15-319 / 15-619
Cloud Computing

Recitation 9

March 21, 2017

Overview

- **Last week's reflection**
  - Last week was Spring Break!
- **This week's schedule**
  - Quiz 8 – due on Friday, March 24th (Module 15)
  - Project 3.3 – due on Sunday, March 26th
- **Twitter Analytics: The team project phase 1 is due next week! Hurry up!**

This Week: Conceptual Content

OLI UNIT 4: Cloud Storage

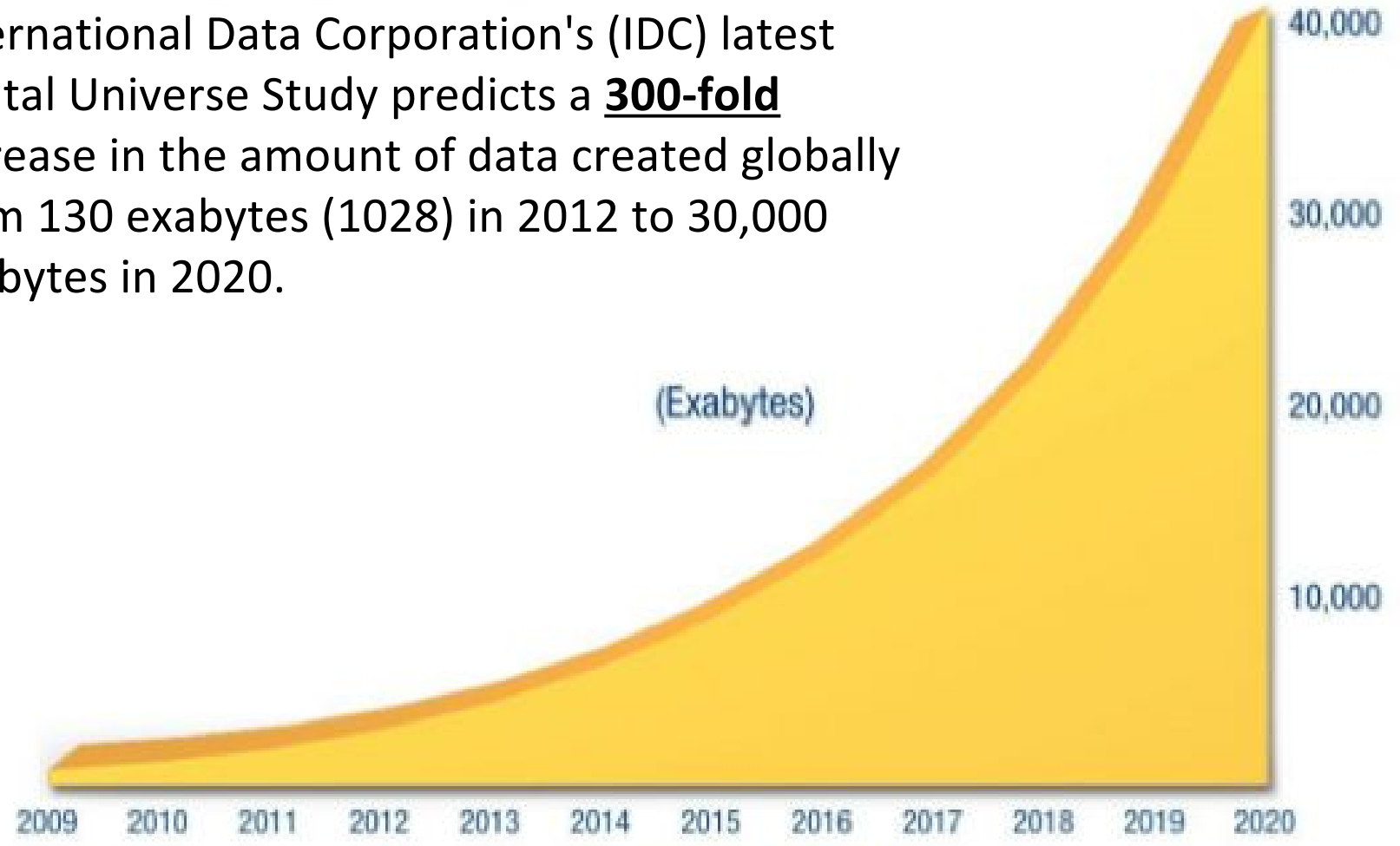
- **Module 15: Case Studies: Distributed File System**
  - HDFS
  - Ceph
- **Quiz 8**
  - **DUE on Friday, March 24th**

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

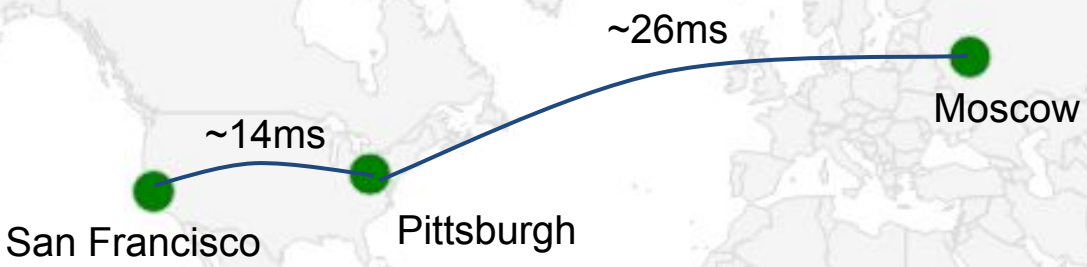
International Data Corporation's (IDC) latest Digital Universe Study predicts a **300-fold** increase in the amount of data created globally from 130 exabytes (1028) in 2012 to 30,000 exabytes in 2020.



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

Users are Global

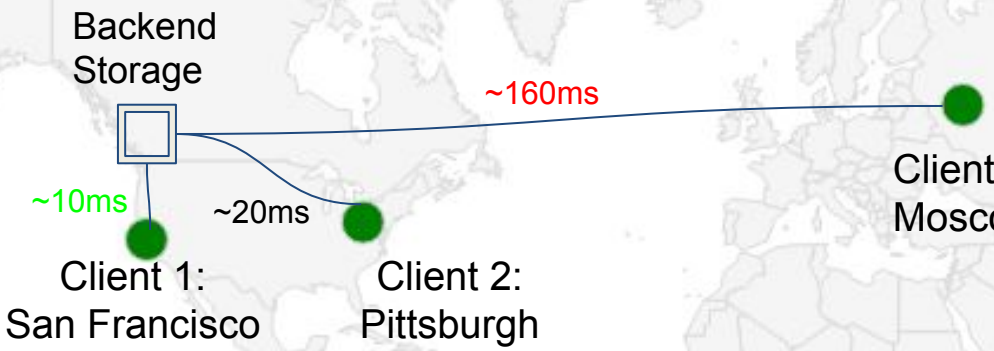
- Speed of Light (≈3.00×10^8 m/s)
- Inherent latencies



Typical End-To-End Latency

- Typical end-to-end latency
  - Client request
  - Network latency (to backend)
  - Server response
    - Includes fetching and processing data from backend
  - Network latency (from backend)
  - Client response

