UNIVERSITY of VIRGINIA

CS4740
CLOUD COMPUTING

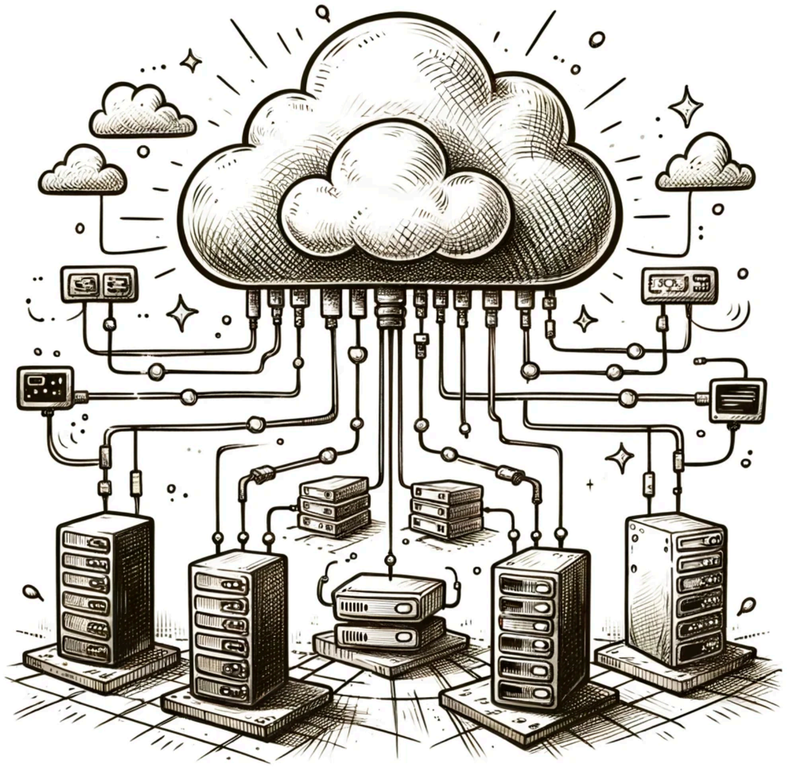
Distributed System Foundation

Prof. Chang Lou, UVA CS, Spring 2024

AGENDA

- Challenges, Goals, and Approaches for DS
- Example: Web Service Architecture

Why building cloud systems is hard?



STARBUCKS

— Imagine you are operating a Starbucks

STARBUCKS

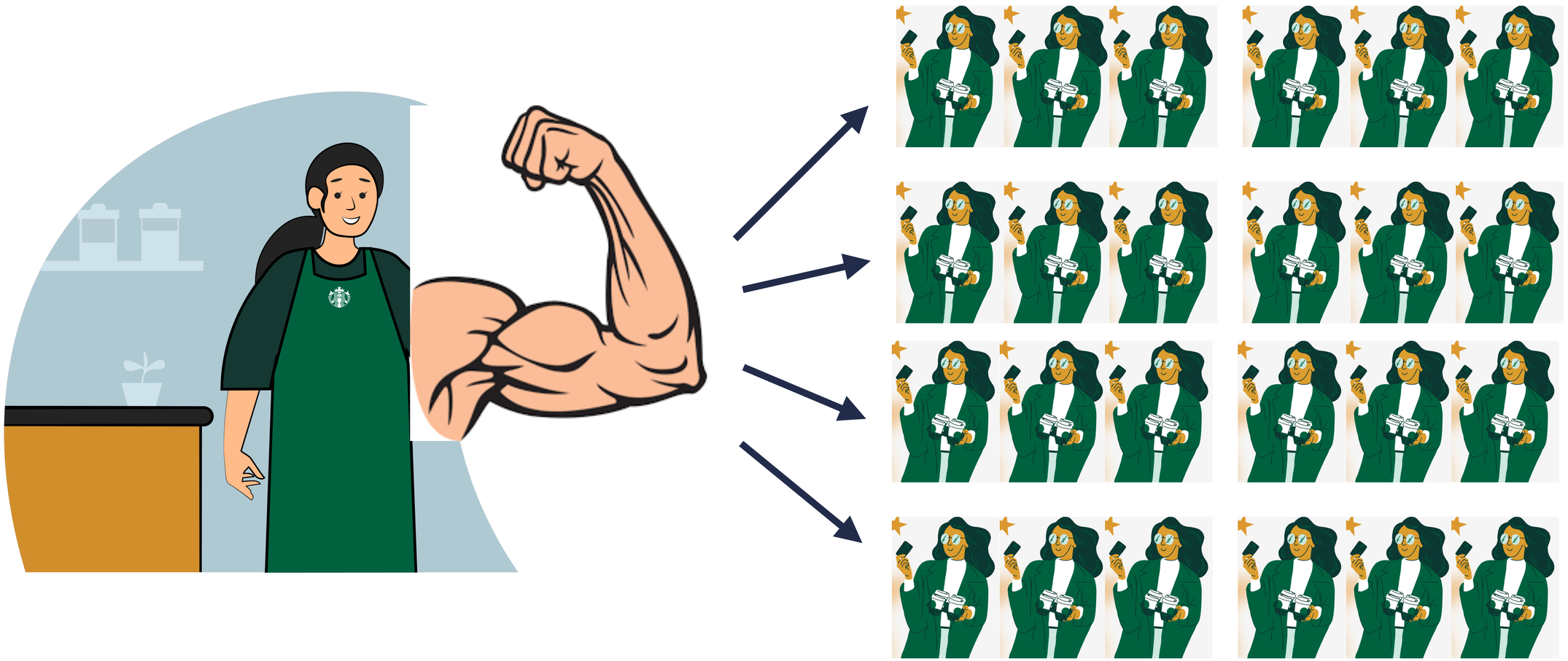
STARBUCKS

Now imagine 1000x customers?

