Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo

Lecture#23: Distributed Database Systems

(R&G ch. 22)

Administrivia – Final Exam

- **Who:** You
- **What:** R&G Chapters 15-22
- **When:** Monday May 11th 5:30pm- 8:30pm
- **Where:** GHC 4401
- **Why:** Databases will help your love life.

Today's Class

- High-level overview of distributed DBMSs.
- Not meant to be a detailed examination of all aspects of these systems.

Today's Class

- Overview & Background
- Design Issues
- Distributed OLTP
- Distributed OLAP

Why Do We Need Parallel/Distributed DBMSs?

- PayPal in 2008…
- Single, monolithic Oracle installation.
- Had to manually move data every xmas.
- Legal restrictions.

Why Do We Need Parallel/Distributed DBMSs?

- Increased Performance.
- Increased Availability.
- Potentially Lower TCO.

Parallel/Distributed DBMS

- Database is spread out across multiple resources to improve parallelism.
- Appears as a single database instance to the application.
  - SQL query for a single-node DBMS should generate same result on a parallel or distributed DBMS.

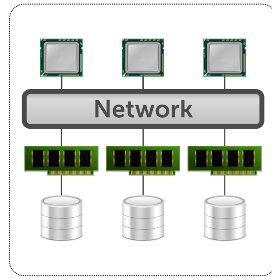
Parallel vs. Distributed

- **Parallel DBMSs:**
  - Nodes are physically close to each other.
  - Nodes connected with high-speed LAN.
  - Communication cost is assumed to be small.
- **Distributed DBMSs:**
  - Nodes can be far from each other.
  - Nodes connected using public network.
  - Communication cost and problems cannot be ignored.

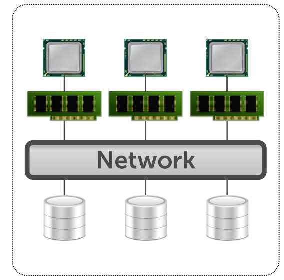
Database Architectures

- The goal is parallelize operations across multiple resources.
  - CPU
  - Memory
  - Network
  - Disk

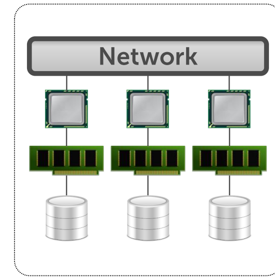
Database Architectures



Shared Memory



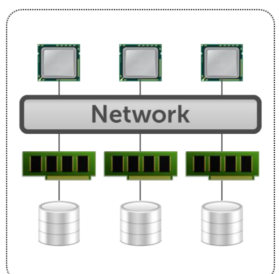
Shared Disk



Shared Nothing

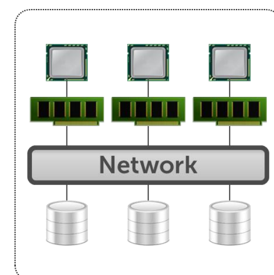
Shared Memory

- CPUs and disks have access to common memory via a fast interconnect.
  - Very efficient to send messages between processors.
  - Sometimes called “shared everything”
- Examples: All single-node DBMSs.



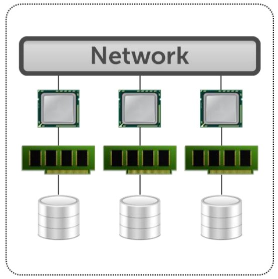
Shared Disk

- All CPUs can access all disks directly via an interconnect but each have their own private memories.
  - Easy fault tolerance.
  - Easy consistency since there is a single copy of DB.
- Examples: Oracle Exadata, ScaleDB.



Shared Nothing

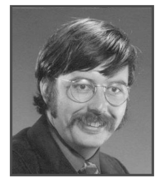
- Each DBMS instance has its own CPU, memory, and disk.
- Nodes only communicate with each other via network.
  - Easy to increase capacity.
  - Hard to ensure consistency.



