# Cloud Databases

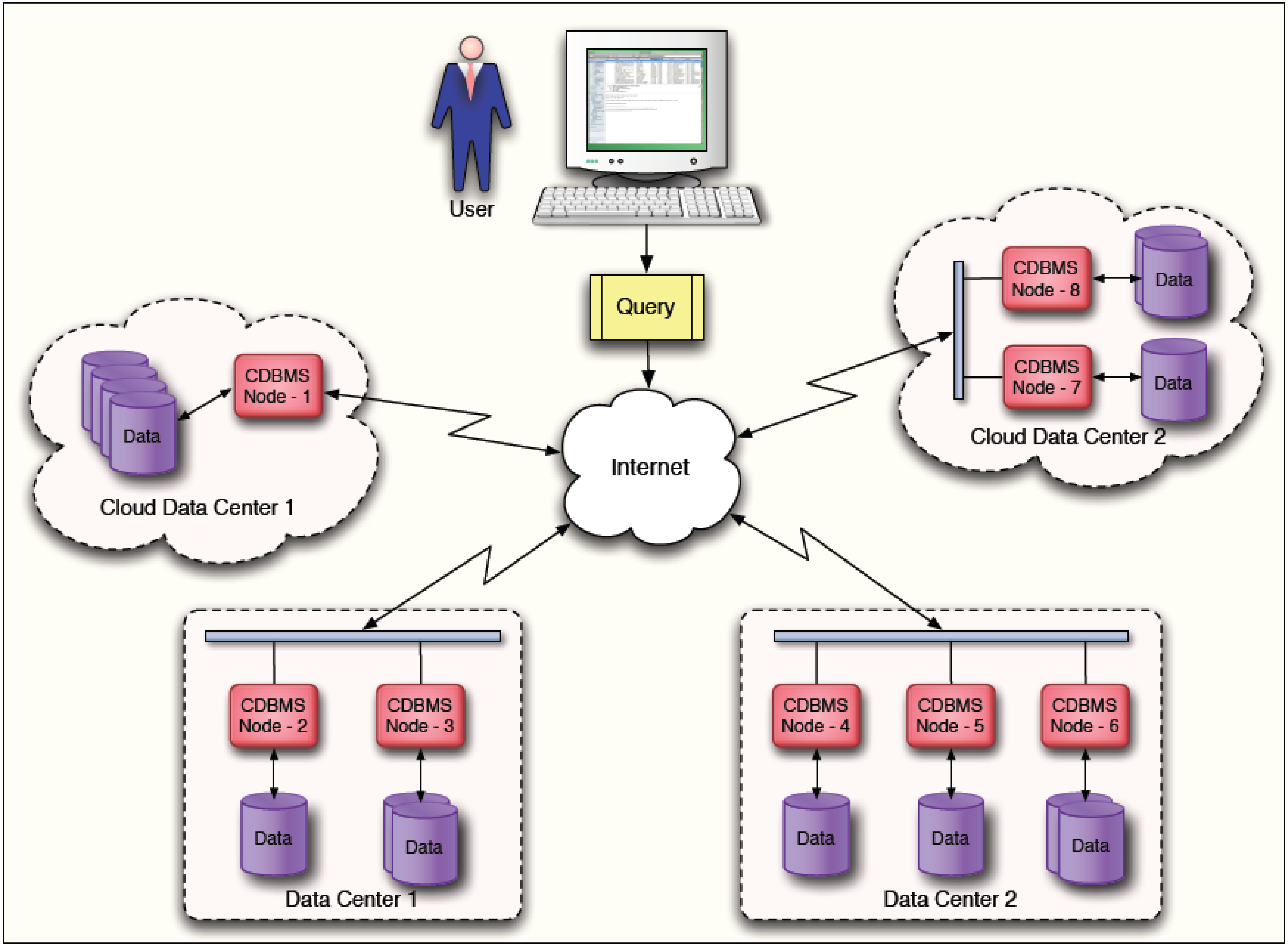
# Introduction

- 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**

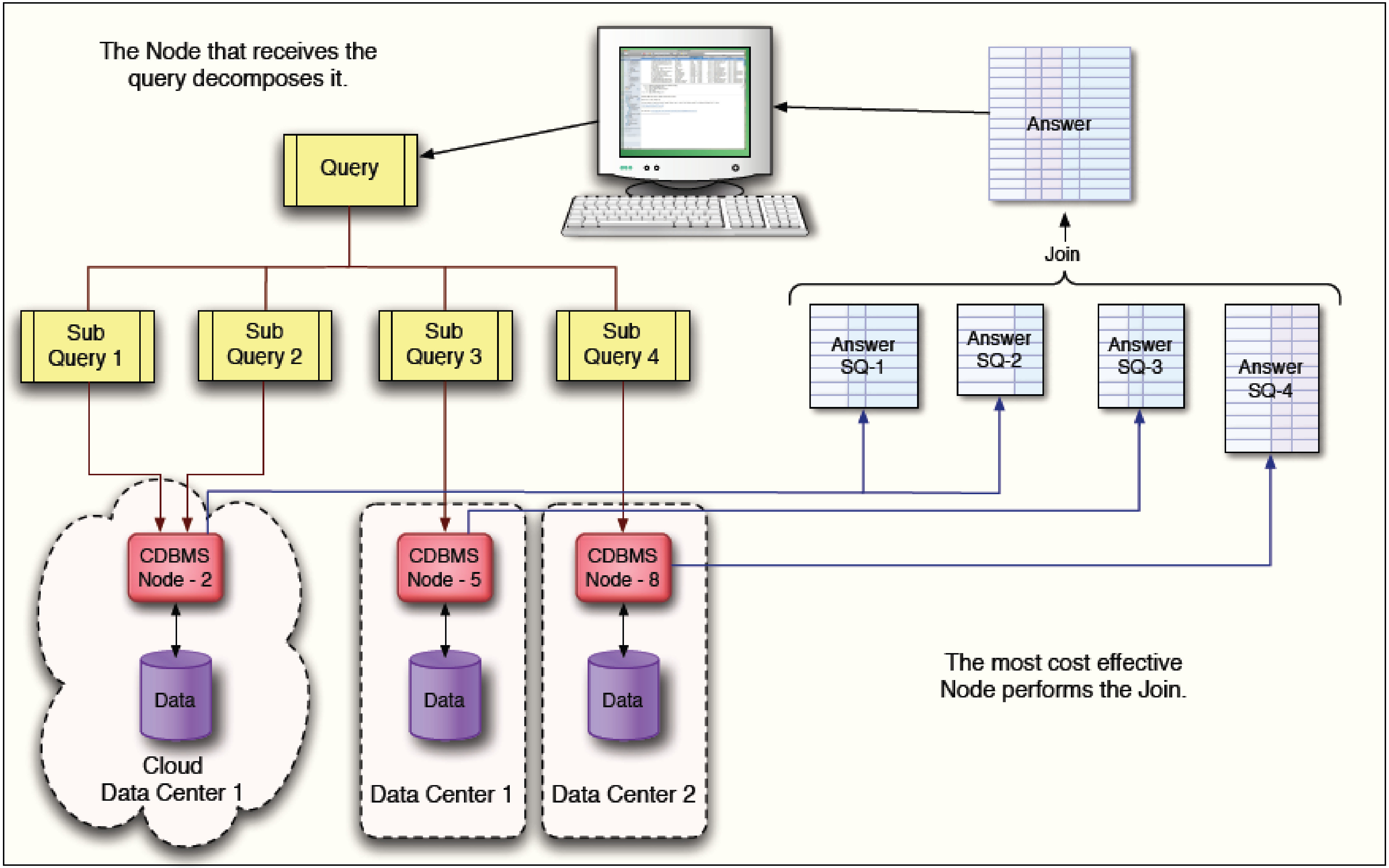
# Definition

**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:**

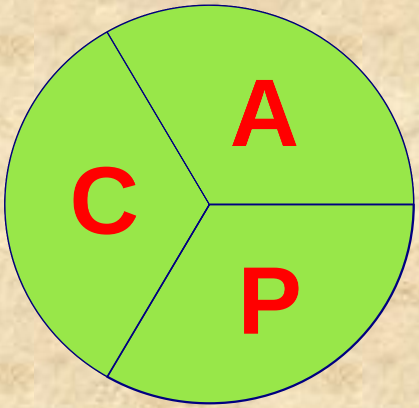
1. Strong consistency – ACID
   (**A**tomicity, **C**onsistency, **I**solation, **D**urability)
   ***do not supported by NoSQL***

