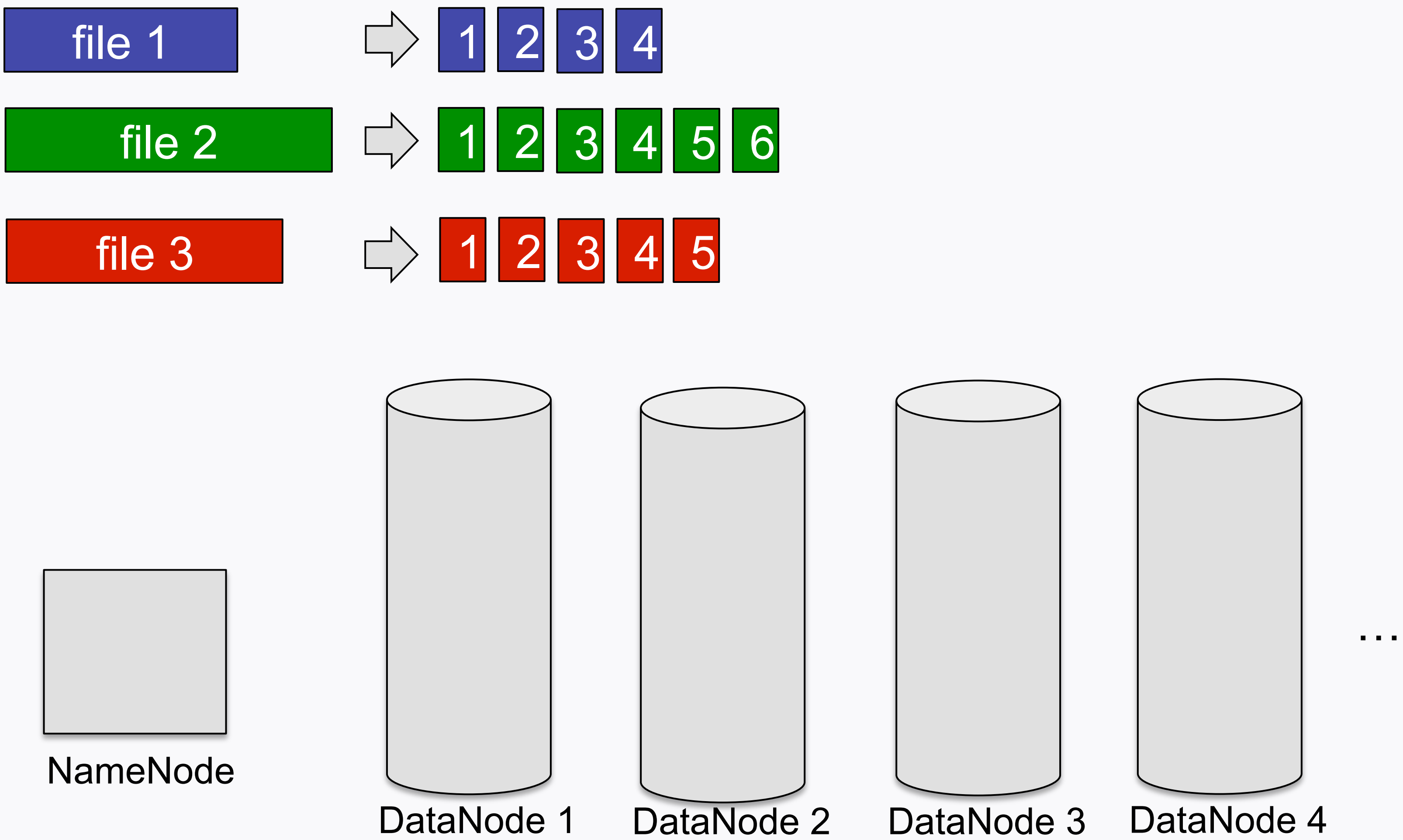
Overview

- How distributed file systems work
- Part 1: The code repair problem
- Regenerating Codes

