15-319 / 15-619 Cloud Computing

Recitation 9

March 15th, 2016

Overview

- **Administrative issues**
  Office Hours, Piazza guidelines
- **Last week's reflection**
  Project 3.2, OLI Unit 4, Module 14, Quiz 7
- **This week's schedule**
  - 15619 Project - Query 1 & 2 - March 16th
  - Quiz 8 - March 18th (Unit 4, Module 15)
  - Project 3.3 - March 20th

Last Week : A Reflection

- Content, Unit 4 - Module 14:
  - Cloud Storage - Big picture
  - Quiz 7 completed

- P3.2: You began exploring distributed storage systems
  - Handled requests from a single front end coordinator and 3 backend datastores
  - Basics of concurrency
  - Learned about Sharding and Replication

This Week: Content

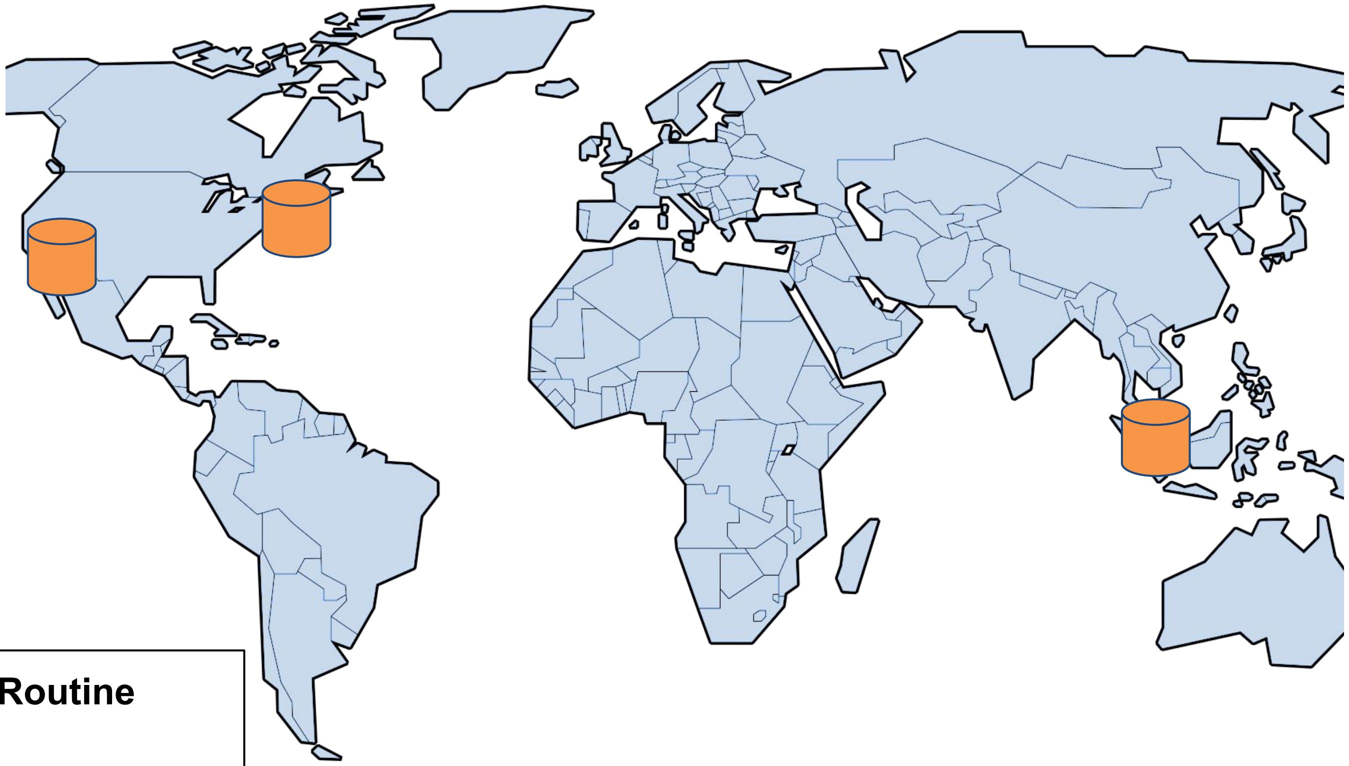
UNIT 4: Cloud Storage

- Module 14: Cloud Storage
  - Quiz 7 - Introduction to Cloud Storage
- Module 15: Case Studies: Distributed File Systems
  - Quiz 8: Distributed File Systems
    - Friday, March 18th
- Module 16: Case Studies: NoSQL Databases
- Module 17: Case Studies: Cloud Object Storage
  - Quiz 9: NoSQL and Object Stores

Project 3 Weekly Modules

- P3.1: Files, SQL and NoSQL
- P3.2: Sharding and Replication
- **P3.3: Consistency**
    - **Due Sunday, March 20th**
- P3.4: Social network and heterogeneous back end storage
- P3.5: Data warehousing and OLAP

