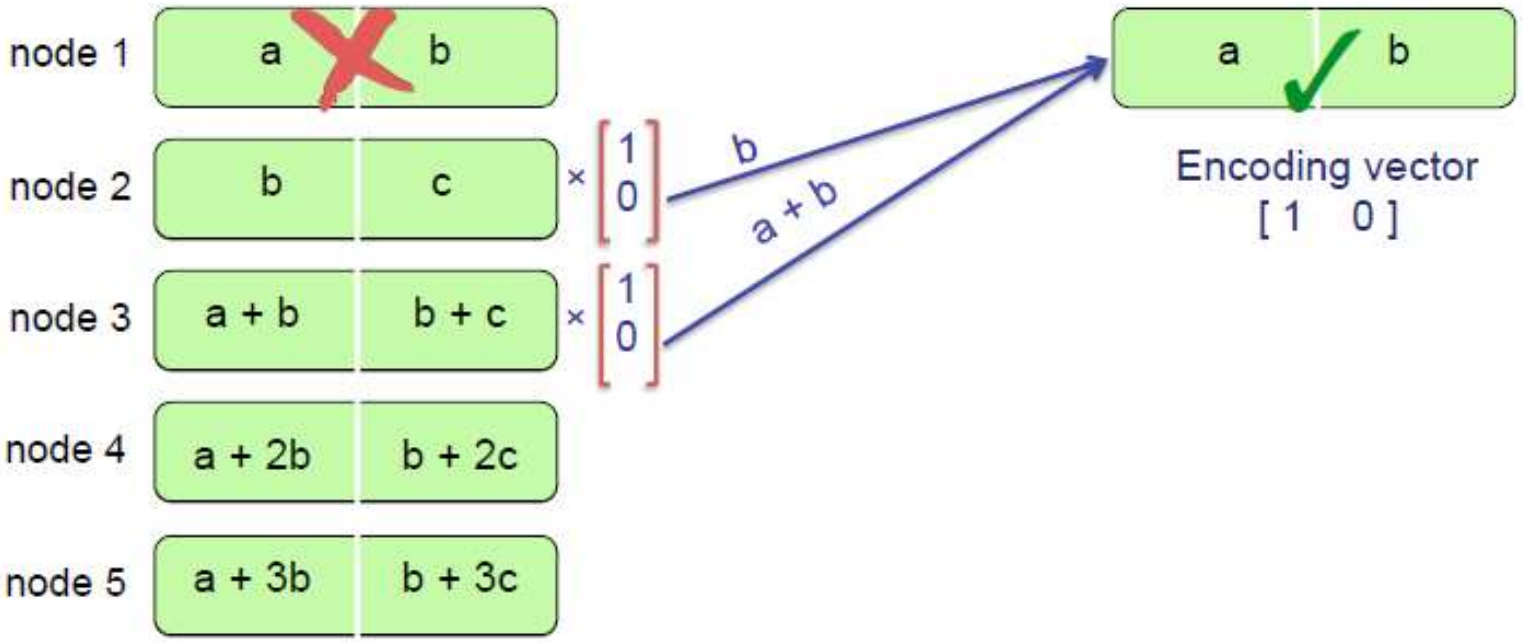
Regenerating Codes (2)

