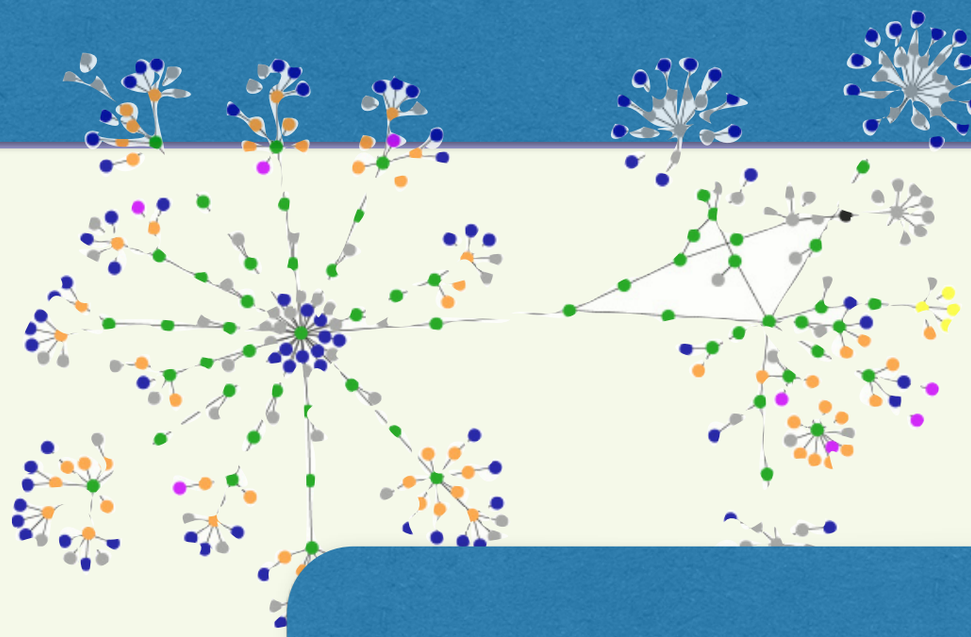
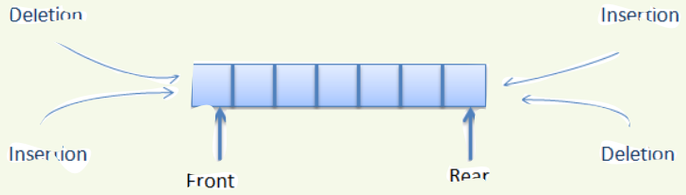
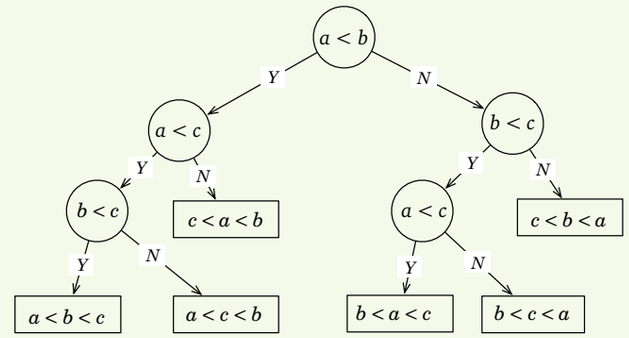




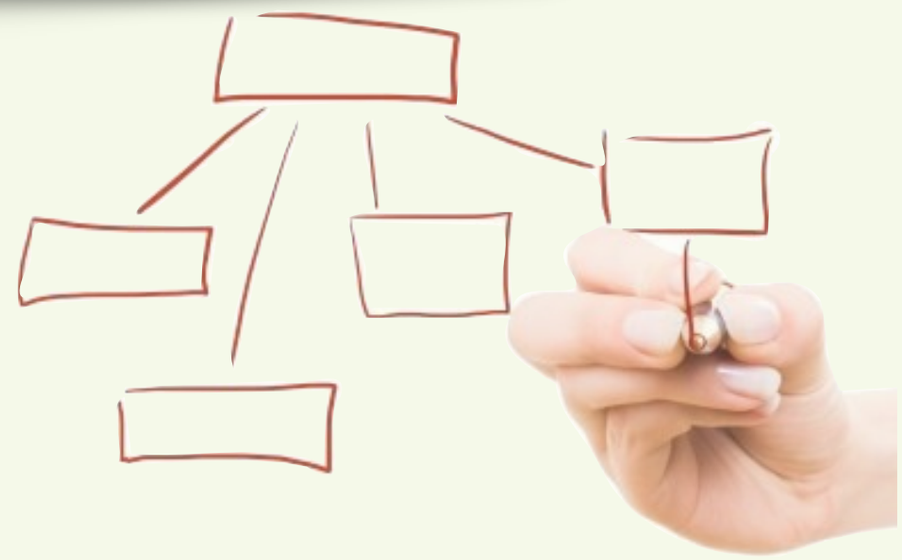

# COURSE
# "Técnicas Matemáticas para Big Data"
## University of Aveiro

**Algorithm 2.1:** INSERTION-SORT($A$)

```
1  for j ← 2 to A.size do
2      key ← A[j]
       // Insert A[j] into the sorted sequence A[1 . . j − 1]
3      i ← j − 1
4      while i > 0 and A[i] > key do
5          A[i + 1] ← A[i]
6          i−−
7      A[i + 1] ← key
```

**Master in Data Science**

**Other Masters (optional)**



**EXTRA INFO** - this label will appear in slides with complementary information

**SUMMARY:** Streaming K-means or Canopy Clustering. Frequent items in a data stream:

[T] Formal methods and concepts for streaming; Quality of an algorithm's answer; Averages and standard deviations over a stream; Quantile outliers and z-scores outliers; Finding frequent items deterministically; Frequency estimation by the Misra–Gries algorithm.

[P] Implementation of naive mean, std, and z-score outlier algorithms; Test and implementation of the MG-algorithm; Probabilistic top-10 elements

**EXTRA INFO** - this label will appear in slides with complementary information

# Streaming Algorithm
(bins+probabilistic)

Stream: $\sigma = \langle a_1, a_2, ..., a_m \rangle$

PROBLEM: How can we estimate the frequency of items, and even better, **find all elements that occur more than *m/k* times in the stream, for a defined *k* ?**

Stream: $\sigma = \langle a_1, a_2, ..., a_m \rangle$

PROBLEM: How can we estimate the frequency of items, and even better, **find all elements that occur more than *m/k* times in the stream, for a defined *k* ?**

## 1.2 Finding Frequent Items Deterministically

### 1.2.1 The problem

For a given stream $\sigma = \langle a_1, \ldots, a_m \rangle$, with each $a_i \in [n]$, we have a frequency vector $f = (f_1, \ldots, f_n)$, with $f_1 + \cdots + f_n = m$.

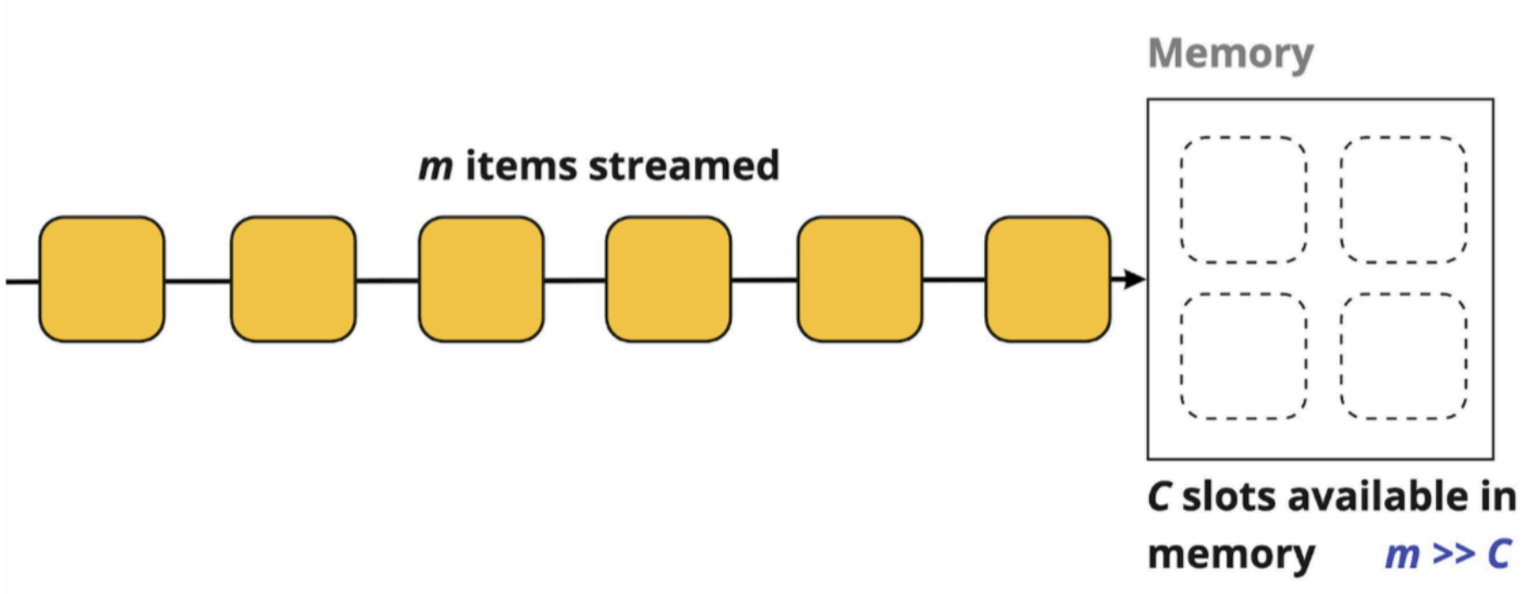
- **MAJORITY problem**: If $\exists j \in [n] : f_j > \frac{m}{2}$, then output $\{j\}$, otherwise, output the empty set $\{\}$.
- **FREQUENT problem**: Given $k \in \mathbb{N}$, output the set $\{j \in [n] : f_j > \frac{m}{k}\}$, where the set may be empty.

Hence, a MAJORITY problem is a FREQUENT problem with $k = 2$.

Stream: $\sigma = \langle a_1, a_2, ..., a_m \rangle$

PROBLEM: How can we estimate the frequency of items, and even better, **find all elements that occur more than *m/k* times in the stream, for a defined *k* ?**

**A:** With the **Misra-Gries** algorithm




A **one-pass** algorithm counting all elements would require ***Ω(m)*** space in the computer, which is not what we want. But by estimating the frequency of an element with the Misra-Gries algorithm, the space usage can be bounded by **O(*k*( log(m)+log(n))*)**. The estimation done by this algorithm allows for a one-sided error. This means that even if all the most frequent items are in the output at the end, there could also be some **infrequent elements**.

THEOREM. When the dictionary $A$ is implemented as a balanced binary search tree, the Misra-Gries algorithm with a given key/value bound of $k \in \mathbb{N}$ uses only one pass, $O(k \log(m) + k \log(n))$ bits of space, and provides a frequency approximation estimate $\hat{f}_j \equiv A[j]$, satisfying $f_j - \frac{m}{k} \leq \hat{f}_j \leq f_j$.

