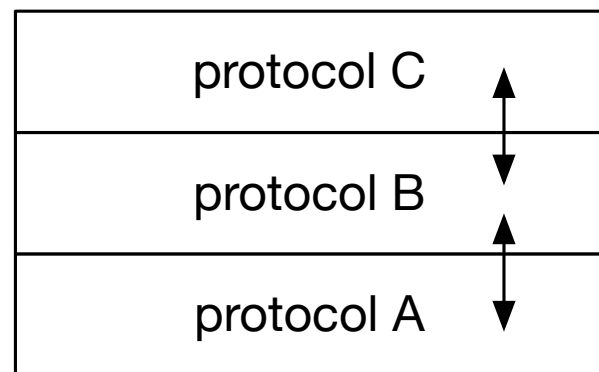# Practice questions

416 2021 W2 (Winter 2022)

These questions are intended to simulate the final exam. They cover previous lecture/assignment material that is fair game for the final exam.

PQ 1

- 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?
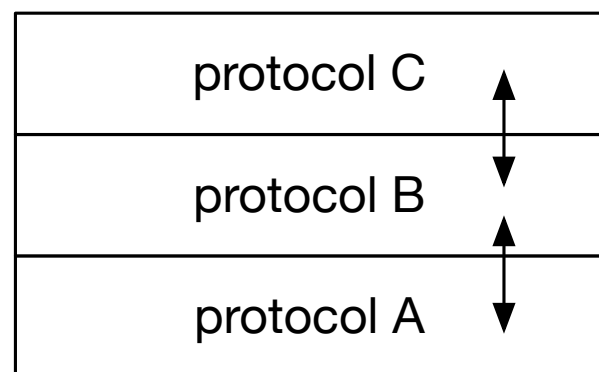

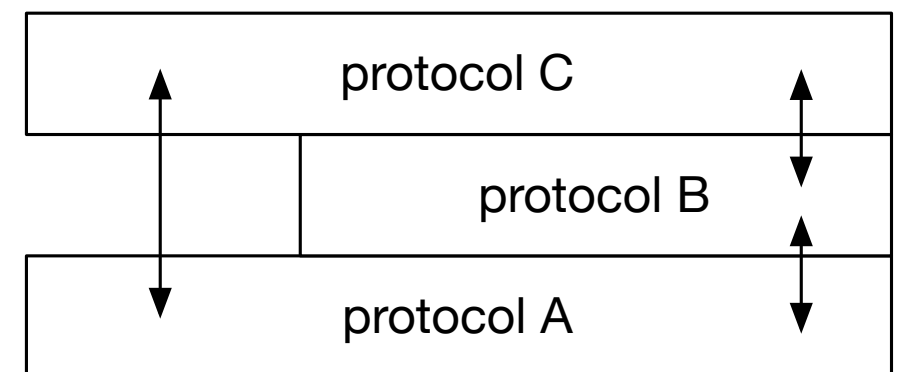(a)   (b)
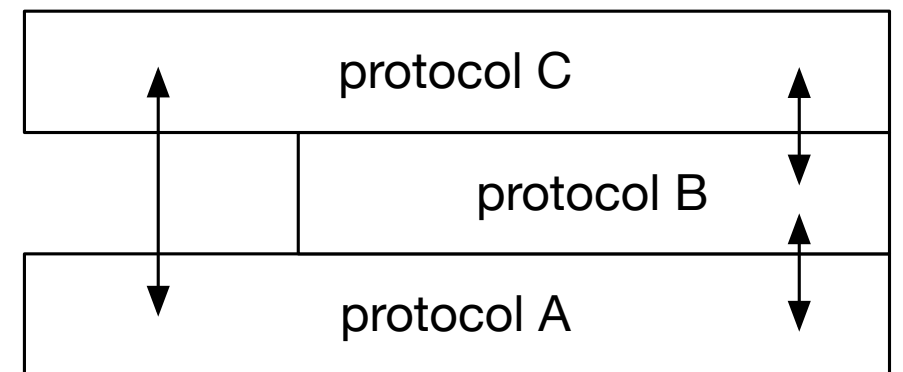
PQ 1

- 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?

protocol C
protocol B
protocol A

(a)

protocol C	
	protocol B
protocol A	

(b)

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

PQ 2

- 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 2

- 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 requires more memory/cpu. Also requires interpreting 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.*

- But, it's not impossible! See "software middleboxes" or "network function virtualization"

PQ 3

- You plan to disrupt the RPC concept by not only sending arguments to the remote procedure, but also sending the *procedure* itself (as a lambda fn). What challenges do you expect with this idea?

