CS425 Fall 2022 – Homework 4

(a.k.a “The Interview”)

Due: December 2, 2022, 2:00PM

Mohan Amritesh Dasari

mdasari2

***Solution to Question 1:***

**Part a)**

Mining is the process of solving a computationally challenging equation or puzzle to create new blocks in the blockchain network. Every node in the blockchain network does this computational work to earn tokens or “coins” as an incentive for finishing the work.

Proof of Work (PoW) is a consensus algorithm used by majority of cryptocurrencies to verify and validate any transactions taking place on the network and create a new block in the blockchain. By default, nodes don’t trust other nodes in the network. By using this PoW algorithm the nodes come to agreement and trust other nodes in the network. This algorithm is popular because of 2 of its most important features: -

1. Very Hard to find solution to the problem, complexity of problem increases over time.
2. Very easy to verify the correctness of the solution to the problem

**Part b)**

Blockchain is essentially a digital ledger. A consensus algorithm such as Proof-of-Work and Proof-of-Stake are used by these blockchains to verify/validate transactions. This fault-tolerant mechanism is used by blockchain networks to achieve the necessary agreement on a single value among the distributed processes or nodes.

One of most popular cryptocurrencies, Bitcoin uses Proof-of-Work (PoW) as the consensus algorithm in its blockchain networks. This requires very high computational power to solve a difficult but arbitrary problem to keep all nodes in the network authorized (Trusted).

**Part c)**

The main difference between permissioned and permissionless blockchains is that permissionless blockchains don’t need you to be authorized to join the network (e.g., All public cryptocurrencies), whereas in case of permissioned blockchains, you need to have permission to access and use the private blockchain.

Public Blockchains such as Bitcoin, Ethereum, Dogecoin, etc. don’t require a new node to have permission to join the blockchains. The benefit of such networks is that unless any one individual owns 51% or more of the network, the network doesn’t belong to anyone. Everyone is free to use it. As soon as anyone has access to 51% or more nodes of the network, that person is known to be the owner of the network and hence has access to change/modify the network as they seem fit.

Permissioned Blockchains serve as private blockchain networks which essentially grant an organization to keep their private data confidential while allowing them to use the benefits of blockchain networks. Use cases for such systems include Data Storage, ID Authentication Systems, Inventory, Supply-Chain management. Examples of popular permissioned blockchains include Quorum, R3 Corda, Hyperledger Fabric, etc.

**Part d)**

Proof-of-Stake (PoS) does use less energy than Proof-of-Work (PoW).

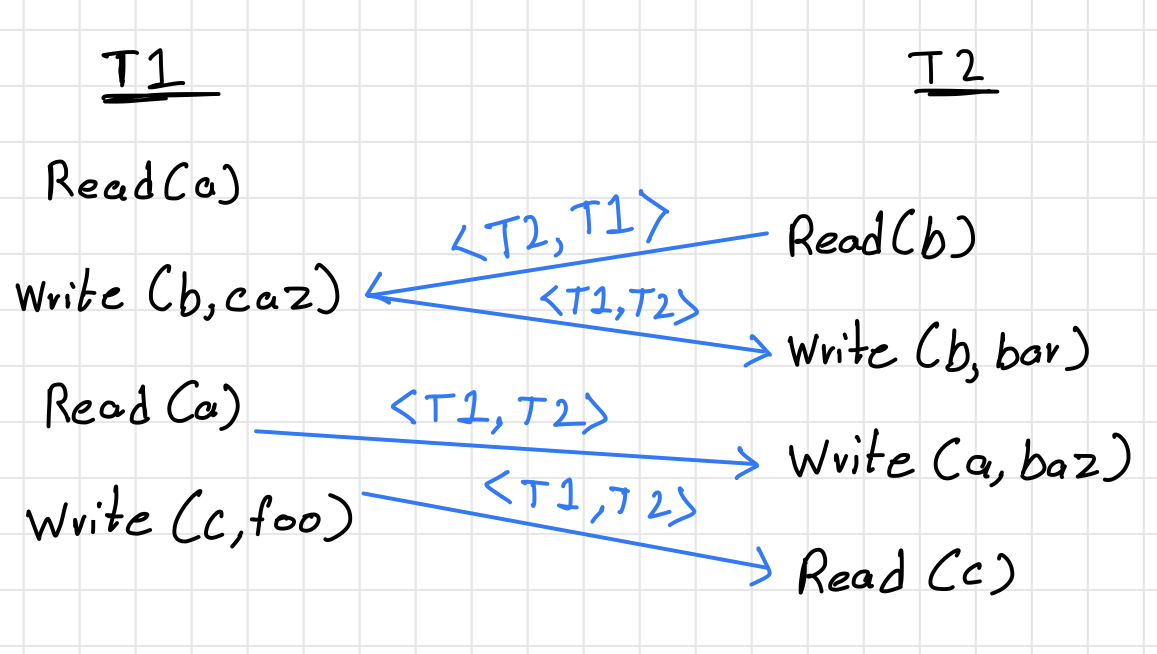
PoW requires nodes to prove that the work done and submitted by them qualifies them to get the access to write or add new transactions to the blockchain. This mining process requires high energy processing and a long processing time and often need very high-end GPUs and CPUs.

