CS 7172
Parallel and Distributed Computation

CAP

Kun Suo

Computer Science, Kennesaw State University

https://kevinsuo.github.io/

Outline

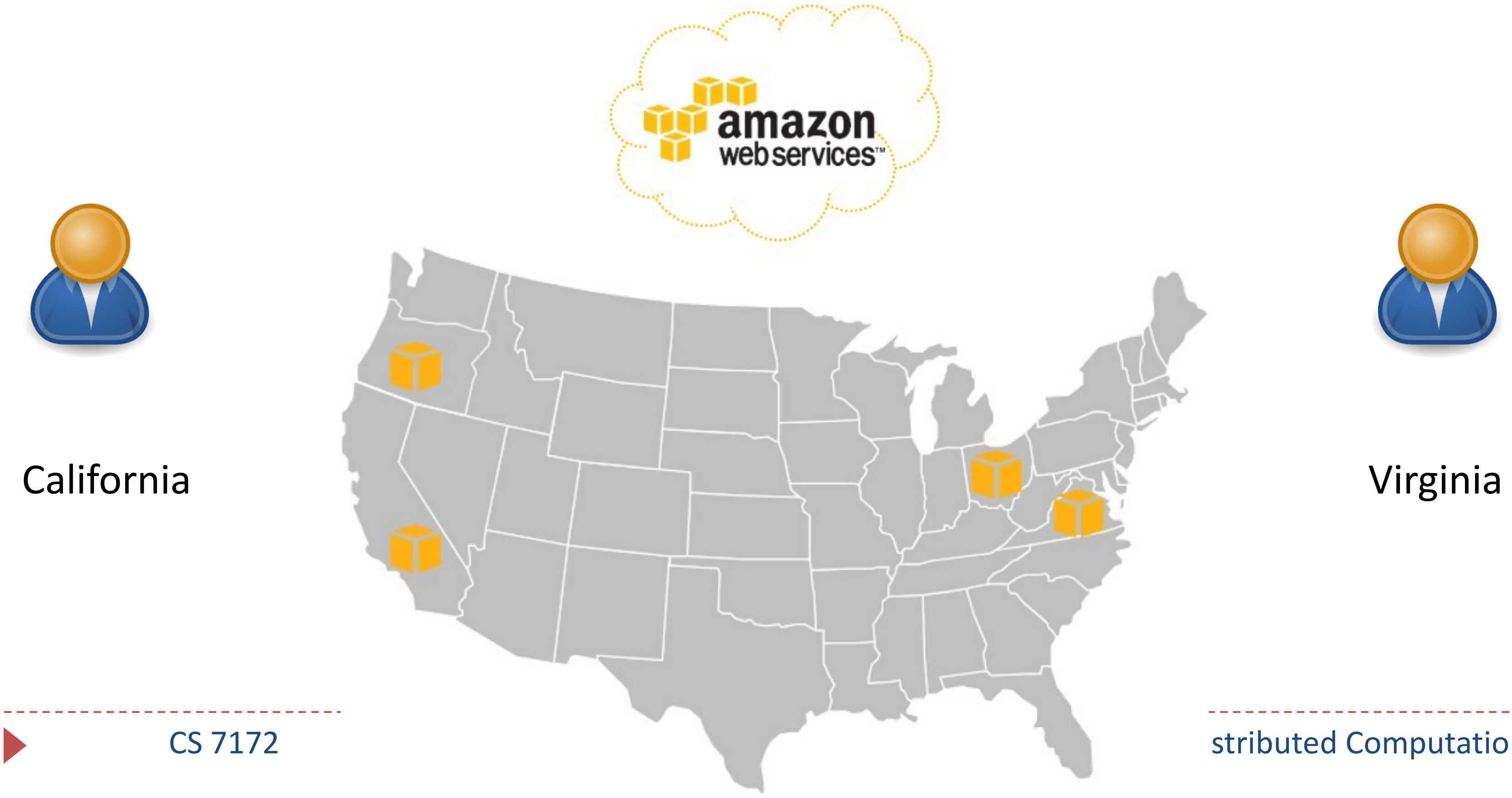
- Computer networks, primarily from an application perspective
- Protocol layering
- Client-server architecture
- End-to-end principle
- TCP
- Socket programming

Data in Distributed System

- The key object of distributed system processing is data
- How to process data in distributed system?

Data in Distributed System

- How to ensure that the data queried by different regions accessing different servers is consistent



California

Virginia

CAP

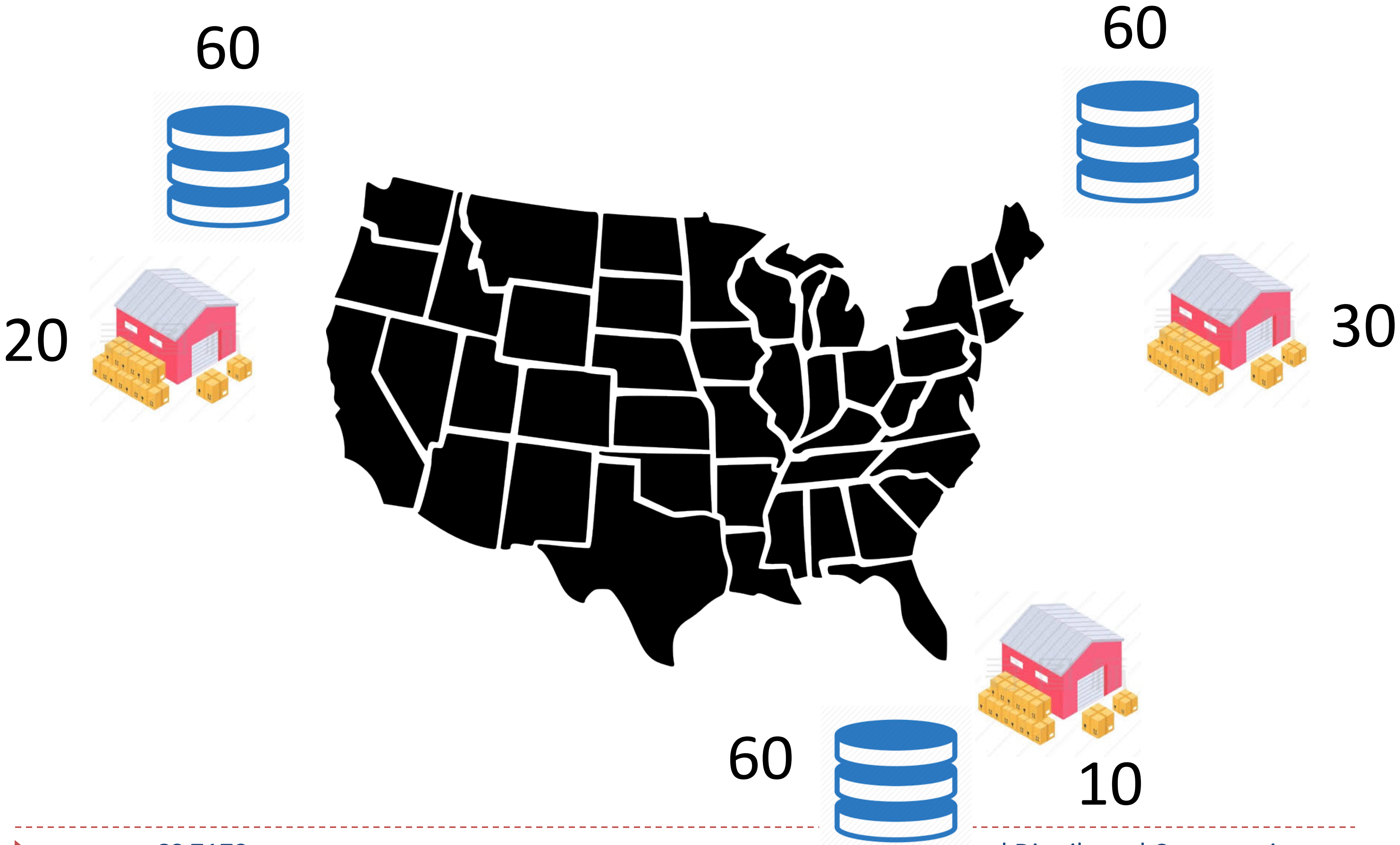
- What is CAP?

CAP: consistency, availability, partition tolerance

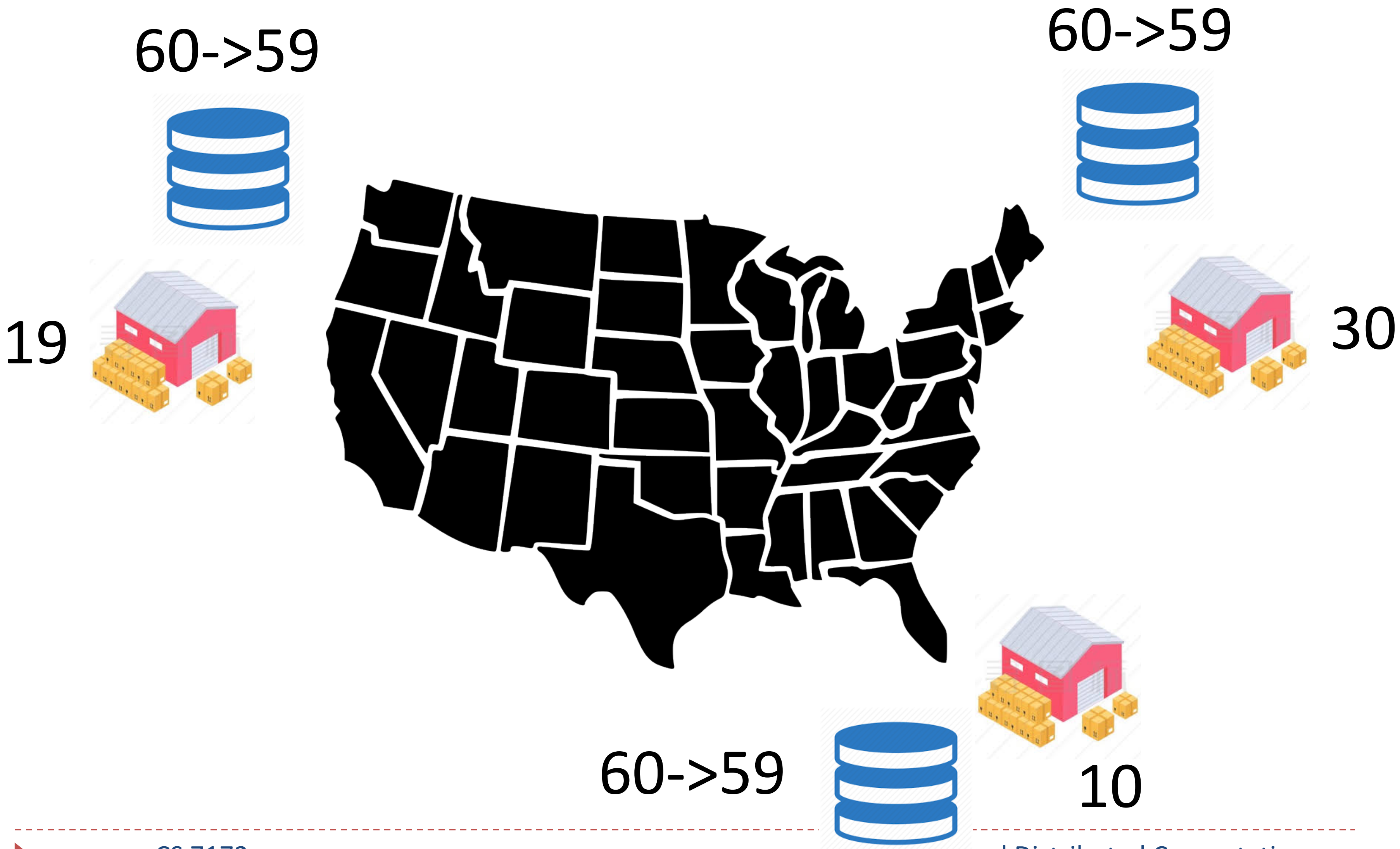
- Consistency: the data of all nodes at the same time is the same

