CS150: Database & Datamining
Lecture 28: NoSQL I

Xuming He
Spring 2019

Acknowledgement: Slides are adopted from the Berkeley course CS186 by Joey Gonzalez and Joe Hellerstein, Stanford CS145 by Peter Bailis, IIT Course 236363.

SQL Review

- SQL stands for the query language
- But commonly refers to the traditional RDBMS:
  - Relational storage of data
    - Each tuple is stored consecutively
  - Joins as first-class citizens
    - In fact, normal forms prefer joins to maintenance
  - Strong guarantees on transaction management
    - No consistency worries when many transactions operate simultaneously on common data
- Focus on *scaling up*
  - That is, make a single machine do more, faster

Trends Drive Common Requirements

Social media + mobile computing

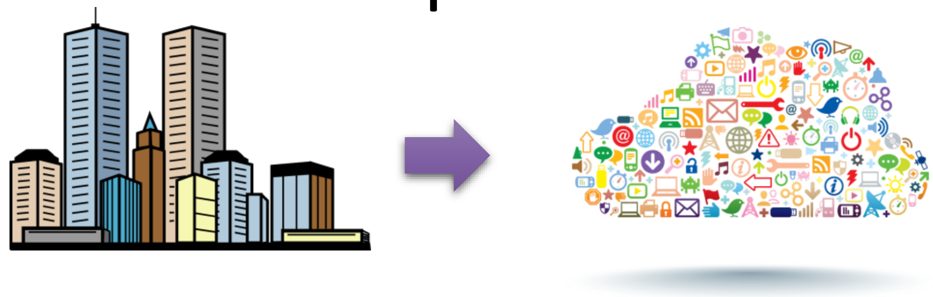
- Explosion in data, always available, constantly read and updated
- High load of simple requests of a common nature
- Some consistency can be compromised (e.g., 👍)

Cloud computing + open source

- Affordable resources for management / analysis of data
- People of various skills / budgets need software solutions for distributed analysis of massive data

Database solutions need to *scale out*
(utilize distribution, “scale horizontally”)

Compromises Required



What is needed for effective distributed, data- and user-intensive applications?

1. Use data models and storage that allow to avoid joins of big objects
2. Relax the guarantees on consistency

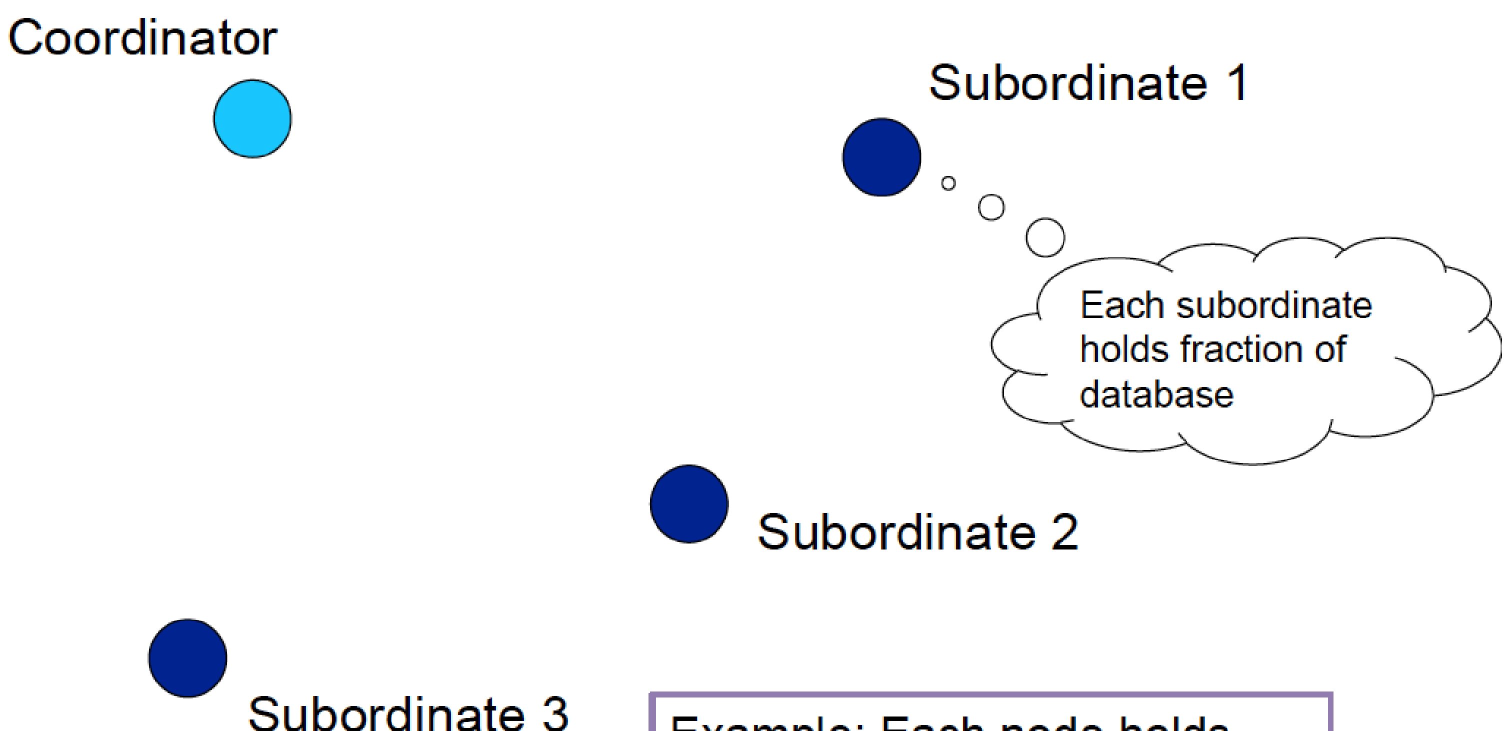
Transaction in distributed systems

- Consider a parallel or distributed database
  - One that handles updates
  - Unlike, say, Spark
- Data is *partitioned (sharded)* across nodes
  - This is how we scale up data volume!
  - Assume only one copy of each record (for now)
- If a transaction touches one machine
  - Life is good
- If a transaction touches multiple machines
  - ACID becomes extremely expensive!

Consistency

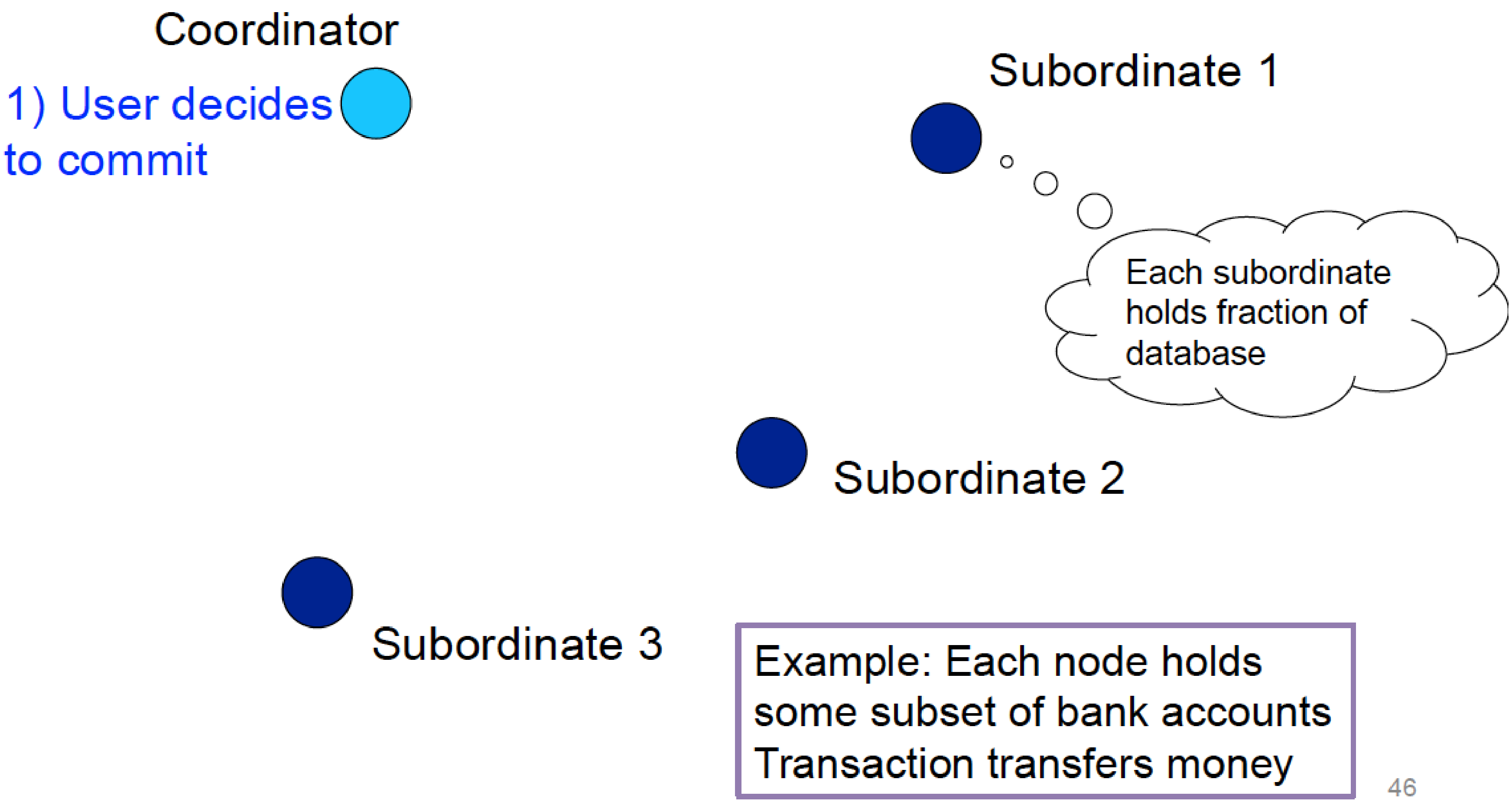
- Assume each transaction is assigned to a node that serves as its "coordinator"
- When a multi-node txn finishes, the DBMS needs to ask all of the nodes involved whether it is safe to commit
  - All nodes must agree on the outcome
- Need two-phase commit