PoS was modelled such that it is a low-cost, low-energy consumption alternative to PoW algorithm. It allocates responsibility of maintaining the public ledger to a participant node with respect to the number of virtual tokens owned by it. Since there is no computationally intensive calculations in this algorithm, it doesn’t require very high energy consumption, hence it is more energy efficient than PoW.

***Solution to Question 2:***

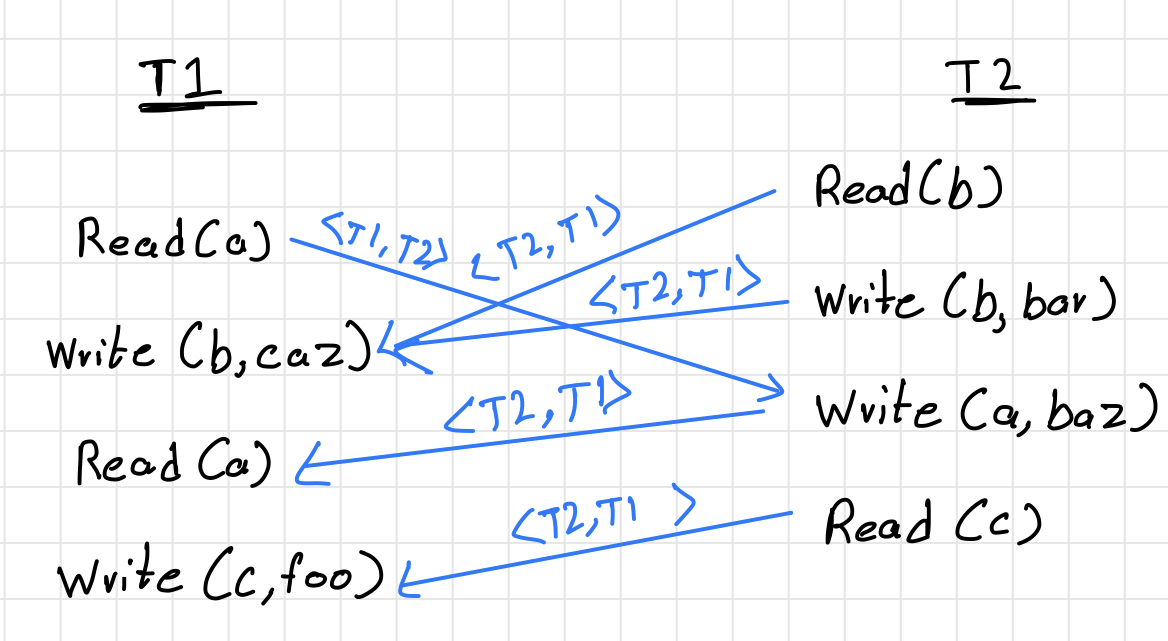
**Part a)**

This case is not serially equivalent, as every conflicting operation is not in the same order. The first conflicting operation i.e., Read(b) -> Write (b, caz) is in the order <T2, T1>, whereas the remaining conflicting operation ordering is in <T1, T2>.



