

# GORDA: An Open Architecture for Scalable Database Clustering

Rui Oliveira

Computer Science and Technology Center  
University of Minho

IST FP6 GORDA Project



# Project and Goals



# Project and Goals

- In a nutshell, GORDA intended to foster database replication as a means to address the challenges of trust, integration, performance, and cost in current database systems.

# Project and Goals

- In a nutshell, GORDA intended to foster database replication as a means to address the challenges of trust, integration, performance, and cost in current database systems.
- This is to be achieved with an open architecture and interfaces, innovative replication protocols, management tools and an interim middleware-based version (wrapper) to ease the migration path.

# Project and Goals

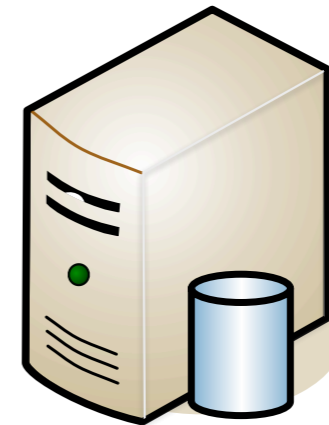
- In a nutshell, GORDA intended to foster database replication as a means to address the challenges of trust, integration, performance, and cost in current database systems.
- This is to be achieved with an open architecture and interfaces, innovative replication protocols, management tools and an interim middleware-based version (wrapper) to ease the migration path.
- In particular, GORDA addressed strictly consistent replication, 1-copy-equivalence consistency and group communication based protocols.

# Database Replication



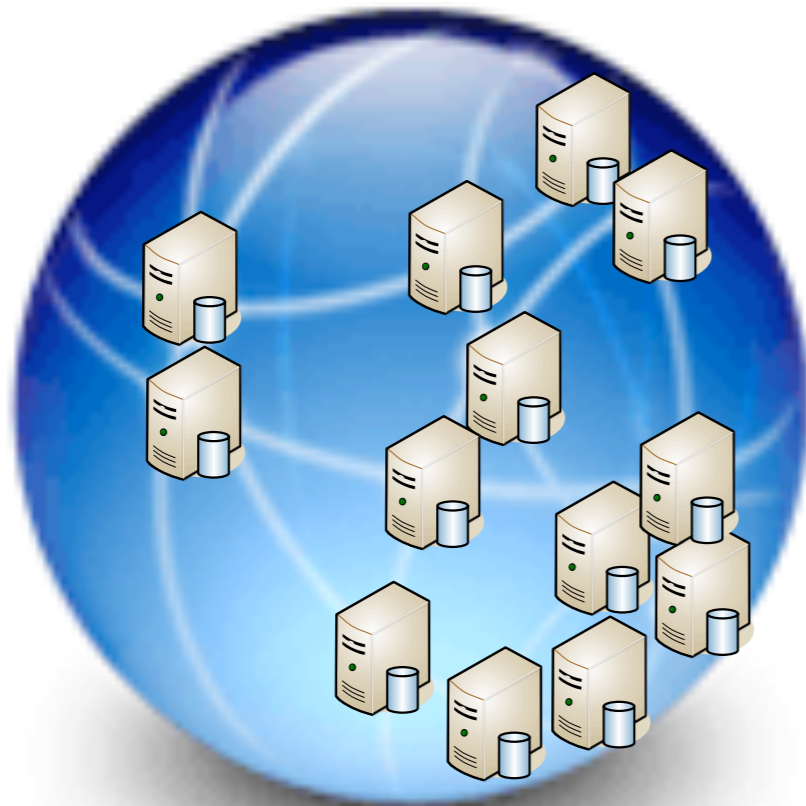
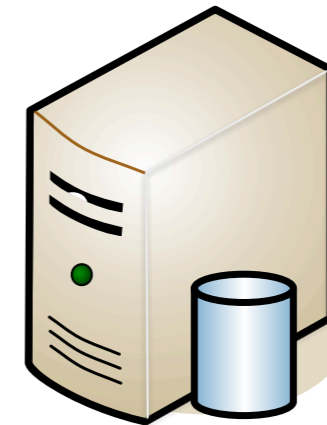
# Database Replication

- Database servers
  - Transactional requests
  - Multi-interaction requests
  - Highly concurrent servers



# Database Replication

- Database servers
  - Transactional requests
  - Multi-interaction requests
  - Highly concurrent servers



- Replication goals
  - Increased performance
    - Load-balancing
    - Read/Write quorums
    - Consistency criteria
  - Fault tolerance
    - Zero downtime
    - Fault model
    - Consistency criteria

# Strictly Consistent Database Replication



# Strictly Consistent Database Replication

- Strictly consistent database replication to provide transparent distribution and fault tolerance
  - Strictly consistent replicas on transaction boundaries
  - No reconciliation
  - Automatic fail-over without lost updates

# Strictly Consistent Database Replication

- Strictly consistent database replication to provide transparent distribution and fault tolerance
  - Strictly consistent replicas on transaction boundaries
  - No reconciliation
  - Automatic fail-over without lost updates
- Focus on group communication based techniques instead of traditional distributed locking and atomic commitment protocols

# Strictly Consistent Database Replication

- Strictly consistent database replication to provide transparent distribution and fault tolerance
  - Strictly consistent replicas on transaction boundaries
  - No reconciliation
  - Automatic fail-over without lost updates
- Focus on group communication based techniques instead of traditional distributed locking and atomic commitment protocols
- Multi-master, update-anywhere protocols

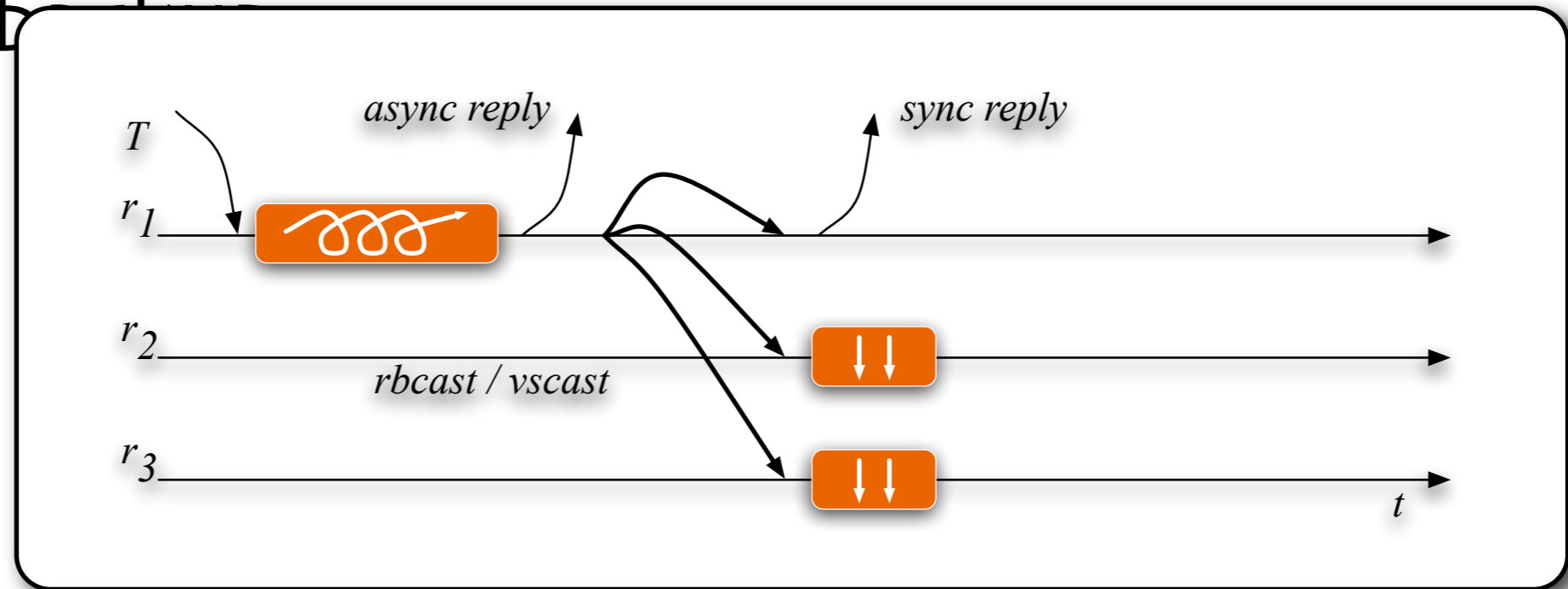
# Generic Replication Strategies

# Generic Replication Strategies

- Primary-backup

# Generic Replication Strategies

## Primary-backup



# Generic Replication Strategies

- Primary-backup
  - Asynch PB is standard replication in most DBMS
  - Asynch vs. synch update propagation
  - Adequate to handle non-deterministic servers
  - Extensible to multi-master (partitioning, reconciliation)

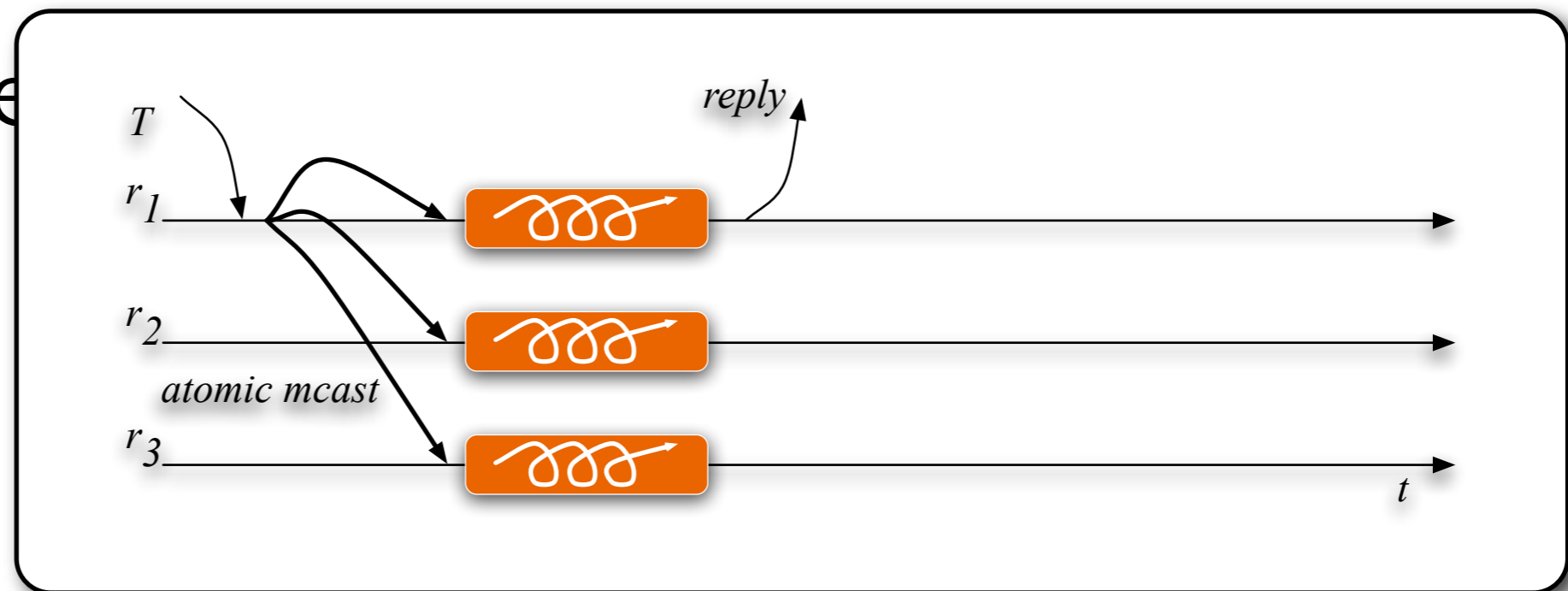
# Generic Replication Strategies

- Primary-backup
  - Asynch PB is standard replication in most DBMS
  - Asynch vs. synch update propagation
  - Adequate to handle non-deterministic servers
  - Extensible to multi-master (partitioning, reconciliation)
- Replicated State-machine

# Generic Replication Strategies

- Primary-backup
  - Asynch PB is standard replication in most DBMS
  - Asynch vs. synch update propagation
  - Adequate to handle non-deterministic servers
  - Extensible to multi-master (partitioning, reconciliation)

- Replicate



# Generic Replication Strategies

- Primary-backup
  - Asynch PB is standard replication in most DBMS
  - Asynch vs. synch update propagation
  - Adequate to handle non-deterministic servers
  - Extensible to multi-master (partitioning, reconciliation)
- Replicated State-machine
  - Simplicity and failure transparency
  - Requires deterministic processing (SQL, scheduling)

# Database Specific Replication Protocols



# Database Specific Replication Protocols

- Conflict-class based protocols
  - A priori classification of transactions in conflict classes
  - Conservative execution of transactions
  - Passive replication

# Database Specific Replication Protocols

- Conflict-class based protocols
  - A priori classification of transactions in conflict classes
  - Conservative execution of transactions
  - Passive replication
- Certification-based protocols
  - Optimistic execution
  - Passive replication
  - Certification involves transactions' read and write sets

# Conservative Execution



# Conservative Execution

- Transactions are classified in conflict classes

# Conservative Execution

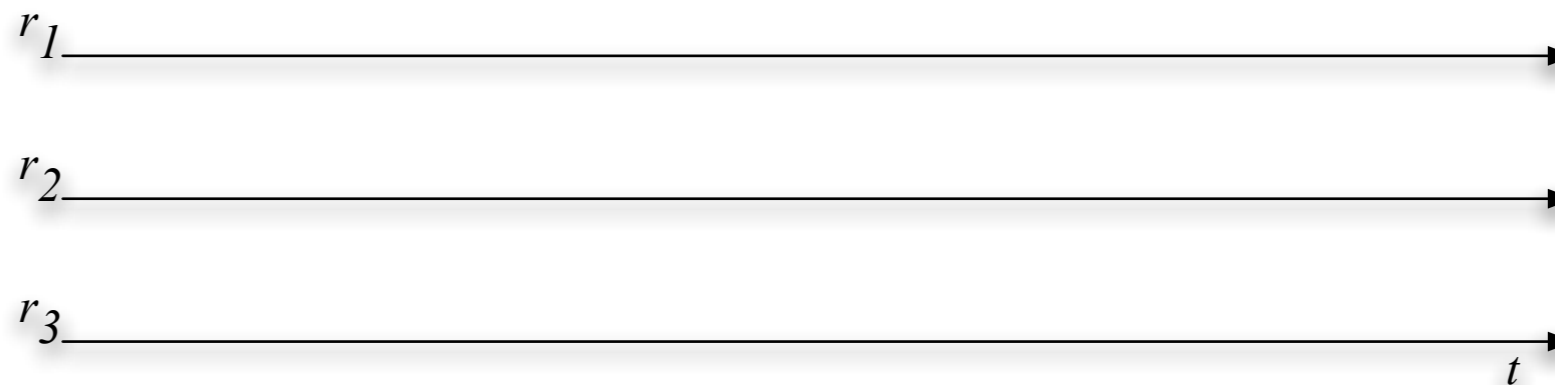
- Transactions are classified in conflict classes
- Transactions are ordered before their execution

# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed

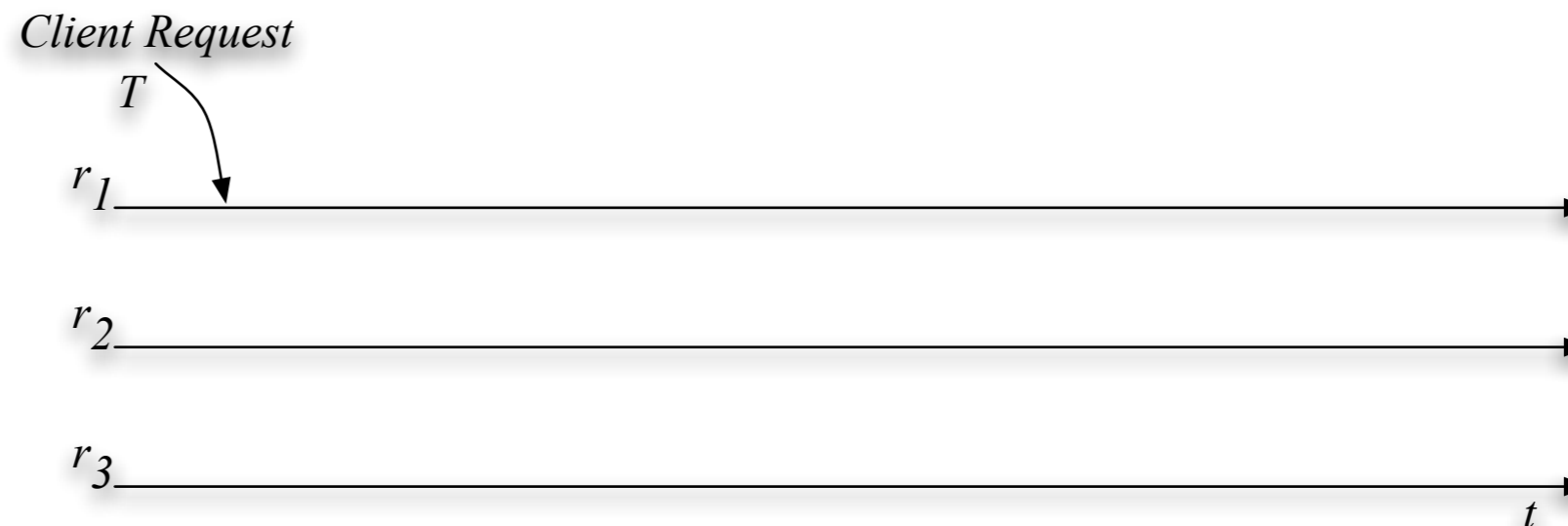
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



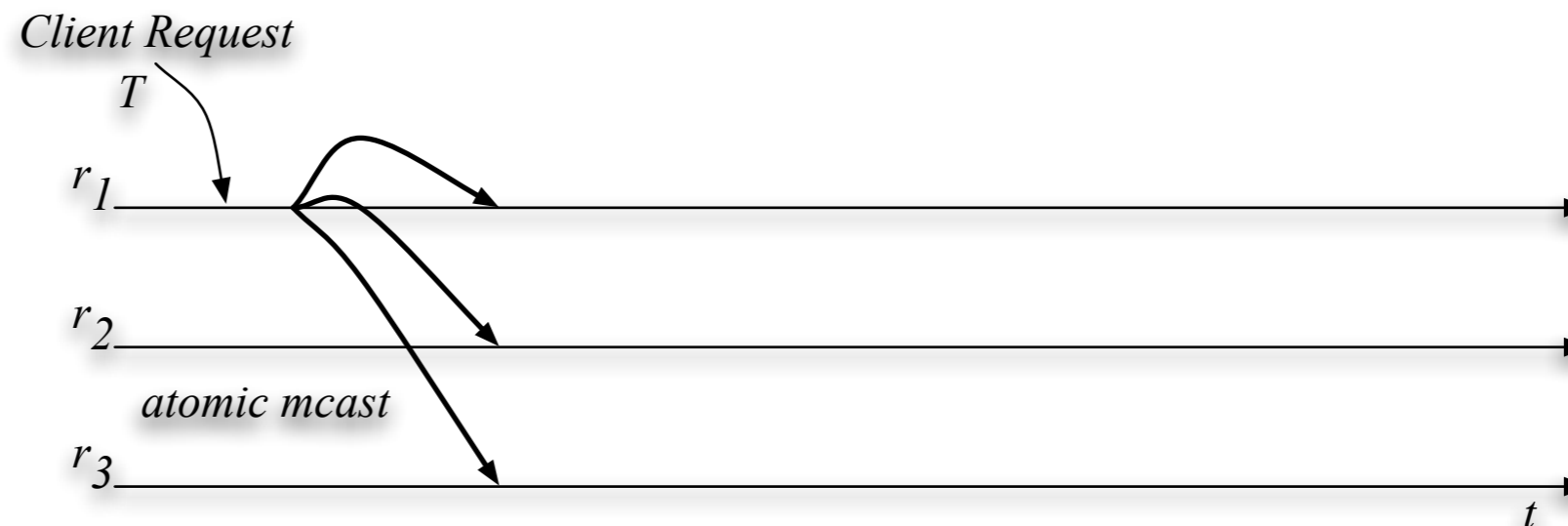
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



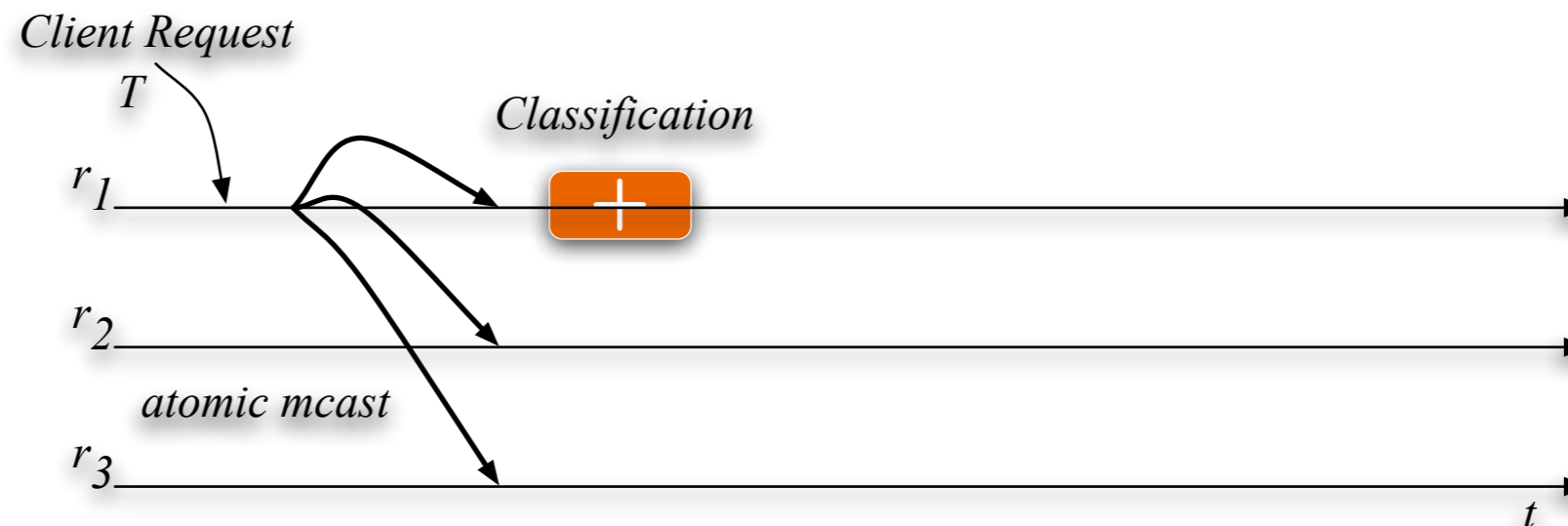
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



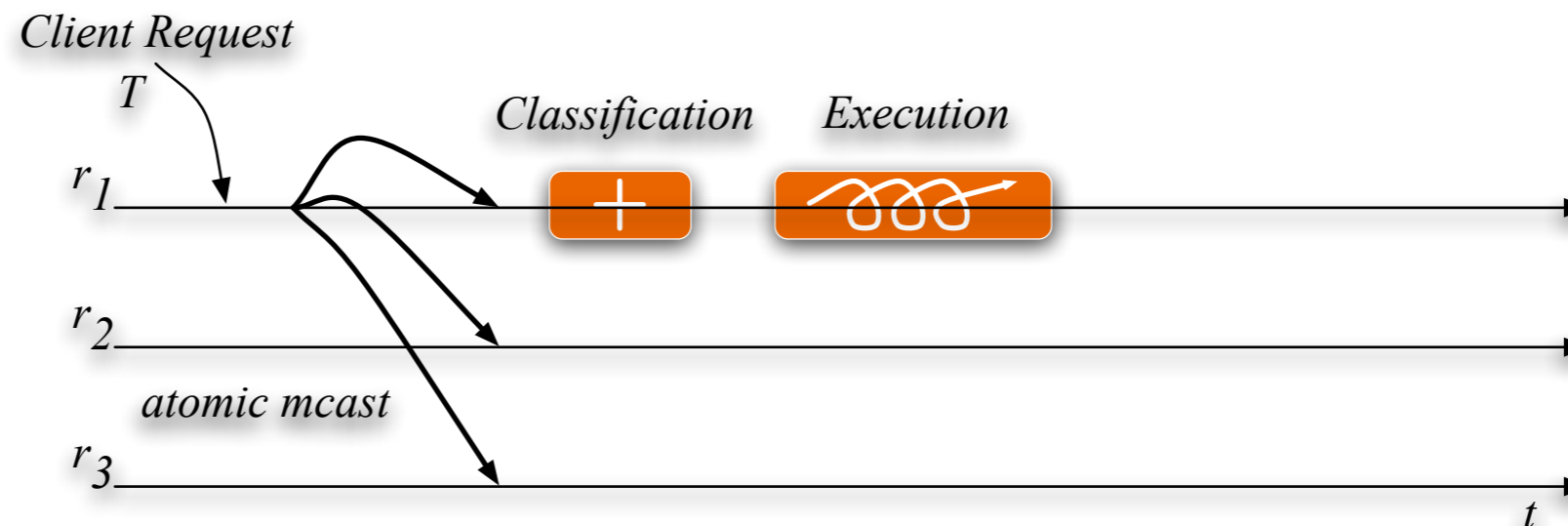
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



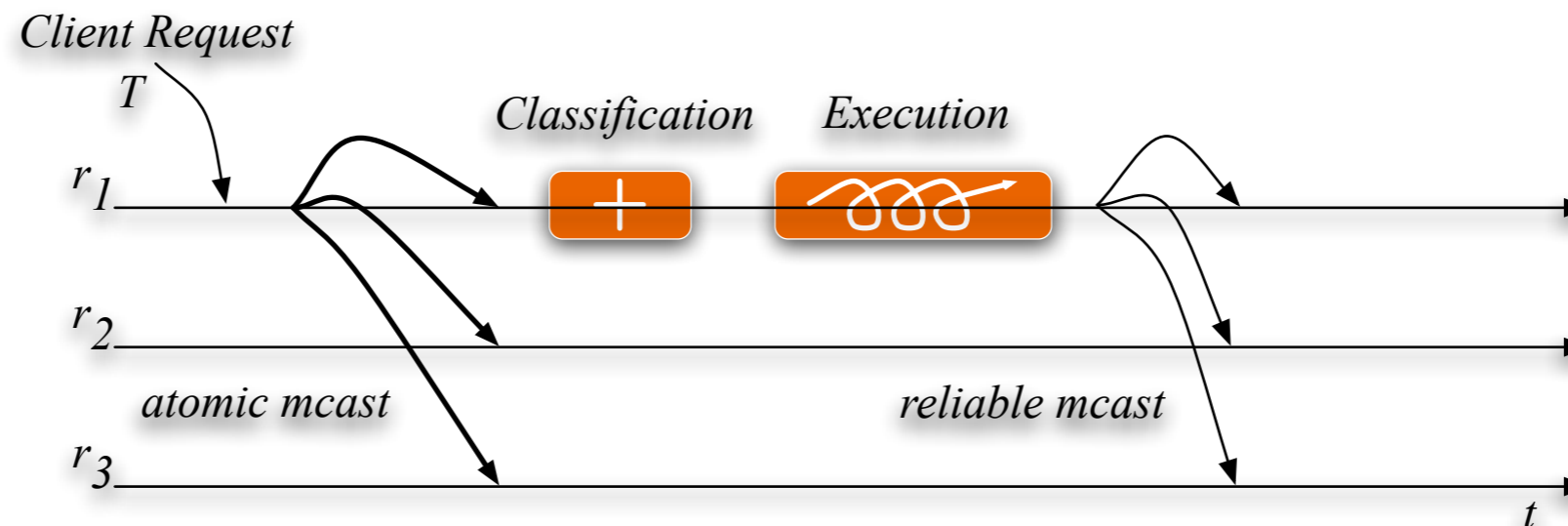
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



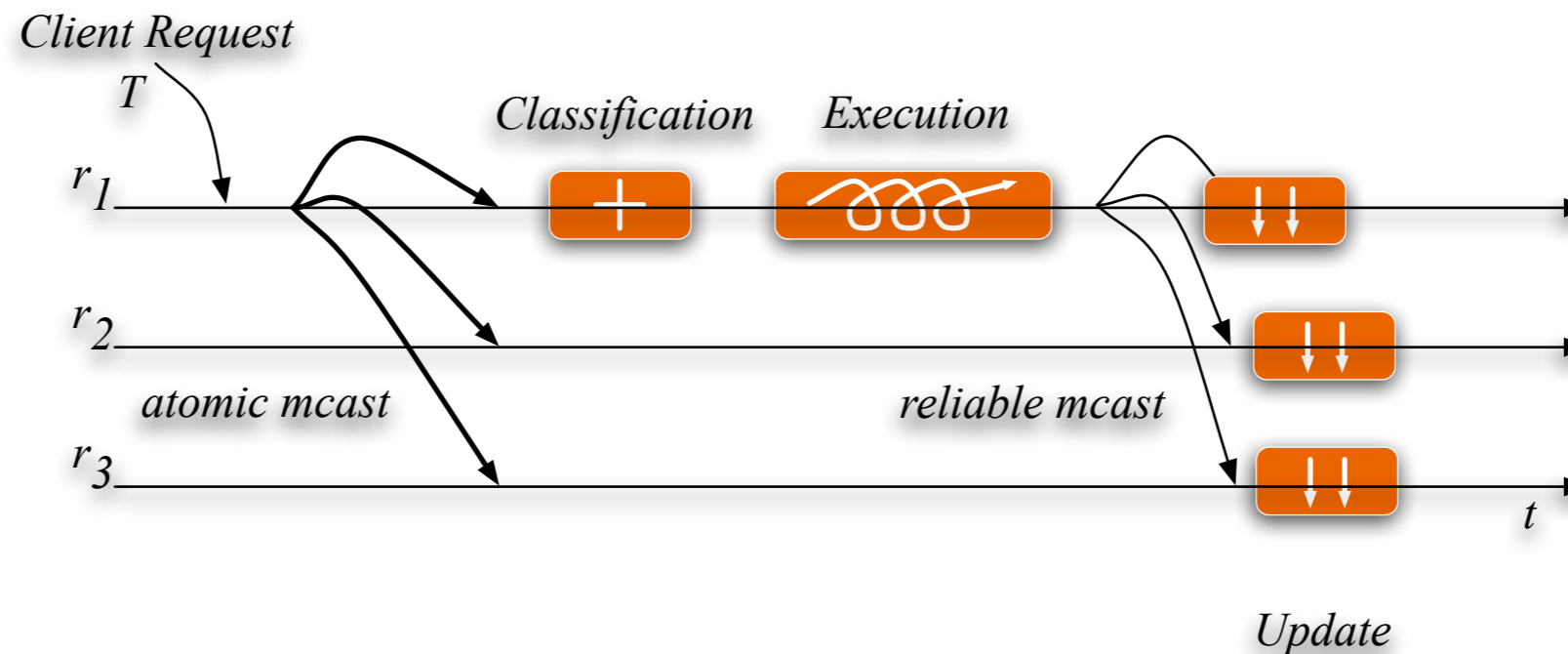
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



