# ***SYSTEM DESIGN:***

Master -> Slave Architecture !!

Pizza shop we have to Scale:

* Three chef 1,2,3,
* 1 – pizzas, 2- garlic bread, this is what we called micro services Architecture

Create a backup of the shop and replicate the same for pizza shop

HLD VS LLD:  
  
High level is like how will u make a system distributed and how will make u a system deployable and how to logging the system and all

LLD Is more related to Code stuff classes and objects

Needs in System Design:  
1. Load balancer

2. microservice architecture

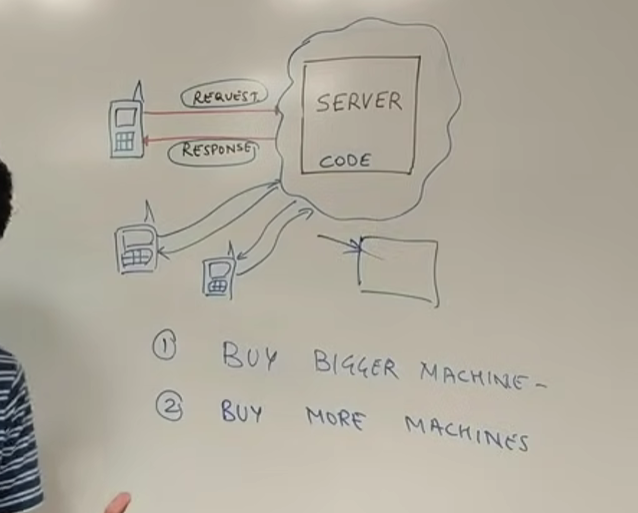
3. Distritbuted System (Partitions)

4. Decoupling

5. Logging and metrics

**#Horizontal VS vertical scaling in Design**

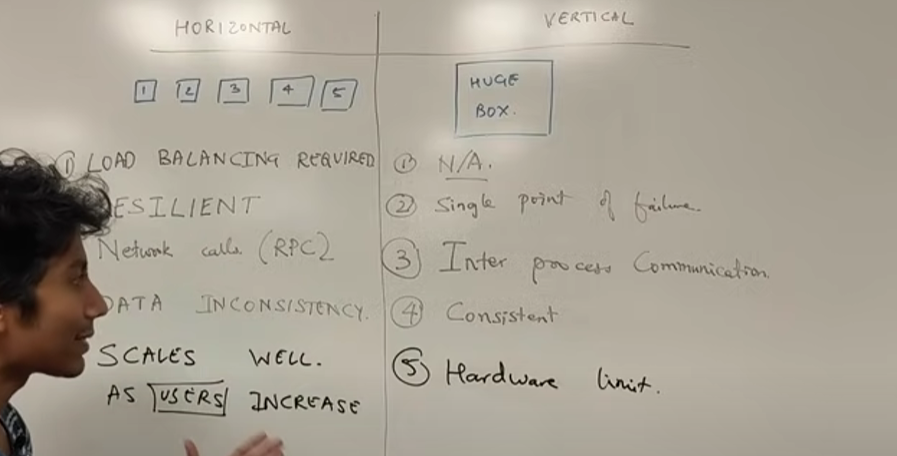
**Cloud** is *set of computer to make a reliable system:*

**

1. *Buy Bigger machine is a Vertical Scaling in this we will increase the ram and rom of the value increase the disc size of the machine –* ***Vertical Scaling.***
2. *Make a chunks of machine with same capability like ram and rom and size of the disk “Buy More Machine” –* ***Horizontal Scaling.***

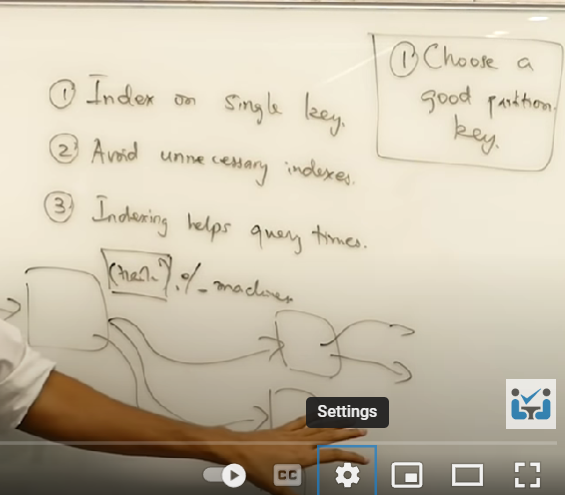
***These Are 2 mechanism where we can increase the scalability.***

