# 15-319 / 15-619
# Cloud Computing

Recitation 7

February 26, 2019

# Overview

- **Last week's reflection**
  - OLI Unit 3 - Module 10, 11, 12
  - Quiz 5
  - Project 2.3
- **This week's schedule**
  - OLI Unit 3 - Module 13
  - Quiz 6
  - Project 3.1
  - Sign up for the Multi-threaded OPE session
- **Team Project, Twitter Analytics**
  - Phase 1 is out, checkpoint due on Sunday 3/3!

# Last Week

- **Unit 3: Virtualizing Resources for the Cloud**
  - Module 10: Resource virtualization (memory)
  - Module 11: Resource virtualization (I/O devices)
  - Module 12: Case Study
- **Quiz 5**
- **Project 2.3, Functions as a Service (FaaS)**
  - Task 1, Explore functions on various CSPs
    - Azure Functions, GCP Cloud Functions, AWS Lambda
  - Task 2, extract thumbnails from video stream
    - Lambda and FFmpeg
  - Task 3, get image labels and index
    - AWS Rekognition, AWS CloudSearch

# This Week

- OLI : Module 13
  - Storage and network virtualization
- Quiz 6
- Project 3.1, Files v/s Databases
- Sign up for the multi-threaded OPE session
- Team Project, Phase 1 released

# Conceptual Topics - OLI Content

- **Unit 3 - <u>Module 13: Storage and network virtualization</u>**
  - Software Defined Data Center (SDDC)
  - Software Defined Networking (SDN)
    - Device virtualization
    - Link virtualization
  - Software Defined Storage (SDS)
    - IOFlow
- **Quiz 6**
  - **Quizzes must be submitted within 2 hours once you start.**
  - **Remember to click submit!**

# This Week's Project

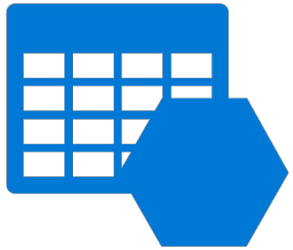
Project 3: Storage and DBs on the cloud

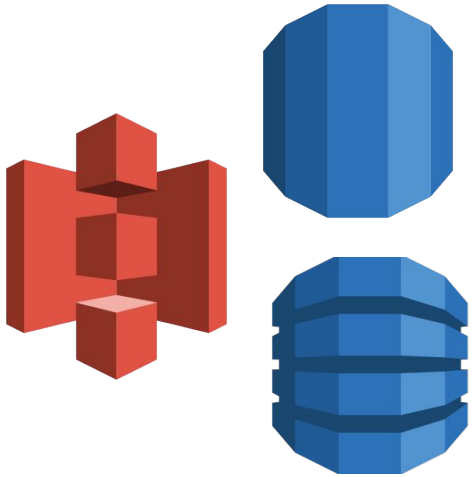
- P3.1: Files and Databases
  - Comparison and usage of Flat files, RDBMS (MySQL) and NoSQL (Redis, HBase)
- P3.2: Social Networking Timeline with Heterogeneous Backends
  - Social Networking Timeline with Heterogeneous Backends (MySQL, Neo4j, MongoDB, S3)
- P3.3: Replication and Consistency
  - Multi-threaded Programming and Consistency

# Project 3 - Storage







# Primers for Project 3

- P3.1: Files, SQL and NoSQL
  - Primer: Storage & IO Benchmarking
  - Primer: NoSQL and HBase primers
- P3.2: Social network with heterogeneous backend storage
  - Primers: MongoDB
- P3.3: Replication and Consistency models
  - Primer: Intro. to Java Multithreading
  - Primer: Intro. to Consistency Models

# Storage & IO Benchmarking Primer

Storage & IO Benchmarking:

- Running sysbench and preparing data
  - Use the prepare option to generate the data.
- Experiments
  - Run sysbench with different storage systems and instance types.
  - Doing this multiple times to reveal different behaviors and results.
- Compare the requests per second.

# Performance Benchmarks Sample Report

| Scenario | Instance Type | Storage Type | RPS Range | RPS Increase Across 3 Iterations |
|---|---|---|---|---|
| 1 | t2.micro | EBS Magnetic Storage | 169.13, 171.81, 181.31 | Trivial (< 5%) |
| 2 | t2.micro | EBS General Purpose SSD | 1465.00, 1473.33, 1490.93 | Trivial (< 5%) |
| 3 | m4.large | EBS Magnetic Storage | 527.70, 973.63, 1246.67 | Significant (can reach ~140% increase with an absolute value of 450-700) |
| 4 | m4.large | EBS General Purpose SSD | 2046.66, 2612.00, 2649.66 | Noticeable (can reach ~30% increase with an absolute value of 500-600) |

What can you conclude from these results?

# I/O Benchmarking Conclusions

- SSD has better performance than magnetic disk
- m4.large instance offers higher performance than a t2.micro instance
- The RPS increase across 3 iterations for m4.large is more significant than that for t2.micro:
  - The reason is an instance with more memory can cache more of the previous requests for repeated tests.
  - Caching is a vital performance tuning mechanism when building high performance applications.

# Project 3.1 Overview

**P3.1: Files, SQL, and NoSQL:**

- Task 1: analyze data in flat files
  - Linux tools (e.g. grep, awk)
  - Data libraries (e.g. pandas)
- Task 2: Explore a SQL database (MySQL)
  - load data, run queries, indexing, auditing
  - plain-SQL v/s ORM
- Task 3: Implement a Key-Value Store
  - prototype of Redis using TDD
- Task 4: Explore a NoSQL DB (HBase)
  - load data, run basic queries

The NoSQL and HBase primers are vital for P3.1

# Flat Files

- Flat files, plain text or binary
  - comma-separated values (CSV) format:
    `Carnegie,Cloud Computing,A,2018`
  - tab-separated values (TSV) format:
    `Carnegie\tCloud Computing\tA\t2018`
  - a custom and verbose format: `Name: Carnegie, Course: Cloud Computing, Section: A, Year: 2018`

# Flat Files

- Lightweight, Flexible, in favor of small tasks
  - Run it once and throw it away
- Performing complicated analysis on data in files can be inconvenient
- Usually flat files should be fixed or append-only
- Writes to files without breaking data integrity is difficult
- Managing the relations among multiple files is also challenging

# Databases

- A collection of organized data
- Database management system (DBMS)
  - Interface between user and data
  - Store/manage/analyze data
- Relational databases
  - Based on the relational model (schema)
  - MySQL, PostgreSQL
- NoSQL Databases
  - Unstructured/semi-structured
  - Redis, HBase, MongoDB, Neo4J

# Databases

- Advantages
  - Logical and physical data independence
  - Concurrency control and transaction support
  - Query the data easily (e.g., SQL)
  - ...

