15-319 / 15-619
Cloud Computing

Recitation 9

Mar 17, 2020

Overview

- **Last week's reflection**
  - Project 3.2
  - OLI Unit 4 - Module 14
  - Quiz 7
  - Online Programming Exercise for Multi-Threading
- **This week's schedule**
  - Project 3.3
  - OLI - Modules 15, 16 & 17
  - Quiz 8 due on **Friday, Mar 20th**
- **Team Project, Twitter Analytics**
  - Phase 1 Q2 Checkpoint, 3/22.

Before Spring Break

- **OLI : Module 14**
  - **Quiz 7**
- **Project 3.2**
  - Social Networking Timeline with Heterogeneous Backends
    - MySQL
    - Neo4j
    - MongoDB
    - Choosing Databases, Storage Types & Tail Latency
- **Team Project**
  - Query 1 Final
- **Multi-Threading OPE Exercise on Cloud9**

This Week

- **OLI : Modules 15, 16 & 17**
  - **Quiz 8** - Friday, Mar 20th
- **Project 3.3** - Sunday, Mar 22nd
  - *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** - Sunday, Mar 22nd
  - Query 2 Checkpoint
- **Spark OPE** - Scheduling

Conceptual Topics - OLI Content

OLI UNIT 4: Cloud Storage

- **Module 15: Case Studies: Distributed File Systems**
  - HDFS
  - Ceph
- **Module 16: Case Studies: NoSQL Databases**
- **Module 17: Case Studies: Cloud Object Storage**
- **Quiz 8**
  - Due on Friday, Mar 20th
    - Remember to click submit
      - Within 2 hours, and
      - Before the deadline!

Individual Projects

- Done
  - P3.1: Files v/s Databases
  - 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 predicts massive data increases:

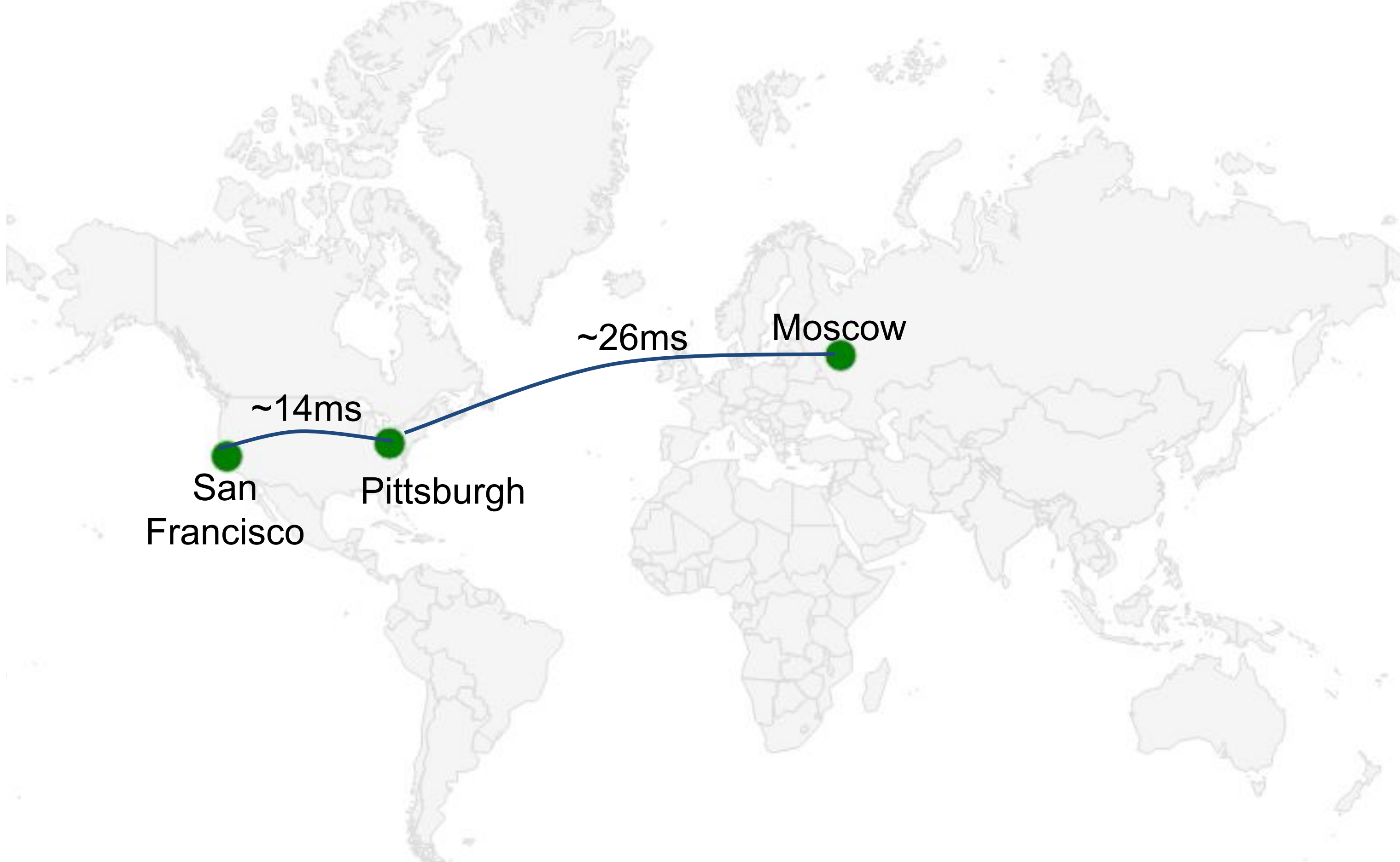
- **From:** 33 zettabytes in 2018

- **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 real-time.

Users are Global

- Information has physical limitations on speed of travel (Speed of light)
- Inherent latencies
    - Especially for real-time information, speed is everything!



Typical End-To-End Latency

1. A client sends a request to our server

   Message takes time to physically reach server

   (Network latency)

2. Server receives request and responds

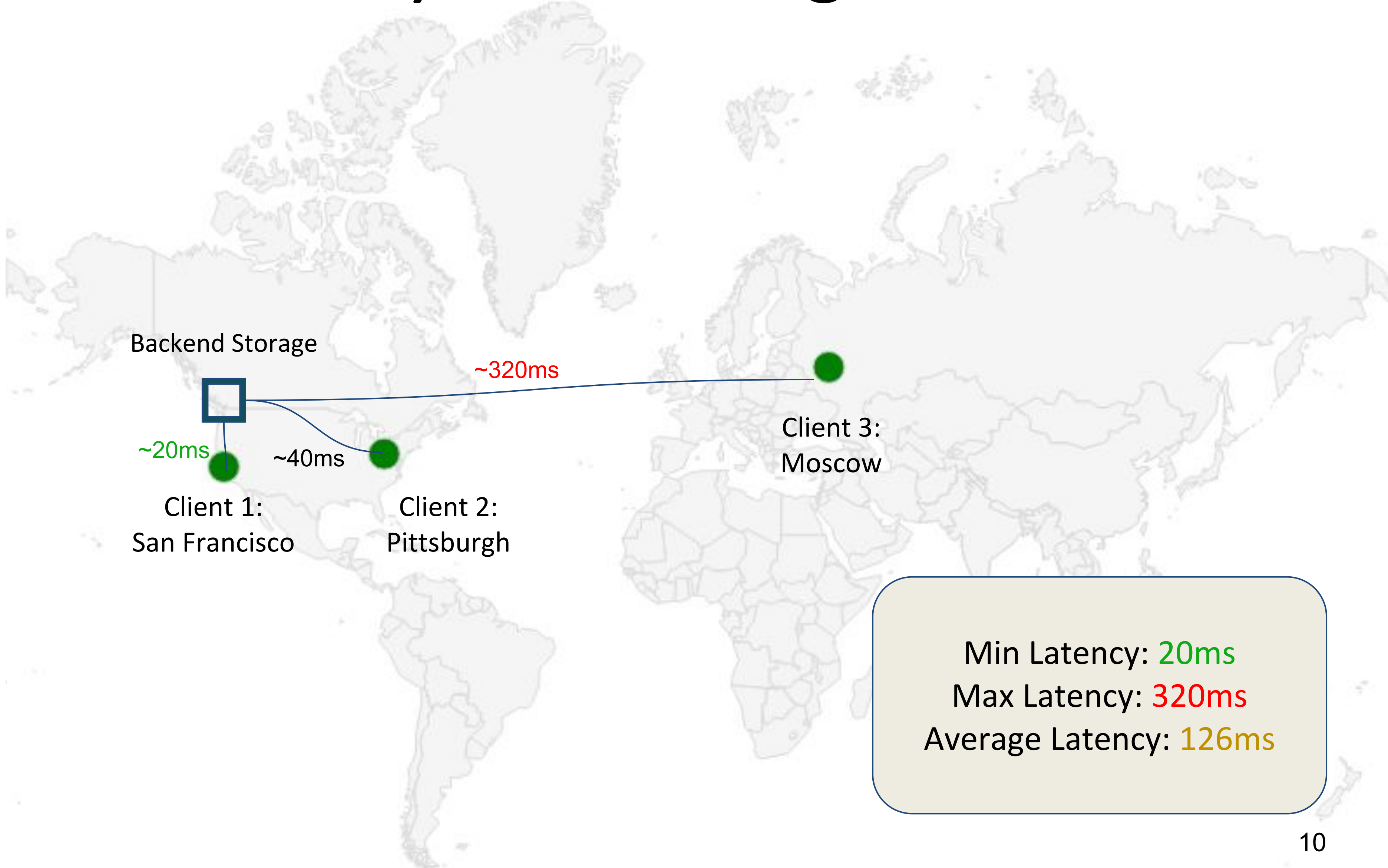
   Server has to read incoming packets and responds

   (IO or Disk latency)

   Message takes time to physically reach client

   (Network latency)