CHALLENGE 1: EVER-GROWING LOAD

- Data is big. Users are many. Requests are even more.
- Google get 8.5 billion searches per day.

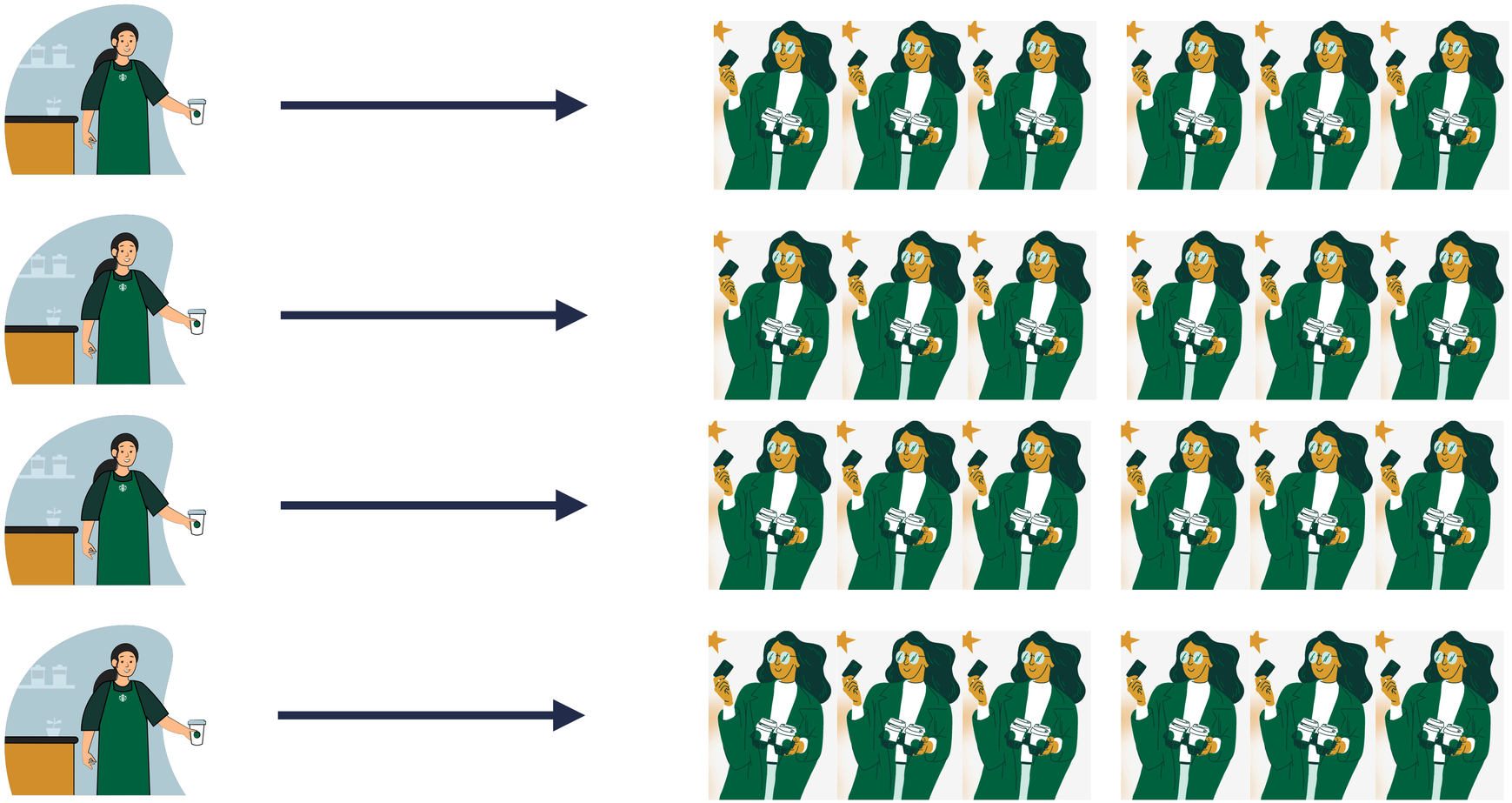
SCALE-UP?



SCALE-UP?

- You can always add more compute resources—such as CPU, memory, and disk capacity.
- But no single machine can handle the ever-growing load.

