Algorithms and Distributed Systems 2019/2020 (Lecture Nine)

MIEI - Integrated Master in Computer Science and Informatics

Specialization block

João Leitão (jc.leitao@fct.unl.pt)

Lecture structure:

- CAP Theorem
- Causal Consistency
- ChainReplication

Previous Class: Weak Consistency

- In particular Eventual Consistency
  - Debatable if it conforms to the specification of a consistency model (imposes no restrictions to the observable state of the system by clients).

- Benefits of Eventual Consistency
  - Allows for faster response times for clients (by avoiding server replicas to coordinate with each client operation)
  - Improves availability (broader conditions in which the system is able to process client requests)

The Dark Side of Eventual Consistency

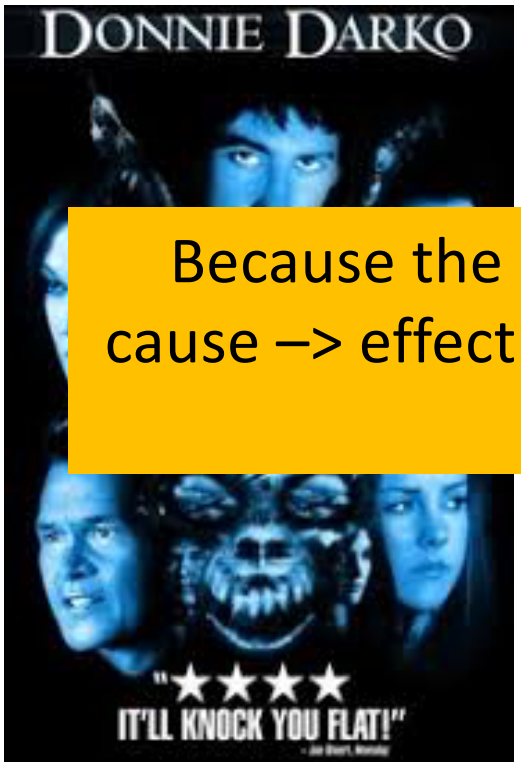
- The behavior of the system might be hard to grasp by users/developers.
- In particular, a sequence of reads executed by a client might exhibit state that is not consistent with the evolution of time by the user.
- Why is this bad?

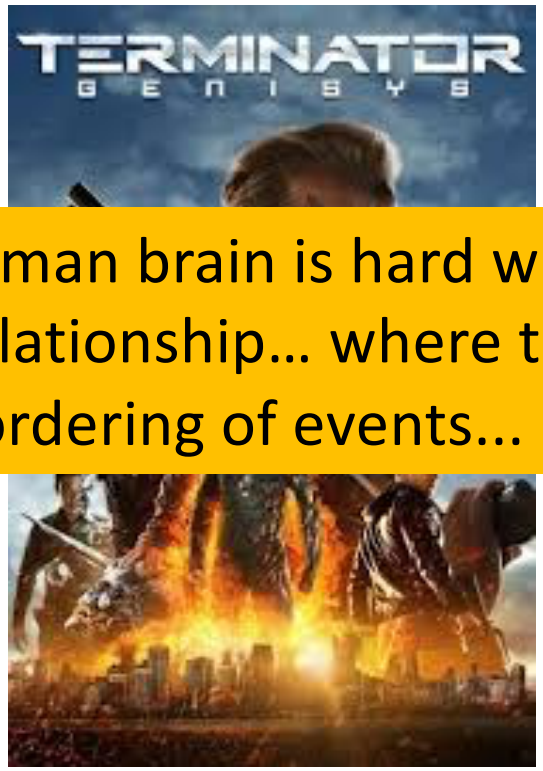
Time should follow in a single direction…