P3.3: Motivation - Consistency

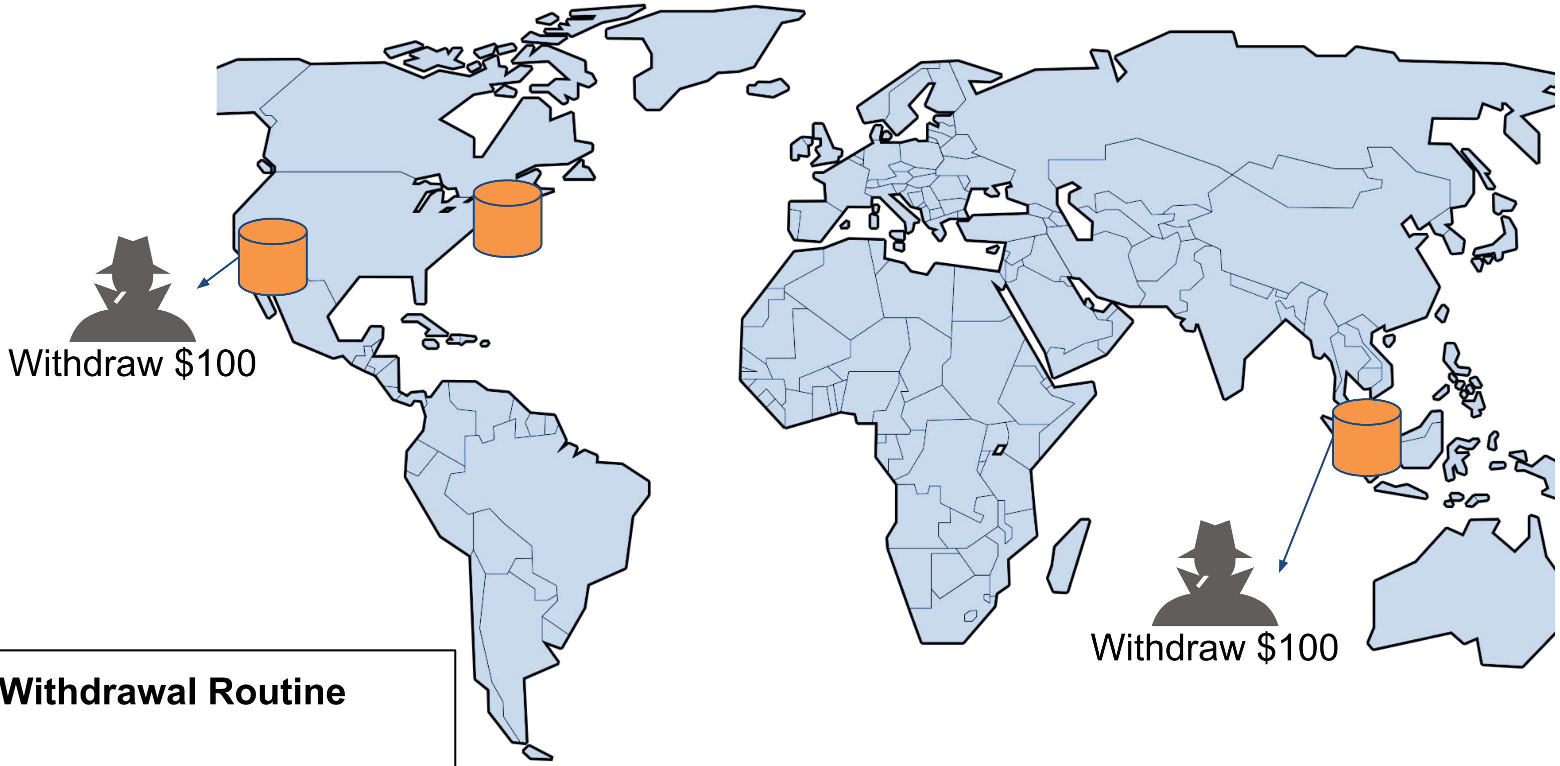


Withdrawal Routine

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

Account	**Balance**
xxxxx-4437	$100

P3.3: Motivation - Consistency

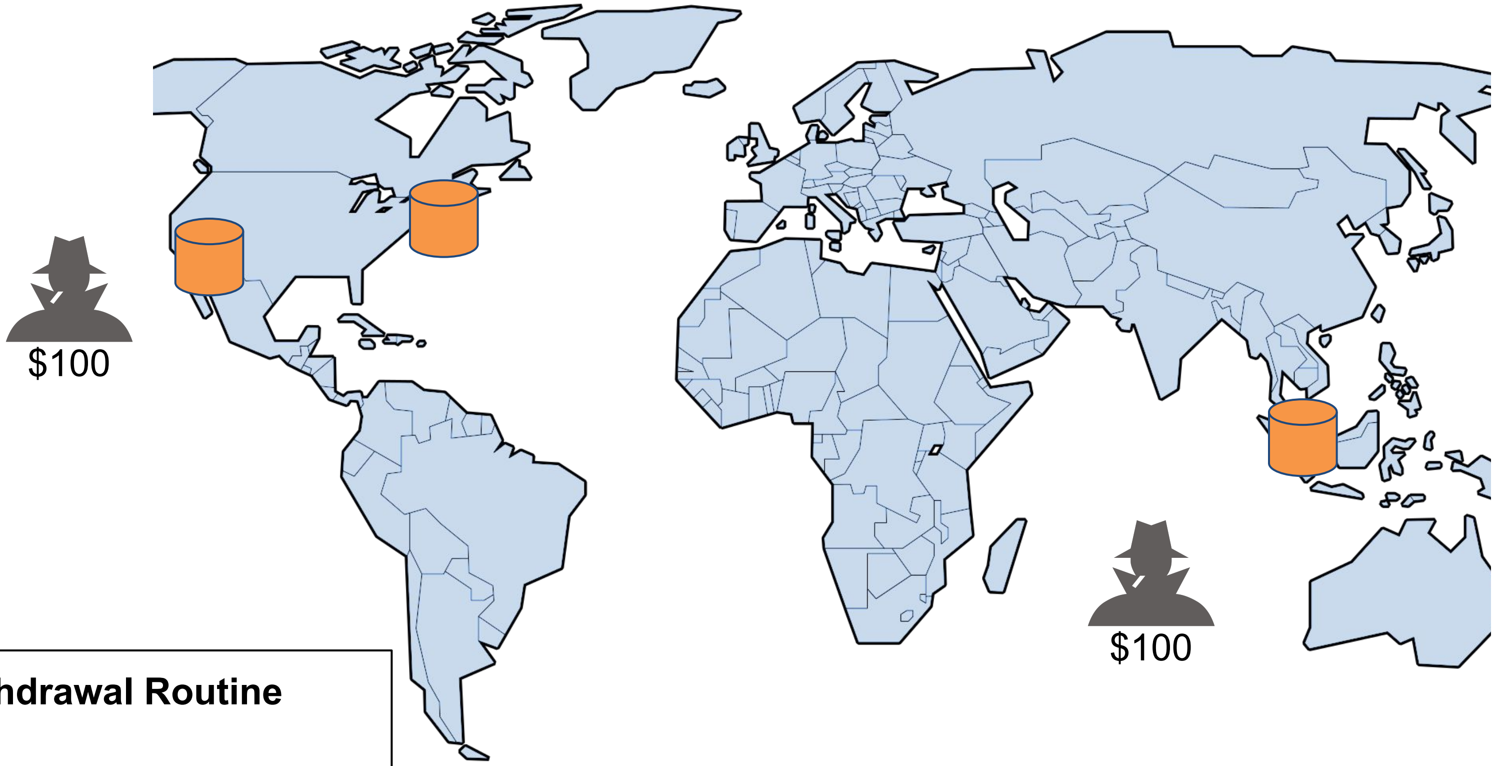


Withdrawal Routine

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

Account	Balance
xxxxx-4437	$100

P3.3: Motivation - Consistency



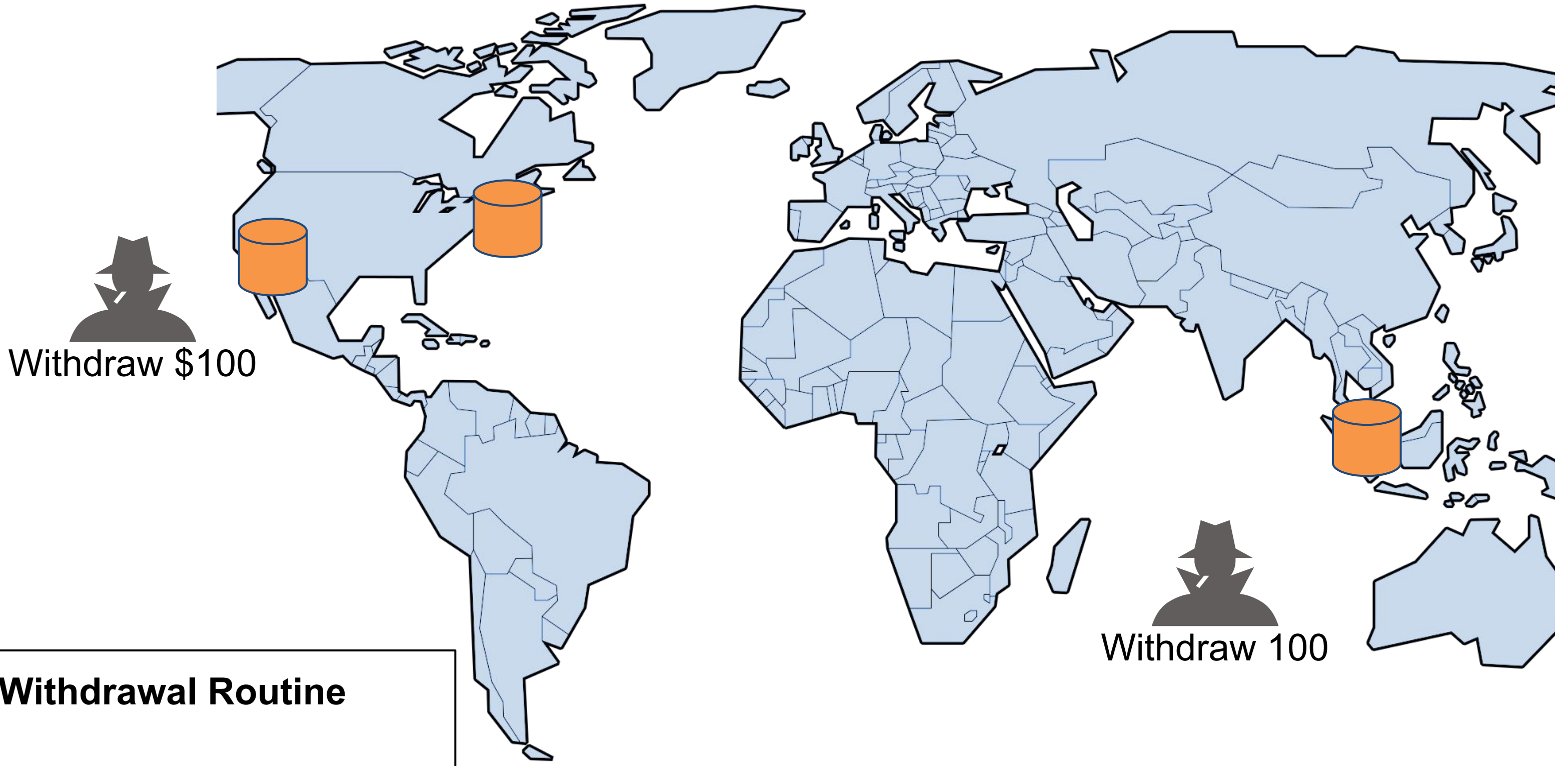
Withdrawal Routine

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

Account	Balance
xxxxx-4437	**$0**

Bank lost $100

P3.3: Motivation - Consistency

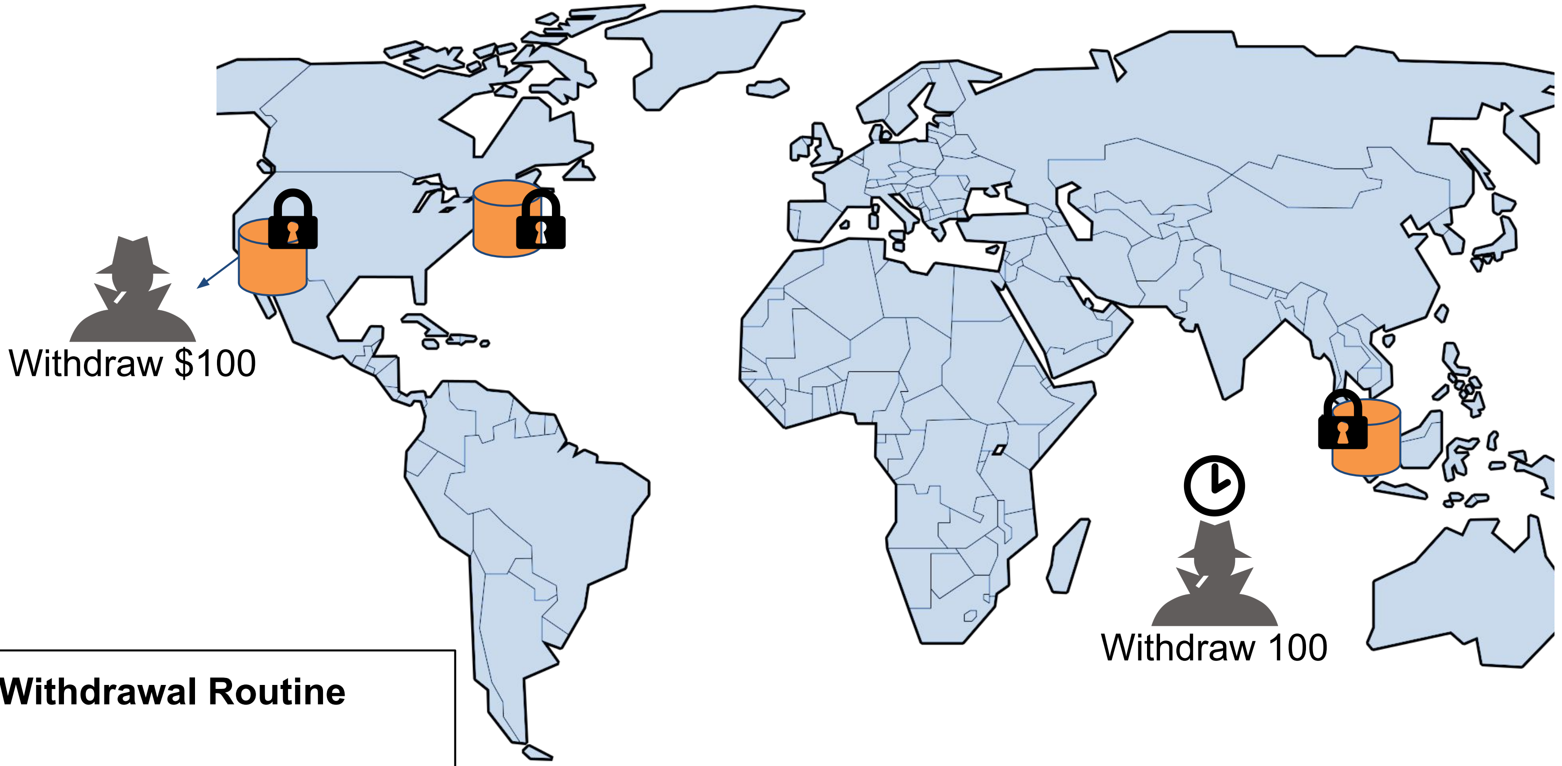


Withdrawal Routine

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

Account	Balance
xxxxx-4437	**$100**

P3.3: Motivation - Consistency

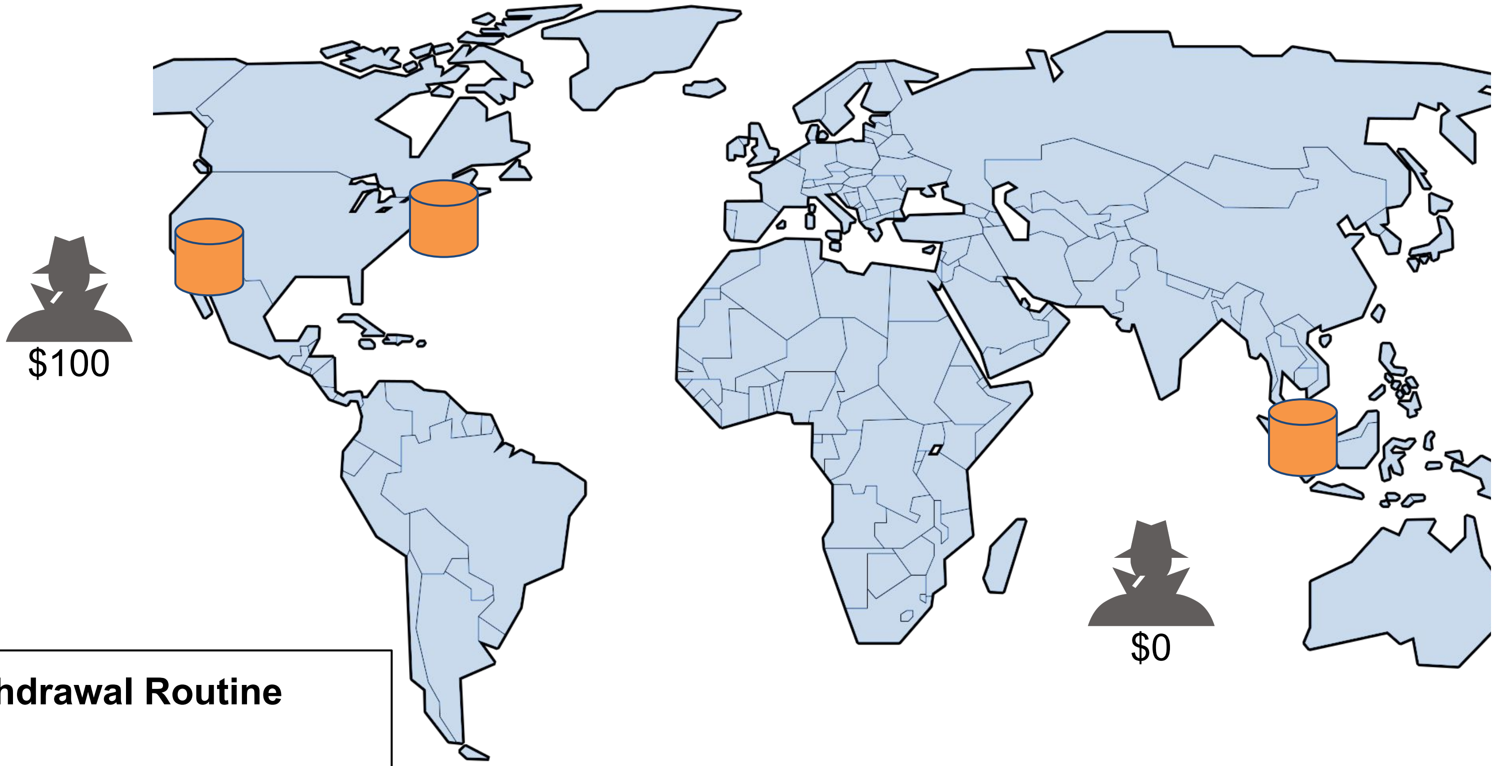


Withdrawal Routine

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

Account	Balance
