15-319 / 15-619
Cloud Computing

Recitation 9

Mar 19, 2019

Overview

- **Last week's reflection**
  - Project 3.2
  - OLI Unit 4 - Module 14 (Storage)
  - Quiz 7
- **This week's schedule**
  - Project 3.3
  - OLI Unit 4 - Modules 15, 16 & 17
  - Quiz 8 due on Friday, March 22nd
- **Team Project, Twitter Analytics**
  - Q2M and Q2H correctness due on 3/24
  - Phase 1 due, Mar 31

Last Week

- **OLI : Module 14 - Cloud Storage**
  - **Quiz 7**
- **Project 3.2**
  - Social Networking Timeline with Heterogeneous Backends
    - MySQL
    - Neo4j
    - MongoDB
    - Choosing Databases
- **Multi-Threaded Online Programming Exercise on Cloud9**

This Week

- **OLI : Module 15, 16 & 17**
  - **Quiz 8** - Friday, March 22
- **Project 3.3** - Sunday, March 24
  - Task 1: Implement a Strong Consistency Model for distributed data stores
  - Task 2: Implement a Strong Consistency Model cross-region data stores
  - Bonus task: Implement an Eventual Consistency Model
- **Team Project, Twitter Analytics** - Sunday, March 24
  - Q2M and Q2H correctness
- **Online Programming Exercise** - Scheduling

Conceptual Topics - OLI 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 22nd**

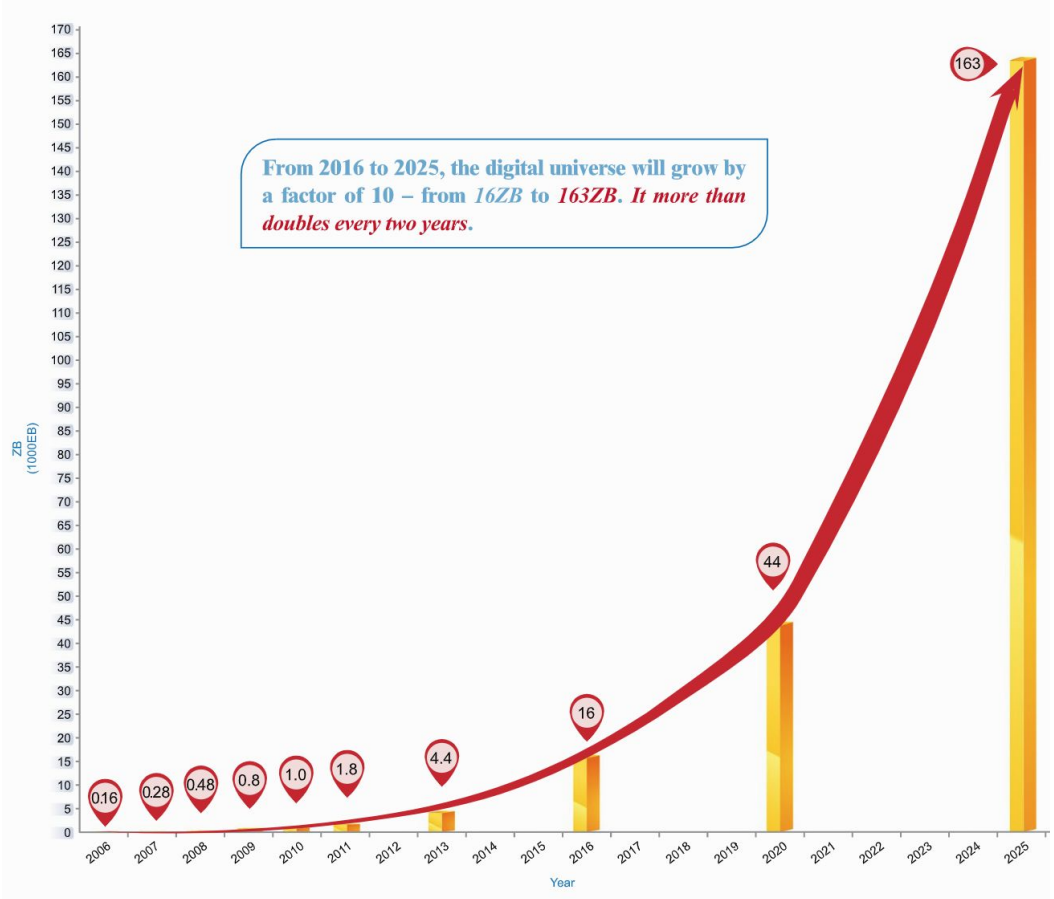
Individual Projects

- DONE
  - P3.1: Files vs Databases - comparison and Usage of flat files, MySQL, Redis, and HBase
  - NoSQL Primer
  - HBase Basics Primer
- Done
  - P3.2: Social networking with heterogeneous backends
  - MongoDB Primer
- **Now**
  - **P3.3: Replication and Consistency models**
  - Introduction to multithreaded programming in Java
  - Introduction to consistency models

Scale of Data is Growing

International Data Corporation's (IDC) Digital Universe Study predicts an increase in the amount of data created globally from

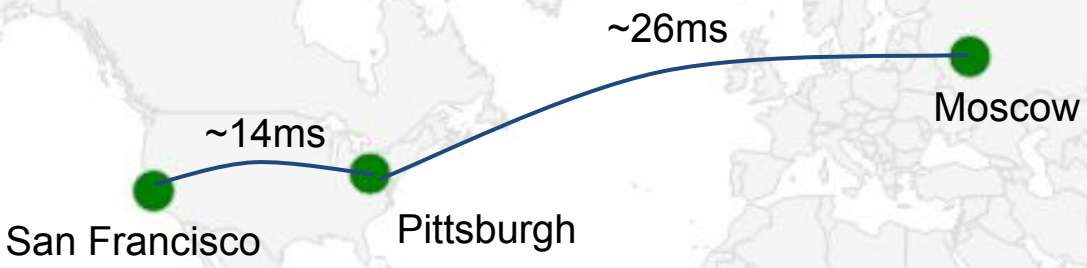
- 16 zettabytes in 2016
  to
- 160 zettabytes in 2025.



Guo H. Big Earth data: A new frontier in Earth and information sciences[J]. Big Earth Data, 2017, 1(1-2): 4-20.

Users are Global

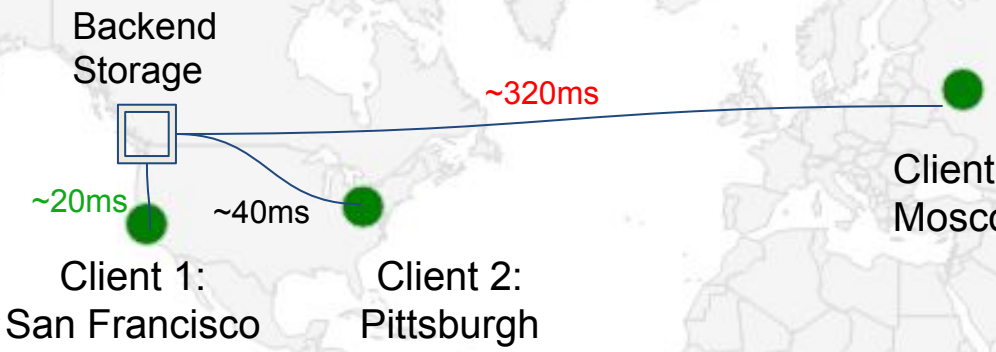
- Speed of Light (≈3.00×10^8 m/s)
- Inherent latencies



Typical End-To-End Latency

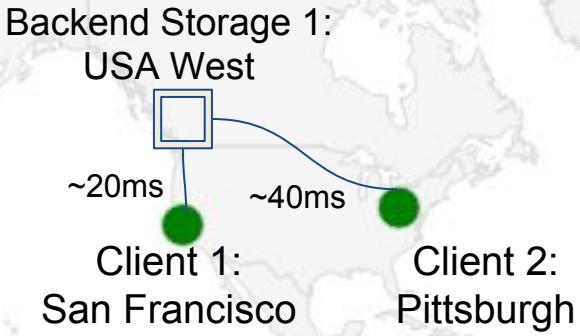
- Typical end-to-end latency
  - The client sends the request to the server
    - Network latency
  - The backend processes the request and sends the response
    - Overhead of fetching and processing data from the backend
    - Network latency
  - The client receives the response