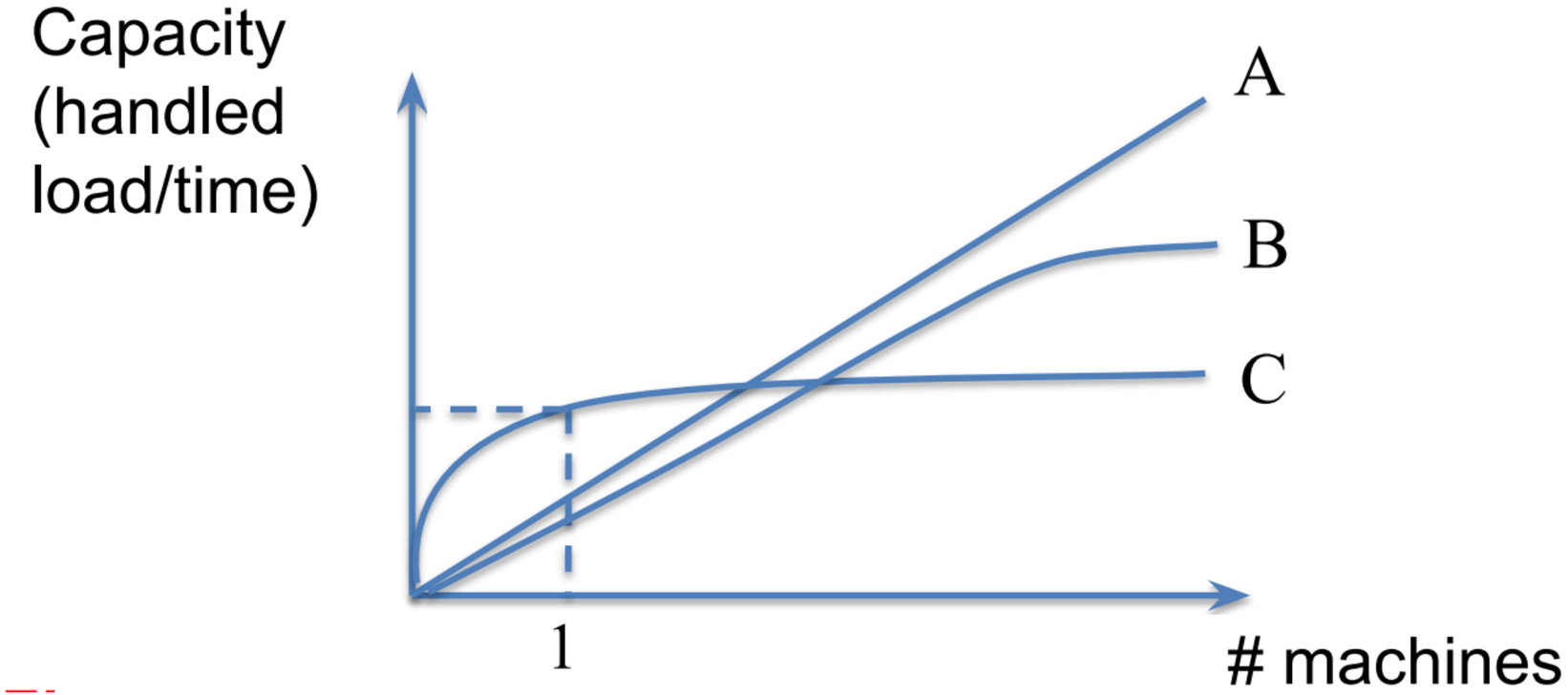
APPROACH 1: SCALE-OUT (SHARDING)



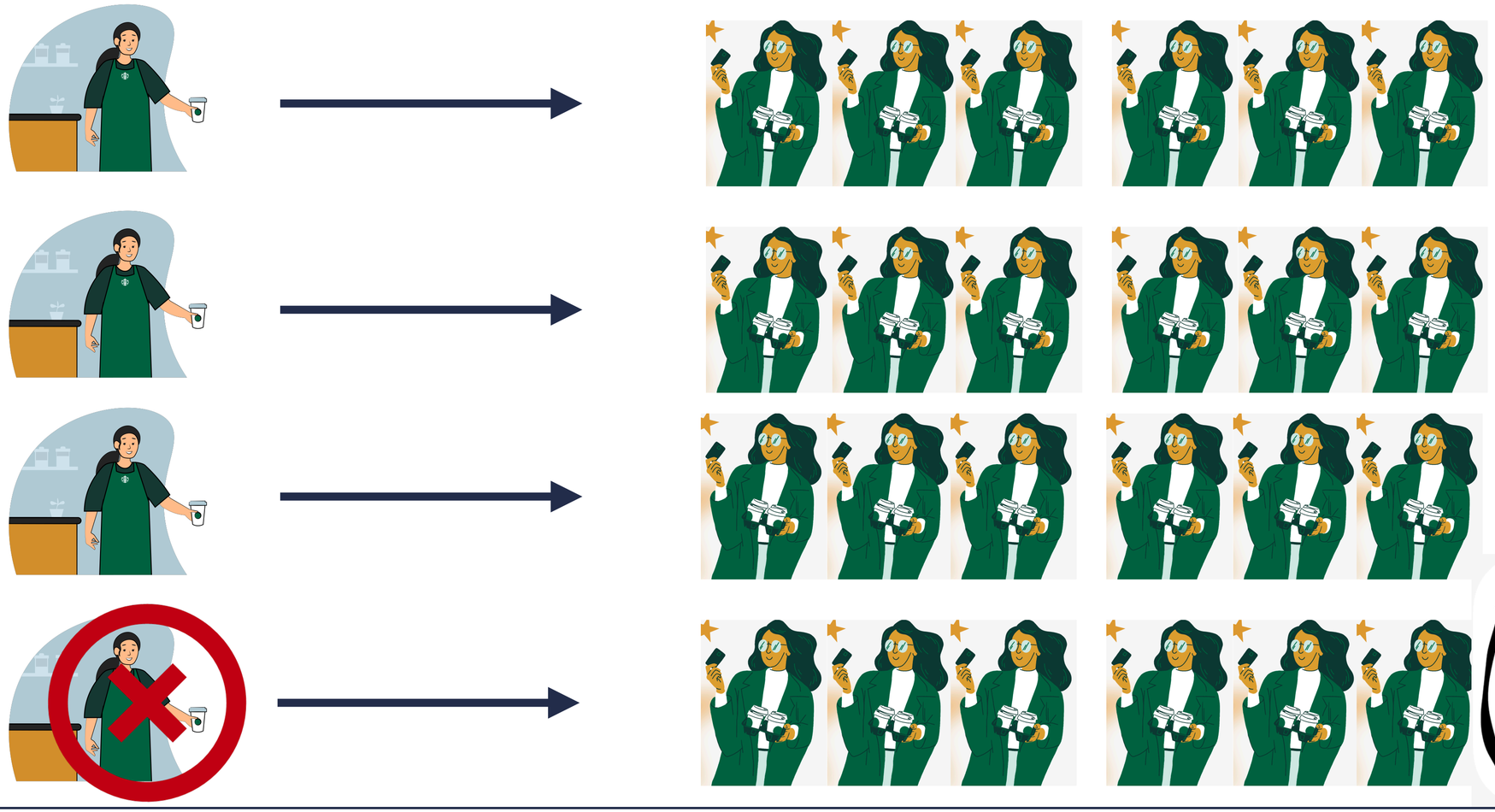
GOAL 1: SCALABILITY

- The more resource you add, the more requests you can serve.
- But never hope for perfect scalability
  - add one machine, increase your capacity proportionally forever?