QUESTION: What $k$ accounts for?

THEOREM. When the dictionary $A$ is implemented as a balanced binary search tree, the Misra-Gries algorithm with a given key/value bound of $k \in \mathbb{N}$ uses only one pass, $O(k \log(m) + k \log(n))$ bits of space, and provides a frequency approximation estimate $\hat{f}_j \equiv A[j]$, satisfying $f_j - \frac{m}{k} \leq \hat{f}_j \leq f_j$.

QUESTION: What $k$ accounts for?

## Algorithm space bounds

There is at most ***k-1*** counters in ***D*** (that we can simplify to ***k***). For each counter, we hold a key that can be from ***1*** to ***n***, and a corresponding value that can be from ***1*** to ***m***.

Storing a key ***n*** require ***log(n)*** space (think of binary representations), and a counter ***m*** requires ***log(m)*** space. So one key-value pair represent ***log(n)+log(m)*** space.

Since we have k-1 keys, we end up with a higher bound **O(*k*( log(m)+log(n))*)** for the algorithm space usage.

## Misra-Gries

### The algorithm

The algorithm itself is quite simple. Suppose that we have a stream ***S*** with ***m*** elements coming from it. We use a dictionary ***D*** that will hold counters for the streamed items.

**Initialize a dictionary *D***

**For each item *a* coming from the stream *S*:**

- **If a is in *D*, then *D[a]++***
- **Else if the size of *D* is less than *k-1*, insert *a* in *D* with value 1**
- **Else decrement all counters in *D* by *1*, and remove all elements with count *0***

Implement your version and try it with: [1,1,2,3,4,5,1,1,1,5,3,3,1,1,2] with ***k=3***

## TOP-5 streaming algorithm

PROBLEM: Implement (an approximated) TOP-5 algorithm.

## TOP-5 streaming algorithm

PROBLEM: Implement (an approximated) TOP-5 algorithm.

### 1.1.3 Quality of an algorithm's answer

Since there are problems that cannot be solved in just one pass or be computed exactly using sublinear space, we typically seek to compute only an estimate or approximation of the value of $\Phi(\sigma) \in \mathbb{R}$.

DEFINITION. Let $A(\sigma)$ denote the output of a randomized streaming algorithm $A$ on the stream input $\sigma$ (which is a random variable), and $\phi$ be the function that $A$ is supposed to compute/approximate. We say that

- (**multiplicative form**): the algorithm $(\varepsilon, \delta)$-estimates $\phi$ if $P\left[\left|\frac{A(\sigma)}{\phi(\sigma)} - 1\right| > \epsilon\right] \leq \delta$;
- (**additive form**): the algorithm $(\varepsilon, \delta)^{+}$-estimates $\phi$ if $P\left[\left|A(\sigma) - \phi(\sigma)\right| > \epsilon\right] \leq \delta$.

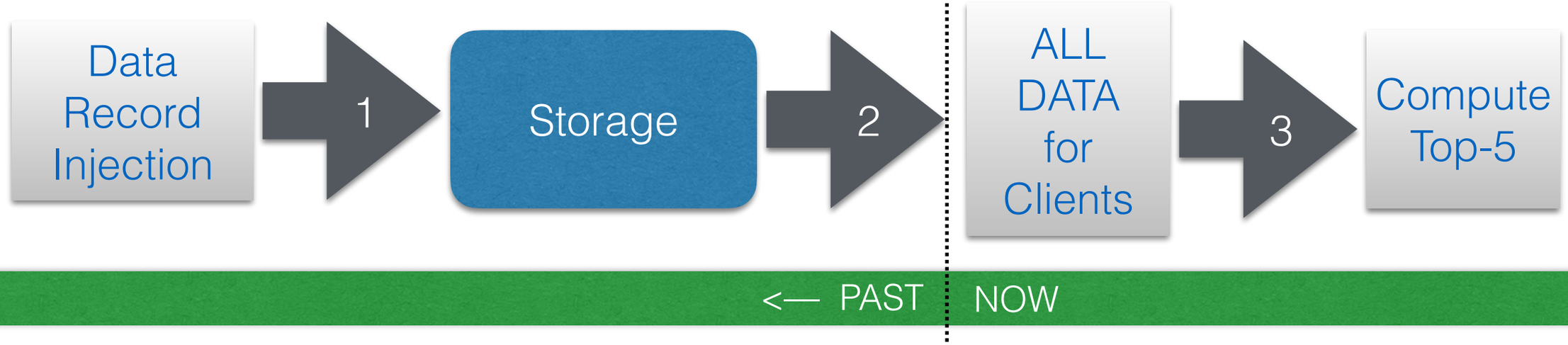
QUESTION: Why do we need two definitions?

ANSWER: ...

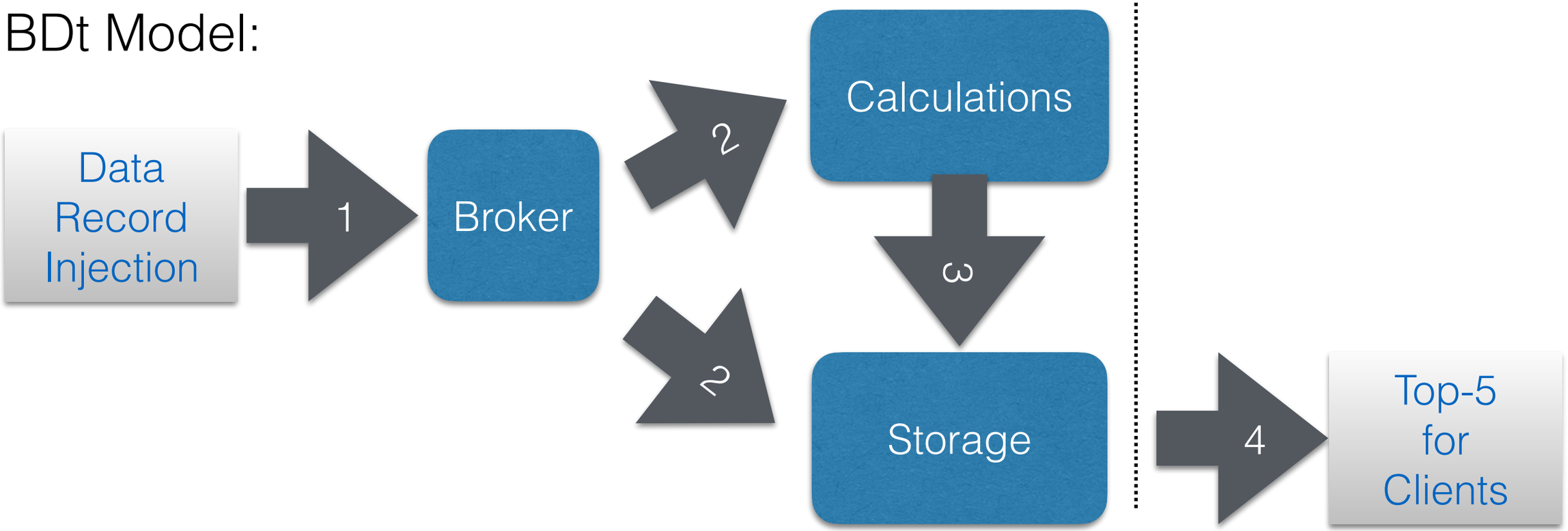
## How can it be used in a (big data) Database Engine?

**How can Streaming Algorithms be used in (big data) Database Engines?**

Classical Model:




BDt Model:



# Data pipeline in Big Data

## EXTRACT


## TRANSFORM


## Technical

### Data Store

mongoDB.

MySQL

Joins are what RDBMS's do for a living

### Transform

Spark

SQL

If you change the way you look at things, the things you look at change.

### Model


pandas

Spark

AI usually beats natural stupidity

### Visualize


Dont trust everything you see, even salt looks like sugar

### Other Tools



I have enough tools, said no data scientist ever

ORACLE®

MySQL®

Microsoft® SQL Server®

PostgreSQL

| SQL | NOSQL |
| --- | --- |
| Relational Database management system | Distributed Database management system |
| Vertically Scalable | Horizontally Scalable |
| Fixed or predifined Schema | Dynamic Schema |
| Not suitable for hierarchical data storage | Best suitable for hierarchical data storage |
| Can be used for complex queries | Not good for complex queries |

mongoDB

elastic

influxdb

neo4j

CouchDB relax

cassandra

## Relational Data Model

**Pros**

- Easy to use and setup.
- Universal, compatible with many tools.
- Good at high-performance workloads.
- Good at structure data.

**Cons**

- Time consuming to understand and design the structure of the database.
- Can be difficult to scale.

## Document Data Model

**Pros**

- No investment to design model.
- Rapid development cycles.
- In general faster than SQL.
- Runs well on the cloud.

**Cons**

- Unsuited for interconnected data.
- Technology still maturing.
- Can have slower response time.

# SQL

**Students**

| ID# | Name | Phone | DOB |
|---|---|---|---|
| 500 | Matt | 555-4141 | 06/03/70 |
| 501 | Jenny | 867-5309 | 3/15/81 |
| 502 | Sean | 876-9123 | 10/31/82 |

| ID# | ClassID | Sem |
|---|---|---|
| 500 | 1001 | Fall02 |
| 501 | 1002 | Fall02 |
| 501 | 1002 | Spr03 |
| 502 | 1003 | S203 |

