W205
Storage & Retrieval

Live Session - Week 4

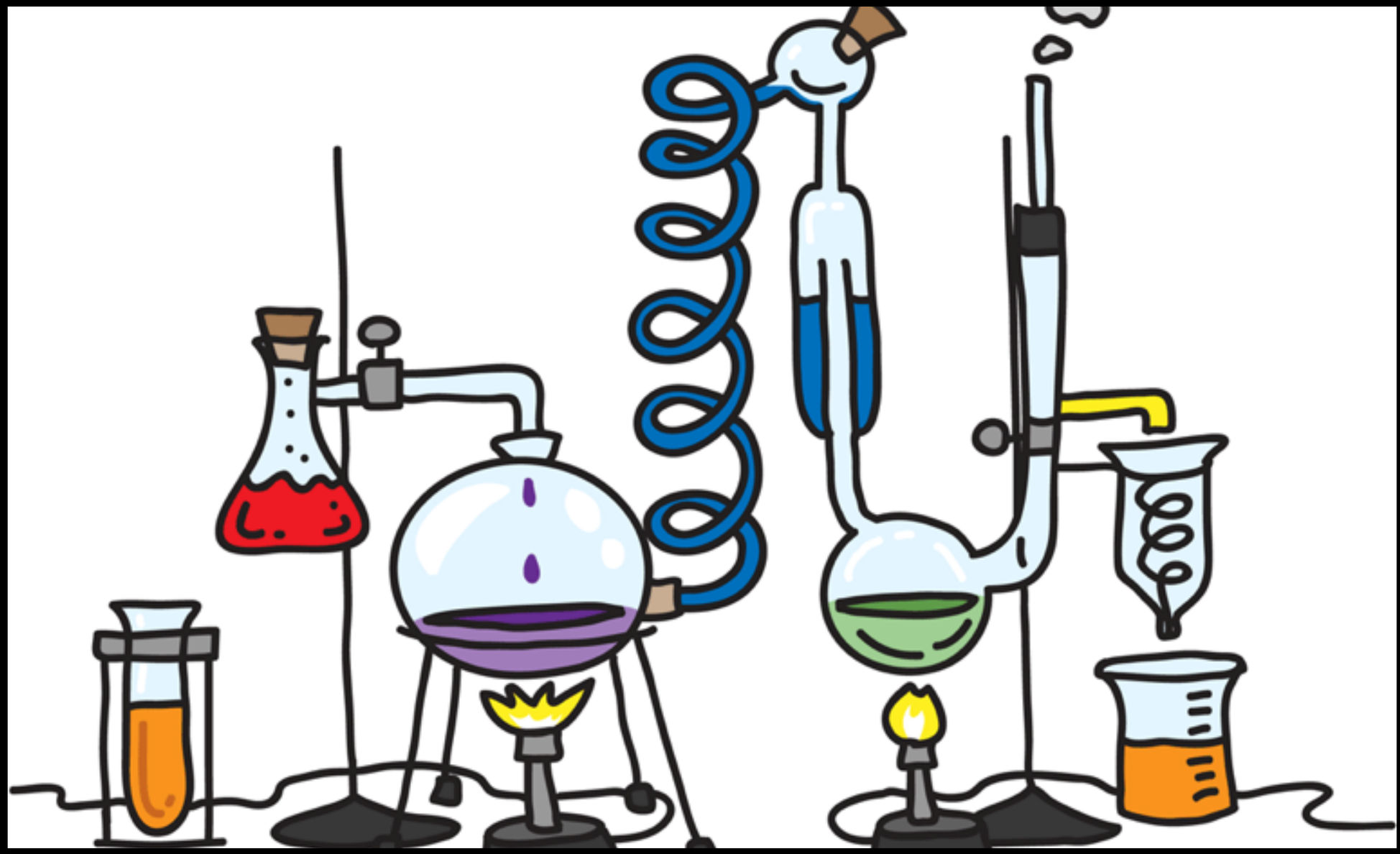
Uri Schonfeld
shuri@berkeley.edu

datascience@berkeley

Checklist

- 1. Start audio
- 2. Enable webcam for participants and turn my webcam on.
- 3. Clear notes and chats, one by one, and clear them.
- 4. Check breakout rooms, make sure all cleared up.
- 5. Start recording before students come in, or have a reminder.
- 6. Have student be a recording reminder.
- 7. Merge students audio.
- 8. While people come in, drag them to their breakout room.

Uri Schonfeld
shuri@berkeley.edu



Lab Grading

Graded labs.