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

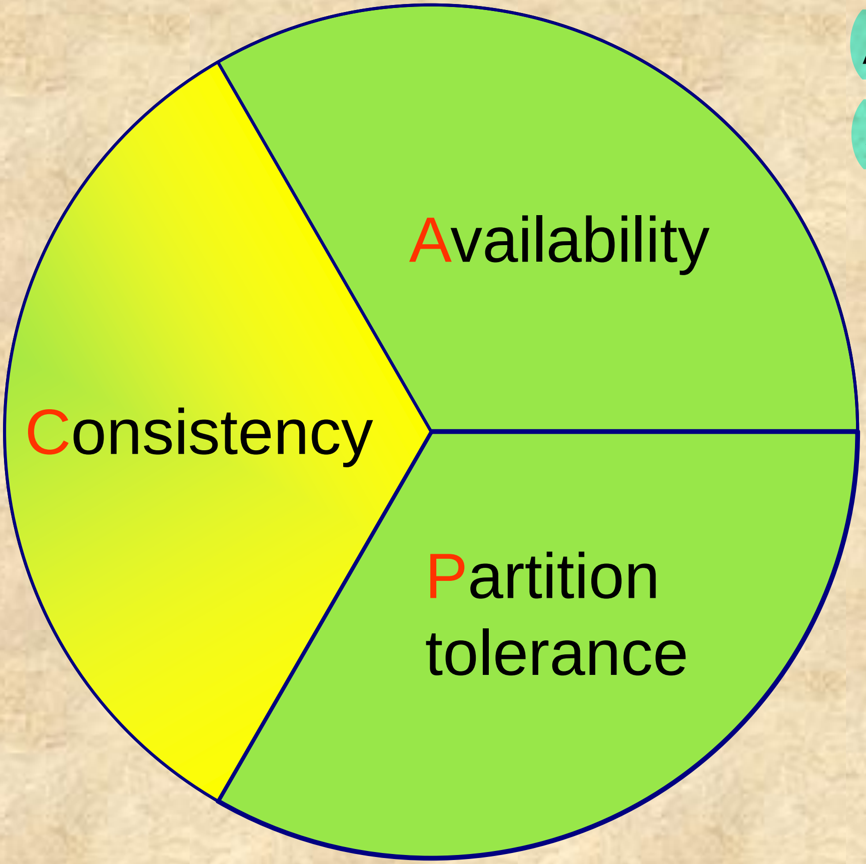
***Based on CAP Theorem***

# *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

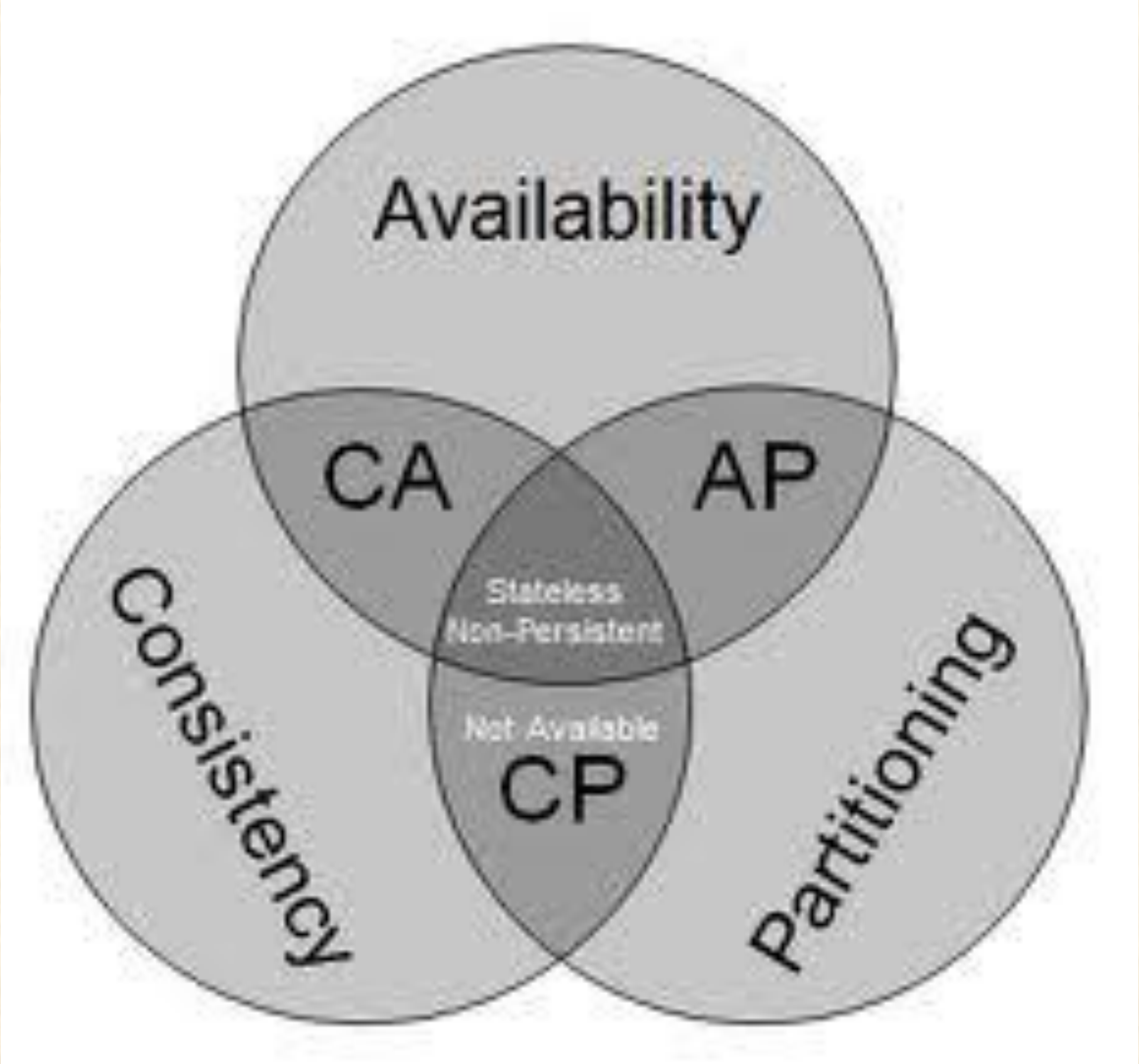


