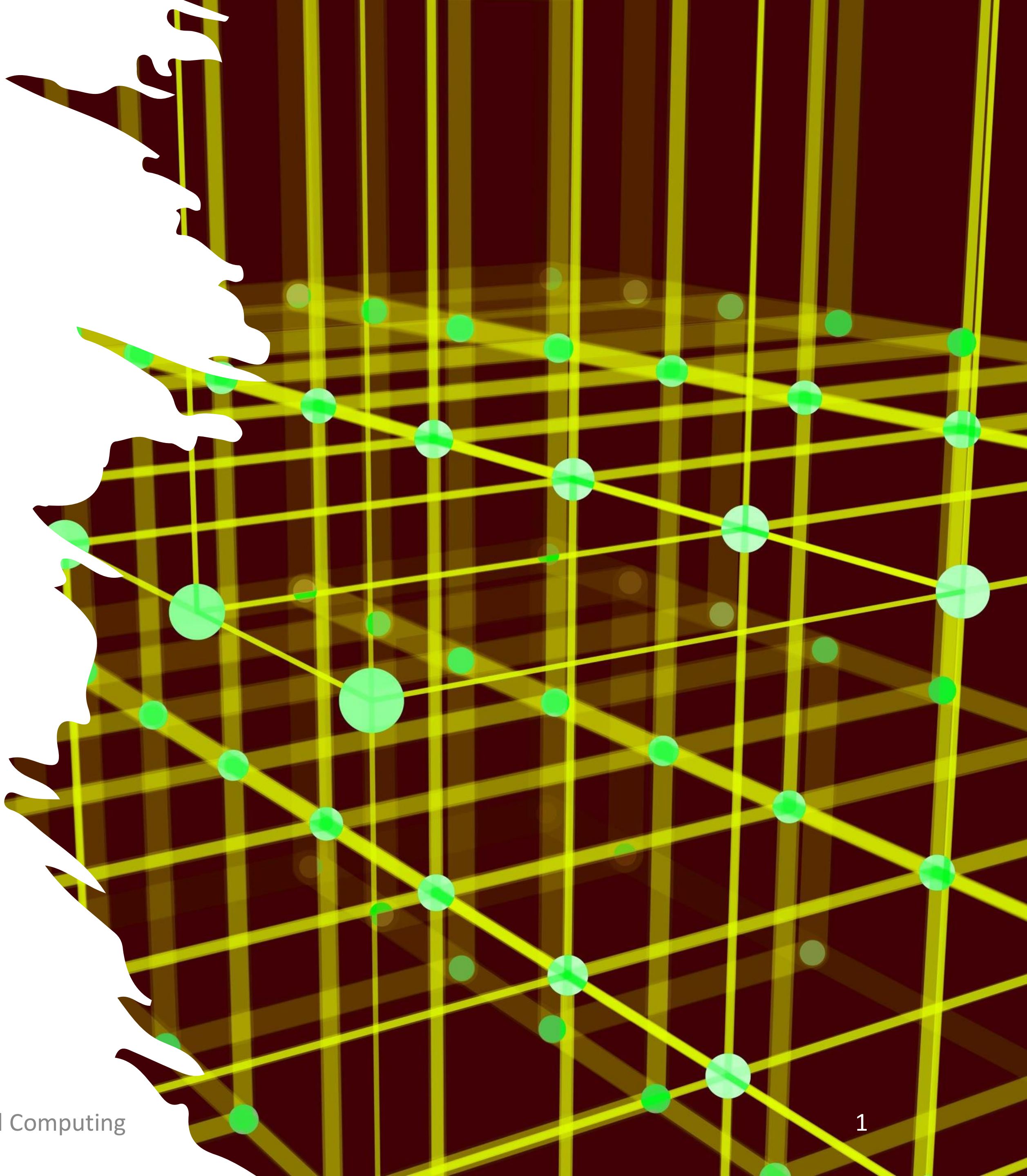
Cloud Databases



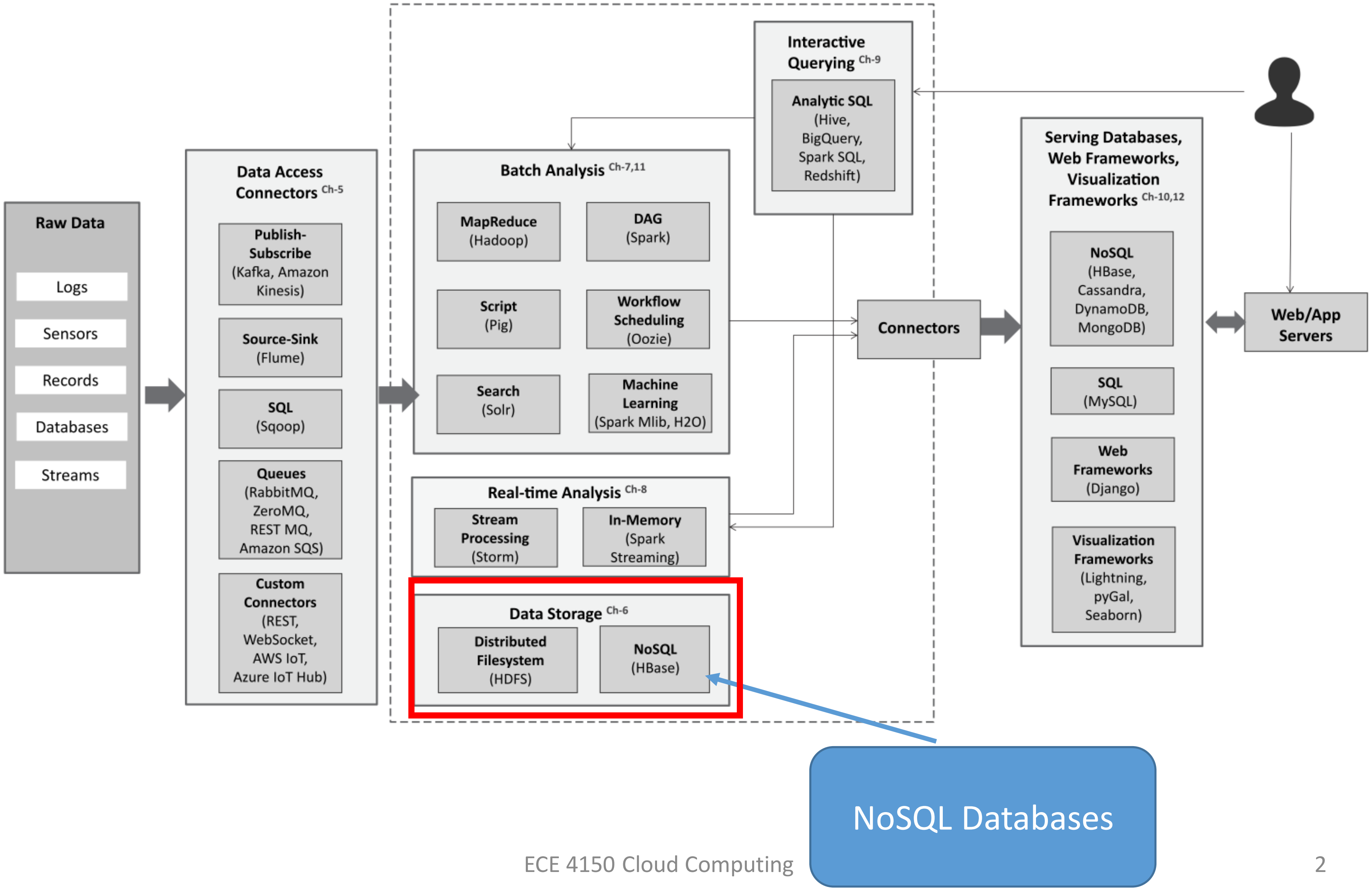
ECE 4150 – Spring 2024

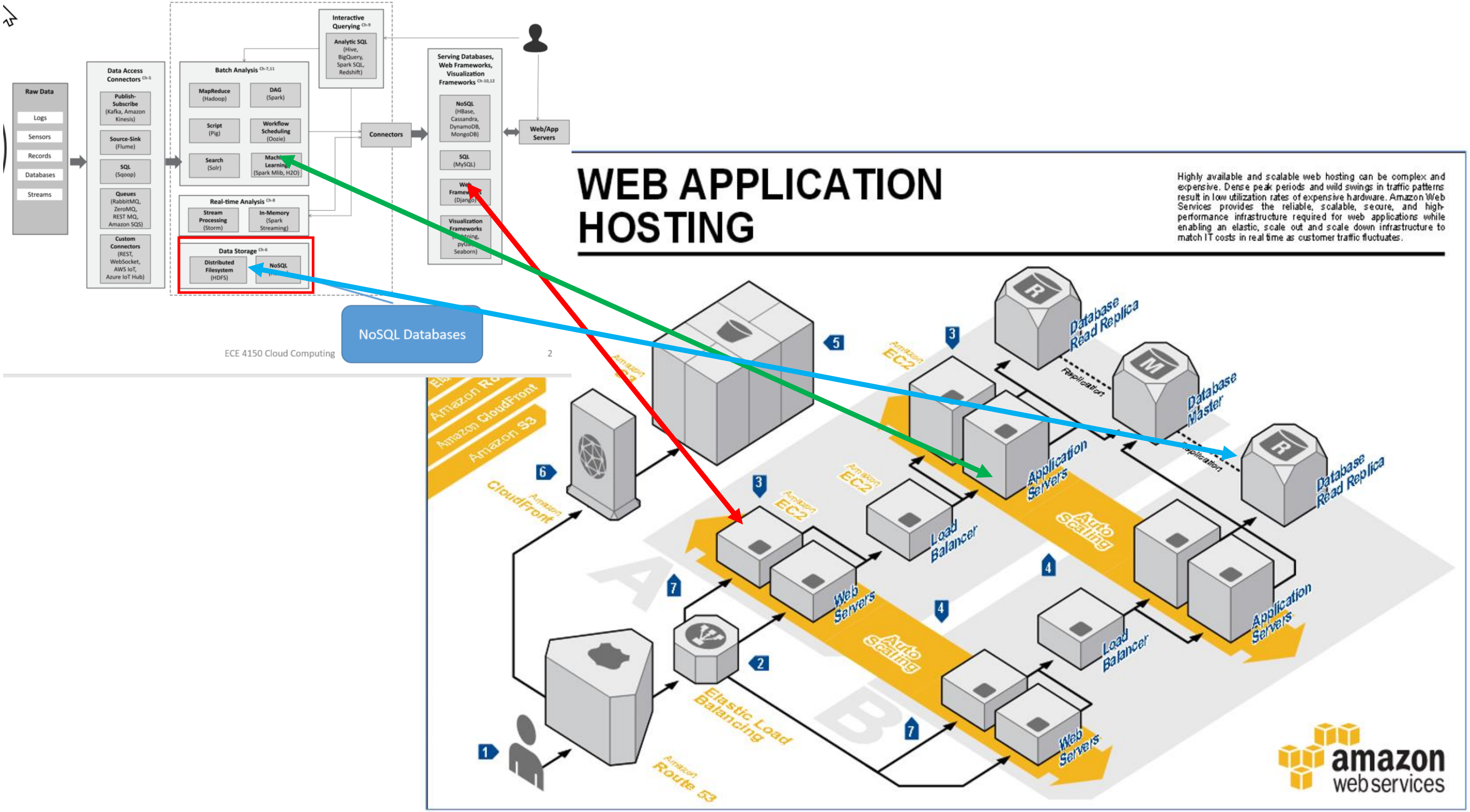
Cloud Computing

Vijay Madisetti

Big Data Stack

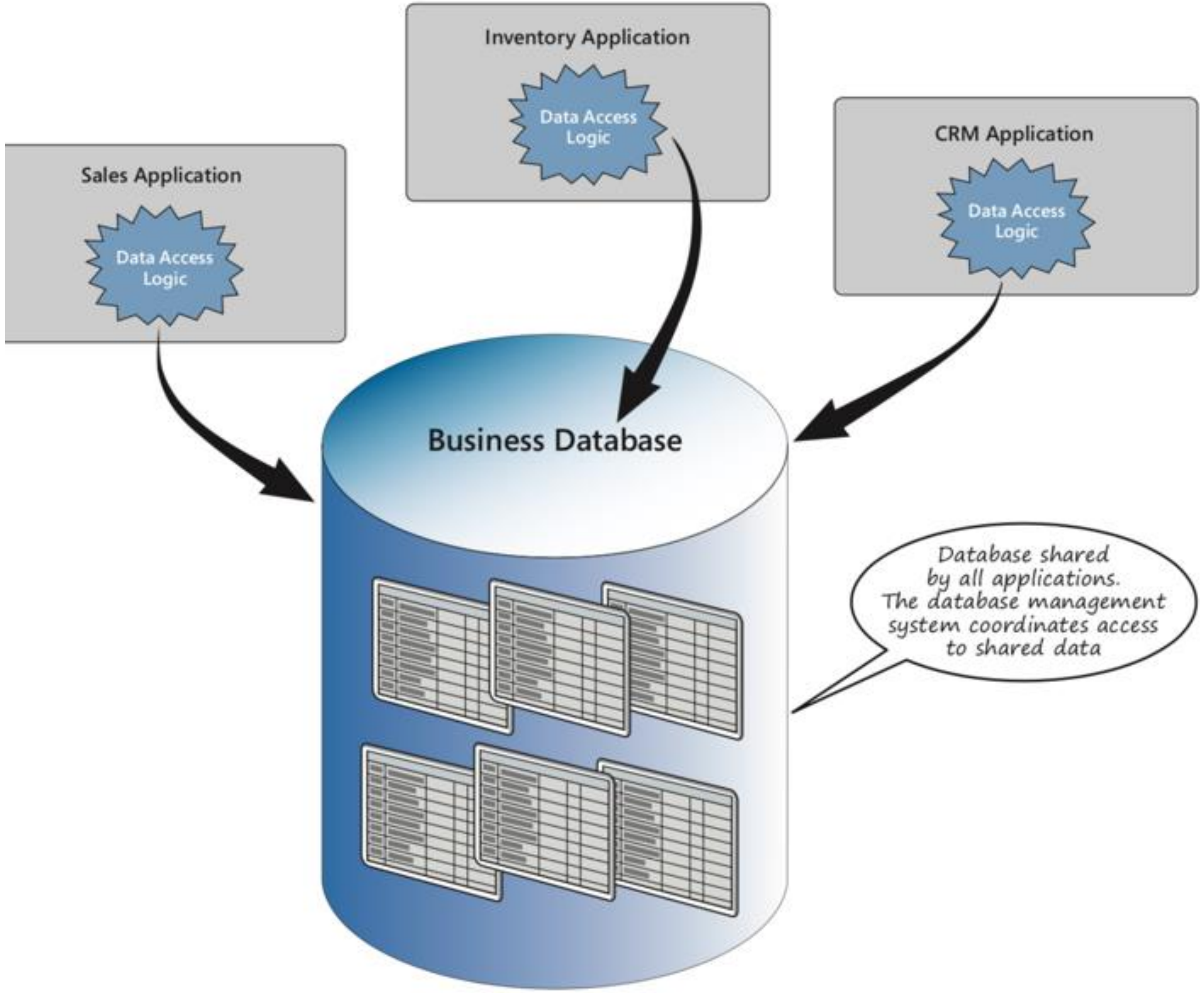
Cloud Computing Applications Architecture





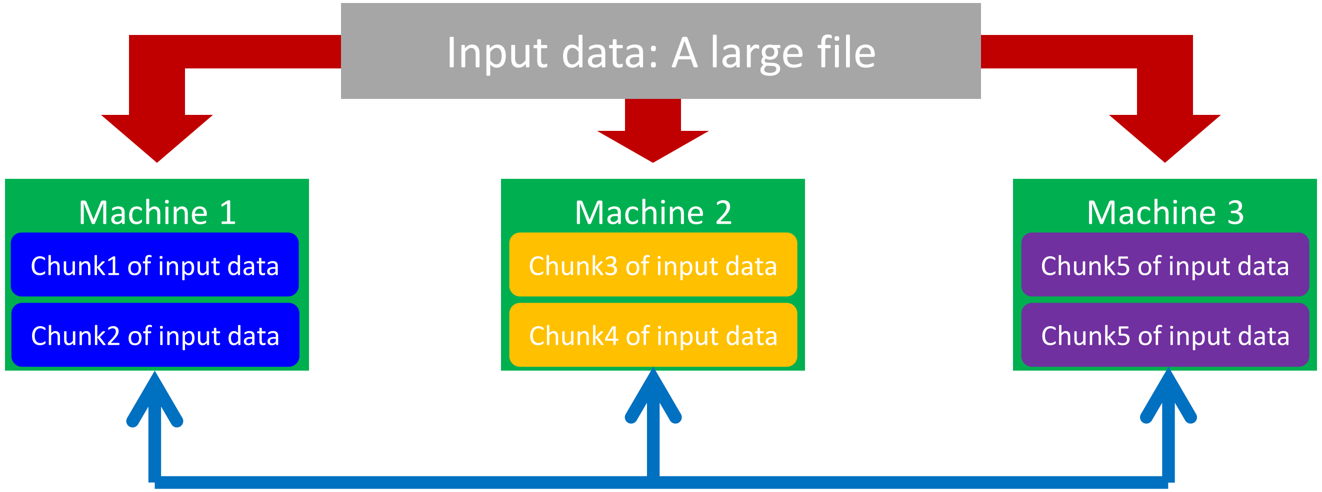
Raw Data
Logs
Sensors
Records
Databases
Streams
Data Access Connectors Ch-5
Publish-Subscribe (Kafka, Amazon Kinesis)
Source-Sink (Flume)
SQL (Sqoop)
Queues (RabbitMQ, ZeroMQ, REST MQ, Amazon SQS)
Custom Connectors (REST, WebSocket, AWS IoT, Azure IoT Hub)
Batch Analysis Ch-7,11
MapReduce (Hadoop)
DAG (Spark)
Script (Pig)
Workflow Scheduling (Oozie)
Search (Solr)
Real-time Analysis Ch-8
Stream Processing (Storm)
In-Memory (Spark Streaming)
Data Storage Ch-6
Distributed Filesystem (HDFS)
NoSQL
Interactive Querying Ch-9
Analytic SQL (Hive, BigQuery, Spark SQL, Redshift)
Connectors
Serving Databases, Web Frameworks, Visualization Frameworks Ch-10,12
