BITS Pilani presentation

BITS Pilani
Hyderabad Campus

D. Powar
Lecturer,
BITS-Pilani, Hyderabad Campus

innovate achieve lead

BITS Pilani
Hyderabad Campus

SSZ G527

Cloud Computing

Agenda:

- Introduction to SQL
- NoSQL
  - Key/value stores
  - Column family stores
  - Document stores
  - Graph databases

Introduction

- In the computing system (web and business applications), there are enormous data that comes out every day from the web.
- A large section of these data is handled by Relational database management systems (RDMBSs).
- Beyond the intended benefits, the relational model is well-suited to client-server programming and today it is predominant technology for storing structured data in web and business applications.

ACID Rules

Atomic : A transaction is a logical unit of work which must be either completed with all of its data modifications, or none of them is performed.

Consistent : At the end of the transaction, all data must be left in a consistent state.

Isolated : Modifications of data performed by a transaction must be independent of another transaction. Unless this happens, the outcome of a transaction may be erroneous.

Durable : When the transaction is completed, effects of the modifications performed by the transaction must be permanent in the system.

Distributed Systems

- Consists of multiple computers and software components that communicate through a computer network.
- Can consist of any number of possible configurations, such as mainframes, workstations, personal computers, and so on.
- The computers interact with each other and share the resources of the system to achieve a common goal.

Advantages of Distributed Systems

Reliability (fault tolerance) :

- The important advantage of distributed computing system is reliability. If some of the machines within the system crash, the rest of the computers remain unaffected and work does not stop.

Scalability :

- In distributed computing the system can easily be expanded by adding more machines as needed.

Sharing of Resources :

- Shared data is essential to many applications such as banking, reservation system. As data or resources are shared in distributed system, other resources can be also shared (e.g. expensive printers).

Advantages of Distributed Systems

Flexibility :

- As the system is very flexible, it is very easy to install, implement and debug new services.

Speed :

- A distributed computing system can have more computing power and it's speed makes it different than other systems.

Open system :

- As it is open system, every service is equally accessible to every client i.e. local or remote.

Performance :

- The collection of processors in the system can provide higher performance (and better price/performance ratio) than a centralized computer.

Disadvantages of Distributed Systems

Troubleshooting :

- Troubleshooting and diagnosing problems.

Software :

- Less software support is the main disadvantage of distributed computing system.

Networking :

- The network infrastructure can create several problems such as transmission problem, overloading, loss of messages.

Security :

- Easy access in distributed computing system increases the risk of security and sharing of data generates the problem of data security

Scalability

- scalability is the ability of a system to expand to meet your business needs.
- For example scaling a web application is all about allowing more people to use your application.
- You scale a system by upgrading the existing hardware without changing much of the application or by adding extra hardware.

Scalability

Vertical scaling

- To scale vertically (or scale up) means to add resources within the same logical unit to increase capacity. For example to add CPUs to an existing server, increase memory in the system or expanding storage by adding hard drive.

Horizontal scaling

- To scale horizontally (or scale out) means to add more nodes to a system, such as adding a new computer to a distributed software application. In NoSQL system, data store can be much faster as it takes advantage of “scaling out” which means to add more nodes to a system and distribute the load over those nodes.

What is NOSQL?

- NoSQL is a non-relational database management systems, different from traditional relational database management systems in some significant ways.
- It is designed for distributed data stores where very large scale of data storing needs (for example Google or Facebook which collects terabits of data every day for their users).
- These type of data storing may not require fixed schema, avoid join operations and typically scale horizontally.
- No to ACID, BASE is enough

Why NOSQL?

- In today's time data is becoming easier to access and capture through third parties such as Facebook, Google+ and others.
- Personal user information, social graphs, geo location data, user-generated content and machine logging data are just a few examples where the data has been increasing exponentially.
- To avail the above service properly, it is required to process huge amount of data.
- The evolution of NoSql databases is to handle these huge data properly.

