Week 07

Topic: Important features of Hadoop version 3, Basics of erasure coding, Intra data node balancer

Instructor: Hassan JAHANGIR

Hadoop Version 3.0

- Hadoop is a framework written in Java used to solve Big Data problems. The initial version of Hadoop is released in April 2006. Apache community has made many changes from the day of the first release of Hadoop in the market. The journey of Hadoop started in 2005 by Doug Cutting and Mike Cafarella. The reason behind developing Hadoop is to support distribution for the **Nutch Search Engine Project**.
- Hadoop conquers the supercomputer in the year 2008 and becomes the fastest system ever made by humans to sort terabytes of data stored. Hadoop has come a long way and has accommodated so many changes from its previous version i.e. Hadoop 2.x. In this article, we are going to discuss the changes made by Apache to the Hadoop version 3.x to make it more efficient and faster.

What’s New in Hadoop 3.0?

1. JDK 8.0 is the Minimum JAVA Version Supported by Hadoop 3.x

- Since **Oracle** has ended the use of JDK 7 in 2015, so to use Hadoop 3 users have to upgrade their Java version to JDK 8 or above to compile and run all the Hadoop files. JDK version below 8 is no more supported for using Hadoop 3.

2. Erasure Coding is Supported

- **Erasure coding** is used to recover the data when the computer hard disk fails. It is a high-level RAID(Redundant Array of Independent Disks) technology used by so many IT company's to recover their data. Hadoop file system HDFS i.e. Hadoop Distributed File System uses Erasure coding to provide fault tolerance in the Hadoop cluster. Since we are using commodity hardware to build our Hadoop cluster, failure of the node is normal. Hadoop 2 uses a replication mechanism to provide a similar kind of fault-tolerance as that of Erasure coding in Hadoop 3.

- In Hadoop 2 replicas of the data, blocks are made which is then stored on different nodes in the Hadoop cluster. Erasure coding consumes less or half storage as that of replication in Hadoop 2 to provide the same level of fault tolerance. With the increasing amount of data in the industry, developers can save a large amount of storage with erasure coding. Erasure encoding minimizes the requirement of hard disk and improves the fault tolerance by 50% with the similar resources provided.

3. More Than Two NameNodes Supported

- The previous version of Hadoop supports a single active NameNode and a single standby NameNode. In the latest version of Hadoop i.e. Hadoop 3.x, the data block replication is done among three JournalNodes(JNs). With the help of that, the Hadoop 3.x architecture is more capable to handle fault tolerance than that of its previous version. Big data problems where high fault tolerance is needed, Hadoop 3.x is very useful in that situation. In this Hadoop, 3.x users can manage the number of standby nodes according to the requirement since the facility of multiple standby nodes is provided.
- For example, developers can now easily configure three NameNodes and Five JournalNodes with that our Hadoop cluster is capable to handle two nodes rather than a single one.

4. Shell Script Rewriting

- The Hadoop file system utilizes various shell-type commands that directly interact with the HDFS and other file systems that Hadoop supports i.e. such as WebHDFS, Local FS, S3 FS, etc. The multiple functionalities of Hadoop are controlled by the shell. The shell script used in the latest version of Hadoop i.e. Hadoop 3.x has fixed lots of bugs. Hadoop 3.x shell scripts also provide the functionality of rewriting the shell script.