xxxxx-4437	**$100**

P3.3: Motivation - Consistency



Withdrawal Routine

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

Account	Balance
xxxxx-4437	**$0**

P3.3: Consistency Models

Tradeoff: 🔒 vs. 🕒

- Strict
- Strong
- Sequential
- Causal
- Eventual

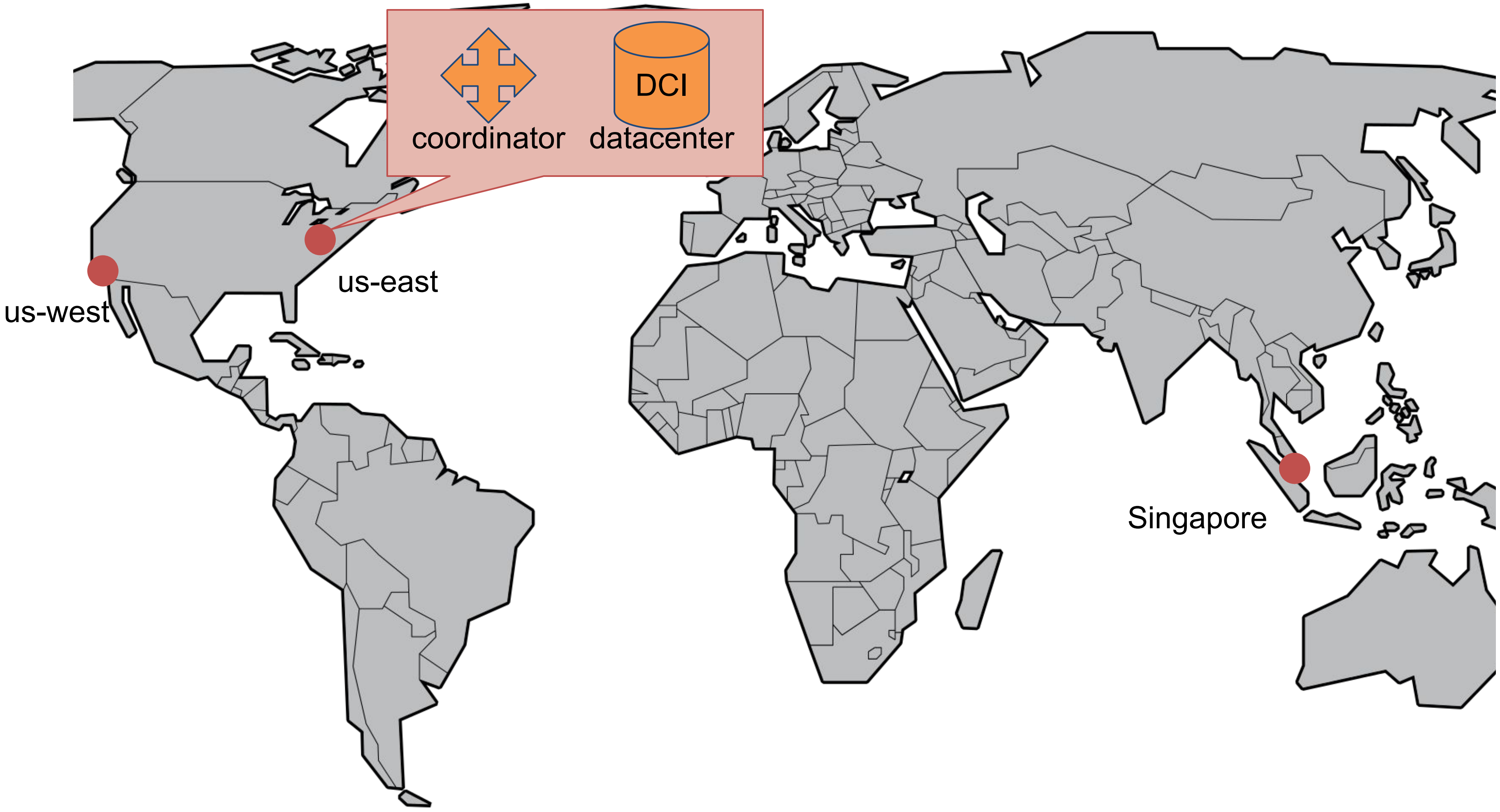
P3.3: Strong Consistency

- 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.

P3.3: Eventual Consistency

- 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

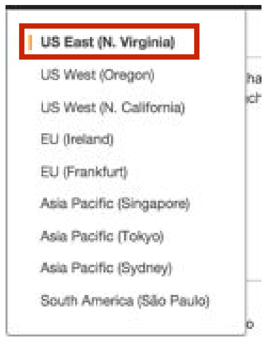
P3.3: Architecture



P3.3: Your Task

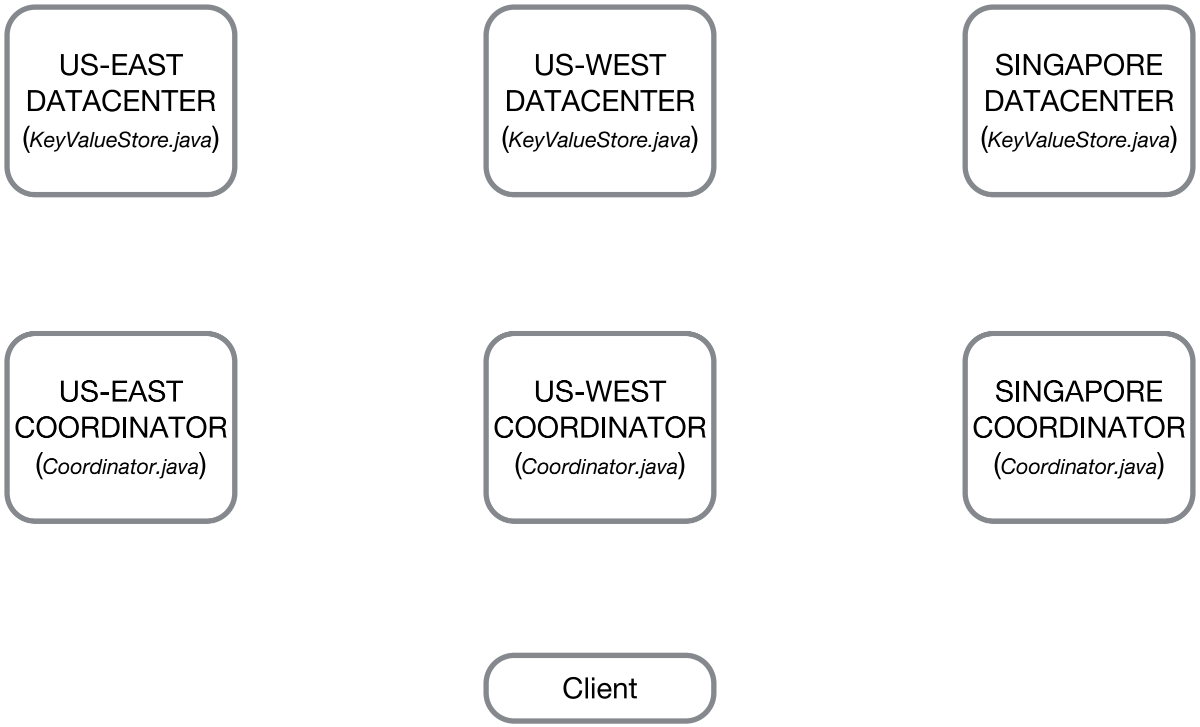
- Launch Coordinators and DCIs
  - All in **us-east**, we simulate global latencies
- Implement the Coordinators
  - Strong Consistency
  - Eventual Consistency

P3.3: Hints

- Launch a total of 7 machines (3 data centers, 3 coordinators and 1 client)
- 

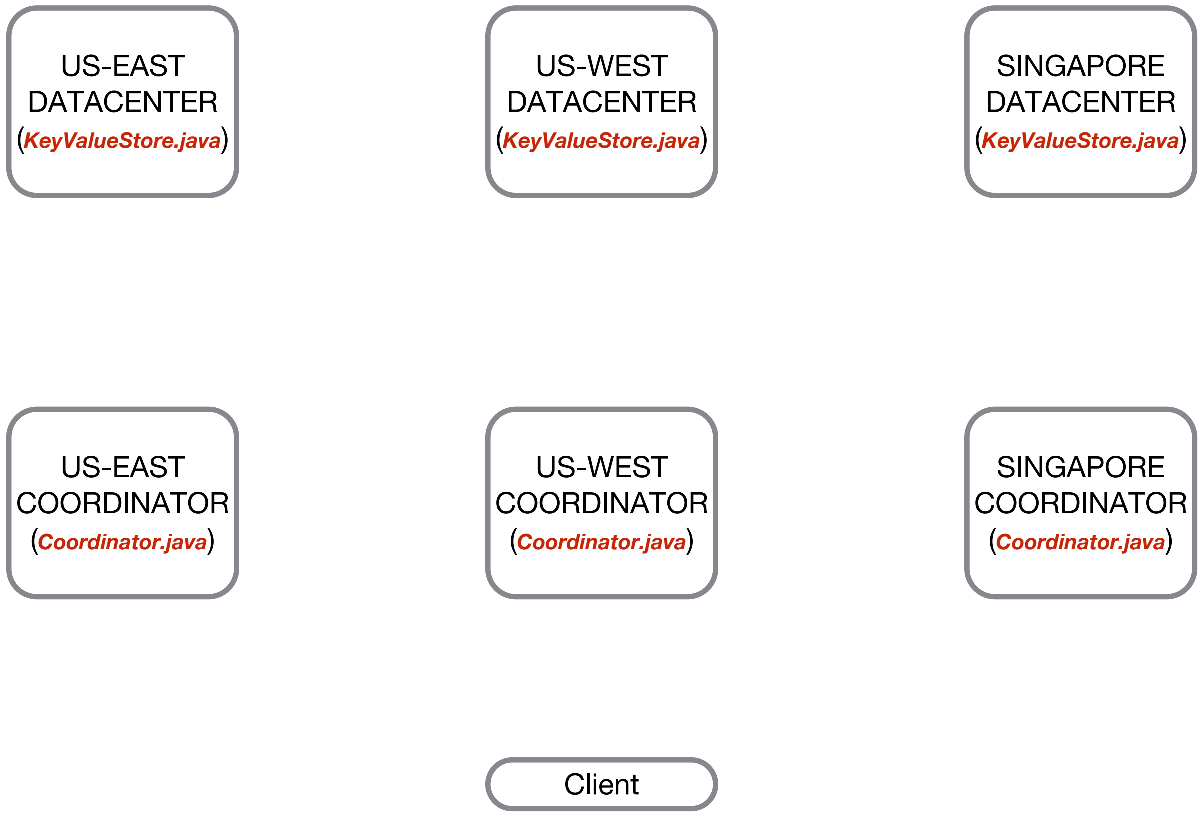
All machines should be launched in US East region.