Two-Phase Commit: Motivation

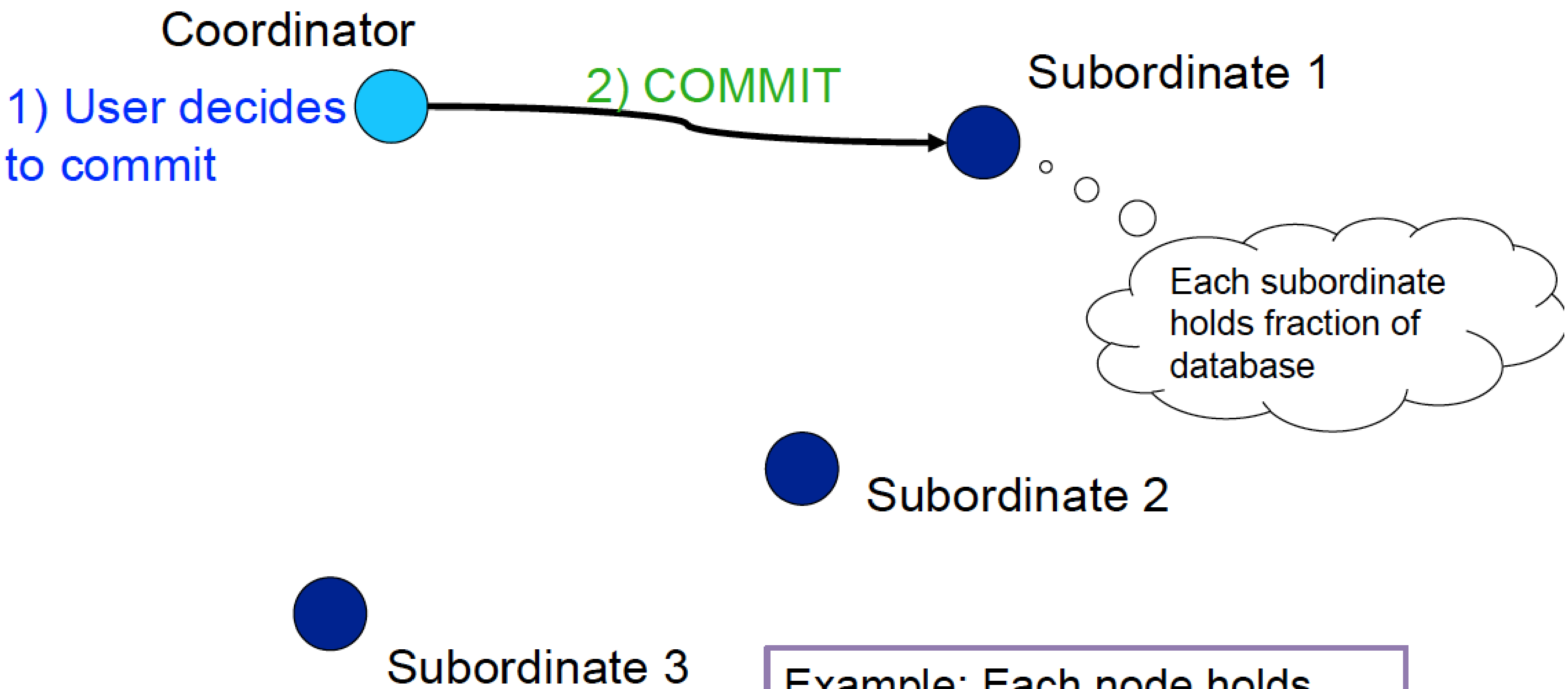


Example: Each node holds some subset of bank accounts
Transaction transfers money

Two-Phase Commit: Motivation

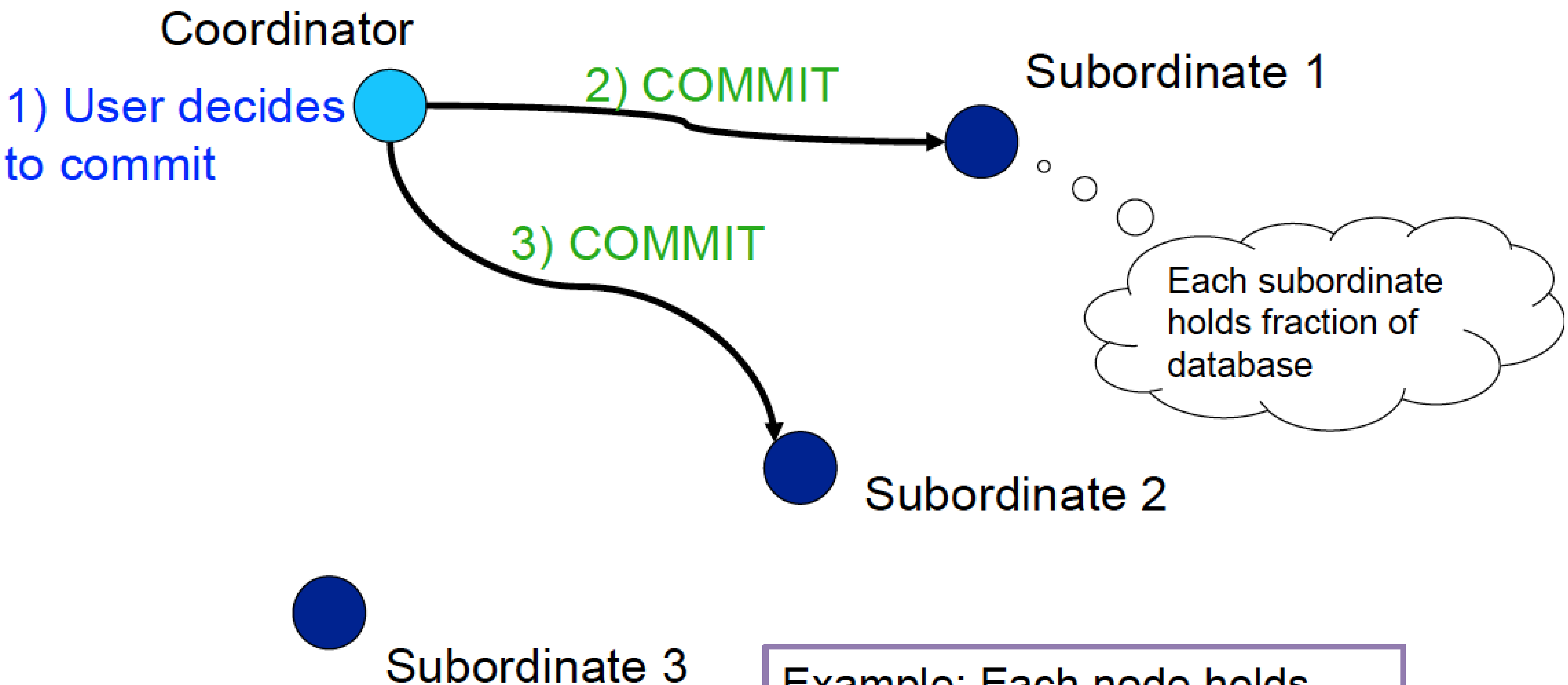


Two-Phase Commit: Motivation



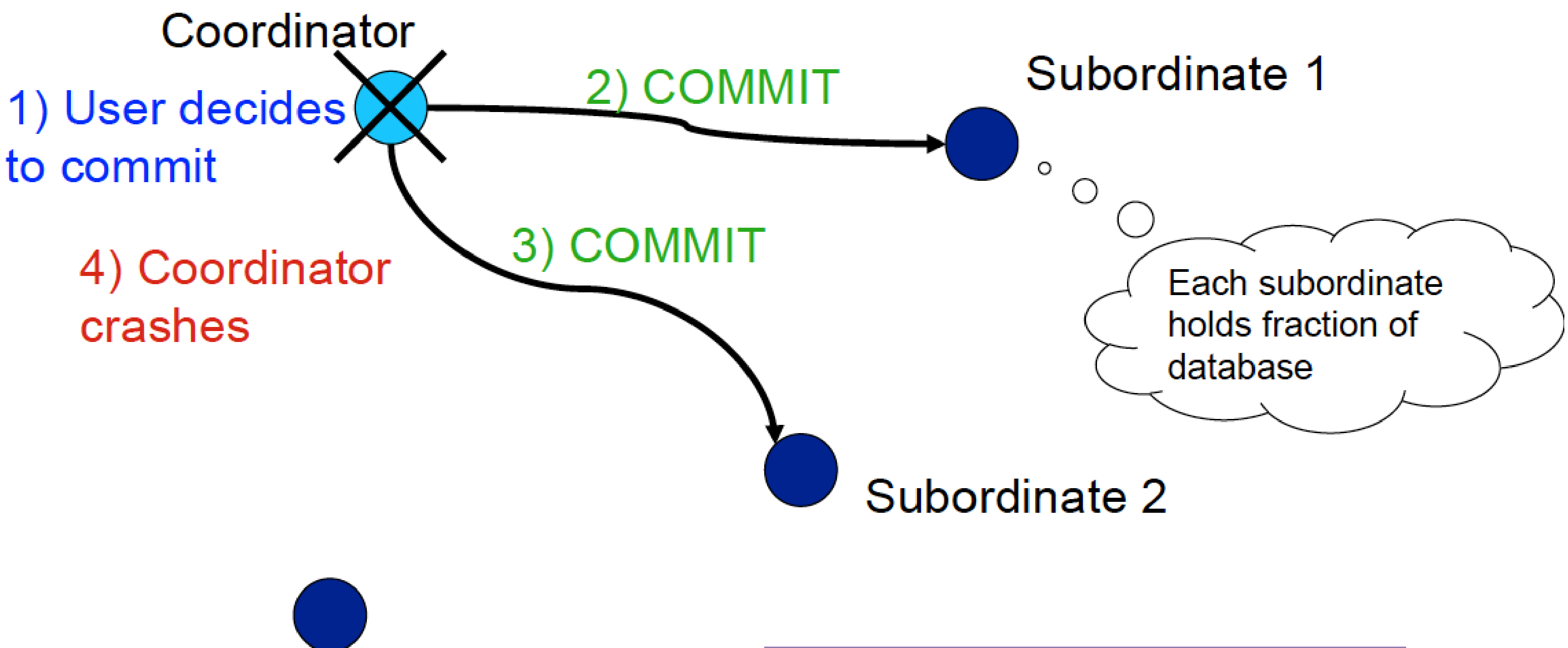
Example: Each node holds some subset of bank accounts
Transaction transfers money

Two-Phase Commit: Motivation



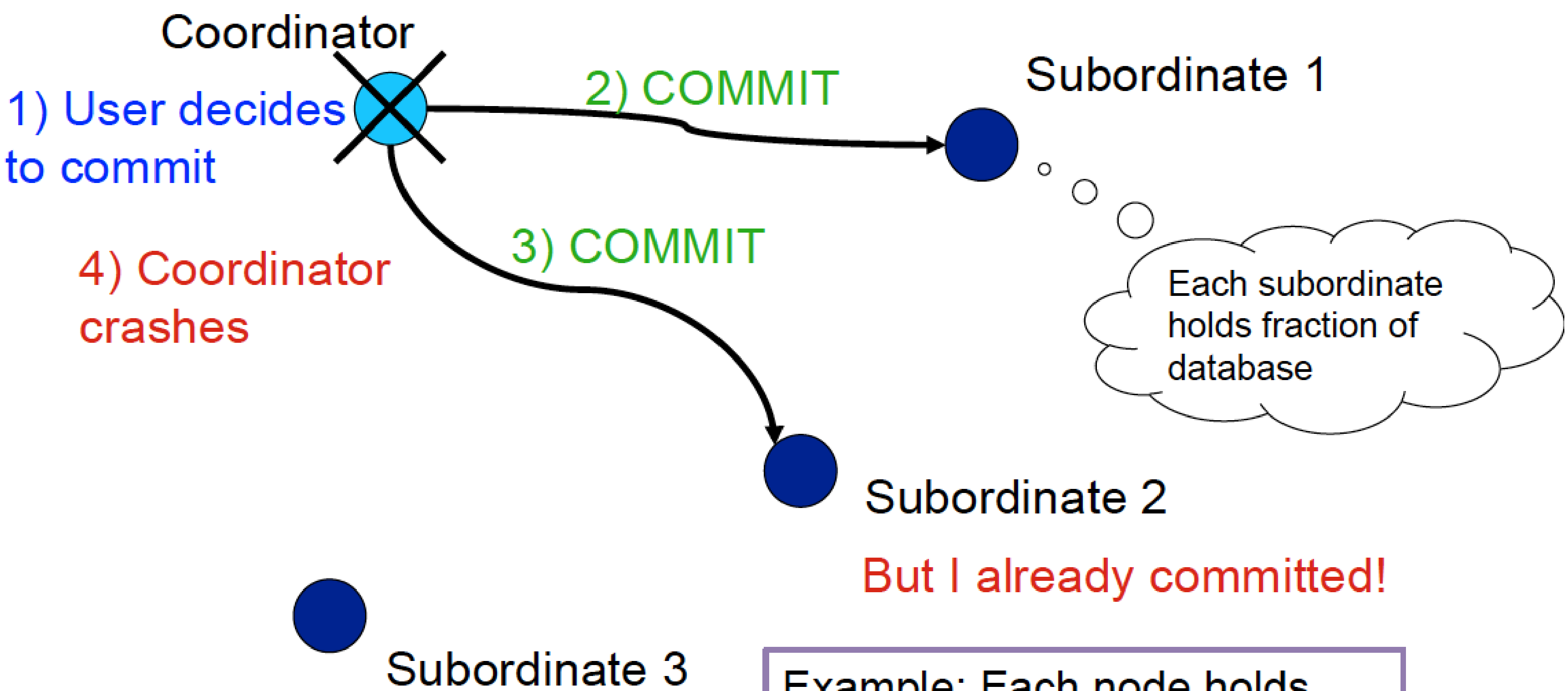
Example: Each node holds some subset of bank accounts
Transaction transfers money