- Breakout rooms + 5 minute discussion with your peers

PQ 3

- You plan to disrupt the RPC concept by not only sending arguments to the remote procedure, but also sending the *procedure* itself (as a lambda fn). What challenges do you expect with this idea?

- Breakout rooms + 5 minute discussion with your peers

- Discussion:

  - RPC server env may be rather different than the caller: lambda fn may be in a lang that can't be interpreted by the server

  - Env typical RPC: addresses/memory

  - Do you pass a pointer to fn, or an object? How do you wrap the code in a way that makes it standalone on the server. Need to capture all you need.

  - SECURITY! Fn needs some kind of sandbox (docker? VM?)

  - State — do you let the function store state somewhere?

  - Remote code execution — always bad? … browsers!

  - Javascript… it sort of does exactly what's above!

PQ 3

- You plan to disrupt the RPC concept by not only sending arguments to the remote procedure, but also sending the *procedure* itself (as a lambda fn). What challenges do you expect with this idea?

  - Defining invocation semantics: at most once/at least once. What happens on failures?

  - Defining the capabilities of the procedure: can it read files? Can it open connections/send data?

  - Can it be STATEFUL!? Where do we store state?

  - Related: defining allowable side effects, if any

  - Guaranteeing determinism (procedure should probably run identically regardless of host server). Have to determinize calls to random, env state like files. Have to provide environment that is identical across machines (e.g., runtime language-based VM like a JVM).

  - Encoding of arguments (as in RPC)

  - Encoding of the procedure + env state that it needs besides args

PQ 4

- True/False: According to the A1 specification, if the nim server fails mid-game, the nim client will send the same *StateMoveMessage* indefinitely

- PQ4 zoom poll

PQ 4

- True/False: According to the A1 specification, if the nim server fails mid-game, the nim client will send the same *StateMoveMessage* indefinitely

- True! In a sense, the A1 aim client cannot tell if the server is slow or failed.

PQ 5

- On the client-side NFS caches only those parts of the file that the local processes have read/written.

  *What are the **pros** and **cons** of NFS caching files **in their entirety**?*

  - Con : Caching entire file = more network load (less timely? Not an issue if delivering content that's being read first)

  - Con : More conflict between clients accessing the same file (more in cache => more bits to conflict on)

  - Con : Takes up more memory (for parts of file that may not be useful..)

  - Pro : Easier to implement!

  - Pro : Fewer requests! If you need the entire file, this is more efficient!

  - Pro : More failure resistant (caching entire file => can survive server crashes!)

  - Con : caching things that you may not need (may only want to write 1 byte in a 1 TB file)

PQ 5

- On the client-side NFS caches only those parts of the file that the local processes have read/written. *What are the pros and cons of NFS caching files in their entirety?*

- Pros: **performance** (more reads handled locally, don't hit the server), **scales better** (can support more clients), **file access is faster** (assumes: client cache faster than server cache; client memory faster than server disk), **robust to server crashes** (client can read cache instead of going to server)

- Cons: **more resources at client** (client needs space for the cache), **scales worse** (with larger files), **worse\complex consistency** (there is less sync. between clients), **security** (trust clients with *all* file data),

PQ 6

- You decide to create a nim server that allows nim clients to play against each other (instead of against the server).

  *What is the **minimal** way in which you would change A1 (protocol/client/sever) to enable this client-v-client game play?*

  - Need to determine who moves first! Server decides!