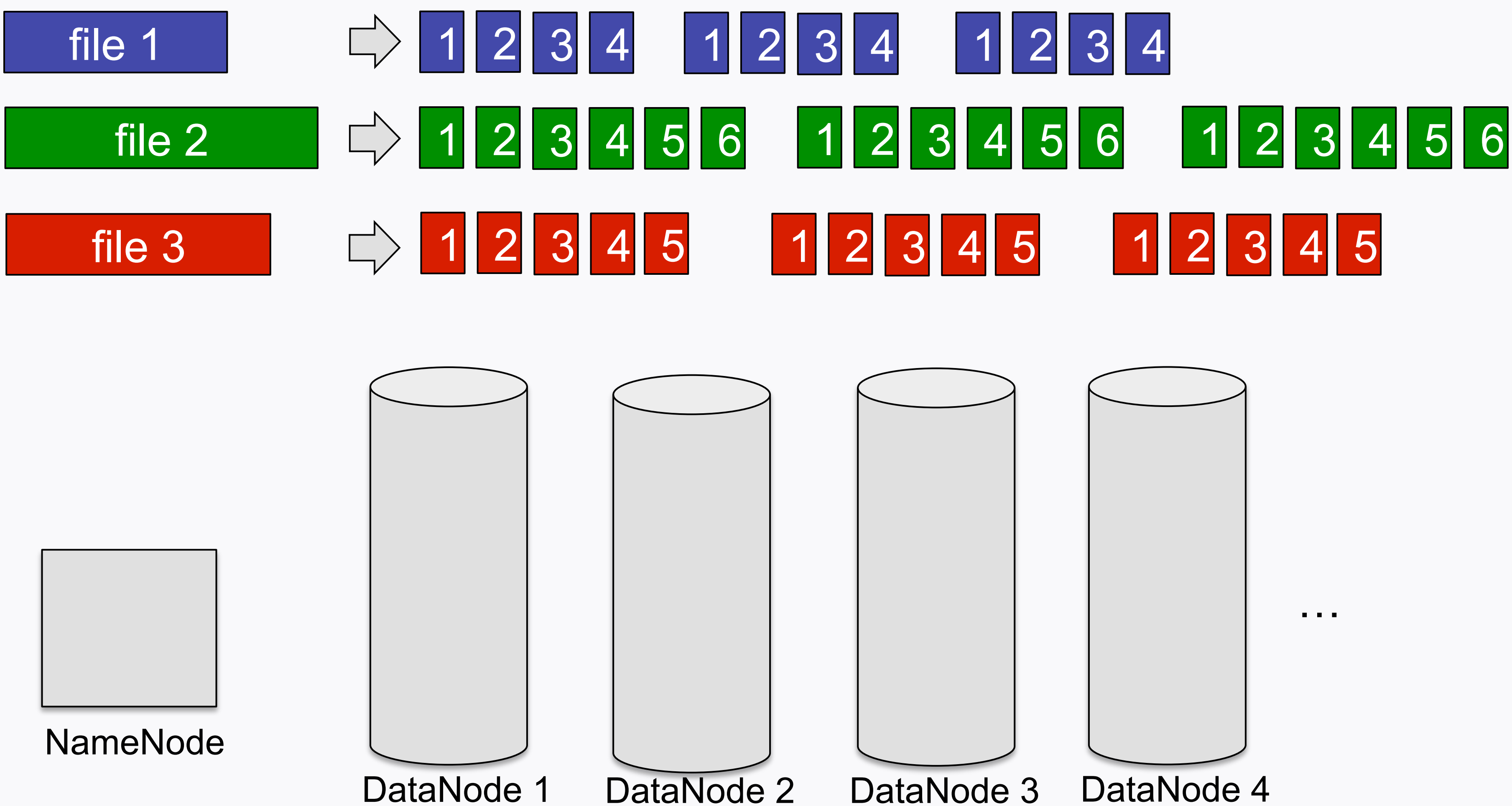
- Part 2: Locally Repairable Codes

- Part 3: Availability of Codes
- Open problems

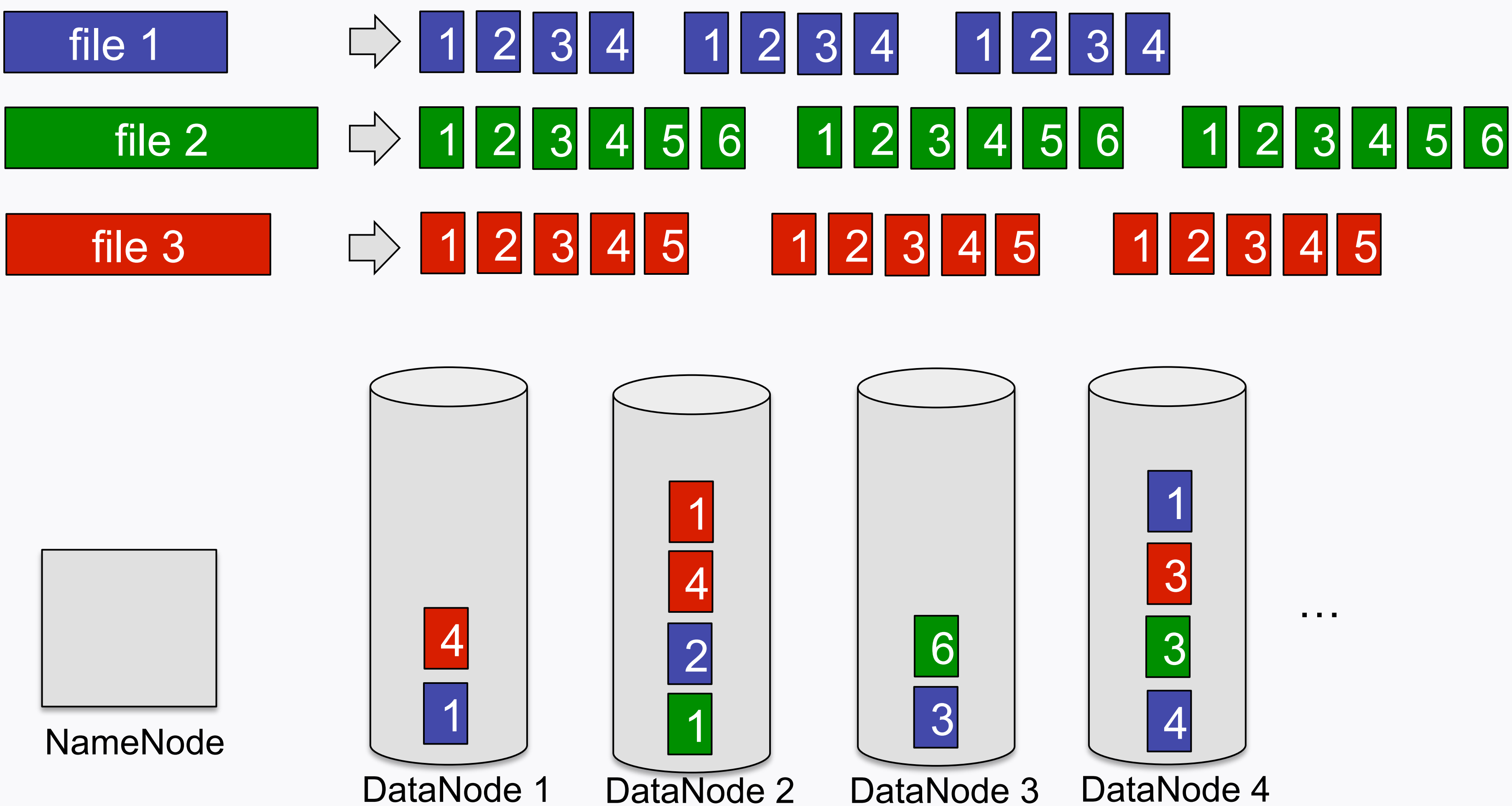
current hadoop architecture



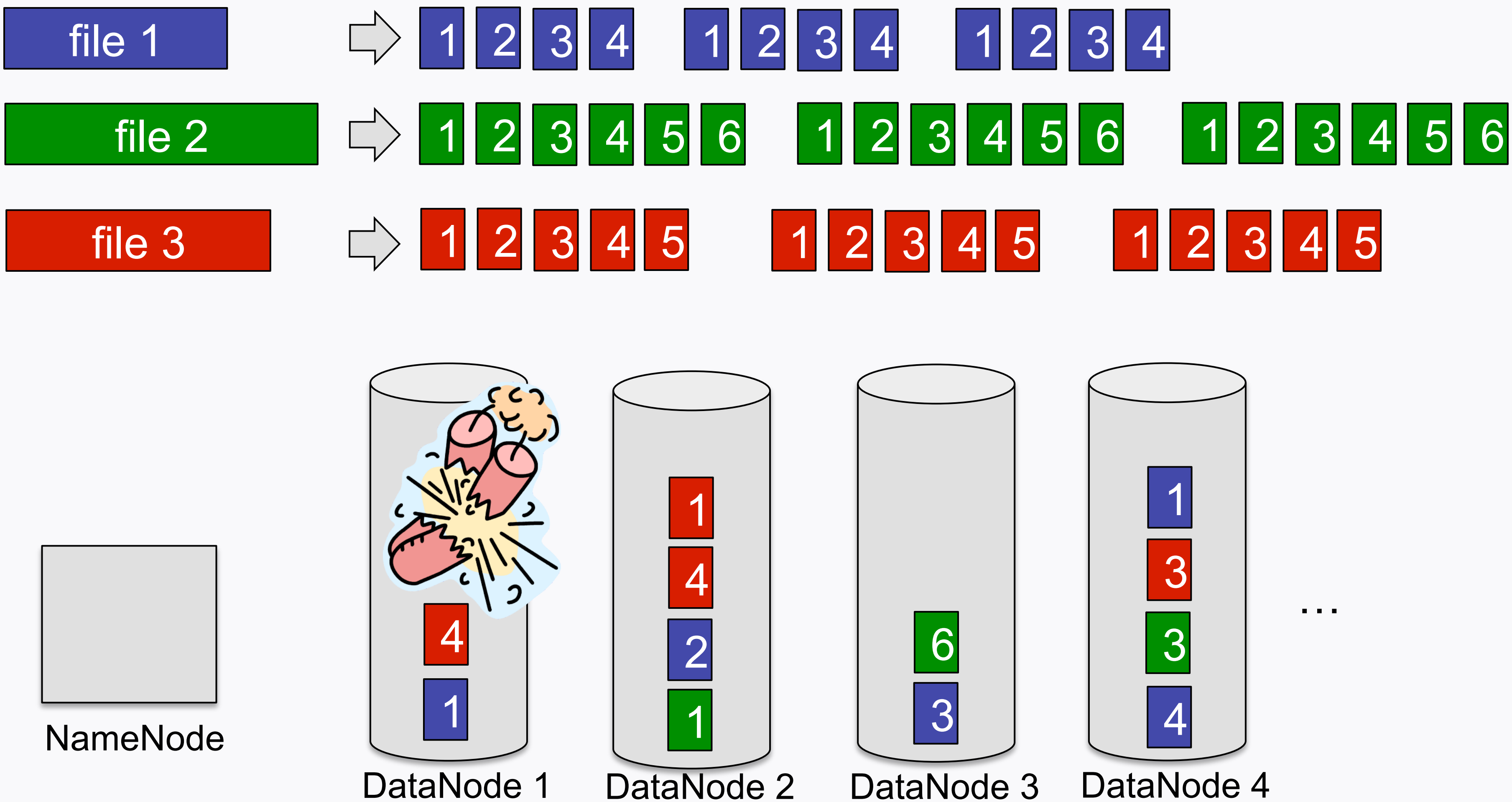
current hadoop architecture



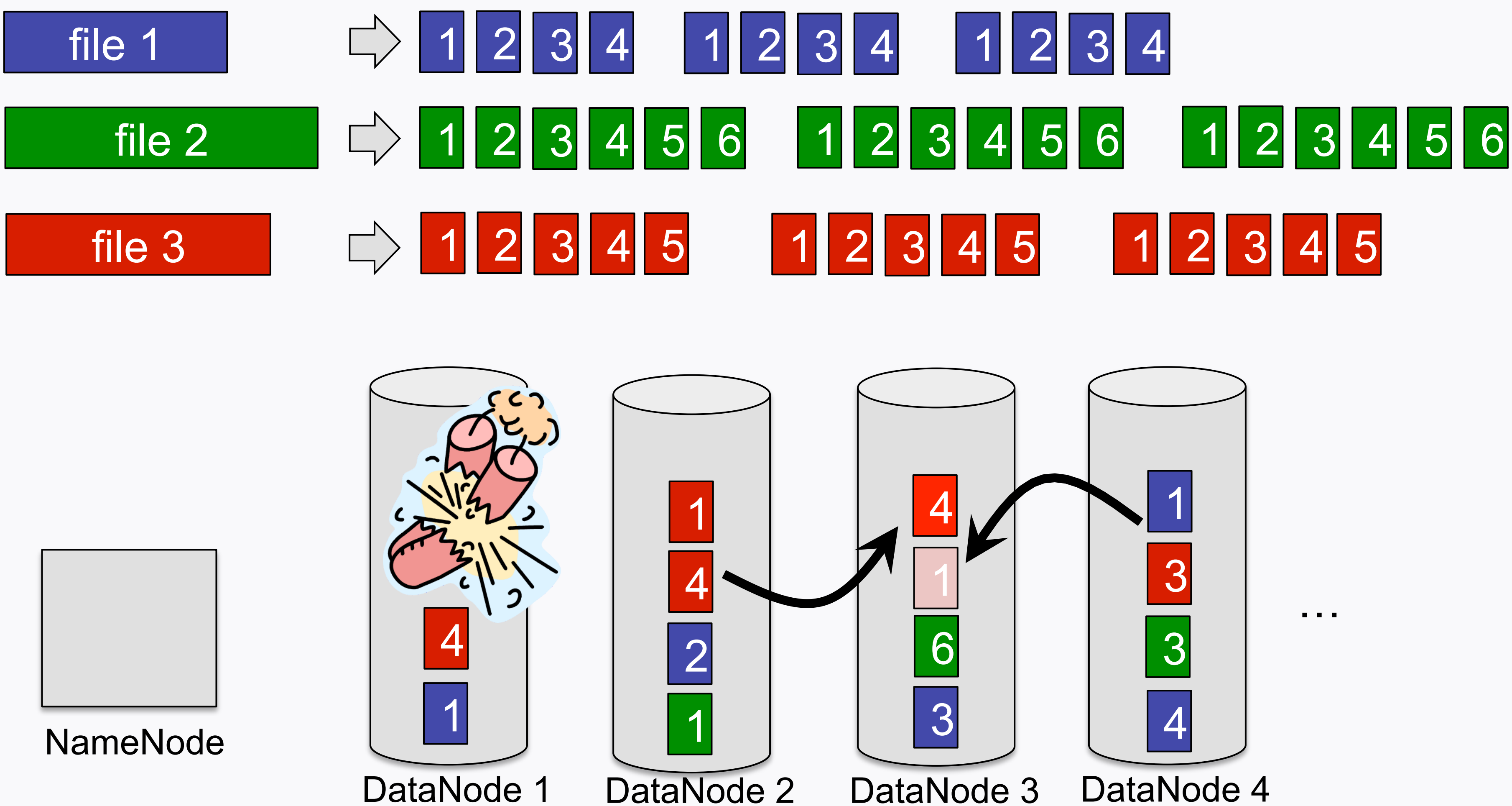
current hadoop architecture



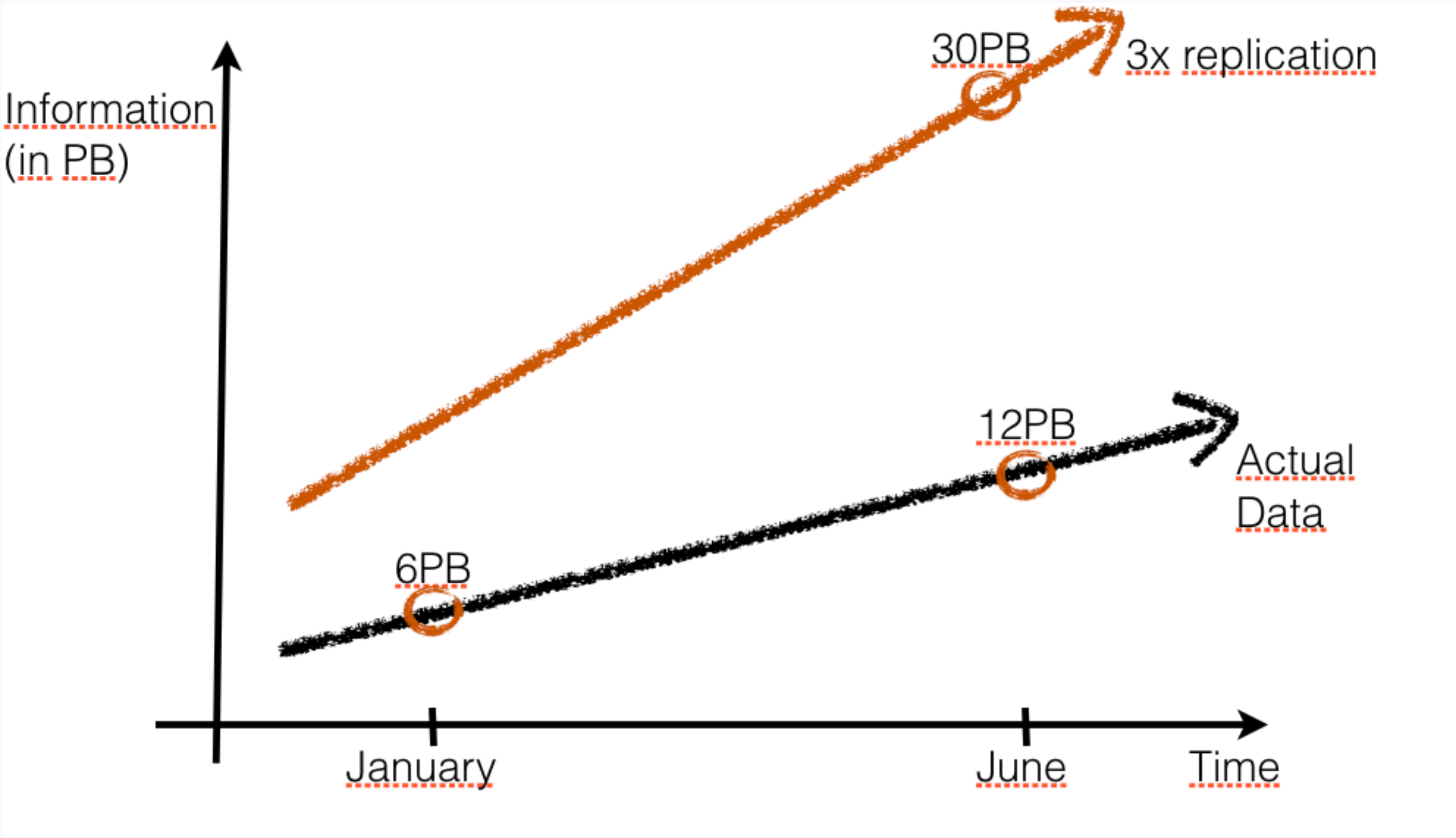
current hadoop architecture



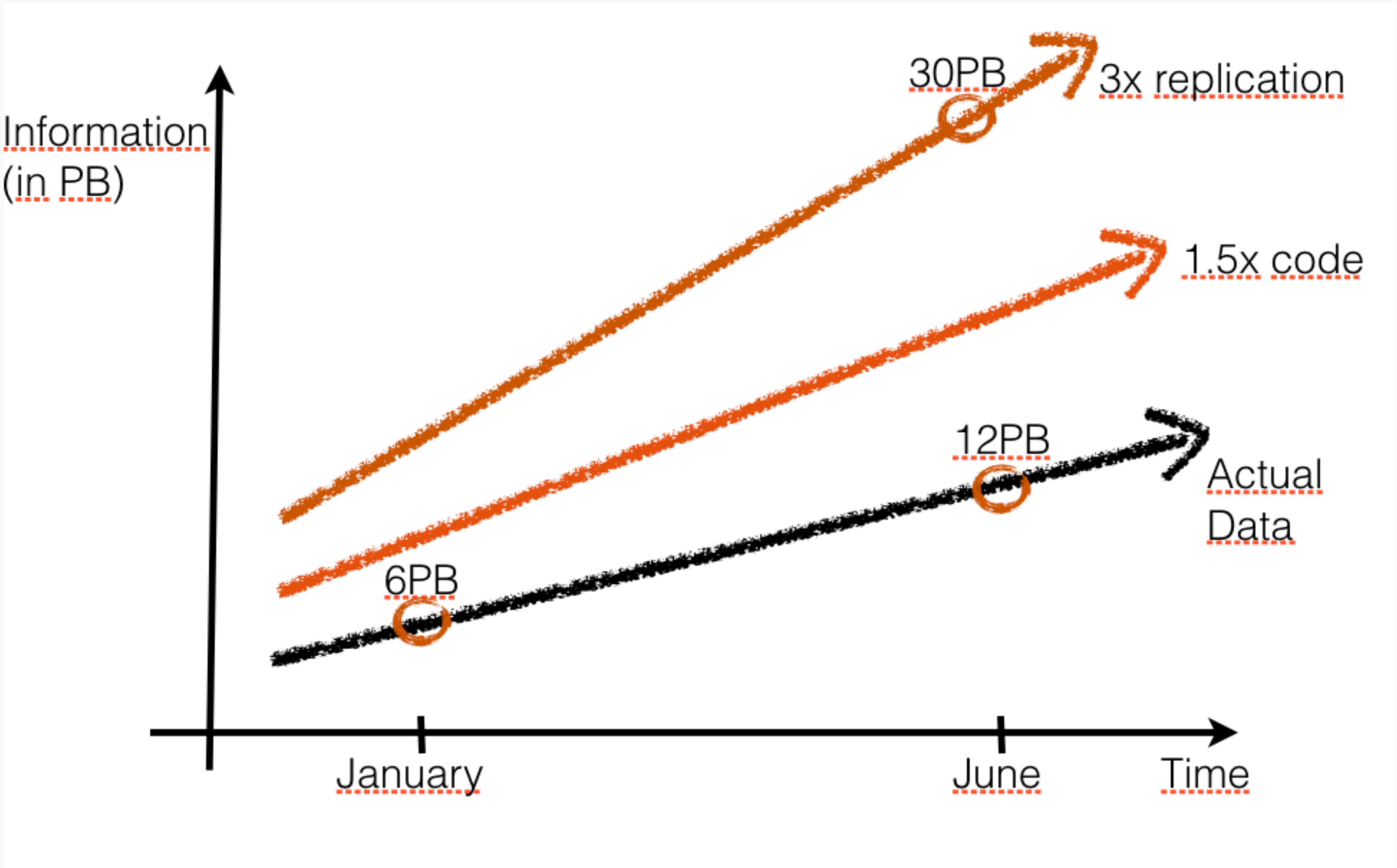
current hadoop architecture



erasure codes save space



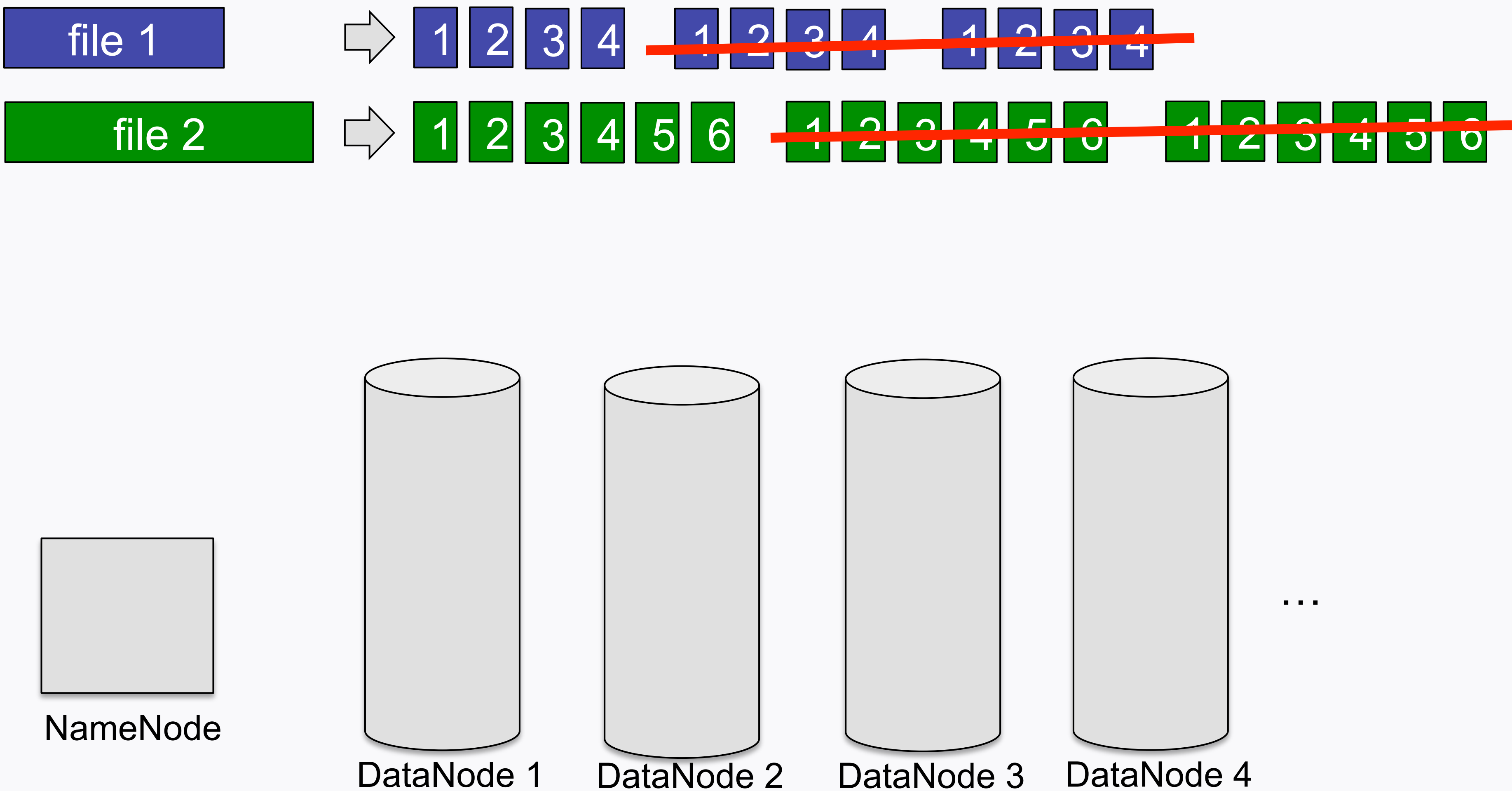
erasure codes save space



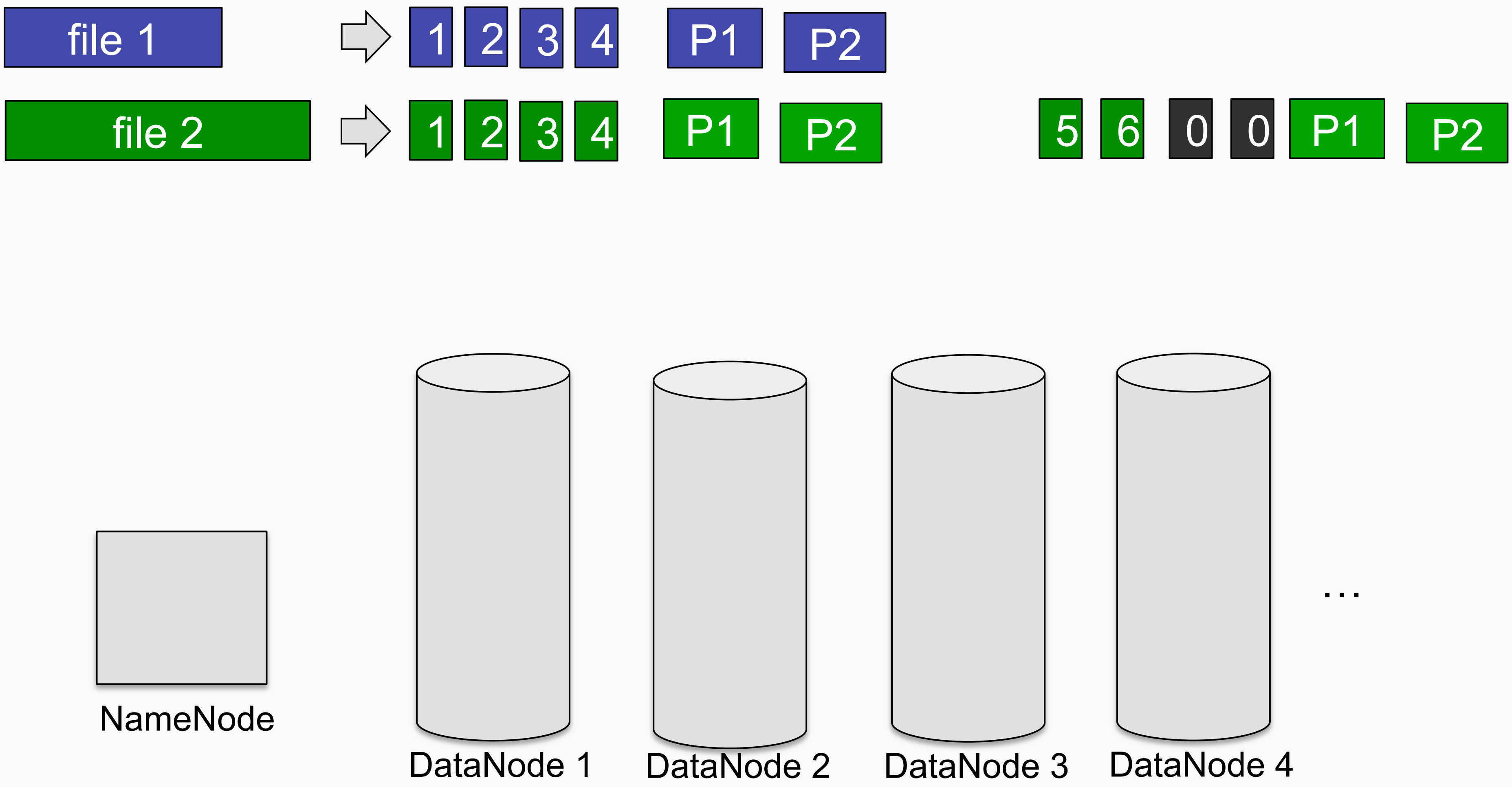
Real systems that use distributed storage codes

- Windows Azure, (Cheng et al. USENIX 2012) (LRC Code)
- Used in production in Microsoft
- CORE (PPC Li et al. MSST 2013) (Regenerating EMSR Code)
- NCCloud (Hu et al. USENIX FAST 2012) (Regenerating Functional MSR)
- ClusterDFS (Pamies Juarez et al.) (SelfRepairing Codes)
- StorageCore (Esmaili et al.) (over Hadoop HDFS)
- HDFS Xorbas (Sathiamoorthy et al. VLDB 2013) (over Hadoop HDFS) (LRC code on Facebook clusters)

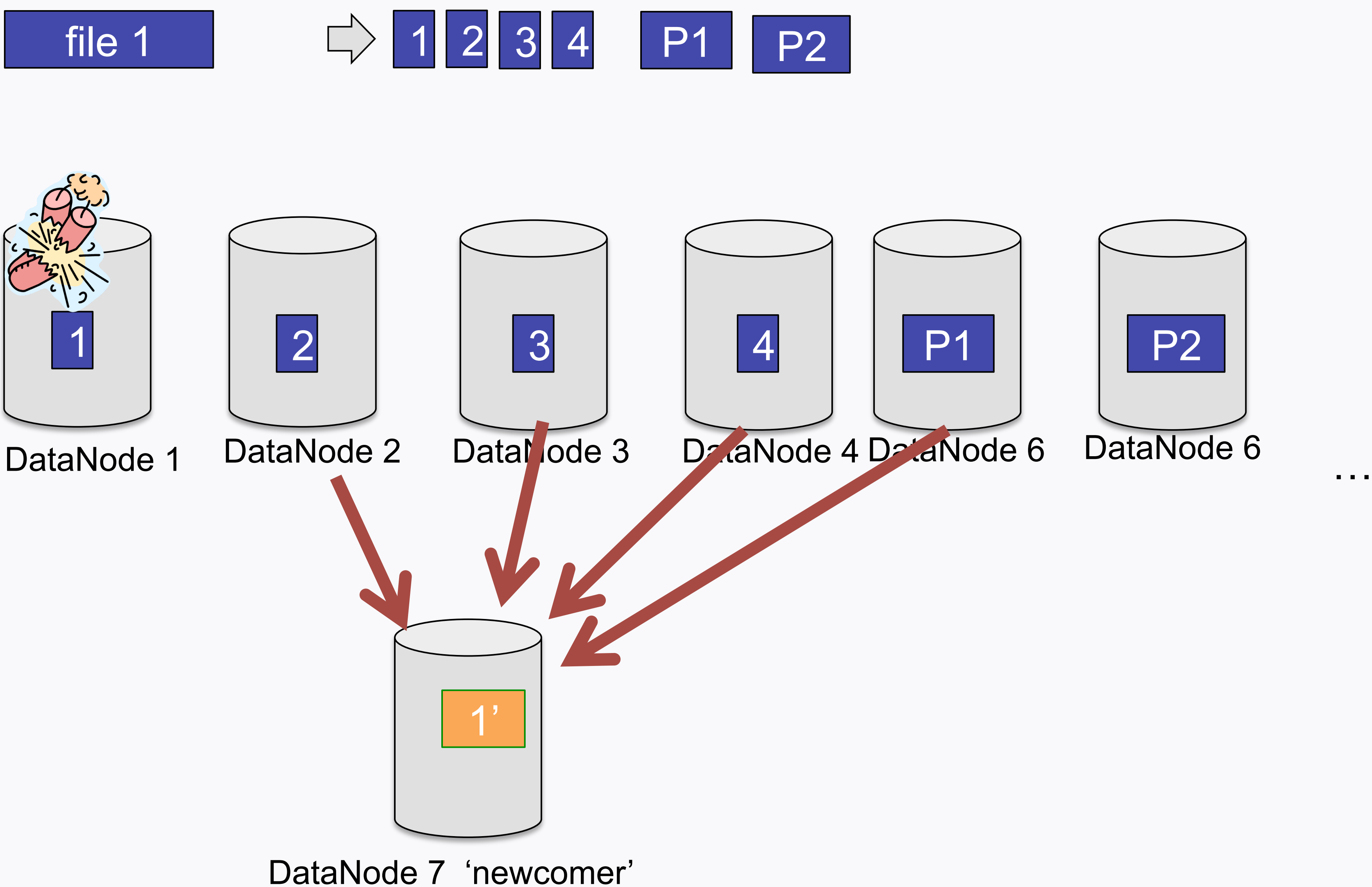
Coded hadoop



Coded hadoop



Code repair



file 1
1
2
3
4
P1
P2
1
2
3
4
P1
P2
DataNode 1
DataNode 2
DataNode 3
DataNode 4
DataNode 6
DataNode 6
…
1'
DataNode 7 'newcomer'

Three repair metrics of interest

1. Number of bits communicated in the network during single node failures (Repair Bandwidth)

2. The number of bits read from disks during single node repairs (Disk IO)

3. The number of nodes accessed to repair a single node failure (Locality)

Three repair metrics of interest

1. Number of bits communicated in the network during single node failures (Repair Bandwidth)

   Capacity known for two points only. My 3-year old conjecture for intermediate points was just disproved. [ISIT13]

2. The number of bits read from disks during single node repairs (Disk IO)

3. The number of nodes accessed to repair a single node failure (Locality)

Three repair metrics of interest

1. Number of bits communicated in the network during single node failures (Repair Bandwidth)

   Capacity known for two points only. My 3-year old conjecture for intermediate points was just disproved. [ISIT13]

2. The number of bits read from disks during single node repairs (Disk IO)

