15-319 / 15-619
Cloud Computing

Recitation 7

October 11, 2016

Overview

- **Administrative issues**
  Office Hours, Piazza guidelines
- **Last week's reflection**
  Project 3.1, OLI Unit 3 Module 10, 11, 12, Quiz 5
- **This week's schedule**
  - Quiz 6 - Friday, October 14th (Module 13)
  - Project 3.1 - Sunday, October 16th
- **Twitter Analytics: The team project is upon us**

Last Week : A Reflection

- Content
  - Unit 3 - Modules 10, 11 and 12:
    Virtualizing Resources on the Cloud
  - Quiz 5 - completed
- Project
  - Files, Databases (SQL & NoSQL)
    - MySQL
    - HBase
    - Bugs discovered by alert students
      - **Thanks:** Scott Meng, Tianyuan Jiang
      - **Thanks:** Srijith Karippure

This Week: Content

UNIT 3: Virtualizing Resources for the Cloud

- Module 10: Resource virtualization (memory)
- Module 11: Resource virtualization (I/O)
- Module 12: Case Study
- **Module 13: Storage and network virtualization**
  - Software Defined Data Center (SDDC)
  - Software Defined Networking (SDN)
    - Device virtualization (Router and NIC virtualization)
    - Link virtualization (Bandwidth/datapath virtualization)
  - Software Defined Storage (SDS)
    - IOFlow
- **Quiz 6, Friday, October 14th**

Project 3.1 Feedback

https://goo.gl/N4Ja4u

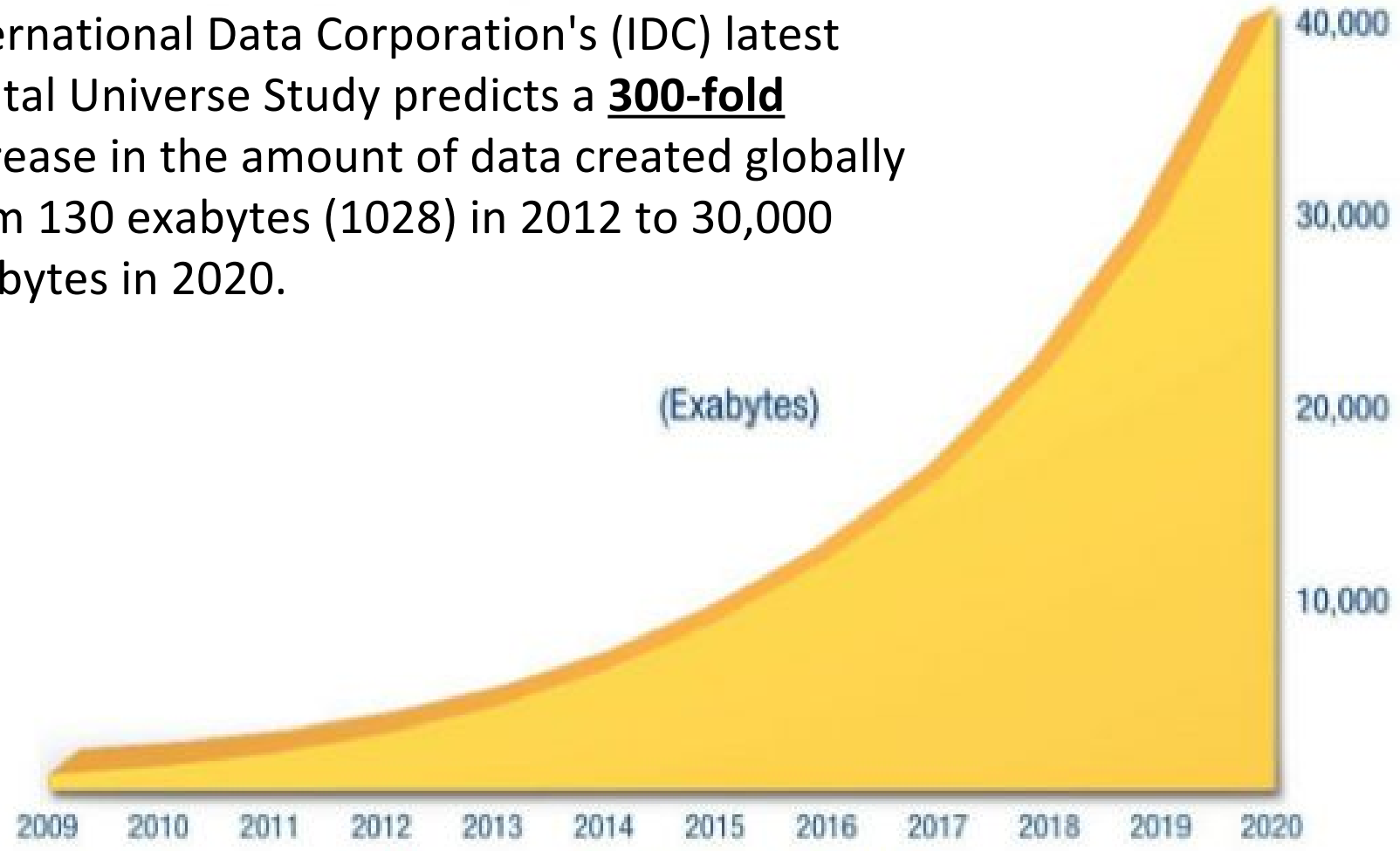
Please leave us feedback

Project 3 Weekly Modules

- P3.1: Files, SQL and NoSQL
  - Primer: Storage Benchmarking
- P3.2: Replication and Consistency models
  - Primer: Intro. to Java Multithreading
  - Primer: Thread-safe programming
  - Primer: Intro. to Consistency Models
- P3.3: Social network with heterogeneous backend storage

P3.2: Motivation - Scale

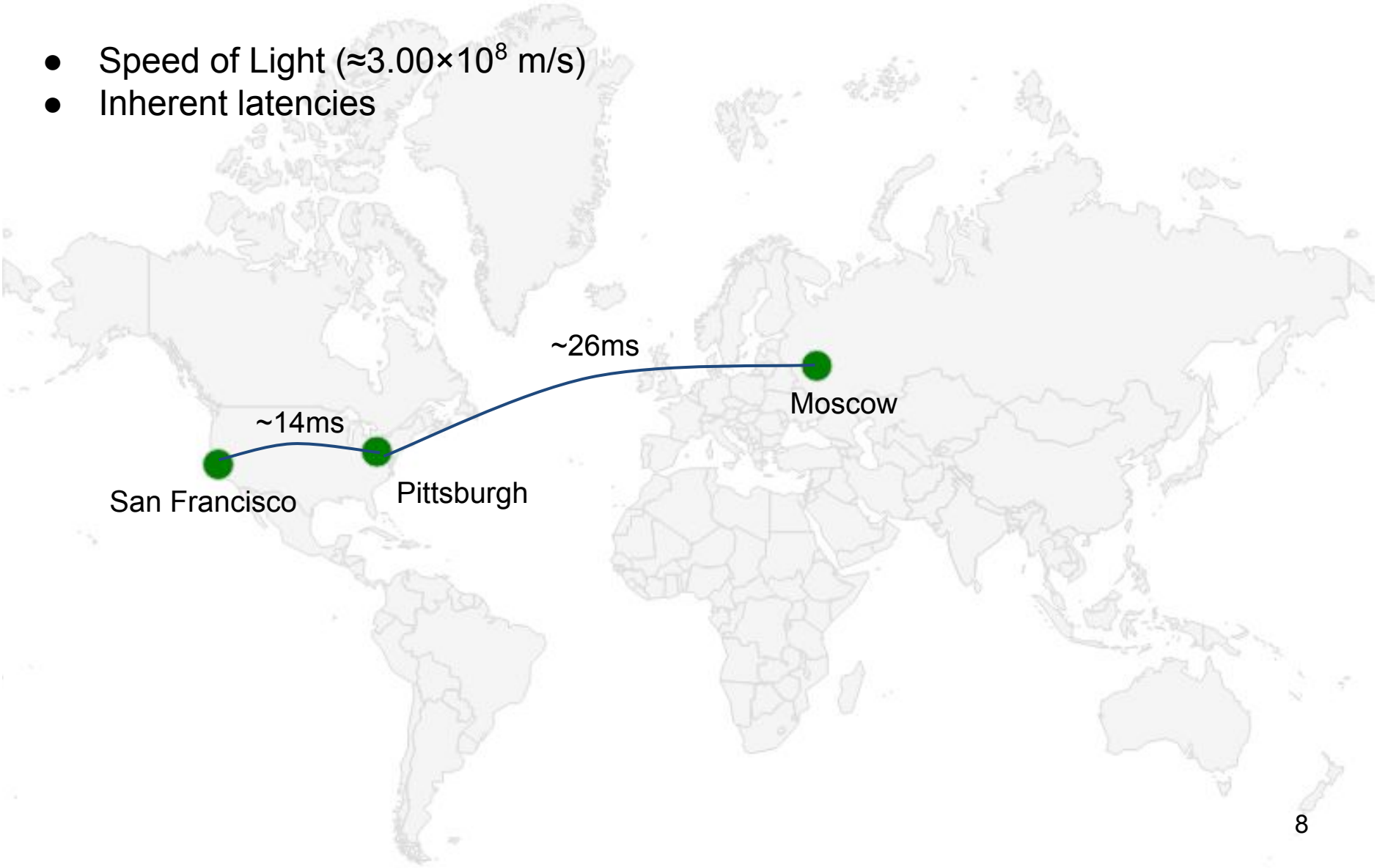
International Data Corporation's (IDC) latest Digital Universe Study predicts a **300-fold** increase in the amount of data created globally from 130 exabytes (1028) in 2012 to 30,000 exabytes in 2020.



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

P3.2: Motivation - Global Users

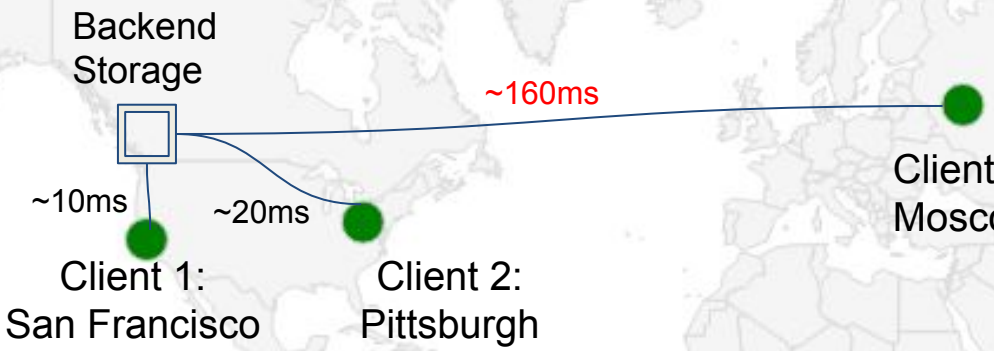
- Speed of Light (≈3.00×10^8 m/s)
- Inherent latencies



