# 15-319 / 15-619
# Cloud Computing

Overview 10

# Overview

- **Last week's reflection**
  - OLI Unit 4 Module 15: Case Studies: Distributed File Systems
  - OLI Unit 4 Module 16: Case Studies: NoSQL Database
  - OLI Unit 4 Module 17: Case Studies: Cloud Object Storage
  - Quiz 8
  - Online Programming Exercise for Multi-Threading
- **This week's schedule**
  - Project 3.3
  - OLI Unit 5 Module 18: Introduction to Distributed Programming for the Cloud
  - Quiz 9 due on **Friday, Apr. 9th**
- **Team Project, Twitter Analytics**
  - Phase 2 - Live test: Next Sunday, Apr. 18th

# This Week

- OLI Unit 5 Module 18: Introduction to Distributed Programming for the Cloud
- **Quiz 9** - Friday, Apr. 9th
- **Project 3.3** - Sunday, Apr. 11th
  - *Task 1:* Implement a Strong Consistency Model for distributed data stores
  - *Task 2:* Implement a Strong Consistency Model cross-region data stores
  - *Bonus:* Implement an Eventual Consistency Model
- **Team Project, Twitter Analytics**
  - Phase 2 - Live test: Next Sunday, Apr. 18th

# Individual Projects

- Done
  - P3.1: Files v/s Databases

- **Now**
  - Introduction to multithreaded programming in Java
  - Introduction to consistency models
  - **P3.3: Replication and Consistency models**

# Scale of Data is Growing

International Data Corporation predicts massive data increases:

- **From:** 33 zettabytes in 2018

- **To:** 160 zettabytes in 2025.
  - appx. 50% of which will be stored in the public cloud!

For context, 1 zettabyte is *1 trillion* gigabytes. And much of this data will be consumed in real-time.

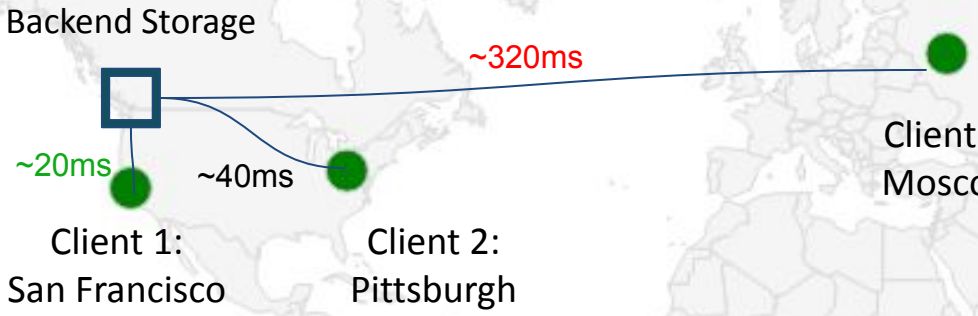
# Users are Global

- Information has physical limitations on speed of transfer (Speed of light)
- Inherent latencies
  - Especially for real-time data access, speed is critical!

# Typical End-To-End Latency

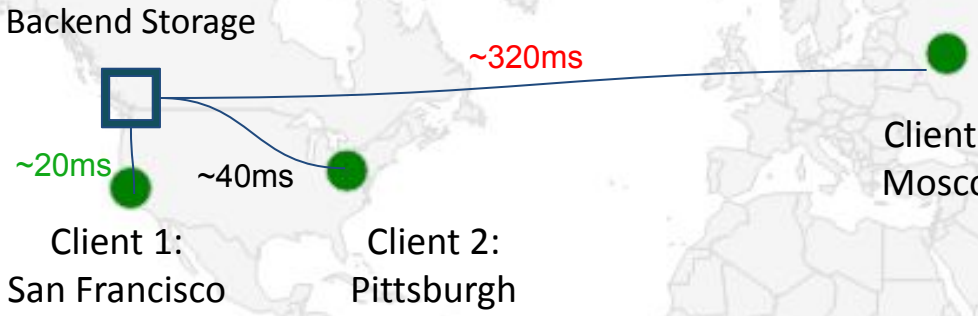
- A client sends a request to a server
  - Message takes time to physically reach the server
    - **(Network latency)**
- Server receives the request and responds
  - Server has to read incoming packets and responds
    - **(IO or Disk latency)**
  - Message takes time to physically reach the client
    - **(Network latency)**

# Latency with a Single Backend




Min Latency: 20ms
Max Latency: 320ms
Average Latency: 126ms

# Latency with a Single Backend




Means only users in the United States will use your service!

# How do you give users the same experience across the globe?

# Option 1: Global Replication





Min Latency: 20ms
Max Latency: 40ms
Average Latency: 26.6ms

# Option 2: Proximity Replication


Min Latency: 20ms
Max Latency: 20ms
Average Latency: 20ms

# You can’t keep replicating forever

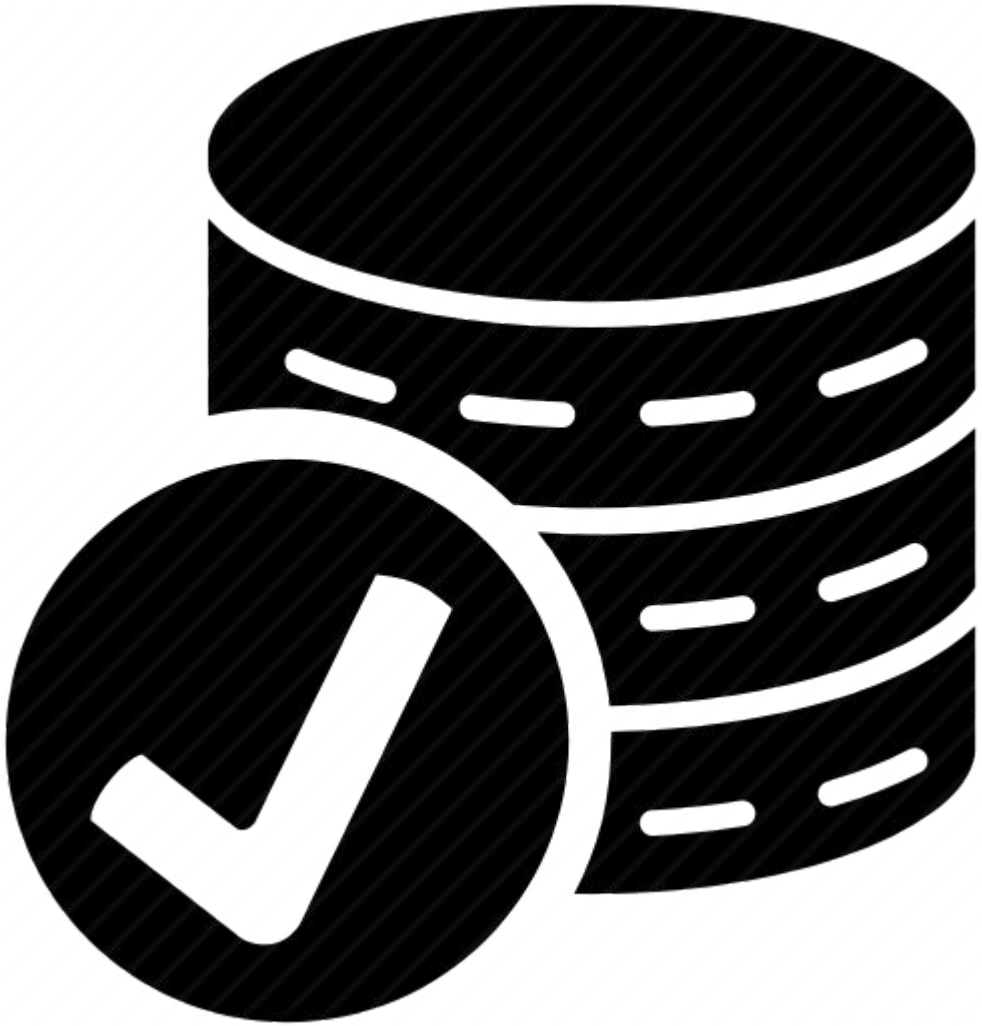
Replication has scalability limitations

# Cost can be a limiting factor

- Since we need to run multiple databases, we incur the following costs:
  - (num replicas) * time * database cost
    - AWS RDS: (num replicas) * hours * $0.226
  - (num replicas) * data * cost per GB
    - AWS RDS: (num replicas) * data (per 10 GB) * $1.15
  - Cost grows quickly relative to replica count!

With database replication you have to consider cost, but what else?

