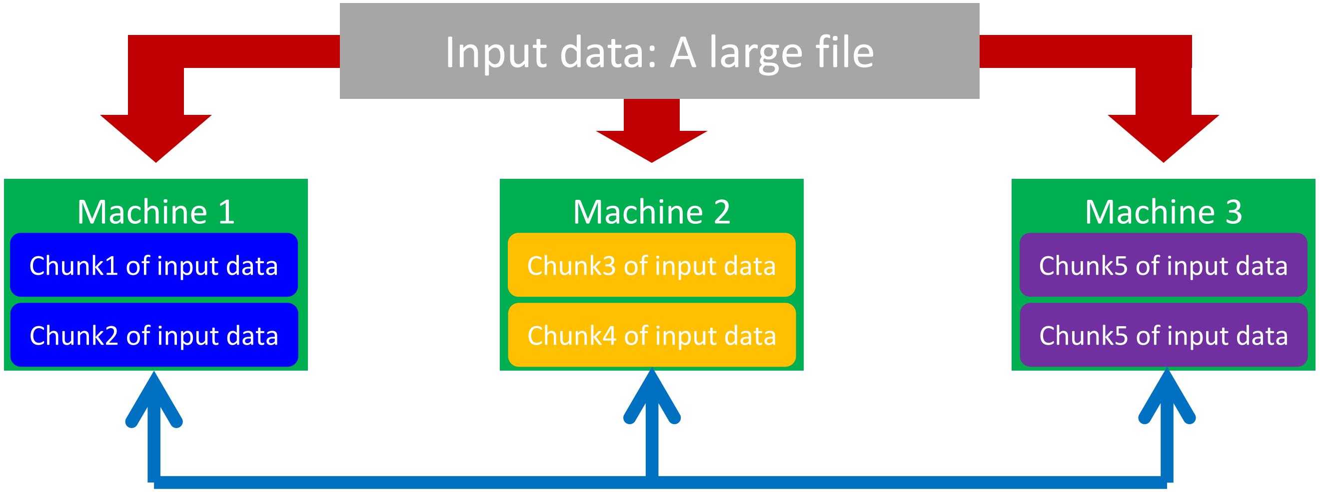
NoSQL Databases

Scaling Traditional Databases

- Traditional RDBMSs can be either scaled:
  - Vertically (or Up)
    - Can be achieved by hardware upgrades (e.g., faster CPU, more memory, or larger disk)
    - Limited by the amount of CPU, RAM and disk that can be configured on a single machine
  - Horizontally (or Out)
    - Can be achieved by adding more machines
    - Requires database *sharding* and probably *replication*
    - Limited by the Read-to-Write ratio and communication overhead

Why Sharding Data?

- Data is typically *sharded* (or *striped*) to allow for concurrent/parallel accesses



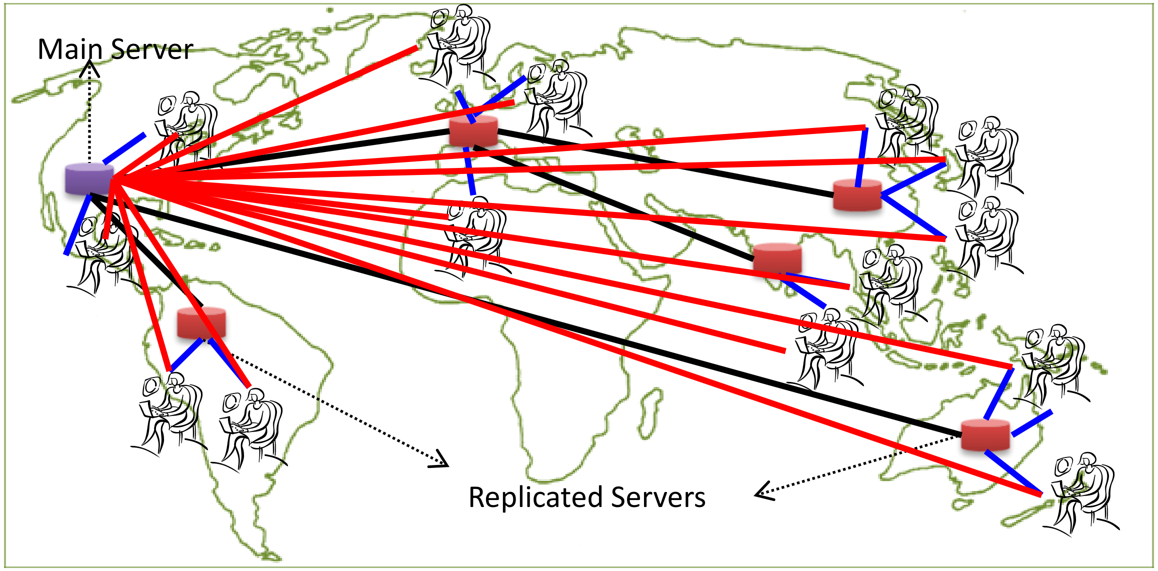
E.g., Chunks 1, 3 and 5 can be accessed in parallel

