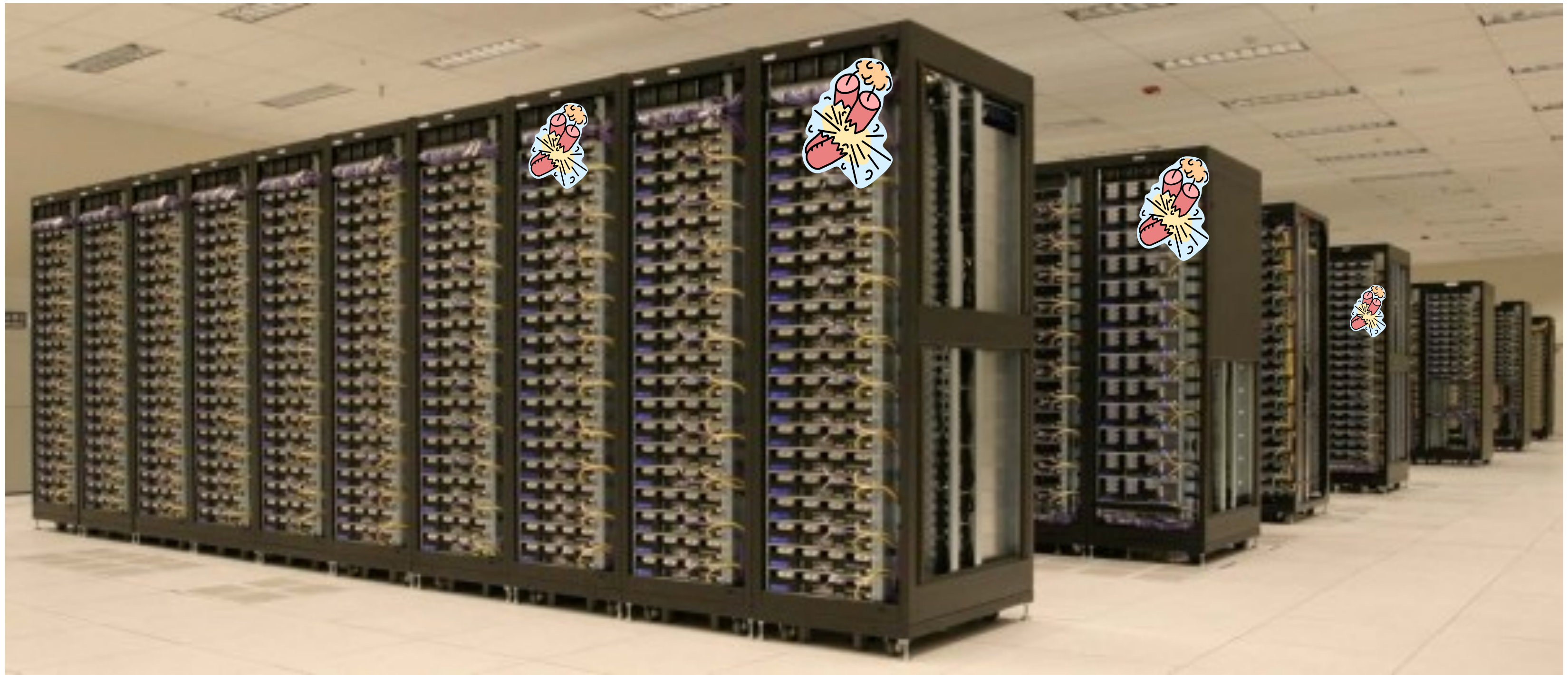
CSCI 381/780
Cloud Computing

Erasure Coding (in the Cloud)

