15-319 / 15-619 Cloud Computing

Recitation 11

March 31st and April 2nd, 2015

Overview

- **Administrative issues**
  - Tagging, 15619Project
- **Last week's reflection**
  - Project 3.4
  - Quiz 4
- **This week's schedule**
  - Project 3.5
  - Unit 5 - Modules 16
- **Demo**
- **Twitter Analytics: The 15619Project**

Caution!

- Tag spot instances in the FIRST 59 mins.
  - Otherwise, it will be considered as an untagged instance for that hour.

- 15619Project is in progress!
  - Phase 2 done - Wednesday, April 1st
  - Phase 3 queries out - Due in 2 weeks (April 15th)
  - Tag all resources used for 15619Project as
    - Key: 15619project, Value: phase3
    - Key: 15619backend, Value: hbase/mysql

Project 3.4 : FAQs - 1

Problem 1: MySQL query takes way too long to run for the first time.

- This is an unexpected scenario. We're not sure what exactly the reason is.
  - AWS varying performance?
  - Spot instances vs on demand instances?

Project 3.4 : FAQs - 2

Problem 2: Does not receive any mark on submission but my answer is correct.

- The runtime of the queries was not in the expected range.
- For optimization queries, not optimized enough.

This week: Project 3.5

- P3.1 Files vs Databases
- P3.2 Partitioning and Replication
- P3.3 Database-as-a-Service
- P3.4 Cloud Data Warehousing
- P3.5 Consistency in Distributed Key-Value Stores

Consistency in Distributed Systems

- Globally distributed services
- Datastores are across multiple geographical locations
  - Replicated datastores enable low latency access
- Need to ensure data consistency across the replicas
- Consistency models
  - Strong Consistency
  - Causal Consistency
  - Eventual Consistency

Different Levels of Consistency

- Strong Consistency
  - All replicas - All operations are done atomically and in order in all replicas
  - All clients will see the same data at any point
  - High overhead and slow
- Causal Consistency
  - All replicas - “causally” related operations in order
  - Better performance compared to strong
  - Still overhead for causally related operations
- Eventual Consistency
  - All replicas - If there are no more updates, eventually all replicas will get the same value.
  - Very low overhead and fast
  - Clients can see stale values

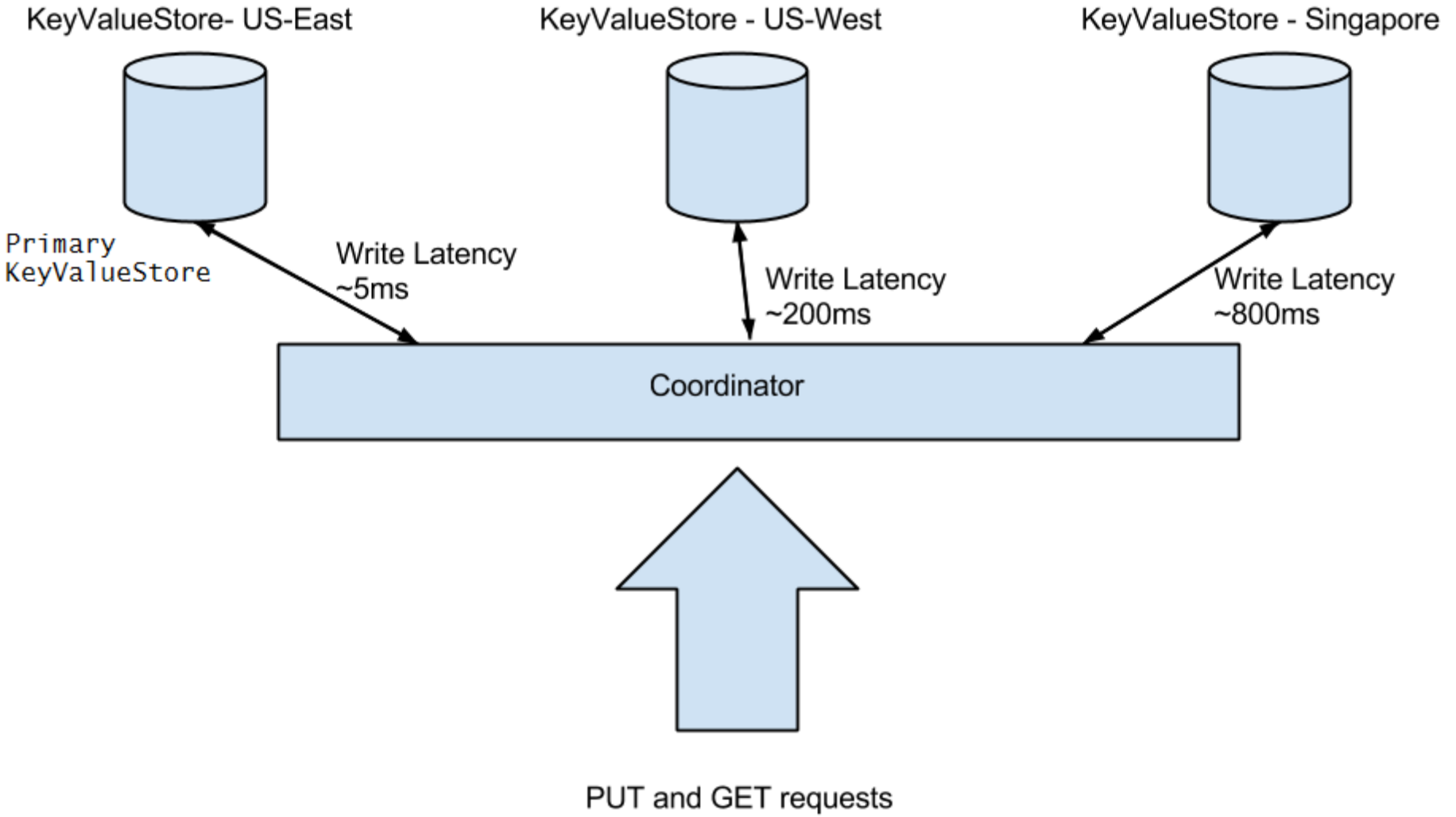
P3.5: Background

- Carnegie Records(CR) is planning to build its own online store, which requires varying levels of consistency modes in the backend stores.
- CR wants you to build a distributed key-value store supporting different levels of consistency.
- CR asked you to implement the distributed key-value store which supports:
  - Strong consistency
  - Causal consistency

P3.5 Distributed Key Value Store

- Replicated globally
  - Different replicas can have different access times.
- Stores (key, value) pairs
- Consistency among the replicas is required
  - Different models of consistency depending on applications and performance requirements
- More during the demo session

P3.5: Overview



P3.5: Tasks

- Start three datacenter instances
- Start a coordinator instance
- Your task is to program the coordinator instance to perform operations on the datacenter instances
- We provide two basic API methods to program
  - PUT(Datacenter, Key, Value)
  - GET(Datacenter, Key)

