CS4225/CS5425 Big Data Systems for Data Science

NoSQL Overview

Bryan Hooi
School of Computing
National University of Singapore
bhooi@comp.nus.edu.sg

School of Computing

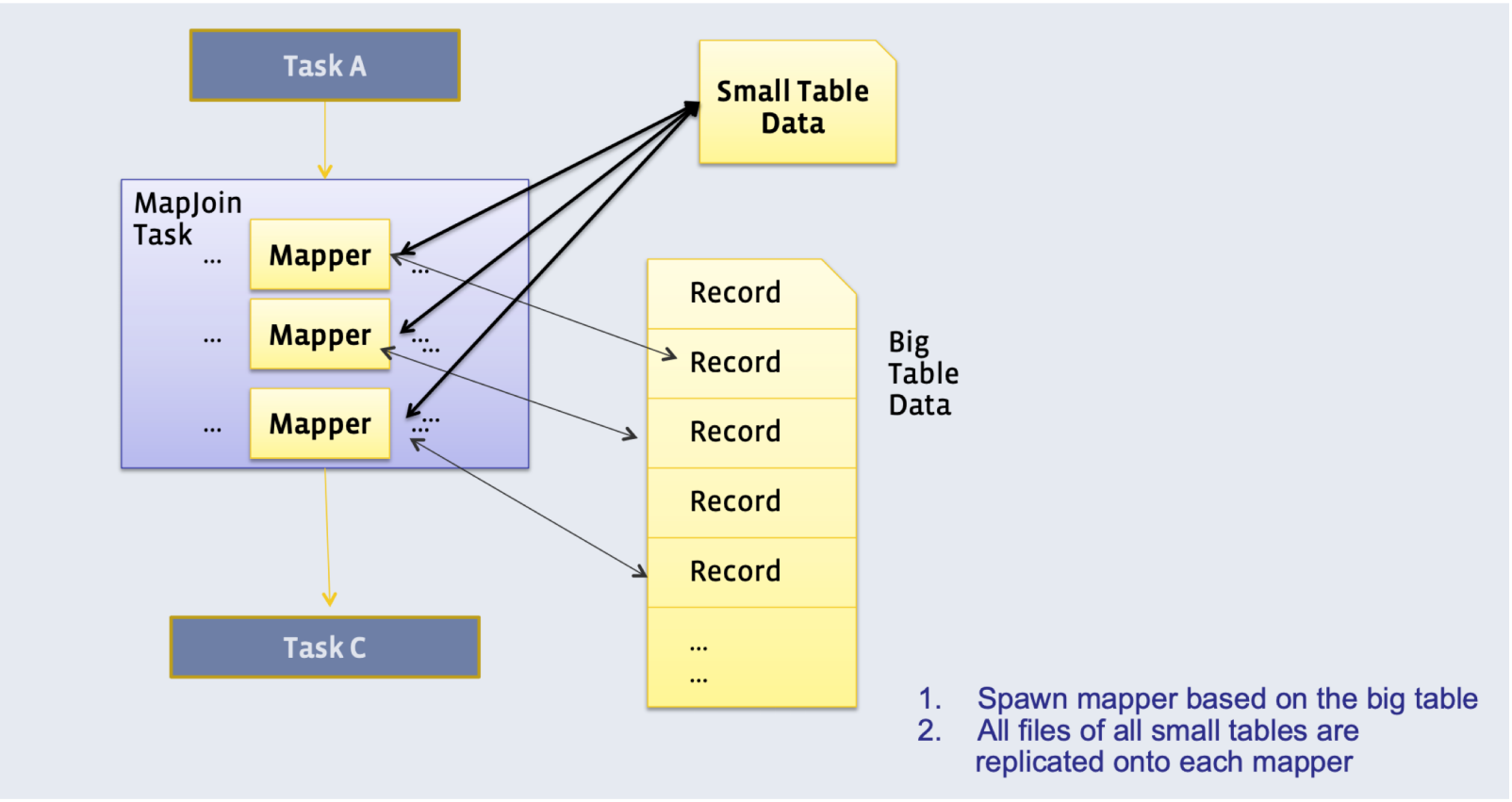
Announcements

- Since we are quite ahead of schedule (compared to the syllabus / last semester), next week will be a shorter than usual lecture
- HW1 will be released by this weekend (due 10 Oct 11.59pm)

Week	Date	Topics	Tutorial	Due Dates
1	12 Aug	Overview and Introduction		
2	29 Aug	MapReduce - Introduction		
3	26 Aug	MapReduce and Relational Databases		
4	2 Sep	MapReduce and Data Mining	Tutorial: Hadoop	Assignment 1 released
5	9 Sep	NoSQL Overview 1		
6	16 Sep	NoSQL Overview 2		
Recess				
7	30 Sep	Apache Spark 1	Tutorial: NoSQL & Spark	Assignment 1 due, Assignment 2 released (3 Oct)
8	7 Oct	Apache Spark 2		
9	14 Oct	Large Graph Processing 1	Tutorial: Graph Processing	
10	21 Oct	Large Graph Processing 2		
11	28 Oct	Stream Processing	Tutorial: Stream Processing	Assignment 2 due (31 Oct)
12	4 Nov	Deepavali – No Class		
13	11 Nov	**Test**		

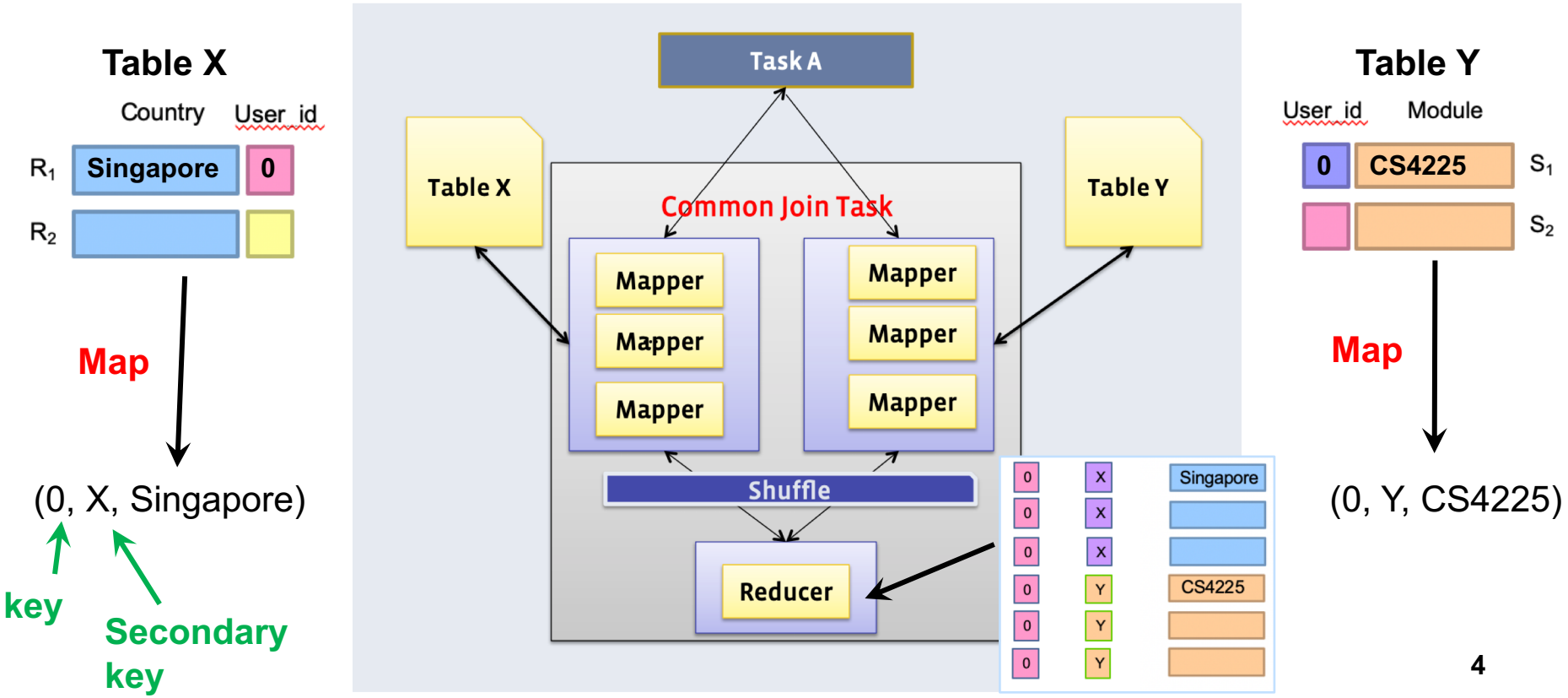
Recap: Map-Side Join

- Requires one of the tables to fit in memory
  - All mappers store a copy of the small table
  - They iterate over the big table, and join the records with the small table