  - Maybe prevent cheating? (would be a nice v2 feature)

  - Matching clients is the key difficulty — how? Server's problem

  -

PQ 6

- You decide to create a nim server that allows nim clients to play against each other (instead of against the server). What is the **minimal** way in which you would change A1 (protocol/client/sever) to enable this client-v-client game play?

- No changes to protocol or client-side logic. Only change the server:

  - Server receives opening move and blocks waiting for another client's opening move.

  - Once it has received two opening moves, the server sends client 1 a new game board.

  - When client 1 responds with a move, the server forwards the game board to client 2 as its opening move reply and waits for a move from client 2.

  - After this point the server acts a proxy between the two clients.

PQ 7

- Which file system can support more clients, given a server that runs on identical hardware and a typical University file access workload? [Choose one answer]

  A. NFS

  B. AFS

PQ 7

- Which file system can support more clients, given a server that runs on identical hardware? [Choose one answer]
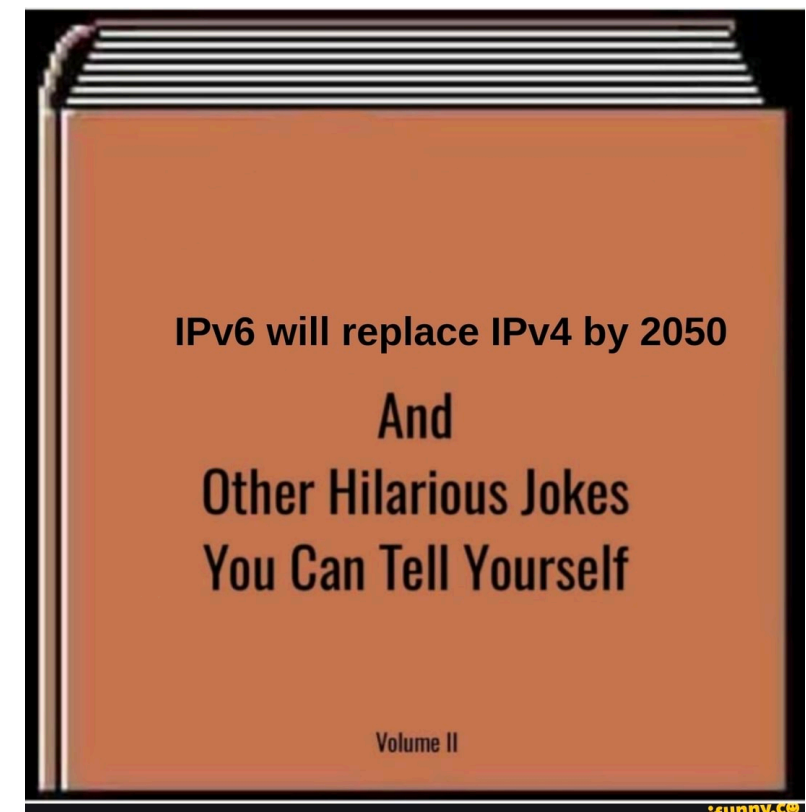
    A. NFS

    B. **AFS**   [AFS pushes client load from the server by caching entire files on the client side. It is strictly more scalable (in terms of number of clients) than NFS.]

PQ 8

- A CDN can improve which of the following for the client:

  - Latency

  - Security

  - Throughput

  - Consistency

  - Availability

IPv6 will replace IPv4 by 2050

And
Other Hilarious Jokes
You Can Tell Yourself

Volume II

PQ 8

- A CDN can improve which of the following for the client:

  - Latency (CDNs nodes closer to client)

  - Security (2 administrative domains)

  - Throughput (CDN nodes more lightly loaded)

  - Consistency (nope)

  - Availability (CDN nodes in same region as client)

PQ 9 (discussion)

- How would you map the network of servers run by a large CDN provider (e.g., Akamai)?

  - Map ~ which servers are available, where they are located, their characteristics (topology, connectivity, latency, bandwidth, server specs), etc. Maybe include DNS servers and underlying policies.

  - Try clients from different locations

  - traceroute to nytimes.com? Useful?

