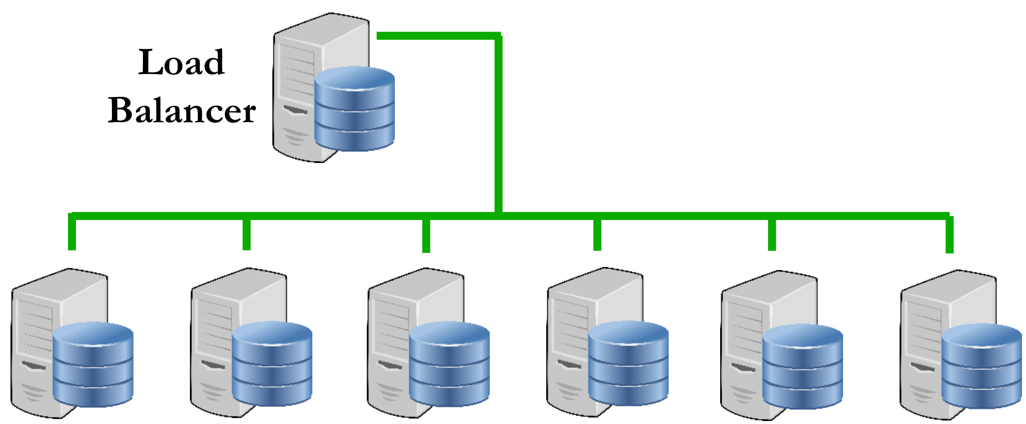
Consistency and CAP

- **Types of consistency**
- **Examples**
- **CAP theorem**

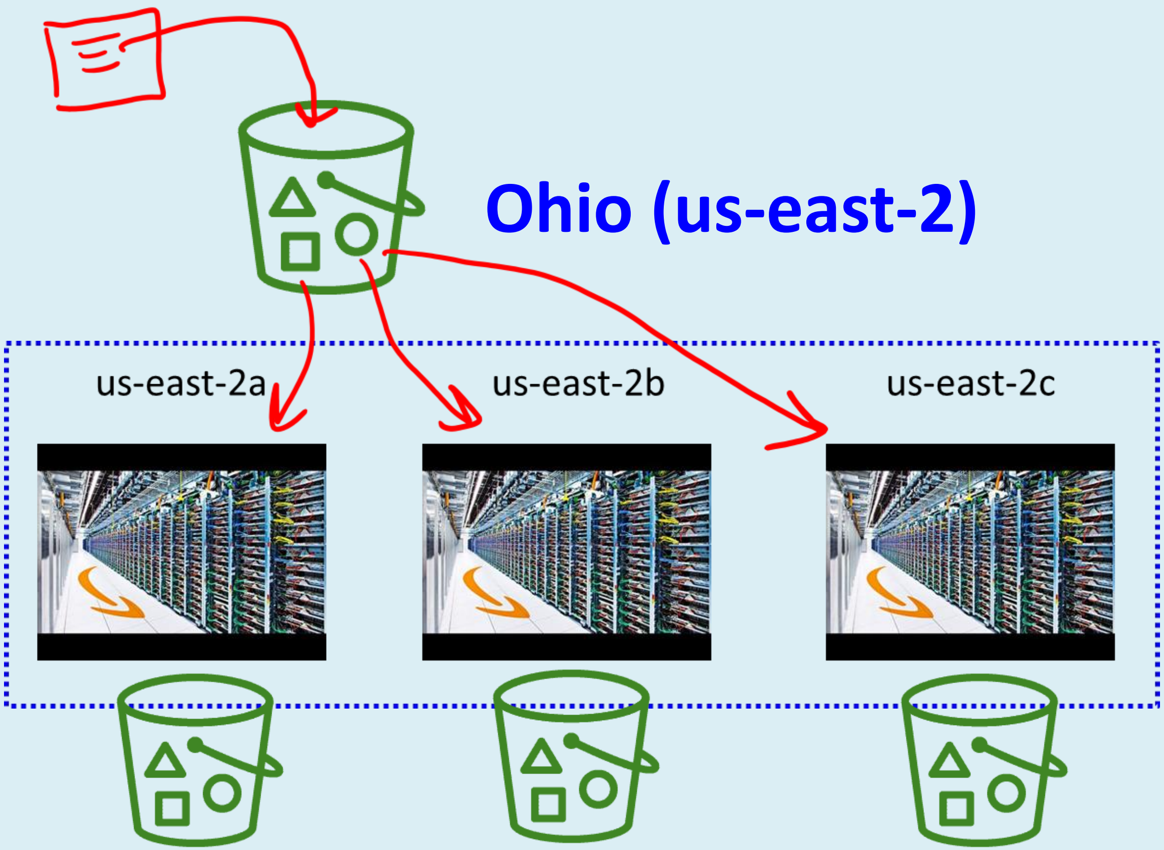
Availability / fault tolerance

- Computers crash, it will happen…
- Only way to keep your system **available** is with multiple computers
- A system that keeps running in presence of failures is **fault tolerant**



Example: S3 is fault tolerant

- **Buckets are replicated across all sites in a region in case one of the sites loses power / network connectivity…**

