Practice questions

416 2017 W2 (Winter 2018)

PQ 1

- True/False: In a practical system that uses proof-of-work, the checking of the proof has to be easy relative to the generation of the proof.

PQ 1

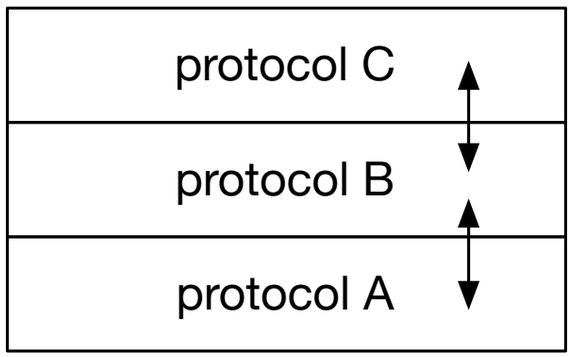
- True/False: In a practical system that uses proof-of-work, the checking of the proof has to be easy relative to the generation of the proof.

- True!

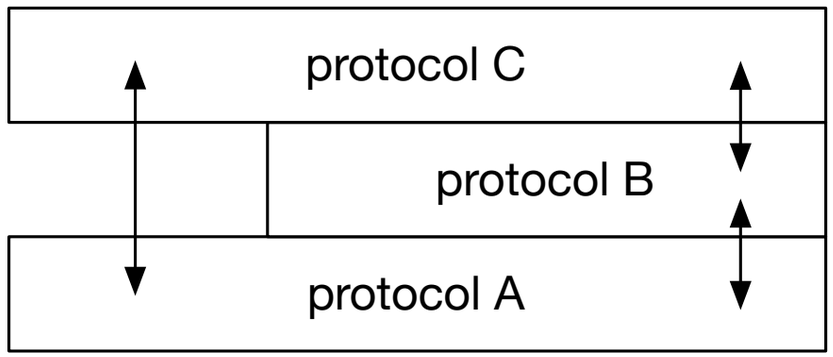
  - (If not, your system's performance has to scale with the number of CPUs that use it)

PQ 2

- You are designing a protocol stack. You have narrowed down your design to two choices. And, you know that the specification for protocol C is likely to change. Which stack design should you use?



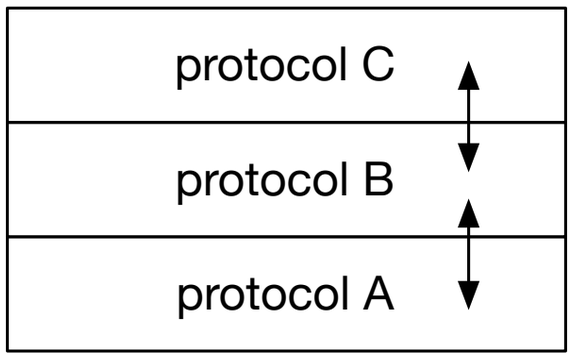
(a)



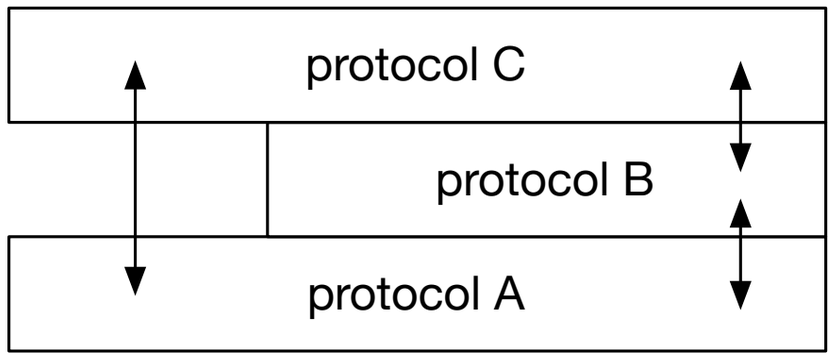
(b)

PQ 2

- You are designing a protocol stack. You have narrowed down your design to two choices. And, you know that the specification for protocol C is likely to change. Which stack design should you use?



(a)



(b)

- If protocol C changes then only protocol B would need to adapt, and not A

PQ 3

- A network element can inspect any of the protocols present in the packet. So, **why not** build e.g., a switch that is aware of HTTP and have it route packets based on HTTP information that it can extract from the packet?

PQ 3

- A network element can inspect any of the protocols present in the packet. So, **why not** build e.g., a switch that is aware of HTTP and have it route packets based on HTTP information that it can extract from the packet?

