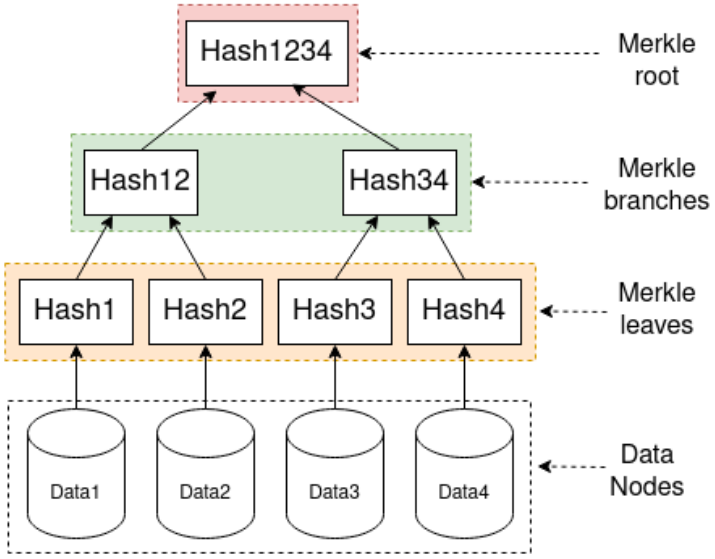
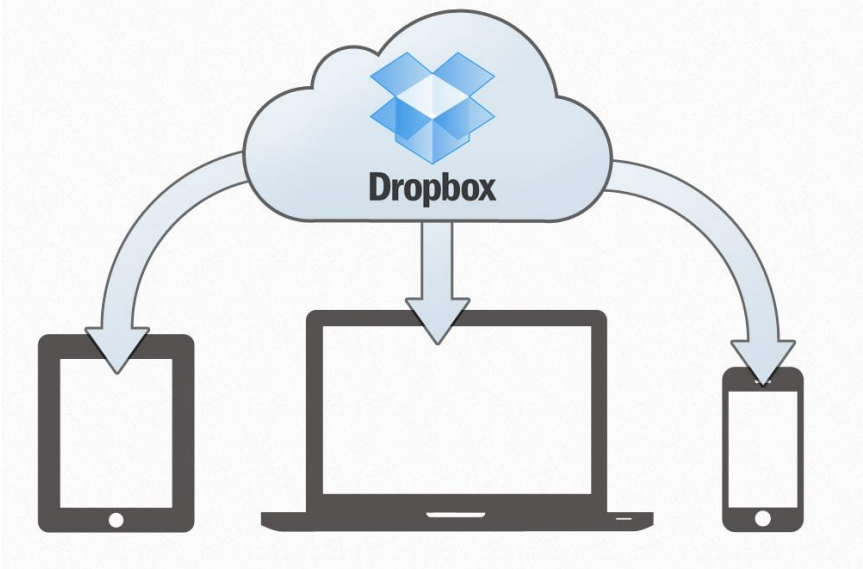
BitTorrent™

cassandra

Data Structures
Programming Project #4



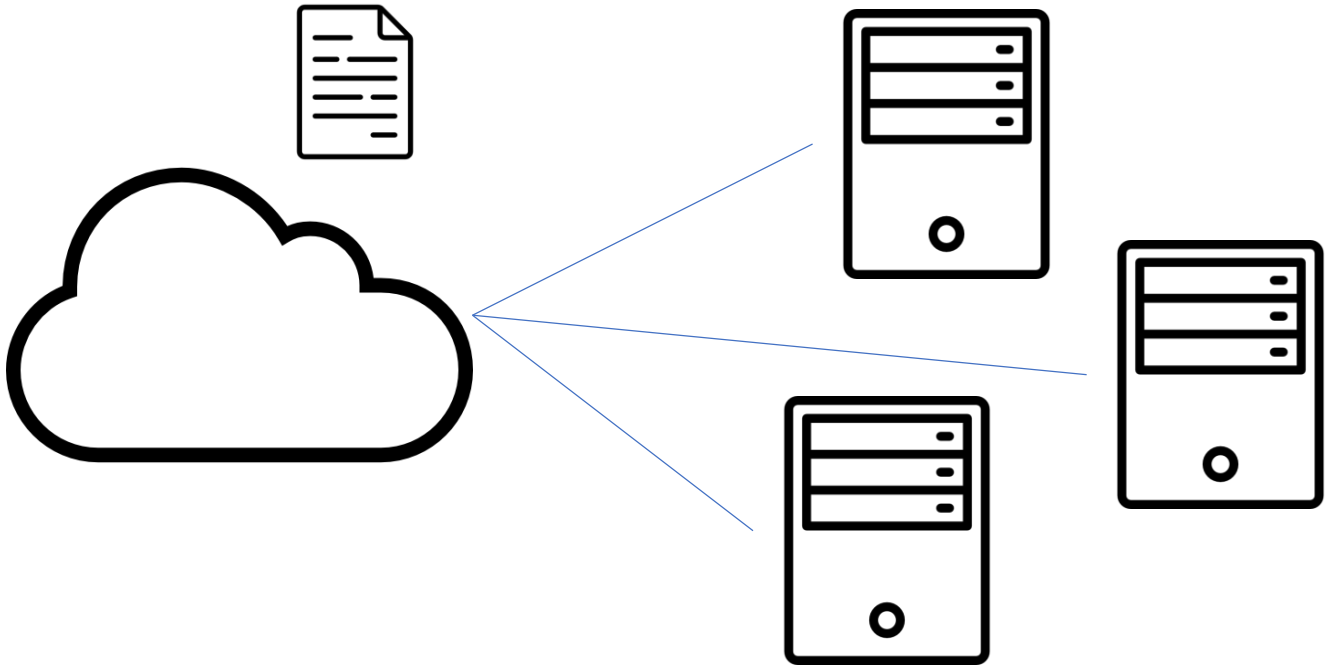


Data Corruption and Verification in Distributed Systems

- 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 Concurrency: Bitcoin
- Distributed Database: Cassandra

