15-319 / 15-619
Cloud Computing

Recitation 8

Oct 15, 2019

Overview

- **Last week's reflection**
  - Project 3.1 (optional)
  - Project 3.2
  - OLI Unit 3 - Module 13
  - Quiz 6
- **This week's schedule**
  - Project 3.3
  - OLI Unit 4 - Module 14
  - Quiz 7 due on **Thurs, Oct 17th**
- **Team Project, Twitter Analytics**
  - Query 1 Final & Query2 Checkpoint due, Oct 20

Last Week

- **OLI : Module 13**
  - **Quiz 6**
- **Project 3.2**
  - Social Networking Timeline with Heterogeneous Backends
    - MySQL
    - Neo4j
    - MongoDB
    - Choosing Databases
- **Team Project**
  - Query 1 Checkpoint
- **Training OPE Exercise on Cloud9**

This Week

- **OLI : Module 14**
  - **Quiz 7** - Thurs, Oct 17th
- **Project 3.3** - Sunday, Oct 20th
  - *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, Oct 20th
  - Query 1 Final
  - Query 2 Checkpoint
- **Spark OPE** - Scheduling

Conceptual Topics - OLI Content

OLI UNIT 4: Storage

- Module 14: Cloud Storage
- **Quiz 7**
  - **DUE EARLY: Thurs, October 17th**

- OLI Modules 15 -17 (not due this week)
  - Will be helpful for P3.3

Individual Projects

- DONE
  - P3.1 (Optional): 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 predicts massive data increases:

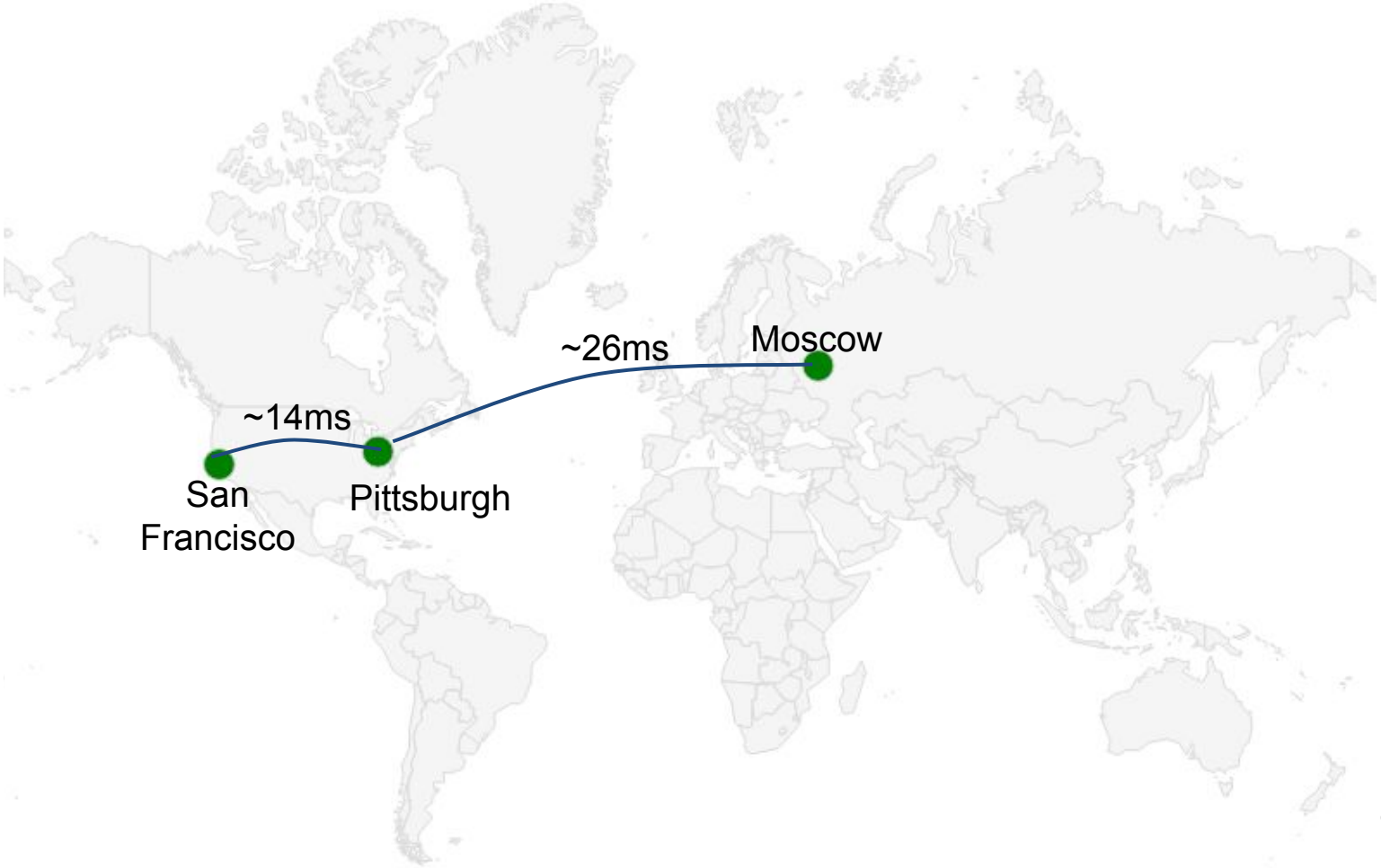
- **From:** 33 zettabytes in 2018

- **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

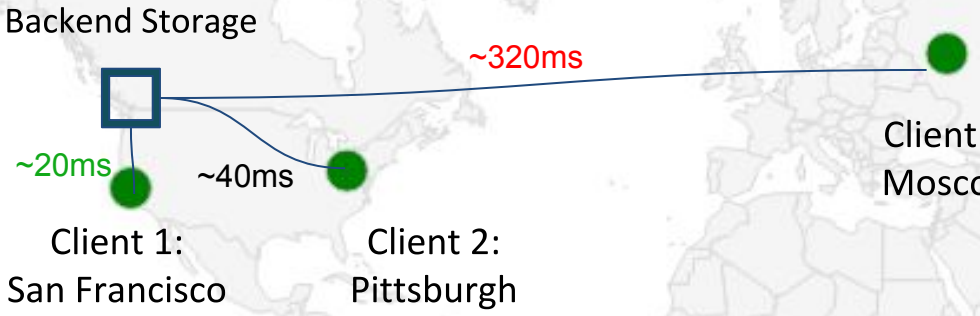
   Server has to read incoming packets and responds

   (IO or Disk latency)

   Message takes time to physically reach client

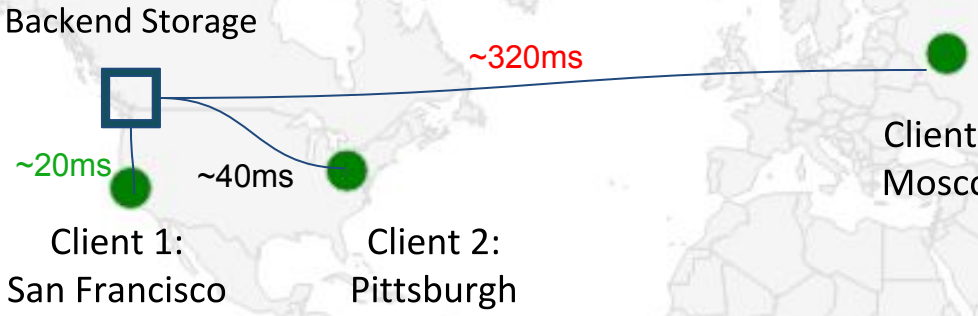
   (Network latency)

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

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

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

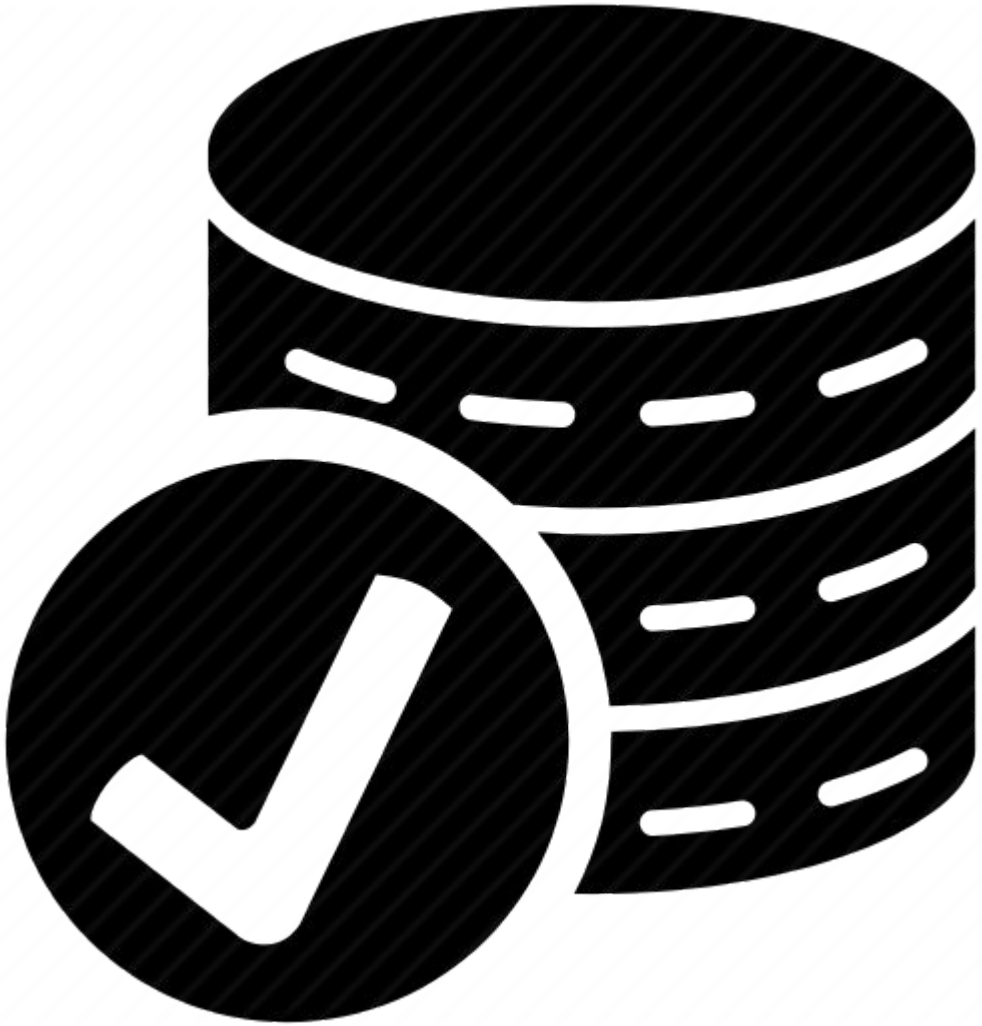
Replication is not infinite

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!

