NoSQL: Column Family

CSCI493.71

Cassandra

CSCI 493.71

What Is It?

- It is a *column-oriented* NoSQL database.
- Its distribution design is based on Amazon's Dynamo
- It adds a more powerful "column family" data model from Google's Bigtable.
- It is created at Facebook, now hosted in Apache.
- It is scalable, fault-tolerant, and consistent.
- It is being used by some of the biggest companies such as Facebook, Twitter, Cisco, Rackspace, ebay, Twitter, Netflix, and more.

Features

- **Elastic scalability** - Highly scalable; it allows to add more hardware to accommodate more customers and more data as per requirement.
- **Always on architecture** - No single point of failure and it is continuously available for business-critical applications that cannot afford a failure.
- **Fast linear-scale performance** - linearly scalable
- **Flexible data storage** - Accommodates all possible data formats including: structured, semi-structured, and unstructured. It can dynamically accommodate changes to your data structures according to your need.
- **Easy data distribution** - Provides the flexibility to distribute data where you need by replicating data across multiple data centers.
- **Transaction support** - Cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (ACID).
- **Fast writes** - Designed to run on cheap commodity hardware. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency.

Architecture

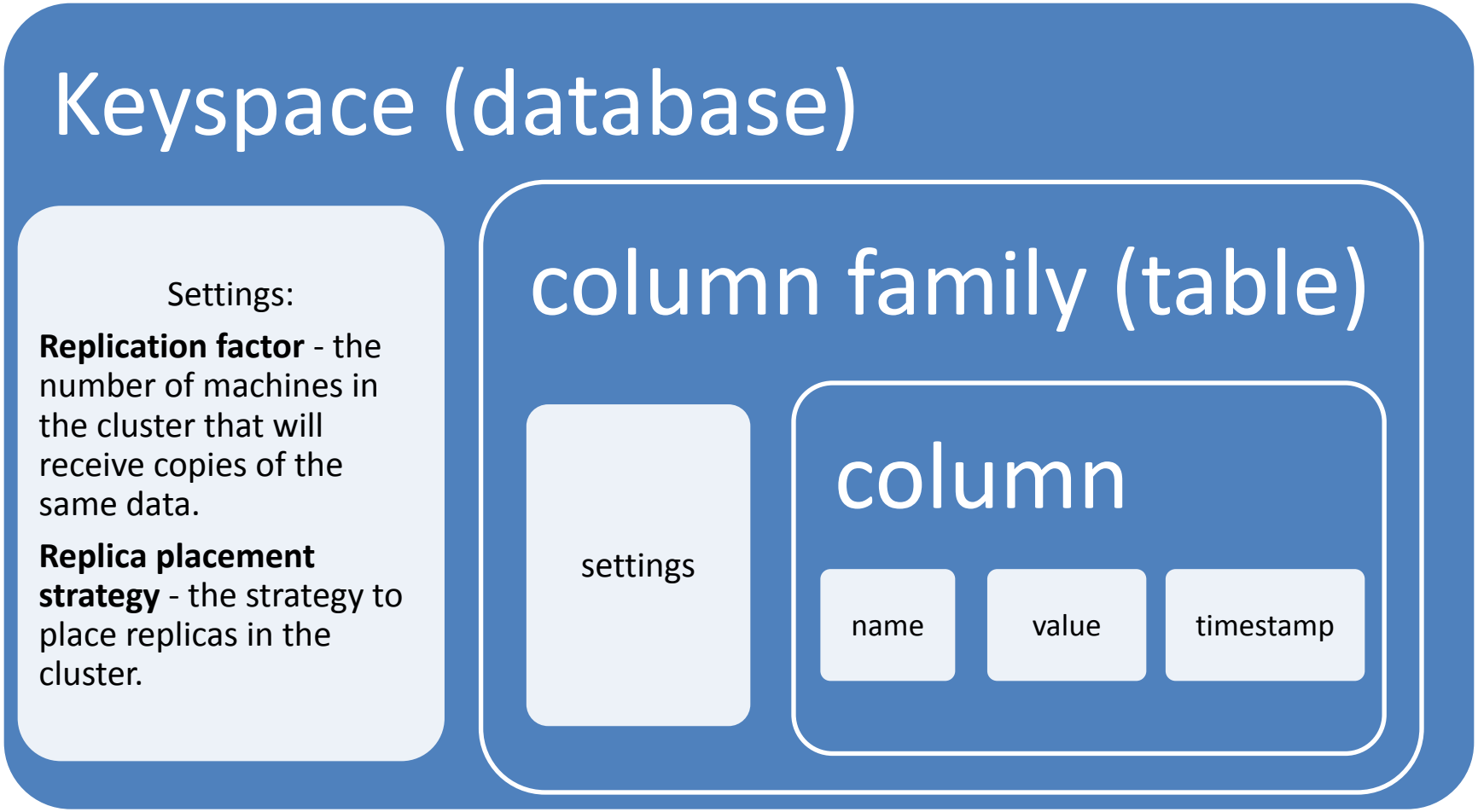
- **Node** - The place where data is stored.
- **Data center** - A collection of related nodes.
- **Cluster** - A component that contains one or more data centers.

