JAIN DEEMED-TO-BE UNIVERSITY | SCHOOL OF COMPUTER SCIENCE AND IT

Master of Computer Applications

18MCA301 – **NoSQL Databases**

Module -1

Introduction to NoSQL Databases

III General

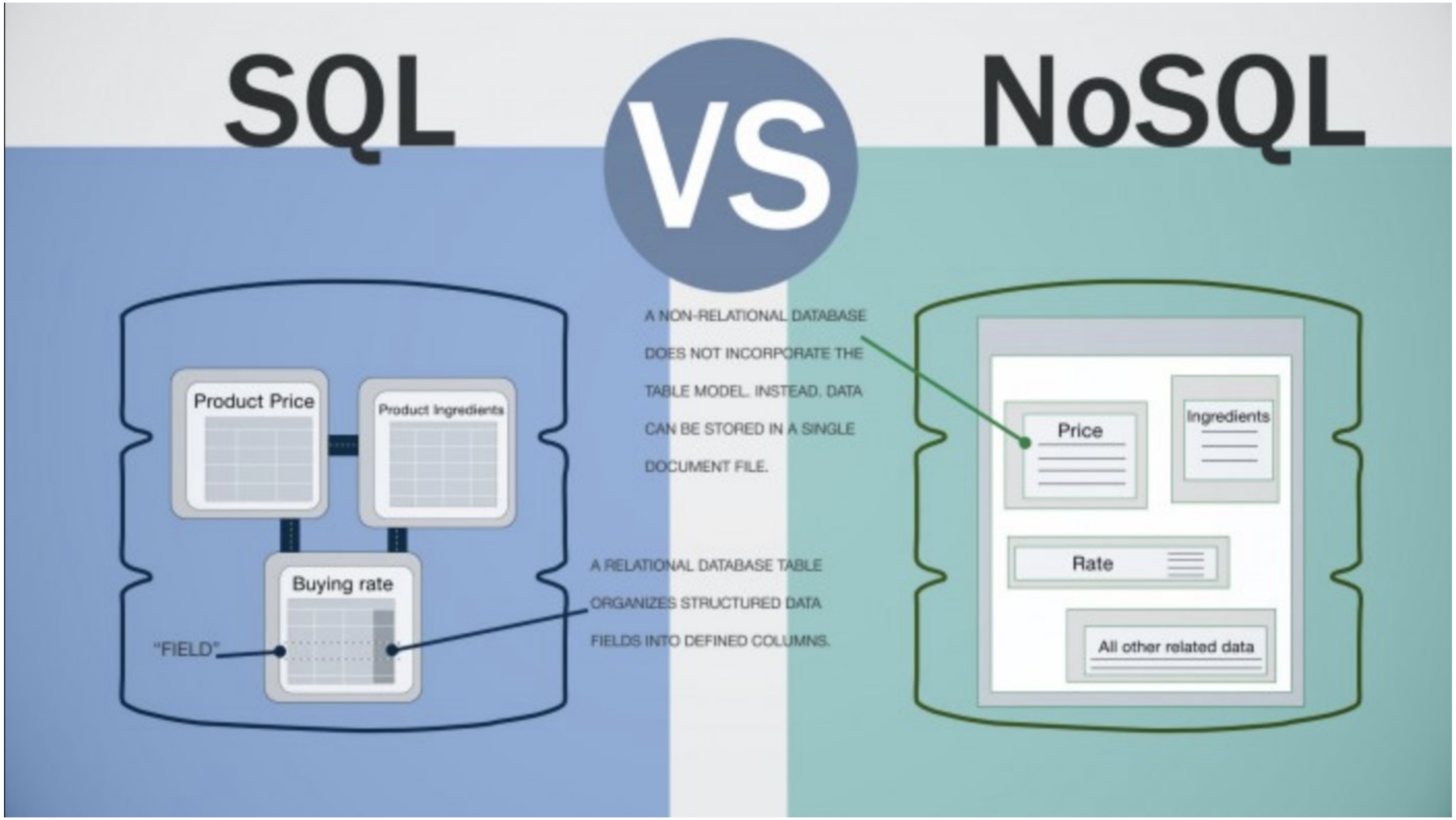
Dr Gangothri

August 2020

SCHOOL OF COMPUTER SCIENCE AND IT

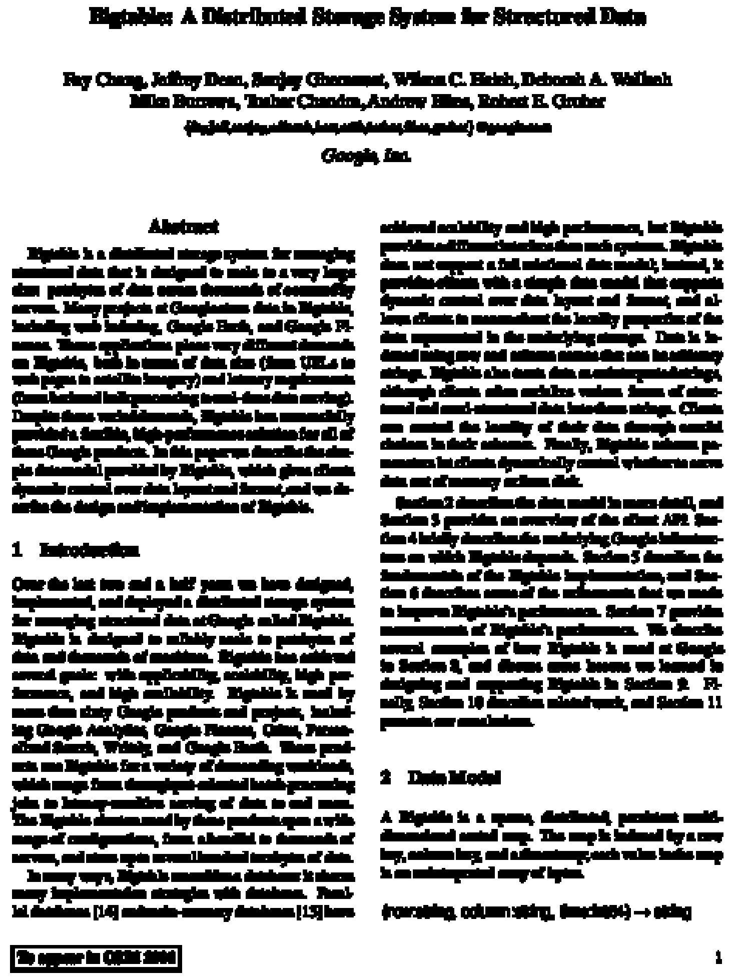
History of NoSQL

- The explosion of WWW, social media, web forms in the past 15 years and greater connectivity to internet reveals a vast array of data generated and used.
- Therefore, instead of conventional systems, a more sophisticated systems must be created to store data and make information readily available.
- A huge array of these kinds of information exists in the form of NOSQL databases.
- NoSQL is launched in 1998.
- NoSQL isn't a single technology invented by a couple of computer guys or a mathematician theorizing about data structures.
- The two major milestones in the history of NoSQL is *Google Bigtable* and *Amazon's Dynamo.*


SQL
VS
NoSQL
Product Price
Product Ingredients
Buying rate
"FIELD"
A RELATIONAL DATABASE TABLE ORGANIZES STRUCTURED DATA FIELDS INTO DEFINED COLUMNS.
A NON-RELATIONAL DATABASE DOES NOT INCORPORATE THE TABLE MODEL. INSTEAD, DATA CAN BE STORED IN A SINGLE DOCUMENT FILE.
Price
Ingredients
Rate
All other related data

Google Bigtable Pap

- In 2006, Google released “Bigtable: A Distributed Storage System for Structured Data”.
- Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers.
- Bigtable stores rows with a single key and stores data i the rows within related column families.
- By using simple keys, related data such as all pages on the same website , can be grouped together, which increases the speed of analysis.
- That is, with Bigtable, column families allow related da to be stored in a single record.



Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
[illegible]
Google, Inc.

Abstract

[illegible]

1 Introduction

[illegible]

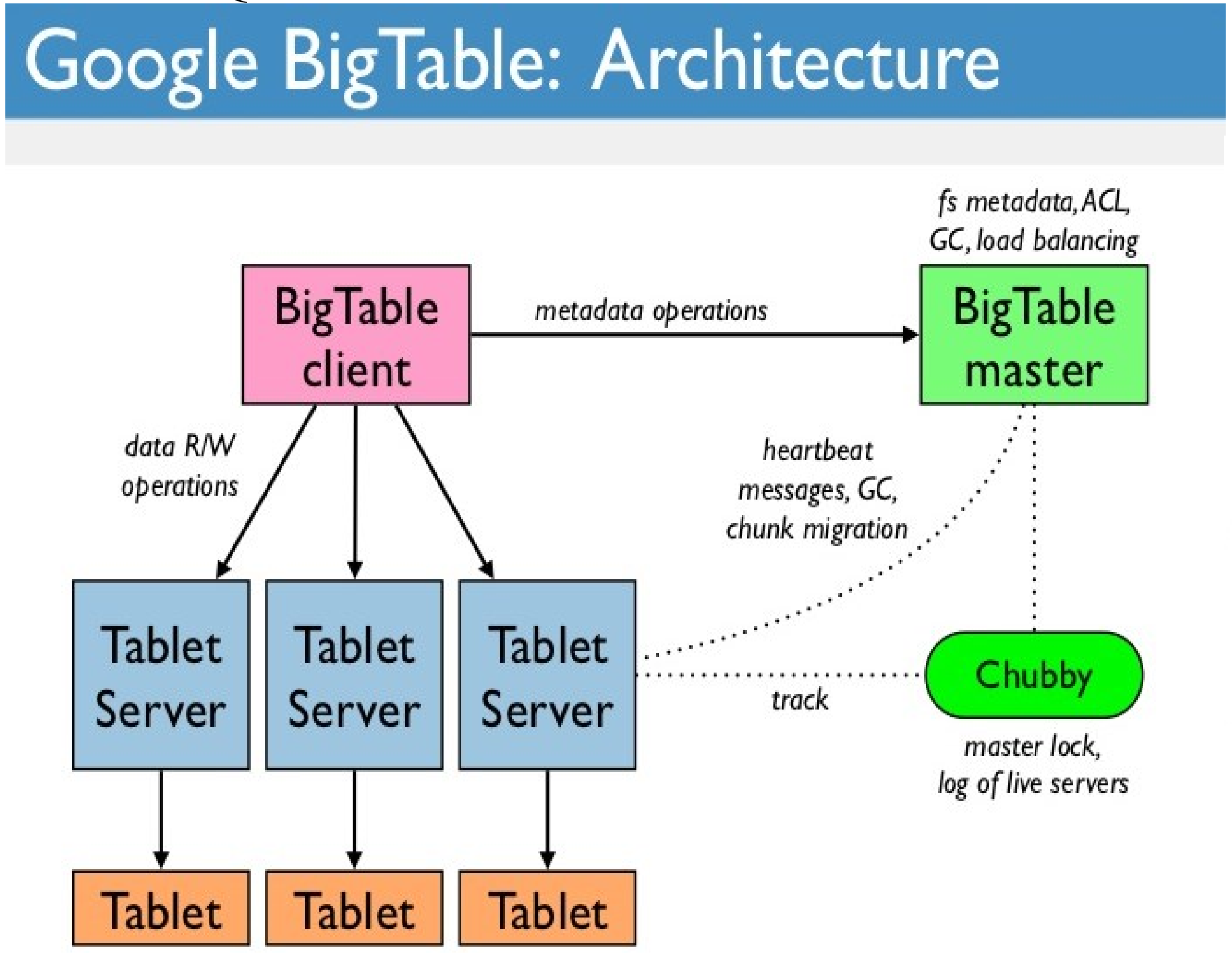
2 Data Model

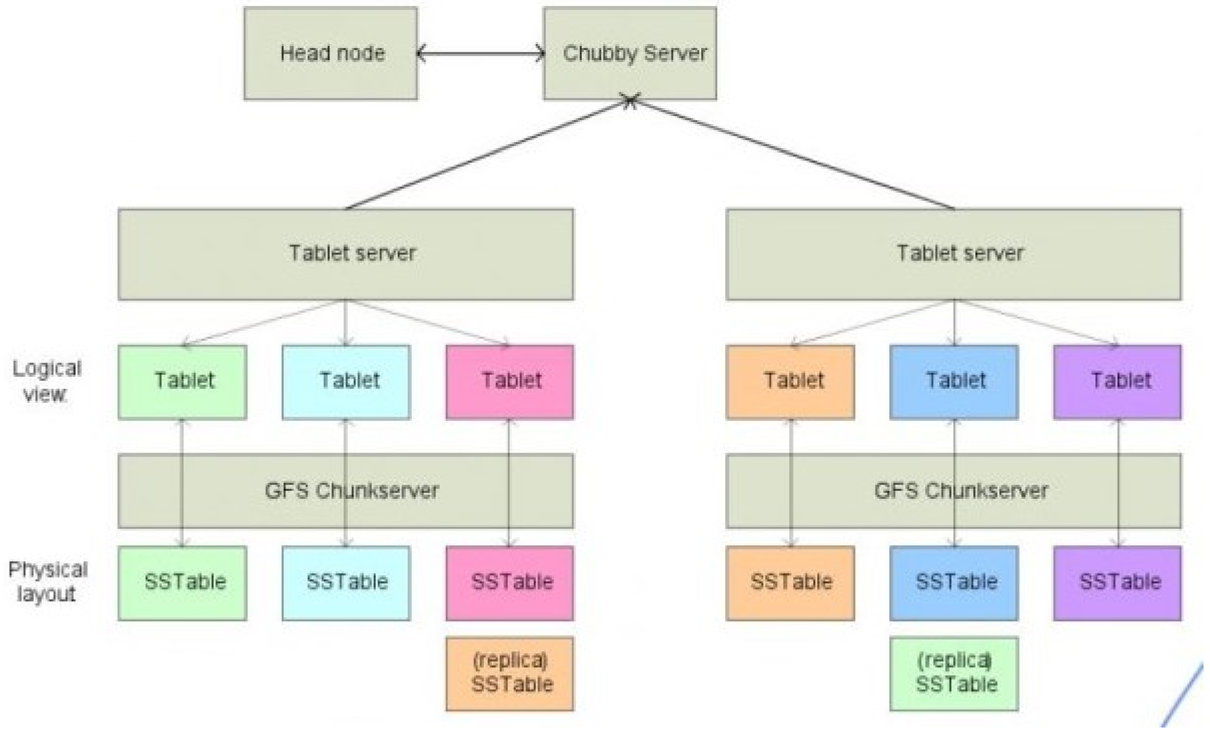
[illegible]

