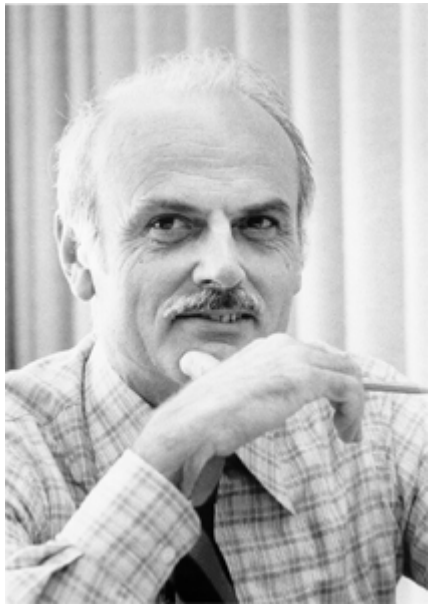
Introduction to NoSQL

Agenda

- History
- What is NoSQL
- Types of NoSQL
- The CAP theorem

History - RDBMS

- Relational DataBase Management Systems were invented in the 1970s.



E. F. Codd, "Relational Model of Data for Large Shared Data Banks"

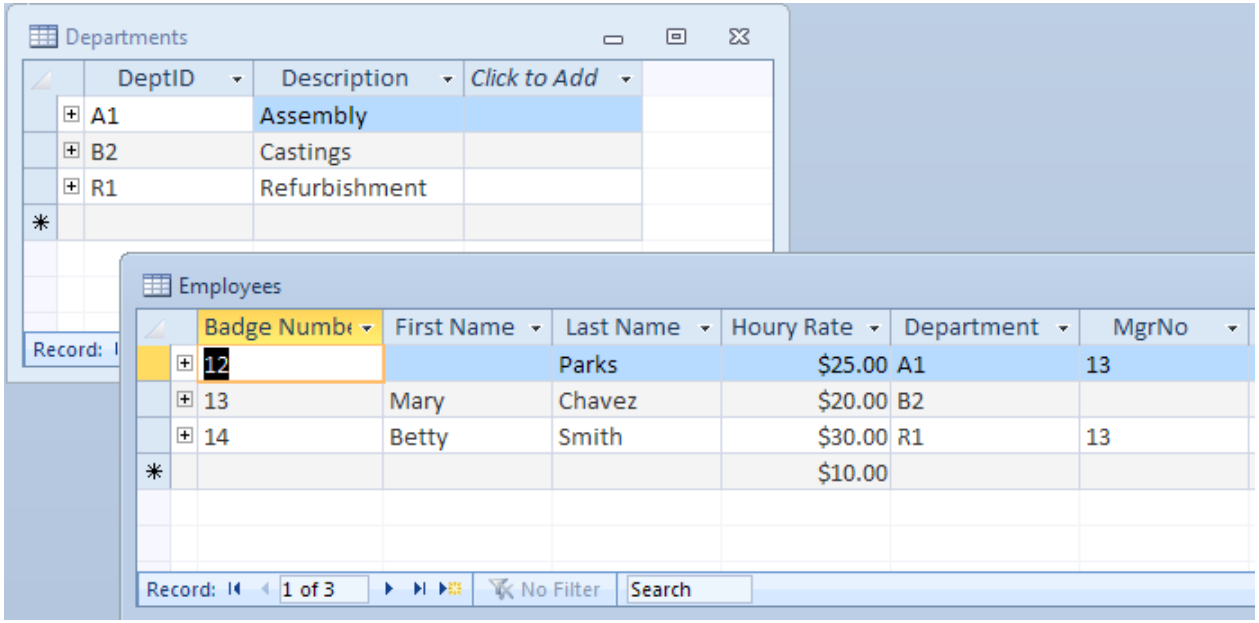
- RDBMS took over the world in early 1980s . . .

Codd's famous 12 rules published in 1985

- . . . and are still ruling the data management landscape.

