# 15-319 / 15-619
# Cloud Computing

Recitation 9

March 20, 2018

# Overview

- **Last week's reflection**
  - Last week was Spring Break!
- **This week's schedule**
  - OLI - Modules 15, 16 & 17
    - Quiz 8 – due on Friday, March 23rd
  - Project 3.3 – due on Sunday, March 25th
- **Team Project, Phase 1**
  - Query 1 is due on Sunday, March 25
  - Query 2 is due next week!

# Last week: Week 8

- **Module 14: Cloud Storage**
  - Quiz 7
- **Project 3.2**
  - Social Network with Heterogenous Backends

# This Week: Conceptual Content

OLI UNIT 4: Cloud Storage

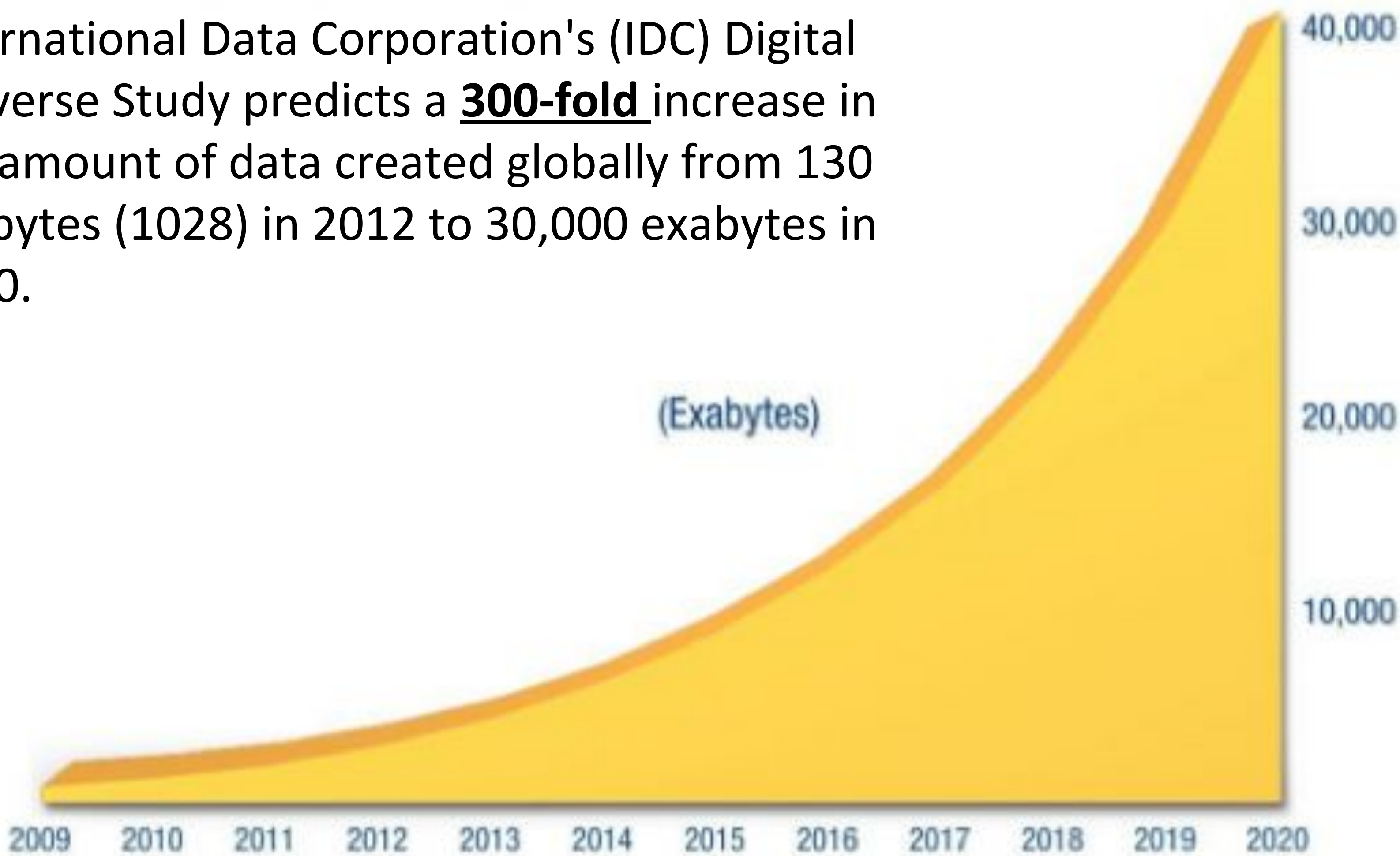
- **Module 15: Case Studies: Distributed File System**
  - HDFS
  - Ceph
- **Module 16: Case Studies: NoSQL Databases**
- **Module 17: Case Studies: Cloud Object Storage**
- **Quiz 8**
  - **DUE on Friday, March 23rd**

# Project 3 Weekly Modules

- P3.1: Files, SQL and NoSQL
- P3.2: Social network with heterogeneous backend storage
- P3.3: Replication and Consistency Models
  - Primer: Intro. to Java Multithreading
  - Primer: Thread-safe Programming
  - Primer: Intro. to Consistency Models

# Scale of Data is Growing

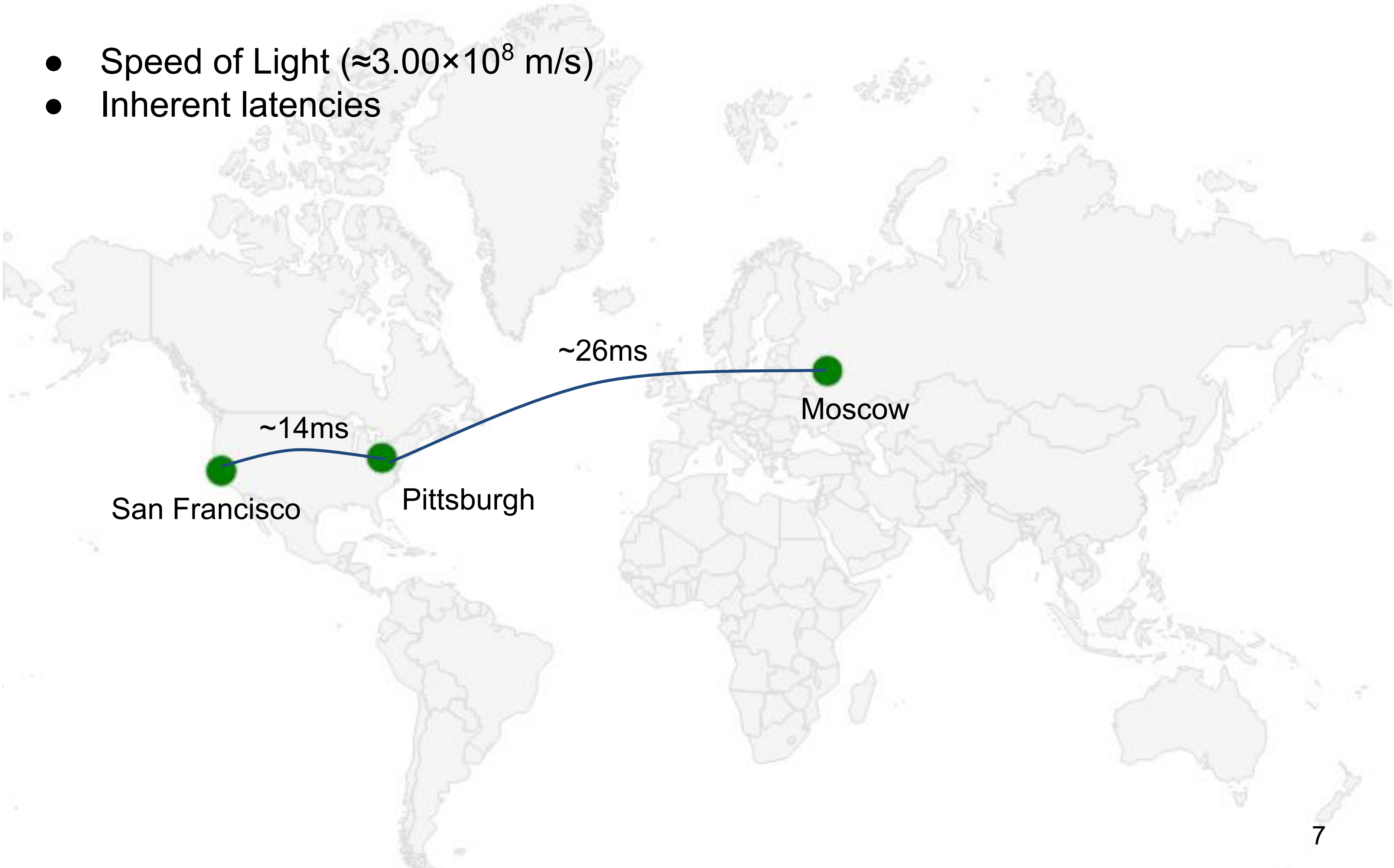
International Data Corporation's (IDC) Digital Universe Study predicts a **<u>300-fold</u>** increase in the amount of data created globally from 130 exabytes (1028) in 2012 to 30,000 exabytes in 2020.




Source: IDC's Digital Universe Study, sponsored by EMC, December 2012

# Users are Global

- Speed of Light ($\approx 3.00 \times 10^8$ m/s)
- Inherent latencies



# Typical End-To-End Latency

- Typical end-to-end latency
  - The client sends the request to the server
    - Network latency
  - The backend processes the request and sends the response
    - Overhead of fetching and processing data from backend
    - Network latency
  - The client receives the response

# Latency with a Single Backend


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

# Replicate the Data Globally


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

# Replicate the Data Close to Users

# Demo

Run:

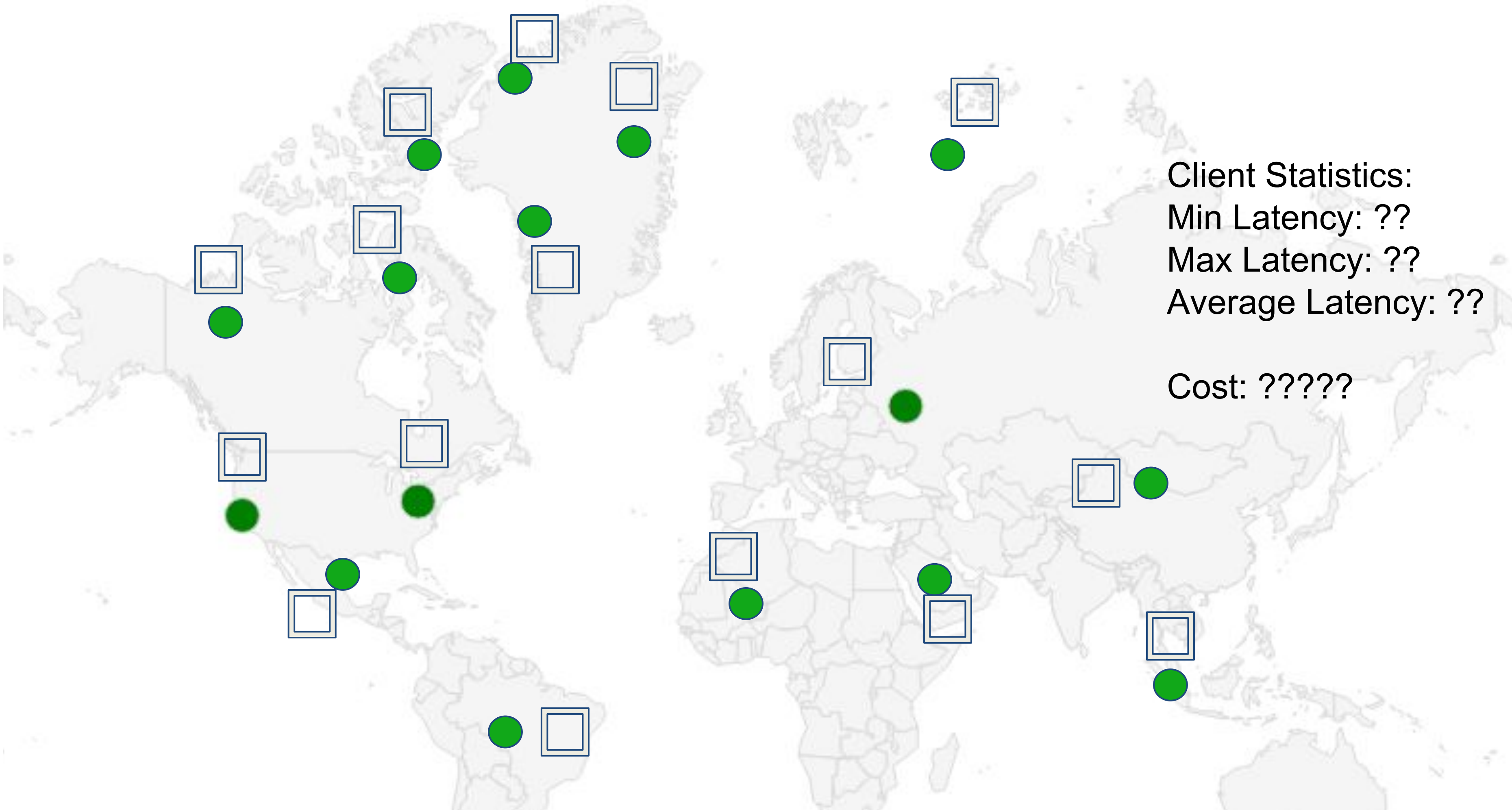
- ping www.cmu.edu
- ping www.google.com
- ping www.berkeley.edu
- ping www.nus.edu.sg

Compare the latencies of these global webpages!

# Replication

- As you can see, by adding replicas to strategic locations in the world, we can significantly reduce the latency seen by our global clients
- Each added datacenter decreases the average latency
- But how about the cost?

# What If We Continue to Replicate?

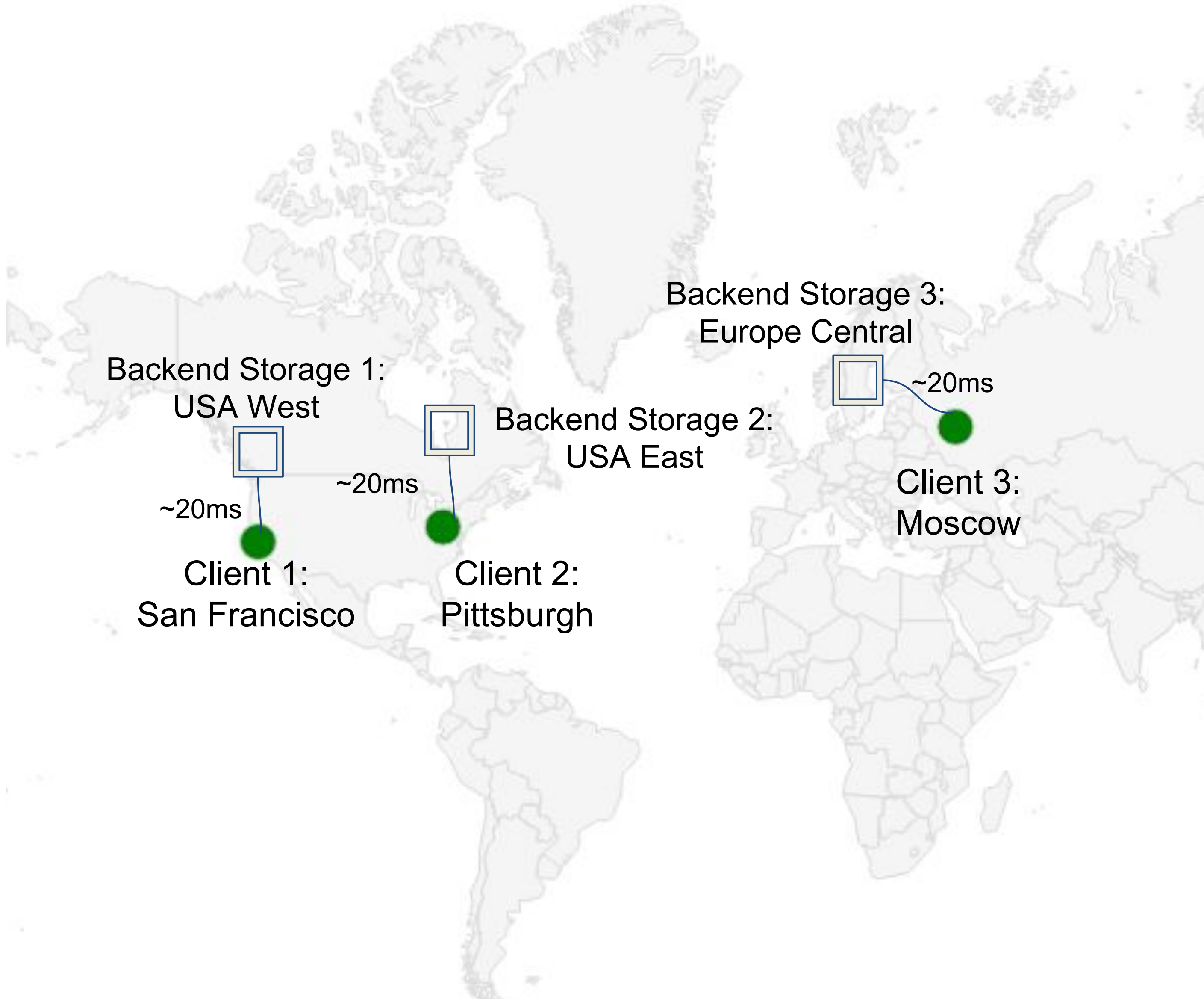


Client Statistics:
Min Latency: ??
Max Latency: ??
Average Latency: ??

Cost: ?????

We have to consider cost as well as data consistency across replicas, which increases the latency for writes.

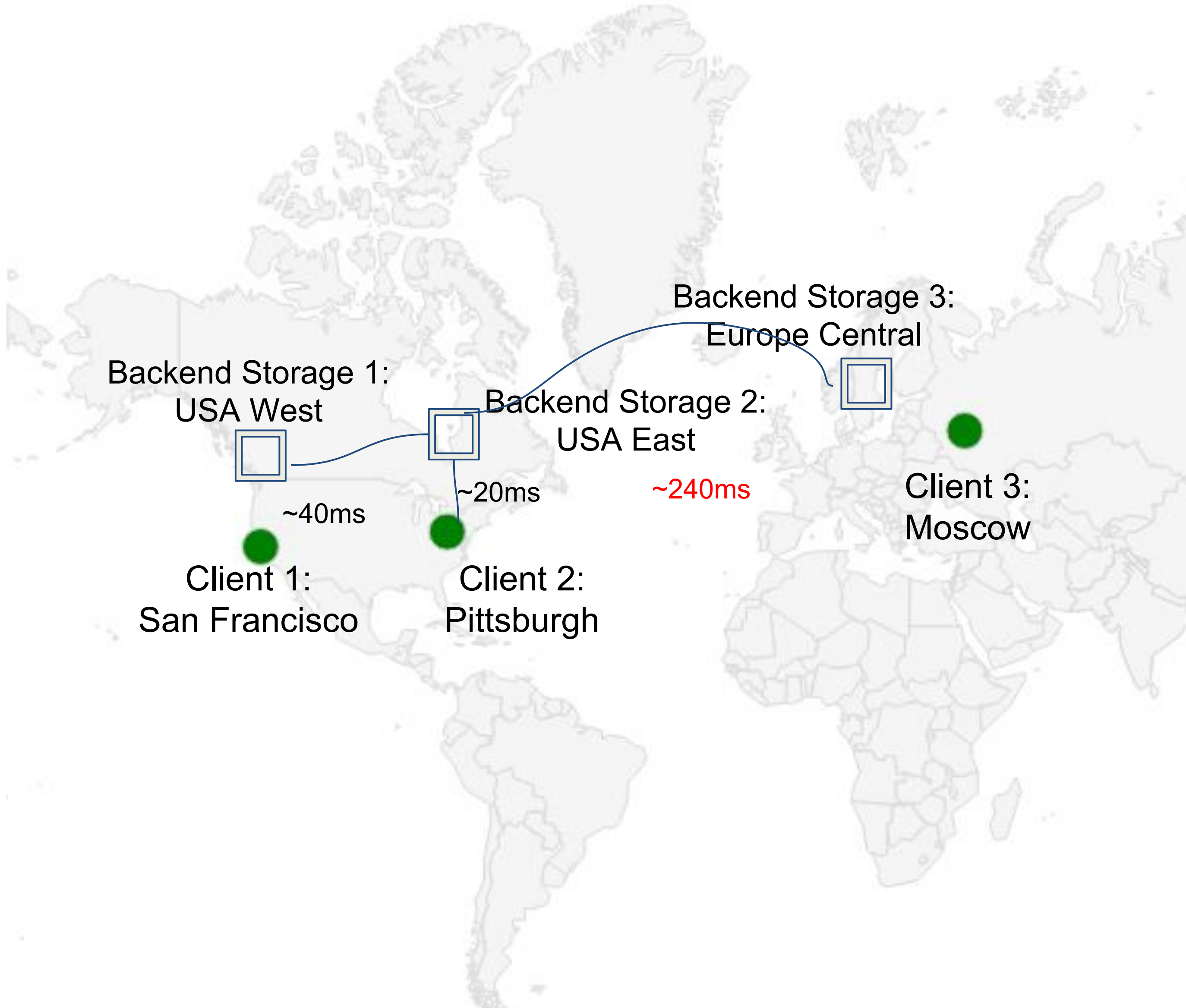
# Replication READ




Read Operation:

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

# Replication WRITE




Write Operation:

Latency for Client 2 = 20ms + MAX(40ms, 240ms) = 260ms

All the clients suffer from long latency

# Replication Reads and Writes

- Read operations are very fast!
  - All clients have a replica close to them to access
- Write requests are quite slow
  - Write requests must update all the replicas
  - If multiple write requests for a certain key, then they may have to wait for each other to complete

# Pros and Cons of Replication

- Duplicate the data across multiple instances
- Advantages
  - Low latency for reads
  - Reduce the workload of a single backend server (Load balance for hot keys)
  - Handle failures of nodes (High availability)
- Disadvantages
  - Requires more storage capacity and cost
  - Updates are slower
  - Changes must reflect on all datastores either instantly or eventually (Data Consistency)

# Data Consistency Becomes Necessary

- Data consistency across replicas is important
    - Five consistency levels:
      Strict, Strong (Linearizability), Sequential, Causal and Eventual Consistency
- <u>**This week's task**</u>: Implement Strong Consistency
    - All datastores must return the same value for a key at all times
    - The order in which the values are updated must be preserved
- <u>**Bonus:**</u> Implement Eventual Consistency