  - 1. Buy cloud resources — around the world (depends on granularity of the map that you want).

  - 2. VPN using those resources to appear as if you are at a different location (geographically) = spoofing your location

  - 3. Traceroute nytimes.com from the different locations — find out route. Reveals hops to a destination at the IP level.

  - 4. Make DNS requests (with a cold DNS cache) — learn about the DNS servers used by the CDN, and ultimately the server that the CDN wants you to use for nytimes.com

  - 5. Send many requests + store results. Build up a dataset (BIGDATA). Take 340. Use MACHINE LEARNING to identify patterns in data.

  - => DERIVE patterns and reconstruct map of CDN

PQ 9

- And how would you use this information to optimize (performance/availability/etc) your browsing performance?

  - -1. Buy a house close to a fast CDN server :-)

  - 0. Bypass the DNS and use the CDN server directly (cut down on RTT to DNS)

  - 1. Objective (availability), Input (set of servers) => Strategy that tells you which server to use when for a specific objective.

PQ 10

What is a blockchain?

A. An eventually consistent data structure

B. A set of distributed protocols

C. A fault tolerant data storage system

D. An implementation of an immutable ledger

E. All of the above

PQ 10

What is a blockchain?

A. An eventually consistent data structure

B. A set of distributed protocols

C. A fault tolerant data storage system

D. An implementation of an immutable ledger

E. **All of the above**
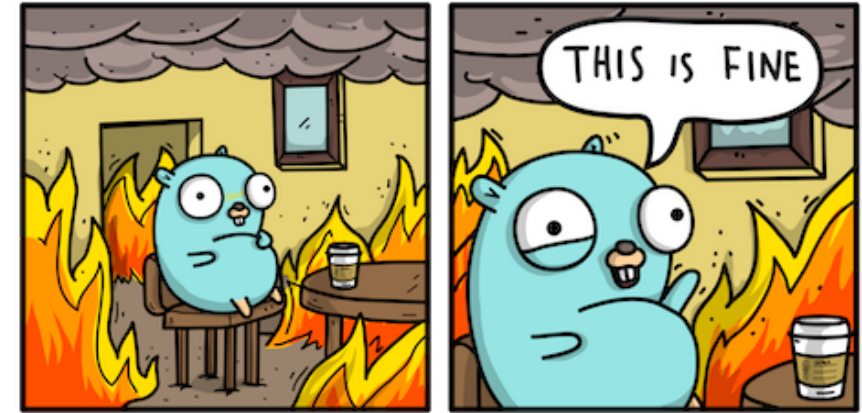
PQ 11

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

   A. Higher scalability

   B. Higher availability

   C. Higher performance

   D. All of the above

PQ 11

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

  A. Higher scalability (supports more clients)

  B. Higher availability (can survive more failures)

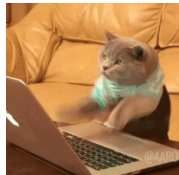 C. Higher performance (perf scales with peers)

  D. All of the above

PQ 12
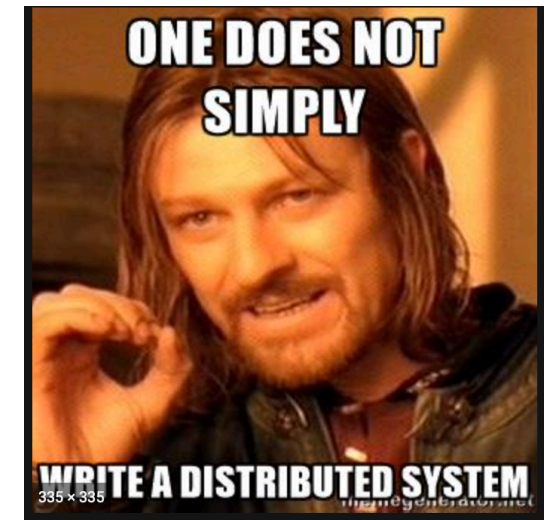
- Which of the following statements is false? (N is the number of peers in the network)

  A. Gnutella has O(N) search scope

  B. Napster has O(1) search scope

  C. BitTorrent has O(1) search scope

PQ 12

- Which of the following statements is false?