Latency with a Single Backend



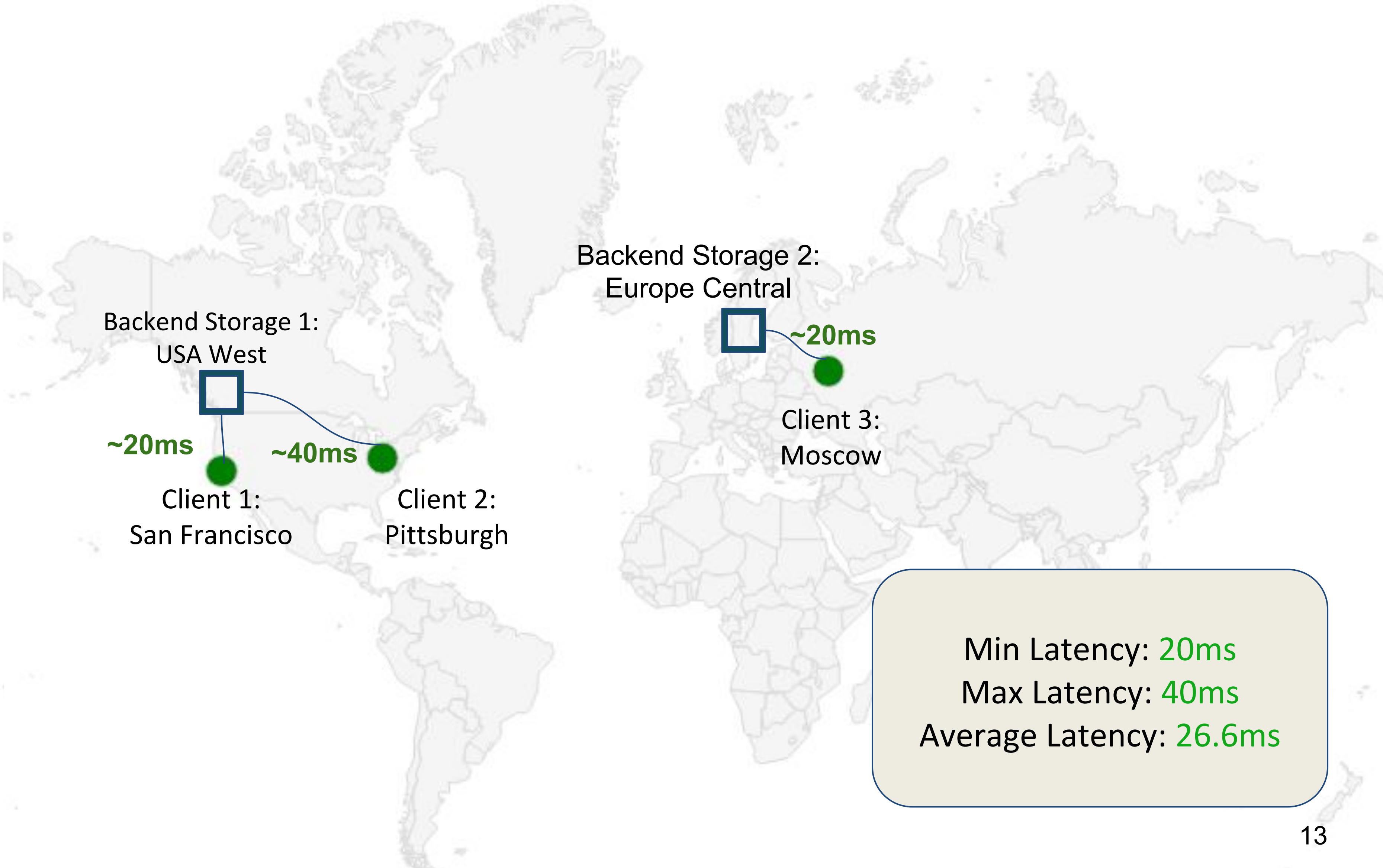
Min Latency: 20ms
Max Latency: 320ms
Average Latency: 126ms

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



Option 2: Proximity Replication

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

Replication

- By adding replicas, we can prevent latency from being too large of an issue
  - Each added datacenter decreases the average latency, as long as they are strategically placed

- But, we need to ensure that data is the same across replicas

- Additionally, replicas increase cost linearly

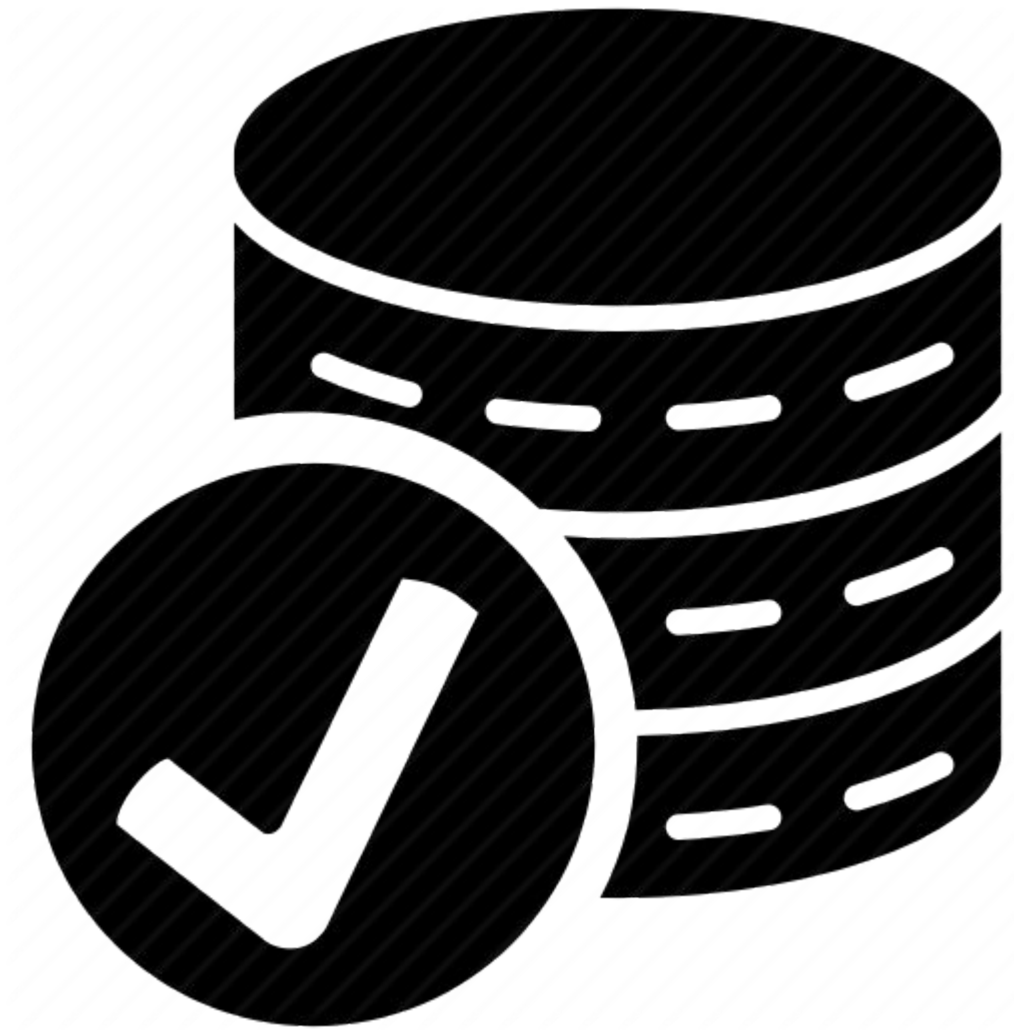
Replication is not infinite

Cost and data consistency are the biggest issues, and place scalability limitations

Cost as 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!