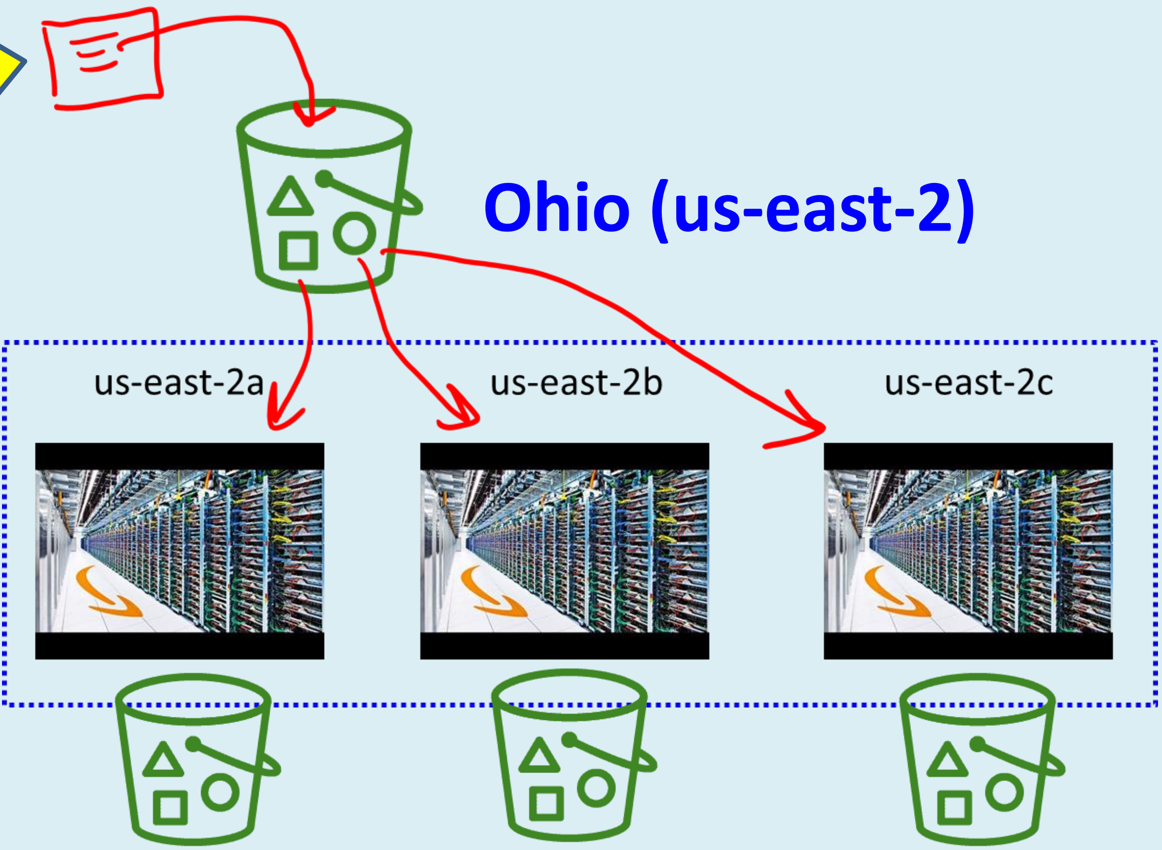


Interesting question…

- **What type of consistency does AWS guarantee?**

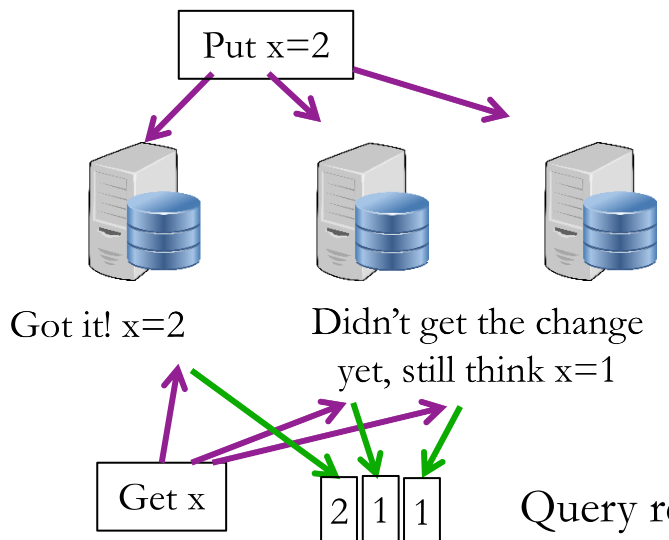
If we upload an image to S3, how soon is it available in all 3 sites?

What if a download request is routed to a site that hasn't finished replicating? What happens? What does AWS guarantee, if anything?



Consistency

- Whenever data is replicated, there is a possibility of **inconsistency**.
