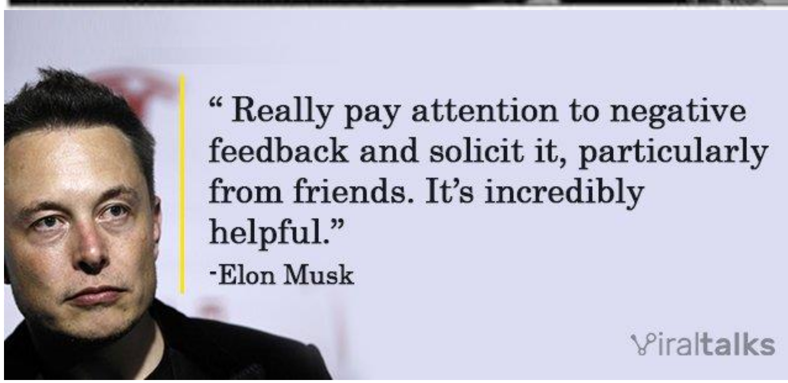
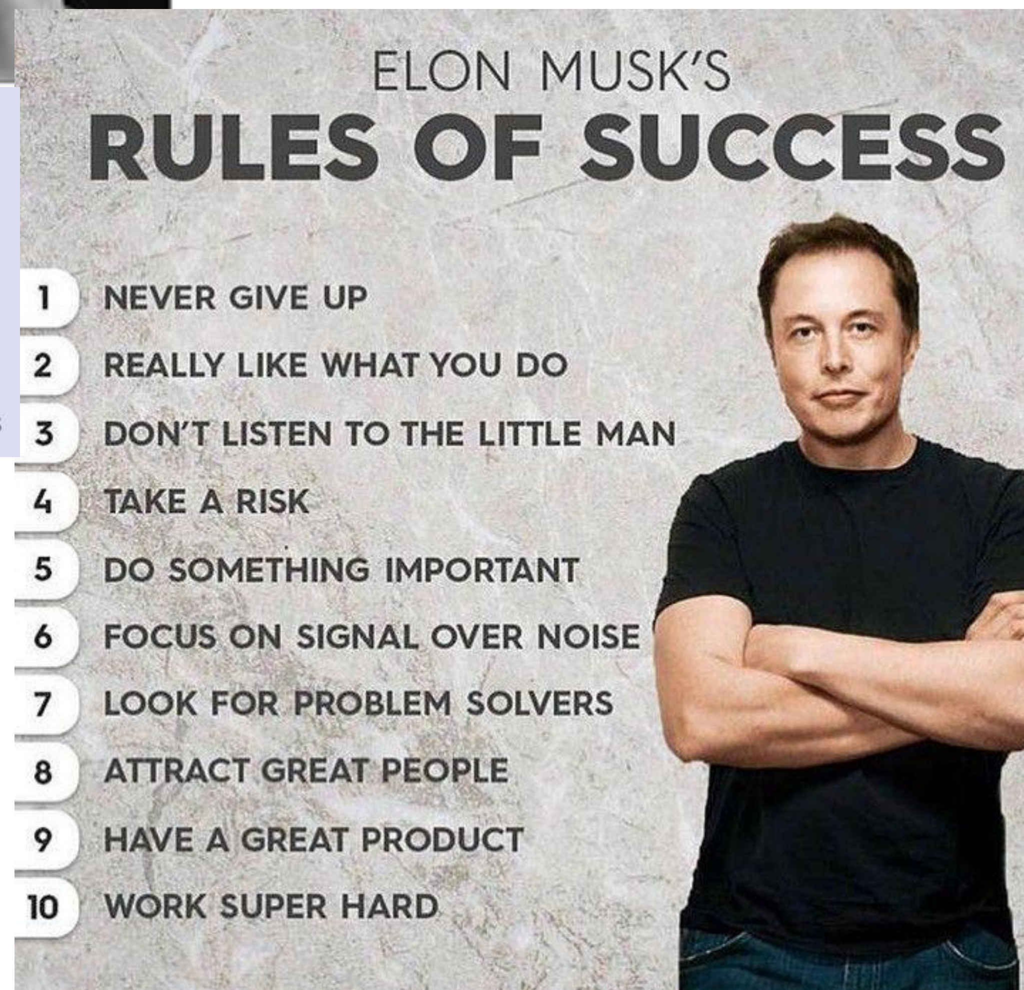

It's very important to have a feedback loop, where you're constantly thinking about what you've done and how you could be doing it better.
- Elon Musk


" Really pay attention to negative feedback and solicit it, particularly from friends. It's incredibly helpful."
-Elon Musk
Viraltalks


ELON MUSK'S
RULES OF SUCCESS
1 NEVER GIVE UP
2 REALLY LIKE WHAT YOU DO
3 DON'T LISTEN TO THE LITTLE MAN
4 TAKE A RISK
5 DO SOMETHING IMPORTANT
6 FOCUS ON SIGNAL OVER NOISE
7 LOOK FOR PROBLEM SOLVERS
8 ATTRACT GREAT PEOPLE
9 HAVE A GREAT PRODUCT
10 WORK SUPER HARD

# Types of Data

- Data can be broadly classified into four types:
  1. Structured Data:
     - Have a predefined model, which organizes data into a form that is relatively easy to store, process, retrieve and manage
     - E.g., relational data
  2. Unstructured Data:
     - Opposite of structured data
     - E.g., Flat binary files containing text, video or audio
     - <u>Note</u>: data is not completely devoid of a structure (e.g., an audio file may still have an encoding structure and some metadata associated with it)

# Types of Data

- Data can be broadly classified into four types:

