Cloud Computing: Storage as a Service

Vijay Dialani, PhD
Boise State University
vijaydialani@boisestate.edu

Storage

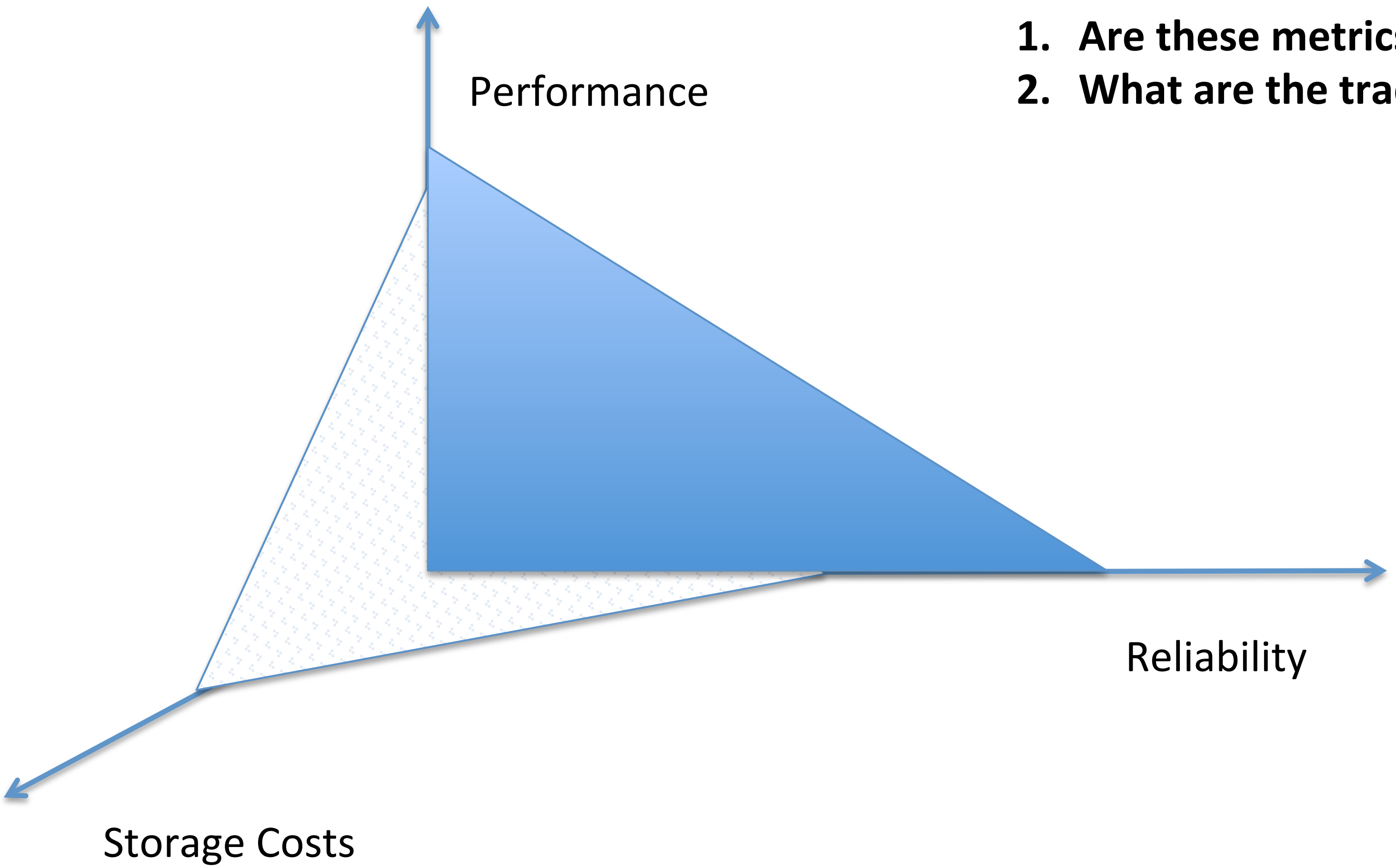
- It is one of the easiest resource to monitor and virtualize
  - Units of measures are well understood (bytes, KB, MB, GB, TB ….)
  - Easy to name (URI) and organize (File Collections)
  - Easy to monitor
  - Easy to partition and share between users

- Is a non-volatile resource

- Challenges
  - How to make it reliable in a system that has failing components?
  - How to make it scale and attain high throughput?