- Optimal Repair



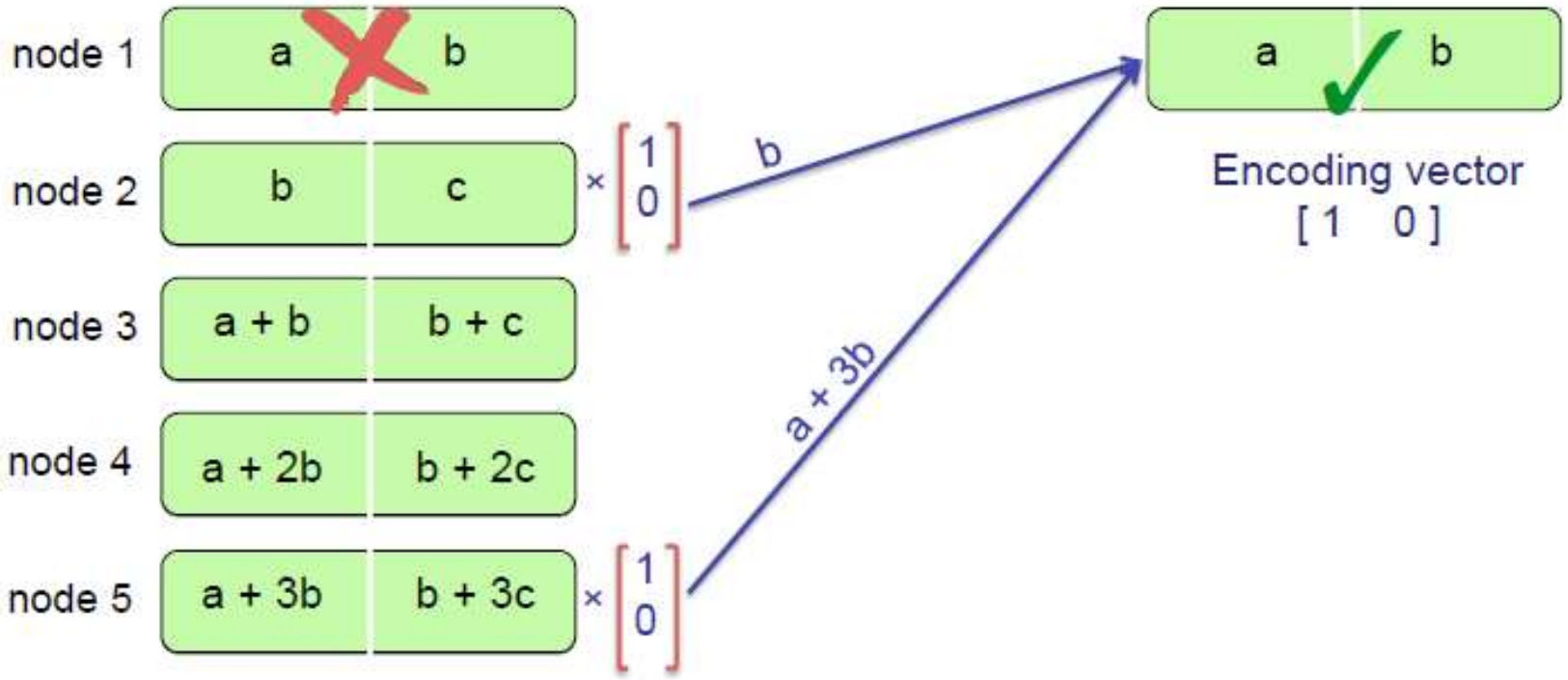
Regenerating Codes (3)

- Optimal Repair



Regenerating Codes (4)

