

**when you share the Redis recorder lecture?** Go through them by next weekend after sql vs no sql class..

ashim again: where would we able to find the Redis recording lacture?

He sharded in whatsapp.. group..

1. Overview of Storage Layer - <https://www.scaler.com/meetings/redis-overview-2/j/a/cmtAL2JbLGgPdaowvMVhs4I9n5031S1n>

2. Redis : Deep Dive - <https://www.scaler.com/meetings/redis-deep-dive-4/j/a/Ayy8fo-Fwb__BAxTgEMle8X3mMyh7Iyi>

3. Redis : Deep Dive 2 - <https://www.scaler.com/meetings/redis-followup/j/a/w-pH_An40x9NAYCzlhTKOhG@eJfuM3EH>

Any updated on network lacture? (apology if you had answered earlier)

Cache coherence is a fundamental concept in computer architecture, particularly in systems with multiple processors or cores that share a common memory. It ensures that all processors in a system observe a consistent view of memory, even when each processor has its own cache.

In a multiprocessor system, each processor often has its own cache, which is a small, fast memory that stores copies of parts of the main memory. Caching improves performance by allowing processors to access frequently used data more quickly. However, it introduces the challenge of ensuring that all processors see a consistent view of memory, especially when multiple processors are reading and writing to the same memory locations.

Cache coherence protocols are employed to manage these challenges. The two main issues addressed by cache coherence are:

1. **Read Coherence:** Ensures that when one processor reads a memory location, it sees the most recent write to that location by any other processor.
2. **Write Coherence:** Ensures that when one processor writes to a memory location, the new value is properly communicated to and updated in the caches of all other processors.

There are several cache coherence protocols, and they can be broadly categorized into two types:

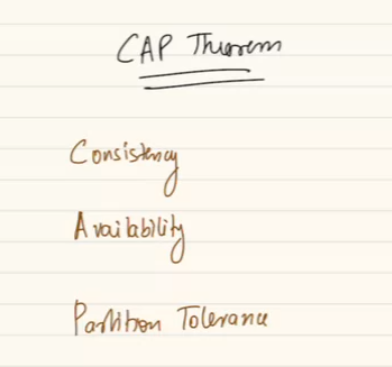
1. **Invalidation-Based Protocols:** Invalidation-based protocols work by invalidating (marking as invalid) the copies of a cached memory block in all other caches when one processor writes to that block. This ensures that only one processor has a valid copy of the data at a time, reducing the risk of inconsistent reads.
2. **Update-Based Protocols:** Update-based protocols involve updating the copies of a memory block in all caches when one processor writes to that block. This ensures that all processors have the most recent data, but it requires more communication overhead compared to invalidation-based protocols.

Examples of cache coherence protocols include MESI (Modified, Exclusive, Shared, Invalid) and MOESI (Modified, Owned, Exclusive, Shared, Invalid).

Cache coherence is essential for the correct and reliable operation of multiprocessor systems. Without proper cache coherence, programs running on different processors might observe inconsistent data, leading to unpredictable behavior and potential errors. Modern multiprocessor systems, including multicore processors, use sophisticated cache coherence mechanisms to maintain consistency and ensure correct execution of parallel programs.

**CAP Theorm:**

We heard consistency in ACID property in sql.



But C in ACID is different in C in CAP theorm.

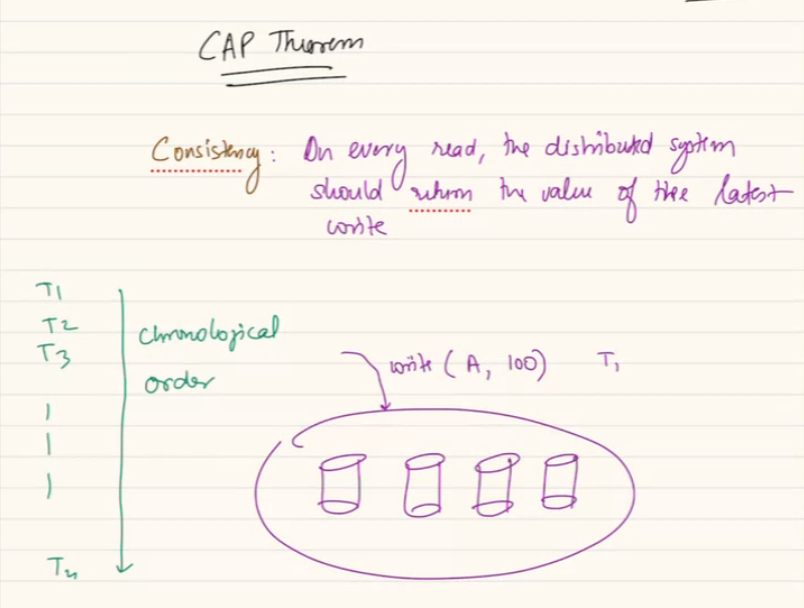
**Consistency:** means your distributed system is going to return on every read, the latest write. On every read the distributed system should return the value of the latest write.