   Capacity unknown.
   Only known technique is bounding by Repair Bandwidth

3. The number of nodes accessed to repair a single node failure (Locality)

Three repair metrics of interest

1. Number of bits communicated in the network during single node failures (Repair Bandwidth)

   Capacity known for two points only. My 3-year old conjecture for intermediate points was just disproved. [ISIT13]

2. The number of bits read from disks during single node repairs (Disk IO)

   Capacity unknown.
   Only known technique is bounding by Repair Bandwidth

3. The number of nodes accessed to repair a single node failure (Locality)

   Capacity known for some cases.
   Practical LRC codes known for some cases. [ISIT12, Usenix12, VLDB13]
   General constructions open

Three repair metrics of interest

1. Number of bits communicated in the network during single node failures (Repair Bandwidth)

   Capacity known for two points only. My 3-year old conjecture for intermediate points was just disproved. [ISIT13]

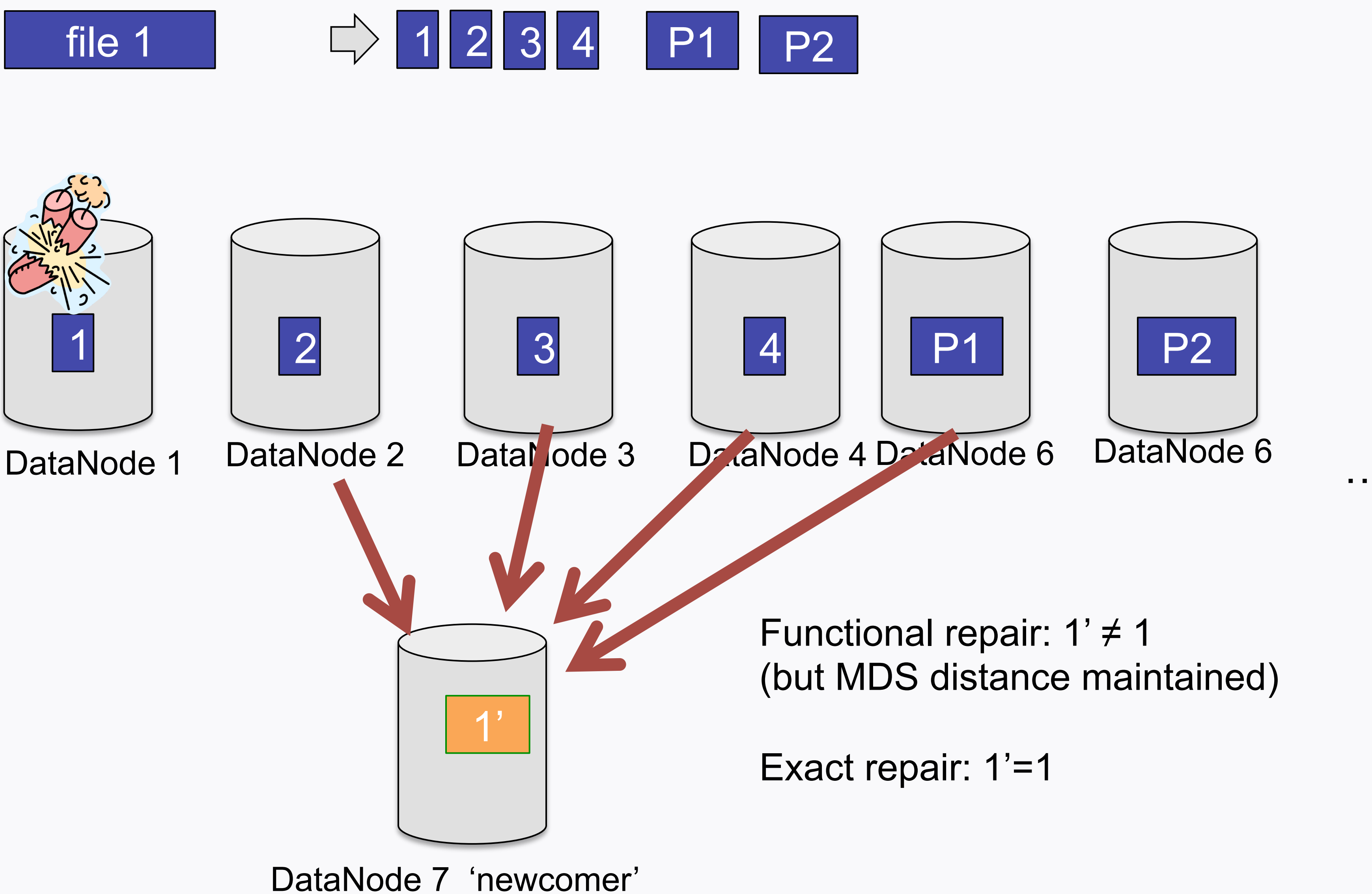
2. The number of bits read from disks during single node repairs (Disk IO)

   Capacity unknown.
   Only known technique is bounding by Repair Bandwidth

3. The number of nodes accessed to repair a single node failure (Locality)

   Capacity known for some cases.
   Practical LRC codes known for some cases. [ISIT12, Usenix12, VLDB13]
   General constructions open

Code repair bandwidth



Functional repair: 1' ≠ 1
(but MDS distance maintained)

Exact repair: 1'=1

Functional repair capacity

Theorem: It is possible to functionally repair a code by communicating only

$$\frac{n-1}{n-k}\frac{B}{k} \qquad \text{bits}$$

As opposed to naïve repair cost of B bits.

(Regenerating Codes, IT Transactions 2010)

Functional repair capacity

Theorem: It is possible to *functionally* repair a code by communicating only

$$\frac{n-1}{n-k}\frac{B}{k} \qquad \text{bits}$$

As opposed to naïve repair cost of B bits.

(Regenerating Codes, IT Transactions 2010)

Quiz: Apply this to the previous code. If each block is 64MB.

Functional repair capacity

Theorem: It is possible to functionally repair a code by communicating only

$$\frac{n-1}{n-k}\frac{B}{k} \qquad \text{bits}$$

As opposed to naïve repair cost of B bits.

