

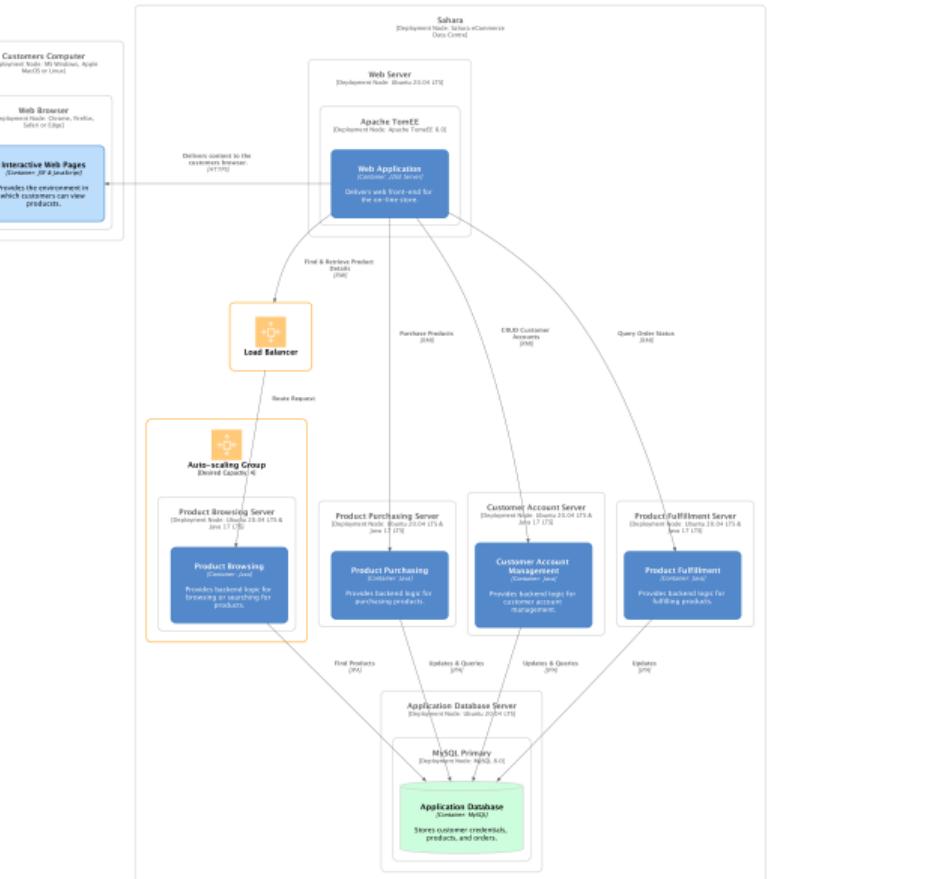
# Distributed Computing II

*Software Architecture*

Brae Webb

March 27, 2023

## Previously in CSSE6400...



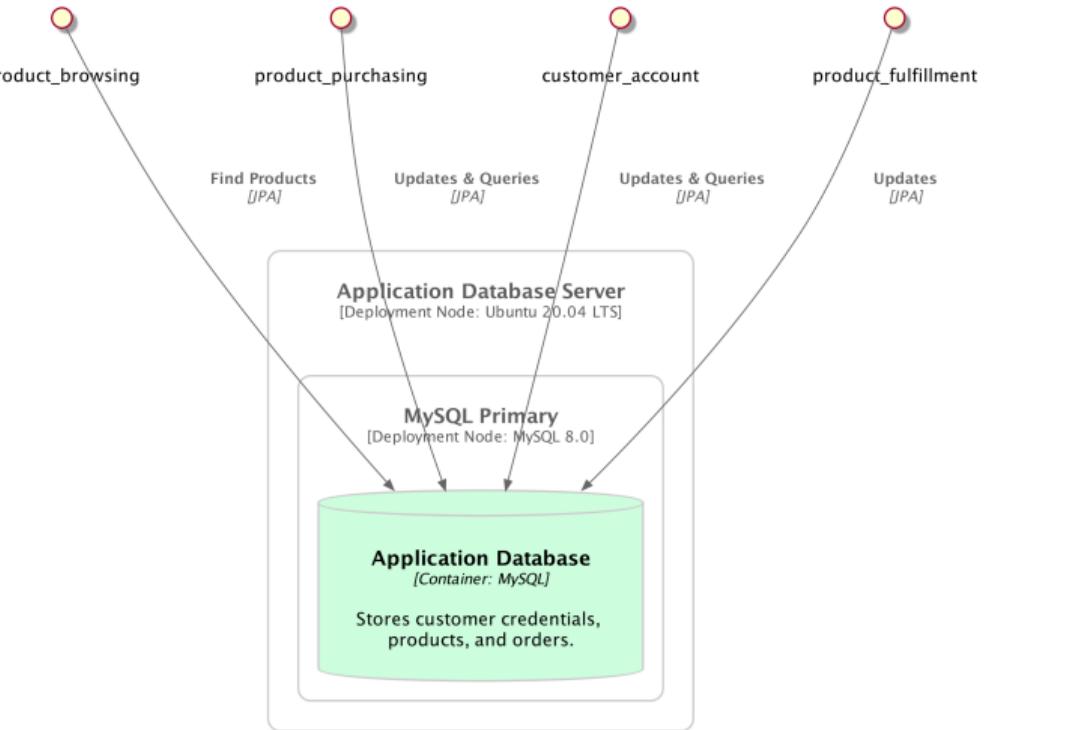
- We scaled a stateless service.
- It was stateless as it didn't require persistent data.
- This is normally easy to do.