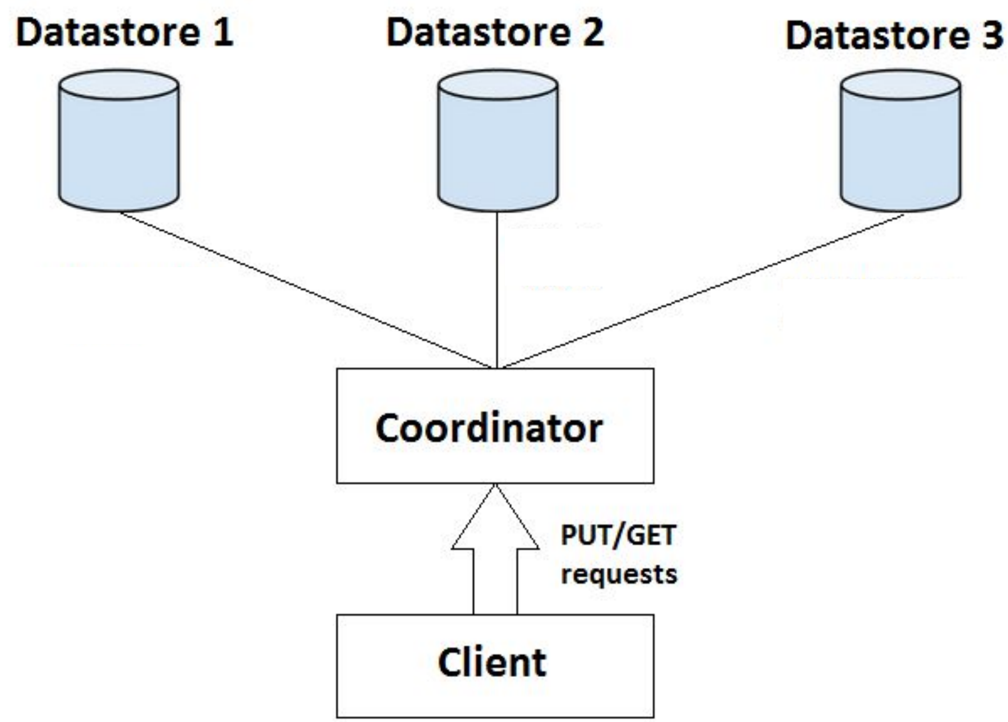
# P3.3 Task 1: Strong Consistency

Coordinator:

- A request router that routes the web requests from the clients to datacenter
- Preserves the order of both READ&WRITE requests

Datastore:

- The actual backend storage that persists collections of data



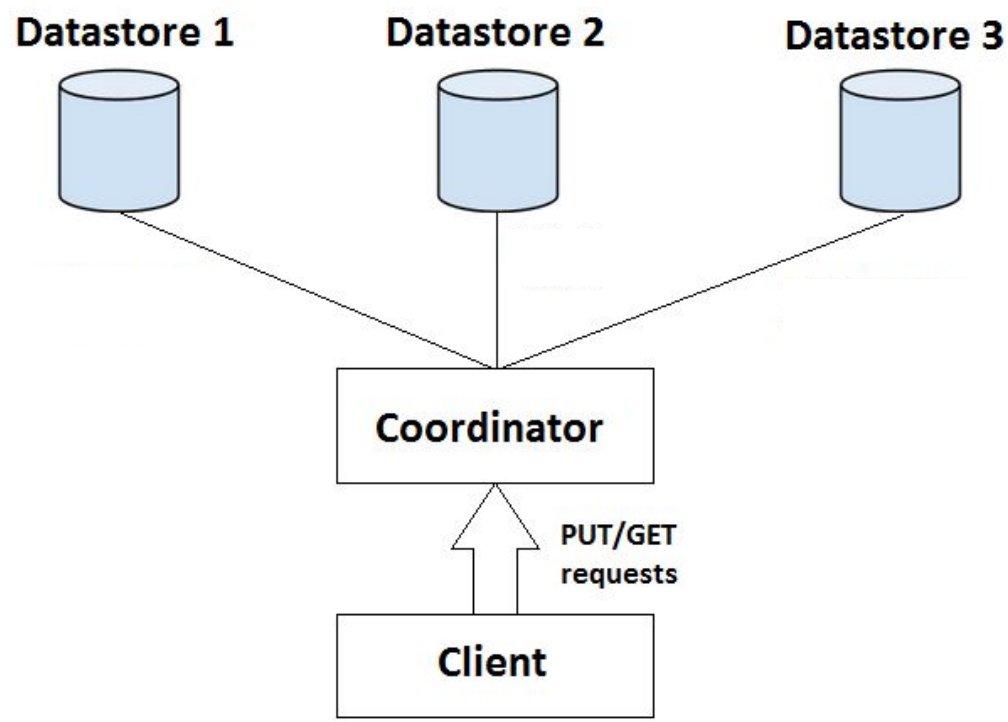
# P3.3 Task 1: Strong Consistency

Single PUT request for key 'X'

- Block all GET 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 when received by Coordinator.
- Any GET request in between 2 PUTs must return the first PUT value

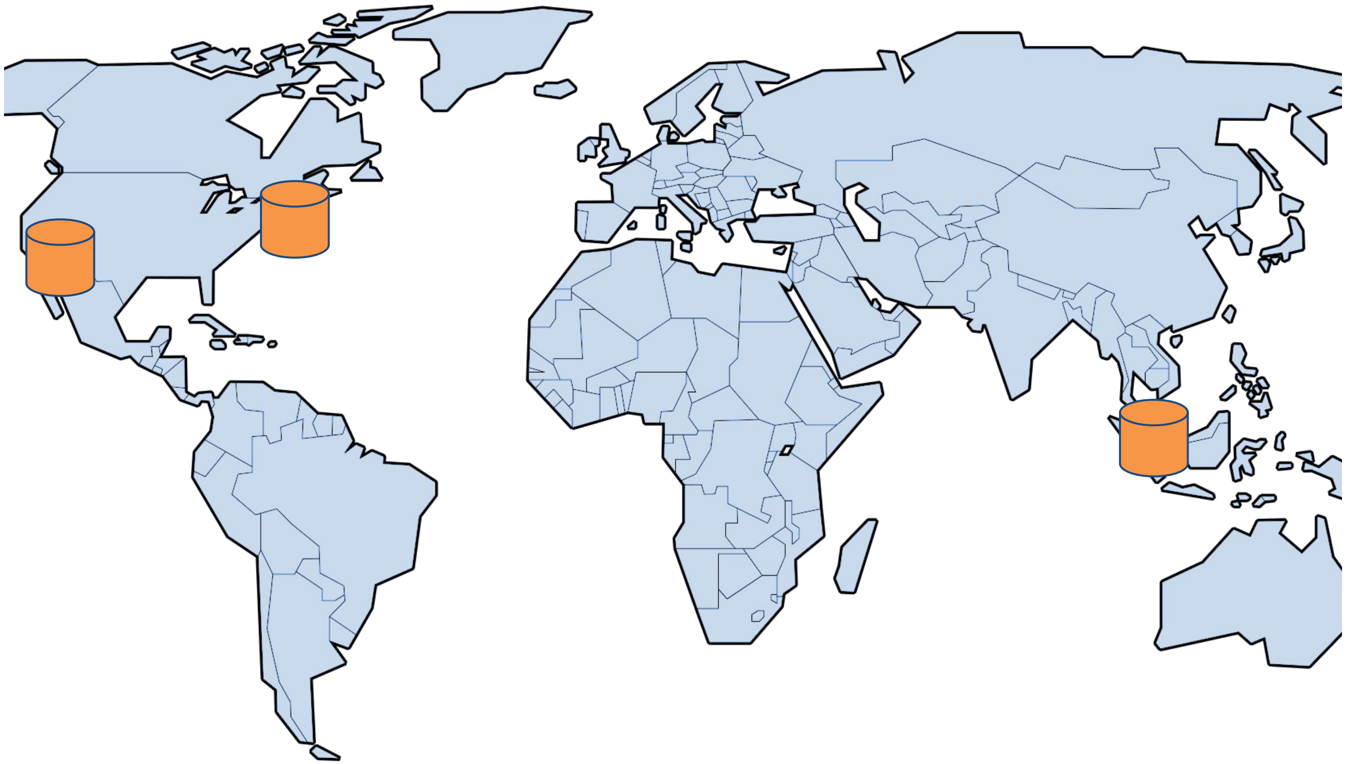


# P3.3 Tasks 1 & 2: Strong Consistency

- Every request has a global timestamp order
  - In task 1, the timestamp is issued by the coordinator
  - In task 2, the timestamp is issued by the client
- Operations must be ordered by the timestamps

