NoSQL
(Very brief introduction)

Learning Outcomes

- Understand the motivation for NoSQL
- Understand and be able to explain the concepts of NoSQL
- Understand and be able to explain the application areas of NoSQL

Big data

- In the past decade, the amount of data being created has skyrocketed.
- The rate of data creation is only accelerating.
- The data we deal with is diverse.

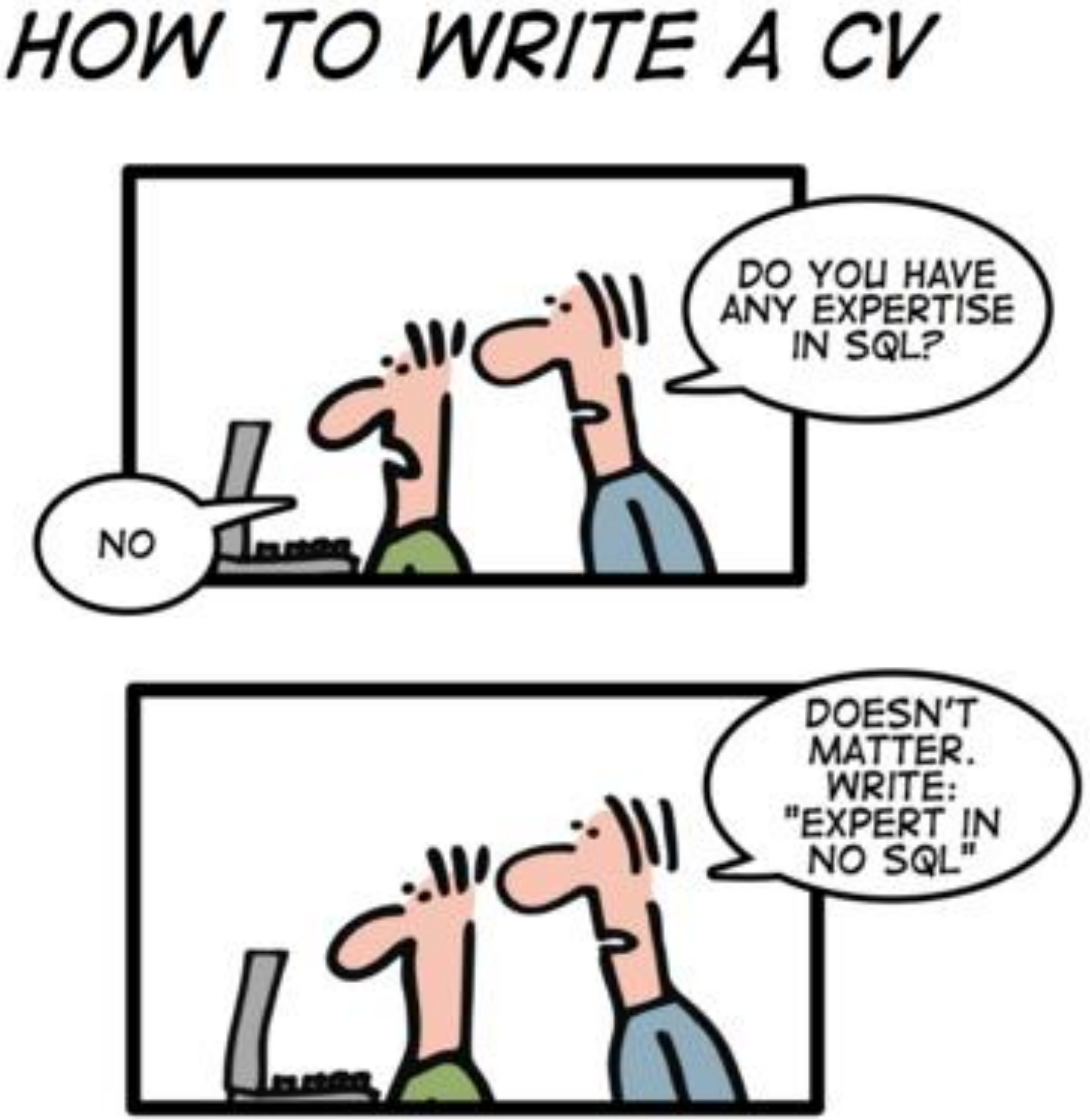
- Traditional database systems, such as relational databases, have been pushed to the limit, and failed to scale to “***Big Data***”.
- To tackle the challenges of Big Data, a new breed of technologies has emerged. Many of these new technologies have been grouped under the term ***NoSQL***.