Recap: Reduce-side ('Common') Join

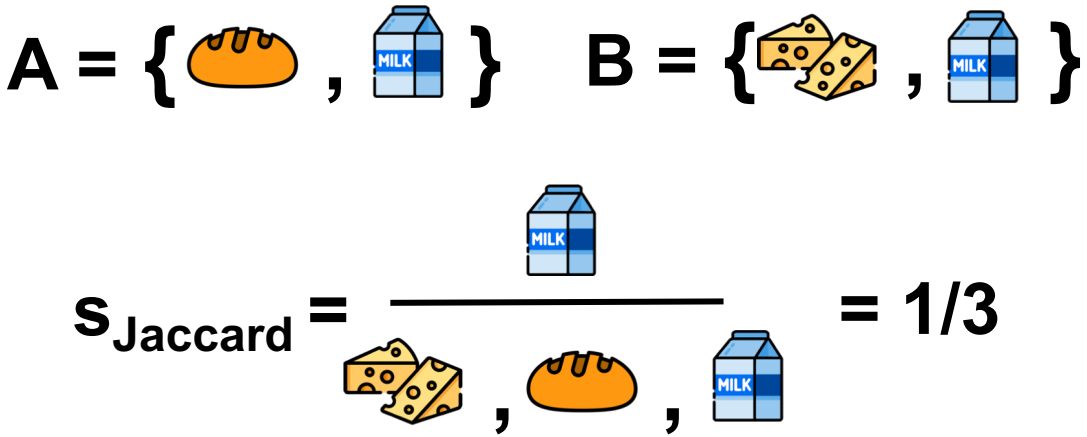
- Doesn't require a dataset to fit in memory, but slower than map-side join
  - Different mappers operate on each table, and emit records, with key as the variable to join by



Recap: Jaccard similarity/distance

- **Jaccard Similarity**
  (between **sets** A and B)

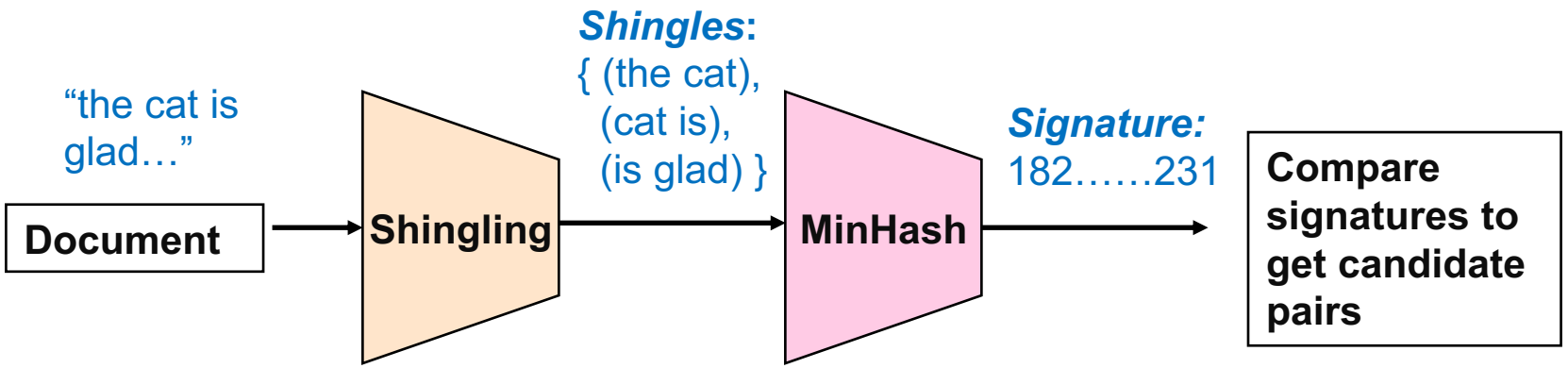
$$s_{\text{Jaccard}}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$



- **Jaccard Distance**

$$d_{\text{Jaccard}}(A, B) = 1 - s_{\text{Jaccard}}(A, B)$$

Recap: Near-Duplicate Document Search



Recap: K-Means Algorithm

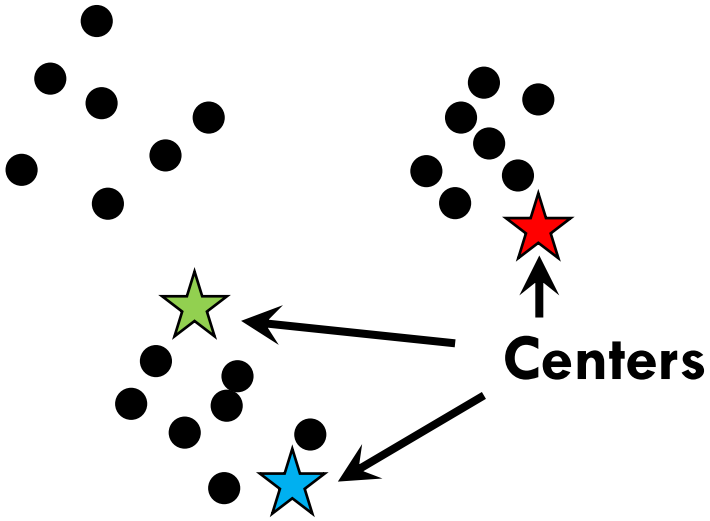
1. Initialization: Pick K random points as centers

2. Repeat:

a) Assignment: assign each point to nearest cluster

b) Update: move each cluster center to average of its assigned points

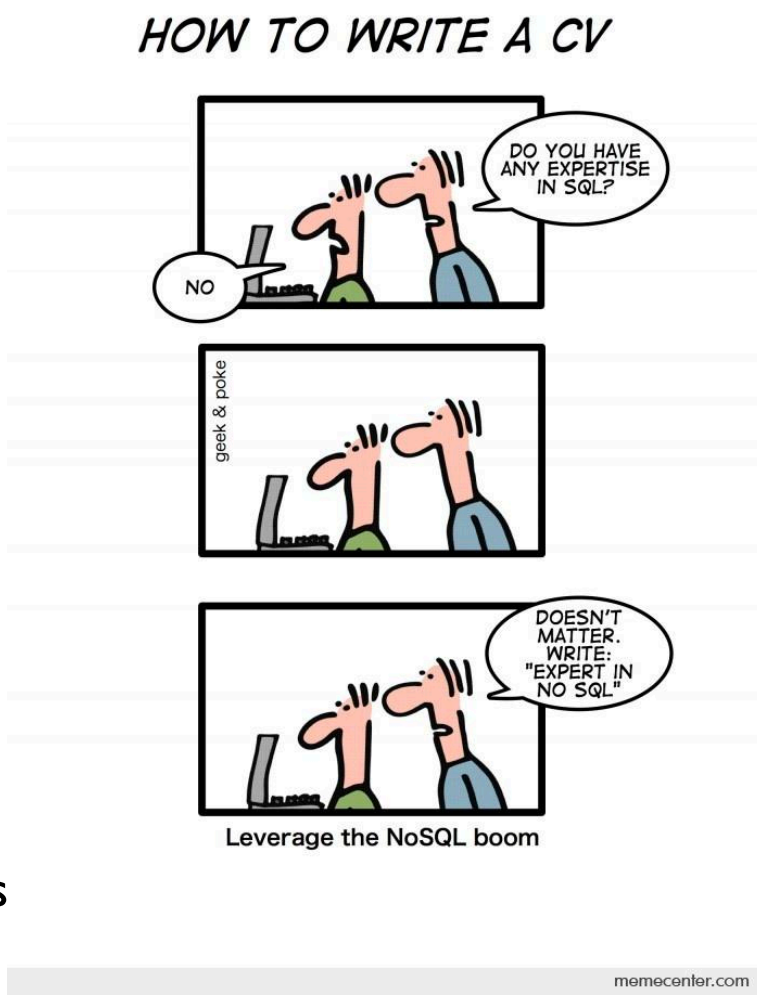
Stop if no assignments change



Today's Plan

1. **What is NoSQL?**
2. Major types of NoSQL systems
3. Key concepts

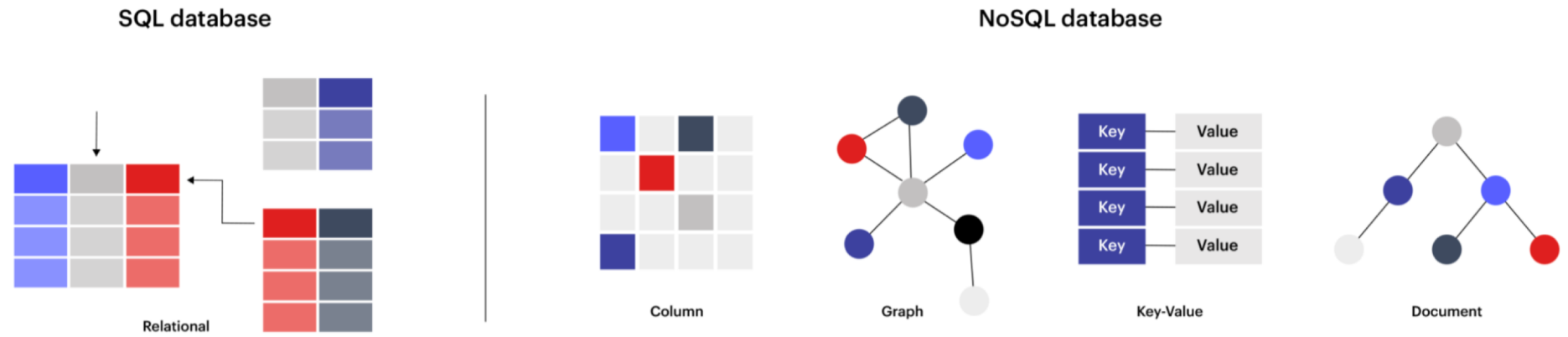
- We will see, but **no need to remember for test / HW:**
  - Historical details
  - Writing code in any NoSQL systems (e.g. MongoDB)
  - Details about any specific NoSQL systems