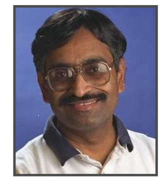
- Examples: Vertica, Parallel DB2, MongoDB.

Early Systems

- **MUFFIN** – UC Berkeley (1979)
- **SDD-1** – CCA (1980)
- **System R*** – IBM Research (1984)
- **Gamma** – Univ. of Wisconsin (1986)
- **NonStop SQL** – Tandem (1987)


Stonebraker


Bernstein


Mohan


DeWitt


Gray

Inter- vs. Intra-query Parallelism

- **Inter-Query:** Different queries or txns are executed concurrently.
  - Increases throughput & reduces latency.
  - Already discussed for shared-memory DBMSs.
- **Intra-Query:** Execute the operations of a single query in parallel.
  - Decreases latency for long-running queries.

Parallel/Distributed DBMSs

- Advantages:
  - Data sharing.
  - Reliability and availability.
  - Speed up of query processing.
- Disadvantages:
  - May increase processing overhead.
  - Harder to ensure ACID guarantees.
  - More database design issues.

Today's Class

- Overview & Background
- Design Issues
- Distributed OLTP
- Distributed OLAP

Design Issues

- How do we store data across nodes?
- How does the application find data?
- How to execute queries on distributed data?
  - Push query to data.
  - Pull data to query.
- How does the DBMS ensure correctness?

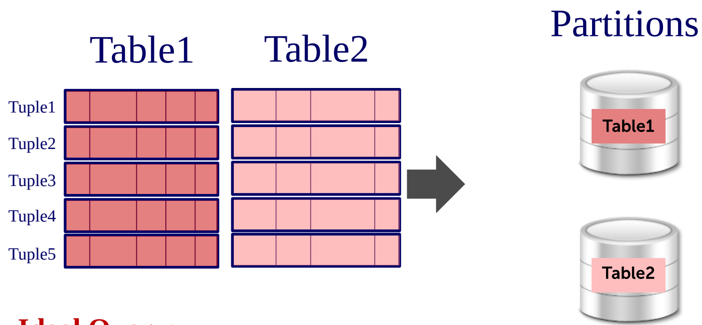
Database Partitioning

- Split database across multiple resources:
  - Disks, nodes, processors.
  - Sometimes called "sharding"
- The DBMS executes query fragments on each partition and then combines the results to produce a single answer.

Naïve Table Partitioning

- Each node stores one and only table.
- Assumes that each node has enough storage space for a table.

Naïve Table Partitioning



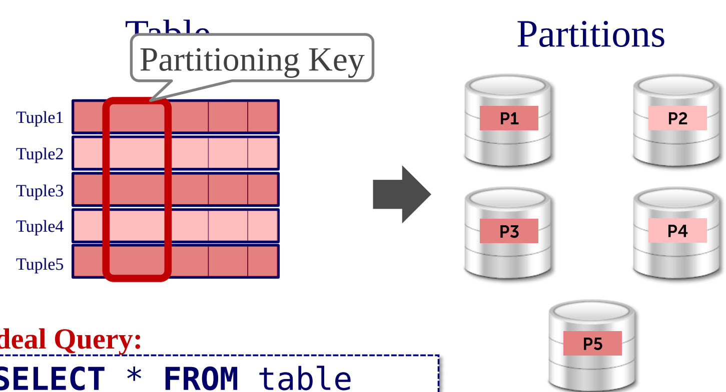
Ideal Query:

```
SELECT * FROM table
```


Horizontal Partitioning

- Split a table's tuples into disjoint subsets.
  - Choose column(s) that divides the database equally in terms of size, load, or usage.
  - Each tuple contains all of its columns.
- Three main approaches:
  - Round-robin Partitioning.
  - Hash Partitioning.
  - Range Partitioning.