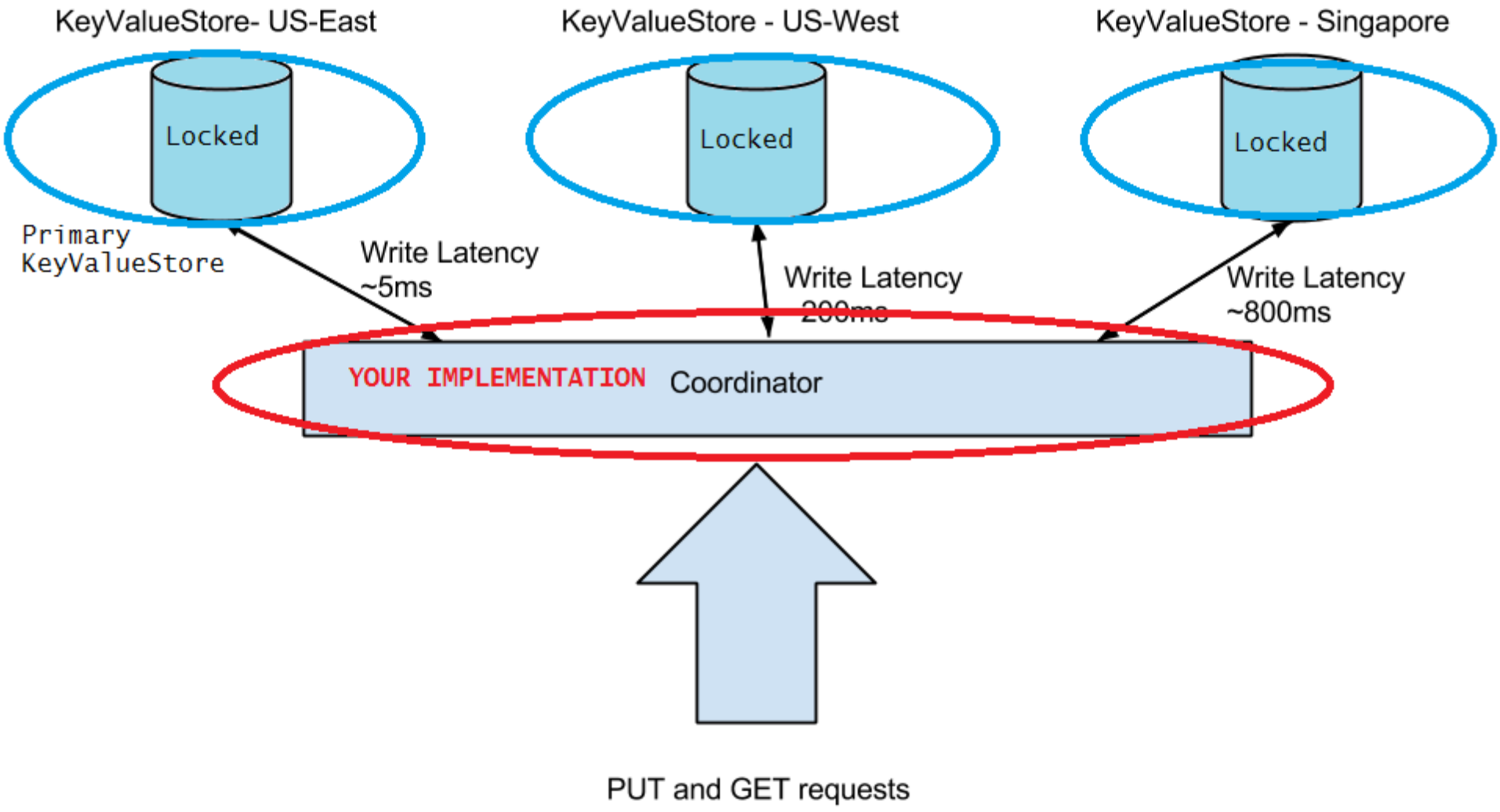
Part 1: Strong Consistency

- Requirements
  - Every PUT request per key should be atomic across all replicas.
  - At any point in time, only one PUT operation per key can be performed on the datacenter instances.
  - A GET operation per key has to be blocked if a PUT operation for the same key is currently being processed.
  - The locking of the datacenters has to be done at the key level, i.e. operations on multiple keys can run in parallel.

Part 2: Causal Consistency

- Requirements
  - All PUT operations per key must be seen in all the datacenters in the same order.
  - At any point in time, different operations can be performed in different datacenter instances in parallel.
    - Lock only one datacenter at a time for the operation for each key.
  - A GET operation should return the value immediately without being blocked even if it is stale.

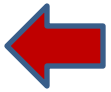
P3.5: Tasks


KeyValueStore- US-East
KeyValueStore - US-West
KeyValueStore - Singapore
Locked
Locked
Locked
Primary KeyValueStore
Write Latency ~5ms
Write Latency
Write Latency ~800ms
YOUR IMPLEMENTATION
Coordinator
PUT and GET requests

P3.5: Possible hurdles

- Synchronization between multiple threads in the Coordinator
  - Race conditions
- Incorrect order of updates on the datacenter instances
  - Race conditions
- Debugging
  - Consistency checker

Module to Read

- UNIT 5: Distributed Programming and Analytics Engines for the Cloud
  - Module 16: Intro to distributed programming for the Cloud 
  - Module 17: Distributed analytics engines: MapReduce
  - Module 18: Distributed analytics engines: Spark
  - Module 19: Distributed analytics engines: GraphLab
  - Quiz 5: Distributed Programming and Analytics Engines for the Cloud

Distributed Programming

- **Taxonomy of Programs:**
  - Sequential
  - Concurrent
  - Parallel

- **Challenges in programming the cloud:**
  - Scalability
  - Communication overhead
  - Heterogeneity
  - Synchronization
  - Fault Tolerance
  - Scheduling

Upcoming Deadlines

- Modules 16
  - Due on April 5th
- P3.5 Consistency in Distributed Key-Value Stores
  - Due: 11:59PM ET April 5th (Sunday)
- 15619Project Phase3
  - Deadline: 16:59PM ET Apr 15th (Wednesday)
    - Live Test, due 16:59PM ET Apr 15th

