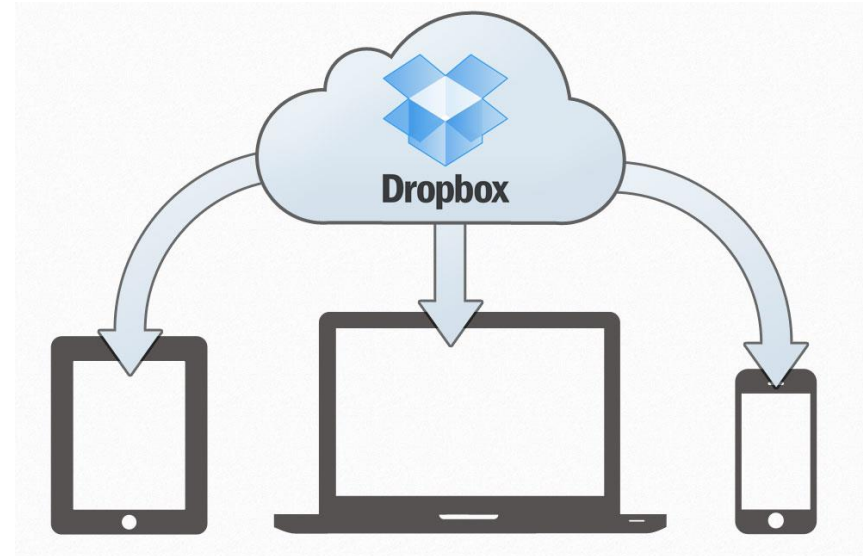
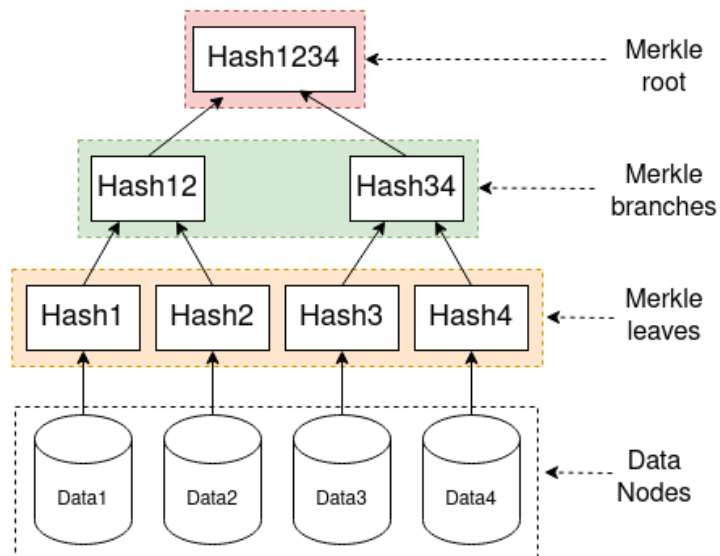