[illegible]

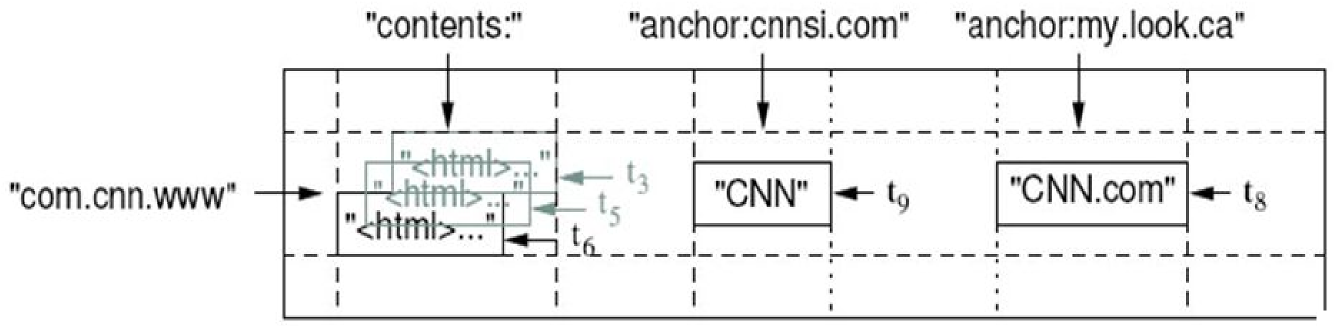
1

- Bigtable is designed to be distributed on commodity servers, a common theme for all NoSQL databases.

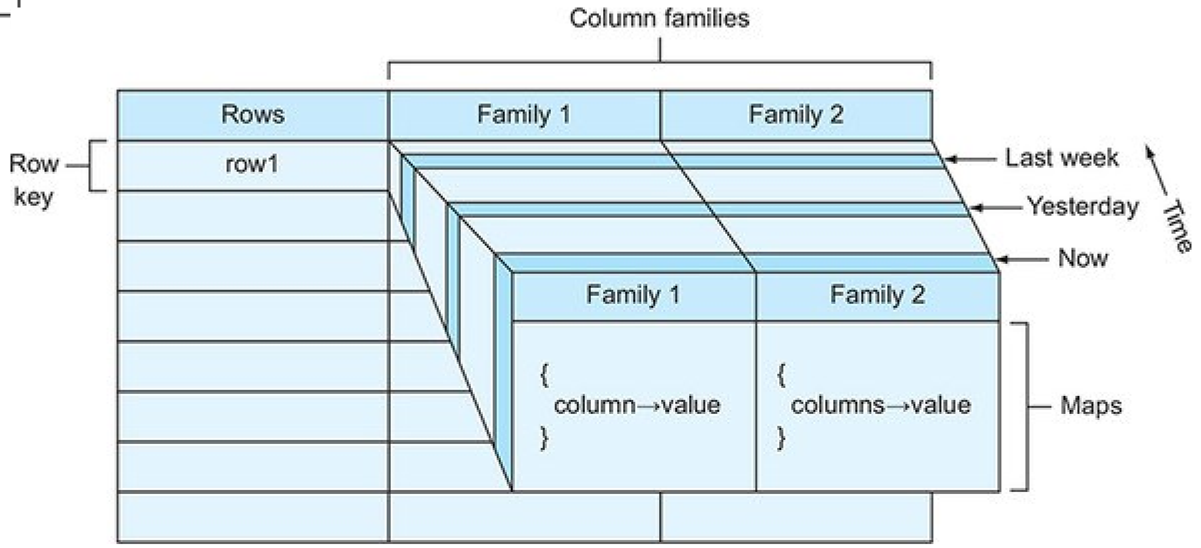

Google BigTable: Architecture
fs metadata, ACL, GC, load balancing
BigTable client
metadata operations
BigTable master
data R/W operations
heartbeat messages, GC, chunk migration
Tablet Server
Tablet Server
Tablet Server
track
Chubby
master lock, log of live servers
Tablet
Tablet
Tablet



Google's BigTable Example



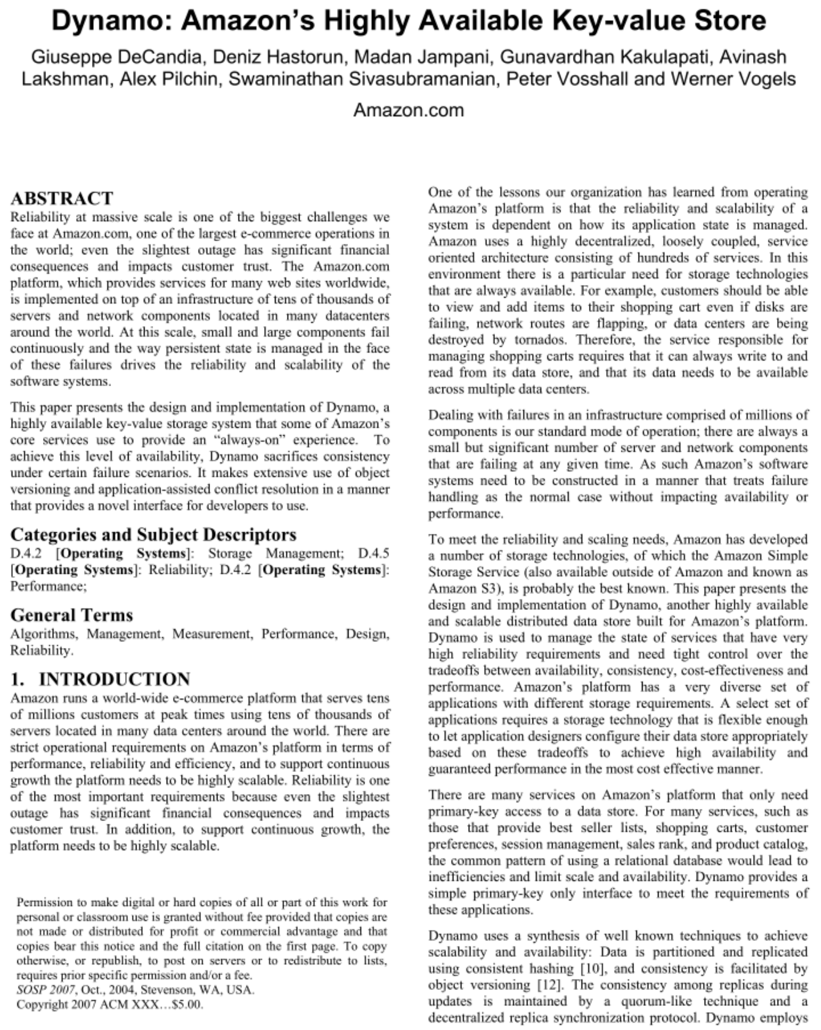
- Use URLs as row keys
- Various aspects of web page as column names
- Store contents of web pages in the `contents:` column under the timestamps when they were fetched. The anchor column family contains the text of any anchors that reference the page.
- Column keys are grouped into sets called column families, which form the basic unit of access control.
- All data stored in a column family is usually of the same type (and can be compressed together).



JAIN DEEMED-TO-BE UNIVERSITY SCHOOL OF COMPUTER SCIENCE AND IT

Amazon's Dynamo Paper

- In 2007, Amazon released it's Dynamo data storage application.
- Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performance.
- It describes:
  - how a lot of Amazon data is stored by use of a primary key
  - how consistent hashing is used to partition and distribute data
  - how object versioning is used to maintain consistency across data centers
- The keys are logical IDs, and the values can be any binary value of interest to the developer.



Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels
Amazon.com

ABSTRACT
Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

Categories and Subject Descriptors
D.4.2 [**Operating Systems**]: Storage Management; D.4.5 [**Operating Systems**]: Reliability; D.4.2 [**Operating Systems**]: Performance;

General Terms
Algorithms, Management, Measurement, Performance, Design, Reliability.

1. INTRODUCTION
Amazon runs a world-wide e-commerce platform that serves tens of millions customers at peak times using tens of thousands of servers located in many data centers around the world. There are strict operational requirements on Amazon's platform in terms of performance, reliability and efficiency, and to support continuous growth the platform needs to be highly scalable. Reliability is one of the most important requirements because even the slightest outage has significant financial consequences and impacts customer trust. In addition, to support continuous growth, the platform needs to be highly scalable.

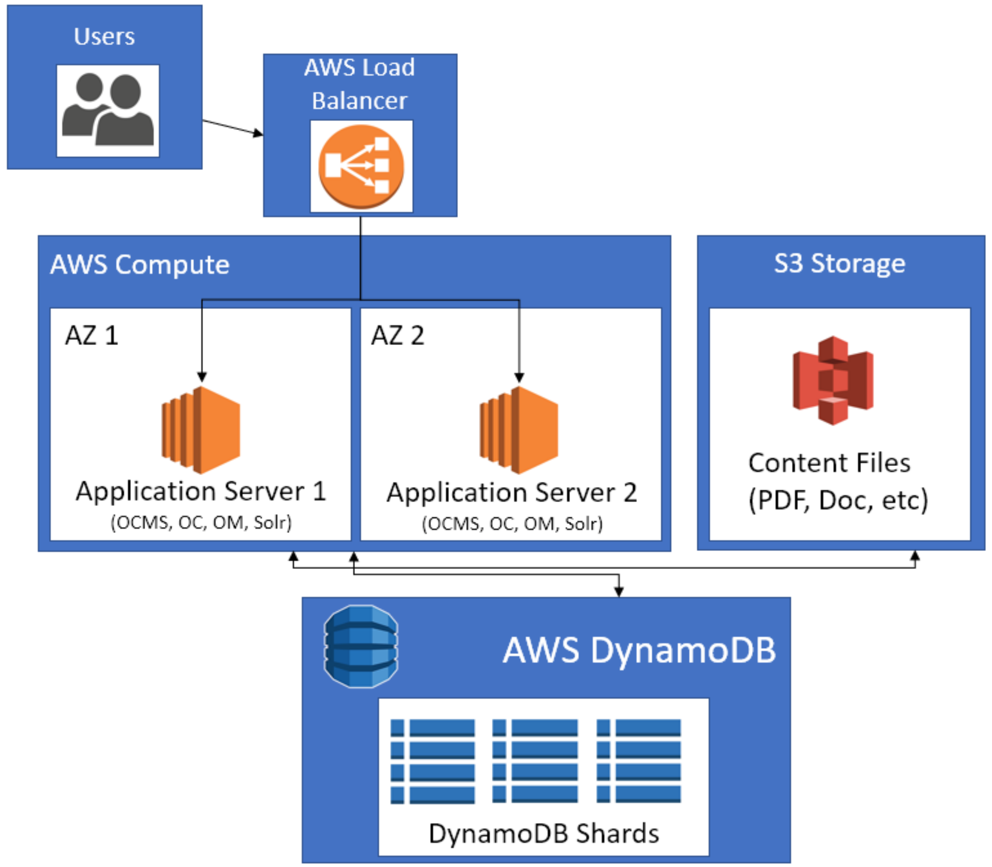
One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