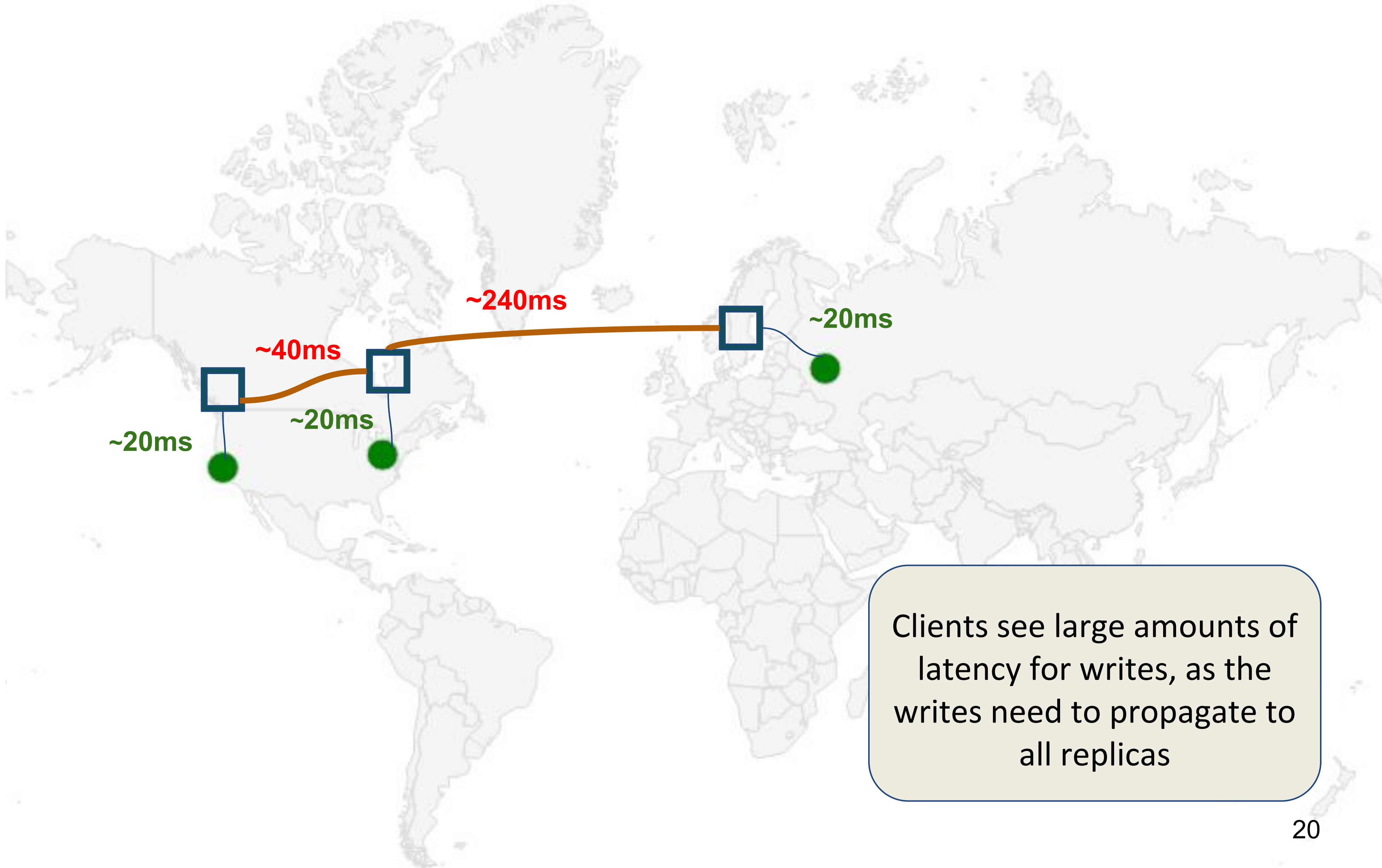
Data Consistency



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

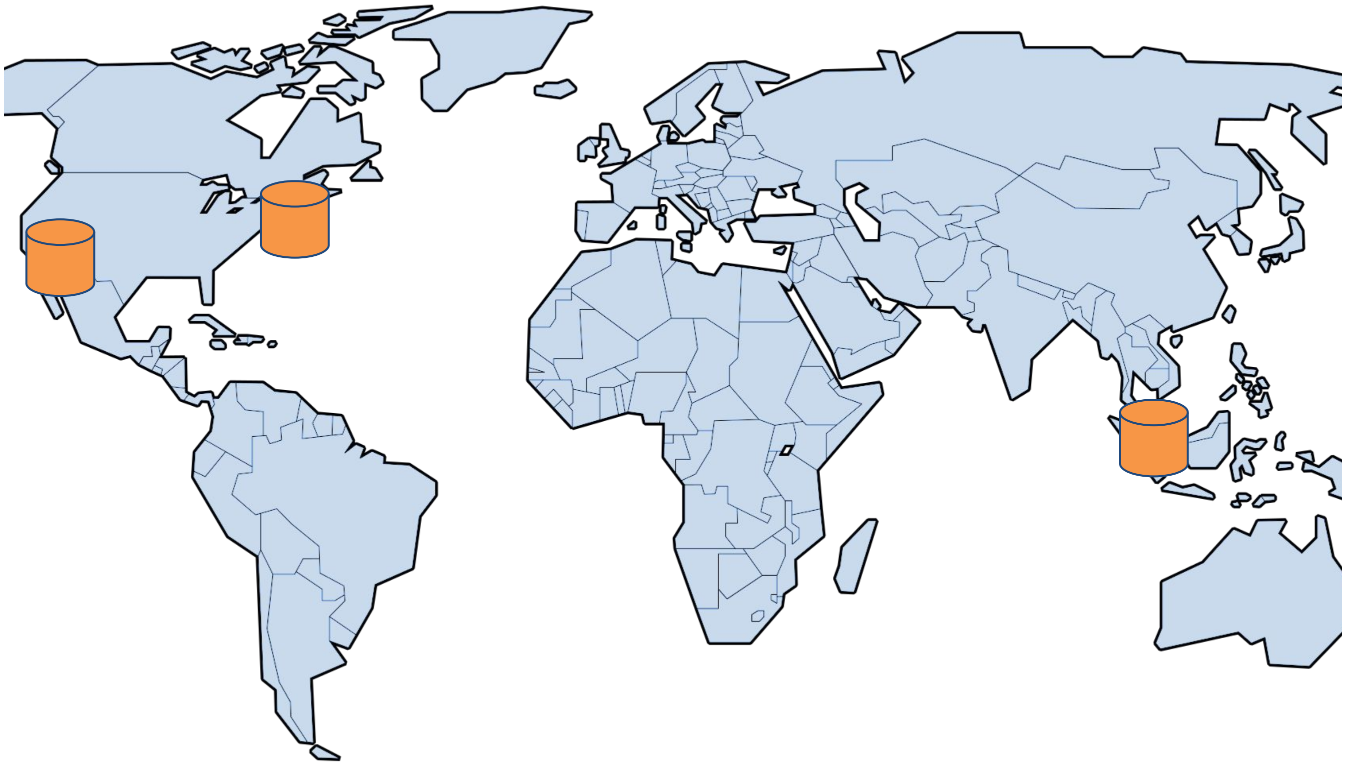
- **Advantages**
  - Low latency for reads
  - Reduce the workload of a single backend server
  - Handle failures of nodes by rerouting to alternative backup replica
- **Disadvantages**
  - Requires more storage capacity and cost
  - Updates are significantly slower
  - Changes must reflect on all datastores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

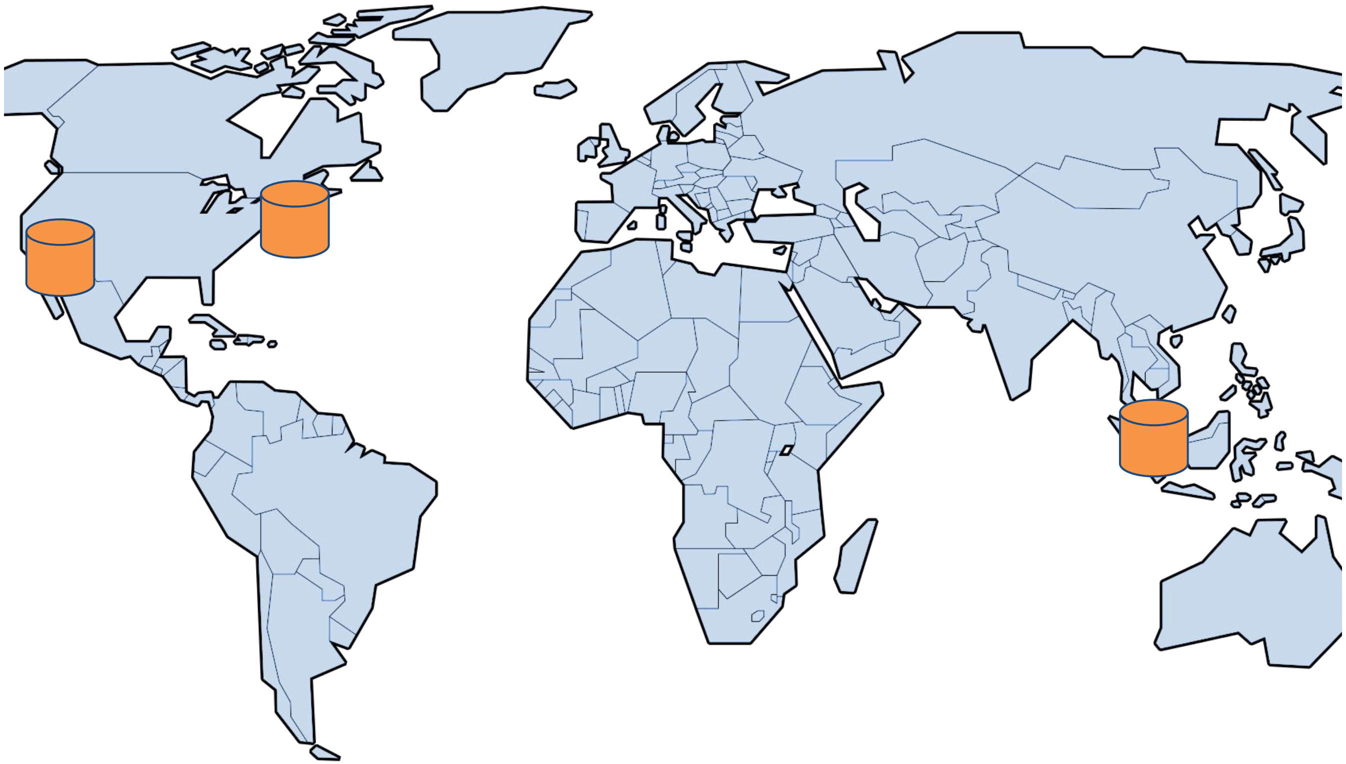
Data Consistency Example: Consider a Bank



