Big Data – Exercises

Fall 2023 – Week 2 – ETH Zurich

Exercise 1: Storage devices (Optional)

In this exercise, we want to understand the differences between SSD, HDD, and SDRAM in terms of **capacity**, **speed** and **price**.

Task 1

Fill in the table below by visiting your local online hardware store and choosing the storage device with largest capacity available but optimizing for read/write speed. For instance, you can visit Digitec.ch to explore the prices on SSDs, HDDs, and SDRAMs. You are free to use any other website for filling the table.

Storage Device	Maximum capacity, GB	Price, CHF/GB	Read speed, GB/s	Write speed, GB/s	Link
HDD					
SSD					

Storage Device	Maximum capacity, GB	Price, CHF/GB	Read speed, GB/s	Write speed, GB/s	Link
DRAM					

Task 2

Answer the following questions:

1. What type of storage devices above is the cheapest one?
2. What type of storage devices above is the fastest in terms of read speed?

Solution:

Looking at Digitec, we fill the table as follows:

Storage Device	Maximum capacity, GB	Price, CHF/GB	Read speed, GB/s	Write speed, GB/s	Link
HDD	18 TB;	0.032 CHF/GB	0.25 GB/s	0.25 GB/s	link1 link2
SSD	4 TB;	0.25 CHF/GB	7.3 GB/s	6.9 GB/s	link
DRAM	128 GB per module;	12.5 CHF/GB	60 GB/s	48 GB/s	link1 link2

1. HDDs are the cheapest storage device among mentioned devices

2. DRAMs are the fastest storage device among mentioned devices

SDRAM (Synchronous Dynamic Random-Access Memory) VS. DRAM (Dynamic Random-Access Memory):

1. **Synchronous Operation**:
    - SDRAM is synchronized with the computer's bus speed, meaning it operates in sync with the computer's clock speed. This synchronization allows for faster data access and transfer.
    - DRAM, on the other hand, is asynchronous and does not synchronize with the computer's clock. This results in a less efficient use of the available bandwidth.
2. **Access Speed and Efficiency**:
    - SDRAM is faster and more efficient in terms of accessing and transferring data compared to traditional DRAM. The synchronous nature of SDRAM allows for higher data transfer rates.
    - DRAM has slower access times and data transfer rates compared to SDRAM due to its asynchronous operation.
3. **Latency**:
    - SDRAM has lower latency compared to DRAM. Latency refers to the time it takes for the memory to respond to a read request. SDRAM's synchronous operation helps in reducing latency.

- DRAM has higher latency due to its asynchronous nature, leading to longer delays in accessing data.

4. **Burst Mode**:

   - SDRAM supports burst mode, where it can transfer multiple data words in a sequence with a single access command. This is more efficient for reading sequential data.
   - DRAM does not support burst mode, so each data access requires a separate access command.

5. **Power Consumption**:

   - SDRAM typically consumes less power compared to DRAM, making it more energy-efficient.

6. **Price and Availability**:

   - Historically, SDRAM has been more expensive than DRAM due to its improved performance and features. However, as technology advances and newer generations of memory are introduced, price differences may vary.

In summary, SDRAM offers better performance, lower latency, and higher efficiency compared to DRAM due to its synchronous operation and support for burst mode. However, DRAM is simpler in design and may still be used in applications where performance requirements are not as demanding or where cost considerations are more significant.

Additional question

How much would storing 2GB of data cost in 80's? Translate to dollars in today's value.

$7M ⇒ Storage was super expensive relational DBs

Now, storage is cheap, compute is expensive

Avoid expensive joins

NoSQL

Exercise 2. KeyValue Vector Clocks

As pointed out in the lecture, the concepts are clearly explained in the Dynamo paper by the DeCandia, G., et. al. (2007). "Dynamo: Amazon's Highly Available Key-value Store." In SOSP '07 (Vol. 41, p. 205). DOI