Uri Schonfeld
shuri@berkeley.edu

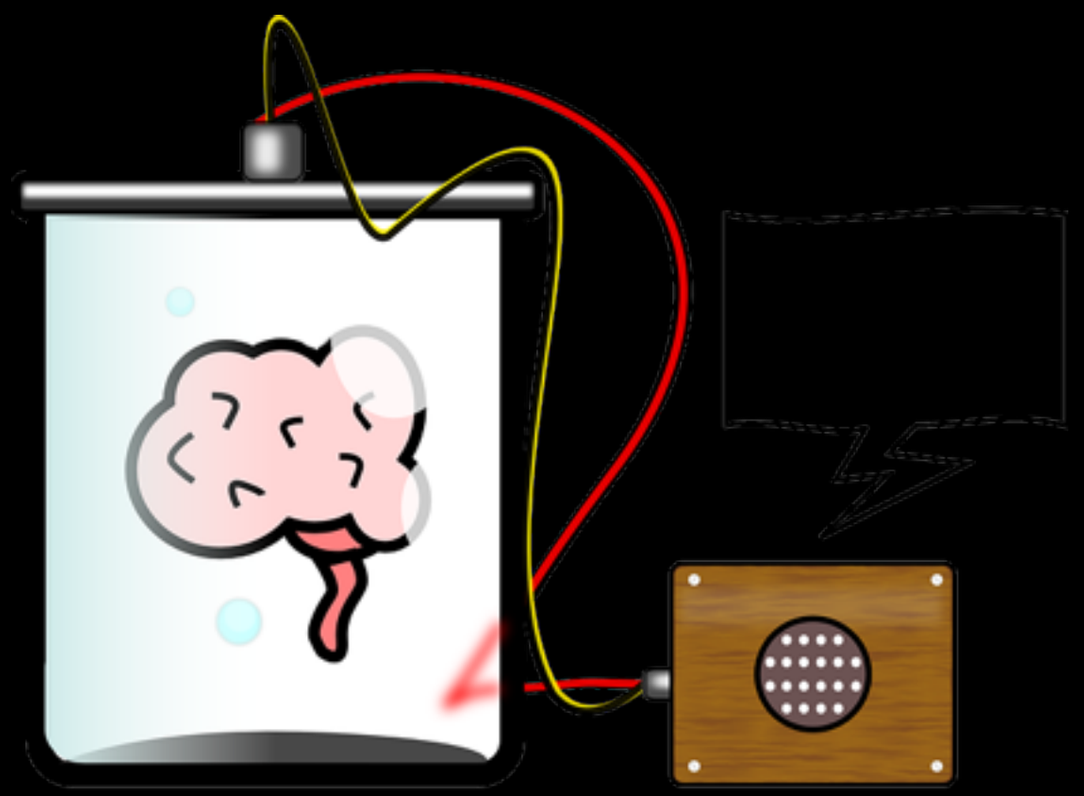
datascience@berkeley

Lab Grading

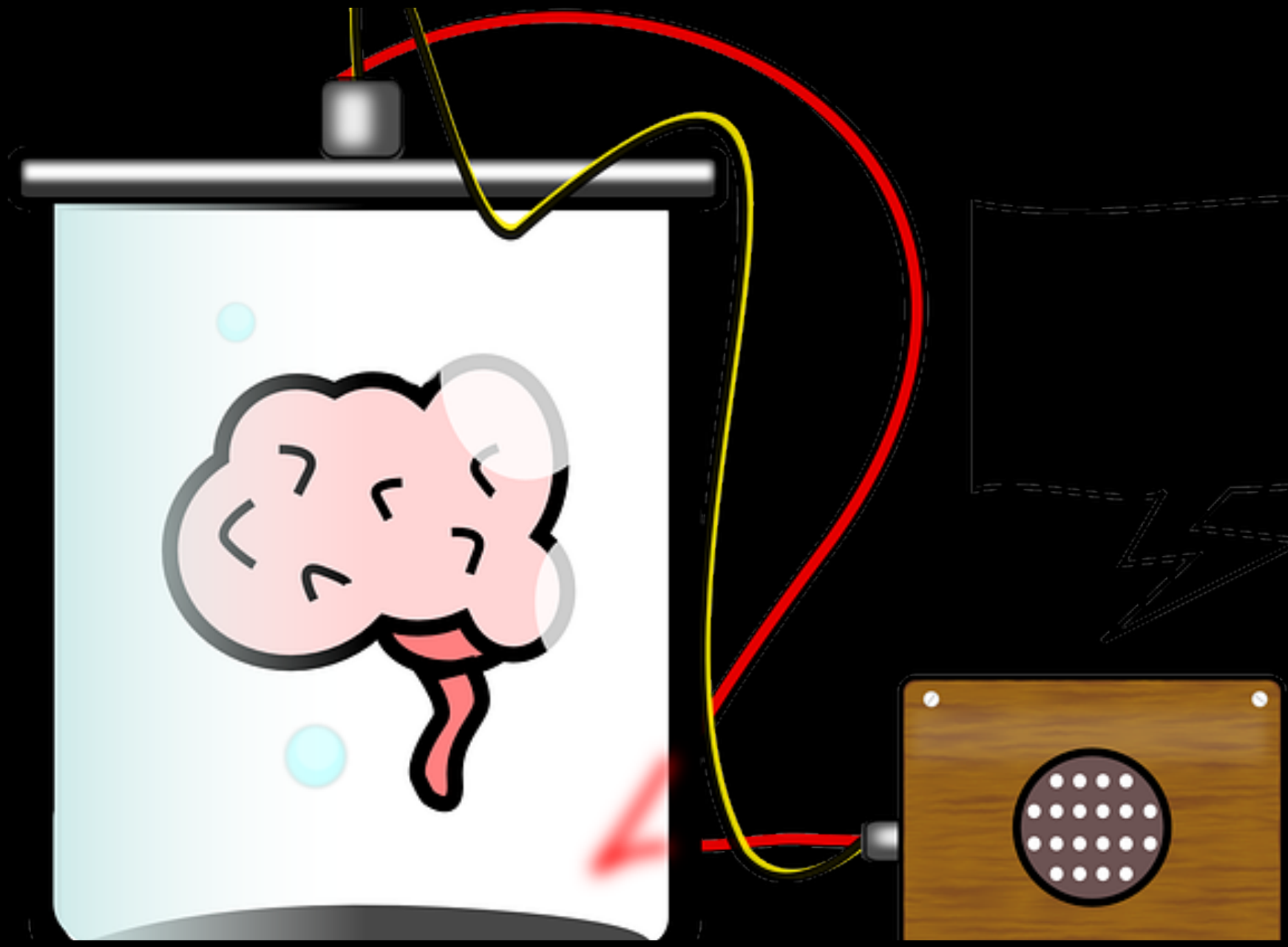
- Don’t lightly adapt or copy paste things.
- Lot’s of good answers. I deducted points if things were wrong or if what I was checking for wasn’t there.
- S3 vs EBS: Many differences. I wanted to stress the different semantics. EBS block storage (not hierarchical, not file system) and S3 is an object store.
- When to use cloud infrastructure?
  (and not local clusters / local cloud / distributed systems)
- Spot vs Reserved: Spot may go away,

Project

- Groups of 3, max 4.
- You need to form your groups.