Jun Li
Queens College

Large Scale Storage Systems

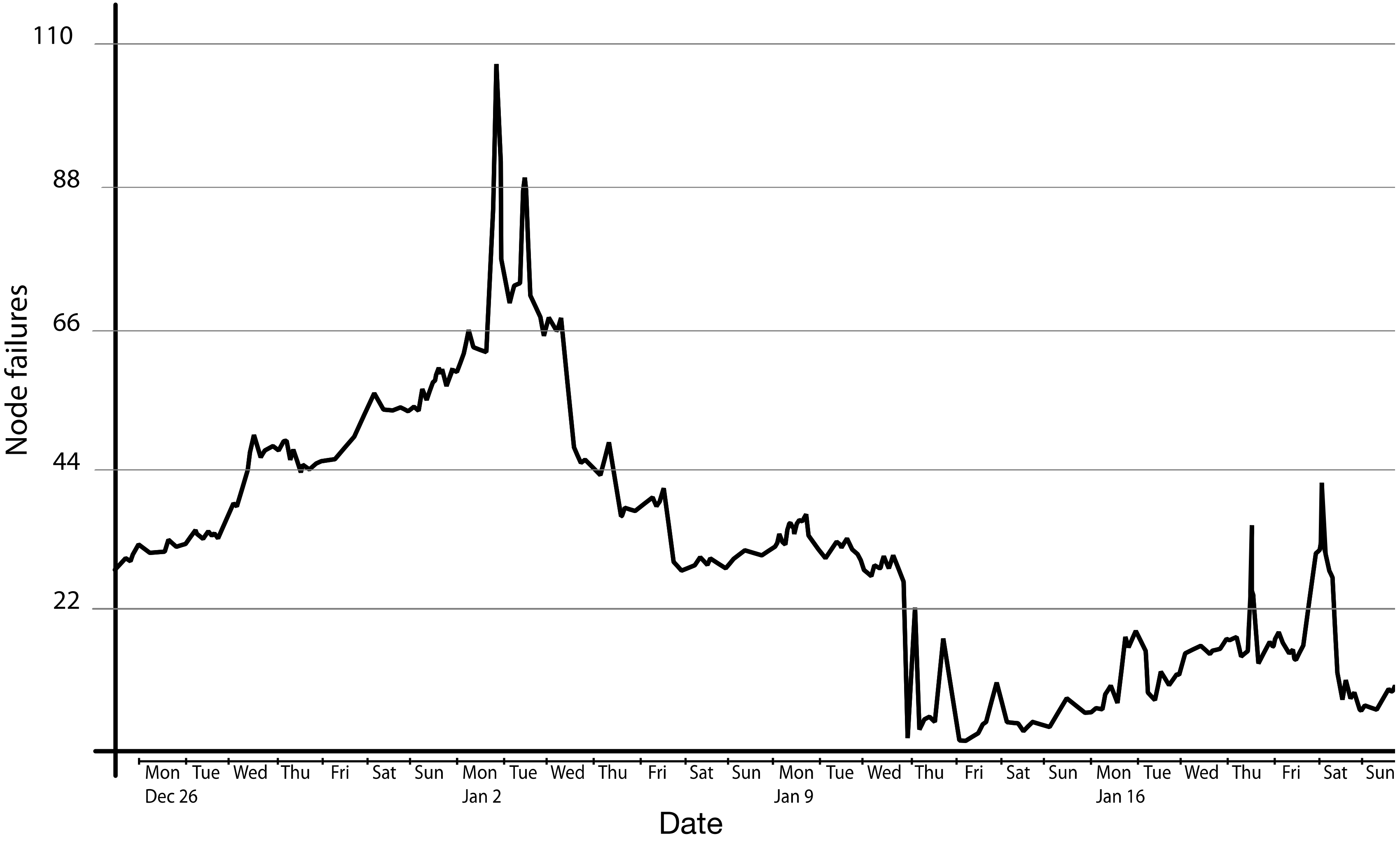
- Big Data Players: Facebook, Amazon, Google,…



Cluster of machines running Hadoop at Yahoo! (Source: Yahoo!)

- Failures are the **norm**

Node failures at Facebook



XORing Elephants: Novel Erasure Codes for Big Data M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, VLDB 2013

Things Fail, Let's Not Lose Data

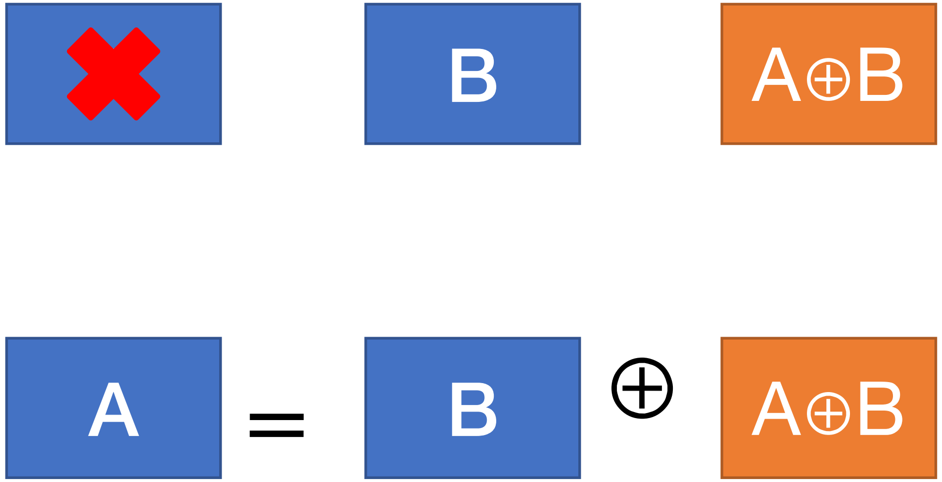
- Replication
  - Store multiple copies of the data
  - Simple and very commonly used!
  - But, requires a lot of extra storage

- Erasure coding
  - Store extra information we can use to recover the data
  - Fault tolerance with less storage overhead

Erasure Codes, a simple example w/ XOR

Erasure Codes, a simple example w/ XOR

Erasure Codes, a simple example w/ XOR



Reed-Solomon Codes (1960)

- N data blocks
- K coding blocks
- M = N+K total blocks

- Recover any block from any N other blocks!

- Tolerates up to K simultaneous failures

- Works for any N and K (within reason)

Reed-Solomon Code Notation

- N data blocks
- K coding blocks
- M = N+K total blocks

- RS(N,K)
  - (10,4): 10 data blocks, 4 coding blocks
    - f4 uses this, FB HDFS for data warehouse does too

- Will also see [M, N] notation sometimes
  - [14,10]: 14 total blocks, 10 data blocks, (4 coding blocks)