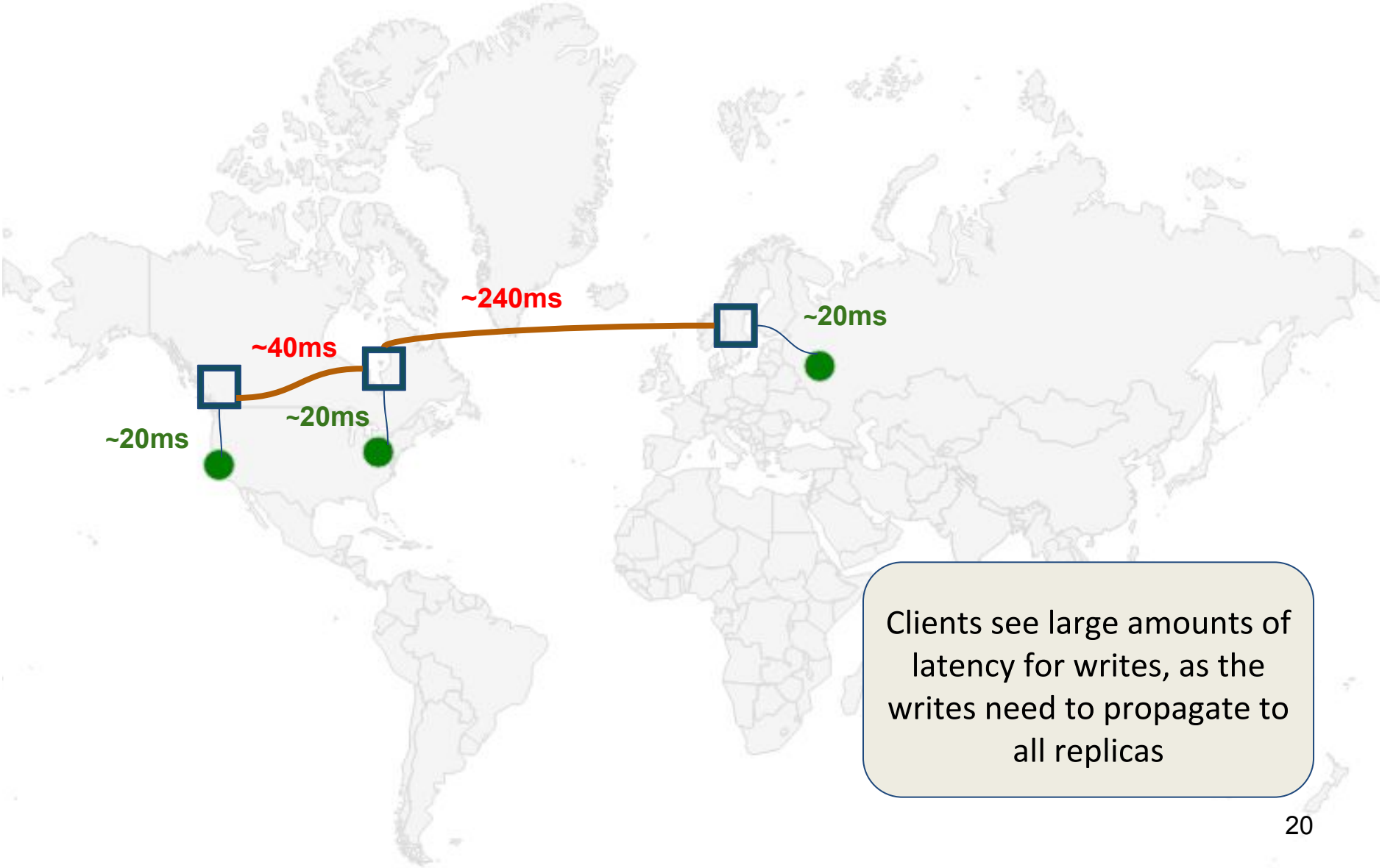
Data Consistency



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

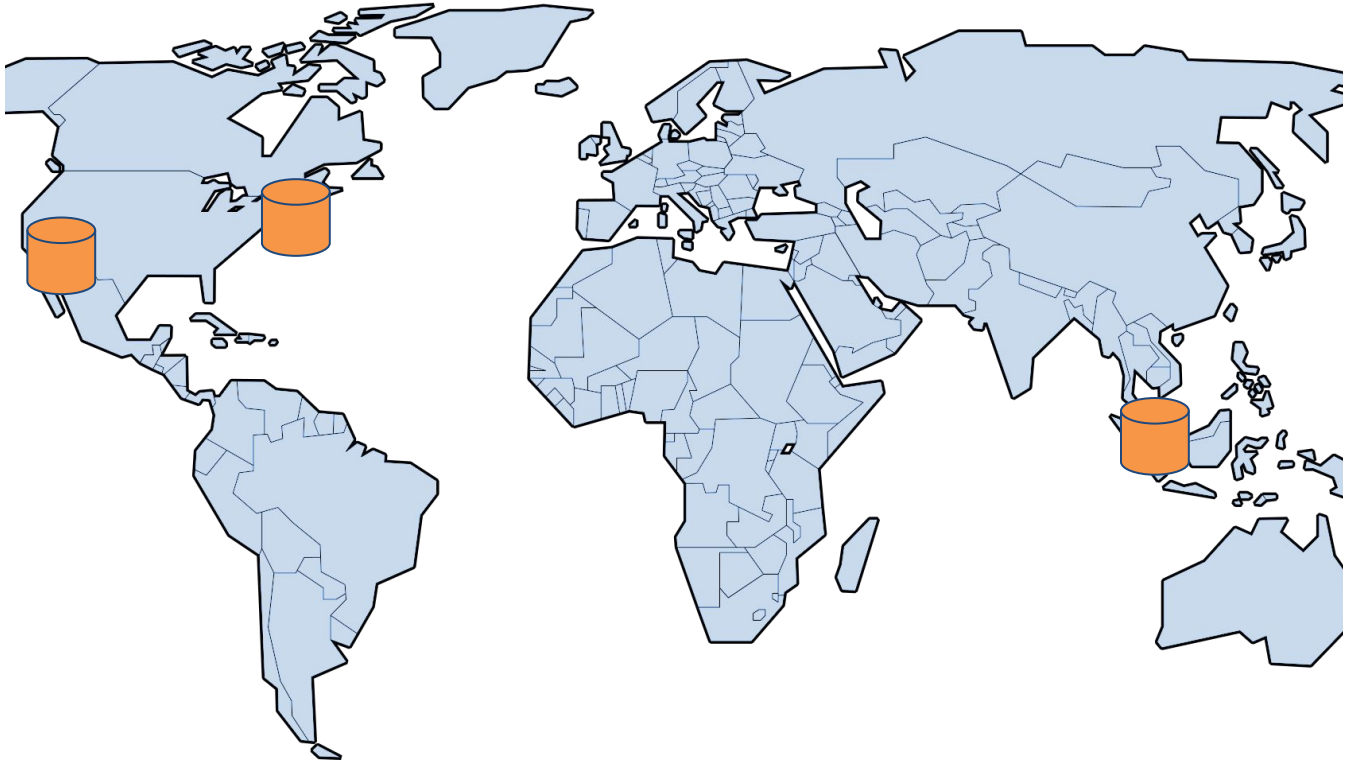
- **Advantages**
  - Low latency for reads
  - Reduce the workload of a single backend server
  - Handle failures of nodes by rerouting to alternative backup replica
- **Disadvantages**
  - Requires more storage capacity and cost
  - Updates are significantly slower
  - Changes must reflect on all datastores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

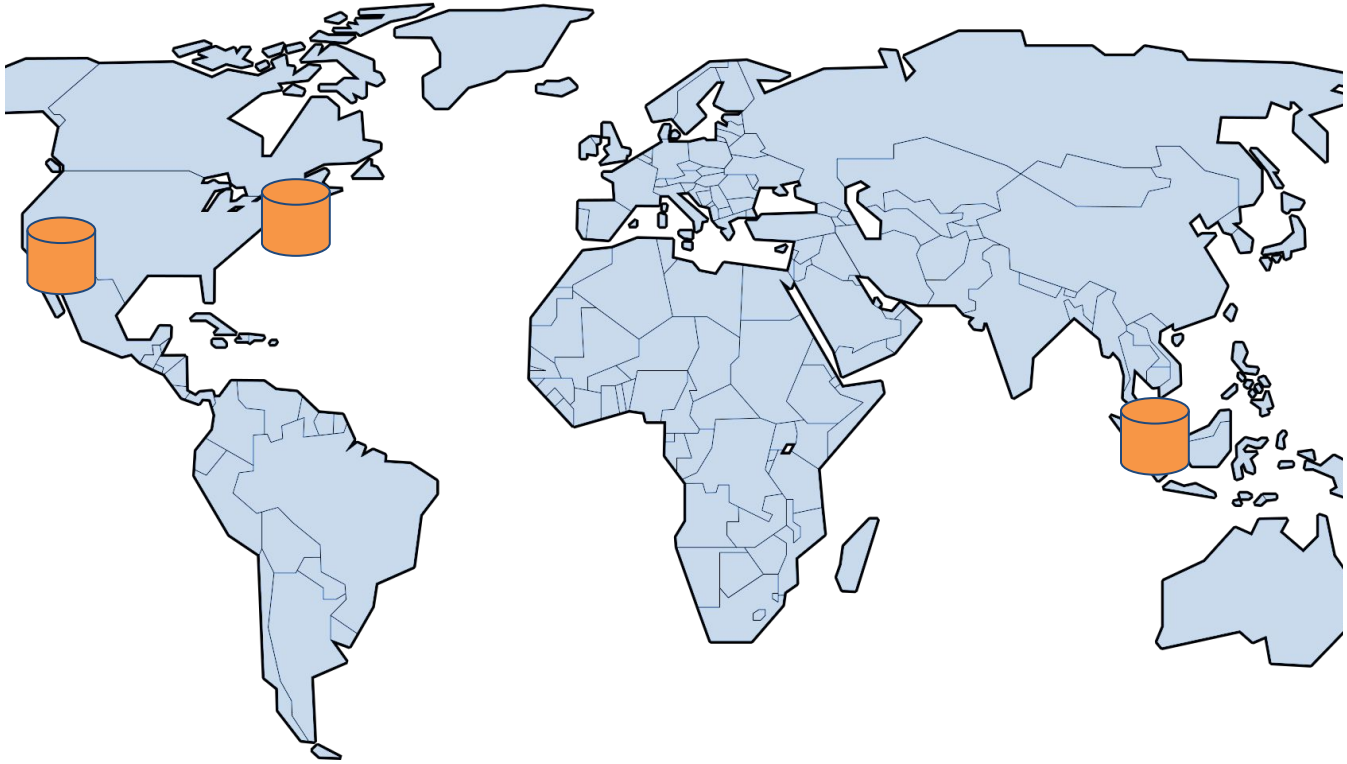
Data Consistency Example: Consider a Bank