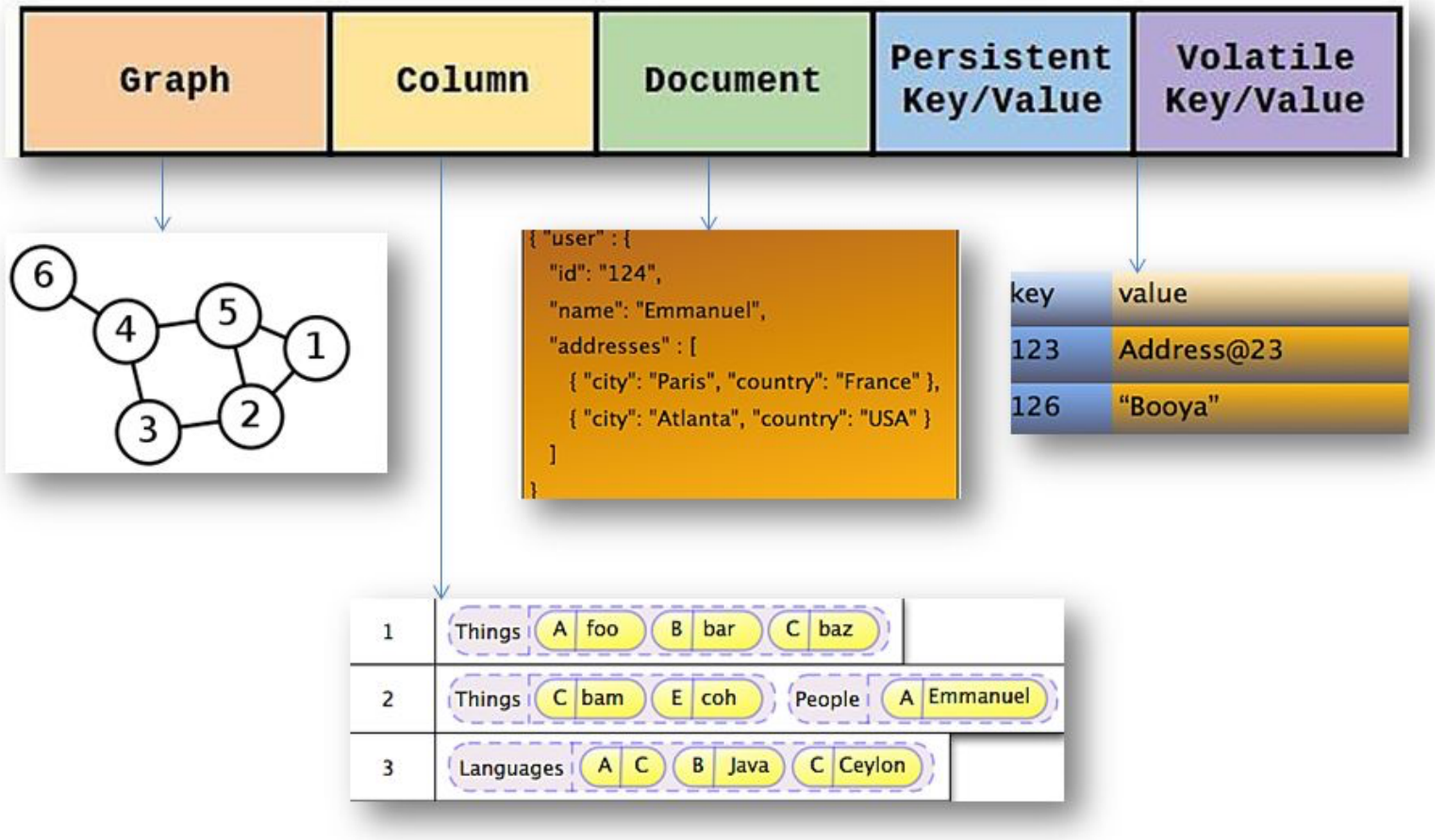
**Takes_Course**

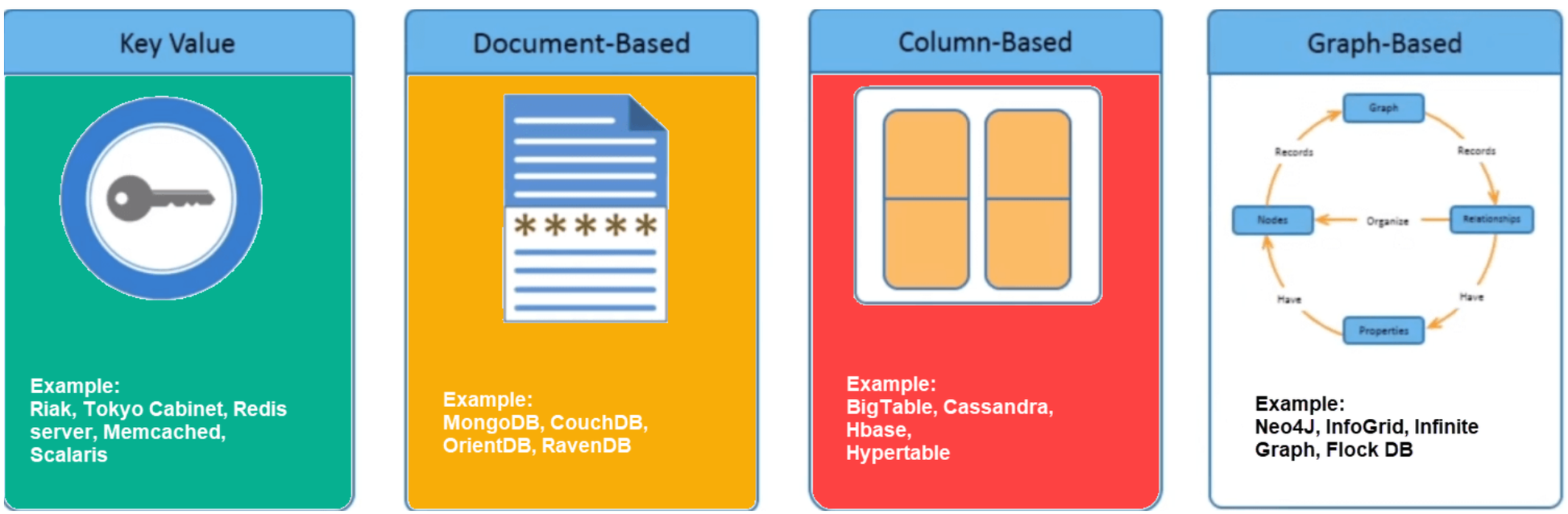
| ClassID | Title | ClassNum |
|---|---|---|
| 1001 | Intro to Informatics | I101 |
| 1002 | Data Mining | I400 |
| 1003 | Internet and Society | I400 |

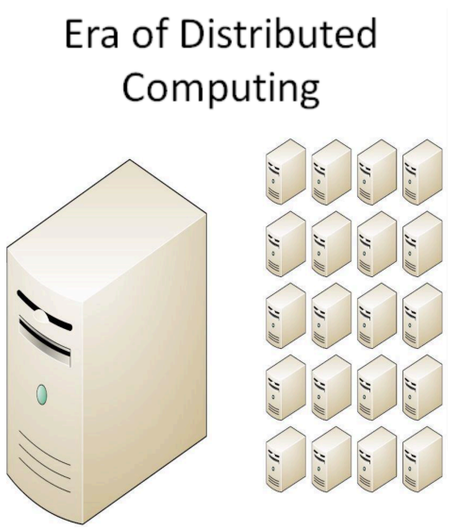
**Courses**

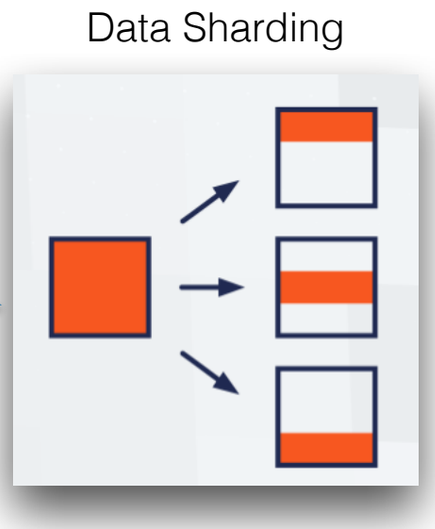
- Fixed scheme
- Joint oriented
- Normalized data

# Flavors of NoSQL






# STORAGE PARADIGMA

We need a distributed database system having such features:

- **Fault tolerance**
- **High availability**
- **Consistency**
- **Scalability**

## Database Engine Design

- ACID vs BASE
- Distribution and Scalability

**Table 1** Database categories and features.

| Data model | Performance | Scalability | Flexibility | Complexity | Functionality |
|---|---|---|---|---|---|
| Key – value store | High | High | High | Low | Variable (none) |
| Column store | High | High | Moderate | Low | Minimal |
| Document store | High | Variable (high) | High | Low | Variable (low) |
| Graph database | Variable | Variable | High | High | Graph theory |
| Relational database | Variable | Variable | Low | Moderate | Relational algebra |

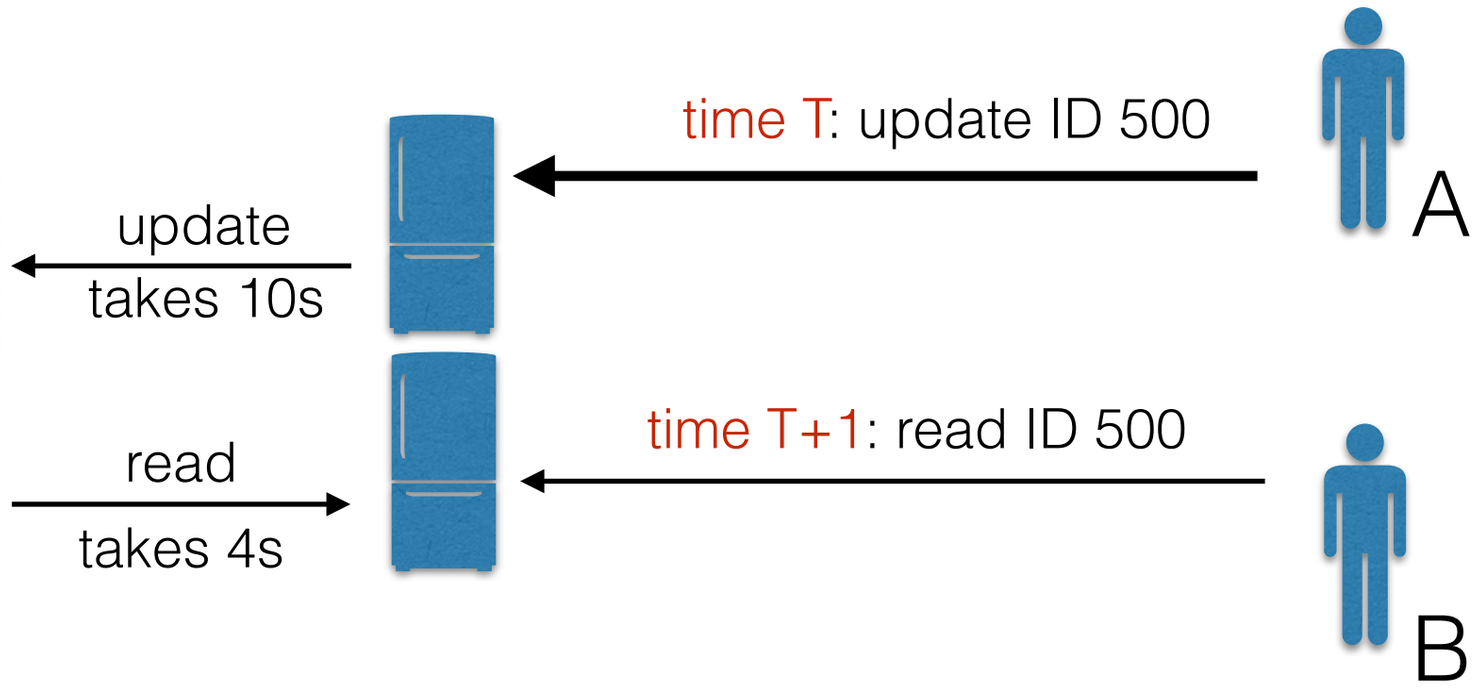
## ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

## INCONSISTENCY




**Students**

| ID# | Name | Phone | DOB |
|---|---|---|---|
| 500 | Matt | 555-4141 | 06/03/70 |
| 501 | Jenny | 867-5309 | 3/15/81 |
| 502 | Sean | 876-9123 | 10/31/82 |

**Takes_Course**

| ID# | ClassID | Sem |
|---|---|---|
| 500 | 1001 | Fall02 |
| 501 | 1002 | Fall02 |
| 501 | 1002 | Spr03 |
| 502 | 1003 | S203 |