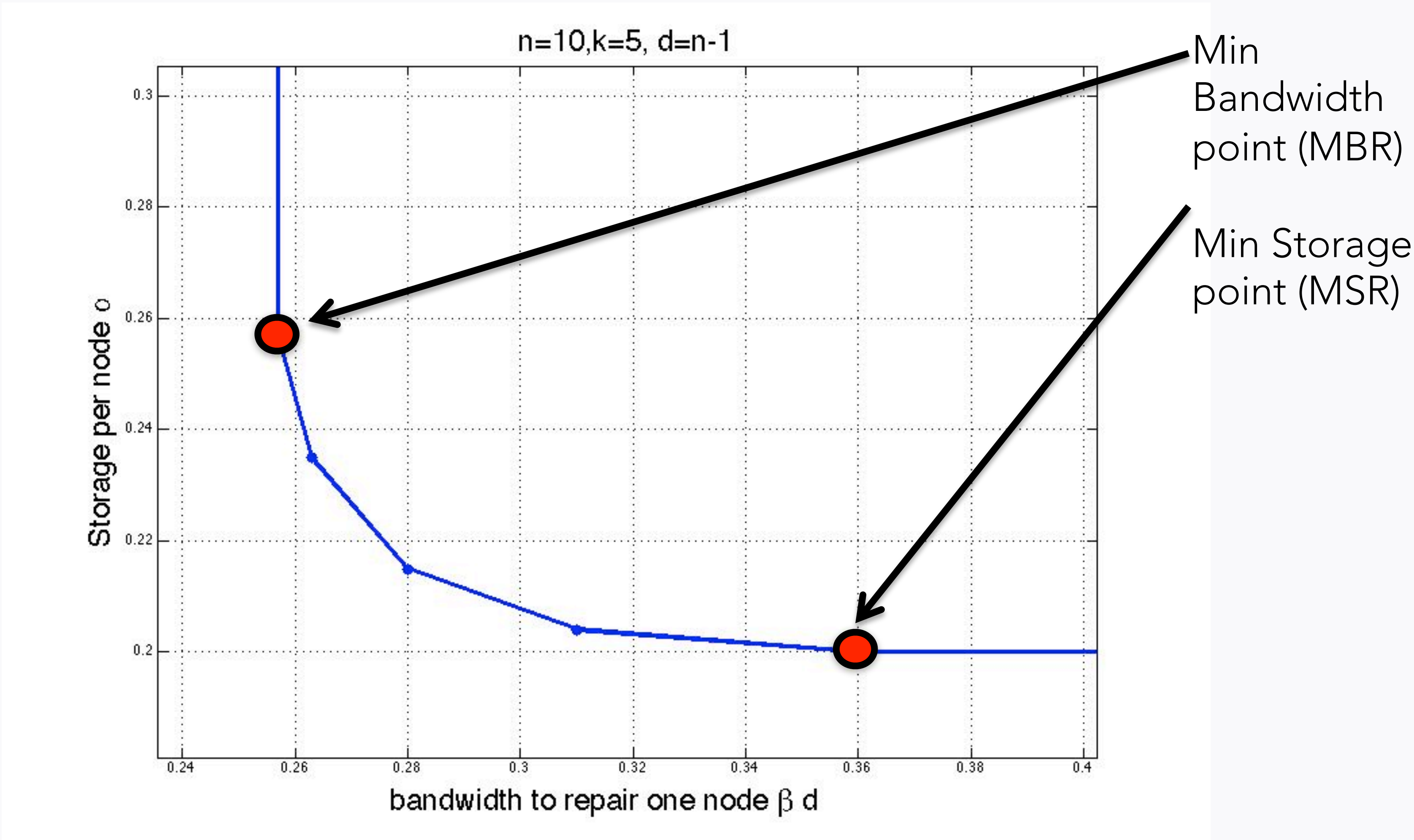
(Regenerating Codes, IT Transactions 2010)

Quiz: Apply this to the previous code. If each block is 64MB.
k=4, n=6.
B=64 k = 256MB= naïve repair communication
5/2 * 64 = 160 MB = optimal repair communication

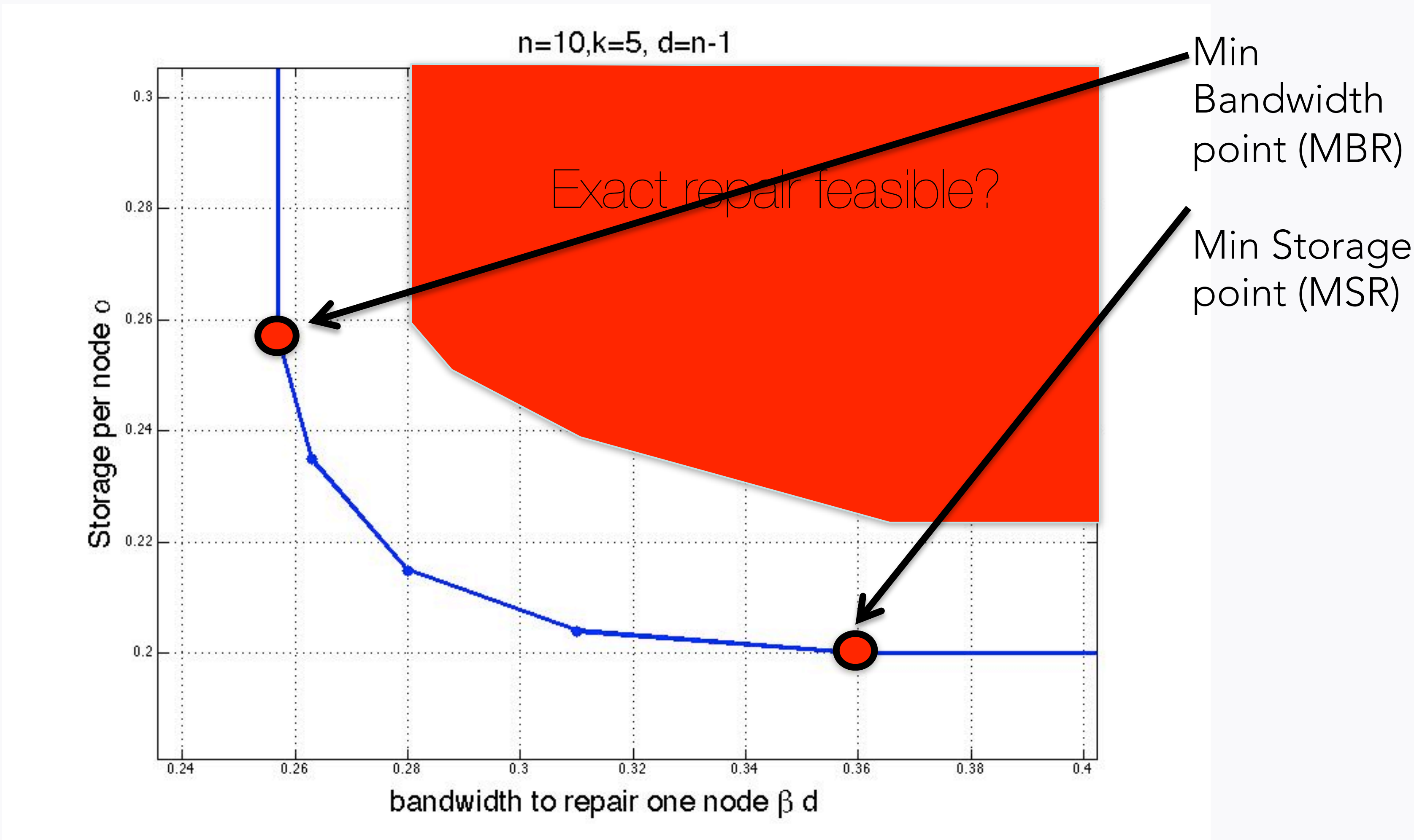
Storage-Bandwidth tradeoff

- For any (n,k) code we can find the minimum functional repair bandwidth
- There is a tradeoff between storage α and repair bandwidth β.
- This defines a tradeoff region for functional repair.

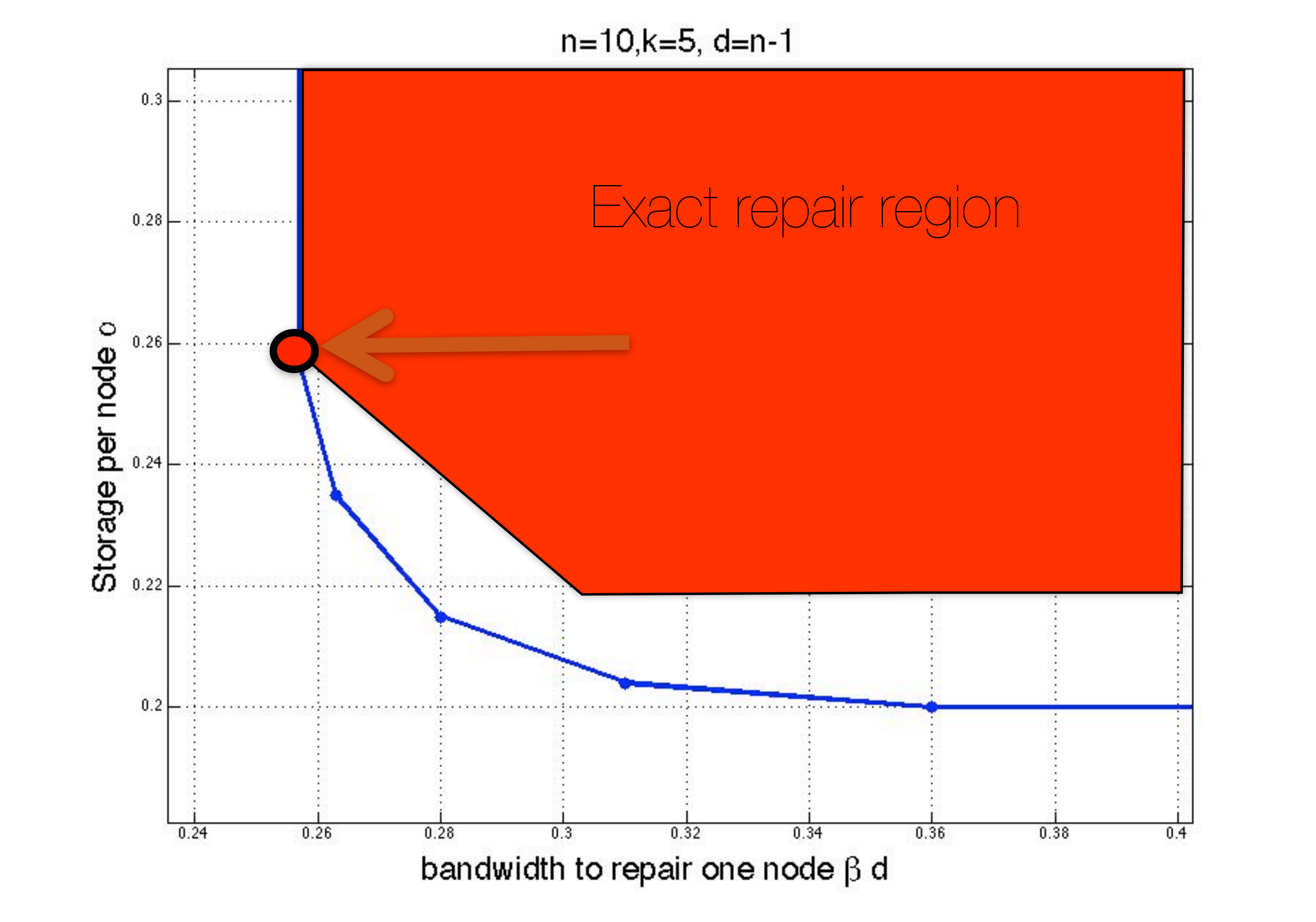
Repair Bandwidth Tradeoff Region



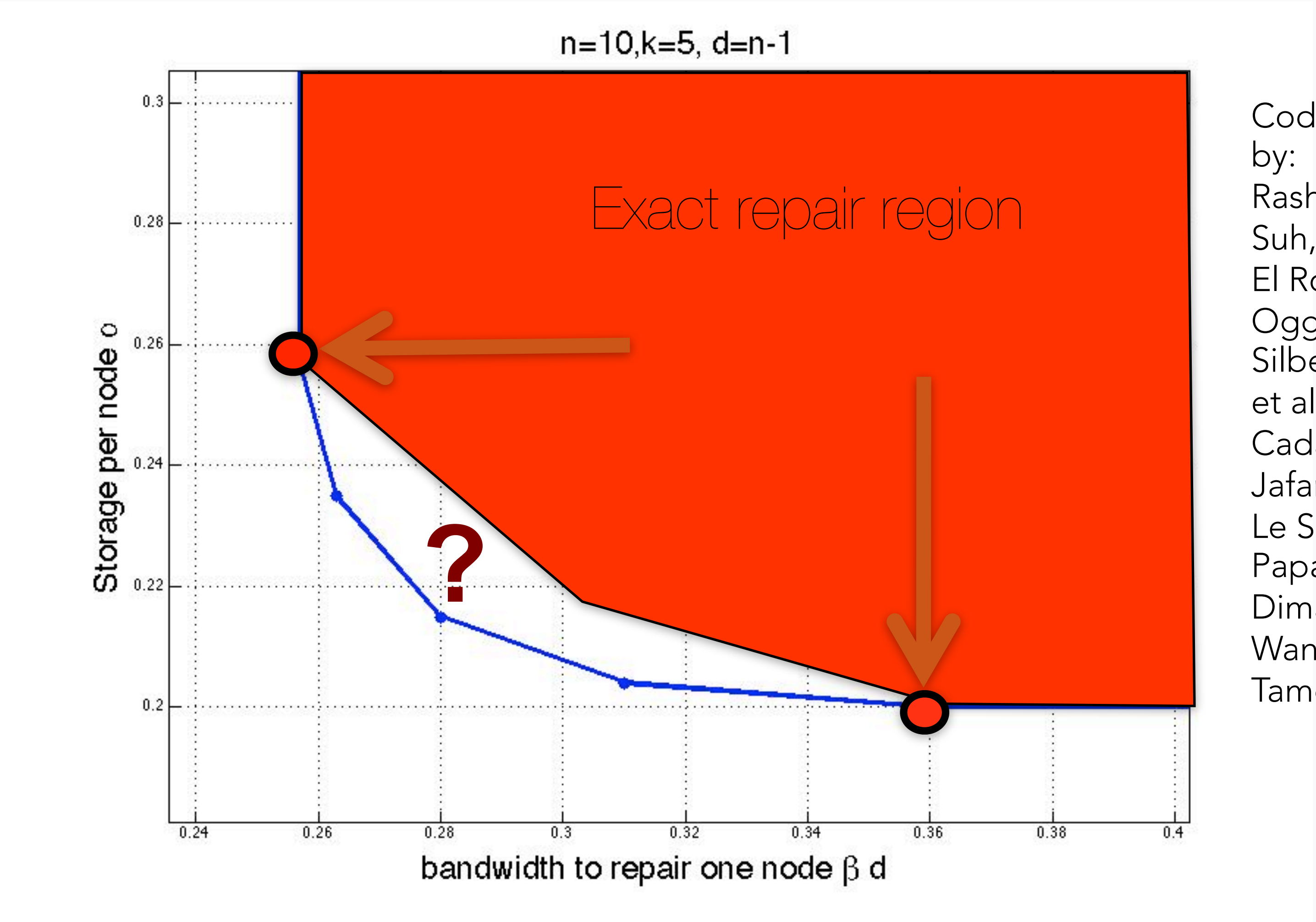
Exact repair region?