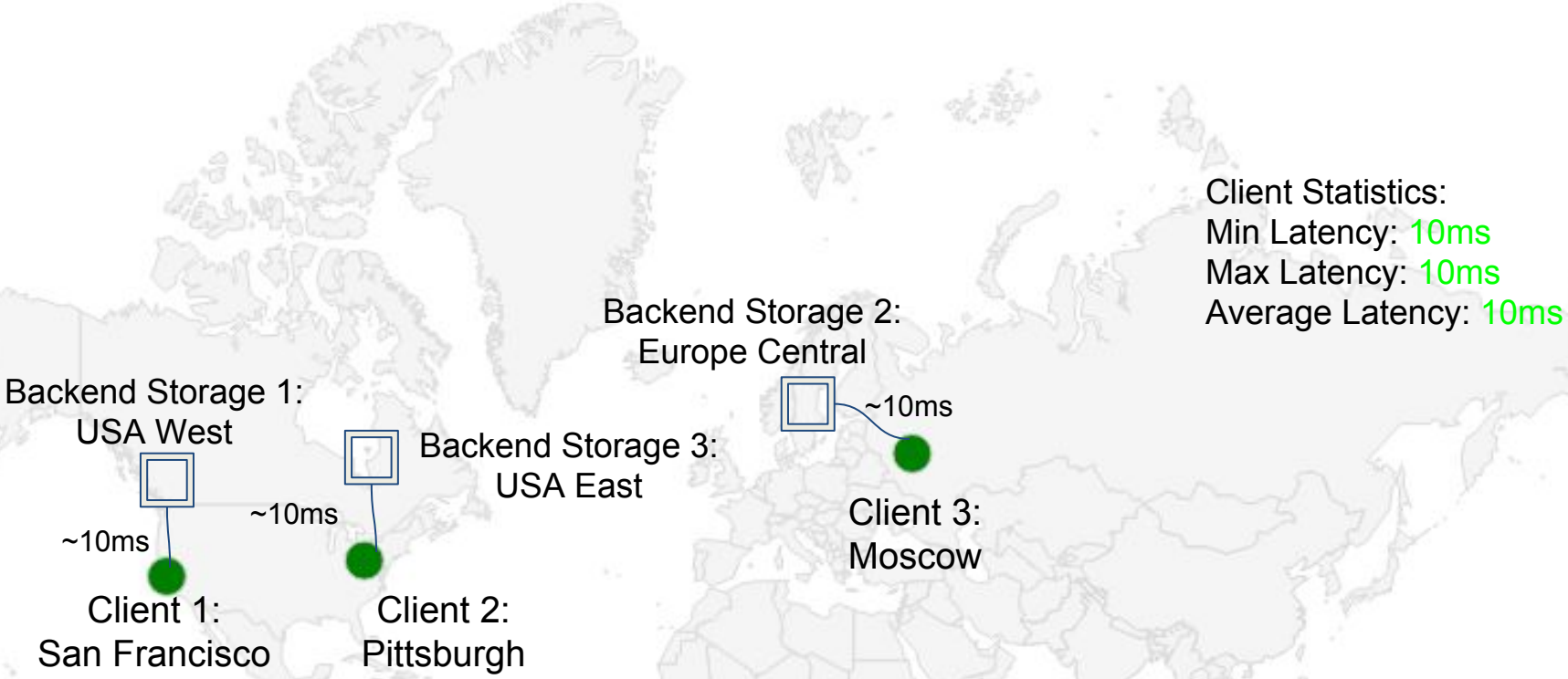
Latency with a Single Backend



Client Statistics:
Min Latency: 10ms
Max Latency: 160ms
Average Latency: 63ms

Replicate the Data Globally

Replicate the Data Close to Users

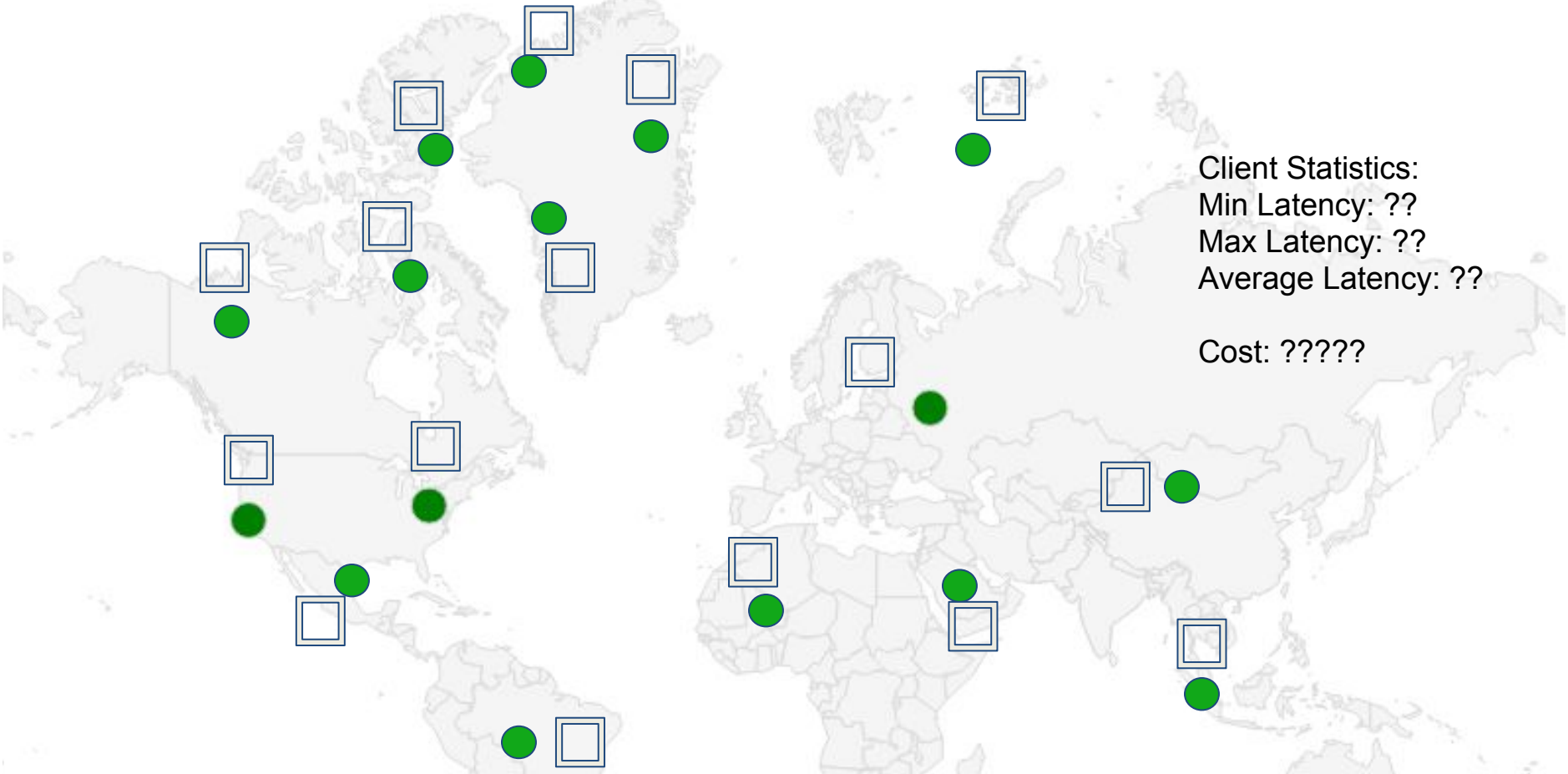


Client Statistics:
Min Latency: 10ms
Max Latency: 10ms
Average Latency: 10ms

Replicas

- 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 seen by clients
- But how about the cost?

What If We Continue to Replicate?

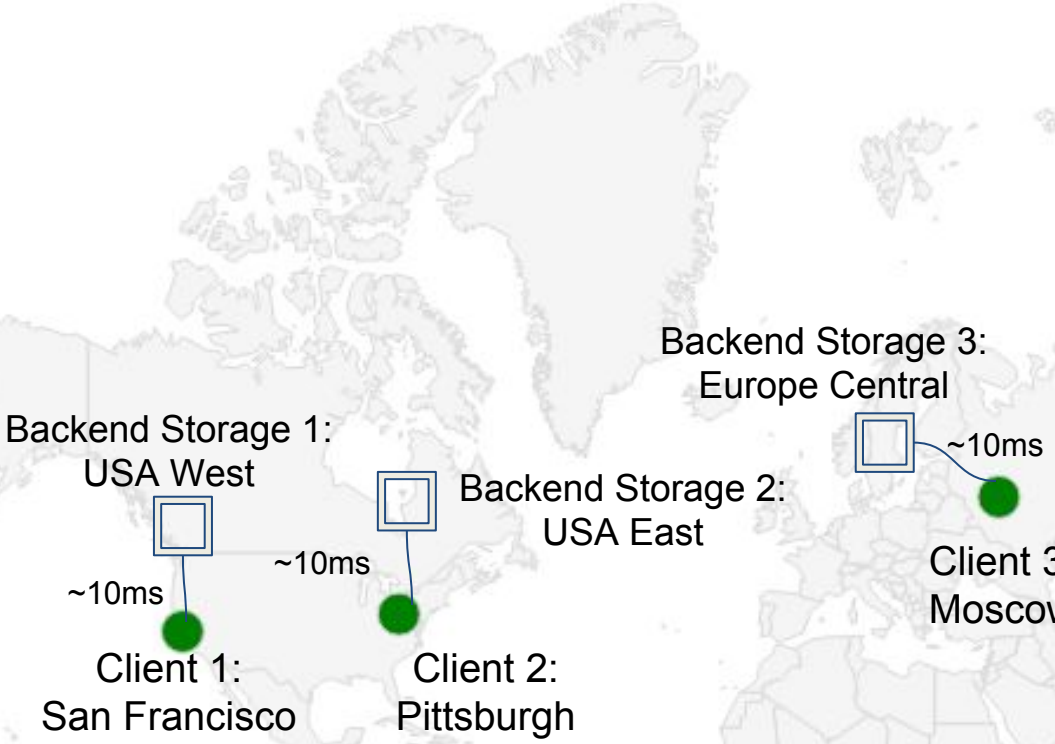


Client Statistics:
Min Latency: ??
Max Latency: ??
Average Latency: ??

Cost: ?????

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

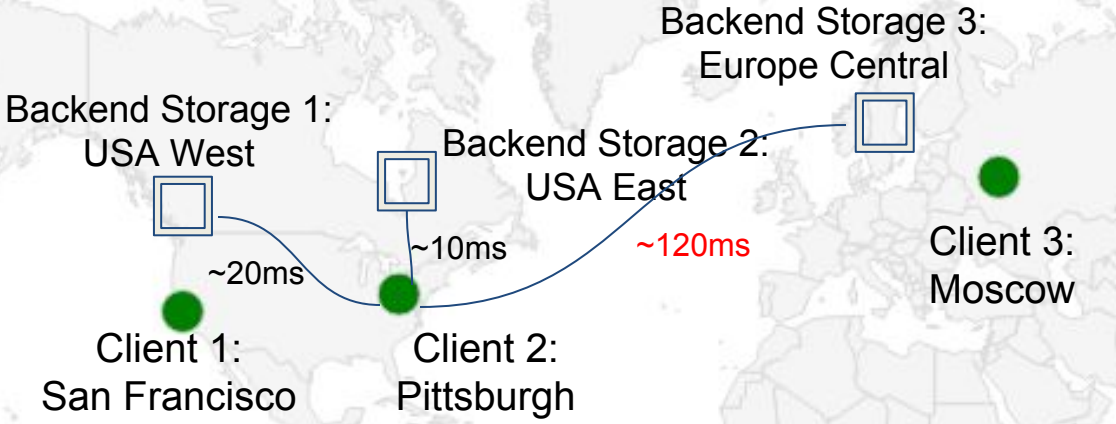
Replication READ



Read Operation:

Min Latency: 10ms
Max Latency: 10ms
Average Latency: 10ms

Replication WRITE



Write Operation:

Latency for Client 2 = MAX(10ms, 20ms, 120ms) = 120ms

Same across all clients

Even worse if the operations block each other!

Replication Reads and Writes

- Read requests are very fast!
  - All clients have a replica close to them to access
- Write requests are quite slow
  - Instead of updating a single data center, write requests must now update all 3 replicas
  - If multiple write requests for a certain key, then they all have to wait for each other to complete

Pros and Cons of Replication

- Duplicate the data across multiple instances
- Advantages
  - Fetching data can be faster
  - Fetching of “hot” data can be load balanced
  - Data can be retrieved from any datastore
  - System can handle failures of nodes
- Disadvantages
  - Requires more storage capacity
  - Updates are slower
  - Changes must reflect on all datastores

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
  - The order in which the values are updated must be preserved
- **Bonus:** Implement Eventual Consistency

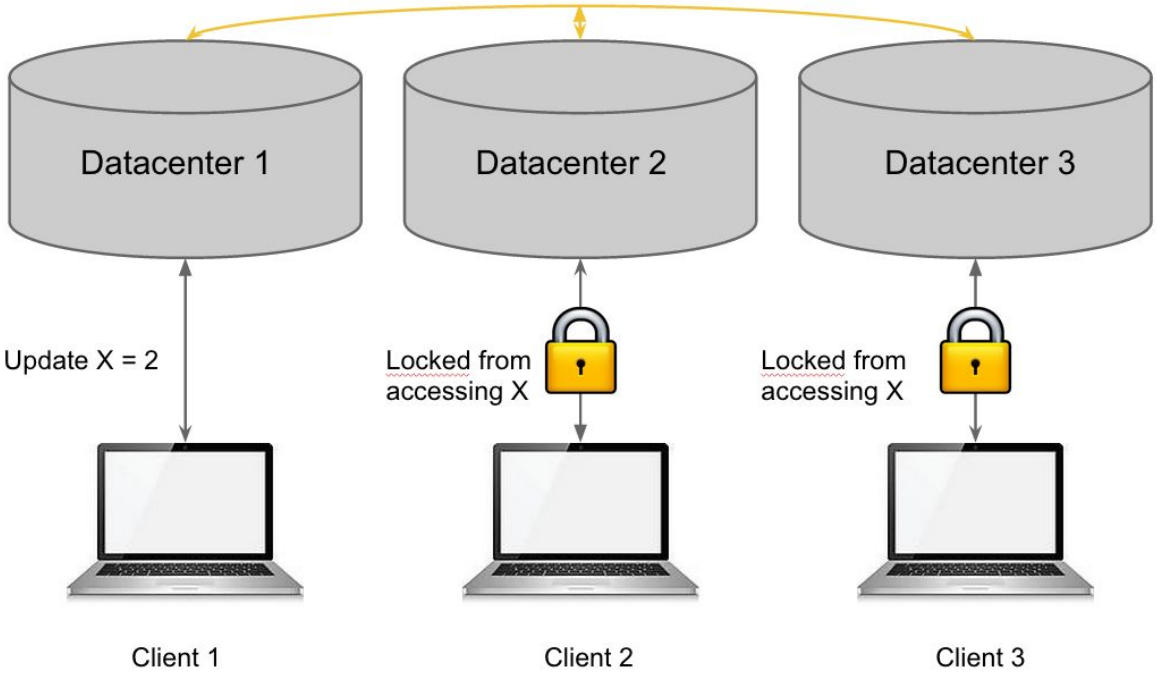
P3.3 Task 1: Strong Consistency

Single PUT request for key ‘X’