Horizontal Partitioning



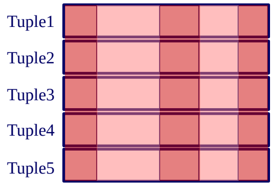
Ideal Query:

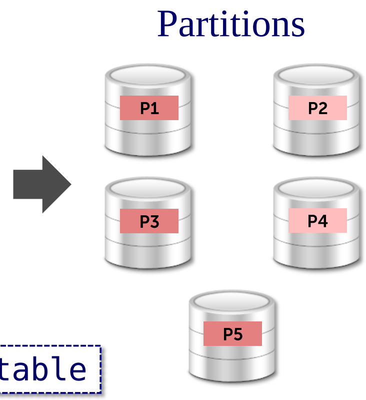
```
SELECT * FROM table
 WHERE partitionKey = ?
```


Vertical Partitioning

- Split the columns of tuples into fragments:
  - Each fragment contains all of the tuples' values for column(s).
- Need to include primary key or unique record id with each partition to ensure that the original tuple can be reconstructed.

Vertical Partitioning





Ideal Query:

```
SELECT column FROM table
```

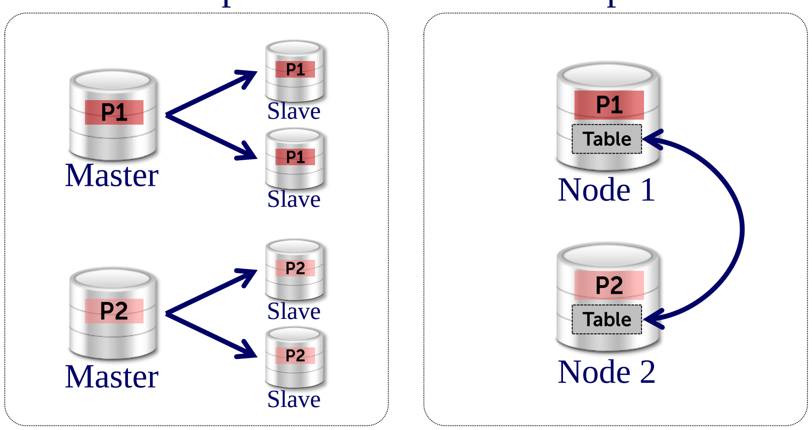

Replication

- **Partition Replication:** Store a copy of an entire partition in multiple locations.
  - Master – Slave Replication
- **Table Replication:** Store an entire copy of a table in each partition.
  - Usually small, read-only tables.
- The DBMS ensures that updates are propagated to all replicas in either case.

Replication

Partition Replication

Table Replication



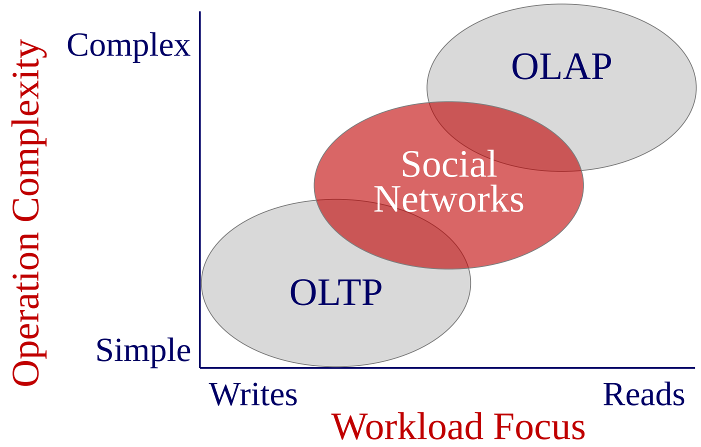
Data Transparency

- Users should not be required to know where data is physically located, how tables are partitioned or replicated.
- A SQL query that works on a single-node DBMS should work the same on a distributed DBMS.

OLTP vs. OLAP

- On-line Transaction Processing:
  - Short-lived txns.
  - Small footprint.
  - Repetitive operations.
- On-line Analytical Processing:
  - Long running queries.
  - Complex joins.
  - Exploratory queries.