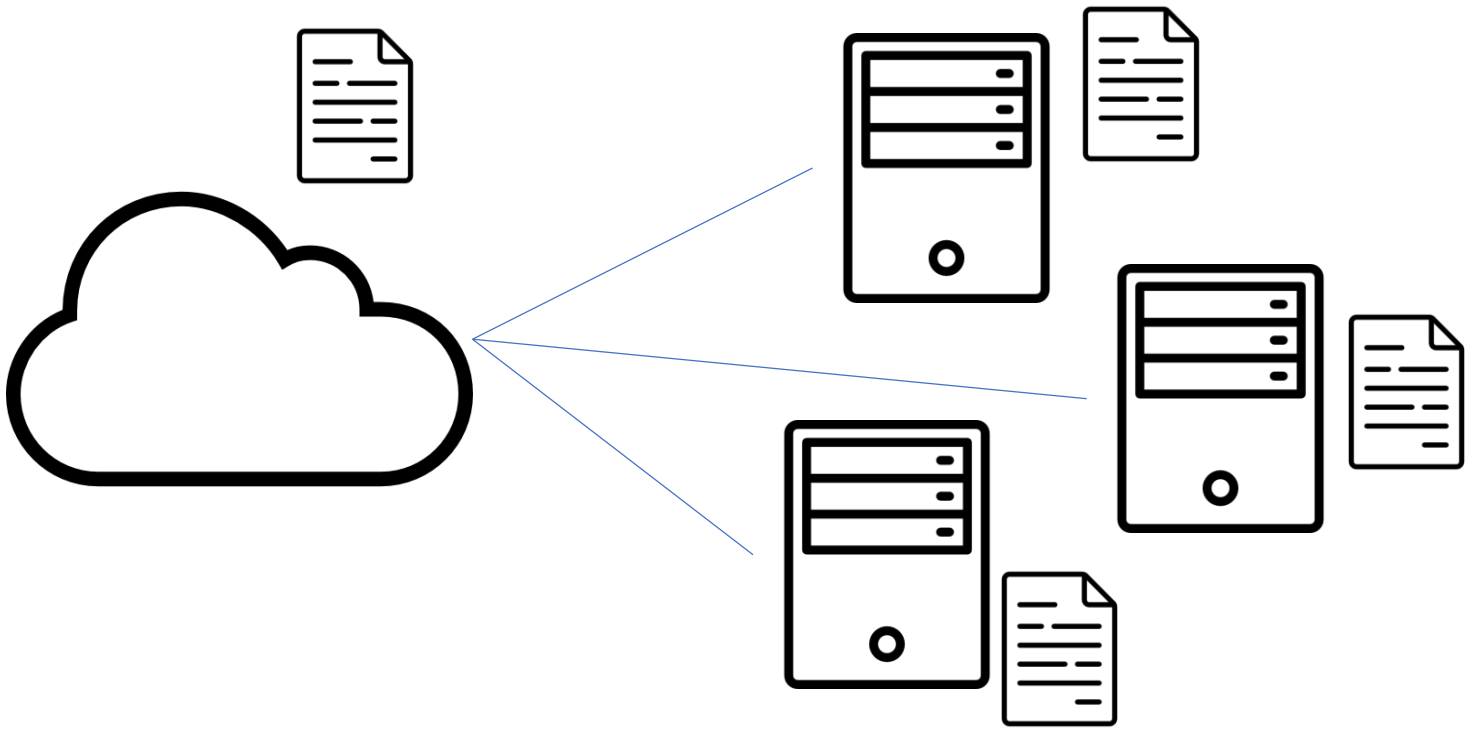
Data Replicas in Distributed Systems

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



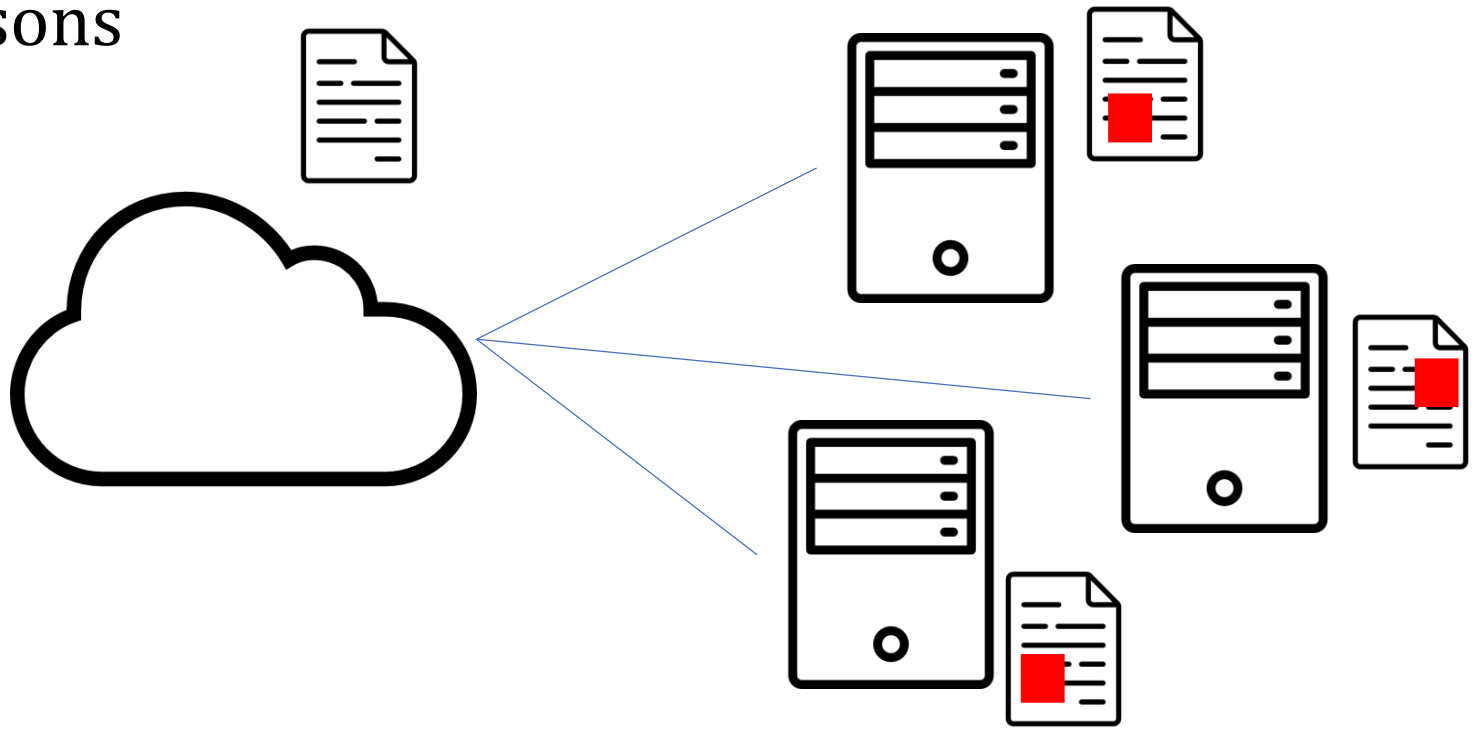
Data Replicas in Distributed Systems

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



