CARLSON SCHOOL OF MANAGEMENT

UNIVERSITY OF MINNESOTA

Introduction to NoSQL Data Stores

MSBA 6330 Prof Liu

Learning Objectives

- Understand strengths and limitations of traditional SQL databases
- Understand common characteristics of NoSQL databases
- Understand the trade-offs facing distributed databases in light of the CAP theorem
- Distinguish between ACID and BASE consistency models
- Be familiar with different types of NoSQL / NewSQL databases and their use cases
- Understand the complexity of choosing different database technologies for big data applications

Topics

- Limitations of RDBMS for the Web
- What is NoSQL?
- NoSQL and the CAP theorem
- Different kinds of NoSQL data stores
- NewSQL
- How to choose?

Introduction to NoSQL

LIMITATIONS OF RDBMS FOR THE WEB

RDBMS has been a dominant data storage model for a few decades

- Relational databases have been a mainstay of how data is stored since 1970s (and still are)
  - Shines with complicated queries
  - Declarative queries (SQL) rather than programing
  - Fast query & analysis, but not necessarily on large datasets
- Strong support for transactions – ACID Properties
  - **Atomic** – All of the work in a transaction completes (commit) or none of it
  - **Consistent** – A transaction transforms the database from one consistent state to another consistent state. Consistency is defined in terms of constraints.
  - **Isolated** – The results of any changes made during a transaction are not visible until the transaction has committed.
  - **Durable** – The results of a committed transaction survive failures

But there are limits in scaling RDBMSs

- Best way to provide ACID and a rich query model is to have the dataset on a single machine.
- However, there are limits to scaling up (vertical scaling) as the dataset gets big
- Past a certain point, an organization will find it is cheaper and more feasible to scale out (horizontal scaling) than scaling up
- DBAs began to look at ways of scaling out
  - *master-slave*: all writes go to the master. all reads can perform against a fleet of replicated slave databases (inconsistency; do not scale well)
  - *sharding*: partition database based on value range, hash-key, or dictionaries (unable to join across partitions; no referential integrity)

http://adam.herokuapp.com/past/2009/7/6/sql_databases_dont_scale/

Inadequacy of RDMS for the web

- Hooking your RDBMS to a web-based application was a recipe for headaches
  - "Slashdot effect": when a popular website links to a smaller website, causing a massive spike in traffic
- Sharding is like putting a bandage on a broken leg
  - It does not scale to handle hundreds of thousands of visitors in a short-time span
  - People with the largest datasets (terabyte/petabyte), e.g. Google, Facebook, Twitter, began to explore alternative ways to store data, especially around 2008/2009.

Introduction to NoSQL

WHAT IS NOSQL?

Drivers of the NoSQL

- Two papers were the seeds of the NoSQL movement
  - BigTable (Google, 2006):
    - Wide-column data model
    - Consistent, fault-tolerant, scale to large clusters (1000+ nodes)
  - Dynamo (Amazon, 2007):
    - Distributed key-value data store
    - Eventual consistency, always writable, fault-tolerant
- In addition,
  - The advent of Amazon S3 signals to others it was okay to look at alternative storage solutions
  - Many NoSQL solutions are open source software

What is NoSQL?

- Stands for **N**ot **O**nly **SQL** (coined by Eric Evans)
  - *"to solve a problem that relational databases are a bad fit for."*
- (No standard definition) nosql-database.org:
  - Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontal scalable.

NoSQL-Databases.org:

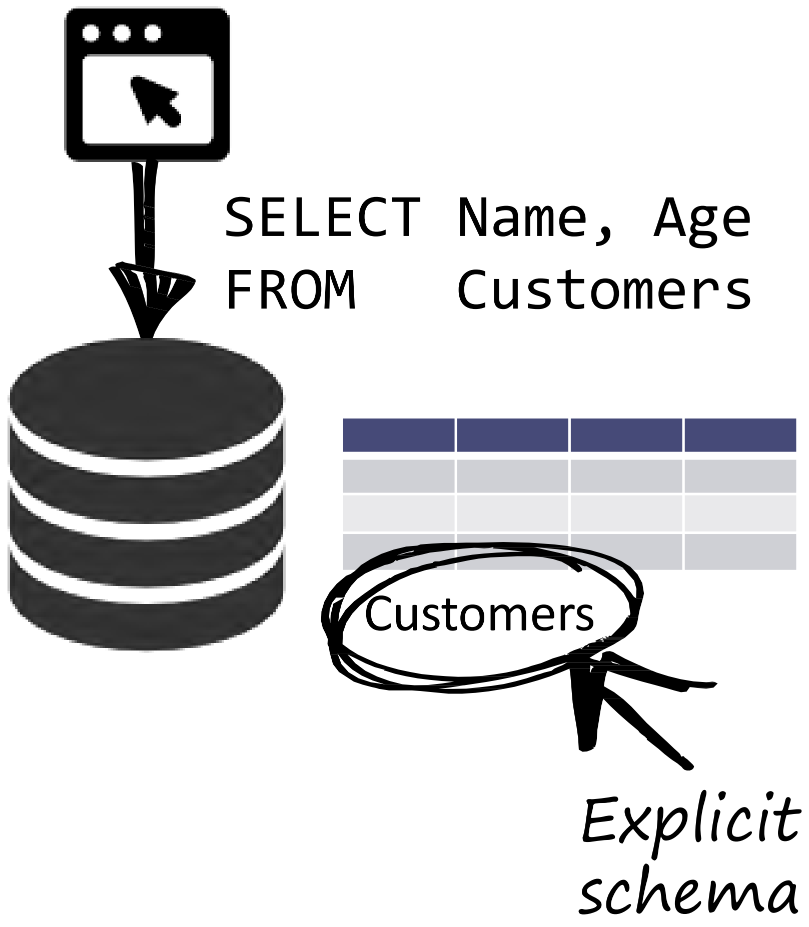
Current list has over 150 NoSQL systems

Common NoSQL Characteristics

- don't use the relational data model
- optimized for running on a cluster
- open source
- schema-less (schema-free)
- tunable consistency

