* Consistency Models
* Data Centric Consistency Models
* Strong Consistency Models 🡪 Operations on shared data are synchronized
* Strict Consistency
* Linearizability Consistency
* Sequential Consistency
* Causal Consistency
* FIFO Consistency
* Weak Consistency Models 🡪 Synchronization occurs only when shared data are locked and unlocked
* Weak Consistency
* Release Consistency
* Entry Consistency
* Client Centric Consistency Models
* Monotonic Reads
* Monotonic Writes
* Read Your Writes
* Writes Follow Reads
* Distribution Protocols
* Replica Placement
* Server Initiated Replicas 🡪 Drop/Replicate/Migrate file to the closest server of a client or to a server where client congestion is high
* Client Initiated Replicas
* Client caches
* Share cache among clients, normally place at some machine
* LAN / WAN cache
* Update Propagation
* State vs Operation
* Propagate only notification/invalidate of update
* Transfer value/copies from one copy to other
* Propagate the update operation to other copies
* Push vs Pull
* Based on lease
* Epidemic Protocols
* Anti-Entropy
* Gossip
* Deleting Values
* Consistency Protocols
* Primary-Based Protocols
* Remote-Write protocols
* Local-Write protocols
* Replicated-Write Protocols
* Active Replication
* Quorum-based protocols
* Cache-Coherence Protocols
* Rad-only cache
* Write-through cache
* Write-back cache

-----------------------------------------------------------------------------------------------------------------------------------------

Linearizability 🡪 means write are happening on one data that is present in **single database**. When multiple writes are happening, each process will take a lock and then it will update the data item. The order is defined whichever request reaches first and get the lock will write first. i.e. if T1 is invoked before T2, then it will guarantee that order T1 🡪 T2

Serializability 🡪 means updating item ‘x’ which is present in **more than one replica** with respect to multiple transactions. Let say we have R1 & R2 and two transaction trying to update it T1 & T2. Serializability will guarantee that item ‘x’ after T1 & T2 in both replica will be in consistent state.   
Both no guarantee which will happen first, i.e. T1 🡪 T2 or T2 🡪 T1. But it guarantees that order of execution of transaction will be same at all replicas

It involve overlapping of two transactions as long as the result is equivalent when they are executed serially. Refer to Neha Narula’s talk

Strict Consistency 🡪 is basically Serializability with Linearizability. It means the order will be maintained and replicas will be consistent. I.e. If T1 is invoked before T2, then its guarantee that order of execution will be T1 🡪 T2 at all the replicas. But this will degrade performance.  
Strict consistency can be achieved by **2-Phase Commit** synchronously.

Sequential Consistency 🡪 This is at read/write operations levels. Will allow read & write operations of multiple transaction to interleave with each other. There could be several sequences of execution out of which few will be valid. But only one valid sequence is chosen & the same sequence is executed at all the replica

-----------------------------------------------------------------------------------------------------------------------------------------

**Difference between Consistency in ACID and CAP**

C in ACID 🡪 the consistency property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including [constraints](https://en.wikipedia.org/wiki/Integrity_constraints), [cascades](https://en.wikipedia.org/wiki/Cascading_rollback), [triggers](https://en.wikipedia.org/wiki/Database_trigger), and any combination thereof. This does not guarantee correctness of the transaction in all ways the application programmer might have wanted (that is the responsibility of application-level code), but merely that any programming errors cannot result in the violation of any defined rules.

C in CAP 🡪 In the CAP definition, consistency means that all the nodes in a cluster (e.g. all your database servers, leaders and replicas) see the same data at any given point in time. In practice, it means that if you query any of your database servers at the exact same time, you will get the same result back

-----------------------------------------------------------------------------------------------------------------------------------------

**For avoiding write-write conflict**

* Use qurom based protocol 2W > N to avoid write-write conflict
* Database sharding 🡪 user profile is present in separate database. All write is send to that data base only. All the update operations for a user is send to a specific database i.e. sticky session
* Use time-stamp (Lamport or Vector) and perform all the write operation in that order after checking if it’s not violating any constraint.   
  E.g. Two users trying to create profile with same user id at two different geographical separated regions.
* Before performing any write operation, the program should validate if user id is not existing in the system.
* Write conflict can be avoided by using **strict consistency** for user registration operation
* This can be achieved by using sequential consistency with time stamp