Navigating Transactions

ACID Complexity in Modern Databases

Shivji Kumar Jha

Shivji Kumar Jha

Data Platforms & OSS

- Databases, streams, app architecture
- Loves open source software (OSS)
- And communities (meetups)
- Regular speaker (talk # 23)
- Staff Engineer at Nutanix

https://www.linkedin.com/in/shivjijha

https://youtube.com/@ShivjiKumarJha

https://t.me/theDbShots

Contents

- Transactions & ACID
- Implementing Transactions
- Distributed transactions
- Cloud scale databases

Transactions

Transactions

Historical Perspective

Almost all relational and even some non relational databases

Most of them follow system R - first SQL DB by IBM in 1975.

General ideas has remained same over 45+ years

MySQL, Postgres, Oracle, SQL server have similar transactions

Transactions

Historical Perspective

NoSQL Camp

Transactions are antithesis of scalability

Large scale systems have to abandon transaction

Go for good performance and high availability

SQL Camp

Transactions are essential requirements for serious applications with valuable data

Transactions

Historical Perspective

Both are exaggerated!

Trade-offs!

NoSQL Camp

Transactions are antithesis of scalability

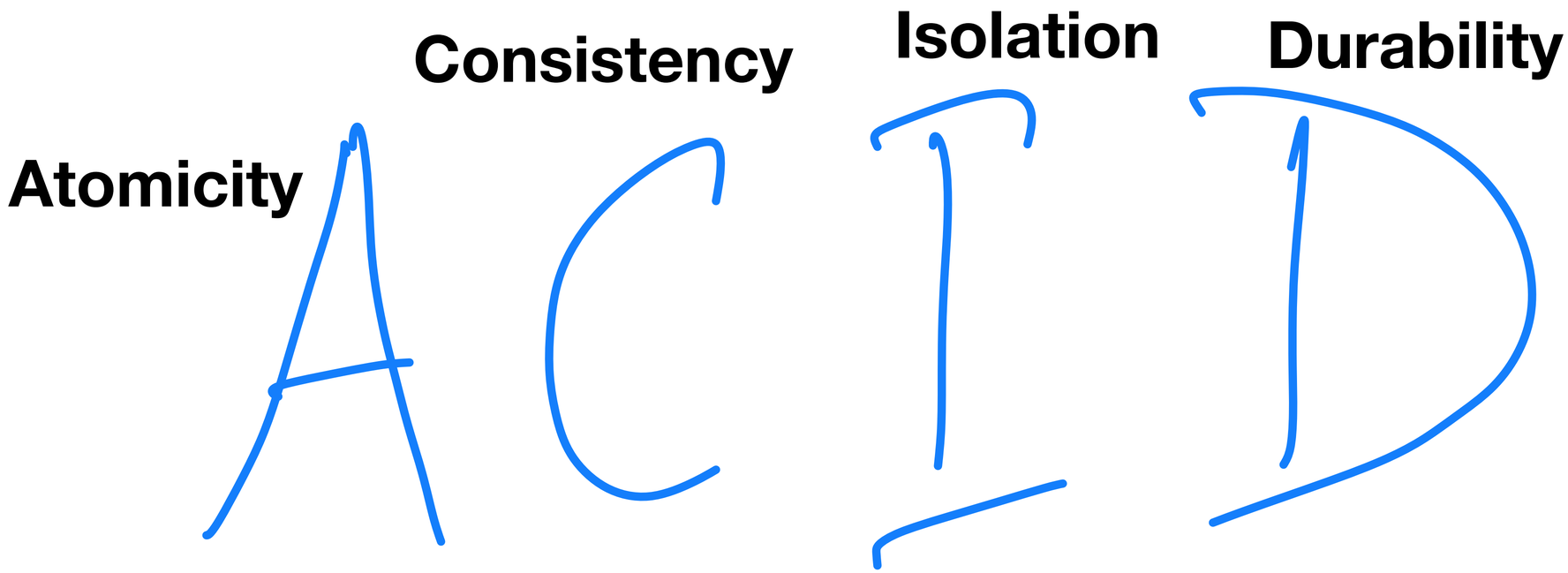
Large scale systems have to abandon transaction

Go for good performance and high availability

SQL Camp

Transactions are essential requirements for serious applications with valuable data

Coined in 1983 by Theo Harder and Andreas Reuter



Coined in 1983 by Theo Harder and Andreas Reuter

Transactions

The slippery slope!

In practice, one database's implementation of ACID does not equal another's implementation

For example, a lot of ambiguity in meaning of "isolation"

Devil is in details!

When a system claims ACID, unclear what guarantees it provides

ACID has unfortunately become a marketing term

How about

BASE?

Basically Available, Soft state
& Eventual Consistency

Atomicity

ACID

- ALL OR NOTHING!
- Ability to undo (abort)
- Easy to RETRY transactions

Are retries really safe with transactions?

Are retries really safe with transactions?

- Network failed while server tried to acknowledge commit to client. Retrying means executing twice. Idempotency or de-duplication required in app!
- What if the error is due to overload?
- Transient or permanent error?
- What if transaction had side effects? Send email again?
- What if client fails while retrying? Data lost?

Consistency

ACID

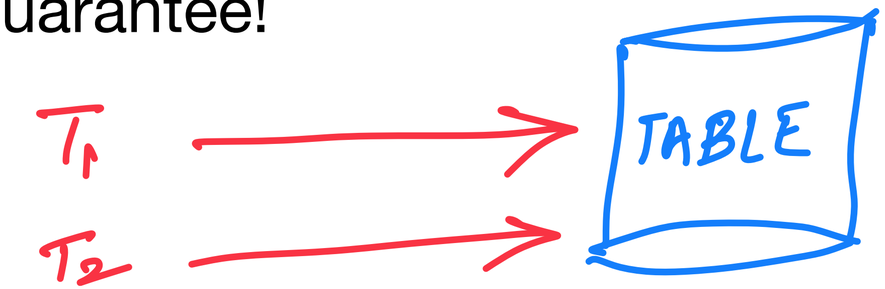
- Certain statements about data (invariants) always true
- Database can’t promise
- Application specific guarantees
- Most weakly defined property in ACID

	ACCOUNT "A"	ACCOUNT "B"
BEFORE	200	100
AFTER	190	110

Isolation

ACID

- Many clients access data at the same time
- Accessing same records you can run into concurrency problems (race)
- Database guarantees concurrently executing transactions are isolated
- Textbook definitions- serializability - same result as running serially
- In practice, serializability rarely used because of performance penalty
- Actually snapshot isolation. Much weaker guarantee!