- Block all GET for key ‘X’ until all datastores are updated
- GET requests for a different key ‘Y’ must be allowed

Multiple PUT requests for ‘X’

- Resolved in order of their arrival
- Any GET request in between 2 PUTs must return the last PUT value

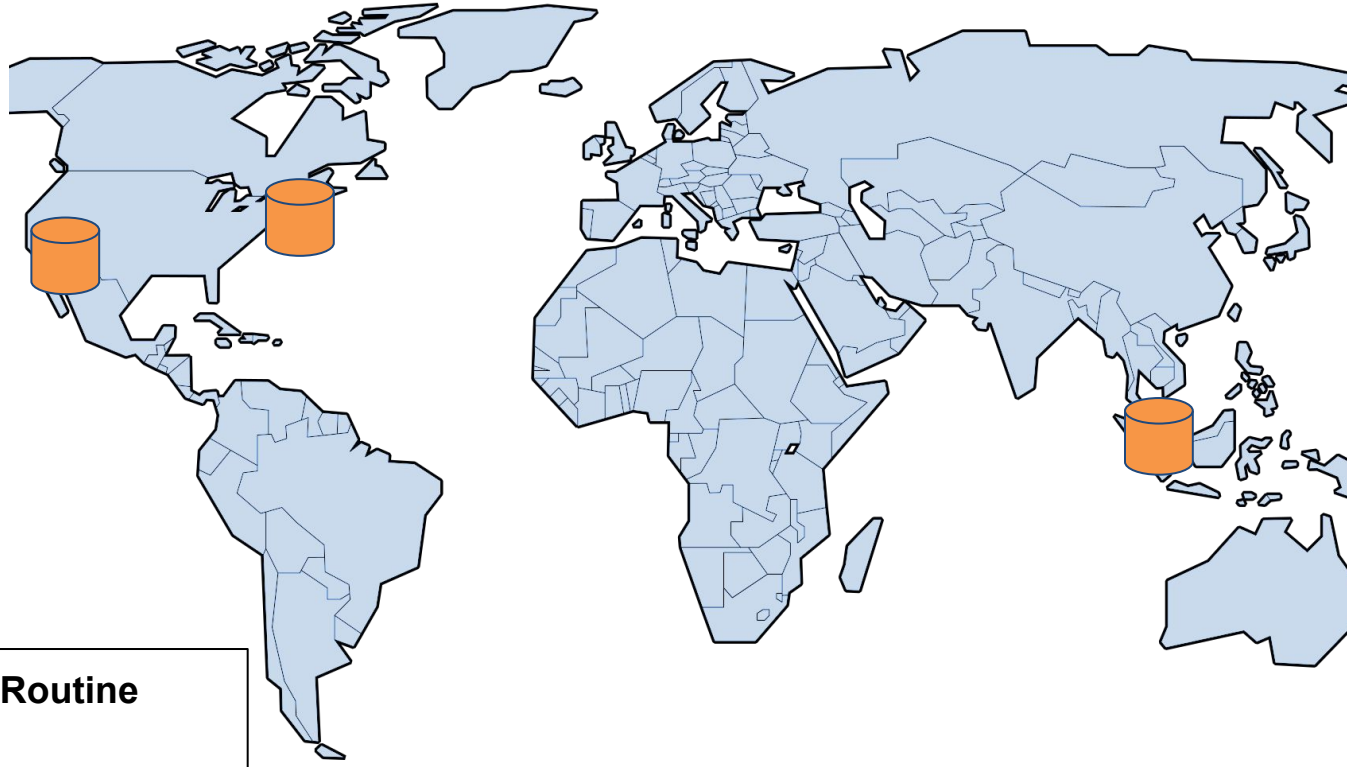


P3.3 Tasks 1 & 2: Strong Consistency

- Every request has a global timestamp order.
  - Typically the order in which they arrive at the coordinator
- 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



Withdrawal Routine

```
if(amt < balance):
  bal = bal - amt
  return amt
else:
  return 0
```

Account	Balance
xxxxx-4437	$100

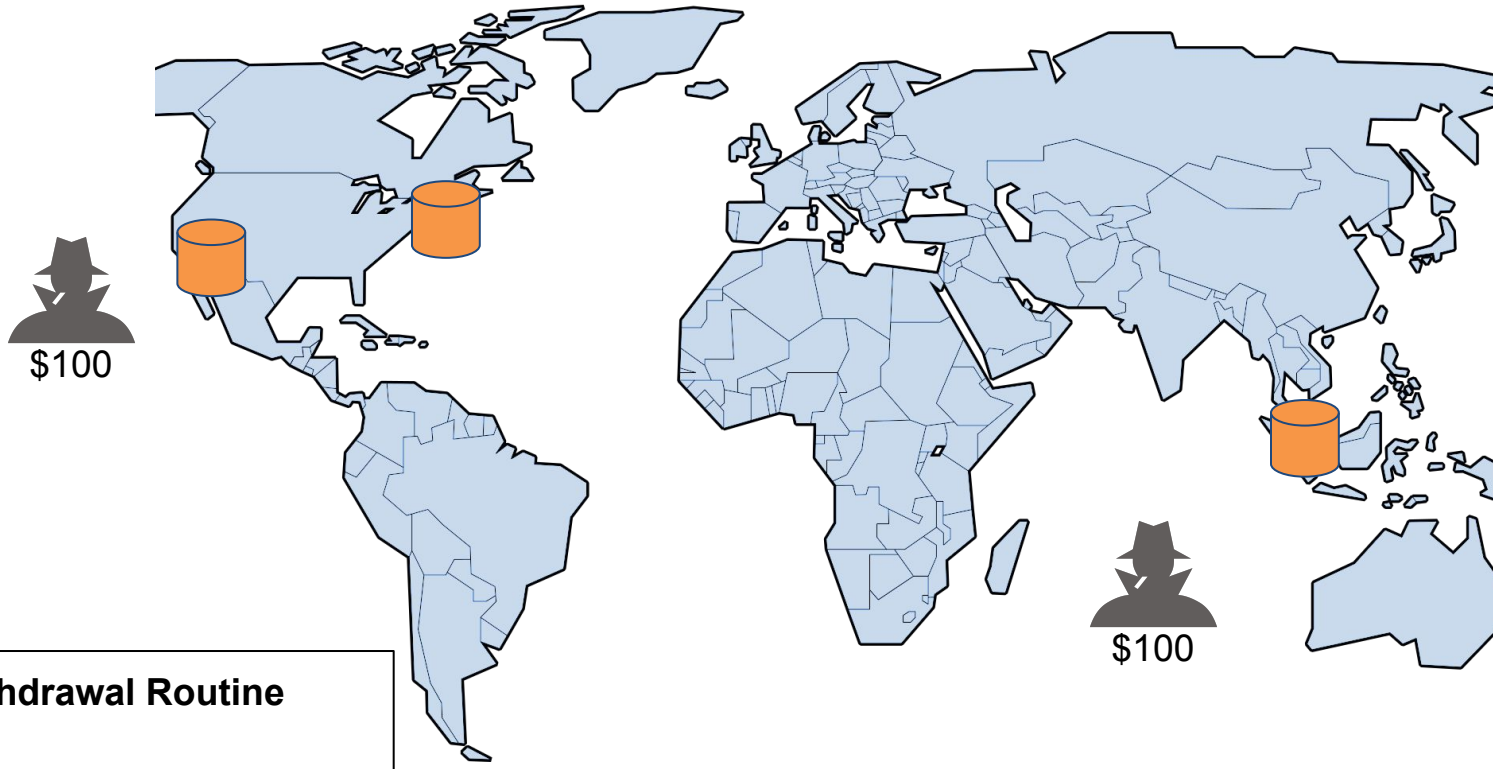
Choosing a Consistency Level
Bad Example

Withdrawal Routine

```
if(amt < balance):
  bal = bal - amt
  return amt
else:
  return 0
```

Account	Balance
xxxxx-4437	$100

Choosing a Consistency Level
Bad Example



Withdrawal Routine

```
if(amt < balance):
  bal = bal - amt
  return amt
else:
  return 0
```

Account	Balance
xxxxx-4437	**$0**

Bank lost $100

Choosing a Consistency Level
Good Example

Withdrawal Routine

```
lock(balance)
if(amt< balance):
  bal = bal - amt
  return amt
else:
  return 0
unlock(balance)
```