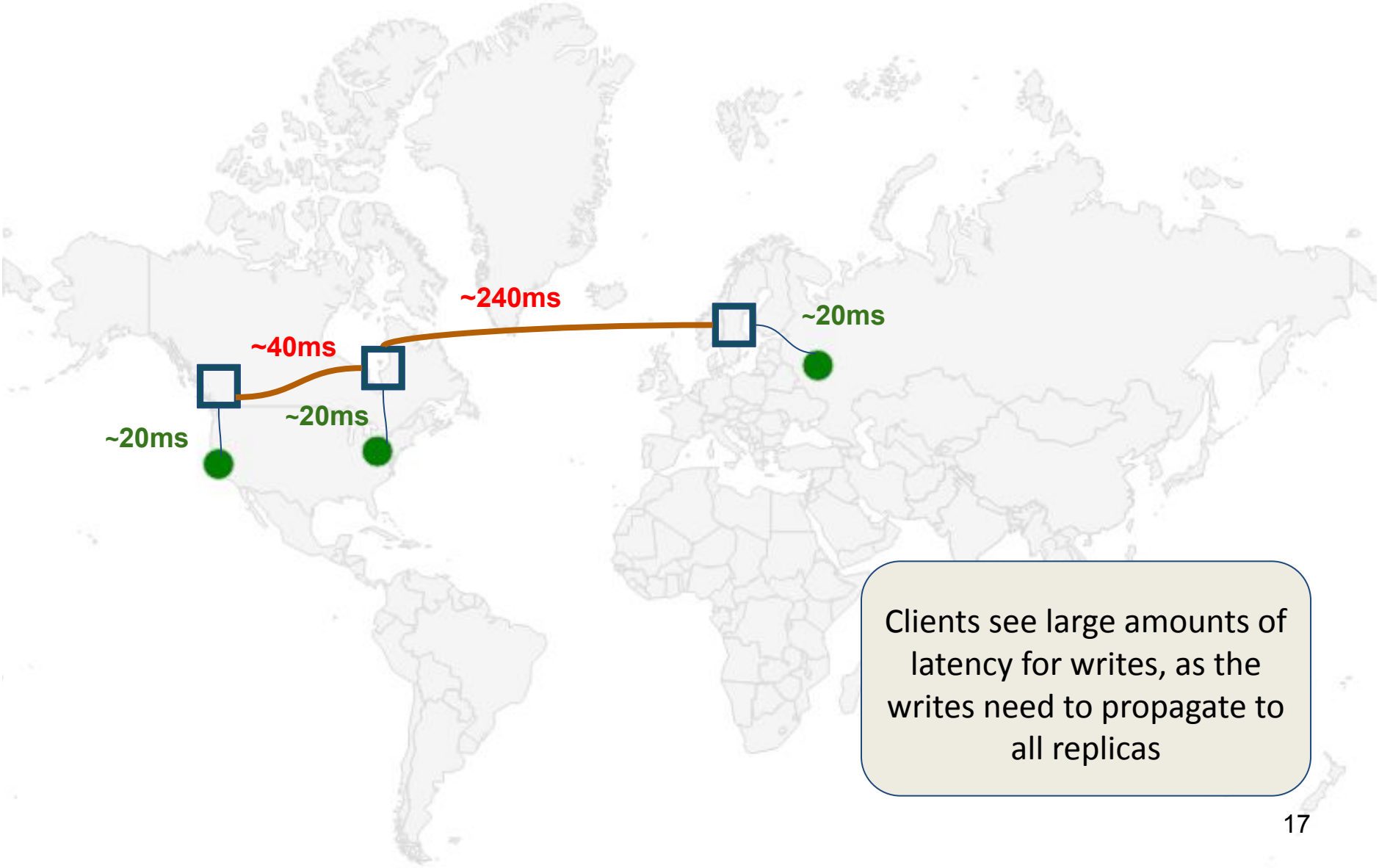
# Data Consistency



# Database Reads


Read operations are sent to the closest replica to minimize latency

# Database Writes




Clients see large amounts of latency for writes, as the writes need to propagate to all replicas

# Replication Reads and Writes

- Read operations are *fast*
  - All clients have a replica close to them to access
- Write requests are *slow*
  - Write requests must update all the replicas
  - If a certain key has multiple write requests, newer write requests may have to wait for older requests to complete

# Pros and Cons of Replication

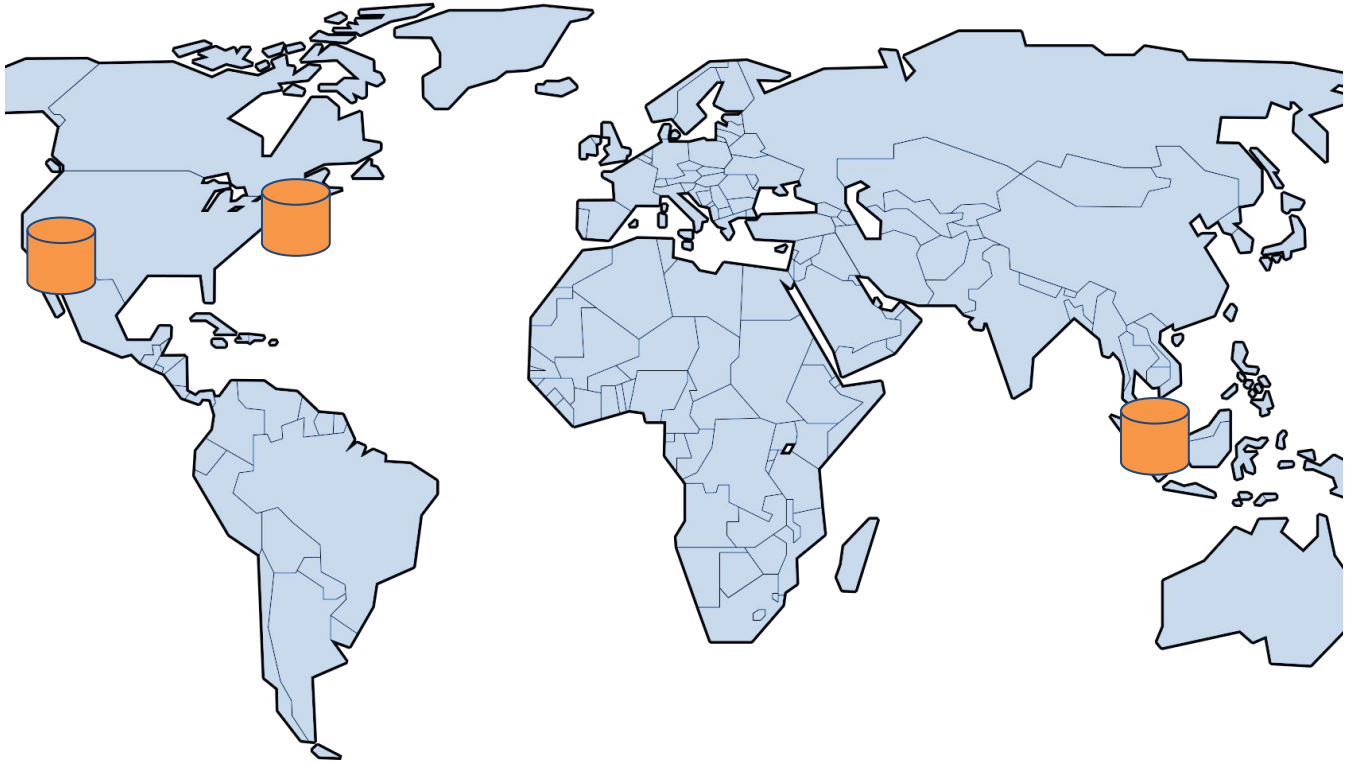
- **Advantages**
  - Low latency for reads
  - Reduce the workload of a single backend server
  - Handle failures of nodes by rerouting to alternative backup replicas
- **Disadvantages**
  - Requires more storage capacity and cost
  - Updates are significantly slower
  - Changes must reflect on all datastores (using various consistency models)

# Data Consistency Models

- Data consistency across replicas is important
  - Five consistency levels (explained in primers):
    - Strict
    - Strong (Linearizability)
    - Sequential
    - Causal
    - Eventual Consistency

- **This week's project!**

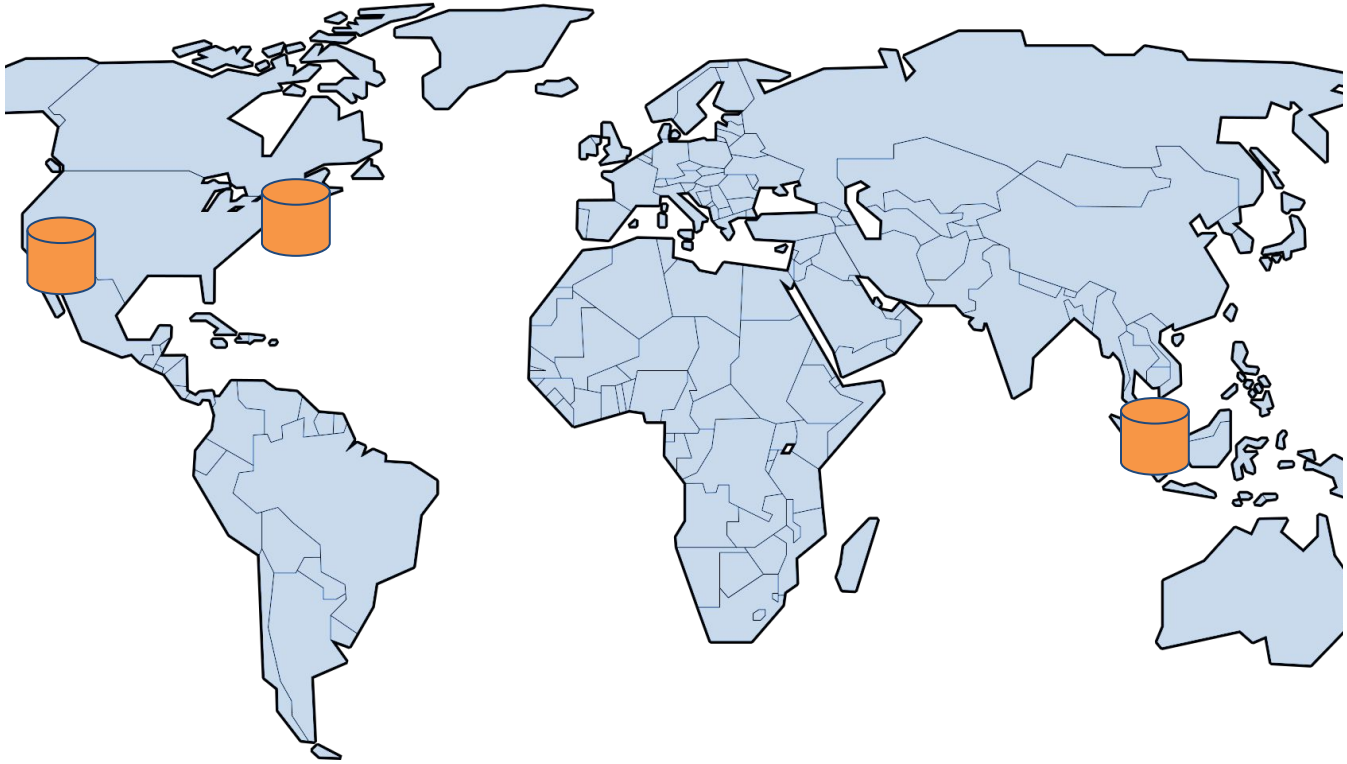
# Data Consistency Example: Consider a Bank