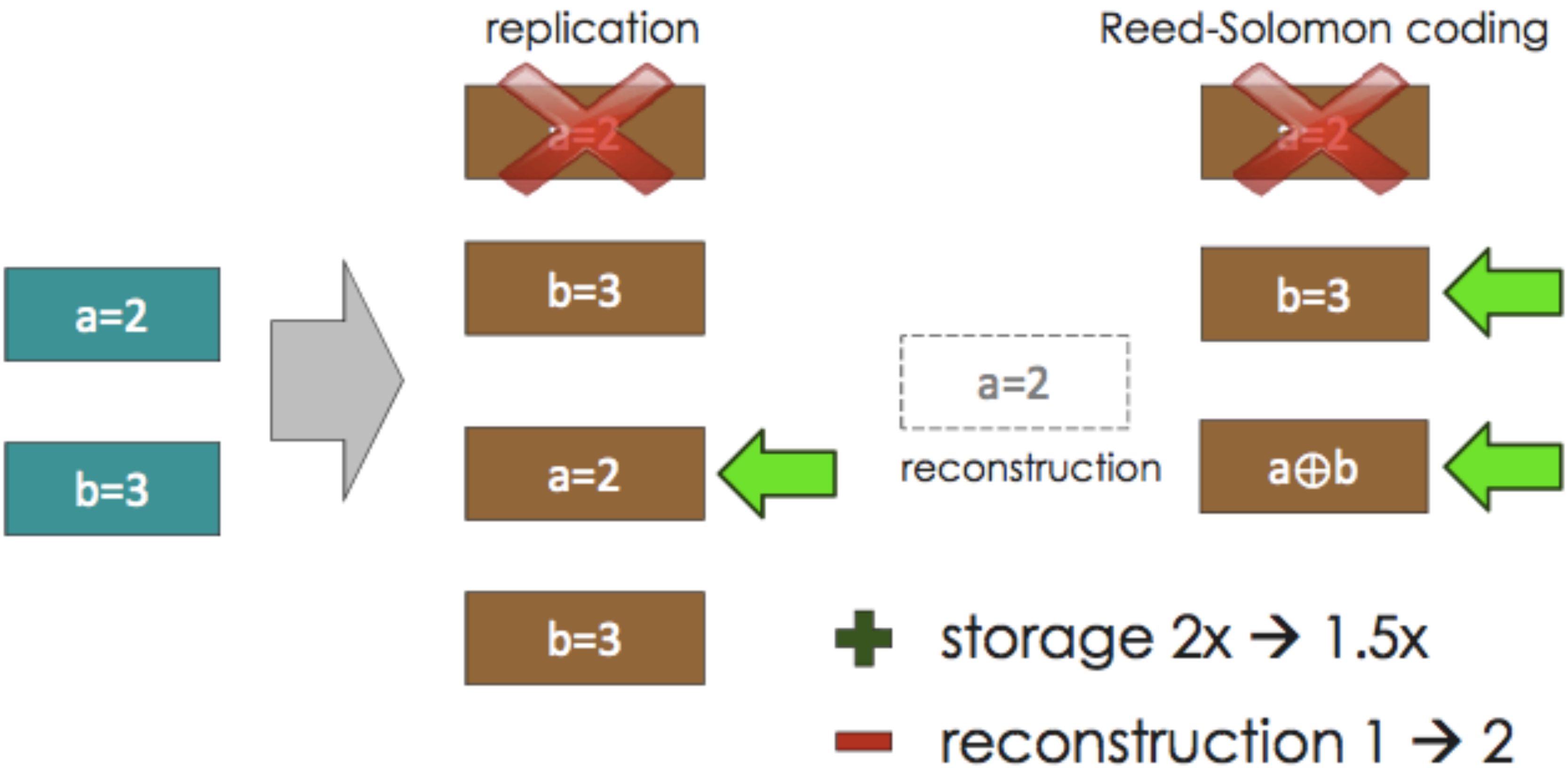
As of 2012, S3 storage service stored a total of 1.3 trillion objects and handles over 830,000 requests per second.

Three dimensions of Cloud Storage



1. **Are these metrics tunable?**
2. **What are the trade-offs?**

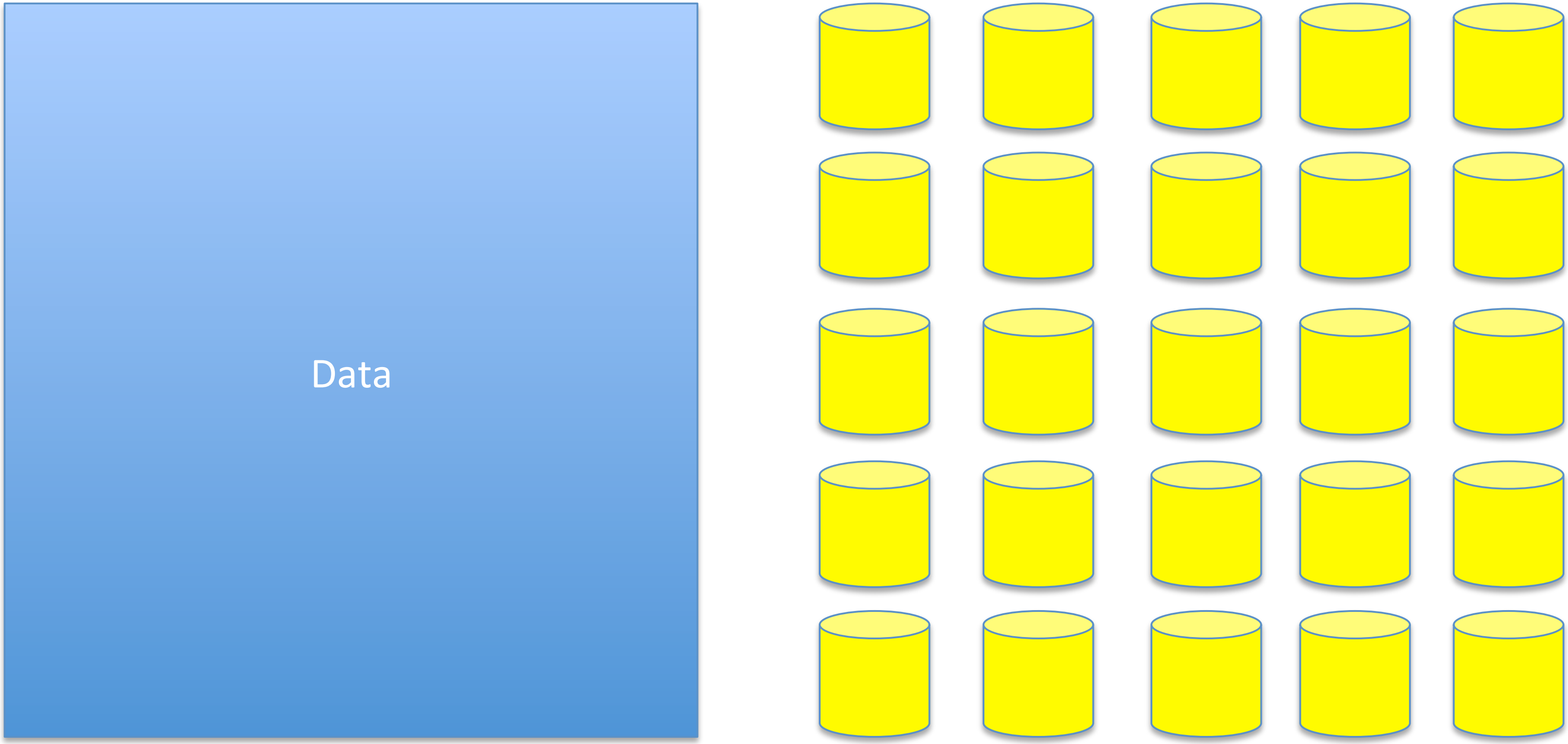
Why pure replication does not scale?