Dynamo DB ATC'22

→ Paxos

Lamport Clock

LC

K8S ⇒ etcd

Hadoop → HeadNode

Time ↓

A

B

C

LC=0

LC=0

LC=0

LC

x

y

z

x'

z'

y'

h

LC(x)=1

LC(y)=2

LC(z)=3

LC(x')=2

LC(z')=3

LC(y')=4

LC(h)=1

z_1 z_2 z_3

① Consistent with causality

$x \to y \Rightarrow LC(x) < LC(y)$

② $LC(x) < LC(y) \not\Rightarrow x \to y$

Characterizes causality

Vector Clock: ① ②

V([2 2 0]) context → [2 2 1] ← [0 0 1]

$VC(x) > VC(y) \iff x > y$

Task 1

Multiple distributed hash tables use vector clocks for capturing causality among different versions of the same object. In Amazon's Dynamo, a vector clock is associated with every version of every object.

Let VC be an N-element array which contains non-negative integers, initialized to 0, representing N logical clocks of the N processes (nodes) of the system. VC gets its j element incremented by one everytime node j performs a write operation on it.
Moreover, $VC(x)$ denotes the vector clock of a write event, and $VC(x)_z$ denotes the element of that clock for the node z.

Try to **formally define** the partial ordering that we get from using vector clocks.

Solution

$$VC(x) \lesssim VC(y) \iff \forall z : VC_z(x) \leq VC_z(y)$$

Task 2

Vector clock antisymmetry property is defined as follows:

$$\text{If } VC(x) < VC(y), \text{ then } \neg [VC(y) < VC(x)]$$

Prove this property.

Solution : PBC

Assume that ① $VC(x) < VC(y)$ AND ② $VC(y) < VC(x)$

* Given ①, we have $\forall z : VC(x)_z \leq VC(y)_z$ ③

AND $\exists k : VC(x)_k < VC(y)_k$ ⑤

* Given ②, ... $\forall z : VC(y)_z \leq VC(x)_z$ ④

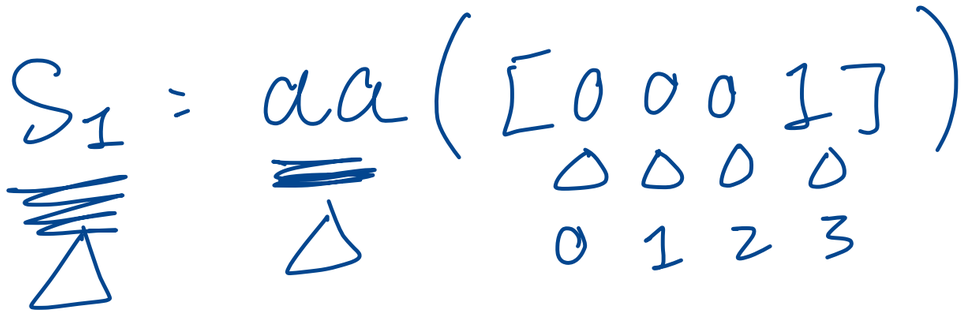
AND $\exists k : VC(y)_k < VC(x)_k$ ⑥

③ and ④ $\Rightarrow \forall z : VC(x)_z = VC(y)_z$ ⑦

⑦ ↯ (⑤, ⑥) : we've arrived at a contradiction.

$\Rightarrow$ The assumption that ① and ② hold is invalid.

