15-319 / 15-619
Cloud Computing

Recitation 8

October 17th, 2017

Overview

- **Last week's reflection**
  - Project 3.2 Done!
- **This week's schedule**
  - Quiz 7 – due on **Thursday, Oct 19th** (Module 14)
  - Project 3.3 – due on Sunday, Oct 22nd
- **Twitter Analytics: The Team Project Phase 1, Query 1 is due Sunday Oct 22! Hurry up!**

This Week: Conceptual Content

OLI UNIT 4: Cloud Storage

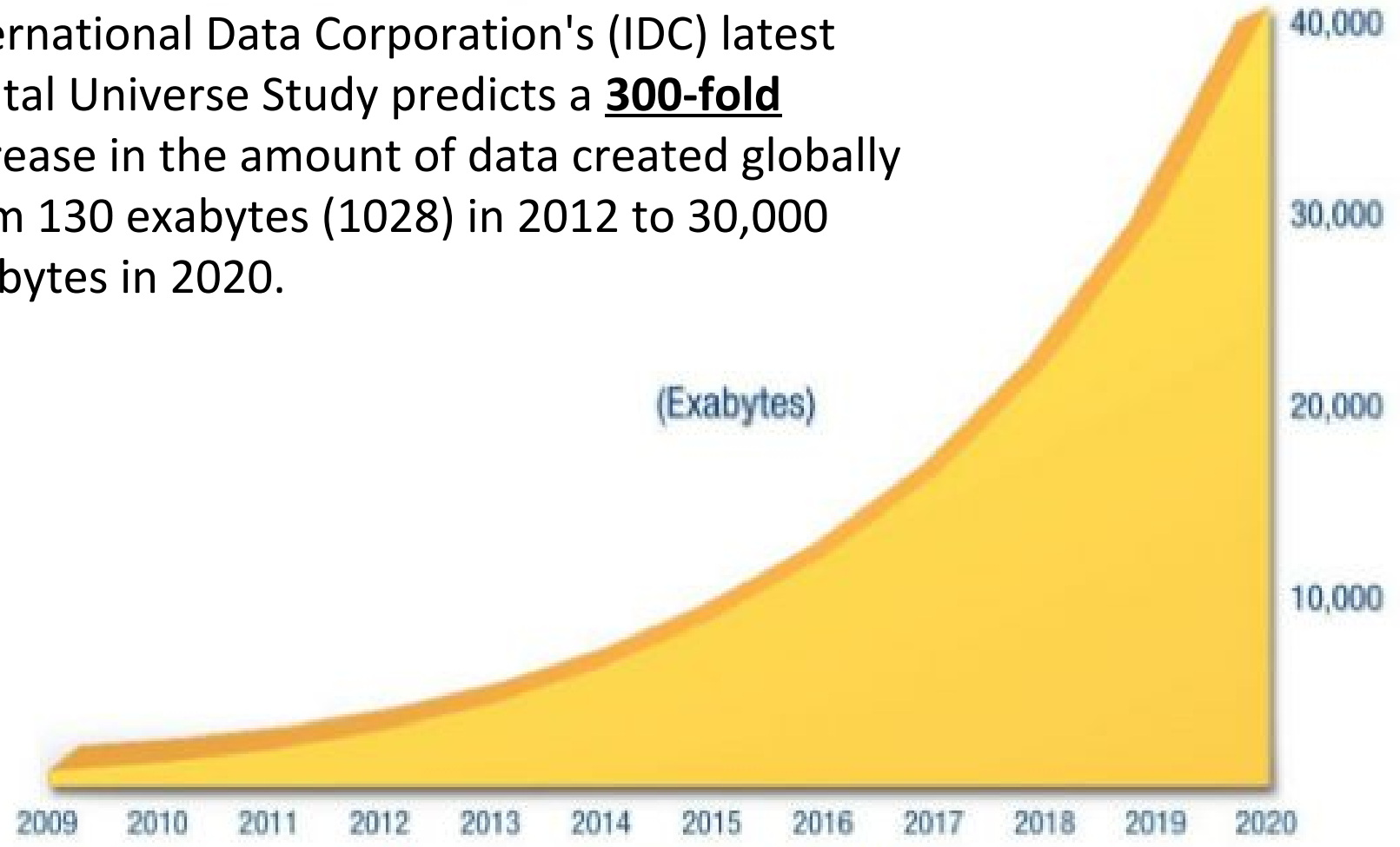
- **Module 14: Cloud Storage**
  - File Systems and Databases
  - Scalability and Consistency
  - NoSQL, NewSQL and Object Storage
- **Quiz 7**
  - **DUE on Thursday, October 19th**

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

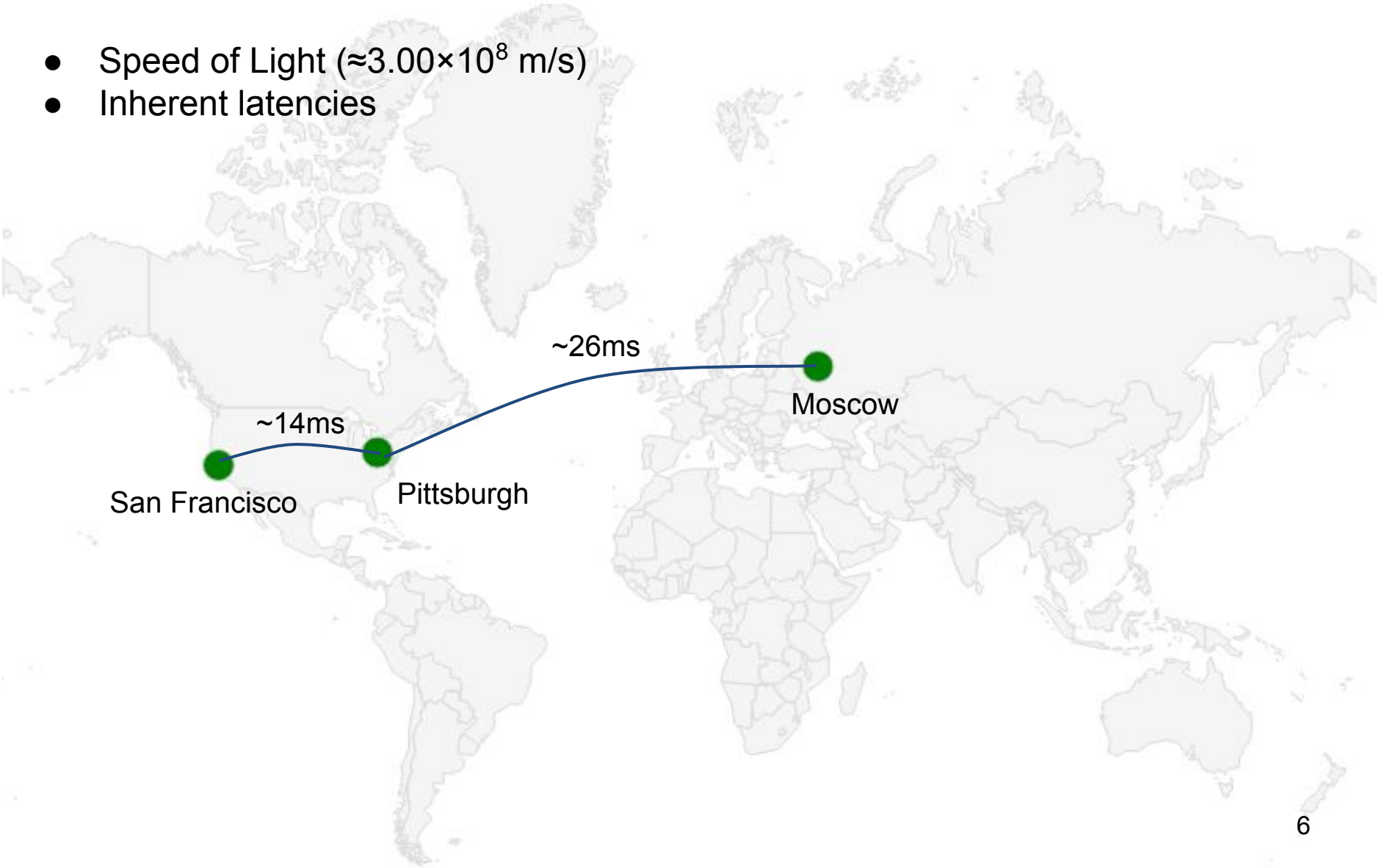
International Data Corporation's (IDC) latest Digital Universe Study predicts a **300-fold** increase in the amount of data created globally from 130 exabytes (1028) in 2012 to 30,000 exabytes in 2020.



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

Users are Global

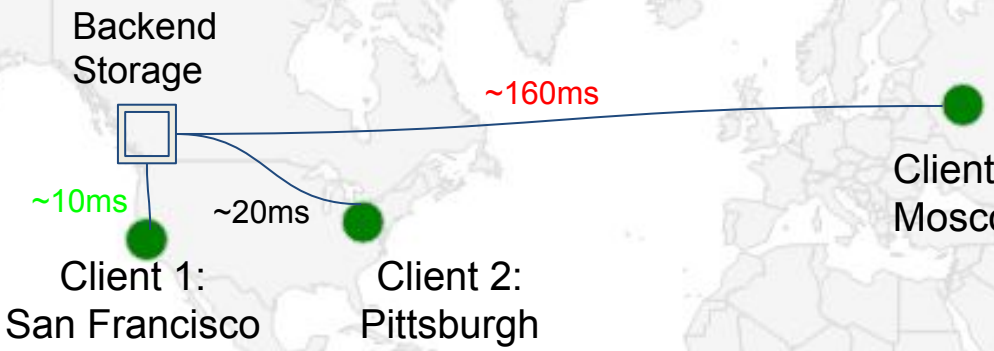
- Speed of Light (≈3.00×10^8 m/s)
- Inherent latencies



Typical End-To-End Latency

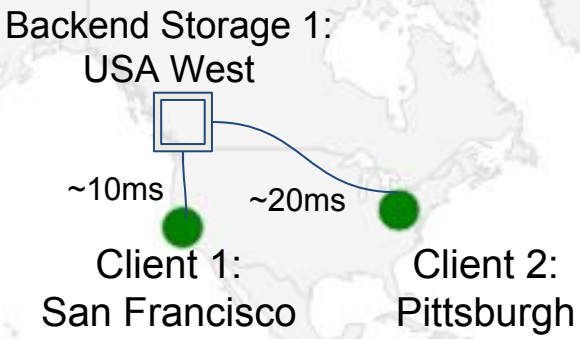
- Typical end-to-end latency
  - Network latency (from client to backend)
  - Server response
    - Includes fetching and processing data from backend
  - Network latency (from backend to client)