The "US East" here has nothing to do with the simulated location of datacenters and coordinators in the project.

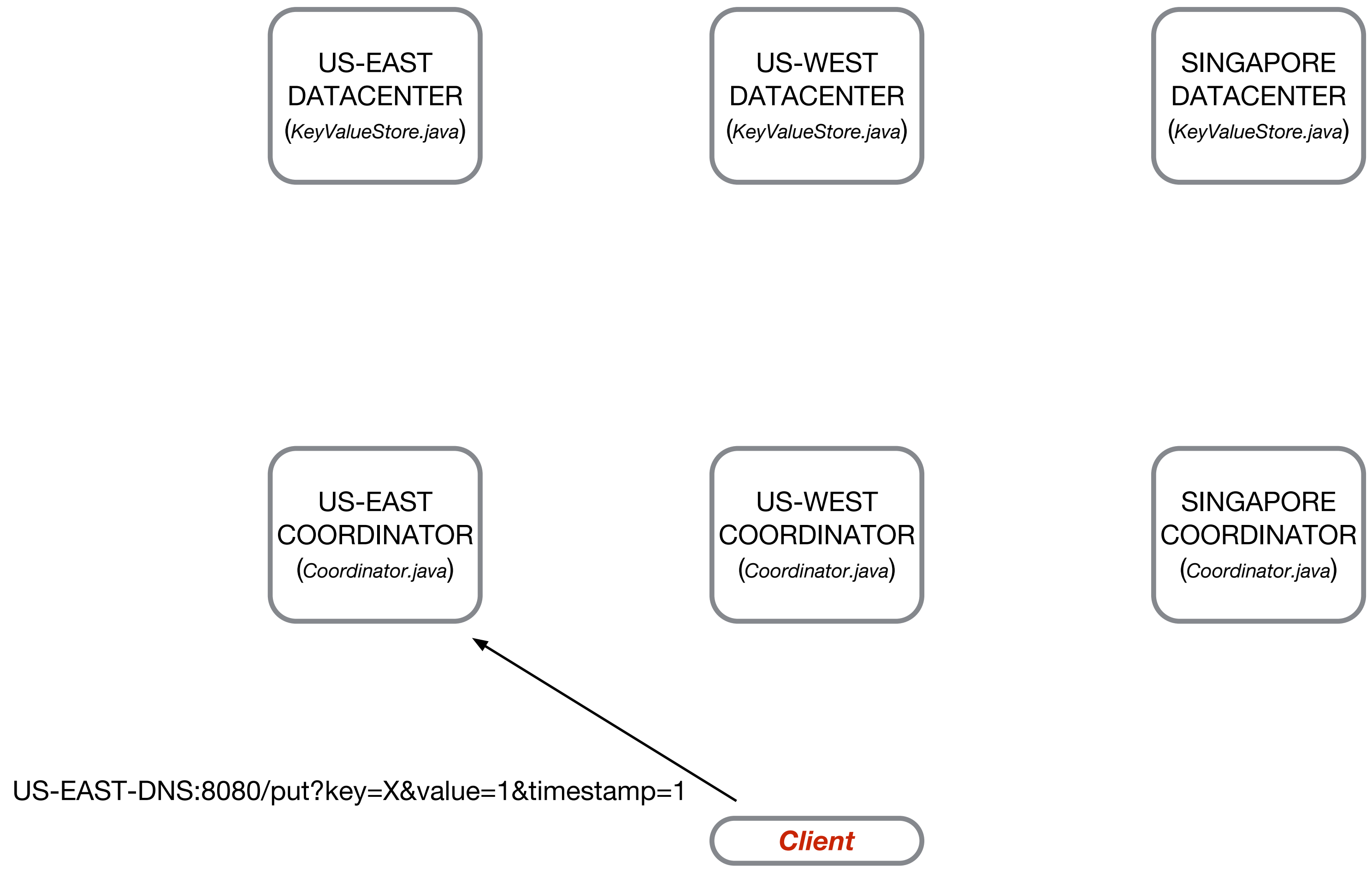

US-EAST DATACENTER (*KeyValueStore.java*)
US-WEST DATACENTER (*KeyValueStore.java*)
SINGAPORE DATACENTER (*KeyValueStore.java*)
US-EAST COORDINATOR (*Coordinator.java*)
US-WEST COORDINATOR (*Coordinator.java*)
SINGAPORE COORDINATOR (*Coordinator.java*)
Client

P3.3 TODO:

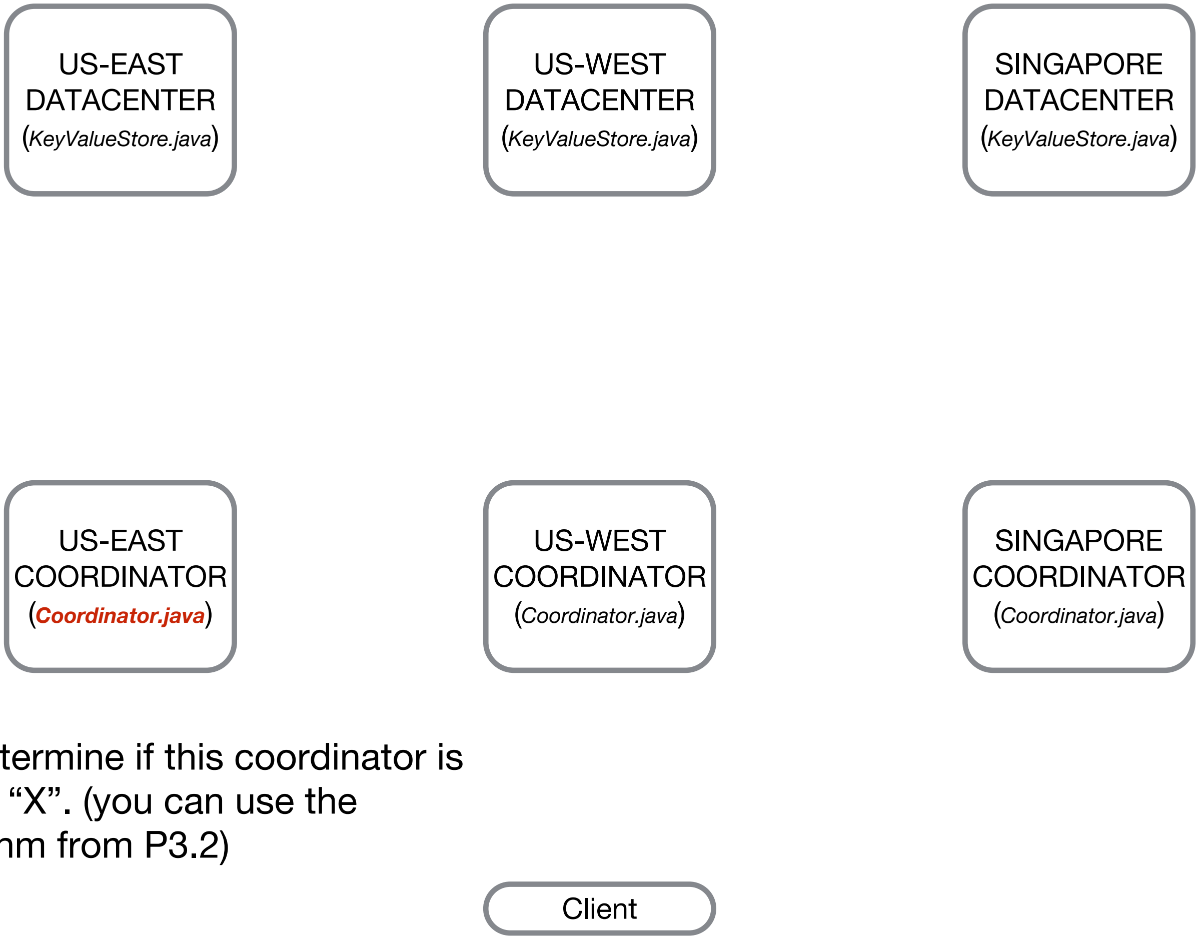
- Complete the KeyValueStore.java (on the datacenter instance) and Coordinator.java (on the coordinator instance).
- Support 2 consistencies for PUT/GET request: Strong and Eventual.


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

Example workflow for PUT request in strong consistency


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*)
US-EAST-DNS:8080/put?key=X&value=1&timestamp=1
Client

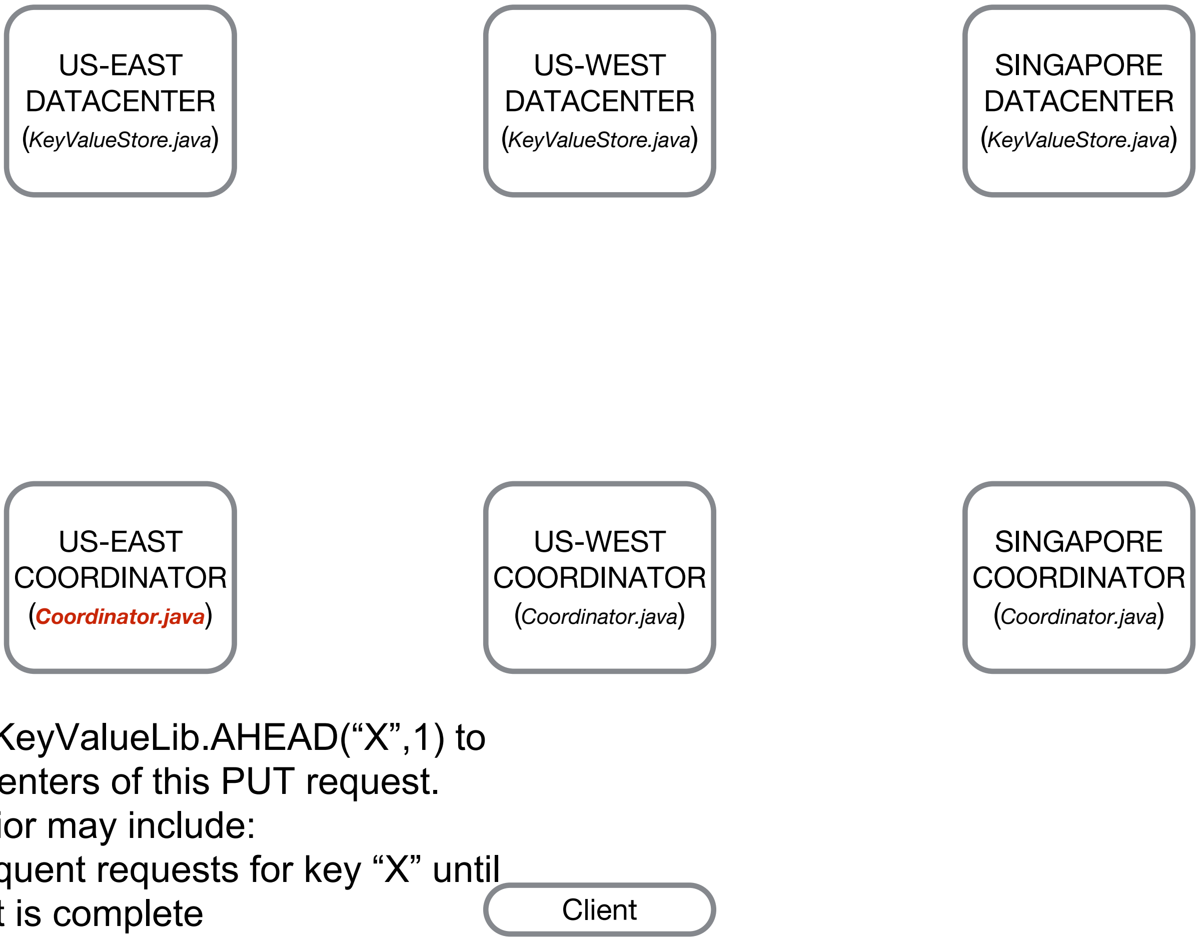
Example workflow for PUT request in strong consistency