- Expensive! Line-rate HTTP processing. Also requires interpreting all the protocol below HTTP (e.g., TCP/IP)
- Higher-level protocols change, often more frequently (HTTP 2.0)
- More software can access/manipulate HTTP content (not just your OS TCP/IP stack). Requires more robustness/more security considerations.

PQ 4

- Which of these design scenarios contain elements of fate sharing?

  1. Authenticating an ATM card by requiring a retina scan

  2. Placing a passenger's checked baggage on the same flight as the passenger

  3. Attaching a spare car key to the inside of the car's bumper

PQ 4

- Which of these design scenarios contain elements of fate sharing?

  1. Authenticating an ATM card by requiring a retina scan

  2. Placing a passenger's checked baggage on the same flight as the passenger

  3. Attaching a spare car key to the inside of the car's bumper

Fate sharing: lose state information for an entity <u>if and only if</u> the entity itself is lost.

PQ 4

Fate sharing: lose state information for an entity <u>if and only if</u> the entity itself is lost.

- Which of these design scenarios contain elements of fate sharing?

1. Authenticating a debit card by requiring a retina scan

   - Not really. People with eye injuries may want to use debit cads. And, debit card loss does not result in losing your retinas!

2. Placing a passenger's checked baggage on the same flight as the passenger

   - Some! If lose plane, then lose both. If passenger is "lost", chances are the baggage is lost too. If baggage is lost/destroyed, then probably passenger is lost, too. (But, they don't fate share when not in the air!)

3. Attaching a spare car key to the inside of the car's bumper

   - Complete fate sharing. If car is lost, then key is also lost. And, key can't really be lost without the car (it's attached). i.e., key is lost, the car is lost, too.

PQ 5

- True/False: remote procedure call systems provide the same semantics as local procedure calls.

- True

- False

PQ 5

- True/False: remote procedure call systems provide the same semantics as local procedure calls.
- True
- False

LPC provides exactly once semantics; RPC cannot provide this (in the presence of failures).

PQ 6

- You decide to extend DFS in A2 with **DWRITE** mode (disconnected writes). *What are the DFS semantics you will have to revisit in your DWRITE design?* (List all that come to mind)

PQ 6.1

- You decide to extend DFS in A2 with DWRITE mode (disconnected writes). What are the DFS semantics you will have to revisit in your DWRITE design?

  - Revisit READ mode semantics — what's the "latest chunk version" if there are multiple **identical** versions available? Which one should win? How would you solve this?

  - Revisit **D**READ mode semantics —- should reads in DREAD mode observe writes in DWRITE mode?

  - Revisit Open file fetch semantics — why require a fetch for a file opened in DWRITE mode (for disconnected writing)?

PQ 6.2

- You decide to extend DFS in A2 with DWRITE mode (disconnected writes). What are the DFS semantics you will have to revisit in your DWRITE design?
  - Revisit READ mode semantics — what's the "latest chunk version" if there are multiple **identical** versions available? Which one should win? How would you solve this?
    - Change READ mode to not be *'latest'* but *"most up to date, or local first if multiple identical versions"* ?
    - Decide latest chunk based on *clocks*; use clock synchronization to decide versions offline.
    - When offline assign non-deterministic versions to writes (random offline writer wins)
    - Let server arbitrate multiple offline writes and pick winner
    - Return all the writes and let the application figure it out (conflict resolution)

PQ 7

All problems in computer science can be solved by adding another level of indirection. But that will usually create another problem." -- David Wheeler

- A2 design uses indirection. What does it use it for? (i.e., what advantages do you get from indirection in A2)?

PQ 7

All problems in computer science can be solved by adding another level of indirection. But that will usually create another problem." -- David Wheeler

- A2 design uses indirection. What does it use it for? (i.e., what advantages do you get from indirection in A2)?

- Server interposes on client requests: clients don't know each other identities, who has which chunks, who has which files open, do not observe client failures

PQ 8

- GlobalFileExists() in A2 is an idempotent operation
  - True
  - False

PQ 8

- GlobalFileExists() in A2 is an idempotent operation

- **True (with 1 client in the system)**

- **False (with multiple clients in the system)**

  Clients can make that same call repeatedly while producing the same result. In other words, making multiple identical requests has the same effect as making a single request.

