Experiences in Engineering Active Replication into a Traditional Three-tiered Client-server System

Gabriel L. Zenarosa

gzen@cs.cmu.edu

Joint work with Soumya Simanta

Special thanks to Jinhee Lee, Luis Maya, and Min Wang

Outline

- Project Context
- Baseline System Overview
- Towards a Petri Net Model
- Petri Net Model Analysis: Discovering Key Design Elements
  - Replication Manager
  - Global Naming Service
  - Multicast Messaging
  - Voting Policies
  - Atomic Transactions
  - Idempotent Operations
- Conclusion
- Questions

Background

CMU Course 18-749: Fault-tolerant Distributed Systems

- "Learning by doing"

Course Goal

- Understand—***tangibly***—the tradeoffs in system qualities
  - Fault-tolerance (i.e., primarily middle ***logic tier*** node crashes)
  - Real-time responsiveness
  - High-performance

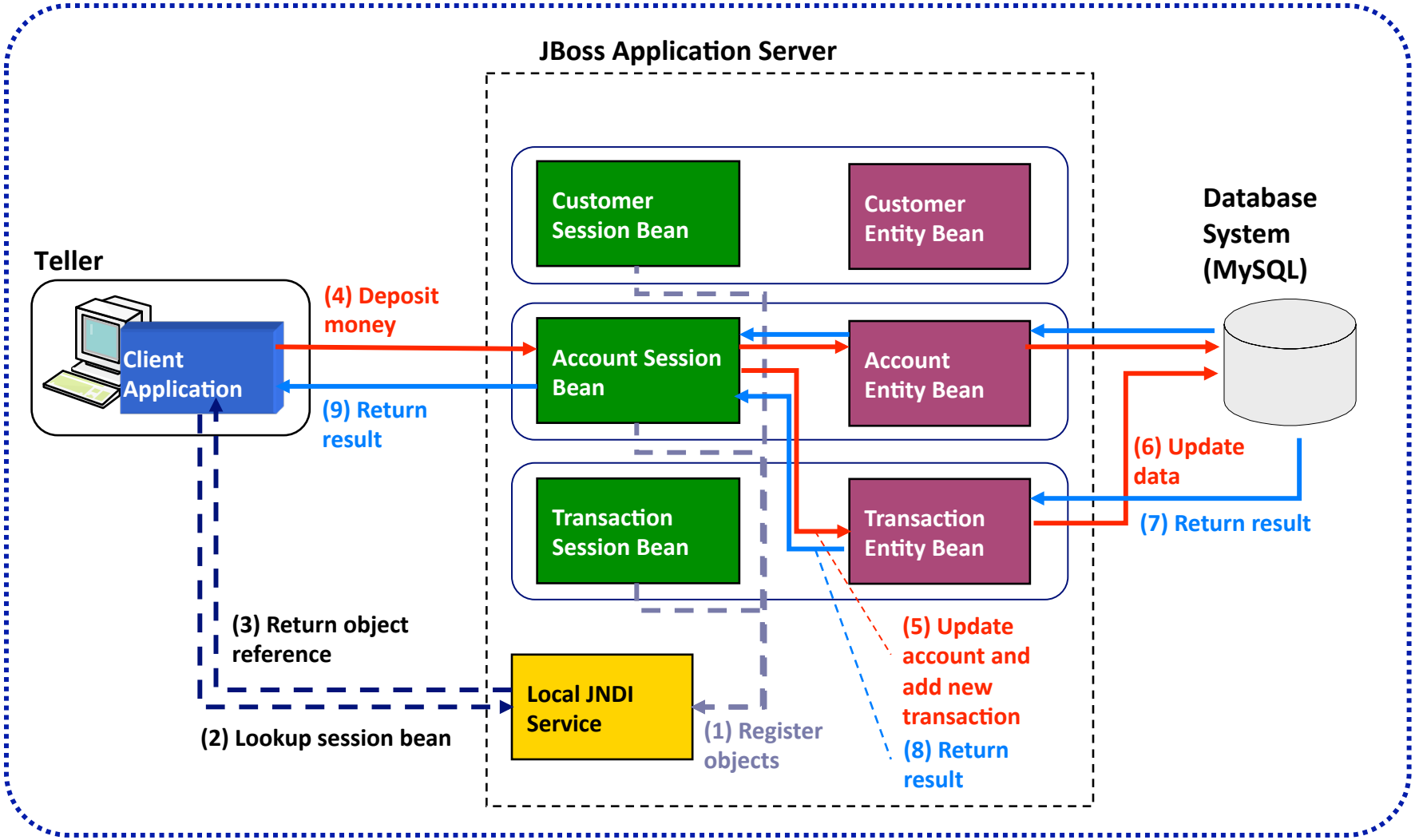
Course Project Scope

- Select a baseline three-tier distributed application
- Evolve it to meet some quality requirements using ***learned*** tactics