P3.2: Motivation - Requirements

- 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

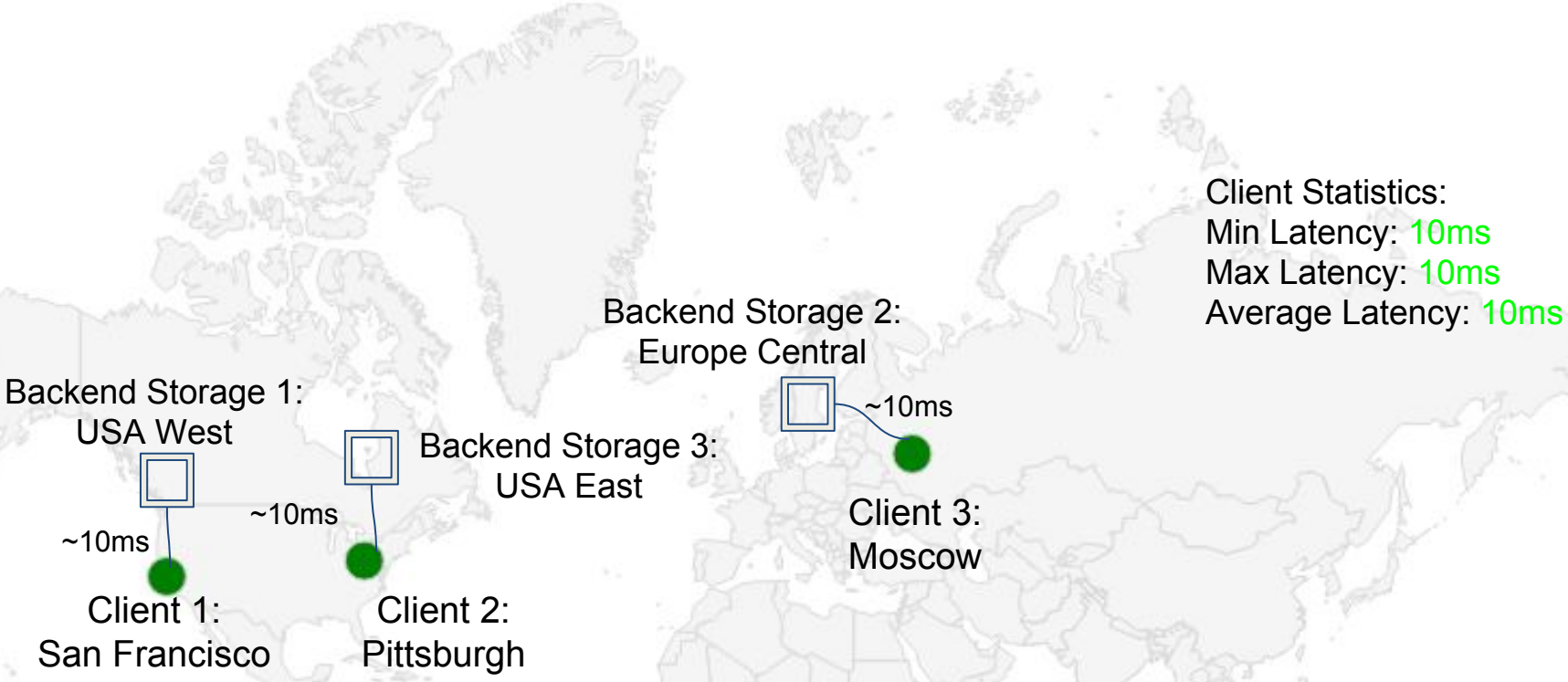
P3.2: Motivation - Latency



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

P3.2: Motivation - Replicas

P3.2: Motivation - Replicas cont



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

P3.2: Motivation - Replicas

- As you can see, by adding replicas to strategic locations in the world, we can greatly reduce the latency seen by outlying clients
- Each added datacenter decreases the average latency seen by all clients
- But at what cost?

P3.2: Motivation - Replicas?

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

Cost: ?????

P3.2: Motivation - Replicas

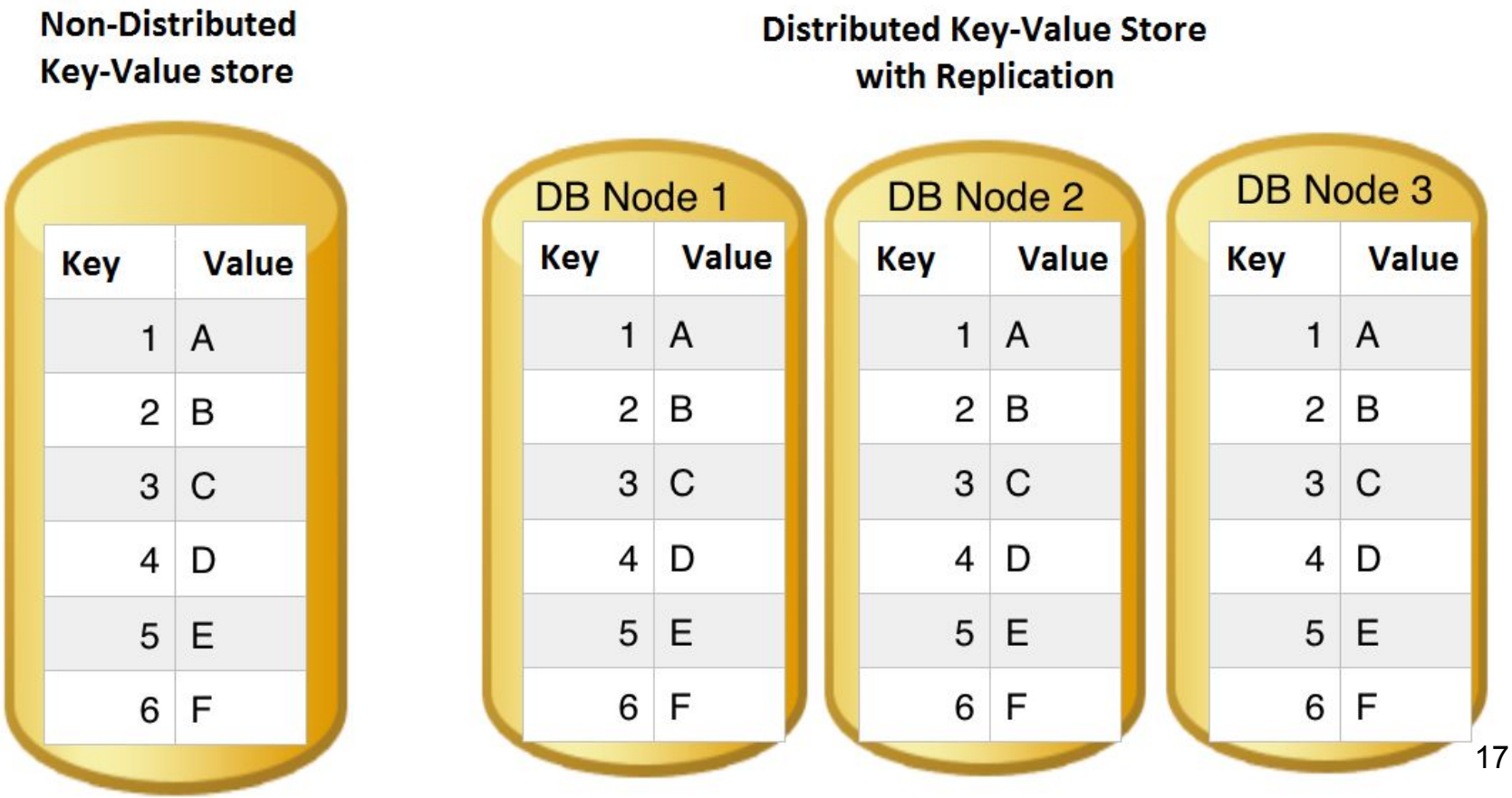
- There are downsides to adding additional replicas
  - Cost for new data center to host the replica
  - May increase latency for write requests during replication, because we need to update every single replica
- As you add more replicas throughout the world, the benefit per replica tapers off

P3.2: Motivation - Requirements

- Inherent dynamicity of data
- Large scale data intensive systems
  - Many reads/ writes impact throughput performance
    - Hot records
  - Global reach creates latency issues
  - Expectation to be 100% available
- Possible Solutions
  - Vertically scale storage backend
  - Replication
  - Sharding

P3.2: Replication (1)

- Duplicate the data across multiple instances and/or locations



P3.2: Replication (2)

- Duplicate the data across multiple instances
- Advantages
  - Fetching data can be faster
    - 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

P3.2: Consistency

- Data consistency across replicas is important
  - Strict, Strong (Linearizability), Sequential, Causal and Eventual Consistency
- This Week - Strong Consistency
  - All datastores must return the same value for a key
  - The order in which the values are updated must be preserved
- Bonus - Eventual Consistency

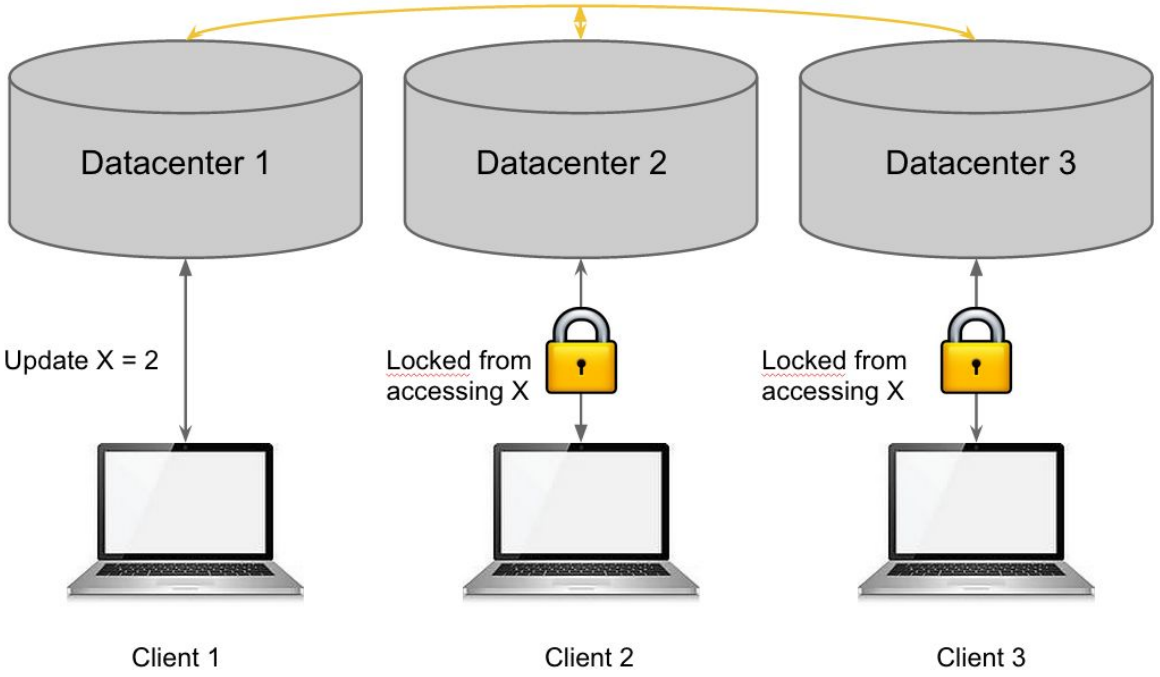
P3.2: Strong Consistency

Single PUT request for key ‘X’