The NoSQL Movement

- An emerging “movement” around non-relational software for Big Data
- Origin:
  - Google (BigTable, MapReduce framework)
  - Amazon (distributed key/value store called Dynamo).
- Vibrant Open Source community
  - followed with Hadoop, HBase, MongoDB, Cassandra, RabbitMQ, and countless other projects.
- Currently defined as “**<u>what it’s not</u>**”

NoSQL – 1/2

- Not Only SQL or "Not Relational".
- Key features:
  - Flexible schema
  - Quicker/cheaper to set up
  - Massive scalability
  - Relaxed consistency --> higher performance & availability
- Disadvantages:
  - No declarative query language --> more programming
  - Relaxed consistency --> fewer guarantees



NoSQL – 2/2

- NoSQL systems frequently do not provide ACID properties
- Updates are eventually propagated, but there are limited guarantees on the consistency of reads.
- “BASE” instead of “ACID”:
  - BASE = Basically Available, Soft state, Eventually consistent
  - ACID = Atomicity, Consistency, Isolation, Durability

- ***B**asically **A**vailable*:
  - Any request => An answer
  - Even in a changing state
- ***S**oft State*:
  - Opposite to Durability.
  - System’s state (servers or data) could change over time (without any update)
- ***E**ventually consistent*:
  - With time, data can be consistent
  - Updates have to be propagated

- By giving up ACID constraints, one can achieve much higher performance and scalability.

Why NoSQL?