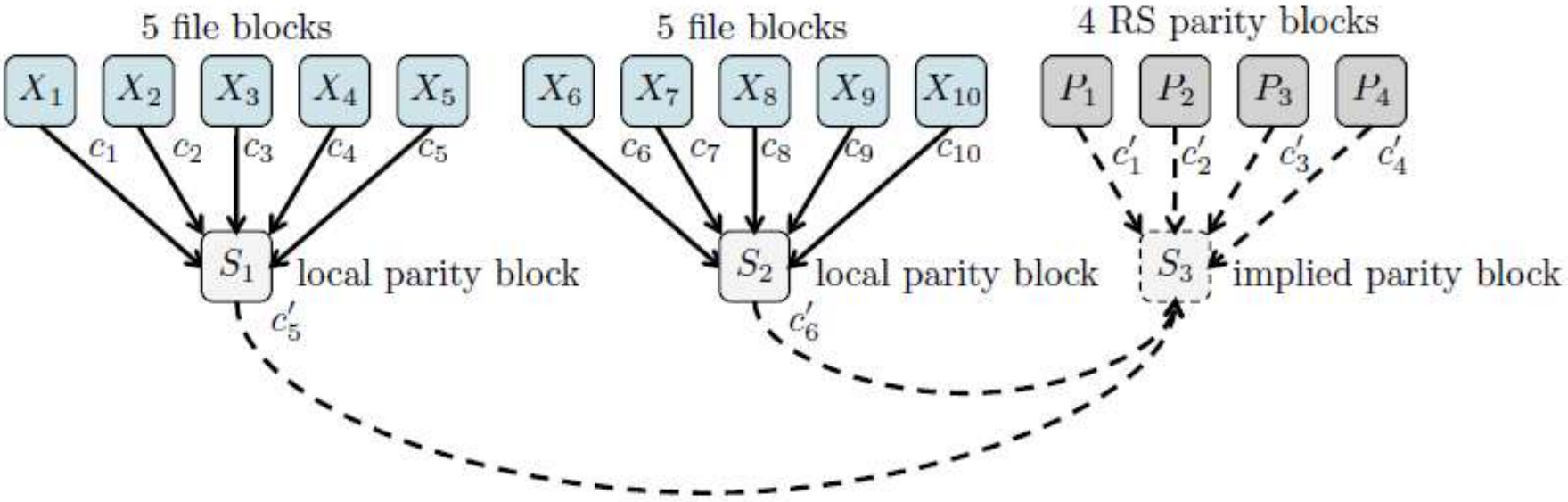
- Optimal Repair