Reed-Solomon Codes, How They Work

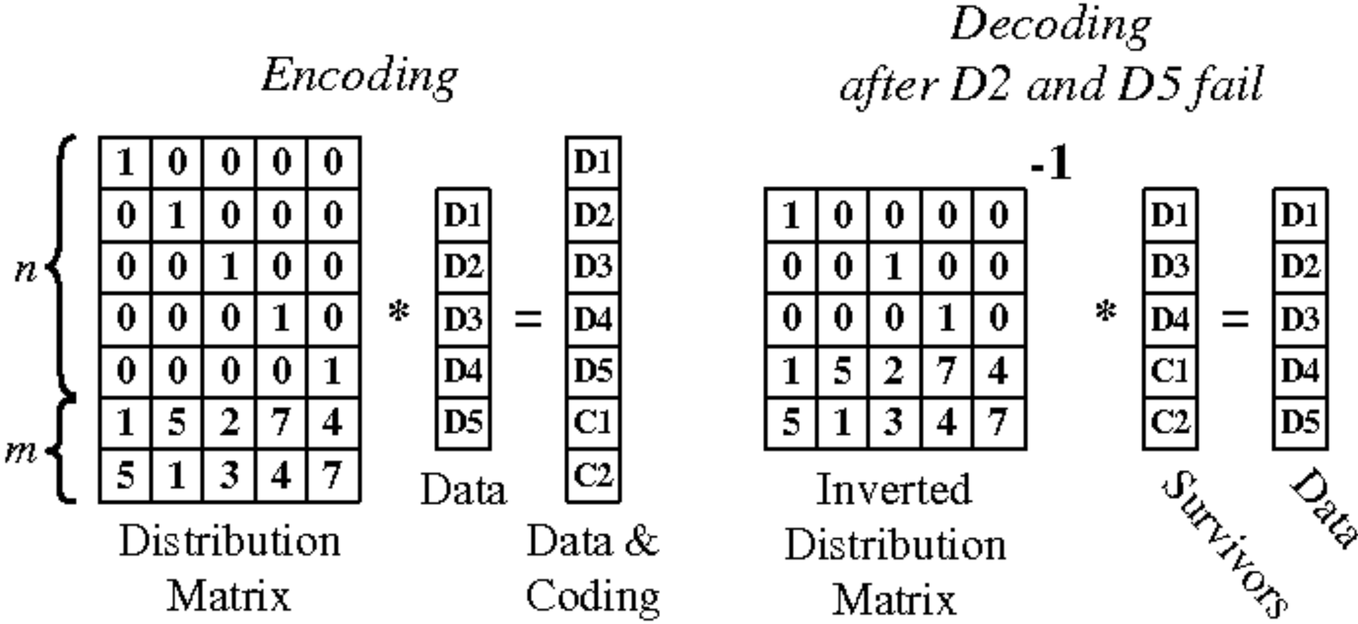
- Galois field arithmetic is the secret sauce

	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

Addition

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

Multiplication



- See “J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Software—Practice & Experience 27(9):995–1012, September 1997.”

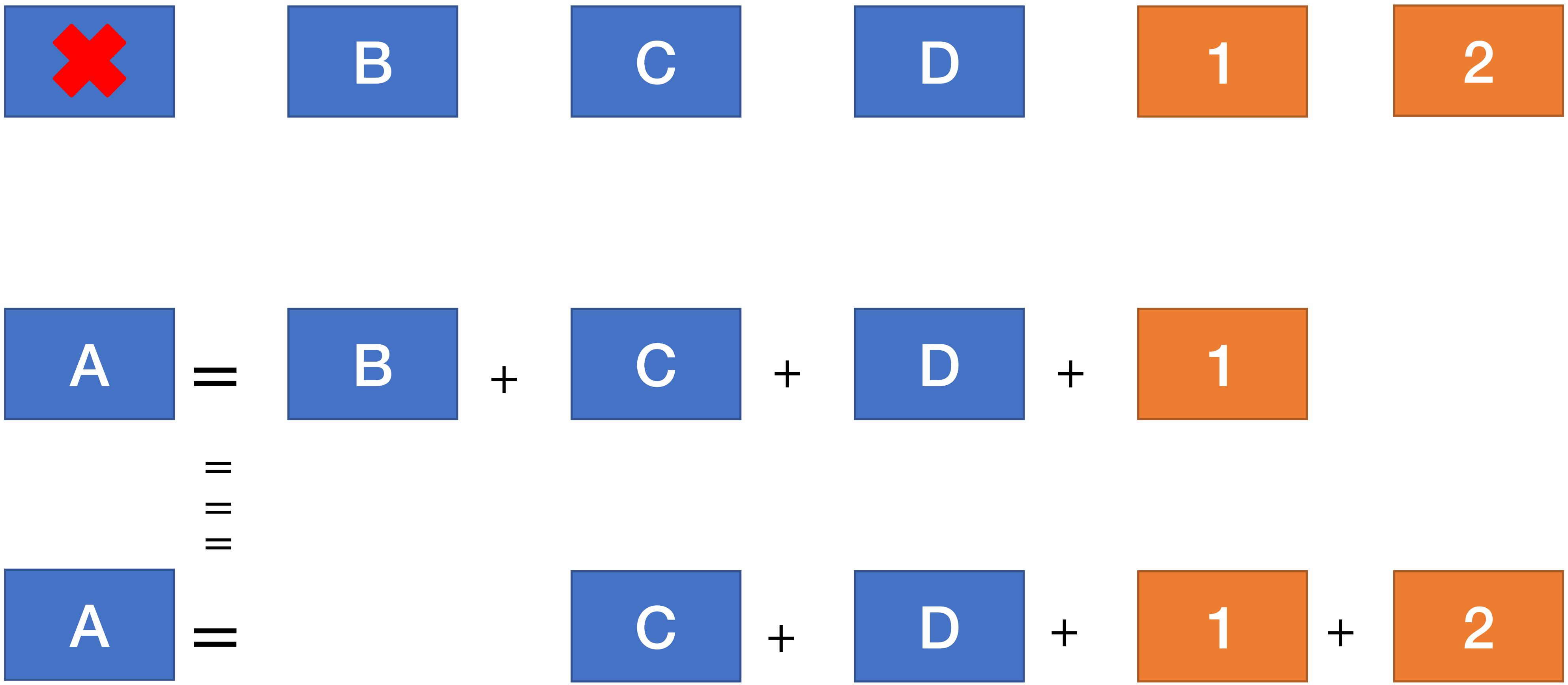
Reed-Solomon (4,2) Example

Reed-Solomon (4,2) Example

Reed-Solomon (4,2) Example

A = B + C + D + 1

Reed-Solomon (4,2) Example



Erasure Codes Save Storage

- Tolerating 2 failures
    - 3x replication = ___ storage overhead

Erasure Codes Save Storage

- Tolerating 2 failures
  - 3x replication = 3x storage overhead
  - RS(4,2) = ___ storage overhead

