1

a. A blockchain is the core technology that enables the creation of a decentralized and secure ledger, and cryptocurrencies are a specific application of this technology. Cryptocurrencies utilize blockchain to facilitate transactions that are secure, transparent, and decentralized. It's important to note that, even though all cryptocurrencies use blockchain, the applications of blockchain extend beyond cryptocurrencies. Blockchain finds diverse uses across various industries, ensuring secure and transparent record-keeping beyond financial transactions.

b. Mining is the process of adding transactions to the blockchain and generating new units of cryptocurrency. Conversely, Proof of Work is a consensus mechanism that requires participants to undertake computationally intensive tasks to validate transactions and enhance network security. In the initial stages, Bitcoin, along with several other cryptocurrencies, opted for Proof of Work as its primary consensus mechanism.

c. Permissioned and permissionless blockchains differ primarily in their approach to participant access and control. In a permissioned blockchain, participation is limited to a pre-defined group of known entities, a practice implemented for reasons such as privacy, operational efficiency, and regulatory compliance. This restricted access ensures that only trusted parties are part of the network. On the other hand, permissionless blockchains operate on an inclusive model where anyone can join, participate in transaction validation, and contribute to the consensus process.

d. No, Bitcoin does not use consensus algorithms such as Paxos or Raft. Instead, it uses a consensus mechanism called Proof of Work. In this system, miners compete to solve intricate mathematical problems that need a lot of computational power. The miner who solves the problem broadcasts the solution to the network, and other nodes verify its validity. If confirmed, the new block of transactions is added to the blockchain. Miners are awarded with newly created bitcoins for their participation in this process.

2.

i.

The interleaving is considered serially equivalent, despite the presence of conflicting operations, such as write(a, caz, T1) and write(a, bar, T2), or write(c, foo, T1) and write(c, baz, T2). This equivalence arises because the transactions' statements are interleaved alternately. Specifically, for two transactions to be serially equivalent, all pairs denoted as (T1, T2), where each pair comprises one operation from each transaction, must be executed in the same order (transaction order) for all objects (data) they both access.

ii.

The interviewer is incorrect because for example in our case we have two conflicting transactions from the last question, where all pairs of conflict operations are marked as (T1, T2). If we were to mark the last conflict operation to (T2, T1) and keep the first as (T1, T2), then the transaction will not be serially equivalent since all pairs of conflicting operations are not executed in the same order.

iii.

Serially Equivalent:

- **i.** write(a, caz, T1); write(a, bar, T2); read(b, T1); read(b, T2)
- **ii.** write(a, bar, T2); write(a, caz, T1); read(b, T2); read(b, T1);
- **iii.** write(a, caz, T1); read(b, T2); write(a, bar, T2); read(b, T1);
- **iv.** write(a, bar, T2); read(b, T1); write(a, caz, T1); read(b, T2);
- **v.** write(a, caz, T1); write(a, bar, T2); read(b, T2); read(b, T1);
- **vi.** write(a, bar, T2); write(a, caz, T1); read(b, T1); read(b, T2);

3.

a. In this scenario, the system ensures that transactions acquire locks for objects in the order of their initial access, and all locks are obtained at the transaction's start point. Additionally, the locks are acquired in increasing order of the special field associated with each object. This ordering eliminates the potential for circular wait (cycle) in the Wait-for graph, a crucial condition for deadlock formation. Consequently, we can confidently assert that this system is deadlock-free.

b. In this scenario, the 'special field' corresponds to the object's ID and locks are required to be acquired in lexicographically increasing order based on this ID where there is a risk of encountering deadlocks. Deadlocks may arise when transactions seek locks on multiple objects, and the order in which these locks are acquired is not uniformly maintained across all transactions. If two transactions request locks on different objects, and each transaction holds a lock that the other transaction requires, the system may experience a deadlock.

c. This scenario is similar to case b, the potential for deadlocks exists if the "special field" is defined by the unique ID (UTC time, with a random salt/nonce appended) of the object, and transactions acquire locks in lexicographically increasing order of these timestamps. The risk of deadlocks arises since the order in which objects are accessed is determined by their creation time rather than following the logical flow of transactions.

4.