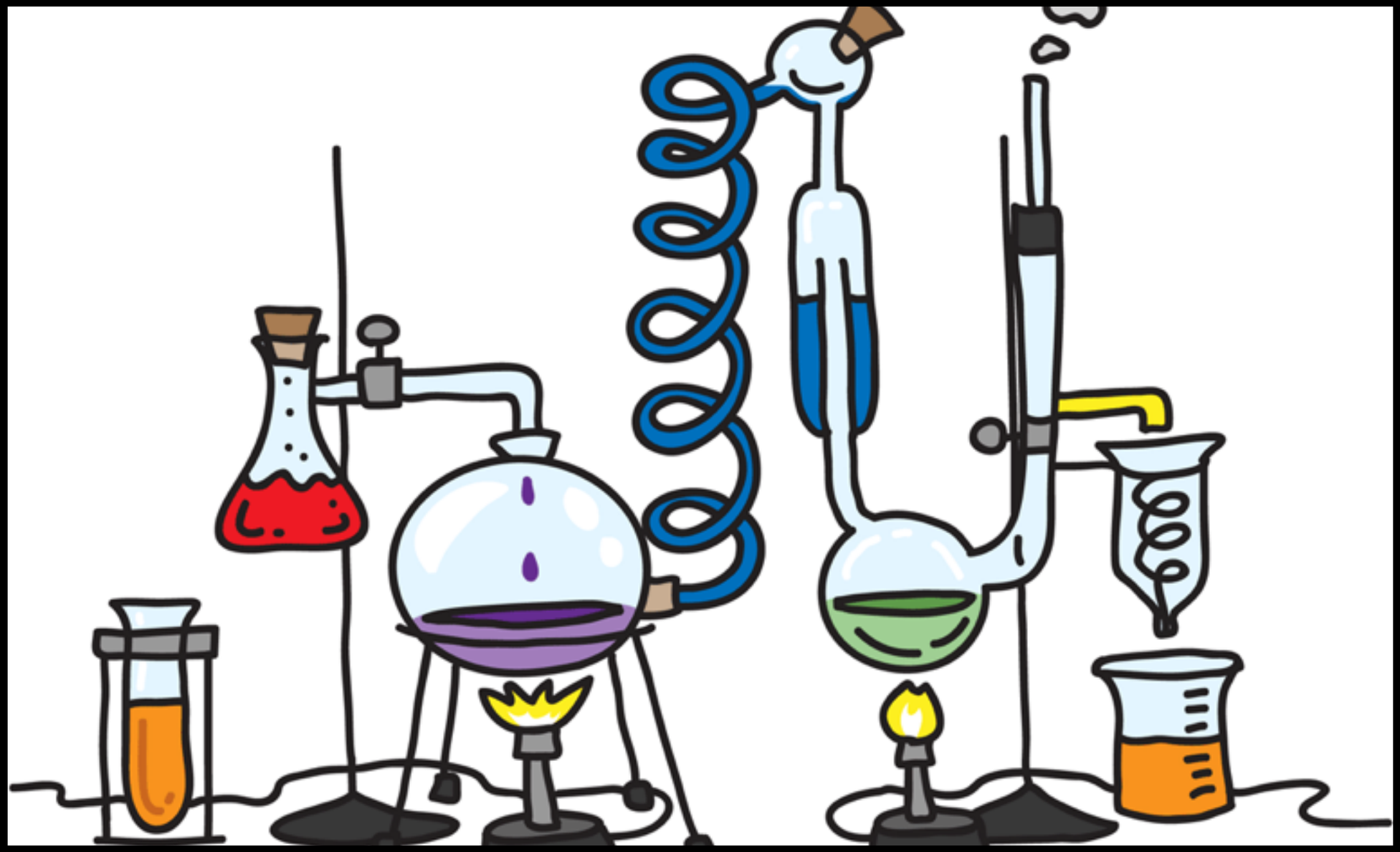
Uri Schonfeld
shuri@berkeley.edu



Breakout solution!

Posted, in the files section.

Uri Schonfeld
shuri@berkeley.edu



Next Lab

Next lab is Goooood :) !

Uri Schonfeld
shuri@berkeley.edu

Next lab

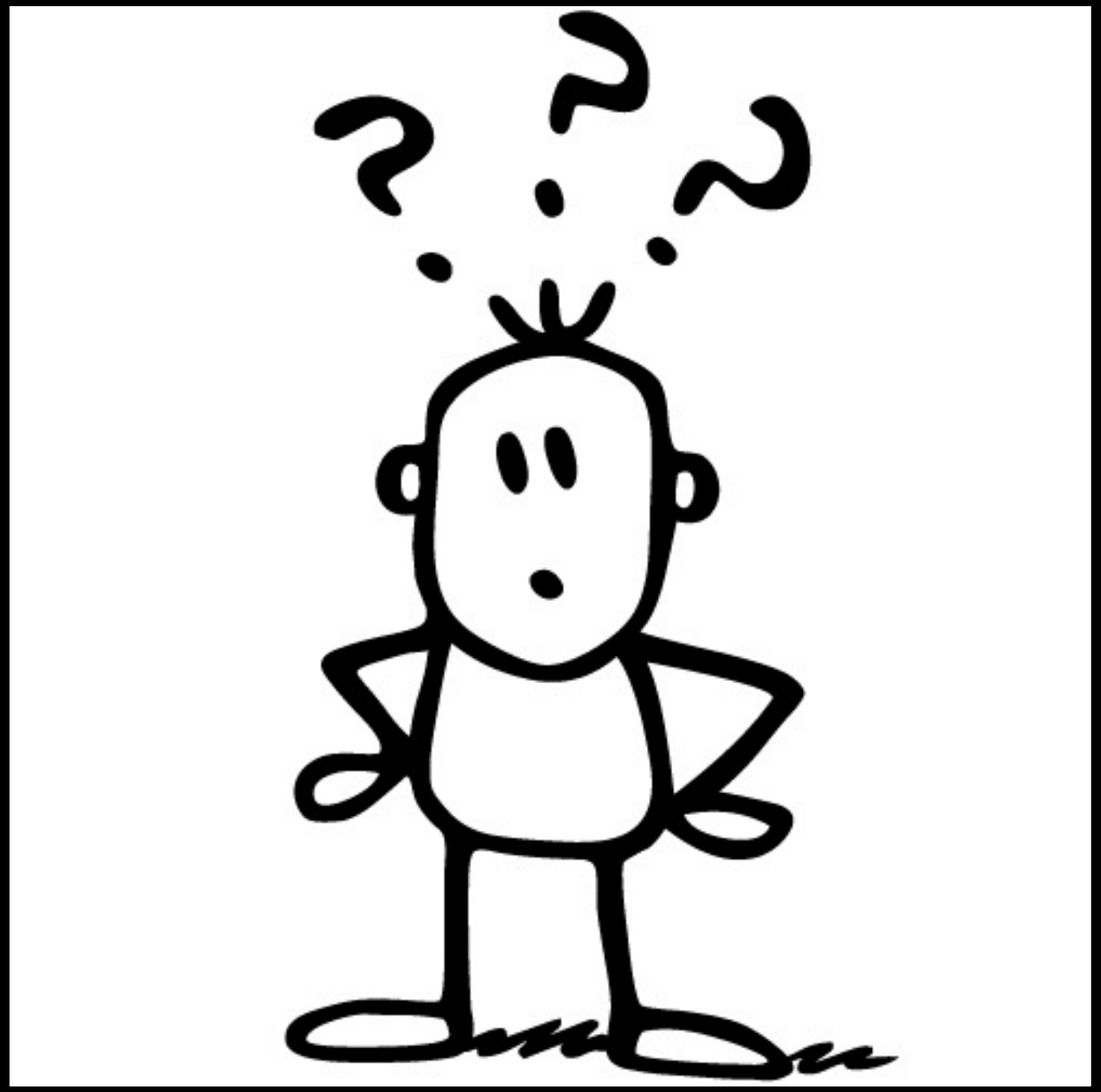
- You'll do Hive.
- You'll see how two "schema"s can be applied to the same exact file.
- You'll run spark.
- Good start.