Durability

ACID

- Once committed, data written will not be forgotten
- Even if there is a hardware fault or database crashes
- Single node - write to HDD or SDD
- Databases usually uses a write ahead log & dirty cached pages
- Replicated databases - written to multiple nodes, wait until that happens!
- No such thing as perfect durability

Picture: https://www.alphr.com/tell-regular-or-ssd-hard-drive/

Implementing Transactions

Implementing Transactions

Balance between two problems

Improve Efficiency

Allow transactions to execute concurrently

Preserve Correctness

Ensure concurrently executing transactions preserve ACID properties

Implementing Transactions

Balance between two problems

Improve Efficiency

Allow transactions to execute concurrently

Preserve Correctness

Ensure concurrently executing transactions preserve ACID properties

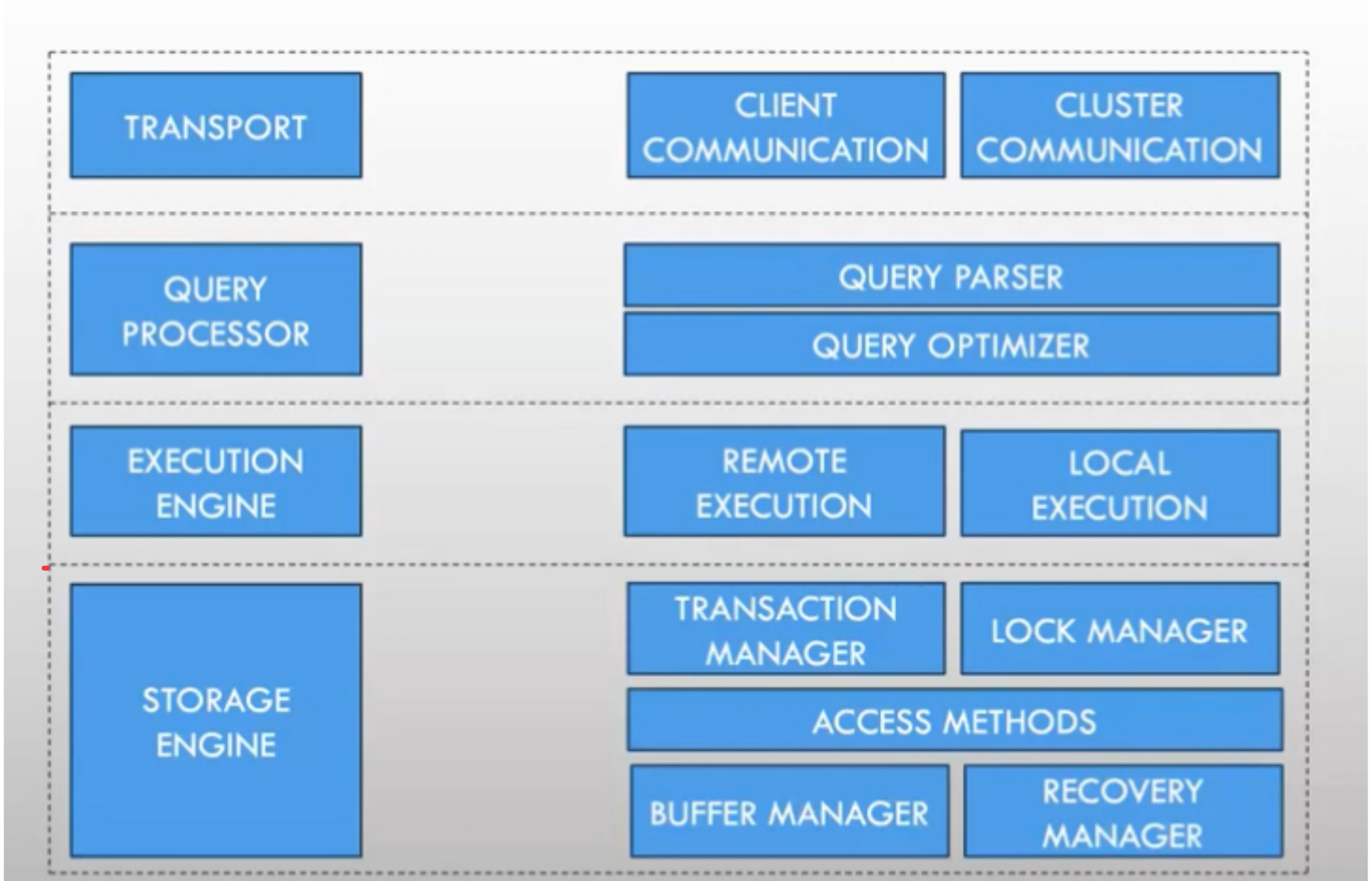
Concurrently executing transactions can cause read & write anomalies

Isolation levels Presence or absence of read & write anomalies

Concurrency Control How transactions are scheduled & executed

Single Node Transactions

Storage Engine's responsibility



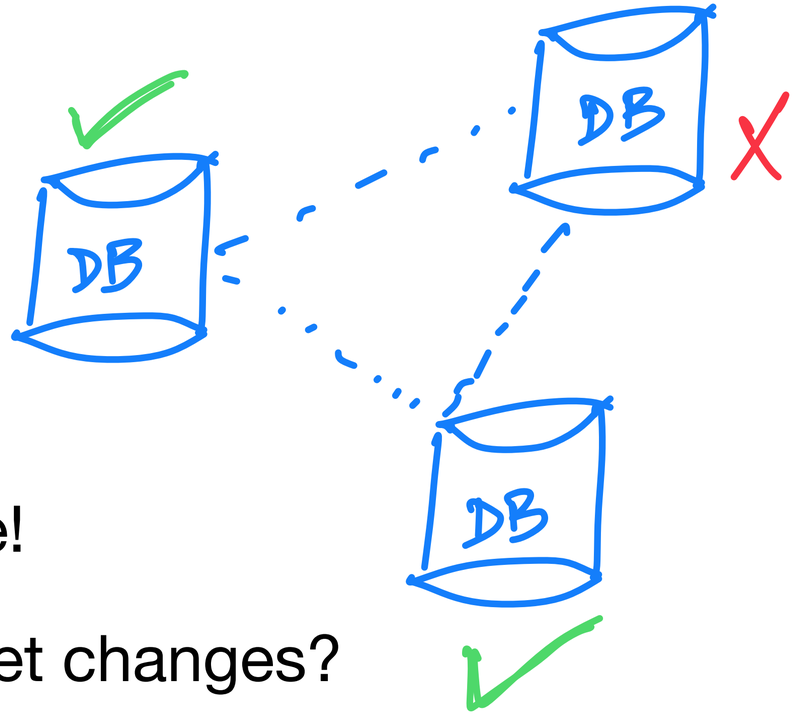
Distributed Transactions

What if multiple nodes are involved in a Transaction?