$\Rightarrow VC(x) < VC(y) \rightarrow VC(y) \not< VC(x)$: Antisymmetry □

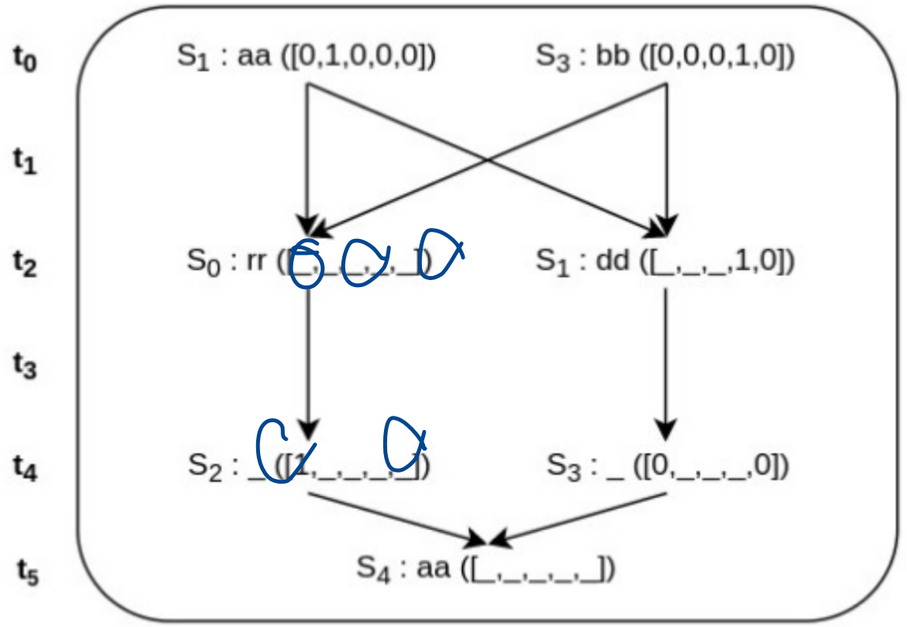


Task 3

Consider j servers in a cluster where S_j denotes the jth node.
In this exercise, we adopt a slightly modified notation from the Dynamo paper:

- The Dynamo paper indicates the *writing server* on the edge, we however write it before the colon.
- For brevity, we index server by position and omit server name in the vector clock.

For example **aa ([S_0,0],[S_1,4])** with S_1 as writing server become **S_1 : aa ([0,4])**
So, given the following version evolution DAG for a particular object, complete the vector clocks computed at the corresponding version.