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

hash("X") to determine if this coordinator is responsible for "X". (you can use the hashing algorithm from P3.2)

Example workflow for a PUT request in strong consistency

- If US-EAST is responsible for key "X"



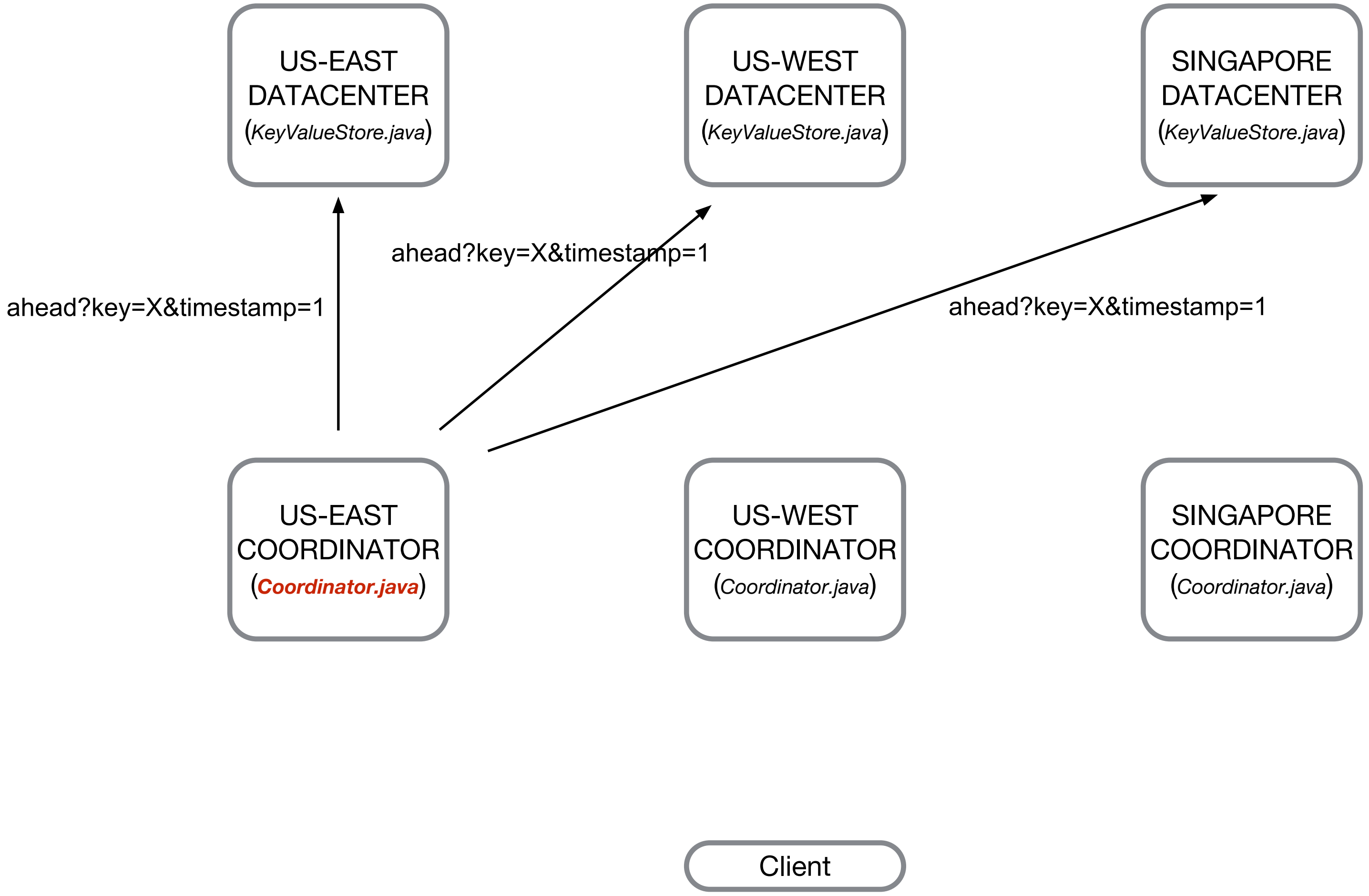
US-EAST DATACENTER (KeyValueStore.java)
US-WEST DATACENTER (KeyValueStore.java)
SINGAPORE DATACENTER (KeyValueStore.java)
US-EAST COORDINATOR (Coordinator.java)
US-WEST COORDINATOR (Coordinator.java)
SINGAPORE COORDINATOR (Coordinator.java)
Client

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

Example workflow for PUT request in strong consistency

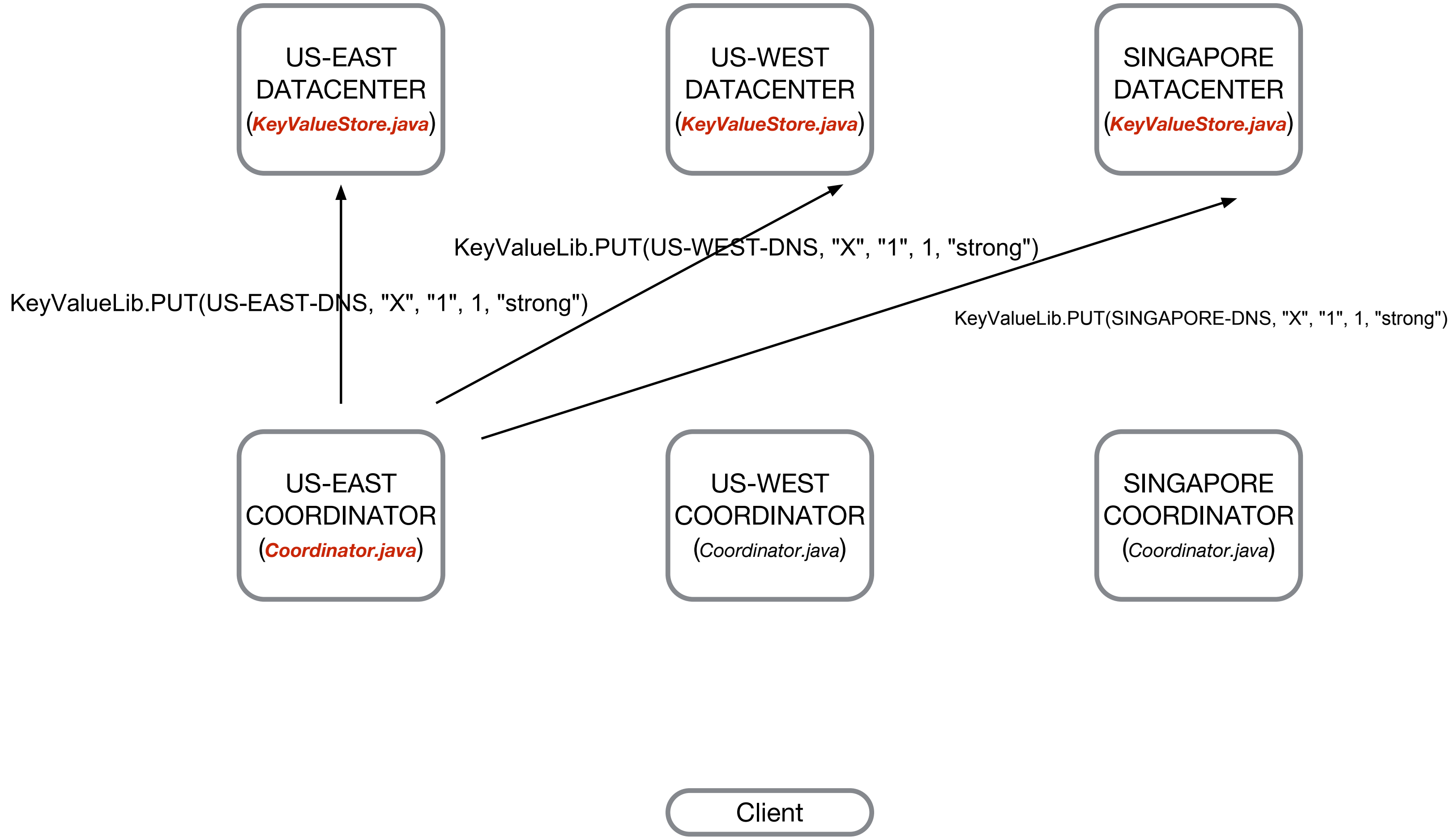
- If US-EAST is responsible for key "X"


US-EAST DATACENTER (KeyValueStore.java)
US-WEST DATACENTER (KeyValueStore.java)
SINGAPORE DATACENTER (KeyValueStore.java)
ahead?key=X&timestamp=1
ahead?key=X&timestamp=1
ahead?key=X&timestamp=1
US-EAST COORDINATOR (Coordinator.java)
US-WEST COORDINATOR (Coordinator.java)
SINGAPORE COORDINATOR (Coordinator.java)
Client