Account	Balance
xxxxx-4437	**$100**

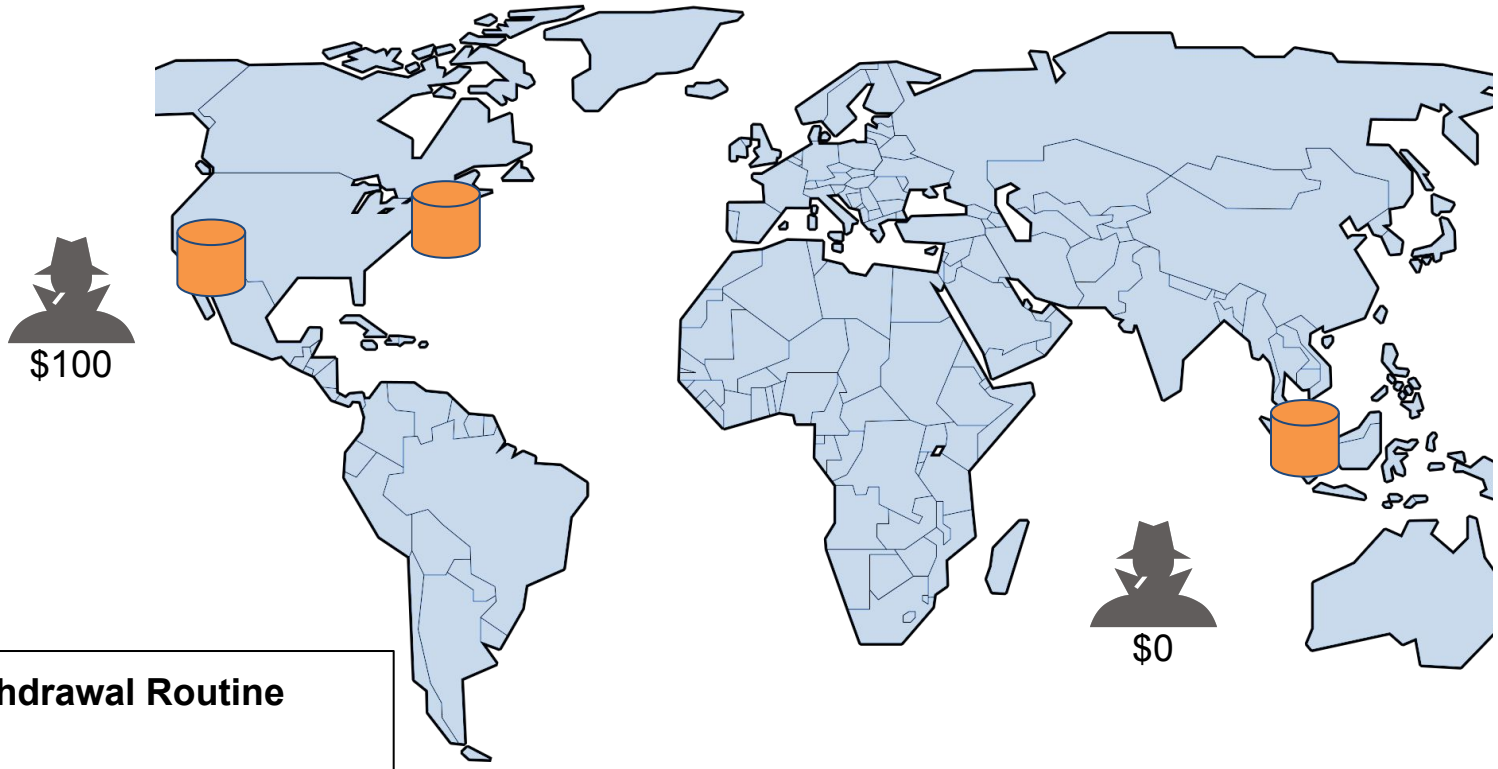
Choosing a Consistency Level
Good Example

Withdrawal Routine

```
lock(balance)
if(amt< balance):
  bal = bal - amt
  return amt
else:
  return 0
unlock(balance)
```

Account	Balance
xxxxx-4437	**$100**

Choosing a Consistency Level
Good Example

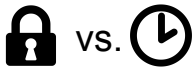


Withdrawal Routine

```
lock(balance)
if(amt< balance):
  bal = bal - amt
  return amt
else:
  return 0
unlock(balance)
```

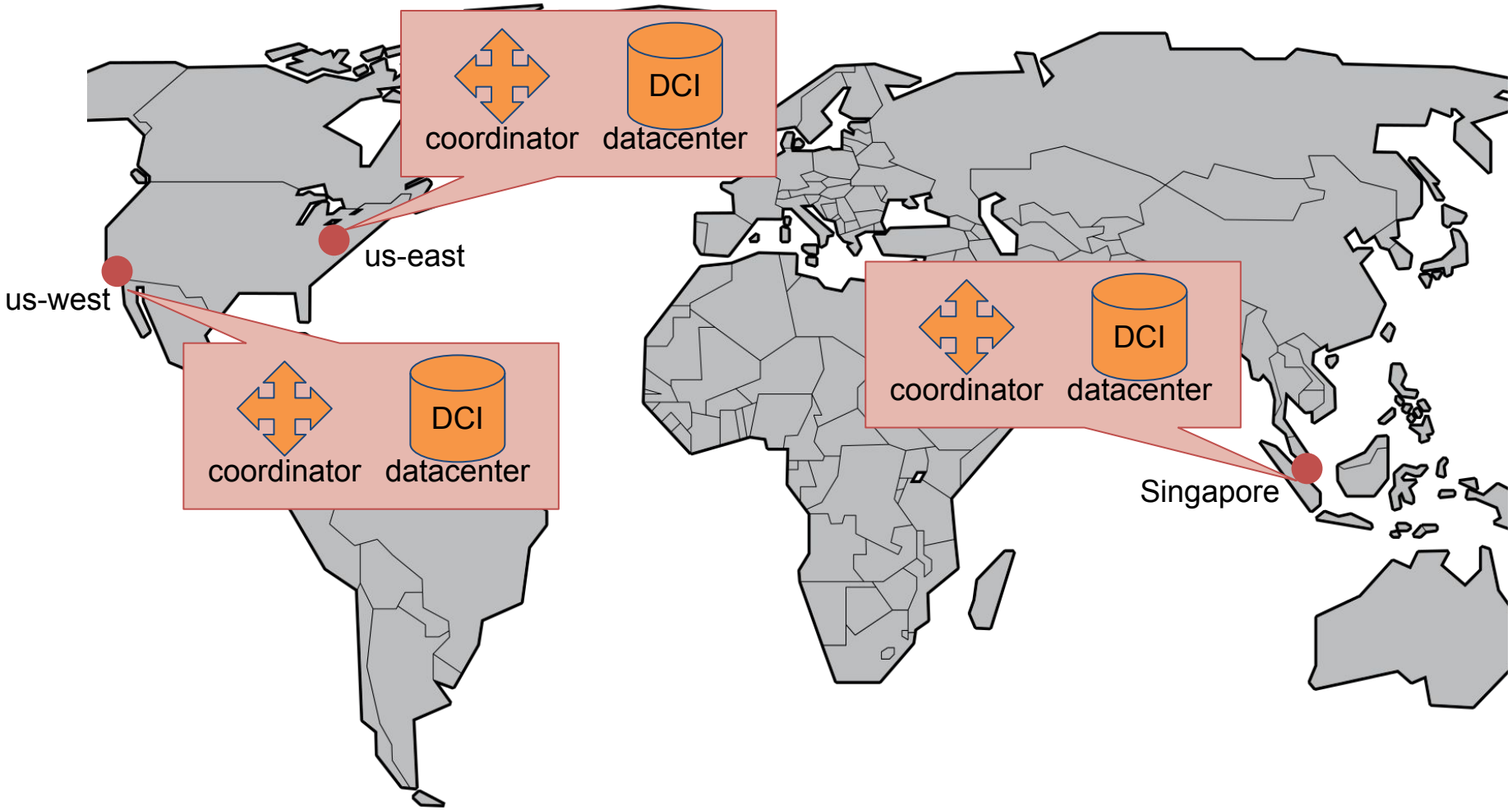
Account	Balance
xxxxx-4437	**$0**

P3.3: Consistency Models

Tradeoff: 

- Strict
- Strong
- Sequential
- Causal
- Eventual

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



P3.3: Eventual Consistency (Bonus)

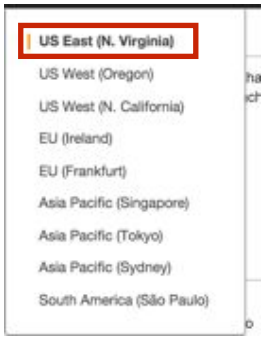
- Write requests are performed in the order received by the coordinator
  - Operations may not be blocked for replica consensus
- Clients that request data may receive multiple versions of the data, or stale data
  - Problems left for the application owner to resolve

P3.3: Tasks

- Launch Coordinators and DCs All in **us-east**
  - We'll simulate global latencies for you
- Implement the Coordinators and Datastores
  - Strong Consistency
    - Tasks 1 & 2
  - Eventual Consistency
    - bonus

- 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.

US-EAST DATACENTER (*KeyValueStore.java*)

US-WEST DATACENTER (*KeyValueStore.java*)

SINGAPORE DATACENTER (*KeyValueStore.java*)

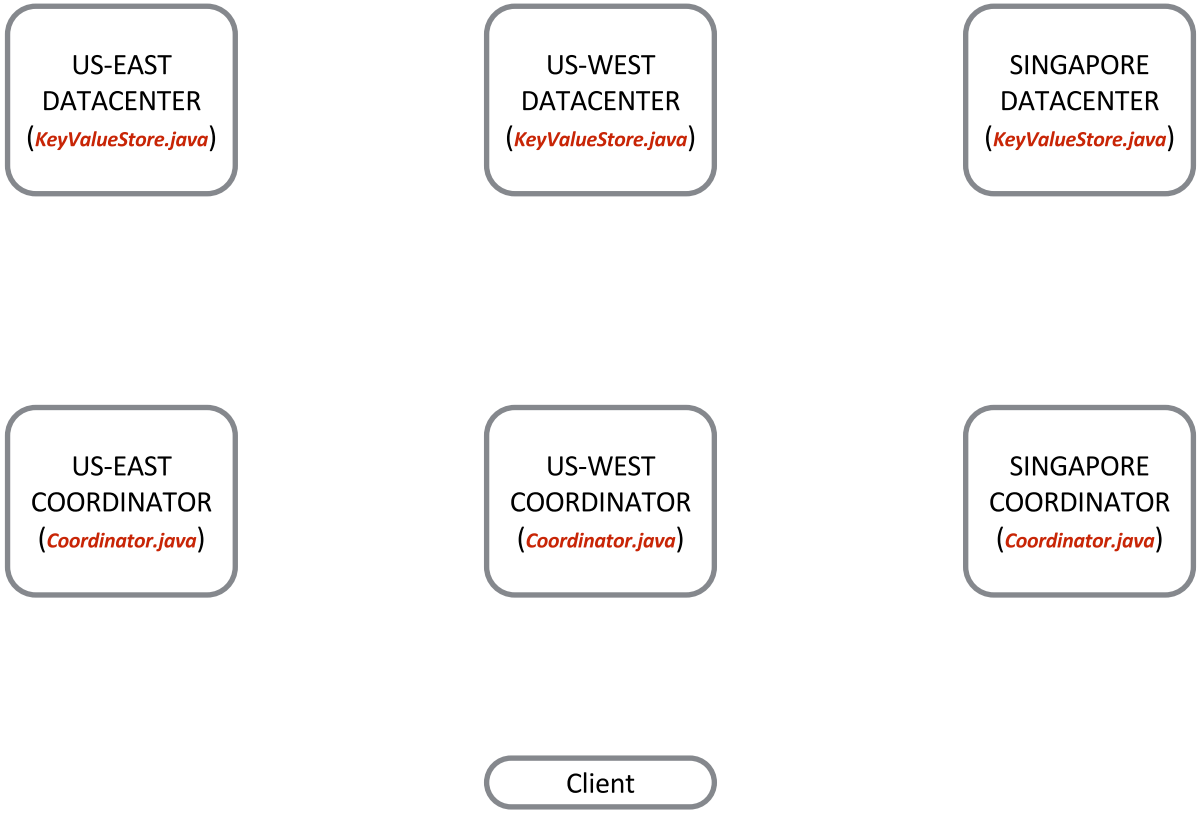
US-EAST COORDINATOR (*Coordinator.java*)