Account	Balance
xxxxx-4437	$100

Bad Example

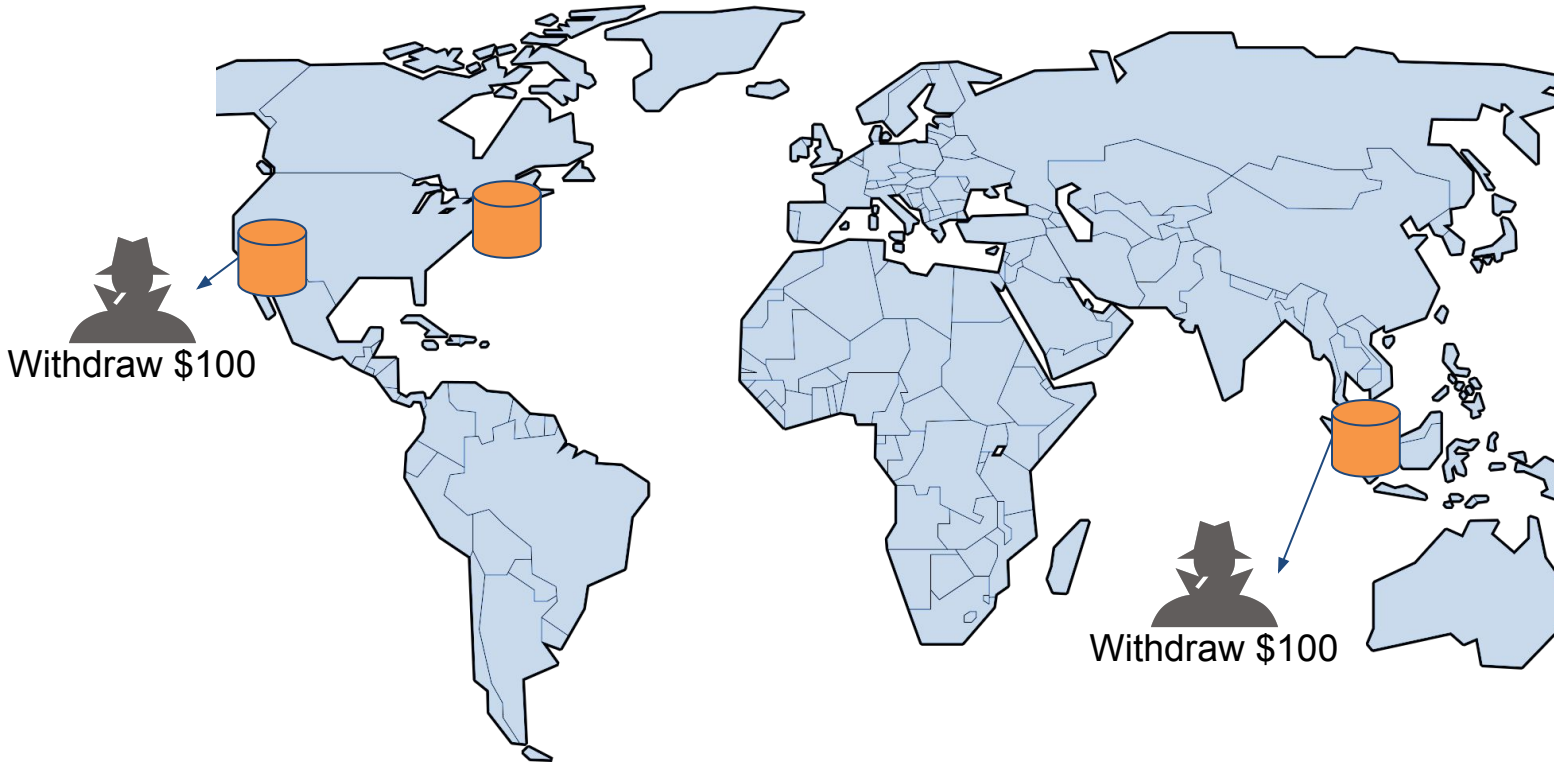
Allow concurrent writes



Account	Balance
xxxxx-4437	$100

Bad Example

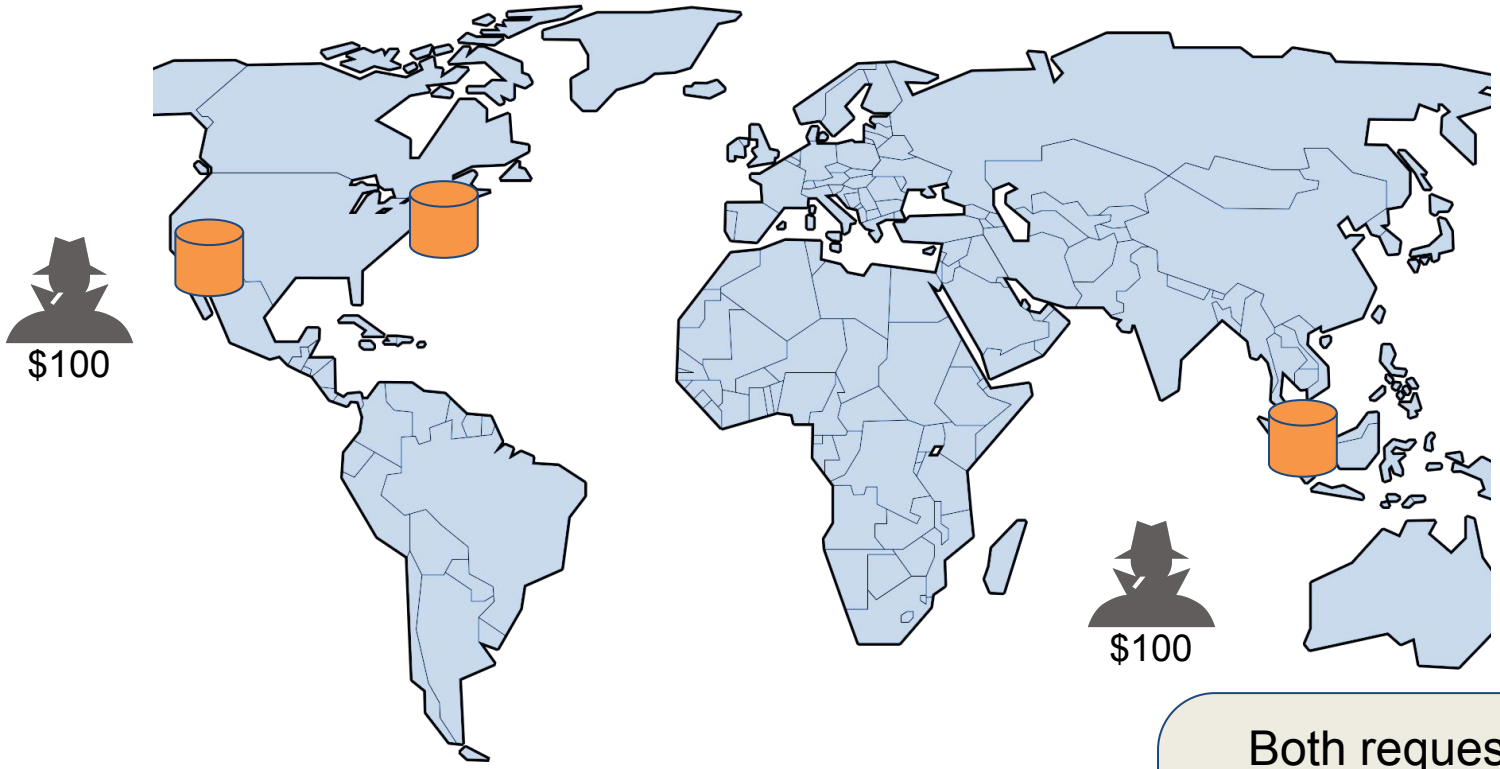
Allow concurrent writes



Account	Balance
xxxxx-4437	$100

Bad Example

Allow concurrent writes

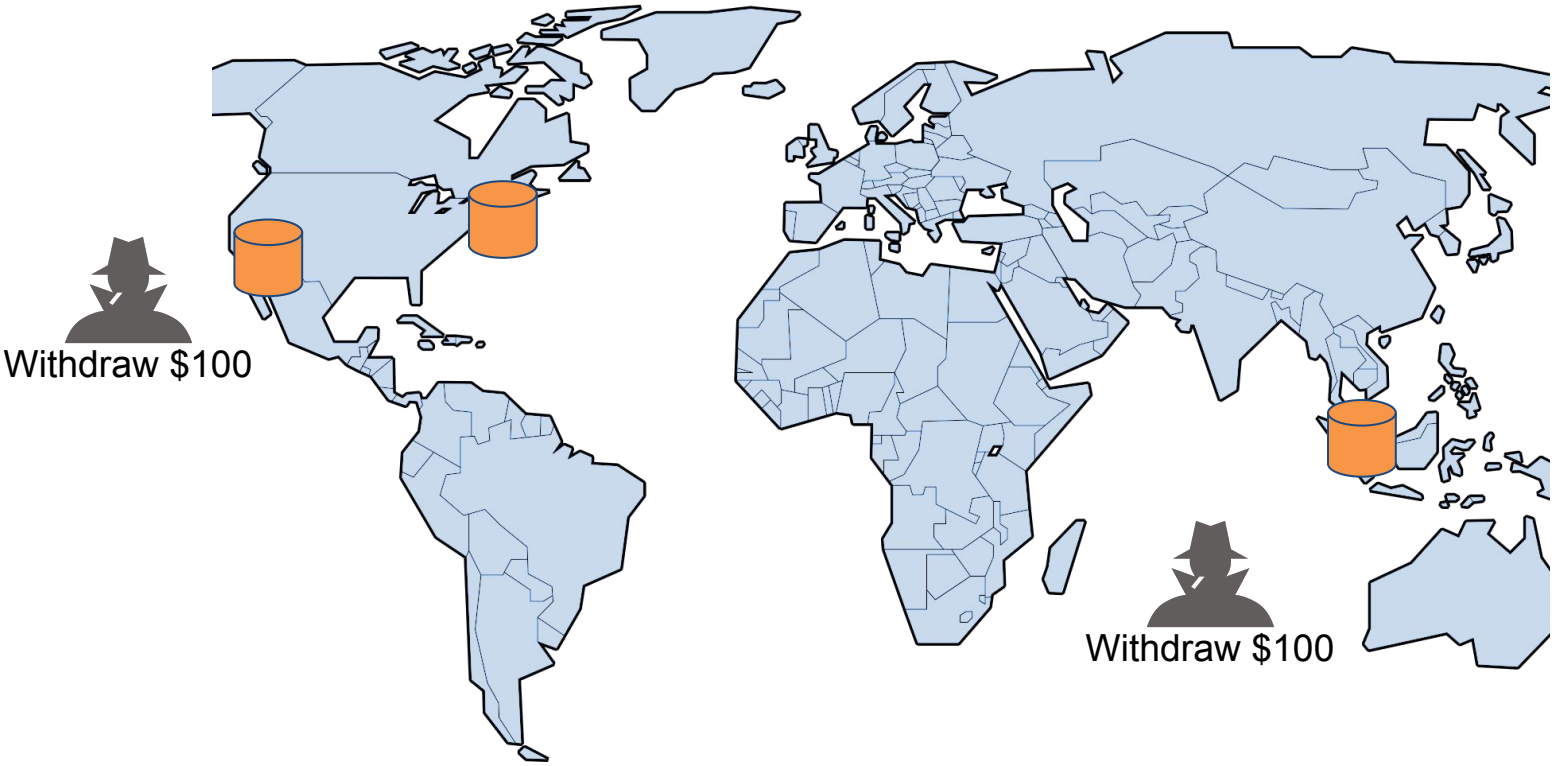


Account	Balance
xxxxx-4437	**$0**

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

Good Example

Global Locking



Account	Balance
xxxxx-4437	**$100**

Good Example

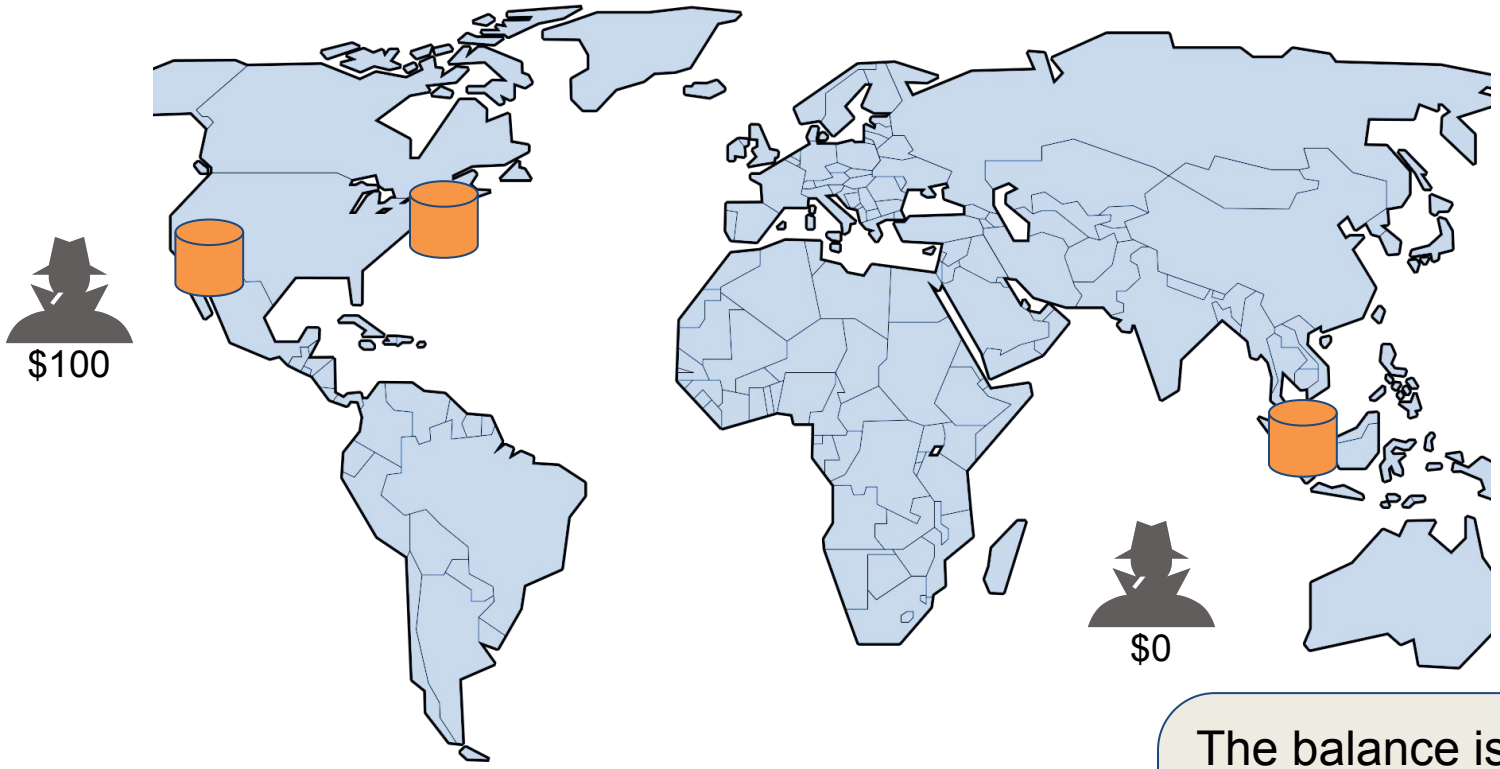
Global Locking

Account	Balance
xxxxx-4437	**$100**