Erasure Codes Save Storage

- Tolerating 2 failures
  - 3x replication = 3x storage overhead
  - RS(4,2) = (4+2)/4 = 1.5x storage overhead

Erasure Codes Save Storage

- Tolerating 2 failures
    - 3x replication = 3x storage overhead
    - RS(4,2) = (4+2)/4 = 1.5x storage overhead

- Tolerating 4 failures
    - 5x replication = 5x storage overhead
    - RS(10,4) = ___ storage overhead

Erasure Codes Save Storage

- Tolerating 2 failures
  - 3x replication = 3x storage overhead
  - RS(4,2) = (4+2)/4 = 1.5x storage overhead

- Tolerating 4 failures
  - 5x replication = 5x storage overhead
  - RS(10,4) = (10+4)/10 = 1.4x storage overhead
  - RS(100,4) = ___ storage overhead

Erasure Codes Save Storage

- Tolerating 2 failures
  - 3x replication = 3x storage overhead
  - RS(4,2) = (4+2)/4 = 1.5x storage overhead
- Tolerating 4 failures
  - 5x replication = 5x storage overhead
  - RS(10,4) = (10+4)/10 = 1.4x storage overhead
  - RS(100,4) = (100+4)/100 = 1.04x storage overhead

What’s the Catch?

Catch 1: Encoding Overhead

- Replication:
  - Just copy the data

- Erasure coding:
  - Compute codes over N data blocks for each of the K coding blocks

Catch 2: Decoding Overhead

- Replication
  - Just read the data
- Erasure Coding
  - Normal case is no failures -> just read the data!
  - If there are failures
    - Read N blocks from disks and over the network
    - Compute code over N blocks to reconstruct the failed block

Catch 3: Updating Overhead

- Replication:
  - Update the data in each copy
- Erasure coding
  - Update the data in the data block
  - And all of the coding blocks

Catch 3’: Deleting Overhead

- Replication:
  - Delete the data in each copy
- Erasure coding
  - Delete the data in the data block
  - Update all of the coding blocks

Catch 4: Update Consistency

- Replication:
  - Consensus protocol (Paxos!)

- Erasure coding
  - Need to consistently update all coding blocks with a data block
  - Need to consistently apply updates in a total order across all blocks
  - Need to ensure reads, including decoding, are consistent

Catch 5: Fewer Copies for Reading

- Replication
  - Read from any of the copies

- Erasure coding
  - Read from the data block
  - Or reconstruct the data on fly if there is a failure

Catch 6: Larger Min System Size

- Replication
    - Need K+1 disjoint places to store data
    - e.g., 3 disks for 3x replication

- Erasure coding
    - Need M=N+K disjoint places to store data
    - e.g., 14 disks for RS(10,4) replication

Different codes make different tradeoffs

- Encoding, decoding, and updating overheads
- Storage overheads
  - Best are “Maximum Distance Separable” or “MDS” codes where K extra blocks allows you to tolerate any K failures
- Configuration options
  - Some allow any (N,K), some restrict choices of N and K

- See “Erasure Codes for Storage Systems, A Brief Primer. James S. Plank. Usenix ;login: Dec 2013” for a good jumping off point
  - Also a good, accessible resource generally

Let’s Improve Our New Hammer!

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!

- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - Updating overhead
    - Deleting overhead
  - Update consistency
  - Fewer copies for serving reads
  - Larger minimum system size

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!

- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - ~~Updating overhead~~
    - Deleting overhead
  - ~~Update consistency~~
  - Fewer copies for serving reads
  - Larger minimum system size

Immutable data

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!

- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - ~~Updating overhead~~
    - Deleting overhead
  - ~~Update consistency~~
  - Fewer copies for serving reads
  - ~~Larger minimum system size~~

Immutable data

Storing lots of data
(when storage overhead
actually matters this is true)

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!

- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - ~~Updating overhead~~
    - Deleting overhead
  - ~~Update consistency~~
  - Fewer copies for serving reads
  - ~~Larger minimum system size~~

Data is stored for a long time after being written

Immutable data

Storing lots of data (when storage overhead actually matters this is true)

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!
- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - ~~Updating overhead~~
    - Deleting overhead
  - ~~Update consistency~~
  - ~~Fewer copies for serving reads~~
  - ~~Larger minimum system size~~