SAMPLE SCALABILITY CURVES



CHALLENGE 2: AT SCALE, FAILURES ARE INEVITABLE



GOAL 2: FAULT TOLERANCE

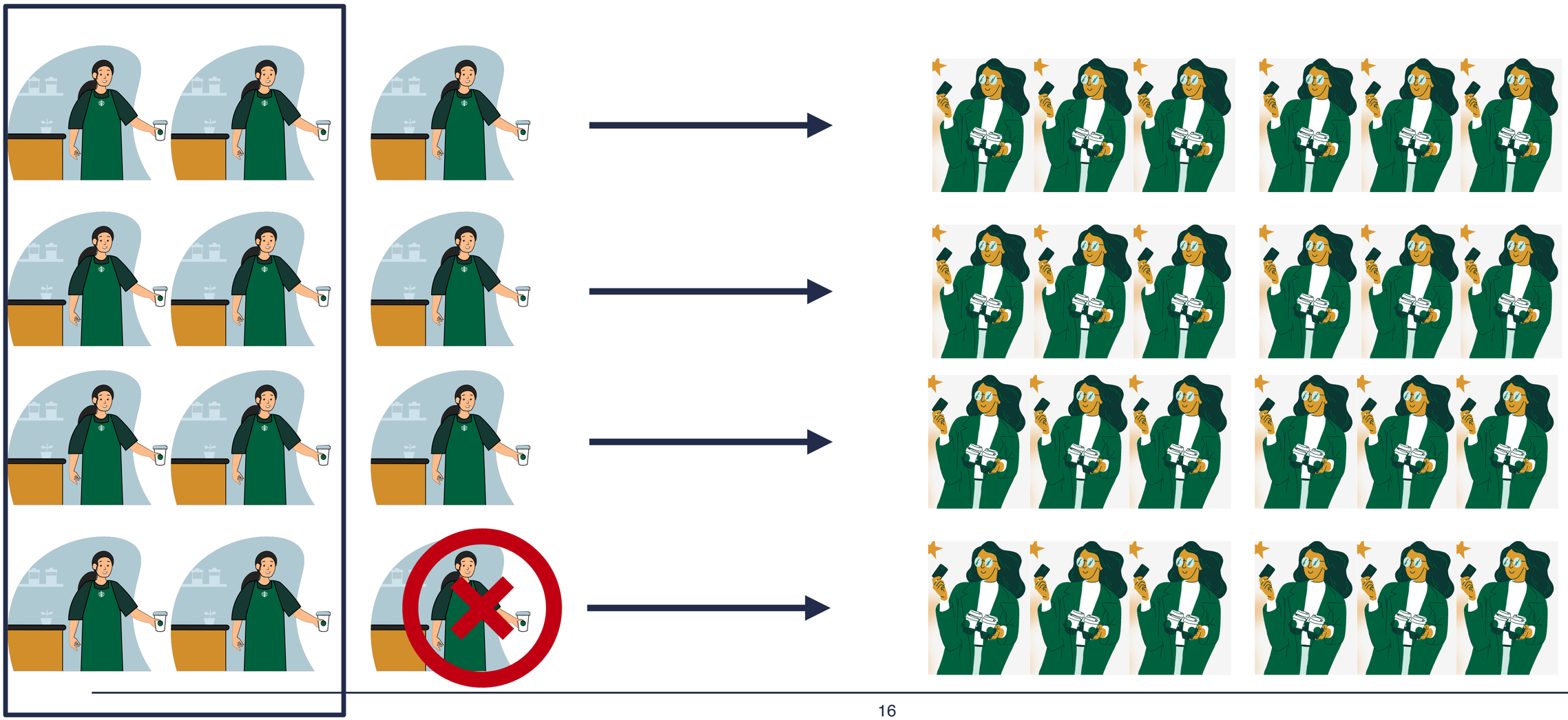
- Goal is to **hide** failures as much as possible to provide a service that e.g., finishes the computation fast despite failures, stores some data reliably despite failures, …
- Fault tolerance subsumes:
  - Availability: the service/data continues to be operational despite failures.
  - Durability: some data or updates that have been acknowledged by the system will persist despite failures and will eventually become available.

GOAL 2: FAULT TOLERANCE

Availability: I expect someone will always take my order ..

Durability: once I placed my order, I expect nobody ask me again..