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

Good Example

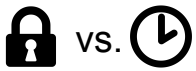
Global Locking



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

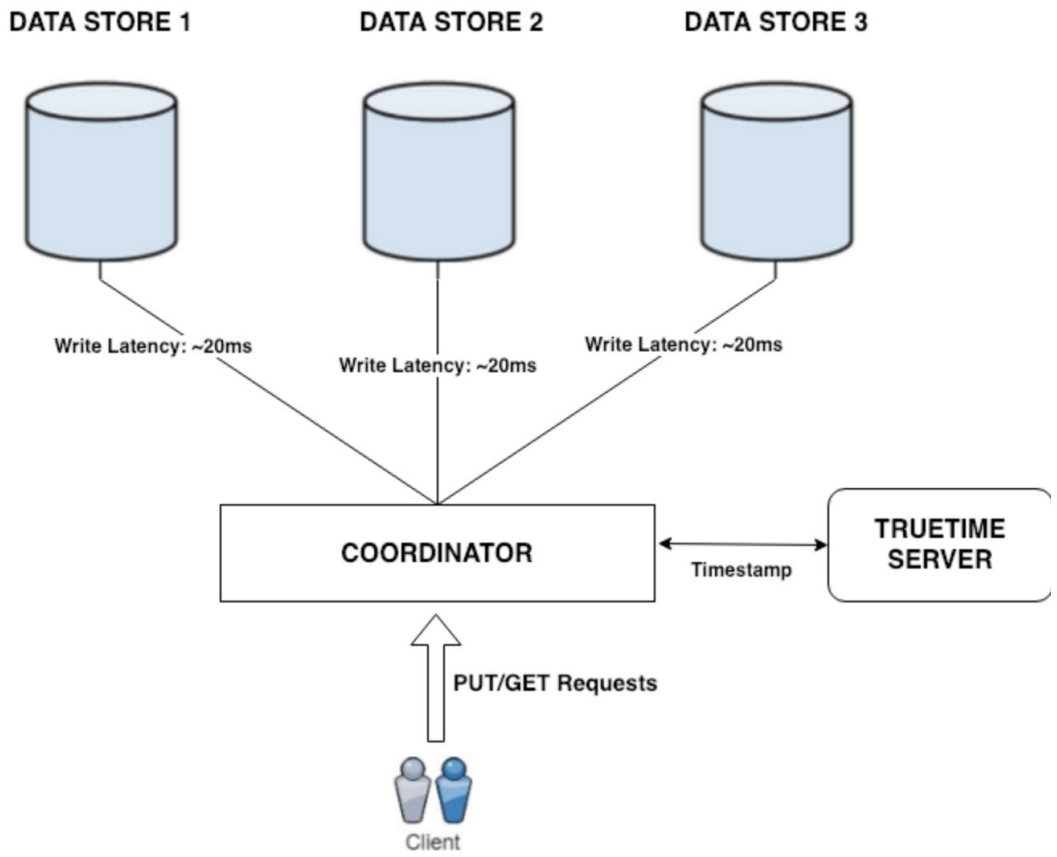
P3.3 Task 1: Strong Consistency

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



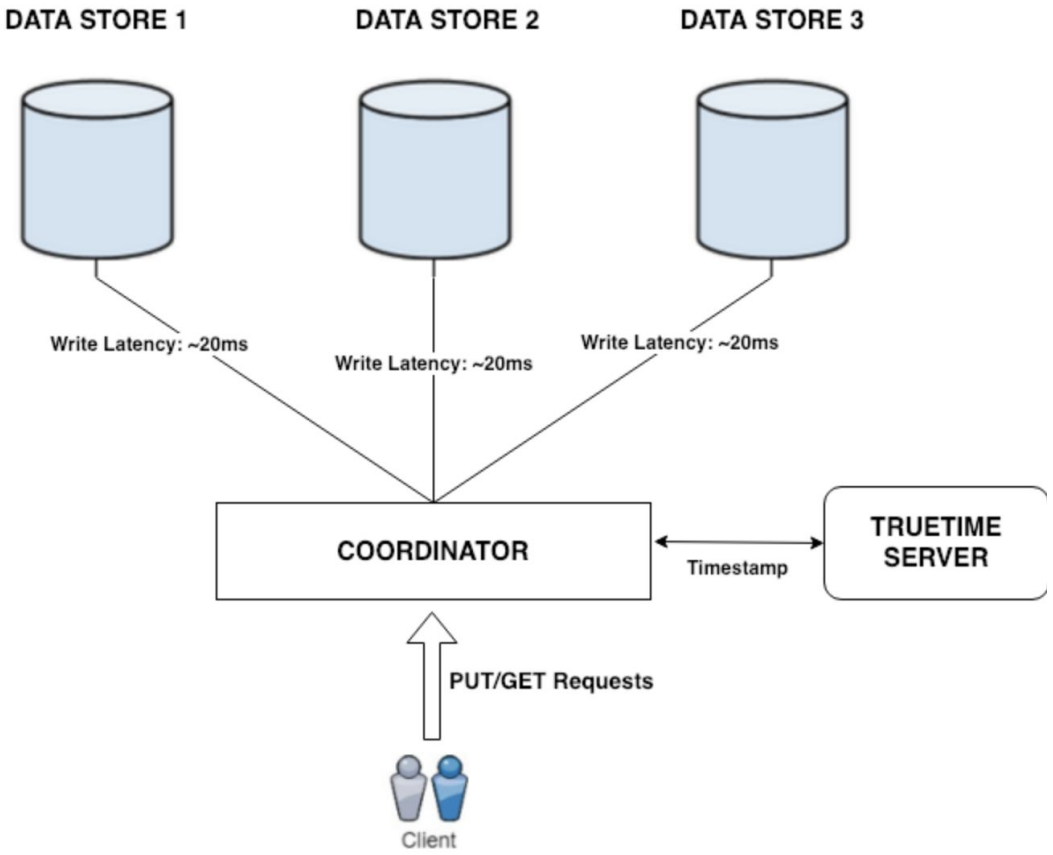
P3.3 Task 1: Strong Consistency

Single PUT request for key ‘X’

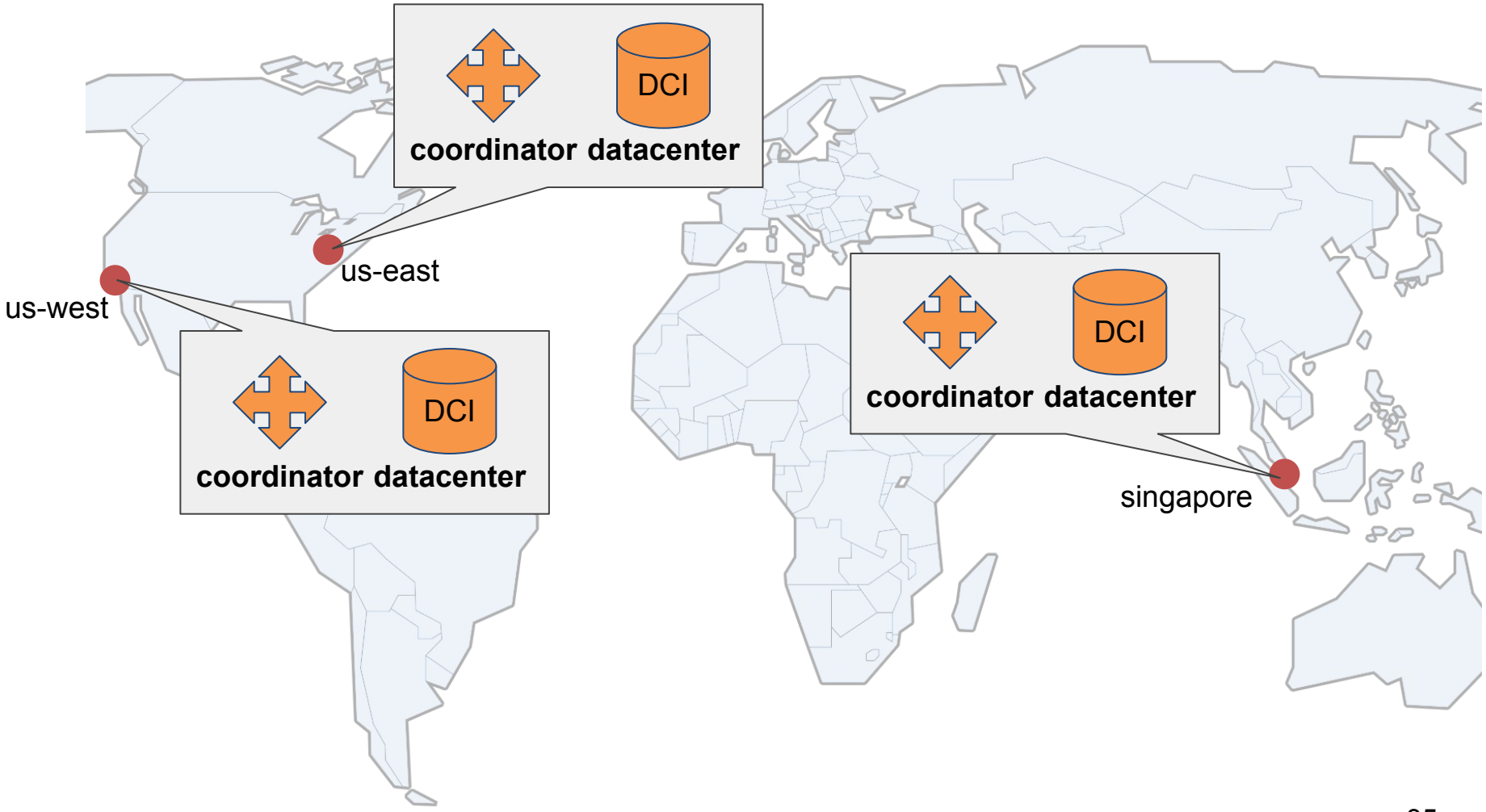
- Block all GETs for key ‘X’ until all datastores are updated
- GET requests for a different key ‘Y’ should **not** be blocked