All client always have the same view of the data

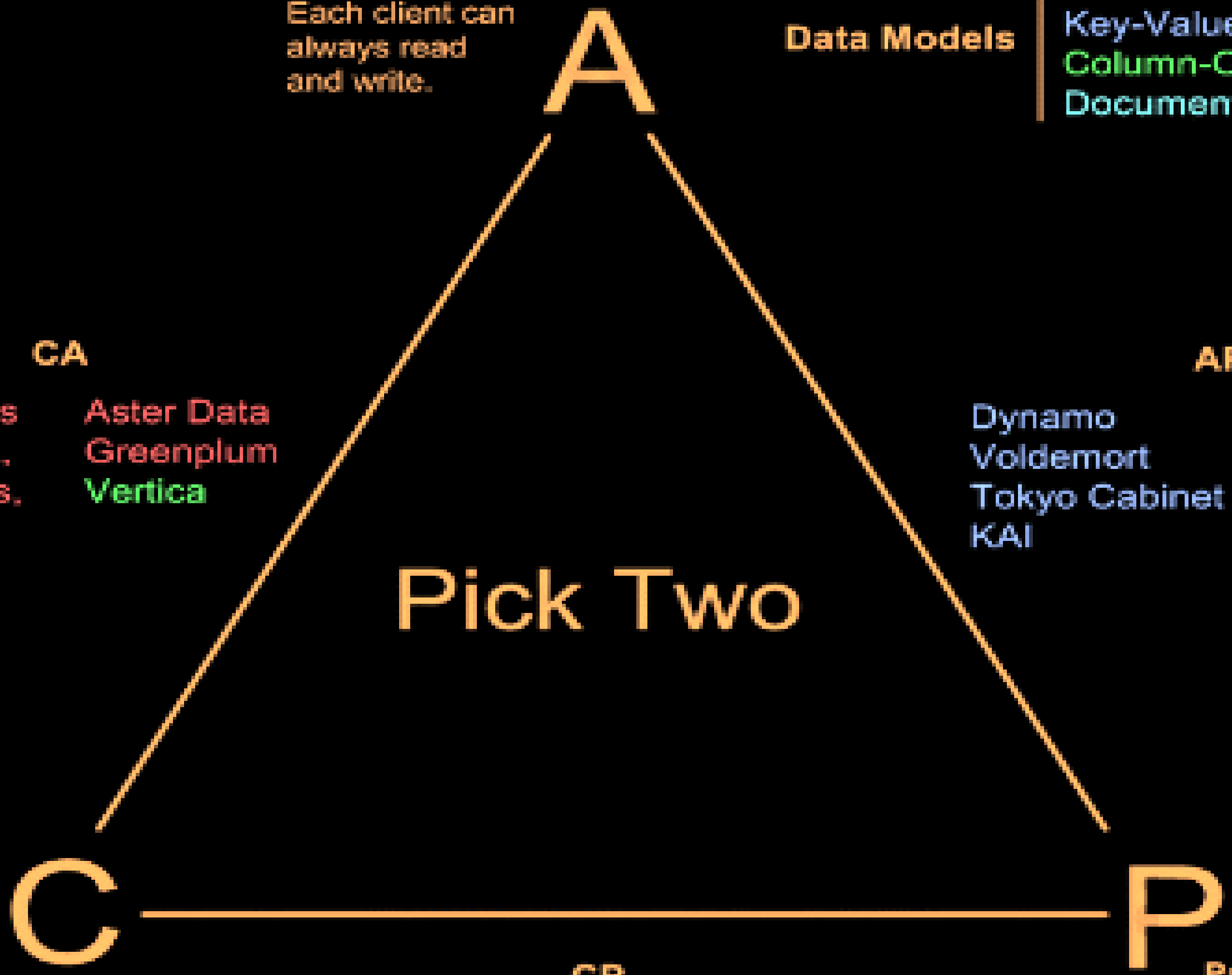
# Brewer's CAP Theorem:

- *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



A

C

P

**Availability:**
Each client can
always read
and write.

**Data Models**

- Relational (comparison)
- Key-Value
- Column-Oriented/Tabular
- Document-Oriented

**CA**

RDBMSs (MySQL, Postgres, etc)

