McGill

School of Computer Science

Replication in Multi-tier Architectures

Bettina Kemme
McGill University
Montreal, Canada

What is a client/server system?

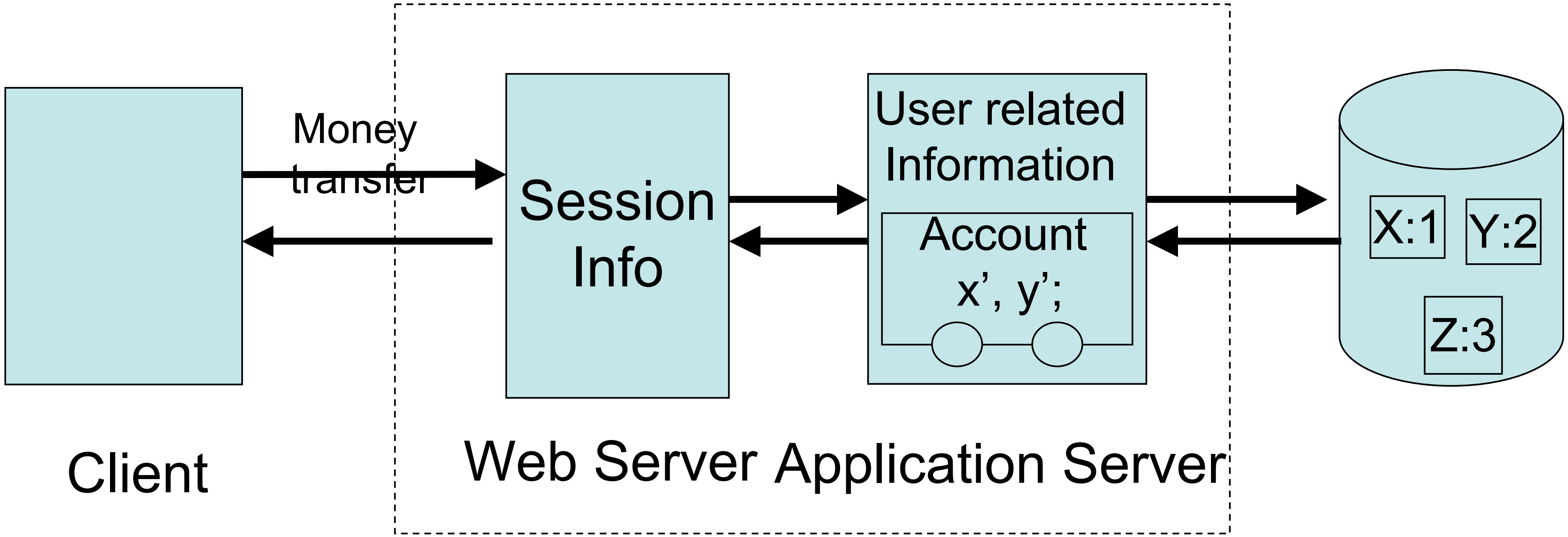
- From Client perspective, there is one server
- Server provides service that can be called

Client

Server

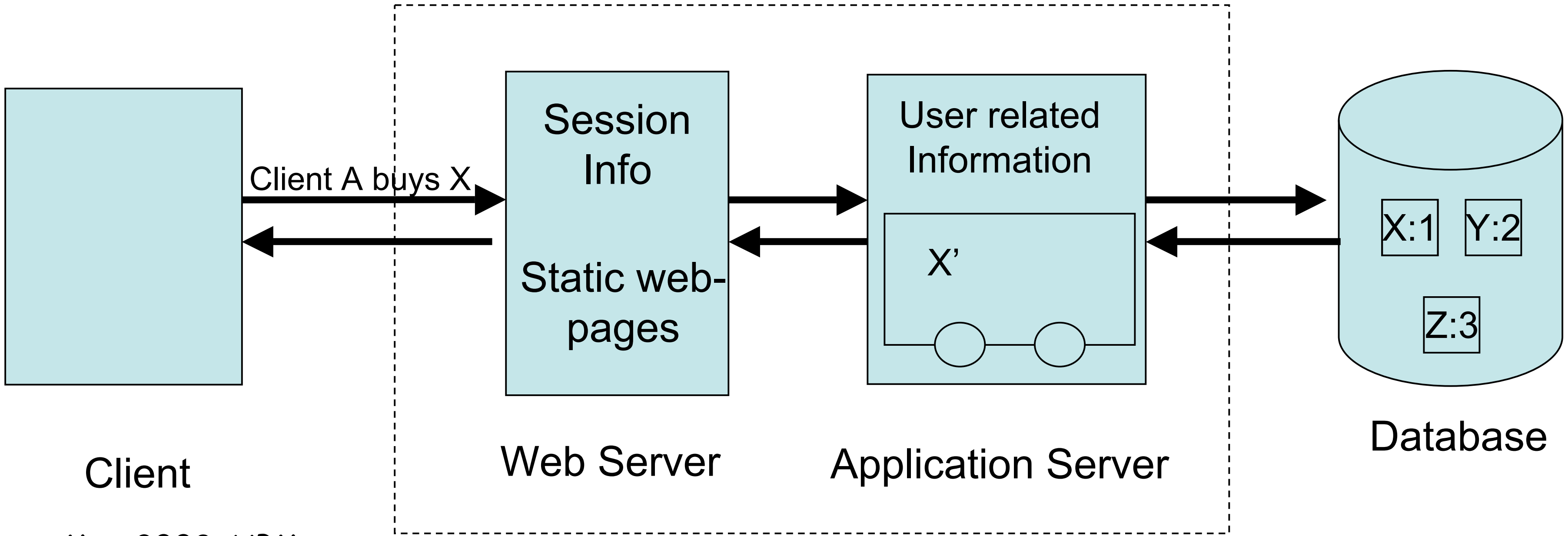
Multi-tier Architecture

- Server itself consists actually of several servers or components
  - each has different functionality



Multi-tier Architecture (contd)

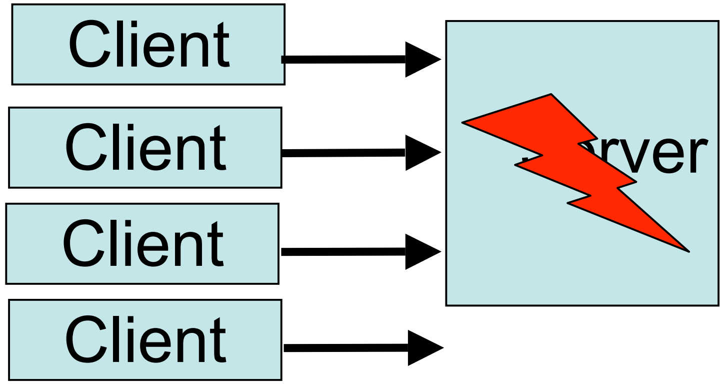
- Front tier: browser, application programs, web-service clients
- Middle-tier
  - Web server (WS): presentation logic
  - Application Server (AS): business logic
- Backend tier (database): persistent data



Multi-tier Architecture (contd.)

- Used widely in most enterprise applications.
- Central role for Web applications, especially for e-commerce.
- WS / Application Server
  - Most common framework:
    - J2EE 1.4 --> Java EE 5 platform
  - http://www.theserverside.com compares 34 AS products
    - 22 have J2EE license => Java EE
    - big players: BEA WebLogic, IBM WebSphere and many others
    - open source: Jboss, JOnAs
  - differentiation:
    - scalability, high availability, ease of use, application integration, extensions
  - Application server markets are expected to reach $5.2 billion by 2009 (http://www.researchandmarkets.com/reports/c7868/)
- Database:
  - Well established for a long time
  - Few big players: Oracle, IBM, Microsoft SQL Server PostgreSQL, MySQL

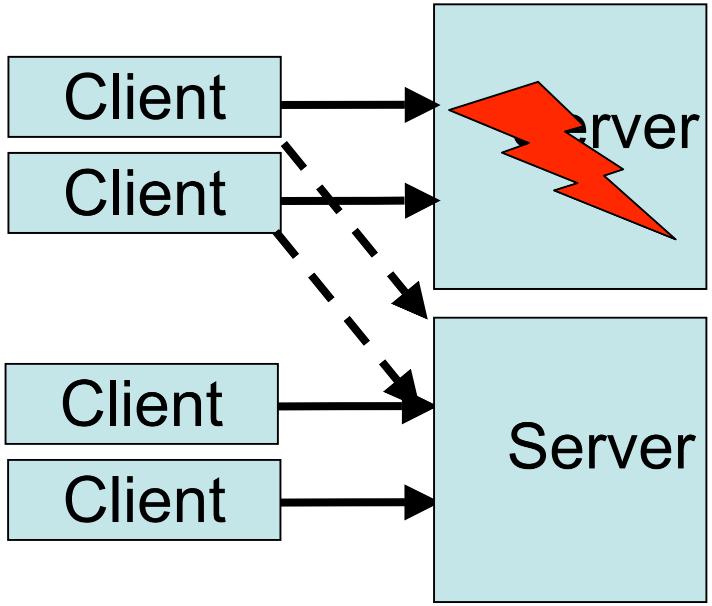
Problem



- Component can become
  - bottleneck
  - single point of failure

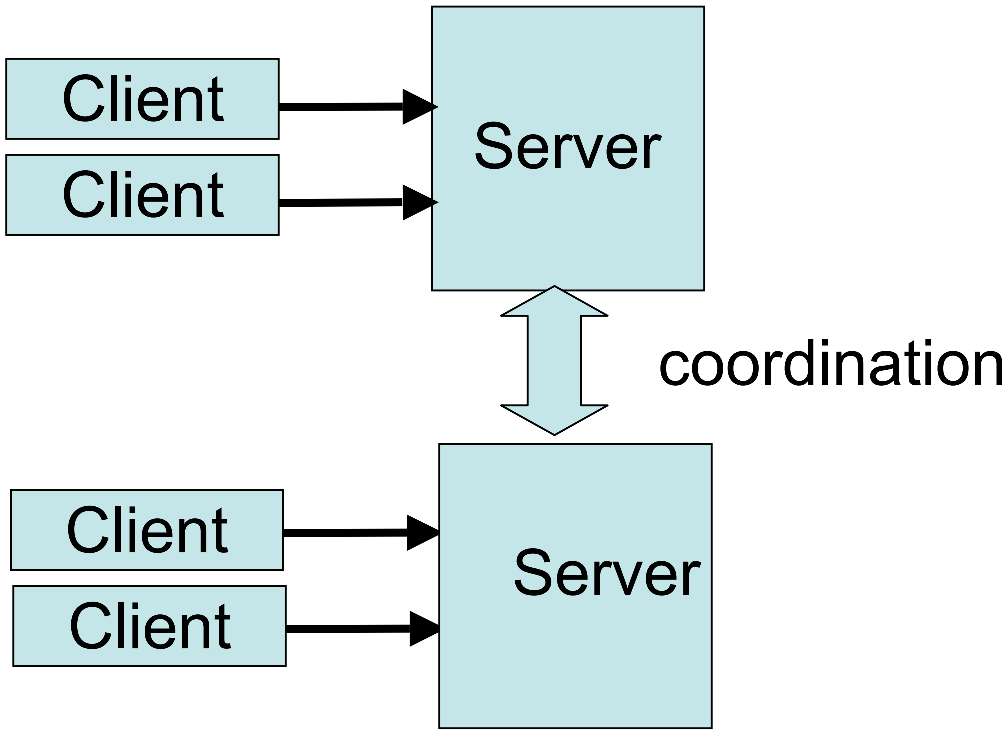
Clusters

- replicate component
  - process and/or data



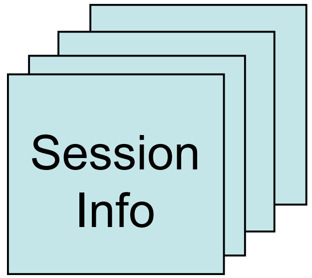
- distribute load over replicated components in cluster
- in case of crash, failover clients to other replica

Added Complexity

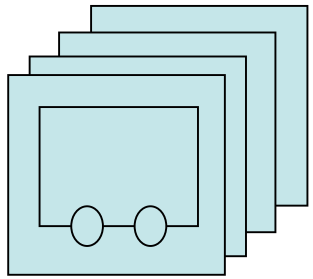


Challenges

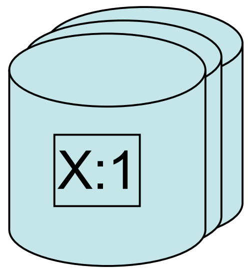
Replica control: keep data copies consistent



- Many physical copies should appear as one logical copy
  - Distribution / replication transparent
  - Same semantics as non-replicated system

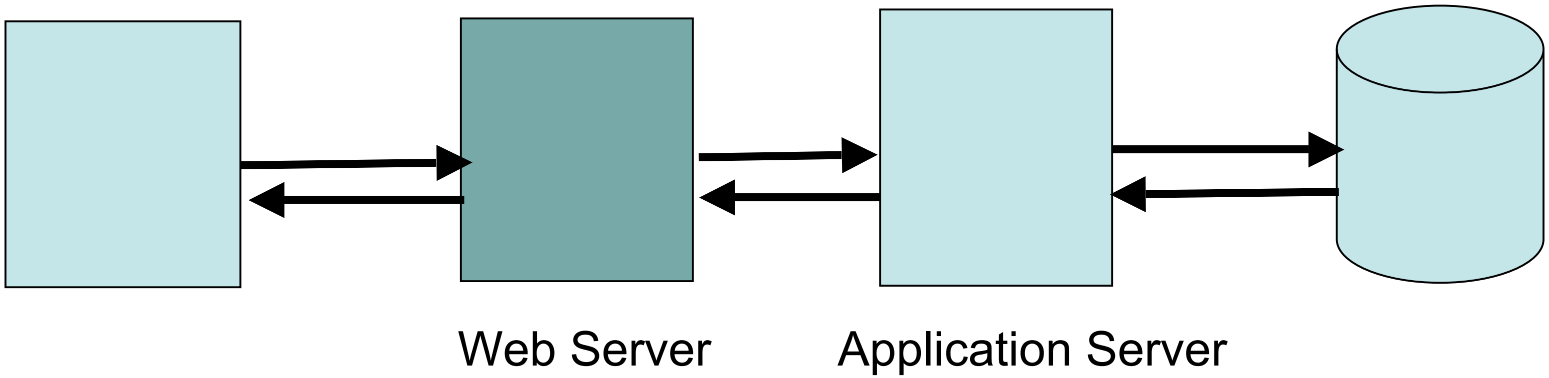


- Failure handling
  - **exactly once** execution
    - (changes appropriate data at the different tiers)
  - Transparent Failover



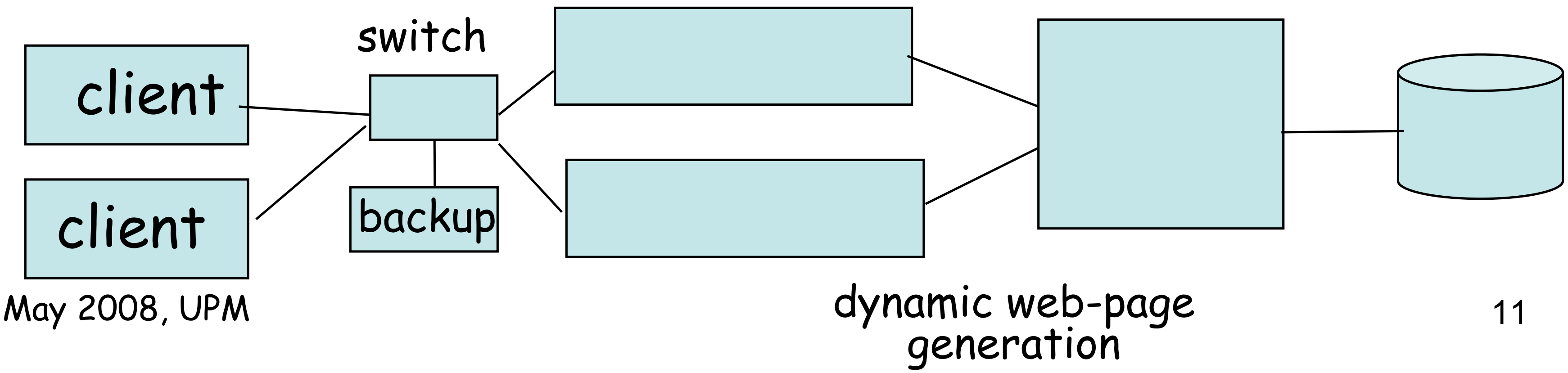
- Online Reconfiguration
  - New replicas need data
- Load-Balancing and Provisioning
- Cooperation between tiers

WS replication

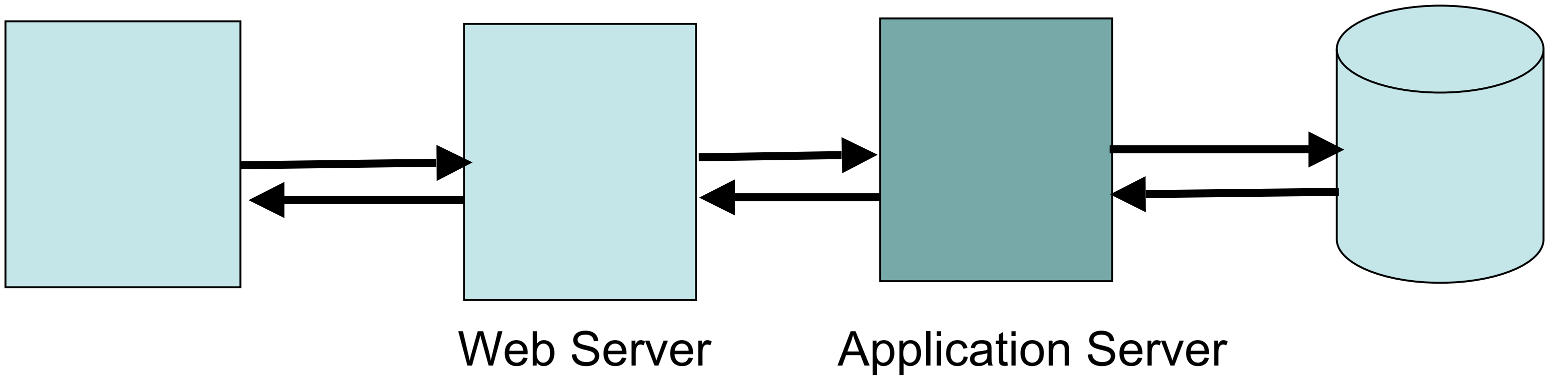


Replication of the WS

- Service and static web-pages replicated
  - scalability and fault-tolerance easily achieved
  - load distribution mechanisms:
    - Stateless: round robin, random…
    - Stateful:
      - send next request to the least loaded server most of the time
      - Cluster request types to exploit cache at web-server
    - Sessions



Replication of the AS tier



Data Replication

- From user perspective there is one logical copy of each data item
- Users submit operations against logical copies
- these operations must be translated into operations against one, some, or all physical copies
- Nearly all existing approaches follow a ROWA(A) approach:
  - Read-one-write-all-(available)
  - Update has to be (eventually) executed at all replicas to keep them consistent
  - Read can be performed at one replica

Data Consistency

- Strong consistency
  - All available copies of an object have the same value at the end of the execution of an update request
  - Clients always read latest versions of data
  - High overhead
  - Tricky if crashes and network partitions
- weak consistency
  - temporal divergence allowed
  - *eventual consistency*
    - if update activity ceases, then all copies of a data item converge eventually to the same value
  - Clients might read stale data