- After the response is completed and the update operation finishes, the data stored by all nodes must remain the same

Consistency



Consistency

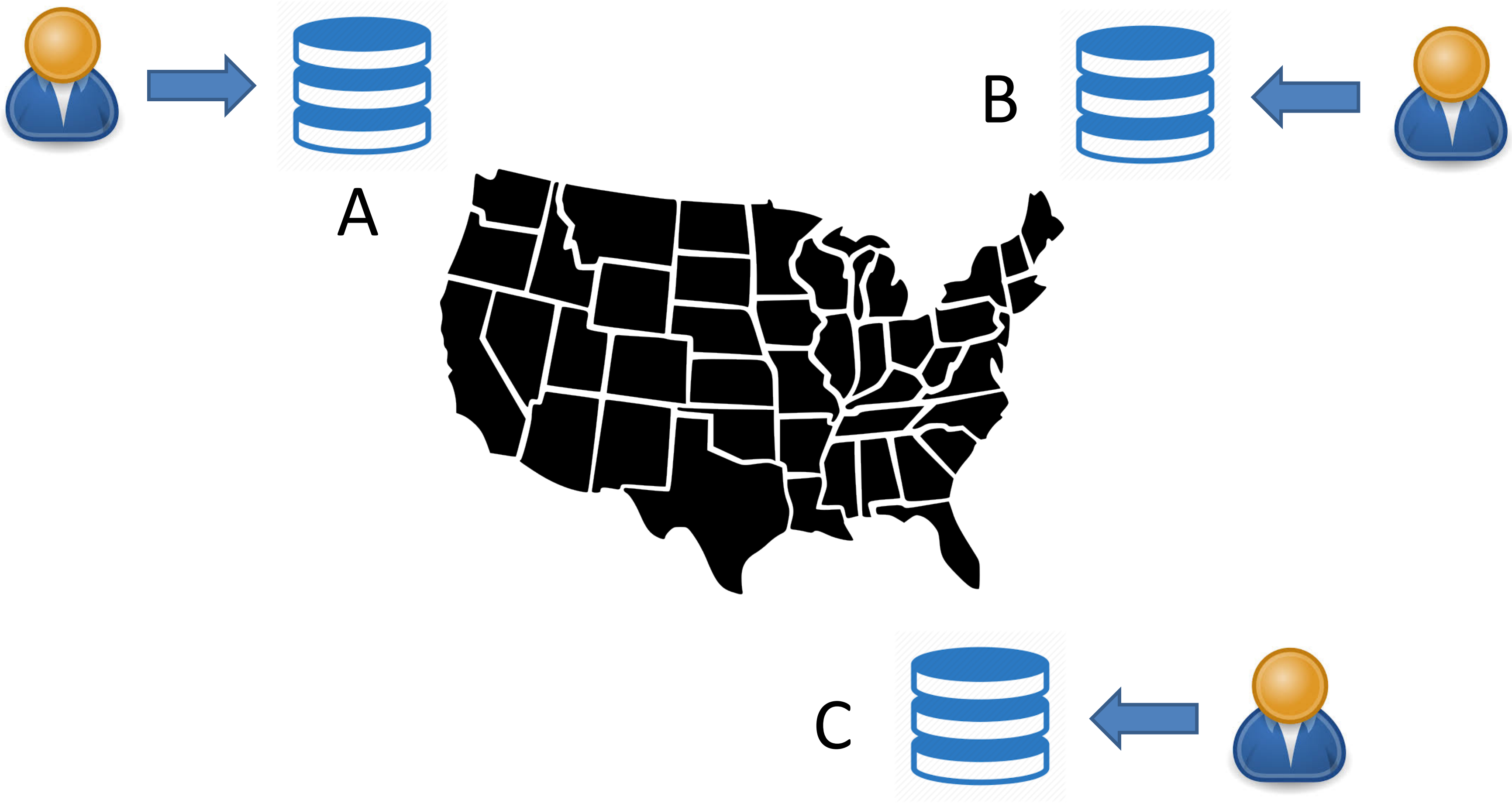


Availability

- Availability means that the services provided by the system are always available and can respond to user requests immediately

- E.g., server can response to user whenever the user sends requests to server A, or B, or C.

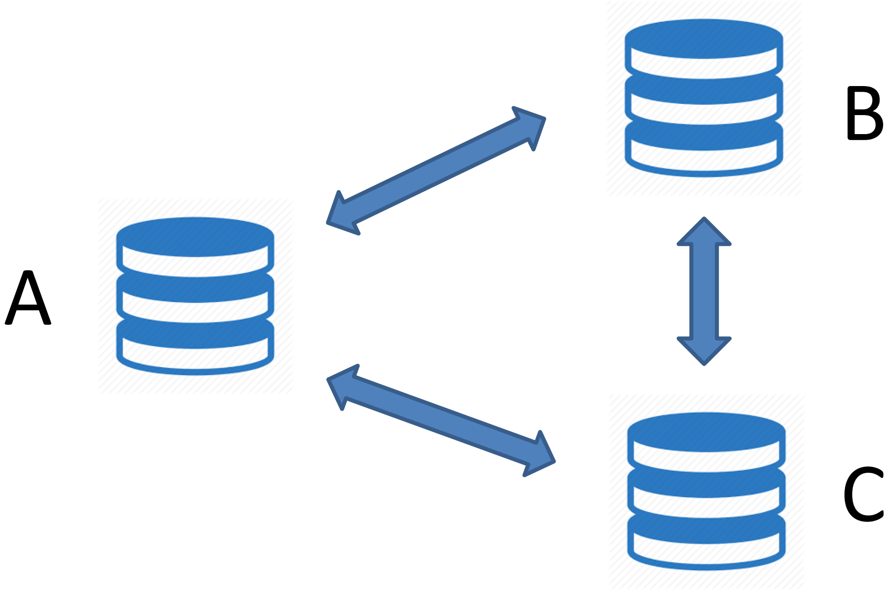
Availability: when a user sends a query request



Partition tolerance

- ***Network partitioning***: the network is disconnected due to a network failure. Different nodes are distributed in different subnetworks, and the network within each subnetwork is normal.

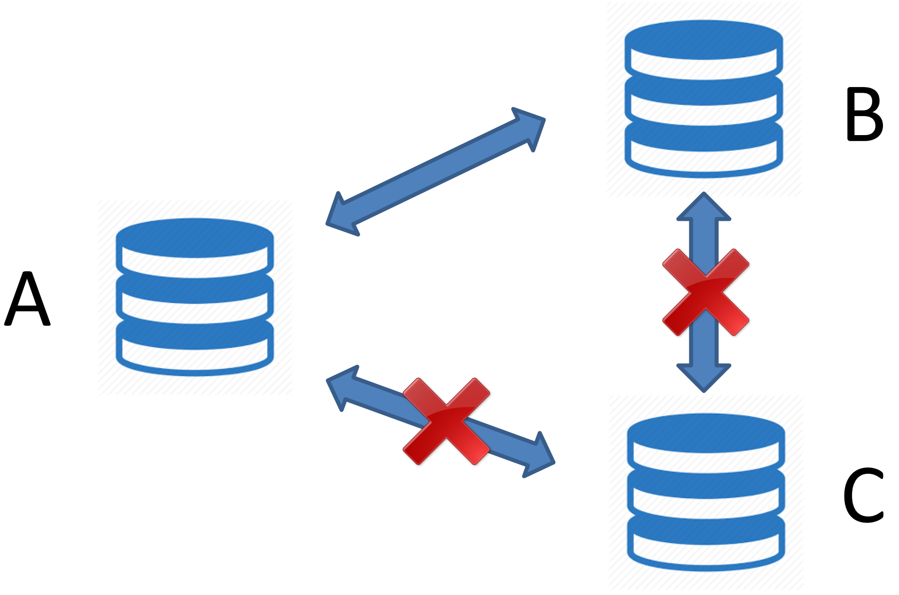
{A, B, C}



Partition tolerance

- ***Network partitioning***: the network is disconnected due to a network failure. Different nodes are distributed in different subnetworks, and the network within each subnetwork is normal.

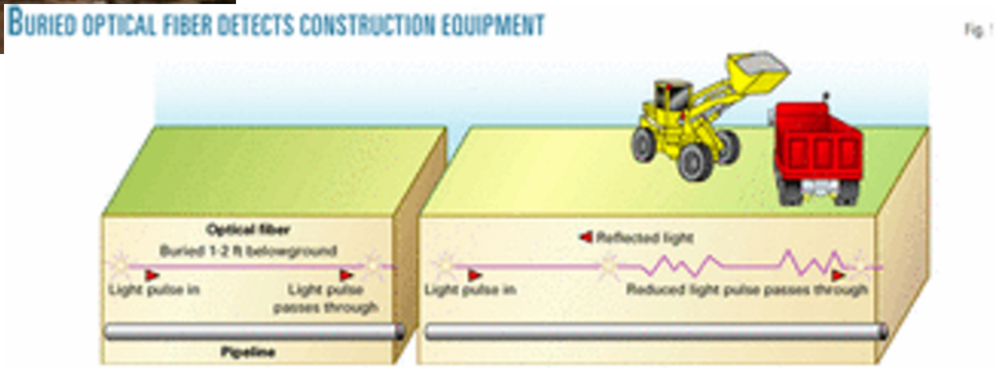
{A, B}
{C}



Network partition

Chris Willis/US Air Force

broken optical fiber

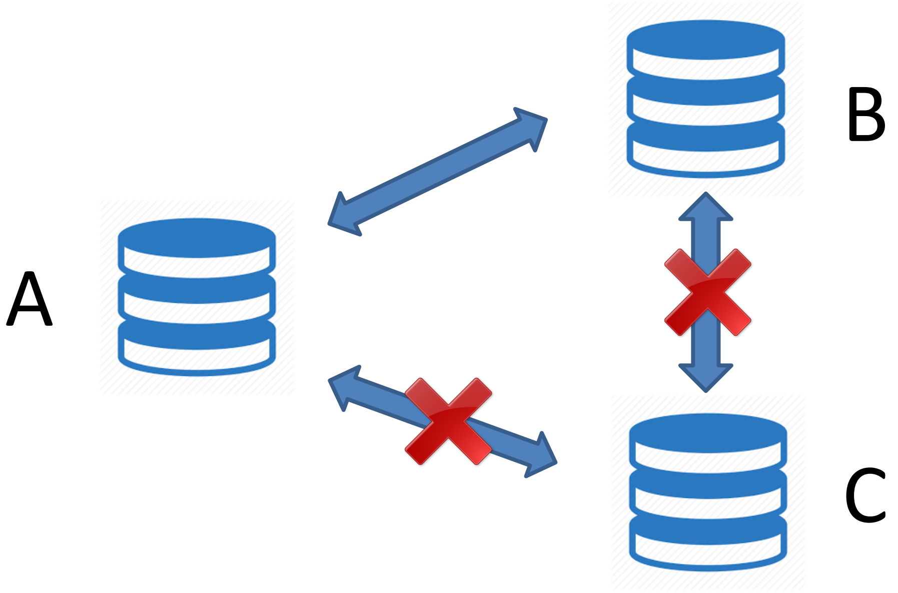


BURIED OPTICAL FIBER DETECTS CONSTRUCTION EQUIPMENT
Fig. 1
Optical fiber
Buried 1-2 ft belowground
Light pulse in
Light pulse passes through
Pipeline
Reflected light
Light pulse in
Reduced light pulse passes through