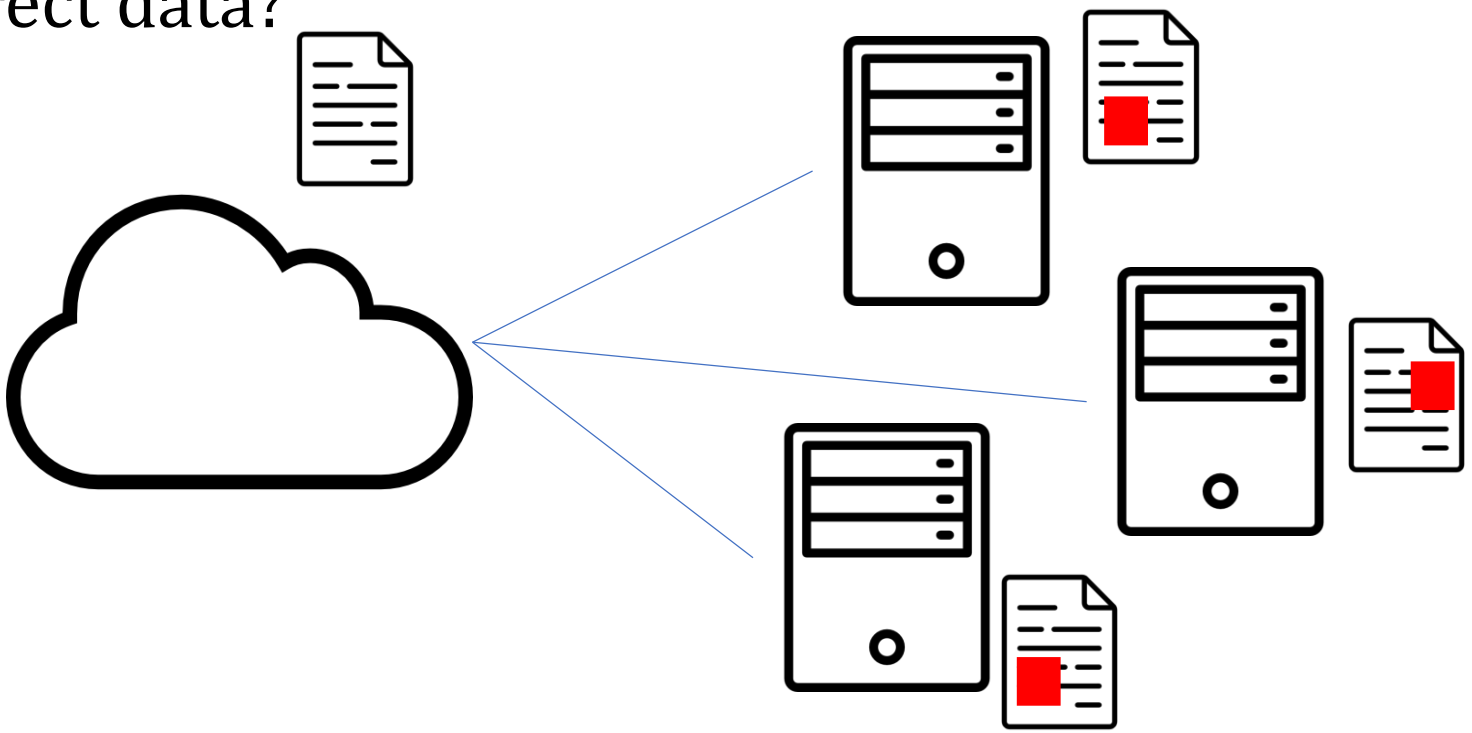
Data Replicas in Distributed Systems

- 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



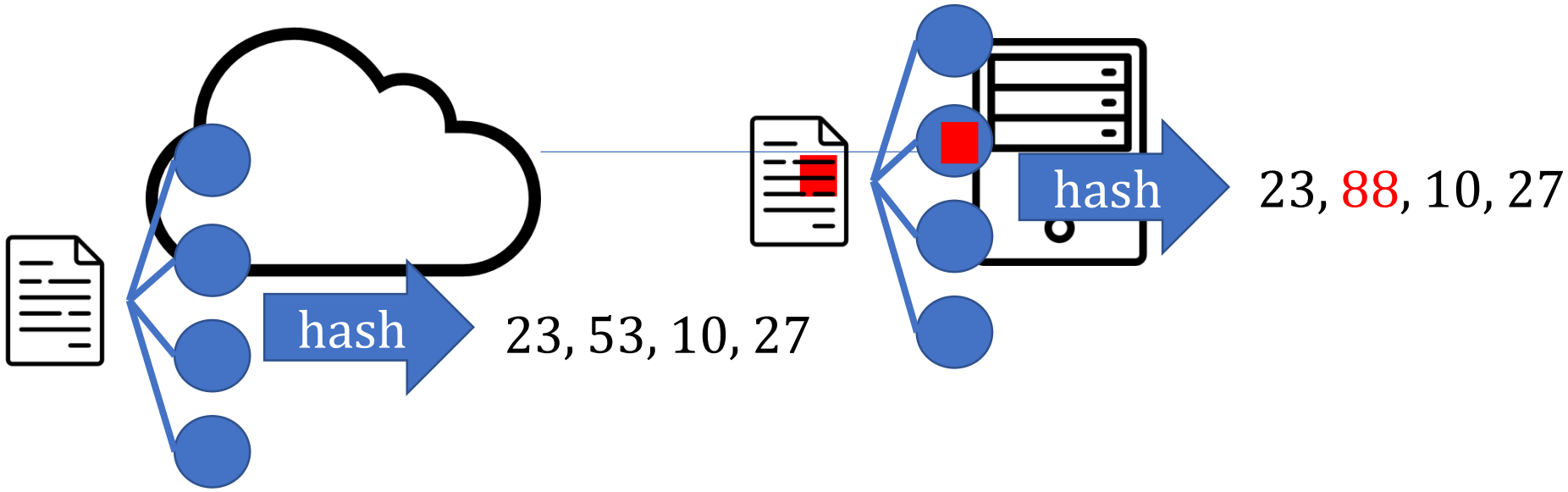
Data Replicas in Distributed Systems