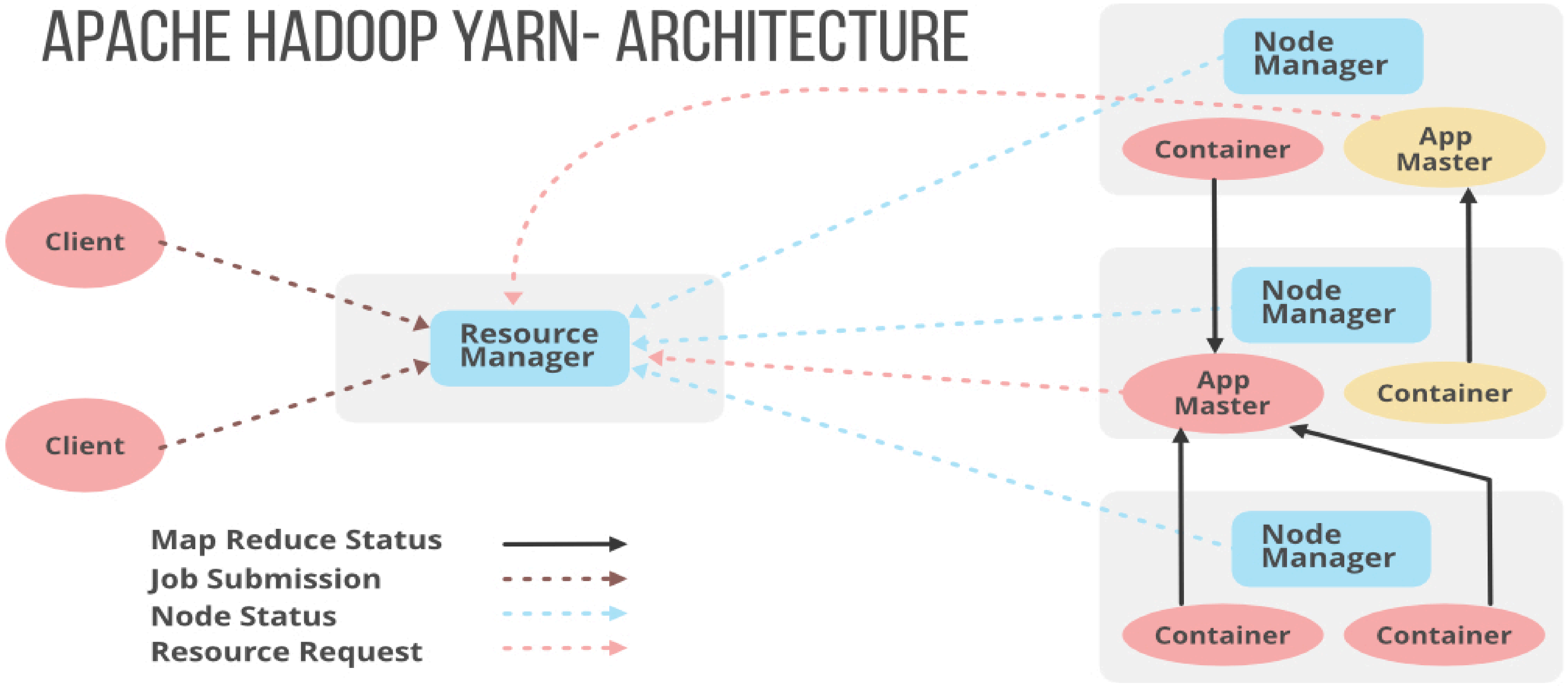
5. Timeline Service v.2 for YARN

- The YARN Timeline service stores and retrieve the applicant’s information(The information can be ongoing or historical). Timeline service v.2 was much important to improve the reliability and scalability of our Hadoop. System usability is enhanced with the help of flows and aggregation. In Hadoop 1.x with **TimeLine service, v.1** users can only make a single instance of reader/writer and storage architecture that can not be scaled further.
- Hadoop 2.x uses distributed writer architecture where data read and write operations are separable. Here distributed collectors are provided for every YARN(Yet Another Resource Negotiator) application. Timeline service v.2 uses HBase for storage purposes which can be scaled to massive size along with providing good response time for reading and writing operations.

The information that Timeline service v.2 stores can be of major 2 types:

- **A. Generic information of the completed application**
- user information
- queue name
- count of attempts made per application
- container information which runs for each attempt on application
- **B. Per framework information about running and completed application**
- count of Map and Reduce Task
- counters
- information broadcast by the developer for TimeLine Server with the help of Timeline client.

APACHE HADOOP YARN- ARCHITECTURE



6.Intra-Datanode Balancer

- DataNodes are utilized in the Hadoop cluster for storage purposes. The DataNodes handles multiple disks at a time. This Disk’s got filled evenly during write operations. Adding or Removing the disk can cause significant skewness in a DataNode. The existing **HDFS-BALANCER** can not handle this significant skewness, which concerns itself with inter-, not intra-, DN skew. The latest **intra-DataNode balancing** feature can manage this situation which is invoked with the help of HDFS disk balancer CLI.

7. Shaded Client Jars

- The new **Hadoop–client-API** and **Hadoop-client-runtime** are made available in Hadoop 3.x which provides Hadoop dependencies in a single packet or single jar file. In Hadoop 3.x the Hadoop –client-API have compile-time scope while Hadoop-client-runtime has runtime scope. Both of these contain third-party dependencies provided by Hadoop-client. Now, the developers can easily bundle all the dependencies in a single jar file and can easily test the jars for any version conflicts. using this way, the Hadoop dependencies onto application classpath can be easily withdrawn.

8. Task Heap and Daemon Management

- In Hadoop version 3.x we can easily configure Hadoop daemon heap size with some newly added ways. With the help of the memory size of the host auto-tuning is made available. Instead of **HADOOP_HEAPSIZE**, developers can use the **HEAP_MAX_SIZE** and **HEAP_MIN_SIZE** variables. **JAVA_HEAP_SIZE** internal variable is also removed in this latest Hadoop version 3.x. Default heap sizes are also removed which is used for auto-tuning by JVM(Java Virtual Machine). If you want to use the older default then enable it by configuring **HADOOP_HEAPSIZE_MAX** in Hadoop-env.sh file.

