AWS DynamoDB

Transactions at Scale

Amrith Kumar

Senior Principal Engineer,
AWS, DynamoDB

Outline

- What is DynamoDB
- High-level architecture
- Transactions
- Q&A

What is DynamoDB

Fully Managed

What is DynamoDB

- Fully Managed

Multi-tenant

What is DynamoDB

- Fully Managed
- Multi-tenant

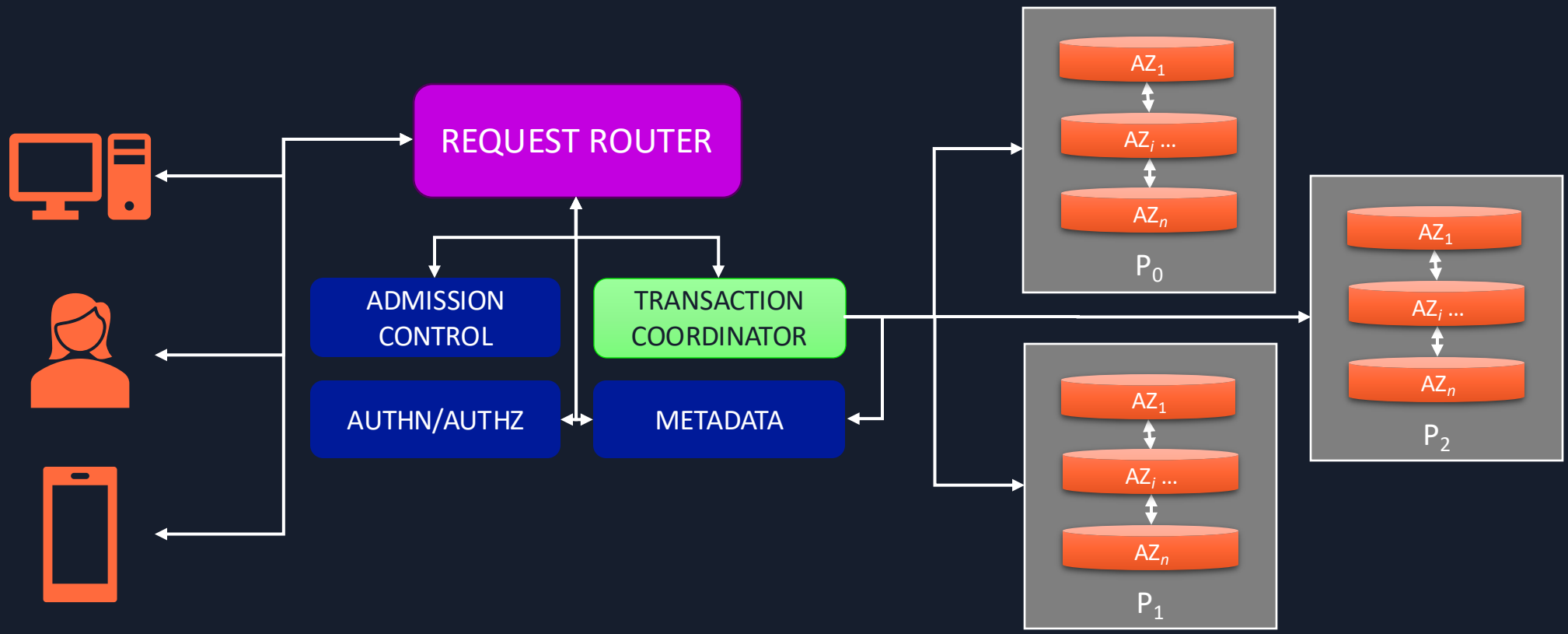
Predictable Latency at any scale

What is DynamoDB

- Fully Managed
- Multi-tenant
- Predictable Latency at any scale

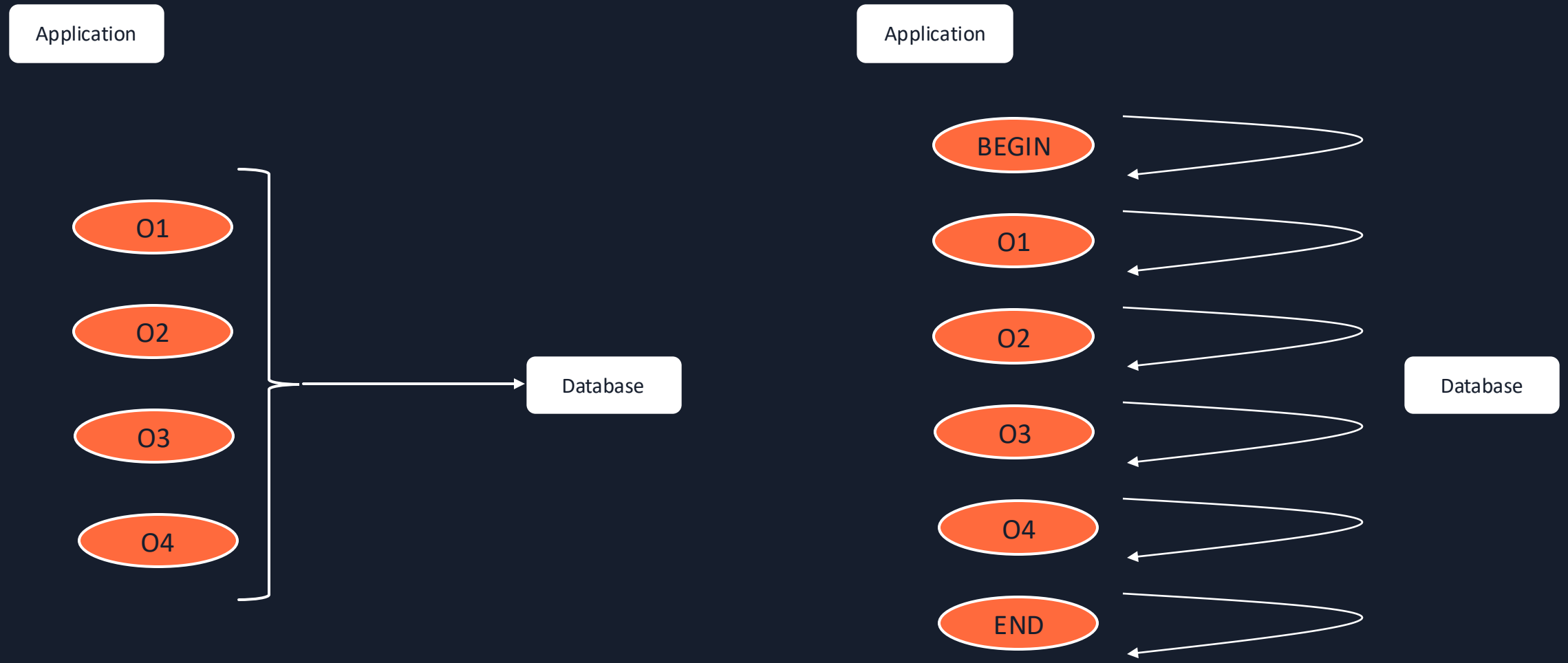
Non-Relational

High-level architecture



One shot vs. BEGIN-END

SINGLE REQUEST VS. MULTI REQUEST TRANSACTIONS



One shot vs. BEGIN-END

SINGLE REQUEST VS. MULTI REQUEST TRANSACTIONS

DynamoDB Single-Request

- Two types of transactions
  - TransactGetItems()
  - TransactWriteItems()