- 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

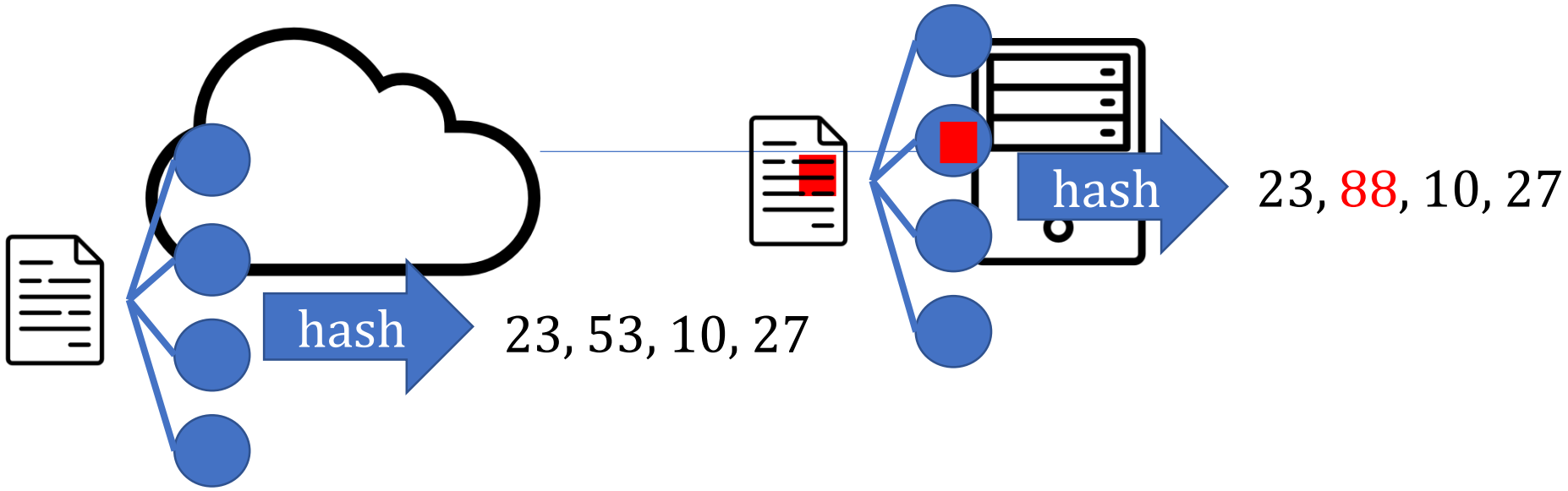
- Divide the file into multiple smaller chunks
- Hash each chunk to get the hash value
- Compare the hash values one by one
- Re-send the chunk with a different hash value