  - Example: x =1. An update is then sent to three replicas:



Communication and queueing delays are unpredictable. We can send messages, but we don't know exactly when they'll arrive.

Query returns inconsistent answers!

Types of consistency

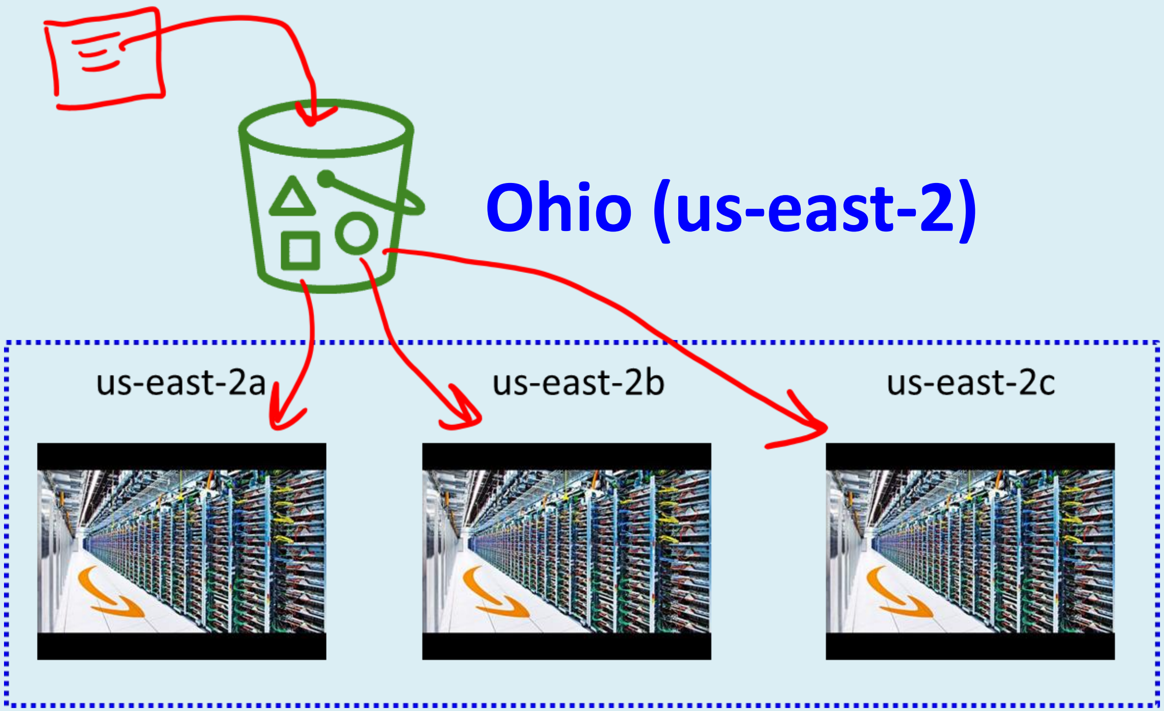
- **Eventual**
  - *An update to a distributed system will *eventually* yield a consistent view*
  - *BASE:*
    - *Basically-available*
    - *Soft-state*
    - *Eventually-consistent*