- Disadvantages
  - Cost (computational resources, fixed schema)
  - Maintenance and management
  - Complex and time-consuming to design schema
  - ...

# Files vs. Databases

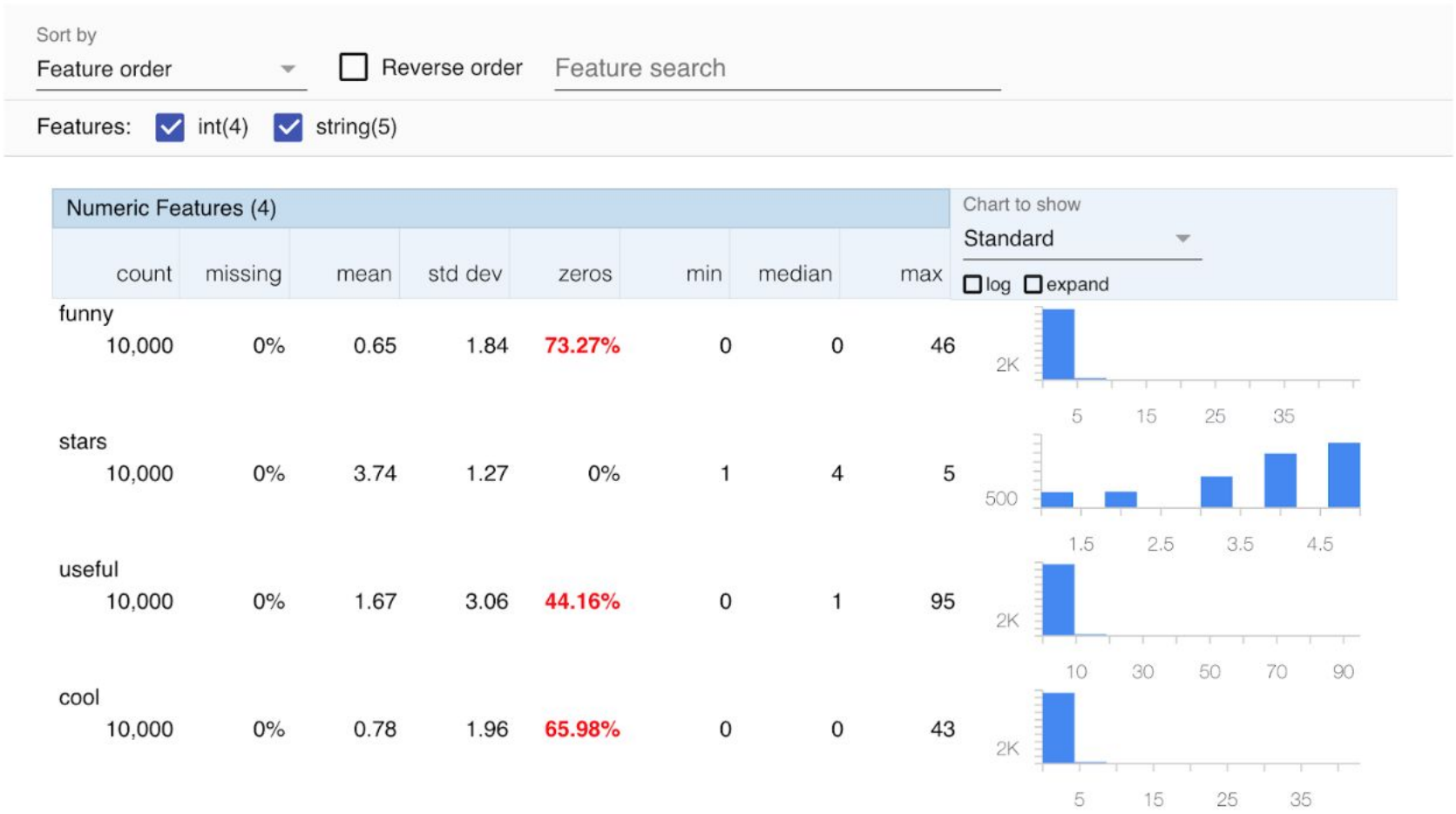
- Compare flat files to databases

- Think about:
  - What are the advantages and disadvantages of using flat files or databases?
  - In what situations would you use a flat file or a database?
  - How to design your own database? How to load, index and query data in a database?

# Dataset

- Analyze Yelp's Academic Dataset
  - https://www.yelp.com/dataset_challenge
  - business
  - checkin
  - review
  - tip
  - user

# Inspect and visualize data using Facets



# Task 1: Flat File

- Answer questions in runner.sh
  - Use tools such as awk and pandas
  - Similar to what you did in Project 1
- Merge TSV files by joining on a common field
- Identify the disadvantages of flat files
- You may use Jupyter Notebook to help you solve the questions in Python.