- Predictable latency
- No distributed locking
- Simple recovery

Traditional RDBMS - Multi-Request

- One transaction type
  - Intersperse reads and writes

- Unpredictable latency
- Locking nightmares
- Recovery complexity

Question

- What operations does DynamoDB provide?

DynamoDB operations

- Not Atomic
  - Query, Scan
  - Batch (Get, Put, Update, Delete)
- Atomic
  - Multi-statement transactions (explicitly atomic)
    - Mutating (write)
    - Non-mutating (read)
  - Single item (implicitly atomic)
    - GetItem, PutItem, UpdateItem, DeleteItem

Question

- As an item in the database is mutated over time, how many versions (history) does DynamoDB store?

DynamoDB data storage

- DynamoDB stores one version of an item (the latest)
- It stores 3 identical copies for fault tolerance
- No multi-version-concurrency-control (MVCC)

Question

- What are database “consistency” and “isolation”?

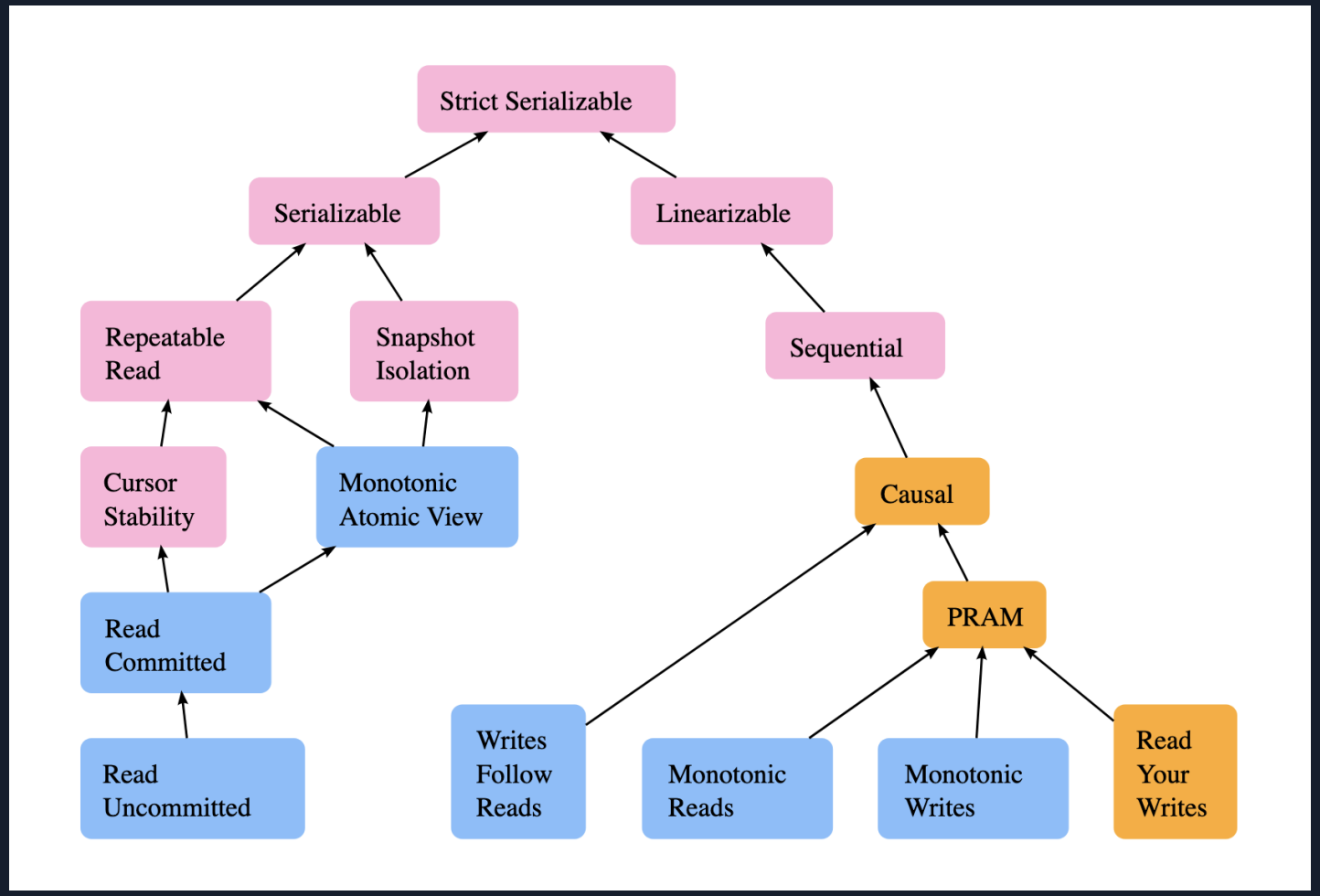
Consistency and Isolation

Consistency - moving a database from one valid state to another

Isolation - keeping concurrent transactions (operations) apart

- Consistency Levels
  - Strong consistency
  - Eventual consistency
  - … many others …
- Isolation Levels
  - Strict Serializable
  - Serializable
  - Read committed
  - … many others …