- **Strong**
  - *A distributed system provides a single, consistent view at all times*
  - *ACID:*
    - *Atomic*
    - *Consistent*
    - *Isolated*
    - *Durable*

The type of consistency impacts how you call / use a given service. Can you trust the response you get?

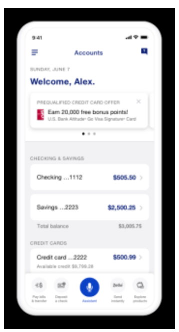
S3? Answer…

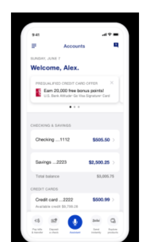
- **The image will be available for download no matter which site is accessed**
  - *AWS S3 guarantees* ***strong consistency***
  - *The client's view of S3 is THE SAME across all sites*



Banks require strongly consistency

- Suppose a bank account has exactly $1,000
- How does the bank prevent two different people from withdrawing $1,000 from that account at the same time?





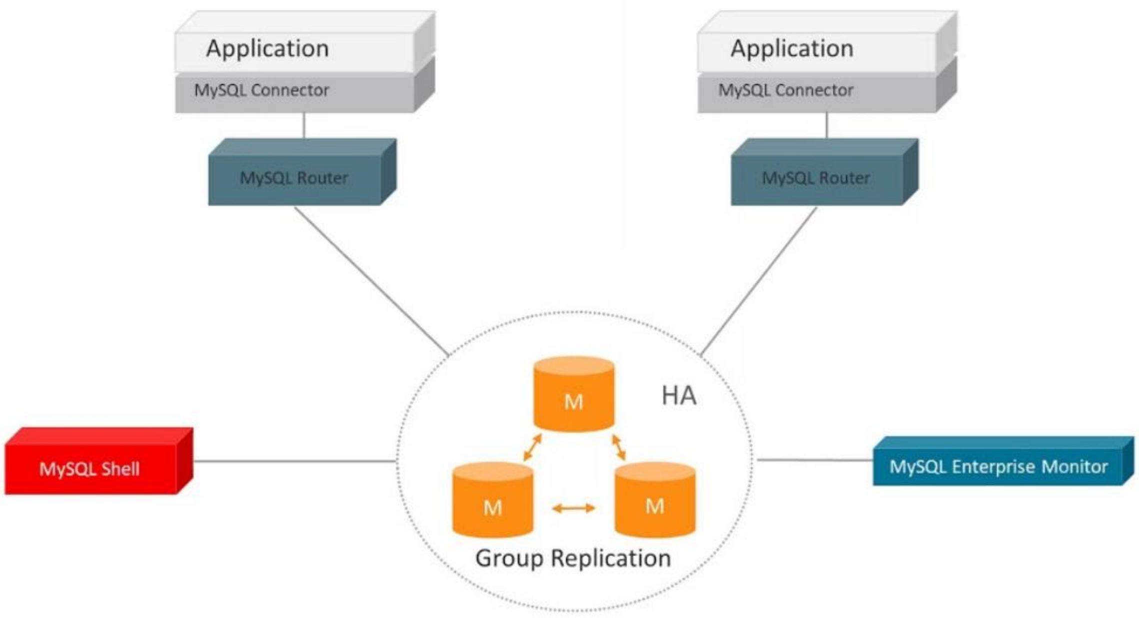
Databases are strongly consistent

- **Most DB systems offer strong consistency**

- **This is HARD to implement:**
  - *Must replicate data so it's accessible at all times anywhere*
  - *Must ensure a strongly consistent view of all data!*

MySQL Cluster

- **MySQL Cluster is a <u>distributed</u> database system**