5-phase request execution

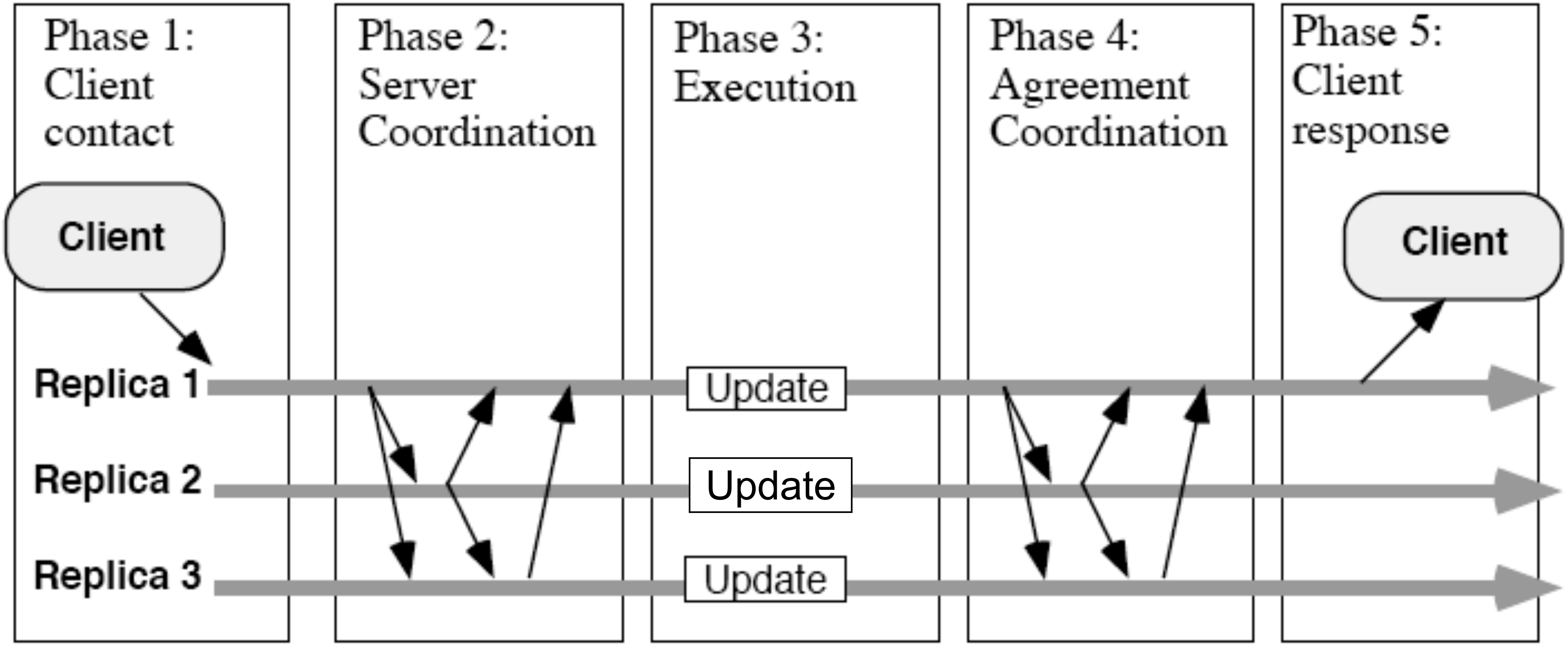
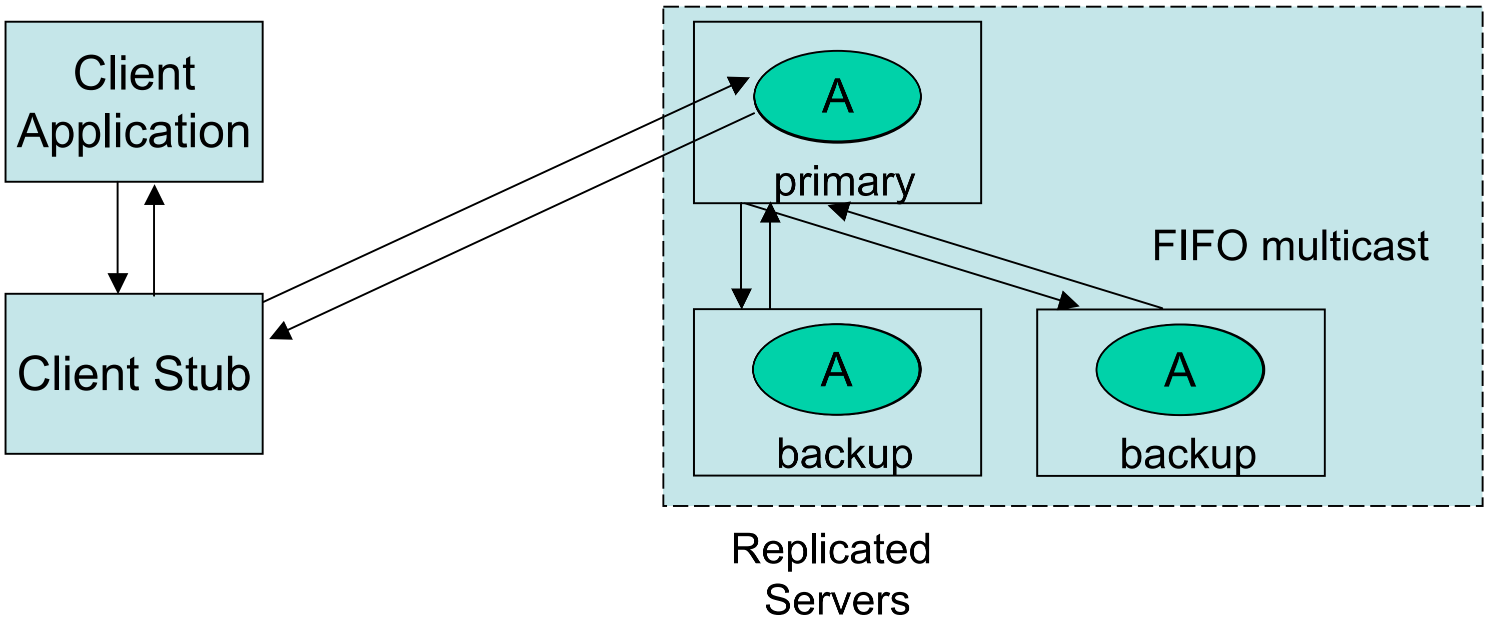


Figure 1. Functional model with the five phases

Object Replication: Fault-tolerance

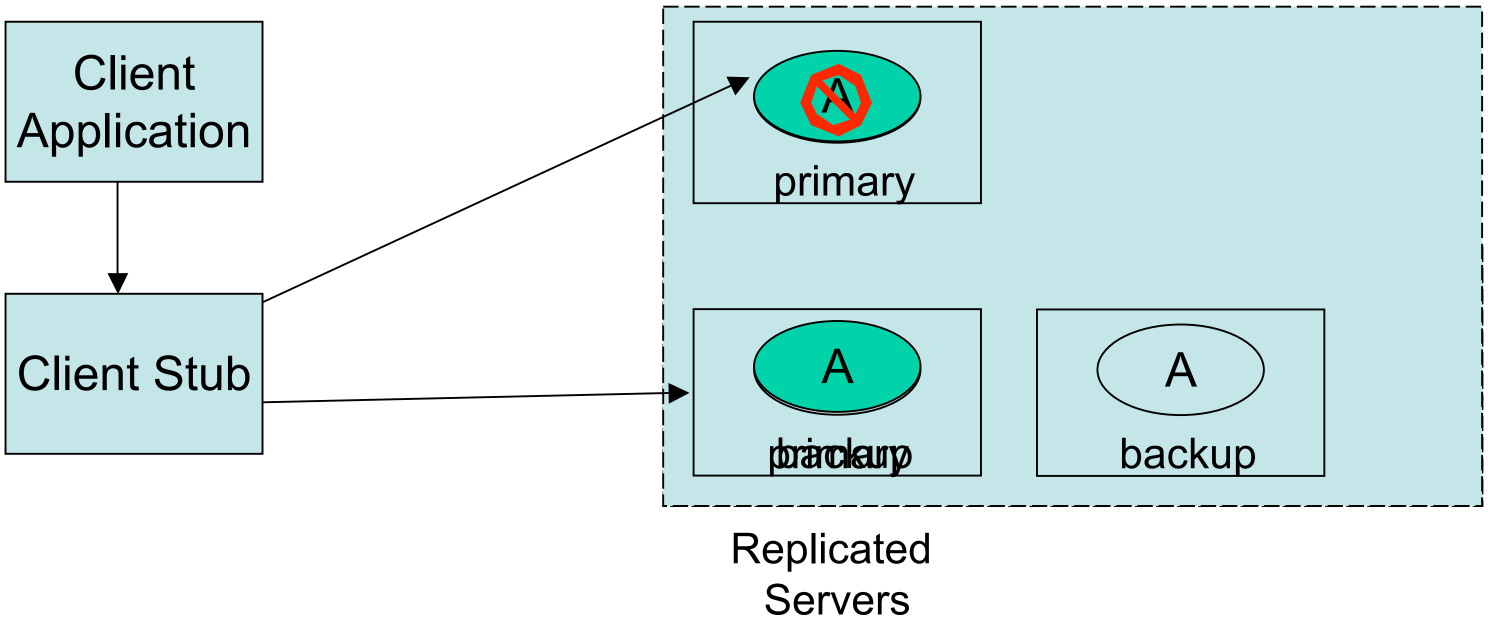
- A large body of research
- implemented in distributed computing environments
  - CORBA developed a standard
    - FT CORBA = Fault-tolerant CORBA
  - similar models for J2EE distributed computing environment
- Often assumes the use of a group communication system
  - multicast
  - group maintenance, failure detection
  - virtual synchrony synchronizes multicast and group changes
- Correctness
  - Replicated System should behave as non-replicated system that has no failures
  - Each request has exactly one "successful" execution
  - Client receives exactly one response (failure transparency)
  - **strong data consistency**: data copies are consistent at the end of request execution
- passive (primary backup) replication vs. active replication

Passive Replication



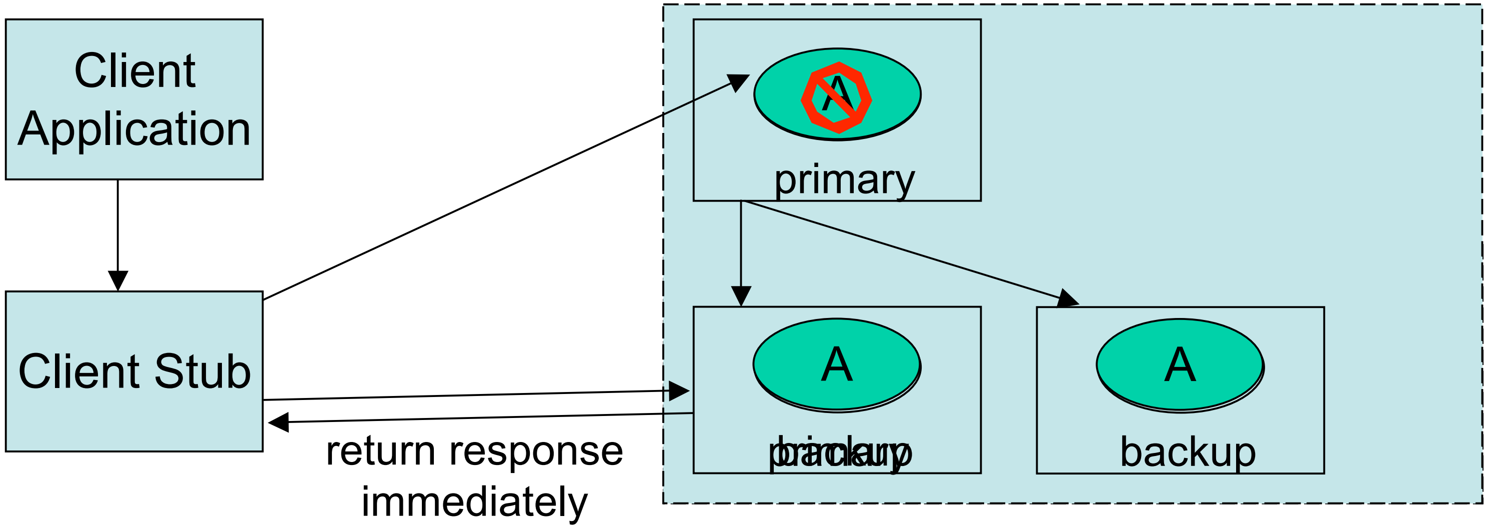
Passive Replication Failures

- Before Update Propagation
  - reexecute on new primary



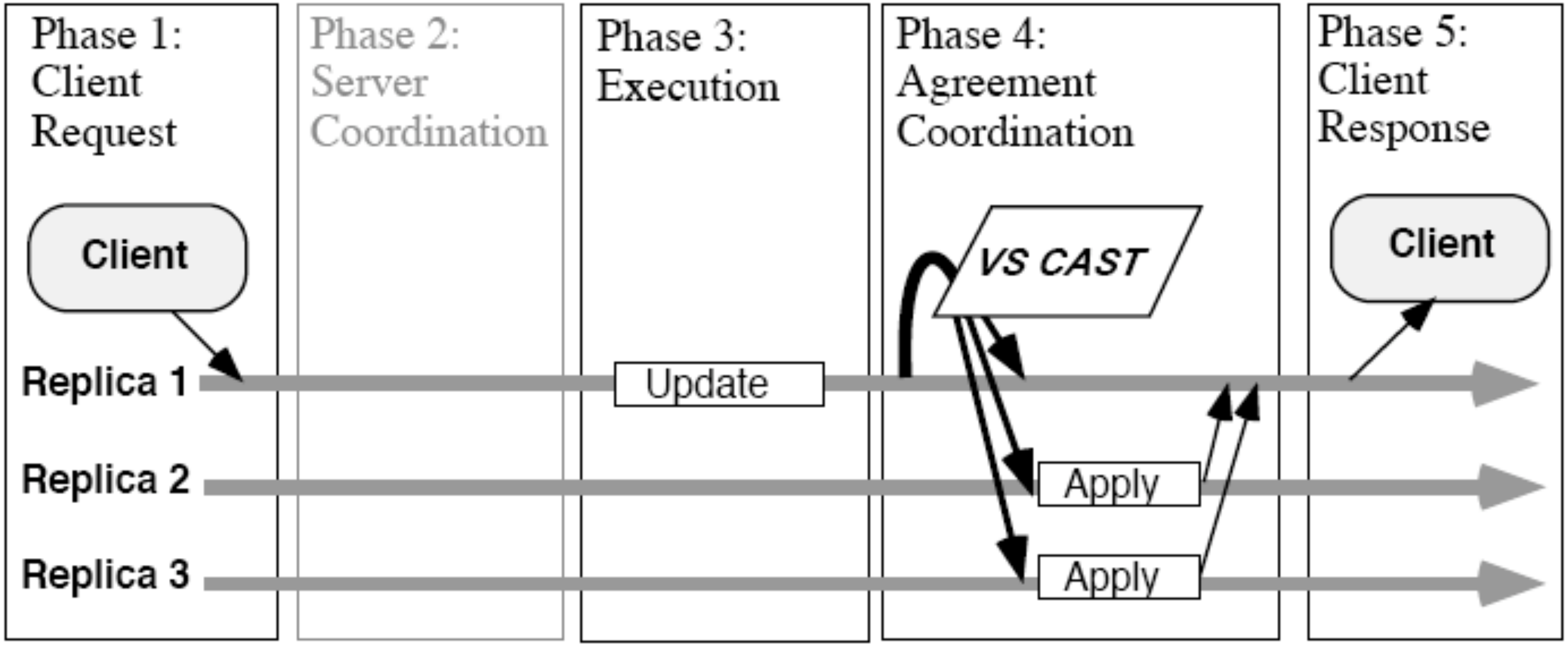
Passive Replication Failures

- After Update Propagation: return result immediately



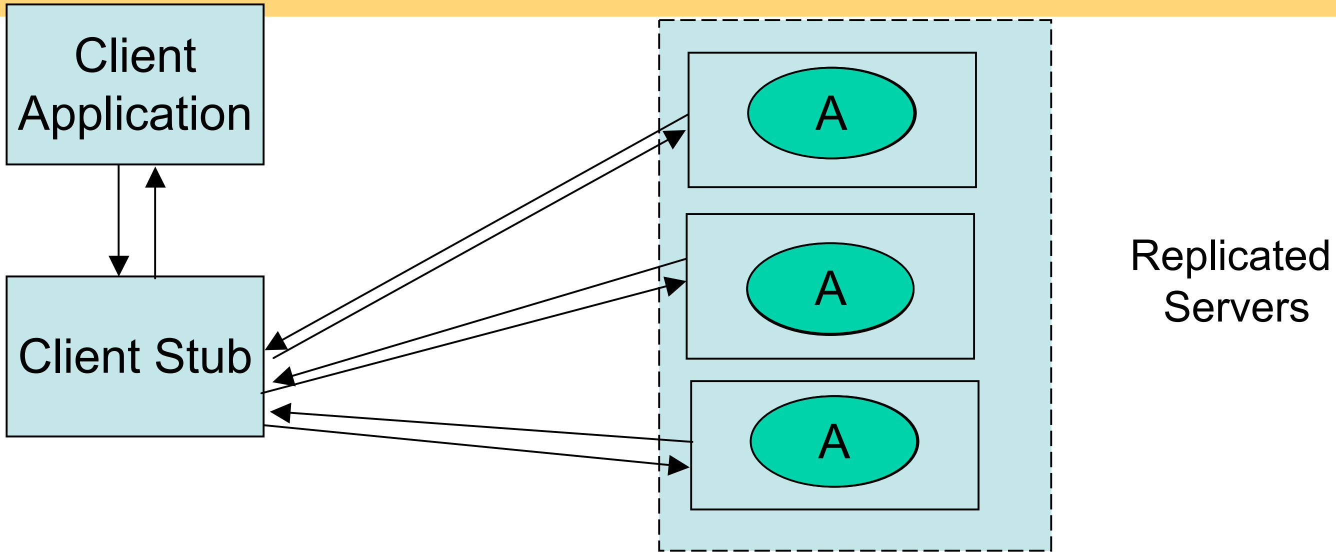
- GCS and virtual synchrony guarantees
  - all or none of the backups have state changes
  - all have same view of who is primary/backup
- Avoiding wrong reexecution
  - request must have unique ids
  - primary must send response with state changes
  - backups must keep responses

Passive Replication



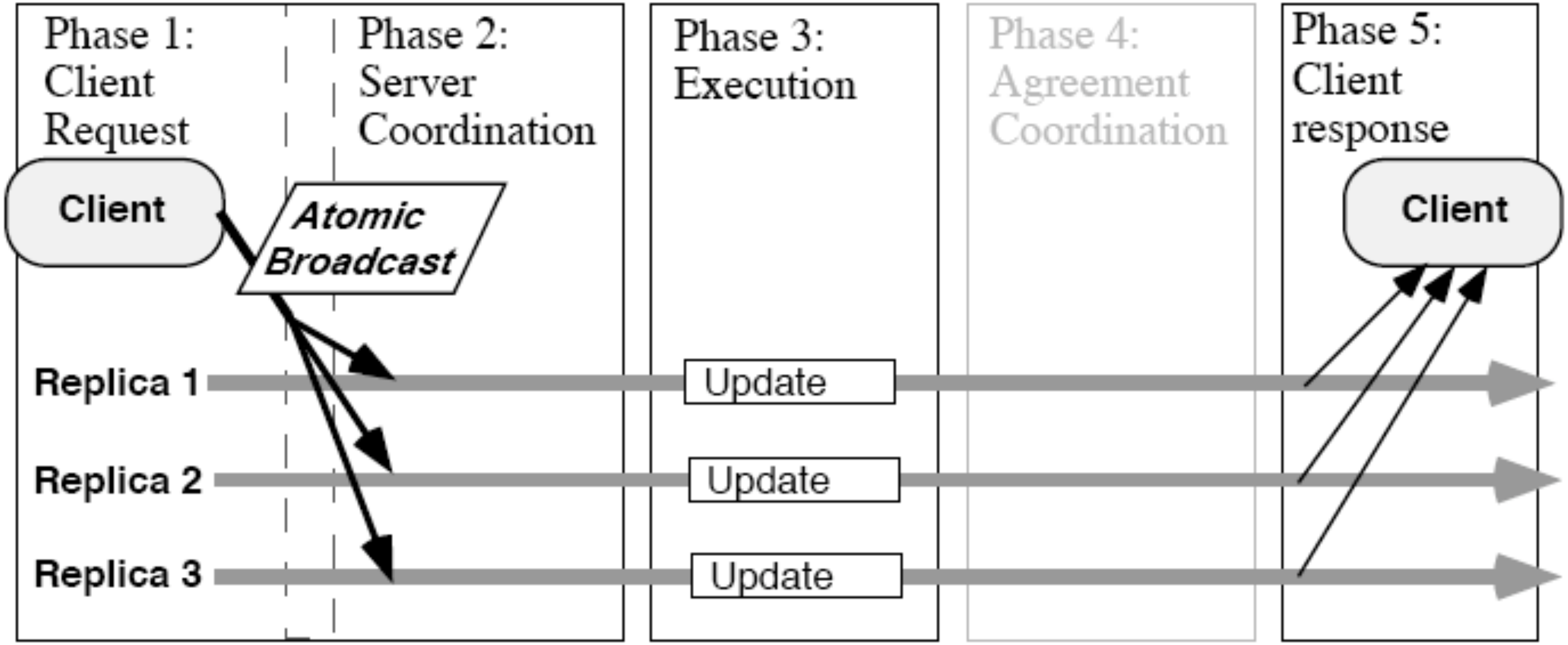
1. The client sends the request to the primary.
2. There is no initial coordination.
3. The primary executes the request.
4. The primary coordinates with the other replicas by sending the update information to the backups.
5. The primary sends the answer to the client.