Send a commit request to each node & independently commit?

Send a commit request to each node & independently commit?

- Some SUCCESS, some FAILURES!
- Inconsistency between nodes
- Abort if some FAILURES?
  - But you can’t go back on a committed promise!
  - How about a compensating transaction to offset changes?
  - Where does this responsibility sit? DB or App?
- OR commit only if everyone promises to commit?

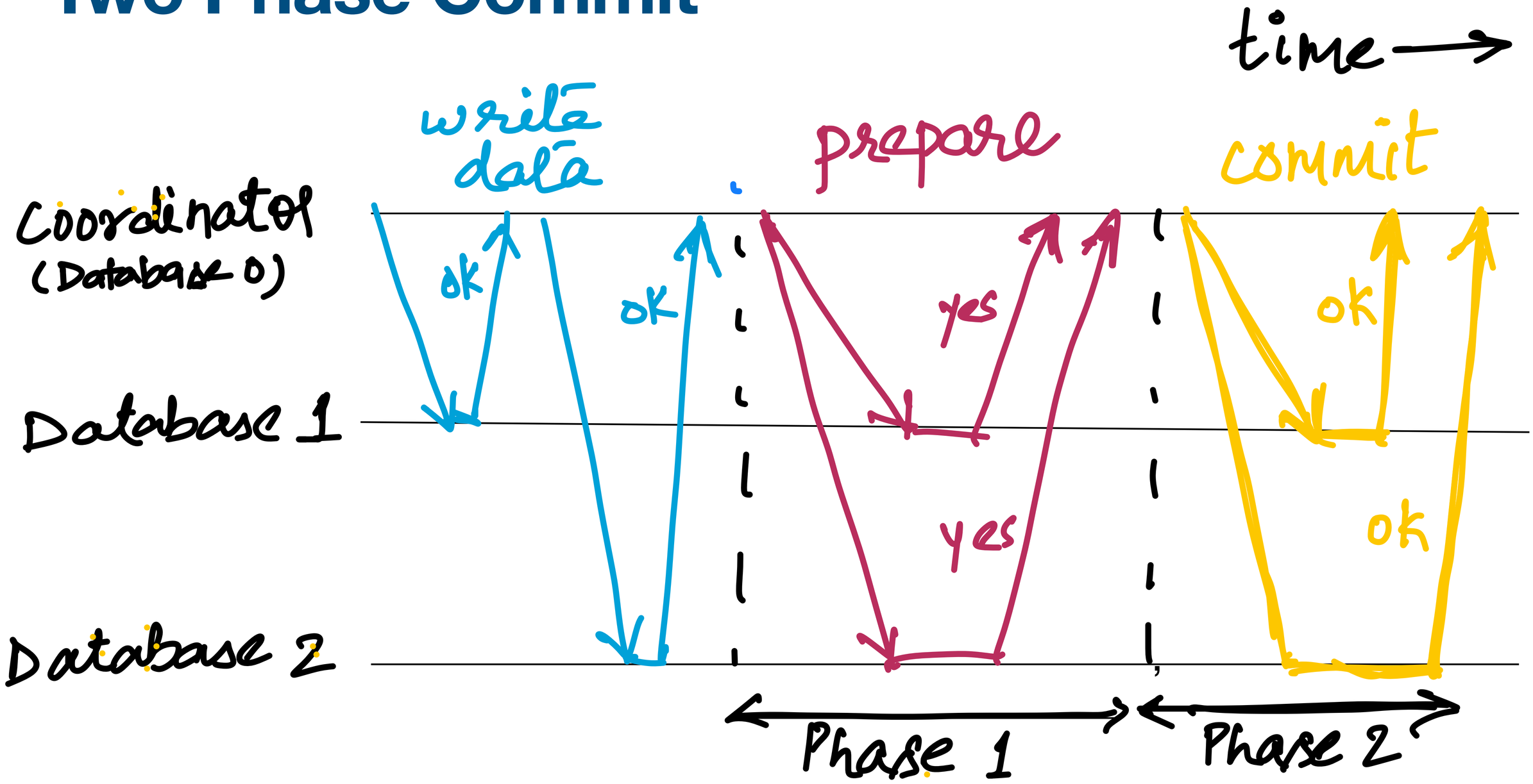


Atomic Commitment

Atomic Commitment

- A transaction will not commit even if one of the participating votes against it.
- Failed processes have to reach the same conclusion as the rest of the cohort.
- Does not work in the case of Byzantine failures- process can’t lie!
- Cohorts can not choose, influence or change proposed transaction, they can only vote on whether or not they are willing to execute it.
- Executed by transaction manager or coordinator
- Example: MySQL , Postgres, dynamoDB, spanner, Kafka for producer and consumer interactions.

