

Database Applications (15-415)

NoSQL Databases

Lecture 23, April 21, 2014

Mohammad Hammoud

Today...

- Last Session:
 - Recovery Management (*finished*)
- Today's Session:
 - NoSQL databases
- Announcements:
 - Final Exam is on Sunday April 27th, at 9:00AM in room 2051 (*all materials are included- open book, open notes*)
 - We will hold a “review session” (for the final exam) on Thursday, April 24th during the recitation
 - PS5 (the “last” assignment) is due on Thursday, April 24th, by midnight

Outline



Types of Data



Scaling Databases & the 2PC Protocol

The CAP Theorem and the BASE Properties

NoSQL Databases

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
- Note: 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:

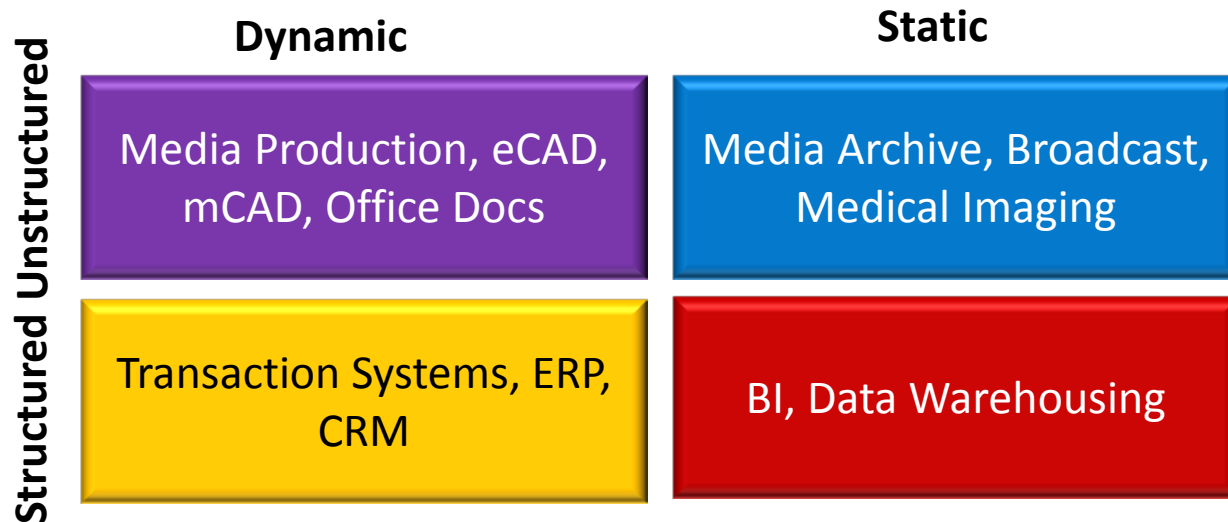
- Data that changes relatively frequently
- E.g., office documents and transactional entries in a financial database

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



- Relational databases are usually used for structured data
- File systems or *NoSQL databases* can be used for (static), unstructured data (*more on these later*)

Outline



Types of Data

Scaling Databases & the 2PC Protocol



The CAP Theorem and the BASE Properties

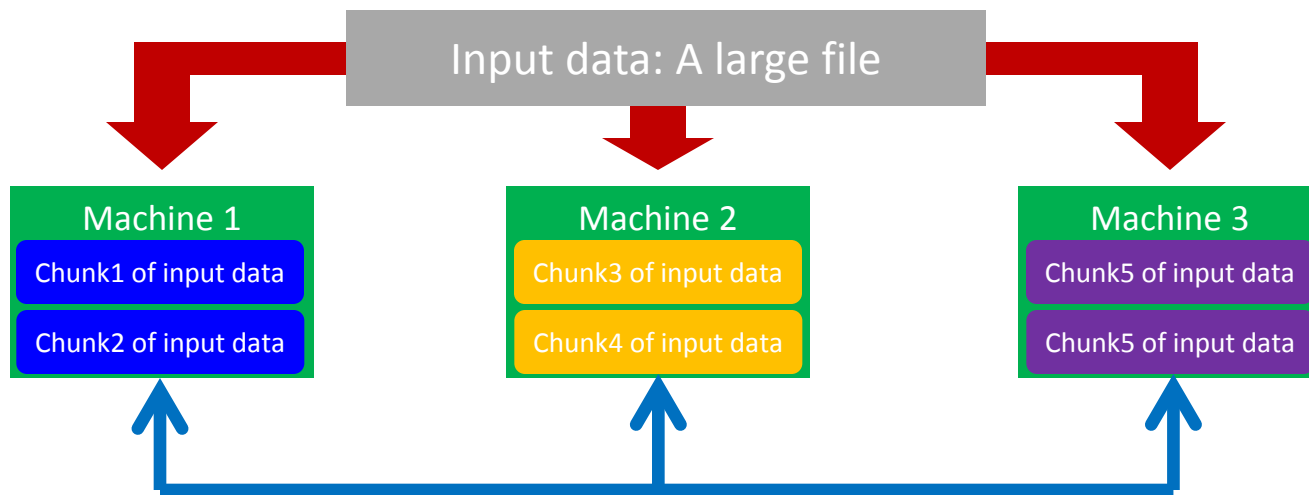
NoSQL Databases

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

Amdahl's Law

- How much faster will a parallel program run?
 - Suppose that the sequential execution of a program takes T_1 time units and the parallel execution on p processors/machines takes T_p time units
 - Suppose that out of the entire execution of the program, s fraction of it is not parallelizable while $1-s$ fraction is parallelizable
 - Then the speedup (**Amdahl's formula**):

$$\frac{T_1}{T_p} = \frac{T_1}{(T_1 \times s + T_1 \times \frac{1-s}{p})} = \frac{1}{s + \frac{1-s}{p}}$$