NoSQL (HBase, Cassandra, DynamoDB, MongoDB)
SQL (MySQL)
Visualization Frameworks
Web/App Servers
NoSQL Databases
ECE 4150 Cloud Computing
2
WEB APPLICATION HOSTING
Highly available and scalable web hosting can be complex and expensive. Dense peak periods and wild swings in traffic patterns result in low utilization rates of expensive hardware. Amazon Web Services provides the reliable, scalable, secure, and high-performance infrastructure required for web applications while enabling an elastic, scale out and scale down infrastructure to match IT costs in real time as customer traffic fluctuates.
Amazon CloudFront
Amazon S3
Amazon S3
Amazon EC2
Database Read Replica
Database Master
Replication
Application Servers
Auto Scaling
Load Balancer
Web Servers
Elastic Load Balancing
Amazon Route 53
A
B
1
2
3
4
5
6
7
amazon web services

What is a Business Database (BI)



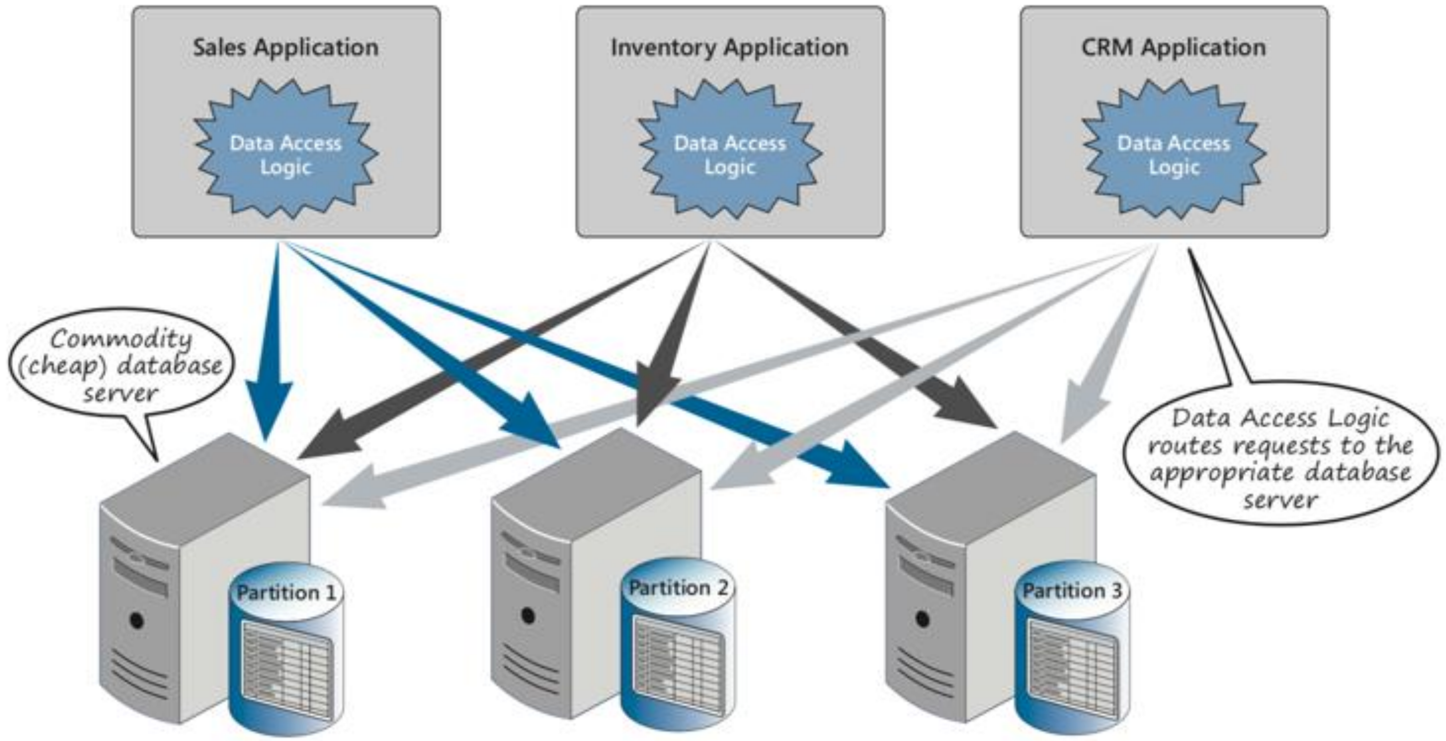
Why Shard Data in Older Relational Databases ?