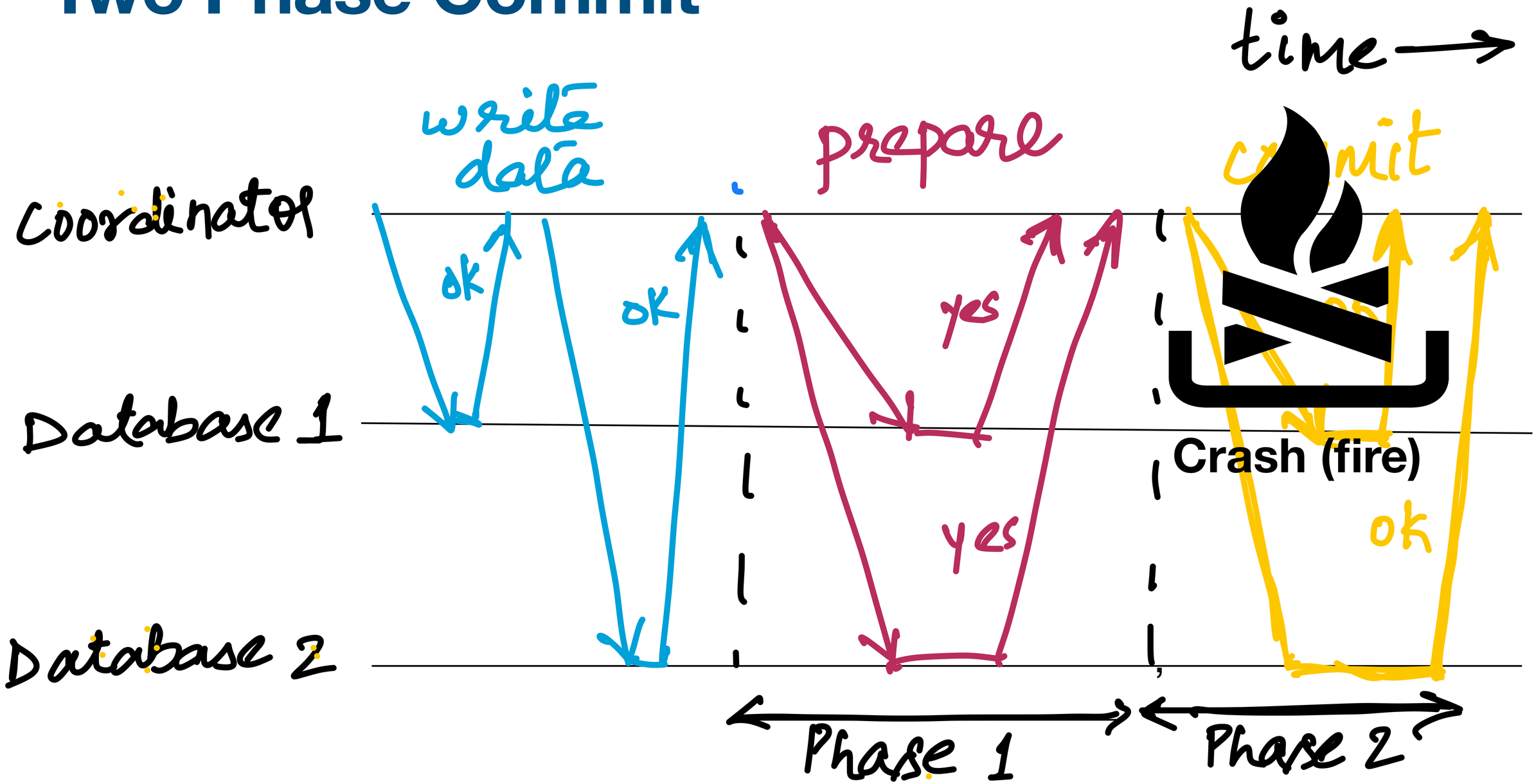
Two Phase Commit



Two Phase Commit

- Coordinator (or transaction manager) a library within database server
- When database is ready to commit, coordinator starts phase 1
- Coordinate sends prepare request to each node. Are you able to commit?
- Coordinator tracks response from each participant
- If all participants vote YES , coordinator sends COMMIT request in phase 2
- If any participant says NO, coordinator sends ABORT to all nodes in phase 2
- Coordinator must write decision in its log on disk to handle crashes

Two Phase Commit



Coordinator Failures

Two Phase Commit

- Two points of no return
- 1. If Participant says YES, it has to commit if coordinator asks to.
- 2. If coordinator decides once, decision is irrevocable
- If participant said yes and didn’t hear back from coordinator, wait forever!
- Coordinator must persist decision before it sends participants
- If no COMMIT record persisted in coordinator, abort on recovery

Three Phase Commit?

- 2PC is a blocking protocol
- 3PC is non blocking
- Difficult to implement in practice. Not well adopted!
- 2 PC quite well adopted in spite of known problems.

Distributed Transactions in Practice

- Carries a heavy performance penalty
  - Additional fsync required for crash recovery
  - Addition network round trips
- Distributed Transactions in MySQL are reported to be over 10 times slower than single node transactions.

Other Choices?

- Single leader? Everyone else executes same transactions in same order!
  - Manually selected leader?
  - Automatic selection of leader?
- Offloaded same problem to different time. Well less frequently!
- Use Consensus Algorithms: Zookeeper (ZAB), etcd (Raft) , Paxos?
- Global transaction order by reaching consensus on sequencer? Calvin(FaunaDB)
- 2PC over consensus groups per shard? Enter Google’s Spanner!