RDBMS Vs. NOSQL

NOSQL:

- ✓ No declarative query language
- ✓ No predefined schema
- ✓ Key-Value pair storage, Column Store, Document Store, Graph databases
- ✓ Eventual consistency rather ACID property
- ✓ Unstructured and unpredictable data
- ✓ CAP Theorem
- ✓ Prioritizes high performance, high availability and scalability

History of NOSQL

- The term NoSQL was coined by Carlo Strozzi in the year 1998. He used this term to name his Open Source, Light Weight, DataBase which did not have an SQL interface.
- In the early 2009, when last.fm wanted to organize an event on open-source distributed databases, Eric Evans, a Rackspace employee, reused the term to refer databases which are non-relational, distributed, and does not conform to ACID - four obvious features of traditional relational database systems.
- In the same year, the "no:sql(east)" conference held in Atlanta, USA, NoSQL was discussed and debated a lot.
- And then, discussion and practice of NoSQL got a momentum, and NoSQL saw an unprecedented growth.

CAP Theorem

CAP theorem states that there are three basic requirements which exist in a special relation when designing applications for a distributed architecture.

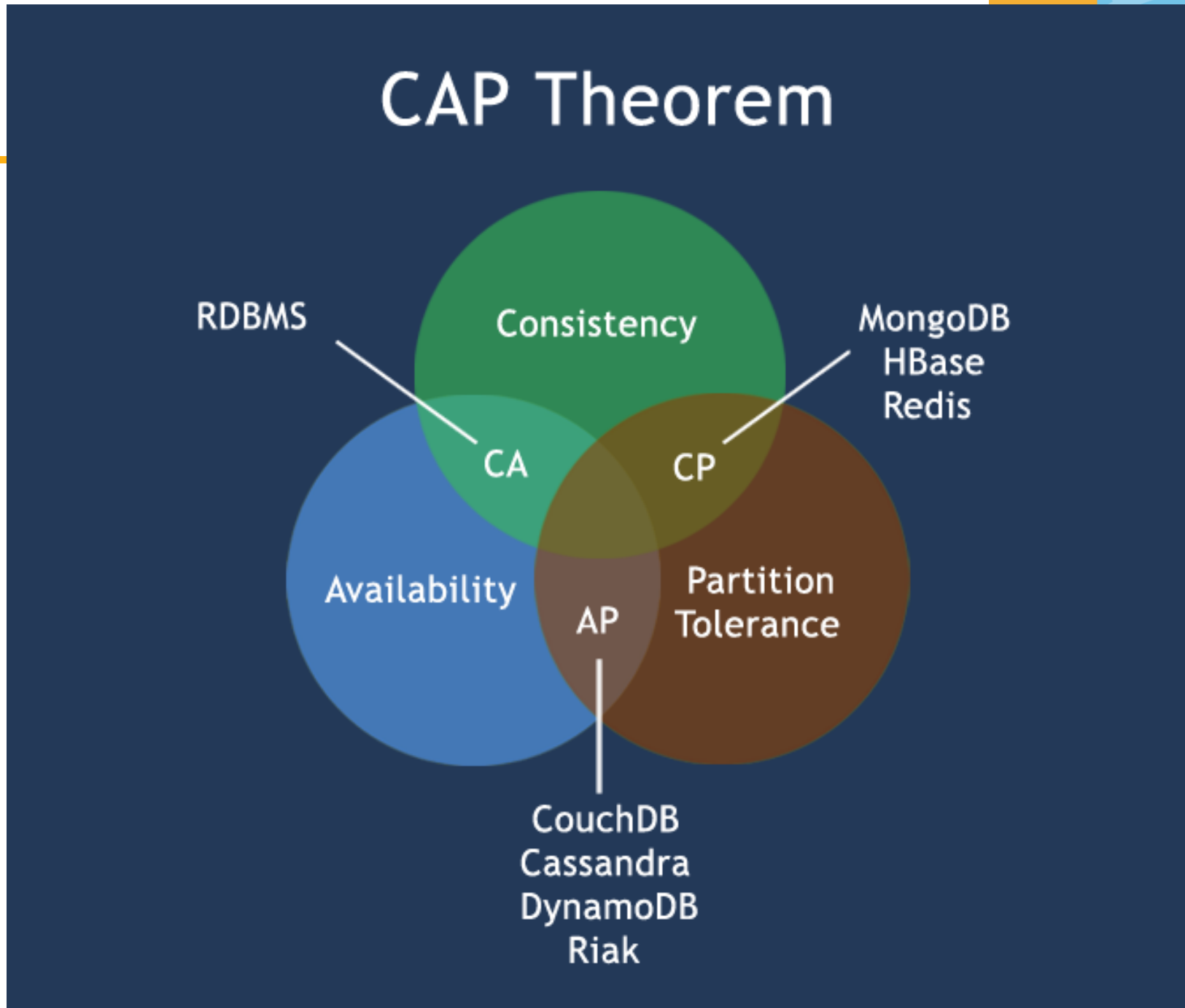
Consistency - This means that the data in the database remains consistent after the execution of an operation. For example after an update operation all clients see the same data.