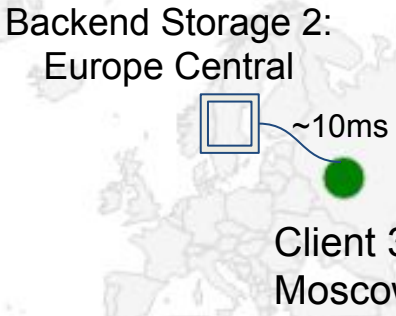
Latency with a Single Backend



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

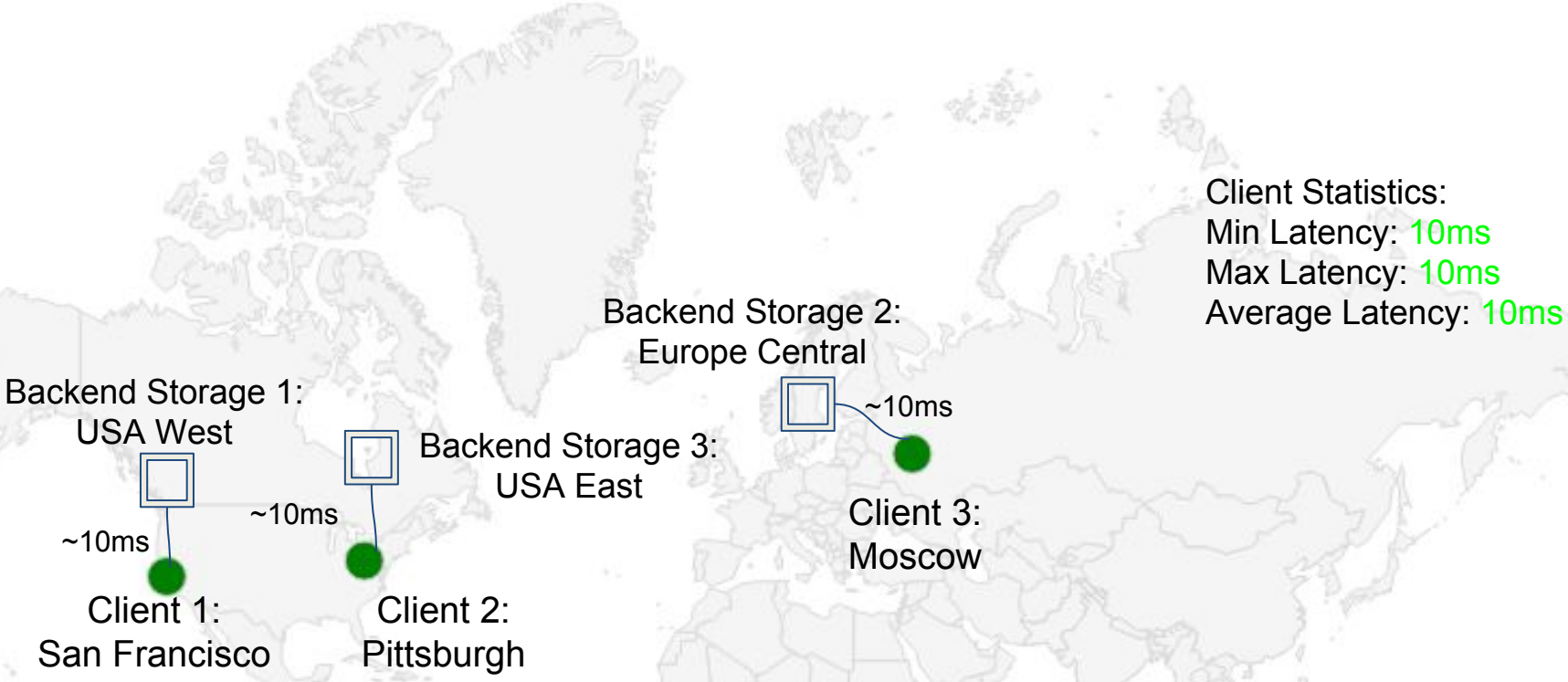
Replicate the Data Globally





Client Statistics:
Min Latency: 10ms
Max Latency: 20ms
Average Latency: 13.3ms

Replicate the Data Close to Users

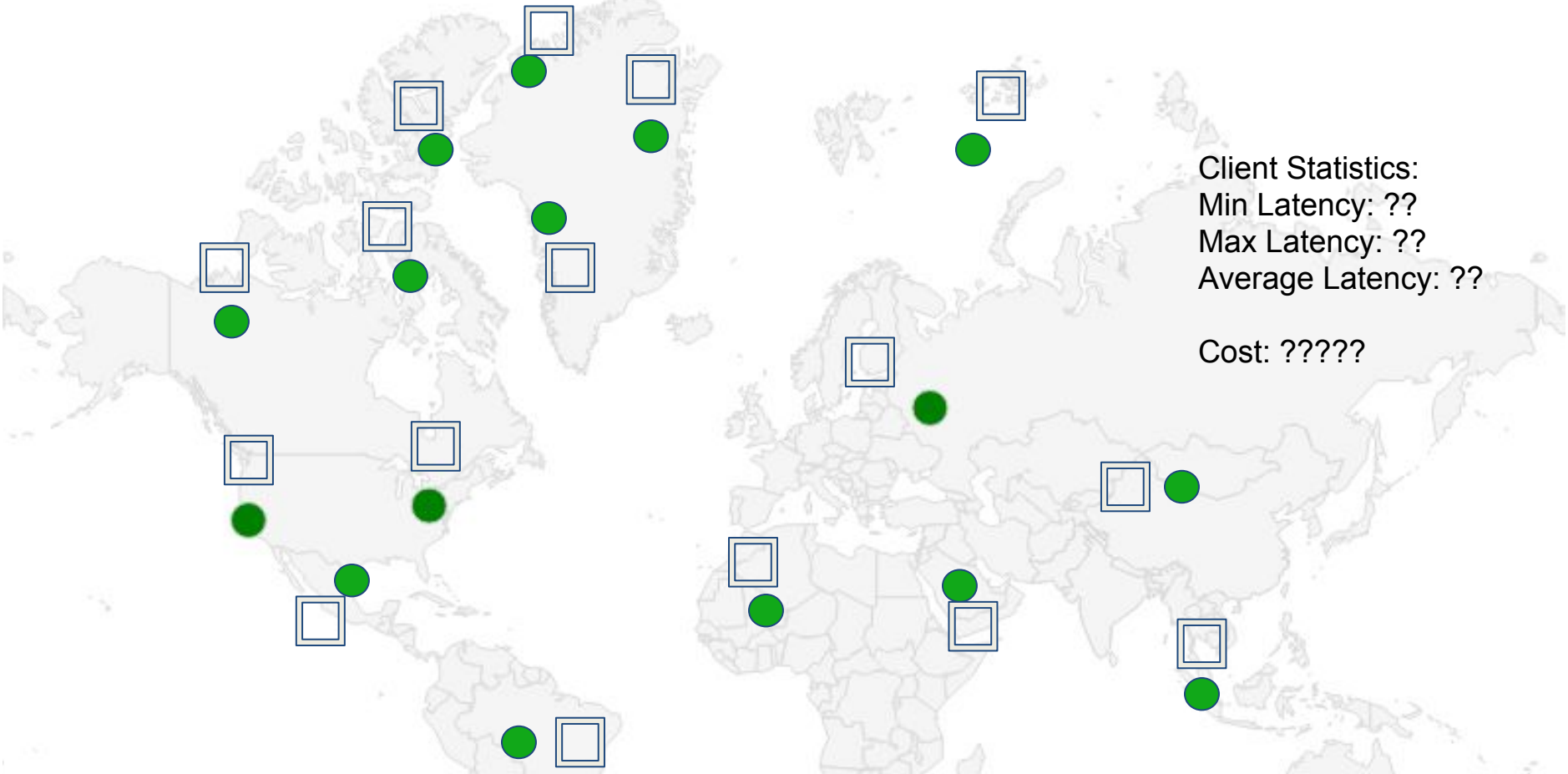


Client Statistics:
Min Latency: 10ms
Max Latency: 10ms
Average Latency: 10ms

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
- **Note**: Replicas also help in improving performance, availability and disaster recovery.
- But how about the cost of using replicas?

What If We Continue to Replicate?

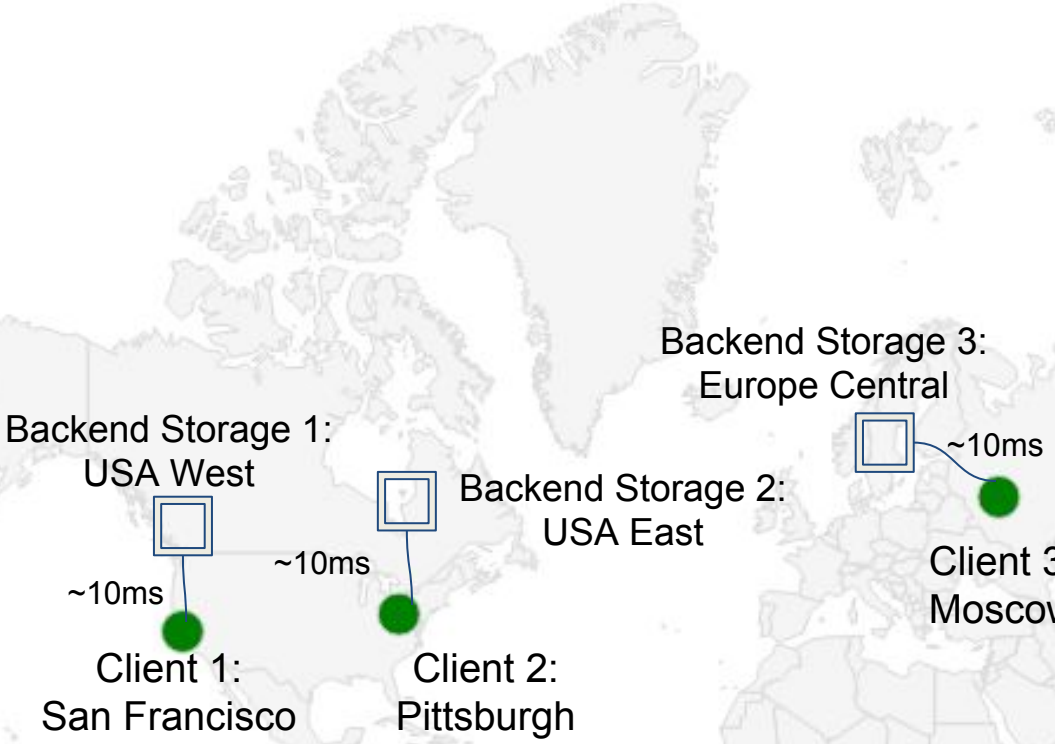


Client Statistics:
Min Latency: ??
Max Latency: ??
Average Latency: ??

Cost: ?????