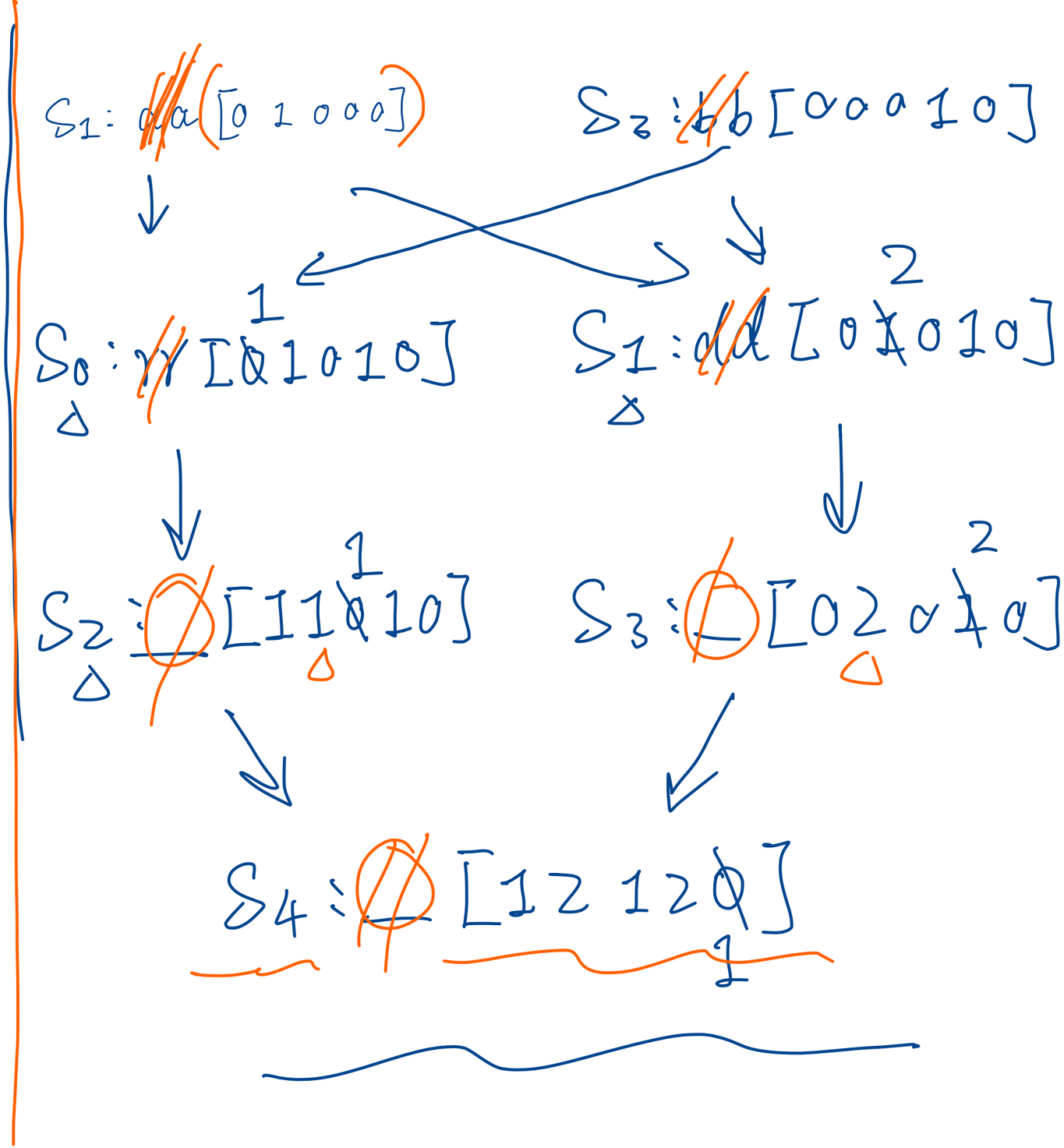


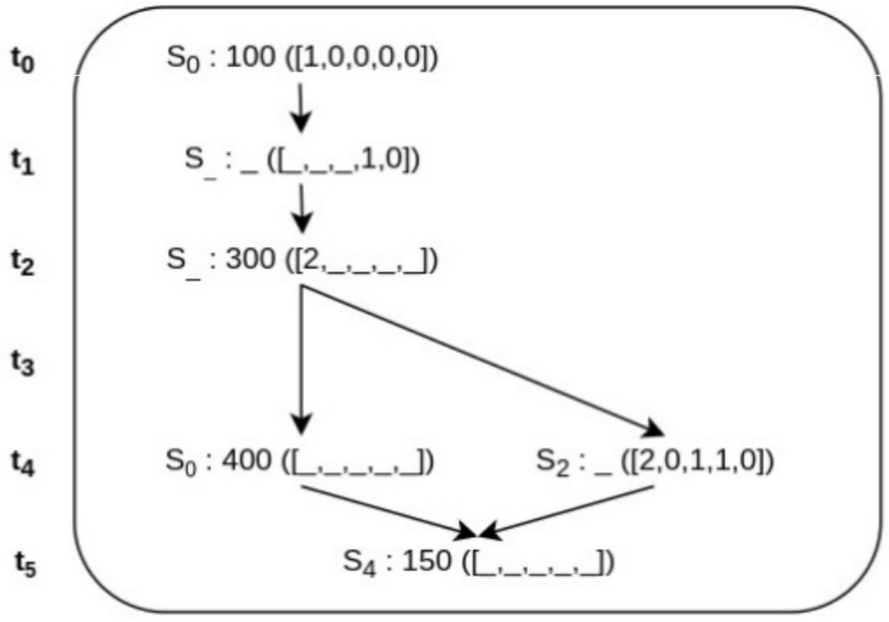
Obj. 2

2-step Method :