Status in 2011

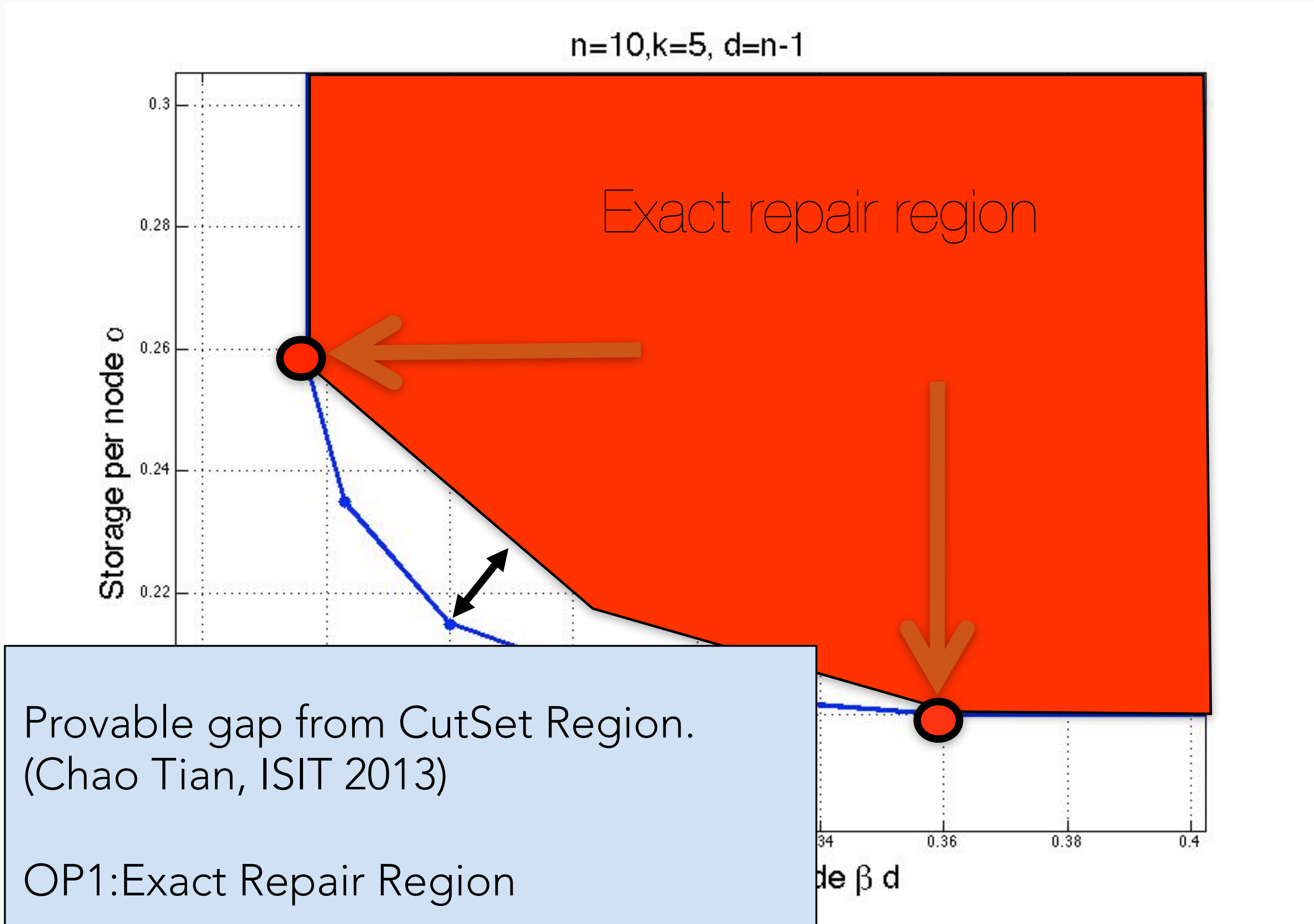


Status in 2012



Code constructions by:
Rashmi,Shah,Kumar
Suh,Ramchandran
El Rouayheb, Oggier,Datta
Silberstein, Viswanath et al.
Cadambe,Maleki, Jafar
Le Scouarnec et al.
Papailiopoulos, Wu, Dimakis
Wang, Tamo, Bruck
Tamo, Barg

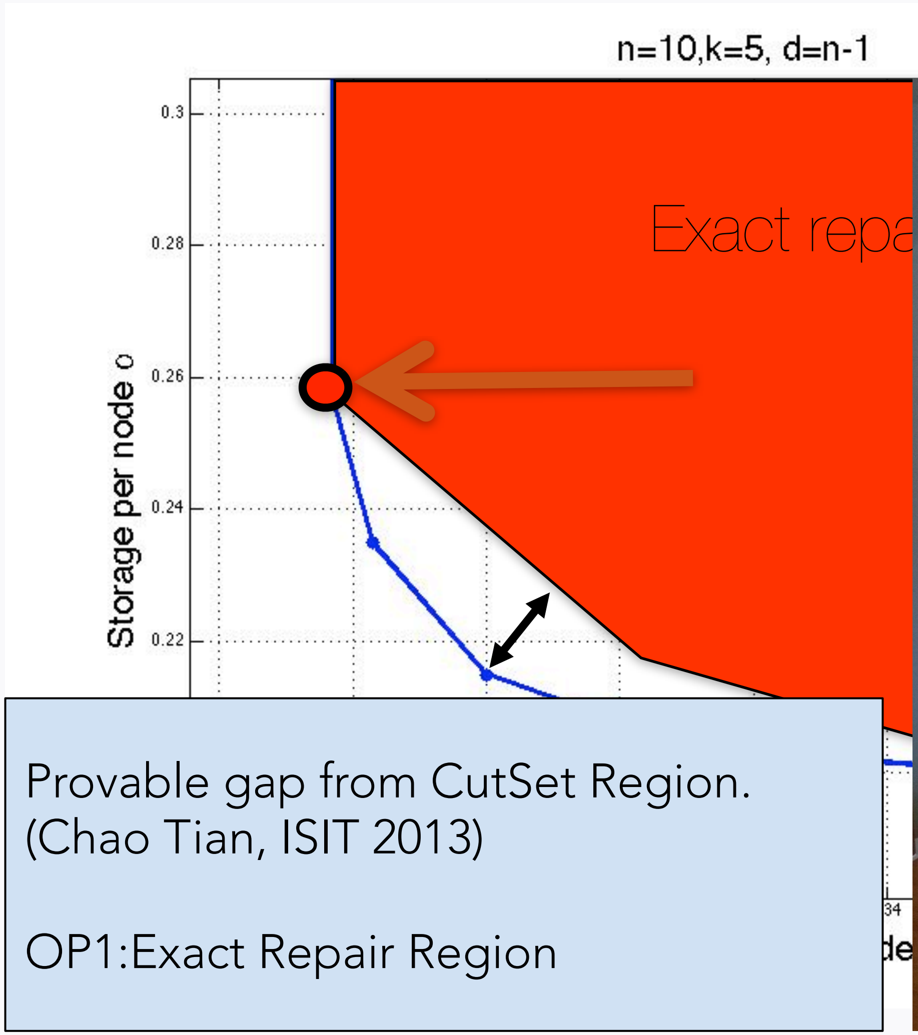
Status in 2013



Provable gap from CutSet Region.
(Chao Tian, ISIT 2013)

OP1:Exact Repair Region

Status in 2014



Provable gap from CutSet Region.
(Chao Tian, ISIT 2013)

OP1:Exact Repair Region

Taking a step back

- Finding exact regenerating codes is still an open problem in coding theory

- What can we do to make progress ?

Taking a step back

- Finding exact regenerating codes is still an open problem in coding theory

- What can we do to make progress ?

Taking a step back

- Finding exact regenerating codes is still an open problem in coding theory

- What can we do to make progress ?

-  or *change the question*

Part 2
Locally Repairable Codes

Three repair metrics of interest

1. Number of bits communicated in the network during single node failures (Repair Bandwidth)

   Capacity known for two points only. My 3-year old conjecture for intermediate points was just disproved. [ISIT13]

2. The number of bits read from disks during single node repairs (Disk IO)

   Capacity unknown.
   Only known technique is bounding by Repair Bandwidth

3. The number of nodes accessed to repair a single node failure (Locality)

   Capacity known for some cases.
   Practical LRC codes known for some cases. [ISIT12, Usenix12, VLDB13]
   General constructions open

Minimum Distance

- The distance of a code **d** is the minimum number of erasures after which data is lost.
- Reed-Solomon (10,14) (n=14, k=10). d= 5
- R. Singleton (1964) showed a bound on the best distance possible:

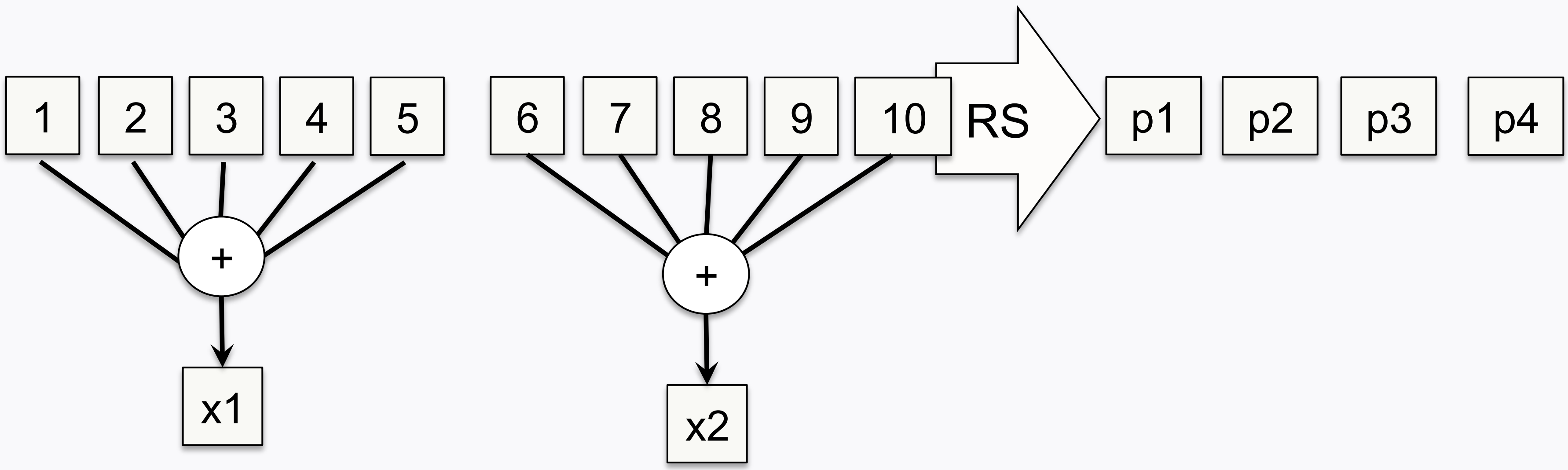
$$d \leq n - k + 1$$

- Reed-Solomon codes achieve the Singleton bound (hence called MDS)

Locality of a code

- A code symbol has locality **r** if it is a function of **r** other codeword symbols.
- A systematic code has **message locality r** if all its systematic symbols have locality r
- A code **has all-symbol locality r** if all its symbols have locality r.
- In an MDS code, all symbols have locality at most r <= k
- Easy lemma: Any MDS code must have trivial locality r=k for every symbol.

Example: code with message locality 5



All k=10 message blocks can be recovered by reading r=5 other blocks.

A single parity block failure requires still 10 reads.

Best distance possible for a code with locality r?

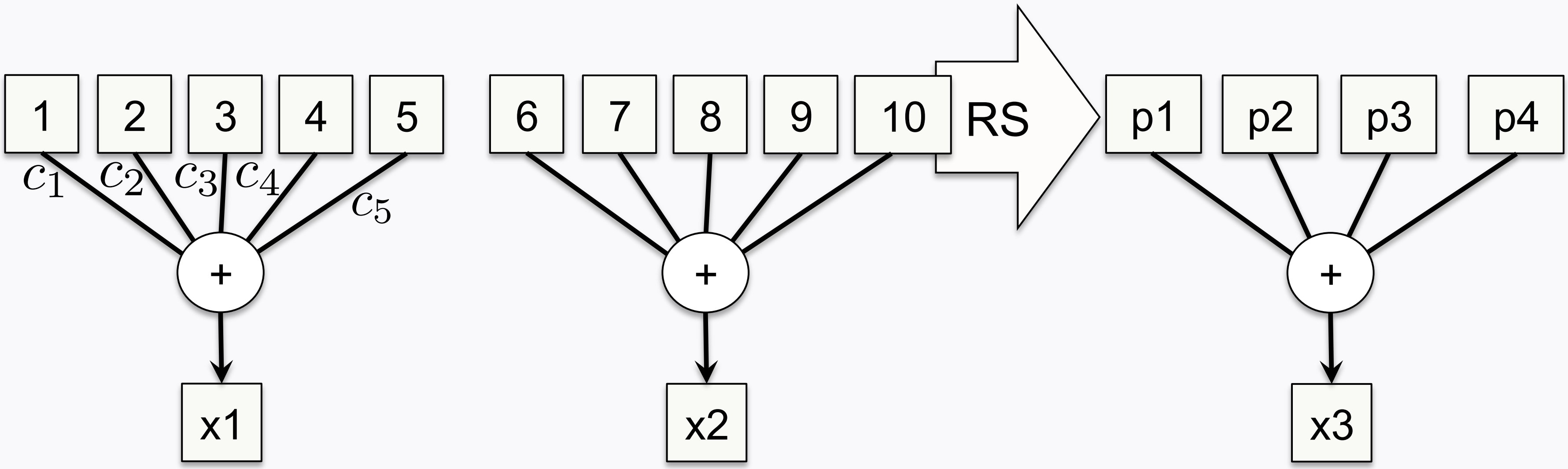
Locality-distance tradeoff

Codes with all-symbol locality r can have distance at most:

$$d \leq n - k - \lceil \frac{k}{r} \rceil + 2$$

- •Shown by Gopalan et al. for scalar linear codes (Allerton 2012)
- •Papailiopoulos et al. information theoretically (ISIT 2012)
- •r=k (trivial locality) gives Singleton Bound.
- •Any non-trivial locality will hurt the fault tolerance of the storage system
- •Pyramid codes (Huang et al) achieve this bound for message-locality
- •Few explicit code constructions known for all-symbol locality.

All-symbol locality



The coefficients need to make the local forks in general position compared to the global parities.

Random works whp in exponentially large field. Checking requires exponential time.

OP2: General Explicit LRCs that are maximally recoverable (MR) are open.

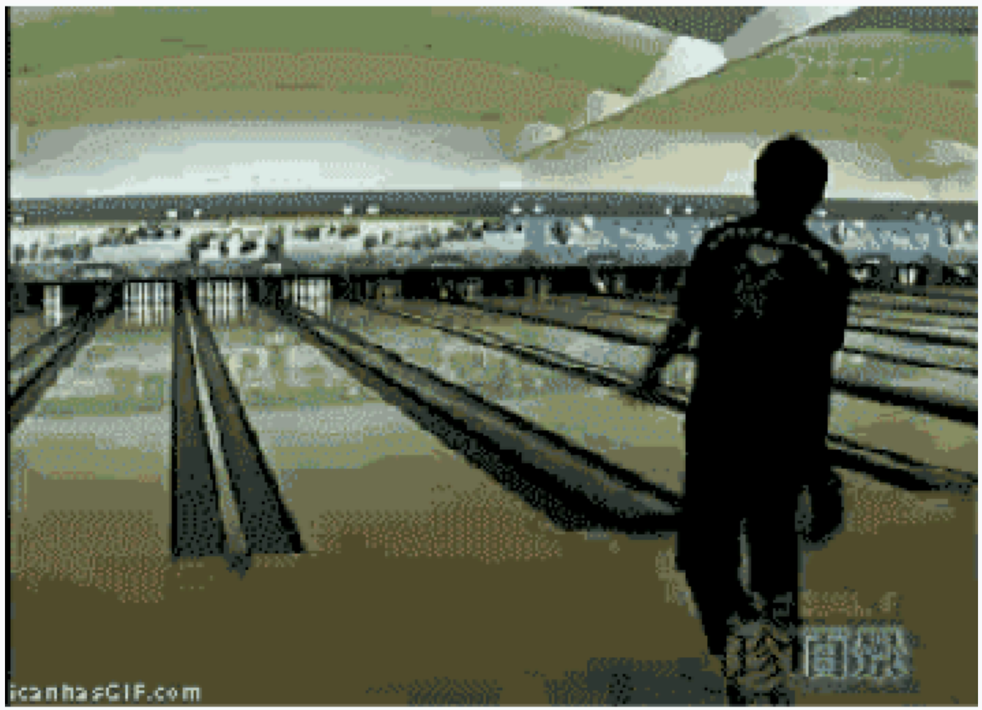
Recent work on LRCs

- Silberstein, Kumar et al., Pamies-Juarez et al.:
  Work on local repair even if multiple failures happen within each group.