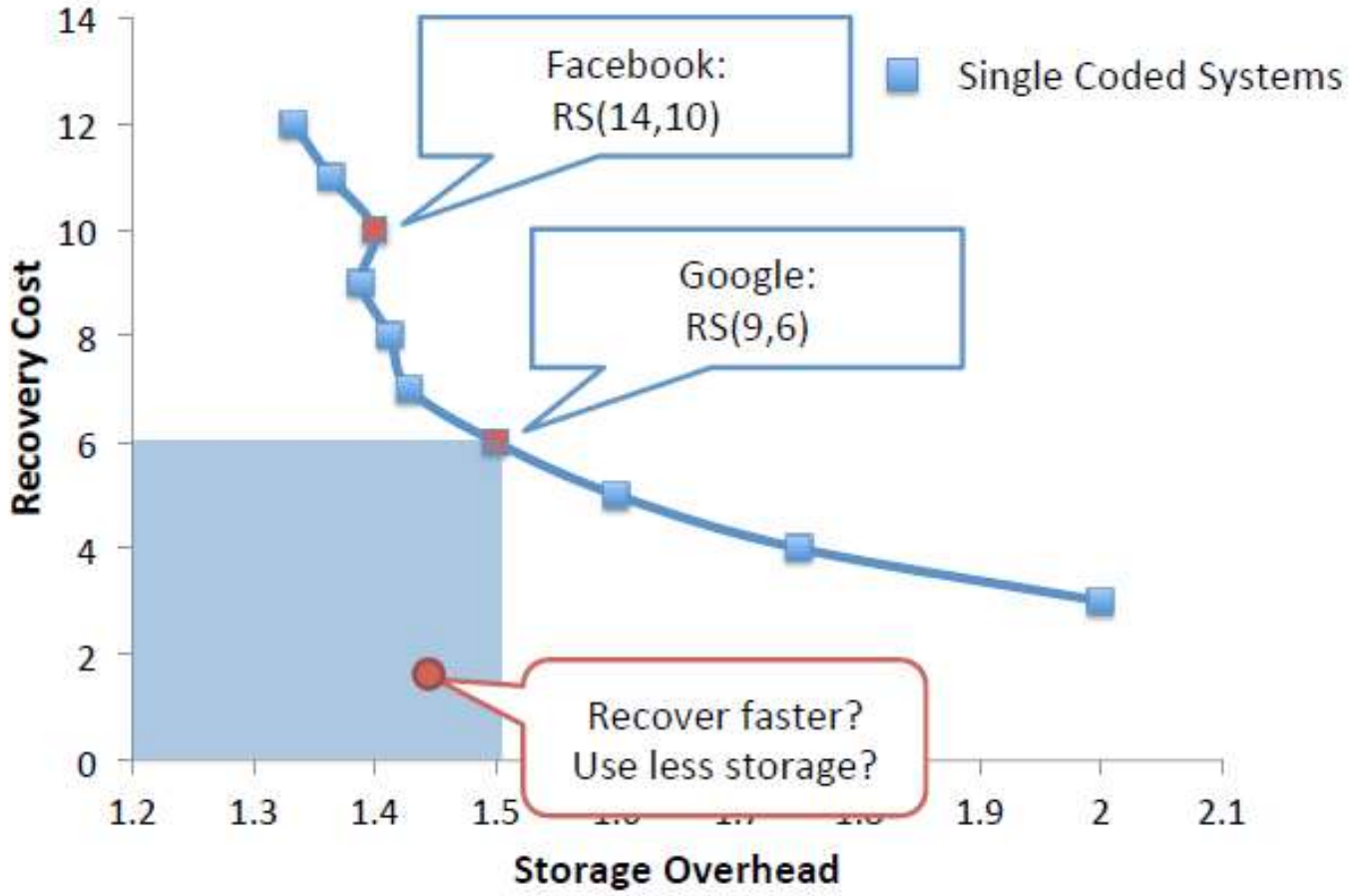
Regenerating Codes (4)
Analysis -- Regenerating vs RS