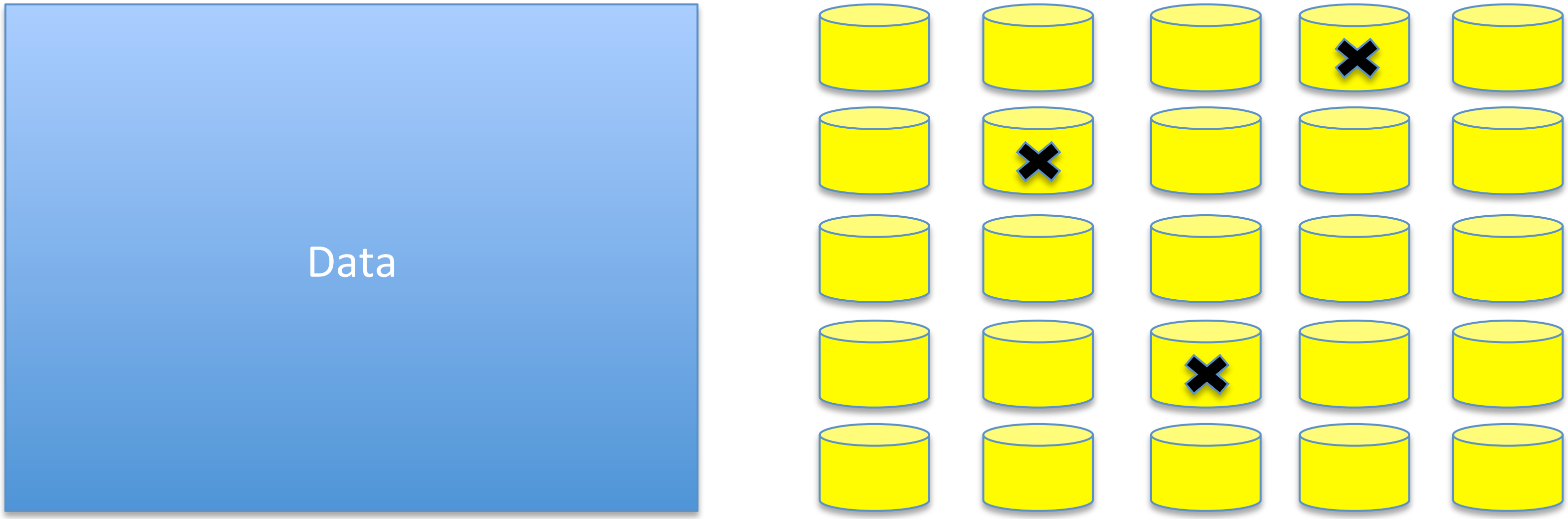
What is the cost of 0.5x of a petabyte? In Oct 2014, about $30K in 1TB disks

Typical data sizes for cloud = Exabytes, potential savings = **$30M**

Storage: Lets look at the basics



Storage: Basics

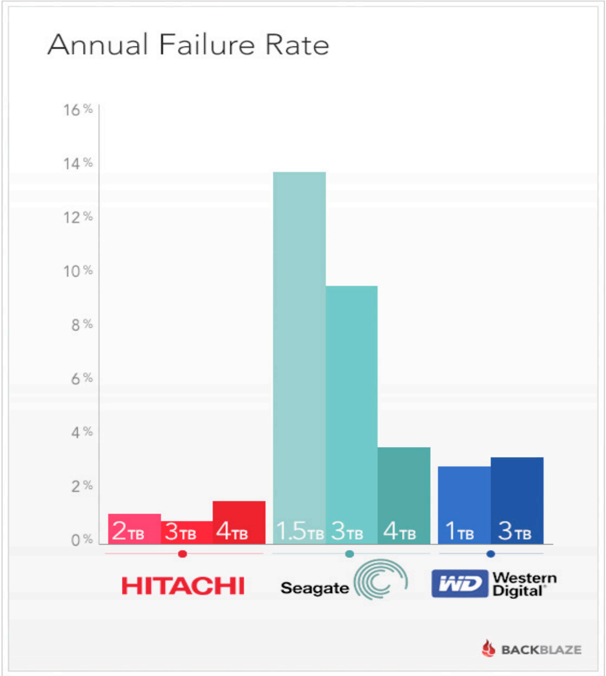


Hardware will fail.