  - *Replicates the data across multiple servers for fault tolerance*
  - *Still guarantees strong consistency*



MySQL High Availability Cluster

Example

- **Banks cannot risk losing track of customer $**
- **Banks use a DB to keep track…**
- **Transferring $ requires two SQL queries:**

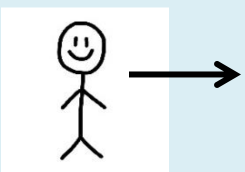
```sql
-- withdraw from checking
UPDATE Accounts
SET    Balance = Balance - 100.00
WHERE  Account = 22197;

-- deposit into savings
UPDATE Accounts
SET    Balance = Balance + 100.00
WHERE  Account = 43992;
```

What if the computer crashes right here? The money would be lost… What must be done to ensure strong consistency?

Answer

- You must wrap the SQL within a transaction
- Databases are strongly consistent only in the presence of properly-written transactions…

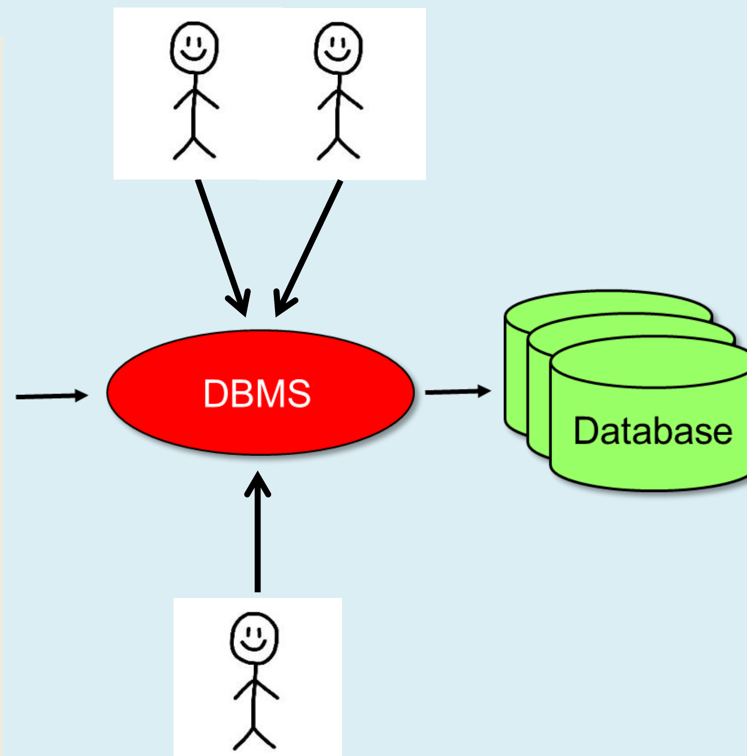


```sql
BEGIN TRANSACTION;

  -- withdraw from checking
  UPDATE Accounts
  SET    Balance = Balance - 100.00
  WHERE  Account = 22197;

  -- deposit into savings
  UPDATE Accounts
  SET    Balance = Balance + 100.00
  WHERE  Account = 43992;

COMMIT;
```