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

# Choosing a Consistency Level
# Bad Example

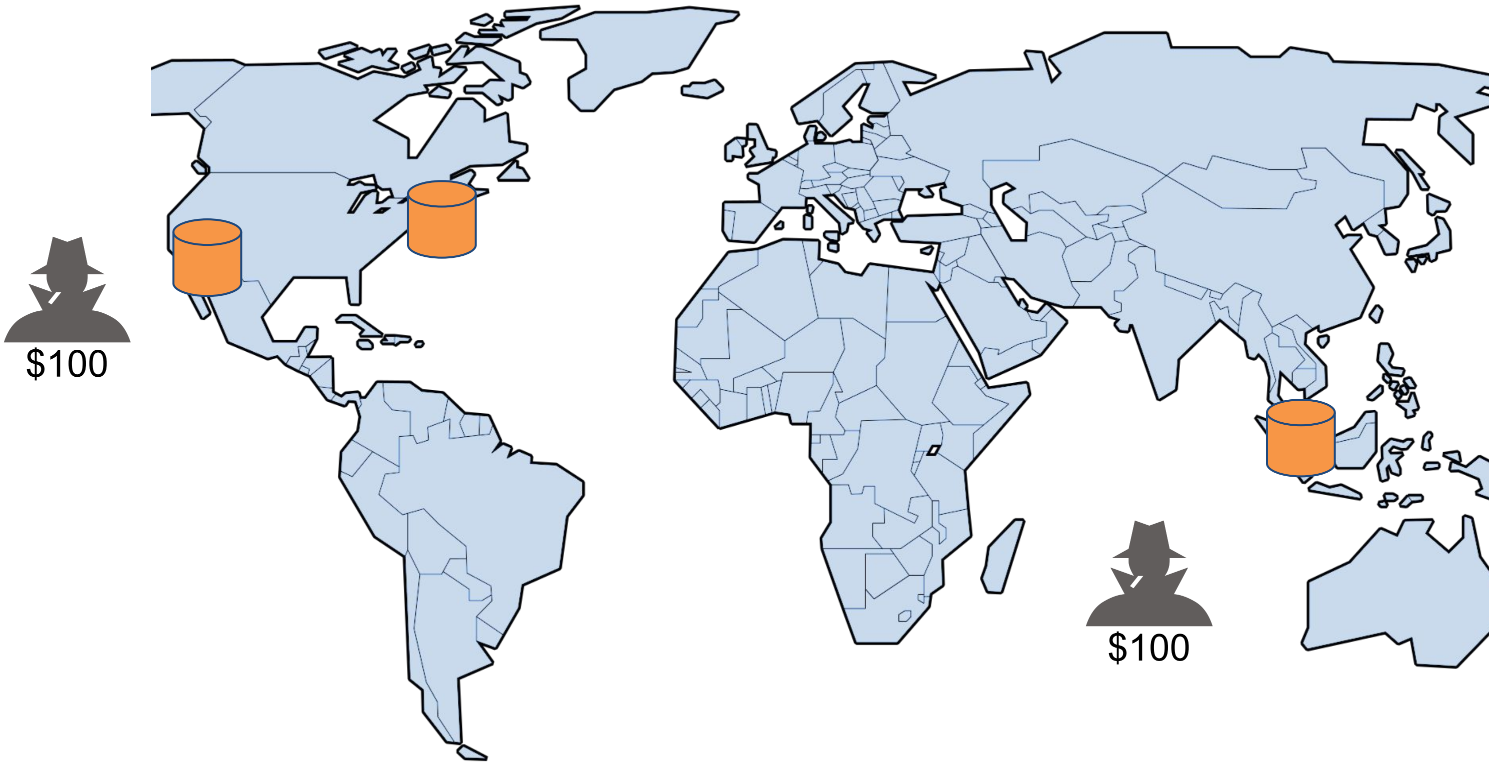


| Account | Balance |
|---|---|
| xxxxx-4437 | $100 |

# Choosing a Consistency Level
# Bad Example


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

# Choosing a Consistency Level
# Bad Example




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

Bank lost $100

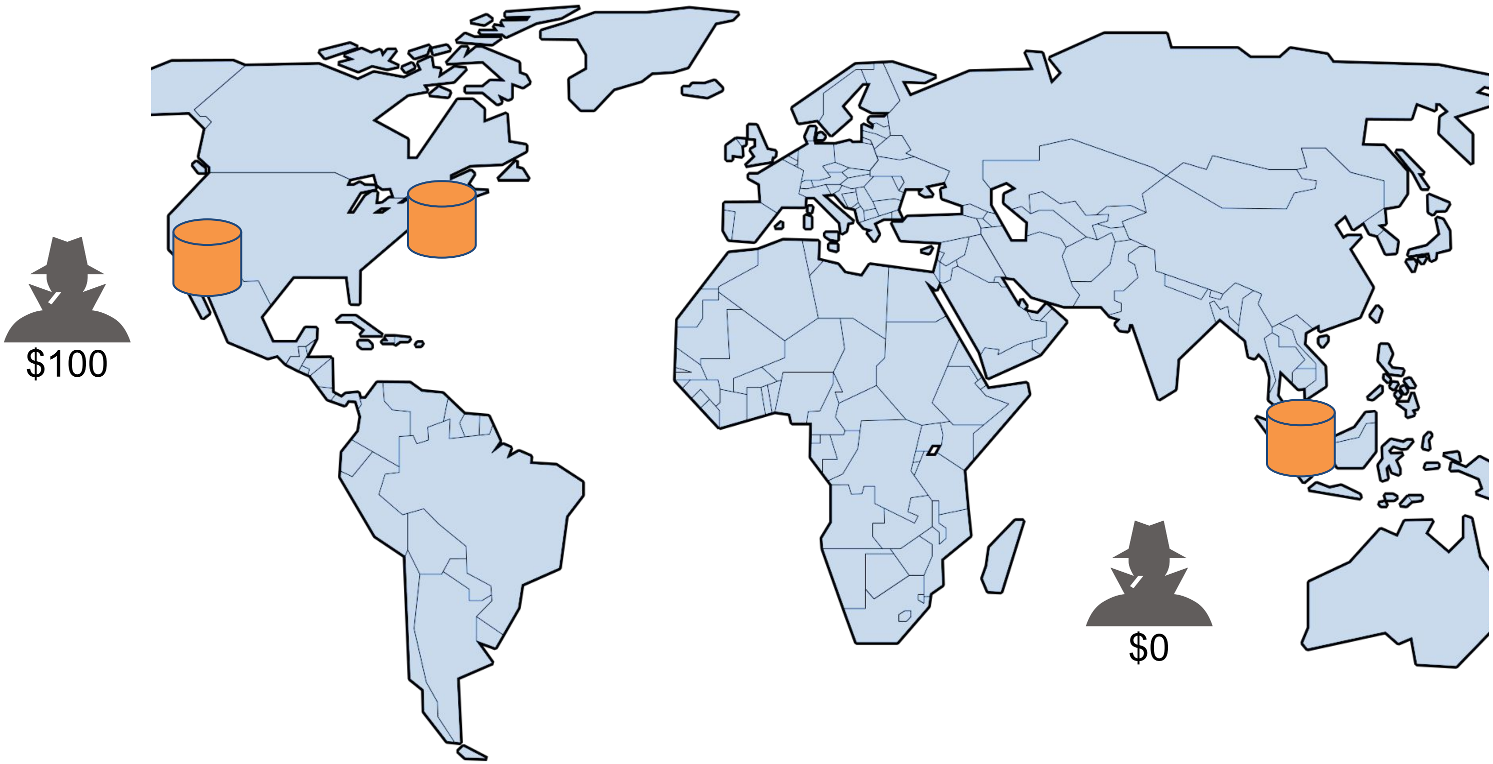
# Choosing a Consistency Level
# Good Example


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

# Choosing a Consistency Level
# Good Example


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

# Choosing a Consistency Level
# Good Example



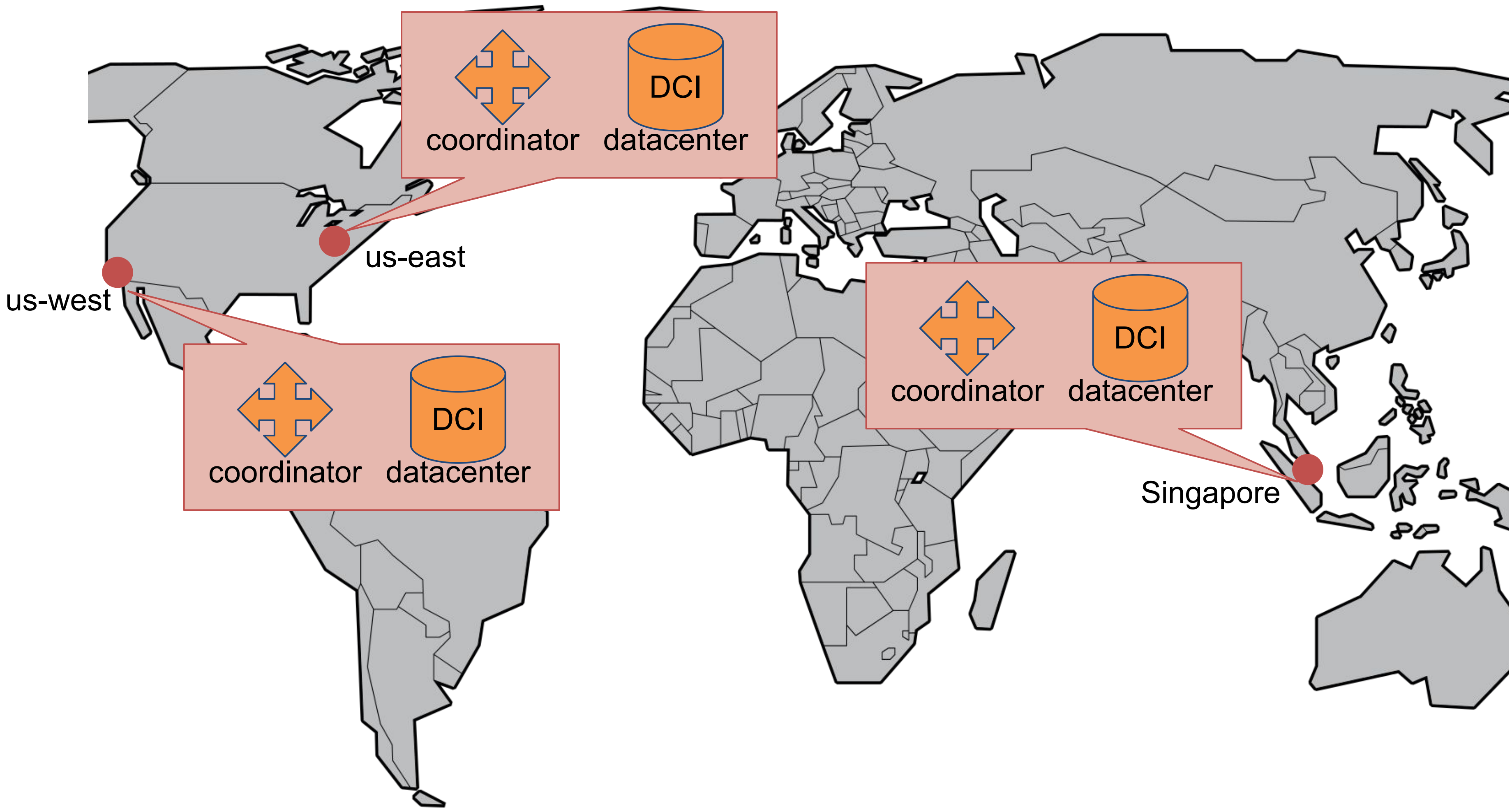

| Account | Balance |
|---|---|
| xxxxx-4437 | **$0** |

# P3.3: Consistency Models

Tradeoff: vs. Consistency vs. Availability

- Strict
- Strong
- Sequential
- Causal
- Eventual

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



# P3.3: Tasks

- Launch the Coordinators and DCs in **us-east-1**
  - We’ll simulate global latencies for you
- Implement the code for the Coordinators and Datastores

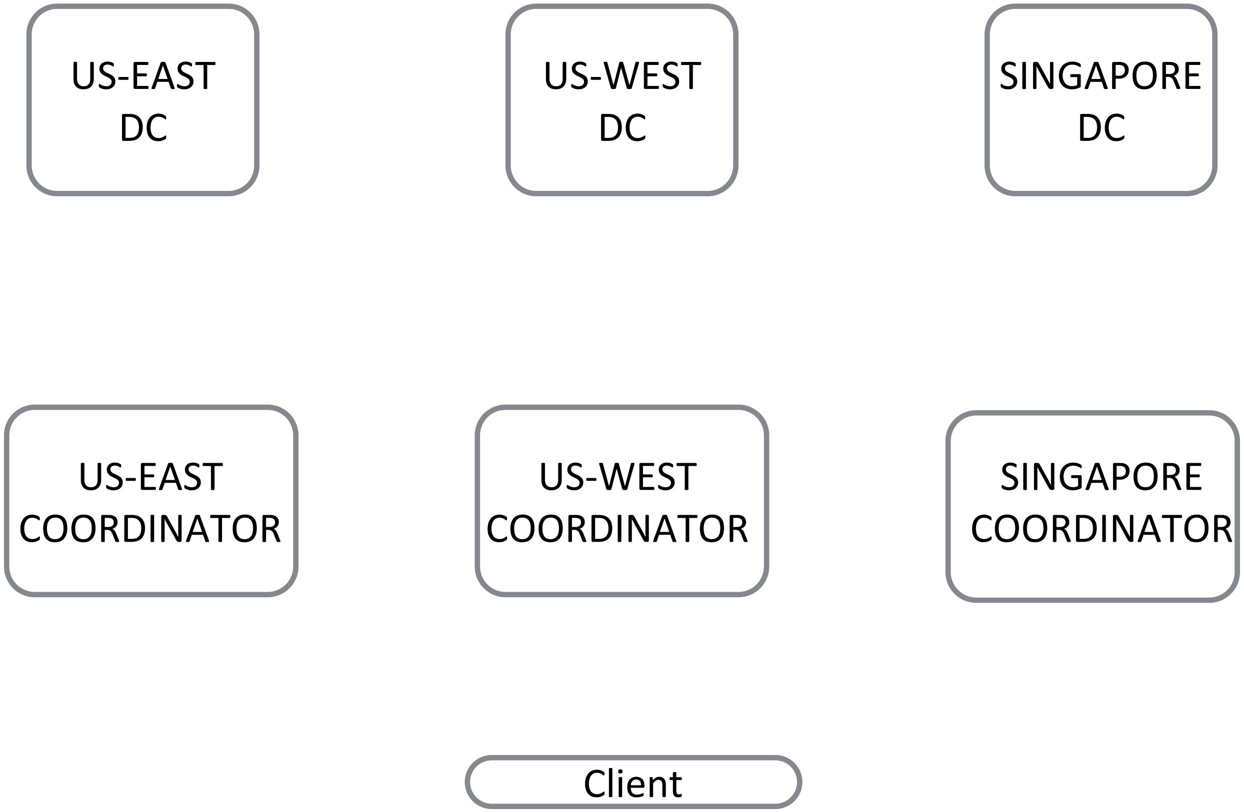
# Task 2 Workflow and Example

- Launch a total of 7 machines (3 data centers, 3 coordinators and 1 client)

- All machines should be launched in US East region.