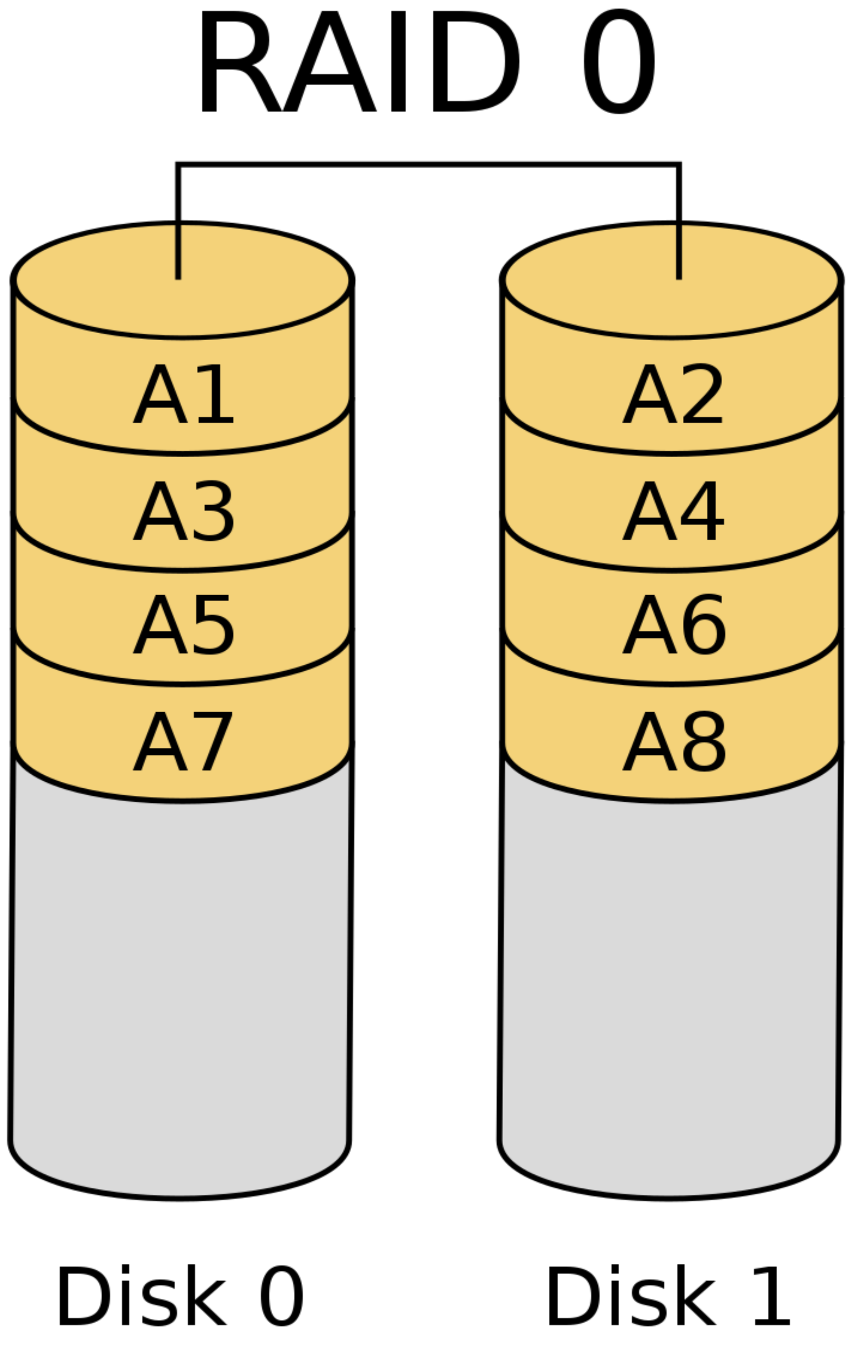
Lets understand RAID first to understand Erasure Coding

RAID

- In computer storage, the **standard RAID levels** comprise a basic set of RAID ("redundant array of independent disks" or "redundant array of inexpensive disks") configurations that employ the techniques of striping, mirroring, or parity to create large reliable data stores from multiple general-purpose computer hard disk drives (HDDs).