  A. Gnutella has O(N) search scope

  B. Napster has O(1) search scope

  C. BitTorrent has O(1) search scope <— False

    - BitTorrent has O( ) scope since it's outside the model of the system! It doesn't provide a lookup function.

PQ 13

- *"To take control over the BitCoin ledger (e.g., to double spend) an attacker needs to control at least X of the processing power in the network."* What's the right X?
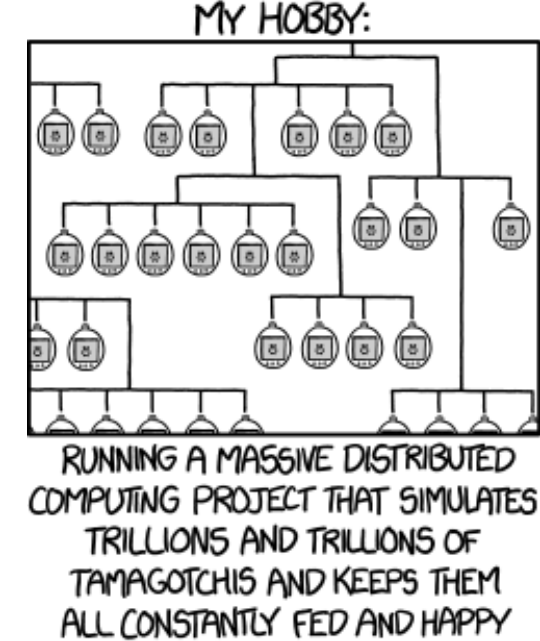
A. 10%

B. 25%

C. 33%

D. 50%

E. 75%

When your boss thanks you for staying late to work but you were just watching the price of Bitcoin and lost track of time

PQ 13

- *"To take control over the BitCoin ledger (e.g., to double spend) an attacker needs to control at least X of the processing power in the network."* What's the right X?

  A. 10%

  B. 25%

  C. 33%

  D. 50% (Controlling longest chain requires majority processing power, the "51% attack")

  E. 75%

PQ 14

MY HOBBY:
RUNNING A MASSIVE DISTRIBUTED COMPUTING PROJECT THAT SIMULATES TRILLIONS AND TRILLIONS OF TAMAGOTCHIS AND KEEPS THEM ALL CONSTANTLY FED AND HAPPY

- 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 14

- 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, i.e., A || B**

    - Because [1,2,3] is not > then [3,2,1]; and, [3,2,1] is not > then [1,2,3]

    - **Index 0: 1 < 3**

    - Index 1: 2 == 2

    - **Index 2: 3 > 1**

    - Because of index 0 and 2, neither timestamp dominates the other **=>** underlying events associated with these timestamps are concurrent.

  - Can't tell from timestamps alone

PQ 15

- BitTorrent finds out which chunks exist in the swarm by using flooding.

  - True

  - False

PQ 15

- BitTorrent finds out which chunks exist in the swarm by using flooding.

  - True

  - False: BitTorrent does **not** use flooding. All nodes know the the hashes of all the chunks, peers exchange lists of hashes to learn about each other's chunks.

PQ 16

- The tracing library uses which of the following logical clock mechanisms:

  A. Lamport clocks

  B. Vector clocks

  C. Hierarchial vector clock

  D. Matrix clocks

PQ 16

- The tracing library uses which of the following logical clock mechanisms:

  A. Lamport clocks

  B. Vector clocks : you can see these vector timestamps in the output files

  C. Hierarchial vector clock

  D. Matrix clocks

PQ 17

- In A2 you developed fcheck. Let's say that your fcheck is structured as two go routines:

    1. A go routine to receive heartbeats and reply with acks

    2. A go routine to send heartbeats to the monitored node.

- **How would you change your fcheck design to monitor multiple nodes simultaneously?**

    - Create a go routine per node that you're monitoring

        - Pro: re-use existing go routine

        - Con: many go routines (as many as nodes monitored)

    - Multiprocessing — spin up a process per node being monitored

        - Strictly worse (harder to coordinate/more overhead)

        - But easier to reuse code..maybe

    - Use a single go routine for sending all heartbeats — if a single UDP connection: you can send in a loop

        - More data structures to record information for each monitored node

PQ 18

Ricart-Agrawala uses which of the following mechanisms?