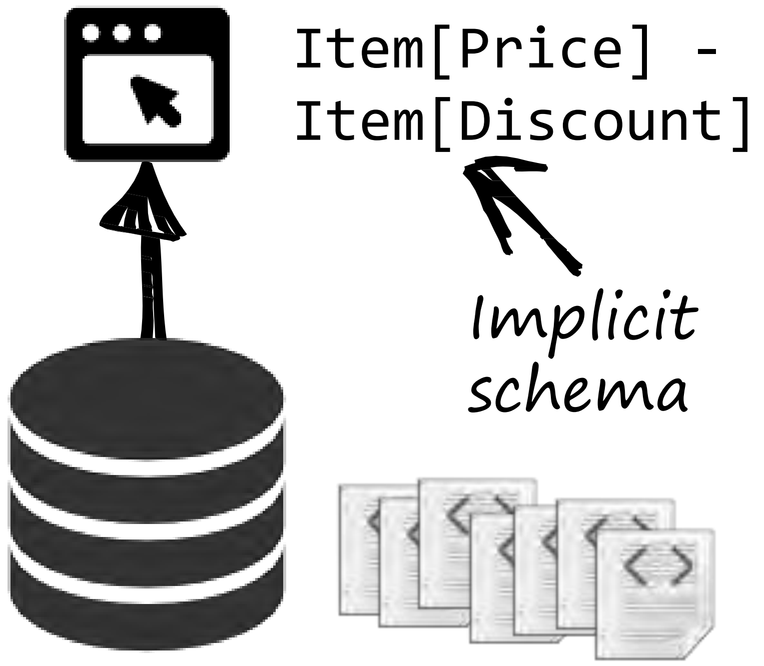
Schema-free Data Modeling

RDBMS:



```
SELECT  Name, Age
FROM    Customers
```

NoSQL DB:



```
Item[Price] -
Item[Discount]
```

NoSQL Paradigm Shift: Open Source & Commodity Hardware

Commercial DBMS		Open-Source DBMS
Specialized DB hardware (Oracle Exadata, etc.)	➡	Commodity hardware
Highly available network (Infiniband, Fabric Path, etc.)	➡	Commodity network (Ethernet, etc.)
Highly Available Storage (SAN, RAID, etc.)	➡	Commodity drives (standard HDDs, JBOD)

NoSQL Pros and Cons

- Pros:
  - Massive scalability
  - Flexible schema
  - Quicker/cheaper to set up
  - Relaxed consistency (NO ACID) $\rightarrow$ higher performance & availability
- Cons:
  - No declarative query language (SQL) $\rightarrow$ more programming
    - May not integrate well with other applications that support SQL
    - May lose powerful SQL queries e.g., joins, group by, order by
  - Relaxed consistency $\rightarrow$ fewer guarantees (NO ACID)

Challenge: Coordination

- The solution to availability and scalability is to decentralize and replicate functions and data…but how do we coordinate the nodes?
  - data consistency
  - update propagation
  - mutual exclusion
  - consistent global states
  - group membership
  - group communication
  - event ordering
  - distributed consensus
  - quorum consensus

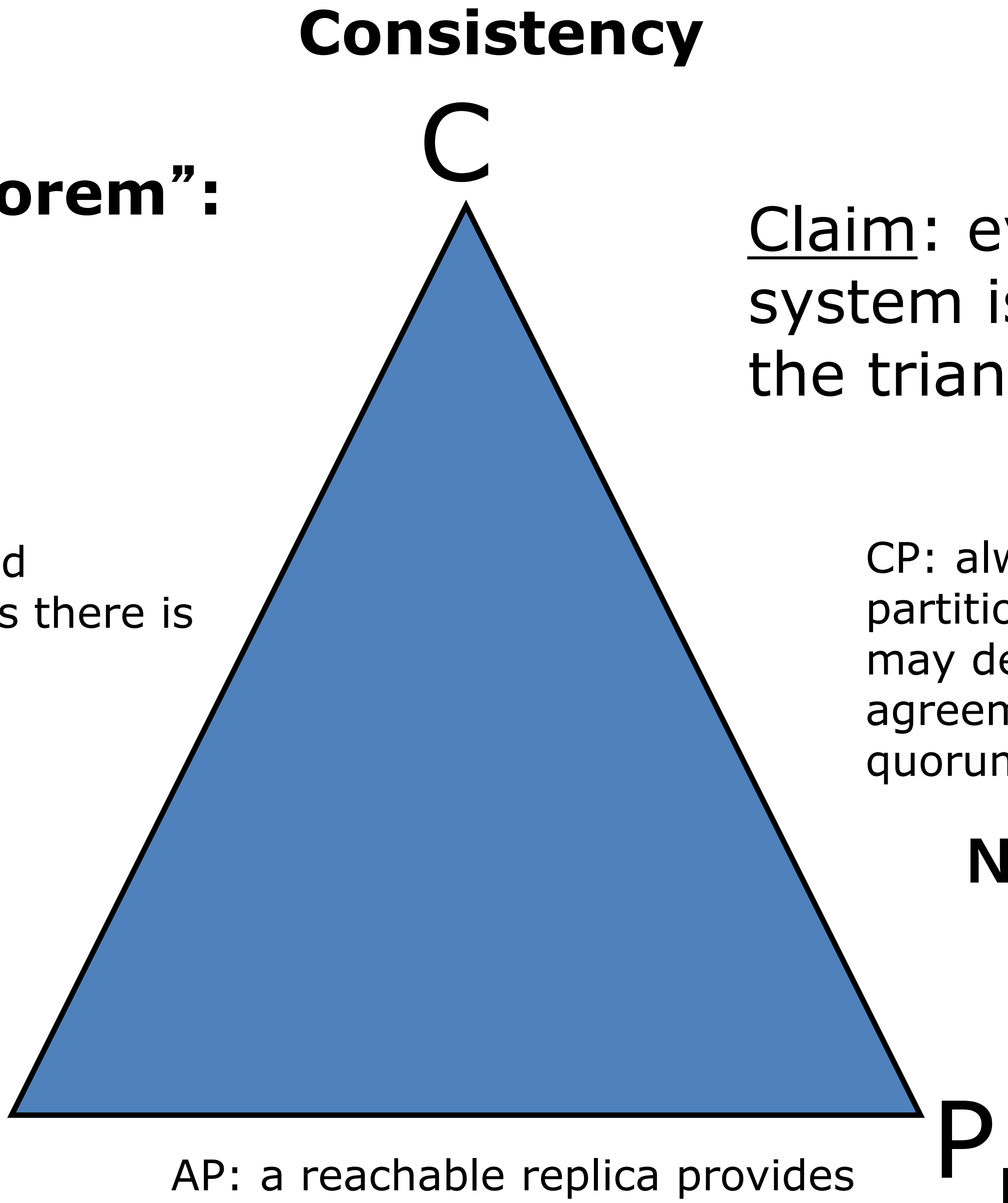
Introduction to NoSQL

NOSQL AND THE CAP THEOREM

CAP Theorem

- Proposed as a conjecture by Eric Brewer (Berkeley ,1999)
  - Subsequently proved by Gilbert and Lynch (MIT) in 2002 as a theorem.
- In a distributed data store you can satisfy **at most 2 out of the 3 guarantees**:
  - **Consistency**: Every read receives the most recent write or an error
  - **Availability**: Every request receives a (non-error) response, without the guarantee that it contains the most recent write
  - **Partition-tolerance**: the system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes
- Proof: When a network partition failure happens, should we
  - Cancel the operation and thus decrease the availability but ensure consistency?
  - Proceed with the operation and thus provide availability but risk inconsistency?

Fox&Brewer "CAP Theorem": C-A-P: choose two.



Claim: every distributed system is on one side of the triangle.

CA: available, and consistent, unless there is a partition.

Do not scale well!

CP: always consistent, even in a partition, but a reachable replica may deny service without agreement of the others (e.g., quorum).