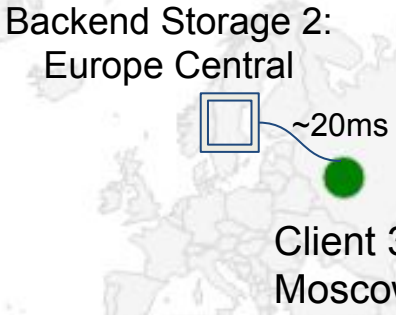
Latency with a Single Backend



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

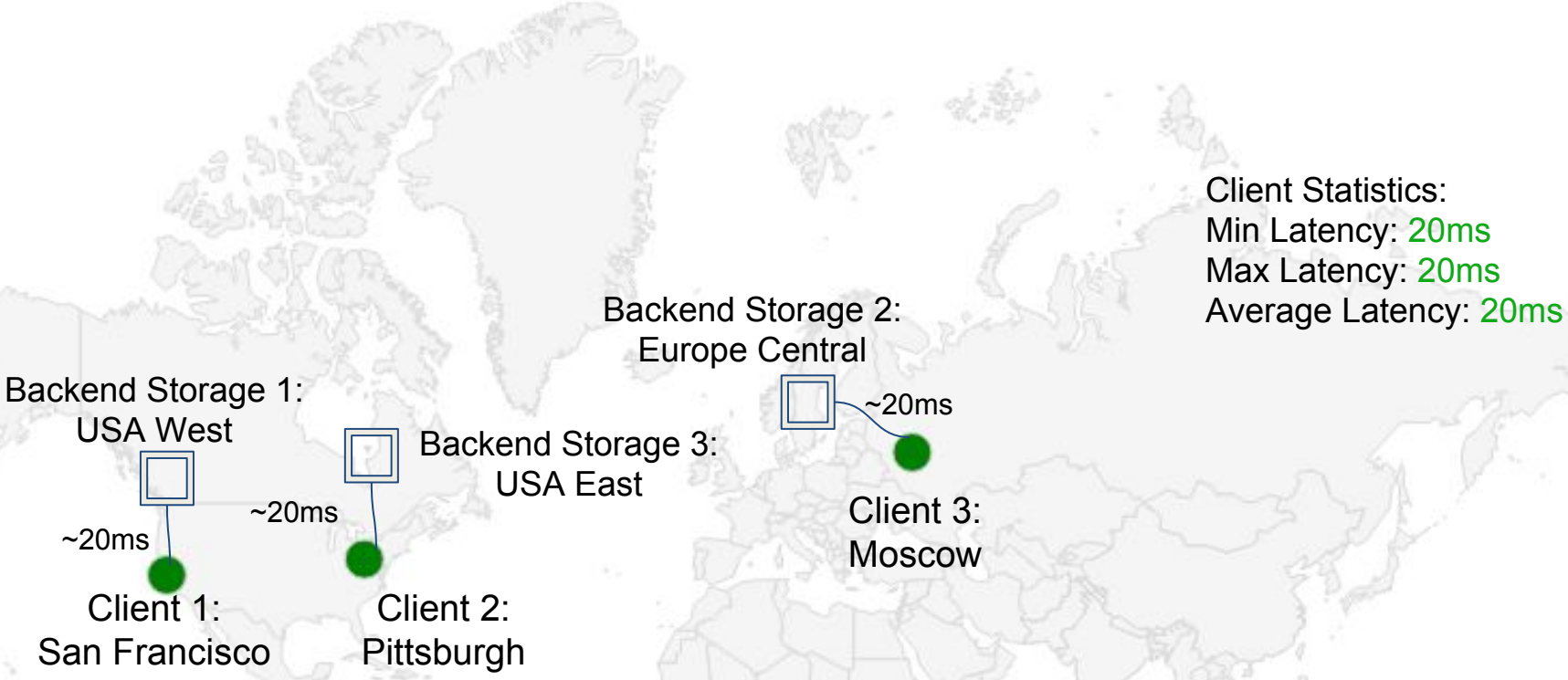
Replicate the Data Globally





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

Replicate the Data Close to Users

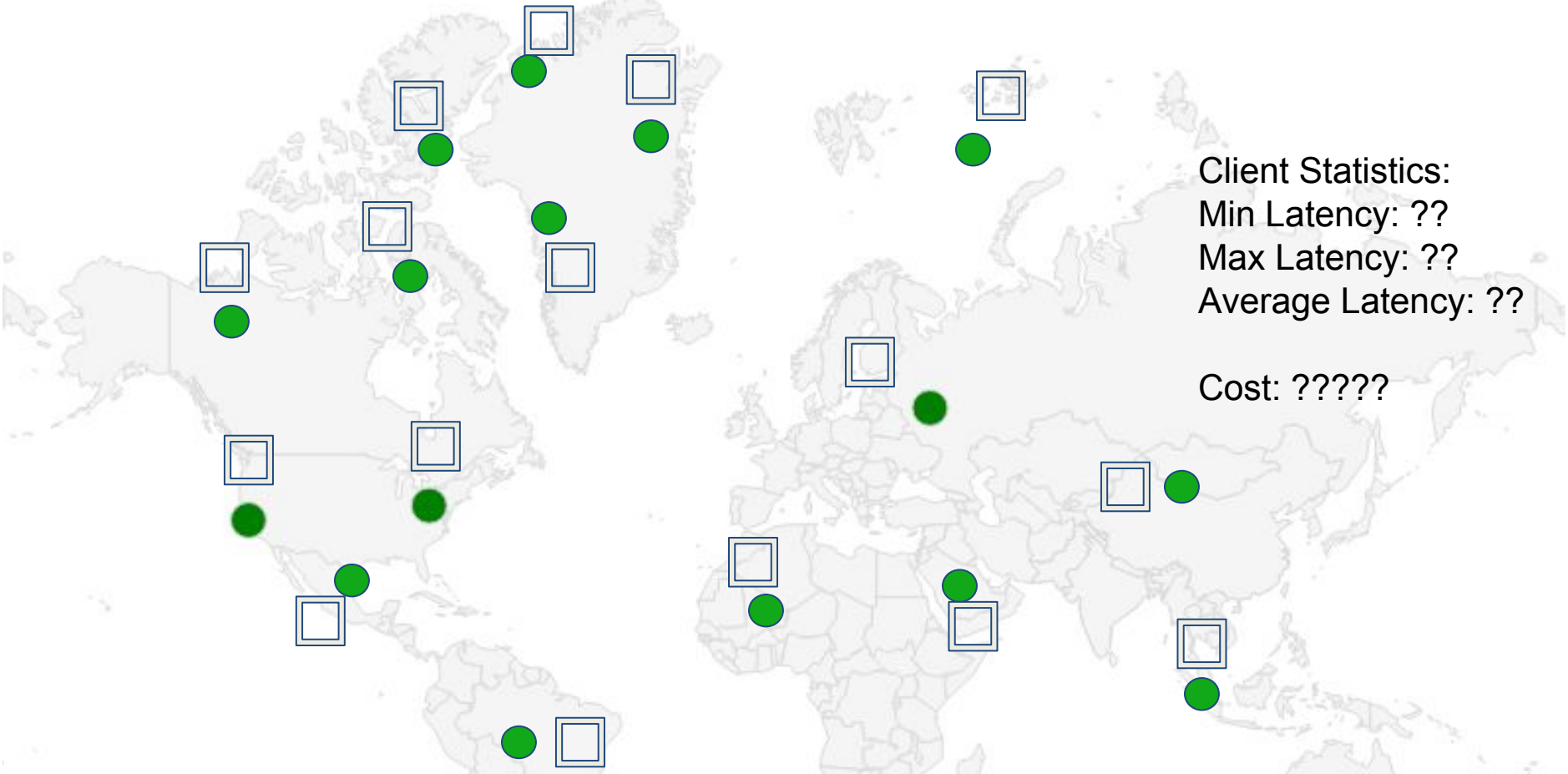


Client Statistics:
Min Latency: 20ms
Max Latency: 20ms
Average Latency: 20ms

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?

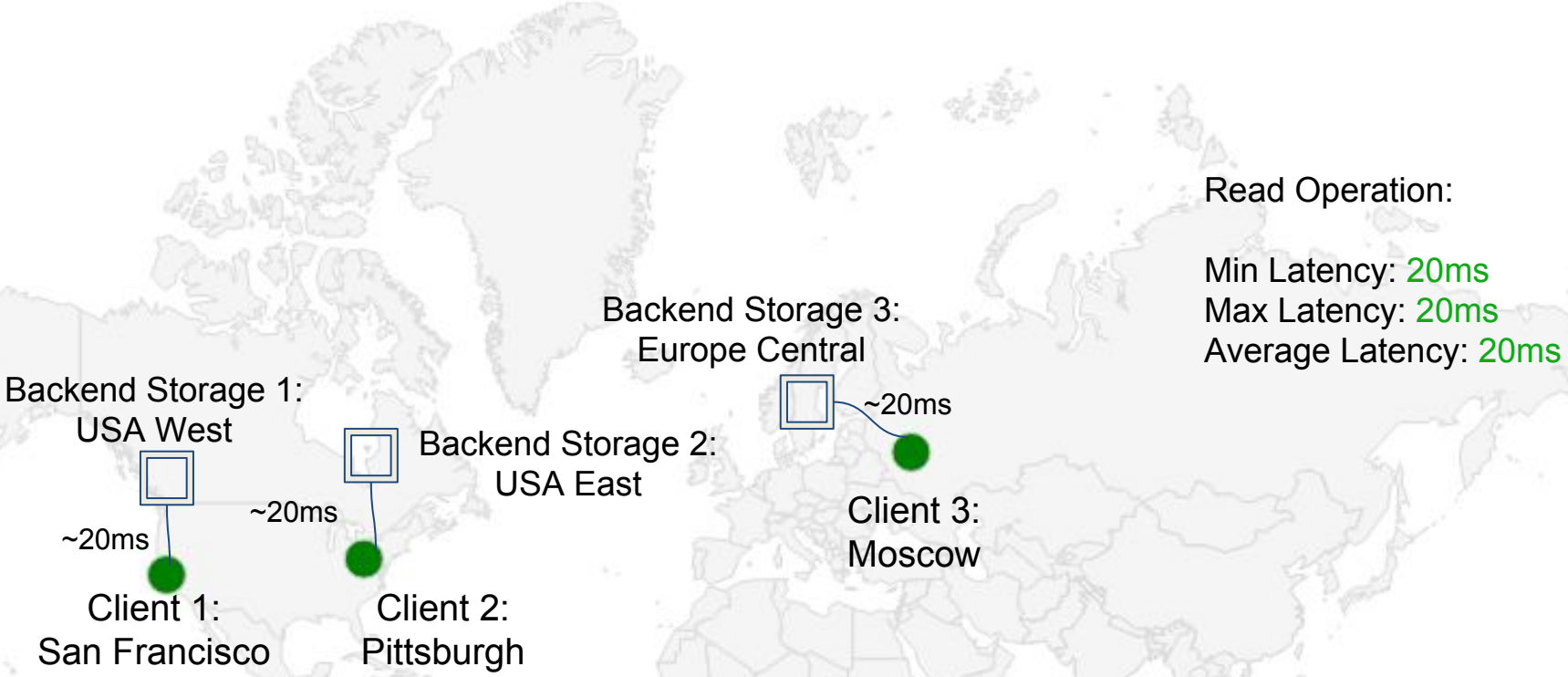


Client Statistics:
Min Latency: ??
Max Latency: ??
Average Latency: ??

Cost: ?????