Partition tolerance

- Distributed systems can still respond to user requests when they encounter network partitions

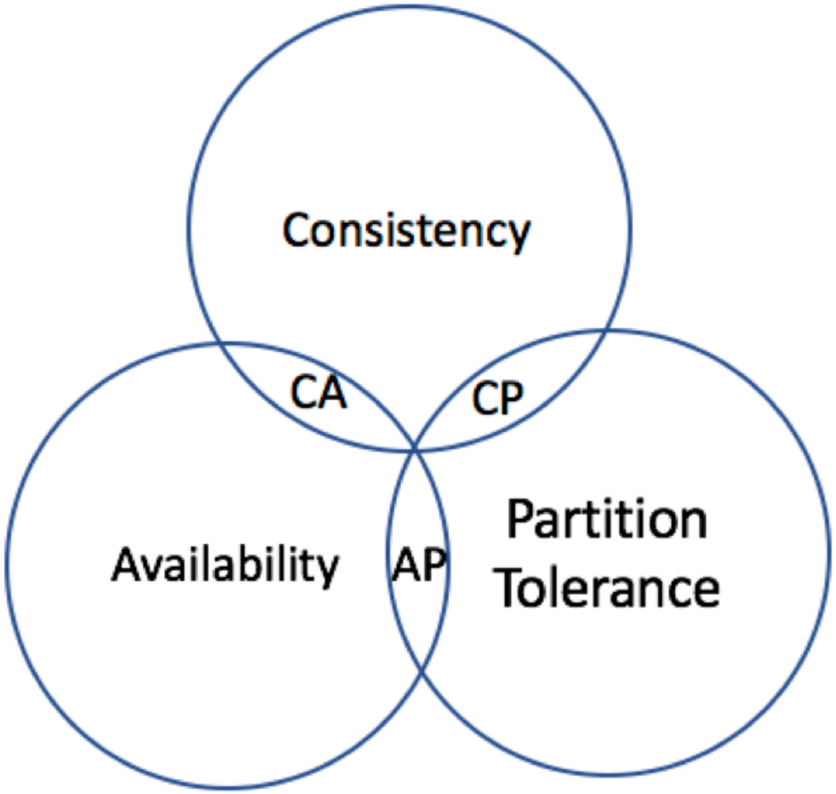
{A, B}
{C}



CAP theorem

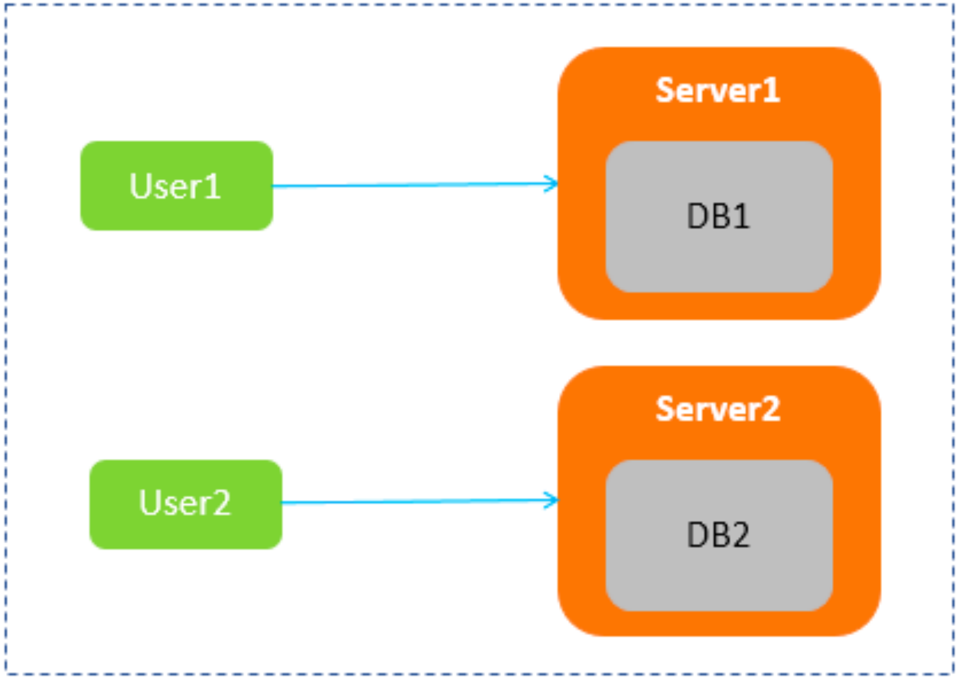
- You cannot realize C, A and P at the same time

- One time you can only have ***at most two*** of them



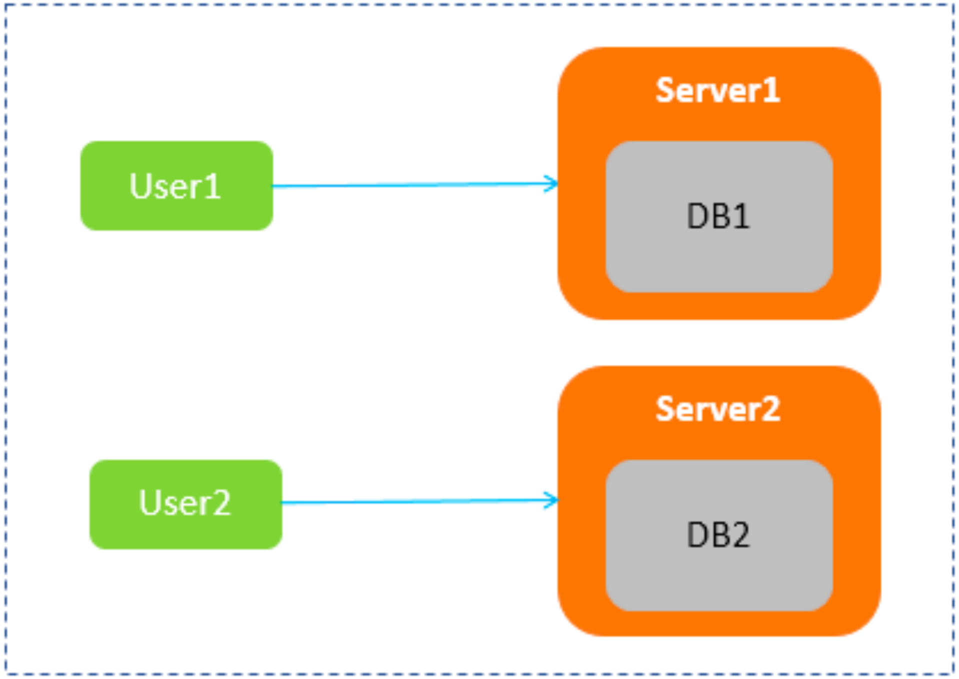
Example: what is CAP and why at most two of CAP exist

- A distributed system has two servers
- DB_{i} runs on server_{i}
- User1 sends requests to Server1
- User2 sends requests to Server2



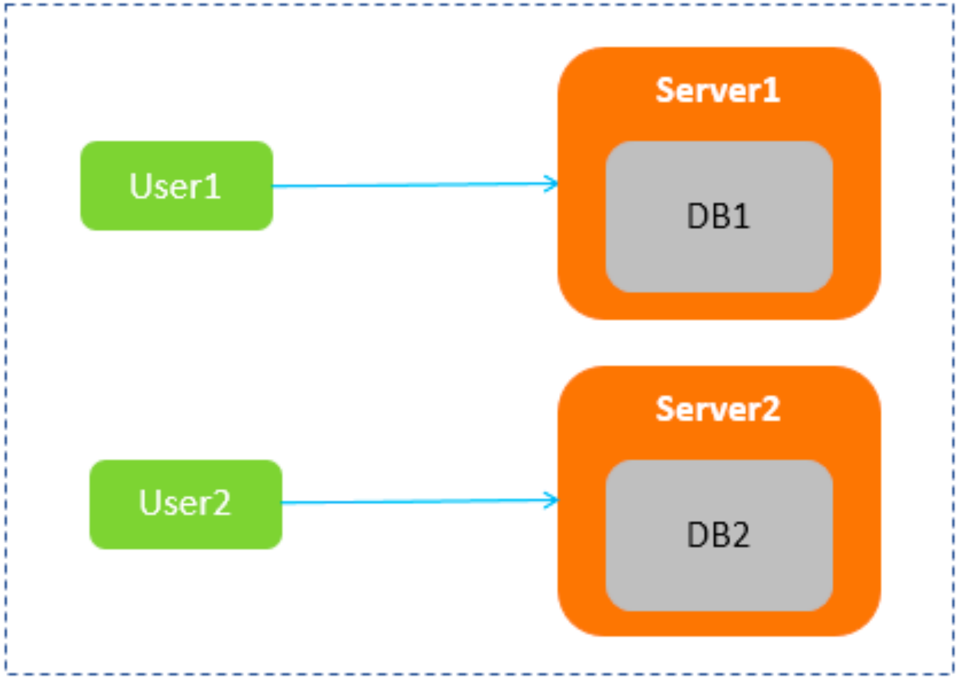
Example: What is CAP and why at most two of CAP exist

- What is C (consistency) in this system?
  - The databases in Server1 and Server2 are always consistent --> the contents of DB1 and DB2 must always be the same



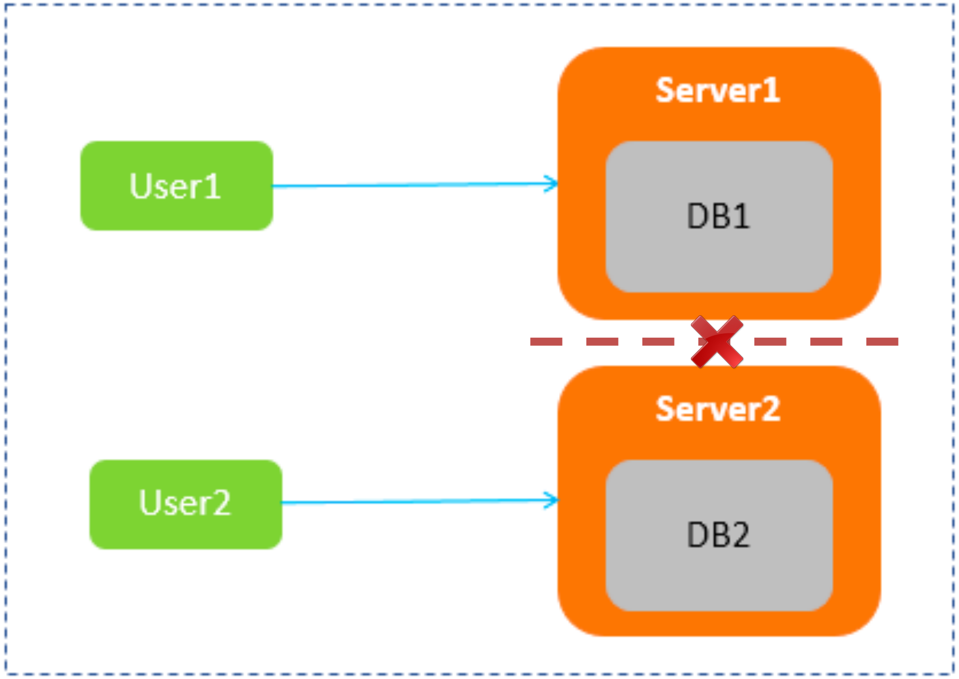
Example: What is CAP and why at most two of CAP exist