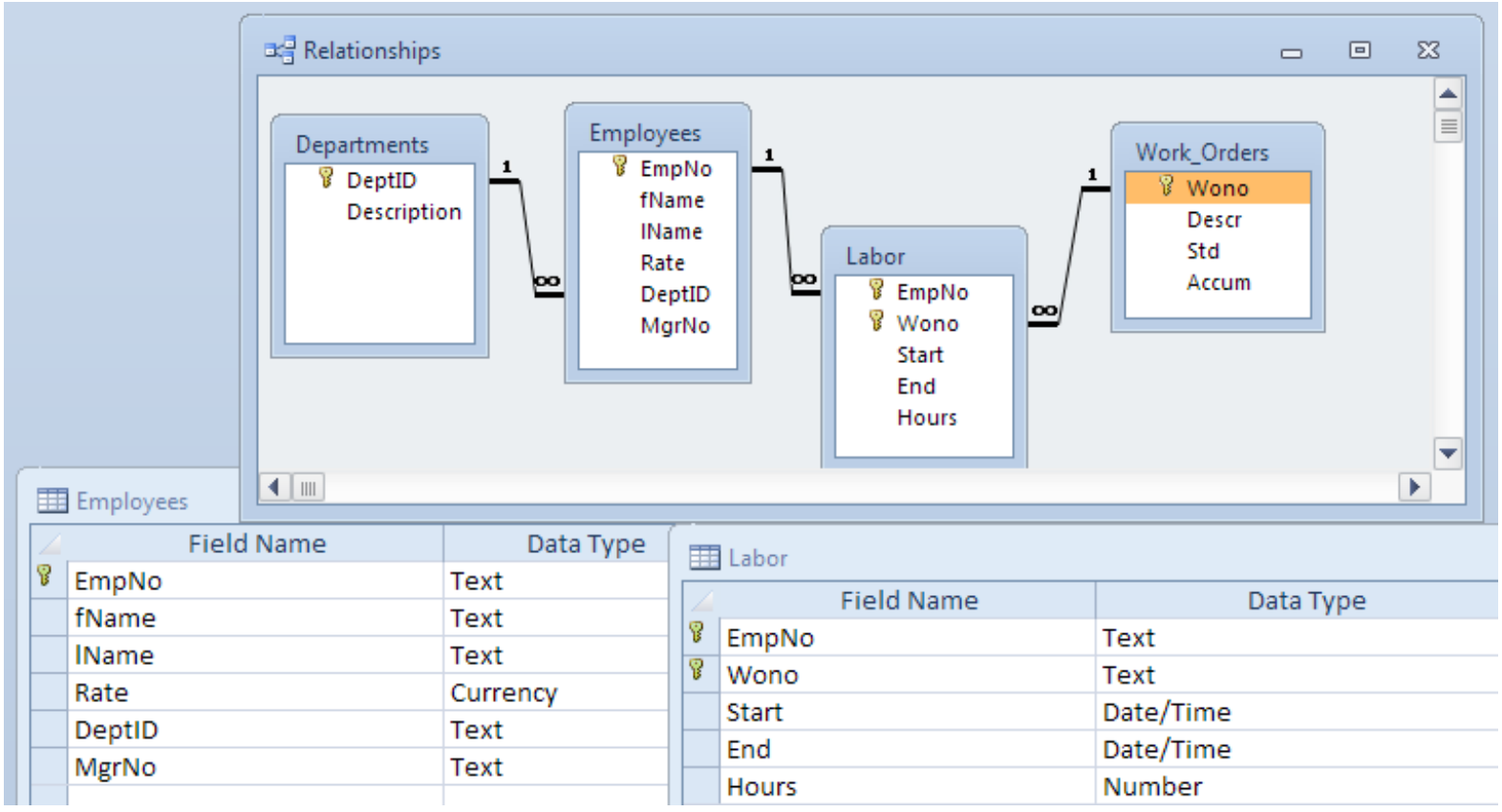
History - RDBMS

- Data is structured and stored in tables.



History - RDBMS

- Data is modelled into a schema using relational algebra.



History - RDBMS

RBDMS have ACID transactions.

- **A**tomic - each transaction is a logical unit and either completed entirely or not at all;
- **C**onsistent - after each transaction the data complies with the schema;
- **I**solated - simultaneous transactions (if allowed at all) are independent;
- **D**urable - modifications by a transaction are permanent.

History - RDBMS

Advantages of RDBMS.

