Cloud Databases

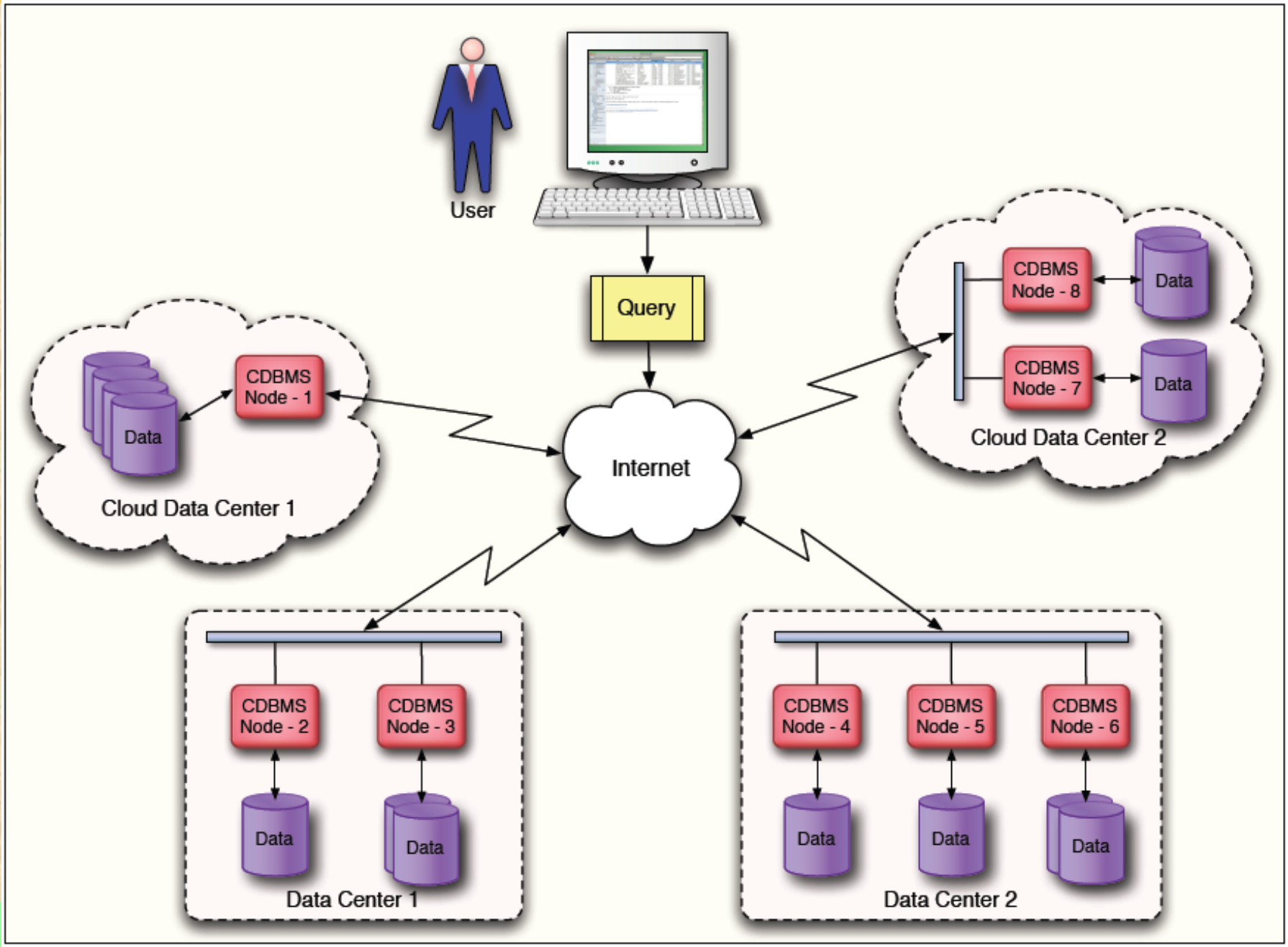
Introduction

- Std. Layered Architecture
- **Cloud Infrastructure**
- A **cloud database** is a database that typically runs on a *cloud computing* platform, such as *Amazon EC2, GoGrid*, *Salesforce*, *Rackspace*, and *Microsoft Azure*.
- Deployment models
  - users can run databases on the cloud independently, using a *virtual machine* image
  - they can purchase access to a database service, maintained by a cloud database provider. ***DBaaS***
- Data Models : **SQL , NoSQL**

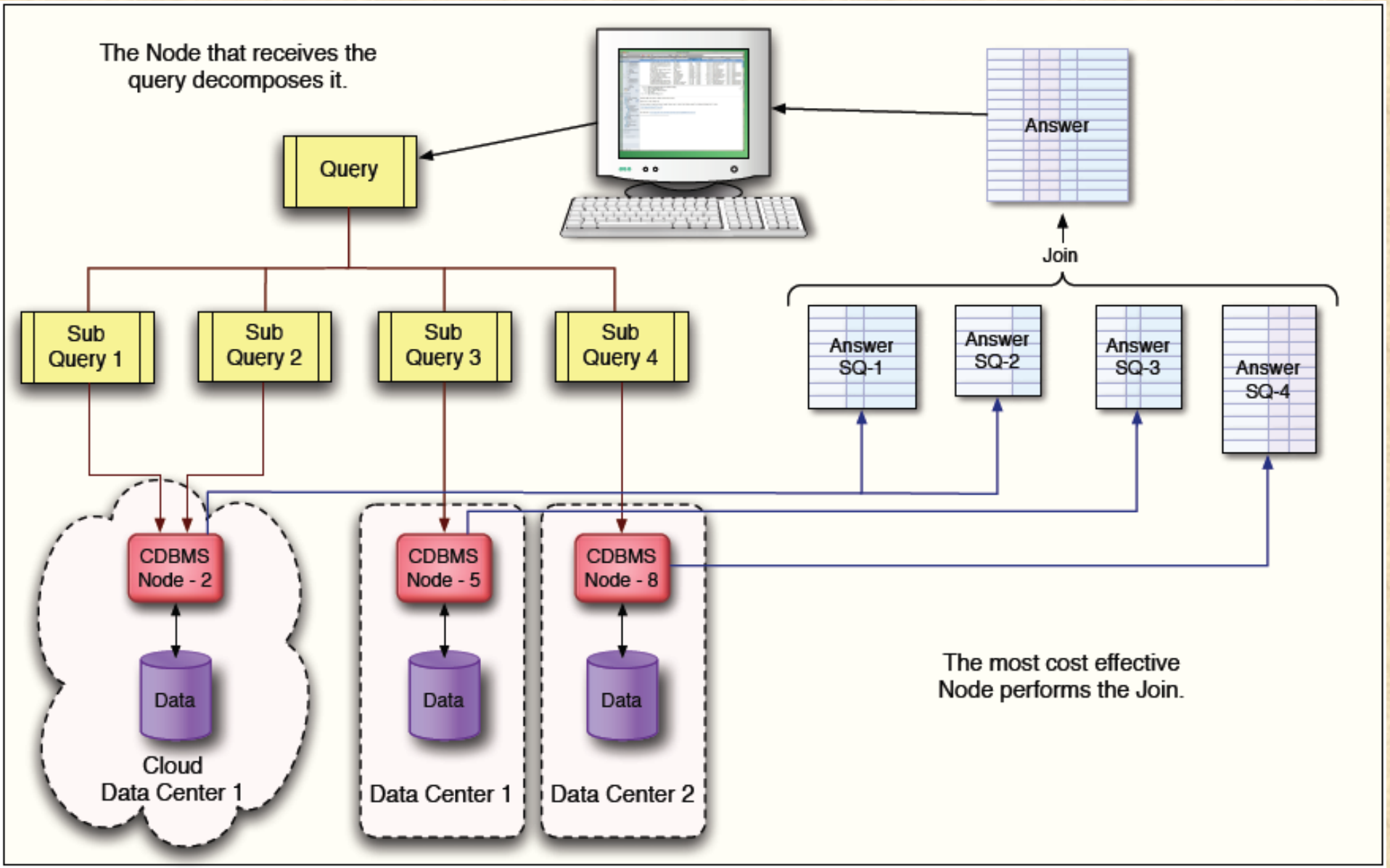
Definition

Cloud dbms (CDBMS) is a distributed database that delivers a query service across multiple distributed database nodes located in multiple geographically-distributed data centers, both corporate data centers and cloud data centers.

Architecture / Layout



Distributed Query Processing



Data Model : SQL

- **SQL database**, such as *NuoDB, Oracle Database, Microsoft SQL Server*, and *MySQL*, are one type of database which can be run on the cloud (either as a Virtual Machine Image or as a service, depending on the vendor).
- SQL databases are difficult to scale, not natively suited to a cloud environment
- Cloud database services based on SQL are attempting to address this challenge

Data Model : NoSQL *www.nosql-database.org*

- NoSQL means '**N**ot **O**nly **SQL**' , 'Not Relational'.
- **NoSQL databases**, such as *Apache Cassandra, CouchDB and MongoDB*, are another type of database which can run on the cloud.
- NoSQL databases are built to service heavy read/write loads and are able scale up and down easily
- More natively suited to running on the cloud.
- working with NoSQL databases often requires a complete rewrite of application code
- Set of APIs to access data. ***no SQL like query***

NoSQL : Advantages

- non-relational
- don’t require schema
- data are replicated to multiple nodes (so, identical & fault-tolerant) and can be partitioned:
  - down nodes easily replaced
  - no single point of failure
- horizontal scalable
- cheap, easy to implement (open-source)
- massive write performance
- fast key-value access

NoSQL : Disadvantages

- Don’t fully support relational features
  - no join, group by, order by operations (except within partitions)
  - no referential integrity constraints across partitions
- No declarative query language (e.g., SQL) → more programming
- Relaxed ACID (see CAP theorem) → fewer guarantees
- No easy integration with other applications that support SQL

NOSQL Modeling Types

1. Key-value
   - Example: DynamoDB, Voldermort, Scalaris
2. **Document-based**
   - Example: MongoDB, CouchDB
3. **Column-based**
   - Example: BigTable, Cassandra, Hbased
4. ***Graph-based***
   - Example: Neo4J, InfoGrid

- "No-schema" is a common characteristics of most NOSQL storage systems
- Provide "flexible" data types

NoSQL Transactions

Types of consistency:

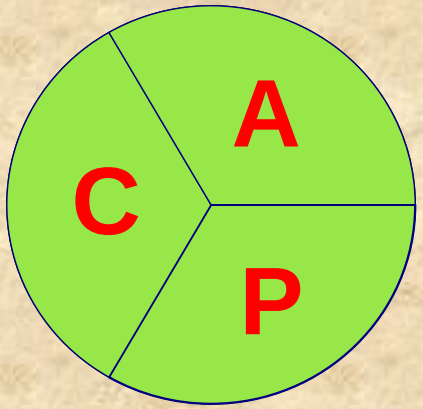
1. Strong consistency – ACID
   (**A**tomicity, **C**onsistency, **I**solation, **D**urability)
   do not supported by NoSQL

2. Weak consistency – BASE
   (**B**asically **A**vailable **S**oft- state **E**ventual consistency)

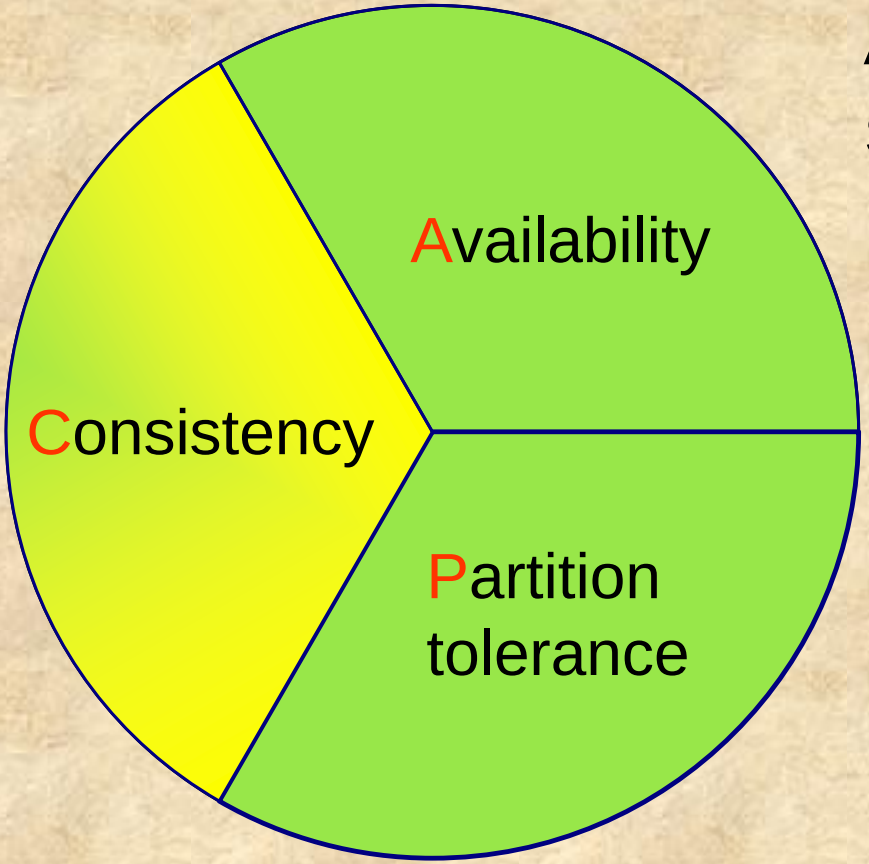
Based on CAP Theorem

CAP Theorem

- Three properties of a distributed system (sharing data)
  - **Consistency:**
    - all copies have same value
  - **Availability:**
    - reads and writes always succeed
  - **Partition-tolerance:**
    - system properties (consistency and/or availability) hold even when network failures prevent some machines from communicating with others



CAP Theorem

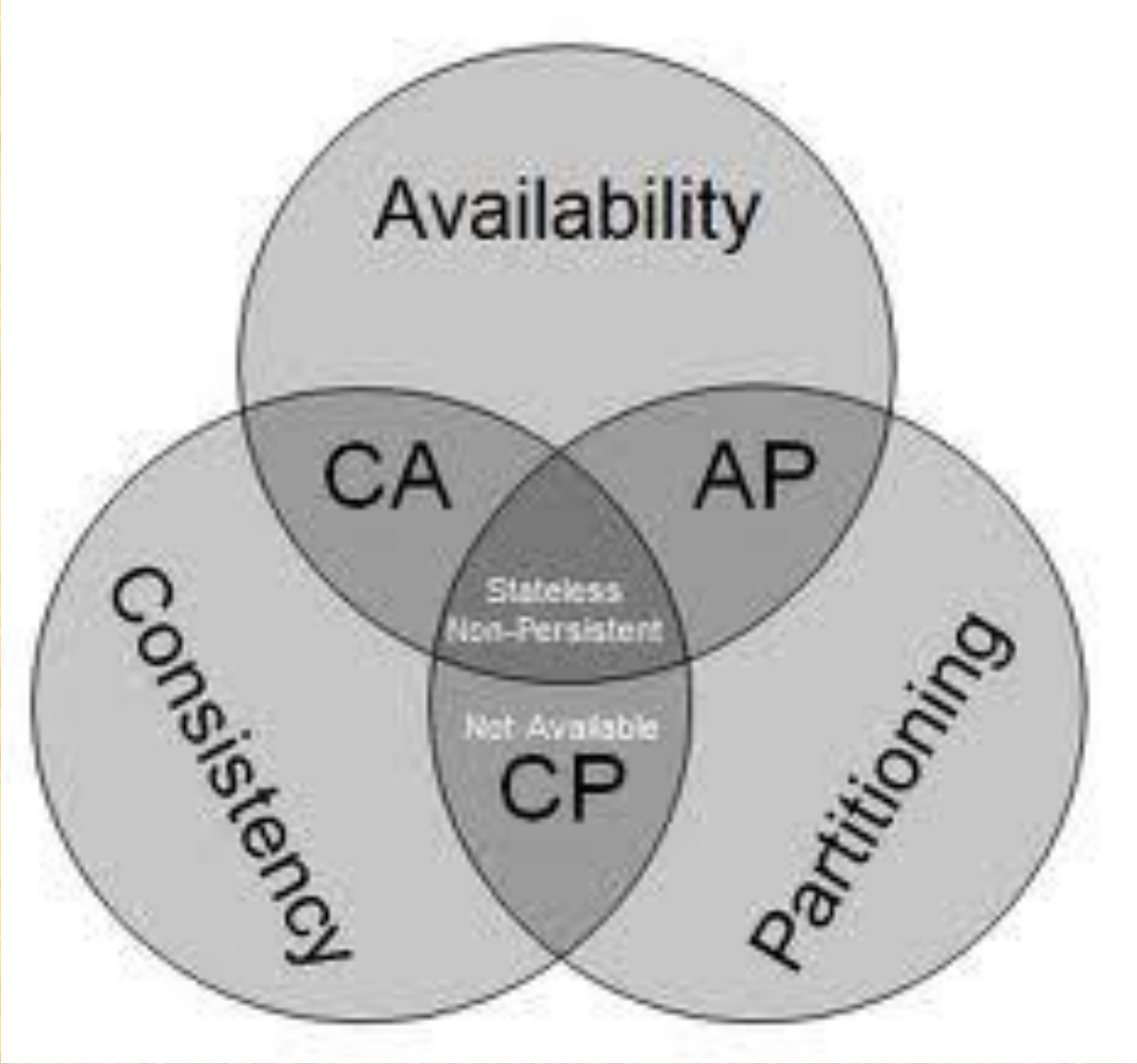


All client always have the same view of the data

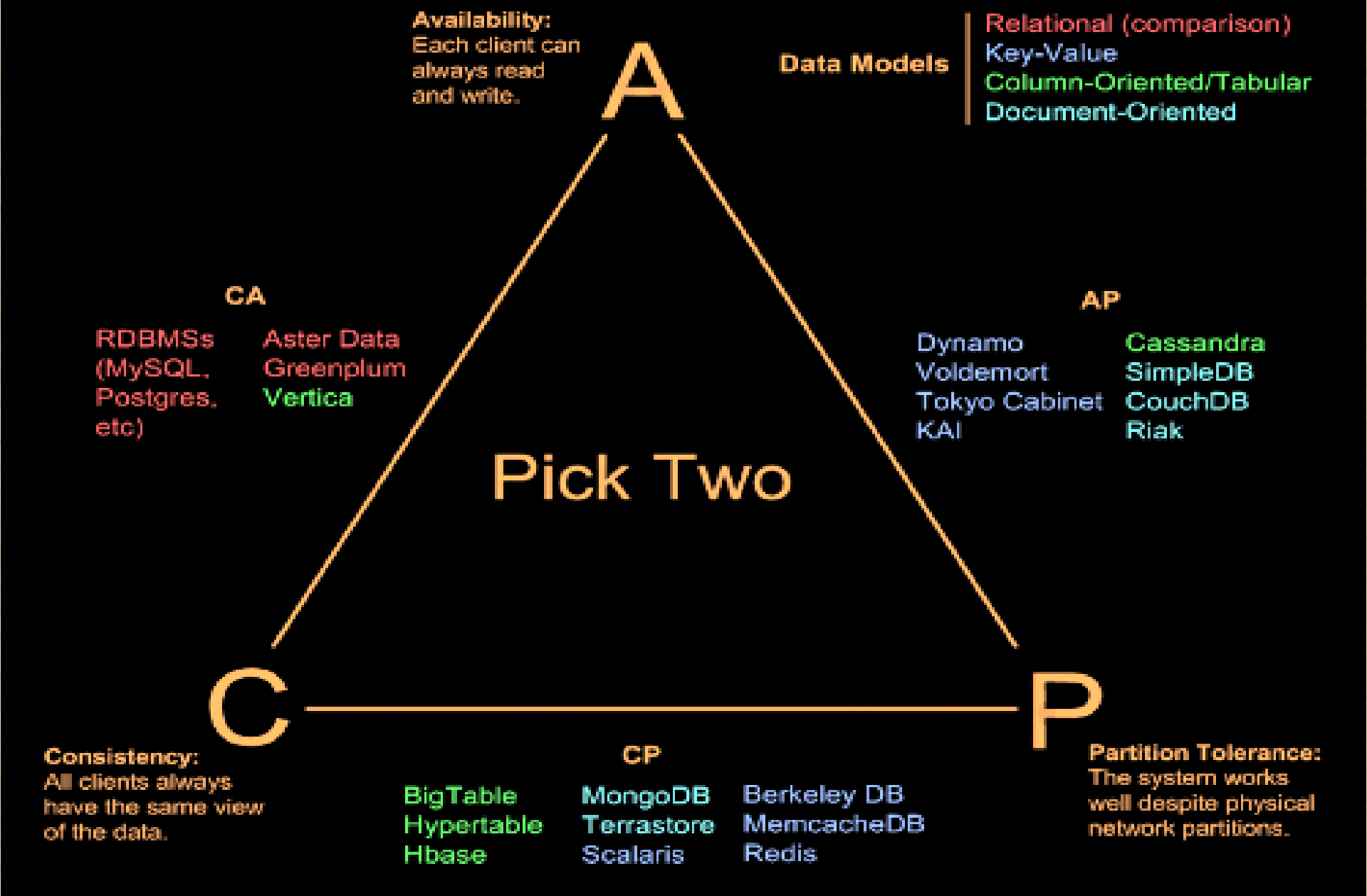
Brewer's CAP Theorem:

- *For any system sharing data, it is "impossible" to guarantee simultaneously all of these three properties*
- You can have at most two of these three properties for any shared-data system
- Very large systems will "partition" at some point:
  - That leaves either **C** or **A** to choose from (traditional DBMS prefers **C** over **A** and **P**)
  - In almost all cases, you would choose **A** over **C** (except in specific applications such as order processing)

Brewer's CAP Theorem



Visual Guide to NoSQL Systems



Storage Architecture for Cloud DB

Shared-nothing Storage Architecture

- It involves data partitioning which splits the data into independent sets - physically located on different database servers.
- suitable for Cloud.
- Needs piece of middleware to route database requests to the appropriate server.
- IBM, Oracle, Amazon's SimpleDB, Hadoop Distributed File System and Yahoo's PNUTS also implement shared-nothing architecture