  3. Dynamic Data:
     - Data that changes relatively frequently
     - E.g., office documents and transactional entries in a financial database

  4. Static Data:
     - Opposite of dynamic data
     - E.g., Medical imaging data from MRI or CT scans

# Why Classifying Data?

- Segmenting data into one of the following 4 quadrants can help in designing and developing a pertaining storage solution

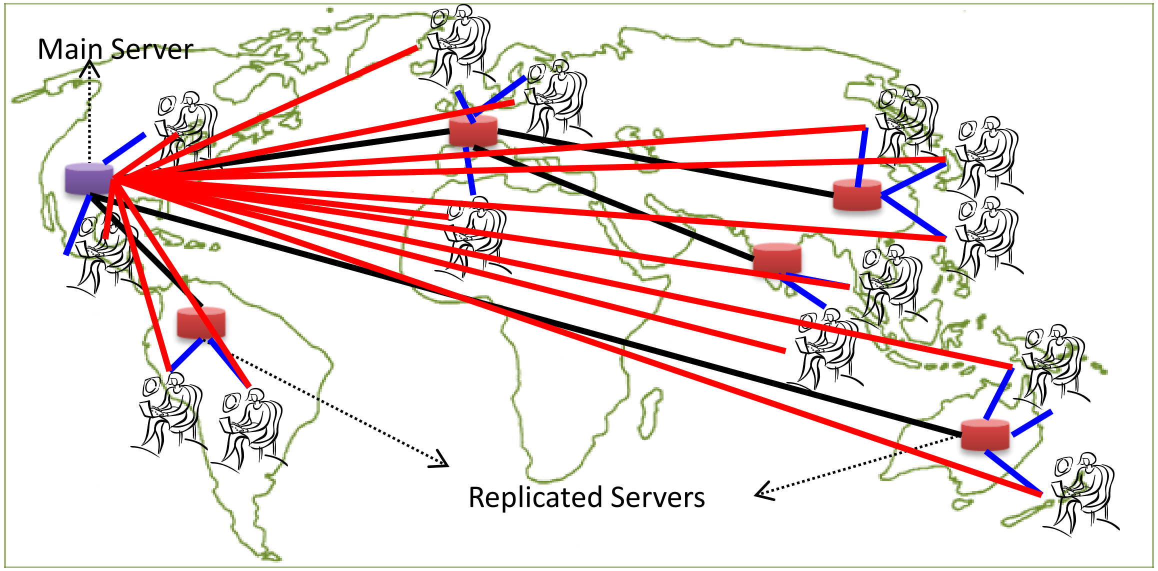
| | **Dynamic** | **Static** |
|---|---|---|
| **Unstructured** | Media Production, eCAD, mCAD, Office Docs | Media Archive, Broadcast, Medical Imaging |
| **Structured** | Transaction Systems, ERP, CRM | BI, Data Warehousing |

- Relational databases are usually used for structured data

- File systems or *NoSQL databases* can be used for (static), unstructured data

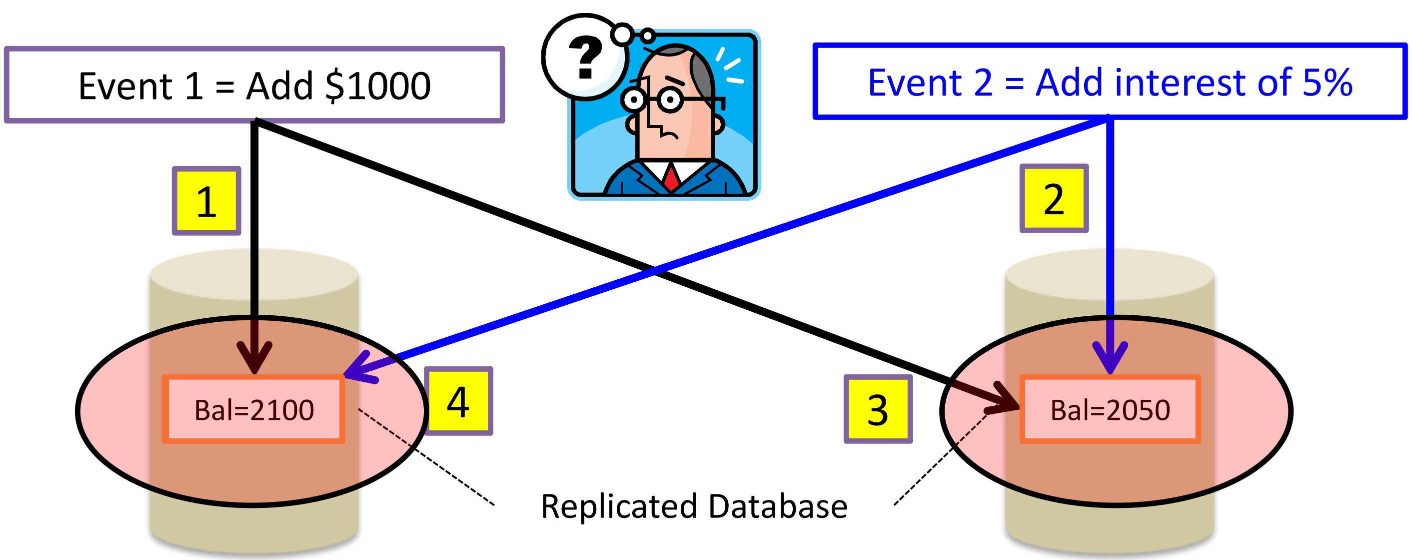
# Why Replicating Data?