Example workflow for PUT request in strong consistency

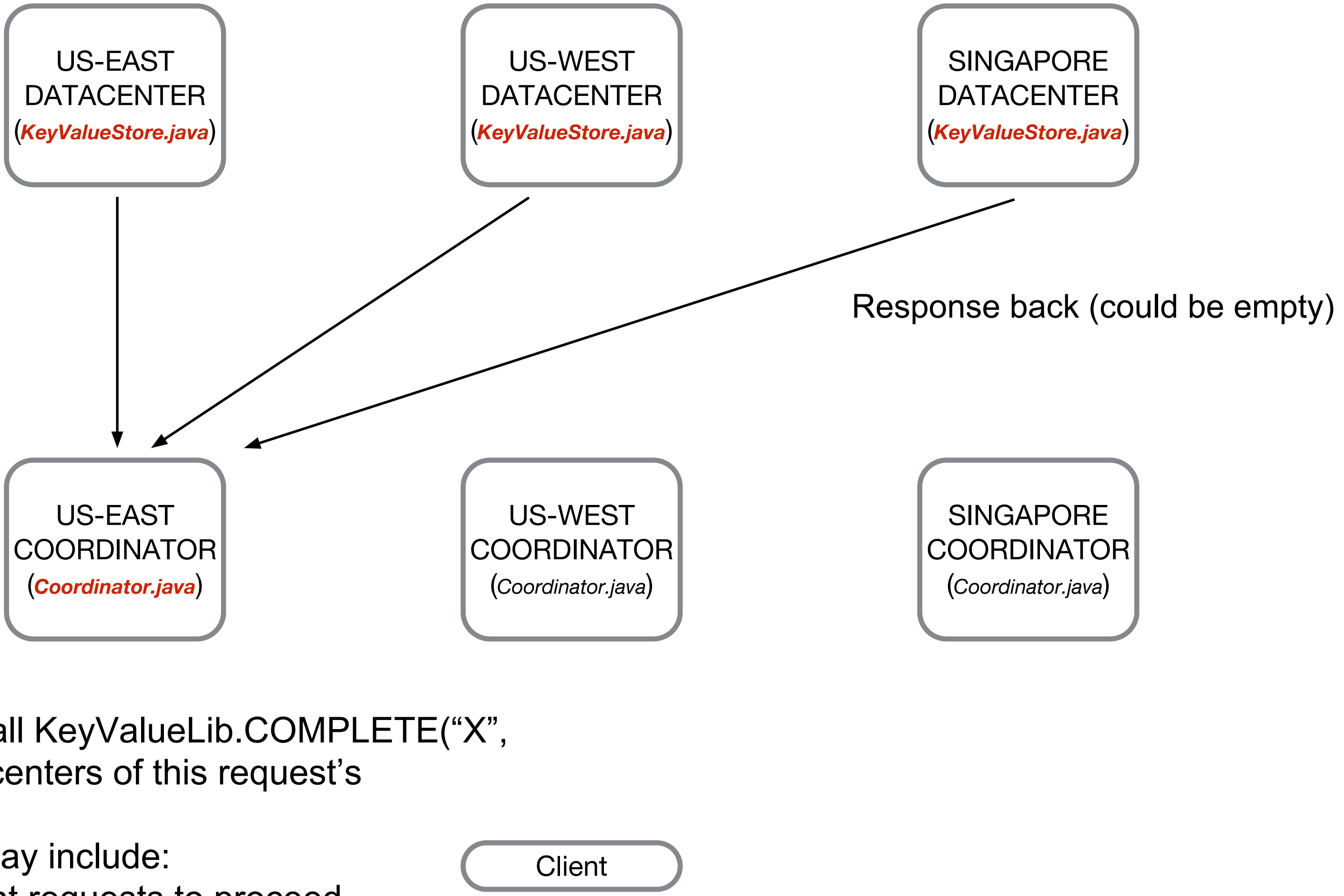
- If US-EAST is responsible for key “X”

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


US-EAST DATACENTER (KeyValueStore.java)
US-WEST DATACENTER (KeyValueStore.java)
SINGAPORE DATACENTER (KeyValueStore.java)
KeyValueLib.PUT(US-WEST-DNS, "X", "1", 1, "strong")
KeyValueLib.PUT(US-EAST-DNS, "X", "1", 1, "strong")
KeyValueLib.PUT(SINGAPORE-DNS, "X", "1", 1, "strong")
US-EAST COORDINATOR (Coordinator.java)
US-WEST COORDINATOR (Coordinator.java)
SINGAPORE COORDINATOR (Coordinator.java)
Client

Example workflow for PUT request in strong consistency

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

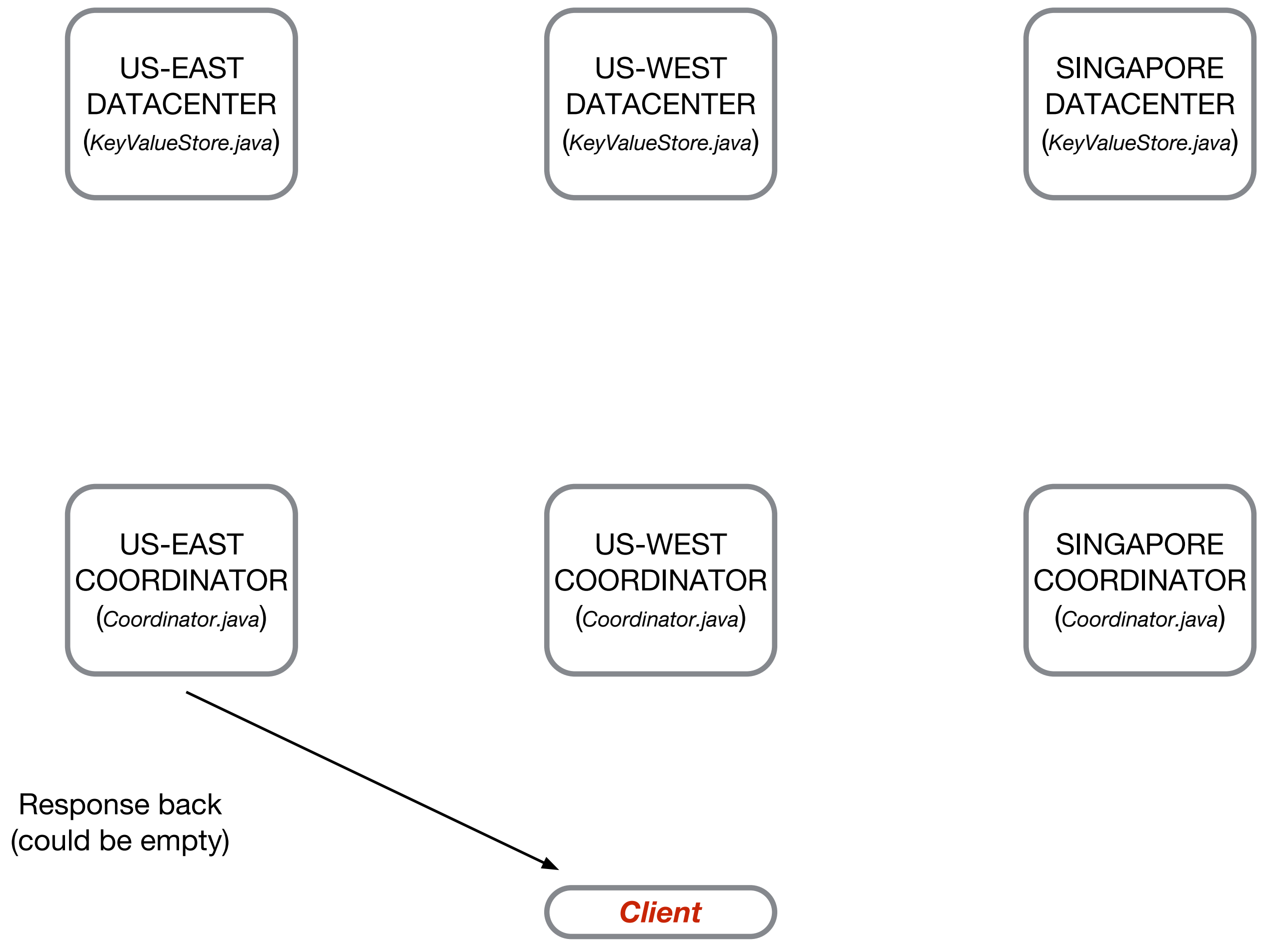

US-EAST DATACENTER (KeyValueStore.java)
US-WEST DATACENTER (KeyValueStore.java)
SINGAPORE DATACENTER (KeyValueStore.java)
Response back (could be empty)
US-EAST COORDINATOR (Coordinator.java)
US-WEST COORDINATOR (Coordinator.java)
SINGAPORE COORDINATOR (Coordinator.java)
Client

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!)

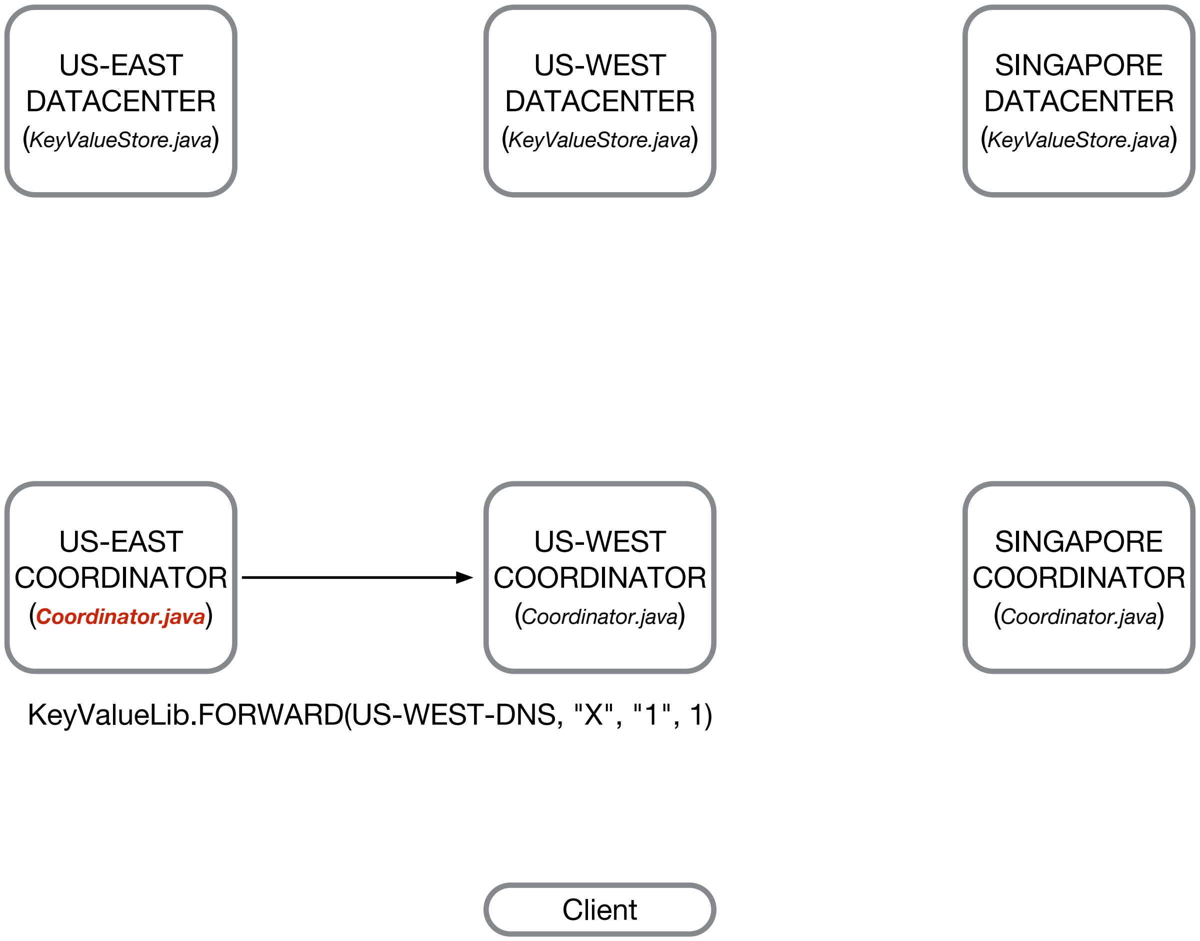
Example workflow for PUT request in strong consistency

