Carnegie Mellon University

Distributed OLTP Systems

Lecture #23

Database Systems
15-445/15-645
Fall 2017

Andy Pavlo
Computer Science Dept.
Carnegie Mellon Univ.

ADMINISTRIVIA

Homework #6: TODAY @ 11:59pm

Project #4: Wednesday December 6th @ 11:59am

ADMINISTRIVIA

Monday Dec 4th – NuoDB
→ Barry Morris (Co-Founder, Exec. Chairman)

Wednesday Dec 6th – Potpourri + Final Review
→ Vote for what system you want me to talk about.
→ http://cmudb.io/f17-systems

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.

DISTRIBUTED DBMSs

Use the building blocks that we covered in single-node DBMSs to now support transaction processing & query execution in distributed environments.

→ Optimization & Planning
→ Concurrency Control
→ Logging & Recovery

OLTP VS. OLAP

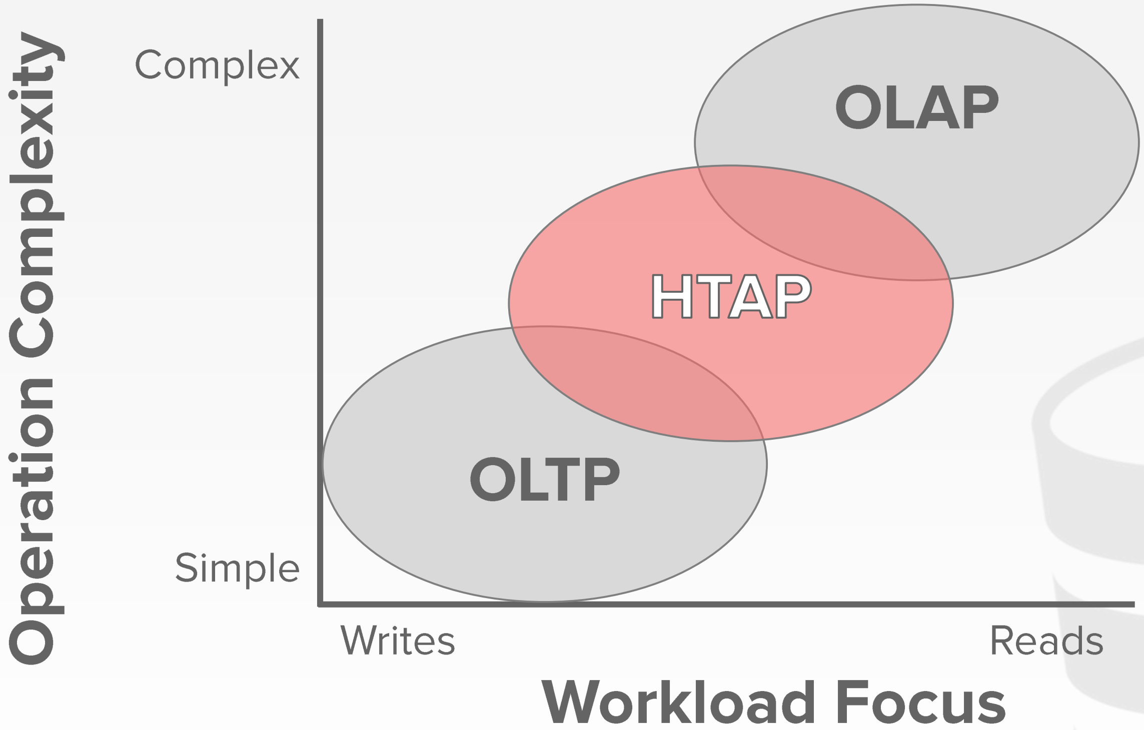
On-line Transaction Processing (OLTP):

→ Short-lived txns.
→ Small footprint.
→ Repetitive operations.

On-line Analytical Processing (OLAP):

→ Long running queries.
→ Complex joins.
→ Exploratory queries.

WORKLOAD CHARACTERIZATION

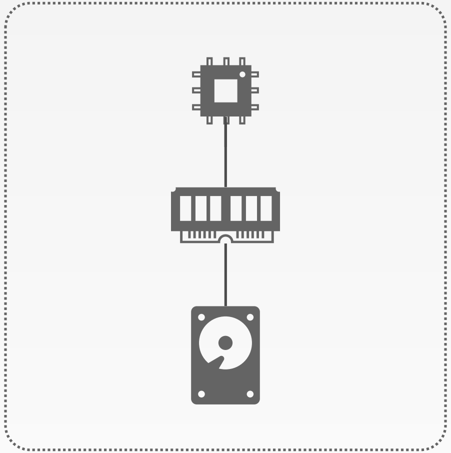


Source: Michael Stonebraker

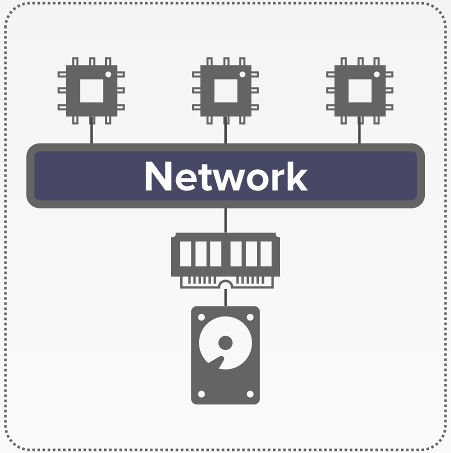
TODAY'S AGENDA

System Architectures
Design Issues
Distributed Concurrency Control
Replication
CAP Theorem

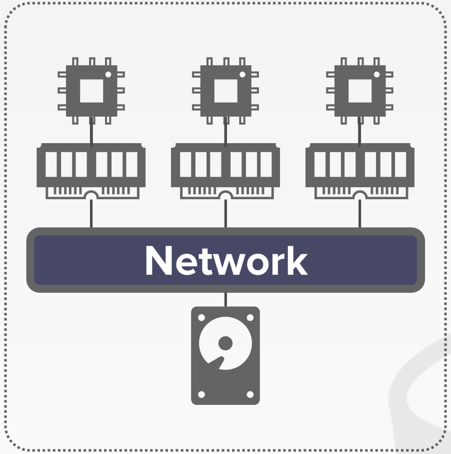
SYSTEM ARCHITECTURES



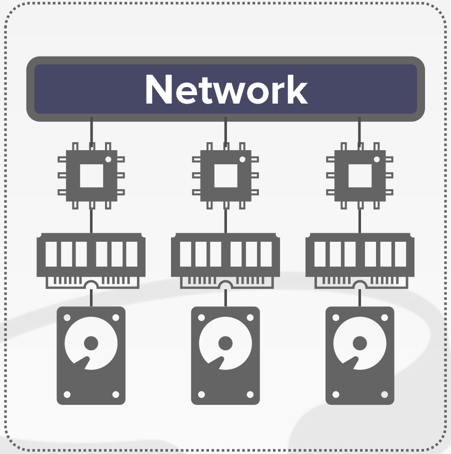
Shared Everything



Shared Memory



Shared Disk



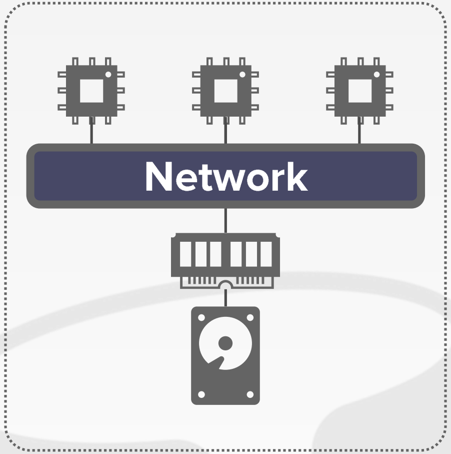
Shared Nothing

SHARED MEMORY

CPUs have access to common memory address space via a fast interconnect.

→ Each processor has a global view of all the in-memory data structures.

→ Each DBMS instance on a processor has to "know" about the other instances.



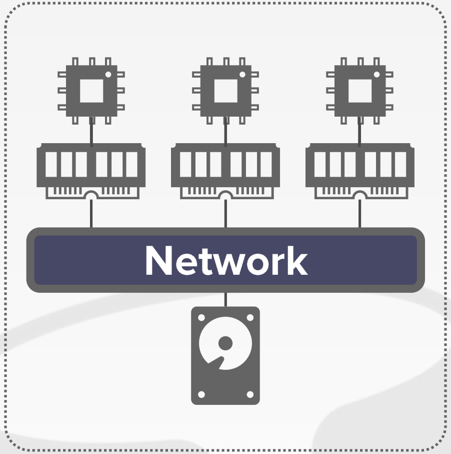
ORACLE RAC

SHARED DISK

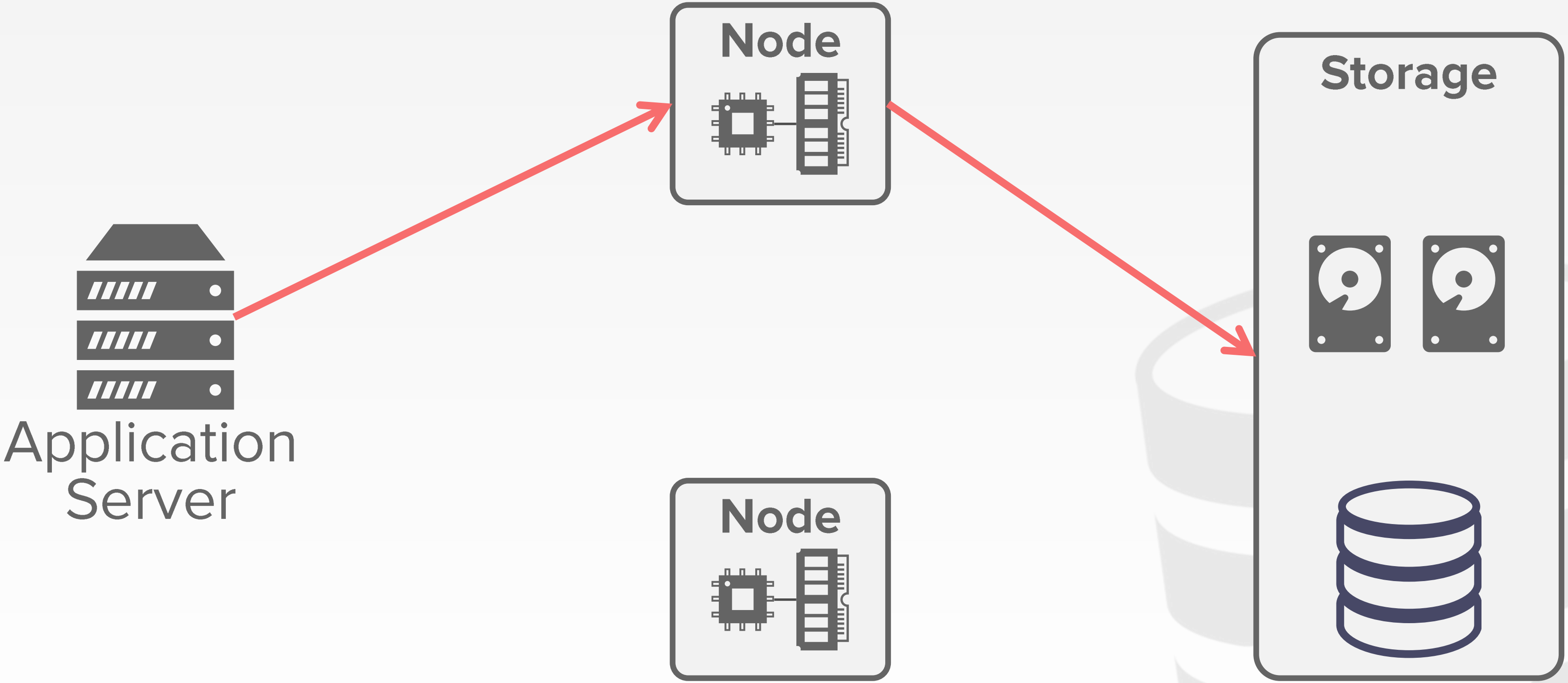
All CPUs can access a single logical disk directly via an interconnect but each have their own private memories.

→ Can scale execution layer independently from the storage layer.

→ Have to send messages between CPUs to learn about their current state.

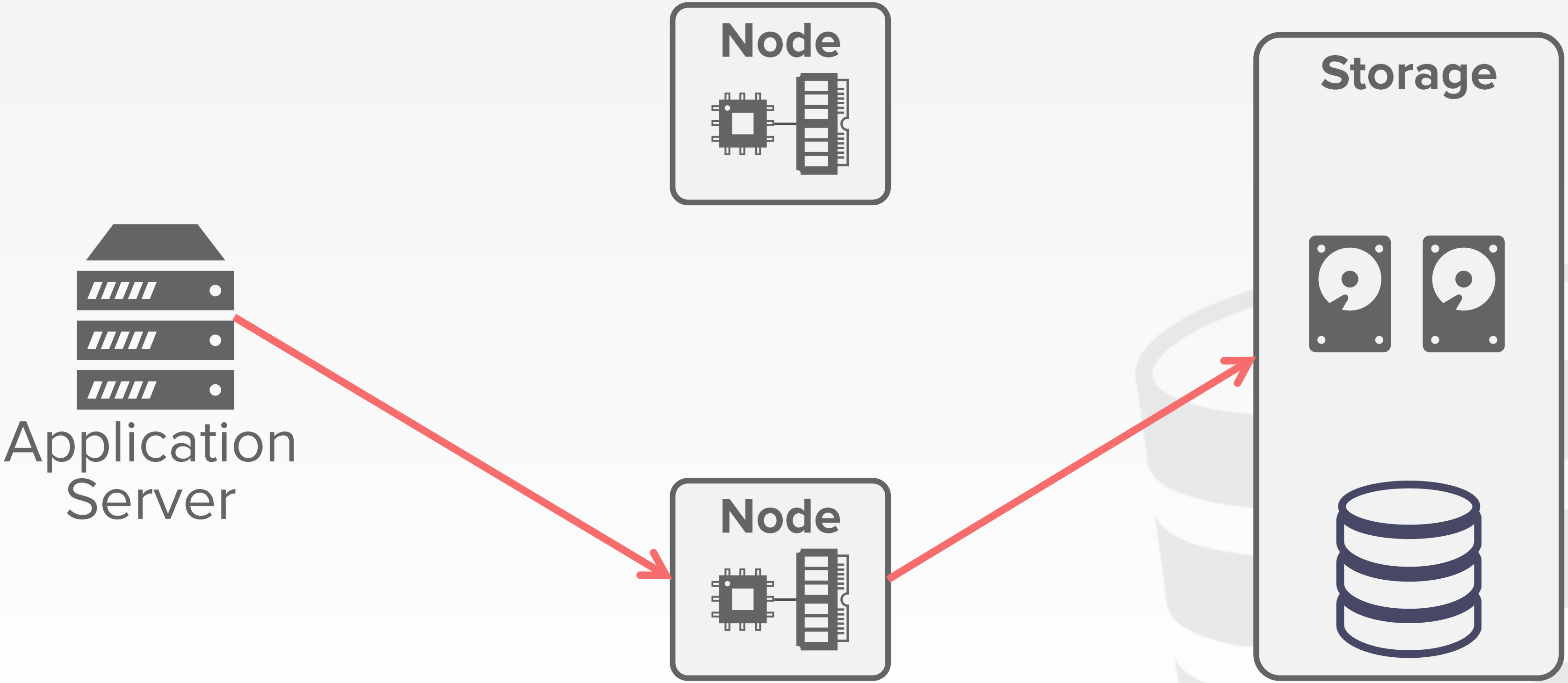


SHARED DISK EXAMPLE

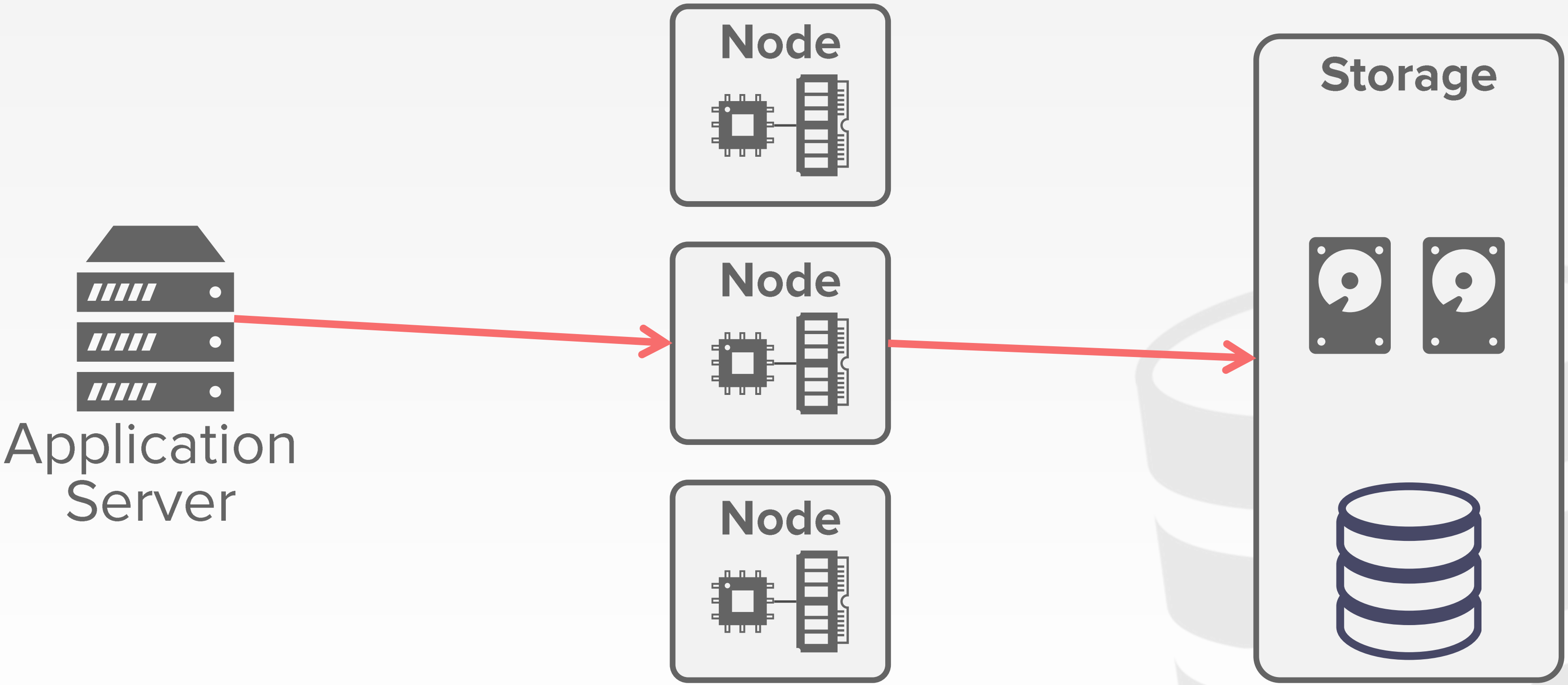


CARNEGIE MELLON DATABASE GROUP

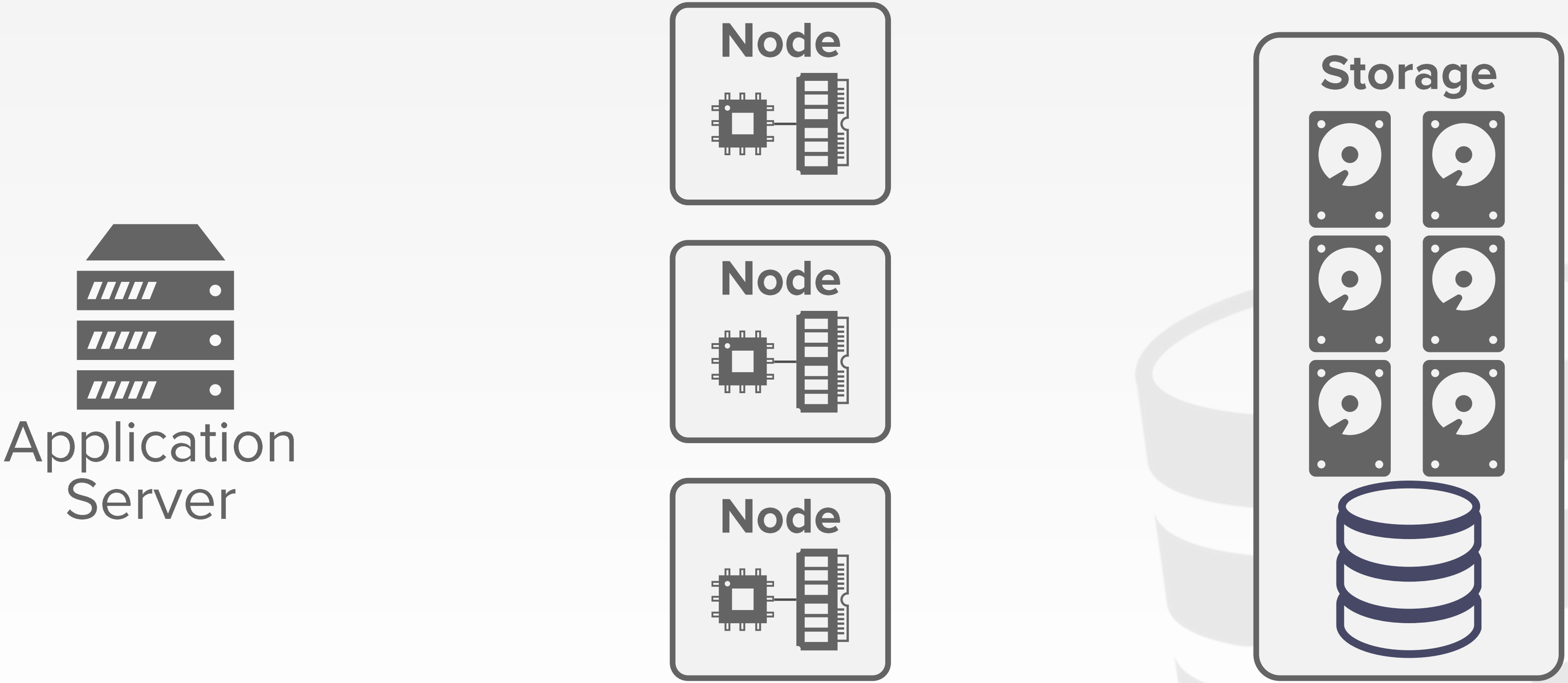
SHARED DISK EXAMPLE



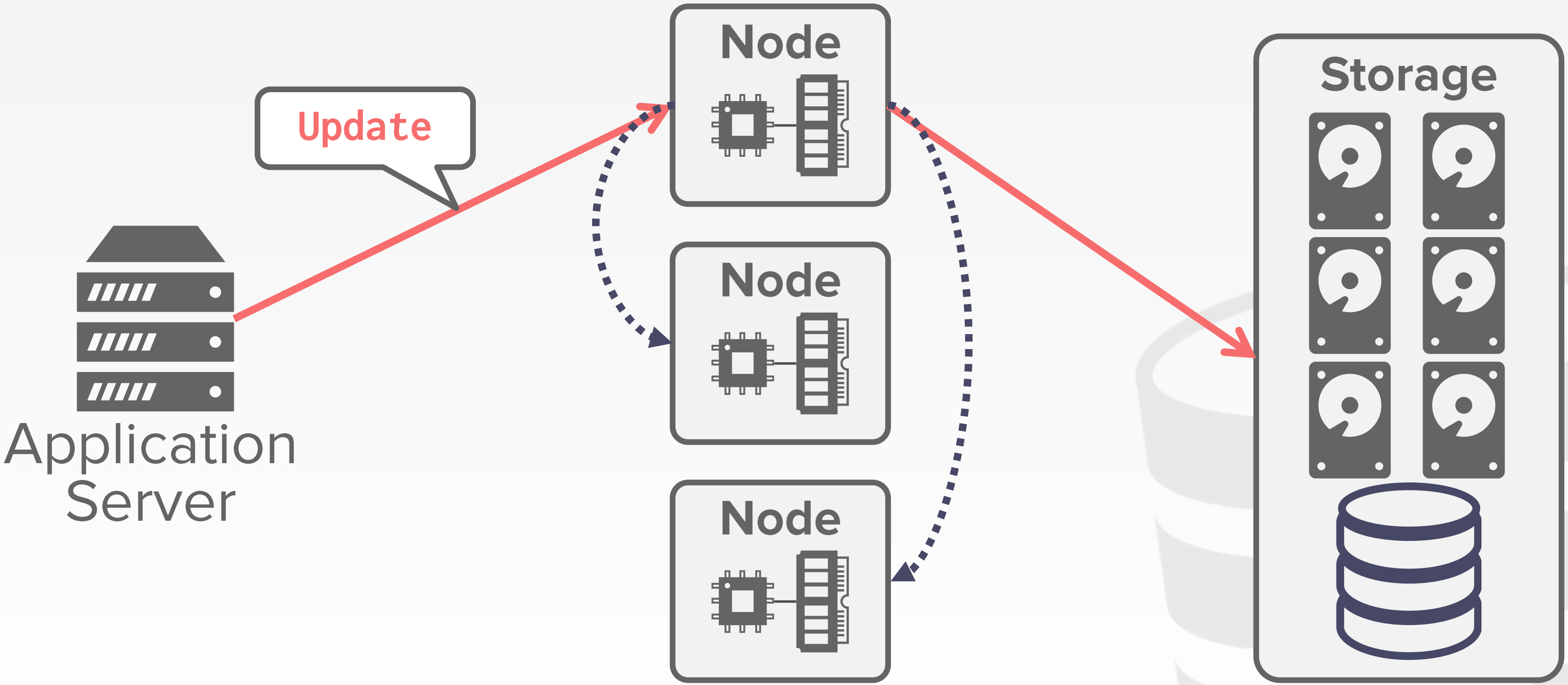
SHARED DISK EXAMPLE



SHARED DISK EXAMPLE



SHARED DISK EXAMPLE



CARNEGIE MELLON DATABASE GROUP

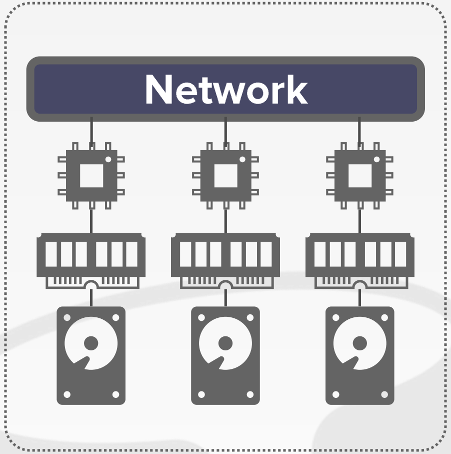
SHARED NOTHING

Each DBMS instance has its own CPU, memory, and disk.

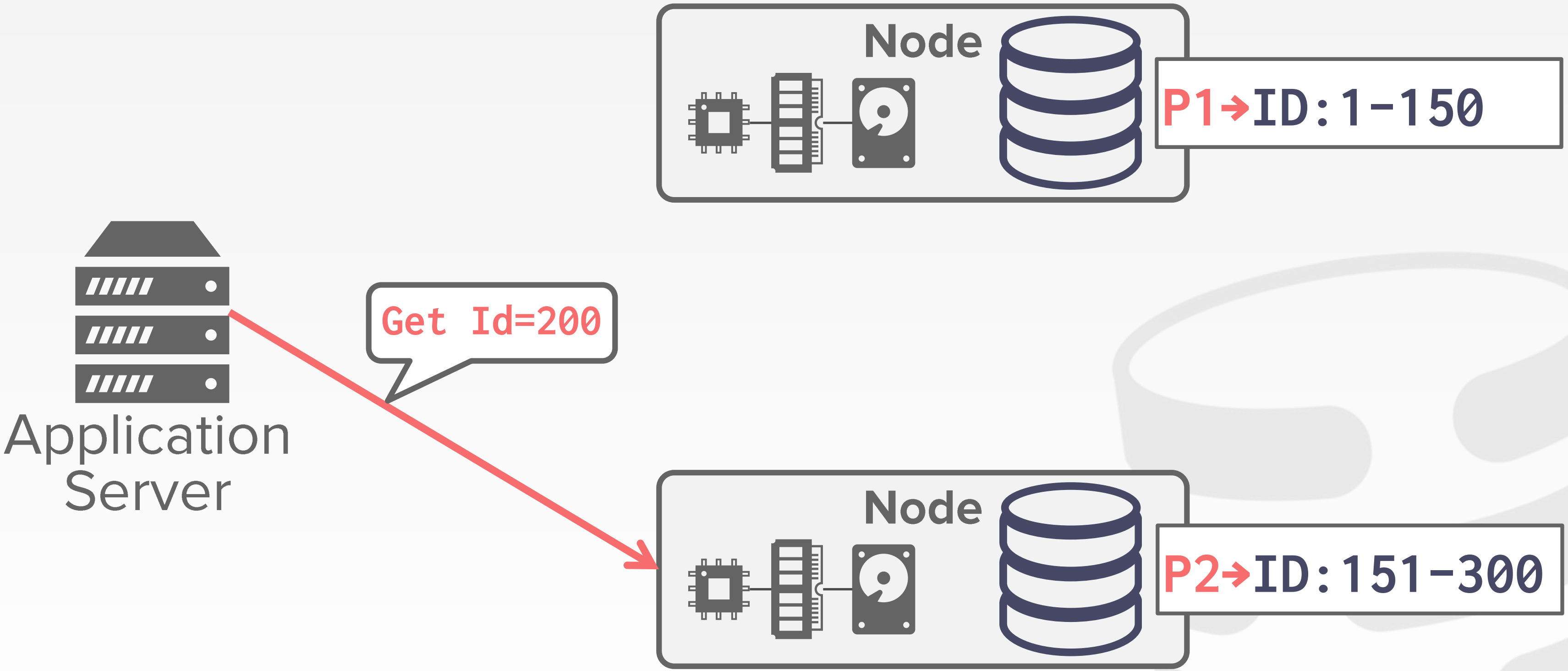
Nodes only communicate with each other via network.

→ Easy to increase capacity.

→ Hard to ensure consistency.