The First Solution

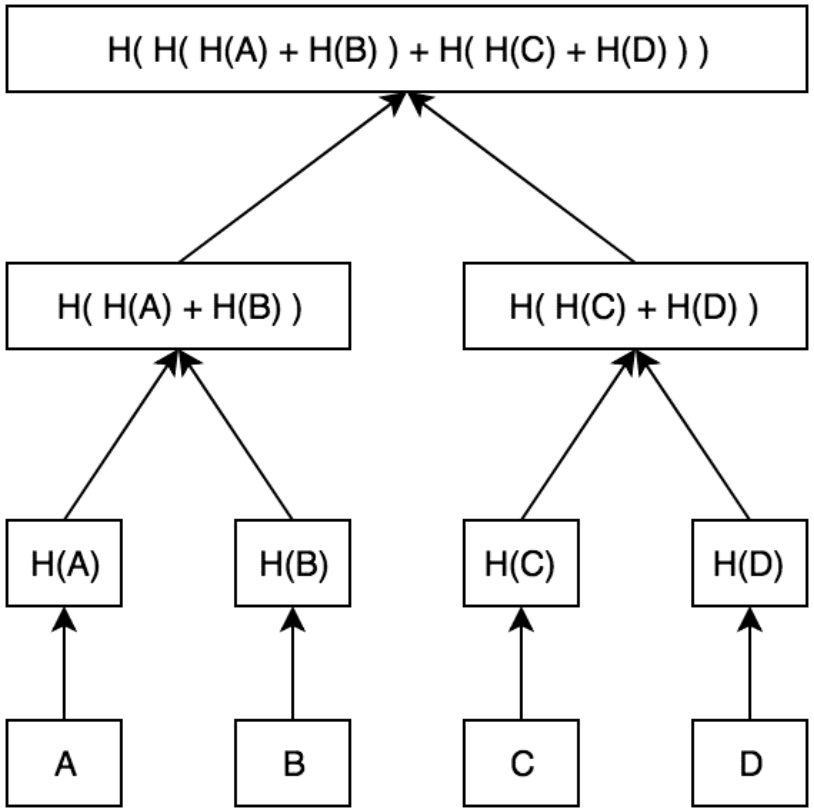
Can we improve this complexity?
Let's say, $O(\log n)$

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

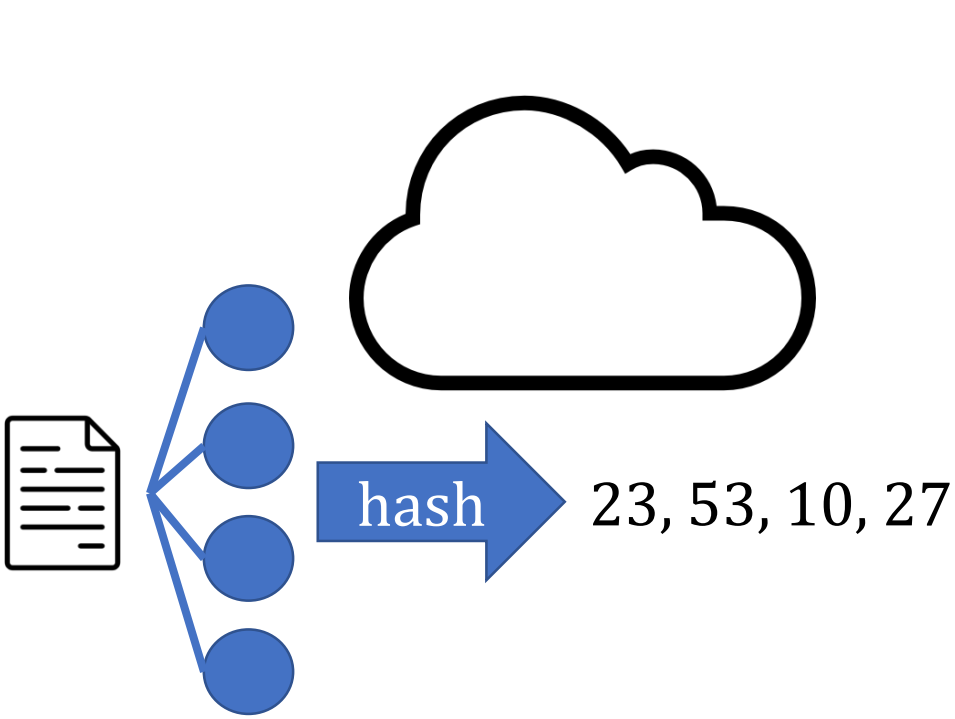


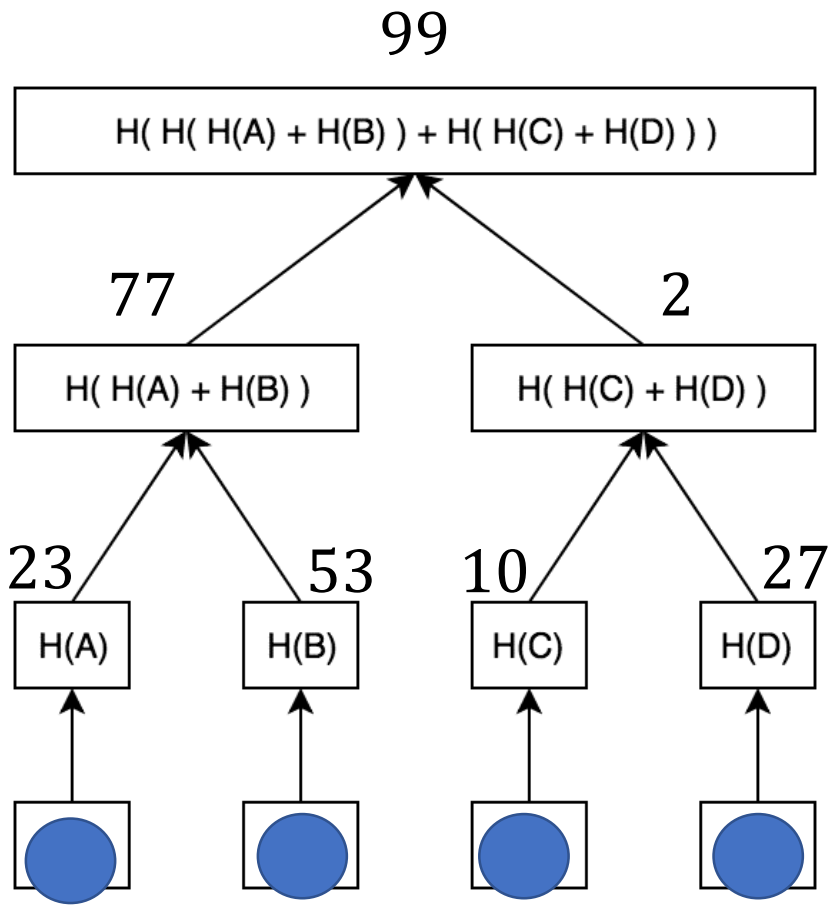
The Second Solution (Using Merkle Tree)