Not always available

Availability

Partition-resilience

AP: a reachable replica provides service even in a partition, but may be inconsistent if there is a failure.

Not always consistent

Availability

- Traditionally, thought of as the server/process available five 9’s (99.999 %).
- However, for large node system, at almost any point in time there’s a good chance that a node is either down or there is a network disruption among the nodes.
  - Want a system that is resilient in the face of network disruption

Consistency Model

- A consistency model determines rules for visibility and apparent order of updates.
- For example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses.
  - Client B reads row X from node M
  - Does client B see the write from client A?
- Consistency is a continuum with tradeoffs
  - CAP Theorem states: Strict Consistency can't be achieved at the same time as availability and partition-tolerance.

Eventual Consistency

- When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
- For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service
- Known as **BASE (Basically Available, Soft state, Eventual consistency)**, as opposed to ACID
  - *Basically Available*: possibilities of faults but not a fault of the whole system
  - *Soft state*: copies of a data item may be inconsistent
  - *Eventually Consistent*: copies becomes consistent at some later time if there are no more updates to that data item

Implications of BASE consistency model

- Characteristics
  - Weak consistency – stale data OK
  - Availability first
  - Best effort
  - Approximate answers OK
  - Aggressive (optimistic)
  - Simpler and faster

Introduction to NoSQL

DIFFERENT KINDS OF NOSQL DATA STORES

NoSQL System Classification

- two common criteria

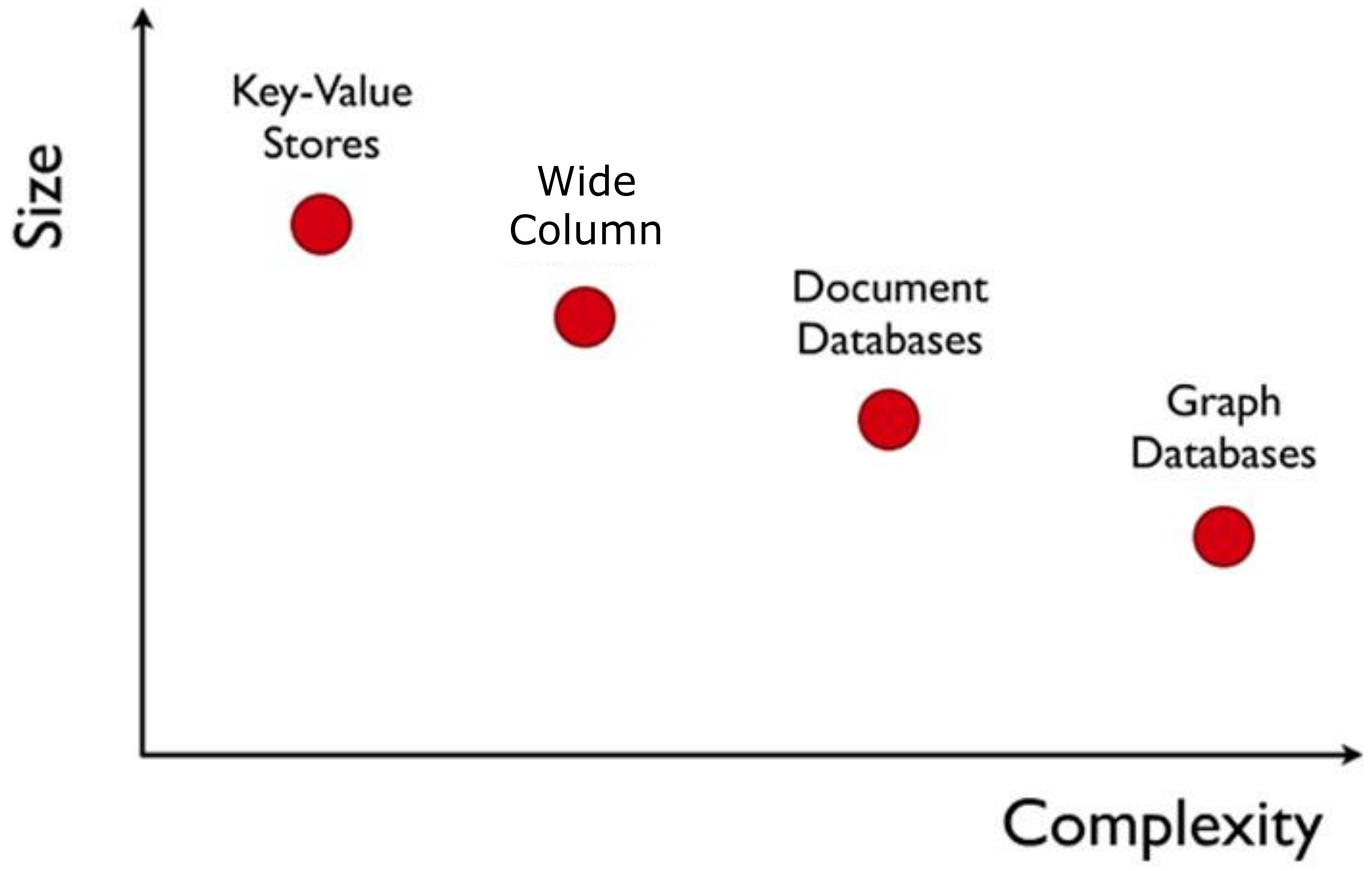
Data Model

- Key-Value
- Wide-Column
- Document
- Graph

Consistency/Availability Trade-Off

- **AP**: Available & Partition Tolerant
- **CP**: Consistent & Partition Tolerant
- **CA**: Not Partition Tolerant

NoSQL Data Models: Size and Complexity

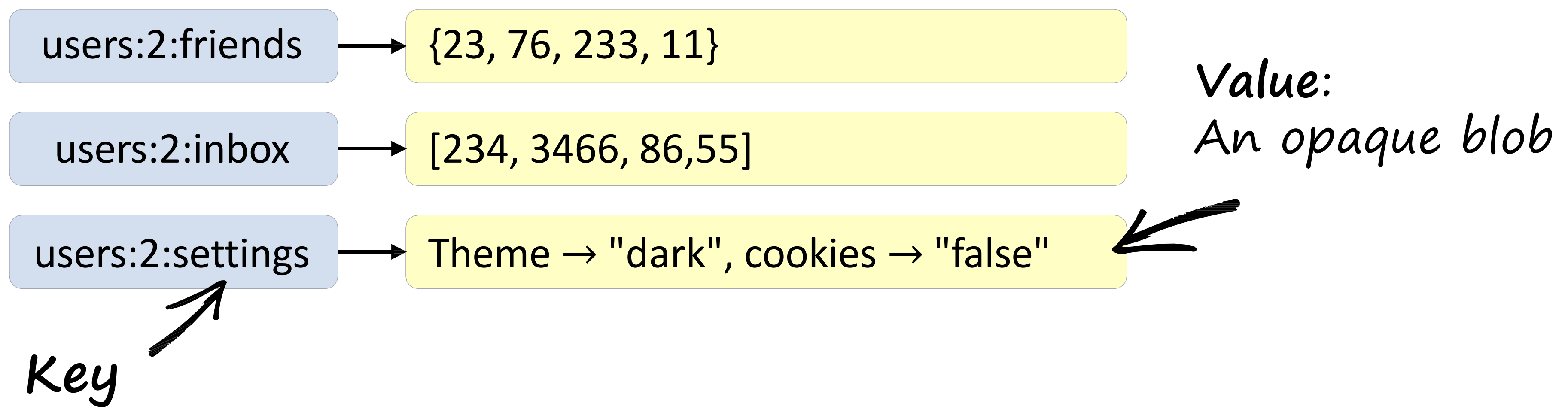


Introduction to NoSQL

KEY-VALUE STORES

Key-Value Stores

- Data model: (key) -> value
- Interface: CRUD (Create, Read, Update, Delete)