Low read rate

Data is stored for a long time after being written

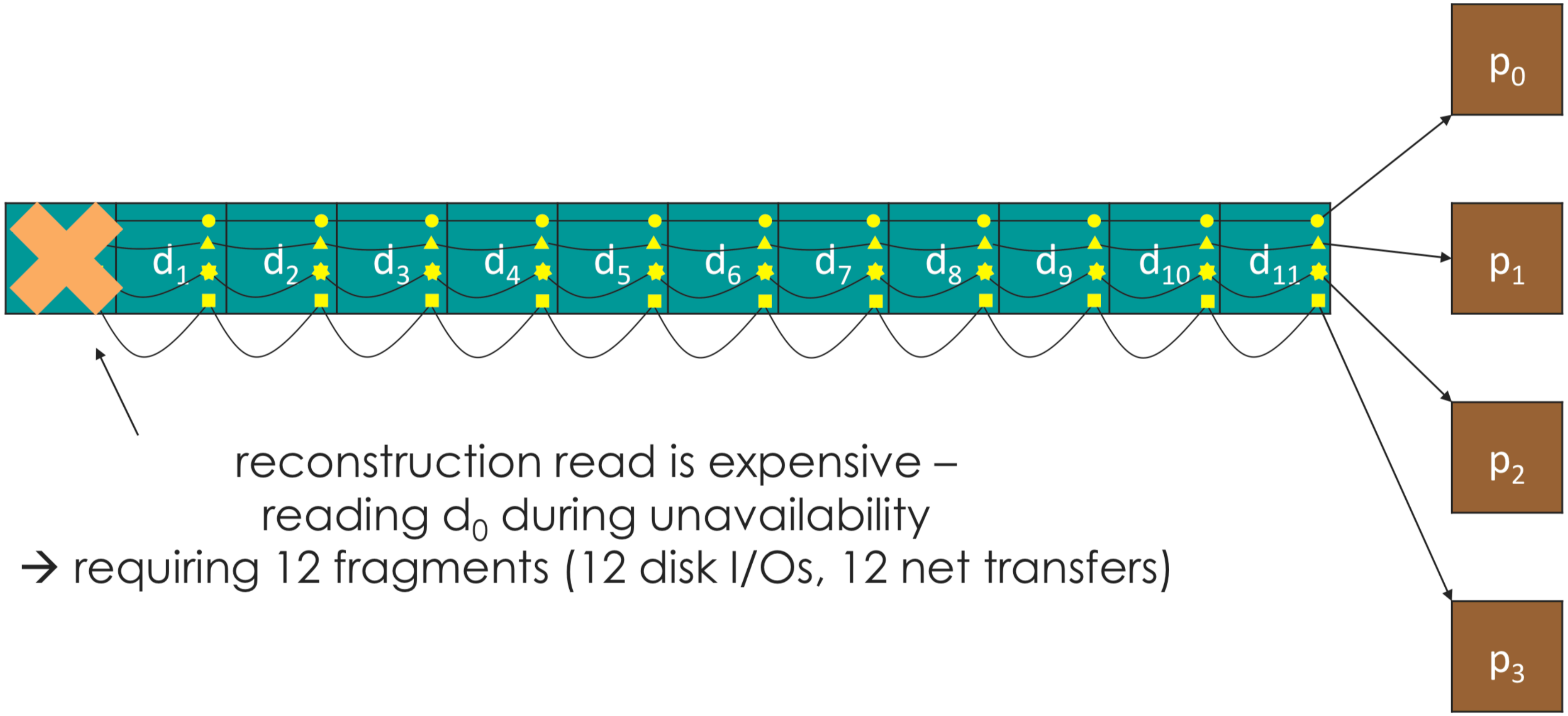
Immutable data

Storing lots of data (when storage overhead actually matters this is true)

Adoption of EC

- Facebook: f4 stores 65PB of BLOBs in EC
- Windows Azure Storage (WAS)
  A PB of new data every 1~2 days - All “sealed” data stored in EC
- Google File System
  Large portion of data stored in EC

EC in Windows Azure Storage



reconstruction read is expensive –
reading d_0 during unavailability
→ requiring 12 fragments (12 disk I/Os, 12 net transfers)

Reconstruction - When?

- Load balancing
  - avoid hot storage node -> serve reads via reconstruction
- rolling upgrade of disks
- transient unavailability and performance failures
- Can we achieve 1.33x overhead while requiring only 6 fragments for reconstruction?

Opportunity

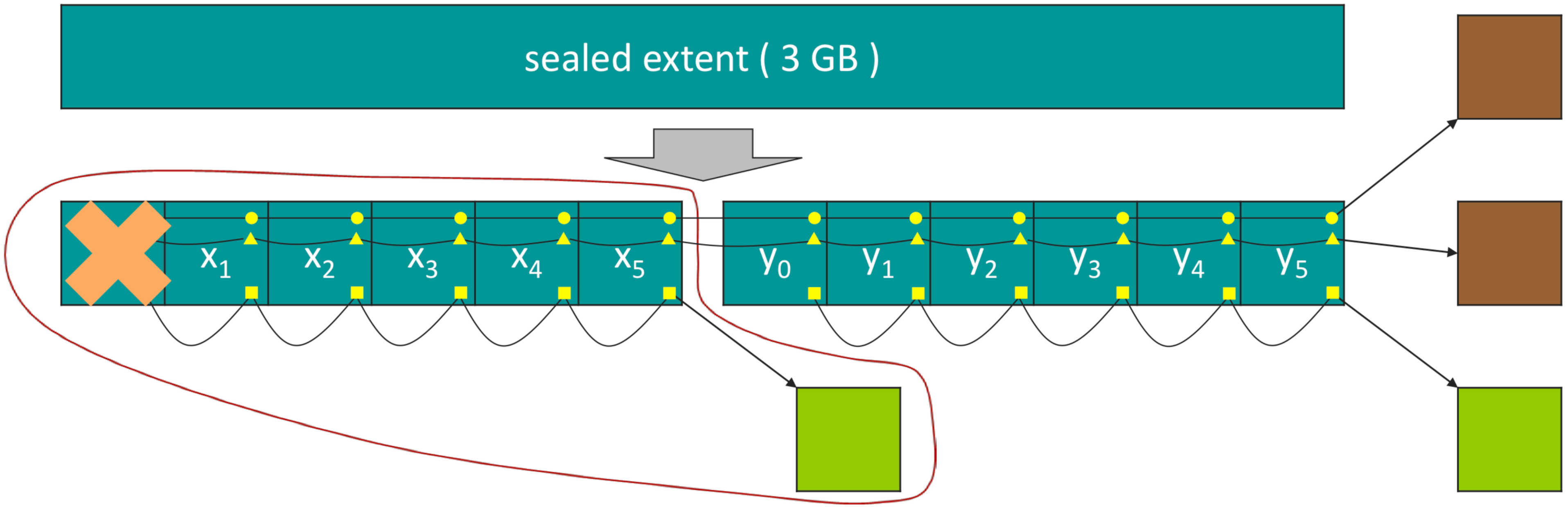
- Conventional EC
  - all failures are equal -> same reconstruction cost, regardless of #failure
  - cloud storage
    - prob (1 failure) >> prob(2 failures)