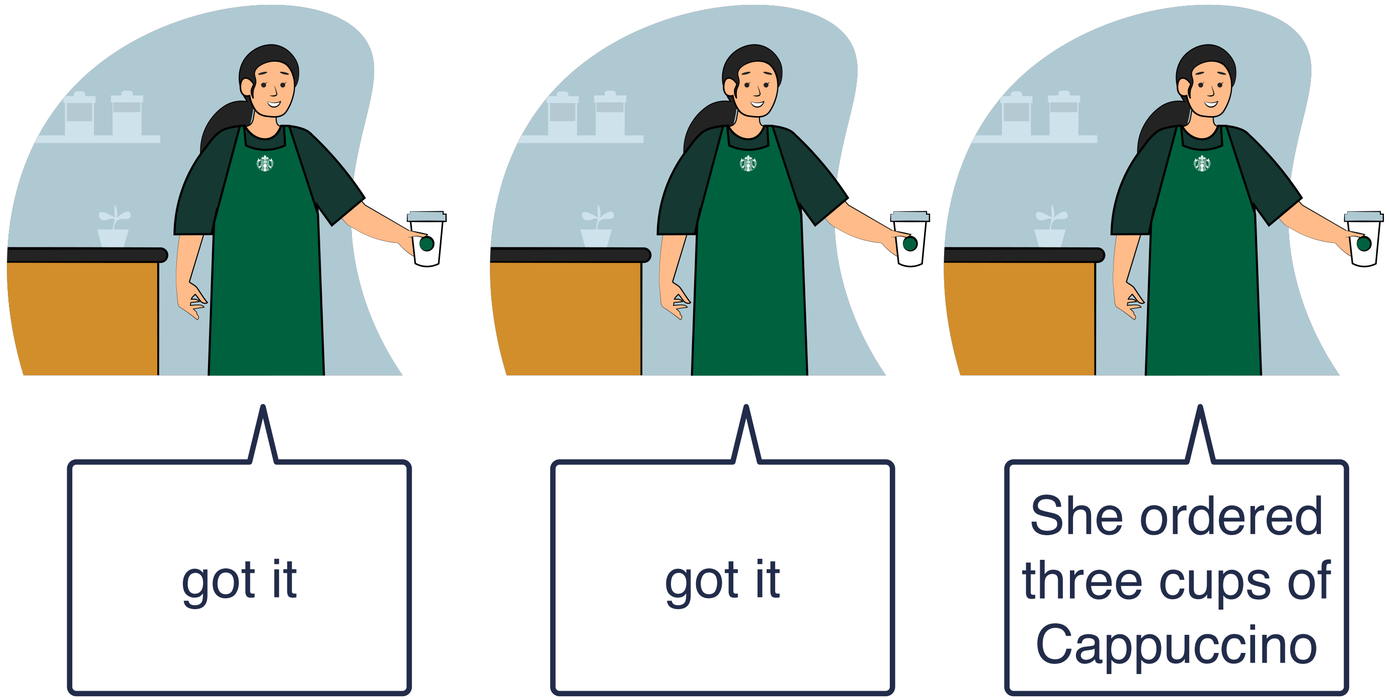
APPROACH 2: REPLICATION



CHALLENGE 3: CONSISTENCY

Three cups of Cappuccino please

CHALLENGE 3: CONSISTENCY



CHALLENGE 3: CONSISTENCY

(didn't hear)

got it

Change to Espresso

Wait, change to Espresso

CHALLENGE 3: CONSISTENCY

Your Cappuccino is ready.

Your Espresso is ready.

???

GOAL 3: CONSISTENCY GUARANTEE

- Distributed systems try to create an **illusion** that users are using one single powerful machine
  - They guarantee that every replica has the same view of data

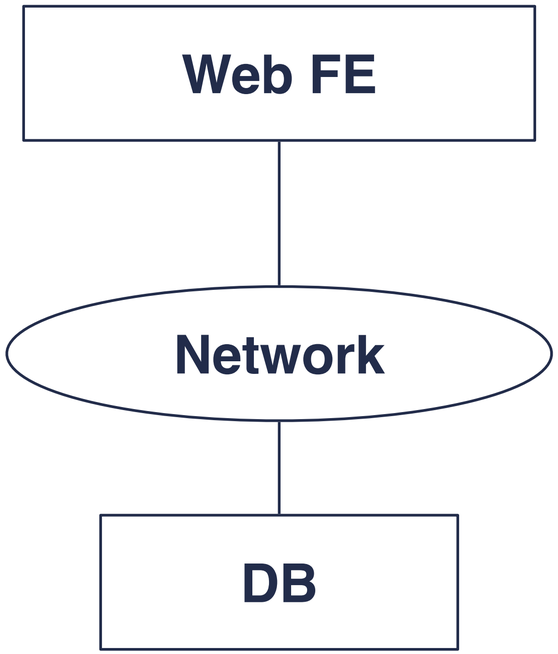
APPROACH 3: PROTOCOLS

- The general approach is to develop rigorous protocols, which we will generically call agreement protocols, that allow workers and replicas to coordinate in a consistent way despite failures.
- Agreement protocols often rely on the notion of majorities: as long as a majority agrees on a value, the idea is that it can be safe to continue making that action.
- Different protocols exist for different consistency challenges, and often the protocols can be composed to address bigger, more realistic challenges.

AGENDA

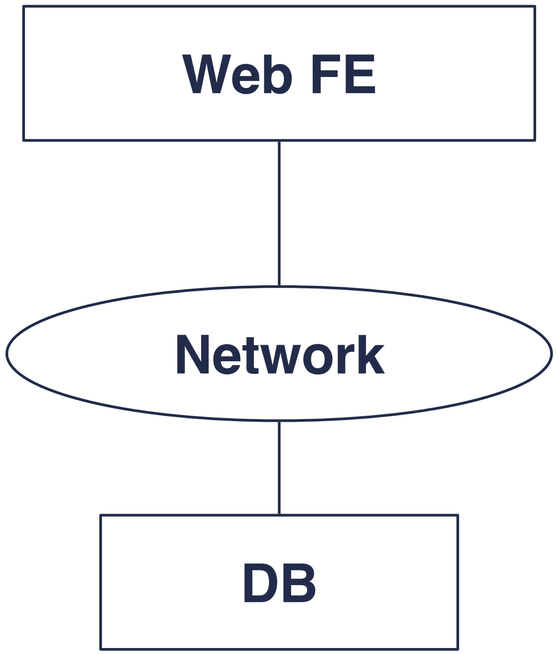
- Challenges, Goals, and Approaches for DS
- **Example: Web Service Architecture**

BASIC ARCHITECTURE



- Web front end (FE), database server (DB), network. FE is stateless, all state in DB.
- Properties:
  - Performance?
  - Fault tolerance?
  - Scalability?
  - Semantics?