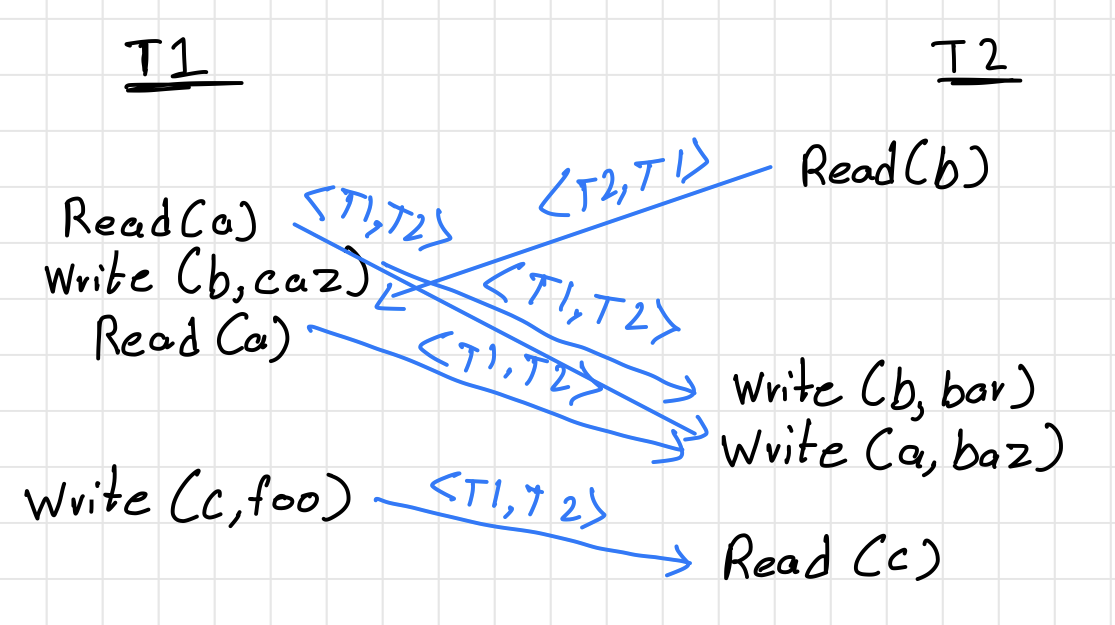
**Part b)**

This case is also not serially equivalency, as Read(a) -> Write (a, baz) is <T1, T2> ordered whereas the remaining conflicting operations are in <T2, T1> ordering.



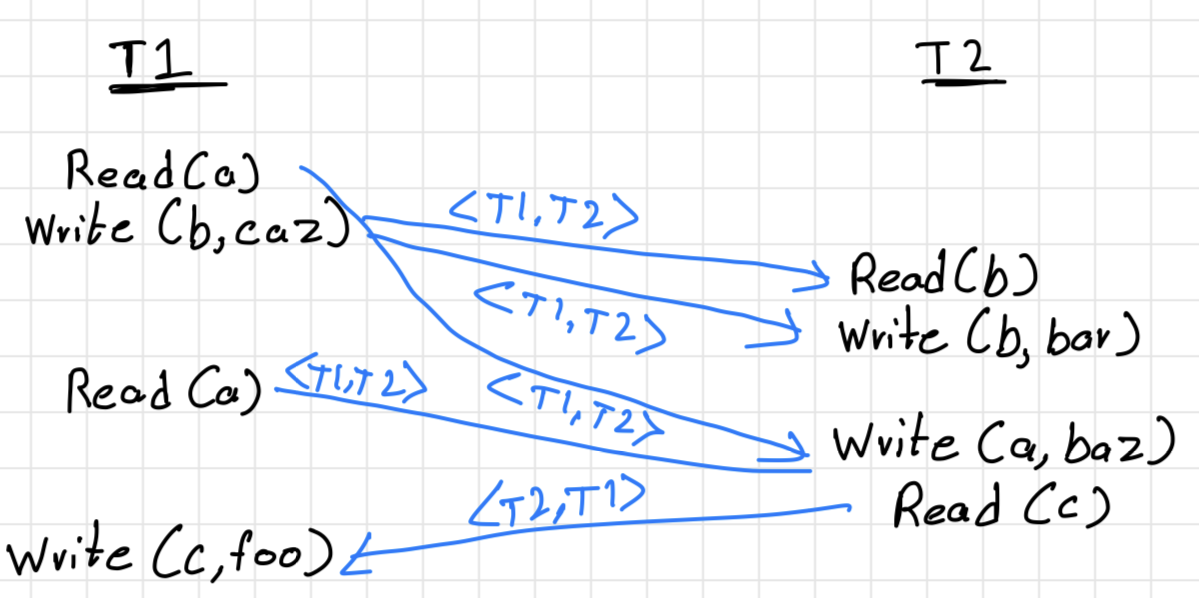
**Part c)**

This case also does not satisfy serial equivalency as Read(b) -> Write (b, caz) is <T2, T1> ordered whereas the rest of the conflicting operations are in <T1, T2> ordering.