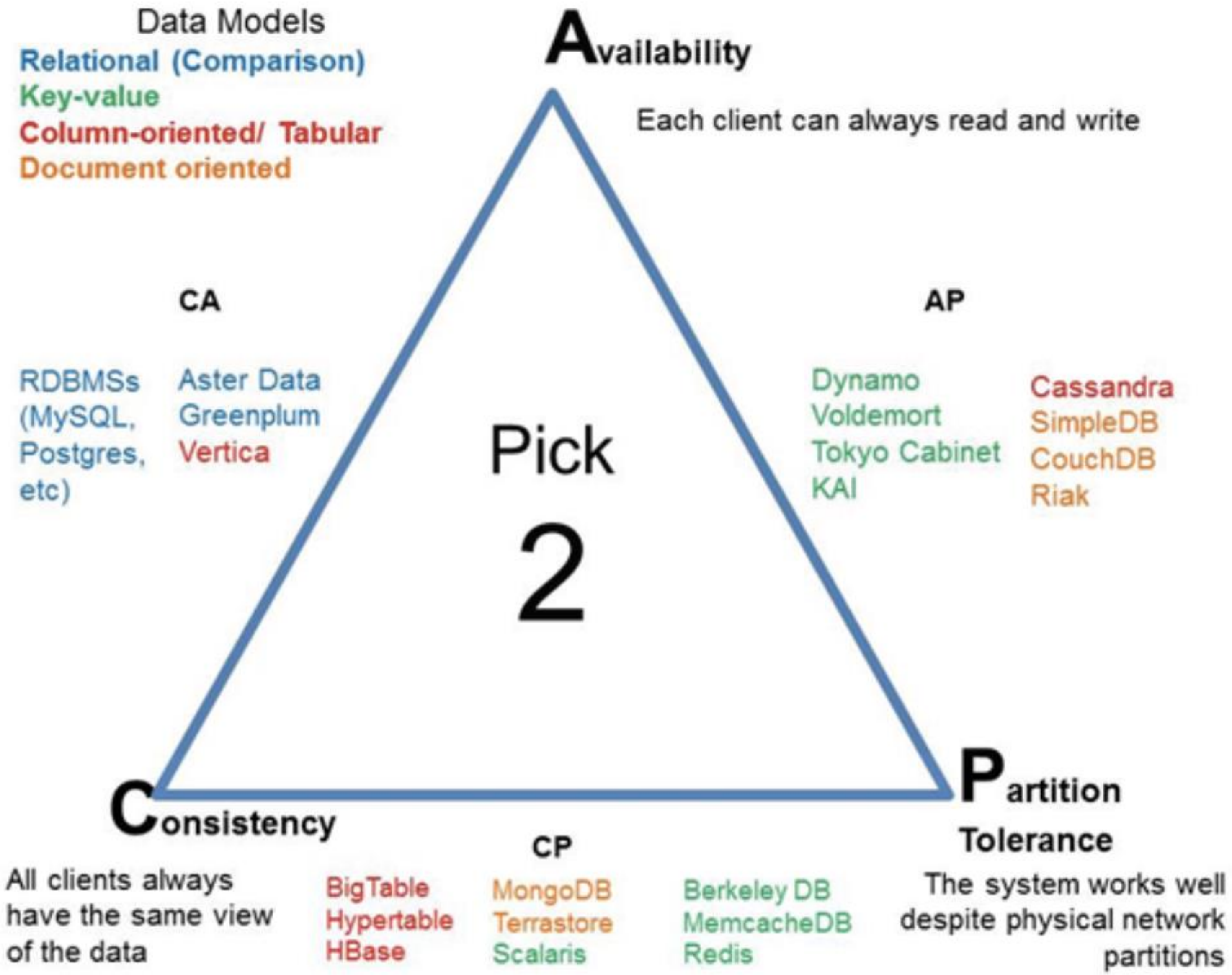
No ACID constraints

Gain performance and scalability

The CAP theorem (Eric Brewer, 2000) – 1/2

- Assumptions:
  - there are many nodes in the system
  - nodes contain **replicas of partitions** of the data
- There are 3 main properties for distributed management:
  1. **C**onsistency → A data item has the same value at the same time (to ensure coherency).
  2. **A**vailability → Data is available, even if a server is down.
  3. **P**artition Tolerance → A query must have an answer, even if the system is partitioned (unless there is a global failure).

The CAP theorem (Eric Brewer, 2000) – 2/2



There is still some controversy about this theorem …

NoSQL systems

- Several incarnations
  - MapReduce framework
  - Key-value stores
  - Document stores
  - Graph database systems

MapReduce Framework

- Originated at Google
- Open source implementation: Hadoop
- For processing and generating big data sets with a parallel, distributed algorithm on a cluster.

MapReduce, Characteristics

- No data model, data stored in files
- Functions:
  - map(), filtering and sorting ← divide problems into subproblems

    <u>Example</u>: sorting students by first name into queues, one queue for each name.
  - reduce(), summary operation ← do work on subproblems + combine results

    <u>Example</u>: counting the number of students in each queue, yielding name frequencies.
- System provides data processing “glue”, fault-tolerance, scalability

Key-Value Stores

- Extremely simple interface:
  - Data model: (key, value) pairs
  - Operations: Insert(key,value), Fetch(key), Update(key), Delete(key)
- Implementation: efficiency, scalability, fault-tolerance
  - Records distributed to nodes based on key
  - Replication
  - Single-record transactions, “eventual consistency”
- <u>Example systems</u>: Google BigTable, Amazon Dynamo, Cassandra, Voldemort, HBase, …

Document Stores

- Like Key-Value Stores except value is document
  - Data model: (key, document) pairs
  - Document: JSON, XML, other semi-structured formats
  - Basic operations: Insert(key,document), Fetch(key),
    Update(key), Delete(key)
  - Also Fetch based on document contents
- <u>Example systems</u>:
  - CouchDB, MongoDB, SimpleDB, …

Graph Database Systems

- Data model: nodes and edges
- Nodes may have properties (including ID)
- Edges may have labels or roles
- Interfaces and query languages vary
- Single-step versus "path expressions" versus full recursion
- Example systems:
    - Neo4j, FlockDB (Twitter), Pregel, …

Why are such (semi-structured) data stores needed?

- Semi-structured or flat files based data stores are best for massive data that is read, possibly frequently, but with minimal updates.
- There is much less overhead to process data in this format.
- We also have the flexibility to process data that doesn’t have a completely fixed structure.

What NoSQL should NOT be used for

- Anything that requires frequent updates as well as reads, or that requires high integrity and atomicity (ACID properties)
- Examples are similar to transaction databases for inventory and financial records
- This is not just a question of massive data or distributed processing!
- There are large, distributed relational databases like Visa or Amazon that need more structured data with transaction semantics
- These applications are better suited to relational databases even at large scale

In a nutshell: NoSQL is an alternative, non-traditional DB technology to be used in large scale environments where (ACID) transactions are not a priority.

What have we learned?

- The motivation for NoSQL (and how it differs from RDBMS)
- Concepts related to NoSQL (inc. “BASE” properties and types of NoSQL systems)
- Application areas of NoSQL