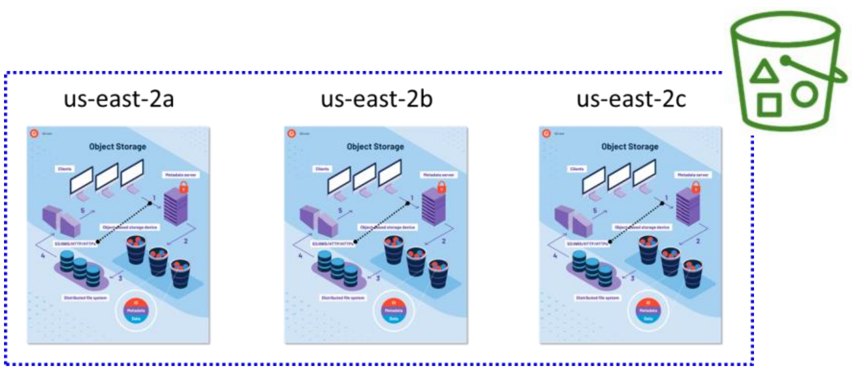


CAP Theorem

One of the most important results in distributed systems theory

Theorem: a distributed system cannot achieve ***all three*** of the following:

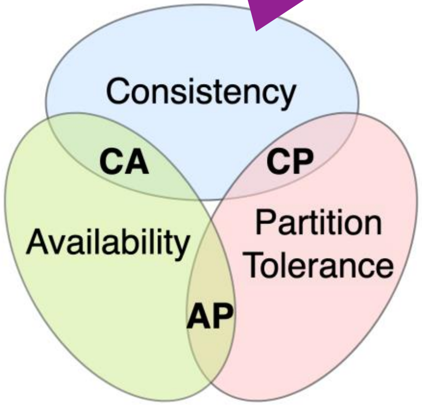
- **C onsistency**: reads always return the most recent write (or an error)
- **A vailability:** every request receives a timely, non-error response
- **P artition tolerance:** the system continues to operate even in the presence of failures (software, hardware, network, power, etc.)



You have to pick two…

CAP Theorem implications

Pick Two: CA, CP, AP



- **C**onsistency: reads always return the most recent write (or error)
- **A**vailability: every request receives a timely, non-error response
- **P**artition tolerance: the system continues to operate even in the presence of failures (software, hardware, network, power, etc.)

redundancy k2 partition tolerance
but that means inconsistency if we want results fast
or slow if we want results consistent

Implications?

Redundancy is the only way to achieve partition tolerance, i.e. fault-tolerant, highly-available systems. **So you always choose P.**

This implies architects have to choose between

- **Availability (AP)**: returning an answer to the client that may be inconsistent (old)

or

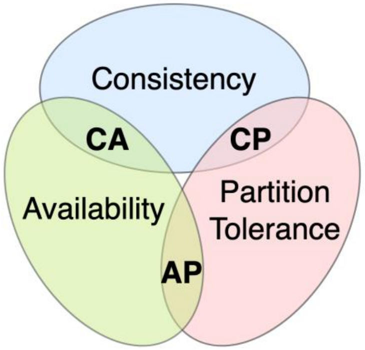
- **Consistency (CP)**: making the client wait until consistent answer is available (in the worst-case, request could error/timeout & client will have to try again)

Client-centric consistency models

- The CAP theorem gives us a tradeoff between **consistency & delay**
- Inconsistency is bothersome --- makes client-side programming harder
  - *Example: imagine if the database gave different answers to the same query?!*
- Delay is usually something client-side apps can handle…

slower, but consistent and fault tolerant

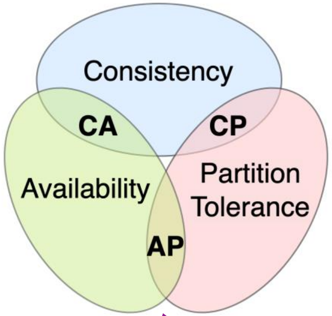
Most architects agree that CP is the right choice