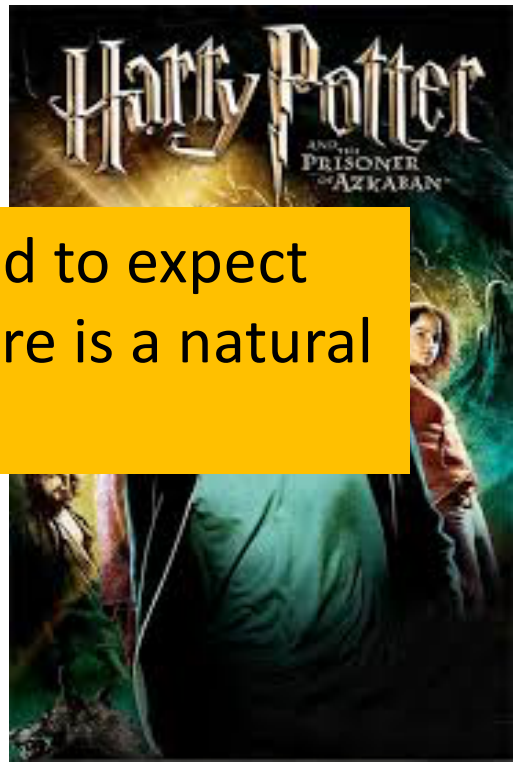
- Why is it that many time travel movies are hard to follow by the audiences?

Time should follow in a single direction…

- Why is it that many time travel movies are hard to follow by the audiences?



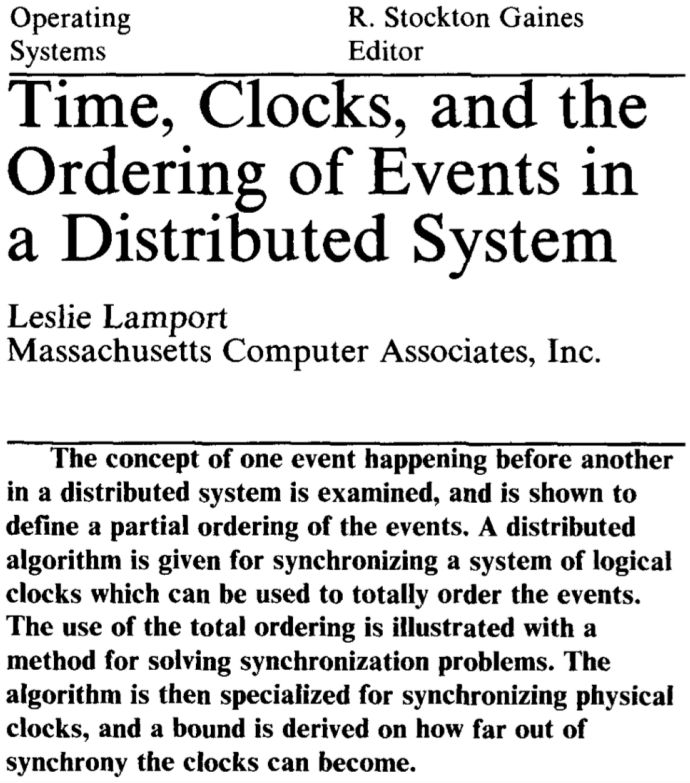




Because the human brain is hard wired to expect cause –> effect relationship… where there is a natural ordering of events...

The happens before relantionship… (in systems)

Lamport, 1978



Operating Systems — R. Stockton Gaines Editor

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events. The use of the total ordering is illustrated with a method for solving synchronization problems. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchrony the clocks can become.

- A common restriction of many consistency systems is related with ensuring that clients observe the state of the system evolving according to some order that respects the happens-before relationship.
- Unfortunately, this is not a requirement in eventual consistency (the weakest of the weak consistency models).

In practice why can this be bad?

In practice why can this be bad?

- An example close to your daily life: Social Networks

- Imagine a system like Facebook.

In practice why can this be bad?

- An example close to your daily life: Social Networks

- Now imagine that there are two relevant data objects: a) your posts; b) access list to your posts.