**Courses**

| ClassID | Title | ClassNum |
|---|---|---|
| 1001 | Intro to Informatics | I101 |
| 1002 | Data Mining | I400 |
| 1003 | Internet and Society | I400 |

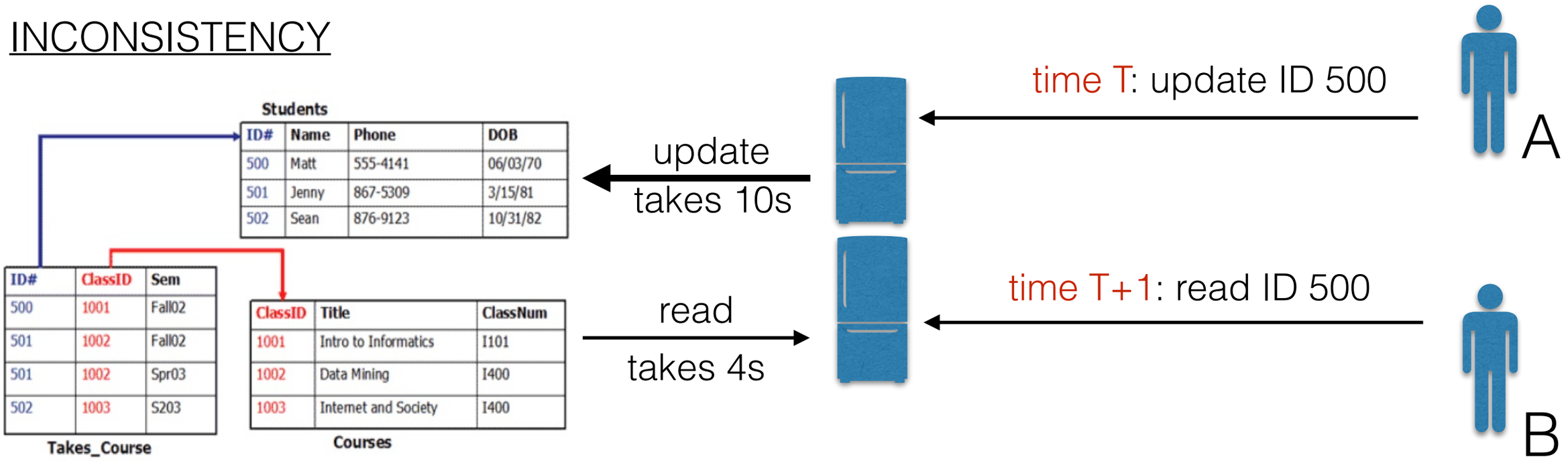
## ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

## INCONSISTENCY



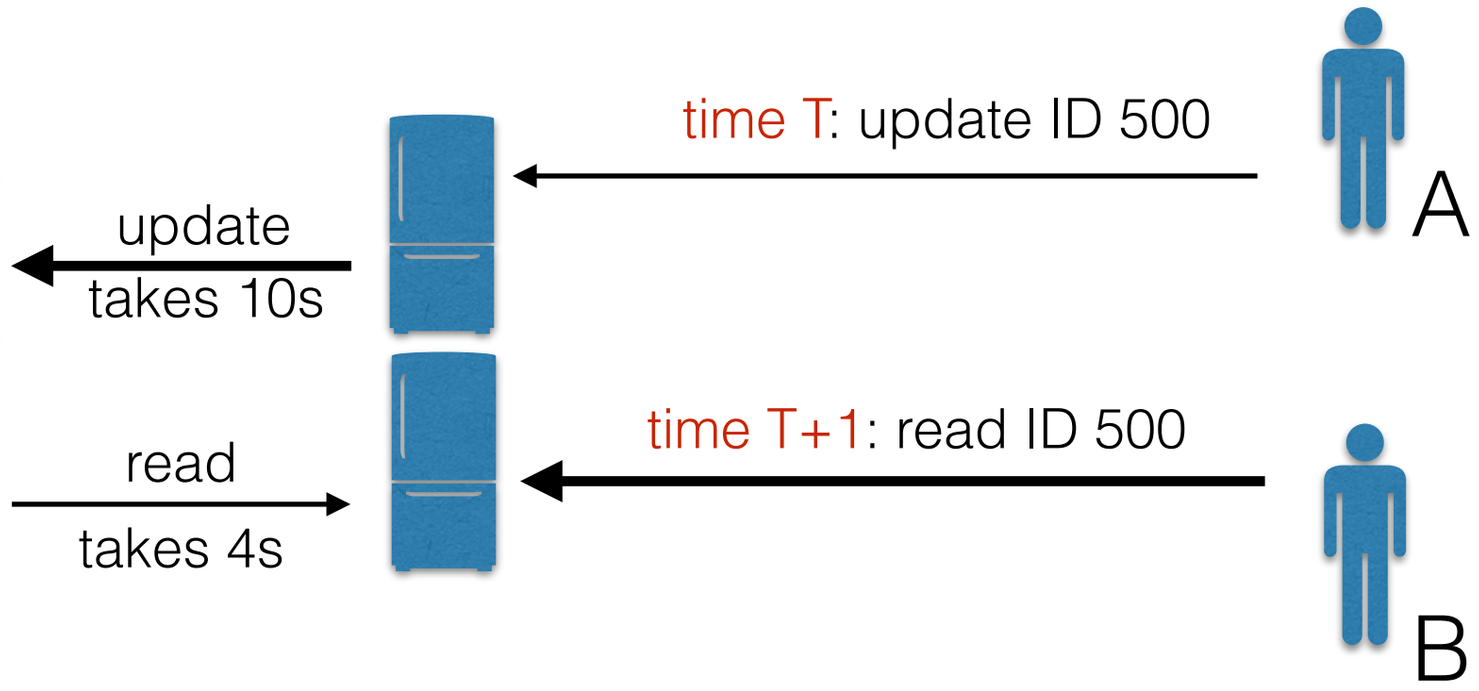
## ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

## INCONSISTENCY




**Students**

| ID# | Name | Phone | DOB |
|---|---|---|---|
| 500 | Matt | 555-4141 | 06/03/70 |
| 501 | Jenny | 867-5309 | 3/15/81 |
| 502 | Sean | 876-9123 | 10/31/82 |

**Takes_Course**

| ID# | ClassID | Sem |
|---|---|---|
| 500 | 1001 | Fall02 |
| 501 | 1002 | Fall02 |
| 501 | 1002 | Spr03 |
| 502 | 1003 | S203 |

**Courses**

| ClassID | Title | ClassNum |
|---|---|---|
| 1001 | Intro to Informatics | I101 |
| 1002 | Data Mining | I400 |
| 1003 | Internet and Society | I400 |

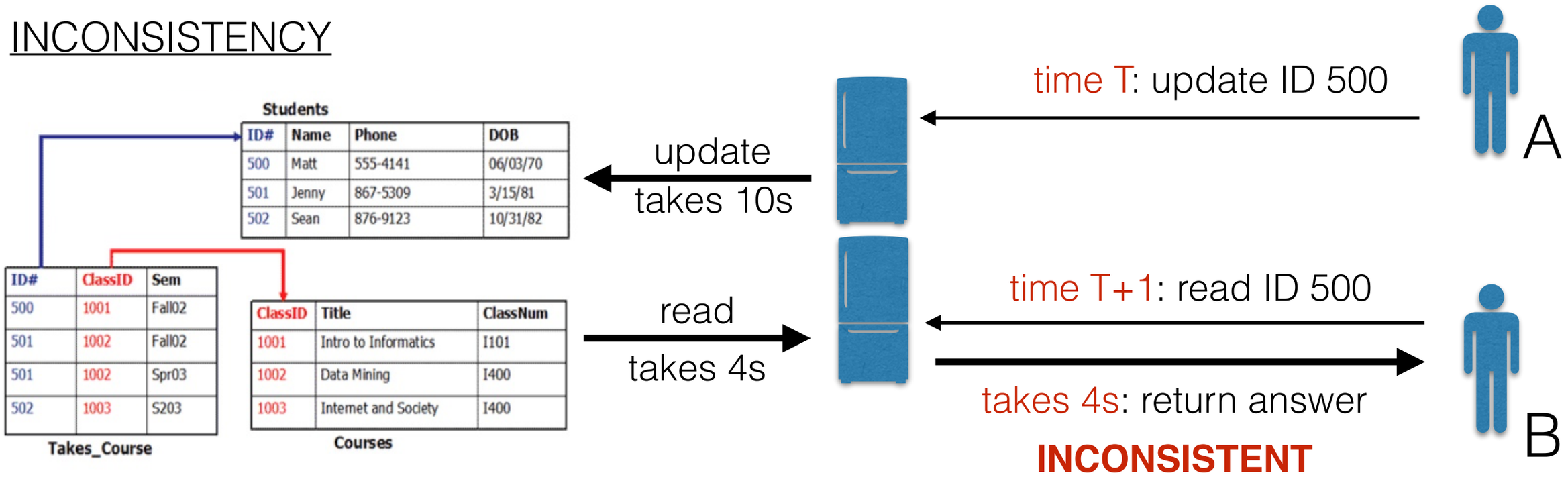
## ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

## INCONSISTENCY



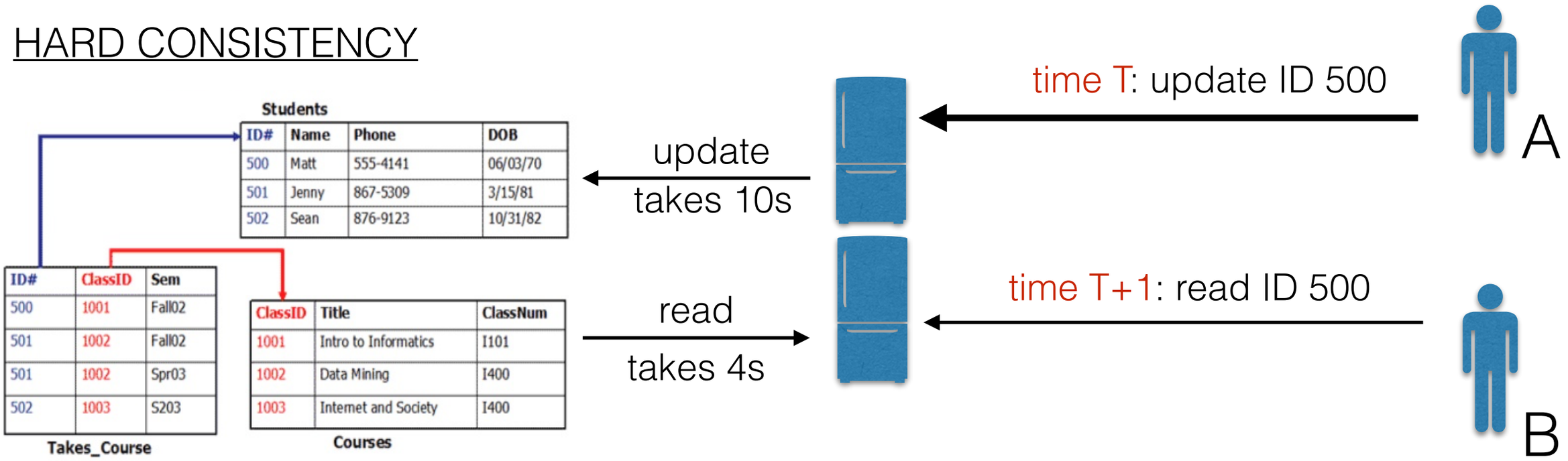
## ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

