CAP Theorem

Some slides from Mohammad Hammoud, Dong Wang

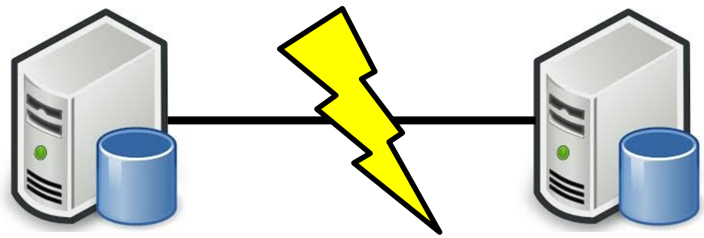
The CAP Theorem

- 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 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 as changes in place cannot be propagated when the system is portioned.

- 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

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

- With databases in 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*)

Types of Consistency

- Strong Consistency
  - After the update completes, **any subsequent access** will return the **same** updated value.
- Weak Consistency
  - It is **not guaranteed** that subsequent accesses will return the updated value.
- **Eventual Consistency**
  - Specific form of weak consistency
  - It is guaranteed that if **no new updates** are made to object, **eventually** all accesses will return the last updated value (e.g., *propagate updates to replicas in a lazy fashion*)

Eventual Consistency Variations

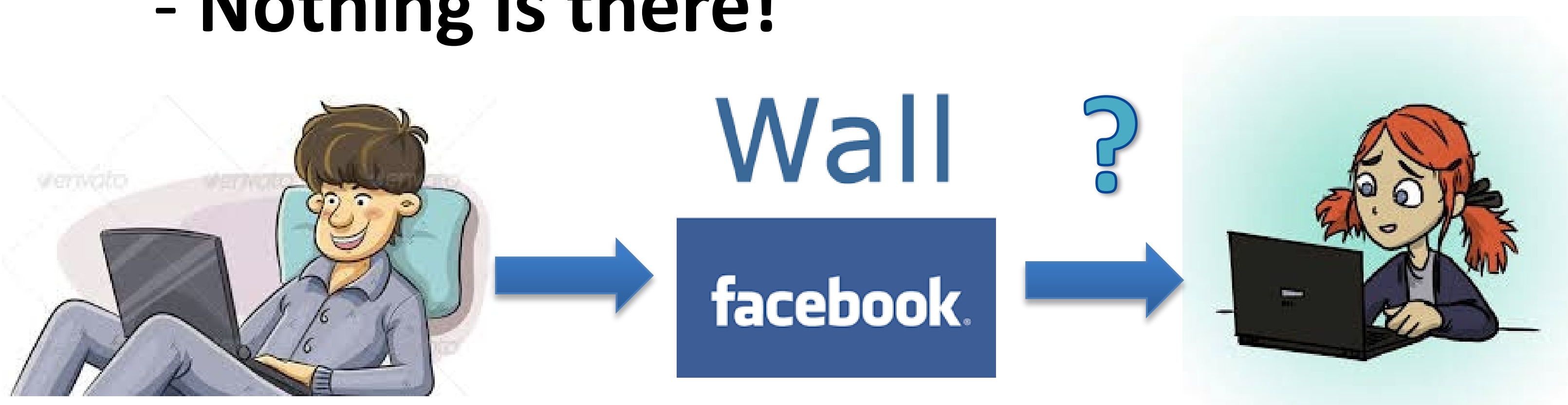
- Causal consistency
  - Processes that have causal relationship will see consistent data
- Read-your-write consistency
  - A process always accesses the data item after it's update operation and never sees an older value
- Session consistency
  - As long as session exists, system guarantees read-your-write consistency
  - Guarantees do not overlap sessions

Eventual Consistency Variations

- Monotonic read consistency
  - If a process has seen a particular value of data item, any subsequent processes will never return any previous values
- Monotonic write consistency
  - The system guarantees to serialize the writes by the *same* process
- In practice
  - A number of these properties can be combined
  - Monotonic reads and read-your-writes are most desirable