Active Replication



- Total order multicast
- System only tolerates crash Failures
  - client stub returns to client first response it receives; discards others
- System tolerates Byzantine Failures
  - client stub waits for all responses; returns to clients response that was received by more than half of server replicas

Active Replication



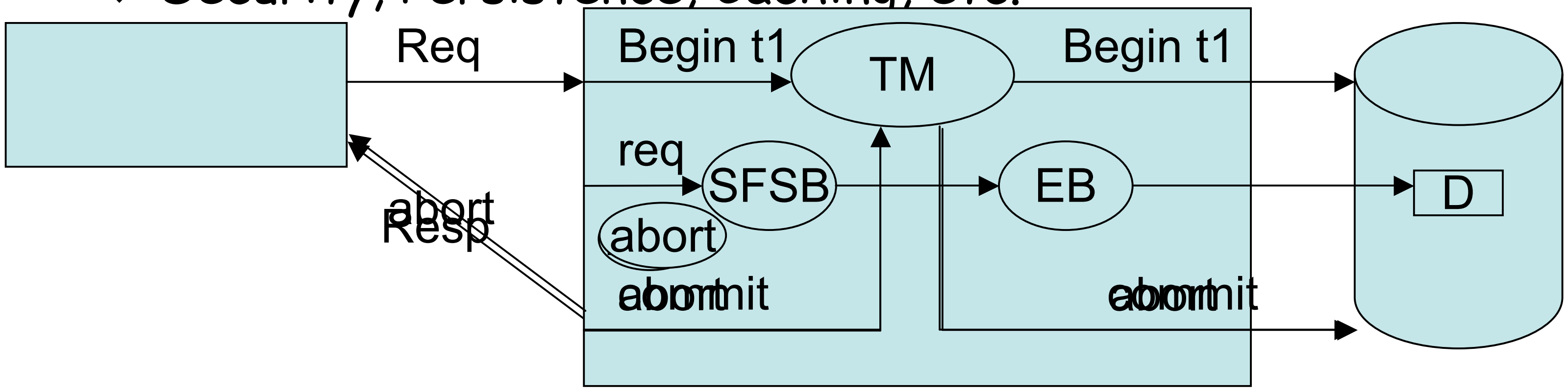
1. The client multicasts request to the servers with total order
2. Server coordination is given by the total order property
3. All replicas execute the request in the order they are delivered.
4. No coordination necessary (Assumption: determinism)
   - All replicas produce the same result
5. All replicas send result to the client; client waits for the first answer

Active vs. Passive Replication

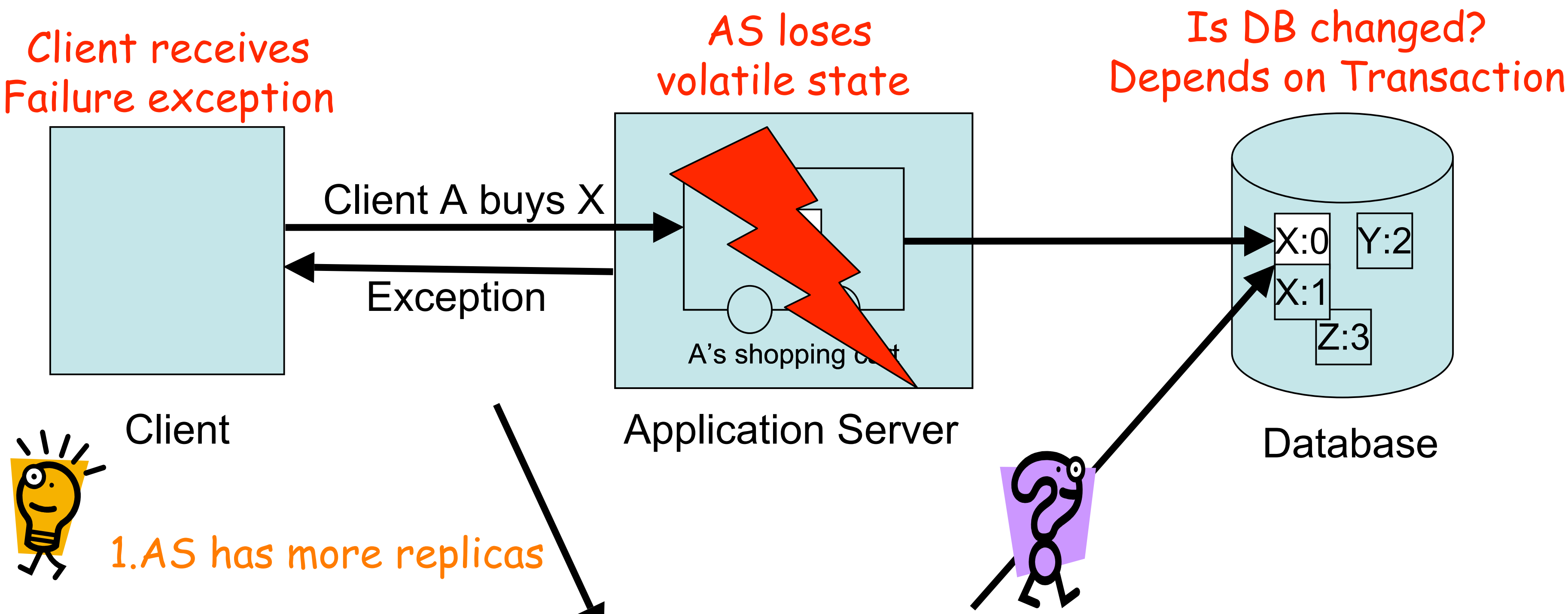
- Determinism
- Execution during normal processing
  - Communication Overhead
  - CPU overhead
  - Complexity
- Termination protocol
- Failure types
- Write / read

AS with DB backend

- J2EE container: runtime environment
- Components: Enterprise JavaBean (EJB)
  - Session Bean (SB):
    - Java class implements business methods (transfer money, …)
    - Stateful bean instance associated with a caller session
  - Entity Bean (EB) (or Entity Object): maps to persistent data
- Services:
  - **Transactions**: all-or-nothing
  - Security, Persistence, Caching, etc.



AS Failures



1.AS has more replicas

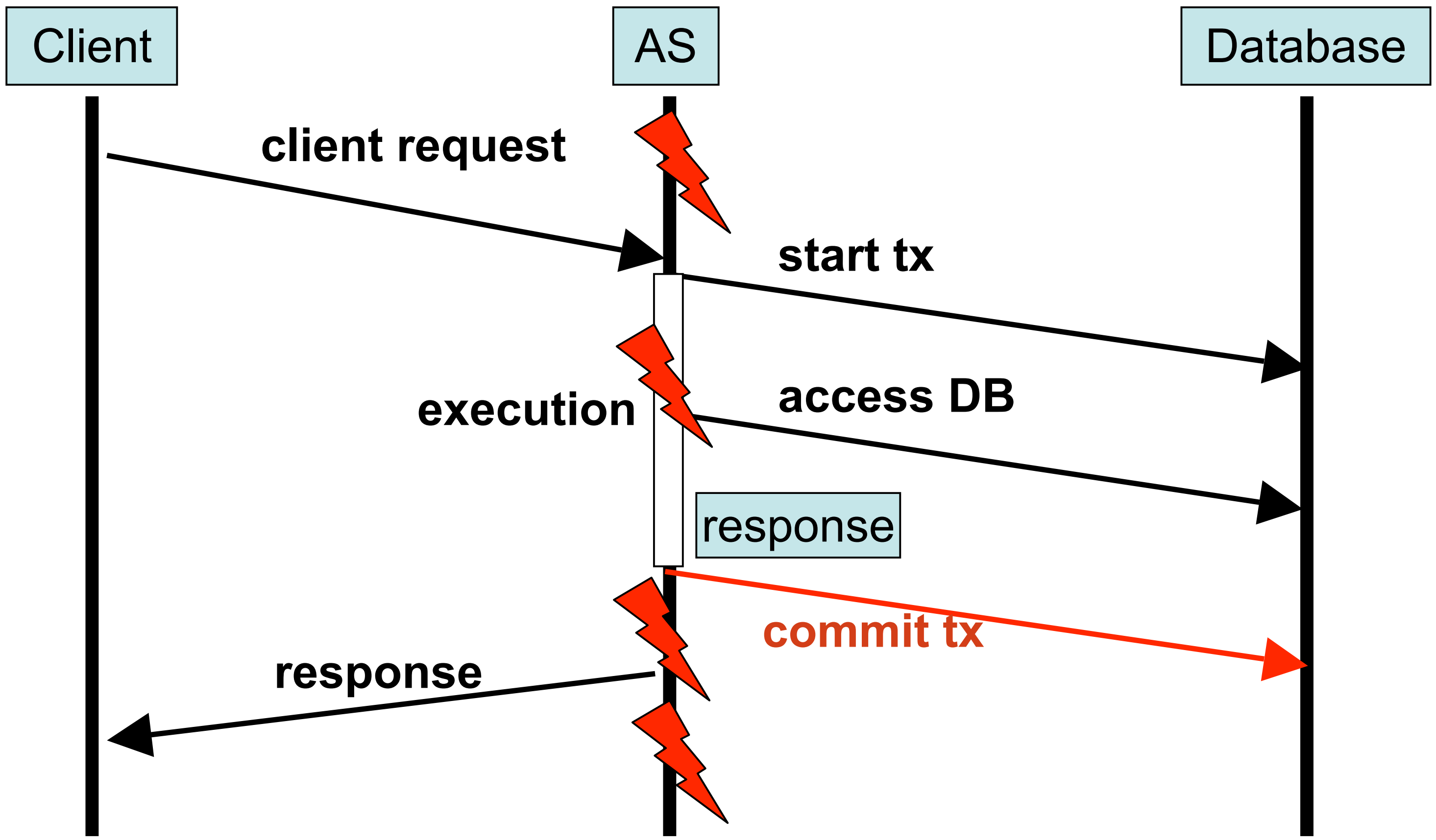
2. Replicate AS's state

3. Re-execute outstanding client requests

Correctness

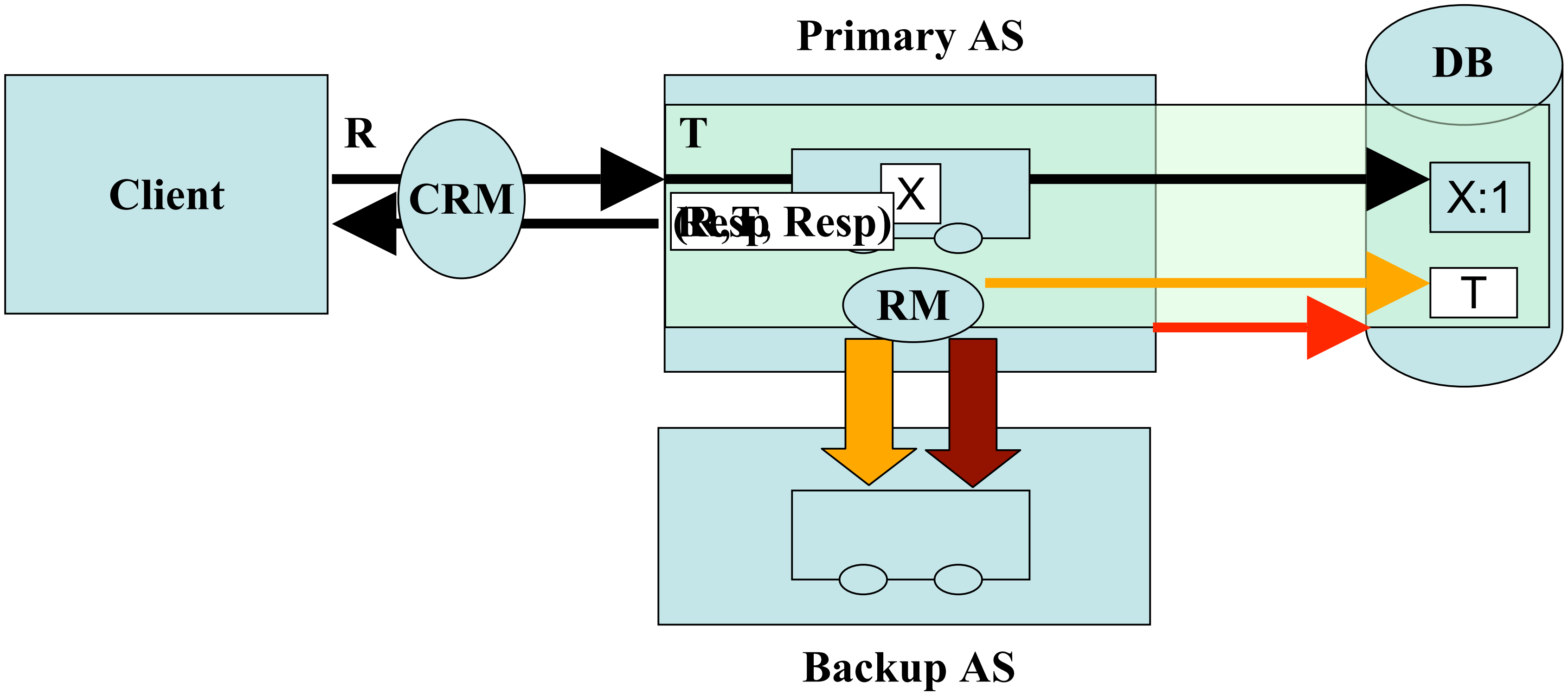
- Replicated System should behave as non-replicated system that has no failures
- Each request has exactly one successful "execution"
  - Client receives exactly one response (failure transparency)
  - Execution represents possible execution in a non-replicated system without failure in regard to
    - Response returned to client
    - State at AS and DBS

1-1 Pattern

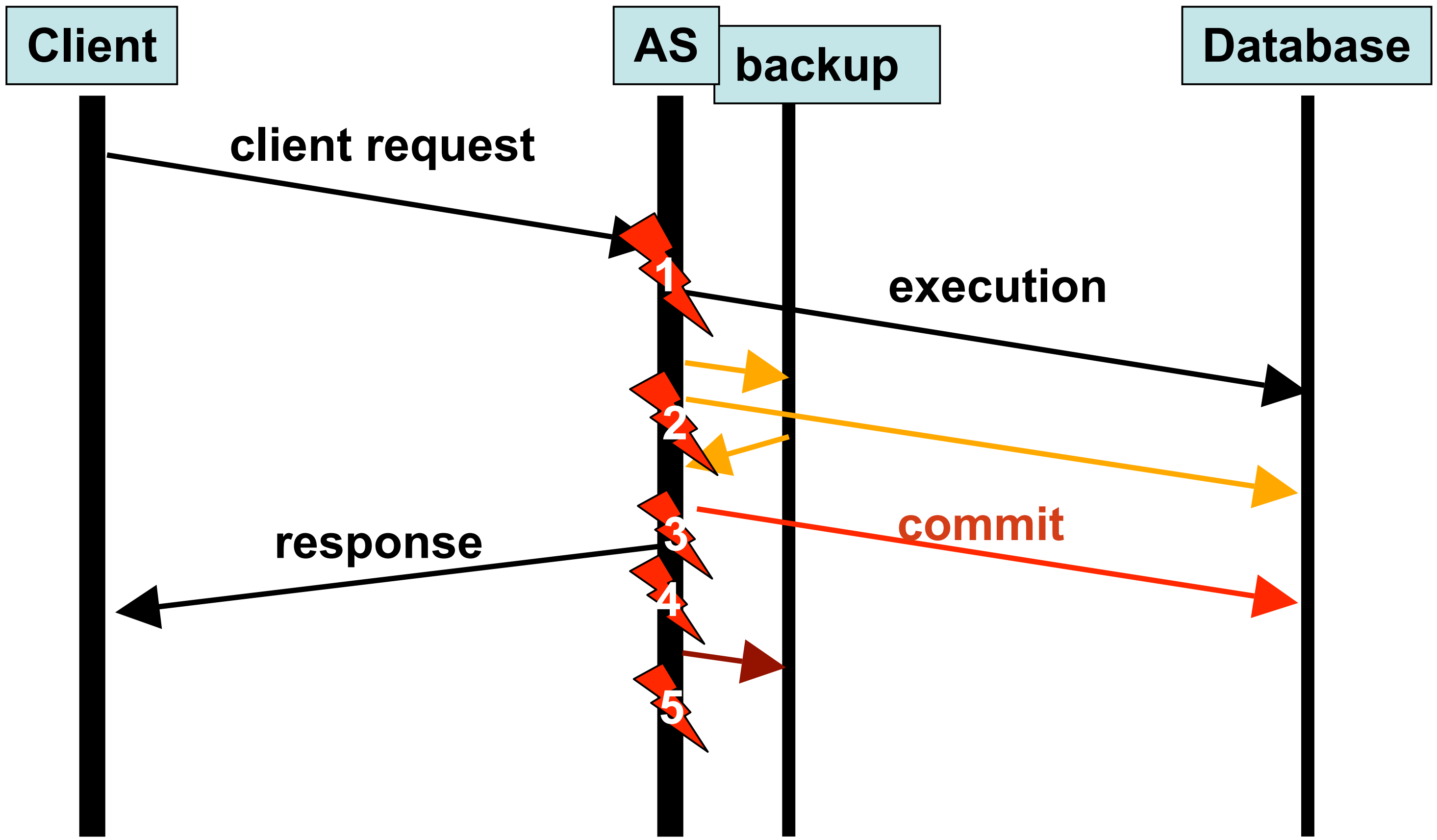

Client
AS
Database
client request
start tx
execution
access DB
response
commit tx
response