US-WEST COORDINATOR (*Coordinator.java*)

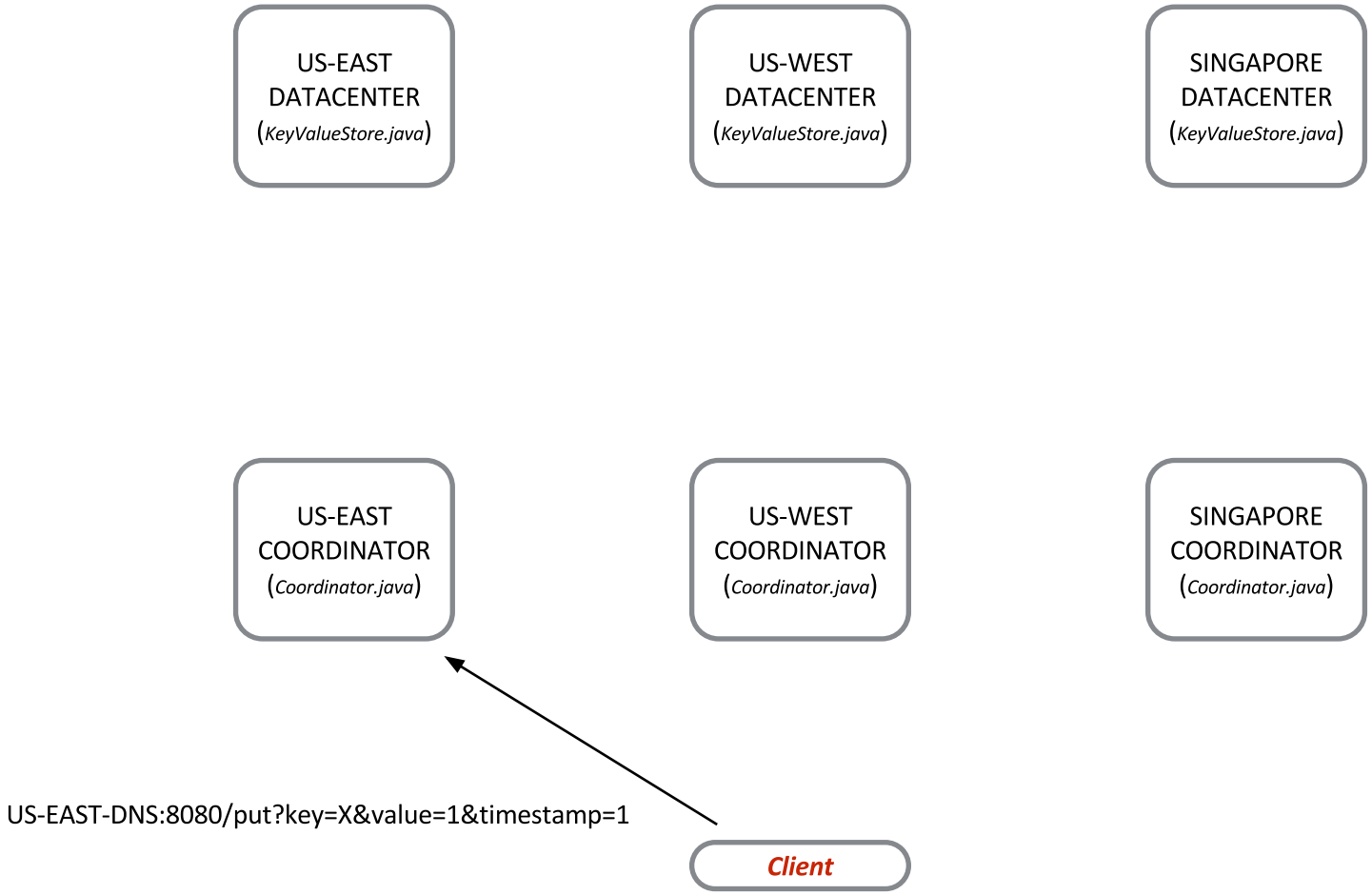
SINGAPORE COORDINATOR (*Coordinator.java*)

Client

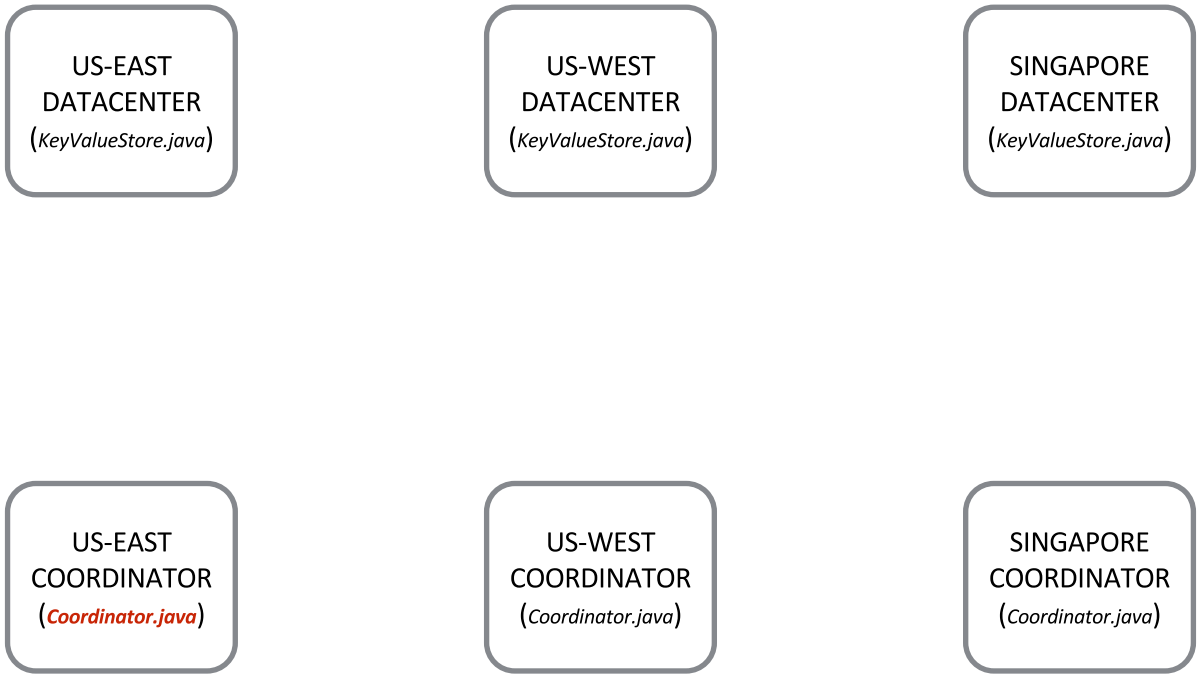
P3.3 Tasks:
Complete KeyValueStore.java (on DCs) and Coordinator.java (on Coordinators)