Workload Characterization



Michael Stonebraker – *"Ten Rules For Scalable Performance In Simple Operation' Datastores"*
http://cacm.acm.org/magazines/2011/6/108651

Today's Class

- Overview & Background
- Design Issues
- Distributed OLTP
- Distributed OLAP

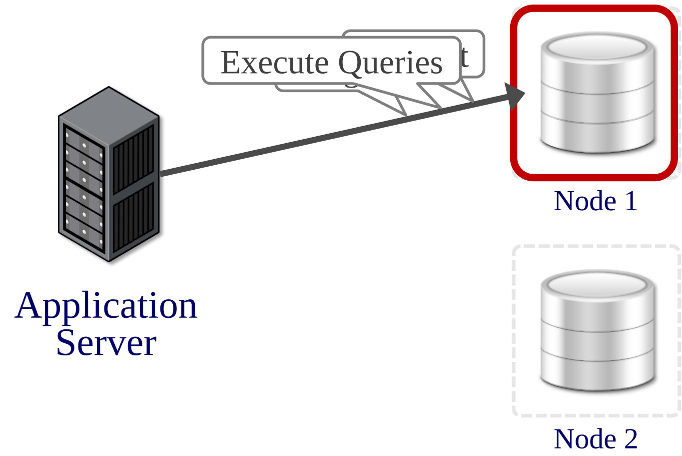
Distributed OLTP

- Execute txns on a distributed DBMS.
- Used for user-facing applications:
  - Example: Credit card processing.
- Key Challenges:
  - Consistency
  - Availability

Single-Node vs. Distributed Transactions

- Single-node txns do not require the DBMS to coordinate behavior between nodes.
- Distributed txns are any txn that involves more than one node.
  - Requires expensive coordination.

Simple Example



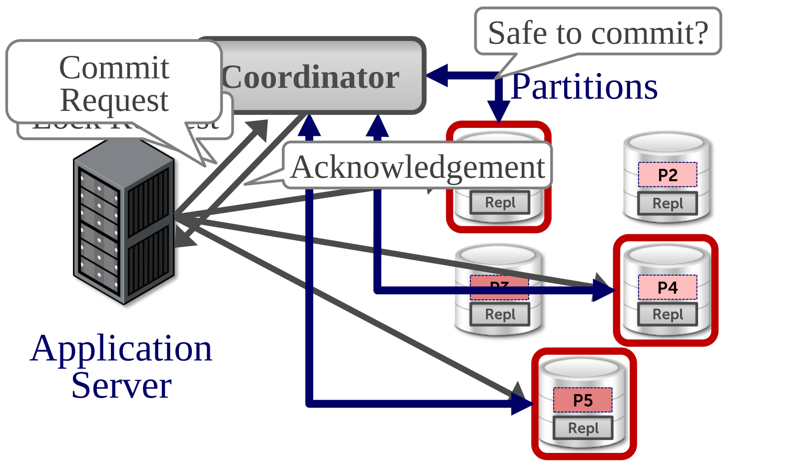
Transaction Coordination

- Assuming that our DBMS supports multi-operation txns, we need some way to coordinate their execution in the system.
- Two different approaches:
  - **Centralized:** Global "traffic cop".
  - **Decentralized:** Nodes organize themselves.

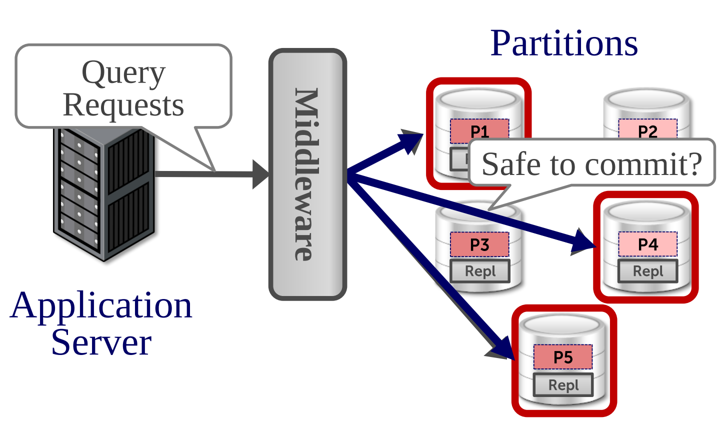
TP Monitors

- Example of a centralized coordinator.
- Originally developed in the 1970-80s to provide txns between terminals + mainframe databases.
  - Examples: ATMs, Airline Reservations.
- Many DBMSs now support the same functionality internally.