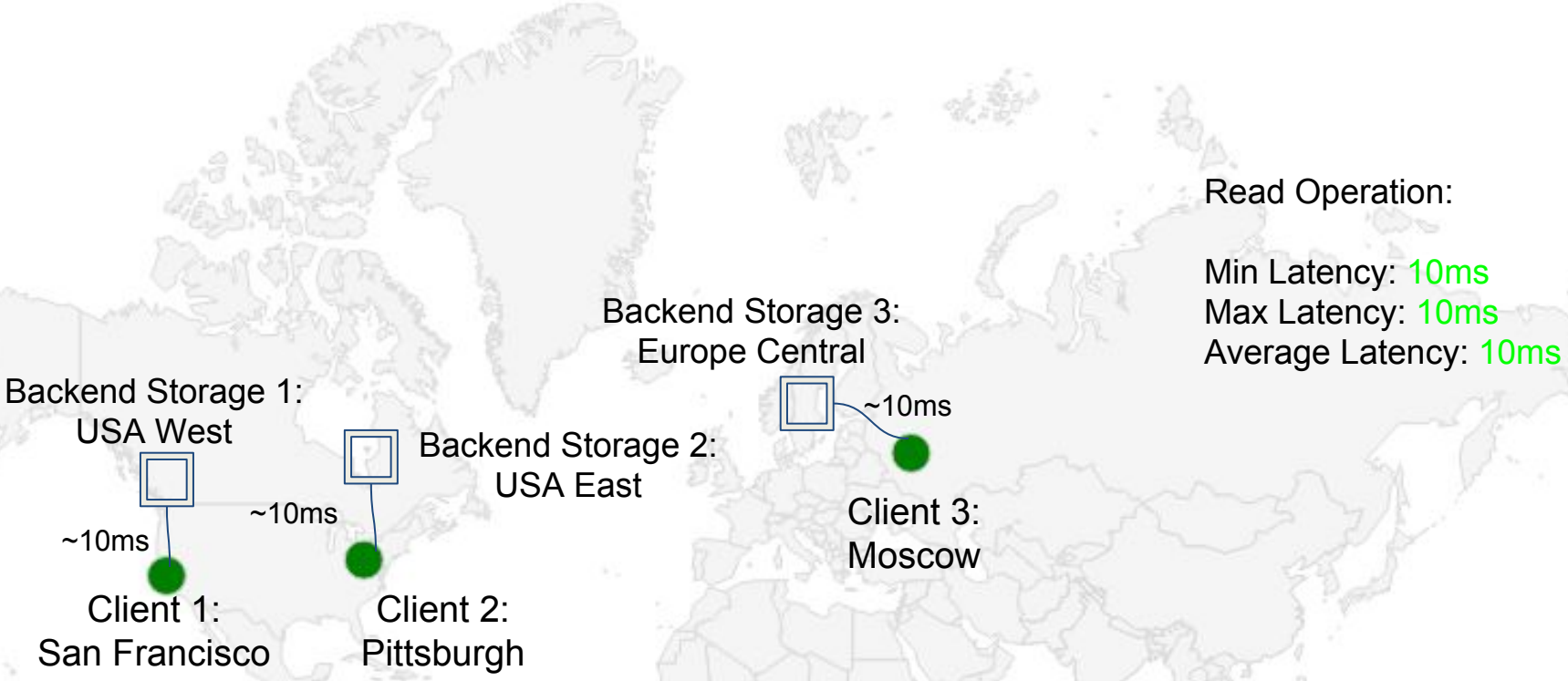
- Block all GET for key ‘X’ until all datastores are updated
- GET requests for a different key ‘Y’ must be allowed

Multiple PUT requests for ‘X’

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



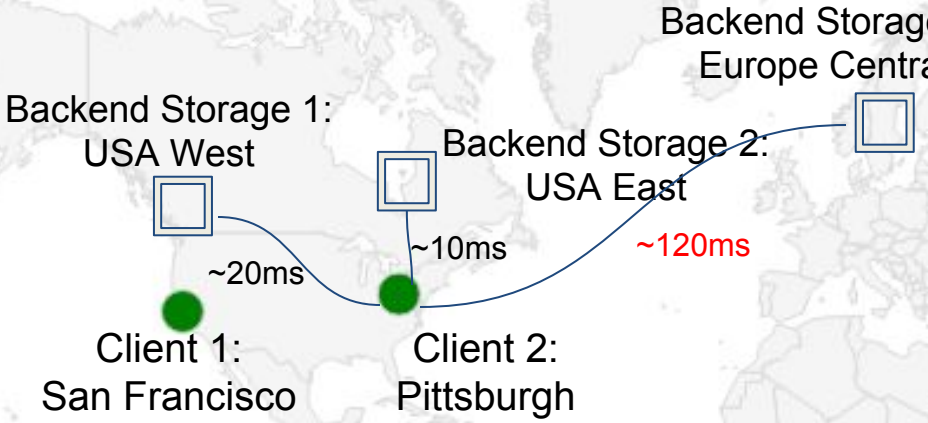
P3.2: Motivation - Replication READ



Read Operation:

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

P3.2: Motivation - Replication WRITE



Write Operation:

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

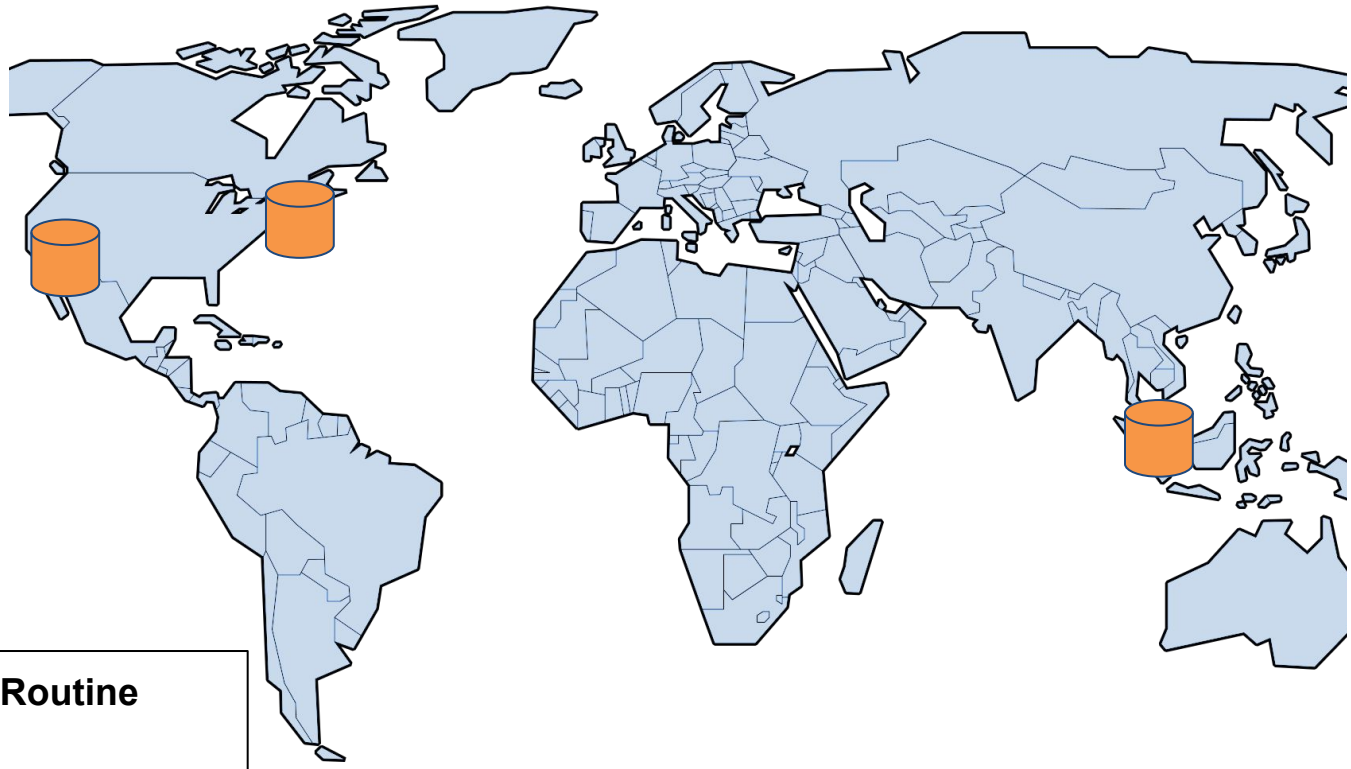
Same across all clients

Even worse if the operations block each other!

P3.2: Motivation - Replication

- 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, all write requests must now update all 3 replicas
  - If write requests for a certain key is created, then they all have to wait for each other to finish