Aster Data
Greenplum
Vertica

**AP**

Dynamo
Voldemort
Tokyo Cabinet
KAI

Cassandra
SimpleDB
CouchDB
Riak

Pick Two

**Consistency:**
All clients always
have the same view
of the data.

**CP**

BigTable
Hypertable
Hbase

MongoDB
Terrastore
Scalaris

Berkeley DB
MemcacheDB
Redis

**Partition Tolerance:**
The system works
well despite physical
network partitions.

# 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

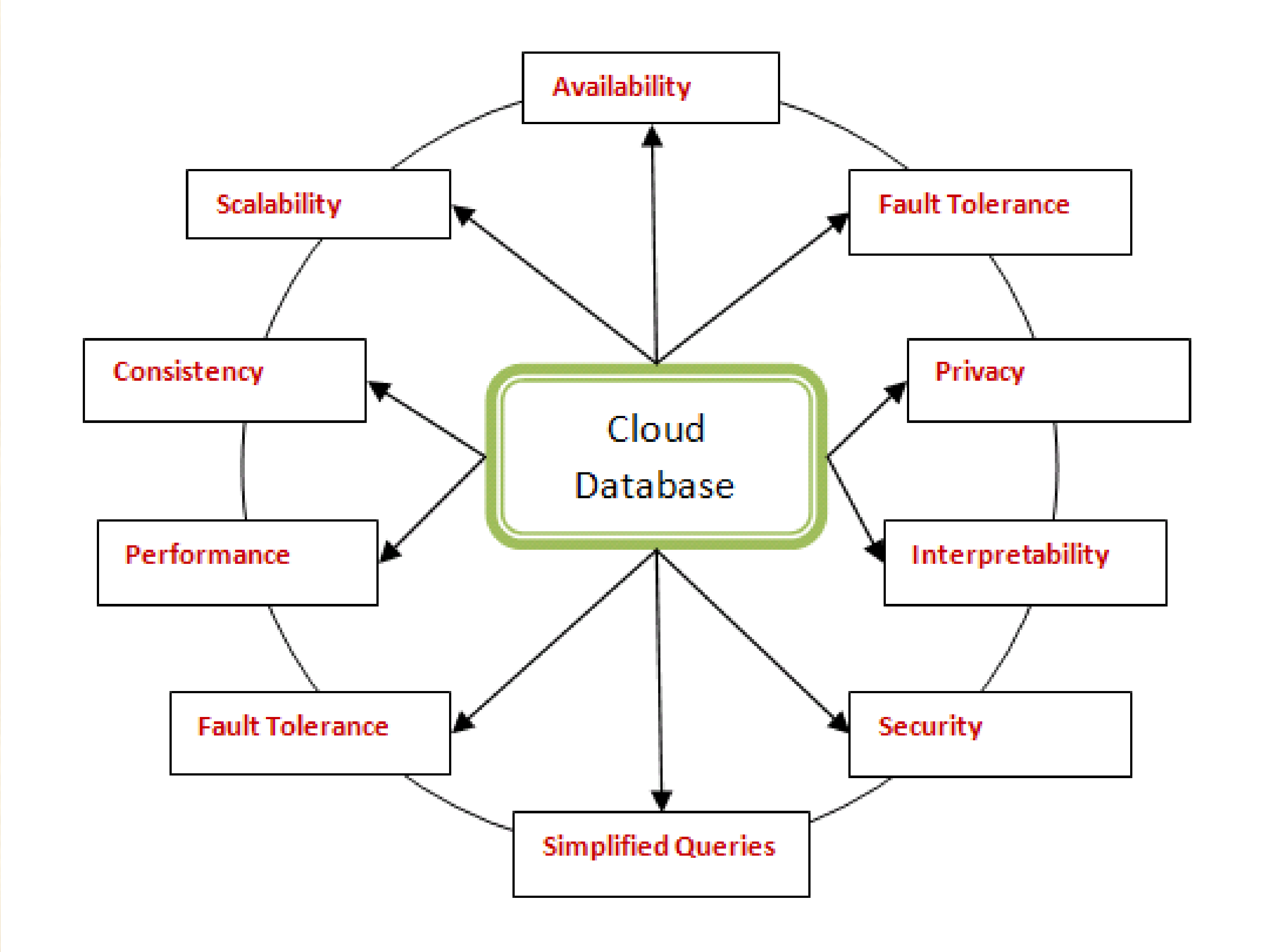
# CouchDB

# 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


Availability
Scalability
Fault Tolerance
Consistency
Cloud Database
Privacy
Performance
Interpretability
Fault Tolerance
Security
Simplified Queries