A. Atomic clocks

B. NTP-synchronized clocks

C. Lamport clocks

D. Vector clocks

E. Other special ninja type of clock not listed above

PQ 18

Ricart-Agrawala uses which of the following mechanisms?

A. Atomic clocks

B. NTP-synchronized clocks

C. **Lamport clocks : provides mut. exclusion + fairness; requires total order: $e < e'$ implies $L(e) < L(e')$**

D. Vector clocks

E. Other special ninja type of clock not listed above

PQ 19

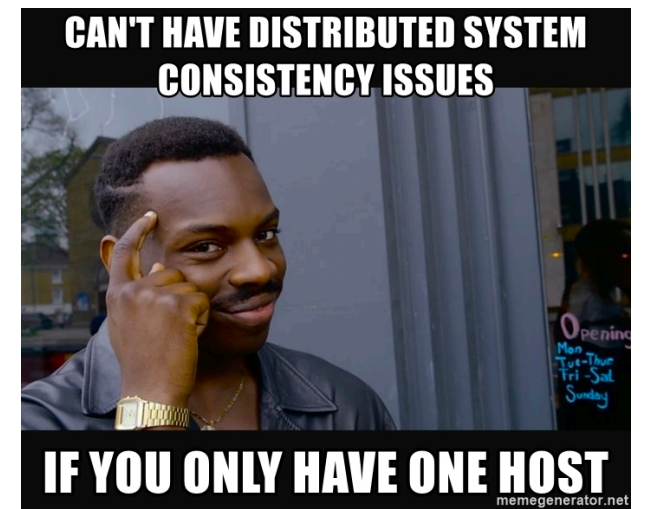- RAID uses complement sum for error detection

  A. Yes

  B. No

PQ 19

- RAID uses complement sum for error detection

  A. Yes

  B. No [RAID uses Parity]

PQ 20

CAN'T HAVE DISTRIBUTED SYSTEM CONSISTENCY ISSUES
IF YOU ONLY HAVE ONE HOST

You are attempting to subvert your co-worker's machine by selecting a RAID level that wastes the most amount of physical space. Which RAID level should you choose?

- RAID 0

- RAID 1

- RAID 4

- RAID 5

PQ 20

You are attempting to <u>subvert</u> your co-worker's machine by selecting a RAID level that wastes the most amount of physical space. Which RAID level should you choose?

- RAID 0 : N

- **RAID 1 : mirroring (N/2 capacity)**

- RAID 4 : N - 1

- RAID 5 : N - 1

PQ 21

- Primary-backup replication is more fault-tolerant than quorum replication.

    - True

    - False

PQ 21

- Primary-backup replication is more *more available* than quorum replication.

  - True

  - False [If the primary dies, the entire system halts. Not so with a quorum-based system: as long as majority is alive, the system is available.]

PQ 22

- 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

- 2

- 1

- 3

PQ 22

- 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.**
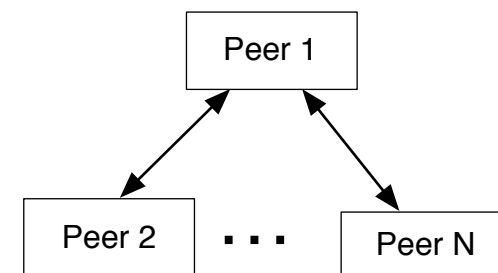
PQ 23