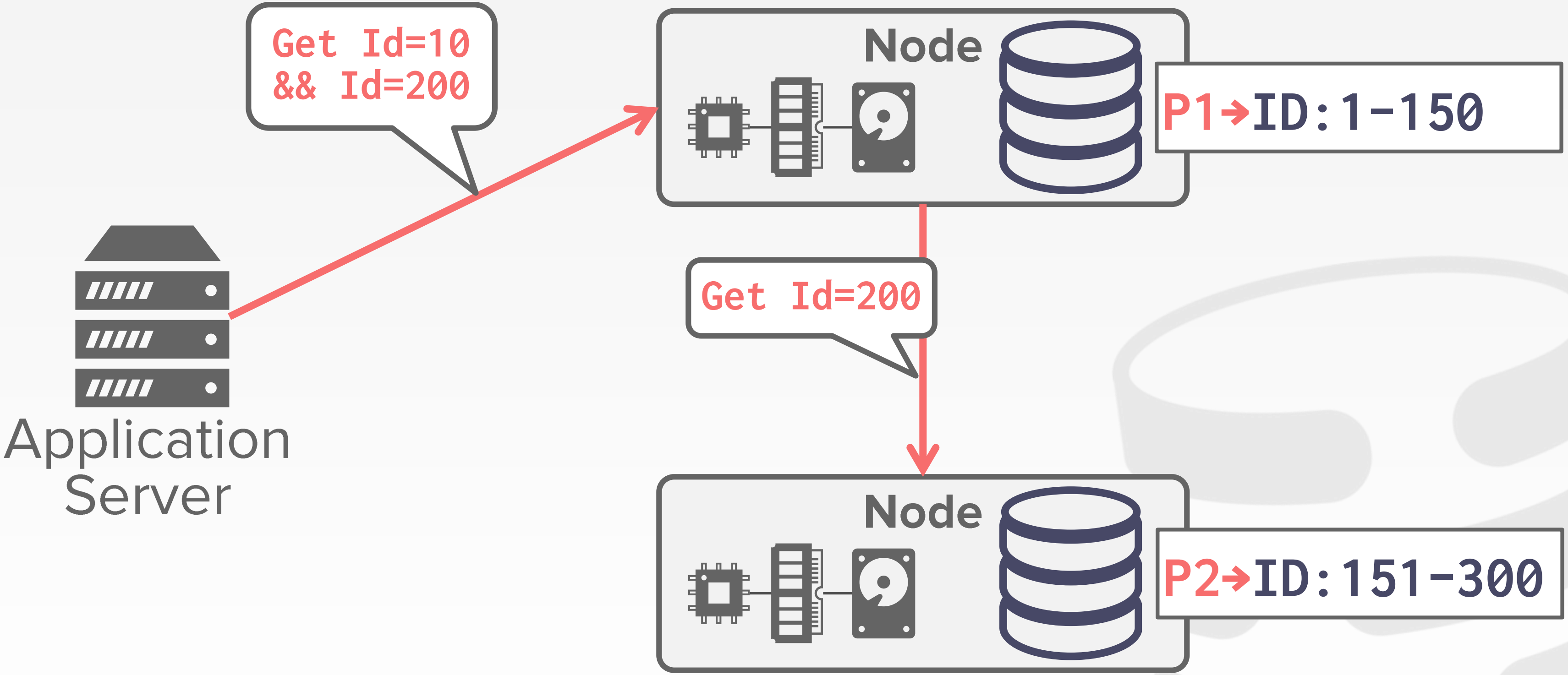


SHARED NOTHING EXAMPLE

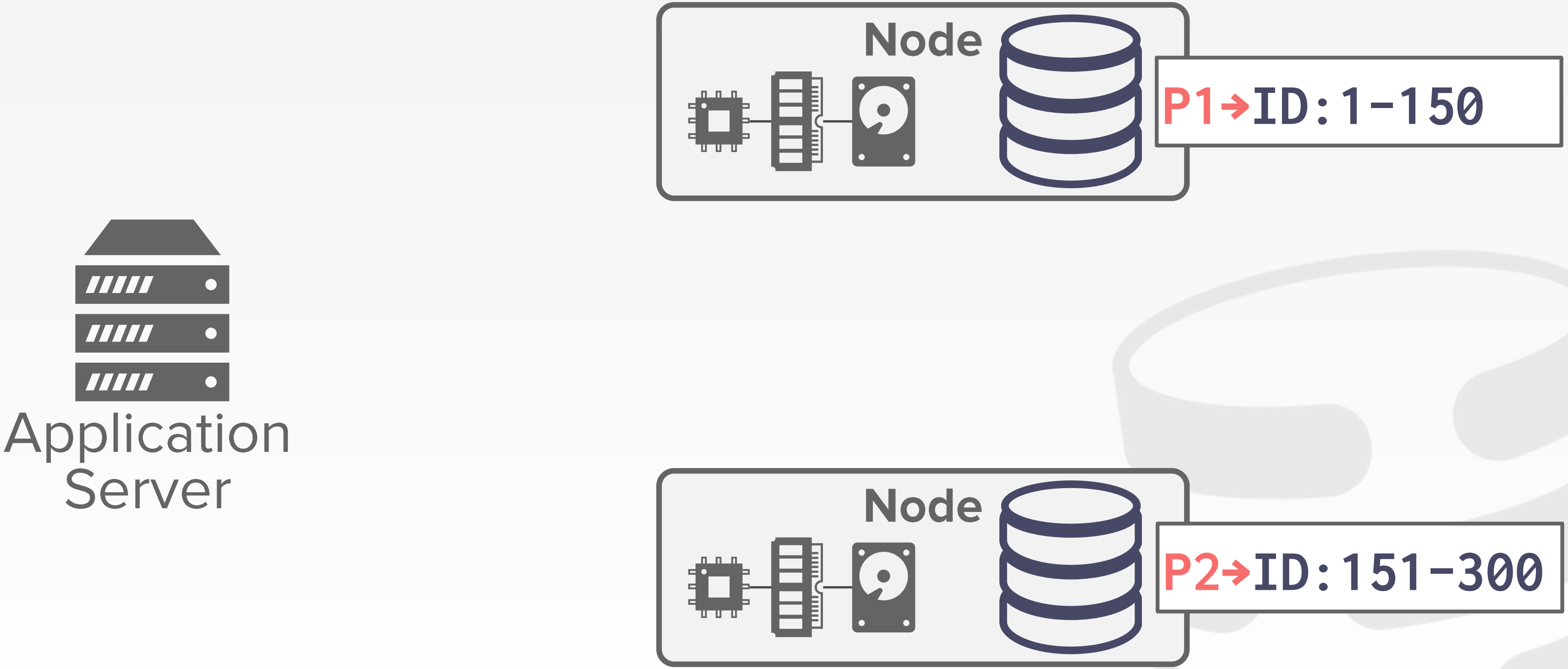


CARNEGIE MELLON DATABASE GROUP

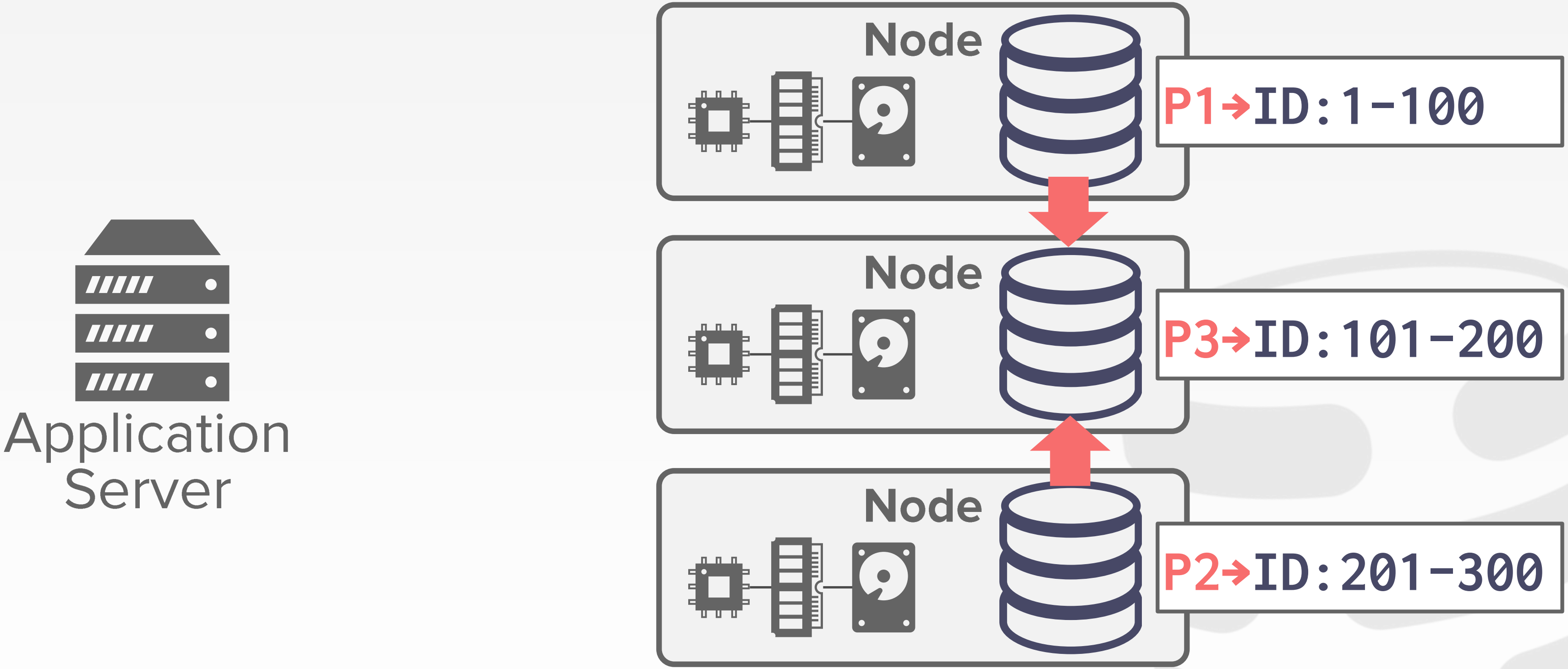
SHARED NOTHING EXAMPLE



SHARED NOTHING EXAMPLE



SHARED NOTHING EXAMPLE



EARLY DISTRIBUTED DATABASE SYSTEMS

MUFFIN – UC Berkeley (1979)

SDD-1 – CCA (1979)

System R* – IBM Research (1984)

Gamma – Univ. of Wisconsin (1986)

NonStop SQL – Tandem (1987)

Stonebraker

Bernstein

Mohan

DeWitt

Gray

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?

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.

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.

HORIZONTAL PARTITIONING

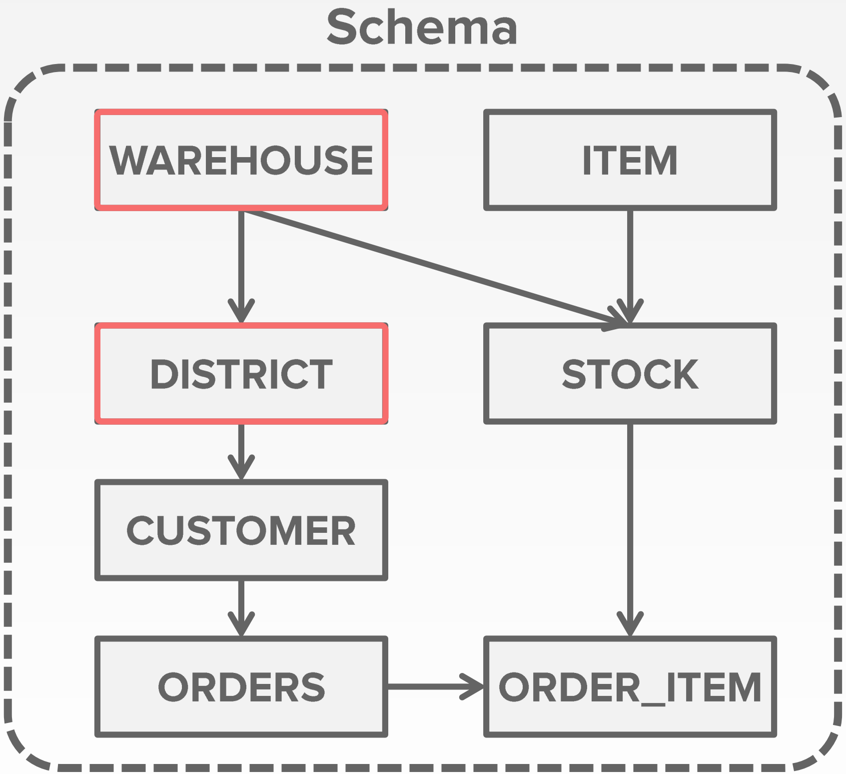
Split a table's tuples into disjoint subsets.

→ 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.

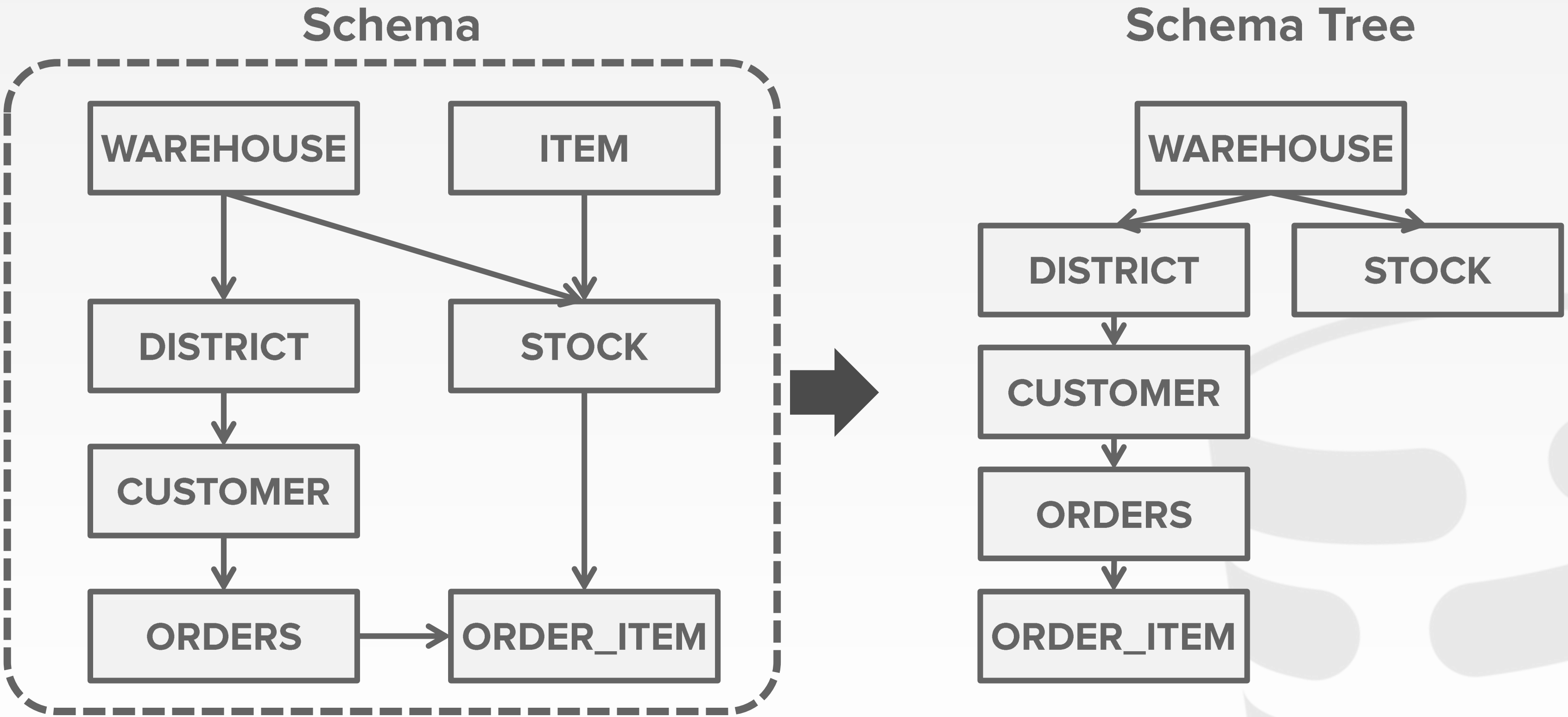
DATABASE PARTITIONING



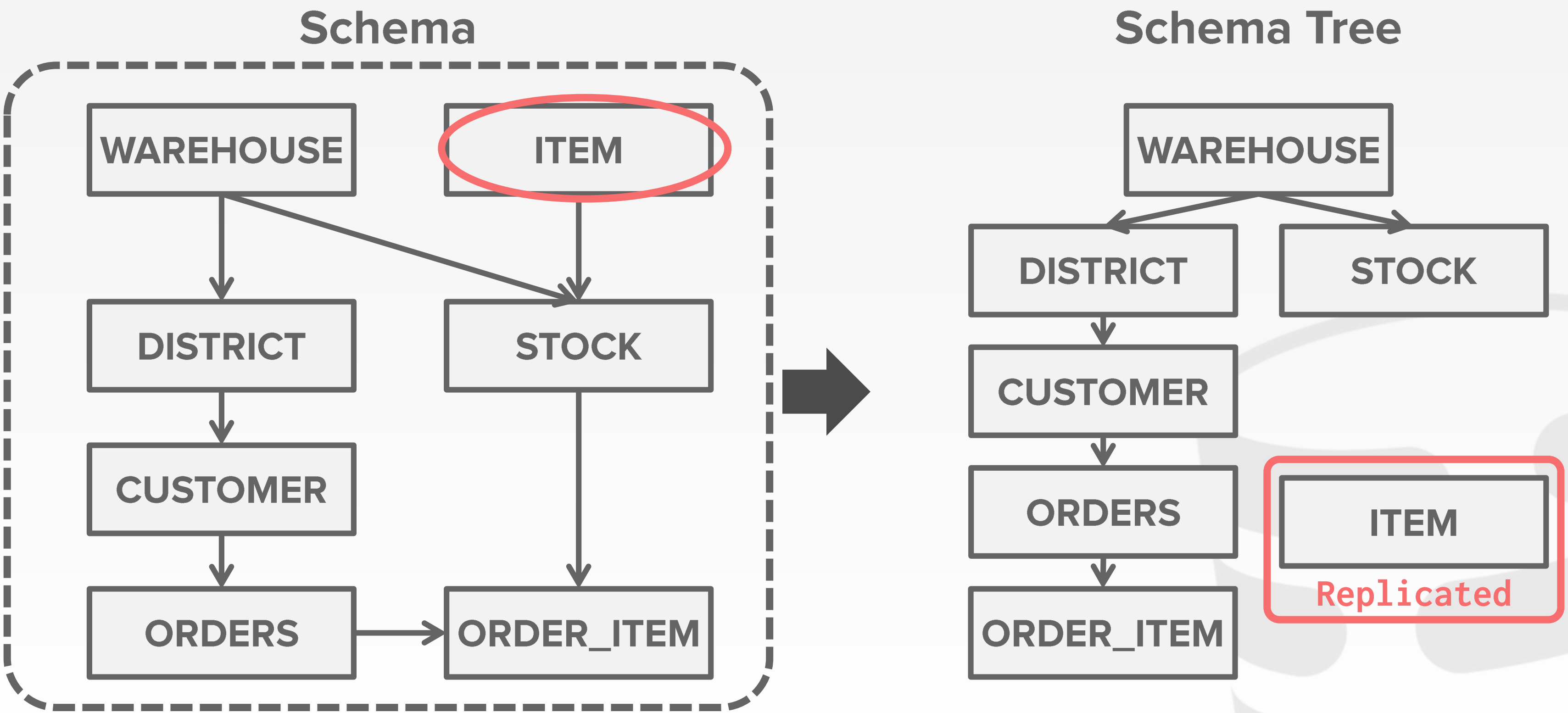

```sql
CREATE TABLE WAREHOUSE (
  w_id INT PRIMARY KEY,
  w_name VARCHAR UNIQUE,
  ⋮
);
```

```sql
CREATE TABLE DISTRICT (
  d_id INT,
  d_w_id INT REFERENCES
             ↪ WAREHOUSE (w_id),
  ⋮
  PRIMARY KEY (d_w_id, d_id)
);
```

DATABASE PARTITIONING



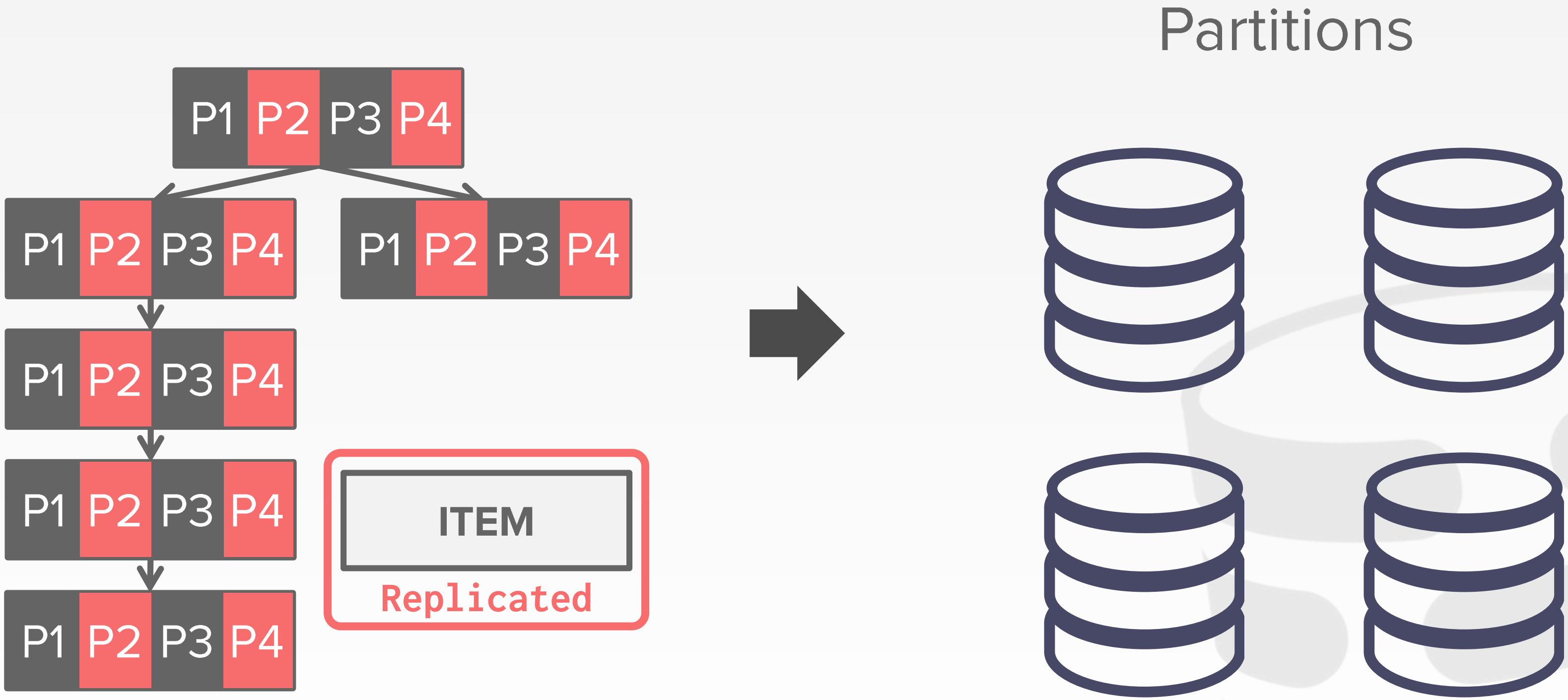
DATABASE PARTITIONING



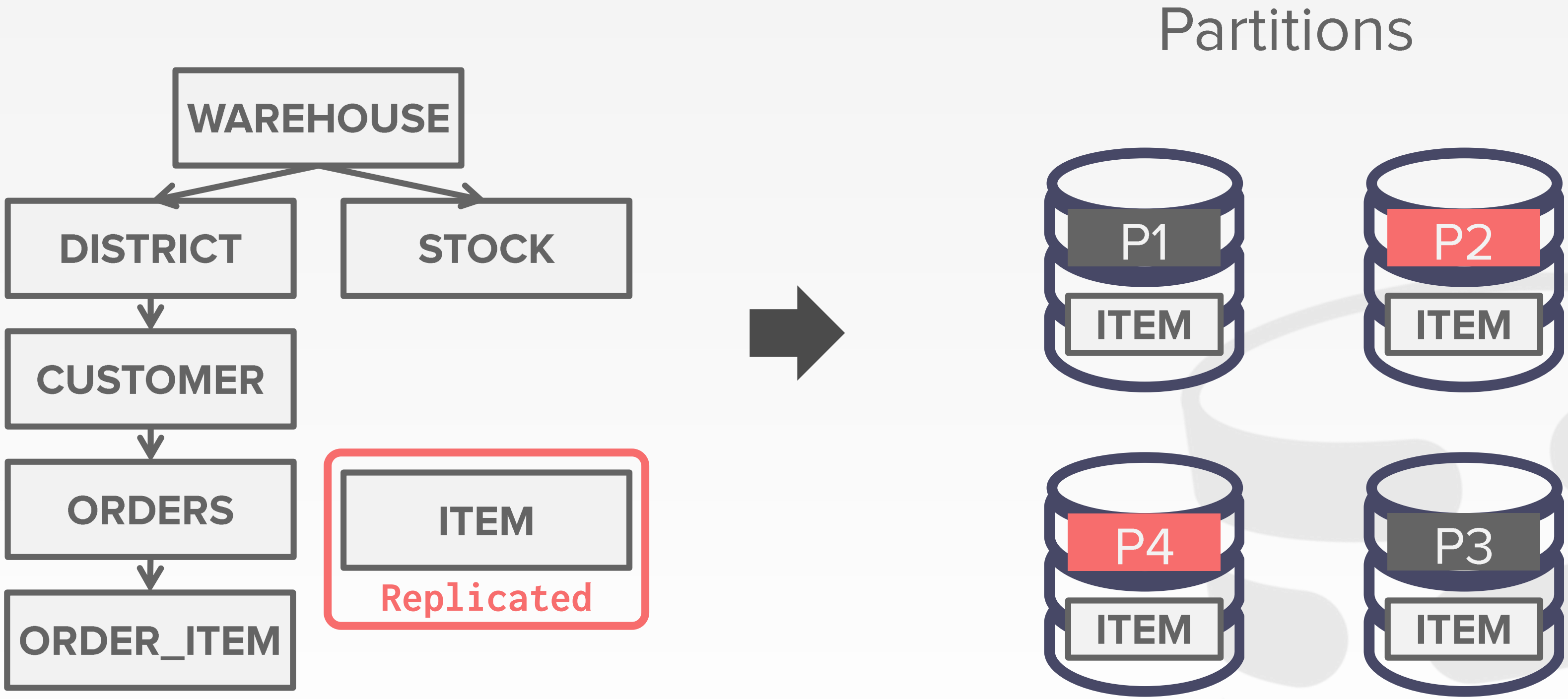
CARNEGIE MELLON
DATABASE GROUP

DATABASE PARTITIONING

DATABASE PARTITIONING

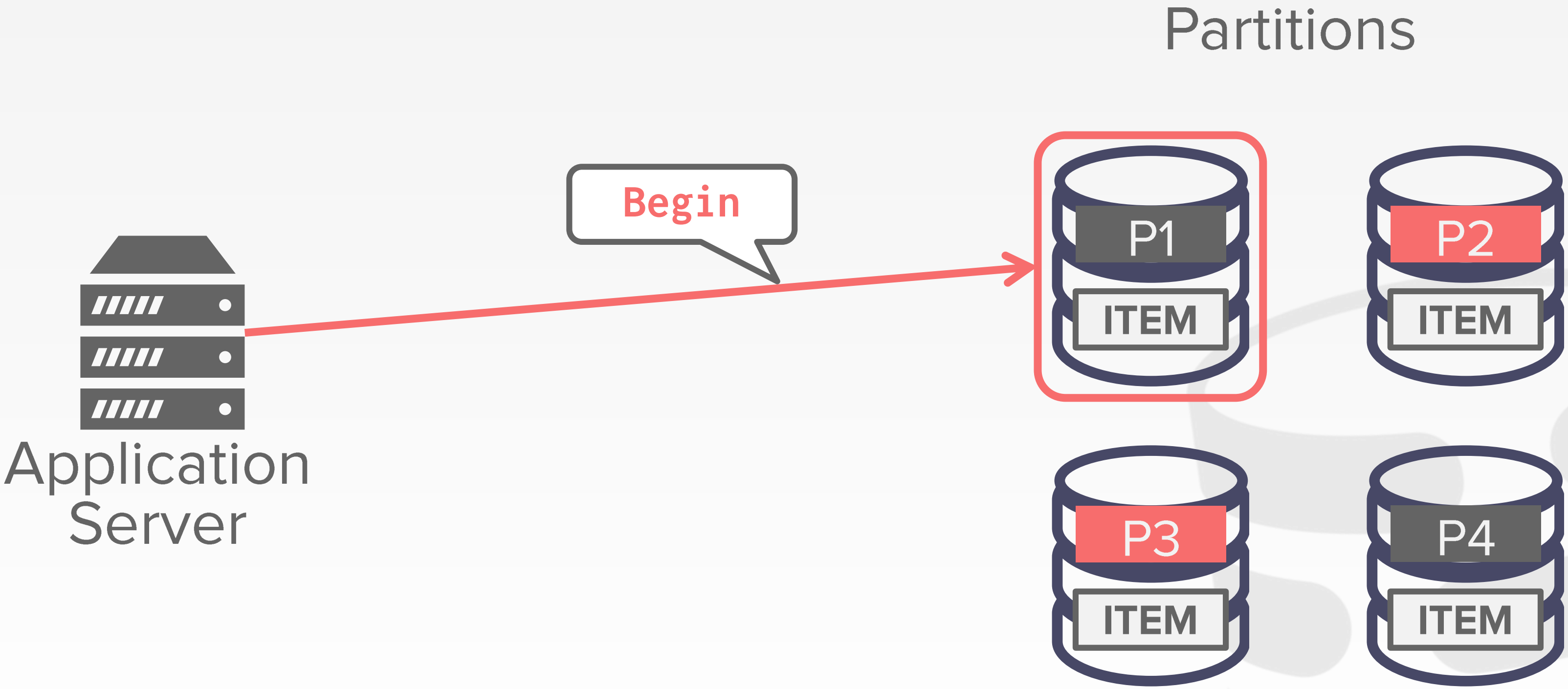


DATABASE PARTITIONING

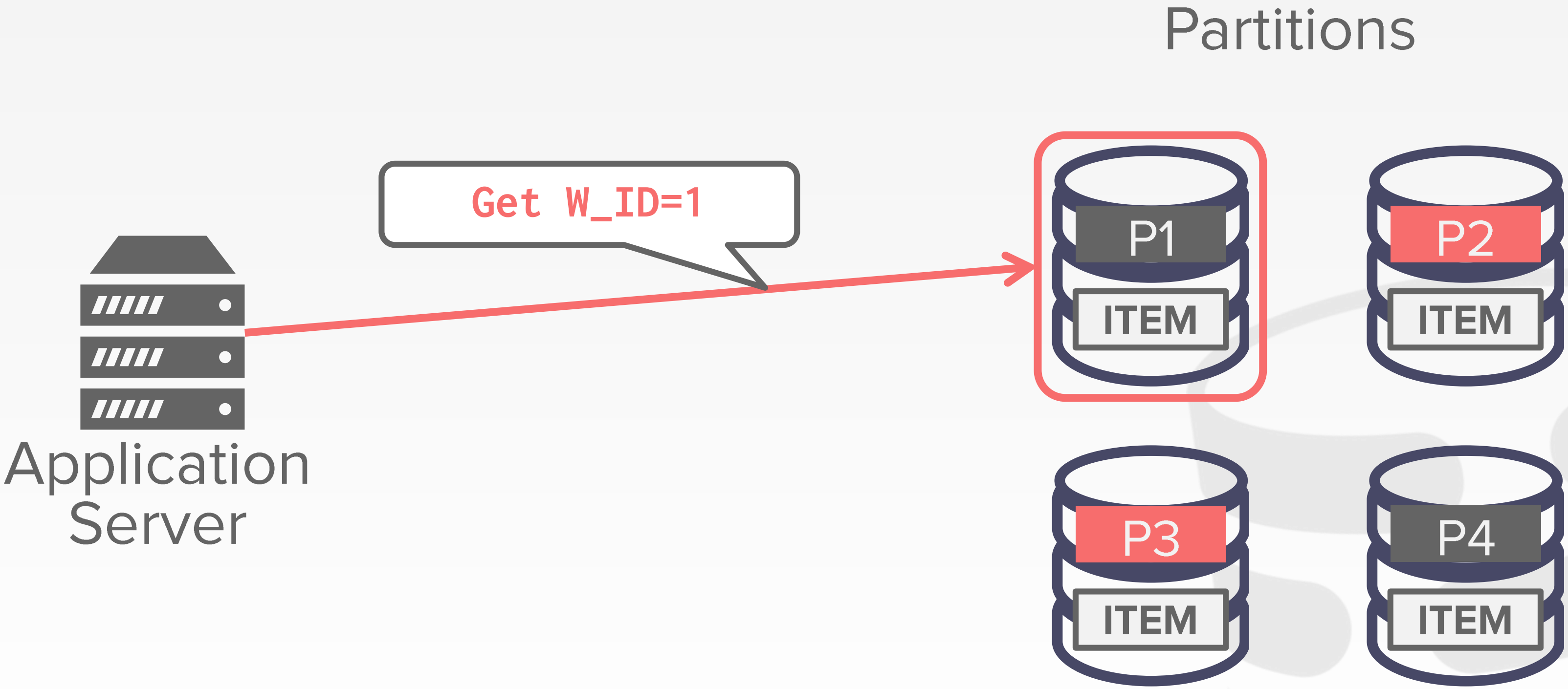


CARNEGIE MELLON DATABASE GROUP

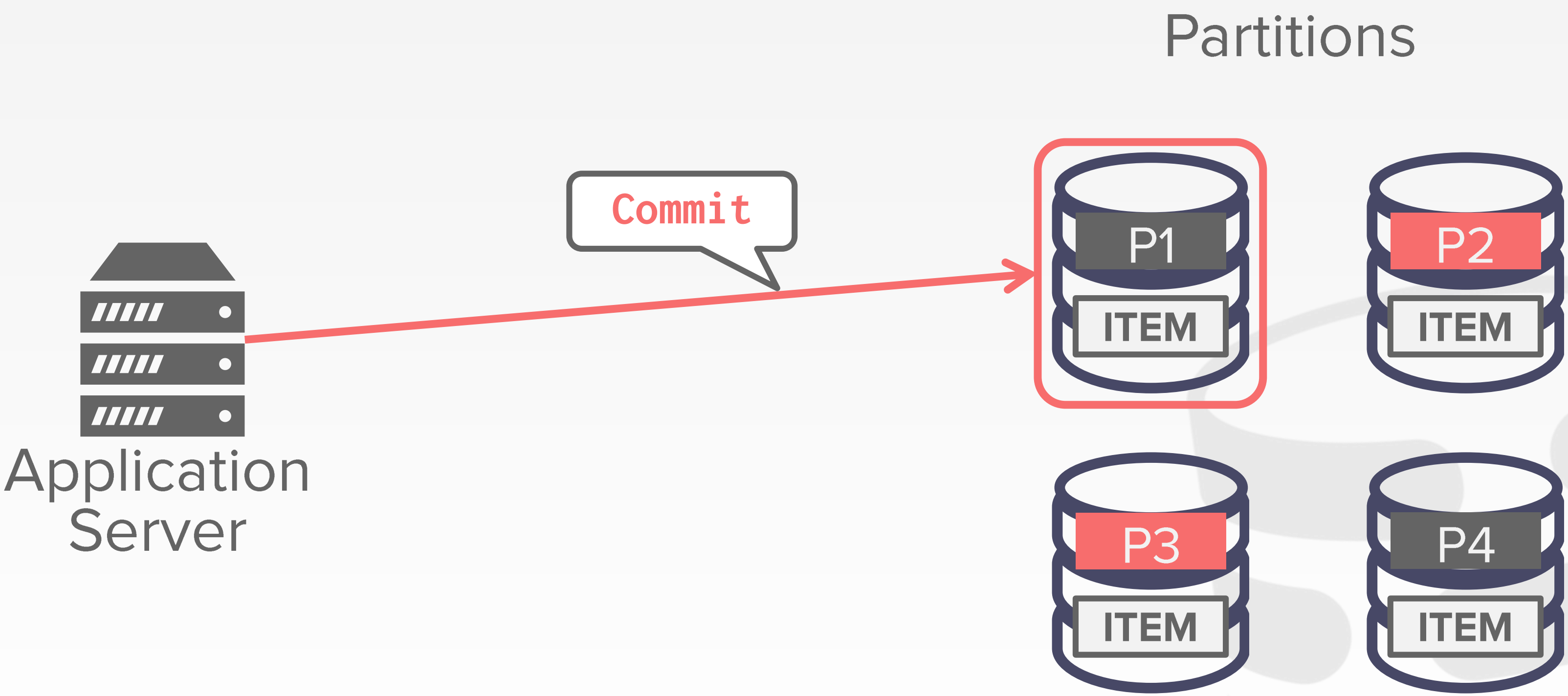
SIMPLE EXAMPLE



SIMPLE EXAMPLE



SIMPLE EXAMPLE



CARNEGIE MELLON DATABASE GROUP

SINGLE-NODE VS. DISTRIBUTED TRANSACTIONS

A **single-node** txn only accesses data that is contained on one partition.