PQ 9

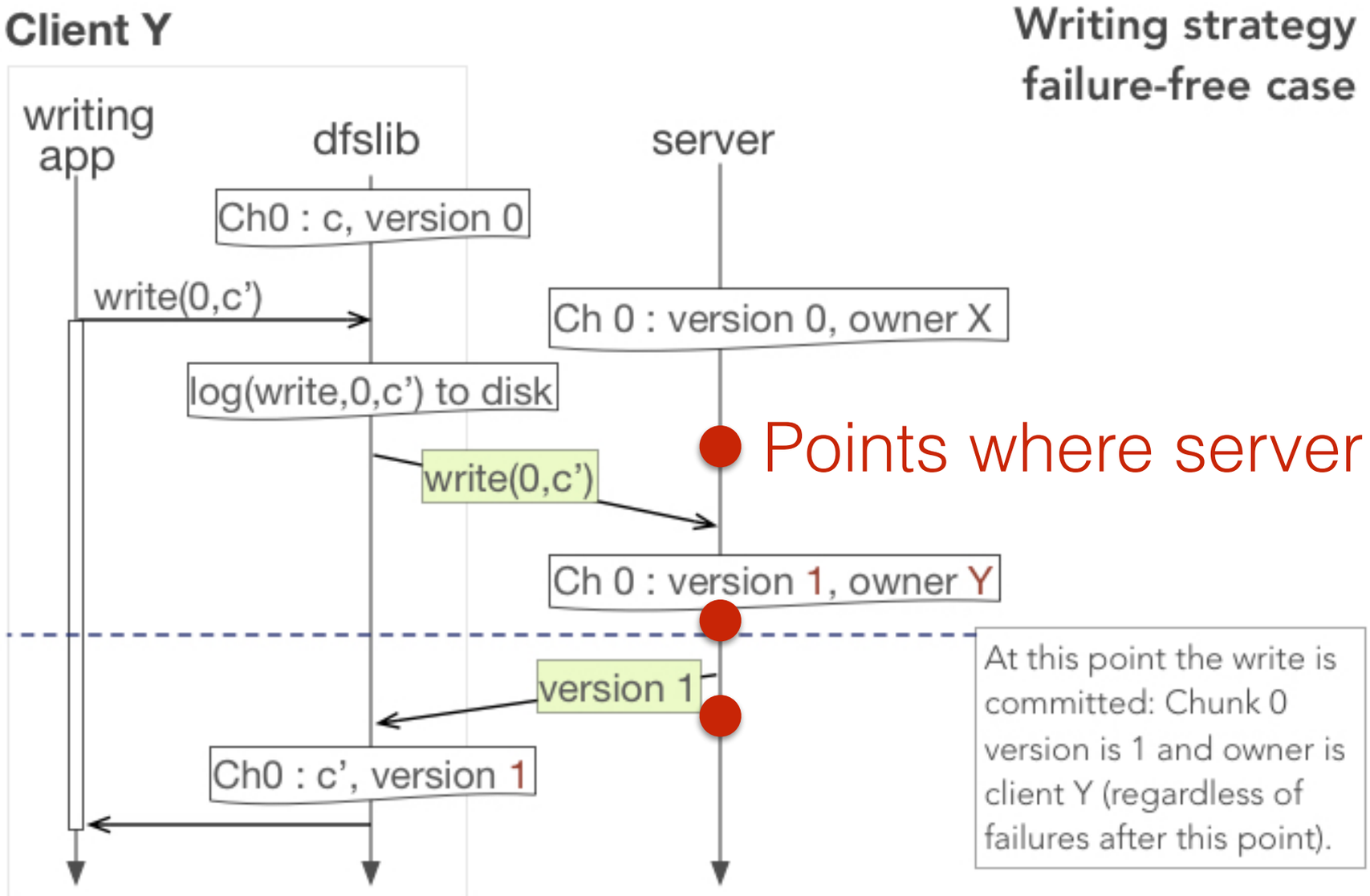
- A2 design assumes the server never fails. How would you extend the design to handle server failures?

PQ 9

- A2 design assumes the server never fails. How would you extend the design to handle server failures?
  - Treat server failures as disconnections at the client!
  - All distributed state at the server must be stored durably on disk
  - Introduce server restart/recovery procedure
  - All operations that modify server state (e.g., write of a chunk generates a new chunk version) must use a logging protocol to ensure durability (same as the suggested write protocol, but server-side)

PQ 9

- A2 design assumes the server never fails. How would you extend the design to handle server failures?



PQ 10

- An API critically determines the design of the system. Imagine that the DFS API in A2 was changed such that there was no DREAD mode and no LocalFileExists call.

Assuming that file contents would still be stored at clients, how would your design change in response?

PQ 10

- An API critically determines the design of the system. Imagine that the DFS API in A2 was changed such that there was no DREAD mode and no LocalFileExists call.