Posts: <Lots of innocent stuff>

Access list: <Lots of people>

In practice why can this be bad?

- An example close to your daily life: Social Networks

- Further imagine that your professor is currently allowed to see your posts (i.e., is in the access list).

Posts: <Lots of innocent stuff>

Access list: <Lots of people> <João Leitão>

In practice why can this be bad?

- An example close to your daily life: Social Networks

You went to a party in the night before the class, where imagine, you had to deliver a homework…

You are in no way ready to go to the class, but since you don't want to tell the truth about why you cannot go, you send an e-mail claiming that you are sick... However... You also wanna post that "sick" picture of the party last night in your facebook...

After sending the e-mail what do you do?

Access list: <Lots of people> <João Leitão>

In practice why can this be bad?

- An example close to your daily life: Social Networks
- First you remove the professor from the access list.

Posts: <Lots of innocent stuff>

Access list: <Lots of people> <João Leitão>

In practice why can this be bad?

- An example close to your daily life: Social Networks
- Then you post that amazing picture from last night.

Posts: <Lots of innocent stuff>

Access list: <Lots of people> <João Leitão>

In practice why can this be bad?

- An example close to your daily life: Social Networks
- And you are safe right?

Posts: <Lots of innocent stuff>

Access list: <Lots of people> <João Leitão>

In practice why can this be bad?

- An example close to your daily life: Social Networks

- Well not exactly… if Facebook is providing only eventual consistency...

Posts: <Lots of innocent stuff>

Access list: <Lots of people> <João Leitão>

In practice why can this be bad?

- An example close to your daily life: Social Networks

- How does the (application server) creates the Wall of a user?

Posts: <Lots of innocent stuff>

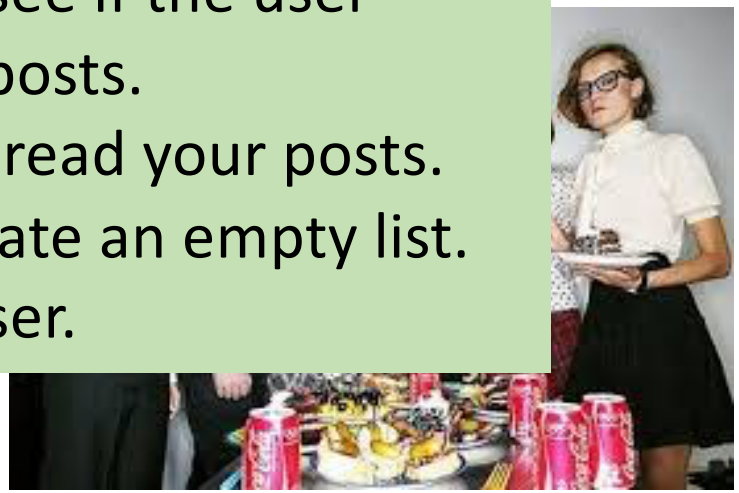
Access list: <Lots of people> <João Leitão>

In practice why can this be bad?

- An ex
- How

When the professor client application issues a read for your posts list, the application server will execute the following steps:

1. Check your access list to see if the user (professor) can see your posts.
2. If the answer is yes, then read your posts.
3. If the answer is no, generate an empty list.
4. Return the reply to the user.



Access list: <Lots of people> <João Leitão>

In practice why can this be bad?

- An ex
- How

of a u

Unfortunately this can go wrong…

Upon checking the access list the application server is allowed to see an out-dated version of the list that still has ”target” user in the list.

The following read over the post list however, can return the most recent version containing ”that” picture.

Your intentions were defeated by eventual consistency.

So what can we do to overcome this sort of scenario...

- A simple strategy is to enforce the happens before relationship in the state that is observed by the two read operations (access list and posts).

A simple way to do this:

- Scrap this notion of using weak consistency and rely instead in some form of strong consistency.
- Strong consistency is the best ☺
- Wait, why did we leave strong consistency?