Example workflow for a PUT request using strong consistency

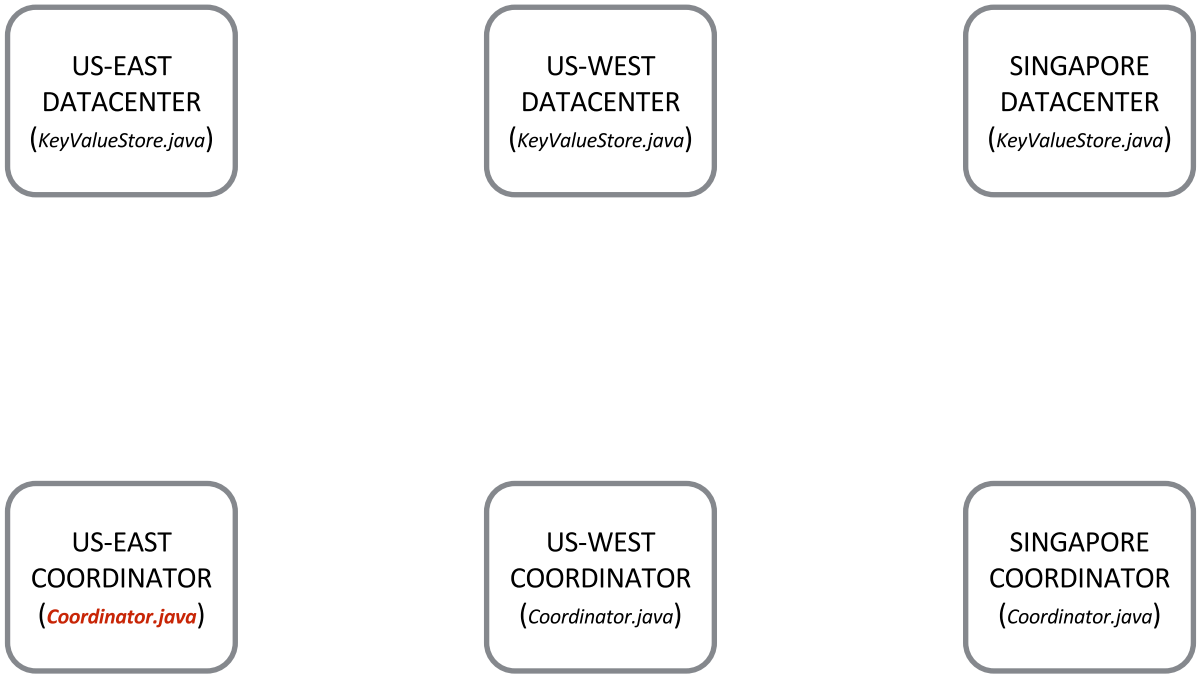


Example workflow for a PUT request using strong consistency



Example workflow for a PUT request using strong consistency

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



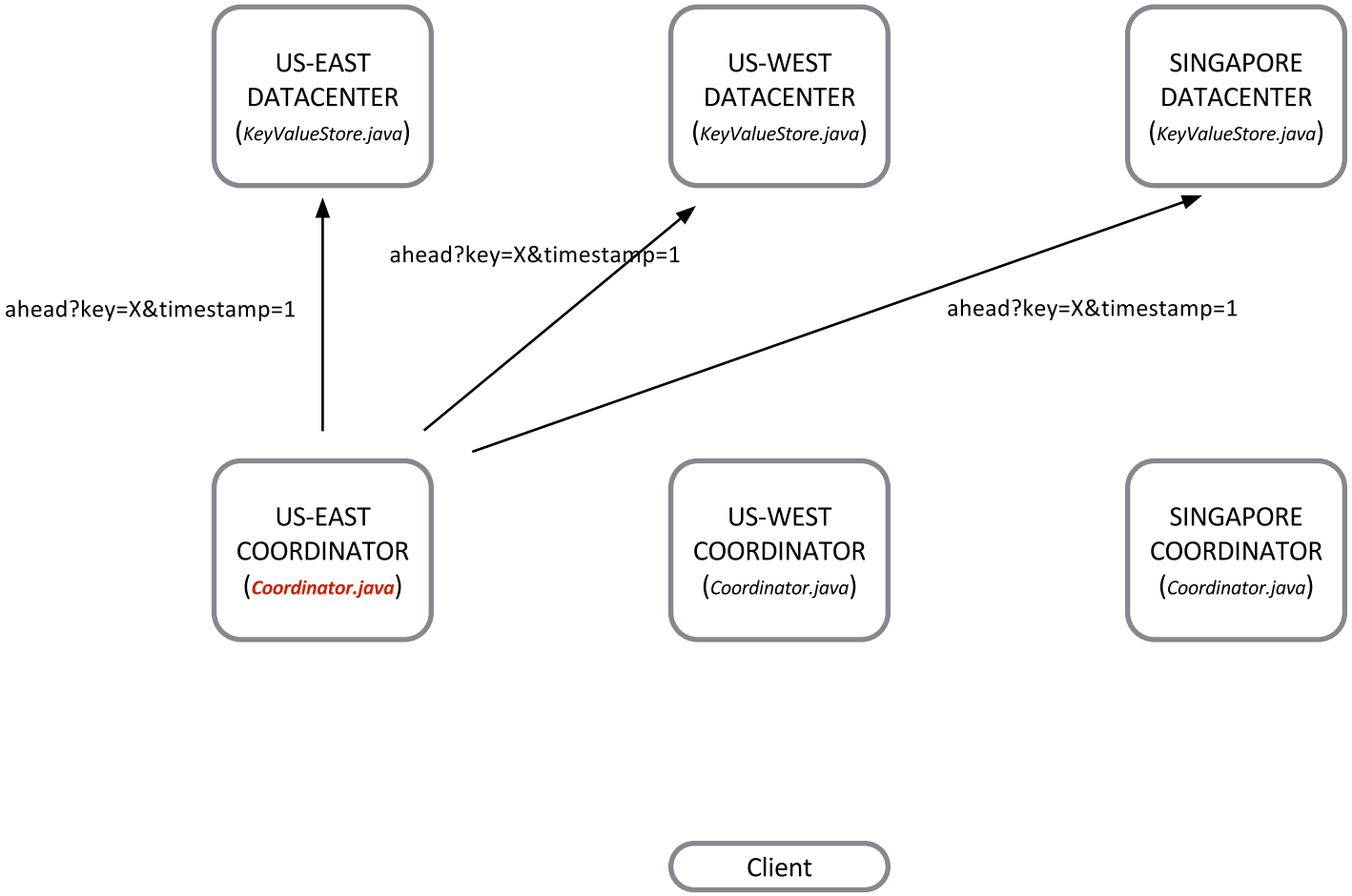
You should call KeyValueLib.AHEAD(“X”,1) to notify all 3 datacenters of this PUT request.
Resulting behavior may include:

- Locking subsequent requests for key “X” until current request is complete
- May be done on datacenter, coordinator, or a combination of both. Up to design

Client

Example workflow for a PUT request using strong consistency

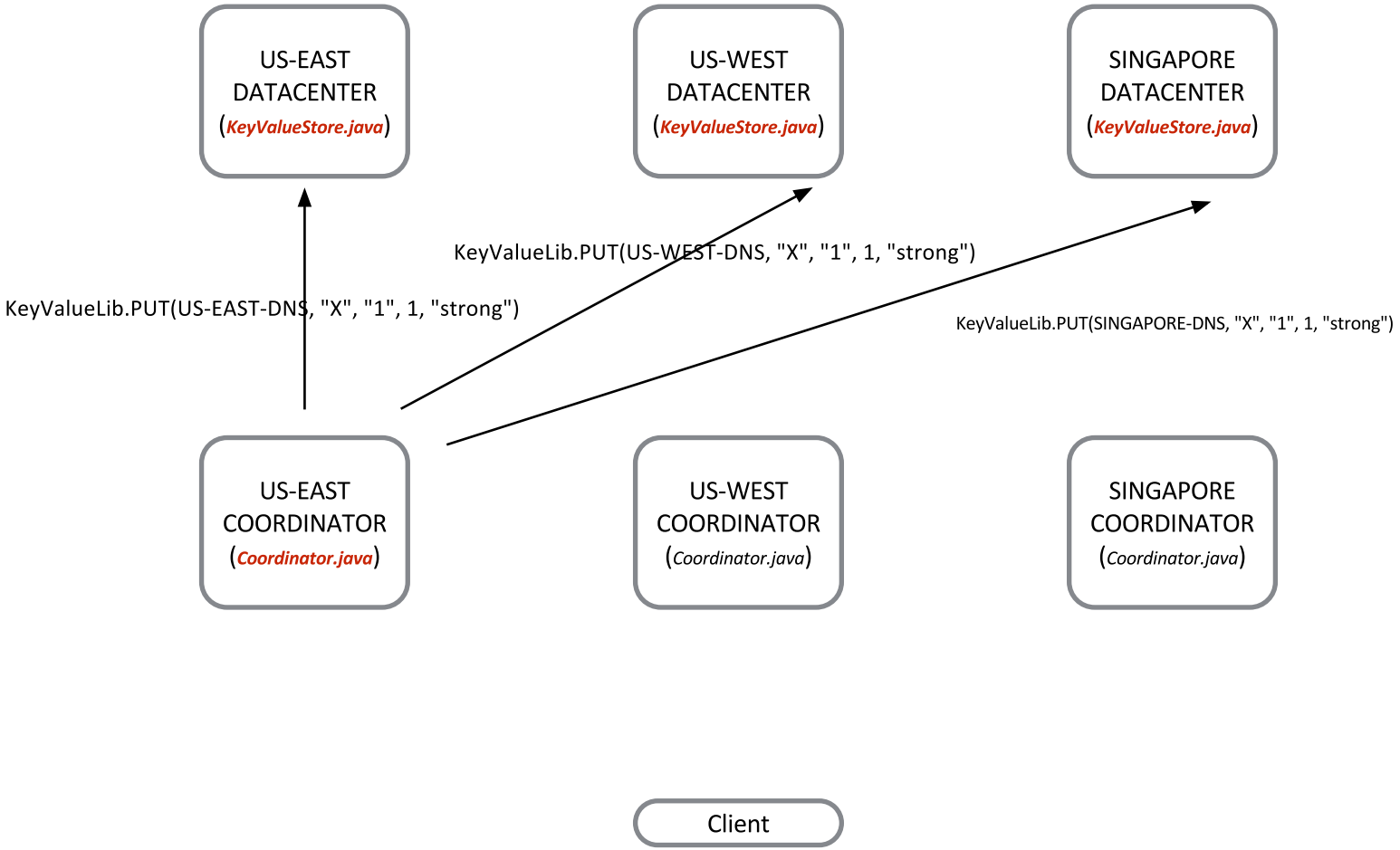
- If US-EAST is responsible for key “X”



Example workflow for a PUT request using strong consistency

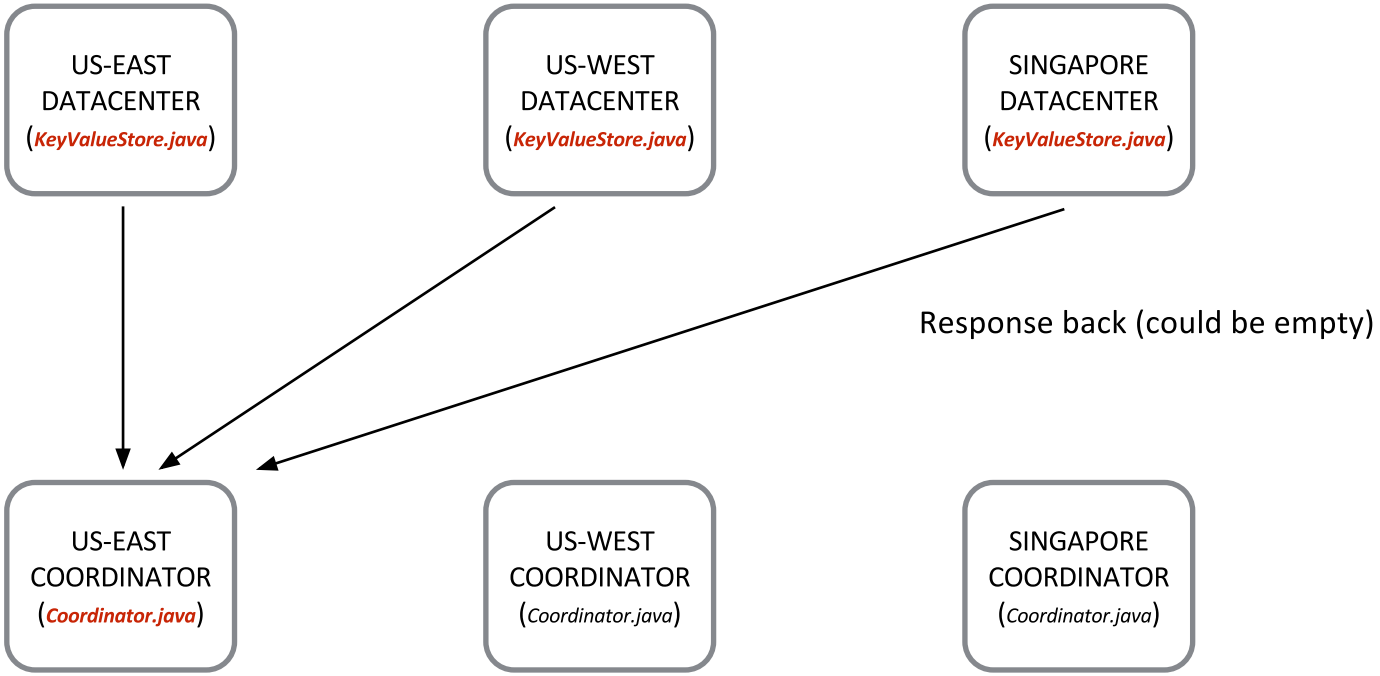
- If US-EAST is responsible for key “X”

Upon receiving the actual request, it will be up to you to decide how and when to update the value. Timestamps are extremely important in this project, so you may choose to store more than just the value associated with each key only for backend purposes.