  Assuming that file contents would still be stored at clients, how would your design change in response?

- Clients no longer fetch content on open. All read/write operations streamed to the remote client replica. No disconnected mode operations means MountDFS does not need to succeed in disconnected mode.

PQ 11

- A2 makes disconnections *visible* to applications. Assume you changed A2 such that disconnections were *invisible* to the application. How would you change the DFS API and the DFS API semantics to accomplish this?

PQ 12

- The bitcoin blockchain serializes concurrent transactions into a single totally-ordered sequence
  - True
  - False

PQ 12

- The bitcoin blockchain serializes concurrent transactions into a single totally-ordered sequence
  - **True : key feature/purpose of blockchain**
  - False

PQ 13

- The BitCoin protocol is a public ledger that maintains a set of accounts. Each block in the blockchain records the updated balance of bitcoins in each account that was part of a transaction.

- True
- False

PQ 13

- The BitCoin protocol is a public ledger that maintains a set of accounts. Each block in the blockchain records the updated balance of bitcoins in each account that was part of a transaction.

- True

- **False : each block records a set of transactions, the blockchain does not explicitly store account balances**

PQ 14

Session semantics means that the first person to close the file “wins” (their copy will persist, while copies generated by later close calls will not)

- True
- False

PQ 14

Session semantics means that the first person to close the file “wins” (their copy will persist, while copies generated by later close calls will not)

- True
- **False : session semantics = last close wins**

PQ 15

Which of the following P2P systems use flooding?

- Napter
- Gnutella
- BitTorrent

PQ 15

Which of the following P2P systems use flooding?

- Napter
- **Gnutella : flood with TTL to search for items**
- BitTorrent

PQ 16

- Event A with vector clock timestamp [1,2,3] happened before event B with timestamp [3,2,1].
    - True
    - False
    - Can’t tell from timestamps alone

PQ 16

- Event A with vector clock timestamp [1,2,3] happened before event B with timestamp [3,2,1].

- True

- **False: A and B are concurrent according to their vector clocks**

- Can’t tell from timestamps alone

PQ 17

- If you are running on an unreliable network and you cannot reach a node using RPC then the node has failed.
- True
- False

PQ 17

- If you are running on an unreliable network and you cannot reach a node using RPC then the node has failed.

- True

- **False: network unreliable, could be unavailable during your RPC call.**

PQ 18

- Assume you are running Ricart-Agrawala algorithm and assume that the critical section takes a long time to run. How could each node reconstruct the sequence in which the critical section was executed?

PQ 18

- Assume you are running Ricart-Agrawala algorithm and assume that the critical section takes a long time to run. How could each node reconstruct the sequence in which the critical section was executed?

  - Use request ids issued by each node to order acquisition, since R-A respects that ordering

  - Observe deferred set (who the node blocks on and who the node is blocking)

PQ 19

- Your distributed system was running for 30 days during which time you had two outages: a disk failed and you had to replace it (outage of 3 days), and a faulty OS update had to be reverted (outage of 2 days). How many 9s of availability did your system achieve during this time?

  - 0
  - 1
  - 2
  - 3

PQ 19

- You were operating your distributed system for 30 days during which time you had two outages: a disk failed and you had to replace it (outage of 3 days), and a faulty OS update had to be reverted (outage of 2 days).

- How many 9s of availability did your system achieve during this time?
  => What was your system's availability during this time?

- Availability = time running / time should have been running

- = (30-2-3) / 30 = 25 / 30 = 83% => **0 9s of avail.**

PQ 20

- Compared to a central file hosting server, a BitTorrent swarm has which of the following features:
  - Higher scalability
  - Higher availability
  - Higher performance

PQ 20

- Compared to a central file hosting server, a BitTorrent swarm has which of the following features:
  - Higher scalability (supports more clients)
  - Higher availability (can survive more failures)
  - Higher performance (perf scales with peers)

PQ 21

- Which of the following RAID levels has the worst capacity (for a fixed set of N drives)?
  - RAID 0
  - RAID 1
  - RAID 4
  - RAID 5

PQ 21

- Which of the following RAID levels has the worst capacity (for a fixed set of N drives)?
  - RAID 0 (capacity = N)
  - RAID 1 (mirroring; capacity = N/2)
  - RAID 4 (capacity = N-1)
  - RAID 5 (capacity = N-1)