- 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

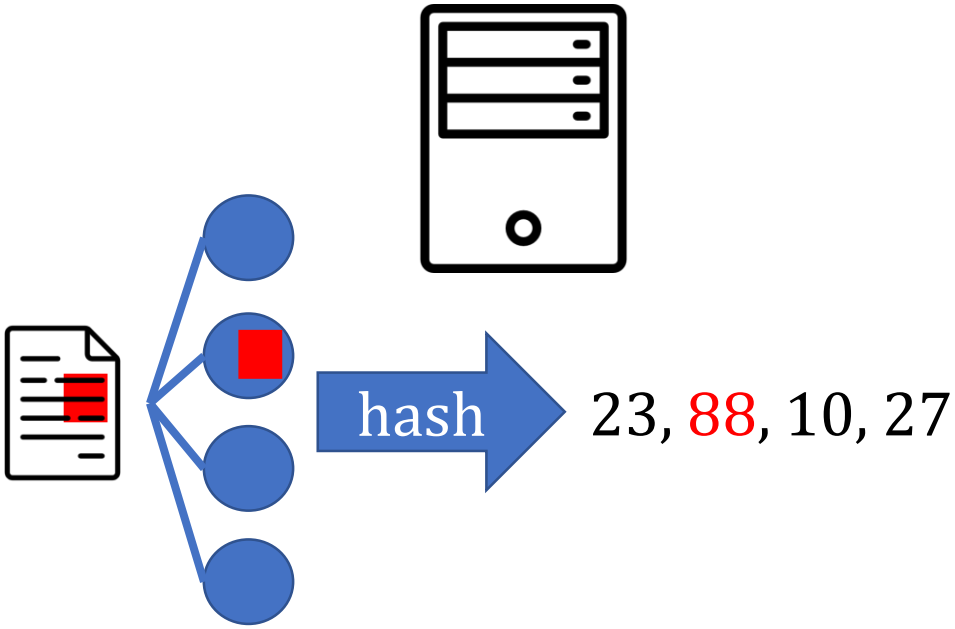


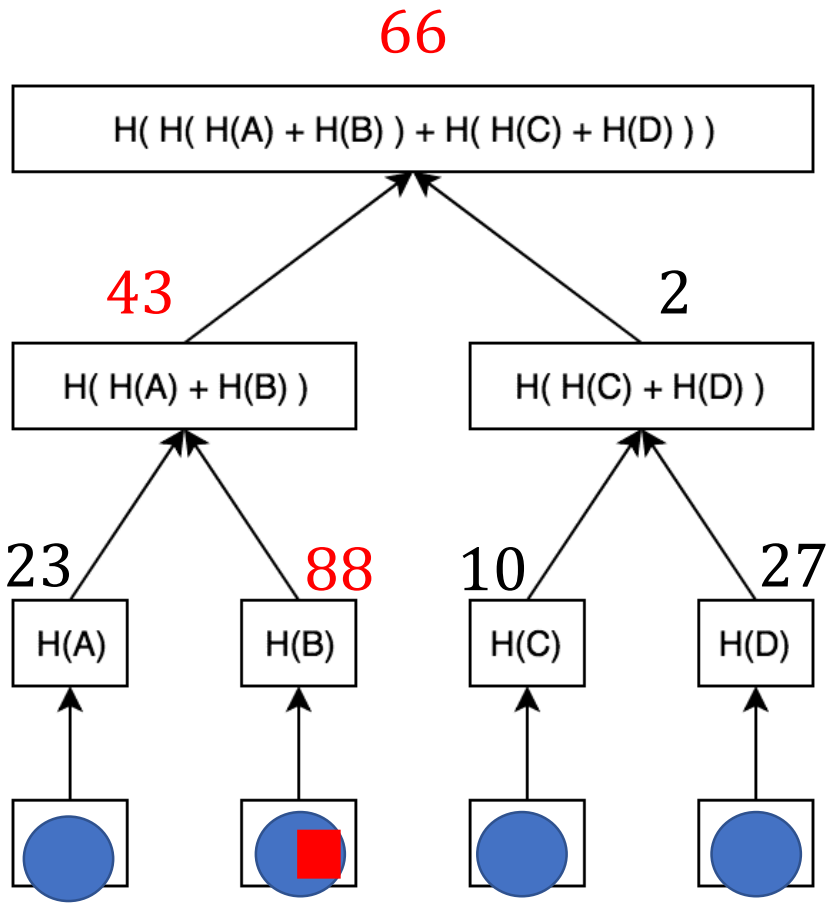
The Second Solution (Using Merkle Tree)