- Examples: Amazon Dynamo (AP), Riak (AP), Redis (CP)

$\underbrace{\text{Amazon Dynamo (AP), Riak (AP)}}_{\text{High availability but not always consistent}}$, $\underbrace{\text{Redis (CP)}}_{\text{Strict consistency but not always available during partitions}}$

High availability but not always consistent

Strict consistency
but not always available
during partitions

Key-Value Store Pros and Cons

- Pros:
  - very fast
  - very scalable
  - simple model
  - able to distribute horizontally
- Cons:
  - many data structures (objects) can't be easily modeled as key value pairs
  - Only primary index: lookup by key

Key-value Store Use Cases

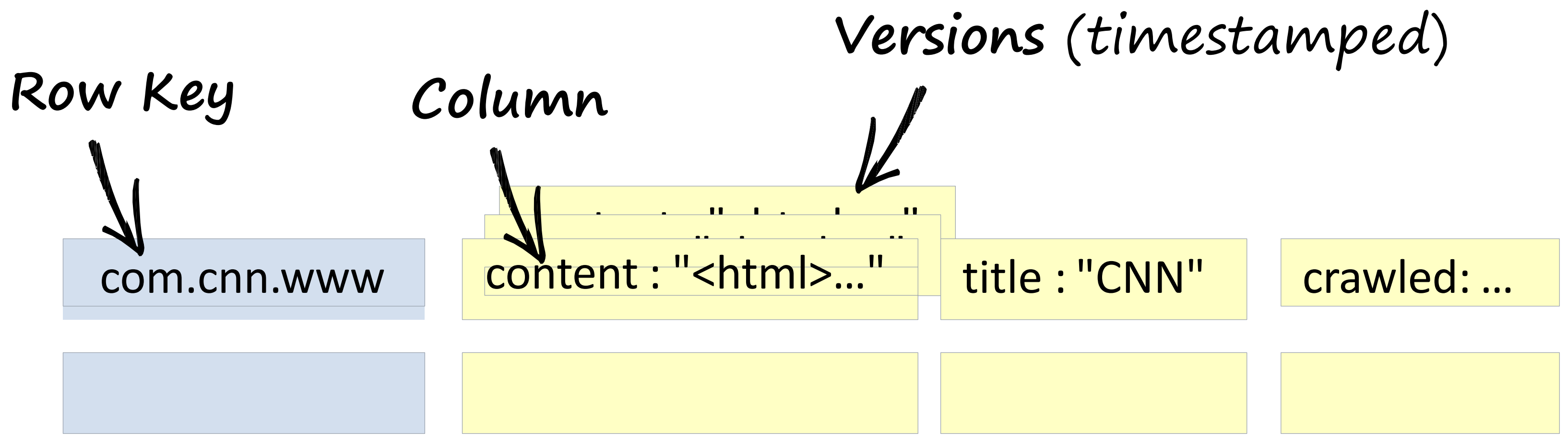
- Key-value data stores are ideal for storing user profiles, blog comments, product recommendations (user preferences), session information, shopping cart data.
  - Twitter uses **Redis** to deliver your Twitter timeline
  - Pinterest uses **Redis** to store lists of users, followers, unfollowers, boards, and more
  - Quora uses **Memcached** to cache results from slower, persistent databases
- avoid if we need to
  - query the database by specific data value
  - need relationships between data values
  - need to operate on multiple unique keys

Introduction to NoSQL

WIDE-COLUMN STORES

Wide-Column Stores

- Data model: (rowkey, column, timestamp) -> value
  - can be interpreted as two/three dimensional key-value store
  - You can group related columns into **column families**.
- Interface: CRUD, Scan



- Examples: Cassandra (AP), Google BigTable (CP), HBase (CP)

High availability but not always consistent

Strict consistency but not always available during partitions

Wide-column Store Advantages & Use Cases

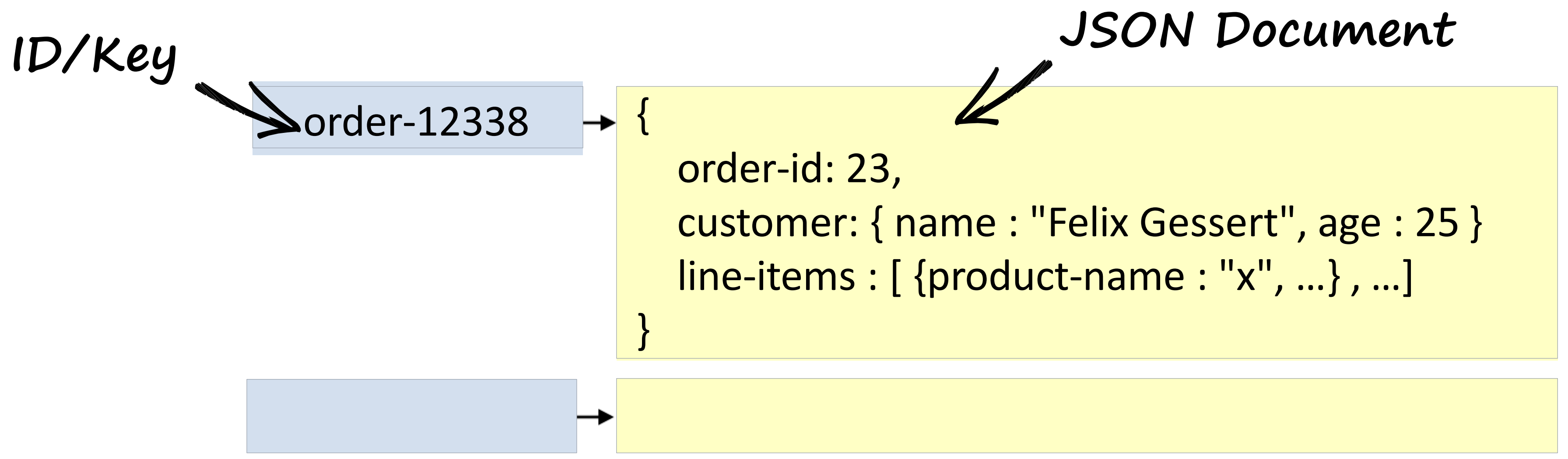
- Column stores offer very high performance and a highly scalable architecture. They are fast to load and query
  - Ideal for real-time data logging, analysis, and random read/write access of huge amounts of data.
  - But does not support transactions
- Popular among companies and organizations dealing with big data, IoT, and user recommendation and personalization engines.
  - Spotify uses **Cassandra** to store user profile attributes and metadata about artists, songs, etc. for better personalization
  - Facebook builts its revamped Messages on top of **HBase**, but is now also used for Nearby Friends feature and search
  - Outbrain uses **Cassandra** to serve over 190 billion personalized content recommendations each month