→ The DBMS does not need coordinate the behavior concurrent txns running on other nodes.

A **distributed** txn accesses data at one or more partitions.

→ Requires expensive coordination.

TRANSACTION COORDINATION

If our DBMS supports multi-operation and distributed txns, we need a way to coordinate their execution in the system.

Two different approaches:

→ **Centralized**: Global "traffic cop".

→ **Decentralized**: Nodes organize themselves.

TP MONITORS

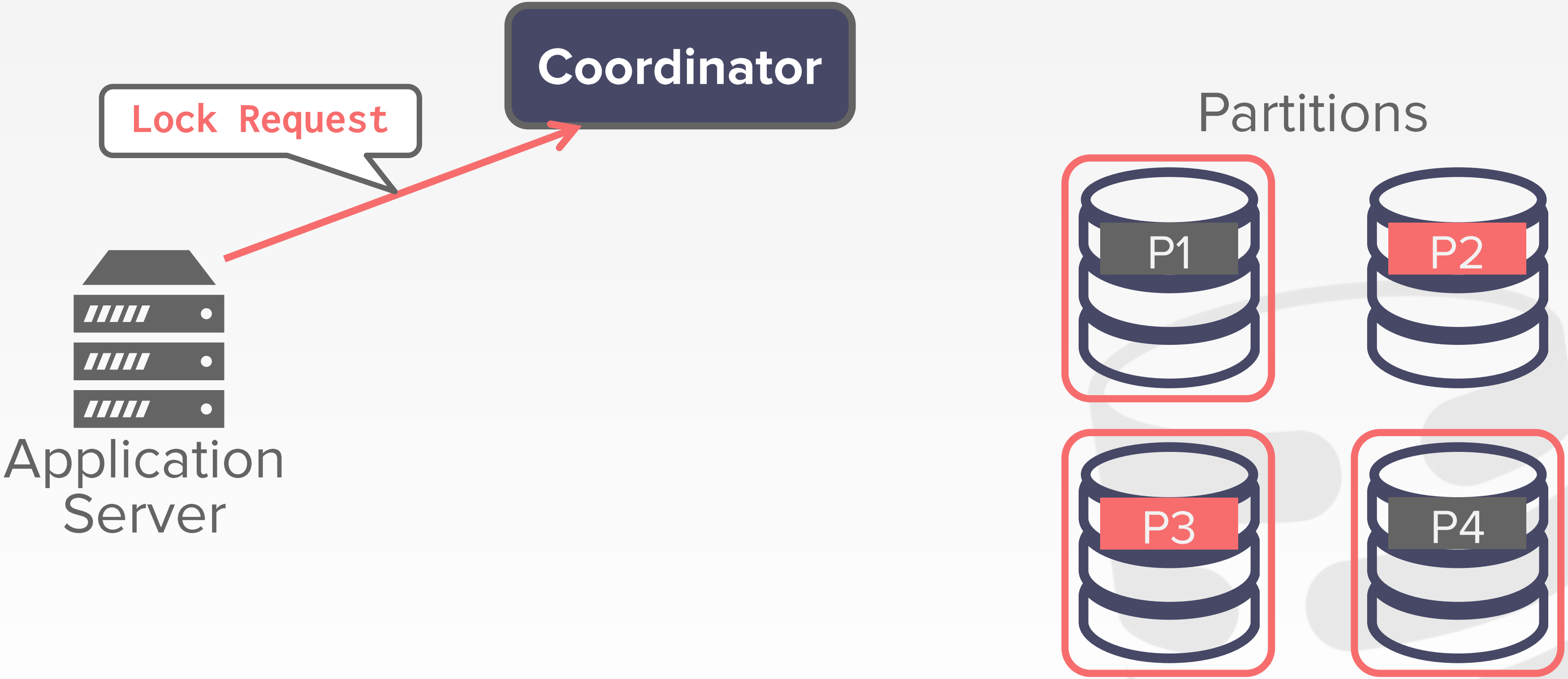
Example of a centralized coordinator.

Originally developed in the 1970-80s to provide txns between terminals and mainframe databases.

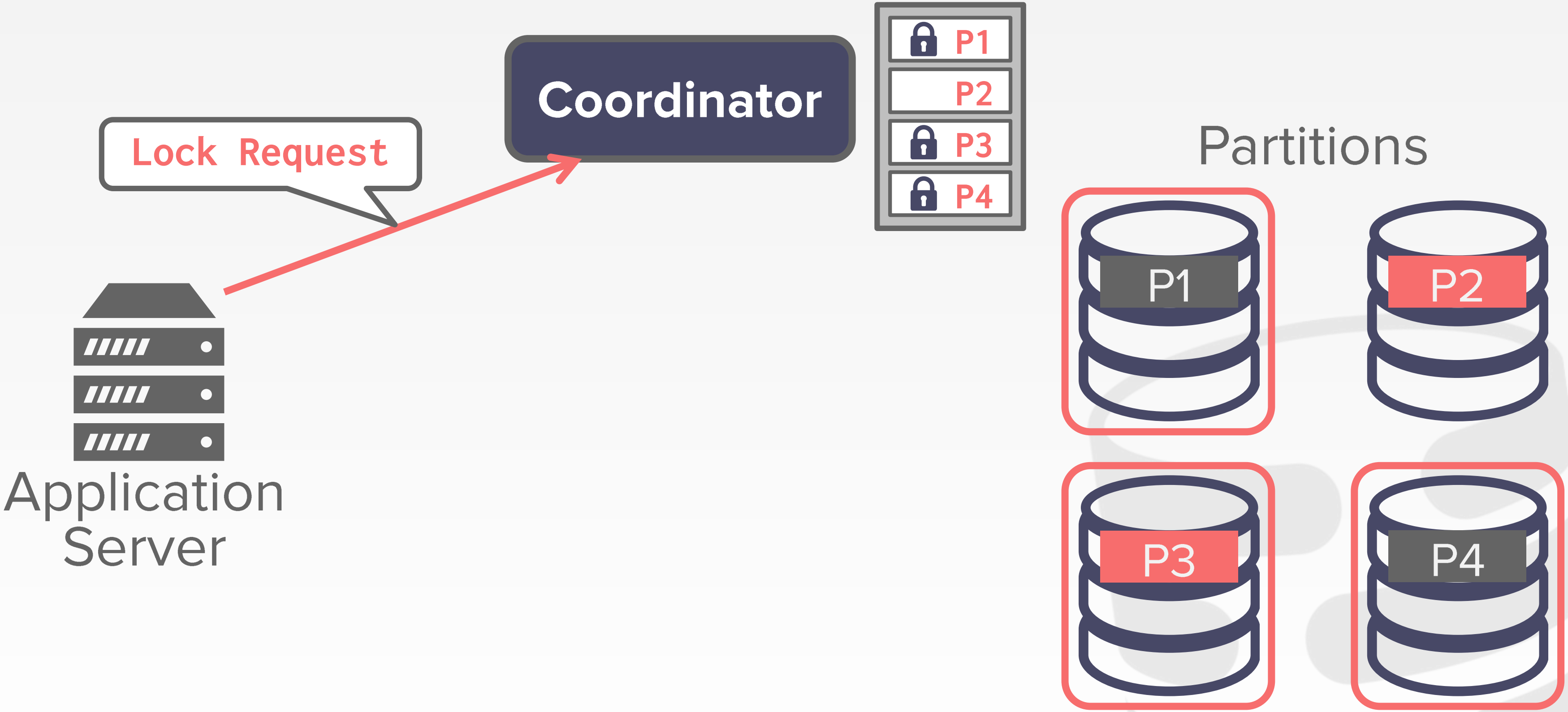
→ Examples: ATMs, Airline Reservations.

Many DBMSs now support the same functionality internally.