Single Failure Repair of 6 + 6 MDS Code	Reed-Solomon Coding	Regenerating Coding
# of nodes participating in repair	6	11
# of network transfers	6x	1.83x
# of disk I/Os	6x	up to 11x

Facebook Xorbas Hadoop Locally Repairable Codes

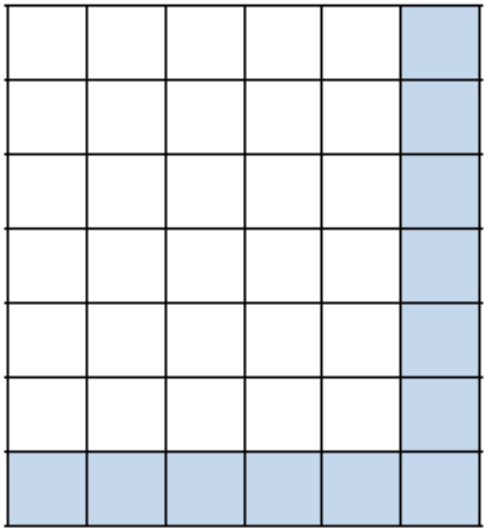


Combination of Two ECs (1)
Recovery Cost vs. Storage Overhead



Combination of Two ECs (2)
Fast Code and Compact Code

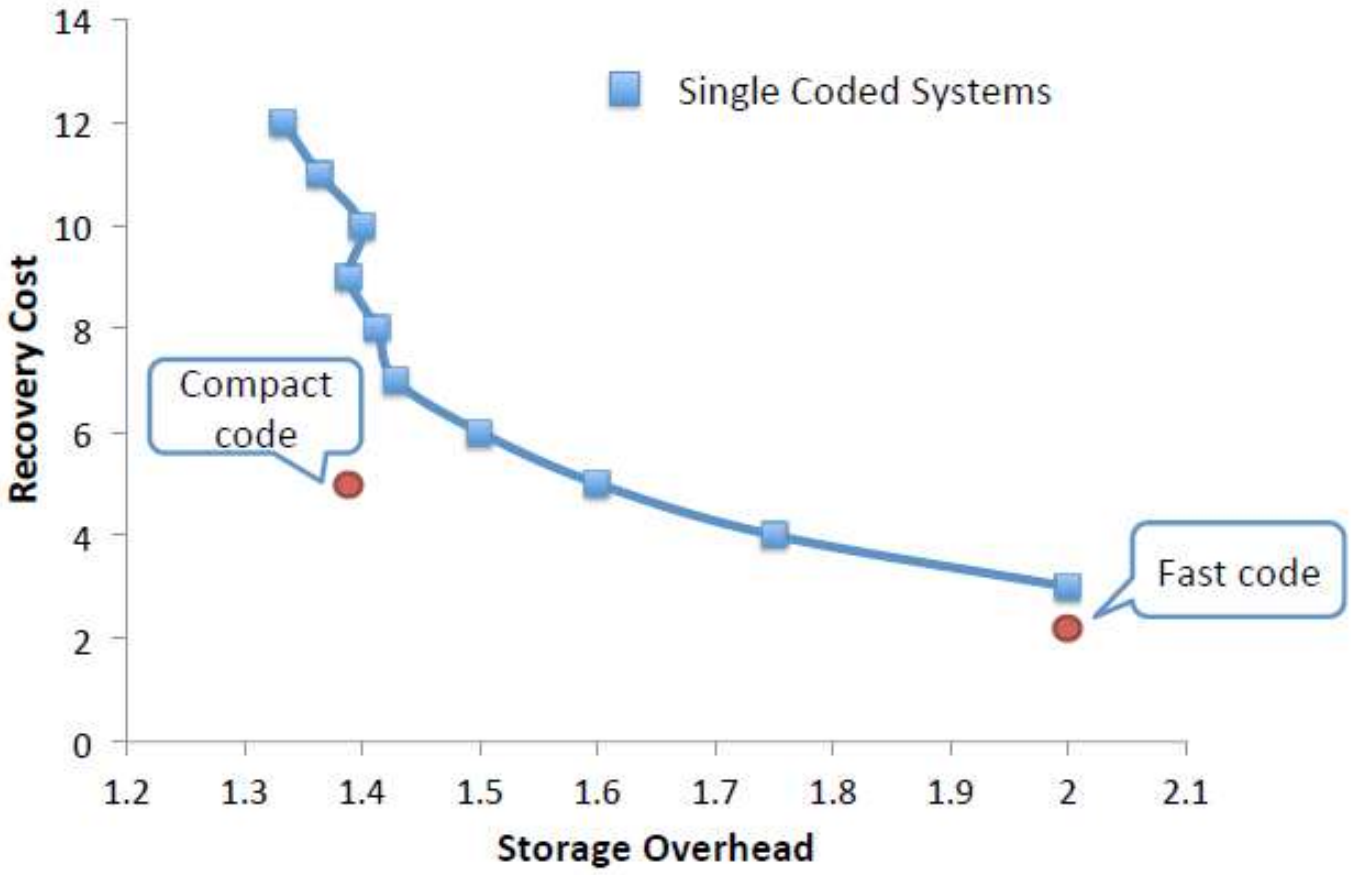
D1	D2	D3	D4	D5	P1
D6	D7	D8	D9	D10	P2
P3	P4	P5	P6	P7	P8