Data Structures

Programming Project #4

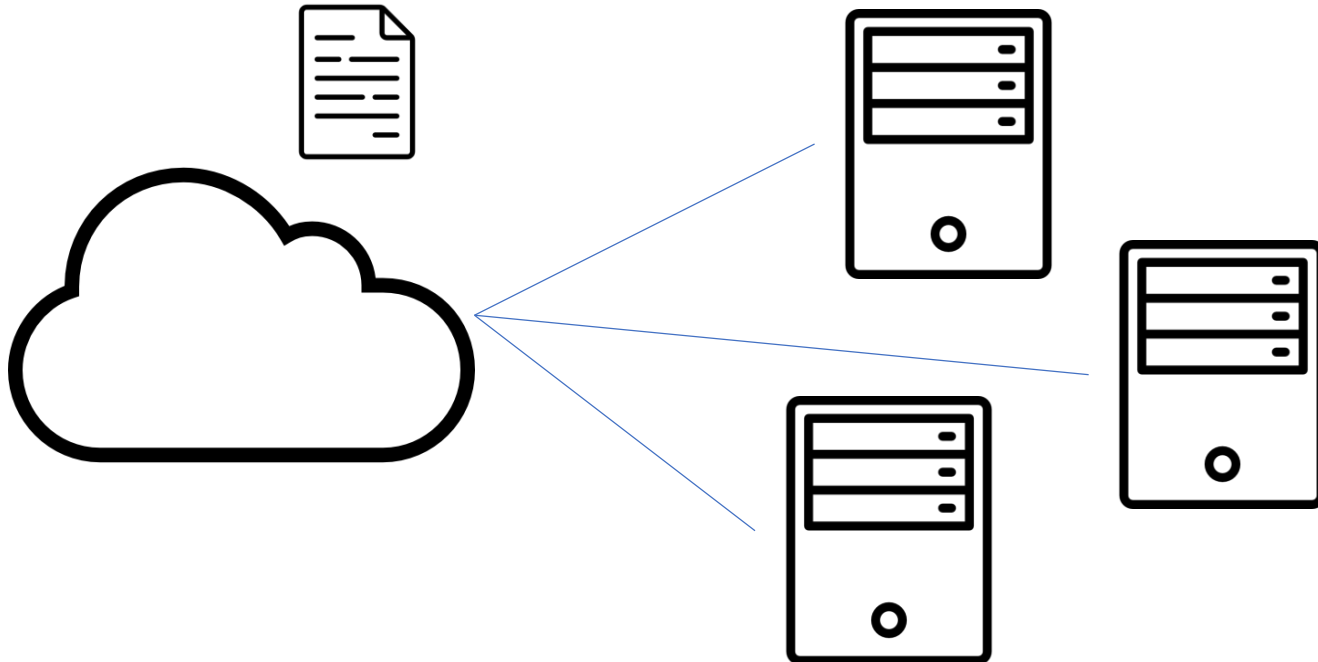


Data Corruption and Verification in Distributed Systems

- A **distributed system** consists of a **master** and several **servers**
- It has to **detect data corruption in servers** and fix the issues on the fly
- Widely happen in **many applications**:
- Cloud Storage: Dropbox
- P2P file sharing: BitTorrent
- Decentralized Digit Currency: Bitcoin
- Distributed Database: Cassandra