Amdahl's Law: An Example

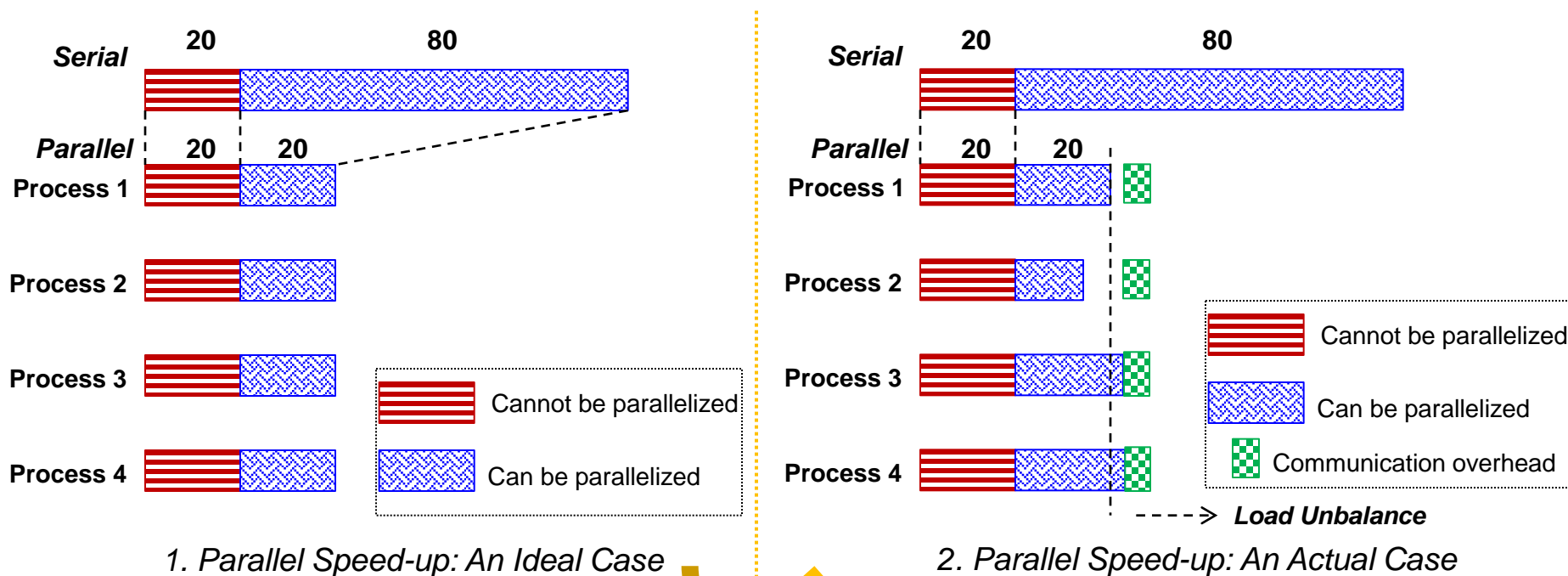
- Suppose that:
 - 80% of your program can be parallelized
 - 4 machines are used to run your parallel version of the program
- The speedup you can get according to Amdahl's law is:

$$\frac{1}{s + \frac{1-s}{p}} = \frac{1}{0.2 + \frac{0.8}{4}} = 2.5 \text{ times}$$

Although you use 4 processors you cannot get a speedup more than 2.5 times!

Real Vs. Actual Cases

- Amdahl's argument is too simplified
- In reality, communication overhead and potential workload imbalance exist upon running parallel programs



Some Guidelines

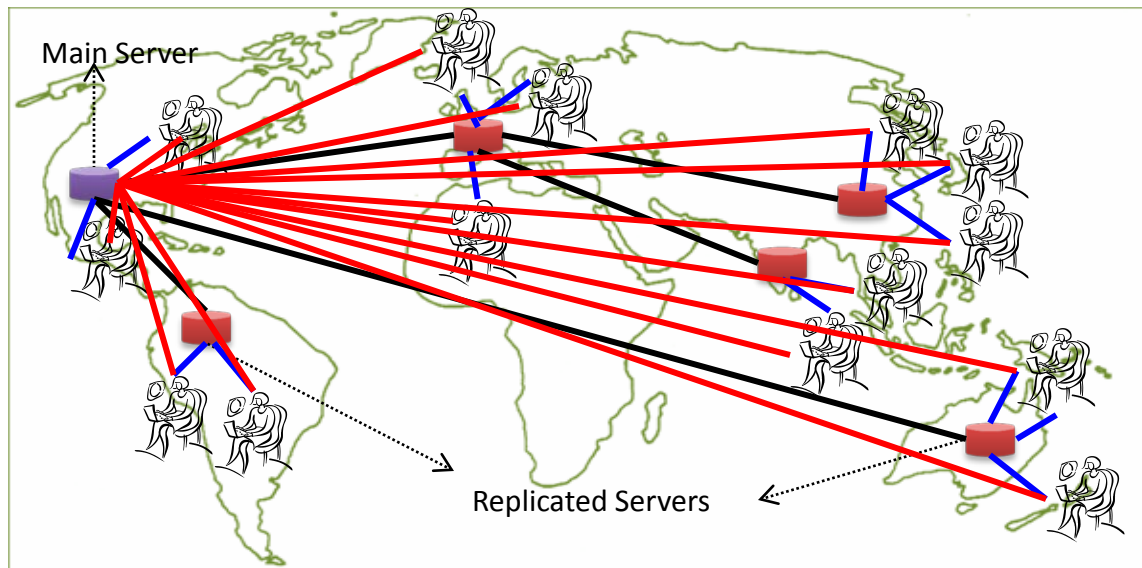
- Here are some guidelines to effectively benefit from parallelization:
 1. Maximize the fraction of your program that can be parallelized
 2. Balance the workload of parallel processes
 3. Minimize the time spent for communication

Why Replicating Data?

