Scaling Storage

Slides by Andrew DeOrio

Scaling road map

- Last last time: scaling static pages
- Last time: scaling server-side dynamic pages
  - Round robin DNS, load balancing, hardware virtualization, containerization
- Today: scaling storage
  - Database
  - Media uploads

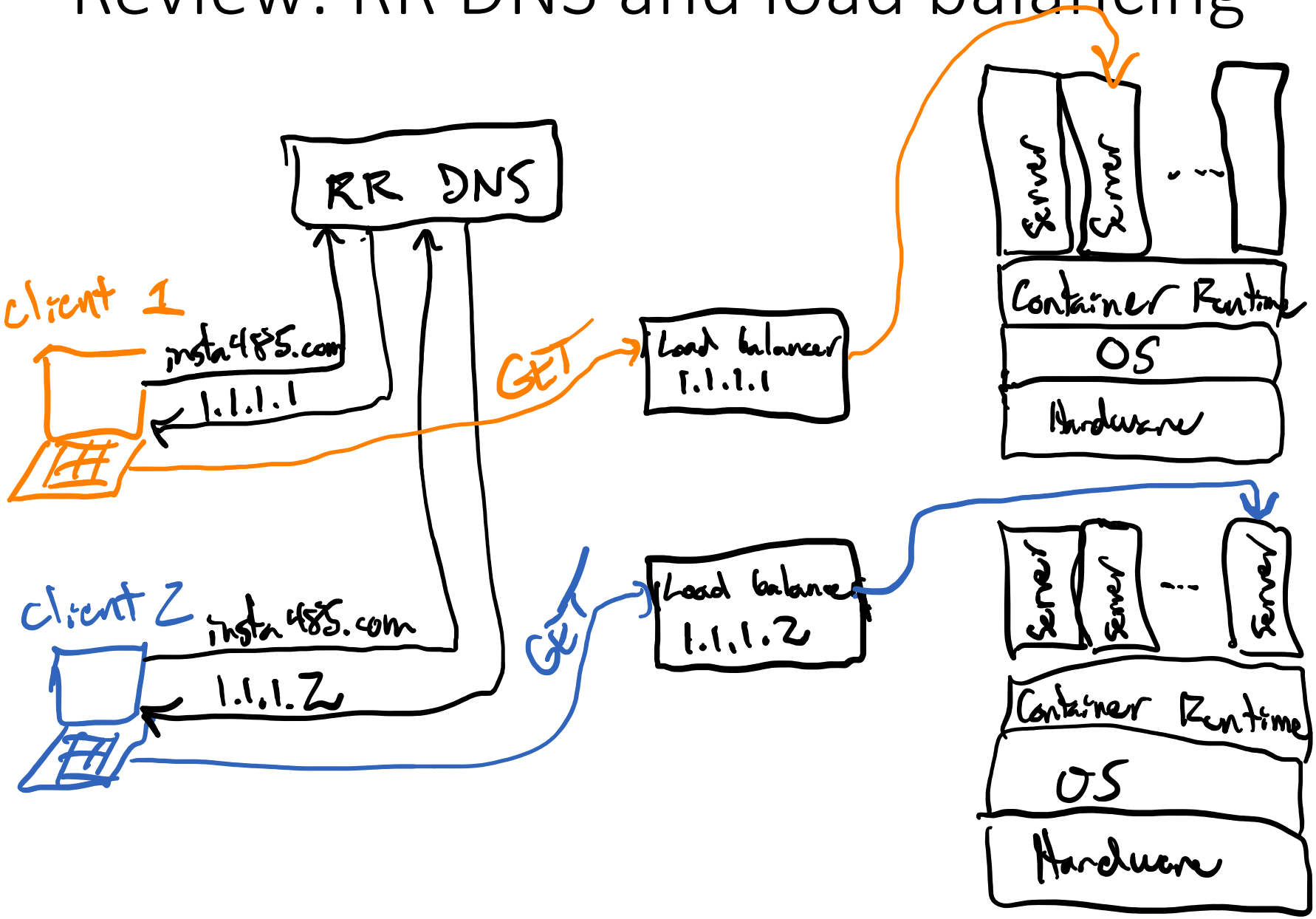
Agenda

- Distributed network database
- Sharding by content, database replication
- CAP Theorem
- Practical database examples
- Distributed file system for media uploads

Review: Scaling dynamic pages

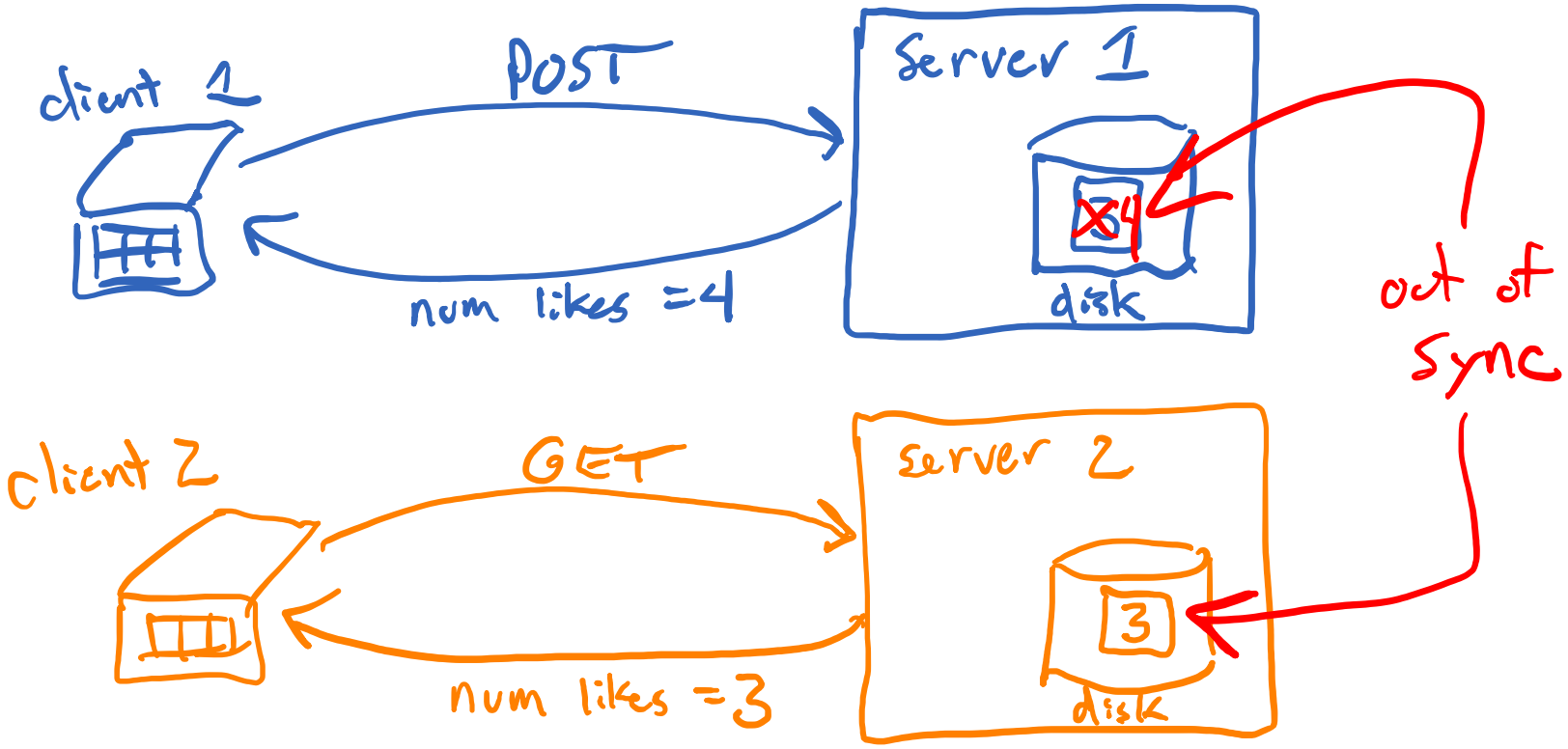
- Dynamic pages servers run in containers
  - AKA "Front end server" AKA "FE"
- Requests routed to servers via round robin DNS and load balancing

Review: RR DNS and load balancing



Problem with many servers

- 2 clients connect to 2 servers and *access the same data*
  - E.g., one likes a post, and the other loads the same post