# blocks missing in stripe	% of stripes with missing blocks
1	98.08
2	1.87
3	0.036
4	9×10^{-6}
≥ 5	9×10^{-9}

Opportunity

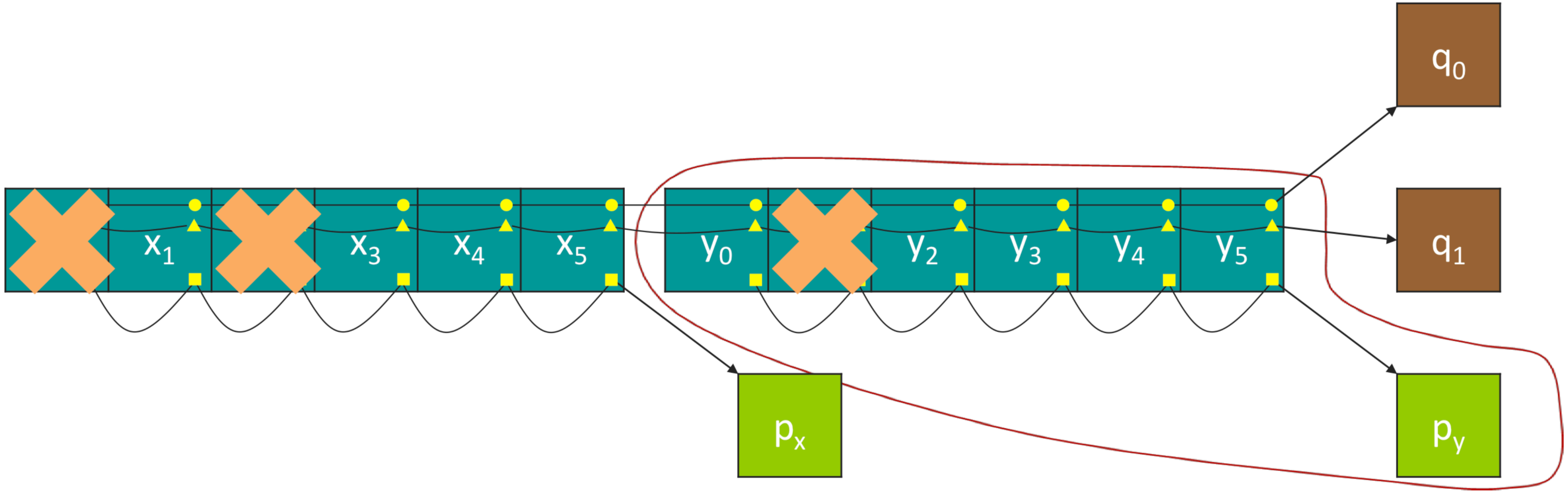
- Conventional EC
  - all failures are equal -> same reconstruction cost, regardless of #failure
  - cloud storage
    - prob (1 failure) >> prob(2 failures)
- Optimize erasure coding for cloud storage
  - making single failure reconstruction most efficient