Uri Schonfeld
shuri@berkeley.edu

Exercise #1

- questions:
  - loading data after creating the table rather than external table.
  - Does the ER include attributes?
  - Does the ER match the hospitals schema exactly?
  - Should the create statement match the hopsital files exactly?

Uri Schonfeld
shuri@berkeley.edu



Questions

Uri Schonfeld
shuri@berkeley.edu

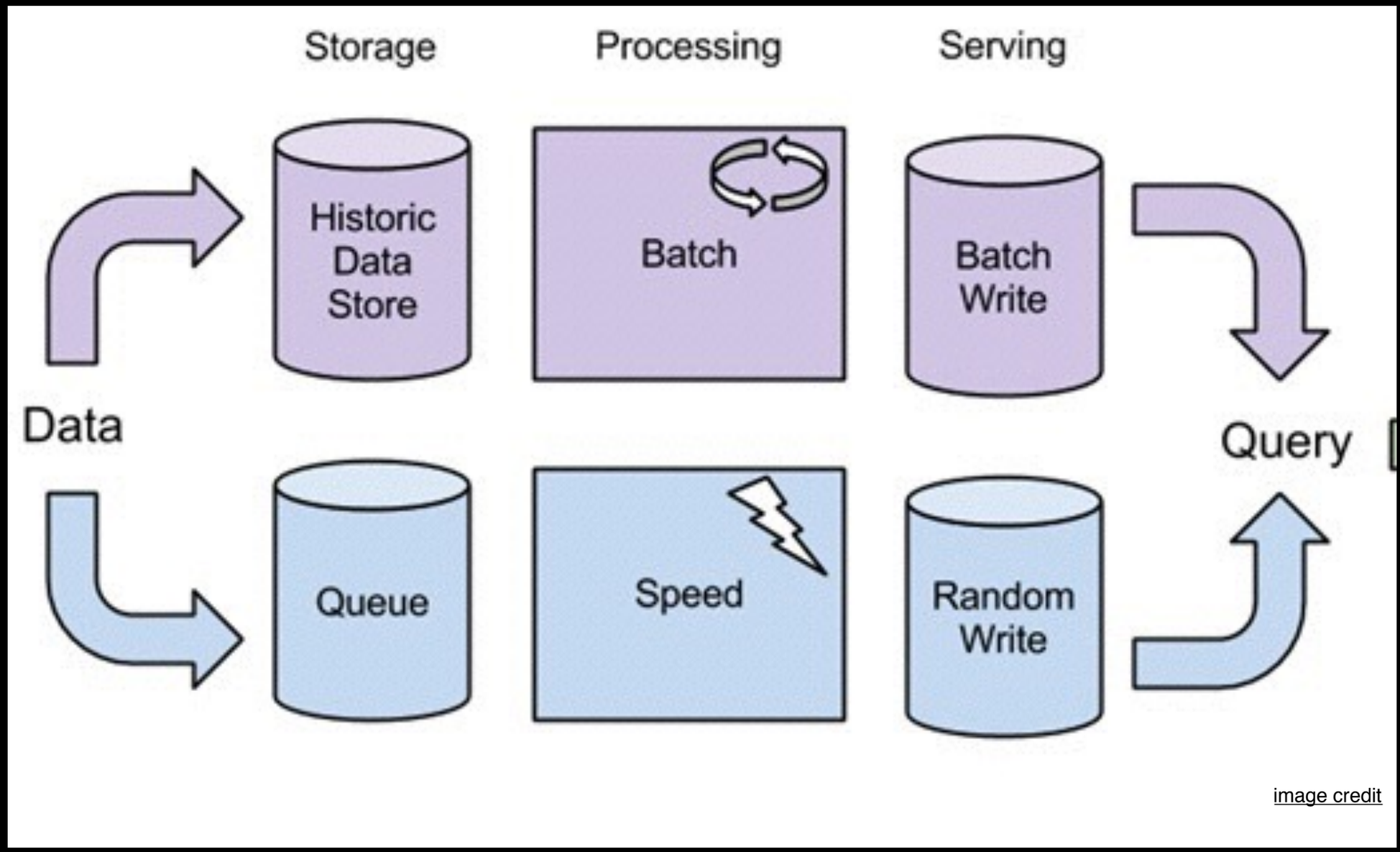


image credit

Lambda Architecture

Uri Schonfeld
shuri@berkeley.edu

Lambda Architecture

- http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html
  - “Yet there's a form of fault-tolerance that's much more important than machine fault-tolerance: human fault-tolerance.”
  - “**We'll store data in flat files on HDFS. A file will contain a sequence of data records. To add new data, you simply append a new file containing new data records to the folder that contains all the data**. “
- “ ElephantDB and Voldemort read-only specialize in exporting key/value data from Hadoop for fast querying. These databases support batch writes and random reads, and they do not support random writes. Random writes cause most of the complexity in databases,”