RAID 0

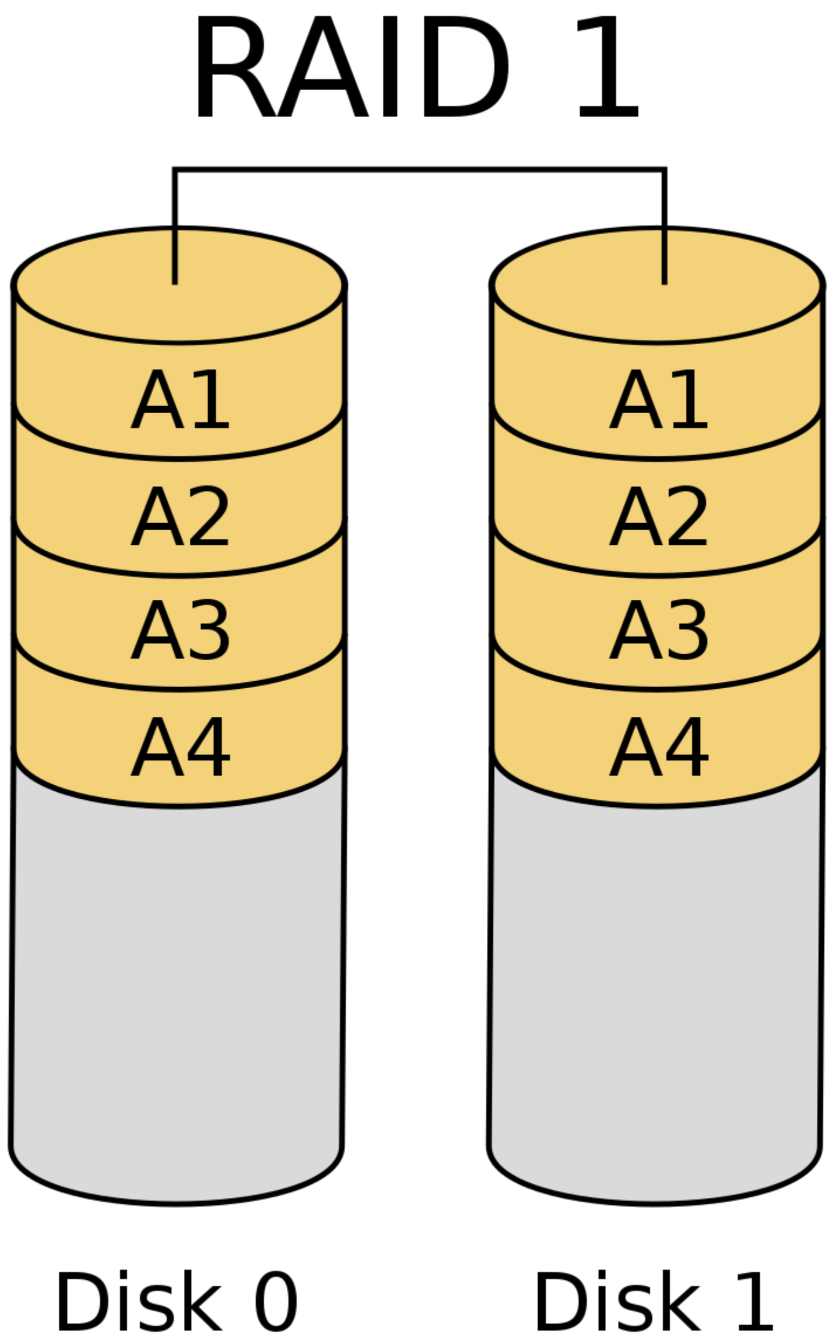
- **RAID 0** (also known as a *stripe set* or *striped volume*) splits ("stripes") data evenly across two or more disks, without parity information, redundancy, or fault tolerance. Since RAID 0 provides no fault tolerance or redundancy, the failure of one drive will cause the entire array to fail, due to data being striped across all disks. This configuration is typically implemented having speed as the intended goal.RAID 0 is normally used to increase performance, although it can also be used as a way to create a large logical volume out of two or more physical disks.



RAID 1

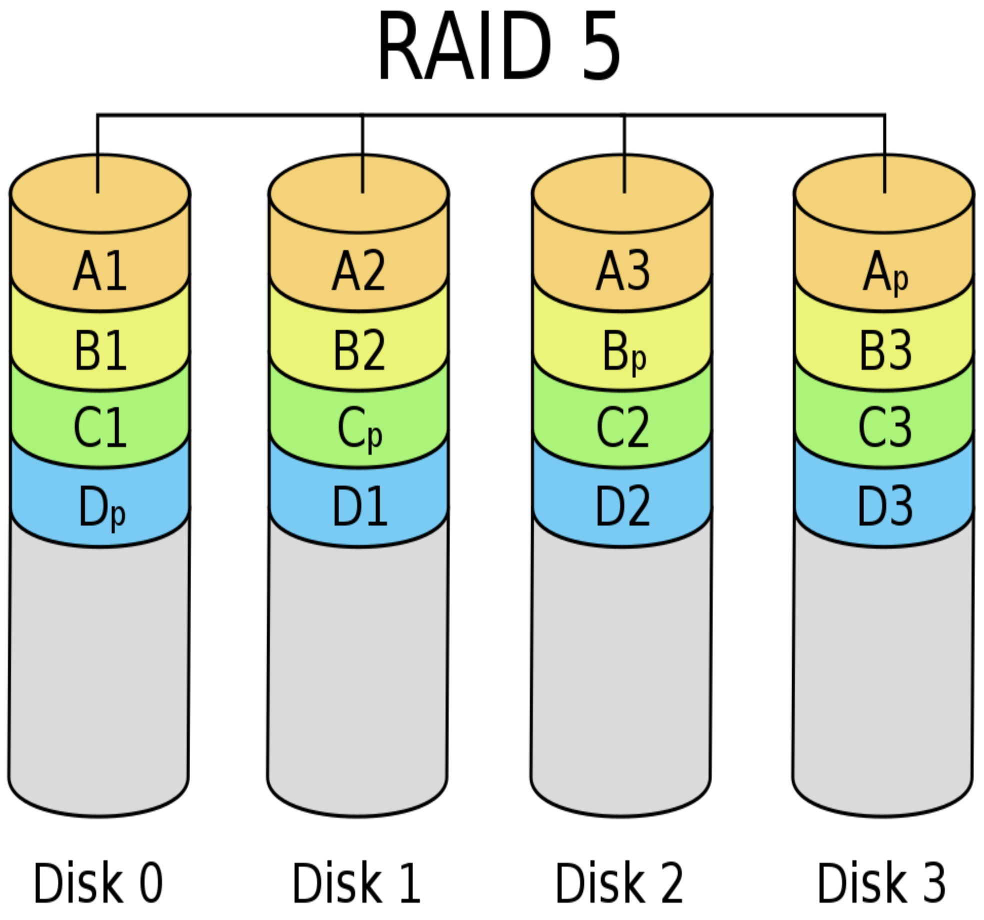
- **RAID 1** consists of an exact copy (or *mirror*) of a set of data on two or more disks; a classic RAID 1 mirrored pair contains two disks. This configuration offers no parity, striping, or spanning of disk space across multiple disks, since the data is mirrored on all disks belonging to the array, and the array can only be as big as the smallest member disk. This layout is useful when read performance or reliability is more important than write performance or the resulting data storage capacity.