Multiple PUT requests for ‘X’

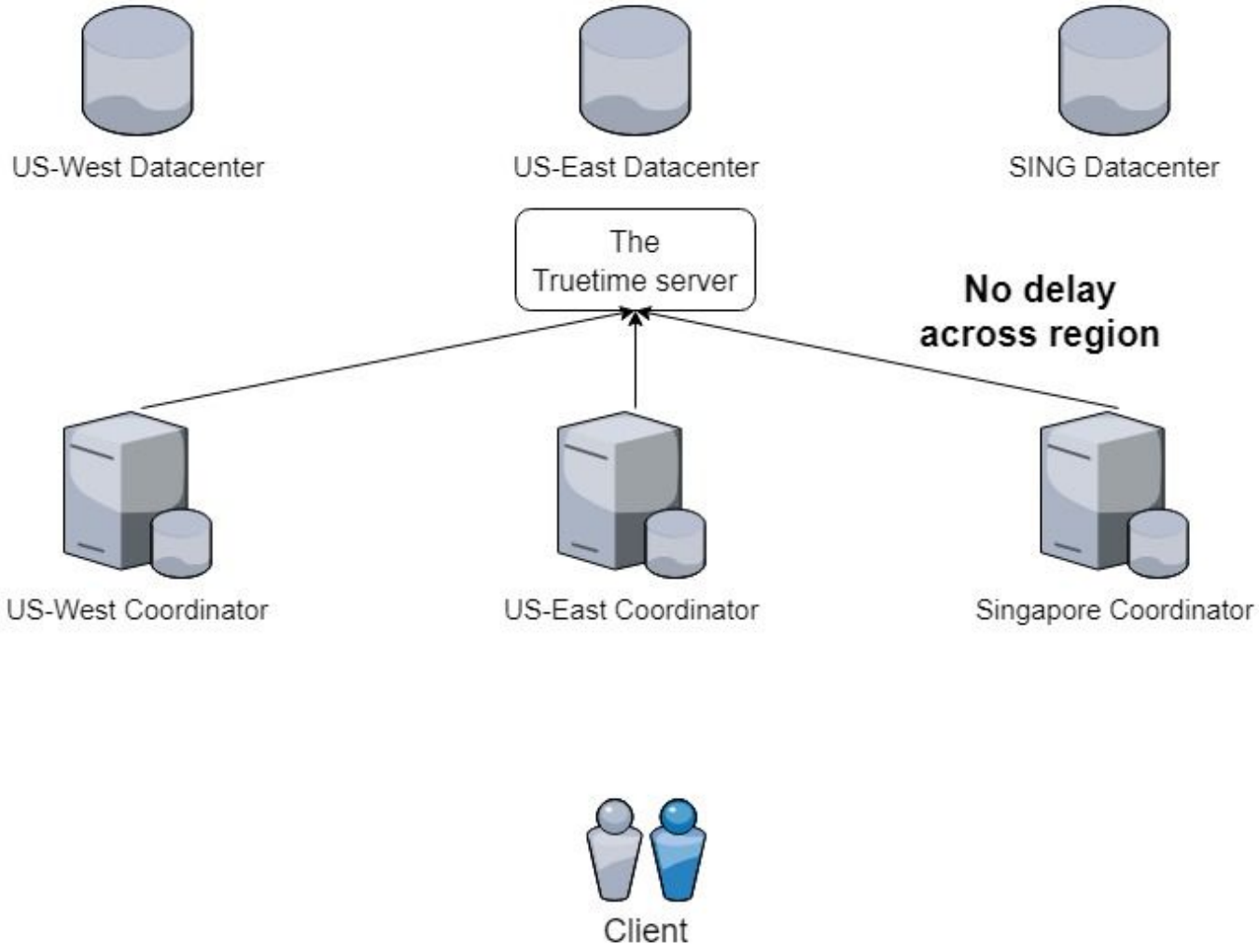
- Resolved in order of their timestamp received from the Truetime Server.
- GET requests must return the most recent value to the request timestamp



P3.3 Task 2:
Global Coordinators and Data Stores



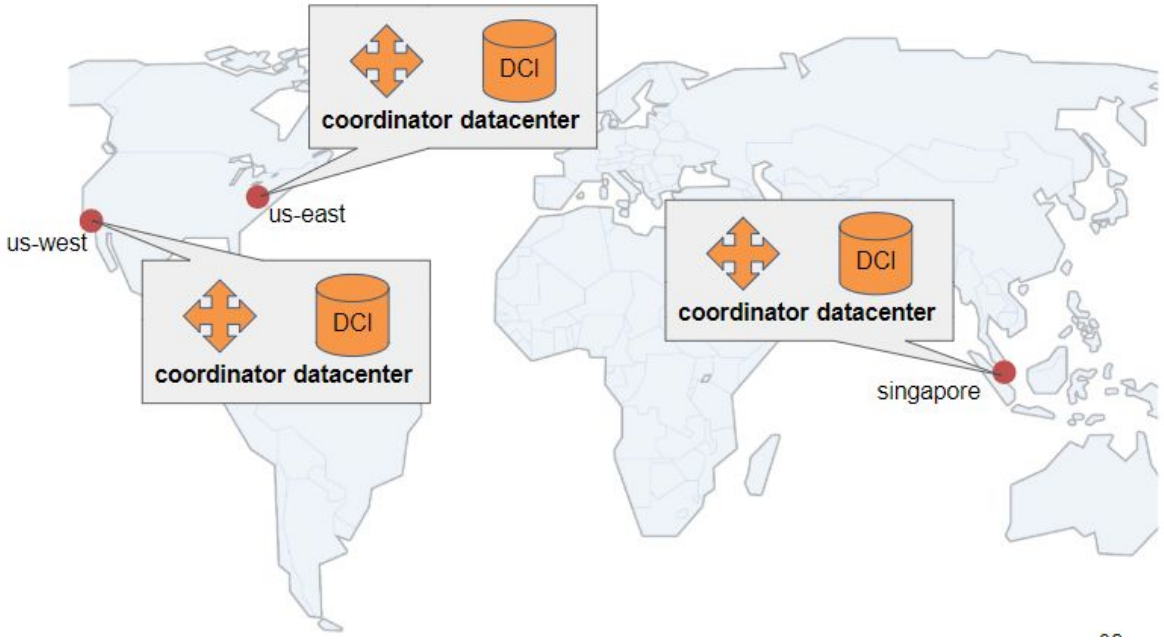
P3.3 Task 2: Architecture



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.