PQ 22

- RAID provides partition tolerance
  - True
  - False

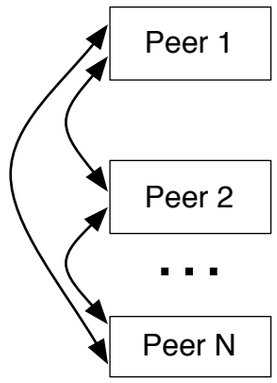
PQ 22

- RAID provides partition tolerance
- True

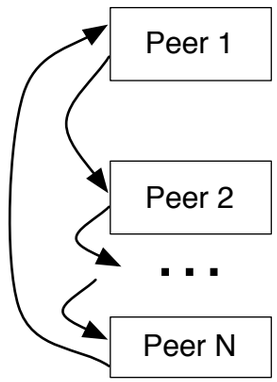
- **False (fault model assume disk fail-stop, with partitions have to reason about unreachable, but fully functional disk subset; RAID doesn’t do that)**

PQ 23

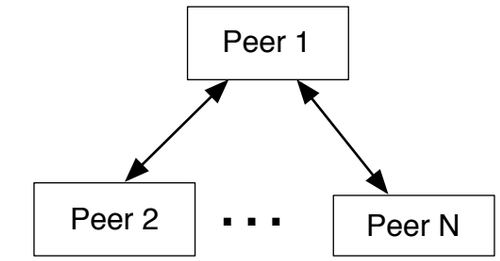
- Consider the following three topologies (e.g., in A3):



(a) all-to-all



(b) linked-list

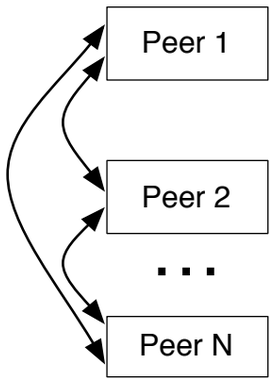


(c) star

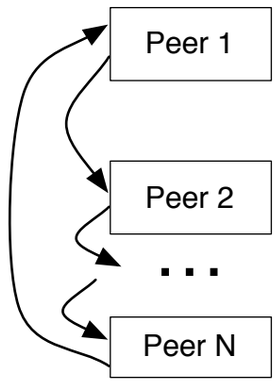
- Which topology makes it easiest for peers to detect peer failures?
- Assuming a large N, which topology (on average) impacts the fewest peers when a peer fails?

PQ 23

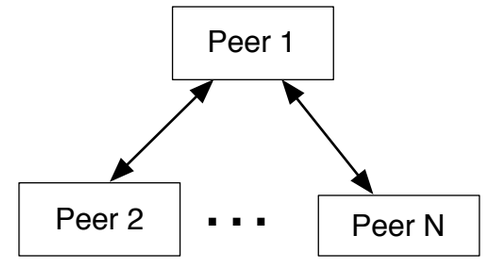
- Consider the following three topologies (e.g., in A3):



(a) all-to-all



(b) linked-list



(c) star

- Which topology makes it easiest for peers to detect peer failures? **A**
- Assuming a large N, which topology (on average) impacts the fewest peers when a peer fails? **C**

PQ 24

- A blockchain (e.g., in blockart) implements pessimistic concurrency control

- True

- False

PQ 24

- A blockchain (e.g., in blockart) implements pessimistic concurrency control

- True

- **False** (Most blockchains have optimistic CC: add block, figure out any forks later)

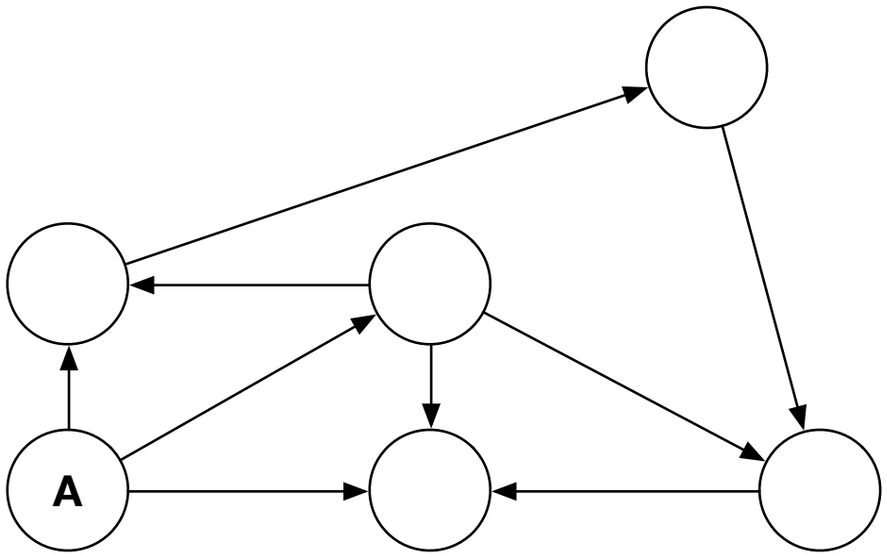
PQ 25

- BitCoin uses a “longest chain” wins policy to resolve conflicts. What if the policy was instead “shortest chain” wins. What would happen in such a system?