The array will continue to operate so long as at least one member drive is operational.



RAID 5

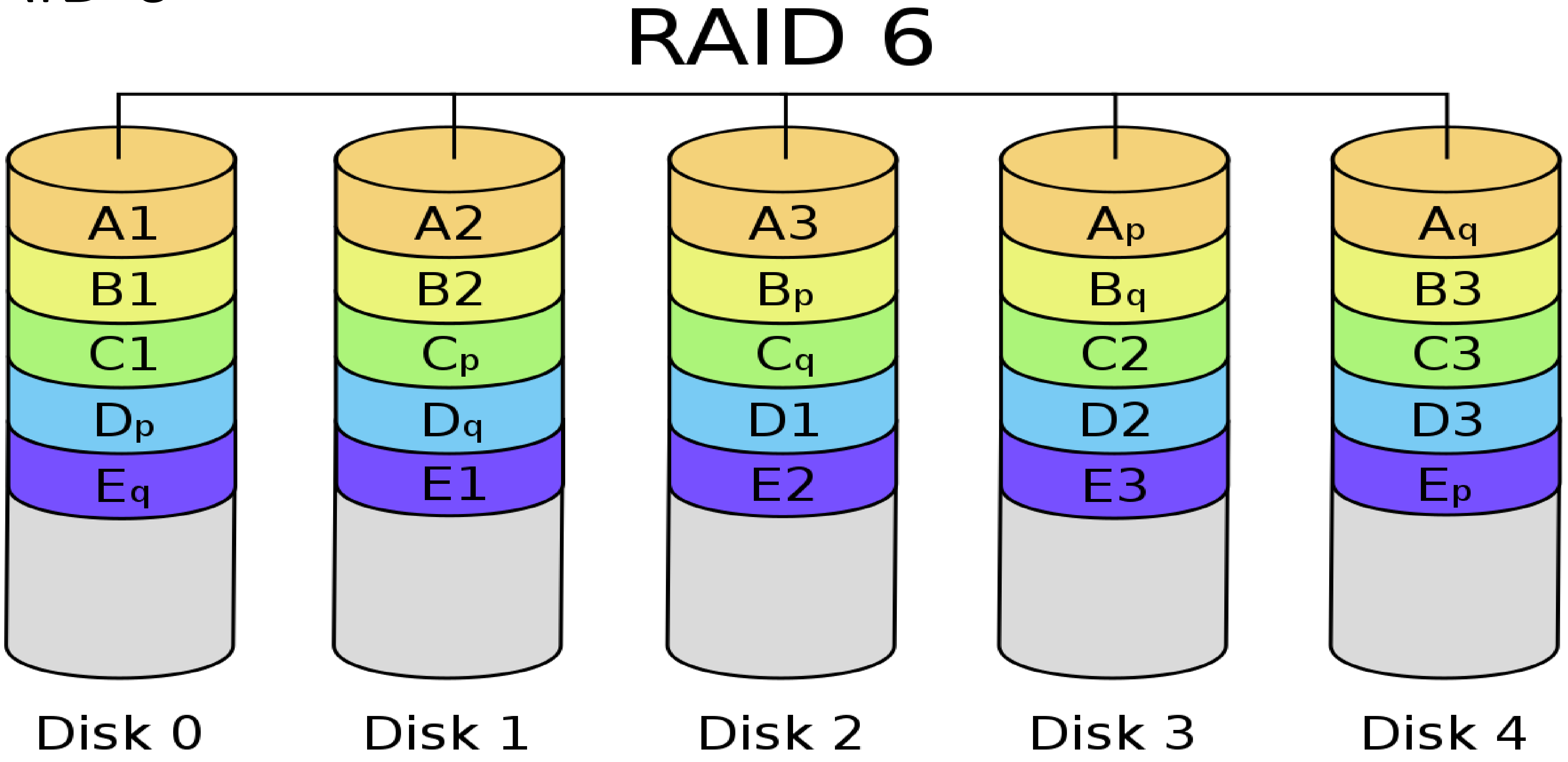
- **RAID 5** consists of block-level striping with distributed parity. Unlike in RAID 4, parity information is distributed among the drives. It requires that all drives but one be present to operate. Upon failure of a single drive, subsequent reads can be calculated from the distributed parity such that no data is lost.RAID 5 requires at least three disks.
- There are many layouts of data and parity in a RAID 5 disk drive array depending upon the sequence of writing across the disks,that is:
- the sequence of data blocks written, left to right or right to left on the disk array, of disks 0 to N.
- the location of the parity block at the beginning or end of the stripe.
- the location of the first block of a stripe with respect to parity of the previous stripe.