*Question*

What is the *problem*?

The database

## Database



- The database has state, persistent data.
- This is much harder to scale.

*Disclaimer*

This is *not* a database course.

## Advanced Database Systems (INFS3200)

**Course level**

Undergraduate

**Faculty**

Engineering, Architecture &amp; Information Technology

**School**

Info Tech &amp; Elec Engineering

**Units**

2

**Duration**

One Semester

**Class contact**

2 Lecture hours, 1 Tutorial hour, 1 Practical or Laboratory hour

**Incompatible**

INF57907

**Prerequisite**

INFS2200

**Assessment methods****Current course offerings****Course offerings**   **Location**   **Mode**   **Course Profile**Semester 1, 2022 St Lucia Internal [COURSE PROFILE](#)Semester 1, 2022 External External [COURSE PROFILE](#)

Semester 2, 2022 External External PROFILE UNAVAILABLE

Semester 2, 2022 St Lucia Internal PROFILE UNAVAILABLE

Please Note: Course profiles marked as not available may still be in development.

**Course description**

Distributed database design, query and transaction processing, data integration, data warehousing, data cleansing, management of spatial data, and data from large scale distributed devices.

**Archived offerings****Course offerings**   **Location**   **Mode**   **Course Profile**Semester 1, 2021 St Lucia Flexible Delivery [COURSE PROFILE](#)Semester 1, 2021 External External [COURSE PROFILE](#)Semester 2, 2021 External External [COURSE PROFILE](#)Semester 2, 2021 St Lucia Internal [COURSE PROFILE](#)Semester 1, 2020 St Lucia Internal [COURSE PROFILE](#)

This is a database course.

*Question*

How do we fix database scaling issues?

*Question*

How do we fix database scaling issues?

*Answer*

- Replication

*Question*

How do we fix database scaling issues?

*Answer*

- Replication
- Partitioning

*Question*

How do we fix database scaling issues?

*Answer*

- Replication
- Partitioning
- Independent databases

*Question*

How do we fix database scaling issues?

*Answer*

- *Replication*
- Partitioning
- Independent databases

*Question*

What is *replication*?

*Definition 1.* Replication

Data copied across multiple different machines.



product_id	name	stock	price
1234	Nicholas Cage Reversible Pillow	10	\$10.00
4321	Lifelike Elephant Inflatable	5	\$50.00



product_id	name	stock	price
1234	Nicholas Cage Reversible Pillow	10	\$10.00
4321	Lifelike Elephant Inflatable	5	\$50.00

*Definition 2.* Replica

Database node which stores a copy of the data.

*Question*

What are the advantages of *replication*?

*Question*

What are the advantages of *replication*?

*Answer*

- Scale out our database to cope with *load*.

*Question*

What are the advantages of *replication*?

*Answer*

- Scale out our database to cope with *load*.
- Provide *fault tolerance* from a single database instance failure.

*Question*

What are the advantages of *replication*?