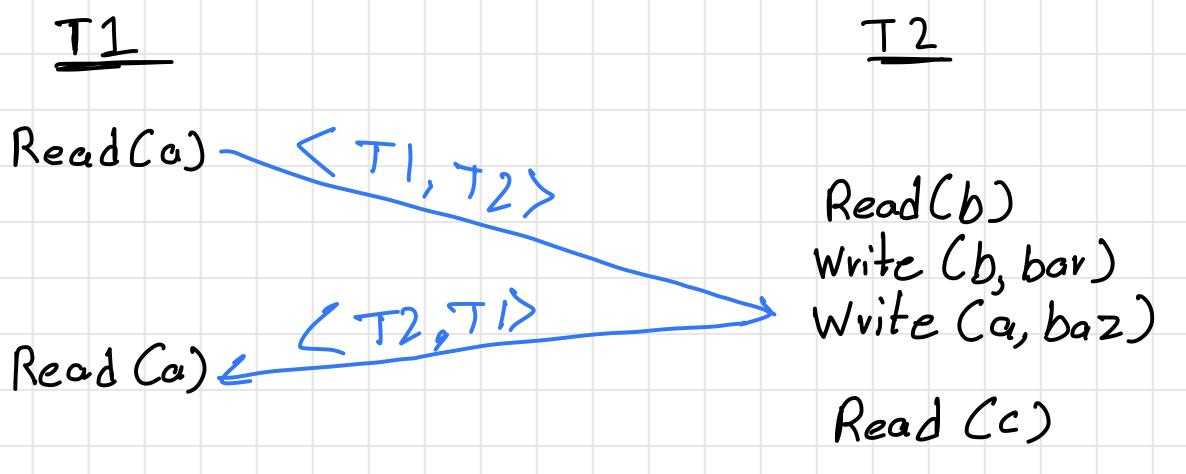
**Part d)**

This case is also not serially equivalent as Read(c) -> Write (c, foo) is <T2, T1> ordered, whereas the remaining conflicting operations are in <T1, T2> ordering.



**Part e)**

In the given case, where write operations are removed from T1, it is still not serially equivalent. Consider the case demonstrated below, the first read in T1 is in <T1, T2> ordering, whereas the second read in T1 is in <T2, T1> ordering, hence it is not serially equivalent.



***Solution to Question 3:***

Our solution that the system will not deadlock even if we try to acquire locks in the middle of transaction is right. Since we are only acquiring locks based on the lexicographical order of the words in the dictionary and since the order of words in the dictionary does not change, there will not be a situation where a word which is lower in the order requests for a lock on a word which is higher in the lexicographical order.

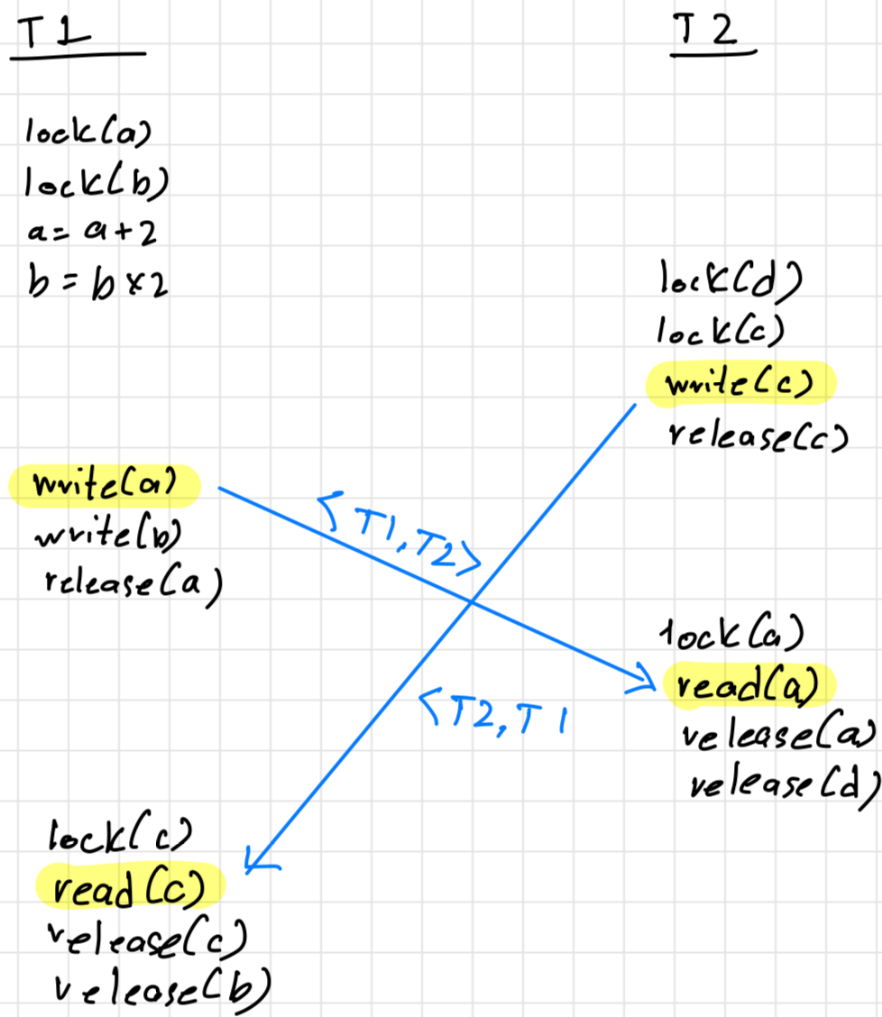


Take for example a transaction requires the locks on these particular transaction ids. Since it is mentioned that the system restricts transactions to acquire locks only in lexicographically decreasing order of object id, it does not matter when the locks are acquired as the order of acquiring locks does not change and even if we tried to acquire locks in the middle of a transaction, the system would not be a deadlock. Assuming that it is a 2-phase locking system, meaning that once a transaction has started acquiring locks, it cannot start releasing these locks until all required locks are acquired and vice-versa.