Availability - This means that the system is always on (service guarantee availability), no downtime.

Partition Tolerance - This means that the system continues to function even the communication among the servers is unreliable, i.e. the servers may be partitioned into multiple groups that cannot communicate with one another.

CAP Theorem

- Theoretically it is impossible to fulfil all 3 requirements:
- CAP provides the basic requirements for a distributed system to follow 2 of the 3 requirements. Therefore all the current NoSQL database follow the different combinations of the C, A, P from the CAP theorem.
- **CA -** Single site cluster, therefore all nodes are always in contact. When a partition occurs, the system blocks.
- **CP -** Some data may not be accessible, but the rest is still consistent/accurate.
- **AP -** System is still available under partitioning, but some of the data returned may be inaccurate.

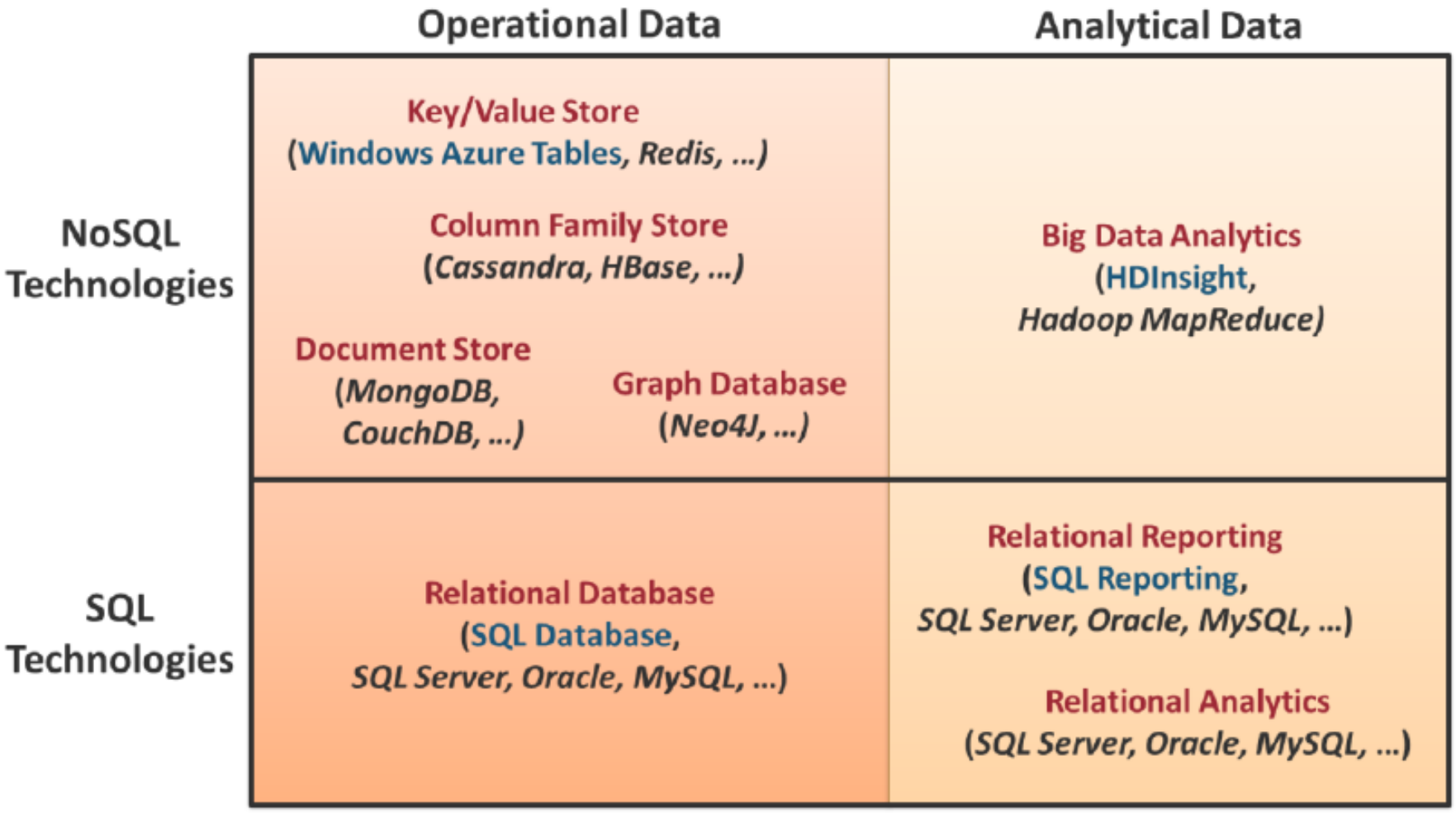

CAP Theorem
RDBMS
Consistency
MongoDB
HBase
Redis
CA
CP
Availability
Partition
Tolerance
AP
CouchDB
Cassandra
DynamoDB
Riak

Data?

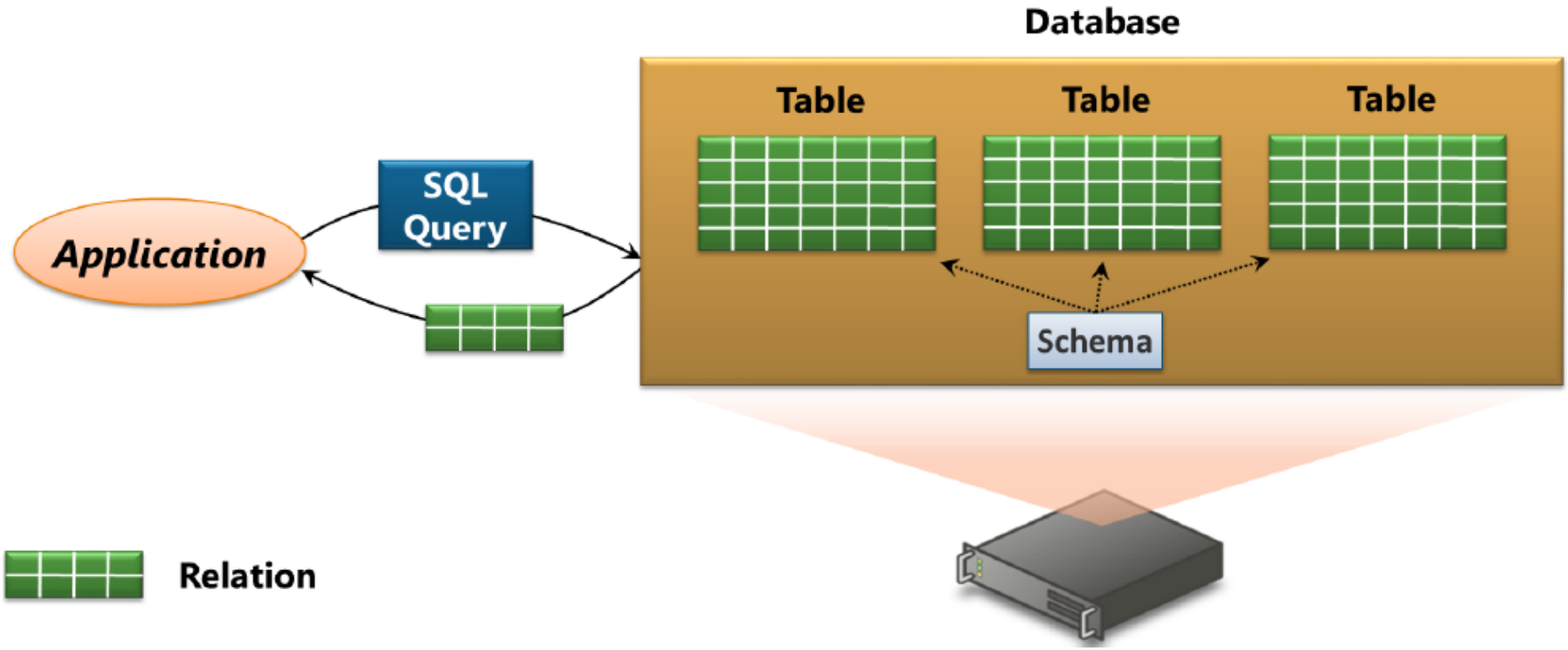
Data can be broadly categorized into 2 types:

- *Operational data*
  - shopping cart data in a web commerce application,
  - information about employees in a HR system,
  - buy/sell prices in a stock-trading application, etc
- *Analytical data*
  - provides business intelligence
  - analyzing data to learn about customer buying habits or market trends

