Real-Time Backend

1. What is a distributed system

A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another

M.van Steen & A.S Tanenbaum

2. Features

a) Parallelism: /ˈpær.ə.lel.ˌɪz.əm/ (n) tính song song

more users/data - solve tasks faster

b) Fault tolerance /ˈtɒl.ər.əns/ (n) chịu lỗi

continue working even a part of a system fails

c) Physical distribution

2 users in different countries

d) Security / Isolation /ˌaɪ.səlˈeɪ.ʃən/ (n)

admin access / regular access

def: the fact that something is separate and not connected to other things (sự cô lập)

e) Separation

premium users / regular users

3. Challenge

Resources
Partial faiures
Performance

4. Infrastructure

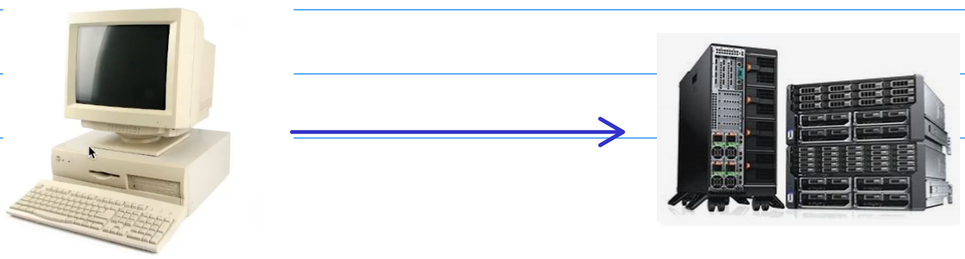
Storage (GFS, S3, SQL db, NoSQL db, Clickhouse)

Communication (Message brokers, GRPC, HTTP)

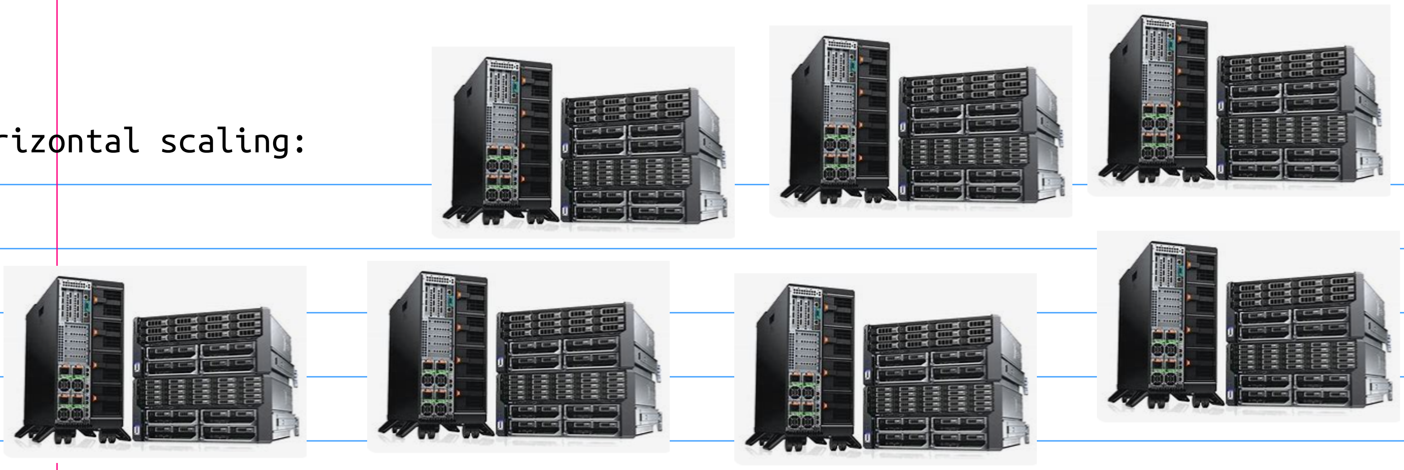
Computation (Flink, MapReduce, Spark)