A write cam eat time T1, plz write value of A as 100.

If at time T2, someone wants to read A, the answer should be 100. System should reply back as answer=100.

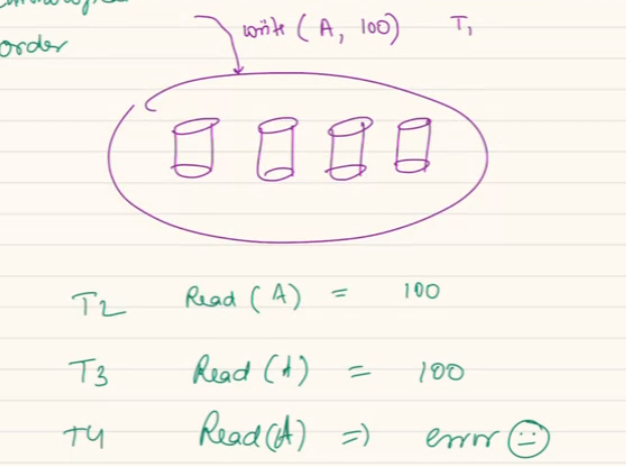
If at time T3 answer should be 100.

T1, T2, T3 are in chronological order.. T5 > T3.

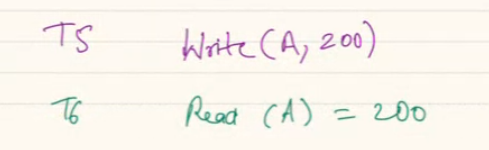


On every read the distributed system should return the value of the latest write Or error.

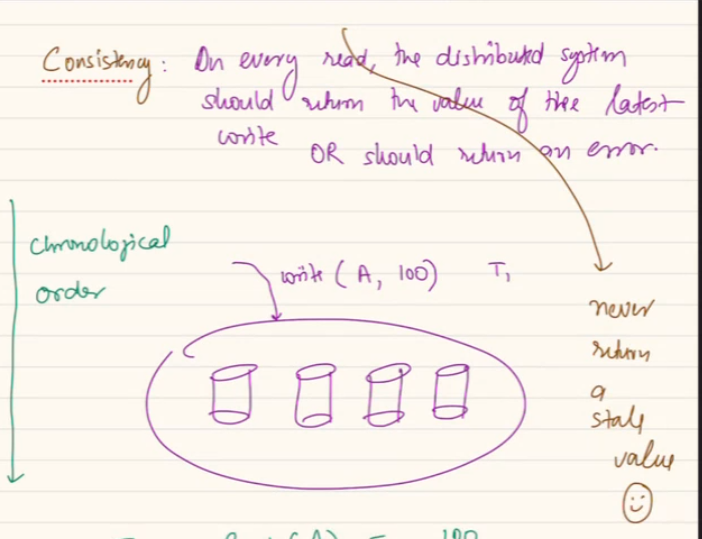
Meaning at Time T4 f someone try to read it can return error.



If at T5 someone write A = 200.. so after T5, at T6 people should get A = 200 (latest write) or error. Never returns a Stale value.



Other way: never return a Stale value.



If at T7 someone write B= 500,

T8 reads A = 200

T9 writes B = 700.,

T10 read either error or 700.

CAP theorem, also known as Brewer's theorem, is a concept in distributed systems that highlights the trade-offs between three desirable properties: Consistency, Availability, and Partition tolerance. These three properties are crucial in the design and operation of distributed databases and systems. Here's a brief overview of each:

1. **Consistency (C):** All nodes in the distributed system see the same data at the same time. In other words, a write operation on one node should be immediately visible to all other nodes.
2. **Availability (A):** Every request made to the distributed system receives a response, without guarantee that it contains the most recent version of the data. This means that even if some nodes in the system fail, the system should still be able to respond to read and write requests.
3. **Partition Tolerance (P):** The system continues to operate and provide both consistency and availability even in the presence of network partitions, meaning that communication failures can occur between nodes.

The CAP theorem posits that in a distributed system, it's impossible to simultaneously achieve all three of these properties. The theorem asserts that, in the face of a network partition, a distributed system must choose between maintaining consistency or availability.

If a network partition occurs (P), a system must decide whether to provide a consistent view of the data (C) to all nodes, even if it means sacrificing availability (not being able to respond to some requests), or to prioritize availability, even if it means sacrificing consistency (some nodes may have outdated data).