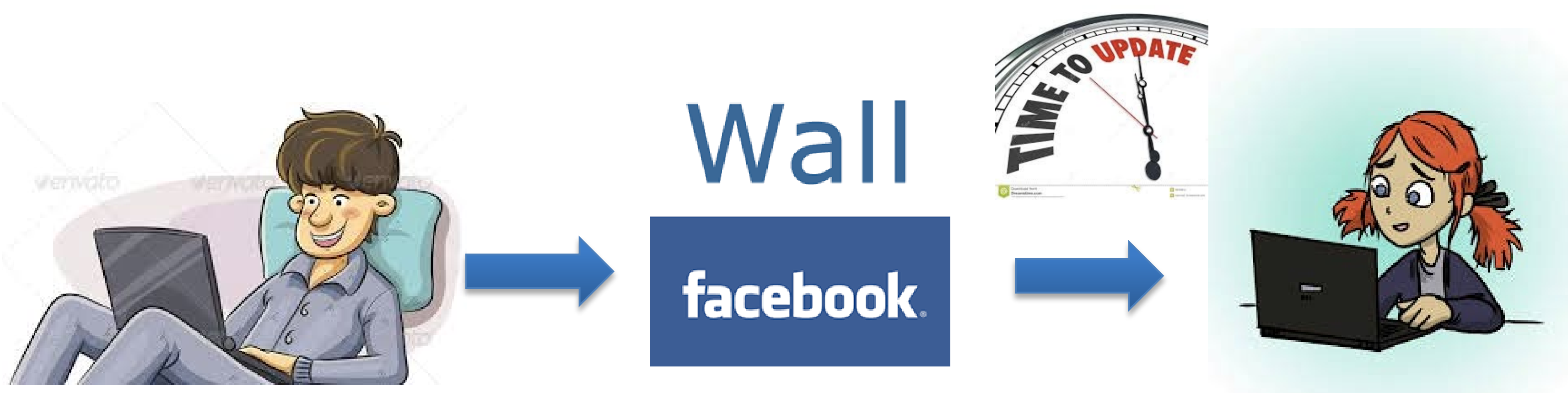
Eventual Consistency
- A Facebook Example

- Bob finds an interesting story and shares with Alice by posting on her Facebook wall
- Bob asks Alice to check it out
- Alice logs in her account, checks her Facebook wall but finds:
  - **Nothing is there!**



Eventual Consistency
- A Facebook Example

- Bob tells Alice to wait a bit and check out later
- Alice waits for a minute or so and checks back:
    - **She finds the story Bob shared with her!**



Eventual Consistency - A Facebook Example

- Reason: it is possible because Facebook uses an **eventual consistent model**
- Why Facebook chooses eventual consistent model over the strong consistent one?
  - Facebook has more than 1 billion active users
  - It is non-trivial to efficiently and reliably store the huge amount of data generated at any given time
  - Eventual consistent model offers the option to **reduce the load and improve availability**

Eventual Consistency
- A Dropbox Example

- Dropbox enabled immediate consistency via synchronization in many cases.
- However, what happens in case of a network partition?

Eventual Consistency
- A Dropbox Example

- Let's do a simple experiment here:
  - Open a file in your drop box
  - Disable your network connection (e.g., WiFi, 4G)
  - Try to edit the file in the drop box: can you do that?
  - Re-enable your network connection: what happens to your dropbox folder?

Eventual Consistency
- A Dropbox Example

- Dropbox embraces eventual consistency:
  - Immediate consistency is impossible in case of a network partition
  - Users will feel bad if their word documents freeze each time they hit Ctrl+S , simply due to the large latency to update all devices across WAN
  - Dropbox is oriented to **personal syncing**, not on collaboration, so it is not a real limitation.

Eventual Consistency
- An ATM Example

- In design of automated teller machine (ATM):
  - Strong consistency appear to be a nature choice
  - However, in practice, **A beats C**
  - Higher availability means **higher revenue**
  - ATM will allow you to withdraw money *even if the machine is partitioned from the network*
  - However, it puts **a limit** on the amount of withdraw (e.g., $200)
  - The bank might also charge you a fee when a overdraft happens

Dynamic Tradeoff between **C** and **A**

- An airline reservation system:
  - When most of seats are available: it is ok to rely on somewhat out-of-date data, availability is more critical
  - When the plane is close to be filled: it needs more accurate data to ensure the plane is not overbooked, consistency is more critical

Heterogeneity: Segmenting **C** and **A**

- No single uniform requirement
  - Some aspects require strong consistency
  - Others require high availability
- Segment the system into different components
  - Each provides different types of guarantees
- Overall guarantees neither consistency nor availability
  - Each part of the service gets exactly what it needs
- Can be partitioned along different dimensions

Partitioning Examples

Data Partitioning

- Different data may require different consistency and availability
- Example:
  - Shopping cart: high availability, responsive, can sometimes suffer anomalies
  - Product information need to be available, slight variation in inventory is sufferable
  - Checkout, billing, shipping records must be consistent

What if there are no partitions?