- **Commit log** - A crash-recovery mechanism. Every write operation is written to the commit log.
- **Mem-table** - A memory-resident data structure. After commit log, the data will be written to the mem-table.
- **SSTable** - A disk file, to which the data is flushed to, from mem-table, when its contents reach a threshold value.
- **Bloom filter** - A quick, nondeterministic, algorithms for testing whether an element is a member of a set.
- **Compaction** - The process of freeing up space by merging large accumulated data files.
  - During compaction, the data is merged, indexed, sorted, and stored in a new SSTable. Compaction also reduces the number of required seeks.

Data Model

- **Table is a multi dimensional map indexed by key (row key).**
- **Columns are grouped into Column Families.**
- **2 Types of Column Families**
  - **Simple**
  - **Super (nested Column Families)**
- **Each Column has**
  - **Name**
  - **Value**
  - **Timestamp**

Data Model


Keyspace (database)
Settings:
Replication factor - the number of machines in the cluster that will receive copies of the same data.
Replica placement strategy - the strategy to place replicas in the cluster.
column family (table)
settings
column
name
value
timestamp

column family

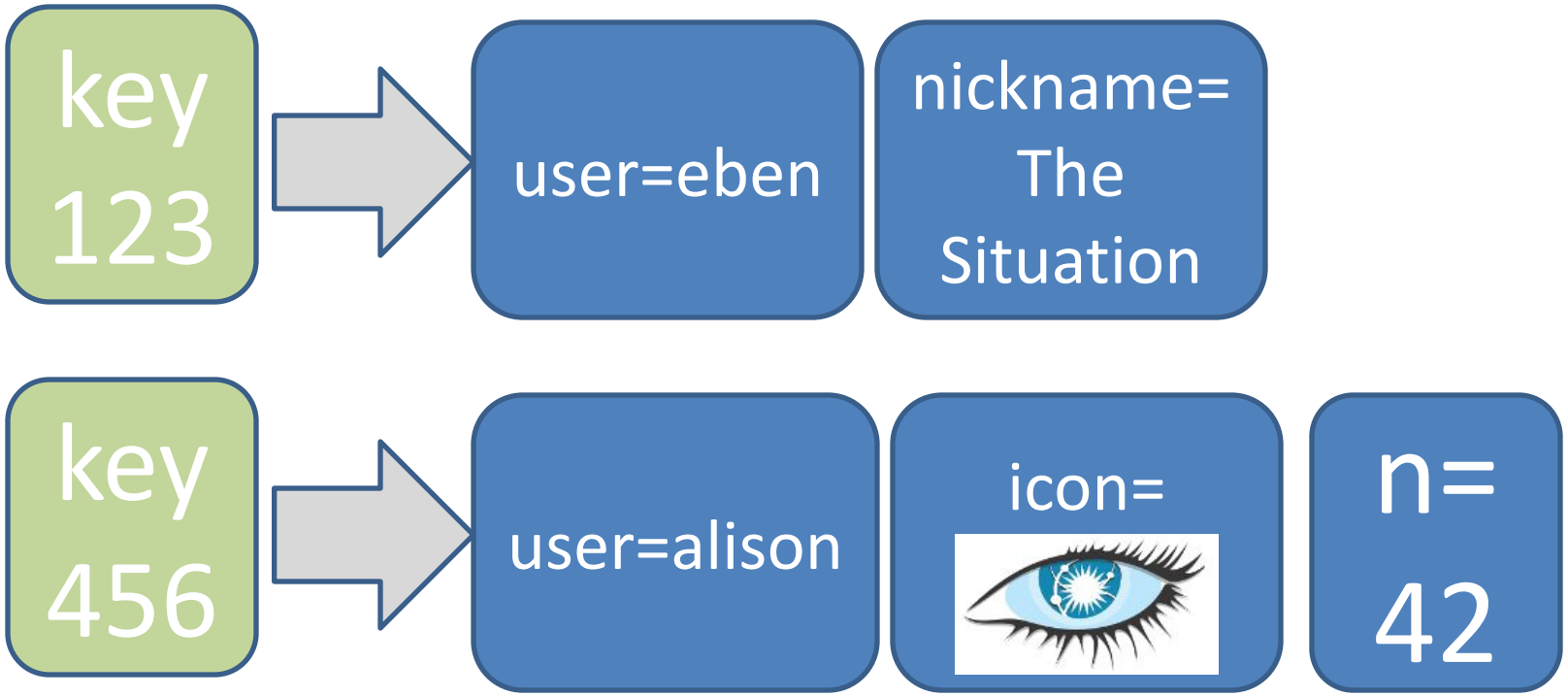
- group records of *similar* kind
- not *same* kind, because CFs are sparse tables
- ex:
  - User
  - Address
  - Tweet
  - PointOfInterest
  - HotelRoom