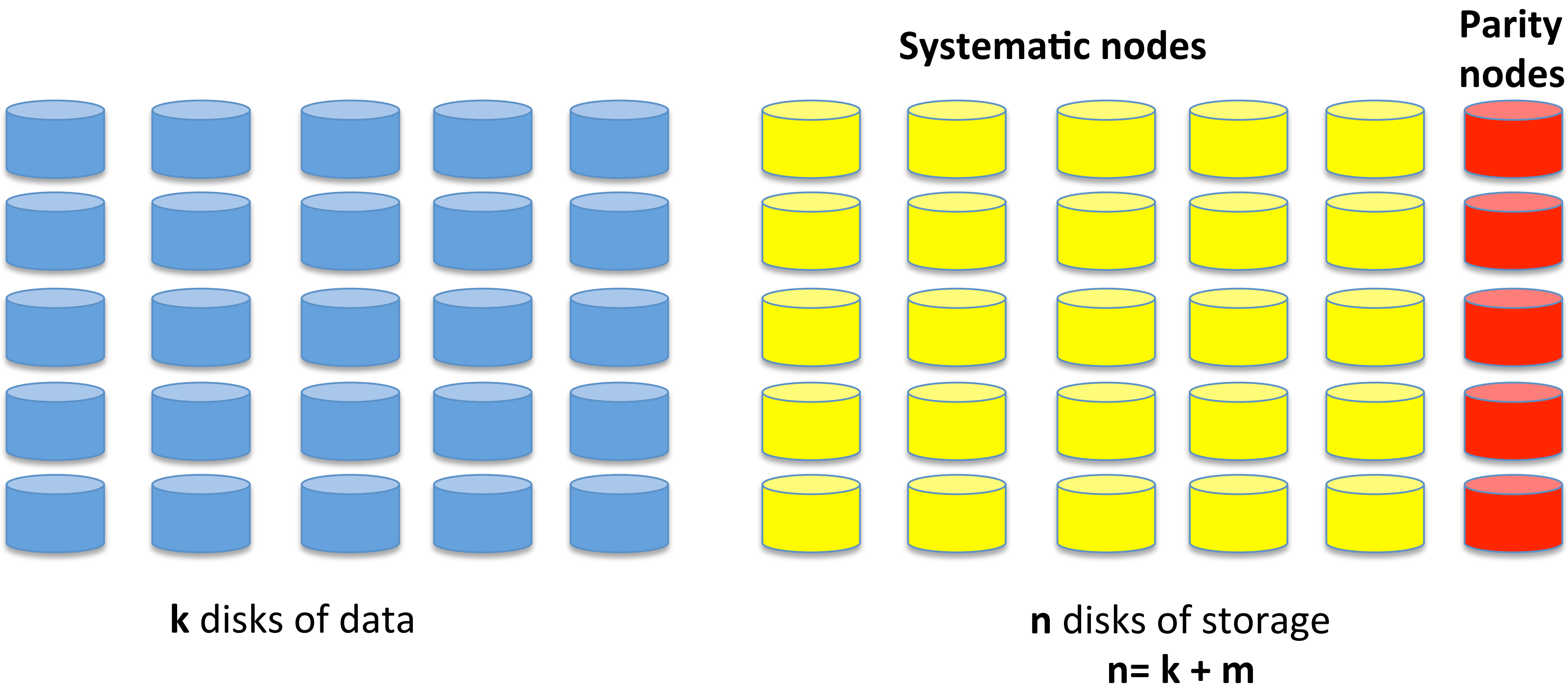
With a 2% failure rate, annually a 1PB data store created from 1TB hard disks will loose 20 disks a year.

For 100PB failure rate is 2000 discs/year.

This does not cover the End of Life replacement for all the discs every 3 years.



Simple Recovery using Parity Codes



- Allocate some of the disks for storing the parity data
- This metadata can be used for recovery in case of failures
- Parity data can aide detection and recovery

What is parity?

A parity bit, or check bit is a bit added to the end of a string of binary code that indicates whether the number of bits in the string with the value one is even or odd. Parity bits are used as the simplest form of error detecting code.

7 bits of data	(count of 1 bits)	8 bits including parity	
		even	odd
0000000	0	00000000	00000001
1010001	3	10100011	10100010
1101001	4	11010010	11010011
1111111	7	11111111	11111110

How does parity work?