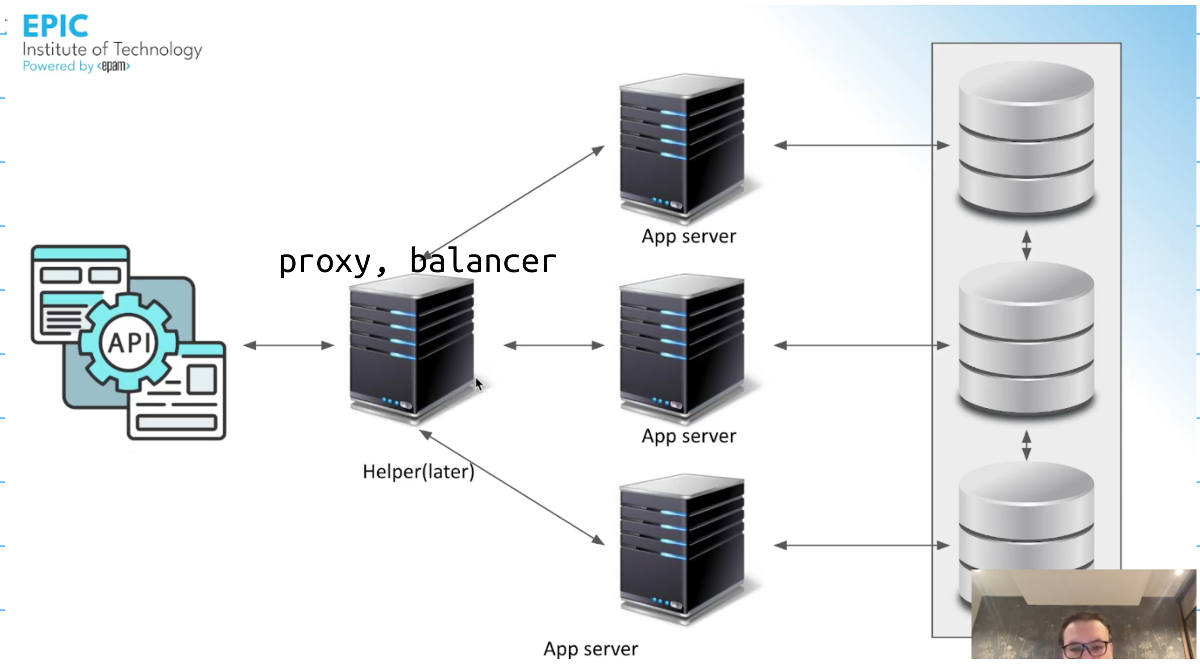
5. Scalability

vertical scaling:



horizontal scaling:





proxy, balancer

What we mean by fault tolerance:

Availability: set of failures we still provide service

Recoverablity: can recover from a failed state

How to achieve

Using permanent storage

Replication

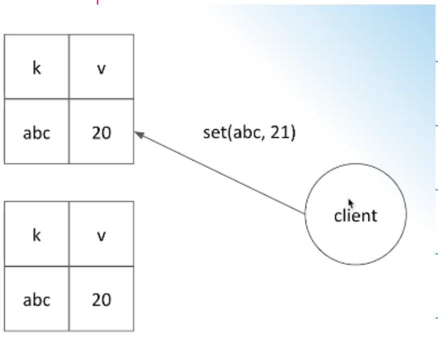
Consistency (tính nhất quán)