- Replicating data across servers helps in:
  - Avoiding performance bottlenecks
  - Avoiding single point of failures
  - And, hence, enhancing scalability and availability



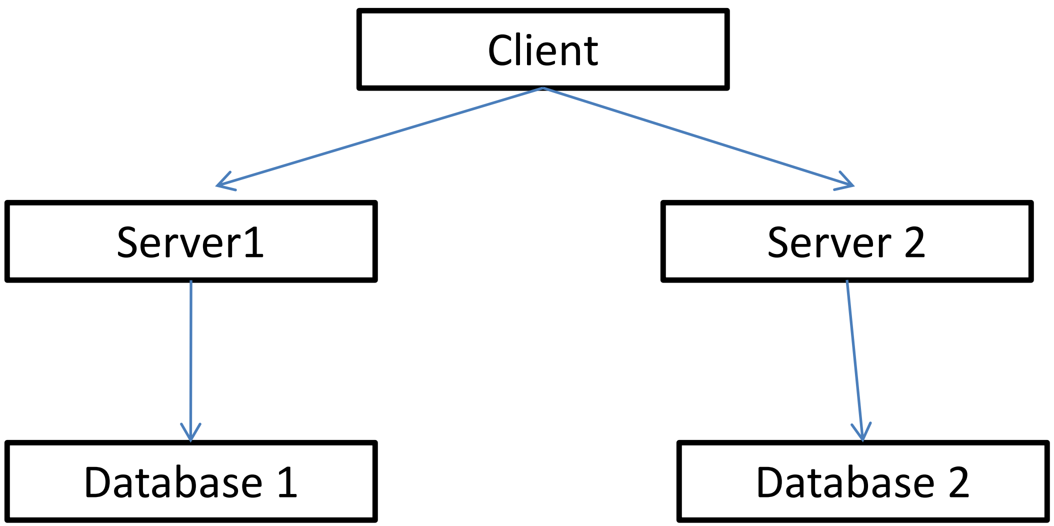
# But, Consistency Becomes a Challenge

- An example:
  - In an e-commerce application, the bank database has been replicated across two servers
  - Maintaining consistency of replicated data is a challenge



# Distributed DB

- Not all the data can reside in the same database
- The application is built on top of the database.

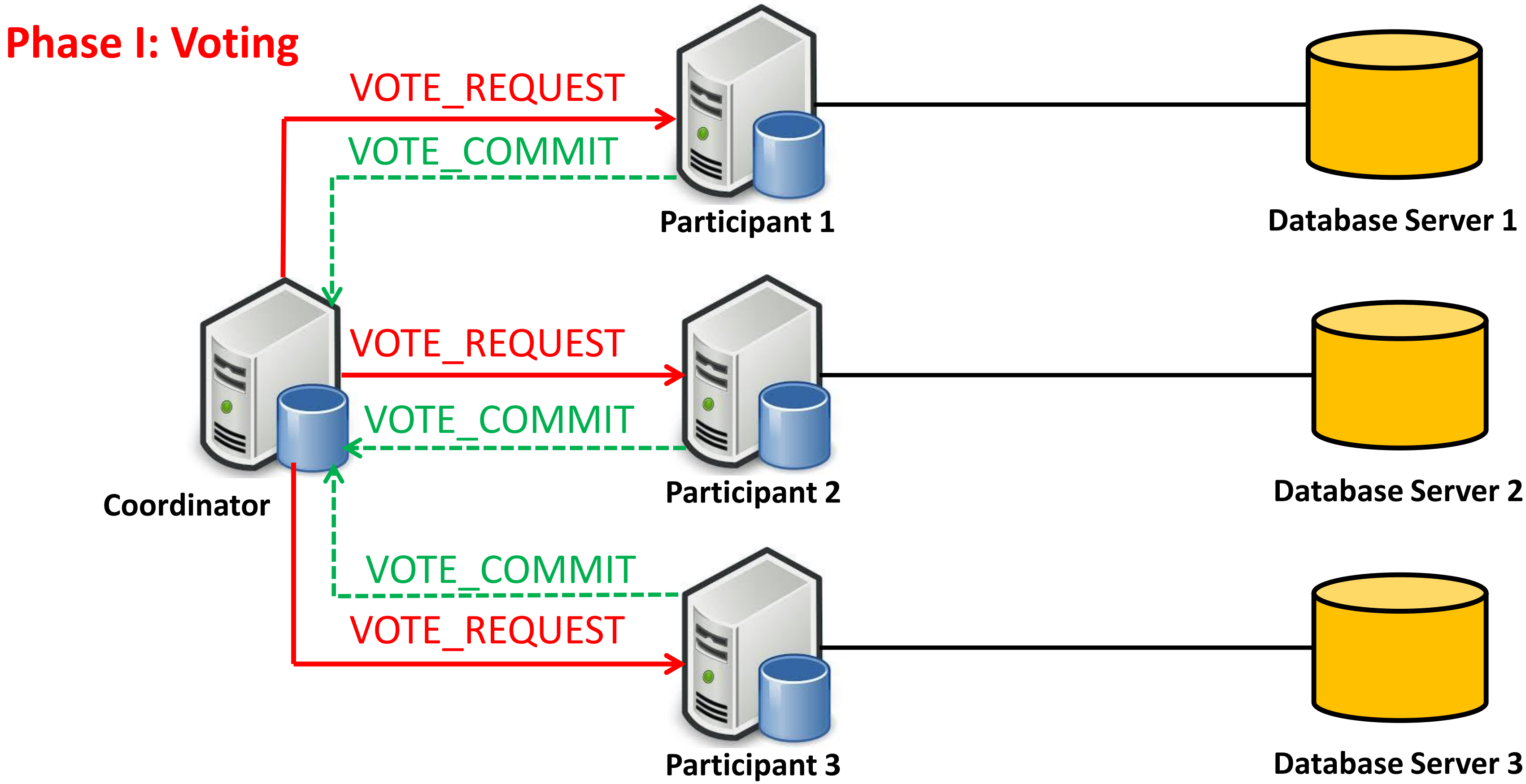


# 2 phase commit protocol

- „prepare to commit“ message to server
- server responds „ready to commit“
  - guarantees successful commit of procedure
- check whether all servers are ready
  - ready: commit results of requests
  - not ready: cancelation, rollback
  - log states of transactions