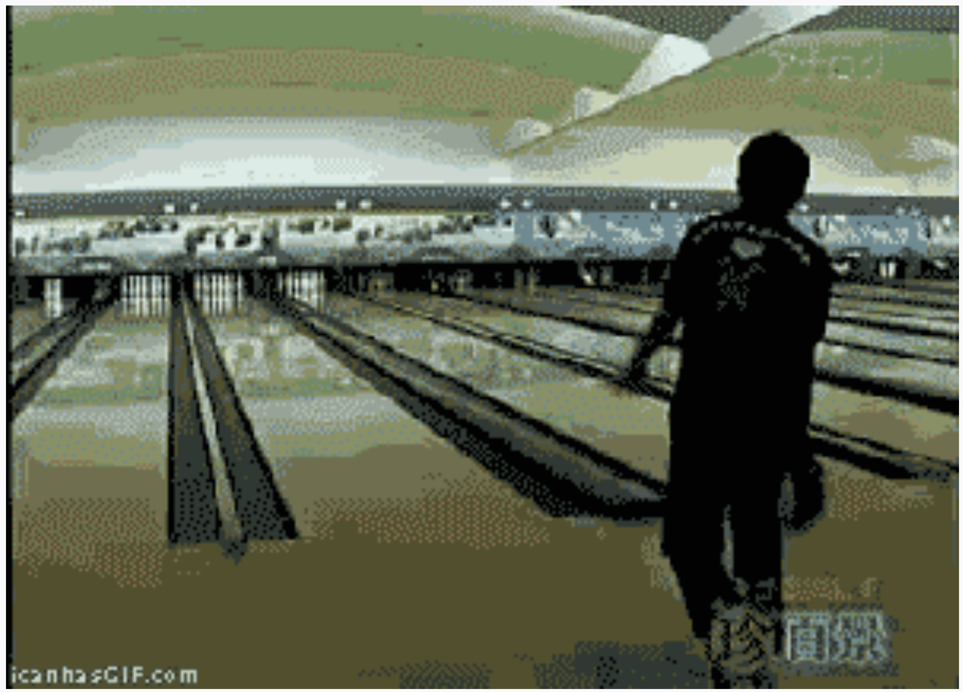
- Tamo et al., Ernvall et al. Explicit LRC constructions (for some values of n,k,r)

- Mazumdar et al. Locality bounds for given field size

Part 3:
Availability: Multiple reads with one code



Part 3:
Availability: Multiple reads with one code

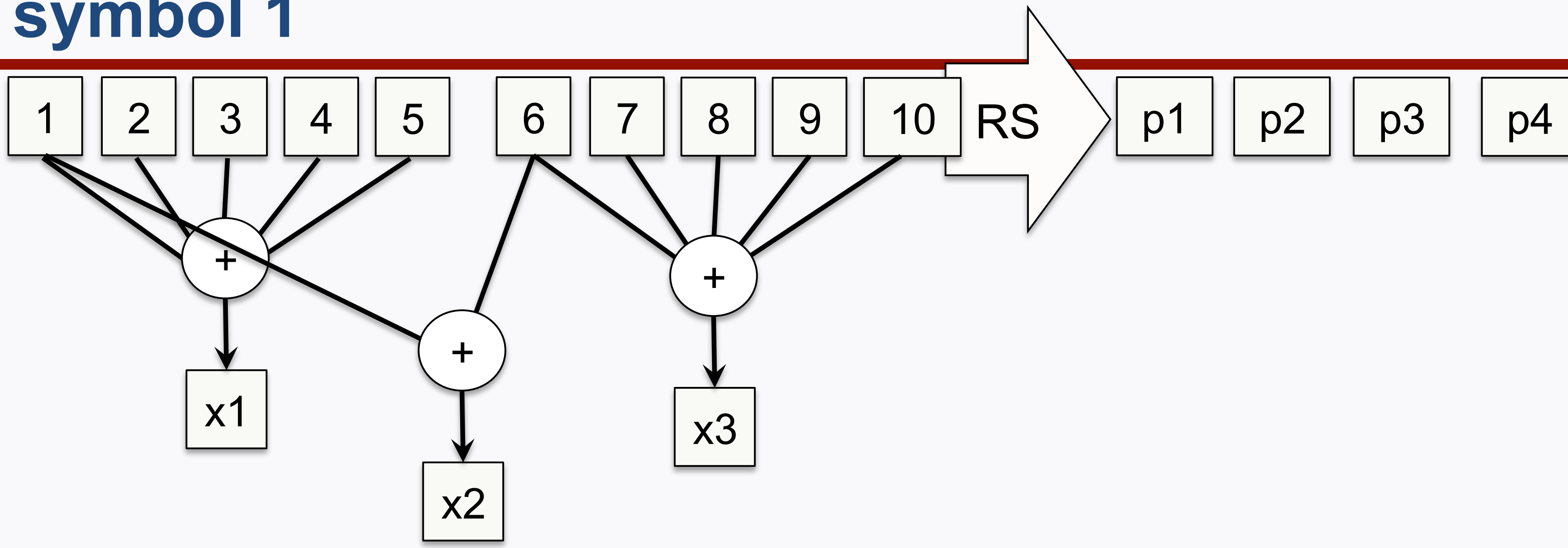


Code Locality r, Code Availability t

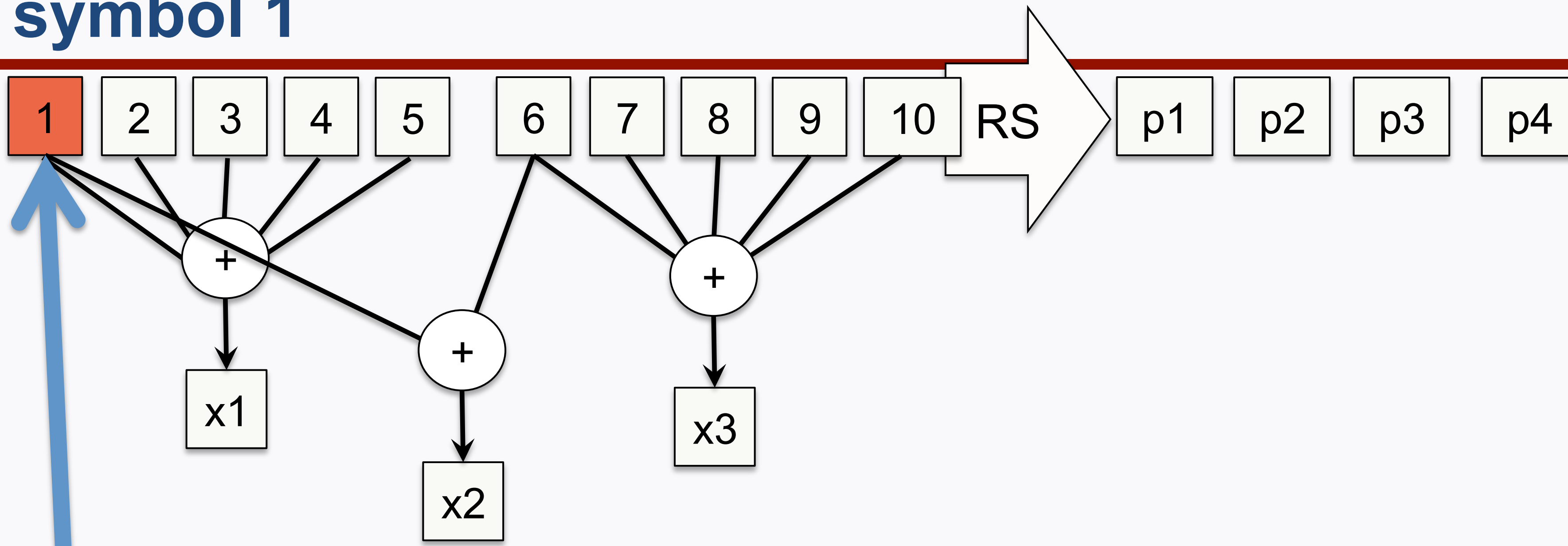
- A symbol has locality **r** if it is a function of **r** other codeword symbols.
- A code **has all-symbol locality r** if all its symbols have locality r.

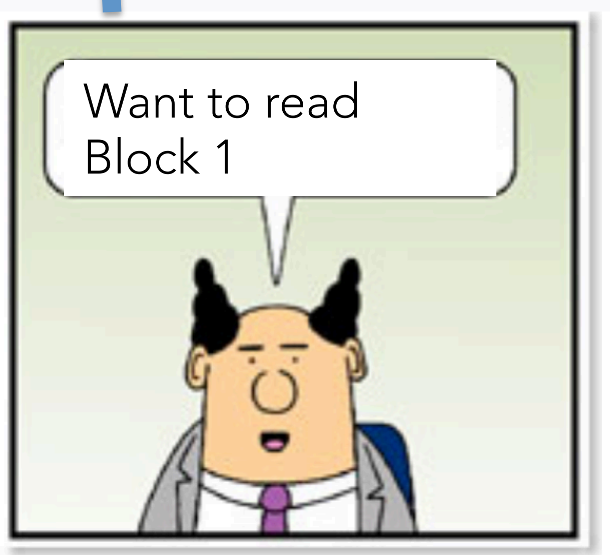
- A symbol has **availability** t if it can be read in parallel by t+1 disjoint groups of symbols.
- These t reads have locality r if they involve up to r symbols each.