Storage Architecture for Cloud DB

Shared-disk Database Architecture

- Treats the whole database as a single large piece of database stored on a Storage Area Network (SAN) or Network Attached Storage (NAS) storage that is shared and accessible through network by all nodes.
- Middleware is not required to route data requests to specific servers as each node/client has access to all of the data.
- Oracle RAC, IBM DB2 pureScale, Sybase etc. support this architecture

Apache Cassandra

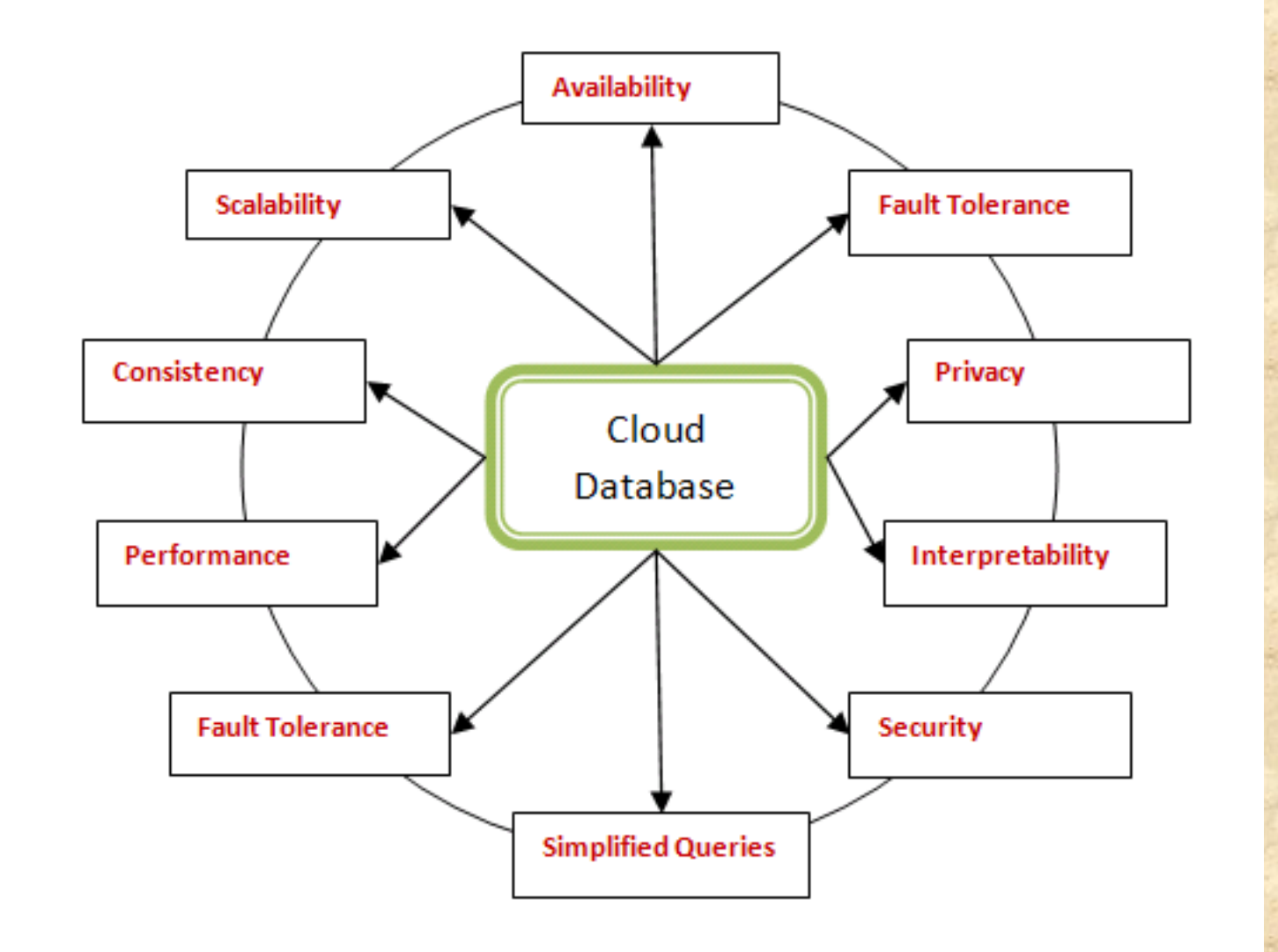
CouchDB

MongoDB

Comparison of RDBMS and NoSQL databases

RDBMS	NoSQL Databases
• Data within a database is treated as a "whole"	• Each entity is considered an independent unit of data and can be freely moved from one machine to the other
• RDBMS support centrally managed architecture.	• They follow distributed architecture.
• They are statically provisioned.	• They are dynamically provisioned.
• It is difficult to scale them.	• They are easily scalable.
• They provide SQL to query data	• They use API to query data (not feature rich as SQL).
• ACID (Atomicity, Consistency, Isolation and Durability) Compliant; DBMS maintains Consistency.	• Follow BASE (Basically Available, Soft state, Eventually consistent); The user accesses are guaranteed only at a single-key level.
• They support on-line Transaction Processing applications.	• They support web2.0 applications.

Challenges to Develop Cloud Databases



Apache Cassandra

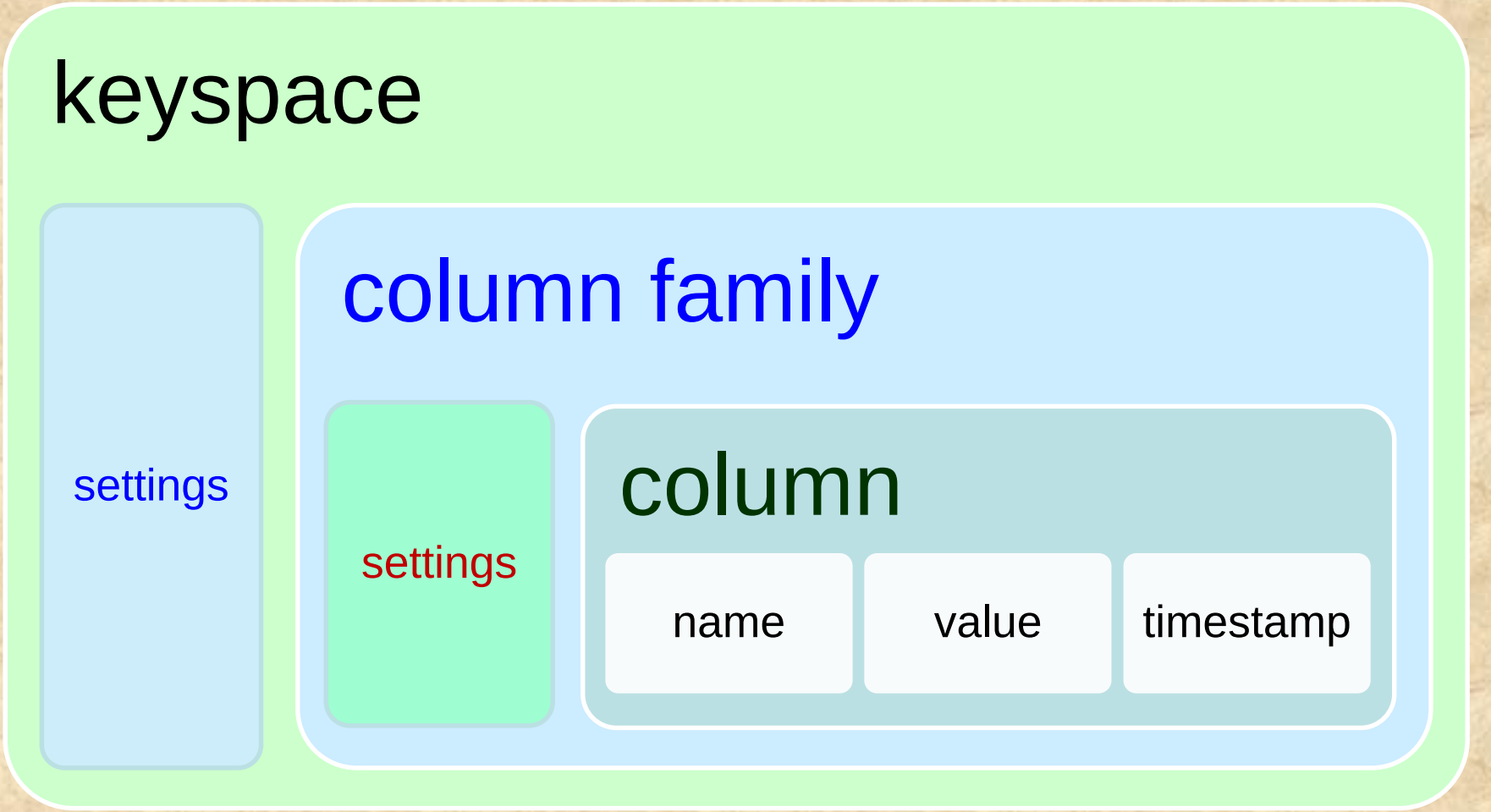
Introduction

- Apache Cassandra is a free, open source, distributed data storage system that differs sharply from relational database management systems.
- Cassandra was created to power the Facebook Inbox Search
- Facebook open-sourced Cassandra in 2008 and became an Apache Incubator project
- In 2010, Cassandra graduated to a top-level project, regular update and releases followed.
- Designed to **handle large amount of data** across multiple servers
- Easy to **implement** and **deploy**
- Mimics traditional relational database systems, but with **triggers** and **lightweight transactions**
- Raw, simple data structures
- *Cassandra is being used by some of the biggest companies such as Facebook, Twitter, Cisco, Rackspace, ebay, Netflix, and more*

Data Model : *Key-Value* Model

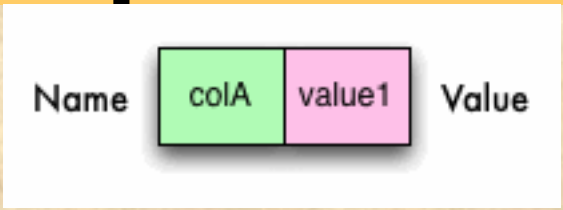
- Cassandra is a column oriented NoSQL system
- Table is a multi dimensional map indexed by key (row key).
- Column families: sets of key-value pairs
  - column family as a table and key-value pairs as a row (using relational database analogy)
- A row is a collection of columns labeled with a name, value, timestamp

Key-Value Model

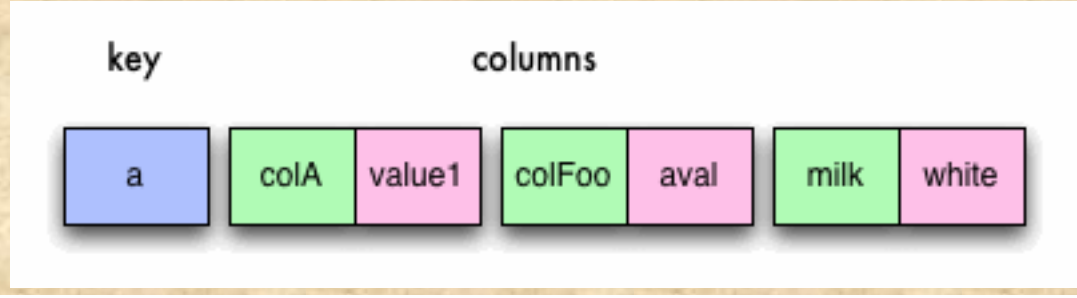


Example

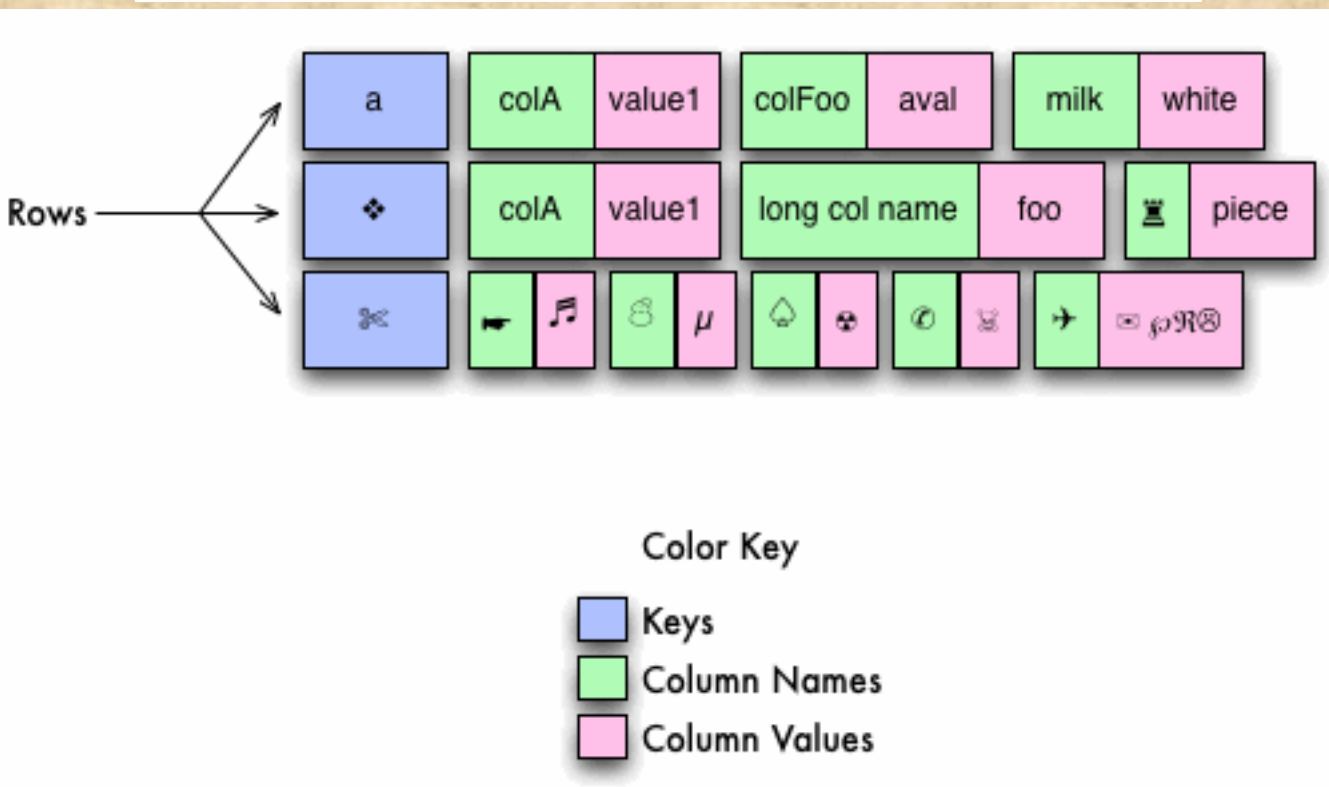
A single column



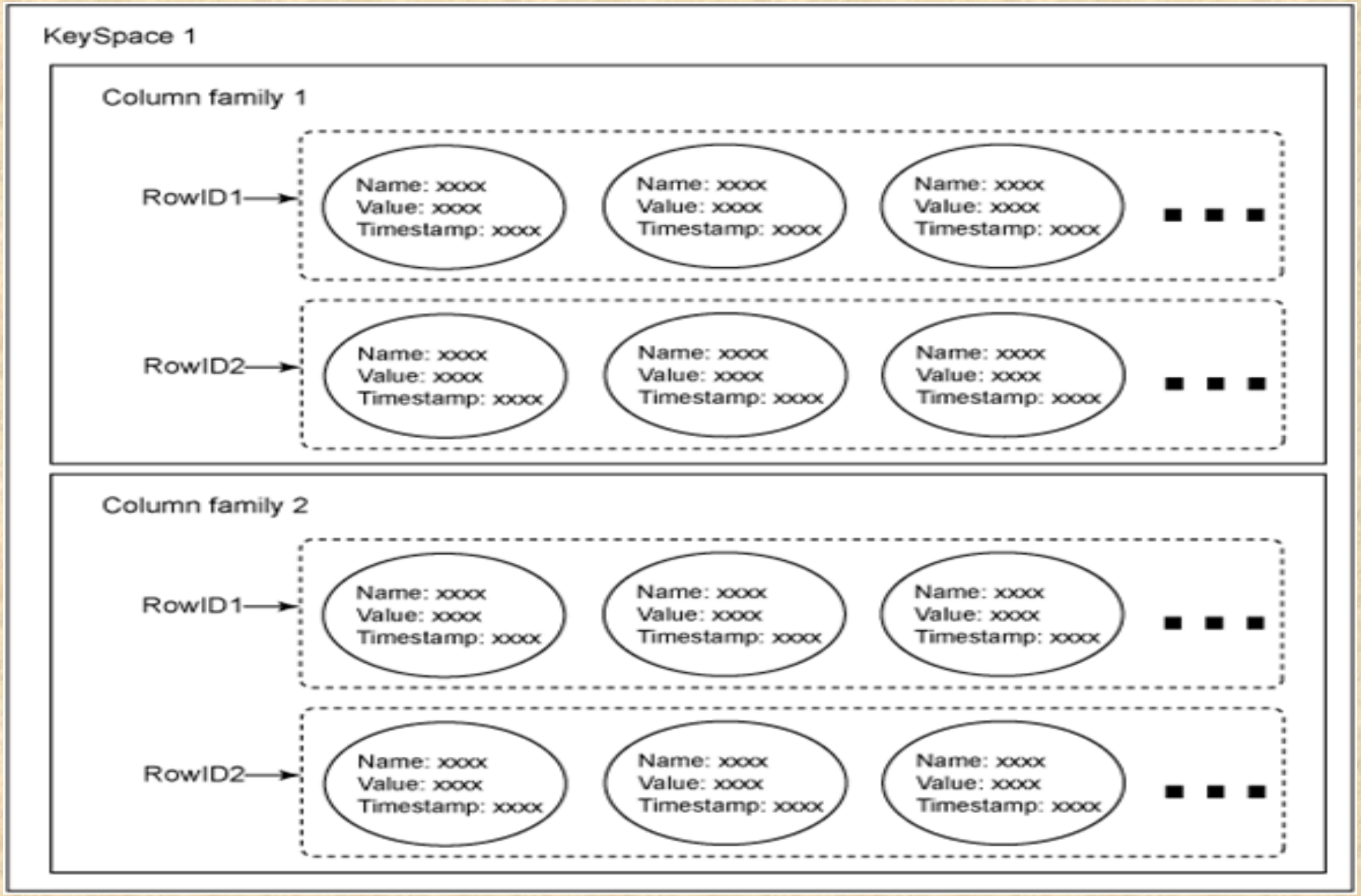
A single row



Column family

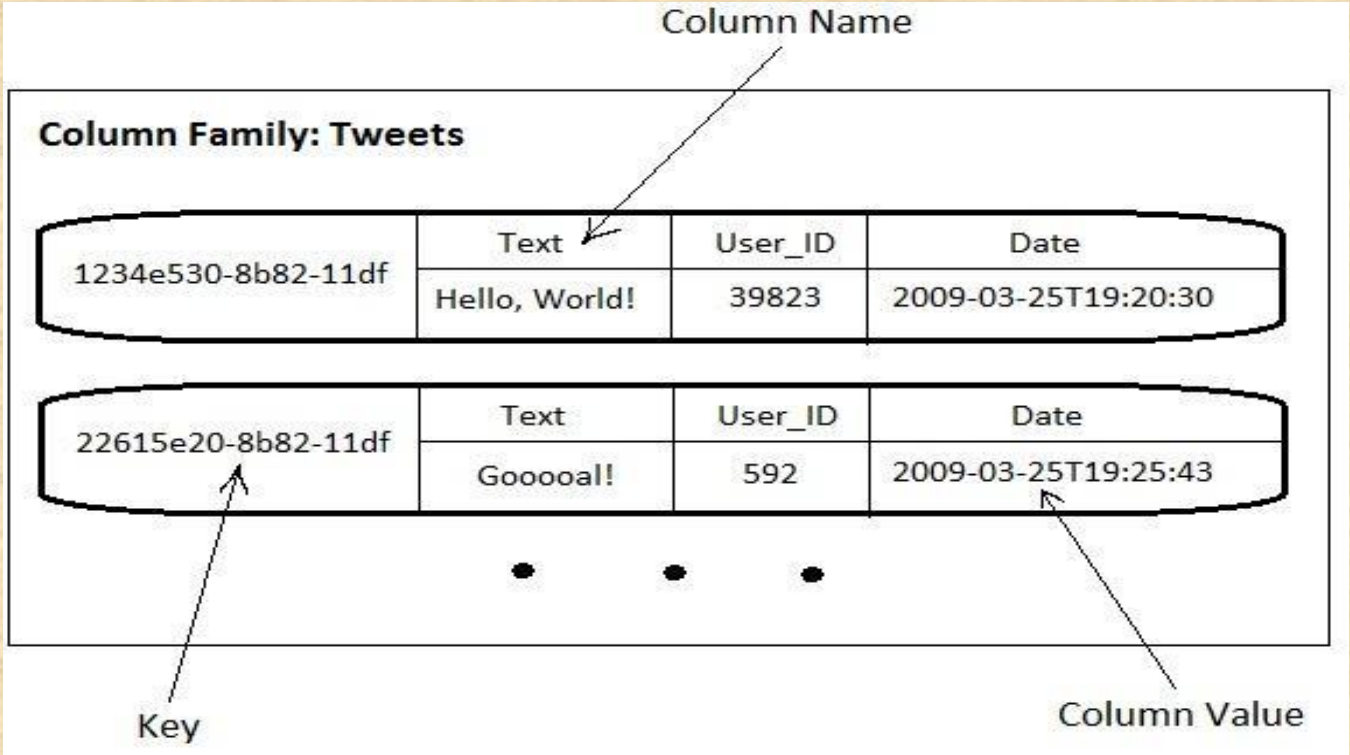


Example


KeySpace 1
Column family 1
RowID1
RowID2
Name: xxxx
Value: xxxx
Timestamp: xxxx
Column family 2
RowID1
RowID2
Name: xxxx
Value: xxxx
Timestamp: xxxx