Example workflow for a PUT request using strong consistency

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



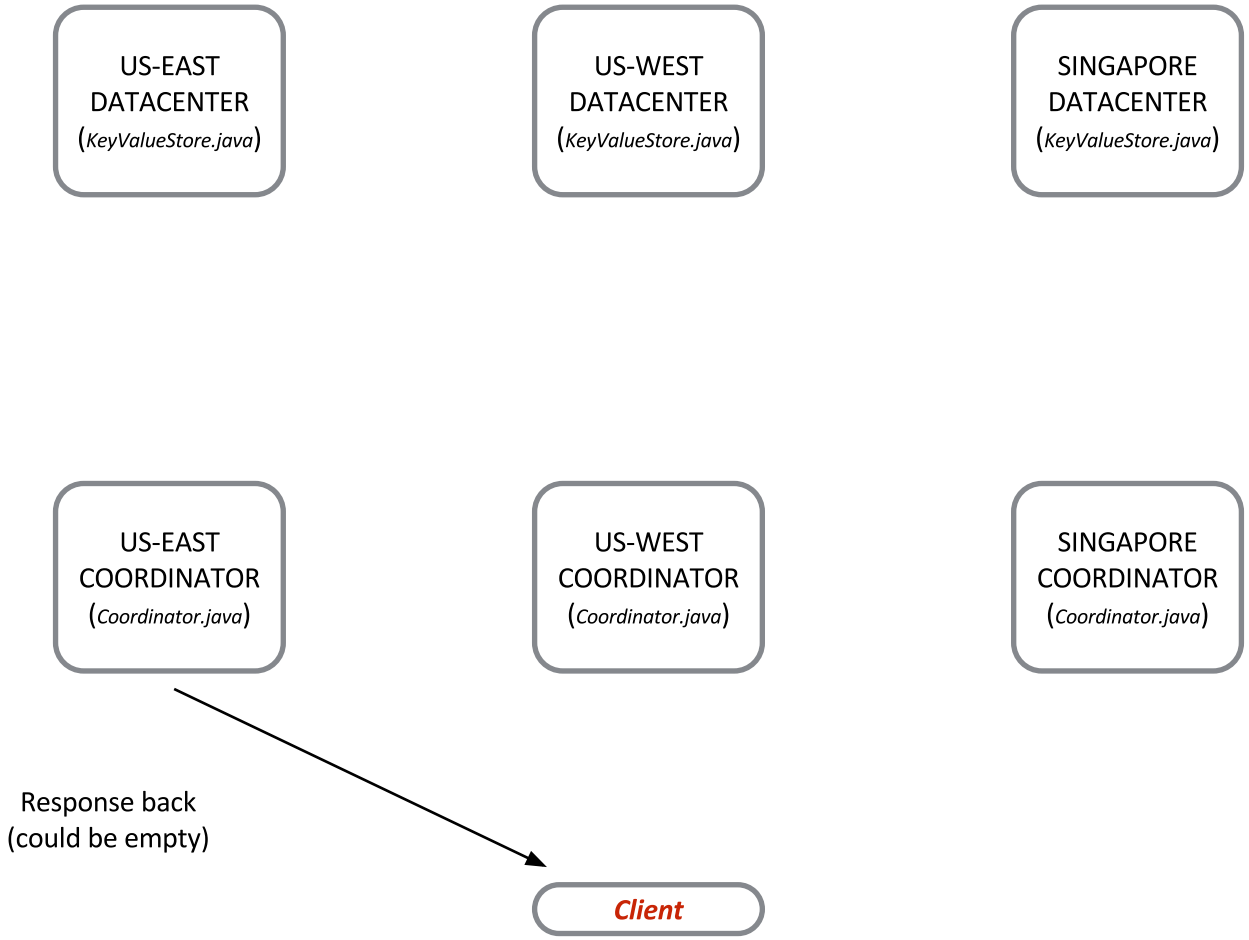
Finally, you should call KeyValueLib.COMPLETE(“X”,1) to notify all 3 datacenters of this request’s completion.
Resulting behavior may include:

- Allowing subsequent requests to proceed
- Allow pending requests to be completed (beware the timestamp ordering!)

Client

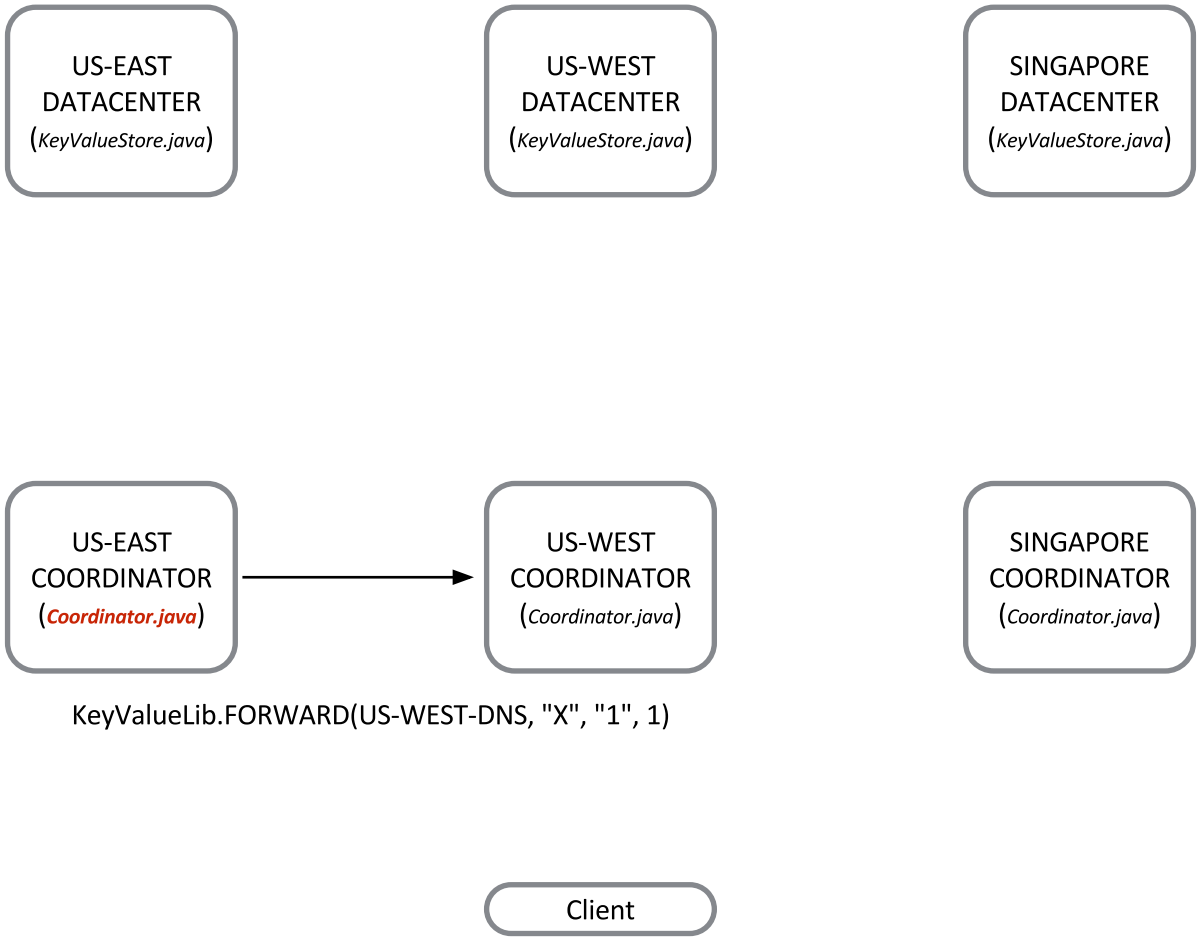
Example workflow for a PUT request using strong consistency

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



Example workflow for a PUT request using strong consistency

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



More Hints

- In strong consistency, “AHEAD” and “COMPLETE” should be useful to help you lock requests because they are able to communicate with datastores with negligible delay, regardless of region
- Lock all datacenters in strong consistency
- Eventual consistency is significantly easier to implement

Suggestions

- Read the primers.
- You should consider the differences between the 2 policies before writing your code.
- Think about possible race conditions.
- Read the hints in the writeup **carefully**.
- Don’t modify any class except Coordinator.java and KeyValueStore.java.

However...

- **In the Real World, there is nothing like AHEAD or COMPLETE with negligible latency.**
  - You may want to enroll in a distributed systems course (like 15440/640) to learn about consistency in detail.

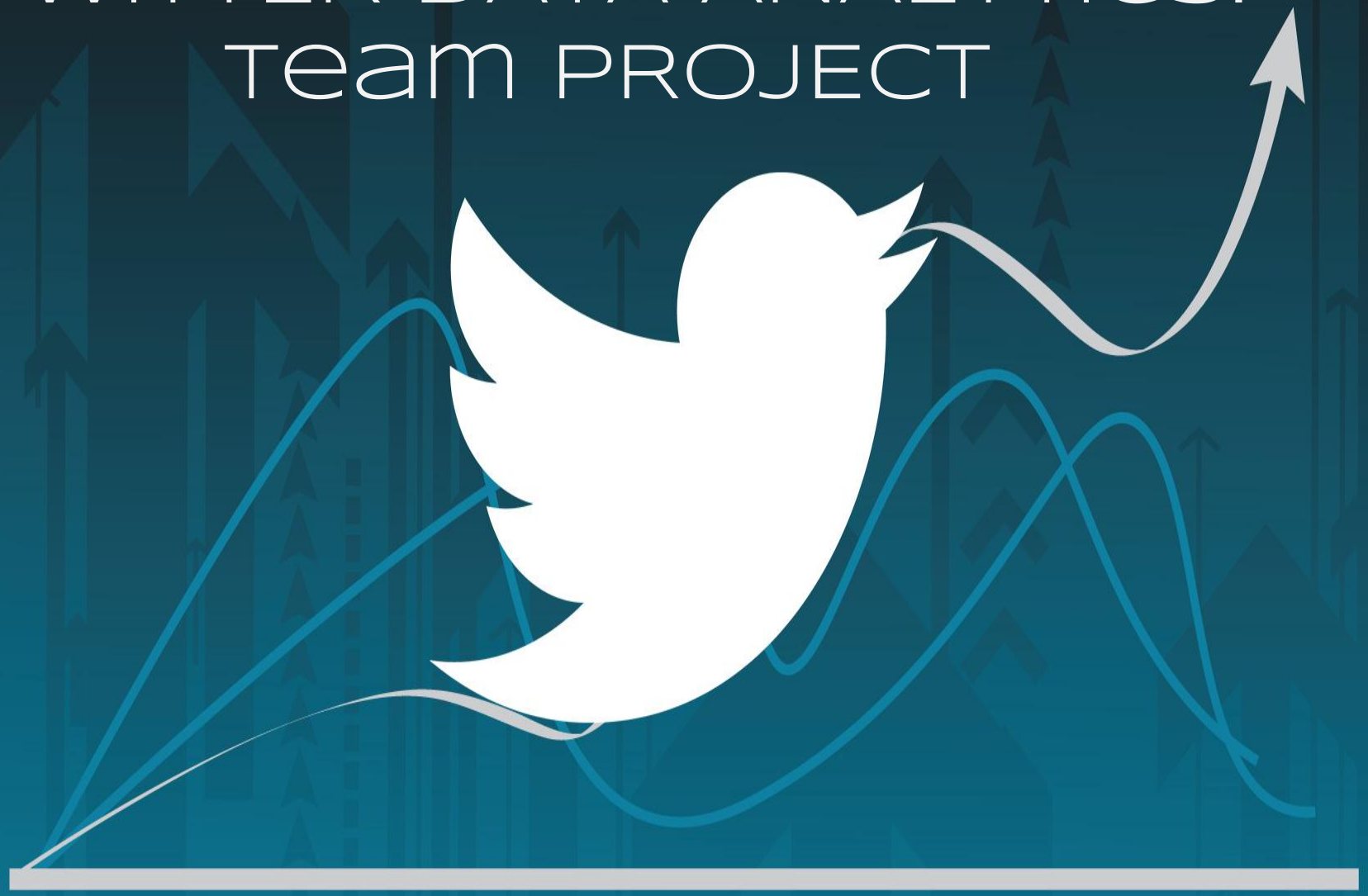
How to Run Your Program

- Run “./vertx run Coordinator.java” and “./vertx run KeyValueStore.java” to start the vertx server on each of the data centers and coordinators. (You could use nohup to run it in background)
- 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 enter the request in your browser.