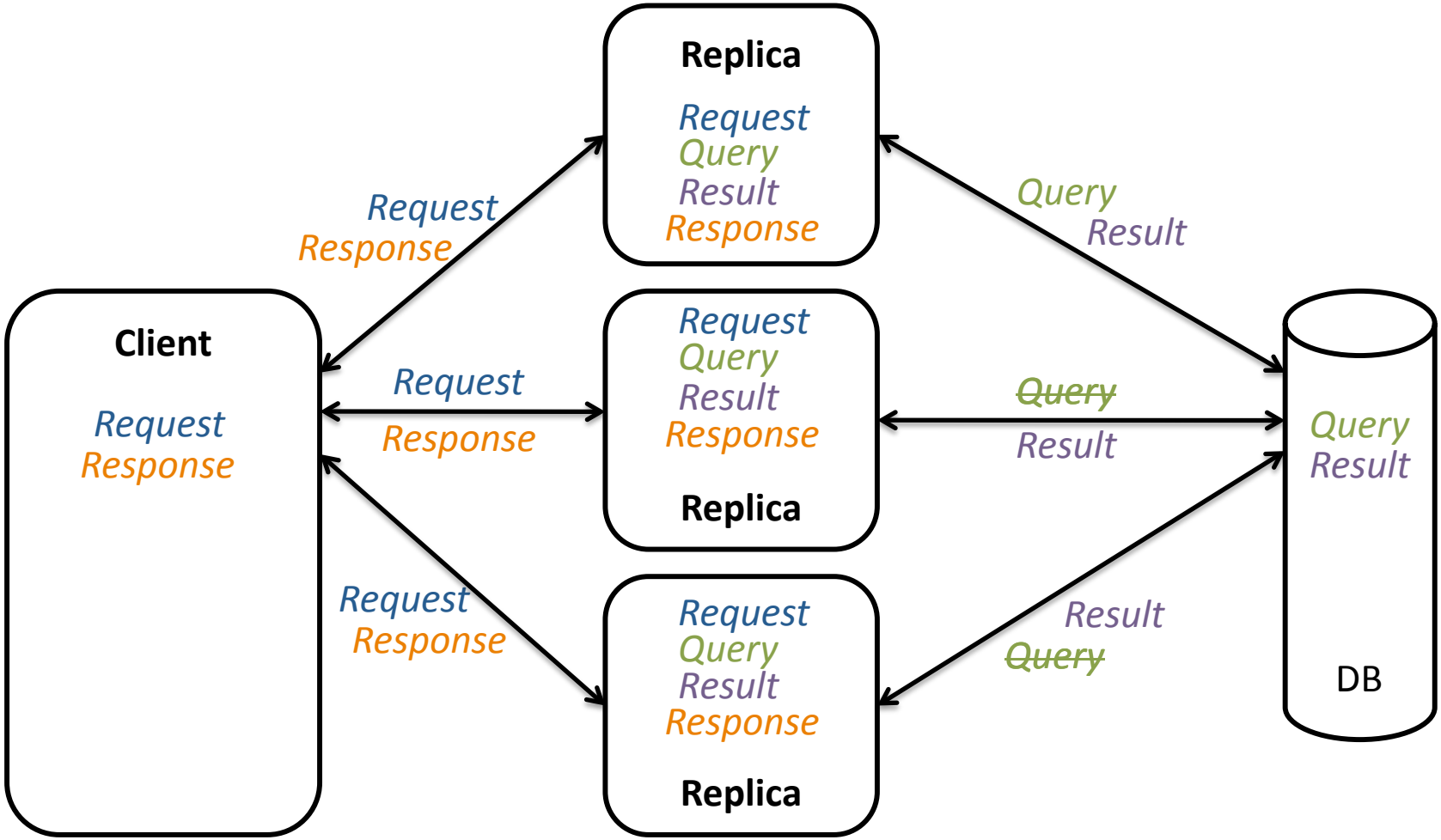
Project Pre-selections

- Baseline System
  - Simplified banking system
- Technologies
  - Enterprise JavaBeans (EJB)
- Constraints
  - Cluster of quad single-core Intel Xeon 2.8 GHz microprocessors, 2 GB of memory, and Linux kernel 2.6
- Initial Tactics
  - Active replication