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

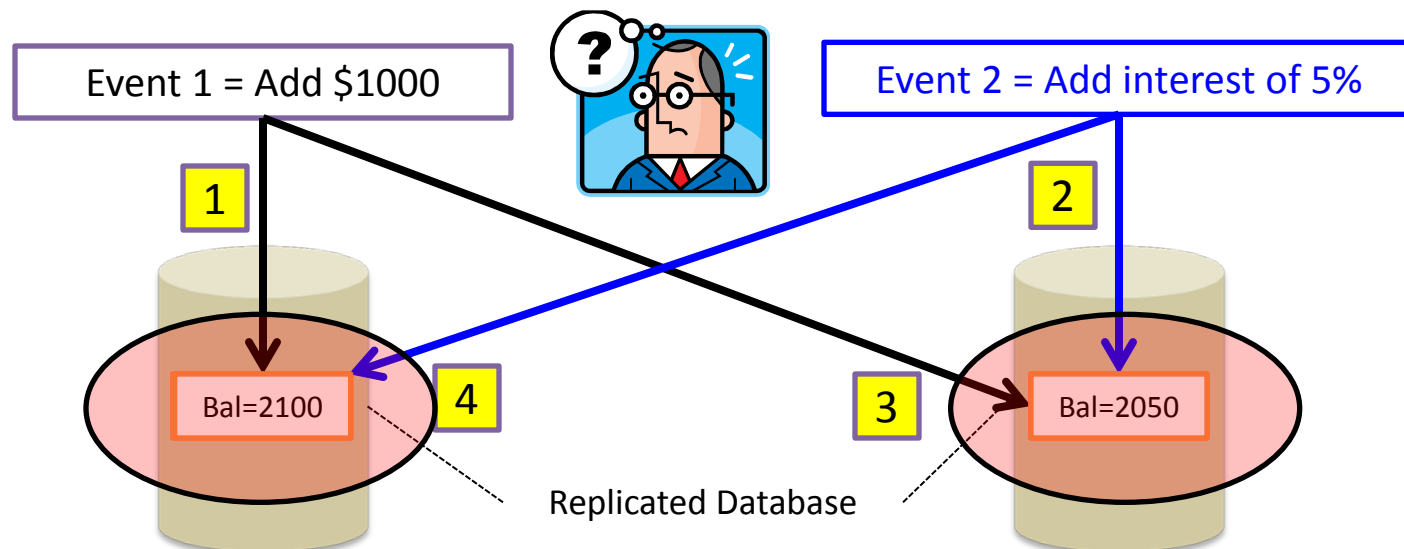
Why Replicating Data?

- 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

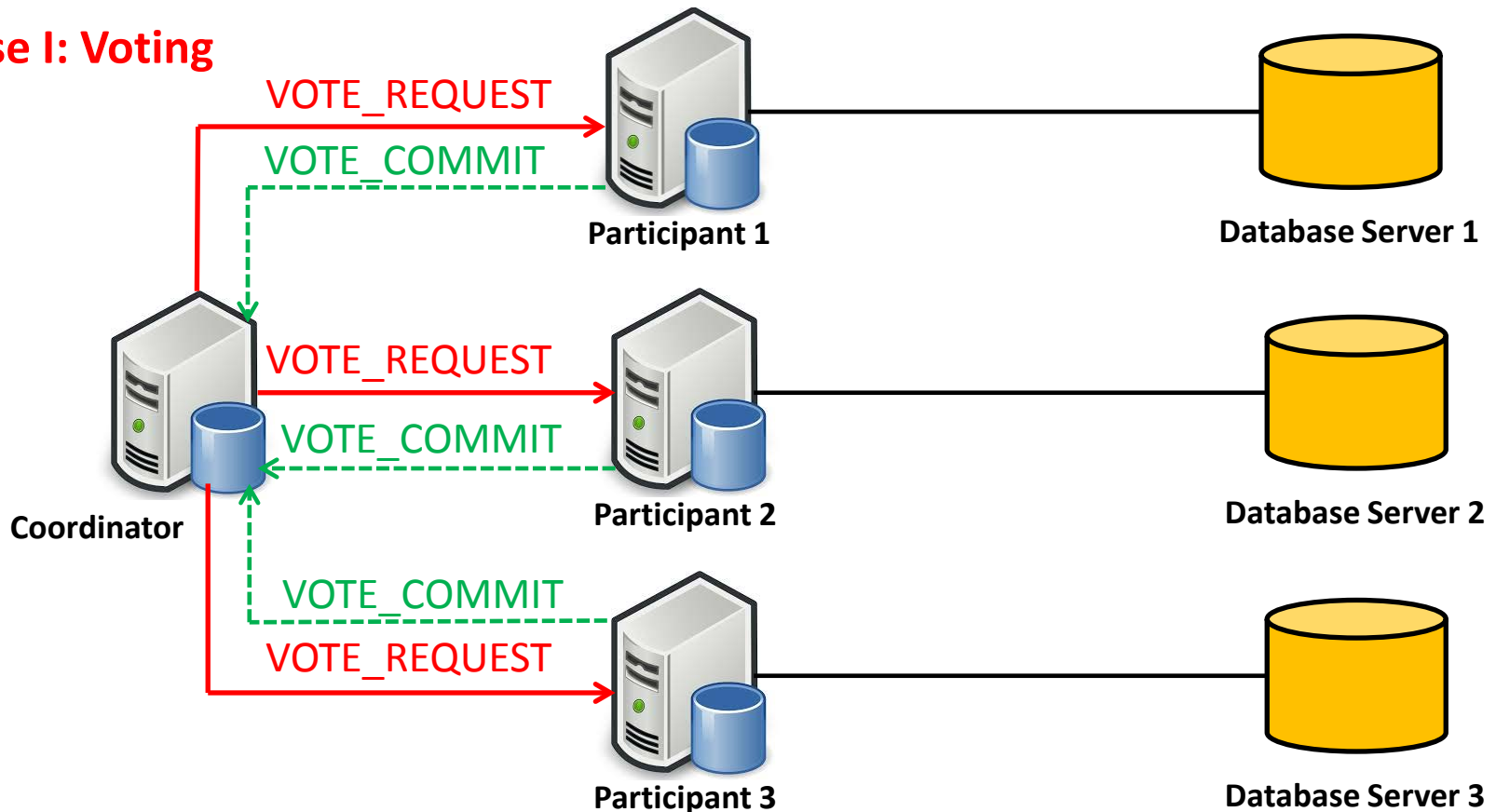
- An example:
 - In an e-commerce application, the bank database has been replicated across two servers
 - Maintaining consistency of replicated data is a challenge