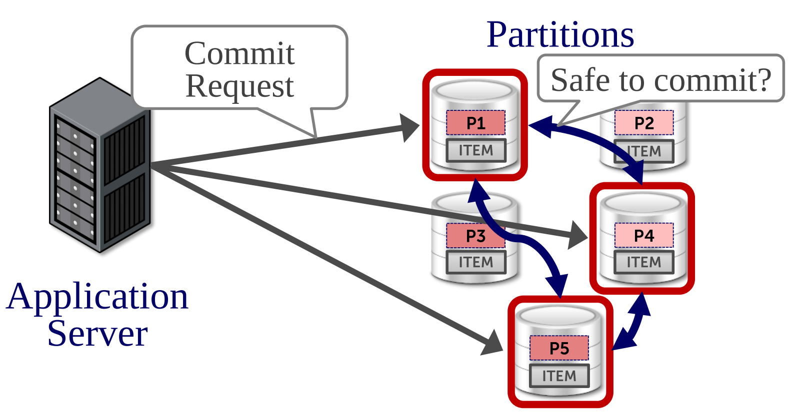
Centralized Coordinator



Centralized Coordinator



Decentralized Coordinator



Observation

- **Q:** How do we ensure that all nodes agree to commit a txn?
  - What happens if a node fails?
  - What happens if our messages show up late?

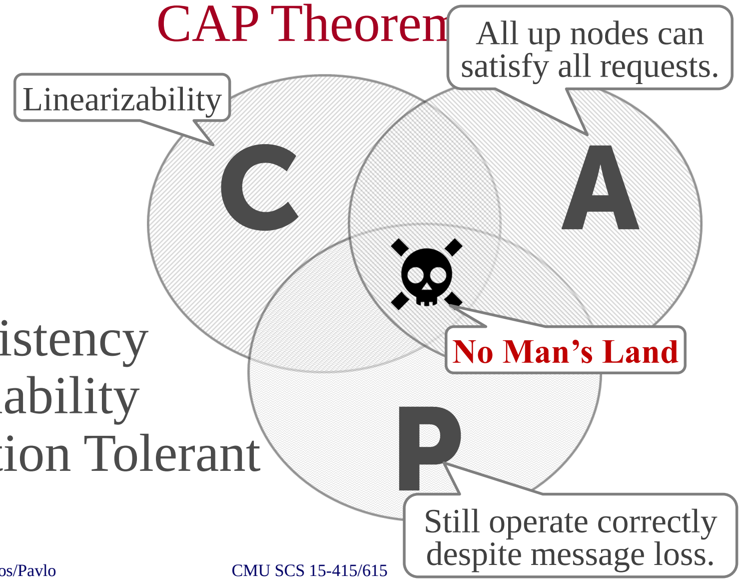
CAP Theorem

- Proposed by Eric Brewer that it is impossible for a distributed system to always be:
  - Consistent
  - Always Available
  - Network Partition Tolerant
- Proved in 2002.

Pick Two!