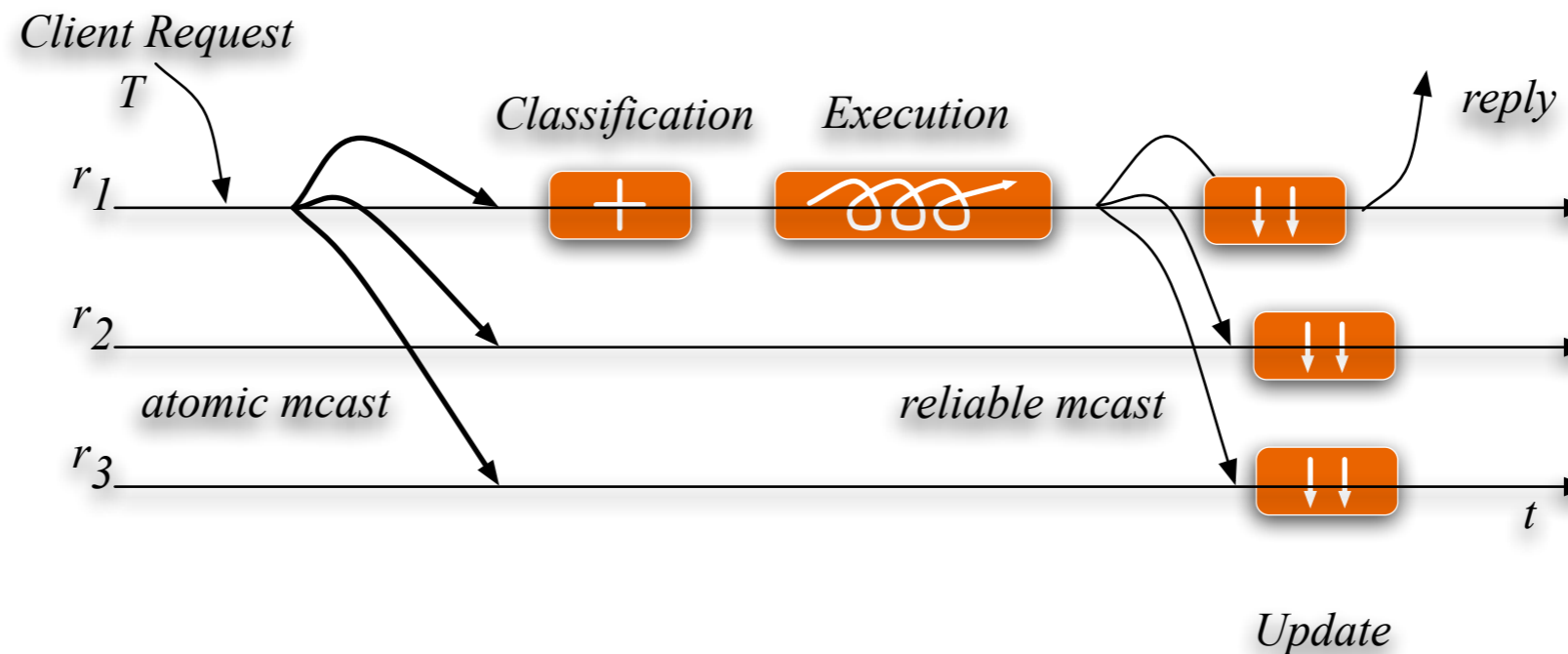
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



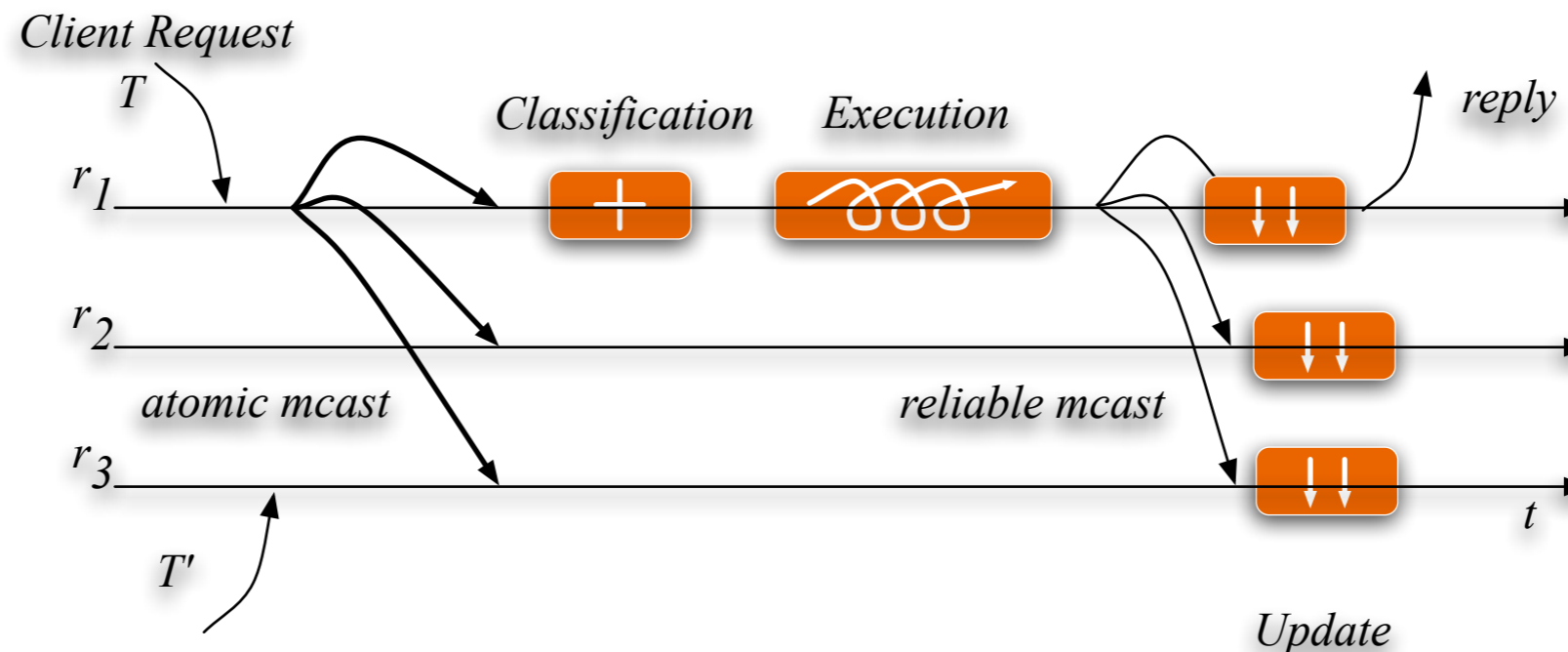
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



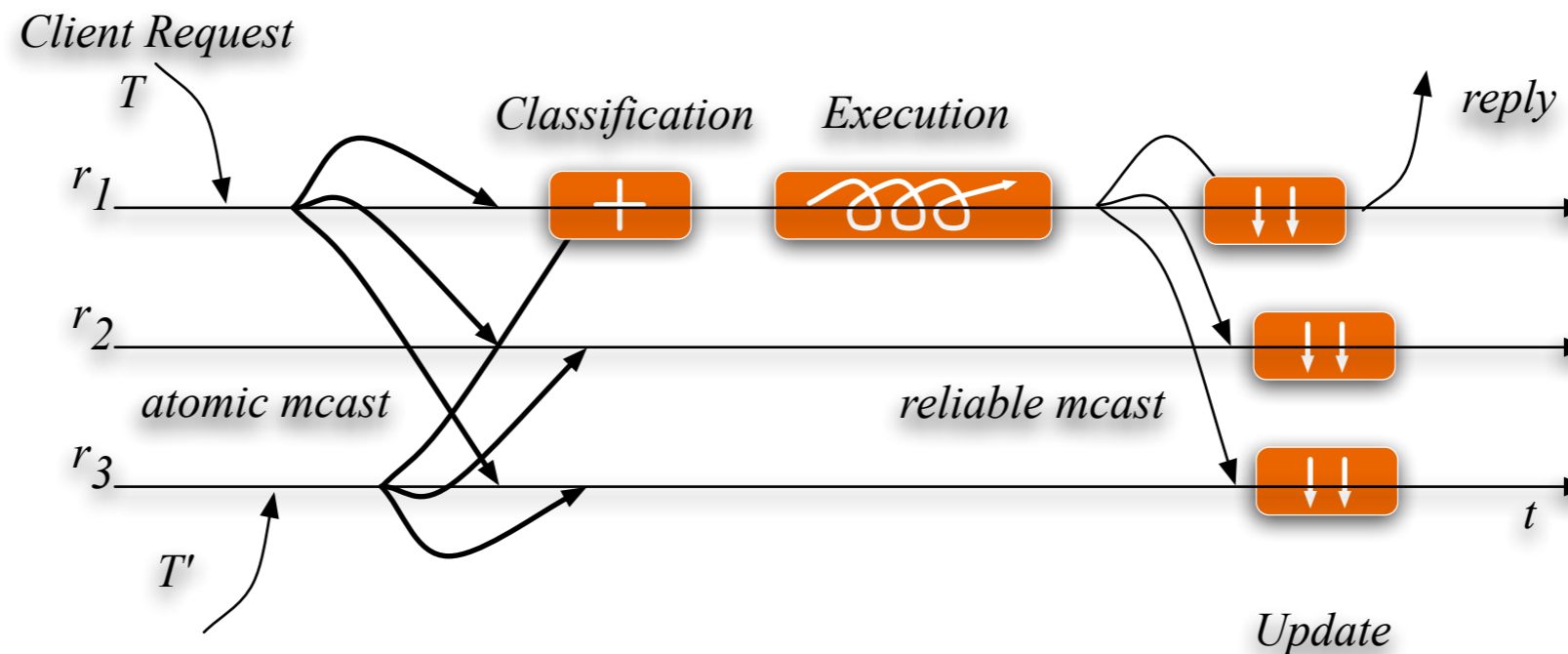
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



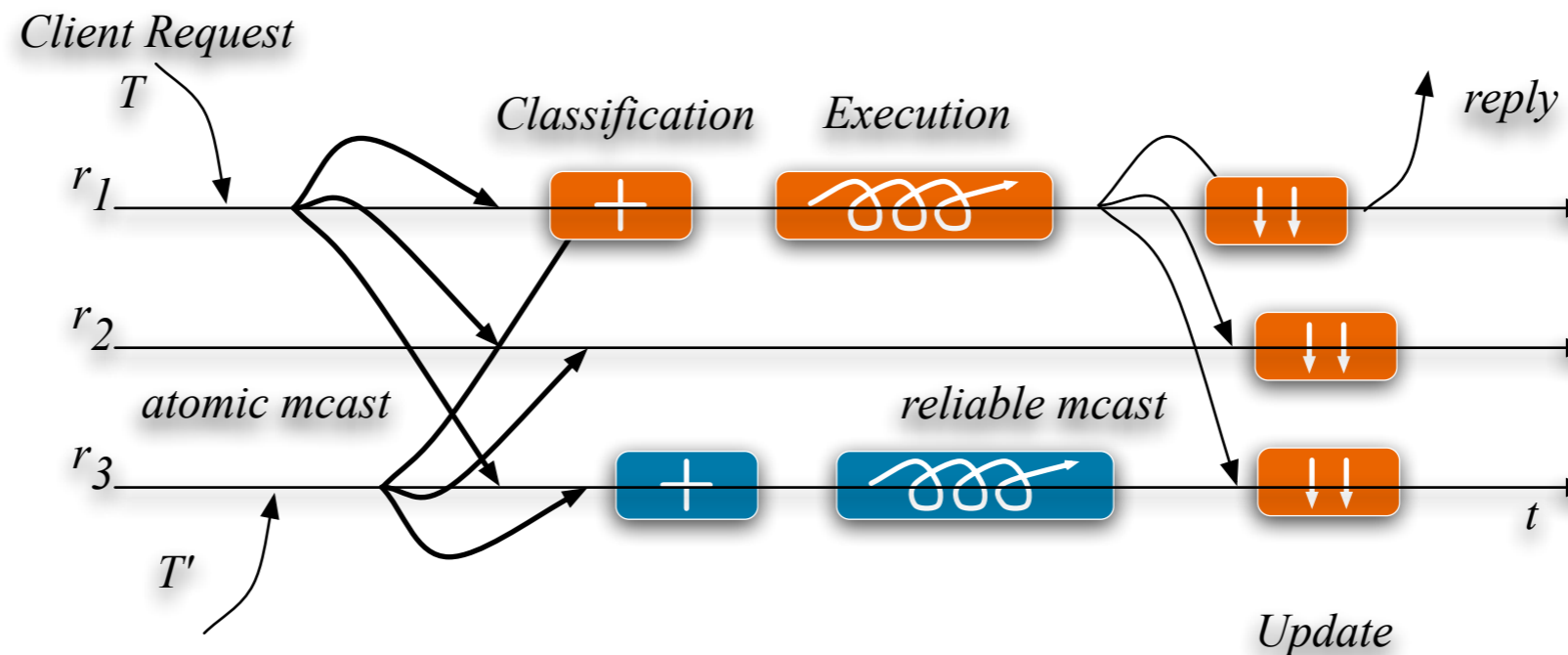
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



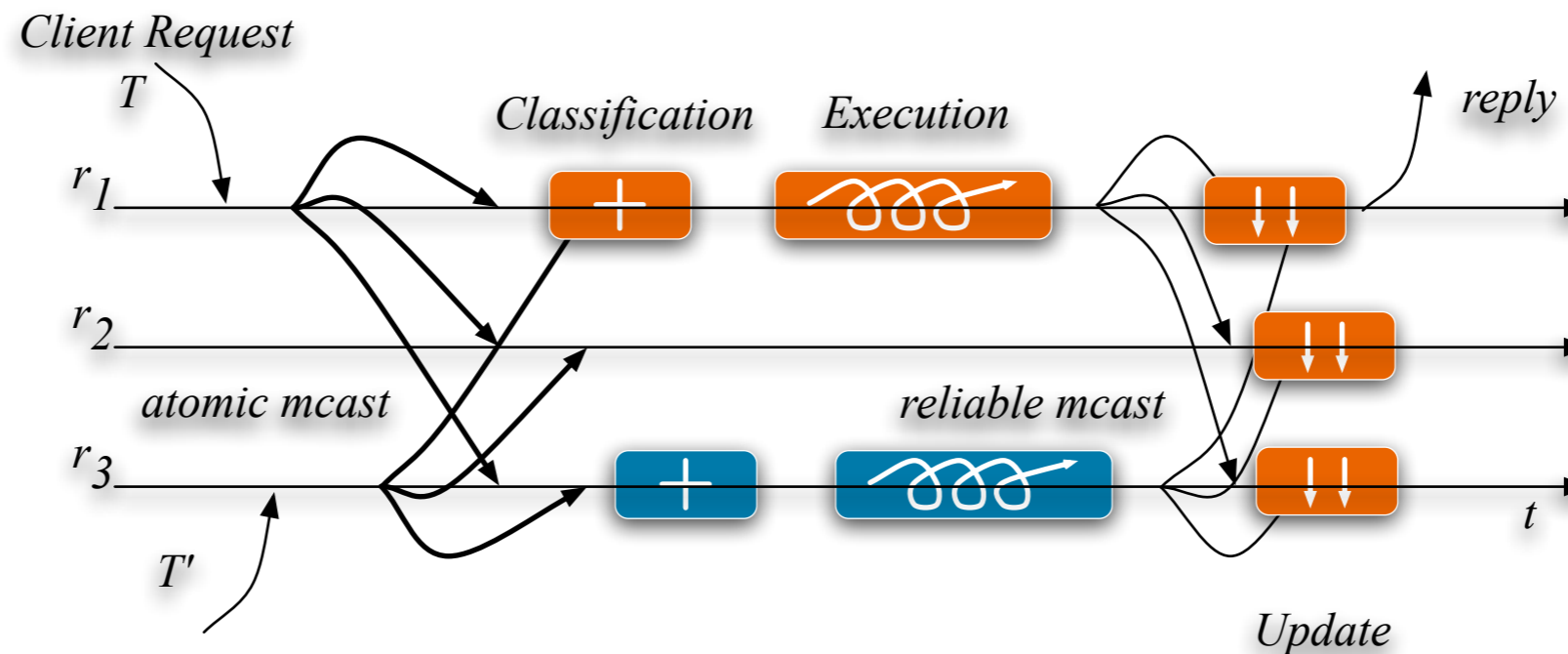
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



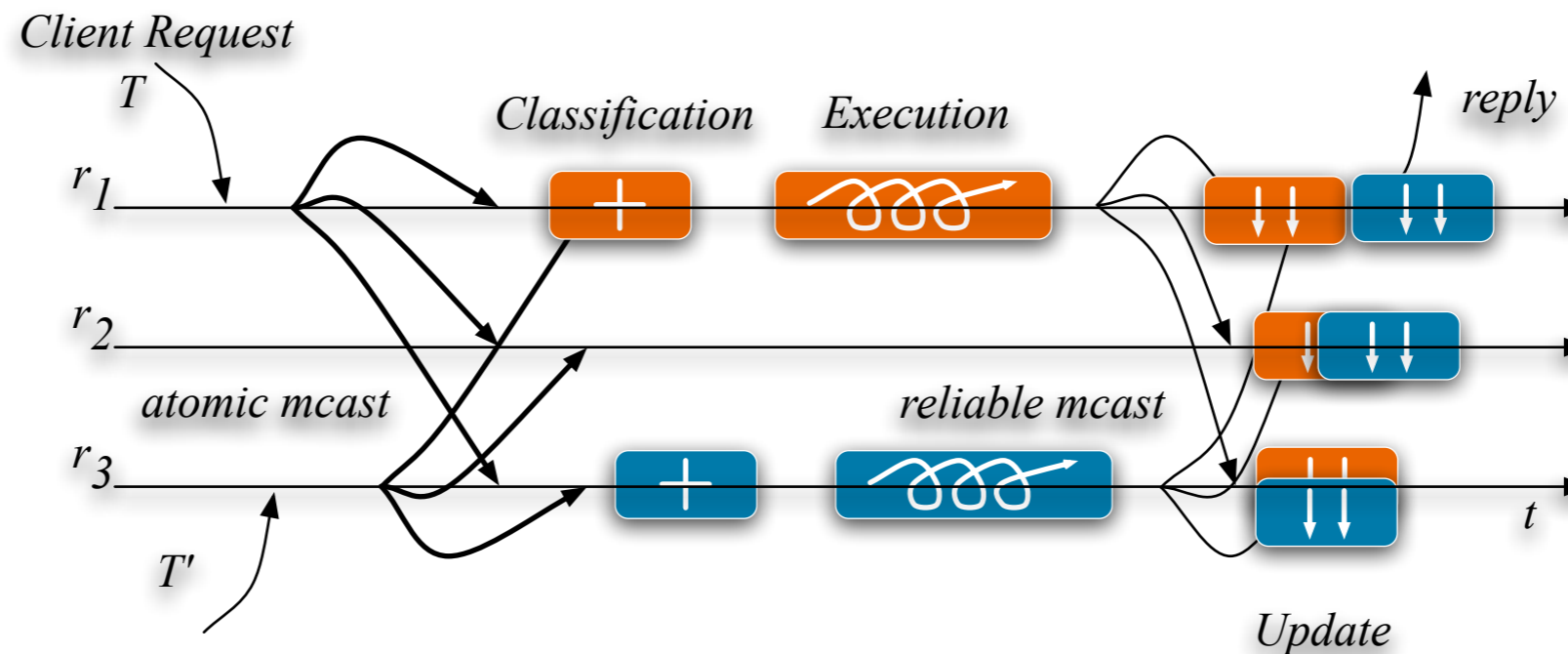
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



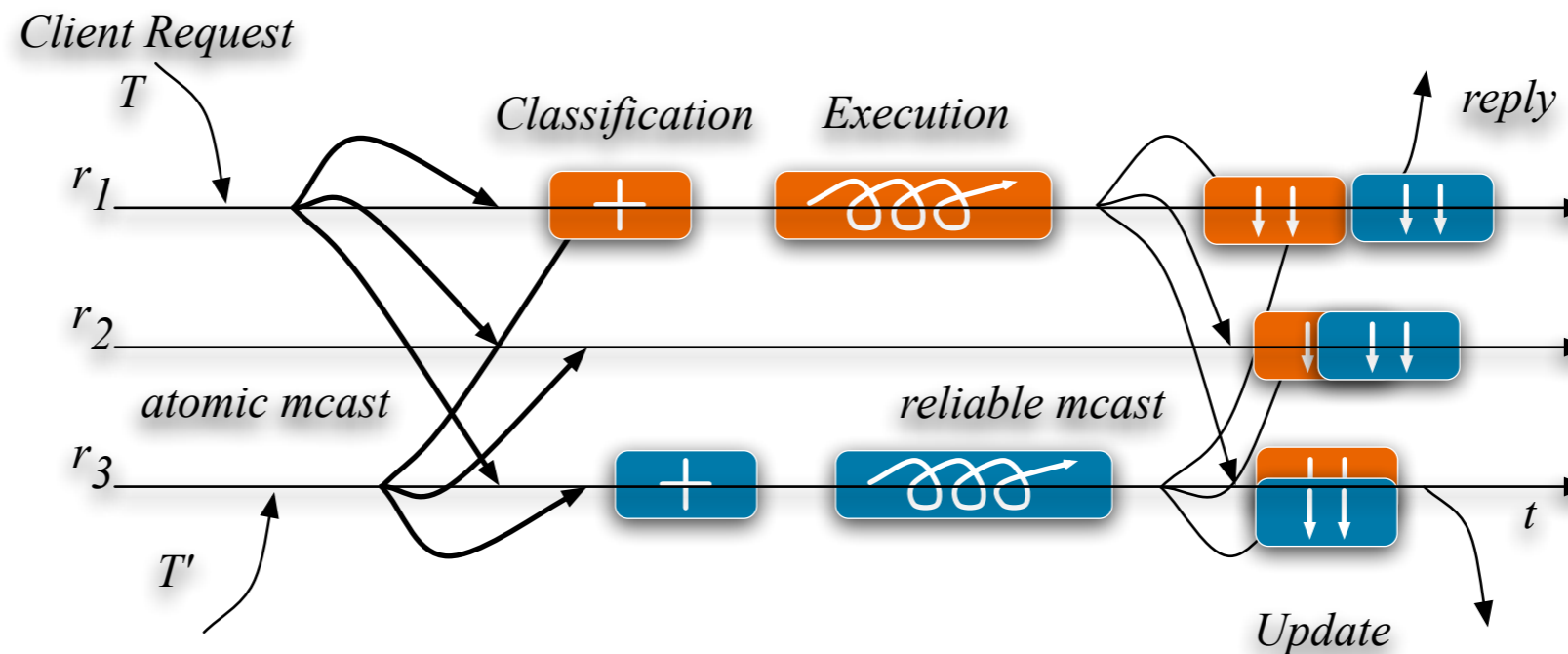
# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



# Conservative Execution

- Transactions are classified in conflict classes
- Transactions are ordered before their execution
- Conflicting transactions are sequentially executed



# Optimistic Execution



# Optimistic Execution

- Transactions are ordered after their execution

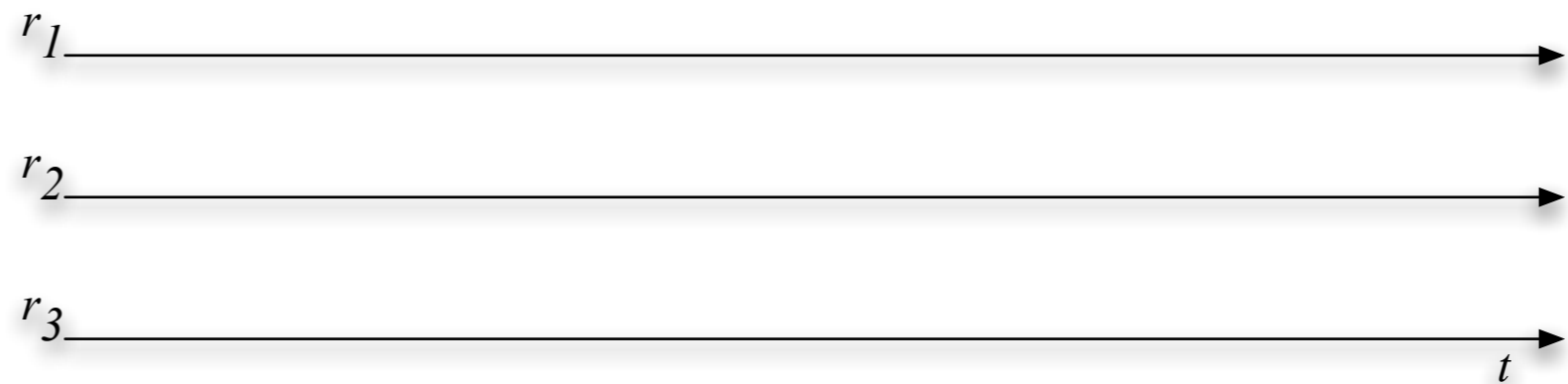
# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