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


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*)
Response back (could be empty)
Client

Example workflow for PUT request in strong consistency

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


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)
KeyValueLib.FORWARD(US-WEST-DNS, "X", "1", 1)
Client

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

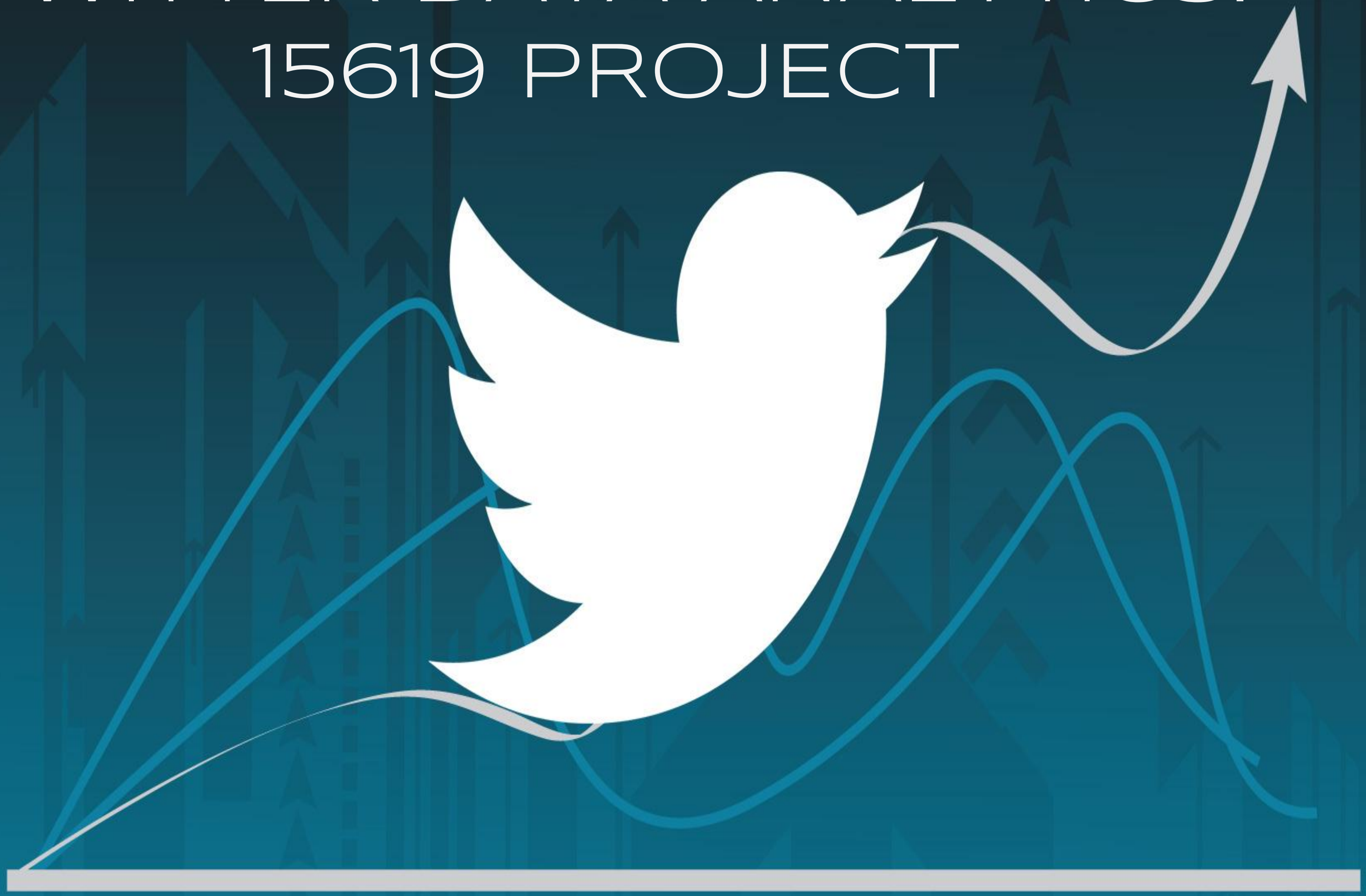
How To Test:

- 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



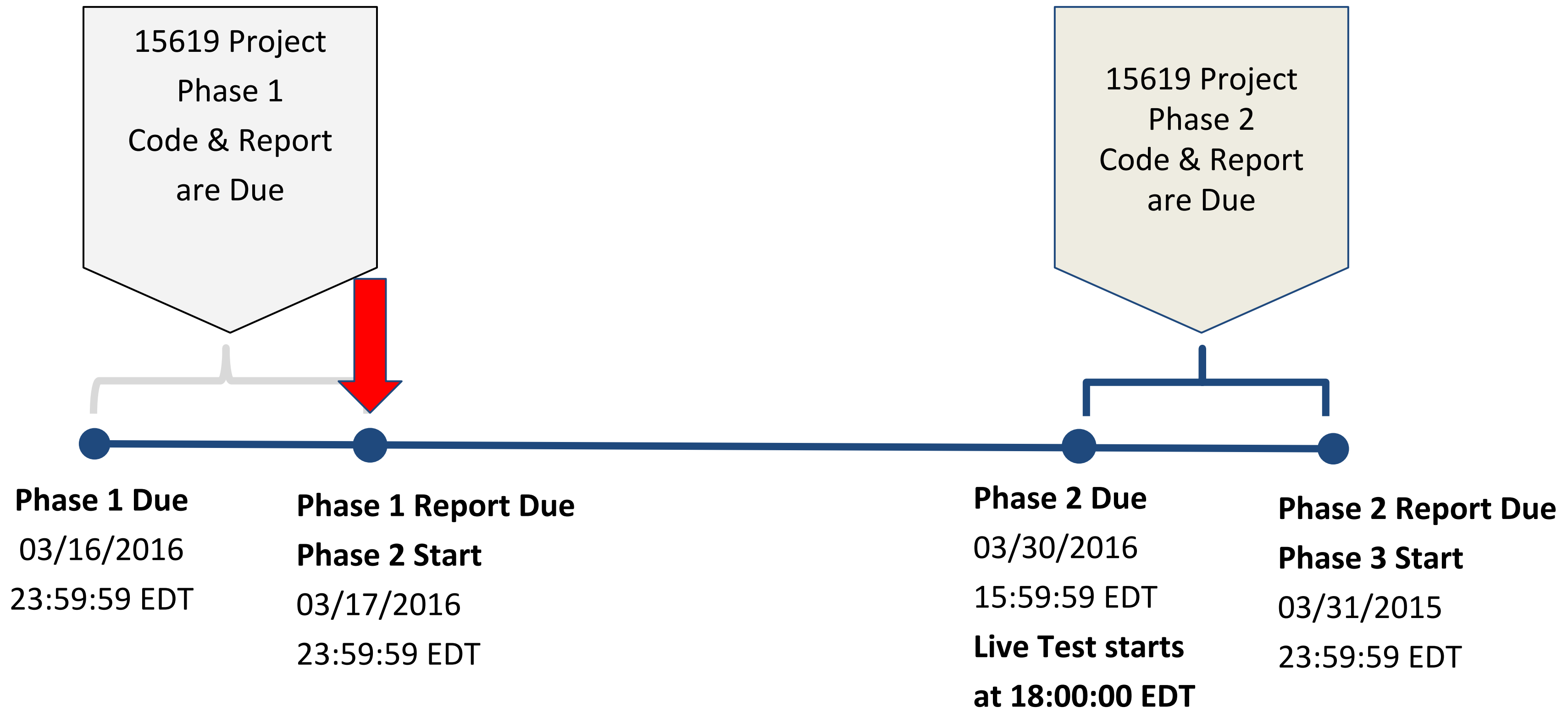
15619 Project Time Table

Phase (and query due)	**Start**	**Deadline**	**Code and Report Due**
Phase 1 Part 1 ● Q1, Q2	Thursday 02/25/2016 00:00:01 EST	Wednesday 03/16/2016 23:59:59 E**D**T	Thursday 03/17/2016 23:59:59 E**D**T
Phase 2 ● Q1, Q2, Q3	Thursday 03/17/2016 00:00:01 E**D**T	Wednesday 03/30/2016 15:59:59 E**D**T	
Phase 2 Live Test (Hbase/MySQL) ● Q1, Q2, Q3	Wednesday 03/30/2016 18:00:01 E**D**T	Wednesday 03/30/2016 23:59:59 E**D**T	Thursday 03/31/2016 23:59:59 E**D**T
Phase 3 ● Q1, Q2, Q3, Q4	Thursday 03/31/2016 00:00:01 E**D**T	Wednesday 04/13/2016 15:59:59 E**D**T	
Phase 3 Live Test ● Q1, Q2, Q3, Q4	Wednesday 04/13/2016 18:00:01 E**D**T	Wednesday 04/13/2016 23:59:59 E**D**T	Thursday 04/13/2016 23:59:59 E**D**T

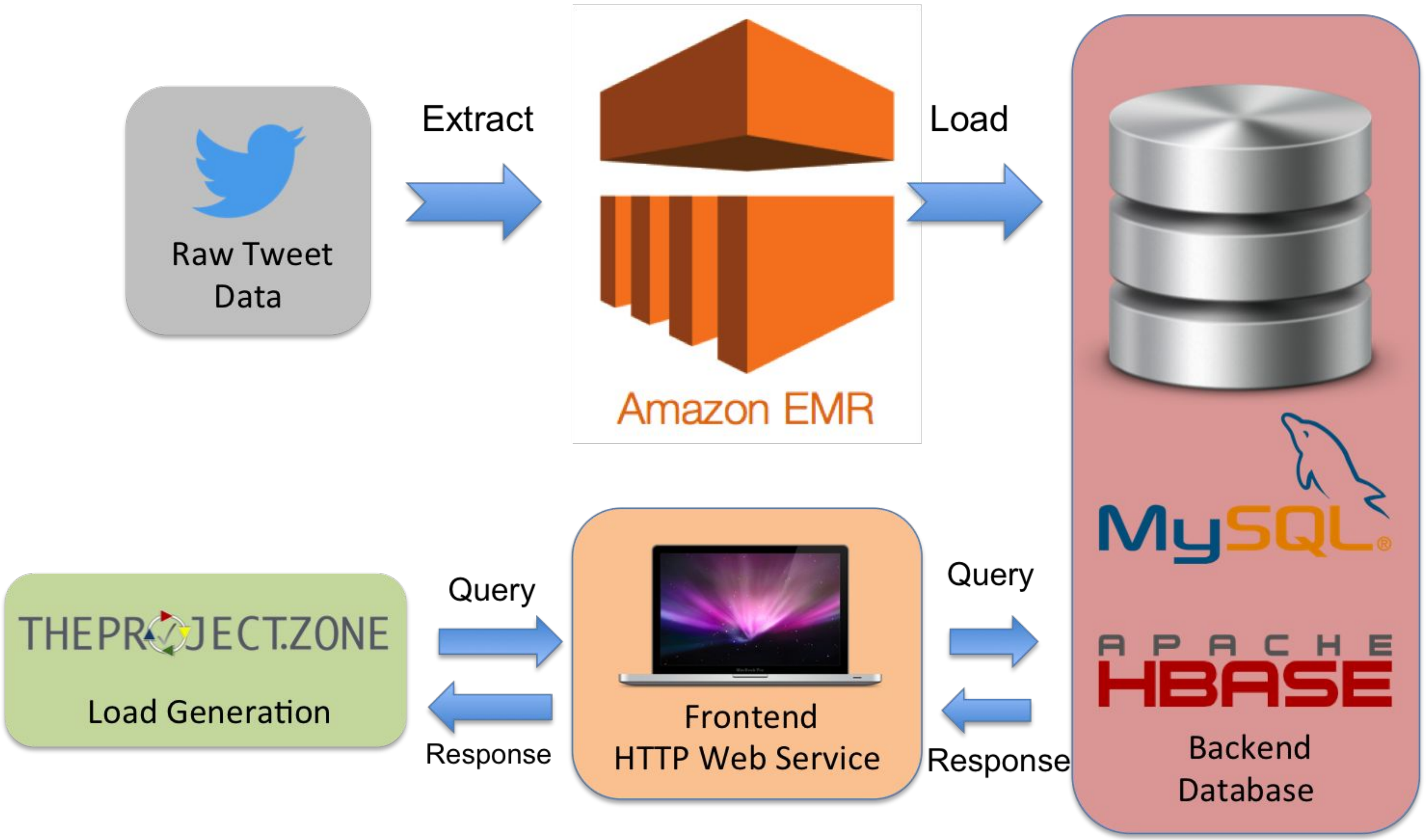
Note:
- There will be a report due at the end of each phase, where you are expected to discuss design, exploration and optimizations.