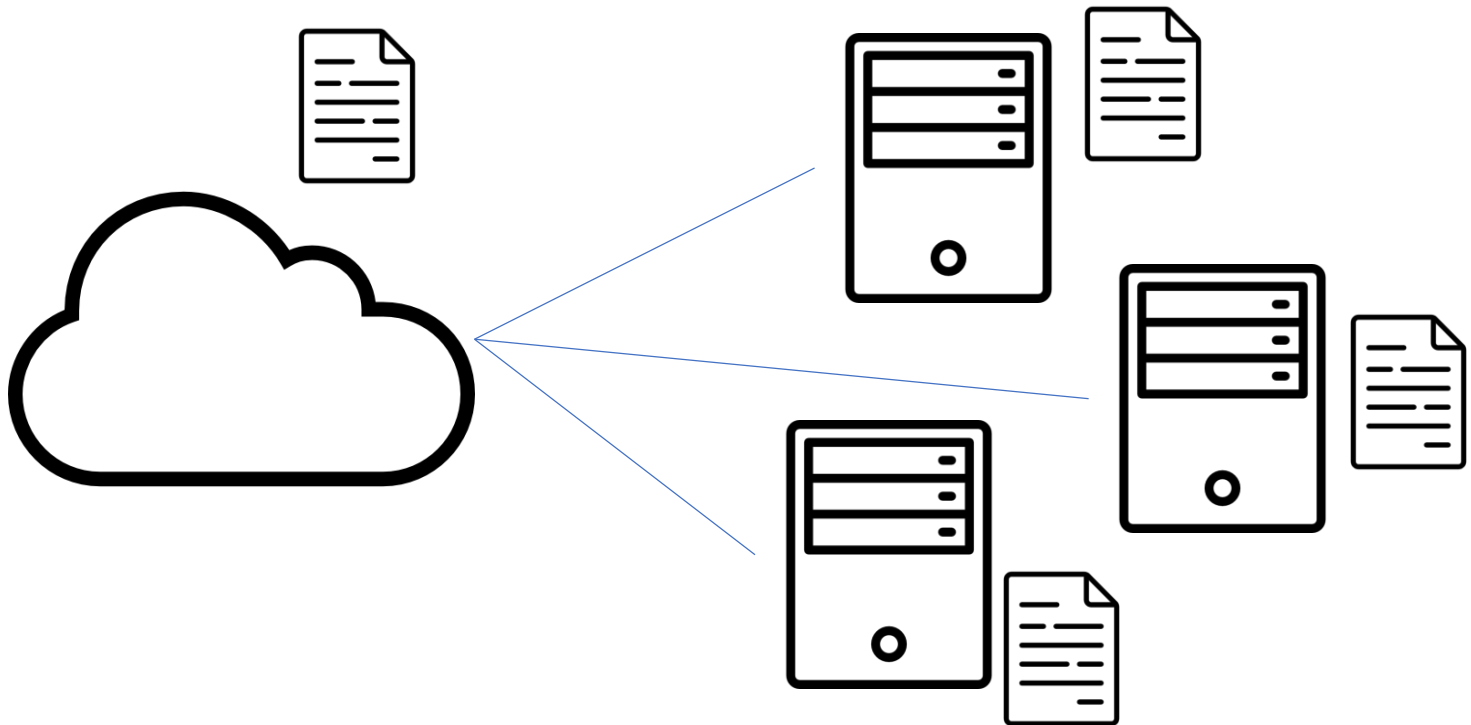
Data Replicas in Distributed Systems

- A **file** could be very **large** such as GBs and TBs



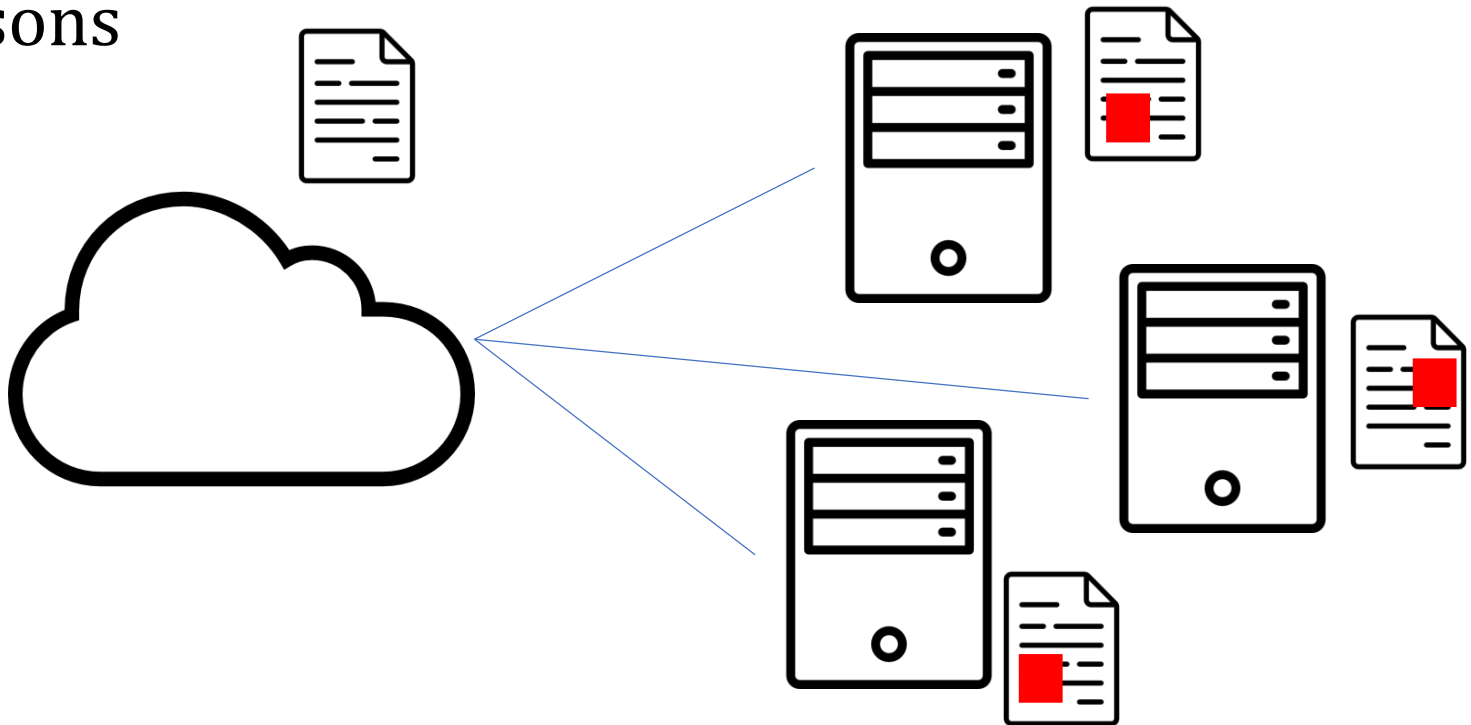
Data Replicas in Distributed Systems

- A file could be very large such as GBs and TBs
- Copy the file to the servers (multiple **replicas**)