Why Replicating Data?

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

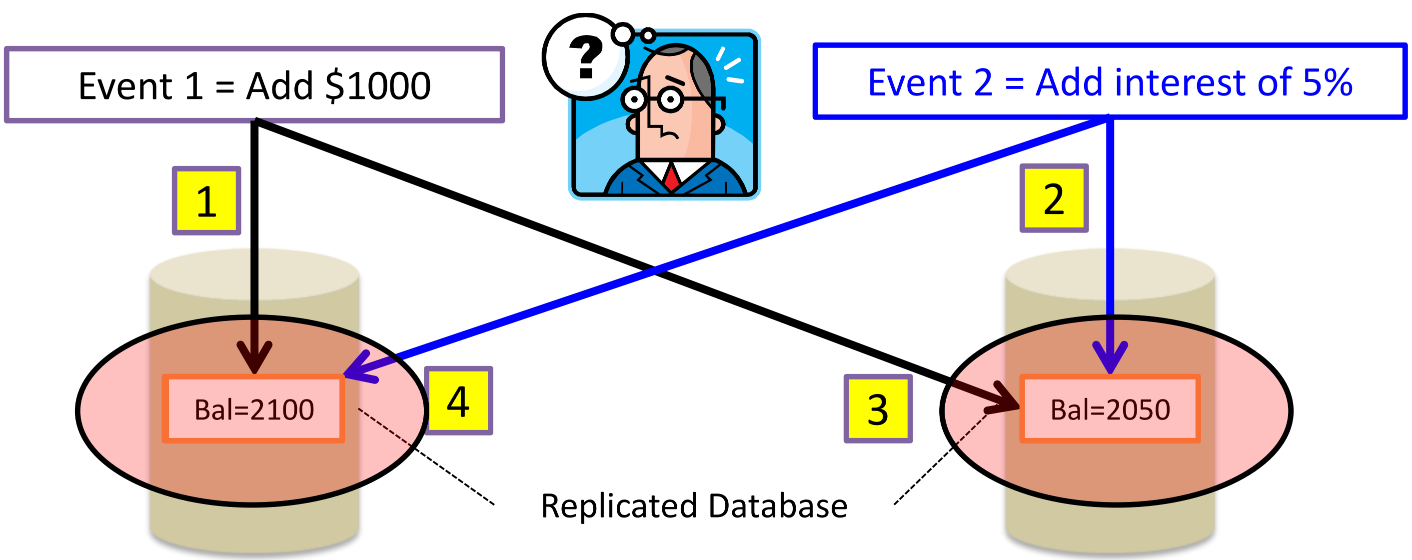
Why Replicating Data?

- 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



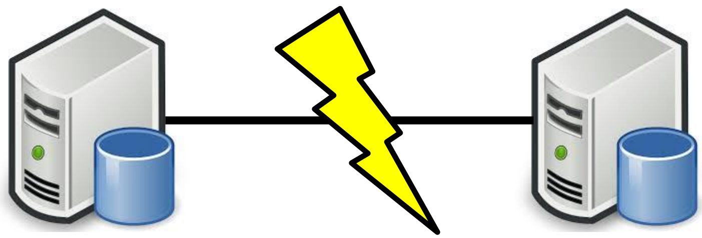
The CAP Theorem

- 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

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*)