- Data is always consistent.
- Standard language (SQL).
- Client-server architecture
  - allows easy integration of applications;
  - allows sharing and access control to data.
- Complex logical relationships in the data can be modelled following simple formal rules (relational algebra).

History - RDBMS

Difficulties with RDBMS for data science.

- Strict schema compliance limits data flexibility.
- Data that is accessed together is often scattered in multiple tables.
- Data is stored differently from the way it is used in applications.
- Not designed to *scale horizontally*.
- Not meant for analytics.

History - RDBMS

Types of scalability (IT lingo)

- "Scale up" or "scale vertically" means that you build a more powerful server to handle increased load.
- "Scale out" or "scale horizontally" is when you build a cluster of servers and distribute the load among them.
- RDBMS are unable to scale out because of strong data consistency.

History - RDBMS

History - RDBMS

"In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox. We shouldn't be trying for bigger computers, but for more systems of computers."

Rear Admiral Grace Murray Hopper (1906-1992)

History - RDBMS

Why RDBMS can't handle analytics?

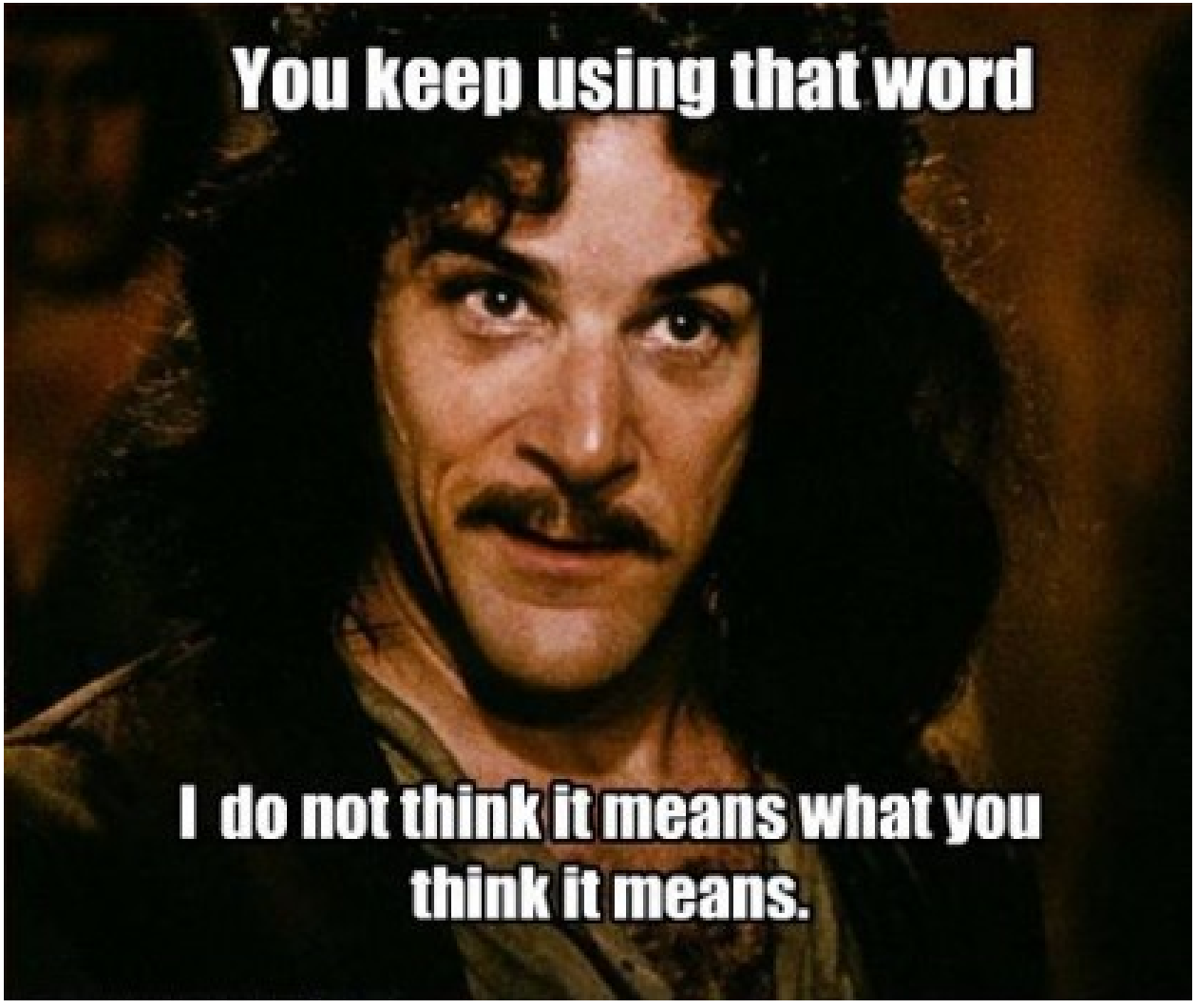
- RDBMS use indices for fast access to data.
- This is great for transactions – they touch individual records.
- If your query touches a large fraction of your database, indices don't help and RDBMS run into performance problems.
- OLAP systems (1990s) address this issue, but they work on static data.