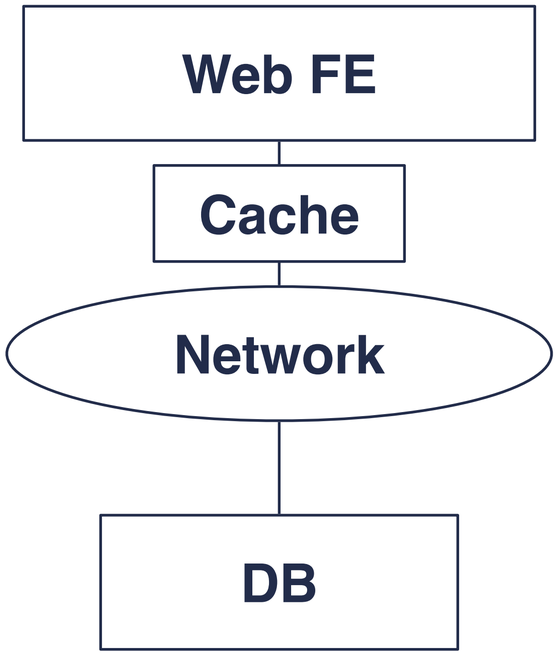
BASIC ARCHITECTURE



- Web front end (FE), database server (DB), network. FE is stateless, all state in DB.
- Properties:
  - Performance: poor
  - Fault tolerance: poor
  - Scalability: poor
  - Semantics (consistency): great!

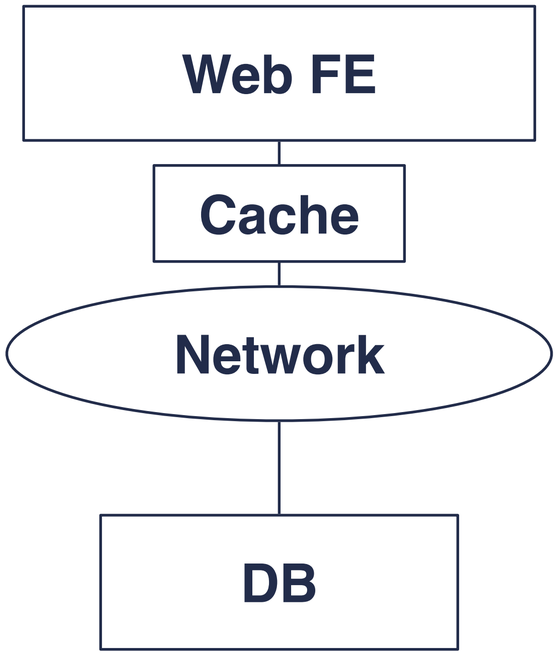
Let's improve performance first!

GOAL: REDUCE LATENCY



- Performance
  - Read?
  - Write?
- Fault tolerance
  - Availability?
  - Durability?
- Scalability?
- Semantics (consistency)?

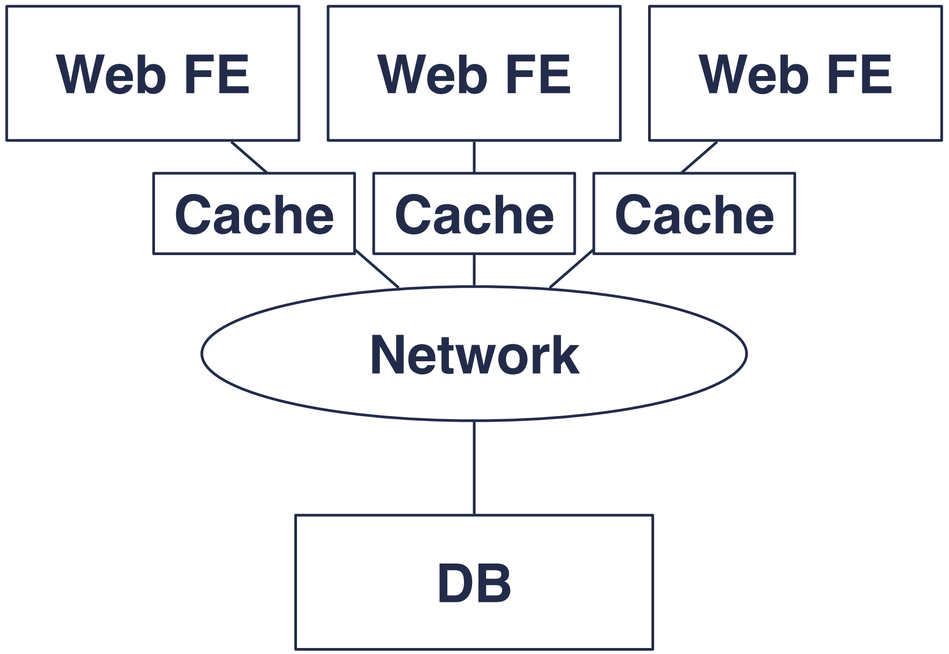
GOAL: REDUCE LATENCY



- Read latency: improved if working set fits in memory.
- Durability: depends on cache: good for write-through $$, poor for write-back $$.
- Write latency is opposite: good with write-back, poor with write-through.
- Consistency: good: you have 1 FE accesses DB, going through 1 $$, so behavior is equivalent to single machine.

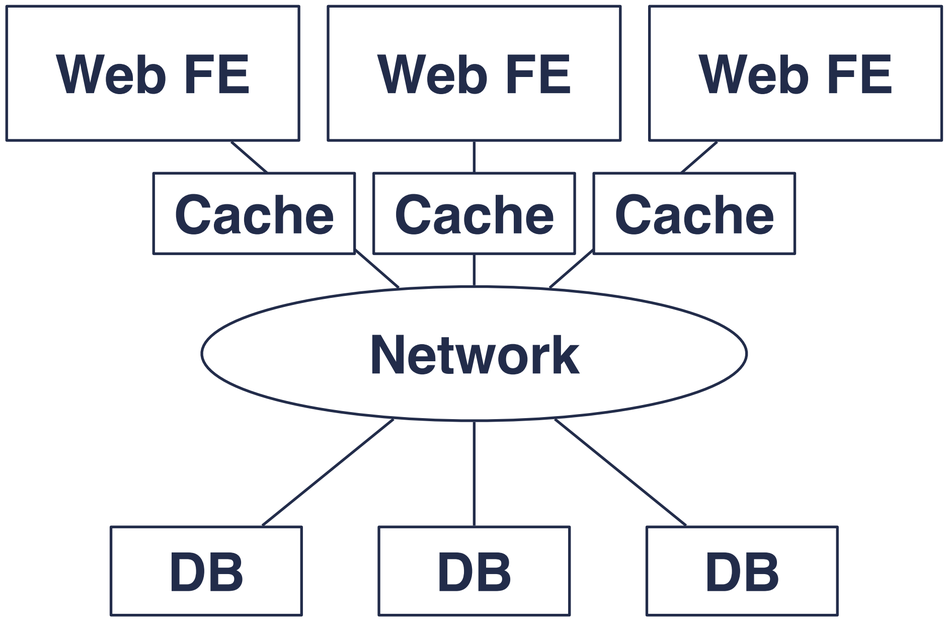
Let's deal with scalability first on FE and later on DB.