CENTRALIZED COORDINATOR

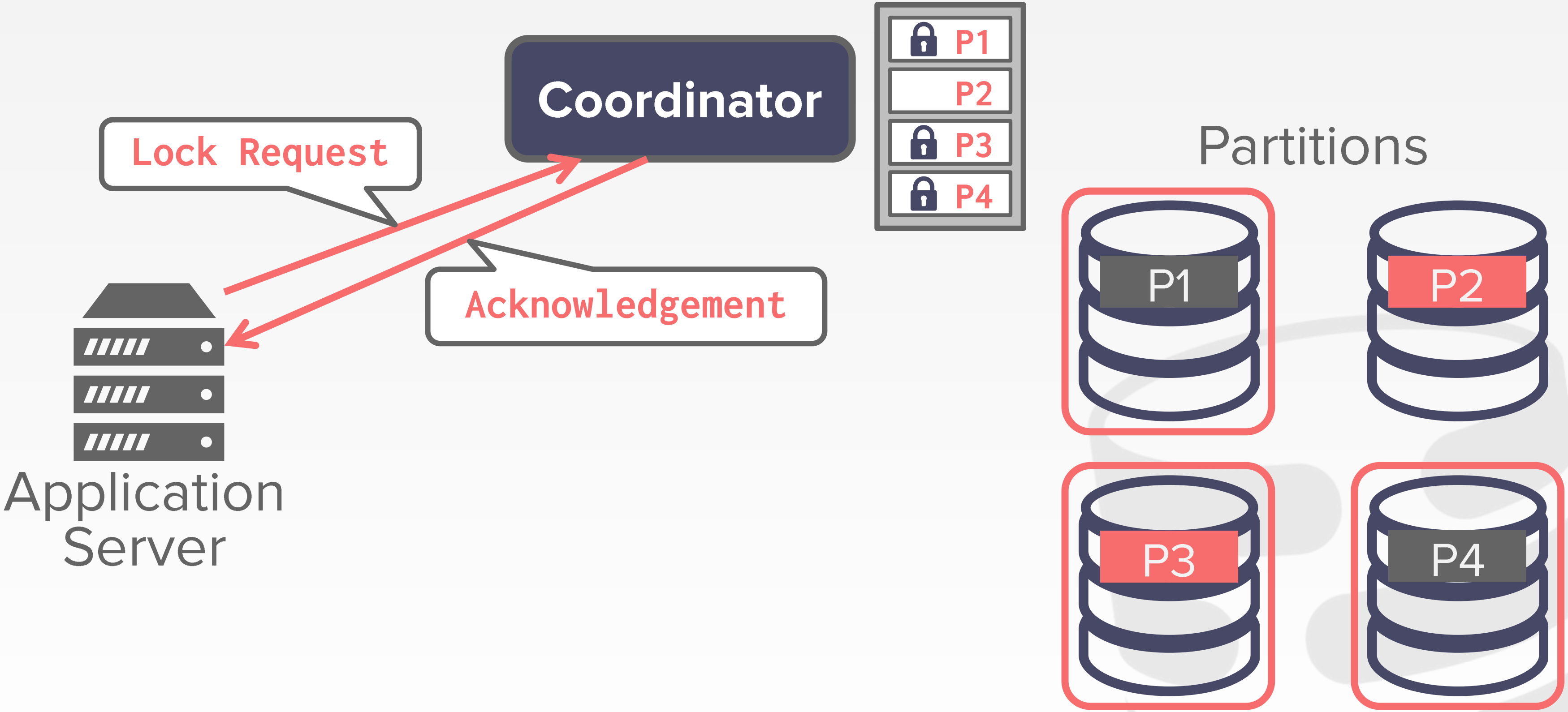


CENTRALIZED COORDINATOR

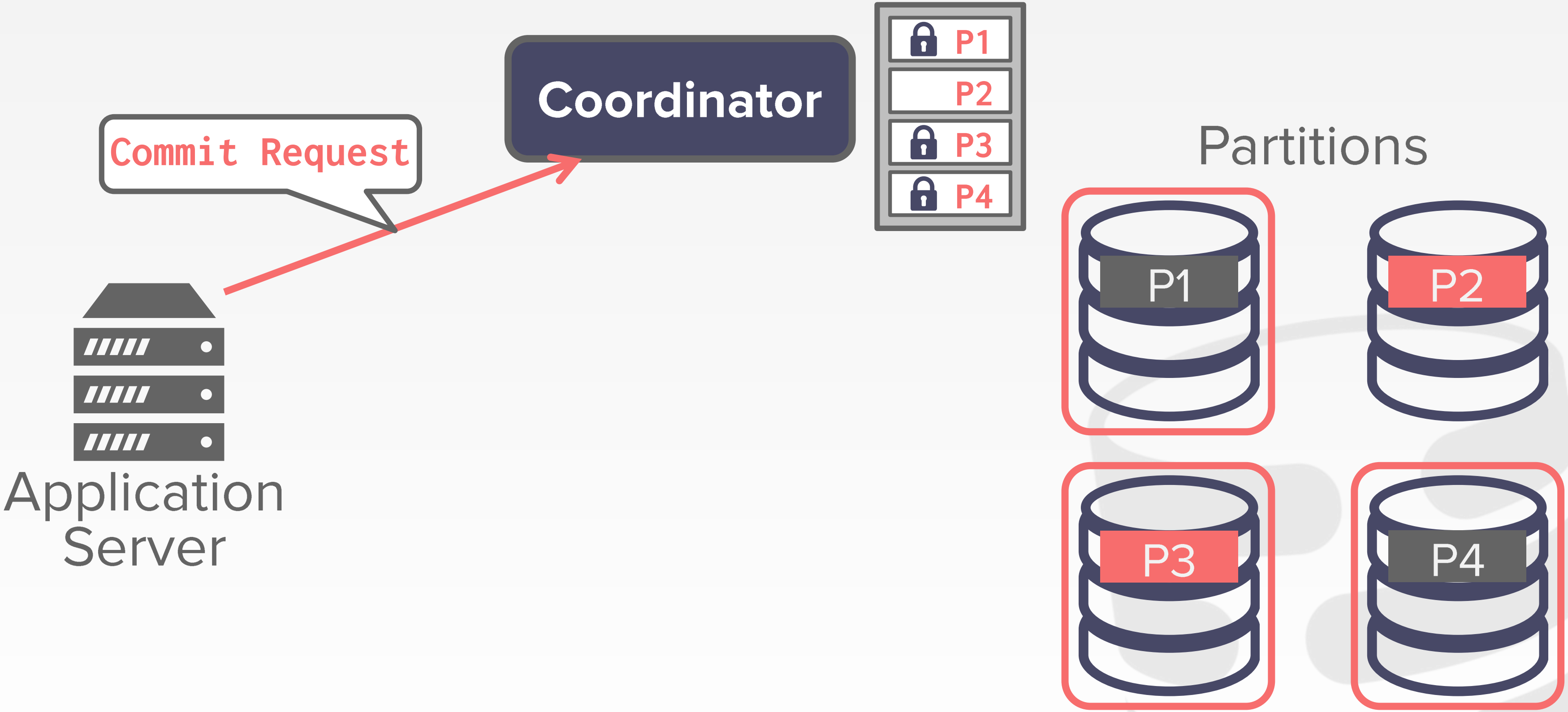


CARNEGIE MELLON DATABASE GROUP

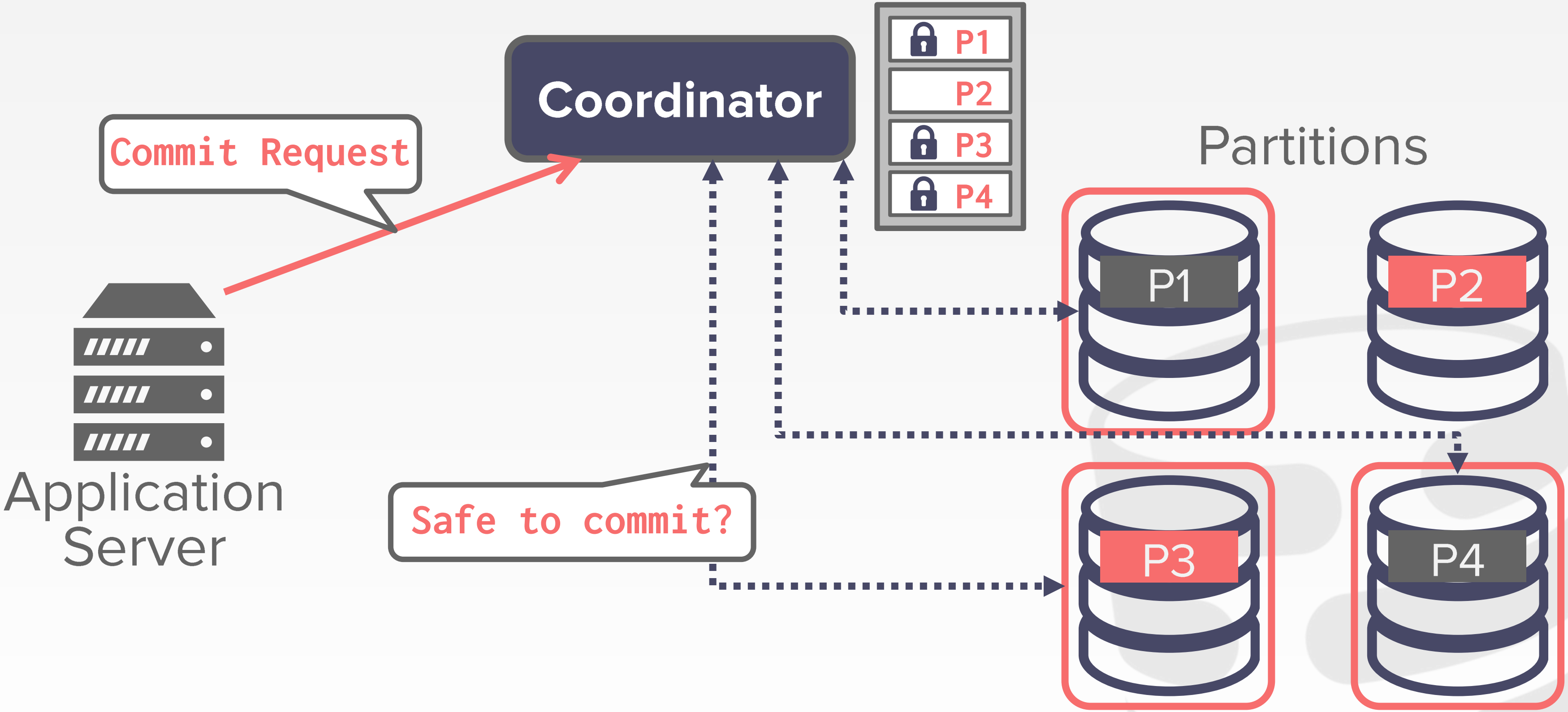
CENTRALIZED COORDINATOR



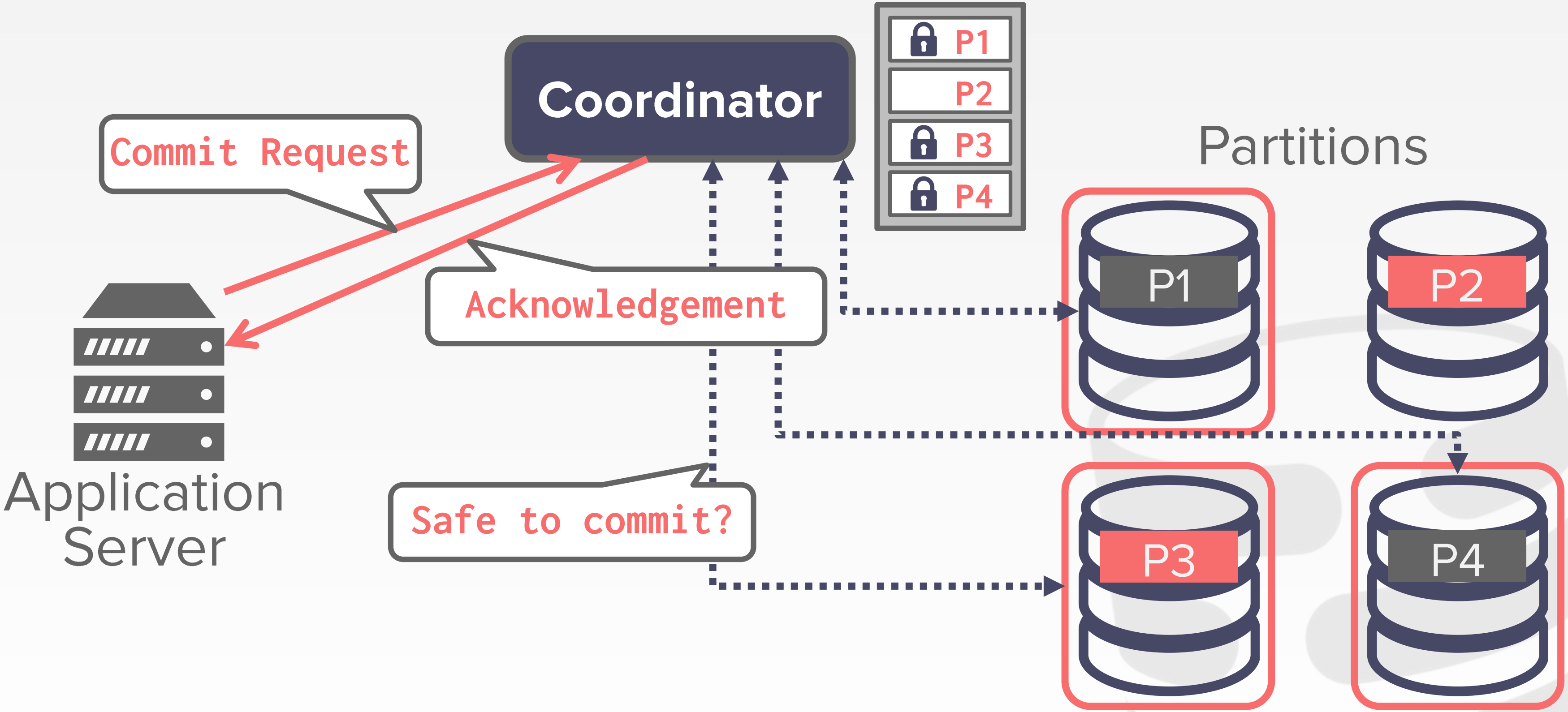
CENTRALIZED COORDINATOR



CENTRALIZED COORDINATOR

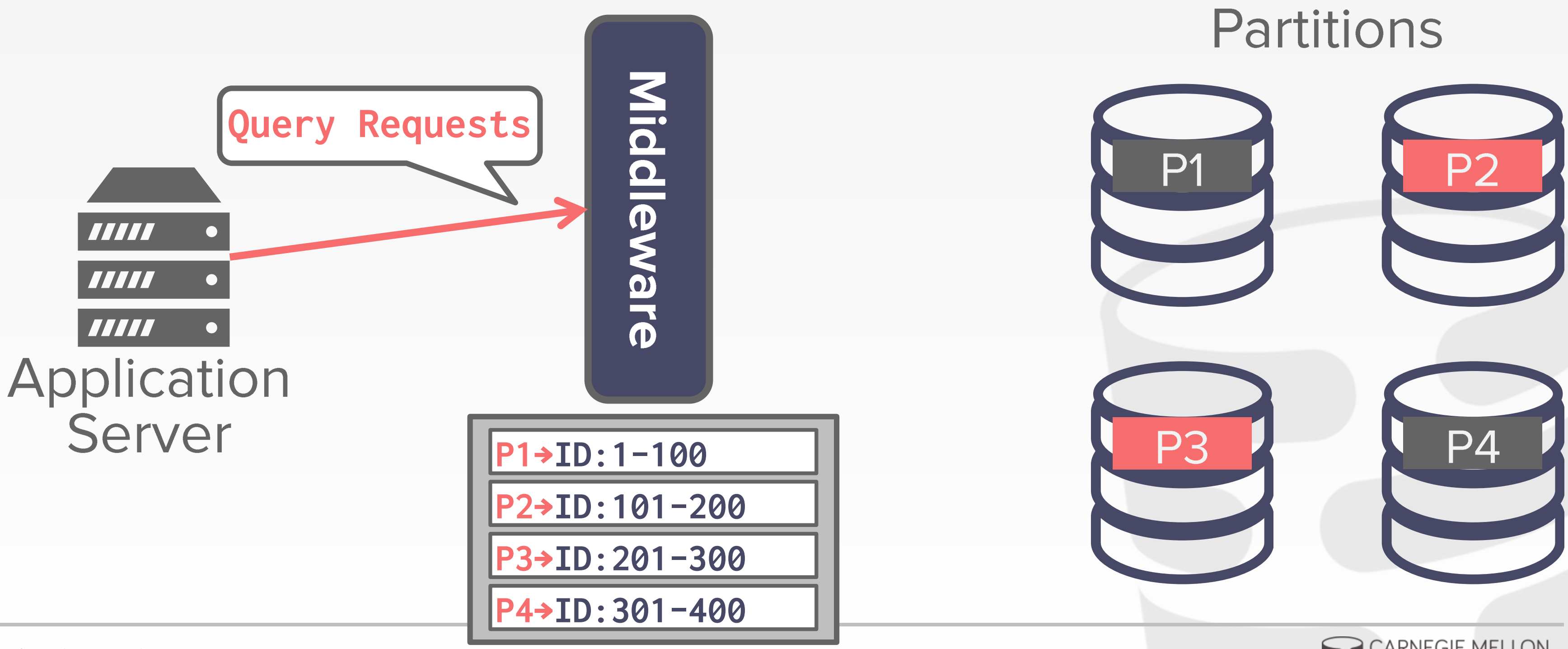


CENTRALIZED COORDINATOR



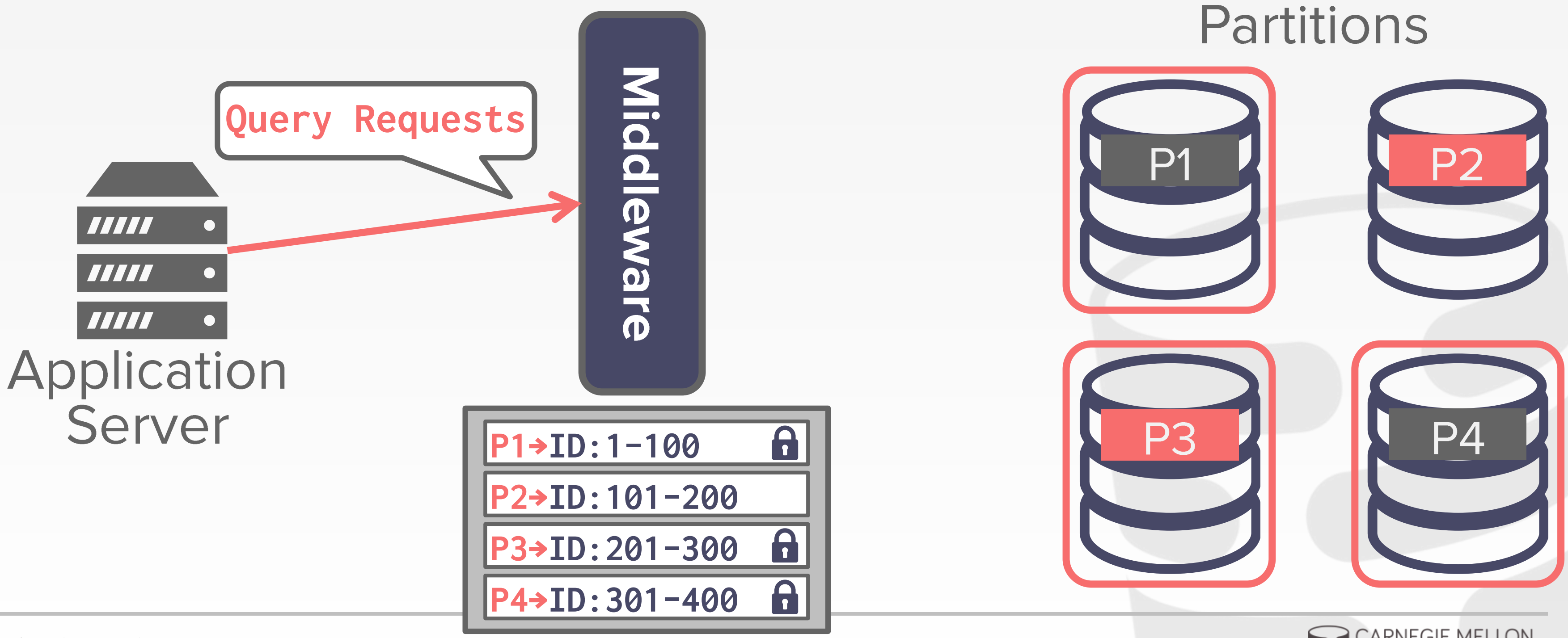
CARNEGIE MELLON DATABASE GROUP

CENTRALIZED COORDINATOR



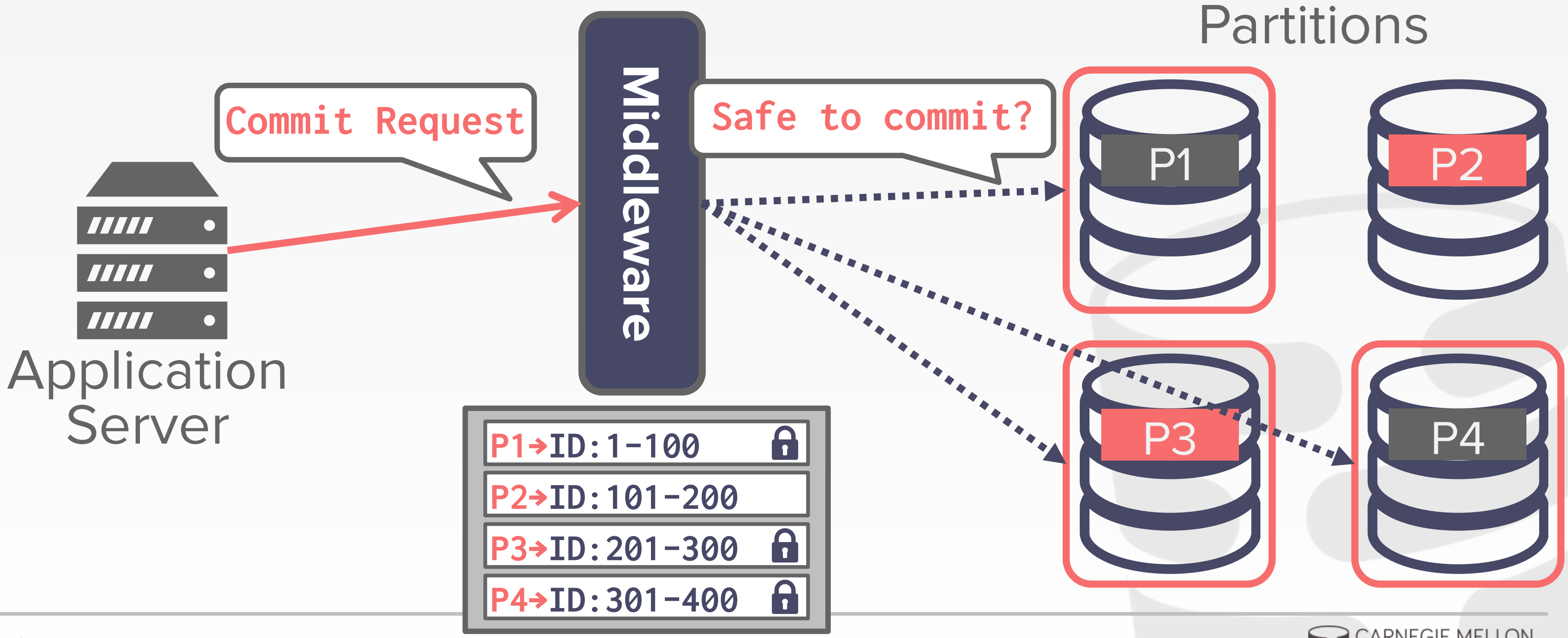
CARNEGIE MELLON DATABASE GROUP

CENTRALIZED COORDINATOR



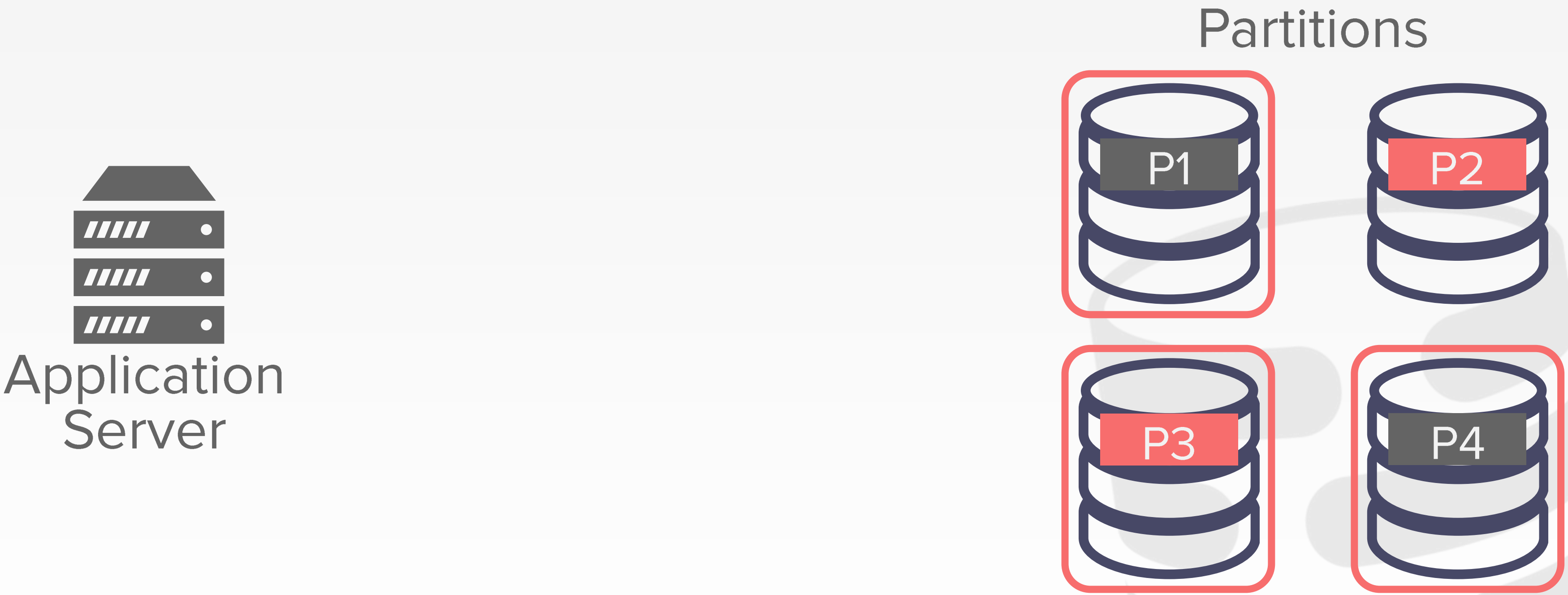
CARNEGIE MELLON DATABASE GROUP

CENTRALIZED COORDINATOR



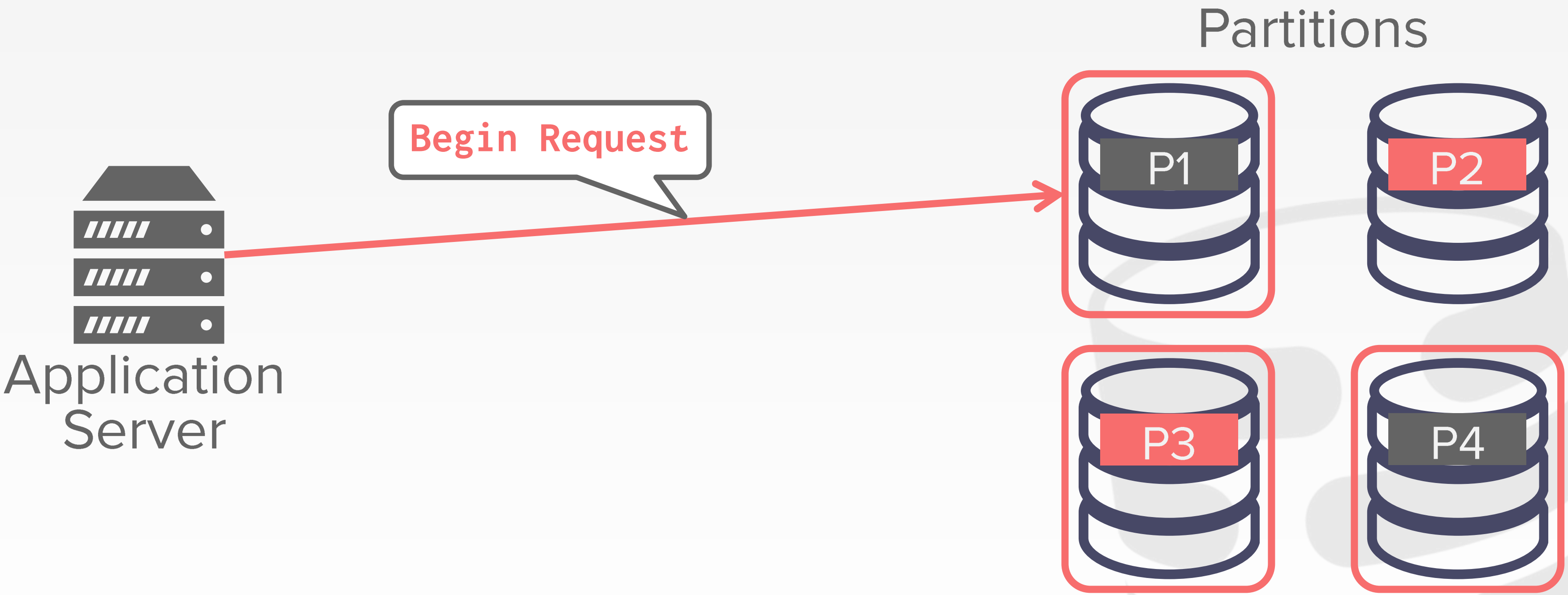
CARNEGIE MELLON DATABASE GROUP

DECENTRALIZED COORDINATOR

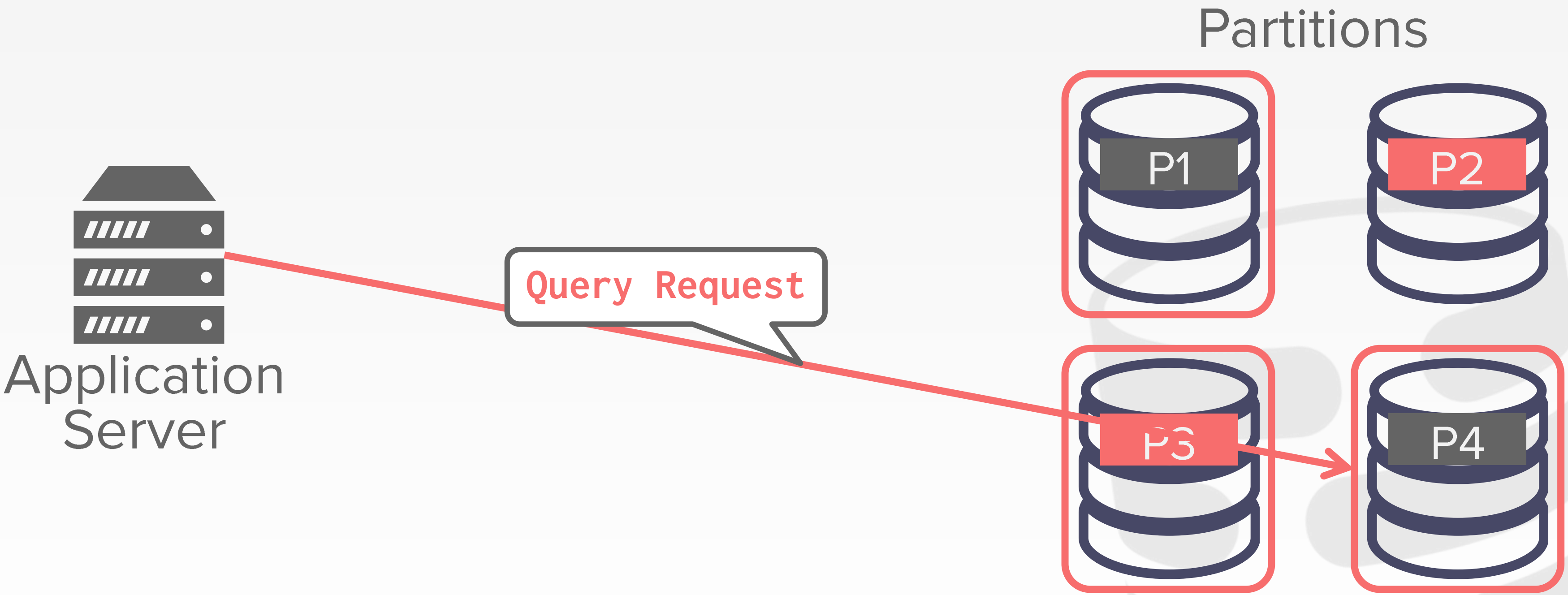


CARNEGIE MELLON DATABASE GROUP

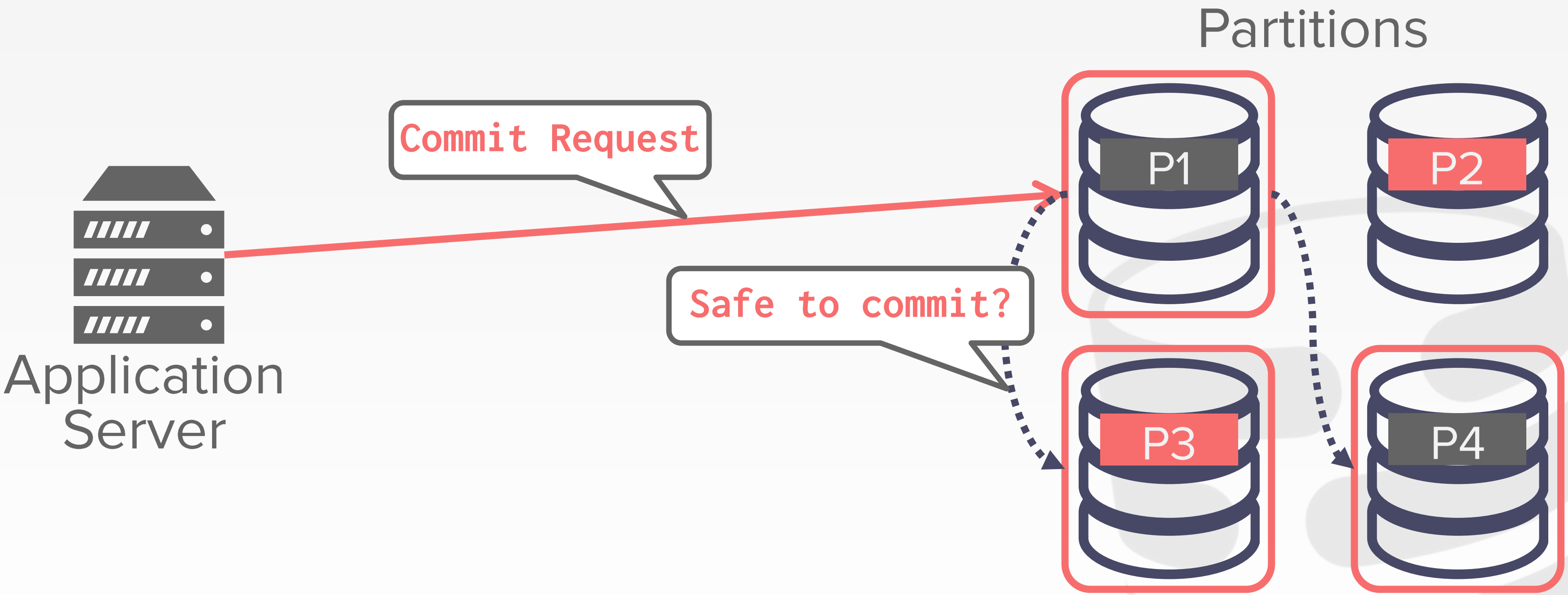
DECENTRALIZED COORDINATOR



DECENTRALIZED COORDINATOR



DECENTRALIZED COORDINATOR



DISTRIBUTED CONCURRENCY CONTROL

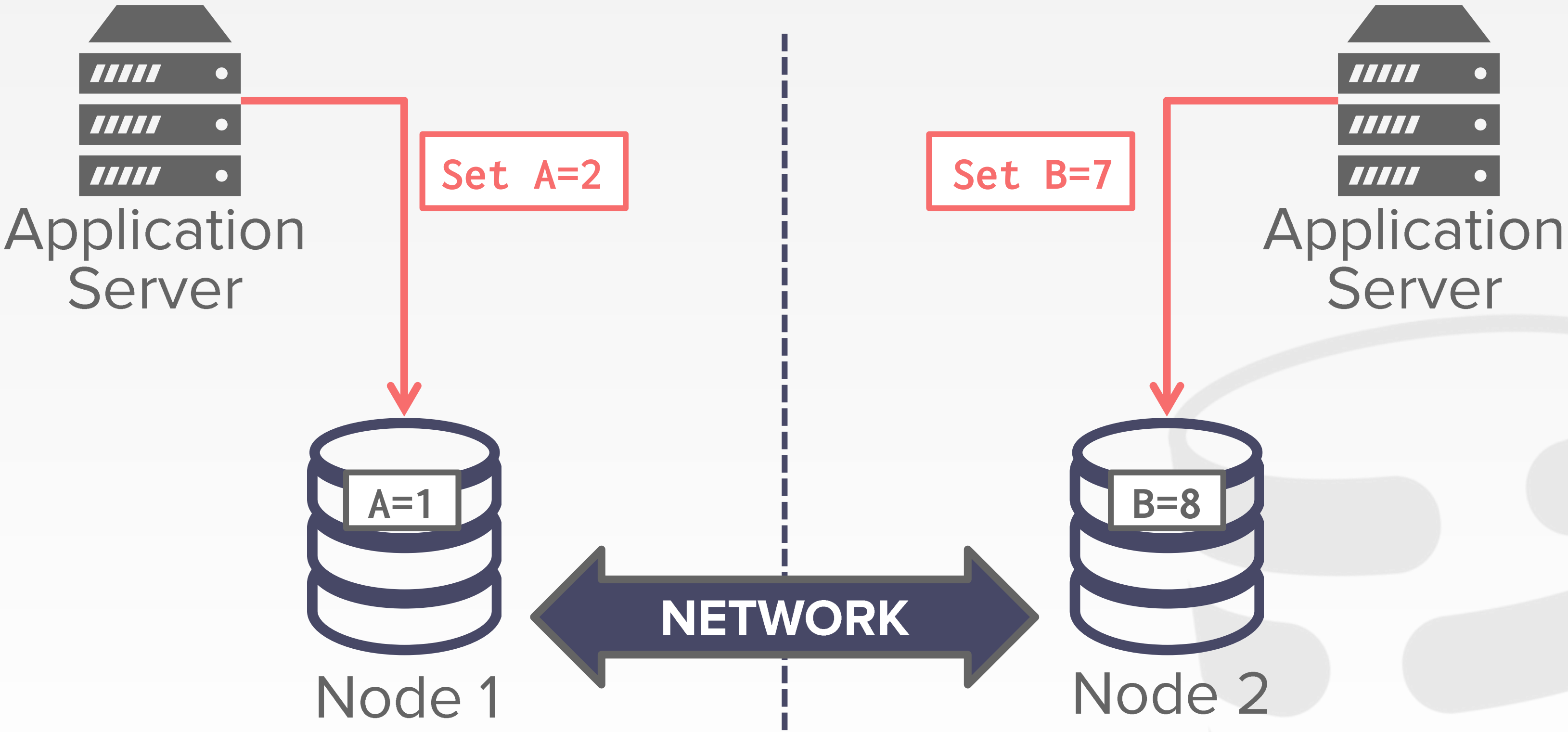
Need to allow multiple txns to execute simultaneously across multiple nodes.

→ Many of the same protocols from single-node DBMSs can be adapted.

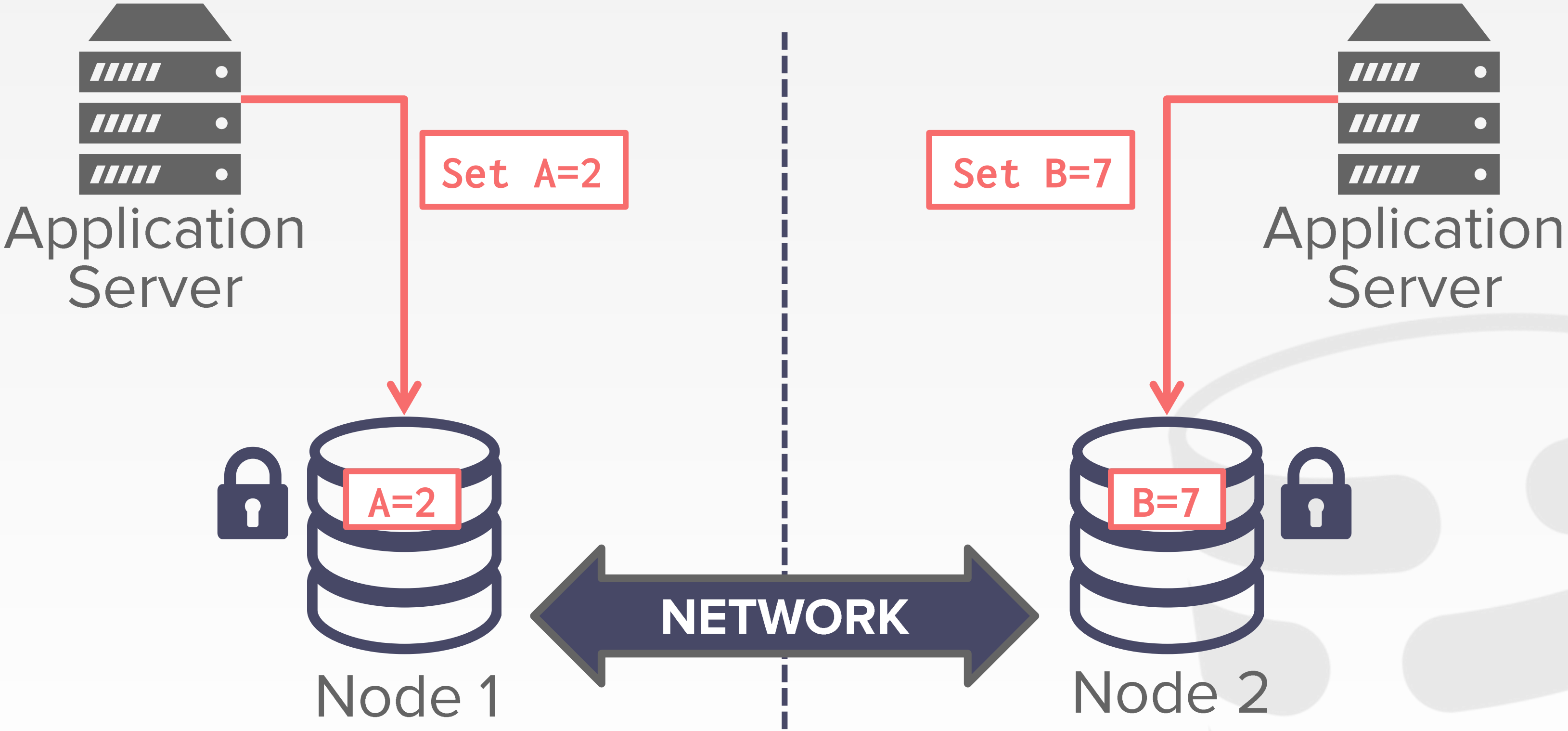
This is harder because of:

→ Replication.
→ Network Communication Overhead.
→ Node Failures.
→ Clock Skew.

DISTRIBUTED 2PL

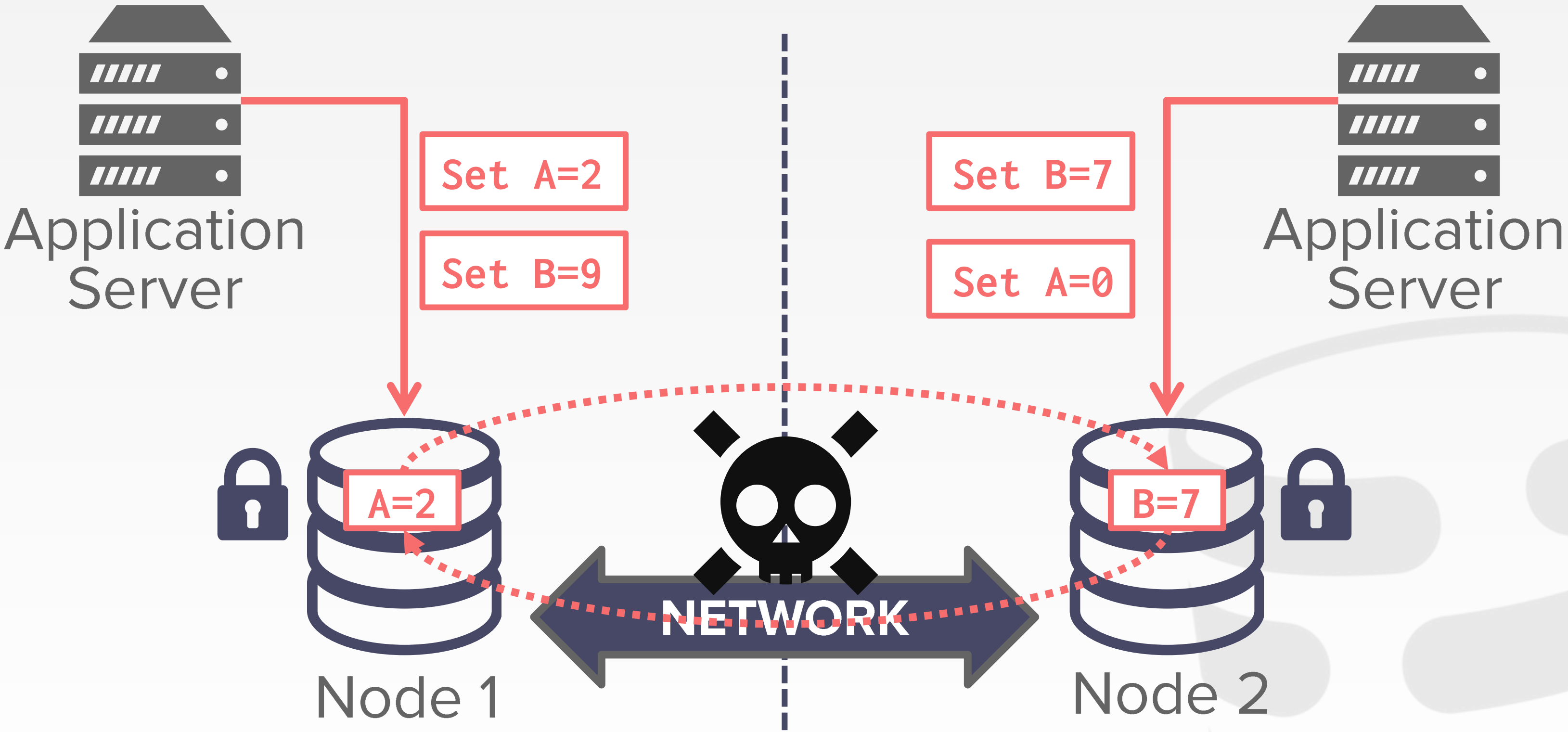


DISTRIBUTED 2PL



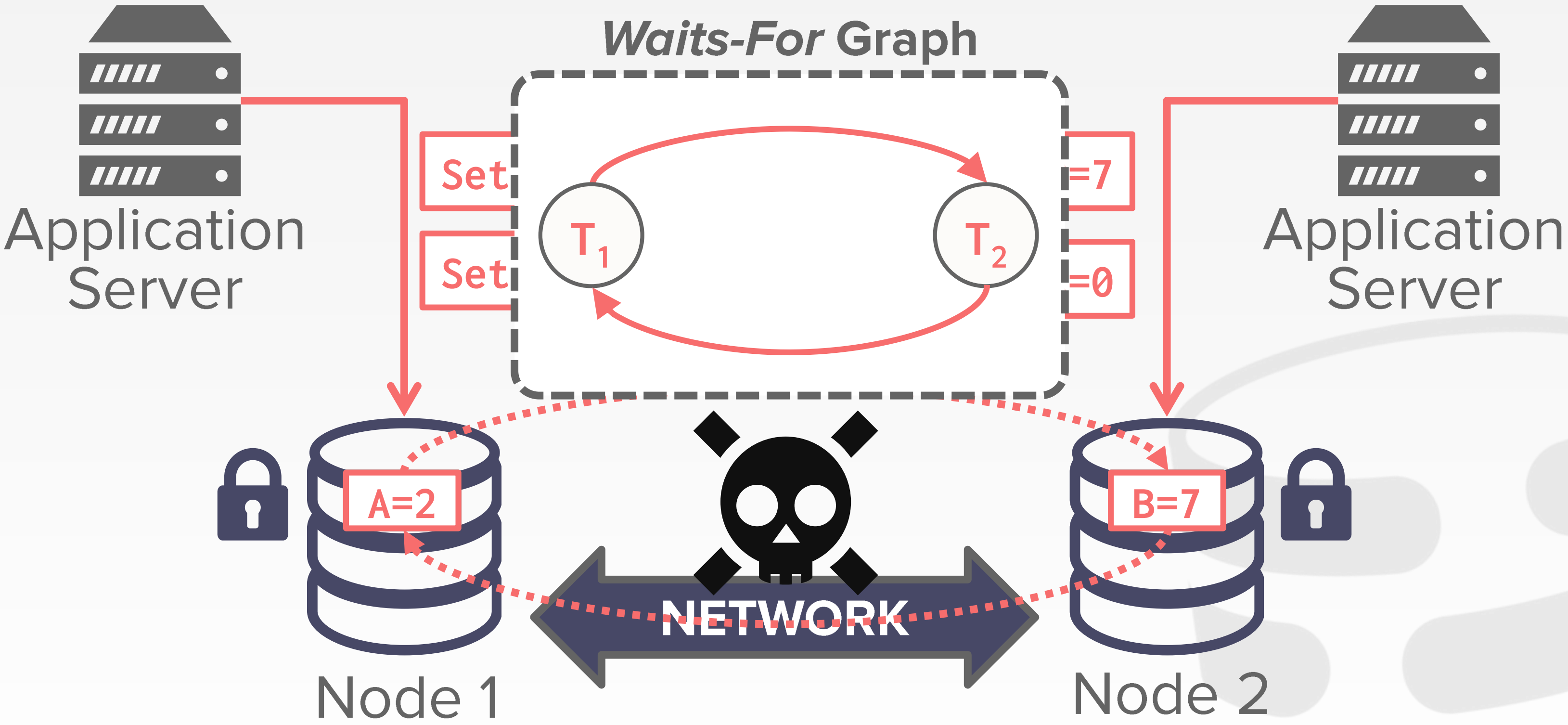
CARNEGIE MELLON DATABASE GROUP

DISTRIBUTED 2PL



CARNEGIE MELLON DATABASE GROUP

DISTRIBUTED 2PL



OBSERVATION

We have not discussed how to ensure that all nodes agree to commit a txn and then to make sure it does commit if we decide that it should.

→ What happens if a node fails?
→ What happens if our messages show up late?