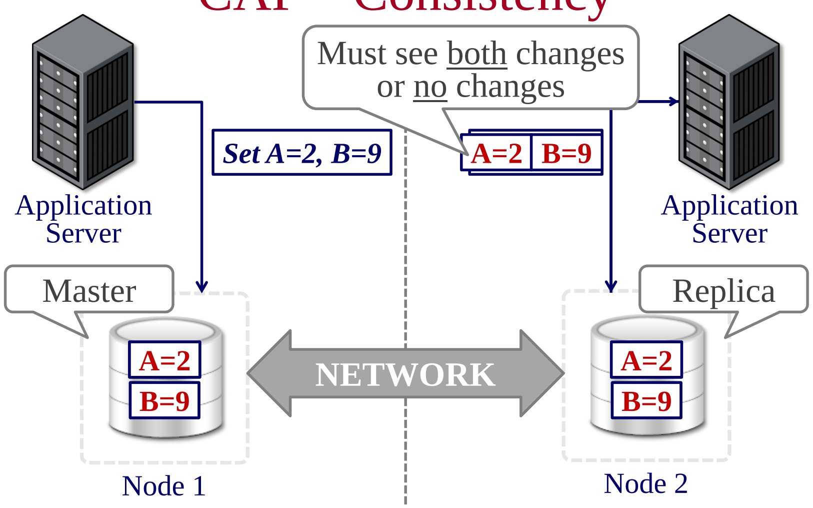
Brewer

CAP Theorem

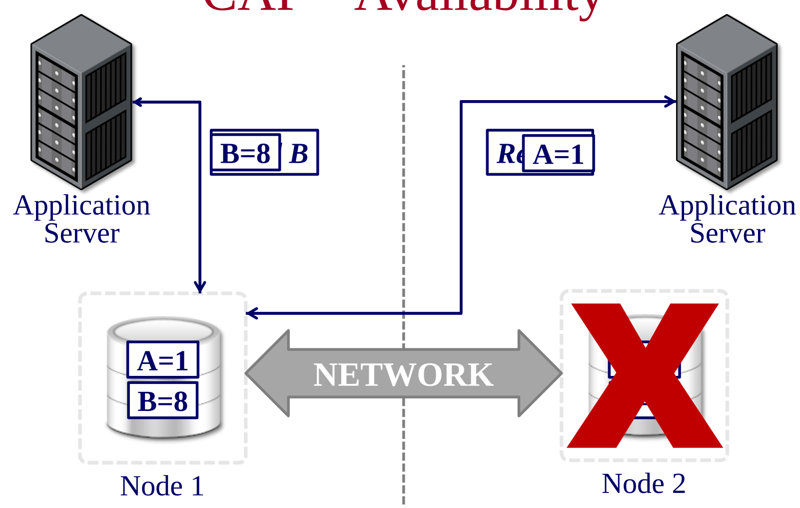


Consistency
Availability
Partition Tolerant

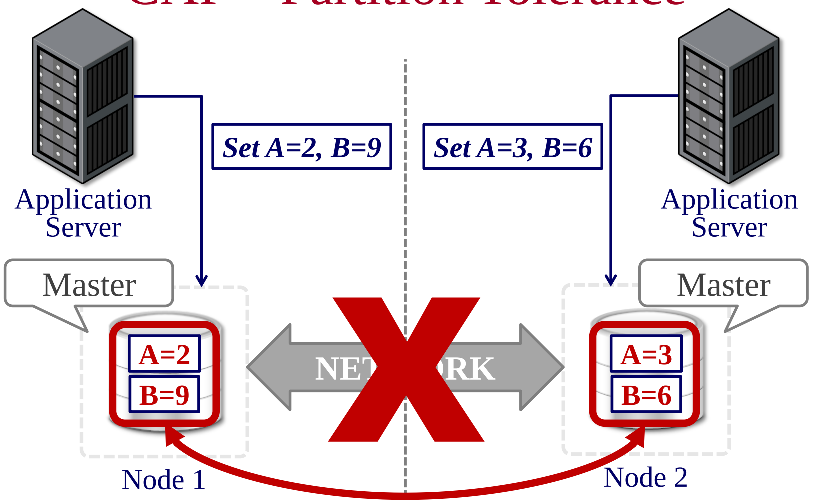
CAP – Consistency



CAP – Availability



CAP – Partition Tolerance



CAP Theorem

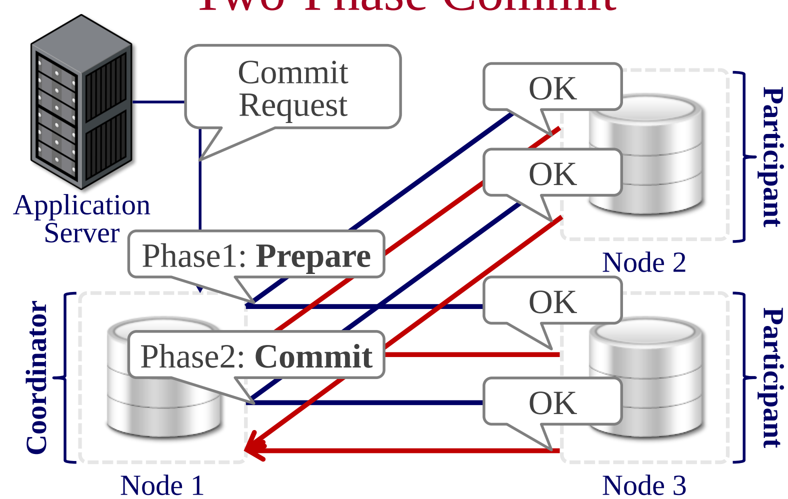
These are essentially the same!

- **Relational DBMSs:** CA/CP
  - Examples: IBM DB2, MySQL Cluster, VoltDB
- **NoSQL DBMSs:** AP
  - Examples: Cassandra, Riak, DynamoDB

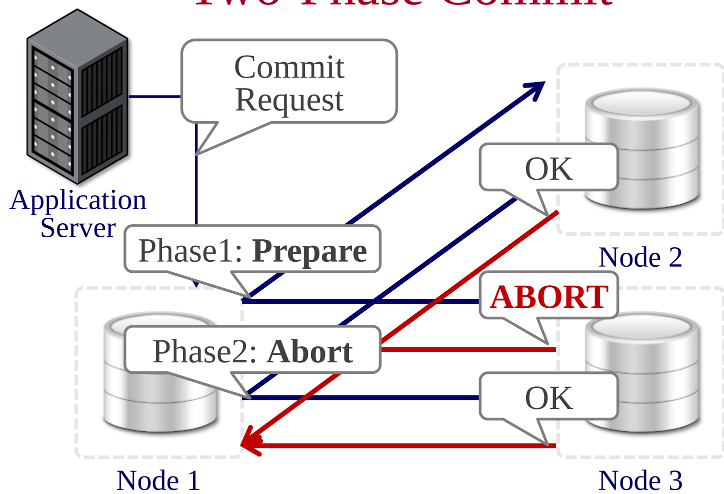
Atomic Commit Protocol

- 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
- Examples:
  - Two-Phase Commit
  - Three-Phase Commit
  - Paxos

Two-Phase Commit



Two-Phase Commit



Two-Phase Commit

- Each node has to record the outcome of each phase in a stable storage log.
- **Q:** What happens if coordinator crashes?
  - Participants have to decide what to do.
- **Q:** What happens if participant crashes?
  - Coordinator assumes that it responded with an abort if it hasn't sent an acknowledgement yet.
- The nodes have to block until they can figure out the correct action to take.

Three-Phase Commit

- The coordinator fi hat it intends to commit the
- If the coordinator fails, then the participants elect a new coordinator and finish commit.
- Nodes do not have to block if there are no network partitions.

Failure doesn't always mean a hard crash.

Paxos

- Consensus protocol where a coordinator proposes an outcome (e.g., commit or abort) and then the participants vote on whether that outcome should succeed.
- Does not block if a majority of participants are available and has provably minimal message delays in the best case.
  - First correct protocol that was provably resilient in the face asynchronous networks