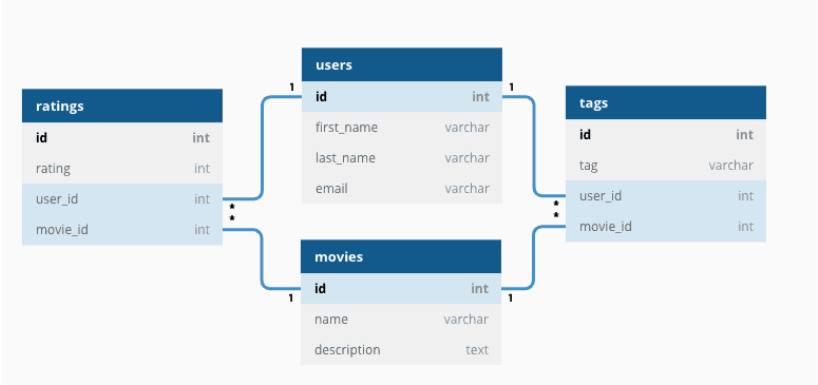
Source: Cattell (2010). Scalable SQL and NoSQL Data Stores. *SIGMOD Record*.

What is NoSQL?

- NoSQL mainly refers to a **non-relational database**, i.e. it stores data in a format other than relational tables
- ”SQL” here refers to traditional relational database management systems (“DBMS”)
  - (This is a slight misnomer; here “SQL” is used to refer to relational databases, not the querying language)
- NoSQL has come to stand for “Not Only SQL”, i.e. using relational and non-relational databases alongside one another, each for the tasks they are most suited for



What is NoSQL?



Traditional Relational Database Management System (DBMS)

```
>>> db.inventory.find()
[
  {
    _id: ObjectId("6138d8b1611d9dc433ad73dd"),
    item: 'canvas',
    qty: 100,
    tags: [ 'cotton' ],
    size: { h: 28, w: 35.5, uom: 'cm' }
  },
  {
    _id: ObjectId("6138d917611d9dc433ad73de"),
    item: 'table',
    qty: 5,
    price: 10
  }
]
```

mongoDB. (Document Store)

Phone directory

Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334

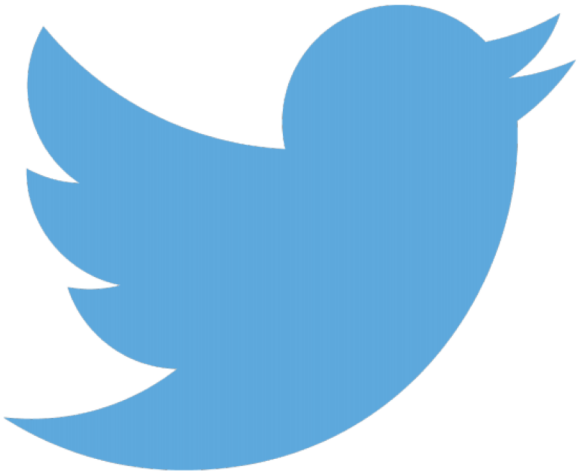
redis (Key-Value Store)

Metaphor: NoSQL = No Rules?

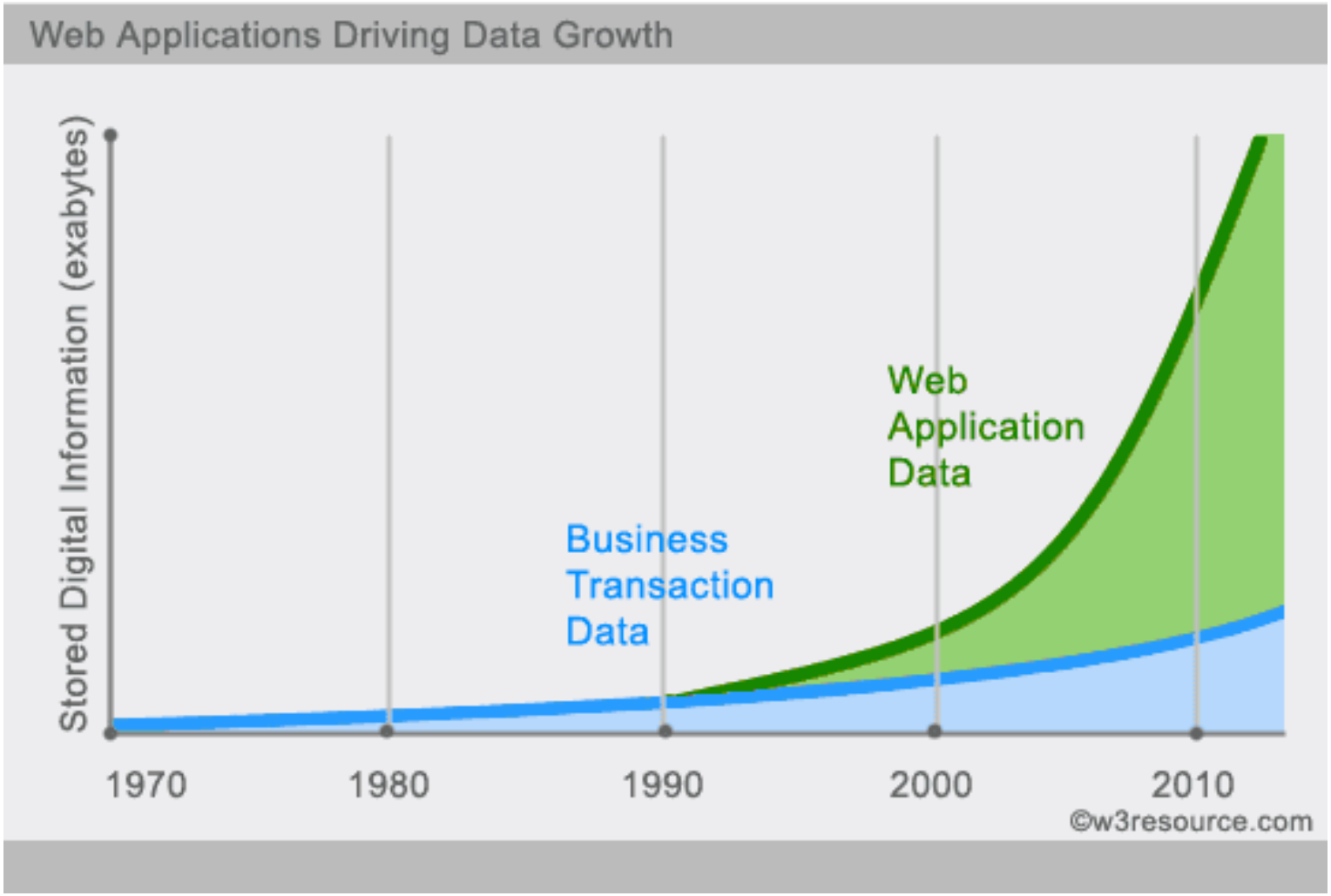
Relational Databases

Non-Relational Databases (NoSQL)