Different distributed systems make different trade-offs based on their requirements and use cases. For example:

* **CP Systems (Consistency and Partition Tolerance):** These systems prioritize consistency over availability. They ensure that all nodes have the same view of the data, even in the presence of network partitions. Examples include traditional relational databases.
* **AP Systems (Availability and Partition Tolerance):** These systems prioritize availability over strict consistency. They are designed to provide a response to every request, even if it means that some nodes may have slightly outdated data. NoSQL databases like Cassandra and Couchbase are examples of AP systems.
* **CA Systems (Consistency and Availability):** These systems sacrifice partition tolerance. They remain consistent and available as long as there are no network partitions. Examples of CA systems are single-node databases or systems that can rely on a highly reliable network.

Understanding the CAP theorem helps architects and developers make informed decisions about the design and trade-offs in distributed systems, especially when it comes to choosing the right database or storage solution for a particular use case.

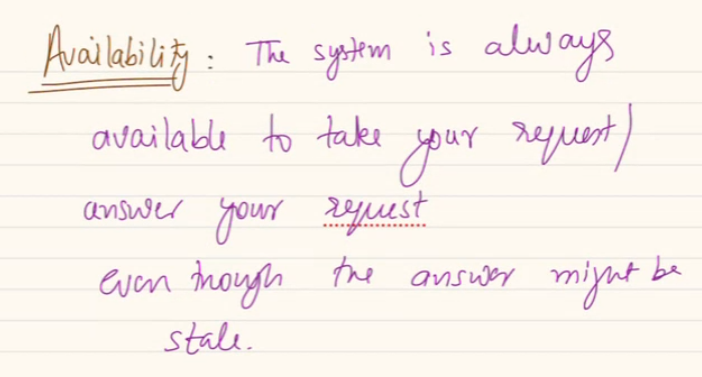
If we have defined certain invariance for my DB, the DB always going to remain True to those invariance after transaction.

If I have a bank 2 people A, B customer. A=100, B = 0 mone. Total money=100. A pays 50 to B. if bank has invariance of sum=100 untill outbound transaction.. The invariance are meet.

If my a column is primary key. Cannt be null, duplicate. B4 tranaaction there should be no entry of null or supplicate, also after transaction no row have these violation. This is means ACID ka consistency.

Consistency of CAP: when you read returns a latest write or error.

**Availability:** Whenever I got ot read or write to my system, system always going to take my request. 2 parts, the system will always available to take my request/ answer my request even though the answer might be Stale.



Answer might be Stale , old.

But in consistency we said never return a Stale. Availability is compromising on that. Lets understand it.

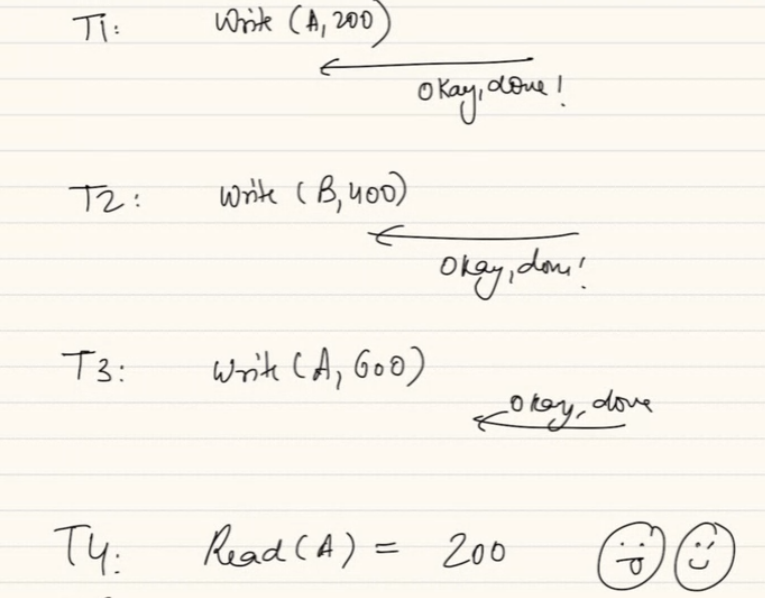
T1, someone write A = 200. If system is available it should return as Done.

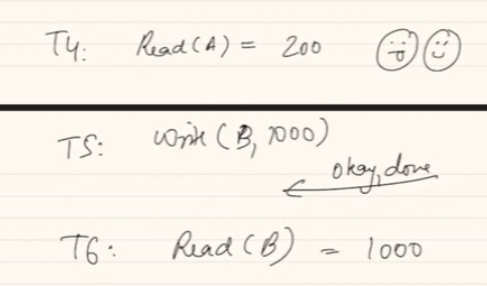
T2, someone write B = 400. Response should be .. ok Done.

T3 write A = 600, system reply should be done.

At T4 if someone read(A), reply should be 600 or 200, fine but should not be error, you might give a wrong/ stale value, buty still you cant give a reply as “not available”

available = low latency, doesn't guarantee latest value





When someone try to read B at T6: it might return 1000 or 200.. but always give a value can be Stale.