A simple way to do this:

- Scrap this notion of using weak consistency and rely instead in some form of strong consistency.
- Strong consistency is the best ☺
- Wait, why did we leave strong consistency?
  - Strong consistency depends on total ordering of all client operations:
    - Increased response times (high latency)
    - Operations cannot happen if a majority of replicas is not reachable.

A simple way to do this:

- Scrap this notion of using weak consistency and rely instead in some form of strong consistency.

Clients can wait… (actually they don't...)

Majority of replicas will always be available...

What are the odds really?

operations:

- Increased response times (high latency)
- Operations cannot happen if a majority of replicas is not reachable.

A simple way to do this:

- Scrap this notion of using weak consistency and rely instead in some form of strong consistency.

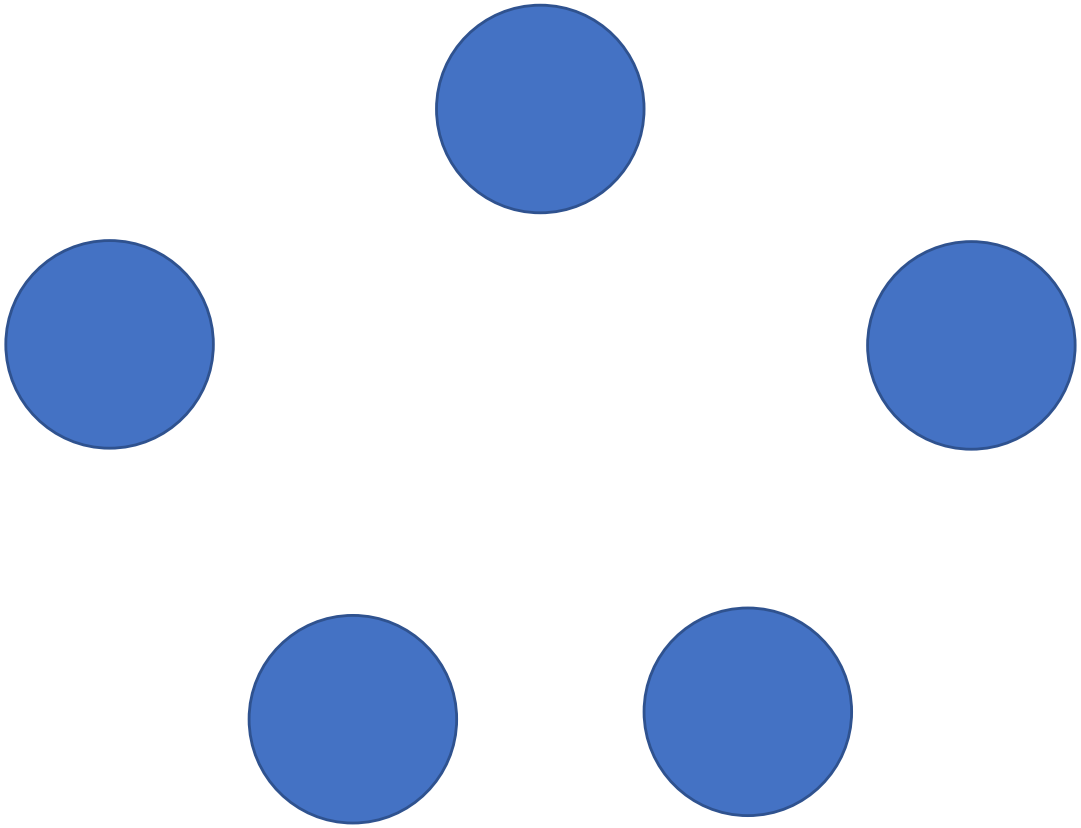
Clients can wait… (actually they don't...)

Majority of replicas will always be available...
What are the odds really?