Example of Locality r and availability t for symbol 1

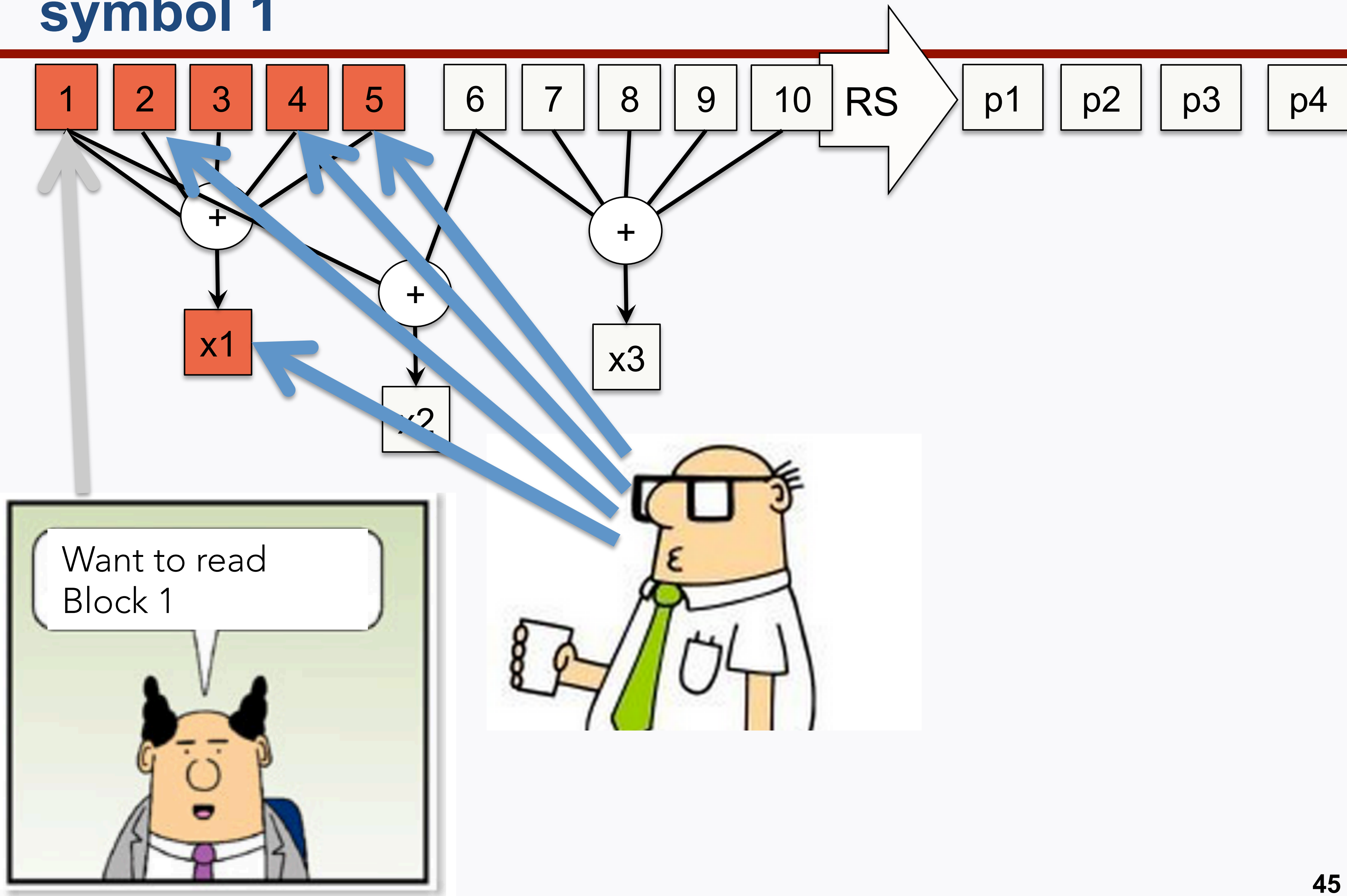


Example of Locality r and availability t for symbol 1

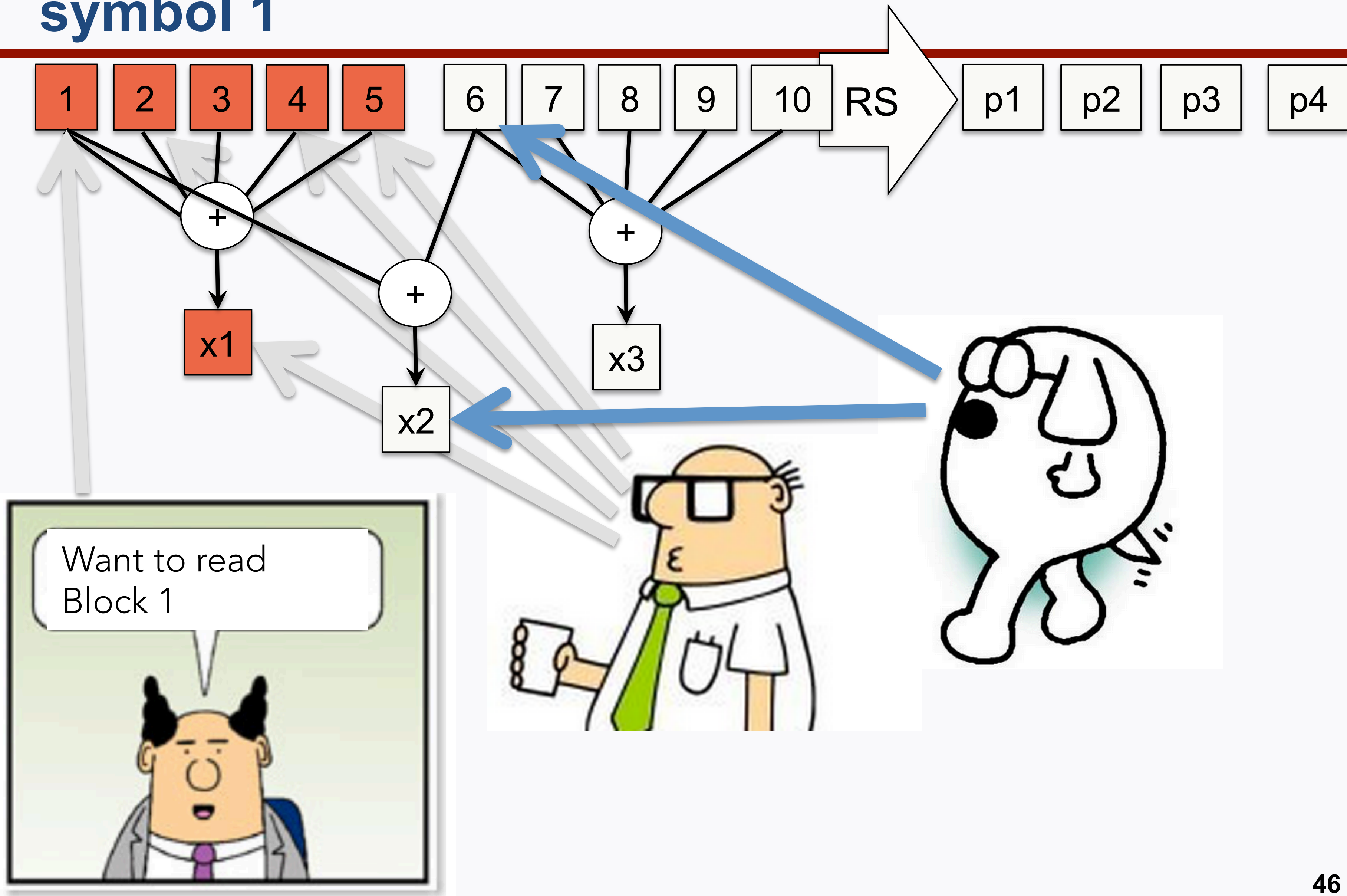




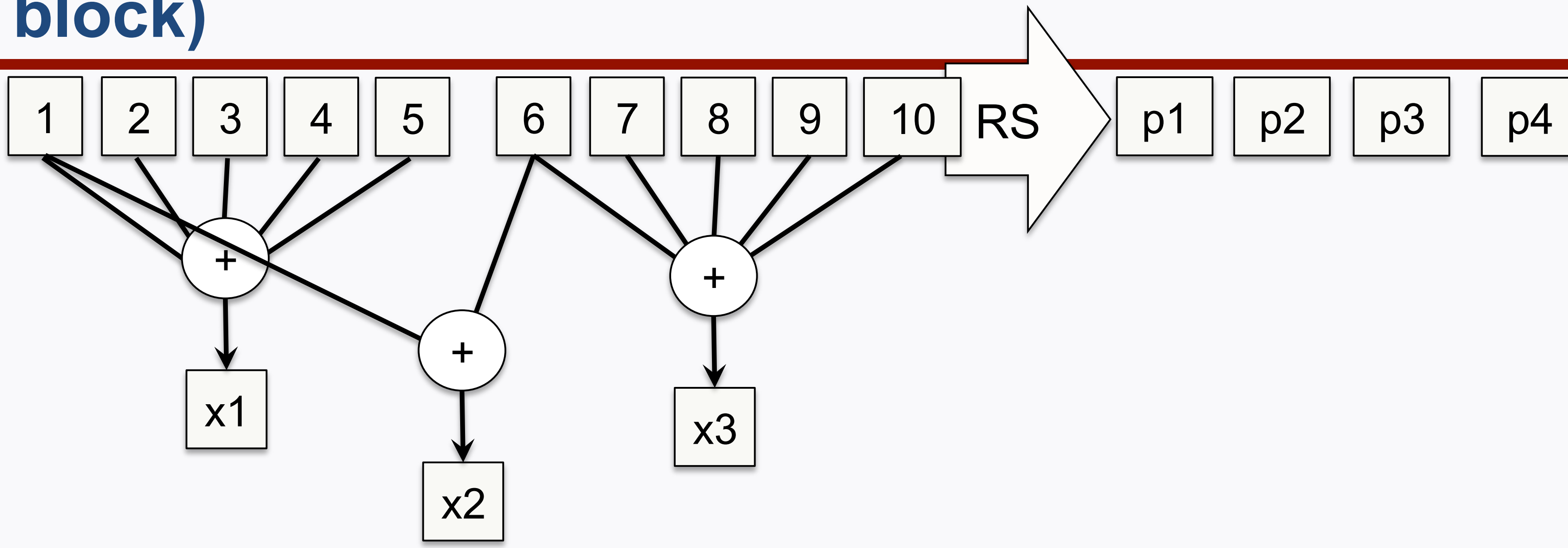
Example of Locality r and availability t for symbol 1



Example of Locality r and availability t for symbol 1

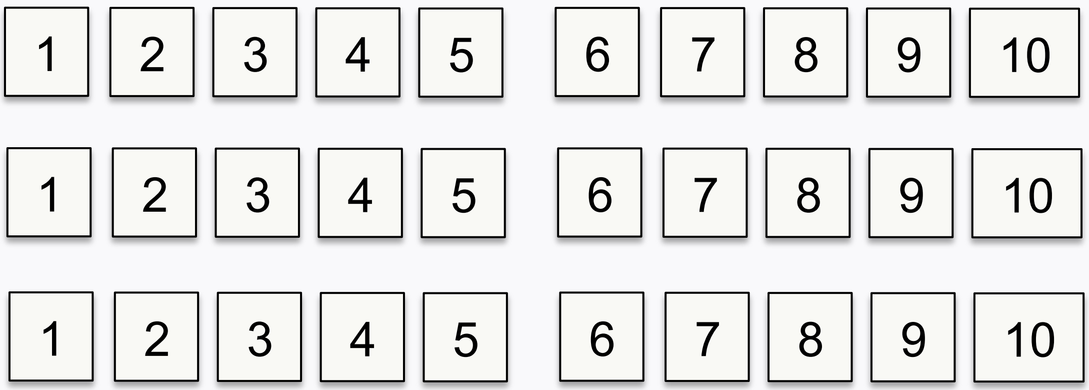


message availability 2 (=2 parallel reads for a block)



- Therefore Block 1 can be read by 1 systematic read + 2 repair reads **simultaneously**
- Block 1 has availability t=2 with groups of locality r1=5 and r2= 2
- Notice also that the group (2,3,4,5,6,7,8,9,10, p1) of locality r=10 can be used to recover 1 (but blocks all others, so not used)

Example: 3 replication



- Each symbol can be read in parallel t+1 =3 times.
- Distance d=3. Rate= 1/3.
- Availability t=2. Locality of these reads r=1.
- If you want to increase availability, rate goes to zero like 1 / (t+1)
- Can we get scaling availability with non-vanishing rate?

Our results

We construct codes with scaling availability and small locality.
For any high rate. With near-MDS distance.

- Polynomial Availability (using Combinatorial designs):

$t= n^{1/3}$
$r=n^{1/3 - \varepsilon}$

- Fundamental Bounds: For a given locality r and availability t requirements, what is the best distance possible?

- We obtain some bounds – Sometimes tight.

Related work

- Locally decodable codes
  (LDCs imply linear availability, $t = c\,n$)

- Batch Codes [Ishai, Kushilevitz, Ostrovsky, Sahai STOC'04].

Very similar parallel reads requirement.

Not good distance.

In fact our results imply the first batch codes with near-MDS distance.
Applicability in cryptography?

Distance vs. Locality-Availability trade-off

New Distance Bound

- For (r, t)-Information local codes*:

$$d_{\min} \leq n - k + 1 - \left(\left\lceil \frac{kt}{r} \right\rceil - t \right)$$

Distance vs. Locality-Availability trade-off

New Distance Bound:

- For (r, t)-Information local codes*:

$$d_{\min} \leq n - k + 1 - \left(\left\lceil \frac{kt}{r} \right\rceil - t \right)$$

*The dirty details:

- We can only prove this for scalar linear codes.
- Only one parity symbol per repair group is assumed.
- For some cases we can achieve this using combinatorial designs.

Conclusions and Open Problems

Which repair metric to optimize?

- Repair BW, All-Symbol Locality, Message-Locality, Fault tolerance, A combination of all?
- Depends on type of storage cluster
 (Cloud, Analytics, Photo Storage, Archival, Hot vs Cold data)

Open problems

1.Repair Bandwidth:

- Exact repair bandwidth region **?**
- Practical E-MSR codes for high rates **?**
- Repairing codes with a small finite field limit **?**
- Better Repair for existing codes (EvenOdd, RDP, Reed-Solomon) **?**

2.Locality:

- Explicit LRCs with Maximum recoverability**?**
- Simple and practical constructions **?**

3.Availability:

- Distance –availability tradeoff **?**
- Practical explicit codes **?**
- Applicability to hot data (photo clusters) **?**

Coding for Storage wiki

[[wiki:definitions:repair_problem]] DISTRIBUTED STORAGE WIKI

Trace: » repair_problem

Show pagesource Old revisions Recent changes Index Login

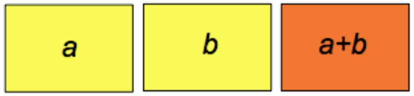
Search

Navigation

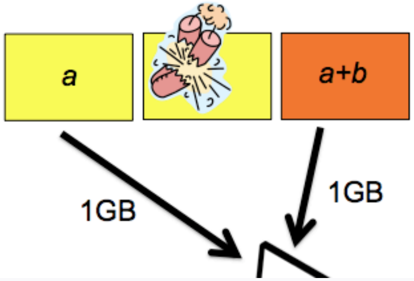
- Homepage
- Papers
- Authors
- Codes
- Software
- Definitions
- Open Problems
- Conferences
- Guidelines
- Playground
- Syntax
- itsociety

The Repair Problem

Consider the very simple (n=3,k=2) binary MDS code shown, which is very simply one disk storing the parity of the others



Clearly, this code has the property it can tolerate any single node failure. This means that even after one node fails, a data collector (shown as a laptop) can communicate the information from the remaining two nodes and reconstruct the file. This is shown here:



Practical Storage Systems and Open Problems

Real systems that use distributed storage codes

- Windows Azure, (Cheng et al. USENIX 2012) (LRC Code)
- Used in production in Microsoft
- CORE (Li et al. MSST 2013) (Regenerating EMSR Code)
- NCCloud (Hu et al. USENIX FAST 2012) (Regenerating Functional MSR)
- ClusterDFS (Pamies Juarez et al.) (SelfRepairing Codes)
- StorageCore (Esmaili et al.) (over Hadoop HDFS)
- HDFS Xorbas (Sathiamoorthy et al. VLDB 2013) (over Hadoop HDFS) (LRC code)
- Testing in Facebook clusters

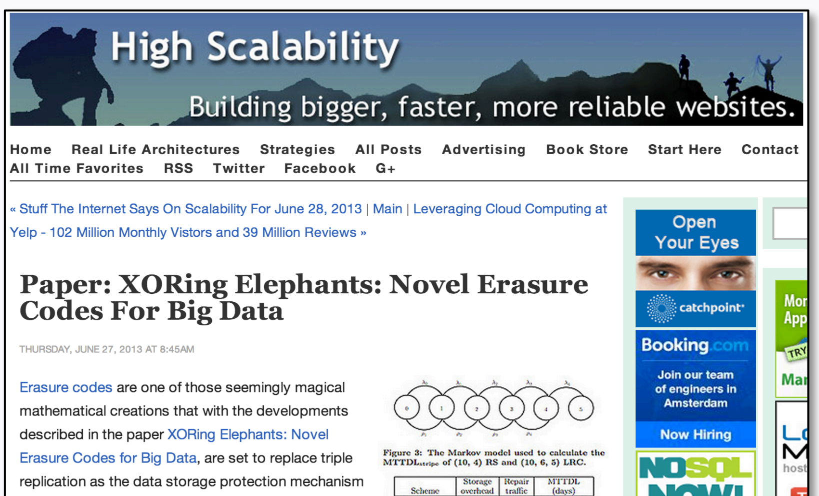
HDFS Xorbas (HDFS with LRC codes)

- Practical open-source Hadoop file system with built in locally repairable code.
- Tested in Facebook analytics cluster and Amazon cluster
- Uses a (16,10) LRC with all-symbol locality 5
- Microsoft LRC has only message-symbol locality
- Saves storage, good degraded read, bad repair.

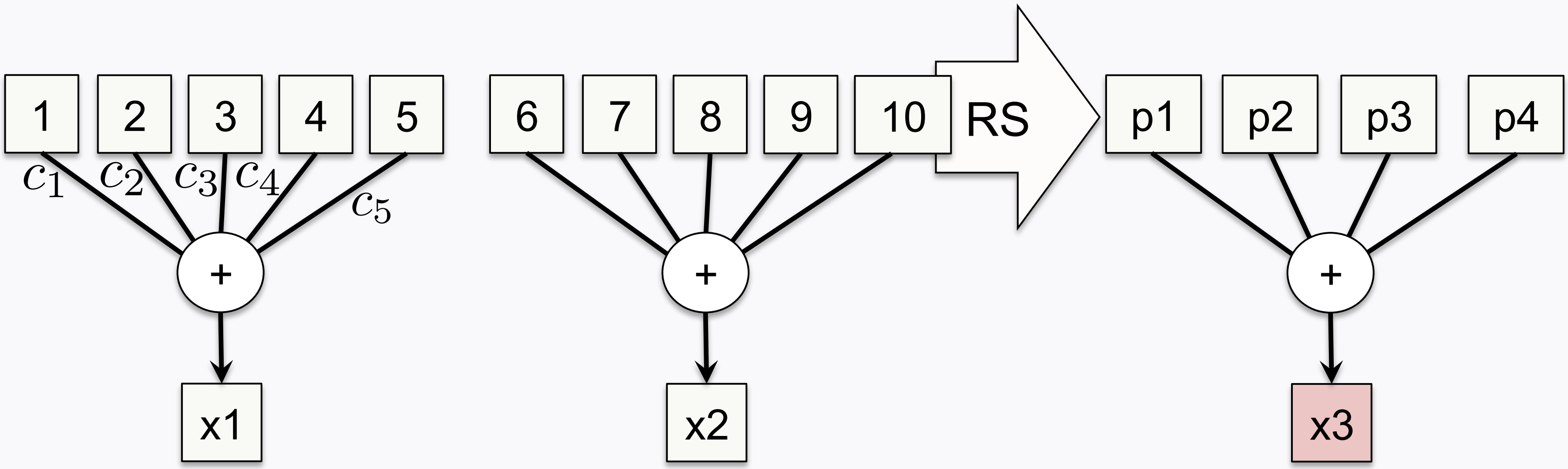
HDFS Xorbas


HOME STARTUPS MOBILE GADGETS EUROPE VIDEO MORE
Adobe Marketing Cloud
Marketing works, and we can prove it.
HOT TOPICS YAHOO APPLE FACEBOOK TWITTER GOOGLE MICROSOFT NSA
Limited Extra Early Bird Tickets Available Until July 15
GET TICKETS NOW
HAVE A TIP, PITCH OR GUEST COLUMN? TELL US.
Check Out Facebook's Nerdy Library Of Its Research Papers
JOSH CONSTINE
Thursday, May 16th, 2013
7 Comments
If subjects like "XORing Elephants: Novel Erasure Codes for Big Data" get you all worked up, you'll dig the "Research Publications At Facebook" site, which collects scientific papers written by Facebook employees and researchers. Ranging from hardcore engineering to the sociology of social networks, the library puts Facebook's open-sourced


ZDNet
White Papers Hot Topics Downloads Reviews Newsletters
US Edition NextGen Networks Data Center M2M BYOD CXO Apple
LEADING EDGE IN CLOU
MUST READ: With Windows 8.1, can Microsoft get its mojo
Topic: Storage Investigate Follow via:
How efficient can storage be?
Summary: We want our data protected from failures. After a failure we want our data back quickly. And we want to pay as little as possible. How?
By Robin Harris for Storage Bits | June 19, 2013 -- 07:00 GMT (00:00 PDT)
Recent research by hyper-scale system managers - mostly Microsoft and Facebook engineers and scientists - has tried to answer that question. And the answers are way better than what we have today.
In XORing Elephants: Novel Erasure Codes for Big Data, authors Maheswaran Sathiamoorthy, Alexandros G. Dimakis, Megasthenis Asteris, Dhruba Borthakur and Dimitris Papailiopoulos of USC and Ramkumar Vadali and Scott Chen of Facebook delve deeply into the issue. Technically it is related to work that Microsoft presented last year.