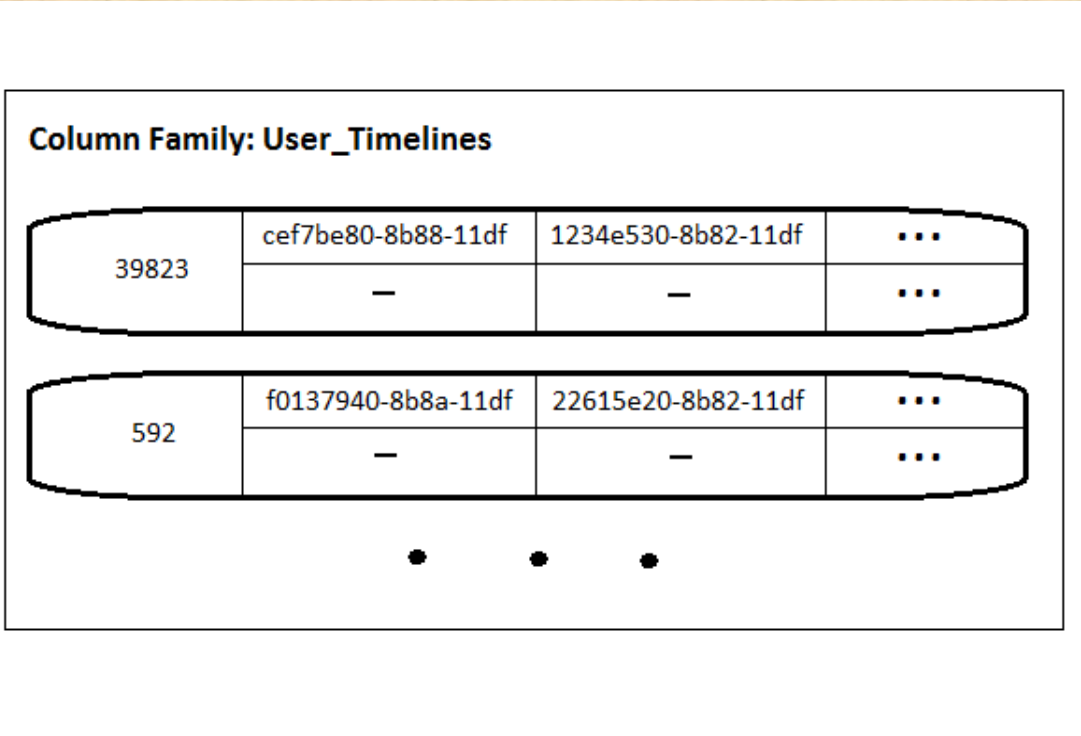
Cassandra Row

- the value of a row is itself a sequence of key-value pairs
- such nested key-value pairs are *columns*
- key = column name
- a row must contain at least 1 column



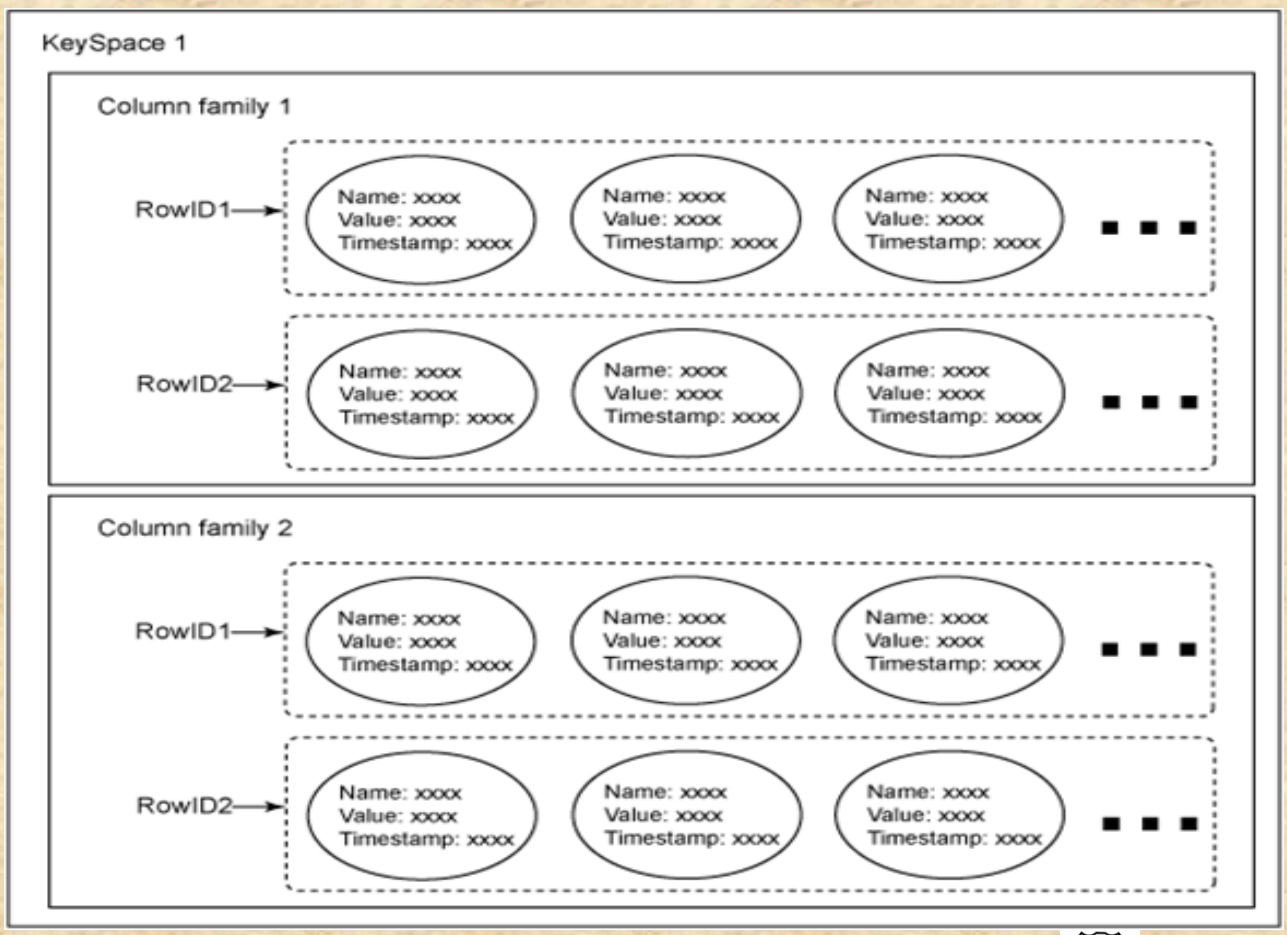
Column names storing values

- key: User ID
- column names store tweet ID values
- values of all column names are set to “-” (empty byte array) as they are not used



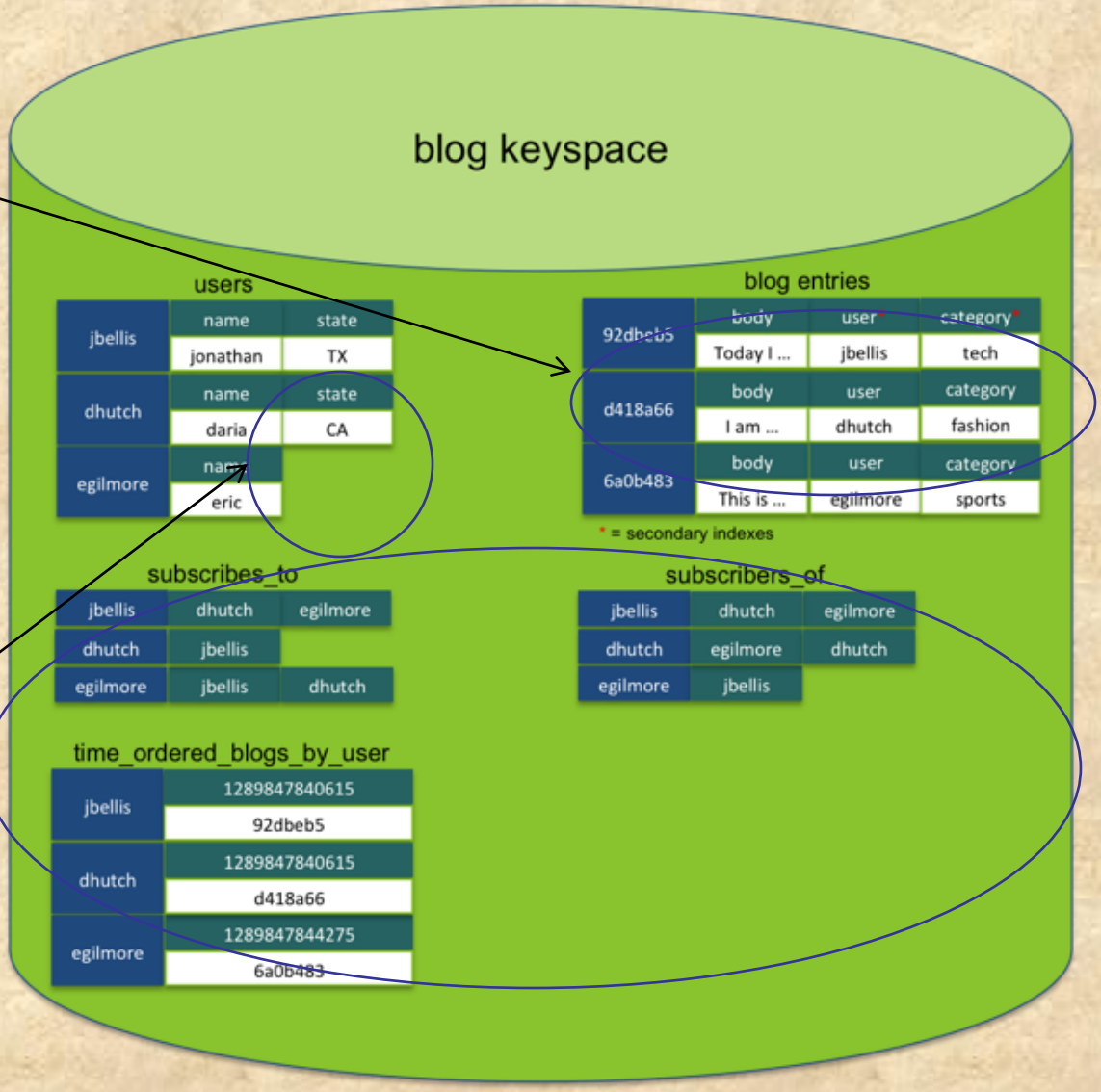
Key Space

- A Key Space is a group of column families together
- It is only a logical grouping of column families and provides an isolated scope for names



Cassandra Data : Storage

- **Column Families:**
  - Like SQL tables
  - but may be unstructured (client-specified)
  - Can have index tables
- **"column-oriented databases"/ "NoSQL"**
  - No schemas
  - Some columns missing from some entries
  - "Not Only SQL"
  - Supports get(key) and put(key, value) operations
  - Often write-heavy workloads



Cassandra Query Language - CQL

- creating a *keyspace* - namespace of tables

```
CREATE KEYSPACE demo
WITH replication = {‘class’: ’SimpleStrategy’,
‘replication_factor’: 3};
```

- to use namespace:

```
USE demo;
```

Cassandra Query Language - CQL

- creating tables

```
CREATE TABLE users(
email varchar,
bio varchar,
birthday timestamp,
active boolean,
time_posted));
PRIMARY KEY (email));
```

```
CREATE TABLE tweets(
email varchar,
time_posted timestamp,
tweet varchar,
PRIMARY KEY (email,
```

- inserting data

```
INSERT INTO users (email, bio, birthday, active)
VALUES ('john.doe@bti360.com', 'BT360 Teammate', 516513600000, true);
```

*** timestamp fields are specified in milliseconds since epoch*

- querying tables

SELECT expression reads one or more records from Cassandra column family and returns a result-set of rows

```
SELECT * FROM users;
```

```
SELECT email FROM users WHERE active = true;
```

Data Models of Cassandra and RDBMS

RDBMS	Cassandra
RDBMS deals with structured data.	Cassandra deals with unstructured data.
It has a fixed schema.	Cassandra has a flexible schema.
In RDBMS, a table is an array of arrays. (ROW x COLUMN)	In Cassandra, a table is a list of "nested key-value pairs". (ROW x COLUMN key x COLUMN value)
Database is the outermost container that contains data corresponding to an application.	Keyspace is the outermost container that contains data corresponding to an application.
Tables are the entities of a database.	Tables or column families are the entity of a keyspace.
Row is an individual record in RDBMS.	Row is a unit of replication in Cassandra.
Column represents the attributes of a relation.	Column is a unit of storage in Cassandra.
RDBMS supports the concepts of foreign keys, joins.	Relationships are represented using collections.

Apache Cassandra Advanced

Need : New Era of data – Big Data

Big data involves data that

1) is high **velocity** in nature;
2) combines structured, semi-structured, and unstructured data - **variety**;
3) can include enormous **volumes**; and
4) typically involves complexity in data distribution and synchronization.

The massive scale, high performance, and never-go-down nature of these applications has forged a new set of technologies that have replaced the legacy RDBMS with NoSQL database like Cassandra

The Architecture of Cassandra

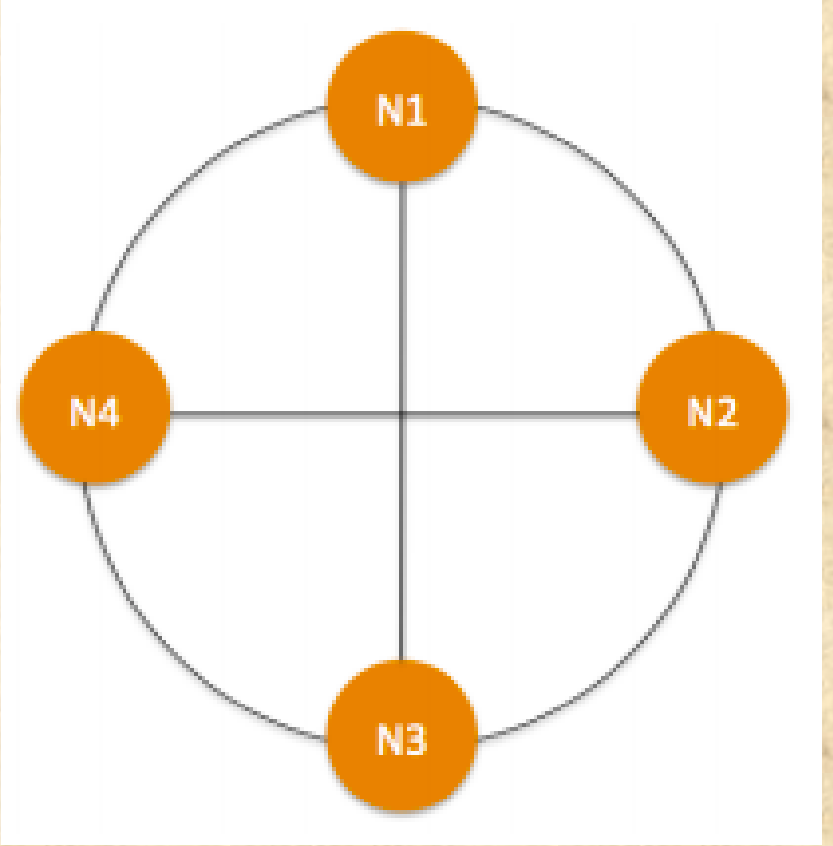
- Set of nodes
- peer-to-peer distributed architecture
- all nodes are the same; there is no concept of a master node
- with all nodes communicating with each other via a ***gossip protocol***
- capable of handling petabytes of information and thousands of concurrent users/operations per second (across multiple data centers)
- no single point of failure, true 24x7 availability

Gossip protocol

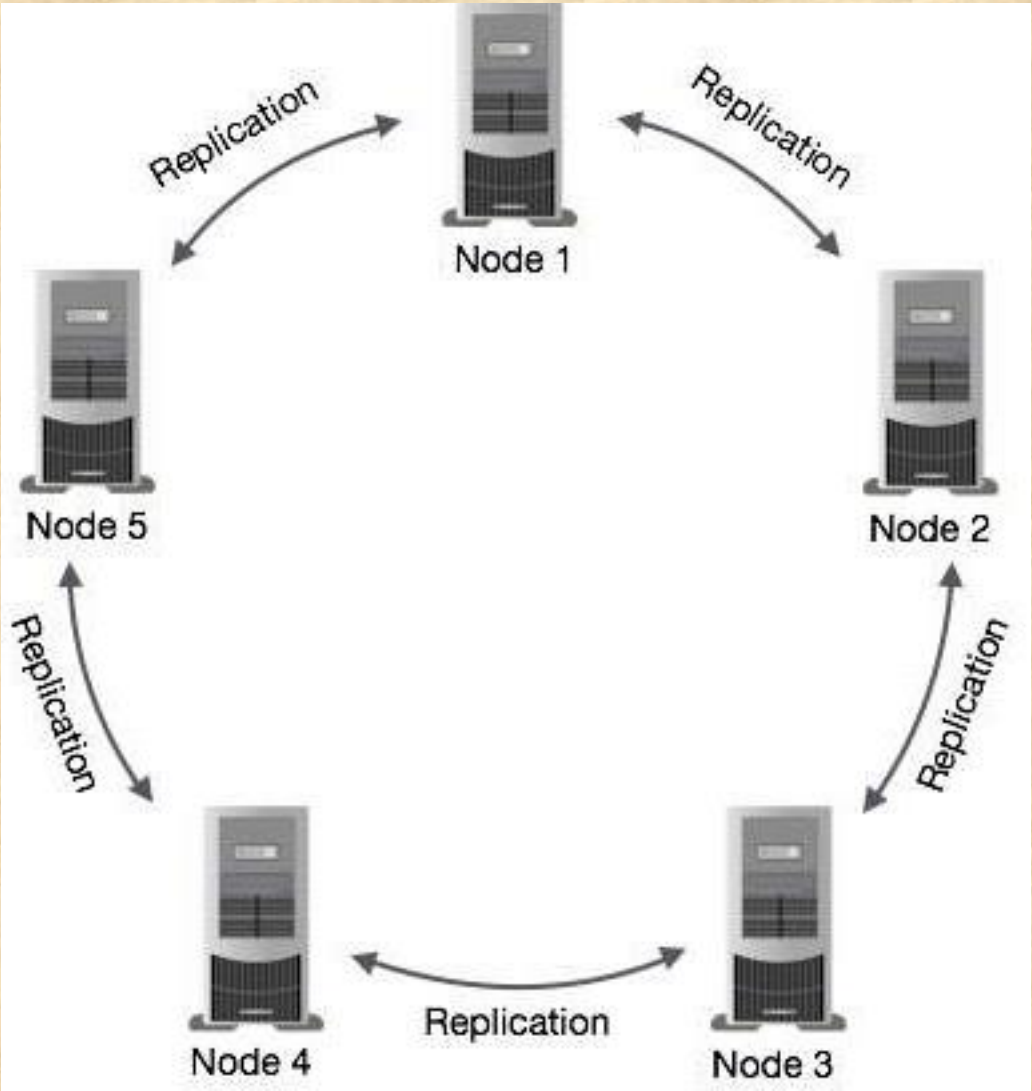
- It is similar to real-world gossip, where a node (say B) tells a few of its peers in the cluster what it knows about the state of a node (say A).
- Those nodes tell a few other nodes about A,
- and over a period of time, all the nodes know about A.