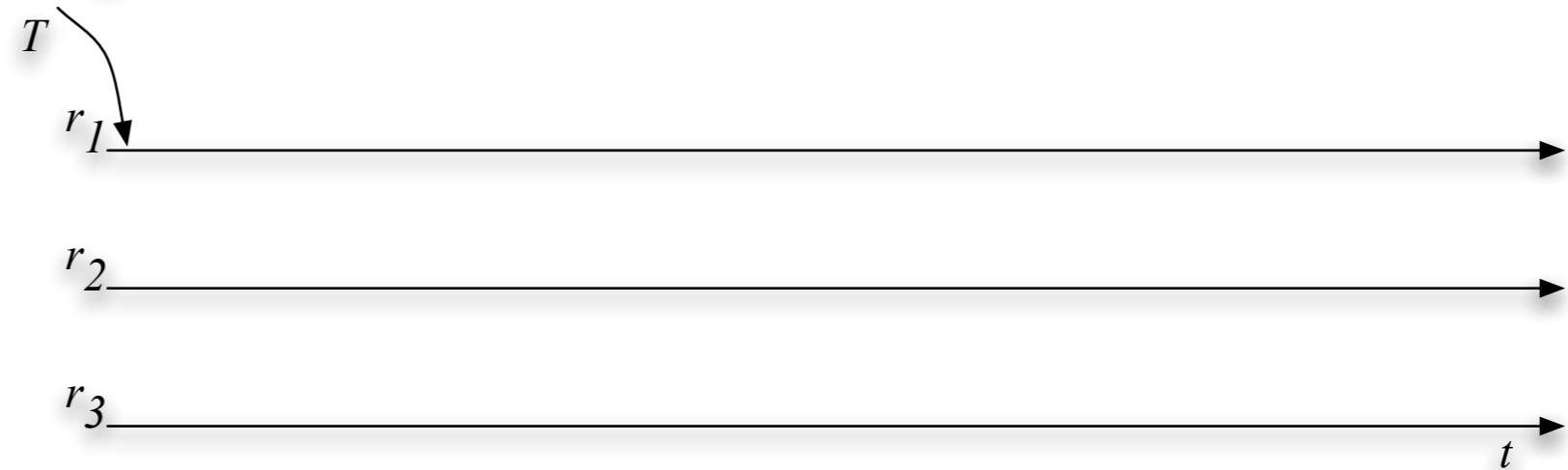
Postgres-R



# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

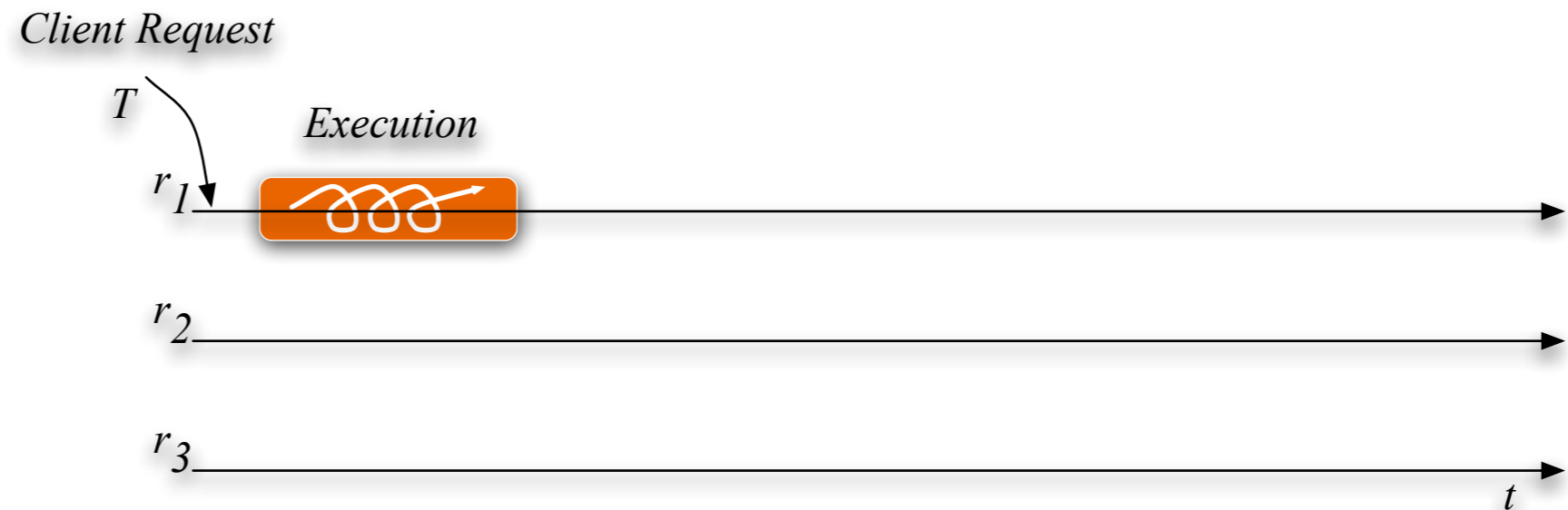
*Client Request*



Postgres-R

# Optimistic Execution

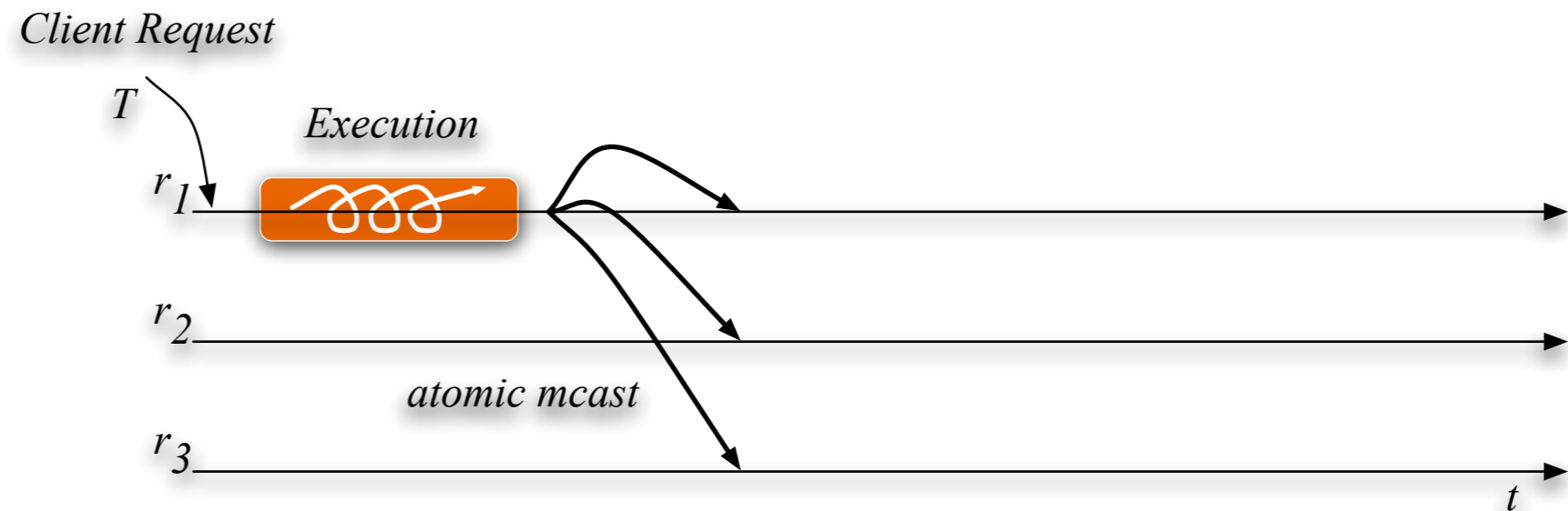
- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently



Postgres-R

# Optimistic Execution

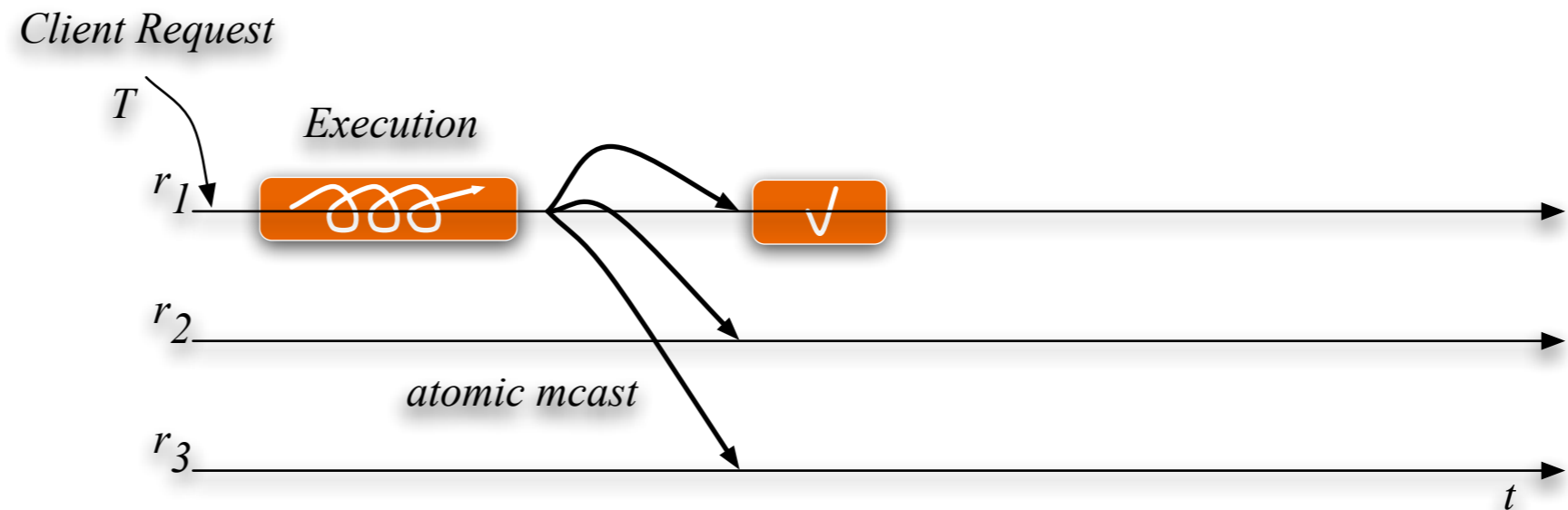
- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently



## Postgres-R

# Optimistic Execution

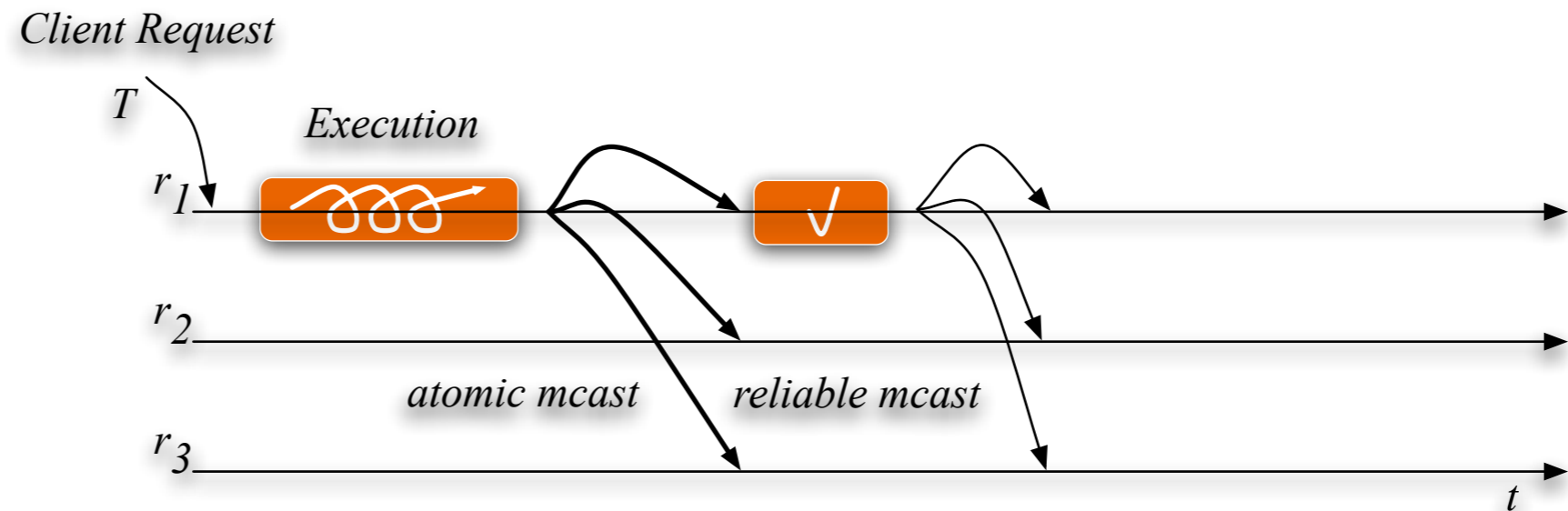
- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently



Postgres-R

# Optimistic Execution

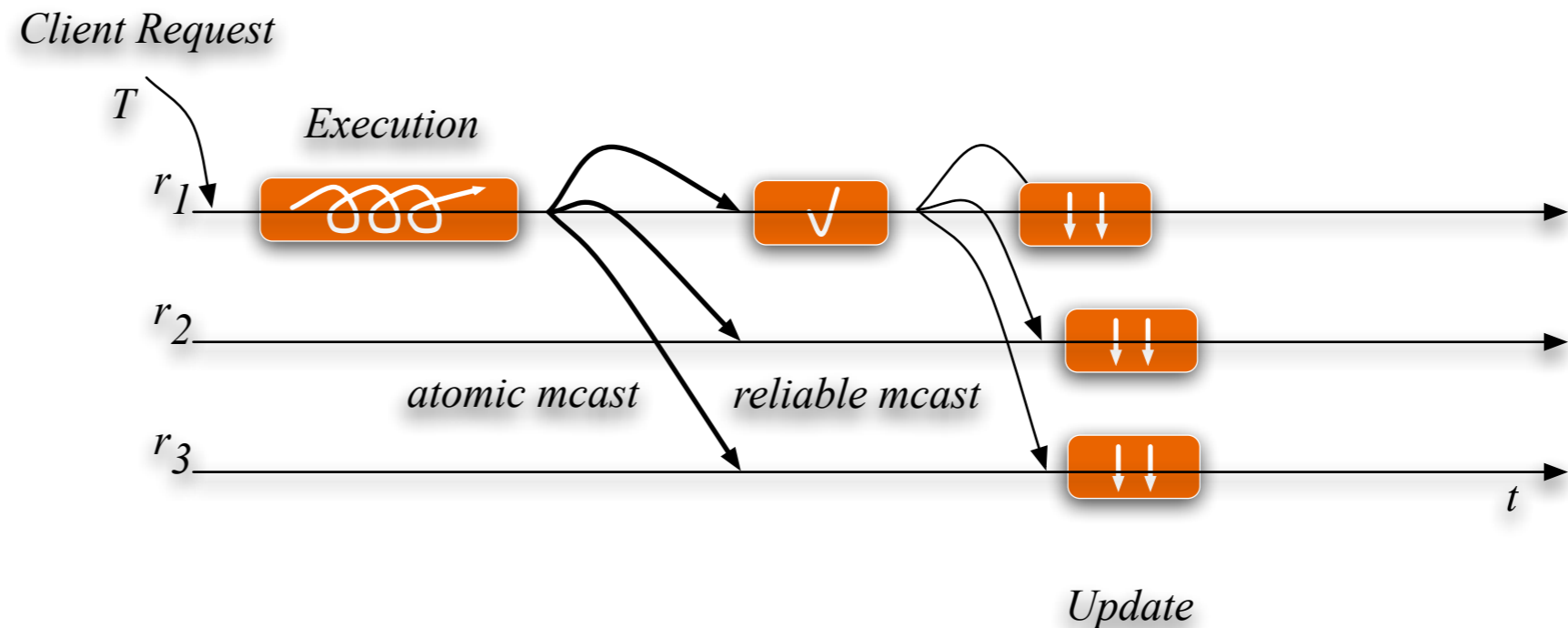
- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently



Postgres-R

# Optimistic Execution

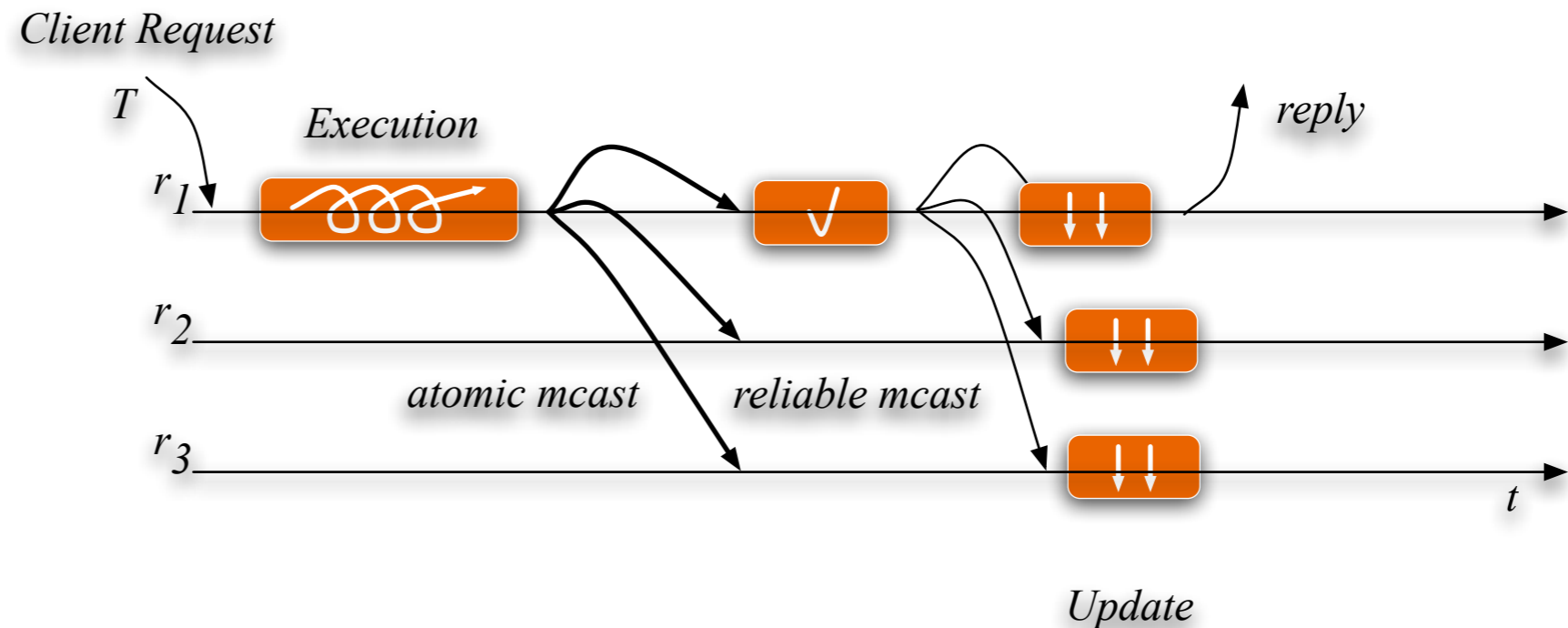
- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently



Postgres-R

# Optimistic Execution

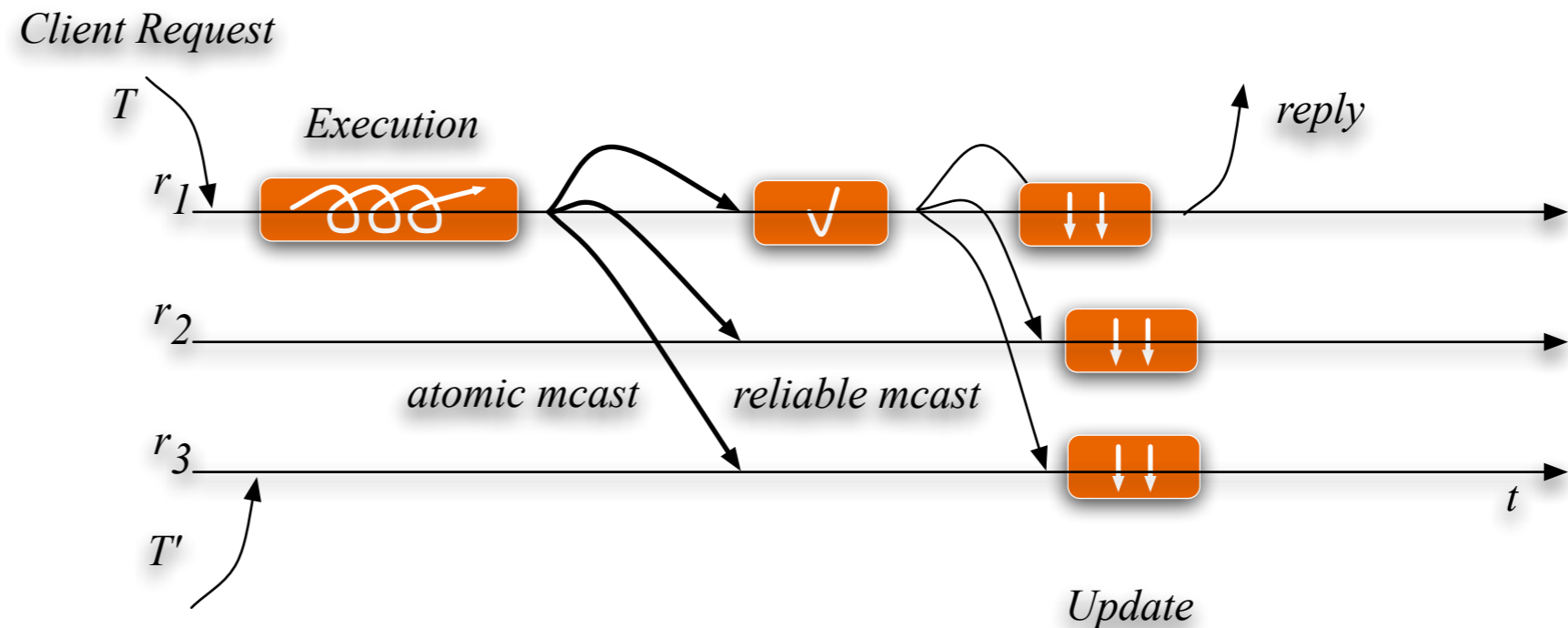
- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently



Postgres-R

# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

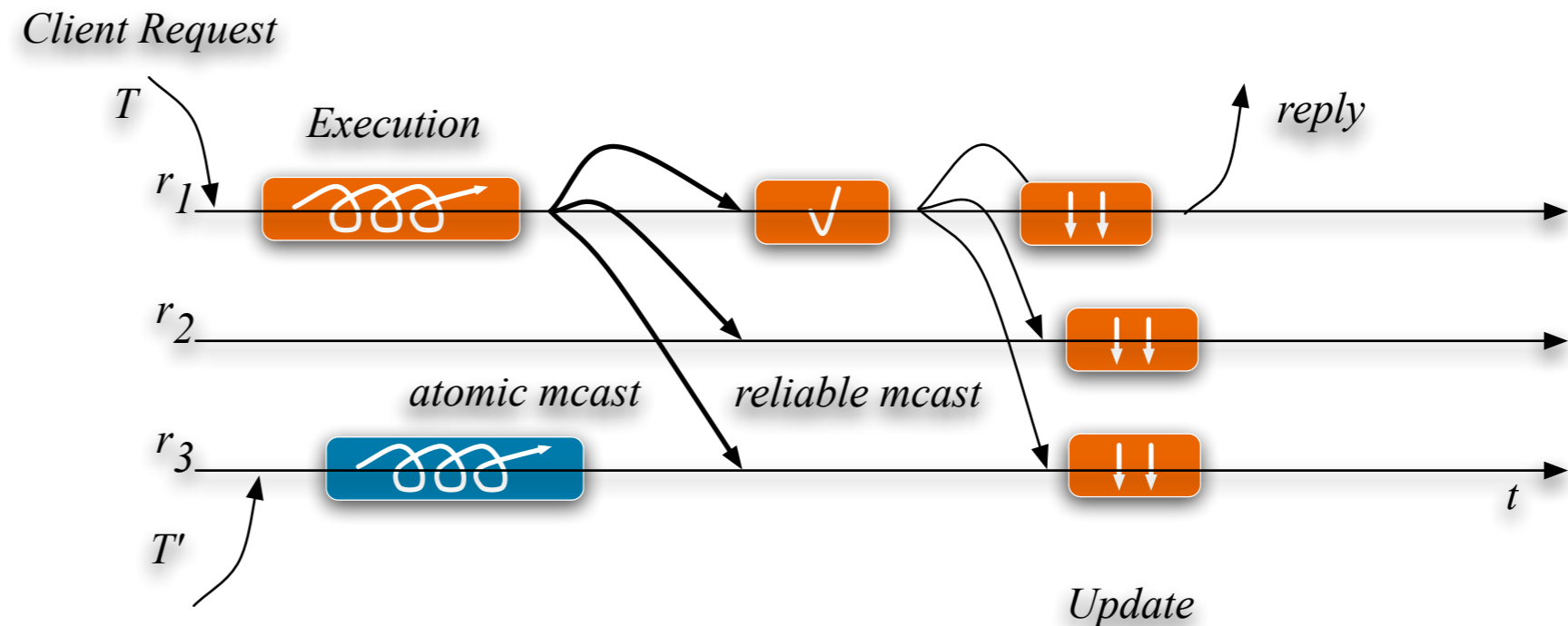


Postgres-R

# Optimistic Execution

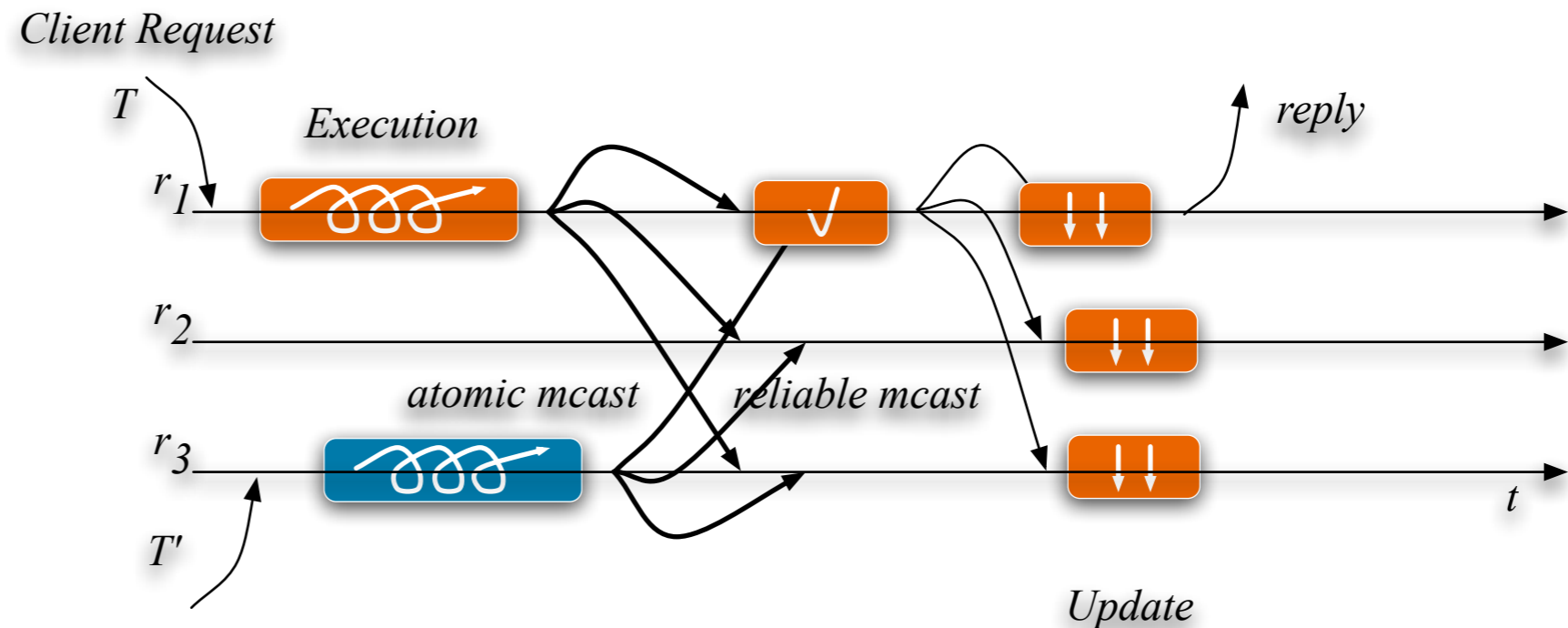
- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R



# Optimistic Execution

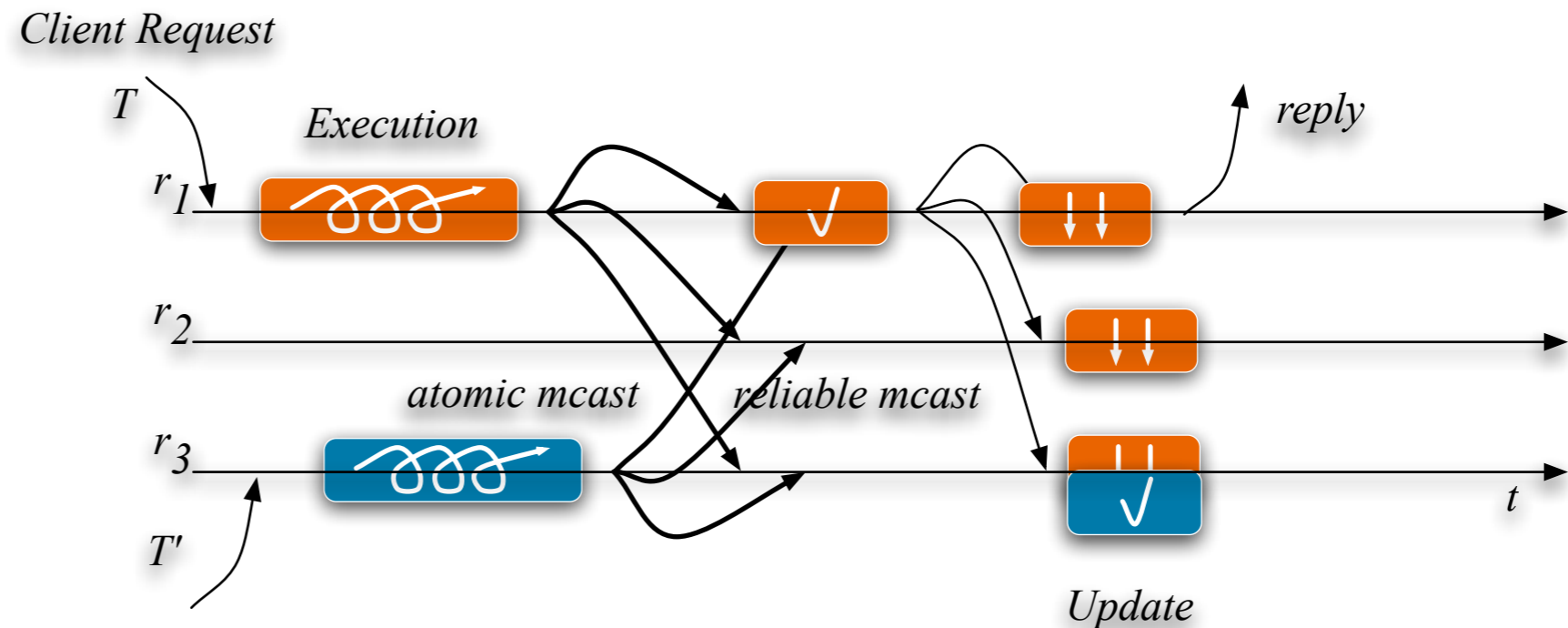
- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently



Postgres-R

# Optimistic Execution

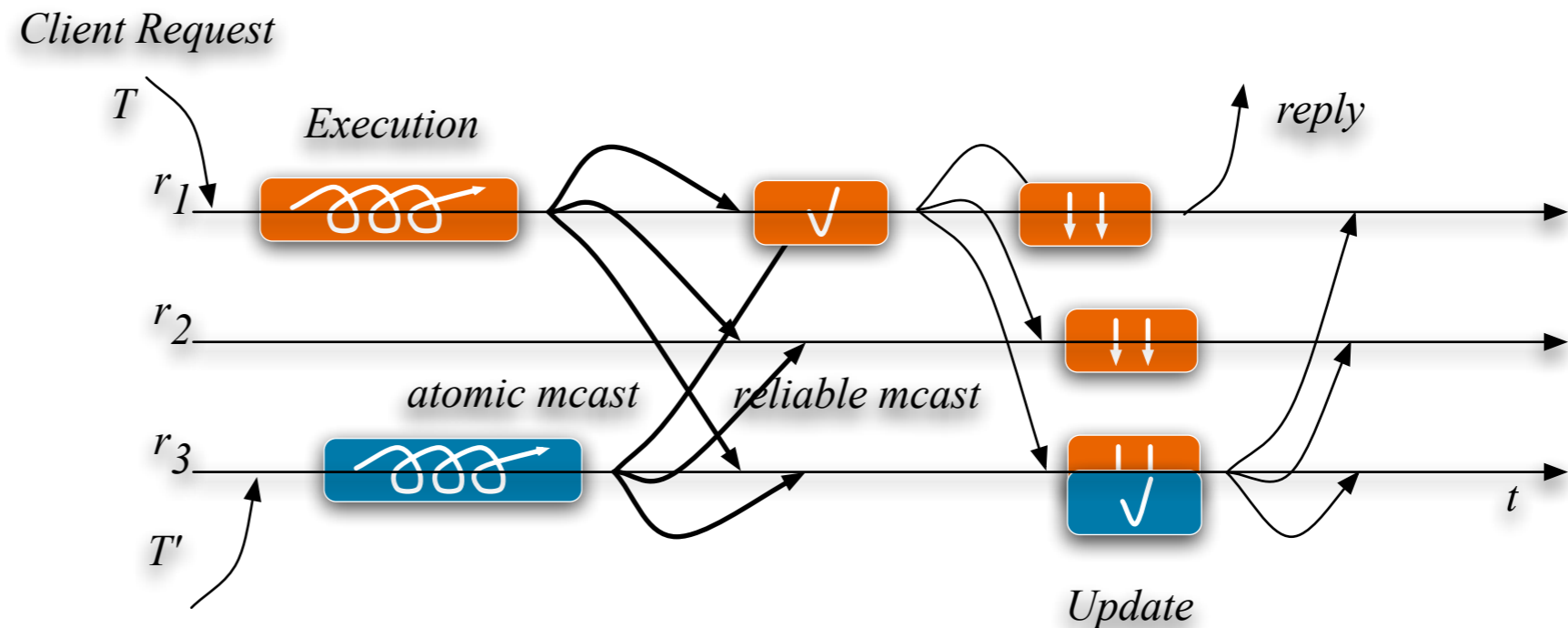
- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently



Postgres-R

# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

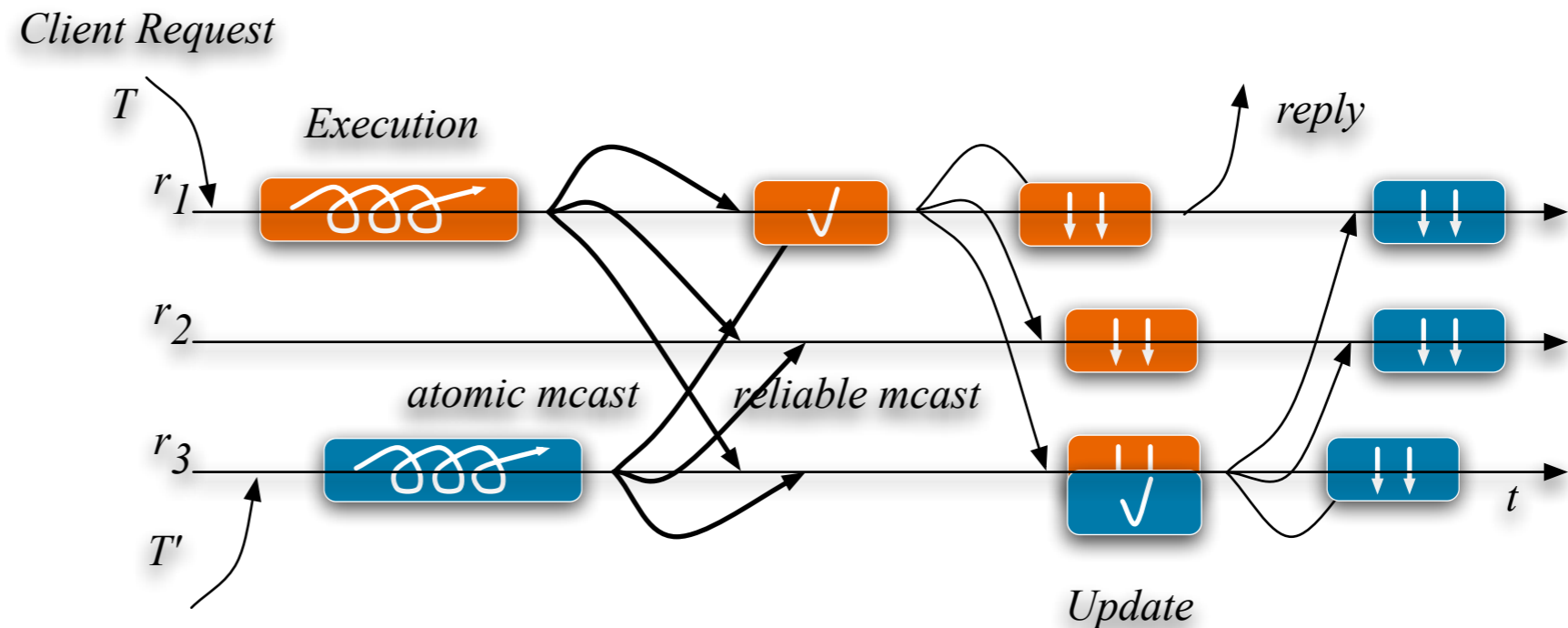


Postgres-R

# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

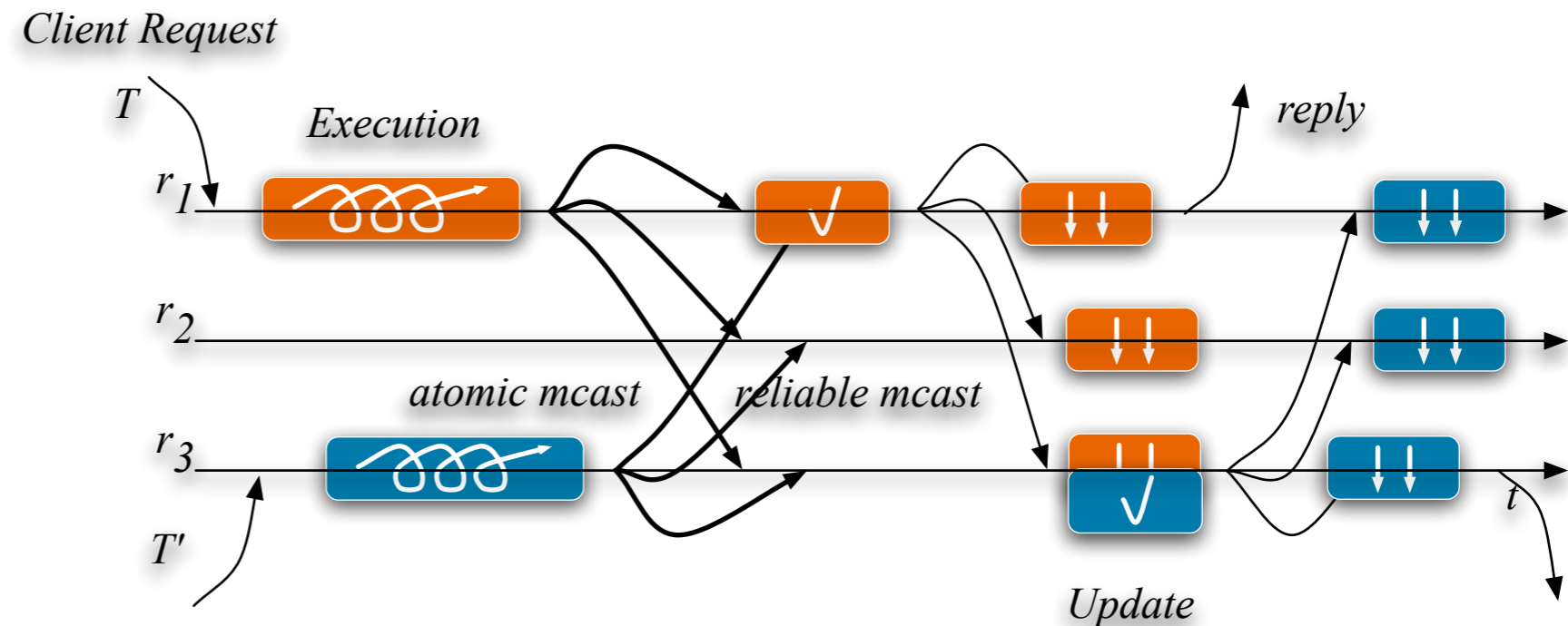
Postgres-R



# Optimistic Execution

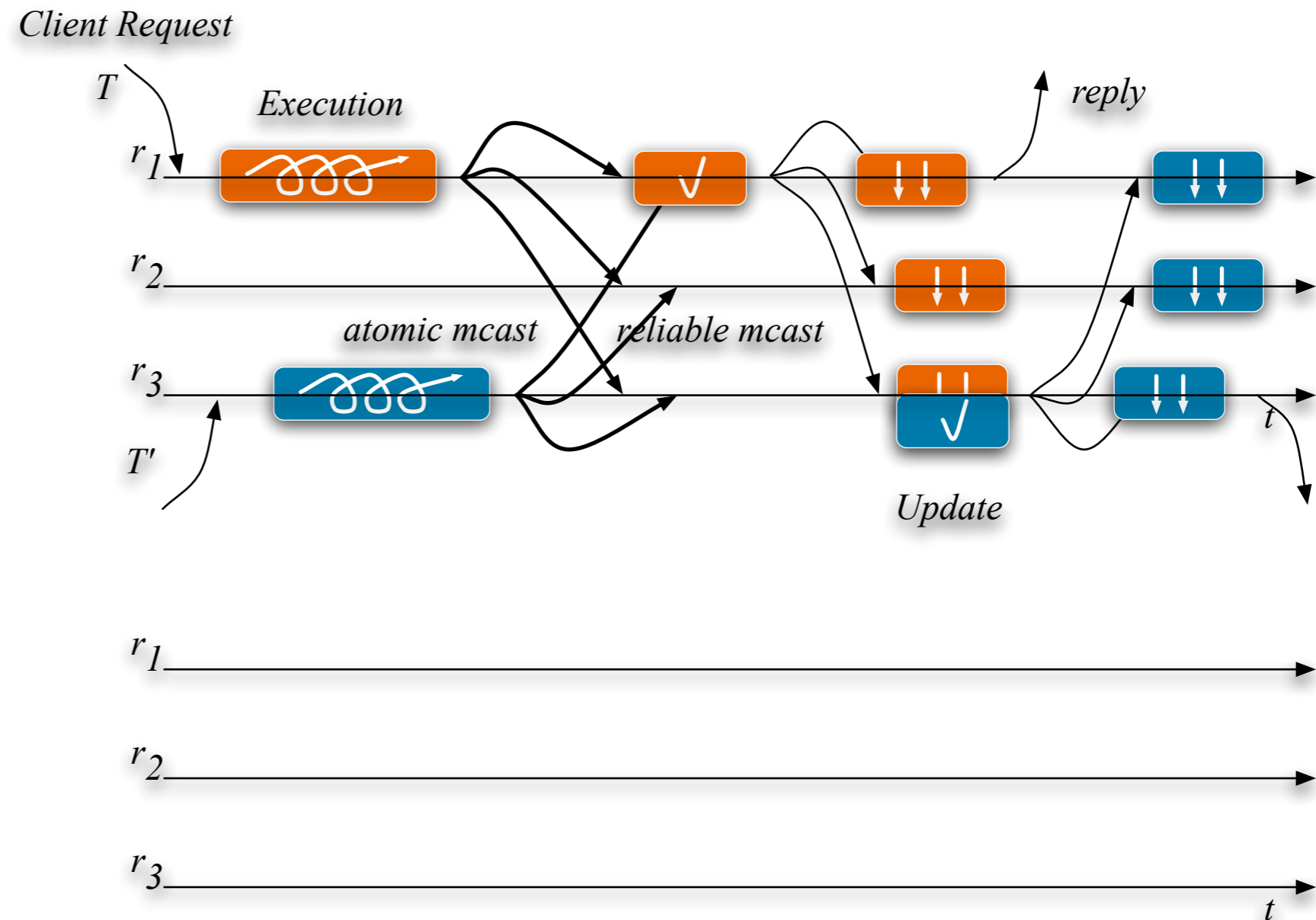
- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R



# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently



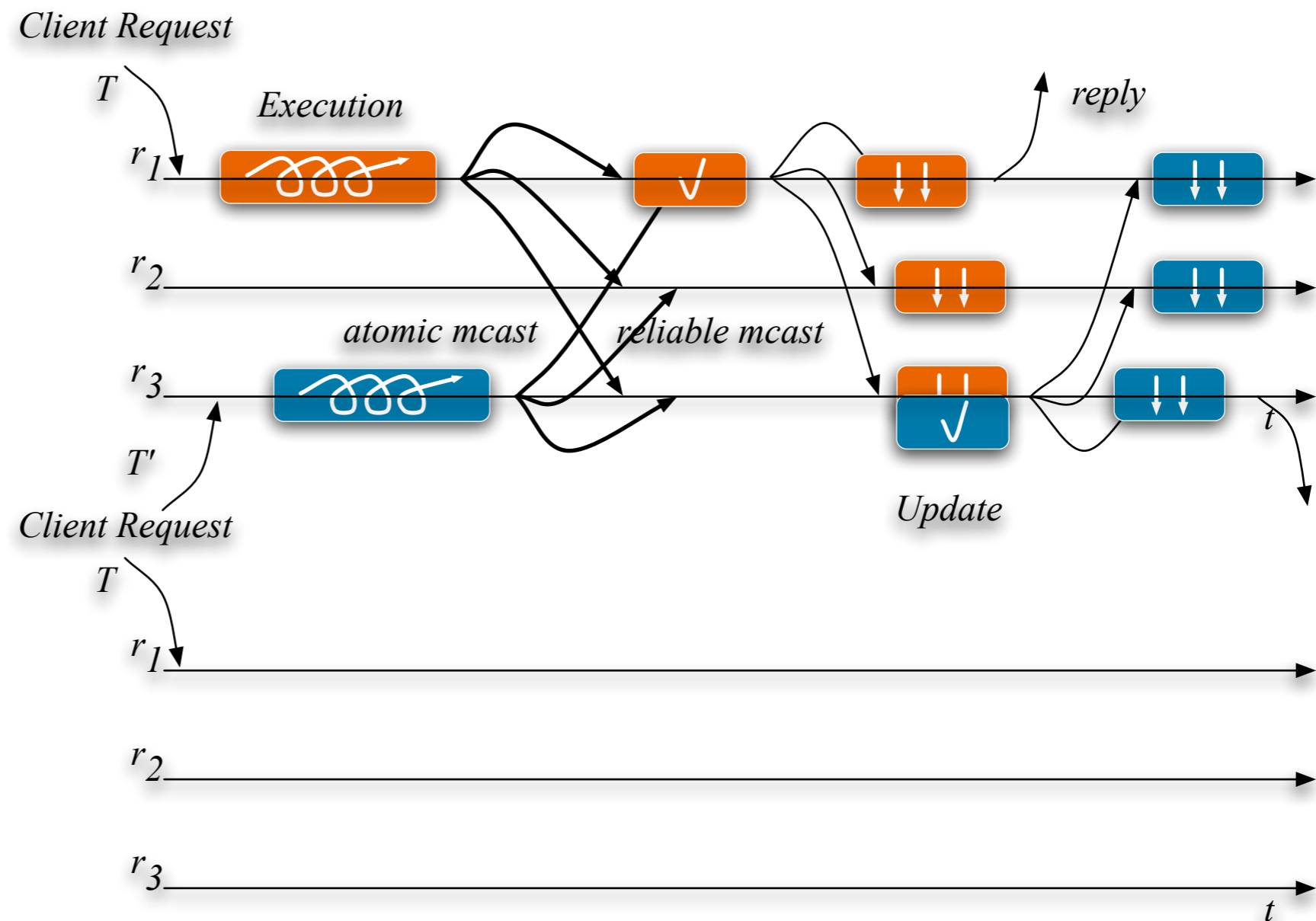
Postgres-R

DBSM

# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R

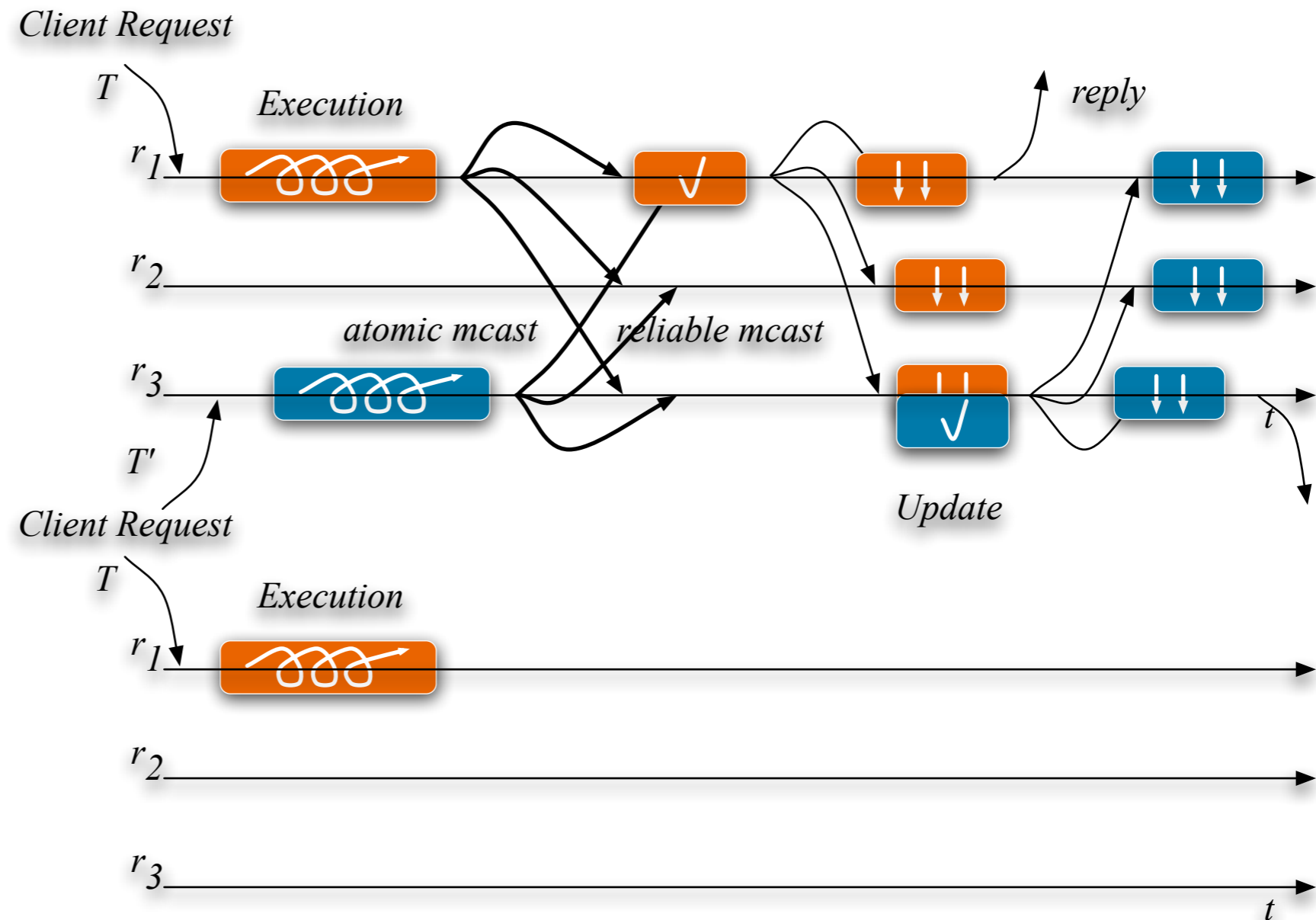


DBSM

# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R

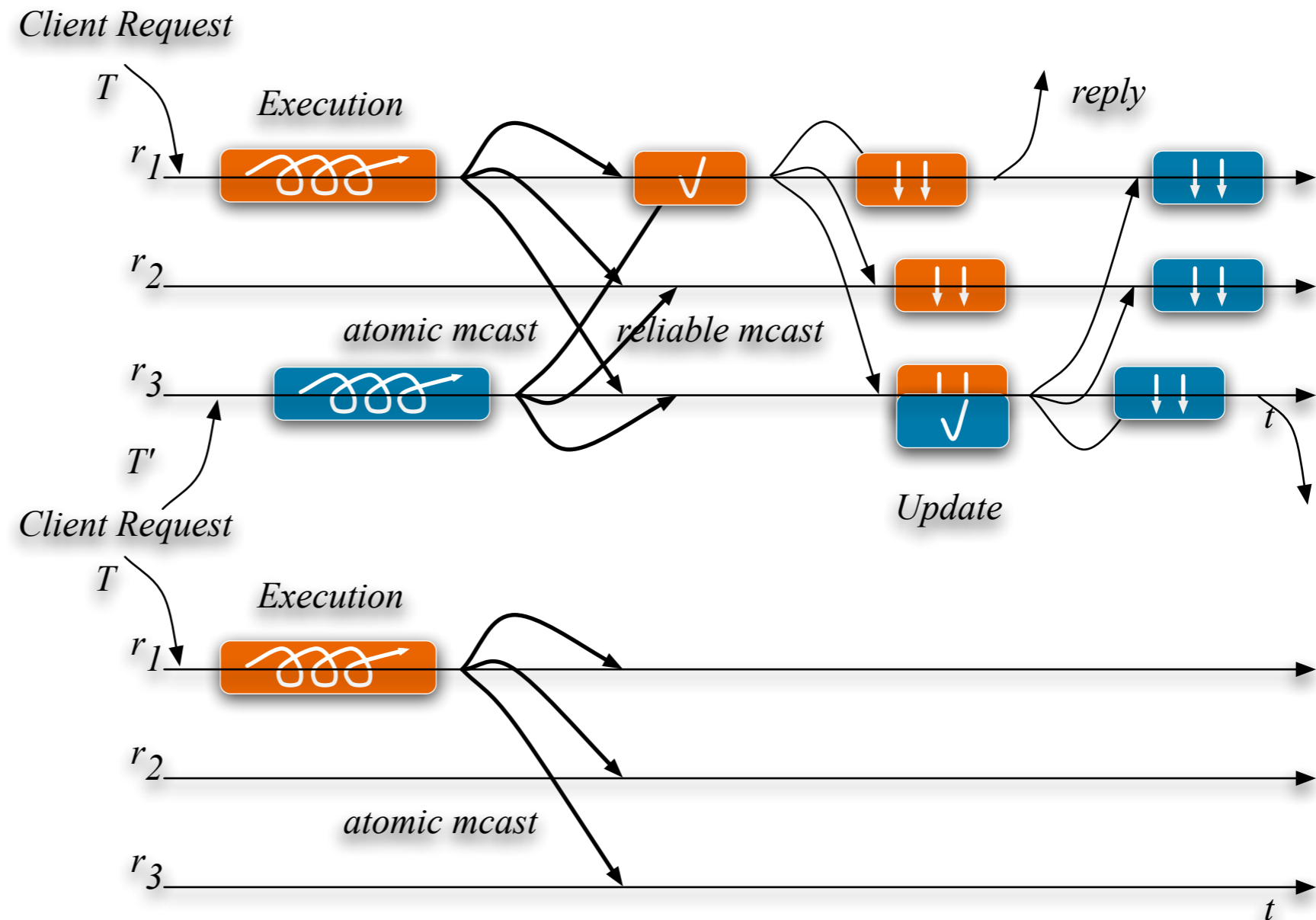


DBSM

# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R

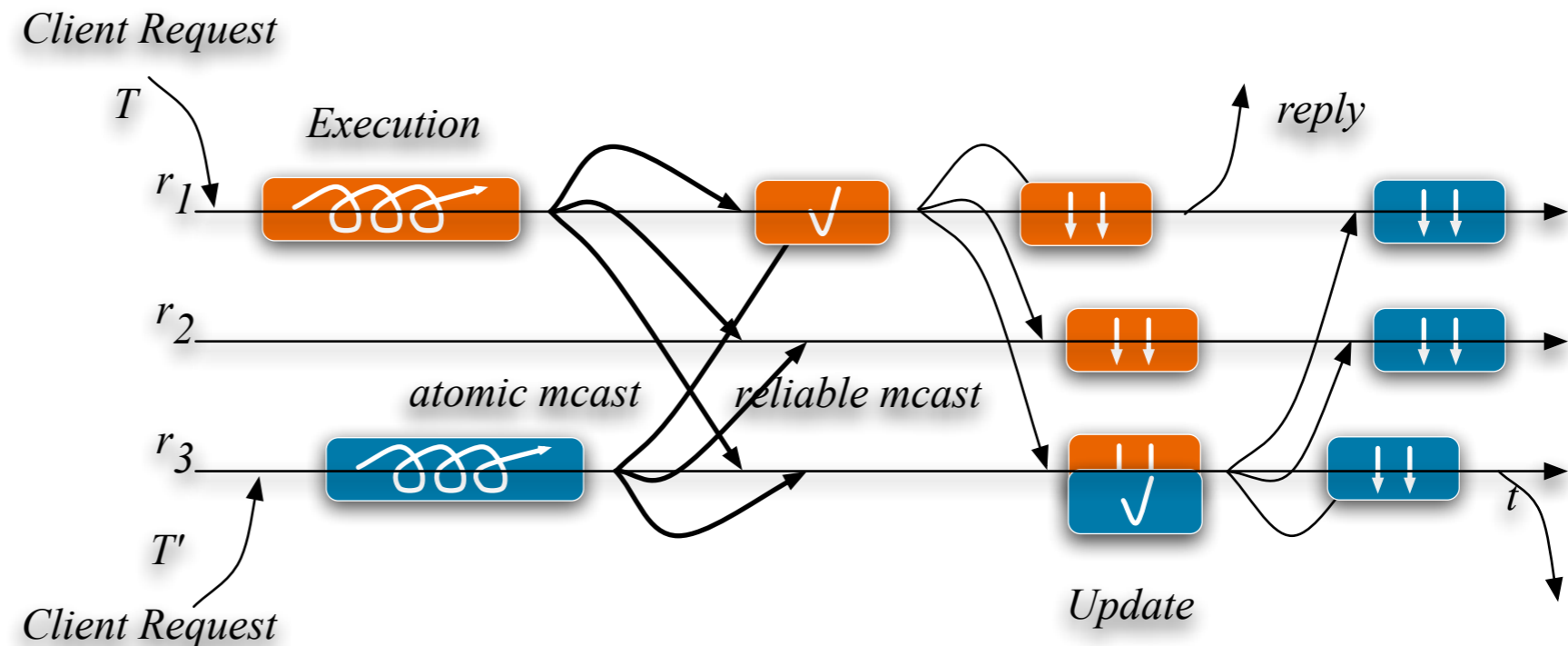


DBSM

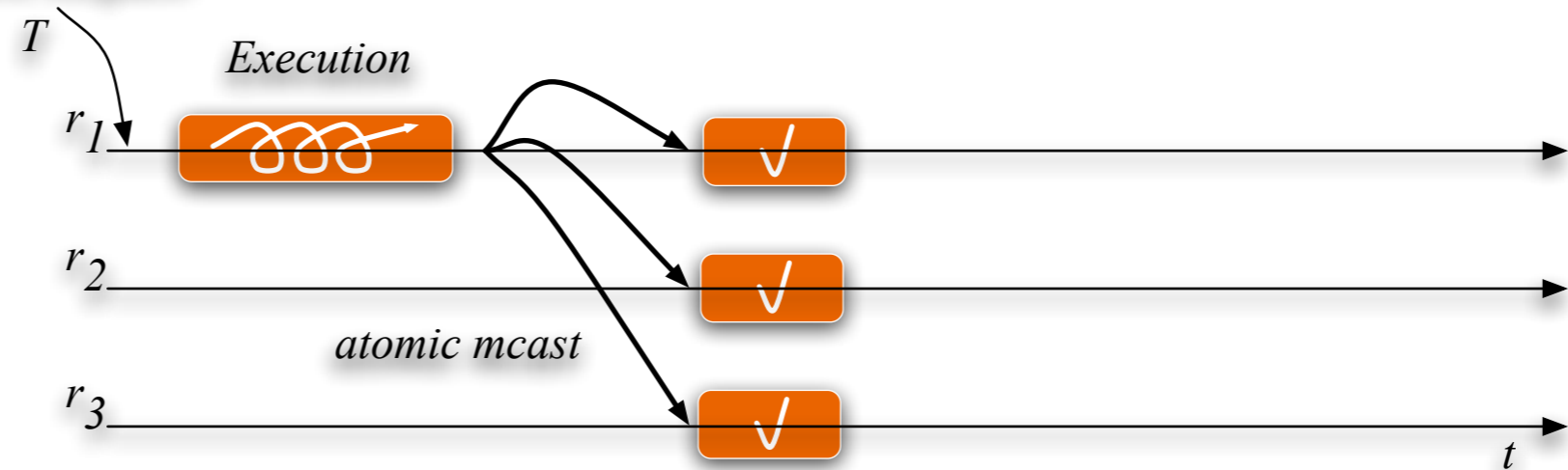
# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R