High Scalability
Building bigger, faster, more reliable websites.
Home Real Life Architectures Strategies All Posts Advertising Book Store Start Here Contact
All Time Favorites RSS Twitter Facebook G+
« Stuff The Internet Says On Scalability For June 28, 2013 | Main | Leveraging Cloud Computing at Yelp - 102 Million Monthly Vistors and 39 Million Reviews »
Paper: XORing Elephants: Novel Erasure Codes For Big Data
THURSDAY, JUNE 27, 2013 AT 8:45AM
Erasure codes are one of those seemingly magical mathematical creations that with the developments described in the paper XORing Elephants: Novel Erasure Codes for Big Data, are set to replace triple replication as the data storage protection mechanism of choice.
Open Your Eyes
catchpoint
Booking.com
Join our team of engineers in Amsterdam
Now Hiring
NOSQL NOW!

code we implemented in HDFS



Single block failures can be repaired by accessing 5 blocks. (vs 10)
Stores 16 blocks
1.6x Storage overhead vs 1.4x in HDFS RAID.

Implemented this in Hadoop (system available on github/madiator)

Java implementation

```java
public void encode(int[] message, int[] parity) {

    assert(message.length == stripeSize && parity.length == paritySizeRS+paritySizeSRC);

    for (int i = 0; i < paritySizeRS; i++) {
        dataBuff[i] = 0;
    }
    for (int i = 0; i < stripeSize; i++) {
        dataBuff[i + paritySizeRS] = message[i];
    }
    GF.remainder(dataBuff, generatingPolynomial);
    for (int i = 0; i < paritySizeRS; i++) {
        parity[paritySizeSRC+i] = dataBuff[i];
    }
    for (int i = 0; i < stripeSize; i++) {
        dataBuff[i + paritySizeRS] = message[i];
    }
    for (int i = 0; i < paritySizeSRC; i++) {
        for (int f = 0; f < simpleParityDegree; f++) {
            parity[i] = GF.add(dataBuff[i*simpleParityDegree+f], parity[i]);
        }
    }


}
```

Some experiments

NameNode 'ip-10-168-201-226.us-west-1.compute.internal:54310'

Started:	Wed Jan 11 23:49:49 UTC 2012
Version:	usc3xor, r
Compiled:	Wed Jan 11 20:12:05 UTC 2012 by root
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

276 files and directories, 1621 blocks = 1897 total. Heap Size is 36.37 MB / 966.69 MB (3%). Commited Heap: 58.69 MB. Init Heap: 16 MB. Non Heap Memory Size is 24.91 MB / 118 MB (21%). Commited Non Heap: 37.5 MB.

WARNING : There are 14 missing blocks. Please check the log or run fsck.

Configured Capacity	:	5.3 TB			
DFS Used	:	100.07 GB			
Non DFS Used	:	284.13 GB			
DFS Remaining	:	4.93 TB			
DFS Used%	:	1.84 %			
DFS Remaining%	:	92.93 %			
DataNodes usages	:	Min %	Median %	Max %	stdev %
		1.5 %	1.85 %	2.15 %	0.15 %
Number of Under-Replicated Blocks	:	0			

DataNode Health:

Live Nodes	37		
In Service		37	
Decommission: Completed		0	
Decommission: In Progress		0	
Dead Nodes	13		
Excluded		0	
Decommission: Completed			0
Decommission: Not Completed			0
Not Excluded		13	

Some experiments

•100 machines on Amazon ec2

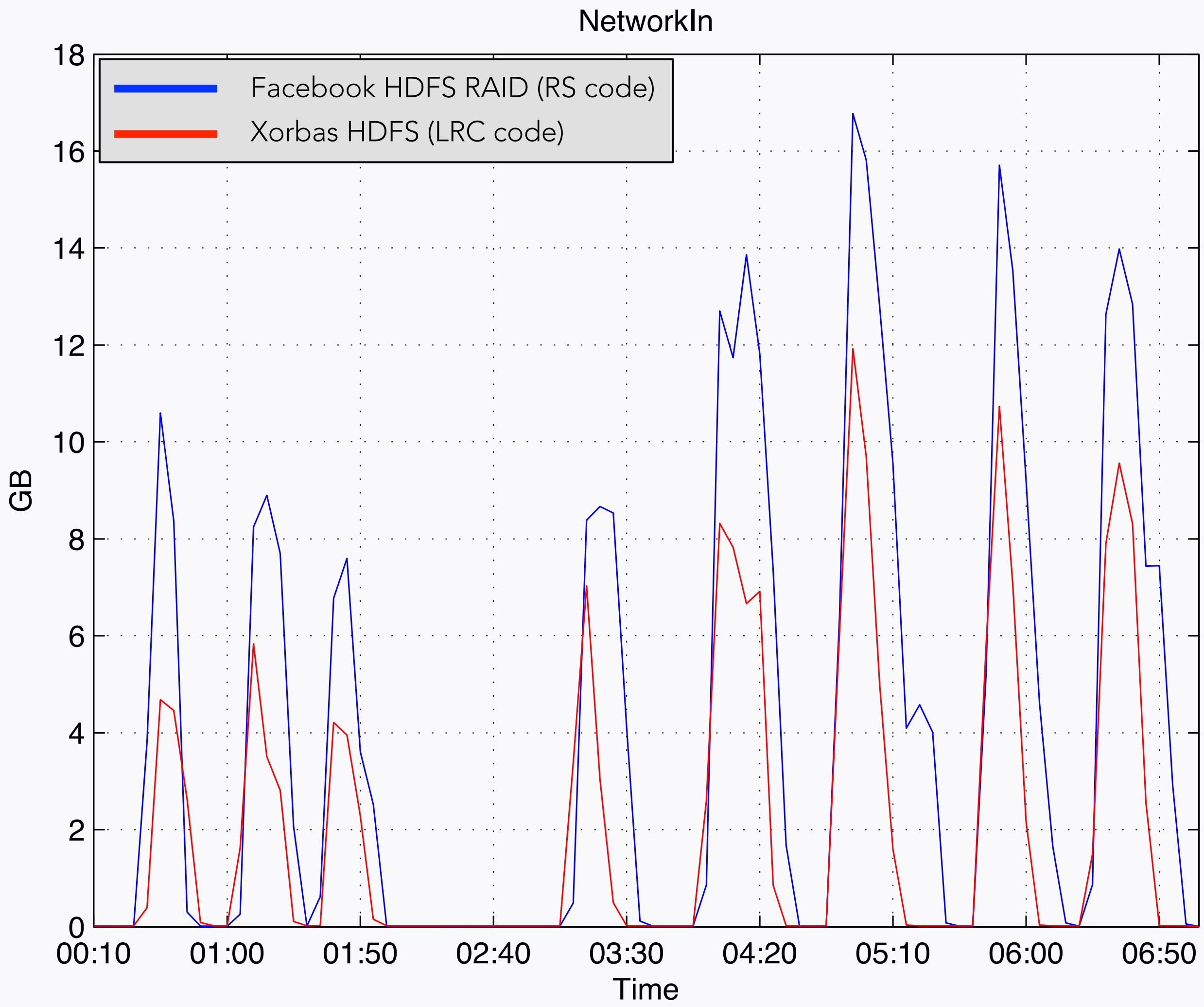
•50 machines running HDFS RAID (facebook version, (14,10) Reed Solomon code)

•50 running our LRC code

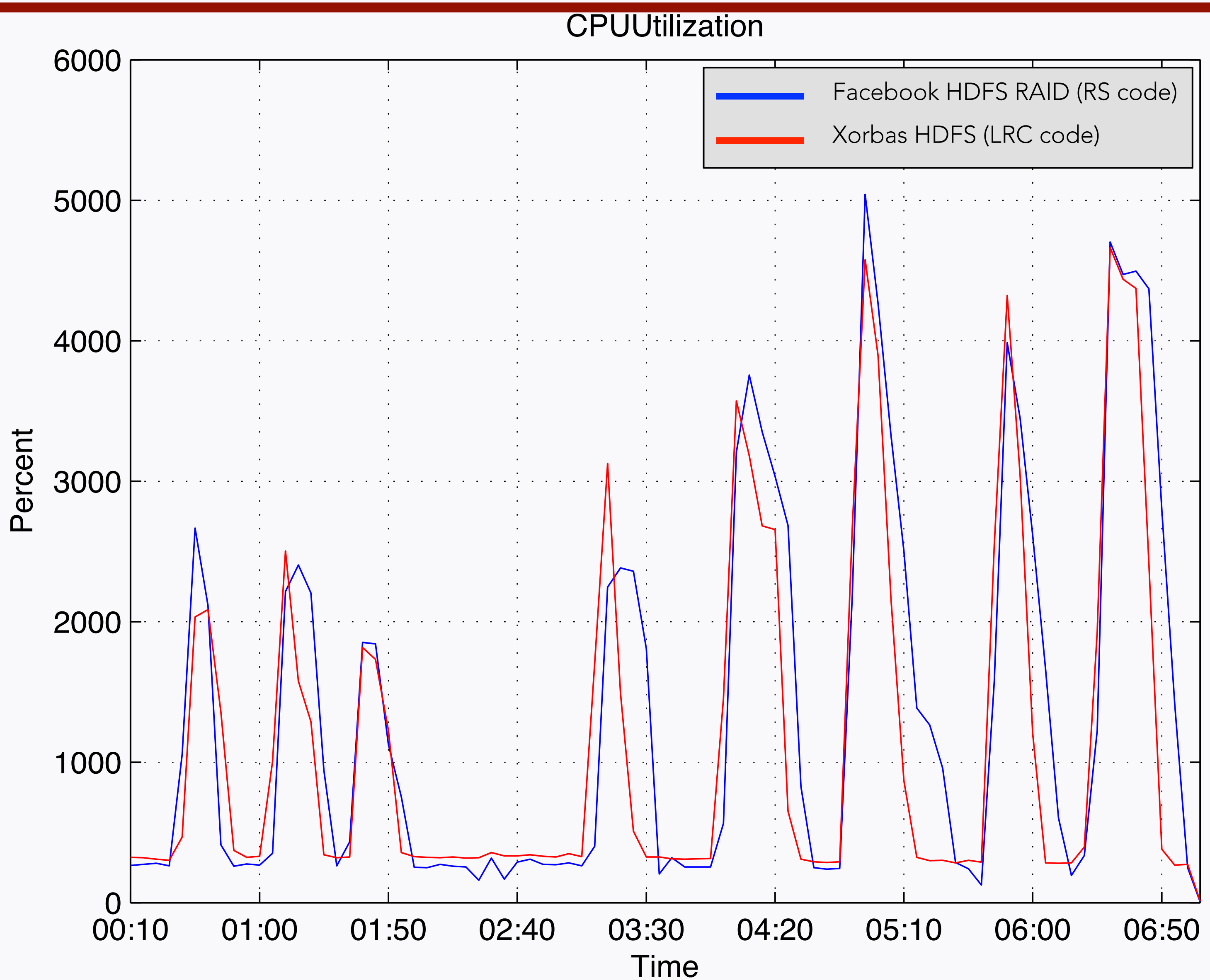
•50 files uploaded on system, 640MB per file

•Killing nodes and measuring network traffic, disk IO, CPU, etc during node repairs.

Repair Network traffic

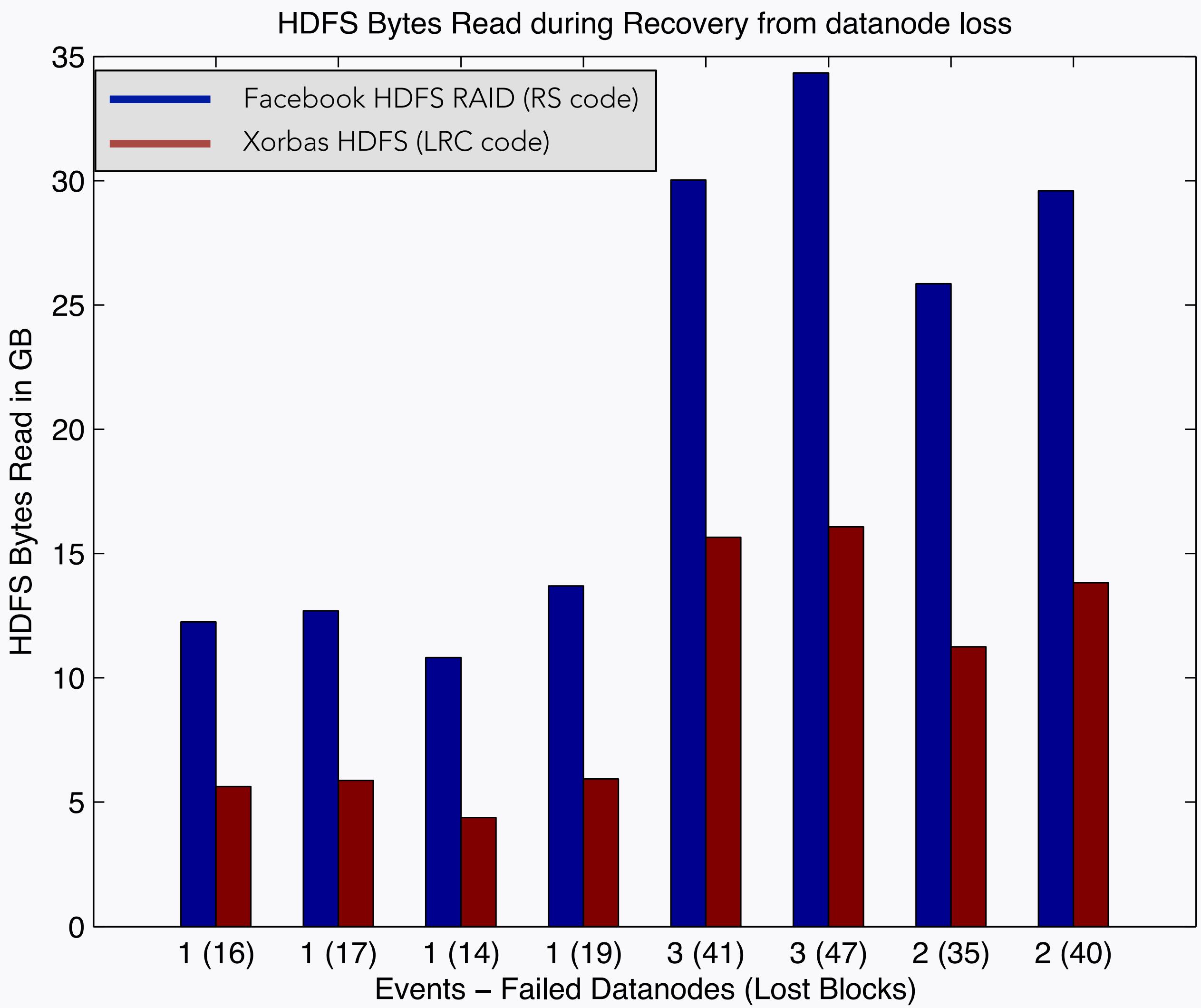


CPU



CPUUtilization
Facebook HDFS RAID (RS code)
Xorbas HDFS (LRC code)
Percent
6000
5000
4000
3000
2000
1000
0
00:10
01:00
01:50
02:40
03:30
04:20
05:10
06:00
06:50
Time

Disk IO



HDFS Bytes Read during Recovery from datanode loss
Facebook HDFS RAID (RS code)
Xorbas HDFS (LRC code)
HDFS Bytes Read in GB
0
5
10
15
20
25
30
35
1 (16)
1 (17)
1 (14)
1 (19)
3 (41)
3 (47)
2 (35)
2 (40)
Events – Failed Datanodes (Lost Blocks)

what we observe

New storage code reduces bytes read by roughly 2.6x

Network bandwidth reduced by approximately 2x

We use 14% more storage. Similar CPU.

In several cases 30-40% faster repairs.

Provides four more zeros of data availability compared to replication

Gains can be much more significant if larger codes are used (i.e. for archival storage systems).

In some cases might be better to save storage, reduce repair bandwidth but lose in locality
(PiggyBack codes: Rashmi et al. USENIX HotStorage 2013)