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

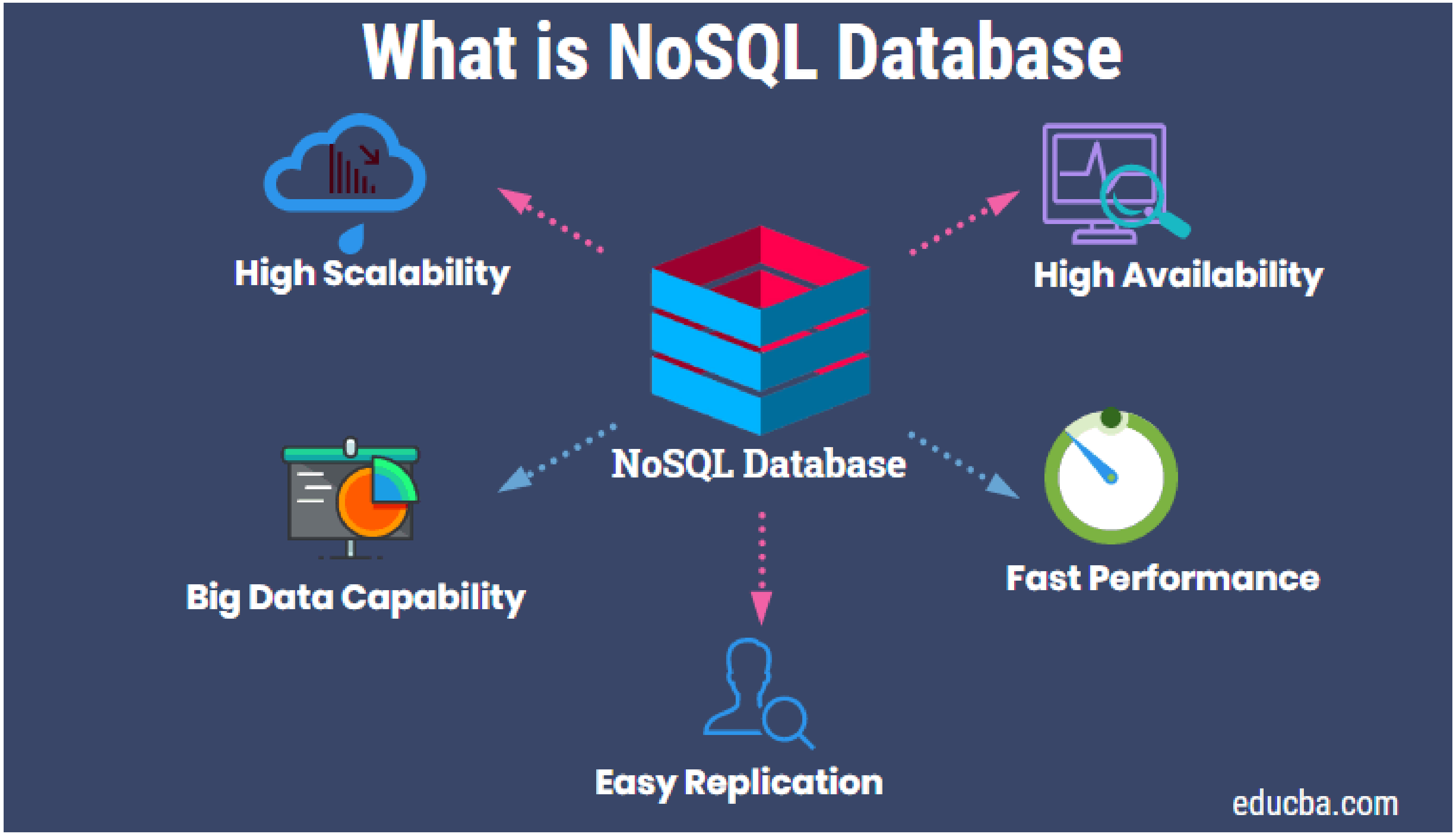
Eventual Consistency

- A database is termed as *Eventually Consistent* if:
  - All replicas will *gradually* become consistent in the absence of updates