Distributing and Replicating Data

- Automatic data distribution across all nodes that participate in a “ring” or database cluster.
- Data is transparently partitioned across all nodes
  - Randomized (default)
  - ordered fashion
- Easy user-defined replication – “replication factor” parameter in keyspace creation

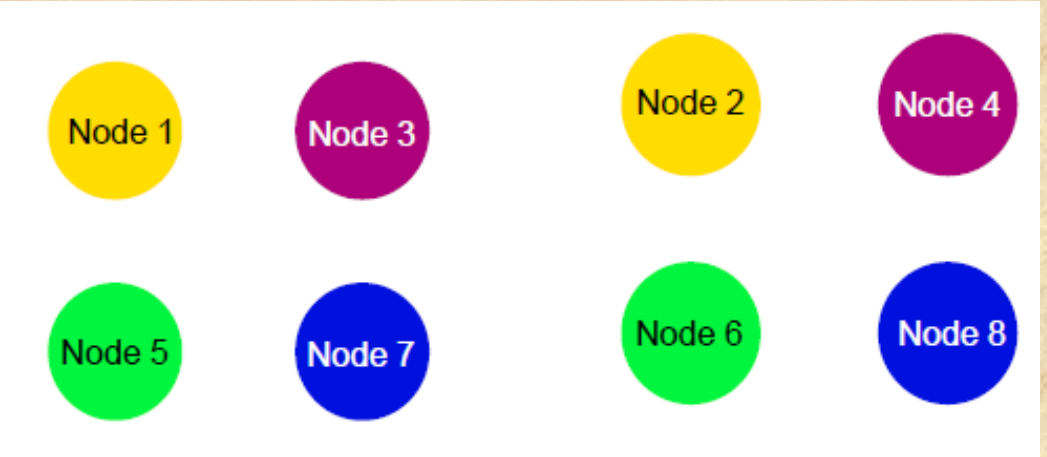


Replication

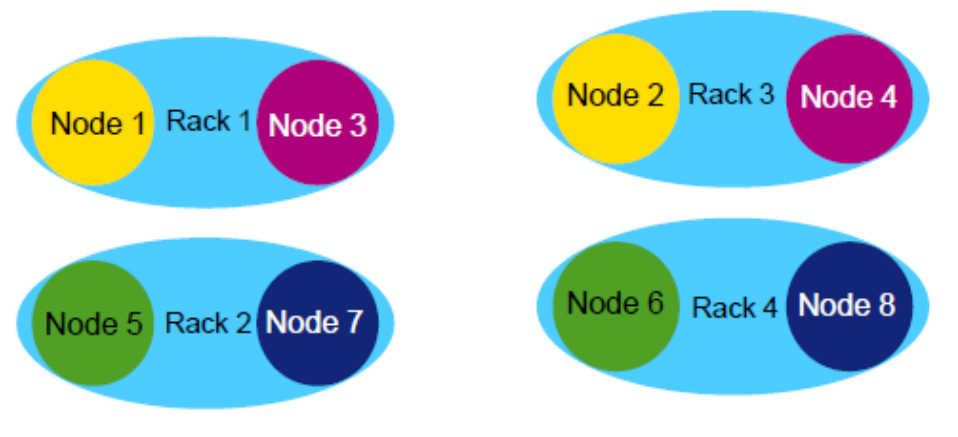


Architecture …

- One Node : Single Cassandra instance

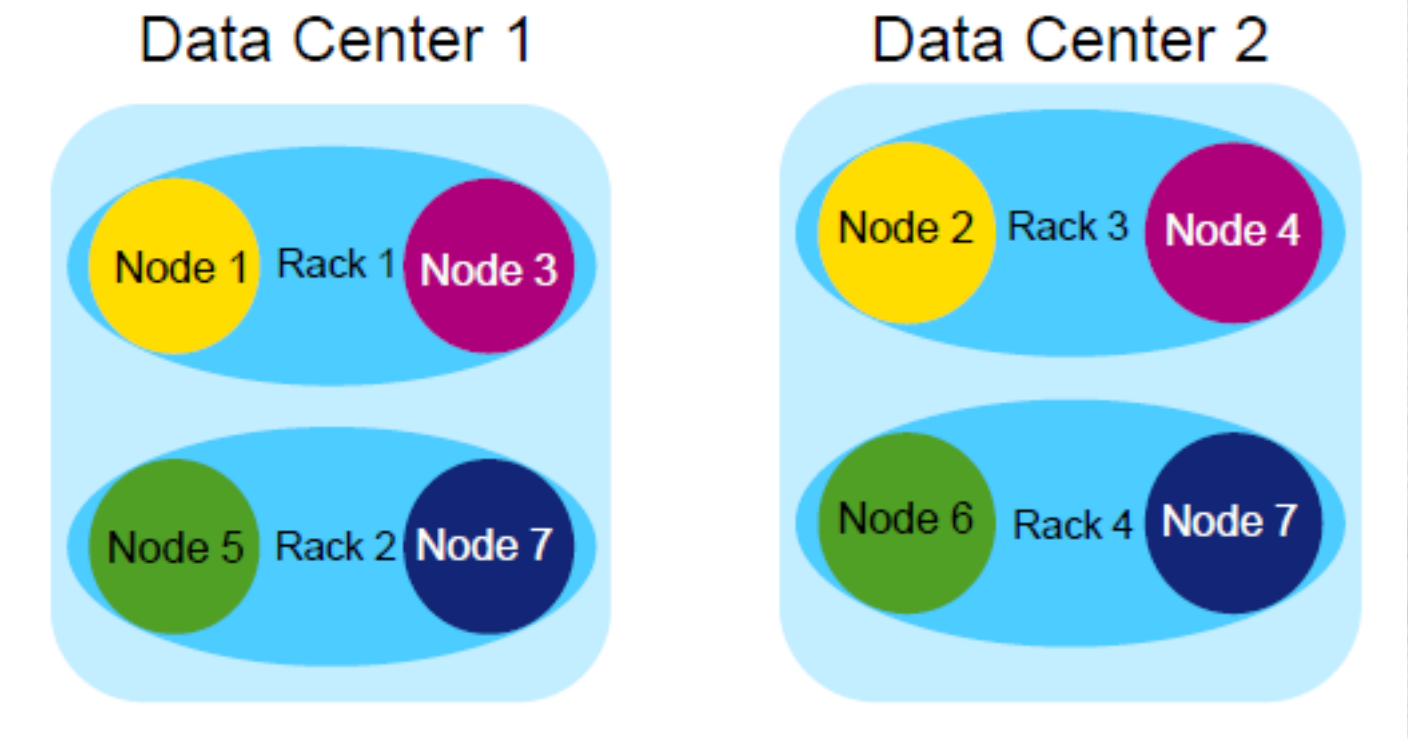


- Rack : Logical set of nodes

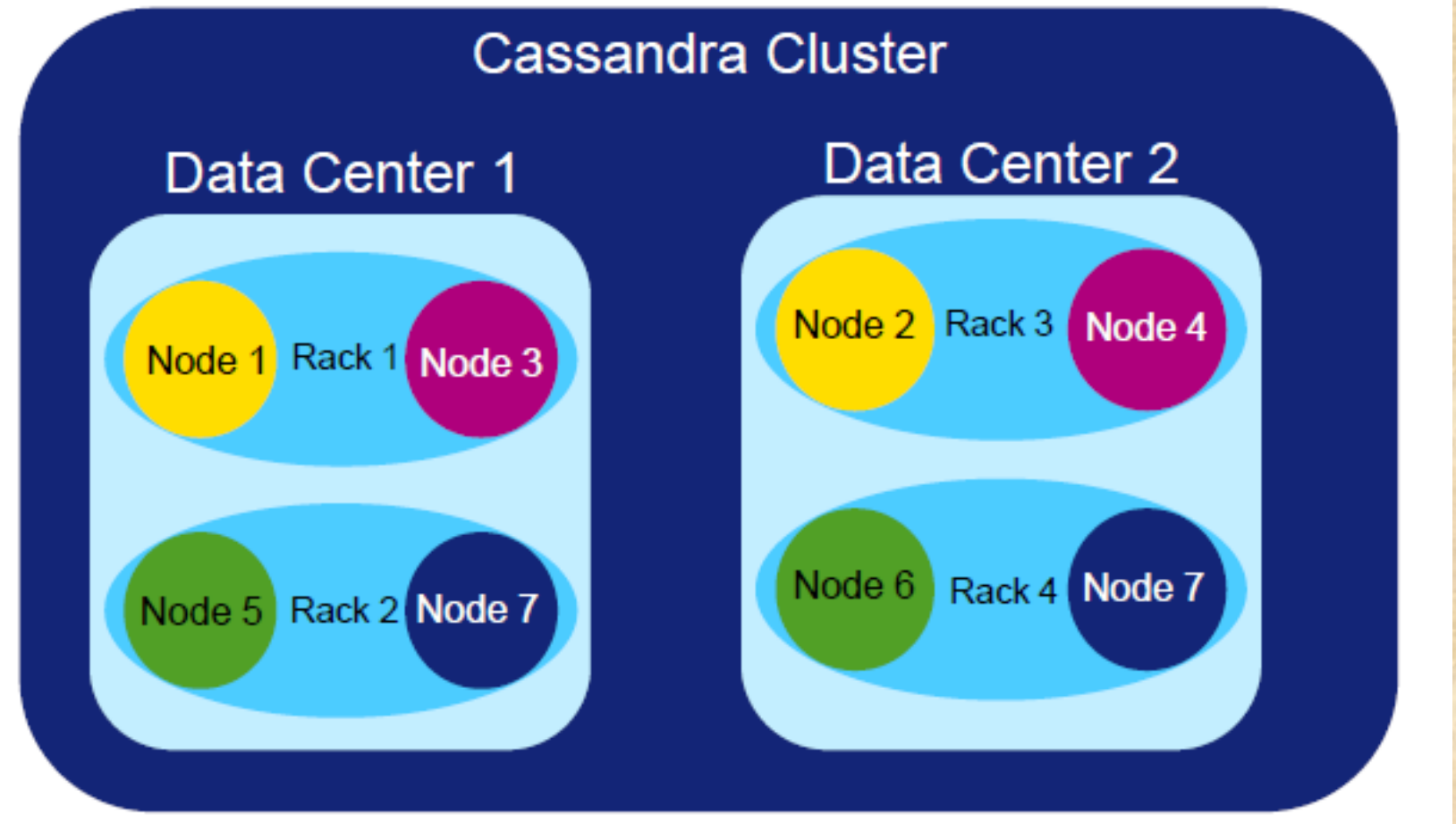


Architecture …

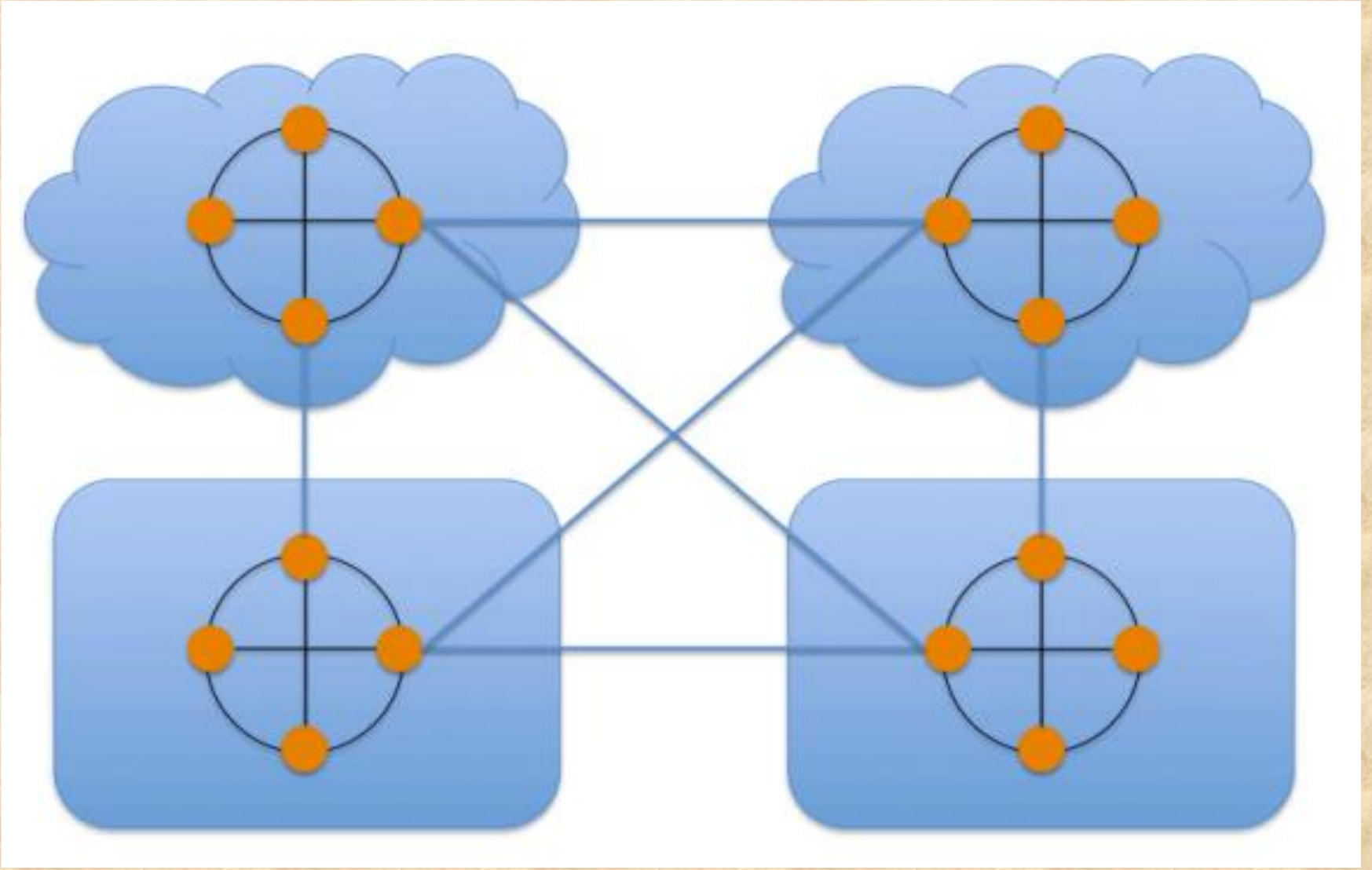
- Data Center : Logical set of Racks



Architecture …



Multi-Data Center and Cloud Support

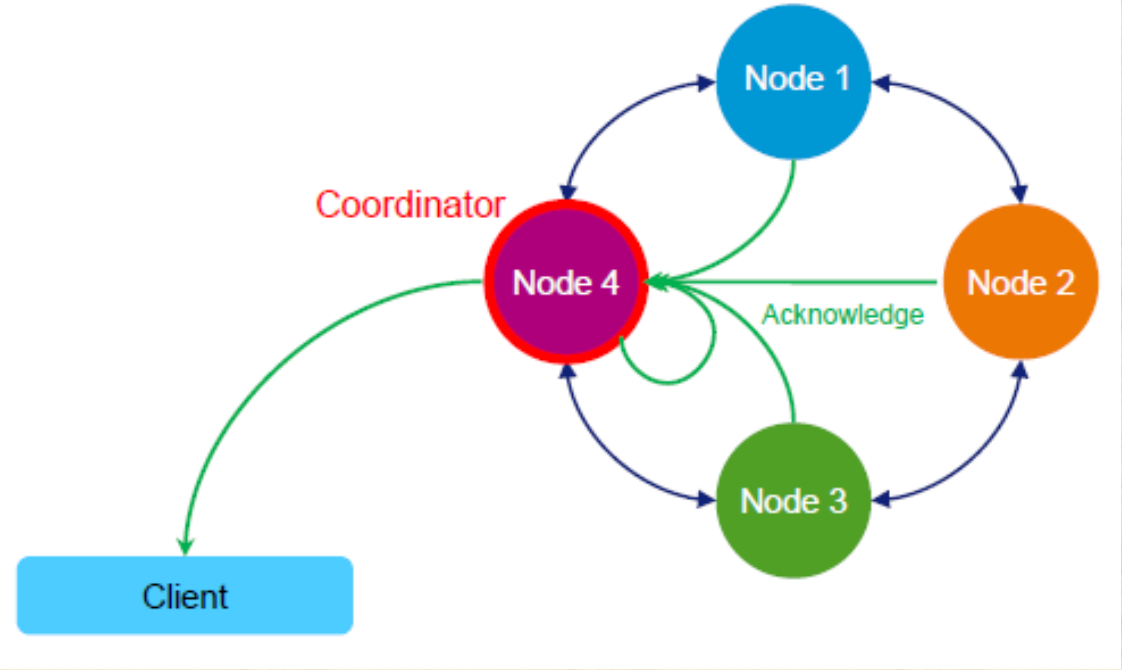


Reading and Writing Data

- true "**location independent**" architecture
- any node in a Cassandra cluster may be read or written to – ***true read/write-anywhere design***.
- it is first written to a **commit log**
- also to ***memtable***
- flushed to a disk structure called an ***sstable***
- user may requests data from any node, Cassandra engine assembled it from other nodes.

Request Coordination

- **Coordinator** : the node chosen by the client to receive a particular read or write request to its cluster.

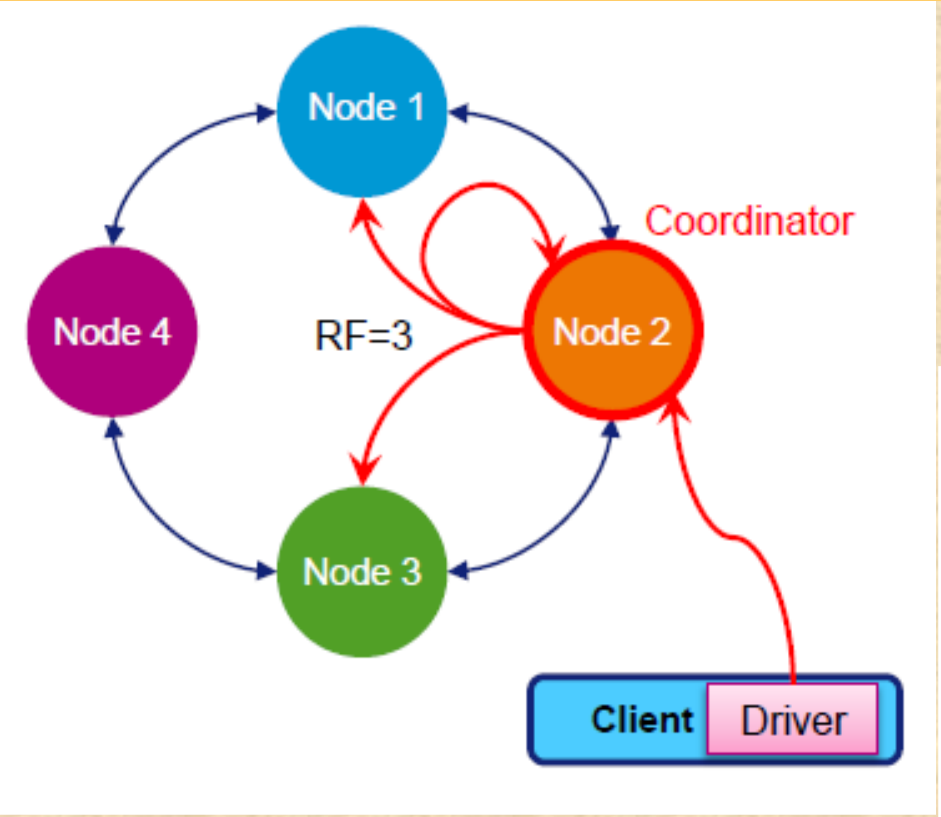


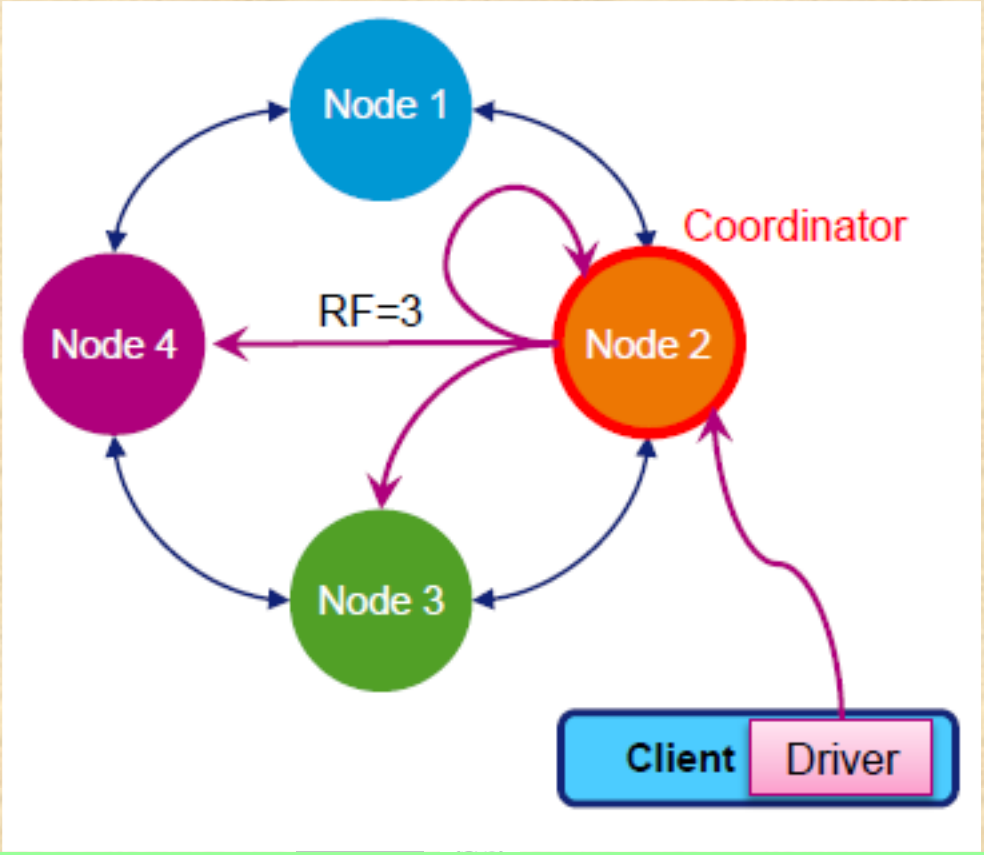
- Any node can coordinate any request
- Each client request may be coordinated by a different node

Replication of Data

- The **coordinator** manages the **replication process**
- **Replication Factor (RF)** : onto how many nodes should a write be copied
- The write will occur on the nodes responsible for that partition
- 1 ≤ **RF** ≤ (#nodes in cluster)
- Every write is time-stamped

Replication of Data





Transactions in Cassandra

- Cassandra is not a transactional database – no ACID.
- It uses "**AID**" - **A**tomic, **I**solated and **D**urable
- Cassandra offers ***tunable*** data **consistency** across a database cluster
- Developer or administrator can decide exactly how strong *(e.g., all nodes must respond)* or eventual *(e.g.,just one node responds, with others being updated eventually)* they want data consistency to be.