Demo/Discussion

Consistency Models for P3.5

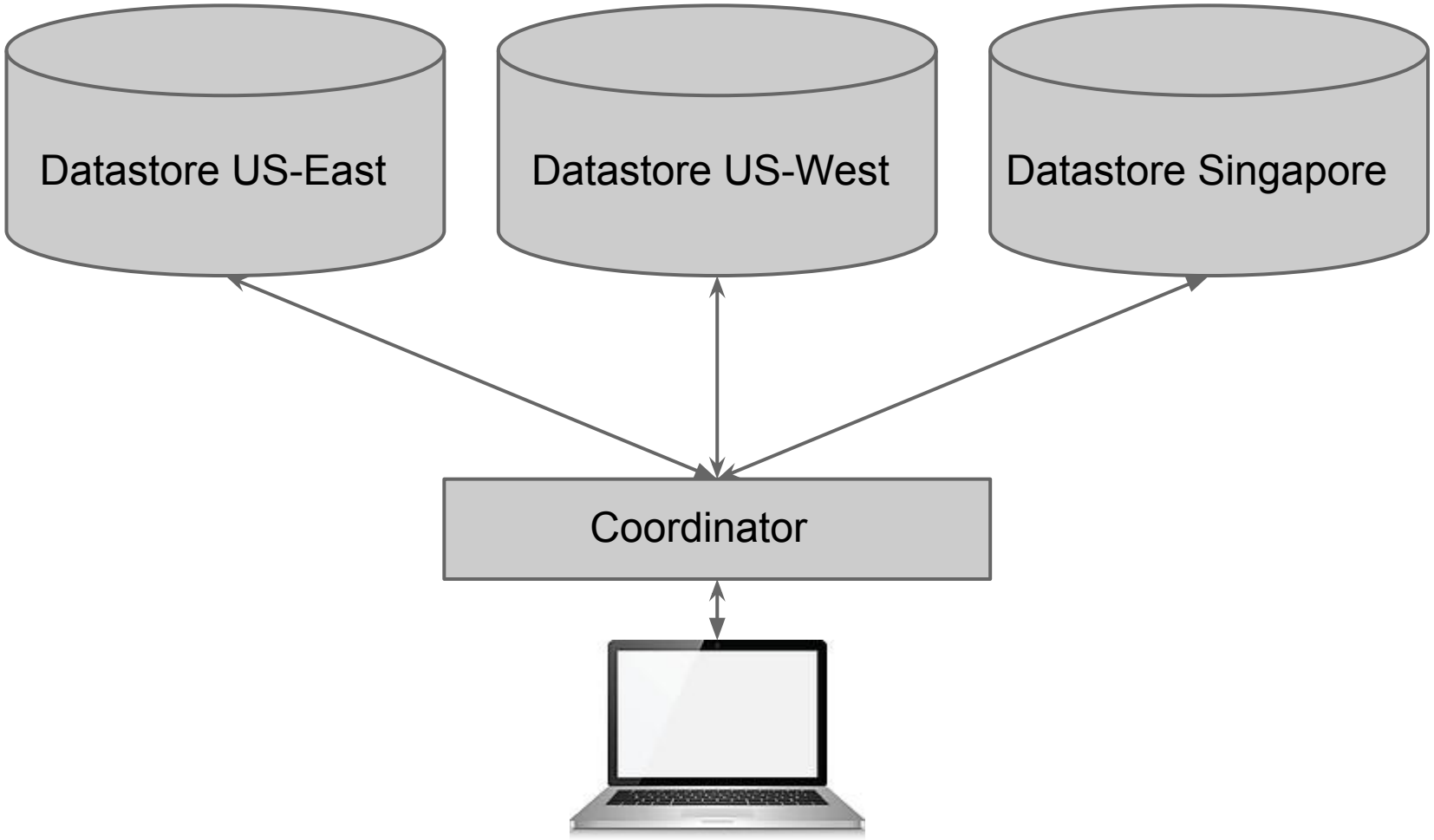
Recall P3.2: Replication v/s Sharding

- **Replication**
  - Fault-tolerance if servers fail
  - Higher throughput for read requests since clients can read from multiple servers / load balancing
  - Can also replicate in other geographical locations
- **Sharding**
  - SPOF if an instance, which had not been replicated, fails
  - Higher throughput for write requests since some of them can be done in parallel

Replication and write/update queries

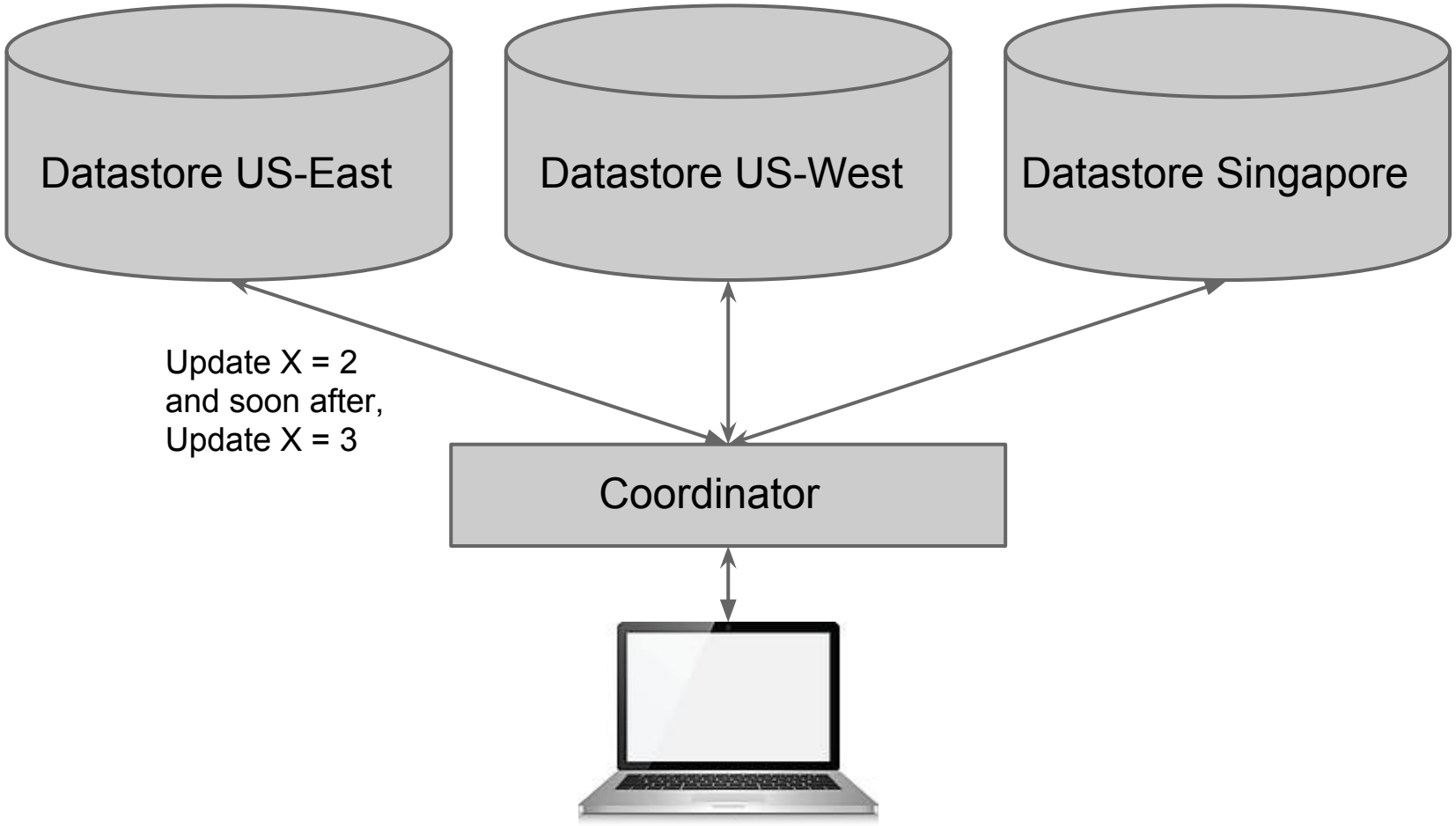
- **Case**
  - Database is replicated across three back-end instances
  - One front-end server, three back-end servers
  - The front-end server receives an update query
  - How will you handle it?