Account	Balance
xxxxx-4437	$100

Bad Example

Allow concurrent writes



Account	Balance
xxxxx-4437	$100

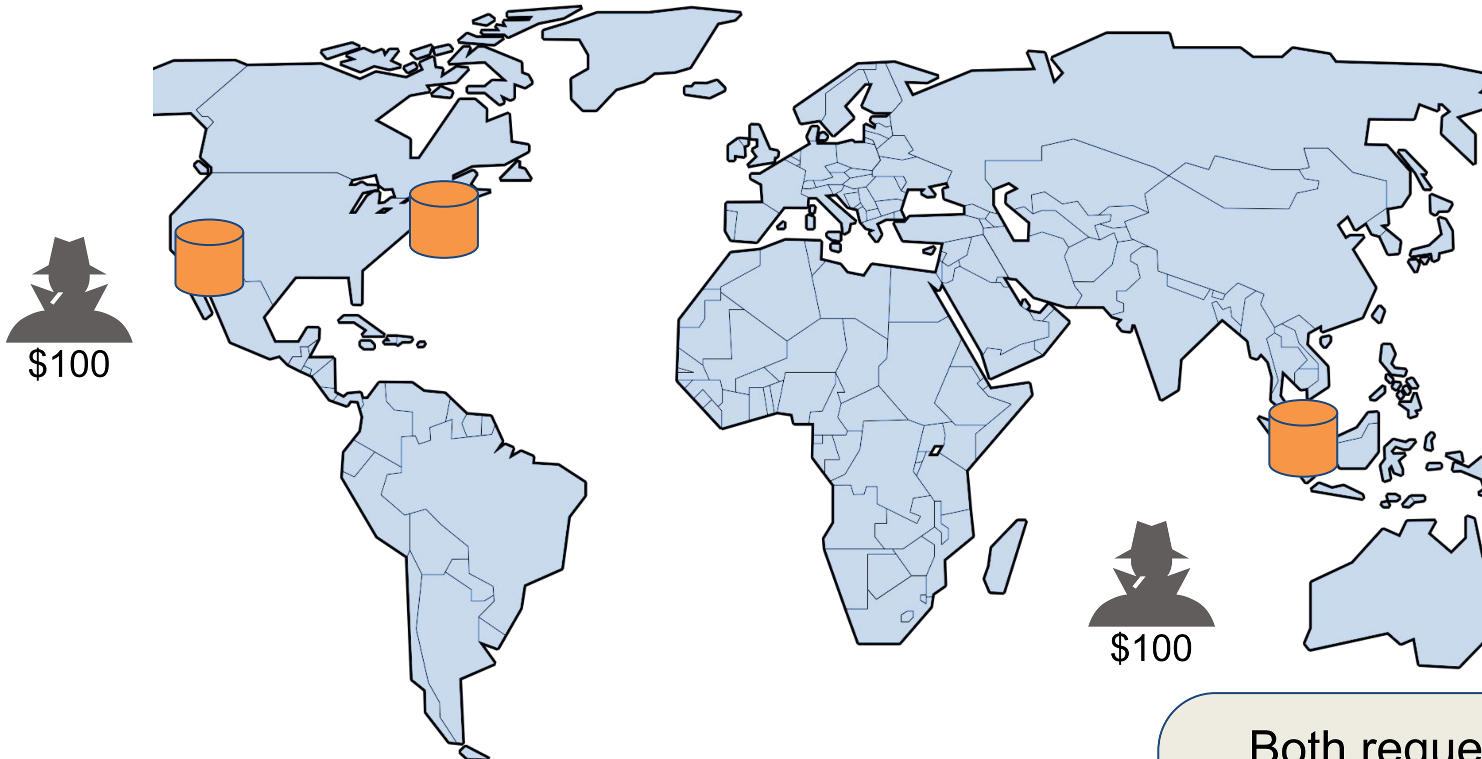
Bad Example

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

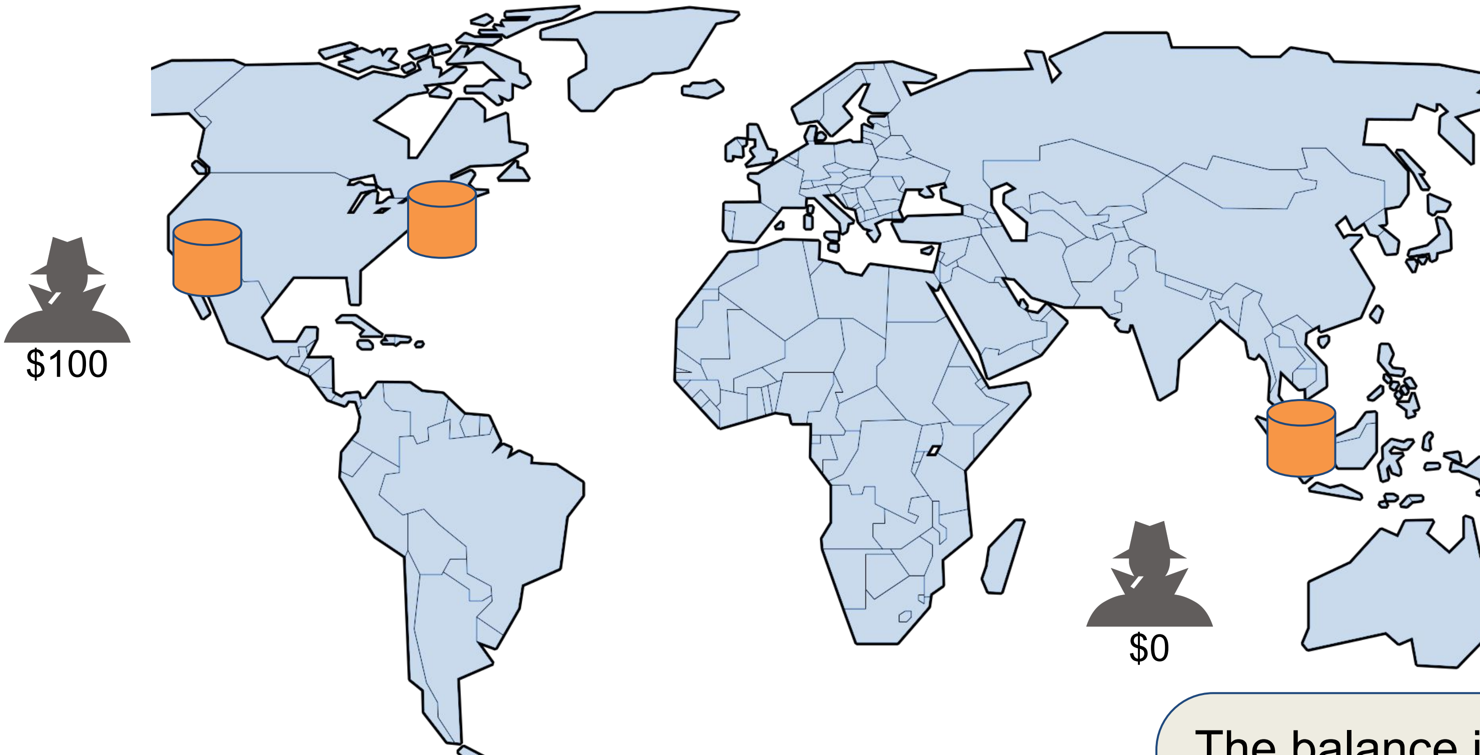
Account	Balance
xxxxx-4437	**$100**

Good Example
Global Locking

Account	Balance
xxxxx-4437	**$100**

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

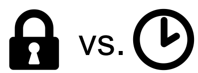
Good Example
Global Locking



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 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

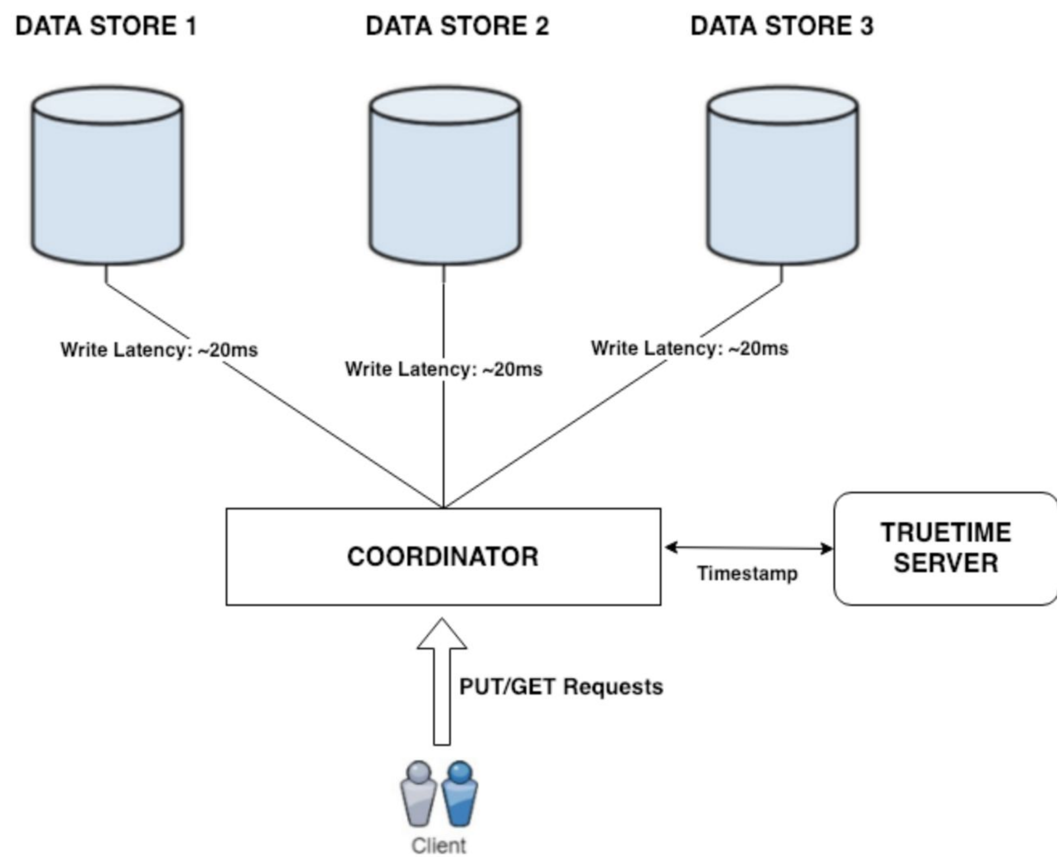
P3.3 Task 1: Strong Consistency

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



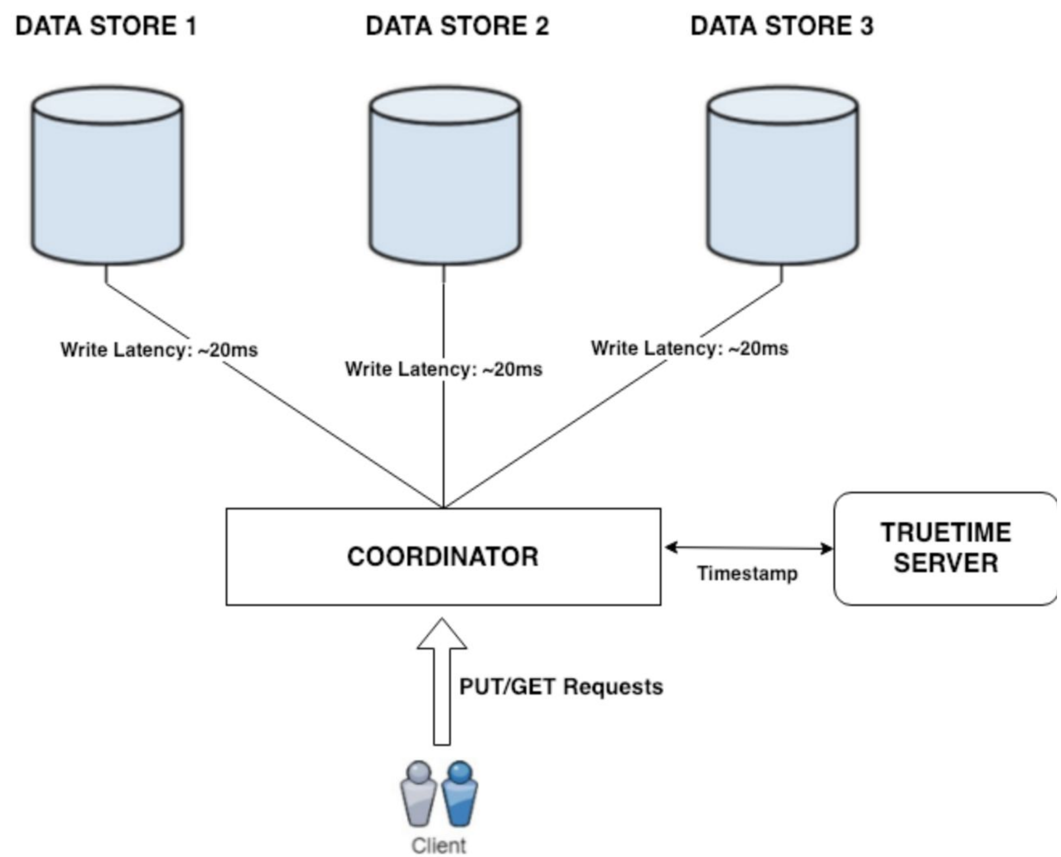
P3.3 Task 1: Strong Consistency

Single PUT request for key ‘X’

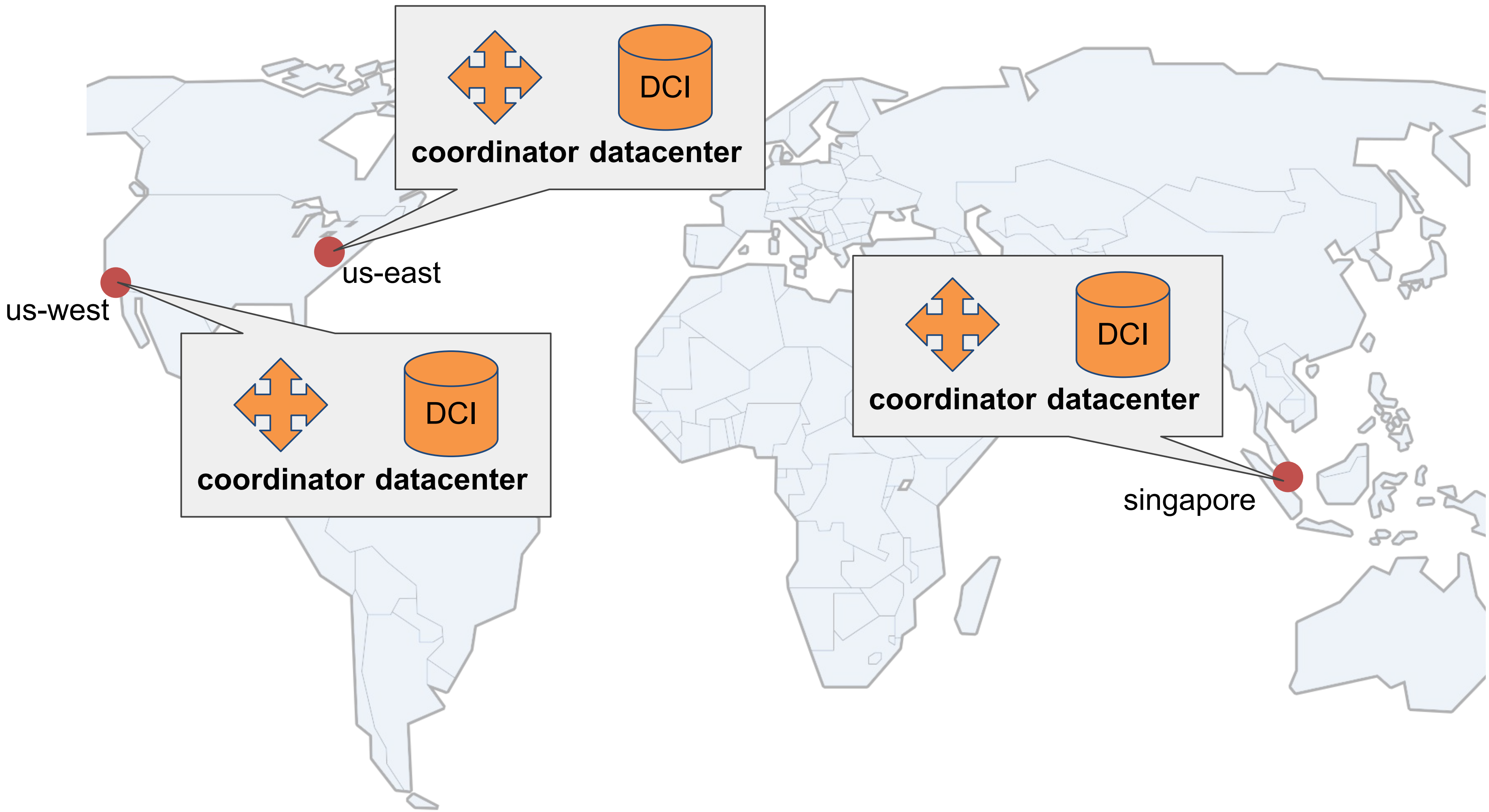
- Block all GETs for key ‘X’ until all datastores are updated
- GET requests for a different key ‘Y’ should **not** be blocked