Two-Phase Commit: Motivation



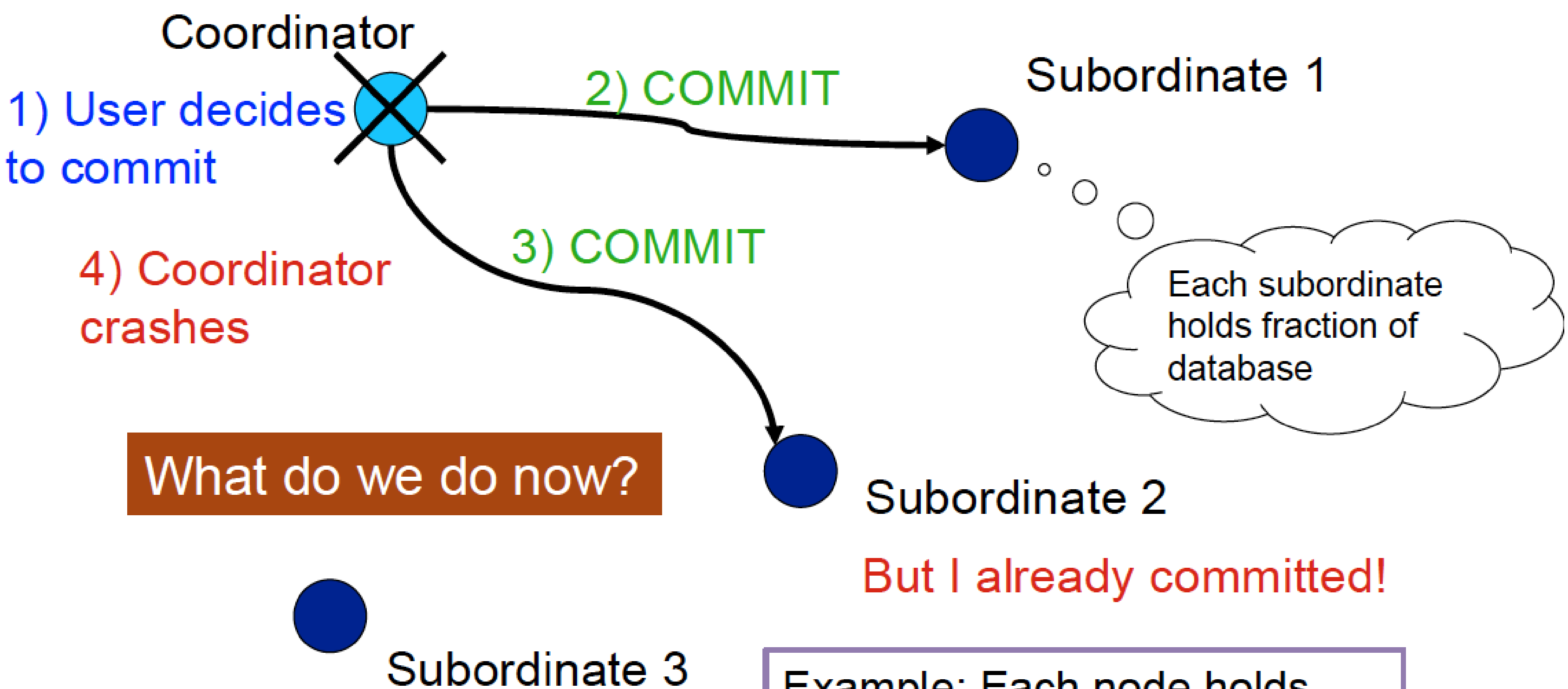
Example: Each node holds some subset of bank accounts
Transaction transfers money

Two-Phase Commit: Motivation



Example: Each node holds some subset of bank accounts
Transaction transfers money

Two-Phase Commit: Motivation

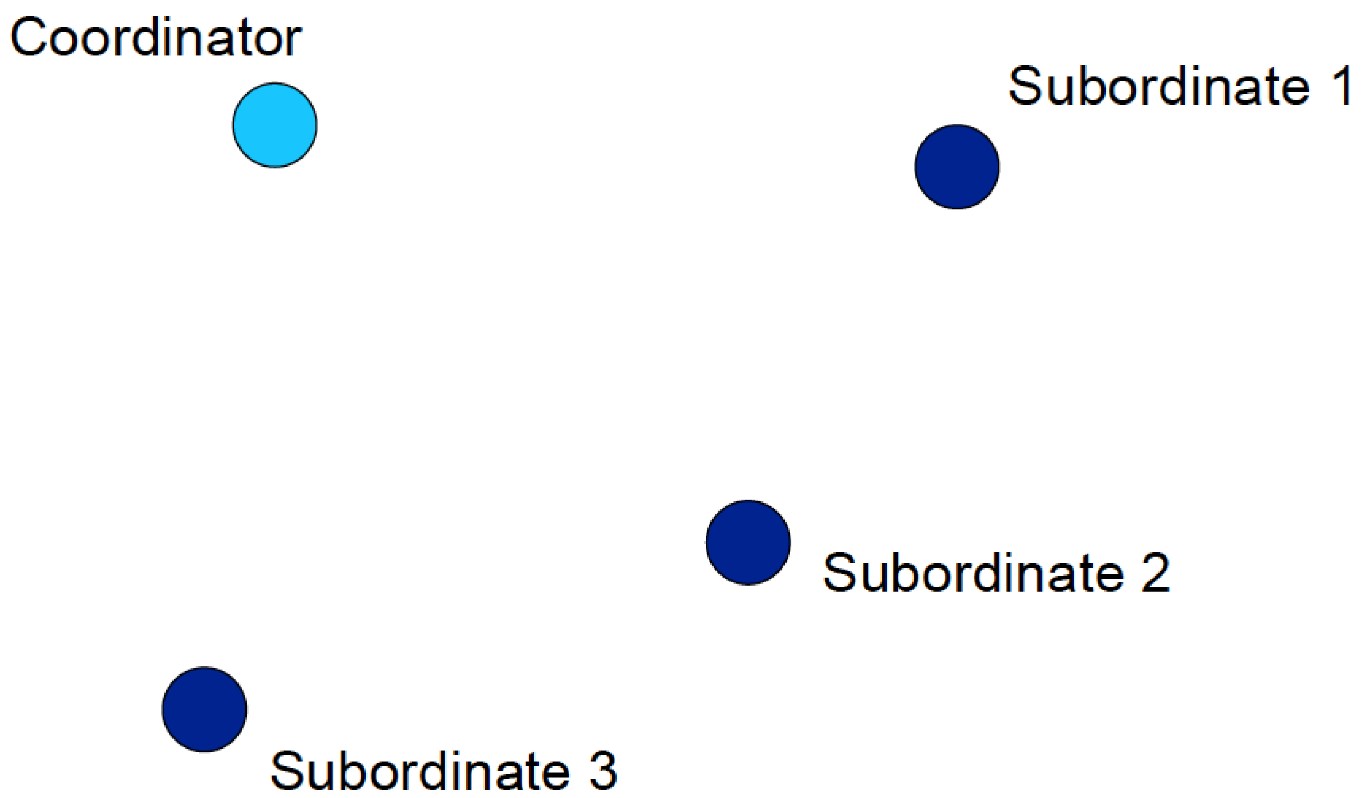


Example: Each node holds some subset of bank accounts
Transaction transfers money

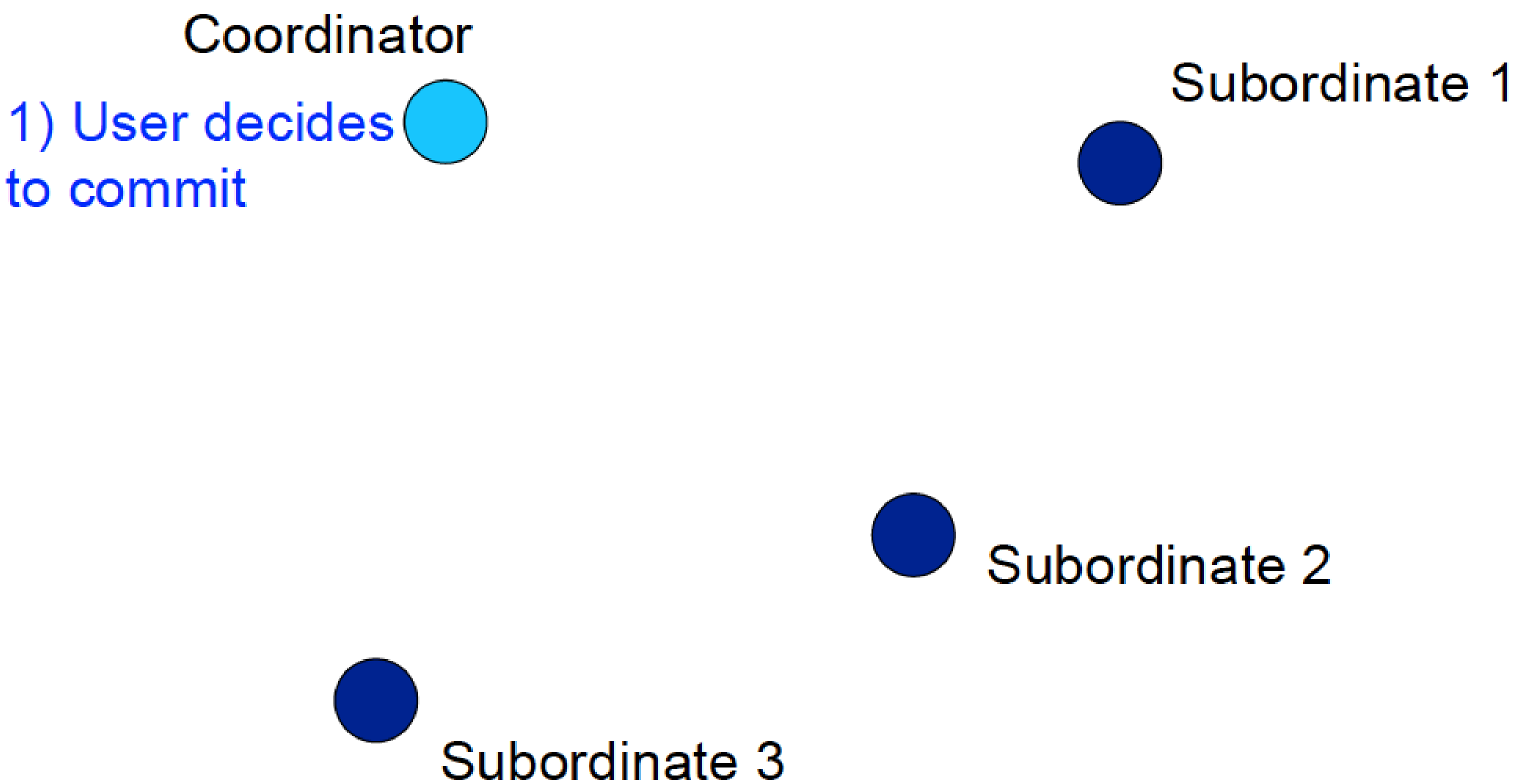
2-Phase Commit

- Phase 1:
  - Coordinator tells participants to "prepare"
  - Participants respond with yes/no votes
    - *Unanimity* required for yes!
- Phase 2:
  - Coordinator disseminates result of the vote
- Need to do some logging for failure handling!