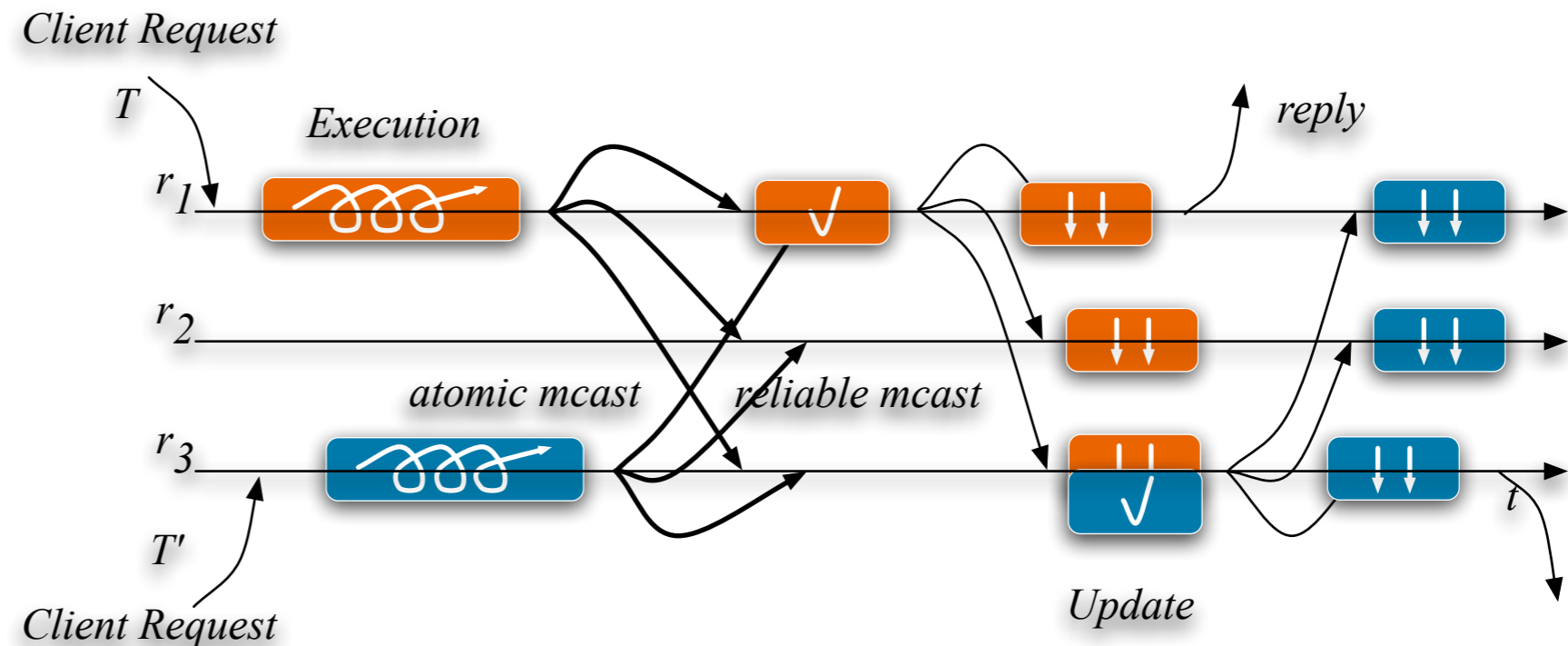
DBSM



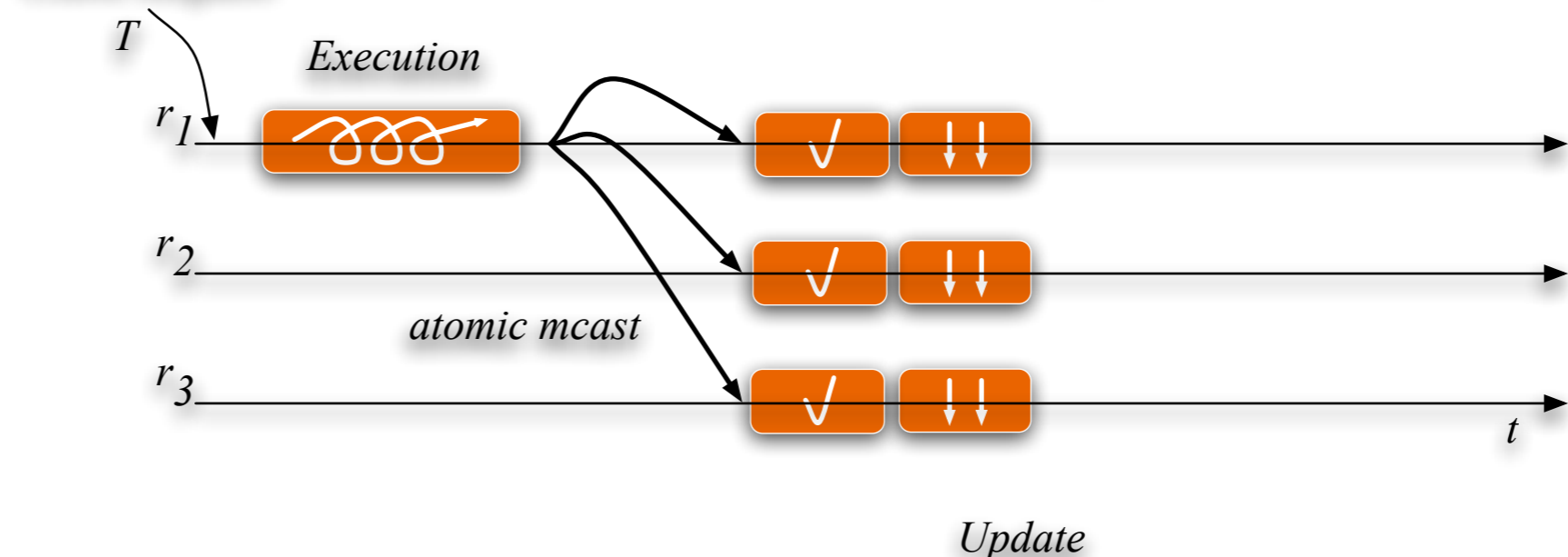
# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R



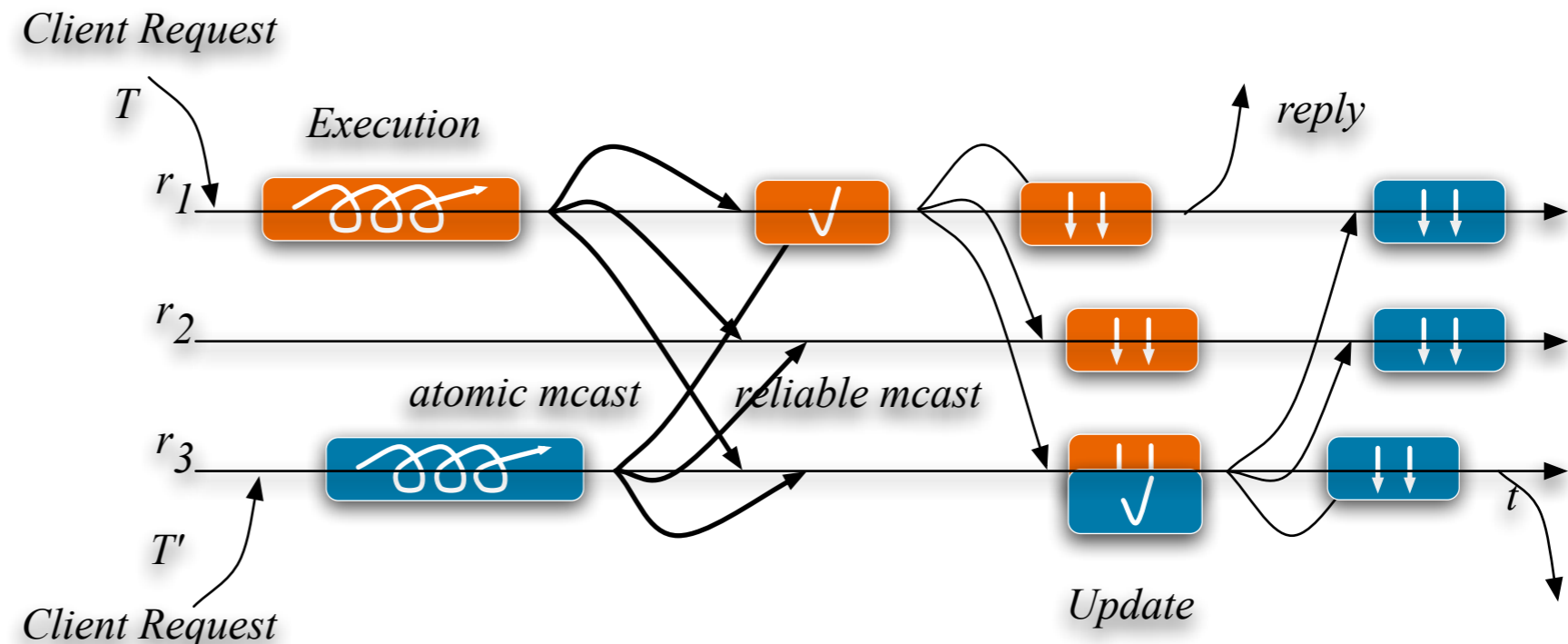
DBSM



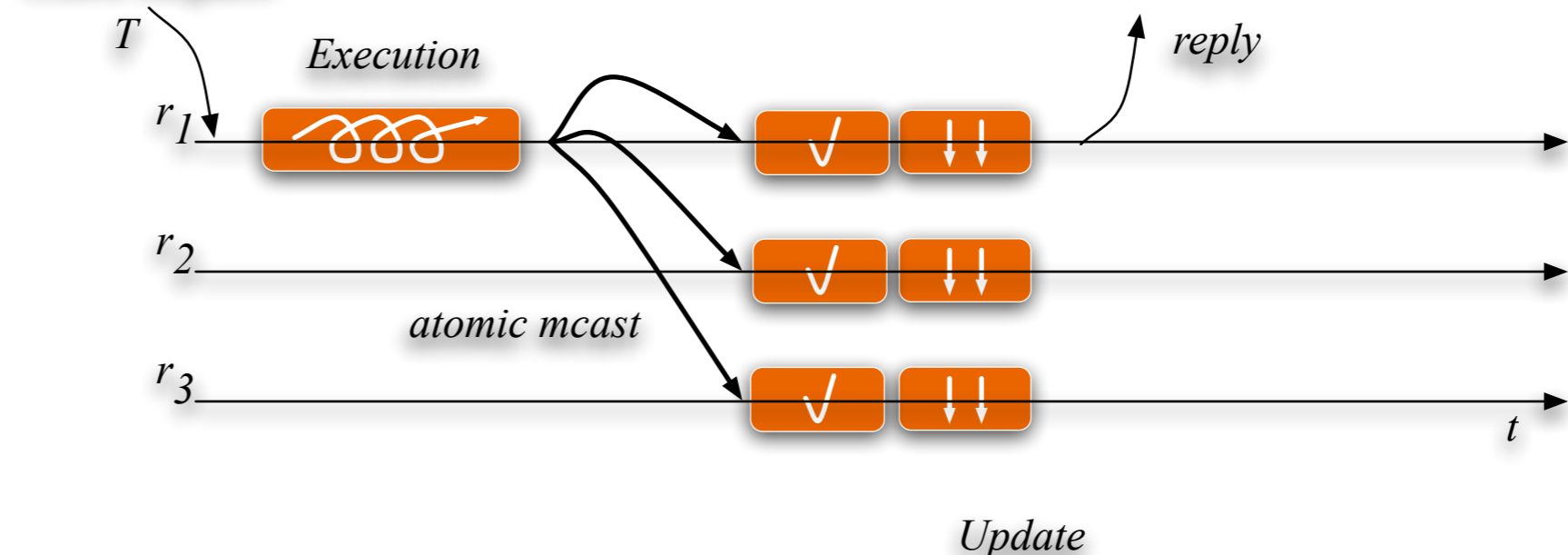
# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R



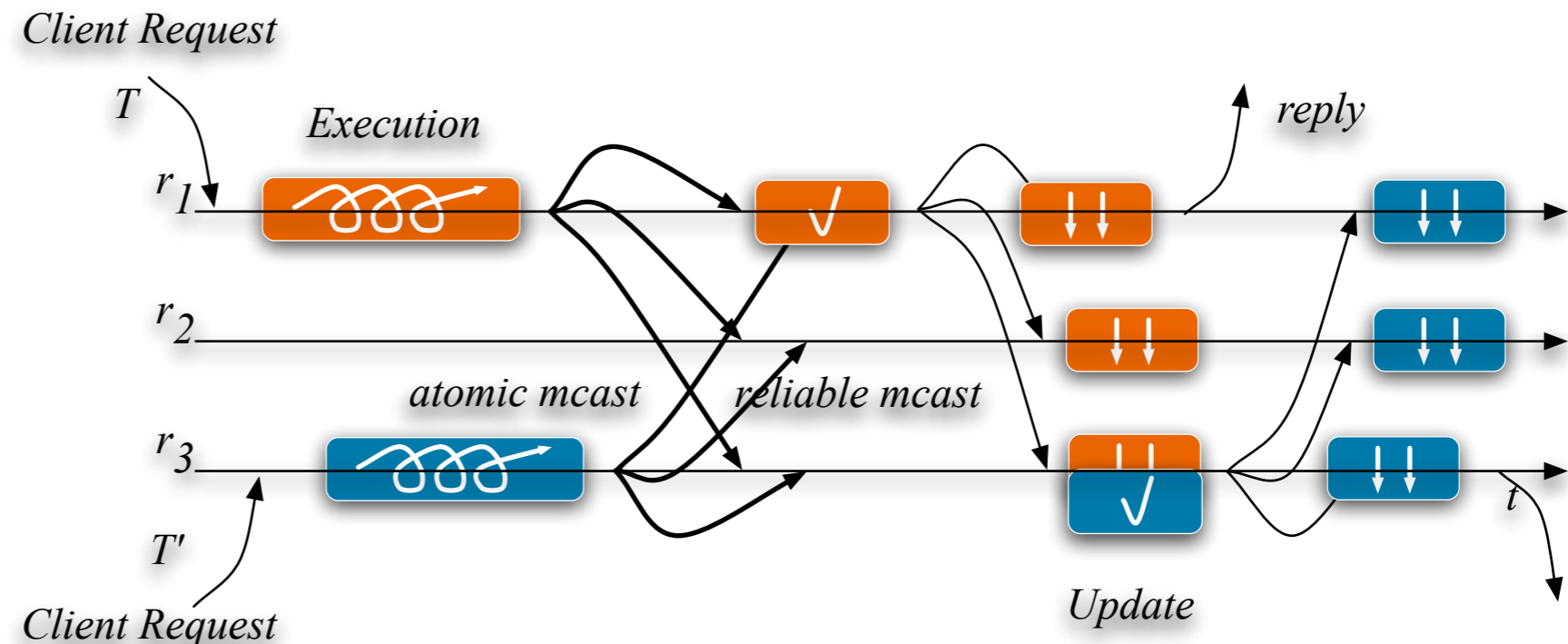
DBSM



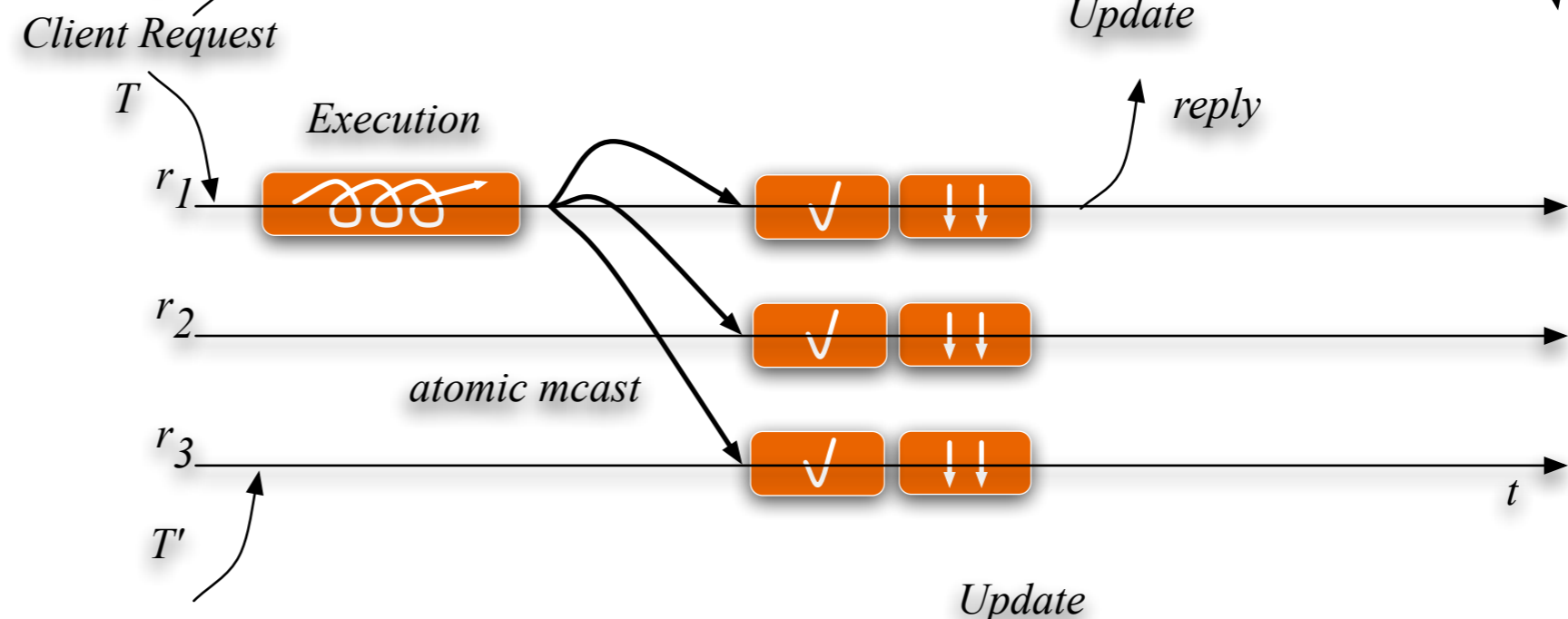
# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R



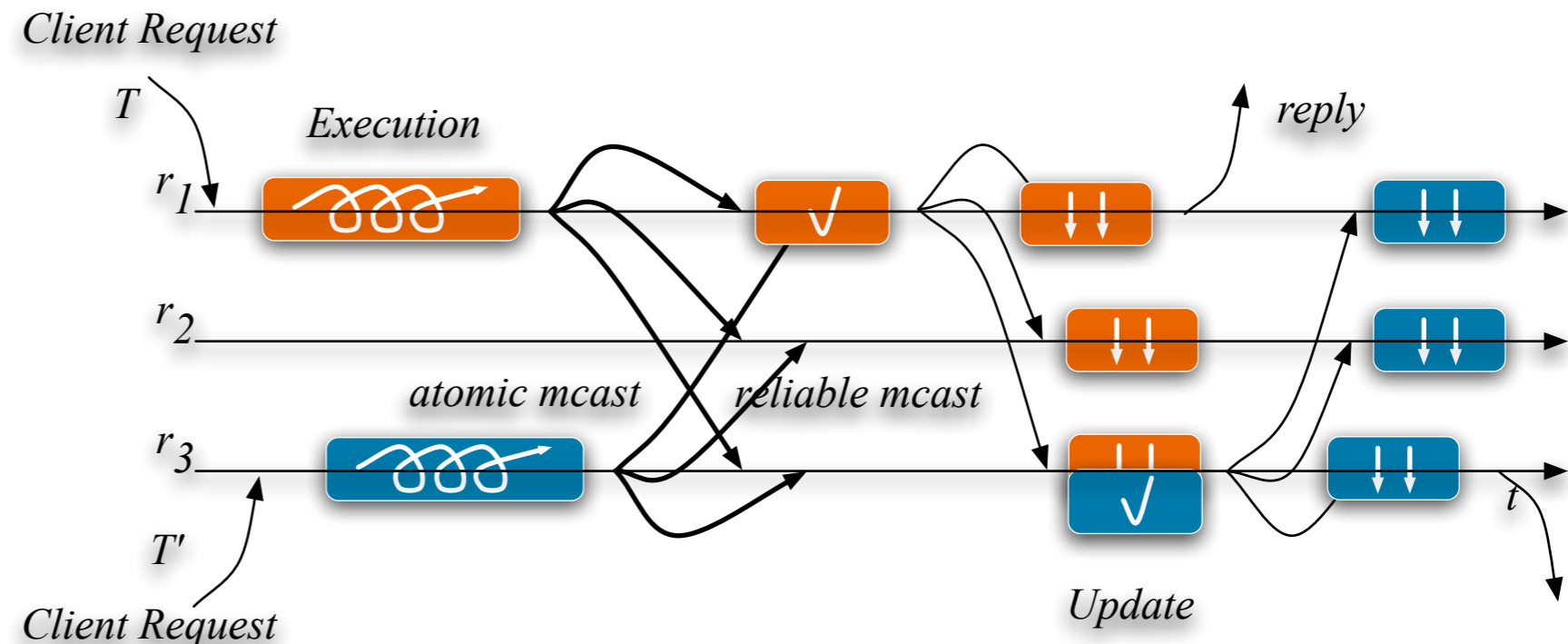
DBSM



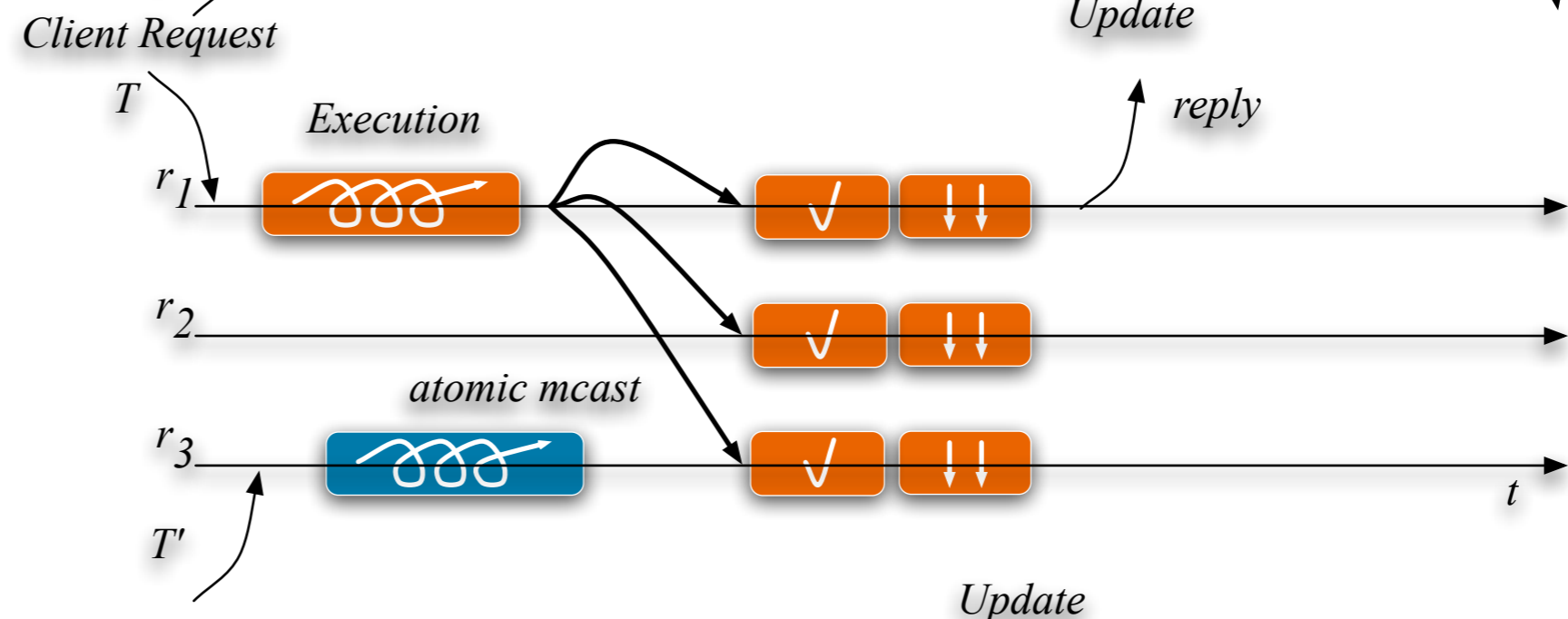
# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R



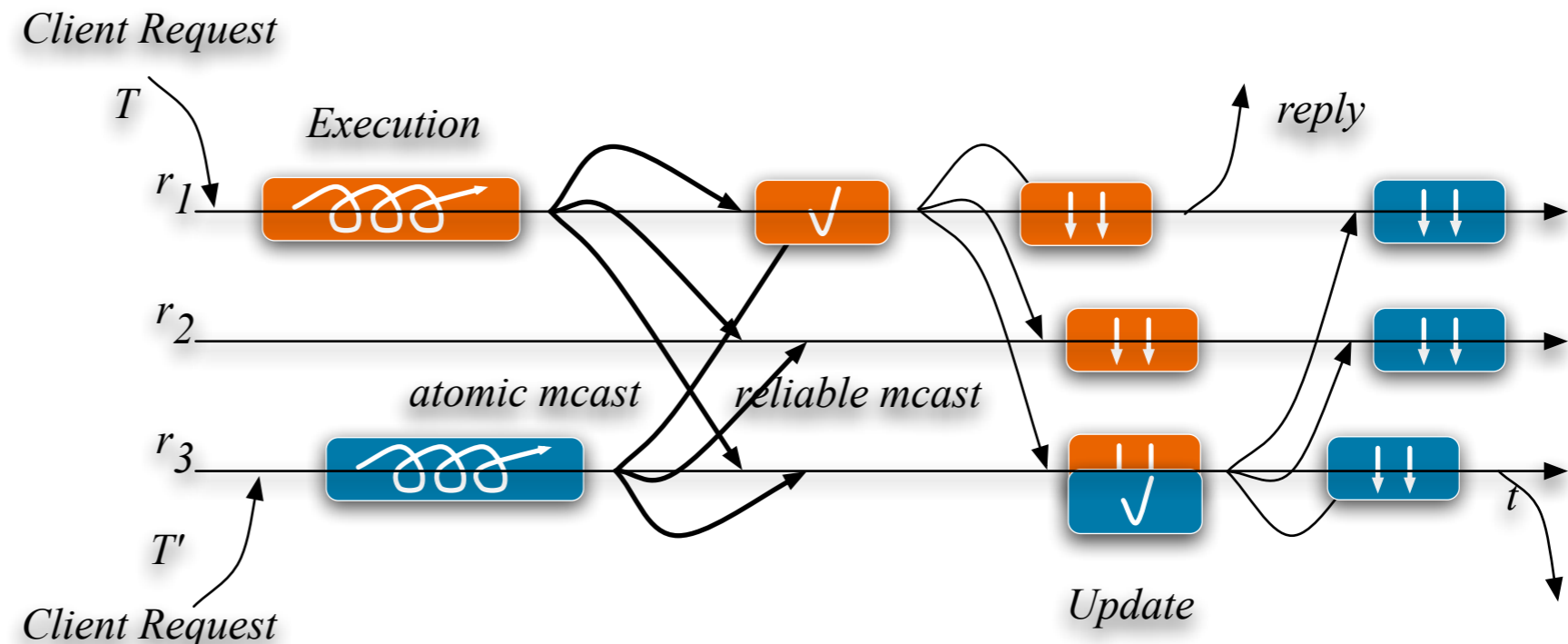
DBSM



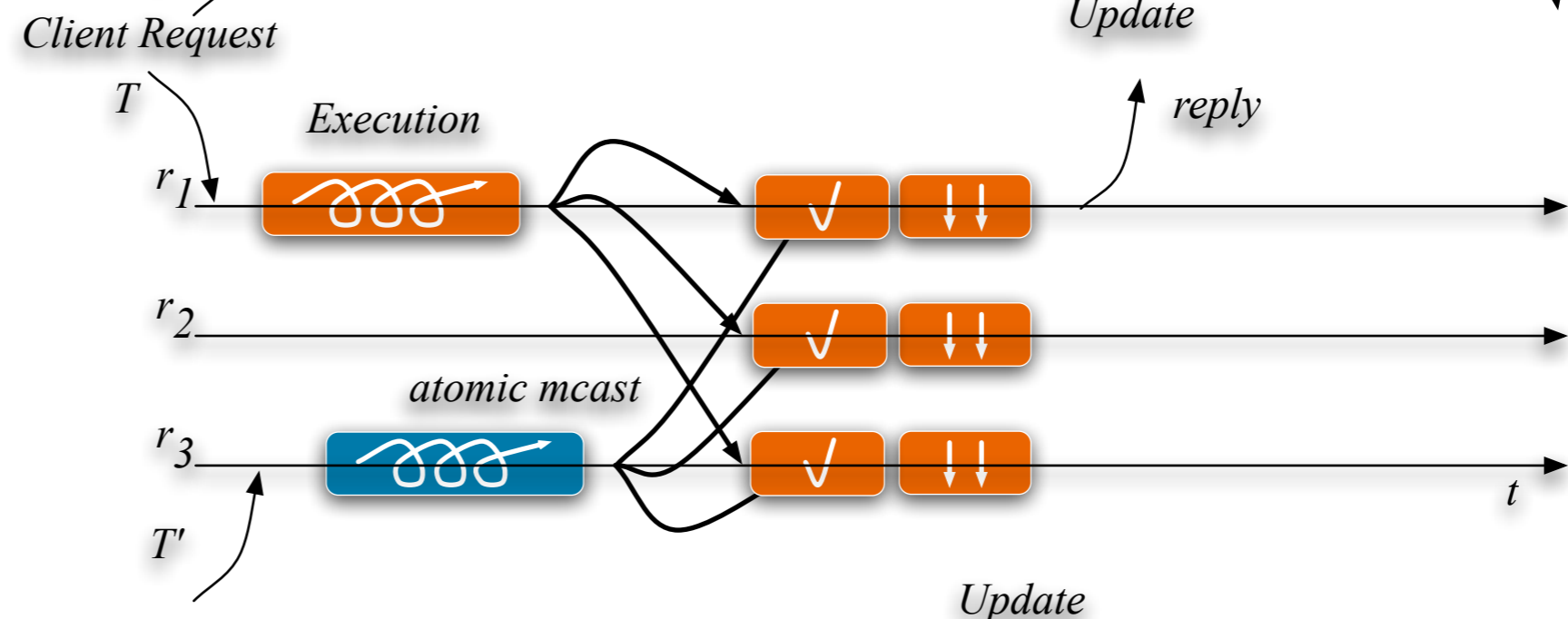
# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R