Consistency : Quorum Algorithm

- **The coordinator applies the Consistency Level (CL)**
- **Consistency Level (CL): Number of nodes which must acknowledge a request**
- **Examples of CL :** ONE , TWO , THREE , ANY , ALL (Not recommended)
- **QUORUM** = (RF/2 + 1)
- CL = QUORUM
- **CL may vary for each request**
- **On success, the coordinator notifies the client**

Cassandra : Application use cases

- Real-time, big data workloads
- Time series data management
- High-velocity device data consumption and analysis
- Media streaming management (e.g., music, movies)
- Social media (i.e., unstructured data) input and analysis
- Online web retail (e.g., shopping carts, user transactions)
- Real-time data analytics
- Online gaming (e.g., real-time messaging)
- Software as a Service (SaaS) applications that utilize web services
- Online portals (e.g., healthcare provider/patient interactions)
- Most write-intensive systems

Time series database example

- Cassandra is an excellent fit for time series data.
- Examples of TSDB use cases:
  - Ratings, recent purchases, and shopping cart
  - Session data, event streams, and click streams
  - Sensor data, application, and performance metrics
  - Velocities or windowed queries for a specific time period

Installation : Cassandra Clustering

- Install Cassandra on each node.
- Choose a name for the cluster.
- Get the IP address of each node.
- For each node, do the following configuration
- Go to ***conf*** folder in the ***apache-cassandra*** home directory.
- Open ***cassandra.yaml*** file in notepad and edit
- listen_address: <ip addres of node>
- rpc_address: <ip addres of node>
- - seeds: “<comma-delimited list of the IP address of each node in the cluster>"

Installation …

- The communication of Cassandra nodes mainly revolves around 2 ports which are :
  I. 7000 – TCP port for commands and data.
  II. 9042 – TCP port for the native transport server ***cqlsh***.
- Configure these ports on each node in the cluster
- Use ***nodetool status*** to see the status of cluster

```
anmols@anmols-15-3568:/opt/Applications/apache-cassandra-3.10$ ./bin/nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address        Load       Tokens       Owns (effective)  Host ID                               Rack
UN  192.168.2.122  7.28 MiB   256          49.4%             19fe4145-d49b-4009-b5d2-93c205452da0  rack1
UN  192.168.2.101  529.85 KiB  256          50.6%             74f9b27e-be87-411b-bae2-99ae1a657d7d  rack1
```

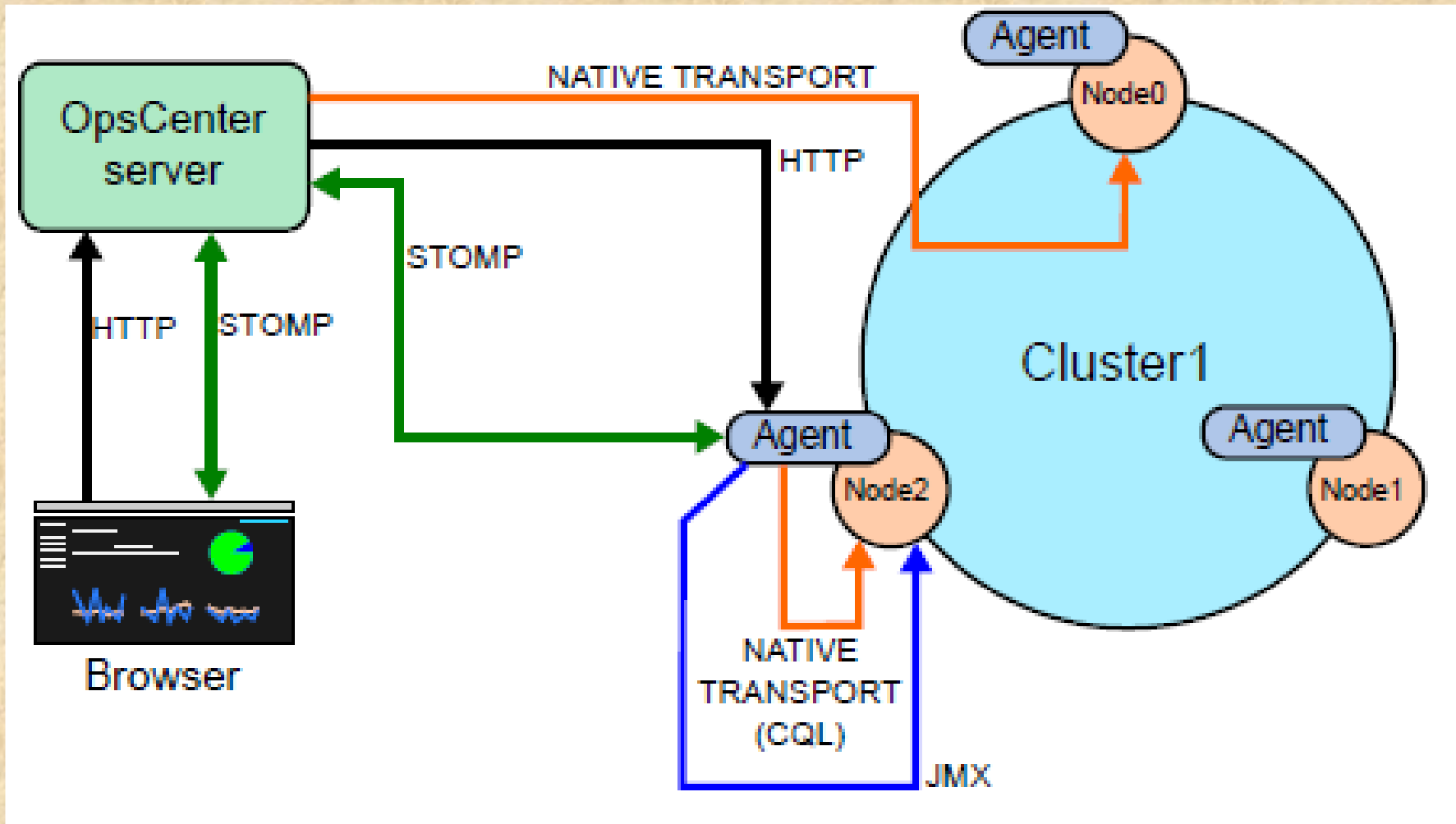
- Cluster is now ready to use.

DataStax OpsCenter

- DataStax OpsCenter is a visual management and monitoring solution for Apache Cassandra and DataStax Enterprise.
- It simplifies administration tasks such as:
  - Adding and expanding clusters
  - Configuring nodes
  - Viewing performance metrics
  - Rectifying issues
  - Monitoring the health of your clusters on the dashboard
- Available as open source Cassandra and DataStax Enterprise.

OpsCenter architecture overview

The agents use Java Management Extensions (JMX) to monitor and manage each node.



How to access OpsCenter ?
Web-based user interface

- **Open the browser**
- **URL : http://hostname:8888**
- **e.g. : http://10.4.1.101:8888**

Apache Cassandra Installation

Cassandra installation on windows

Dependencies / Pre-requisite

Apache Cassandra requires Java 8 to run on a Windows system. Additionally, the Cassandra command-line shell (*cqlsh*) is dependent on Python 2.7 to work correctly.

Cassandra installation on windows

1. Download and Install Java 8 and set environment variables.

2. Download and install Python 2.7 and set environment variables.

3. Download Cassandra tar.gz Folder from Apache Cassandra Download page (https://cassandra.apache.org/download/)

4. Unzip the compressed folder

5. Configure Environment Variables for Cassandra

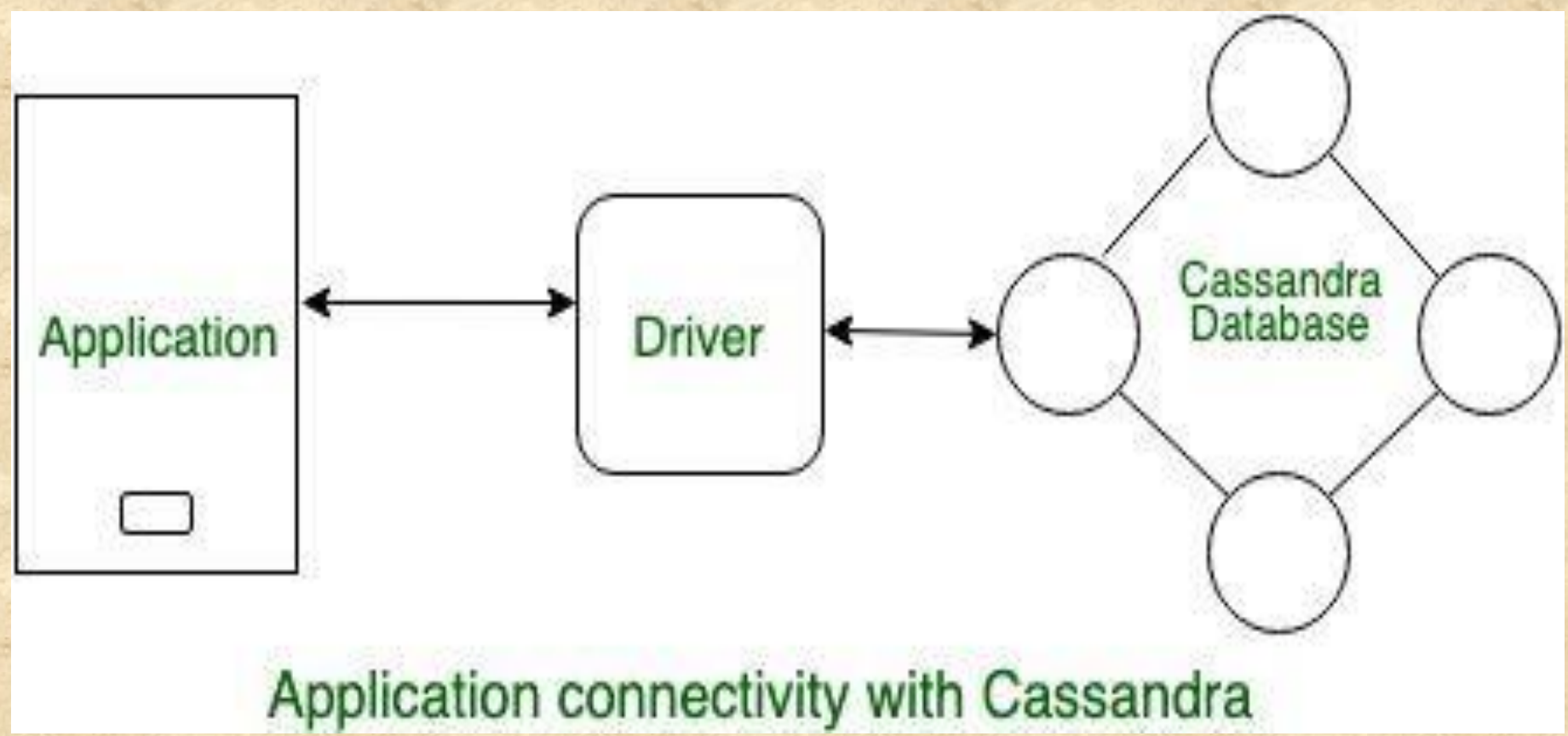
6. Go to Cassandra bin folder , open cmd window and type ***cassandra***

7. This will start cassandra server

8. Open another command window and type ***cqlsh***

9. This will start cassandra shell. Give commands here.

Application Development



Application connectivity with Cassandra

Driver ?

Cassandra Driver

- Python module for working with Cassandra database
- This module contains an ORM API, as well as a core API similar in nature to DB-API for relational databases.
- *pip3 install cassandra-driver*
- Interaction with Cassandra database, is done through Cluster object.

mongoDB

NoSQL

NoSQL Data Models

Cloud Databases

- Key-value
- Graph database
- Document-oriented
- Column family

Motivations

- **Problems with SQL**
  - Rigid schema
  - Not easily scalable (designed for 90's technology or worse)
  - Requires unintuitive joins

- **Perks of mongoDB**
  - Easy interface with common languages (Java, Javascript, PHP, etc.)
  - DB tech should run anywhere (VM's, cloud, etc.)
  - Keeps essential features of RDBMS's while learning from key-value noSQL systems

Introduction

- A Schema-less / Document Oriented Database
  - Data is stored in documents, not tables / relations
- MongoDB is Implemented in C++ for best performance
- **Platforms** : 32/64 bit Windows Linux, Mac OS-X, FreeBSD, Solaris
- **Language drivers :**Ruby/Ruby-on-Rails, Java, C#, JavaScript , C / C++ ,Erlang,Python, Perl…
- Replication & High Availability
- Map/Reduce
- Querying & Fast In-Place Updates

Why use MongoDB

- Simple queries
- Makes sense with most web applications
- Easier and faster integration of data
- Not well suited for heavy and complex transactions systems.
- Performance / Scalability / Availability
  - No Joins + No multi-row transactions
    - Fast Reads / Writes
  - Async writes
    - you don’t wait for inserts to complete
  - Secondary Indexes
    - Index on embedded document fields for superfast ad-hoc queries

Document store : analogy wrt RDBM

RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨ ⇨	Collection
Row	⇨ ⇨	Document (JSON, BSON)
Column		Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨	Reference
Partition		Shard

MongoDB's Data Model

- A MongoDB instance may have zero or more databases
- A Database has "Collections"
  - Collections have "Documents"
    - Documents have "Fields"
      - Fields are key = value pairs

  - A Collection does not enforce the structure of its documents*

  *i.e. Schemaless

JSON

- “JavaScript Object Notation”
- Easy for humans to write/read, easy for computers to parse/generate
- Objects can be nested
- Built on
  - name/value pairs
  - Ordered list of values

BSON

- “Binary JSON”
- Binary-encoded serialization of JSON-like docs
- Also allows “referencing”
- Embedded structure reduces need for joins
- Goals
  - Lightweight
  - Traversable
  - Efficient (decoding and encoding)

BSON Example

```
{
"_id" :         "37010"
"city" :        "ADAMS",
"pop" :         2660,
"state" :       "TN",
“councilman” : {
                    name: “John Smith”
                    address: “13 Scenic Way”
                  }
}
```

BSON Types

Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Object id	7
Boolean	8
Date	9
Null	10
Regular Expression	11
JavaScript	13
Symbol	14
JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Min key	255
Max key	127

The number can be used with the $type operator to query by type!

The _id Field

- By default, each document contains an _id field. This field has a number of special characteristics:
  - Value serves as primary key for collection.
  - Value is unique, immutable, and may be any non-array type.
  - Default data type is ObjectId, which is “small, likely unique, fast to generate, and ordered.” Sorting on an ObjectId value is roughly equivalent to sorting on creation time.

mongoDB vs. SQL

mongoDB	SQL
Document	Tuple
Collection	Table/View
PK: _id Field	PK: Any Attribute(s)
Uniformity not Required	Uniform Relation Schema
Index	Index
Embedded Structure	Joins
Shard	Partition

Installation and Running MongoDB

1. Download from mongodb.org
2. Unzip
3. Create data directory

```
>mkdir c:\data\db
```

4. Run MongoDB (mongod):

```
>cd c:\mongodb-1.6.3\bin
>mongod
```

5. Run Mongo shell (mongo):

```
>mongo
```

The Mongo Shell

>mongo

>help()

>show dbs

>use <dbname>

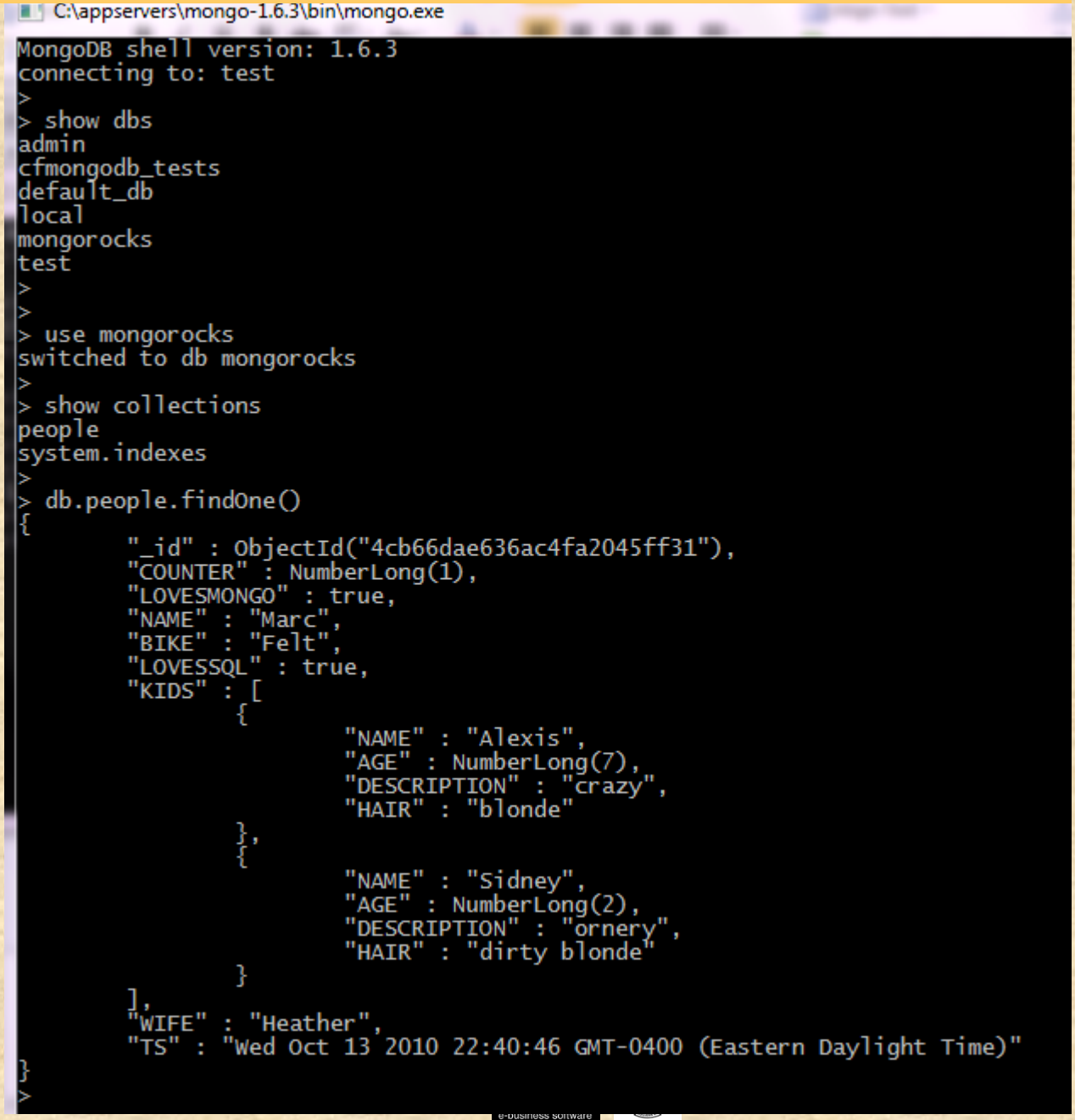
>show collections

>db.collectionName.findOne()

>db.collectionName.find()

>db.help()

>db.collectionName.help()



CRUD

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

CRUD example

```
> db.user.insert({
        first: "John",
        last : "Doe",
        age: 39
})
```

```
> db.user.find ()
{
        "_id" : ObjectId("51…"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39
}
```

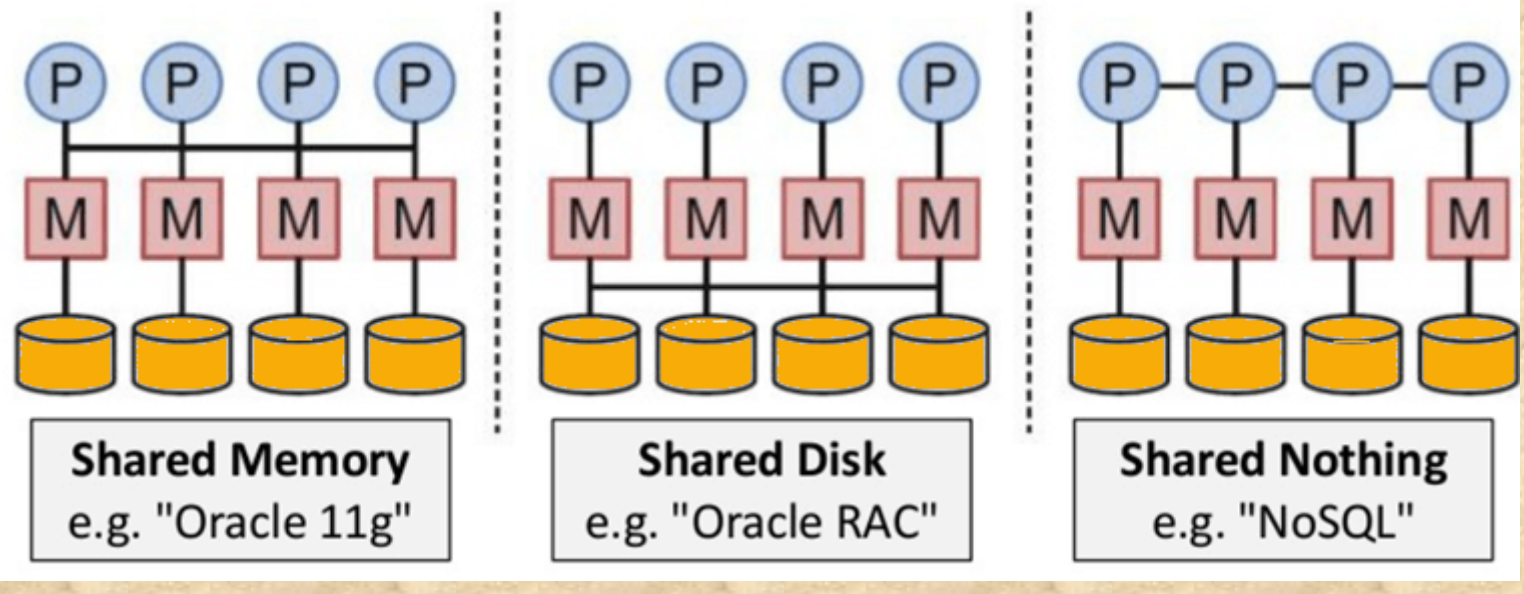
```
> db.user.update(
        {"_id" : ObjectId("51…")},
        {
            $set: {
                age: 40,
                salary: 7000}
        }
)
```

```
> db.user.remove({
        "first": /^J/
})
```

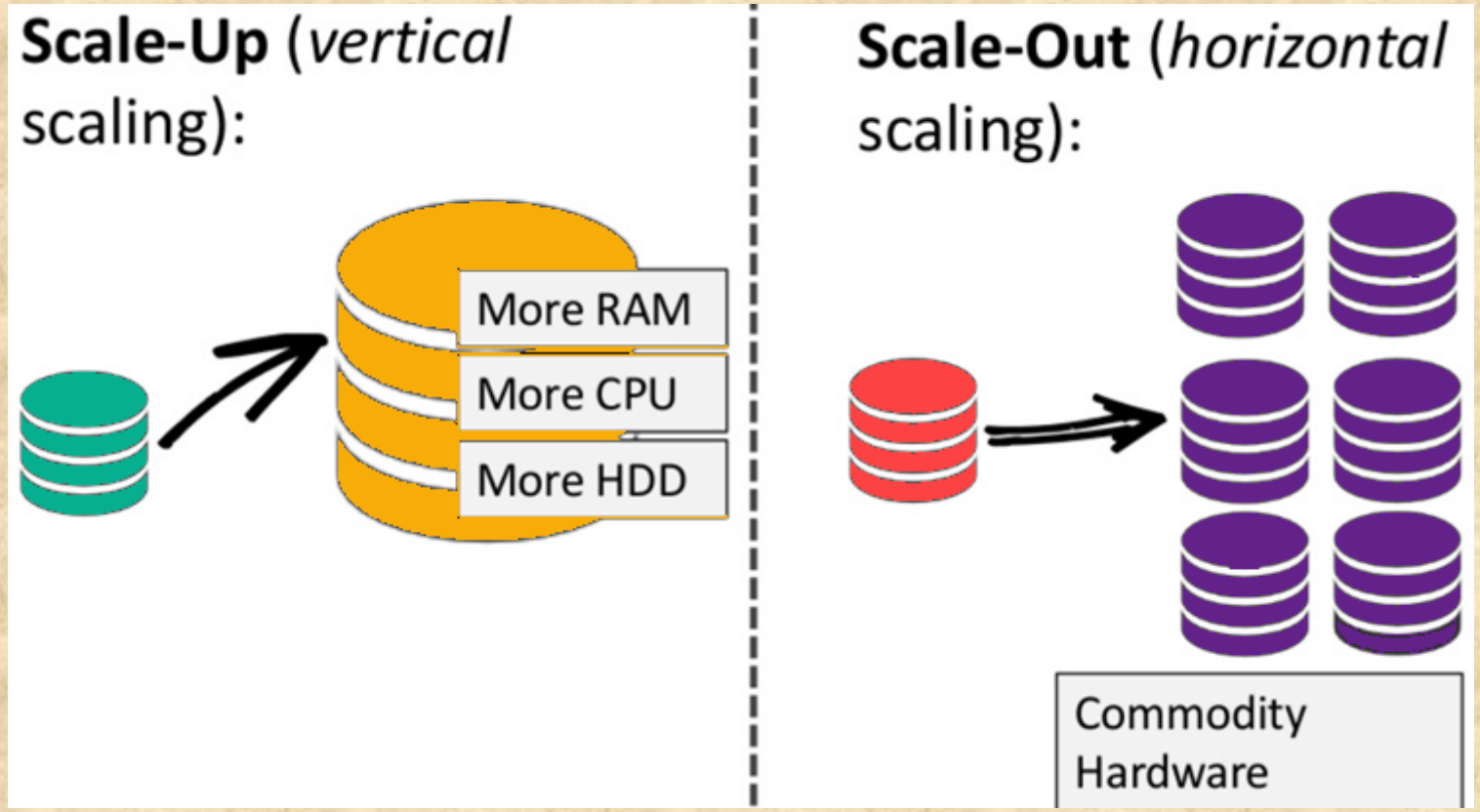
mongoDB

Storage Architecture of mongoDB

- **Replication**
- ***Sharding* (partitioning)**



Scalability



Why mongoDB

- exceptional scalability
- It makes it easy to fetch the data and provides continuous and automatic integration.
- No downtime while the application is being scaled
- Performs in-memory processing
- Text search
- Graph processing
- Global replication
- Economical

What is MongoDB Used For and when Should Use it

- used in a wide variety of ways to support applications in IoT, Gaming,Logistics, Banking, e-Commerce, and Content Management.
- Integrating large amounts of diverse data: If you are bringing together tens or hundreds of data sources, the flexibility and power of the document model can create a unified single view in ways that other databases cannot. MongoDB has succeeded in bringing such projects to life when approaches using other databases failed.