Data technologies on Windows Azure can be organized into four quadrants



A relational database



Key/Value Stores

- Relational technologies impose a significant amount of structure on data.
- But what if an application needs almost no structure?
- Key/value stores hold just simple properties
- Ex: shopping cart in a web commerce application

Key/Value Stores

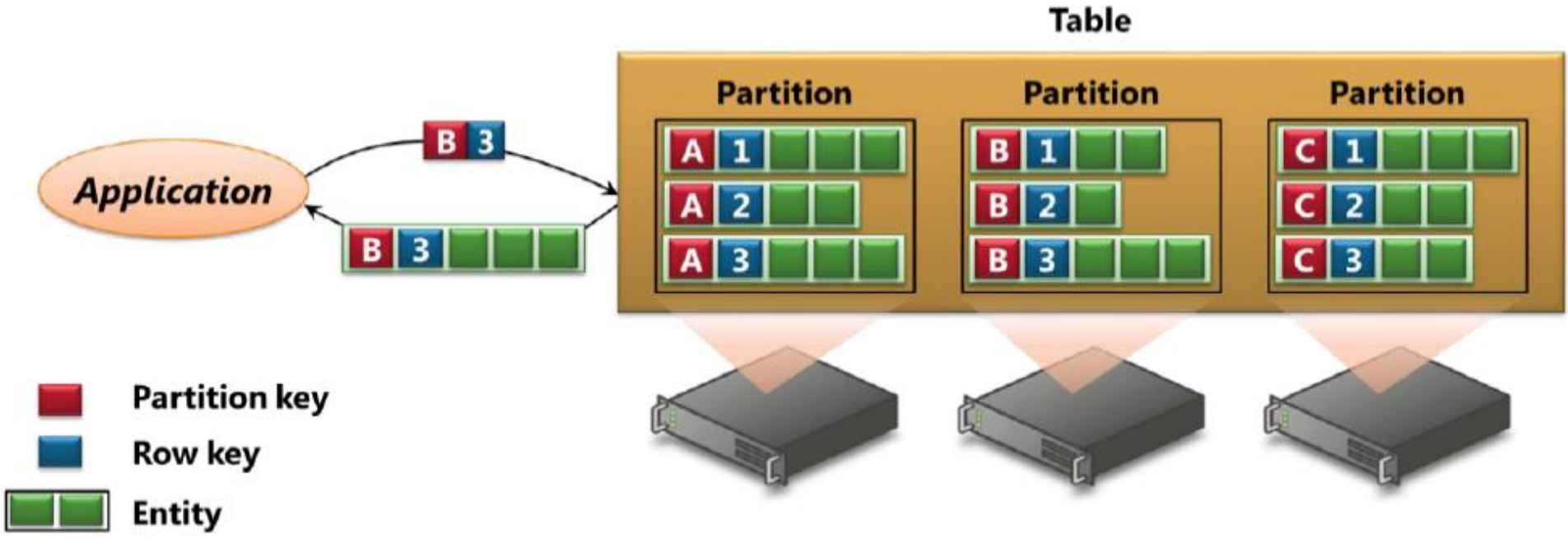


Fig. Windows Azure Tables

Sharding - the partitions in a single table can be spread across multiple machines
Sharding also brings some constraints
Provides strong consistency