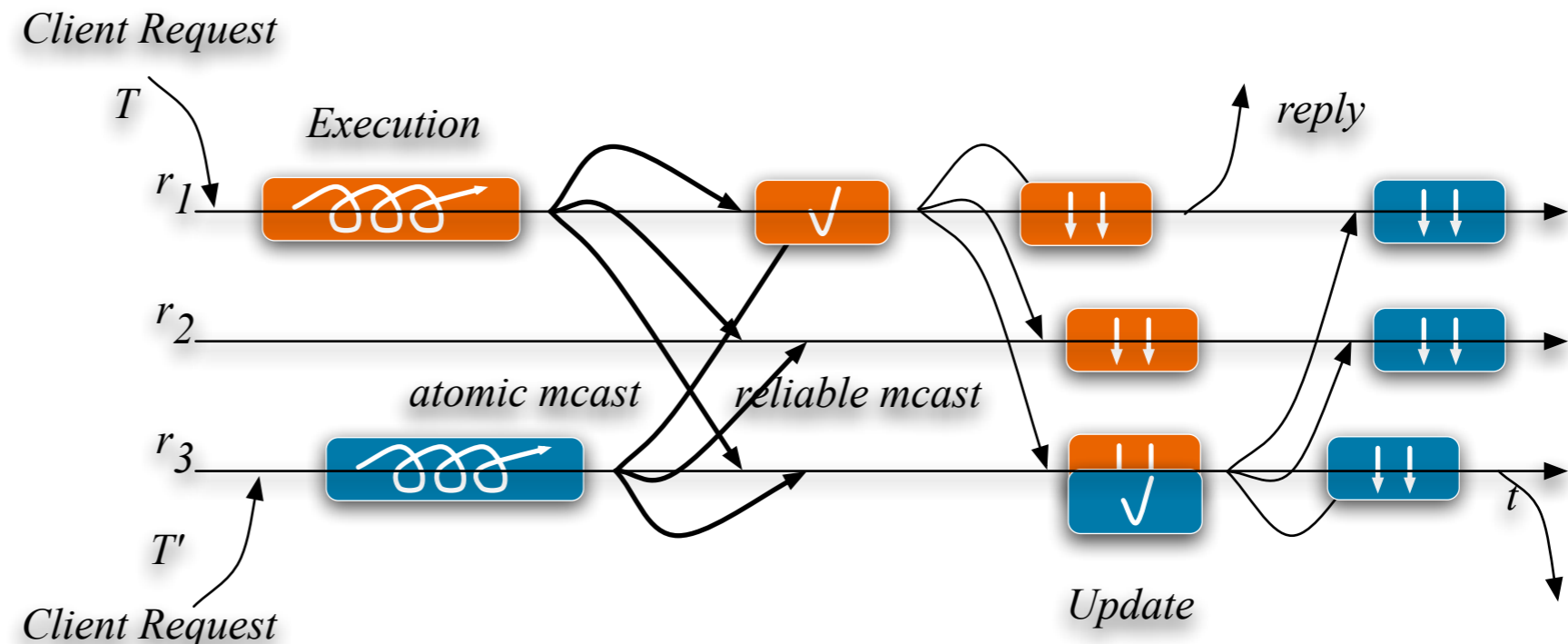
DBSM



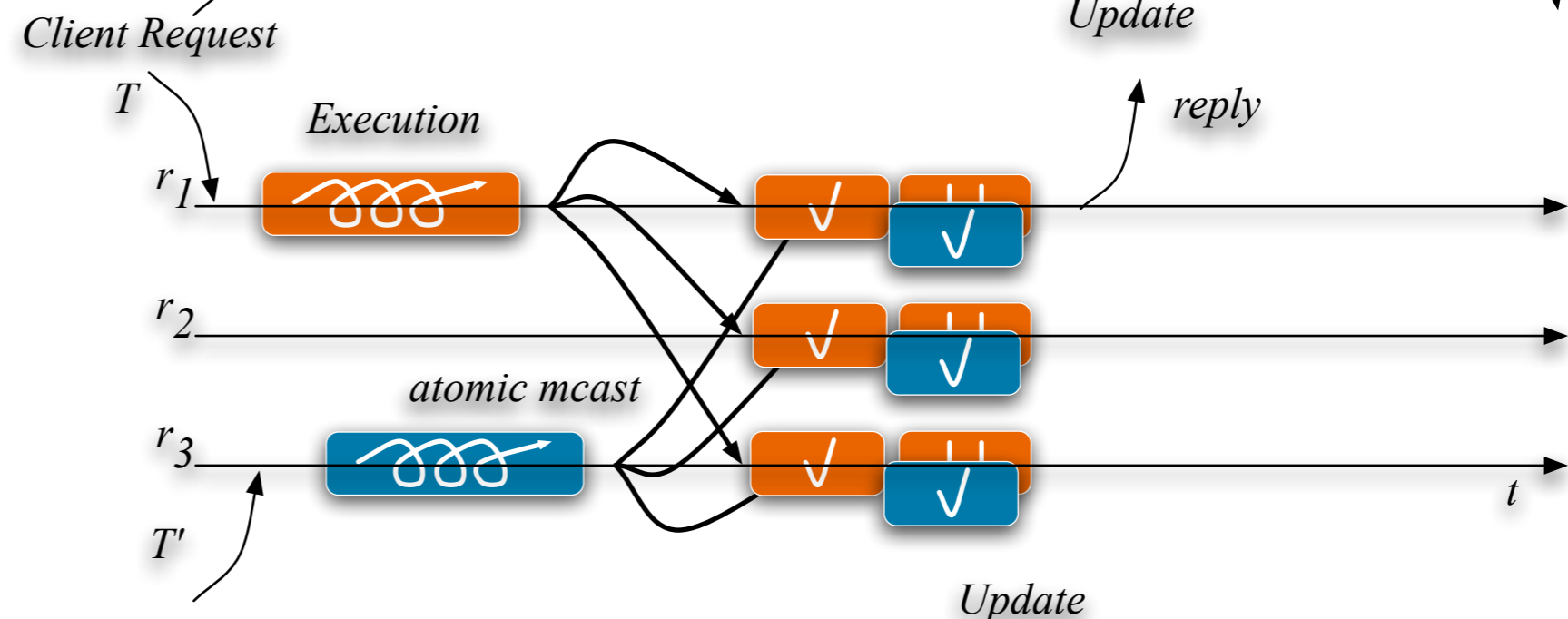
# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R



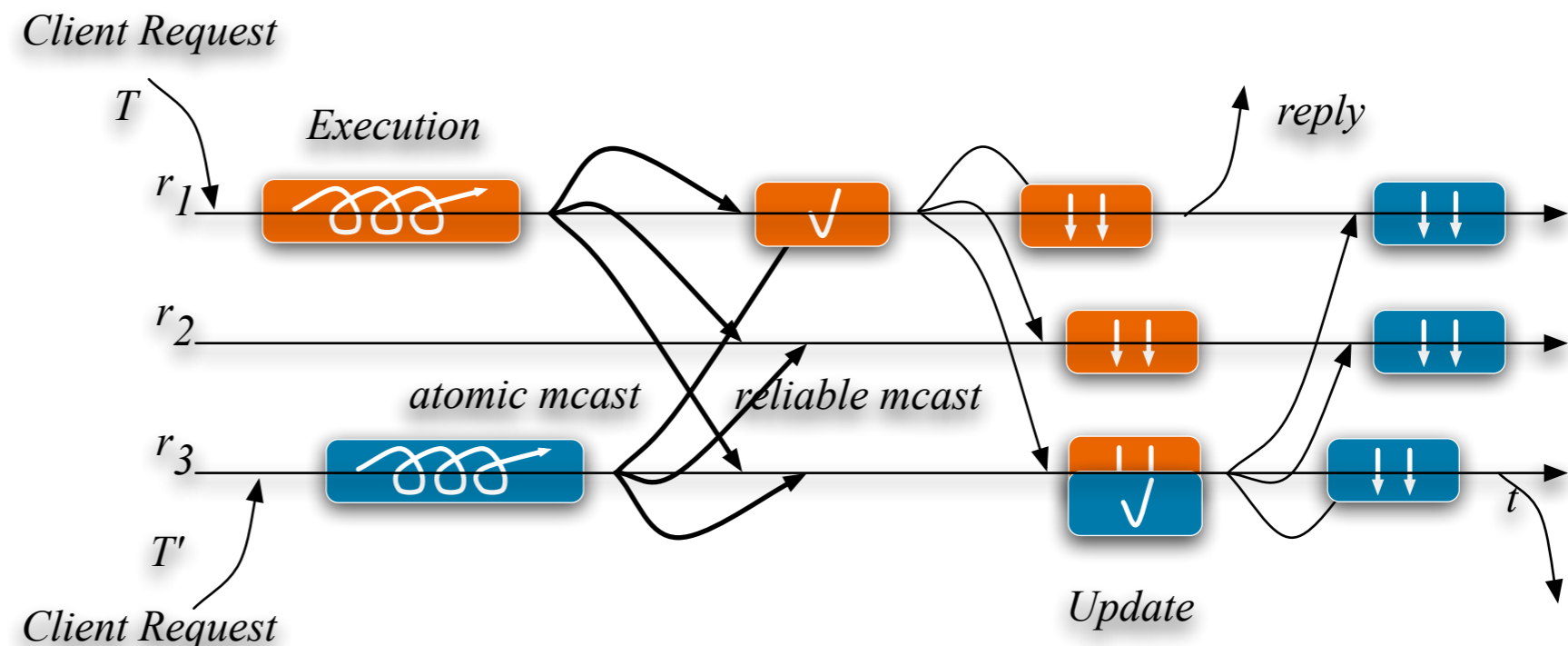
DBSM



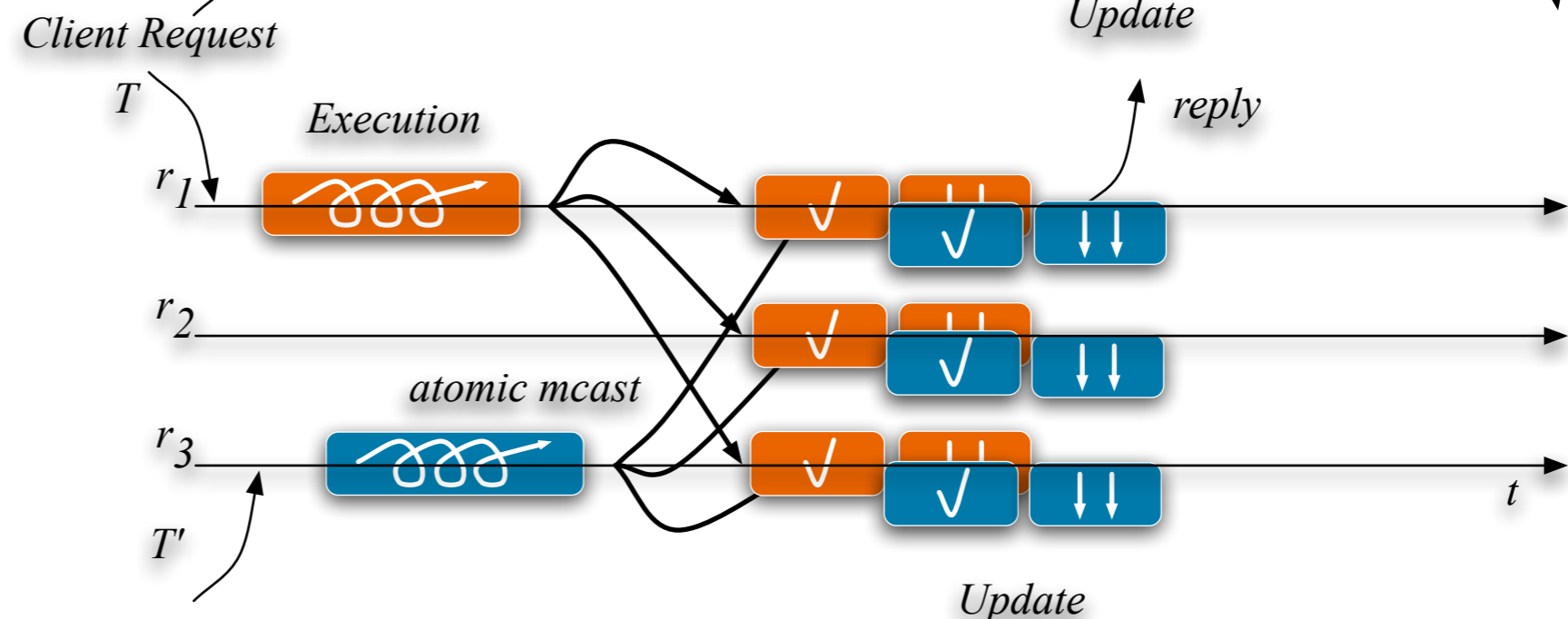
# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R



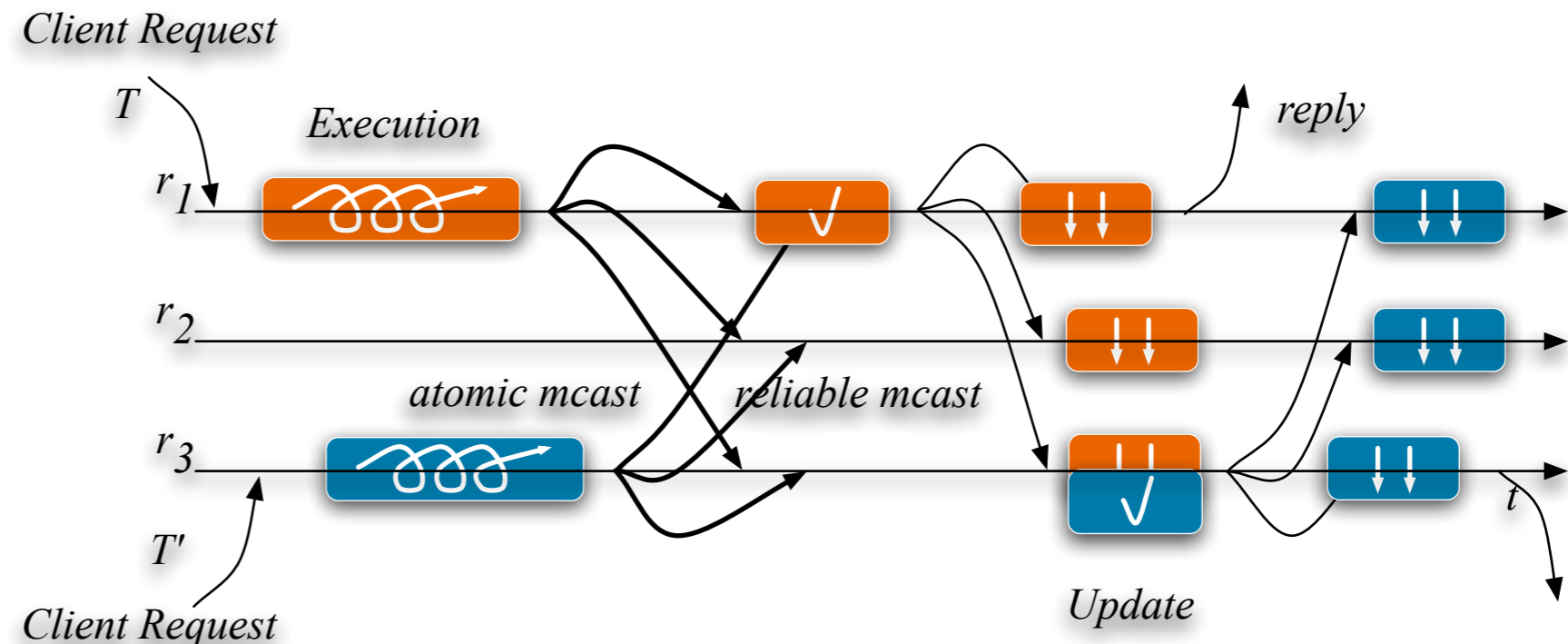
DBSM



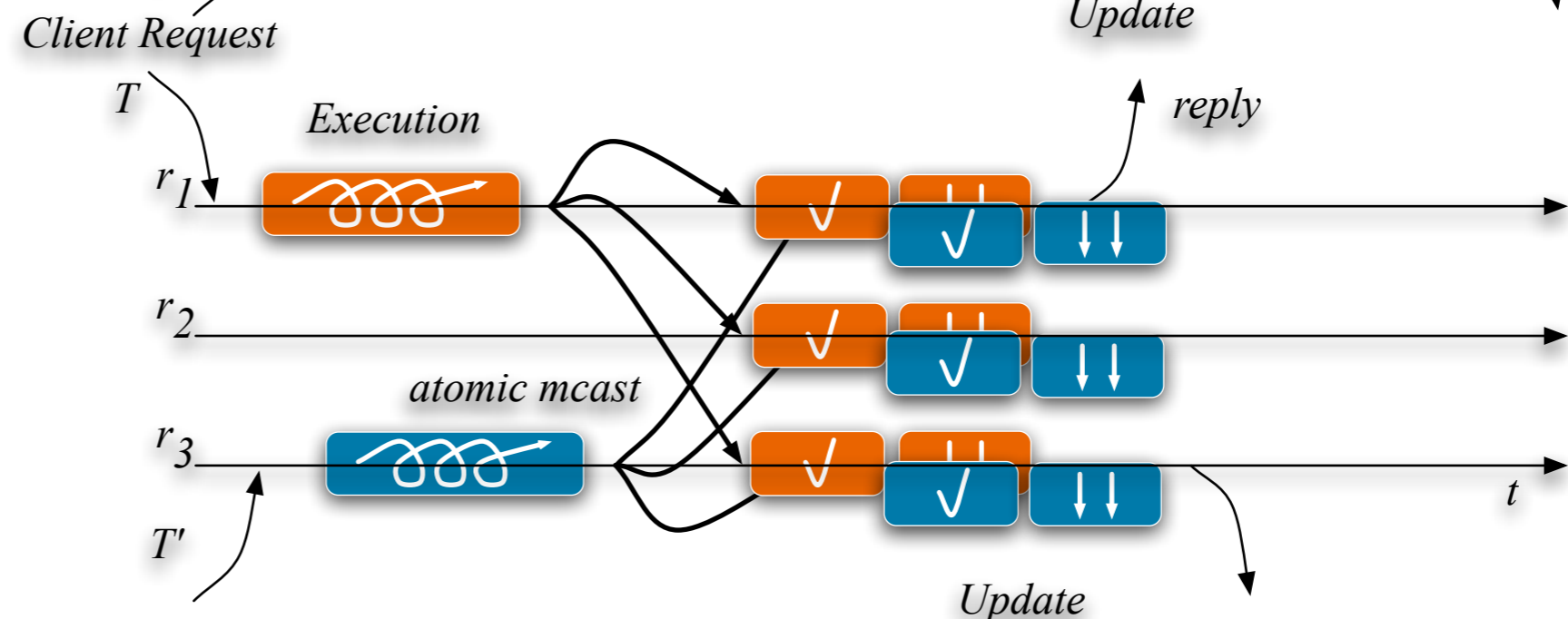
# Optimistic Execution

- Transactions are ordered after their execution
- Conflicting transactions may execute concurrently

Postgres-R



DBSM



# DBMS Interface Approaches



# DBMS Interface Approaches

- Normal client interfaces
  - Portable and reasonably efficient
  - Confined to async PB strategy

# DBMS Interface Approaches

- Normal client interfaces
  - Portable and reasonably efficient
  - Confined to async PB strategy
- Server wrapper
  - Middleware providing a virtual DB
  - Intercept, parse, delay, modify and route statements
  - PB and SM strategies

# DBMS Interface Approaches

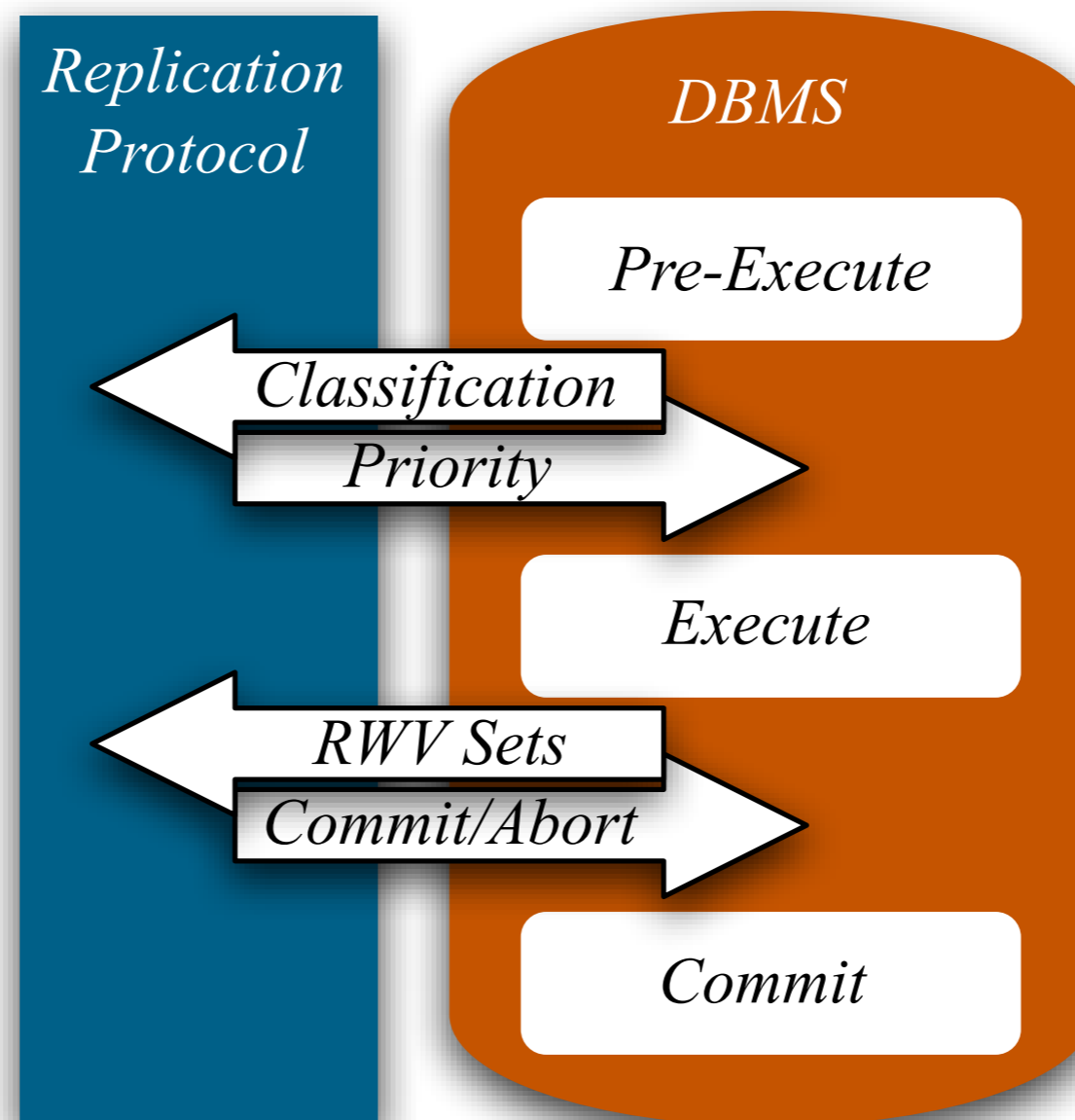
- Normal client interfaces
  - Portable and reasonably efficient
  - Confined to async PB strategy
- Server wrapper
  - Middleware providing a virtual DB
  - Intercept, parse, delay, modify and route statements
  - PB and SM strategies
- Server patch
  - Specific changes to the underlying DB
  - Need of source code, portability and maintenance issues

# DBMS Interface Approaches

- Normal client interfaces
  - Portable and reasonably efficient
  - Confined to async PB strategy
- Server wrapper
  - Middleware providing a virtual DB
  - Intercept, parse, delay, modify and route statements
  - PB and SM strategies
- Server patch
  - Specific changes to the underlying DB
  - Need of source code, portability and maintenance issues
- Proprietary interfaces

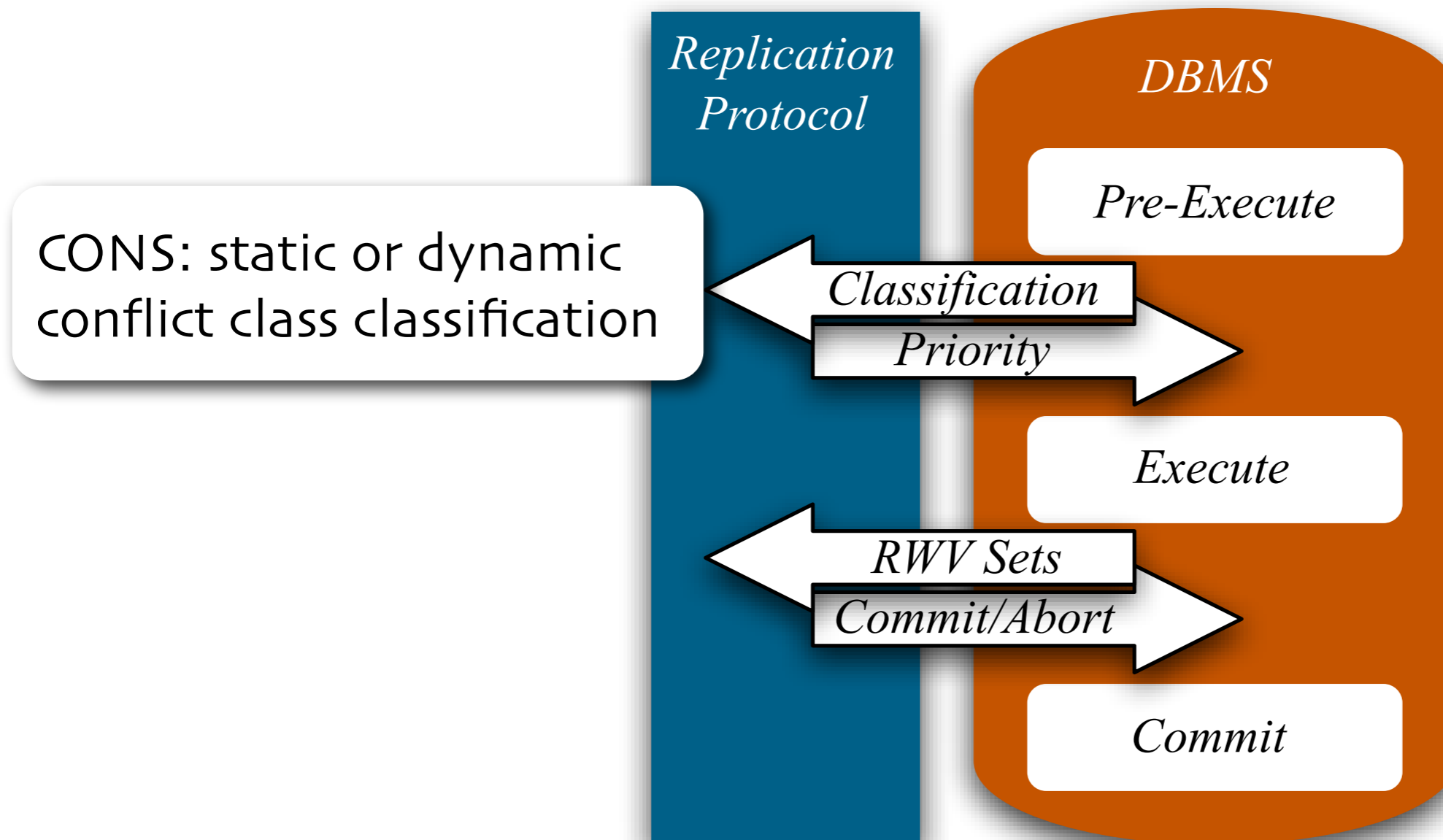
# Database Replication Interface

- The efficient implementation of these protocols requires specific interactions with the database engine