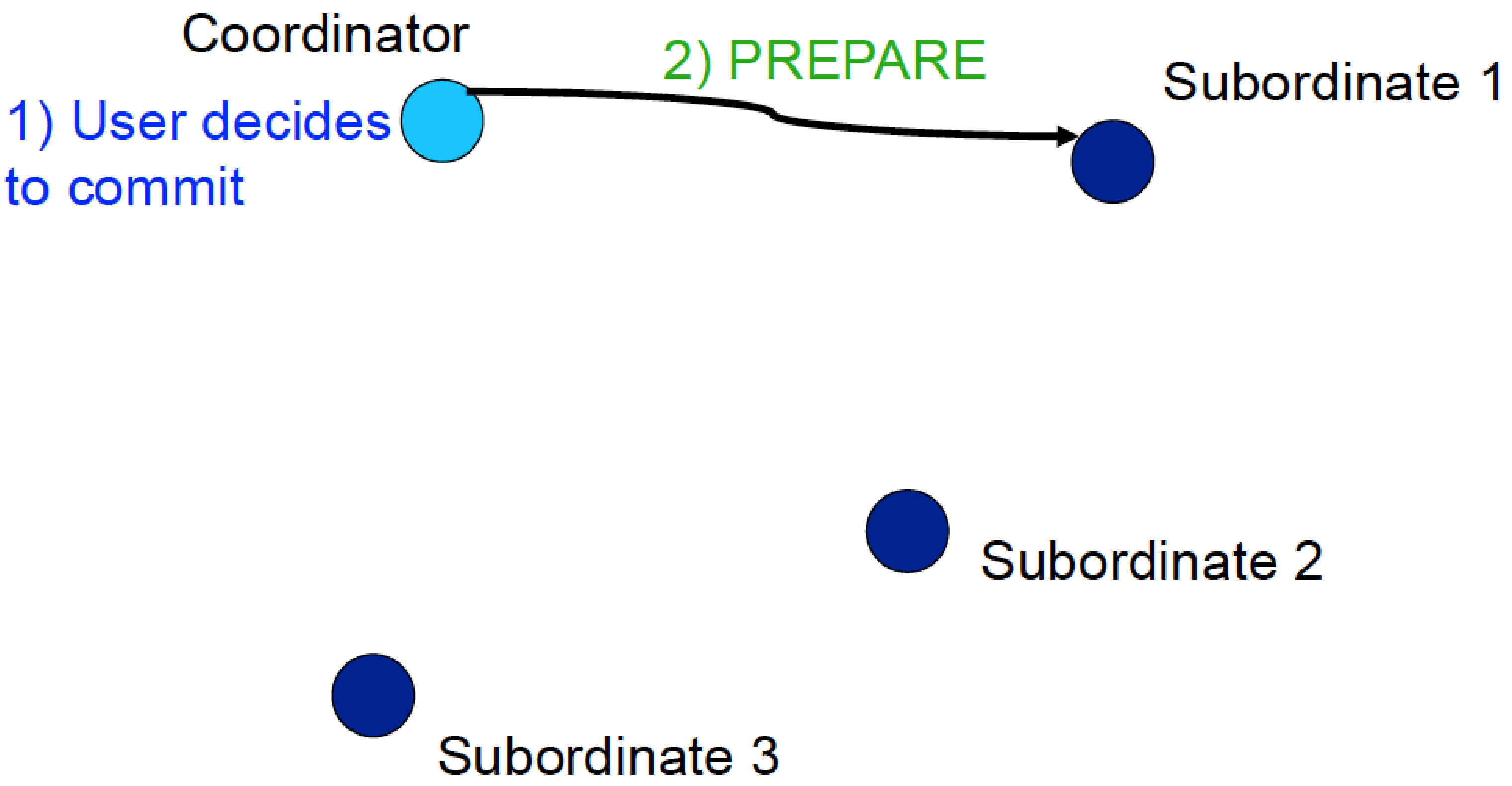
2PC: Phase 1 illustrated



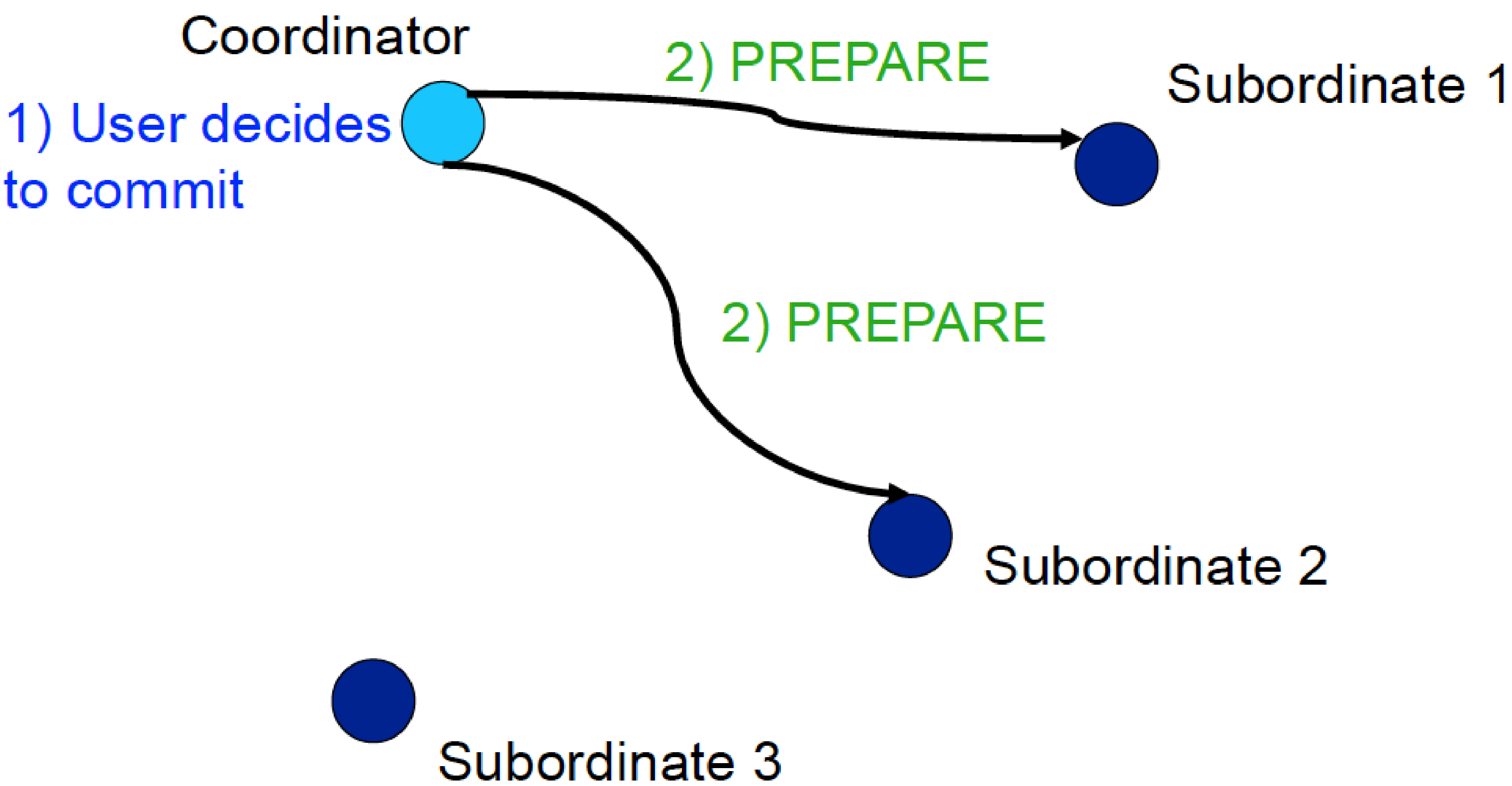
2PC: Phase 1 illustrated



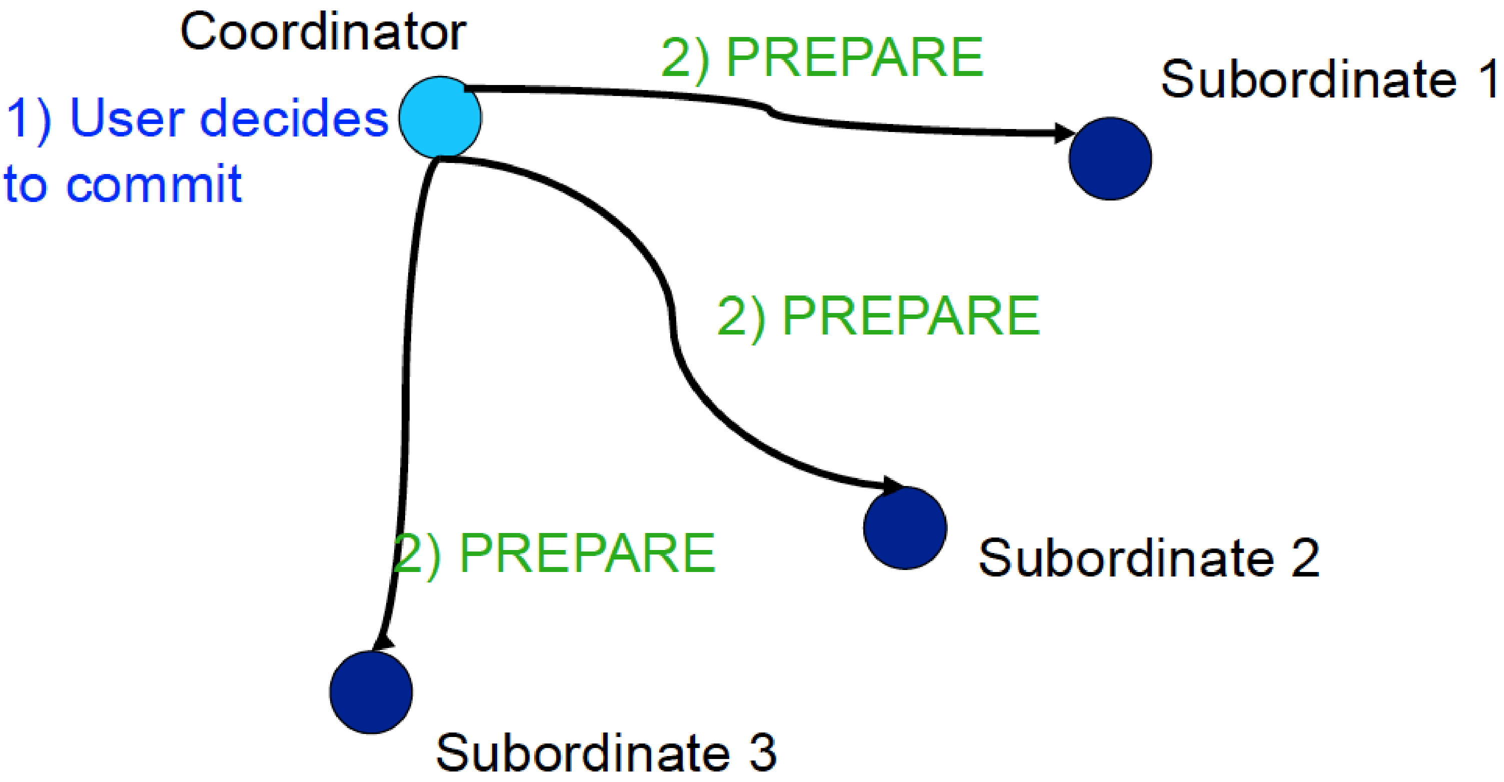
2PC: Phase 1 illustrated



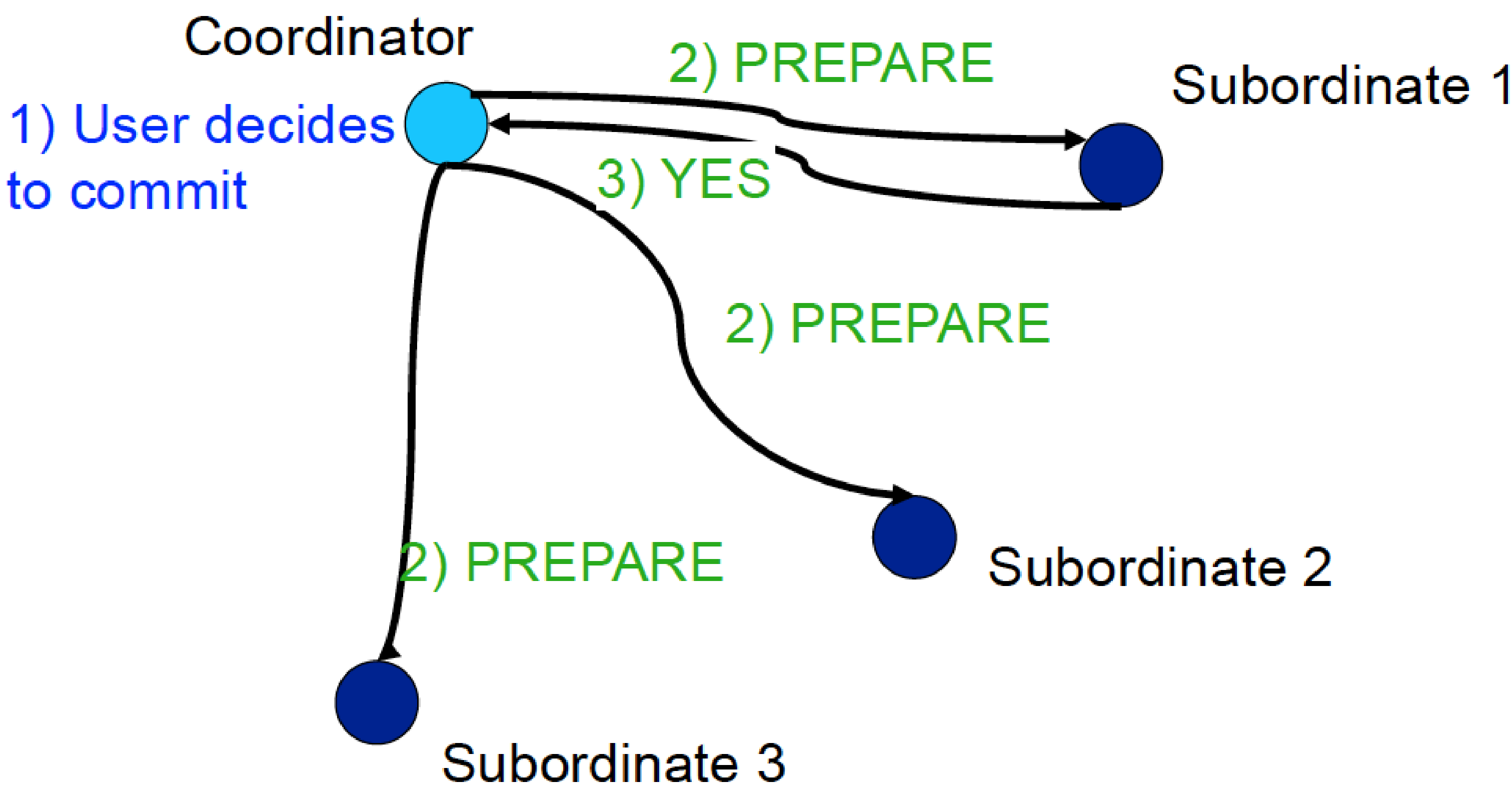
2PC: Phase 1 illustrated



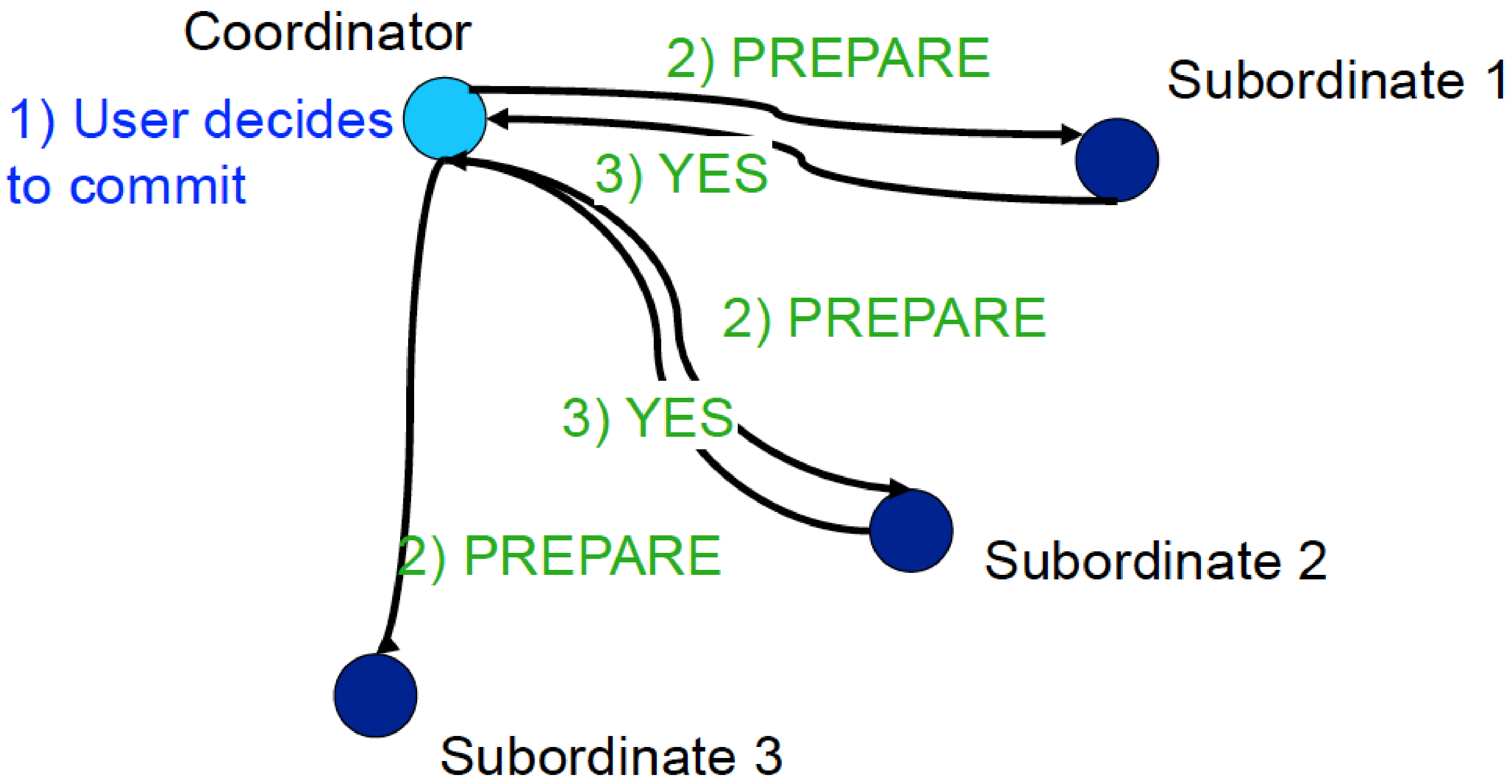
2PC: Phase 1 illustrated



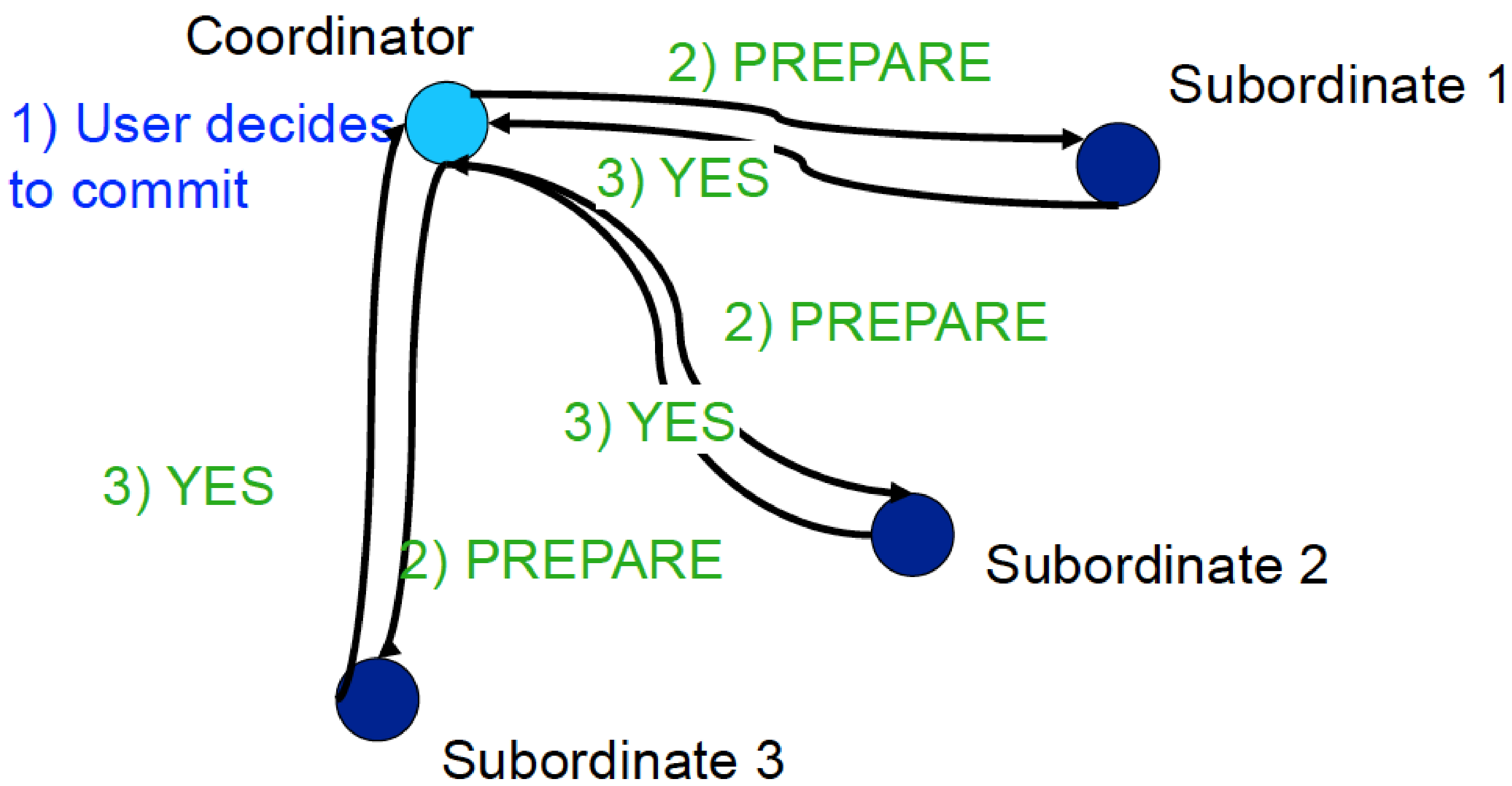
2PC: Phase 1 illustrated



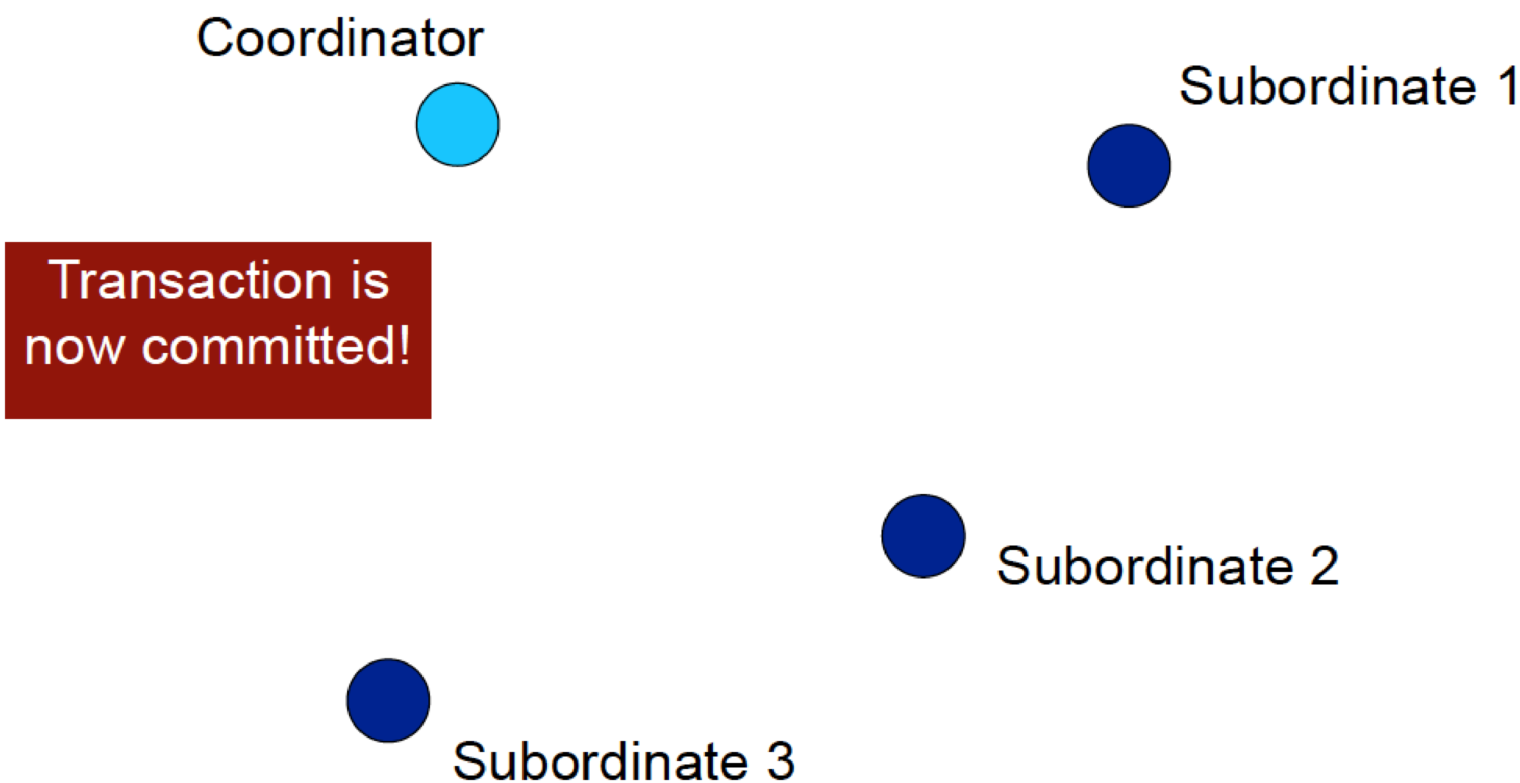
2PC: Phase 1 illustrated