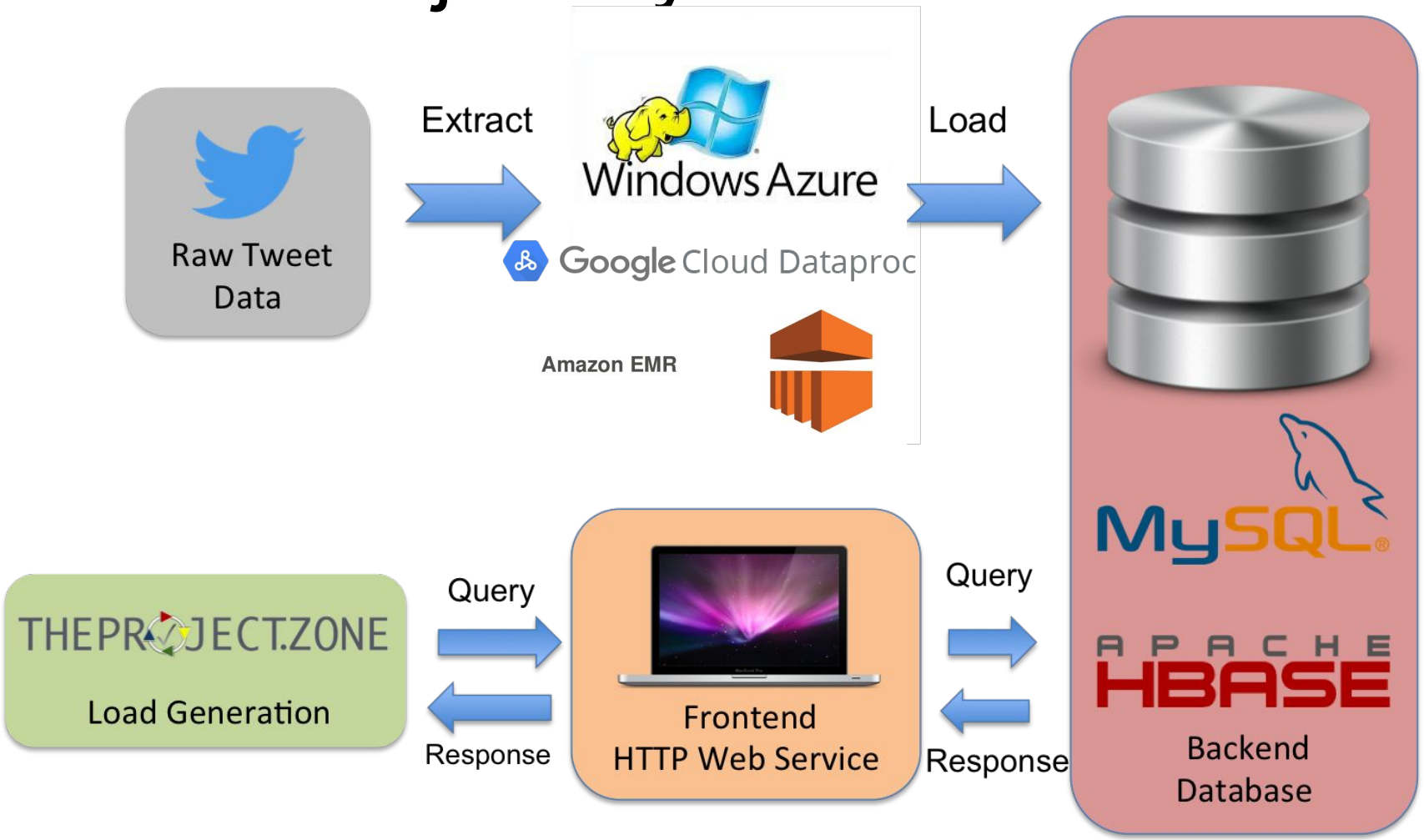
Start early!

Trickiest Individual Project!

TWITTER DATA ANALYTICS: Team PROJECT



Team Project System Architecture



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

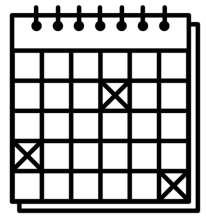
Team Project

- Phase 1:
  - Q1
  - Q2 & Q3 (MySQL **AND** HBase)

- Phase 2
  - Q1
  - Q2 & Q3 & Q4 (MySQL **AND** HBase)

- Phase 3
  - Q1
  - Q2 & Q3 & Q4 & Q5 (MySQL **OR** HBase)

Team Project Time Table



Phase	Query	Start	Deadline	Code and Report Due
Phase 1	Q1	Monday 2/27/2017 00:00:01 EST	Sunday 3/12/2017 23:59:59 EST	-
	Q2 & Q3	Monday 2/27/2017 00:00:01 EST	Sunday 4/2/2017 23:59:59 EST	Tuesday 4/4/2017 23:59:59 EST
Phase 2	Q1, Q2, Q3, Q4	Monday 4/3/2017 00:00:01 EST	Sunday 4/16/2017 15:59:59 EST	-
	Live Test Q1, Q2, Q3, Q4	Sunday 4/16/2017 18:00:01 ET	Sunday 4/16/2017 23:59:59 EST	Tuesday 4/18/2017 23:59:59 EST
Phase 3	Q1, Q2, Q3, Q4, Q5	Monday 4/17/2017 00:00:01 ET	Sunday 4/30/2017 15:59:59 EST	-
	Live Test Q1, Q2, Q3, Q4, Q5	Sunday 4/30/2017 18:00:01 ET	Sunday 4/30/2017 23:59:59 EST	Tuesday 5/2/2017 23:59:59 EST

Note:

- There will be a report due at the end of each phase, where you are expected to discuss optimizations
- **WARNING: Check your AWS instance limits on the new account (should be > 10 instances)**

Team Project Phase 1

- Three queries
  - Q1 Task:
    - Front end (web tier) development.
  - Q2 Tasks:
    - ETL + web tier + database tier,
    - **GET** query on both MySQL (Relational DBMS),
    - **GET** query on HBase (NoSQL).
  - Q3 Tasks:
    - ETL + web tier + database tier,
    - **RANGE GET** query on MySQL,
    - **RANGE GET** query on HBase.

Team Project Phase 1

- Grading, three queries
  - Every submission on theproject.zone will get
    - Error Rate
    - Correctness
    - RPS
  - Higher RPS, higher correctness, lower error rate ⇒ higher grade
  - Q1, Q2MySQL, Q2HBase, Q3MySQL, Q3HBase ⇒ 15% each * 5 = 75%
  - Phase 1 report = 25%

Query 1 Notes

- The auto deployment script
  - **Running Environment**
    - Need to be able to be runnable.
      - Your own laptop or AWS VMs
        - Specify in your report
  - **Restriction**
    - No restriction on how you actually launch your own testing environment. If you like to launch your server manually, it's your choice.
  - **Submitting**
    - Submit along with your code and report after Phase 1 is finished.

Team Project, Phase 1, Q2 & Q3

- **Step 1:** Extract tabular data from raw tweets
  - Input file: JSON Tweets (150GB, compressed)
  - Consider using a MapReduce Job for data processing
    - ETL is expensive and there's the potential for errors, so plan carefully, test on smaller data sets
    - Feel free to do ETL on Azure / GCP since they won't eat up your AWS budget ;)
    - Start early, or no time to optimize the backend
- **Step 2:** Load the data into **both** HBase **and** MySQL
- **Step 3:** Deploy
  - a web service for handling HTTP requests, responds with data from the backend
  - an optimized backend (MySQL and HBase)

Hints

- Be careful about encoding
  - Non-english characters
  - Emojis 😁
- Read the write up carefully on the ETL requirement
  - Use Regex to eliminate short URLs
  - Different stop word lists in Q2 & Q3
- ETL is expensive and time-consuming
  - Big data challenge will easily eat up your time and money if you are careless. Think, calculate, & test before you start doing ETL.

More Hints on Query 2

- Use regex “\p{L}+” to match words. **FOR Q2 ONLY.**
  - So that we only match the unicode letters. Simply ignore the unicode marks, i.e. “\p{M}”.
- If one hashtag appears in a tweet multiple times, the tweet should be weighted based on the frequency of this hashtag showing up.
  - Word frequency in this tweet should be calculated multiple times if it contains the hashtag more than once.
- Treat all contents in the `text` field the same, including hashtag. Meaning if the text contains hashtags, these hashtags can be considered as a word and need to be included in the keyword frequency calculation.
- Only hashtags in the request should be considered as case sensitive.

Reminder

- Your team has a total AWS budget of $50 for Phase 1
- Your web service should not cost more than $0.88 per hour, this includes (see write-ups for details):
  - Use EC2 on-demand instance cost
    - even if you use spot instances, we will calculate your cost using the on-demand instance price
  - EBS cost
  - ELB cost
  - We will not consider data transfer and EMR cost
- Targets:
  - Query 2 - 7000 rps (for both MySQL and HBase)
  - Query 3 - 1500 rps (for both MySQL and HBase)

Upcoming Deadlines

- Quiz 8: Unit 4 - Module 15

  Due: Firday 3/24/2017 11:59 PM EST

- Individual Project: P3.3, Replication and Consistency Models

  Due: 3/26/2017 11:59 PM EST.

- Team Project: Phase 1

  Due: 4/2/2017 11:59 PM EST