Local reconstruction code



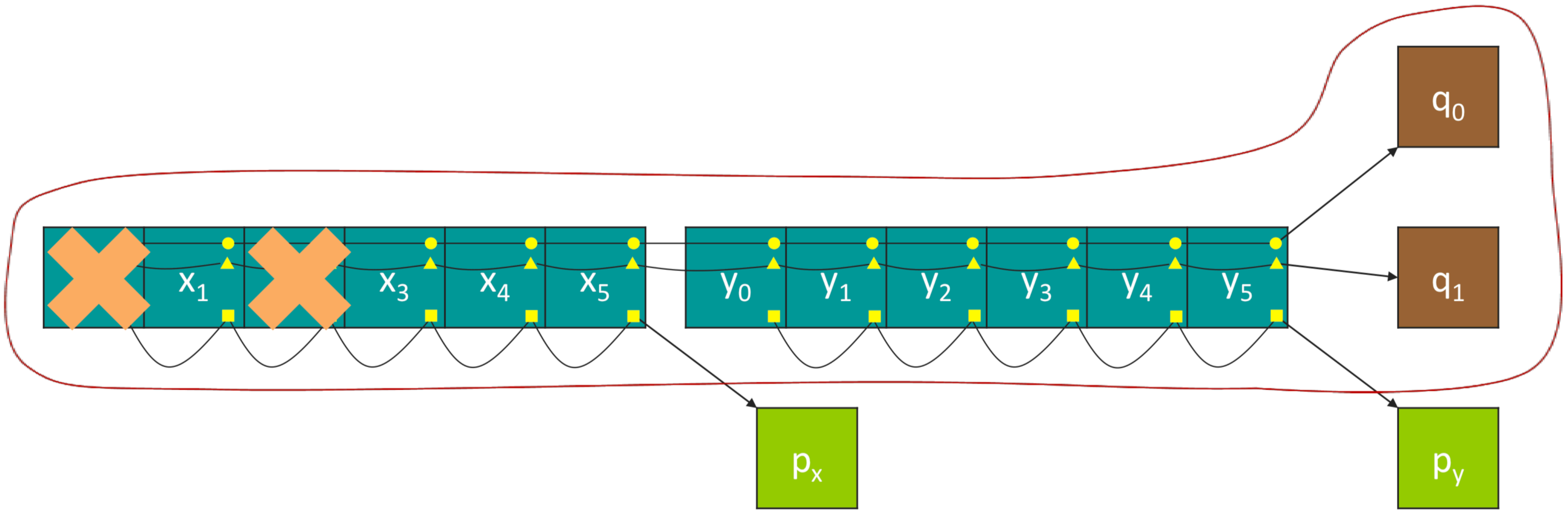
- LRC_{12+2+2}: **12** data fragments, **2** local parities and **2** global parities
  - storage overhead: (12 + 2 + 2) / 12 = 1.33x
- Local parity: reconstruction requires only 6 fragments

Reconstruct 3 failures



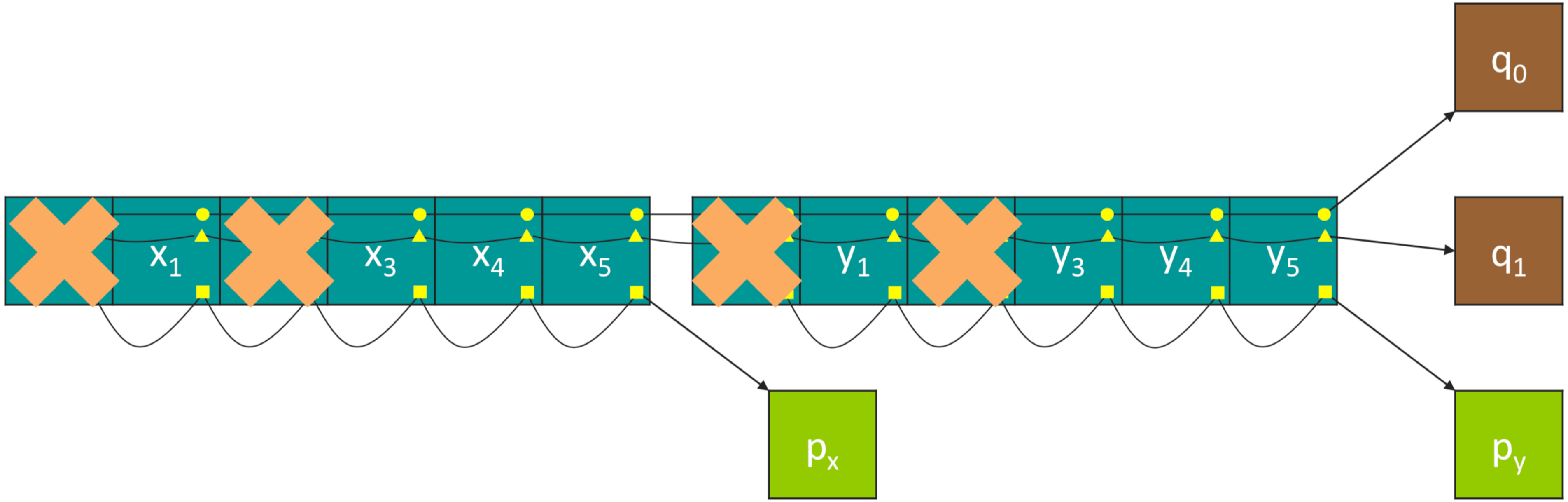
recover y_1 from p_y (group y)