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

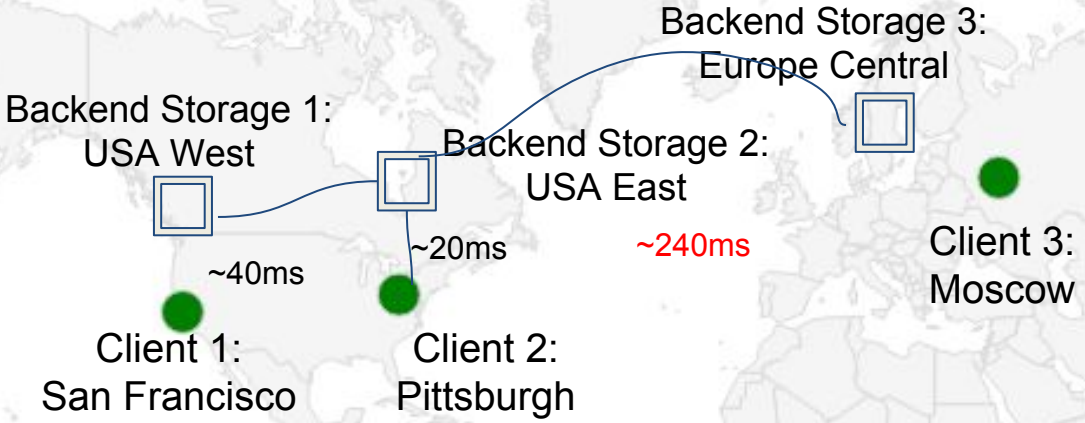
Replication READ



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

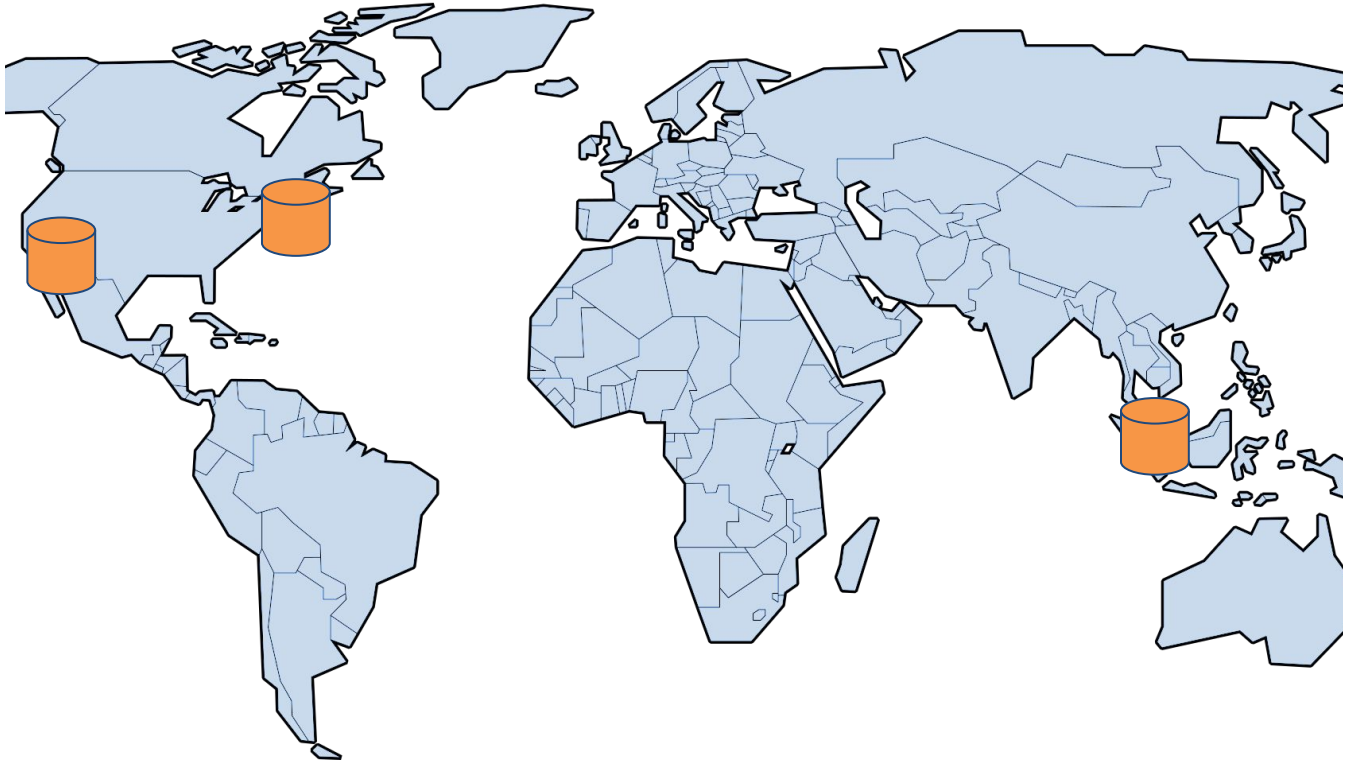
Pros and Cons of Replication

- 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

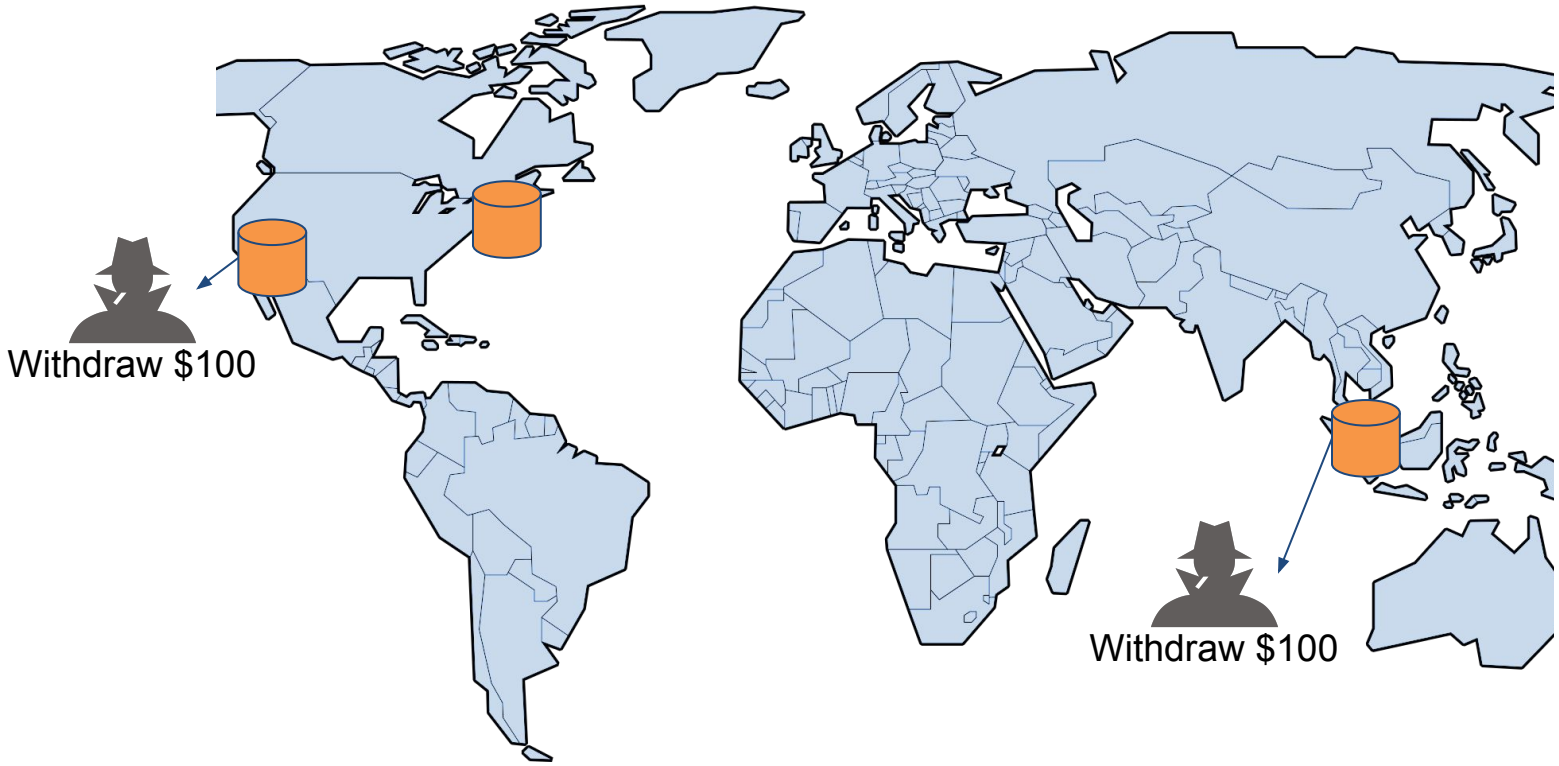
- Data consistency across replicas is important
  - Five consistency levels:
    Strict, Strong (Linearizability), Sequential, Causal and Eventual Consistency
- **This week's task**: Implement Strong Consistency
  - All datastores must return the same value for a key at all times
  - The order in which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