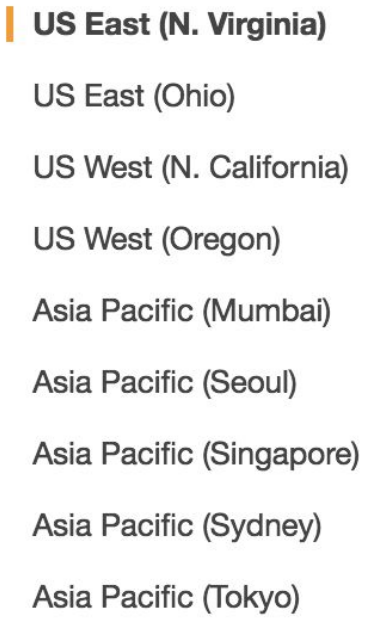


Task 2 Workflow and Example

- 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

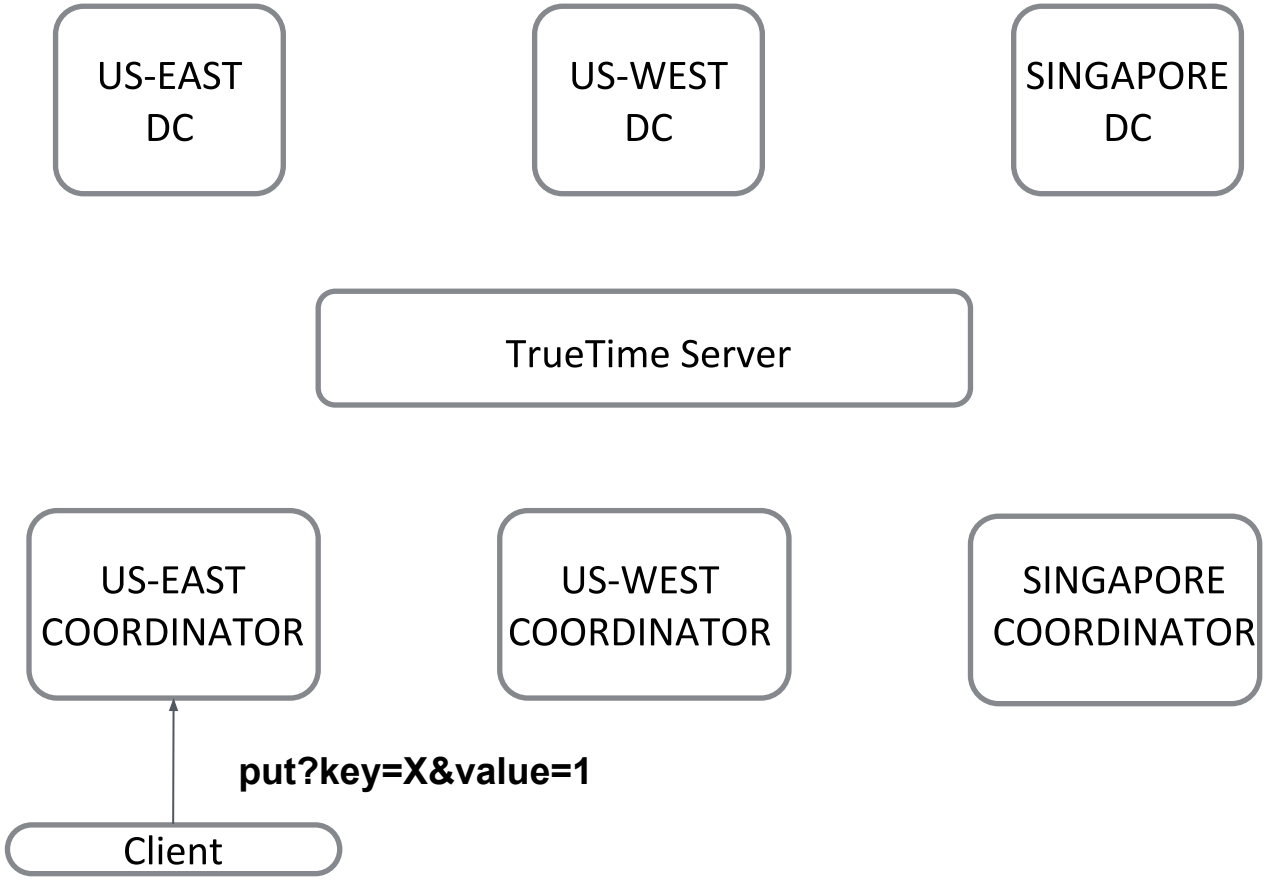


PRECOMMIT

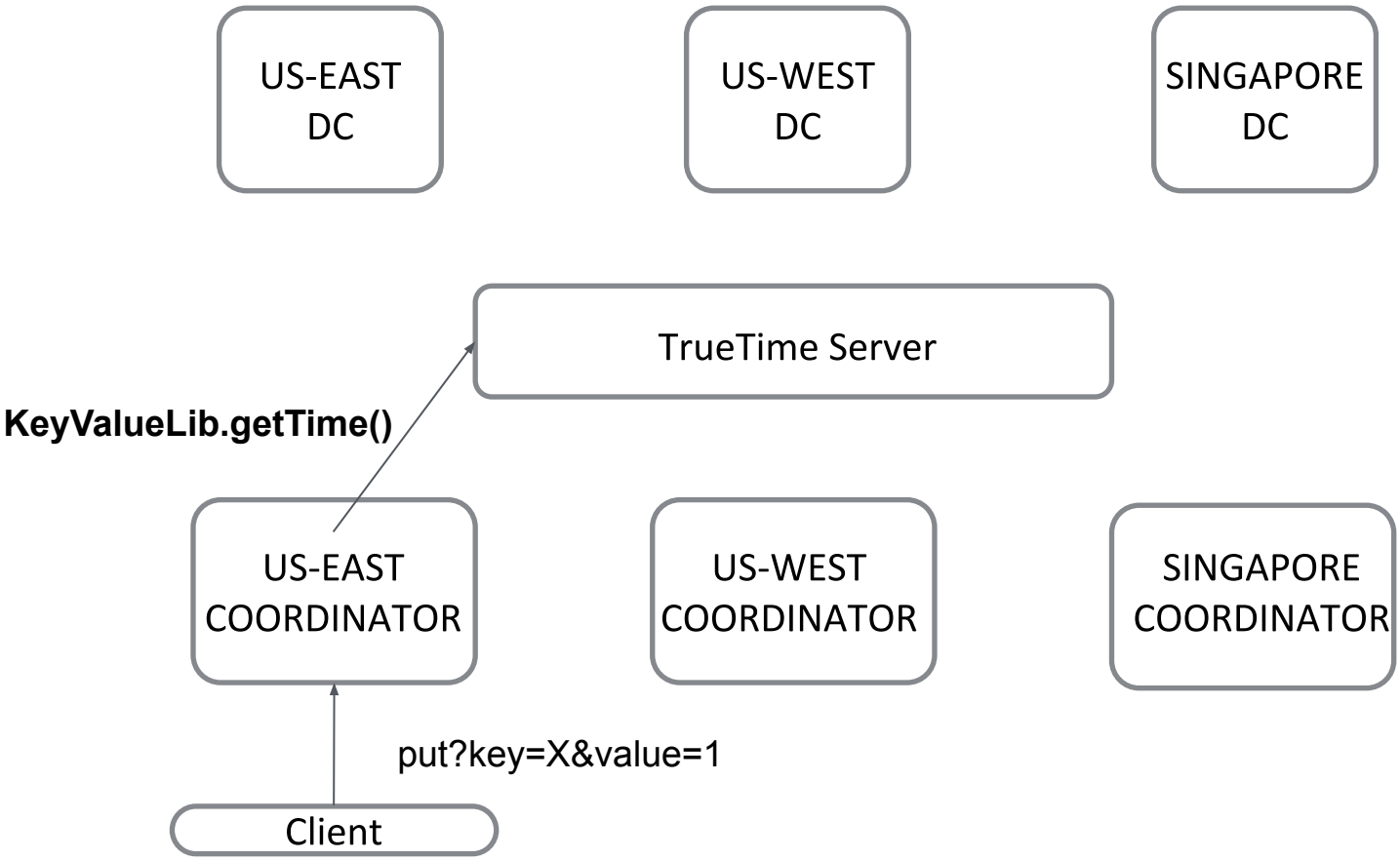
Contacts the Data Center of a given region and notifies it that a PUT request is being serviced for the specified key with the corresponding timestamp.

P3.3 Task 2:

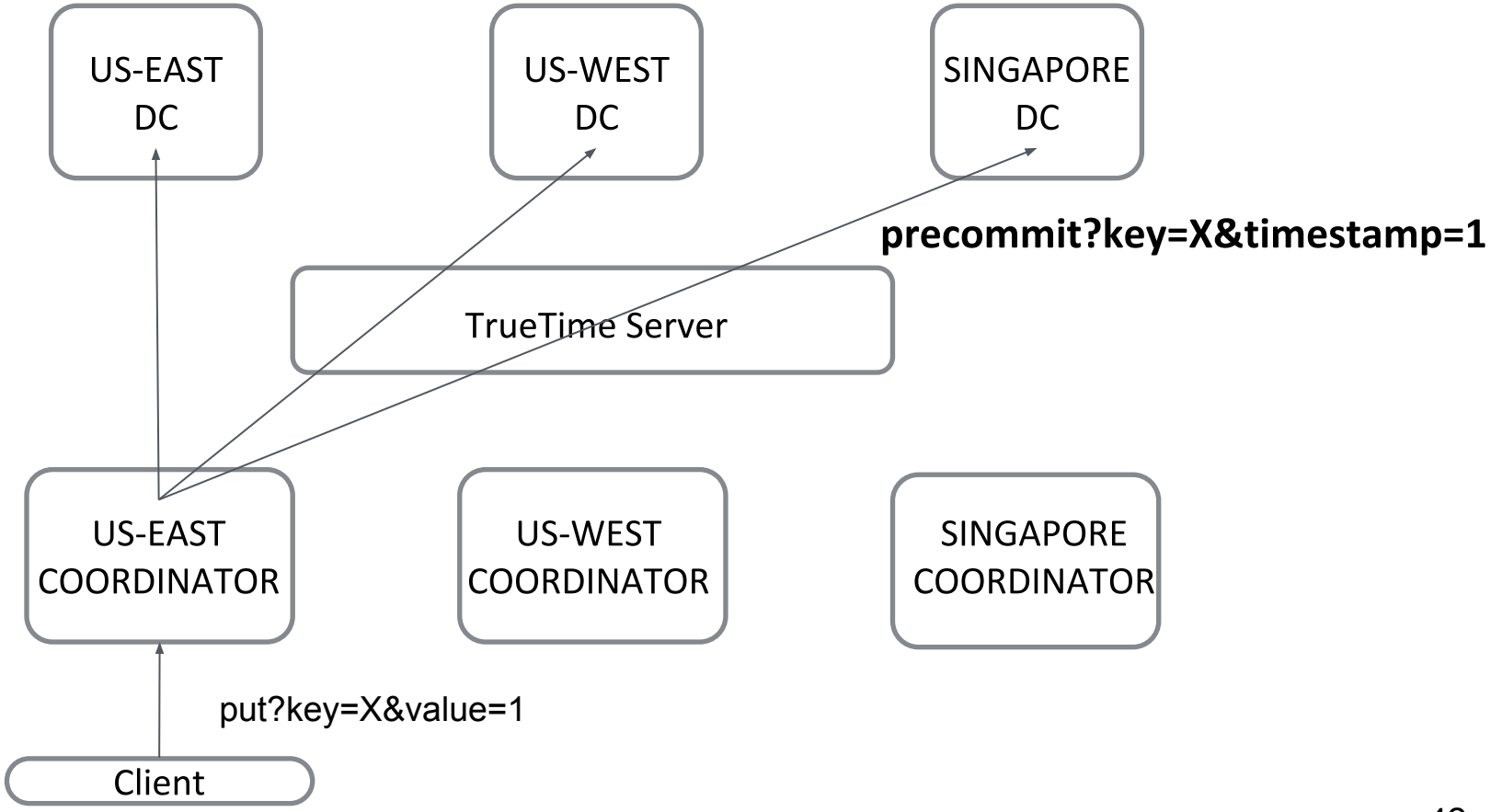
Complete KeyValueStore.java and Coordinator.java



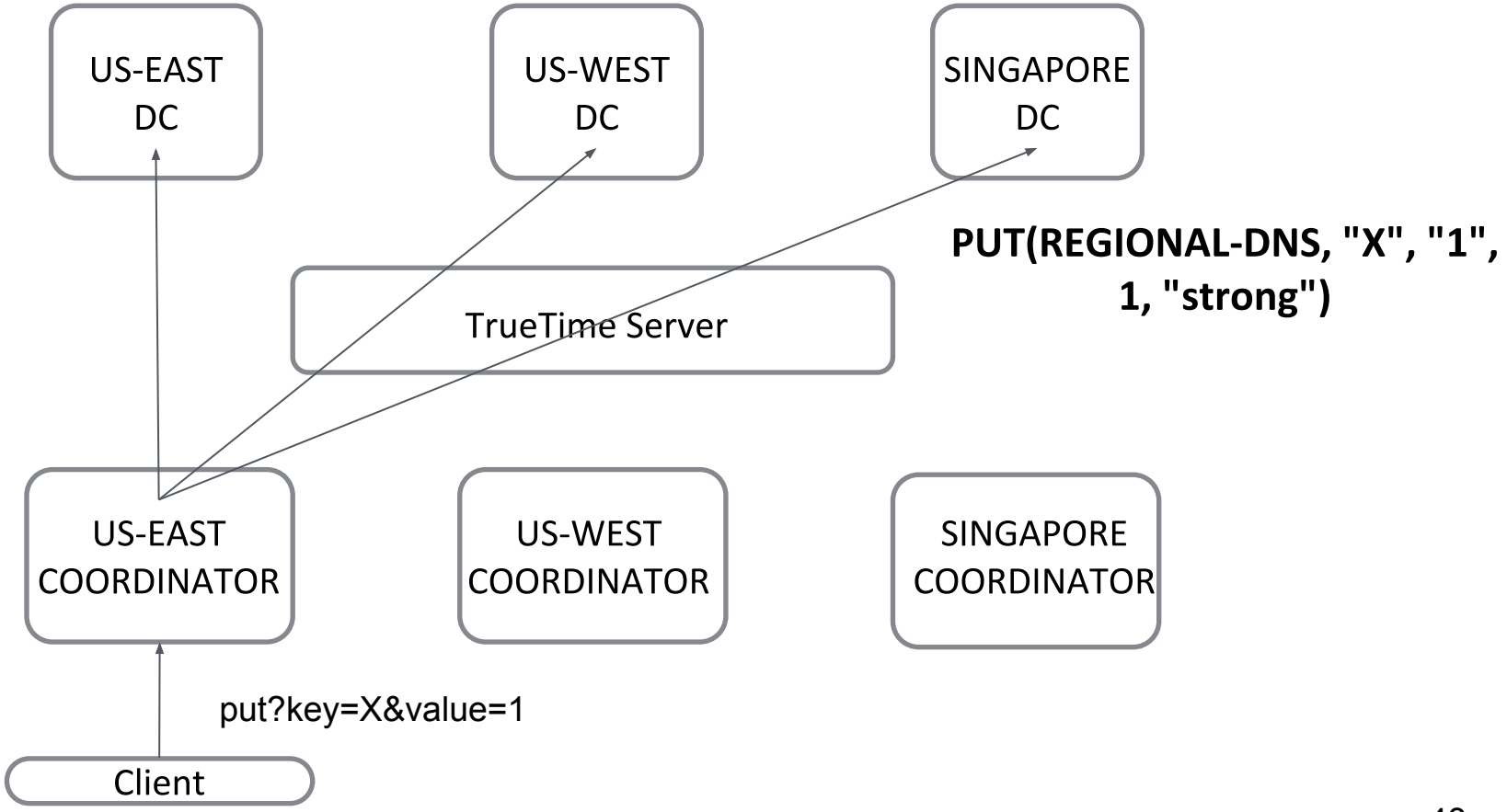
P3.3 Task 2:
Complete KeyValueStore.java (in DCs) and Coordinator.java (in Coordinators)