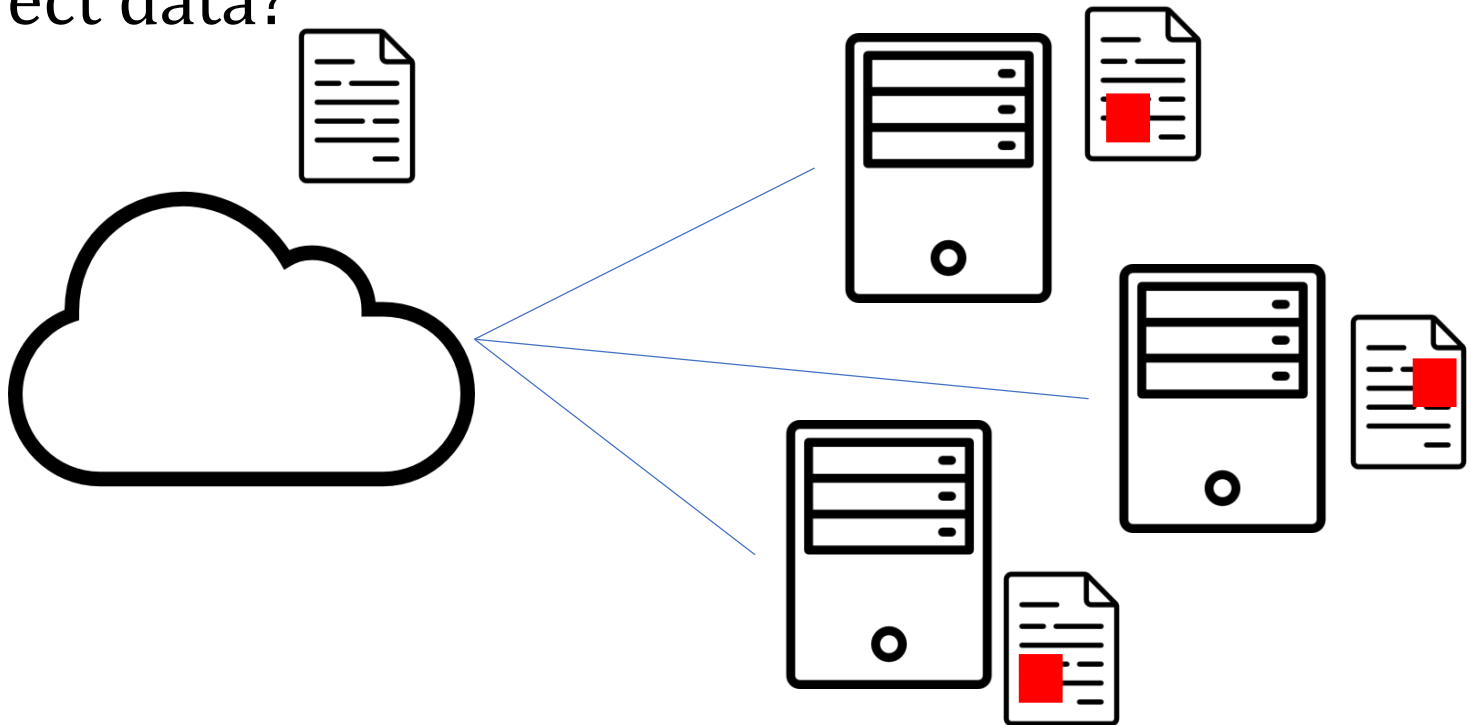
Data Replicas in Distributed Systems

- A file could be very large such as GBs and TBs
- Copy the file to the servers (multiple replicas)
- The file may have **corrupted data** due to some reasons



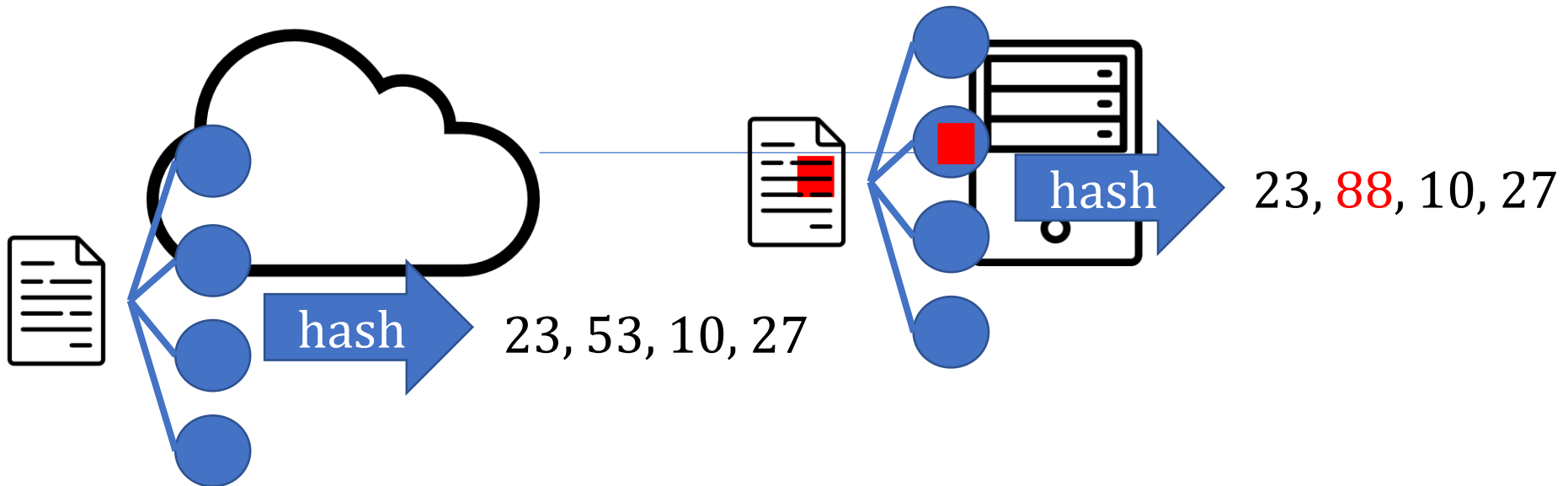
Data Replicas in Distributed Systems

- We need a technique to **detect data corruption** and **repair the failures**
- How to consume as much as **less bandwidth** to correct data?