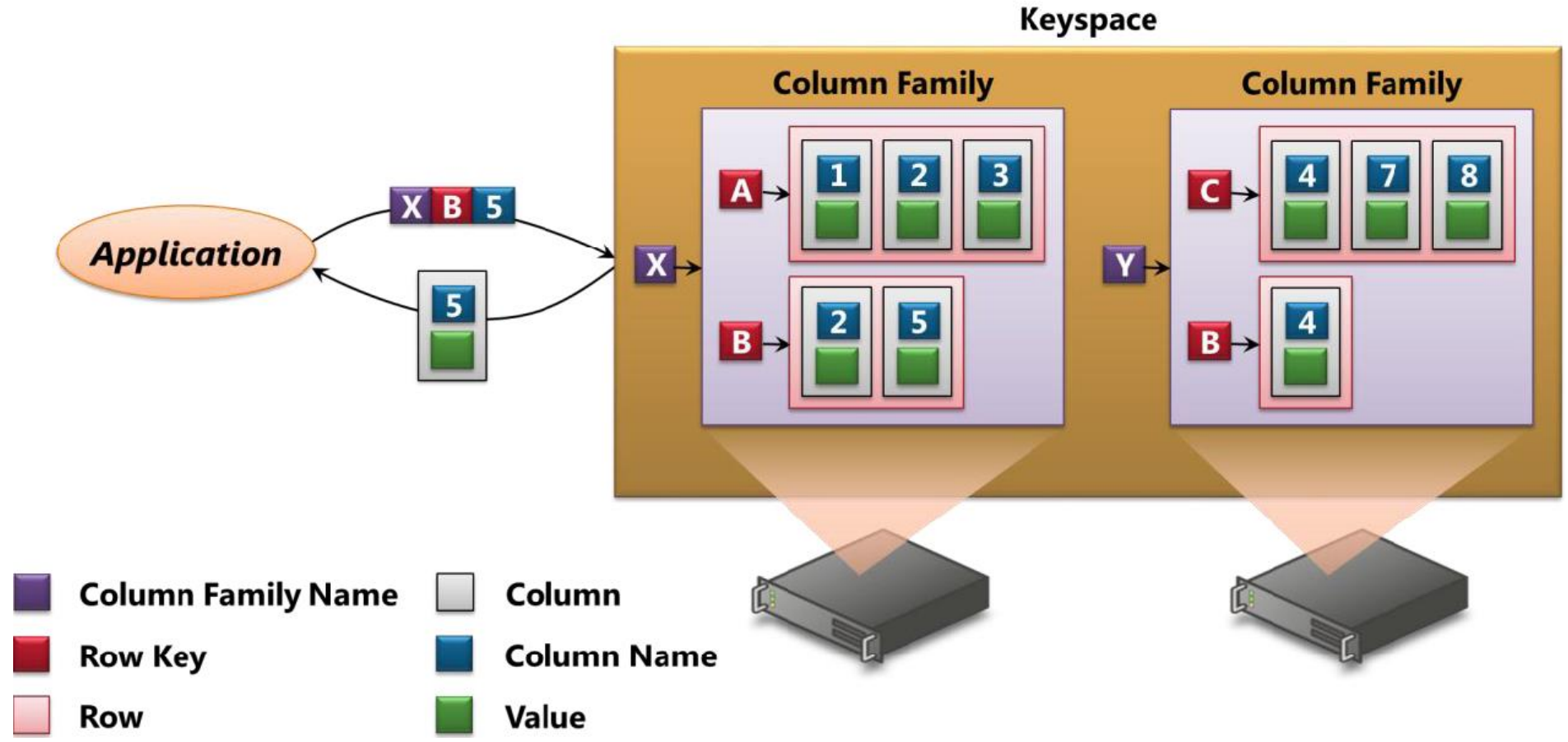
Use??

- Does the application work with relatively simple data? (applications that need the powerful query capabilities of a relational database aren't a particularly good fit)
- Does the application need to work with large amounts of data?
- Low-cost managed service (Ex- Windows Azure Tables are significantly cheaper than SQL Database)

Column Family Stores

- A little more structure in your data (organize properties in a somewhat more structured way) - maybe a simple set of keys and values isn't enough.
- A column family store must be your choice
- A column family store is similar to a key/value store in that an application supplies a unique key and gets back a set of values
- These values are organized in a somewhat more complex fashion than with a key/value store

Column Family Stores (Cassandra stores)



Key <column family, row key, column name>

Column Family Stores (contd..)

- writes don't replace old values. By default, the latest version is returned, but it's possible to explicitly request an earlier version of a column's value.
- rows and columns are sorted by their keys
- consistency: eventual or strong? Cassandra's answer is interesting; an application can decide what level of consistency it wants on a per-request basis.

Column Family Stores (Use?)

- Does the application work with data that's too simple to need a relational database but could benefit from more structure than a key/value store provides?
- Does the application need to work with large amounts of data?
  - Cassandra and HBase are designed to scale well beyond the limits of a relational database