- Tradeoff between **Consistency** and **Latency**:
- Caused by the **possibility of failure** in distributed systems
  - High availability -> replicate data -> consistency problem
- Basic idea:
  - Availability and latency are arguably **the same thing**: unavailable -> extreme high latency
  - Achieving different levels of consistency/availability takes different amount of time

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*

Loose Consistency

Strict Consistency

Easier to implement, and is efficient

Generally hard to implement, and is inefficient

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

CAP -> PACELC

- A more complete description of the space of potential tradeoffs for distributed system:
  - If there is a **partition (P)**, how does the system trade off **availability and consistency (A and C)**; **else (E)**, when the system is running normally in the absence of partitions, how does the system trade off **latency (L) and consistency (C)**?

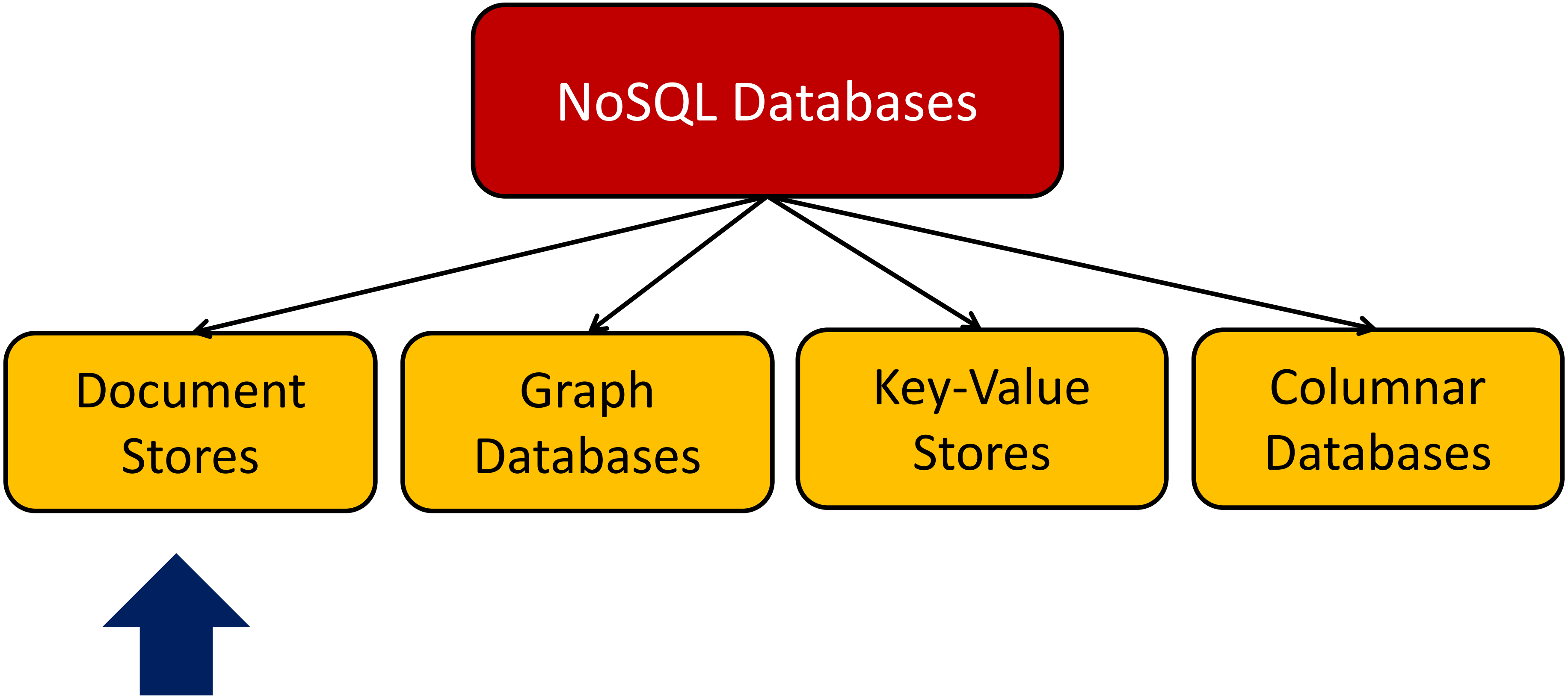
Abadi, Daniel J. "Consistency tradeoffs in modern distributed database system design." Computer-IEEE Computer Magazine 45.2 (2012): 37.