P3.5 Setup


Datastore US-East
Datastore US-West
Datastore Singapore
Coordinator

Creating an inconsistency

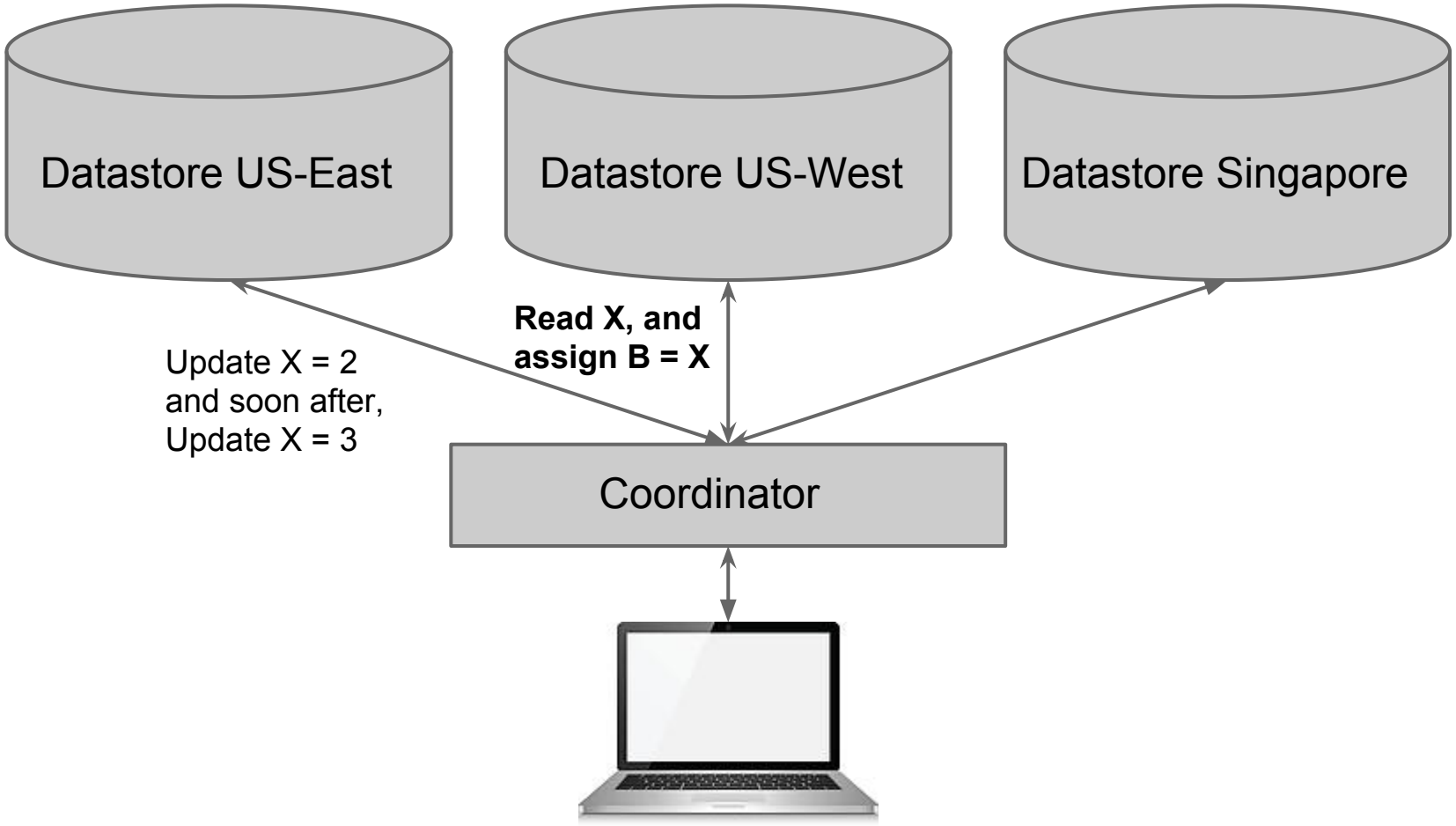
Consider our key-value store. Let a particular record be denoted by X. Its current value is 1 in all datacenters. Now, the following operations occur:



Client sends a query to update X =2, and soon after, sends another query to update X = 3

Creating an inconsistency (contd..)

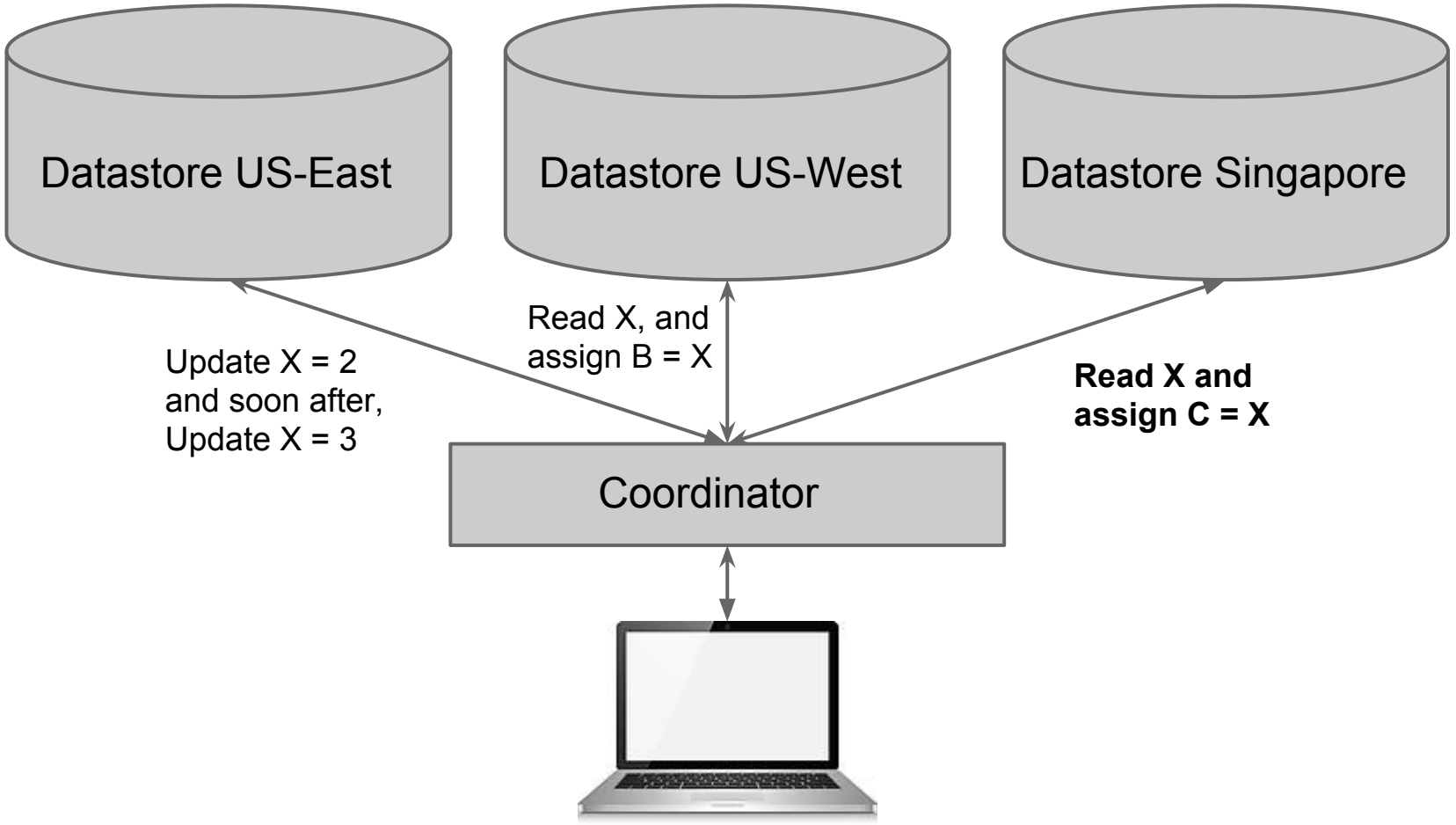
Consider our key-value store. Let a particular record be denoted by X. Its current value is 1 in all datacenters. Now, the following operations occur:



The client reads X and puts it into a variable, B

Creating an inconsistency (contd..)

Consider our key-value store. Let a particular record be denoted by X. Its current value is 1 in all datacenters. Now, the following operations occur:



Assuming that the clients reads X **after** the update X=3 has been completed on Datastore US-East, What are the values of B and C?

Creating an inconsistency (contd..)

What are the values of B and C?