***#Difference Between Horizontal and Vertical scaling:***

******

***These are the major differences of both the scaling techniques.***

***Database Indexing:-***

******

***A sort key define the order in which data is to be stored in the cluster nodes.***

***DynamoDb : Sort Key***

***ElasticSearch : Index Sorting***

***Cassandra : Clustering Column***

***Example : Select \* from user where country =”India” and gender=”Females”***

***Orderby City;***

***Expensive order because get the value is O(n) + sortBy(city) – O(nlogn)***

***Better Example :****SELECT \* from user where sort\_id LIKE “India + Female%”;*

*You will run through the data where binary search will work*

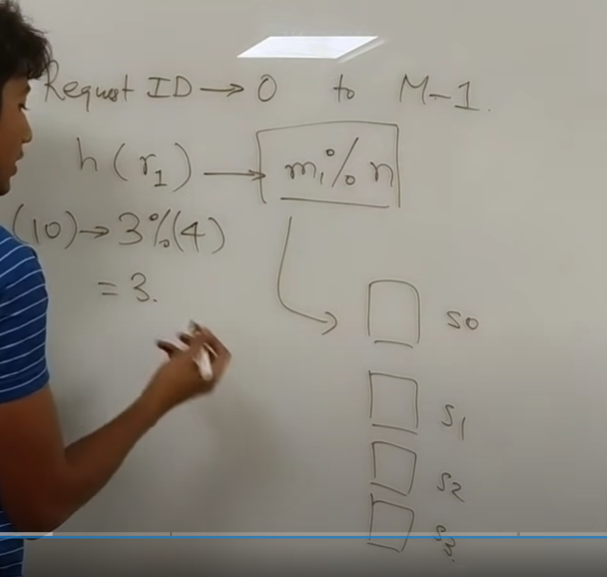
***#Load-Balancing: & Consistent Hashing***

*A box we have that is nothing but a computer a server is something which serves something to the request*

*If we can say if we are getting lot more request and we have to scale our system we make the replication system after scaling then request will redirect to which server to serve the request*

*We have load on the server and how to balance this load is called load balancing!!*

* *We will get request id in each request from 0 to m-1 on to the server* 
  + *Hash(r1) = m1%N (this number can be mapped to a particular server after % n we will get index we will redirect to that server )*