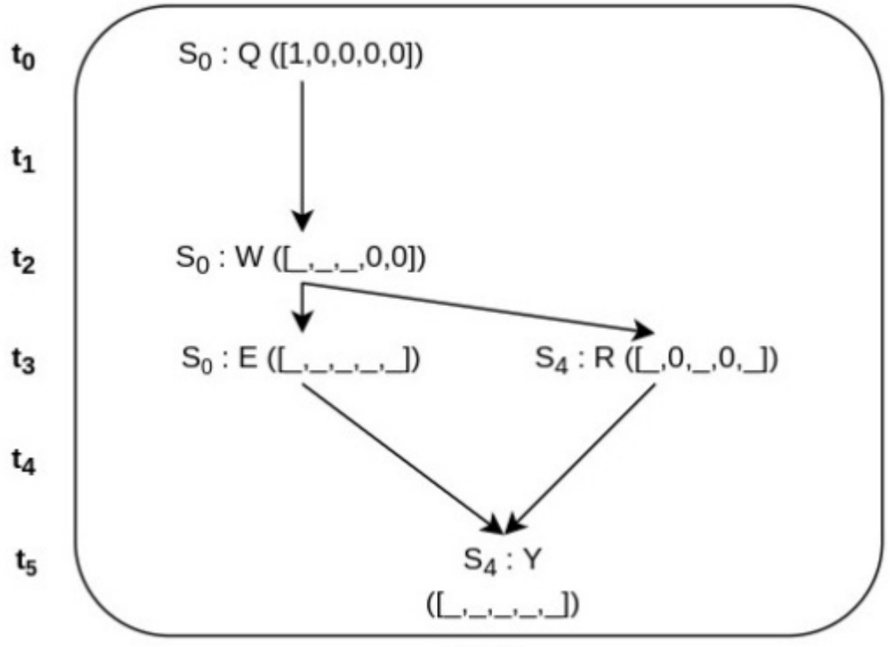
① Element-wise max

② Increament





Obj. 3



Obj. 1

Task 4

When a get request comes in to Amazon Dynamo with some key, then:

- The coordinator node (selected from the preference list as the top node for this key) is taking care of this request
- The coordinator node requests from other nodes (itself + the next N-1 healthy ones on the preference list), and receives, a set of versions for the value associated with the key, that are modelled as **value (vector clock)** pairs such as a ([1, 3, 2])

Task 4.1

Given the following list of versions, draw the version DAG that the coordinator node will build for returning available versions.

1 ([0,0,1])
1 ([0,1,1])
2 ([1,1,1])
3 ([0,2,1])
10 ([1,3,1])

Task 4.2

Given the following list of versions, draw the version DAG that the coordinator node will build for returning the correct version.

a ([1,0,0])
b ([0,1,0])
c ([2,1,0])
d ([2,1,1])
e ([3,1,1])
f ([2,2,1])
g ([3,1,2])
h ([3,2,3])
i ([4,2,2])
j ([5,2,2])
k ([4,3,3])
l ([5,2,3])
m ([5,4,3])
n ([6,3,3])
o ([6,4,4])

S_0 [1 0 0] S_1 [0 1 0]

S_0 [1 1 0]

S_2 [2 1 1]

S_0 [3 1 1] S_1 [2 2 1]

S_2 [3 1 2]

S_2 [3 2 3]

S_0 [3 2 2]

S_0 [5 2 2]

S_1 [4 3 3]

433

S_2 [5 2 3]

S_1 [5 4 3]

S_0 [6 3 3]

S_2 [6 4 4]

Task 4.3

Given the following list of versions, draw the version DAG that the coordinator node will build for returning the correct version.