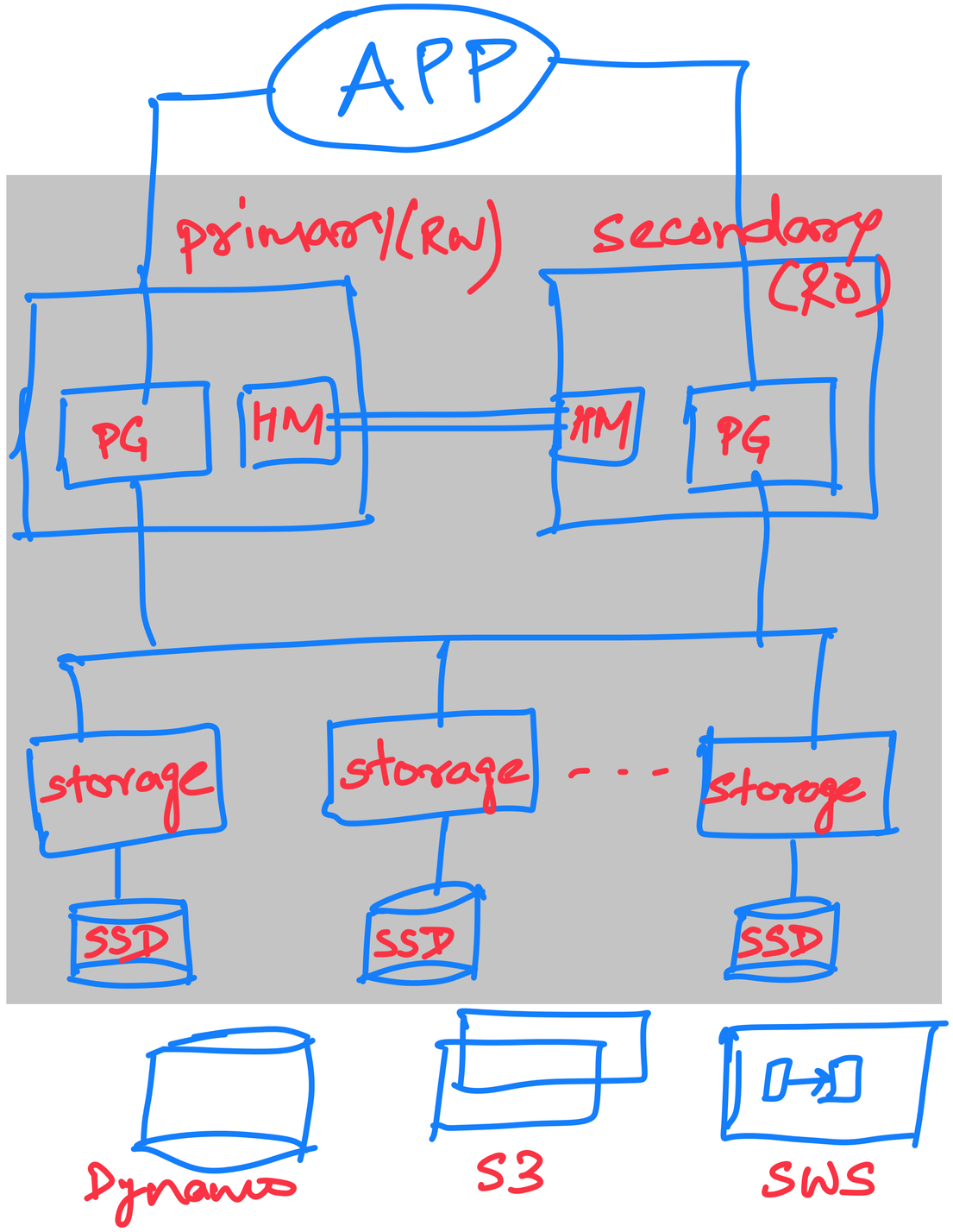
Transactions in Modern Databases

Amazon Aurora

Amazon Aurora

Quick Overview

- Fully managed relational RDS
- Service Oriented Design
- Separation of Compute, storage
- multi-tenant scale out storage
- Segmented redo log
- Throughput 5x MySQL, 3x Postgres



Aurora Functional Separation

DB instance

Query Processing

Access Methods

Transactions

Locking

Page Cache

Undo Management

Storage fleet

Redo logging

Materialisation of data blocks

Garbage collection

Backup / Restore

Aurora: Quorum Style distributed coordination

Not 2PC, to avoid network chatter!

Read set & write set must overlap on at least one copy

$$V_r + V_w > V$$

Write set must overlap with previous write sets

$$V_w > \frac{V}{2}$$

6 copies (3 AZs)

$V = 6$

$V_w = 4$

$V_r = 3$

(configurable)

Amazon DynamoDB

Dynamo DB

Highly available, weaker consistency

- Always “on” Key Value store, single key operations
- Sacrifice consistency under failure scenarios. Eventually consistent!
- Extensive use of object versioning, branching OK, resolve on read
- Example: shopping cart; merges carts. Can’t loose write, deleted can appear
- Consistency among replicas using quorum like technique (sloppy quorum)
- Gossip based distributed failure detection (hinted handoffs)

Dynamo DB

Sloppy Quorum & hinted handoff

- Each data item is replicated at N hosts. N distinct physical nodes
- List of nodes storing a key is called it’s preference list
- R : minimum no of nodes in successful read operation
- W: minimum no of nodes in successful write operation

Dynamo DB

Sloppy Quorum & hinted handoff

- Each data item is replicated at N hosts. N distinct physical nodes
- List of nodes storing a key is called it's preference list
- R : minimum no of nodes in successful read operation
- W: minimum no of nodes in successful write operation

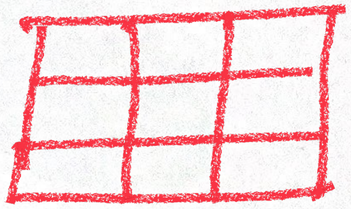
$R + W > N$

Typically, $(N, R, W) \simeq (3, 2, 2)$

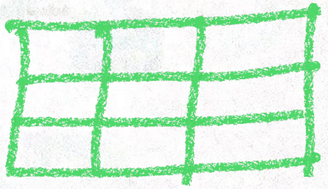
High performance read; $R = 1, W = N$

Max write availability; $W = 1$

```
//Check if customer exists
Check checkItem = new Check()
        .withTableName("Customers")
        .withKey("CustomerUniqueId")
        .withConditionExpression("attribute_exists(CustomerId)");

//Update status of the item in Products
Update updateItem = new Update()
        .withTableName("Products")
        .withKey("BookUniqueId")
        .withConditionExpression("expected_status" = "IN_STOCK")
        .withUpdateExpression("SET ProductStatus = SOLD");

//Insert the order item in the orders table
Put putItem = new Put()
        .withTableName("Orders")
        .withItem("{"OrderId": "OrderUniqueId", "ProductId" :"BookUniqueId", "CustomerId"
:"CustomerUniqueId", "OrderStatus":"CONFIRMED","OrderCost": 100}")
        .withConditionExpression("attribute_not_exists(OrderId)")

TransactWriteItemsRequest twiReq = new TransactWriteItemsRequest()
            .withTransactItems([checkItem,putItem, updateItem]);

//Single transaction call to DynamoDB
DynamoDBclient.transactWriteItems(twiReq);
```