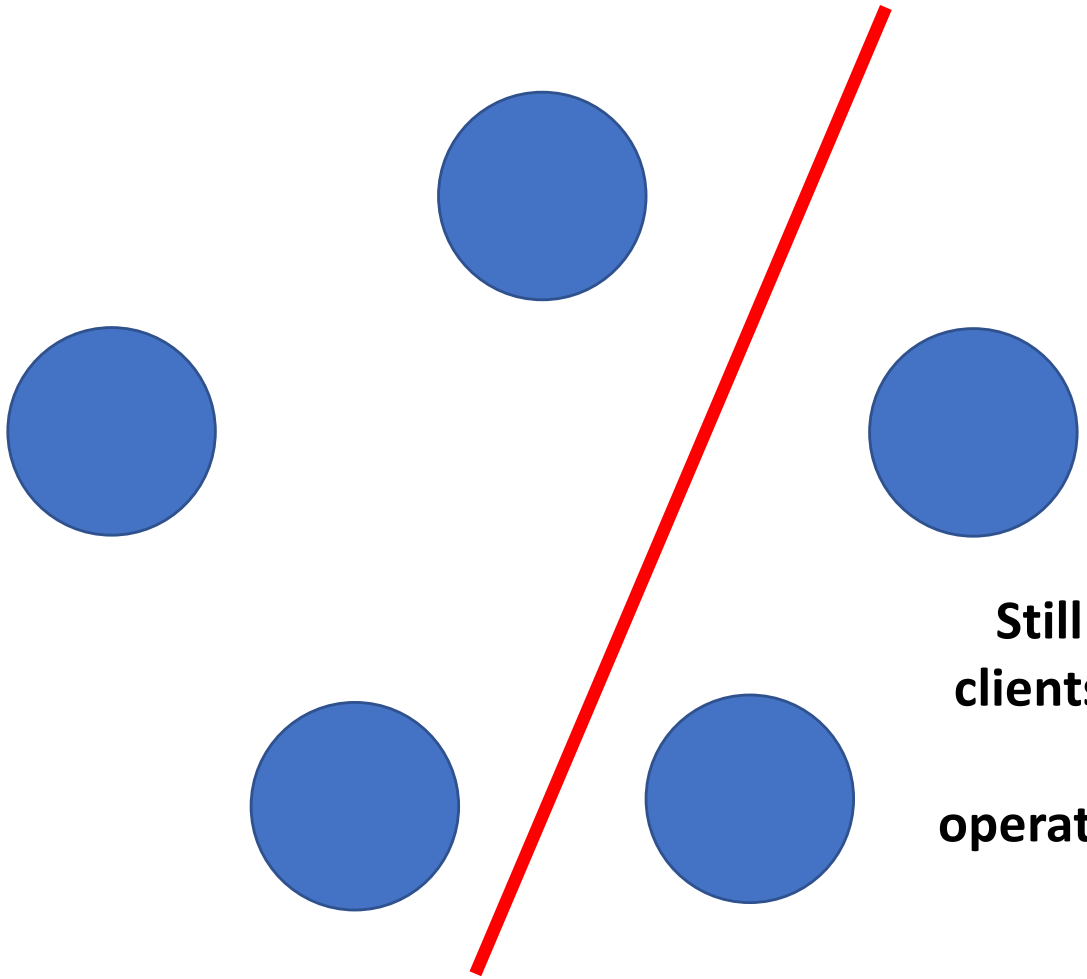
operations:

- Increased response times (high latency)
- Operations cannot happen if a majority of replicas is not reachable.