| Account | Balance |
|---|---|
| xxxxx-4437 | $100 |

# Bad Example

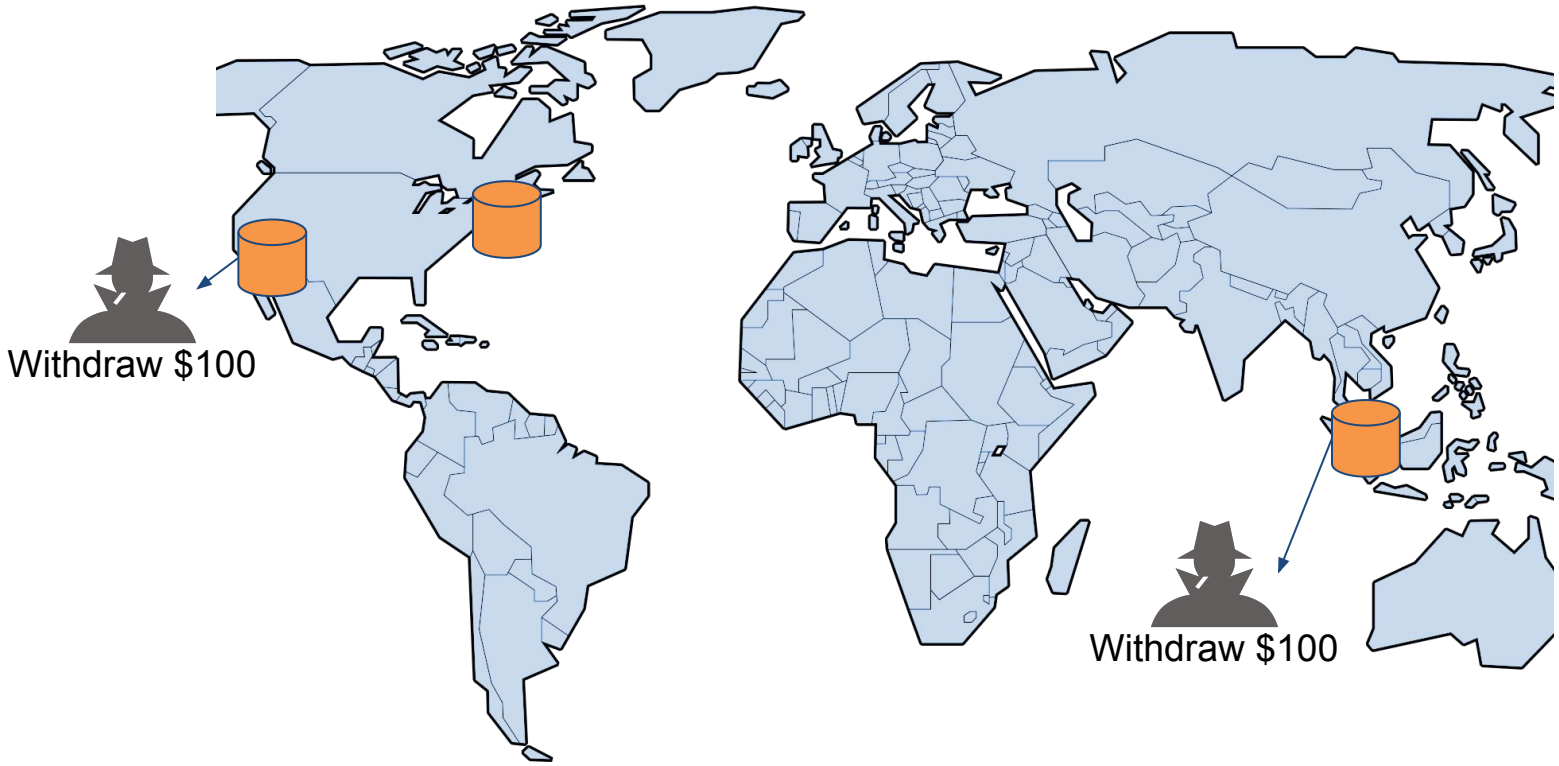
## Allow concurrent writes



| Account | Balance |
| --- | --- |
| xxxxx-4437 | $100 |

# Bad Example

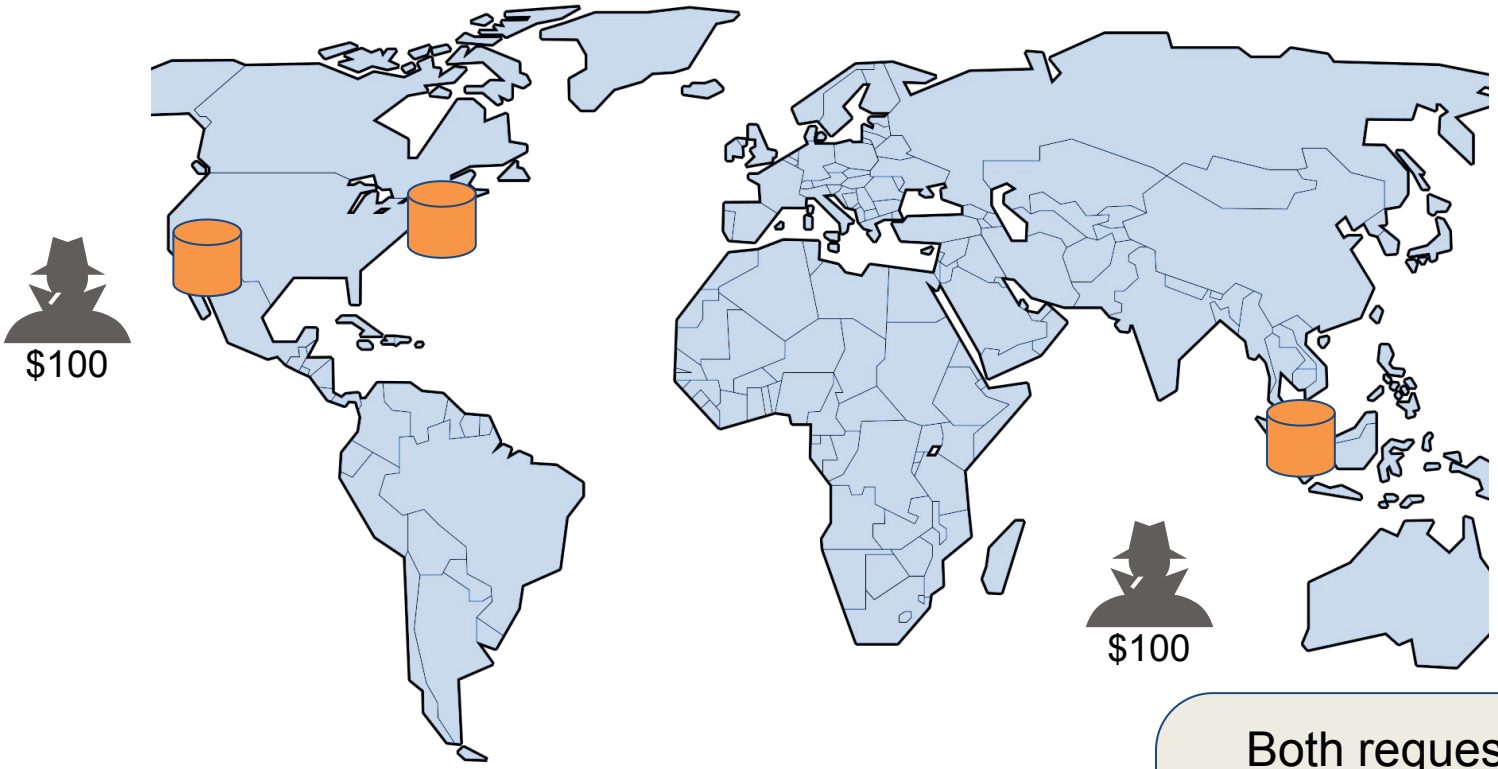
## Allow concurrent writes




| Account | Balance |
|---|---|
| xxxxx-4437 | $100 |

# Bad Example

## Allow concurrent writes

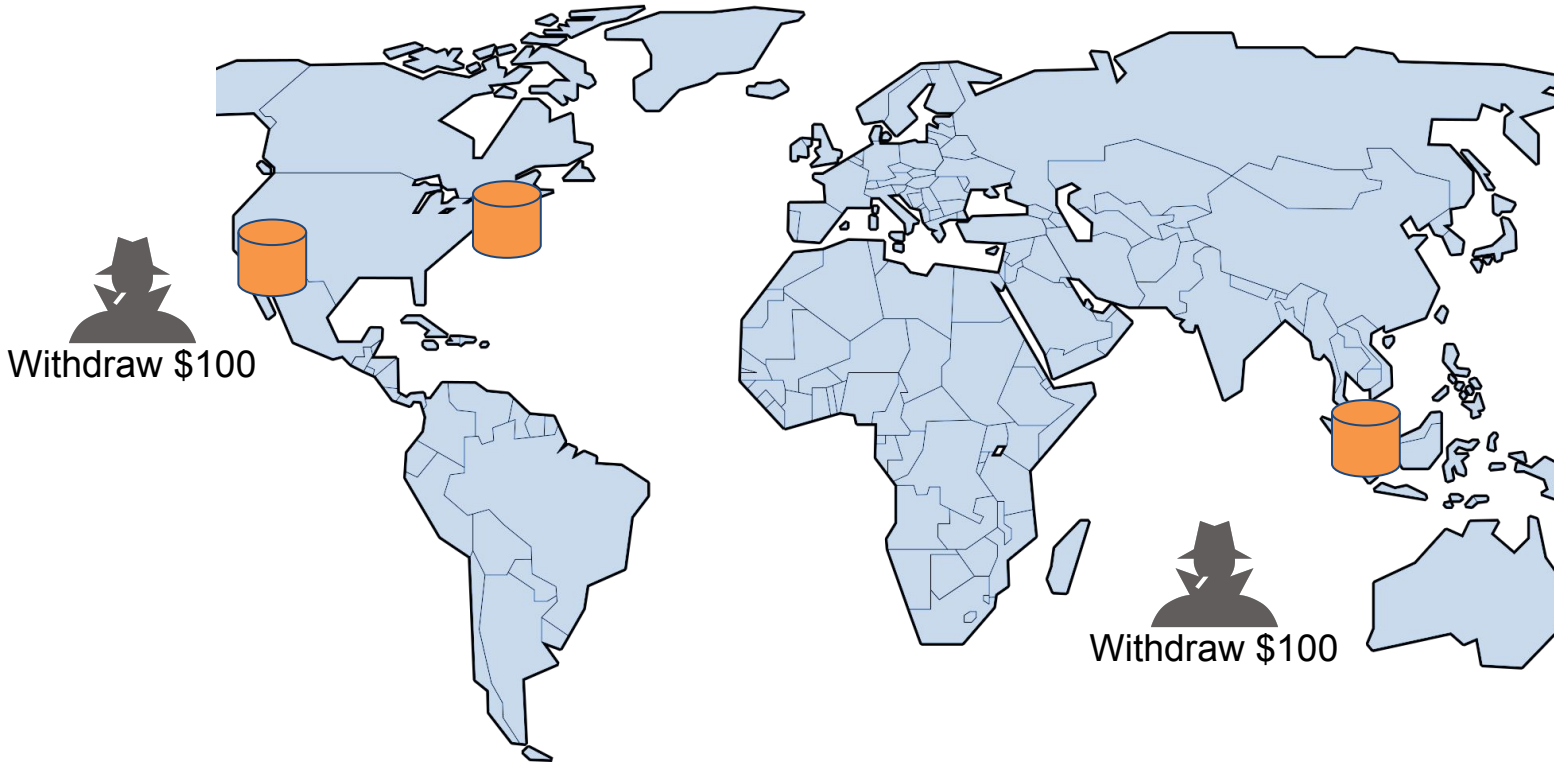



| Account | Balance |
|---|---|
| xxxxx-4437 | **$0** |

Both requests are processed concurrently, and we lose $100 as both are accepted

# Good Example

## Global Locking




| Account | Balance |
|---|---|
| xxxxx-4437 | **$100** |

# Good Example

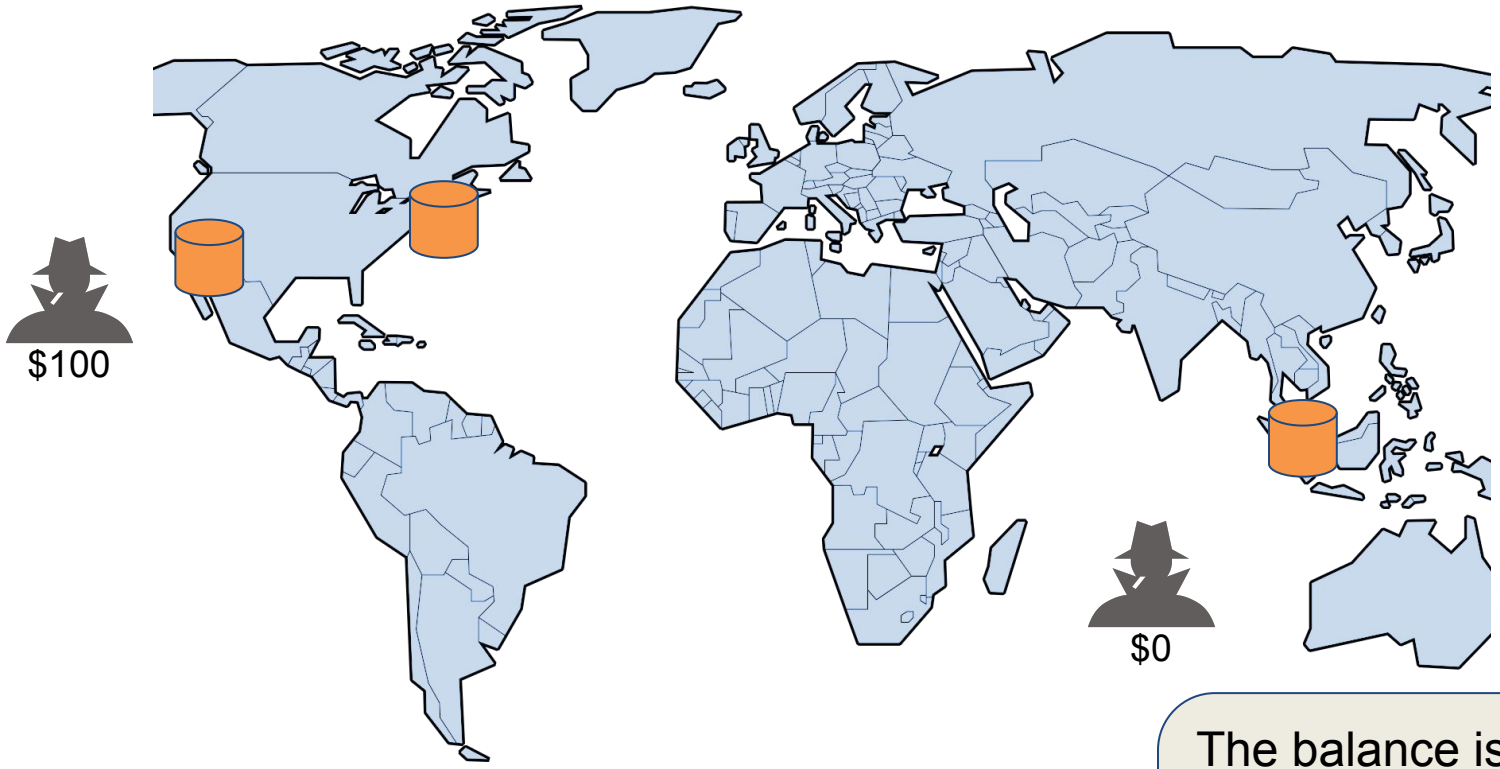
## Global Locking


| Account | Balance |
|---|---|
| xxxxx-4437 | **$100** |

Only one write request can be processed per key at a time, preventing double withdrawals!

# Good Example

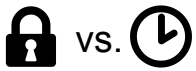
## Global Locking




| Account | Balance |
|---|---|
| xxxxx-4437 | **$0** |

The balance is set to 0 as soon as the money is withdrawn, and the second request is **denied**

# P3.3: Consistency Models

Tradeoff:  Consistency vs. Latency

- Strict
- Strong
- Sequential
- Causal
- Eventual

Please read the primers to ensure you know what each of these models mean!

# P3.3 Tasks 1 & 2: Strong Consistency

- Every request has a global timestamp order

where a timestamp is issued by a Truetime Server.