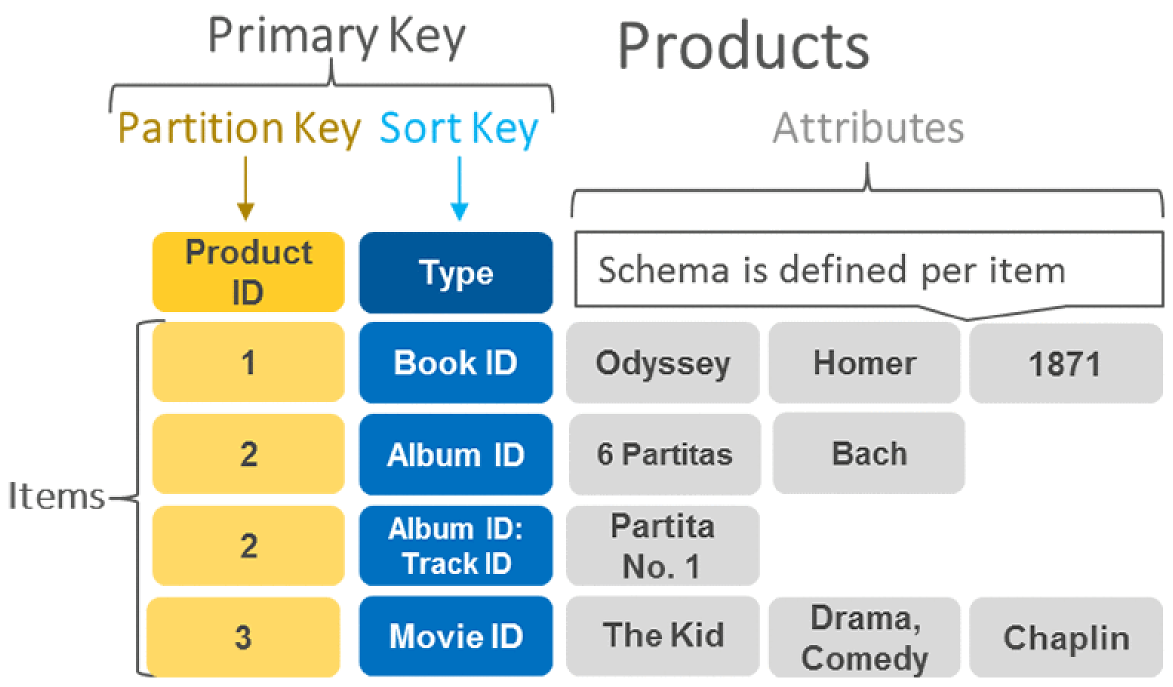
Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon's software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

To meet the reliability and scaling needs, Amazon has developed a number of storage technologies, of which the Amazon Simple Storage Service (also available outside of Amazon and known as Amazon S3), is probably the best known. This paper presents the design and implementation of Dynamo, another highly available and scalable distributed data store built for Amazon's platform. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performance. Amazon's platform has a very diverse set of applications with different storage requirements. A select set of applications requires a storage technology that is flexible enough to let application designers configure their data store appropriately based on these tradeoffs to achieve high availability and guaranteed performance in the most cost effective manner.

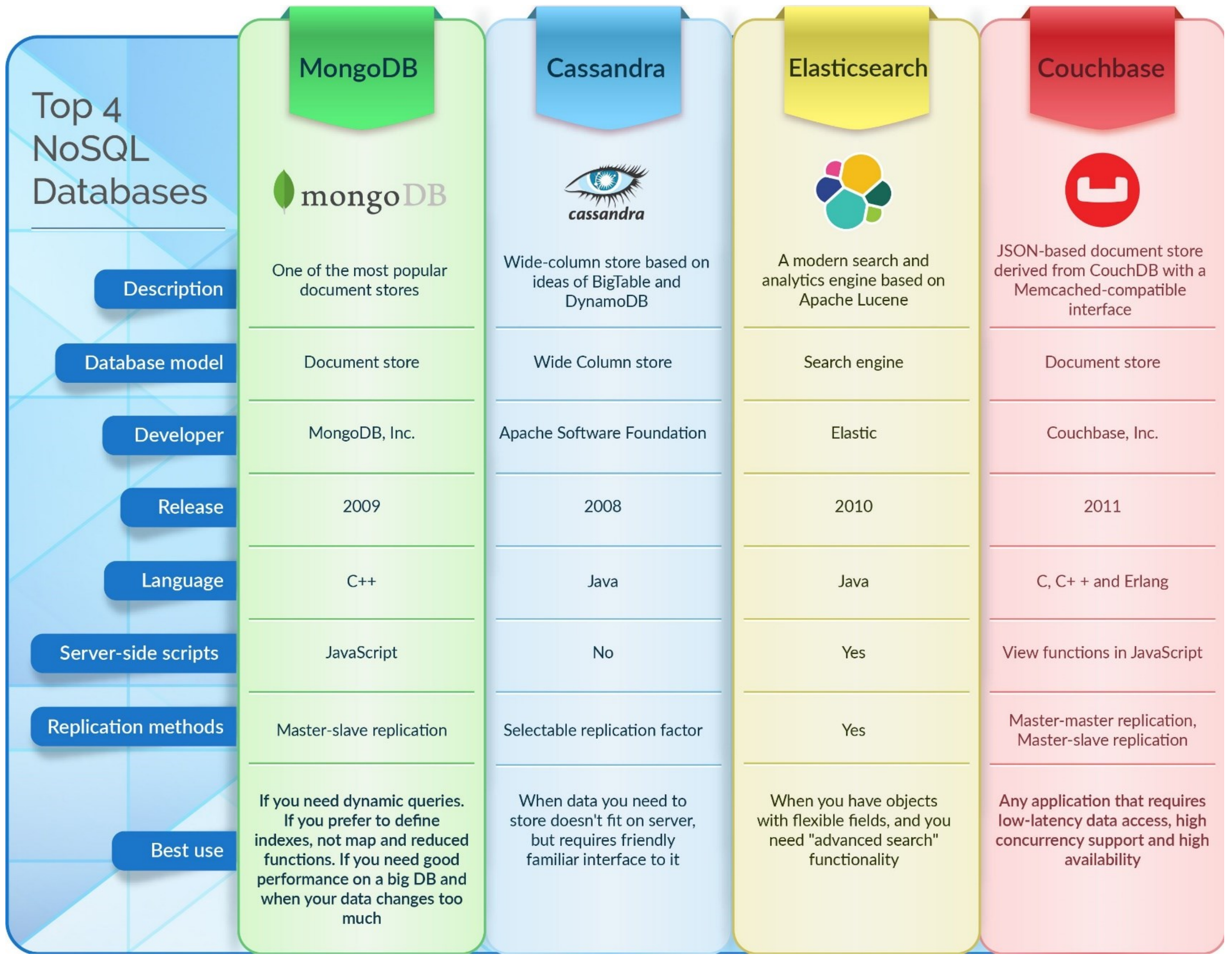
There are many services on Amazon's platform that only need primary-key access to a data store. For many services, such as those that provide best seller lists, shopping carts, customer preferences, session management, sales rank, and product catalog, the common pattern of using a relational database would lead to inefficiencies and limit scale and availability. Dynamo provides a simple primary-key only interface to meet the requirements of these applications.

Dynamo uses a synthesis of well known techniques to achieve scalability and availability: Data is partitioned and replicated using consistent hashing [10], and consistency is facilitated by object versioning [12]. The consistency among replicas during updates is maintained by a quorum-like technique and a decentralized replica synchronization protocol. Dynamo employs


Users
AWS Load Balancer
AWS Compute
AZ 1
Application Server 1
(OCMS, OC, OM, Solr)
AZ 2
Application Server 2
(OCMS, OC, OM, Solr)
S3 Storage
Content Files (PDF, Doc, etc)
AWS DynamoDB
DynamoDB Shards


Primary Key
Products
Partition Key
Sort Key
Attributes
Product ID
Type
Schema is defined per item
Items
1
Book ID
Odyssey
Homer
1871
2
Album ID
6 Partitas
Bach
2
Album ID: Track ID
Partita No. 1
3
Movie ID
The Kid
Drama, Comedy
Chaplin

- Today NoSQL includes hundreds of NoSQL database products.
- The reason behind these explosion of databases are:
  - Non-relational approaches have been applied to a wide range of problems where an RDBMS has traditionally been weak
  - NoSQL databases were also created for data structures and models that in an RDBMS required considerable management or shredding and the reconstitution of data in complex plumbing code



Top 4 NoSQL Databases

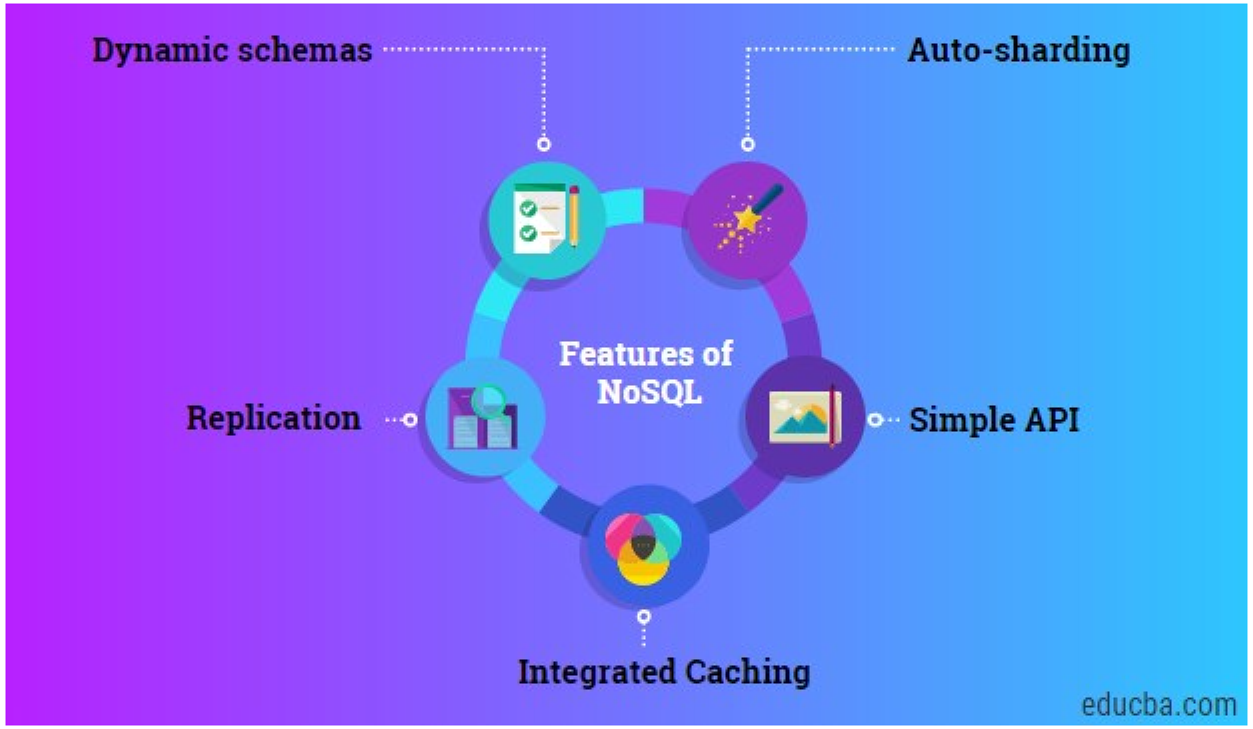
	MongoDB	Cassandra	Elasticsearch	Couchbase
Description	One of the most popular document stores	Wide-column store based on ideas of BigTable and DynamoDB	A modern search and analytics engine based on Apache Lucene	JSON-based document store derived from CouchDB with a Memcached-compatible interface
Database model	Document store	Wide Column store	Search engine	Document store
Developer	MongoDB, Inc.	Apache Software Foundation	Elastic	Couchbase, Inc.
Release	2009	2008	2010	2011
Language	C++	Java	Java	C, C+ + and Erlang
Server-side scripts	JavaScript	No	Yes	View functions in JavaScript
Replication methods	Master-slave replication	Selectable replication factor	Yes	Master-master replication, Master-slave replication
Best use	If you need dynamic queries. If you prefer to define indexes, not map and reduced functions. If you need good performance on a big DB and when your data changes too much	When data you need to store doesn't fit on server, but requires friendly familiar interface to it	When you have objects with flexible fields, and you need "advanced search" functionality	Any application that requires low-latency data access, high concurrency support and high availability

SCHOOL OF COMPUTER SCIENCE AND IT

Features of NoSQL

- The common features of NoSQL are:

1. Schema agnostic
2. Non-relational
3. Commodity Hardware
4. Highly Distributable



- **Schema agnostic**:

A database schema is the description of all possible data and data structures in a relational database. With a NoSQL database, a schema isn't required, giving you the freedom to store information without doing up-front schema design.