think of cassandra as

row-oriented

- each row is uniquely identifiable by key
- rows group columns and super columns

column family



json-like notation

User {

123 : { email: alison@foo.com,

icon: 

456 : { email: eben@bar.com,

location: The Danger Zone}

}

a column has 3 parts

1. name
   - byte[]
   - determines sort order
   - used in queries
   - indexed
2. value
   - byte[]
   - *you don't query on column values*
3. timestamp
   - long (clock)
   - last write wins conflict resolution

System Design

- **Partitioning**

  Consistent Hashing

- **Replication**

  How data is duplicated across nodes

- **Cluster Membership**

  How nodes are added, deleted to the cluster

Replication

- **Each data item is replicated at N (replication factor) nodes.**

- **Different Replication Policies**
  - **Rack Unaware –** replicate data at N-1 successive nodes after its coordinator
  - **Rack Aware –** uses ‘Zookeeper’ to choose a leader which tells nodes the range they are replicas for
  - **Datacenter Aware –** similar to Rack Aware but leader is chosen at Datacenter level instead of Rack level.

Gossip Protocols

- **Network Communication protocols inspired for real life rumor spreading.**
- **Periodic, Pairwise, inter-node communication.**
- **Low frequency communication ensures low cost.**
- **Random selection of peers.**
- **Example – Node A wish to search for pattern in data**
  - **Round 1 – Node A searches locally and then gossips with node B.**
  - **Round 2 – Node A,B gossips with C and D.**
  - **Round 3 – Nodes A,B,C and D gossips with 4 other nodes ……**
- **Round by round doubling makes protocol very robust.**

Cluster Management

- **Uses *Scuttlebutt* (an efficient Gossip protocol) to manage nodes.**
  - http://www.cs.cornell.edu/home/rvr/papers/flowgossip.pdf
- **Uses gossip for node membership and to transmit system control state.**
- **Node Fail state is given by variable 'phi' which tells how likely a node might fail (suspicion level) instead of simple binary value (up/down).**
- **This type of system is known as Accrual Failure Detector.**

Accrual Failure Detector

- A server (node A) suspects that a node is down if it hasn't received **heartbeats** from node B.
- If a node is faulty, the suspicion level monotonically increases with time.

  $\Phi(t) \rightarrow k$ as $t \rightarrow k$

  Where k is a threshold variable (depends on system load) which tells a node is dead.
- If node is correct, phi will be constant set by application. Generally

  $\Phi(t) = 0$

Scaling

- **Two ways to add new node**
  - **New node gets assigned a random token which gives its position in the ring. It gossips its location to rest of the ring**
  - **New node reads its config file to contact it initial contact points.**
- **New nodes are added manually by administrator via CLI or Web interface provided by Cassandra.**

- **Scaling in Cassandra is designed to be easy.**
- **Lightly loaded nodes can move in the ring to alleviate heavily loaded nodes.**