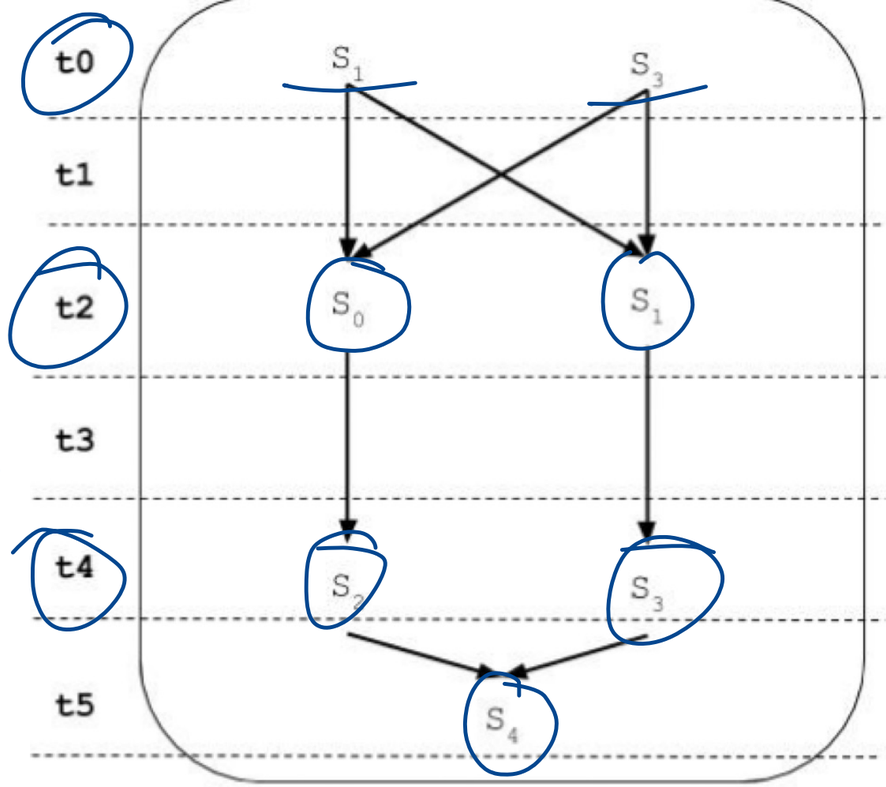
a ([1,0,0,0])
b ([0,0,0,1])
aa ([0,0,1,0])
bb ([0,1,0,0])
c ([1,2,0,1])
cc ([0,1,1,2])
d ([1,3,0,1])
f ([1,2,1,3])
e ([2,1,1,2])
g ([2,2,2,3])

Task 5

We assume only put/write operation increases the clock

key = 1

Same DAG and answer of Task 3 obj 2



	S0	S1	S2	S3	S4
t0	[0 1 0 0 0]	Get(1)→ [*], C_1 ([*]) Put(1, 1, "a")	Get(1)→ [*], C_2 ([*]) Put(1, 2, "bb")	[0 0 0 1 0]	
t2	Get(1)→ a, bb, C_4 ([0 1 0 1 0]) Put (1, 1, "rr")	Get(1)→ [a, bb] C_5 ([a, bb]) Put (1, 2, "dd")	[0 2 0 1 0]		
t4	[1 1 0 1 0]		Get(1)→ [rr], C_9 ([rr]) Put(1, 1, "ccc")	Get(1)→ [dd], C_{10} ([dd]) Put(1, 2, "dd")	[1 2 1 2 0]
t5			[1 1 1 1 0] ⇔ [0 2 0 2 0]		Get(1)→ [ccc, dd], C_{11} ([ccc, dd]) Put(1, 1, "fff") [1 2 1 2 1]

Exercise 3. Merkle Trees

A hash tree or Merkle tree is a binary tree in which every leaf node gets as its label a data block and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes.

Some KeyValue stores use Merkle trees for efficiently detecting inconsistencies in data between replicas.