Multiple PUT requests for ‘X’

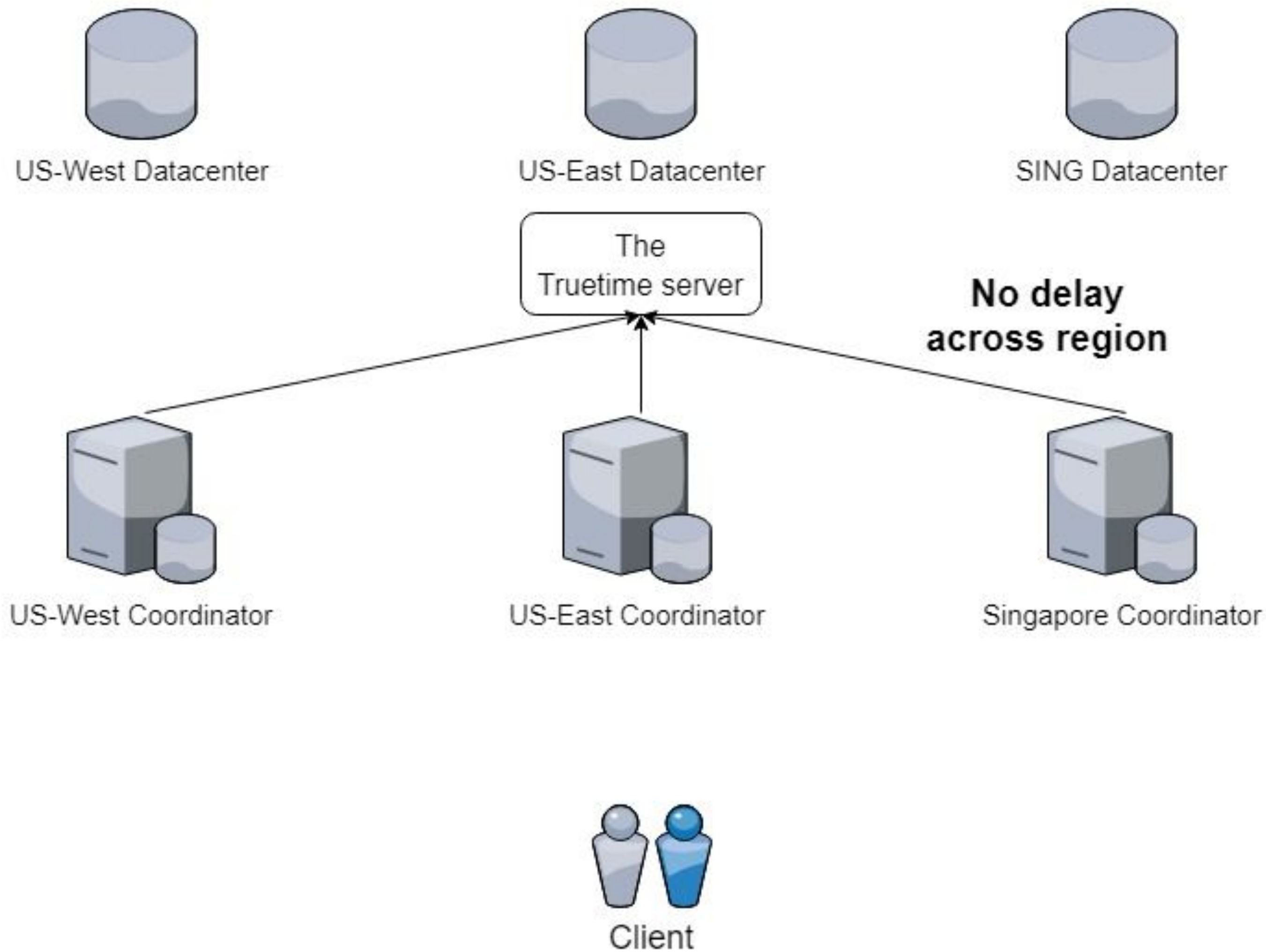
- Resolved in order of their timestamp received from the Truetime Server.
- GET requests must return the most recent value to the request timestamp



P3.3 Task 2: Global Coordinators and Data Stores



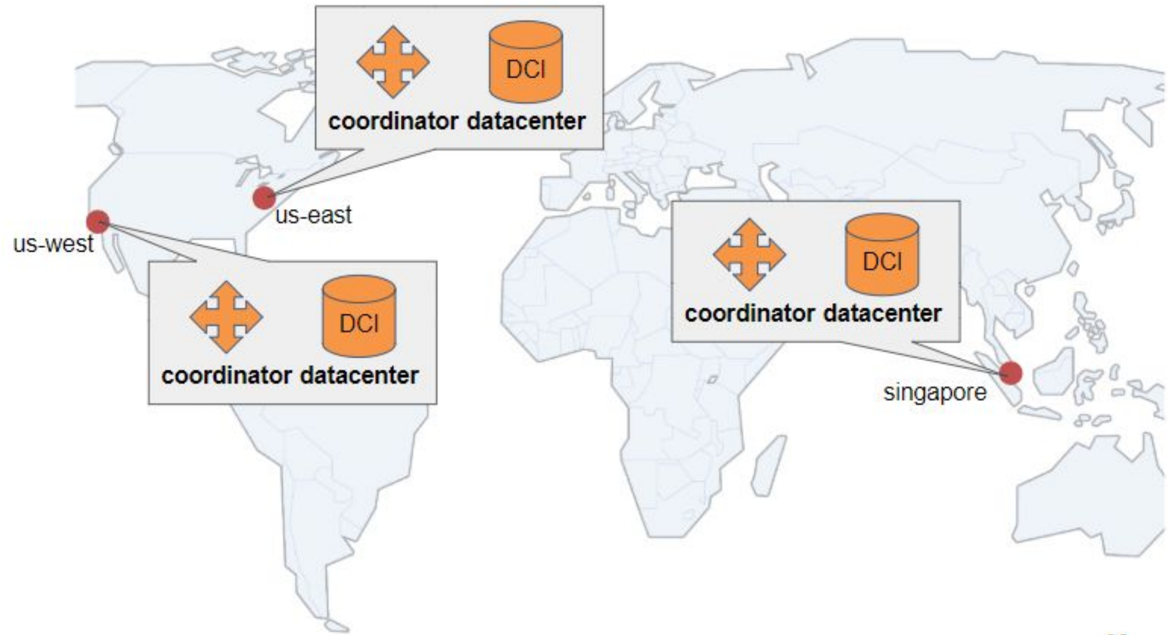
P3.3 Task 2: Architecture



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

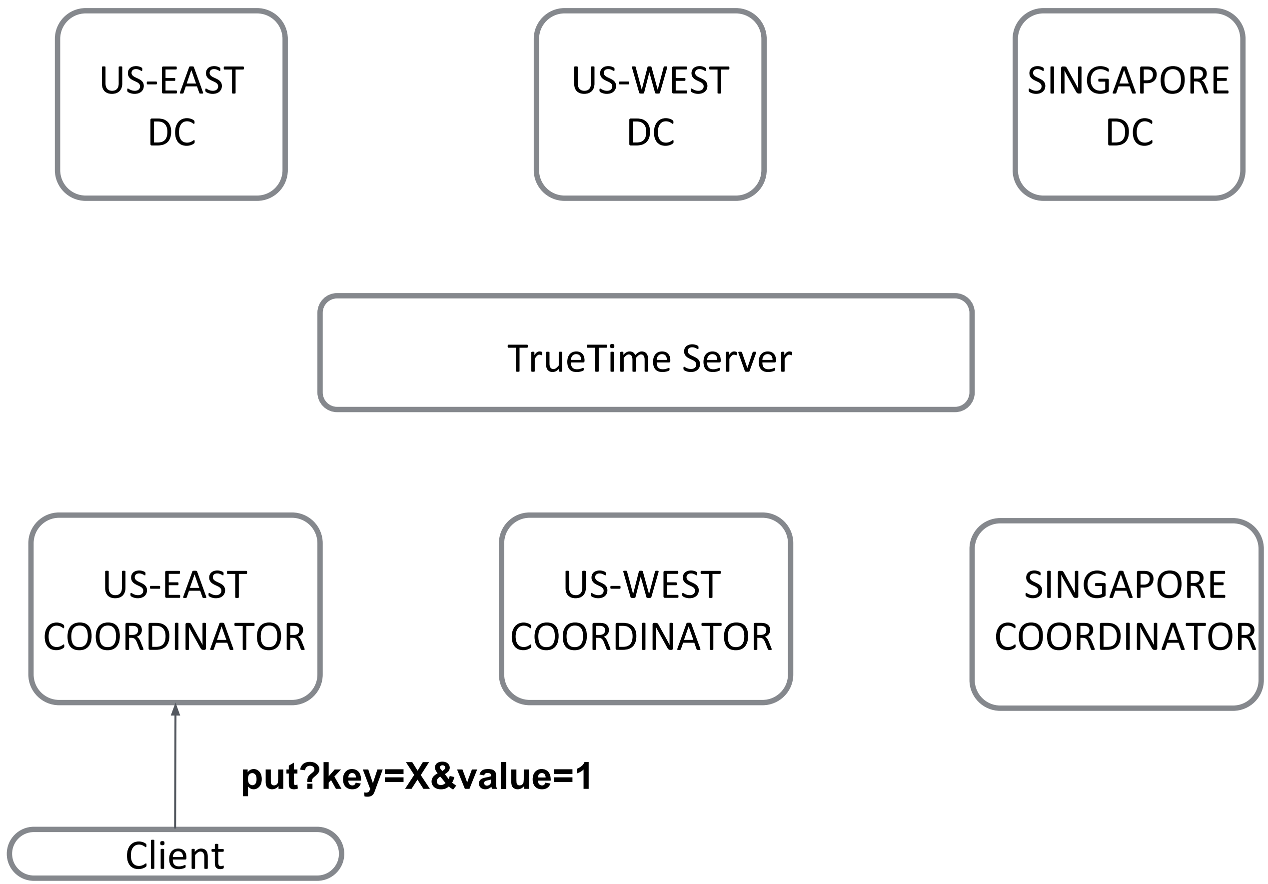
US East (N. Virginia)
US East (Ohio)
US West (N. California)
US West (Oregon)
Asia Pacific (Mumbai)
Asia Pacific (Seoul)
Asia Pacific (Singapore)
Asia Pacific (Sydney)
Asia Pacific (Tokyo)