15619 Project TimeLine

- 3 weeks for Phase 1 (Q1,Q2)
- 2 weeks for Phase 2 (Q1, Q2, & Q3)


15619 Project Phase 1 Code & Report are Due
15619 Project Phase 2 Code & Report are Due
Phase 1 Due 03/16/2016 23:59:59 EDT
Phase 1 Report Due **Phase 2 Start** 03/17/2016 23:59:59 EDT
Phase 2 Due 03/30/2016 15:59:59 EDT **Live Test starts at 18:00:00 EDT**
Phase 2 Report Due **Phase 3 Start** 03/31/2015 23:59:59 EDT

15619 Project System Architecture



Phase 1 Submission Status

Q1: Up to 06:00 am, EDT 03/15/2016

75 teams get score: 100

5 teams < 90

Highest RPS: 33405 rps -- MyLittlePony

Q2: Up to 06:00 am, EDT 03/15/2016

Highest RPS: 22503.4 -- Sugoyi

28 Teams reach target RPS

51 Teams have >0 RPS

Phase 1 Common Issues

1. Same machine, same query, multiple runs, but different RPS.
   - Some web frameworks have their built-in buffer, multiple runs of same test set will generate higher and higher rps.

2. Very fast when using browser to test single requests, but it turns out to be very slow when using TPZ generated load.
   - Is your web frontend blocked? DB connection & query operations in block mode?
   - Exception captured? Make sure all requests have a response.
   - “connection: close”

3. Speed through Load Balancer very slow
   - It takes a long time to warm up the ELB. It would be very helpful if you try to build your own warm-up script.

Q2 Hints - MySQL

- System Environment
  - Storage Medium
  - Storage Engine
  - Character set
  - Import data (SHOW WARNINGS)
  - Indexing

- Profiling/Optimization
  - EXPLAIN
  - SET PROFILING=1
  - htop, iotop

Q2 Hints - HBase

- Loading data:
  - Pig, thrift

- HBase schema:
  - GET is much faster than SCAN
  - How to design rowkey?

- HBase cluster:
  - Cloudera Manager - easy deployment and management of cluster
  - Deploy your own HBase cluster and automate it
  - Using EMR will lead to higher cost ⇒ must use less instances <$.85

- HBase configuration tuning:
  - Region size／number http://archive.cloudera.com/cdh5/cdh/5/hbase-0.98.1-cdh5.1.5/book/ops.capacity.html

Q3 : Handling Complex Read Queries

- Calculate word occurrences in tweet text within a certain user id range and a data range. (Two-range query)
- Request Format

```
GET/q3?start_date=yyyy-mm-dd&end_date=yyyy-mm-dd&start_userid=uid&end_userid=uid&words=w1,w2,w3
```

- Response Format

```
TEAMID,TEAM_AWS_ACCOUNT_ID\n
w1:count1\n
w2:count2\n
W3:count3\n
```

- Target RPS 6000

Q3 : Handling Complex Read Queries

- Request Example (Double Range Query)

```
GET/q3?start_date=2014-04-01&end_date=2014-05-28&start_userid=51538630&end_userid=51539182&words=u,petition,loving
```

- Response Format

```
Team,1234-5678-1234
u:7\n
petition:2\n
loving:5\n
```

Q3: ETL

1. Split words when a non-alphanumeric character ([^a-zA-Z0-9]) is encountered.

2. Words are case INSENSITIVE in word count.

3. Banned words in Q2 will not appear in Q3 requests.

4. Ignore words from stop words list.

We will give reference file and reference server just as Q2.

Q3 Hints

- ETL:
  - Simpler than Q2, only using English tweets
  - Make sure to remove duplicate tweets

- Try to design a good schema, especially for HBase

- HBase:
  - Get is faster than Scan
  - If using Scan, do not scan the whole table

- Try Cloudera or deploy your own HBase cluster

- Is the load balanced in your cluster? Any hot nodes?

Phase 2 Live Test

HBase LiveTest

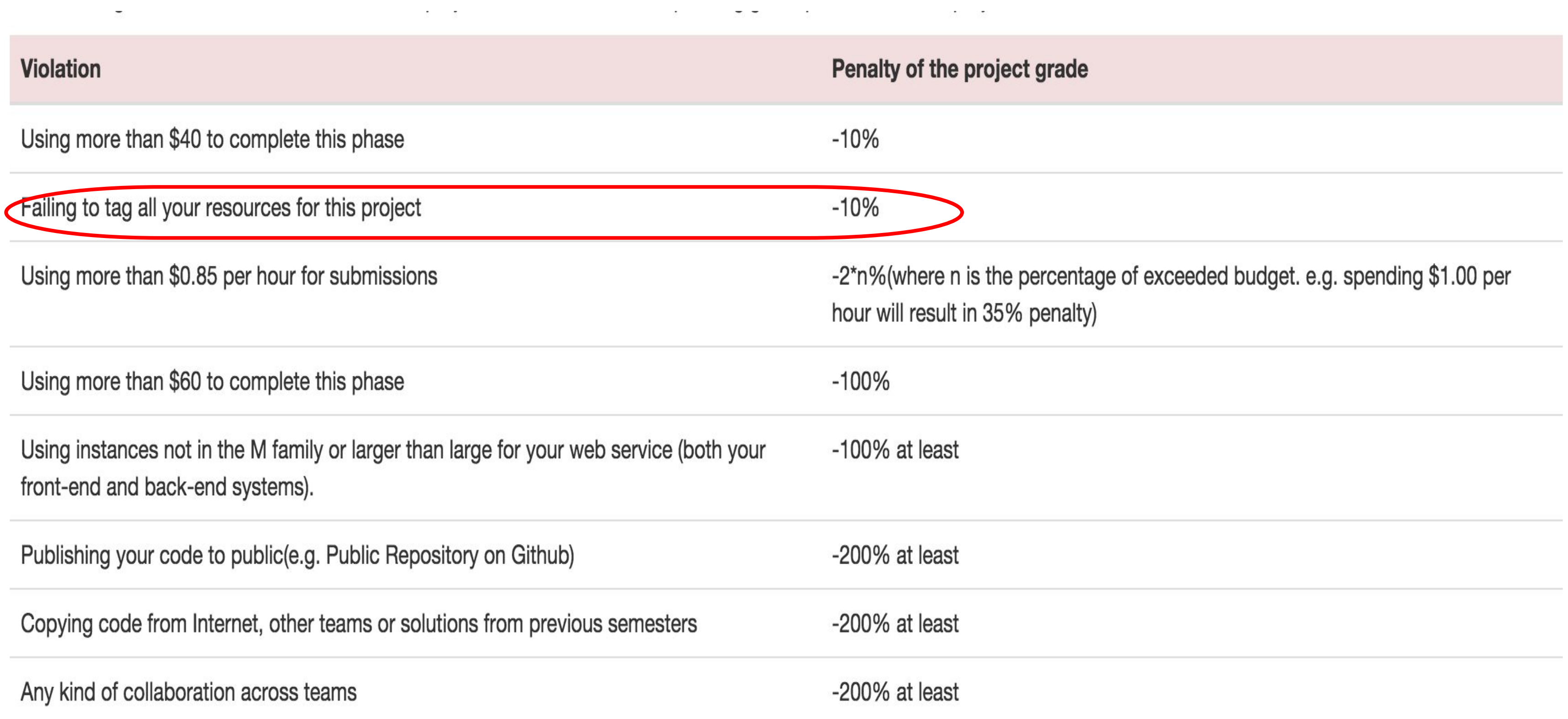
Time	Value	Target	Weight
6:00 pm - 6:30 pm	Warm-up (Q1 only)	-	0%
6:30 pm - 7:00 pm	Q1	27000	5%
7:00 pm - 7:30 pm	Q2	10000	10%
7:30 pm - 8:00 pm	Q3	6000	10%
8:00 pm - 8:30 pm	Mixed Reads(Q1,Q2,Q3)	TBD	5+5+5 = 15%

Half Hour Break

MySQL LiveTest

Time	Value	Target	Weight
9:00 pm : 9:30 pm	Warm-up	-	0%
9:30 pm - 10:00 pm	Q1	27000	5%
10:00 pm - 10:30 pm	Q2	10000	10%
10:30 pm - 11:00 pm	Q3	6000	10%
11:00 pm - 11:30 pm	Mixed Reads (Q1,Q2,Q3)	TBD/TBD/TBD	5+5+5 = 15%

Important Phase 1 Reminders



Violation	Penalty of the project grade
Using more than $40 to complete this phase	-10%
Failing to tag all your resources for this project	-10%
Using more than $0.85 per hour for submissions	-2*n%(where n is the percentage of exceeded budget. e.g. spending $1.00 per hour will result in 35% penalty)
Using more than $60 to complete this phase	-100%
Using instances not in the M family or larger than large for your web service (both your front-end and back-end systems).	-100% at least
Publishing your code to public(e.g. Public Repository on Github)	-200% at least
Copying code from Internet, other teams or solutions from previous semesters	-200% at least
Any kind of collaboration across teams	-200% at least

Phase 1 Report due soon

- Phase 1 <u>Report & Code</u> Deadline
    - **[11.59 PM EDT 03/17/2016]**
    - Upload to TheProject.Zone
    - No code ⇒ ZERO POINTS FOR ENTIRE PHASE 1
    - Missing files ⇒ ZERO POINTS FOR ENTIRE PHASE 1

- Very High Standard Expected in Report (25%)
    - Make sure you highlight failures and learning
    - If you didn’t do well, explain why
    - If you did, explain how
    - <u>Compare at least two web frameworks</u>

Upcoming Deadlines

- Quiz 8: Unit 4 - Module 15 - Case Studies: DFSs

  Due: 3/18/2016 11:59PM Pittsburgh

- Project 3.3: Consistency in Distributed K-V Stores

  Due: 3/20/2016 11:59PM Pittsburgh

- 15619Project: Phase 1

  Due: 03/16/2016 11:59PM Pittsburgh

- 15619Project: Phase 1, Report

  Due: 03/17/2016 11:59PM Pittsburgh