### HARD CONSISTENCY



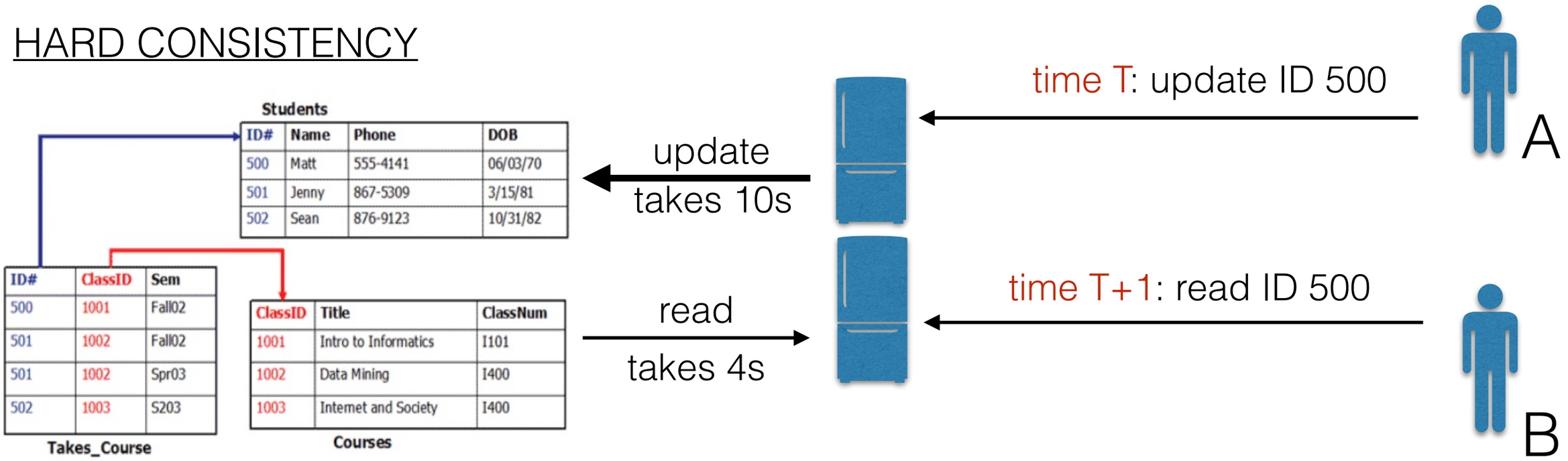
## ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

### HARD CONSISTENCY



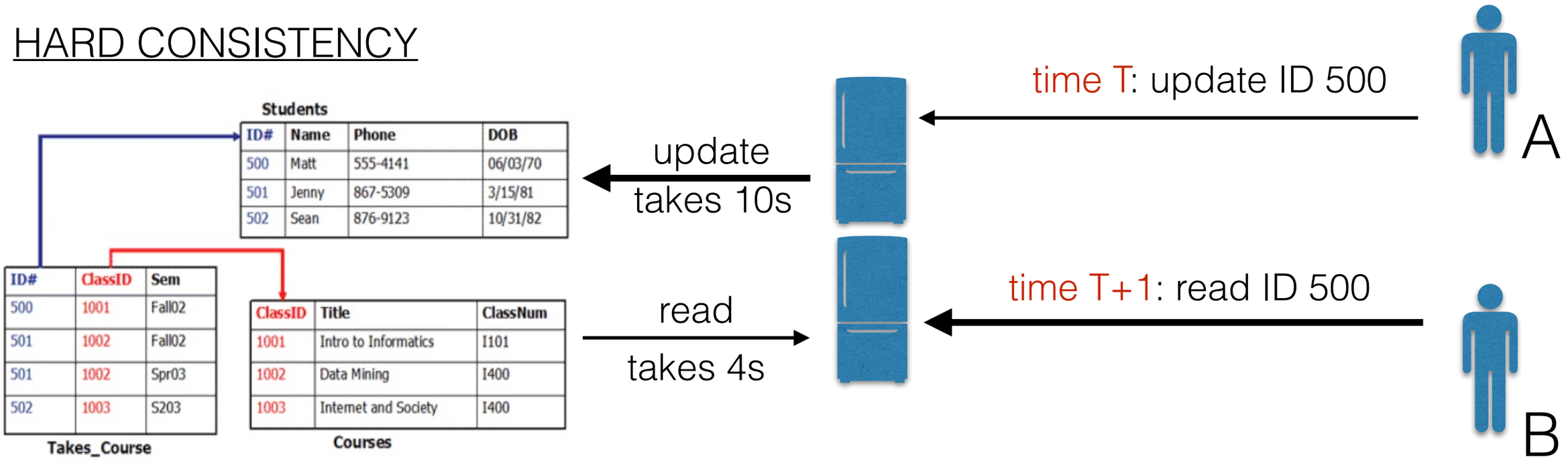
# ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

## HARD CONSISTENCY




**Students**

| ID# | Name | Phone | DOB |
|---|---|---|---|
| 500 | Matt | 555-4141 | 06/03/70 |
| 501 | Jenny | 867-5309 | 3/15/81 |
| 502 | Sean | 876-9123 | 10/31/82 |

**Takes_Course**

| ID# | ClassID | Sem |
|---|---|---|
| 500 | 1001 | Fall02 |
| 501 | 1002 | Fall02 |
| 501 | 1002 | Spr03 |
| 502 | 1003 | S203 |

**Courses**

| ClassID | Title | ClassNum |
|---|---|---|
| 1001 | Intro to Informatics | I101 |
| 1002 | Data Mining | I400 |
| 1003 | Internet and Society | I400 |

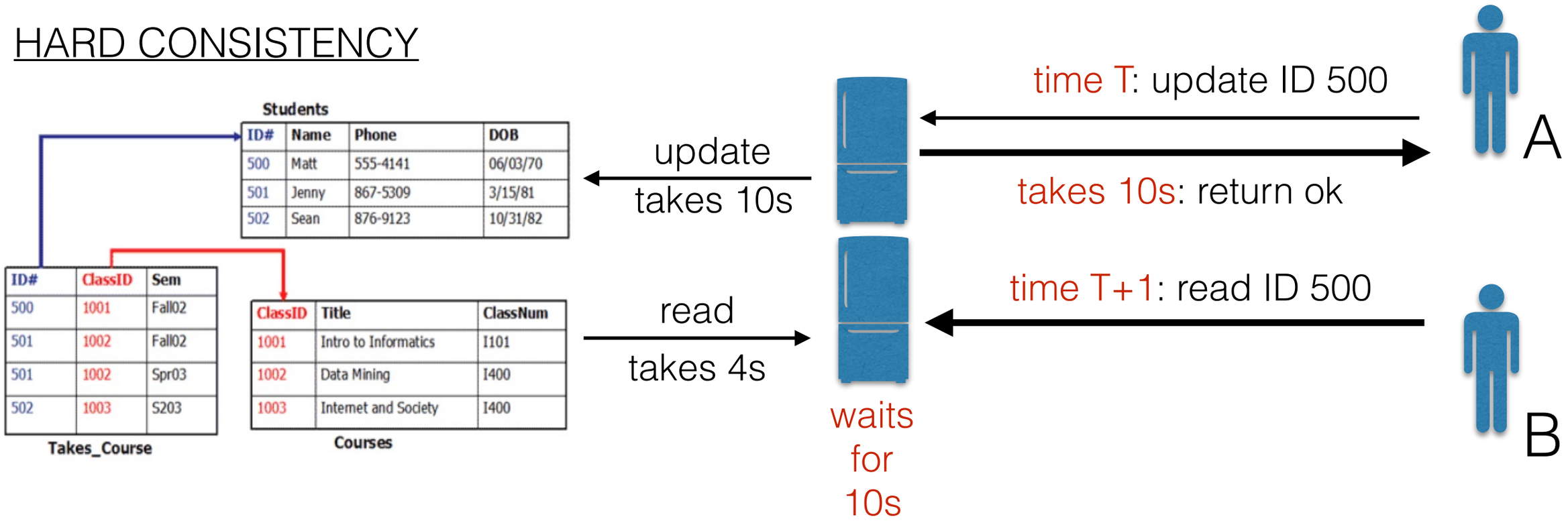
## ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

### HARD CONSISTENCY



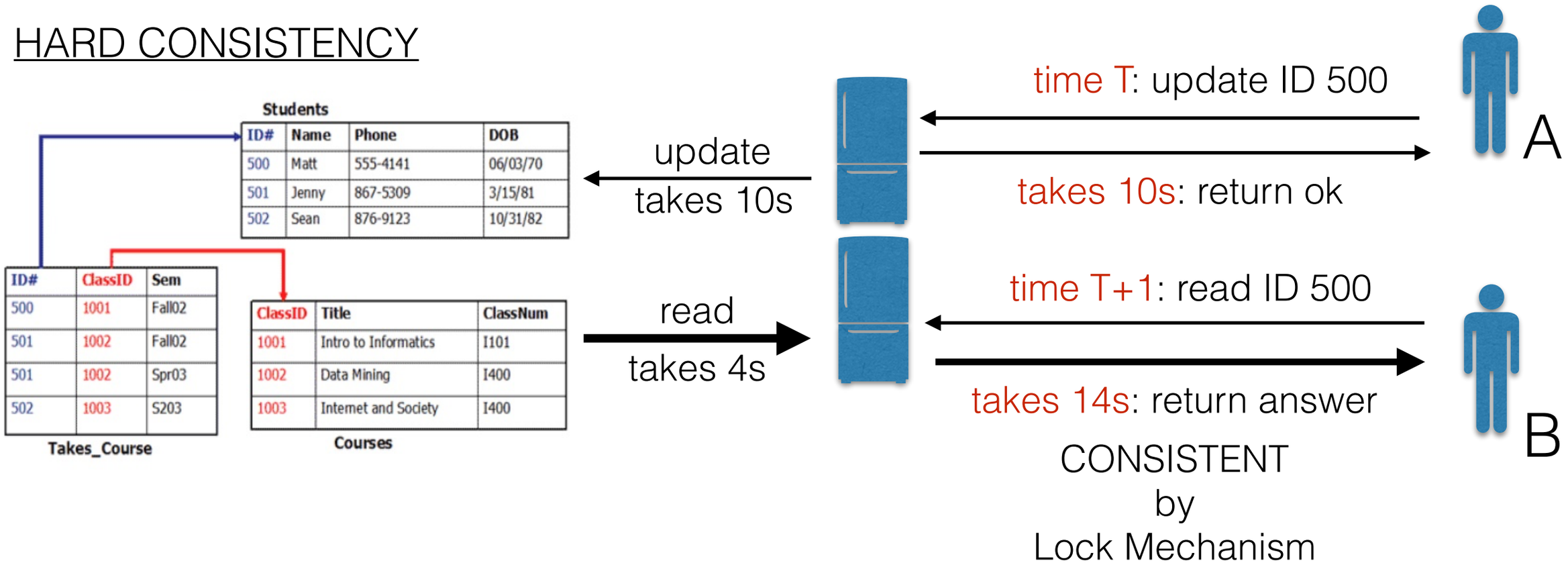
# ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