  – You can start coding and store and retrieve data without knowing how the database stores and works internally.

  – Schema agnosticism may be the most significant difference between NoSQL and relational databases.

  – You need to know how the data is stored only when constructing a query.

  – The great benefit to a schema agnostic database is that development time is shortened.

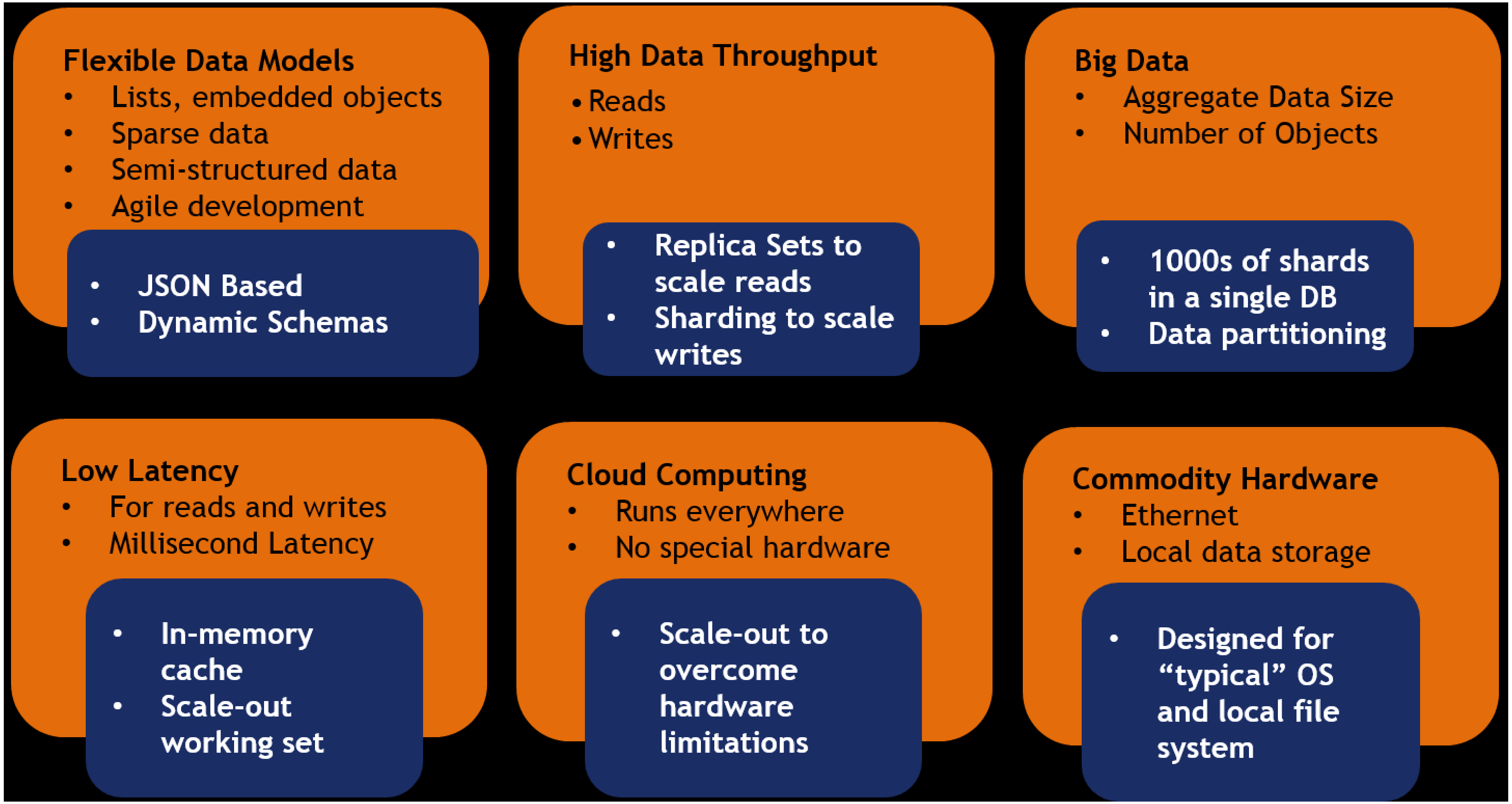
- **Nonrelational:**
- Relations in a database establish connections between tables of data. For example, a list of transaction details can be connected to a separate list of delivery details.
- With a NoSQL database, this information is stored as an aggregate — a single record with everything about the transaction, including the delivery address.

- **Commodity hardware:**
- Some databases are designed to operate best with specialized storage and processing hardware.
- With a NoSQL database, cheap off-the-shelf servers can be used.
- Adding more of these cheap servers allows NoSQL databases to scale to handle more data.

- **Highly distributable:**
- Distributed databases can store and process a set of information on more than one device.
- With a NoSQL database, a cluster of servers can be used to hold a single large database.

In general, NoSQL databases have the following properties:

1. They have higher scalability.
2. They use distributed computing.
3. They are cost effective.
4. They support flexible schema.
5. They're able to process both unstructured and semi-structured data.
6. There are no complex relationships, such as the ones between tables in an RDBMS.



Flexible Data Models
- Lists, embedded objects
- Sparse data
- Semi-structured data
- Agile development

- JSON Based
- Dynamic Schemas

High Data Throughput
- Reads
- Writes

- Replica Sets to scale reads
- Sharding to scale writes

Big Data
- Aggregate Data Size
- Number of Objects

- 1000s of shards in a single DB
- Data partitioning

Low Latency
- For reads and writes
- Millisecond Latency

- In-memory cache
- Scale-out working set

Cloud Computing
- Runs everywhere
- No special hardware

- Scale-out to overcome hardware limitations

Commodity Hardware
- Ethernet
- Local data storage

- Designed for “typical” OS and local file system

BASE versus ACID

- ACID-compliant transaction means the database is designed so it absolutely will not lose data:

✓▶Each operation moves the database from one valid state to another (Atomic).

✓▶Everyone has the same view of the data at any point in time (Consistent).

✓▶Operations on the database don’t interfere with each other (Isolation).

✓▶When a database says it has saved data, you know the data is safe (Durable).

Not many NoSQL databases have ACID transactions.

Exceptions to that norm are FoundationDB, Neo4j, and MarkLogic Server, which do provide fully serializable ACID transactions.

SCHOOL OF COMPUTER SCIENCE AND IT

Problems with conventional approaches

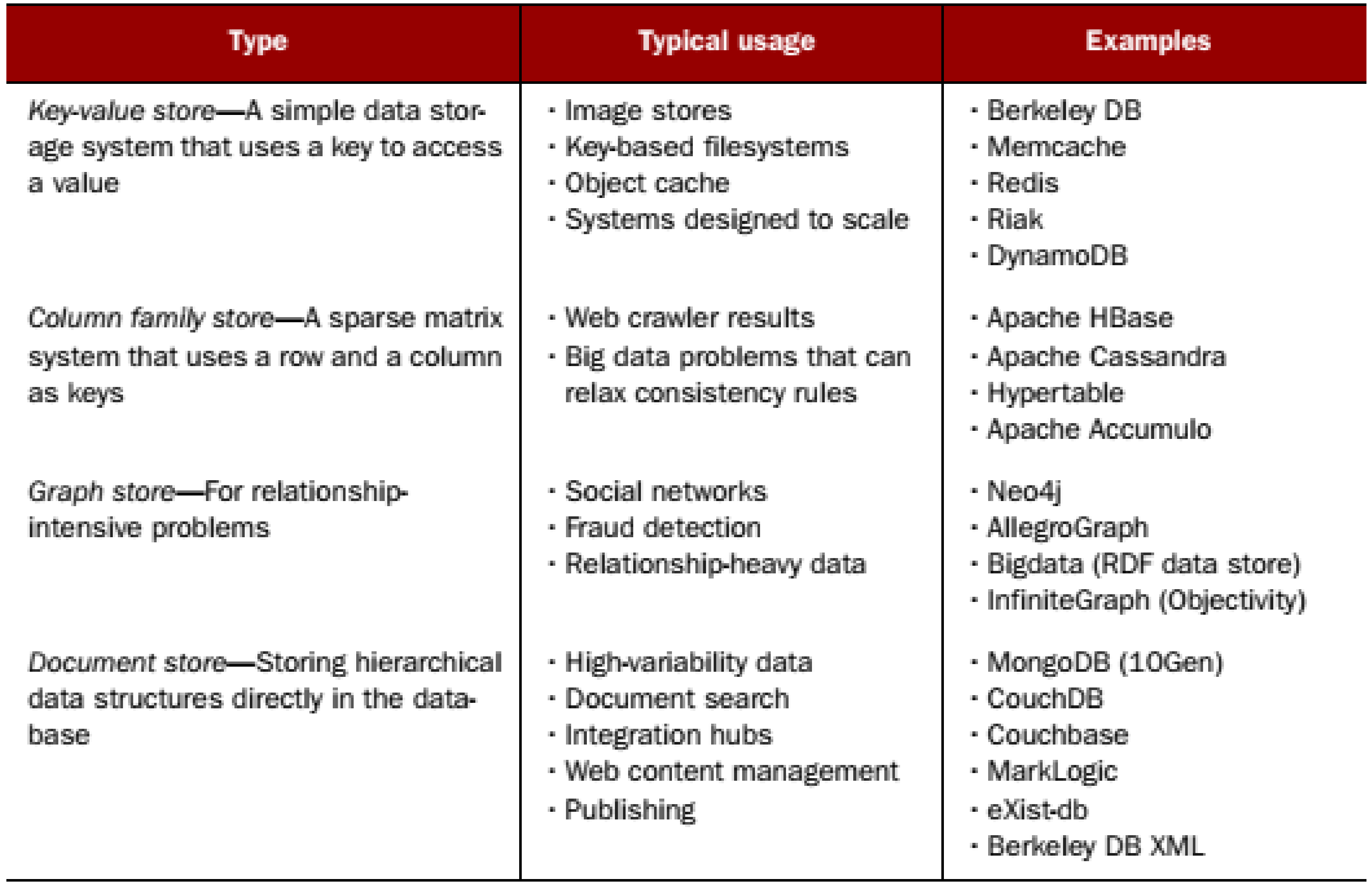
- Relational databases are great for things that fit easily into rows and columns.
- Not everything fits well into rows and columns
- Breaking this out into another sheet or table is a bit of overkill, and makes it harder to work with the data as a single unit.
- RDBMS just relates records in tables using structures about the relationships known at design time.
- Each of the preceding scenarios has a type of NoSQL database that overcomes the limitations of an RDBMS for those data types: key-value, columnar, and triple stores, respectively.

Problems with conventional approaches

- Schema redesign overhead
- Unstructured data explosion
- The sparse data problem
- Dynamically changing relationships
- Global distribution and access

SCHOOL OF COMPUTER SCIENCE AND IT

Table 1.1 Types of NoSQL data stores—the four main categories of NoSQL systems, and sample products for each data store type



Type	Typical usage	Examples
Key-value store—A simple data storage system that uses a key to access a value	· Image stores · Key-based filesystems · Object cache · Systems designed to scale	· Berkeley DB · Memcache · Redis · Riak · DynamoDB
Column family store—A sparse matrix system that uses a row and a column as keys	· Web crawler results · Big data problems that can relax consistency rules	· Apache HBase · Apache Cassandra · Hypertable · Apache Accumulo
Graph store—For relationship-intensive problems	· Social networks · Fraud detection · Relationship-heavy data	· Neo4j · AllegroGraph · Bigdata (RDF data store) · InfiniteGraph (Objectivity)
Document store—Storing hierarchical data structures directly in the database	· High-variability data · Document search · Integration hubs · Web content management · Publishing	· MongoDB (10Gen) · CouchDB · Couchbase · MarkLogic · eXist-db · Berkeley DB XML

NoSQL business drivers

- “paradigm shift”-NoSQL movement and the changes in thought patterns, architectures, and methods emerging today.