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

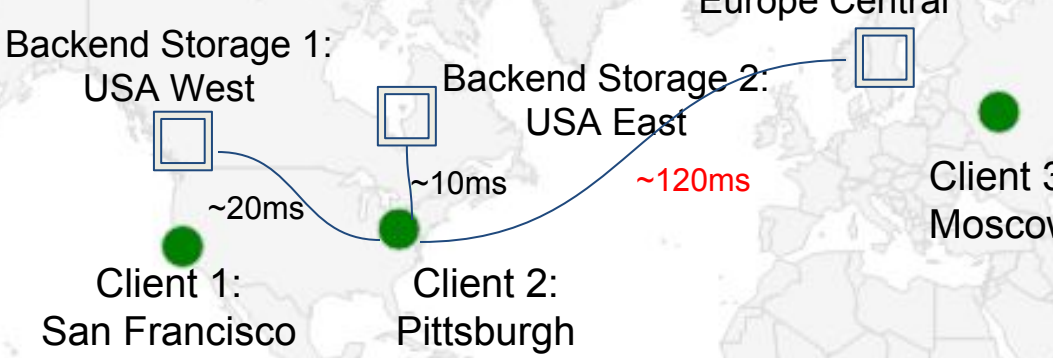
Replication READ



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

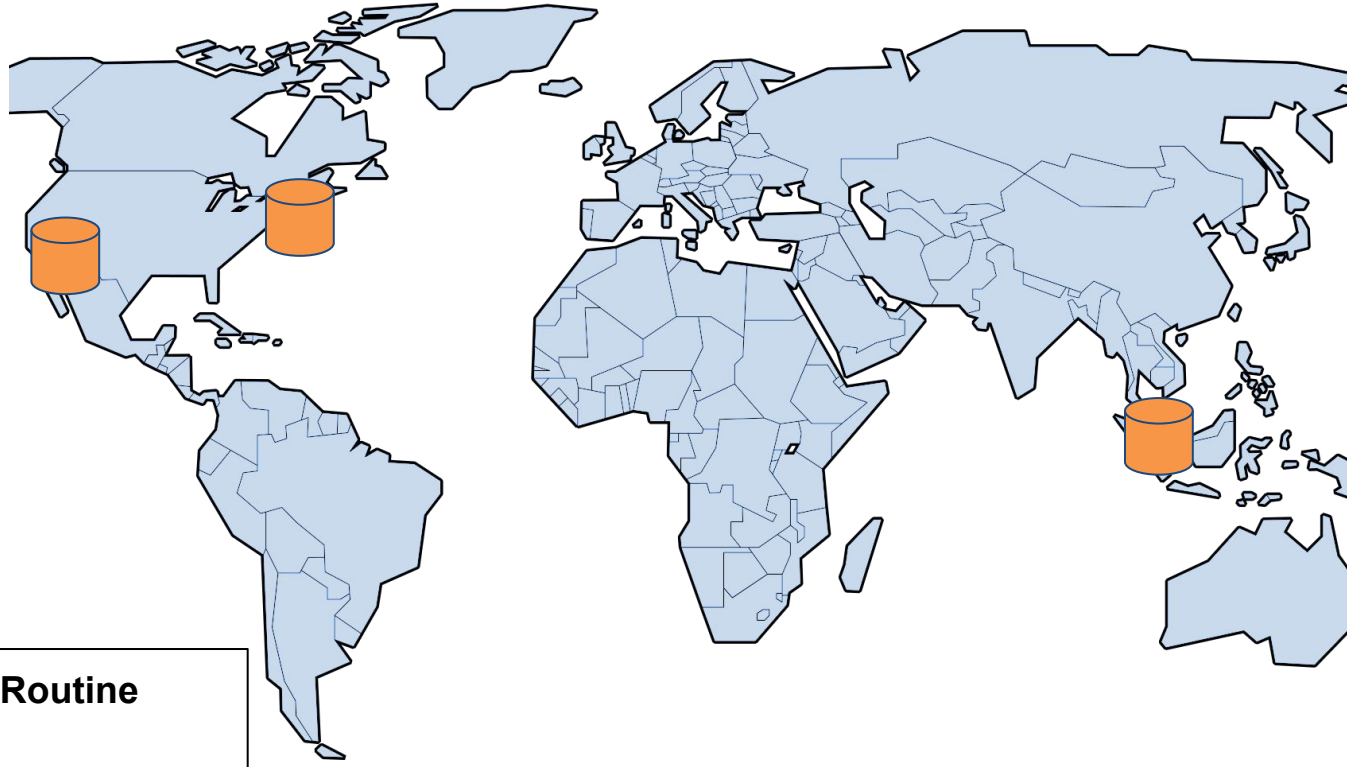
- Duplicate the data across multiple instances
- Advantages
  - Fetching data can be faster
  - Fetching of “hot” data can be load balanced
  - Data can be retrieved from any datastore
  - System can handle failures of nodes
- Disadvantages
  - Requires more storage capacity
  - Updates are slower
 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

Choosing a Consistency Level
Bad Example



Withdrawal Routine

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

Account	Balance
xxxxx-4437	$100

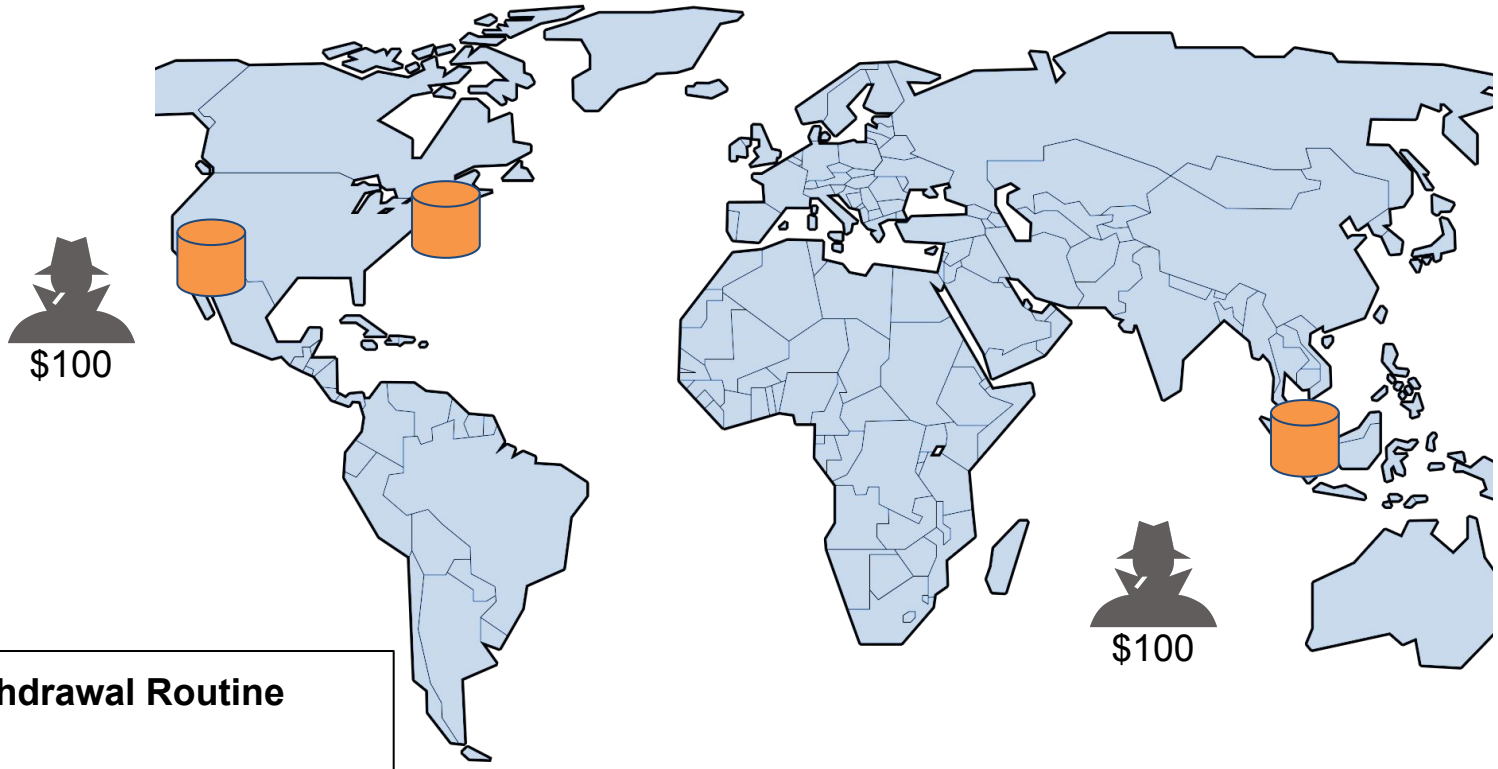
Choosing a Consistency Level
Bad Example