- And, what if the policy was “shortest chain” wins with a max branching factor of 2. How would such a system behave?

PQ 26

A block mined at node **A** in a project 1 deployment is flooded to a network of nodes that looks like this:



A. 1

B. 2

C. 3

D. 4

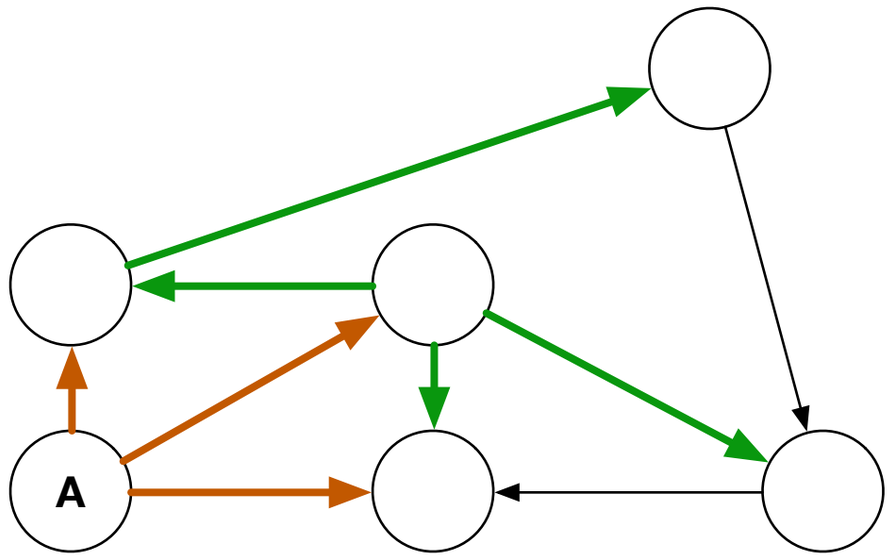
E. 5

F. 6

What is the minimum number of **rounds** before all nodes in the network receive the block?

PQ 26

A block mined at node **A** in a project 1 deployment is flooded to a network of nodes that looks like this:



A. 1

B. 2

C. 3

D. 4

E. 5

F. 6

What is the minimum number of **rounds** before all nodes in the network receive the block?

PQ 27

- Three-phase commit is a blocking protocol (blocks indefinitely during failures)

- True

- False

PQ 27

- Three-phase commit is a blocking protocol (blocks indefinitely during failures)

- True

- **False : 3PC trades off safety for liveness, it does not block during failures**

PQ 28

- Paxos is always **safe**, but not always **live**

- True
- False

PQ 28

- Paxos is always **safe**, but not always **live**
  - **True : has an execution that never terminates, but will never violate safety conditions**
  - False

PQ 29

- Given a paxos system with 2n+1 nodes, how many nodes can paxos survive (continue to operate even though this number of nodes have failed)?
    - 2
    - n
    - n+1
    - 2n
    - 2n-1

PQ 29

- Given a paxos system with 2n+1 nodes, what's the largest number of nodes can paxos survive (continue to operate even though this number of nodes have failed)?
  - 2
  - **n : if n fail, then n+1 remain, which is the smallest majority possible (for 2n+1 nodes).**
  - n+1
  - 2n
  - 2n-1

PQ 30

- You are building a Paxos-based system with n acceptors and each node is contributed by an organization that wants a “seat at the voting table”. However, some organization nodes are more important than others (think EU). You are tasked with re-designing the Paxos semantics such that **(1)** enough ‘important’ nodes must accept a value to achieve consensus, **(2)** if the important nodes fail, then Paxos should block (as if a majority has failed)

- How would you re-design Paxos for such semantics?