- Client-side code is less complex
- In almost all systems, you want correct responses over fast (but wrong) responses
- Good summary from a Google architect

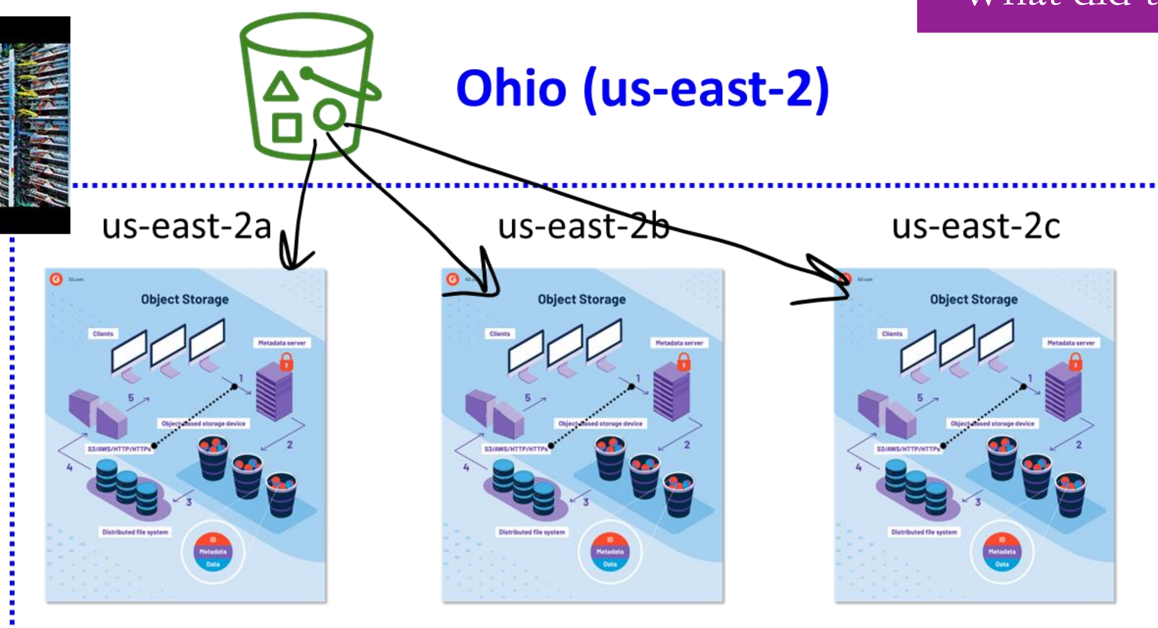
S3: eventual to strongly consistent

- **In 2020, S3 was redesigned to be strongly consistent**
  - *https://www.allthingsdistributed.com/2021/04/s3-strong-consistency.html*