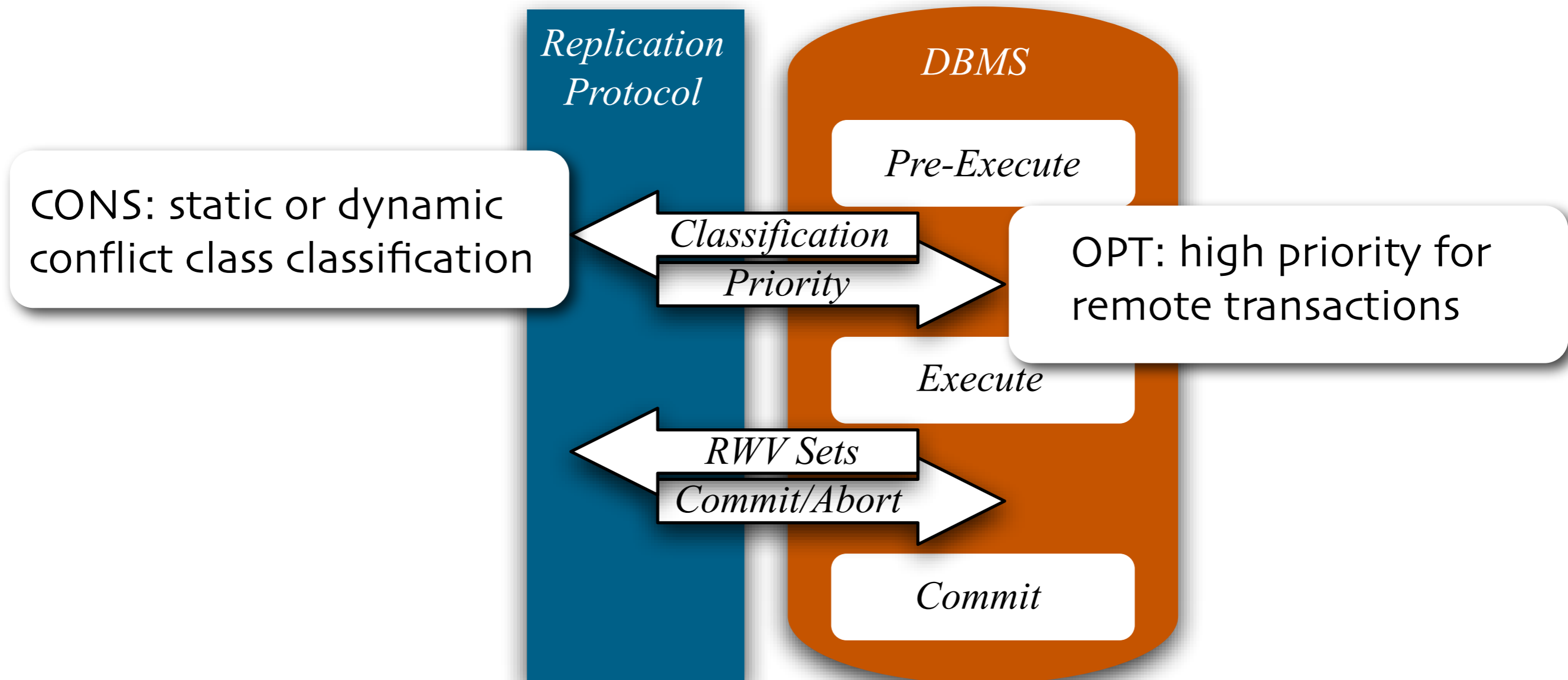
# Database Replication Interface

- The efficient implementation of these protocols requires specific interactions with the database engine



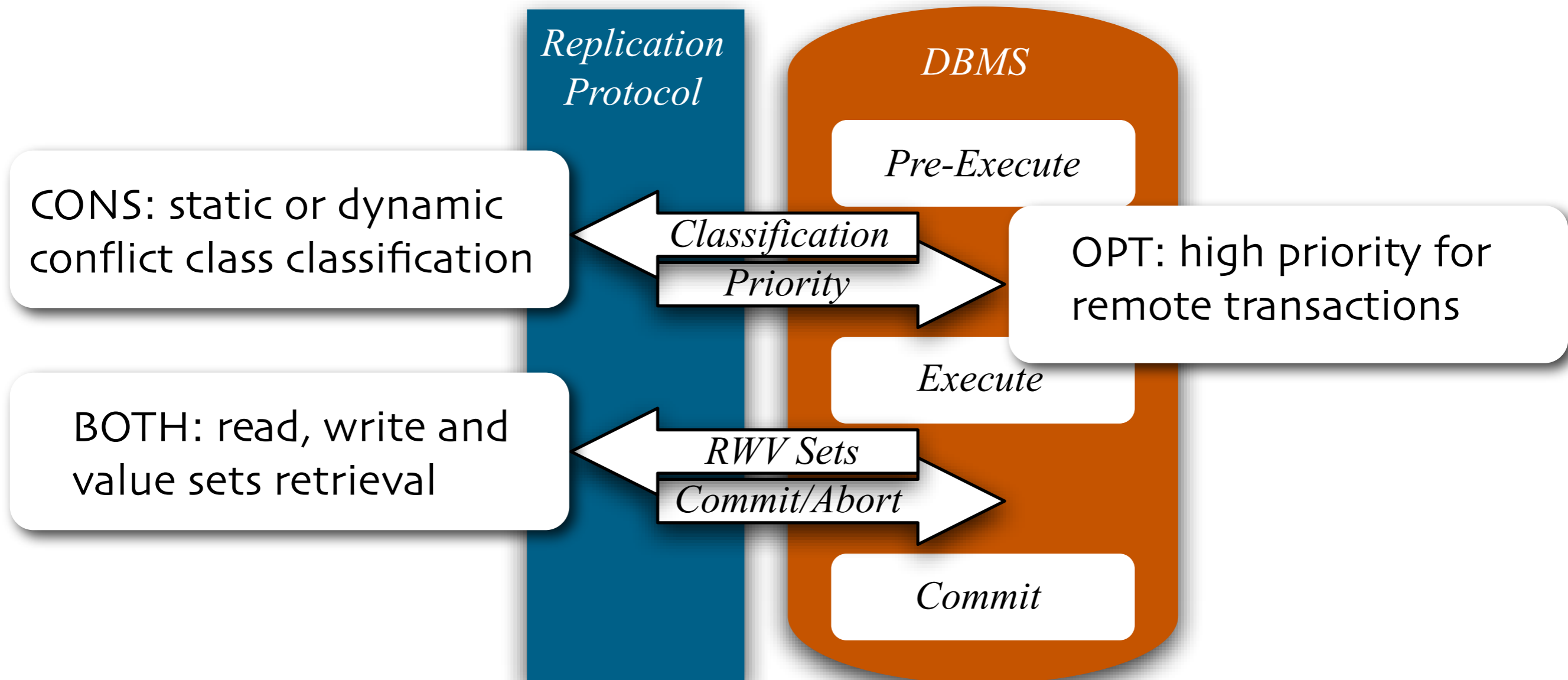
# Database Replication Interface

- The efficient implementation of these protocols requires specific interactions with the database engine



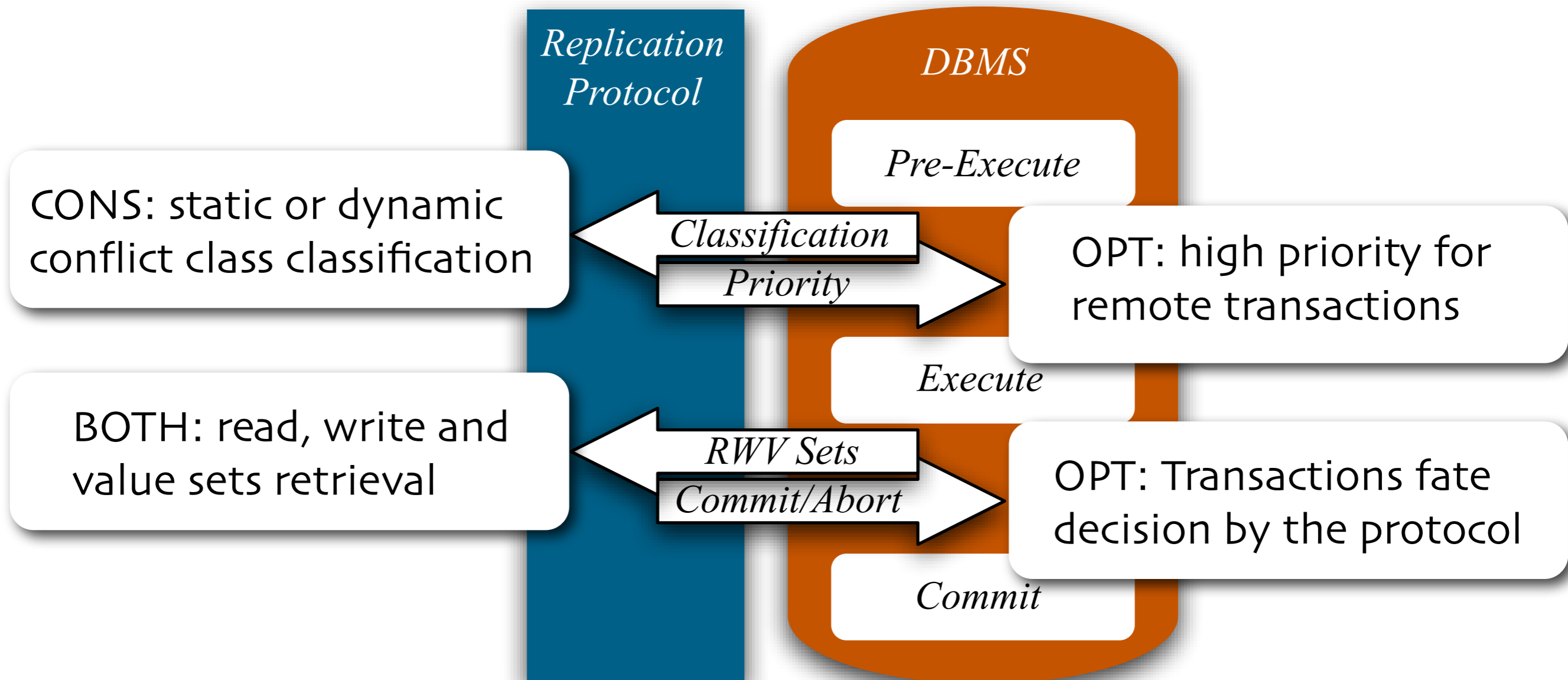
# Database Replication Interface

- The efficient implementation of these protocols requires specific interactions with the database engine



# Database Replication Interface

- The efficient implementation of these protocols requires specific interactions with the database engine



# Replication protocols requirements



# Replication protocols requirements

- Lifecycle events

# Replication protocols requirements

- Lifecycle events

Observe and control the lifecycle of a DBSMS: start, recover, go-live.

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information

Record and retrieve protocol specific meta-information associated with objects and tx.

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection

Interception of statements submitted by clients in textual form or as a parsed tree.  
(SM+DDL+CClasses)

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification

Modification or  
cancellation of statements.  
(Non-det, partial, hetero)

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction

Updates to be propagated.  
Tx info for certification.

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection

Efficient write-set application preserving correctness and order constraints.

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection
- Transactional events

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection
- Transactional events

Observe transactions' begin, rollback or commit events.

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection
- Transactional events
- Commit validation

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection
- Transactional events
- Commit validation

Interception and validation of commit requests on an on-going transaction.

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection
- Transactional events
- Commit validation
- Predictable deadlock handling

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection
- Transactional events
- Commit validation
- Predictable deadlock handling

Mechanism that can be controlled by middleware.  
(Cert'd tx)

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection
- Transactional events
- Commit validation
- Predictable deadlock handling
- Result-set injection



# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection
- Transactional events
- Commit validation
- Predictable deadlock handling
- Result-set injection

Ability to reconcile result sets for distributed (and mixed) execution.

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection
- Transactional events
- Commit validation
- Predictable deadlock handling
- Result-set injection
- Runtime model

# Replication protocols requirements

- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection
- Transactional events
- Commit validation
- Predictable deadlocks
- Result-set injection
- Runtime model

Uniformity in kernel's  
concurrency model

# Replication protocols requirements

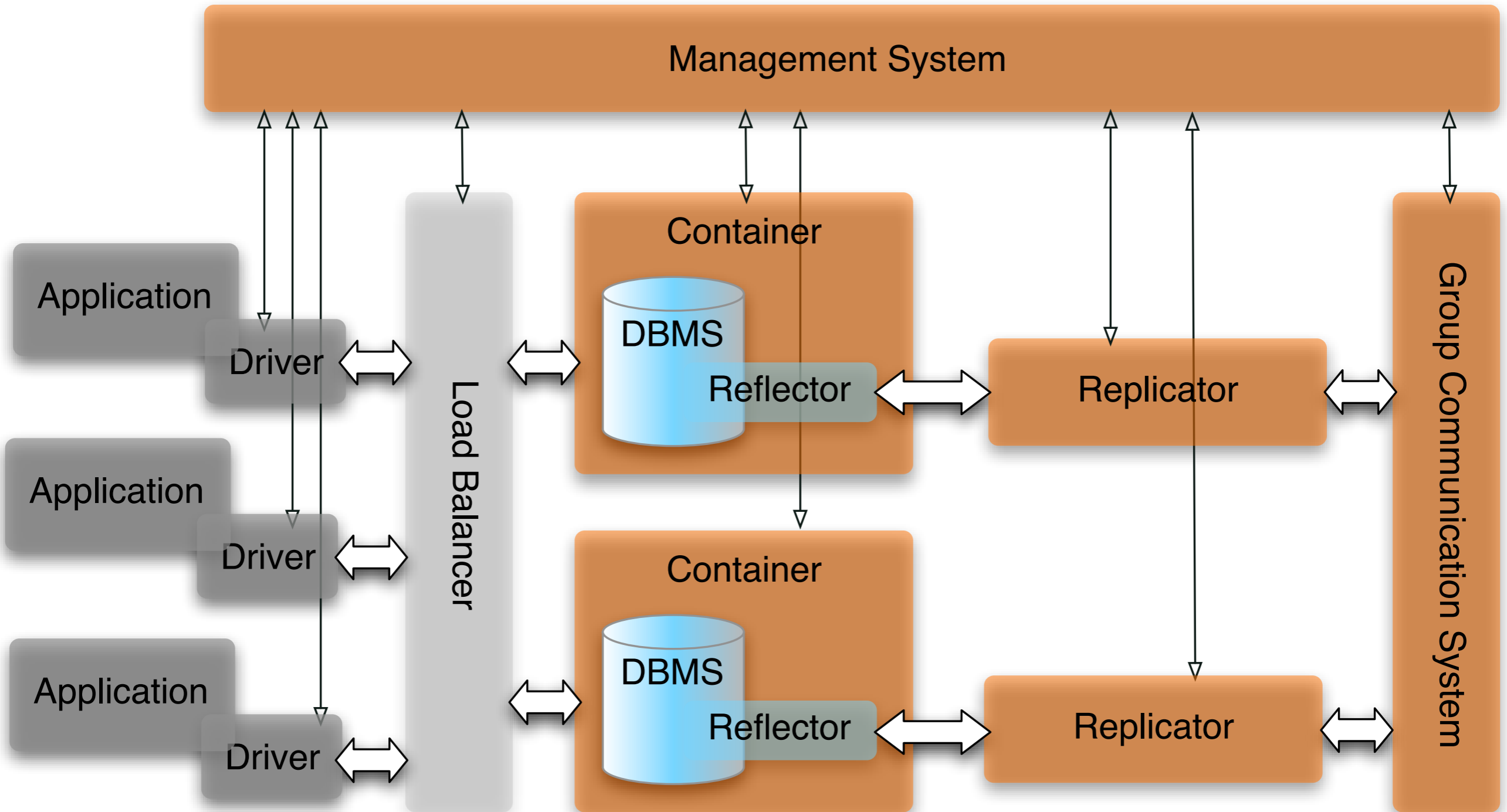
- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection
- Transactional events
- Commit validation
- Predictable deadlock handling
- Result-set injection
- Runtime model
- Configuration storage

# Replication protocols requirements

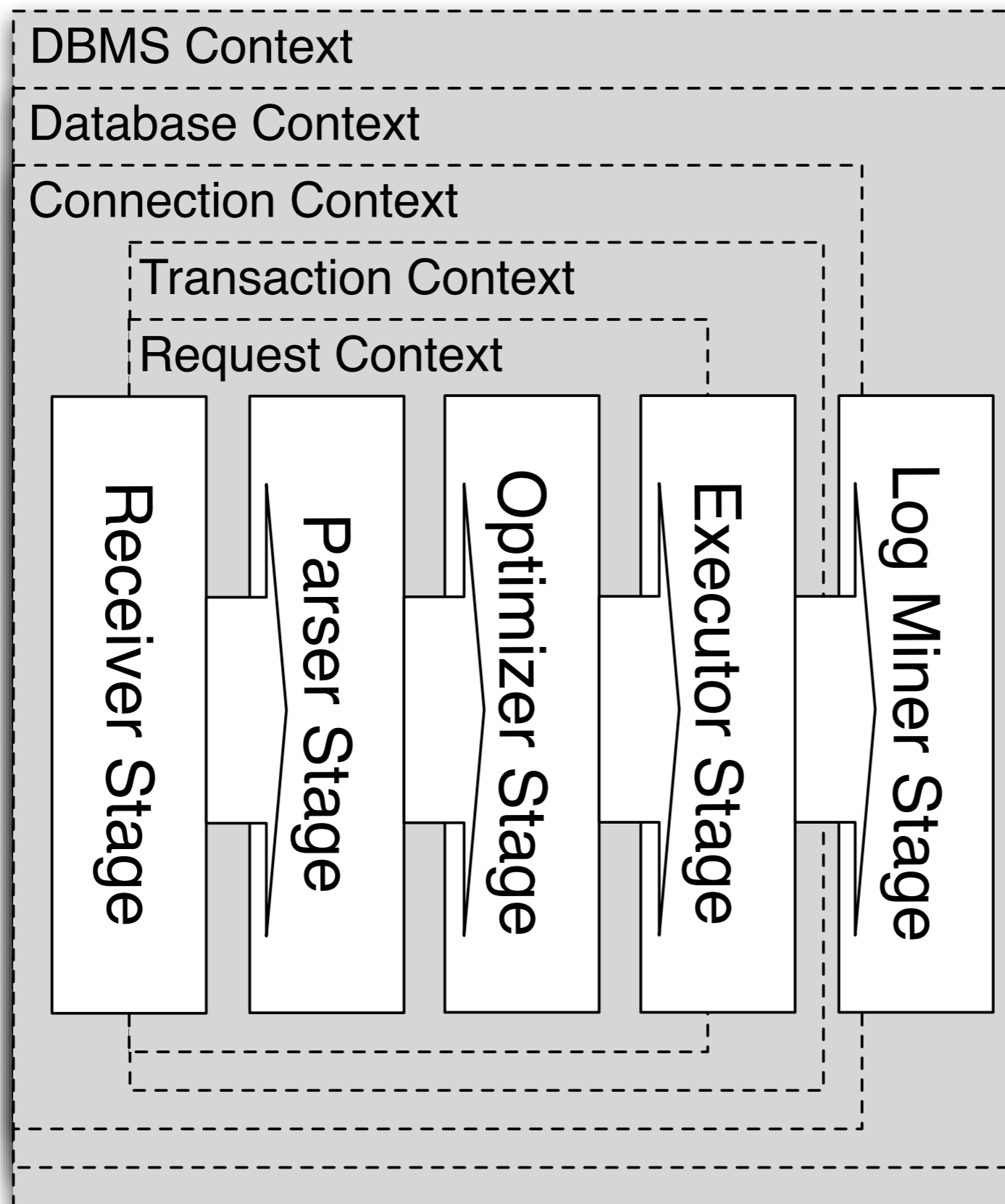
- Lifecycle events
- Object and transaction meta-information
- Statement inspection
- Statement modification
- Write-set and Read-set extraction
- Efficient write-set injection
- Transactional events
- Commit validation
- Predictable deadlock handling
- Result-set injection
- Runtime model
- Configuration storage

Protocols' meta-information stored on the target DB itself.