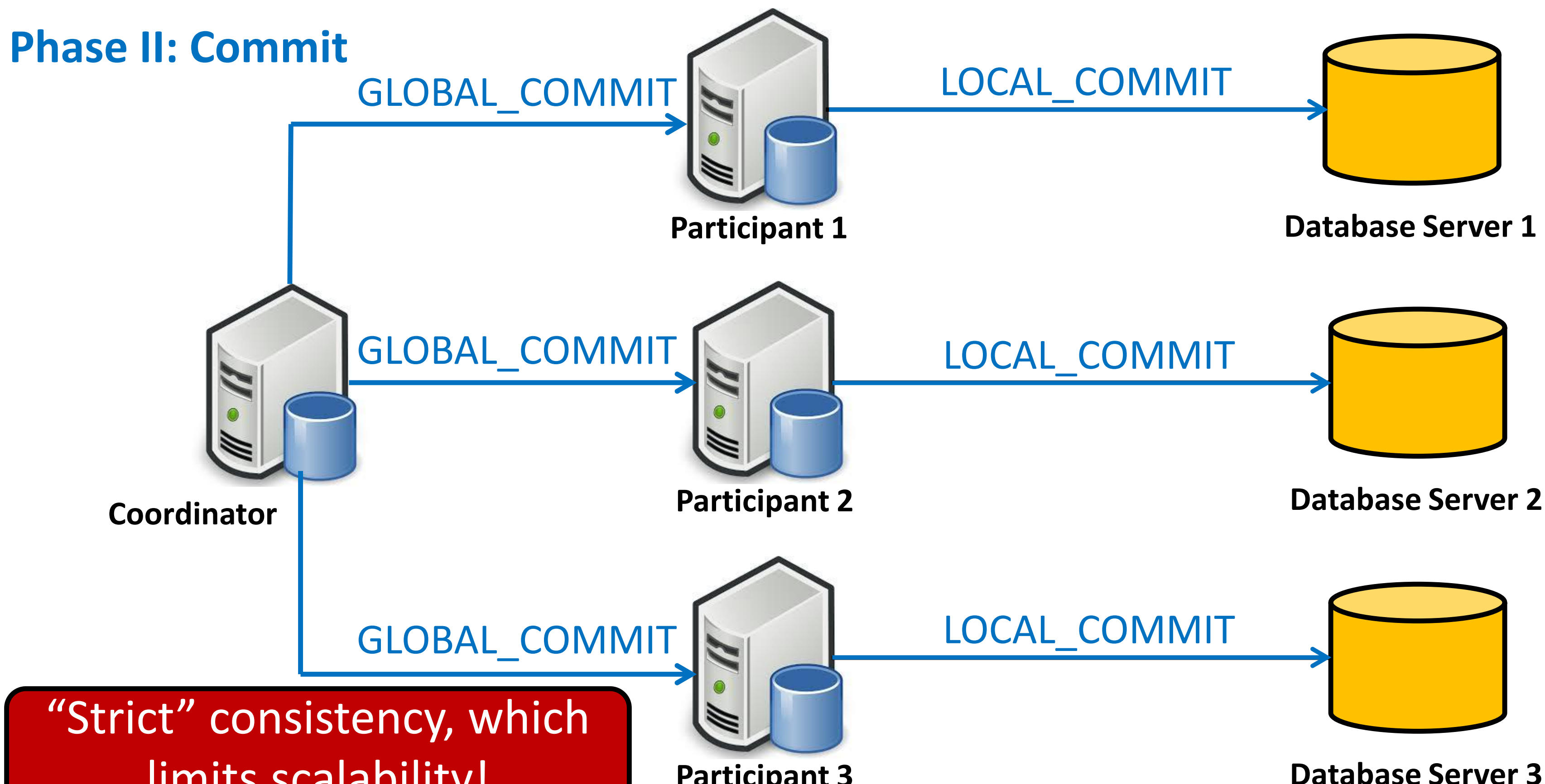
# The Two-Phase Commit Protocol

- The two-phase commit protocol (2PC) can be used to ensure atomicity and consistency



# The Two-Phase Commit Protocol

- The two-phase commit protocol (2PC) can be used to ensure atomicity and consistency