The “US East” here has nothing to do with the simulated location of datacenters and coordinators in the project.

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



# Hash Functions and Primary Coordinator

- Primary coordinator is responsible for handling all PUT requests for that particular key. I.e., a PUT operation on a specific key will be handled only by its Primary Coordinator.
- A hash function will determine the primary coordinator of the key.

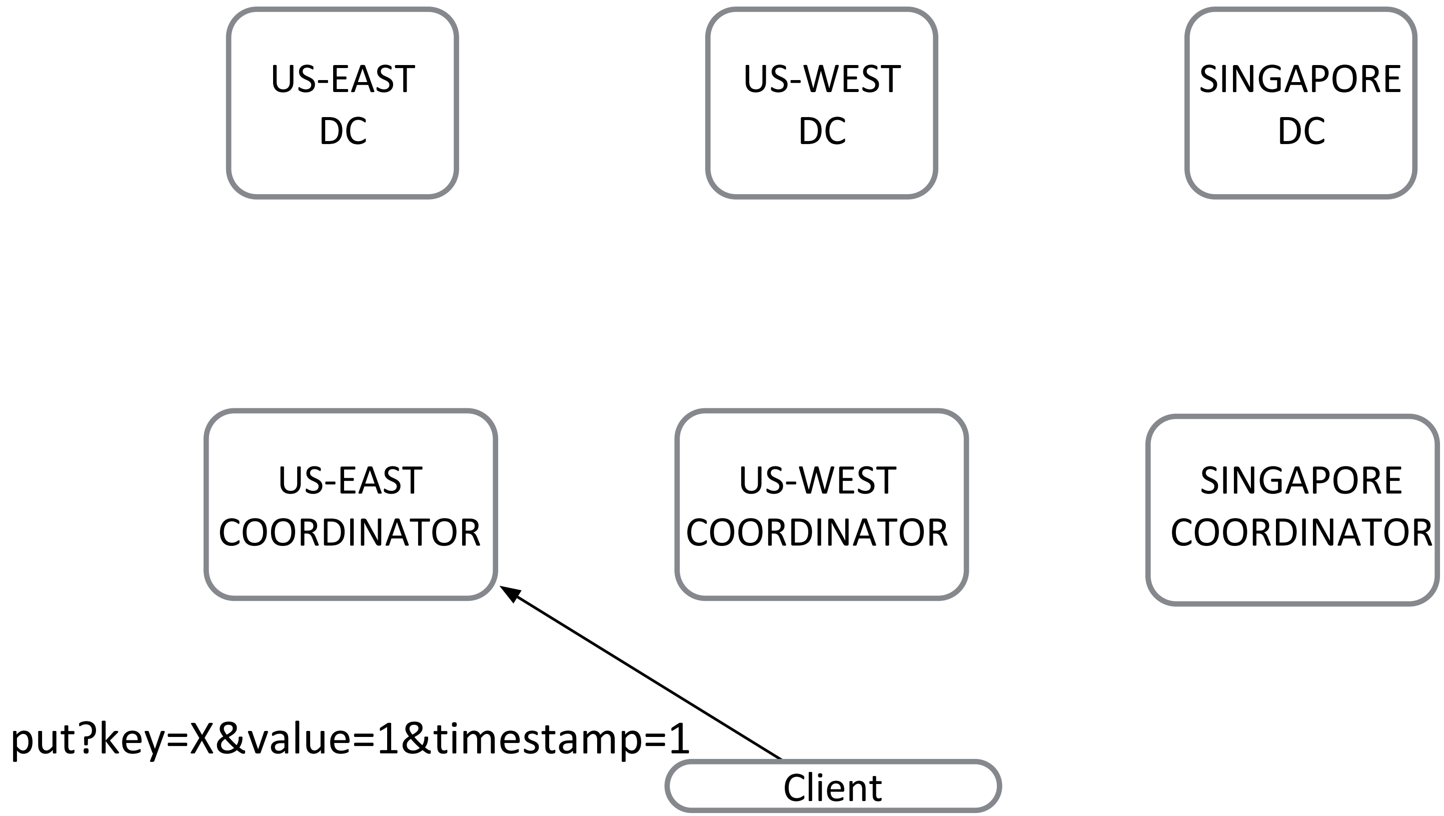
Here's what coordinator will do upon receiving a PUT request.

1. Calculate the hash value of that key.
2. If the Coordinator finds that the hash function maps the key to itself (i.e., the receiving coordinator is the primary coordinator for that key), then it should handle the request.
3. Otherwise, the Coordinator should forward (by calling KeyValueLib.FORWARD) the request to the Primary Coordinator of that key and send a PRECOMMIT message to datacenters.

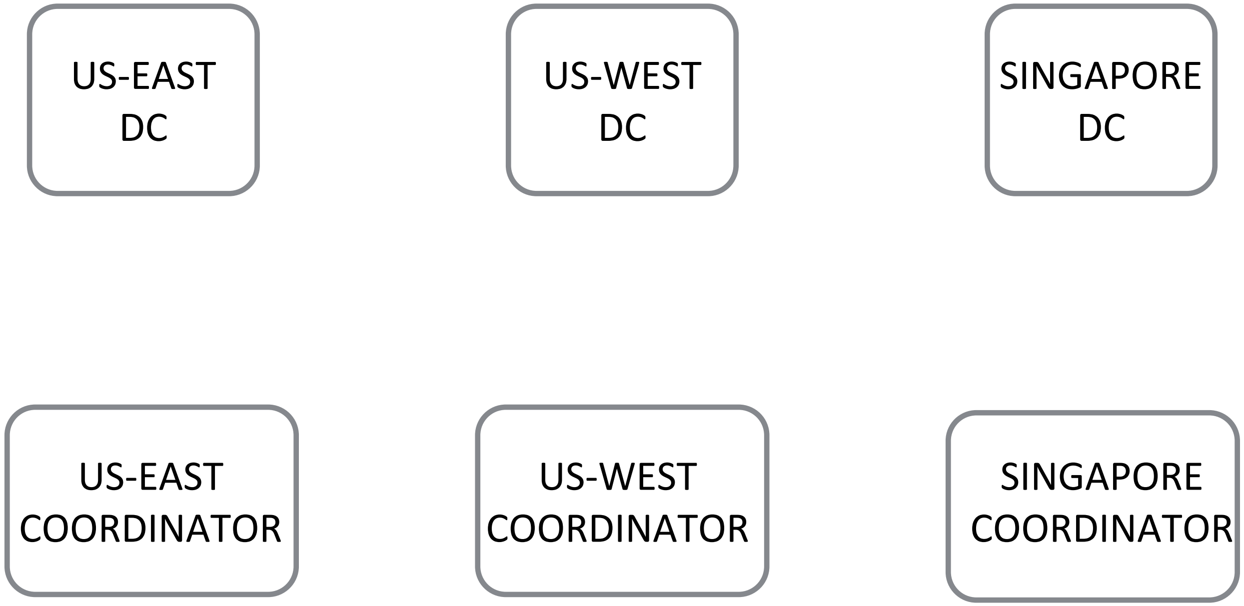
# PRECOMMIT

- This API method will contact the datastores of a given region, and notify it that a PUT request is being serviced for the specified key, starting at the specified timestamp.

# Example workflow for a PUT request using strong consistency



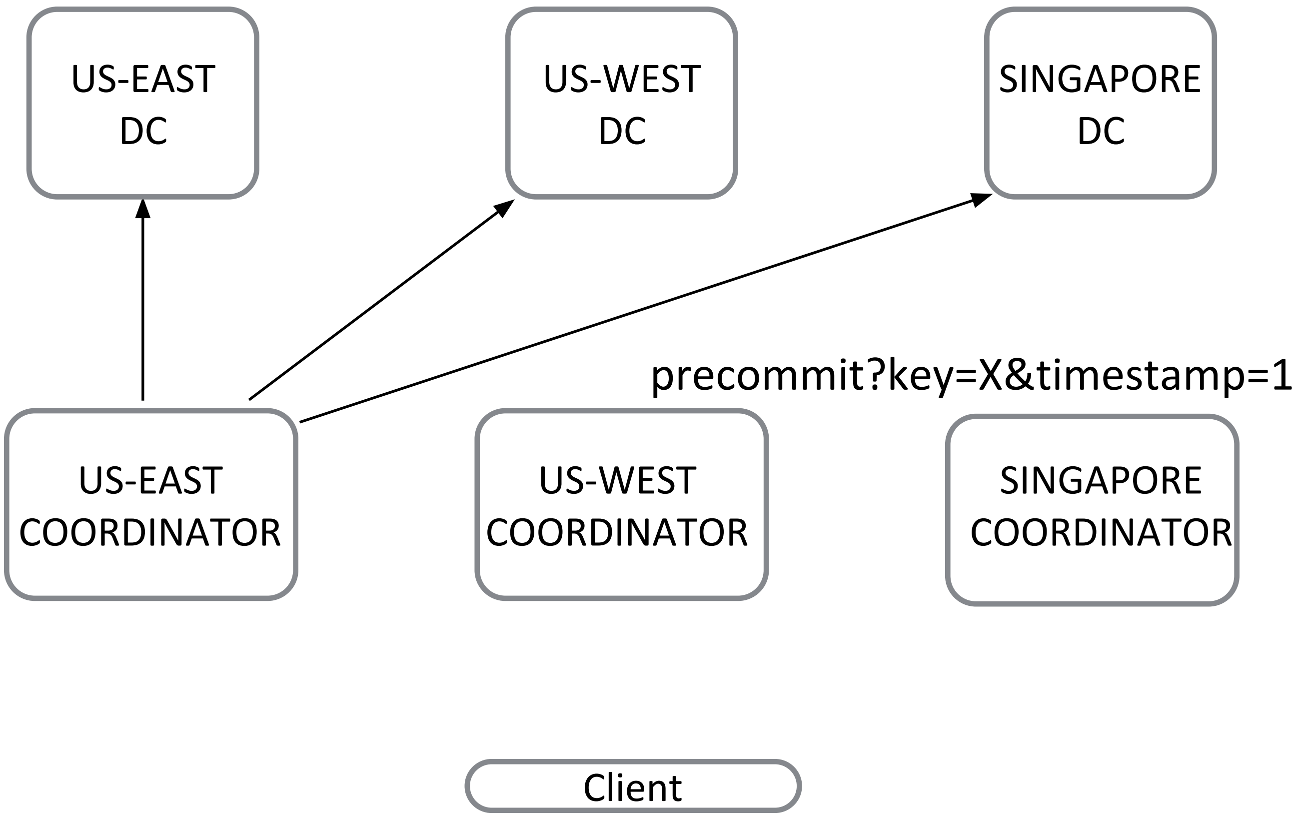
# Example workflow for a PUT request using strong consistency




hash(“X”) to determine if this coordinator is responsible for “X”.