Database consistency models



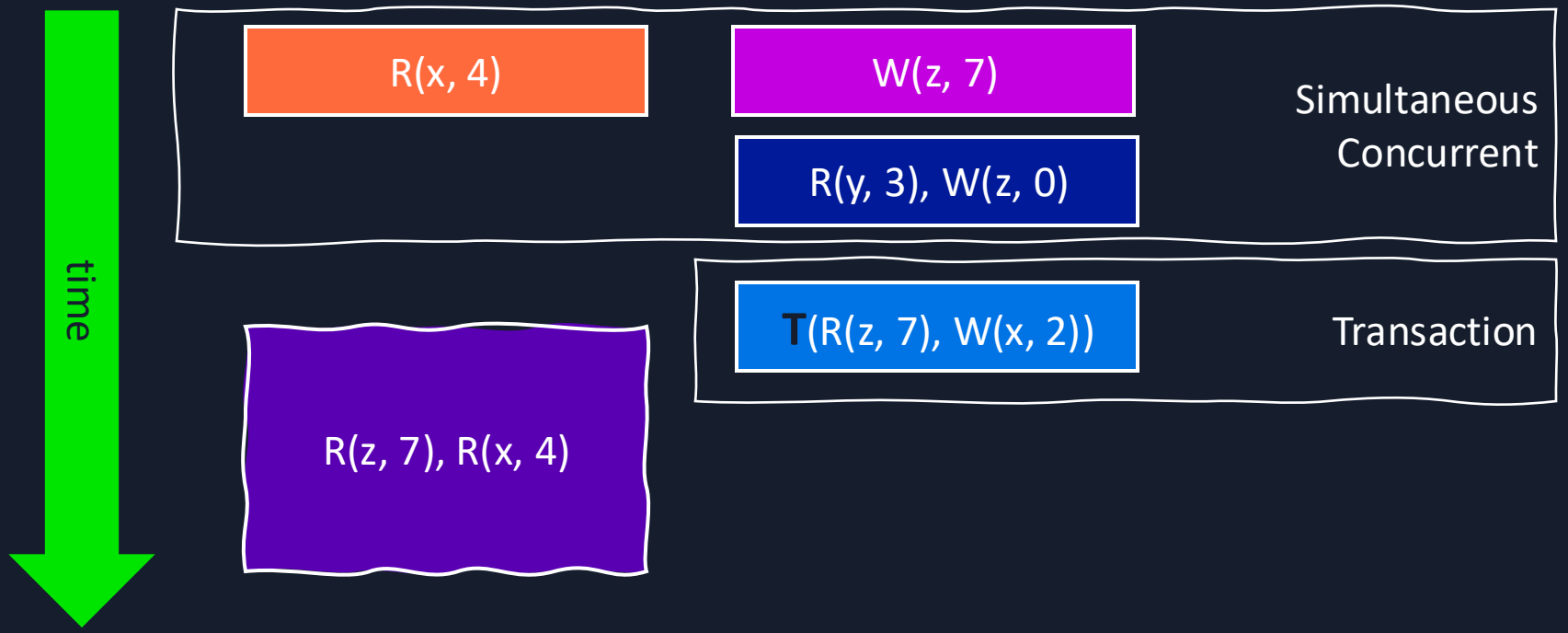
Source: https://jepsen.io/consistency/models. Copyright © 2016–2025 Jepsen, LLC. Reproduced with permission. Adapted from "Highly Available Transactions: Virtues and Limitations"; Bailis et al., VLDB 2013.

Question

- What are database histories?

Database histories (aka schedules)

- A sequence of interleaved operations, potentially representing multiple transactions over time
- Understand violations of isolation and consistency semantics



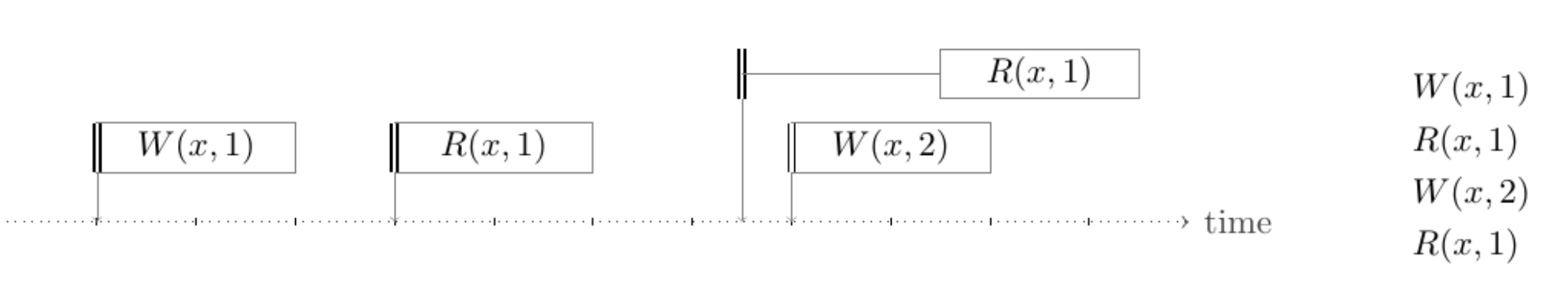
Stretch

Question

- What is “Serializability” or “Serializable Isolation”?

Serializable Isolation

an execution of concurrent operations that produces the same effect as some serial execution of those operations.



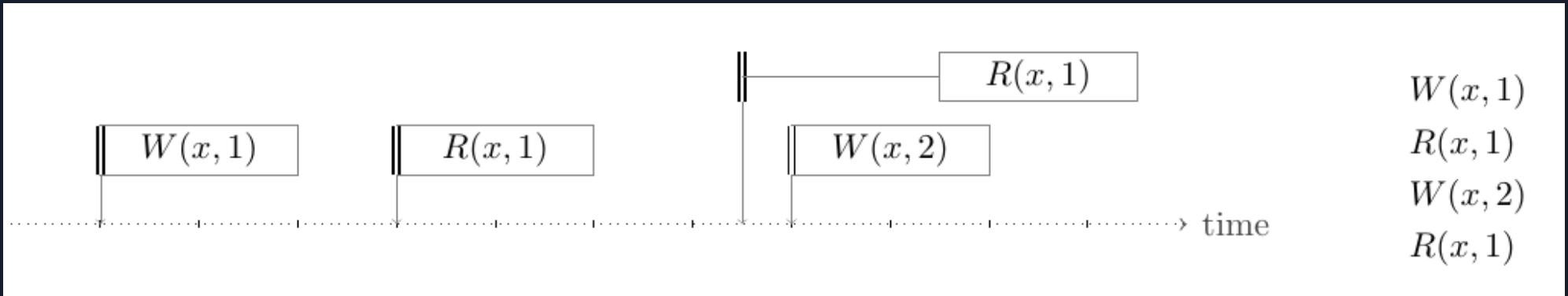
Question

- What is “timestamp ordering”?

Timestamp Ordering

Use timestamps to establish the logical order of execution of operations.