The Two-Phase Commit Protocol

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

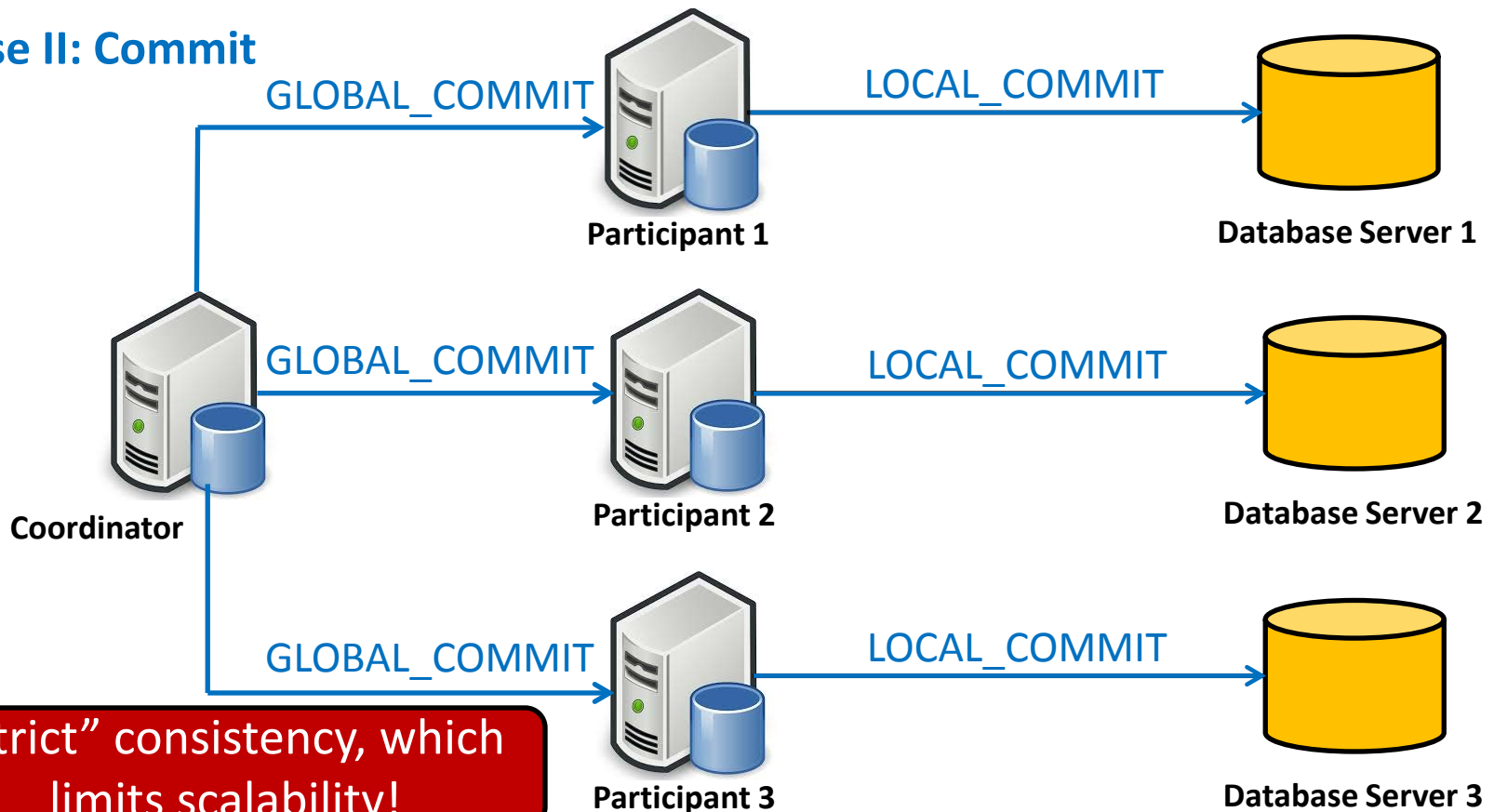
Phase I: Voting



The Two-Phase Commit Protocol

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

Phase II: Commit



Outline



Types of Data

Scaling Databases & the 2PC Protocol

The CAP Theorem and the BASE Properties

NoSQL Databases



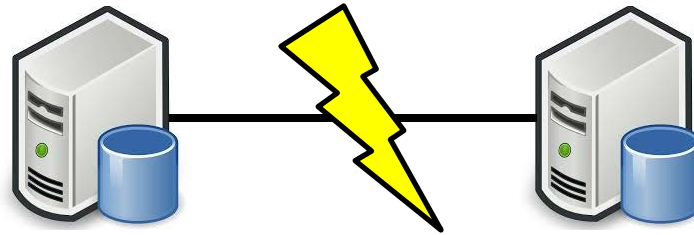
The CAP Theorem

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

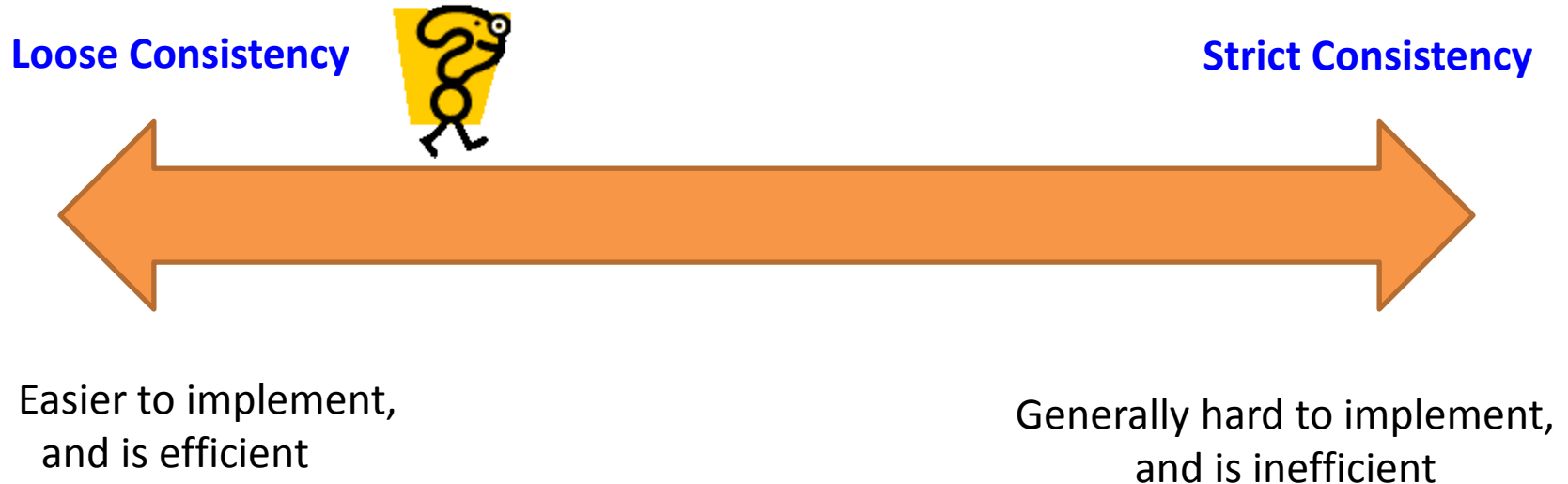
- When companies such as Google and Amazon were designing large-scale databases, 24/7 Availability was a key
 - A few minutes of downtime means lost revenue
- When *horizontally* scaling databases to 1000s of machines, the likelihood of a node or a network failure increases tremendously
- Therefore, in order to have strong guarantees on Availability and Partition Tolerance, they had to sacrifice “strict” Consistency (*implied by the CAP theorem*)