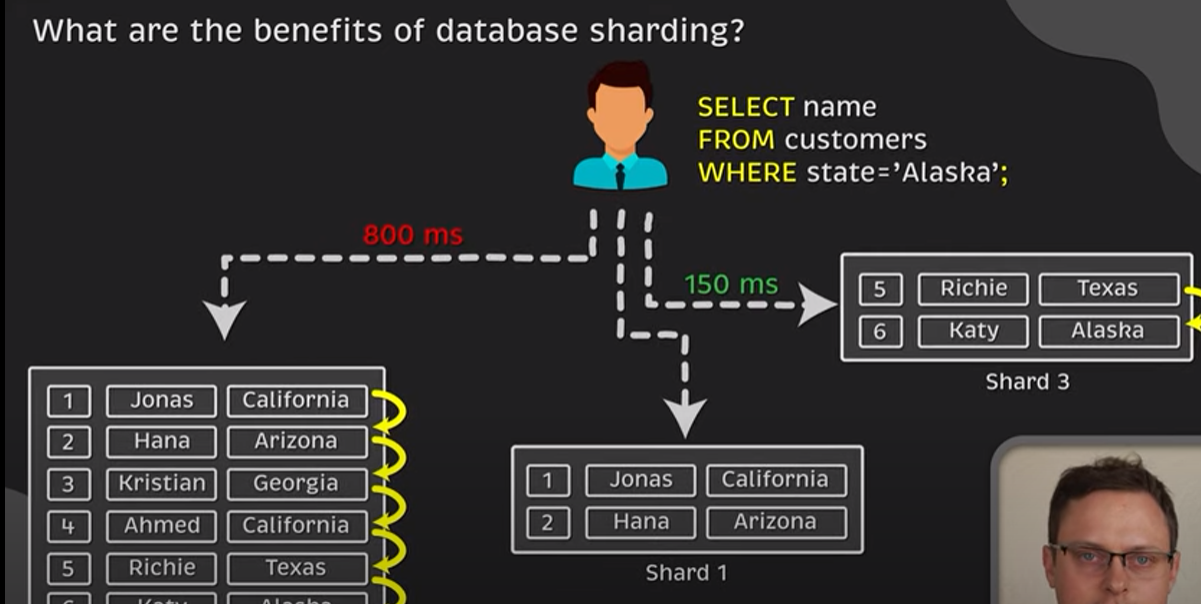
*  
uniformly serving the request!*

*What if we consistently get the same hash value then there is a chance one server can handle or loaded with lot of request!!*

*#****Problem is adding new servers and hashing we are going to use :****How to change the scenario*