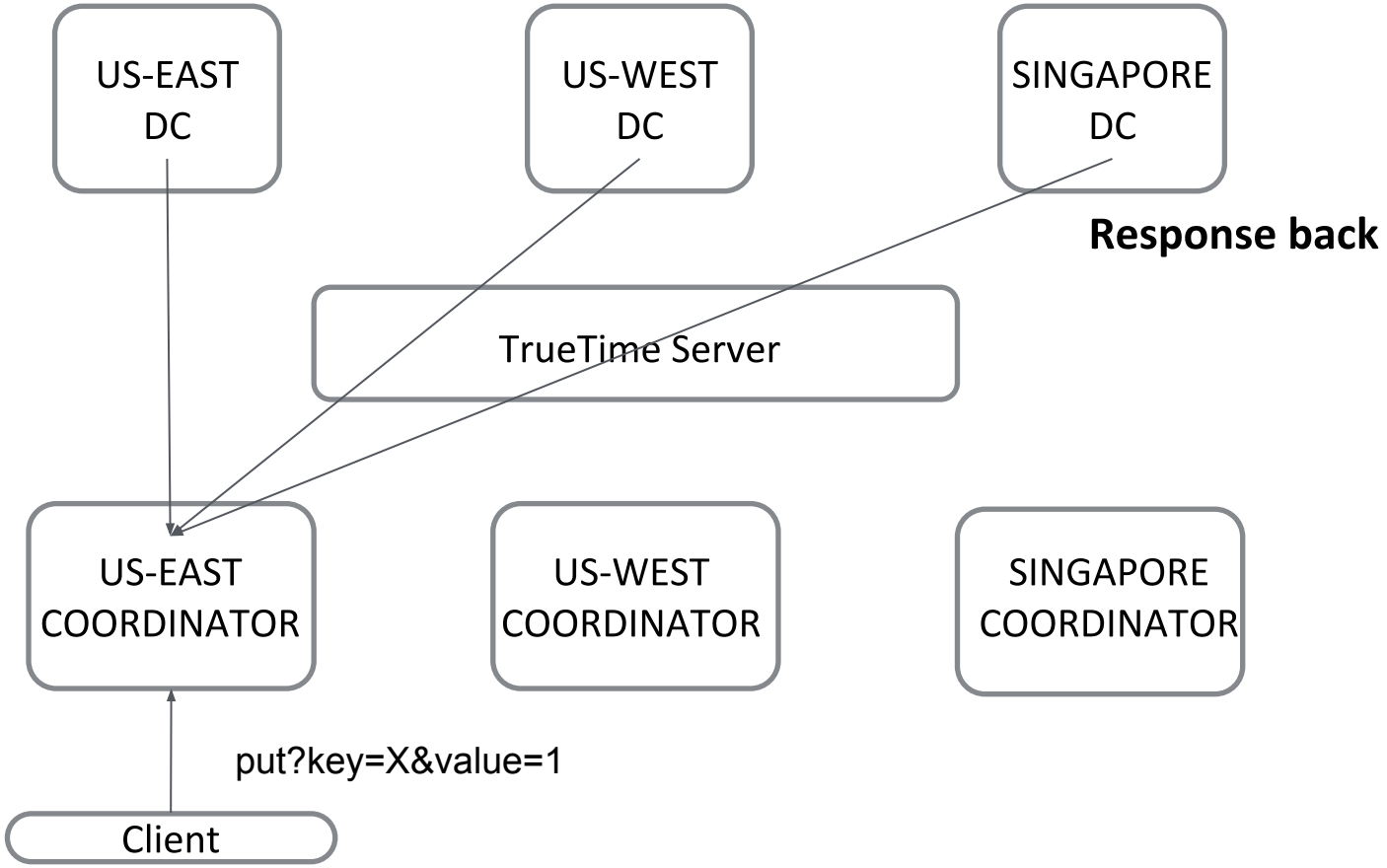
P3.3 Task 2:
Complete KeyValueStore.java (in DCs) and Coordinator.java (in Coordinators)



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 centers.

- Locking needs to be performed on Data centers.

- Lock by the key across all the Data centers 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 “./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!

Piazza FAQ

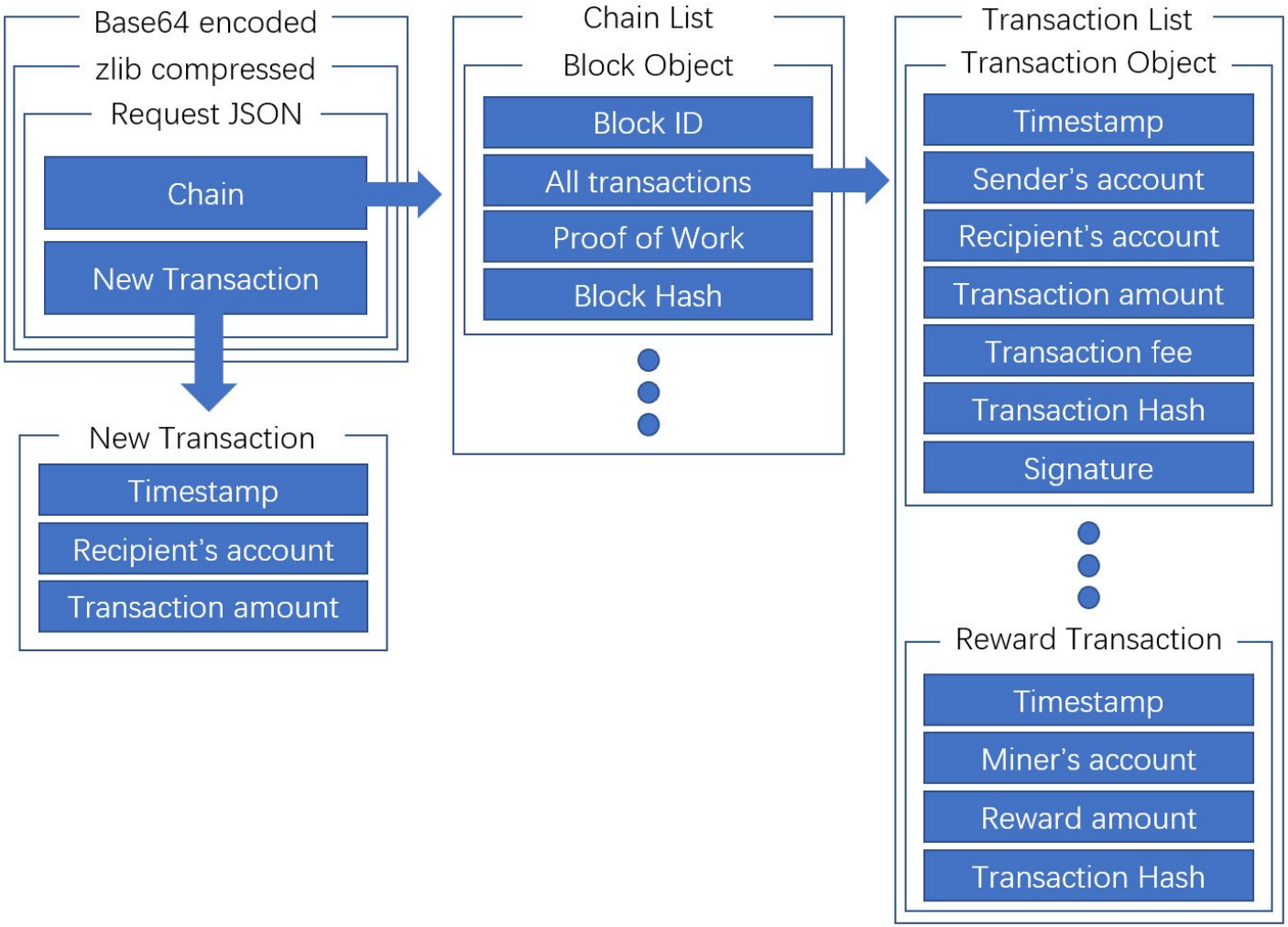
1. Search before asking a question

2. Post public questions when possible

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

TEAM PROJECT
Twitter Data Analytics

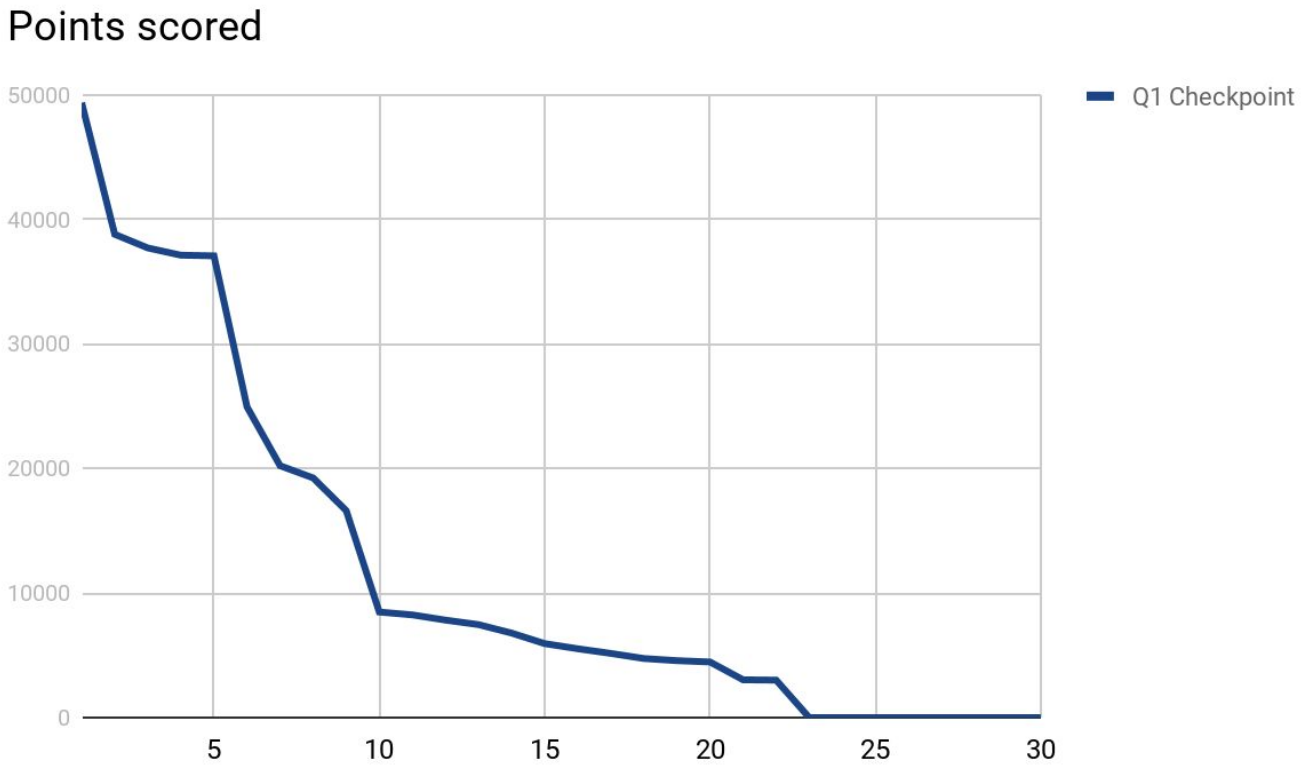
Query 1 Recap




```
{
  "chain": [
    {
      "all_tx": [
        {
          "recv": 509015179679,
          "amt": 500000000,
          "time": "1550721967779362304",
          "hash": "d50e5266"
        }
      ],
      "id": 0,
      "hash": "02899b89",
      "pow": "postpone"
    },
    {
      "all_tx": [
        {
          "send": 509015179679,
          "recv": 484054352161,
          "amt": 126848946,
          "fee": 12488,
          "time": "1550721967779391744",
          "hash": "5a2b4d71",
          "sig": 463884077351
        },
        {
          "recv": 1284110893049,
          "amt": 500000000,
          "time": "1550721967779424000",
          "hash": "7924c55e"
        }
      ],
      "id": 1,
      "hash": "0fce51c1",
      "pow": "fountain"
    },
    {
      "all_tx": [
        {
          "send": 1284110893049,
          "recv": 484054352161,
          "amt": 58759591,
          "fee": 5048,
          "time": "1550721967779447040",
          "hash": "b43737af",
          "sig": 1084970046728
        },
        {
          "recv": 34123506233,
          "amt": 500000000,
          "time": "1550721967779474176",
          "hash": "d705e74e"
        }
      ],
      "id": 2,
      "hash": "03635f77",
      "pow": "jeans"
    }
  ],
  "new_tx": {
    "recv": 837939704897,
    "amt": 430642077,
    "time": "1550721967779486720"
  }
}
```

