# Practice questions

416 2016 W2 (Winter 2017)

# PQ 1

- Assume you were designing a protocol between two hosts that communicate over links that guarantee 0 packet loss.

  Would you still need to implement reliability measures like retransmission? Why or why not?

## PQ 1

- Assume you were designing a protocol between two hosts that communicate over links that guarantee 0 packet loss.

  Would you still need to implement/consider reliability measures like retransmission? Why or why not?

  **Yes. Routers could drop packets, as can the end-host software (OS). i.e., Links are just one part of a much larger picture.**

## PQ 2

- You are designing the printing service on campus with physical “printing cards” given to students. You can choose to store the printing balance information **in the cloud**, in **on-campus servers**, or **on the card**.

  To have a higher degree of fate sharing in your system, where should you store the printing balance?

## PQ 2

- You are designing the printing service on campus with physical “printing cards” given to students. You can choose to store the printing balance information **in the cloud**, in **on-campus servers**, or **on the card**.

  To have a higher degree of fate sharing in your system, where should you store the printing balance?

  **On the card. Lose state for ability to print if and only if the card (that is necessary to print) is lost.**

# PQ 2

- You are designing the printing service on campus with physical “printing cards” given to students. You can choose to store the printing balance information **in the cloud**, in **on-campus servers**, or **on the card**.

  To have a higher degree of fate sharing in your system, where should you store the printing balance?

  **On the card. Lose state for ability to print if and only if the card (that is necessary to print) is lost.**

  **(What if you stored the balance on each printer?)**

# PQ 3

- You are designing an IoT lock device that receives commands from an iPhone via the cloud. You are considering different semantics for the command that actuates the device to lock the door. Which semantics is the most appropriate?
    - At least once
    - At most once
    - Zero or once

# PQ 3

- You are designing an IoT lock device that receives commands from an iPhone via the cloud. You are considering different semantics for the command that actuates the device to lock the door. Which semantics is the most appropriate?
  - **At least once [lock means lock + idempotent]**
  - At most once
  - Zero or once

## PQ 4

- Traditional NFS has a stateless server, allowing it to reboot without impacting existing client connections (much). What does NFS trade-off to gain this stateless server advantage? i.e., where does it lose in return?

# PQ 4

- Traditional NFS has a stateless server, allowing it to reboot without impacting existing client connections (much). What does NFS trade-off to gain this stateless server advantage?

  **1. Client have to maintain more state (more memory usage at client)**

  **2. Clients must continuously pass the necessary client operation state to the server (less bandwidth-efficient)**

# PQ 5

- Consider the assignment 1 goldilocks fortune protocol. In this protocol the server, disregarding the right-number/fortune, is stateless, but the client is stateful.
    - True
    - False

# PQ 5

- Consider the assignment 1 goldilocks fortune protocol. In this protocol the server, disregarding the right-number/fortune, is stateless, but the client is stateful.

  - **True: server replies with high/low while client maintain protocol state (id)**

  - False

# PQ 6

- In the goldilocks fortune protocol a client restart loses all of its “work” (explored id space). How would you (minimally) change this system so that a client restart does not lose the client’s work?

- Hint: introduce server state and change the protocol

# PQ 6

- In the goldilocks fortune protocol a client restart loses all of its “work” (explored id space). How would you (minimally) change this system so that a client restart does not lose the client’s work?
- **(1) Introduce server state (closest id to right number used by client) and (2) change the protocol (attach the closest id to each reply from server, e.g., “high, 42”)**
- **More efficient: attach closest id to reply if have not heard from client for a while**
- **More complex: add handshake protocol for client to determine current closest id**

# PQ 7

- Assume a case in which there are no failures and there is a single process in the system that is accessing a file for the first time that is hosted by a distributed file server. In this case both AFS and NFS provide identical semantics to this process.
    - True
    - False

# PQ 7

- Assume a case in which there are no failures and there is a single process in the system that is accessing a file for the first time that is hosted by a distributed file server. In this case both AFS and NFS provide identical semantics to this process.