- What is A (availability) in this system?
  - Users get instant response no matter they access Server1 or Server2



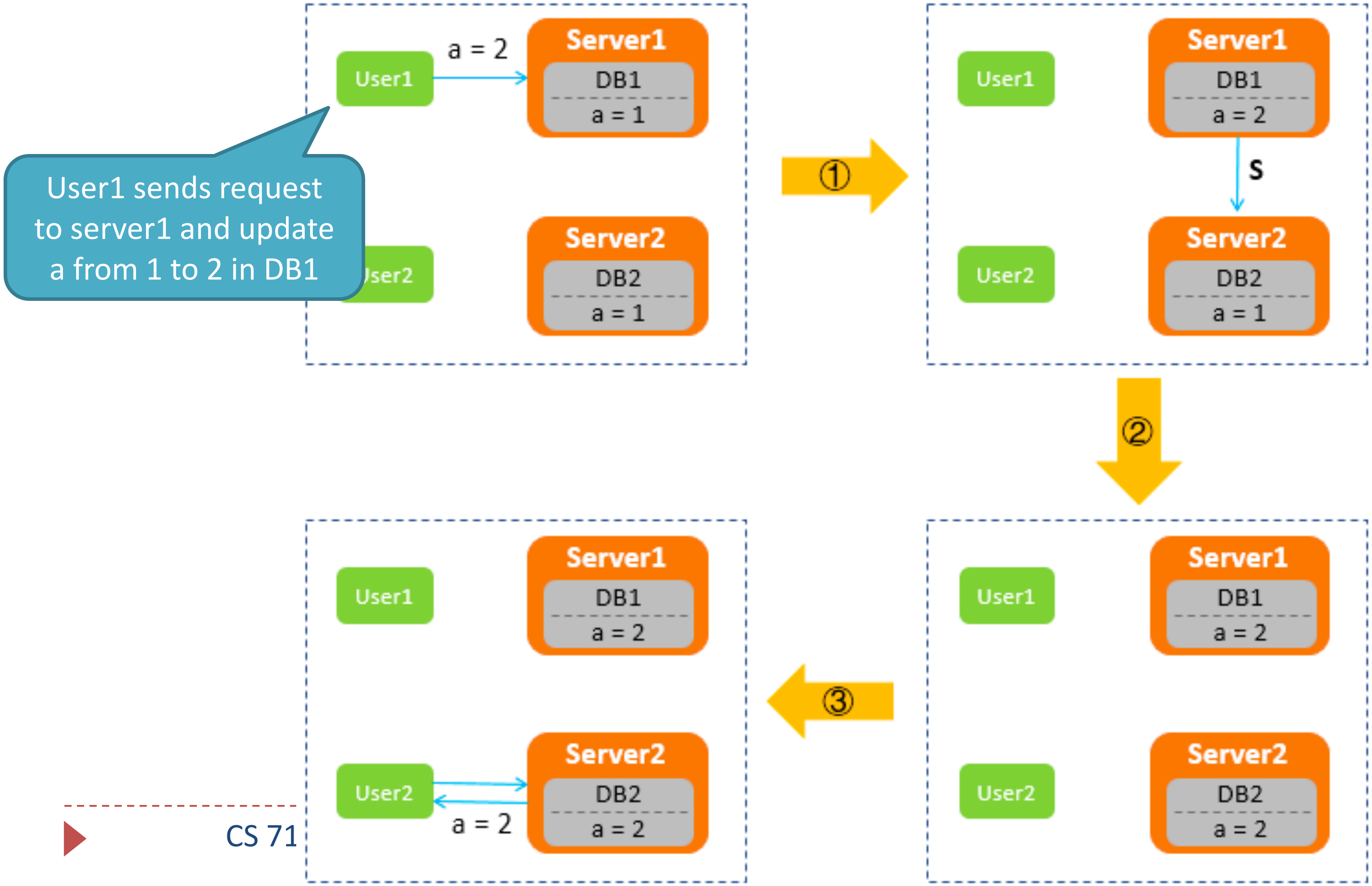
Example: What is CAP and why at most two of CAP exist

- What is P (partition tolerance) in this system?
  - Even if a network failure occurs between Server1 and Server2, it will not affect Server1 and Server2 to process user requests