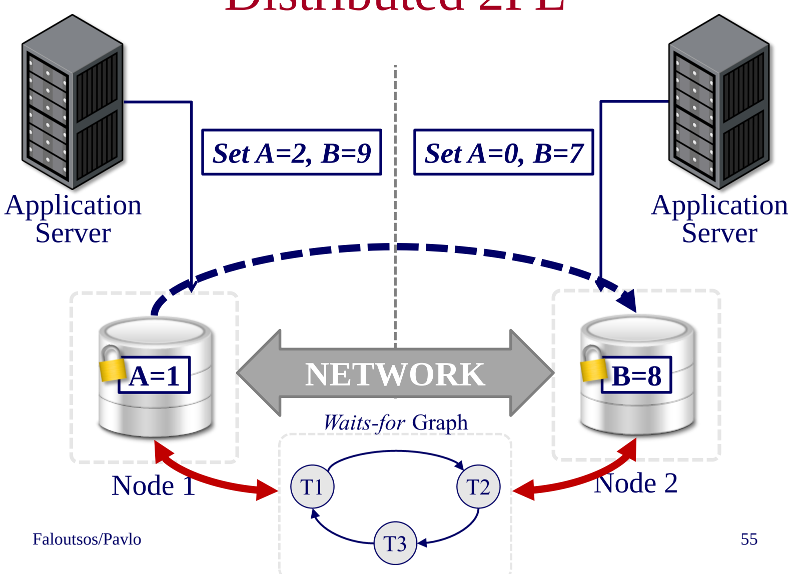
2PC vs. Paxos

- **Two-Phase Commit:** blocks if coordinator fails after the prepare message is sent, until coordinator recovers.
- **Paxos:** non-blocking as long as a majority participants are alive, provided there is a sufficiently long period without further failures.

Distributed Concurrency Control

- Need to allow multiple txns to execute simultaneously across multiple nodes.
  – Many of the same protocols from single-node DBMSs can be adapted.
- This is harder because of:
  – Replication.
  – Network Communication Overhead.
  – Node Failures.

Distributed 2PL



Recovery

- **Q:** What do we do if a node crashes in CA/CP DBMS?
- If node is replicated, use Paxos to elect a new primary.
  – If node is last replica, halt the DBMS.
- Node can recover from checkpoints + logs and then catch up with primary.

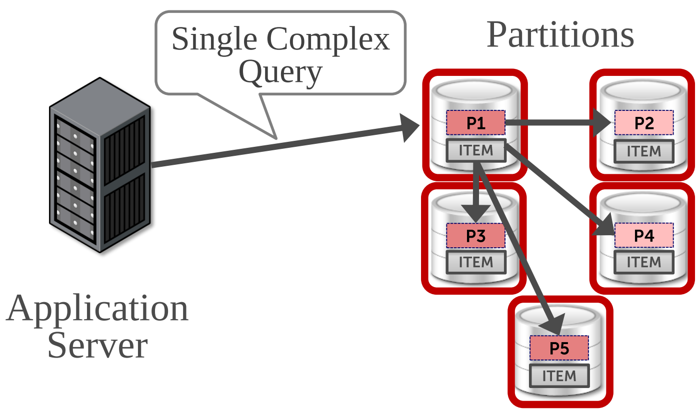
Today's Class

- Overview & Background
- Design Issues
- Distributed OLTP
- Distributed OLAP

Distributed OLAP

- Execute analytical queries that examine large portions of the database.
- Used for back-end data warehouses:
  - Example: Data mining
- Key Challenges:
  - Data movement.
  - Query planning.

Distributed OLAP



Distributed Joins Are Hard

```
SELECT * FROM table1, table2
 WHERE table1.val = table2.val
```

- Assume tables are horizontally partitioned:
  - Table1 Partition Key → table1.key
  - Table2 Partition Key → table2.key
- **Q:** How to execute?
- Naïve solution is to send all partitions to a single node and compute join.

Semi-Joins

- Main Idea: First distribute the join attributes between nodes and then recreate the full tuples in the final output.
  - Send just enough data from each table to compute which rows to include in output.
- Lots of choices make this problem hard:
  - What to materialize?
  - Which table to send?

Summary

- Everything is harder in a distributed setting:
  - Concurrency Control
  - Query Execution
  - Recovery

Next Class

- Discuss distributed OLAP more.
  - You'll learn why MapReduce was a bad idea.
- Compare OldSQL vs. NoSQL vs. NewSQL
- Real-world Systems