History

In the 21-st century

- Development of SOAP and XML – integration of applications is now simpler than using RDBMS.
- Large volumes of data generated on the Web – Internet giants (Google, Amazon) start inventing data management systems that scale out.

What is NoSQL?



What is NoSQL?

- NoSQL is not a standard, there is neither definition nor specification.
- The origins of the term are unclear – different sources disagree.
- Some say it stands for "Not Only SQL".
- Some say it started in 2009 with Twitter "#nosql" to organize a conference on distributed databases.
- The term definitely existed before 2009.

What is NoSQL?

Some common characteristics of NoSQL databases.

- Open-source projects.
- Non-relational.
- Most are cluster-friendly.
- Schema-less.
- Web-friendly access

What is NoSQL?

Some examples . . .

- just to name a few.
- Long list at `http://nosql-database.org/`

Types of NoSQL databases

Four flavours of NoSQL

- Key-Value
- Document
- Column-families
- Graph databases

Types of NoSQL databases

Key-Value

- Each data item has a unique key.
- The value can be anything.

- Most basic NoSQL.
- Can handle huge amounts of data very efficiently.

Types of NoSQL databases

Document

- Each data item is a "document".
- A document is a collection of key-value pairs.
- No schema – documents are not required to have the same keys and if they do the corresponding values don't have to be the same type.
- Documents can be nested.