Lambda Architecture

- “Implementing this is easy. **Each data record contains a single page view. Those data records are stored in files on HDFS. A function that rolls up page views per URL by hour is implemented as a series of MapReduce job.”**
- **“The function emits key/value pairs, where each key is a [URL, hour] pair and each value is a count of the number of page views. “**
- “**Those key/value pairs are exported into an** ElephantDB database so that an **application can quickly get the value for any [URL, hour] pair.** When an application wants to know the number of page views for a time range, it **queries ElephantDB for the number of page views for each hour in that time range and adds them up to get the final result.**”

Kappa Architecture

- Logs core architecture.
- Facts input, and analytics may be
- Long term storage? hdfs.
- Jay Kreps introduced https://www.oreilly.com/ideas/questioning-the-lambda-architecture because he **wasn't happy with lambda**
    - "discipline of modeling data transformation as a series of materialized stages from an original input has a lot of merit"… This is one of the things that makes large MapReduce workflows tractable"
    - Why lambda: "Reprocessing is one of the key challenges of stream processing but is very often ignored. By "**reprocessing," I mean processing input data over again to re-derive output…. The Lambda Architecture deserves a lot of credit for highlighting this problem**."
    - why not lambda: "**the operational burden of running and debugging two systems is going to be very high." "any new abstraction can only provide the features supported by the intersection of the two systems… walls off the rich ecosystem of tools and languages that makes Hadoop so powerful (Hive, Pig, Crunch, Cascading, Oozie, etc)"**
    - "**Keeping code written in two different systems perfectly in sync was really, really hard**."
    - "The API meant to hide the underlying frameworks proved to be the leakiest of abstractions. "
    - "Why can't you do both real-time processing and also handle the reprocessing when code changes? handle reprocessing by increasing the parallelism and replaying history very, very fast"

Uri Schonfeld
shuri@berkeley.edu

Kappa Architecture

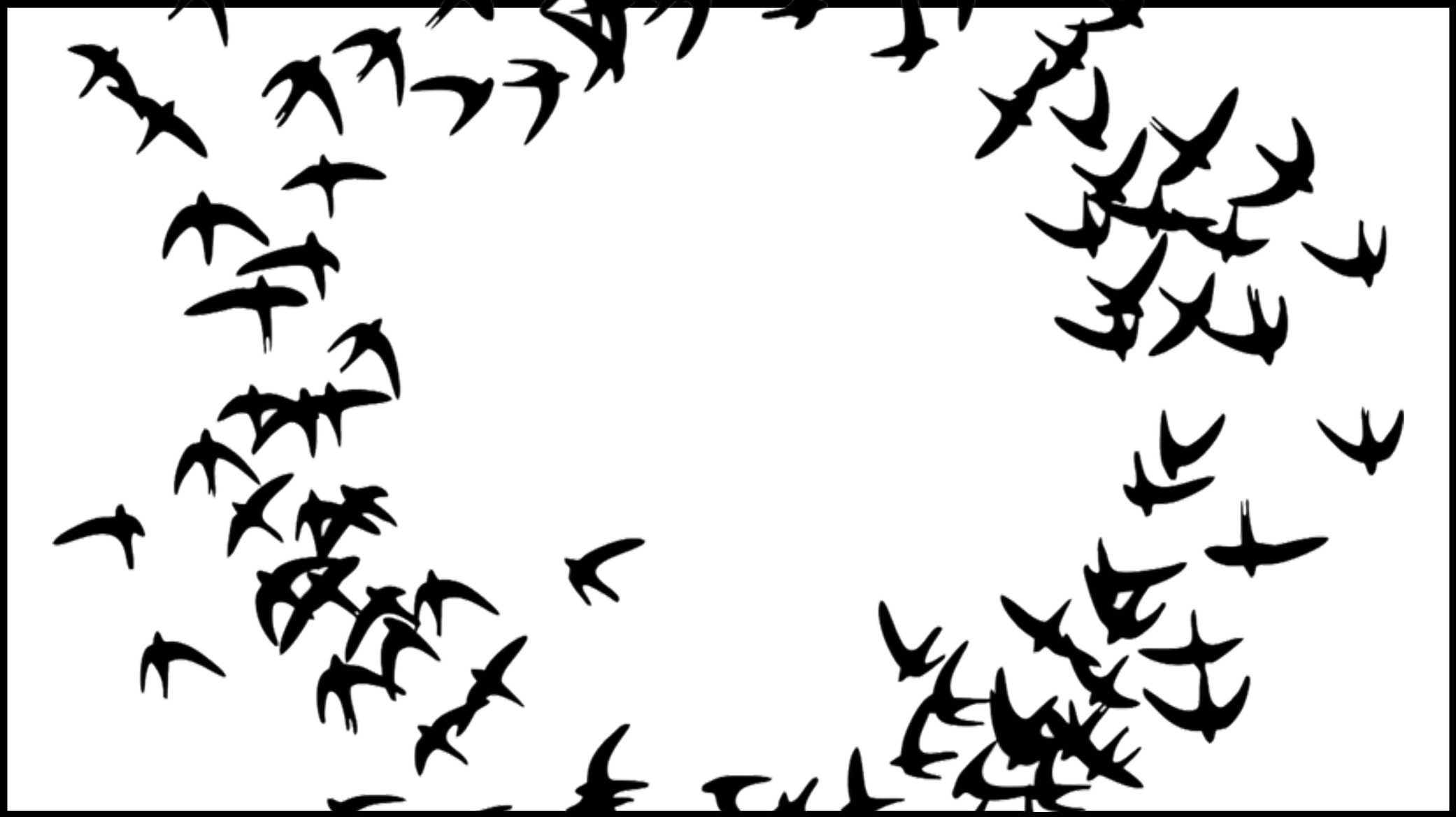
- **Kappa = Lambda - batch + extra stream processing.**
- Kappa Overview:
  - “set your retention in Kafka to 30 days.”
  - “to do the reprocessing, start a second instance of your stream processing job that starts processing from the beginning of the retained data, but direct this output data to a new output table.”
  - “second job has caught up, switch the application to read from the new table.”
  - “Stop the old version of the job, and delete the old output table.”
  - “only do reprocessing when your processing code changes”
  - “bump up the parallelism on your reprocessing job so it completes very quickly”