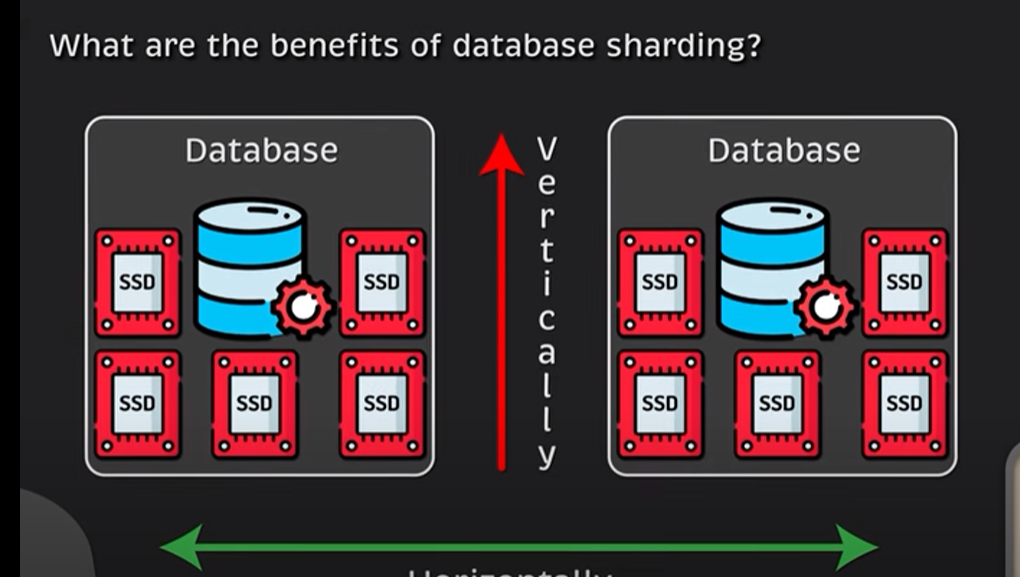
*Ring of hash value we have*

***DATABASE SHARDING :***

**

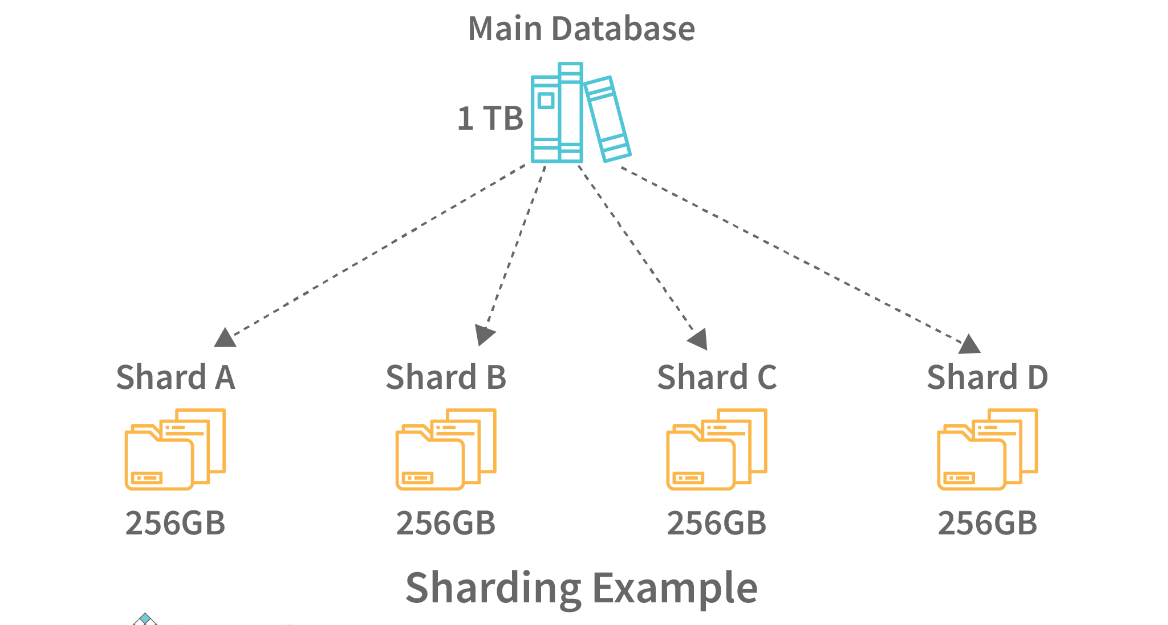
*It can help to avoid total service outage*

*Sharding – Vertically & Horizontally*

**

Sharding is a process of splitting the large logical dataset into multiple databases. It also refers to horizontal partitioning of data as it will be stored on multiple machines. By doing so, a sharded database becomes capable of handling more requests than a single large machine.

1TB of data we have we divide that into 4 SHARDS each of which containing 250GB data

**

*In Shading join across the shards are expensive thing to do*

* *Index on the shard are good to do things*
* *If shard fails then we can have master slave architecture write request always come to the master and slave polls the master for reading and updating the data consistency !!*

*#****CAP – Theorem:***

*C – Consistency, A – Availability, p –* **Partition Tolerance**

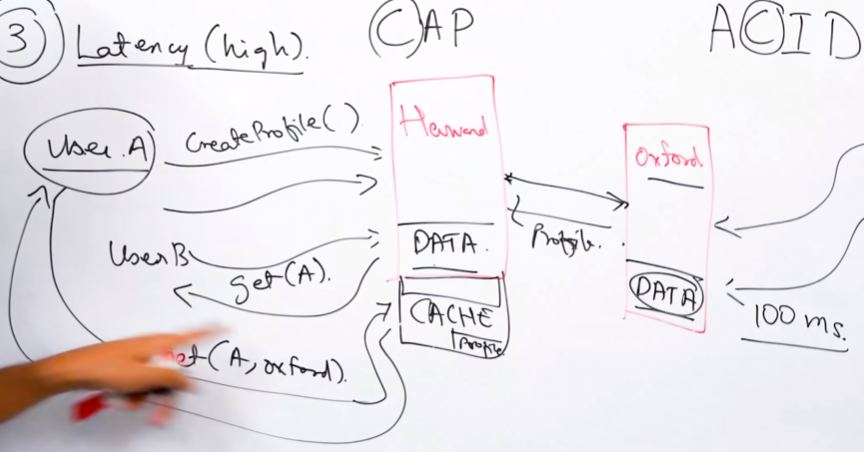
*If we have a single server*

*User (a) have some profile and user (b) need to access the profile they can fetch the details for the same!!*

*But there is only one server and that is single point of failure if unexpected event occurred.*