Introduction to NoSQL

DOCUMENT STORES

Document Stores

- Data model: (collection, key) -> document (JSON, XML, BSON)
  - can index common columns for faster performance
- Interface: CRUD, Queries, Map-Reduce



```
{
   order-id: 23,
   customer: { name : "Felix Gessert", age : 25 }
   line-items : [ {product-name : "x", ...} , ...]
}
```

- Examples: CouchDB (AP), RethinkDB (CP), MongoDB (CP)

Document Databases Advantages & Use Cases

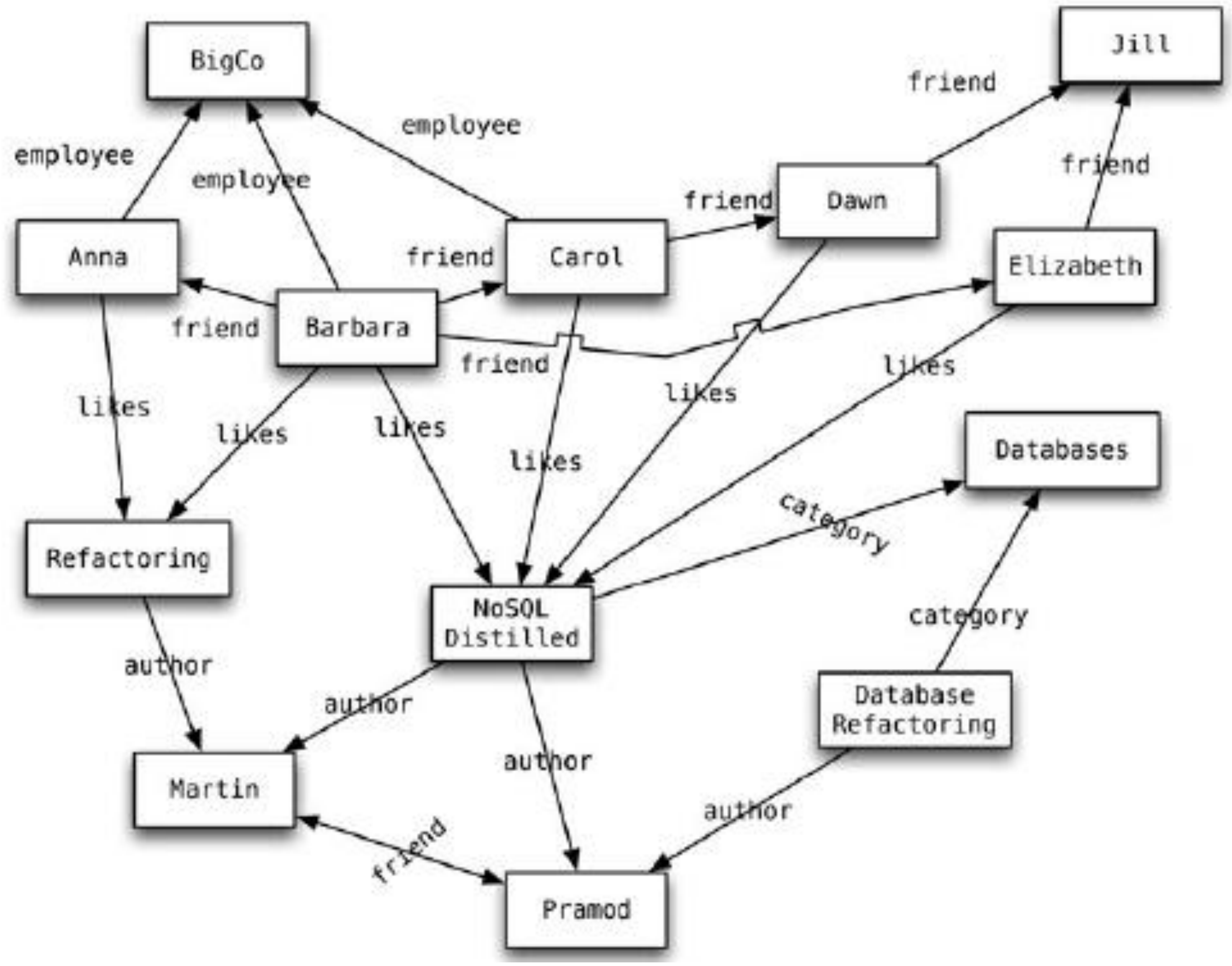
- Can store data with flexible and evolving schema, scale out well, and perform fast ad hoc queries using secondary indexes.
- Good for content management systems, blogging platforms, analytics platforms, e-commerce platforms
  - SEGA uses cloud hosted MongoDB for handling 11 million in-game accounts
  - Aer Lingus uses MongoDB with Studio 3T to handle ticketing and internal apps
  - Built on MongoDB, The Weather Channel's iOS and Android apps deliver weather alerts to 40 million users in real-time
- avoid if one must run complex search queries or require complex transactions

Introduction to NoSQL

GRAPH DATABASES

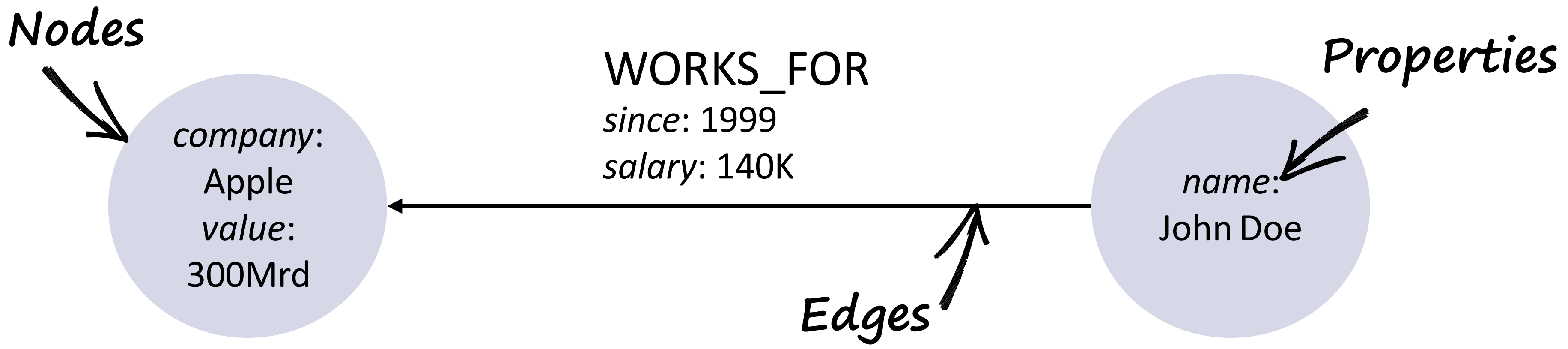
Graph Databases

- Some data are more naturally represented as a graph



Graph Databases

- Data model: G = (V, E): Graph-Property Model
- Interface: Traversal algorithms, queries, transactions