***Solution to Question 4:***

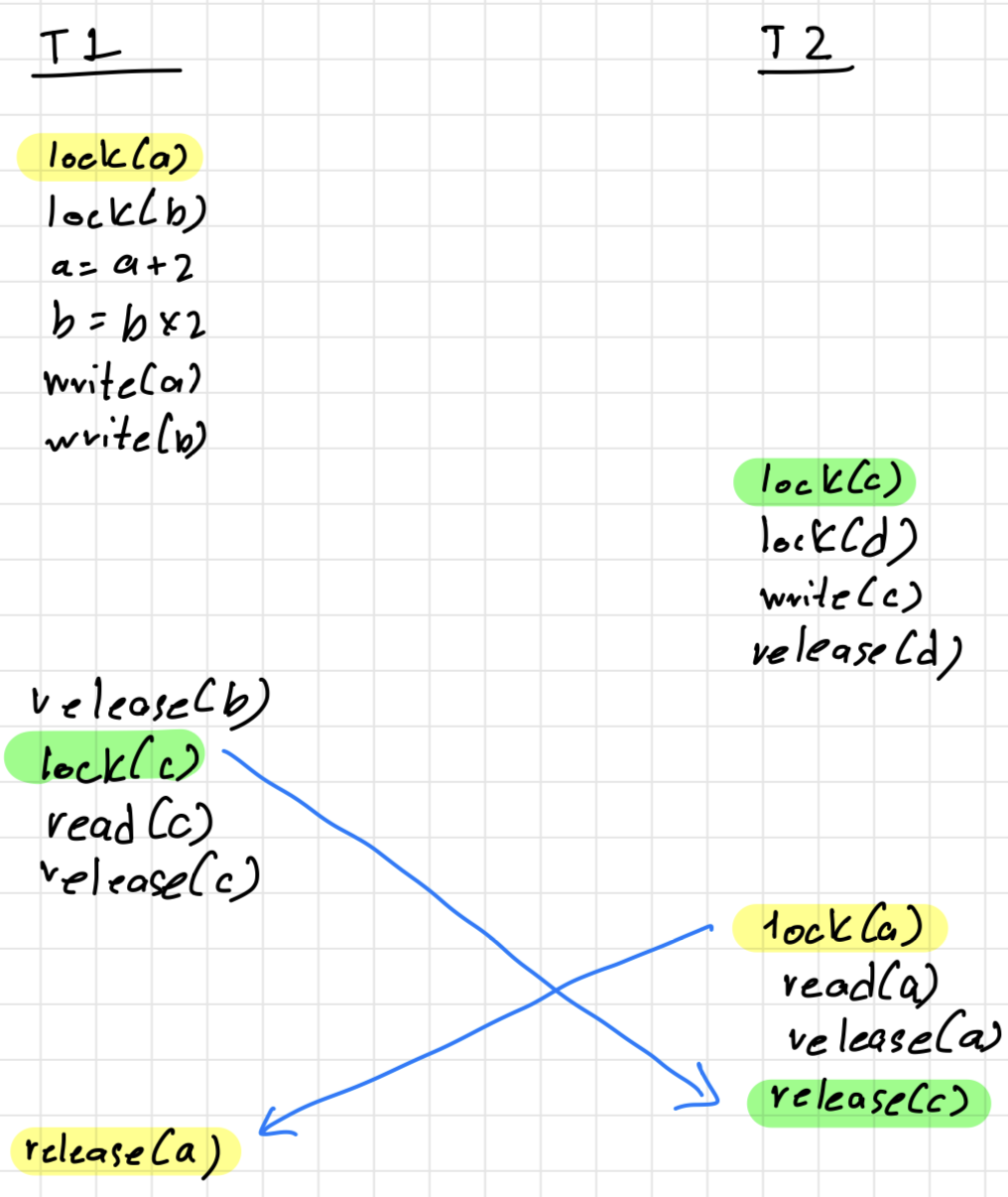
**Part a)**

Limited-Greedy two-phase locking does not satisfy serial equivalence. This is demonstrated in the example below.



**Part b)**

This limited-greedy two-phase locking algorithm can lead to a deadlock as shown in the example below.