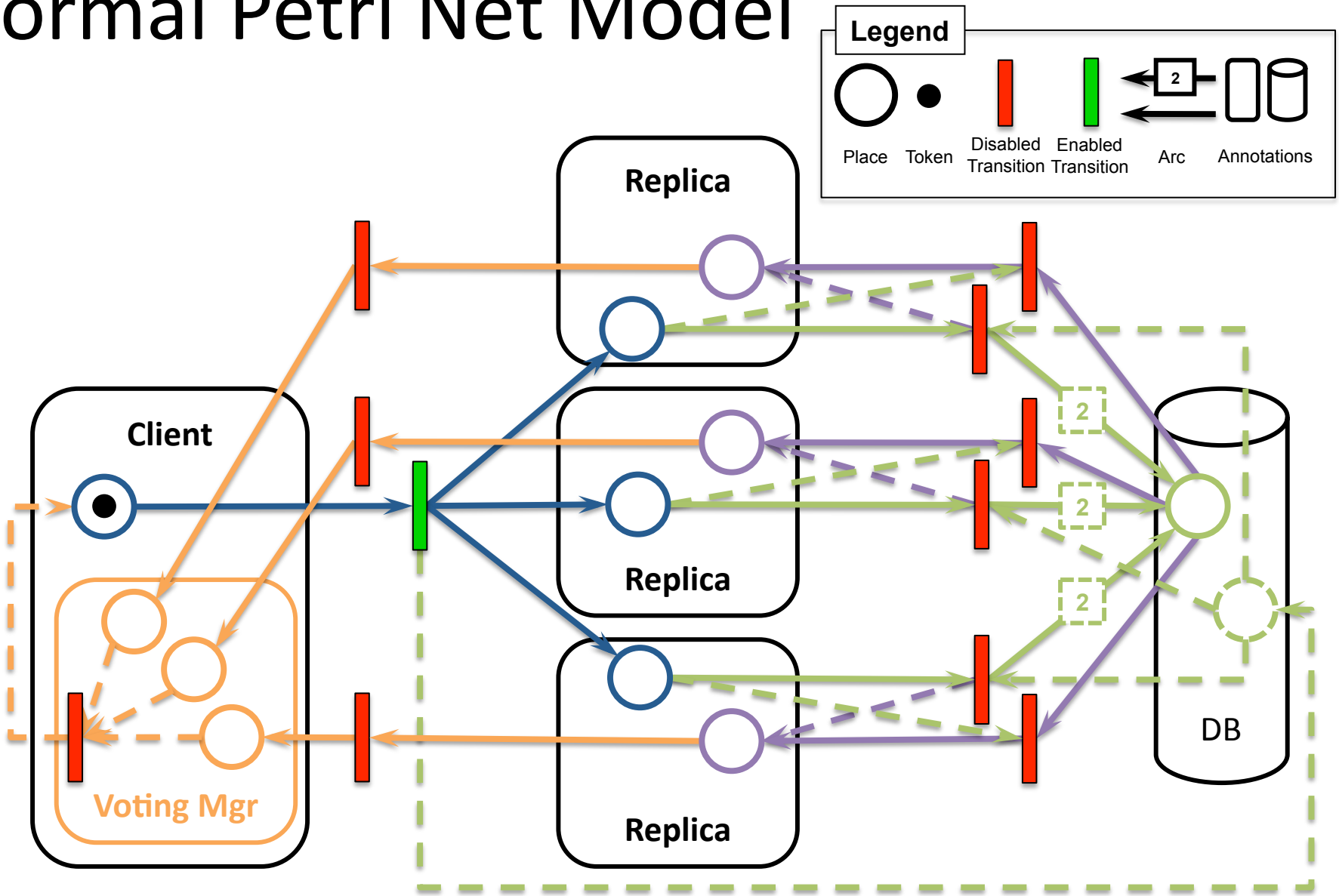
Base 3-tiered Client/Server Schematic