customers



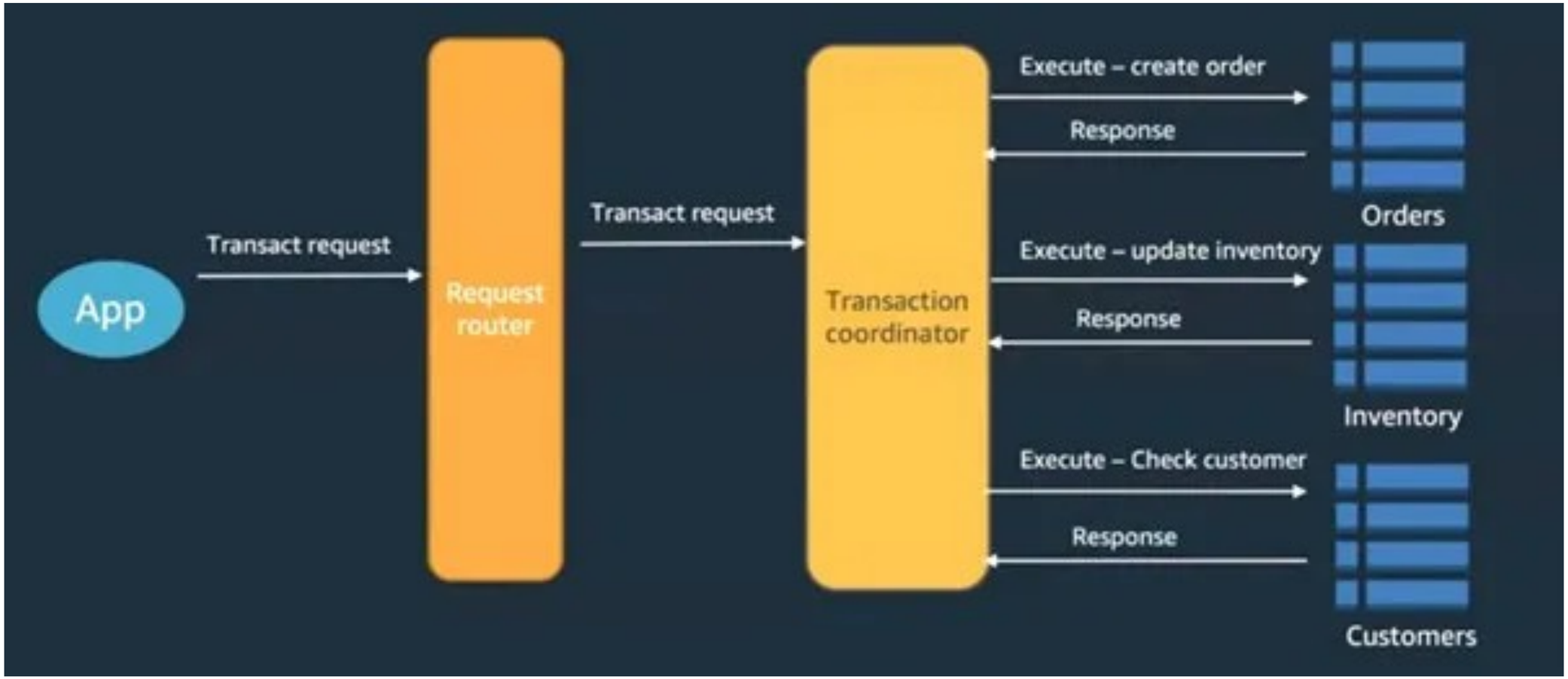
inventory

orders

Listing 1: DynamoDB Write Transaction Example (purchase a book)

Transaction coordinator

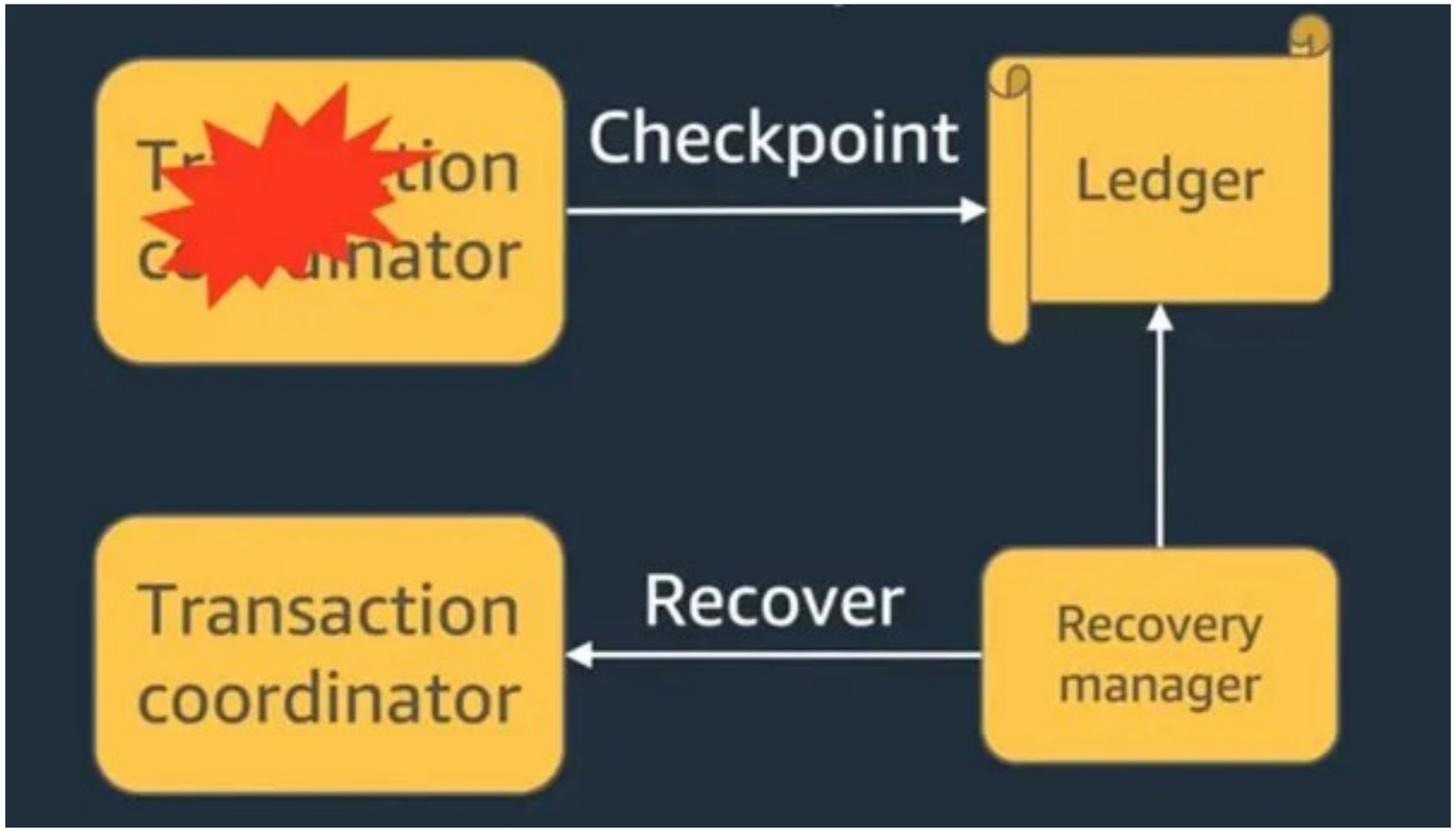
DynamoDB



https://www.infoq.com/articles/amazon-dynamodb-transactions/

Transaction coordinator failure/recovery

DynamoDB



https://www.infoq.com/articles/amazon-dynamodb-transactions/

Google Spanner

Spanner

Google’s globally distributed SQL* database

- Also inspiration for cockroachDB and yugabyteDB
- Tables with rows, columns and versioned values
- Supports transactions and SQL based query language
- Replication configs dynamically controlled at fine grain by apps - which data-centre’s to use, how far from users(read latency), how far are replicas (write latency), how many replicas
- Clients automatically failover between replicas
- Data dynamically moved between data-centres to balance resources