  - **True: can’t tell difference between AFS/NFS/local FS if no failures and no concurrent clients (except performance).**

  - False

# PQ 8

- You are running a single process on a client machine that is using a distributed file system. You see the following sequence of operations:

```
fd=open('foo')           // open file 'foo'
lseek(fd, 0)             // goto position 0 in file
read(fd, buf, 1024)      // buf contains 'hello'
lseek(fd, 0)             // goto position 0 in file
read(fd, buf, 1024)      // buf contains 'refrigerator'
close(fd)                // close file
```

Based on above could the client be using NFS or AFS?

- **NFS**
- **AFS**
- **Either**
- **Neither**

# PQ 8

- You are running a single process on a client machine that is using a distributed file system. You see the following sequence of operations:

```
fd=open('foo')          // open file 'foo'
lseek(fd, 0)            // goto position 0 in file
read(fd, buf, 1024)     // buf contains 'hello'
lseek(fd, 0)            // goto position 0 in file
read(fd, buf, 1024)     // buf contains 'refrigerator'
close(fd)               // close file
```

Based on above could the client be using NFS or AFS?

- **NFS (impossible with AFS — it provides session consistency)**

# PQ 9

- A mutual exclusion lock (i.e., mutex) is a form of pessimistic concurrency control mechanism
    - True
    - False

# PQ 9

- Using a mutual exclusion mutex (i.e., lock) is a form of pessimistic concurrency control
  - **True: you use locking under the assumption that there are other concurrent, competing, clients who might execute the critical section**
  - False

## PQ 10

- Is a *lease* a performance-improving mechanism?
  - If yes, how does it improve performance?
  - If no, how does it degrade performance?

# PQ 10

- Is a *lease* a performance-improving mechanism? If so, how does it improve performance?

  - **Yes. A lease improves performance by removing the need for coordination — it allows the lease-holder to proceed with an operation unilaterally.**

  - Note: answer partly depends on how you define performance (e.g., No if you consider AFS with one client — a lease would actually degrade (server) performance, though just slightly).

# PQ 11

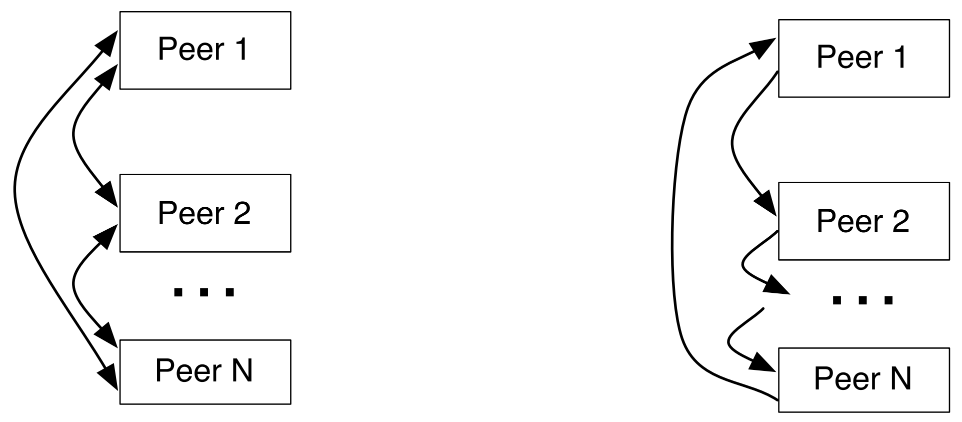
- Assume that the peers in assignment 2 had synchronized clocks. Would this help in maintaining constraint 2 (two consecutive GetResource invocations cannot come from the same peer)?
  - If yes, then state how it would help.
  - If no, then state why it would not help.