NoSQL Databases

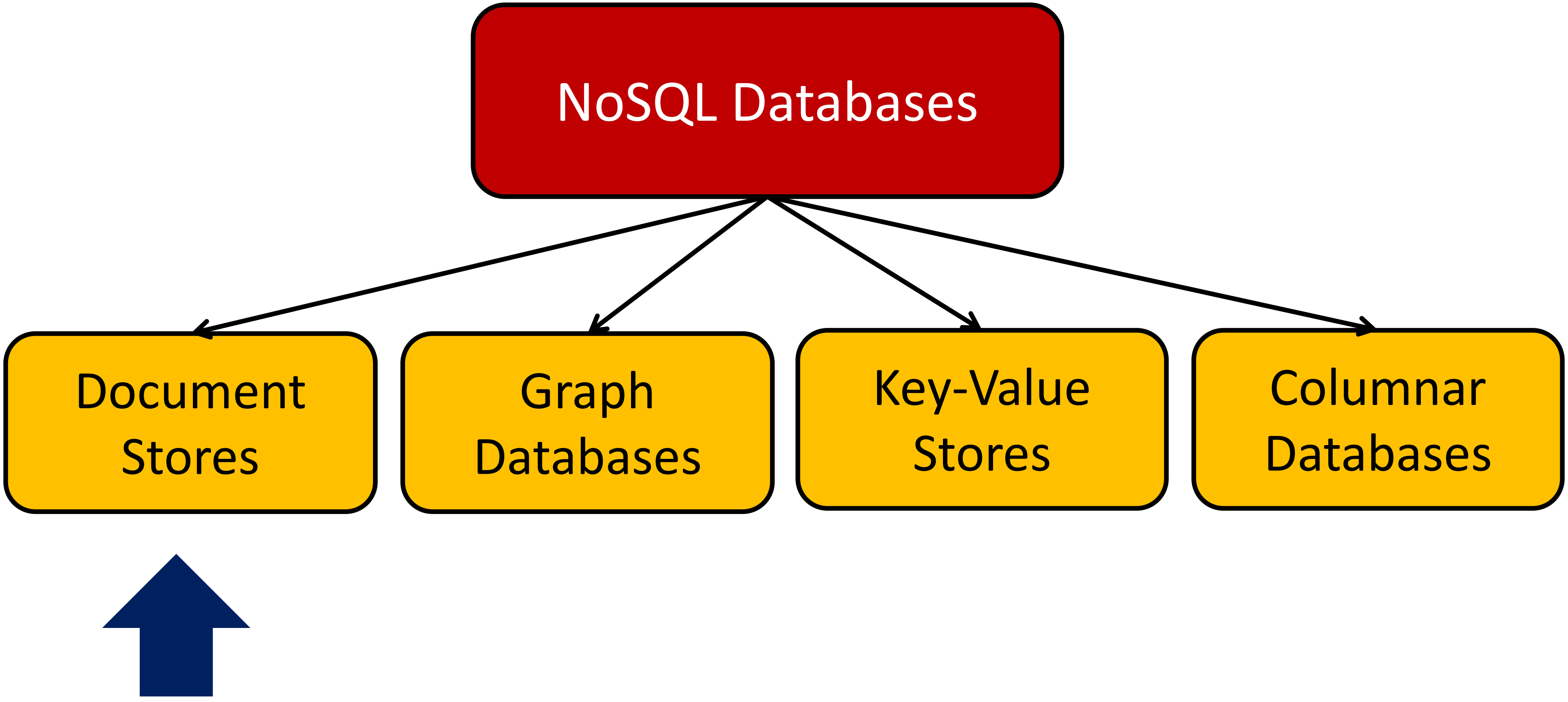
- To this end, a new class of databases emerged, which mainly follow the BASE properties
  - These were dubbed as NoSQL databases
  - E.g., Amazon’s Dynamo and Google’s Bigtable

- Main characteristics of NoSQL databases include:
  - No strict schema requirements
  - No strict adherence to ACID properties
  - Consistency is traded in favor of Availability


What is NoSQL Database
High Scalability
High Availability
NoSQL Database
Big Data Capability
Fast Performance
Easy Replication
educba.com