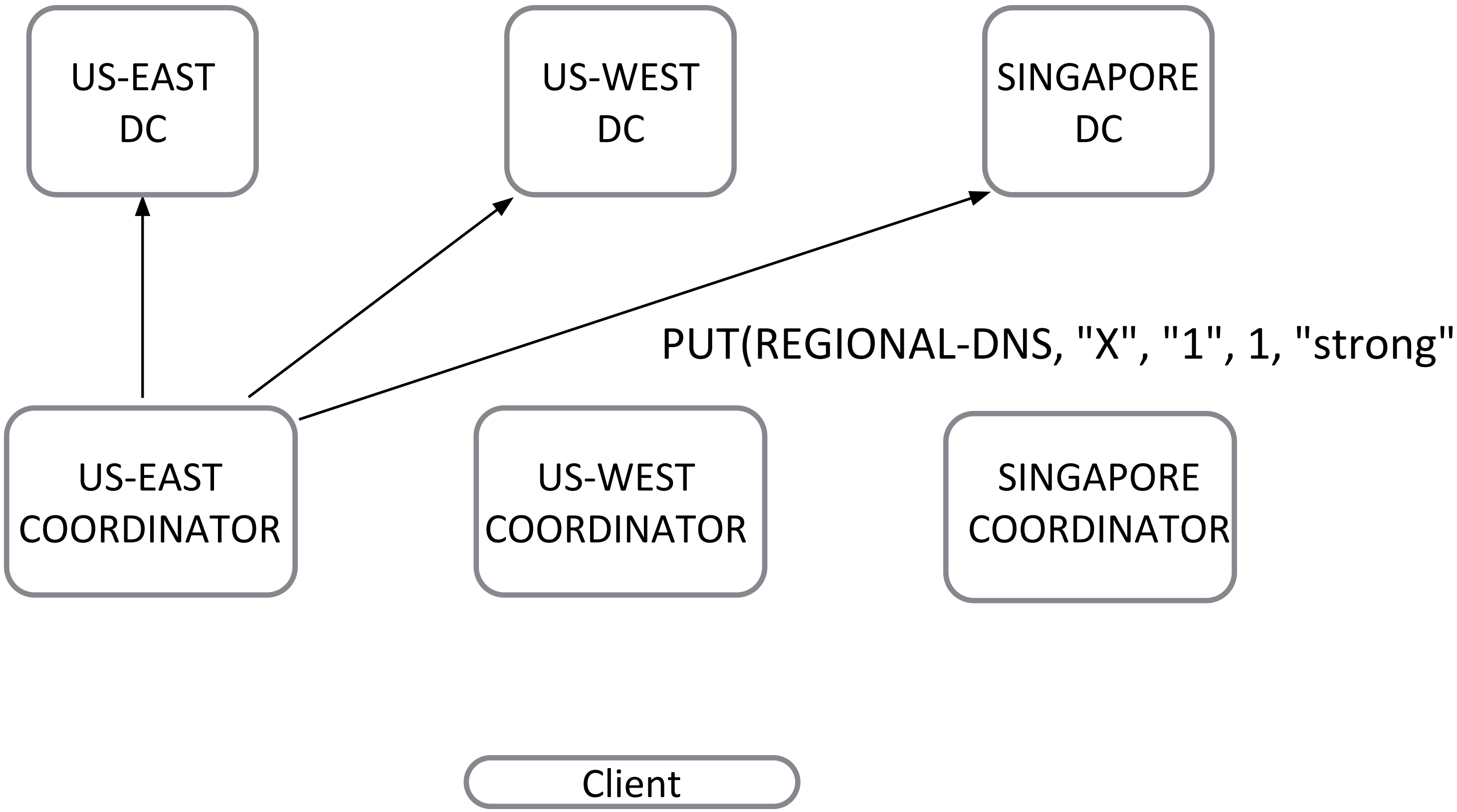
Example workflow for a PUT request using strong consistency

- If US-EAST is responsible for key "X"



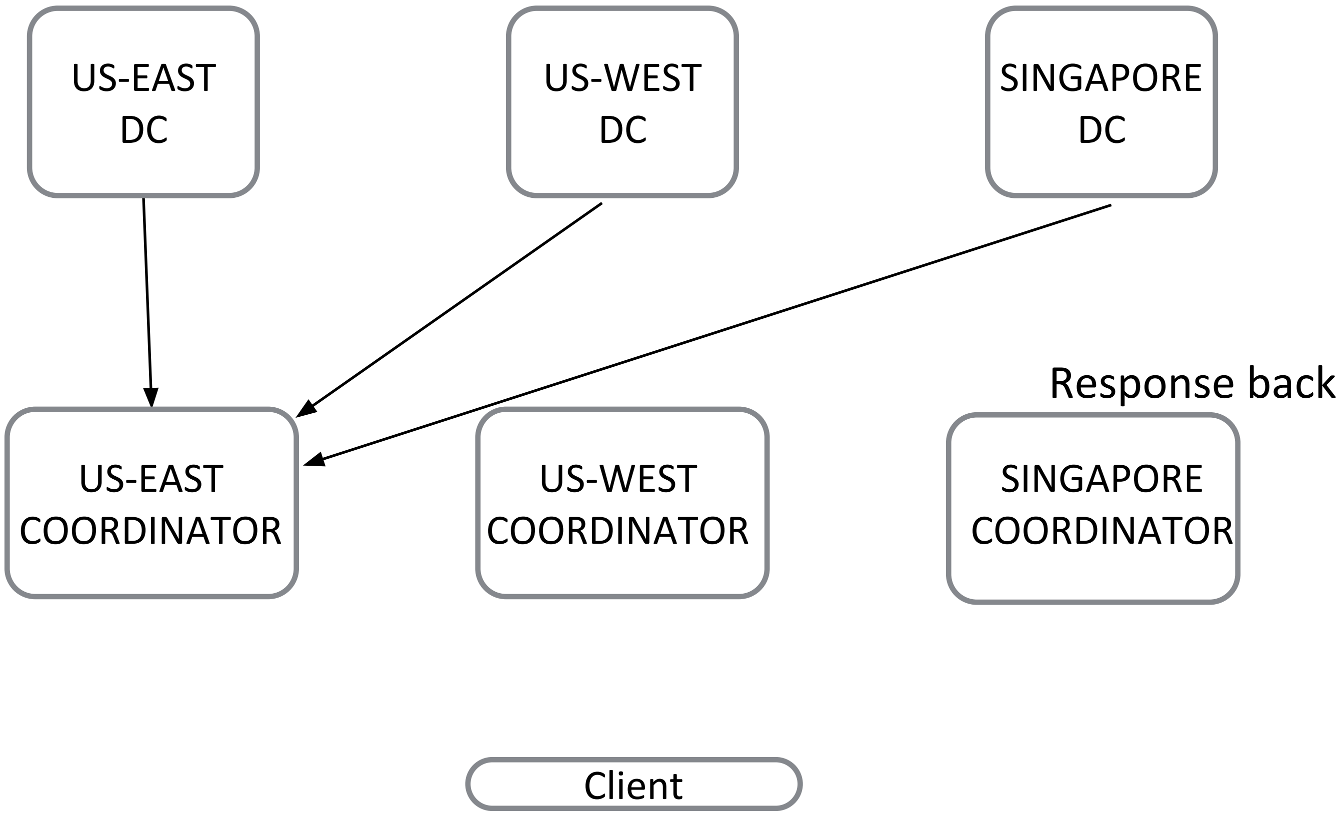
# Example workflow for a PUT request using strong consistency

- If US-EAST is responsible for key “X”



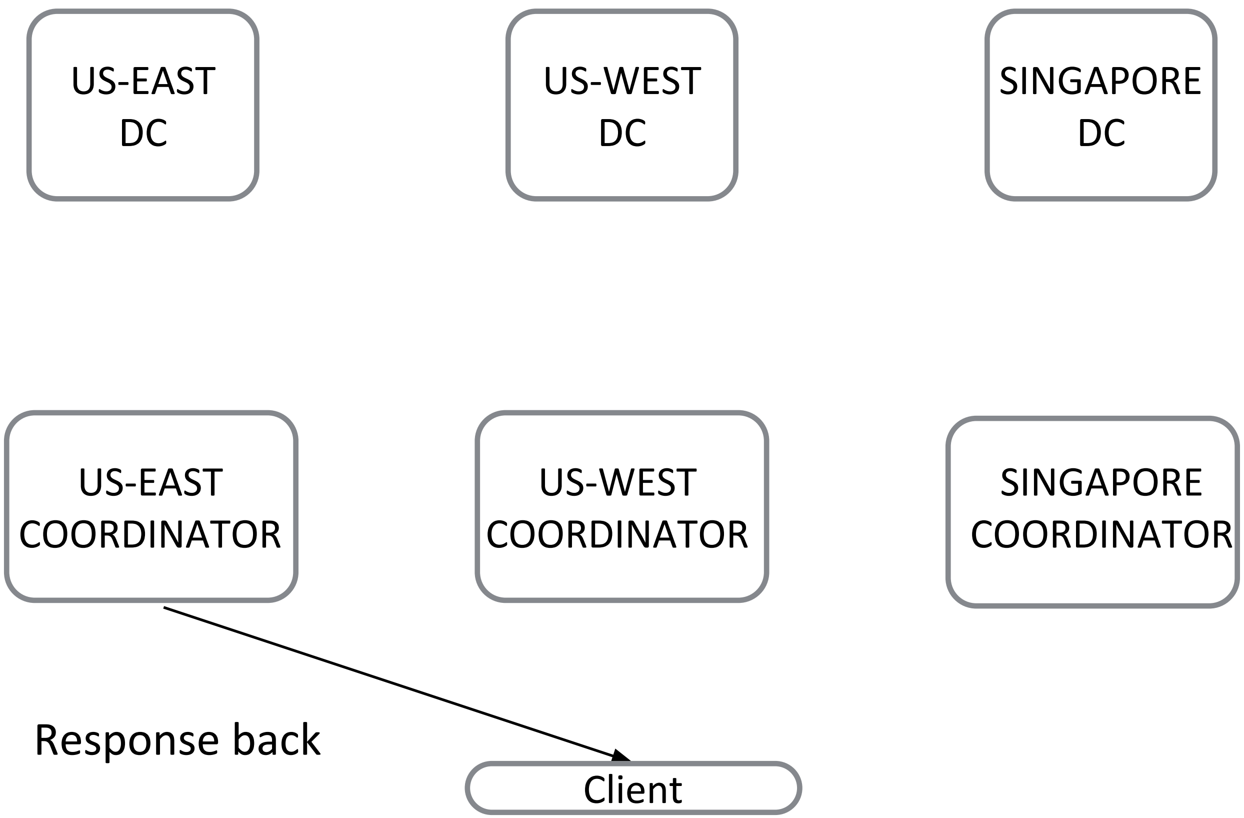
Example workflow for a PUT request using strong consistency