## HARD CONSISTENCY




The read operation turned to be 3.5 times slower

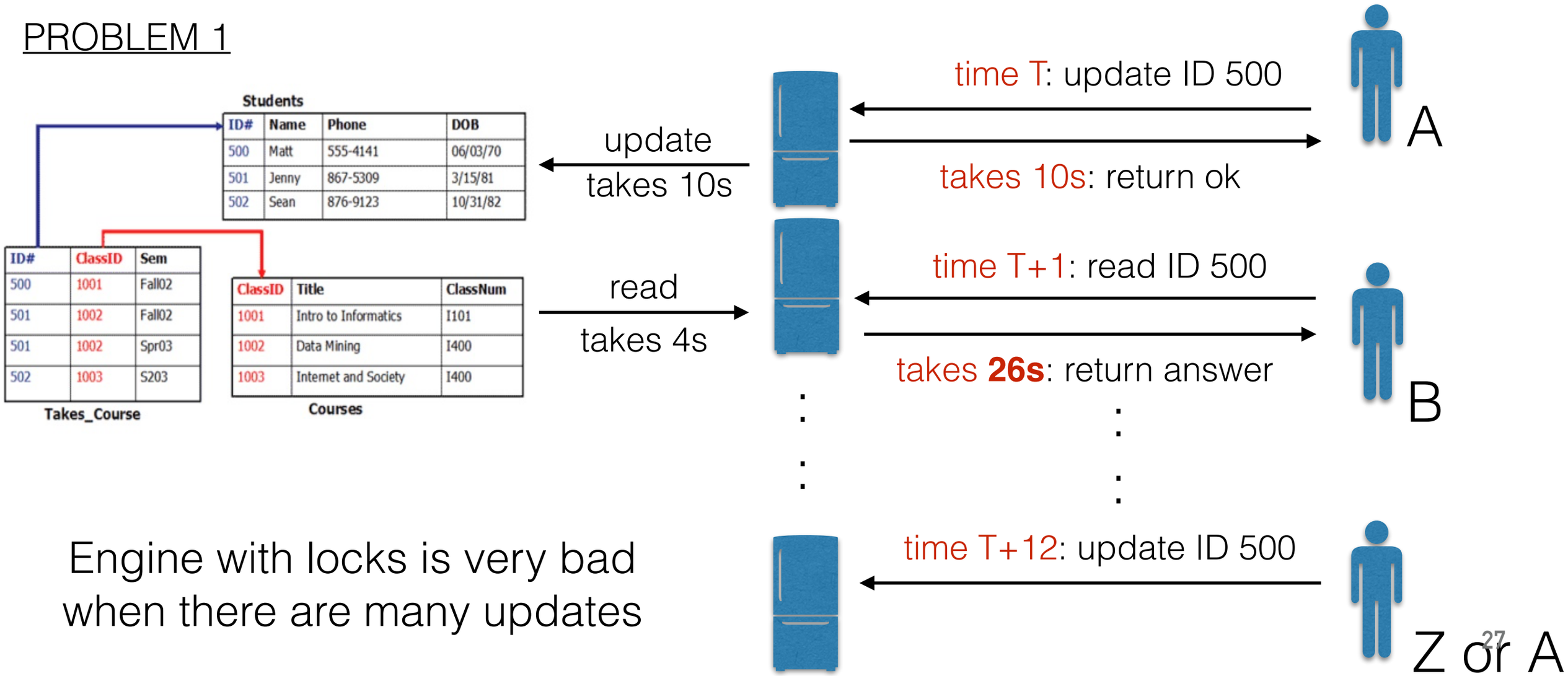
# ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

PROBLEM 1




Engine with locks is very bad when there are many updates

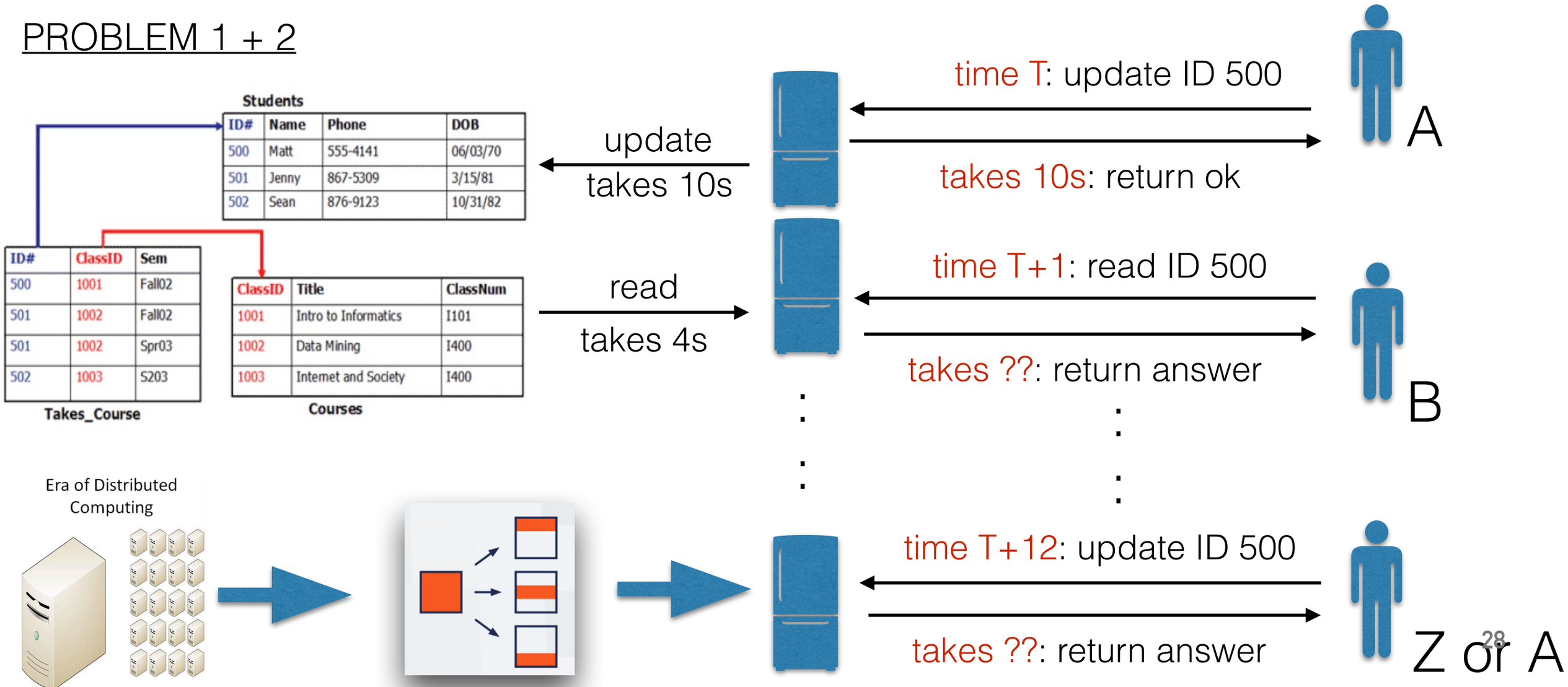
# ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

PROBLEM 1 + 2



## ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

Other non-relational NoSQL databases, however, turn to the BASE model to achieve benefits like scale and resilience. BASE stands for:

- **Basic Availability** — Data is available most of the time, even during a failure that disrupts part of the database. Data is replicated and spread across many different storage systems.
- **Soft-state** — Replicas are not consistent all the time.
- **Eventual consistency** — Data will become consistent at some point in time, with no guarantee when.

BASE principles are less strict than ACID guarantees. BASE values availability over any other value because BASE users care most about scalability. Industries like banking that require data reliability and consistency depend on an ACID-compliant database.

# BENCHMARK

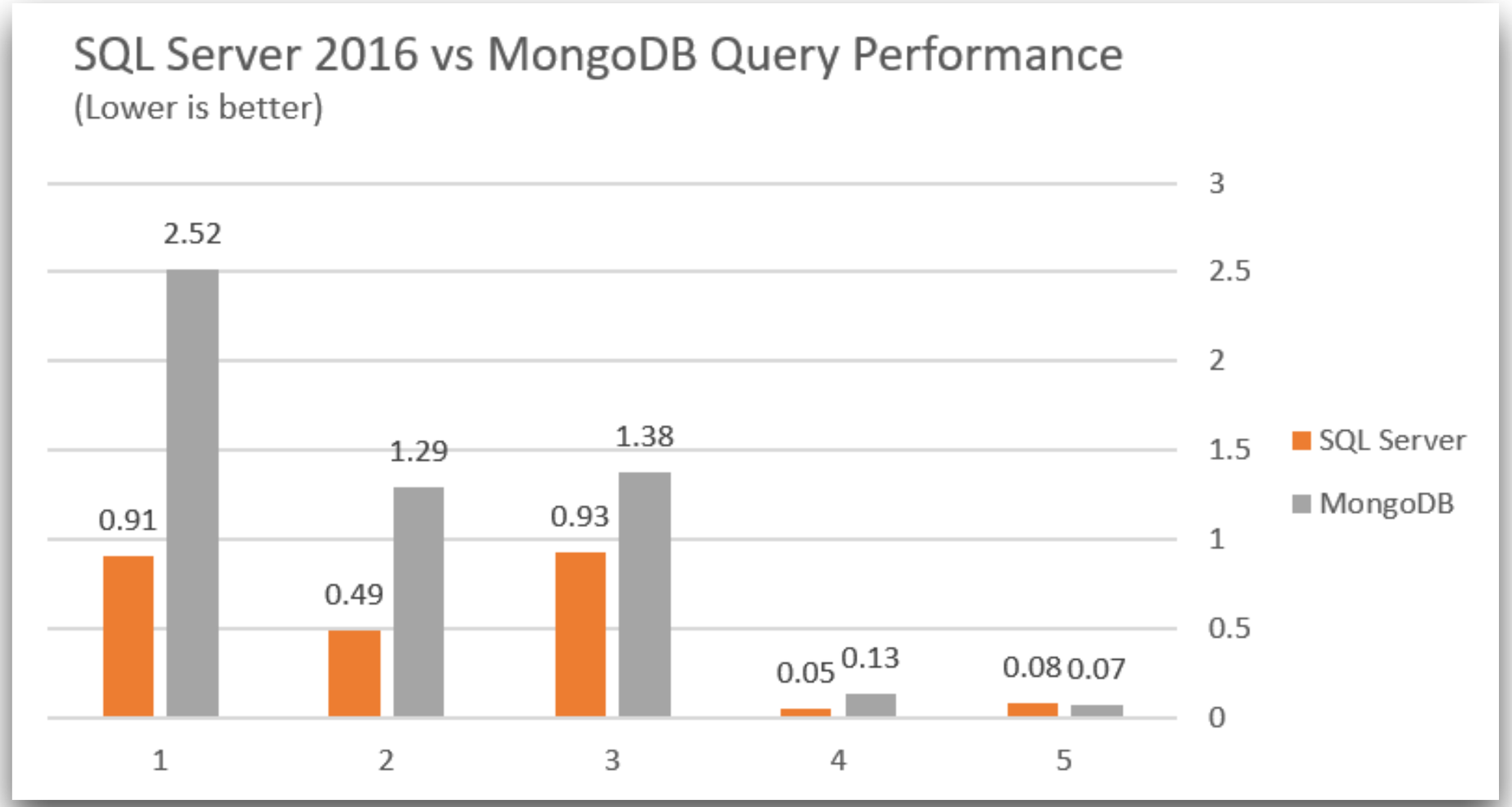
**Some statistics about Facebook Search** (using **Cassandra**)