- Data is typically *sharded* (or *striped*) to allow for concurrent/parallel accesses



E.g., Chunks 1, 3 and 5 can be accessed in parallel

Scaling Performance Using Sharding



Scaling Traditional Databases by Sharding Does not Scale Well

- ***Traditional RDBMSs can be either scaled:***
  - Vertically (or Up)
    - Can be achieved by hardware upgrades (e.g., faster CPU, more memory, or larger disk)
    - Limited by the amount of CPU, RAM and disk that can be configured on a single machine
  - Horizontally (or Out)
    - Can be achieved by adding more machines
    - Requires database *sharding* and probably *replication*
    - Limited by the Read-to-Write ratio and communication overhead

Amdahl's Law

- How much faster will a parallel program run?
  - Suppose that the sequential execution of a program takes T_1 time units and the parallel execution on p processors/machines takes T_p time units
  - Suppose that out of the entire execution of the program, s fraction of it is not parallelizable while $1-s$ fraction is parallelizable
  - Then the speedup (***Amdahl's formula***):

$$\frac{T_1}{T_p} = \frac{T_1}{(T_1 \times s + T_1 \times \frac{1-s}{p})} = \frac{1}{s + \frac{1-s}{p}}$$

Amdahl’s Law: An Example

- Suppose that:
  - 80% of your program can be parallelized
  - 4 machines are used to run your parallel version of the program

- The speedup you can get according to Amdahl’s law is:

$$\frac{1}{s+\frac{1-s}{p}} = \frac{1}{0.2+\frac{0.8}{4}} = 2.5 \text{ times}$$

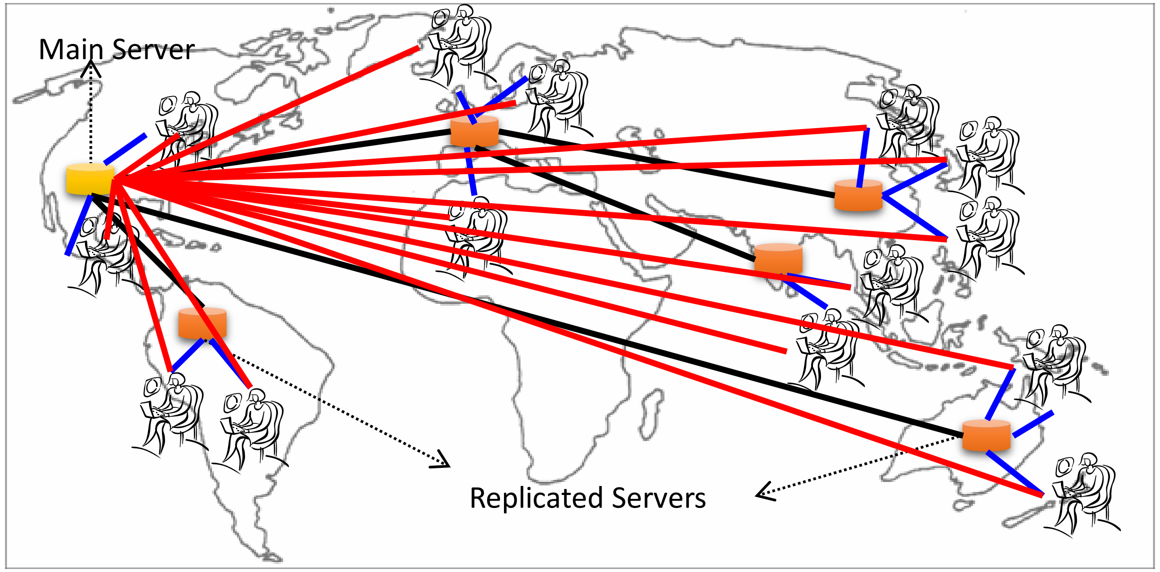
Although you use 4 processors you cannot get a speedup more than 2.5 times!

Why Replicating Data?

- Replicating data across servers helps in:
  - Avoiding performance bottlenecks
  - Avoiding single point of failures
  - And, hence, enhancing scalability and availability

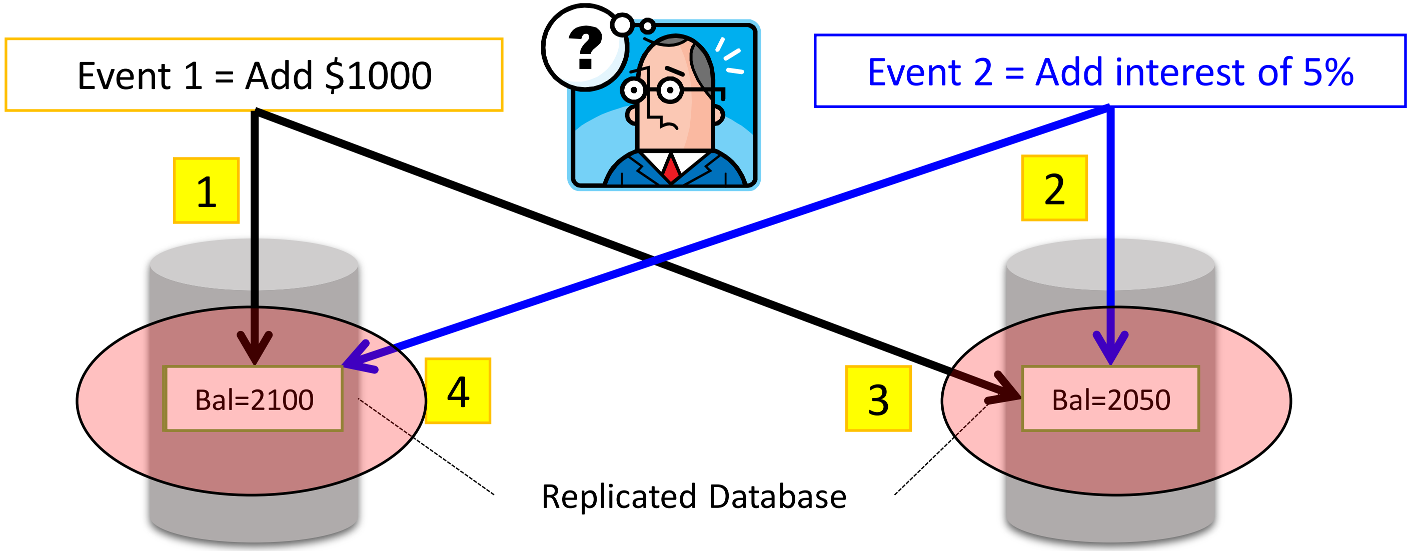
Why Replicate Data?

- Replicating data across servers helps in:
  - Avoiding performance bottlenecks
  - Avoiding single point of failures
  - And, hence, enhancing scalability and availability