Example: Why only at most two of CAP exist

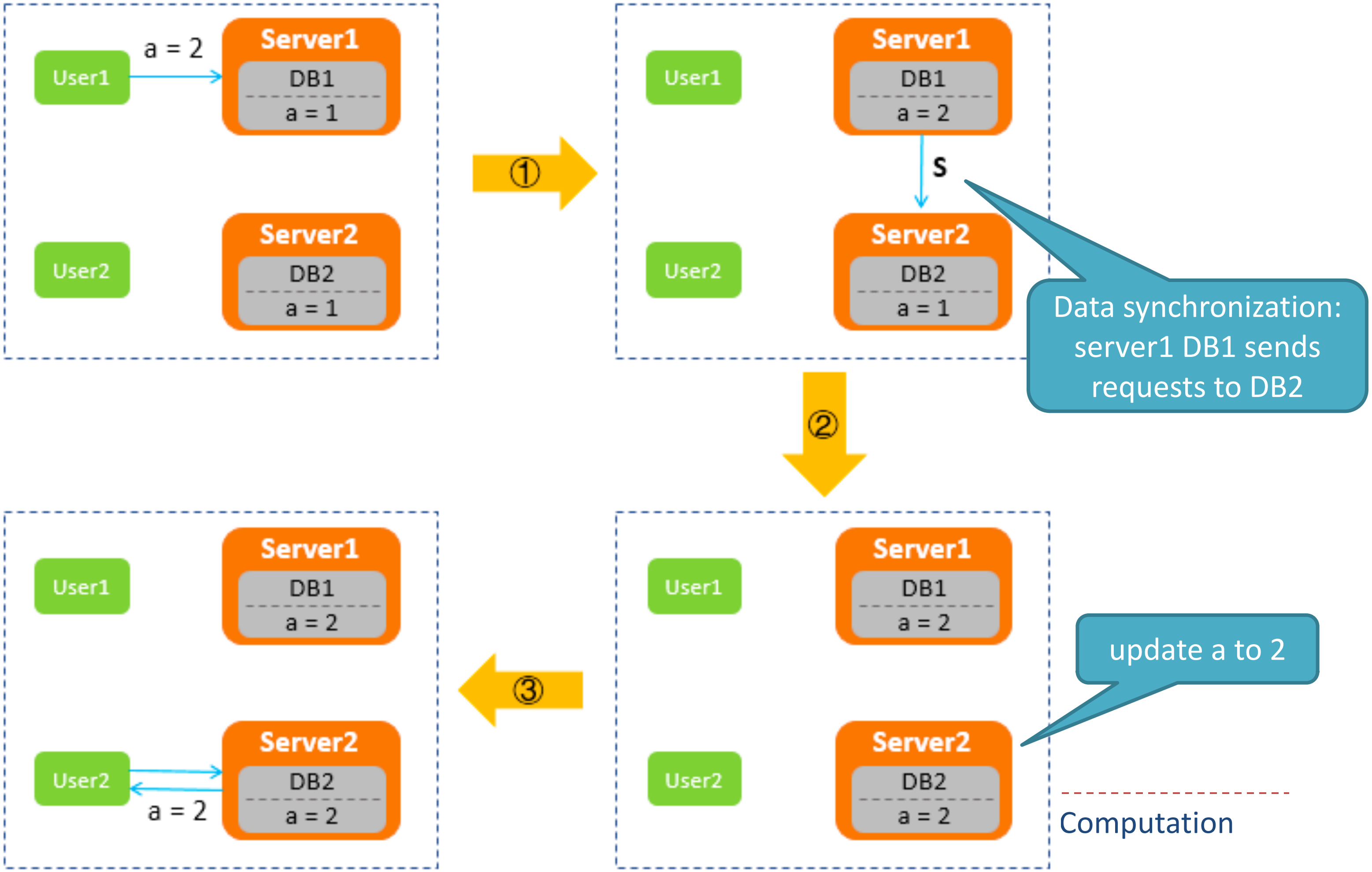
- A normal case: everything works well



User1 sends request to server1 and update a from 1 to 2 in DB1

Example: Why only at most two of CAP exist

- A normal case: everything works well

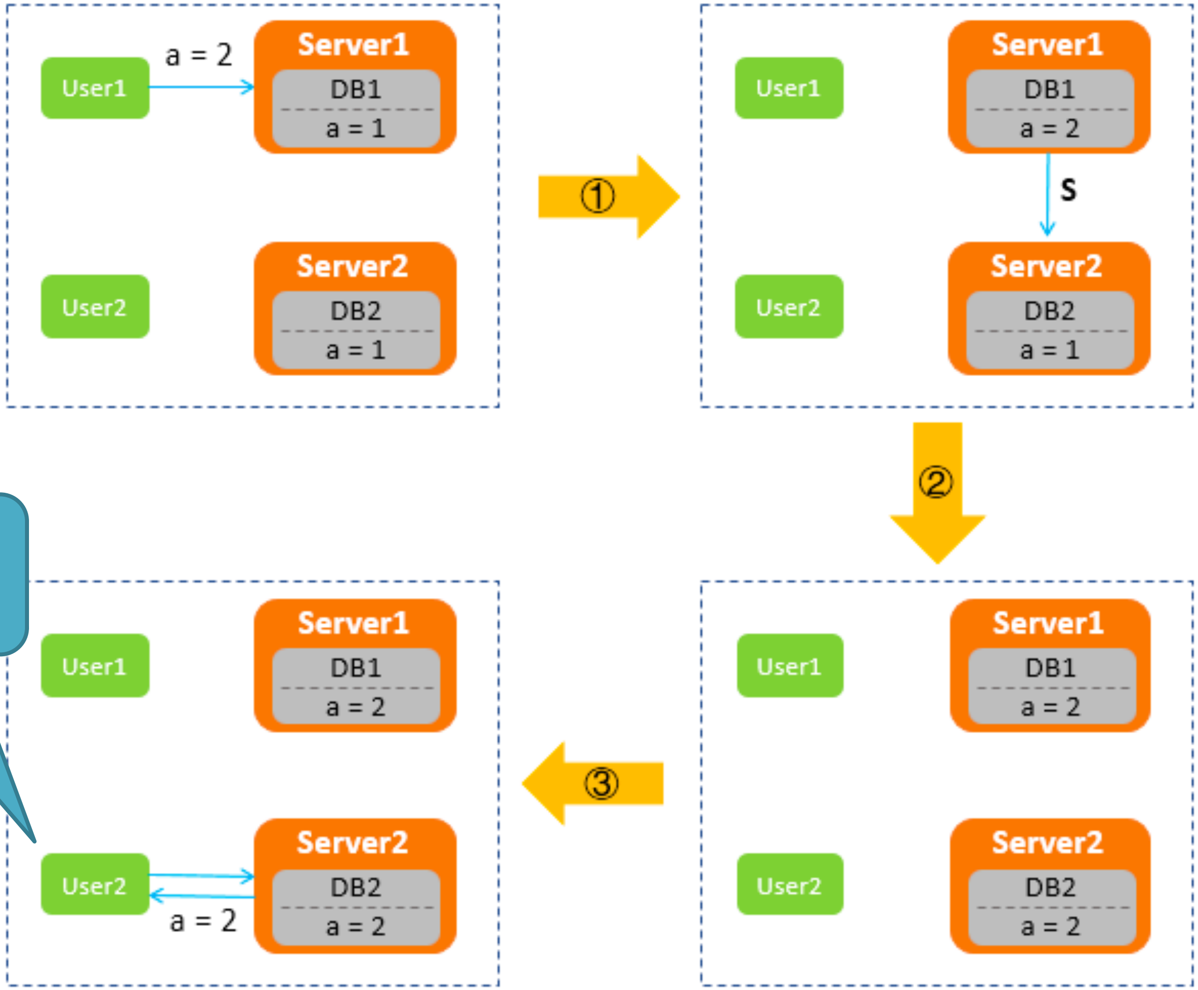


Data synchronization: server1 DB1 sends requests to DB2

update a to 2

Example: Why only at most two of CAP exist

- A normal case: everything works well



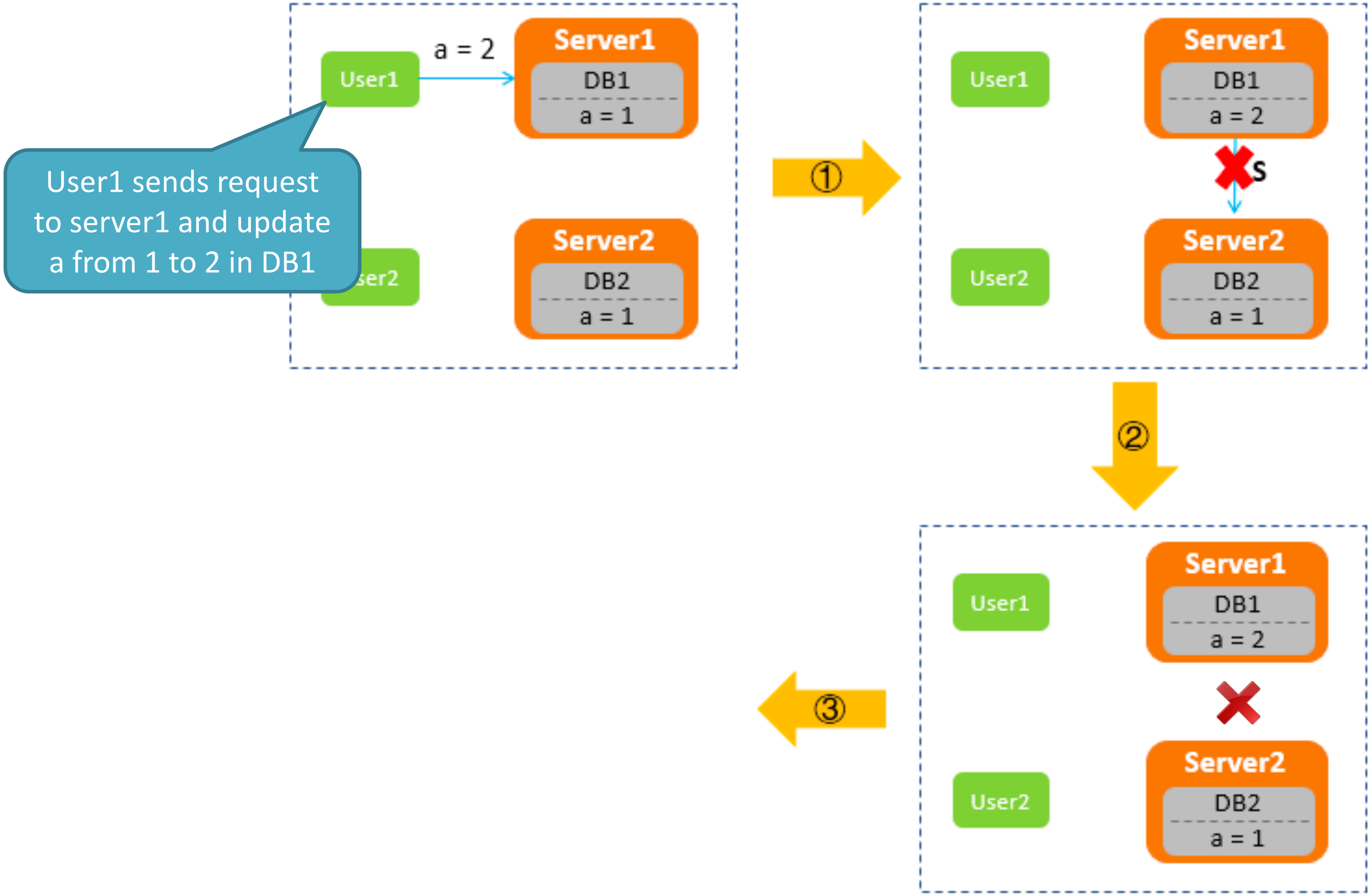
User2 sends request to server2 and gets a which is 2

Example: Why only at most two of CAP exist

- A normal case: everything works well --> Workflow in a stable network environment and no system failure

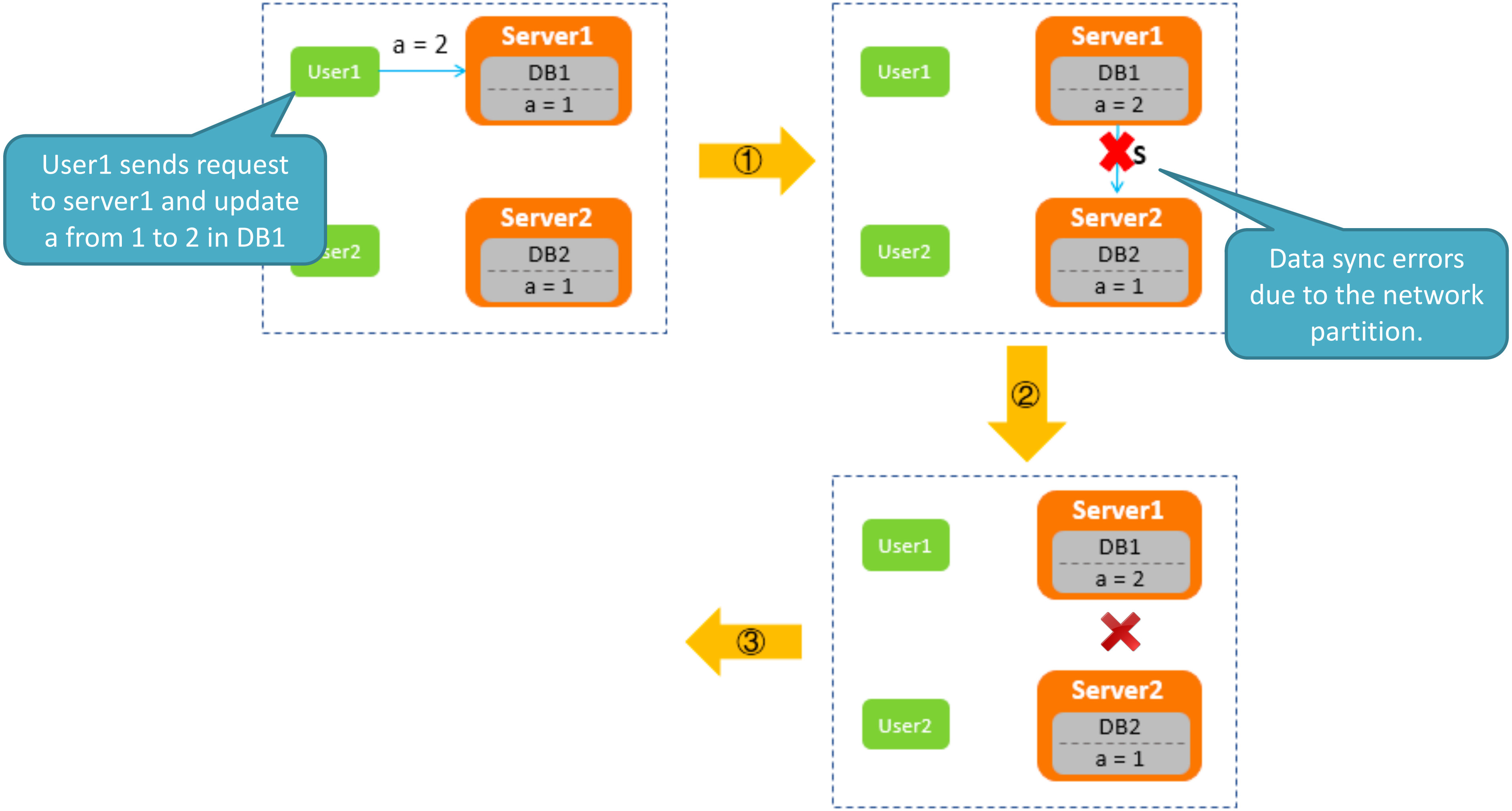
- Reality: network failure always happens, such as network congestion, network hardware broken, etc.
  - → network partitions in distributed system is inevitable
  - → P(partition tolerance) must be guaranteed
  - → Question: Can we satisfy C and A at the same time?