*Answer*

- Scale out our database to cope with *load*.
- Provide *fault tolerance* from a single database instance failure.
- Locate databases *closer to end-users*.
- Scalability
- Reliability
- Performance

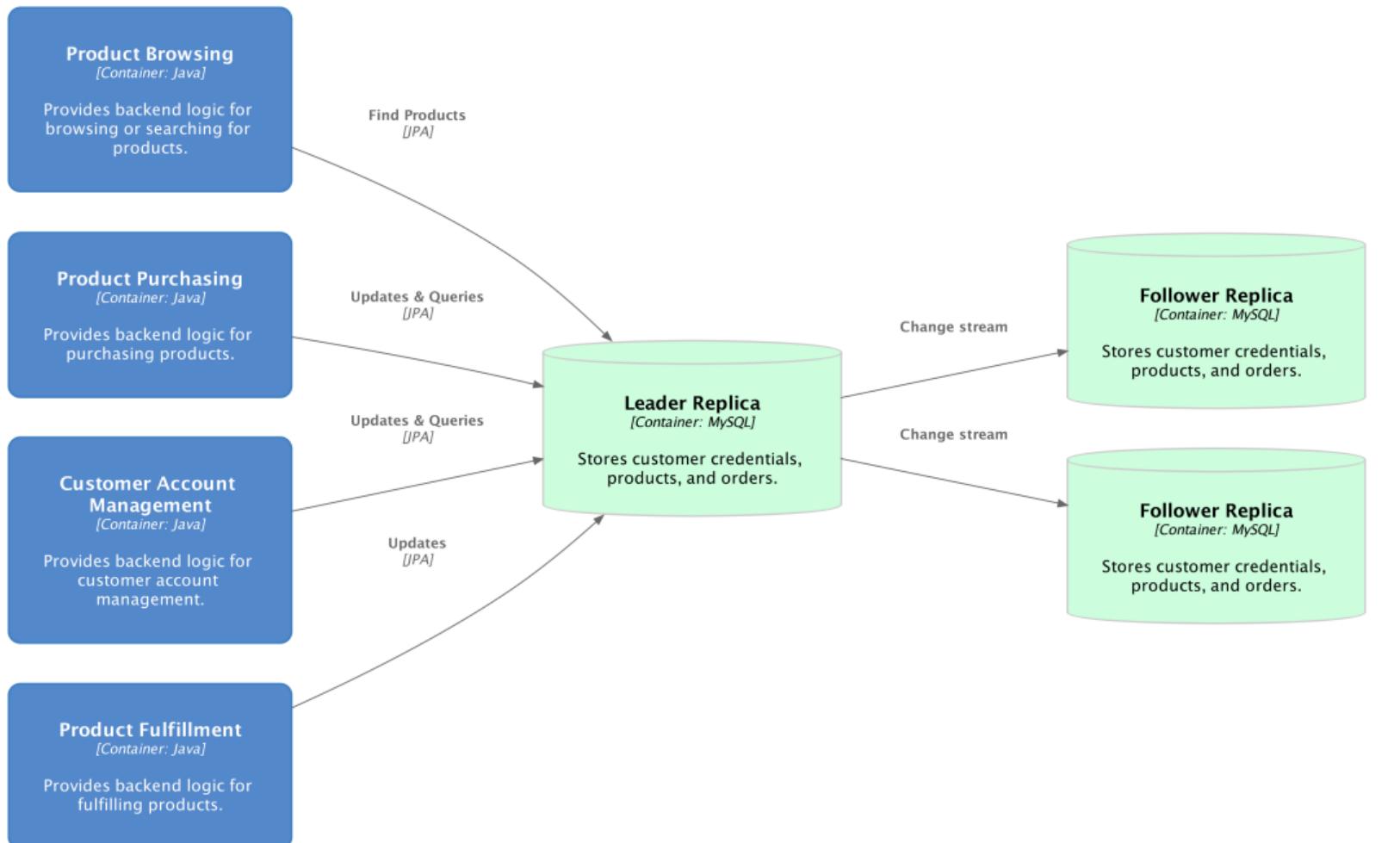
*Question*

How do we replicate our data?

- Easy without updates, just copy it.
- Updates, or writes, must propagate changes.