The network is not your friend:
Network Partitions

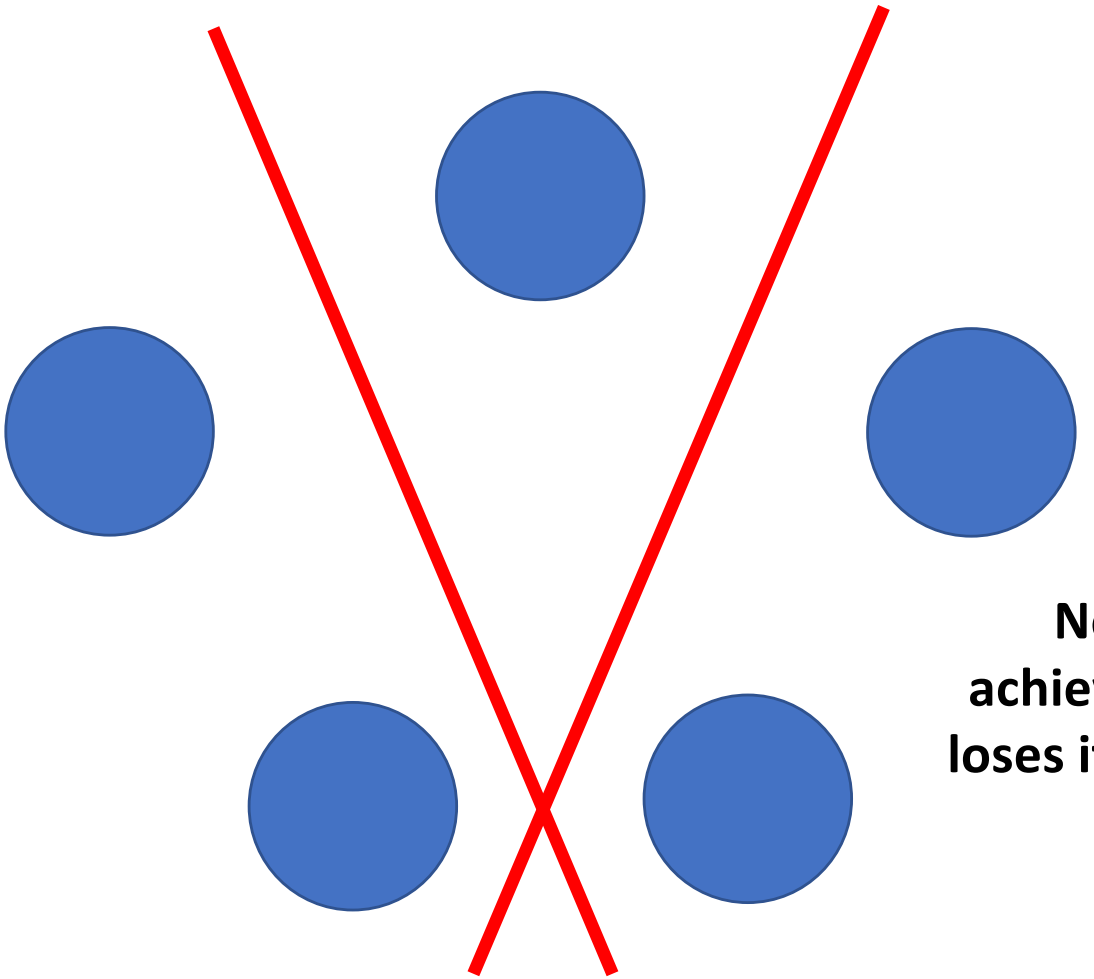


The network is not your friend:
Network Partitions



Still have a majority… clients on the right side cannot complete operation that require a majority

The network is not your friend:
Network Partitions



No majority can be achieved... The system loses its availability (for every client)

The network is not your friend: Network Partitions

- Network partitions can happen at any time due to multiple effects:
  - Hardware failure (router/switches)
  - IP Routing instability (see for instance the stabilization times of algorithms such as BGP and OSPF)
  - Poorly configured Firewalls.
  - Political decisions (blocking traffic from given sources, see for instance the great firewall of China, or laws in Russia)
- Network partitions are more likely in scenarios where replicas are scattered across long distances (E.g., Geo-Replication).

What can we do?