1-1 Algorithm

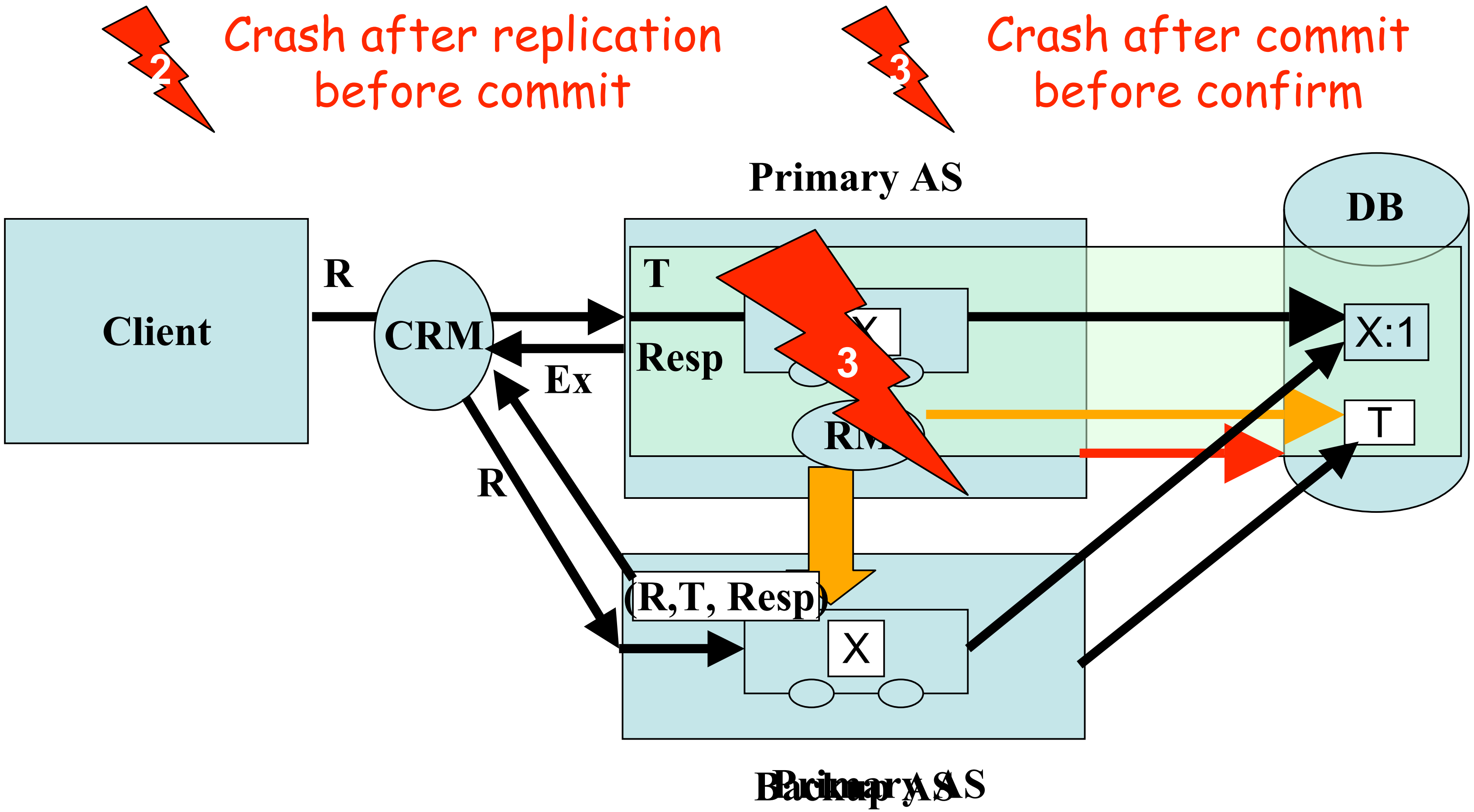
Normal Execution | Replication | Commit | Confirm



Failure Cases



1-1 Algorithm (II)



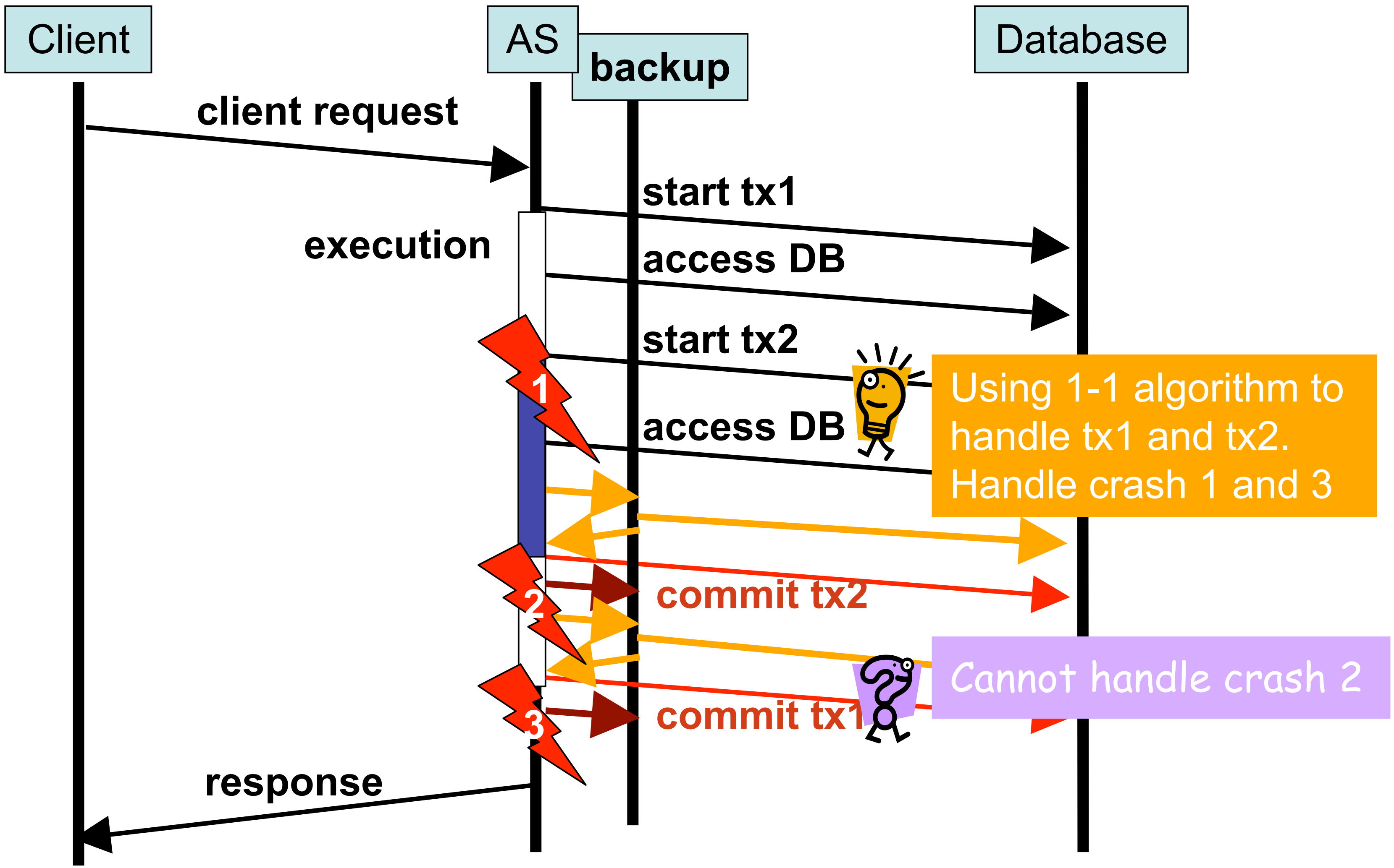
Transaction Patterns

- Relationship between client requests and server side transactions
  - Basic transaction pattern:
    - 1 client request is 1 transaction
  - Advanced transaction patterns
    - 1 transaction spans more than one client request
    - 1 client request leads to more than one transaction
    - …

Practical Applications:

Advanced Patterns are widely used

1-N Pattern



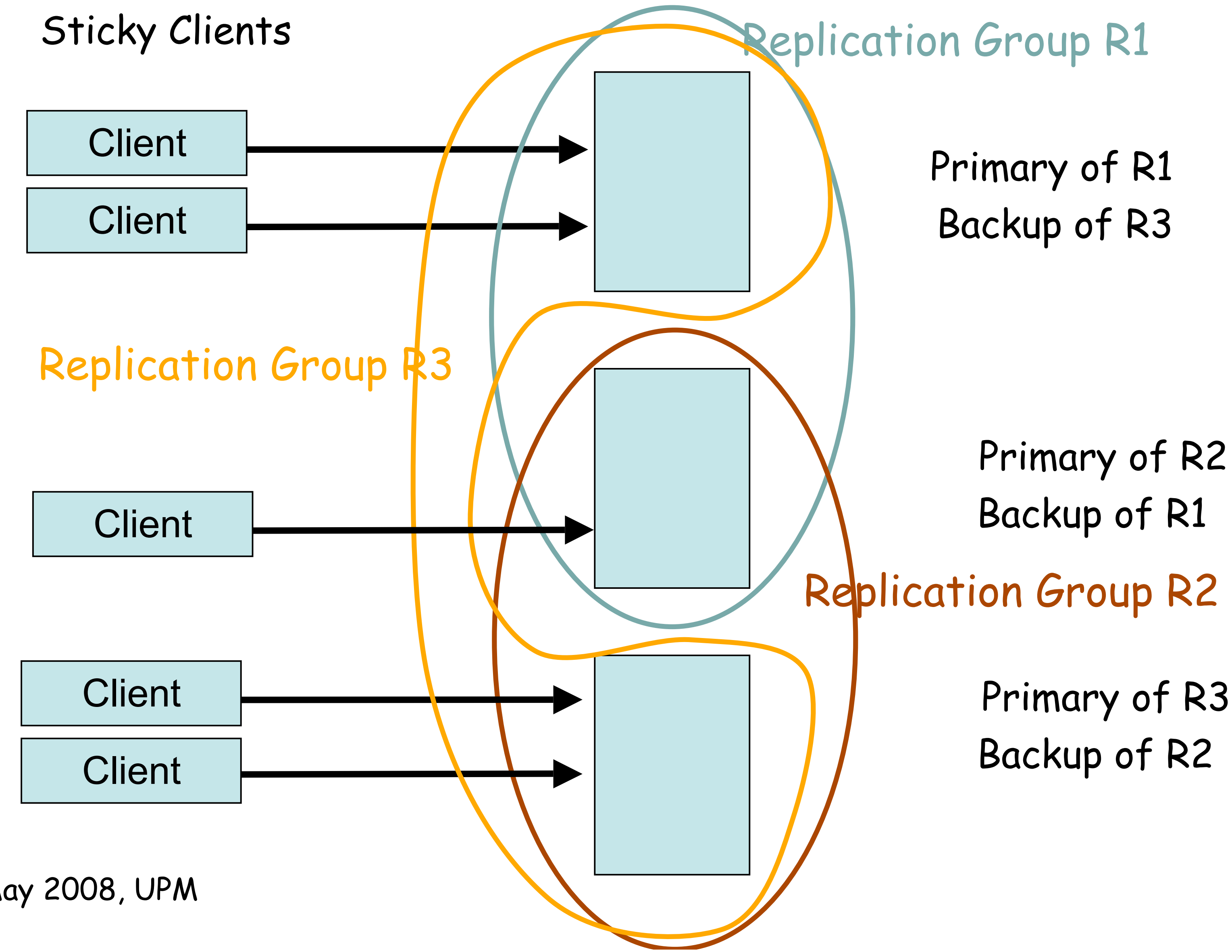
Client
AS
backup
Database
client request
start tx1
execution
access DB
start tx2
1
access DB
Using 1-1 algorithm to handle tx1 and tx2. Handle crash 1 and 3
2
commit tx2
Cannot handle crash 2
3
commit tx1
response

Further Issues

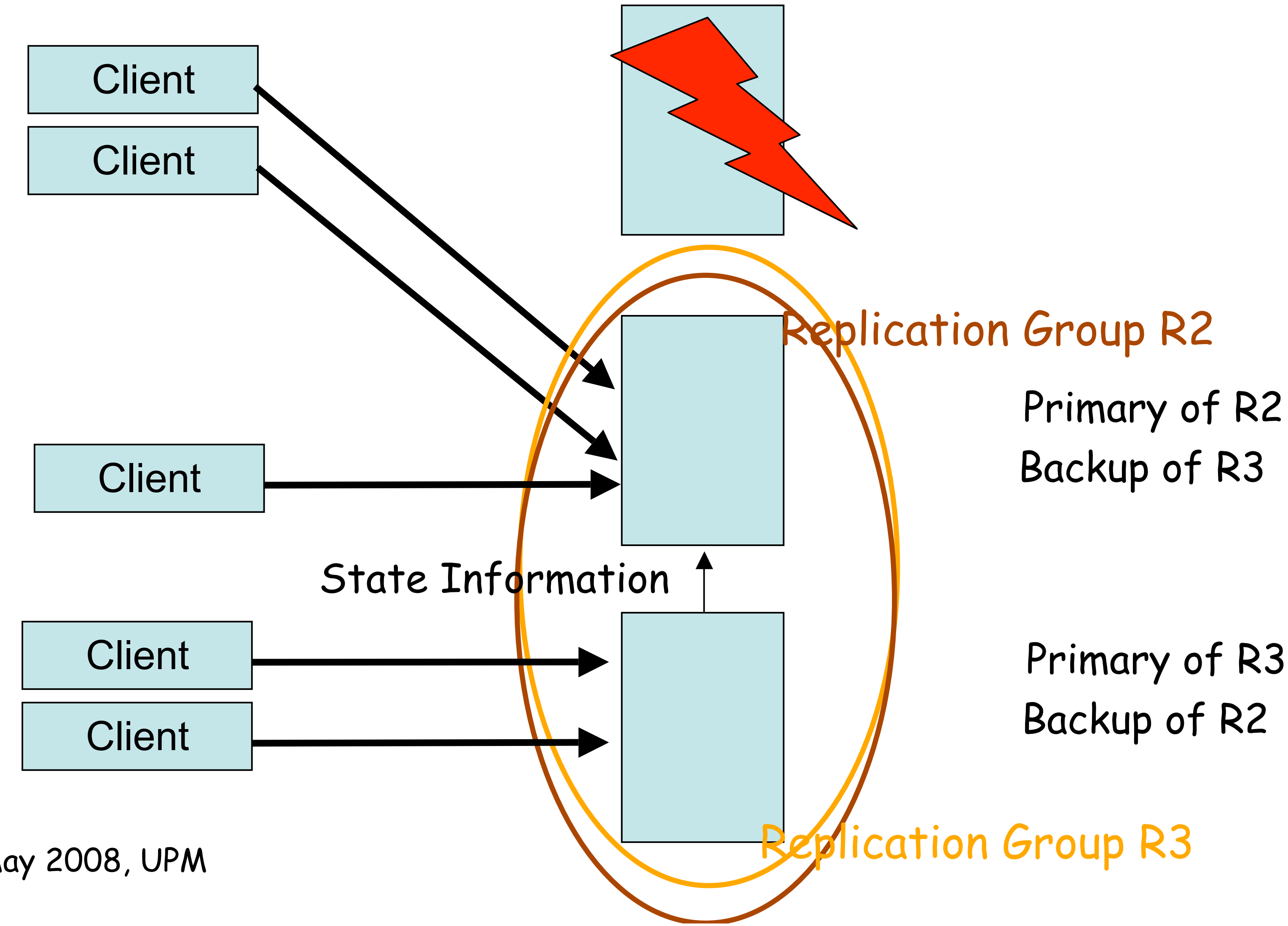
- Transaction Patterns
  - 1-N: 1 client request triggers several nested transactions
  - N-1: several client requests build one transaction
  - N-N pattern
  - Access more than one database
    - Combine replication with 2PC
- Reaction of AS on failures of other tiers
  - Client / DBS
- Recovery
  - Failed or new replicas rejoin as backups
  - Receive necessary backup information
- When to install changes at backup
  - Immediately when received
  - Only upon failover