Team Project - Q1 CKPT1

- 30 teams attempted a Query 1 submission
- 23 teams got a 10-minute submission
- 6 teams reached 35000 RPS



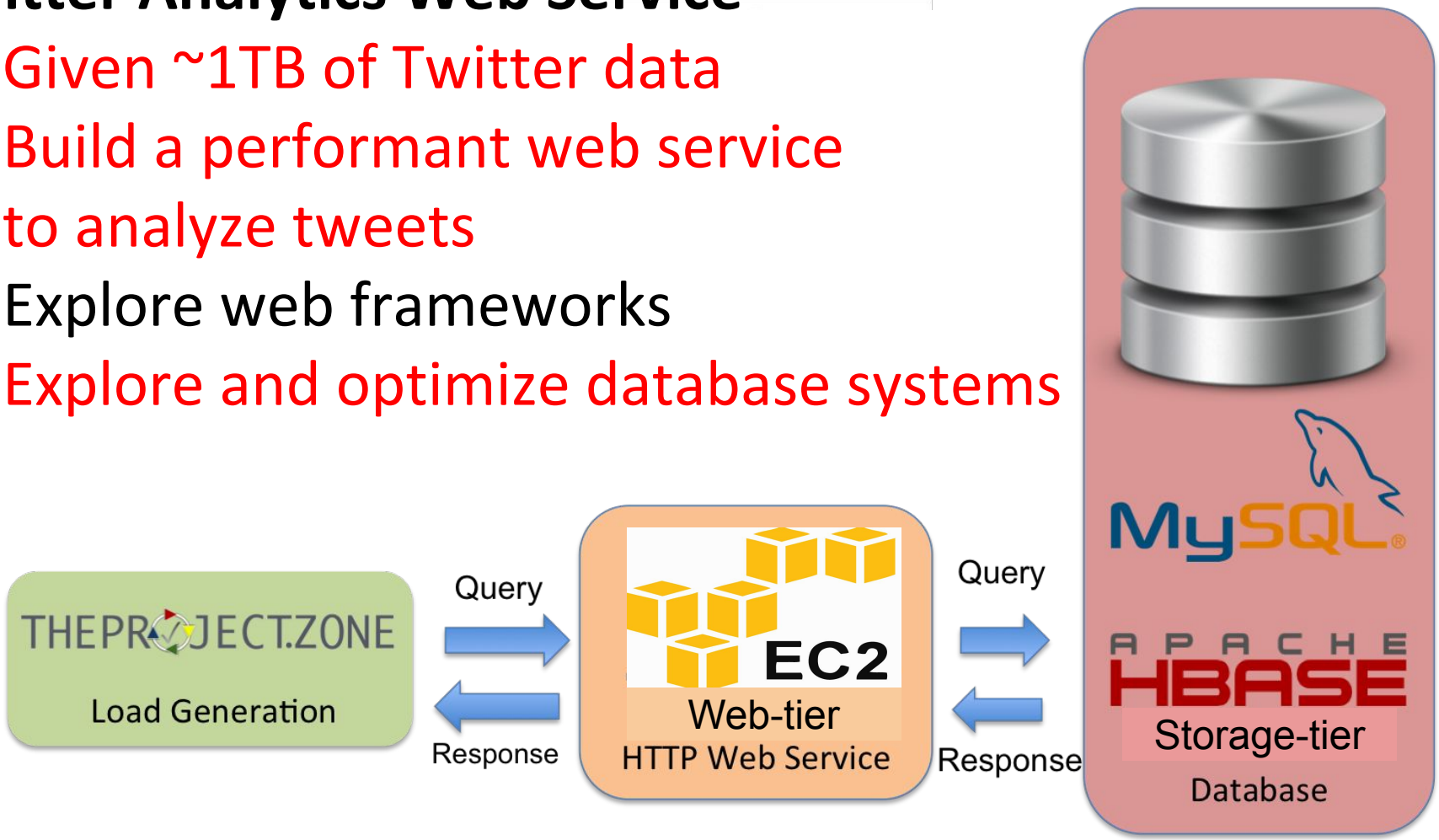
Read about Query 2 Now.
Start ETL Now.

Value	Target RPS	Weight	Due date (at 11:59PM EST)
Checkpoint Report	-	5%	Sunday, October 13
Query 1 Checkpoint	-	5%	Sunday, October 13
Query 1 Final	35000	10%	Sunday, October 20
Query 2 Checkpoint	-	10%	Sunday, October 20
Query 2 Final	10000	50%	Sunday, October 27
Final Report + Code	-	20%	Tuesday, October 29

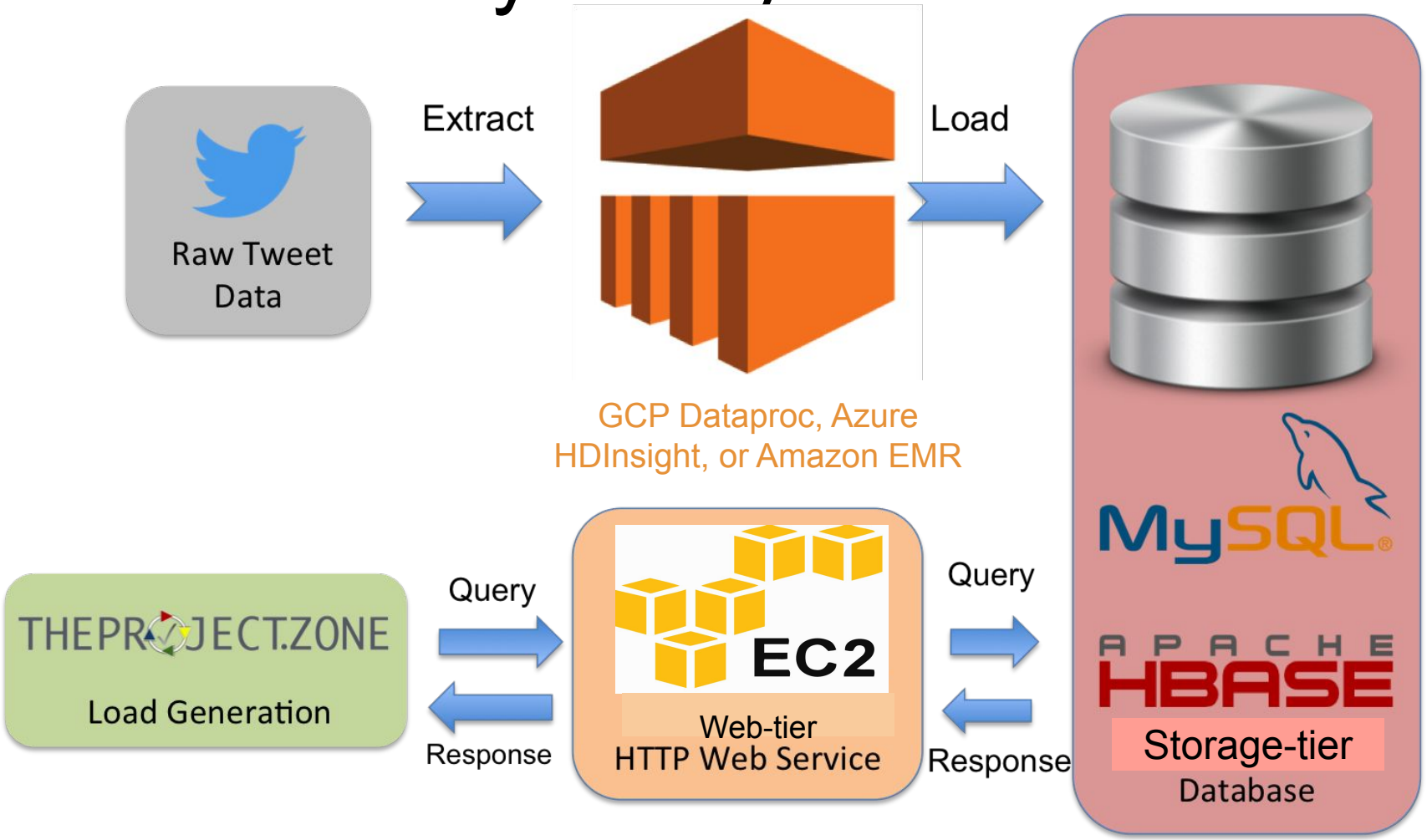
Team Project

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

Query 2 - User Recommendation System

Use Case: When you follow someone on twitter, recommend close friends.

Three Scores:

- Interaction Score - closeness
- Hashtag Score - common interests
- Keywords Score - to match interests

Final Score: Interaction Score * Hashtag Score * Keywords Score

Query:

GET /q2?
user_id=<ID>&
type=<TYPE>&
phrase=<PHRASE>&
hashtag=<HASHTAG>

Response:

<TEAMNAME>,<AWSID>\n
uid\tname\tdescription\ttweet\n
uid\tname\tdescription\ttweet

Query 2 Example

```
GET /q2?
user_id=100123&
type=retweet&
phrase=hello%20cc&
hashtag=cmu

TeamCoolCloud,1234-0000-0001
100124\tAlan\tScientist\tDo machines think?\n
100125\tKnuth\tprogrammer\thello cc!
```

Reminders about Penalties

- M family instances **only**, smaller than or equal to **large** type
- Only General Purpose (gp2) SSDs are allowed for storage
  - so **m5d is not allowed** since it uses NVMe storage
- 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)

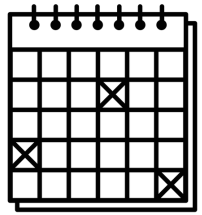
Phase 1 Budget

- AWS budget of $45 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

Tips

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

Suggested Tasks for Phase 1



Phase	Deadline (11:59PM ET)
Phase 1 (20%) - **Query 1** - **Query 2**	• Q1 CKPT (5%): Sun, 10/13 • Report1 (5%): Sun, 10/13 • **Q1 FINAL (10%): Sun, 10/20** • **Q2 CKPT (10%): Sun, 10/20** • Q2M & Q2H FINAL (50%): Sun, 10/27 • Report2 (20%): Tue, 10/29
Phase 2 (30%) - **Add Query 3**	• Live Test on Sun, 11/11
Phase 3 (50%) - **Managed Services**	• Live Test on Sun, 12/02


THANK YOU