- Operations must be ordered by these timestamps

**Requirement**: At any given point of time, all clients should read the same data from any datacenter replica

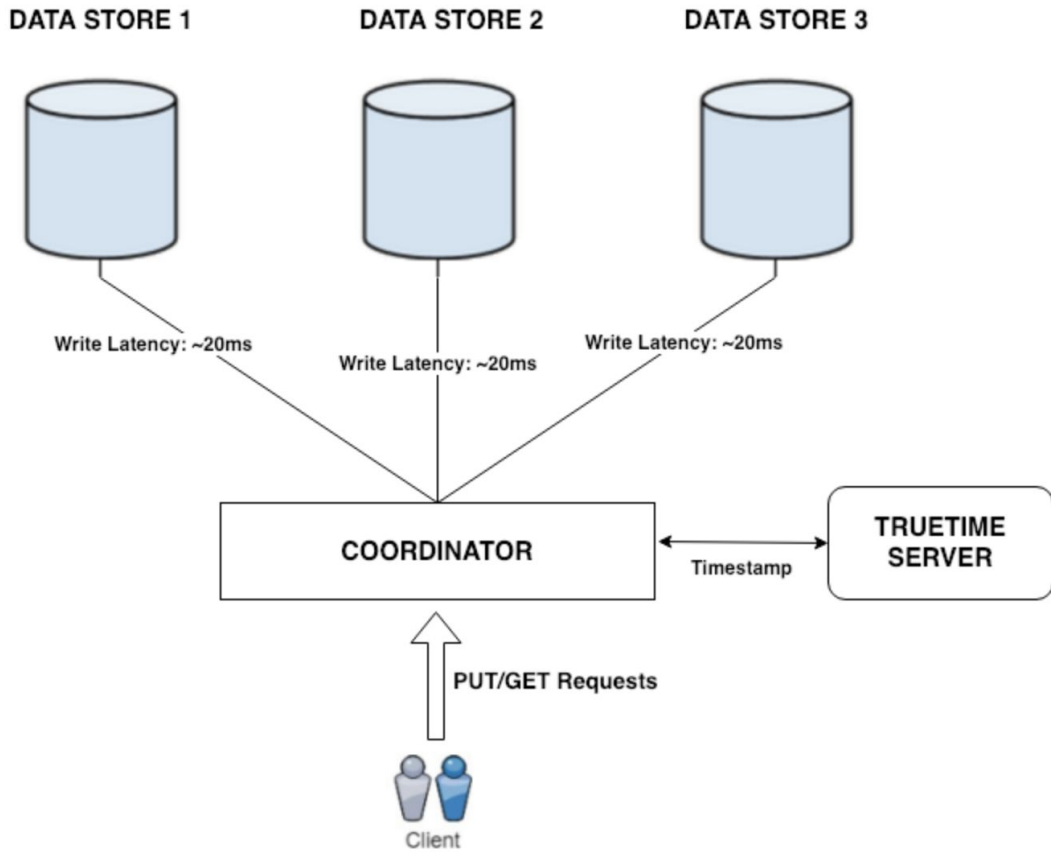
# P3.3 Task 1: Strong Consistency

Coordinator:

- A request router that routes the web requests from the clients to each datastore
- Preserves the order of both read and write requests

Datastore:

- The actual backend storage that persists collections of data



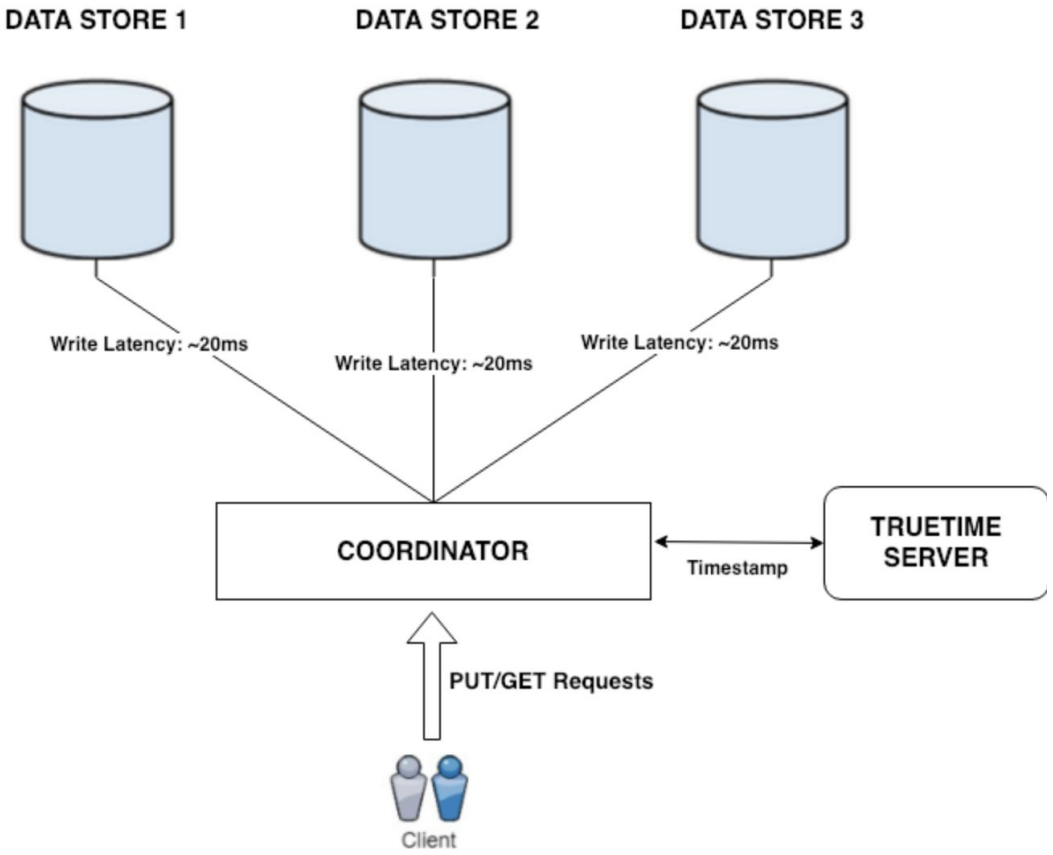
# P3.3 Task 1: Strong Consistency

Single PUT request for key ‘X’

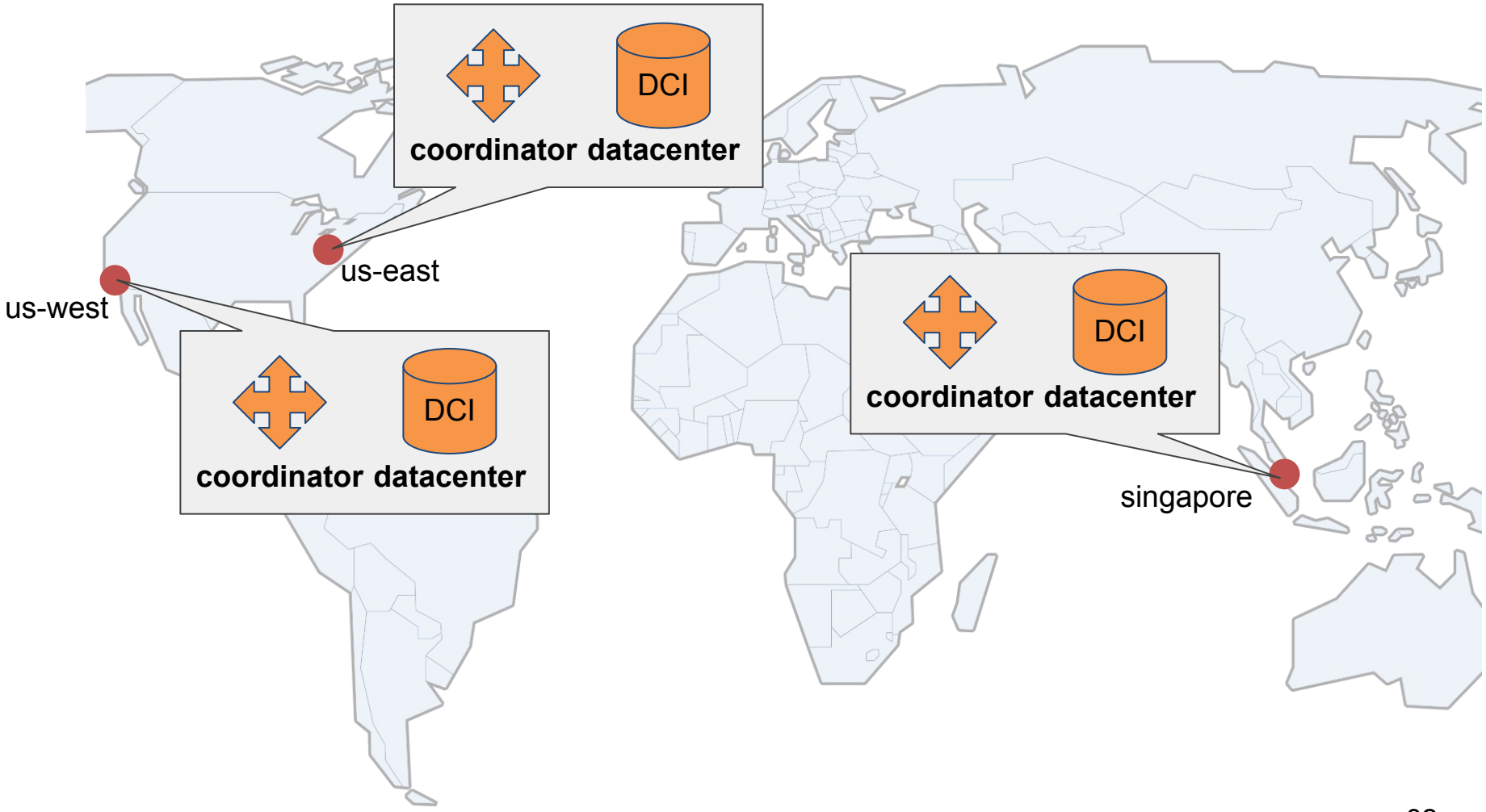
- Block all GETs for key ‘X’ until all datastores are updated
- GET requests for a different key ‘Y’ should **not** be blocked