The First Solution

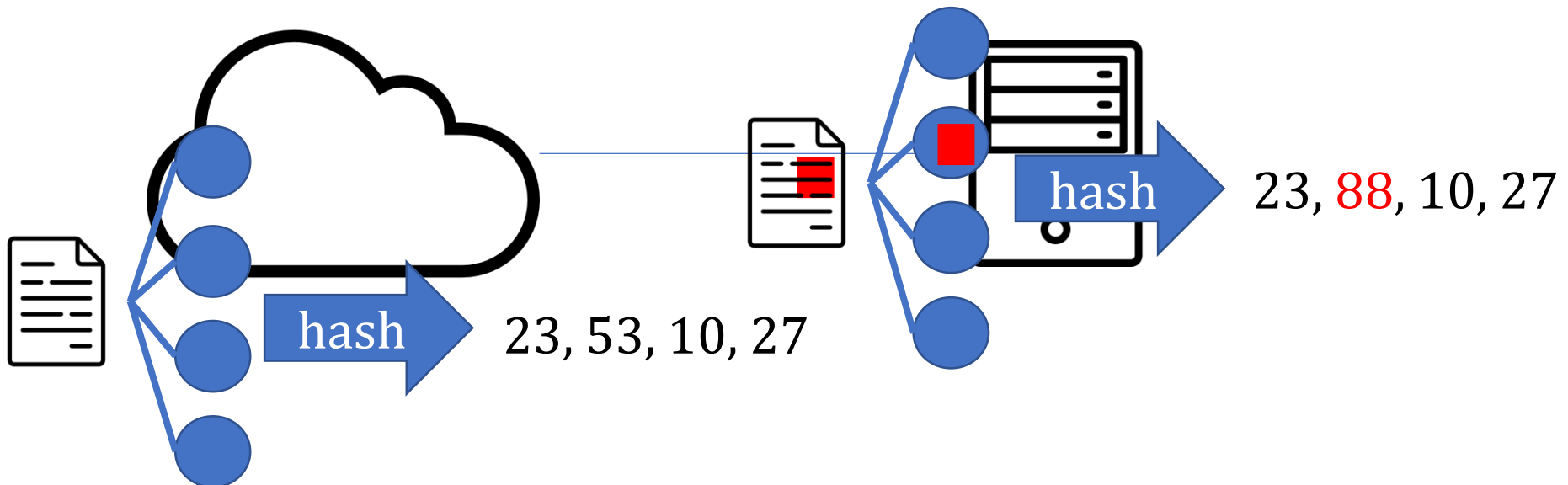
- Divide the file into multiple **smaller chunks**
- Hash each chunk to get the hash value
- Compare the hash values one by one
- **Re-send the chunk** with a different hash value



The First Solution

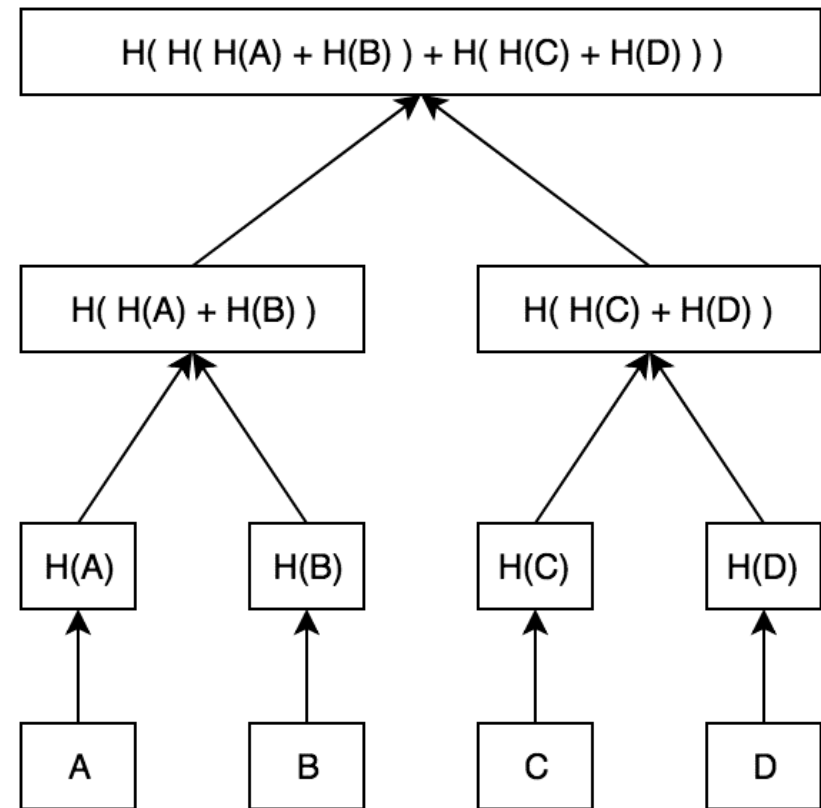
Can we improve this complexity?
Let's say, $O(\log n)$

- Divide the file into multiple small chunks
- Hash each chunk to get the hash value
- Compare the hash values one by one $\rightarrow O(n)$
- Re-send the chunk with a different hash value