Availability can give latest or error, but never Stale.

In HTTP status code, 503 when service is unavailable. But never give you a wrong or stale value, that’s consistency mean.

Availability says, I can give you latest value or Stale value but always available. Doesn’t guarantee giving you latest data, but guarantee return some data but never error.

**Where I want my system to be available and where I want to see consistency?**

In bank I should give correct value, might not be available, but cant compromise consistency.

Youtube video me likes counts: give some incorrect/ not exactly correct value but not fail to return count value. No of ppl watched/ likes, I want YT to give some value, can be stale value.

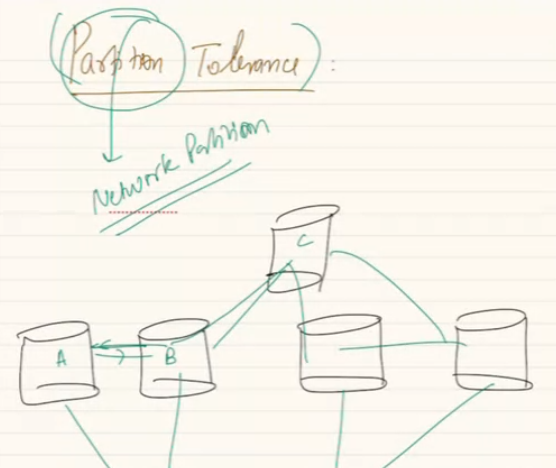
Some system inherently need to be consistent or available.

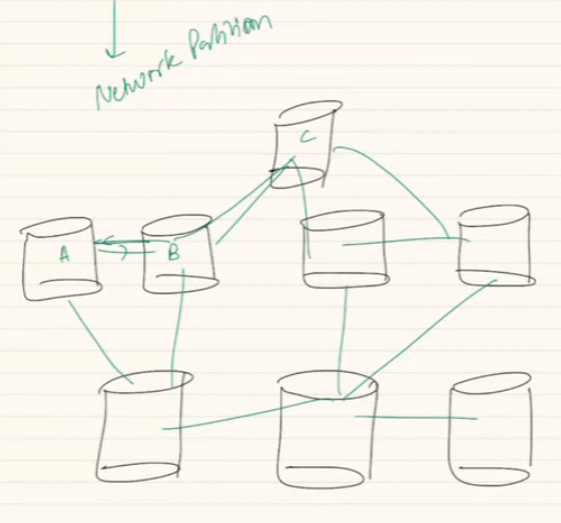
**Partition Tolerance (P)**

In last class Sharding means partitioning. In today class partition means something different.

**Partition Tolerance (P):** The system continues to operate and provide both consistency and availability even in the presence of network partitions, meaning that communication failures can occur between nodes.

Here partition means network partition, whenever I have multiple machines, they talk to each other via network. A cable/ wifi, they can be connected by a private network, internet, wifi, connected by wires.. etc. if system A wants to send a message System B, when received it sends an acknowledge. Sometime sends directly or after a hobs (via C).





Inside same machine you don’t need network, across diff machine if you want to read, write you need network.

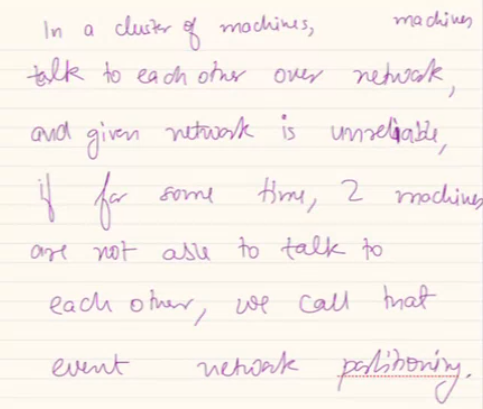
Network partition means, when 2 system are not able to talk to each other for sometimes, because underlying network is dead (rat cut it, overwhelmed, router is dead any reason.).. or not responding. This can happen momentarily. packets are not reaching. Or source packet reaching but response not coming. example: we are in a video call audio not coming etc. somehow for some reason its possible we can’t reach to destination machine.

**Network partitioning means, (important topic): when 2 machine cant talk to each other means NP.**

Partition of network partitioning means. when network is unreliable.

When a cluster of machines are cluster as they are able to talk to each other. There is a network, packets are getting transferred by wifi, optical cables or copper cables. Machines talk over a network. Network is always unreliable, when 2 machine not able to talk each other. We call the event as network partitioned. (they are not able to talk to each other at this point of time.)

Network has portioned between 2 machines.