P3.2: Motivation - Consistency



Withdrawal Routine

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

Account	Balance
xxxxx-4437	$100

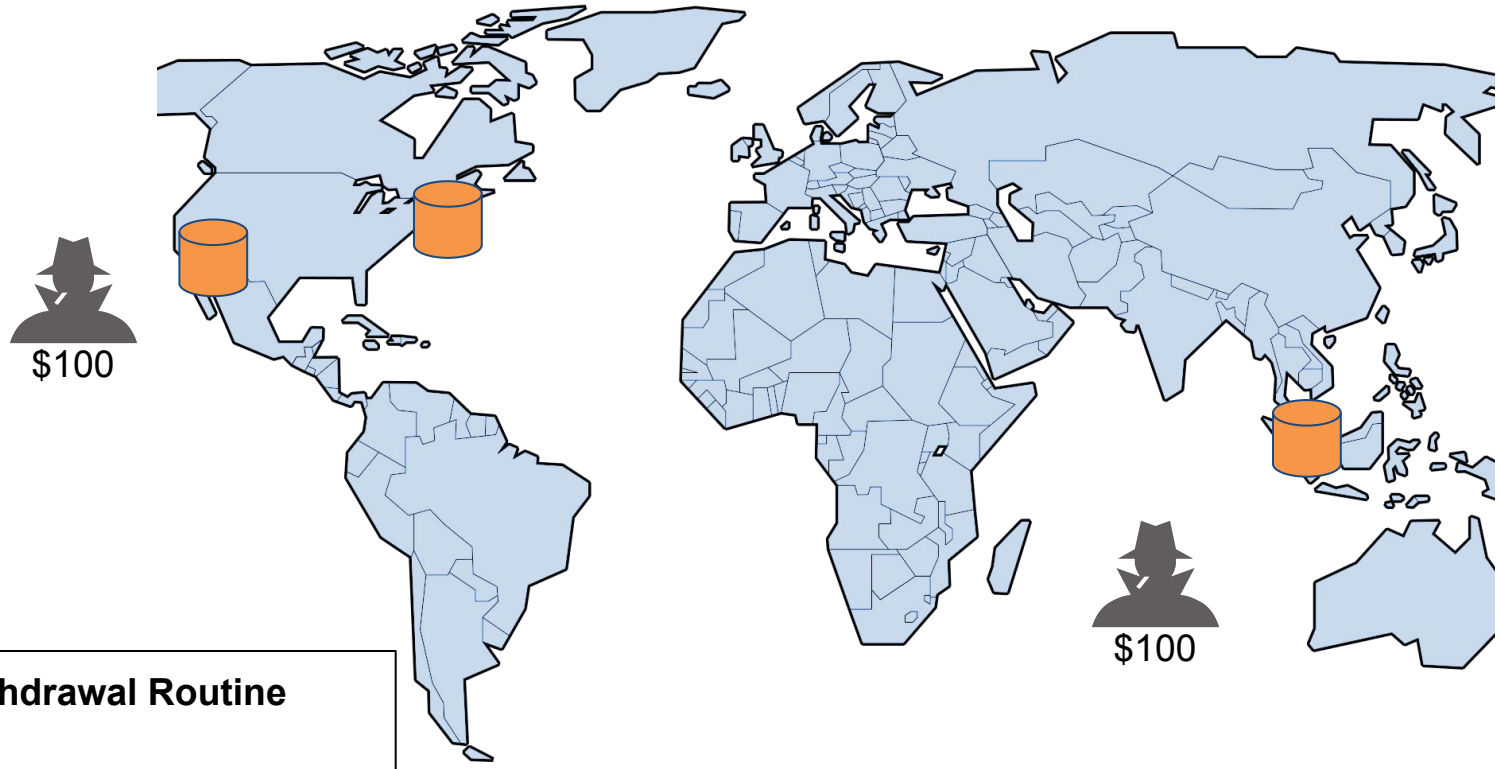
P3.2: Motivation - Consistency

Withdrawal Routine

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

Account	Balance
xxxxx-4437	$100

P3.2: Motivation - Consistency



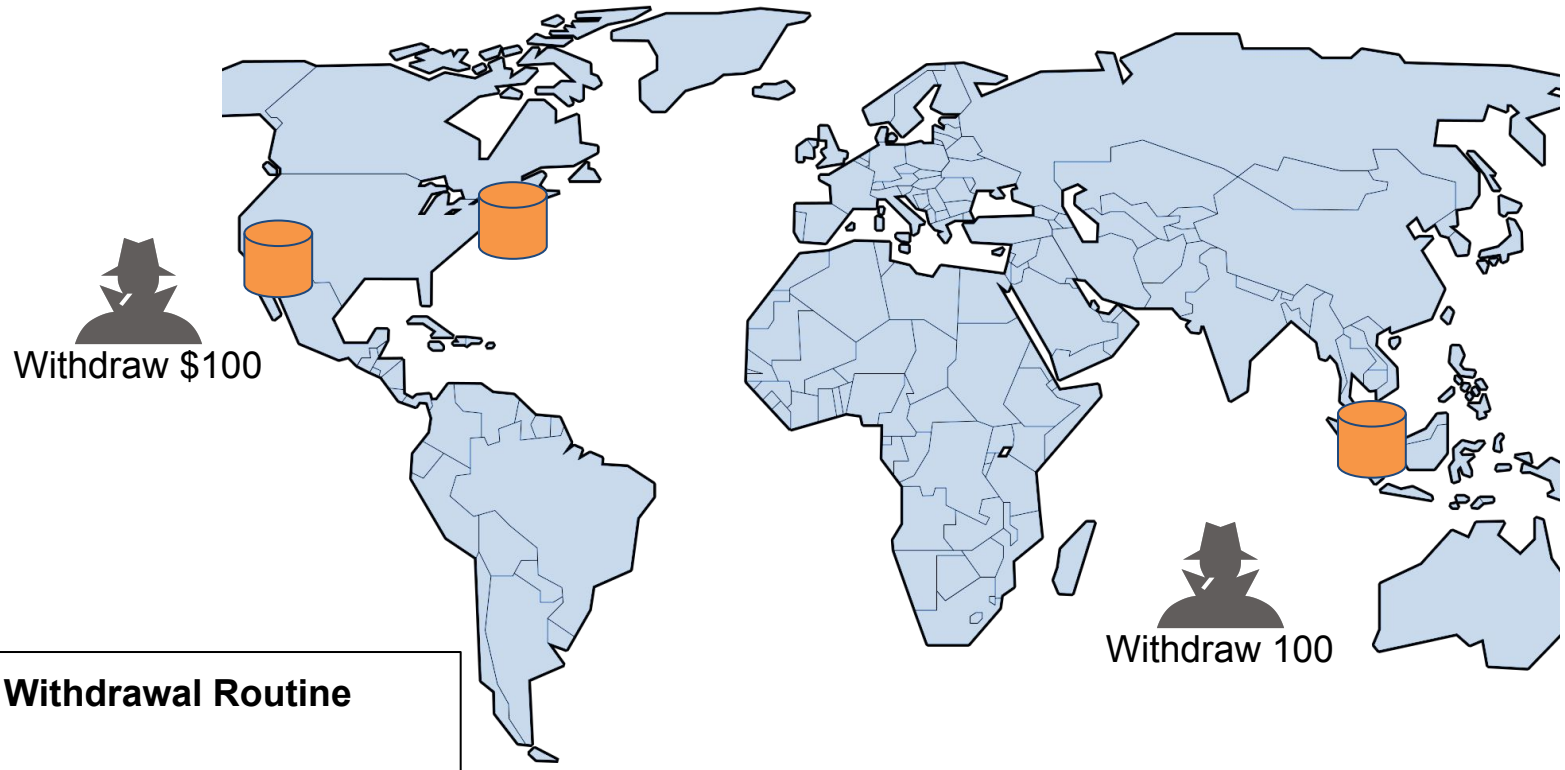
Withdrawal Routine

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

Account	Balance
xxxxx-4437	**$0**

Bank lost $100

P3.2: Motivation - Consistency



Withdrawal Routine

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

Account	Balance
xxxxx-4437	**$100**

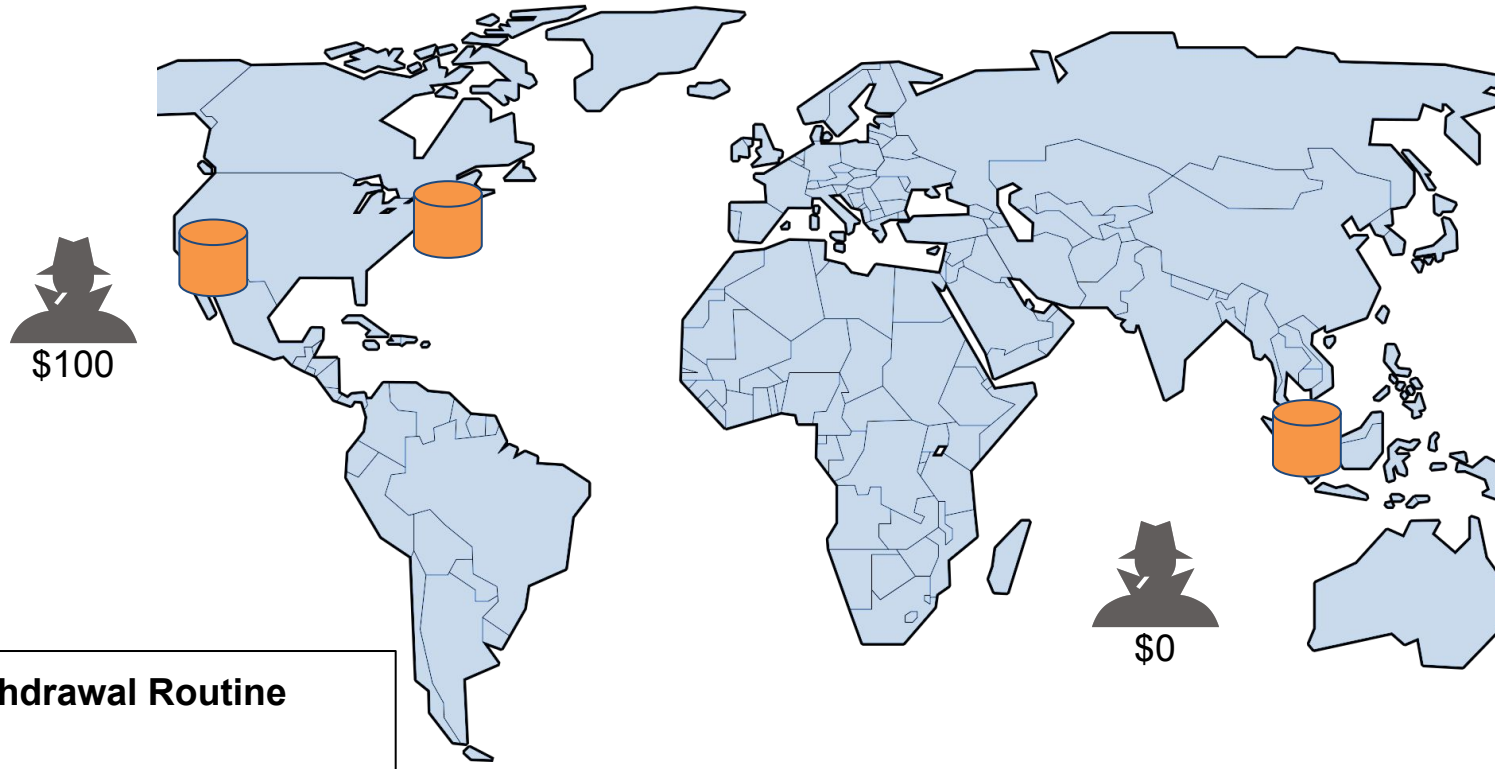
P3.2: Motivation - Consistency

Withdrawal Routine

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

Account	Balance
xxxxx-4437	**$100**

P3.2: Motivation - Consistency

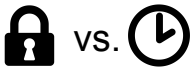


Withdrawal Routine

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

Account	Balance
xxxxx-4437	**$0**

P3.2: Consistency Models

Tradeoff: 

- Strict
- Strong
- Sequential
- Causal
- Eventual

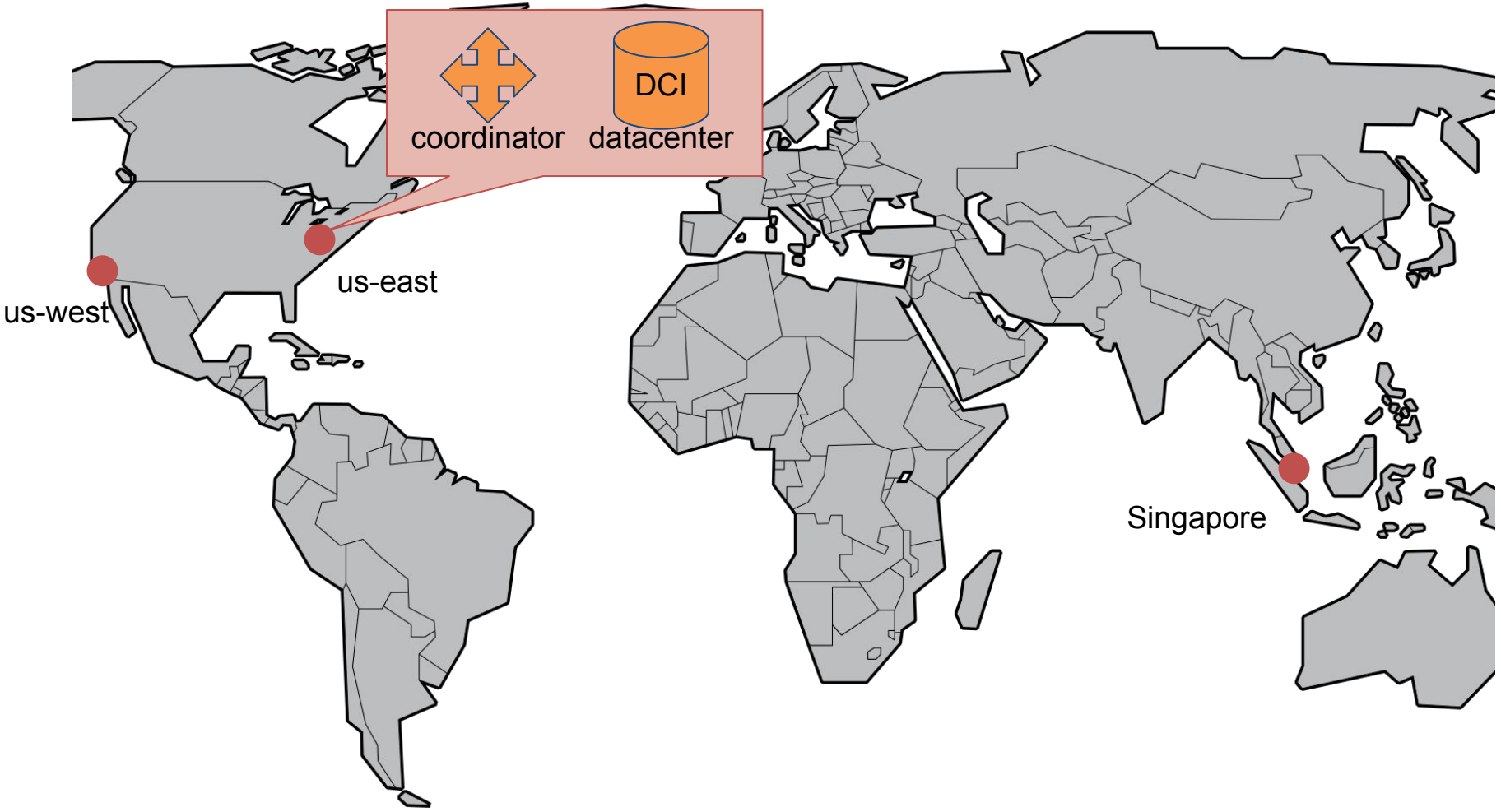
P3.2: Strong Consistency

- Every operation receives a global timestamp order.
  - Typically the order in which they arrive at the coordinator
- Operations must be ordered according to timestamps.
- At any given point of time, all clients should read the same data from any datacenter replica.