- Consider the following three topologies (e.g., in P1):
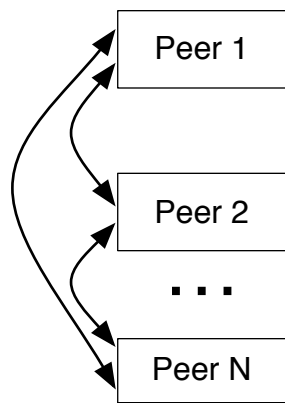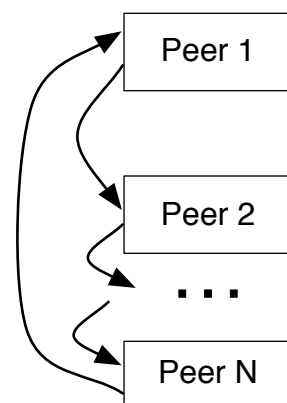
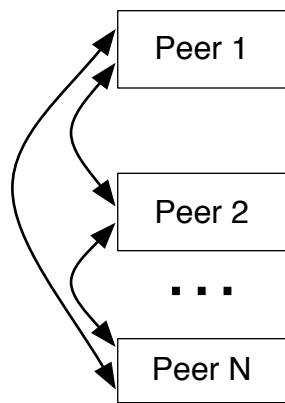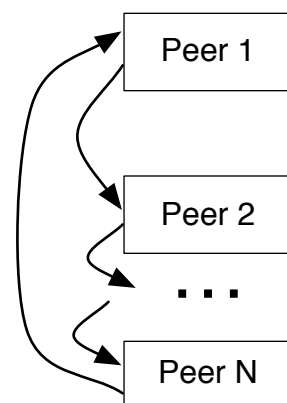(a) all-to-all   (b) linked-list   (c) star

- Q1: Which topology makes it easiest for peers to detect peer failures?

- Q2: Assuming a large N, which topology (on average) impacts the fewest peers when a peer fails?
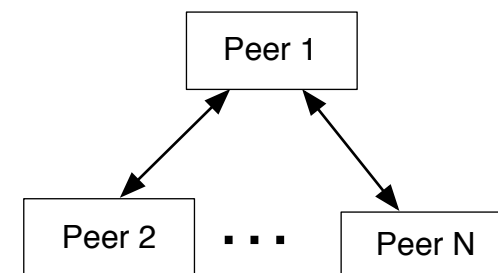
PQ 23

- Consider the following three topologies (e.g., in P1):

(a) all-to-all (b) linked-list (c) star

- Q1: Which topology makes it easiest for peers to detect peer failures? **A**

- Q2: Assuming a large N, which topology (on average) impacts the fewest peers when a peer fails? **C**

PQ 24

- Redo logging writes new values to a log

  - True

  - False

PQ 24

- Redo logging writes new values to a log

  - True [redo logs what must be re-done after a failure: committed txn will commit in recovery]

  - False

PQ 25

- Chain replication has a higher latency for update operations than primary backup

  - True

  - False

PQ 25

- In general, chain replication has a higher latency for update operations than primary backup

  - True [primary backup has a fixed latency of 2 RTT per update, chain replication latency is a function of the number of replicas, > 2 RTT]

  - False

PQ 26

- You got some VC funding and you decide to commercialize A3 as a *"big data storage solution"*. But, first, you have to get rid of the central coordinator in A3's design. How would you go about this?

PQ 26

- You got some VC funding and you decide to commercialize A3 as a *"big data storage solution"*. But, first, you have to get rid of the central coordinator in A3's design. How would you go about this?

  - Coord func: routing clients to head/tail, chain maintenance (failure handling)

  - Something needs to track the chain. Let each server keep track of who is in the chain. Each server monitors every other server (or adjacent server).

    - Why not clients? This would then require clients to participate in failure recovery. Clients are ephemeral — they come and go as they please.

  - No one keeps track of the chain (!). The clients attempt servers given a list to find out which one is the head/tail.

    - Doesn't solve the problem of servers gaining new roles and reconfiguring.

  - Let the head server be the coordinator. And can maintain a backup of the coordinator in the next server.

  - Fault handling

    - Each server monitors adjacent server (but could have a list and could iterate through)

  - Elect one of the server to be the coordinator node

    - Paxos (quorum protocol for consensus on who the coordinator is)

  - Build a DHT! on top of the server