Informal Design for Active Replication



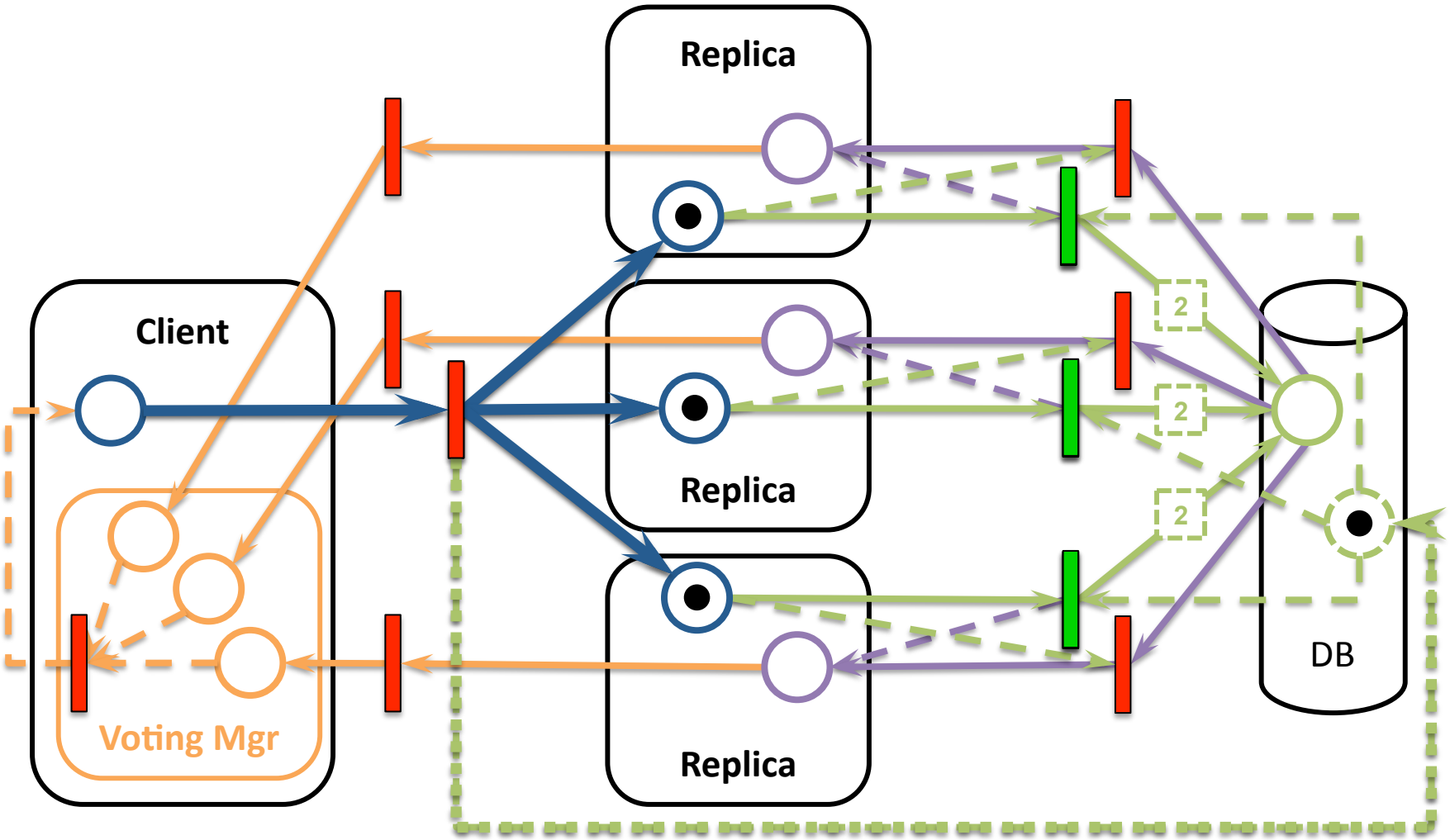
Formal Petri Net Model