PRECOMMIT

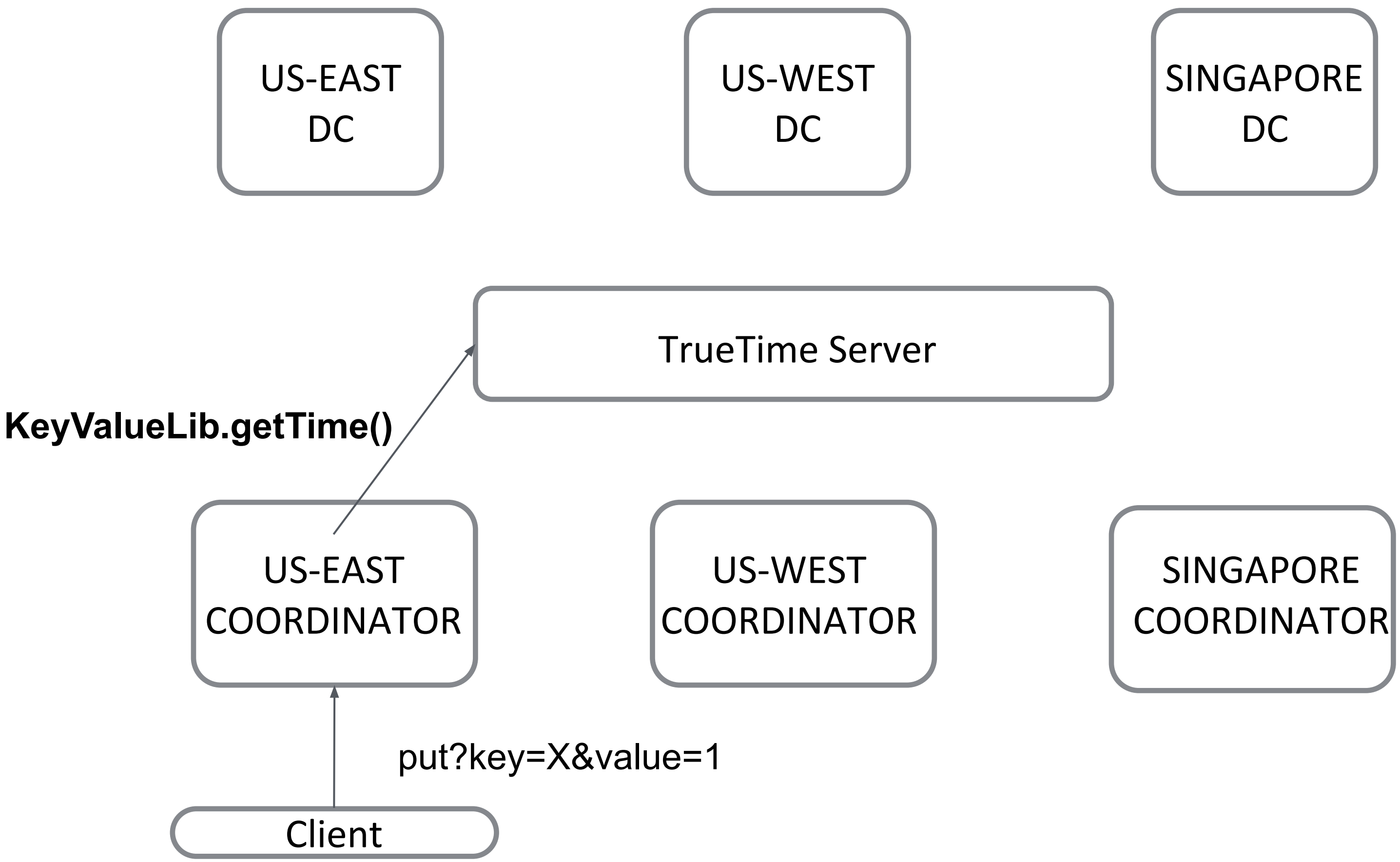
Contacts the Data Center of a given region and notifies it that a PUT request is being serviced for the specified key with the corresponding timestamp.

P3.3 Task 2:

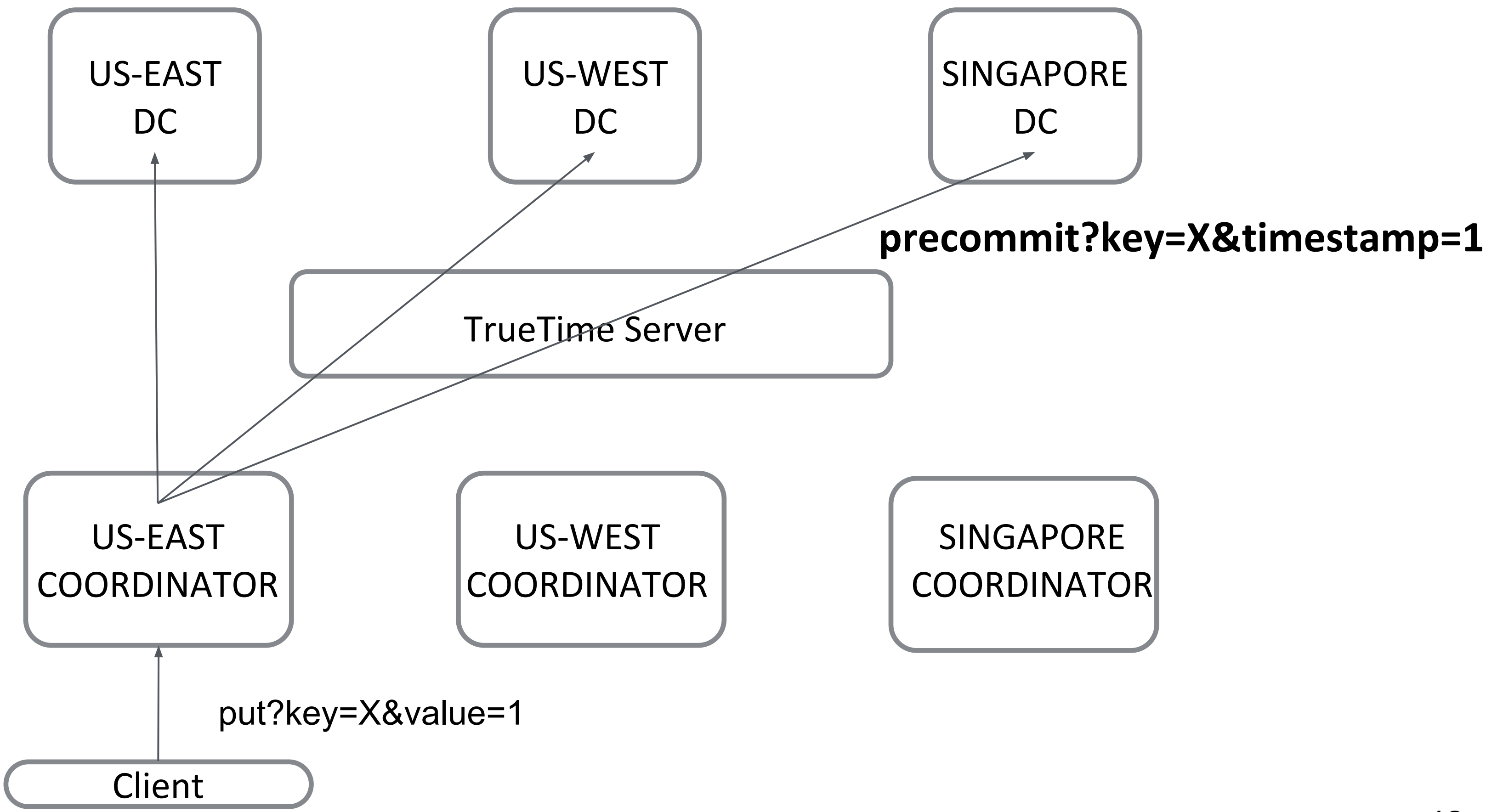
Complete KeyValueStore.java and Coordinator.java



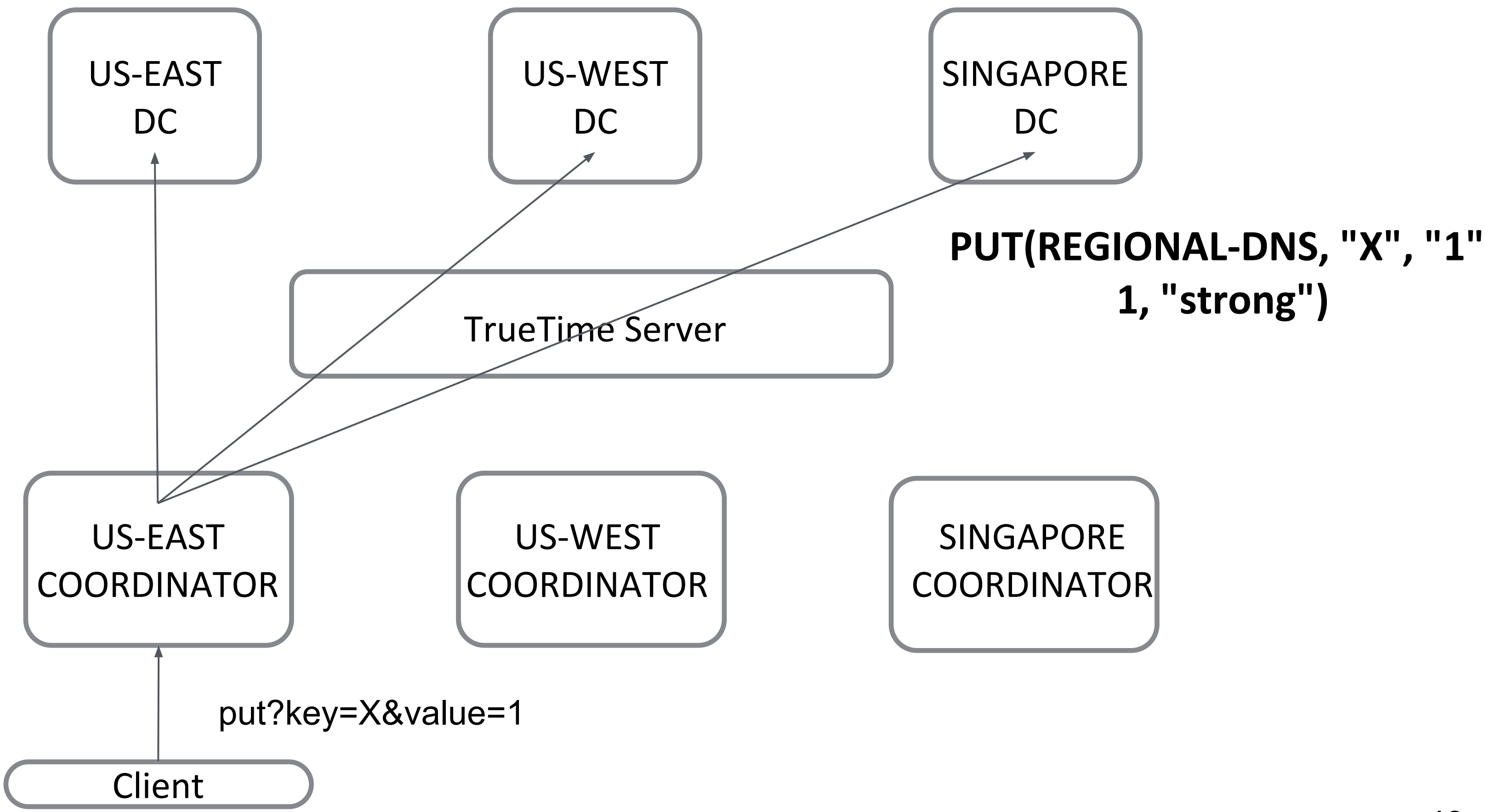
P3.3 Task 2:
Complete KeyValueStore.java (in DCs) and Coordinator.java (in Coordinators)