Types of NoSQL databases

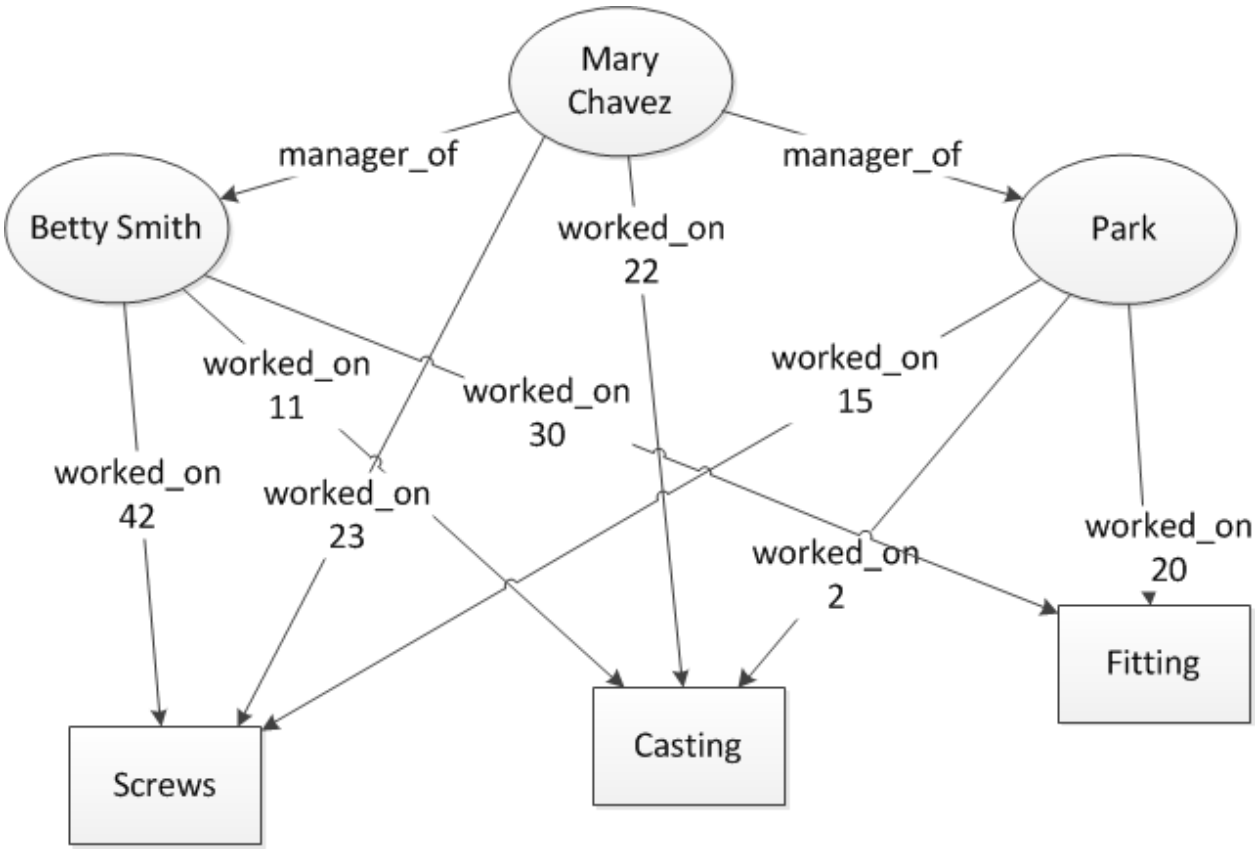
Column-family

- Another variation of the Key-Value type
- The Key is the column name and the Value is the entire column.
- All values in the same column are of the same type.
- "Rows" can be accessed using a "row key".

Types of NoSQL databases

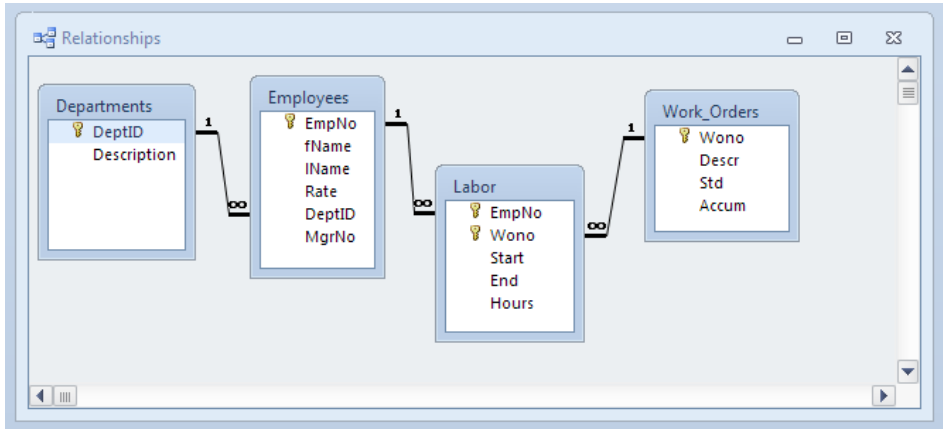
Graph

- Data is stored in lists of nodes and edges.
- Each node is an entity. Each edge is a relationship.



Aggregate oriented

- The key-value, document and column-family type can be viewed as aggregate-oriented storage.
- An aggregate is the data that we like to see together, e.g. work order.
- In RDBMS the aggregate is made of rows collected from multiple tables.