P3.2: Eventual Consistency (Bonus)

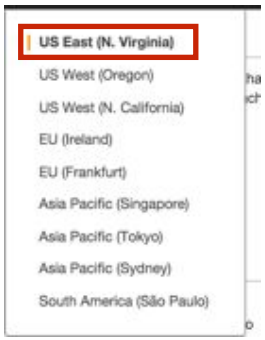
- Writes are performed in the order they are received at each replica
    - Operations may not be blocked for replica consensus
- Clients that request data may receive multiple versions of the data, or stale data
    - Left to the application to resolve

P3.2: Architecture



P3.2: Task 2

- Launch Coordinators and DCIs
  - All in **us-east**, we simulate global latencies
- Implement the Coordinators
  - Strong Consistency
  - Eventual Consistency

- 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*)

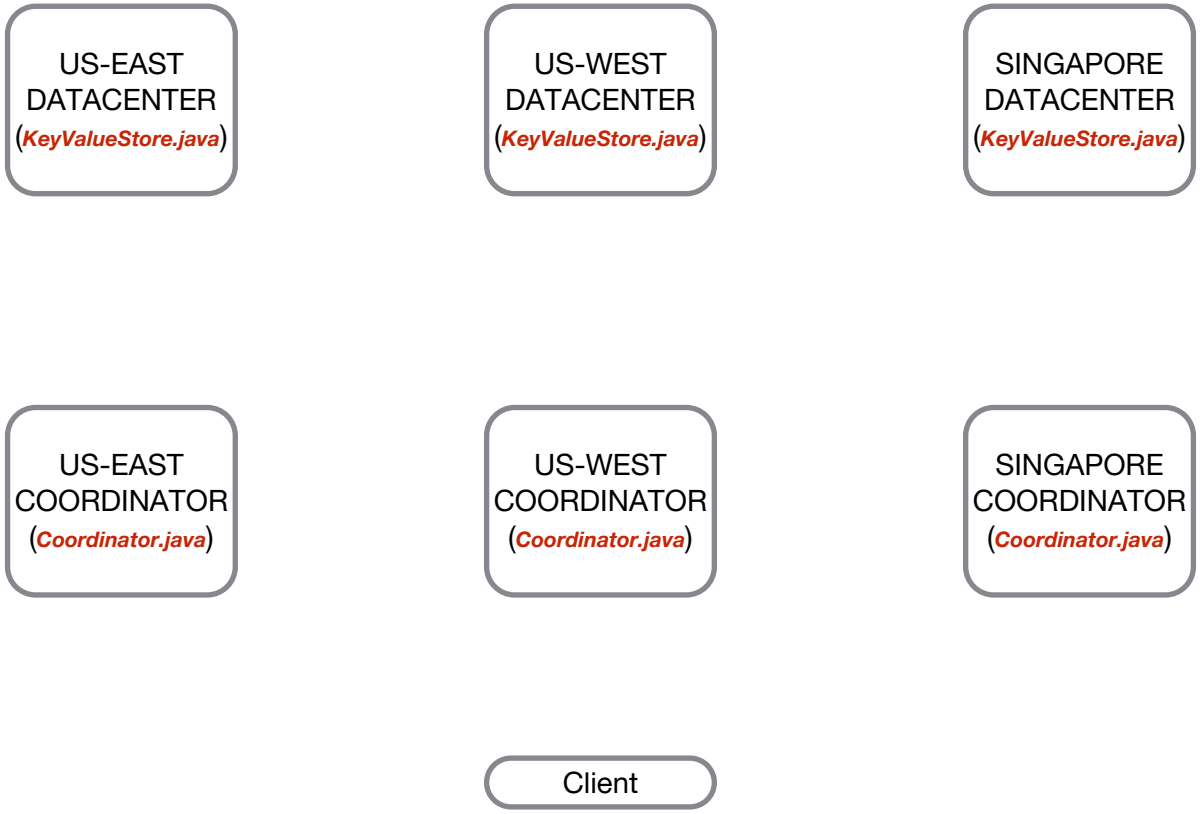
US-WEST COORDINATOR (*Coordinator.java*)

SINGAPORE COORDINATOR (*Coordinator.java*)

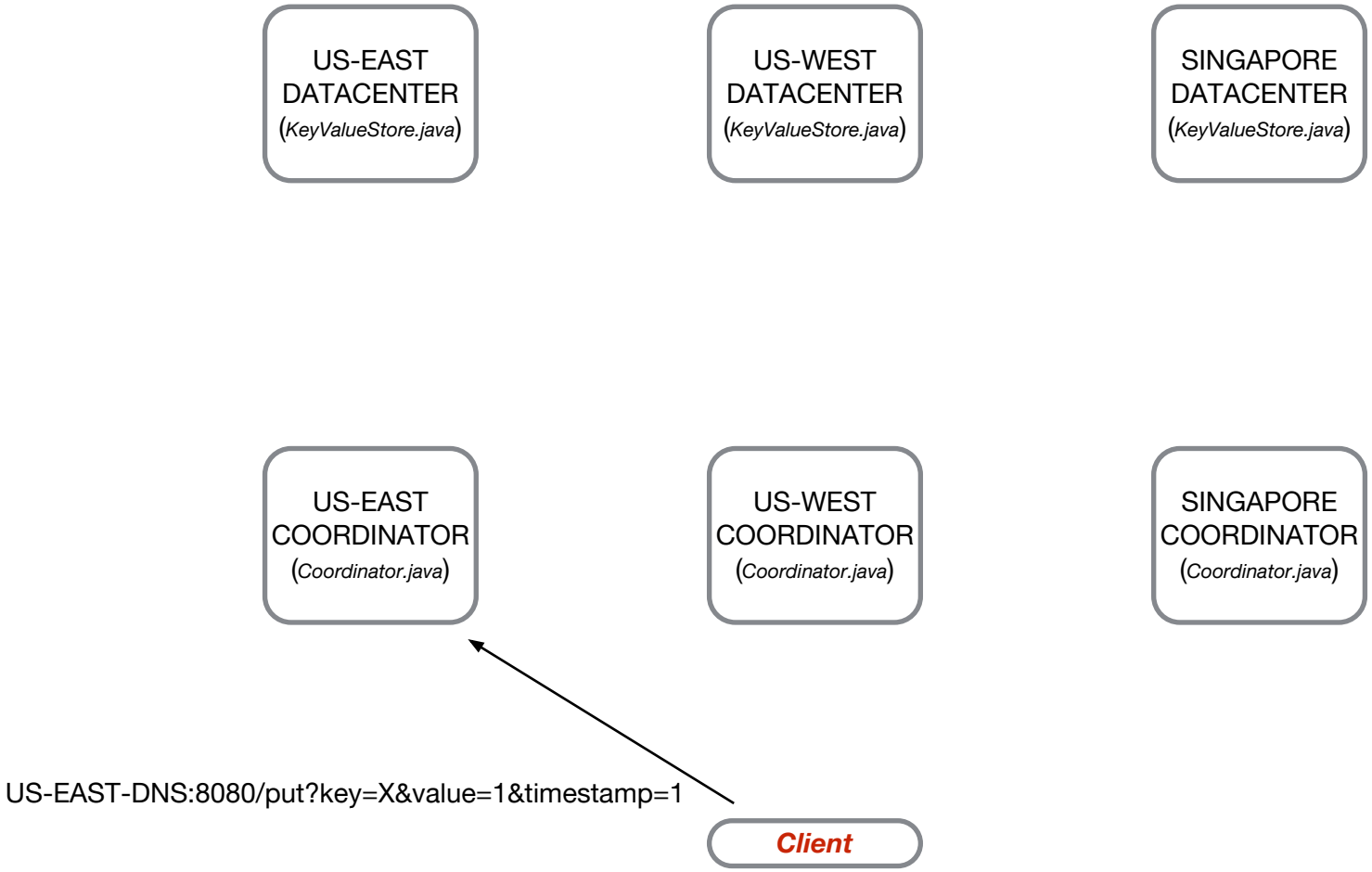
Client

P3.2 TODO:

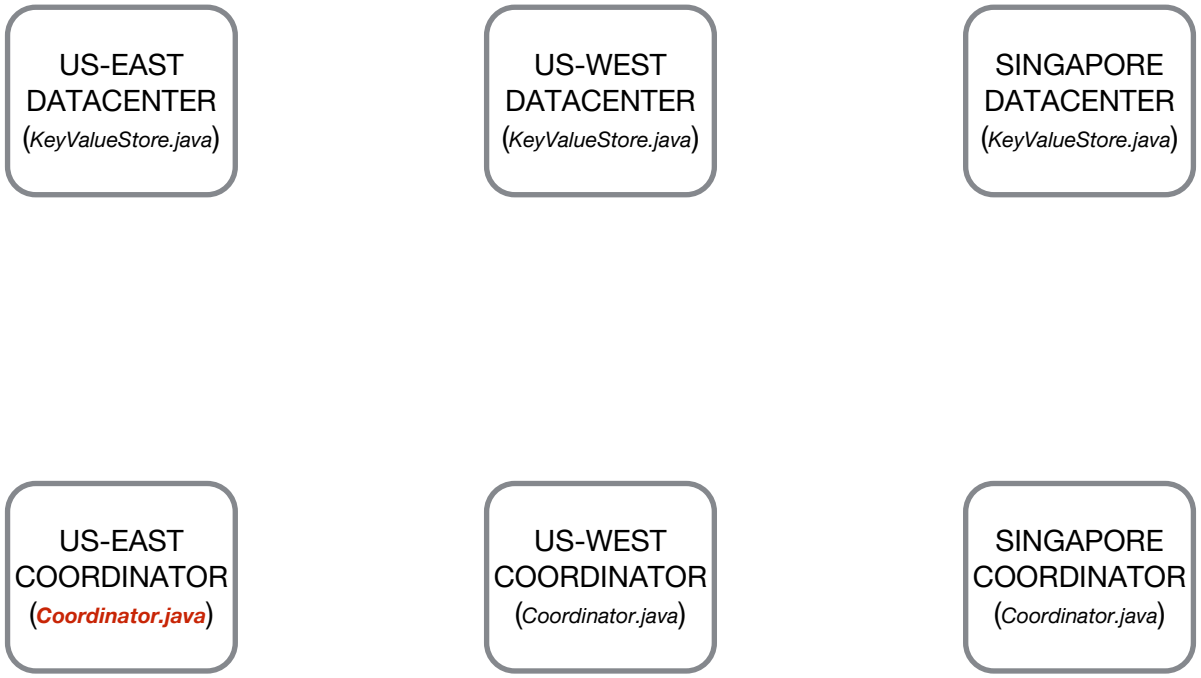
KeyValueStore.java (on the DC) and Coordinator.java (on the coordinator).