Withdrawal Routine

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

Account	Balance
xxxxx-4437	$100

Choosing a Consistency Level
Bad Example



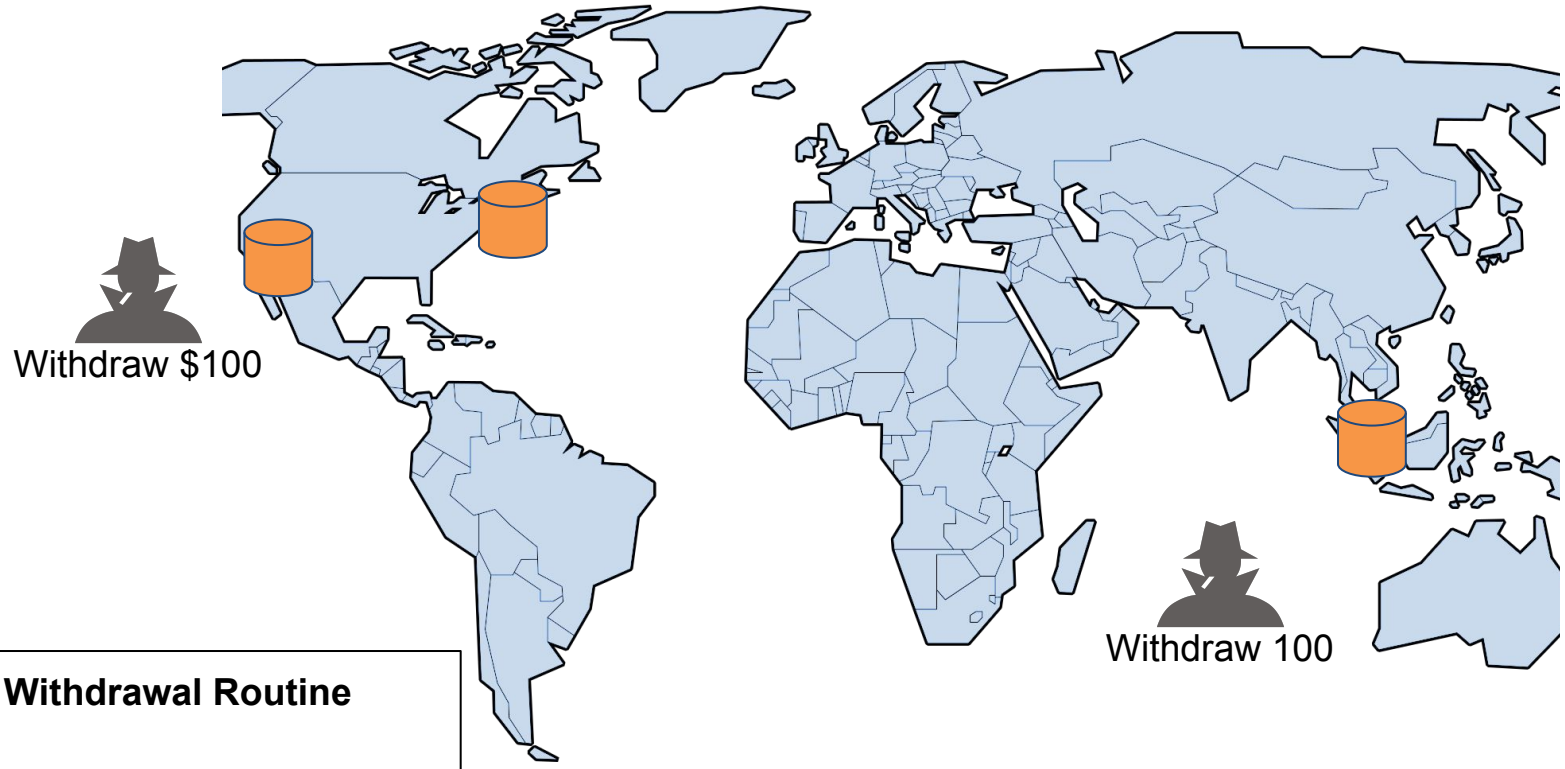
Withdrawal Routine

```
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**

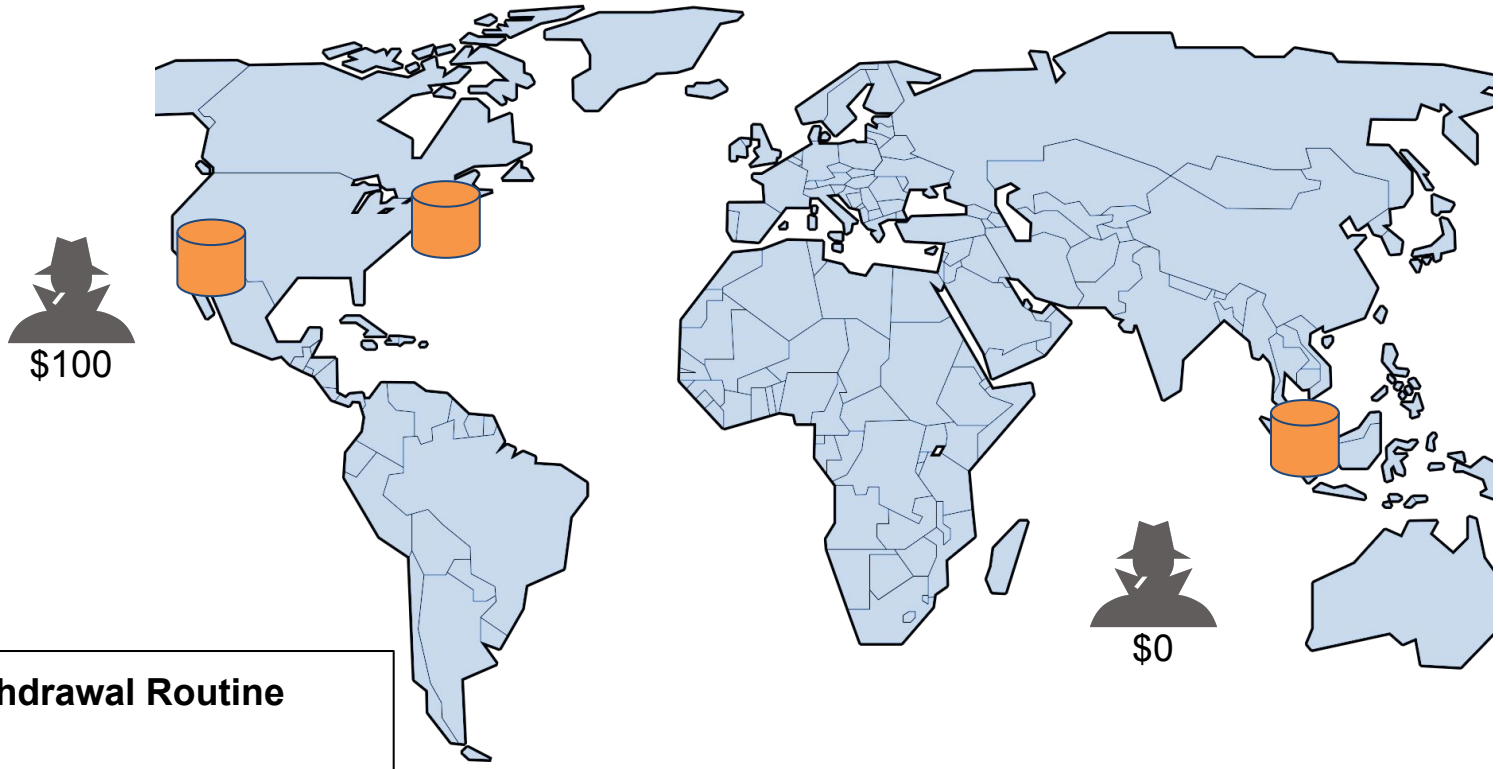
Choosing a Consistency Level
Good Example

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

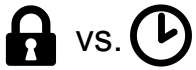


Withdrawal Routine

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

Account	Balance
xxxxx-4437	**$0**

Consistency Models

Tradeoff: 

- Strict
- Strong
- Sequential
- Causal
- Eventual

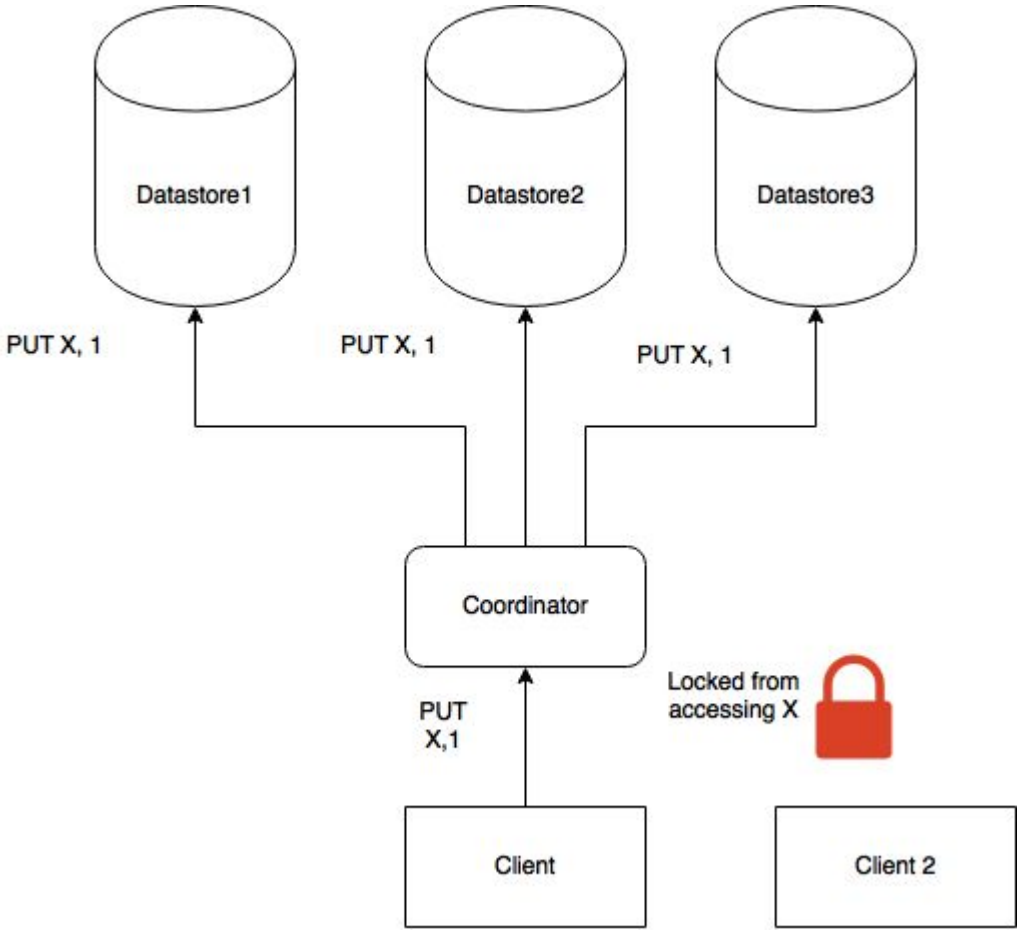
P3.3 Task 1: Strong Consistency

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’

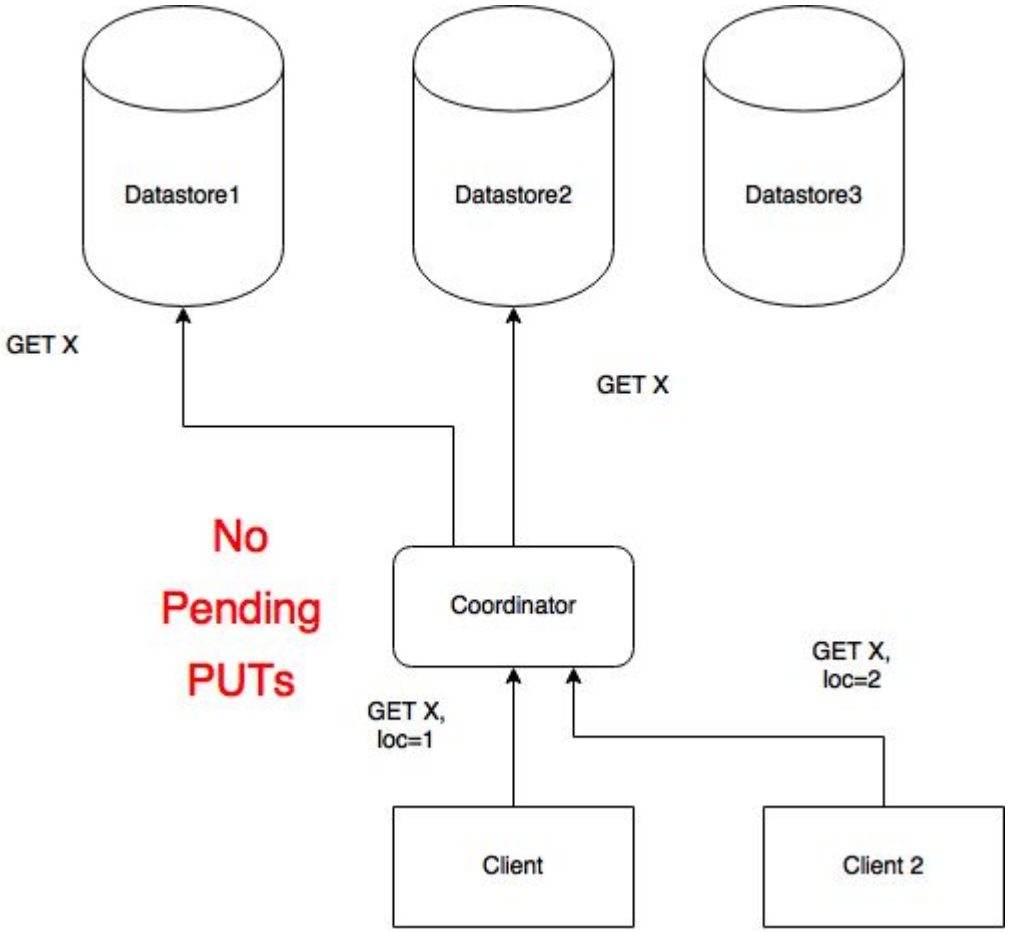
- Resolved in order of their arrival
- Any GET request in between 2 PUTs must return the last PUT value



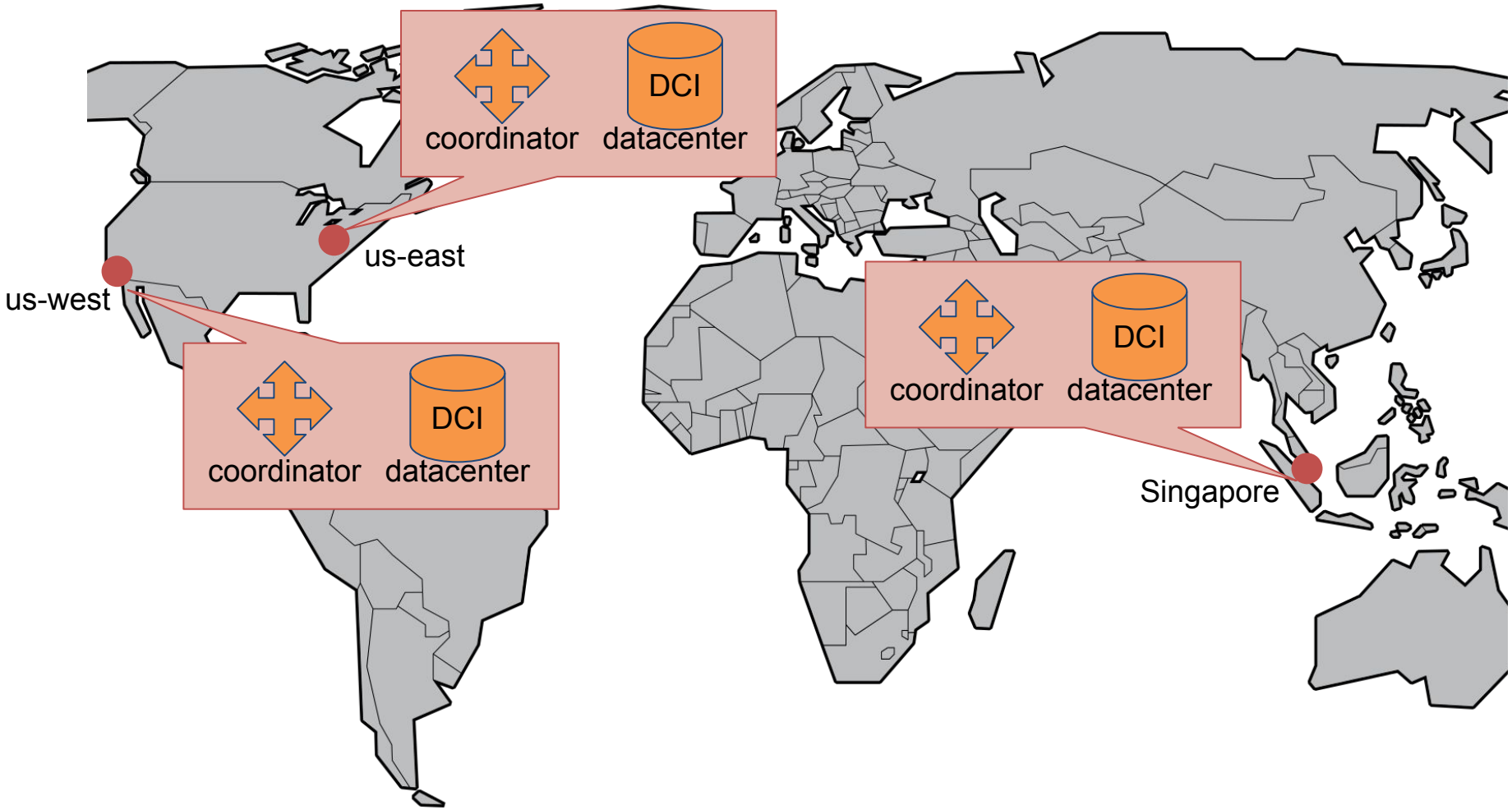
P3.3 Task 1: Strong Consistency

Multiple GET requests for key ‘X’

- All GET for key ‘X’ should be served concurrently if no PUT is pending



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



P3.3 Tasks 1 & 2: Strong Consistency

- Every request has a global order.
  - Task 1: Typically the order in which they arrive at the coordinator.
  - Task 2: Timestamp comes with the request.
- 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.

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

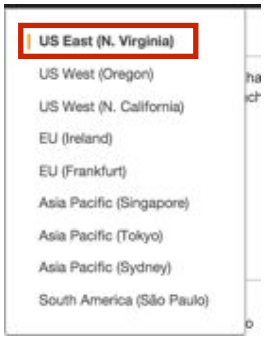
P3.3: Task Overview

- Launch Coordinators and DCs **all in us-east**
  - In Task 2, we will simulate global latencies for you
- Implement the Coordinators and Datastores
  - Strong Consistency
    - Tasks 1 (**Only Coordinator**) & 2
  - Eventual Consistency
    - bonus