Many dynamic pages servers

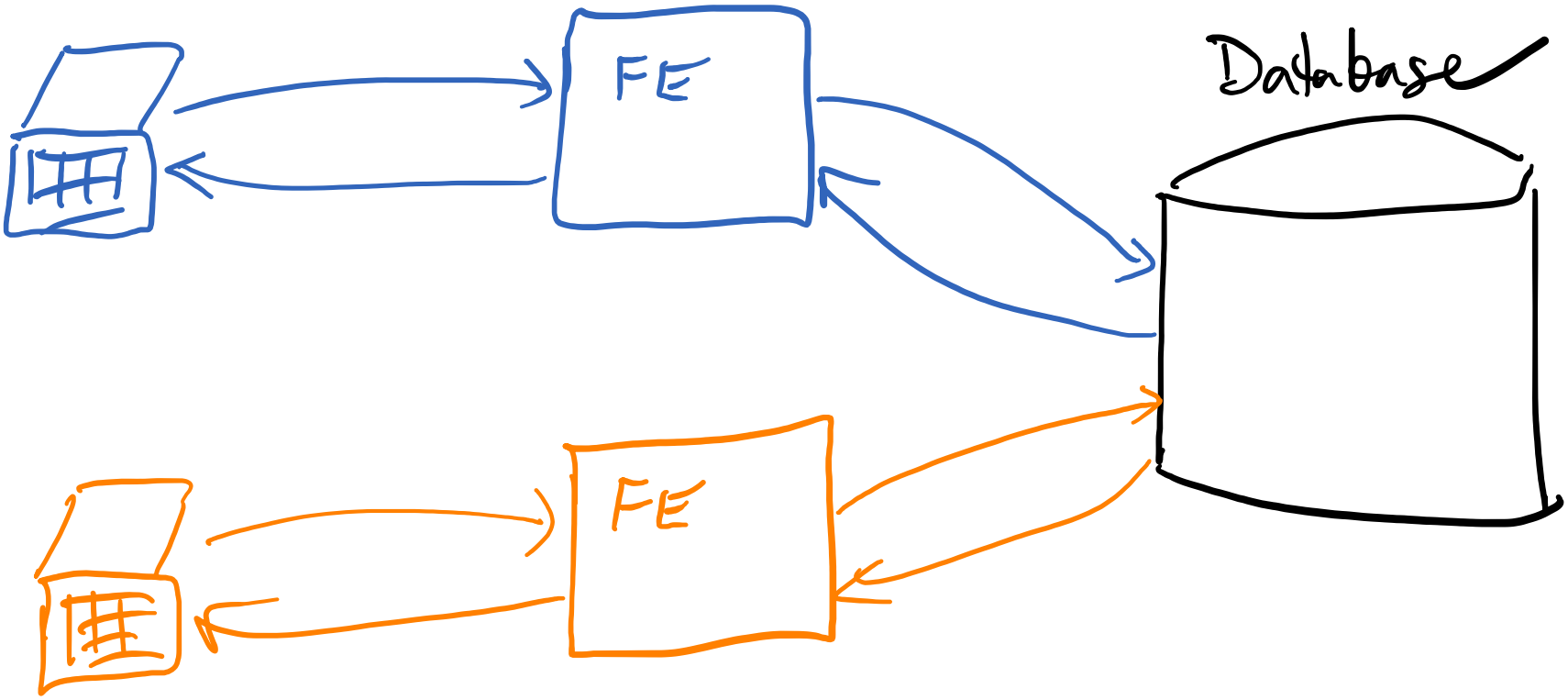
- Different dynamic pages servers on every request
  - Physical machines, virtual machines, or containers
  - Sometimes called "Front end servers"
- Problem: How to synchronize dynamic pages servers?
- Solution: Dynamic pages servers are stateless, store everything in the database

Network database

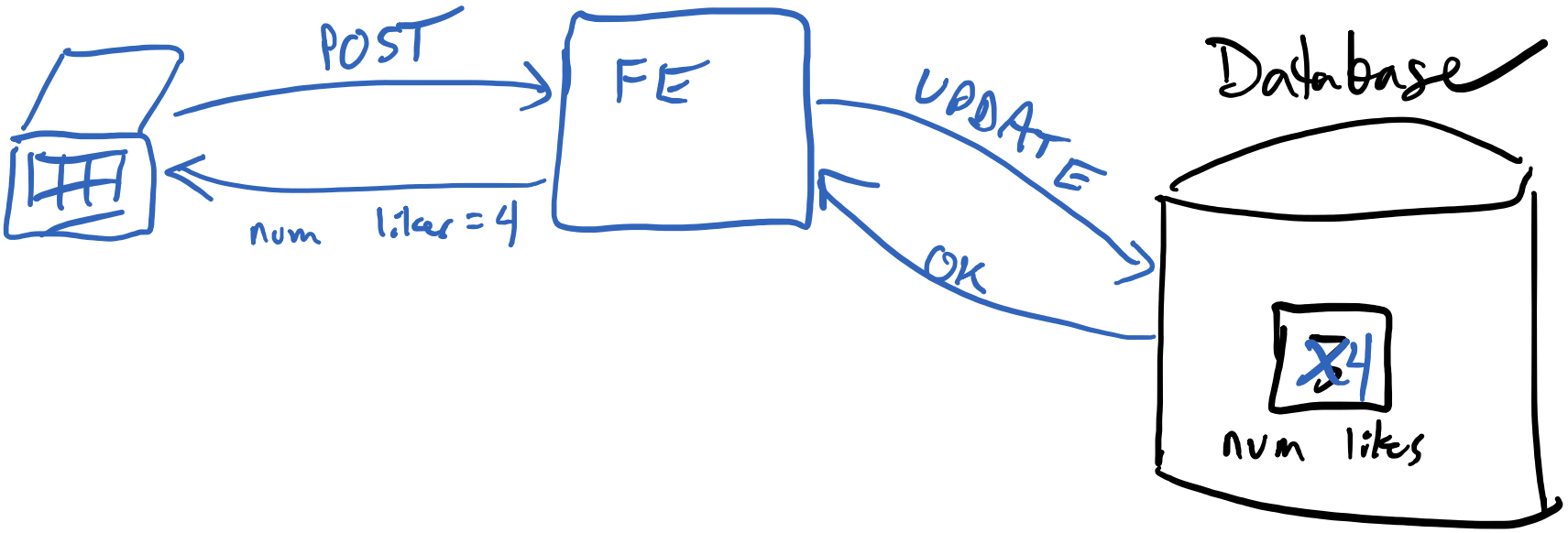
- Many dynamic pages servers make network requests to one network database
- *Network database*: Database software runs on a different server on the network
  - PostgreSQL, MySQL, et al.
- *Centralized network database*: One server

Centralized network database

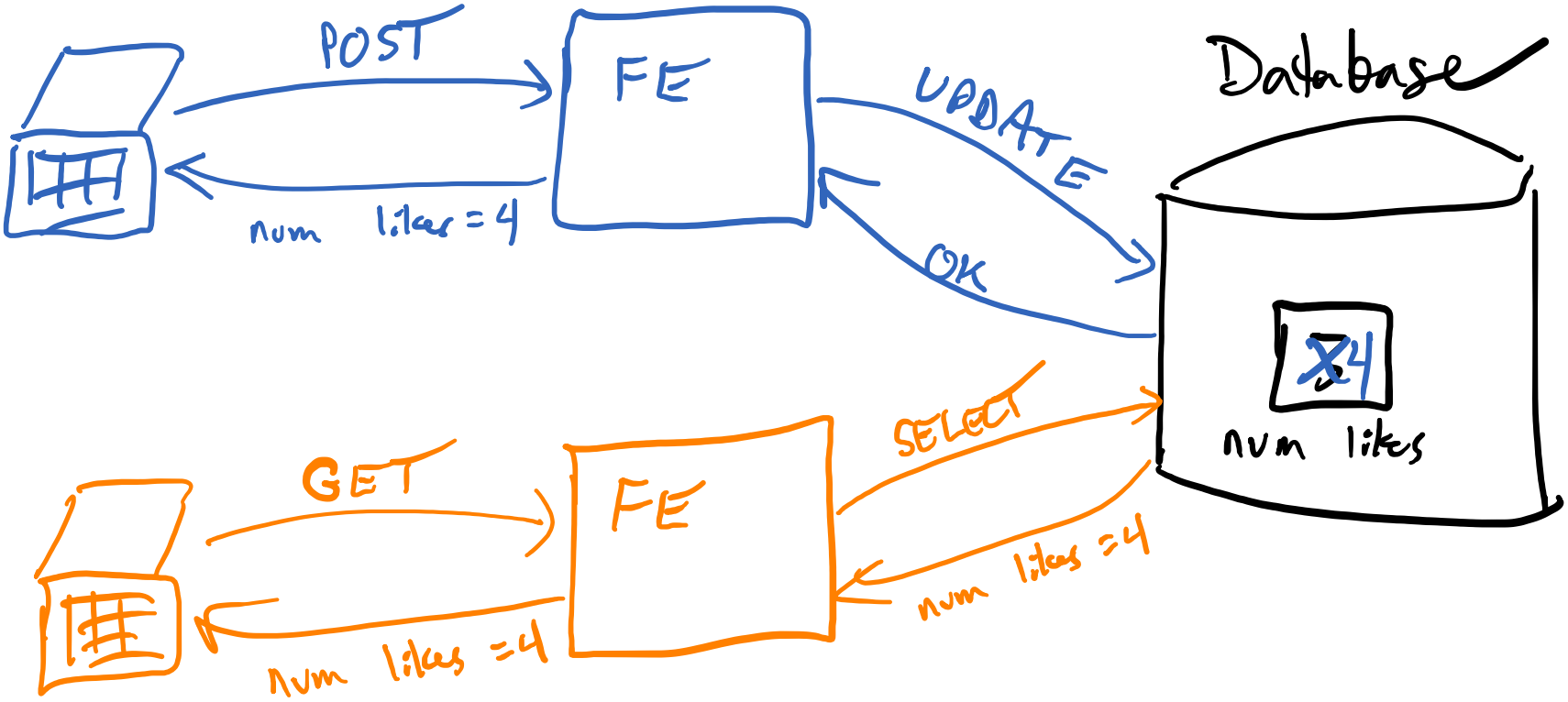
- *Centralized network database:* Single network database server