Example workflow for PUT request in strong consistency

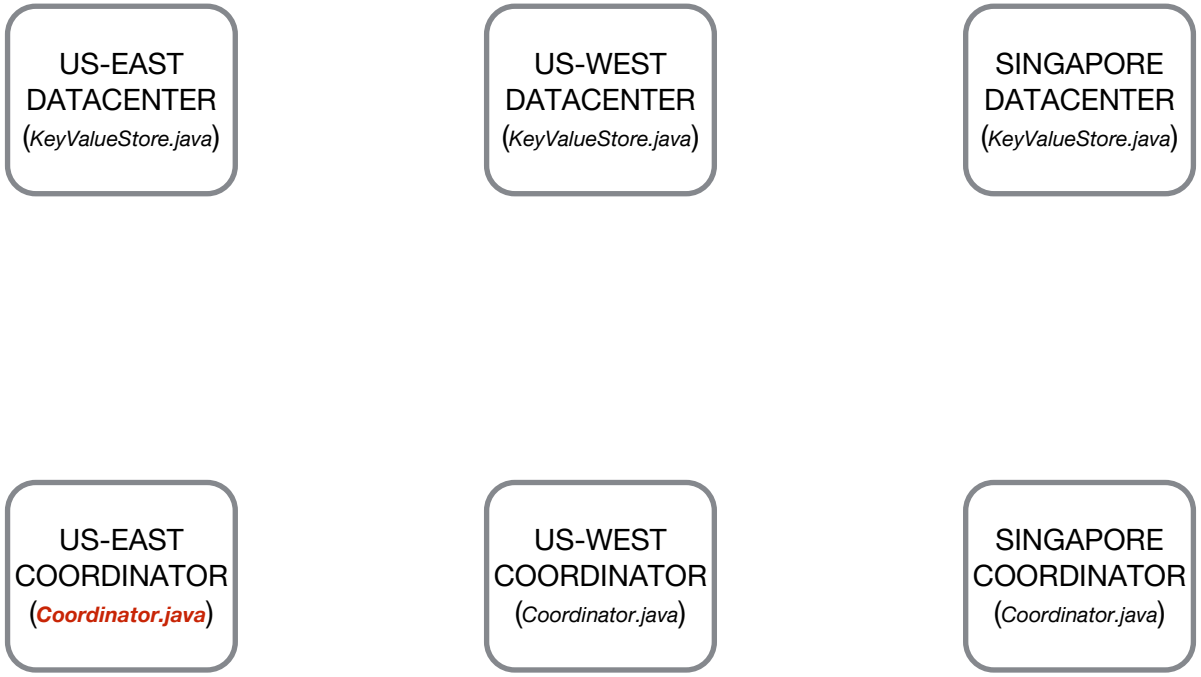


Example workflow for PUT request in strong consistency



Example workflow for a PUT request in strong consistency

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



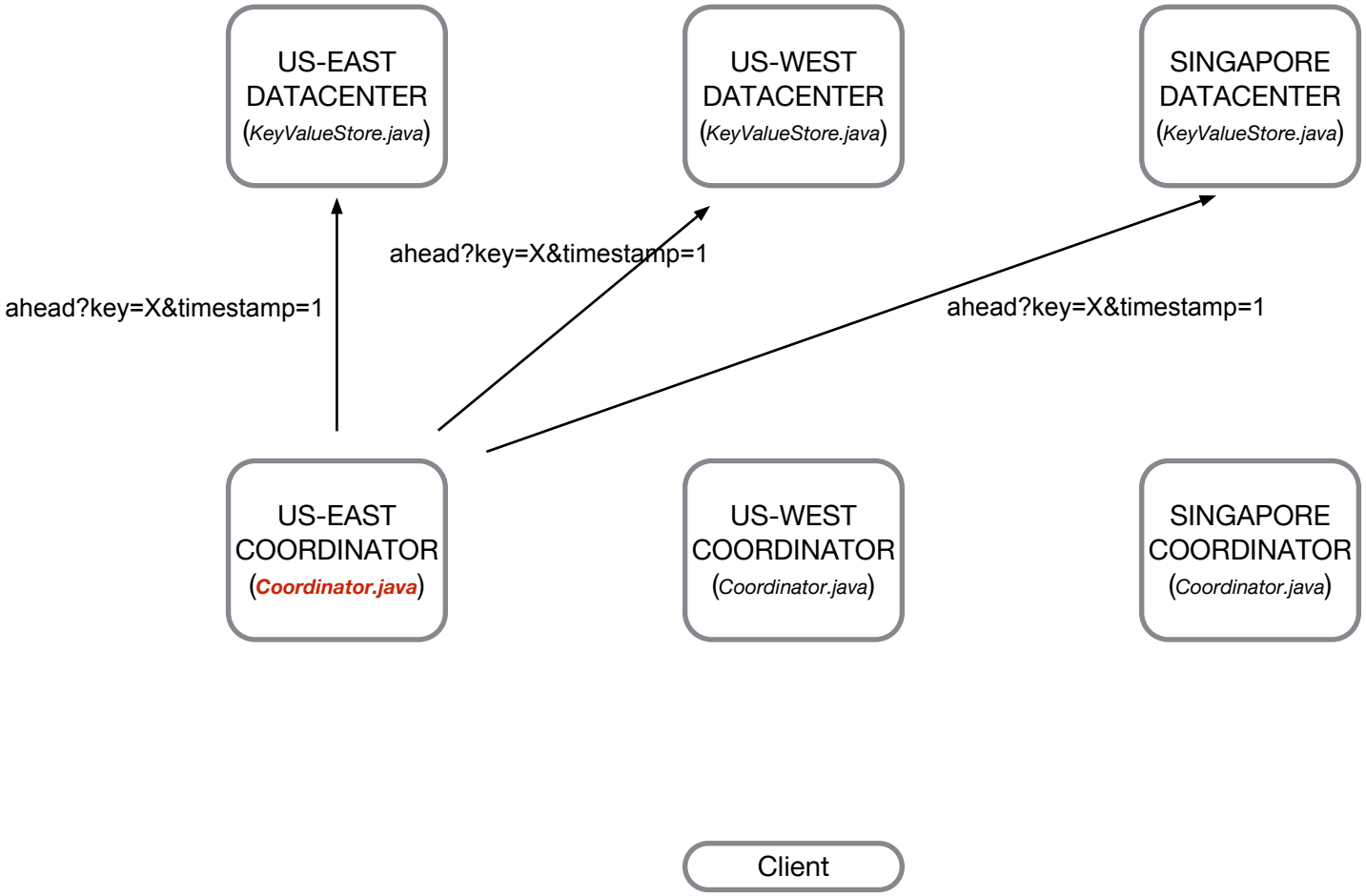
You should call KeyValueLib.AHEAD(“X”,1) to notify all 3 datacenters of this PUT request.
Resulting behavior may include:

- 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 PUT request in strong consistency

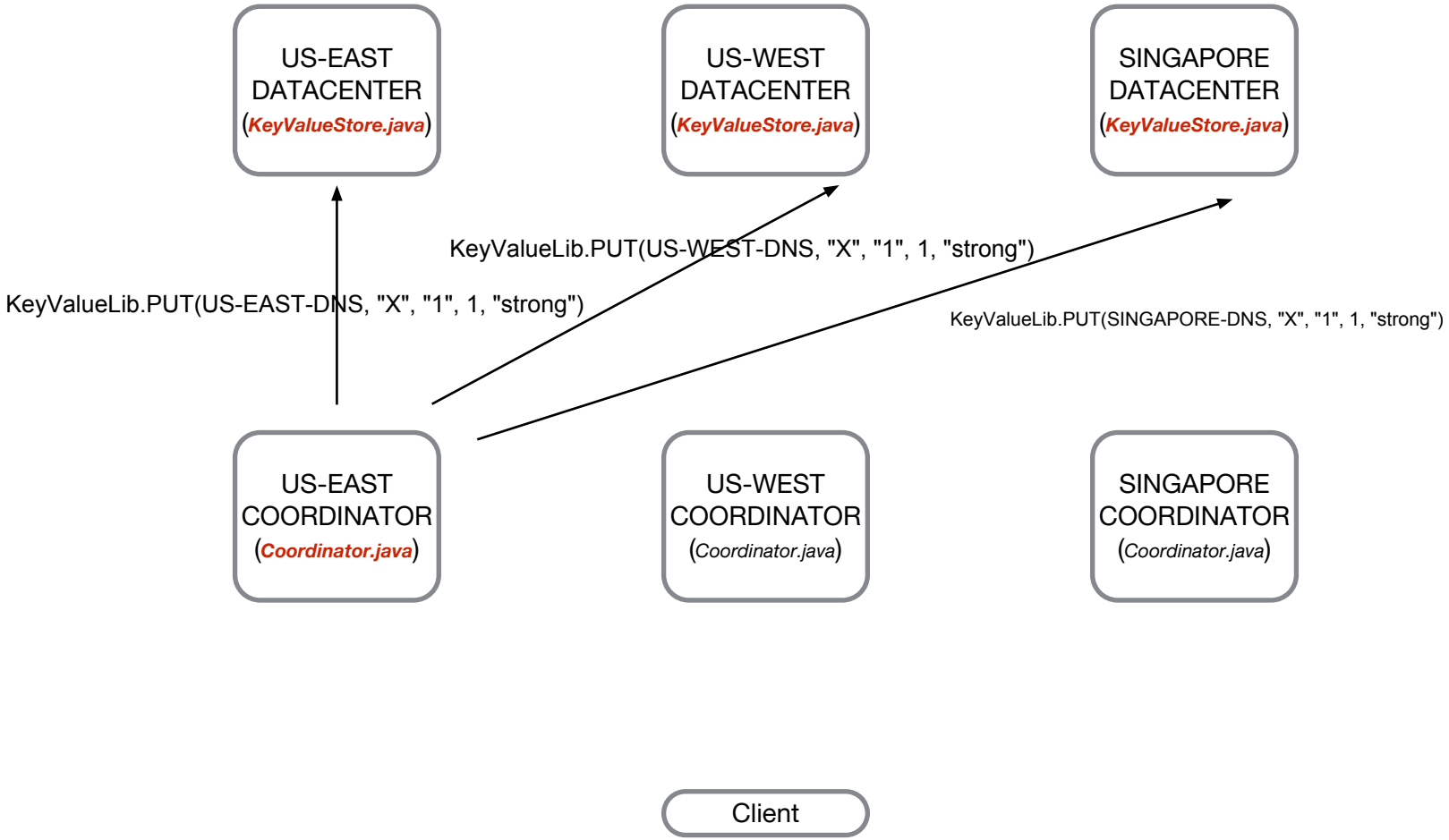
- If US-EAST is responsible for key “X”



Example workflow for PUT request in strong consistency

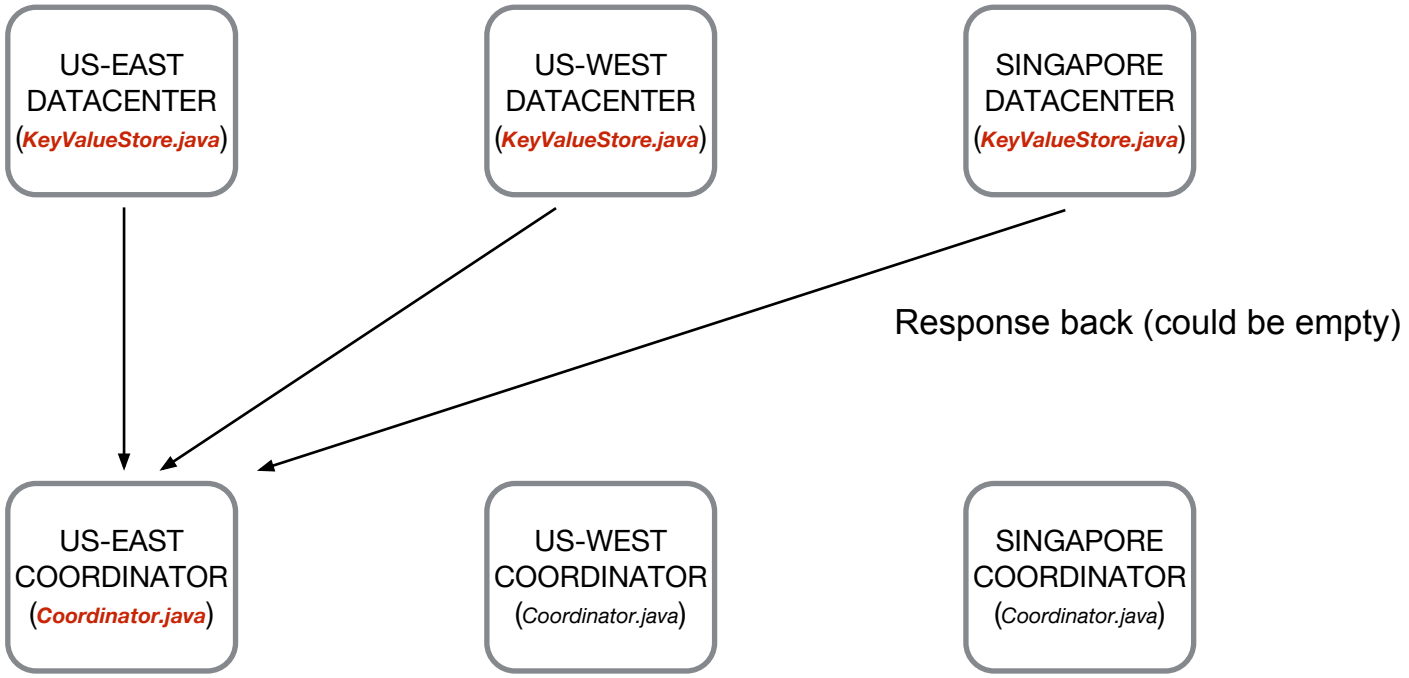
- If US-EAST is responsible for key “X”