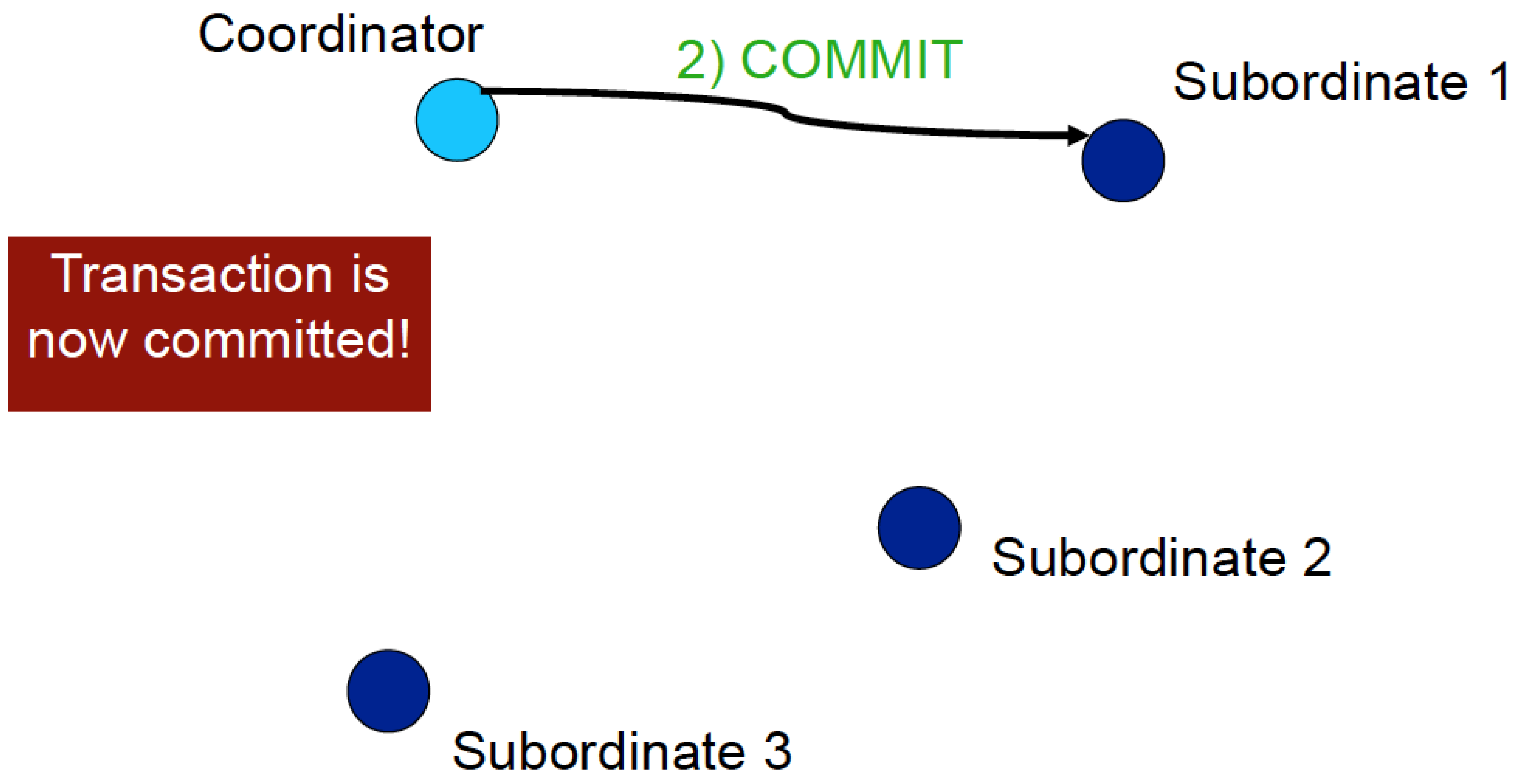
2PC: Phase 1 illustrated



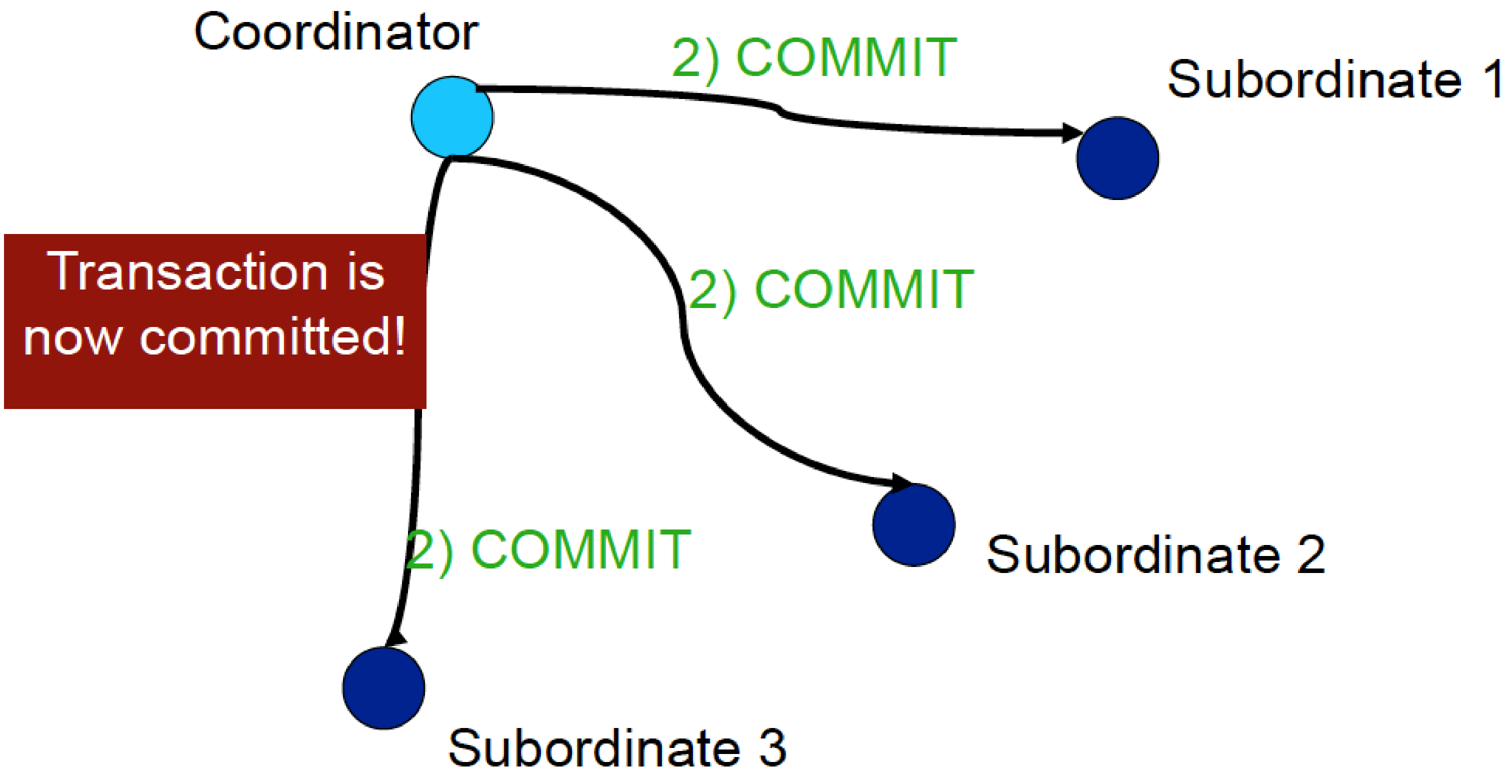
2PC: Phase 2 illustrated



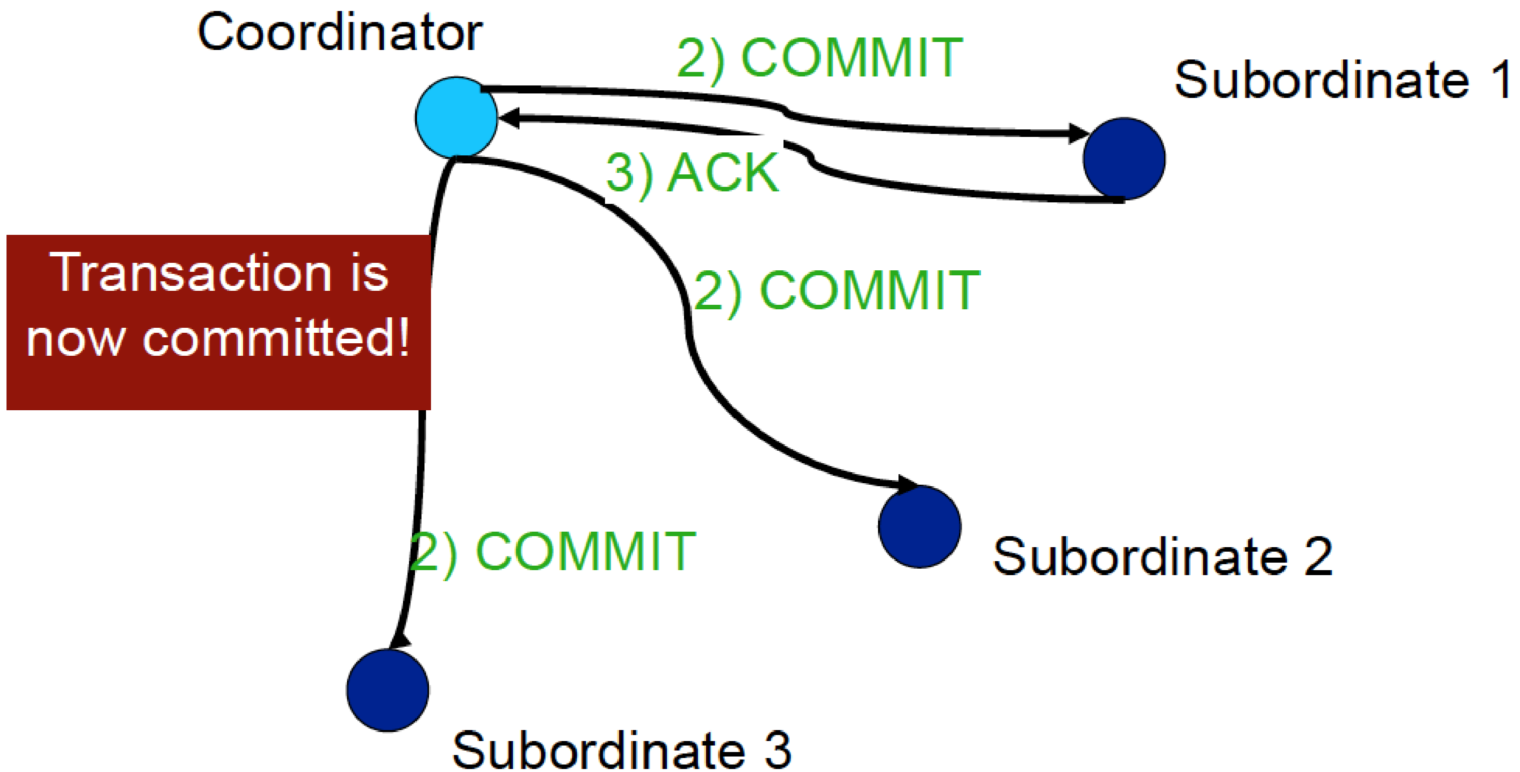
2PC: Phase 2 illustrated



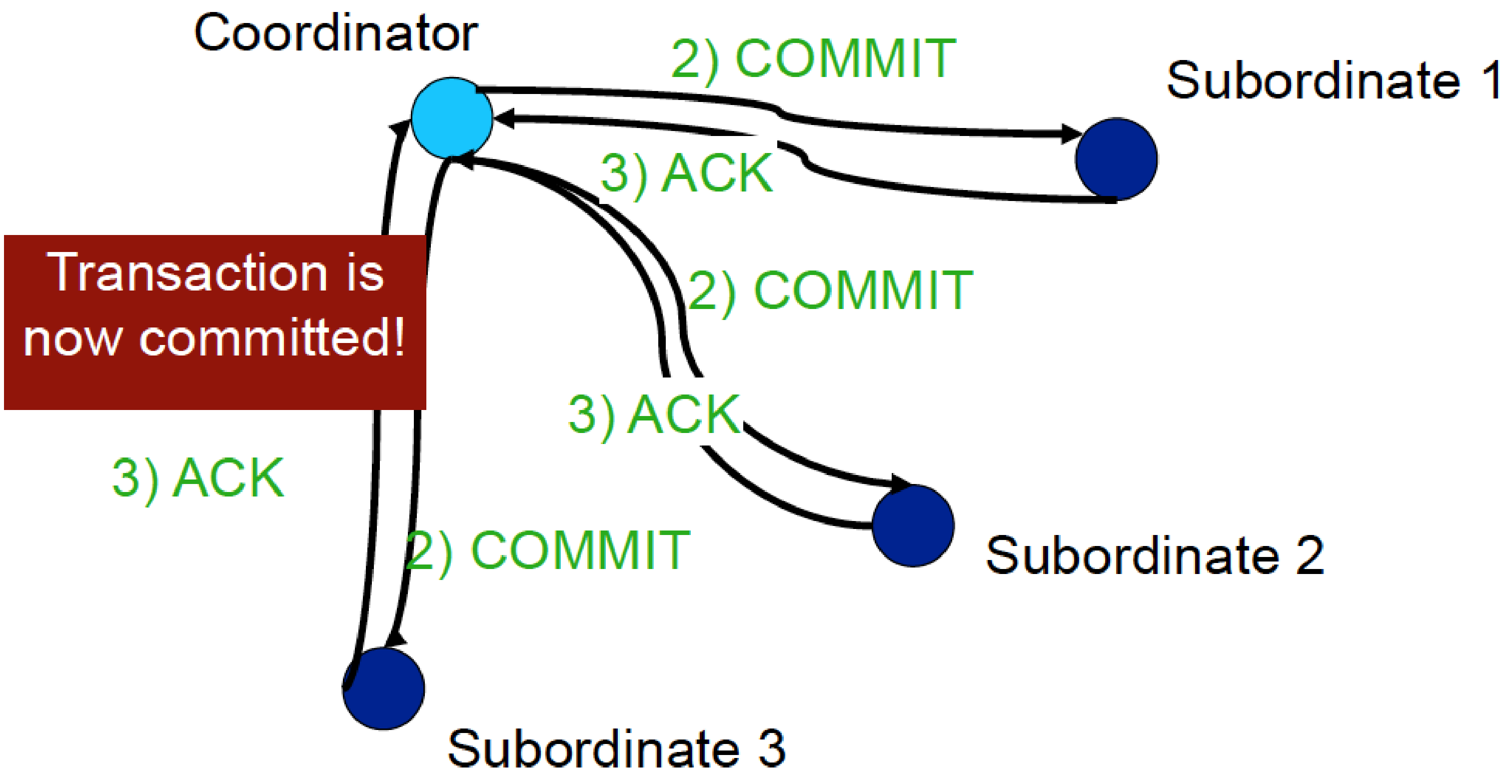
2PC: Phase 2 illustrated



2PC: Phase 2 illustrated



2PC: Phase 2 illustrated



2-Phase Commit

- Multiple servers run parts of the same transaction
- They all must commit, or none should commit
- Two-phase commit is a complicated protocol that ensures that