Type of bit parity	Successful transmission scenario
Even parity	A wants to transmit: 1001 A computes parity bit value: 1+0+0+1 (mod 2) = 0 A adds parity bit and sends: 10010 B receives: 10010 B computes parity: 1+0+0+1+0 (mod 2) = 0 B reports correct transmission after observing expected even result.
Odd parity	A wants to transmit: 1001 A computes parity bit value: 1+0+0+1 + **1** (mod 2) = 1 A adds parity bit and sends: 10011 B receives: 10011 B computes overall parity: 1+0+0+1+1 (mod 2) = 1 B reports correct transmission after observing expected result.

This is XOR operation

http://en.wikipedia.org/wiki/Parity_bit

How will the parity be used for Recovery?

Assume that we have four disks each storing the data and the fifth disks stores the even parity bit. Assuming that disk 3 has failed, recreate the contents of the failed disk from the remaining disks and parity disk.

Disk 1:	0xFE	0xFF	0x10	0x35
Disk 2:	0xAA	0x33	0x1A	0x14
Disk 3:	??	??	??	??
Disk 4:	0x45	0x12	0x11	0x13
Parity:	0x34	0xFF	0x2A	0x26

Partitioning of Parity Blocks

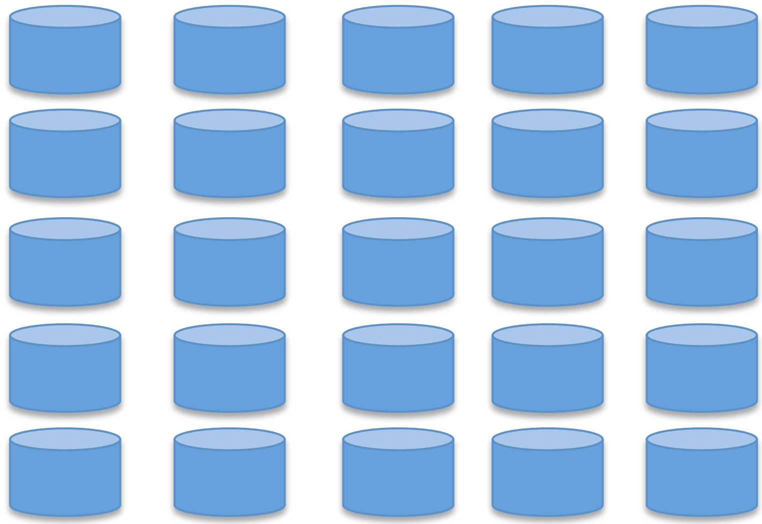
Disk 1:	0xFE	**0xFF**	0x10	0x35
Disk 2:	0xAA	0x33	**0x1A**	0x14
Disk 3:	**0x25**	0x21	0x31	0x14
Disk 4:	0x45	0x12	0x11	**0x13**
Disk 5:	0x34	0xFF	0x2A	0x26

It is possible to stripe the data and the parity information in such a way that both reside on the same disc.

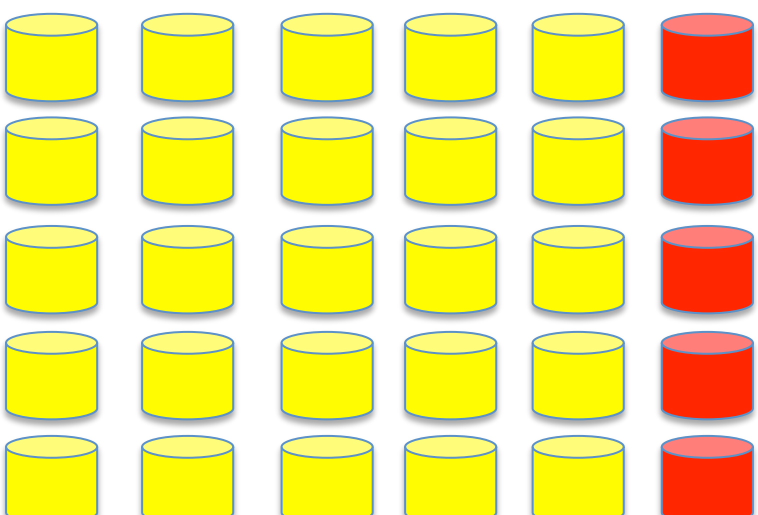
Identification of parity blocks can be based on the location of the block or the metadata associated with the block

Horizontal and Vertical Partitioning

- Total space required by two schemes is the same
- Vertical partitioning allows increased throughput