Aggregate oriented

Labor Query1

Work Order	Description	Start Dtae	End Date	Hours Work	Badge Numbe	First Name	Last Name	Houry Rate
A1	Casting	10/10/2001	10/10/2001	11	14	Betty	Smith	$30.00
A1	Casting	07/03/2003	08/03/2003	22	13	Mary	Chavez	$20.00
A1	Casting	25/10/2002		2	12		Parks	$25.00
A2	Screws	01/02/2001	28/02/2001	42	14	Betty	Smith	$30.00
A2	Screws	03/10/2001	04/10/2001	23	13	Mary	Chavez	$20.00
A2	Screws	09/11/2001	09/11/2001	15	12		Parks	$25.00
B3	Fitting	01/01/2002	28/02/2002	30	14	Betty	Smith	$30.00
B3	Fitting	18/03/2002	20/03/2002	20	12		Parks	$25.00

```
{ "Work Order": "A1",
  "Description": "Casting",
  "Labours": [
     { "Start Date": "10/10/2001", "End Date": "10/10/2001", "Hours Worked": 11,
       "Employee": { "Badge No": 14, "First Name": "Betty", "Last Name": "Smith" } },
     { "Start Date": "07/03/2003", "End Date": "08/03/2003", "Hours Worked": 22,
       "Employee": { "Badge No": 13, "First Name": "Mary", "Last Name": "Chavez" } },
     { "Start Date": "25/10/2002", "End Date": "", "Hours Worked": 2,
       "Employee": { "Badge No": 12, "First Name": "", "Last Name": "Parks" } }
  ]
}
```

Aggregate oriented

- This is great if we always want to look at work orders.
- What if we want to change the aggregation unit? E.g. look at single employee's work on all work orders?

Aggregate oriented

- This is great if we always want to look at labour.
- What if we want to change the aggregation unit? E.g. look at single employee's work on all work orders?
- Very easy to do in SQL.
- Not so easy in NoSQL – requires a Map-Reduce job.

Schema-less?

- Is NoSQL really schema-less? NO!
- The schema-less-ness allows us to store new types and structures of data.
- However, we still use a schema when we access and process the data.

  We assume that "Start Date" is a date and that "Hourly Rate" is a number and is called "Hourly Rate" and not "Hourly Wage".

- In RDBMS the schema is enforced when data enters the system;
- with NoSQL the schema is implied when accessing the data.

Distributed databases

- Replication - the same data resides on multiple servers:
  - improves availability
  - allows distributed reads
  - writes are expensive
- Partitioning (sharding) - data is distributed across the cluster:
  - scale out is the only way to handle big data
  - entire aggregates are stored on each server
  - ensuring the shards are close to equal in size is important for load balancing
- Replication + Partitioning - data is distributed across the cluster with multiple copies of each chunk:
  - Fault-tolerance

Distributed Databases

- Transactions in NoSQL are not ACID! However:
- So long as transactions don't cross aggregate boundaries, they are atomic, isolated and durable.
- What about consistent?

Distributed Databases: Consistency vs Availability

- In RDBMS consistency is guaranteed by the schema and ACID transactions.
- In NoSQL no schema is enforced and transactions are not ACID.
- The concept of *eventual consistency* arises:
  - data may not be consistent after each transaction; however
  - consistency is restored at a later time.

Distributed Databases: Consistency vs Availability

- In RDBMS all data is available for as long as the server is running.
- In NoSQL availability depends on the setup of the cluster.
  - What happens if a part of the cluster gets disconnected?
  - Do you halt or do you keep running with only part of the data?

Distributed Databases: Consistency vs Availability

Example

- Two customers are trying to book the same flight at the same time.
- How does the booking system decide who gets the last seat if cluster is partitioned?
- Two Carleton students are trying to register into the same course at the same time.
- Same question?

Distributed Databases: Consistency vs Availability

The **CAP** Theorem

- **C**onsistency - after data is modified, all clients read the same data.
- **A**vailability - system is always on and all data can be read.
- **P**artition Tolerance - system continues to function even if some servers are not able to communicate with other servers.

- The CAP theorem says that it is impossible to have all three – you can choose any two.