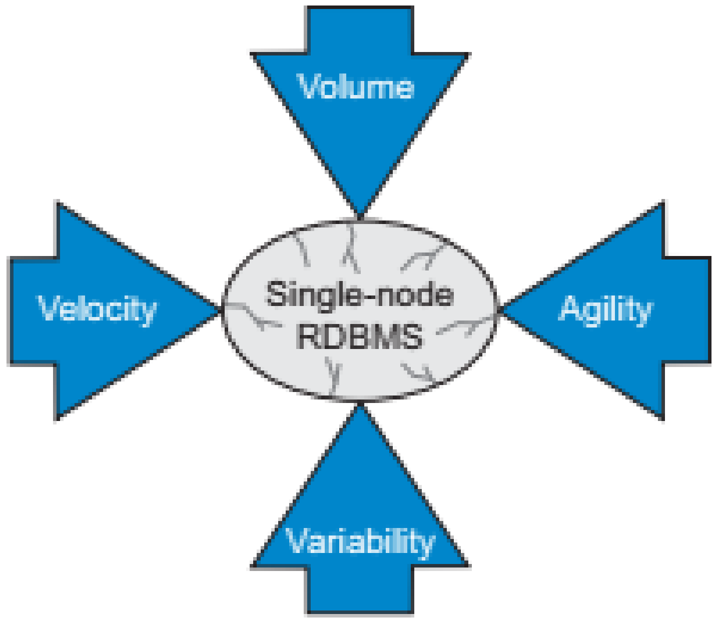
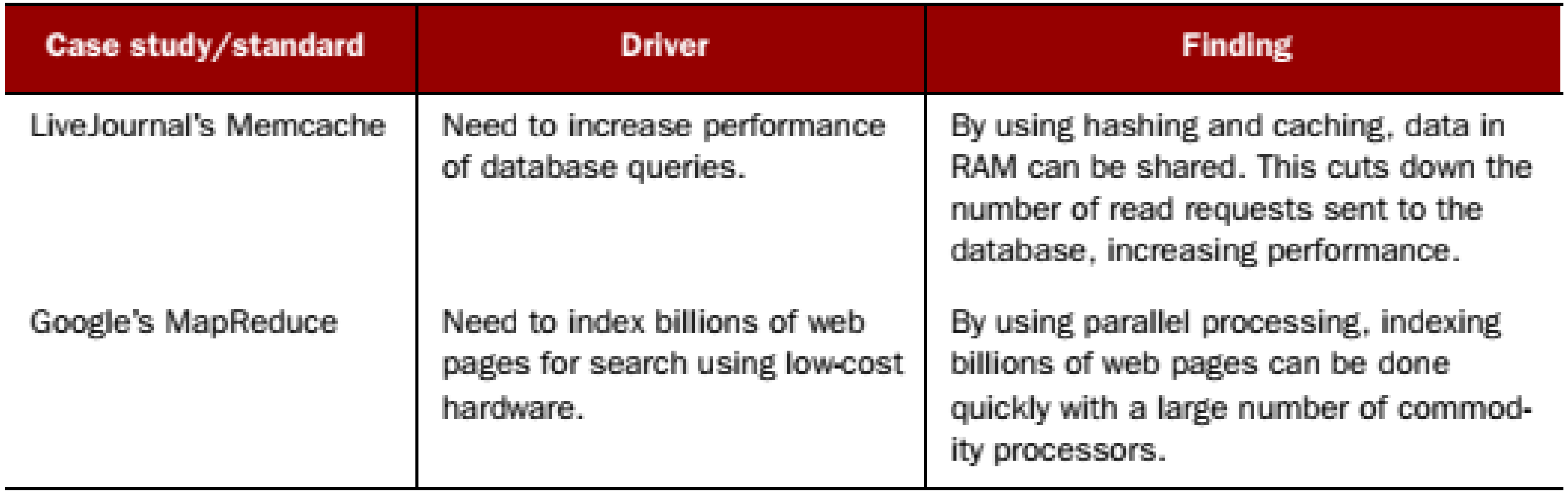


Figure 1.1 In this figure, we see how the business drivers volume, velocity, variability, and agility apply pressure to the single CPU system, resulting in the cracks. Volume and velocity refer to the ability to handle large datasets that arrive quickly. Variability refers to how diverse data types don't fit into structured tables, and agility refers to how quickly an organization responds to business change.

NoSQL business drivers

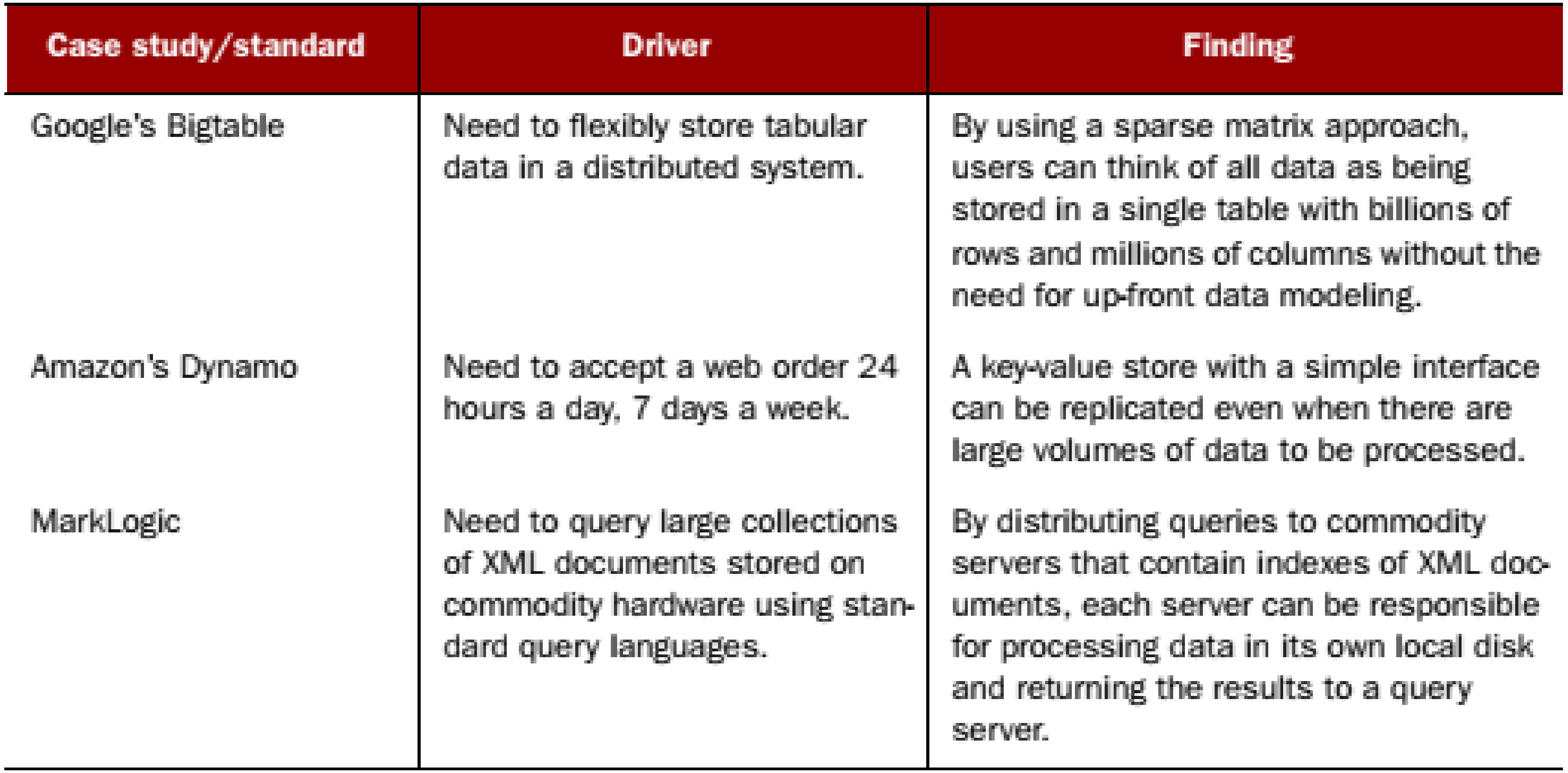
- Volume
- Velocity
- Variability
- Agility

NoSQL case studies



Case study/standard	Driver	Finding
LiveJournal's Memcache	Need to increase performance of database queries.	By using hashing and caching, data in RAM can be shared. This cuts down the number of read requests sent to the database, increasing performance.
Google's MapReduce	Need to index billions of web pages for search using low-cost hardware.	By using parallel processing, indexing billions of web pages can be done quickly with a large number of commodity processors.

NoSQL case studies



Case study/standard	Driver	Finding
Google's Bigtable	Need to flexibly store tabular data in a distributed system.	By using a sparse matrix approach, users can think of all data as being stored in a single table with billions of rows and millions of columns without the need for up-front data modeling.
Amazon's Dynamo	Need to accept a web order 24 hours a day, 7 days a week.	A key-value store with a simple interface can be replicated even when there are large volumes of data to be processed.
MarkLogic	Need to query large collections of XML documents stored on commodity hardware using standard query languages.	By distributing queries to commodity servers that contain indexes of XML documents, each server can be responsible for processing data in its own local disk and returning the results to a query server.

NoSQL case studies

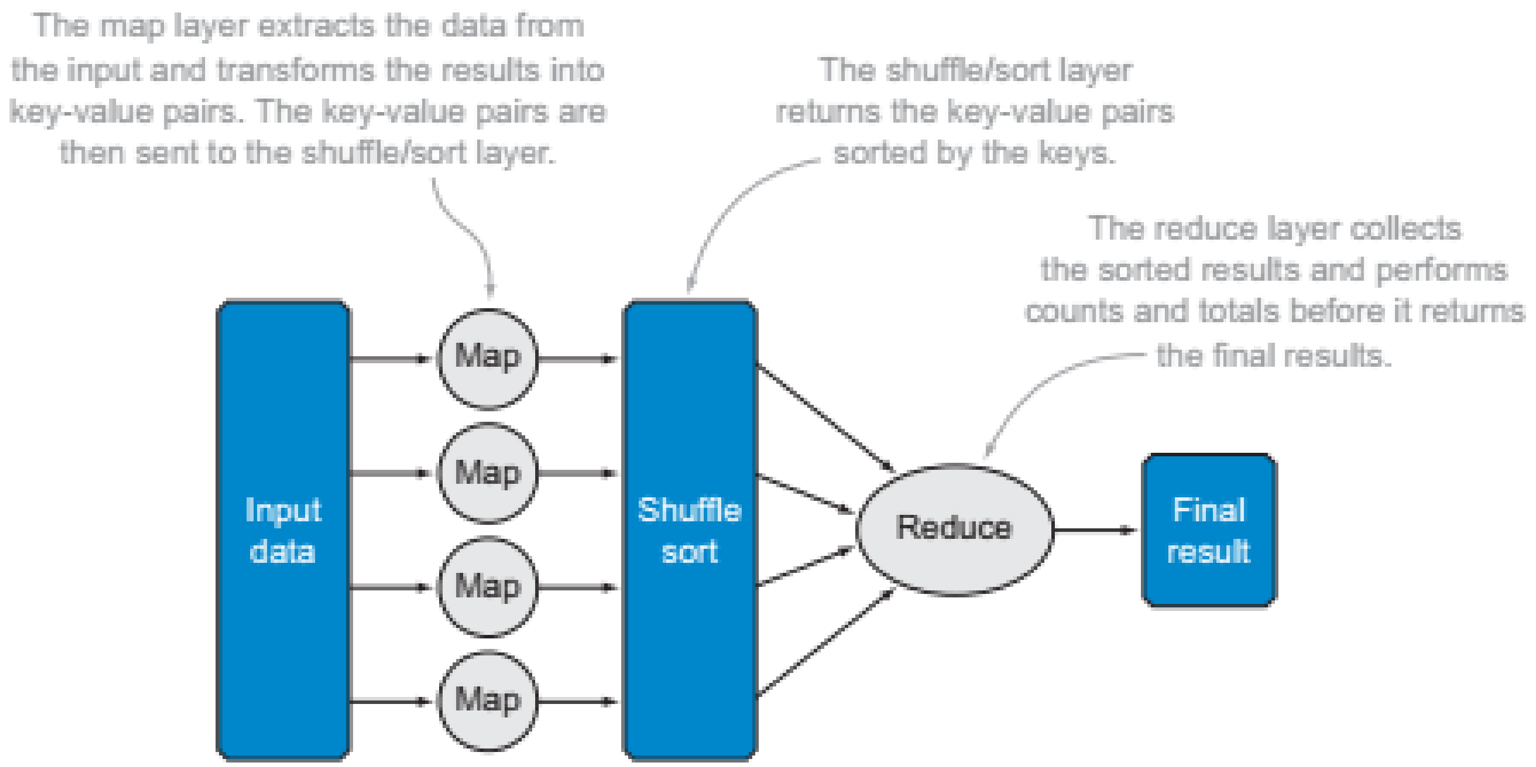

The map layer extracts the data from the input and transforms the results into key-value pairs. The key-value pairs are then sent to the shuffle/sort layer.
The shuffle/sort layer returns the key-value pairs sorted by the keys.
The reduce layer collects the sorted results and performs counts and totals before it returns the final results.
Input data
Map
Map
Map
Map
Shuffle sort
Reduce
Final result

Figure 1.2 The map and reduce functions are ways of partitioning large datasets into smaller chunks that can be transformed on isolated and independent transformation systems. The key is isolating each function so that it can be scaled onto many servers.