ATOMIC COMMIT PROTOCOL

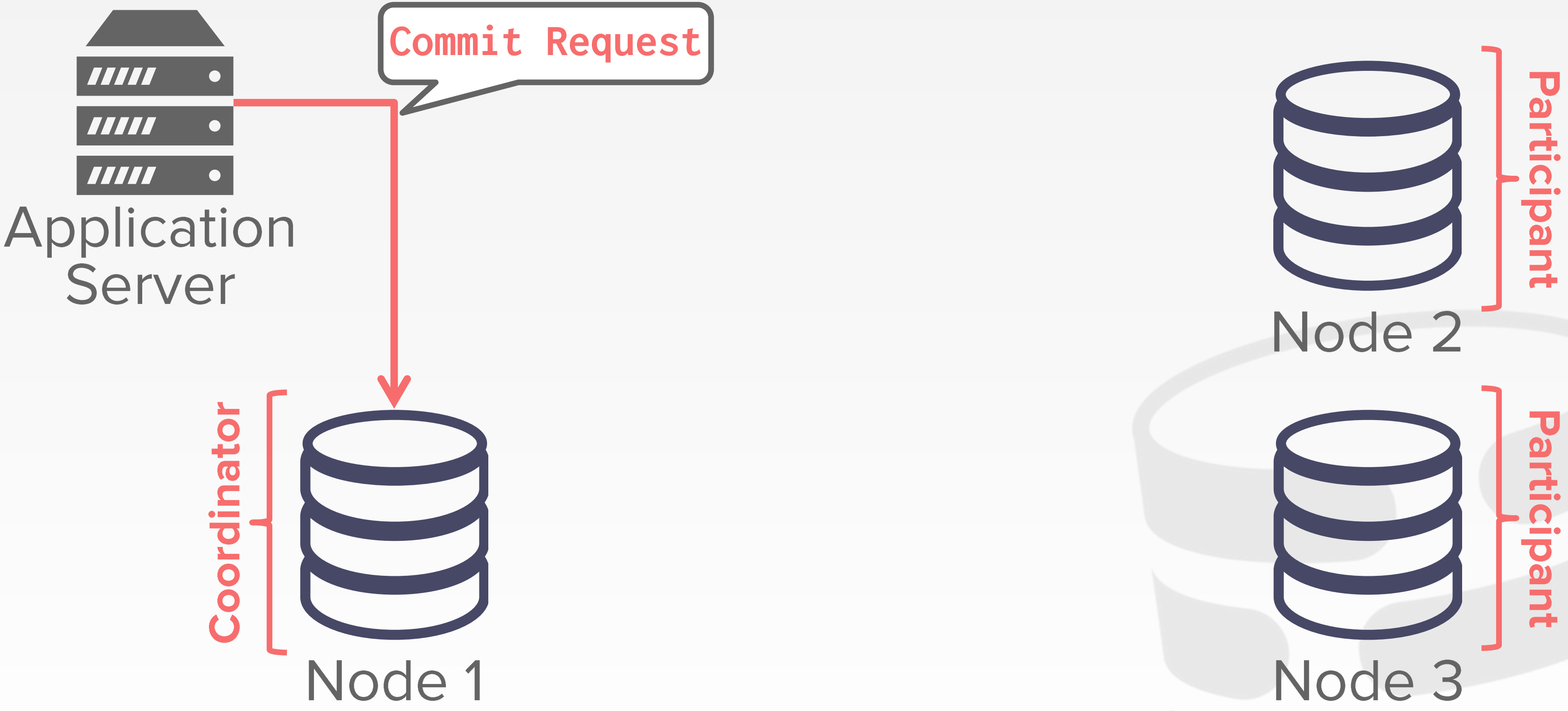
When a multi-node txn finishes, the DBMS needs to ask all of the nodes involved whether it is safe to commit.

→ All nodes must agree on the outcome

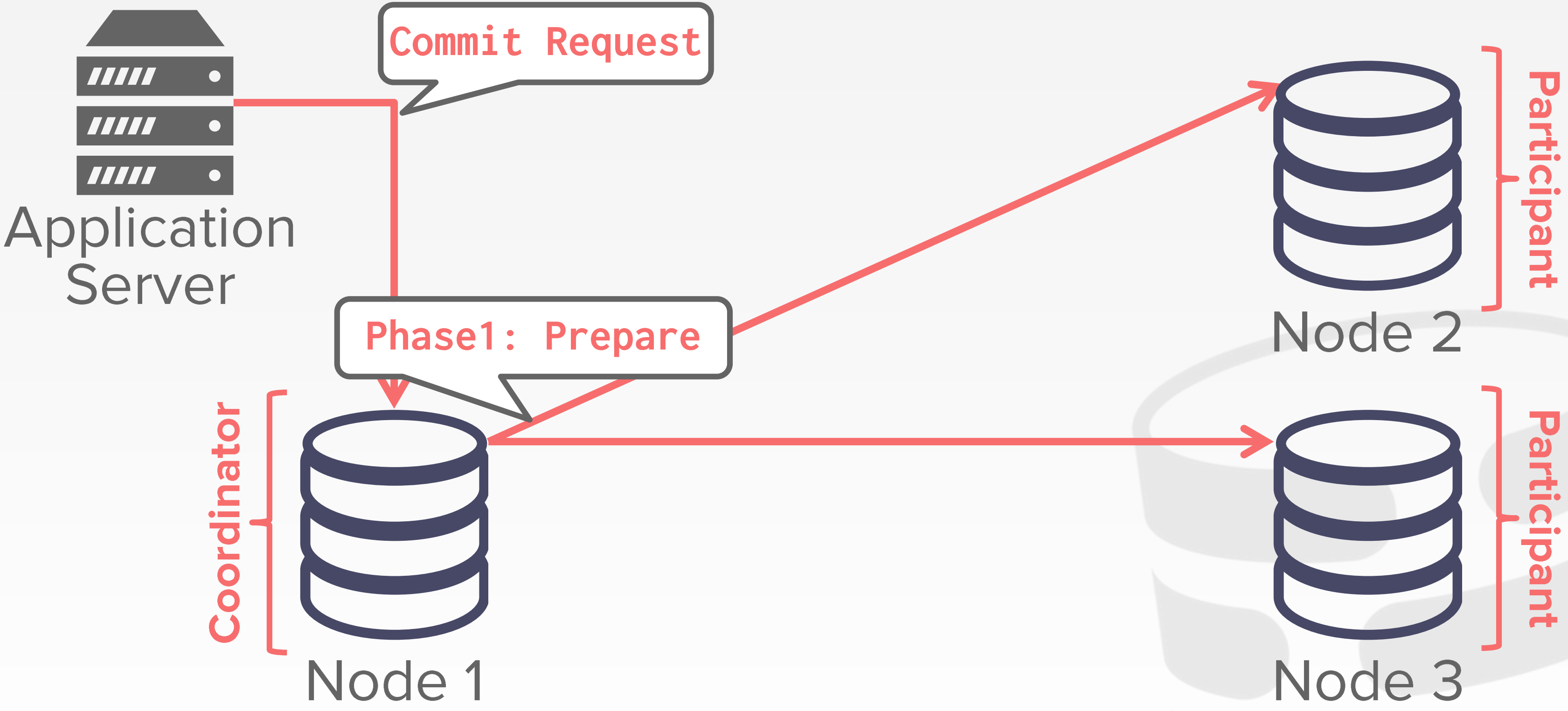
Examples:

→ Two-Phase Commit
→ Three-Phase Commit (not used)
→ Paxos
→ Raft
→ ZAB (Apache Zookeeper)

TWO-PHASE COMMIT (SUCCESS)

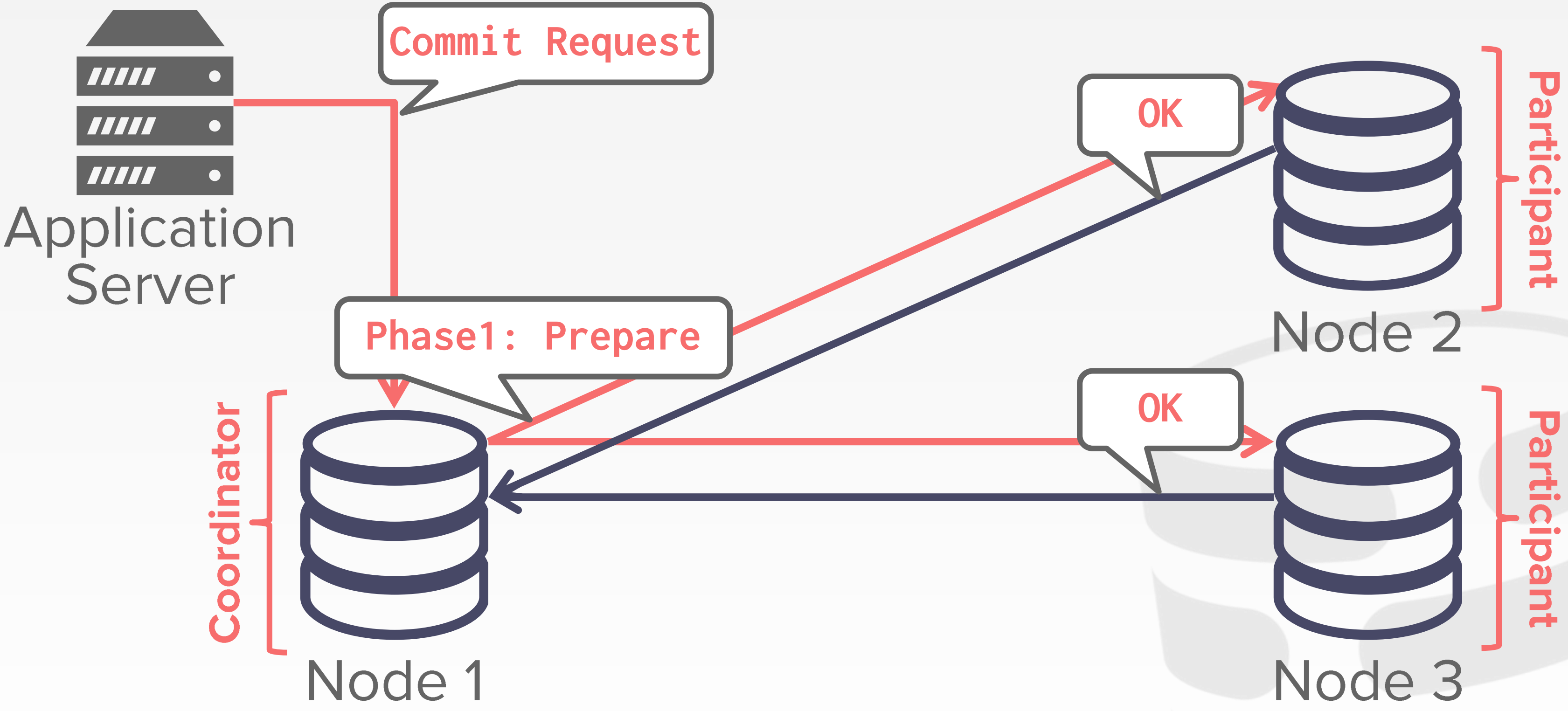


TWO-PHASE COMMIT (SUCCESS)



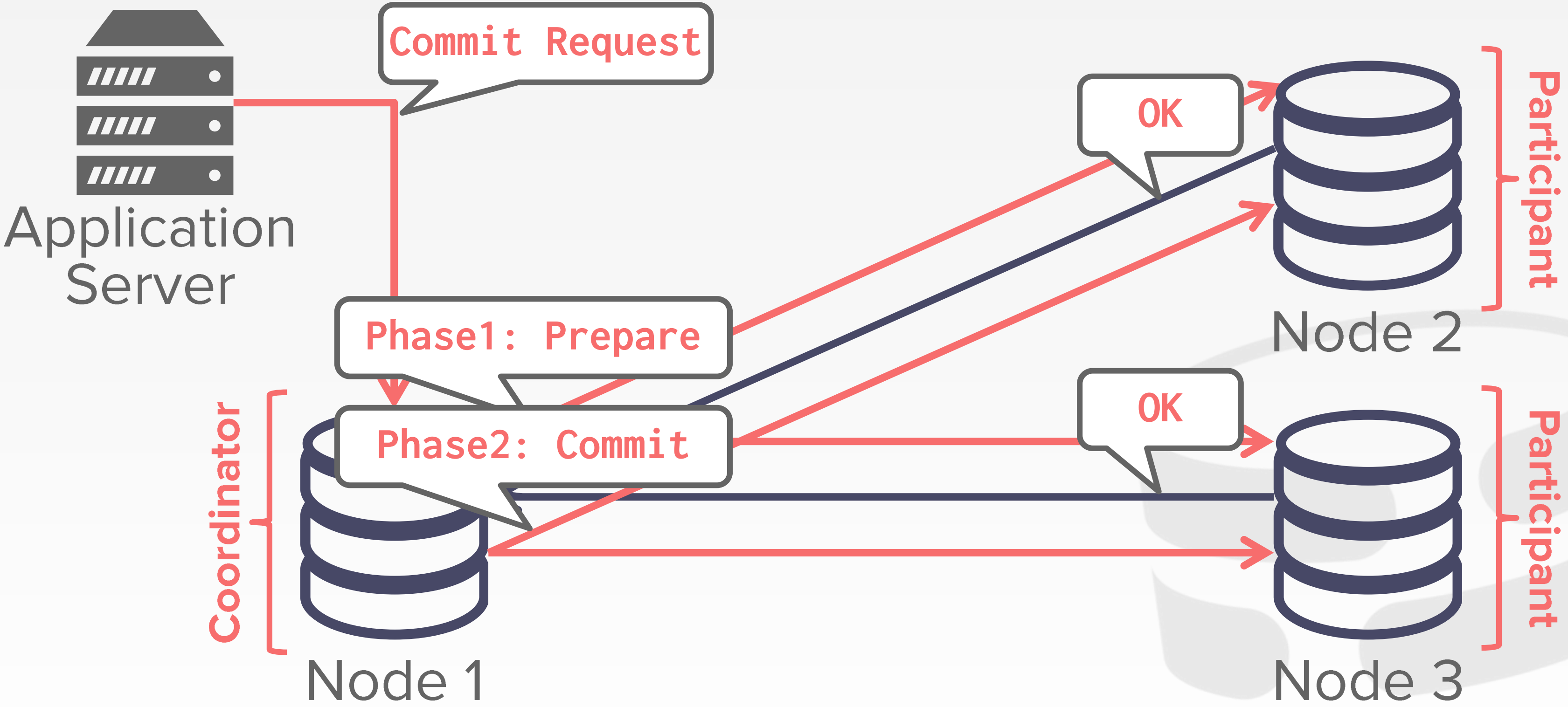
CARNEGIE MELLON DATABASE GROUP

TWO-PHASE COMMIT (SUCCESS)



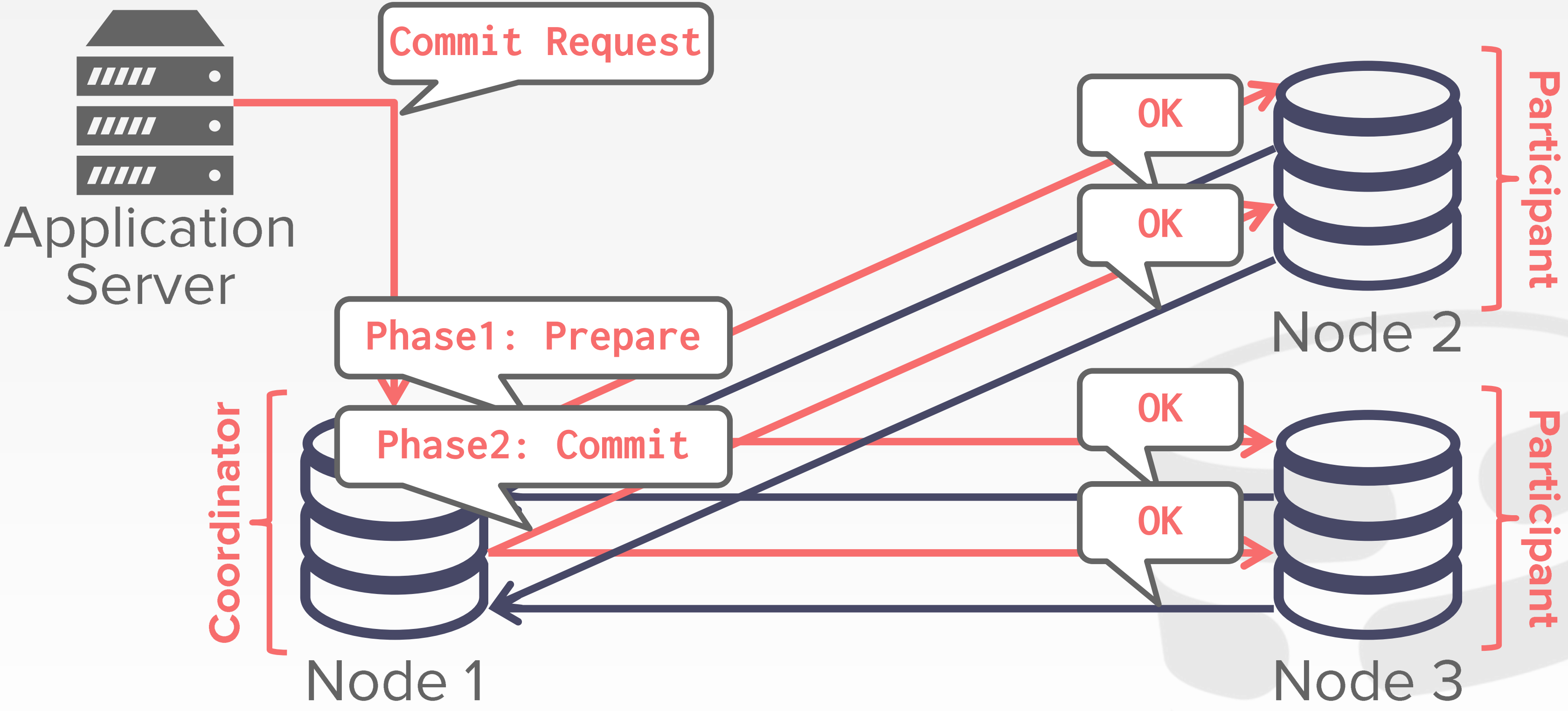
CARNEGIE MELLON DATABASE GROUP

TWO-PHASE COMMIT (SUCCESS)



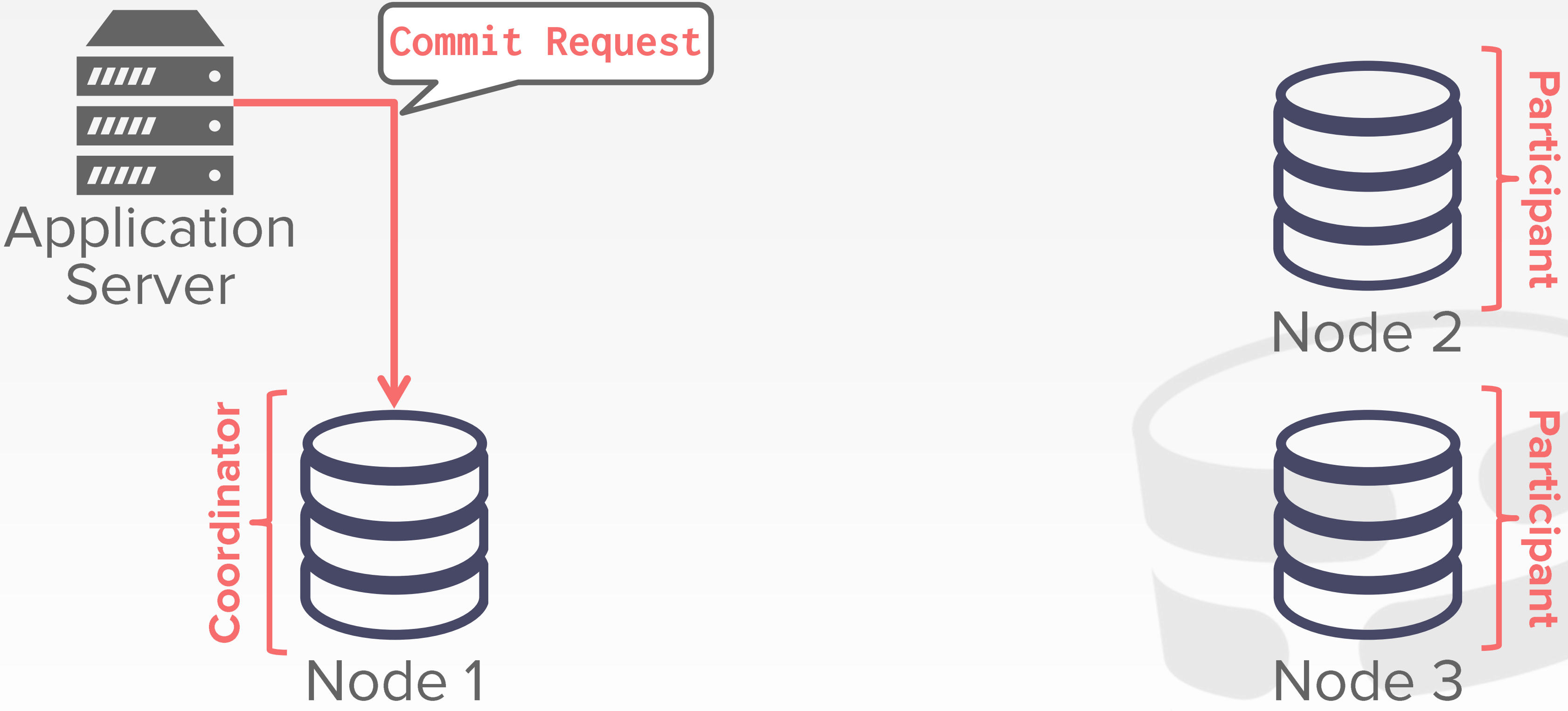
CARNEGIE MELLON DATABASE GROUP

TWO-PHASE COMMIT (SUCCESS)



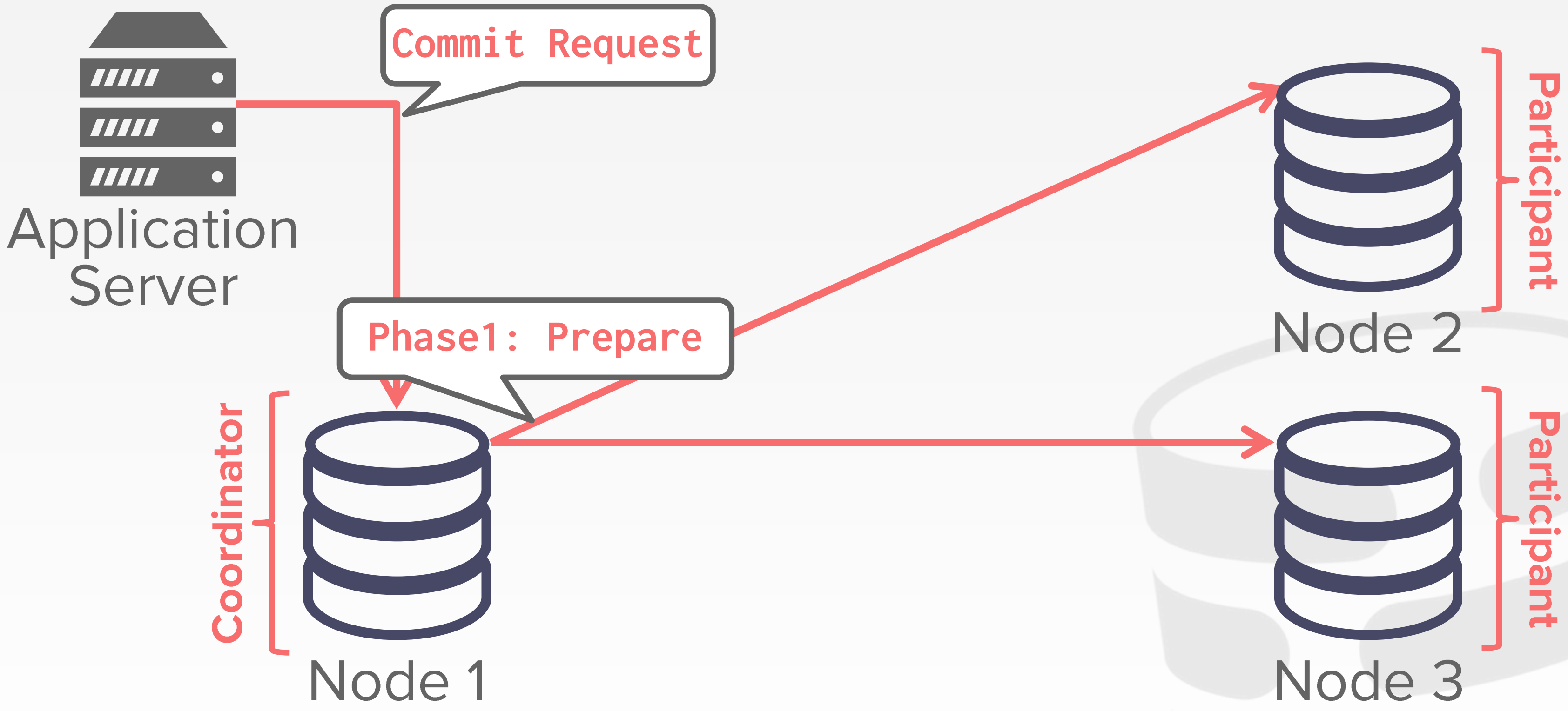
TWO-PHASE COMMIT (SUCCESS)

TWO-PHASE COMMIT (ABORT)

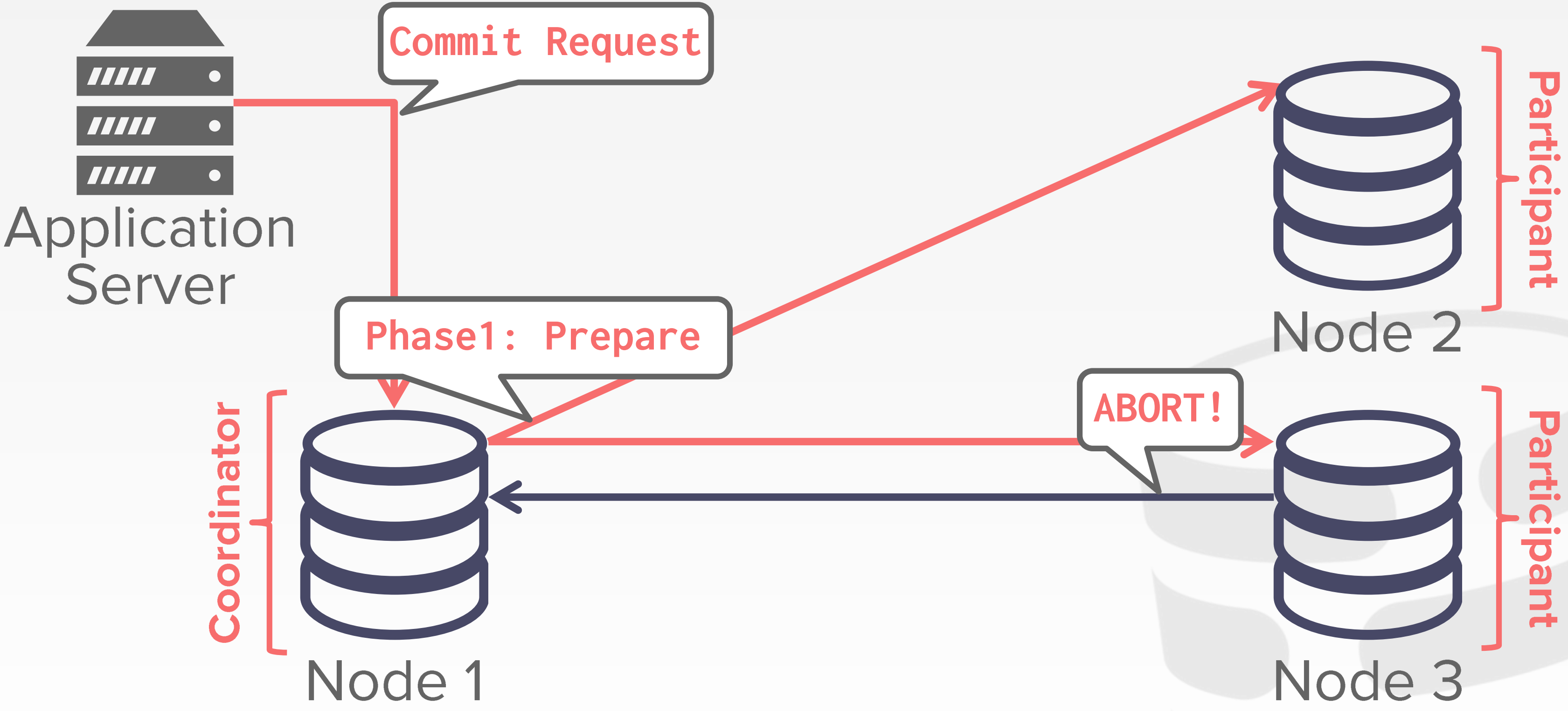


CARNEGIE MELLON DATABASE GROUP

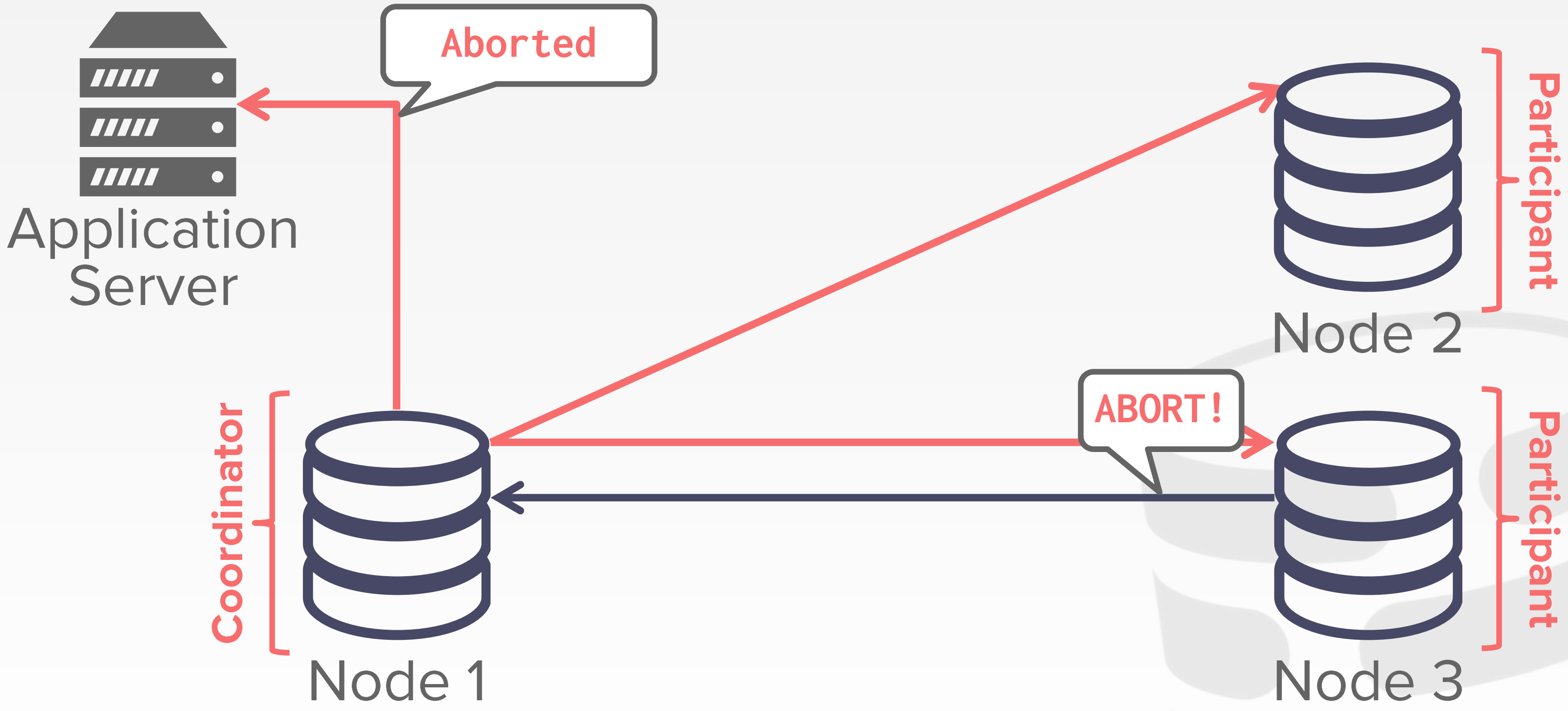
TWO-PHASE COMMIT (ABORT)