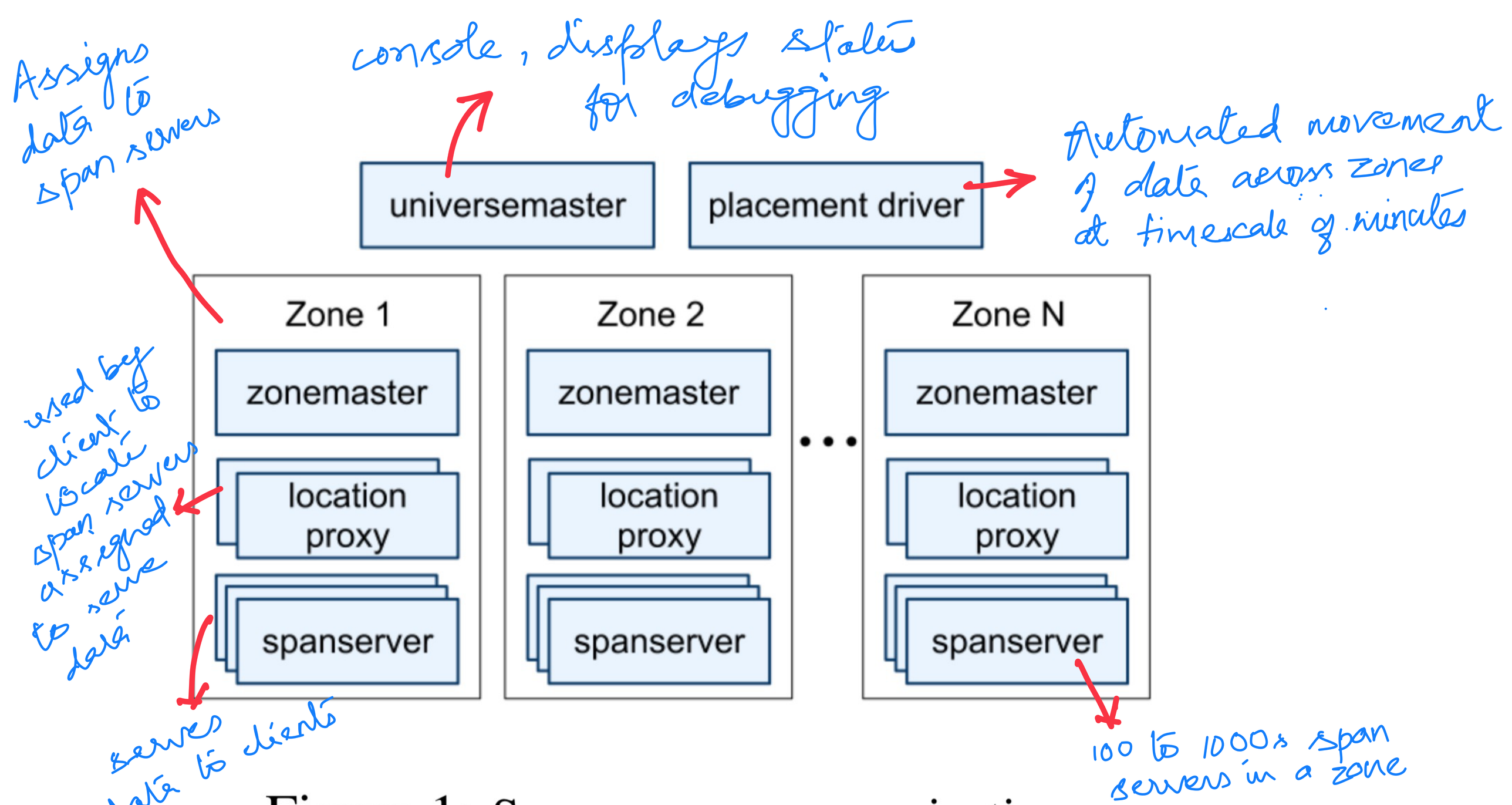
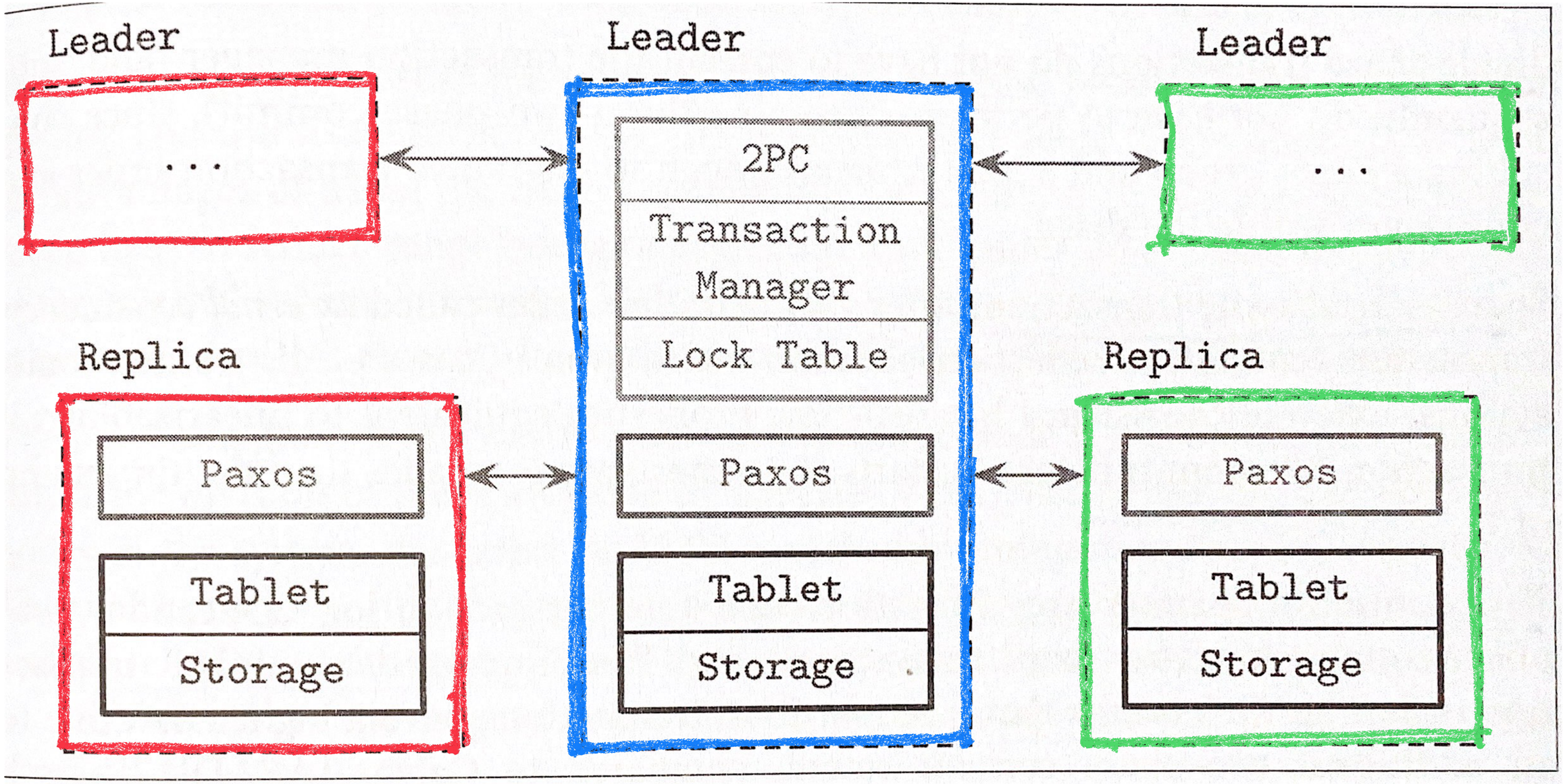