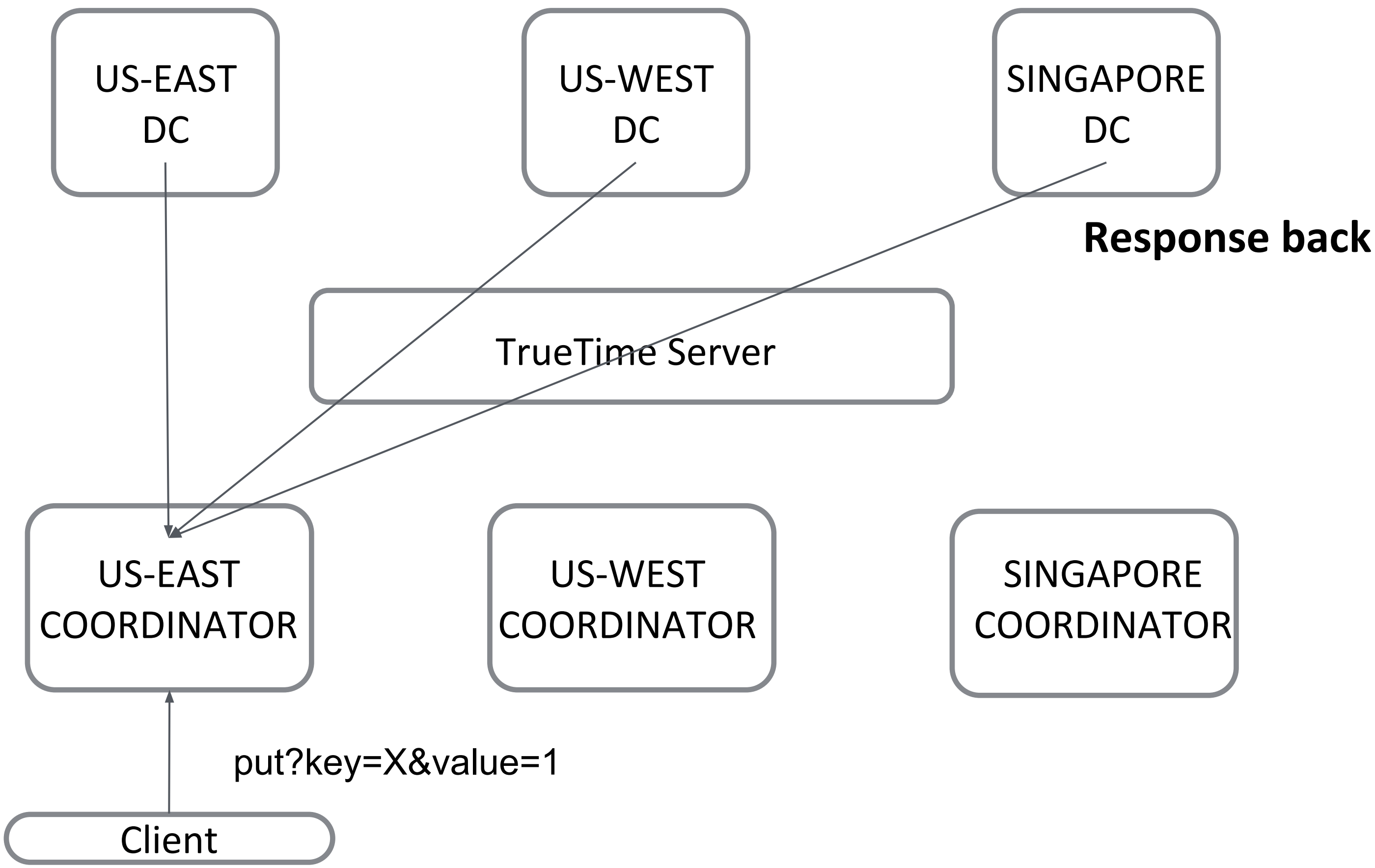
P3.3 Task 2:
Complete KeyValueStore.java (in DCs) and Coordinator.java (in Coordinators)



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)



Hints - PRECOMMIT

- In strong consistency, “PRECOMMIT” should be useful to help you lock requests because they are able to communicate with Data Center instances.
- Locking needs to be performed on Data Center instances.
- Lock by key across all the Data Center instances in strong consistency.

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**
- 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 "./submitter" to test your implementations.
- If you want to test one simple PUT/GET request, you could directly send the request to Data centers or Coordinators.

Start early!