- **It depends!**
  - ...on the consistency model
  - The following answers are possible:

B	**C**
1	1
1	2
1	3
2	1
...	
3	3

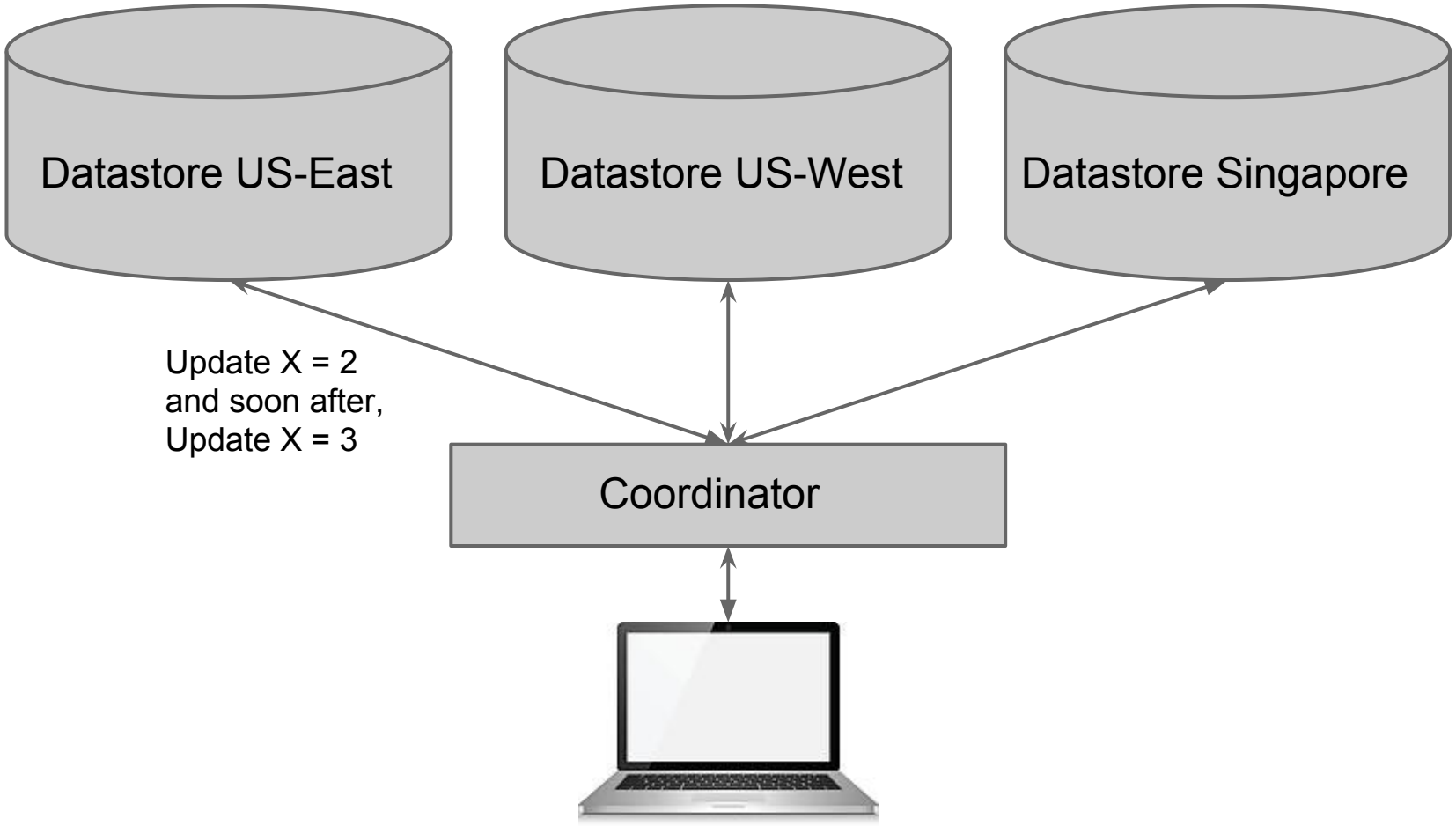
Eventual Consistency

If no new updates are made, all reads will eventually return the last updated value.

- The replica updates can happen after a long time, but it is ensured that the datacenters will be consistent at some point of time provided no more writes have been performed on the same object.

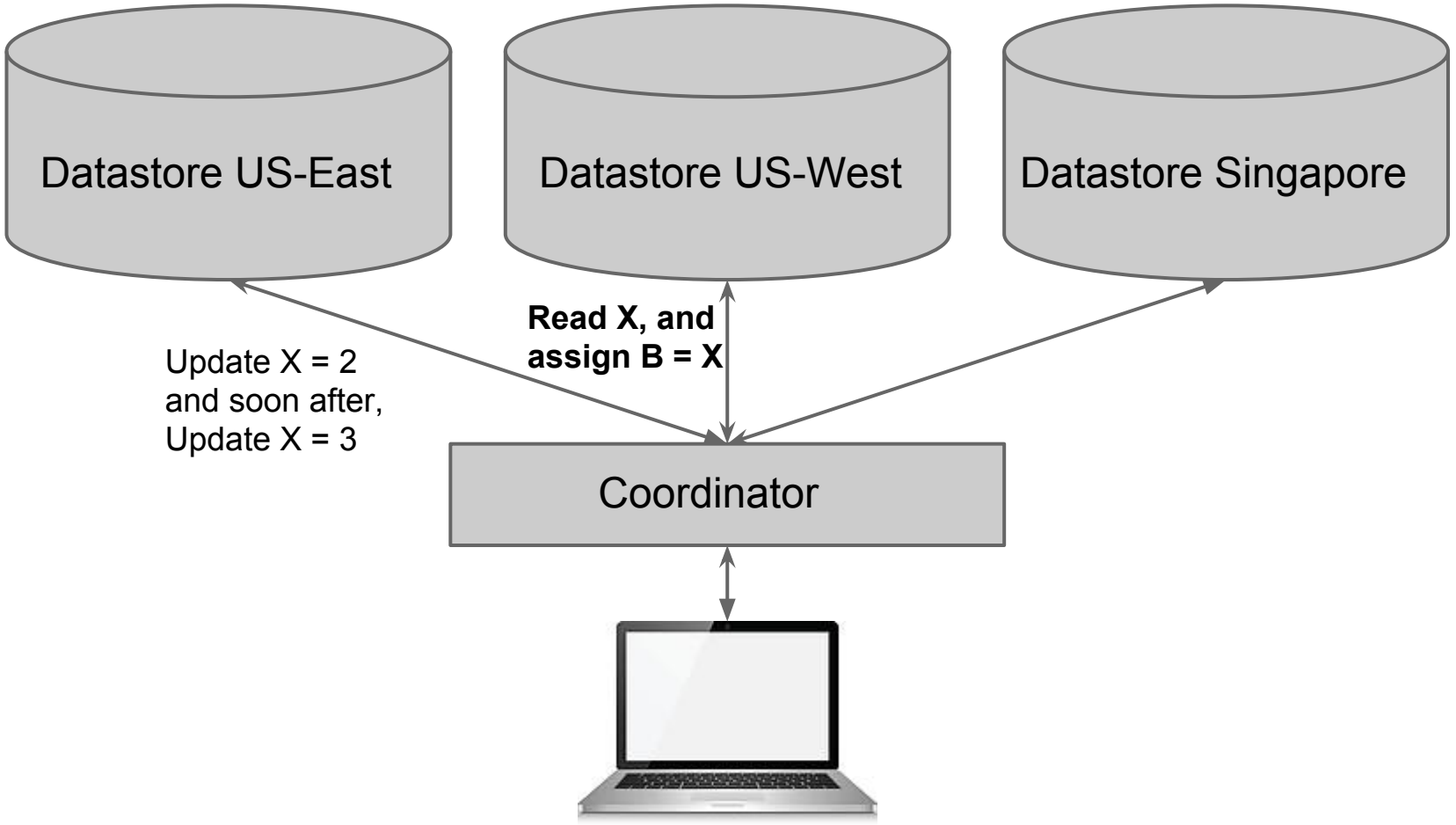
Eventual Consistency (Question)

Consider our key-value store. Let a particular record be denoted by X. Its current value is 1 in all datacenters. Now, the following operations occur:



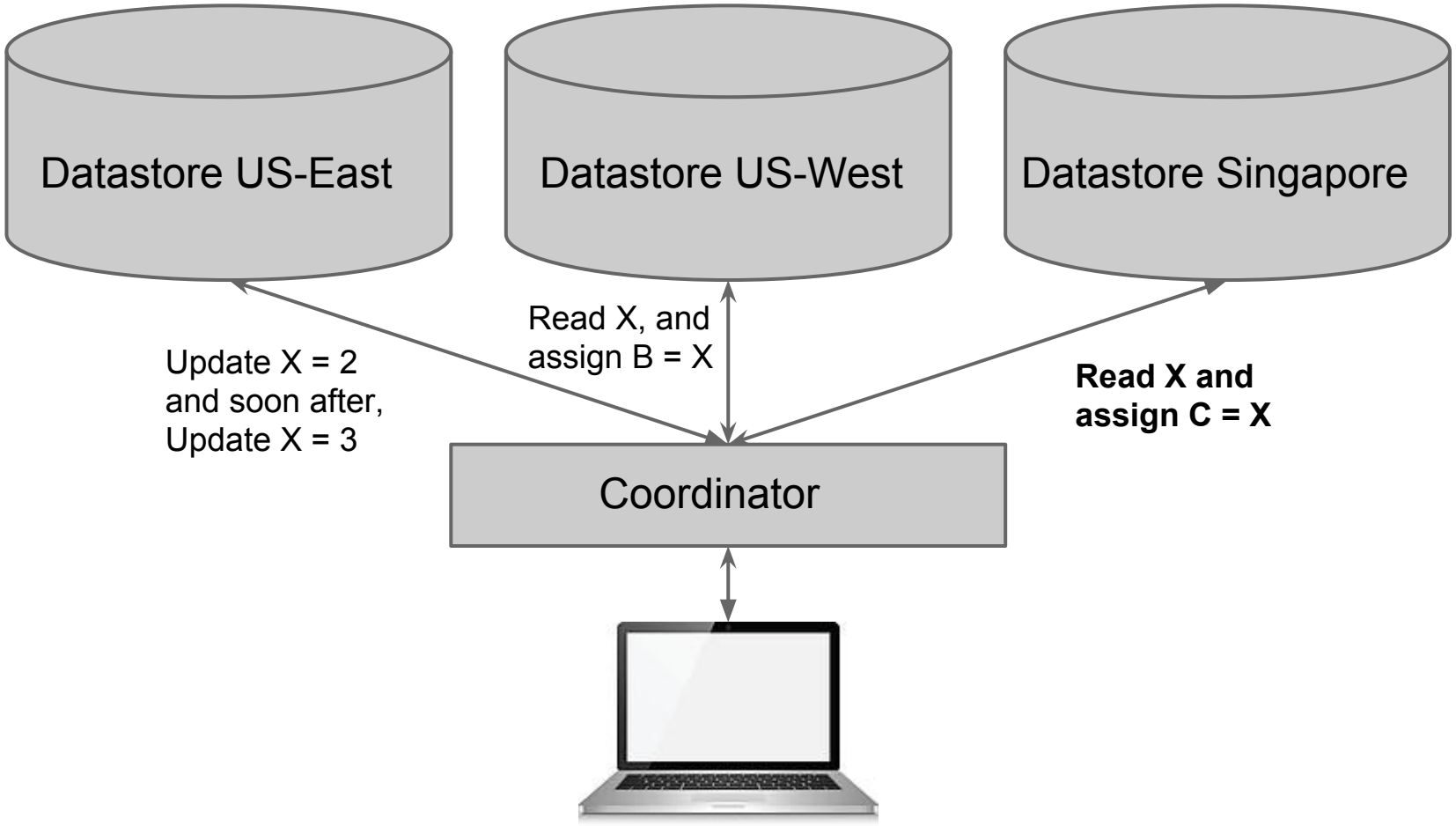
Client sends a request to update X =2, and then again sends a request to update X = 3

Eventual Consistency (Question contd..)



Client reads X, and assigns it to a variable, B

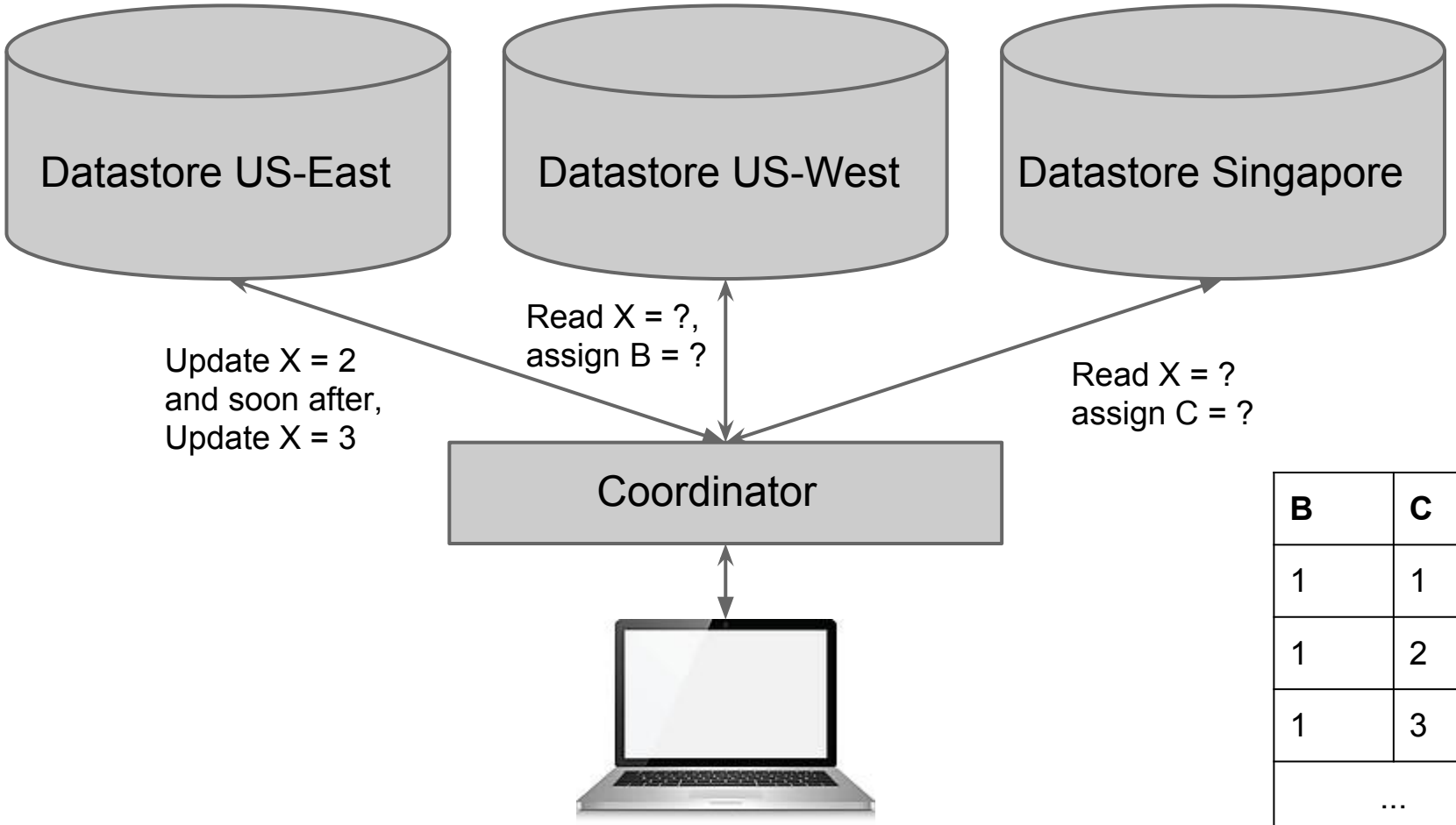
Eventual Consistency (Question contd..)



Client reads X, and assigns it to a variable, C

Eventual Consistency (Answer)

Consider our key-value store. Let a particular record be denoted by X. Its current value is 1 in all datacenters. Now, the following operations occur:



B	C
1	1
1	2
1	3
...	
3	3

B and C can be 1, 2 or 3. It depends on which update has been propagated to the two datacenters at the time of the read.