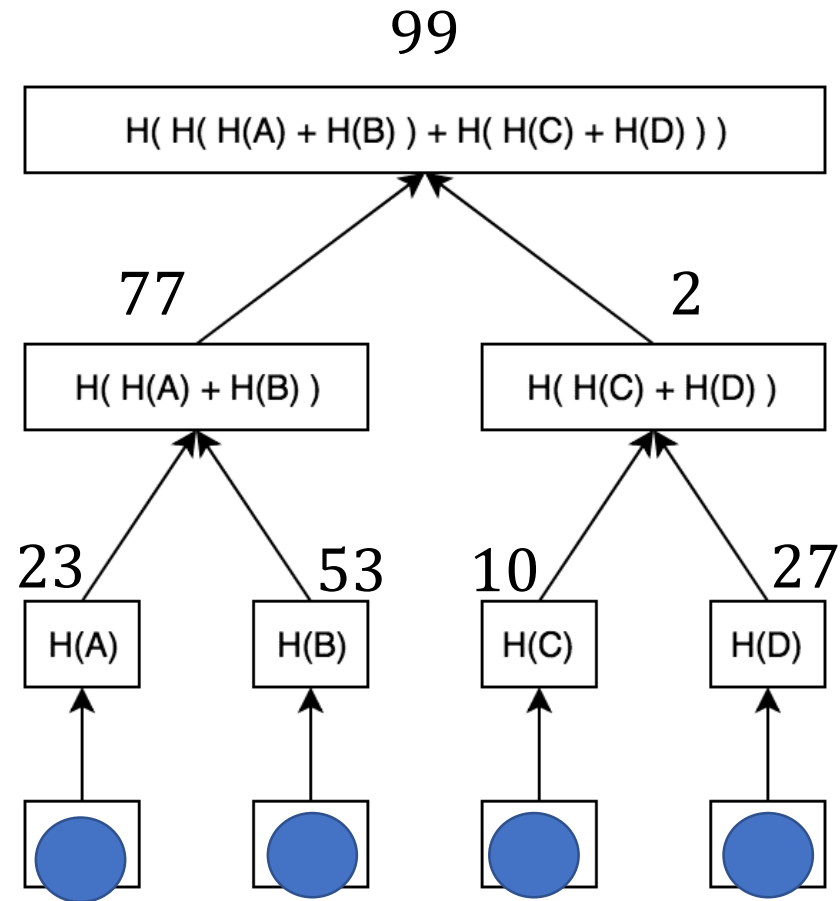
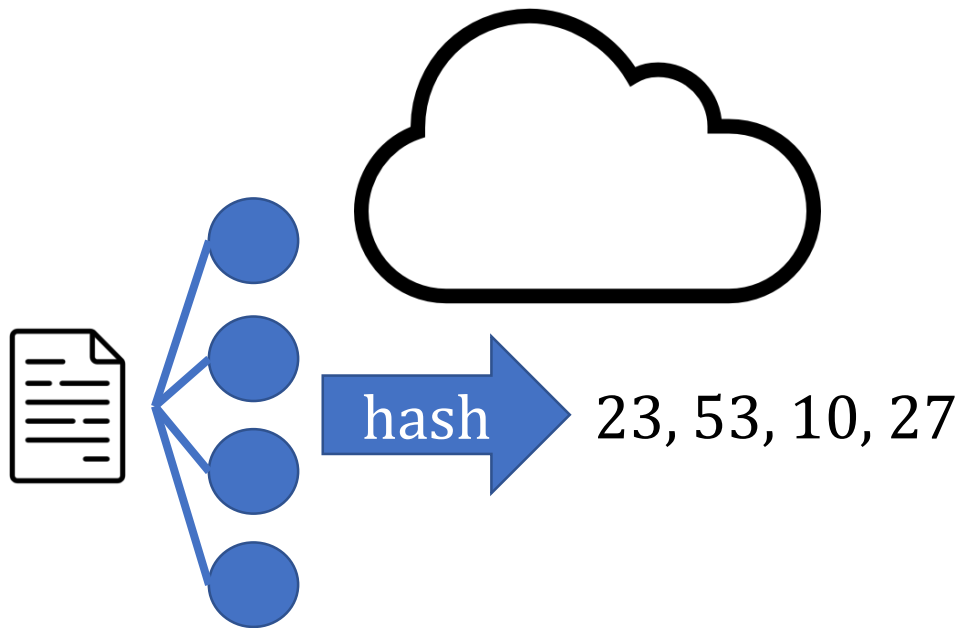


The Second Solution (Using Merkle Tree)