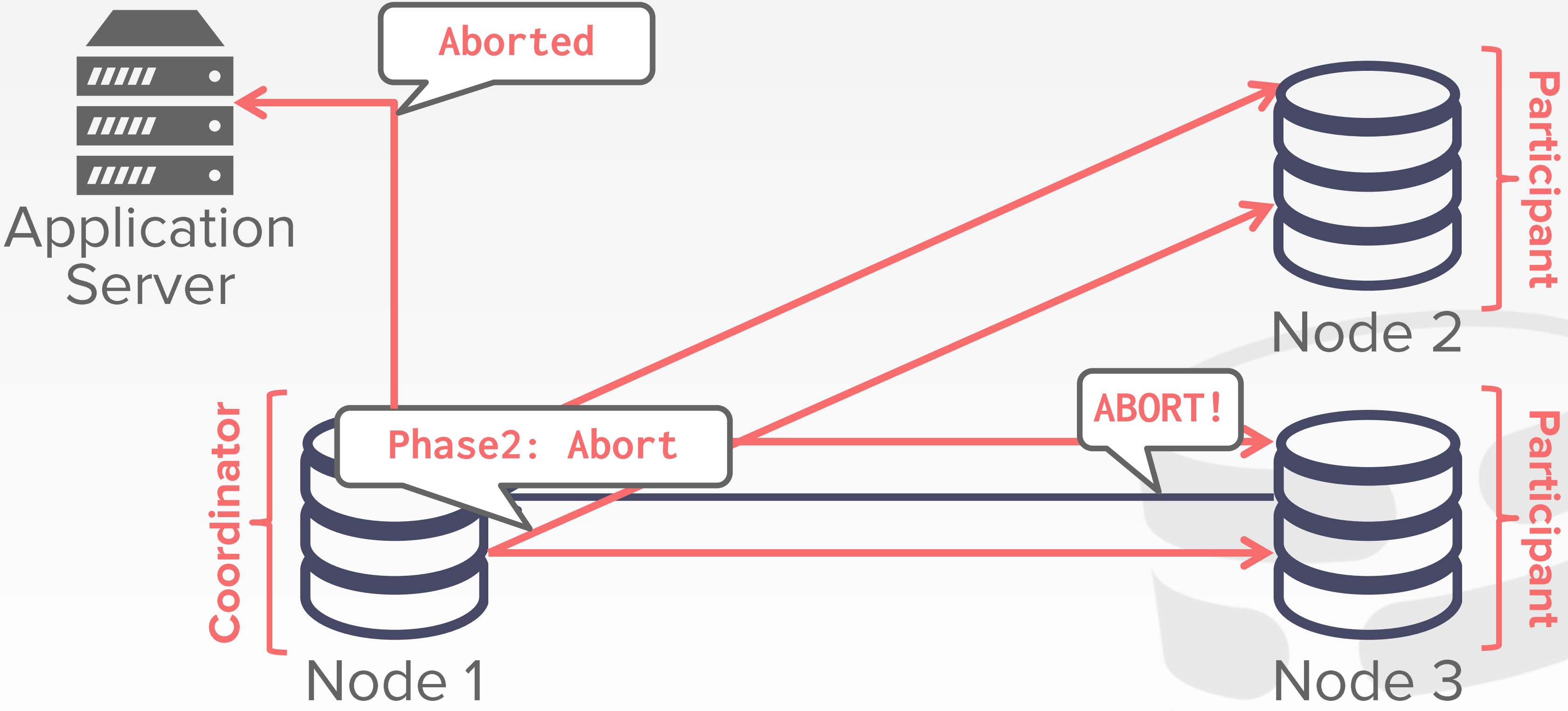
TWO-PHASE COMMIT (ABORT)



TWO-PHASE COMMIT (ABORT)

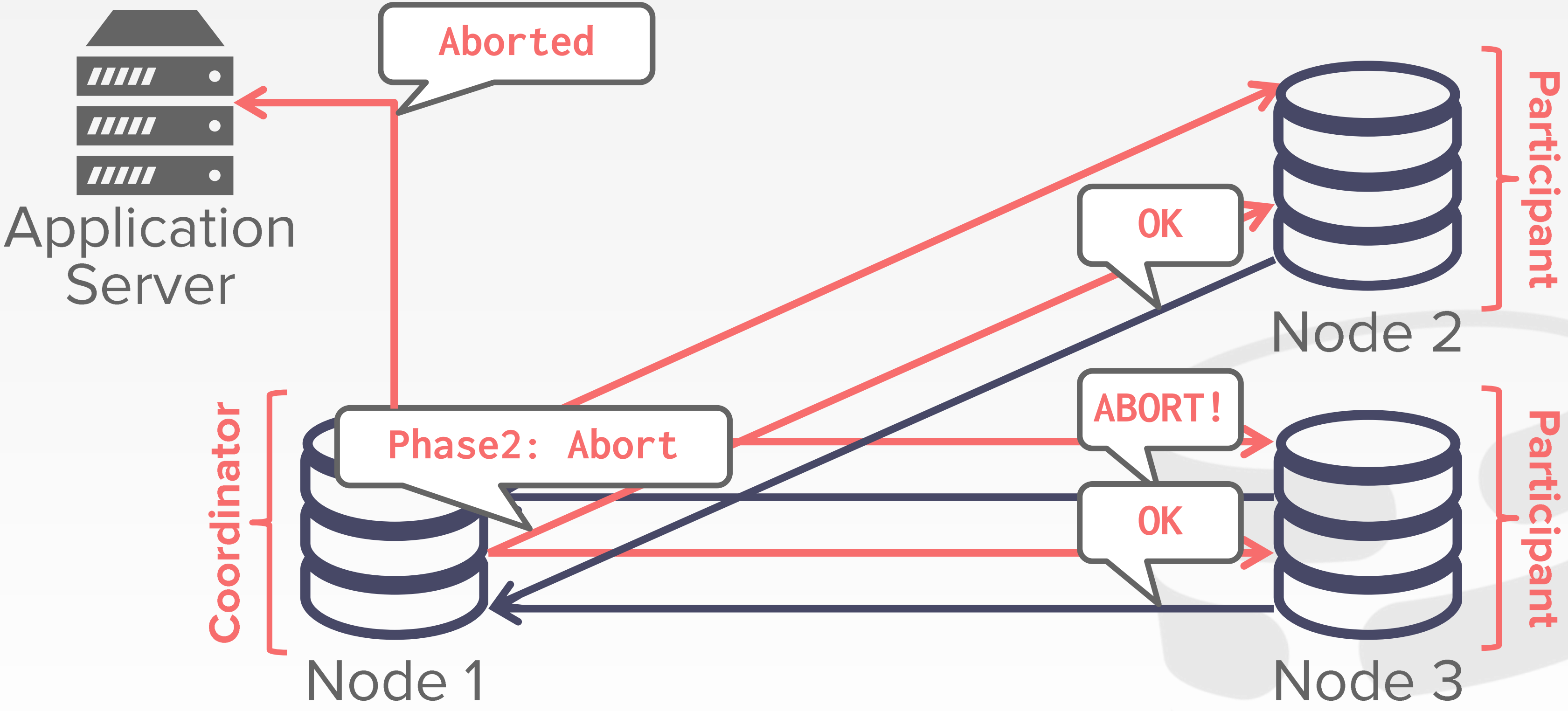


TWO-PHASE COMMIT (ABORT)



CARNEGIE MELLON DATABASE GROUP

TWO-PHASE COMMIT (ABORT)



CARNEGIE MELLON DATABASE GROUP

2PC OPTIMIZATIONS

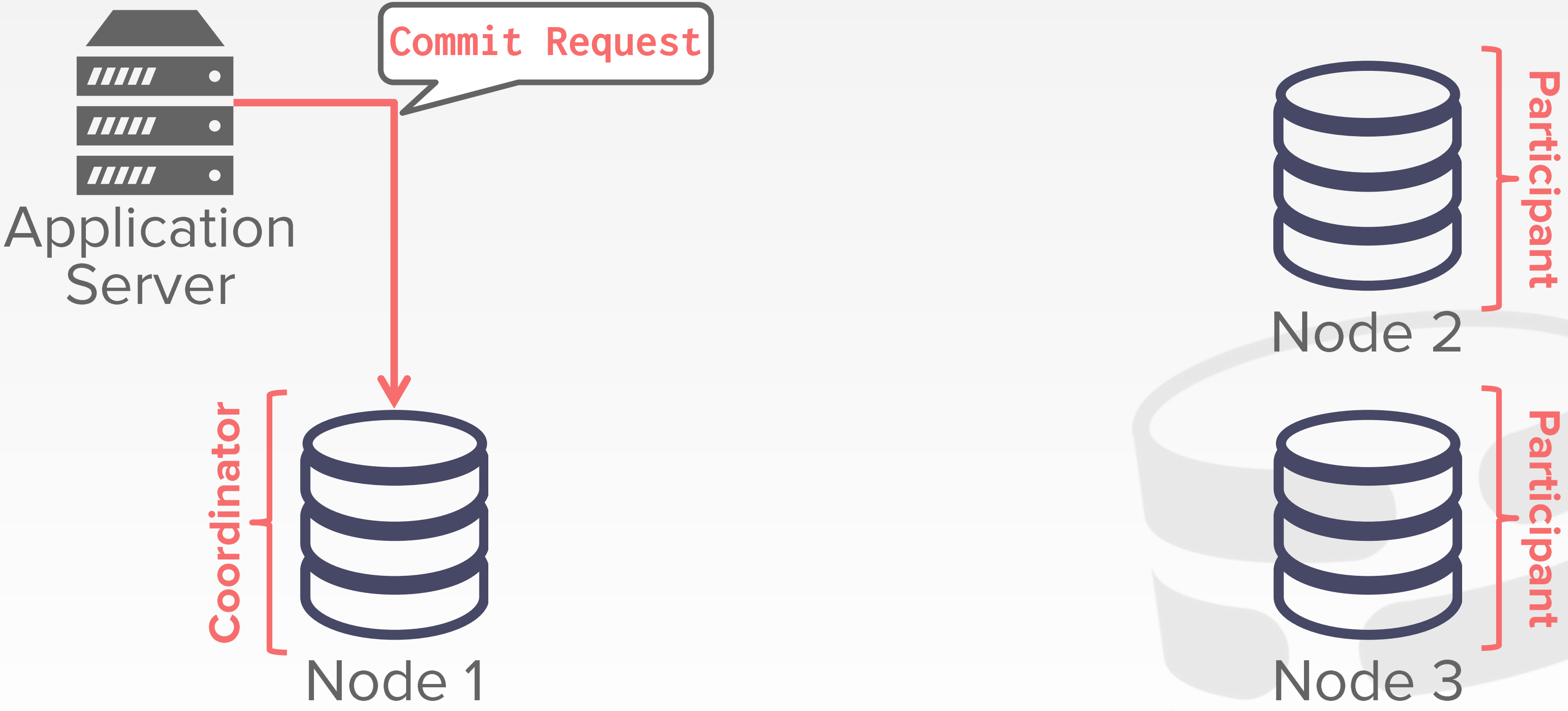
Early Prepare Voting

→ If you send a query to a remote node that you know will be the last one you execute there, then that node will also return their vote for the prepare phase with the query result.

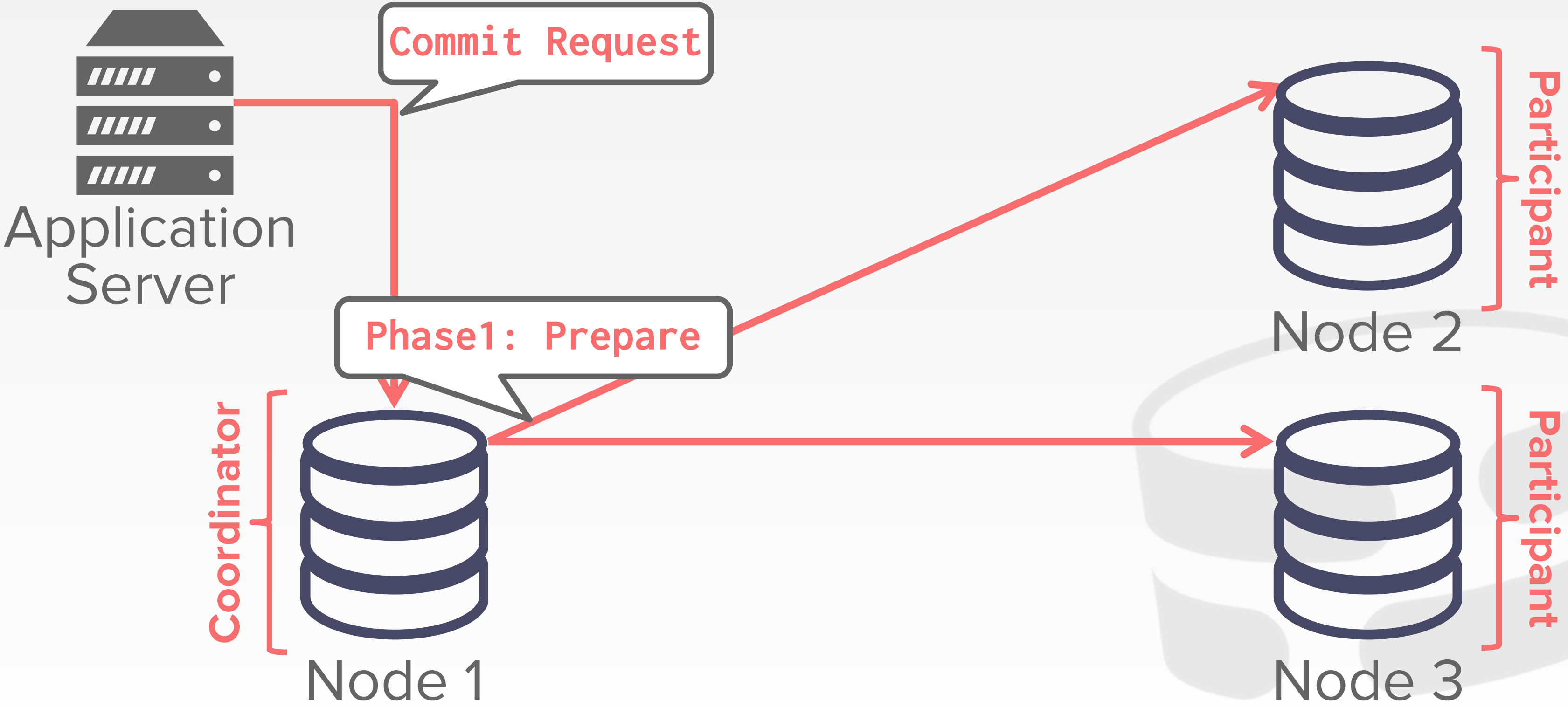
Early Acknowledgement After Prepare

→ If all nodes vote to commit a txn, the coordinator can send the client an acknowledgement that their txn was successful before the commit phase finishes.

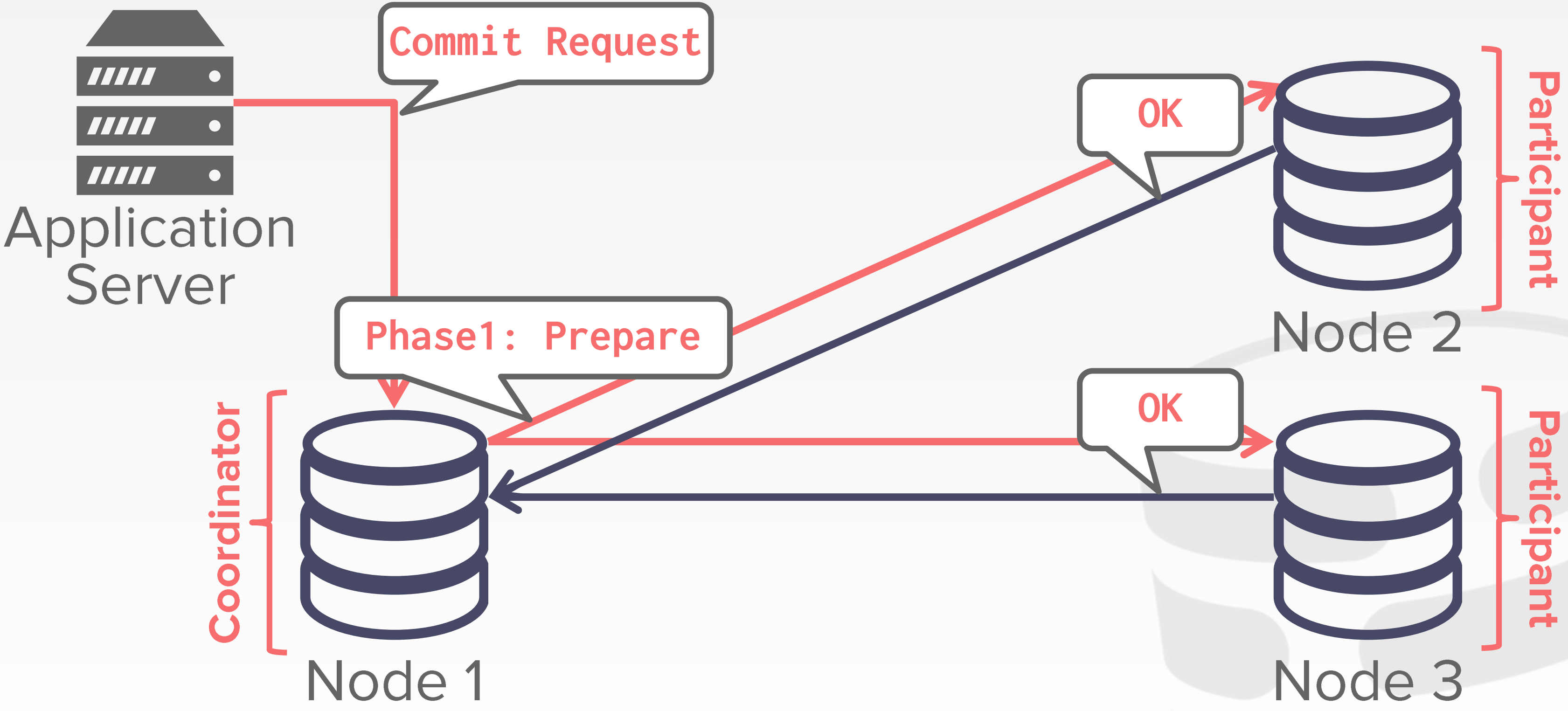
EARLY ACKNOWLEDGEMENT



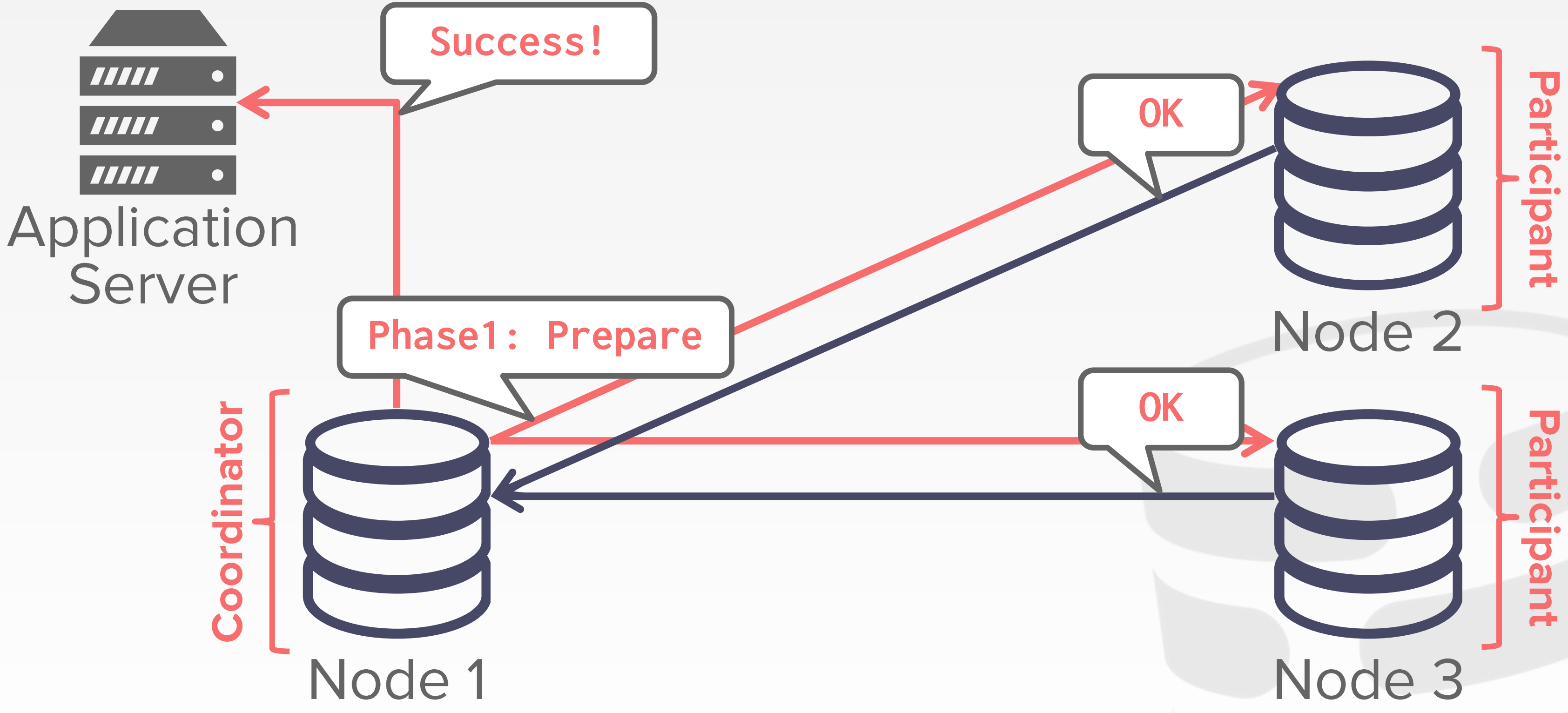
EARLY ACKNOWLEDGEMENT



EARLY ACKNOWLEDGEMENT

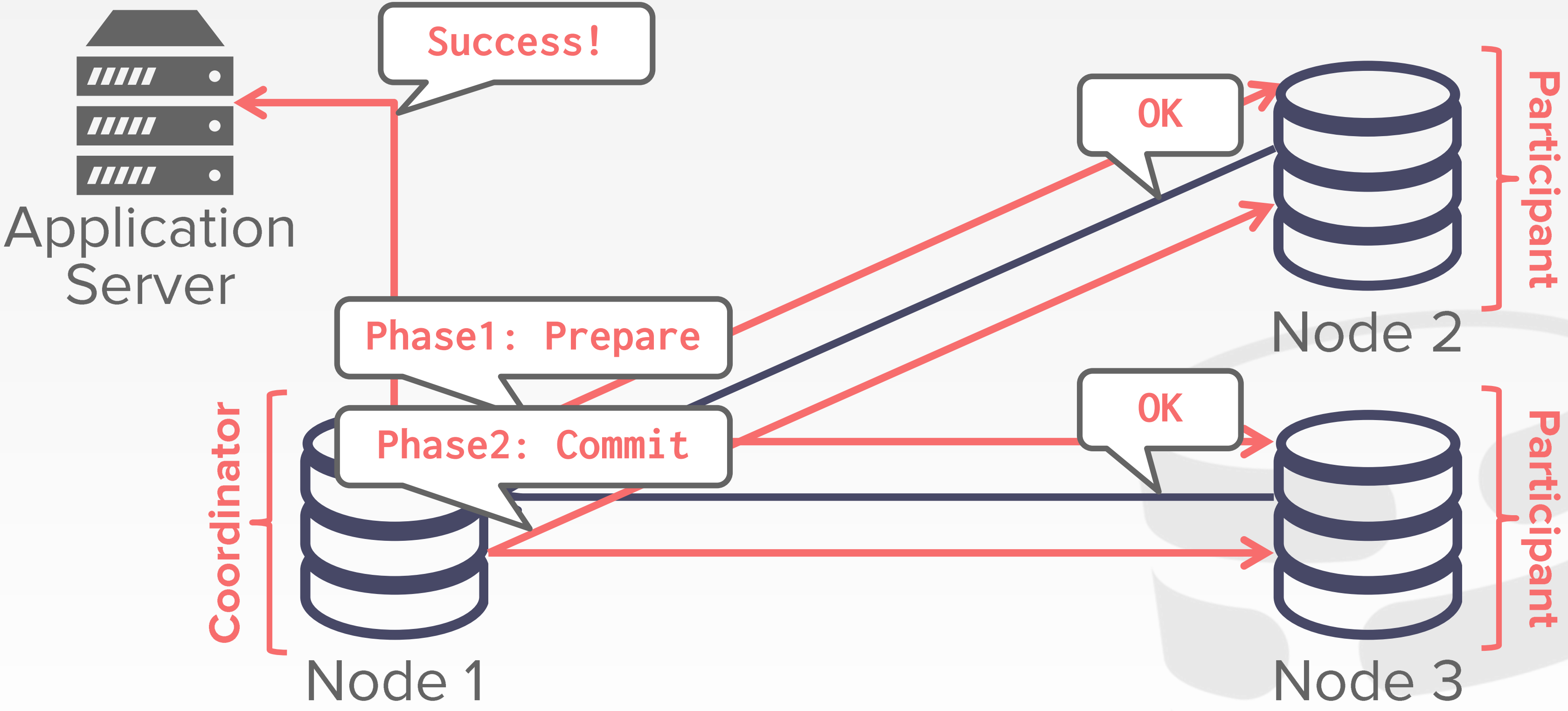


EARLY ACKNOWLEDGEMENT

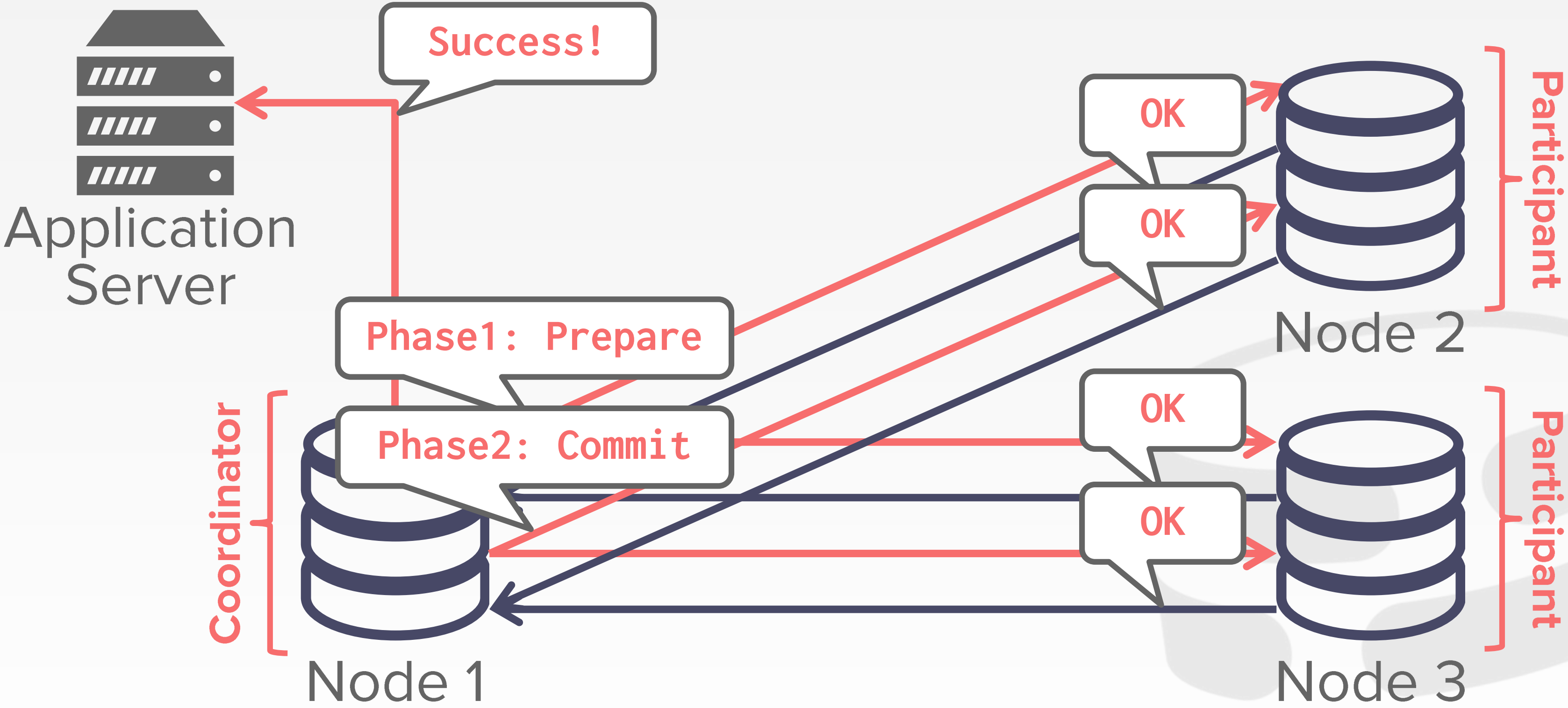


CARNEGIE MELLON DATABASE GROUP

EARLY ACKNOWLEDGEMENT



EARLY ACKNOWLEDGEMENT



TWO-PHASE COMMIT

Each node has to record the outcome of each phase in a stable storage log.

What happens if coordinator crashes?

→ Participants have to decide what to do.

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.

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.

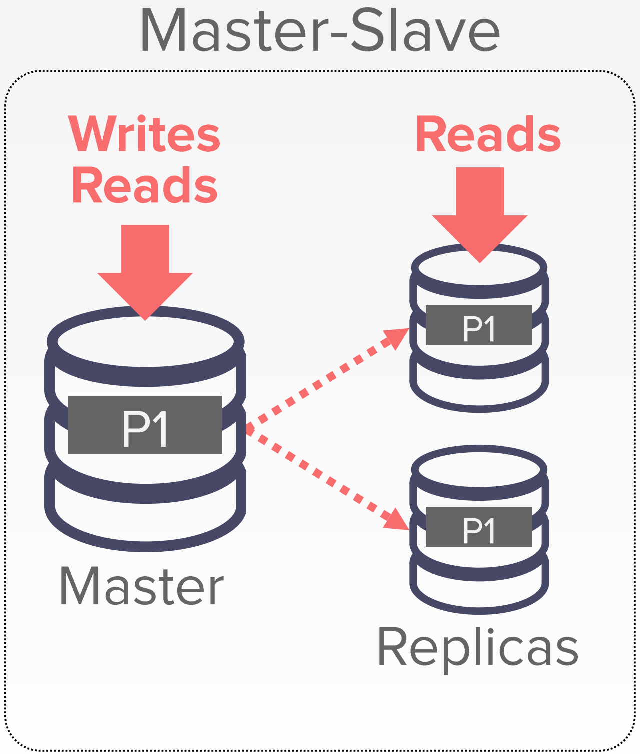
REPLICATION

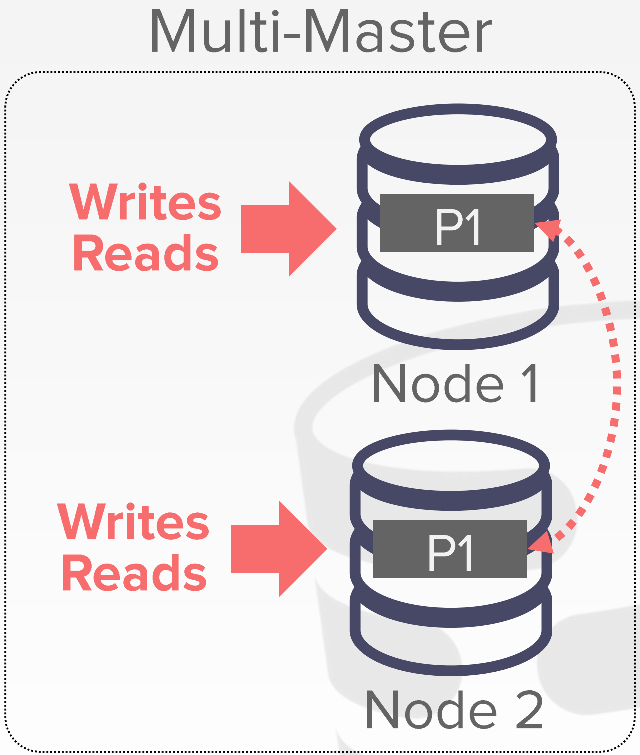
The DBMS can replicate data across redundant nodes to increase availability.