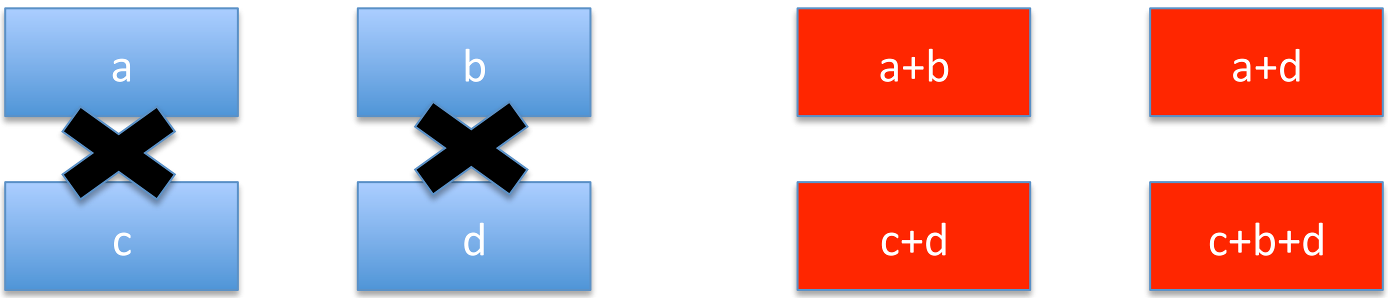
Horizontal Partitioning

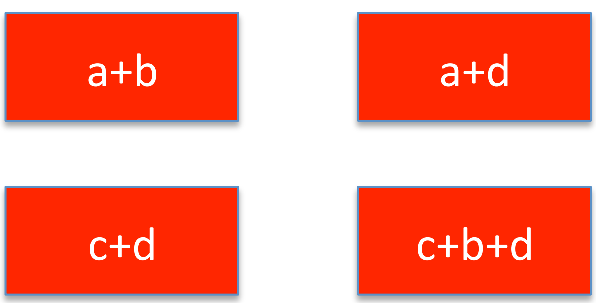


Vertical Partitioning

Recovering from multiple failures: Maximum Distance Separable Codes

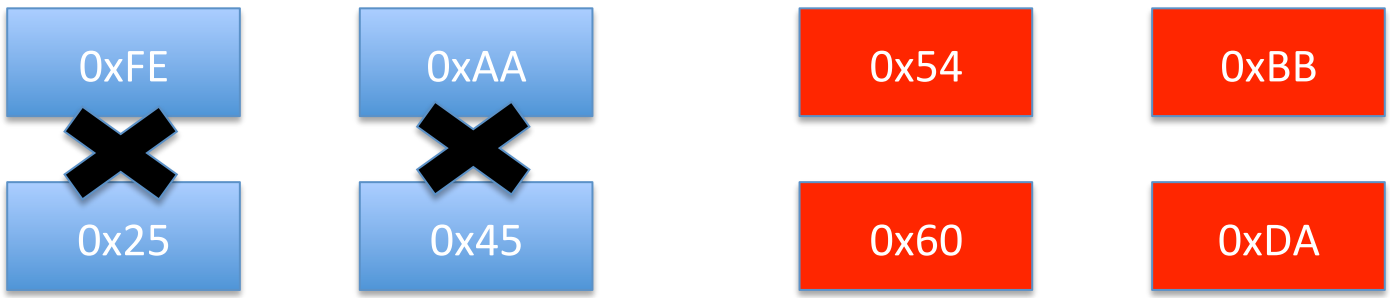
r redundancies → correct **r** erasures

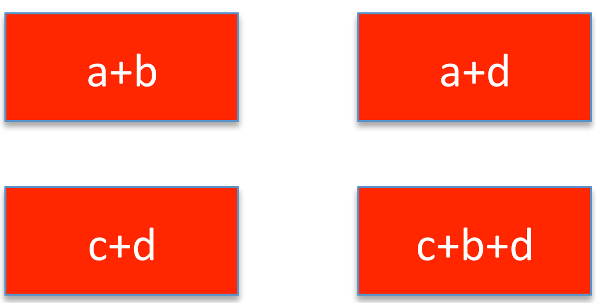




1^{st} rebuilt block = (c+b+d)-(c+d) = b
2^{nd} rebuilt block = (a+b)-b = a
3^{rd} rebuilt block = (a+d) – a = d
4^{th} rebuilt block = (c+d)-d =c

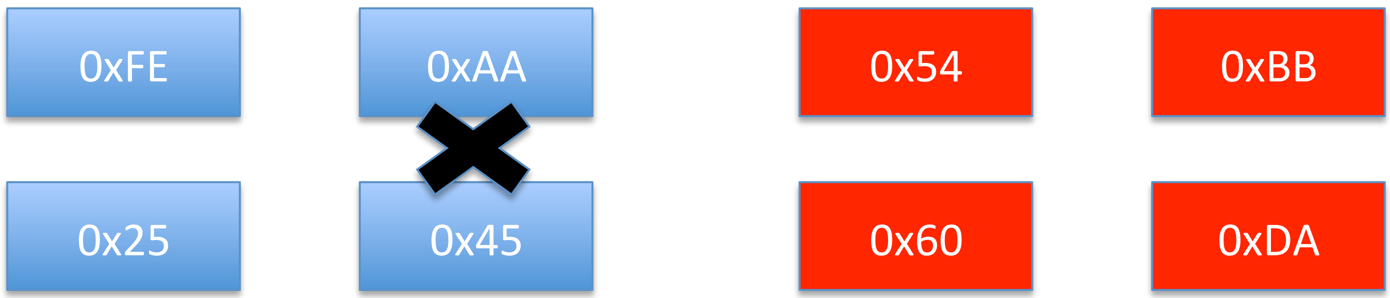
Example for calculating the codes

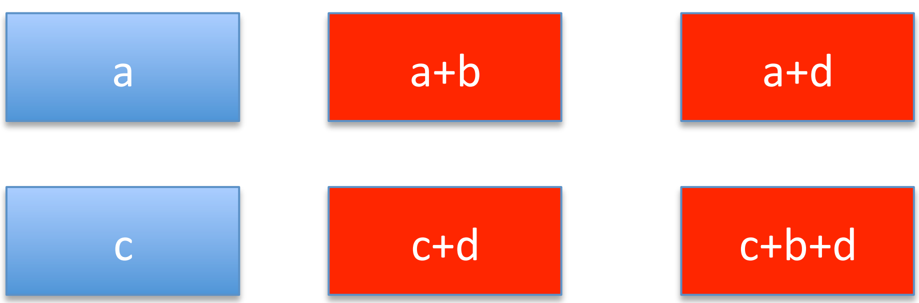




1st rebuilt block = (c+b+d)-(c+d) = b= 0xAA
2nd rebuilt block = (a+b)-b = a =0xFE
3rd rebuilt block = (a+d) – a = d = 0x45
4th rebuilt block = (c+d)-d = c = 0x25