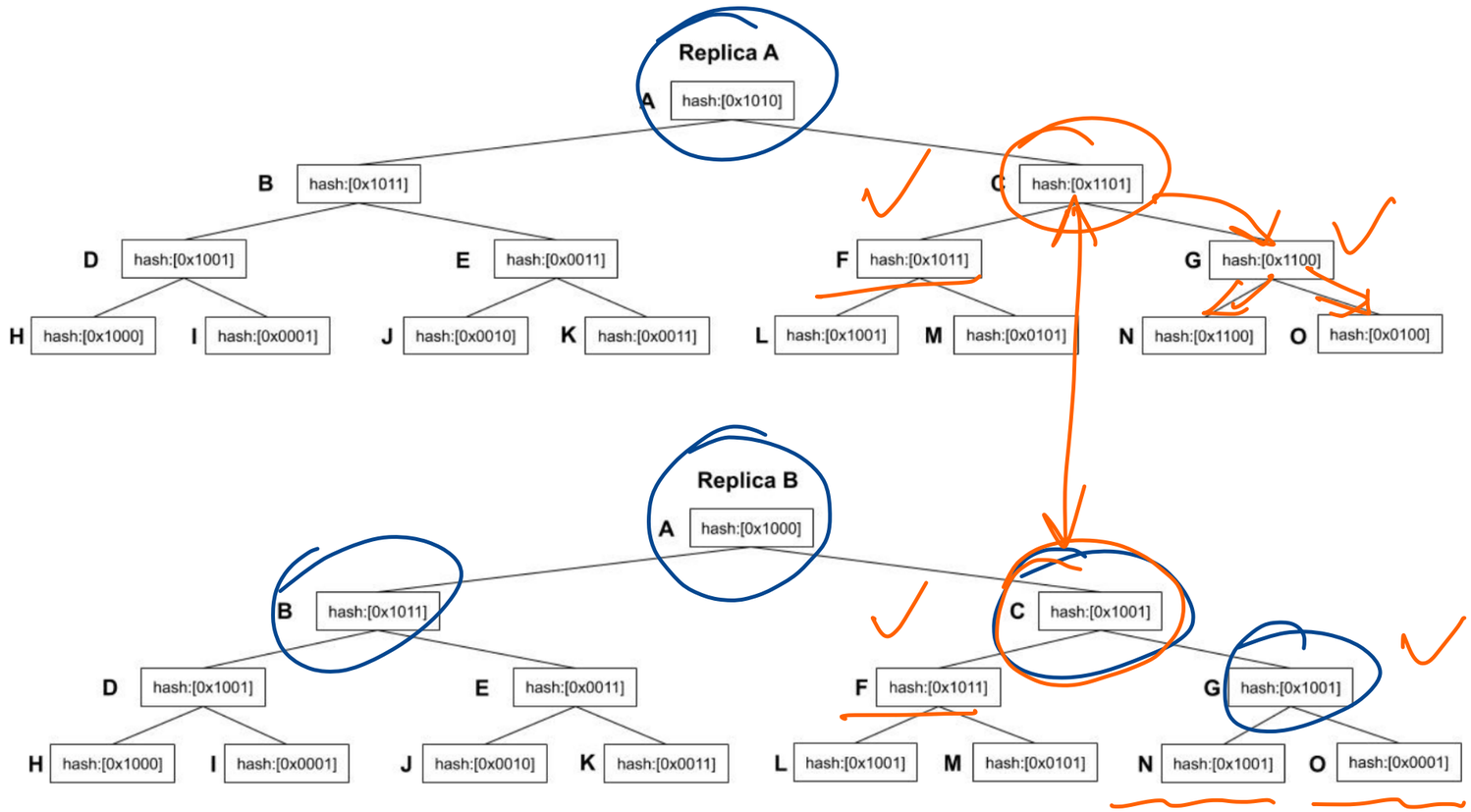
This works by exchaging first the root hash, comparing it with their own. If the hashes match, the replicas are synchronised. If they do not match, then the children of the node (in the Merkle tree) will be retrieved, and their hashes will be compared. This process continues until the inconsistent leave(s) are identified.

(**Used a lot in bitcoin**)

Task 1

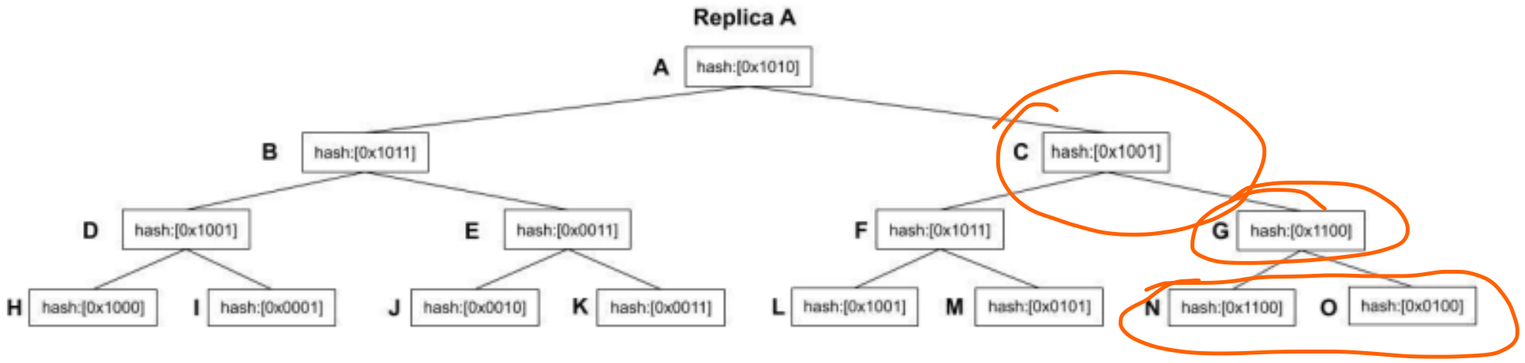
The two pictures below depict two Merkle trees each one belonging to two different replicas. Both should represent the same object.