Piazza FAQ

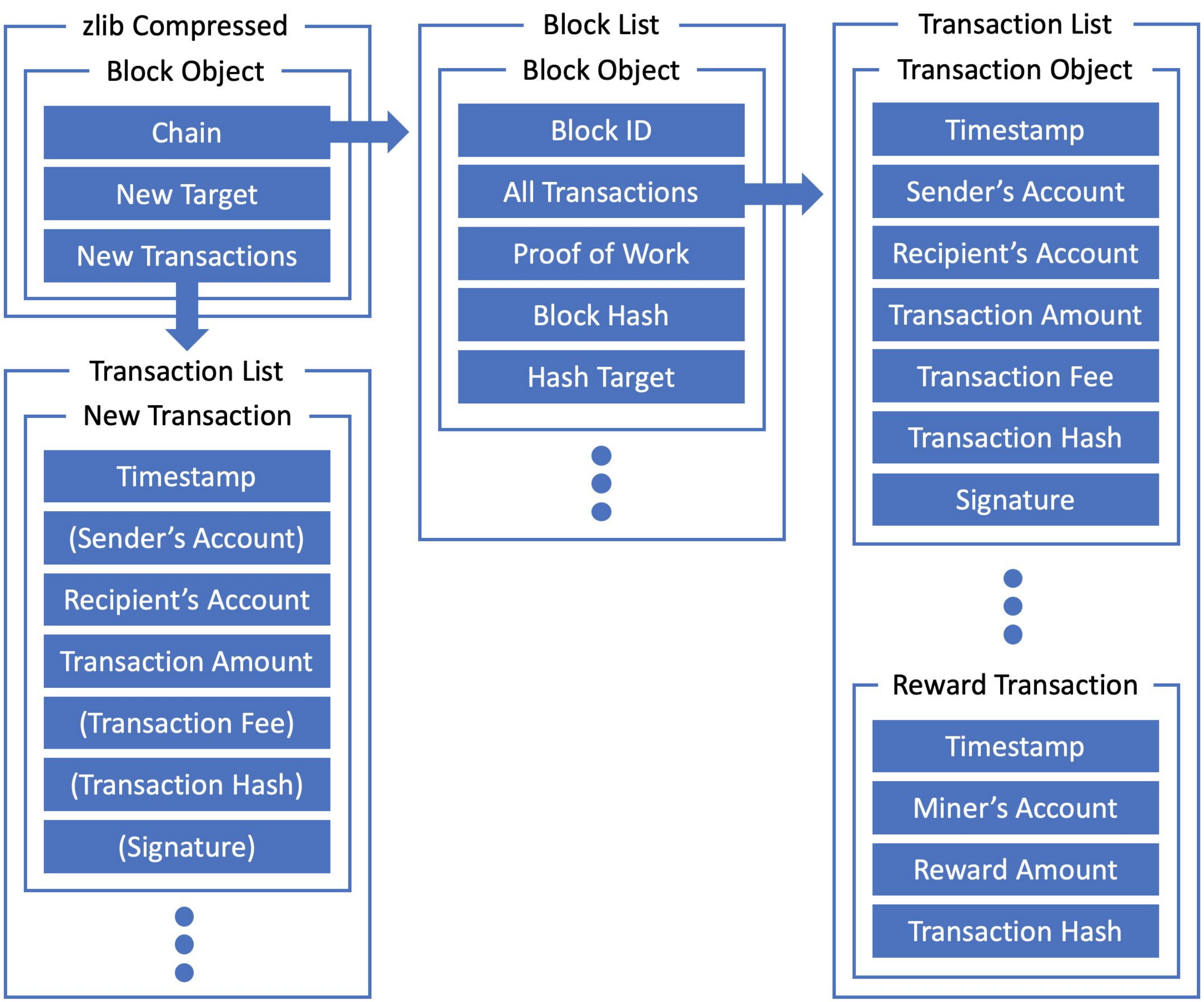
1. Search before asking a question

2. Post public questions when possible

https://piazza.com/class/k562fiaob2hlh

TEAM PROJECT
Twitter Data Analytics

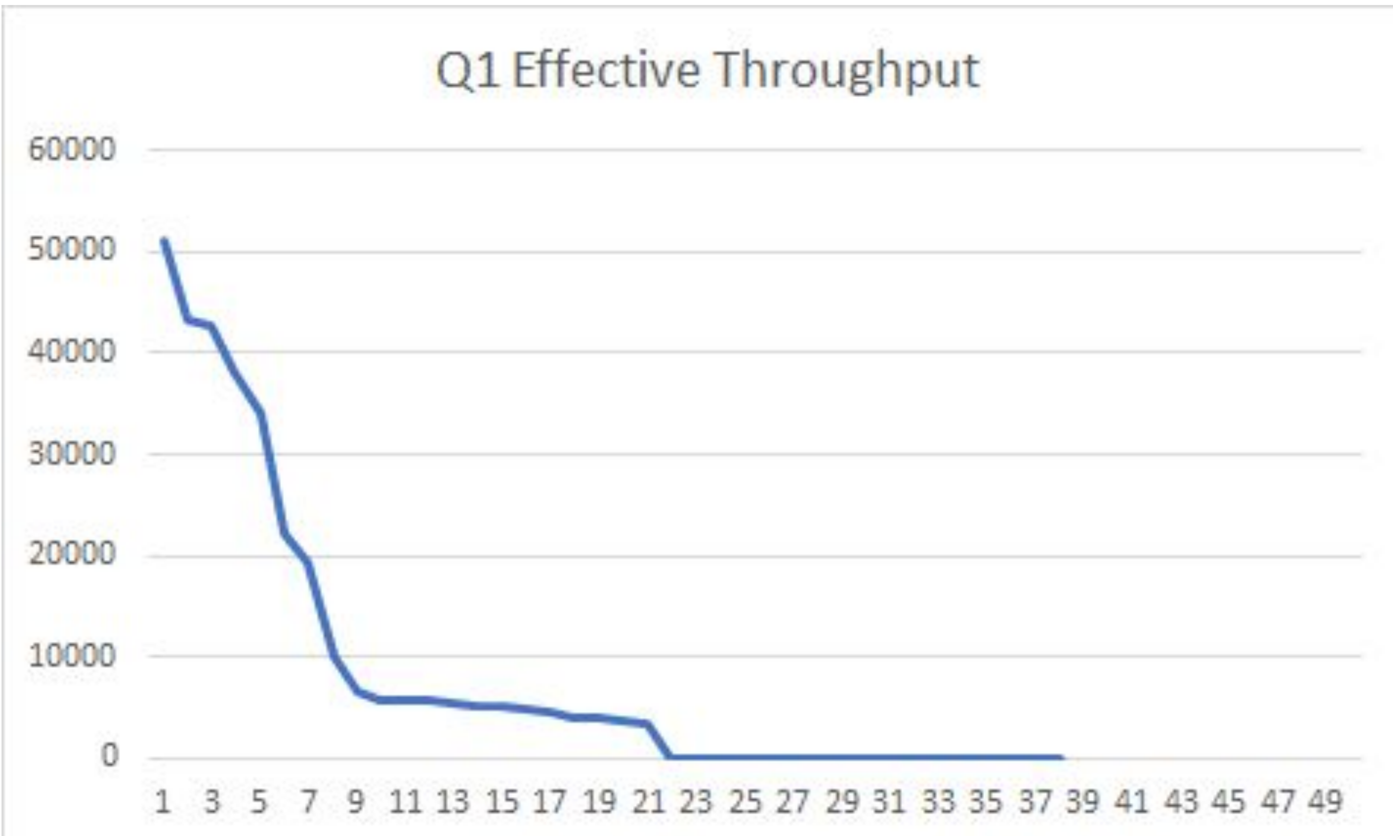
Query 1 Recap




```
{
  "chain": [
    {
      "all_tx": [{
        "recv": 895456882897,
        "amt": 500000000,
        "time": "1582520400000000000",
        "hash": "4b277860"
      }],
      "pow": "0",
      "id": 0,
      "hash": "07c98747",
      "target": "1"
    },
    {
      "all_tx": [
        {
          "sig": 1523500375459,
          "recv": 831361201829,
          "fee": 2408,
          "amt": 126848946,
          "time": "1582520454597521976",
          "send": 895456882897,
          "hash": "c0473abd"
        },
        {
          "recv": 621452032379,
          "amt": 500000000,
          "time": "1582521002184738591",
          "hash": "ab56f1d8"
        }
      ],
      "pow": "202",
      "id": 1,
      "hash": "0055fd15",
      "target": "01"
    },
    {
      "all_tx": [
        {
          "sig": 829022340937,
          "recv": 905790126919,
          "fee": 78125,
          "amt": 4876921,
          "time": "1582521009246242025",
          "send": 831361201829,
          "hash": "46b61f8e"
        },
        {
          "sig": 295281186908,
          "recv": 1097844002039,
          "fee": 0,
          "amt": 83725981,
          "time": "1582521016852310220",
          "send": 895456882897,
          "hash": "b6c1b10f"
        },
        {
          "recv": 905790126919,
          "amt": 250000000,
          "time": "1582521603026667063",
          "hash": "b0750555"
        }
      ],
      "pow": "12",
      "id": 2,
      "hash": "00288a38",
      "target": "0a"
    }
  ],
  "new_target": "007",
  "new_tx": [
    {
      "sig": 160392705122,
      "recv": 658672873303,
      "fee": 3536,
      "amt": 34263741,
      "time": "1582521636327155516",
      "send": 831361201829,
      "hash": "1fb48c71"
    },
    {
      "recv": 895456882897,
      "amt": 34263741,
      "time": "1582521645744862608"
    }
  ]
}
```

Team Project - Q1 CKPT1

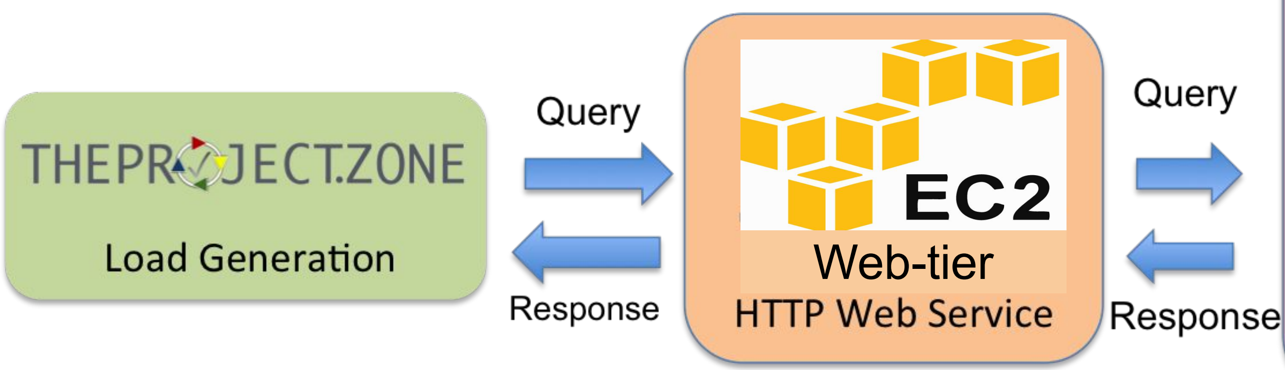
- 38 teams attempted a Query 1 submission.
- 20 teams made successful 10-minute submission.
- 5 teams reached 32,000 RPS.



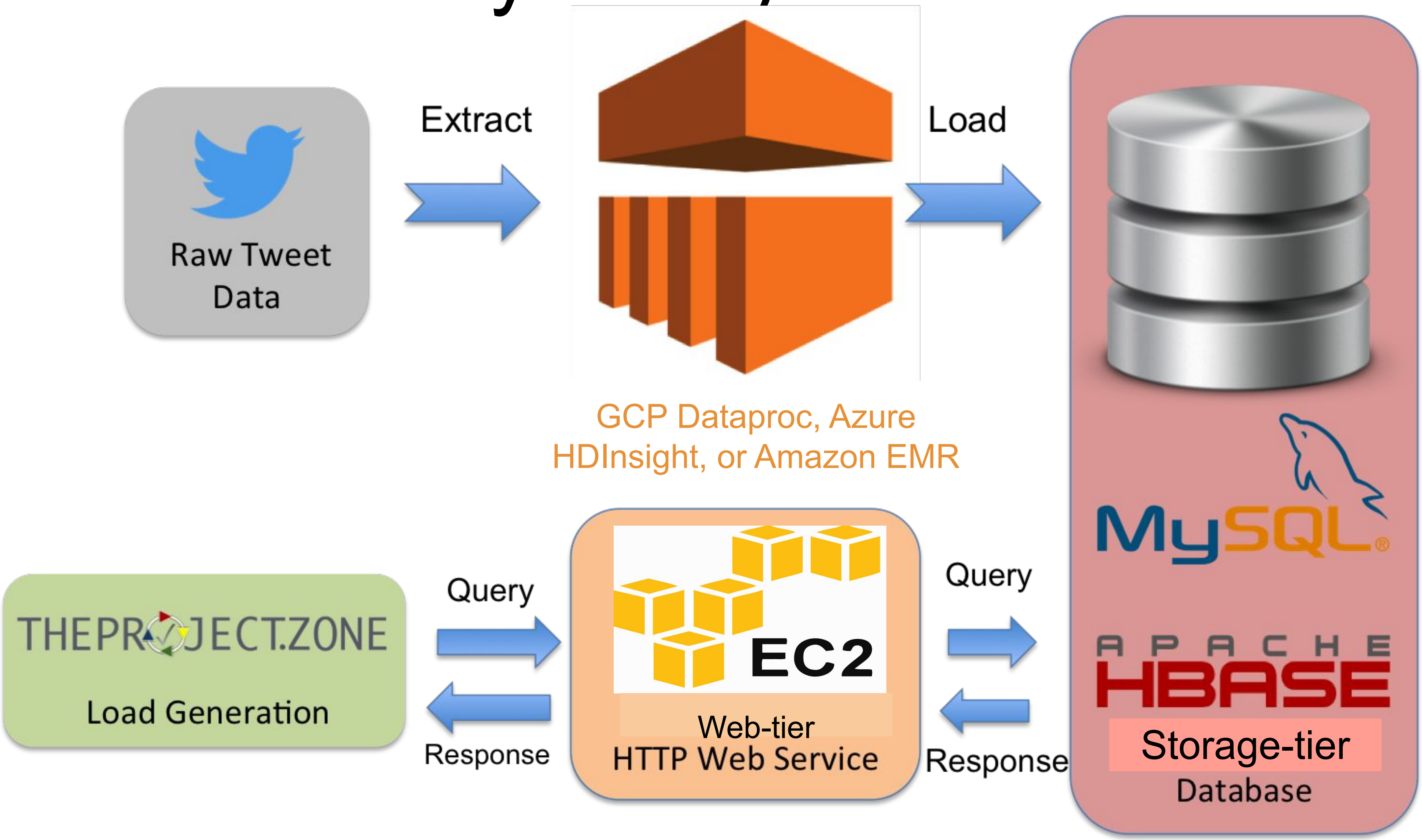
Team Project

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

Reminder 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) are 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
- AWS endpoints only (EC2/ELB).

Phase 1: Budget

- AWS budget of $55 for Phase 1
- Your web service should not cost more than **$0.85 per hour** this includes (see write-up for details):
  - EC2 cost
  - **EBS cost**
  - ELB cost
  - We will not consider the cost of data transfer and EMR
- Even if you use spot instances, we will calculate your cost using the on-demand instance price
- Q2 target throughput: 10000 RPS for both MySQL and HBase

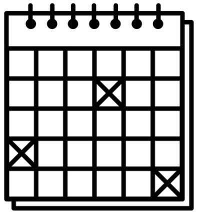
Query 2: Tips

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

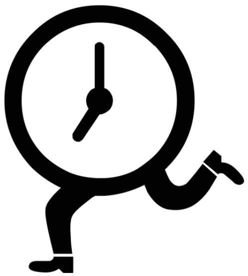
1. Consider doing ETL on GCP/Azure to save AWS budget
2. Be careful about encoding 😁
    - use utf8mb4 in MySQL
3. Pre-compute as much as possible
4. ETL can be expensive, so read the write-up carefully

Suggested Tasks for Phase 1



Phase 1 weeks	Tasks	Deadline
Week 1 ● **2/23**	● **Team meeting** ● **Writeup** ● **Complete Q1 code & achieve correctness** ● **Q2 Schema, think about ETL**	● Q1 Checkpoint due on 3/1 ● Checkpoint Report due on 3/1
Week 2 ● 3/2	● **Q1 target reached** ● **Q2 ETL & Initial schema design completed**	● Q1 final target due on 3/8
Week 3 ● Spring Break	● Take a break or make progress (up to your team)	
Week 4 ● 3/16	● Achieve correctness for both Q2 MySQL, Q2 HBase & basic throughput	● Q2 MySQL Checkpoint due on 3/22 ● Q2 HBase Checkpoint due on 3/22
Week 5 ● 3/23	● Optimizations to achieve target throughputs for Q2 MySQL and Q2 HBase	● Q2 MySQL final target due on 3/29 ● Q2 HBase final target due on 3/29

This Week's Deadlines



- Quiz 8:
  Due: Friday, March 20, 2020 11:59PM ET
- Complete OPE task scheduling
  Due: This week
- Project 3.3: Consistency
  Due: Sunday, March 22, 2020 11:59PM ET
- Team Project Phase 1 Q2 Checkpoint
  Due: Sunday, March 22, 2020 11:59PM ET


THANK YOU