*First approach*

Leader-follower Replication



- Leader-follower is the most common implementation.
- Multiple followers, only one leader.

### *Leader-based Replication*

On write Writes sent to leader, change is propagated via change stream.

### *Leader-based Replication*

On write Writes sent to leader, change is propagated via change stream.

On read Any replica can be queried.



- Built-in to PostgreSQL, MySQL, MongoDB, RethinkDB, and Espresso.
- Can be added to Oracle and SQL Server.

*Propogating changes*

*Synchronous vs. Asynchronous*







### *Synchronous propagation*

- Writes must propagate to *all followers* before being successful.

### *Synchronous propagation*

- Writes must propagate to *all followers* before being successful.
- *Any* replica goes down, *all* replicas are un-writable.

### *Synchronous propagation*

- Writes must propagate to *all followers* before being successful.
- *Any* replica goes down, *all* replicas are un-writable.
- Writes must *wait* for propagation to all replicas.

### *Asynchronous propagation*

- Writes *don't* have to *wait* for propagation.

### *Asynchronous propagation*

- Writes *don't* have to *wait* for propagation.
- If the leader goes down before propagating, the *write is lost*.

### *Asynchronous propagation*

- Writes *don't* have to *wait* for propagation.
- If the leader goes down before propagating, the *write is lost*.
- Replicas can have out-dated or *stale* data.

### *Definition 3.* Replication Lag

The time taken for replicas to update *stale* data.



The time it takes for the change to the name of the product to update across all followers



The purple part is replication lag

*Eventually, all replicas must become consistent*

The system is *eventually consistent*

- If writes stop for long enough
- Eventually is intentionally ambiguous

*Eventual Consistency*

Problems?



**Brae Webb**  
@braewebb



**Brae Webb**  
@braewebb

Name:	<input type="text" value="Brae"/>
<input type="button" value="Cancel"/>	<input type="button" value="Save"/>



**Brae Webb**  
@braewebb

Name:



**Brae Webb**  
@braewebb

- Read user details
- Decide I don't like by name
- Update name
- Read user details



*Definition 4.* Read-your-writes Consistency

Users always see the updates that *they have made*.

Doesn't care what other users see



**Brae Webb**  
@braewebb

My fist post



**Brae Webb**  
@braewebb

My fist post



**Brae Webb**  
@braewebb

My first post



**Brae Webb**  
@braewebb

My fist post



**Brae Webb**  
@braewebb

My first post



**Brae Webb**  
@braewebb

My fist post



### *Definition 5.* Monotonic Reads

Once a user reads an updated value, they don't later see the old value.

User doesn't travel back in time

## *Summary*

- Leader-follower databases allow *reads to scale* more effectively.
- Asynchronous propagation weakens consistency to *eventually consistent*.
- Leader-follower databases still have a *leader write bottle-neck*.

*Second approach*

Multi-leader Replication



### *Why multi-leader?*

- If you have multiple leaders, you can write to any, allowing *writes to scale*.

### *Why multi-leader?*

- If you have multiple leaders, you can write to any, allowing *writes to scale*.
- A leader going down doesn't prevent writes, giving *better fault-tolerance*.
- Available via extensions in most databases, often not natively supported.
- Best to avoid where possible.

*Question*

What might go wrong?