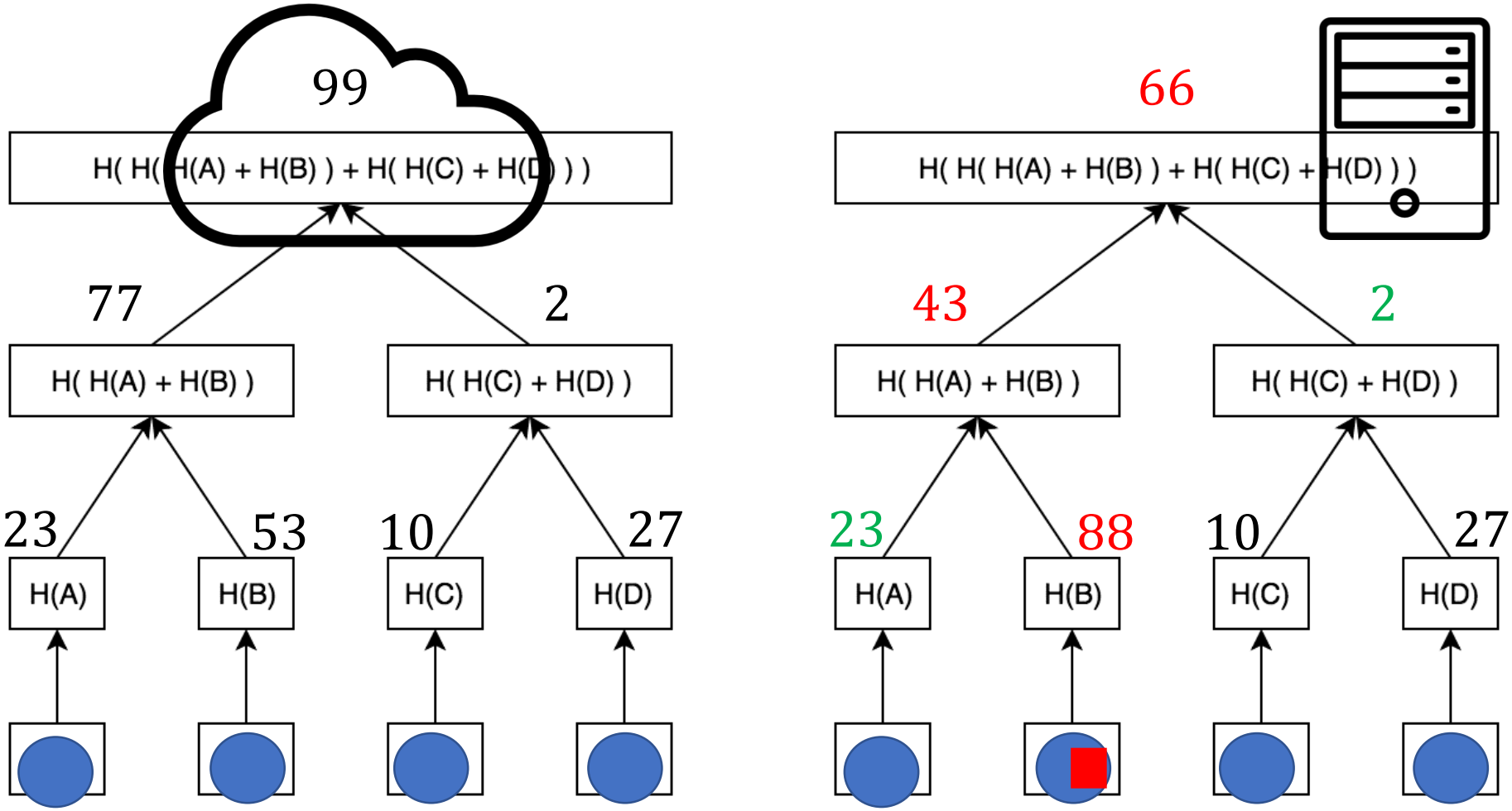


The Second Solution (Using Merkle Tree)





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
- Interactive Action:
  - Query the Merkle tree in the master (with limited times)
- 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