Examples

- **PA/EL Systems:** Give up both Cs for availability and lower latency
  - Dynamo, Cassandra, Riak
- **PC/EC Systems:** Refuse to give up consistency and pay the cost of availability and latency
  - BigTable, Hbase, VoltDB/H-Store
- **PA/EC Systems:** Give up consistency when a partition happens and keep consistency in normal operations
  - MongoDB
- **PC/EL System:** Keep consistency if a partition occurs but gives up consistency for latency in normal operations
  - Yahoo! PNUTS

Types of NoSQL Databases

- Here is a limited taxonomy of NoSQL databases:

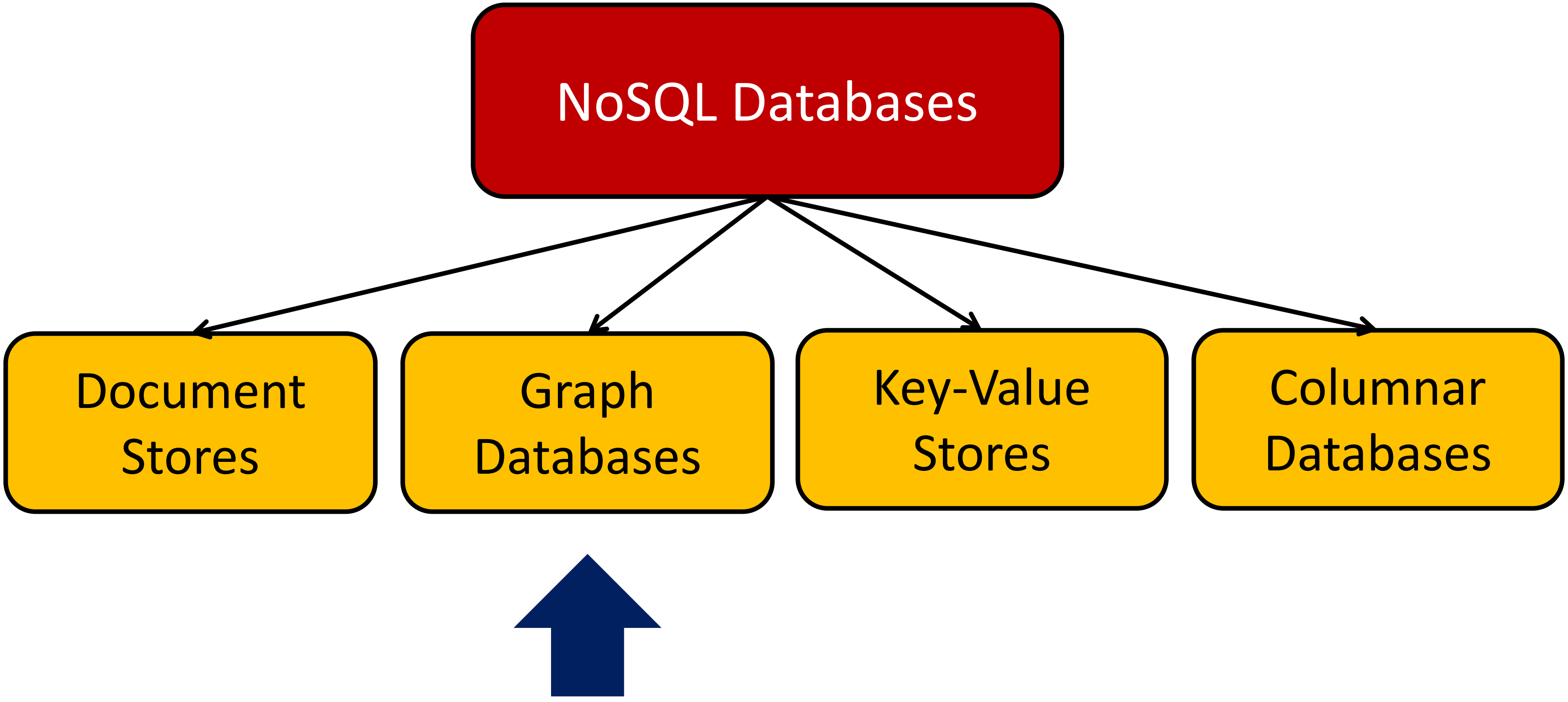


Document Stores

- 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 (both can be queried using MapReduce)

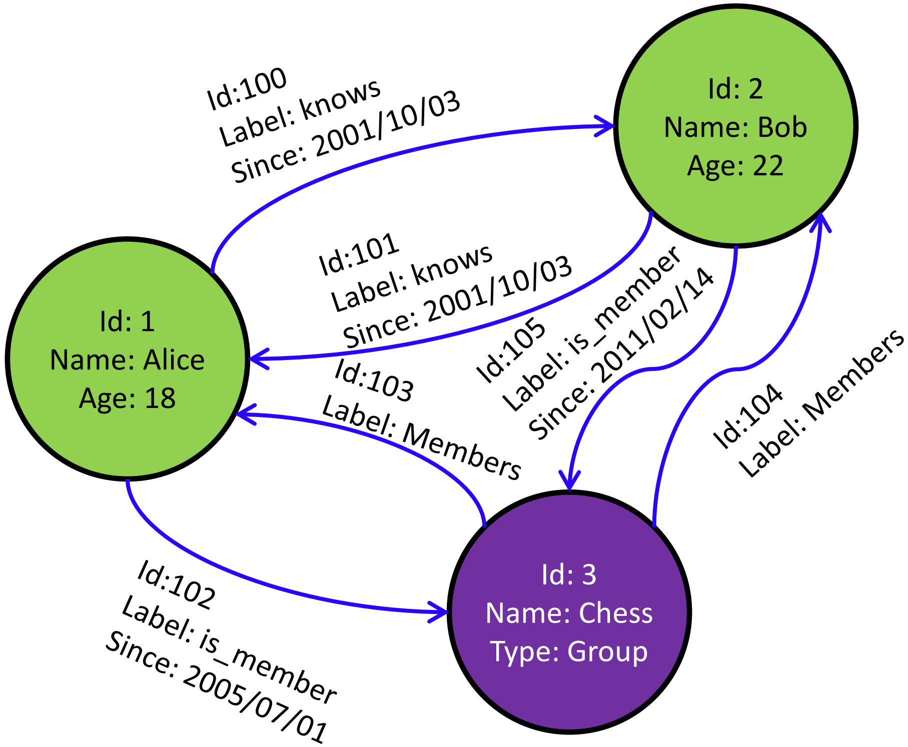
Types of NoSQL Databases

- Here is a limited taxonomy of NoSQL databases:



Graph Databases

- Data are represented as vertices and edges

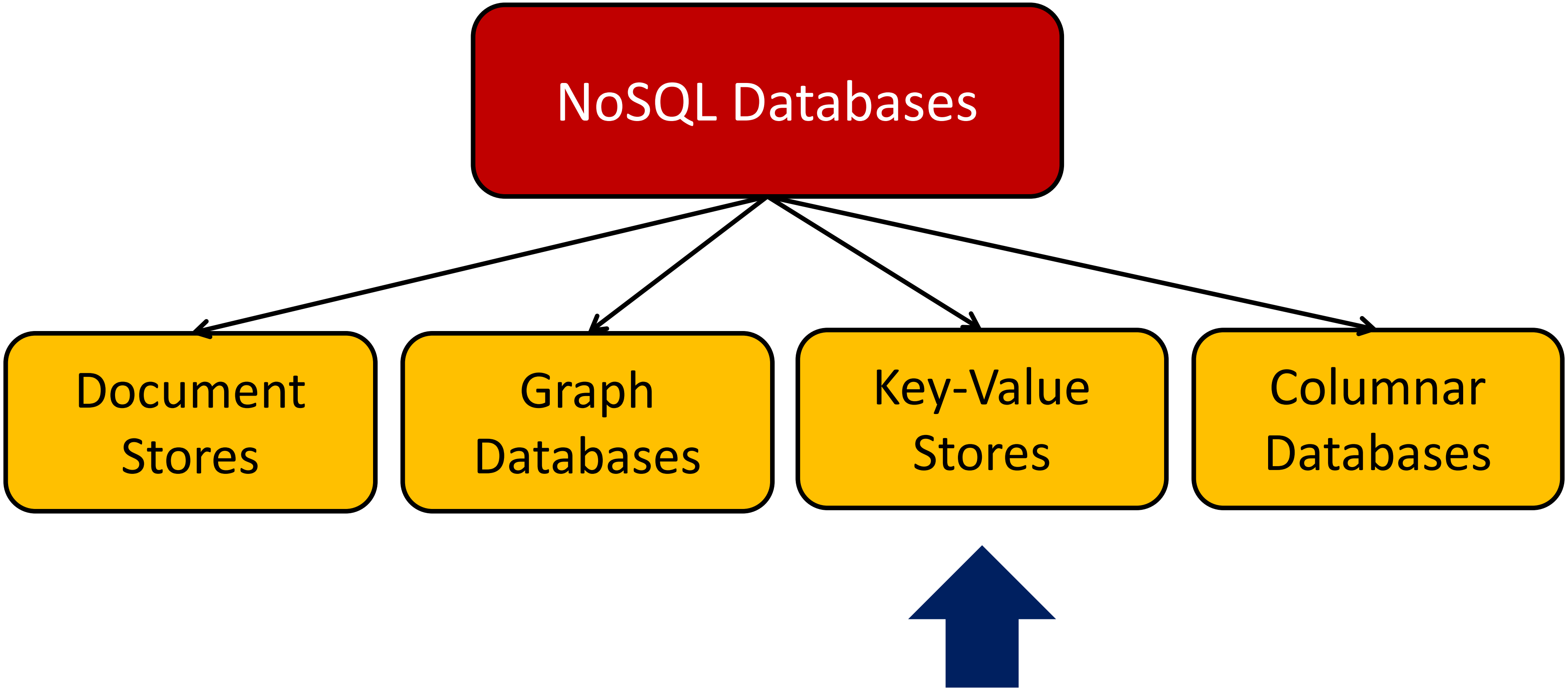


- Graph databases are powerful for graph-like queries (e.g., find the shortest path between two elements)

- E.g., Neo4j and VertexDB

Types of NoSQL Databases

- Here is a limited taxonomy of NoSQL databases:



Key-Value Stores

- Keys are mapped to (possibly) more complex value (e.g., lists)