Document Stores

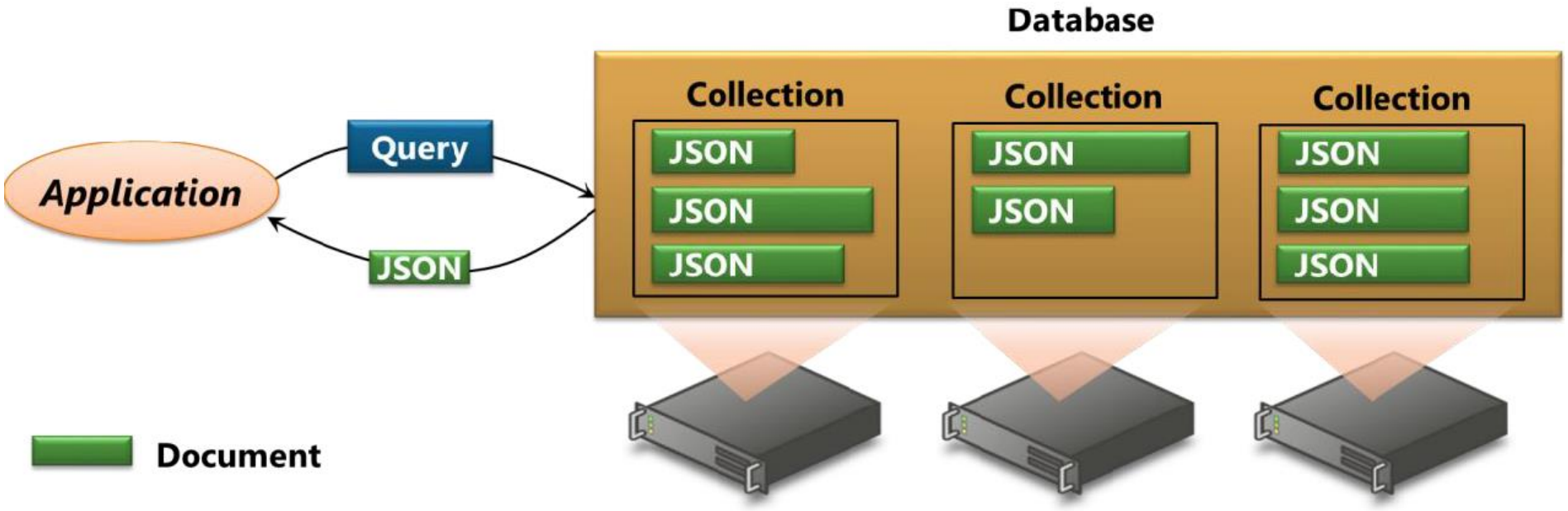
- The rows and columns in a relational table provide structure for data.
- But what if that structure doesn't match the data your application is working with?
- For an application working with **JSON** data, a storage technology designed for JSON may well be a better fit.
- This is a primary motivation for the creation of document stores

Document Stores

- In JSON, information is represented as character strings organized into name/value pairs called properties. For example, information about an order might be described as:

```
{
   "Order": {
     "OrderID": 5630,
     "Status": "Awaiting delivery",
     "CustID": 8499734 }
}
```

Document Stores (MongoDB)



- name “Mongo” is derived from the word “humongous”

Document Stores (MongoDB)

- MongoDB enforces no schema, and so unlike a relational table, where each row holds data in a fixed set of columns, a document in a collection can contain whatever the developer likes
- An application issues queries, each of which returns JSON data.
- Each document has a unique identifier, so it's possible to treat the database like a key/value store: the unique identifier is the key, while the document itself is the value returned.
- MongoDB also allows creating secondary indexes on specific properties, such as OrderID in the JSON example shown earlier, allowing more fine-grained queries.

Document Stores (MongoDB)

- No concept of joins
  - ✓ Data that's often accessed together should be kept in the same collection
  - ✓ Transactions also can't span collections
- For writes, MongoDB supports both eventual and strong consistency
- Windows Azure provides MongoDB offering provided by MongoLabs as managed service

Document Stores (MongoDB) - Use?

- Does the application work with JSON data?
- Does the application need to work with large amounts of data?

- CouchDB, Jackrabbit, OrientDB, SimpleDB,Terrastore, etc.