Example: Why only at most two of CAP exist



User1 sends request to server1 and update a from 1 to 2 in DB1

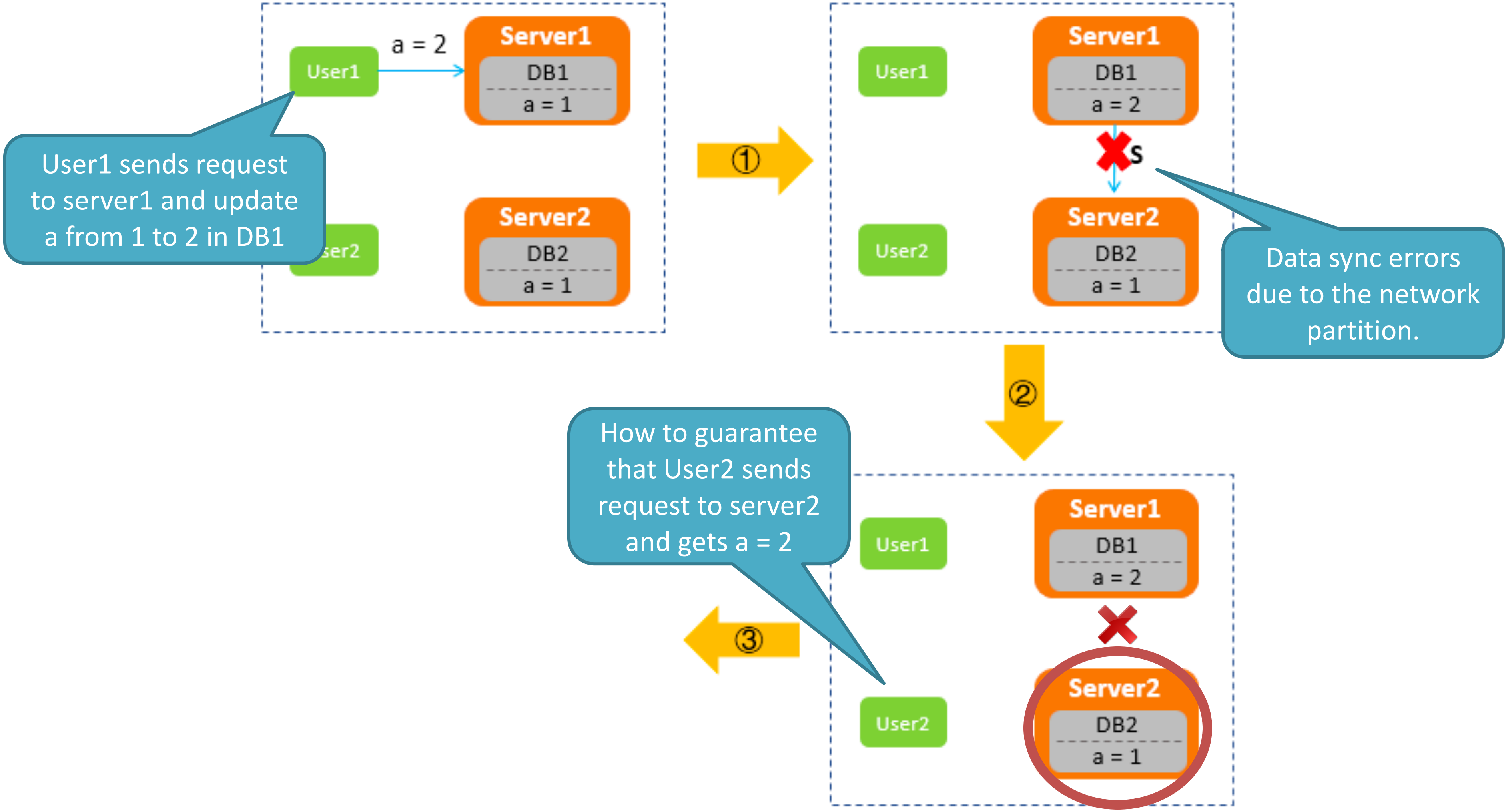
Example: Why only at most two of CAP exist



User1 sends request to server1 and update a from 1 to 2 in DB1

Data sync errors due to the network partition.

Example: Why only at most two of CAP exist

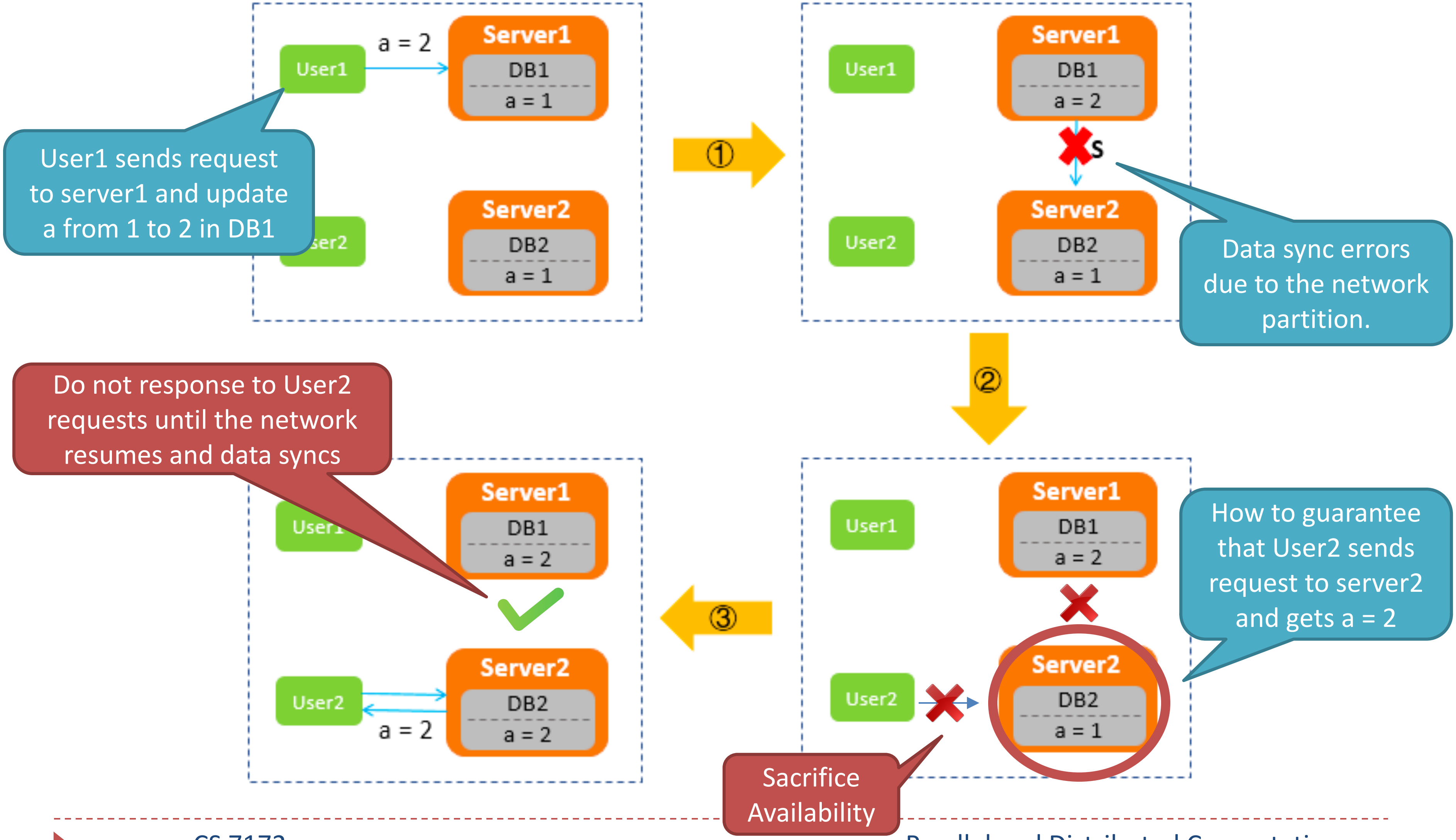


User1 sends request to server1 and update a from 1 to 2 in DB1

Data sync errors due to the network partition.

How to guarantee that User2 sends request to server2 and gets a = 2

Option 1: guarantee Consistency and sacrifice Availability



User1 sends request to server1 and update a from 1 to 2 in DB1

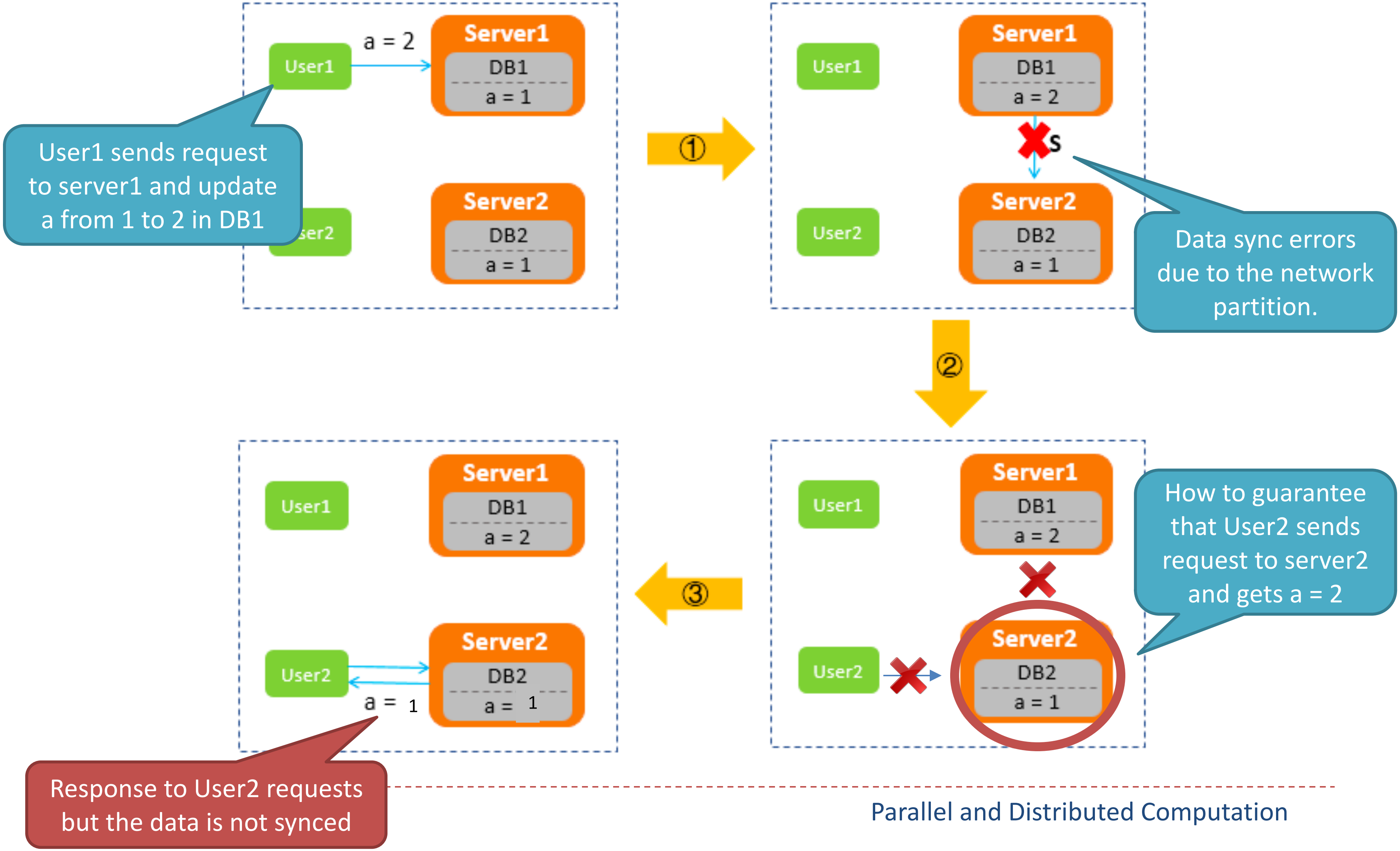
Data sync errors due to the network partition.

How to guarantee that User2 sends request to server2 and gets a = 2

Sacrifice Availability

Do not response to User2 requests until the network resumes and data syncs

Option 2: guarantee Availability and sacrifice Consistency and



User1 sends request to server1 and update a from 1 to 2 in DB1

Data sync errors due to the network partition.