Choosing a Consistency Level
Bad Example



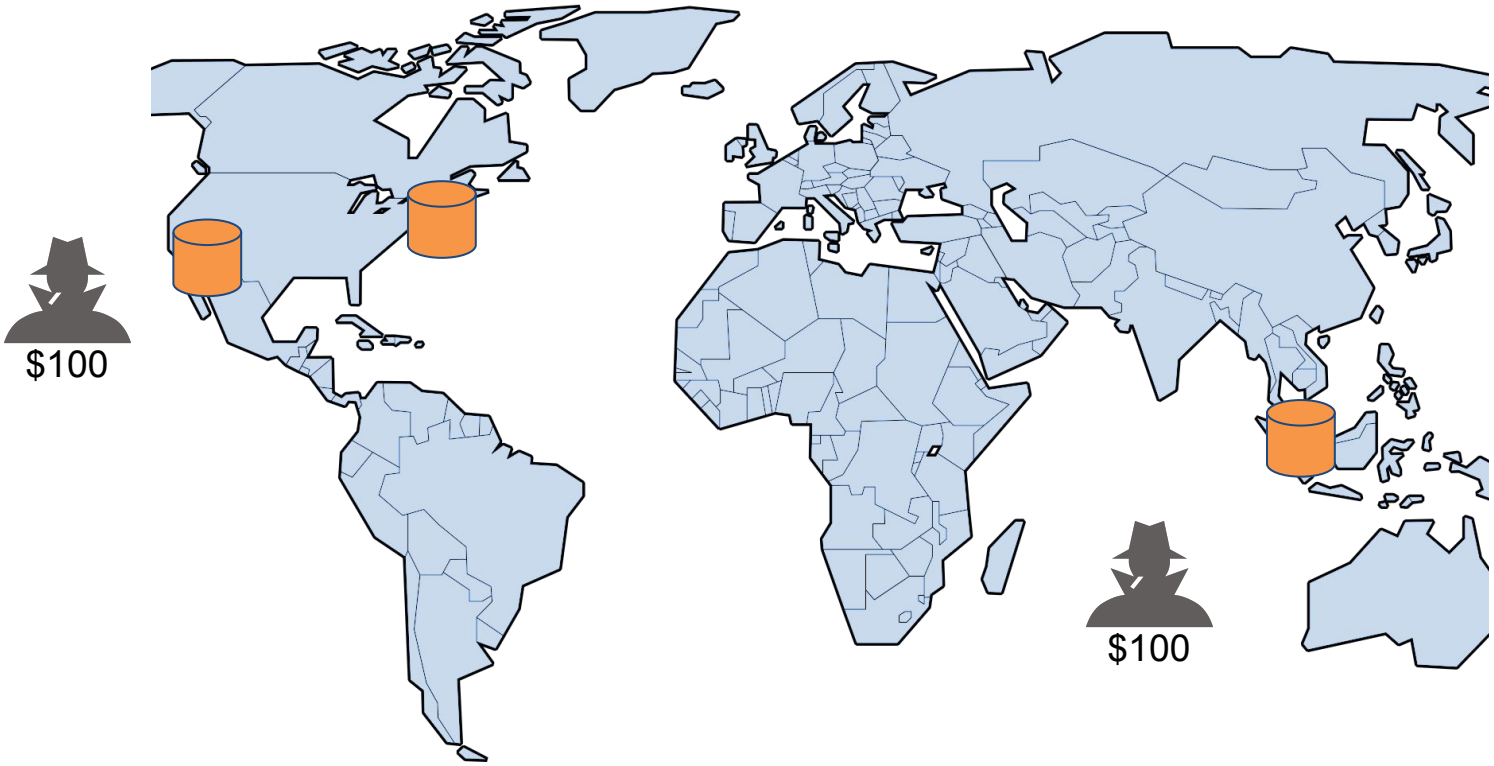
Account	Balance
xxxxx-4437	$100

Choosing a Consistency Level
Bad Example



Account	Balance
xxxxx-4437	$100

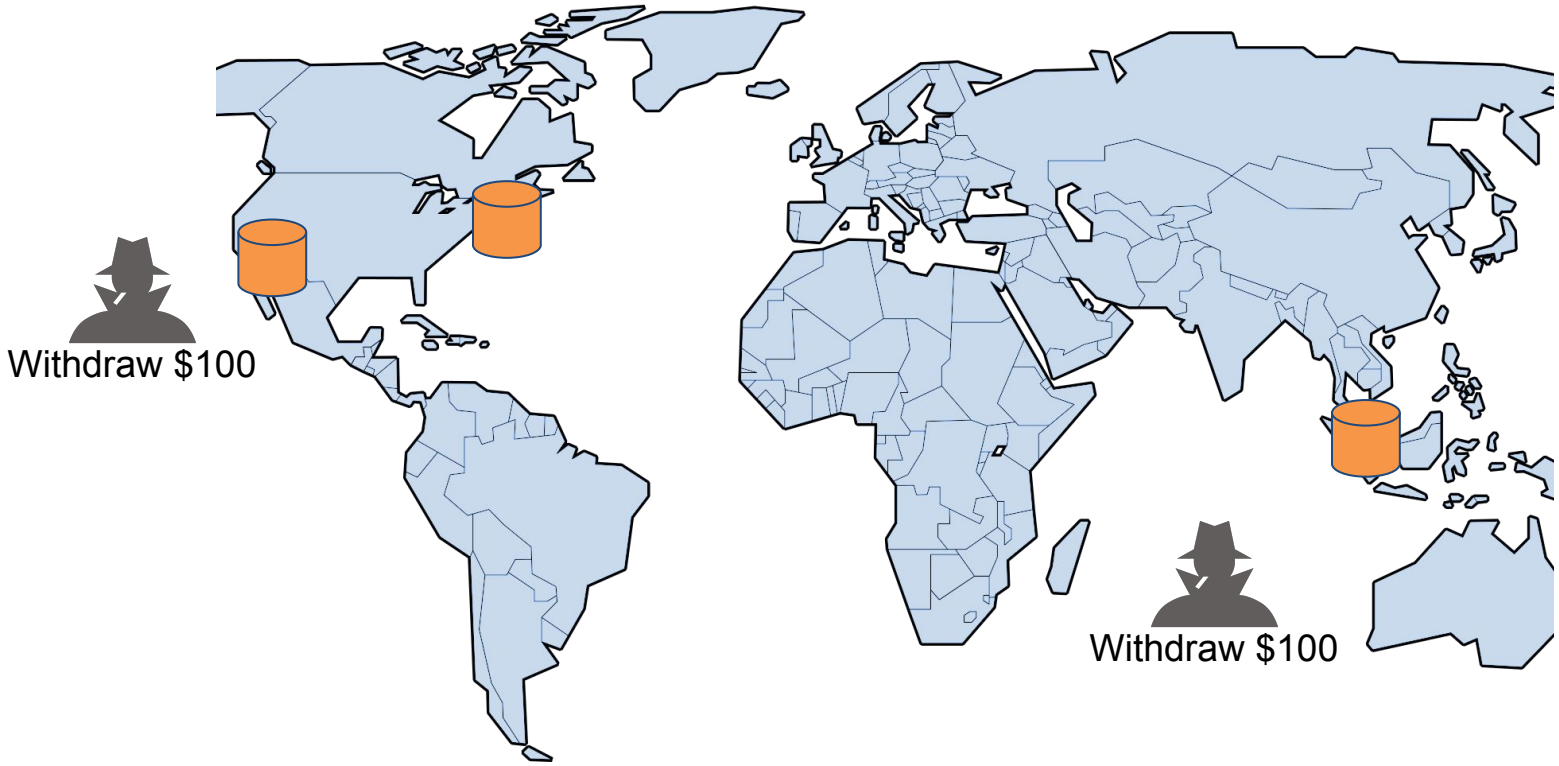
Choosing a Consistency Level
Bad Example