RAID 6

- **RAID 6** extends RAID 5 by adding another parity block; thus, it uses block-level striping with two parity blocks distributed across all member disks.
- As in RAID 5, there are many layouts of RAID 6 disk arrays depending upon the direction the data blocks are written, the location of the parity blocks with respect to the data blocks and whether or not the first data block of a subsequent stripe is written to the same drive as the last parity block of the prior stripe. The figure to the right is just one of many such layouts.
- According to the Storage Networking Industry Association (SNIA), the definition of RAID 6 is: "Any form of RAID that can continue to execute read and write requests to all of a RAID array's virtual disks in the presence of any two concurrent disk failures. Several methods, including dual check data computations (parity and Reed–Solomon), orthogonal dual parity check data and diagonal parity, have been used to implement RAID Level 6."

RAID 6


RAID 6
A1
B1
C1
Dp
Eq
A2
B2
Cp
Dq
E1
A3
Bp
Cq
D1
E2
Ap
Bq
C2
D2
E3
Aq
B3
C3
D3
Ep
Disk 0
Disk 1
Disk 2
Disk 3
Disk 4

Erasure Codes for Systems

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

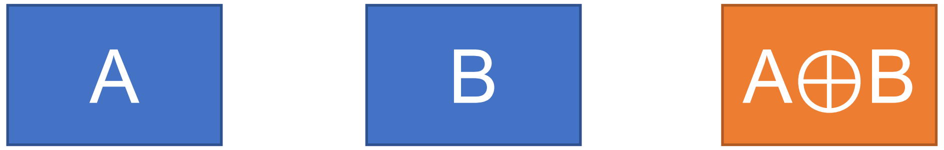
- Erasure coding
  - Store extra information we can use to recover the data
  - Fault tolerance with less storage overhead

Erasure Codes vs Error Correcting Codes

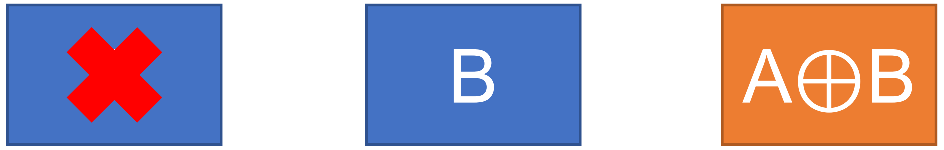
- Error correcting code (ECC):
  - Protects against **errors** is data, i.e., silent corruptions
  - Bit flips can happen in memory -> use ECC memory
  - Bits can flip in network transmissions -> use ECCs

- Erasure code:
  - Data is **erased**, i.e., we know it’s not there
  - Cheaper/easier than ECC
    - Special case of ECC
  - What we’ll discuss today and use in practice
    - Protect against errors with checksums

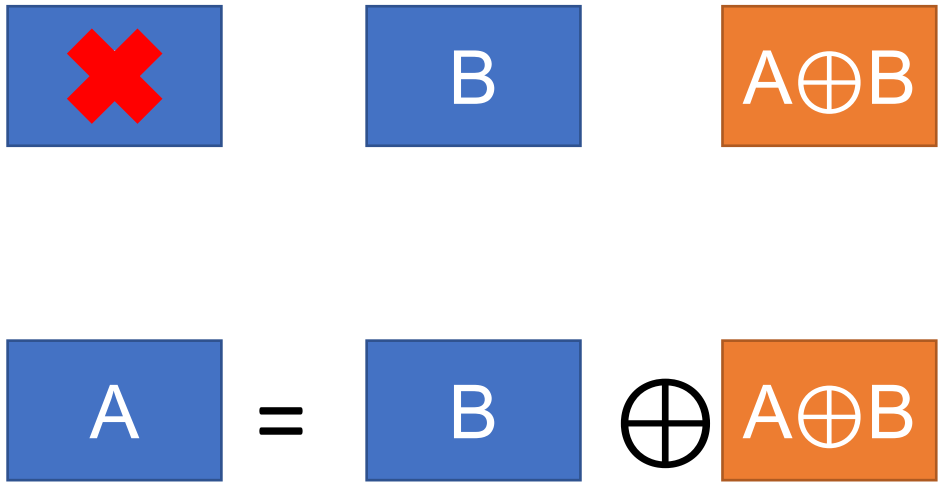
Erasure Codes, a simple example w/ XOR



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
- Details aren’t important for us ☺
- 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.”