Usage ….

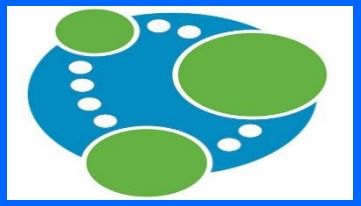
- **Describing complex data structures that evolve:** Document databases allow embedding of documents to describe nested structures and easily tolerate variations in data in generations of documents. Specialized data formats like geospatial are efficiently supported. This results in a repository that is resilient and doesn’tbreak or need to be redesigned every time something changes

Usage …

- **Supporting hybrid and multi-cloud applications:**MongoDB can be deployed and run on a desktop, ahuge cluster of computers in a data center, or in a public cloud, either as installed software or through MongoDB Atlas, a database as a service product. If you have applications that need to run wherever theymake sense, MongoDB supports any configuration now and in the future.

Usage …

- **Supporting agile development and collaboration:** Document databases put developers in charge of the data. Data becomes like code that is friendly to developers. This is far different from making developers use astrange system that requires a specialist. Document databases also allow evolution of the structure of the data as needs are better understood. Collaboration and governance can take place by allowing one team to control one part of a document and another team to control another part.



Neo4j

https://neo4j.com/

Introduction : Graph Database

- A database with an explicit graph structure
- with nodes, edges, and properties to represent and store data
- Each node knows its adjacent nodes
- Thus provides index-free adjacency
- Graph databases are schemaless
- ***Native / built-in*** support to represent relationships
- ACID-compliant transactional DB
- Accessible from Java API , the ***Cypher query language***

What is a Graph?

- An abstract representation of a set of objects where some pairs are connected by links.

Object (Vertex, Node)

Link (Edge, Arc, Relationship)

Graph Database

Data Model :

- **Nodes and**
- **Relationships**

Graph DB vs Relational DB

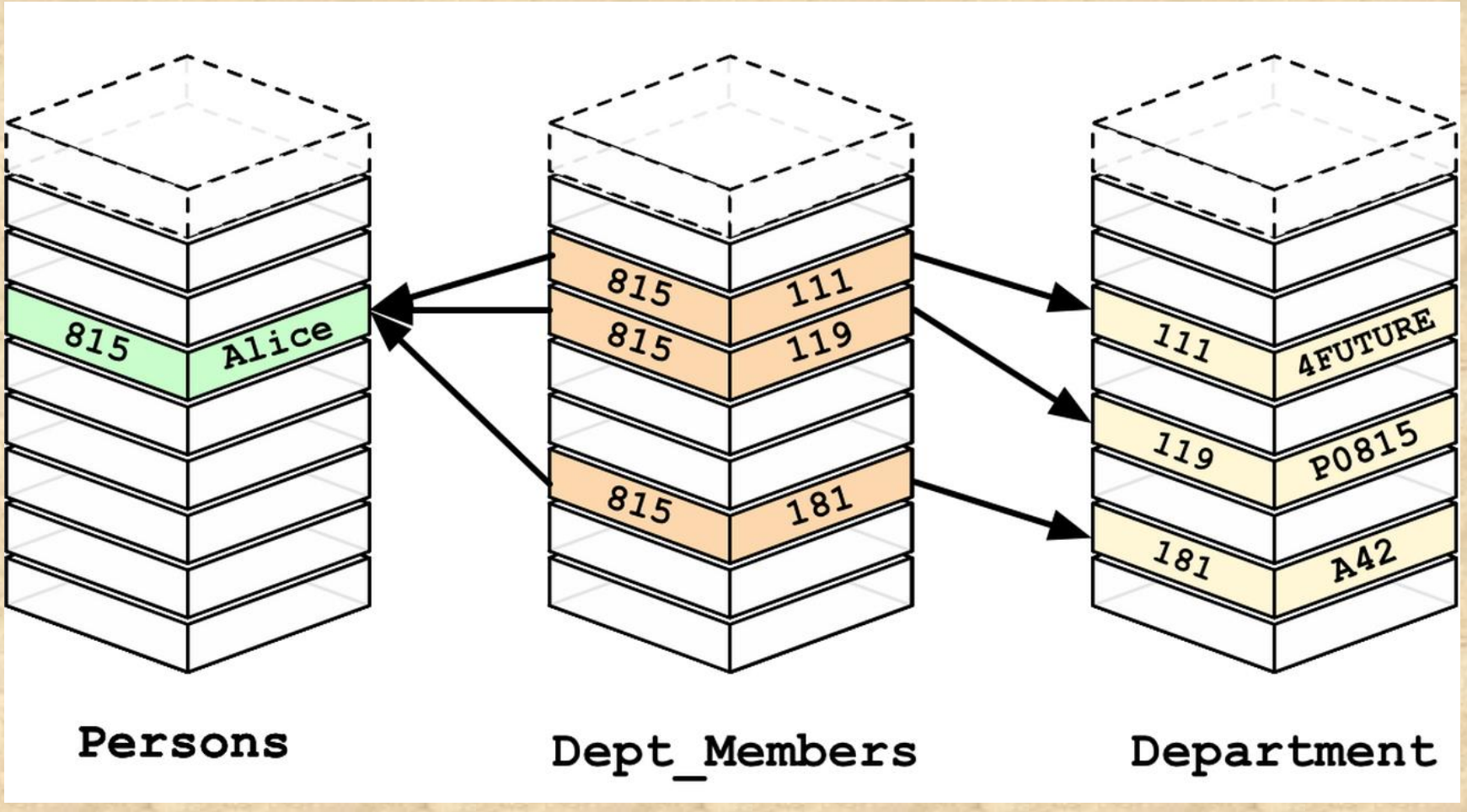
Graph

1. Graphs
2. Nodes
3. Properties & its Values
4. Relationships
5. Traversal

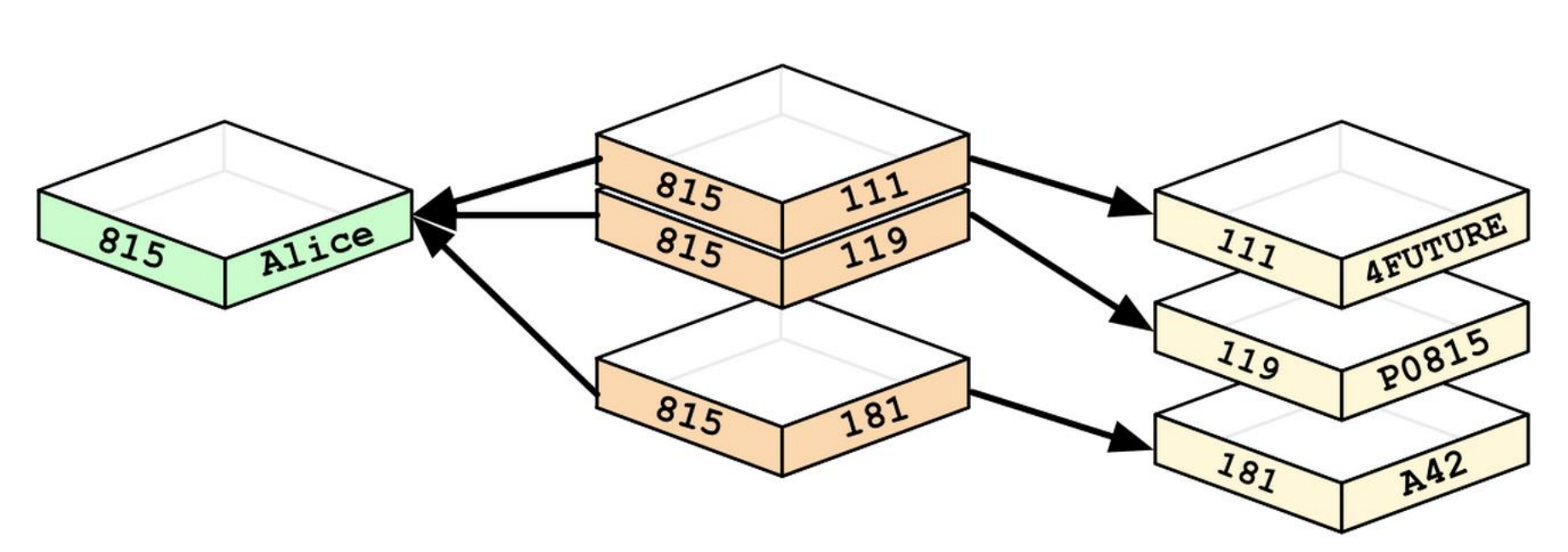
Relational

1. Tables
2. Rows
3. Columns & Data
4. Constraints
5. Joins

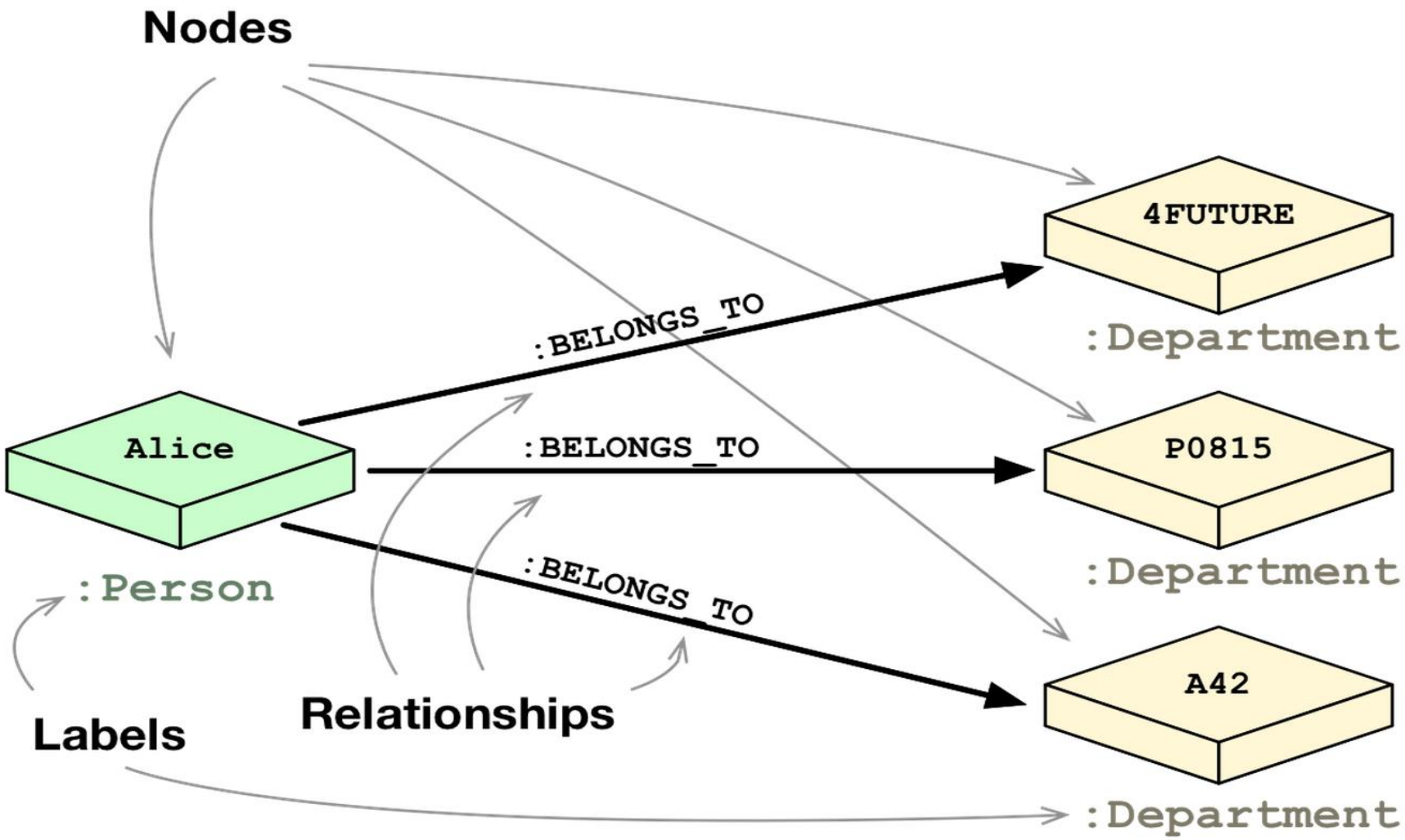
Relational Databases



Graph Databases



Graph Database

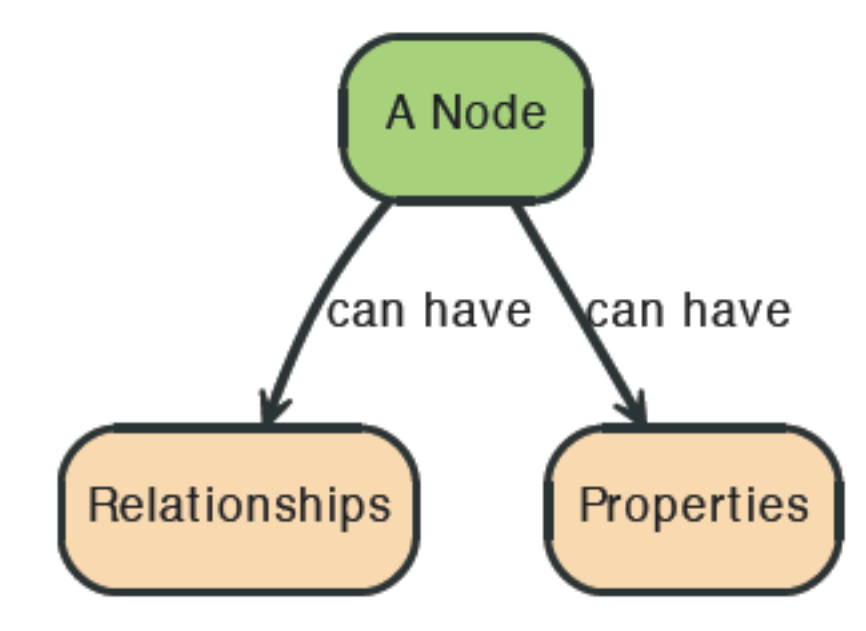


Mapping

- Each entity table is represented by a label on nodes
- Each row in a entity table is a node
- Columns on those tables become node properties.
- Join tables are transformed into relationships, columns on those tables become relationship properties

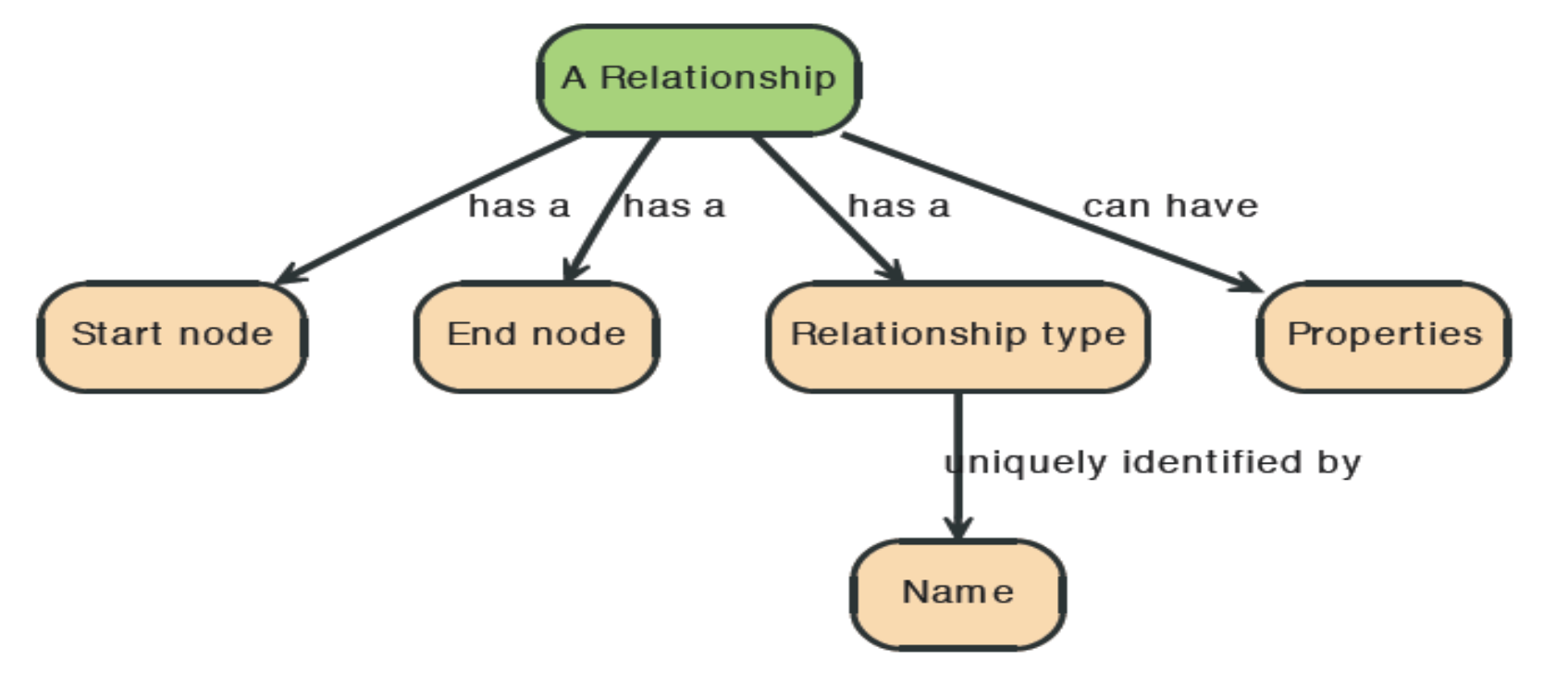
Node in Neo4j : Fundamental unit

contains properties with key-value pairs

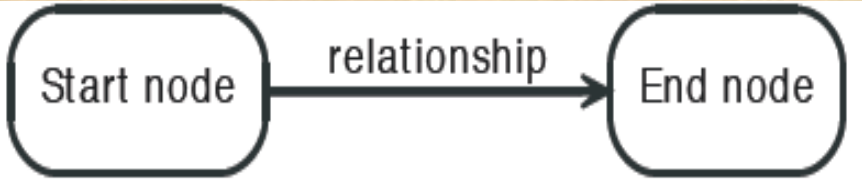


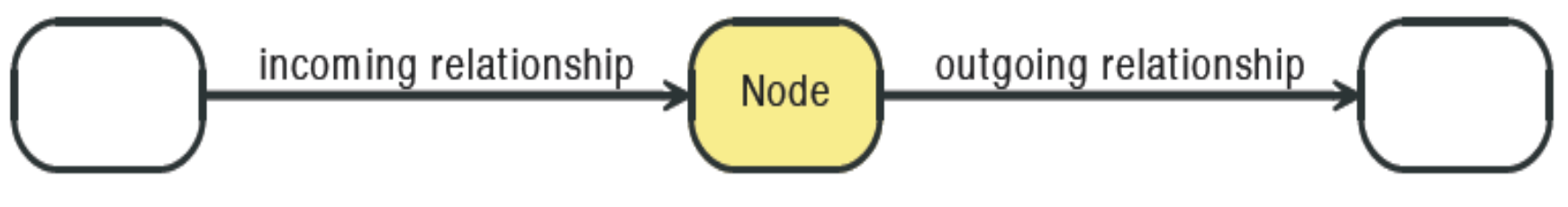
Relationships in Neo4j

- Relationships between nodes are a key part of Neo4j.