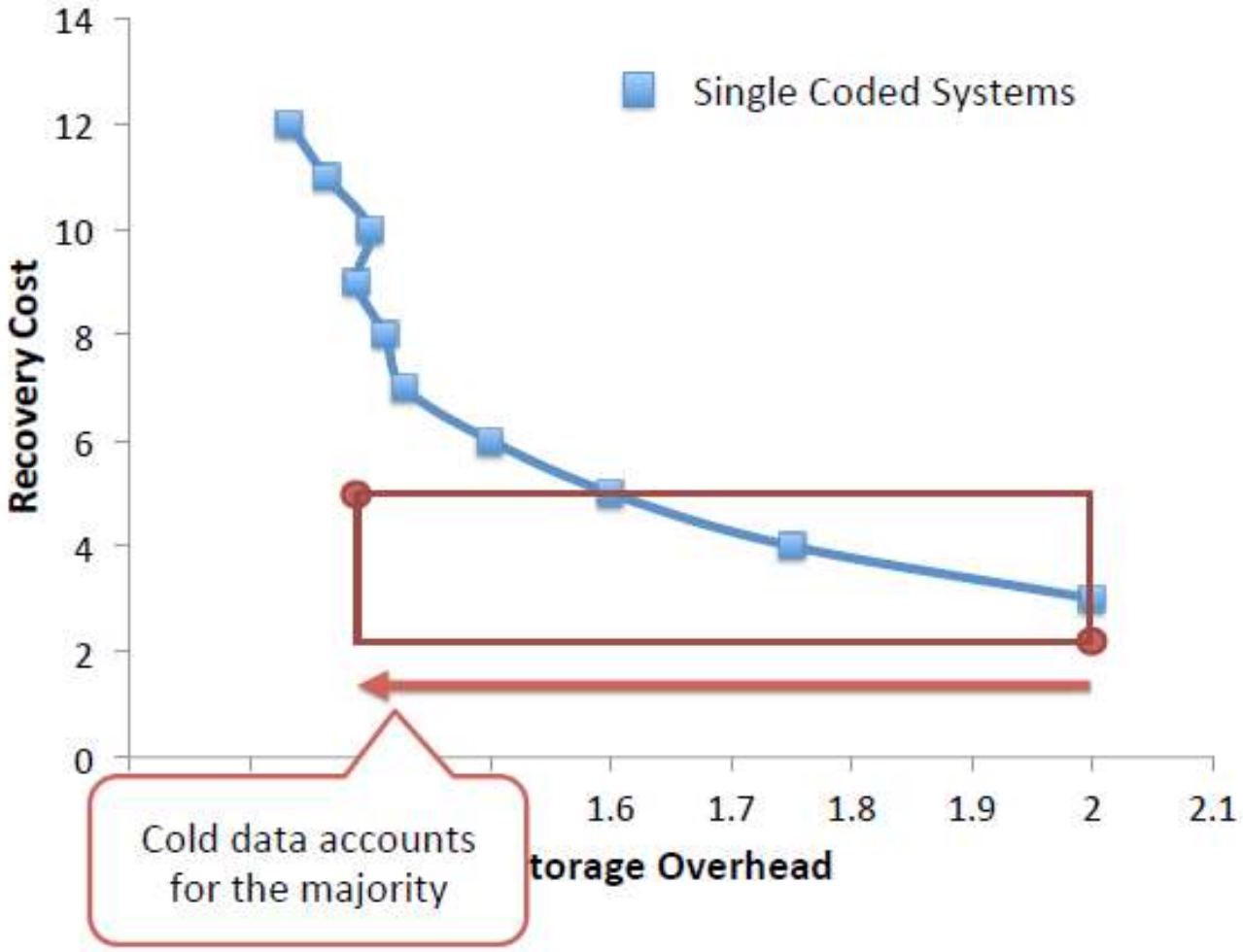
Fast code
(Product Code 2x5)
Recovery cost: 2
Storage overhead: 1.8x