- 2PC can also be used for WRITE with replication: commit the write at all replicas before declaring success

2-Phase Commit

Assumptions:

- Each site logs actions at that site, but there is no global log
- There is a special site, called the coordinator, which plays a special role
- 2PC involves sending certain messages: as each message is sent, it is logged at the sending site, to aid in case of recovery

2-Phase Commit and Recovery

1. Coordinator sends *prepare* message

2. Subordinates receive *prepare* statement; force-write **<prepare>** log entry; answers *yes* or *no*

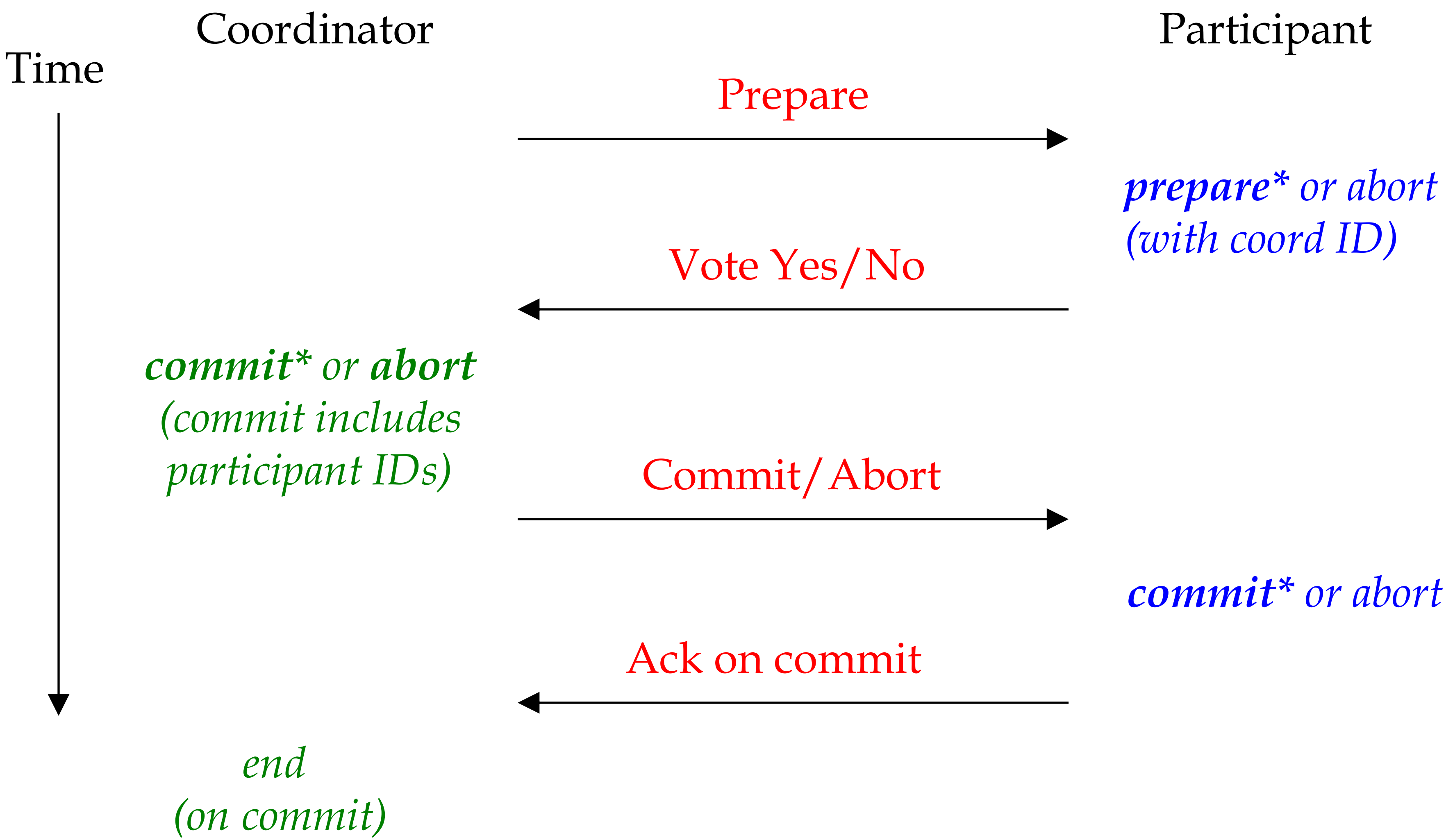
3. If coordinator receives only *yes*, force write **<commit>**, sends *commit* messages;
   If at least one *no*, or timeout, force write **<abort>**, sends *abort* messages

4. If subordinate receives *abort*, force-write **<abort>**, sends *ack* message and aborts; if receives *commit*, force-write **<commit>**, sends *ack*, commits.