Local Persistence

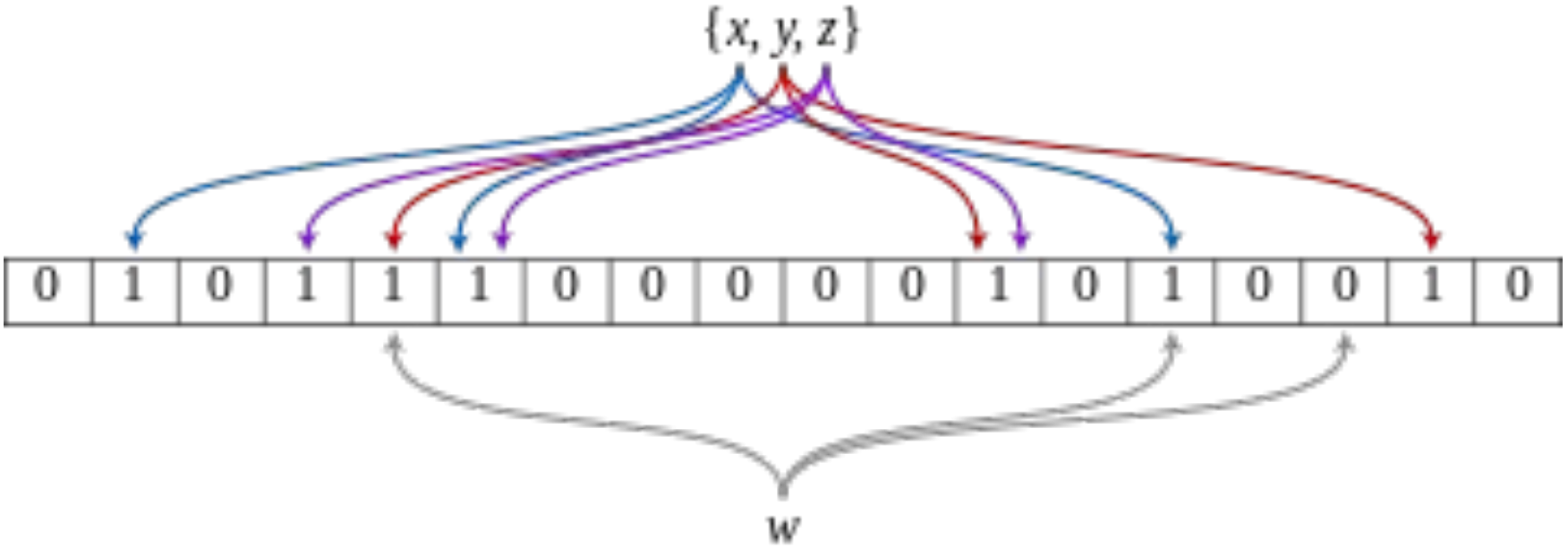
- **Relies on local file system for data persistency.**
- **Write operations happens in 2 steps**
  - **Write to commit log in local disk of the node**
  - **Update in-memory data structure.**
- **Read operation**
  - **Looks up in-memory ds first before looking up files on disk.**
  - **Uses Bloom Filter (summarization of keys in file store in memory) to avoid looking up files that *do not* contain the key.**