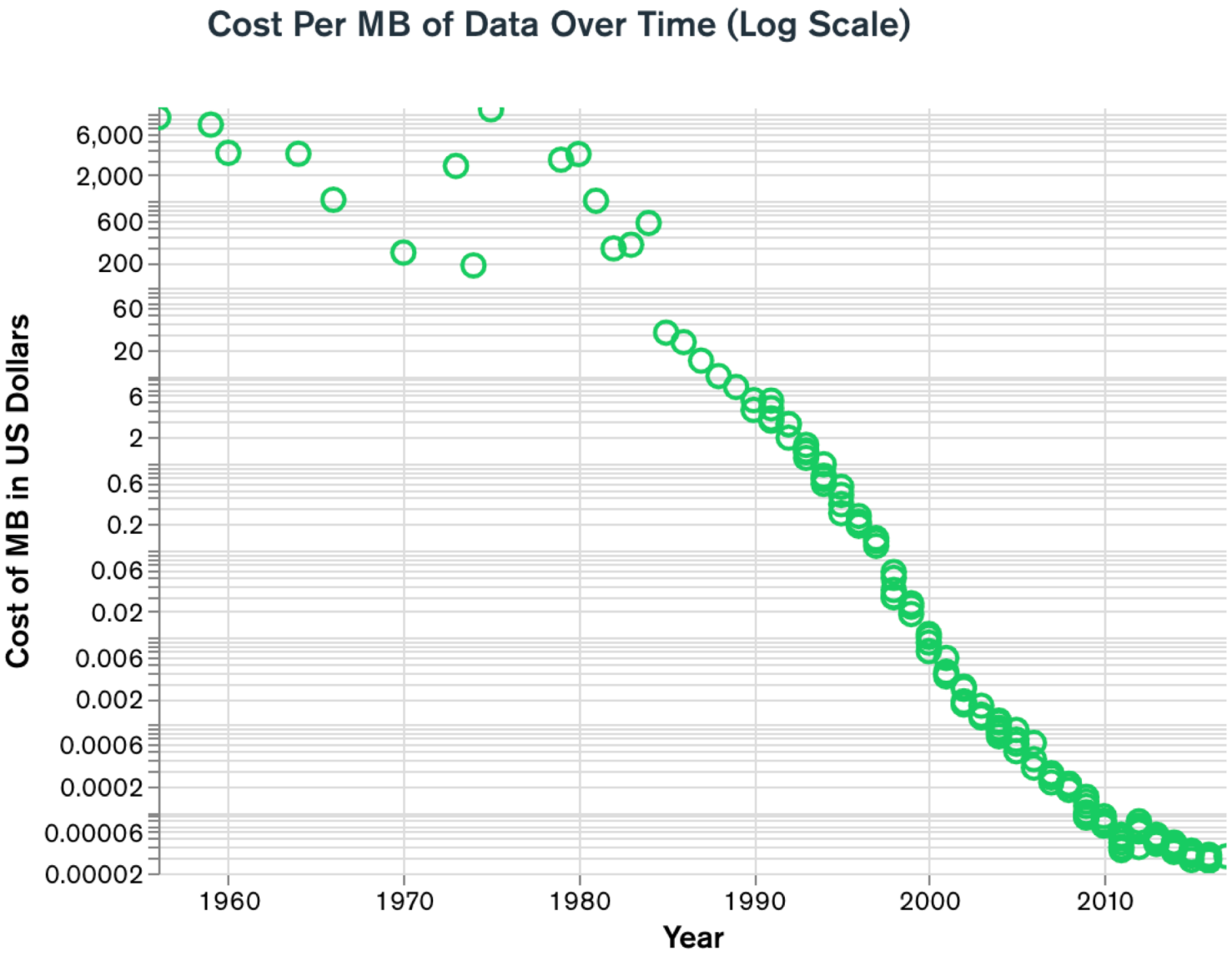
Who uses NoSQL?



Why NoSQL: Growth in Web Applications



Why NoSQL: Volume, Variety, Velocity



A Brief Overview of NoSQL Systems

1. Horizontally scalability
2. Replicate/distribute data over many servers

Volume, Velocity

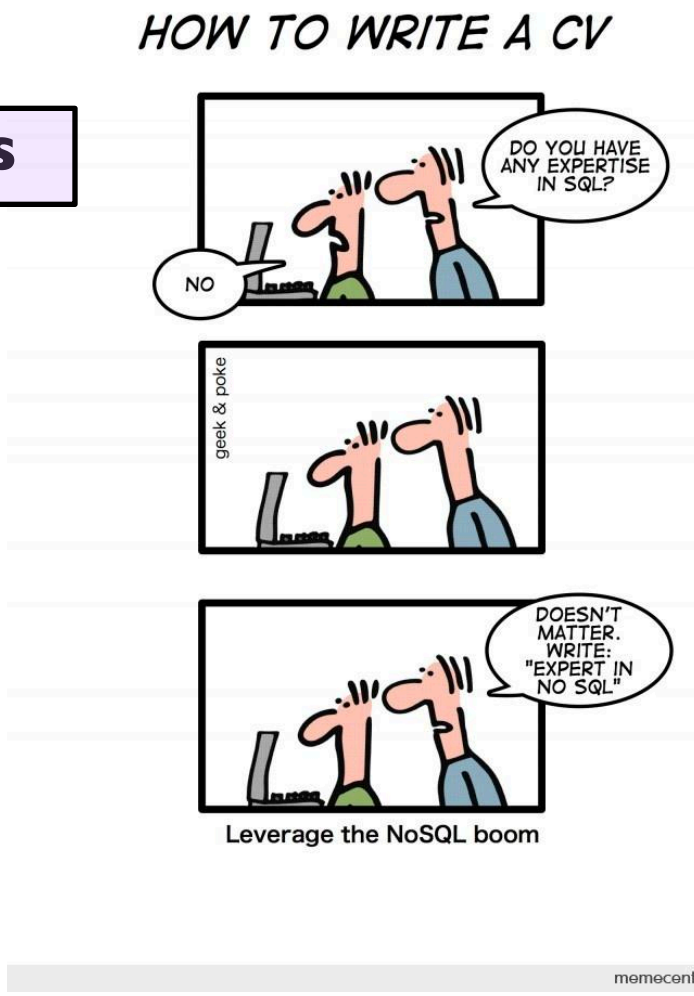
3. Simple call interface
4. Often weaker concurrency model than RDBMS
5. Efficient use of distributed indexes and RAM
6. Flexible schemas

Variety

Source: Cattell (2010). Scalable SQL and NoSQL Data Stores. *SIGMOD Record*.

Today's Plan

1. What is NoSQL?
2. **Major types of NoSQL systems**
3. Key concepts



Source: Cattell (2010). Scalable SQL and NoSQL Data Stores. *SIGMOD Record*.

(Major) Types of NoSQL databases

- Key-value stores

- Wide column databases

- Document stores

- Graph databases

Key-Value Stores

Source: Wikipedia (Keychain)

Key-Value Stores: Data Model

Phone directory

Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334

MAC table

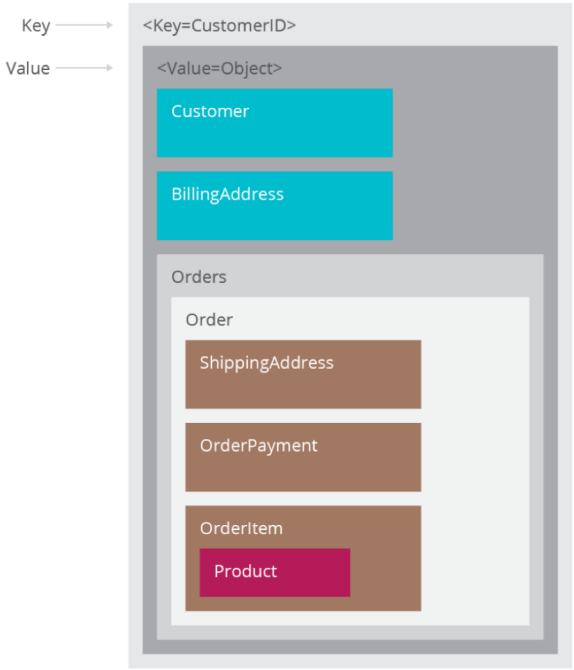
Key	Value
10.94.214.172	3c:22:fb:86:c1:b1
10.94.214.173	00:0a:95:9d:68:16
10.94.214.174	3c:1b:fb:45:c4:b1

- Stores associations between keys and values
  - Based on Amazon's Dynamo paper (2007)
- Keys are usually primitives and can be queried
  - For example, ints, strings, raw bytes, etc.
- Values can be primitive or complex; usually cannot be queried
  - Examples: ints, strings, lists, JSON, HTML fragments, BLOB (basic large object), etc.

DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store." *ACM SIGOPS operating systems review* 41.6 (2007): 205-220.

Key-Value Stores: Operations

- Very simple API:
  - Get – fetch value associated with key
  - Put – set value associated with key
- Optional operations:
  - Multi-get
  - Multi-put
  - Range queries
- Suitable for:
  - Small continuous read and writes
  - Storing 'basic' information (e.g. raw chunks of bytes), or no clear schema
  - When complex queries are not required / rarely required
- Example Applications
  - Storing user sessions
  - Caches
  - User data that is often processed individually: e.g. patient medical data?
    - Only if no queries like 'How many patients who took X treatment recovered?'



Key-Value Stores: Implementation

- Non-persistent:
  - Just a big in-memory hash table
  - Examples: Memcached, Redis
    - (But: these can also back up the data to disk periodically)
- Persistent
  - Data is stored persistently to disk
  - Examples: RocksDB, Dynamo, Riak


Document Stores

Document Stores: Document Model

- A database can have multiple **collections**
- Collections have multiple **documents**
- A document is a JSON-like object: it has **fields and values**
  - Different documents can have different fields
  - Can be nested: i.e. JSON objects as values

DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store." *ACM SIGOPS operating systems review* 41.6 (2007): 205-220.