Strong Consistency

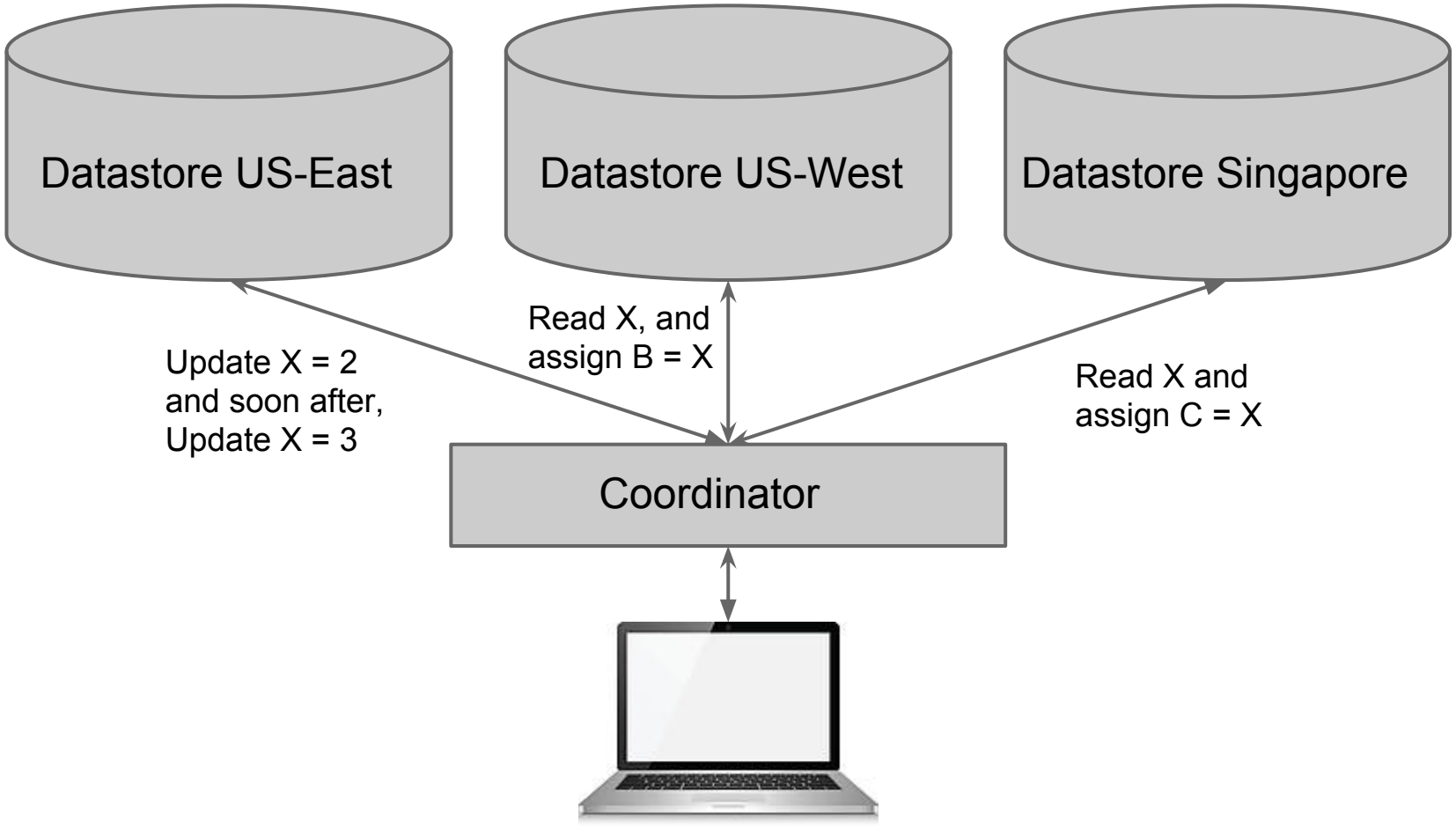
All operations were executed in some sequential order and all replicas see the operations in the exact same order.

In the case of P3.5:

- All datacenters see the operations (PUT/GET) in same order
  - The order in which they came
- Any PUT occurring at a datacenter is instantaneously visible across all datacenters.
  - Reads for the object being written must be locked until the update has been made across all datacenters.

Strong Consistency (Question)

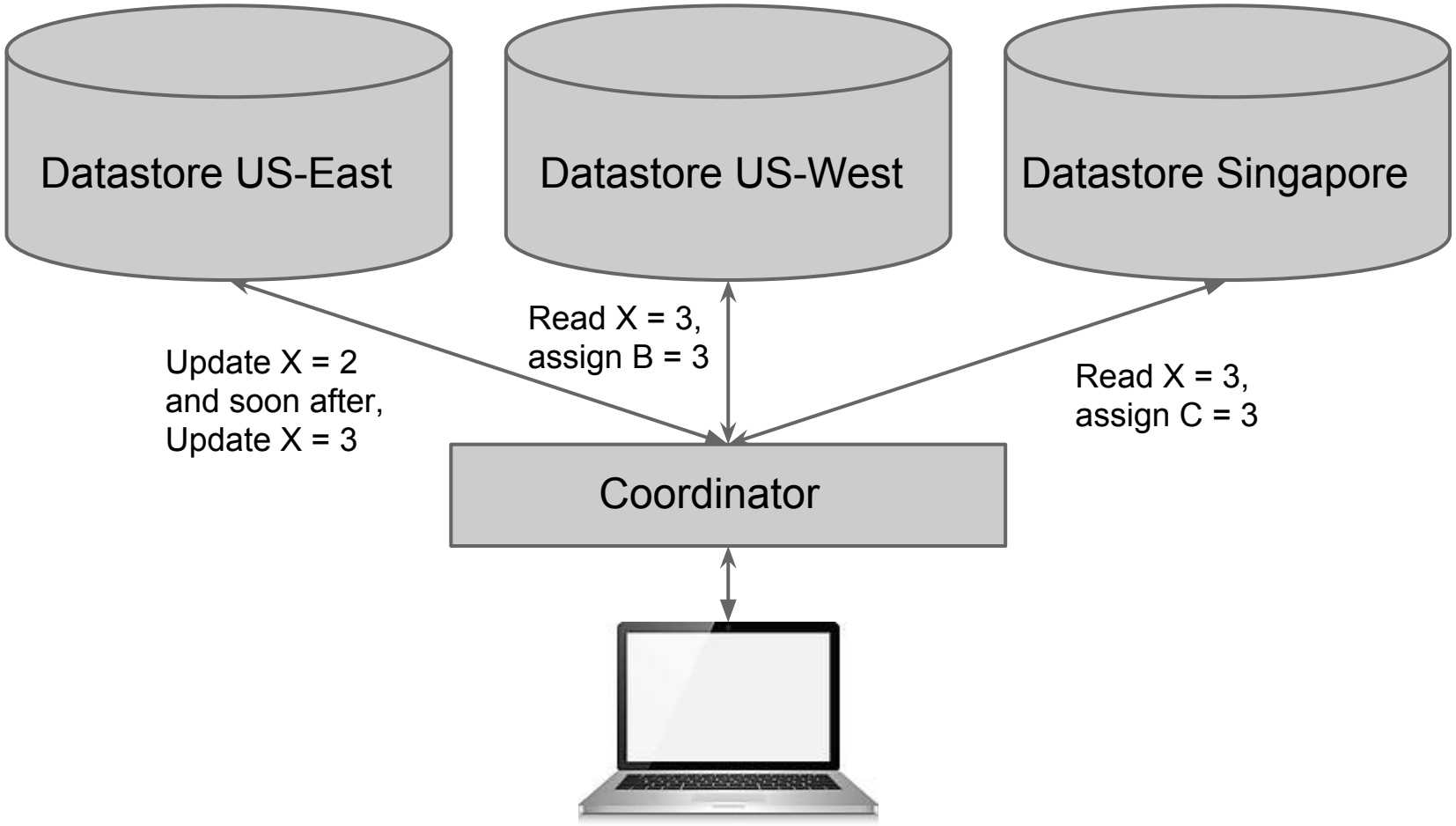
Consider our key-value store. Let a particular record be denoted by X. Its current value is 1 in all datacenters. Now, the following operations occur:



Assuming that the two clients read X **after** the update X=3 has been completed on Datastore US-East, what are the values of B and C?

