

ABHIJIT AGARWAL / DATA SCIENCE RETREAT

NOSQL

# TYPES OF DATABASES



C1	C2	C3	C4
—	—	—	—
—	—	—	—
—	—	—	—
—	—	—	—

## Relational data model

Highly-structured table organization with rigidly-defined data formats and record structure.



## Document data model

Collection of complex documents with arbitrary, nested data formats and varying "record" format.

# WHAT IS NOSQL?

- NoSQL now stands for “Not Only SQL”
- Relational databases fulfill most engineering use-cases, but sometimes more specific solutions are required
  - Fast Look Up based on a key
  - High horizontal scalability
  - Storing unstructured data
  - Faster Aggregations on Columns

# CAP Theorem

- You cannot have Consistency, Availability and Performance at the same time
- When the DB uses multiple nodes, something has to be sacrificed:
  - e.g. data just inserted may not be available to some nodes
  - e.g. data is partitioned and it's harder to perform JOINS

# FAST KEY LOOKUP

- Key-Value Store
- In-Memory, like Redis
- Distributed, like BigTable / HBase
- Document stores: Like MongoDB



FAST KEY LOOKUP

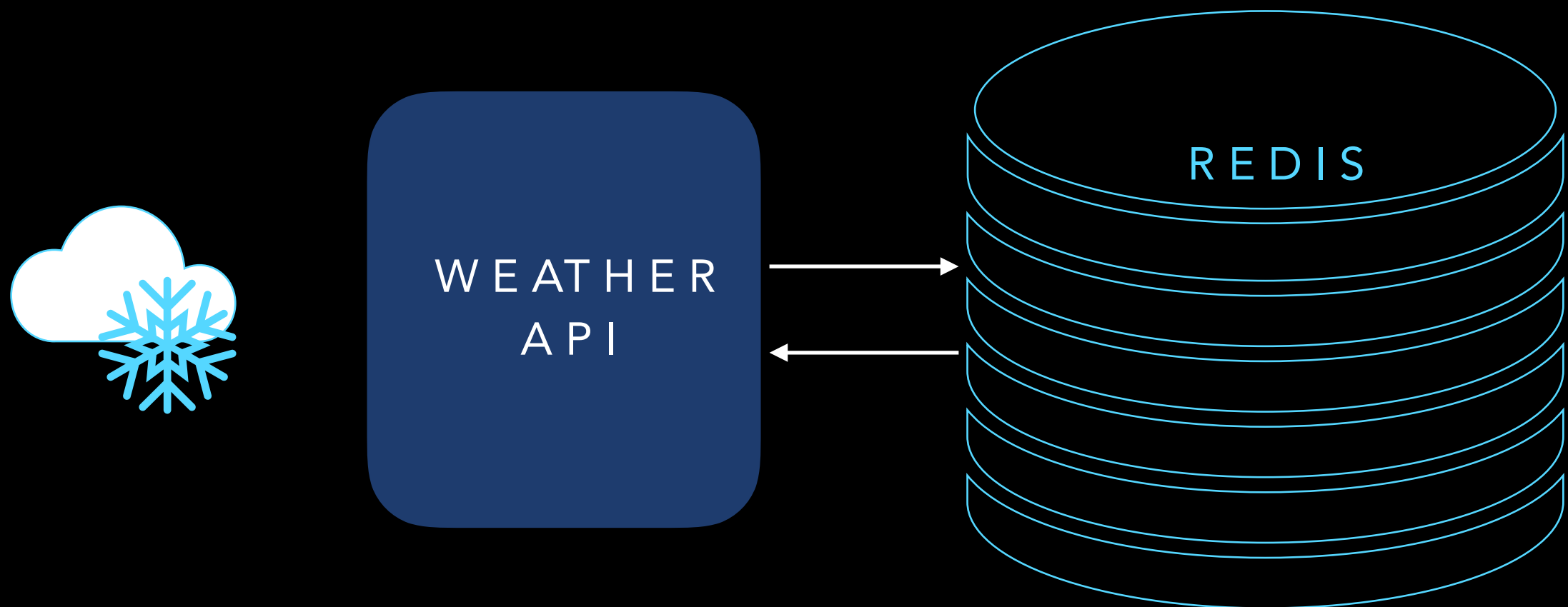
# IN-MEMORY KV STORE: RED

Key	Value
Delhi	{ 'TEMP': 291.15, 'PRESSURE': 1017, 'HUMIDITY': 75, 'TEMP_MIN': 291.15, 'TEMP_MAX': 291.15 }
Berlin	{ 'TEMP': 294.84, 'PRESSURE': 1016, 'HUMIDITY': 75, 'TEMP_MIN': 294.15, 'TEMP_MAX': 295.15 }
Nyc	{ 'TEMP': 294.84, 'PRESSURE': 1016, 'HUMIDITY': 75, 'TEMP_MIN': 294.15, 'TEMP_MAX': 295.15 }

FAST KEY LOOKUP

# IN-MEMORY KV STORE: REDIS

- Unstructured data
- Query only by Key, nothing else
- Single instance
- In-Memory



FAST KEY LOOKUP

DISTRIBUTED KV STORE:

BIGTABLE / HBASE / CASSANDRA

- Unstructured data
- Query only by Key, nothing else
- Multiple instances
- On-Disk