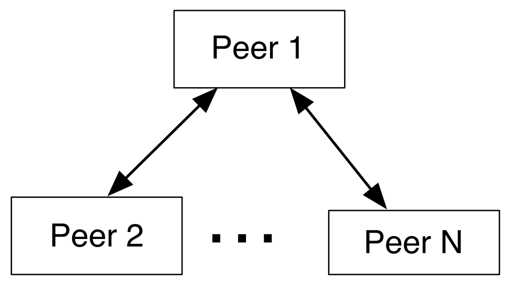
# PQ 11

- Assume that the peers in assignment 2 had synchronized clocks. Would this help in maintaining constraint 2 (two consecutive GetResource invocations cannot come from the same peer)?
  - **In general No, b/c operating in an asynchronous network and network latency is unbounded.**
  - **i.e., even if peers synchronized when to invoke the RPC, no guarantee that their invocations would reach the server in the right order.**

# PQ 12

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





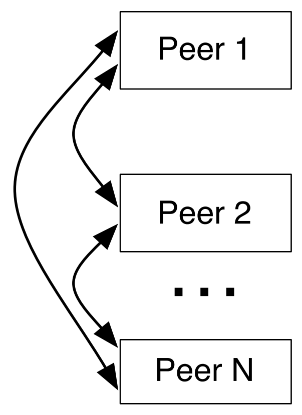
(a) all-to-all (b) linked-list (c) star

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

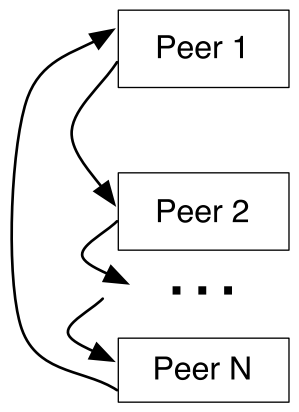
- Assuming a large N, which topology (on average) impacts the fewest peers when a peer fails?

# PQ 12

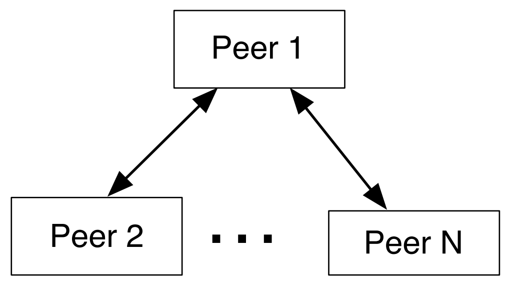
- Consider the following three topologies (e.g., in A3):



(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 13

- You are building a distributed system and are using logical clocks. You find that two events, *a* and *b*, are ordered according to your logical clock mechanism as L(a) > L(b). From this you deduce that *b* happened before *a.* Which logical clock mechanism are you using?

  A. Lamport cocks

  B. Vector clocks

  C. NTP

  D. This is all very confusing, I can't tell

# PQ 13

- You are building a distributed system and are using logical clocks. You find that two events, *a* and *b*, are ordered according to your logical clock mechanism as L(a) > L(b). From this you deduce that *b* happened before *a.* Which logical clock mechanism are you using?

  A. Lamport cocks

  **B. Vector clocks — remember the 'if and only if' holds for VClocks but not for Lamport clocks**

  C. NTP

  D. This is all very confusing, I can't tell

# PQ 14

- A node failure in the classic Ricart-Agrawala algorithm causes a deadlock (other nodes to block indefinitely)
    - True
    - False

# PQ 14

- A node failure in the classic Ricart-Agrawala algorithm causes a deadlock (other nodes to block indefinitely)
  - **True — no built-in fault tolerance**
  - False

# PQ 15

- You decide to optimize the vector clock algorithm by using a single index for two machines, A and B (and not changing anything else). Given vclock timestamps for events a at A, b at B, c at C, and d at D. Which of these statements can you deduce?

  A. a happened before b

  B. b happened before c

  C. a happened before c

  D. c happened before d

# PQ 15

- You decide to optimize the vector clock algorithm by using a single index for two machines, A and B (and not changing anything else). Given vclock timestamps for events a at A, b at B, c at C, and d at D. Which of these statements can you deduce?

  A. a happened before b (nope, not with virtual Lamport clock)

  B. b happened before c (nope; not without coordination btw A and B)

  C. a happened before c (nope; by symmetry with B)

  **D. c happened before d**

# PQ 16

- You decide to use a join design in A3 in which joining node (1) learns from the node it is joining through of other nodes, (2) advertises itself to these nodes, (3) waits to receive acknowledgements from all nodes, and then (4) calls JoinPrint.
- What are the issues with using this algorithm, and how would you fix them?