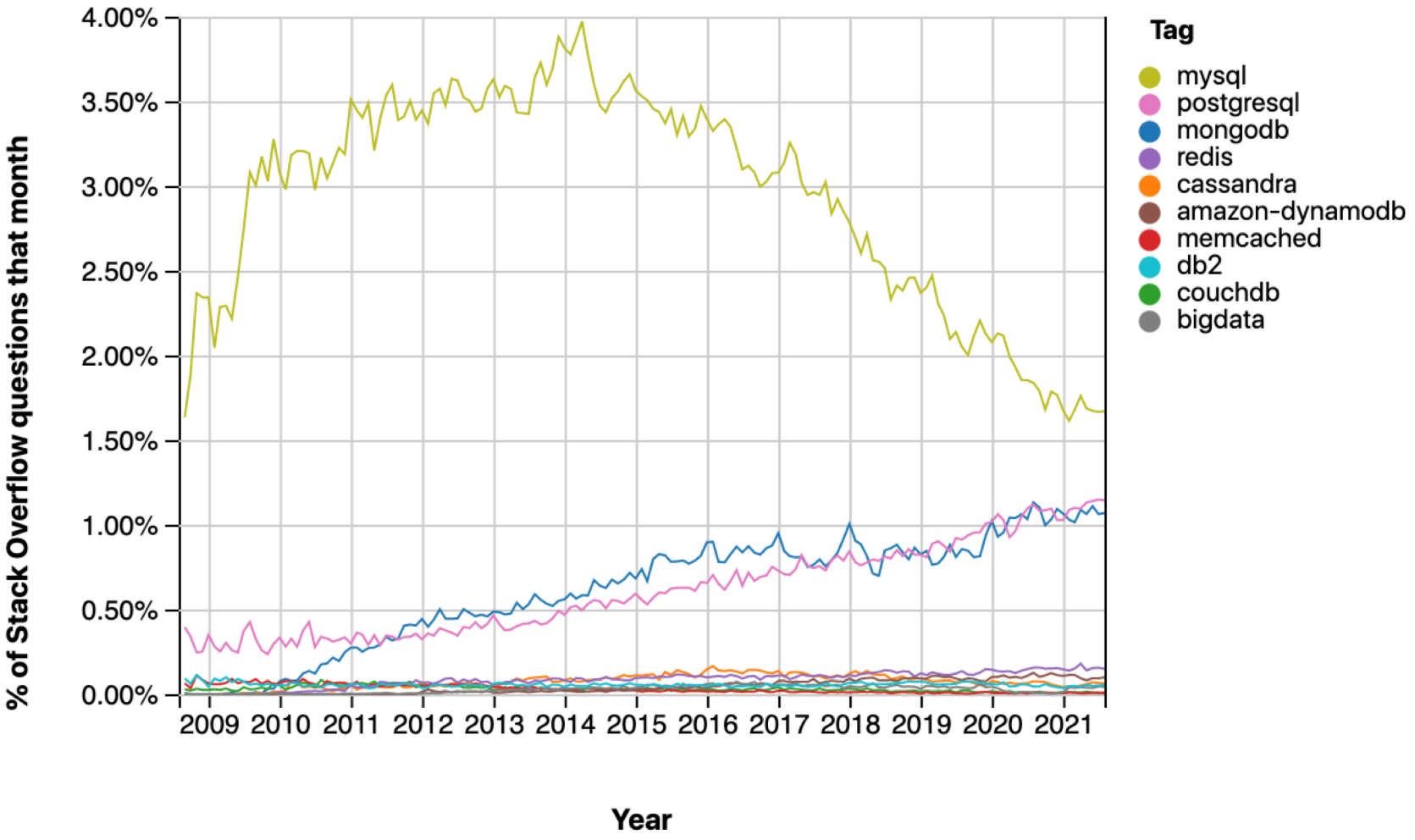
Document Stores: Querying

- Unlike (basic) key value stores, document stores allow some querying based on the content of a document
- CRUD = Create, Read, Update, Delete

Popularity of DBMS

Rank			DBMS	Database Model	Score		
Sep 2021	Aug 2021	Sep 2020			Sep 2021	Aug 2021	Sep 2020
1.	1.	1.	Oracle	Relational, Multi-model	1271.55	+2.29	-97.82
2.	2.	2.	MySQL	Relational, Multi-model	1212.52	-25.69	-51.72
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	970.85	-2.50	-91.91
4.	4.	4.	PostgreSQL	Relational, Multi-model	577.50	+0.45	+35.22
5.	5.	5.	MongoDB	Document, Multi-model	496.50	-0.04	+50.02
6.	6.	↑ 7.	Redis	Key-value, Multi-model	171.94	+2.05	+20.08
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	166.56	+1.09	+5.32
8.	8.	8.	Elasticsearch	Search engine, Multi-model	160.24	+3.16	+9.74
9.	9.	9.	SQLite	Relational	128.65	-1.16	+1.98
10.	↑ 11.	10.	Cassandra	Wide column	118.99	+5.33	-0.18
11.	↓ 10.	11.	Microsoft Access	Relational	116.94	+2.10	-1.51
12.	12.	12.	MariaDB	Relational, Multi-model	100.70	+1.72	+9.09
13.	13.	13.	Splunk	Search engine	91.61	+1.01	+3.71
14.	14.	↑ 15.	Hive	Relational	85.58	+1.64	+14.41
15.	15.	↑ 17.	Microsoft Azure SQL Database	Relational, Multi-model	78.26	+3.11	+17.81
16.	16.	16.	Amazon DynamoDB	Multi-model	76.93	+2.03	+10.75
17.	17.	↓ 14.	Teradata	Relational, Multi-model	69.68	+0.86	-6.72
18.	18.	↑ 21.	Neo4j	Graph	57.63	+0.68	+7.00
19.	19.	19.	SAP HANA	Relational, Multi-model	56.24	+0.66	+3.38
20.	↑ 21.	↑ 23.	FileMaker	Relational	52.32	+2.04	+4.75

Stack Overflow Trends



Source: https://insights.stackoverflow.com/

CRUD: Create

In MongoDB

```
db.users.insert (
   {
     name: "sue",
     age: 26,
     status: "A"
   }
)
```

collection

field: value

field: value

field: value

document

In SQL

```
INSERT INTO users
            ( name, age, status )
VALUES      ( "sue", 26, "A" )
```

table

columns

values/row

CRUD: Read

In MongoDB

```
db.users.find(                          <-- collection
    { age: { $gt: 18 } },               <-- query criteria
    { name: 1, address: 1 }             <-- projection
).limit(5)                              <-- cursor modifier
```

In SQL

```
SELECT _id, name, address   <-- projection
FROM   users                <-- table
WHERE  age > 18             <-- select criteria
LIMIT  5                    <-- cursor modifier
```

CRUD: Update

In MongoDB

```
db.users.update(                          <-- collection
    { age: { $gt: 18 } },                 <-- update criteria
    { $set: { status: "A" } },            <-- update action
    { multi: true }                       <-- update option
)
```

In SQL

```
UPDATE users              <-- table
SET    status = 'A'       <-- update action
WHERE  age > 18           <-- update criteria
```

CRUD: Delete

In MongoDB

```
db.users.remove(
    { status: "D" }
)
```

collection

remove criteria

In SQL

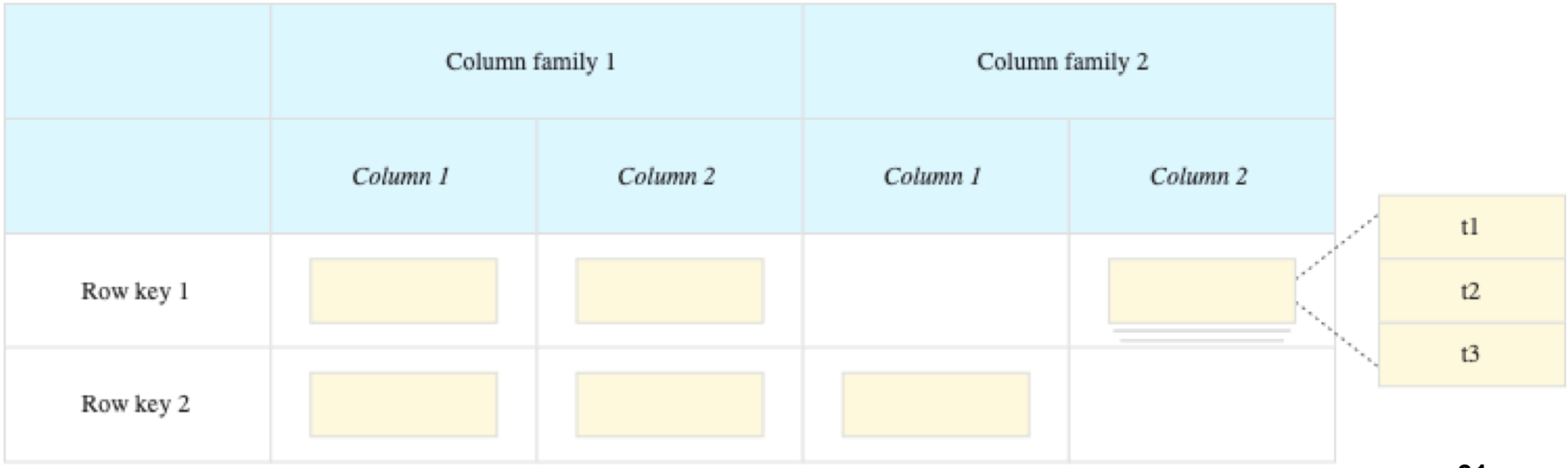
```
DELETE FROM users
WHERE  status = 'D'
```