- We are looking for a solution that:
  - Is available: Allows clients to complete operations despite failures and network partitions.
  - Is strongly consistent: Creates a total order among, at least, the operations of clients that modify the state of the system.

- Any Ideas?

What can we do?

- We are looking for a solution that:
  - Is available: Allows clients to complete operations despite failures and network partitions.
  - Is strongly consistent: Creates a total order among, at least, the operations of clients that modify the state of the system.

- Any Ideas?
- Well… such solution technically does not exists!

CAP Theorem

- Introduced by Eric Brewer in 1999 as a principle.
- In a distributed system it is impossible to simultaneously provide:
  - Strong **C**onsistency (as in atomicity, every client will observe the effects of the most recent write).
  - **A**vailability (all operations finish eventually).
  - **P**artition-Tolerance (network anomalies that fraction the system into multiple components, where only the communication links inside each component are reliable).

CAP Formal Proof

CAP Formal Proof

- By contradiction, assume there is a distributed algorithm that provides C, A, P.
- Assume there is a read/write register, with initial value v_0, and two processes p1 and p2 that are partitioned (all messages between p1 and p2 are lost)
- Execution e1:
  - Process p1 executes write(v), to ensure A this operation concludes at some point in time t1.
  - Process p2 executes read() at time t2=t1+k. By the C property, A must return value v.
- Execution e2:
  - Process p1 does not executes any write.
  - Process p2 executes a read() operation at time t2=t1+k. By property A it has to return some value, and by property C that value must be v_0.
- **From the stand point of p2, executions e1 and e2 are indistinguishable, and hence shuld return the same value.**

So can we do better than eventual consistency?

- While being within the bounds set by the CAP theorem.

So can we do better than eventual consistency?

- While being within the bounds set by the CAP theorem.

- Yes we can: We can rely on Causal Consistency.

Causal Consistency

- It is a consistency model that enforces that each client always observes a system state that respects the cause -> effect relationships between write operations.
- Typically described by the combination of four fundamental properties:
  - Read your writes
  - Monotonic Reads
  - Monotonic Writes
  - Writes Follows Reads

Read your Writes

- A client must always be able to observe the effects of all previous writes issued (and for which it got a reply from the system) by itself.

- Observing the effects does not imply that the same value must be returned. It implies that the returned value is at least as up-to-date as the value written by the client itself.

Monotonic Reads

- Subsequent reads issued by the same client should observe either the same state or an inflation of the system state (i.e., the state observed previously modified by new write operations).

- This property ensures that the system state does not “evolve into the past”.

Monotonic Writes

- The effects of multiple write operations issued by a given client, must be observed respecting that order by every other client.

- E.g.:
  - Assume that a client C writes 1 on object X and 2 on object Y (both objects initially had a value 0).
  - If another client observes Y = 0 and then X = 1, this does not violate Monotonic Writes.
  - If another client observes Y = 2 and then X = 0, this violates Monotonic Writes.

Writes Follows Reads

- If a client observes the effects of a write operation in its session (through a read operation), then any subsequent write operation issued by that client must be ordered after the previously observed write.

- This is also known as the transitivity rule.
  - If you think carefully up to now the properties previously discussed only refer to a single client session.
  - This is the property that establishes a connection between sessions of different clients.

Causality

- Causality by itself does not guarantees that the state of multiple replicas converges to a single value (even if at some point in time no more write operations happen).

- Divergence can last forever without breaking any causality property (provided clients are always connected to the same replica).

Causal+ Consistency Model

- It's a recently formalized consistency model that simply combines the properties of:
  - Causal consistency.
  - Eventual consistency.

- In short it ensures that the state of replicas eventually converges to a single state if no more write operations happen.

Causal+ and CAP

- The Good news is that up to now the Causal+ consistency model is the strongest of the weak consistency models that is compatible with the CAP theorem (i.e., does not falls into the not available solutions).
- There is a caveat however:
  - Clients must stick with a replica.
  - If a replica fails (permanently) then those clients might be forced to restart their sessions (which is a nice way to say sacrifice causal consistency guarantees).