Uri Schonfeld
shuri@berkeley.edu

Netflix Architecture

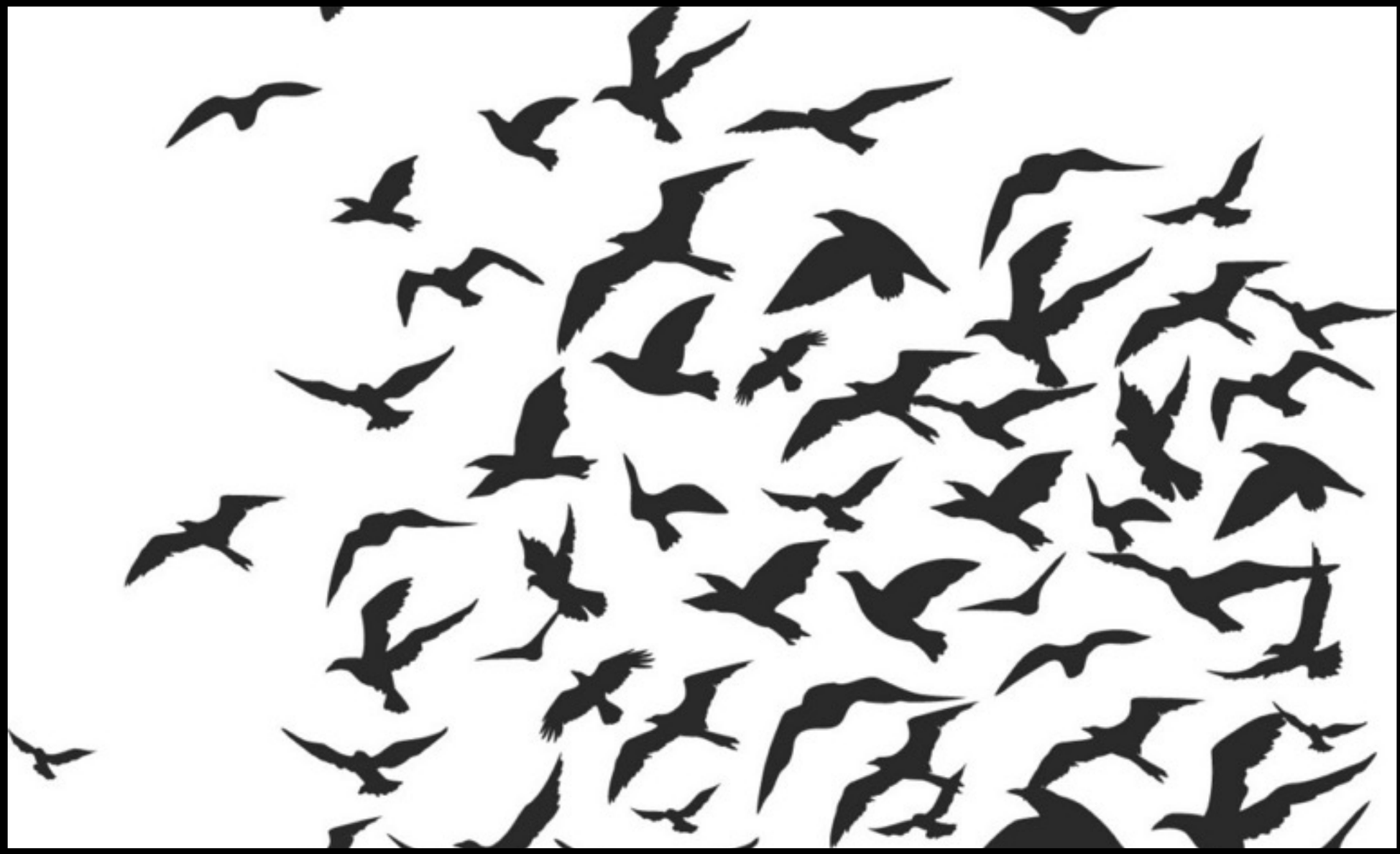
- **Everything in object store.**
- **results? push back to object store.**

Uri Schonfeld
shuri@berkeley.edu



Breakout

Uri Schonfeld
shuri@berkeley.edu



Huddle

Uri Schonfeld
shuri@berkeley.edu

datascience@berkeley

The log: What every software engineer should know about real-time data's unifying abstraction. LinkedIn blog

- local: “a database uses a log to write out information about the records they will be modifying, before applying the changes to all the various data structures it maintains.”