P3.3:Task 2 Example

Task 2 Example

- Launch a total of 7 machines (3 data centers, 3 coordinators and 1 client)
- All machines should be launched in US East region.



All VMs will be launched in "**US East**", we will simulate the latencies for the location of datacenters and coordinators in Task 2.

US-EAST DATACENTER (*KeyValueStore.java*)

US-WEST DATACENTER (*KeyValueStore.java*)

SINGAPORE DATACENTER (*KeyValueStore.java*)

US-EAST COORDINATOR (*Coordinator.java*)

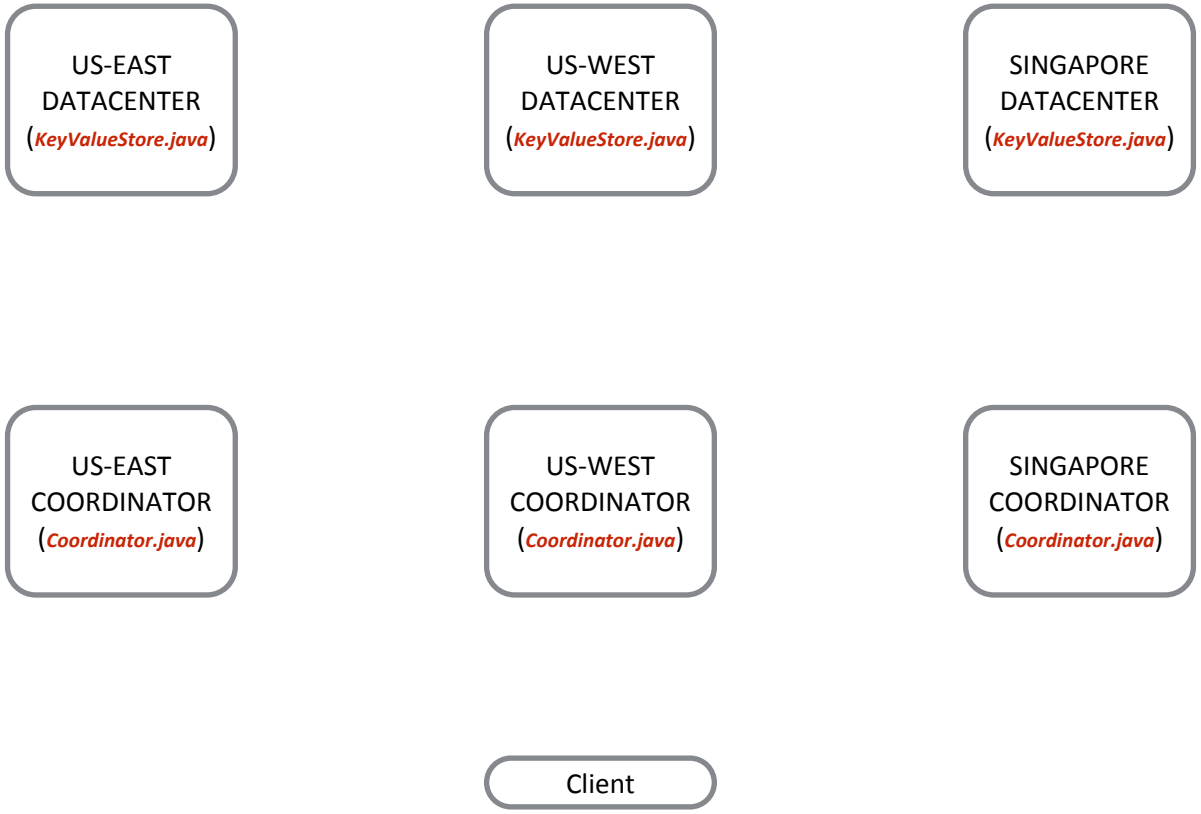
US-WEST COORDINATOR (*Coordinator.java*)

SINGAPORE COORDINATOR (*Coordinator.java*)

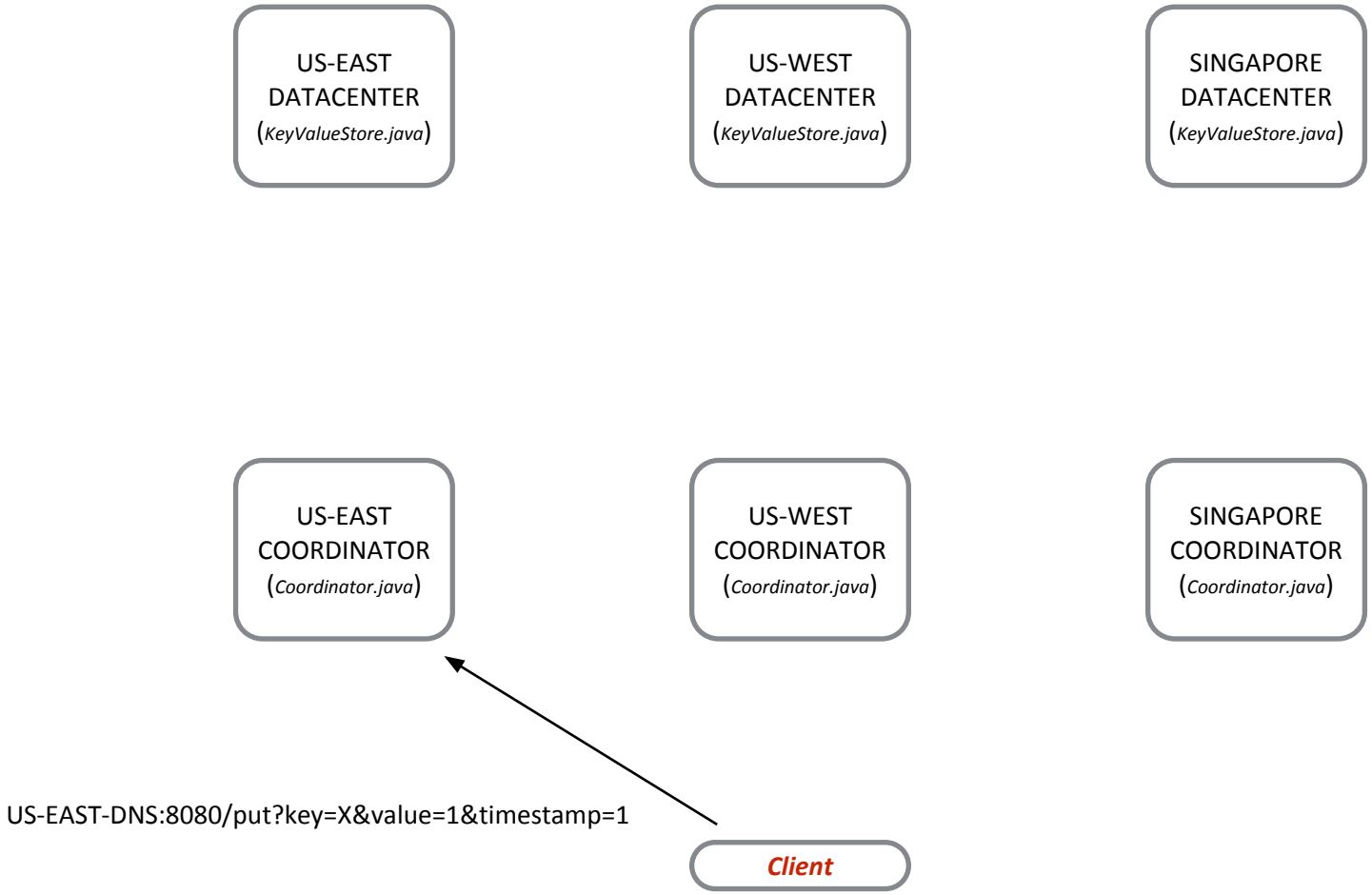
Client

P3.3 Task 2: Complete

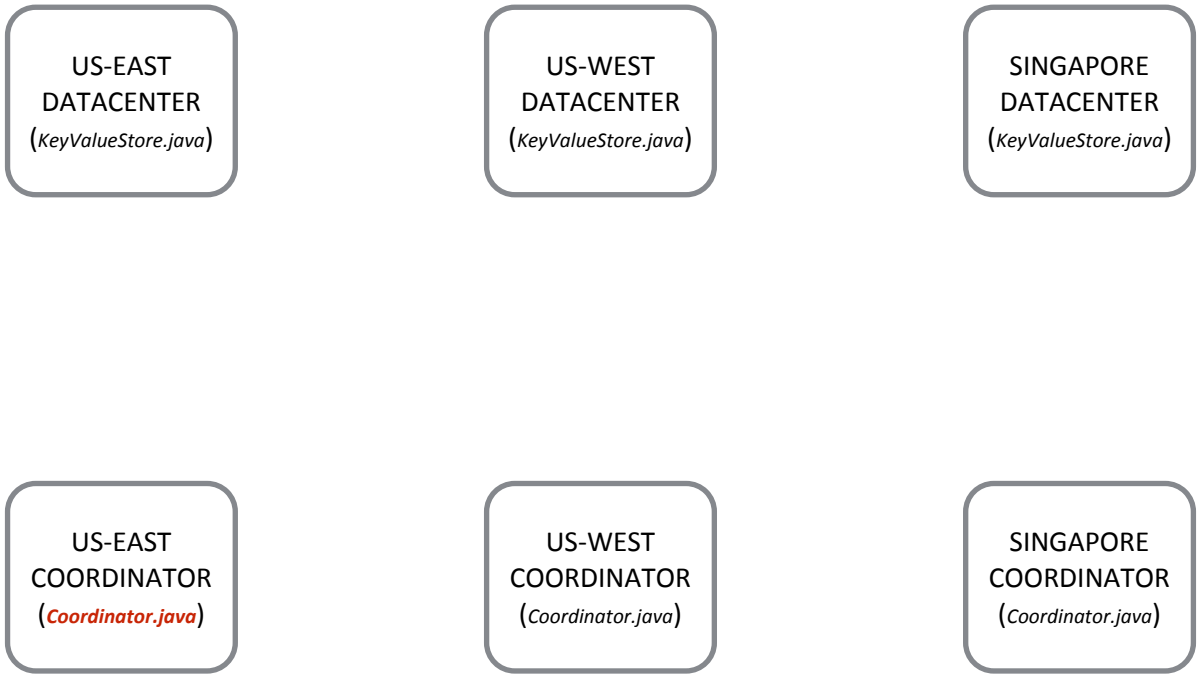
- KeyValueStore.java on Datacenters
- Coordinator.java on Coordinators



Example workflow for a PUT request using strong consistency



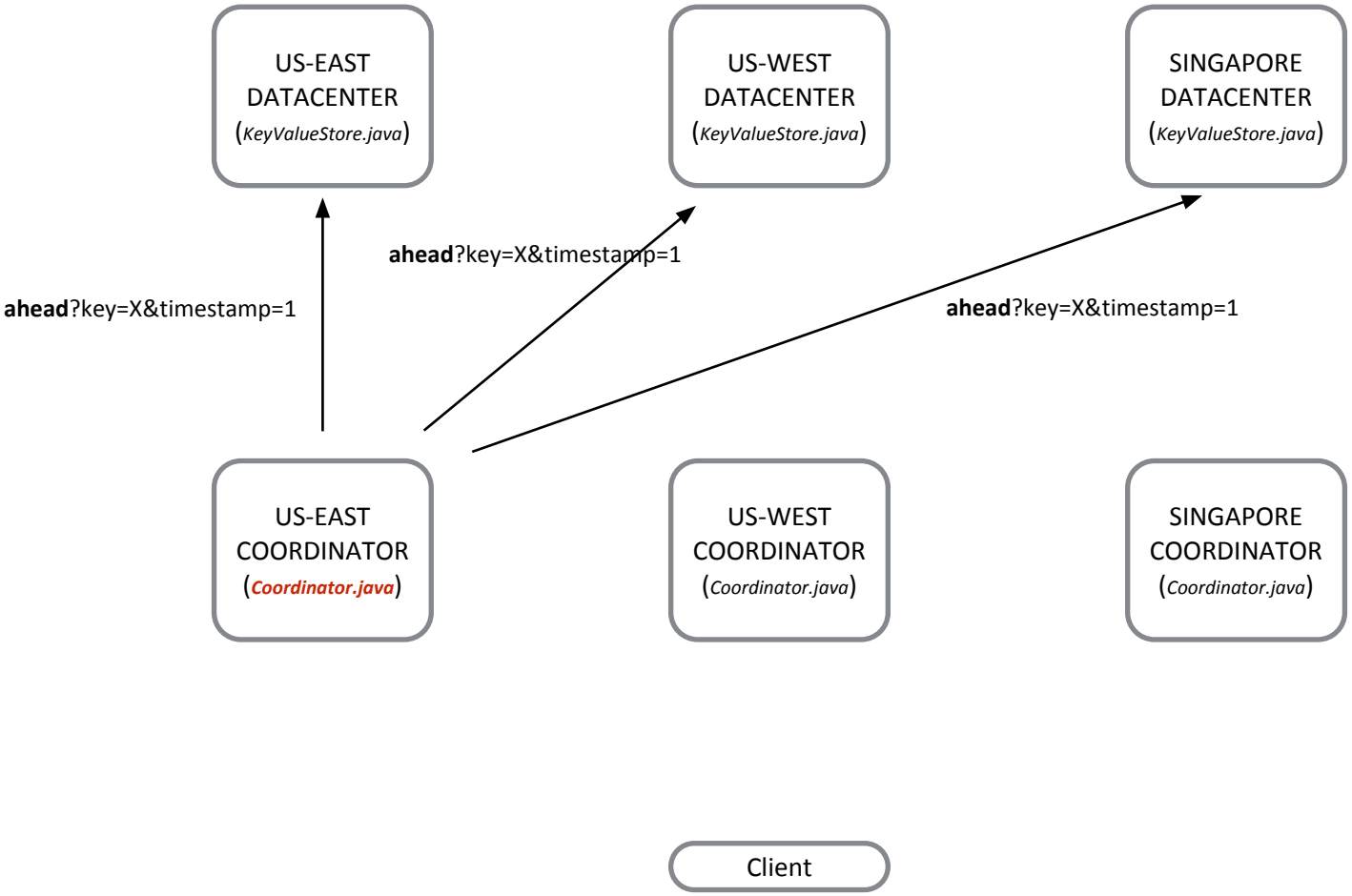
Example workflow for a PUT request using strong consistency



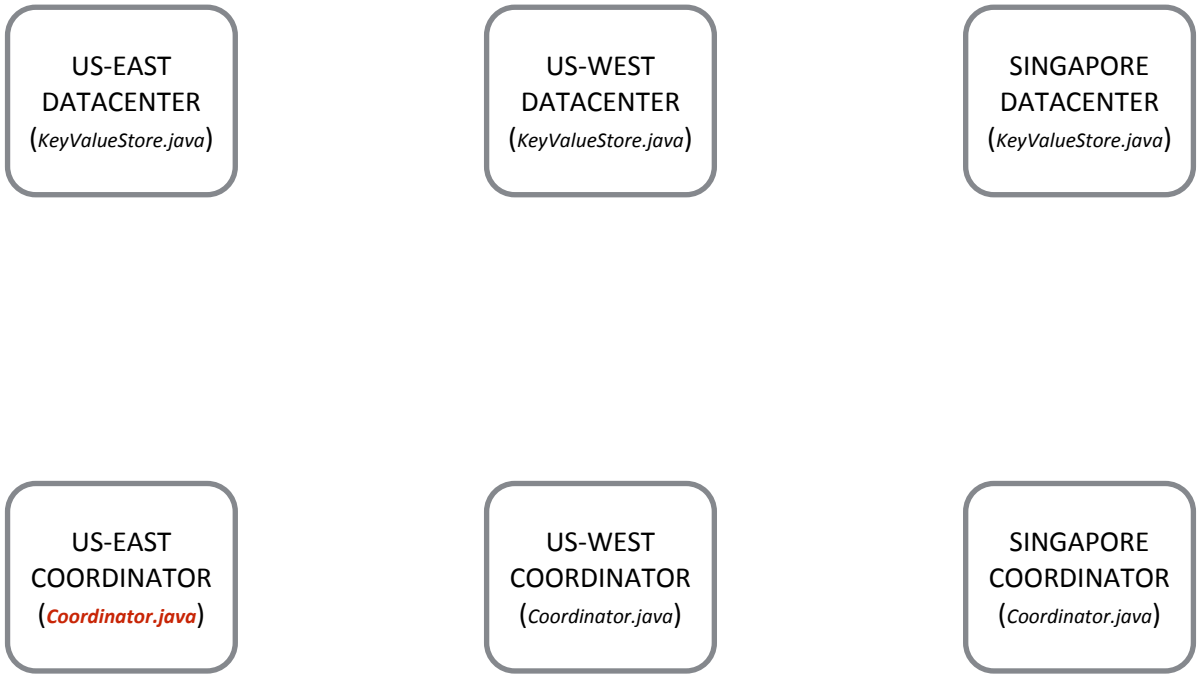
You should call KeyValueLib.AHEAD("X",1) to notify all 3 datacenters of this PUT request.
Resulting behavior may include:

- Locking subsequent requests for key "X" until current request is complete
- Done on datacenter.

Example workflow for a PUT request using strong consistency



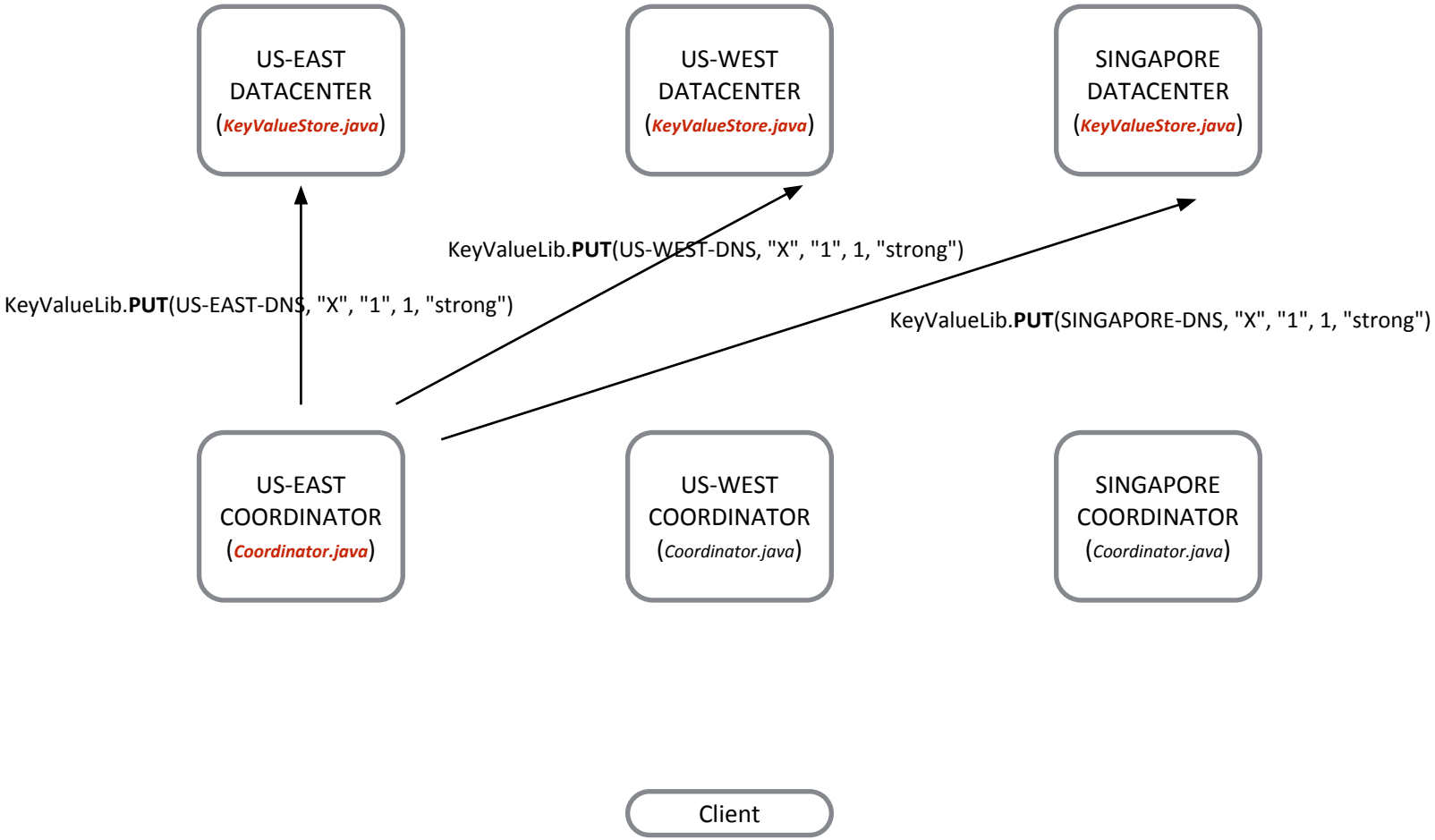
Example workflow for a PUT request using strong consistency



Example workflow for a PUT request using strong consistency

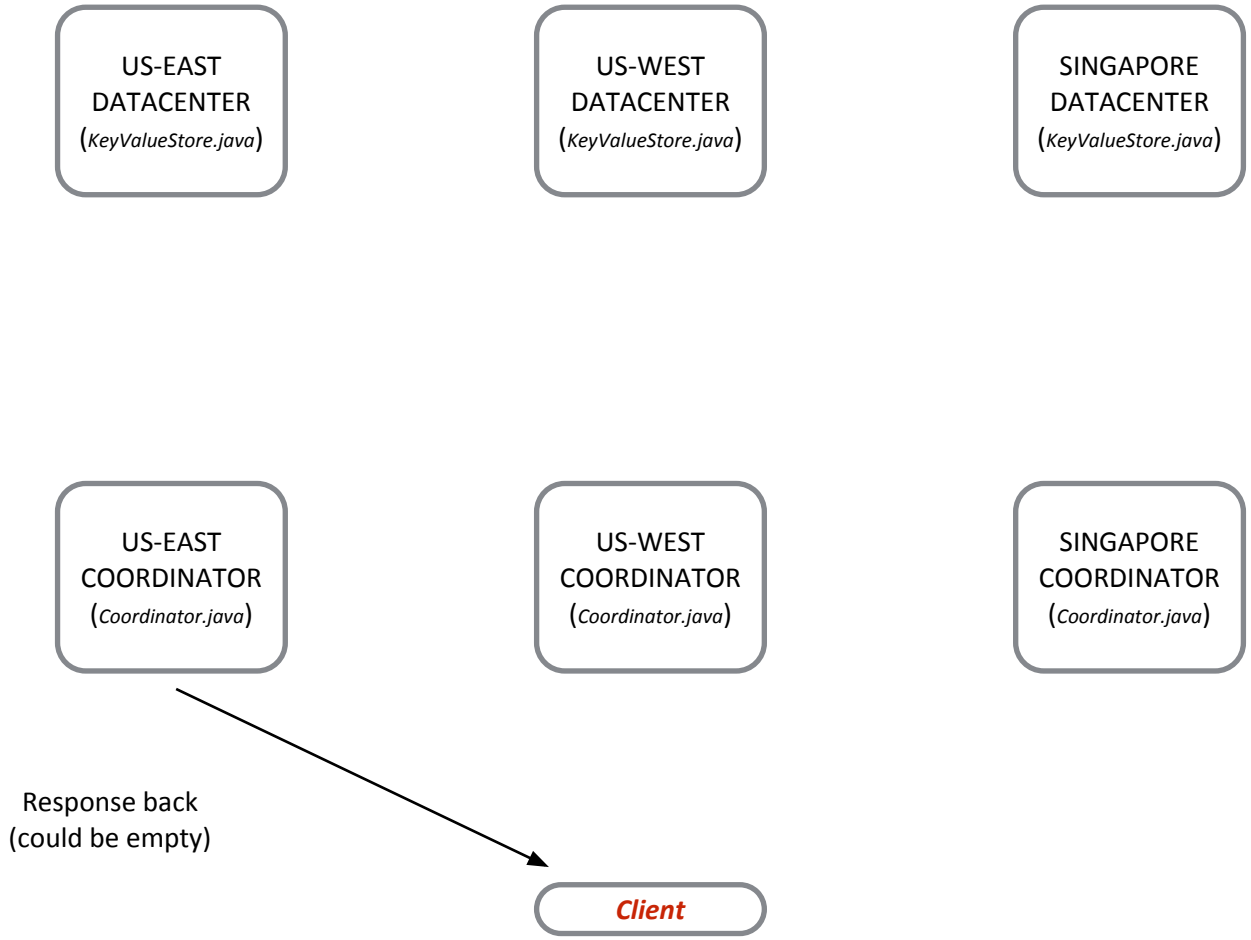
- If US-EAST is responsible for key “X”

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.



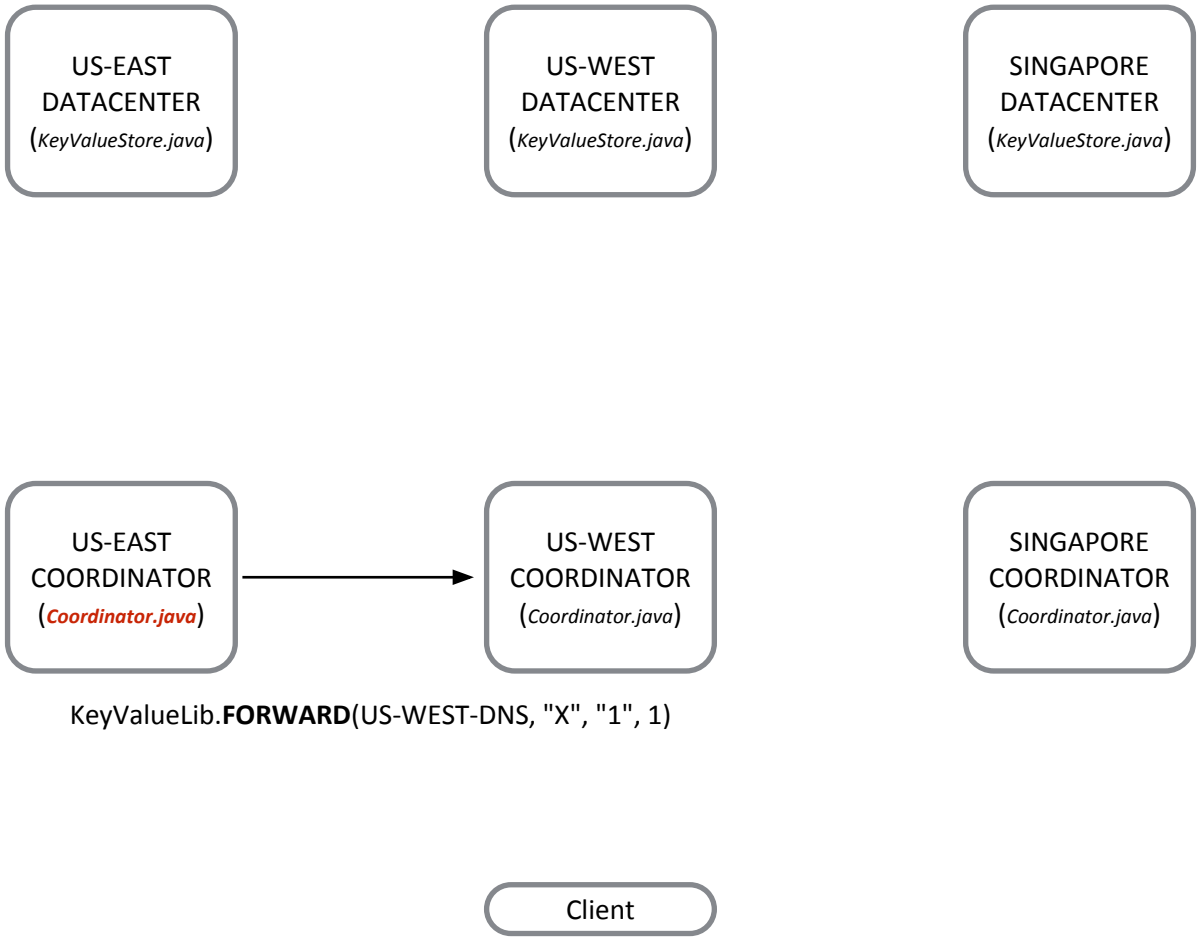
Example workflow for a PUT request using strong consistency

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



Example workflow for a PUT request using strong consistency