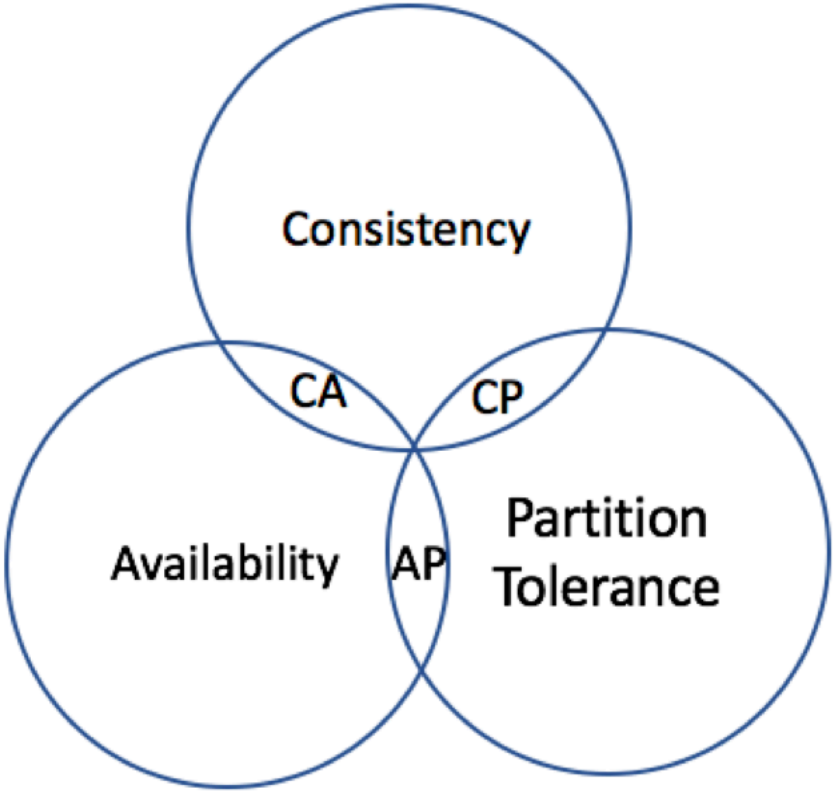
How to guarantee that User2 sends request to server2 and gets a = 2

Response to User2 requests but the data is not synced

CAP theorem

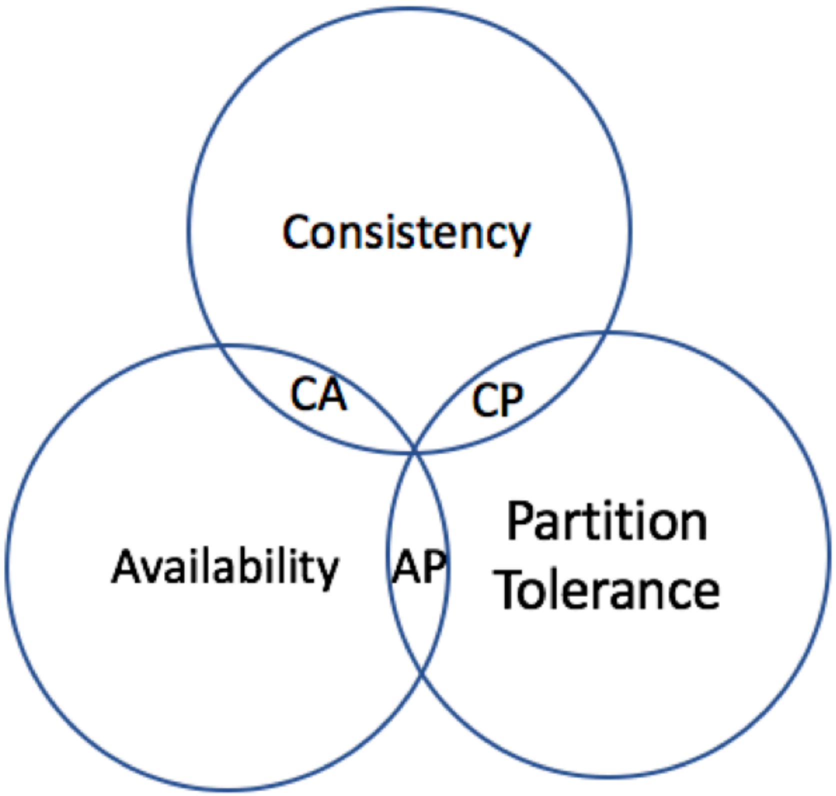
- You cannot realize C, A and P at the same time

- One time you can only have ***at most two*** of them



CAP theorem

- When to choose CA?
- When to choose AP?
- When to choose CP?
- CA/AP/CP works for different scenarios and no one is better than others



Guarantee CA, Sacrifice P

- In distributed systems, the current network infrastructure cannot always be stable, and network partitions (network disconnection) are inevitable

- We cannot sacrifice partition tolerance in distributed systems

- If all services are in a single machine, such as MySQL/Oracle on one server, we do not need to consider partition tolerance and can guarantee consistency and availability

Guarantee CP, Sacrifice A

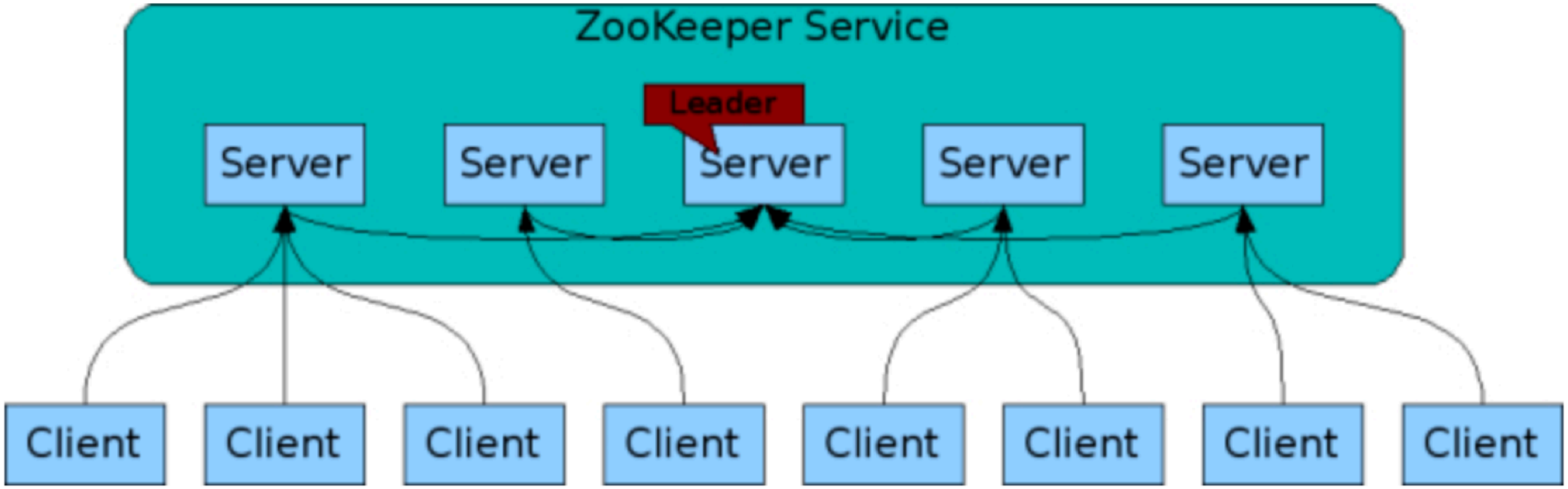
- If a distributed scenario requires ***strong data consistency***, or the scenario ***can tolerate*** long periods of system non-response

- Usually it is used in distributed scenarios involving financial transactions, because it does not allow data inconsistencies at any time, otherwise it will cause losses to users