# PQ 16

- You decide to use a join design in A3 in which joining node (1) learns from the node it is joining through of other nodes, (2) advertises itself to these nodes, (3) waits to receive acknowledgements from all nodes, and then (4) calls JoinPrint.

- One of other nodes could join; ack never arrives

- Another node joins simultaneously; joining nodes do not learn about each other (race condition)

- Alg does not mention resource transfer, if all other nodes fail after it joins, it needs to remember past resources

# PQ 17

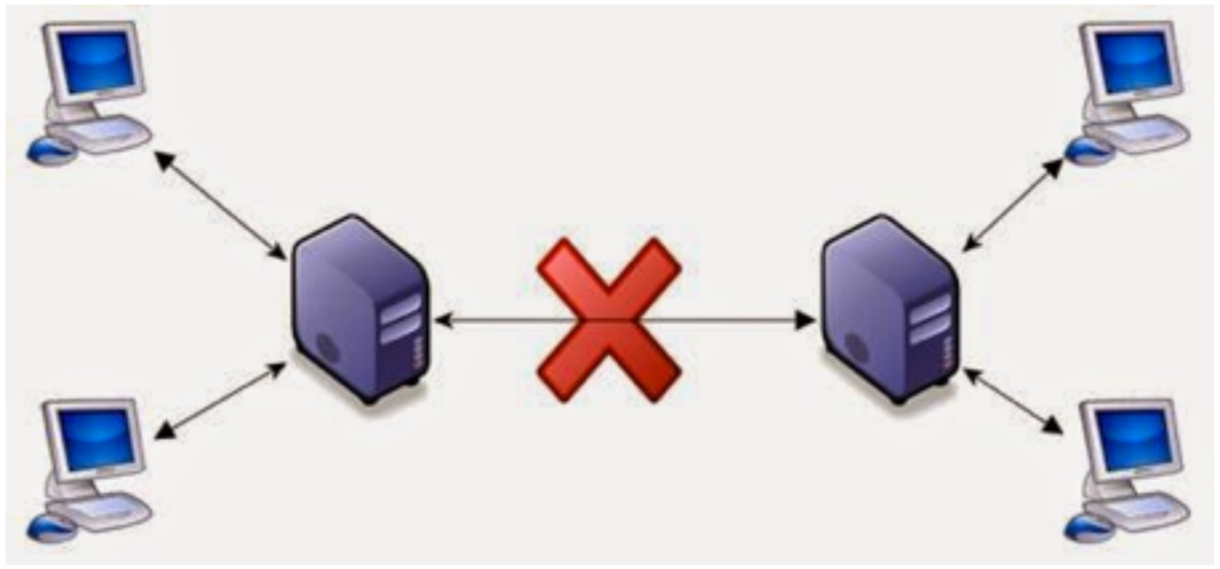
- If you are running on an unreliable network and you cannot reach a node using RPC then the node has failed.
    - True
    - False