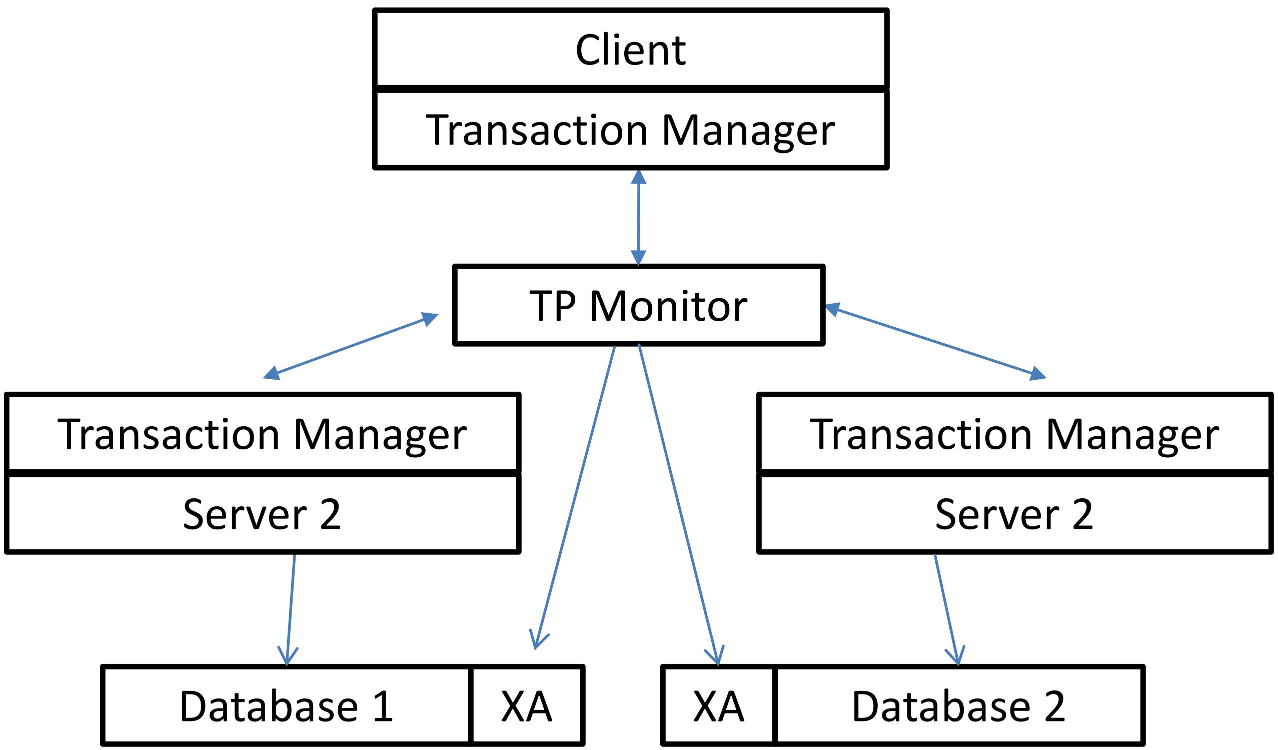

"Strict" consistency, which limits scalability!

# TP-Monitors

- The problems of synchronous interaction are not new. The first systems to provide alternatives were TP-Monitors :
- asynchronous RPC: client makes a call that returns immediately; the client is responsible for making a second call to get the results
- Reliable queuing systems (e.g., Tuxedo) where instead of through procedure calls, client and server interact by exchanging messages. Making the messages persistent by storing them in queues

# Coordinator

- the TM runs 2PC with resource managers instead of with the server



# TP-Monitor

- Common interface to several applications while maintaining or adding transactional properties. Examples: CICS, Tuxedo, Encina.
- A TP-Monitor extends the transactional capabilities of a database beyond the database domain
- TP-Monitors are, perhaps, the best, oldest, and most complex example of middleware.

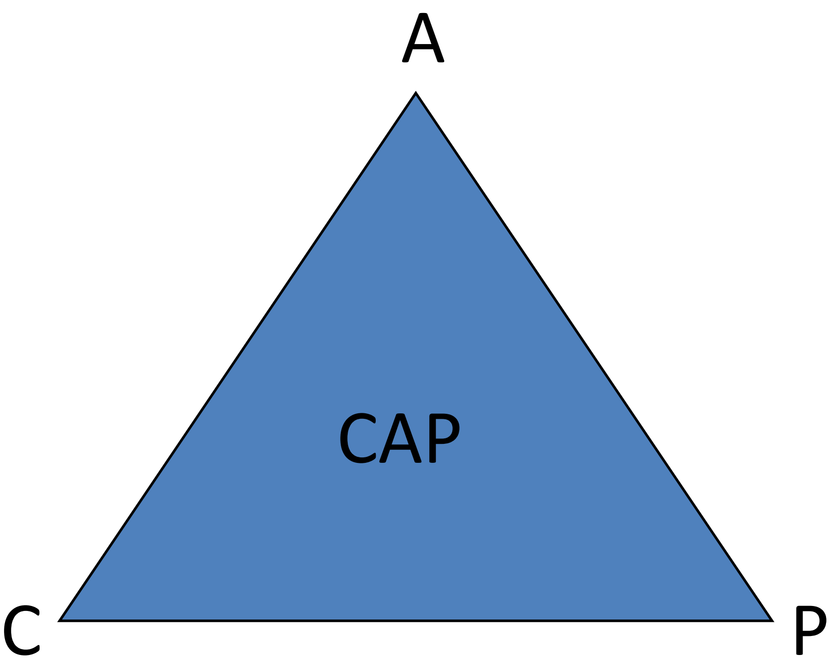
# C and Latency Tradeoff

- Amazon claims that just an extra one tenth of a second on their response times will cost them 1% in sales.
- Google said they noticed that just a half a second increase in latency caused traffic to drop by a fifth.