System will not satisfy serial equivalence:

- For some object O1, there were conflicting operations in T1 and T2 such that the time ordering pair is (T1, T2)
- For some object O2, the conflicting operation pair is (T2, T1)
- => T1 released O1's lock after the last access to the object O1 and T2 acquired it after that
  ⇨ T2 released O2's lock after the last access to the object O2 and T1 acquired it after that

Since any object O lock can be released after the last access to the object O, which in this case T1 acquires O1 just before its access to O1 while T2 acquires O2 since T1 is currently at a point where it doesn't need to acquire O2's lock. T1 releases O1's lock after it's last access to the object O1 and T2 acquired it, while T2 releases O2's lock after the last access to the object O2 and T1 acquired it. Since these two conflicting pair has its ordering pair of (T1,T2) and (T2,T1), it makes sure that this system will not satisfy serial equivalence.

5.

a. Job 1's tasks: 6 CPU; 6 GB

Job 2's tasks: 14 CPU; 14 GB

b. Job 1's tasks: 10 CPU; 20 GB

Job 2's tasks: 10 CPU; 20 GB

c . Job 1's tasks: 6 CPU; 3 GB

Job 2's tasks: 12 CPU; 24 GB

d. Job 1's tasks: 6 CPU; 24 GB

Job 2's tasks: 12 CPU; 4 GB

7. The key difference between MOESI and MESI cache coherence protocols lies in the treatment of exclusive ownership. MOESI introduces an "Exclusive" state, indicating exclusive ownership without the need for immediate write-back to memory, whereas MESI lacks this state.

In terms of the invalidate and update protocols discussed in the lecture, both MOESI and MESI employ invalidation by marking modified data as invalid. The update protocol involves directly updating shared data. MOESI and MESI integrate these concepts to ensure consistency.

8.

a. This setup is wrong because when the process P4 is the owner and is in write state all other processes are forced to invalidate their copies of page. Making sure that other processes cannot read or write while P4 is in write state.

b. This setup is wrong because in order to write you have to be the owner, multiple processes cannot write together and if a process in write state all other processes are forced to invalidate their copies of page.

c.

- Process 3 is owner (O) has page in R state.
- processes P1 and P2 also have page in R state.
- Ask other processes to invalidate their copies of page. Use multicast.
- Mark page as (W).
- Do write.

d. This setup is wrong because multiple processes cannot write together and if a process in write state all other processes are forced to invalidate their copies of page.

e.

- Process 3 does not have page.
- Other process(es) has/have page in (R) state.
- Ask other processes to invalidate their copies of the page. Use multicast.
- Fetch all copies; use the latest copy; mark it as (W); become owner.
- Do Write

f. This setup is wrong because multiple processes cannot write together and if a process in write state all other processes are forced to invalidate their copies of page.

g. This is a tricky situation because the statement does not say anything about Process 1 being the owner, which is clearly stated in most of the previous statements. If it's a trick, then if you have to be in a write state you also have to be the owner. If it's not a trick and it's assumed that P1 is the owner and has a write state:

- Process 3 does not have page
- Process 1 has/have page in (W) state
- Ask other processes to invalidate their copies of the page. Use multicast.
- Fetch all copies; use the latest copy; mark it as (W); become owner
- Do Write

h. Similar to the last question this is a tricky situation because the statement does not say anything about Process 3 being the owner, which is clearly stated in most of the previous statements. If it's a trick, then if you have to be in a write state you also have to be the owner. If it's not a trick and it's assumed that P3 is the owner and has a write state:

- Process 1 is owner (O) and has page in W state
- Write to cache. No messages sent.

i.

- Process 3 does not have page
- Process 4 and 5 has/have page in (R) state
- Ask other processes to invalidate their copies of the page. Use multicast.
- Fetch all copies; use the latest copy; mark it as (W); become owner
- Do Write

10. Barbara Liskov, a distinguished computer scientist, made substantial contributions to distributed systems. She introduced the concept of Byzantine fault tolerance, crucial for ensuring system reliability against malicious faults. Liskov's influential work on the "Practical Byzantine Fault Tolerance" (PBFT) algorithm, presented alongside Miguel Castro, established the groundwork for creating resilient distributed systems capable of withstanding arbitrary faults. Her research has left a lasting impact on the development of fault-tolerant distributed systems. You can find more details in the paper by Castro and Liskov titled "Practical Byzantine Fault Tolerance," presented at OSDI in 1999.