Example for calculating the codes

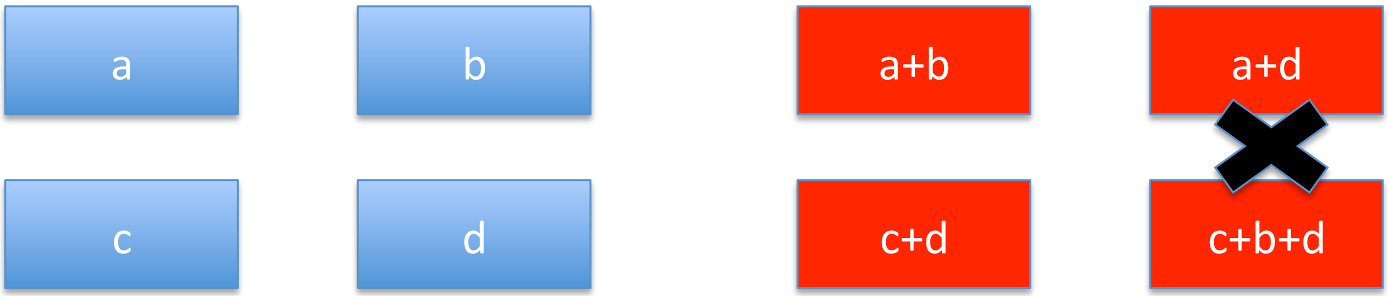




1st rebuilt block = (a+b) - a= b= 0xAA
2nd rebuilt block = (c+d)-d = c = 0x25

Example for calculating the codes

r redundancies → correct **r** erasures



How many network accesses are required for regeneration, when a parity node fails?

Cost of Reconstruction?

When parity node fails

Access: # of Access/ # Remaining = 4/6 = 0.66

Bandwidth: # transmission / # Remaining = 3/6 = ½

When systematic node fails

Access: # of Access/ # Remaining = 3/6 = ½

Bandwidth: # transmission / # Remaining = 3/6 = ½

ZigZag Codes

Zigzag Codes: MDS Array Codes With Optimal Rebuilding

```
@ARTICLE{6352912,
author={Tamo, I and Zhiying Wang and Bruck, J.},
journal={Information Theory, IEEE Transactions on},
title={Zigzag Codes: MDS Array Codes With Optimal Rebuilding},
year={2013},
month={March},
volume={59},
number={3},
pages={1597-1616},
}
```

ZigZag Codes

	Node-0	Node-1	Node-2	Row sum	Parity-2
0	a0	b0	c0	a0+b0+c0	a0+b1+c3
1	a1	b1	c1	a1+b1+c1	a1+b3+c0
2	a2	b2	c2	a2+b2+c2	a2+b2+c1
3	a3	b3	c3	a3+b3+c3	a3+b0+c2

ZigZag Codes

	Node-0	Node-1	Node-2	Row sum	Parity-2
0	0	3	1	a0+b0+c0	a0+b1+c3
1	1	0	2	a1+b1+c1	a1+b3+c0
2	2	2	3	a2+b2+c2	a2+b2+c1
3	3	1	0	a3+b3+c3	a3+b0+c2

	Node-0	Node-1	Node-2	Row sum	ZigZag sum
0	0	3	1	r0	z0
1	1	0	2	r1	z1
2	2	2	3	r2	z2
3	3	1	0	r3	z3

ZigZag Codes

	Node-0	Node-1	Node-2	Row sum	ZigZag sum
0	0	3	1	r0	z0
1		0	2	r1	z1
2		2	3	r2	z2
3		1	0	r3	z3

Requires 12 accesses to recreate the dataset on node0

ZigZag Codes

	Node-0	Node-1	Node-2	Row sum	ZigZag sum
0	0	3	1	r0	z0
1	[illegible]	0	2	r1	z1
2	[illegible]	2	3	r2	z2
3	[illegible]	1	0	r3	z3

1. Naïve read requires 12 accesses to recreate the dataset on node0

2. Smart read requires 8 accesses

ZigZag Codes

	Node-0	Node-1	Node-2	Row sum	ZigZag sum
0	0	3	1	r0	z0
1	1		2	r1	z1
2	2		3	r2	z2
3	3	[illegible]	0	r3	z3