Multiple PUT requests for ‘X’

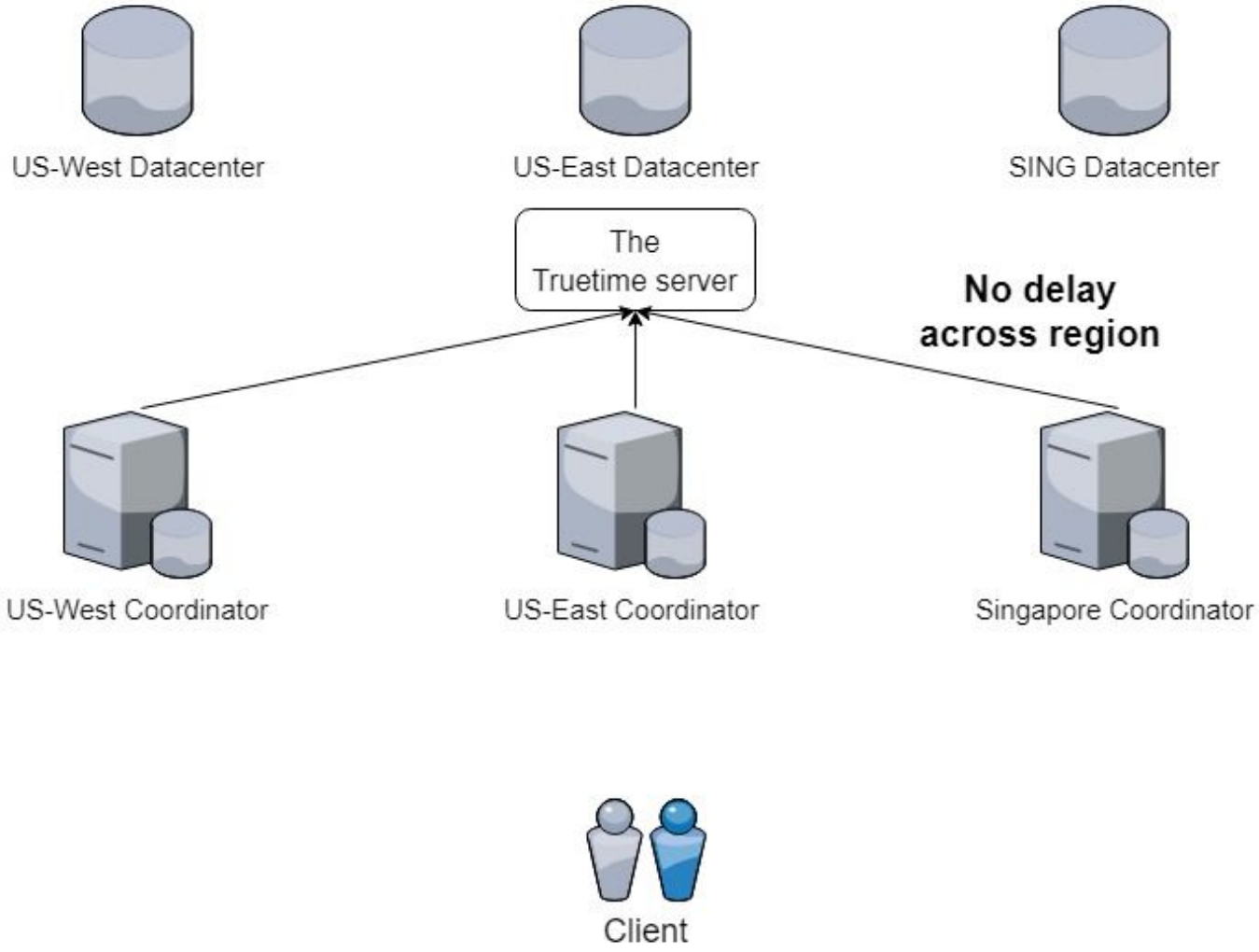
- Resolved in order of their timestamp received from the Truetime Server.
- GET requests must return the most recent value to the request timestamp



# P3.3 Task 2:
# Global Coordinators and Data Stores



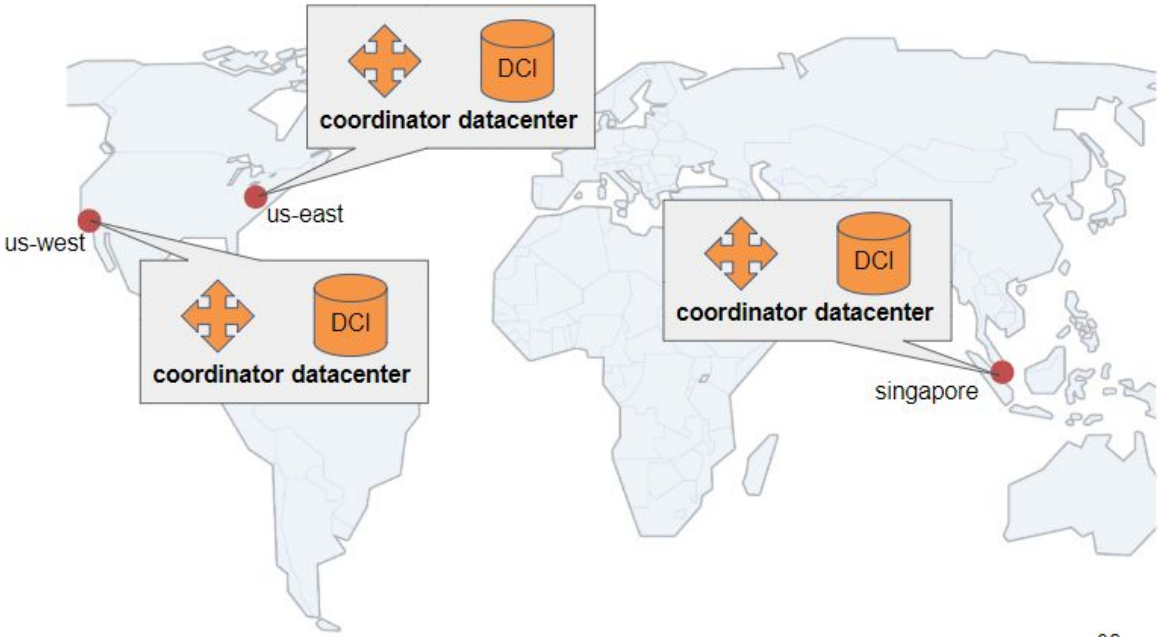
# P3.3 Task 2: Architecture



# P3.3 Task 2: Global Replication

Operates similarly to Task 1, although it requires you to have both coordinator and data centers in all 3 regions rather than just one.

Users will be spread out globally.



# Task 2 Workflow and Example

- Launch a total of 8 machines (3 data centers, 3 coordinators, 1 truetime server and 1 client) in **US East**!

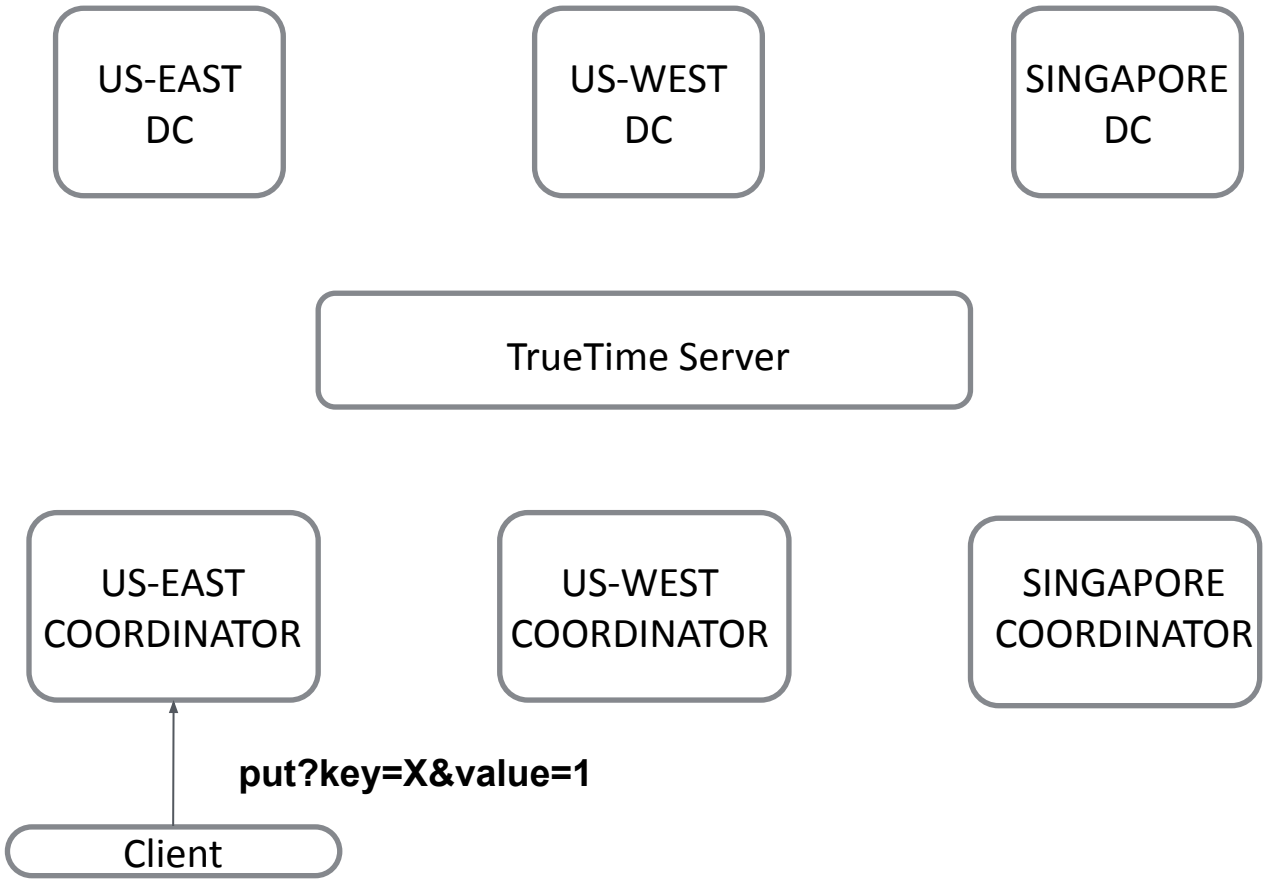
- We will simulate global latencies for you.
  - Do not actually create instances across the globe!