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

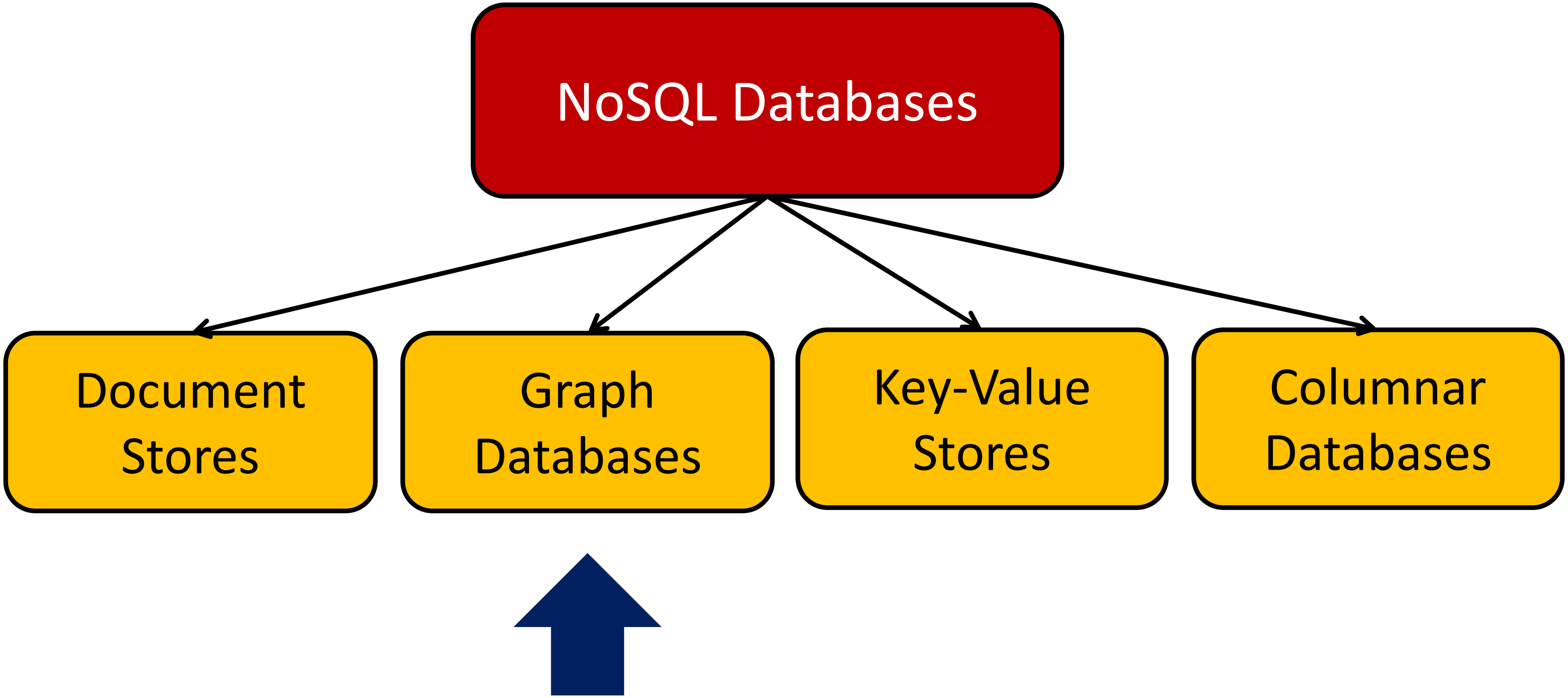
- E.g., MongoDB and CouchDB (both can be queried using MapReduce)

Document Databases, JSON

```
{
    _id: ObjectId("51156a1e056d6f966f268f81"),
    type: "Article",
    author: "Derick Rethans",
    title: "Introduction to Document Databases with MongoDB",
    date: ISODate("2013-04-24T16:26:31.911Z"),
    body: "This arti…"
},
{
    _id: ObjectId("51156a1e056d6f966f268f82"),
    type: "Book",
    author: "Derick Rethans",
    title: "php|architect's Guide to Date and Time Programming with PHP",
    isbn: "978-0-9738621-5-7"
}
```