- MySQL > 50 GB Data
  - Writes Average : ~300 ms
  - Reads Average : ~350 ms

- Rewritten with Cassandra > 50 GB Data
  - Writes Average : 0.12 ms
  - Reads Average : 15 ms


Write Difference: 2 500 times faster
Read Difference: 10 times faster

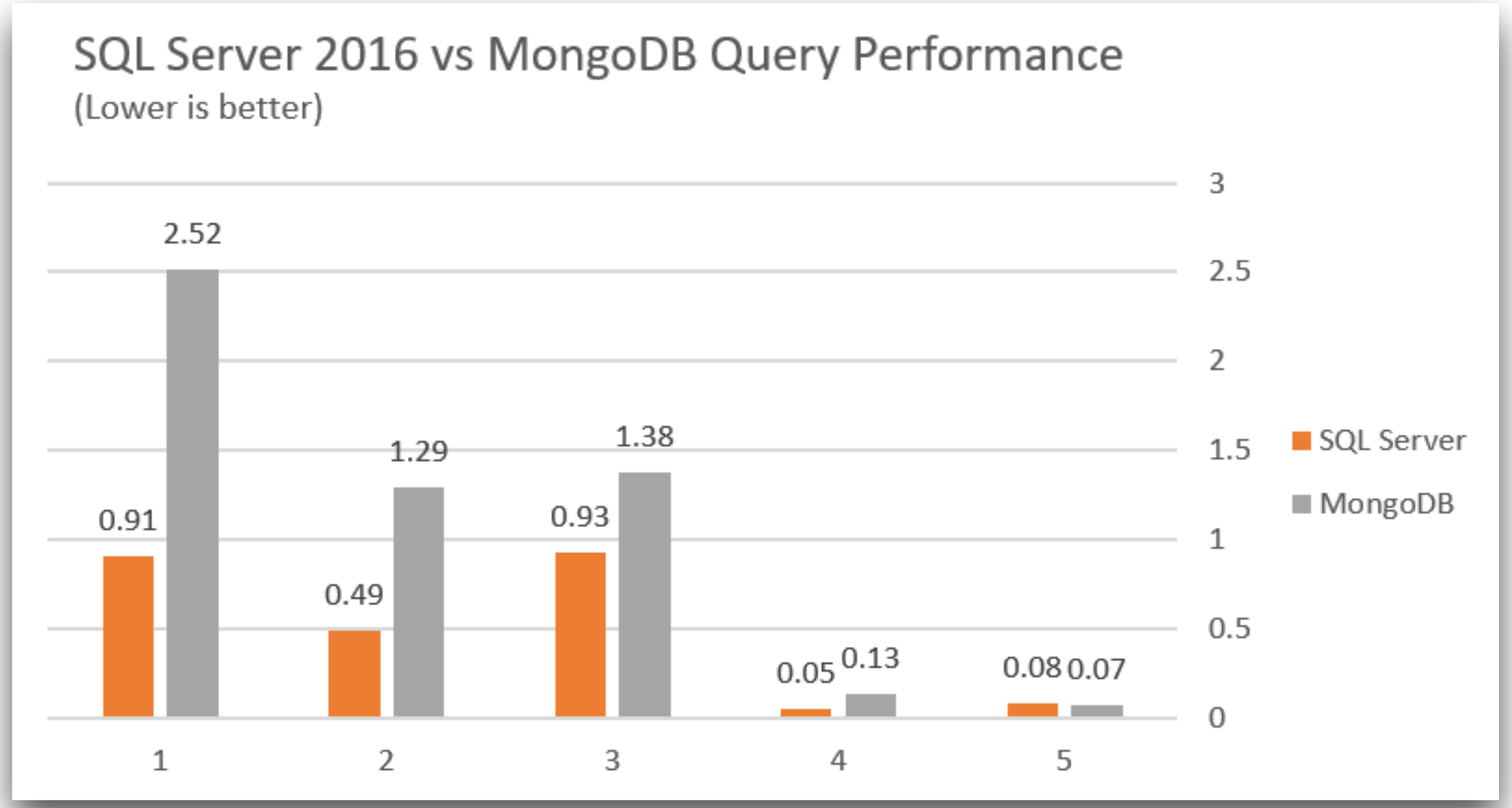
# BENCHMARK




SQL Server is better than MongoDB ?

Then, why is so used in (NodeJS) web apps nowadays?

# BENCHMARK




SQL Server is better than MongoDB ?

Then, why is so used in (NodeJS) web apps nowadays?

Performance clearly depends on (at least) two factors:
- Type of data stored
- Configuration and topology of the noSQL engine

Performance is not everything, Flexibility plays an important role

## ACID vs BASE

ACID and BASE are models of database design. The ACID model has four goals that stand for:

- **Atomicity** — If any part of a transaction isn't executed successfully, the entire operation is rolled back.
- **Consistency** — Ensures that a database remains structurally sound with every transaction.
- **Isolation** — Transactions occur independently of each other and the access to data is moderated.
- **Durability** — All transaction results are permanently preserved in the database.

Relational databases and some NoSQL databases (those that support strong consistency) that meet these four goals are considered ACID-compliant. This means data is consistent after transactions are complete.

Other non-relational NoSQL databases, however, turn to the BASE model to achieve benefits like scale and resilience. BASE stands for:

- **Basic Availability** — Data is available most of the time, even during a failure that disrupts part of the database. Data is replicated and spread across many different storage systems.
- **Soft-state** — Replicas are not consistent all the time.
- **Eventual consistency** — Data will become consistent at some point in time, with no guarantee when.

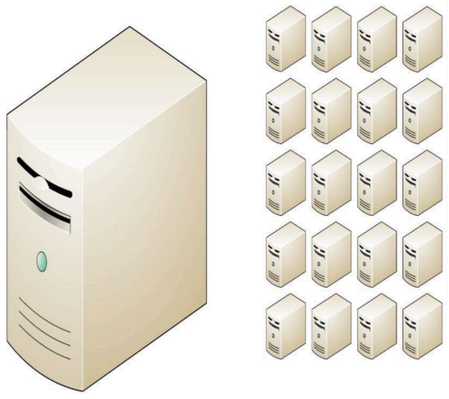
BASE principles are less strict than ACID guarantees. BASE values availability over any other value because BASE users care most about scalability. Industries like banking that require data reliability and consistency depend on an ACID-compliant database.

## Distribution and Scalability

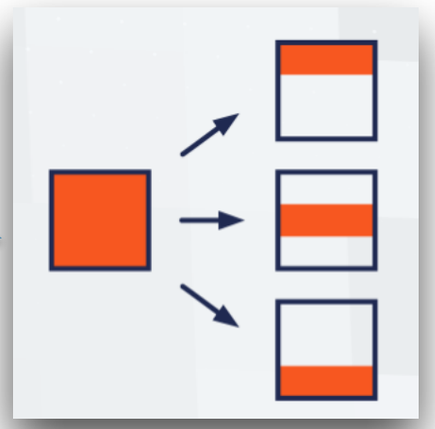
Scaling up a relational SQL database often requires taking it offline so new hardware can be added to the servers. But the hardware would remain even when it came time to scale down.

Single node peer-to-peer architecture, however, allows for easy scalability. As new nodes are added to the database clusters, performance is increased. This scaleout architecture also adds more servers running replicas of the database. A simple click of the mouse can scale clusters up and down to meet demand.

Era of Distributed Computing



Data Sharding

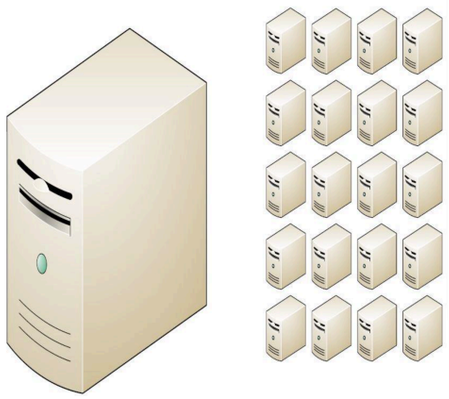


# STORAGE PARADIGMA

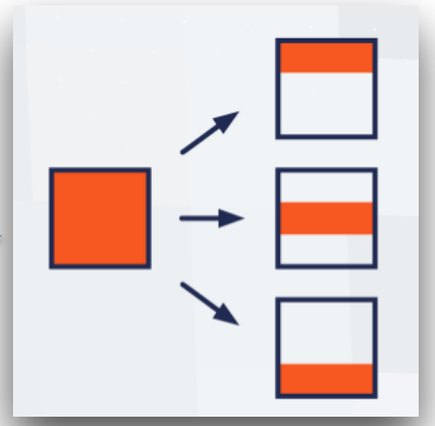
We need a distributed database system having such features:

- **Fault tolerance**
- **High availability**
- **Consistency**
- **Scalability**

Era of Distributed Computing



Data Sharding



# STORAGE PARADIGMA

We need a distributed database system having such features:

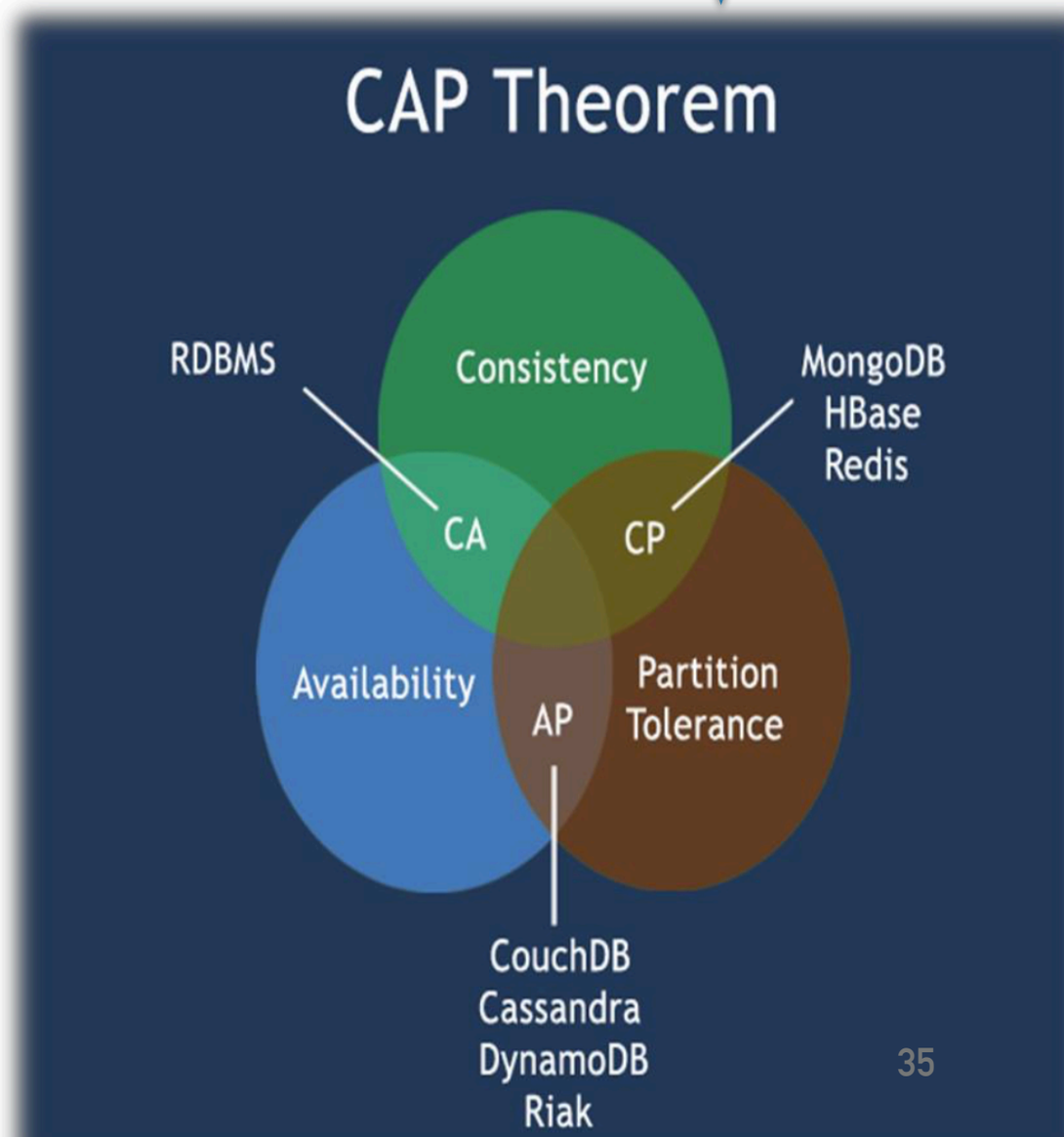
- **Fault tolerance**
- **High availability**
- **Consistency**
- **Scalability**

**Which is impossible!!!**

**According to CAP theorem**

# The CAP Theorem

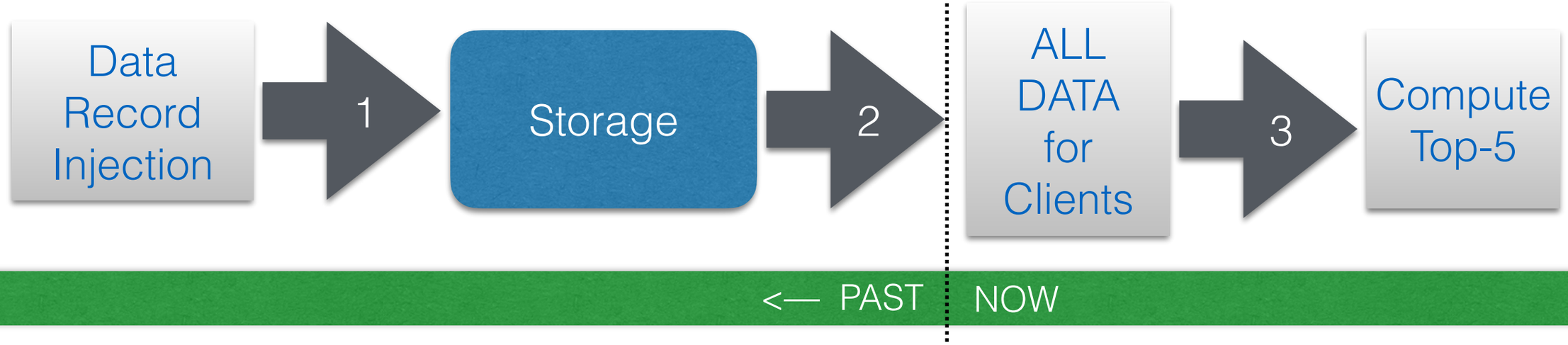
- Impossible for any shared data-system to guarantee simultaneously all of the following three properties:
  - Consistency – once data is written, all future read requests will contain that data
  - Availability – the database is always available and responsive
  - Partition Tolerance – if part of the database is unavailable, other parts are unaffected



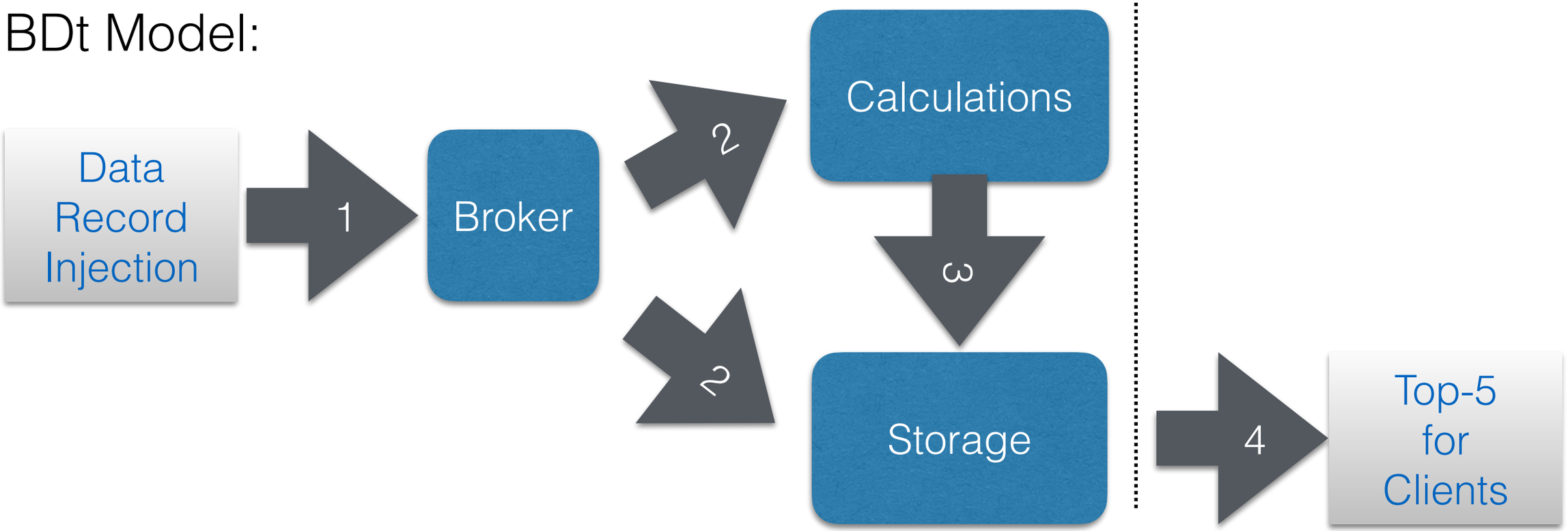

**newSQL**: several noSQL classes with a (dynamic) balance between consistency and performance

**How can Streaming Algorithms be used in (big data) Database Engines?**

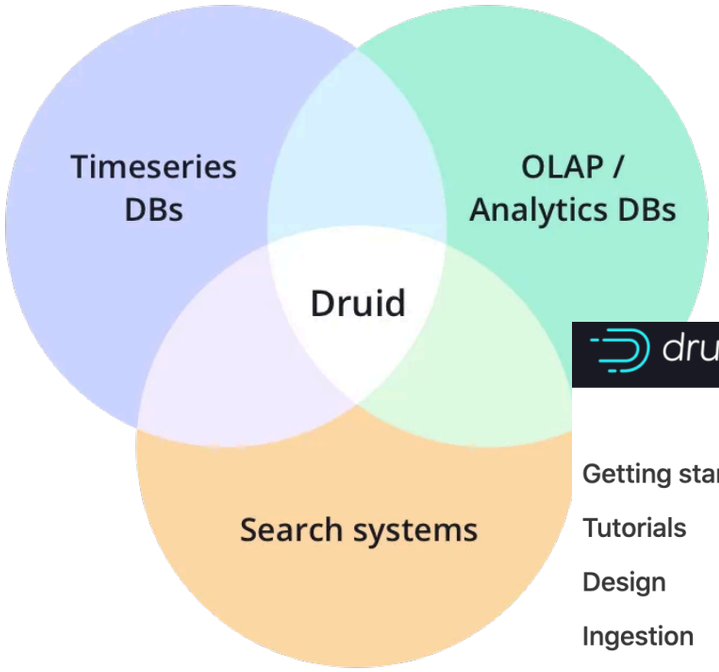
Classical Model:




BDt Model:



# Connection: How noSQL connects to HLL?




druid Technology Use Cases Powered By Docs Community Apache Download Search

- Getting started
- Tutorials
- Design
- Ingestion
- Querying
  - Druid SQL
  - Native queries
  - Query execution
  - Concepts
    - Datasources
    - Joins
    - Lookups
    - Multi-value dimensions
    - Multitenancy
    - Query caching
    - Context parameters
  - Native query types
    - Timeseries
    - TopN
    - GroupBy

## Approximate Aggregations

### Count distinct

#### Apache DataSketches Theta Sketch

The DataSketches Theta Sketch extension-provided aggregator gives distinct count estimates with support for set union, intersection, and difference post-aggregators, using Theta sketches from the Apache DataSketches library.

#### Apache DataSketches HLL Sketch

The DataSketches HLL Sketch extension-provided aggregator gives distinct count estimates using the HyperLogLog algorithm.

Compared to the Theta sketch, the HLL sketch does not support set operations and has slightly slower update and merge speed, but requires significantly less space.

#### Cardinality, hyperUnique

> For new use cases, we recommend evaluating DataSketches Theta Sketch or DataSketches HLL Sketch instead. The DataSketches aggregators are generally able to offer more flexibility and better accuracy than the classic Druid `cardinality` and `hyperUnique` aggregators.

The Cardinality and HyperUnique aggregators are older aggregator implementations available by default in Druid that also provide distinct count estimates using the HyperLogLog algorithm. The newer DataSketches Theta and HLL extension-provided aggregators described above have superior accuracy and performance and are recommended instead.

- Approximate Aggregations
  - Count distinct
  - Histograms and quantiles
- Miscellaneous Aggregations
  - Filtered Aggregator

# Playground

1. Find in the literature: two applications of HLL
2. Explain HLL.ipynb
3. Design some tests for the precision of the algorithm
4. Run the tests on the notebook
5. Deploy (docker) a REDIS container
6. Run the tests using REDIS HLL
7. Discuss the obtained results