- *As long as operations appear to execute at their assigned time, serializability is achieved.*



DynamoDB Transactions - Preamble

Routing
Authentication
Authorization

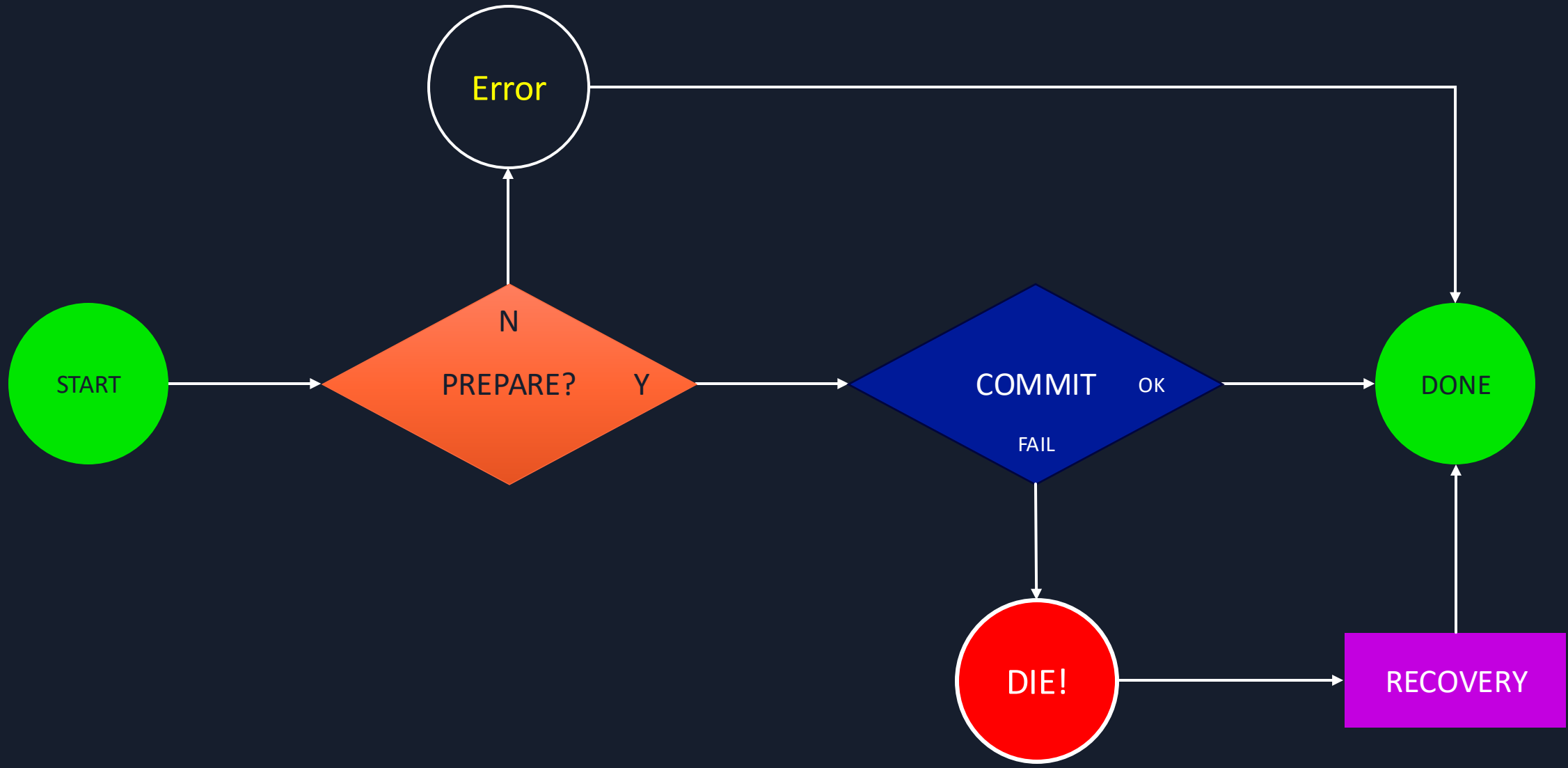
DynamoDB Transactions - Paperwork

Transaction Ledger

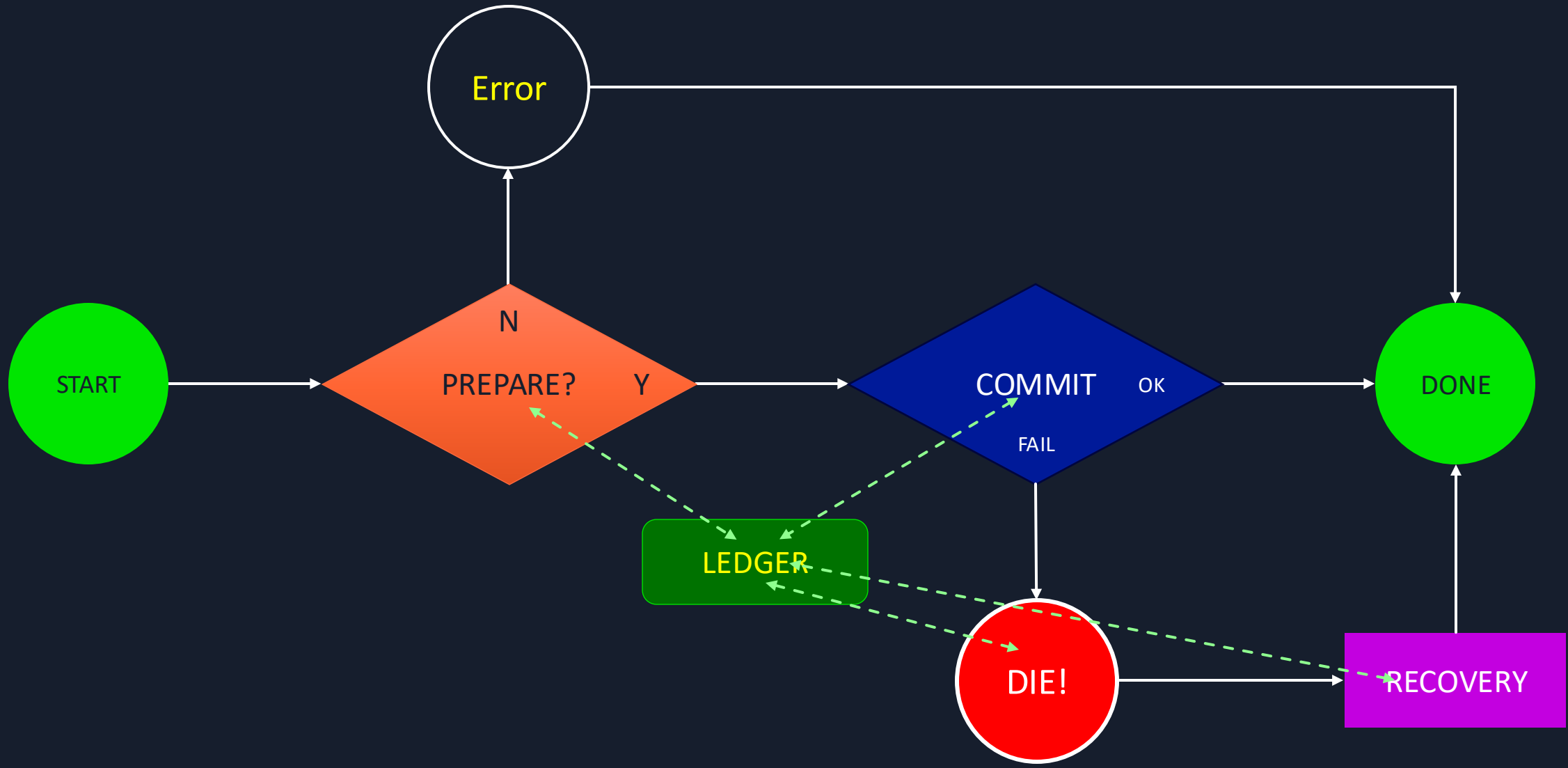
Question

- What is ”Two-phase Commit”?