PQ 30

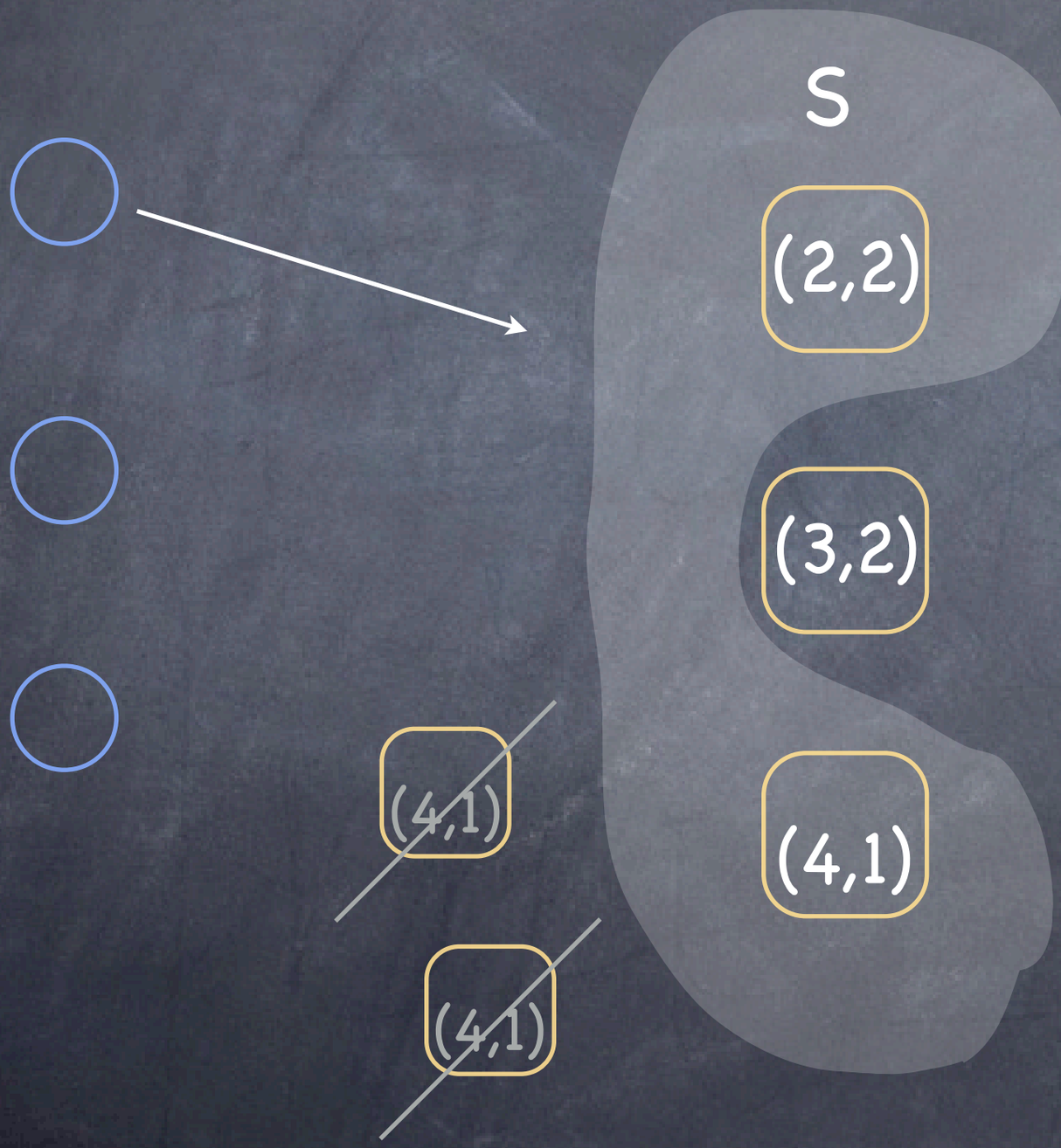
- You are building a Paxos-based system with n acceptors and each node is contributed by an organization that wants a “seat at the voting table”. However, some organization nodes are more important than others (think EU). You are tasked with re-designing the Paxos semantics such that **(1)** enough ‘important’ nodes must accept a value to achieve consensus, **(2)** if the important nodes fail, then Paxos should block (as if a majority has failed)

- Define a static “importance” value/weight associated with each acceptor. Define “acceptor majority” as an importance threshold value (sum of importance weights of the accepting acceptors must clear this threshold).