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



More Hints

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

Suggestions

- Read the primers! Practice the primer code!
- You should consider the differences between the 2 consistency levels 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.

However…

We are making an assumption!

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

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!

Upcoming Deadlines

- Quiz 7: Unit 4 - Module 14

  Due: THURSDAY 10/19/2017 11:59 PM ET

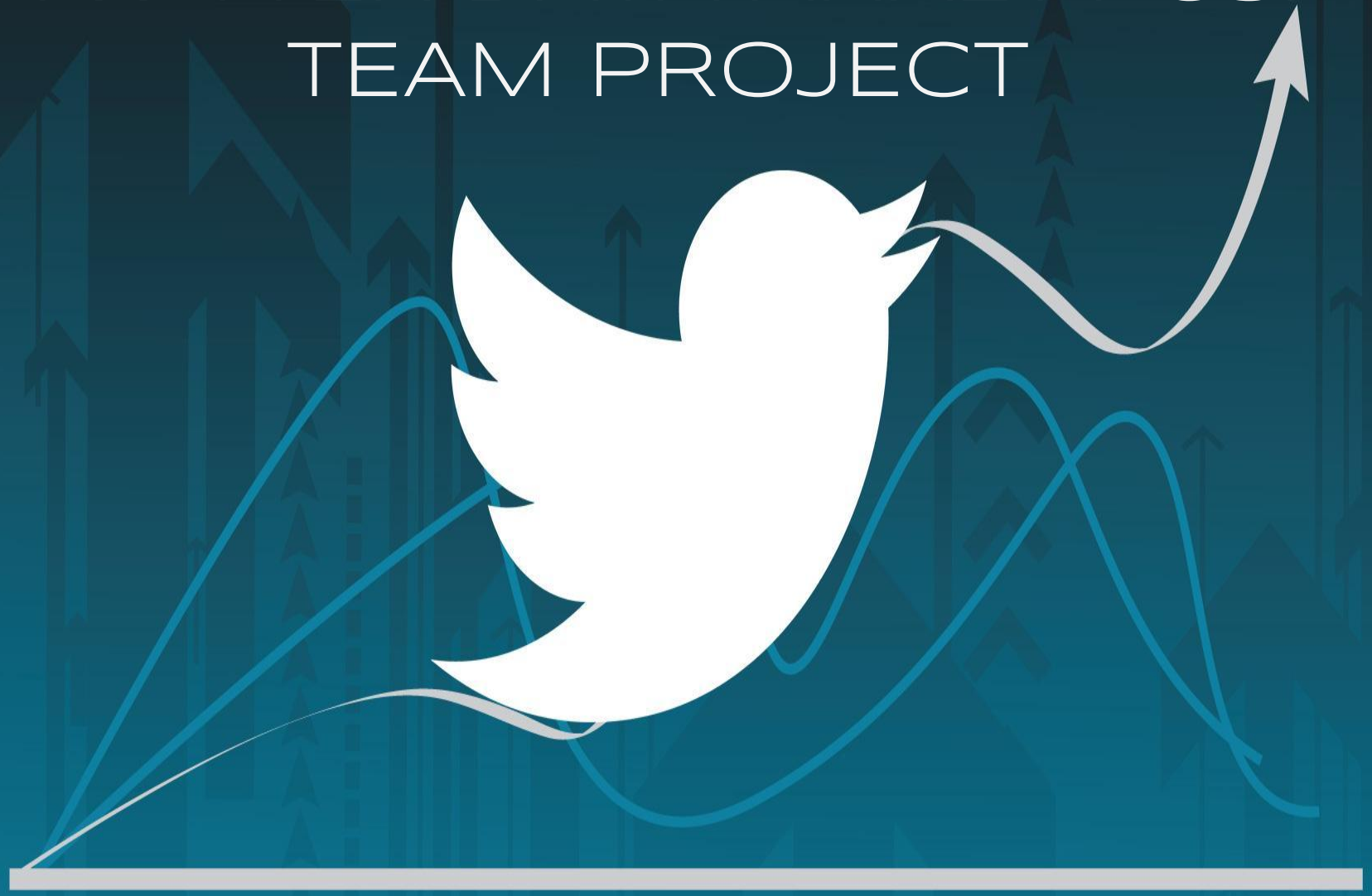
- Individual Project: P3.3, Replication and Consistency Models

  Due: 10/22/2017 11:59 PM ET.

- Team Project: Phase 1, Query 1

  Due: 10/22/2017 11:59 PM ET

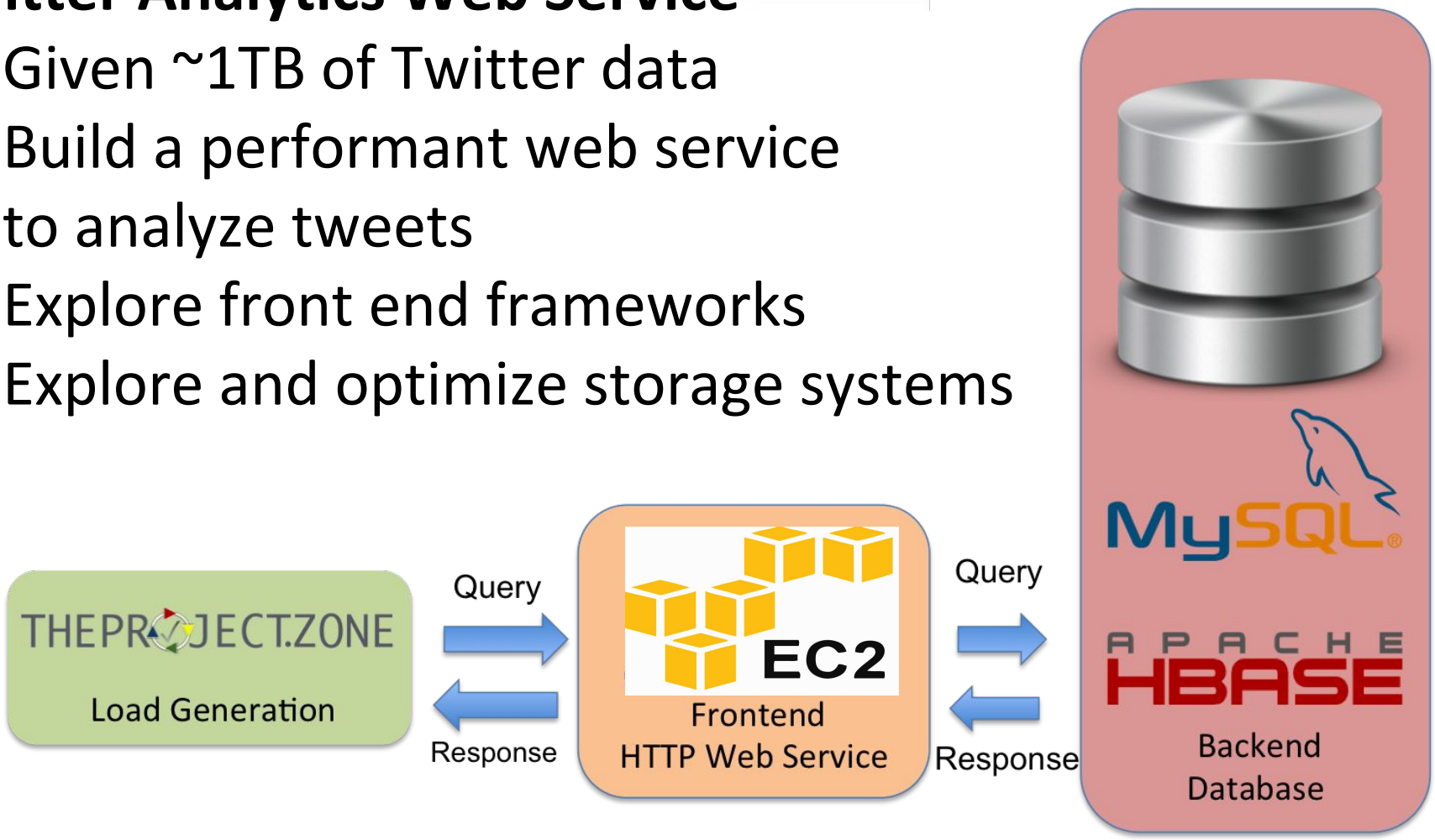
TWITTER DATA ANALYTICS: TEAM PROJECT



Team Project

Twitter Analytics Web Service

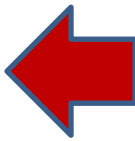
- Given ~1TB of Twitter data
- Build a performant web service to analyze tweets
- Explore front end frameworks
- Explore and optimize storage systems



Team Project

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

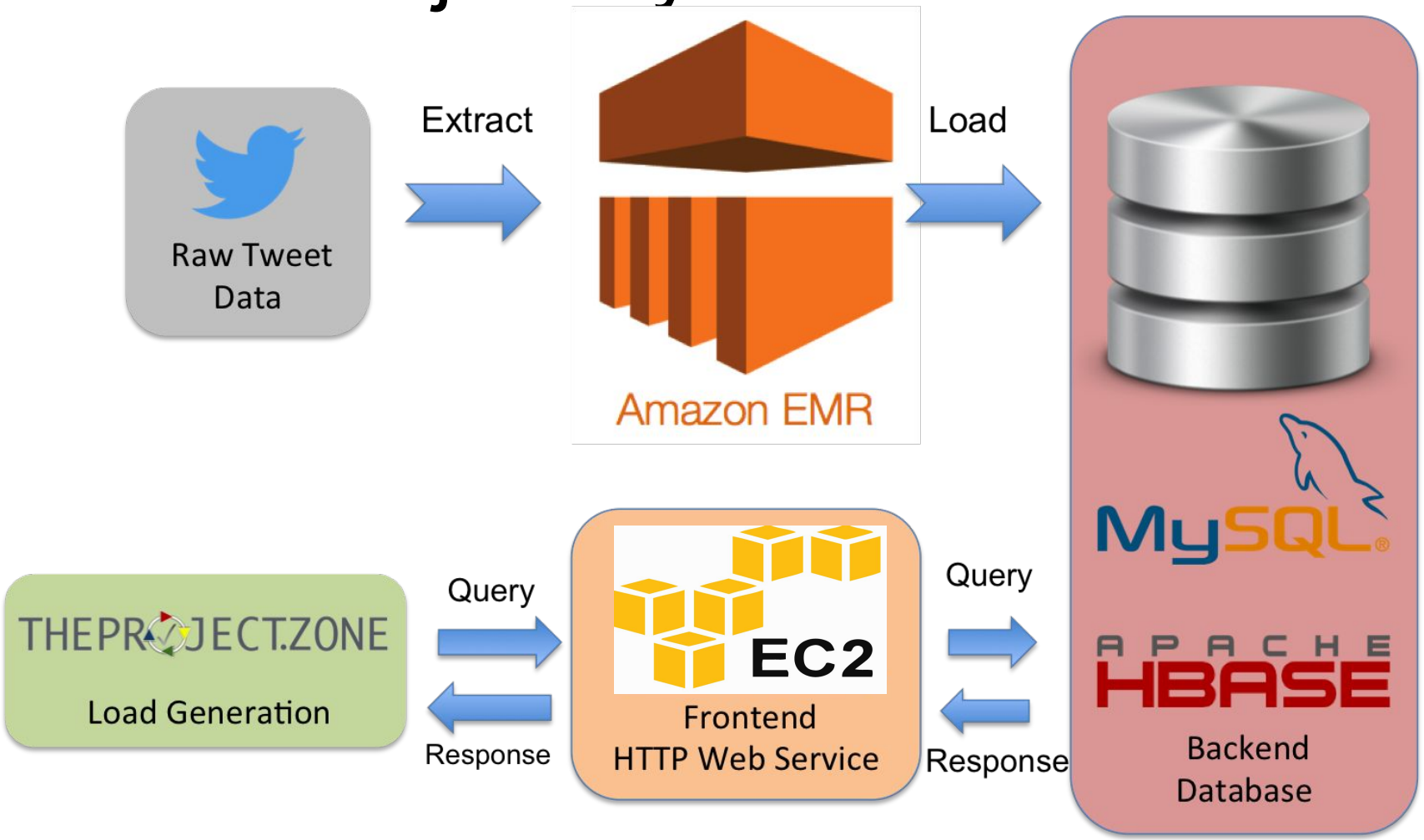
Input your team account ID and GitHub username on TPZ



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

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

Team Project System Architecture



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

Git workflow

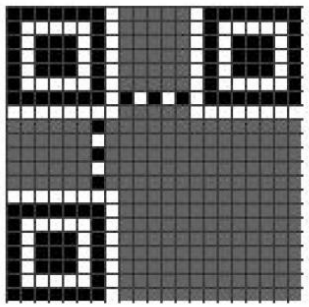
- Commit your code to the private repo we set up
  - Update your GitHub username in TPZ!
- Make changes on a new branch
  - Work on this branch, commit as you wish
  - Open a pull request to merge into the master branch
- Code review
  - Someone else needs to review and accept (or reject) your code change