Account	Balance
xxxxx-4437	**$0**

Bank lost $100

Choosing a Consistency Level
Good Example

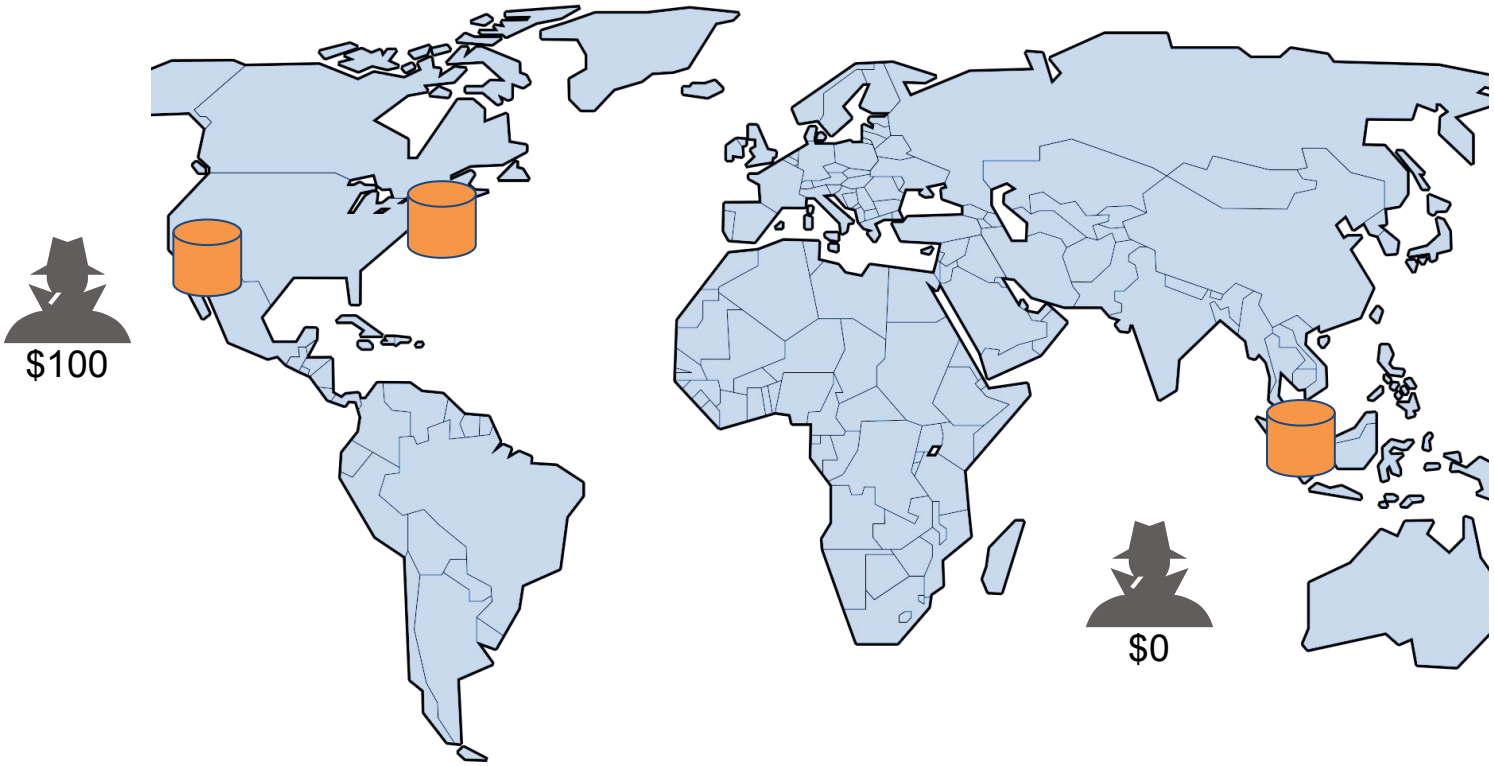


Account	Balance
xxxxx-4437	**$100**

Choosing a Consistency Level
Good Example

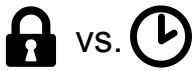
Account	Balance
xxxxx-4437	**$100**

Choosing a Consistency Level
Good Example



Account	Balance
xxxxx-4437	**$0**

P3.3: Consistency Models

Tradeoff:  Consistency vs. Latency

- Strict
- Strong
- Sequential
- Causal
- Eventual

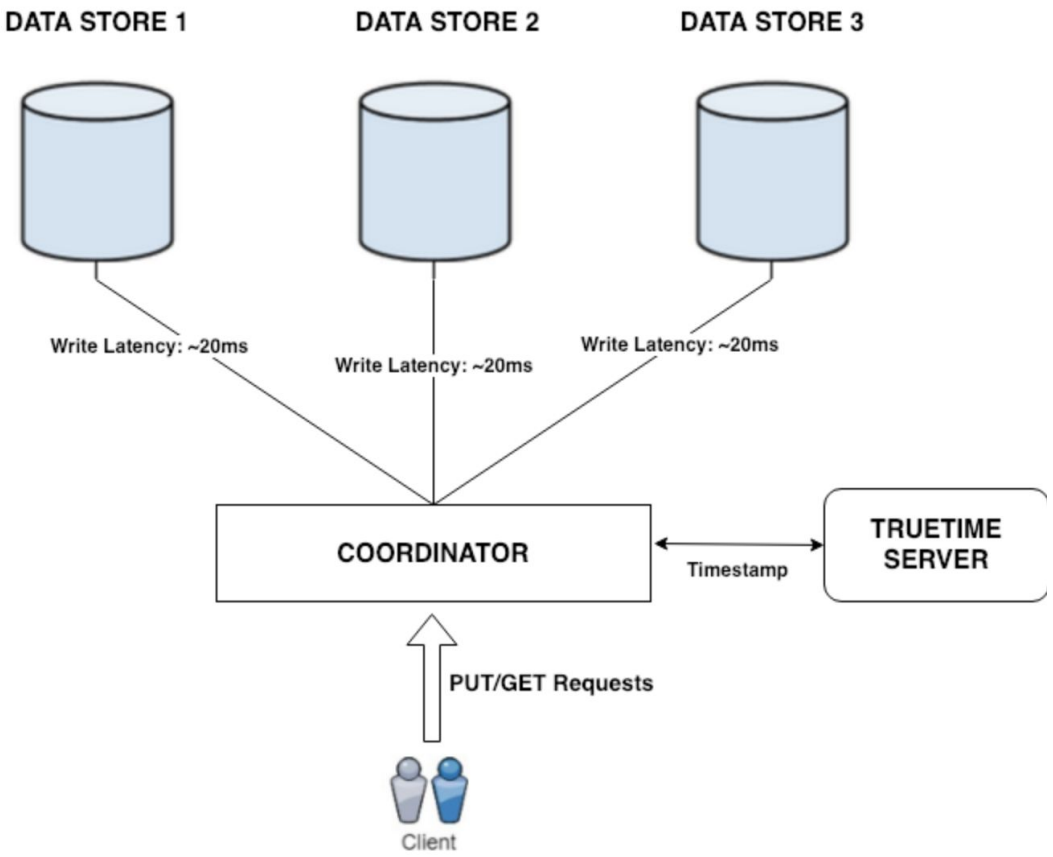
P3.3 Task 1: Strong Consistency

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



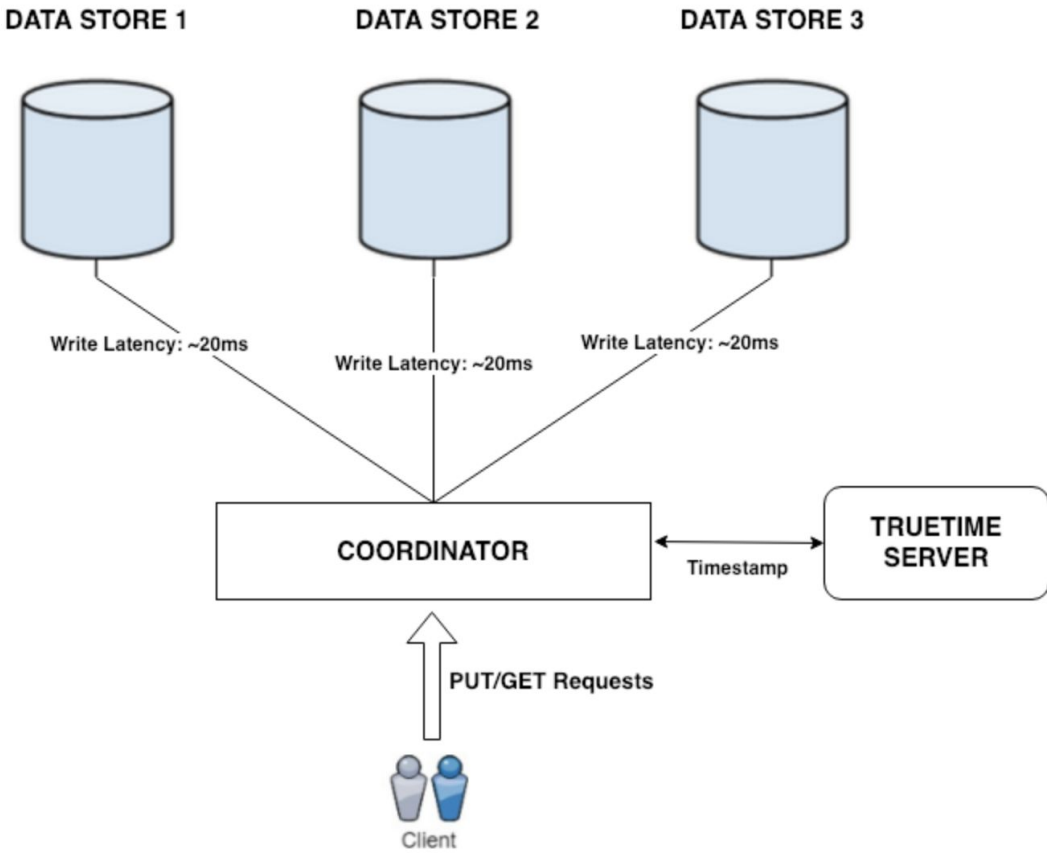
P3.3 Task 1: Strong Consistency

Single PUT request for key ‘X’

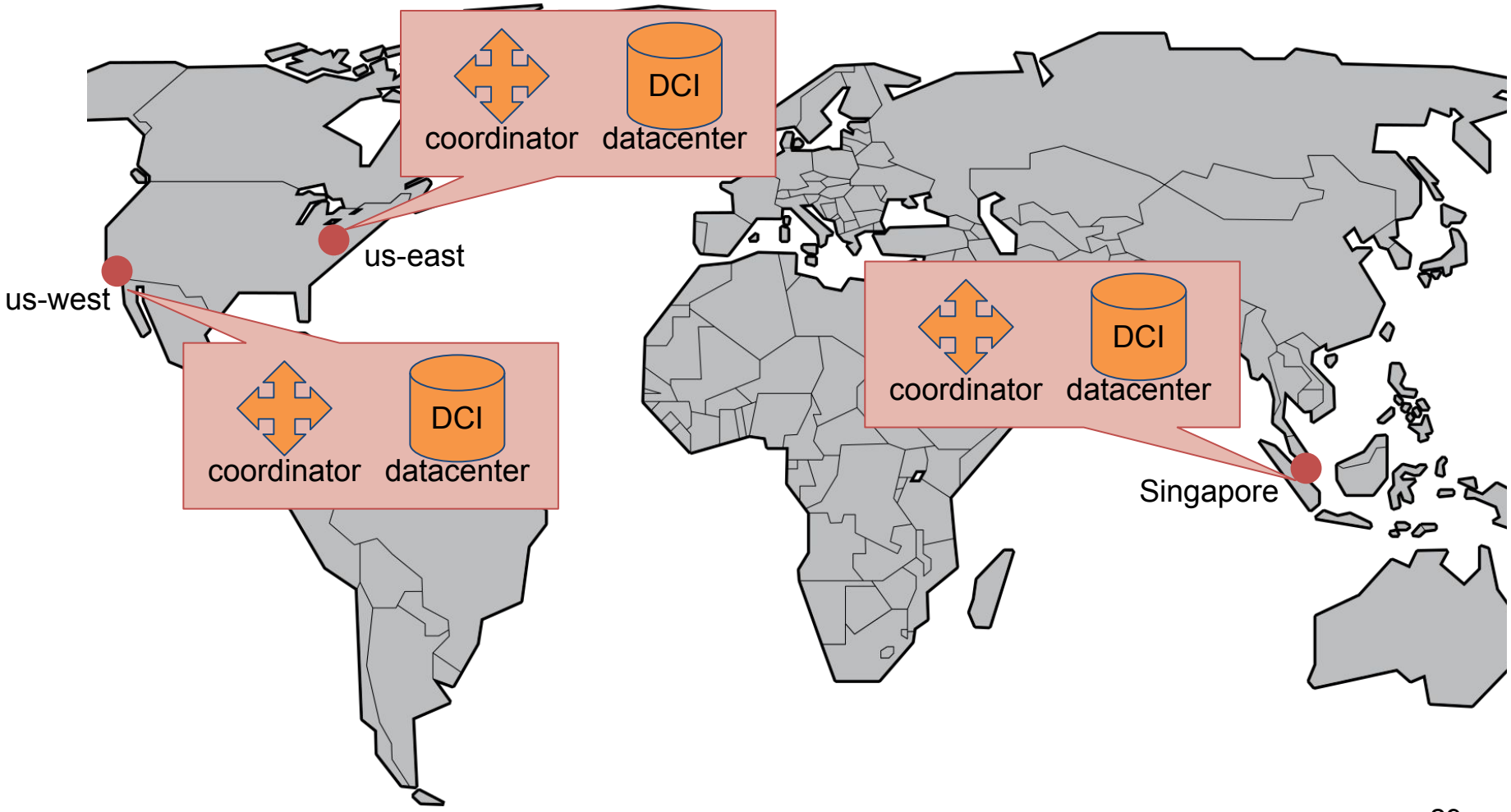
- Block all GET for key ‘X’ until all datastores are updated
- GET requests for a different key ‘Y’ should not be blocked

Multiple PUT requests for ‘X’

- Resolved in order of their timestamp received from the Truetime Server.
- Any GET request in between 2 PUTs must return the first PUT value



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



P3.3 Tasks 1 & 2: Strong Consistency

- Every request has a global timestamp order

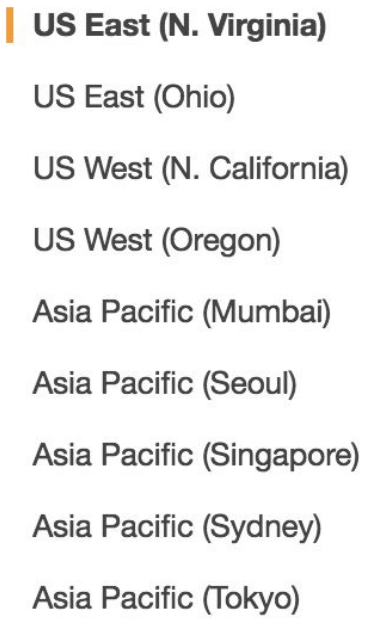
where timestamp is issued by a Truetime Server.

- Operations must be ordered by the timestamps

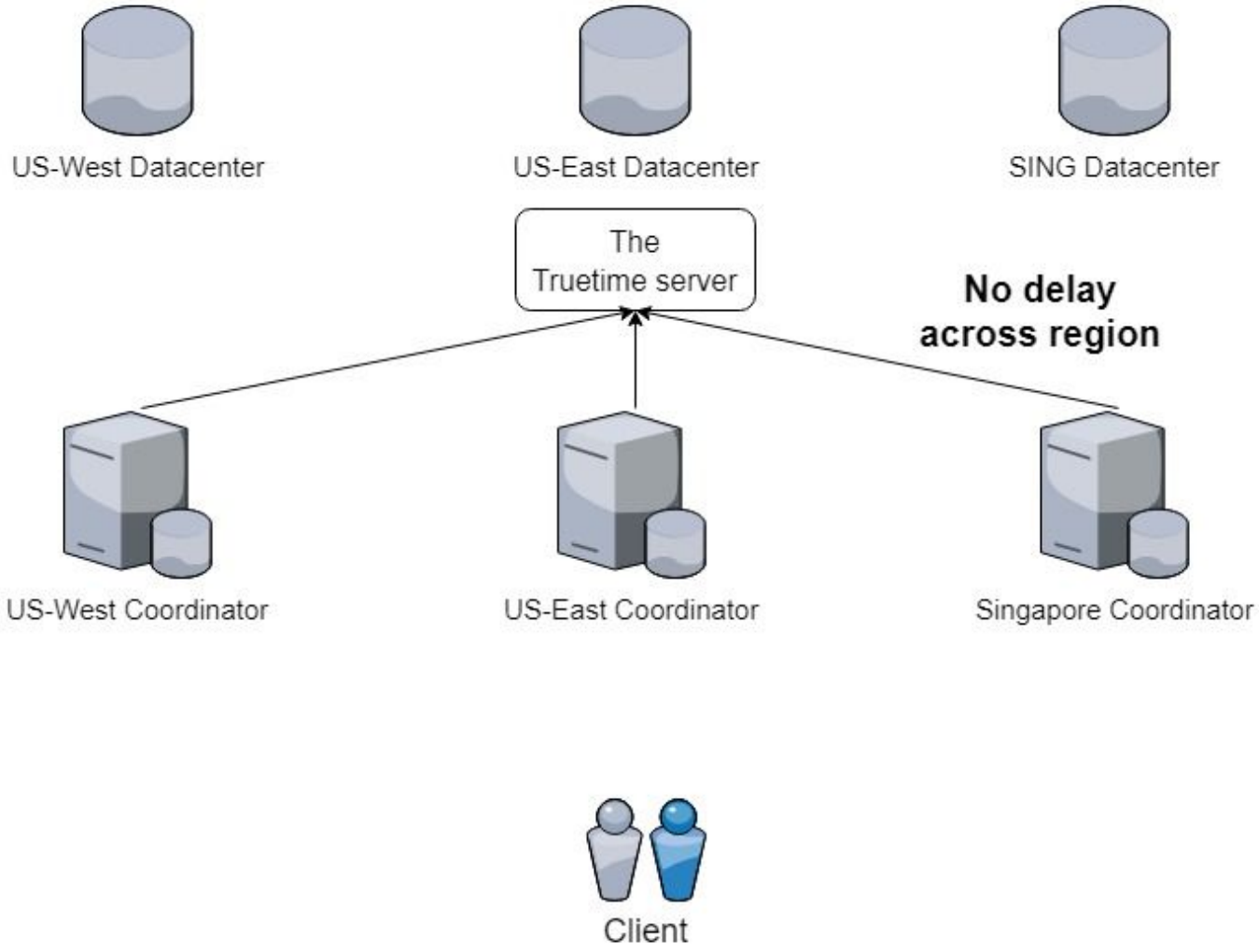
Requirement: At any given point of time, all clients should read the same data from any datacenter replica