Implementing Replication Protocols offering Causal+ Consistency

- Intuition:
  - There is a partial order among write operations defined by a combination of:
  - A) Order of write operations executed by a client:

  W(x,1) W(y,2) => W(x,1) $\prec$ W(y,2)

  - B) Effects of write operations observed by a client (in their read operations) before they issue a write.

  R(x,2) W(y,1) => W(x,2) $\prec$ W(y,1)

  If $W_1 \prec W_2$: We say W_1 is a causal dependency of W_2

Implementing Replication Protocols offering Causal+ Consistency

- Track causal dependencies of operations (potentially associate those to data objects written by each write operation).
- Ensure, for each client, that if a value generated by W_a is observed, no value can be observed that has been overwritten by any W_d such that W_d is a causal dependency of W_a.
- A typical way to do this is to only allow the effects of W_a to become visible after the effects of any operations that are overwritten by any W_d becoming impossible to observe.

Implementing Replication Protocols offering Causal+ Consistency

- This requires clients to memorize all write operations whose effects have been observed in their session (which includes not only the write operations directly observed but all of their causal dependencies): The **client causal history**.
- When performing a write operation, the causal history of the client is sent alongside the write and recorded in the data storage system.
- Dependencies of operations might grow quite significantly.

Famous Systems providing Causal+ Consistency

To appear in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*

Don't Settle for Eventual:
Scalable Causal Consistency for Wide-Area Storage with COPS

Wyatt Lloyd*, Michael J. Freedman*, Michael Kaminsky†, and David G. Andersen‡
*Princeton University, †Intel Labs, ‡Carnegie Mellon University

- Coined the term "Causal+ Consistency" (but did not invent it).
- Tracks dependencies of operations (using a graph).
- Sometimes operations have to be retried.
- Supports special read only transactions.

Famous Systems providing Causal+ Consistency

ChainReaction: a Causal+ Consistent Datastore based on Chain Replication

Sérgio Almeida
INESC-ID, Instituto Superior Técnico, U. Técnica de Lisboa
sergiogarrau@gsd.inesc-id.pt

João Leitão
CITI / DI-FCT-Universidade Nova de Lisboa *and* INESC-ID, IST, UTL
jc.leitao@fct.unl.pt

Luís Rodrigues
INESC-ID, Instituto Superior Técnico, U. Técnica de Lisboa
ler@ist.utl.pt

- Uses a variation of Chain Replication (originally provides linearizability) to provide causal guarantees.
- Delays write operations to minimize metadata overhead (i.e., dependency tracking).
- Operations never need to be retried.
- Also supports special read only transactions.

Famous Systems providing Causal+ Consistency

We will see this one in the labs today.

ChainReaction: a Causal+ Consistent Datastore based on Chain Replication

Sérgio Almeida
INESC-ID, Instituto Superior Técnico, U. Técnica de Lisboa
sergiogarrau@gsd.inesc-id.pt

João Leitão
CITI / DI-FCT-Universidade Nova de Lisboa *and* INESC-ID, IST, UTL
jc.leitao@fct.unl.pt

Luís Rodrigues
INESC-ID, Instituto Superior Técnico, U. Técnica de Lisboa
ler@ist.utl.pt

- Uses a variation of Chain Replication (originally provides linearizability) to provide causal guarantees.
- Delays write operations to minimize metadata overhead (i.e., dependency tracking).
- Operations never need to be retried.
- Also supports special read only transactions.

(Final) Homework 4:

- Consider the exemple of the social network with the access control list and the photo.

- If operations are isolated, and you start by updating the access control list and then post the photo you do not want to be seen, causality might not be enough to avoid the scenario discussed here.

- Explain why with an exemple and considering the properties of causal consistency
  (no need for pseudo-code).