Scalability and Fault-tolerance

Sticky Clients

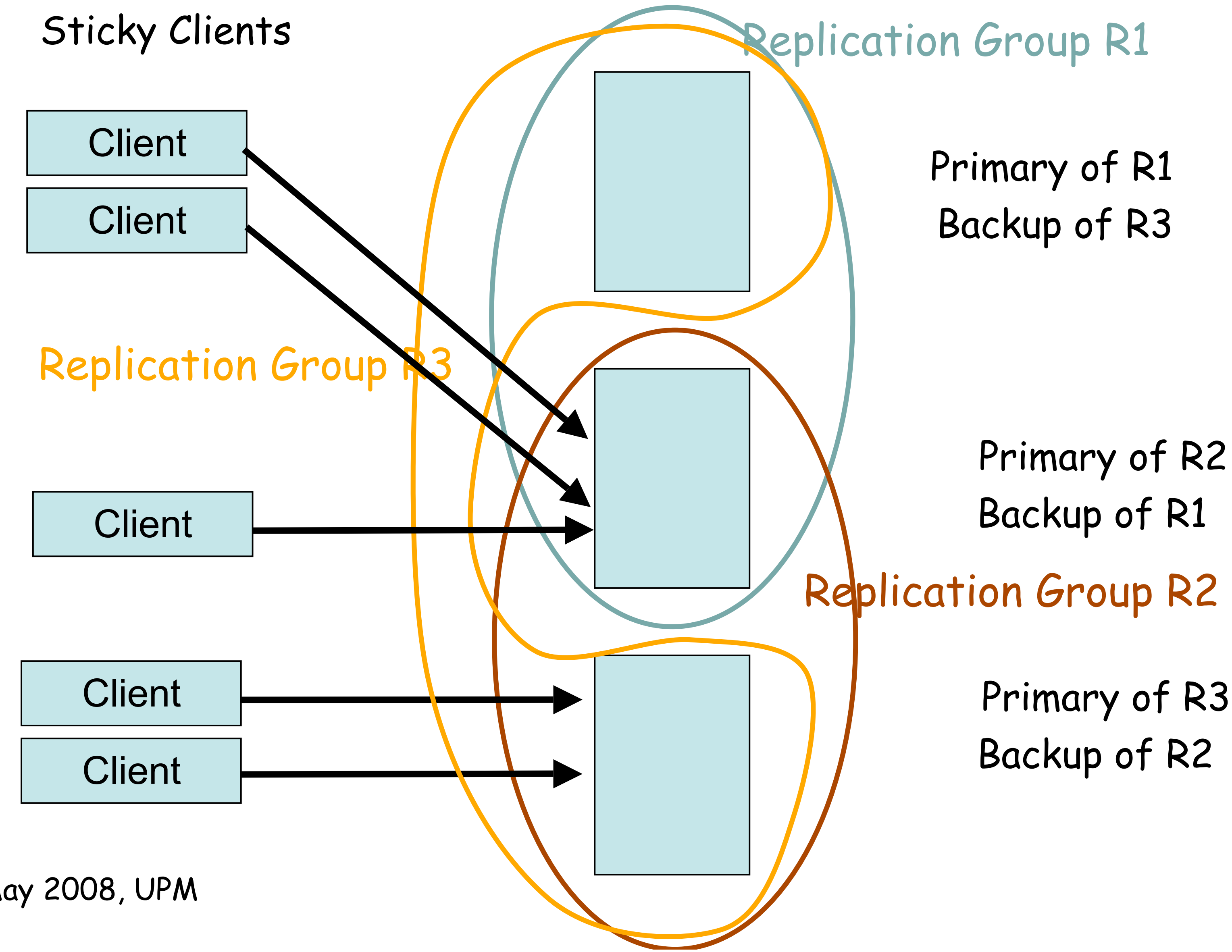


Scalability and Fault-tolerance

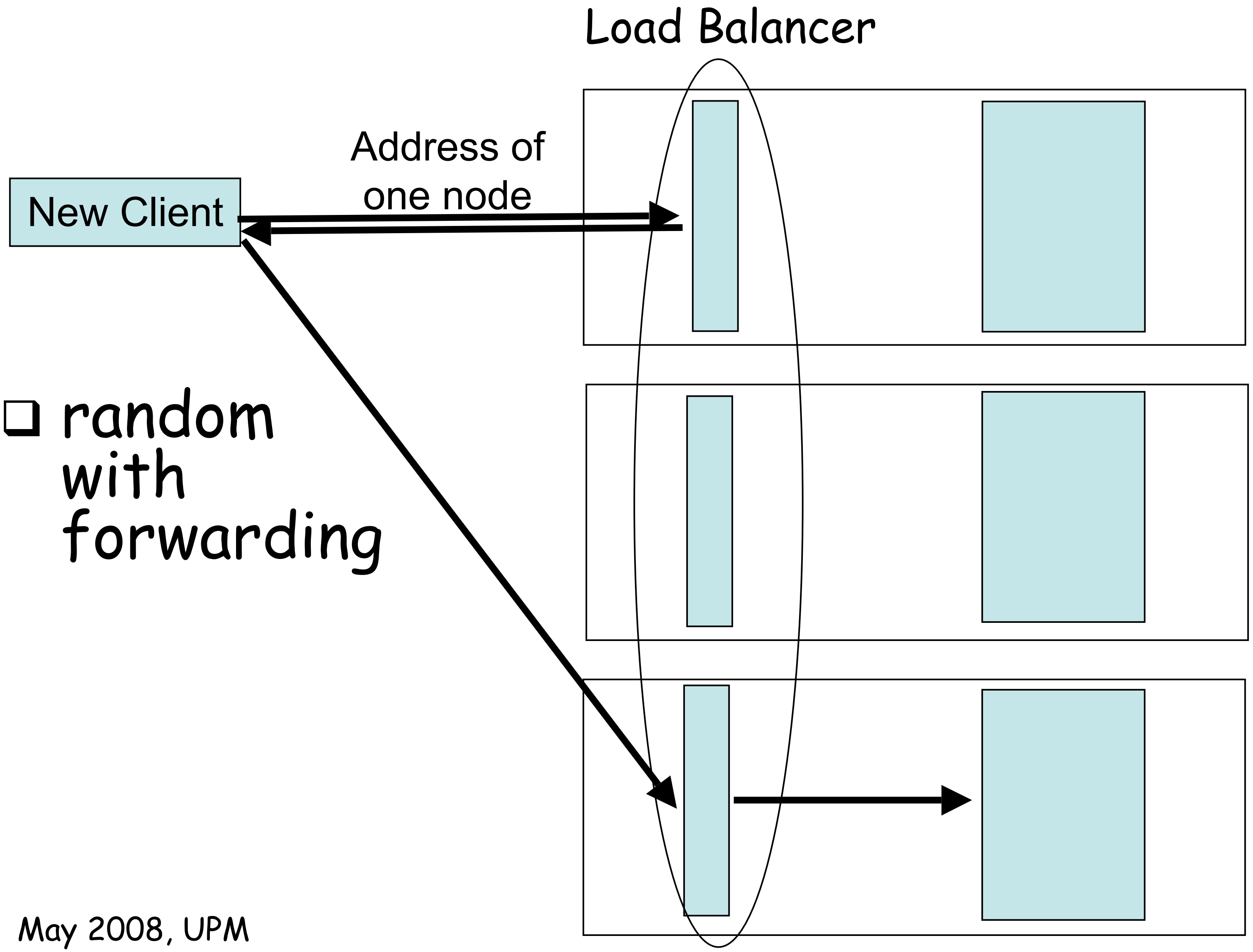


Scalability and Fault-tolerance

Sticky Clients



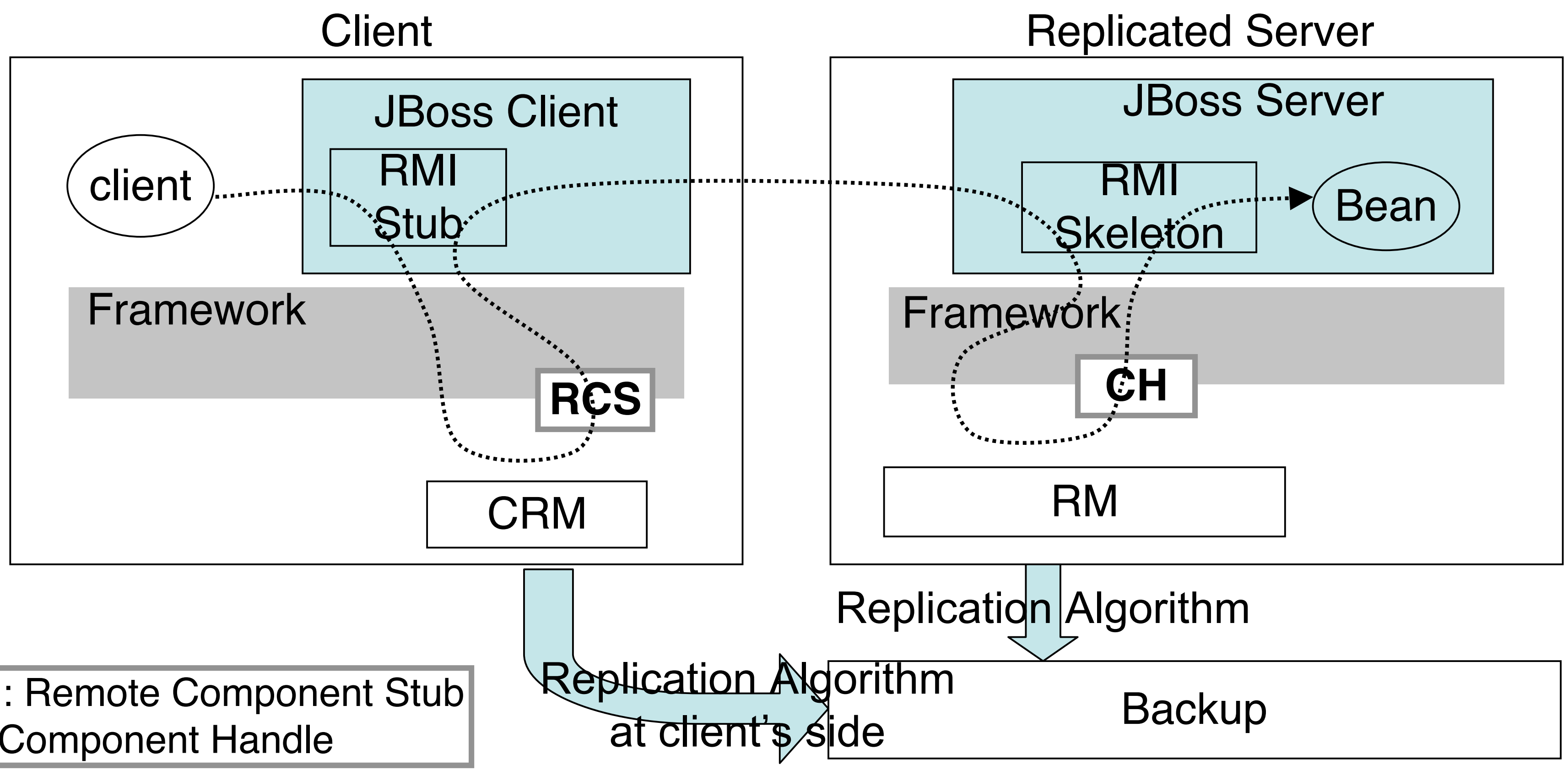
Client assignment



- random with forwarding

Implementation into JBoss 3.2.3

- Interceptor-based framework allows for plug-in of algorithms at client and server



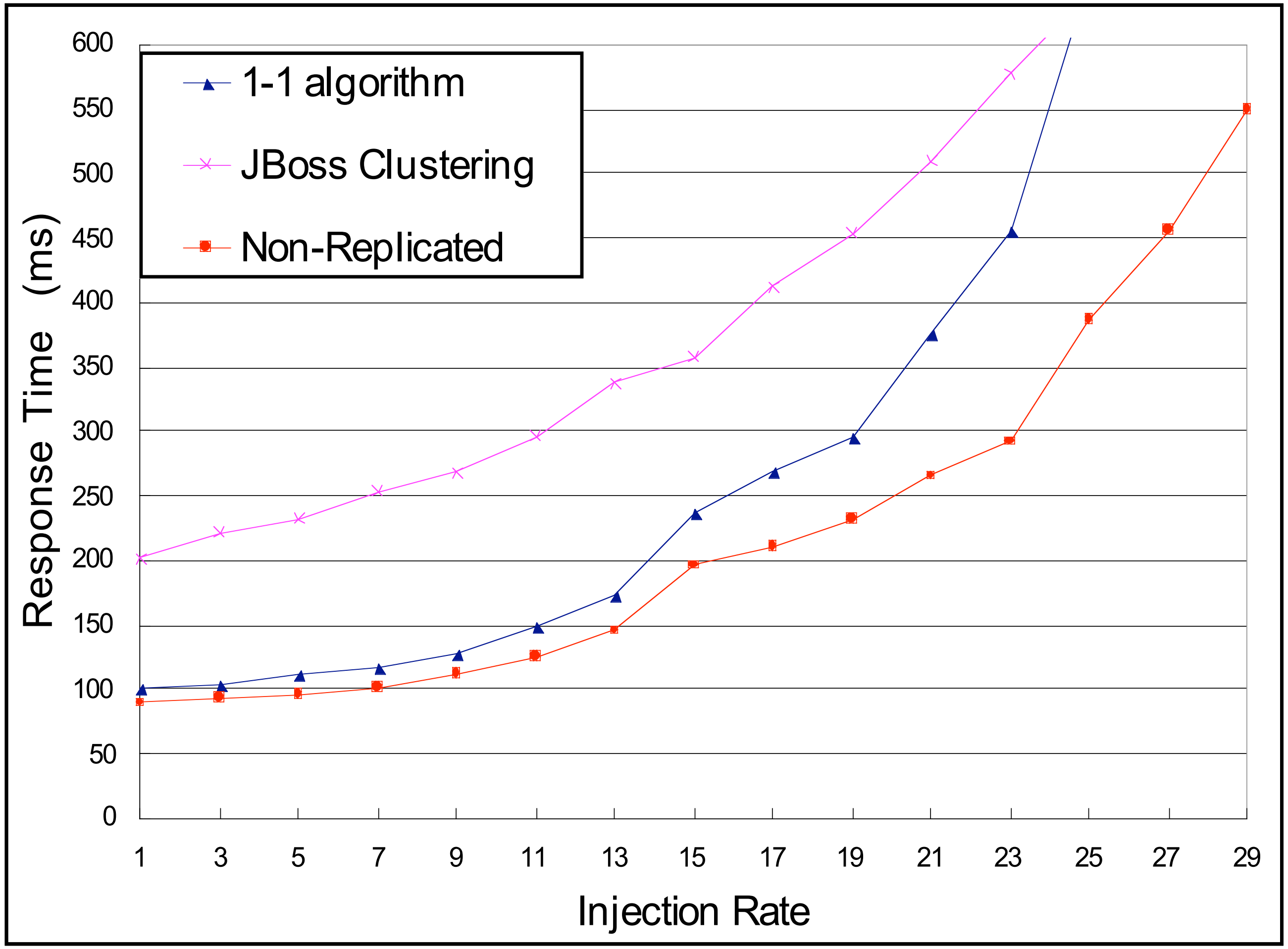
RCS: Remote Component Stub
CH: Component Handle

Performance Evaluation based on Modified ECperf benchmark

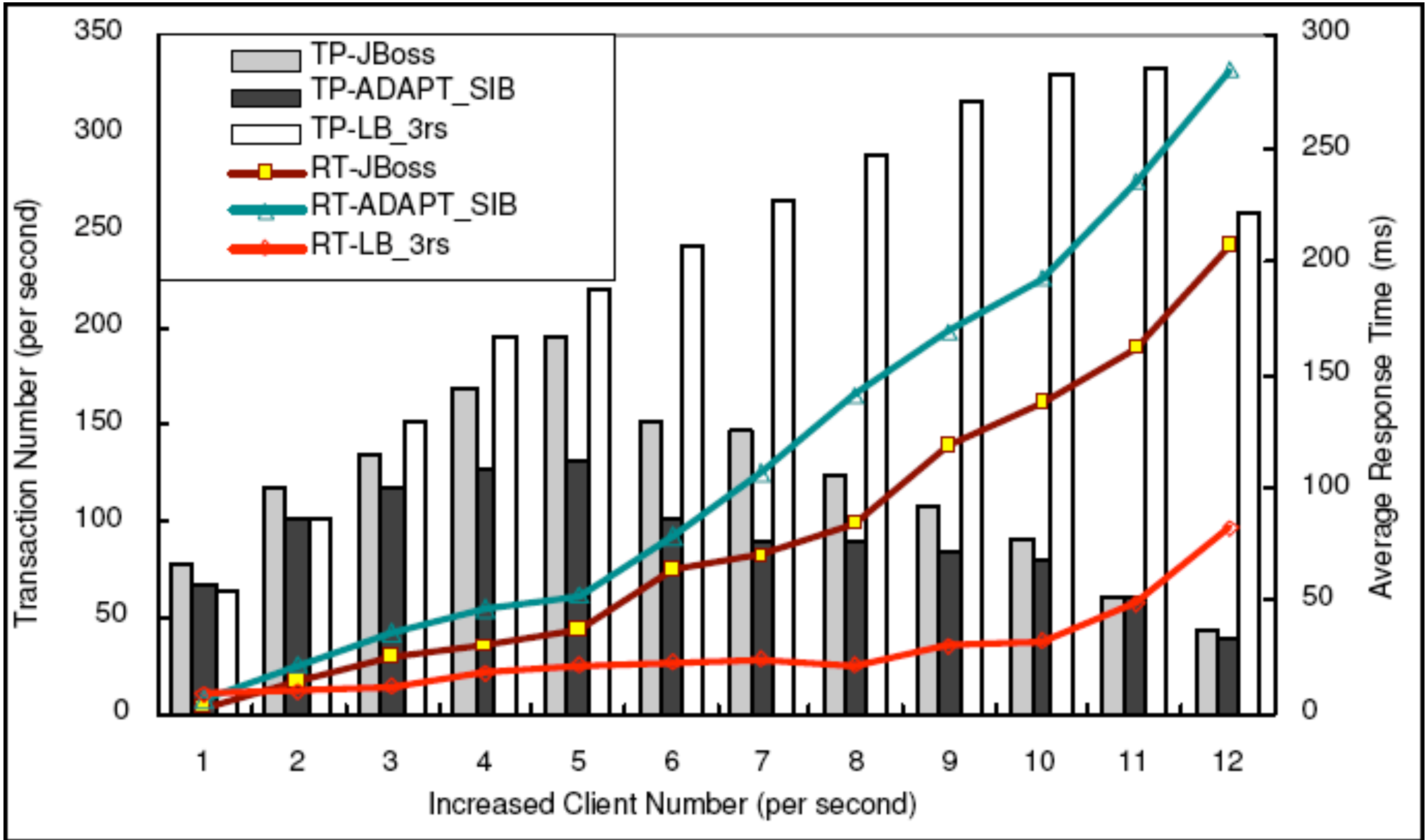
- Comparison:
  - Non-replicated JBoss
  - JBoss + our algorithm (Replicated JBoss)
  - JBoss' clustering (does not provide transactional exactly-once semantics)
- Sun ECPerf benchmark
  - Ordering/ manufacturing / supply-chain application

ECperf Response Time: 1-1 algorithm

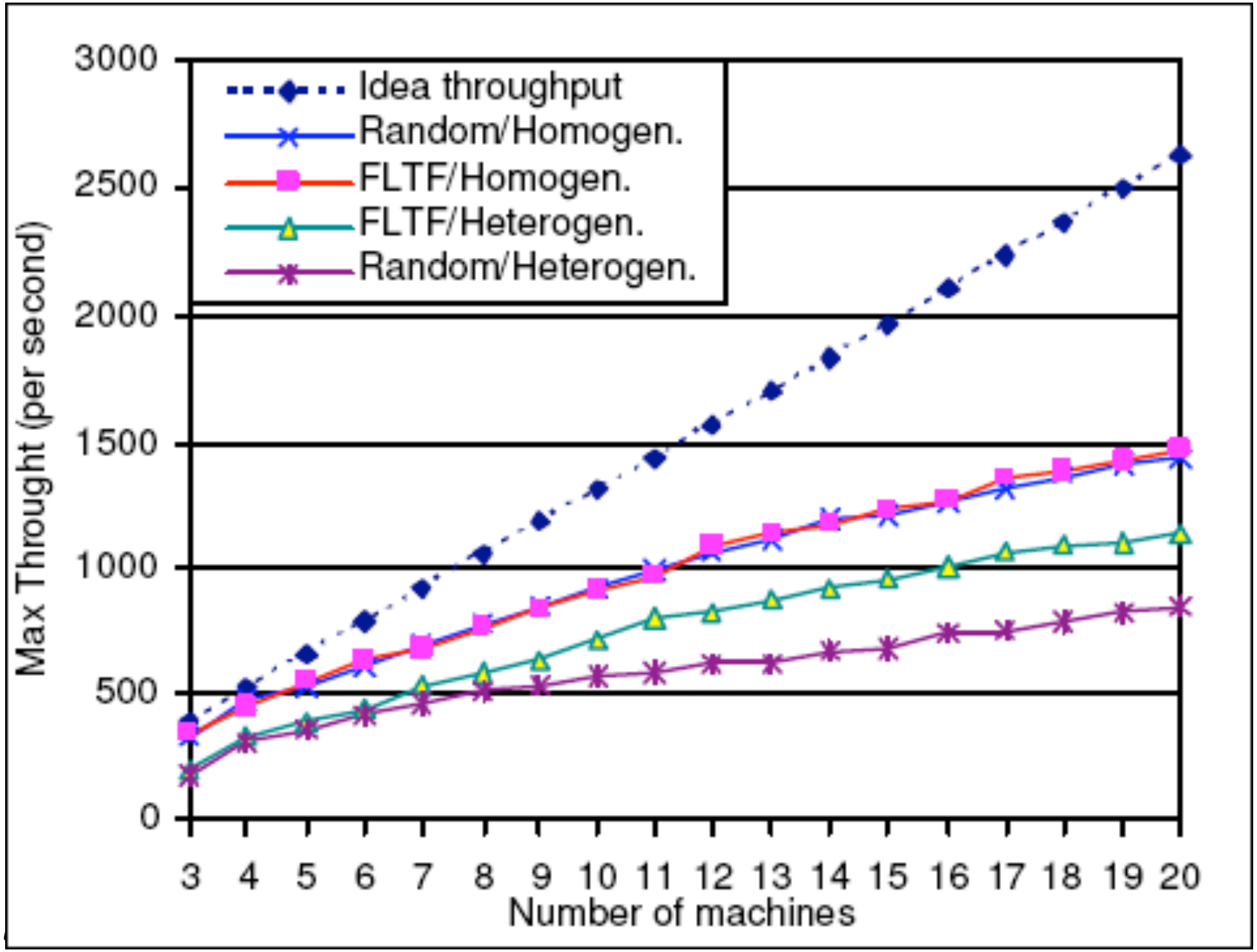
- Without load-balancing: 1 primary, 1 backup