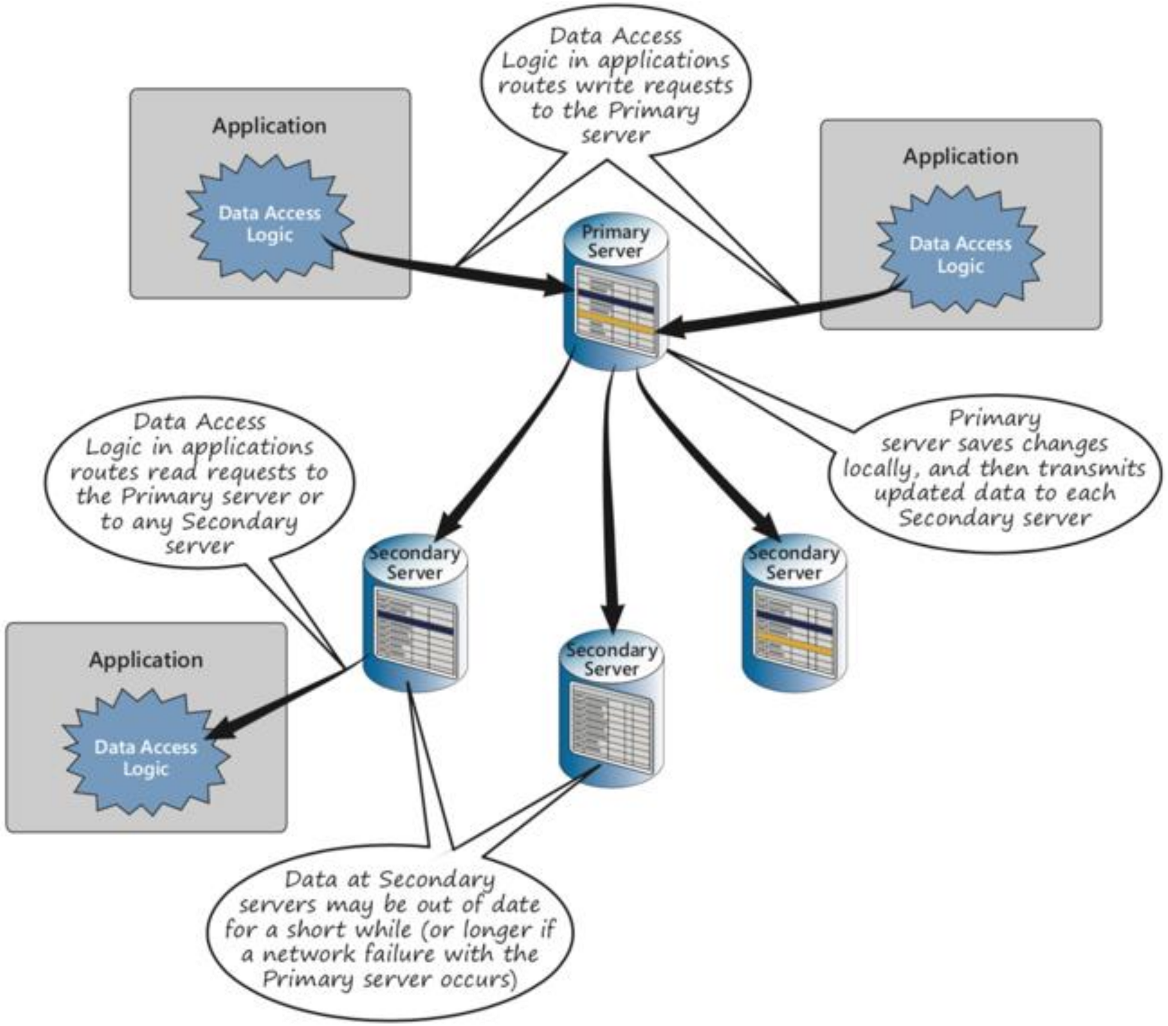


But, *Consistency* Becomes a Challenge

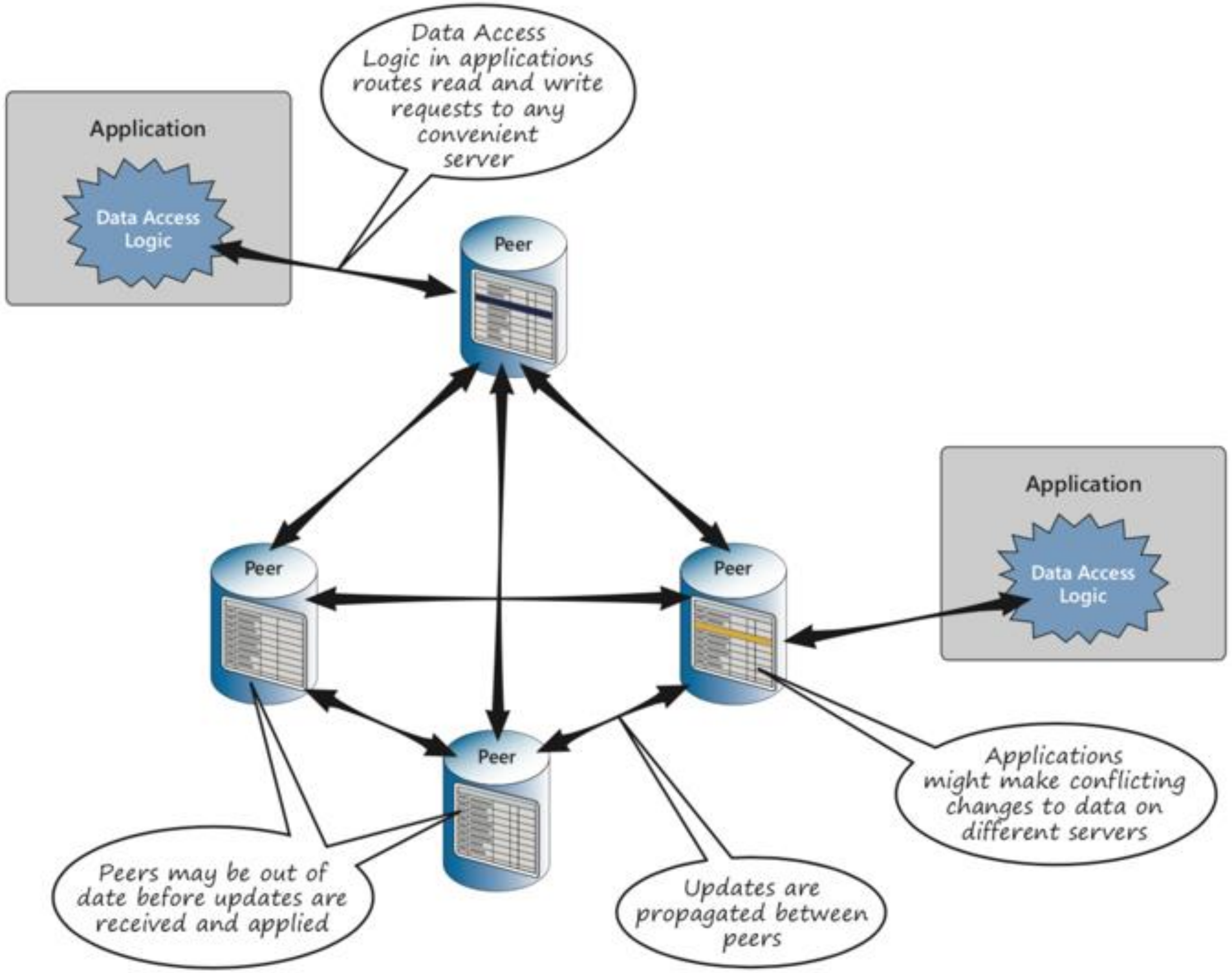
- An example:
  - In an e-commerce application, the bank database has been replicated across two servers
  - Maintaining consistency of replicated data is a challenge



Replication of Databases Using Primary/Secondary Model



Replication of Database Using Peer-to-Peer Model



In computer science, ACID is a set of properties of database transactions.

A - Atomicity

C - Consistency

I - Isolation

D - Durability

ACID (Traditional Databases) versus BASE (Big Data)

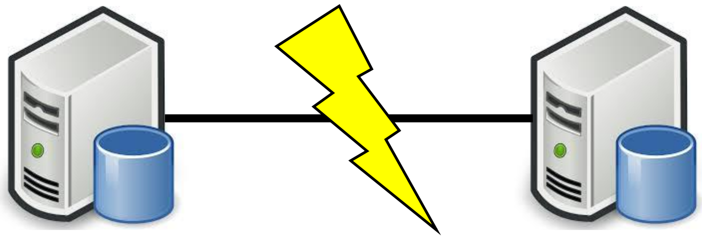
The CAP Theorem for Cloud Computing

- The limitations of distributed databases can be described in the so called the CAP theorem
  - **C**onsistency: every node always sees the same data at any given instance (i.e., strict consistency)
  - **A**vailability: the system continues to operate, even if nodes in a cluster crash, or some hardware or software parts are down due to upgrades
  - **P**artition Tolerance: the system continues to operate in the presence of network partitions

CAP theorem: any distributed database with shared data, can have ***at most two*** of the three desirable properties, C, A or P

The CAP Theorem (*Cont'd*)

- Let us assume two nodes on opposite sides of a network partition:



- Availability + Partition Tolerance <u>forfeits</u> Consistency

- Consistency + Partition Tolerance entails that one side of the partition must act as if it is unavailable, thus forfeiting Availability

- Consistency + Availability is only possible if there is no network partition, thereby forfeiting Partition Tolerance

Large-Scale Databases

Observations

- *When companies such as Google and Amazon were designing large-scale databases, 24/7 Availability was a key*
    - *A few minutes of downtime means lost revenue*
- *When horizontally scaling databases to 1000s of machines, the likelihood of a node or a network failure increases tremendously*
- *Therefore, in order to have strong guarantees on Availability and Partition Tolerance, they had to sacrifice "strict" Consistency (implied by the CAP theorem)*

Trading-Off Consistency

- Maintaining consistency should balance between the strictness of consistency versus availability/scalability
  - Good-enough consistency *depends on your application*

Trading-Off Consistency

- Maintaining consistency should balance between the strictness of consistency versus availability/scalability
    - Good-enough consistency *<u>depends on your application</u>*

Loose Consistency

Strict Consistency

Easier to implement, and is efficient

Generally hard to implement, and is inefficient

The BASE Properties for Cloud Computing

- The CAP theorem proves that it is impossible to guarantee strict Consistency and Availability while being able to tolerate network partitions

- This resulted in databases with relaxed **ACID** guarantees

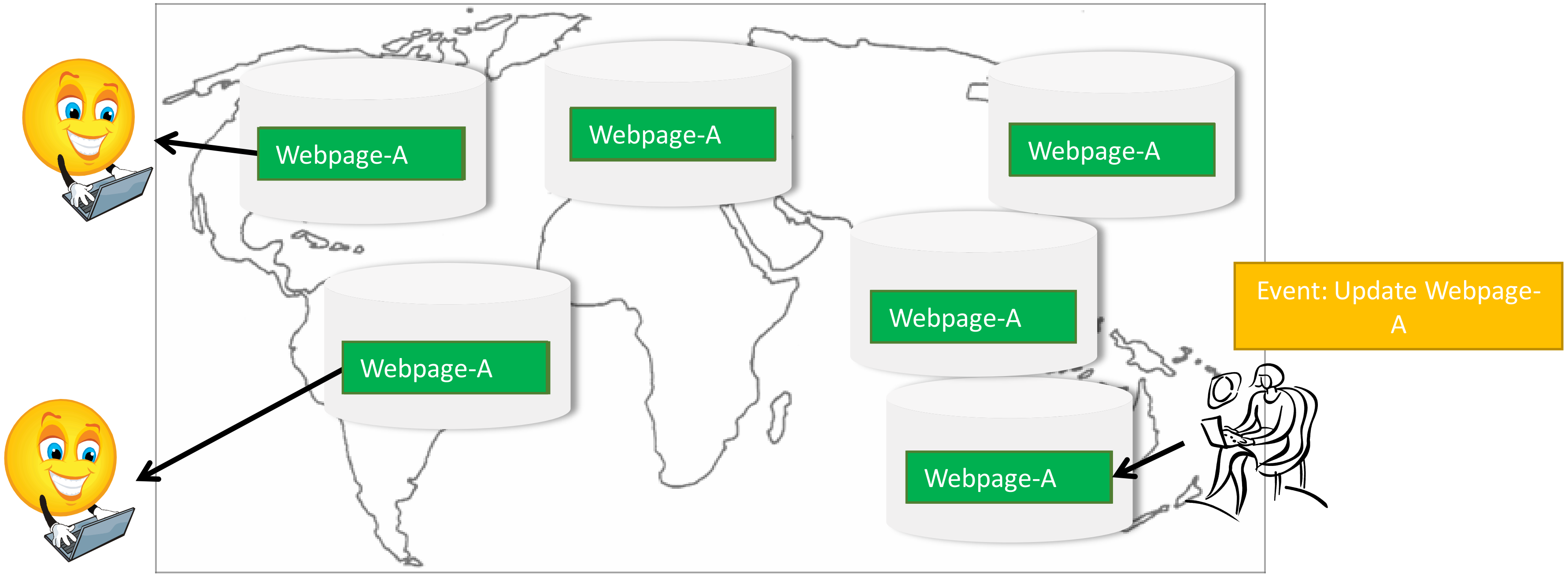
- In particular, such databases apply the BASE properties:
  - **B**asically **A**vailable: the system guarantees Availability
  - **S**oft-State: the state of the system may change over time
  - **E**ventual Consistency: the system will *eventually* become consistent

Eventual Consistency

- A database is termed as *Eventually Consistent* if:
  - All replicas will *gradually* become consistent in the absence of updates