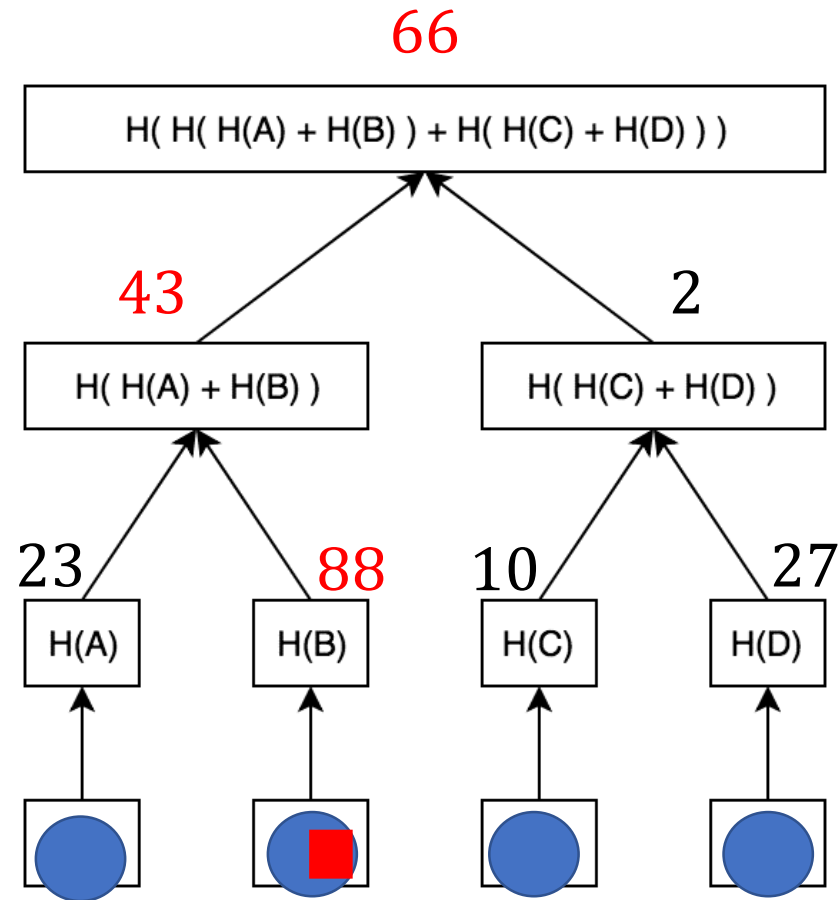
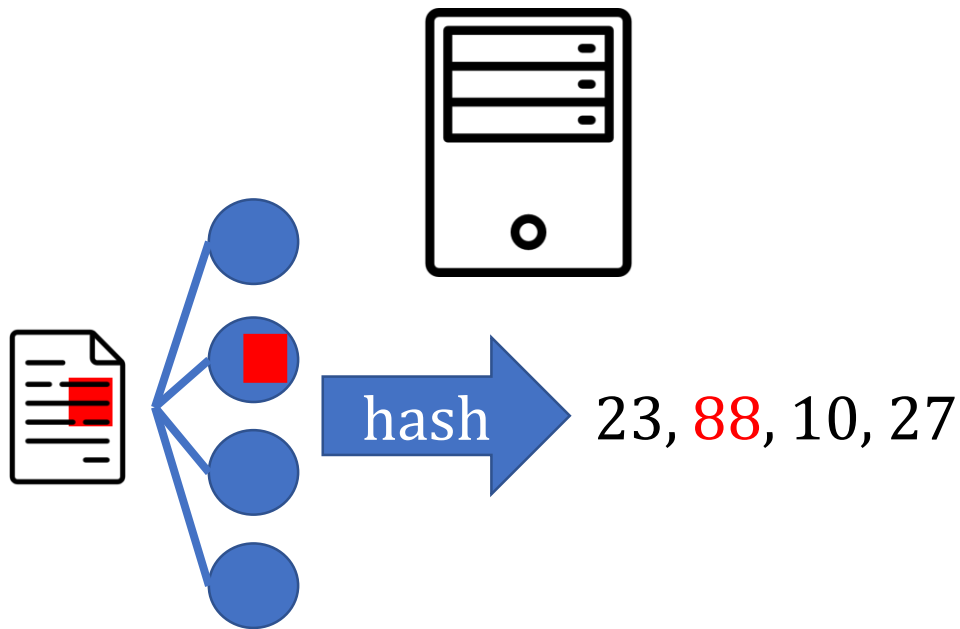
- Divide the file into multiple smaller chunks
- Hash each chunk to get the hash value
- Construct a **Merkle Tree** to abstract the files hierarchically
- Use the Merkle Tree to compare the hash values
- Re-send the chunk with a different hash value