****

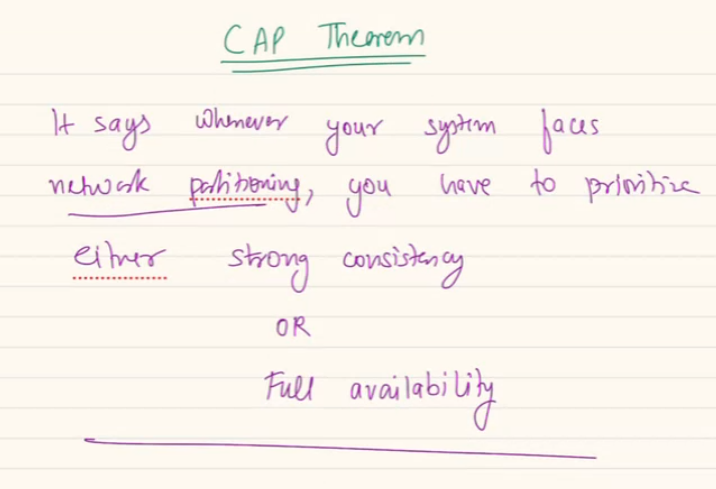
**CAP Theorm:**

A theorem formalized by google engineer, Eric, it says, a system tries to be consistent, tries to be available. Even if I try to achieve these together, unfortunately whenever network partitioning occurs

In a best-case scenario, fairy tale world, I want it to be available and consistent, in ideal world we want both things to be true at same time.

In real world whenever network portioning happens, when they can’t talk to each other, system is bound to take a decision to be consistent or available. You have to prioritize either strong consistency or full availability. System cannot be both at the same time.

Hen unavailable/ network partition, take a decision either consistent or available. Gpay goes on consistency, YT goes on availability. Pick any one when network portioning happens, when partition will never happen ..



If my cluster has only one machine it will be consistence and available both if the machine is running.

Analogous example: read in blog post, scaler use that example.

Why app server have to talk to each other: in consistence hashing data should be moved to new machine, so these machine also have to talk to each other.

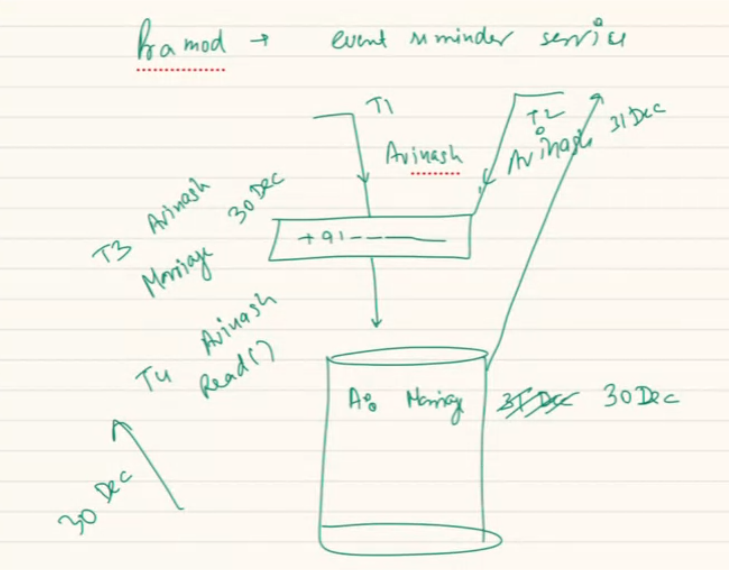
Also master-slave machine/ replication. here also mc has to talk to other mc. So not only LB talk to app in above cases also one MC has to talk to each other.

**Real world use case/ Example:**

Say we have a company and building a **event reminder service**.

It takes a phone number, promod can hear the call.. another user says, I want you to remember my birthday at 31st dec. when 2nd user asks.. promod answer and tell the event date.

A simple read and write, promod is doing..



One machine Pramod who is writing and reading…

Also can update the date… change the event from 31st to 30..

So in this case the same machine is giving the reply and giving the same correct reply.. not seeing the probability of portioning happening. as Pramod can talk to himself.

Available to give and correct answer. Unavailable happens only if the phone line is down..