# Task 2: MySQL

- Prepare tables
  - A script to create the table and load the data is already provided
- Use MySQL queries to answer questions
  - Learn JDBC
  - Complete MySQLTasks.java
  - Aggregate functions, joins
  - Statement and PreparedStatement
  - SQL injection
- Learn how to use proper indexes to improve performance

# MySQL Indexing

- **Schema**
  - The structure of the tables and the relations between tables
  - Based on the structure of the data and the application requirement
- **Index**:
  - An index is simply a pointer to data in a table. It's a data structure (lookup table) that helps speed up the retrieval of data from tables (e.g., B-Tree, Hash indexes, etc.)
  - Based on the data **as well as queries**
- You can build effective indexes only if you are aware of the queries you need
- We have an insightful section about the practice of indexing, read it carefully! **Very helpful for the team project**

# EXPLAIN statements in MySQL

- How do we evaluate the performance of a query?
  - Run it.

- What if we want/need to predict the performance without execution?
  - Use EXPLAIN statements.

- An EXPLAIN statement on a query will predict:
  - The number of rows to scan
  - Whether it makes use of indexes or not
  - etc.

# Object Relational Mapping (ORM)

ORM is a technique to abstract away the work for you to:

1. Map the domain class with the database table
2. Map each field of the domain class with a column of the table
3. Map instances of the classes (objects) with rows in the corresponding tables

| | Mapped to | |
|---|---|---|
| public class Course { | → | course |
| String courseId; | → | course_id (PK) |
| String name; | → | name |
| } | | |
| Domain Class | → | Database Table |
| Objects | → | Rows |

# Benefits of adopting ORM

- Separation of concerns
  - ORM decouples the CRUD operations and the business logic code.
- Productivity
  - You don’t need to keep switching between your OOP language such as Java/Python, etc. and SQL.
- Flexibility to meet evolving business requirements
  - ORM cannot eliminate the schema update problem, but it may ease the difficulty, especially when used together with data migration tools.
- Persistence transparency
  - Any changes to a persistent object will be automatically propagated to the database without explicit SQL queries.
- Vendor independence
  - ORM can abstract the application away from the underlying SQL database and SQL dialect.

# ORM Question in the MySQL Task

- The current business application exposes an API that returns the most popular Pittsburgh businesses.
- It is based on a SQLite3 database with an outdated schema.
- Your task:
  - Plug the business application to the MySQL database and update the definition of the domain class to match the new schema.
- The API will be backward compatible without modifying any business logic code.

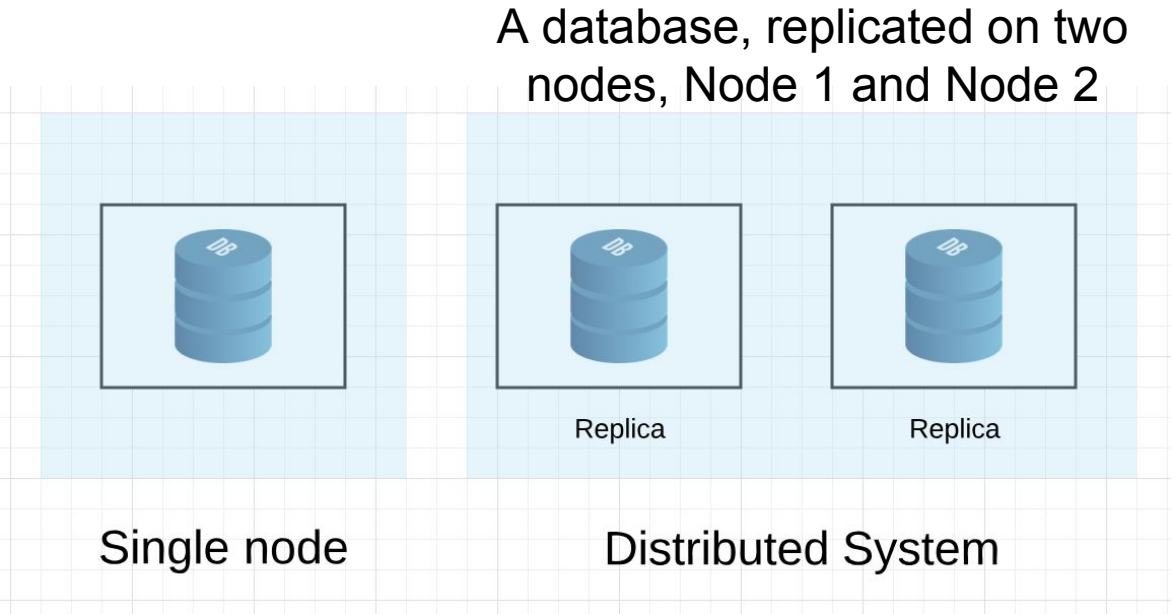
# NoSQL

- Non-SQL or NotOnly-SQL
  - Non-relational
- Why NoSQL if we already have SQL solutions?
  - Flexible data model (schemaless, can change)
  - Designed to be distributed (scale horizontally)
  - Certain applications require improved performance at the cost of reduced data consistency (data staleness)
- Basic Types of NoSQL Databases
  - Schema-less Key-Value Stores (Redis)
  - Wide Column Stores (Column Family Stores) (HBase)
  - Document Stores (MongoDB)
  - Graph DBMS (Neo4j)

# **CAP** Theorem

- The CAP theorem: it is impossible for a distributed data store to provide all the following three guarantees at the same time
  - Consistency: no stale data
  - Availability: no downtime
  - Partition Tolerance: network failure tolerance in a distributed system