- If US-EAST is responsible for key "X"



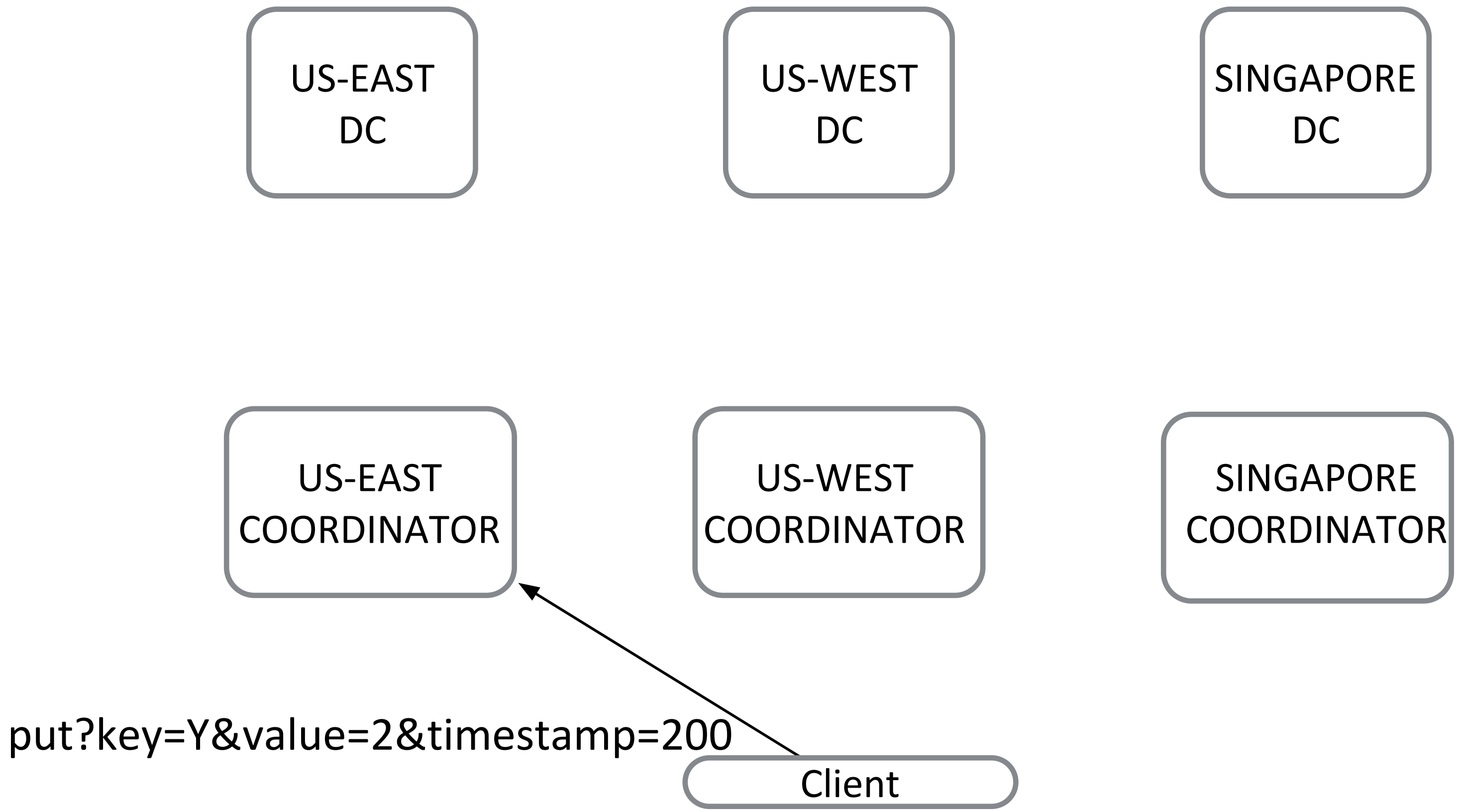
Example workflow for a PUT request using strong consistency

- If US-EAST is responsible for key "X"



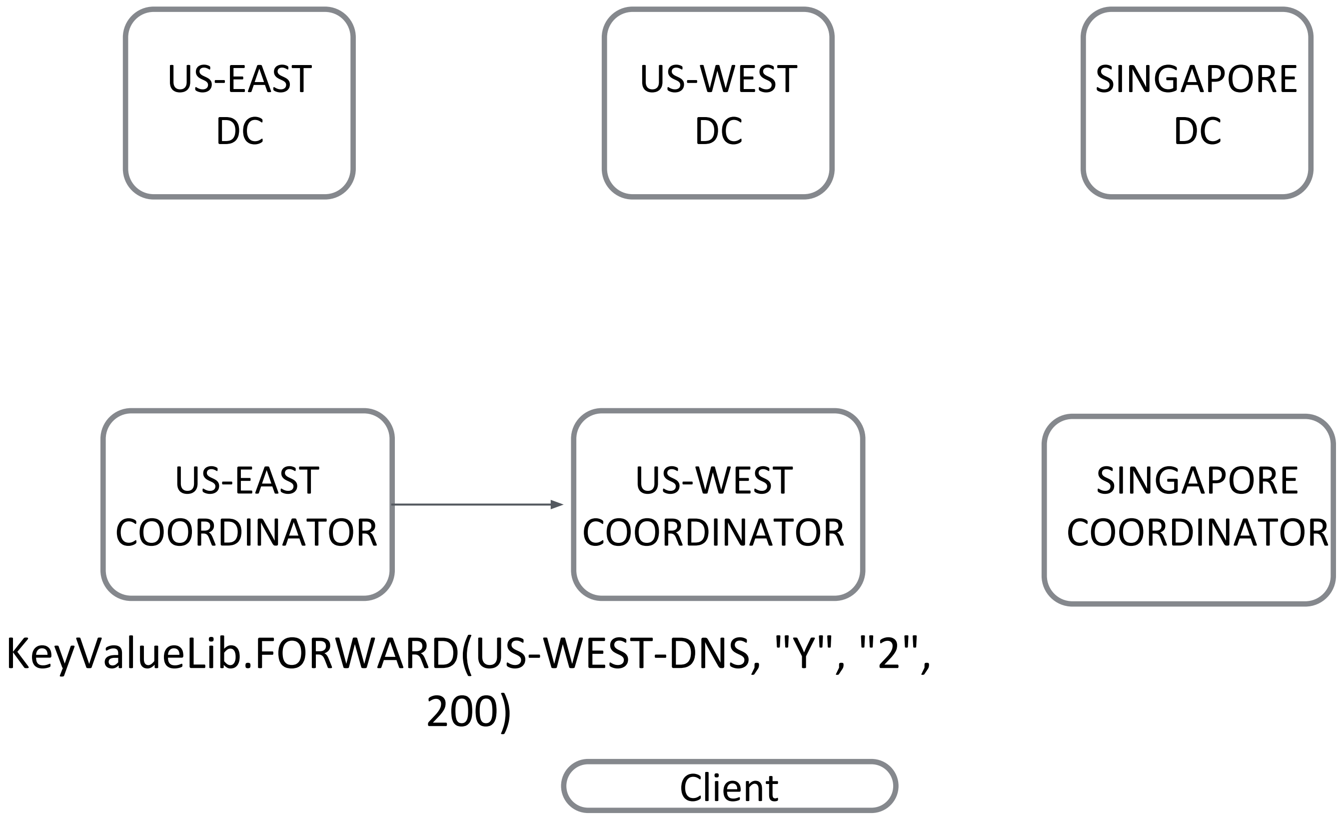
Example workflow for a PUT request using strong consistency

- If US-WEST is responsible for key “Y”



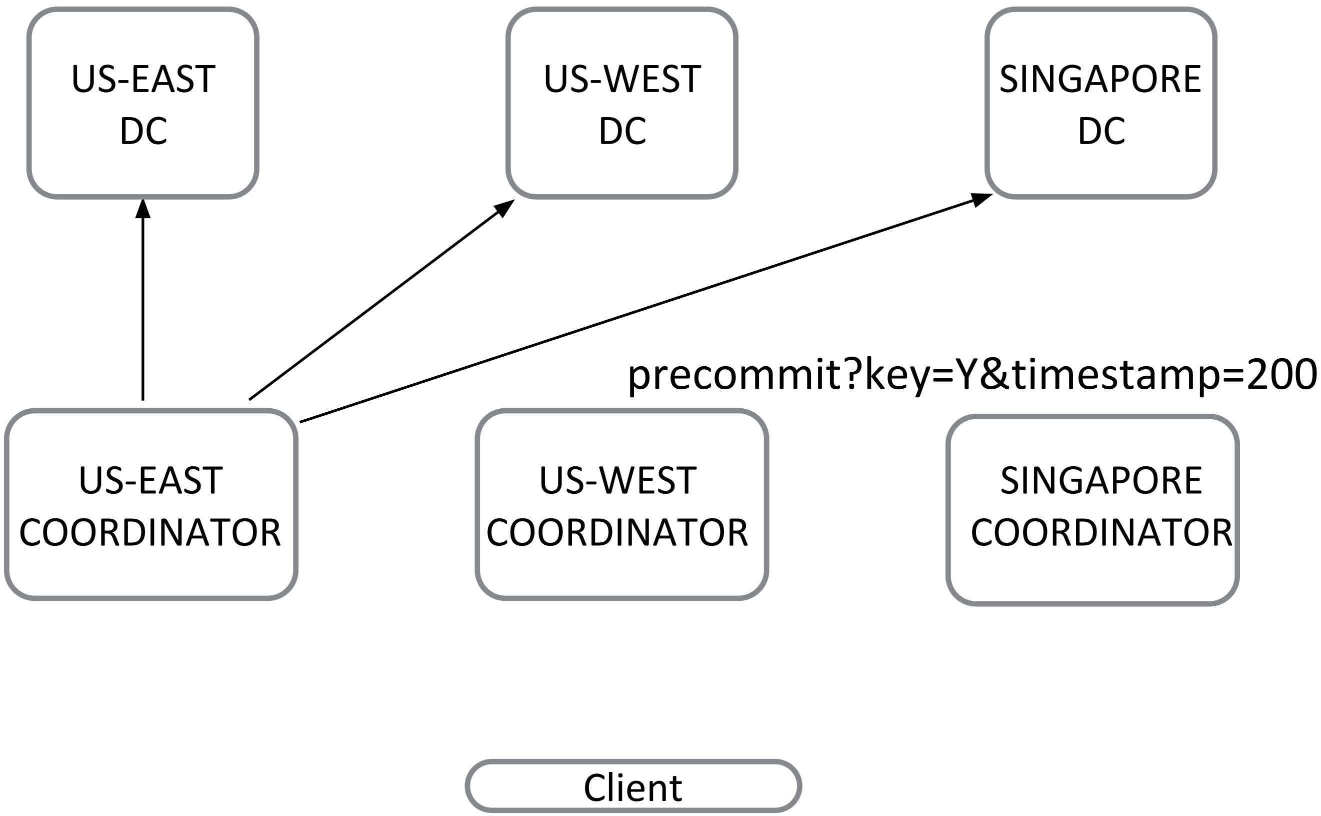
Example workflow for a PUT request using strong consistency