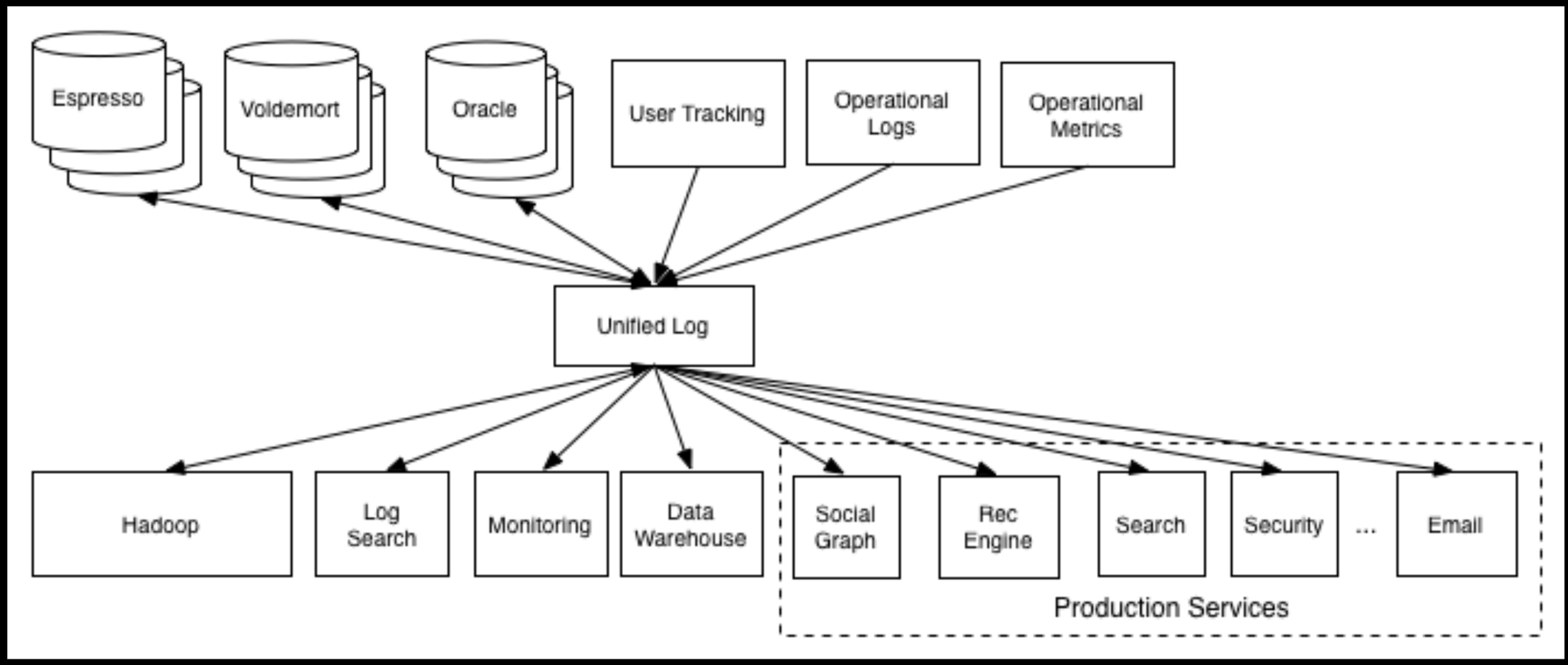
- distributed: “the sequence of changes that happened on the database is exactly what is needed to keep a remote replica database in sync”

- “transmit portions of log to replica databases which act as slaves”

- “you can describe each replica by a single number, the timestamp for the maximum log entry it has processed. “

Uri Schonfeld
shuri@berkeley.edu

The log: What every software engineer should know about real-time data's unifying abstraction. LinkedIn blog



- "building Kafka to combine what we had seen in messaging systems with the log concept popular in databases and distributed system internals."

What we should be able to answer?

- What's a **data lake**? **data warehouse**? difference?
- ETL? ELT? Difference?
- Draw **Lambda** architecture? What does the batch layer do? Why do we need a serving layer? Speed layer?
- Draw **kappa** architecture? What's a topic?
- How is the **Netflix** architecture different? Pro's? Con's?
- Data warehouse?
- What information is stored in each (size? sources? schema? immutable? log?) How is it processed?
- What's OLAP? OLTP? Difference?
- Star Schema? Difference from Snowflake Schema?
- Immutable? Storing history vs storing current state? Log items? Aggregation? Roll-ups? Cuboids?

Uri Schonfeld
shuri@berkeley.edu

What you should be able to answer?

- HDFS: How is a file stored? Why replicate?
- What does fault tolerance mean? How is that different from High availability?
- Key-value store?
- Sharding?
- RDBMs?
    - Column
        - (analytical workloads (read mostly, bulk insertion))
        - More likely to read all values of a column
    - vs Row store?
    - What's an index? Secondary index?
    - Materialized views? Stored procedures?
- Object store semantics? Software-Defined Object Storage? (IS there a difference?)
- Swift architecture?

Uri Schonfeld
shuri@berkeley.edu

What you should be able to answer?

- object store:
  - why? Storing of any object that does not require indexed access.
  - When high-availability, scale and cost are key.
  - When “looser” consistency is acceptable.
  - “unique-as-possible” and minimizing node crash impact?
  - Hash and Hash Ring? (read the chord paper)
  - Lineage? “Data lineage is defined as a data life cycle that includes the data's origins and where it moves over time. It describes what happens to data as it goes through diverse processes. It helps provide visibility into the analytics pipeline and simplifies tracing errors back to their sources.”
  - Provenance? “providing a historical record of the data and its origins. The generated evidence supports essential forensic activities such as data-dependency analysis, error/compromise detection and recovery, and auditing and compliance analysis. Lineage is a simple type of why provenance.”
  - “Cui et al. [27] were among the first to formalize a notion of provenance, of data in the context of relational databases, called lineage.“ from “Provenance in Databases: Why, How, and Where“
  - Governance (who can see, backup. health care)