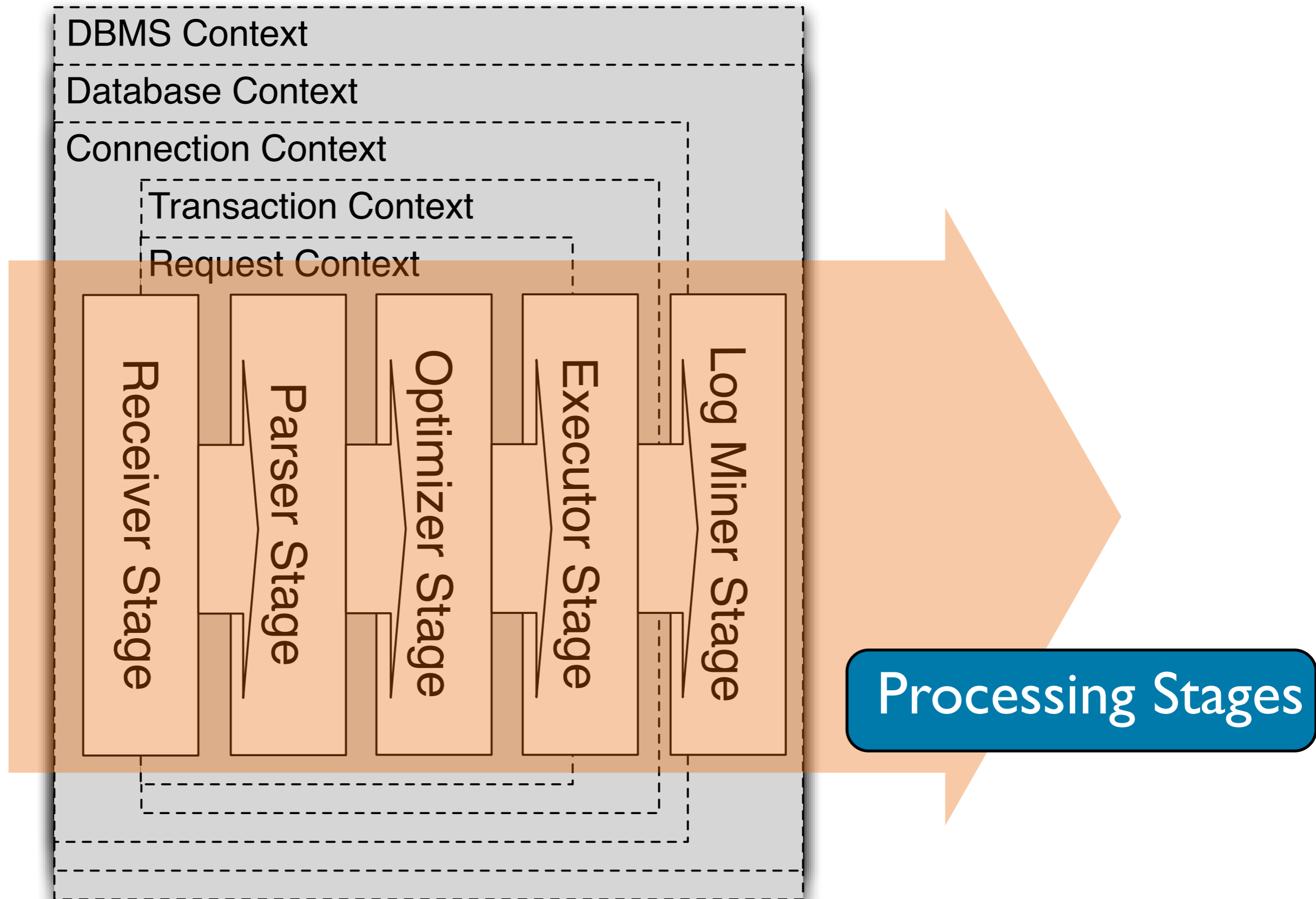
# Generic GORDA architecture



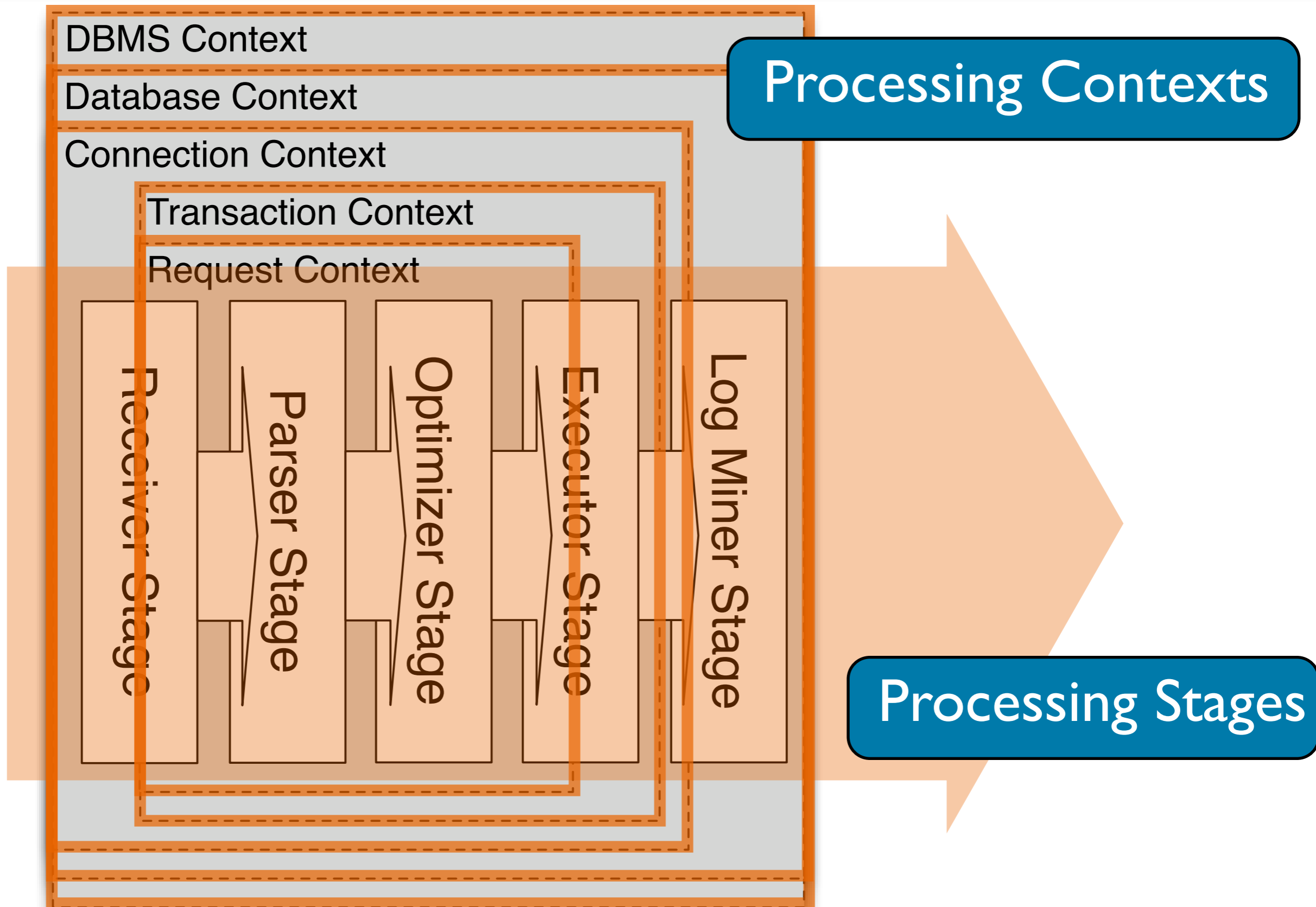
# Reflective interface



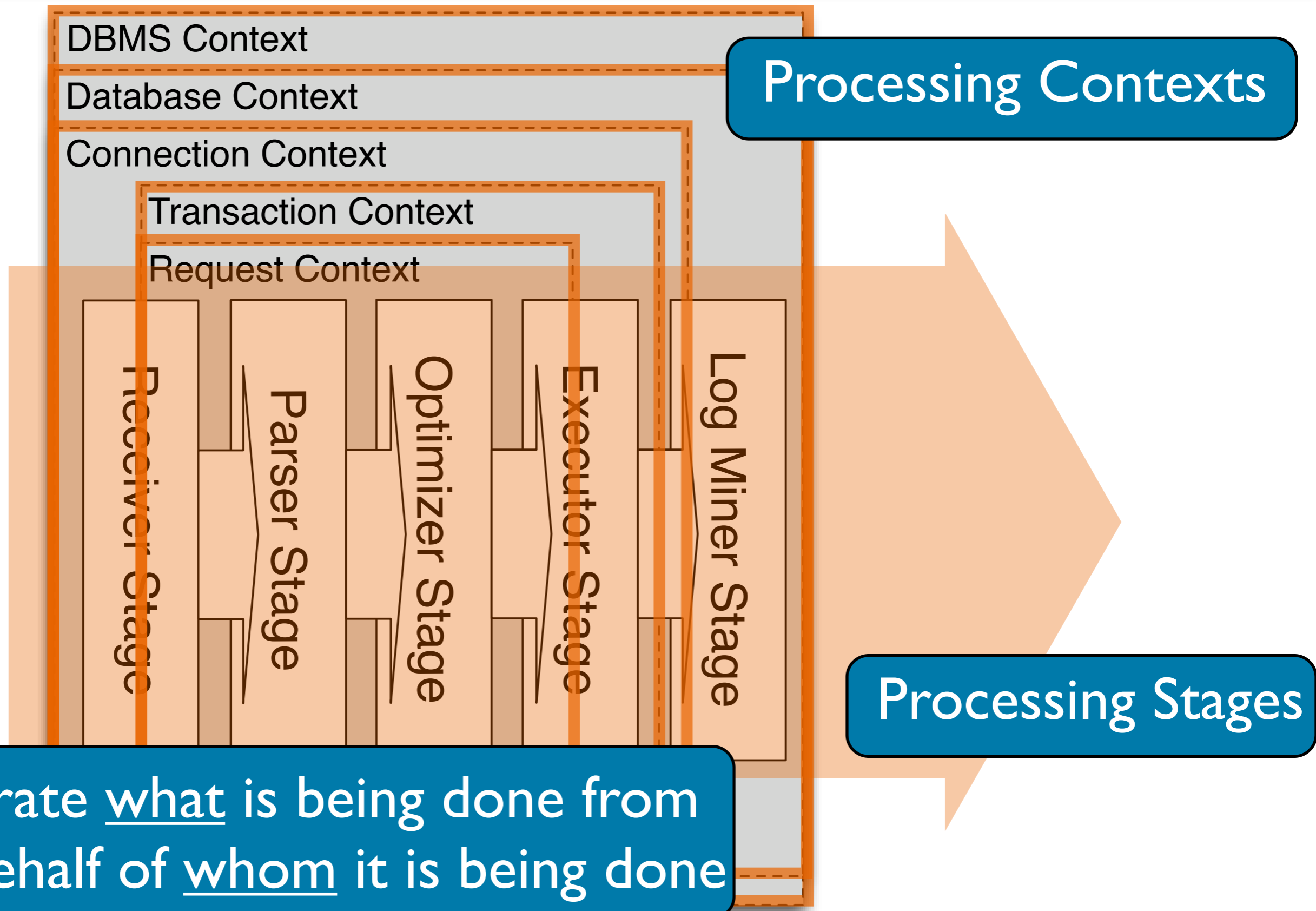
# Reflective interface



# Reflective interface

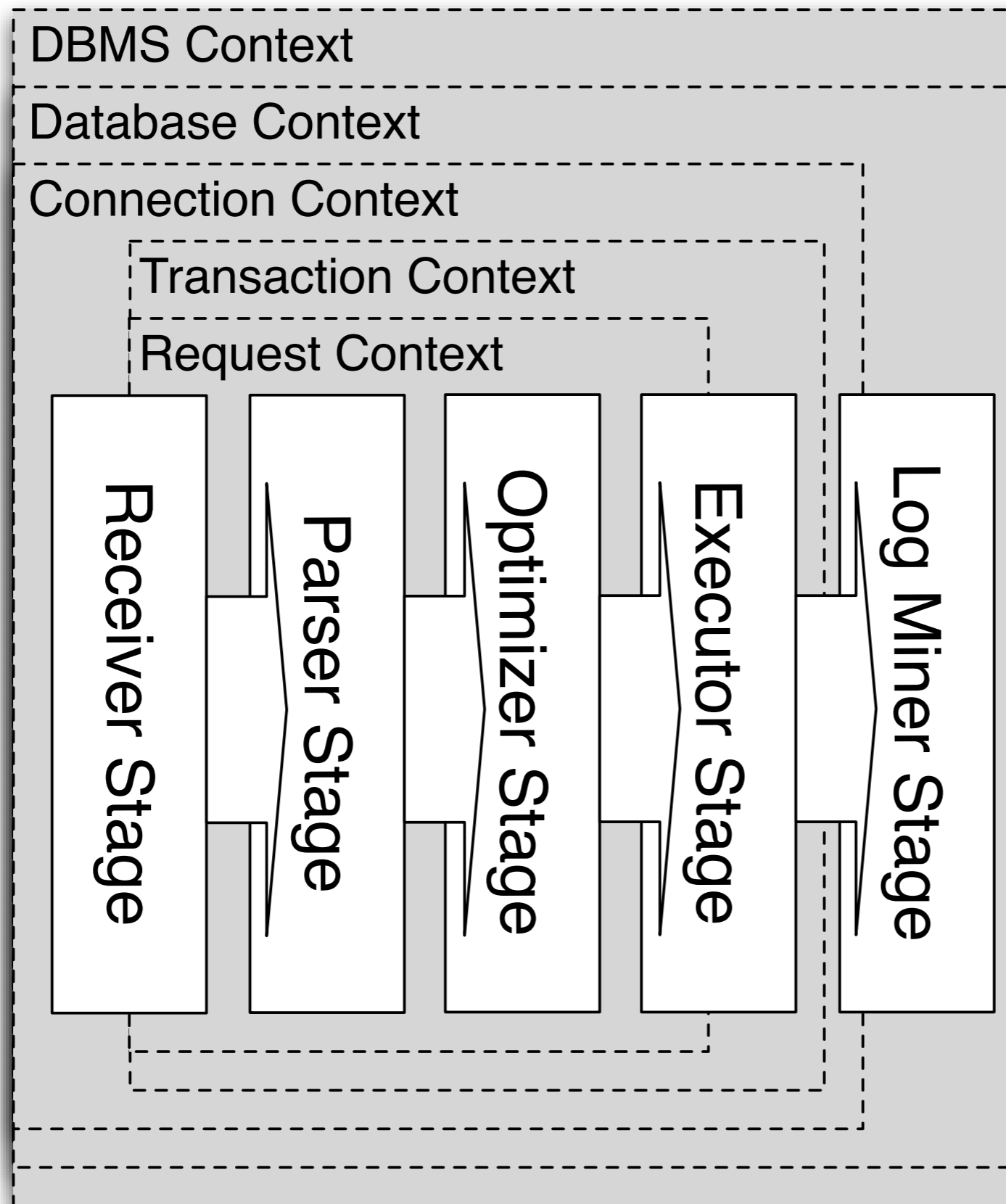


# Reflective interface

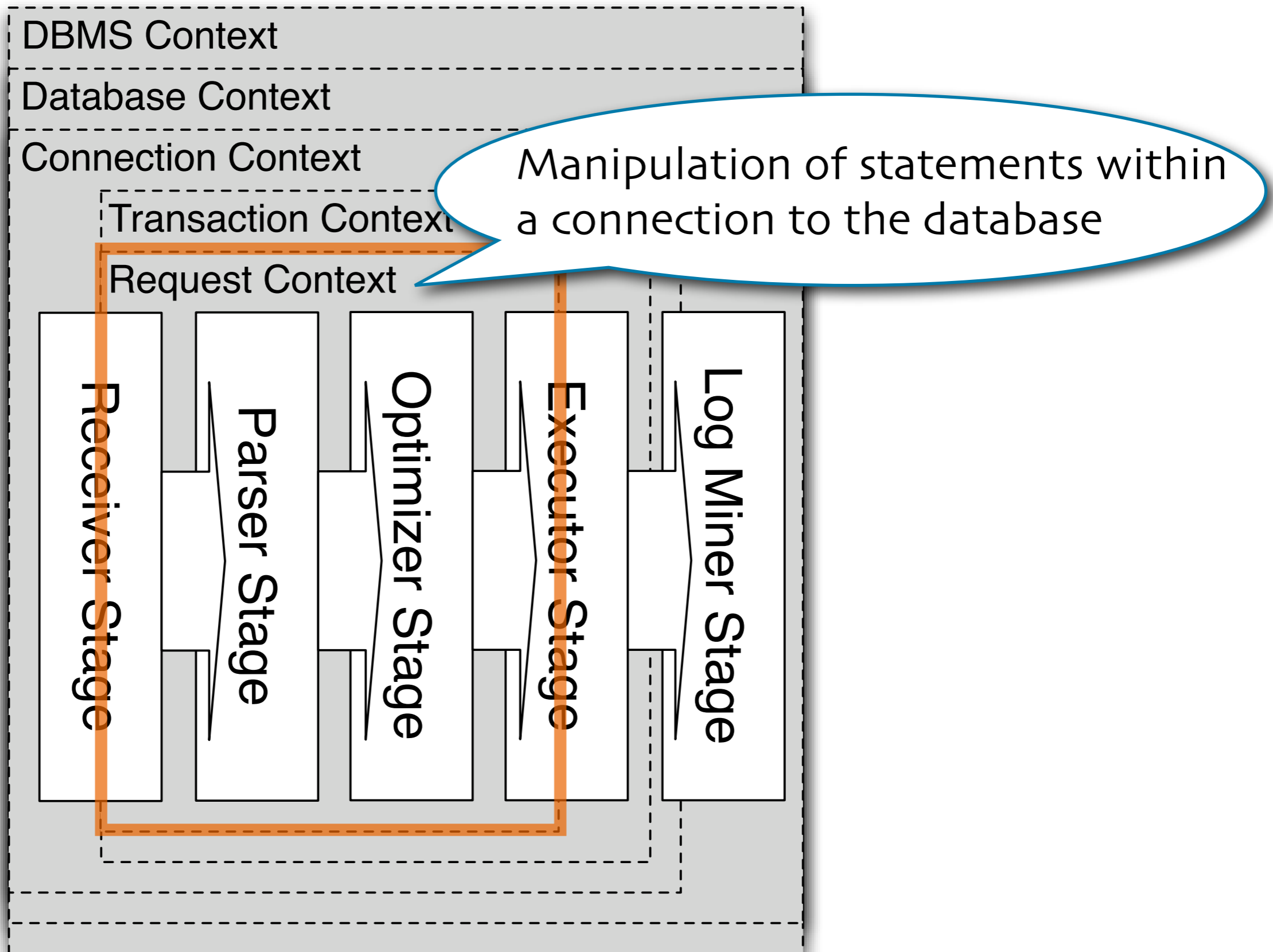


Separate what is being done from on behalf of whom it is being done

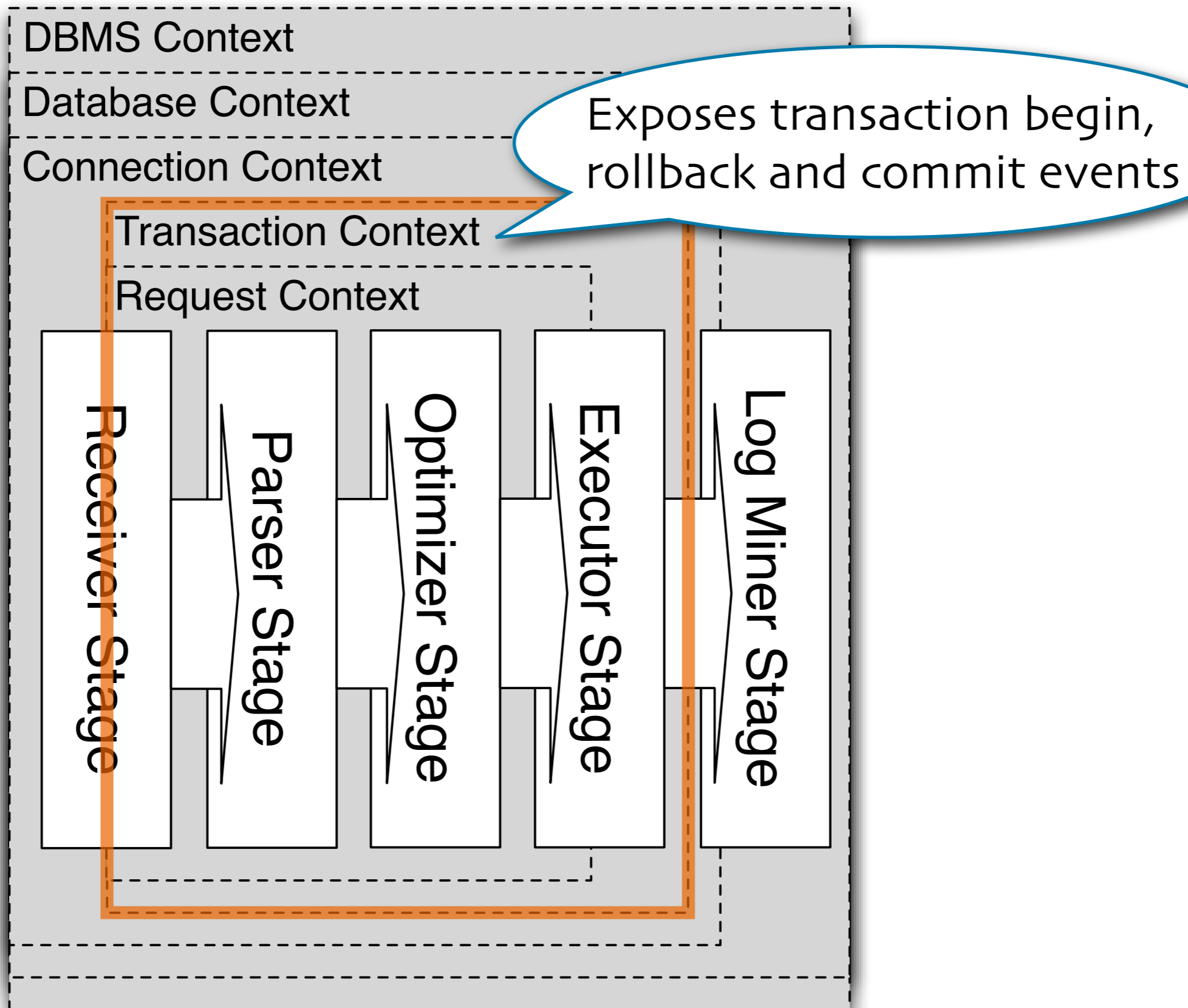
# Processing Stages



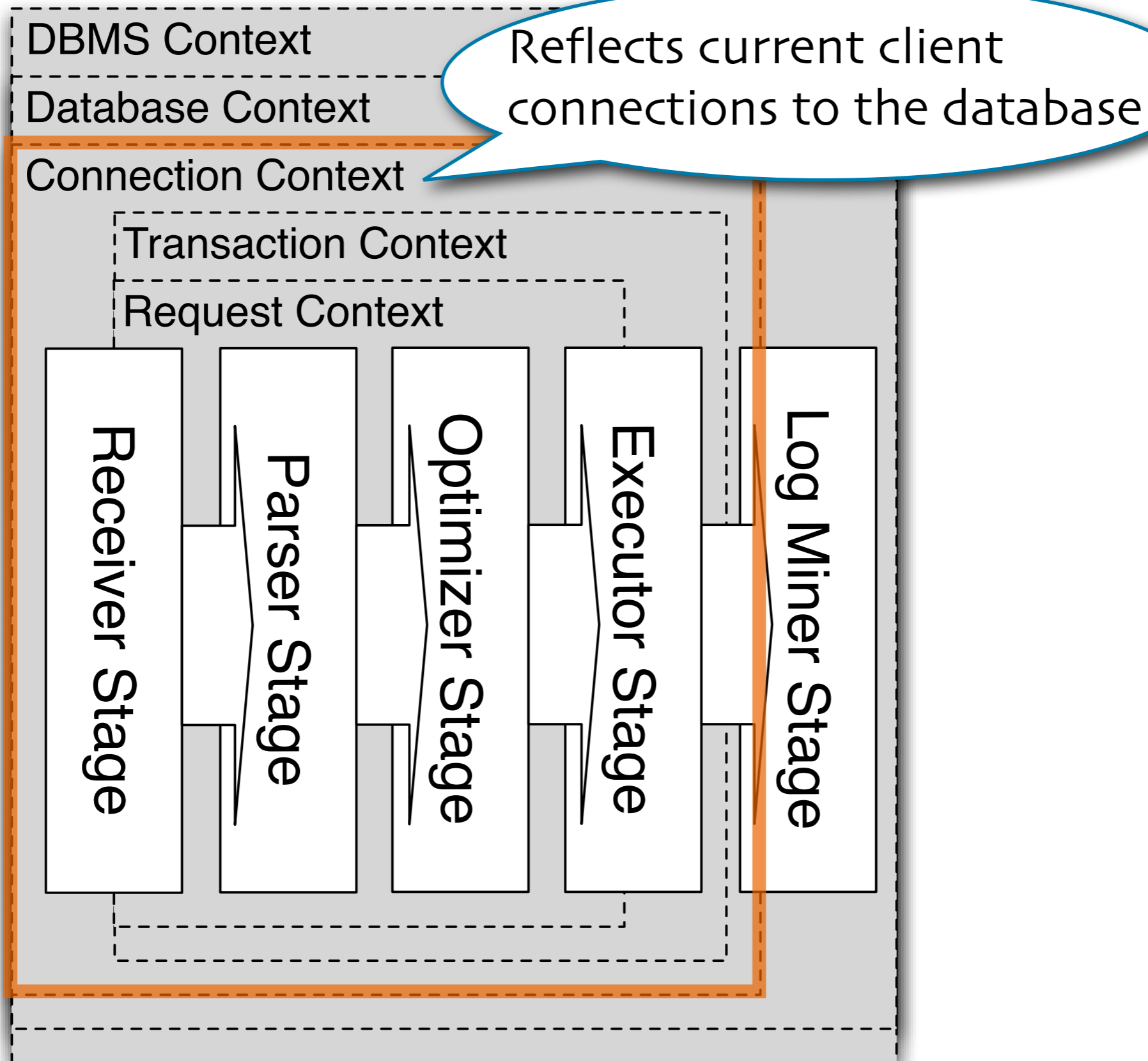
# Processing Stages



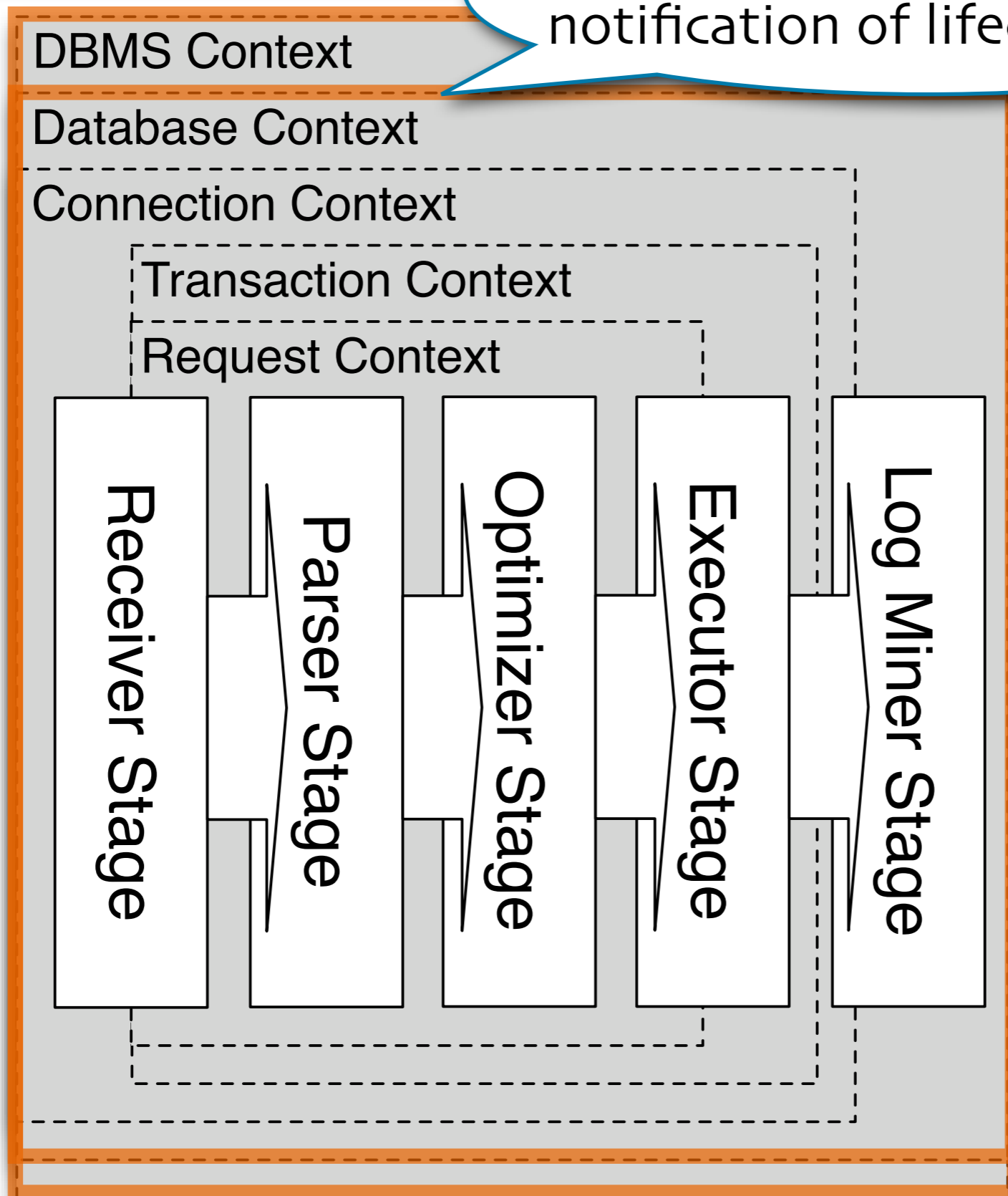
# Processing Stages



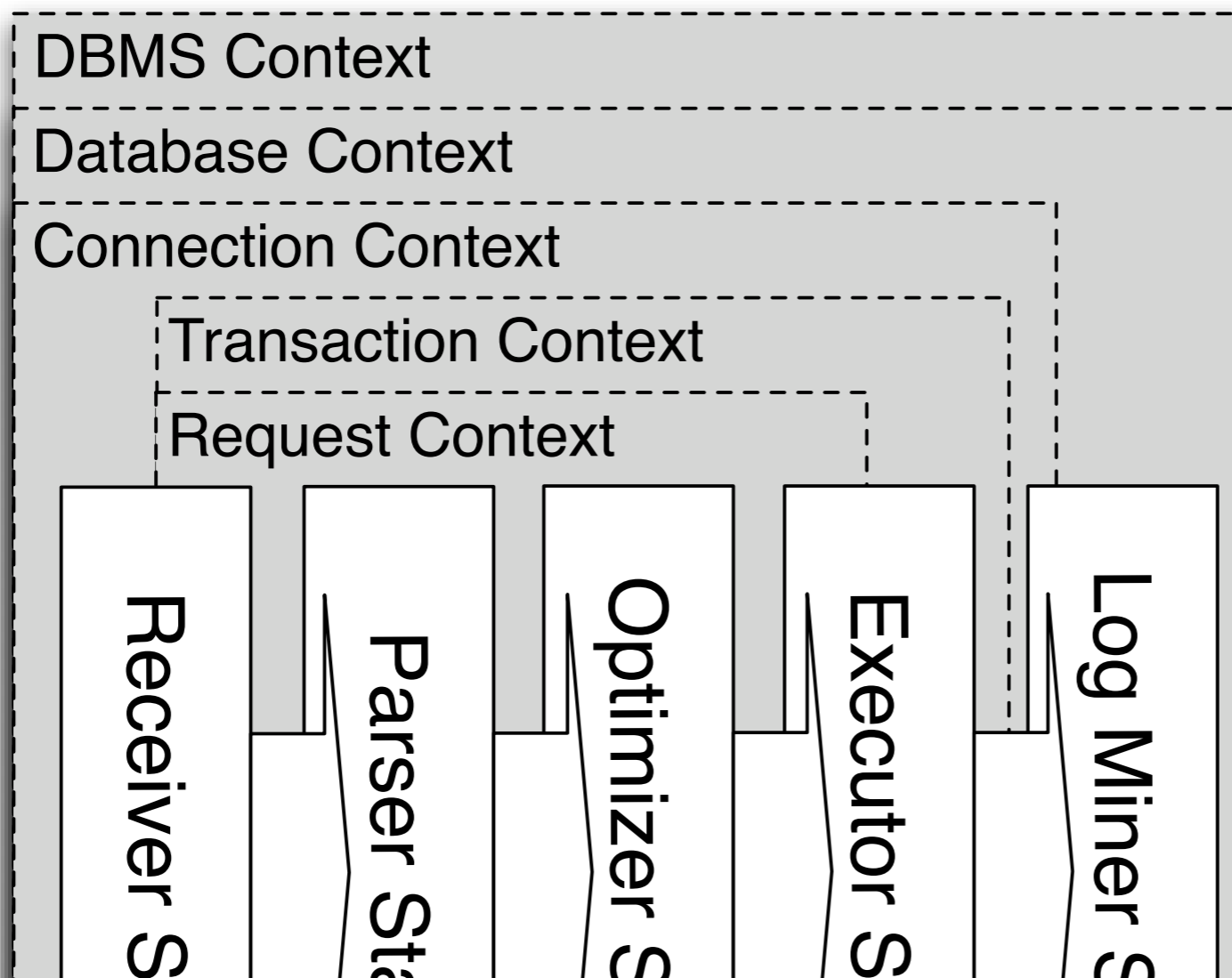
# Processing Stages



Exposes metadata and allows notification of lifecycle events



# Processing Stages



Its useful to group events from single or multiple processing stages, eg.:

- all connections to a database
- current active transaction in a specific connection

# Implementations



# Implementations

- Apache Derby 10.2
  - IBM's Cloudscape and JavaDB
  - All processing contexts already available
  - Current patch size is 0.2% of the code

# Implementations

- Apache Derby 10.2
  - IBM's Cloudscape and JavaDB
  - All processing contexts already available
  - Current patch size is 0.2% of the code
- PostgreSQL 8.1
  - Hybrid approach (part in-core, part loadbale into existing client interfaces)
  - All processing contexts already available
  - Current patch size is 3.6% of the code

# Implementations

- Apache Derby 10.2
  - IBM's Cloudscape and JavaDB
  - All processing contexts already available
  - Current patch size is 0.2% of the code
- PostgreSQL 8.1
  - Hybrid approach (part in-core, part loadbale into existing client interfaces)
  - All processing contexts already available
  - Current patch size is 3.6% of the code
- Sequoia 3.0
  - State-of-the-art JDBC interceptor - Virtual DB
  - Not all processing contexts are available

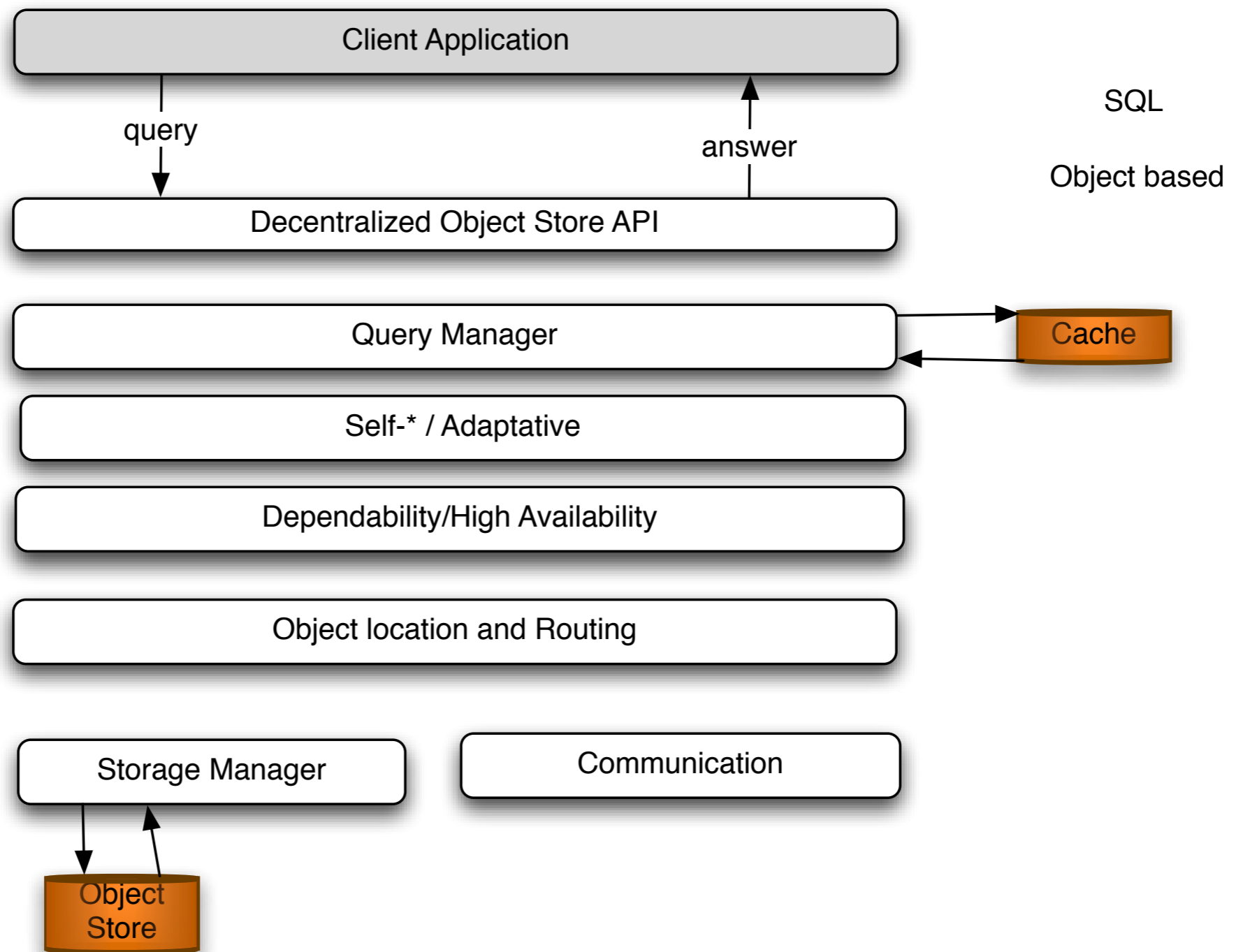
# Scalable Clustering

- Traditional RDBMS aren't designed to massive scale
- Currently, data quadruples every 18 months
- Continuous consistency criteria space requirements
  - BASE model: trade consistency for availability
- Galloping Cloud Computing business
  - "Centralized" storage and processing requires extensive and flexible data partitioning

# Massive data and processing partitioning

- **Balanced P2P environments**
  - Massive scale implies P2P
  - Massive participant numbers imply high churn rates
- **Dependability**
  - Fault tolerance through large scale replications
  - Requires reasonably stable membership
- **Highly “serviceable” system architecture**
  - Massive scale implies high modularity of processing and storage
  - Cloud “centralization” requires flexible orchestration of services

# Clouder: large-scale decentralized object store



# Conclusions

- Database replication and clustering place new demands on DBMS interfaces.
- This is particularly true for approaches relying on GC offering synchronous replication.
- Current attempts (DB kernel patching, complex DB wrappers) require a large development and support effort, cause unavoidable overhead, and reduce portability.
- GORDA proposed an interface to attach replication protocols and DB designed to support a variety of DBMS, protocols and implementation strategies.
- The Cloud Computing model requires nuclear modules to become “serviceable” and a flexible and scalable orchestration