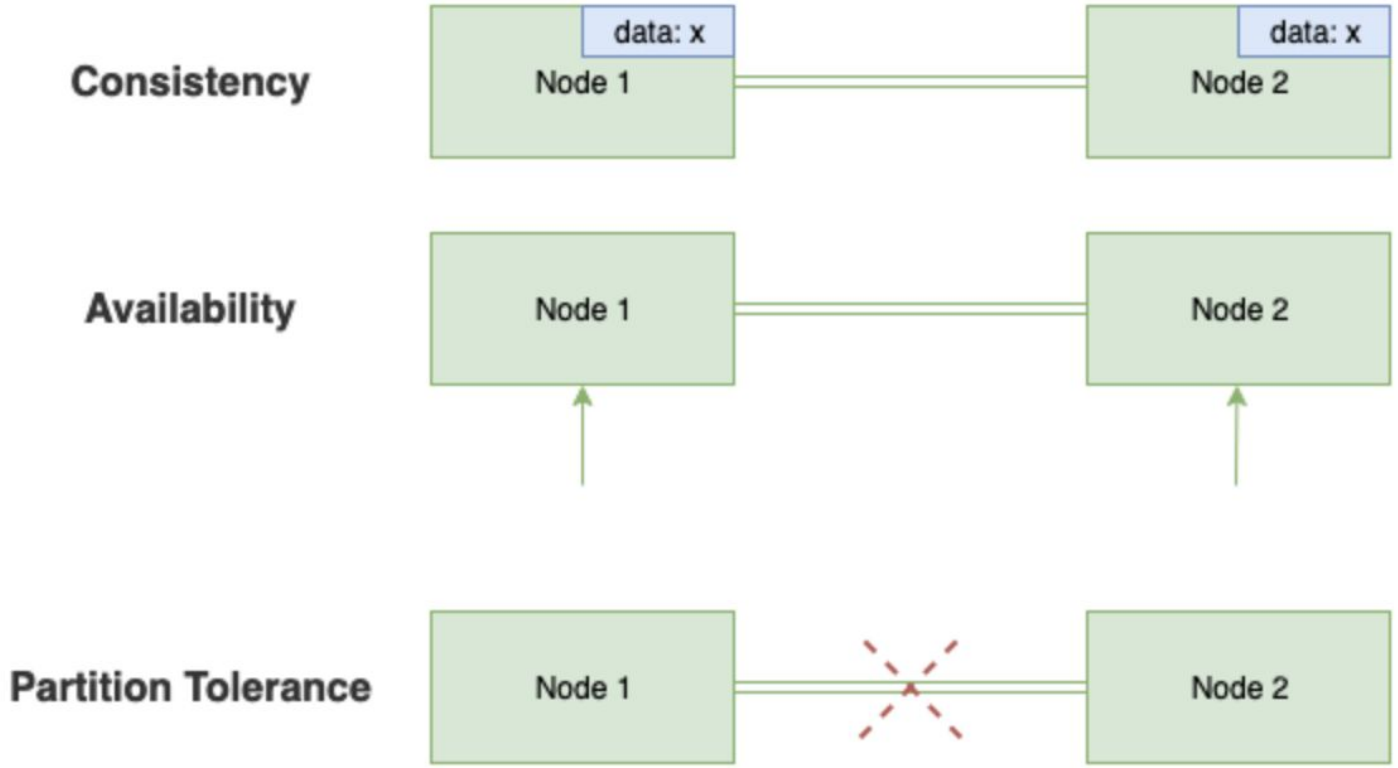
# Single Node to Distributed Databases




Three issues emerge:

- Since DB is replicated, how to maintain consistency?
- Since the data is replicated, if one replica is down, is the entire service down?
- How will the service behave during a network failure?

# C, A, P in Distributed Databases

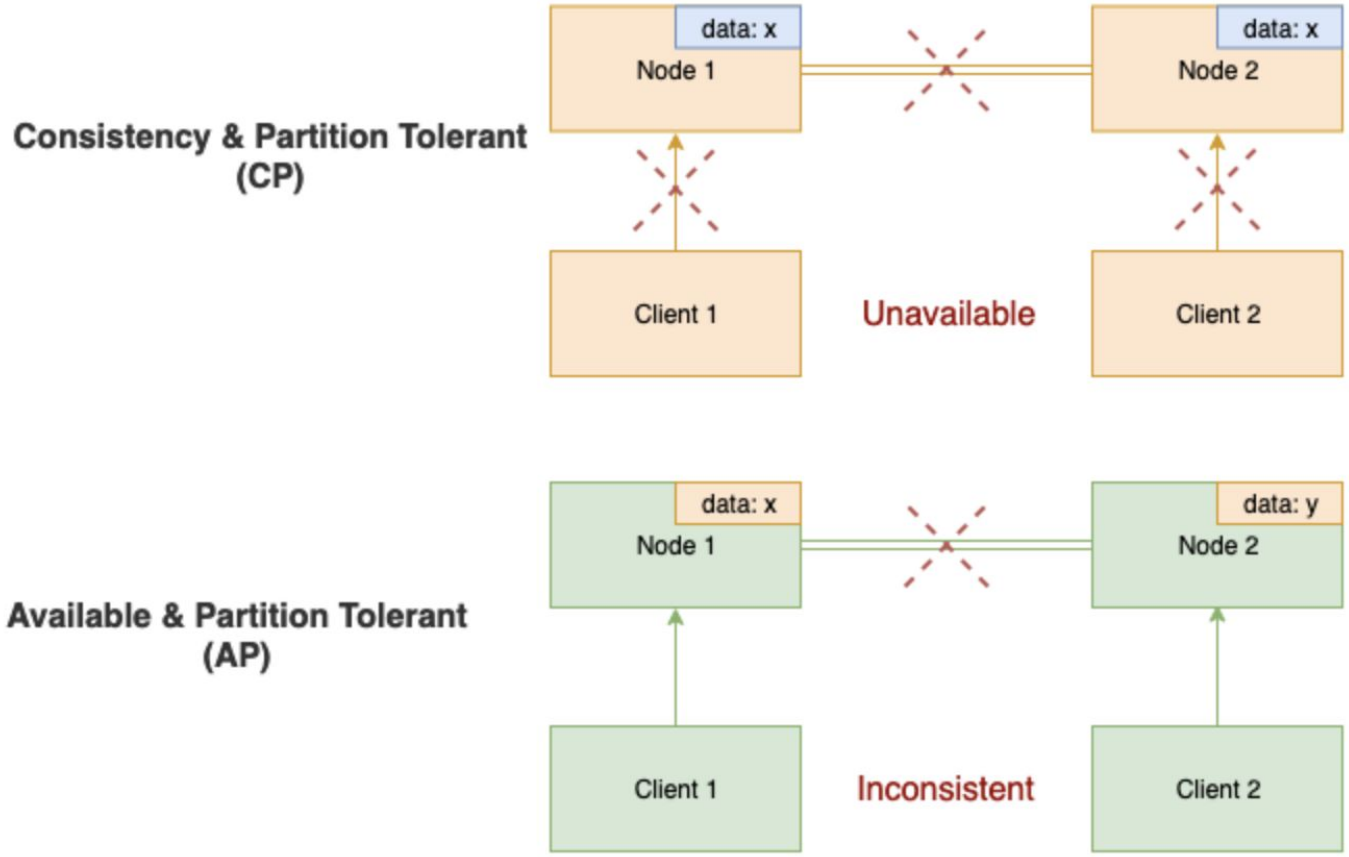


# CAP Theorem

Only two out of the three are feasible:

- CA: non-distributed (MySQL, PostgreSQL)
  - Traditional databases like MySQL and PostgresQL have only one server. They meet the requirement of CA and don’t provide partition tolerance
- CP: downtime (HBase, MongoDB)
  - Stop responding if there is partition. There will be downtime
- AP: stale data (Amazon DynamoDB)
  - Always available. Data may be inconsistent among nodes if there is a partition

# Only two out of three in CAP are feasible



# Task 3: Implement Redis

- Key-value store is a type of NoSQL database e.g., Redis and Memcached
- Widely used as an in-memory cache

Your task:

- Implement a simplified version of Redis
- We provide starter code `Redis.java`, you will implement two of the commonly used data structures supported by Redis:
  - hashes and lists
- TDD with 100% code coverage

# Task 4: Explore HBase

HBase is an open source, column-oriented, distributed database developed as part of the Apache Hadoop project

Steps to complete:

1. Launch an HDInsight cluster with HBase installed.
2. Follow the write-up to download and load the data into HBase.
3. Try different querying commands in the HBase shell.
4. Complete HBaseTasks.java using HBase Java APIs.

# P3.1 Reminders

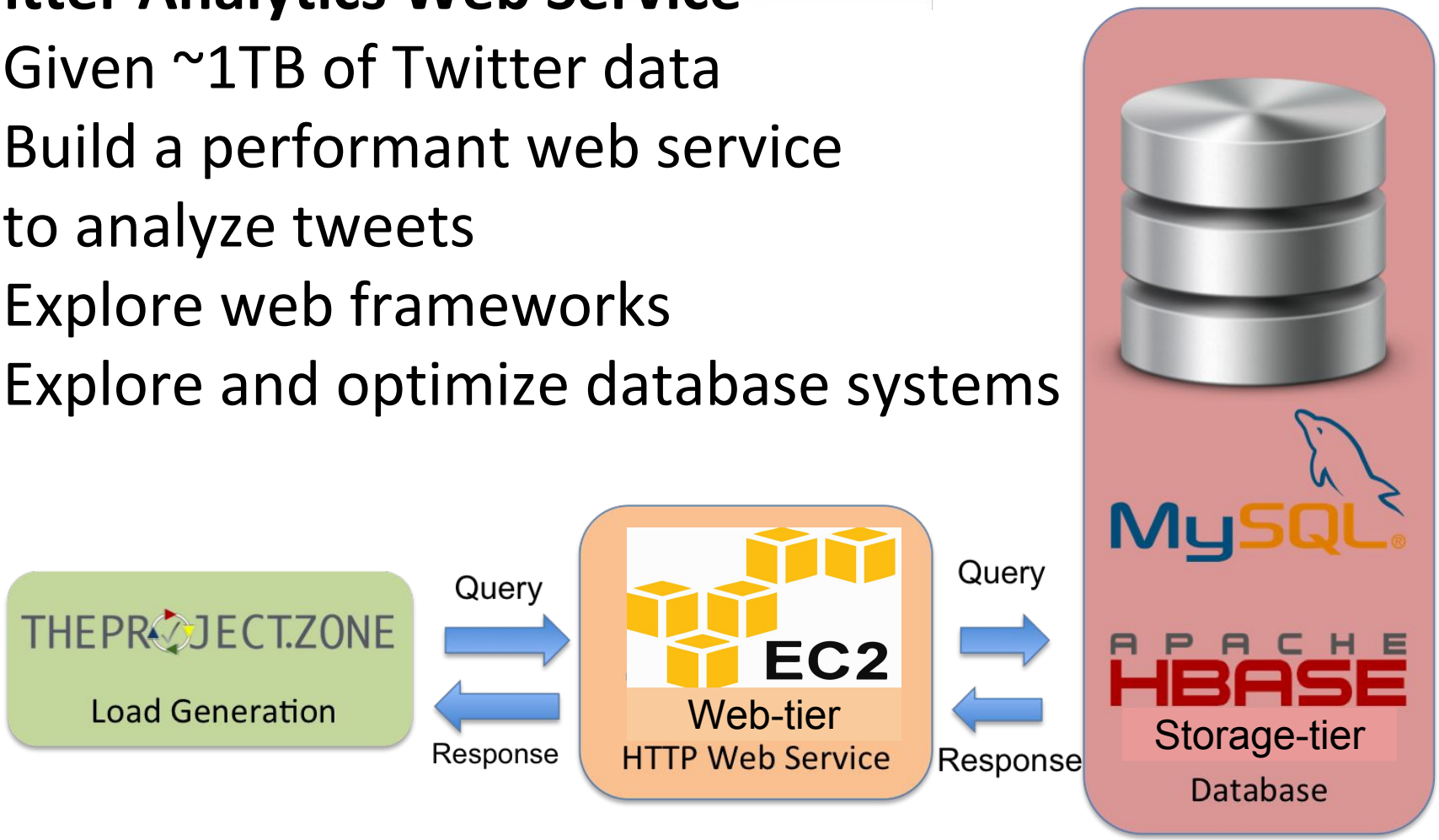
- Tag your resources with:
  - Key: Project, Value: 3.1
- An HDInsight cluster is very expensive.
- Your subscription will be disabled if you run out of your subscription budget. Please exercise caution to plan the budget.
- Remember to delete the Azure resource group to clean up all the resources in the end.