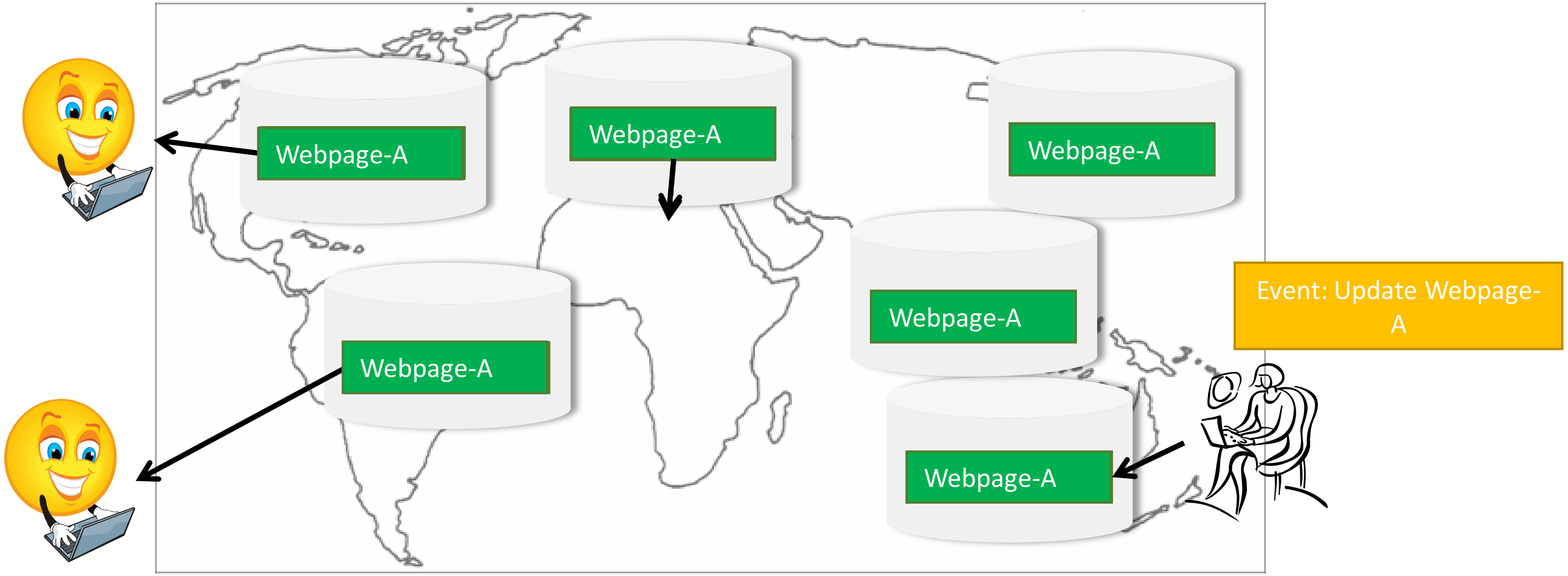
Eventual Consistency

- A database is termed as *Eventually Consistent* if:
  - All replicas will *gradually* become consistent in the absence of updates



Eventual Consistency: A Main Challenge

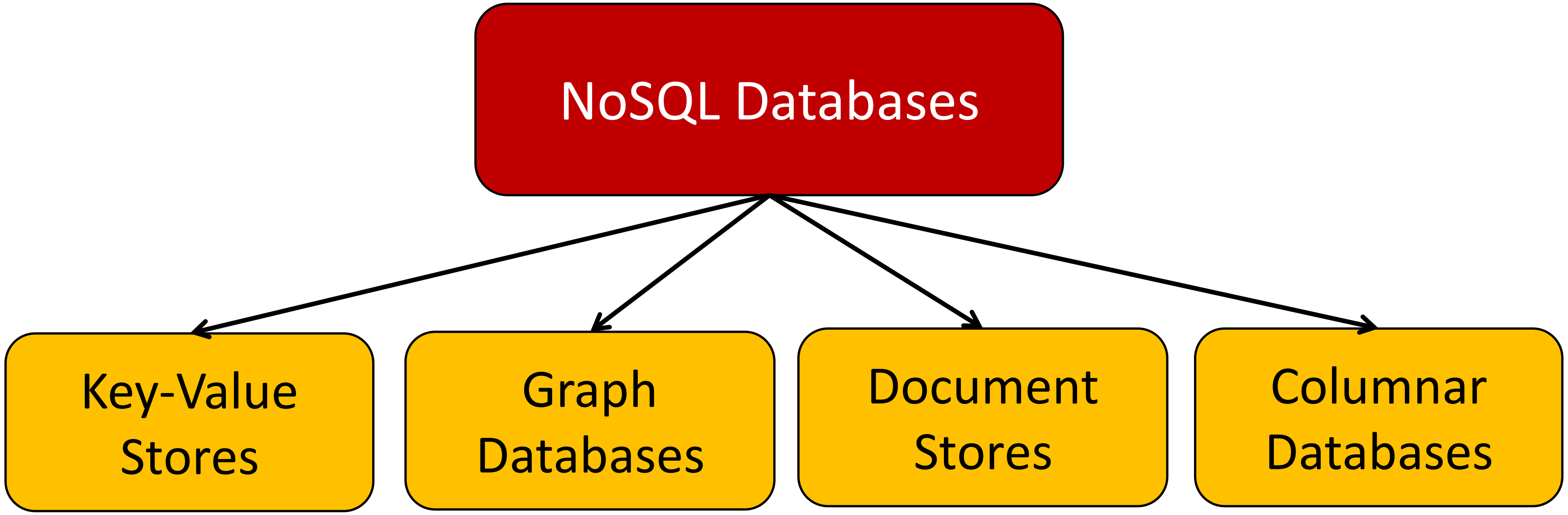
- But, what if the client accesses the data from different replicas?



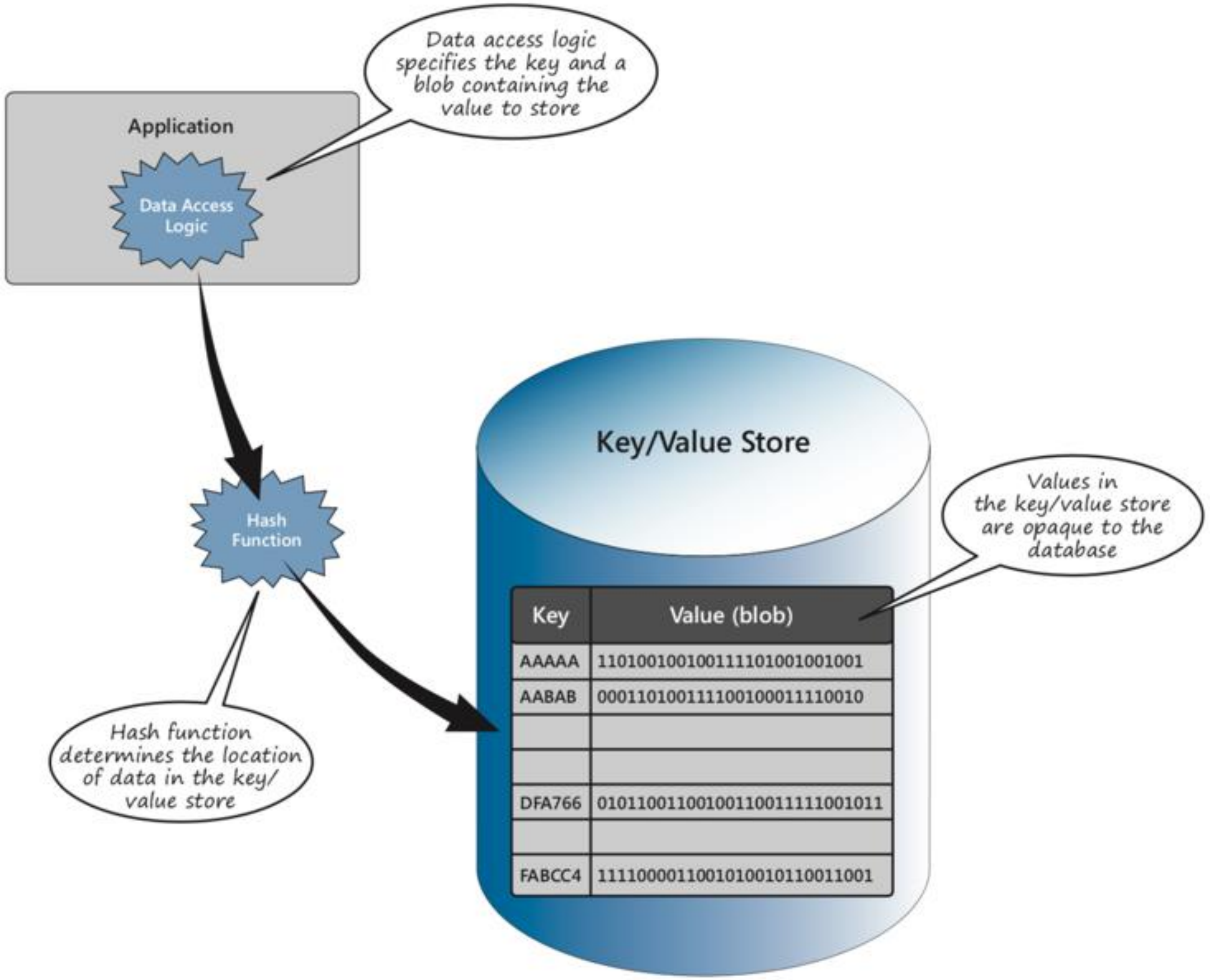
Protocols like Read Your Own Writes (RYOW) can be applied!

Types of NoSQL Databases

- Here is a limited taxonomy of NoSQL databases:



Key/Value Store – Example of NoSQL Database



MongoDB

- Key Value Stores: Used for Real-Time Random Data Access (user session for gaming or Finance)
- Caching mechanism for frequently access data
- Application that uses key-based queries.

Document Database – Another NoSQL Database

Row Key	Document
1001	OrderDate: 06/06/2013 OrderItems: ProductID: 2010 Quantity: 2 Cost: 520 ProductID: 4365 Quantity: 1 Cost: 18 OrderTotal: 1058 Customer ID: 99 ShippingAddress: StreetAddress: 999 500th Ave City: Bellevue State: WA ZipCode: 12345
1002	OrderDate: 07/07/2013 OrderItems: ProductID: 1285 Quantity: 1 Cost: 120 OrderTotal: 120 Customer ID: 220 ShippingAddress: StreetAddress: 888 W. Front St City: Boise State: ID ZipCode: 54321

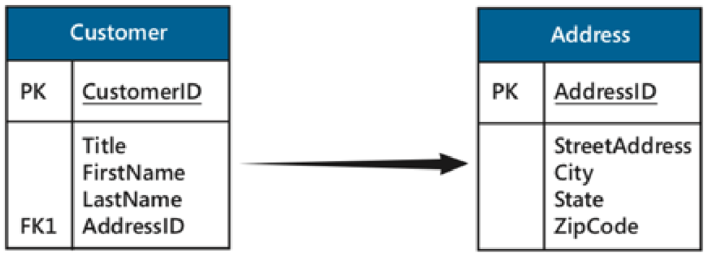
MongoDB

```
{
    "_id": "sammyshark",
    "firstName": "Sammy",
    "lastName": "Shark",
    "email": "sammy.shark@digitalocean.com",
    "department": "Finance"
}
```

```
{
    "_id": "tomjohnson",
    "firstName": "Tom",
    "middleName": "William",
    "lastName": "Johnson",
    "email": "tom.johnson@digitalocean.com",
    "department": ["Finance", "Accounting"]
}
```

```
{
    "_id": "tomjohnson",
    "firstName": "Tom",
    "middleName": "William",
    "lastName": "Johnson",
    "email": "tom.johnson@digitalocean.com",
    "department": ["Finance", "Accounting"],
    "socialMediaAccounts": [
        {
            "type": "facebook",
            "username": "tom_william_johnson_23"
        },
        {
            "type": "twitter",
            "username": "@tomwilliamjohnson23"
        }
    ]
}
```