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

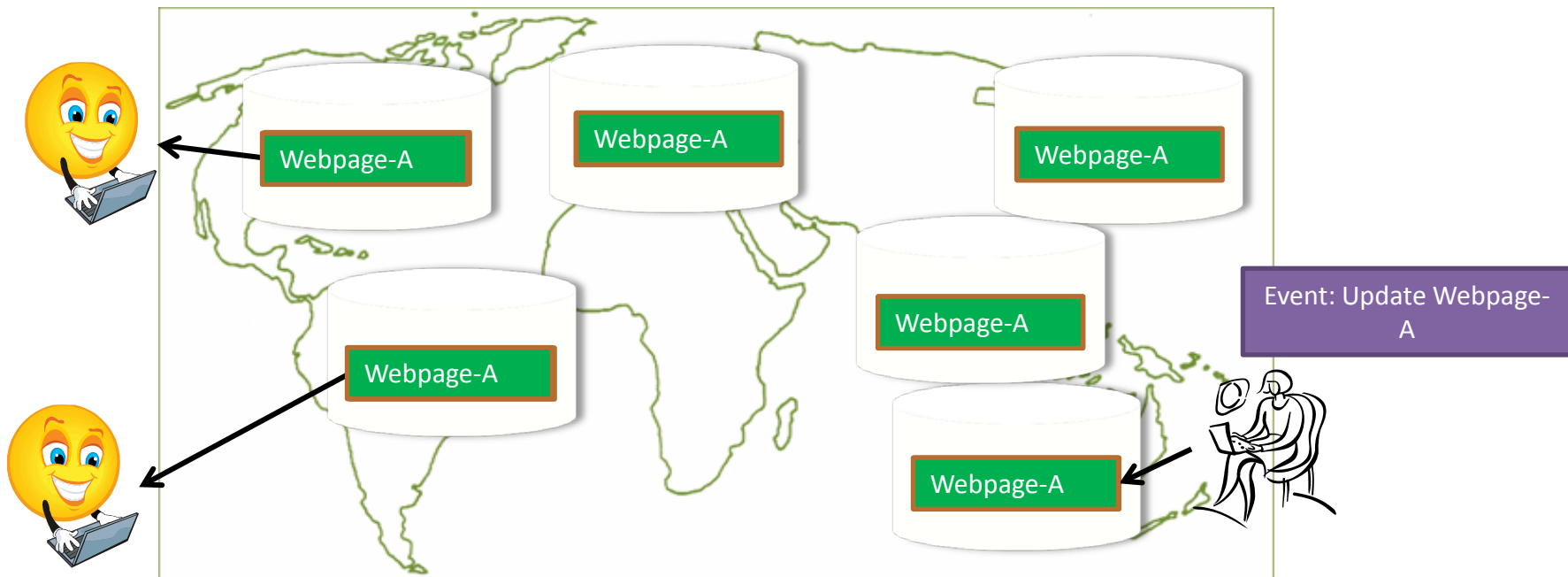
- The CAP theorem proves that it is impossible to guarantee strict Consistency and Availability while being able to tolerate network partitions
- This resulted in databases with relaxed ACID guarantees
- In particular, such databases apply the BASE properties:
 - Basically Available: the system guarantees Availability
 - Soft-State: the state of the system may change over time
 - Eventual 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

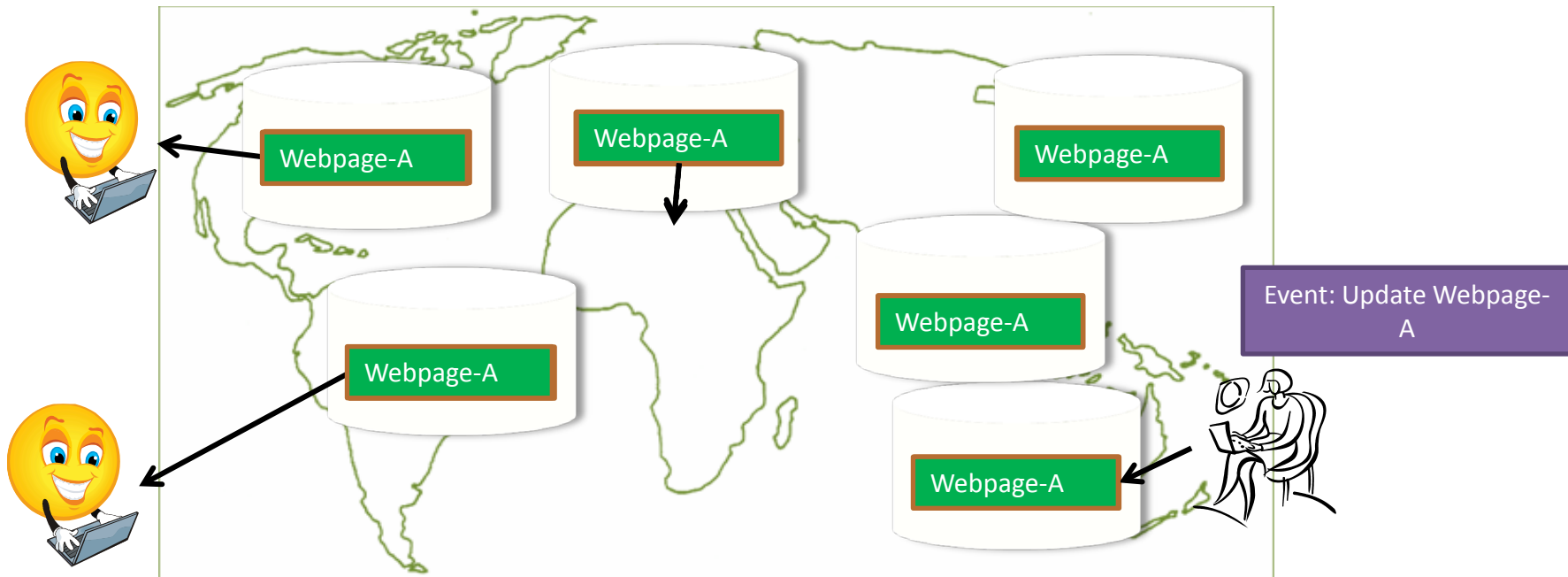
Eventual Consistency

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



Eventual Consistency: A Main Challenge

- But, what if the client accesses the data from different replicas?