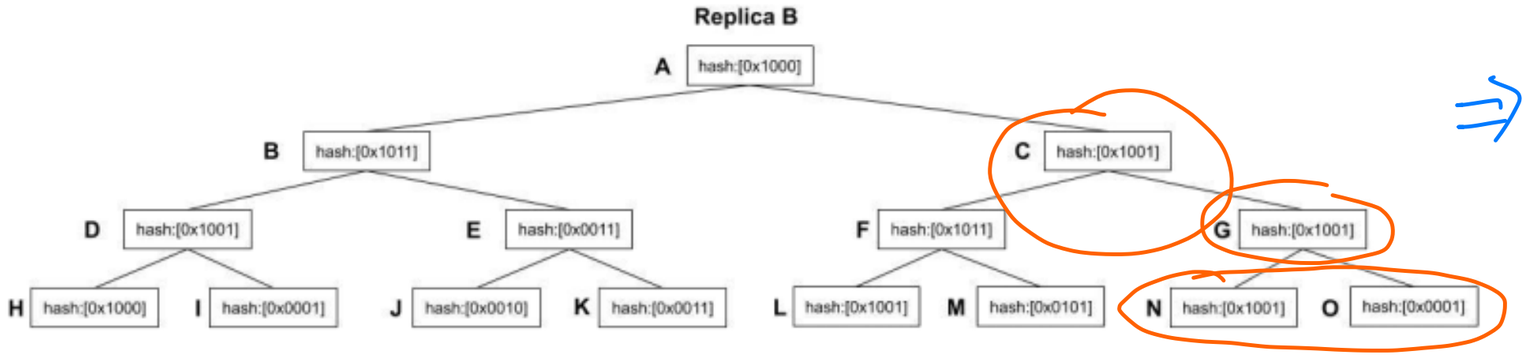
For the two pairs of trees below. Specify if it is a possible configuration as well as which nodes have to be exchanged in order to sync the trees, if applicable.


Replica A
A hash:[0x1010]
B hash:[0x1011]
C hash:[0x1101]
D hash:[0x1001]
E hash:[0x0011]
F hash:[0x1011]
G hash:[0x1100]
H hash:[0x1000]
I hash:[0x0001]
J hash:[0x0010]
K hash:[0x0011]
L hash:[0x1001]
M hash:[0x0101]
N hash:[0x1100]
O hash:[0x0100]
Replica B
A hash:[0x1000]
B hash:[0x1011]
C hash:[0x1001]
D hash:[0x1001]
E hash:[0x0011]
F hash:[0x1011]
G hash:[0x1001]
H hash:[0x1000]
I hash:[0x0001]
J hash:[0x0010]
K hash:[0x0011]
L hash:[0x1001]
M hash:[0x0101]
N hash:[0x1001]
O hash:[0x0001]

Task 2

Repeat the exercise for the following pair of Merkle Trees.





Unlikely

⇒ Either hash collision or something wrong

Exercise 4. Virtual nodes

Virtual nodes were introduced to avoid assigning data in an unbalanced manner and coping with hardware heterogeneity by taking into consideration the physical capacity of each server

Let assume we have ten servers (i_1 to i_{10}) each with the following amount of main memory: `8GiB, 16GiB, 32GiB, 8GiB, 16GiB, 0.5TiB, 1TiB, 0.25TiB, 10GiB, 20GiB`. Calculate the number of virtual nodes/tokens each server should get according to its main memory capacity if we want to have a total of `256` virtual nodes/tokens.

Just for the purpose of the exercises if you get a fractional number of virtual nodes, always round up, even if the total sum of nodes in the end exceeds `256`.