Centralized network database write



Centralized network database read



Centralized network database

- Data consistency is easy
- Problem: Performance bottleneck
- Solution: More database servers

Distributed network database

- *Distributed network database:* Many network database servers
- Problem: How to keep database servers in sync?
- Solution: Data consistency is hard, depends on which servers contain which data

- Problem: Which servers contain which data?
- Solution 1: Sharding by content
- Solution 2: Database replication

Agenda

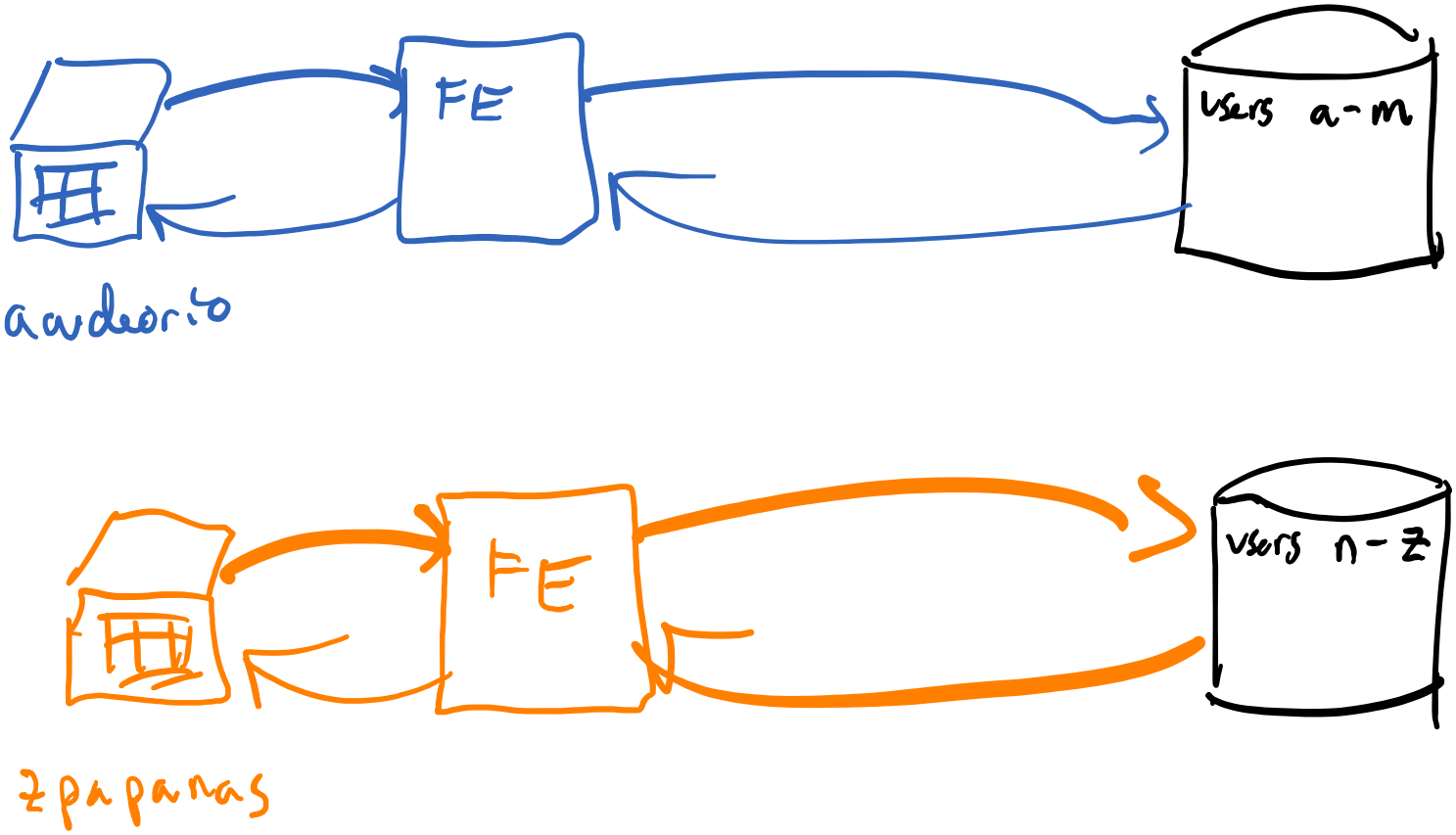
- Distributed network database
- **Sharding by content, database replication**
- CAP Theorem
- Practical database examples
- Distributed file system for media uploads

Sharding and replication

- Problem: Which servers contain which data?
- Solution 1: Sharding by content
- Solution 2: Database replication