Bloom Filter

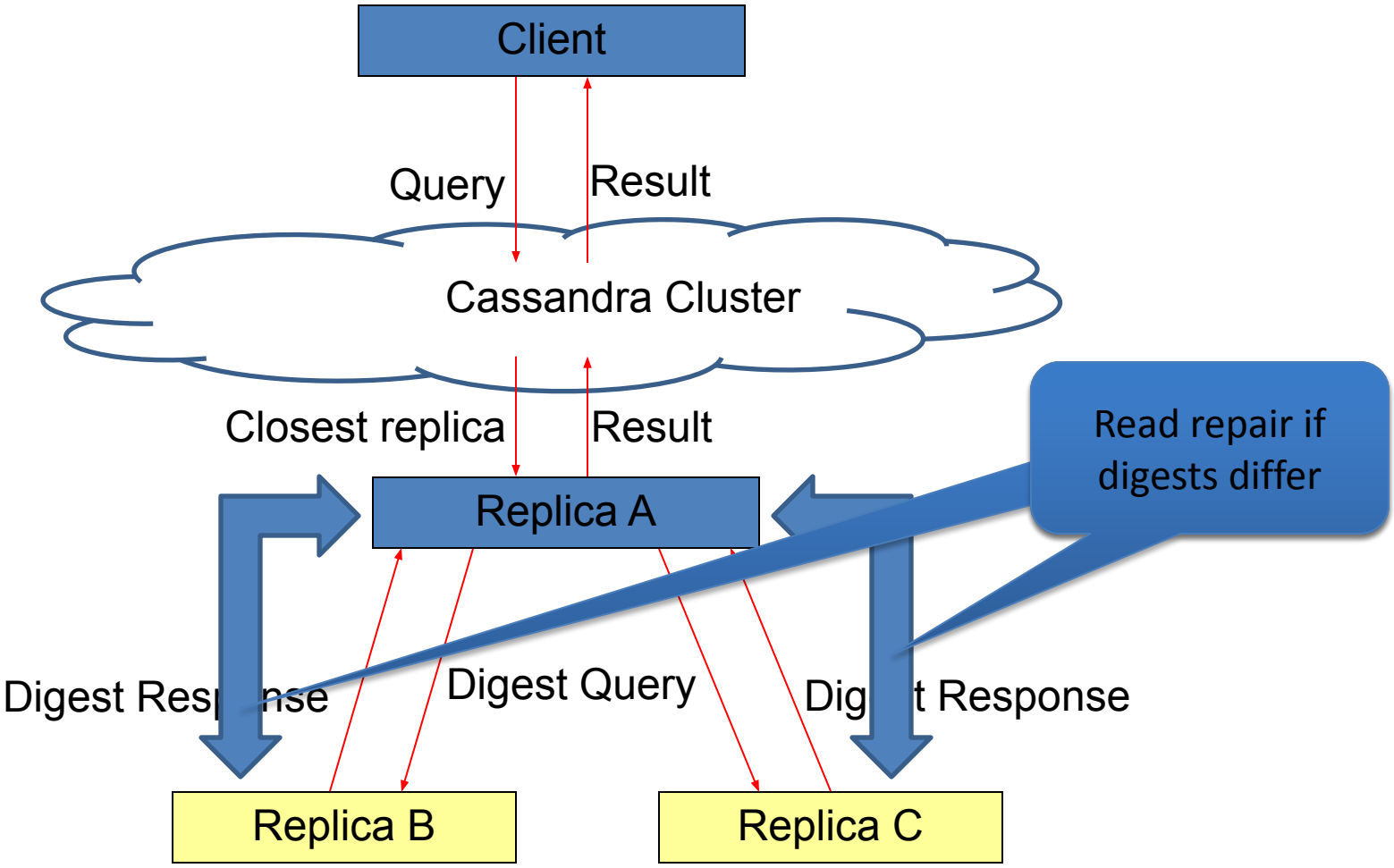
- A space-efficient probabilistic data structure
- Conceived by Burton Howard Bloom in 1970
- Used to test whether an element is a member of a set.
- False positive matches are possible, but false negatives are not, thus a Bloom filter has a 100% recall rate.
- A query returns either "possibly in set" or "definitely not in set".

https://www.youtube.com/watch?v=kfFacplFY4Y

Bloom Filter



Read Operation



Getting Started

- **Prerequisites**
  - Install Java 8 or Java 11
- **Download**
  - http://cassandra.apache.org/download/
- **Configuration**
  - In conf/cassandra.yaml
- **Getting started**
  - http://cassandra.apache.org/doc/latest/getting_started/index.html

cqlsh

- an interactive command line interface for Cassandra.
- cqlsh allows you to execute *CQL* (Cassandra Query Language) statements against Cassandra.
- Invoke cqlsh

```
$ bin/cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.1.2 | CQL spec 3.2.0 | Native protocol v3]
Use HELP for help.
cqlsh>
cqlsh>EXIT
```

Working on Keyspace

- A keyspace in Cassandra is a namespace that defines data replication on nodes.
- A cluster contains one keyspace per node.
- Create keyspace syntax:

  CREATE KEYSPACE <identifier> WITH <properties>
- Example

  >CREATE KEYSPACE csci493.71

  WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 3};
- Use keyspace syntax:

  USE <keyspace>;
- Drop keyspace syntax:

  DROP KEYSPACE <identifier>

Create Column Family

- Defining a primary key is mandatory
- Example:

```
CREATE TABLE tablename (
  column1 name datatype PRIMARYKEY,
column2 name datatype,
  column3 name datatype
);
```

Modify Column Family

- Syntax

  ALTER TABLE|COLUMNFAMILY <table_name> *instruction*

  Where instruction is

  ALTER column_name TYPE cql_type

  | **(** ADD column_name cql_type **)**

  | **(** DROP column_name **)**

  | **(** RENAME column_name TO column_name **)**

  | **(** WITH property *AND property ...* **)**

- Example

  ALTER TABLE users ALTER bio TYPE text;

  ALTER TABLE users ADD top_places list<text>;

  ALTER TABLE addamsFamily DROP gender;

Create Data

- Syntax

  INSERT INTO <tablename>

  (<column1 name>, <column2 name>....)

  VALUES (<value1>, <value2>....)

  USING <option>

- Example:

  cqlsh> INSERT INTO emp

  (emp_id, emp_name, emp_city, emp_phone, emp_sal)

  VALUES(1,'ram', 'Hyderabad', 9848022338, 50000);

Update Data

- Syntax

  UPDATE <tablename>

  SET <colname> = <new value>, <colname> = <value>....

  WHERE <condition>

- Example

  cqlsh> UPDATE emp SET emp_city='Delhi',emp_sal=50000
  WHERE emp_id=2;

Read Data

- Syntax

  SELECT *|column-list FROM <table name>

  WHERE <condition>;

- Example

  cqlsh> SELECT * FROM emp

  WHERE emp_sal=50000;

Delete Data

- Deleting entire row

  DELETE FROM <tablename>

  WHERE <condition>;

- Deleting data syntax

  DELETE <colname> FROM <tablename>

  WHERE <condition>;

Collections

- Three types: Set, List, and Map
- Example
  - List

    cqlsh>ALTER TABLE users ADD top_places list<text>;

    cqlsh>UPDATE users SET top_places = ['rivendell', 'rohan']
    WHERE user_id = 'frodo';
  - Map

    cqlsh>ALTER TABLE users ADD todo map<timestamp, text>

    cqlsh>UPDATE users

    SET todo = { '2015-3-24' : ‘project', '2015-4-3' : ‘spring break' }

    WHERE user_id = 'frodo';

Python Driver

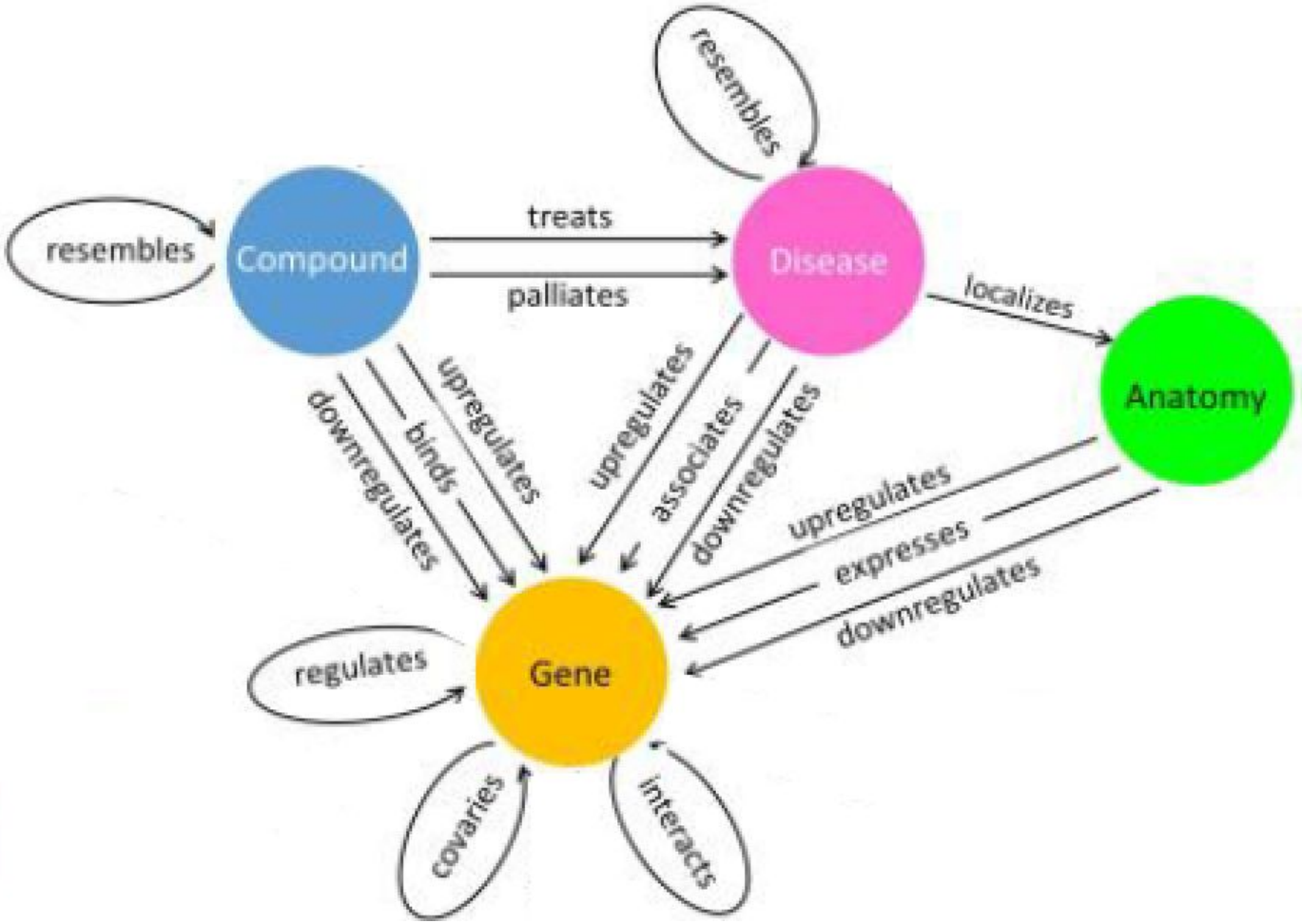
- https://docs.datastax.com/en/developer/python-driver/