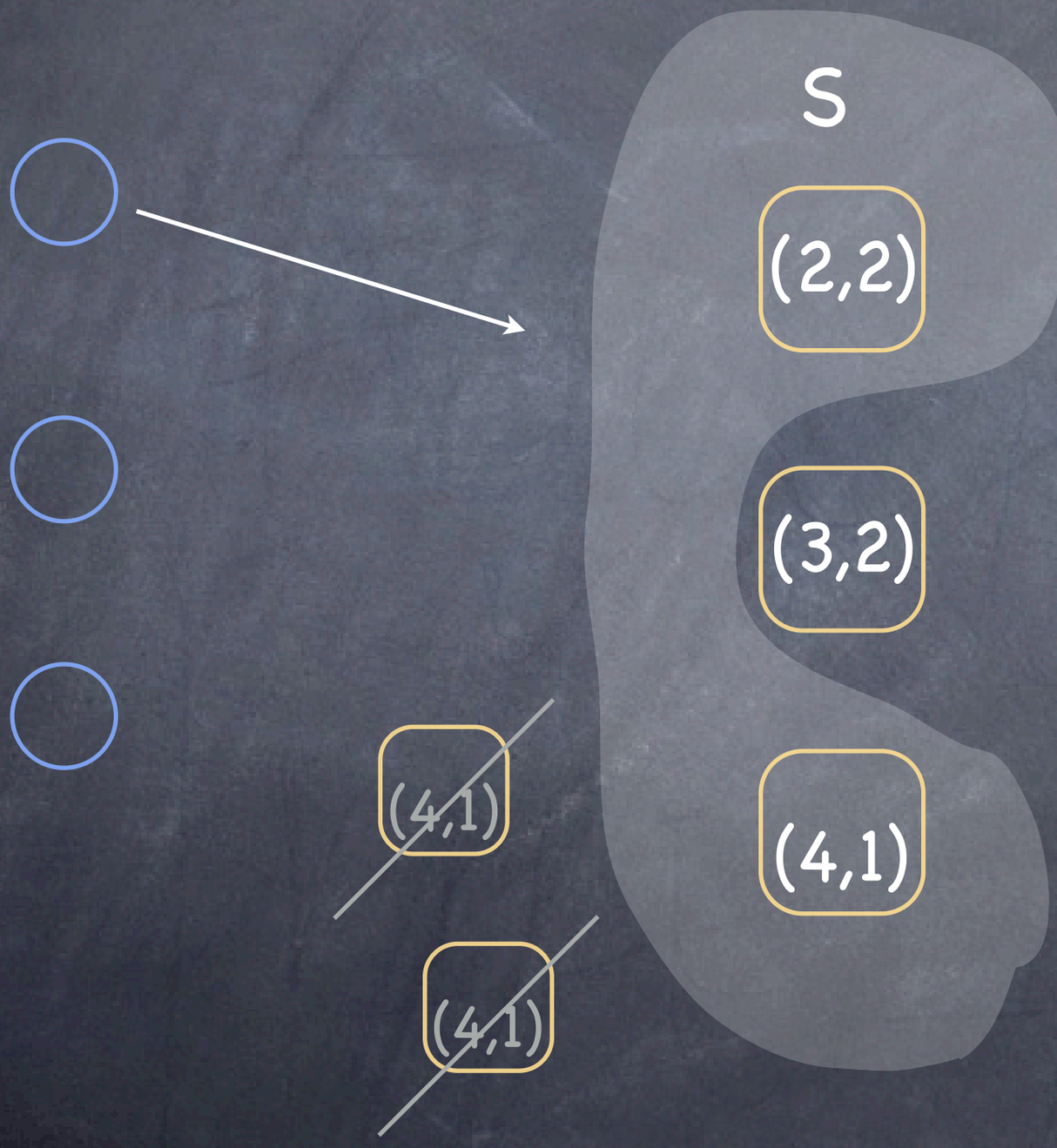
- If alive acceptors cannot clear the threshold, then the system blocks and cannot make progress.

- Defining threshold requires care: what if majority clears threshold but doesn’t know past state!?

PQ 31: Bug in slide?



PQ 31: Bug in slide?



Yes!

Majority has to be evaluated relative to original size of the Paxos group (5, not 3). So, majority should include 3 nodes (not 2).

PQ 32

- Mark each statement below as TRUE or FALSE
  - Paxos has the same round complexity as centralized 2PC *(assuming majority accept a proposal without failures/dueling proposers)*
  - Paxos has the same round complexity as centralized 3PC *(assuming majority accept a proposal without failures/dueling proposers)*
  - Paxos has arbitrarily high round complexity in the worst case

PQ 32

- Mark each statement below as TRUE or FALSE
  - Paxos has the same round complexity as centralized 2PC *(assuming majority accept a proposal without failures/dueling proposers)* **TRUE**
  - Paxos has the same round complexity as centralized 3PC *(assuming majority accept a proposal without failures/dueling proposers)* **FALSE**
  - Paxos has arbitrarily high round complexity in the worst case **TRUE**