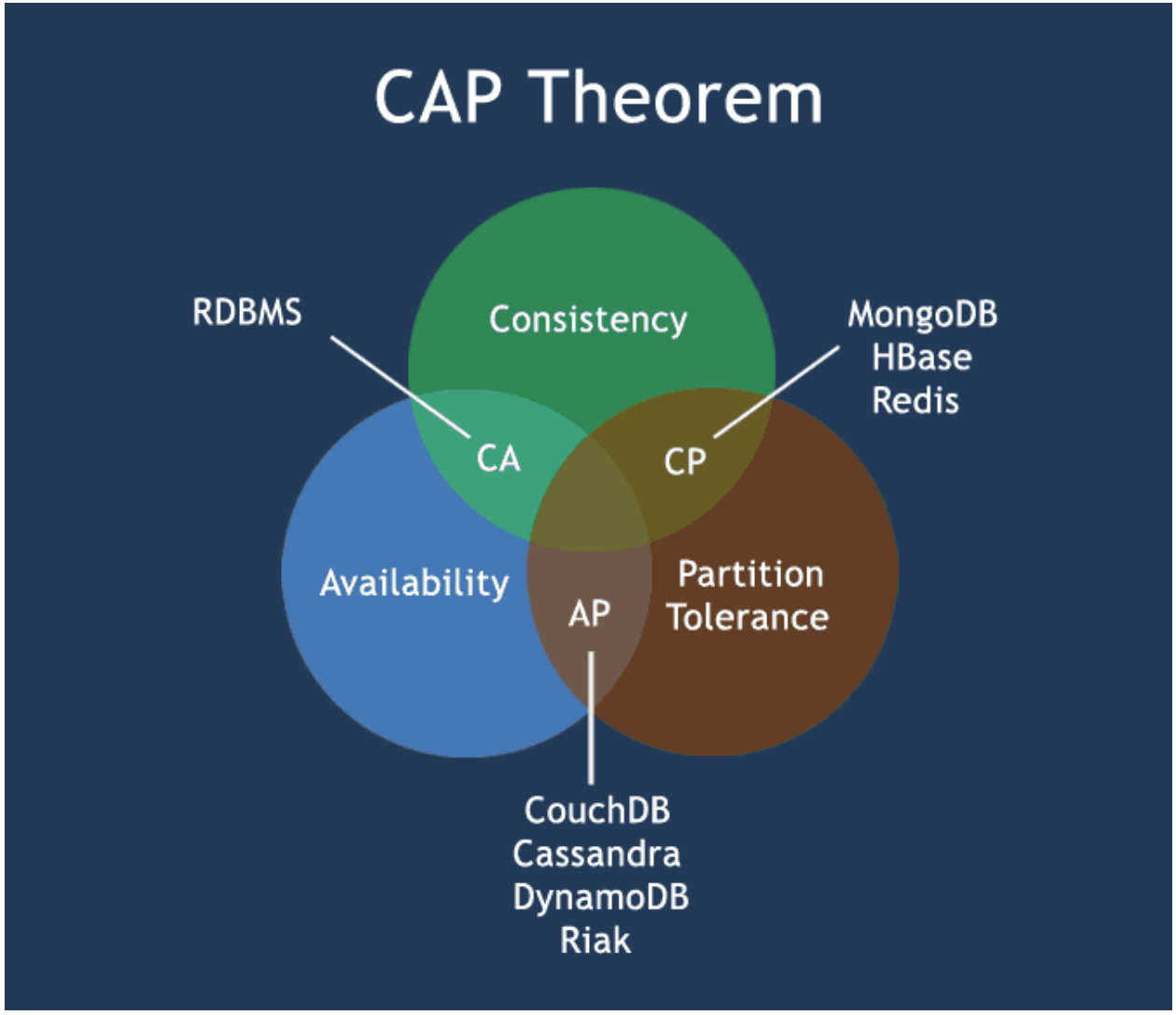
Distributed Databases: Consistency vs Availability

The CAP Theorem

- **CA** - A single site cluster where communication is always reliable. RDBMS are in this category.
- **CP** - Availability is compromised when network partition occurs – either the entire system or part of the data become unavailable until communication is restored. The available data is correct.
- **AP** - The system continues to function under network partition, however some of the data may be incorrect. Consistency is eventually restored.

Distributed Databases: Consistency vs Availability

The CAP Theorem



Why use NoSQL?

- If data is too much for one server, RDBMS is no longer an option.
  - Not only if disk space is not enough for all data;
  - also, if memory is not enough to hold the working set!
- Even if data is not too much, NoSQL gives flexibility – data can be stored as it is internally represented in your application.

New kid on the block

SciDB

- Data model is multidimensional array.
- Data items are points in d-dimensional space. At each point you can set multiple key-values.
- Designed for scientific data.
- In-data analytics.
- ACID.
- AC – no partition tolerance.