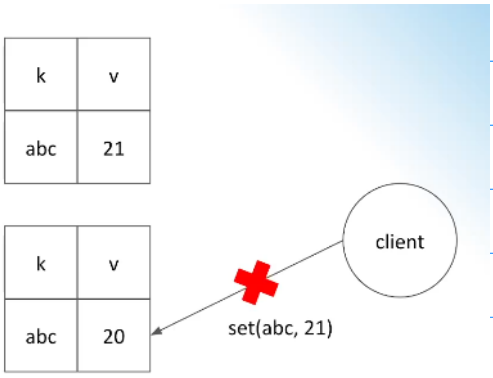
KV service has 2 api

set(k, v): set the value to v
get(k): it should return v

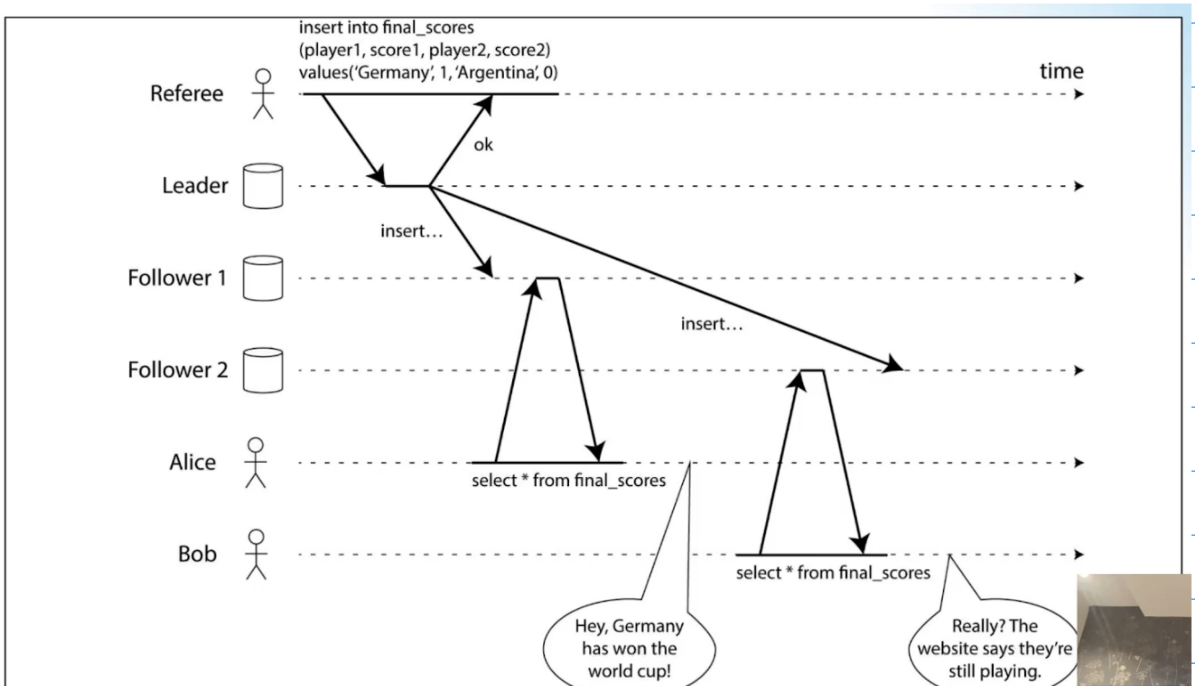
A: Hey update abc to 21

S: hey client, go by yourself and ask one more time.





An example



insert into final_scores
(player1, score1, player2, score2)
values('Germany', 1, 'Argentina', 0)
time
Referee
Leader
Follower 1
Follower 2
Alice
Bob
ok
insert...
insert...
select * from final_scores
select * from final_scores
Hey, Germany has won the world cup!
Really? The website says they're still playing.

CAP Theorem

A shared-data system can have at most 2 of 3 following properties:

Consistency, **A**vailability, and tolerance to network **P**artitions

On consistency

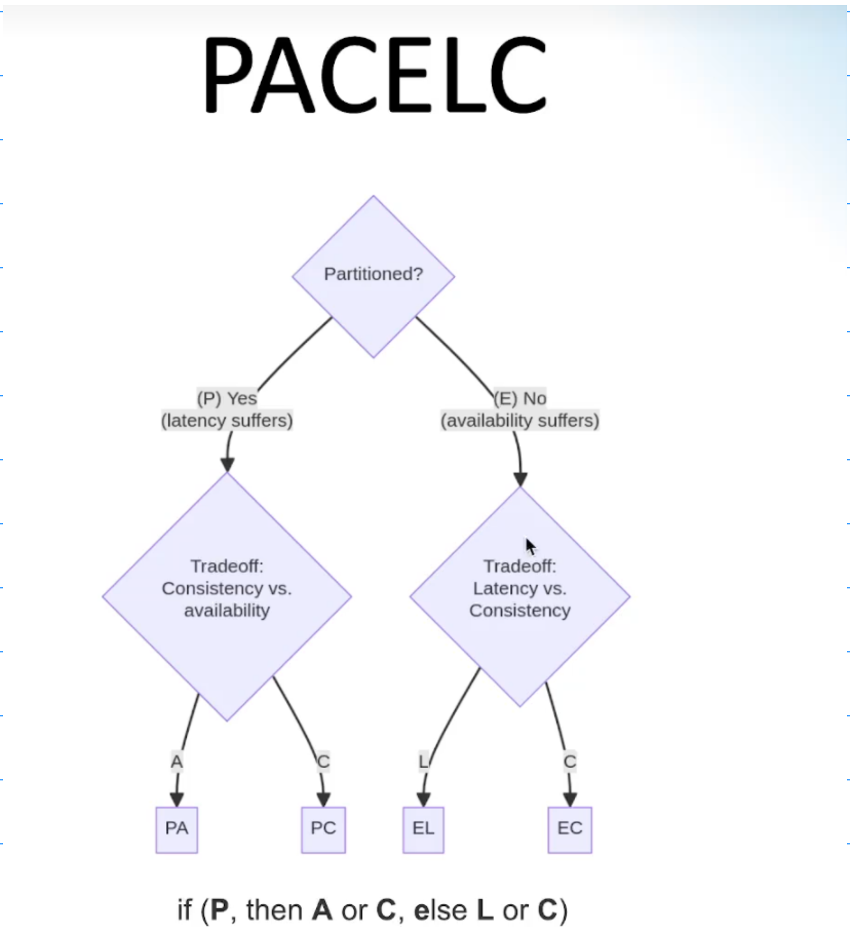
Atomic, or linearizable, consistency is the condition expected by most web services today. Under this consistency guarantee, there must exist a total order on all operations such that each operation looks as if it were completed at a single instant. This is equavalent to requiring requests of the distributed shared memory to act as if they were executing on a single node, responding to operations one at a time.