Graph Databases

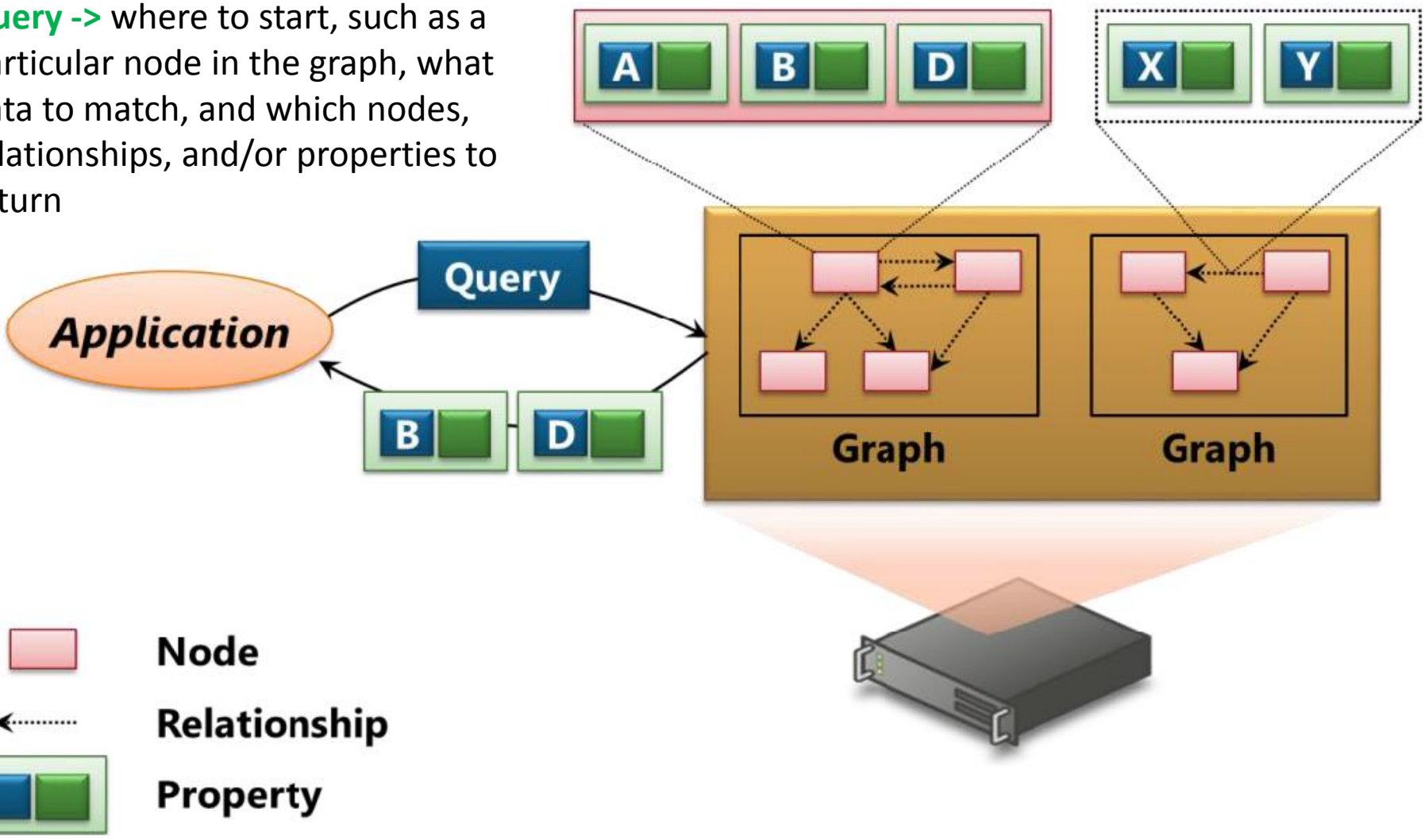
- Data items (customer, order, and web page, etc), are clearly important, but how important are the relationships among these data items?
- The answer depends on what your application needs to do with that data.
- If the relationships matter so much, why make your application reconstruct them from the data using joins? Why not embed the relationships in the data itself?

Graph Databases

- Graph database represents data as a graph
- The data items are called as nodes, while the edges that connect those nodes are called as relationships
- Both nodes and relationships can have properties, which are just name/value pairs
- It doesn’t enforce schema, so the properties can contain whatever information you want (most useful)

Graph Database (Neo4J)

Query -> where to start, such as a particular node in the graph, what data to match, and which nodes, relationships, and/or properties to return



Graph Databases (Use?)

- Is a graph the most natural way to work with your data?
- Can your application live within the scale constraints imposed by a graph database (Single server)?
  - ✓ It's not designed for the massive datasets supported by key/value stores or column family stores or document stores.
  - ✓ If your application needs operational access to very large amounts of data, another NoSQL technology is probably a better choice

- Ex: AllegroGraph, DEX, FlockDB, Sones GraphDB, etc

Summary

- Introduction to SQL
- NoSQL
  - Key/value stores
  - Column family stores
  - Document stores
  - Graph databases