Relationships in Neo4j





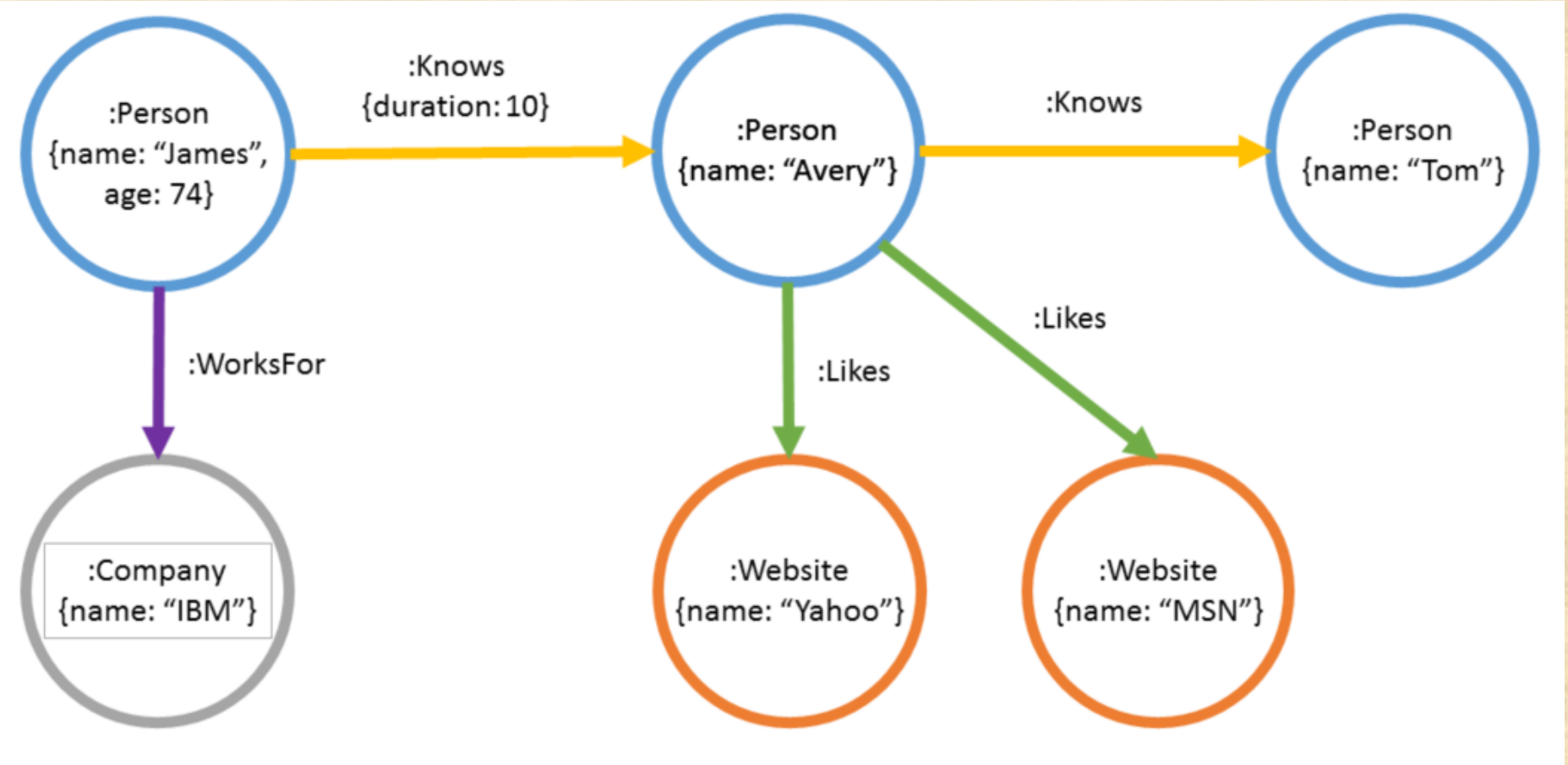
Properties

- Both nodes and relationships can have properties.
- Properties are key-value pairs where the key is a string.
- Property values can be either a primitive or an array of one primitive type.
- For example String, int and int[] values are valid for properties.

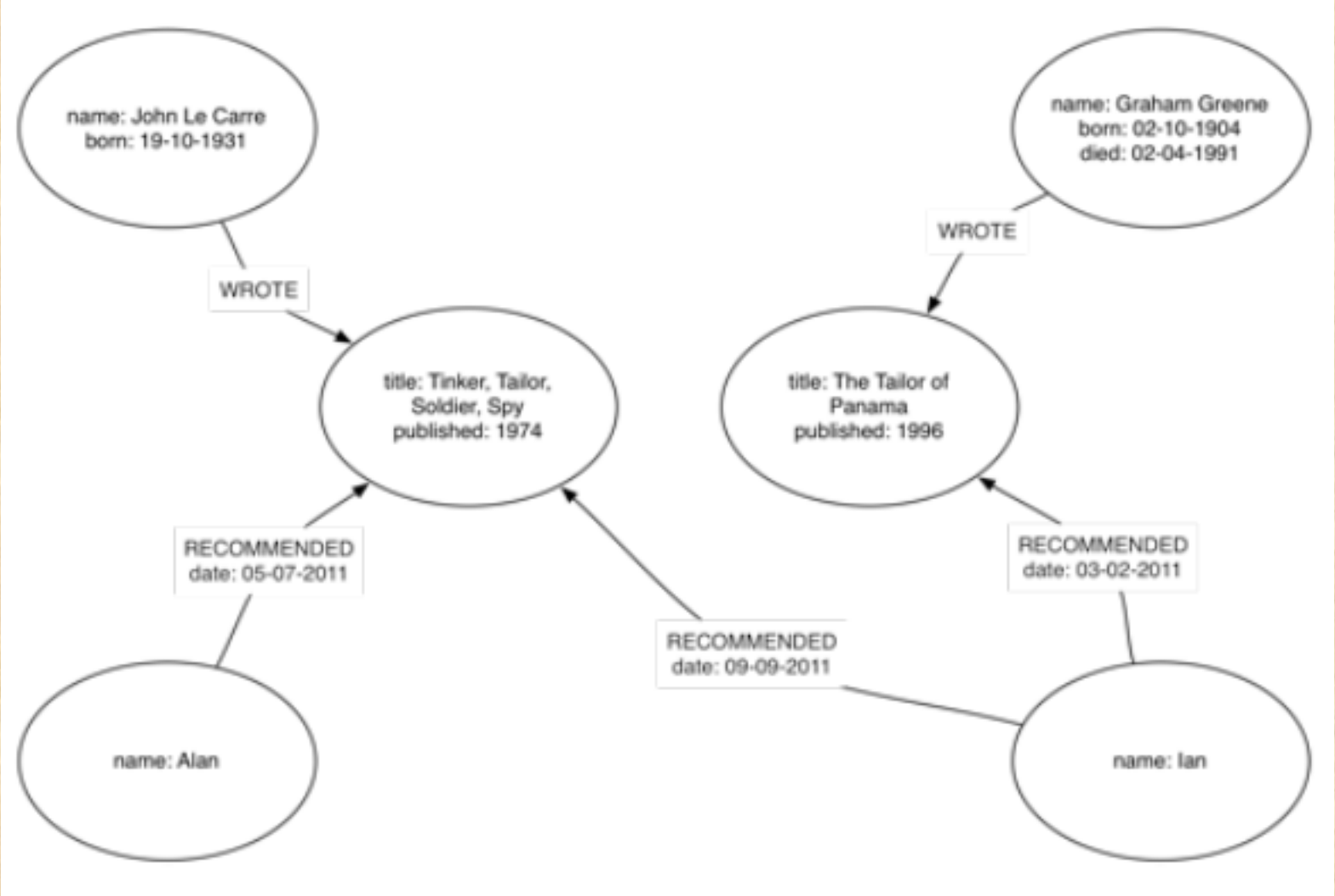
Supported data type in Neo4j

- **Number**, an abstract type, which has the subtypes **Integer** and **Float**
- **String**
- **Boolean**
- The spatial type **Point**
- Temporal types: **Date**, **Time**, **LocalTime**, **DateTime**, **LocalDateTime** and **Duration**

Example

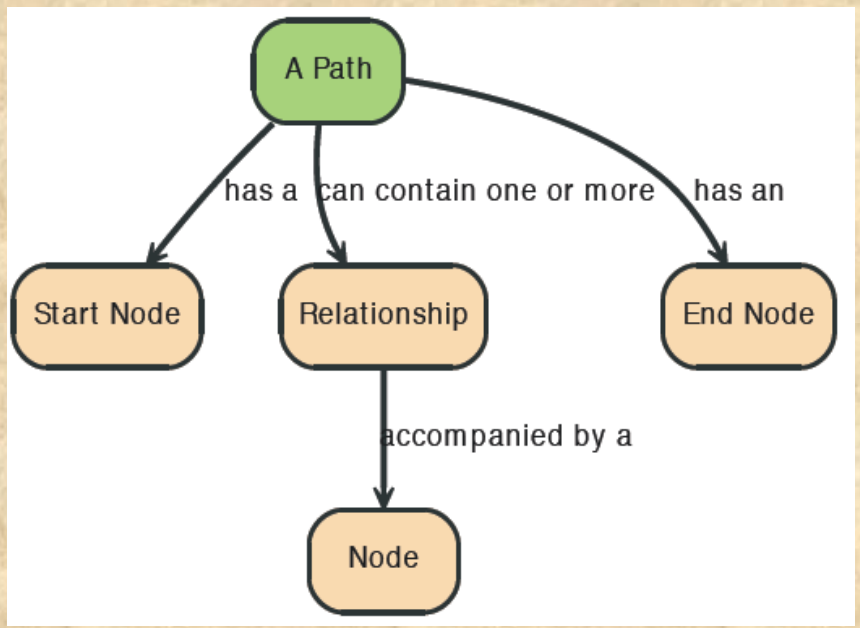

:Person
{name: "James",
age: 74}
:Knows
{duration: 10}
:Person
{name: "Avery"}
:Knows
:Person
{name: "Tom"}
:WorksFor
:Likes
:Likes
:Company
{name: "IBM"}
:Website
{name: "Yahoo"}
:Website
{name: "MSN"}

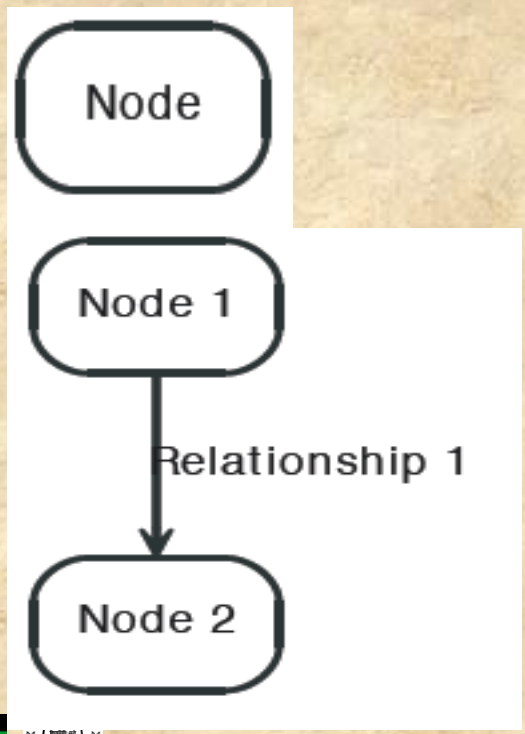
Example


name: John Le Carre
born: 19-10-1931
WROTE
title: Tinker, Tailor, Soldier, Spy
published: 1974
RECOMMENDED
date: 05-07-2011
name: Alan
name: Graham Greene
born: 02-10-1904
died: 02-04-1991
WROTE
title: The Tailor of Panama
published: 1996
RECOMMENDED
date: 03-02-2011
RECOMMENDED
date: 09-09-2011
name: Ian

Paths in Neo4j – Traversing : Query

- A path is one or more nodes with connecting relationships, typically retrieved as a query or traversal result.

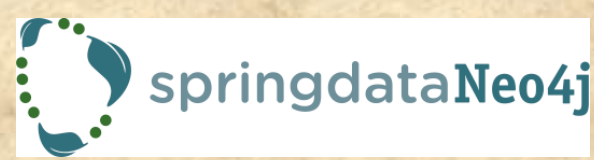




Powered by

Dr. Bashirahamad F. Momin
CSE Dept., Walchand COE, Sangli.

Supported platforms



REST://

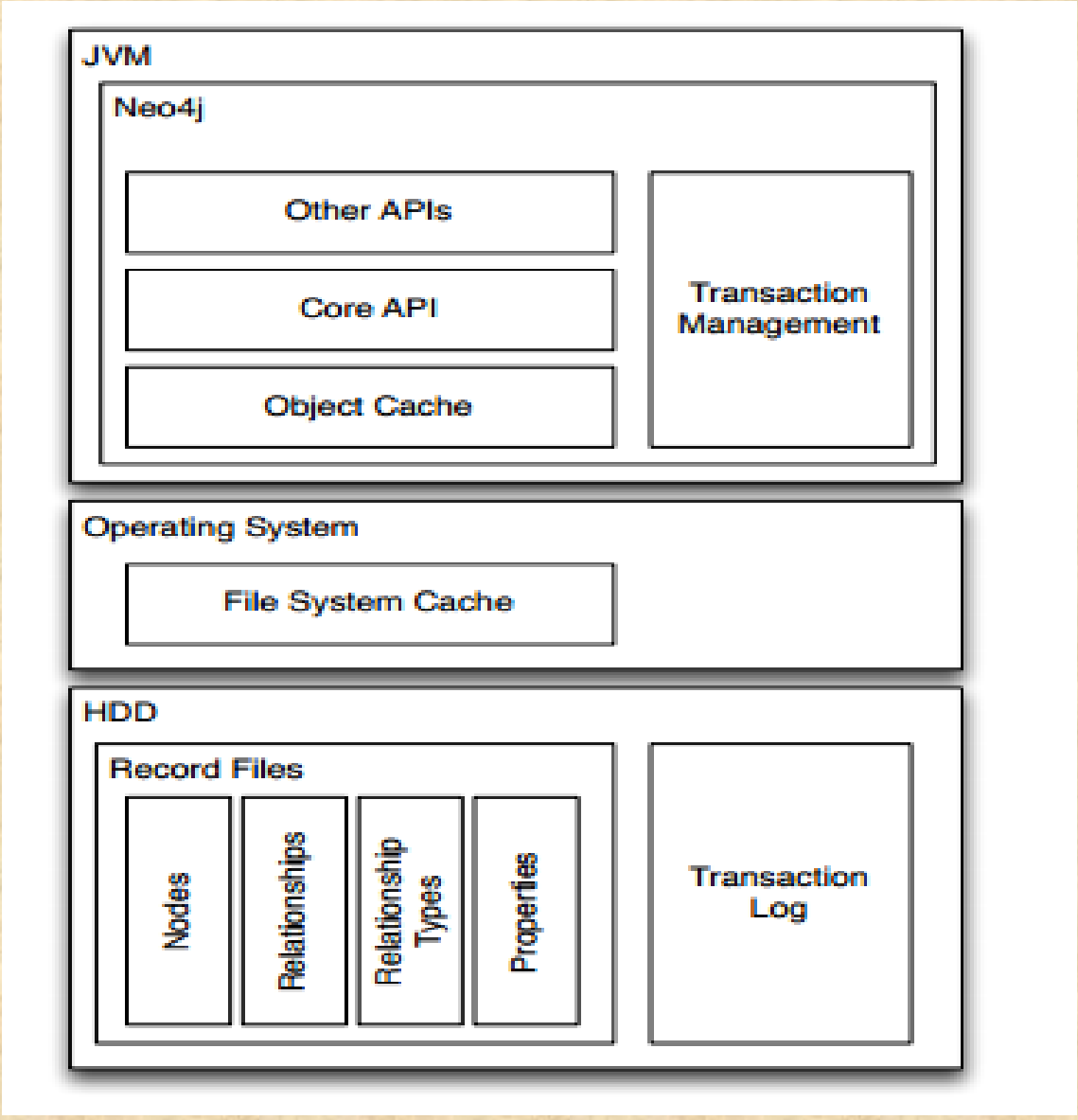
When to use Graph Databases ?

- Complex data
- Densely--connected, semi--structured domains
  - Lots of join tables? Connectedness
  - Lots of sparse tables? Semi--structure
- Data Model Volatility
- Easy to evolve
- Join Complexity and Performance
- Millions of "joins" per second
- Consistent query times as dataset grows

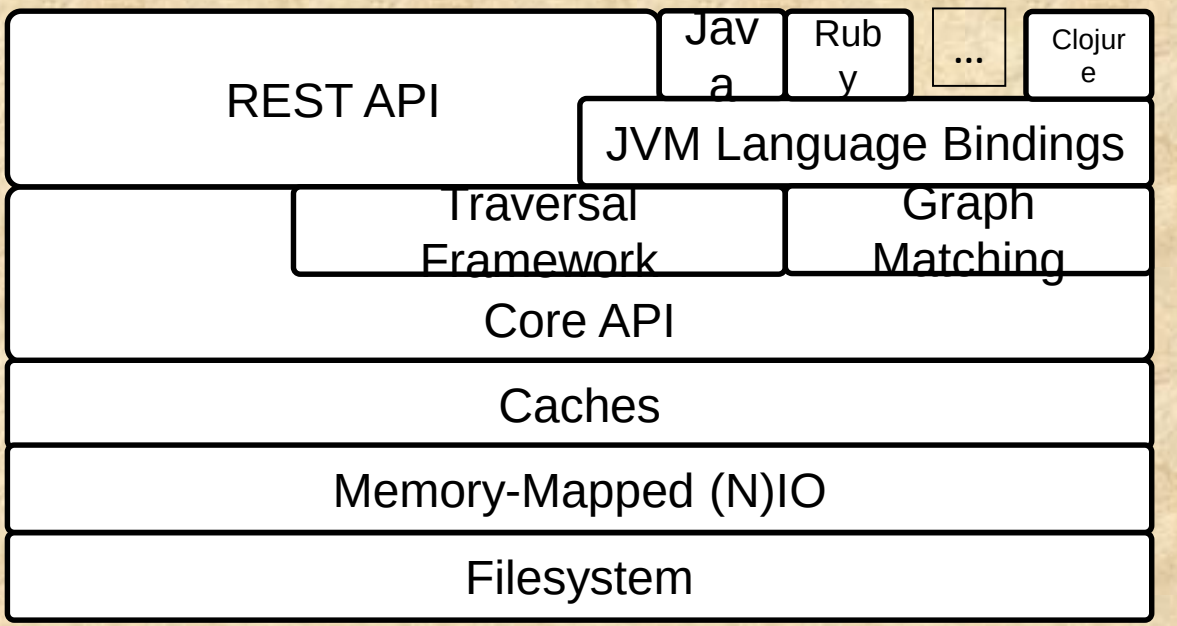
Target applications of graphs database

- Recommendations
- Business intelligence
- Social computing
- Geospatial
- Systems management
- Web of things
- Genealogy
- Time series data
- Product catalogue
- Web analytics
- Scientific computing (especially bioinformatics)
- Indexing your *slow* RDBMS
- And much more!

Neo4j Software Architecture



Neo4j Logical Architecture



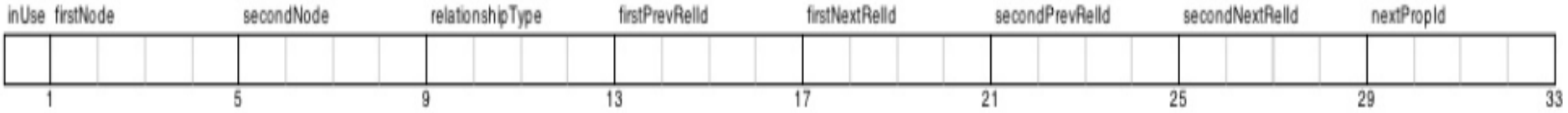
Storage File Organization

- Neo4j stores graph data in a number of different store files.
- Each store file contains the data for a specific part of the graph e.g. **nodes**, **relationships**, **properties**

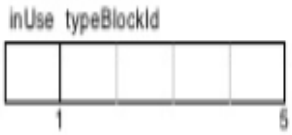
Store File Formats

Node (9 bytes)

Relationship (33 bytes)



Relationship Type (5 bytes)



Node store

- Size:9 bytes
  - **First byte**:in-use flag
  - **Next 4 bytes**:ID of first relationship
  - **Last 4 bytes**:ID of first property of node

Relationship store

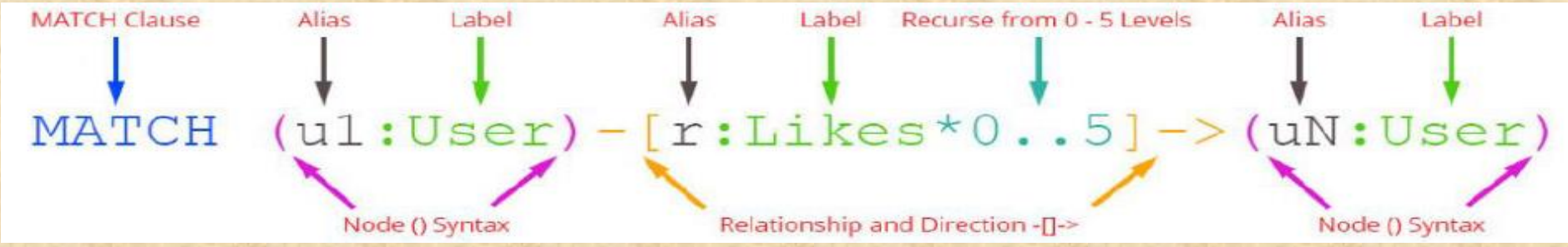
- Size:33 bytes
  - **First byte**:In use flag
  - **Next 8 bytes**:IDs of the nodes at the start and end of the relationship
  - **4 bytes**:Pointer to the relationship type
  - **16 bytes**:pointers for the next and previous relationship records for each of the start and end nodes
  - **4 bytes**:next property id

Cypher Query Langauge (CQL)

- SQL has been the de facto language for RDBMS
- Cypher is a declarative language that serves the same purpose as SQL
- Uses ASCII-Art to represent patterns
- Nodes are surrounded with parentheses
- Use arbitrary variables to refer to nodes
  Variable scope restricted to single statement
- Case Sensitive – standard naming convention
- https://neo4j.com/docs/developer-manual/current/cypher/syntax/naming/

Cypher Query : Syntax

- Relationships are specified using an arrow (- ->) between nodes
- Square bracket inside arrow for specification
  - Relationships - 1 type
  - Nodes - 0 or more labels
- Cypher allows patterns to be assigned to variables that increase modularity and reduce repetition



Cypher Query Clauses

- Minimum/simplest query consist of a **MATCH** clause followed by a **RETURN** clause.

```
MATCH (a:Person {name:'Jim'})-[:KNOWS]->(b)-[:KNOWS]-
>(c), (a)-[:KNOWS]->(c)

RETURN b, c
```

- **WHERE :** Provides criteria for filtering pattern matching results.
- **CREATE and CREATE UNIQUE :** Create nodes and relationships.
- **MERGE :** Ensures that the supplied pattern exists in the graph, either by reusing existing nodes and relationships that match the supplied predicates, or by creating new nodes and relationships

Cypher Query Clauses …

- **DELETE/REMOVE :** Removes nodes, relationships, and properties.
- **SET :** Sets property values and labels
- **ORDER BY :** Sorts results as part of a **RETURN**
- **SKIP LIMIT :** Skip results at the top and limit the number of results
- **FOREACH :** Performs an updating action for each element in a list.
- **UNION :** Merges results from two or more queries.
- **WITH :** Chains subsequent query parts and forwards results from one to the next. Similar to piping commands in Unix.
- For more detail https://neo4j.com/docs/cypher-refcard/current/?ref=beginners-ebook

Getting started

- Multiple ways to start
  - Neo4j Sandboxes (cloud containers)
  - VMs (VirtualBox - Windows & Mac)
  - VMs (Linux - VirtualBox or KVM)
  - **Desktop installation**
  - **Server – single instance**
  - **Clustering** – Enterprise

Neo4j Database Server Installation on Windows Machine

- Pre-requisite : JDK 8.0
- Visit the Neo4j official site using https://neo4j.com/try-neo4j/
- Download Neo4J Community Server Edition
- Download setup/run or **zip file/extract**
- Place the extracted files in a permanent home on your server, for example **D:\neo4j**. The top level directory is referred to as **NEO4J_HOME**.