Protocols like Read Your Own Writes (RYOW) can be applied!

Outline



Types of Data

Scaling Databases & the 2PC Protocol

The CAP Theorem and the BASE Properties

NoSQL Databases

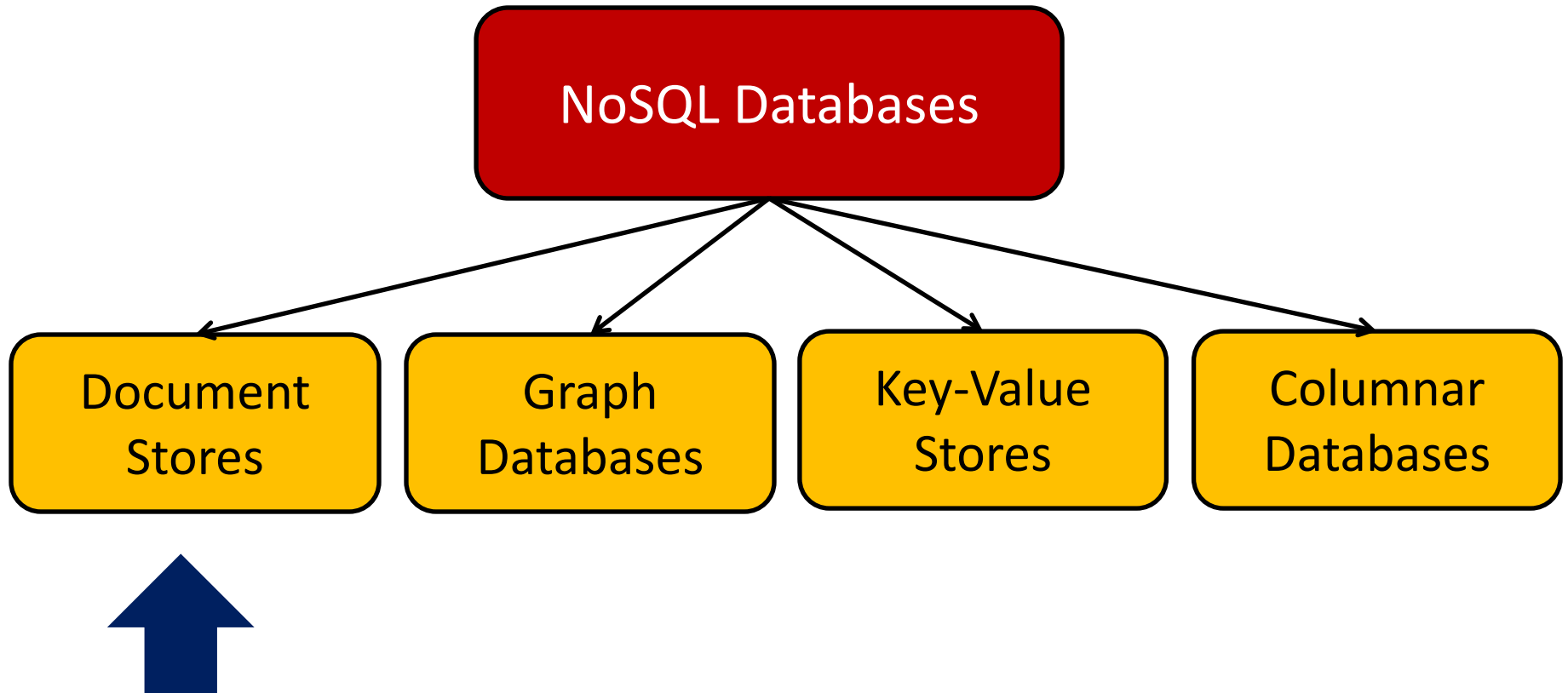


NoSQL Databases

- 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

Types of NoSQL Databases

- Here is a limited taxonomy of NoSQL databases:

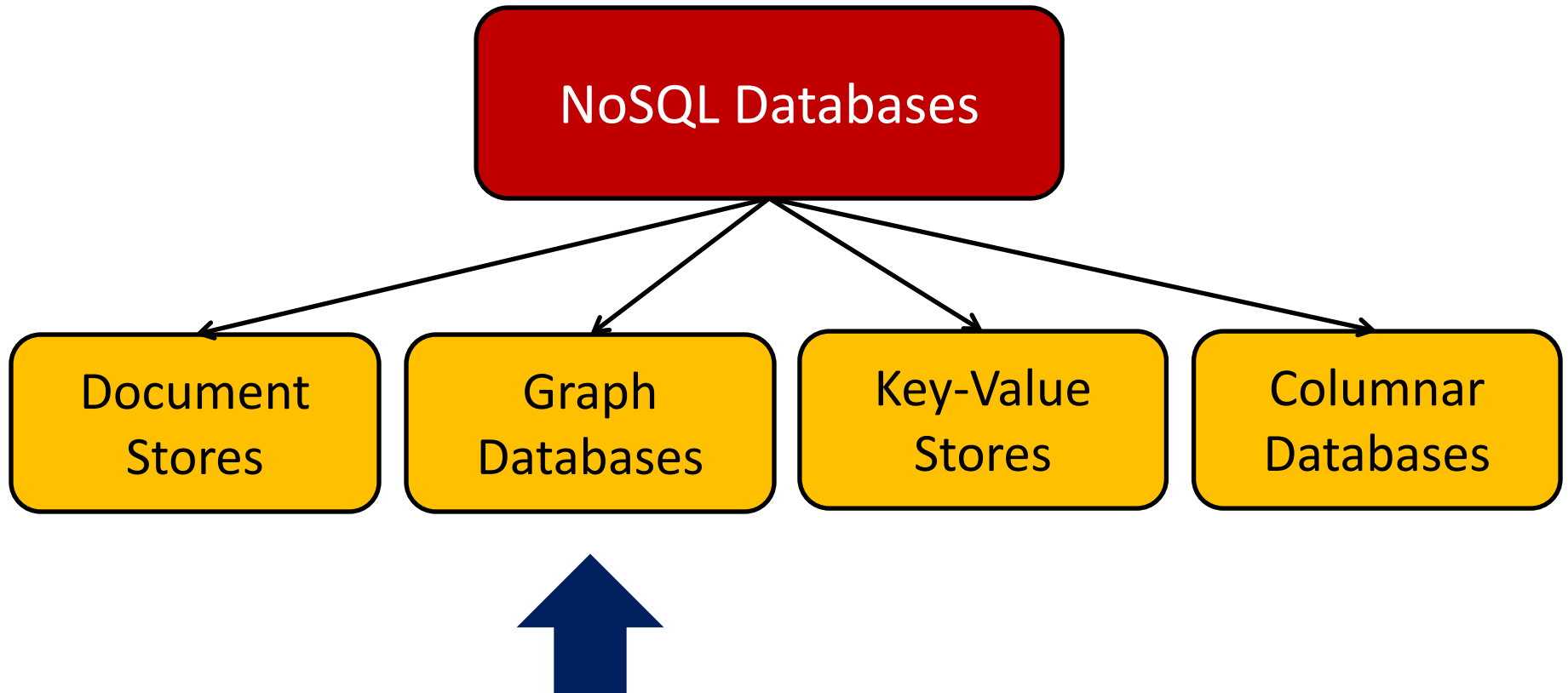


Document Stores

- 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- *next lecture*)

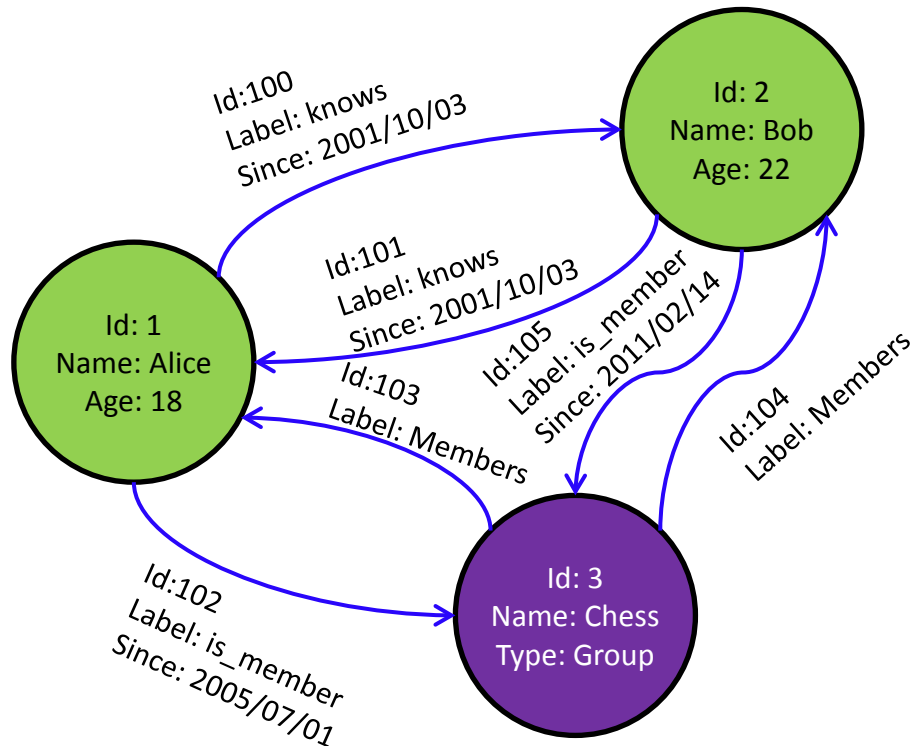
Types of NoSQL Databases

- Here is a limited taxonomy of NoSQL databases:



Graph Databases

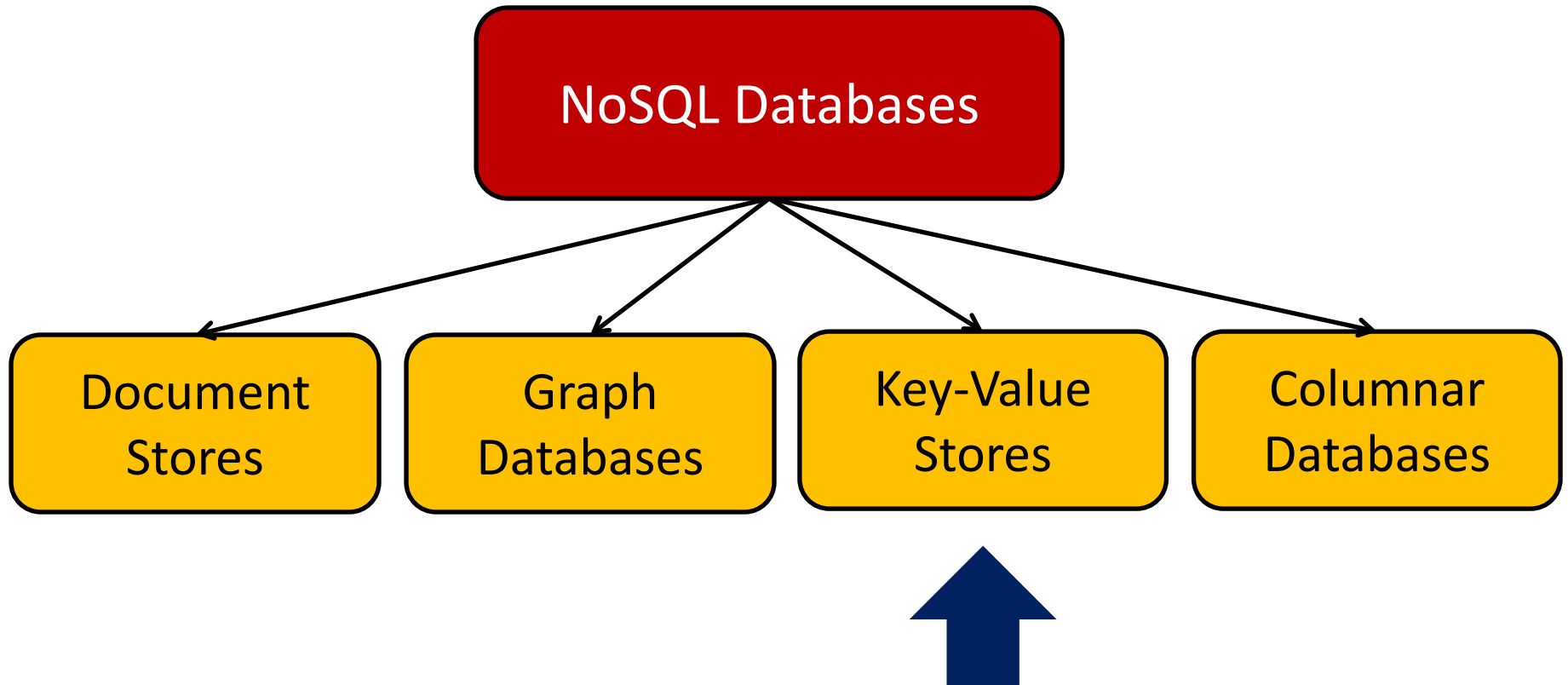
- Data are represented as vertices and edges