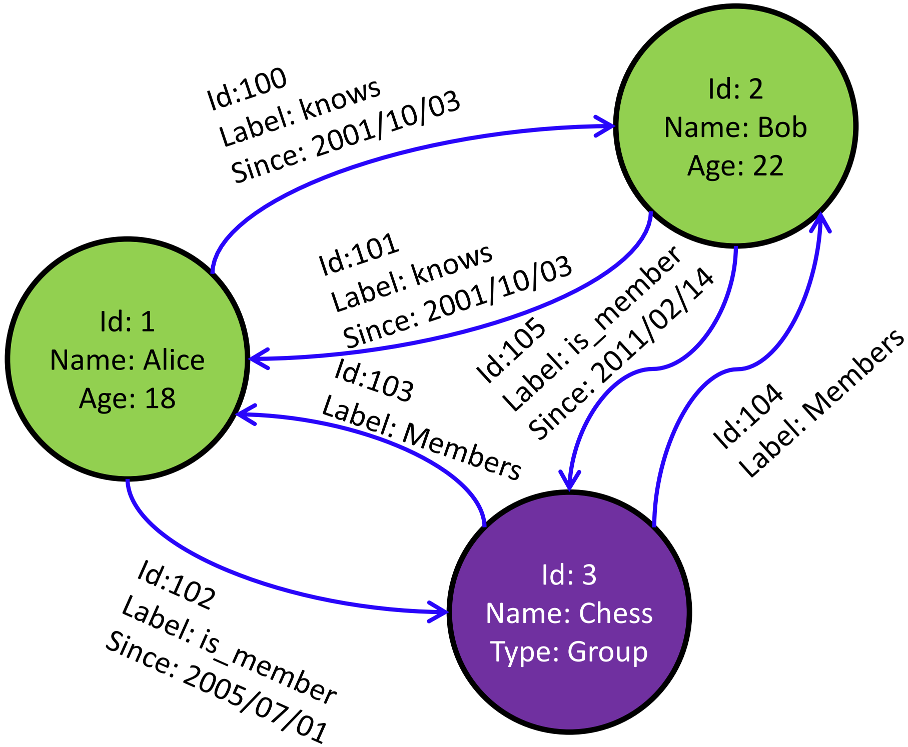
Types of NoSQL Databases

- Here is a limited taxonomy of NoSQL databases:



Graph Databases

- Data are represented as vertices and edges

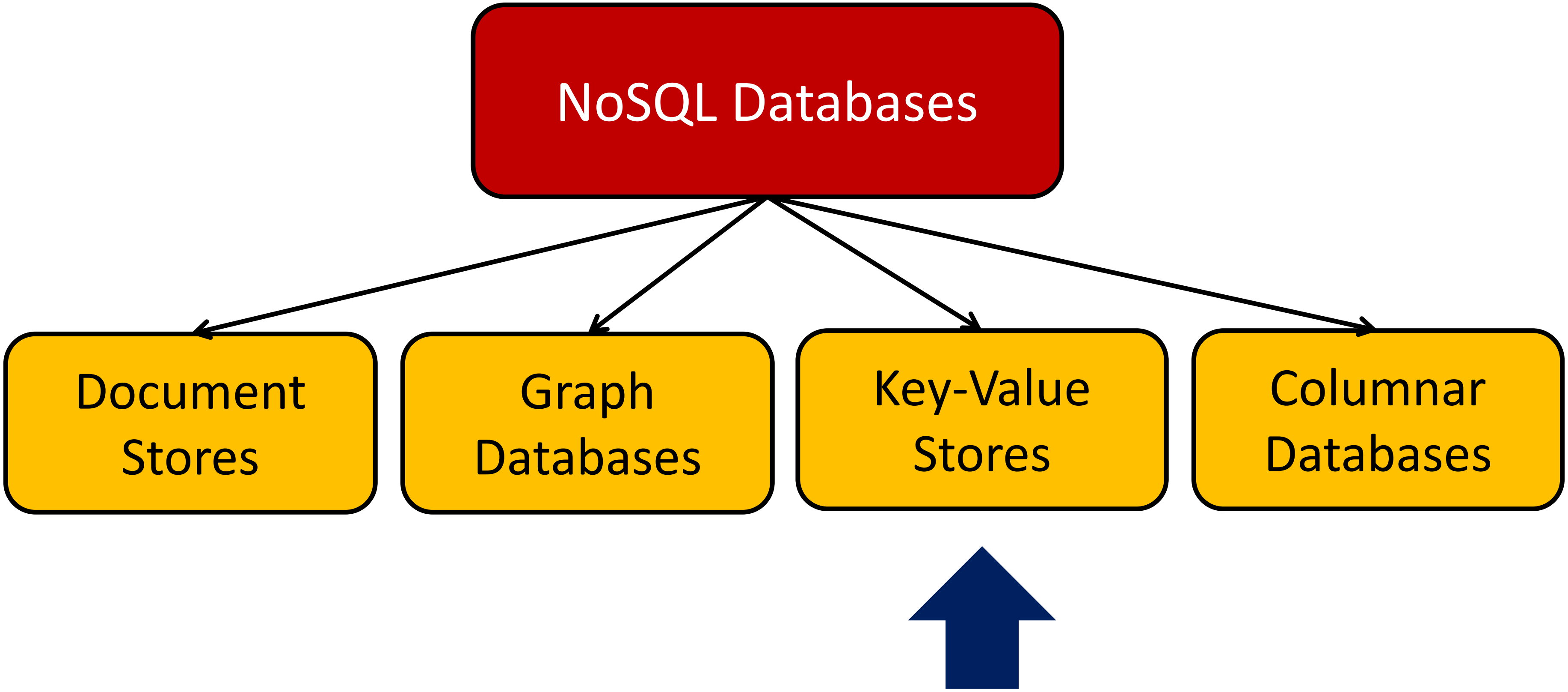


- Graph databases are powerful for graph-like queries (e.g., find the shortest path between two elements)

- E.g., Neo4j and VertexDB

Types of NoSQL Databases

- Here is a limited taxonomy of NoSQL databases:

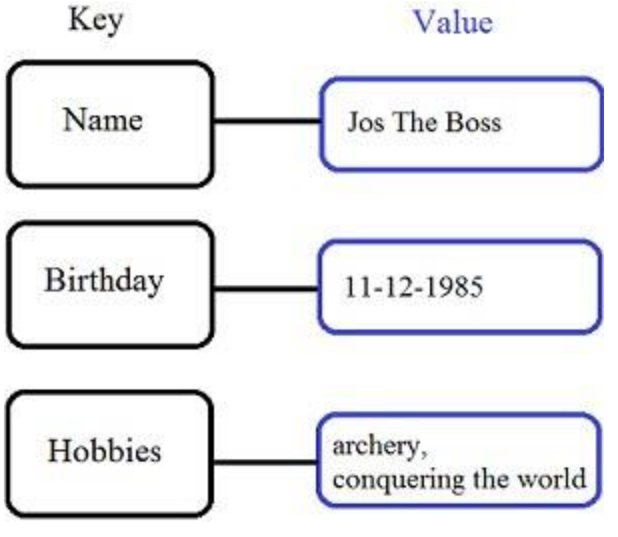


Key-Value Stores

- 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

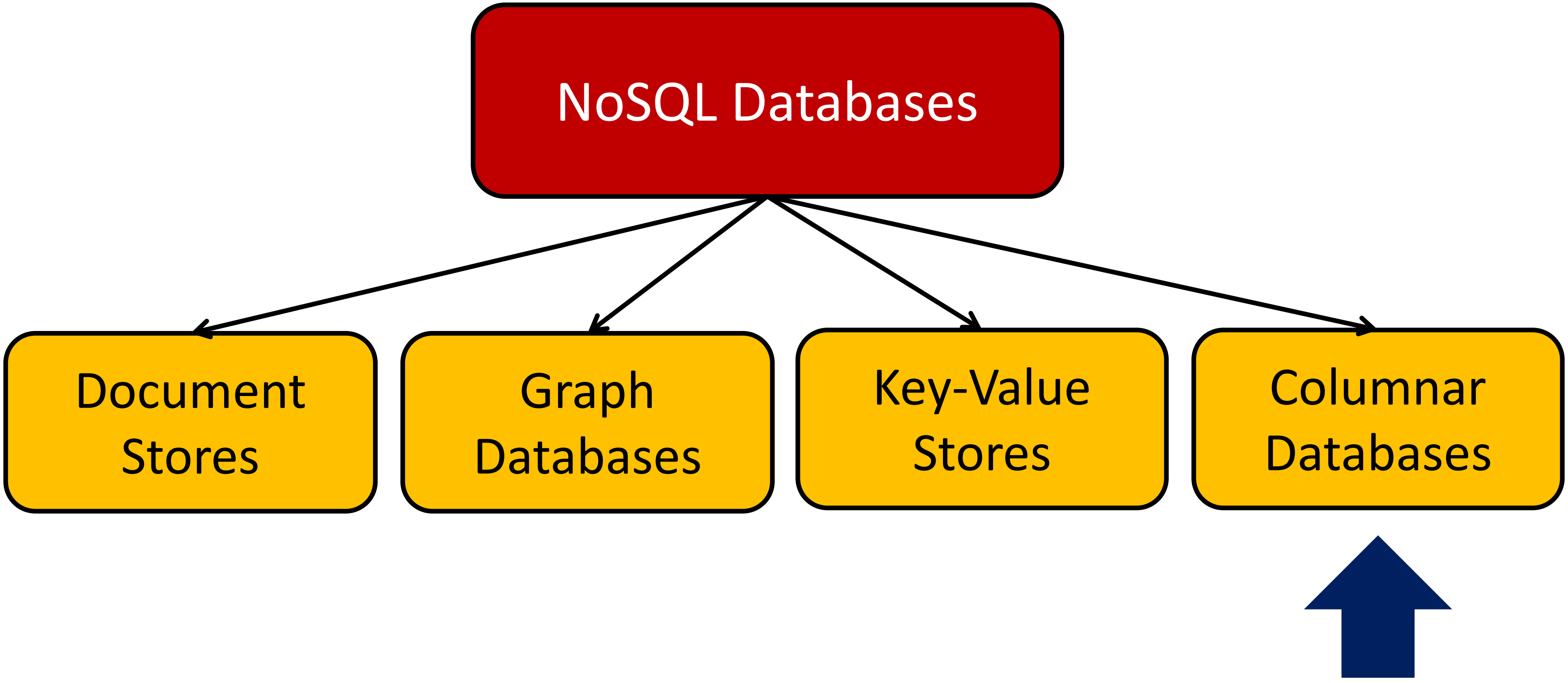
Key/Value stores

- Store data in a schema-less way
- Store data as maps
  - Provide a very efficient average running time algorithm for accessing data
- Notable for:
  - Couchbase
  - Redis
  - Amazon Dynamodb



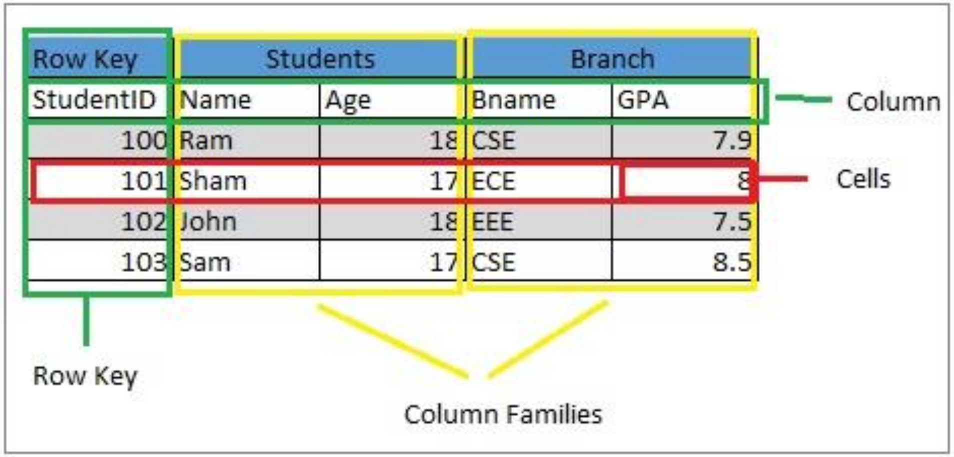
Types of NoSQL Databases

- Here is a limited taxonomy of NoSQL databases:



Column-Oriented Stores

- Data are stored in a column-oriented way
  - Data efficiently stored
  - Avoids consuming space for storing nulls
  - Columns are grouped in column-families
  - Data isn't stored as a single table but is stored by column families
  - Unit of data is a set of key/value pairs
    - Identified by "row-key"
    - Ordered and sorted based on row-key
- Notable for:
  - Google's Bigtable (used in all Google's services)
  - HBase (Facebook, StumbleUpon, Hulu, Yahoo!, ...)



Where would I use it?

- Where would I use a NoSQL database?
- Do you have somewhere a large set of uncontrolled, unstructured, data that you are trying to fit into a RDBMS?
  - Log Analysis
  - Social Networking Feeds (many firms hooked in through Facebook or Twitter)
  - External feeds from partners
  - Data that is not easily analyzed in a RDBMS such as time-based data
  - Large data feeds that need to be massaged before entry into an RDBMS

Summary

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

- NoSQL databases have different types:
  - Document Stores
  - Graph Databases
  - Key-Value Stores
  - Columnar Databases