- If US-WEST is responsible for key “Y”



# Example workflow for a PUT request using strong consistency

- If US-WEST is responsible for key "Y"



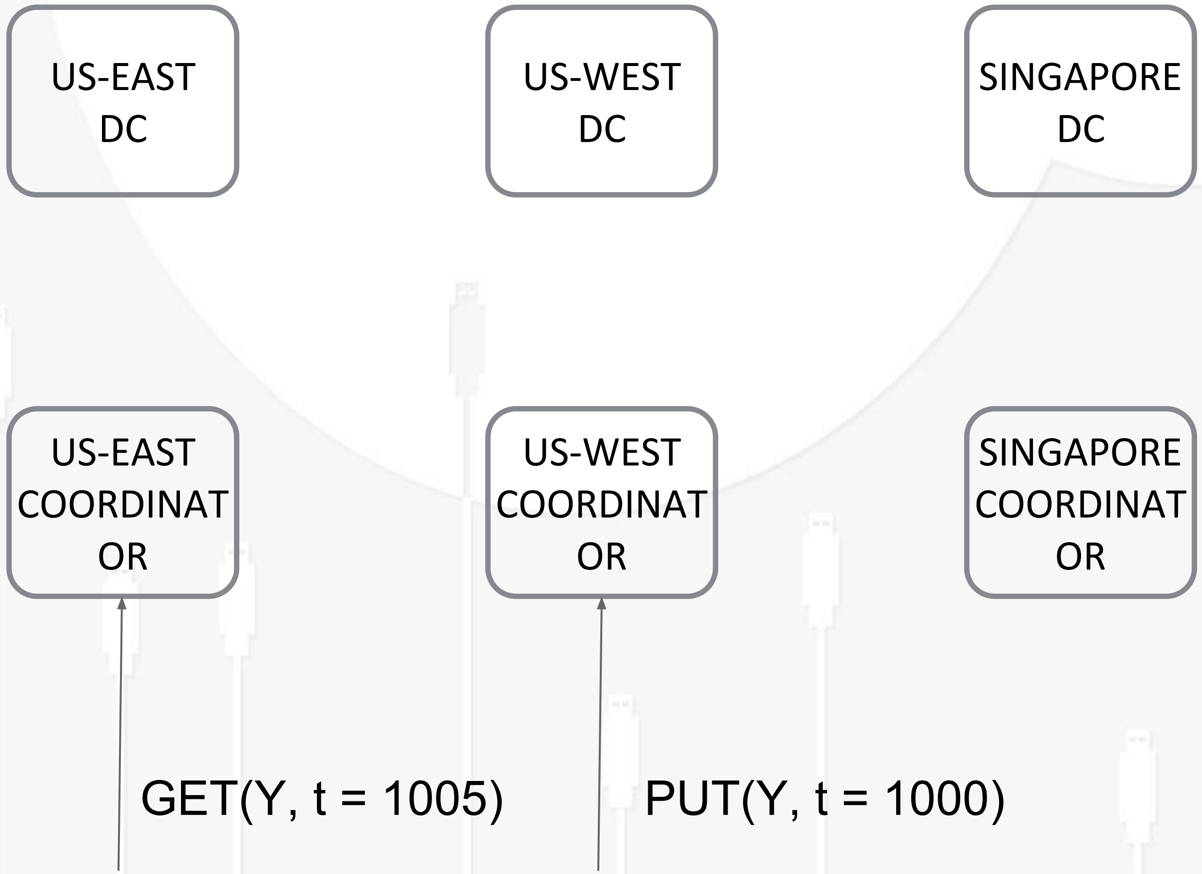
# P3.3: Eventual Consistency (Bonus)

- Write requests are performed in the order received by 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

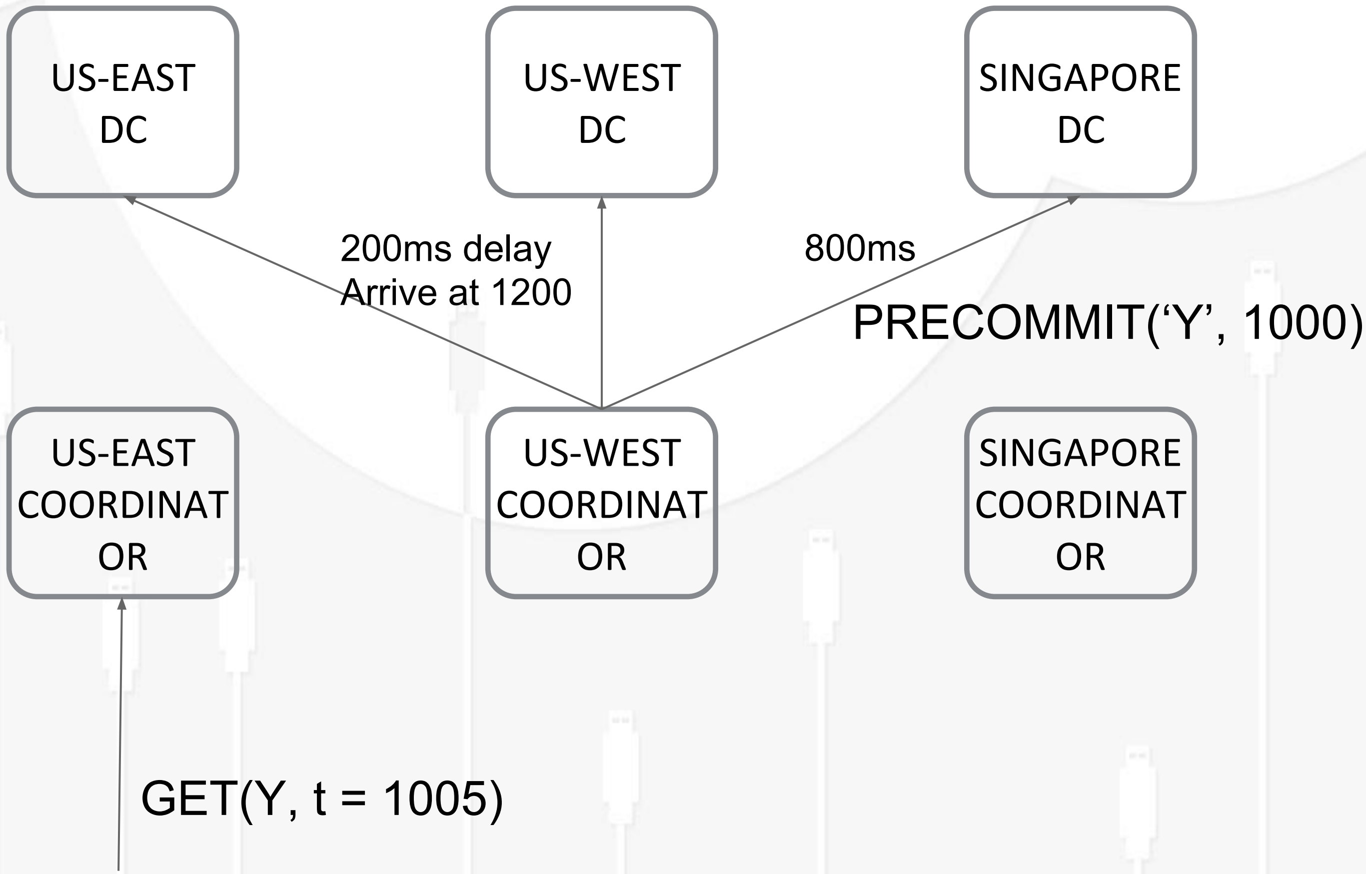
# More Hints

- In strong consistency, “PRECOMMIT” should be useful to help you lock requests because they are able to communicate with datastores
- Don’t wait for the PRECOMMIT messages that might be sent from other coordinators halfway, or you cannot pass all the test cases
- Lock by the key across all the datacenters in strong consistency
- Remember to update both KeyValueStore.java and Coordinator.java in Eventual Consistency

## Assume US-WEST is the primary Coordinator for key ‘Y’



# Assume US-WEST is the primary Coordinator for key ‘Y’



# Suggestions

- Read all three primers (PLEASE!)
- Consider the differences between the 2 consistency models before writing code
- Think about possible race conditions
- Read the hints in the writeup **carefully**
- Don’t modify any class except Coordinator.java and KeyValueStore.java

# How to Run Your Program

- Run “./sudo run_server.sh” to start the server on each of the data centers and coordinators.
- Use “./consistency_checker strong”, or “./consistency_checker eventual” to test your implementation of each consistency. (Our grader uses the same checker)
- If you want to test one simple PUT/GET request, you could directly send the request to datacenters or coordinators.

# Start early!

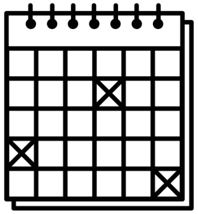
**Trickiest Individual Project!**

# TWITTER DATA ANALYTICS: TEAM PROJECT

# Team Project Phase-1 Deadlines

- Writeup and queries were released on Monday, **Feb 26th, 2018.**
- Phase 1 milestones:
  - Checkpoint 1:
    - Report, due on **Sunday, 3/11**
  - Checkpoint 2:
    - Q1 on scoreboard, due on **Sunday, 3/25**
  - Phase 1 Deadline:
    - Q2 on scoreboard, due on **Sunday, 4/1**
  - Phase 1, code and report:
    - due on **Tuesday, 4/3**

# Team Project Time Table



| Phase (and query due) | Start | Deadlines | Code and Report Due |
|---|---|---|---|
| Phase 1<br>• Q1, Q2 | Monday 02/26/2018<br>00:00:00 ET | Checkpoint 1, Report: Sunday 03/11/2018 23:59:59 ET<br>**Checkpoint 2, Q1: Sunday 03/25/2018 23:59:59 ET**<br>**Phase 1, Q2: Sunday 04/01/2018 23:59:59 ET** | Phase 1: Tuesday<br>04/03/2018 23:59:59 ET |
| Phase 2<br>• Q1, Q2,Q3 | Monday 04/02/2018<br>00:00:00 ET | Sunday 04/15/2018<br>15:59:59 ET | |
| Phase 2 Live Test (Hbase **AND** MySQL)<br>• Q1, Q2, Q3 | Sunday 04/15/2018<br>17:00:00 ET | Sunday 04/15/2018<br>23:59:59 ET | Tuesday 04/17/2018<br>23:59:59 ET |
| Phase 3<br>• Q1, Q2, Q3, Q4 | Monday 04/16/2018<br>00:00:00 ET | Sunday 04/29/2018<br>15:59:59 ET | |
| Phase 3 Live Test (Hbase **OR** MySQL)<br>• Q1, Q2, Q3, Q4 | Sunday 04/29/2018<br>17:00:00 ET | Sunday 04/29/2018<br>23:59:59 ET | Tuesday 05/01/2018<br>23:59:59 ET |

# Upcoming Deadlines

- Conceptual Topics: OLI (Modules 15, 16, & 17)
  Quiz 8 due: **<u>Friday</u>, 03/23/2018 11:59 PM Pittsburgh**

- P3.3: Replication and Consistency Models
  Due: **<u>Sunday</u>, 03/25/2018 11:59 PM Pittsburgh**

- Team Project: Phase 1 - Query 1
  Due: **<u>Sunday,</u> 03/25/2018 11:59 PM Pittsburgh**