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



Example workflow for PUT request in strong consistency

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



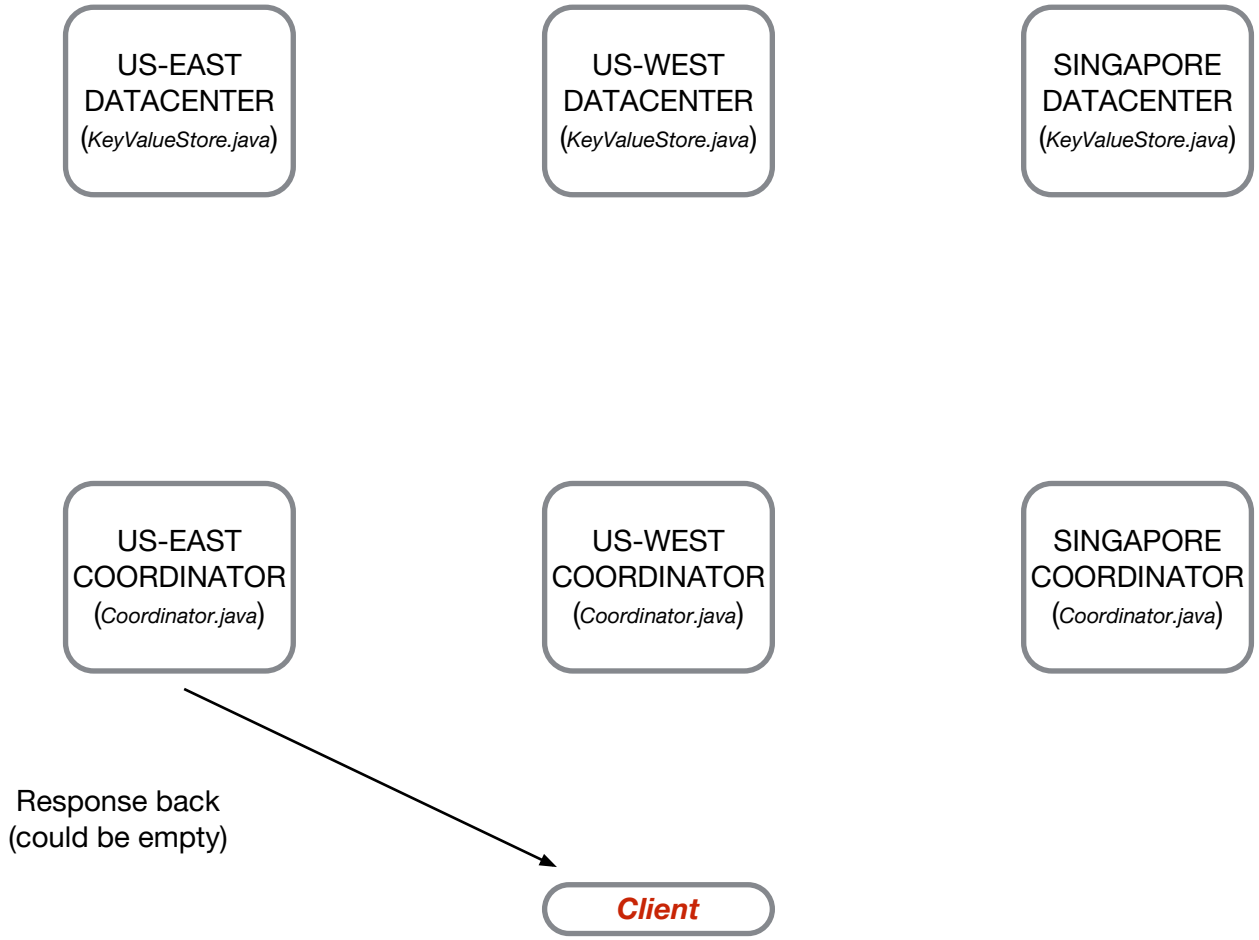
Finally, you should call KeyValueLib.COMPLETE(“X”,1) to notify all 3 datacenters of this request’s completion.
Resulting behavior may include:

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

Client

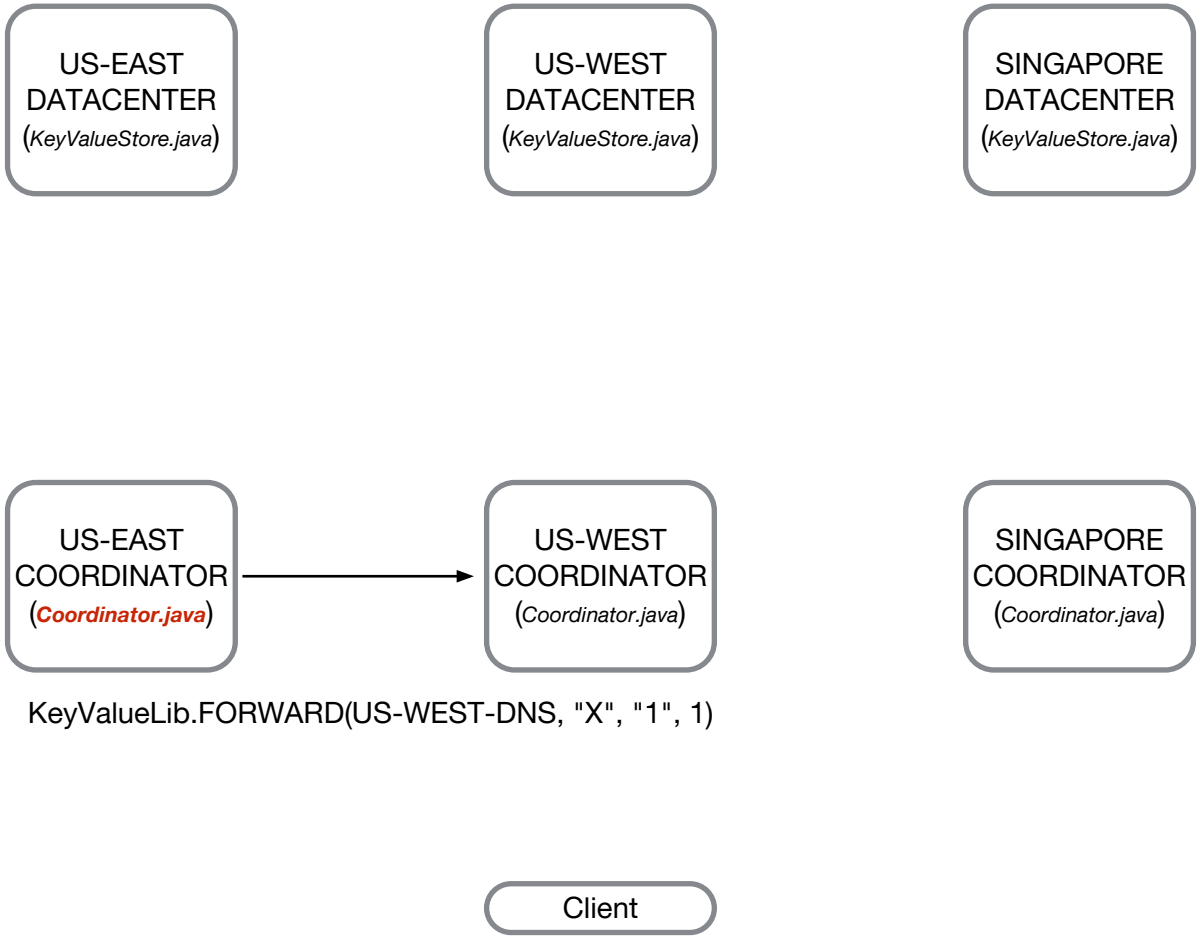
Example workflow for PUT request in strong consistency

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



Example workflow for PUT request in strong consistency

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



More Hints:

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

Suggestions:

- You should first know the difference between the 2 policies before writing your code.
- Think about possible race conditions.
- Read the hints on the TPZ handout carefully.
- Don’t modify any class except Coordinator.java and KeyValueStore.java.
- There are a number of various performance enhancements and optimizations you can do if you want to go for the bonus task. Try applying the different techniques you’ve learned

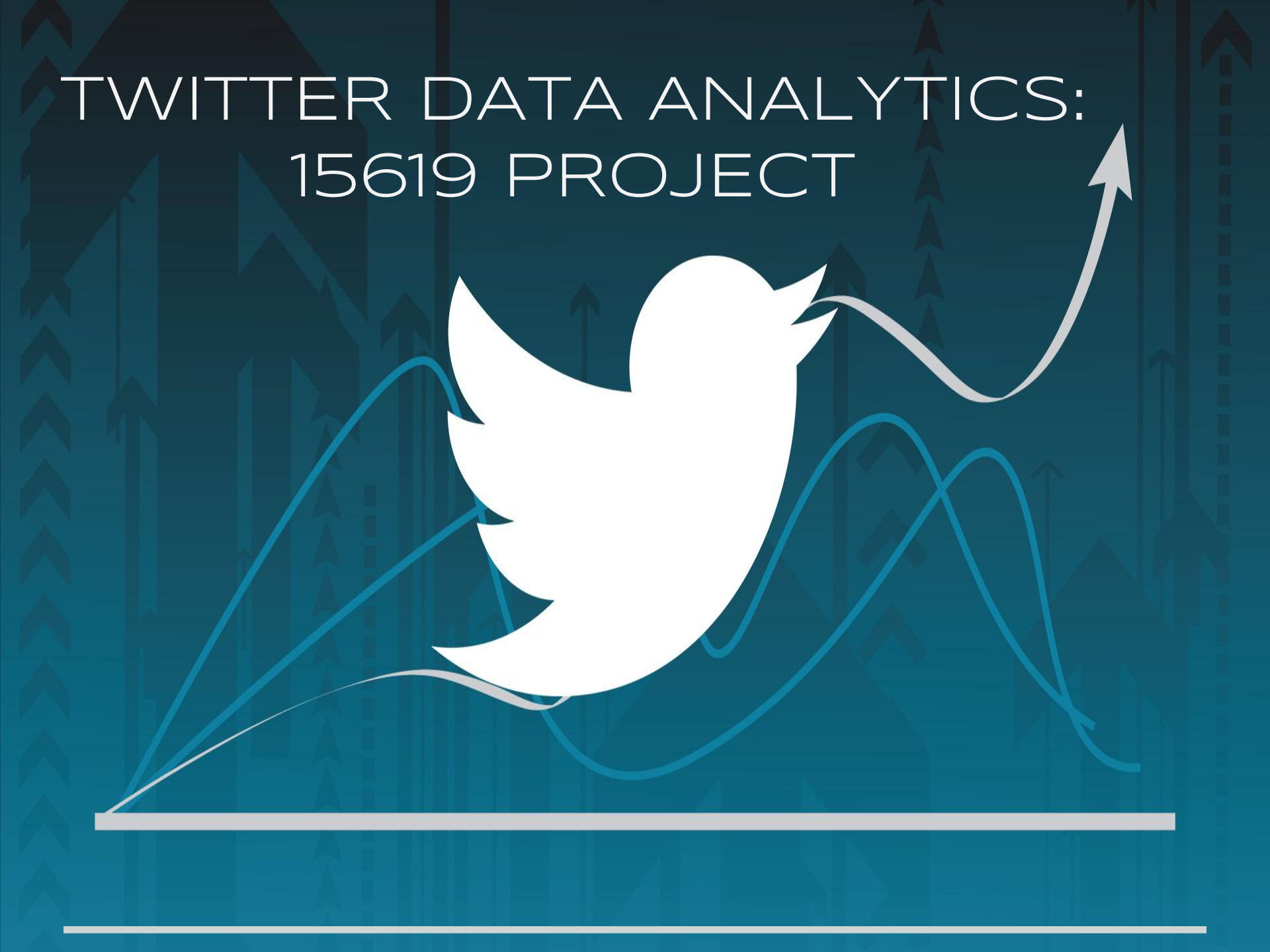
How To Test:

- 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)