Two-Phase Commit



Write Transactions – Transaction Coordinator



Write Transactions - Prepare

```
def processPrepare(PrepareInput input):
  item = readItem(input)

  if item != NONE:
    if evaluateConditionsOnItem(item, input.conditions)
      AND evaluateSystemRestrictions(item, input)
      AND item.timestamp < input.timestamp
      AND item.ongoingTransactions == NONE:
        item.ongoingTransaction = input.transactionId
        return SUCCESS
    else:
        return FAILED
  else: #item does not exist
    item = new Item(input.item)
    if evaluateConditionsOnItem(input.conditions)
      AND evaluateSystemRestrictions(input)
      AND partition.maxDeleteTimestamp < input.timestamp:
        item.ongoingTransaction = input.transactionId
        return SUCCESS
  return FAILED
```

Listing 3: `TransactWriteItem` protocol - Prepare phase

Write Transactions

Deletes?

Writes to nonexistent items

Write Transactions

No tombstones

Instead of maintaining tombstones for deleted items, which would incur both a high storage cost and garbage collection cost if items are frequently created and deleted, DynamoDB stores a partition-level max delete timestamp. When an item is deleted, if the deleting transaction's timestamp is greater than the current max delete timestamp, then the max delete timestamp is set to the transaction's timestamp.

Correct, but paranoid!

Write Transactions - Prepare

```
def processPrepare(PrepareInput input):
  item = readItem(input)

  if item != NONE:
    if evaluateConditionsOnItem(item, input.conditions)
      AND evaluateSystemRestrictions(item, input)
      AND item.timestamp < input.timestamp
      AND item.ongoingTransactions == NONE:
        item.ongoingTransaction = input.transactionId
        return SUCCESS
    else:
        return FAILED
  else: #item does not exist
    item = new Item(input.item)
    if evaluateConditionsOnItem(input.conditions)
      AND evaluateSystemRestrictions(input)
      AND partition.maxDeleteTimestamp < input.timestamp:
        item.ongoingTransaction = input.transactionId
        return SUCCESS
  return FAILED
```

Listing 3: `TransactWriteItem` protocol - Prepare phase

Write Transaction – Commit

```
def processCommit(CommitInput input):
  item = readItem(input)

  if item == NONE
    OR item.ongoingTransaction != input.transactionId:
    return COMMIT_FAILED

  applyChangeForCommit(item, input.writeOperation)
  item.ongoingTransaction = NONE
  item.timestamp = input.timestamp
  return SUCCESS
```

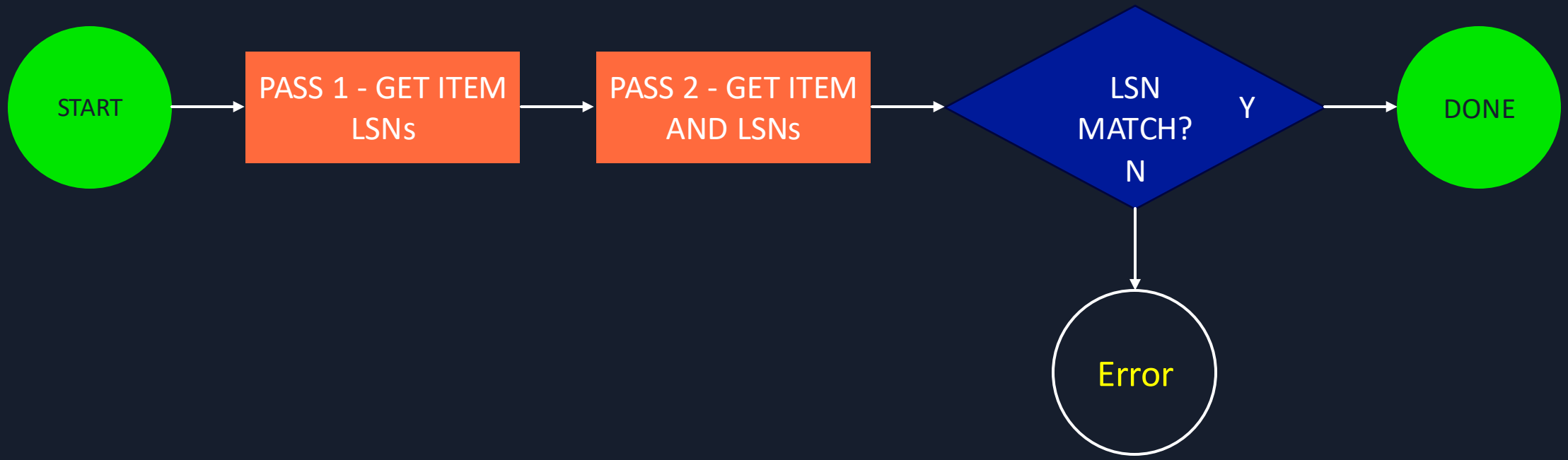
Listing 4: `TransactWriteItem` protocol - Commit/Cancel phase

Write Transaction – Cancel

```
def processCancel(CancellationInput input):
    item = readItem(input)

    if item == NONE
     OR item.ongoingTransaction != input.transactionId:
        return CANCELLATION_FAILED

    item.ongoingTransaction = NONE

    #item was only created as part of this transaction
    if item was created during prepare:
        deleteItem(item)

    return SUCCESS
```

Listing 4: `TransactWriteItem` protocol - Commit/Cancel phase

Read Transactions – Transaction Coordinator



Read Transactions

```
def txRead(phase):
    item = readItem()

    if item == NONE:
        return ITEM_NOT_FOUND

    if item.ongoingTransaction != NONE:
        return ERROR

    if phase == 1:
        return LSN
    else:
        return item and LSN
```