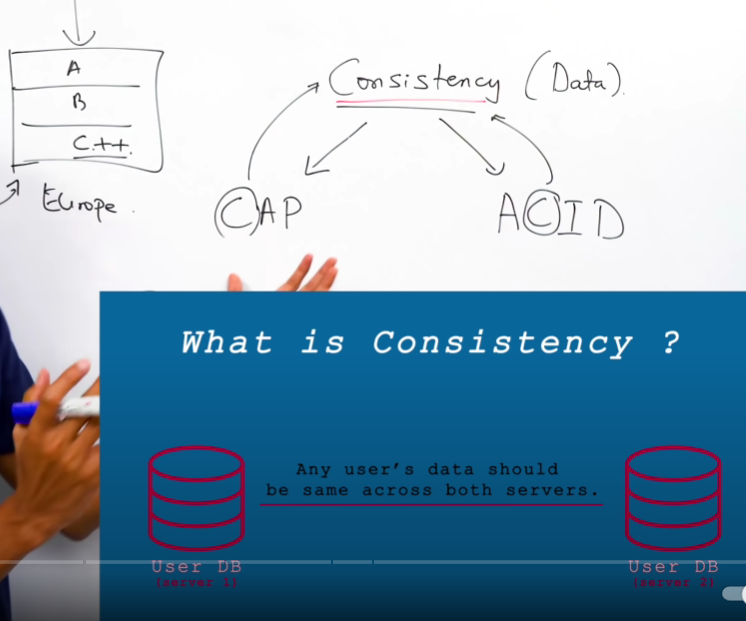
*#Problems:*

1. *Single point of failure*
2. *Cost of vertical scaling (High) after certain time it will hit it’s limit*
3. *Single node can have a high latency – Solution “We need to do horizontal scaling region wise”.*
4. *We can cache the data value for different server*

**

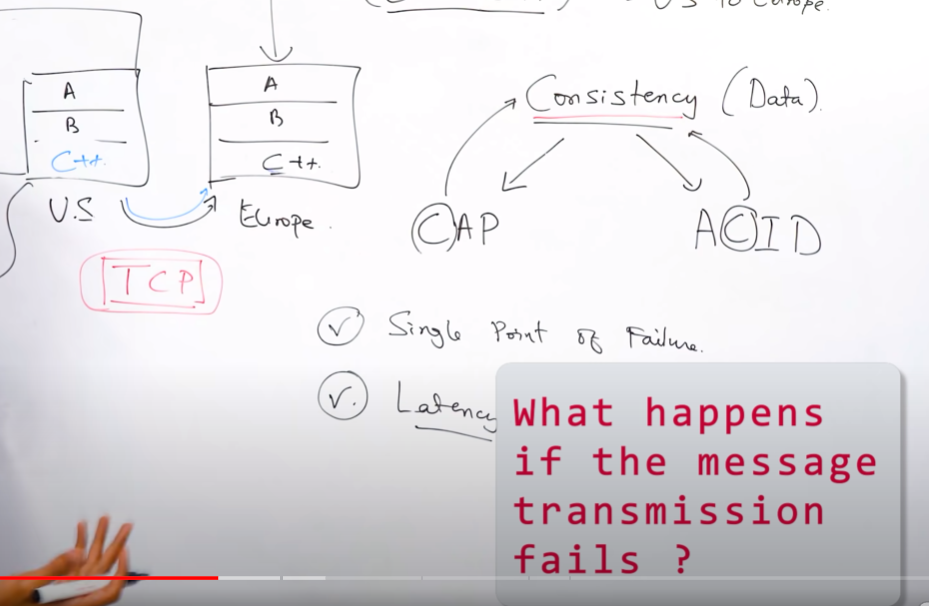
*If cache miss then still there is a latency issue*

*Solution for* ***Single Point Failure we have*** *replicate database in different region*