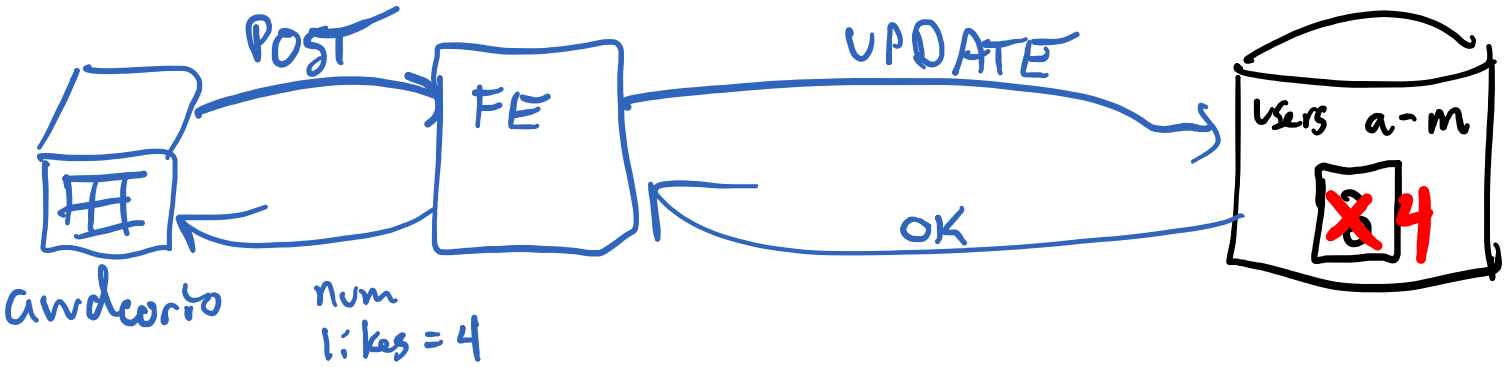
Sharding by content

- Different rows or tables on different DB servers



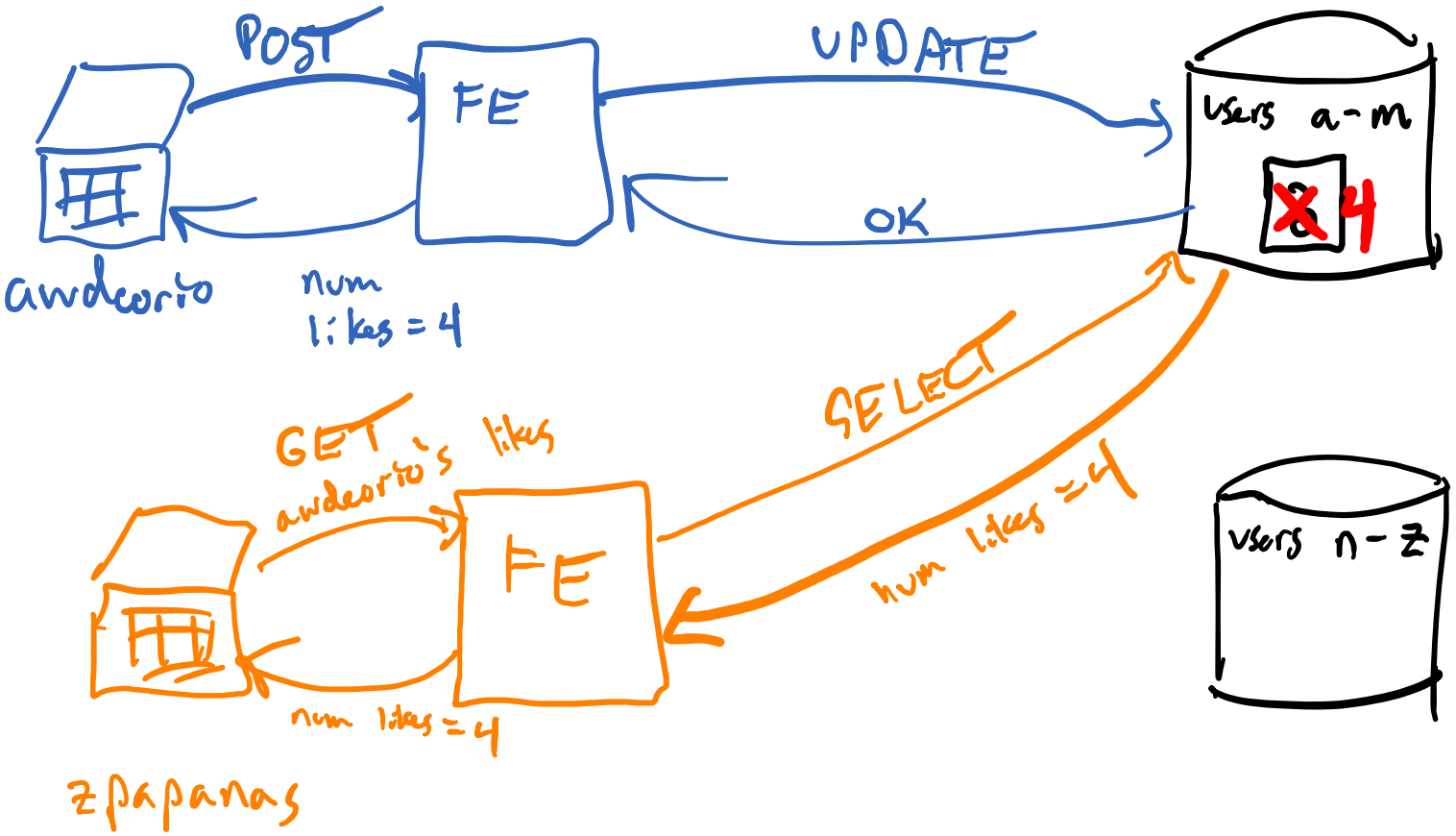
Sharding by content write

- A write modifies the *only copy*



Sharding by content read

- A read accesses the *only copy*

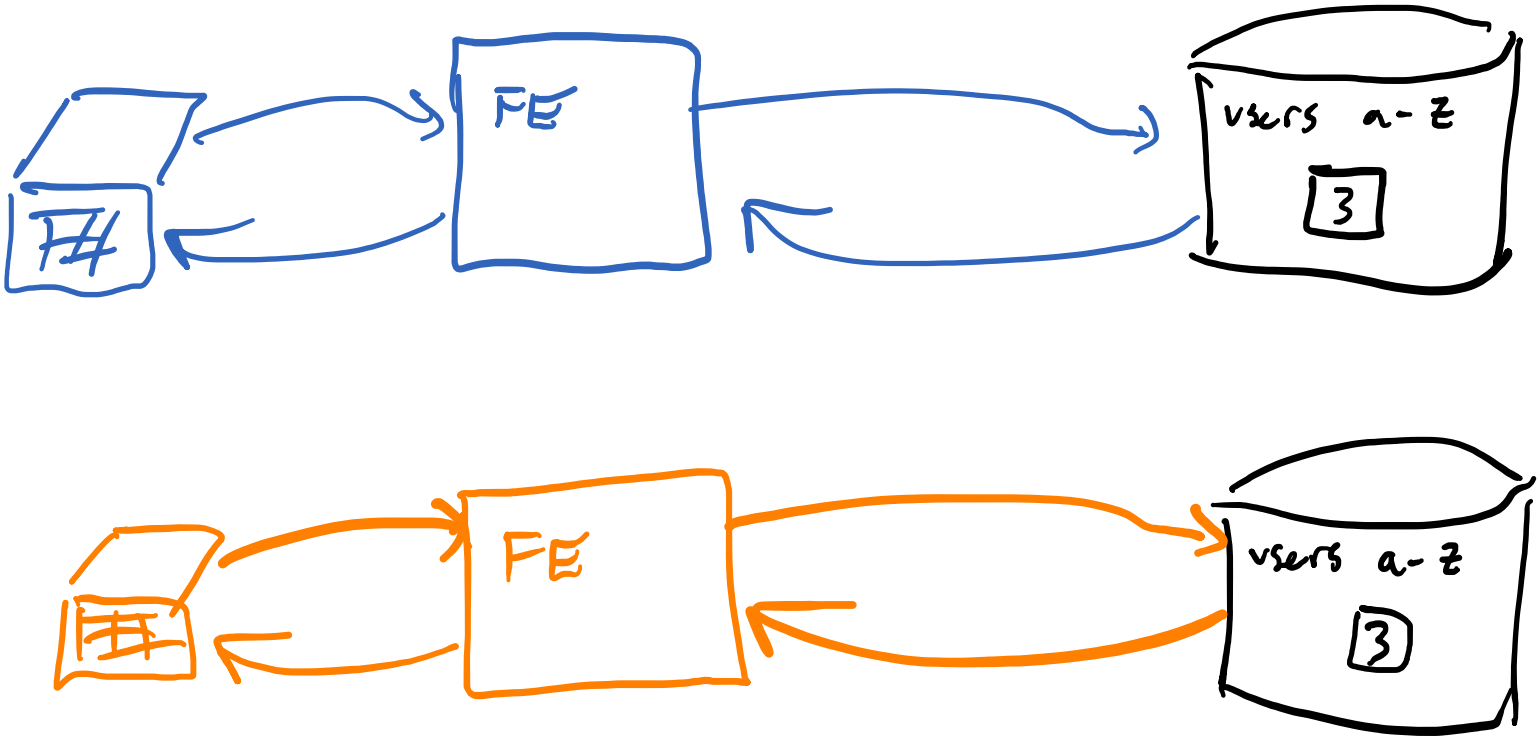


Sharding by content pros and cons

- Different rows or tables on different DB servers
- Strengths
  - Scalable because data is split among servers
  - Consistency is easier because no copies
- Weaknesses
  - Throughput: Single copy could be a bottleneck for frequently accessed data