Column Family Database – Another NoSQL Database



Customer Table

CustomerID	Title	FirstName	LastName	AddressID
1	Mr	Mark	Hanson	500
2	Ms	Lisa	Andrews	501
3	Mr	Walter	Harp	500

Address Table

AddressID	StreetAddress	City	State	ZipCode
500	999 500th Ave	Bellevue	WA	12345
501	888 W. Front St	Boise	ID	54321

Relational Databases

Row

Row Key	Column Families			
CustomerID	CustomerInfo		AddressInfo	
1	CustomerInfo:Title CustomerInfo:FirstName CustomerInfo:LastName	Mr Mark Hanson	AddressInfo:StreetAddress AddressInfo:City AddressInfo:State AddressInfo:ZipCode	999 500th Ave Bellevue WA 12345
2	CustomerInfo:Title CustomerInfo:FirstName CustomerInfo:LastName	Ms Lisa Andrews	AddressInfo:StreetAddress AddressInfo:City AddressInfo:State AddressInfo:ZipCode	888 W. Front St Boise ID 54321
3	CustomerInfo:Title CustomerInfo:FirstName CustomerInfo:LastName	Mr Walter Harp	AddressInfo:StreetAddress AddressInfo:City AddressInfo:State AddressInfo:ZipCode	999 500th Ave Bellevue WA 12345

Column Family Database

Aggregate

Column Family Database (Example – Cassandra)

Row-oriented

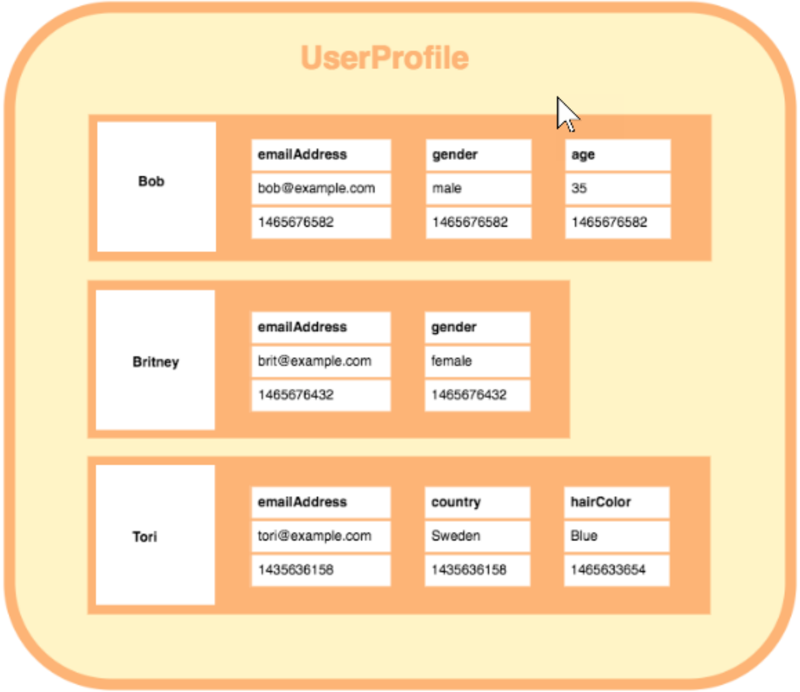
ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

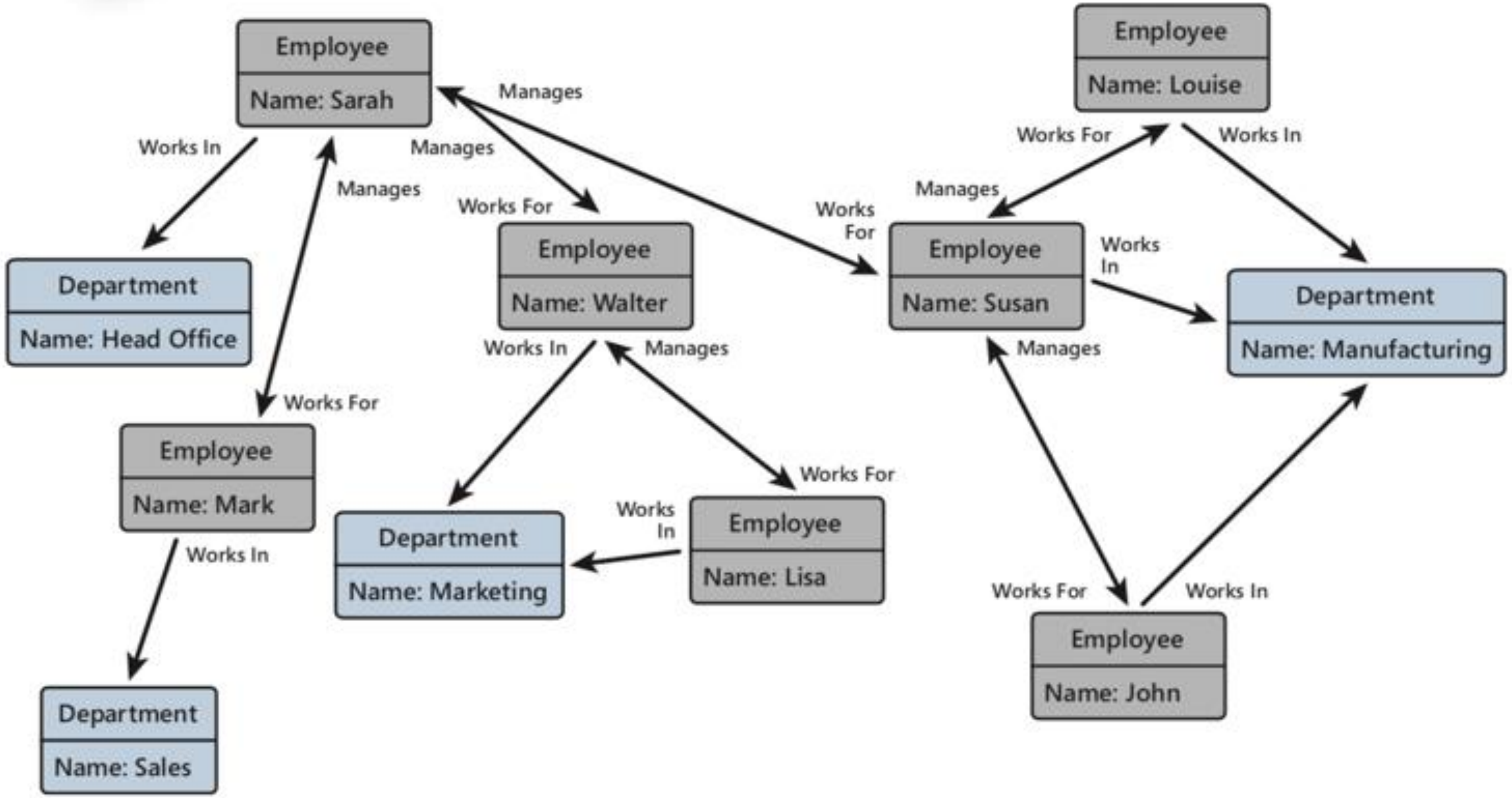
Name	ID
John	001
Karen	002
Bill	003

Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003



Graph Databases – Another Example of NoSQL Databases



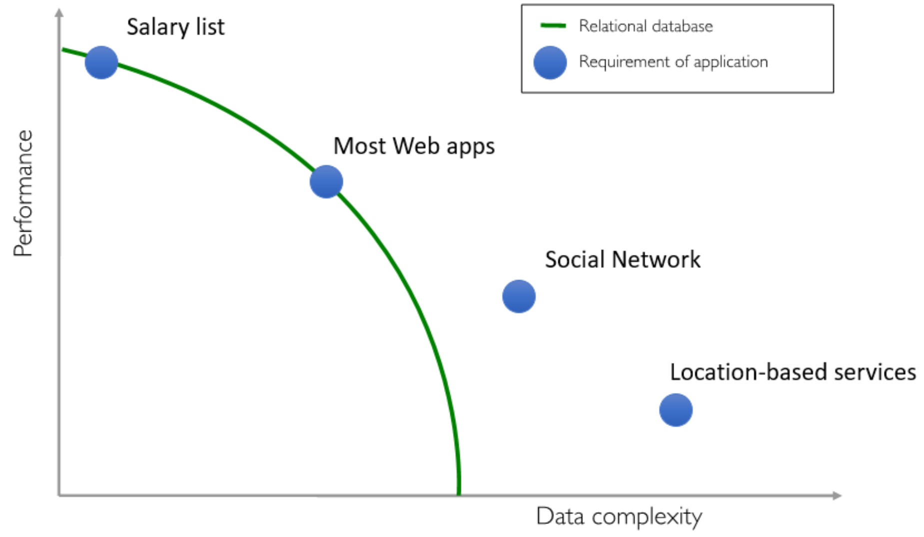
Graph Databases – Why? Relational Databases Do Not Scale Well

Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

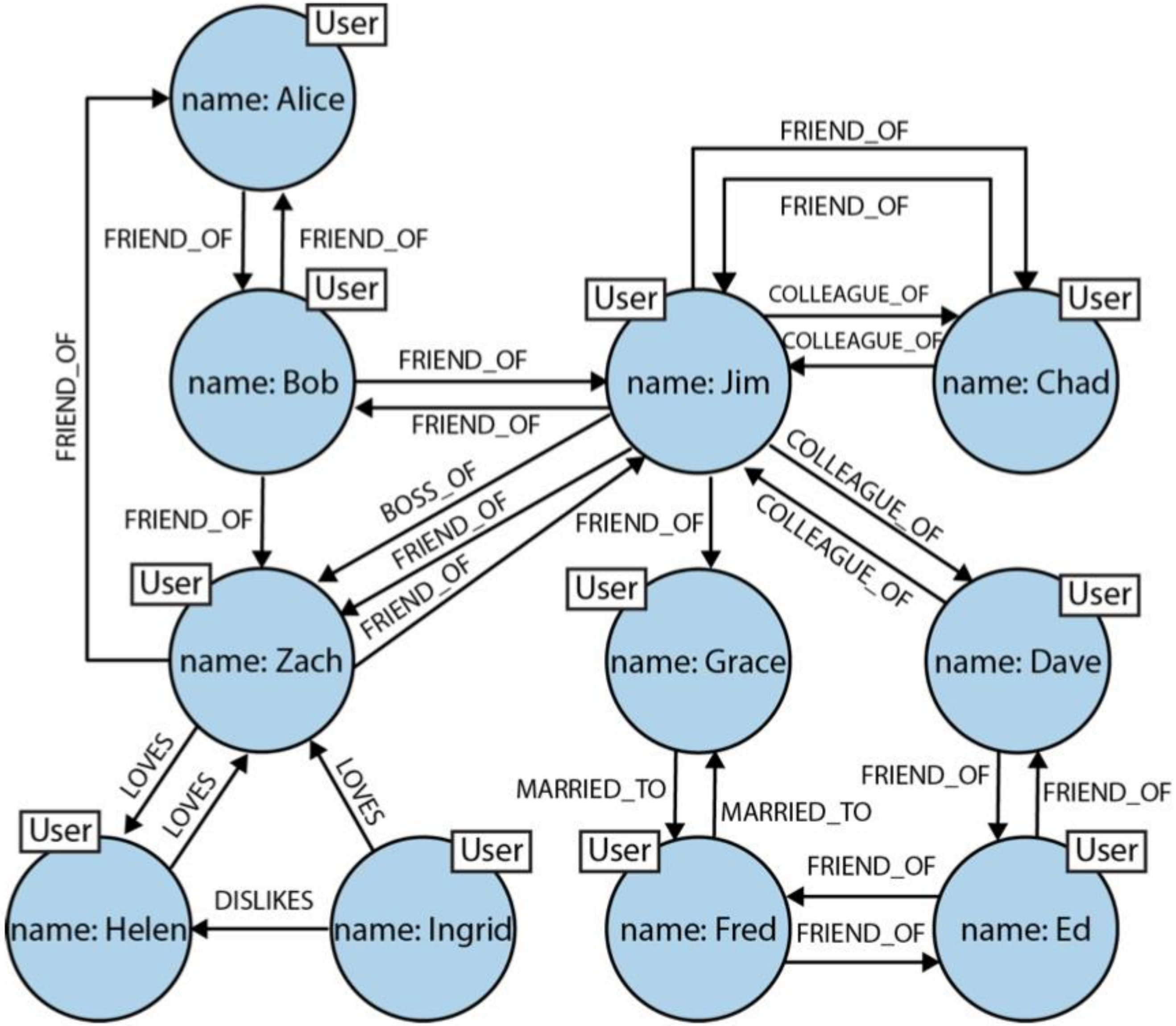
PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

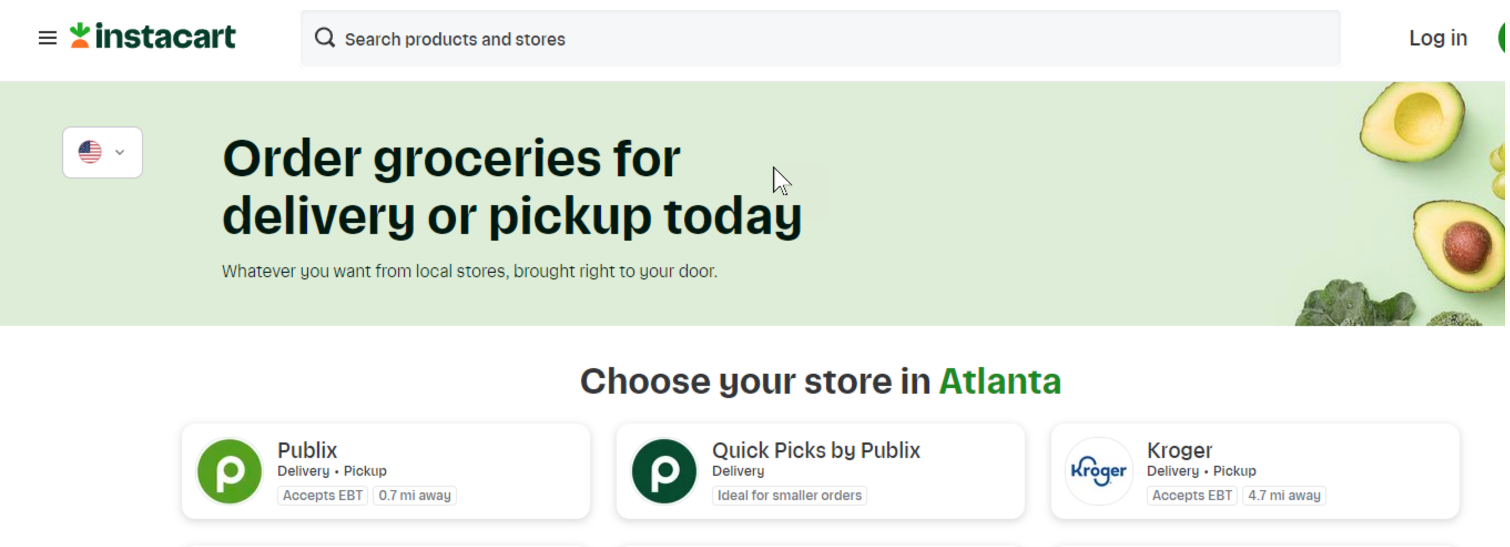
Old SQL Relational Database Way

```sql
SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
  ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
  ON pf2.PersonID = pf1.FriendID
JOIN Person p2
  ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```

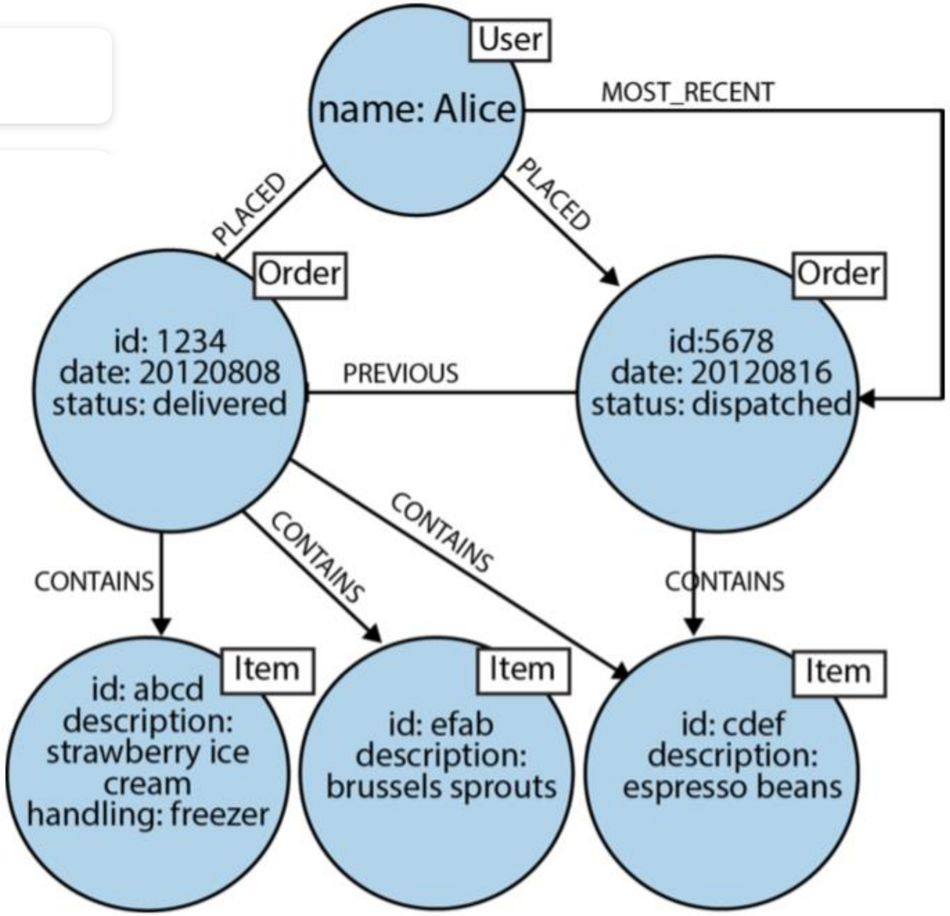


Graph Databases – Why?



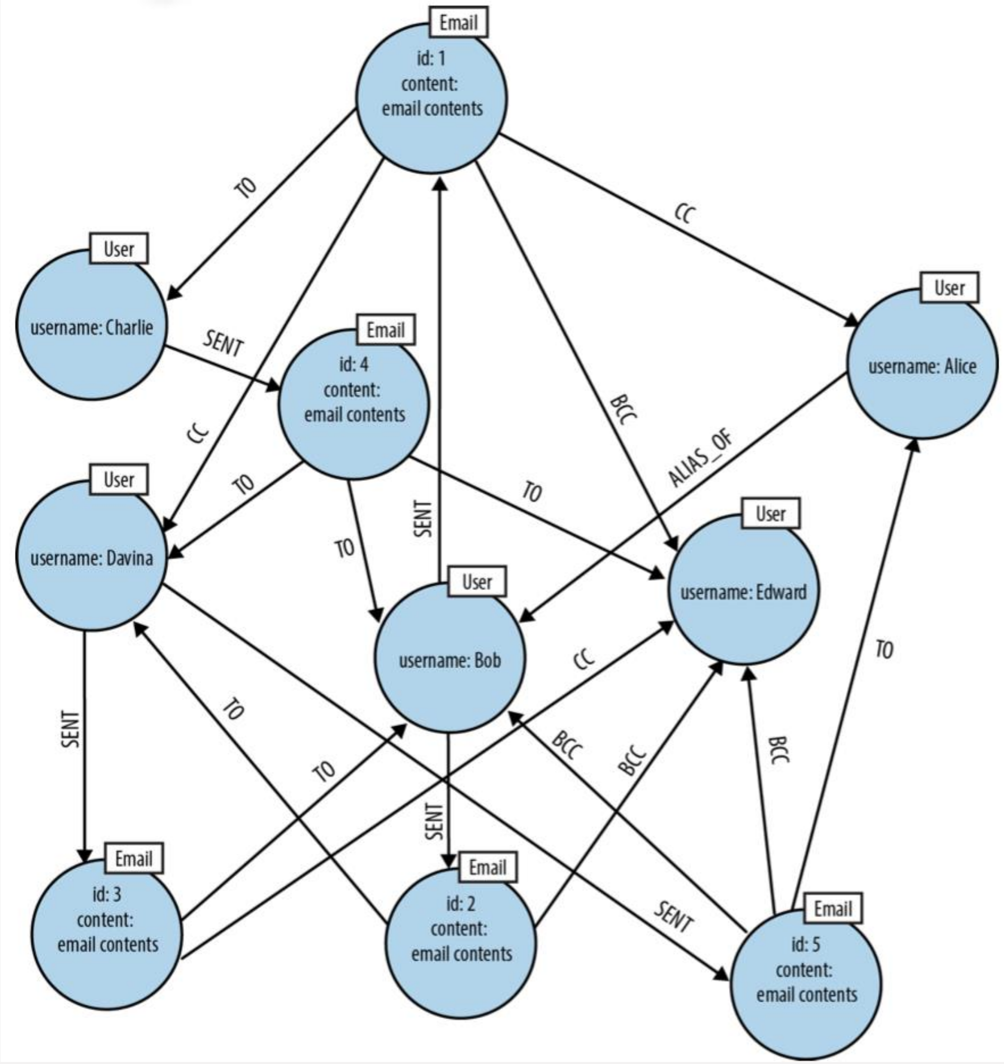


How Instacart Stores Your Data



Examples of User of Graph Databases

- **Tracing Suspicious Email**
- **Where Copies of Email are sent to sender's Alias.**

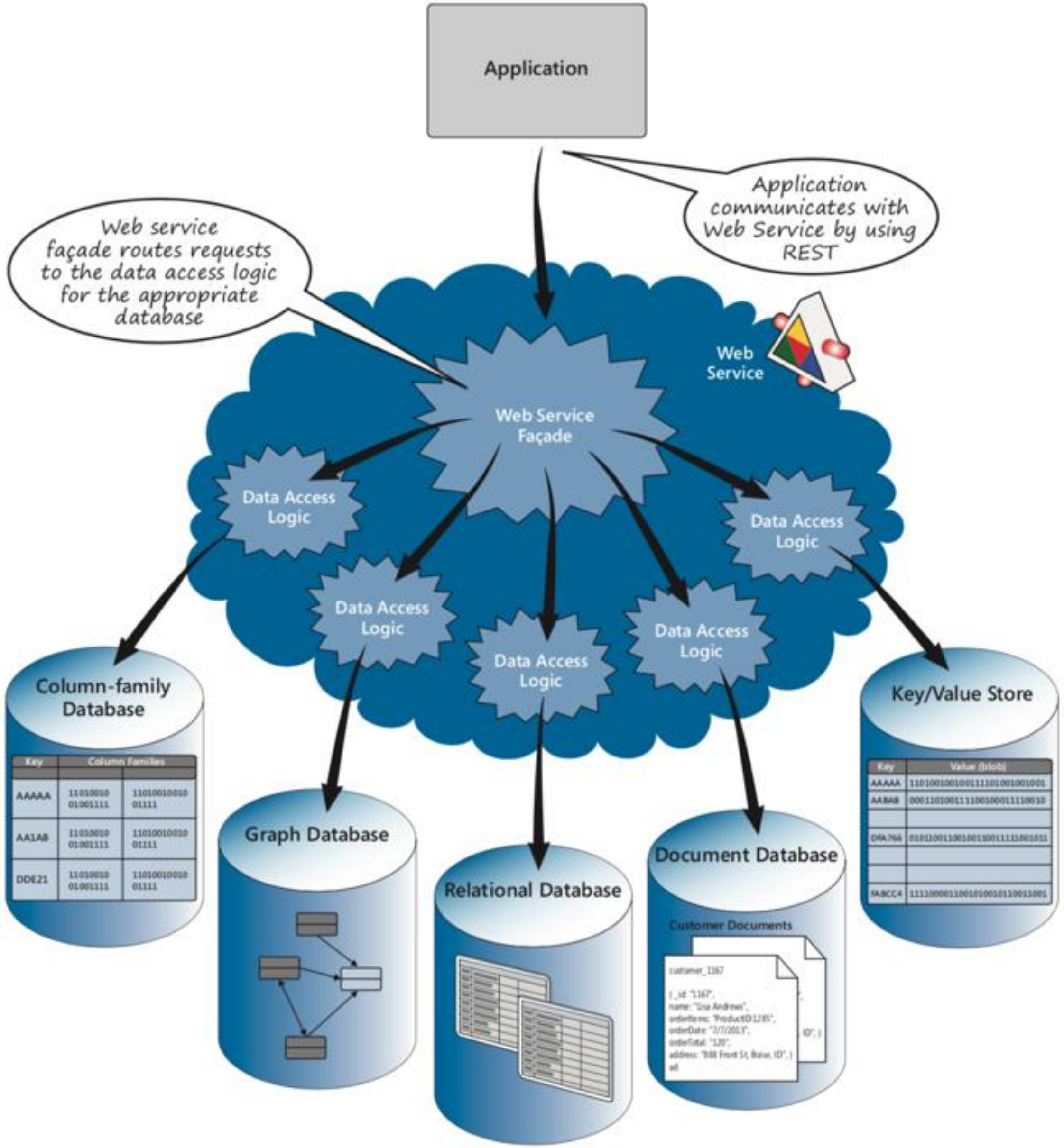