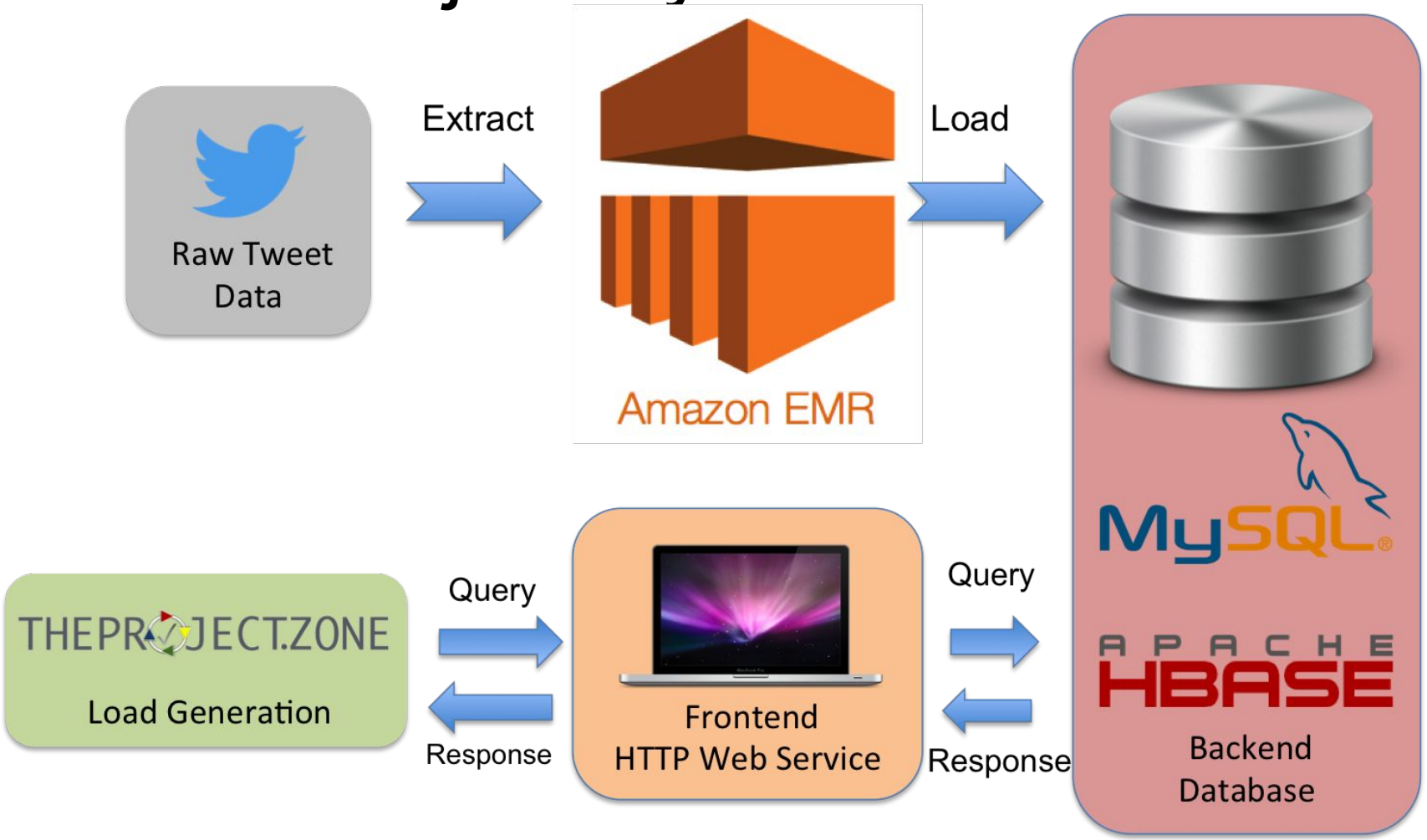
- Use “./consistency_checker strong”, or “./consistency_checker eventual” to test your implementation of each consistency. (Our grader uses the same checker)

- If you want to test one simple PUT/GET request, you could directly enter the request in your browser.

TWITTER DATA ANALYTICS: 15619 PROJECT



15619 Project System Architecture

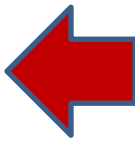


- Web server architectures
- Dealing with large scale real world tweet data
- HBase and MySQL optimization

15619 Project

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

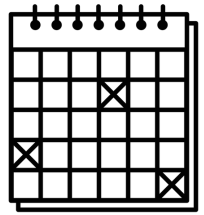
CONFIRM YOUR AWS ACCOUNT AND TEAM INFO



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

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

15619 Project Time Table



Phase (and query due)	Start	Deadline	Code and Report Due
Phase 1 • Q1	Monday 10/10/2016 00:00:01 EST	Sunday 10/23/2016 23:59:59 ET	
• Q2		Sunday 10/30/2016 23:59:59 ET	Tuesday 11/01/2016 23:59:59 ET
Phase 2 • Q1, Q2, Q3	Monday 10/31/2016 00:00:01 ET	Sunday 11/13/2016 15:59:59 ET	
Phase 2 Live Test (Hbase/MySQL) • Q1, Q2, Q3	Sunday 11/13/2016 18:00:01 ET	Sunday 11/13/2016 23:59:59 ET	Tuesday 11/15/2016 23:59:59 ET
Phase 3 • Q1, Q2, Q3, Q4	Monday 11/14/2016 00:00:01 ET	Sunday 12/04/2016 15:59:59 ET	
Phase 3 Live Test • Q1, Q2, Q3, Q4	Sunday 12/04/2016 18:00:01 ET	Sunday 12/04/2016 23:59:59 ET	Tuesday 12/06/2016 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)**

15619 Project Phase 1

- Two queries
  - Q1: Pure front end
  - Q2: ETL + back end + front end, do both MySQL (relational DBMS) and HBase (NoSQL)

- Grading
  - Submit on TPZ, you will get several numbers:
    - Error Rate, Correctness and RPS
  - Higher RPS, higher correctness, lower error rate ⇒ higher grade
  - Q1 is 25% of phase 1, Q2 MySQL is 25% of phase 1, Q2 HBase is 25% of phase 1, report is 25% of phase 1

15619 Project, Phase 1, Q1

- **Step 1:** Compare different front-end frameworks
- **Step 2:** Deploy the front-end
- **Step 3:** Perform decryption of a secret message

Pure front end, no database needed.

Need to consider scaling horizontally

15619 Project, Phase 1, Q2

- **Step 1:** Extract tabular data from raw tweets
  - Input file: JSON Tweets (approx. 1 TB)
  - Consider using a MapReduce Job for ETL
    - ETL is expensive and there's the potential for errors, so plan carefully, test on smaller data sets
    - Start early, or no time to optimize the backend
- **Step 2:** Load the data into HBase **and** MySQL (both!)
- **Step 3:** Deploy
  - a web service for handling HTTP requests, responds with data from the backend
  - an optimized backend (MySQL and HBase)

Higher throughput = More points

Winner gets grades, fame (?), job (?)

Hints

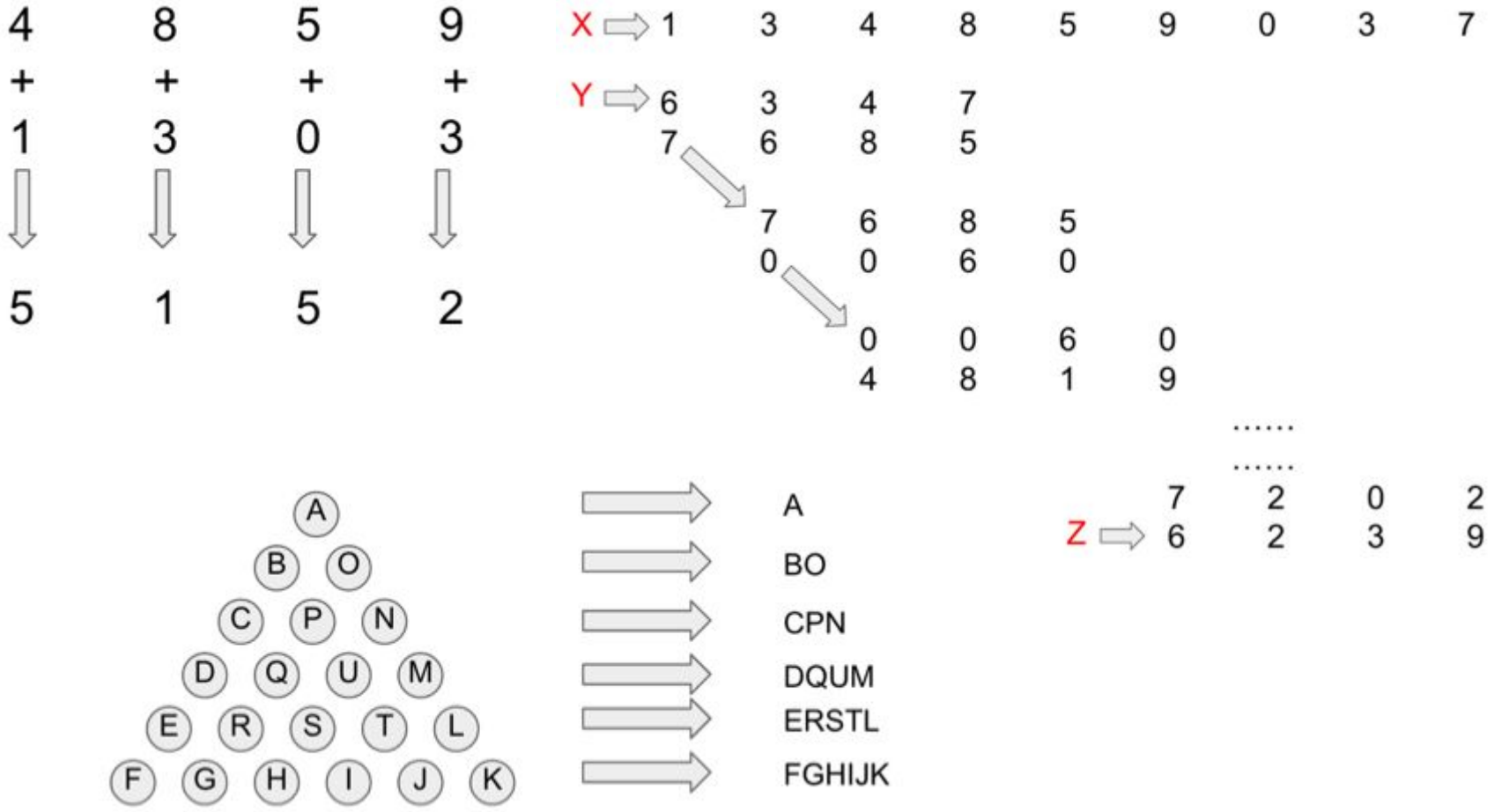
- Read the write-up carefully (read more than once)
- You can test only if you have a **front end**
- ETL has many corner cases, can be time consuming and expensive
    - Start early (from the first day), your backend will be meaningless if you have incorrect data
    - The reference server is your friend
- Big data challenge will easily eat up your time and money if you are careless. Think, calculate, & test before you launch an EMR cluster with 20 machines

Reminder

- You have a total budget of $50 for Phase 1
- Your system should not cost more than $0.95 per hour, this includes (see write-ups for details):
  - EC2 on demand instance cost
    - even if you use spot instances, we will calculate your cost using the on demand instance price
  - Storage cost
  - ELB cost

Q1 : Heartbeat and Authentication

- Task: Decrypt messages



Q2 : Text Cleaning And Analysis

- ETL Task
  - Similarity Score : Dot product of word vectors
  - Impact Score Calculation
    - Influential Score
    - Sentiment Score
      - Effective Word Count
  - Text Censoring
  - Reference Server and Data Set

Q2 : Text Cleaning And Analysis

- Request
  - userid1
  - userid2
  - n
- Return
  - Similarity Score
  - Shared Words
  - Top n tweets from each user

Deadlines : This Week

- Quiz 6: Unit 3 - Storage and network virtualization

  Due: 10/14/2016 11:59PM Pittsburgh

- Project 3.2: Multithreaded Programming and Consistency

  Due: 10/16/2016 11:59PM Pittsburgh