Recap

Two kinds

Read, Write

Recap

- Two kinds – read and write

Write uses 2 PC

Recap

- Two kinds – read and write
- Write uses Two Phase commit

Read twice

Stretch

Transaction Cost

$$ and ms

Transaction Cost

- Dollars and Milliseconds

Network

Transaction Cost

- Dollars and Milliseconds
- Network latency

Ledger writes

Transaction Cost

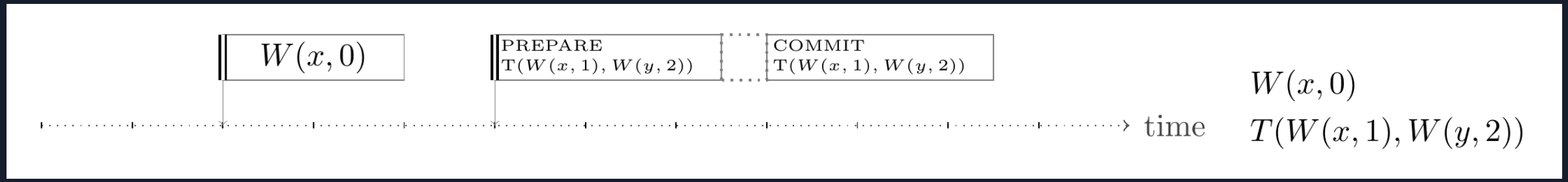
- Dollars and Milliseconds
- Network latency
- Ledger writes