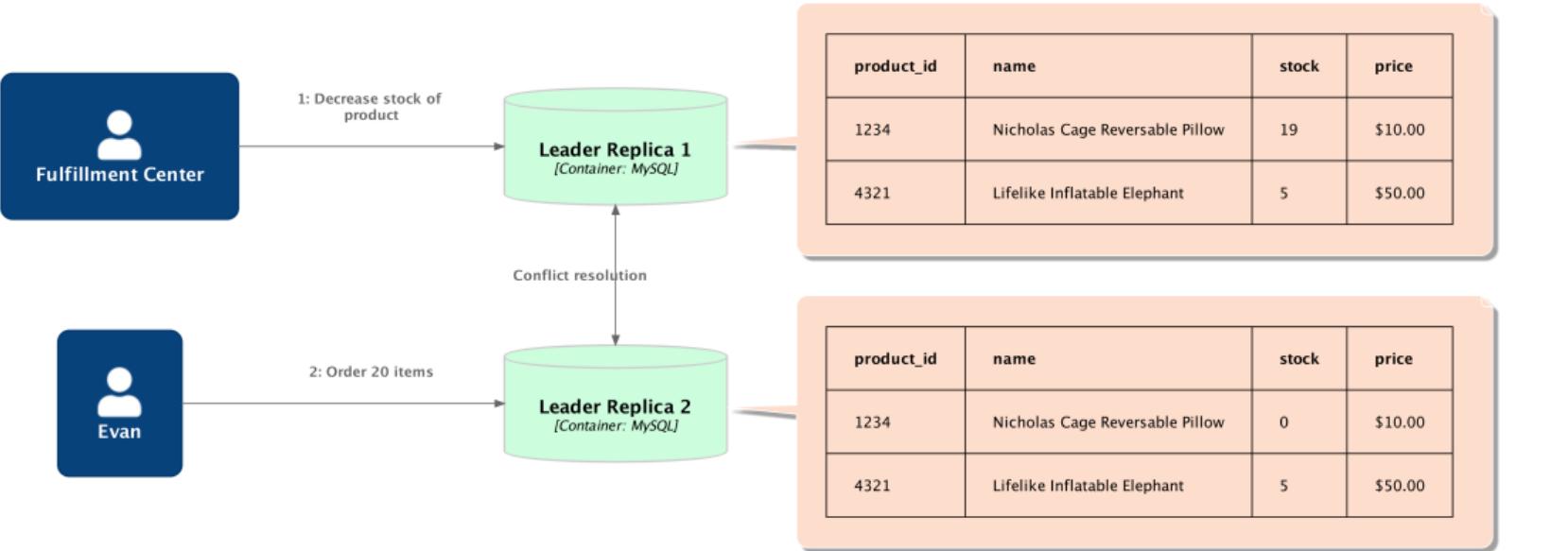
*Question*

What might go wrong?

*Answer*

Write conflicts

Write conflicts require the conflict to be resolved.



-1 Pillows? How do we resolve this?

*Where possible*

Avoid write conflicts



*Where impossible*

Convergence

### *Convergence Strategies*

- Assign each *write* a unique ID.

### *Convergence Strategies*

- Assign each *write* a unique ID.
- Assign each *leader replica* a unique ID.

### *Convergence Strategies*

- Assign each *write* a unique ID.
- Assign each *leader replica* a unique ID.
- Custom resolution logic.



## *Resolving Conflicts*

**On Write** When a conflict is first noticed, take proactive resolution action.

**On Read** When a conflict is next read, ask for a resolution.

- Bucardo allows a perl script for on write resolution.
- CouchDB prompts reads to resolve the conflict.

### *Third Approach*

## Leaderless Replication

- Early distributed databases were leaderless.
- Resurgence after Amazon created Dynamo.
- Dynamo is an internal service and not DynamoDB.
- Riak, Cassandra, and Voldemort are leaderless databases.



Reads and writes can be written to any node.

*How do they work?*

Each read/write is sent to *multiple* replicas.





At least one of the reads has the updated value

*How are changes propagated?*

- Read Repair

*How are changes propagated?*

- Read Repair
- Anti-entropy Process

*Question*

How do we know it's consistent?



*Question*

How do we know it's consistent?

*Question*

How do we know it's consistent?