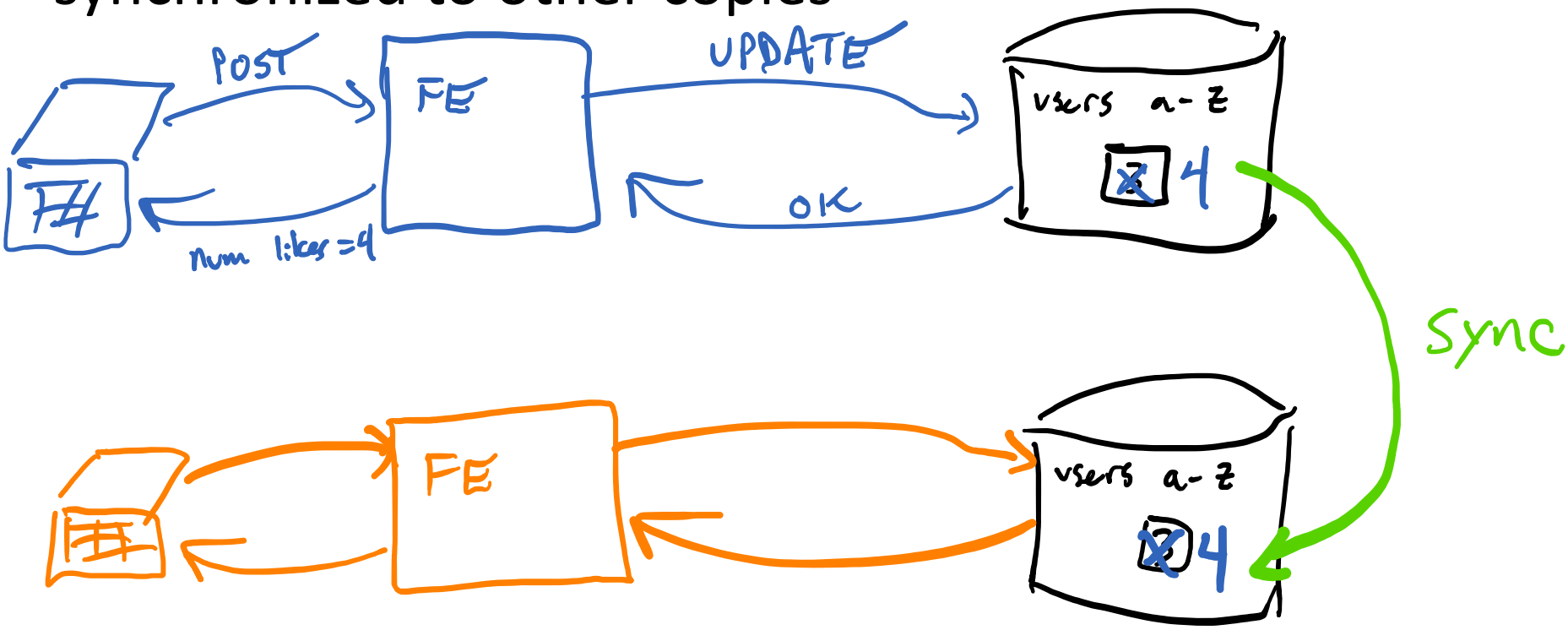
Database replication

- Multiple copies of the entire database



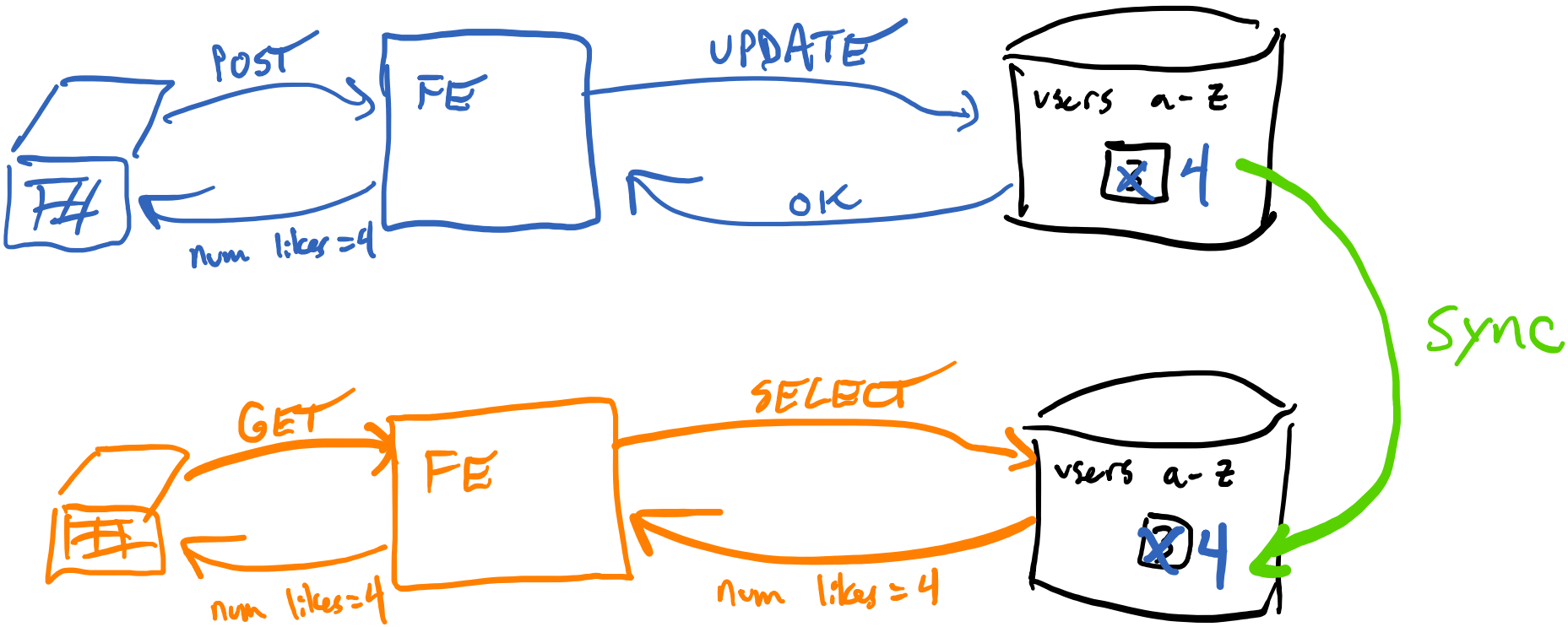
Database replication write

- A write can access any copy and must be synchronized to other copies



Database replication read

- A read can access any copy



Database replication pros and cons

- Multiple copies of the entire database
- Strengths
  - Throughput: Avoid bottlenecks through multiple copies
- Weaknesses
  - Less scalable: Data might be too big to store on all servers
  - Consistency: Hard to keep copies in sync
    - Write to one needs to propagate to all the others
    - Moments of inconsistency: post shows up for some users and not others

Examples

- What kind of websites might need:
  - More front-end servers than database servers?
  - More database servers than front-end servers?

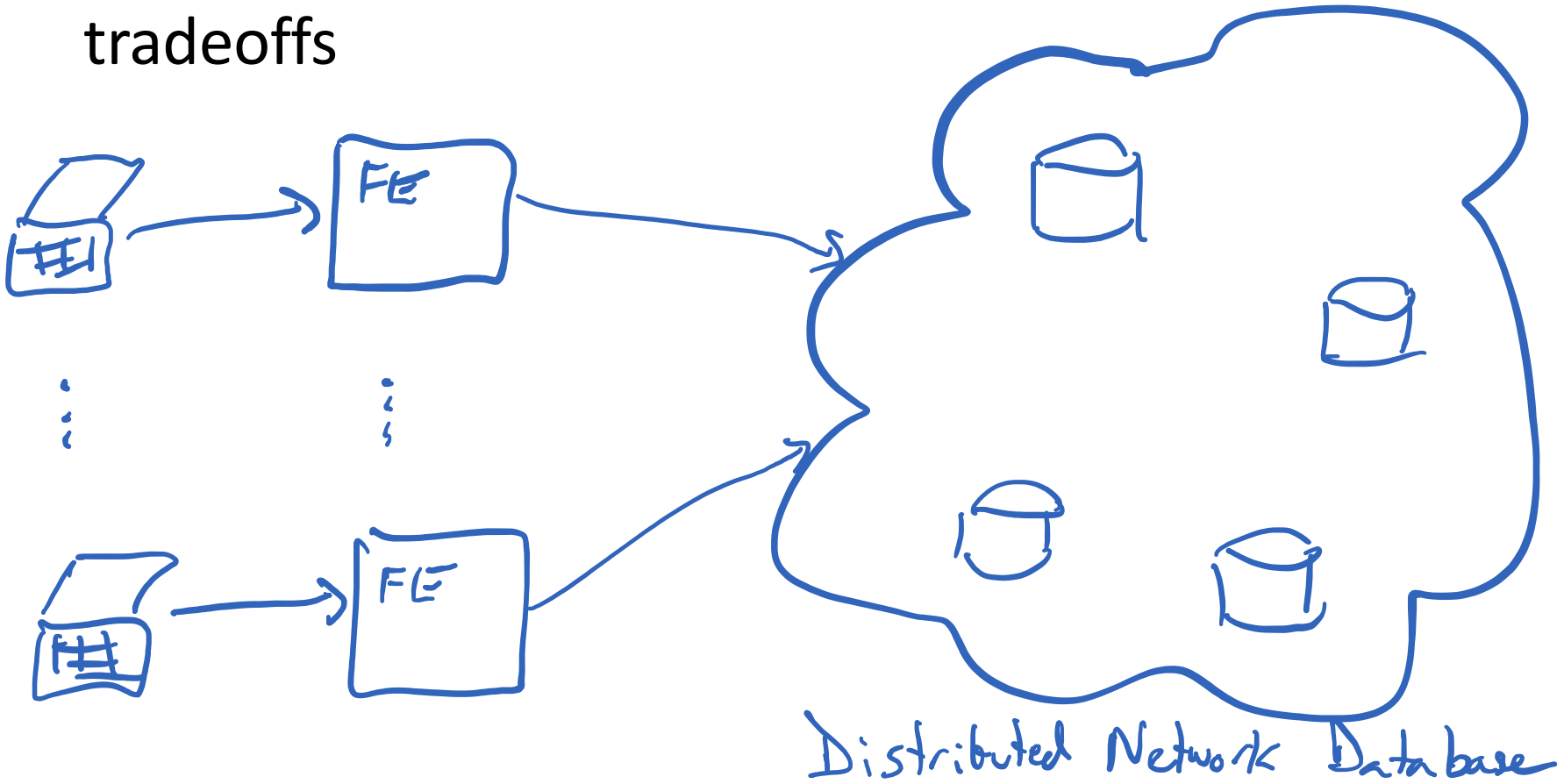
- Which of sharding or replication would be better for scaling databases that store the following data?
  - Edits to Wikipedia articles
  - Tweets on Twitter made by different users
  - Airline flight reservation system

Agenda

- Distributed network database
- Sharding by content, database replication
- **CAP Theorem**
- Practical database examples
- Distributed file system for media uploads