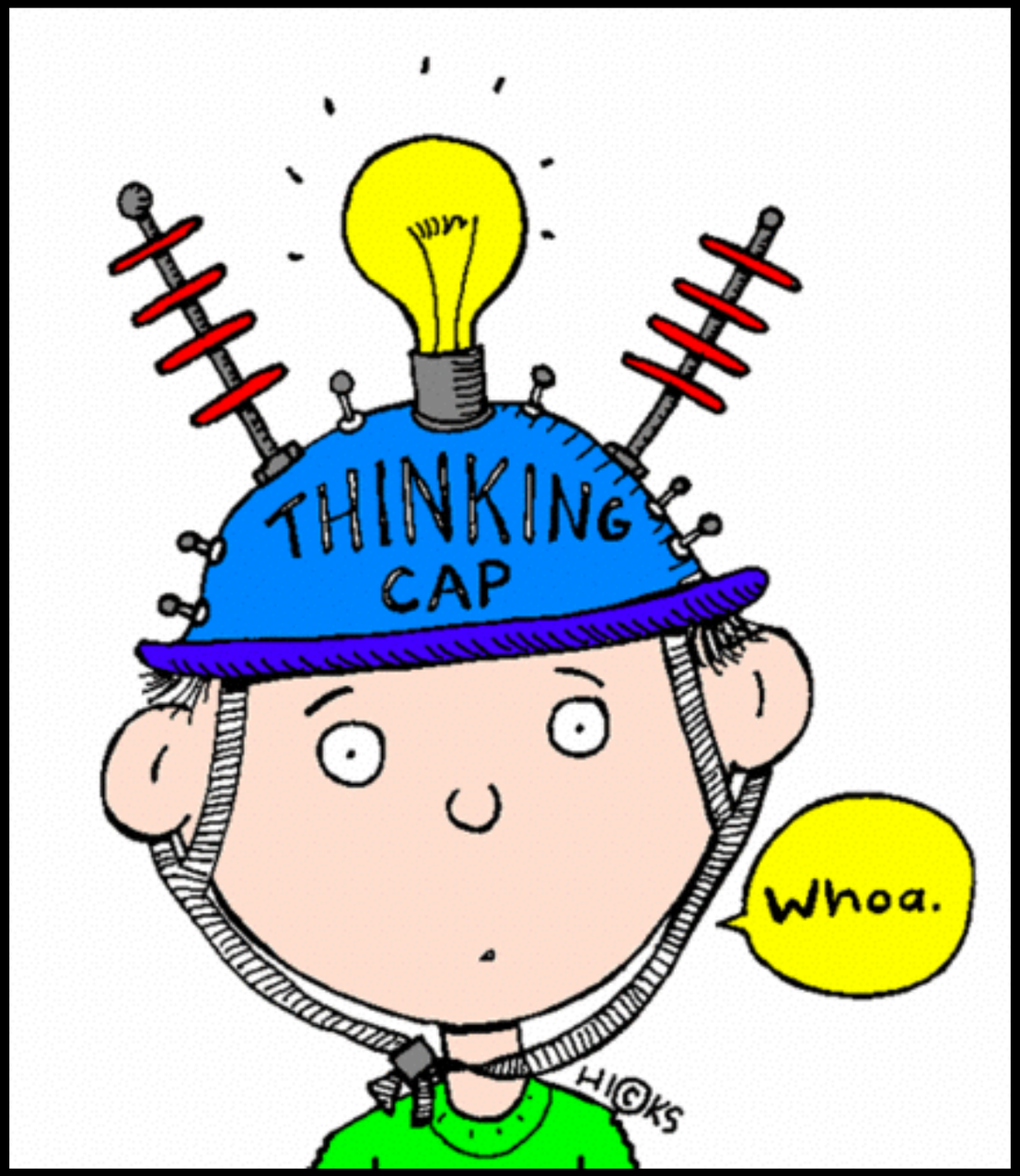
Uri Schonfeld
shuri@berkeley.edu


The end


The end


The end


The end



Quiz Questions?

Uri Schonfeld
shuri@berkeley.edu

Quiz

- 4.12 True or false? An HDFS block can survive k server failures.??? False
- 4.14 Partitions can never move from one disk to another.??? False.

Uri Schonfeld
shuri@berkeley.edu

Hive?

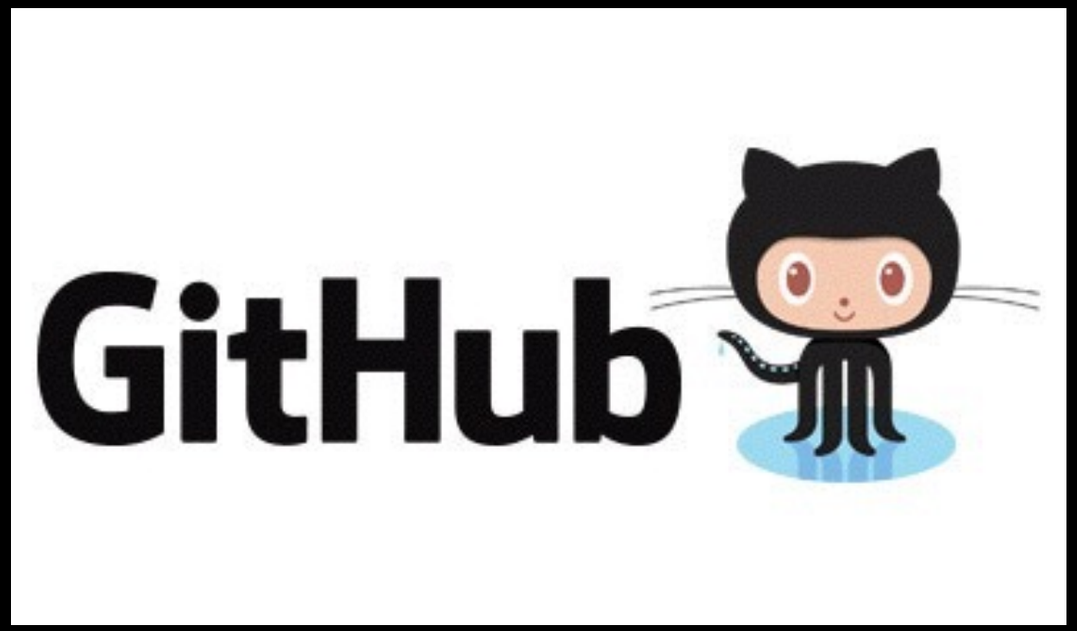
Uri Schonfeld
shuri@berkeley.edu

Hive vs RDBMS

- http://blog.matthewrathbone.com/2015/12/08/hive-vs-mysql.html
- http://hadooptutorial.info/hive-vs-rdbms/
- http://stackoverflow.com/questions/6919089/what-difference-of-rdbms-and-hive

github?

https://guides.github.com/activities/hello-world/



Uri Schonfeld
shuri@berkeley.edu