Summary

- Good user cases
  - Event logging
  - Content management
  - …
- When not to use
  - Aggregation
  - Query pattern is uncertain (query change is more expensive than schema change)

Project I: User Case

- HetioNet



Project I: User Case

- HetioNet
- Nodes

nodes.tsv

Id	name	kind
Gene::9997	SC02	Gene
Compound::DB09028	Cytisine	Compound

- Relationships

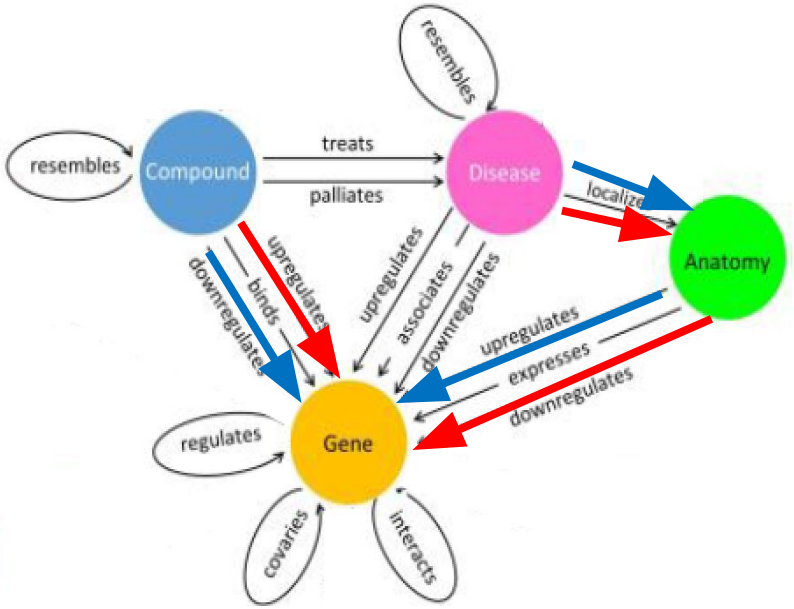
edges.tsv

source	metaedge	target
Gene::801	GiG	Gene::7428
Disease::DOID:263	DuG	Gene::857

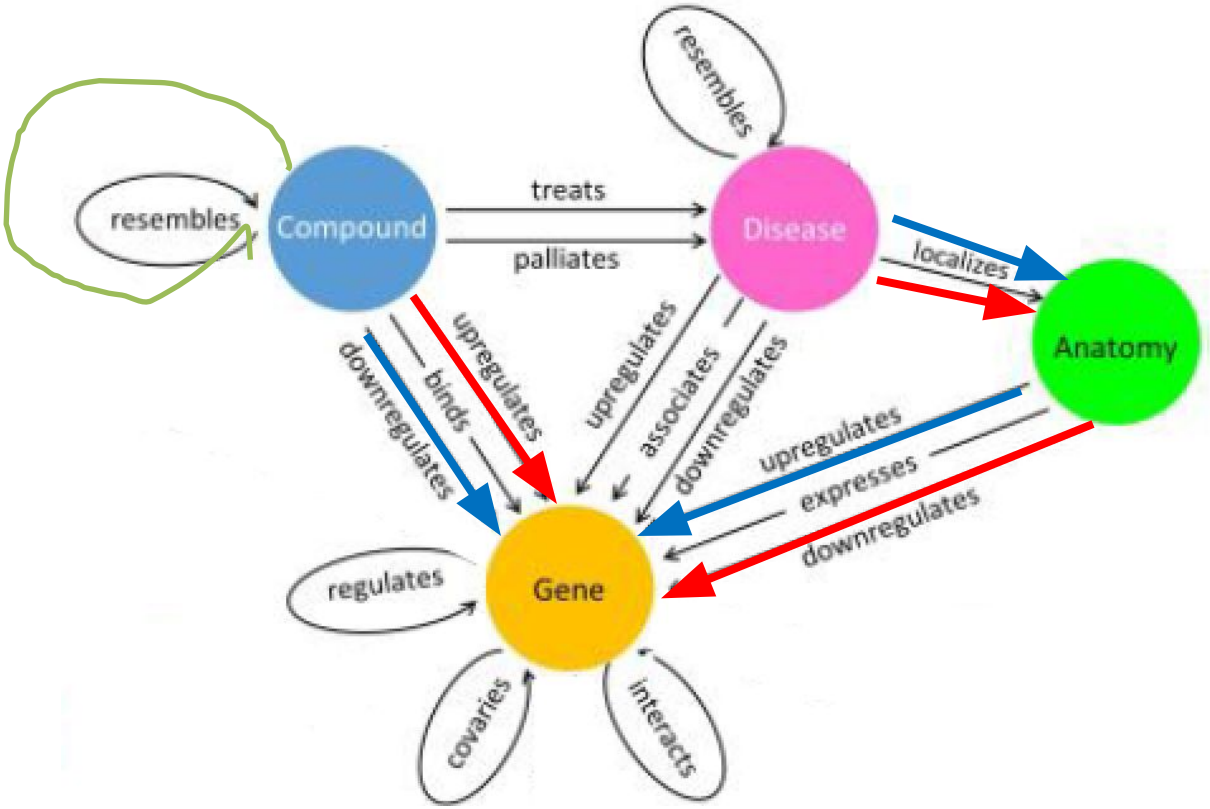
Project I: Requirement

- Build a database system to model *HetioNet*
- The database should at least answer the following questions in the quickest response time

  1. Given a disease id, what is its name, what are drug names that can treat or palliate this disease, what are gene names that cause this disease, and where this disease occurs? Obtain and output this information in a single query.



2. We assume that a compound can treat a disease if the compound up-regulates/down-regulates a gene, but the location down-regulates/up-regulates the gene in an opposite direction where the disease occurs. Find all compounds that can treat a new disease (i.e. the missing edges between compound and disease excluding existing drugs). Obtain and output all drugs in a single query.



Project I: Requirement

- A Python command-line client interface for database creation and query
- Use at least two types of NoSQL stores (Document, Graph, Key-value, Column Family)

Project I: Requirement

- Document (no hand-writing, in print!)
  - Design diagram
  - All queries
  - Potential improvements (e.g. how to speed up query)
- All source codes (upload to brightspace)
- Two-person team
- Due: 5 pm, March 17, Monday
- Project demo: 5:30 pm-8:15 pm, March 17 (Monday)

Project I: Rubric

- Database design: 30%
  - Rationale (15%)
  - Implementation (15%)
- Query functionality: 40%
  - 20% each query
- Client interface: 20% (GUI: 10 points)
- Presentation: 10%

Next

- Quiz on Cassandra