Reconstruct 3 failures

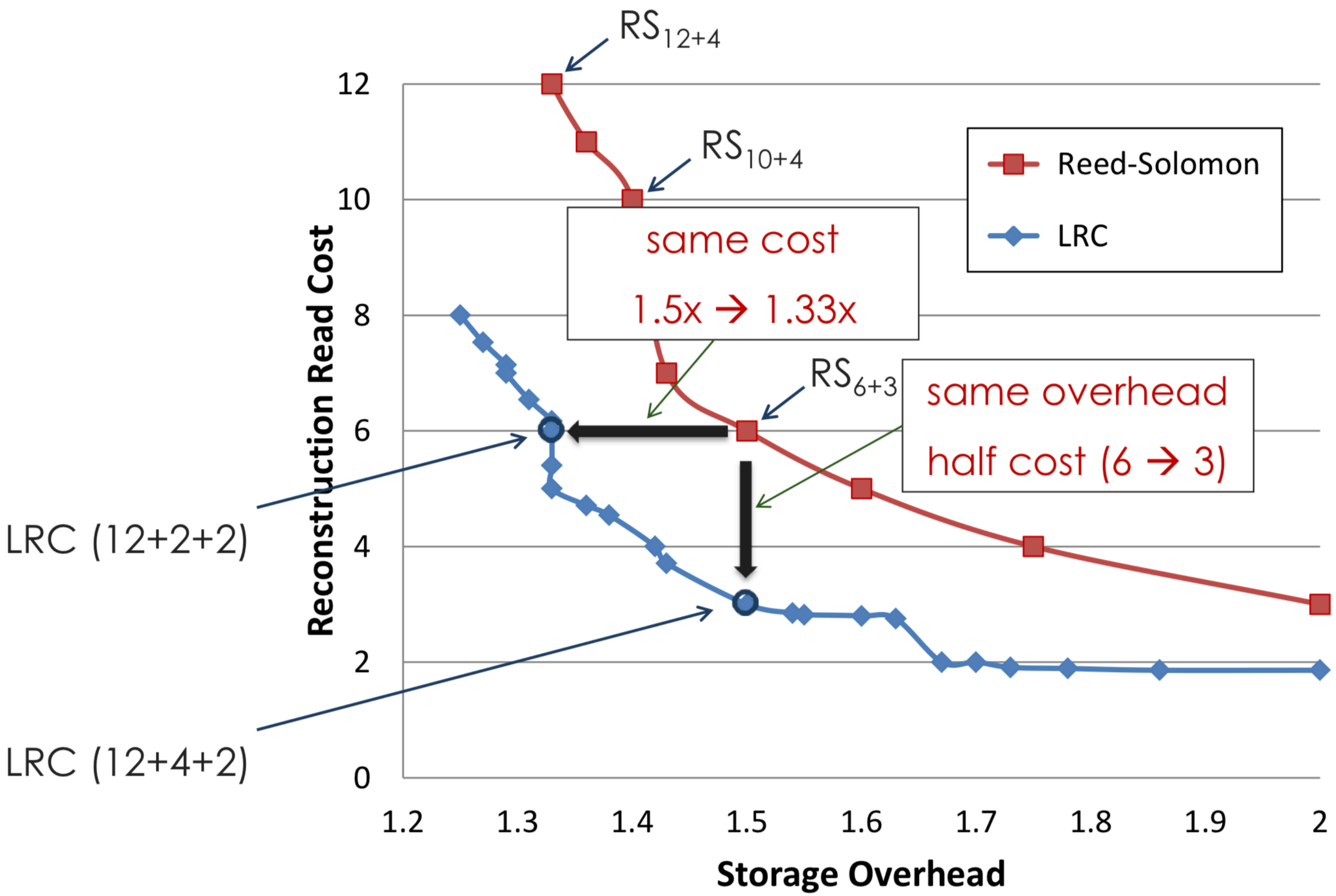


recover y_1 from p_y (group y)
recover x_0 and x_2 from q_0 and q_1

Reconstruct 4 failures



how to recover the 4 failures and all similar cases?



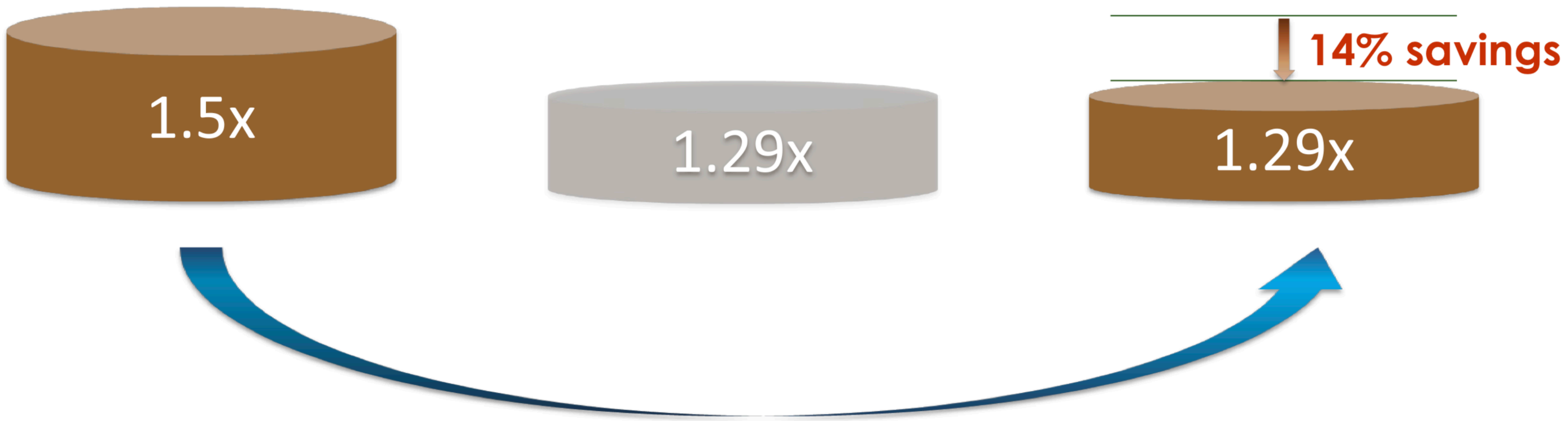
- RS_{10+4}: HDFS-RAID at Facebook
- RS_{6+3}: GFS II (Colossus) at Google

Choice of Windows Azure Storage

RS (6 + 3)
reconstruction cost = 6

RS (14 + 4)
reconstruction cost = 14

LRC (14 + 2 + 2)
reconstruction cost = 7



Properties of LRC

- Reliability
  - LRC_{12+2+2}: arbitrary 3 failures and 86% of 4 failures
  - reliability: RS_{12+4} > LRC_{12+2+2} > RS_{6+3}
- Requiring minimum storage overhead given
  - reconstruction cost and fault tolerance