Task 2 Workflow and Example

- Launch a total of 8 machines (3 data centers, 3 coordinators, 1 truetime server and 1 client).
- All machines should be launched in the US East region. We will simulate global latencies for you.
- The “US East” here has nothing to do with the simulated location of datacenters and coordinators in the project.
- Your task: implement the code for the Coordinators and Datastores



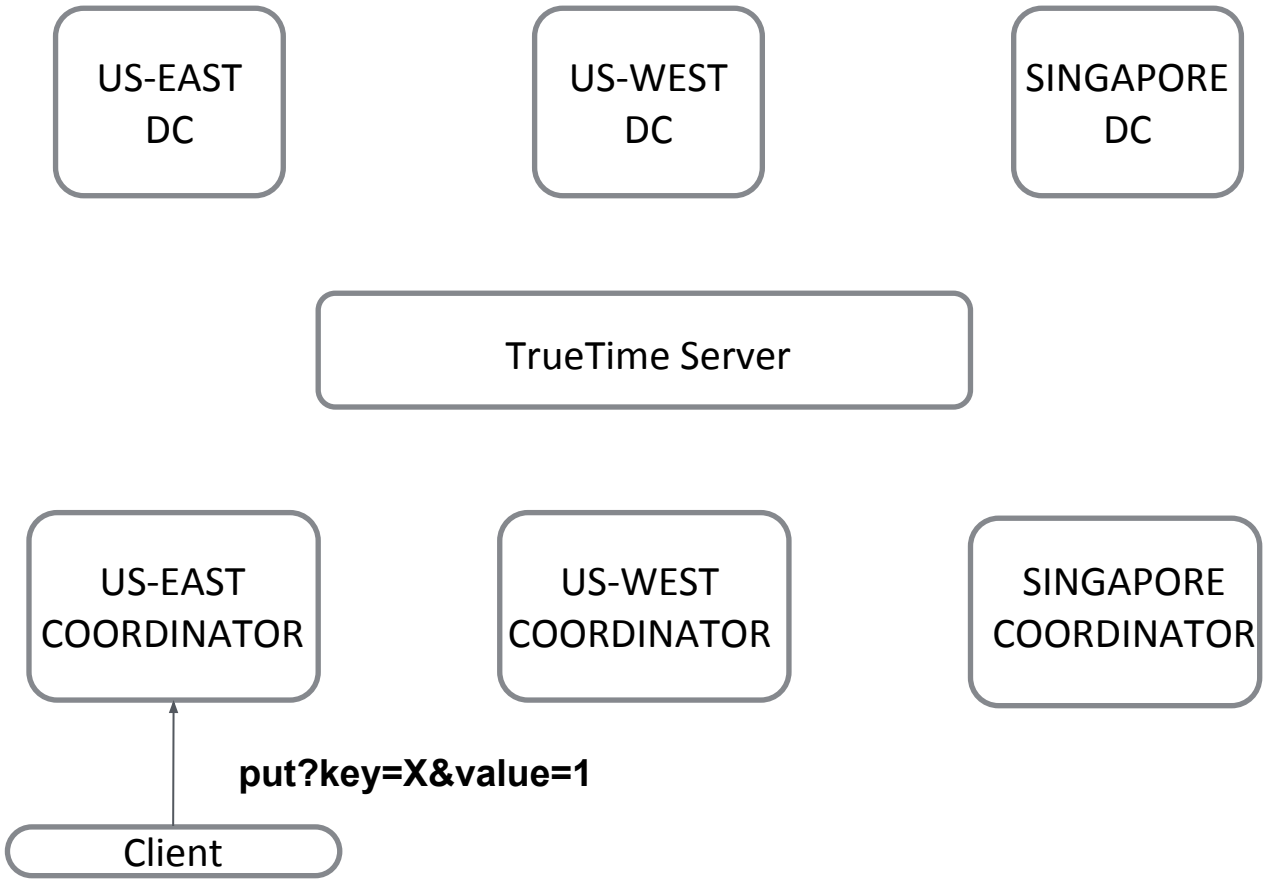
P3.3 Task 2: Architecture



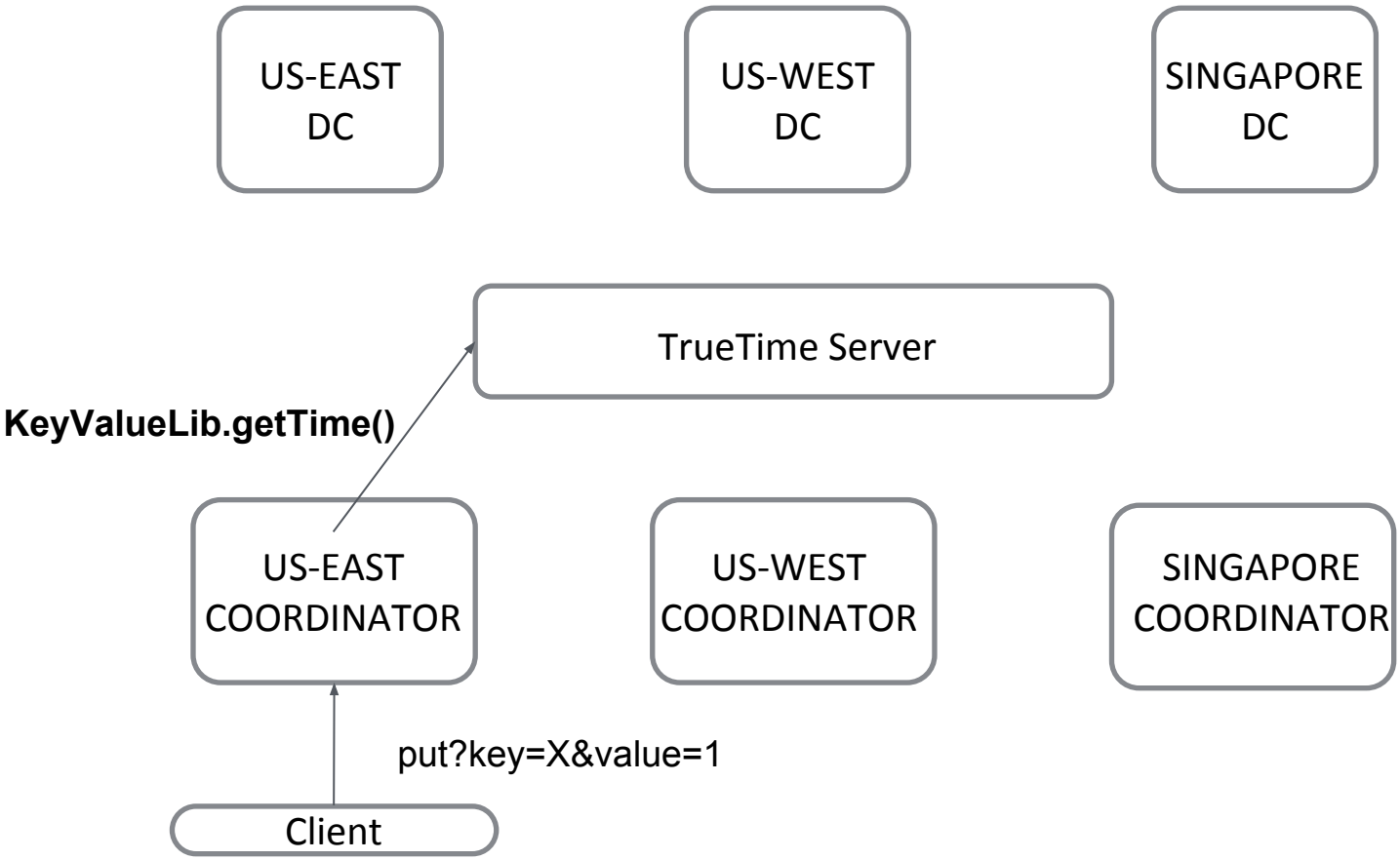
PRECOMMIT

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

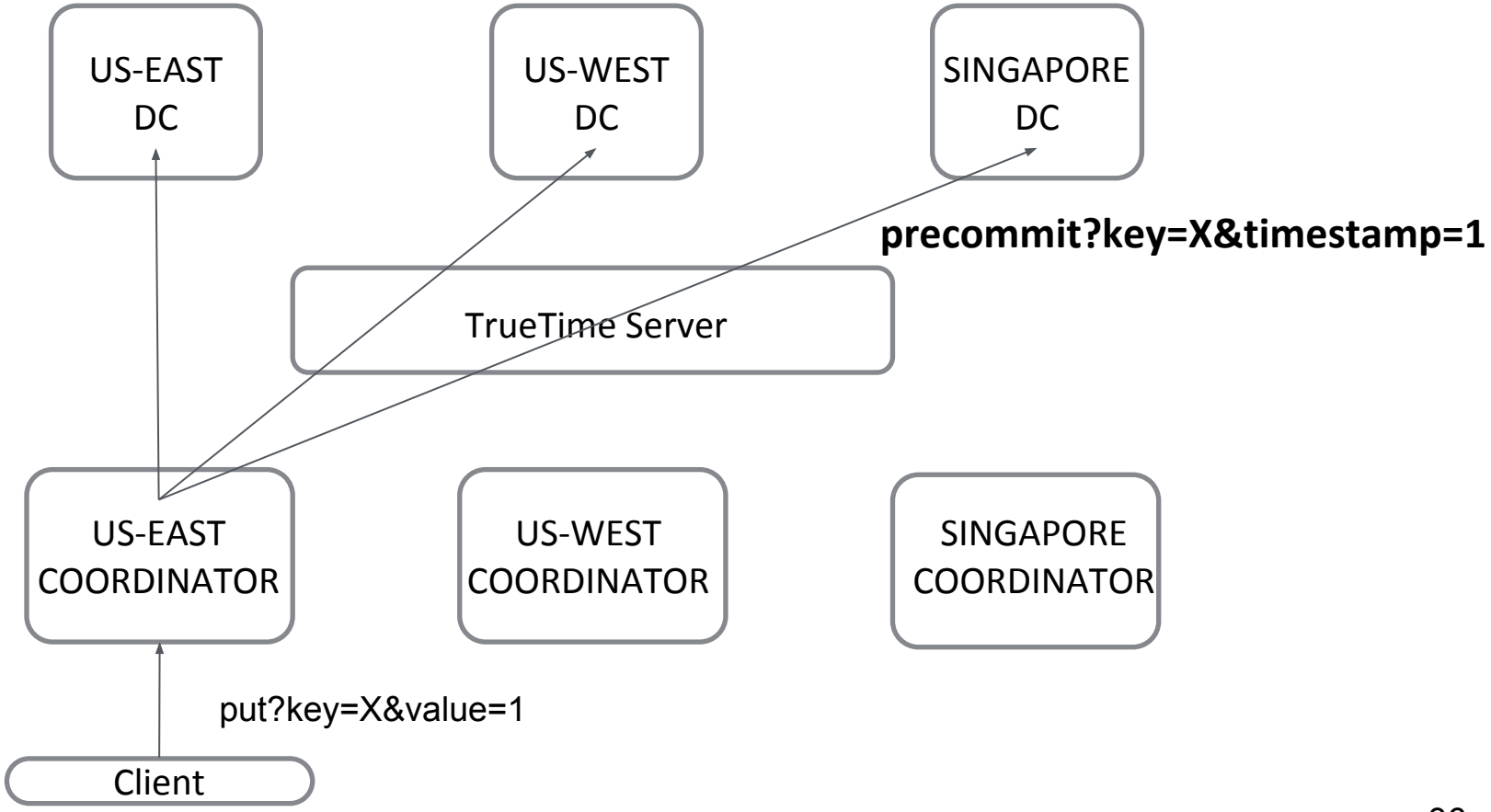
P3.3 Task 2:
Complete KeyValueStore.java (in DCs) and Coordinator.java (in Coordinators)



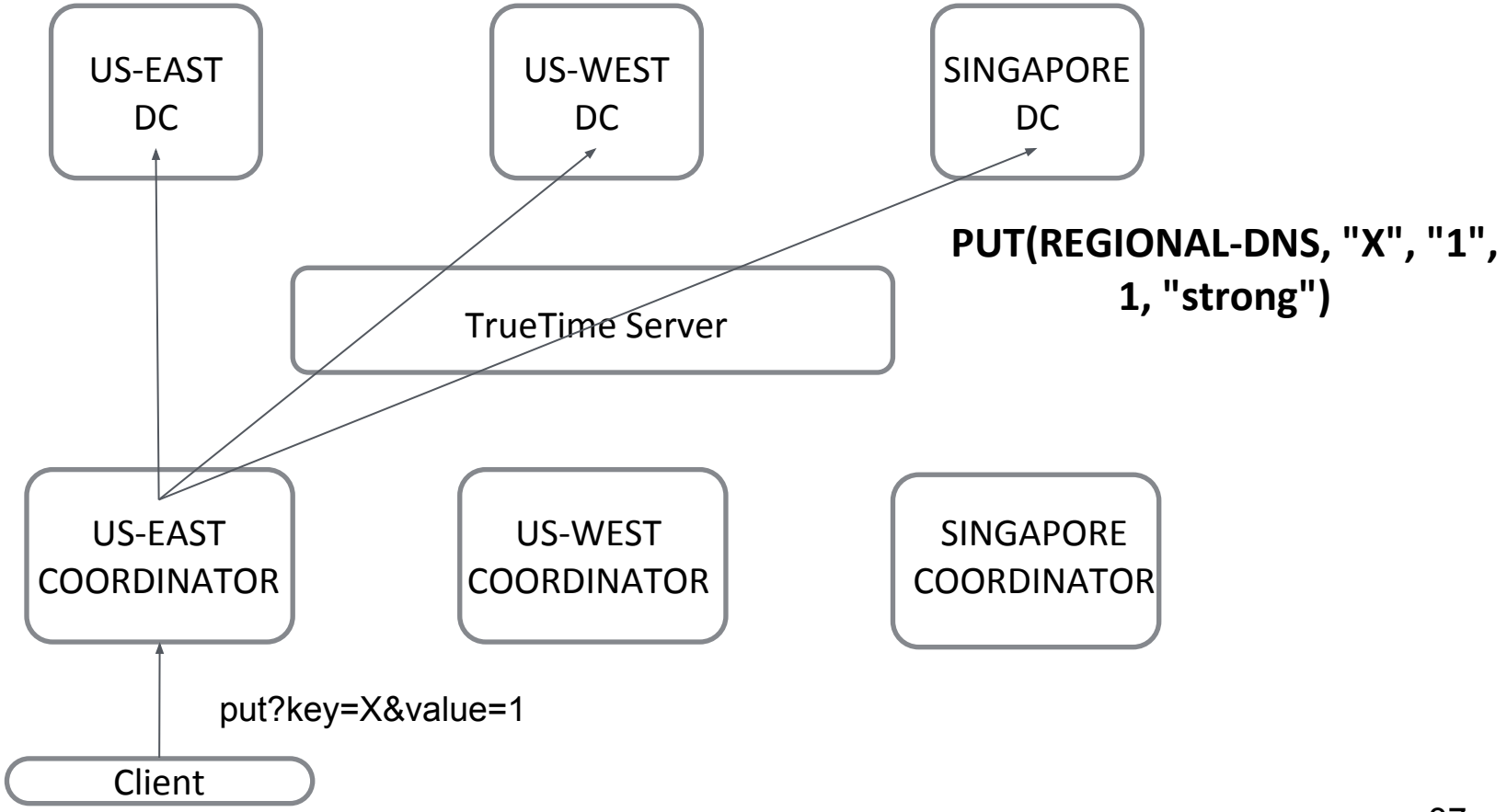
P3.3 Task 2:
Complete KeyValueStore.java (in DCs) and Coordinator.java (in Coordinators)



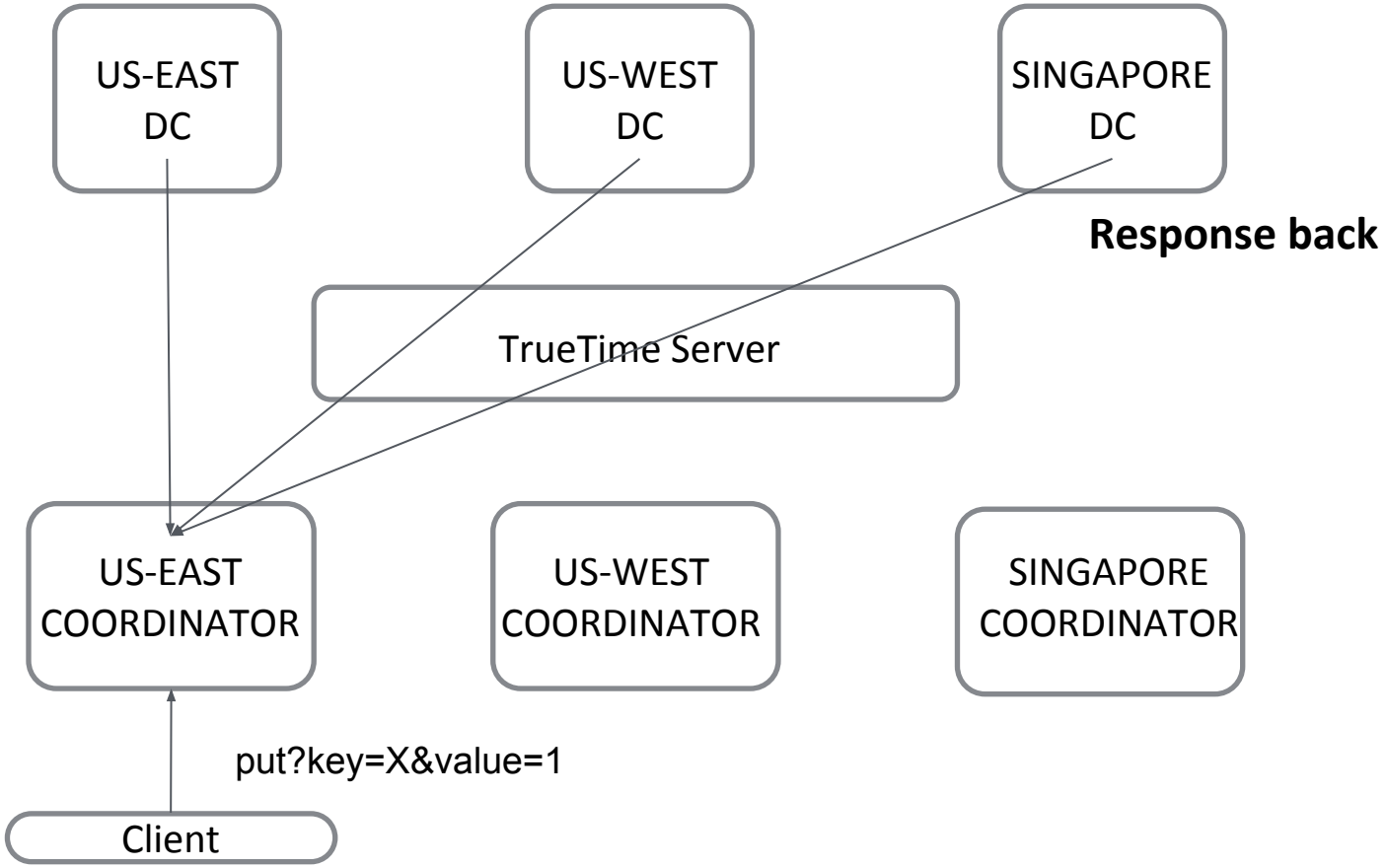
P3.3 Task 2:
Complete KeyValueStore.java (in DCs) and Coordinator.java (in Coordinators)



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: 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

Hints - PRECOMMIT

- In strong consistency, “PRECOMMIT” should be useful to help you lock requests because they are able to communicate with Data centers.
- Locking needs to be performed on Data centers.
- Lock by the key across all the Data centers in strong consistency
- Remember to update both KeyValueStore.java and Coordinator.java in Eventual Consistency

Suggestions

- Read the two primers (PLEASE!)
- 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**
- Don’t modify any class except Coordinator.java and KeyValueStore.java

How to Run Your Program

- Run “./copy_code_to_instances” in client instance to copy your code to servers on each of the Data centers instance, Coordinators instance.
- Run “./start_servers” in the client instance to start the servers on each of the data center instances, coordinator instances and the truetime server instance.
- 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 Data centers or Coordinators.