CAP theorem overview

- CAP theorem describes distributed database tradeoffs



CAP theorem: 3 properties

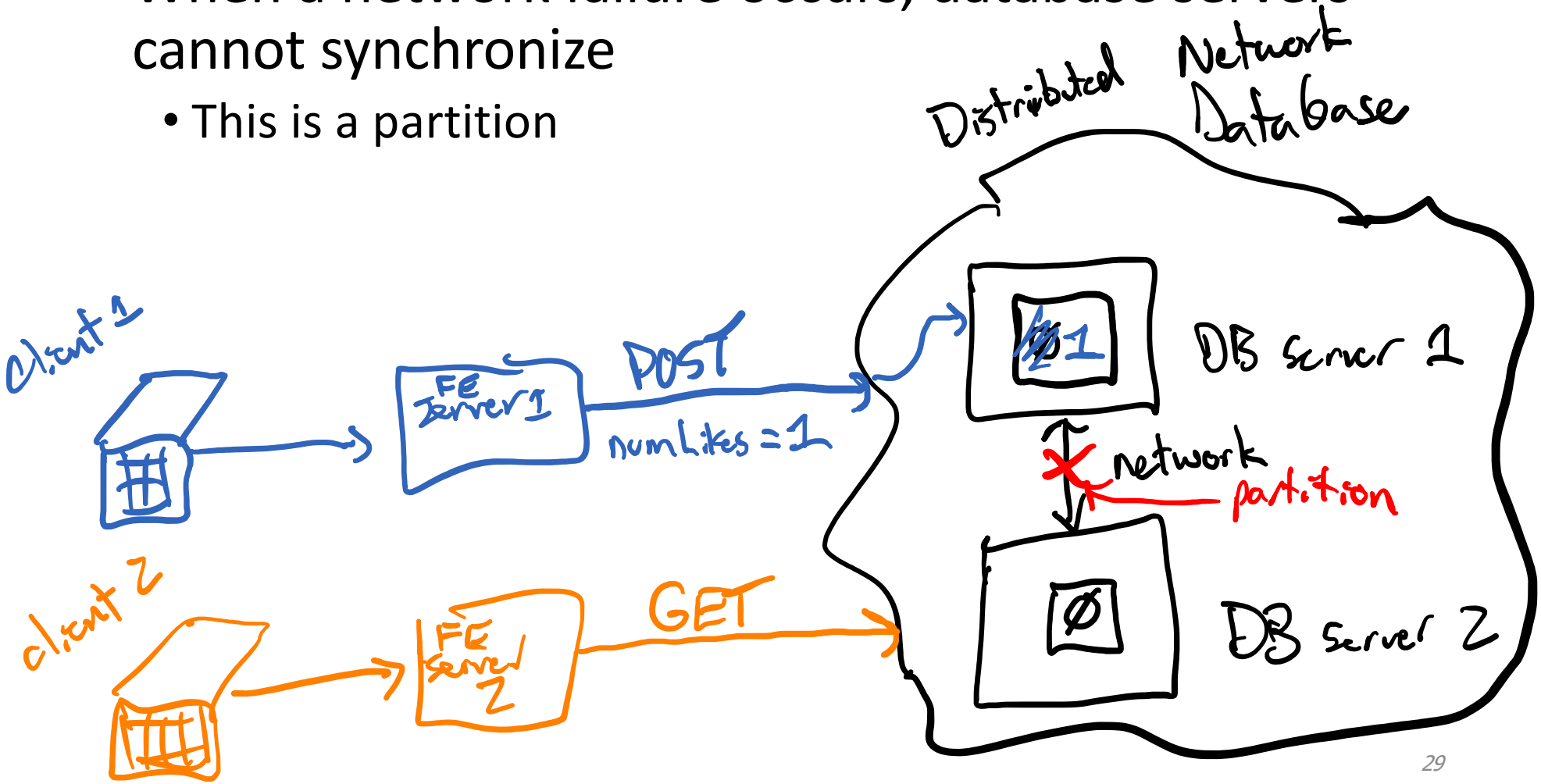
- *Consistency*: Every read receives the most recent write or an error
- *Availability*: Every request receives a (non-error) response, without the guarantee that it contains the most recent write
- *Partition-tolerance*: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes
- CAP Theorem: Pick 2 of 3

Network with no failures

- When the network is working correctly, you get Consistency and Availability
- Database servers can synchronize
- There is no partition

Network failure AKA partition

- When a network failure occurs, database servers cannot synchronize
  - This is a partition

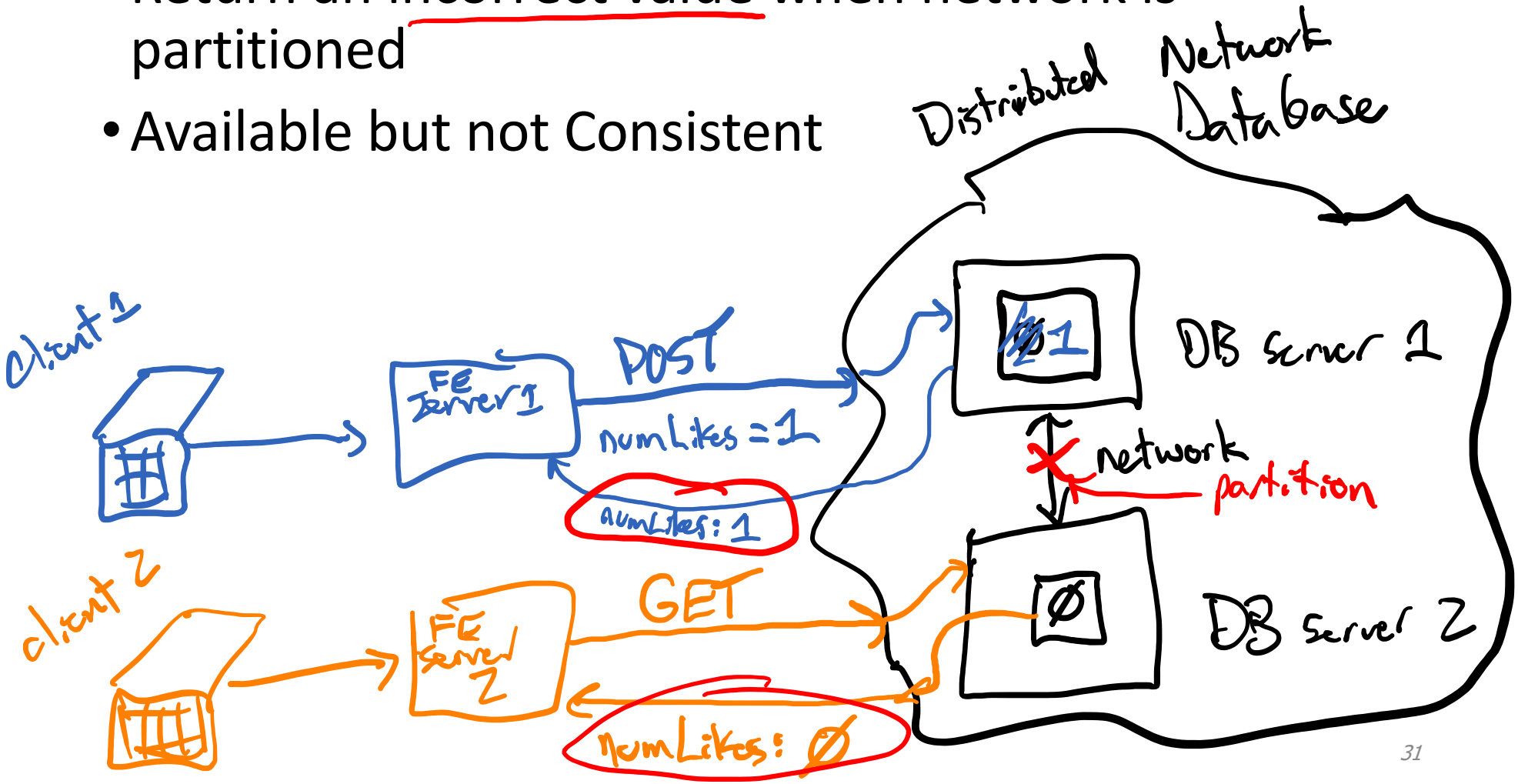


Network failure AKA partition

- When a network failure occurs, database servers cannot synchronize
  - This is a partition
- We have a choice
  - Return an incorrect value
  - Return an error