- Finish the code for the Coordinators and Datastores

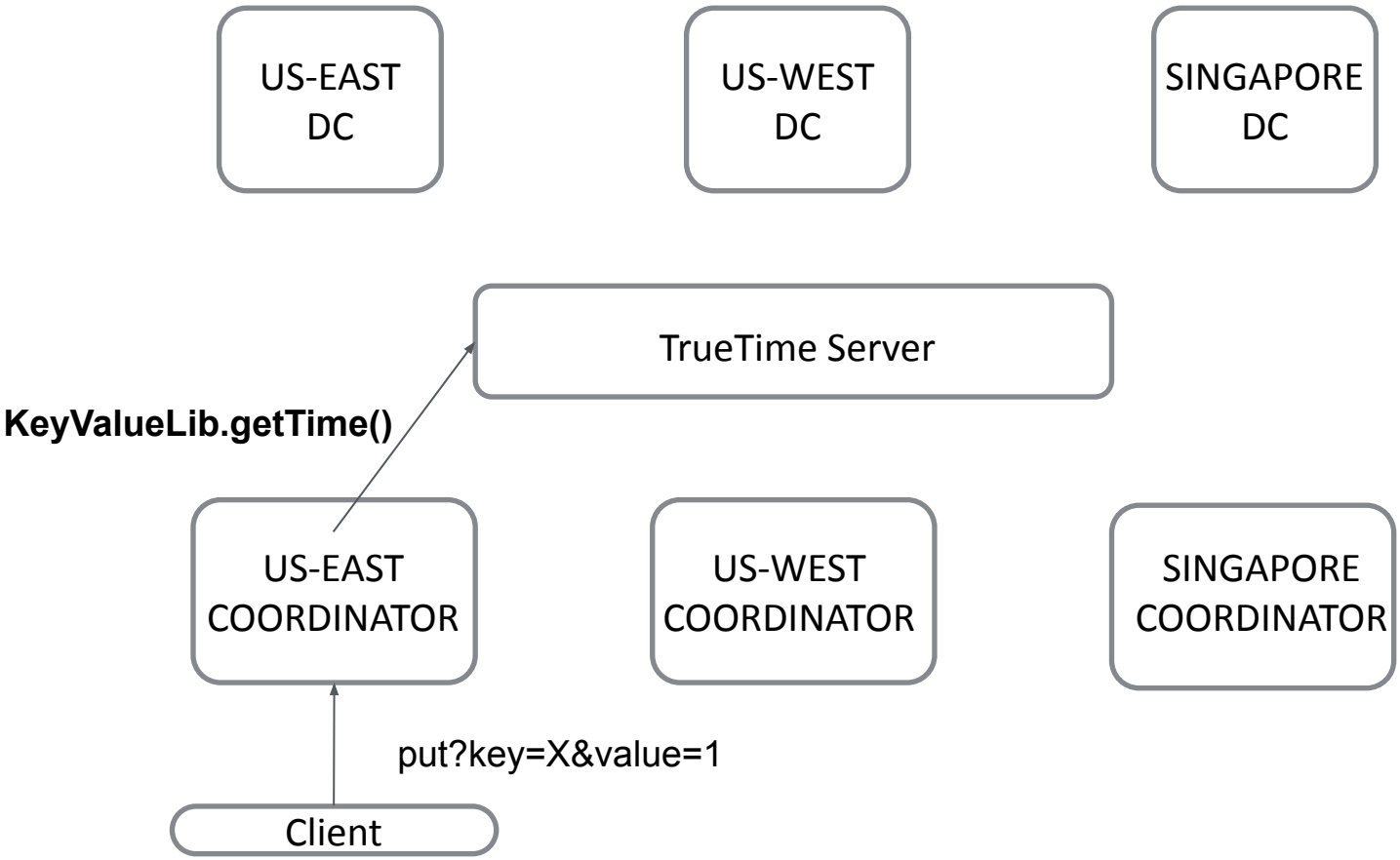
**US East (N. Virginia)**
US East (Ohio)
US West (N. California)
US West (Oregon)
Asia Pacific (Mumbai)
Asia Pacific (Seoul)
Asia Pacific (Singapore)
Asia Pacific (Sydney)
Asia Pacific (Tokyo)

P3.3 Task 2:

Complete KeyValueStore.java and Coordinator.java



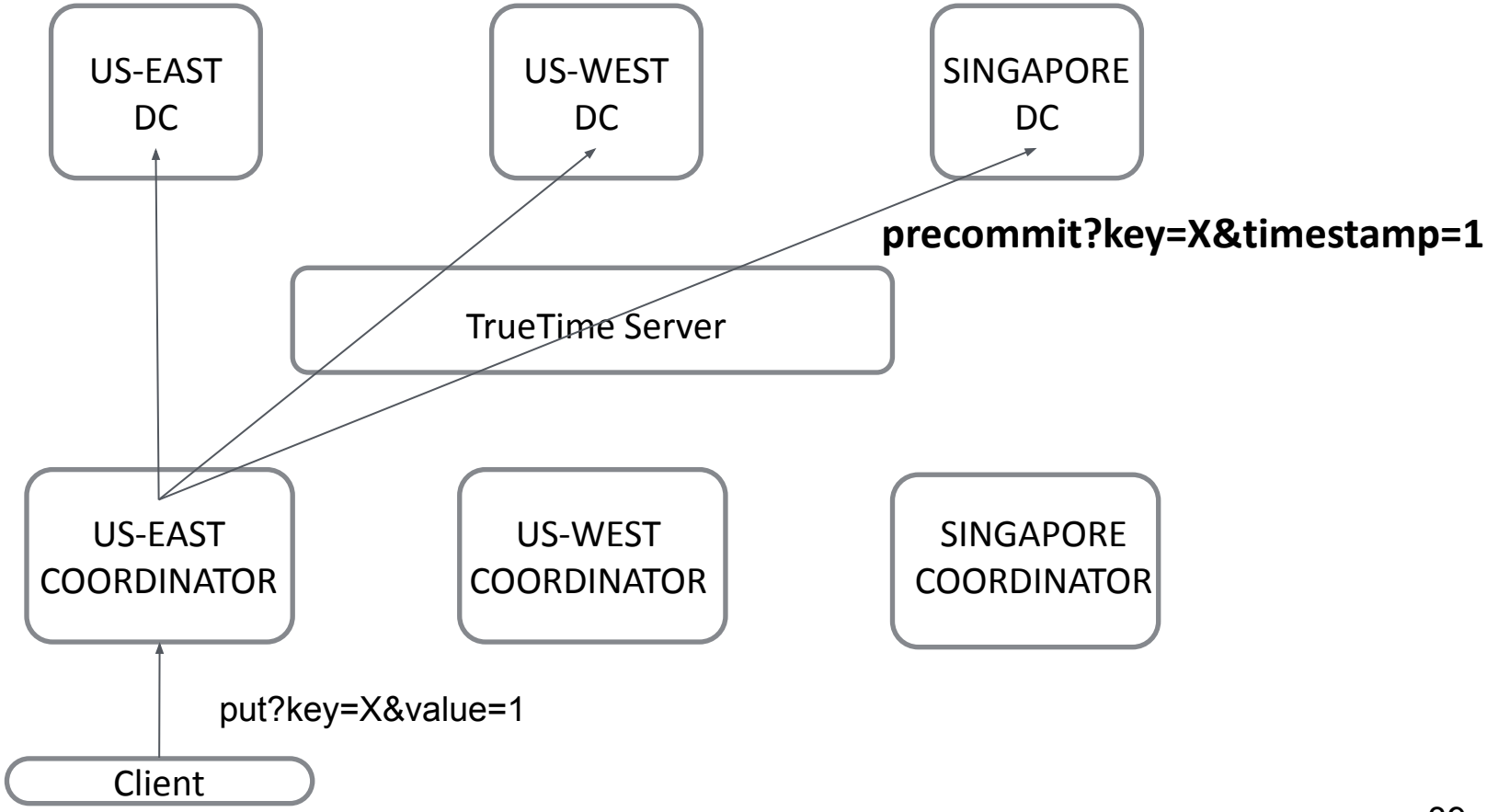
# P3.3 Task 2:
# Complete KeyValueStore.java (in DCs) and Coordinator.java (in Coordinators)