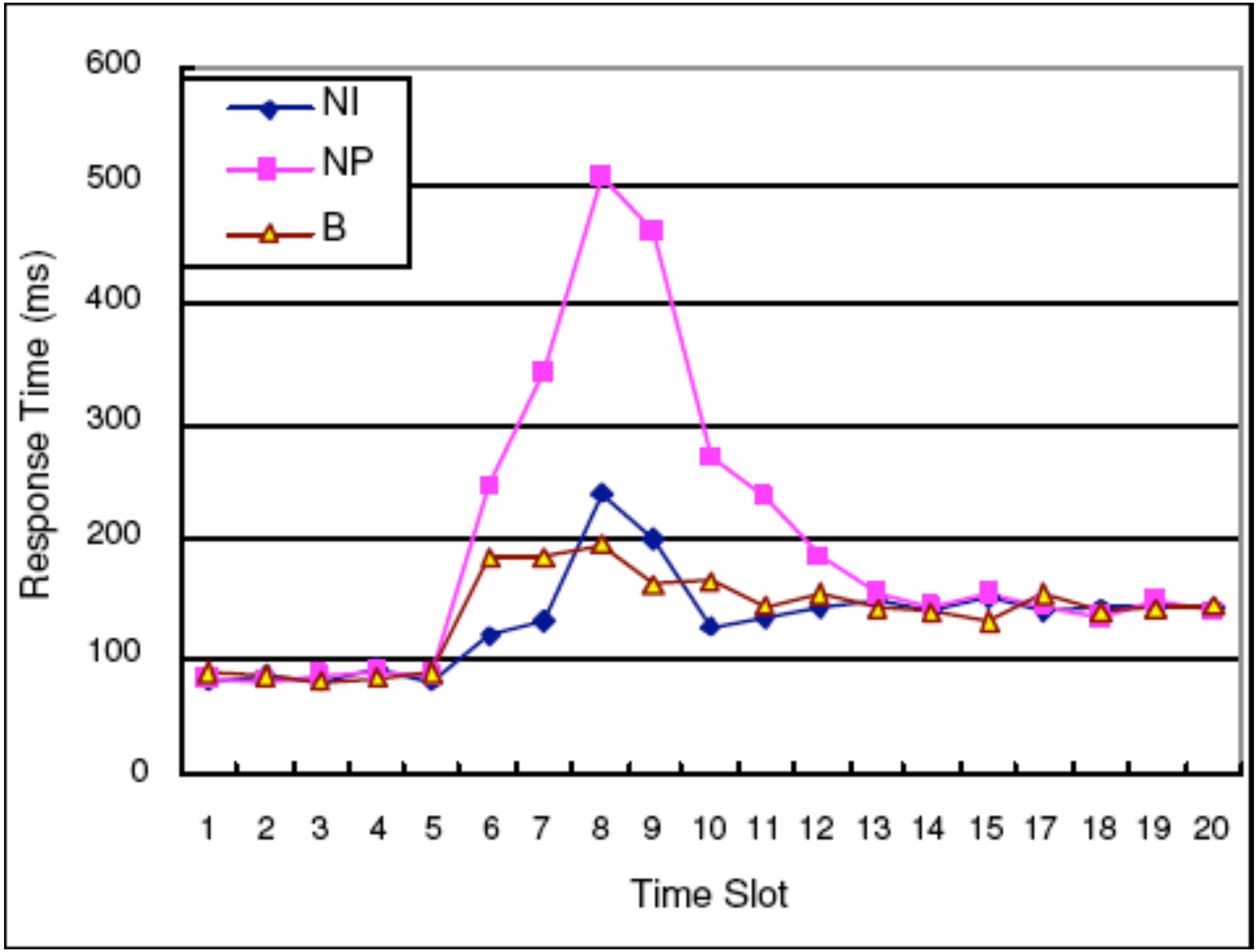
With load-balancing three replicas



Scalability

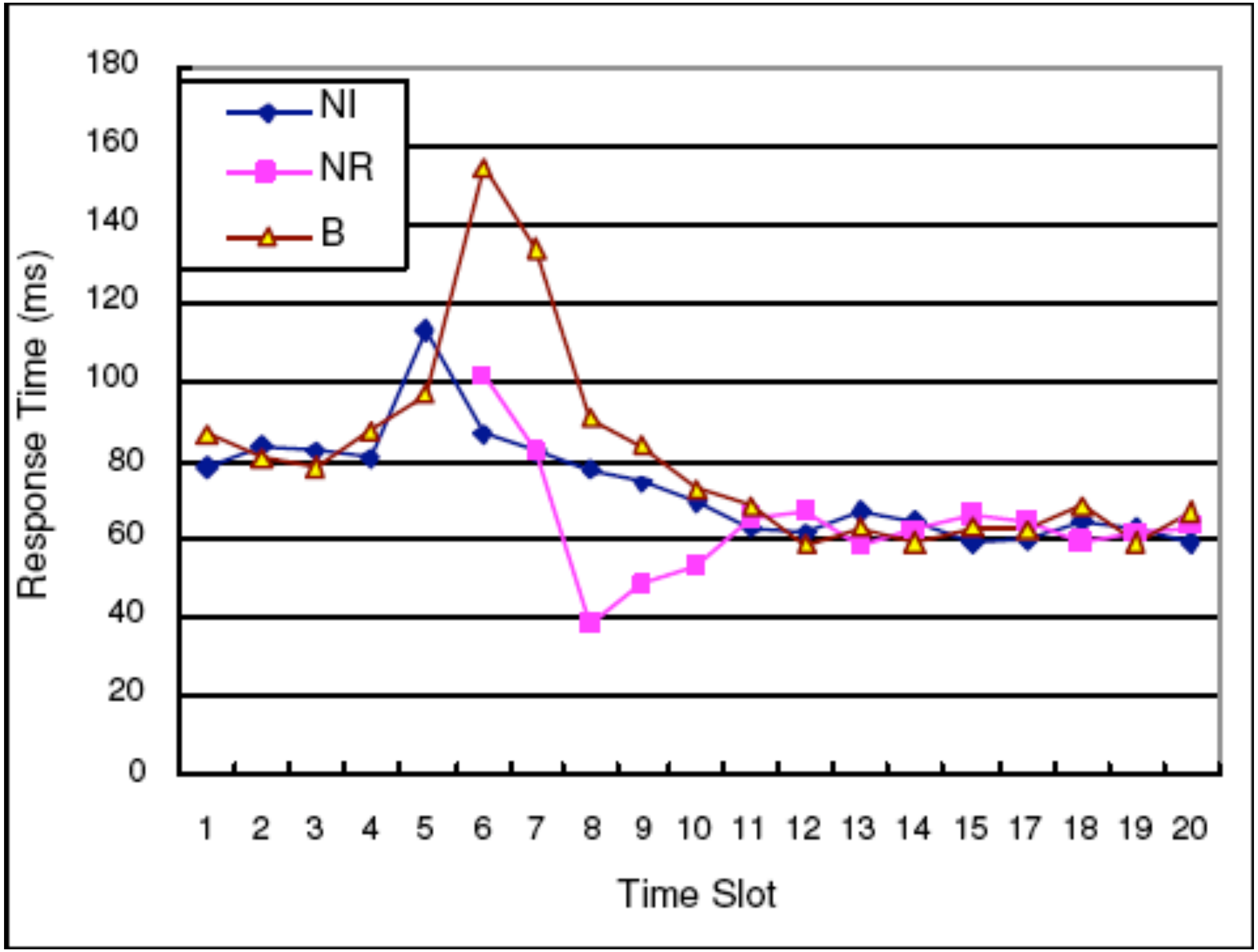


Failure



600
500
400
300
200
100
0
Response Time (ms)
NI
NP
B
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20
Time Slot

Recovery



180
160
140
120
100
80
60
40
20
0
Response Time (ms)
NI
NR
B
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20
Time Slot

Commercial AS replication

- Nearly all AS servers provide cluster support
  - Often lazy propagation
  - Several use logging instead of replication
  - Behavior often not well-defined
    - Often not correct transactional exactly-once execution
  - No advanced patterns

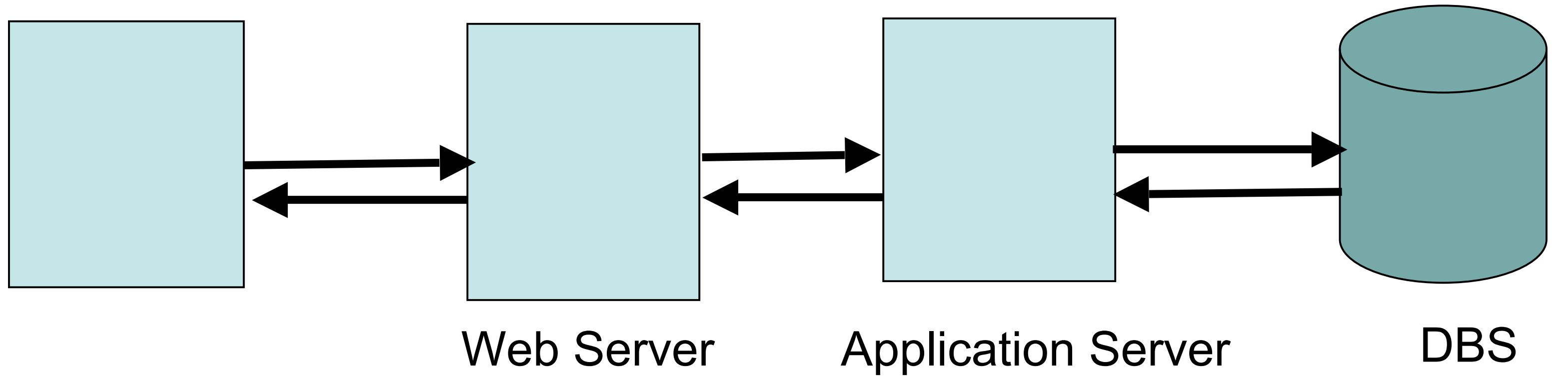
Other Research on AS replication

- Stateless / 1-1:
  - FRØLUND, S. AND GUERRAOUI, R. 2002. E-transactions: end-to-end reliability for three-tier architectures. IEEE Transactions on Software Engineering (TSE) 28, 4.
- Corba / 1-1:
  - ZHAO, W., MOSER, L. E., AND MELLIAR-SMITH, P. M. 2002. Unification of replication and transaction processing in three-tier architectures. In Int. Conf. on Distributed Computing Systems (ICDCS).
  - FELBER, P. AND NARASIMHAN, P. 2002. Reconciling replication and transactions for the end-to-end reliability of CORBA applications. In Int. Symp. on Distributed Objects and Applications (DOA).
- .Net / 1-1,1-N:
  - BARGA, R., CHEN, S., AND LOMET, D. 2004. Improving logging and recovery performance in Phoenix/App. In Int. Conf. on Data Engineering (ICDE).

Other Research

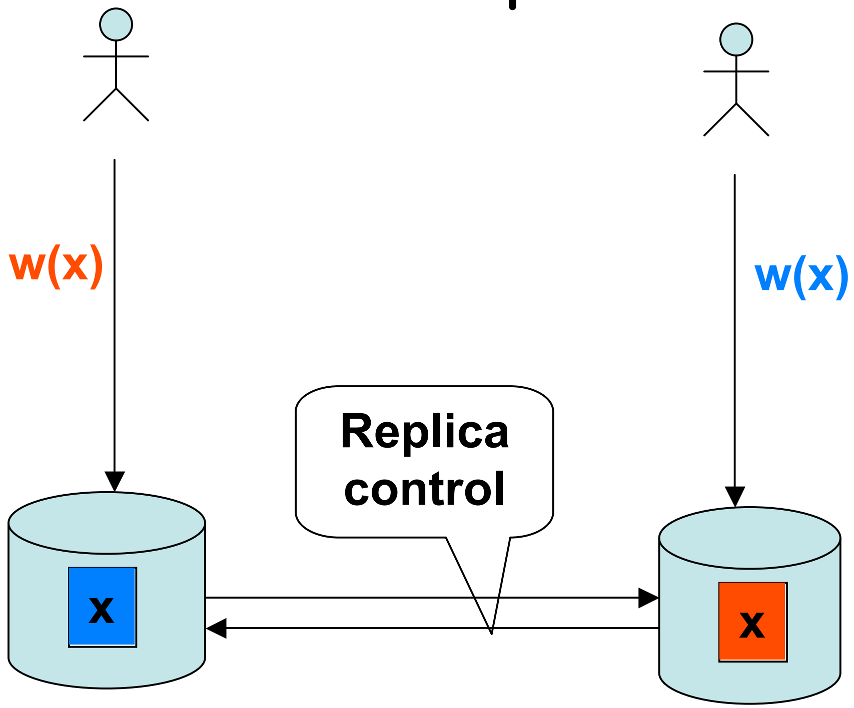
- Multi-tier Replication
  - FRØLUND, S. AND GUERRAOUI, R. 2000b. X-ability: a theory of replication. In Symp. on Princ. of Distrib. Comp. (PODC).
  - DEKEL, E. AND GOFT, G. 2004. ITRA: inter-tier relationship architecture for end-to-end QoS. The Journal of Supercomputing 28.

Replication of the DBS tier



Replica Control

- Keep copies consistent: **replica control**
- 1-copy-serializability
- Difference to AS Replication



Data Consistency

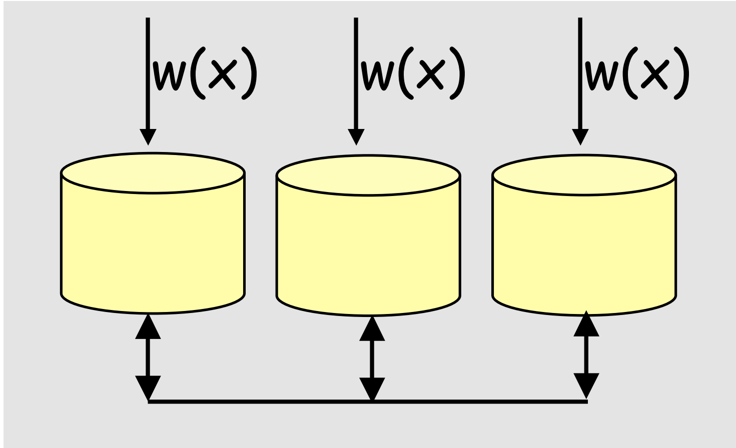
- Strong consistency
  - All available copies of an object have the same value at the end of the execution of an update request
  - Clients always read latest versions of data
  - High overhead
  - Tricky if crashes and network partitions
- weak consistency
  - temporal divergence allowed
  - *eventual consistency*
    - if update activity ceases, then all copies of a data item converge eventually to the same value
  - Clients might read stale or inconsistent data

Correctness

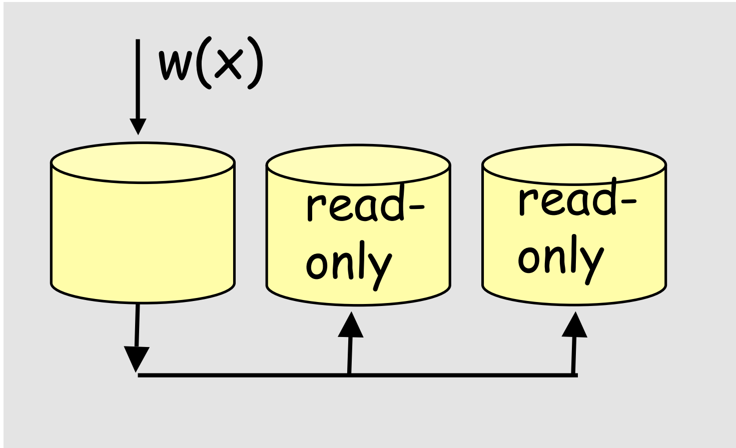
- (Replicated) Data is accessed within the boundaries of transactions with ACID properties
  - A transaction is a sequence of read and write operations
  - ROWA: read one - write all
- Global serializability:
  - The execution of transactions over the physical copies D^i of the replicated system is equivalent to a serial execution over the logical single-copy database D.
- Data consistency vs. 1CSR
  - A system can provide both strong consistency and global serializability
  - A system can provide weak consistency and global serializability
  - A system can provide strong consistency but no serializability
  - A system can provide provide only weak consistency and no serializability

Where can updates be submitted?

- Update Anywhere:
  - Update transactions can be submitted to any site
  - Site forwards updates to other sites



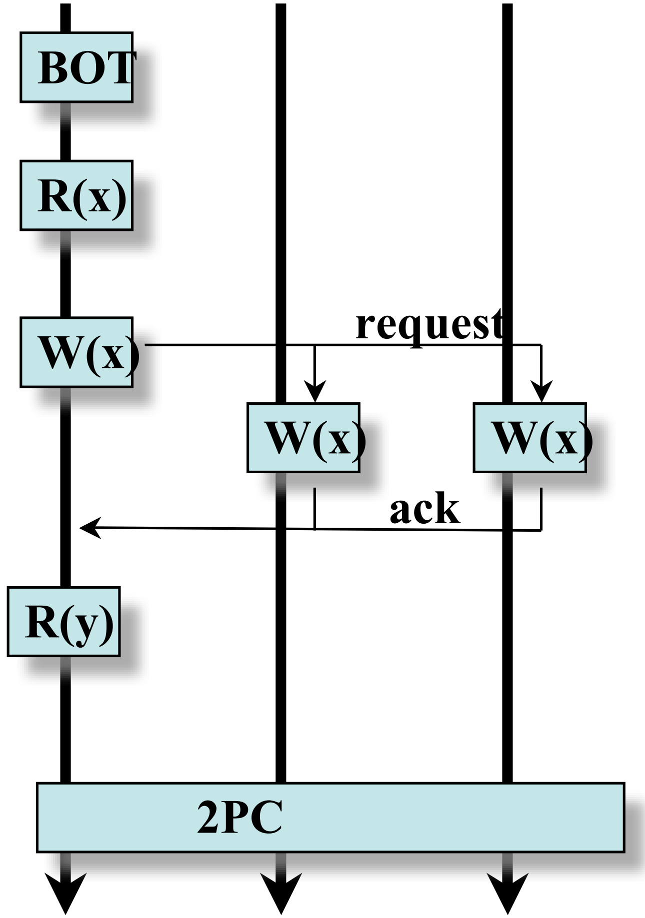
- Primary Copy:
  - Update transactions can only execute at the primary copy (master)
  - Primary forwards updates to secondaries



When to propagate

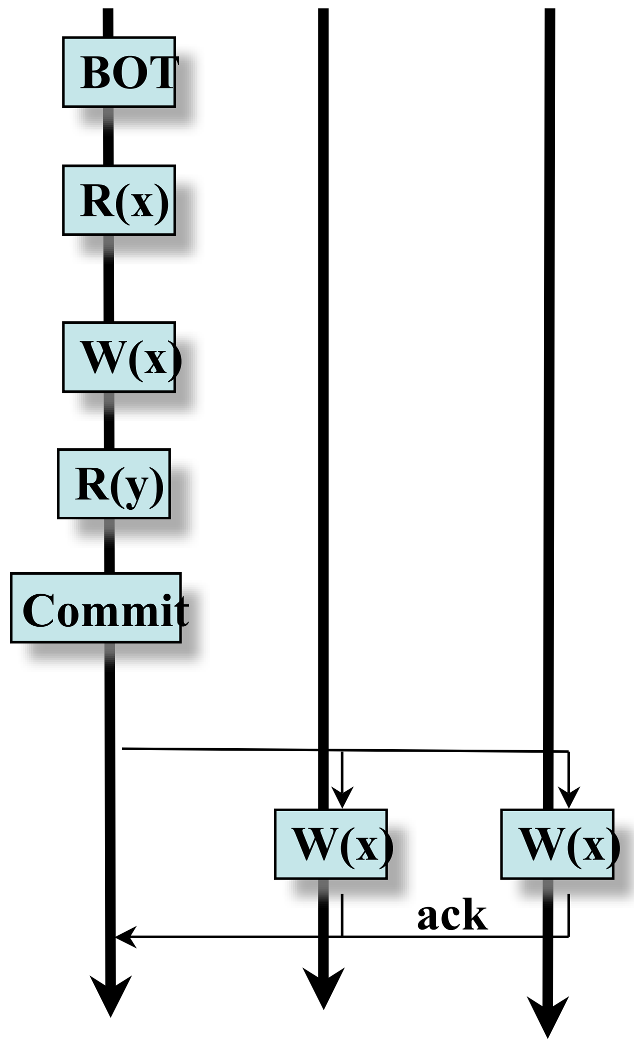
- Eager:
  - within the boundaries of the transaction
  - Transactions terminate usually with 2PC

Eager



When to propagate

- Lazy:
  - after the commit of the transaction



Basic Eager / Primary Copy

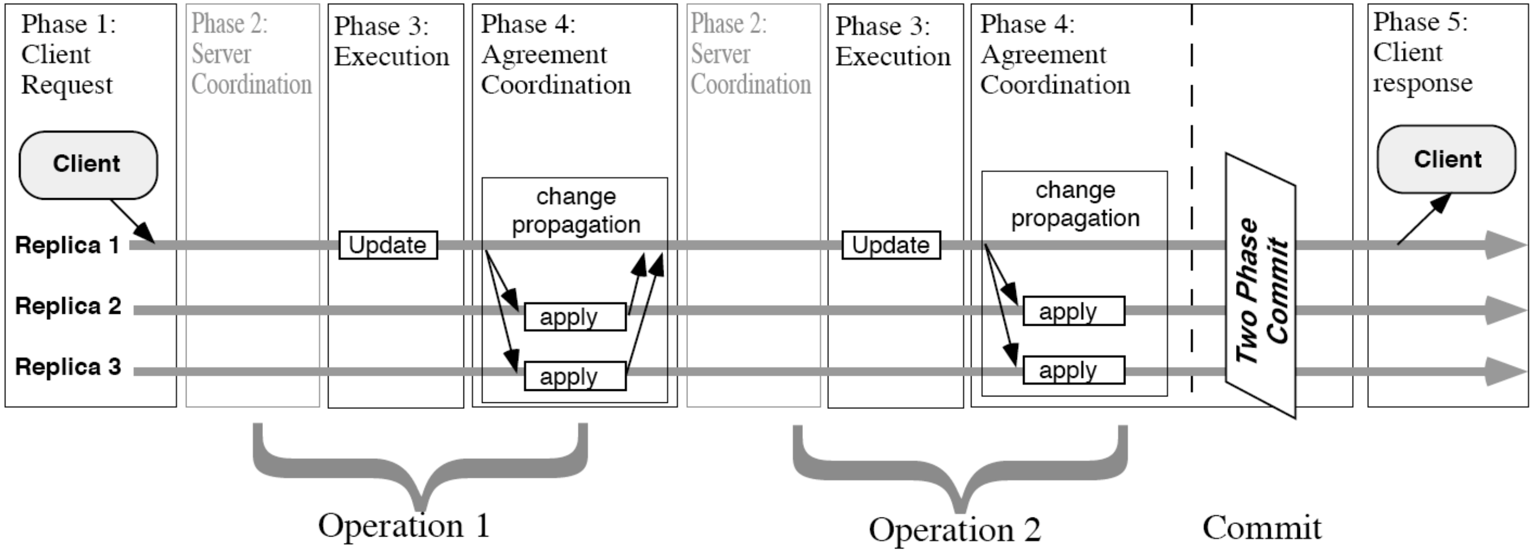
- Primary Copy:
  - Upon read: get local lock, read locally and return to user
  - Upon write: get local lock, write locally, multicast write to other replicas in FIFO order; return to user immediately
  - Upon commit request: run 2PC (coordinator) to ensure that all have really installed the changes.
  - Upon abort: abort and inform other sites about abort
- Secondary copy:
  - Upon read: get local lock, read locally
  - Upon write from primary copy: get locks in FIFO order and execute conflicting writes in FIFO order
  - Upon write from client: refuse (writing clients must submit to primary copy)
  - Upon commit request from read-only: commit locally
  - Participant of 2PC for update transaction running on primary
- In case of deadlocks:
  - Secondary copies should abort the reading transaction