*#****Consistency*****

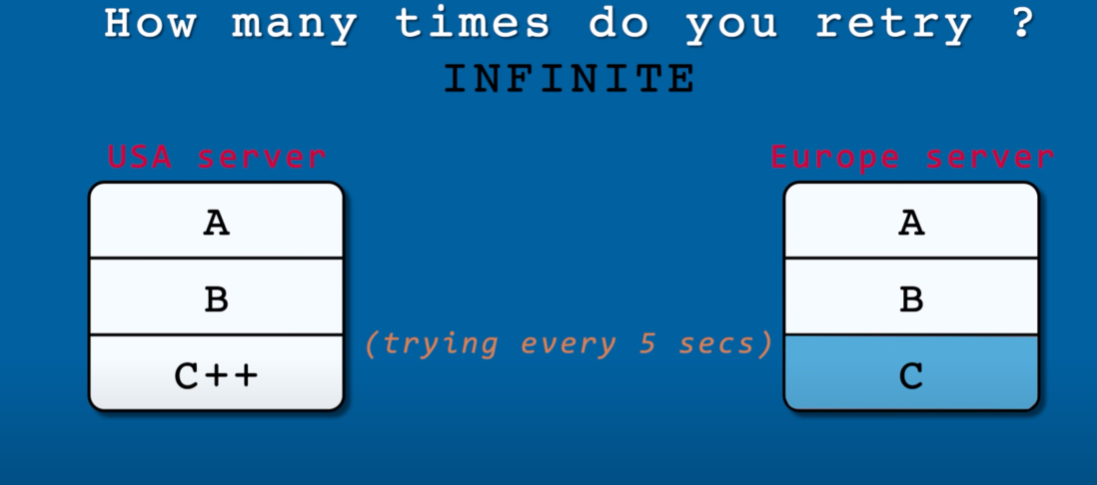
***Data must be consistent across the region of all the db and updates Must Follow the ACID Properties!***

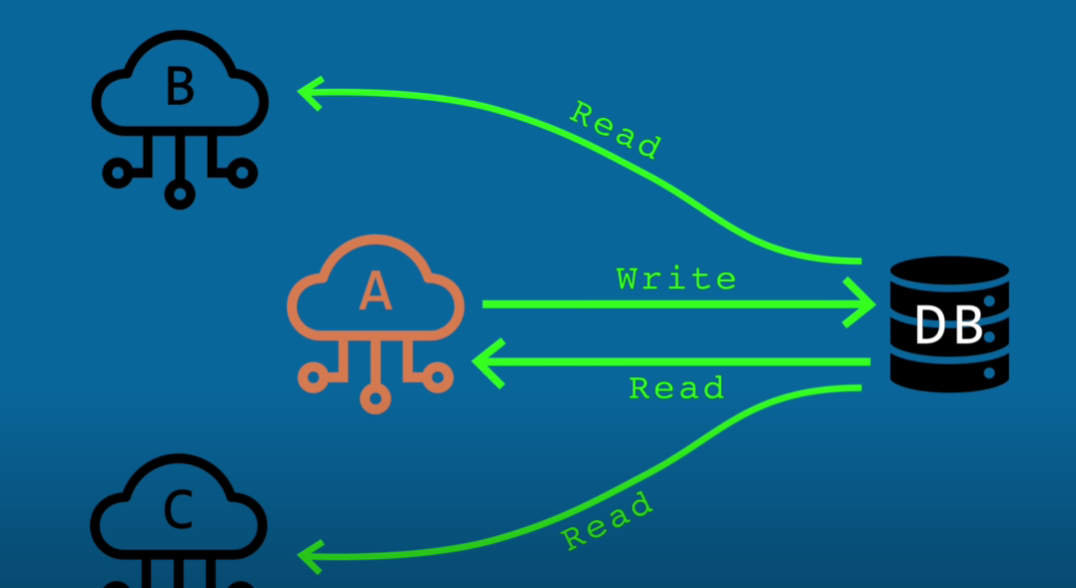
***TCP is reliable delivery protocol***

**

*If we are changing the data in Europe dB from c -> c++ if message transmission get fails then what we can do for this*