# 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.**

**(In A3, we won't make the network more unreliable, e.g., by dropping packets)**

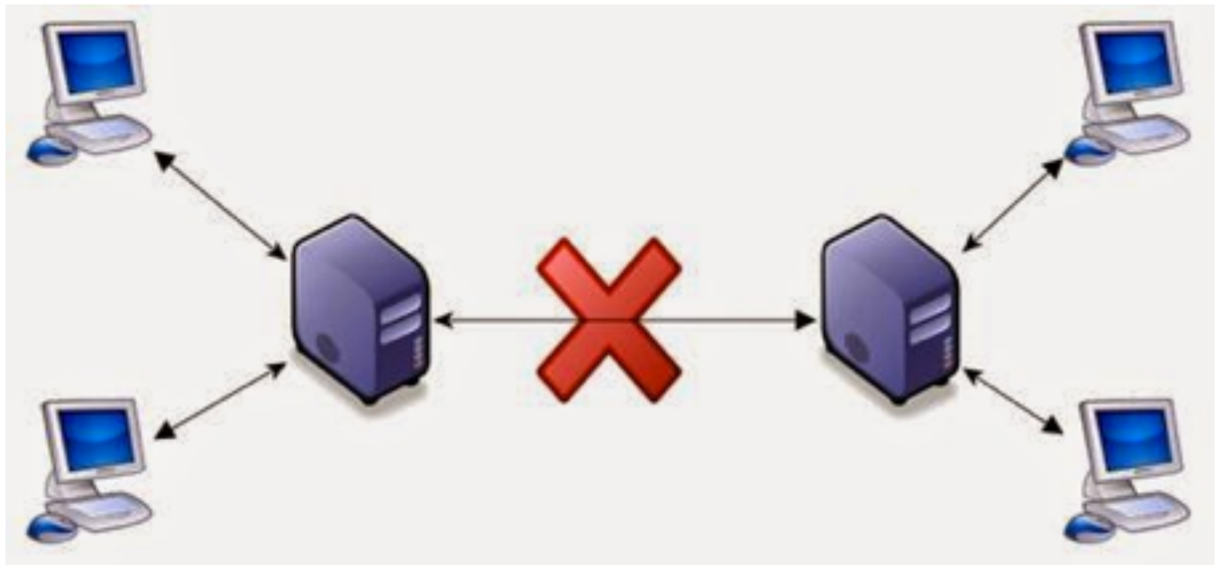
## PQ 18



- You are using Ricart-Agrawala on a network that might experience partition failures. You decide that you want to prioritize safety over all else (mutual exclusion should be guaranteed at all times).

- With above as a hard design constraint; when your system experiences a network partitioning, what should the set of nodes in a partition that is not currently holding the mutex do?

## PQ 18



- You are using Ricart-Agrawala on a network that might experience partition failures. You decide that you want to prioritize safety over all else (mutual exclusion should be guaranteed at all times).
- With above as a hard design constraint; when your system experiences a network partitioning, what should the set of nodes in a partition that is not currently holding the mutex do?
- **Fail, or wait.**

# PQ 19

- The most space efficient RAID level is RAID-1
    - True
    - False

## PQ 19

- The most space efficient RAID level is RAID-1
  - True
  - **False (RAID-0 is more space efficient)**

# PQ 20

- RAID provides partition tolerance
  - True
  - False

# PQ 20

- RAID provides partition tolerance
  - True
  - **False (fault model assume disk fail-stop, with partitions have to reason about writes; RAID doesn't do that)**

# PQ 21

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

# PQ 21

- Event A with 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 22

- A content distribution network provides a way for content providers to shed load from their servers
  - True
  - False

# PQ 22

- A content distribution network provides a way for content providers to shed load from their servers
  - True [A CDN is fancy cache]
  - False

## PQ 23

- You are considering a startup that will use a peer-to-peer architecture. Your co-founder is a psychology major. Explain (all the possible) **advantages** of a peer-to-peer design to your business partner.

## PQ 23

- Elastic: scales with demand (peers bring resources to support more peers)
- Cheap: resources are not centrally provided/ administered
- Peer diversity = higher resilience to failures (e.g., geographical/network diversity)
- No single point of failure = more fault tolerant
- Resilient to gov. oversight/control

## PQ 24

- For A5 you decide to implement messaging between workers by routing all messages through the server. What are the advantages and disadvantages of this design decision?

# PQ 24

- Advantages:
  - Simple: centralized comm. between all workers (e.g., debugging is easy)
  - Single point of control/policy enforcement — (e.g., preventing some workers from communicating is easy)
  - Fate sharing of outstanding client request and msgs to get it processed
  - Short worker unavailability can be hidden by the server (appears as delay to other worker)
  - Can use worker names independent of physical identifiers (IPs/ports); e.g., allows workers to be mobile
- Disadvantages:
  - Server load: bottleneck
  - Extra latency in communication (particularly bad if worker in another data center!)

# PQ 25

- RAID uses complement sum for error detection

  A. Yes

  B. No

## PQ 25

- RAID uses complement sum for error detection

A. Yes

B. No [RAID uses Parity]

## PQ 26

- In A4/A5 workers do not fail. What if workers could fail in a fail-stop manner. How would you re-design your A5 system to provide identical guarantees to clients even if workers fail?