Systems to guarantee CP

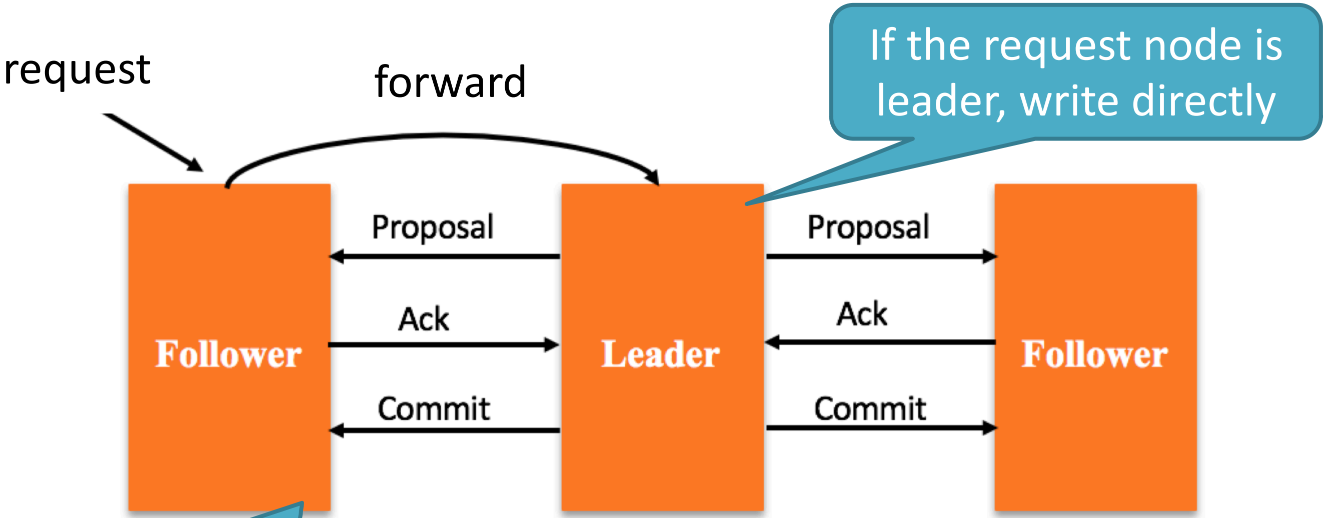
- Redis, Hbase, ZooKeeper

- ZooKeeper architecture: one leader, multiple follower servers



Systems to guarantee CP

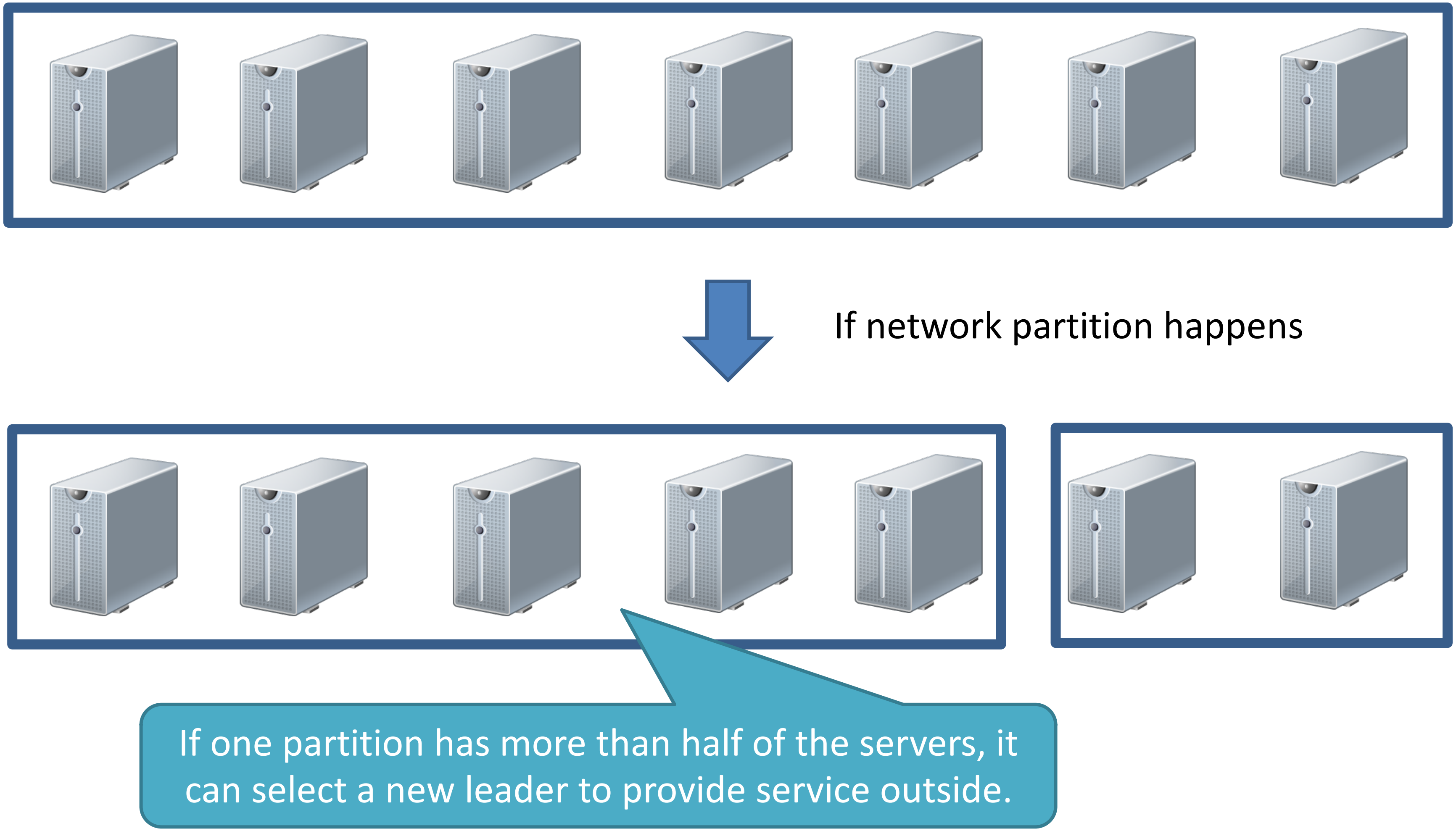
- Let's consider write request



If the request node is leader, write directly

If the request node is follower, it will forward requests to leader node. Leader sends proposal to all followers. If more than half nodes agree, executes the "write" to guarantee the strong consistency,

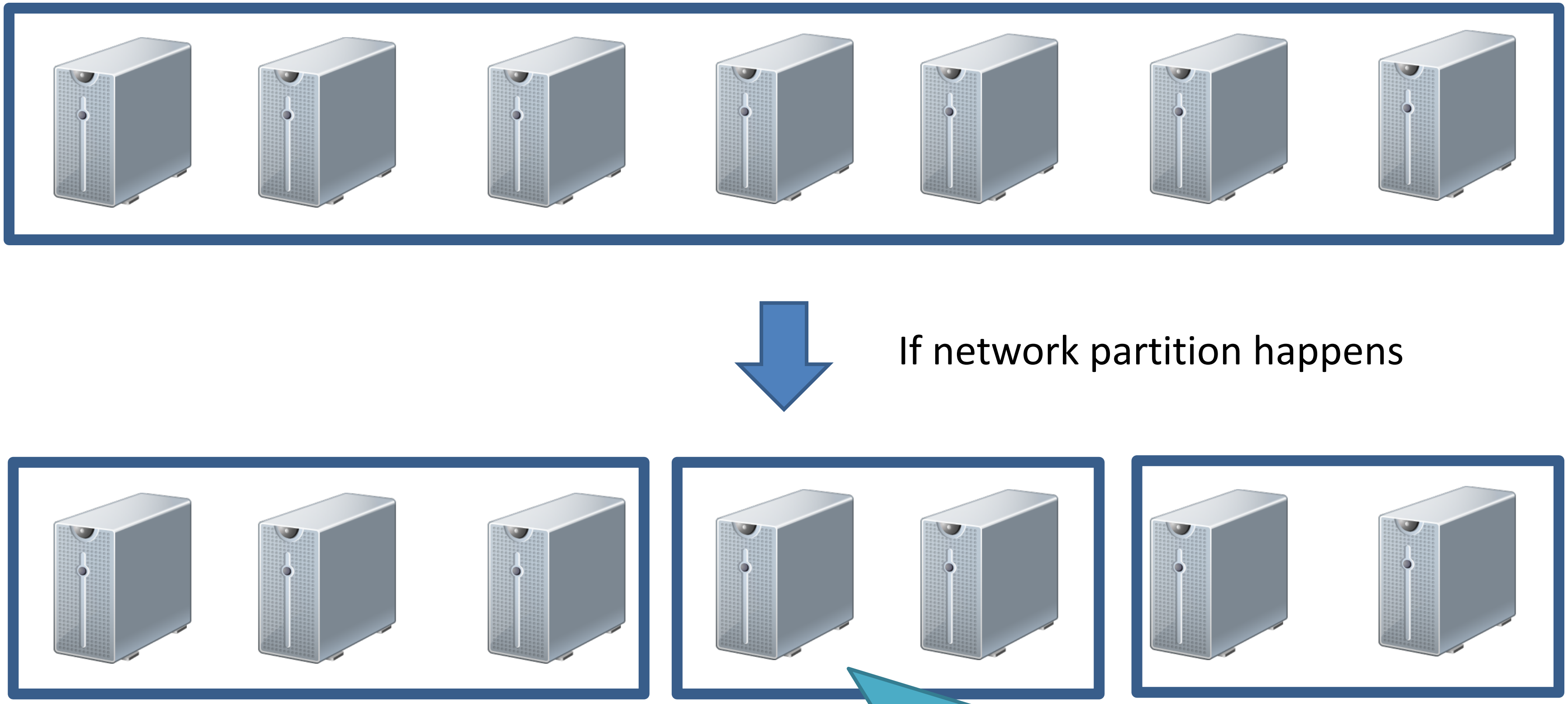
Systems to guarantee CP



If network partition happens

If one partition has more than half of the servers, it can select a new leader to provide service outside.

Systems to guarantee CP



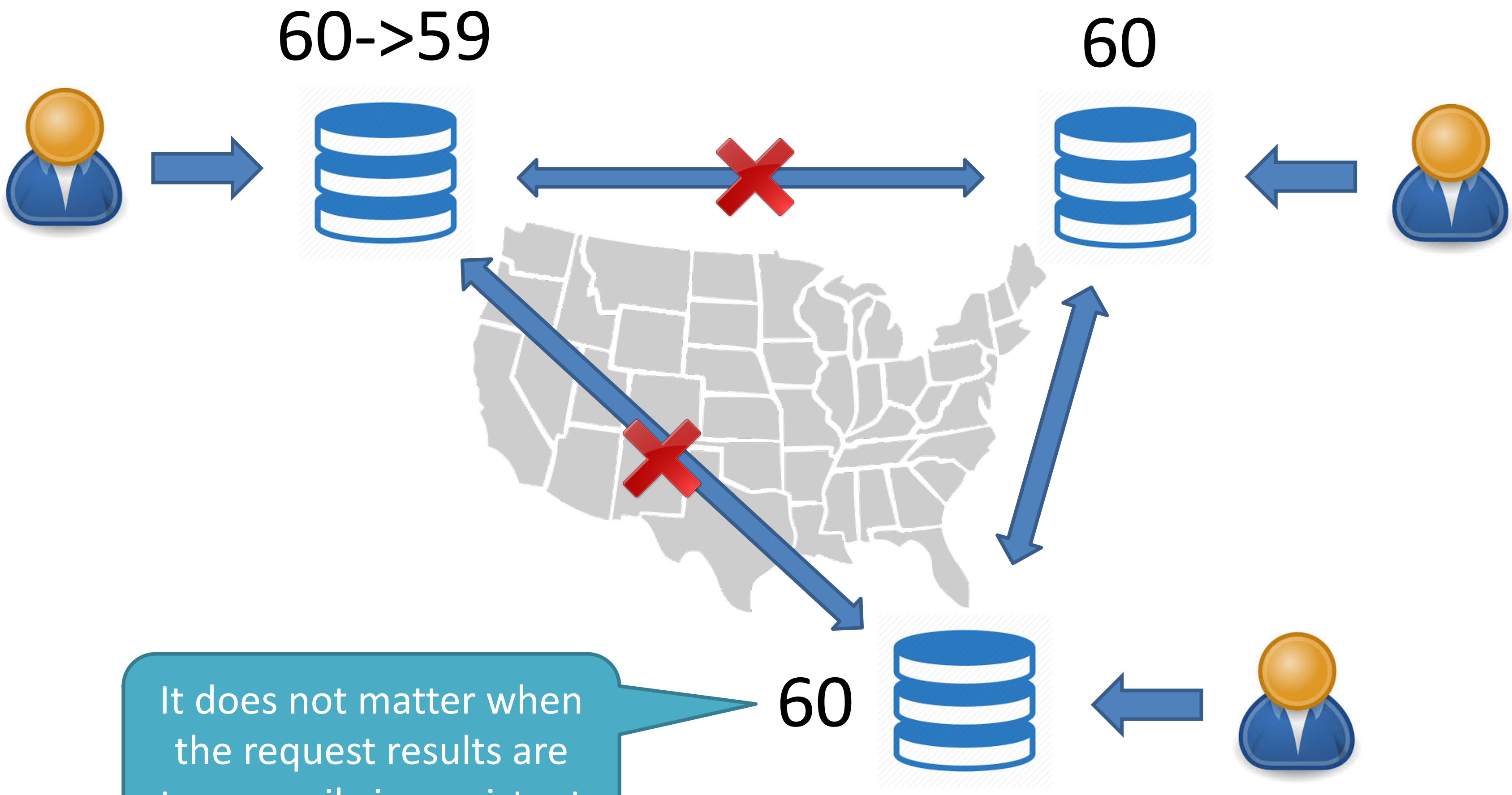
If no partition has more than half of the servers, sacrifice the availability until the network connection resumes.

Guarantee AP, Sacrifice C

- If a distributed scenario requires high availability, it allows data to be ***temporarily inconsistent*** to provide better user experience

- Examples: many product queries in e-commerce systems, etc., the user experience is very important, so most of them will ensure the availability and sacrifice certain data consistency

Guarantee AP, Sacrifice C



It does not matter when the request results are temporarily inconsistent

Comparison

	Guarantee CA, Sacrifice P	Guarantee CP, Sacrifice A	Guarantee AP, Sacrifice C
Feature	Consistent Availability	Consistent Partition tolerance	Availability Partition tolerance
Scenario	Single server	Strong consistent requirement such as banks	High requirements to low user request latency
Example	MySQL Oracle	Redis Hbase ZooKeeper	Cassandra DynamoDB Eureka CoachDB