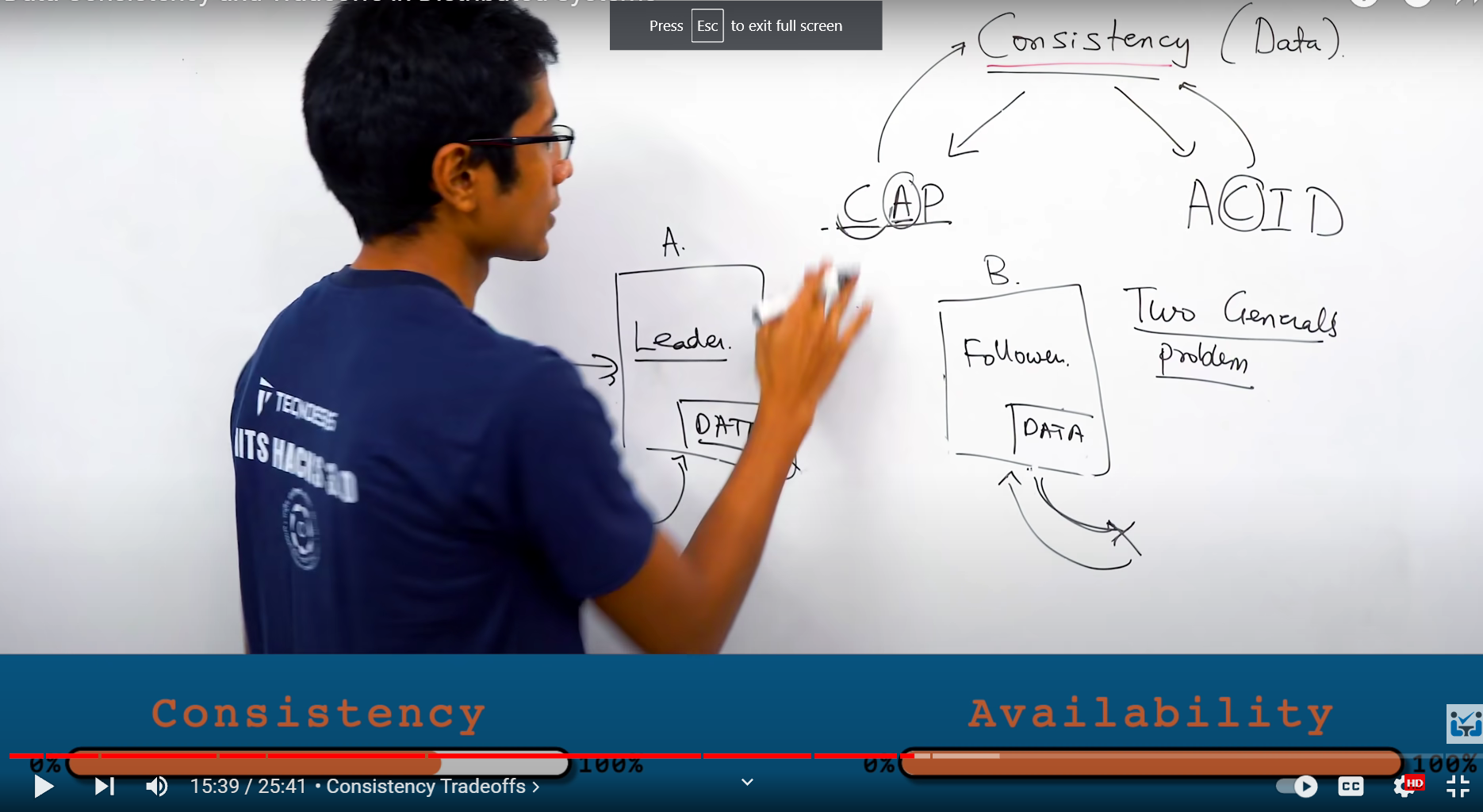
*Then Europe server should send ACK , What if ack also fails*

**

**

*Here what we have did we form a* ***leader*** *Server – A which is the only person responsible for writing anything into the DB if update needs to be send then it must goto Server – A not b or c*

*If u hit the system ans system is not available then there is a dirty read for the system*

**

*If u have pure consistency then there is no availability if u have available system then there is a chance of consistency issue*

*#2-phase commit protocol :*

*When a leader gets the update we send the prepare request to our follower and our follower givers the ack after ack u ask follower to commit the data*

*ACID - Consistency hold into the give system !!*

*When we commit the data then it should reflect to every where*

*Once the leader rollback*

*Note : If one of the node or server commits fail but leader update the value the failure node will roll back the value and the data become inconsistent*

*As from above chart we can see that Server – A update to c -> c++ but server b commits fails it roll back to previous value at this point of time we have the data inconsistency “We can’t allow this” the problem is “Roll Back”*

***We have to wait master message for rollback without that we aren’t allowed to the roll back***