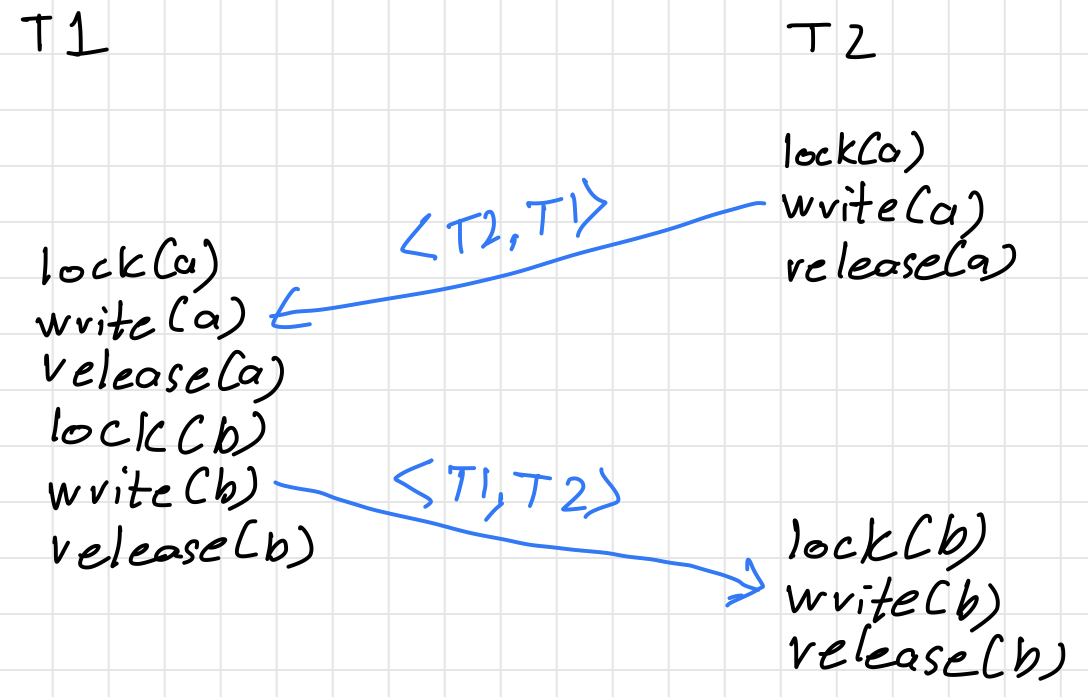
As shown in the above diagram, T1 acquires lock on object a and b, releases b and then tries to acquire lock on object c, whereas T2 acquires lock on objects c and d, releases d and then tries to acquire lock on object a which T1 has not released yet. This leads to a deadlock in the system as both transactions cannot proceed without one of them releases the required resource.

***Solution to Question 5:***

**Part a)**

This system does not satisfy serial equivalence. As mentioned in the question, we do not release the lock until the last access of that object in a transaction. Meaning if any other transaction tries to acquire lock while the first transaction is still using the object, it must wait till the first transaction releases that lock. But in case that there are multiple objects that need to be used in the transaction, this system will fail serial equivalence, as shown in the example below.

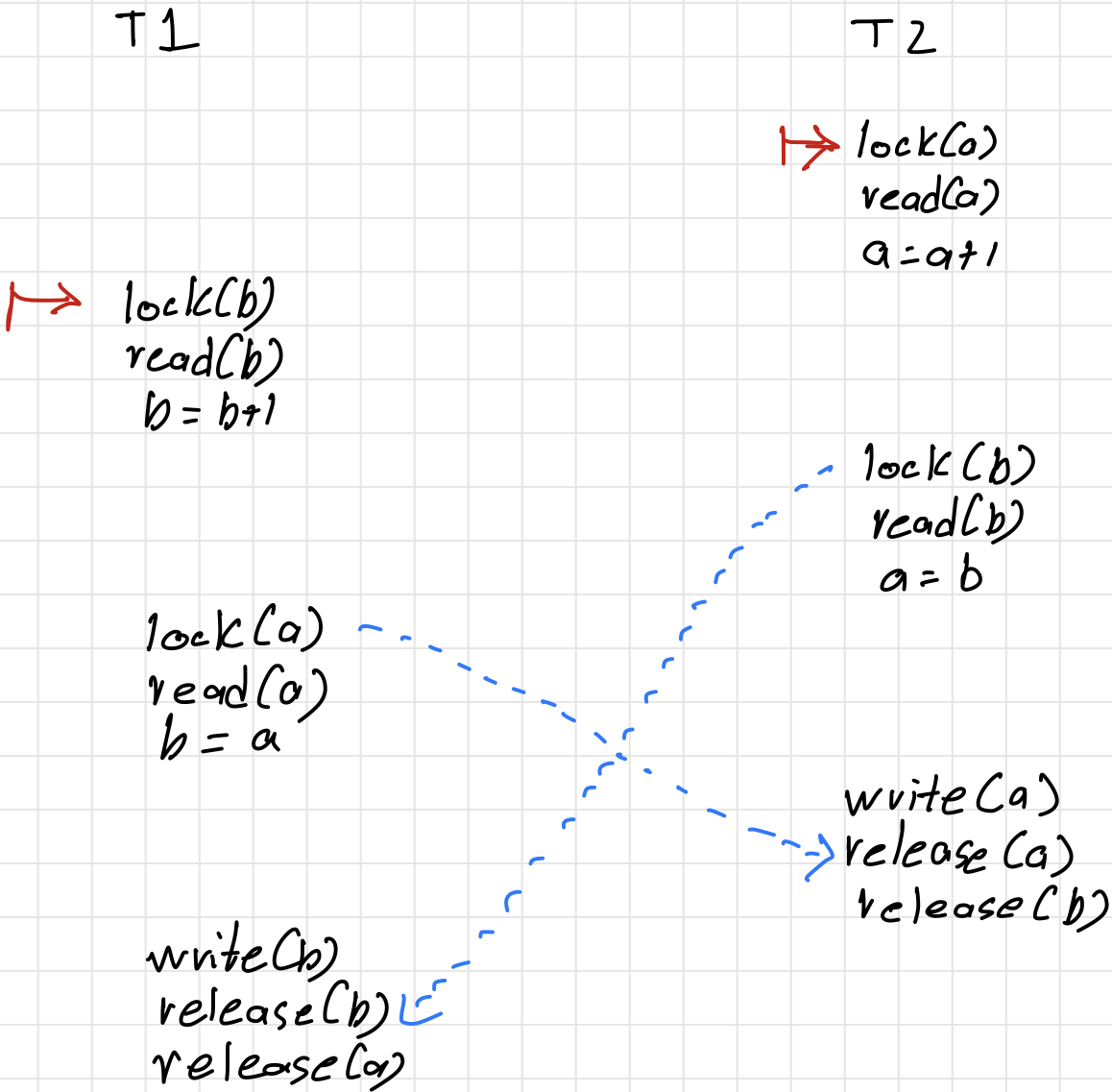
Consider this example:



Here, transaction T1 acquires the lock after T2 is done with object a, hence the ordering will be <T2, T1>, whereas in case of object b, transaction T1 acquires lock before T2 and hence T2 acquires lock after T1 is done with object b, leading to the order <T1, T2>. Hence proved, this system does not satisfy serial equivalence.

**Part b)**

This system will have deadlocks, explained in the diagram below.



Transaction T2 acquires lock on object **a**, reads a and does a+1. Transaction T1 acquires lock on object **b**,reads from it and does b+1. Now Transaction T2 is waiting to acquire a lock on object **b**, whereas Transaction T1 is waiting to acquire a lock on object **a**. Since both these transactions were waiting to get access just in time to read/write to these objects. This leads to a deadlock and hence the system can lead to a deadlock.

***Solution to Question 7:***

**Part a)**

Cloud: 20CPUs, 50GB RAM

Job1: 2CPUs, 4GB RAM  
Job2: 2CPUs, 4GB RAM

**Job1:**CPU: 2/20 = 10/100 **(CPU Intensive)**  
RAM: 4/50 = 8/100

**Job2:**CPU: 2/20 = 10/100 **(CPU Intensive)**RAM: 4/50 = 8/100

**Equations:**2x+2y<=204x+4y<=502x=2ySolving these equations we get **x=5** and **y=5**.

**Resources Allocated:**Job1 gets <10CPUs,20GB RAM>Job2 gets <10CPUs,20GB RAM>

**Part b)**

Cloud: 20CPUs, 50GB RAM

Job1: 4CPUs, 3GB RAM  
Job2: 2CPUs, 8GB RAM

**Job1:**CPU: 4/20 = 20/100 **(CPU Intensive)**RAM: 3/50 = 6/100

**Job2:**CPU: 2/20 = 10/100RAM: 8/50 = 16/100 **(Memory Intensive)**

**Equations:**4x+2y <= 203x+8y <= 505x = 4ySolving these equations we get **x=3** and **y=4**. Solving the above equations using a graphing calculator to find where the point coincides and then finding the closest integers that satisfy the above conditions**.**

**Resources Allocated:**Job1 gets <12CPUs,9GB RAM>Job2 gets <8CPUs,32GB RAM>

**Part c)**

Cloud: 20CPUs, 50GB RAM

Job1: 2CPUs, 8GB RAM  
Job2: 8CPUs, 1GB RAM

**Job1:**CPU: 2/20 = 10/100  
RAM: 8/50 = 16/100 **(Memory Intensive)**

**Job2:**CPU: 8/20 = 40/100 **(CPU Intensive)**  
RAM: 1/50 = 2/100

**Equations:**2x+8y=20  
8x+1y=50  
2x=5y  
Solving these equations we get **x=3** and **y=1**, Solving the above equations using a graphing calculator to find where the point coincides and then finding the closest integers that satisfy the above conditions**.**

**Resources Allocated:**Job1 gets <6CPUs,24GB RAM>Job2 gets <8CPUs,1GB RAM>

***Solution to Question 8:***

**Part a)** Speed and IDs of the top 5 fastest whales

The leaf nodes send a list of key-value pairs which are {id:speed} to it’s parent node. The parent node compares individual elements in the lists of key-value pairs it receives from its children nodes and makes a new list of length 5 key-value pairs of the fastest speeds from all the data it received. This node then compares the values from this list and the local data it had collected to see if it has whales with higher speed, if so replaces the ids and values in the list. It sends this new list to its parent node with the {id:speed} of the 5 fastest whales from the data it received and collected. This process goes on until it reaches the node just before the base station. Each node in this process follows the same steps mentioned above. Once this node has done the calculations and has the list of key-value pairs of whales with the fastest speeds, it sends a radio signal to the base station on the shore with this data.