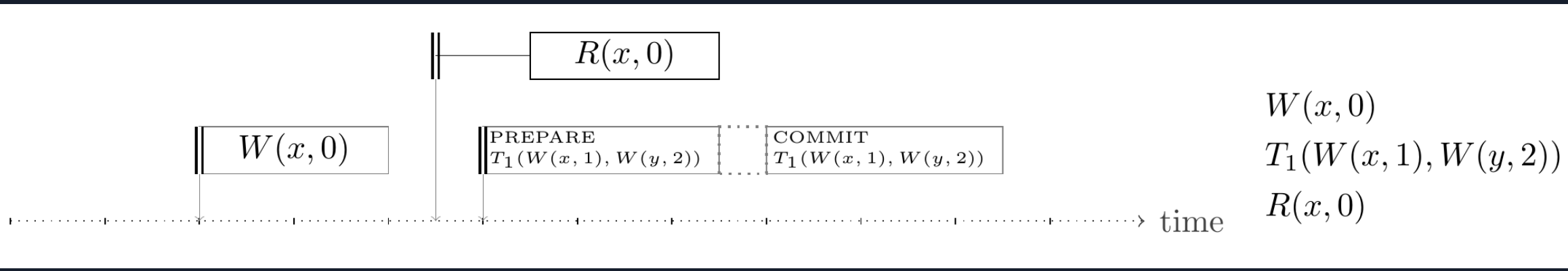
Data writes

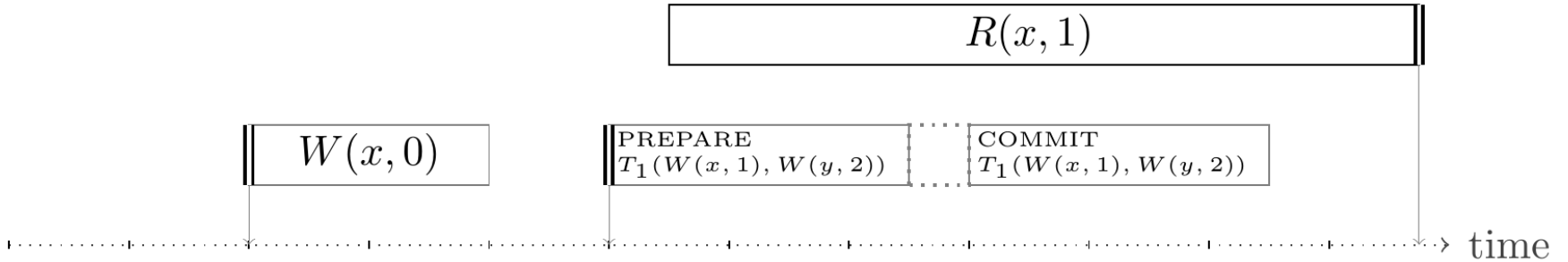
Transaction Cost

- Dollars and Milliseconds
- Network latency
- Ledger writes
- Data writes

2 PC vs Read Twice



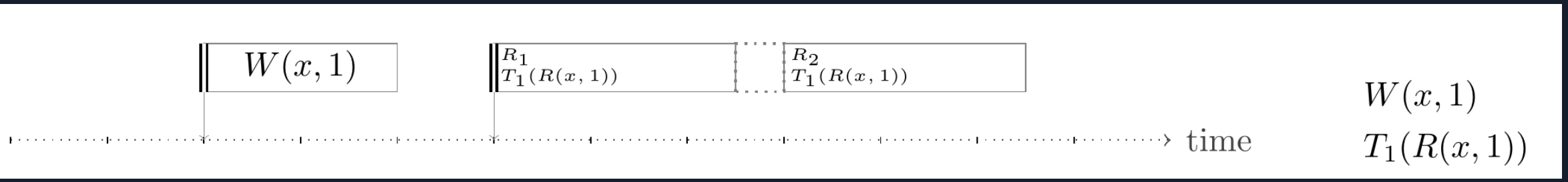


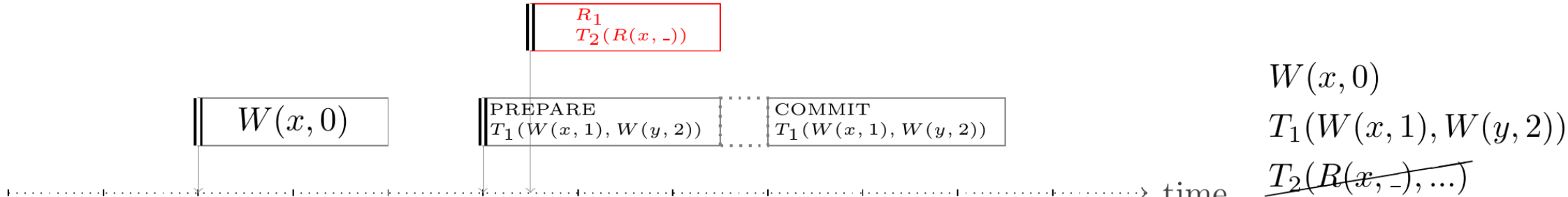


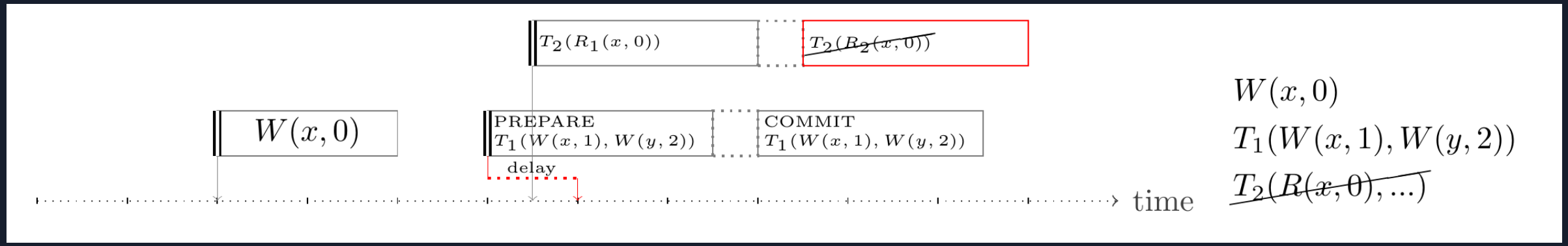
$W(x, 0)$

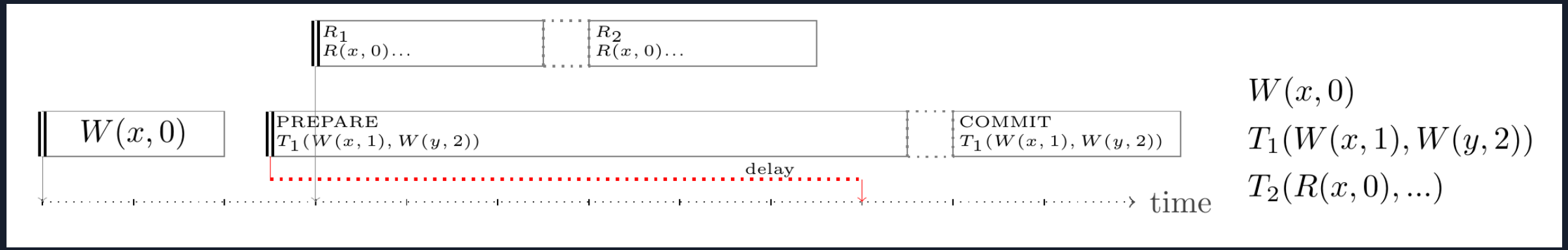
$T_1(W(x, 1), W(y, 2))$

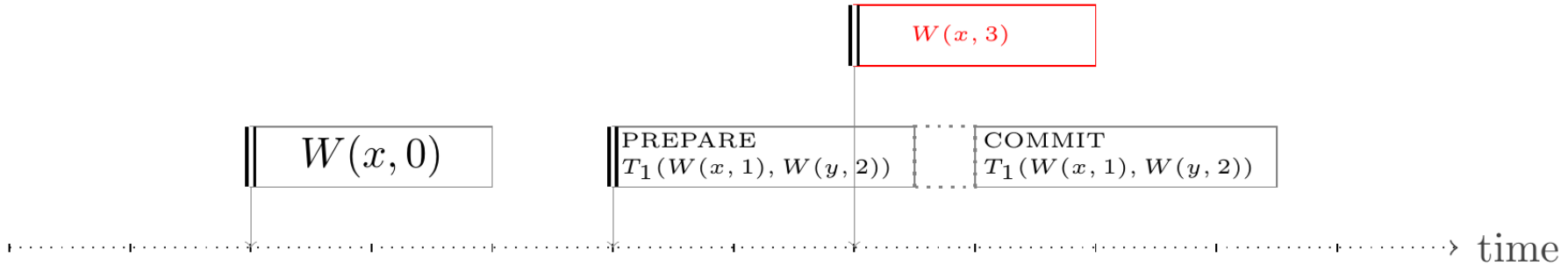
$R(x, 1)$









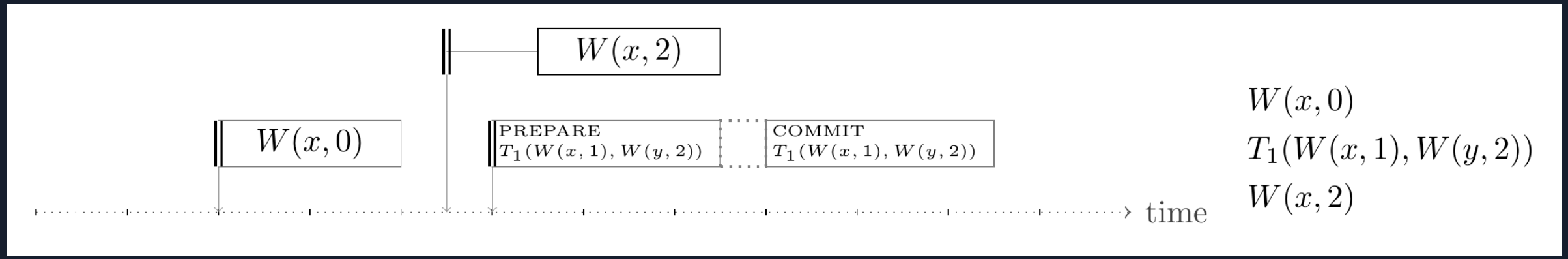


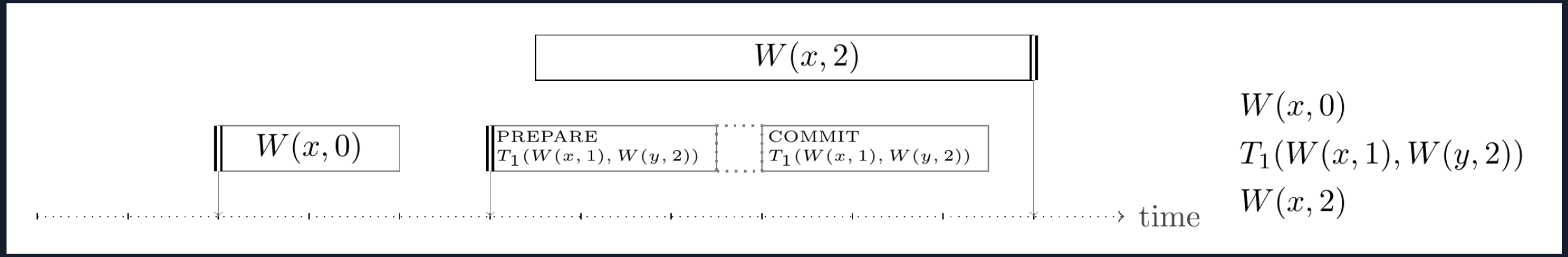
$W(x,0)$

$T_1(W(x,1),W(y,2))$

~~$W(x,3)$~~

```
def nonTxWrite(input):
    item = readItem(input)

    if item.ongoingTransaction != NONE:
        return ERROR

    ...
```





Question

- What isolation and consistency does DynamoDB Transactions provide?