*Answer*

Quorum Reads and Writes

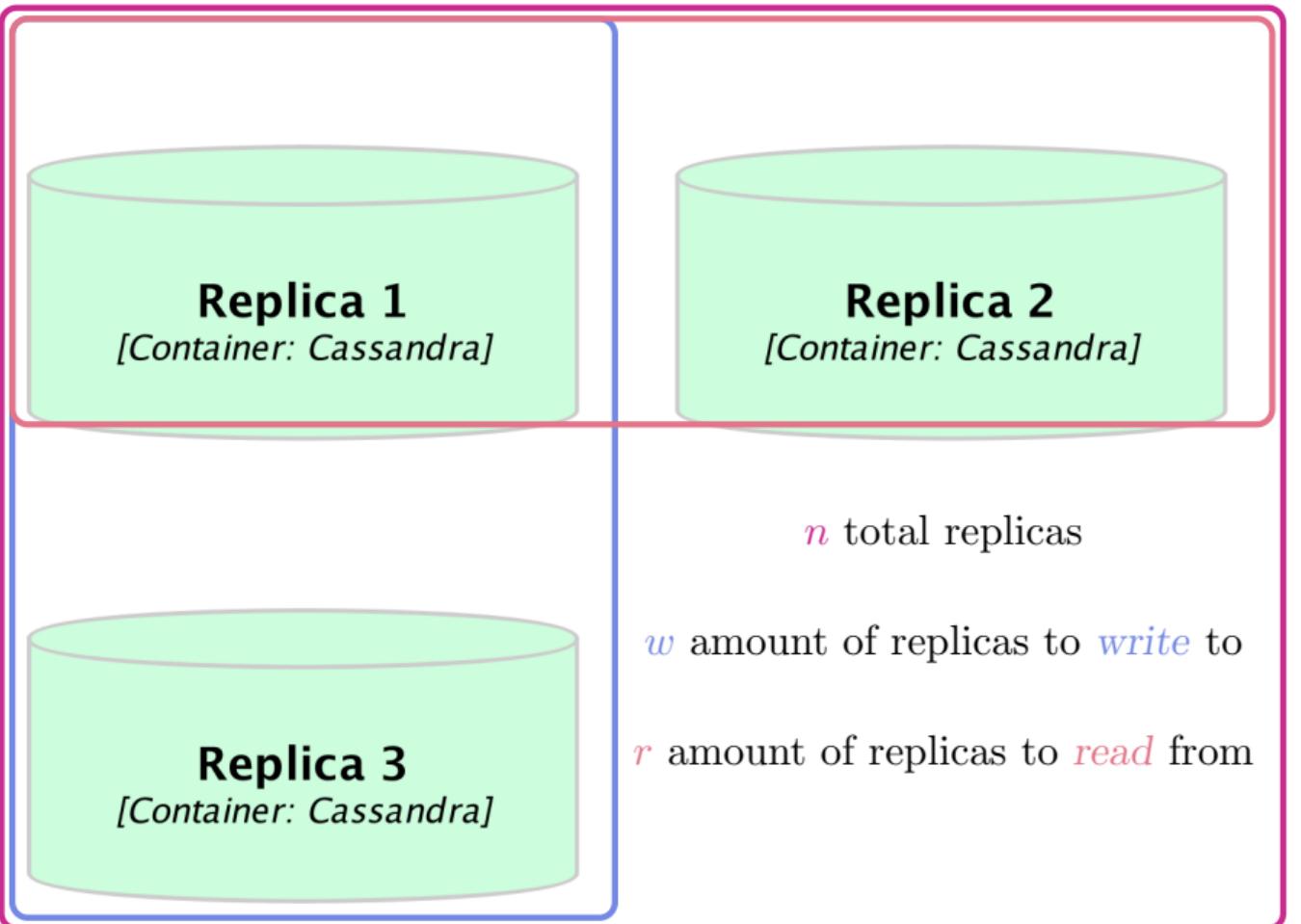
## *Quorum Consistency*

$$w + r > n$$

$n$  total replicas

- $w$  amount of replicas to *write* to
- $r$  amount of replicas to *read* from

The nodes read from must overlap with the nodes written to



$n$  total replicas

$w$  amount of replicas to *write* to

$r$  amount of replicas to *read* from

## *Summary*

- *Replication* copies data to multiple replicas.

### *Summary*

- *Replication* copies data to multiple replicas.
- *Leader-based* replication is most common and simplest.

## *Summary*

- *Replication* copies data to multiple replicas.
- *Leader-based* replication is most common and simplest.
- Replication introduces *eventual consistency*.

## *Summary*

- *Replication* copies data to multiple replicas.
- *Leader-based* replication is most common and simplest.
- Replication introduces *eventual consistency*.
- *Multi-leader* replication scales writes as well as reads but introduces *write conflicts*.

## *Summary*

- *Replication* copies data to multiple replicas.
- *Leader-based* replication is most common and simplest.
- Replication introduces *eventual consistency*.
- *Multi-leader* replication scales writes as well as reads but introduces *write conflicts*.
- *Leaderless* replication is another approach which keeps the problems of multi-leader.

*Question*

How do we fix database scaling issues?

*Question*

How do we fix database scaling issues?

*Answer*

- *Replication*
- Partitioning
- Independent databases

*Question*

How do we fix database scaling issues?