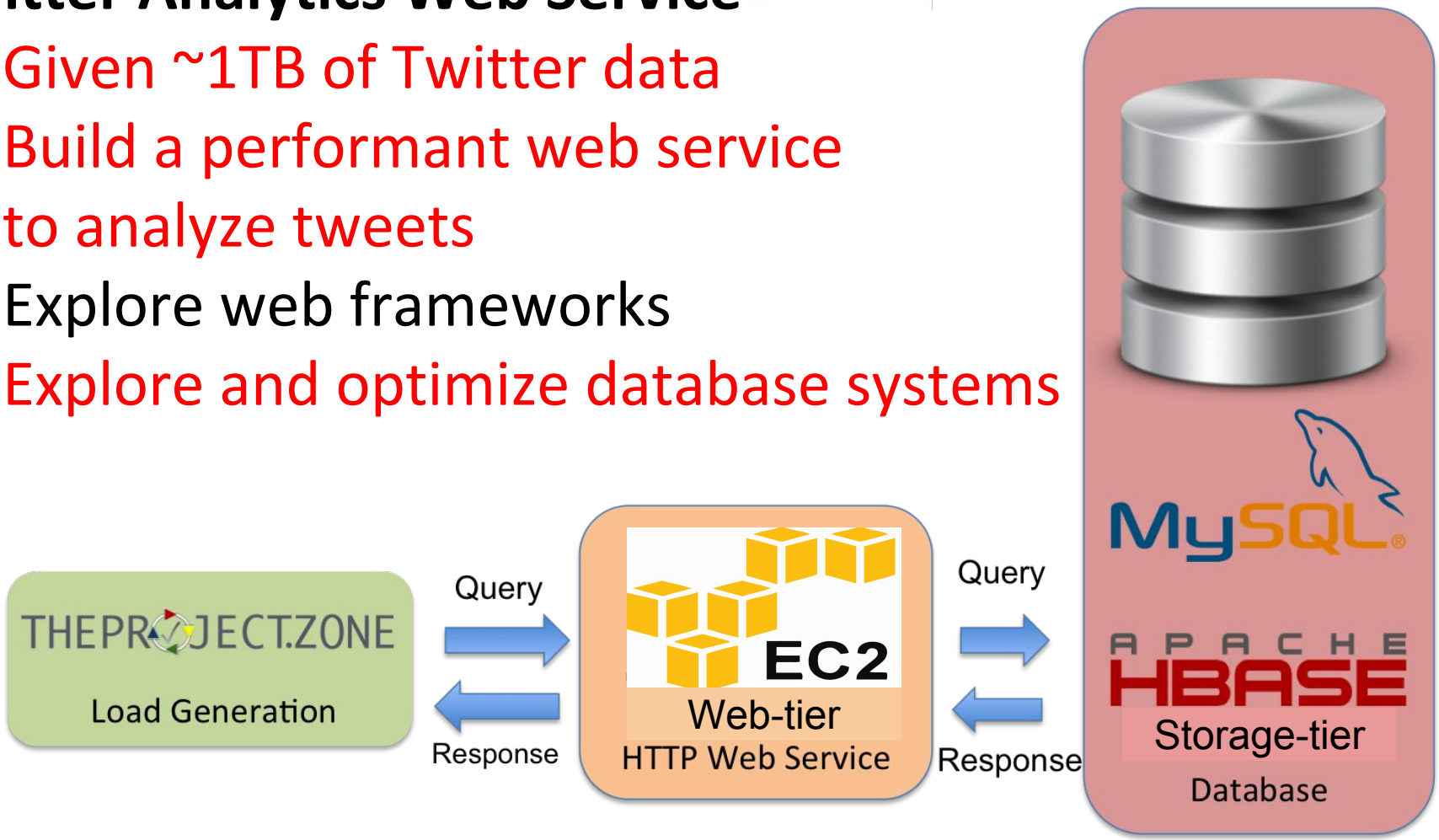
Start early!

TEAM PROJECT
Twitter Data Analytics

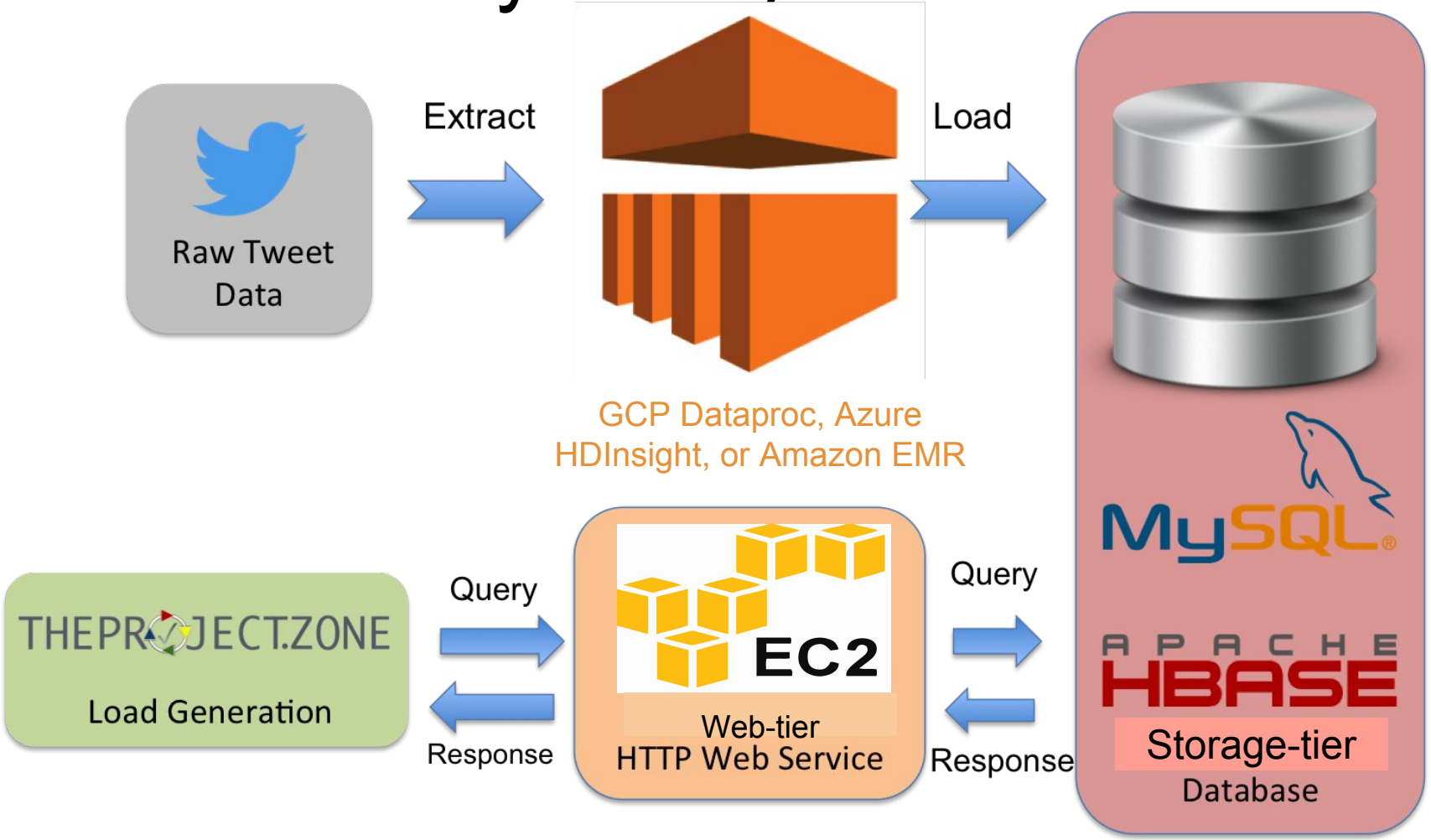
Team Project

Twitter Analytics Web Service

- Given ~1TB of Twitter data
- Build a performant web service to analyze tweets
- Explore web frameworks
- Explore and optimize database systems

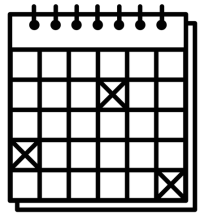


Twitter Analytics System Architecture



- Web server architectures
- Dealing with large scale real world tweet data
- HBase and MySQL optimization

Suggested Tasks for Phase 1



Phase 1 weeks	Tasks	Deadline
Week 1 • **2/25**	• **Team meeting** • **Writeup** • **Complete Q1 code & achieve correctness** • **Q2 Schema, think about ETL**	• **Q1 Checkpoint due on 3/3** • **Checkpoint Report due on 3/3**
Week 2 • 3/4	• **Q1 target reached** • **Q2 ETL & Initial schema design completed**	• **Q1 final target due on 3/10**
Week 3 • Spring Break	• Take a break or make progress (up to your team)	
Week 4 • 3/18	• Achieve correctness for both Q2 MySQL, Q2 HBase & basic throughput	• Q2 MySQL Checkpoint due on 3/24 • Q2 HBase Checkpoint due on 3/24
Week 5 • 3/25	• Optimizations to achieve target throughputs for Q2 MySQL and Q2 HBase	• Q2 MySQL final target due on 3/31 • Q2 HBase final target due on 3/31

Reminders on penalties

- M family instances **only**; must be ≤ **large** type

  ✓ **m5.large, m5.medium, m4.large** ✗ **m5.2xlarge, m3.medium, t2.micro**

- Only General Purpose (gp2) SSDs are allowed for storage
  - **m5d** (which uses NVMe storage) is forbidden
- Other types are allowed (e.g., t2.micro) **but only for testing**
  - Using these for any submissions = 100% penalty
- **$0.85/hour** applies to every submission, not just the livetest
- AWS endpoints only (EC2/ELB).

Budget

- AWS budget of $45 for Phase 1
- Your web service should cost at most $0.85 per hour
  - Including: EC2 cost, EBS cost, ELB cost
  - Excluding: data transfer, EMR
- Even if you use spot instances, we will calculate your cost using the on-demand instance price
- Q2 target RPS: 12000 for both MySQL and HBase

Query 2: Tips

1. Libraries can be bottlenecks
2. MySQL connection configuration
3. MySQL warmup
4. Response formatting: be careful with **\n \t**
5. Understand the three types of scores completely.

Query 2: More Tips

1. Consider doing ETL on GCP/Azure to save AWS budget
2. Be careful about encoding 😁 (use utf8mb4 in MySQL)
3. Pre-compute as much as possible
4. ETL can be expensive, so read the write-up carefully

Piazza FAQ

1. Search before asking a question

2. Post public questions when possible

https://piazza.com/class/jqsp37y8m572vm?cid=1336

This Week's Deadlines

- Quiz 8:

  Due: Friday, March 22nd, 2019 11:59PM ET

- Complete OPE task scheduling

  Due: This week

- Project 3.3: Consistency

  Due: Sunday, March 24th, 2019 11:59PM ET

- Team Project Phase 1 Q2M and Q2H Correctness

  Due: Sunday, March 24th, 2019 11:59PM ET


THANK YOU