GOAL: SCALE OUT THE FE



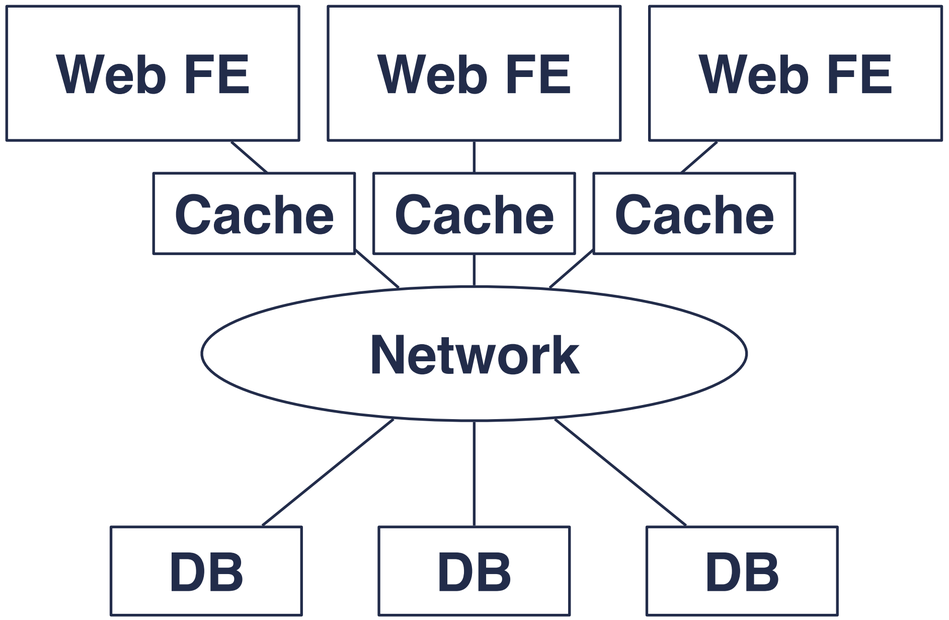
- Launch multiple FEs. Each has its own local cache, which we'll assume is write-through.
- Properties:
  - Performance?
  - Fault tolerance?
  - Scalability?
  - Semantics?

GOAL: SCALE OUT THE FE



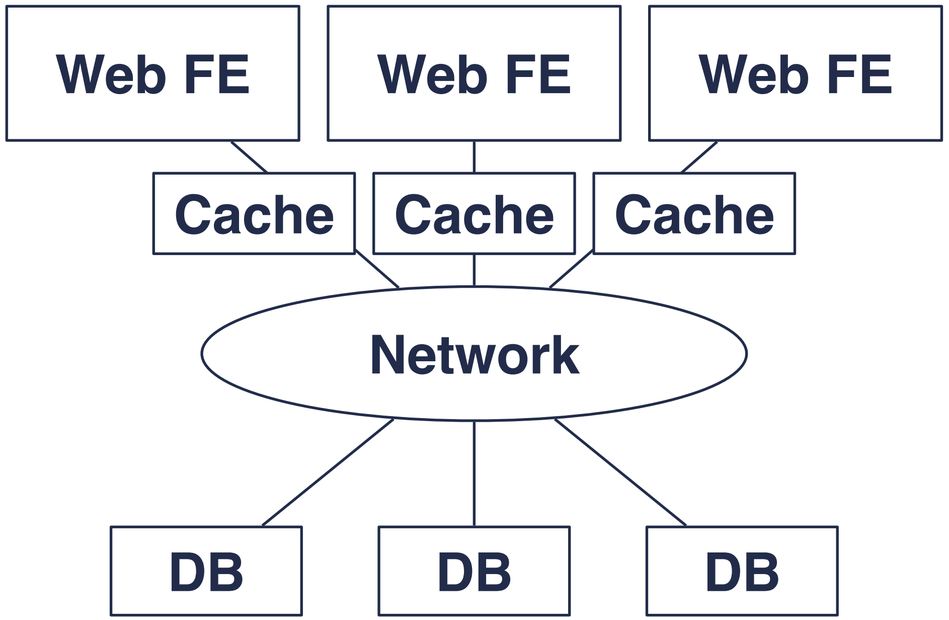
- Launch **identical** replicas of the DB server, each with its disk. All replicas hold all data, writes go to all.
- Properties:
  - Performance?
  - Fault tolerance?
  - Scalability?
  - Semantics?

GOAL: FAULT TOLERANCE FOR DB



- Writes now need to propagate to all replicas. So they are much slower! Even if done in parallel, because FE now needs to wait for the slowest of DB replicas to commit (assuming write-through cache, which offers the best durability).
- All replicas must see all writes IN THE SAME ORDER! If order is exchanged, they can quickly go “out of sync”! So lots more consistency issues.