Properties

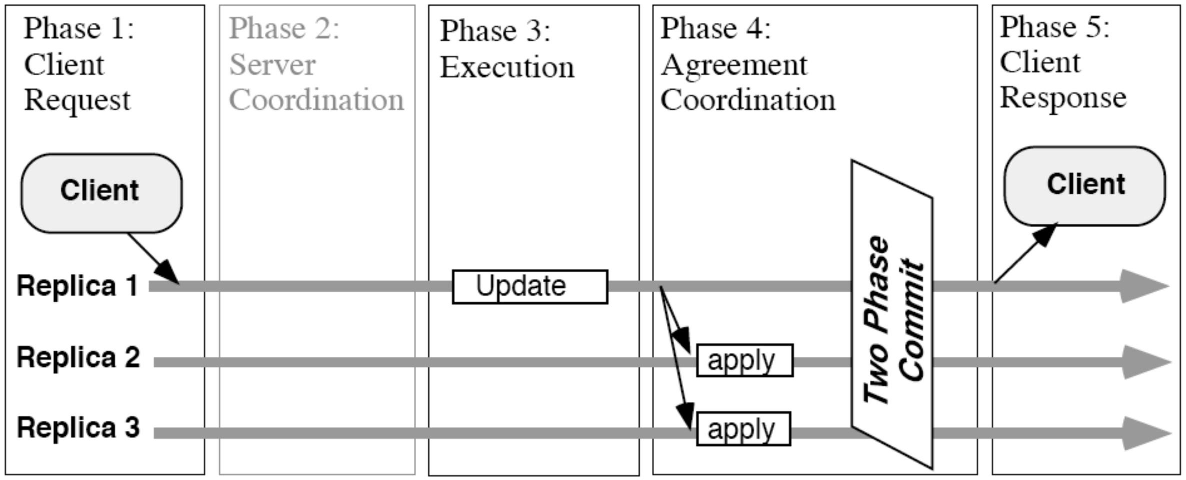
- No replication transparency
  - update transactions must be submitted to specific primary
  - How to achieve transparency?
- Global serializability and strong data consistency
- Reduce message overhead by sending all write operations (write set) within vote request message of 2PC
- Widely used for fault-tolerance
  - e.g. DB2 high availability solution
  - conceptually very similar to passive replication for object replication
- Multiple primary copies
  - each object can have primary copy on different server
  - What happens with a transaction that updates x and y, and x and y have their primary copies on different servers?

Eager Primary Copy

- Update propagation after each update



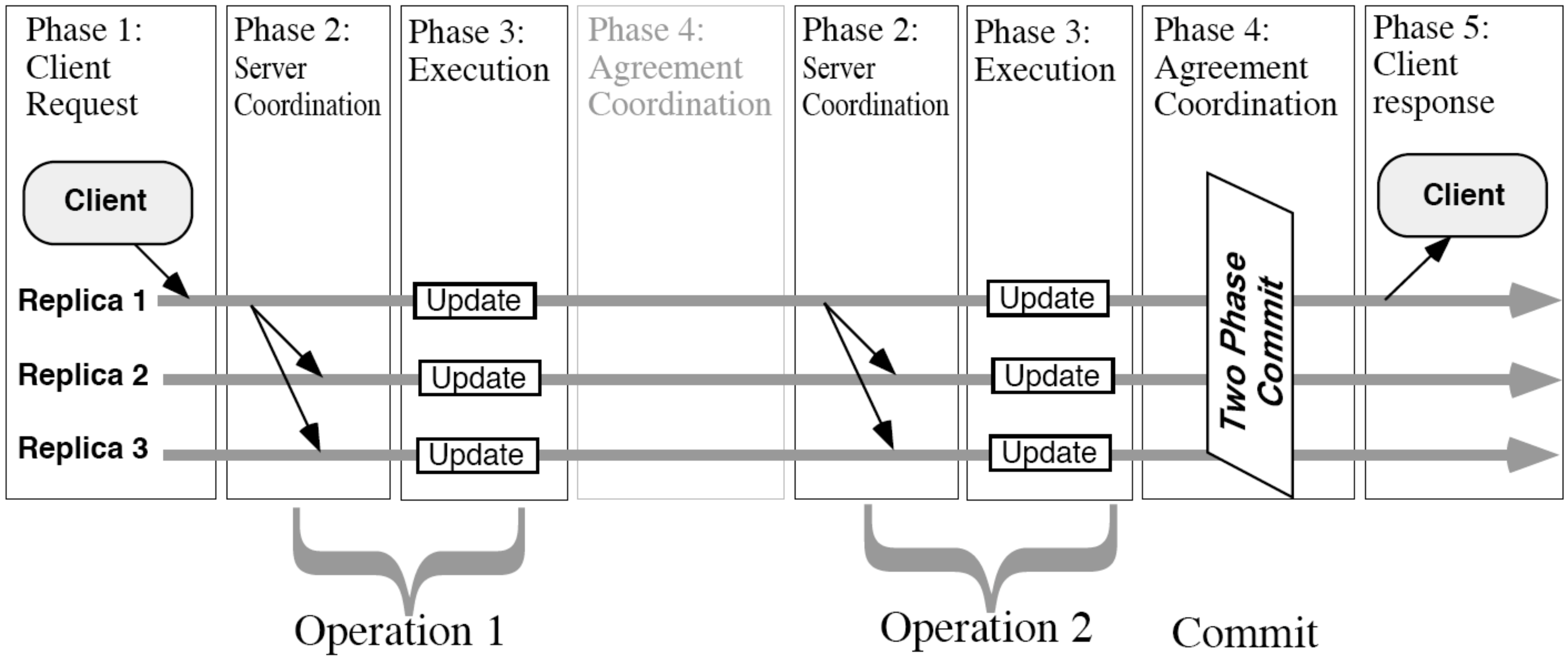
- Update propagation at end of transaction



Eager / Update Everywhere with distributed locking

- Upon read: request local read lock and read locally and return value to user
- Upon write from client: request local write lock and write locally, multicast write request to other sites.
- Upon write from other site: request local write lock, write locally, and send ok back to user
- Upon receiving ok from all other sites, return ok to the user
- Upon commit request: run 2PC to ensure that all have really installed the changes.
- Upon abort: abort and inform other sites about abort
- Deadlocks might occur.

Eager / Update Everywhere



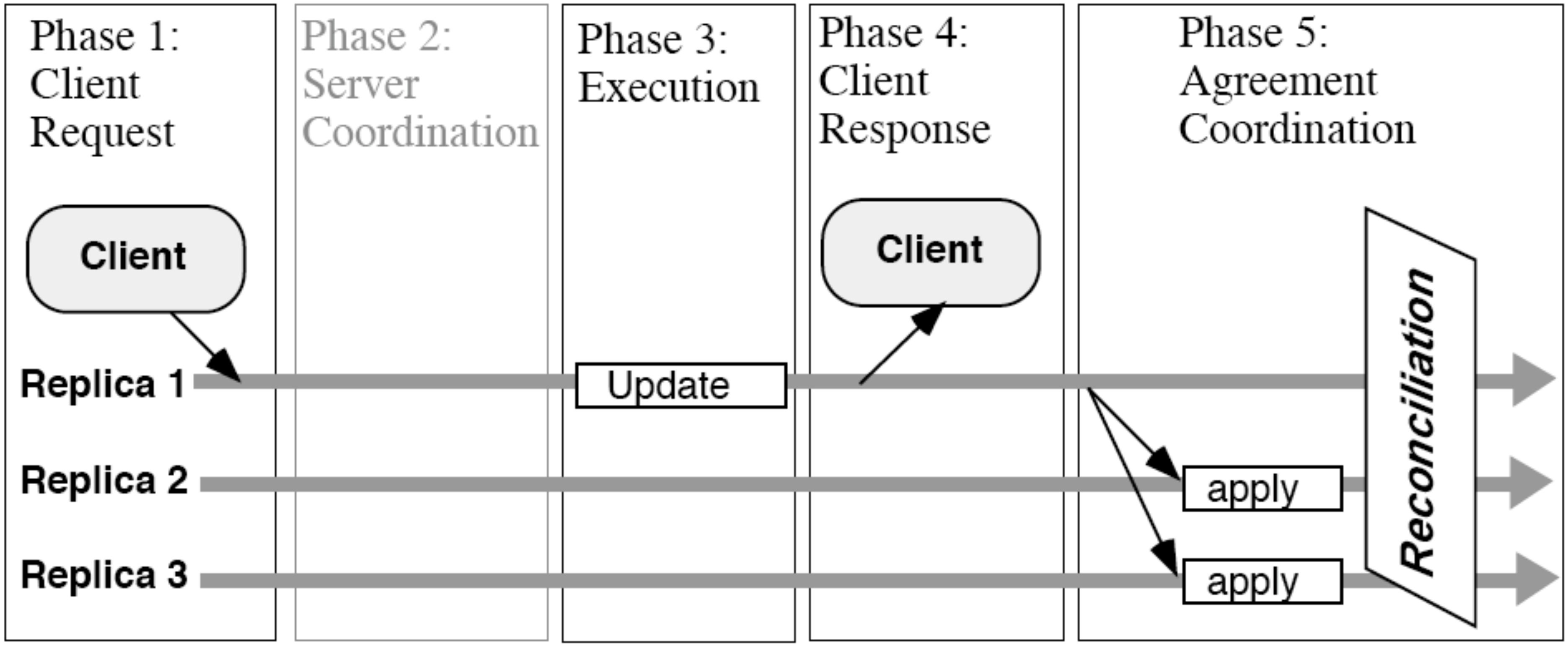
Properties

- Replication transparency achieved
- serializability and strong data consistency
- Concurrency control and coordination more complicated than with primary copy
- Better load balancing and distribution than Primary copy
- Reduce message overhead by sending write set at end of the transaction as part of 2PC
  - more complicated: how is this coordinated with locking?
- Basically no database system supports eager update anywhere
- but many middleware based solutions!

Lazy / Primary Copy

- Primary Copy:
  - Upon read: read locally and return to user
  - Upon write: write locally and return to user
  - Upon commit/abort: terminate locally
  - Sometime after commit: multicast changed objects in a single message to other sites (in FIFO)
- Secondary copy:
  - Upon read: read locally
  - Upon message from primary copy: install all changes (FIFO)
  - Upon write from client: refuse (writing clients must submit to primary copy)
  - Upon commit/abort request (only for read-only txn): local commit
  - Note: transaction might write local data that is NOT replicated or for which the site is the primary copy
- Only local deadlocks
- Note: existing systems allow different objects to have different primary copies
  - A transaction that wants to write X (primary copy is site S1) and Y (primary copy on site S2) is usually disallowed

Lazy Primary Copy



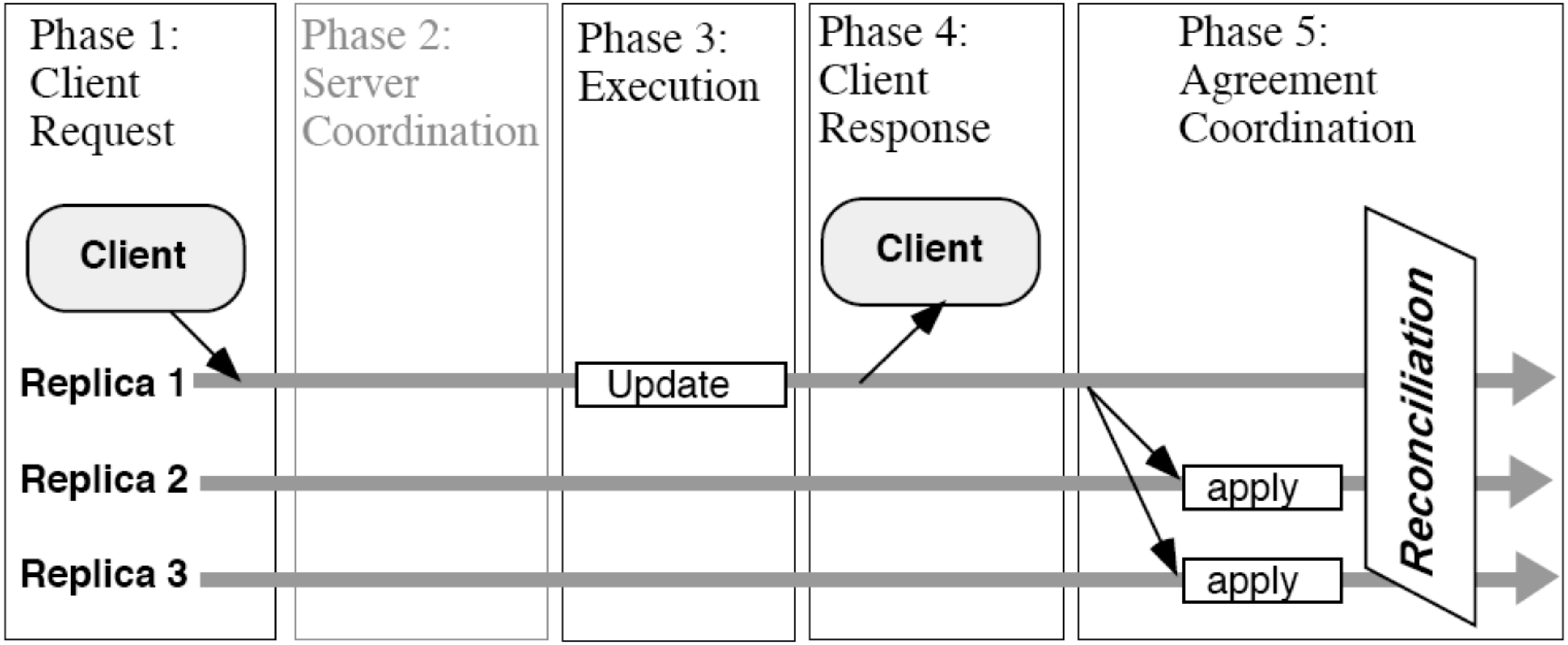
Discussion

- Lazy replication has no server/agreement coordination within response time
  - faster for clients close to primary copy
  - transactions might be lost in case of primary crash
- serializability and weak data consistency
  - simple to achieve
  - secondaries only need to apply updates in FIFO order
  - Data at secondaries might be stale
- Multiple Primary possible
  - more locality
- Optimizations for update propagation possible

Lazy / Update Everywhere

- Any site
  - Upon read: read locally and return to user
  - Upon write: write locally and return to user
  - Upon commit/abort: terminate locally
  - Sometime after commit: multicast changed objects in a single message to other sites (in FIFO)
  - Upon message from other site:
    - Detect conflicts
    - Resolve conflicts
      - for numeric types (or types with comparison):
        - average:
        - minimum/maximum:
        - additive:
      - discard new value, overwrite old value
      - Site priority
      - value priority
      - earliest/latest timestamp
    - Install changes

Lazy Update Everywhere



Discussion

- Weak data consistency and no global serializability
  - Data can be temporarily inconsistent
  - Reconciliation necessary
- No communication within transaction response time for all transactions
- Possible transaction loss in case of crash
- Conflict detection and resolution complex

Primary vs. Update everywere

- Simpler concurrency control
- Less coordination necessary / optimizations are easier
- Inflexible model:
  - Clients must know primary to submit update transactions
  - Have to distinguish update from read-only transactions
- Primary is single point of failure and potential bottleneck
- Multiple primaries
  - some type of transaction disallowed
  - More locality than one primary
  - Less bottleneck

Lazy vs. Eager

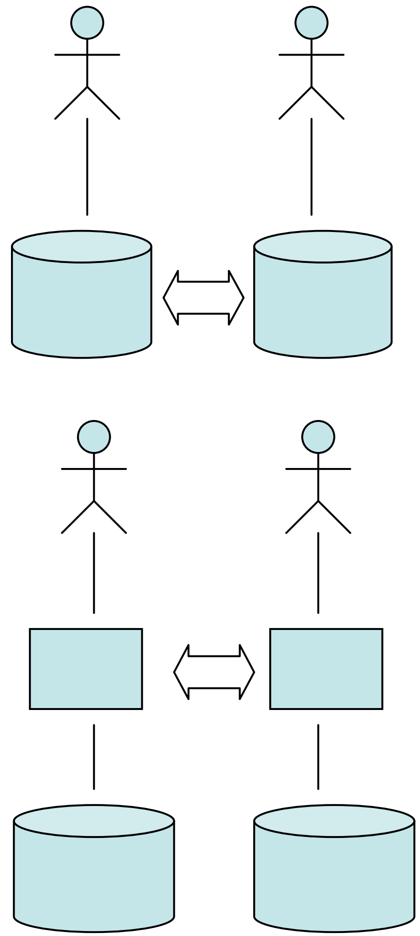
- Lazy primary copy: stale reads
- Lazy update everywhere: inconsistencies and reconciliation
- No communication within transaction response time
- Possible transaction loss in case of crash
- Optimizations for update propagation possible

Eager Protocols and Failures

- So far: read-one-write-all protocols (ROWA)
- Site failures:
  - Read-one-write-all-AVAILABLE (ROWAA)
- Communication failures:
  - Combine ROWAA with quorums

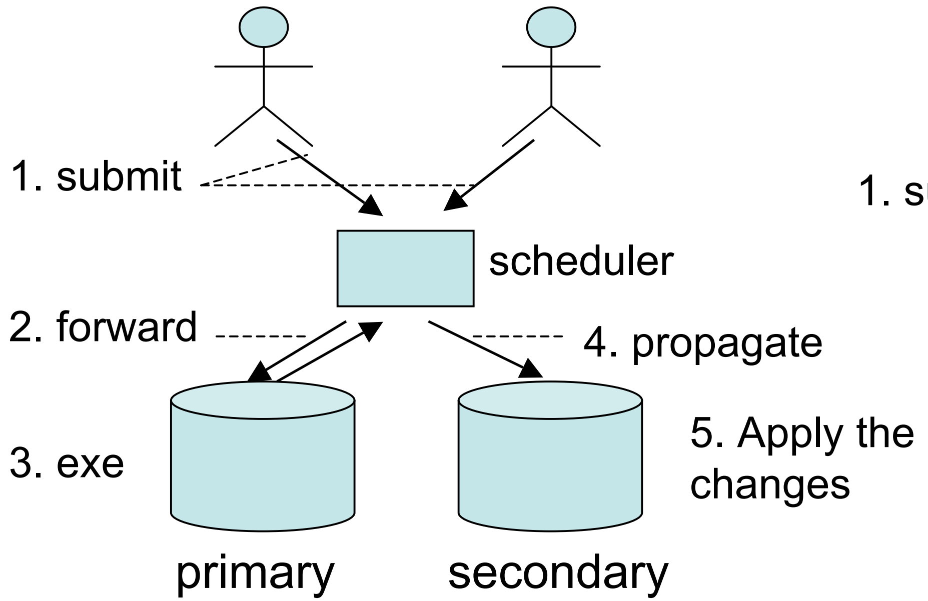
Recent Replica Control Approaches

- so far: Kernel-based approach
- new: Middleware-based approach
  - Advantages
    - Modular
    - Do not need access to DB code
    - Reusability
  - Disadvantages
    - No access to concurrency control information in the kernel