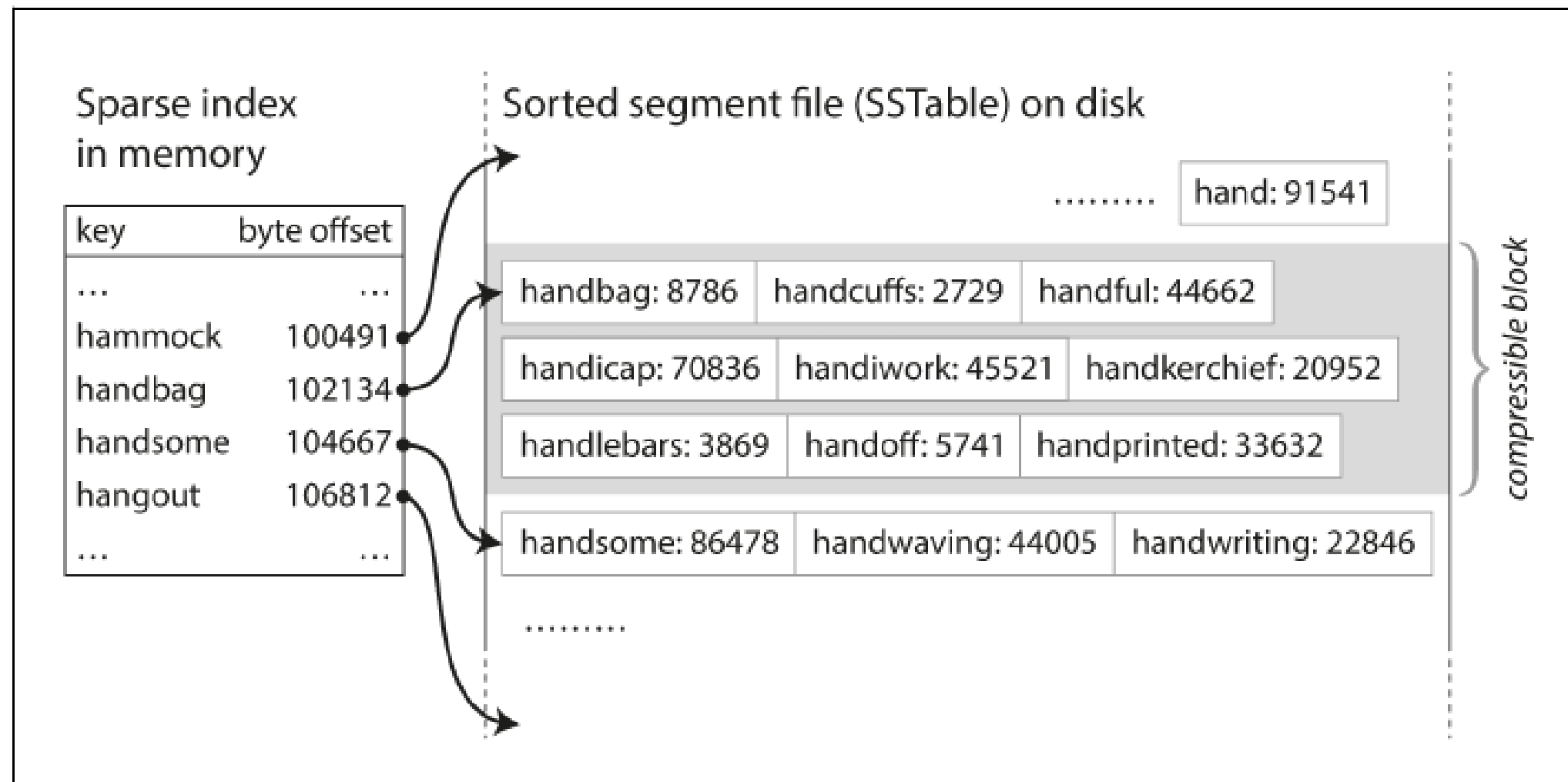


# WHAT IS PARTITIONING?

- Split by Hash of Key

OR

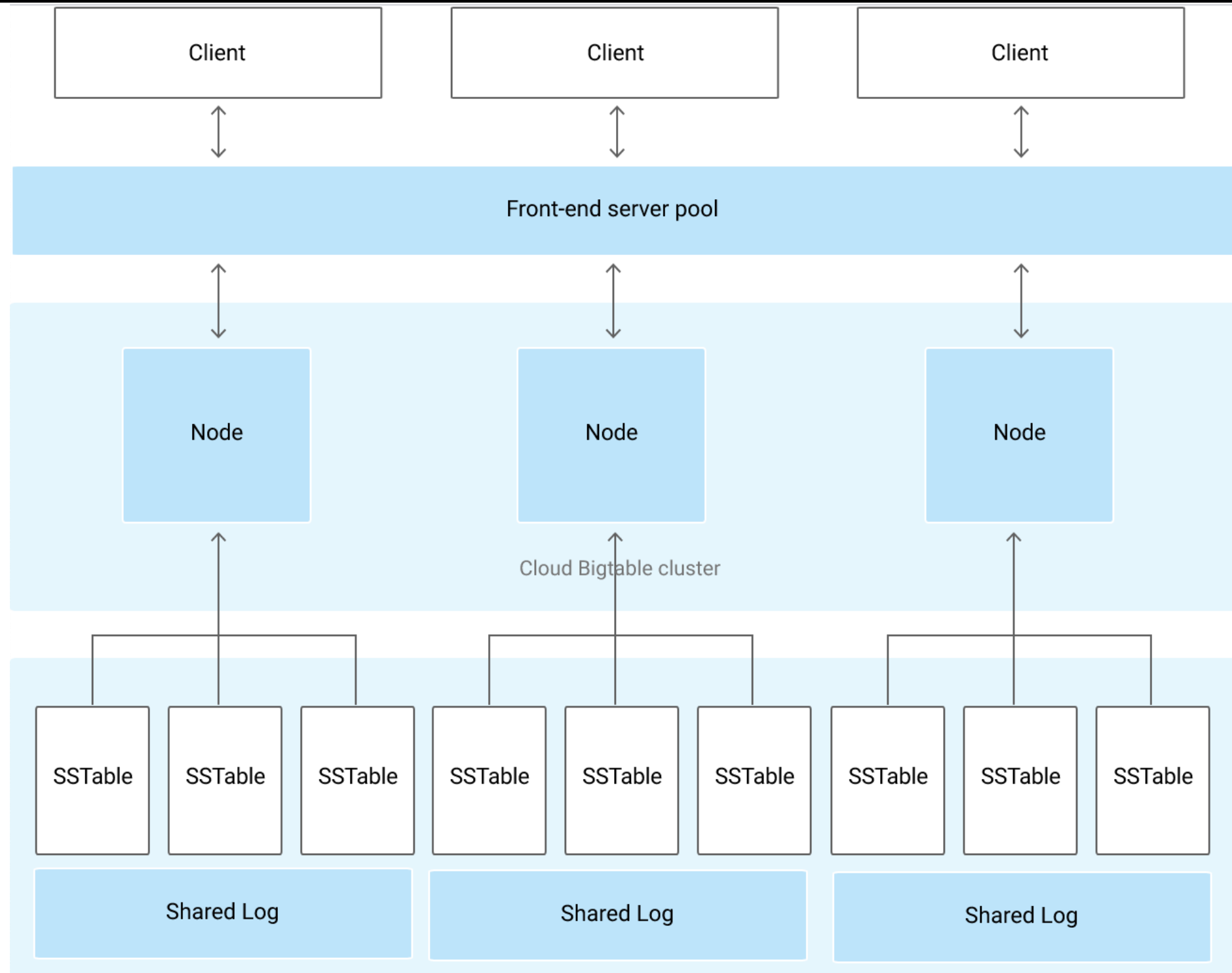
- SSTable: Stored in sorted segments



FAST KEY LOOKUP

DISTRIBUTED KV STORE:

BIGTABLE / HBASE / CASSANDRA



FAST KEY LOOKUP

# DOCUMENT STORE

- Special case of Key Value Stores
- Flexible “Schema”, Nested values

```
ID: breakfast
{
  "type": "toast",
  "bread": "whole wheat",
  "spread": [
    "butter",
    "jam"
  ]
}
```

```
ID: lunch
{
  "type": "salad",
  "vegetarian": false,
  "ingredients": [
    "spinach",
    "tomato",
    "cucumber",
    "carrot",
    "dressing": [
      "olive oil",
      "vinegar",
      "honey",
      "lemon",
      "salt",
      "pepper",
    ],
    "tuna",
    "walnuts"
  ],
  "rating": "5 stars",
  "restaurant": "Skylight Diner"
}
```

```
ID: dinner
{
  "type": "pizza",
  "size": "large",
  "toppings": [
    "pepperoni",
    "tomato",
    "sausage"
  ],
  "price": 9.00,
  "presliced": true
}
```

# DOWNSIDES OF NOSQL

- You are responsible for managing your schema