Figure 1: Spanner server organization.

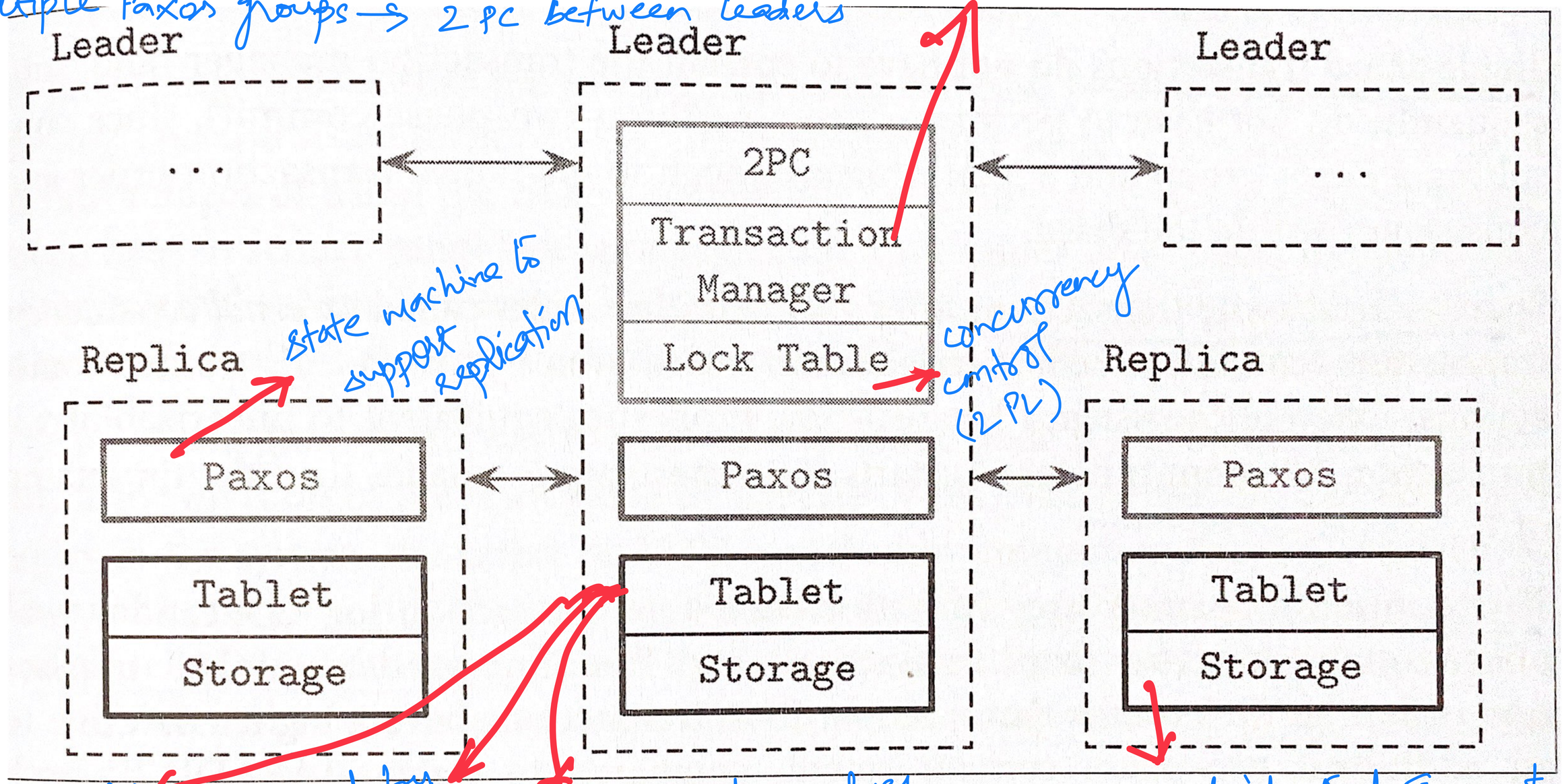
Span server stack & transactions



3 Paxos groups

Transactions with single Paxos group bypass transaction manager

Multiple Paxos groups → 2 PC between leaders



References

Books

- Designing Data-Intensive Applications (Chapter 7 & 9) By Martin Kleppmann
- Database Internals (Chapter 5 & 13) By Alex Petrov

Whitepapers

- Amazon Aurora: Design considerations for high throughput cloud native relational databases
- Amazon Aurora: On avoiding distributed consensus….
- Dynamo: Amazon’s highly available key value store
- Distributed Transactions at scale in Amazon DynamoDB
- Spanner : Google’s Globally Distributed Database

Blog

https://www.infoq.com/articles/amazon-dynamodb-transactions/

Questions?

Staying in touch:

https://www.linkedin.com/in/shivjijha

https://youtube.com/@ShivjiKumarJha

https://t.me/theDbShots

https://www.slideshare.net/shiv4289/presentations