Types of NoSQL Databases

- NoSQL databases come in four core types — one for each type of data the database is expected to manage:
  - **Columnar**: Extension to traditional table structures. Supports variable sets of columns (column families) and is optimized for column-wide operations (such as count, sum, and mean average).
  - **Key-value**: A very simple structure. Sets of named keys and their value(s), typically an uninterpreted chunk of data. Sometimes that simple value may in fact be a JSON or binary document.

Types of NoSQL Databases

– **Triple:** A single fact represented by three elements:
  - The subject you're describing
  - The name of its property or relationship to another subject
  - The value — either an intrinsic value (such as an integer) or the unique ID of another subject (if it's a relationship)

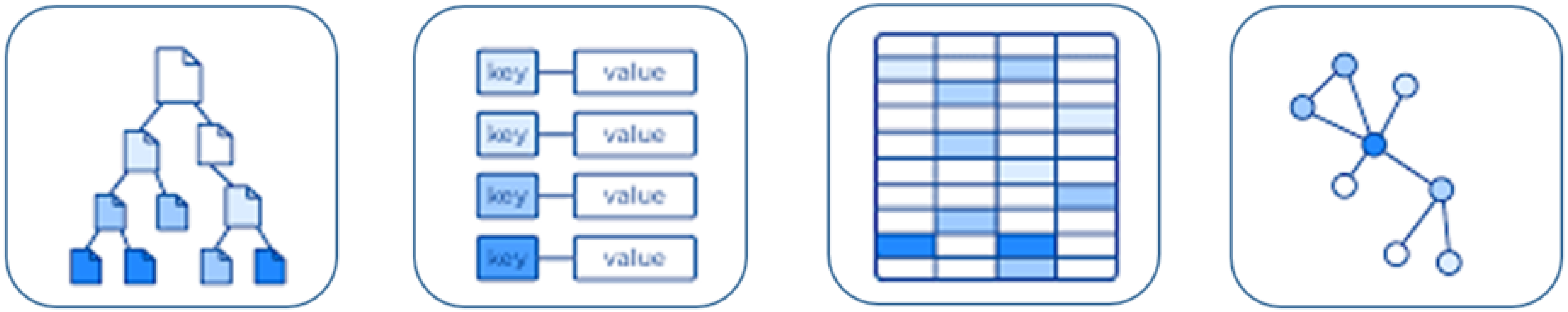
– **Document**: XML, JSON, text, or binary blob. Any treelike structure can be represented as an XML or JSON document, including things such as an order that includes a delivery address, billing details, and a list of products and quantities.

Some document NoSQL databases support storing a separate list (or document) of properties about the document, too.

Types of NoSQL Databases

– **Search engines**: If you're storing information that has a variable structure or copious text, you need a common way across structures to find relevant information, which search engines provide.

– **Hybrid NoSQL databases:** These databases provide a mix of the core features of multiple NoSQL database types — such as key-value, document, and triple stores — all in the same product..

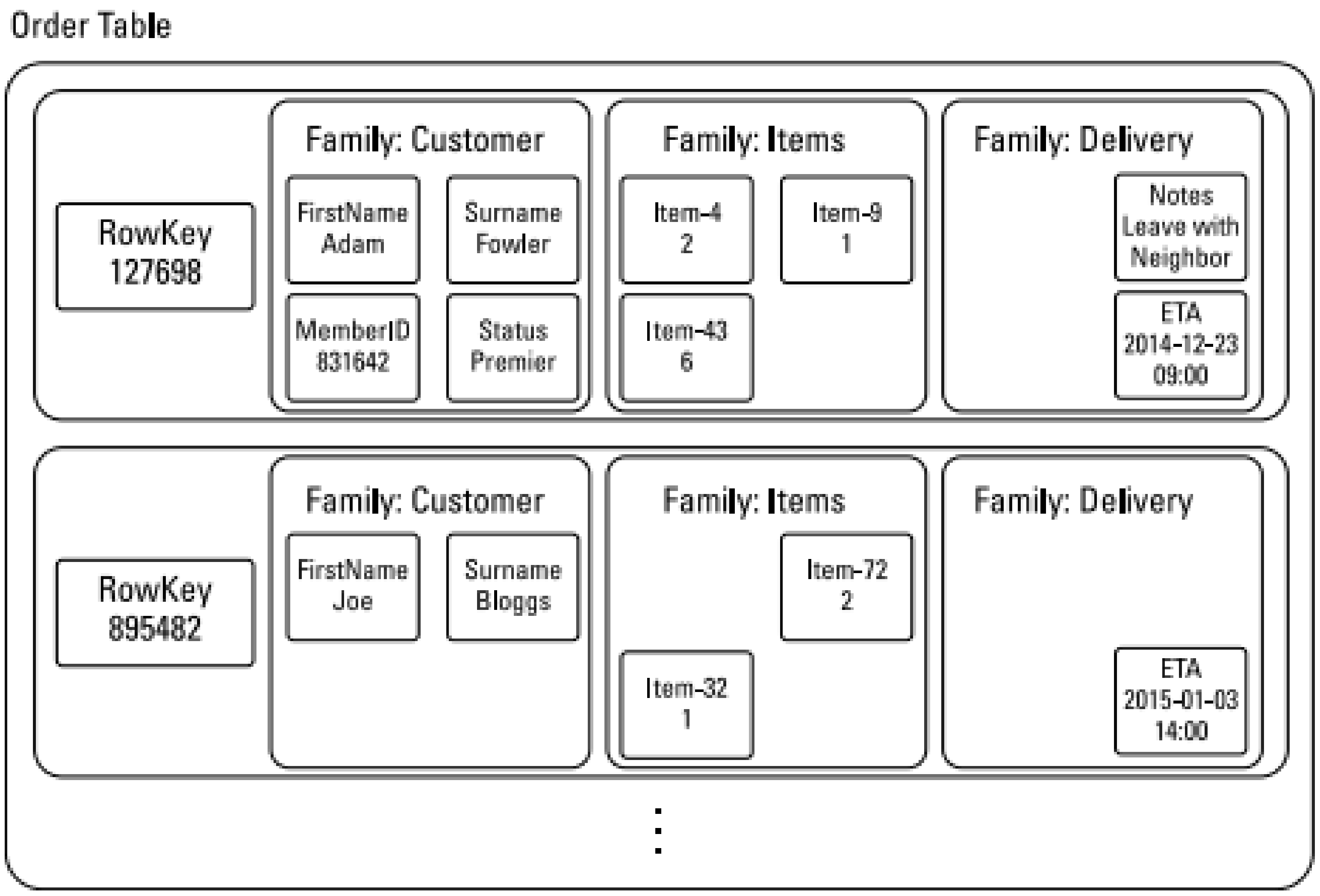


SCHOOL OF COMPUTER SCIENCE AND IT

Columnar

- Column stores are similar at first appearance to traditional relational DBMS
- Instead of storing data in a row for fast access, data is organized for fast column operations.
- This column-centric view makes column stores ideal for running aggregate functions or for looking up records that match multiple columns.
- Aggregate functions are data combinations or analysis functions.
- Column stores are also sometimes referred to as Big Tables or Big Table clones, reflecting their common ancestor, Google's Bigtable.
- A key benefit of a column store over an RDBMS is that column stores don't require fields to always be present and don't require a blank padding null value like an RDBMS does. This feature prevents the sparse data problem.

Columnar


Order Table
RowKey 127698
Family: Customer
FirstName Adam
Surname Fowler
MemberID 831642
Status Premier
Family: Items
Item-4 2
Item-9 1
Item-43 6
Family: Delivery
Notes Leave with Neighbor
ETA 2014-12-23 09:00
RowKey 895482
Family: Customer
FirstName Joe
Surname Bloggs
Family: Items
Item-72 2
Item-32 1
Family: Delivery
ETA 2015-01-03 14:00

SCHOOL OF COMPUTER SCIENCE AND IT

Columnar

- The great thing about column stores is that you can retrieve all related information using a single record ID, rather than using the complex SQL join as in an RDBMS.
- So, for complex and variable relational data structures, a column store may be more efficient in storage and less error prone in development than its RDBMS ancestors.
- If you know the data fields involved up front and need to quickly retrieve related data together as a single record, then consider a column store.

SCHOOL OF COMPUTER SCIENCE AND IT

Key-value stores

- Key-value stores also have a record with an ID field — the key in key-value stores — and a set of data. This data can be one of the following:

✓An arbitrary piece of data that the application developer interprets (as opposed to the database)

✓Any set of name-value pairs (called bins)

- Key-value stores are the simplest type of storage in the NoSQL world
- Some key-value stores support typing (such as integers, strings, and Booleans) and more complex structures for values (such as maps and lists).
- This setup aids developers because they don’t have to hand-code or decode string data held in a key-value store.

Key-value stores

- Maps are a simple type of key-value storage.
- A unique key in a map has a single arbitrary value associated with it.
- The value could be a list of another map. So, it's possible to store tree structures within key-value stores.
- If you have ***numerous maps in your key-value store***, consider a document store instead, which will likely minimize the amount of code required to operate on your data and make search and retrieval easier.
- Key-value stores are optimized for speed of ingestion and retrieval.
- If you need ***very high ingest speed on a limited numbers of nodes*** and can afford to sacrifice complex ad hoc query support, then a key-value store may be for you.

Triple and graph stores

- Every fact (or more correctly, assertion) is described as a triple of subject, predicate, and object:

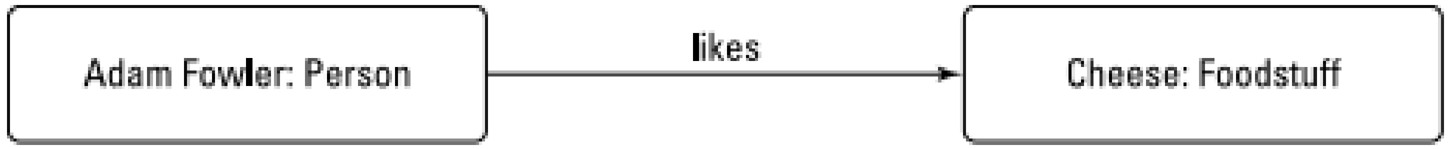
▶A *subject* is the thing you're describing. It has a unique ID called an IRI. It may also have a type, which could be a physical object (like a person) or a concept (like a meeting).

▶A *predicate* is the property or relationship belonging to the subject. This again is a unique IRI that is used for all subjects with this property.

▶An *object* is the intrinsic value of a property (such as integer or Boolean, text) or another subject IRI for the target of a relationship

Triple and graph stores

- Graph math is complex and specialized and may not be required in all situations where storing triples are required.
- The directed graphs can contain complex and changing webs of relationships, or triples.
- Being able to store and query them efficiently, either on their own or as part of a larger multi-data structure application, is very useful for solving particular data storage and analytics problems.
- If you need to store facts, dynamically changing relationships, or provenance information, then consider a triple store.
- If you need to know statistics about the graph (such as how many degrees of separation are between two subjects or how many third level social connections a person has), then you should consider a graph store.



```
AdamFowler is_a Person
AdamFowler likes Cheese
Cheese is_a Foodstuff
```

More accurately, though, such triple information is conveyed with full IRI information in a format such as Turtle, like this:

```
<http://www.mydomain.org/people#AdamFowler> a <http://www.mydomain.
            org/rdftypes#Person> .
<http://www.mydomain.org/people#AdamFowler> <http://www.mydomain.
            org/predicates#likes> <http://www.mydomain.org/
            foodstuffs#Cheese> .
<http://www.mydomain.org/foodstuffs#Cheese> a <http://www.mydomain.
            org/rdftypes#Foodstuff> .
```

Document

- Document databases are sometimes called aggregate databases because they tend to hold documents that combine information in a single logical unit — an aggregate.
- Retrieving all information from a single document is easier with a database (no complex joins as in an RDBMS) and is more logical for applications (less complex code).
- Loosely, a document is any unstructured or tree-structured piece of information.
- A table, for example, can be modeled as a very flat XML document –i.e, one with only a single set of elements, and no sub-element hierarchies.
- A set of triples (aka subgraph) can be stored within a single document, or across documents, too.
- The utility of doing so depends, of course, on the indexing and query mechanisms supported.

Search engines

- Many search engines are even capable of acting as a key-value or document store in their own right.
- Their indexes and query processing are highly distributed.
- NoSQL databases are often used to store unstructured data, documents, or data that may be stored in a variety of structures, such as social media posts or web pages.
- Search engines are great for storing many structures in a single database; necessitates a way to provide a standard query mechanism over all content.
- Consider search as a key requirement to unstructured data management with NoSQL Document databases.