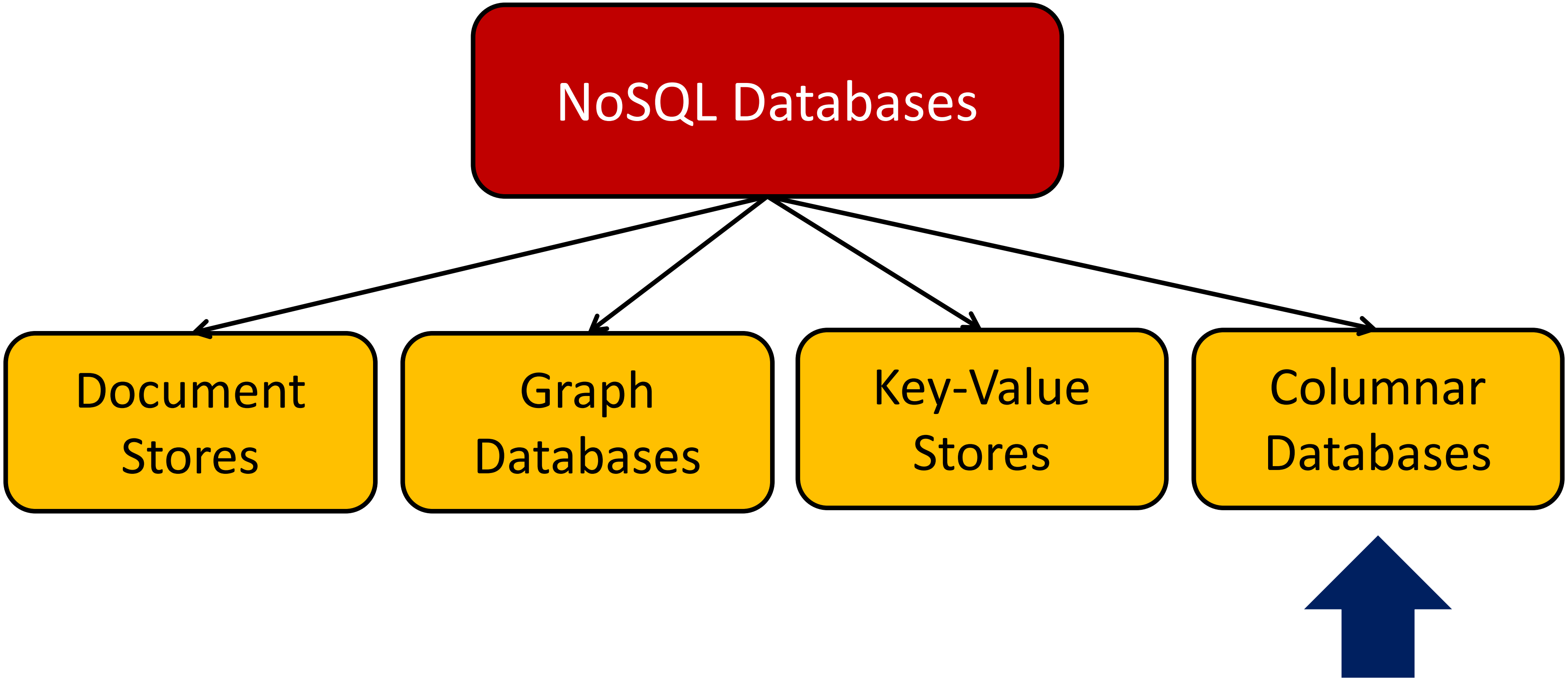
- Keys can be stored in a hash table and can be distributed easily

- 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

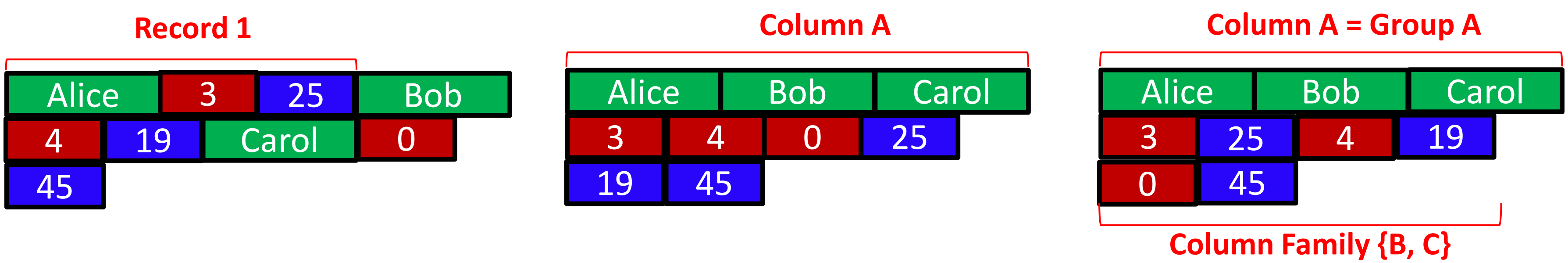
Types of NoSQL Databases

- Here is a limited taxonomy of NoSQL databases:



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

Summary

- The *CAP theorem* states that any distributed database with shared data can have at most two of the three desirable properties:
  - **C**onsistency
  - **A**vailability
  - **P**artition Tolerance

- The CAP theorem leads to various designs of databases with *relaxed* ACID guarantees

Summary (*Cont'd*)

- *NoSQL* (or *Not-Only-SQL*) databases follow the *BASE properties*:
  - **B**asically **A**vailable
  - **S**oft-State
  - **E**ventual Consistency