What is erasure coding?

- Erasure Coding is a storage data protection method that leverages advanced mathematics to allow file system software to regenerate missing data using pieces of known data called *parity blocks*. As we will explain below, erasure coding offers superior data protection to a mirror copy mainly because it doesn’t require a full second copy of the data, yet can restore any missing portion.

Qumulo

- Qumulo is an American data storage company based in Seattle, Washington. Founded in 2012, it offers products and services to help other companies manage and curate large amounts of data.

Erasure coding

- Qumulo Core architecture is built around Qumulo Scalable Block Store (SBS), which is the foundation layer that enables efficient block-based data protection with erasure coding.
- Erasure coding is entirely different from RAID and solves RAID’s shortcomings. Unlike RAID striping or mirroring, erasure coding is scalable protection for massive data storage, far more performant, more configurable, and more space-efficient, allowing clusters unlimited growth while maintaining full data protection and responsiveness.
- Erasure coding uses advanced mathematics (i.e. the Reed-Solomon formula, in this case) to enable regeneration of missing data from pieces of known data (parity blocks).
- So, unlike RAID mirroring which requires a complete second copy, erasure coding allows greater efficiency, requiring just one parity block for every three data blocks (called 3,2 encoding).

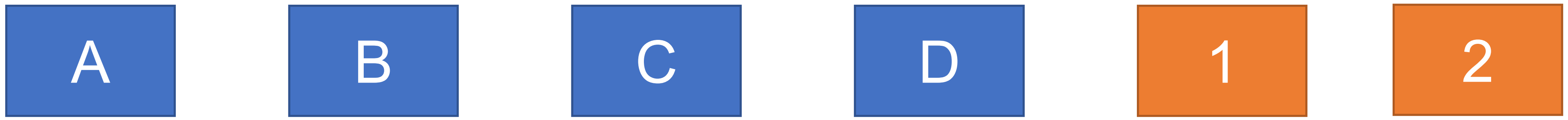
Erasure coding explained (examples)

- Erasure coding is easiest to understand with examples. Here is our 3,2 encoding example:
-
- In a 3,2 encoding, three blocks (m = 3) are spread across three distinct physical devices. Blocks 1 and 2 contain the user data we want to protect (n = 2), and the third is called a parity block. The contents of the parity block are calculated using the erasure coding algorithm.
- Since each block is written to a separate drive, any one of the three drives could fail and the information stored in blocks 1 and 2 is still safe because it can be recreated from the parity block.

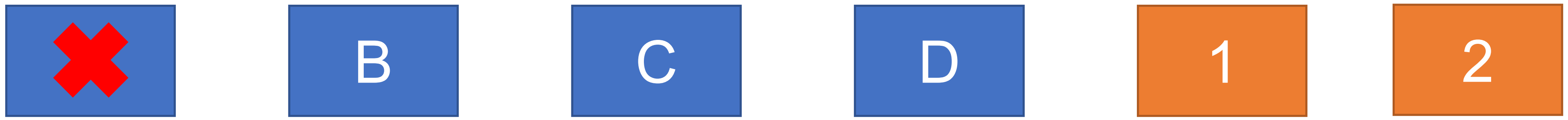
How erasure coding works

- *Here's how it works.* If data block 1 is available, the system simply reads it. The same is true for data block 2. However, if data block 1 is missing, the erasure coding system reads data block 2, plus the parity block, and reconstructs the value of data block 1.
- Similarly, if data block 2 resides on the failed disk, the system reads data block 1 and the parity block. SBS always makes sure that the blocks are on different spindles so the system can read from blocks simultaneously.
- A 3,2 encoding has efficiency of 2 / 3 (n/m), or 67%. While it is better than the 50% efficiency of mirroring, 3,2 encoding can still only protect against a single disk failure.