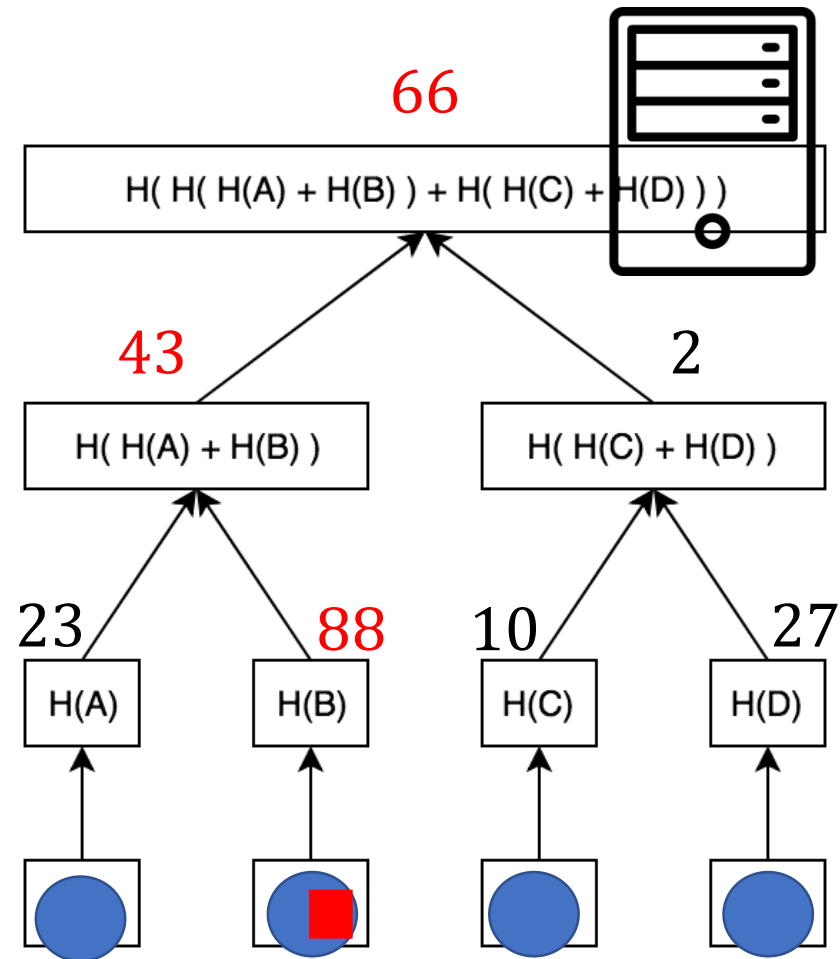
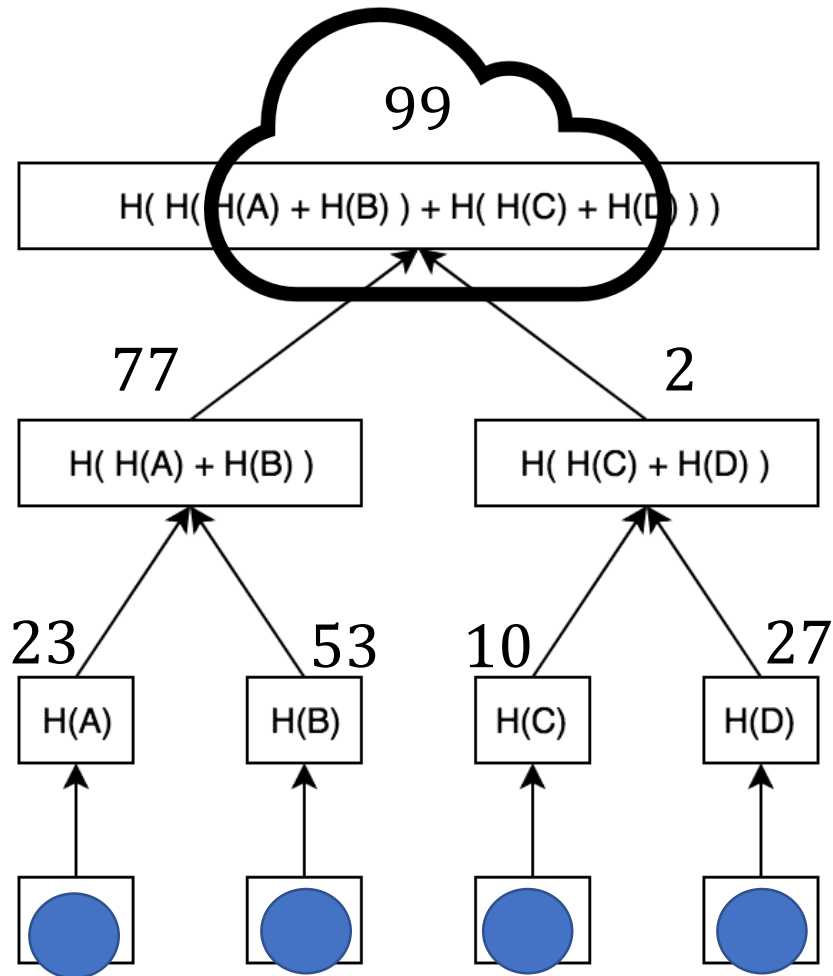
The Second Solution (Using Merkle Tree)



The Second Solution (Using Merkle Tree)



The Second Solution (Using Merkle Tree)



Programming Project #4:

Data Verification with Merkle Tree

- Input:
 - The **data strings** stored in the server
 - The **Merkle tree** built by the master
- Procedure:
 - Compute the **Merkle tree** in the server
- Output:
 - The **incorrect data string** in the server

The Hashing Function

```
unsigned long MurmurOAAT32 (char * key)
{
    unsigned long h = 3323198485ul;
    for (; *key; ++key) {
        h ^= *key;
        h *= 0x5bd1e995;
        h ^= h >> 15;
    }
    return h;
}
```

The Hashing Function

1. Regard the long hash value as 4 char variables
2. Concatenate two long hash values (i.e., 8 char variables) to generate a new long hash value

Input Sample: use scanf

Format:

#Strings StrMaxLen

String1

String2

String3

...

Output Sample: use printf

Format:

#IncorrectStrings

IncorrectString1

IncorrectString2

IncorrectString3

...

Note

- Superb deadline: 1/29 Thu
- Deadline: 1/5 Thu
- Pass the test of our [online judge](#) platform
- Submit your code to [E-course2](#)
- Demonstrate your code in [EA401B](#) or [remotely](#) with TA
- [C Source code \(i.e., only .c\)](#)
- Show a good programming style