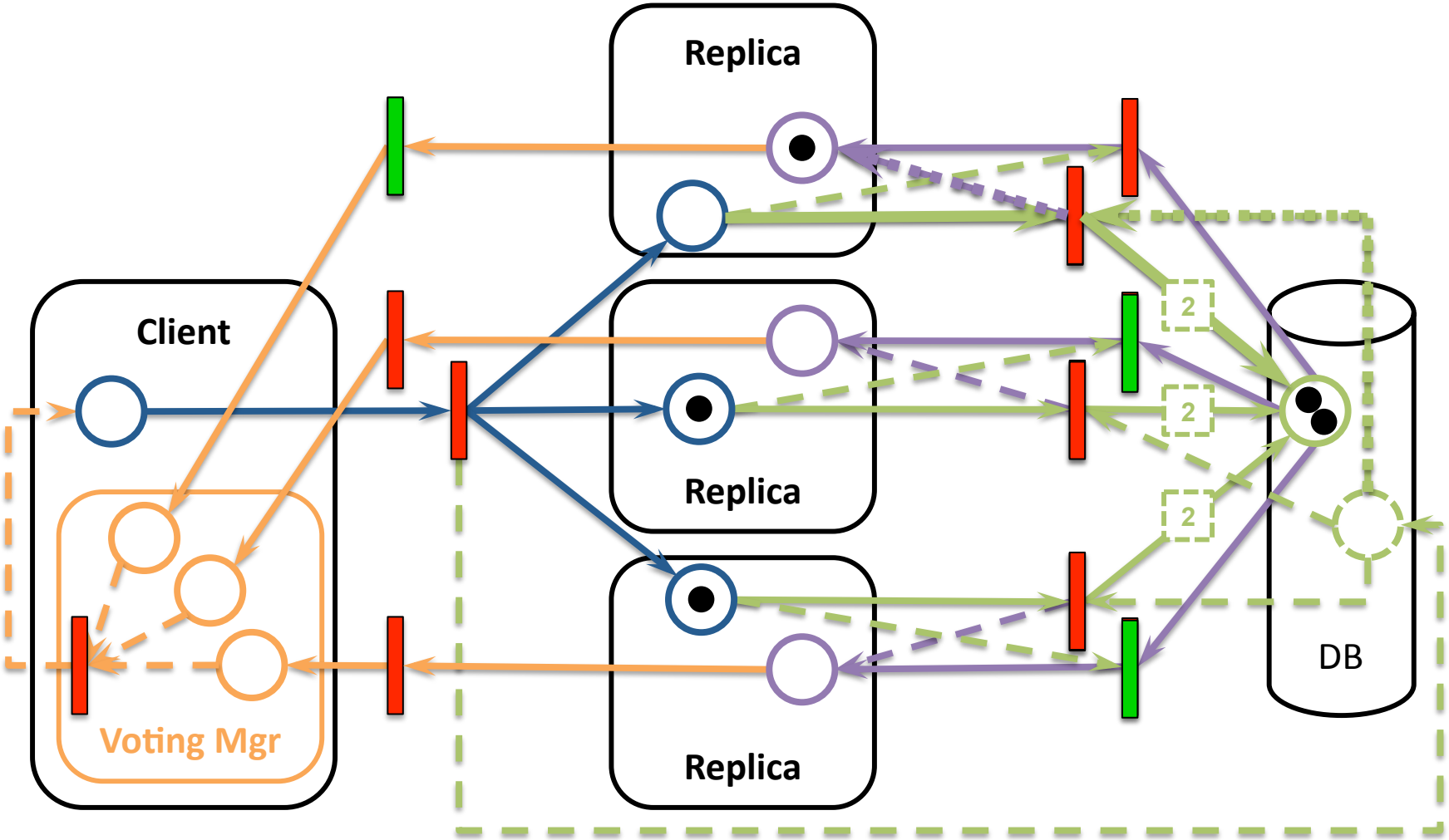
Petri Net Model Simulation

End-to-end Walkthrough