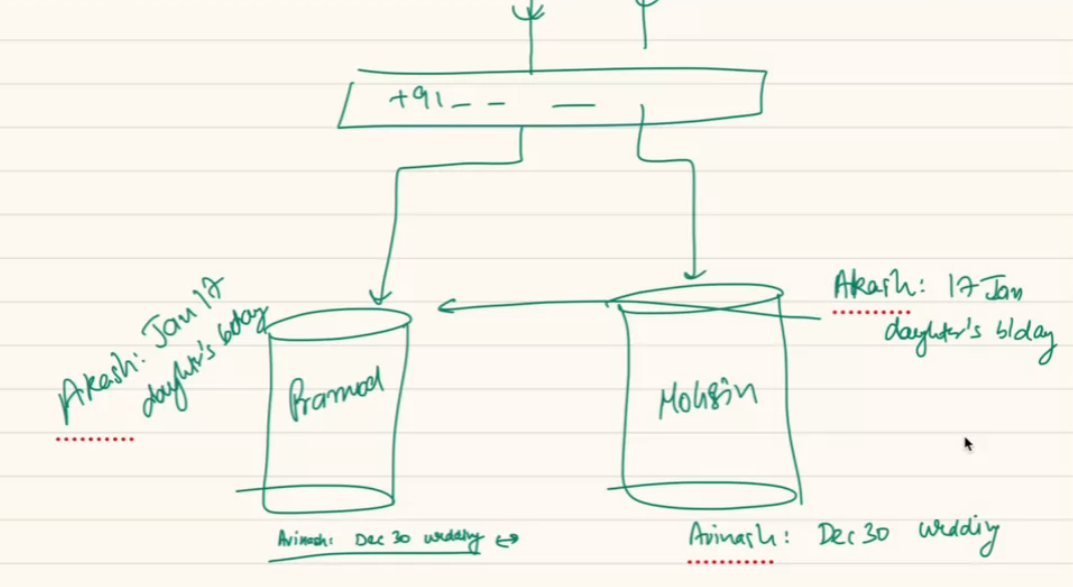
If I am only one reding/ writing I will be both consistent and available…

burning question: network partition within the cluster or outside cluster/system boundary as well ? we are talking about withing the cluster…

when we are talking, we are considering app server or db server cluster.. I am not talking about outside the cluster.

If Mohsin also joins with Promod…means we have 2 machine..

Call gets transferred to any of them when one writes the same is also persisted to other machine. Both of them are consistent.



If these machine no longer talk to each other..choice to make. Either take writes..

talk to each other.. between T8 🡪 T12… either Pramod can write vishal bday at 4 july.. but they are compromising on consistency. Mohsin wont reply.

Service is choose to available but not consistant.

Or Pramod can tell, I am not talking to my employee, I cant take your request. No write request.

Or Mohsin can pick and say I cant read, other machine is not available. Can you come back later..

So you either can be consistent or available.

**In case of network portioning a system can be available or consistant.**

**Choose Consistency over Availability (CP):**

* In this case, the system sacrifices availability to maintain consistency.
* The system will not respond to requests if it cannot guarantee that the response is based on the most recent data.
* It waits for the partition to be resolved before processing any requests to maintain a consistent state across all nodes.

**Choose Availability over Consistency (AP):**

* In this scenario, the system sacrifices consistency to maintain availability.
* The system continues to respond to read and write requests even during a partition, potentially serving stale or outdated data.
* Each node may operate independently during the partition, and conflicts may need to be resolved once the partition is resolved.

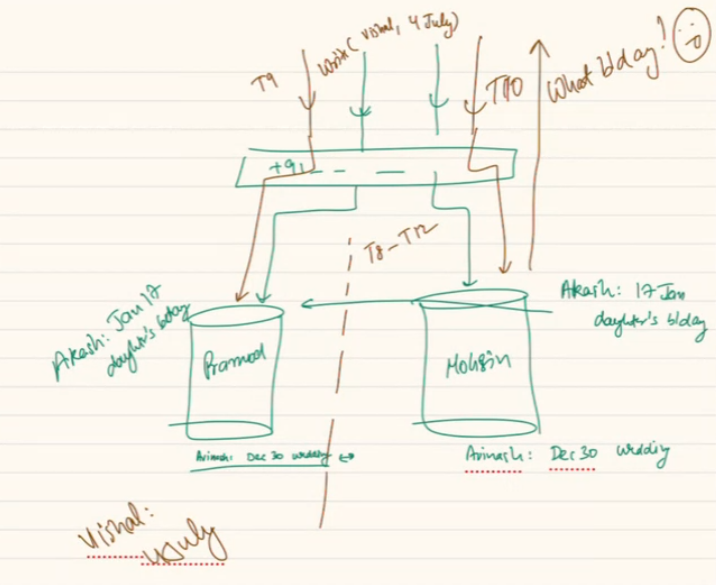
1. **Balancing between Consistency and Availability:**
   * Some systems attempt to strike a balance between consistency and availability by allowing a certain degree of inconsistency during network partitions while striving to provide some level of availability.
   * Techniques like eventual consistency, conflict resolution, or quorum systems may be employed to manage trade-offs between consistency and availability.

It's important to note that network partitions are considered a reality in distributed systems, and therefore, Partition Tolerance is a non-negotiable requirement. The choice between Consistency and Availability depends on the specific requirements of the application and the desired trade-offs.

Examples of databases/systems adhering to these principles include:

* **CP Systems:** MongoDB with strict consistency settings.
* **AP Systems:** Cassandra, Couchbase, or Amazon DynamoDB with eventual consistency.