Operations not in the Transactions API

Operation	Atomic	Eventually Consistent	Strongly Consistent
GetItem(), Query(), Scan()	Y[1]	Serializable[2], Monotonic Read	Serializable, Sequential
UpdateItem(), PutItem(), DeleteItem()	Y	Serializable, Sequential	
BatchGetItem()	N	Serializable[3], Monotonic Read	Individually Serializable, Sequential
BatchWriteItem()	N	Serializable, Sequential	

[1] Each invocation for Query() and Scan()
[2] Each invocation for Query() and Scan().
[3] Each individual Get in the batch.

Operations in the Transactions API

Serializable isolation

Sequential consistency

Question

- In a highly distributed system, is strict serializability possible?

Question

- In a highly distributed system, is there such a thing as universal time?

Recovery

$$N \times P(f) = N(f)$$

Recovery

- At scale, if it can fail, it will fail – often!

Just ask the TC!

Question

- When a transaction outcome is not known, what does an application do?

Transaction Idempotency

Unique token in request

Transaction Idempotency

- Unique token in request

Tokens stored for 10m

Transaction Idempotency

- Unique token in request
- Tokens stored for 10m

Retry with the same token

Transaction Idempotency

- Unique token in request
- Tokens stored for 10m
- Retry with the same token

Returns last status

Algorithm Correctness

Prove algorithms

Algorithm Correctness

- Prove algorithms

Formal Modeling

Algorithm Correctness

- Prove algorithms
- Formal Modeling

Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers – Leslie Lamport

Operations through the Transactions API

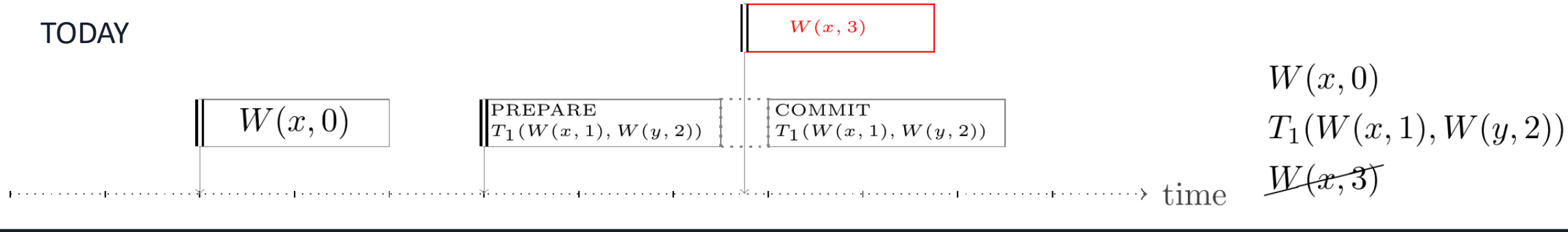
Serializable isolation

Sequential consistency

Can we do better?

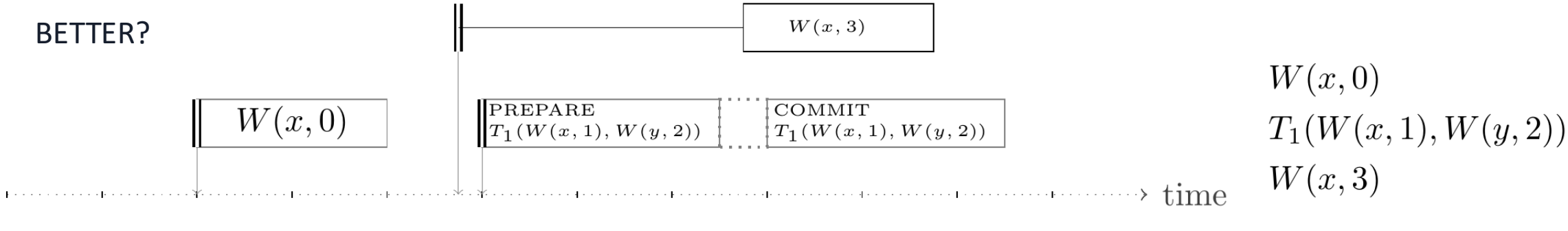
Optimizations

TODAY



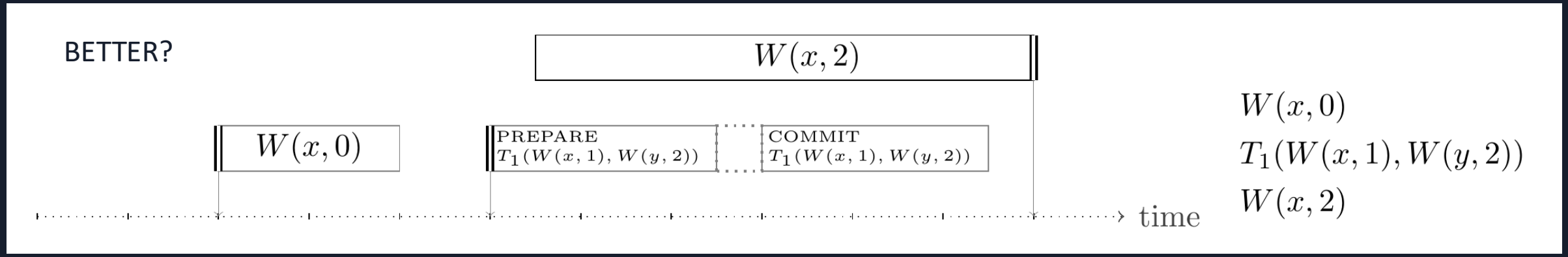
$W(x,0)$
$T_1(W(x,1), W(y,2))$
~~$W(x,3)$~~

BETTER?



$W(x,0)$
$T_1(W(x,1), W(y,2))$
$W(x,3)$

Optimizations



Performance Characteristics

Item size

Performance Characteristics

- Item size

Throughput

Performance Characteristics

- Item size
- Throughput

Workload mix

Performance Characteristics

- Item size
- Throughput
- Workload mix

Transaction size

Performance Characteristics

- Item size
- Throughput
- Workload mix
- Transaction size

Not through Txn API

Performance Characteristics

- Item size
- Throughput
- Workload mix
- Transaction size
- Operations in the Transactions API and other operations

Contention

Conclusion

Predictable Latency

Conclusion

- Predictable Latency

At any scale

References

- "Database System Concepts" by Silberschatz, Korth, and Sudarshan
- "Transaction Processing: Concepts and Techniques" by Jim Gray and Andreas Reuter
- Consistency Models – Jepsen (Kyle Kingsbury)
- Highly Available Transactions: Virtues and Limitations; Bailis et al., VLDB 2013
- Information technology - Database languages SQL — Part 2: Foundation (SQL/Foundation), Part 2, 2023