  - Capture bugs and know what others are doing

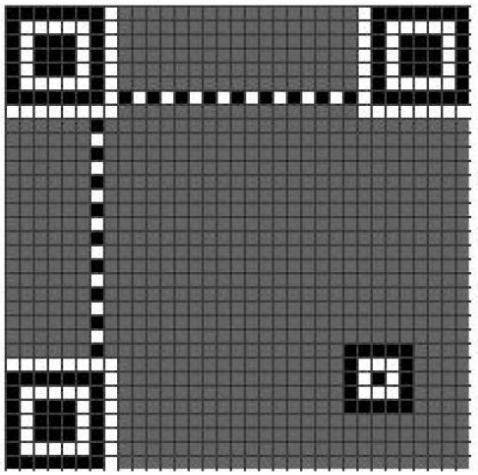
Query 1, QR code

- Query 1 is a simple heartbeat query
  - Implement encoding and decoding of QR code
    - A simplified version
  - You must explore different web frameworks
    - Get at least 2 different web frameworks working
    - Select the better performance of framework
    - Provide evidence of your experimentation
  - In this query, there is no backend database involved

Query 1 Specification

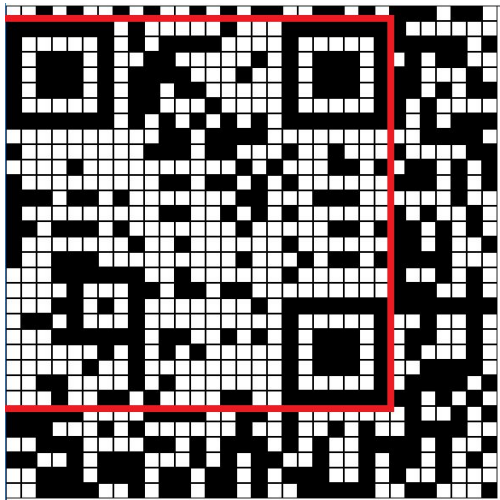
- Structural components
  - Two sizes of QR, depending on the message length
  - Position detection patterns
  - Alignment patterns
  - Timing Patterns
- Fill in message
  - Interleave chars & error detection codes
  - Put in the payload, move in S shape





Query 1 Example

- Encoding
  - For a message, return the image encoded as a hex string
  - The area in the red box in the image below
- Decoding
  - Find the encoded image in a random background
  - The image may be rotated 0,90,180,270 degrees
    - Use patterns to find the QR code!
  - Then do the inverse of encoding
  - Sorry, can’t scan it with a camera yet :(



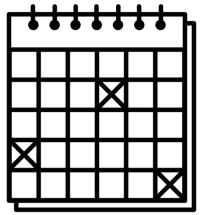
Query 2, Hashtag Recommendation

- Query 2 is a DB read query
  - Given a sentence, return hashtags that appeared most frequently with words in it
  - You have to perform Extract, Transform and Load (ETL)
    - Can be done on AWS, Azure or GCP
    - Dataset is available on all three platforms
  - Pay close attention to the ETL rules and related definitions
  - Make good use of the reference data & server
  - In this query, you will need both front end + back end
    - MySQL and HBase
    - You need two 10-minute submissions, (Q2M & Q2H) by the deadline

Query 2 Example