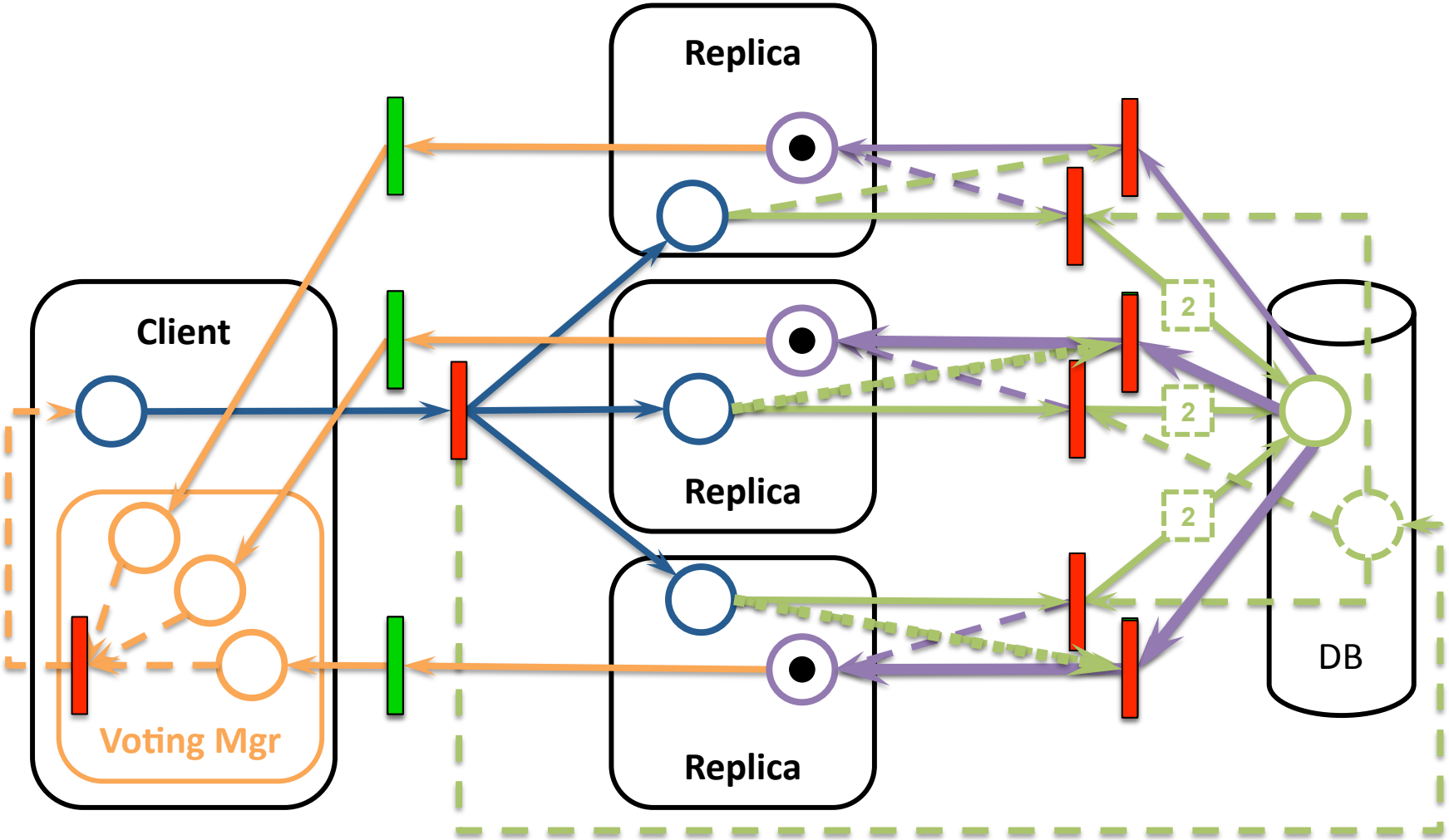
1. Client Sends a Request