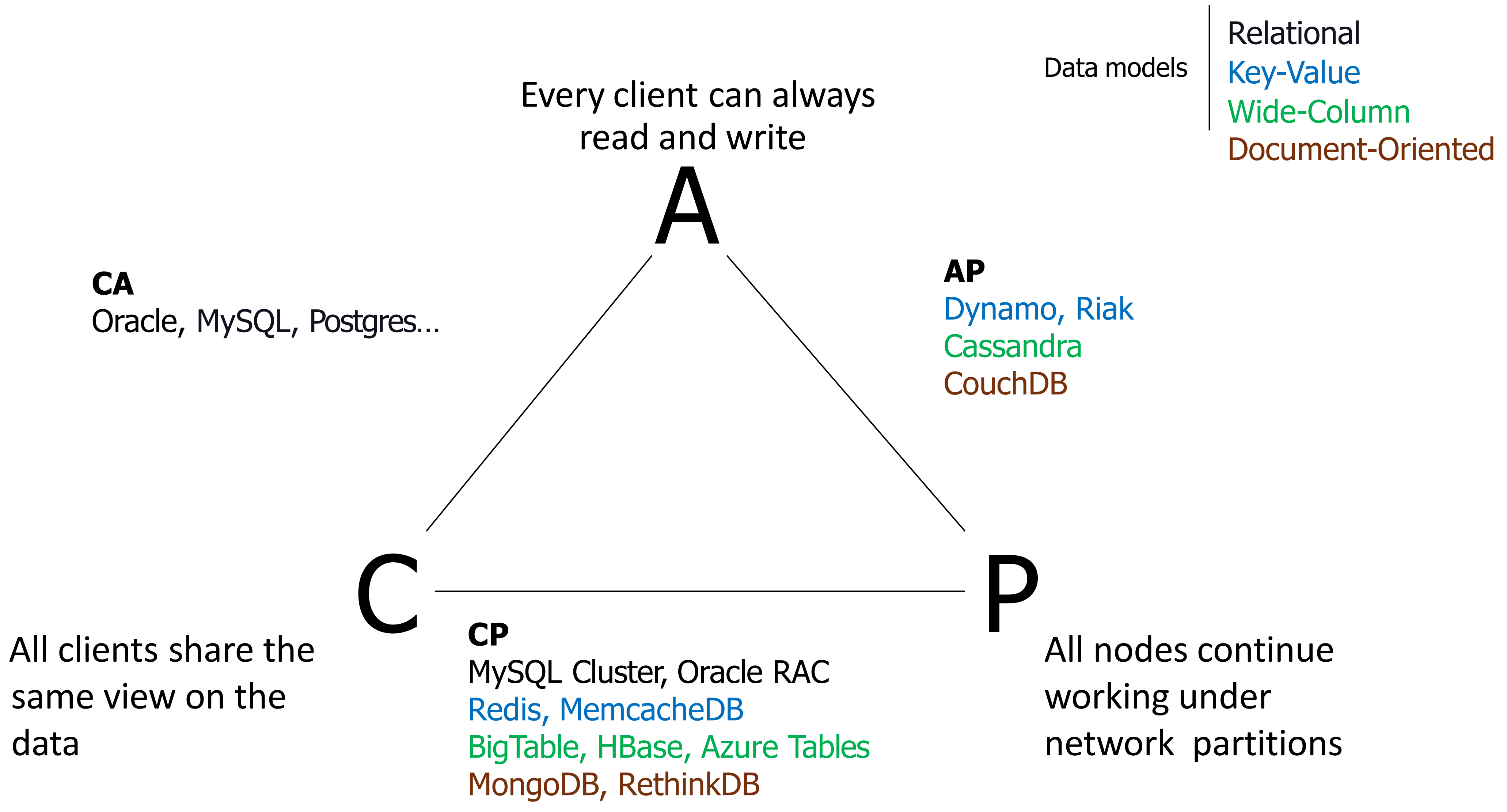
usually unscalable (optimal partitioning is NP-complete)

- Examples: Neo4j (CA), InfiniteGraph (CA), OrientDB (CA)

Graph Database Advantages and Use Cases

- Graph databases are great for establishing data relationships especially when dealing with large data sets.
  - Fault detection, real-time recommendation engines, master data management, network and IT operations, identity and access management.
- They offer blazing fast query performance and flexibility that relational databases cannot compete with, especially as data grows much deeper.
  - Walmart uses Neo4j to provide customers personalized, relevant product recommendations and promotions
  - Medium uses Neo4j to build their social graph to enhance content personalization
  - Cisco uses Neo4j to mine customer support cases to anticipate bugs

NoSQL Triangle


Data models
Relational
Key-Value
Wide-Column
Document-Oriented
Every client can always read and write
A
CA
Oracle, MySQL, Postgres…
AP
Dynamo, Riak
Cassandra
CouchDB
C
P
All clients share the same view on the data
CP
MySQL Cluster, Oracle RAC
Redis, MemcacheDB
BigTable, HBase, Azure Tables
MongoDB, RethinkDB
All nodes continue working under network partitions

Nathan Hurst: Visual Guide to NoSQL Systems
http://blog.nahurst.com/visual-guide-to-nosql-systems