Strong Consistency (Answer)

Consider our key-value store. Let a particular record be denoted by X. Its current value is 1 in all datacenters. Now, the following operations occur:



Assuming that the two clients read X **after** the update X=3 has been completed on Datastore US-East, what are the values of B and C?

Consistency Models (P3.5)

Strong Consistency:

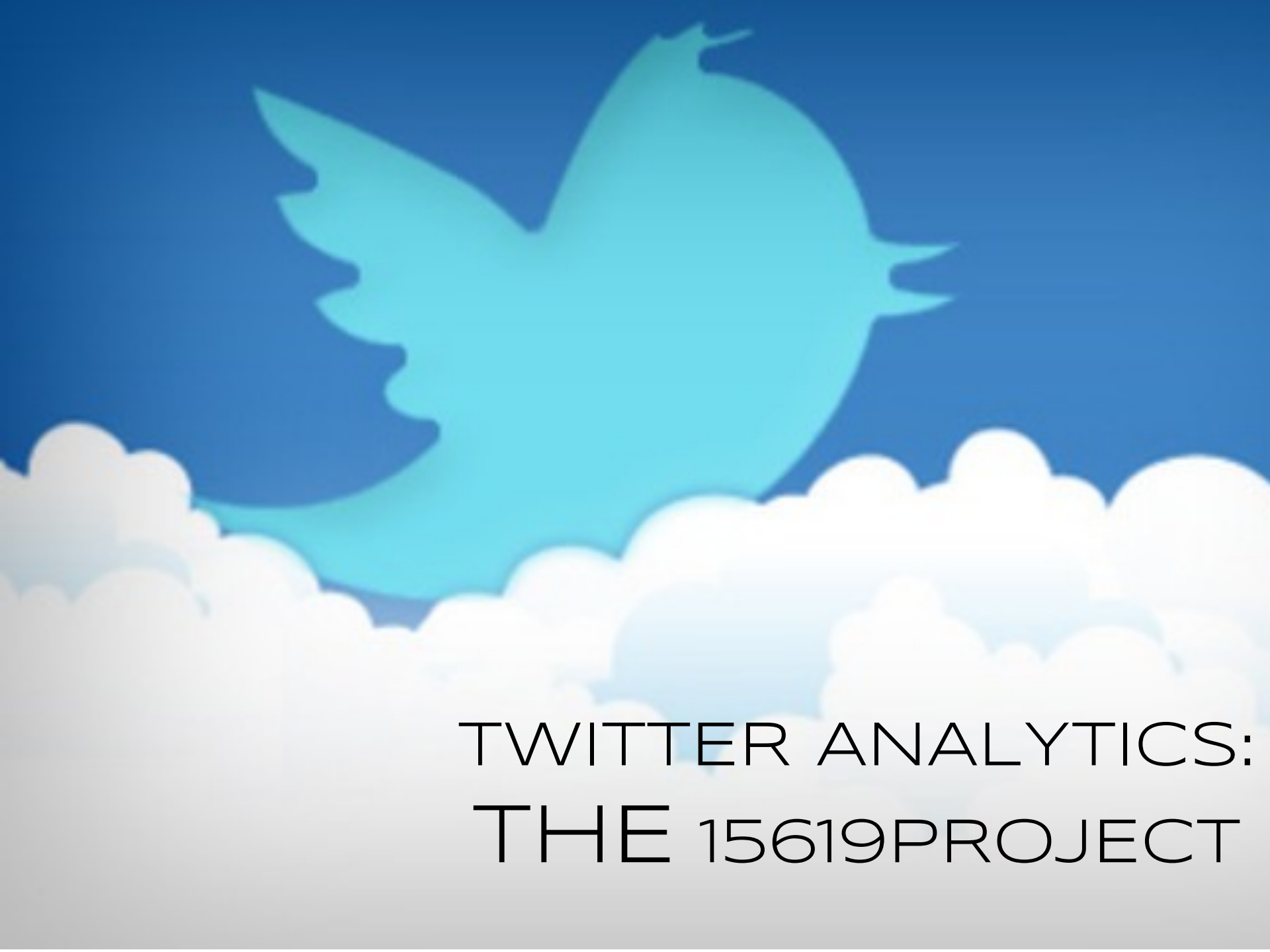
- In the case of P3.5:
  - All datacenters see the operations in order of their timestamps
  - Any PUT operation occurring at a datacenter is instantaneously visible across all datacenters

Causal Consistency:

- In this case of P3.5:
  - All datacenters should see operations on the same object in the order of their timestamps
    - For this project, assume that only PUT operations on the same object are causally related

THE END

Any questions?



TWITTER ANALYTICS:
THE 15619PROJECT

Phase 2 Live Test

Congratulations UnusualItem!

MySQL

Team	Phase Score
UnusualItem	203
ComeATeamName	186
HYPER	160
CloudWalker	149
z2m	99
T.K.Lim	95
Lays	92
Oak	89
Penguins	89
YouKnowWho	86
etc	75

HBase

Team	Phase Score
UnusualItem	90
T.K.Lim	82
YouKnowWho	68
The Three Foes	57
Thunder & Lightning	57
ComeATeamName	49
Wegotateam	46
Oak	44
etc	43
Cloud007	41
9527	41

What's due soon?

- Report at the end of Phase 2
  - **Due by 23:59 ET (Pittsburgh) Thu 4/2**
  - Make sure you highlight failures and learning
  - If you didn't do well, explain why
  - If you did, explain how

What's due soon?

- Phase 3 Deadline
  - **Submission of one URL by 18:59 ET (Pittsburgh) Wed 4/15**
    - **Live Test from 8 PM to midnight ET**
  - Choose one database
  - Fix Q1,Q2,Q3,Q4 if your Phase 2 did not go well
  - New queries Q5 and Q6.
  - Phase 3 counts for 60% of 15619 grade
  - **Live Test!**
    - **Mix-Load**
    - **No more pre-caching of known requests**

Query 5: Twitter Rankster

- Request: a list of userids and a date range
- You should award points to the users based on these rules:
  - +1 per unique tweet sent by the user in the time interval
  - +3 per friend (based on the maximum value of user.friends_count in the time interval)
  - +5 per follower (based on the maximum value of user.followers_count in the time interval)

Query 5: Twitter Rankster

```
GET /q5?userlist=12,14,16,18,20&start=2010-01-
01&end=2014-12-31


Team,1234-5678-1234,1234-5678-1234,1234-5678-1234
12,173
16,155
14,99
20,99
18,55
```

Query 6: Hermit Finder

- Request: A range of userids
- You should count the number of users where:
  - userid is between M and N inclusive,
  - has at least one tweet but none of his/her tweets contain location information.

```
GET /q6?m=0&n=9999999999
```

```
Team,1234-5678-1234,1234-5678-1234,1234-5678-1234
55811730
```

Tips

- Avoid Tagging Penalties
- Keep a watch on budget.
  - $60 (phase + livetest)
- Preparing for the live test
  - You are required to submit two DNS each for MySQL and HBase for the live test
  - Budget limited to $1.75/hr for MySQL and HBase web service separately.
  - Caching known requests will not work(unless you are smart)
  - Need to have all Qs running at the same time

THE END

Any questions?