table

delete criteria

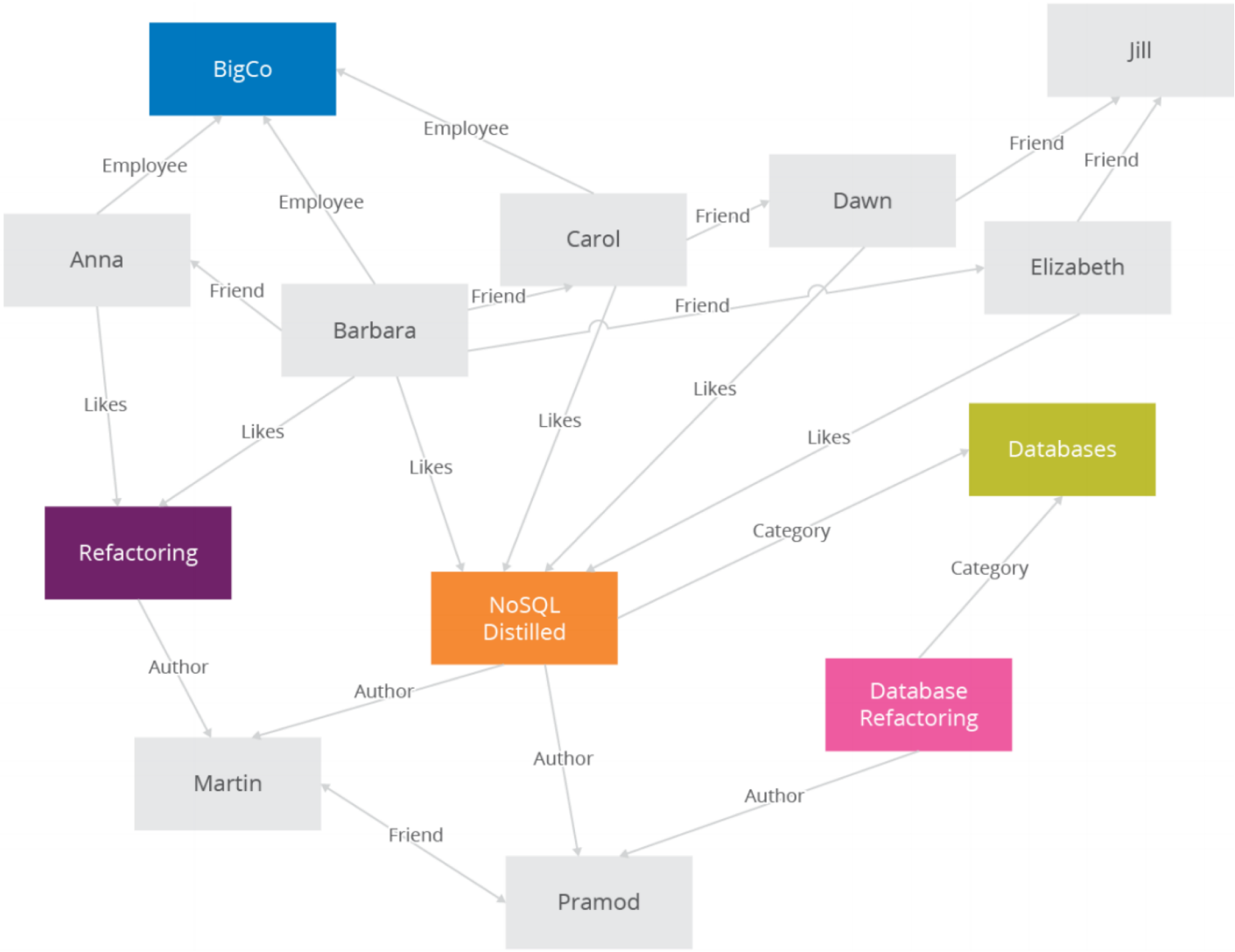
Wide Column Stores

- Rows describe entities
- Related groups of columns are grouped as **column families**
- **Sparsity**: if a column is not used for a row, it doesn't use space
- Examples: BigTable, Cassandra, HBase



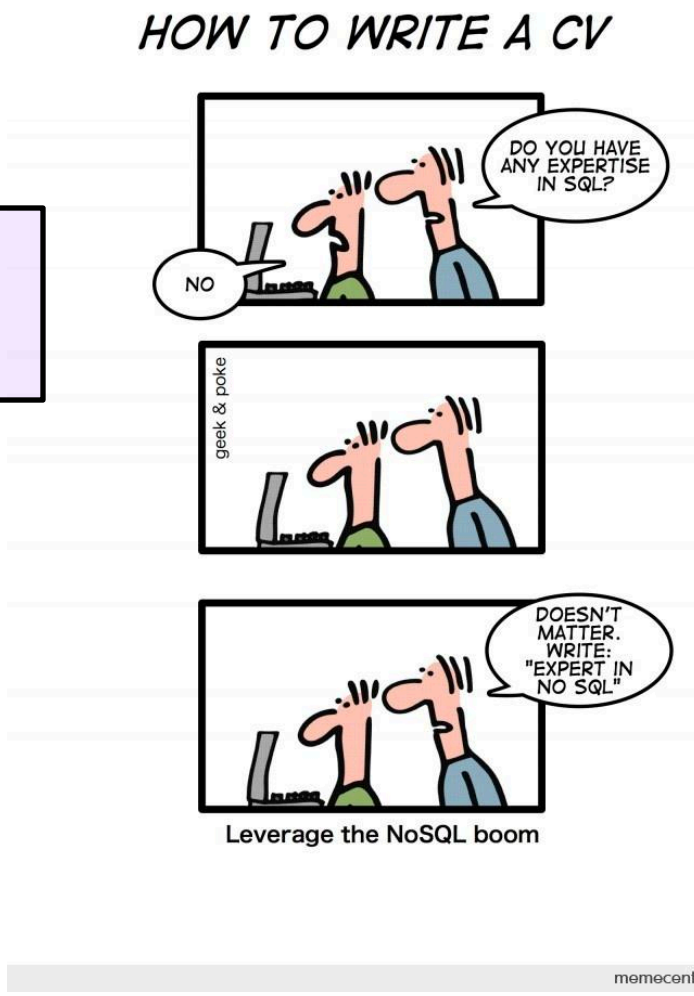
https://cloud.google.com/bigtable/docs/overview

Graph Databases



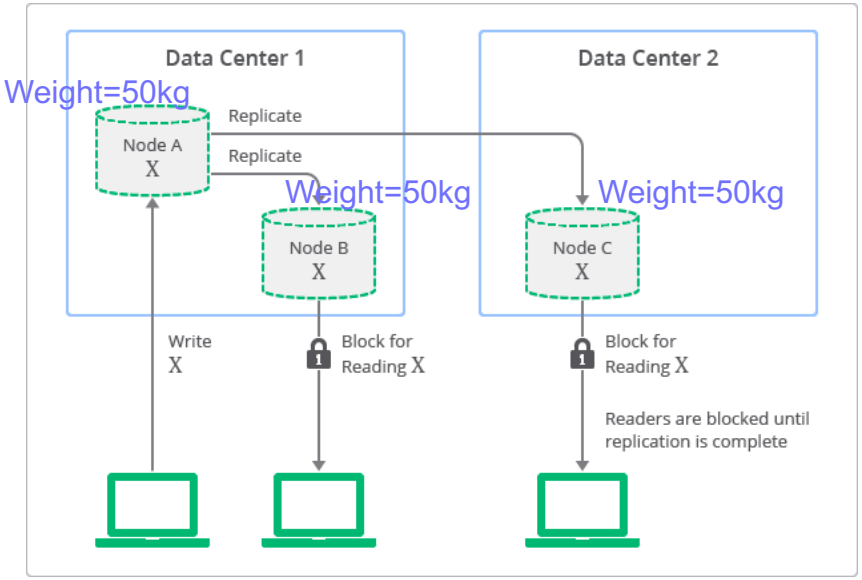
Today's Plan

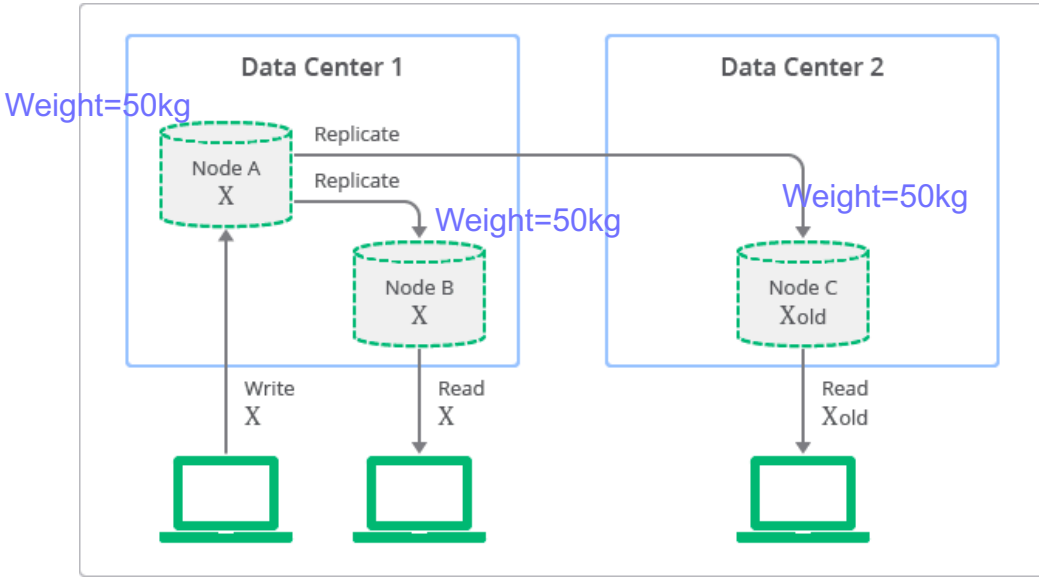
1. What is NoSQL?
2. Major types of NoSQL systems
3. **Key concepts**
   - **Eventual Consistency**
   - Duplication



Source: Cattell (2010). Scalable SQL and NoSQL Data Stores. *SIGMOD Record*.