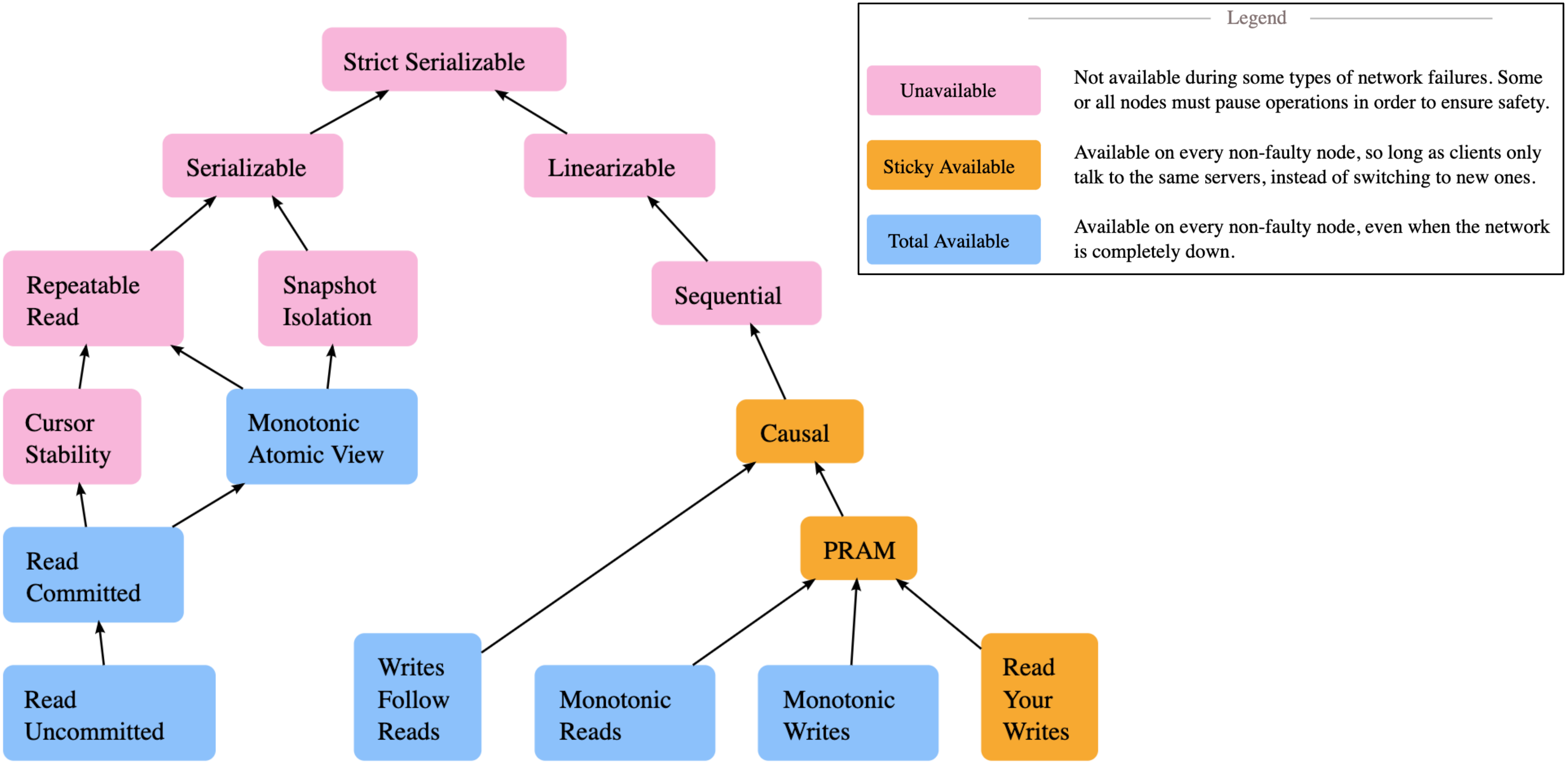


What did they give up?

avail



Consistency is a subtle topic, with [many models](#).




Core Challenges Distributed Systems
Complexity
Multiple components requiring seamless integration and coordination
01
02
Consistency
Difficult to maintain across different nodes, especially in stateful apps.
Fault Tolerance
Crucial to design for failure recovery with minimal downtime.
03
04
Scalability
Must scale efficiently under varying loads without performance loss.
Concurrency
Managing simultaneous operations and conflict resolution in components.
05
06
Security
Ensuring confidentiality, integrity, and availability in a distributed network.
Network Issues
Susceptible to latency and partitioning in network communication.
07
08
Deployment and Management:
Resource-intensive, often needs specialized infrastructure and tools.
Debugging and Monitoring
More challenging due to distributed systems' dispersed nature.
09
10
Decoupling
Improves flexibility but complicates coherence and system coordination.

That's it, thank you!