*Answer*

- Replication
- *Partitioning*
- Independent databases

### *Definition 6.* Partitioning

Split the data of a system onto multiple nodes, these nodes are *partitions*.

Also called shardes, regions, tablets, etc.



product_id	name	stock	price
4321	Lifelike Elephant Inflatable	5	\$50.00



product_id	name	stock	price
1234	Nicholas Cage Reversible Pillow	10	\$10.00

- Pioneered in the 1980s
- Allow scalability of large data, not just large load.
- Partitioning is normally combined with replication.

*Question*

How should we decide which data is stored where?



An example partitioning based on primary key, student ID

*Question*

What is the problem with this?

*Question*

What is the problem with this?

*Answer*

Over time some partitions become inactive,  
while others receive almost all load.

*Question*

How should we decide which data is stored where?

*Question*

How should we decide which data is stored where?

*Answer*

Maximize spread of requests, avoiding *skewing*.

*Question*

Have we seen this before?

*Question*

Have we seen this before?

*Answer*

Hashing?

Hash tables hash entries to maximize the spread between buckets.

*Question*

What is the problem with this?

*Question*

What is the problem with this?

*Answer*

Range queries are inefficient, i.e. get all  
students between s4444444 and s4565656

*Question*

How do we route queries?

Unlike stateless, only one node can process queries.

### *Query-insensitive Load Balancer*

Randomly route to any node, responsibility of the node to re-route to the correct node.



### *Query-sensitive Load Balancer*

A load balancer which understands which queries should be forwarded to which node.



### *Client-aware Queries*

Place the responsibility on clients to choose the correct node.



## *Summary*

- *Partitioning* splits data across multiple nodes.

## *Summary*

- *Partitioning* splits data across multiple nodes.
- A *consistent method* to chose which node is required.

## *Summary*

- *Partitioning* splits data across multiple nodes.
- A *consistent method* to chose which node is required.
- Partitioning by *primary key* can create *skewing*.

## *Summary*

- *Partitioning* splits data across multiple nodes.
- A *consistent method* to chose which node is required.
- Partitioning by *primary key* can create *skewing*.
- Partitioning by *hash* makes range queries less efficient.

## *Summary*

- *Partitioning* splits data across multiple nodes.
- A *consistent method* to chose which node is required.
- Partitioning by *primary key* can create *skewing*.
- Partitioning by *hash* makes range queries less efficient.
- Three approaches to *routing requests*.

*Disclaimer*

We have ignored the hard parts of replication.

*Question*

How do we fix database scaling issues?

*Question*

How do we fix database scaling issues?

*Answer*

- Replication
- *Partitioning*
- Independent databases

*Question*

How do we fix database scaling issues?

*Answer*

- Replication
- Partitioning
- *Independent databases*

Distributed state creates a lot of *complexity*

And when programmers have complexity, they create bugs

*When programmers are faced with complexity*

*They create abstractions*

*One key database abstraction*

## Transactions

Introduced by IBM System R in 1975

*Definition 7.* Transaction

A group of operations performed as if they were one.

What does as if it were one mean?

ACID

A tomic

C onsistent

I solated

D urable

### *The pushback*

NoSQL and microservice architectures pushed back against transactions.

- Transactions were used fairly universally for a long time.
- Push back occurred when people decided they weren't scalable.

*For more on transactions*



*Summary*

- Replications

## *Summary*

- Replications

- Leader-based, multi-leader, and leaderless

## *Summary*

- Replications

- Leader-based, multi-leader, and leaderless
- Eventual consistency

## *Summary*

- Replications

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

### *Summary*

- Replications

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

- Partitioning

### *Summary*

- Replications

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

- Partitioning

- Consistent method to pick nodes for data

## *Summary*

- Replications

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

- Partitioning

- Consistent method to pick nodes for data
- Avoiding skewing

## *Summary*

- Replications

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

- Partitioning

- Consistent method to pick nodes for data
- Avoiding skewing

- Transactions

## *Summary*

- Replications

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

- Partitioning

- Consistent method to pick nodes for data
- Avoiding skewing

- Transactions

- ACID properties

## *Summary*

- Replications

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

- Partitioning

- Consistent method to pick nodes for data
- Avoiding skewing

- Transactions

- ACID properties
- Pushback causing headaches