On availability

for a distributed system, to be continuously available, every request received by a non-failling node in the system must result in a response. That is, any algorith used by the service must eventually terminate ... [When] qualified by the need for partition tolerance, this can be seen as a string definition of availability: even when severe network failures occur, every request must terminate

On partition tolerance

In order to model partition tolerance, the network will be allowed to lose arbitrarily many messages sent from one node to another. When a network is partitioned, all messages sent from nodes in one component of the partition to nodes in another component are lost


PACELC
Partitioned?
(P) Yes (latency suffers)
(E) No (availability suffers)
Tradeoff: Consistency vs. availability
Tradeoff: Latency vs. Consistency
A
C
L
C
PA
PC
EL
EC
if (P, then A or C, else L or C)

Communication, distributed systems models. Fault tolerance

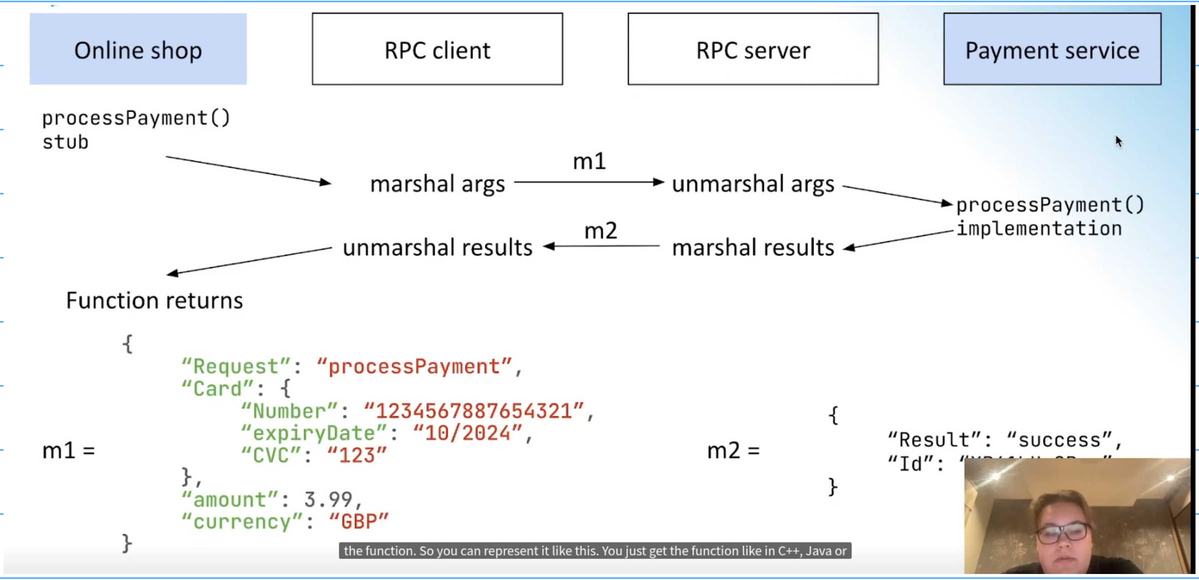
HTTP:

Communication is stateless (each req is self-contained and independent from other reqs

Resources are presented by URLs

The state of a resource is updated by making a HTTP request with a standard method type, such as POST or PUT, to the appropriate URL

RPC (Remote procedure call):



m1 =
```
{
    "Request": "processPayment",
    "Card": {
        "Number": "1234567887654321",
        "expiryDate": "10/2024",
        "CVC": "123"
    },
    "amount": 3.99,
    "currency": "GBP"
}
```

m2 =
```
{
    "Result": "success",
    "Id":
}
```

the function. So you can represent it like this. You just get the function like in C++, Java or

Service-oriented architecture (SOA) / "microservices":

Splitting a large software application into multiple services (on multiple noc communicate via RPC

Broker