**Part b)** Count of all whales currently in the system

Leaf nodes start the process by counting the number of unique ids of whales it has in its system every hour. This node then sends it to its parent node. The parent node sums up all the count of whales it receives from its children nodes, and then adds its count of unique ids of whales to this counter, and sends it to its parent. This process goes on till it reaches the node just before the base station on the shore. This node also does the same above steps and once it is done adding its count of unique ids, it sends a radio signal to the base station on the shore with this data.

**Part c)** Average speed across all whales

Leaf nodes start the process after every hour, they add all the speeds of the unique ids of whales and as well as send a count of those unique ids of whales to their parents. The parent nodes on receiving this data, sum all the speeds they get from their children nodes as well as sum up the counts of the ids of whales. These nodes then add their sum of speeds of unique ids to the resulting speed sum and add their sum of unique ids to the sum of calculated sum of ids. These nodes then send the sum of speeds and count of unique ids to their parents. And the above process continues until the node before the base station is reached. Here, the node also does the same steps of summing up the speeds of unique ids and count of unique ids. Once this is done, it divides the resulting speeds of unique ids with the resulting sum of unique ids present in the system. This gives us the average speed across all whales. This node then sends a radio signal to the base station with this data.

\*NOTE: In this part, we can also use the data transmitted in part b for the count of unique ids to reduce in-network transmission.

**Part d)** 75th percentile value of speed across all whales

From part a, instead of sending just the top 5 fastest whale ids and speeds, if every node sends a sorted list of all unique whales with their speeds and the parent nodes, just add their data to this list and sort it until the node just before the base station receives all this data. With this data, this node can calculate the 75th percentile with the following equation (n+1)\*3/4, where n is the total number of unique whales. This equation will give the index of the value of the 75th percentile. This index of the sorted list, will show us the speed of the whale with 75th percentile speed.

***Solution to Question 10:***

**Part a)**

Since Process P3 is the owner of the page and has it in Read Mode, for P3 to write to it, it first has to multicast an invalidate message to all the remaining processes. Once it receives the acknowledgement back from all the processes, it can update its State to Write Mode and can write to the page without any issues.

**Part b)**

There are several inconsistencies in this system.  
First is that if P3 is owner and has it in Read state, remaining processes can either have it in Read state or not have it at all, so P4 cannot have that same page in write state. Secondly, if P4 does have the required page in write state, it should be the owner as well as no other processor should have the page, so P3’s state would be wrong.

**Part c)**

Since P4 is the owner and is holding it in write state, this means that no other process has this page in their memory. P3 requests the page from P4. P4 sends a copy of requested page to P3. P3 on receiving this page, sends a multicast invalidate message to every process. P4 invalidates this page and P3 sets this page as Write State and Owner. P3 can now write to this page.

**Part d)**

This system is in an invalid state, as according to the question Both P1 and P2 have the page in write mode at the same time, which shouldn’t be possible. Also, P3 is the owner of the page but the state is not mentioned, either case (read or write state) the system state would be invalid. If Read state and owner on P3, the remaining Processes should be Read state and not owner. If Write state and owner on P3, no other process should have the same page.

**Part e)**

Again, the system is invalid for the same reasons mentioned in part d. If multiple processes have the same page, it can only be in Read state and only one process should have owner over that page. But in this part, processes P1, P2 and P3 all have the page in write state, which is invalid.

**Part f)**

Since P2 is the owner in this network, P3 will send a multicast to all processes to invalidate their pages. Once P3 receives the ack from all the pages, it changes the state of the page to Write and Owner. Now P3 can write to this page.

**Part g)**

P3 will send a request for the page since it does not have that page. P4 will send the page back to P3. P3 on receiving the page, will send out a multicast message to all the processes to invalidate the page on all the other processes. Once it receives the ack from them, P3 sets the state to Write state and owner. Now P3 can write to this page.

**Part h)**

Invalid state as a page can only exist in write state and owner in only one processor at a time, so both P4 and P1 can not have the same page in write state. This means that only one processor can write to page a time, hence multiple processors can’t get write state for the same page.

**Part i)**

P3 will send a request for the page, P1 will send the page back to P3. Once P3 receives the page, it multicasts message to invalidate the page from the remaining processes. P1 invalidates the page and P3 sets itself as owner and write state. Now P3 can write to this page.

**Part j)**

This is the same case as part g. So P3 sends a request for the page, P4 sends the page. P3 on receiving the page will send out a multicast to invalidate the other pages on other processes. Once P3 receives the ack from P4 and P5, it sets Write state and owner for the page.