5. When coordinator receives all *ack*, writes **<end log>**

2PC: Messages

NOTE
asterisk*: wait for log flush before sending next msg



2-Phase Commit and Recovery

Restart after failure: each server recovers locally

1. If it finds a **<commit>** or **<abort>** log entry, then: redo or undo; if the server is coordinator, then re-request all *ack* messages, then write **<end log>**

2. If it finds a **<prepare>** entry, then re-contact the coordinator to ask for commit/abort

3. If no **<prepare>** , **<commit>** or **<abort>,** presume abort

2-Phase Commit: Summary

- Transaction: ACID properties, but expensive
- Relies on central coordinator: both performance bottleneck, and single-point-of-failure
- Solution: Paxos = distributed protocol
  - Complex: will not discuss at all

NoSQL Databases

NoSQL

- Not Only SQL
  - Not the other thing!
  - Term introduced by Carlo Strozzi in 1998 to describe an alternative database model
  - Became the name of a movement following Eric Evans's reuse for a distributed-database event
- Seminal papers:
  - Google's BigTable
    - Chang, Dean, Ghemawat, Hsieh, Wallach, Burrows, Chandra, Fikes, Gruber: Bigtable: A Distributed Storage System for Structured Data. OSDI 2006: 205-218
  - Amazon's DynamoDB
    - DeCandia, Hastorun, Jampani, Kakulapati, Lakshman, Pilchin, Sivasubramanian, Vosshall, Vogels: Dynamo: amazon's highly available key-value store. SOSP 2007: 205-220

NoSQL from nosql-database.org

"

- Next Generation Databases mostly addressing some of the points: being *non-relational*, *distributed*, *open-source* and *horizontally scalable*.
- The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge amount of data and more.
- So the misleading term "nosql" (the community now translates it mostly with "not only sql") should be seen as an alias to something like the definition above.

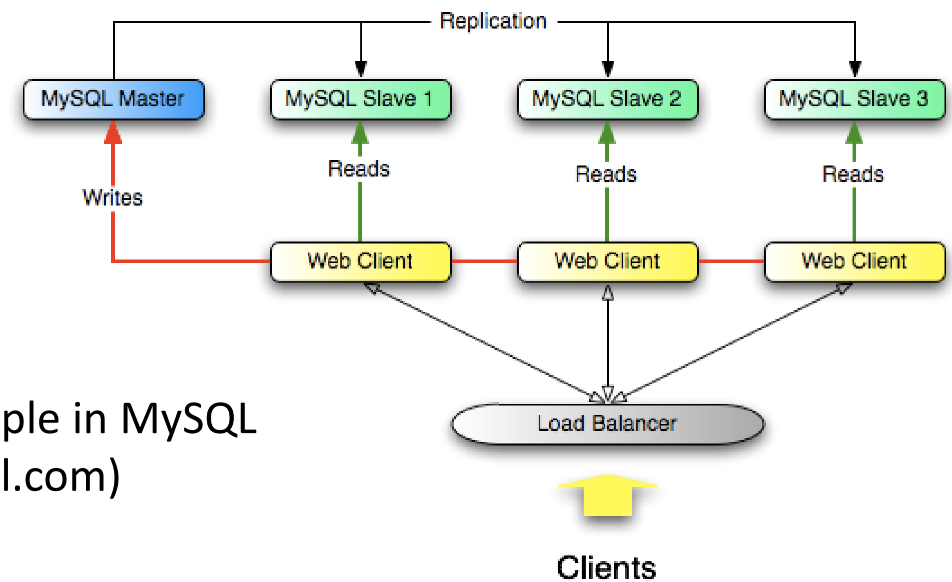
"

Common NoSQL Features

- Non-relational data models
- Flexible structure
  - No need to fix a schema, attributes can be added and replaced on the fly
- Massive read/write performance; availability via horizontal scaling
  - Replication and sharding (data partitioning)
  - Potentially thousands of machines worldwide
- Open source (very often)
- APIs to impose locality

Database Replication

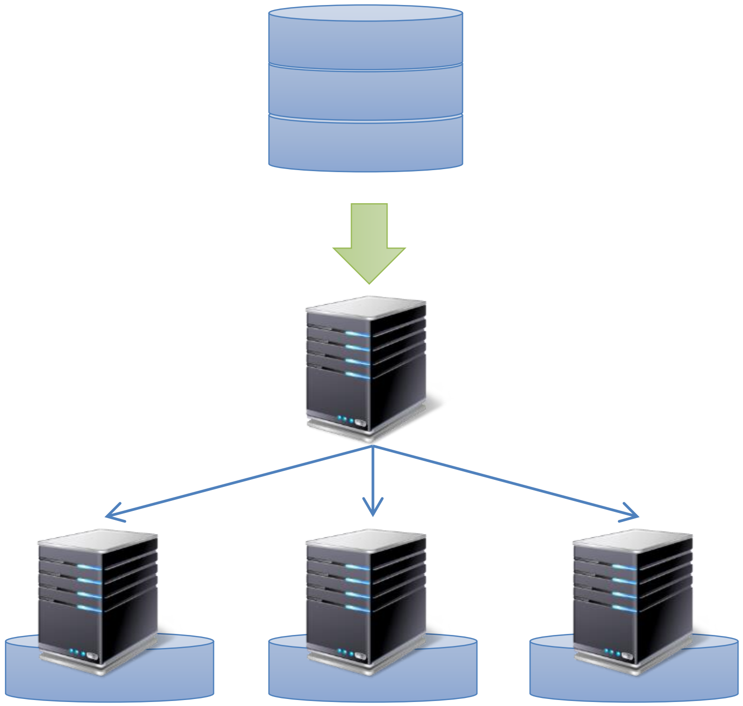
- Data replication: storing the same data on several machines ("nodes")
- Useful for:
  - Availability (parallel requests are made against replicas)
  - Reliability (data can survive hardware faults)
  - Fault tolerance (system stays alive when nodes/network fail)
- Typical architecture: master-slave

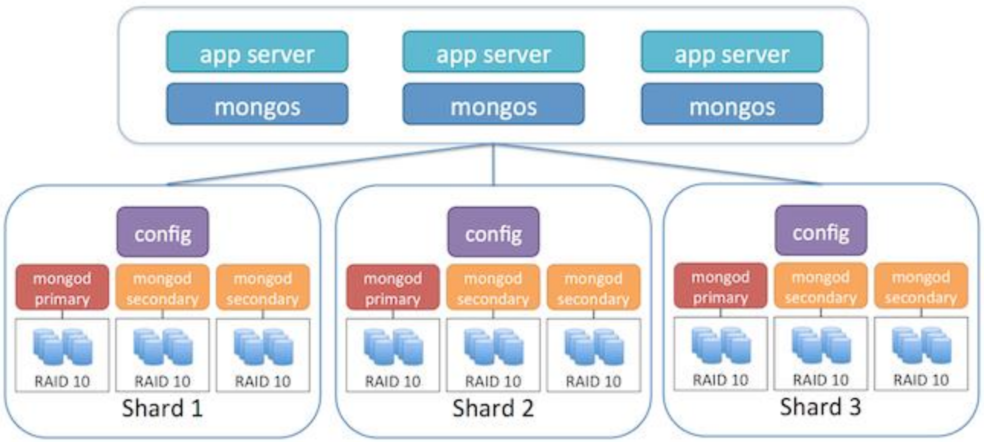


Replication example in MySQL (dev.mysql.com)

Database Sharding

- Simply partitioning data across multiple nodes
- Useful for
  - Scaling (more data)
  - Availability





Replication + sharding example in MongoDB
(mongodb-documentation.readthedocs.org)

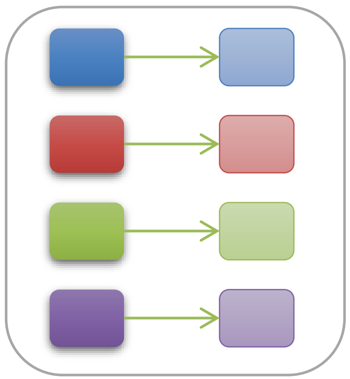
True and False Conceptions

- True:
  - SQL does not effectively handle common Web needs of massive (datacenter) data
  - SQL has guarantees that can sometimes be compromised for the sake of scaling
  - Joins are not for free, sometimes undoable
- False:
  - NoSQL says NO to SQL
  - Nowadays NoSQL is the only way to go
  - Joins can always be avoided by structure redesign

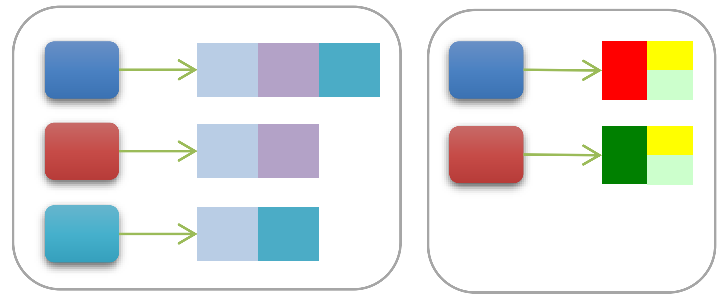
Highlighted Database Features

- Data model
  - What data is being stored?
- CRUD interface
  - API for Create, Read, Update, Delete
  - Sometimes preceding S for Search
- Transaction consistency guarantees
- Replication and sharding model
  - What's automated and what's manual?

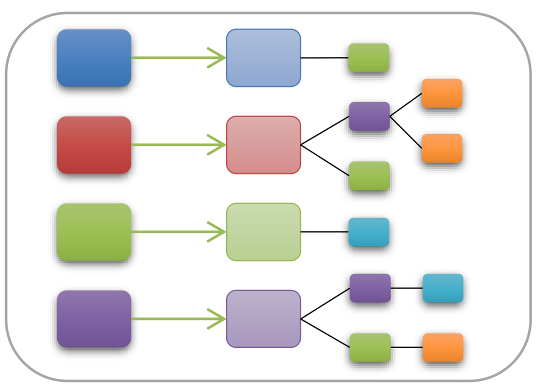
We Will Look at 4 Data Models



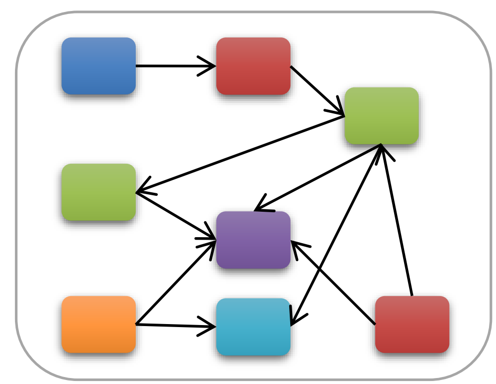
Key/Value Store



Column-Family Store



Document Store



Graph Databases