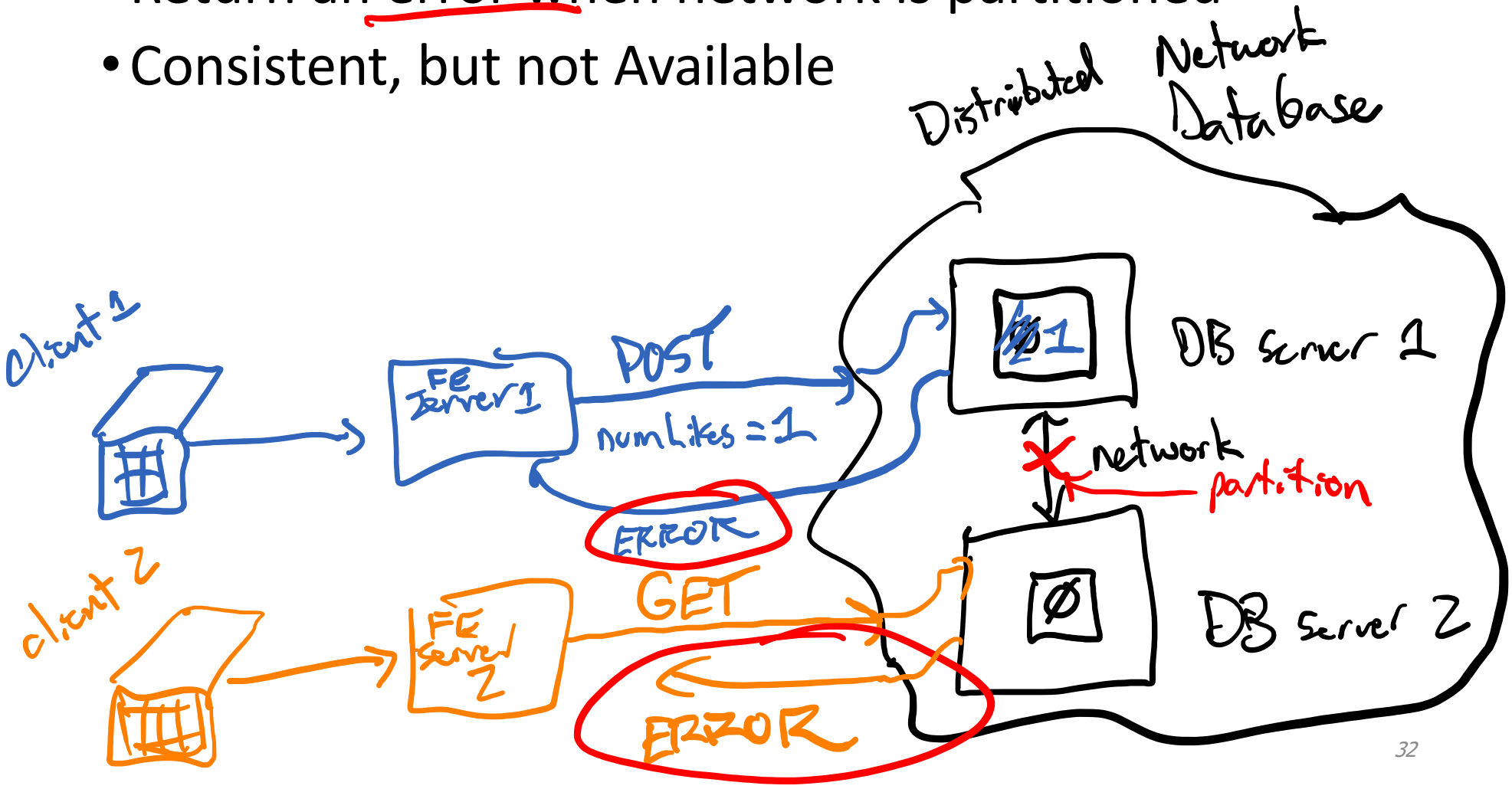
Available but not Consistent

- Return an incorrect value when network is partitioned
- Available but not Consistent



Consistent but not Available

- Return an error when network is partitioned
- Consistent, but not Available



Which to choose?

- Prioritize consistency over availability
  - Relational databases (AKA RDBMS AKA SQL) usually here
  - Postgresql: synchronous replication
- Prioritize availability over consistency
  - "NoSQL" databases
  - MongoDB: primary which is consistent, secondaries that might have stale data
- Avoid the problem
  - Buy a giant server for your DB so there's only one
  - *Centralized network database*

Examples

- Which of these applications requires consistency? Which require availability?
  - Instagram feed
  - Airline flight reservation system
  - Amazon product search

Agenda

- Distributed network database
- Sharding by content, database replication
- CAP Theorem
- **Practical database examples**
- Distributed file system for media uploads

Practical database examples

- We'll compare 3 widely used software packages:
  - SQLite
  - MySQL
  - PostgreSQL

SQLite strengths and weaknesses

Strengths

- Serverless
- File-based
- Low memory
- Easy setup

Weaknesses

- No network
- No replication

MySQL strengths and weaknesses

Strengths

- Distributed network database
- Fastest read operations

Weaknesses

- Partial SQL standard compliance
- Slow development

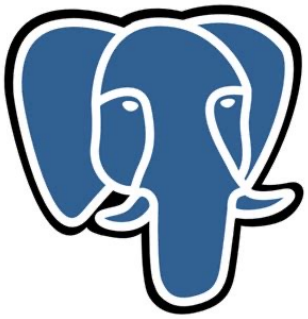
PostgreSQL strengths and weaknesses

Strengths

- Distributed network database
- SQL compliance
- Balanced read and write optimization
- Stronger consistency with simultaneous reads & writes

Weaknesses

- Reads not as fast as MySQL



PostgreSQL

SQLite vs. MySQL vs. PostgreSQL

SQLite	MySQL	PostgreSQL
Single-user applications	Distributed applications	Distributed applications
Small data sets	Large datasets	Large datasets
	Fastest possible reads	Balanced reads and writes
		Stronger consistency with simultaneous reads & writes

Agenda

- Distributed network database
- Sharding by content, database replication
- CAP Theorem
- Practical database examples
- **Distributed file system for media uploads**

Media uploads and the database

- Media uploads: Instagram photos, TikTok videos, etc.
- Problem: Media uploads are expensive to store in a database
- Solution: Store to disk

- Write once, read many times. Don't need a database to maintain consistency.

Scaling media uploads

- Different dynamic pages servers on every request
  - Physical machines, virtual machines, containers
- Problem: How do dynamic pages server disks stay in sync?
- Solution: Dynamic pages servers are stateless. Store media uploads on network storage

Properties of media uploads

- Media uploads have something in common with both static pages and dynamic pages
- Media uploads are like static pages
  - Once they're uploaded, they never change
- Media uploads are like dynamic pages
  - Content created by users
  - Access permissions per-user

Media storage implementation

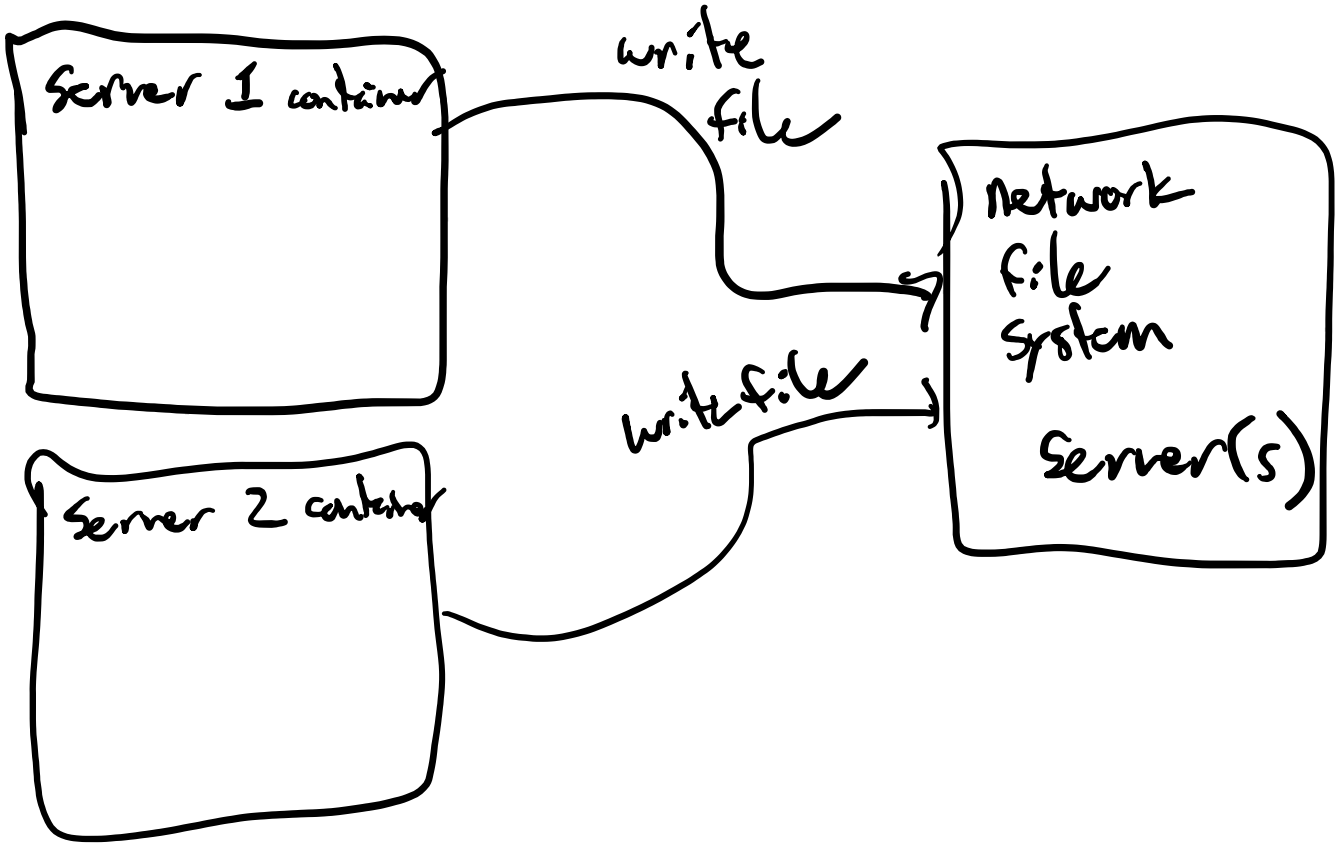
- Two subproblems for media storage
- Problem 1: Store the files
  - Many files
  - Large size
  - Exabytes of data (10^18)
- Problem 2: Serve the files
  - Many requests
  - Permissions control per-user

Store the files

- Problem 1: Store files
- Solution 1: Network file system
- Solution 2: Distributed file system

Network file system

- *Network file system:* Access files over the network much like local storage (hard drive)



Network file system PaaS

- PaaS managed network file systems

- AWS Elastic File System (EFS)
- Google Filestore
- Microsoft File Storage

Network file system pros and cons

- Network file system acts like a local file system
  - Near zero client code changes
- Problem 1: Speed and scalability
- Problem 2: Fault tolerance. What happens when network file system server goes down?