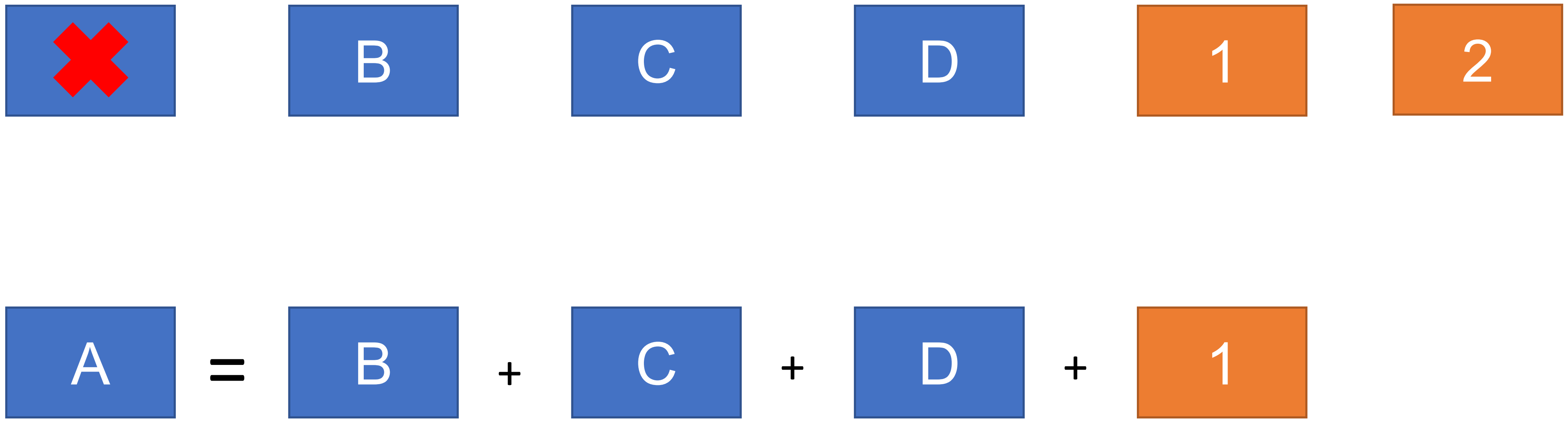
Reed-Solomon (4,2) Example



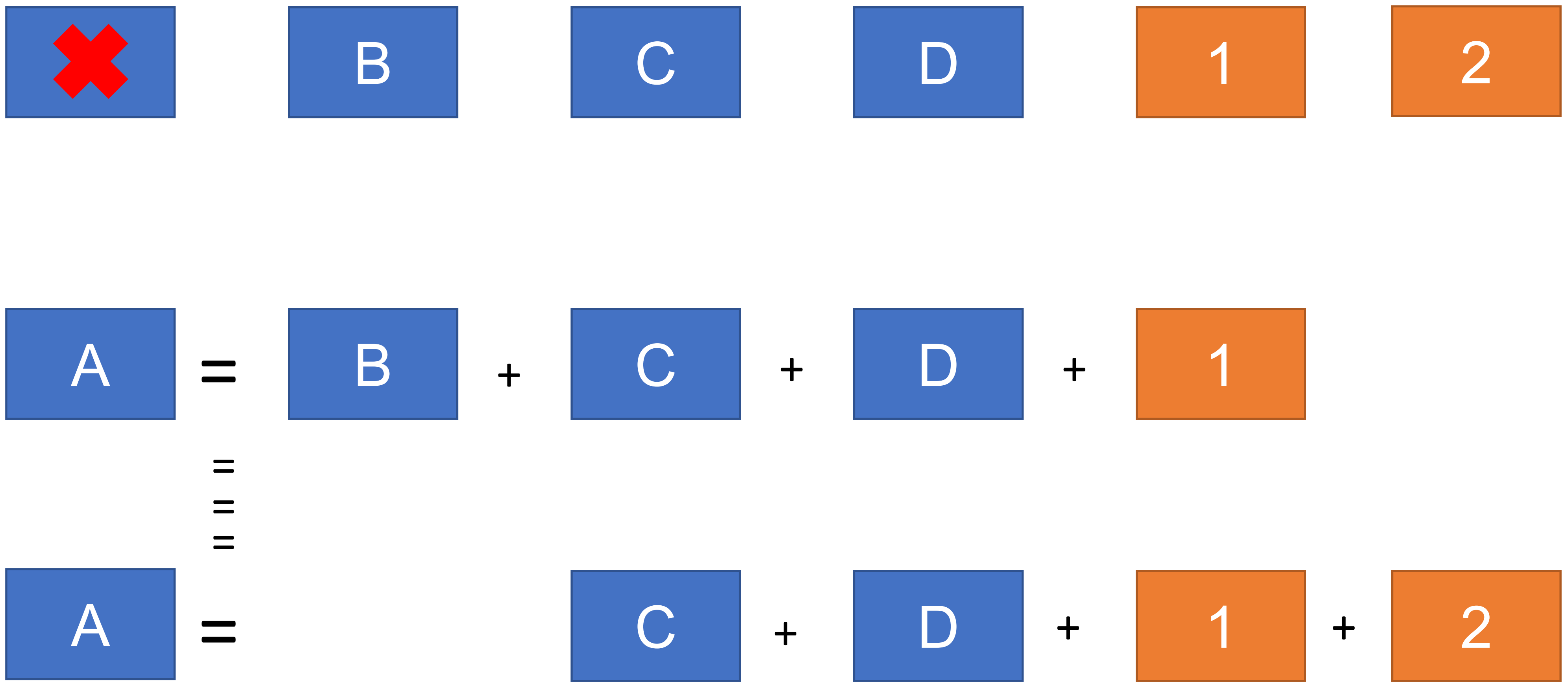
Reed-Solomon (4,2) Example



Reed-Solomon (4,2) Example



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

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:

- Erasure coding

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

What’s the Catch?

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

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

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