Ultimately, the decision on how to handle consistency during network partitions depends on the specific goals and requirements of the application being developed.



what kind of question can be asked on this topic. pure theoritical? Going to form a base. In further case study we will take a decision. Ask question WTR to design a system. In terms of real time system/ problem..

If we are building a multiplayer gaming app: availability…

If building a banking: prefer consistency..

Comments on Facebook post: show some comments if one or two unavailable.



Video Ingestion of Hotstar: if I am uploading a TV series on hotstar… consistency > availability..



**what about trading app ? because both are important:**

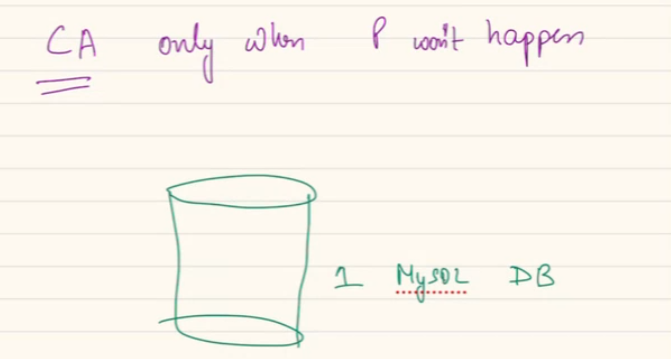
interesting, both are needed, but CAP says in case of Partion tolerance one should eb available. We will build a system when both system not talking to each other/ or probabllity of PT is less. May be a big single machine… great intranet.. stromg redundant connection so that Partion reduces. When partition happens I prefer consistency.. as I don’t want anybody to pay x+10… given alvailablity is also important …

at times in trading app we see delayed rates , isnt that example of availability ? yes

a real life example: Spanner DB by google… this is a DB created by google. Great piece of tech. this DB uses propriety hardware, uses specilized hardware.. super redundant network cables. Probability of not responding is low..

we connect 2 machne with multiple routes.. hardware is made such the probability of network partion reduces.. spanner usually shows consistency and availability. But it cant give a guarentee of this. So when Partion happens Spanner chooses consistency over availability.

If someone says.. Spanner destroys the CAP theorm.. wrong thing.. you say. “Spanner ensures that they have great propriety hardware and grate connection which ensures partition never happens. even if happens C > A.



Consistency is also a spectrum.. one end is called as strong consistency.

**Strong Consistency** means; every read will give the latest write..

In between we call it, **eventual consistency**: we are compromising our consistency, even we are not able to read the latest value right now, but guarantee after some time/ ultimately we will read the latest write.

**Eventual consistency** is often used in systems where low-latency reads and high availability are crucial, and it's acceptable to have a temporary inconsistency window

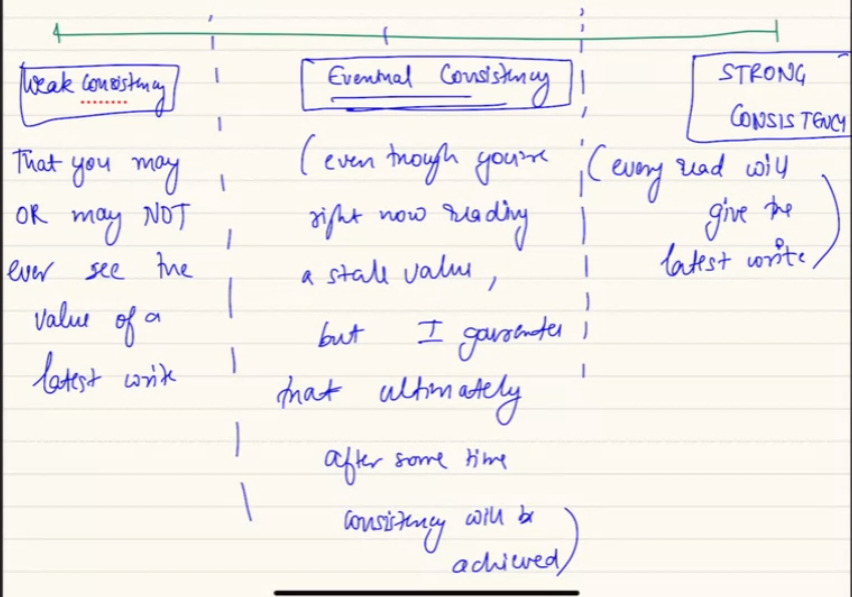
When we post, after sometime when we read the post we don’t get the latest write, means FB post is not strong consistent. But they still give us some consistency. means if you have written a comment, it might not be visible for next 5-10 means, eventually you will always see your comment. It might take sometime to reach consistency, but once it reads it will always be consistence. Even you are reading a stale value, but after sometime consistency will be achieved.