# TEAM PROJECT
# Twitter Data Analytics

# Team Project

**Twitter Analytics Web Service**

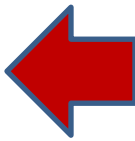
- Given ~1TB of Twitter data
- Build a performant web service to analyze tweets
- Explore web frameworks
- Explore and optimize database systems



# 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 (Managed Cloud Services)

# Query 1 (CloudCoin)

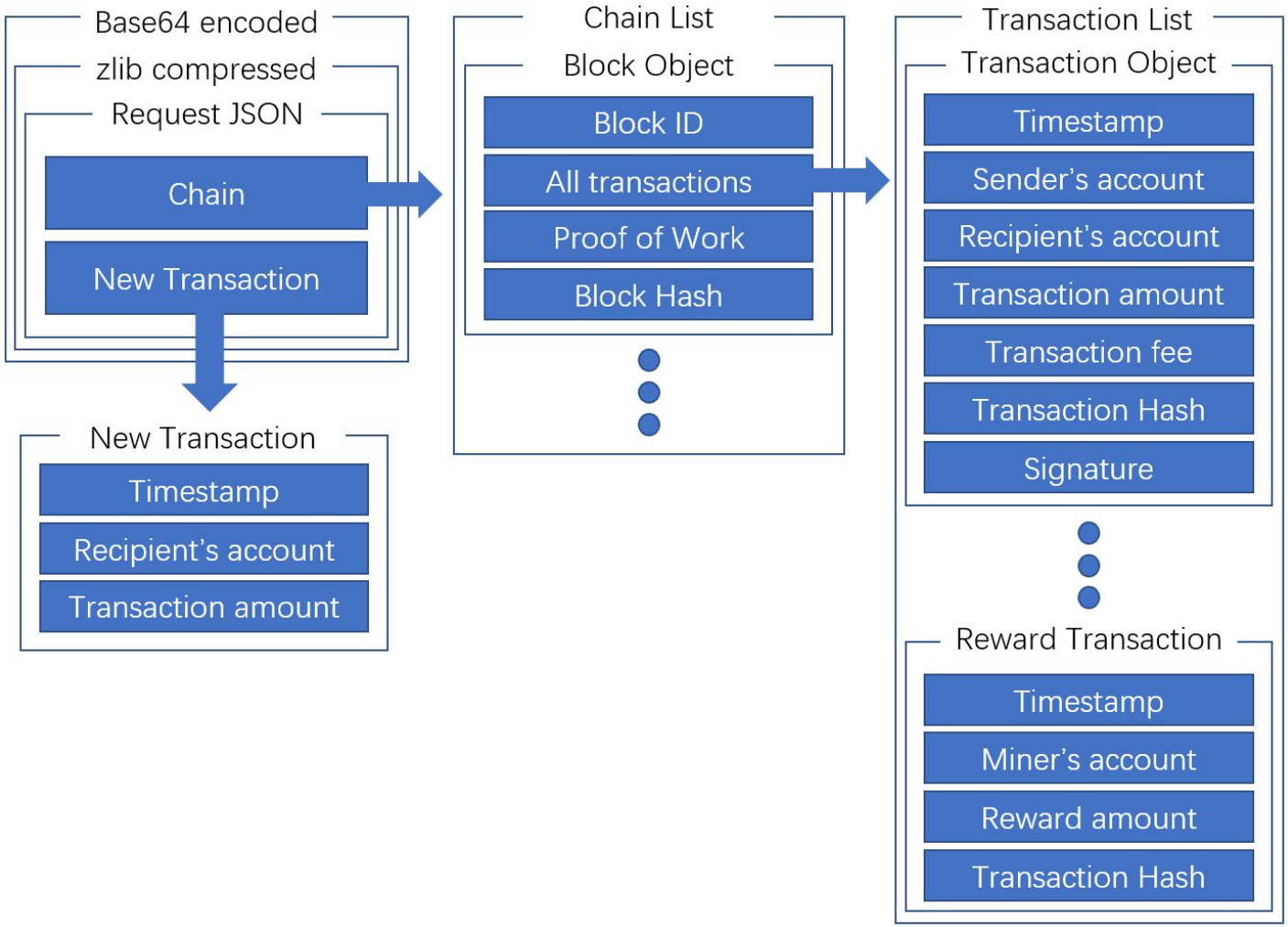
- Query 1 does not require a database (storage tier)
- Implement a web service that verifies and updates blockchains.
- You must explore different web frameworks
  - Get at least 2 different web frameworks working
  - Select the framework with the better performance
  - Provide evidence of your experimentations
  - Read the report first

# What is a blockchain, though?

- Data structure that supports digital currency.
- Designed to be untamperable.
- Distributed. Shared among all user nodes.
  - Decentralized
  - Fault Tolerant.
- Consists of chained blocks.
- Each block consists of transactions.