# PQ 26

- Need a failure detector, either at server or workers.
- Need to store crawled worker state — can replicate to other works, but they can fail, so have to continuously replicate/migrate state. Or, store it at server. Crawled state = URLs crawled by worker (assuming can re-crawl to get exact state lost)
- On failure, notify workers if they are coupled (e.g., direct comm link). If not, then decouple further and delay requests to failed worker until new worker catches up.
- Determine workers who will take over domains of failed worker — trigger latency compute to URLs, aggregate at server, farm out URLs to crawl to the right workers.
- Determine if existing client operation is impacted. If so, cancel operation at other workers (e.g., overlap) if coupling between workers; restart operation (blocking client does not observe this). If no impact, continue operation.

## PQ 27

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

  A. NFS

  B. AFS

## PQ 27

- 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 28

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

# PQ 28

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

## PQ 29

- In A5 a single client makes blocking requests. How would you redesign your A5 to support an arbitrary number of clients who make blocking requests? (Without changing the client-server API)

# PQ 29

- Non-conflicting operations can run concurrently. But, conflicting operations require concurrency control.
- We also need to extend semantics of API for concurrent ops — e.g., serializable semantics?
- Conflict: crawl(u1), crawl(u2) where u1, u2 in same domain
- Conflict: crawl(u1), overlap(u2,u3,) where u1, u2 in same domain
- Conflict: crawl(u1,d1), crawl(u2,d2) where u1 and some crawl depth < d1 reaches a domain that u2 reaches at crawl depth < d2
- Optimistic: Locking! (key question: what granularity?)
- Pessimistic: Run first, check for conflicts later (possibly recompute)

## PQ 30

- You plan to build a CDN based on Akamai that not only caches static content, but also caches dynamically-generated results. Sketch out a high-level design for this kind of CDN. (Hint: what properties must this kind of CDN provide?)

# PQ 30

- Basic design:
    - Server S computing the dynamically-generated content embeds a special hash H along with the Akamai link to the content
        - H is a pointer to state necessary to generate the content, this state can be maintained at S until some timeout. Assumption: given two requests, if they resolve to the same hash H, then the dynamic content response is identical.
    - Client requests and downloads index.html containing Akamai links. Client resolves Akamai links to the Akamai servers in the usual way.
    - Akamai server A sees the hash H, and first determines if the (dynamic) content corresponding to H is in its cache.
        - If content for H is in cache, A checks if this content has expired. If not expired then return the content to client.
        - If content is not in the cache, contact S, sending along H, and receive the generated content. S will also send along an expiration TTL for the dynamically generated content in its reply. Cache this content, then reply to client.

# PQ 31

- Transactions in A6 mix put and get operations. Imagine a different API that allowed transactions containing either puts or gets, but never both. How and why would this change simplify your system?

# PQ 31

- Conflicts between operations more structured: get TXs never conflict, put TXs only conflict when overlap on keys.

- Knowledge of keys involved in put TXs can be aggregated before the puts are executed — can lock in batch at commit time.

- Can retry put TXs! Application does not see intermediate failure of a put TXs (but it does see intermediate values for a get TXs, making those more complex from a concurrency control pov)

# PQ 32

- In A6, supporting nodes that restart is more complicated than supporting nodes that fail and never come back
    - True
    - False

# PQ 32

- In A6, supporting nodes that restart is more complicated than supporting nodes that fail and never come back
  - True: not knowing when a node restarts makes the problem as hard as nodes that never come back. Nodes that do come back require extra logic/coordination. If you assume no durable state, then this is identical to supporting node joins!
  - False

# PQ 33

- Cars are transactions and intersections are shared data. In this context, traffic lights (red/yellow/green) are a form of optimistic concurrency control.
  - True
  - False

# PQ 33

- Cars are transactions and intersections are shared data. In this context, traffic lights (red/yellow/green) are a form of optimistic concurrency control.
    - True
    - False. Analogy is a stretch (cars can't change traffic lights), but it's certainly exclusionary/preventative measure. Thus, pessimistic, not optimistic.