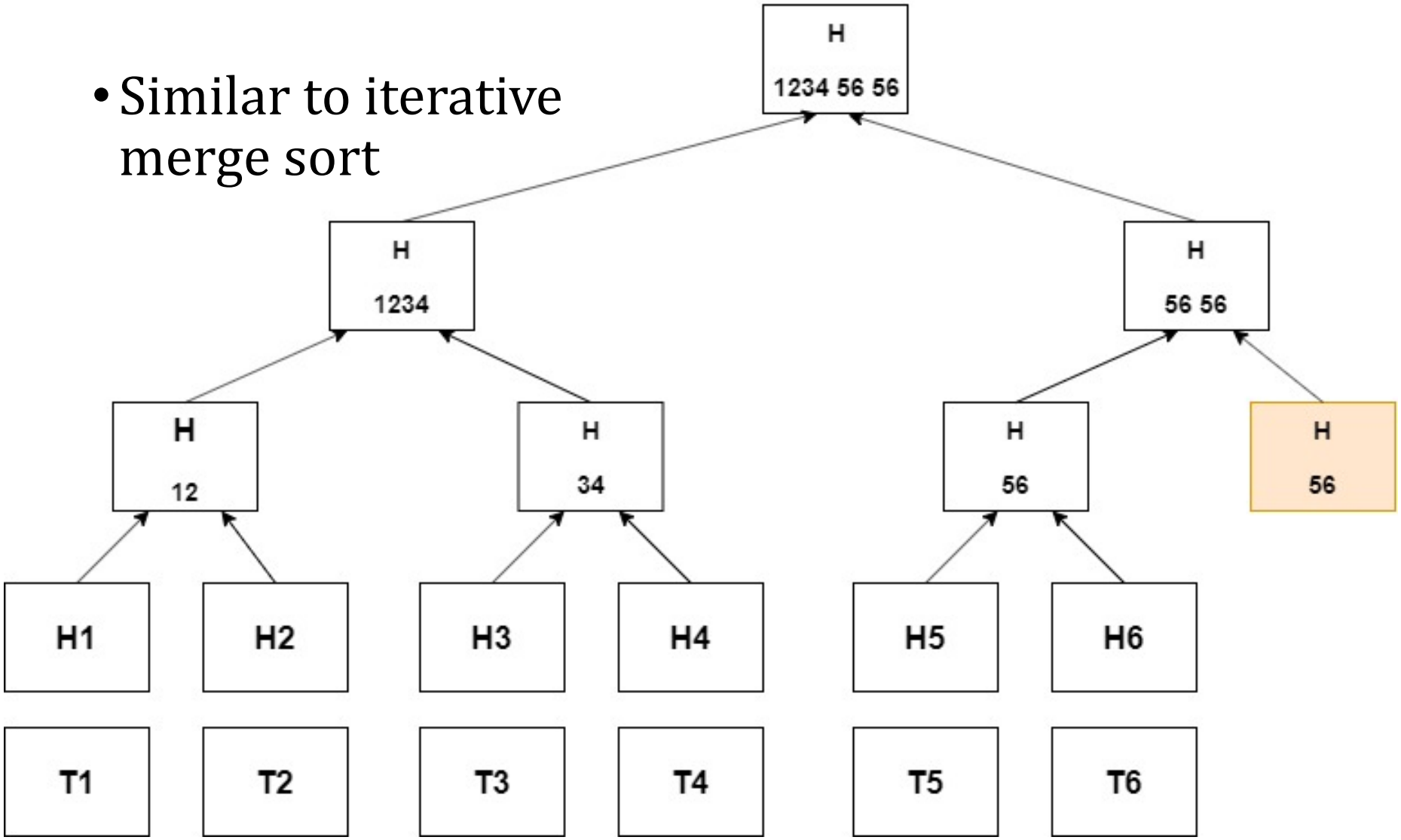
1. (Add) Get an unsigned long value by adding the two unsigned long hash values, e.g., 23 + 53 = 76

2. (I2S) Transform the unsigned long hash value to a char string, e.g., 76 → ”76”

3. (Hash) Hash the char string to acquire the hashed value

If a tree node has no sibling node…

- Similar to iterative merge sort



Input Sample: use scanf

Format:

#Strings StrMaxLen
String1
String2
String3
…

Interactive Input/Output Sample: use scanf/printf

Format:

Query Action

Input: 1 Level1 NodeIndex1

Output: MasterHashValue1

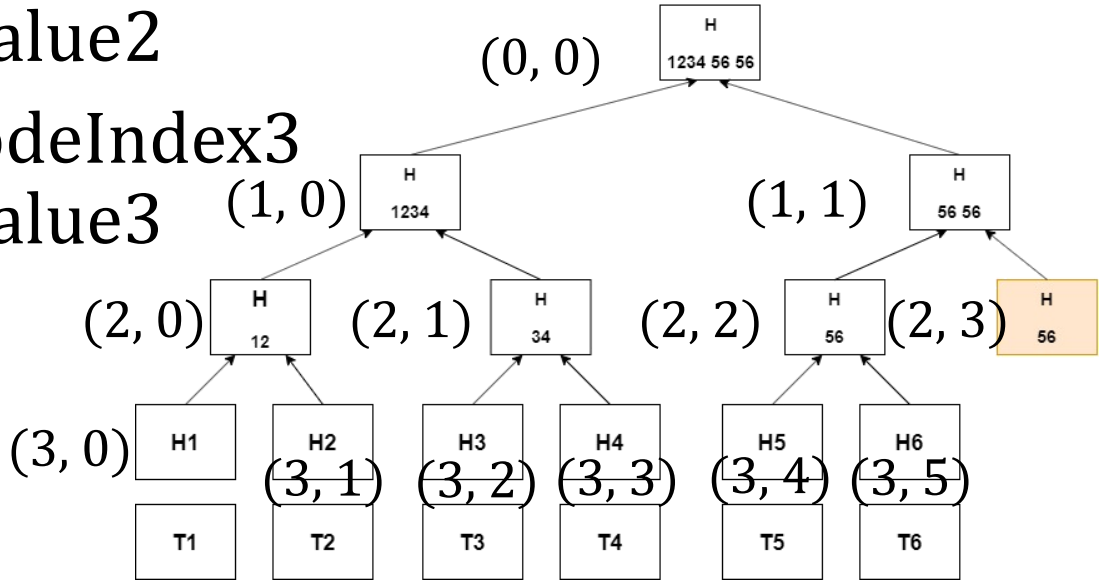
Input: 1 Level2 NodeIndex2

Output: MasterHashValue2

Input: 1 Level3 NodeIndex3

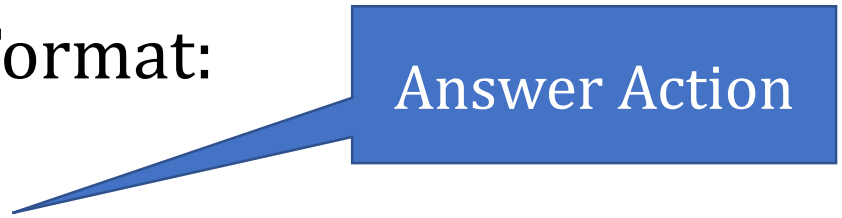
Output: MasterHashValue3

…



Output Sample: use printf

Format:



2 #IncorrectStrings

IncorrectString1

IncorrectString2

IncorrectString3

…

Note

- Superb deadline: 12/29 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