- 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:

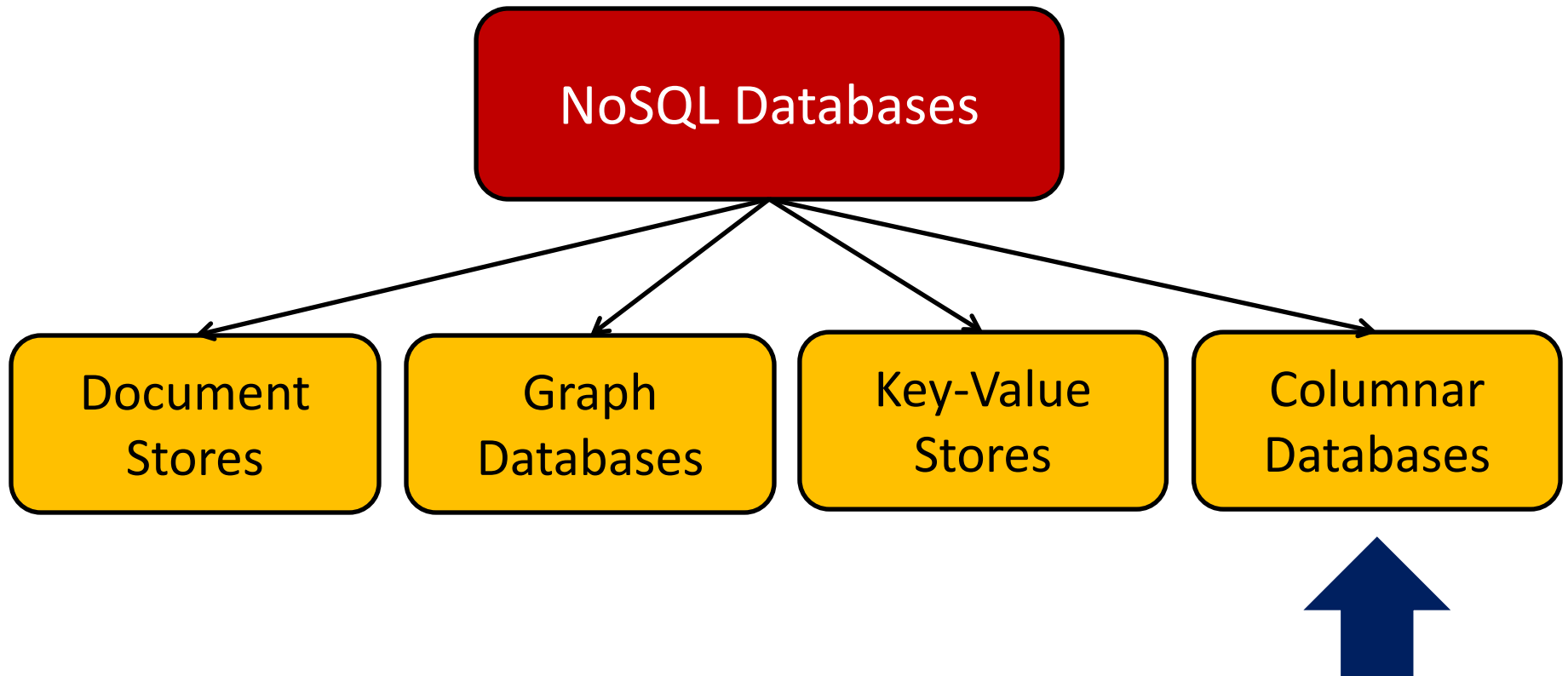


Key-Value Stores

- 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)

Record 1

Alice	3	25	Bob
4	19	Carol	0
45			

Row-Order

Column A

Alice	Bob	Carol
3	4	0
19	45	

Columnar (or Column-Order)

Column A = Group A

Alice	Bob	Carol
3	25	4
0	45	19

Column Family {B, C}

Columnar with Locality Groups

- Values are queried by matching keys
- E.g., HBase and Vertica

Summary

- Data can be classified into 4 types, *structured*, *unstructured*, *dynamic* and *static*
- Different data types usually entail different database designs
- Databases can be scaled *up* or *out*
- The *2PC protocol* can be used to ensure strict consistency
- Strict consistency limits scalability

Summary (*Cont'd*)

- The *CAP theorem* states that any distributed database with shared data can have at most two of the three desirable properties:
 - Consistency
 - Availability
 - Partition Tolerance
- The CAP theorem lead to various designs of databases with *relaxed* ACID guarantees

Summary (*Cont'd*)

- *NoSQL* (or *Not-Only-SQL*) databases follow the *BASE* properties:
 - Basically Available
 - Soft-State
 - Eventual Consistency
- NoSQL databases have different types:
 - Document Stores
 - Graph Databases
 - Key-Value Stores
 - Columnar Databases

Next Class

Hadoop = MapReduce + HDFS