It is not propotional to time. it will be achieved with time.

**Weak consistency:** least value of consistency. Means that you may or may not ever see the value of a latest write. Your thing might get lost forever, it might never seen by you.

In YT live stream upvote, downvote, emoji. They pop up.. at the end of video.. would you be fine the emoji is lost forever.. here its fine.. not super critical.. here that event never got completely registered. Write happened never got registered..

Hotstar live count… never counter 1 person who is watching his value ever. Not even ultimately count him.



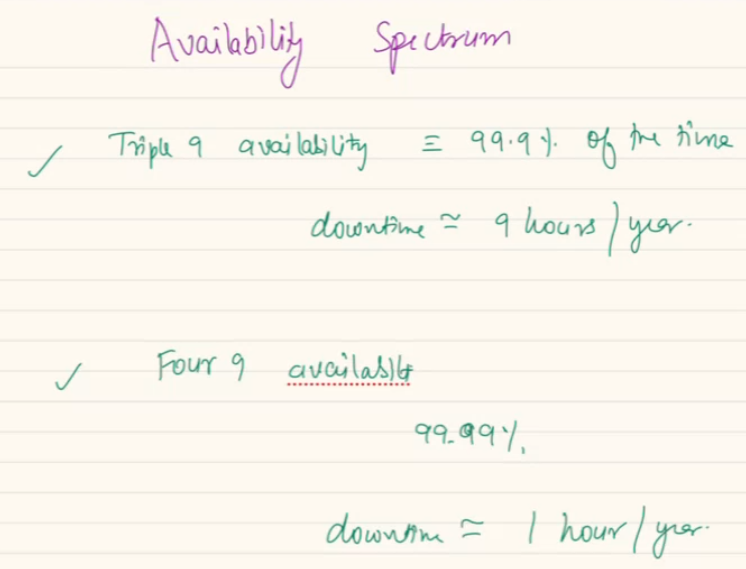
**BQ: If we have network partition for an hour and after that data synchronises then do we call this eventual consistency?** Yes, we call it eventual consistency.

In CAP theorm when we call consistency, we mean strong consistency..

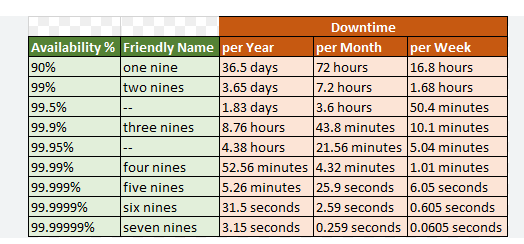
**Availablity Spectrum:**

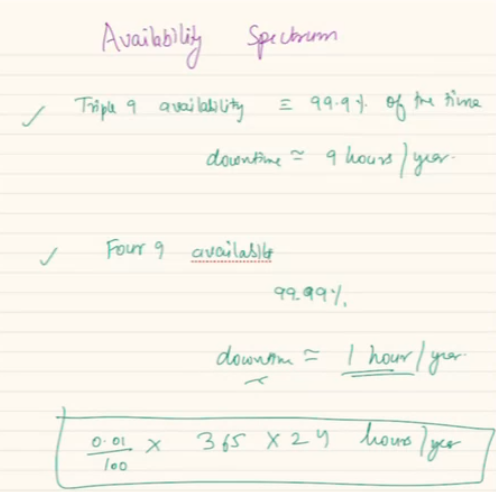
Define in 9s.

999: means your system is available 99.9% of time. if we build such system, means throughout the year it will have a downtime of 9 hour/ year.



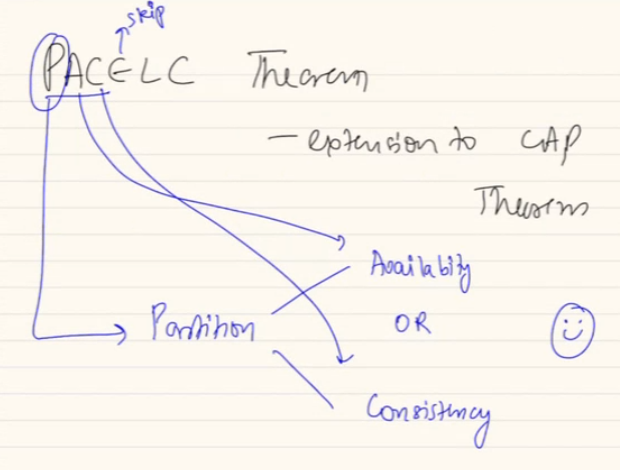
Four 9: 99.99%: downtime 1 hour/year.





**PACELC theorem: an extension of CAP theorm..**

If you have an partition, when partition happens.. then you will choose between availability and consistency.



PAC is partition, availability, consistency.

Even if partition happens you chose..

11.08..