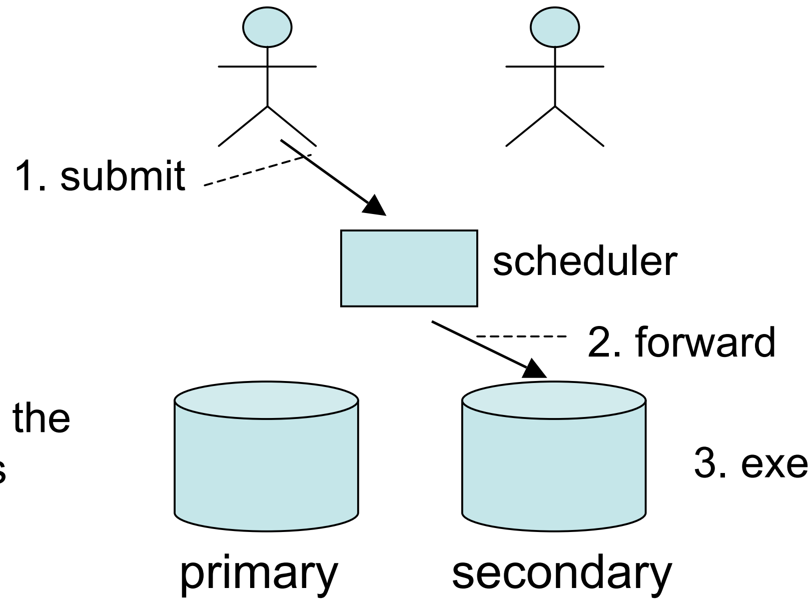


Middleware Primary Copy

- (e.g. Ganymed)



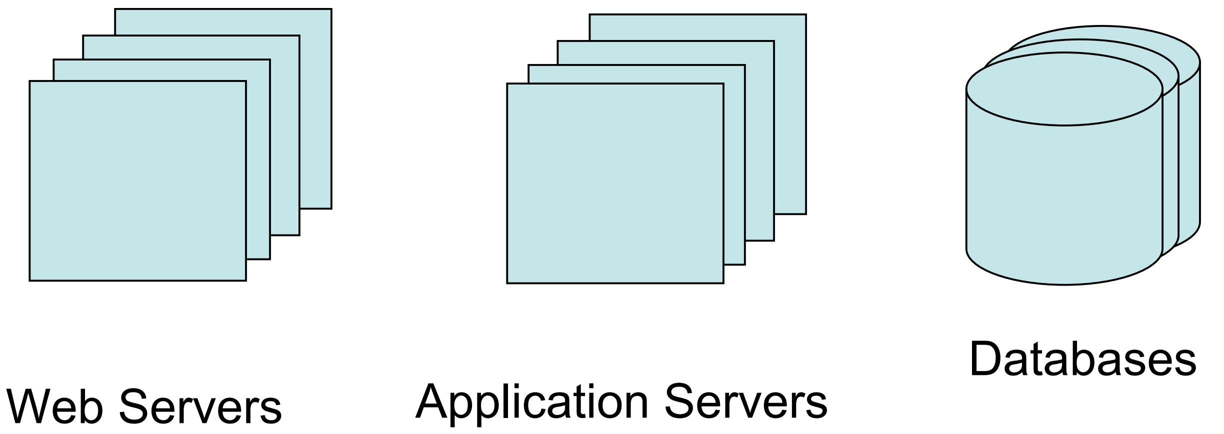
Update transactions



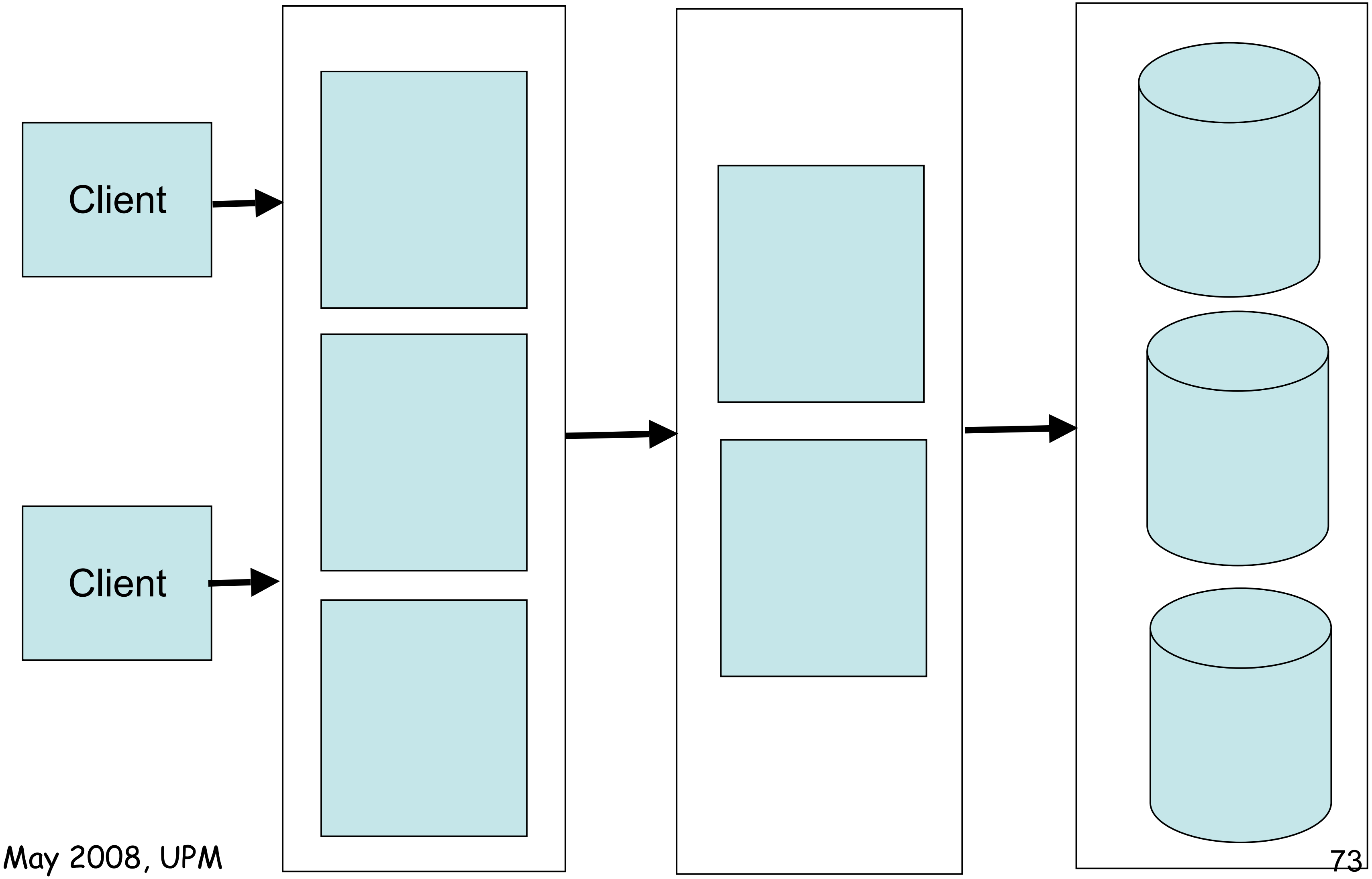
Read only transactions

Adaptability across the entire cluster

- Transparent Failover
- Load-balancing
- Combination of both
- Data Management



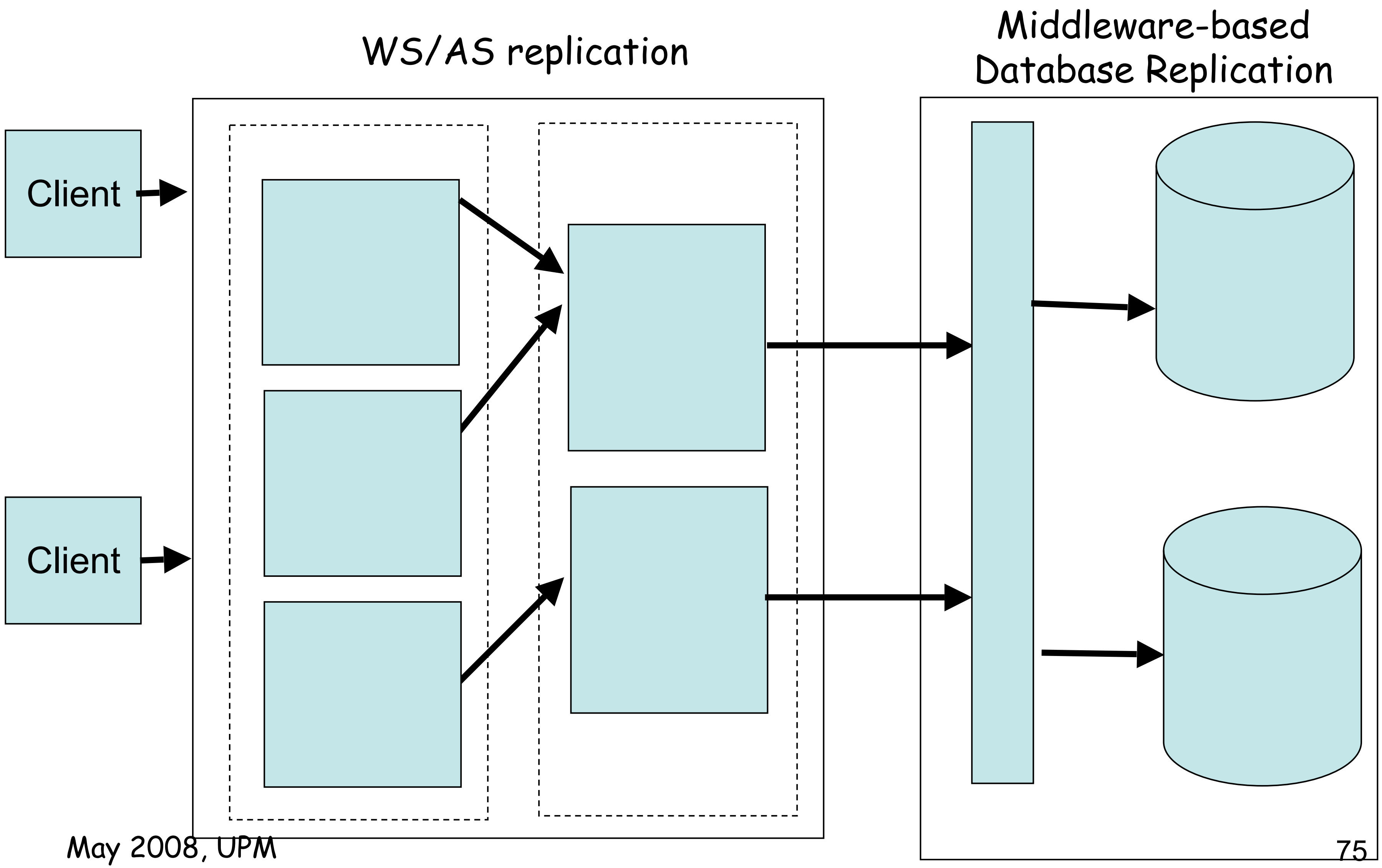
Horizontal Replication



Loose Coupling

- Each tier has its own replication algorithm
- Each tier not aware of replication of neighboring tiers
- Tier only needs to know behavior of tiers it directly calls (later tiers are hidden)

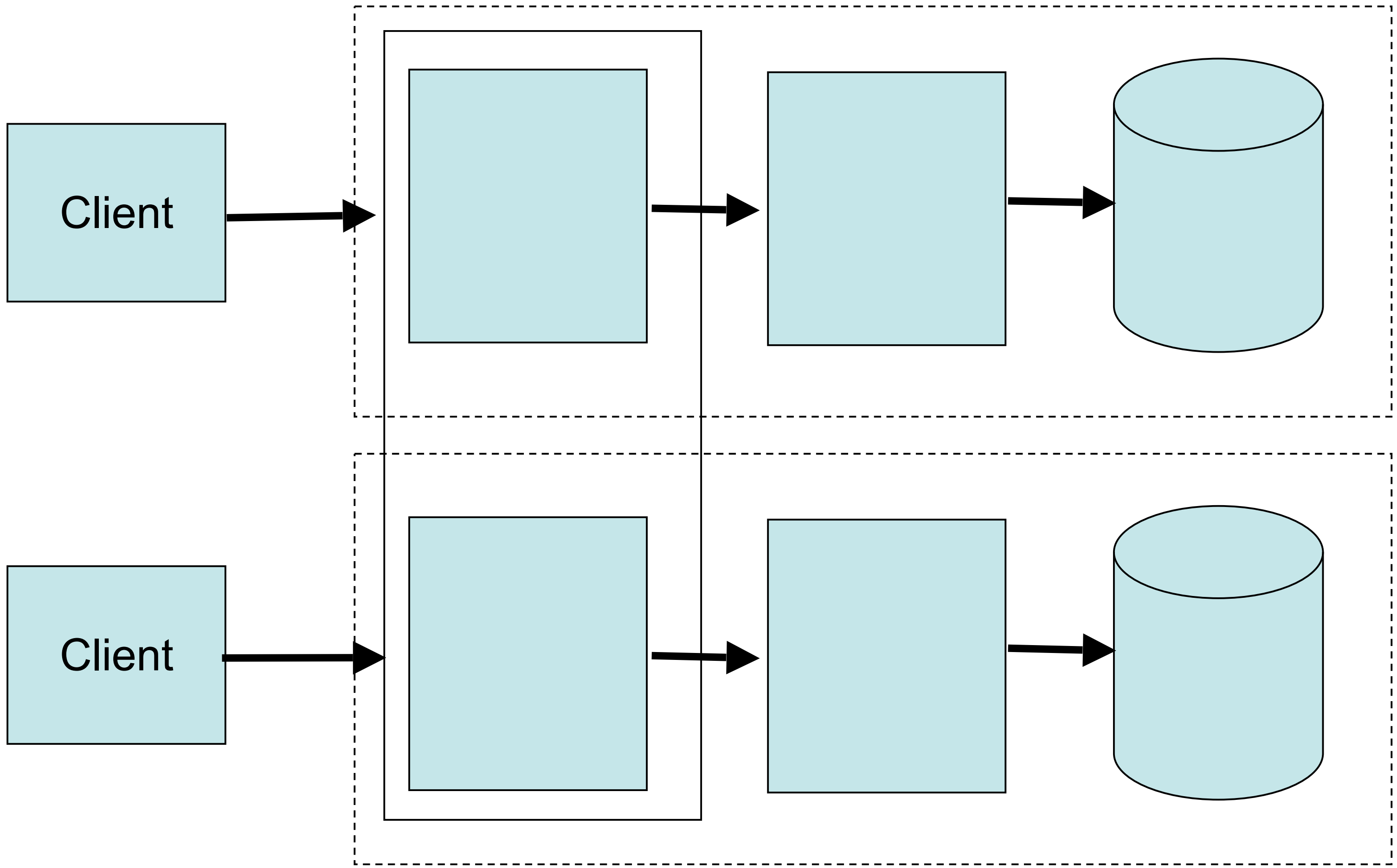
Layered


WS/AS replication
Middleware-based
Database Replication
Client
Client

Limitations

- Called tier must be aware of replication of calling tier to some degree
- Exactly-once at called tier does not guarantee that exactly-once is possible at local tier

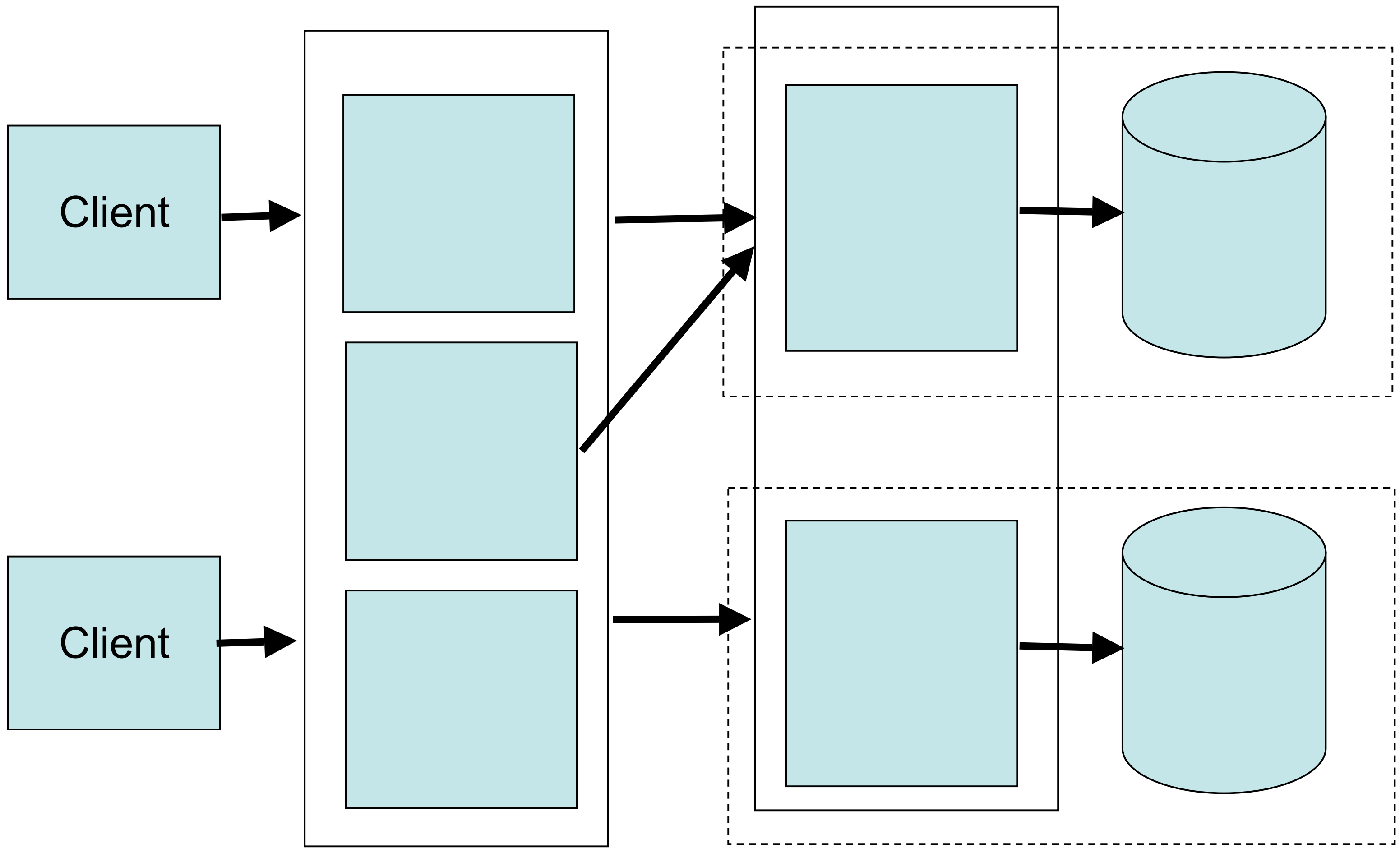
Vertical Replication



Ideas and Challenges

- Idea
  - Only the first tier runs replication algorithm
  - Other tiers are used as block box
  - Are not aware of any replication
  - In case of failure, one vertical partition fails
- Challenges
  - Load-balancing if load of individual tiers differs

Combination



Summary

- Replication crucial for
  - Fault-tolerance and scalability
- AS and DBS replication different requirements
- 3 protocol types needed
  - Execution and coordination during normal processing
  - Failover (termination)
  - Recovery