Installation …

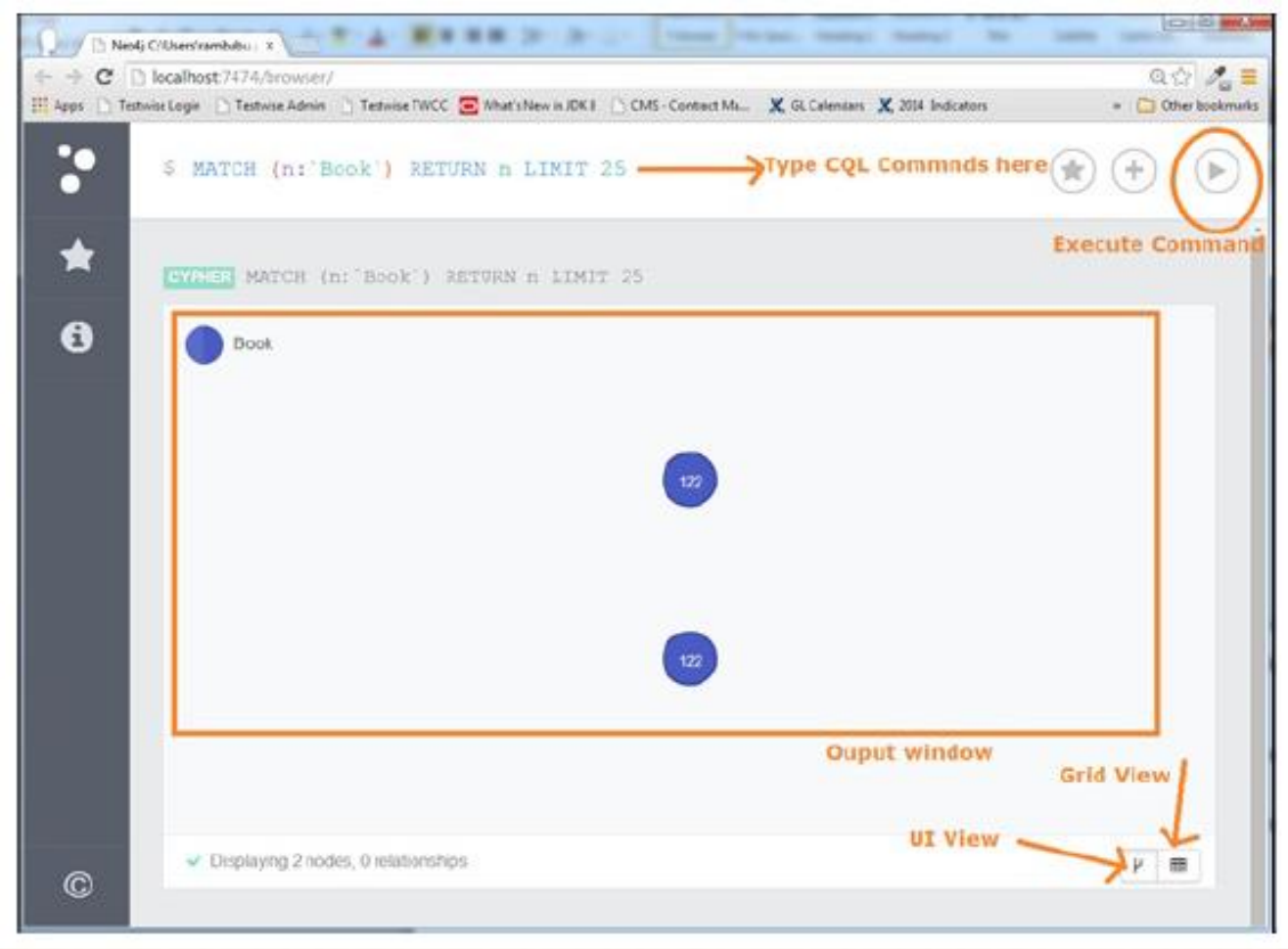
- To run Neo4j as a console application, use:

  <NEO4J_HOME>\bin\neo4j console
- To install Neo4j as a service use:

  <NEO4J_HOME>\bin\neo4j install-service
- Start Neo4J browser : **http://localhost:7474**
- Connect using the **username 'neo4j'** with default **password 'neo4j'**. You'll then be prompted to change the password.
- **$** :play movie graph
- **$** :play northwind graph

Neo4j Data Browser

- Open using URL
  http://localhost:7474/browser/



Remote Access to Data Browser

- Open the `neo4j.conf` file in an editor
- Add following entries in **HTTP Connector** section
- **dbms.connector.http.type=HTTP**
- **dbms.connector.http.enabled=true**
- **dbms.connector.http.address=0.0.0.0:7474**
- **Instead of 0.0.0.0 , put here actual IP**
- **E.g. 10.10.7.101**
- **dbms.connector.http.address=10.10.7.101:7474**

Application Developments

- Connecting through programming languages
- Neo4j officially supported drivers
  - Java
  - Javascript
  - **C#**
  - **Python**

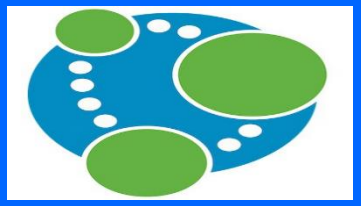
Neo4j for C#.NET Developers

- PM> Install-Package **Neo4j.Driver-4.2.0**
- Neo4j Community Drivers
- **Neo4jClient :** A .NET client for Neo4j, which makes it easy to write Cypher queries in C# with IntelliSense
- GitHub Link : https://github.com/DotNet4Neo4j/neo4jclient

Neo4j Python Driver

- Find out / download the latest version of the driver at **https://pypi.python.org/pypi/neo4j-driver**
- Or Install the latest version of the driver if you are online :

pip install neo4j



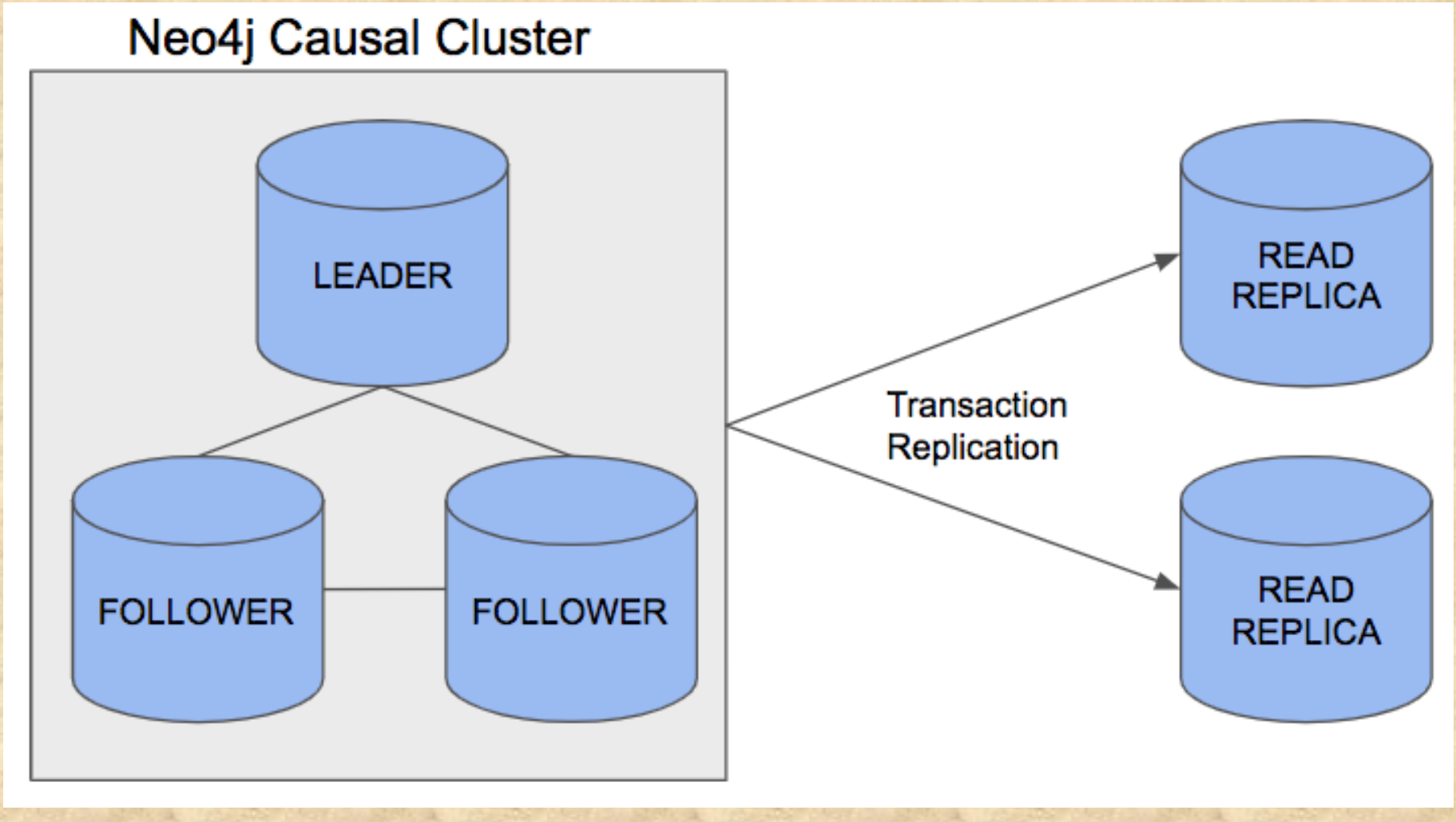
Neo4j

Causal Clustering

Neo4j Causal Clustering

- A cluster is composed of three or more Neo4j instances that communicate with one another to provide fault-tolerance and high-availability
- Uses a **consensus protocol (RAFT)** to coordinate the cluster
- each database has a perfect, complete copy of the entire database (no partitioning)
- Each machine in the cluster has a "**role**" –
  - **Leader or**
  - **Follower**

Cluster Architecture



Cluster Roles

- The **leader** is responsible for coordinating the cluster and accepting all writes
- **Followers** help scale the read workload ability of the cluster and provide for high-availability of data
- If any one follower fails, show continue
- can have any number caches in the form of read replicas

Topology changes

- In the lifecycle of a cluster, cluster roles are temporary.
- Suppose you have machines A, B, and C.
- If A fails, then the remaining nodes (B and C) will elect a new leader amongst themselves.
- When A restarts, later on, it will rejoin the cluster, but probably as a follower.
- Roles can change through the lifecycle of the cluster
- Role changes are not a cause for concern

Driver API consists of 4 key parts

Driver

Top-level object for all Neo4j interaction

Session

Logical context for sequence of transactions

Transaction

Unit of work

Statement Result

Stream of records plus metadata

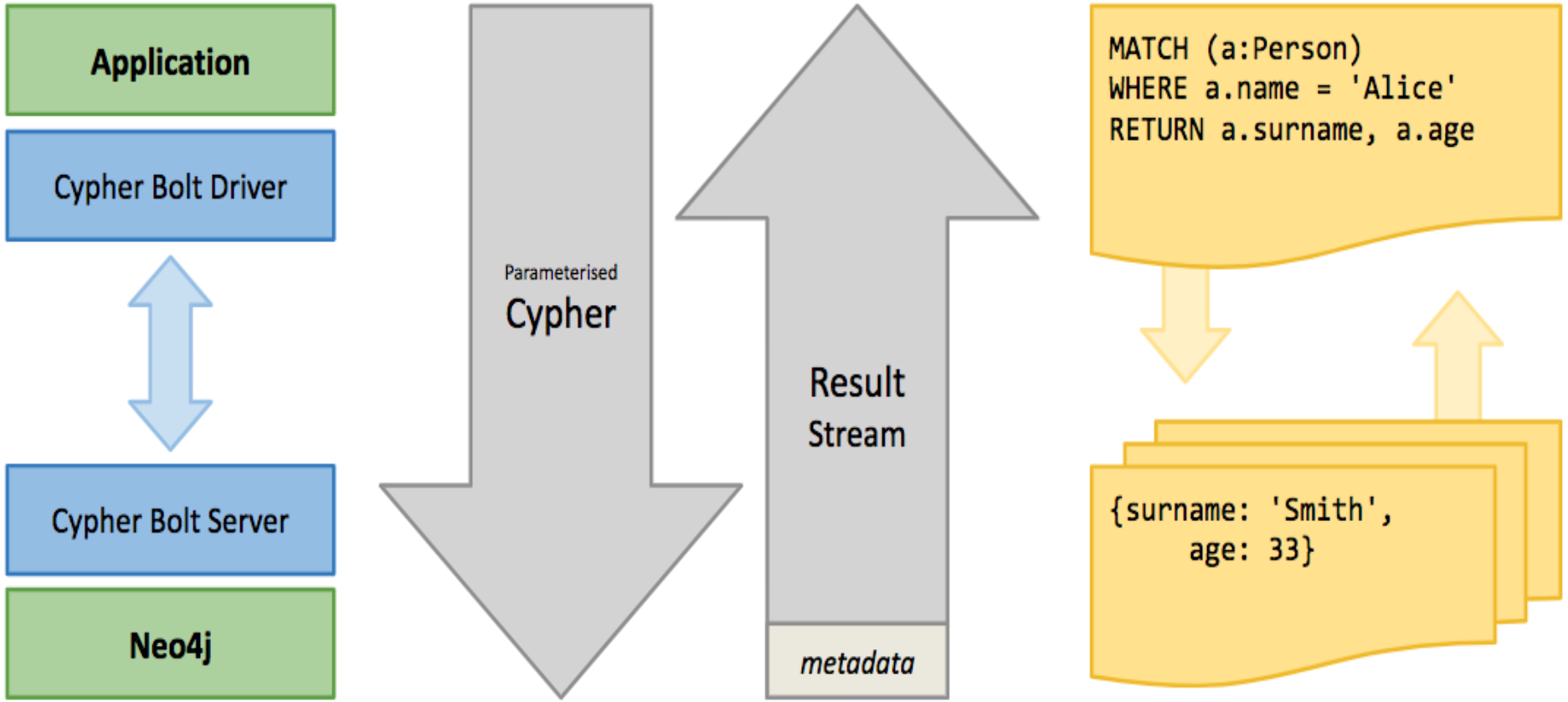
Routing Drivers

- In one of the supported Neo4j drivers (Java, Javascript, Python, .Net and Go)
- there is an option to use the **bolt+routing** protocol
- E.g. URI of connection string

  bolt+routing://neo4j.myhost.com

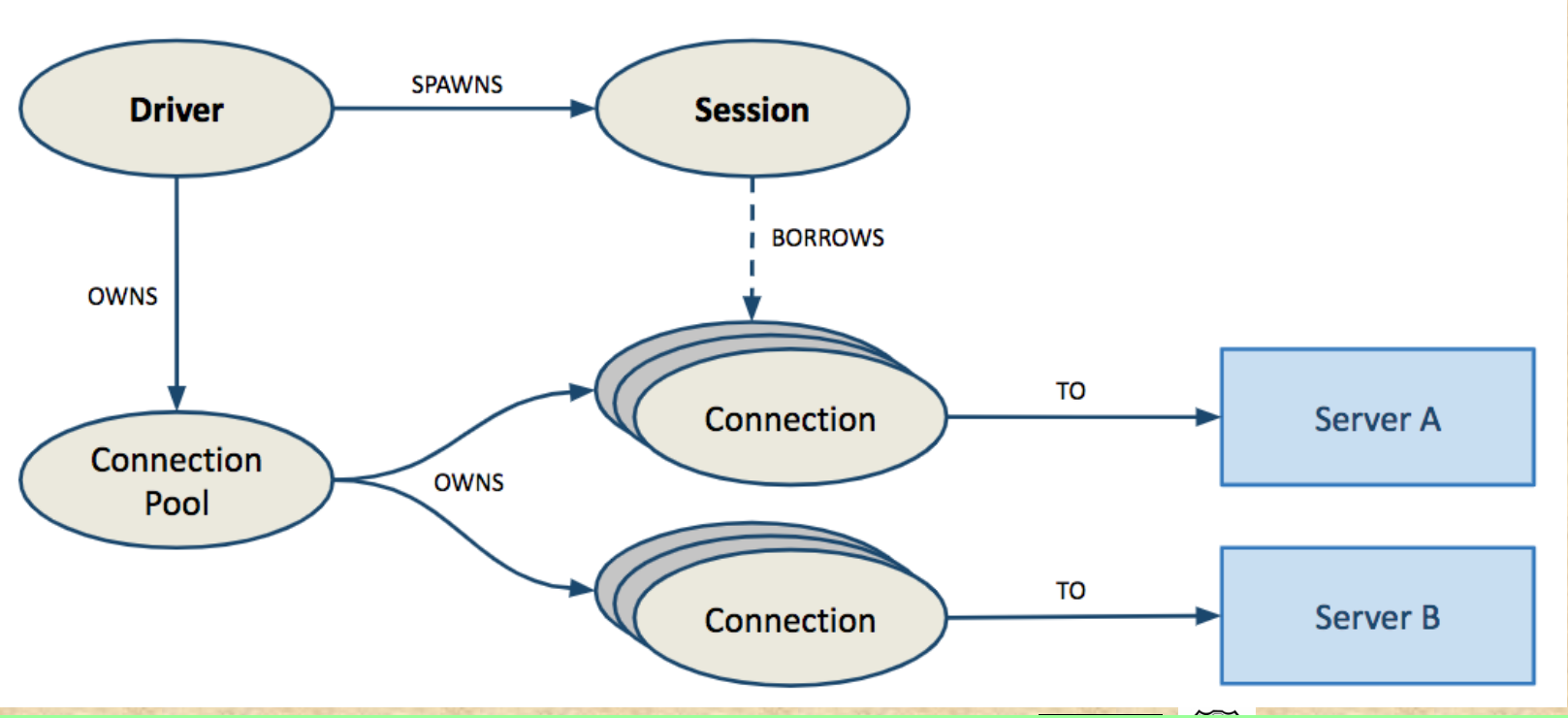
- Routing driver decides how / where to execute the transactions/queries

How queries get run on Neo4j



Connection Management

- driver manages a **pool of connections** to all machines in the cluster
- user needs to just create the sessions, and run the queries from those sessions !



Installation of Neuo4j Causal Cluster on single machine

Steps

1. Download the enterprise server release from the following link

   http://neo4j.com/download/other-releases/#releases

2. Copy the downloaded compressed file in 3 separate directories that effectively creates 3 instances
3. Configure ***neo4j.conf*** file in each instance directory as follows

Modifications in *Neo4j.conf* for each instance

- dbms.backup.enabled=true
- dbms.backup.address=127.0.0.1:6362
- dbms.connector.bolt.address=127.0.0.1:7687
- dbms.connector.http.address=127.0.0.1:**7474**
- dbms.connector.https.address=127.0.0.1:7473
- dbms.mode=HA
- ha.server_id=1
- ha.initial_hosts=127.0.0.1:5001,127.0.0.1:5002,127.0.0.1:5003
- ha.host.coordination=127.0.0.1:5001
- ha.host.data=127.0.0.1:6001

Steps …

4. Start up each Neo4j instance with the following commands

   /Home directory instance1/bin/Neo4j

   /Home directory instance2/bin/Neo4j

   /Home directory instance3/bin/Neo4j

5. Startup Neo4J web based admin console in your Browser

   http://127.0.0.1:7474 (instance one, HTTP)

   http://127.0.0.1:7475 (instance two, HTTP)

   http://127.0.0.1:7476 (instance three, HTTP)

Steps …

6. View the status of your cluster on the monitoring and metrics page of the Neo4J administration console