# Q1 Example

- Q1 input

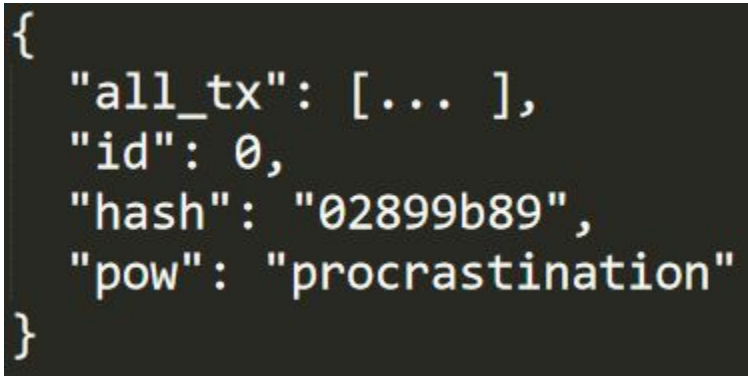




```
{
  "chain": [
    {
      "all_tx": [
        {
          "recv": 509015179679,
          "amt": 500000000,
          "time": "1550721967779362304",
          "hash": "d50e5266"
        }
      ],
      "id": 0,
      "hash": "02899b89",
      "pow": "postpone"
    },
    {
      "all_tx": [
        {
          "send": 509015179679,
          "recv": 484054352161,
          "amt": 126848946,
          "fee": 12488,
          "time": "1550721967779391744",
          "hash": "5a2b4d71",
          "sig": 463884077351
        },
        {
          "recv": 1284110893049,
          "amt": 500000000,
          "time": "1550721967779424000",
          "hash": "7924c55e"
        }
      ],
      "id": 1,
      "hash": "0fce51c1",
      "pow": "fountain"
    },
    {
      "all_tx": [
        {
          "send": 1284110893049,
          "recv": 484054352161,
          "amt": 58759591,
          "fee": 5048,
          "time": "1550721967779447040",
          "hash": "b43737af",
          "sig": 1084970046728
        },
        {
          "recv": 34123506233,
          "amt": 500000000,
          "time": "1550721967779474176",
          "hash": "d705e74e"
        }
      ],
      "id": 2,
      "hash": "03635f77",
      "pow": "jeans"
    }
  ],
  "new_tx": {
    "recv": 837939704897,
    "amt": 430642077,
    "time": "1550721967779486720"
  }
}
```

# Q1 Example

- Block:
  - Created by “miners”.
  - Has a list of transactions.
  - Block hash encapsulates all transaction info and block Metadata, as well as the hash of the previous block.
  - Block hash, required to start with a 0.
  - PoW (Proof of Work), which makes the hash start with a 0.
  - PoW is found by miner through brute forcing.



# Q1 Example

- Transaction:
  - Hash value computed using all info in the blue box.
  - Signature is computed with hash value using RSA.
    sig=RSA(hash, key)


```
{
  "send": 1284110893049,
  "recv": 484054352161,
  "amt": 58759591,
  "fee": 5048,
  "time": "15507219677794470 40",
  "hash": "b43737af",
  "sig": 1084970046728
},
```

# Q1 Example

- Reward:
  - Special type of transaction.
  - Created by miner.
  - Is the last transaction in the block's transaction list.


```
{
  "recv": 34123506233,
  "amt": 500000000,
  "time": "15507219677794741 76",
  "hash": "d705e74e"
}
```

# Q1 Example

- New transaction:
  - You need to fill in missing fields.
  - You also need to sign the transaction using the key given to you.

```
"new_tx": {
  "recv": 837939704897,
  "amt": 430642077,
  "time": "15507219677794867 20"
}
```

# Q1 Example

- Output:
  - Complete the new transaction.
  - Create a reward transaction.
  - Mine a new block that only has those two transactions.
  - Return the new transaction signature and new block PoW.
  - E.g. <125648413415l|i_love_cc>

# Q1 Example

- Output:
  - There will be malicious attempts to break the blockchain.
  - You need to check the validity of the chain.
  - If the chain is not valid, return INVALID.
  - E.g. <INVALID|any_debug_info_you'd_like>

# Query 2 - User Recommendation System

**Use Case**: When you follow someone on twitter, recommend close friends.

**Three Scores**:

- Interaction Score - closeness
- Hashtag Score - common interests
- Keywords Score - to match interests

**Final Score**: Interaction Score * Hashtag Score * Keywords Score

**Query**:

GET /q2?
user_id=<ID>&
type=<TYPE>&
phrase=<PHRASE>&
hashtag=<HASHTAG>

**Response**:

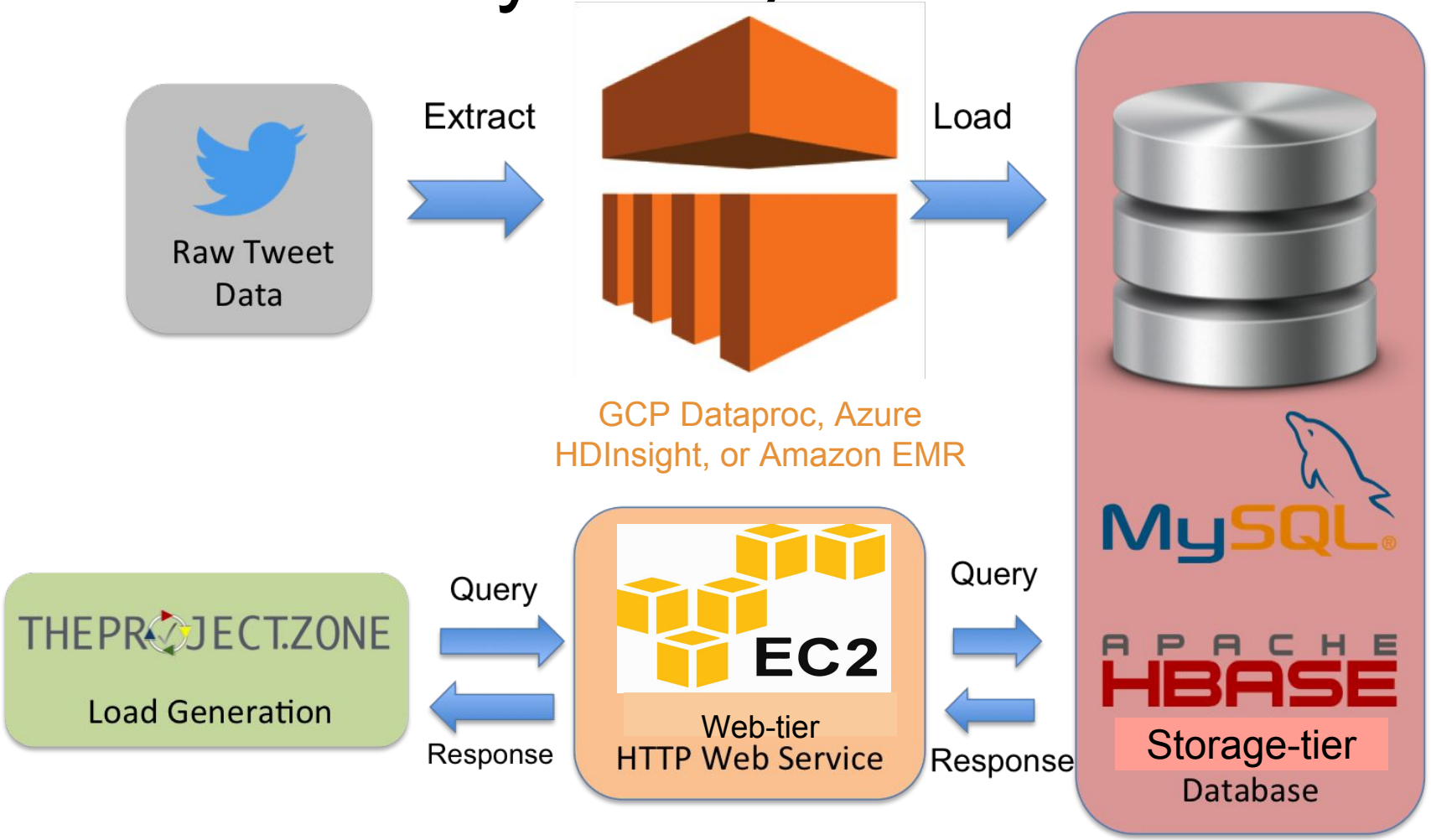
<TEAMNAME>,<AWSID>\n
uid\tname\tdescription\ttweet\n
uid\tname\tdescription\ttweet