Hybrid NoSQL databases

- Hybrid databases can easily handle document and key-value storage needs, while also allowing fast aggregate operations similar to how column stores work.
- Typically, this goal is achieved by using search engine term indexes, rather than tabular field indexes within a table column in the database schema design itself.
- The functionality provided, though, is often the same as in column stores.
- So, these products have three or four of the preceding types covered: key-value, document, and column stores, as well as search engines.

NoSQL products

1. **Columnar**: *DataStax, Apache Cassandra, HBase, Apache Accumulo, Hypertable*
2. **Key-value**: *Basho Riak, Redis, Voldemort, Aerospike, Oracle NoSQL*
3. **Triple/graph**: *Neo4j, Ontotext's GraphDB (formerly OWLIM), MarkLogic, OrientDB, AllegroGraph, YarcData*
4. **Document**: *MongoDB, MarkLogic, CouchDB, FoundationDB, IBM Cloudant, Couchbase*
5. **Search engine**: *Apache Solr, Elasticsearch, MarkLogic*
6. **Hybrid**: *OrientDB, MarkLogic, ArangoDB*

Describing NoSQL: KEY TERMS

❖ DATABASE CONSTRUCTION

- ***Database:*** A single logical unit, potential spread over multiple machines, into which data can be added and that can be queried for data it contains. The relational term tablespace could also be applied to a NoSQL database or collection.
- ***Data farm:*** A term from RDBMS referring to a set of read-only replica sets stored across a managed cluster of machines. In an RDBMS, these typically can't have machines added without down time. In NoSQL clusters, it's desirable to quickly scale out.
- ***Partition***: A set of data to be stored together on a single node for processing efficiency, or to be replicated. Could also be used for querying. In this case, it can be thought of as a collection.

❖ DATABASE STRUCTURE

- ***Collection:*** A set of records, typically documents, that are grouped together. This is based not on a property within the record set, but within its metadata. Assigning a record to a collection is usually done at creation or update time.
- ***Schema:*** In RDBMS and to a certain extent column stores. The structure of the data must be configured in the database before any data is loaded.

In document databases, although any structure can be stored, it is sometimes better to limit the structures by enforcing schema, such as in an XML Schema Definition. NoSQL generally, though, is regarded as schema-free, or as supporting variable schema.

❖ RECORDS

- Record: A single atomic unit of data representation in the particular database being described. It is a *row in column stores*, a *value in a key-value store*, a *document in a document store*, or *a subject (not triple) in a triple store*.
- Row: Atomic unit of record in an RDBMS or column store. Could be modeled as *an element within a document store* or as a *map in a key-value store*.
- Field: A single field within a record. A column in an RDBMS. May not be present in all records, but when present should be of the same type or structure.
- Table: A single class of record. In *Bigtable*, they are also called *tables*. In a *triple store*→subject RDF types or named be graphs, depending on the context. In a document store→collections.

❖ RECORD ASSOCIATIONS

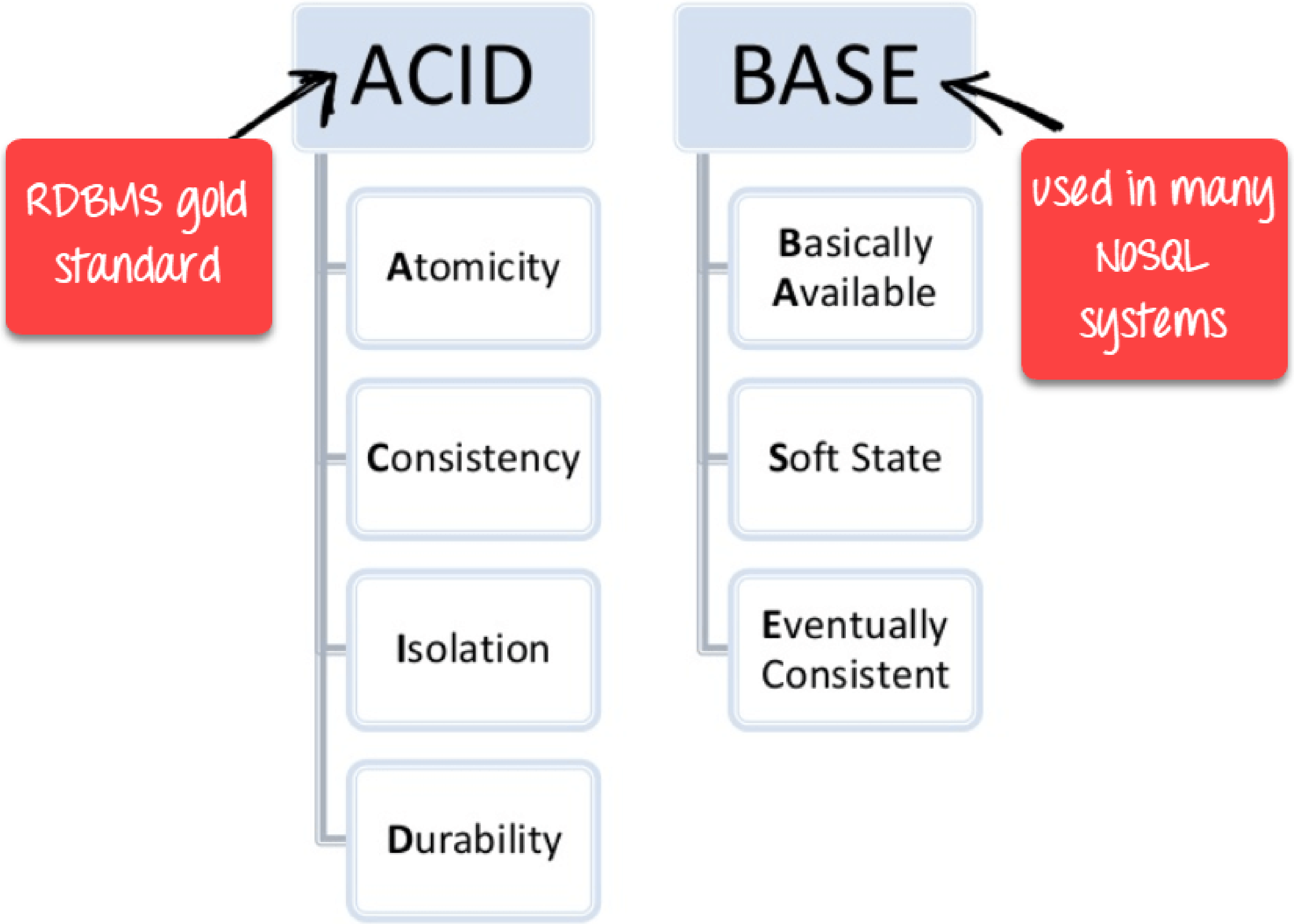
- Primary key: A guaranteed unique value in a particular table that can be used to always reference a record. A key in a key-value store, URI in a document store, or IRI in a triple or graph store.
- Foreign key: A data value that indicates a record is related to a record in a different table or record set. Has the same value as the primary key in the related table.
- Relationship: A link, or edge in graph theory, that indicates two records have a semantic link. The relationship can be between two records in the same or different tables.

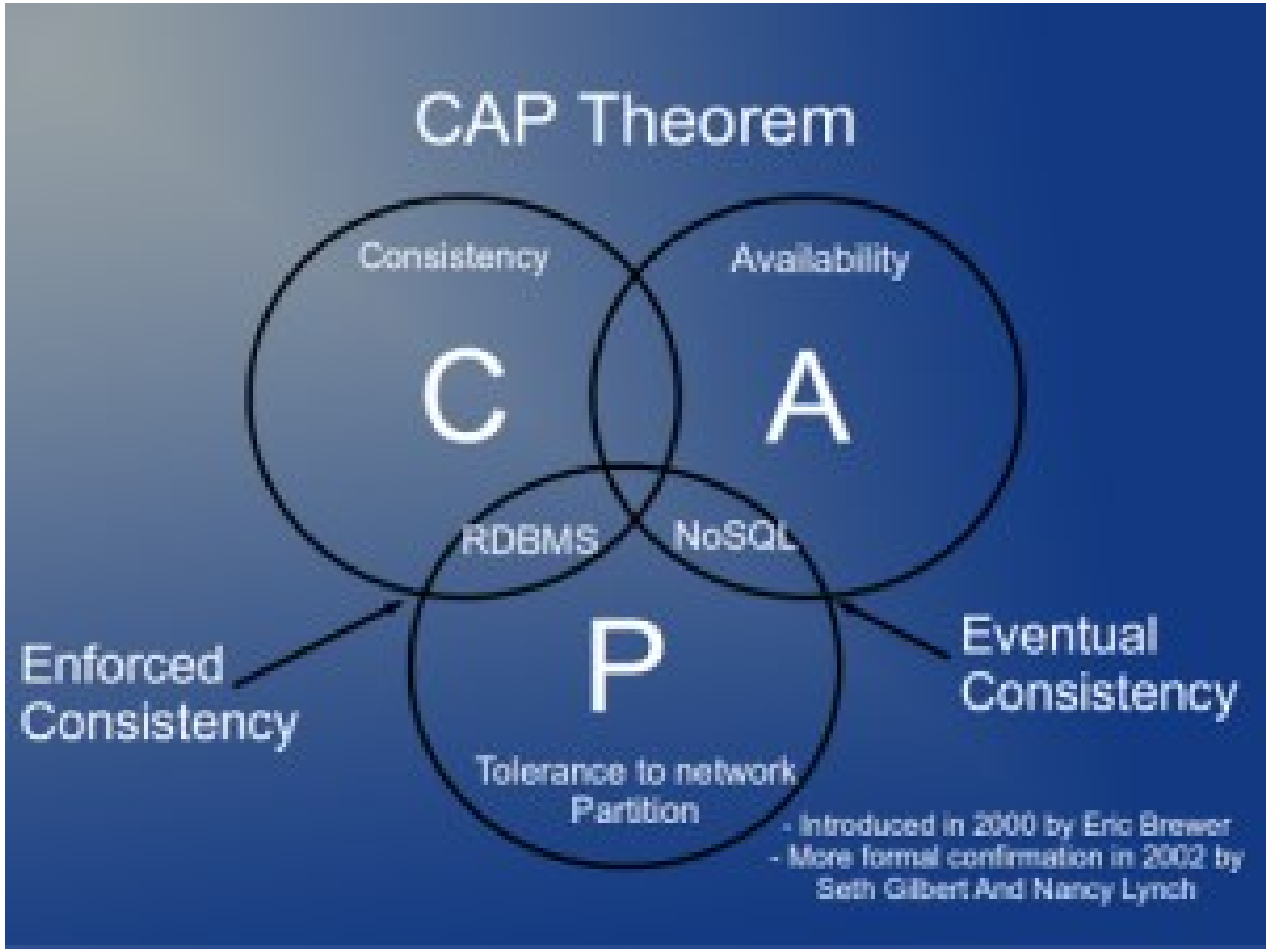
ACID vs BASE

- The consistency property of a database means that once data is written to a database successfully, queries that follow are able to access the data and get a consistent view of the data.
- In the NoSQL world, consistency generally falls into one of two camps:

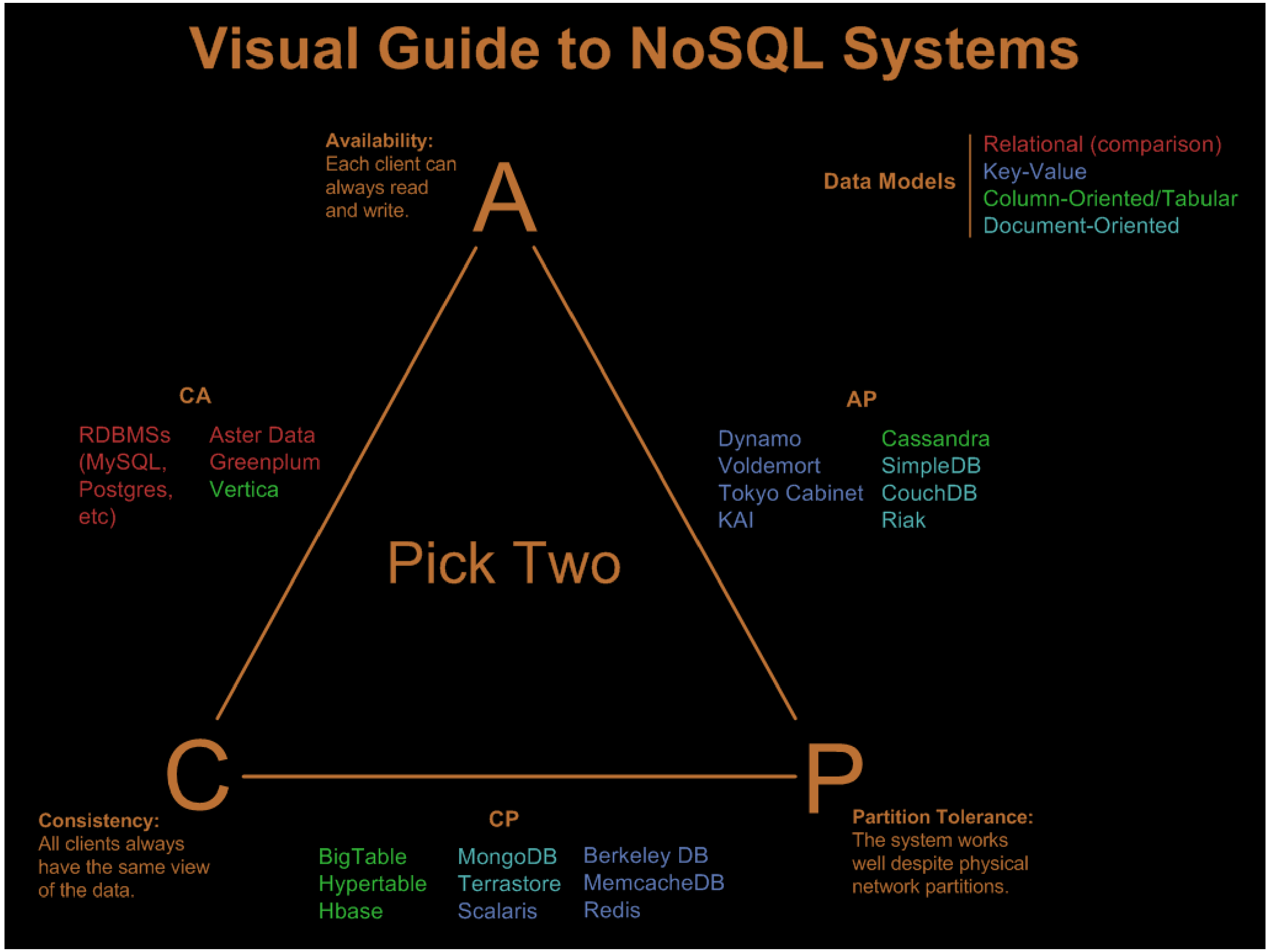
▶ACID Consistency : ACID means that once data is written, you have full consistency in reads.

▶Eventual Consistency (BASE): BASE means that once data is written, it will eventually appear for reading.

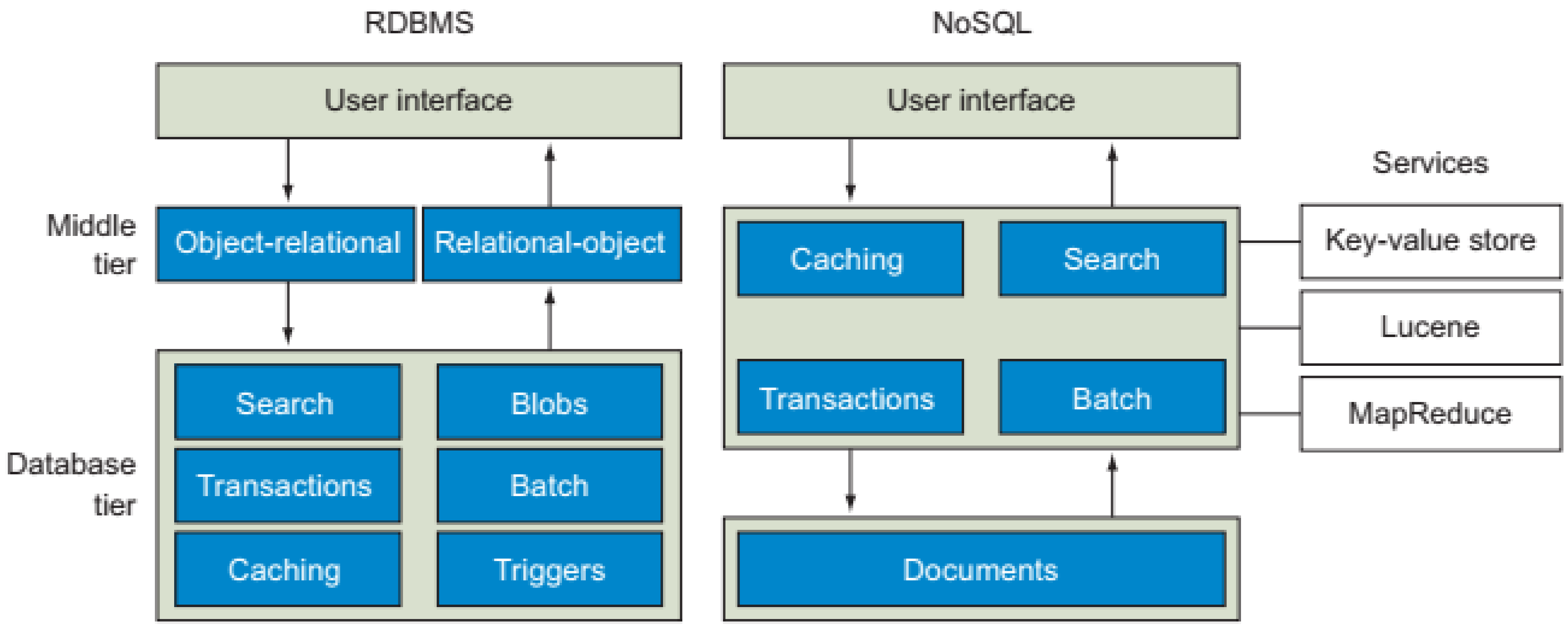

ACID
Atomicity
Consistency
Isolation
Durability
RDBMS gold standard
BASE
Basically Available
Soft State
Eventually Consistent
used in many NoSQL systems

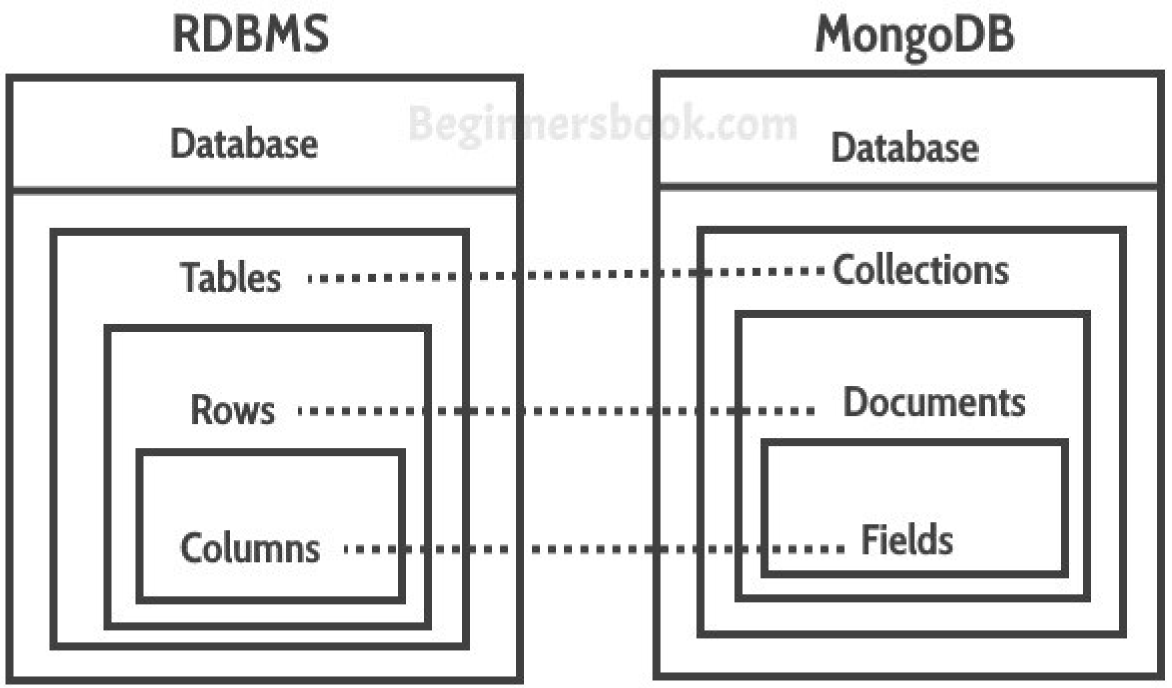

CAP Theorem
Consistency
C
Availability
A
RDBMS
NoSQL
P
Tolerance to network Partition
Enforced Consistency
Eventual Consistency
- Introduced in 2000 by Eric Brewer
- More formal confirmation in 2002 by Seth Gilbert And Nancy Lynch

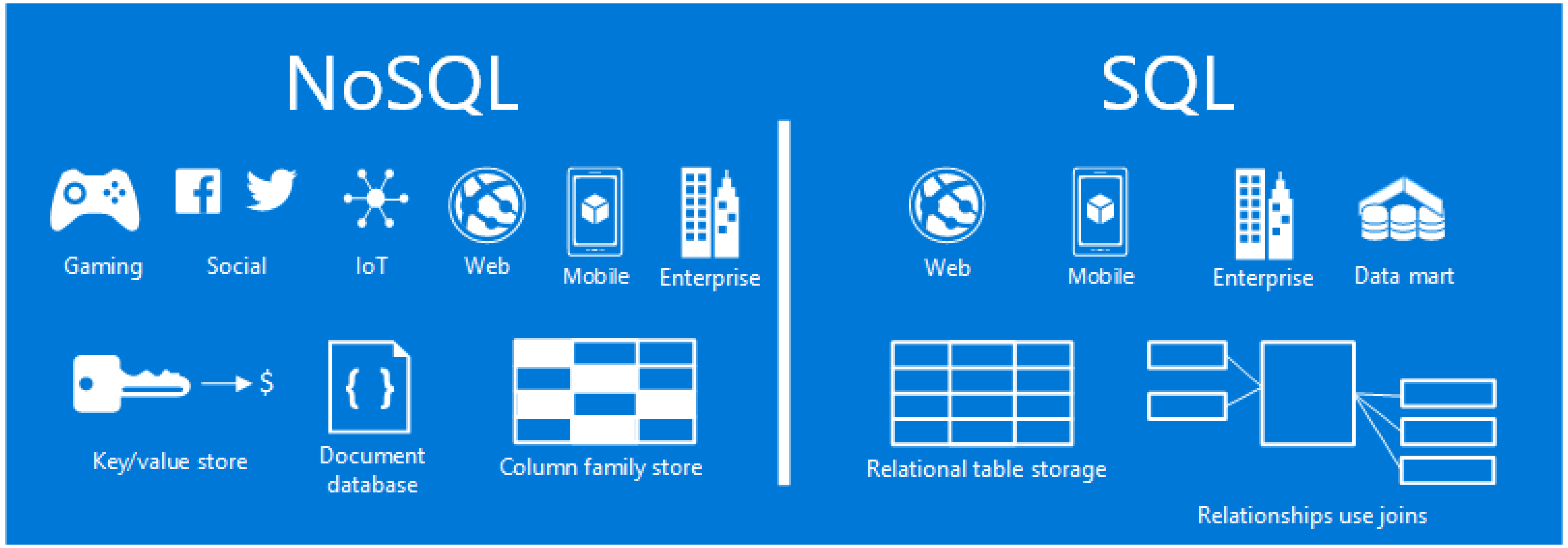
SCHOOL OF COMPUTER SCIENCE AND IT



Comparing NoSQL & RDBMS




RDBMS
MongoDB
Database
Database
Beginnersbook.com
Tables
Collections
Rows
Documents
Columns
Fields


NoSQL
SQL
Gaming
Social
IoT
Web
Mobile
Enterprise
Web
Mobile
Enterprise
Data mart
Key/value store
Document database
Column family store
Relational table storage
Relationships use joins

NoSQL

PROS

1. Loading test data can be done with drag-and-drop tools before ER modeling is complete.
2. Modular architecture allows components to be exchanged.
3. Linear scaling takes place as new processing nodes are added to the cluster.
4. Lower operational costs are obtained by autosharding.
5. Integrated search functions provide high-quality ranked search results.
6. There's no need for an object-relational mapping layer.
7. It's easy to store high-variability data.

CONS

- ACID transactions can be done only within a document at the database level. Other transactions must be done at the application level.
- Document stores don't provide fine-grained security at the element level.
- NoSQL systems are new to many staff members and additional training may be required.
- The document store has its own proprietary nonstandard query language, which prohibits portability.
- The document store won't work with existing reporting and OLAP tools.

SQL

Relational Data Model

Pros
- Easy to use and setup.
- Universal, compatible with many tools.
- Good at high-performance workloads.
- Good at structure data.

Cons
- Time consuming to understand and design the structure of the database.
- Can be difficult to scale.

No SQL

Document Data Model

Pros
- No investment to design model.
- Rapid development cycles.
- In general faster than SQL.
- Runs well on the cloud.

Cons
- Unsuited for interconnected data.
- Technology still maturing.
- Can have slower response time.