# CAP Theorem

- Consistency
- Availability
- Partition Tolerance

• Choose two




[CAP Twelve Years Later: How the Rules Have Changed (Eric Brewer)](https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6133253)
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6133253

[The CAP Theorem's Growing Impact](https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6155651)
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6155651
(Simon Shim)

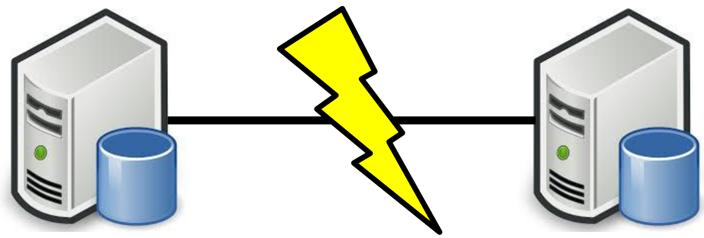
# The CAP Theorem

- The limitations of distributed databases can be described in the so called the CAP theorem
  - **C**onsistency: every node always sees the same data at any given instance (i.e., strict consistency)
  - **A**vailability: the system continues to operate, even if nodes in a cluster crash, or some hardware or software parts are down due to upgrades
  - **P**artition Tolerance: the system continues to operate in the presence of network partitions

CAP theorem: any distributed database with shared data, can have *at most two* of the three desirable properties, C, A or P

# The CAP Theorem (*Cont'd*)

- Let us assume two nodes on opposite sides of a network partition:



- Availability + Partition Tolerance forfeit Consistency

- Consistency + Partition Tolerance entails that one side of the partition must act as if it is unavailable, thus forfeiting Availability

- Consistency + Availability is only possible if there is no network partition, thereby forfeiting Partition Tolerance

# Questions?

- Which one would you choose when network partition?
  (a) C (b) A

- Which of CAP is essential for a distributed system?
  (a) C (b) A (c) P (d) none of the above

# CAP

- Dynamo does not guarantee C by default
- The event of P forces systems to decide on reducing C or A
- What is the probability of P?
  - Local network
  - Wide area network

# Large-Scale Databases

- When companies such as Google and Amazon were designing large-scale databases, 24/7 Availability was a key
  - A few minutes of downtime means lost revenue

- When *horizontally* scaling databases to 1000s of machines, the likelihood of a node or a network failure increases tremendously

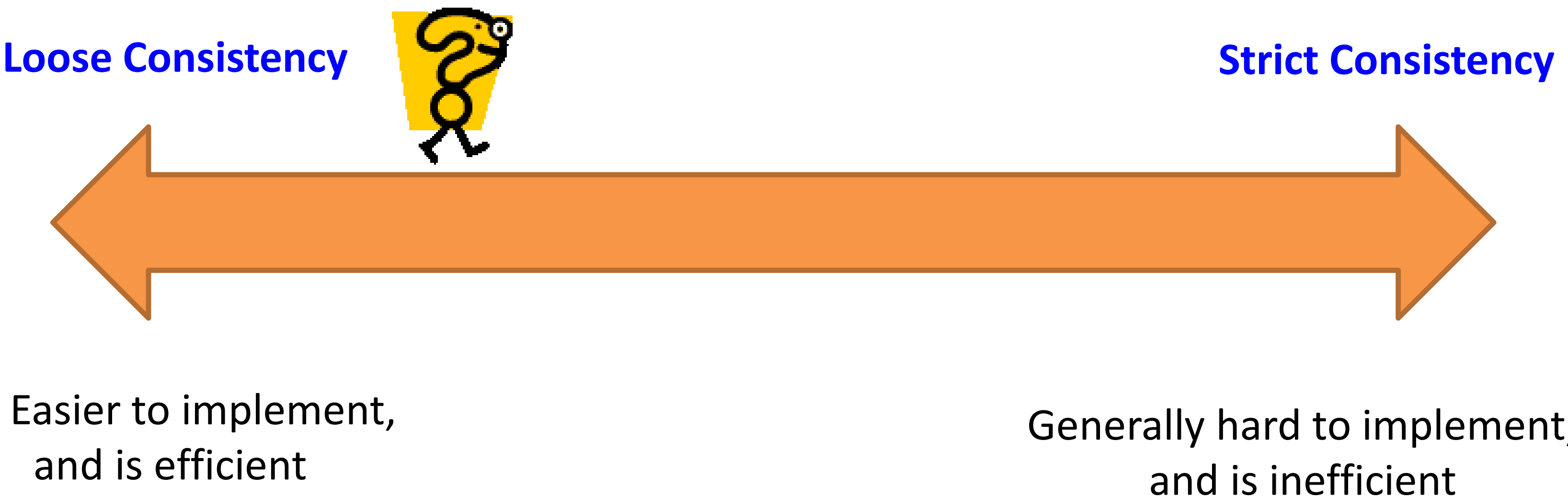
- Therefore, in order to have strong guarantees on Availability and Partition Tolerance, they had to sacrifice “strict” Consistency (*implied by the CAP theorem*)

# Trading-Off Consistency

- Maintaining consistency should balance between the strictness of consistency versus availability/scalability
  - Good-enough consistency *depends on your application*

# Trading-Off Consistency

- Maintaining consistency should balance between the strictness of consistency versus availability/scalability
  - Good-enough consistency *depends on your application*



# The BASE Properties

- The CAP theorem proves that it is impossible to guarantee
- strict Consistency and Availability while being able to tolerate network partitions