- Assume dataset contains
  - user 123456789: Cloud is great #aws
  - user 987654321: Cloud computing is tough #azure
- Sample request and response
  - Request: keywords=cloud,computing&userid=123456789&n=3
  - #aws: 2 points; #azure: 2 points
  - Response: #aws,#azure
    - Return top n hashtags (or all), break ties lexicographically

Team Project Time Table



Phase (and query due)	Start	Deadline	Code and Report Due
Phase 1 ● **Q1, Q2**	**Monday 10/09/2017 00:00:00 ET**	**Q1: Sunday 10/22/2017 23:59:59 ET** **Q2: Sunday 10/29/2017 23:59:59 ET**	**Tuesday 10/31/2017 23:59:59 ET**
Phase 2 ● Q1, Q2,Q3	Monday 10/30/2017 00:00:00 ET	Sunday 11/12/2017 15:59:59 ET	
Phase 2 Live Test (Hbase **AND** MySQL) ● Q1, Q2, Q3	Sunday 11/12/2017 18:00:00 ET	Sunday 11/12/2017 23:59:59 ET	Tuesday 11/14/2017 23:59:59 ET
Phase 3 ● Q1, Q2, Q3, Q4	Monday 11/13/2017 00:00:00 ET	Sunday 12/03/2017 15:59:59 ET	
Phase 3 Live Test (Hbase **OR** MySQL) ● Q1, Q2, Q3, Q4	Sunday 12/03/2017 18:00:00 ET	Sunday 12/03/2017 23:59:59 ET	Tuesday 12/05/2017 23:59:59 ET

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)**
- **Query 1 is due on Oct 22, not Oct 29! We want you to start early!**

Hints/Suggestions

- Use Azure/GCP for ETL to save budget.
- Experiment with the different MySQL Storage Engines.
- Connection Pooling to parallelize access to the database.
- Scans vs Gets in HBase
- HBase configuration parameters
- MySQL configuration parameters

Start early!

Team Project Q1 Due
Sunday 10/22

Upcoming Deadlines

- Conceptual Topics: OLI (Module 14)

  Quiz 7 due: **Thursday, 10/19/2017 11:59 PM Pittsburgh**

- P3.3: Replication and Consistency Models

  Due: **Sunday, 10/22/2017 11:59 PM Pittsburgh**

- Team Project: Phase 1 - Query 1

  Due: **Sunday 10/22/2017 11:59 PM Pittsburgh**

Q&A