```
MATCH (bob:User {username:'Bob'})-[:SENT]->(email)-[:CC]->(alias),
      (alias)-[:ALIAS_OF]->(bob)
RETURN email.id
```

Performance of Graph DBs, like Neo4j

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Summary - Web Services & Cloud Applications Today



SQL versus NoSQL

Feature/Functionality	SQL	NoSQL
Standardization and interoperability	Mature technology, well understood. Subject to many ANSI and ISO standards, enabling interoperability between conforming implementations. Standard APIs (ODBC, SQL/CLI, and so on) enable applications to operate with databases from different vendors.	New technologies with no common overarching model or standardization. Each NoSQL database vendor defines their own APIs and data formats. Minimal interoperability between different vendors is possible at the database level. Application code written for one NoSQL database is unlikely to work against a NoSQL database from a different vendor.
Storing complex data	Data for complex objects is often normalized into several entities spanning multiple tables to reduce data redundancy. It may require intricate SQL queries to reconstruct the data for a single complex object from these tables. ORMs can abstract some of the details, but this extra layer can lead to inefficiencies.	Can store data for complex objects without splitting across aggregates, enabling a much simpler mapping between objects used by applications and the data stored in the database. This enables fast query access at the possible cost of additional complexity in application code when storing or updating denormalized data.
Performing queries	Relational model is very generalized. SQL supports ad-hoc queries by joining tables and using subqueries. Very good at summarizing and grouping relational data. Less good at handling complex non-relational queries.	Databases are designed to optimize specific queries, most commonly retrieving data from a single aggregate by using a key. Most NoSQL databases do not support data retrieval from multiple aggregates within the same query. Summarizing and grouping data may require implementing map/reduce functions to work efficiently. Graph databases can be excellent for handling complex non-relational queries.

SQL versus NoSQL …..

Feature/Functionality	SQL	NoSQL
Scalability	Most suited to scale-up scenarios rather than scale out due to the performance of distributed transactions and cross-database queries. Some relational vendors support clustering and sharding as optional extensions.	Mostly designed with support for scaling out built-in. Many NoSQL databases provide seamless support for clustering and sharding data.
Performance with large datasets	Can require significant tuning to read from or write to large datasets efficiently.	Designed to be very efficient for handling large datasets.
Data consistency and transactions	Designed to be highly consistent (ACID), but at the cost of transactional performance. Transactional consistency can slow down operations in a distributed environment.	Designed to be eventually consistent (BASE) in a distributed environment. Some support for ACID properties for updates within an aggregate, but cross-aggregate operations are not guaranteed to be atomic. Careful use of quorums can help to reduce instances of inconsistency.
Integration	Relational databases can be easily shared between different applications. The database acts as the point of integration between applications.	Databases are usually designed specifically to support a single application. Application integration is usually handled by application code.

aws | Contact Sales | Products | Solutions | Pricing | Getting Started | Documentation | AWS Marketplace | More | English | My Account | Sign In to t

What is NoSQL? | Overview | Key-value | Document | Graph | In-memory | Search

SQL (relational) vs. NoSQL (nonrelational) databases

Though there are many types of NoSQL databases with varying features, the following table shows some of the differences between SQL and NoSQL databases.

	Relational databases	NoSQL databases
Optimal workloads	Relational databases are designed for transactional and strongly consistent online transaction processing (OLTP) applications and are good for online analytical processing (OLAP).	NoSQL key-value, document, graph, and in-memory databases are designed for OLTP for a number of data access patterns that include low-latency applications. NoSQL search databases are designed for analytics over semi-structured data.
Data model	The relational model normalizes data into tables that are composed of rows and columns. A schema strictly defines the tables, rows, columns, indexes, relationships between tables, and other database elements. The database enforces the referential integrity in relationships between tables.	NoSQL databases provide a variety of data models that includes document, graph, key-value, in-memory, and search.
ACID properties	Relational databases provide atomicity, consistency, isolation, and durability (ACID) properties: • Atomicity requires a transaction to execute completely or not at all. • Consistency requires that when a transaction has been committed, the data must conform to the database schema. • Isolation requires that concurrent transactions execute separately from each other. • Durability requires the ability to recover from an unexpected system failure or power outage to the last known state.	NoSQL databases often make tradeoffs by relaxing some of the ACID properties of relational databases for a more flexible data model that can scale horizontally. This makes NoSQL databases an excellent choice for high throughput, low-latency use cases that need to scale horizontally beyond the limitations of a single instance.
Performance	Performance is generally dependent on the disk subsystem. The optimization of queries, indexes, and table structure is often required to achieve peak performance.	Performance is generally a function of the underlying hardware cluster size, network latency, and the calling application.
Scale	Relational databases typically scale up by increasing the compute capabilities of the hardware or scale-out by adding replicas for read-only workloads.	NoSQL databases typically are partitionable because key-value access patterns are able to scale out by using distributed architecture to increase throughput that provides consistent performance at near boundless scale.
APIs	Requests to store and retrieve data are communicated using queries that conform to a structured query language (SQL). These queries are parsed and executed by the relational database.	Object-based APIs allow app developers to easily store and retrieve in-memory data structures. Partition keys let apps look up key-value pairs, column sets, or semistructured documents that contain serialized app objects and attributes.

NoSQL

United States

Search

22 2 17

Jobs

Date posted

Experience level

Salary

Company

Remote

Easy Apply

All filters

NoSQL in United States

11,932 results

Set alert

Promoted · Easy Apply

Senior Director of Engineering

Demand.io

Los Angeles Metropolitan Area (On-site)

$350K/yr - $550K/yr · 401(k), +6 benefits

Response time is typically 4 days

Promoted · Easy Apply

Lead Data Engineer-Vice President

JPMorgan Chase & Co.

Wilmington, DE

692 school alumni work here

Promoted · 6 applicants

Python Developer

ApTask

Jersey City, NJ (On-site)

$120K/hr - $130K/hr

Promoted · Easy Apply

Data Modeler (Hybrid onsite W2 only)

Lead Data Engineer-Vice President

JPMorgan Chase & Co. · Wilmington, DE

Apply

Save

data asset security with minimal supervision

- Adds to team culture of diversity, equity, inclusion, and respect

Required Qualifications, Capabilities, And Skills

- Formal training or certification on data related field and 5+ years applied experience
- Subject matter expertise in Financial Services and/or Data Engineering/Architecture
- Working experience with both relational and NoSQL databases
- Strong knowledge of cloud platforms, AWS, AWS Certifications, Project Management, Agile, JIRA and SDLC
- Technical knowledge of data management and governance, and database systems including AWS Aurora, Cassandra, Oracle, Cockroach and DB2
- Technical knowledge of Apache Kafka and REST API
- Proficiency in data modeling tools and software (e.g., Erwin, SQL)
- Exceptional problem solving and critical thinking abilities
- Demonstrated ability to manage delivery timelines, and ensure product teams stay on track to meet goals
- Strong interpersonal and communication skills, adept at explaining and converting complex concepts into digestible information to be consumed by audiences of varying levels of expertise and seniority.

Preferred Qualifications, Capabilities, And Skills

Summary

- NoSQL databases are very popular for cloud applications
- We will discuss different types of NoSQL databases in detail in future classes and labs