Compact code
(Product Code 6x5)
Recovery cost: 5
Storage overhead: 1.4x

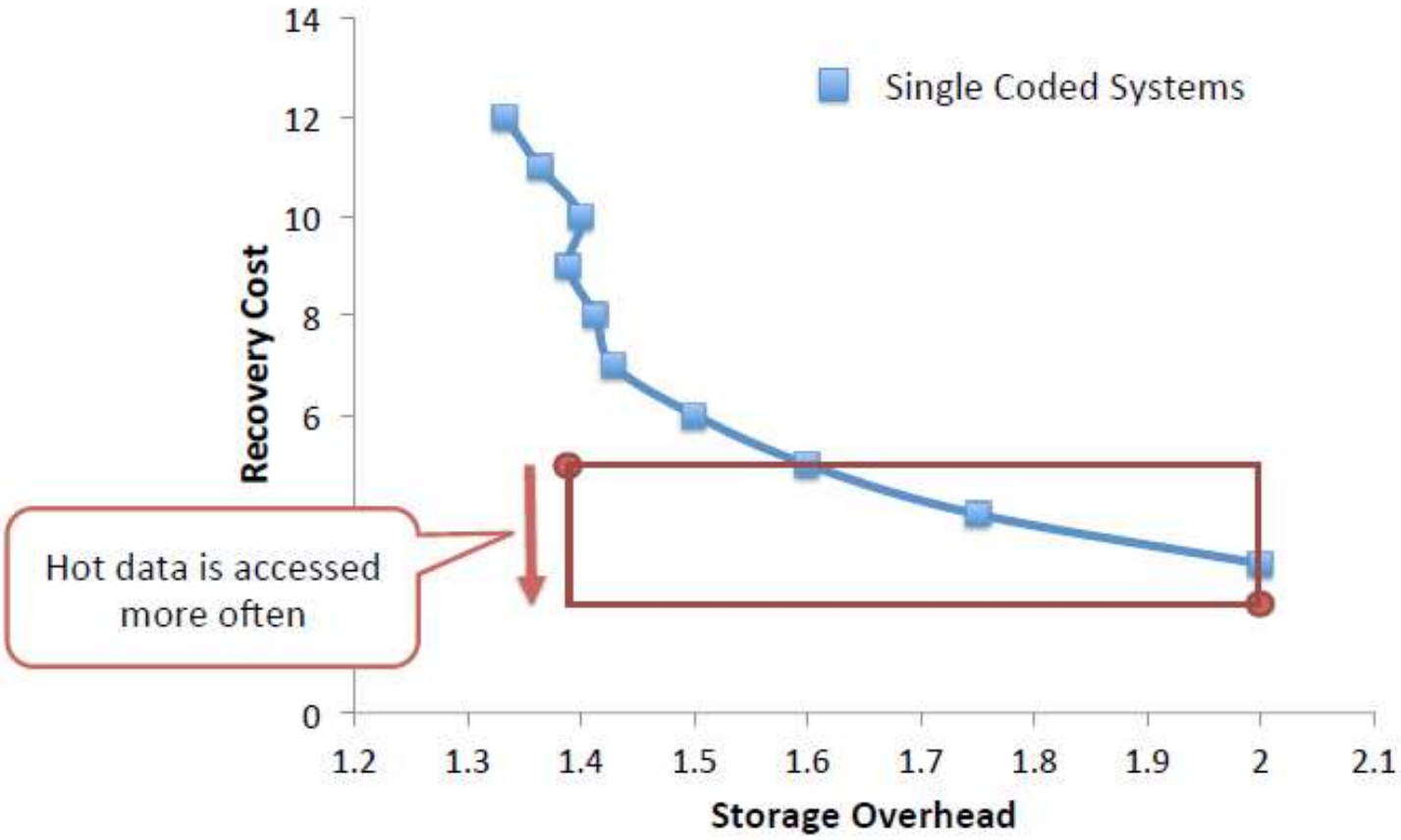
Combination of Two ECs (3) Analysis



Combination of Two ECs (4) Analysis

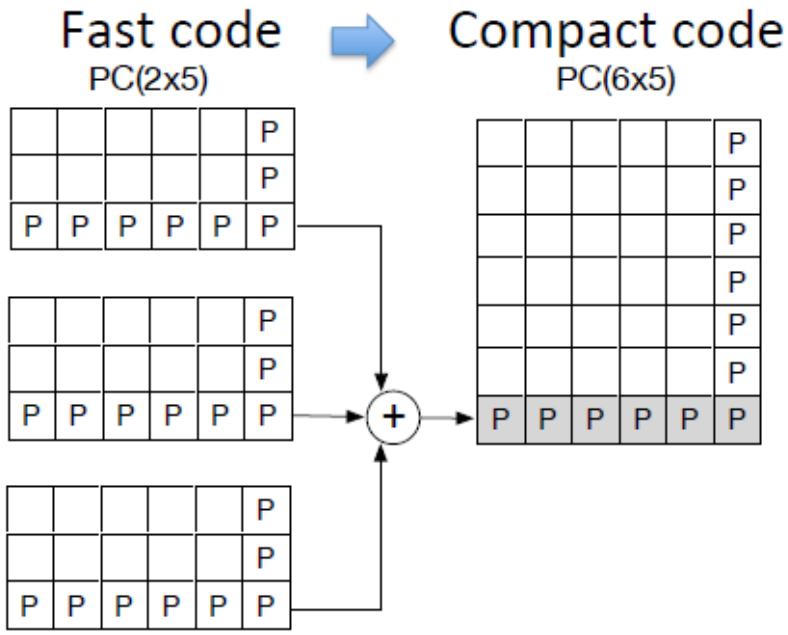


Combination of Two ECs (5) Analysis



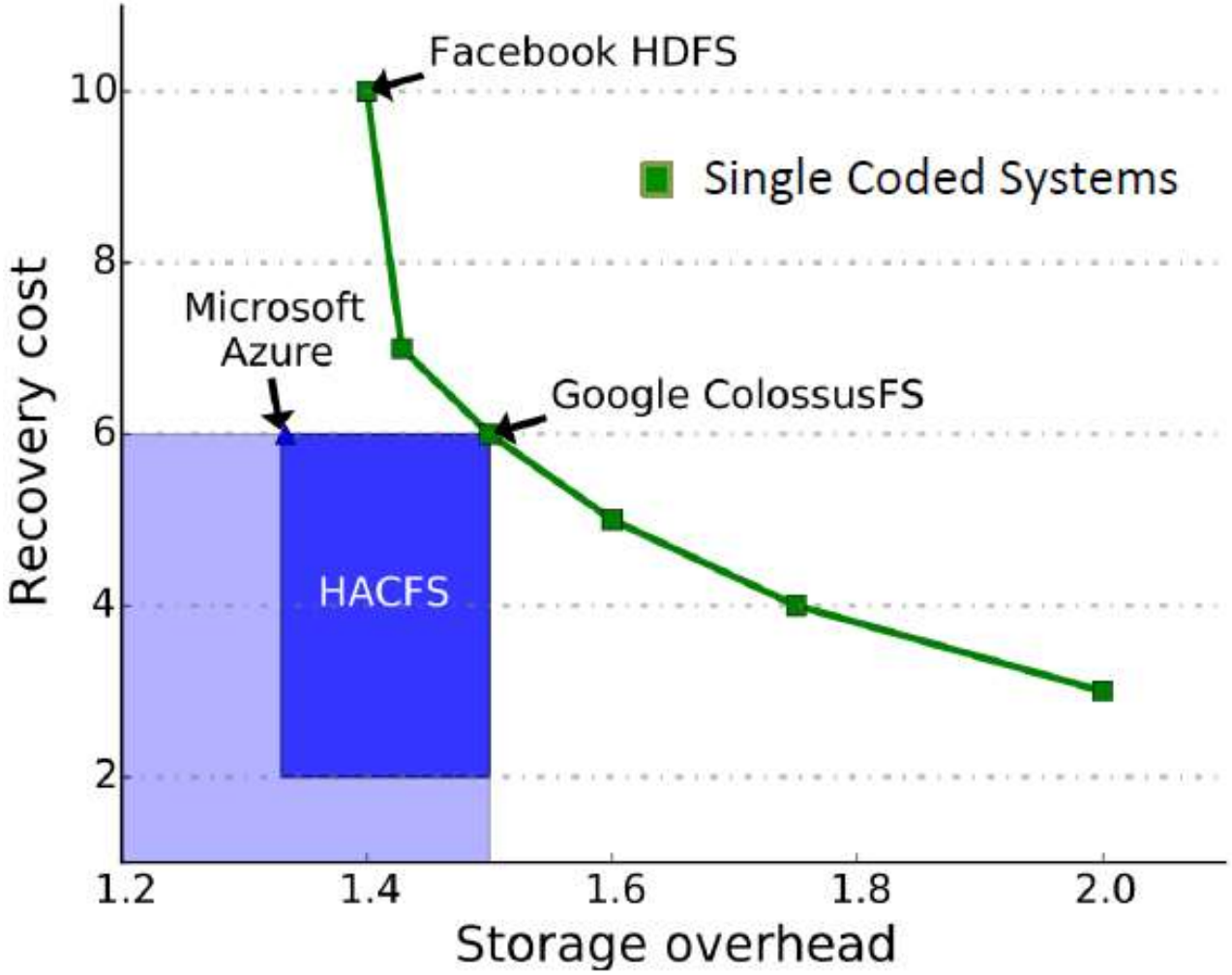
Combination of Two ECs (6) Conversion

- Horizontal parities require no re-computation
- Vertical parities require no data block transfer
- All parity updates can be done in parallel and in a distributed manner



Parity-only Conversion

Combination of Two ECs (7) Results



4 **Project 1**

Erasure Code in Hadoop (1)

- Implement an erasure code into Hadoop system
- Hadoop Version: 2.7 or higher
- Erasure Code: you can select one, but not RS
- Test the storage efficiency of your proposed code
- Report and Source Code are required
- Source Code should be checked by TA
- Deadline: June 30th

Erasure Code in Hadoop (2)

- References
- Jerasure

  http://web.eecs.utk.edu/~plank/plank/www/software.html
- HDFS-Xorbas

  http://smahesh.com/HadoopUSC/

Thank you!