Introduction to NoSQL

NEWSQL

NewSQL

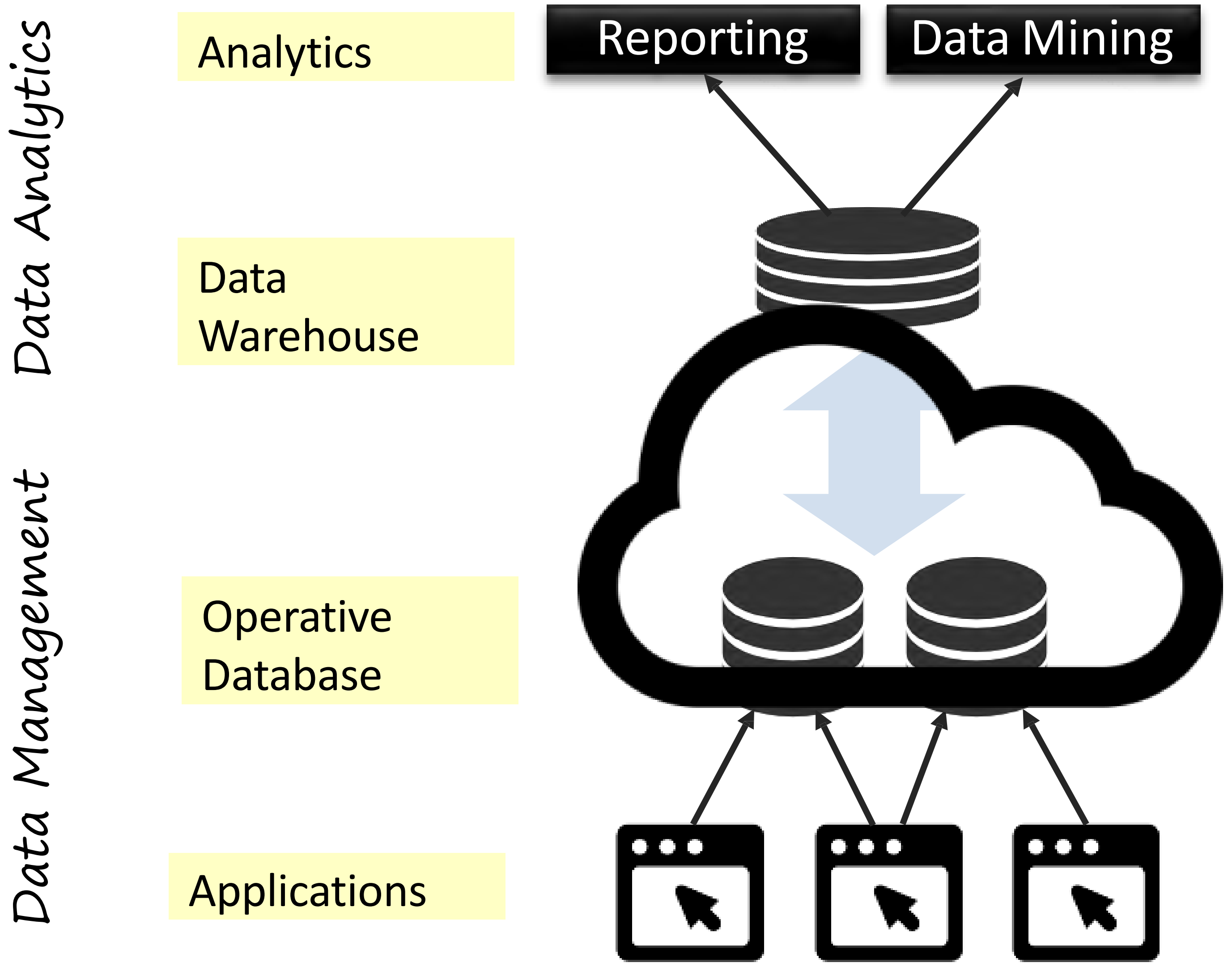
- NewSQL refers to a new class of databases providing transactional consistency and massive scalability
  - Transactional support of SQL + NoSQL massive scalability
  - Some guarantee stronger consistency, others (MemSQL) provide tunable consistency
  - NewSQL favors consistency over availability (CP)
  - They are cloud-first databases
    - lock-free concurrency control + share nothing architecture
- Examples
  - SAP HANA, NuoDB, MemSQL, VoltDB, Google Spanner, CockRoachDB

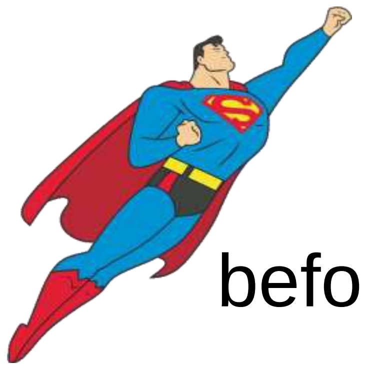
Introduction to NoSQL

HOW TO CHOOSE?

The era of one-size-fits-all database systems is over

- Typical Data Architecture:





before: RDBMS does it all

now: RDBMS + NoSQL

RDBMS:

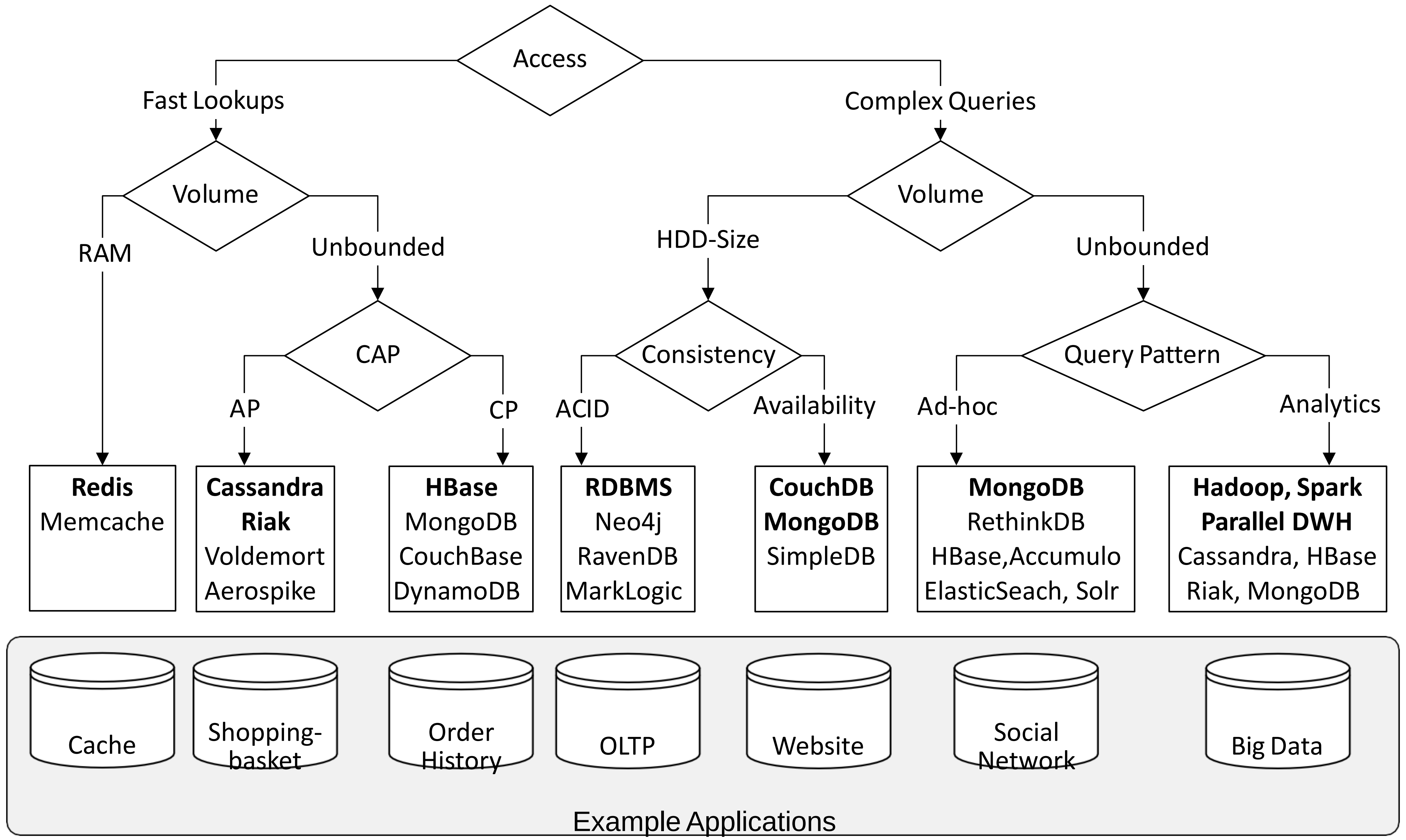
OLTP
OLAP

Cassandra
HBASE
mongoDB
CouchDB relax
riak
redis

Which database to choose? Why?