- Network file systems weren't designed with the scale of the web in mind

Distributed file system

- *Distributed file system:* Access files over the network, often with a special API
- Reliable, scalable file storage implemented with commodity hardware
- Google File System is an historical example

Distributed file system PaaS

- PaaS managed distributed file systems

- AWS S3
- Google Cloud Storage
- Microsoft Blob Storage

Distributed file system pros and cons

- Distributed file systems are more scalable and fault tolerant than network file systems
- Distributed file systems often use a special API
  - Requires client code changes

Network FS vs. distributed FS

- Technically, both are distributed systems
- Optimized for different things

- *Network file system* is optimized to be as similar as possible to traditional file system
- *Distributed file system* is optimized for scale and reliability on unreliable hardware

Connecting dynamic pages servers to distributed file system

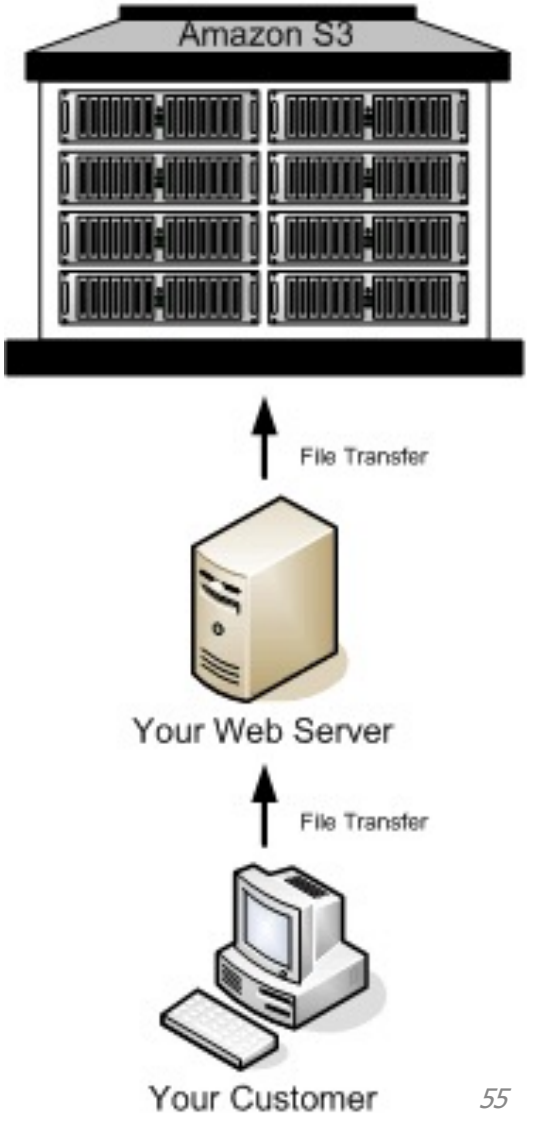
- Our data is now stored in a distributed file system
  - User connects to dynamic pages web server

- Problem: How do we upload and download?
- Solution 1: Dynamic pages server is a middleman between client and distributed file system
- Solution 2: Client connects directly to distributed file system

Media uploads proxy

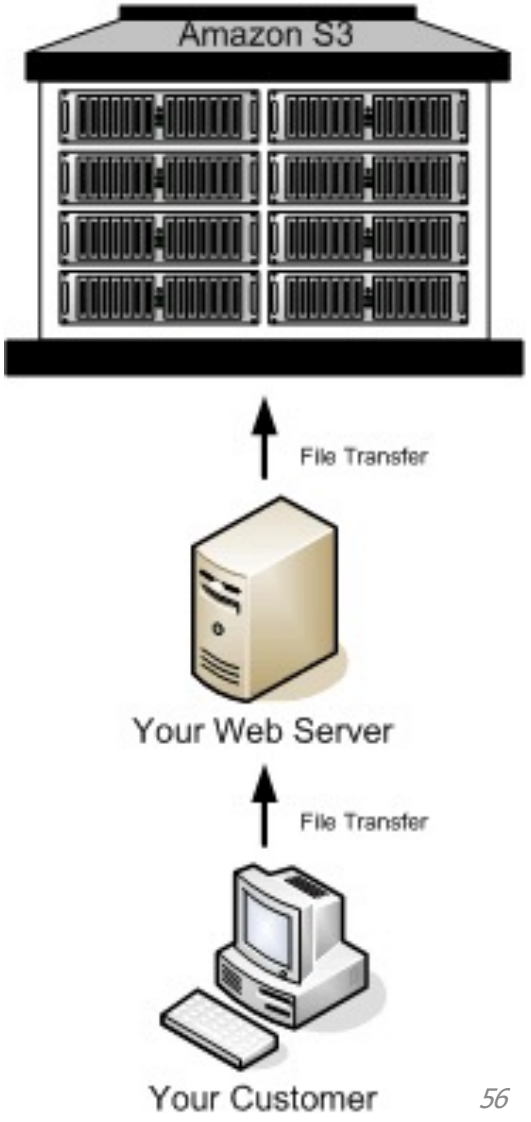
- Client sends upload to to dynamic pages server
- Dynamic pages server sends upload to distributed file system
  - Example: AWS S3

- Dynamic pages server is "Middleman" or "proxy"



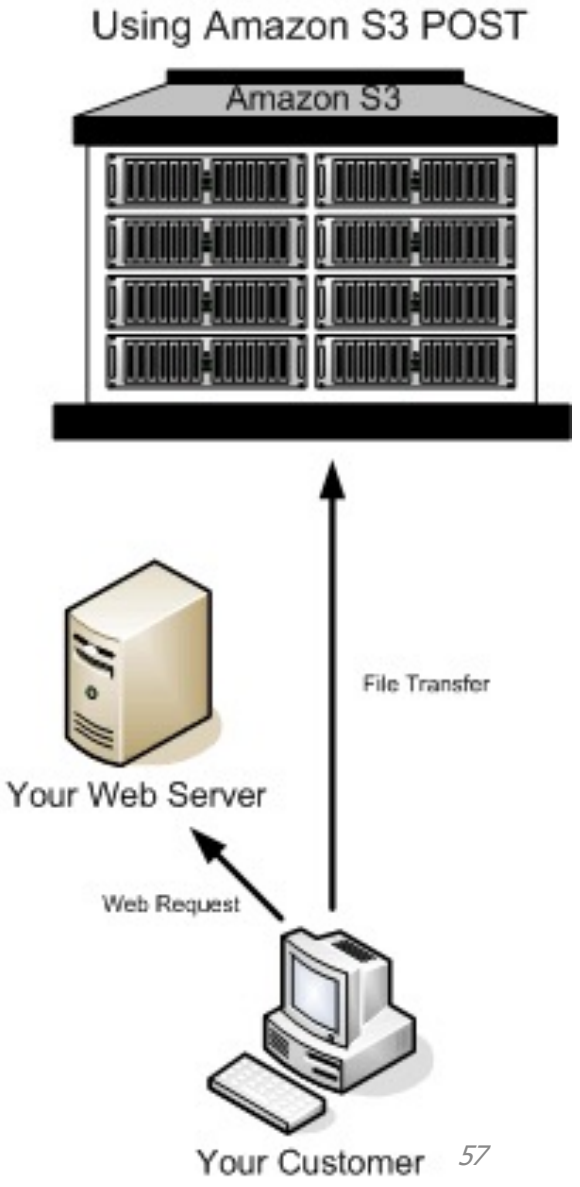
Media uploads proxy pros and cons

- Easy to implement
- Problem: Large file transfer goes through multiple servers. Higher latency, higher bandwidth
- Solution: Direct upload



Direct upload

- Direct upload from client to distributed file system
- Dynamic pages server generates signed URL for client
- Client uploads large file directly to distributed file system



Why not use a CDN?

- What would be the advantages and disadvantages if we used a CDN to distribute media uploads?

- Advantages: fast
- Disadvantages:
  - Content changes a lot, CDNs best for content that changes infrequently
  - How to prevent users from accessing media they aren't supposed to?
  - Huge size -> expensive, maybe not possible

Summary

- Distributed network databases solve the problem of dynamic pages server synchronization
  - Dynamic pages servers are stateless
  - Storage goes to the database
- Keeping DB servers in sync is hard
  - Sharding by content puts different data on different DB servers
  - Database replication puts all the data on all the servers
- CAP Theorem
  - Consistency, Availability, Partition-tolerance. Pick two.

Summary

- Practical database examples
  - Unsure? Use PostgreSQL
- Media upload storage
  - PaaS distributed file system
  - Option 1: Dynamic pages servers act as middleman between client and distributed file system
  - Option 2: Client connect directly to distributed file system with temporary "secure" link