Strong vs Eventual Consistency

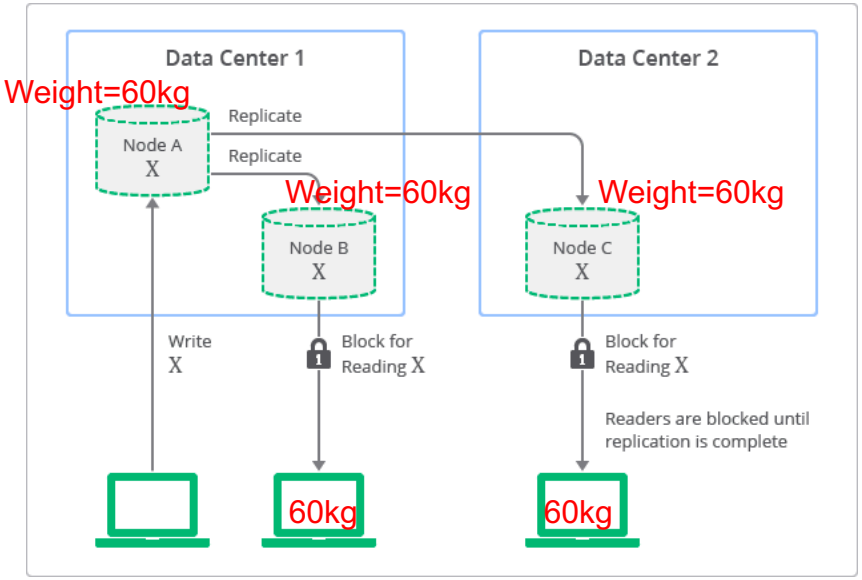


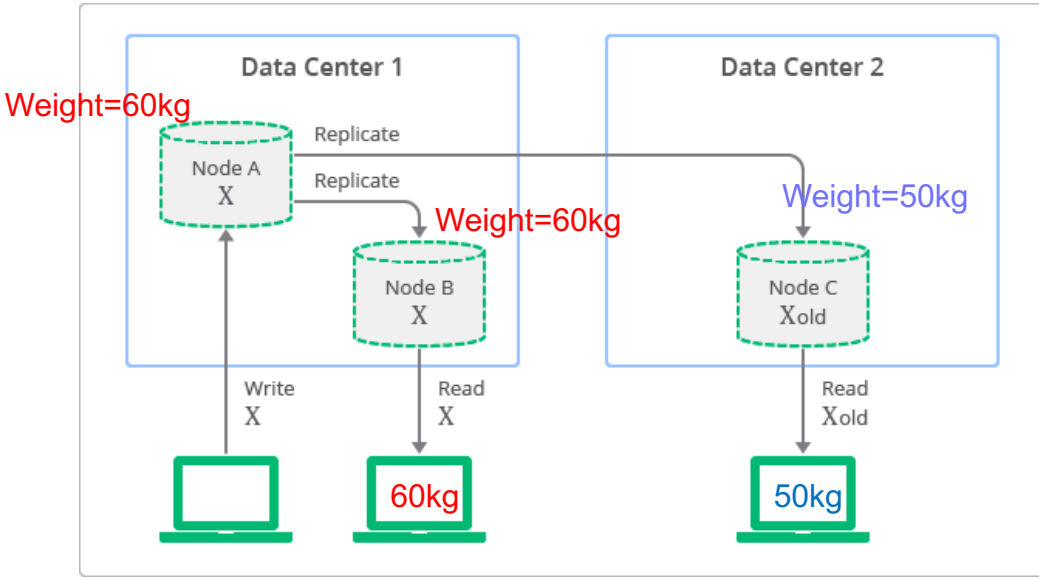


Strong consistency: any reads immediately after an update must give the same result on all observers

Eventual consistency: if the system is functioning and we wait long enough, eventually all reads will return the last written value

Strong vs Eventual Consistency

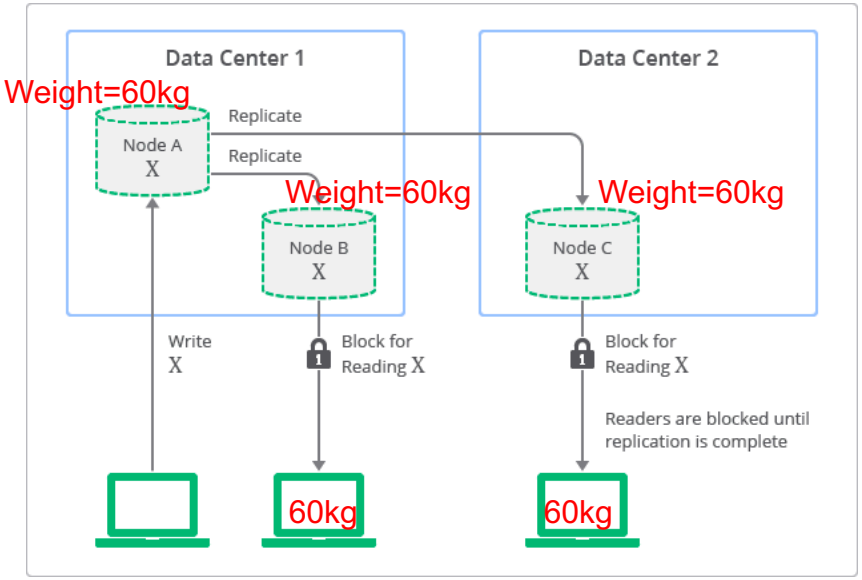


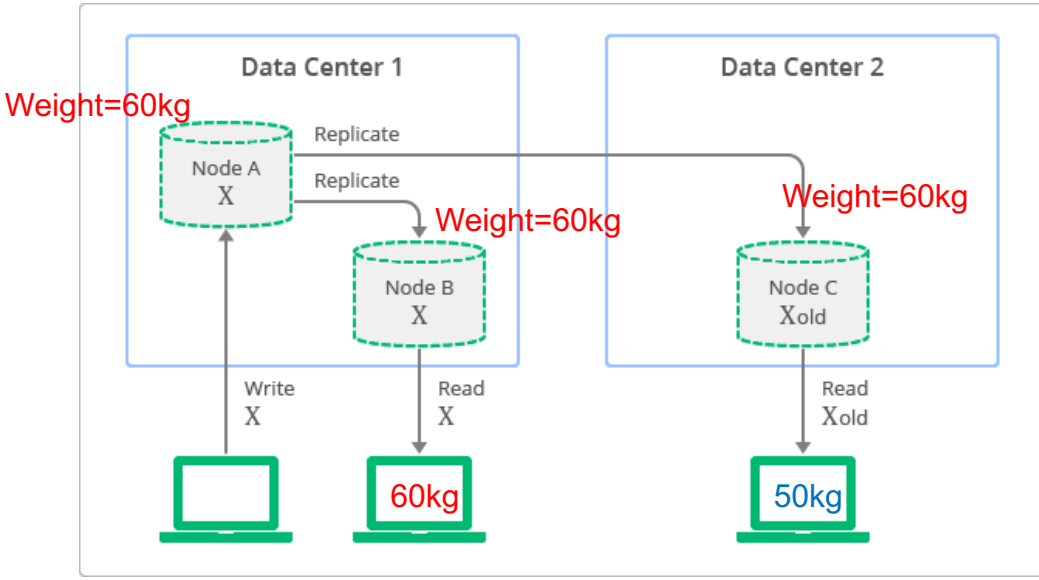


Strong consistency: any reads immediately after an update must give the same result on all observers

Eventual consistency: if the system is functioning and we wait long enough, eventually all reads will return the last written value

Strong vs Eventual Consistency





Strong consistency: any reads immediately after an update must give the same result on all observers

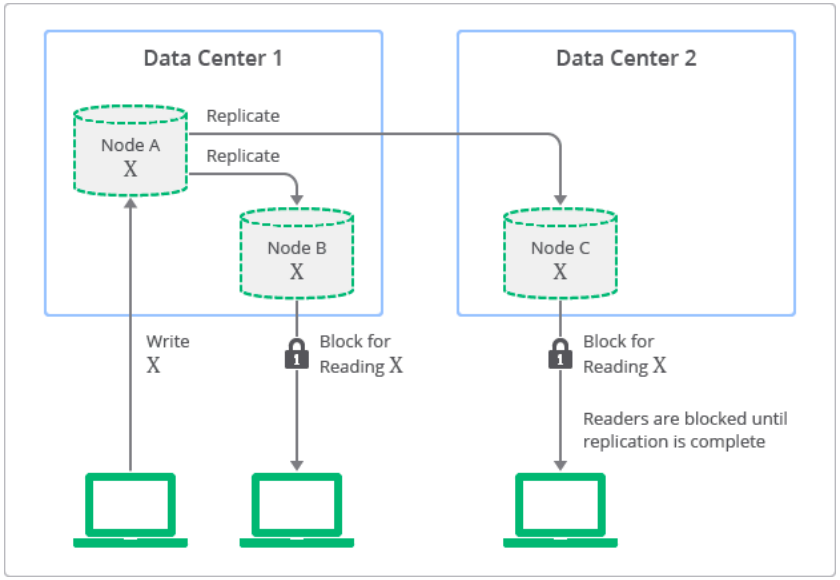
Eventual consistency: if the system is functioning and we wait long enough, eventually all reads will return the last written value