# Query 2 Example

```
GET /q2?
user_id=100123&
type=retweet&
phrase=hello%20cc&
hashtag=cmu

TeamCoolCloud,1234-0000-0001
100124\tAlan\tScientist\tDo machines think?\n
100125\tKnuth\tprogrammer\thello cc!
```

# Twitter Analytics System Architecture




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

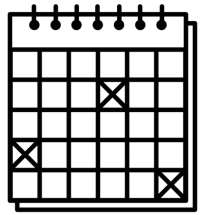
# Git workflow

- 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 changes
  - This process will allow you to capture bugs and remain informed on what others are doing

# Heartwarming Tips from Your Beloved TAs

1. Design your architecture early and apply for limit increase.
2. EC2 VM is not the only thing that costs money.
3. Primers and individual projects are helpful.
4. You don’t need all your hourly budget to get Q1 target.
5. Coding is the least time consuming part.
6. Think before you do. Esp. for ETL (Azure, GCP, or AWS).
7. Divide workload appropriately. Take up your responsibility.
8. Read the write-up.
9. Read the write-up again.
10. Start early. You cannot make-up the time lost. Lots to finish.
11. I’m not kidding. Drama happens frequently.

# Team Project Time Table

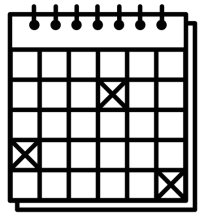


| Phase | Deadline (11:59PM EST) |
|---|---|
| **Phase 1 (20%)**<br>- **Query 1**<br>- **Query 2** | • **Q1 CKPT (5%): Sun, 3/3**<br>• **Report1 (5%): Sun, 3/3**<br>• Q1 FINAL (10%): Sun, 3/10<br>• Q2M & Q2H CKPT (10%): Sun, 3/24<br>• Q2M & Q2H FINAL (50%): Sun, 3/31<br>• Report2 (20%): Tue, 4/2 |
| **Phase 2 (30%)**<br>- **Add Query 3** | • Live Test on Sun, 4/14 |
| **Phase 3 (50%)**<br>- **Managed Services** | • Live Test on Sun, 4/28 |

# Team Project Deadlines - Phase 1

- Writeup and queries were released on Monday.
- Phase 1 milestones:
  - Q1 Checkpoint: **Sunday, Mar 3**
    - A successful 10-min submission for Q1
    - Checkpoint 1 Report (link)
  - Q1 final due: **Sunday, Mar 10**
    - Achieve the Q1 target
  - Q2 Checkpoint: **Sunday, Mar 24**
    - A successful 10-min submissions:
      - Q2 MySQL **and** Q2 HBase.
  - Q2 final due: **Sunday, Mar 31**
    - Achieve the Q2 target for Q2 MySQL **and** Q2 HBase.
  - Phase 1, code and report: **Tuesday, Apr 2** (link)
- Start early, read the report and earn bonus points!

# Suggested Tasks for Phase 1



| Phase 1 weeks | Tasks | Deadline |
|---|---|---|
| **Week 1**<br>● **2/25** | ● **Team meeting**<br>● **Writeup**<br>● **Complete Q1 code & achieve correctness**<br>● **Q2 Schema, think about ETL** | ● **Q1 Checkpoint due on 3/3**<br>● **Checkpoint Report due on 3/3** |
| Week 2<br>● 3/4 | ● Q1 target reached<br>● Q2 ETL & Initial schema design completed | ● **Q1 final target due on 3/10** |
| Week 3<br>● Spring Break | ● Take a break or make progress (up to your team) | |
| Week 4<br>● 3/18 | ● Achieve correctness for both Q2 MySQL and HBase & basic throughput | ● Q2 MySQL Checkpoint due on 3/24<br>● Q2 HBase Checkpoint due on 3/24 |
| Week 5<br>● 3/25 | ● Optimizations to achieve target throughputs for Q2 MySQL and HBase | ● Q2 MySQL final target due on 3/31<br>● Q2 HBase final target due on 3/31 |

# This Week's Deadlines

- Quiz 6:

  Due: Friday, March 1st, 2019 11:59PM ET

- Sign up for the Multi-Threading OPE task

  Due: Saturday, March 2nd, 2019 11:59PM ET

- Project 3.1: Files and Databases

  Due: Sunday, March 3rd, 2019 11:59PM ET

- Team Project Phase 1 Q1 Checkpoint 1

  Due: Sunday, March 3rd, 2019 11:59PM ET

# START EARLY AND PLAN AHEAD

# Q&A