- This resulted in databases with relaxed ACID guarantees

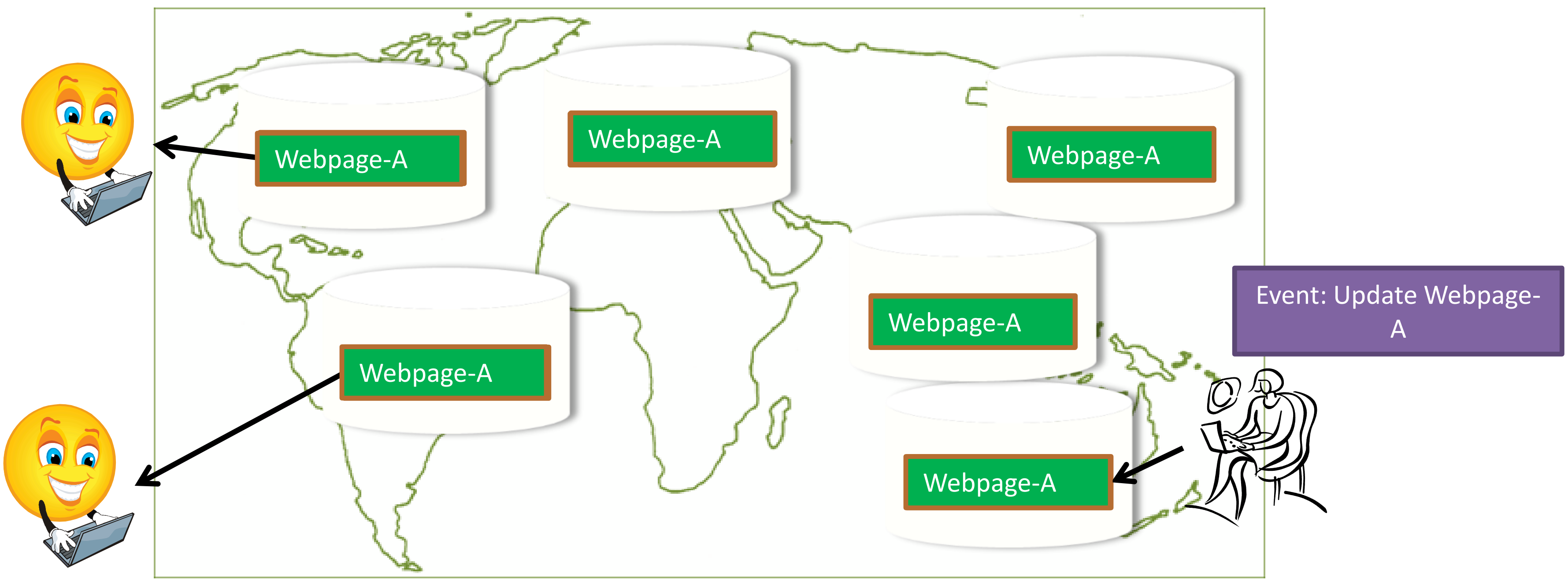
- In particular, such databases apply the BASE properties:
  - **B**asically **A**vailable: the system guarantees Availability
  - **S**oft-State: the state of the system may change over time
  - **E**ventual Consistency: the system will *eventually* become consistent

# Eventual Consistency

- A database is termed as *Eventually Consistent* if:
  - All replicas will *gradually* become consistent

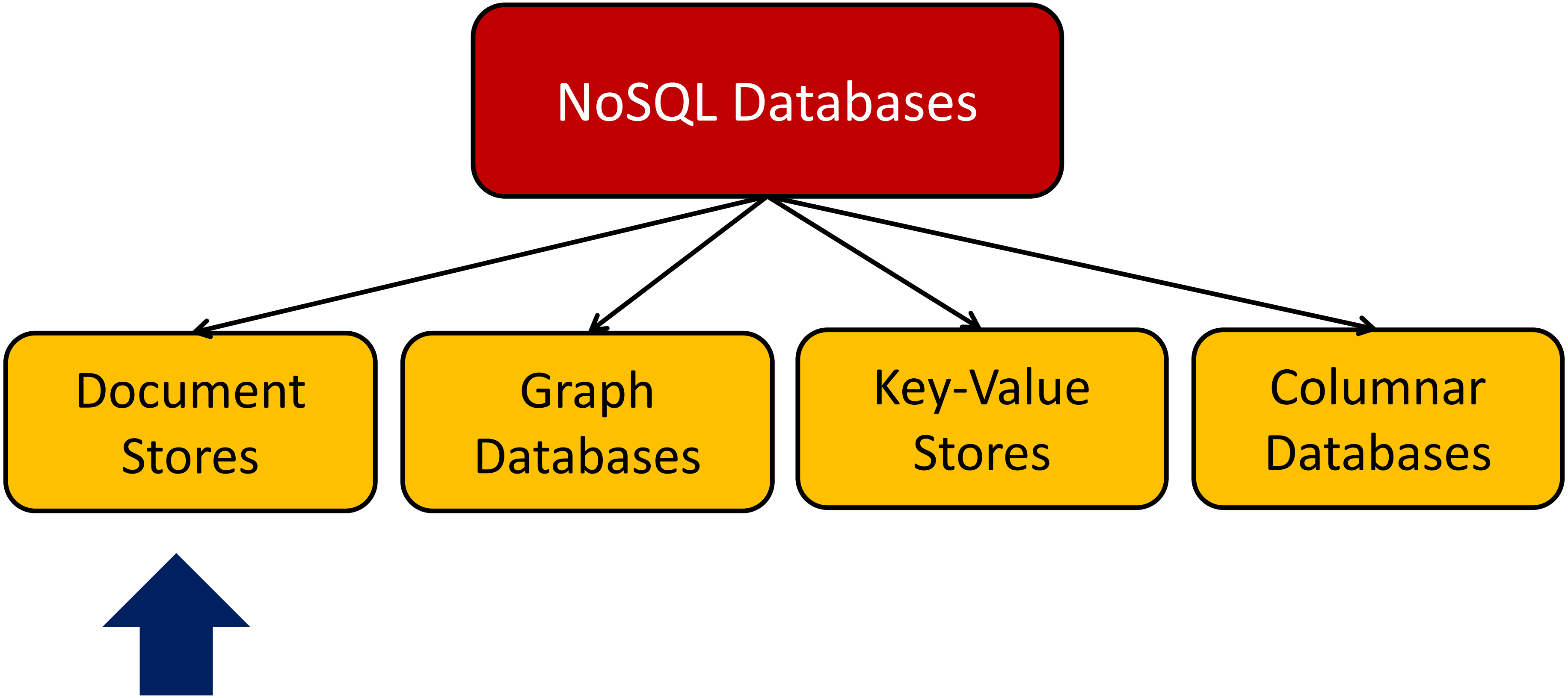
# Eventual Consistency

- A database is termed as *Eventually Consistent* if:
  - All replicas will *gradually* become consistent