- Large part of the choice depends on the business use-case and tradeoffs between scale, consistency, and availability
  - The CAP Theorem
- It also depends on the type of queries you plan to perform on the data?
  - Are they simple look ups or more complex queries?
  - Different SQL/NoSQL has different
- One need also take into account what is the cost of not having ACID?

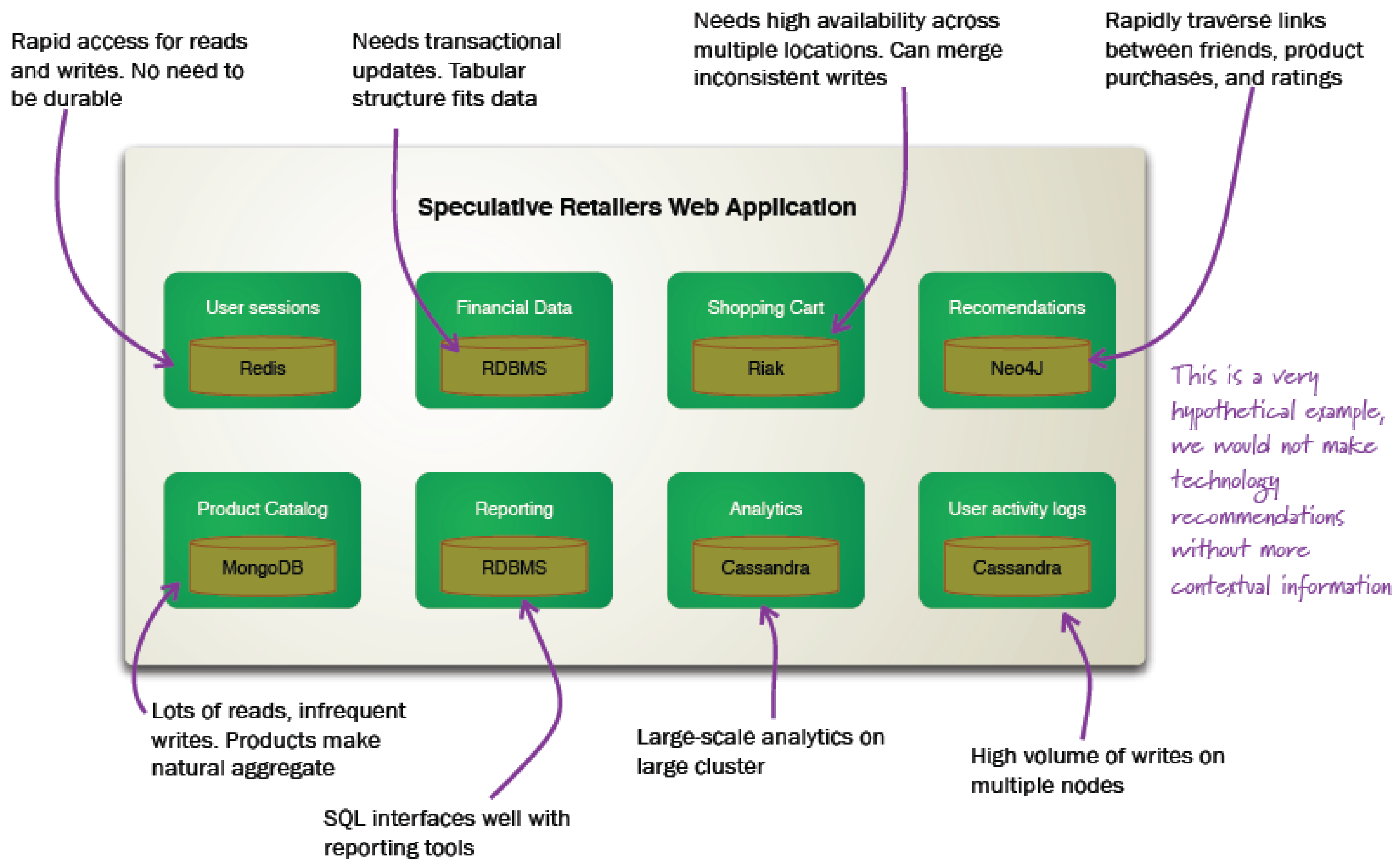
NoSQL Decision Tree


Access
Fast Lookups
Complex Queries
Volume
RAM
Unbounded
CAP
AP
CP
Volume
HDD-Size
Unbounded
Consistency
ACID
Availability
Query Pattern
Ad-hoc
Analytics
Redis Memcache
Cassandra Riak Voldemort Aerospike
HBase MongoDB CouchBase DynamoDB
RDBMS Neo4j RavenDB MarkLogic
CouchDB MongoDB SimpleDB
MongoDB RethinkDB HBase,Accumulo ElasticSeach, Solr
Hadoop, Spark Parallel DWH Cassandra, HBase Riak, MongoDB
Cache
Shopping-basket
Order History
OLTP
Website
Social Network
Big Data
Example Applications

source: An In-Depth Tutorial on NoSQL Data Stores, by Gesset, Wingerath, and Ritter, 2017

Using multiple data storage technologies

- Using multiple data storage technologies, chosen based upon the way data is being used by individual applications.


Rapid access for reads and writes. No need to be durable
Needs transactional updates. Tabular structure fits data
Needs high availability across multiple locations. Can merge inconsistent writes
Rapidly traverse links between friends, product purchases, and ratings
Speculative Retailers Web Application
User sessions
Redis
Financial Data
RDBMS
Shopping Cart
Riak
Recomendations
Neo4J
Product Catalog
MongoDB
Reporting
RDBMS
Analytics
Cassandra
User activity logs
Cassandra
This is a very hypothetical example, we would not make technology recommendations without more contextual information
Lots of reads, infrequent writes. Products make natural aggregate
SQL interfaces well with reporting tools
Large-scale analytics on large cluster
High volume of writes on multiple nodes

NoSQL - Summary

- NoSQL was initially motivated by inability of traditional RDBMS in dealing with web scale data request
- NoSQL databases reject:
  - Overhead of ACID transactions
  - “Complexity” of SQL
  - Burden of up-front schema design
  - Declarative query expression
- But CAP theorem dictates that there is no one size-fit all solution
  - NewSQL embraces ACID and scalability at the cost of availability
- Choices of form of databases depend on business use-case and nature of data