- Constraints are either enforced on primary keys, or not at all. No unique columns, no foreign keys.
- Query optimized for specific use-cases, not generic

# COLUMN ORIENTED DATABASES

- Relational Databases are used for transactional purposes.  
For example: Recording each sale that happens in an H&M Store.
- Column databases are used for analytical purposes.  
For example: Getting a report on revenue of all H&Ms across the world
- Examples: Amazon RedShift, Google BigQuery, InfoBright

# COLUMN ORIENTED DATABASES

Logical table representation

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Row layout

a1	b1	c1	a2	b2	c2	a3	b3	c3	a4	b4	c4	a5	b5	c5
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Column layout

a1	a2	a3	a4	a5	b1	b2	b3	b4	b5	c1	c2	c3	c4	c5
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

# RELATIONAL VS COLUMN-ORIENTED

- Writing data:
  - In row-oriented database, data is added row by row and appended to a file
  - In column-oriented database, first data needs to be split by columns, and then it is added to the file for each column.  
Writing a single row is much more expensive.

# RELATIONAL VS COLUMN-ORIENTED

- Reading data:
  - In row-oriented database, first all data for a row is loaded from disk, and then required columns are selected from that row
  - In column-oriented database, first all data for required columns is loaded. And then some rows are filtered out, if needed.  
Reading only a few columns and doing aggregations is much faster.



# EXERCISE

- Choose one of the following Database Types:
  - Relational Database (MySQL / Postgres)
  - In-Memory KV (Redis)
  - Distributed KV (BigTable / HBase / Cassandra)
  - Column Oriented Databases (RedShift / BigQuery / InfoBright)

# EXERCISE - CONTINUED

- Choose an appropriate database type for each application:
  - An application that needs to store the most recent products viewed by each user on Amazon.de
  - An application to manage a blog with blog posts, tags, comments, etc.
  - An application to create reports on global statistics of a dating app: top 5 countries, most used feature, busiest hour of the week