# Types of NoSQL Databases

- Here is a limited taxonomy of NoSQL databases:

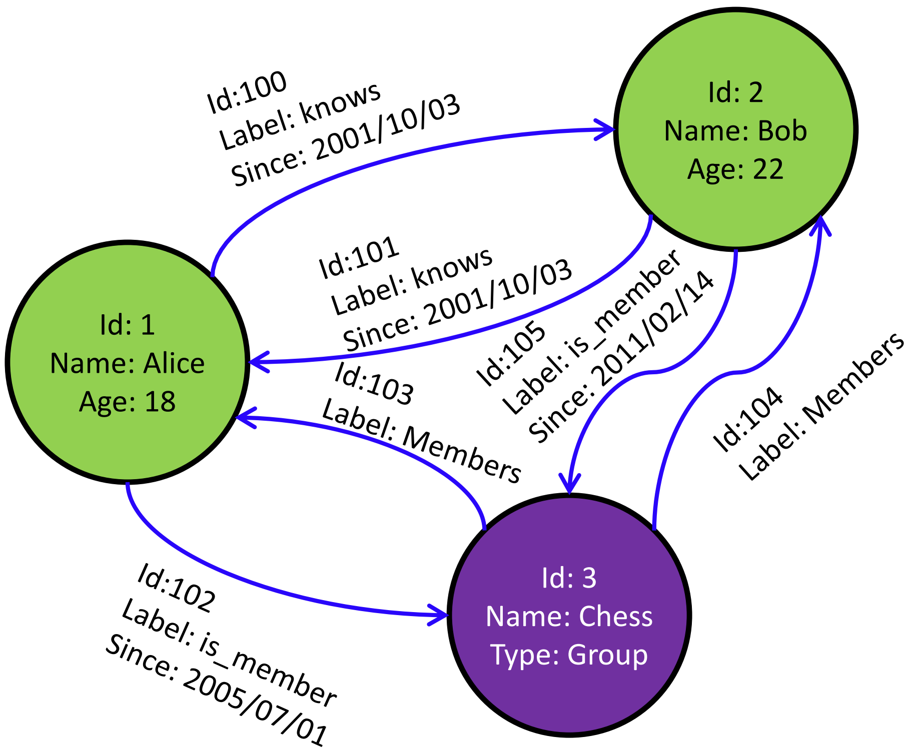


# Document Stores

- Documents are stored in some standard format or encoding (e.g., XML, JSON, PDF or Office Documents)
  - These are typically referred to as Binary Large Objects (BLOBs)
- Documents can be indexed
  - This allows document stores to outperform traditional file systems
- E.g., MongoDB and CouchDB

# Graph Databases

- Data are represented as vertices and edges




- Graph databases are powerful for graph-like queries (e.g., find the shortest path between two elements)

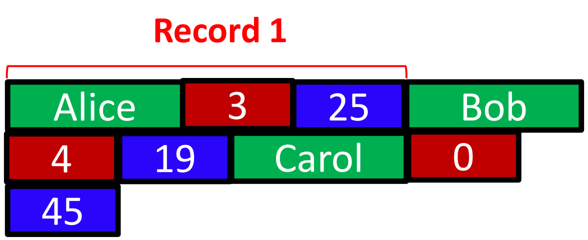
- E.g., Neo4j and VertexDB

# Key-Value Stores

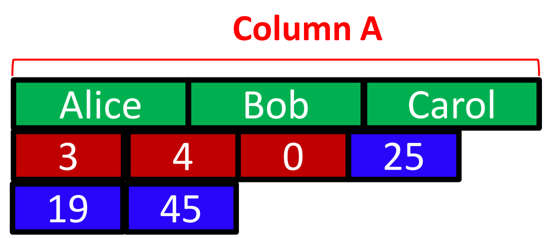
- Keys are mapped to (possibly) more complex value (e.g., lists)
- Keys can be stored in a hash table and can be distributed easily
- Such stores typically support regular CRUD (create, read, update, and delete) operations
  - That is, no joins and aggregate functions
- E.g., Amazon DynamoDB and Apache Cassandra

# Columnar Databases

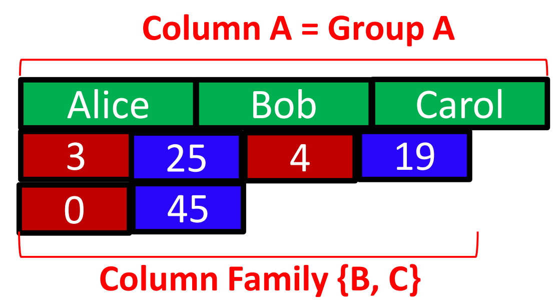
- Columnar databases are a hybrid of RDBMSs and Key-Value stores
  - Values are stored in groups of zero or more columns, but in Column-Order (as opposed to Row-Order)




*Row-Order*




*Columnar (or Column-Order)*




*Columnar with Locality Groups*

  - Values are queried by matching keys

- E.g., HBase and Vertica