Eventual Consistency

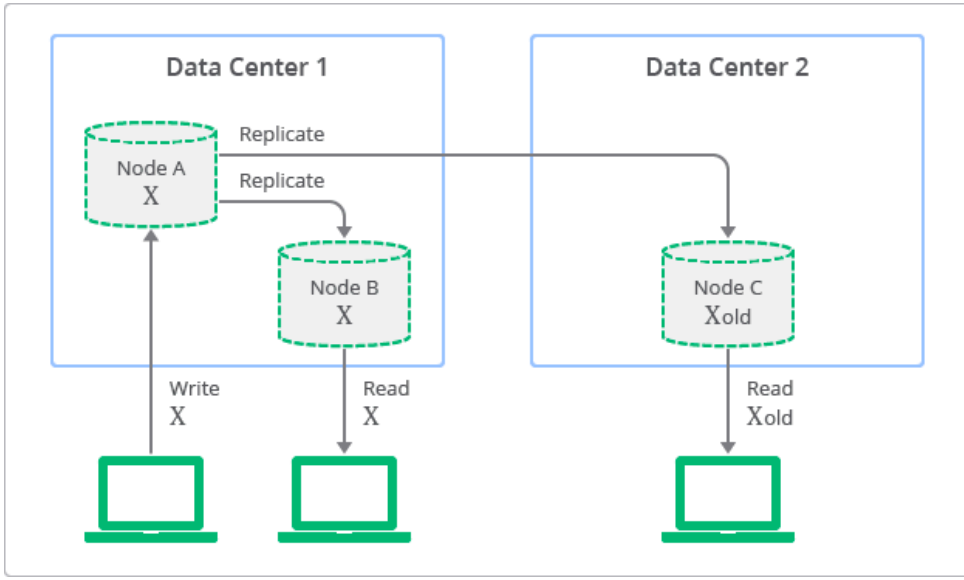
- **ACID vs BASE:** Relational DBMS provide stronger (ACID) guarantees, but many NoSQL system relax this to weaker "BASE" approach:
  - **Basically Available**: basic reading and writing operations are available most of the time
  - **Soft State**: without guarantees, we only have some probability of knowing the state at any time
  - **Eventually Consistent**: Contrast to **"strong consistency"**:

Strong vs Eventual Consistency

Strong consistency



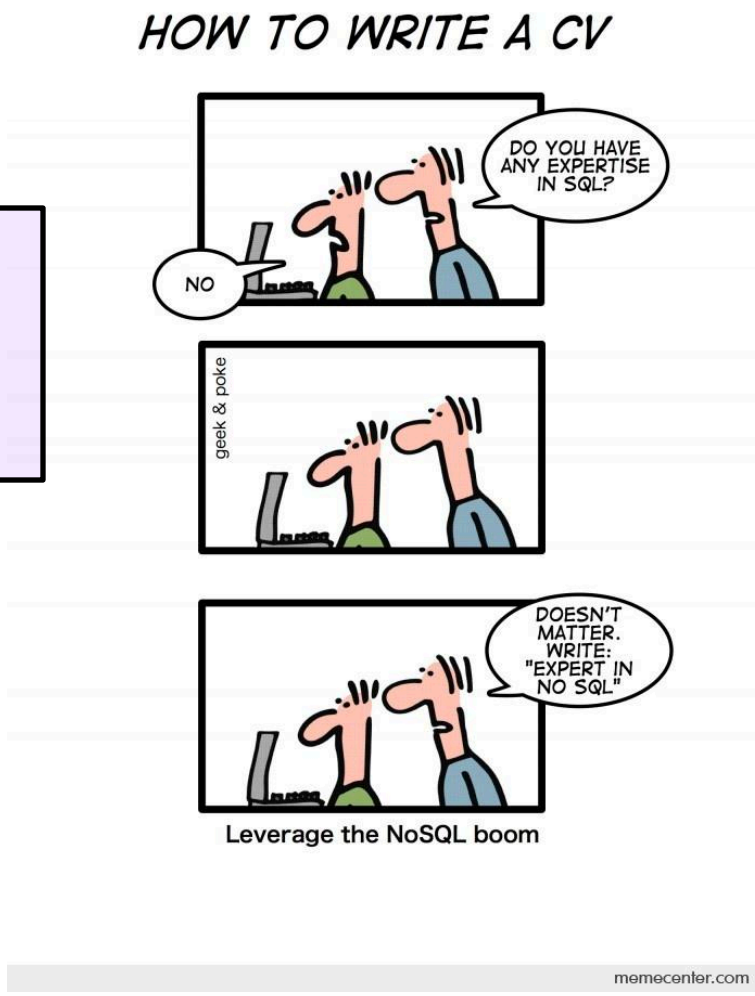
Eventual consistency



- **Implications**: eventual consistency offers better **availability** at the cost of a much weaker consistency guarantee. This may be acceptable for some applications (e.g. statistical queries, tweets, social network feed, …) but not for others (e.g. financial transactions)
  - Note that while NoSQL systems allow for weaker consistency guarantees, many more recent systems / versions are often configurable, i.e. can be configured for multiple different consistency levels (including strong) – *'tunable consistency'*

Today's Plan

1. What is NoSQL?
2. Major types of NoSQL systems
3. **Key concepts**
   - Eventual Consistency
   - **Duplication**



Source: Cattell (2010). Scalable SQL and NoSQL Data Stores. *SIGMOD Record*.

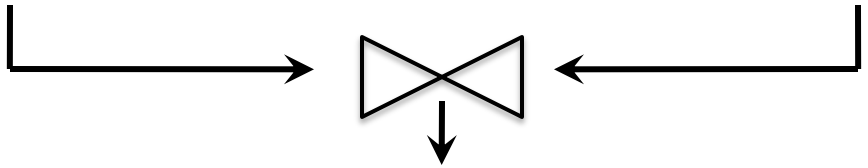
Recall: Joins in RDBMS

Users

user_id	name
100	Alice
101	Bob

Posts

post_id	post	user_id
1	Hello world	100
2	Bye world	101



post_id	post	user_id	name
1	Hello world	100	Alice
2	Bye world	101	Bob

What if we want to display posts with usernames?

Users

```
[
  {
    "user_id": 100,
    "name": "Alice"
  },
  {
    "user_id": 101,
    "name": "Bob"
  },
]
```

Posts

```
[
  {
    "post_id": "1",
    "post": "Hello world",
    "user_id": 100
  },
  {
    "post_id": "2",
    "post": "Bye world",
    "user_id": 101
  },
]
```

Posts (+ username)

```
[
  {
    "post_id": "1",
    "post": "Hello world",
    "user_id": 100,
    "name": "Alice",
  },
  {
    "post_id": "2",
    "post": "Bye world",
    "user_id": 101,
    "name: "Bob"
  },
]
```

- Answer 1: some NoSQL databases do support joins (e.g. later versions of MongoDB)
- Answer 2: Duplication (i.e. 'denormalization')
  - 'Storage is cheap: why not just duplicate data to improve efficiency?'
  - Tables are designed around the queries we expect to receive
  - Leads to a new problem: what if user changes their name? (this needs to be propagated to multiple tables)

Conclusion: Pros & Cons of NoSQL Systems

Pros	Cons
+ **Flexible / Dynamic Schema**: suitable for less well-structured data + **Massive Scalability / High Performance** + **High Availability** + **Relaxed consistency:** higher performance and availability	- **No declarative query language**: query logic (e.g. joins) may have to be handled on the application side, which can add additional programming - **Relaxed consistency**: fewer guarantees; application may receive stale data that may need to be handled on the application side

Conclusion: no one size fits all. Depends on needs of application: complexity of queries (joins vs simple read/writes), importance of consistency (e.g. financial transactions vs tweets), data volume / need for availability.

- Binary view is oversimplified: NoSQL databases often still have SQL-like query languages (e.g. Cassandra) and tunable consistency levels.
- Other alternative solutions: "NewSQL", distributed SQL (e.g. Presto), PostgreSQL JSON, …
- Need to understand tradeoffs to make an informed decision!

Source: Cattell (2010). Scalable SQL and NoSQL Data Stores. *SIGMOD Record*.

Acknowledgements

- CS4225 slides by He Bingsheng
- Penn State CMPSC 431W Database Management Systems (Fall 2015) – McGrawHill, Wang-Chien Lee, Yu-San Lin
- mongodb.com, redis.com
- Otter video from https://www.reddit.com/r/singapore/comments/mezgg3/otters_on_the_grass_at_botanic_gardens_today/