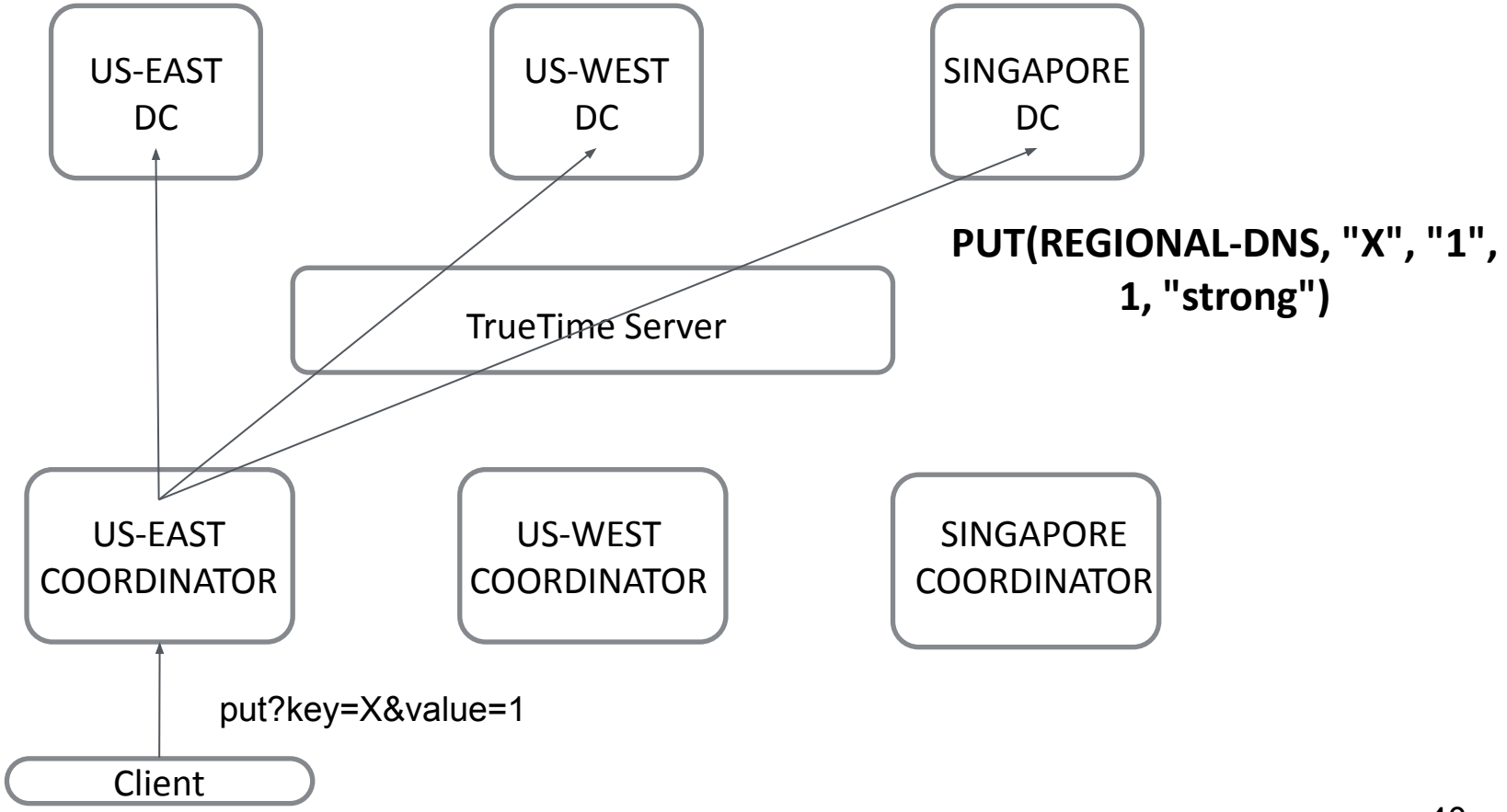
# PRECOMMIT

Contacts the Data Center of a given region and notifies it that a PUT request is being serviced for the specified key with the corresponding timestamp.

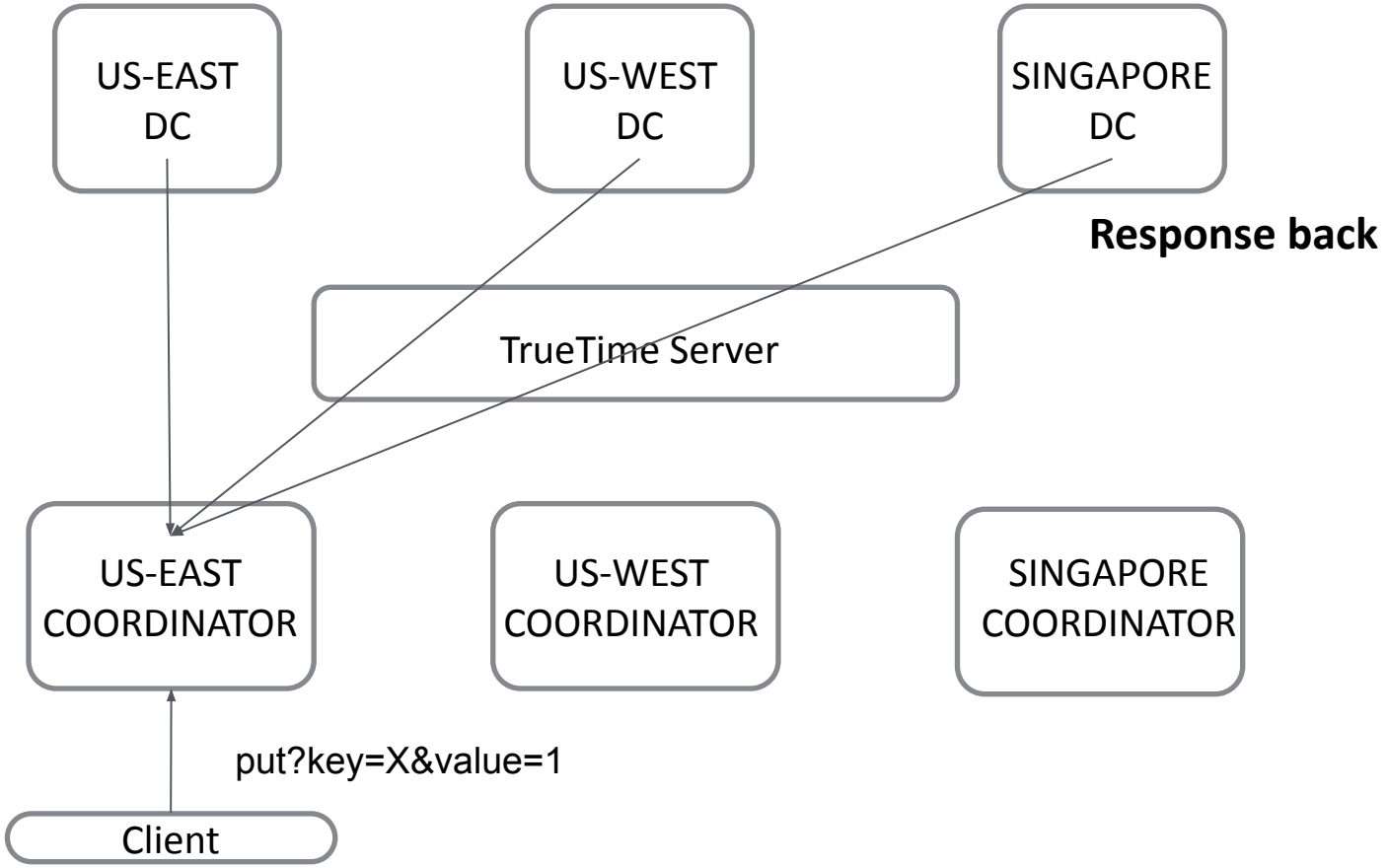
P3.3 Task 2:
Complete KeyValueStore.java (in DCs) and Coordinator.java (in Coordinators)



P3.3 Task 2:
Complete KeyValueStore.java (in DCs) and Coordinator.java (in Coordinators)



P3.3 Task 2:
Complete KeyValueStore.java (in DCs) and Coordinator.java (in Coordinators)



# Hints

- In strong consistency, “PRECOMMIT” should be useful to help you lock requests because they are able to communicate with Data Center instances.

- In strong consistency, lock by key across all the Data Center instances.

# P3.3: Eventual Consistency (Bonus)

- Write requests are performed in the order received by the local coordinator
  - Operations may not be blocked for replica consensus (no communication between servers across region)
- Clients that request data may receive multiple versions of the data, or stale data
  - Problems left for the application owner to resolve

# Suggestions

- **Read the two primers**
- Consider the differences between the 2 consistency models before writing code
- Think about possible race conditions
- Read the hints in the writeup and skeleton code **carefully**
- Only modify the following classes:
  - Coordinator.java and KeyValueStore.java

# Start early!

# Deadlines

- OLI Unit 5 Module 18: Introduction to Distributed Programming for the Cloud
- **Quiz 9** - Friday, Apr. 9th
- **Project 3.3** - Sunday, Apr. 11th
- **Team Project, Twitter Analytics**
  - Phase 2 - Live test: Next Sunday, Apr. 18th