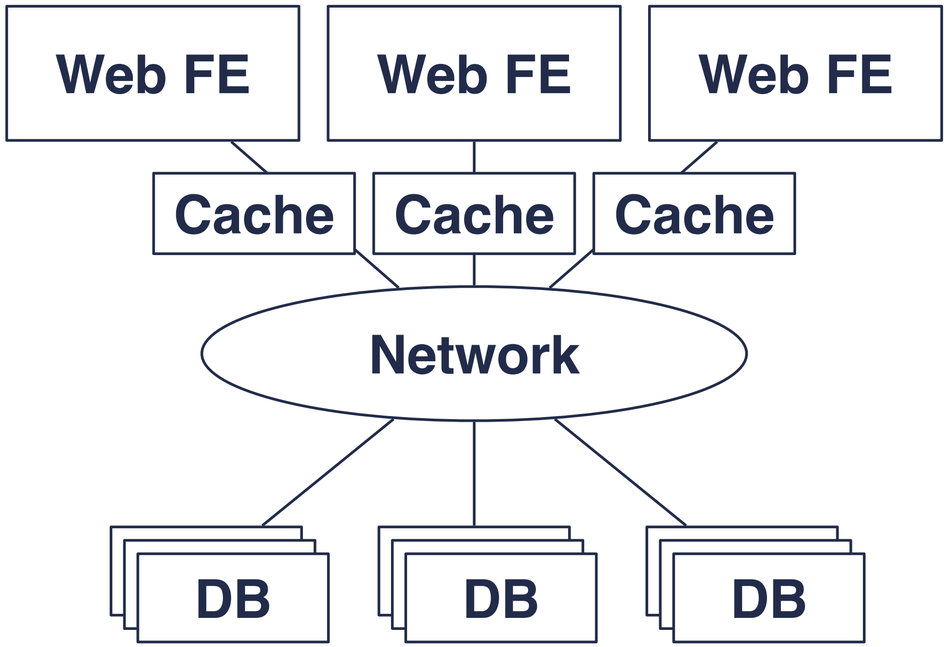
GOAL: FAULT TOLERANCE FOR DB



Let's deal with DB scalability.

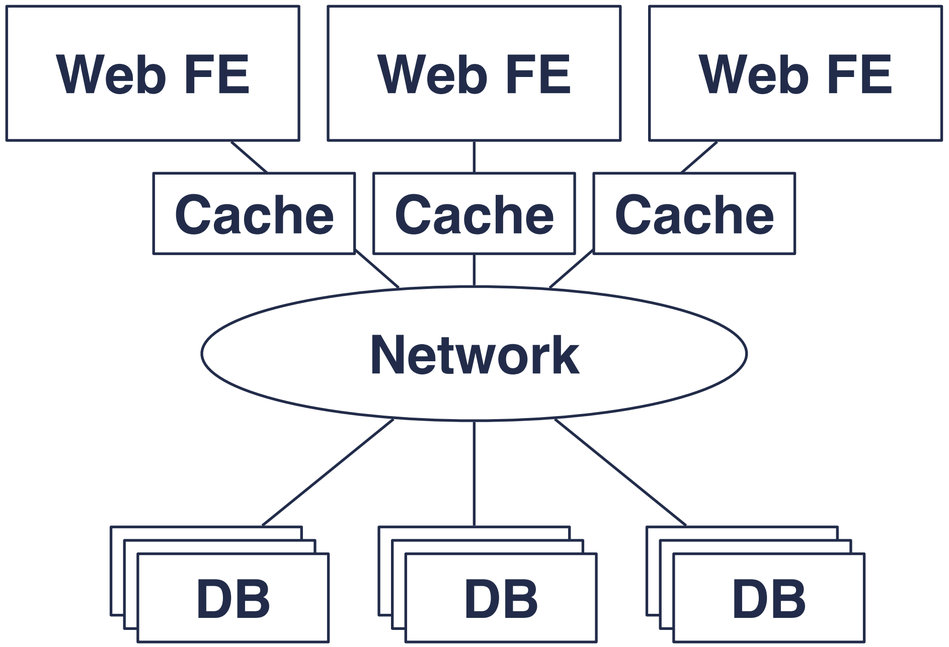
- There are also availability issues. If you require all the replicas to be available when a write is satisfied (for durability), availability goes DOWN! Consensus protocols, which work on a majority of replicas, address this.
- Another consistency challenge: how are reads handled? If you read from one replica, which one should you read given that updates to the item you're interested in might be in flight as you attempt to read it? We'll address these issues in future lectures by structuring the replica set.

LAST GOAL: SCALE OUT THE DB



— Partition the database into multiple shards, replicate each multiple times for fault tolerance. Requests for different shards go to different replica groups.

LAST GOAL: SCALE OUT THE DB



- New challenges that arise:
  - How should data be sharded? Based on users, on some property of the data?
  - How should different partitions get assigned to replica groups? How do clients know which servers serve/store which shards?
  - If the FE wants to write/read multiple entries in the DB, how can it do that atomically if they span multiple shards? If different replica groups need to coordinate to implement atomic updates across shards, won't that hinder scalability?

TAKEAWAYS

- Scalability, fault tolerance, consistency, and performance are difficult goals to achieve together.
- Solving them requires rigorous protocols and system architectures.

- This class teaches such protocols and architectures, plus how they are incorporated in real systems.
- Next class: **MapReduce**

ACKNOWLEDGEMENT

THIS COURSE IS DEVELOPED HEAVILY BASED ON INFLUENTIAL DISTRIBUTED SYSTEM COURSES INCLUDING UIUC DISTRIBUTED SYSTEMS CS425, MIT 6.5840 DISTRIBUTED SYSTEMS AND COLUMBIA DISTRIBUTED SYSTEMS FUNDAMENTALS. MANY THANKS TO PROF. INDRANIL GUPTA, PROF. ROBERT MORRIS AND PROF. ROXANA GEAMBASU FOR GENEROUSLY SHARING THEIR MATERIALS AND TEACHING INSIGHTS.