System Configurations:
→ **Master/Slave** (aka Leader-Follower)
→ **Multi-Master** (aka Multi-Home)

Propagation Protocols:
→ **Physical Logging**
→ **Logical Logging**

REPLICATION CONFIGURATIONS





REPLICATION PROPAGATION

Synchronous

→ The master sends updates to replicas and then waits for them to acknowledge that they fully applied (i.e., logged) the changes.

Asynchronous

→ The master immediately returns the acknowledgement to the client without waiting for replicas to apply the changes.

Semi-Synchronous

→ Replicas immediately send acknowledgements without logging them.

Recovery

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.

CAP THEOREM

Proposed by Eric Brewer that it is impossible for a distributed system to always be:

→ Consistent
→ Always Available
→ Network Partition Tolerant

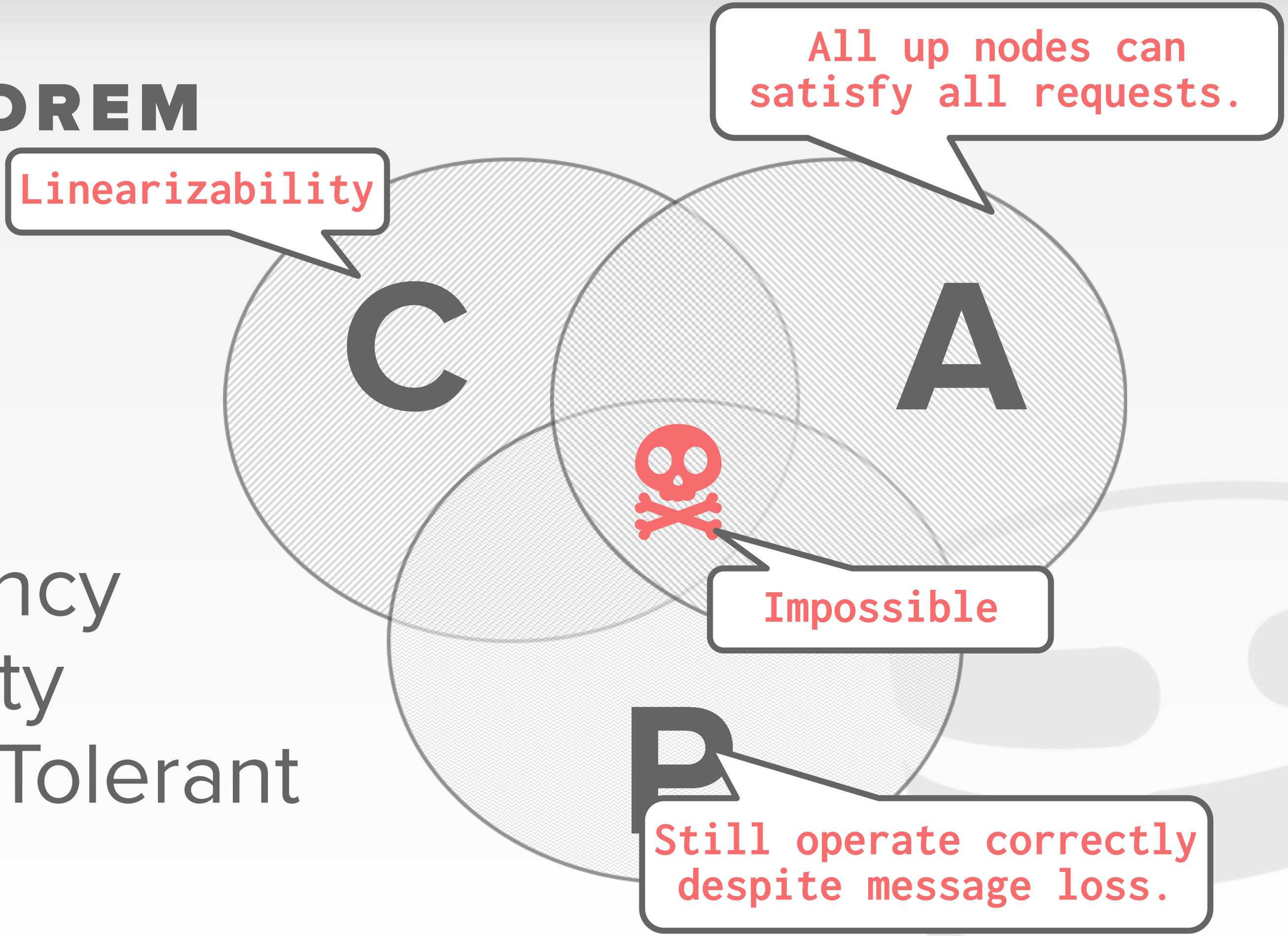
Proved in 2002.