1. Naïve read requires 12 accesses to recreate the dataset on node0

2. Smart read requires 9 accesses

ZigZag Codes

Permutations

1. Four rows three columns
2. Column 0 : identity permutation
3. Column I : Flip i^{th} bit of the index From msb

	Node-0	Node-1	Node-2	Row sum	ZigZag sum
0	0				
1	1				
2	2				
3	3				

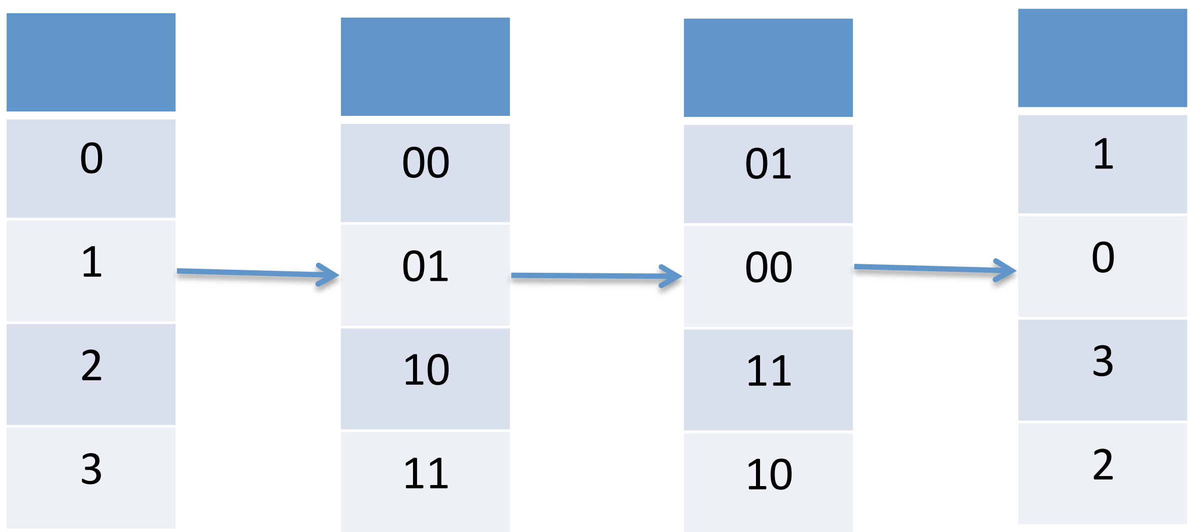
0	00	10	2
1	01	11	3
2	10	00	0
3	11	01	1

ZigZag Codes

Permutations

1. Four rows three columns
2. Column 0 : identity permutation
3. Column I : Flip i^{th} bit of the index From msb

	Node-0	Node-1	Node-2	Row sum	ZigZag sum
0	0	2			
1	1	3			
2	2	0			
3	3	1			

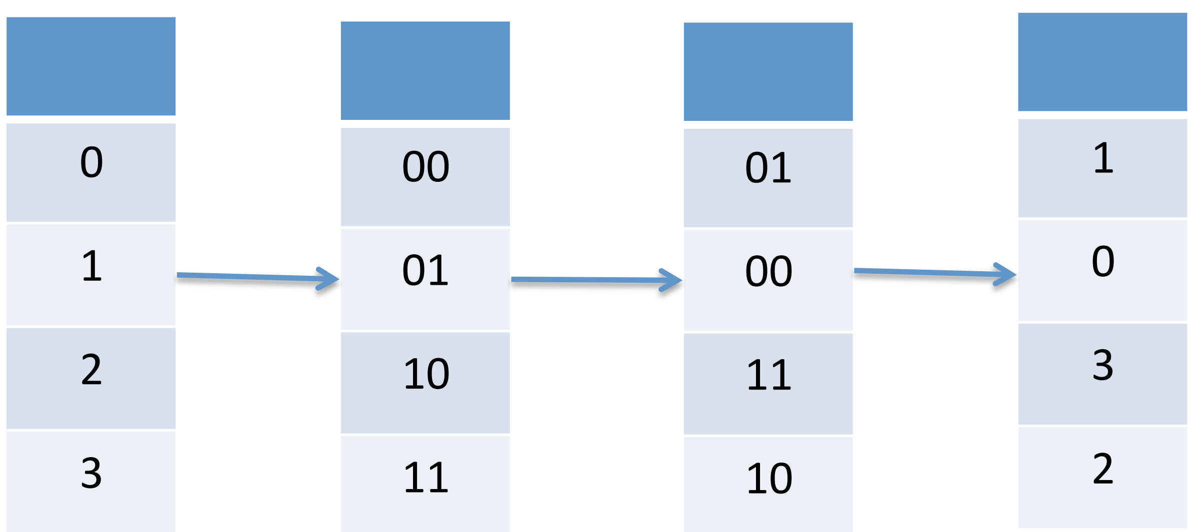


ZigZag Codes

Permutations

1. Four rows three columns
2. Column 0 : identity permutation
3. Column I : Flip i^{th} bit of the index From msb

	Node-0	Node-1	Node-2	Row sum	ZigZag sum
0	0	2	1		
1	1	3	0		
2	2	0	3		
3	3	1	2		



ZigZag Codes

	Node-0	Node-1	Node-2	Row sum	ZigZag sum
0	m	a	w	m+a+w	2m+c+2z
1	n	b	x	n+b+x	2n+d+y
2	p	c	y	p+c+y	p+a+x
3	q	d	z	q+d+z	q+b+2w

Access v/s Bandwidth
For M rows

	Optimal Access	Optimal Bandwidth
Optimal Update	M+1	M+1
Non-optimal update	$3m<k<2^{2m}$ **(More)**	2M **(Less)**

References

Extending striping scheme to recover from failure of multiple devices.
https://www.usenix.org/legacy/publications/library/proceedings/fast04/tech/corbett/corbett_html/