Brewer

CARNEGIE MELLON DATABASE GROUP

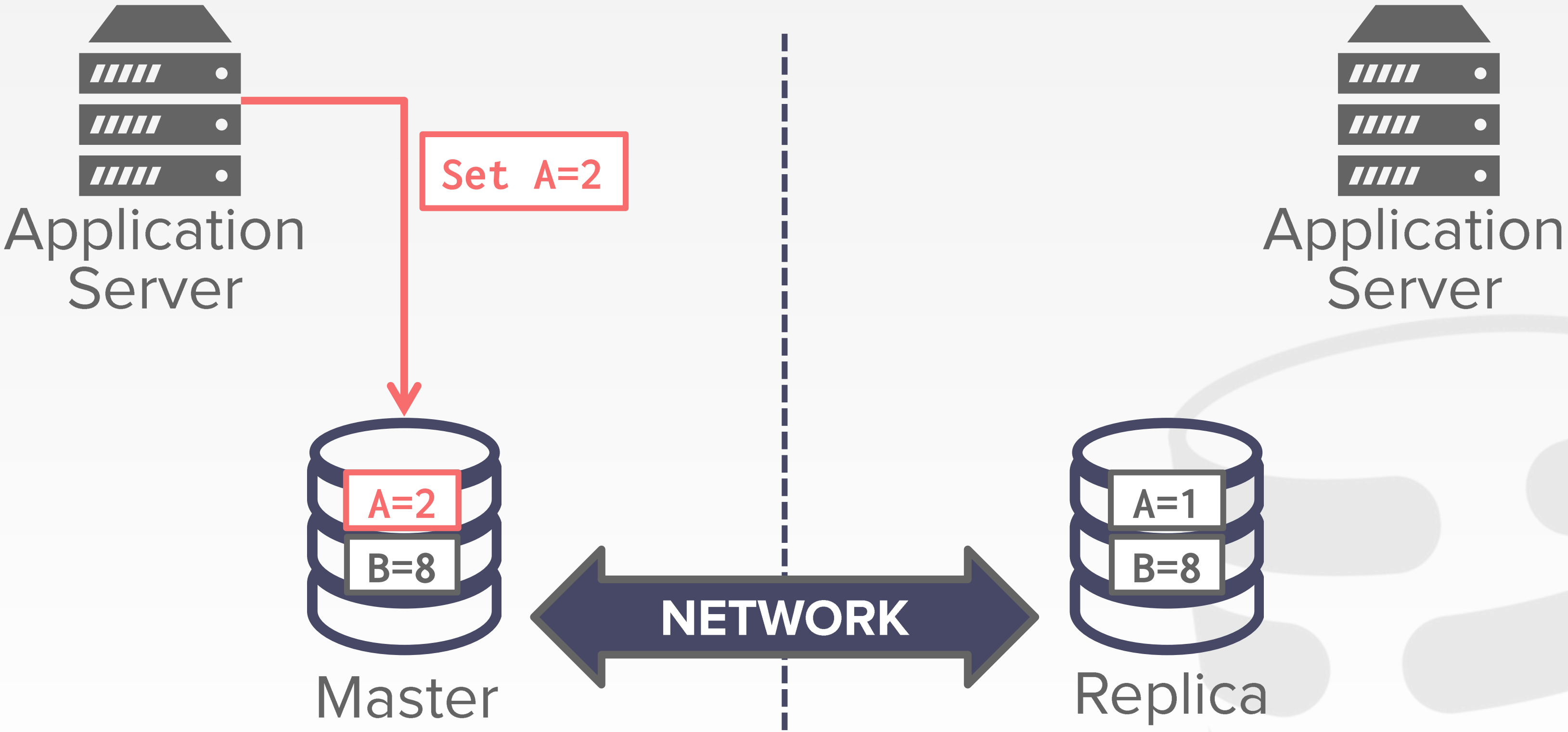
CAP THEOREM

Consistency
Availability
Partition Tolerant



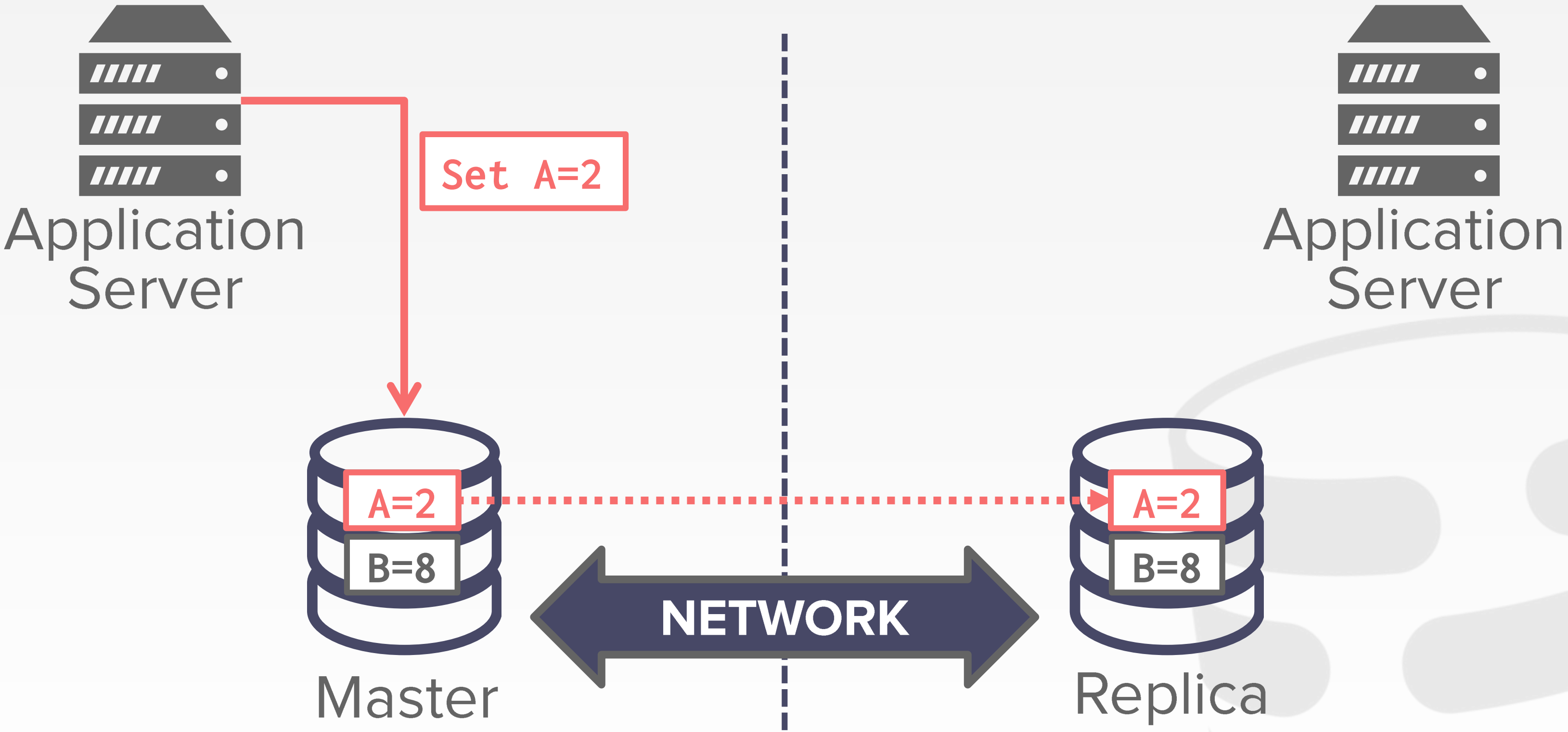
CARNEGIE MELLON DATABASE GROUP

CAP – CONSISTENCY

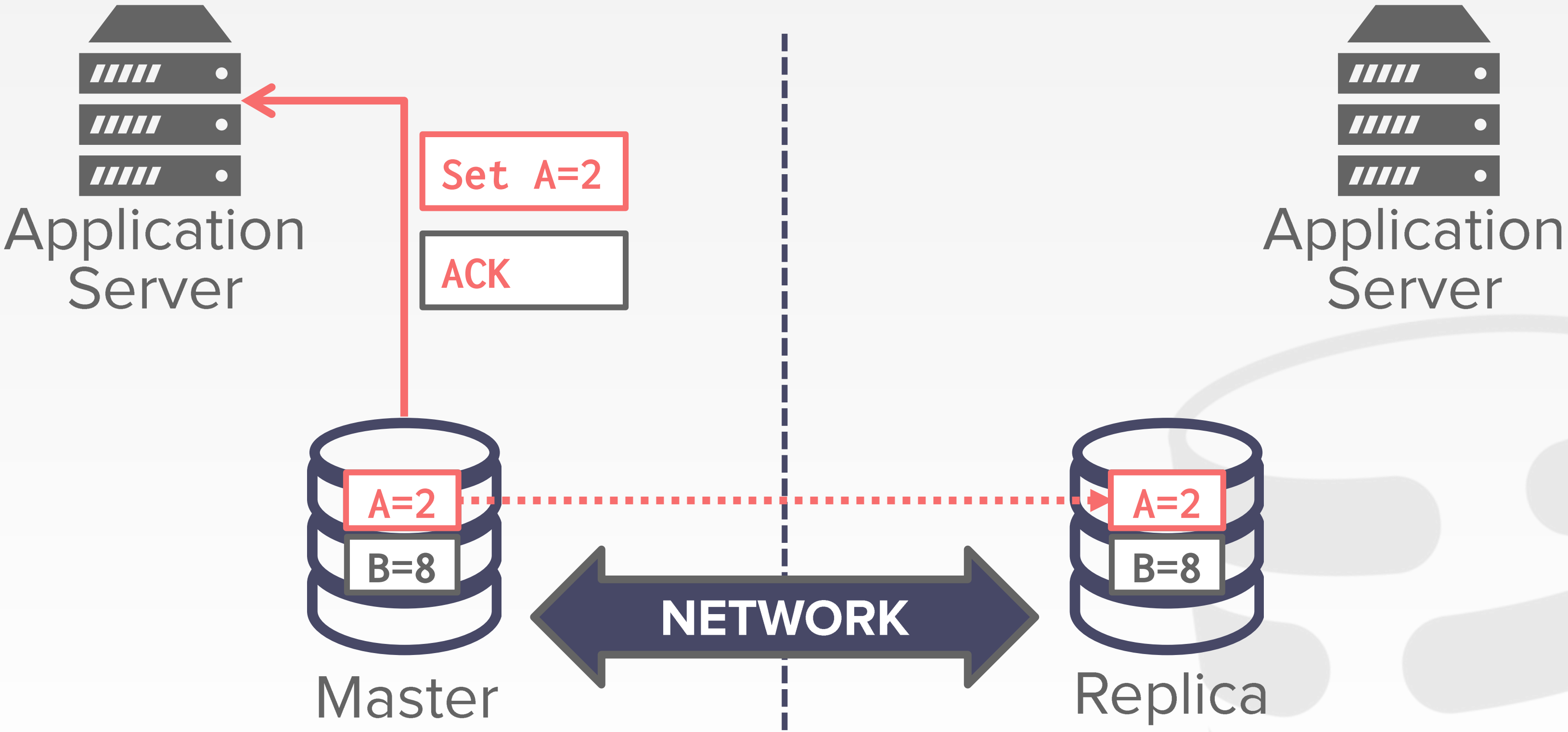


CARNEGIE MELLON DATABASE GROUP

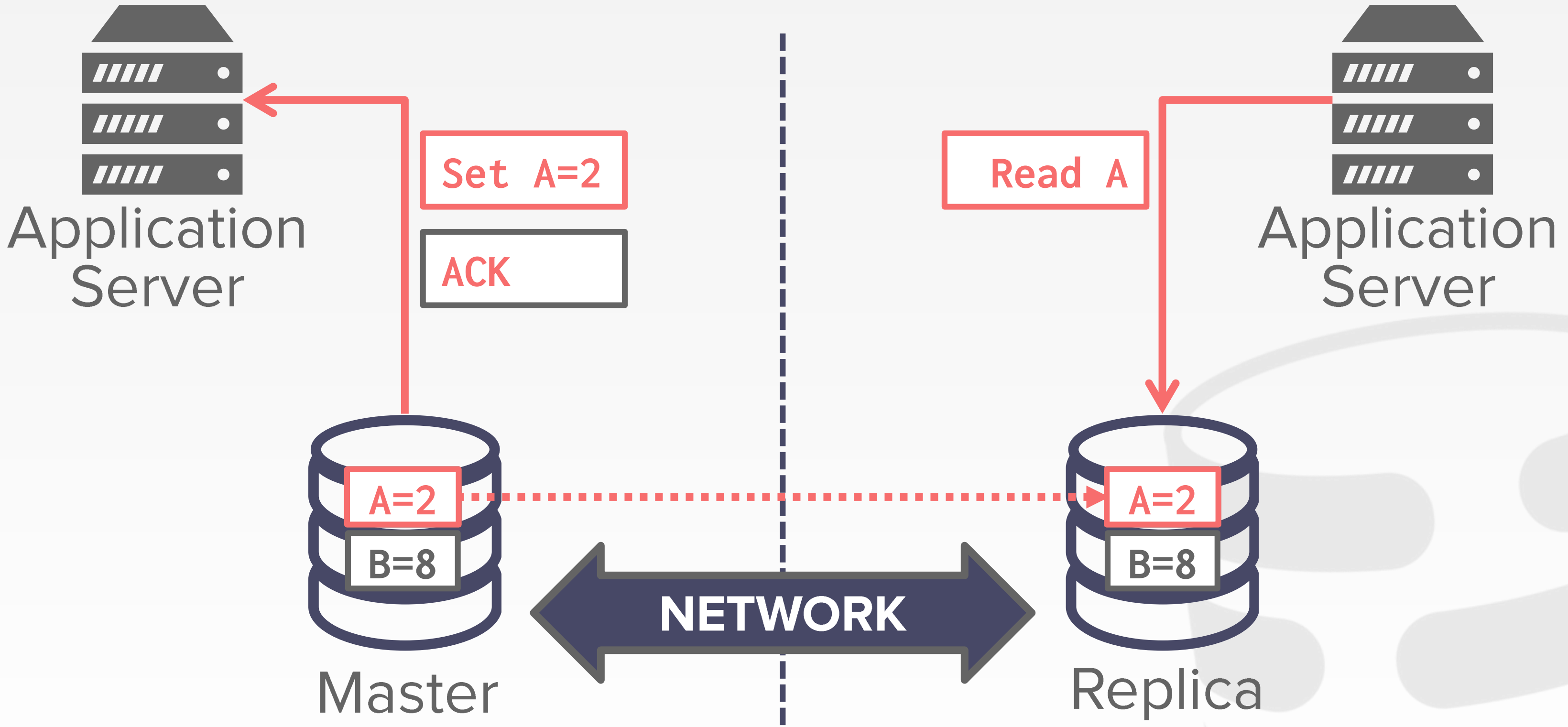
CAP – CONSISTENCY



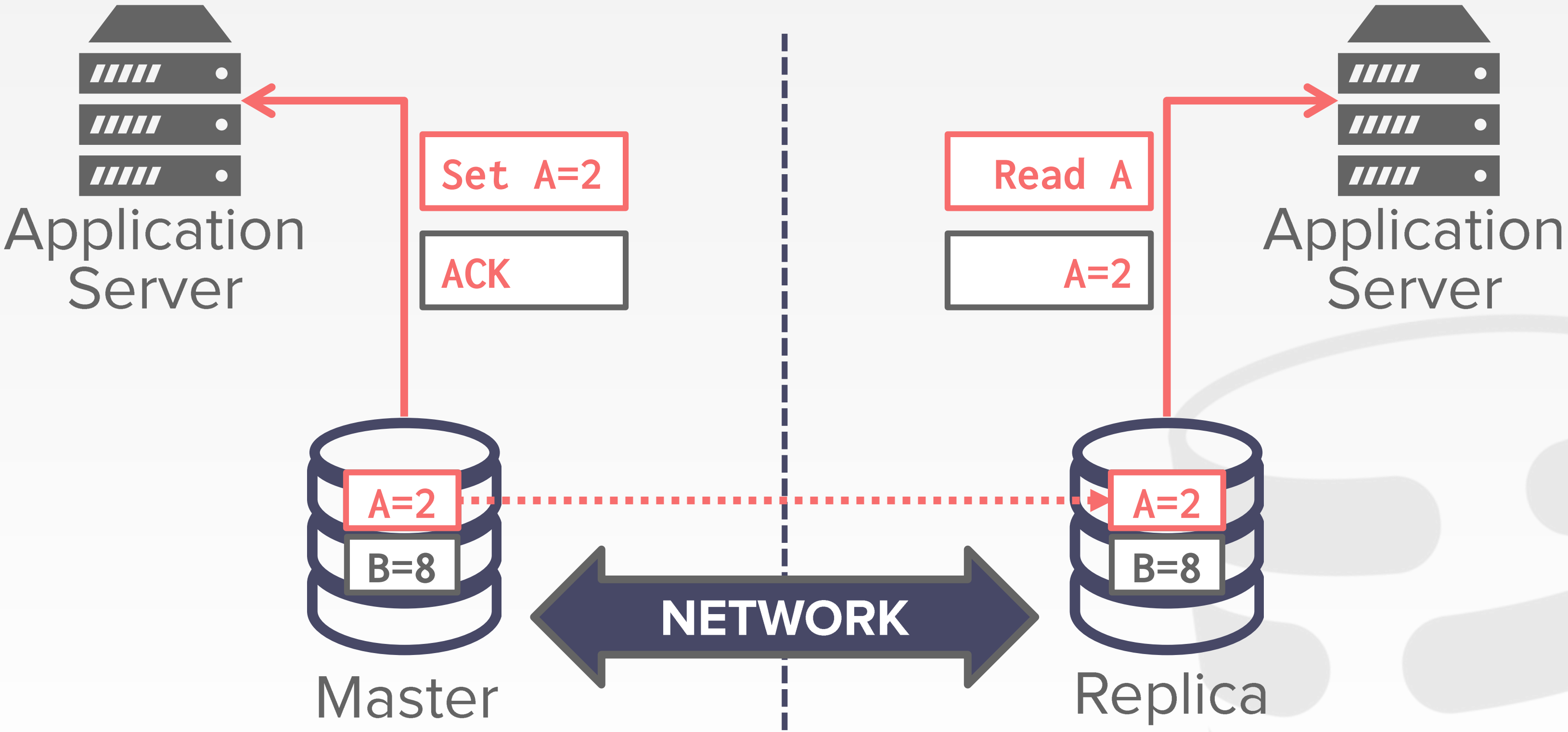
CAP – CONSISTENCY



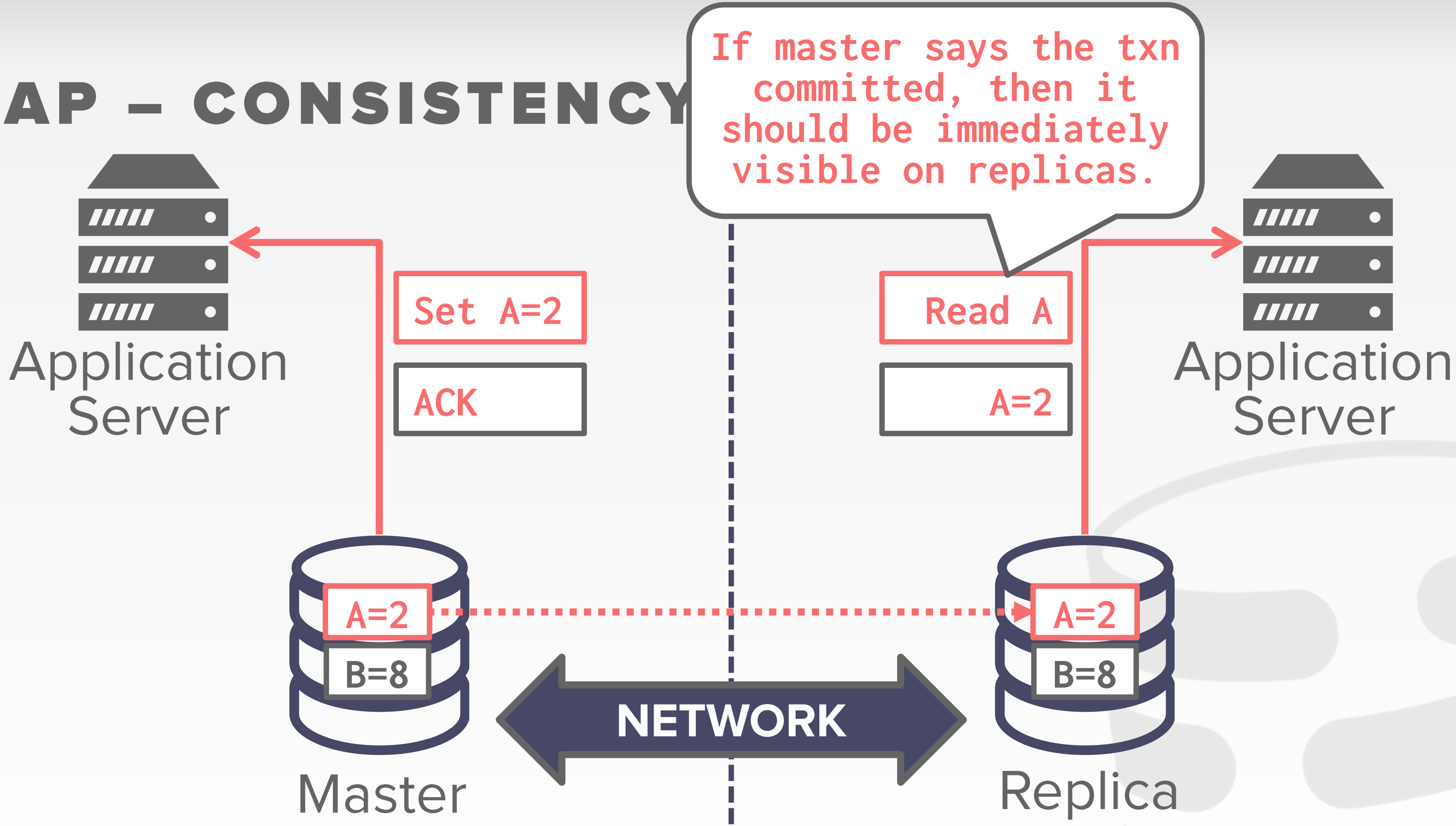
CAP – CONSISTENCY



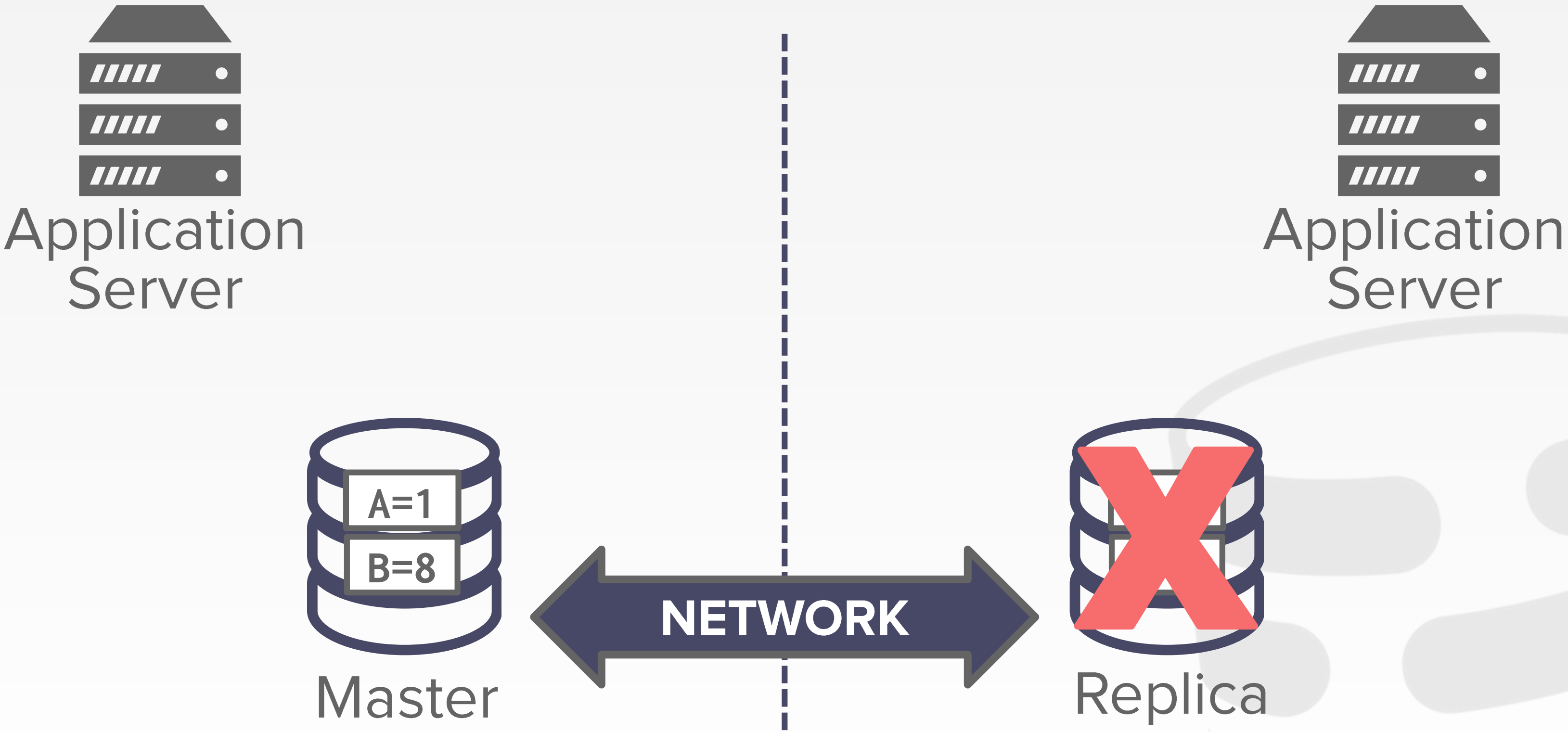
CAP – CONSISTENCY



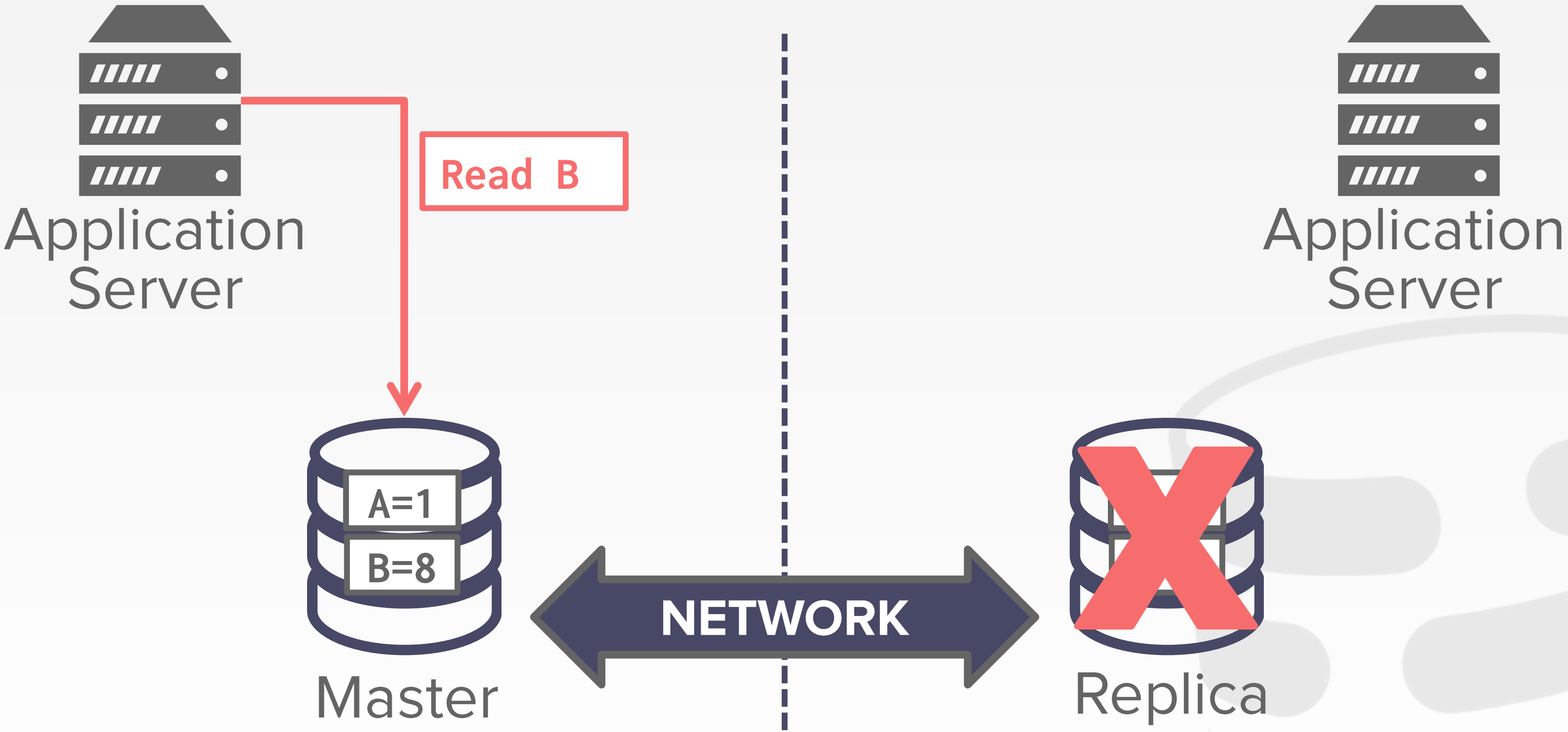
CAP – CONSISTENCY



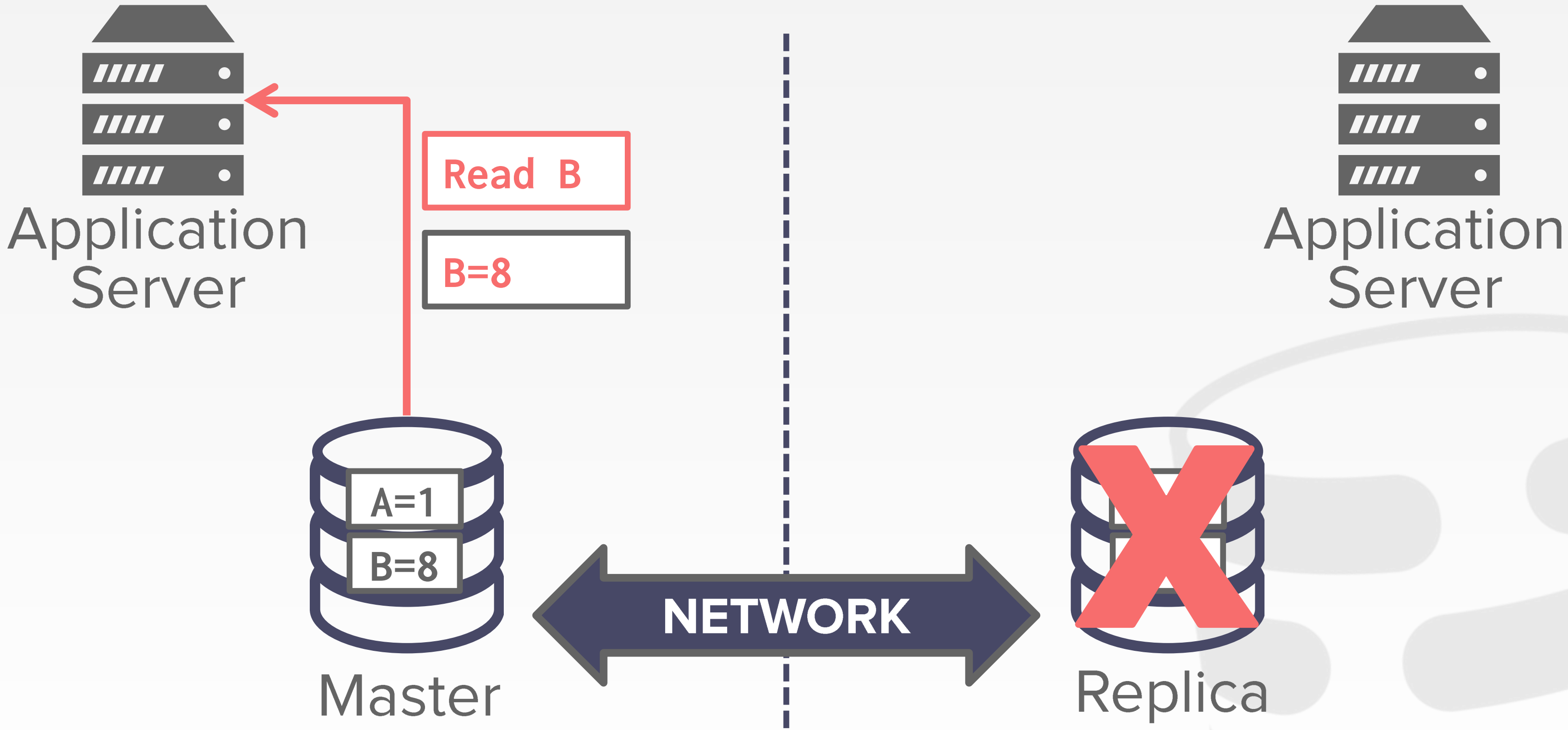
CAP – AVAILABILITY



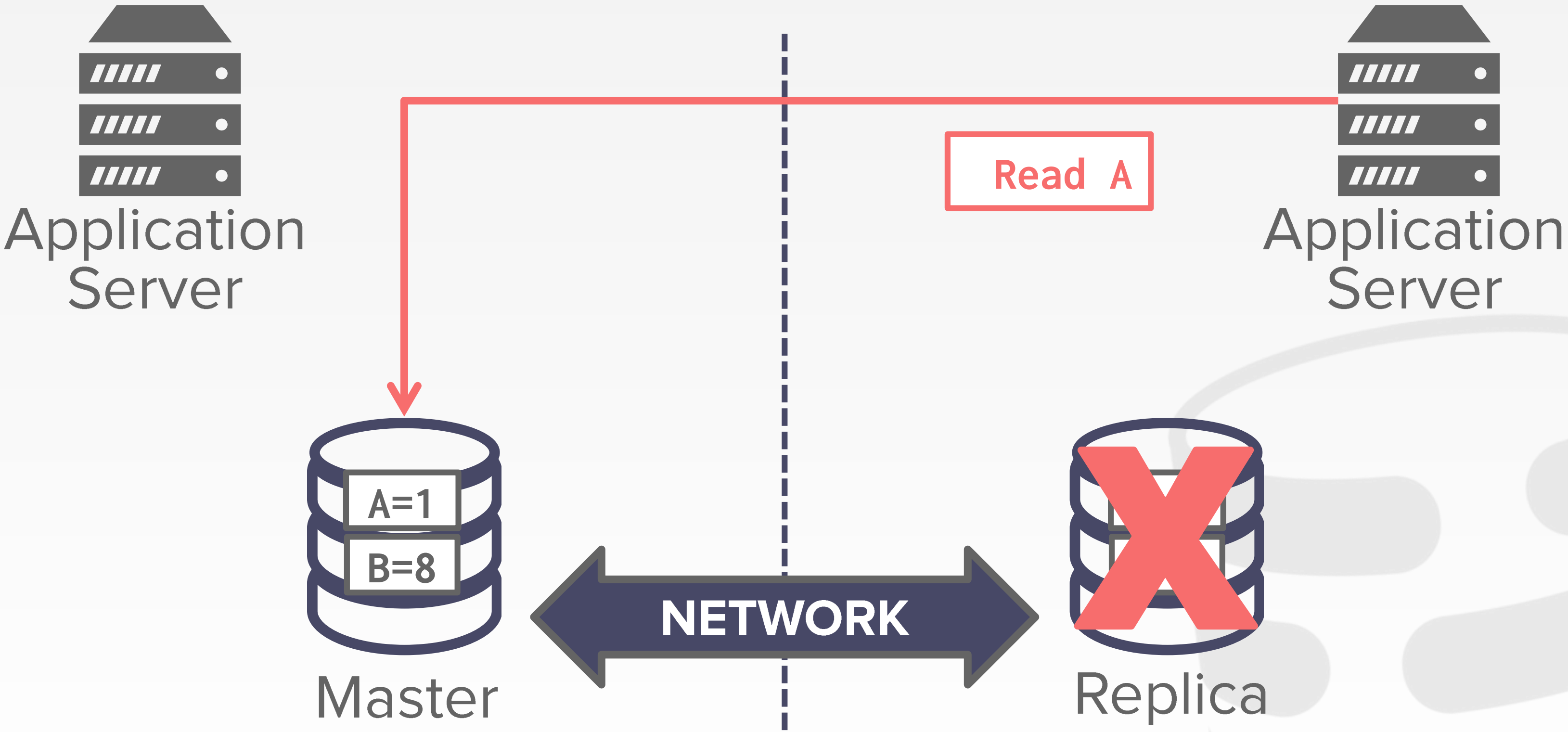
CAP – AVAILABILITY



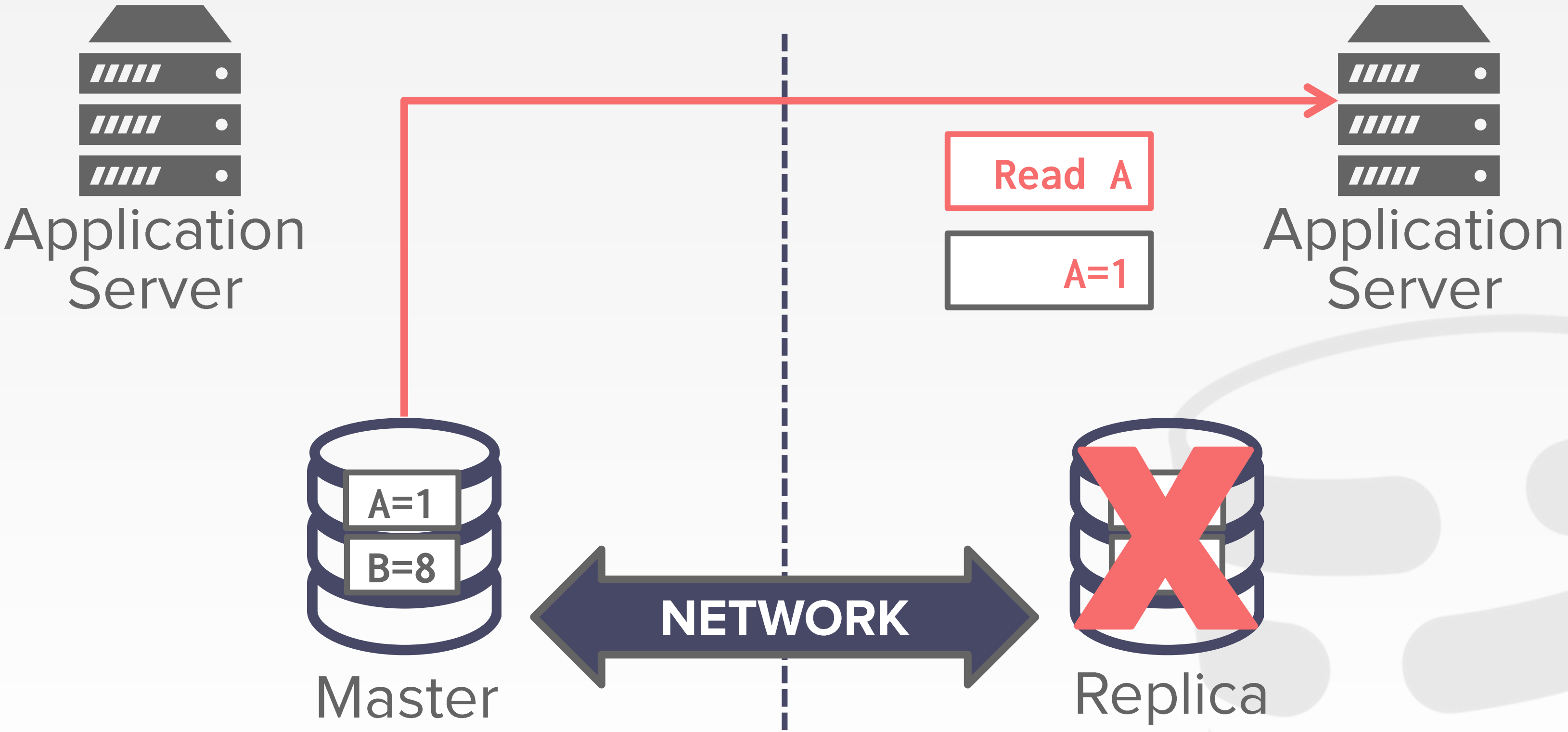
CAP – AVAILABILITY



CAP – AVAILABILITY

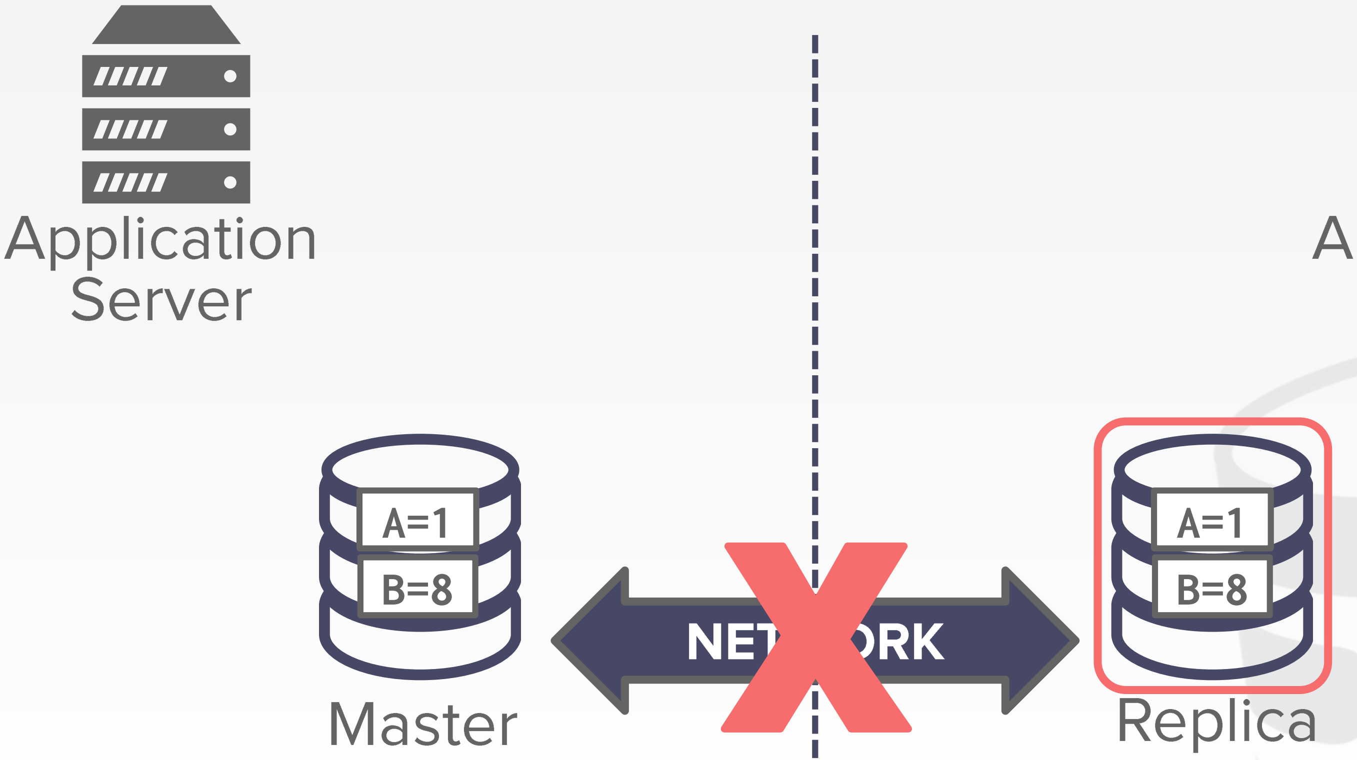


CAP – AVAILABILITY

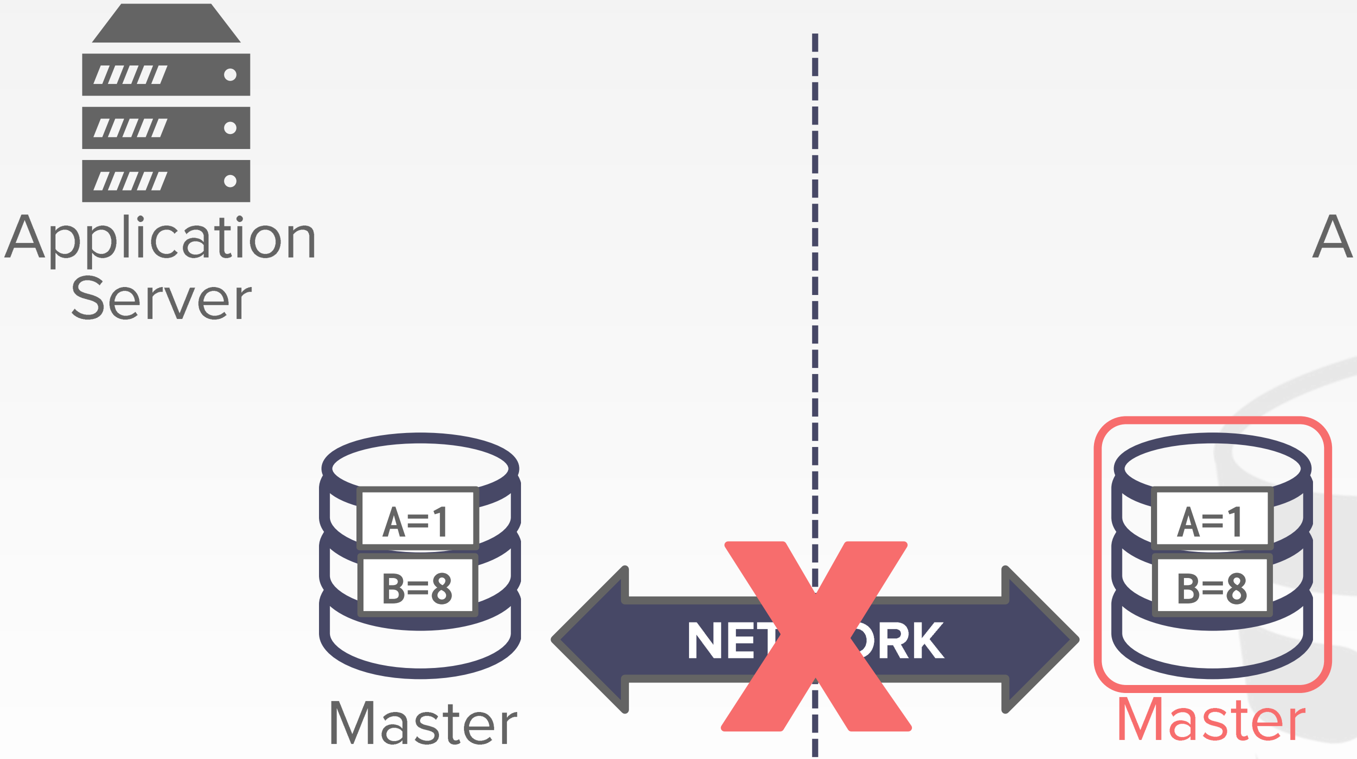


CARNEGIE MELLON DATABASE GROUP

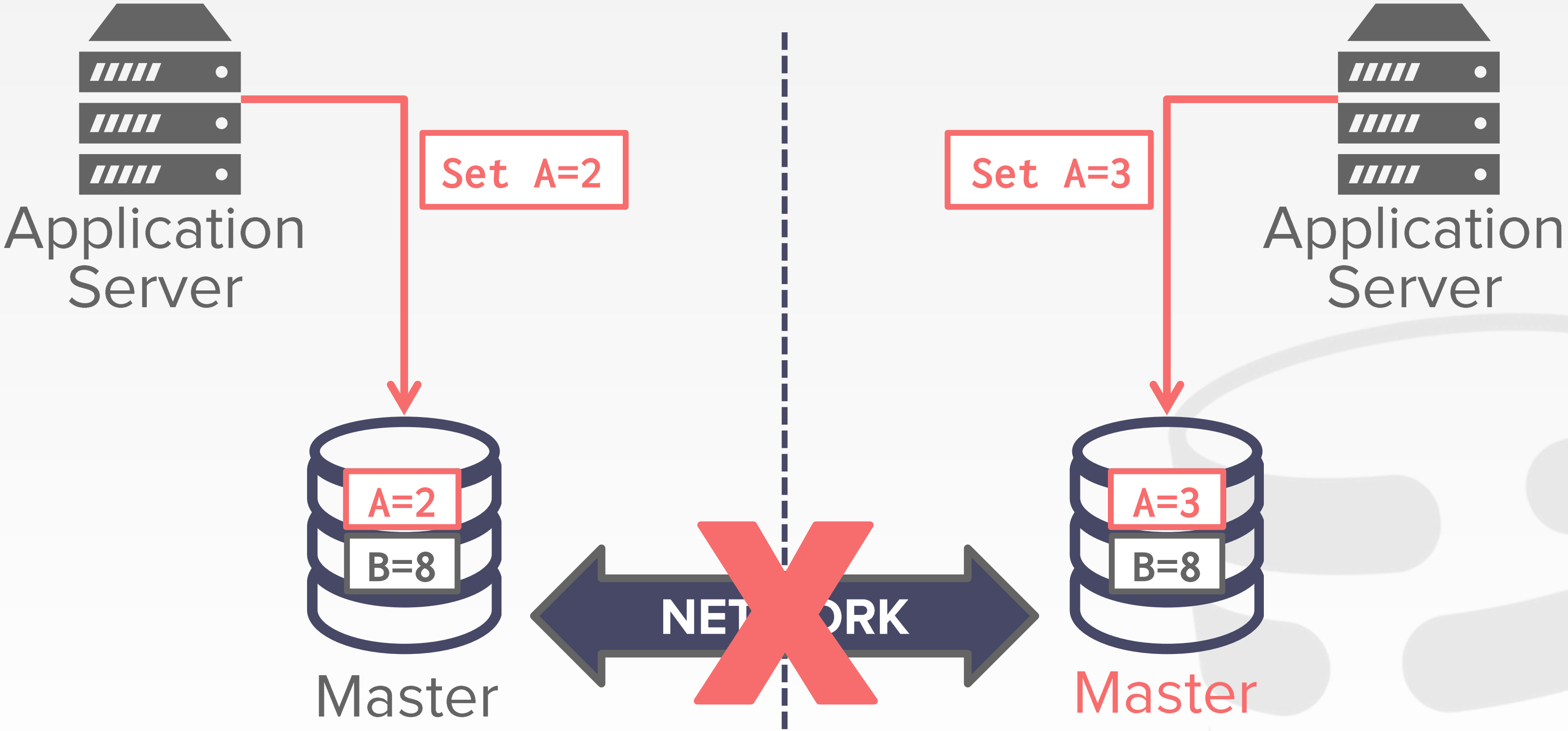
CAP – PARTITION TOLERANCE



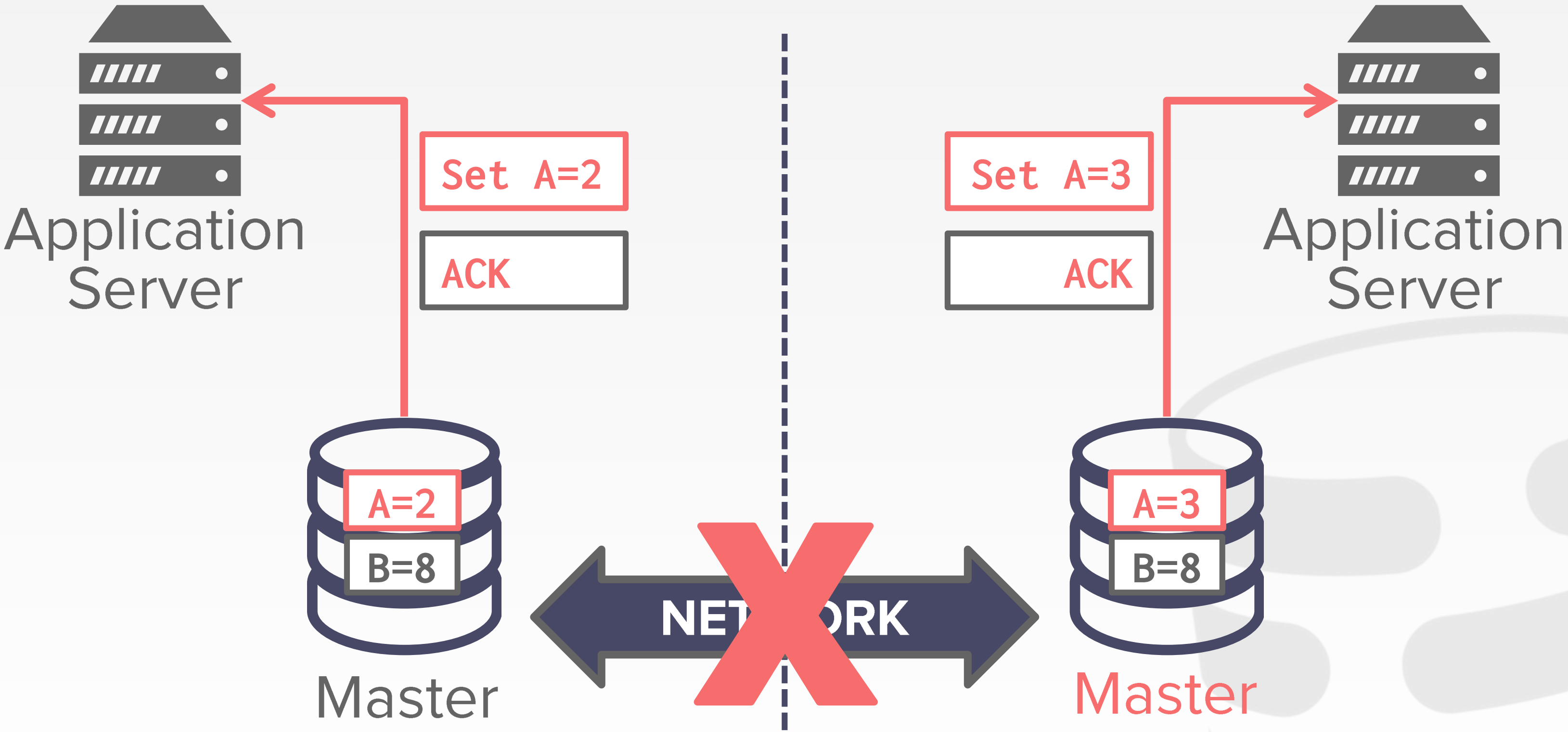
CAP – PARTITION TOLERANCE



CAP – PARTITION TOLERANCE

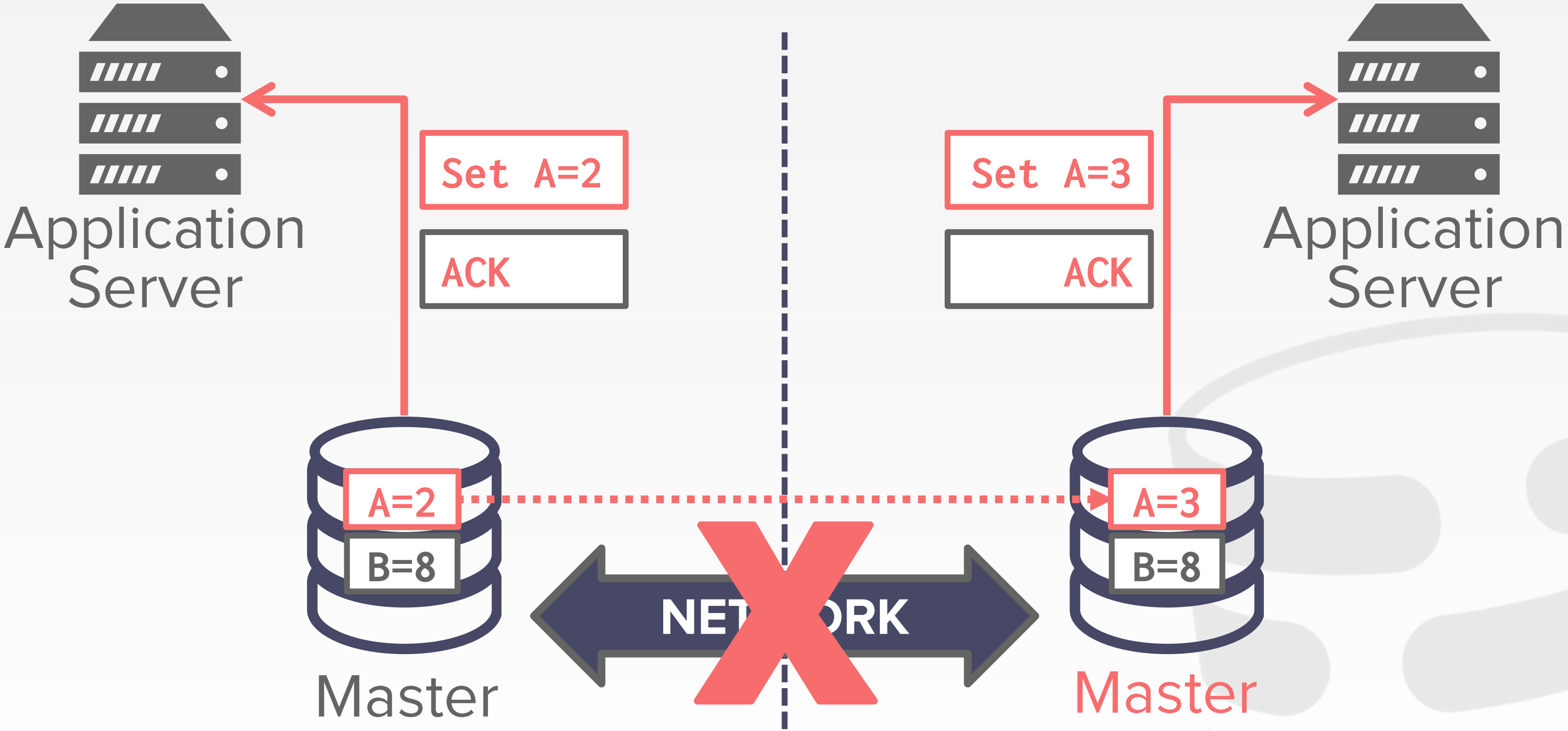


CAP – PARTITION TOLERANCE



CARNEGIE MELLON DATABASE GROUP

CAP – PARTITION TOLERANCE



CARNEGIE MELLON DATABASE GROUP

CAP FOR OLTO DBMSs

How a DBMS handles failures determines which elements of the CAP theorem they support.

Traditional/NewSQL DBMSs

→ Stop allowing updates until a majority of nodes are reconnected.

NoSQL DBMSs

→ Provide mechanisms to resolve conflicts after nodes are reconnected.

CONCLUSION

I have barely scratched the surface on distributed txn processing…

It is **really** hard to get right.

More info (and humiliation):
→ Kyle Kingsbury's Jepsen Project

NEXT CLASS

Distributed OLAP Systems