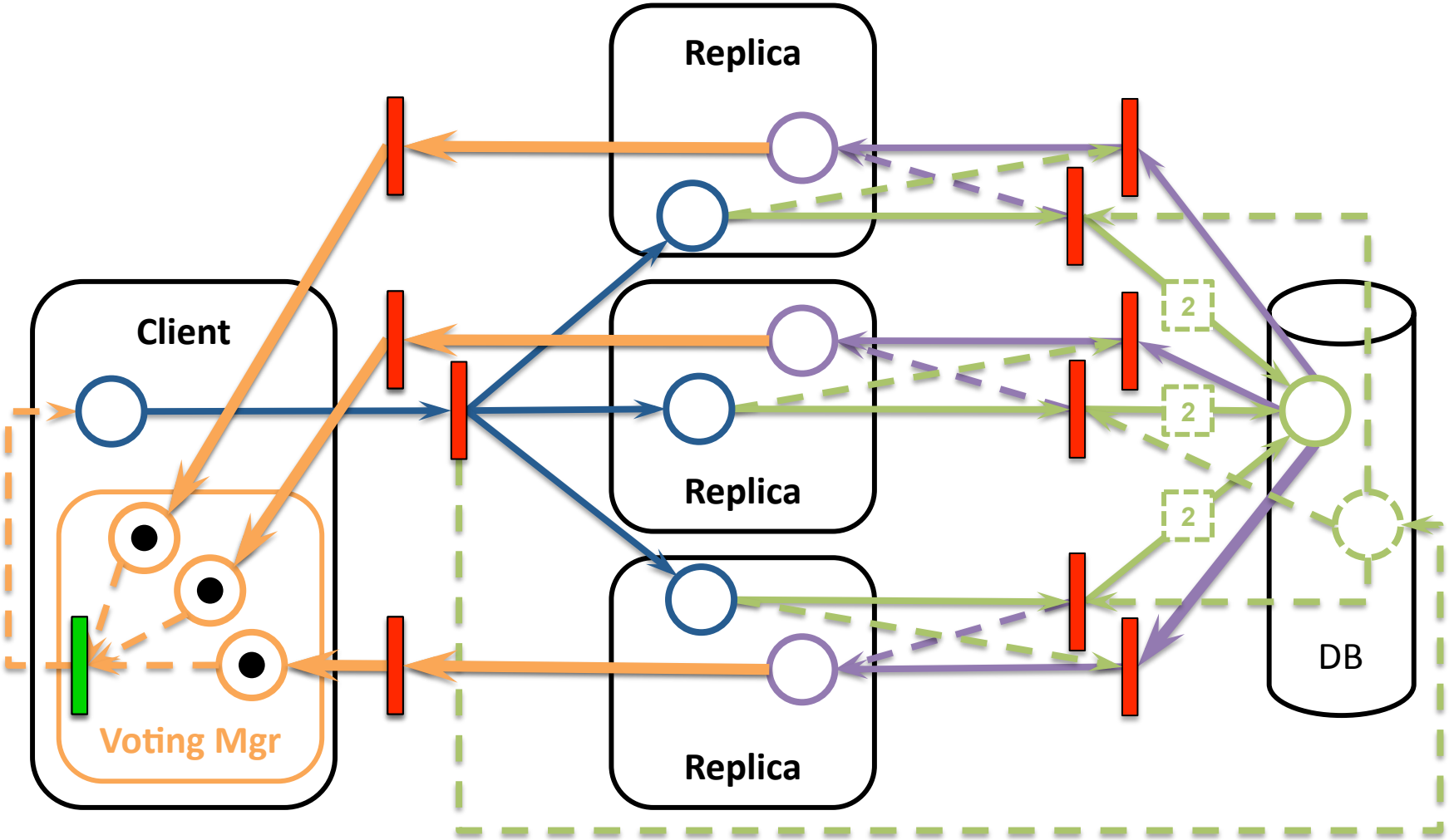
2. First Replica Executes Normally



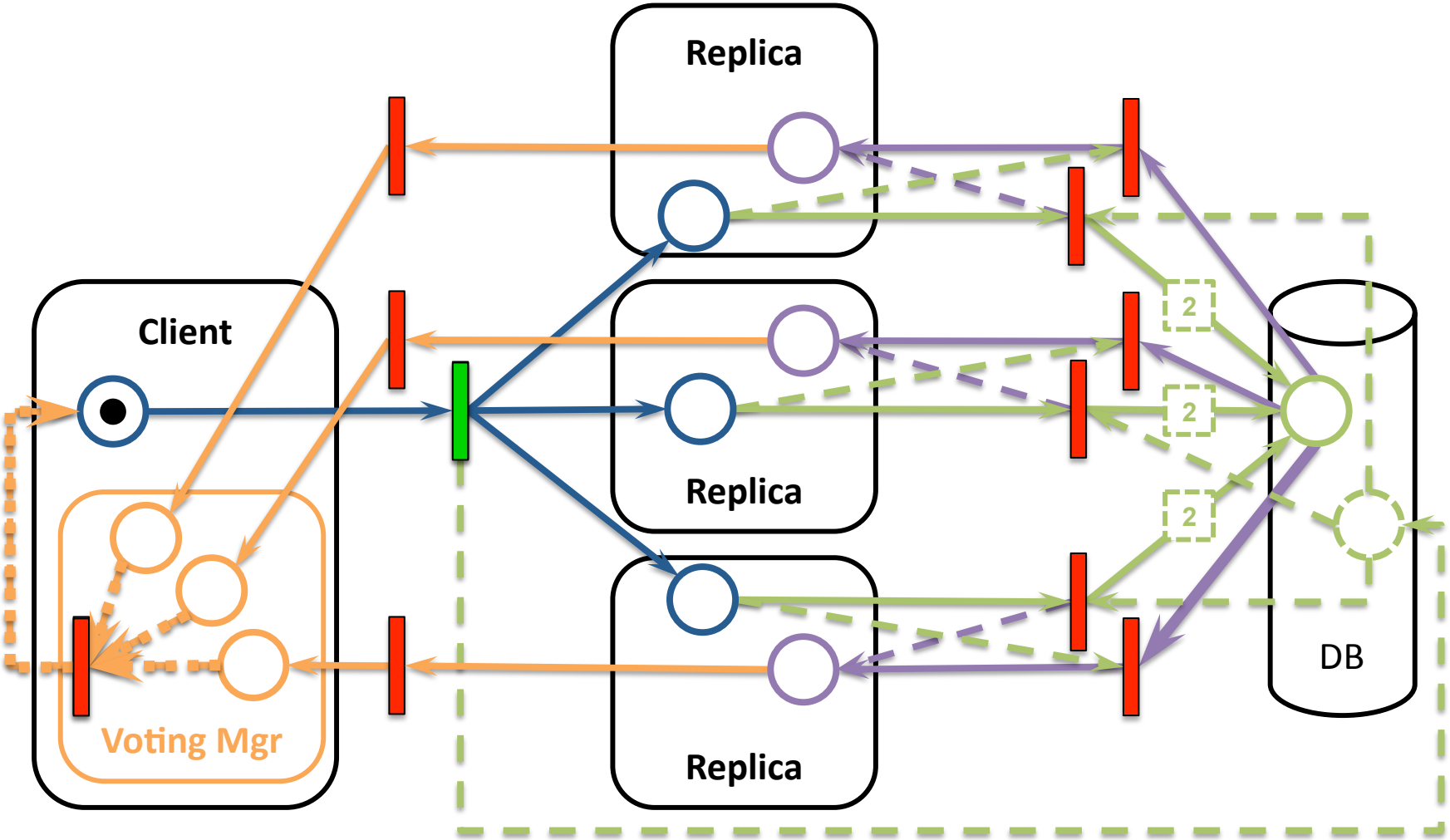
3. Other Replicas Obtain Copies



4. All Replicas Return their Responses



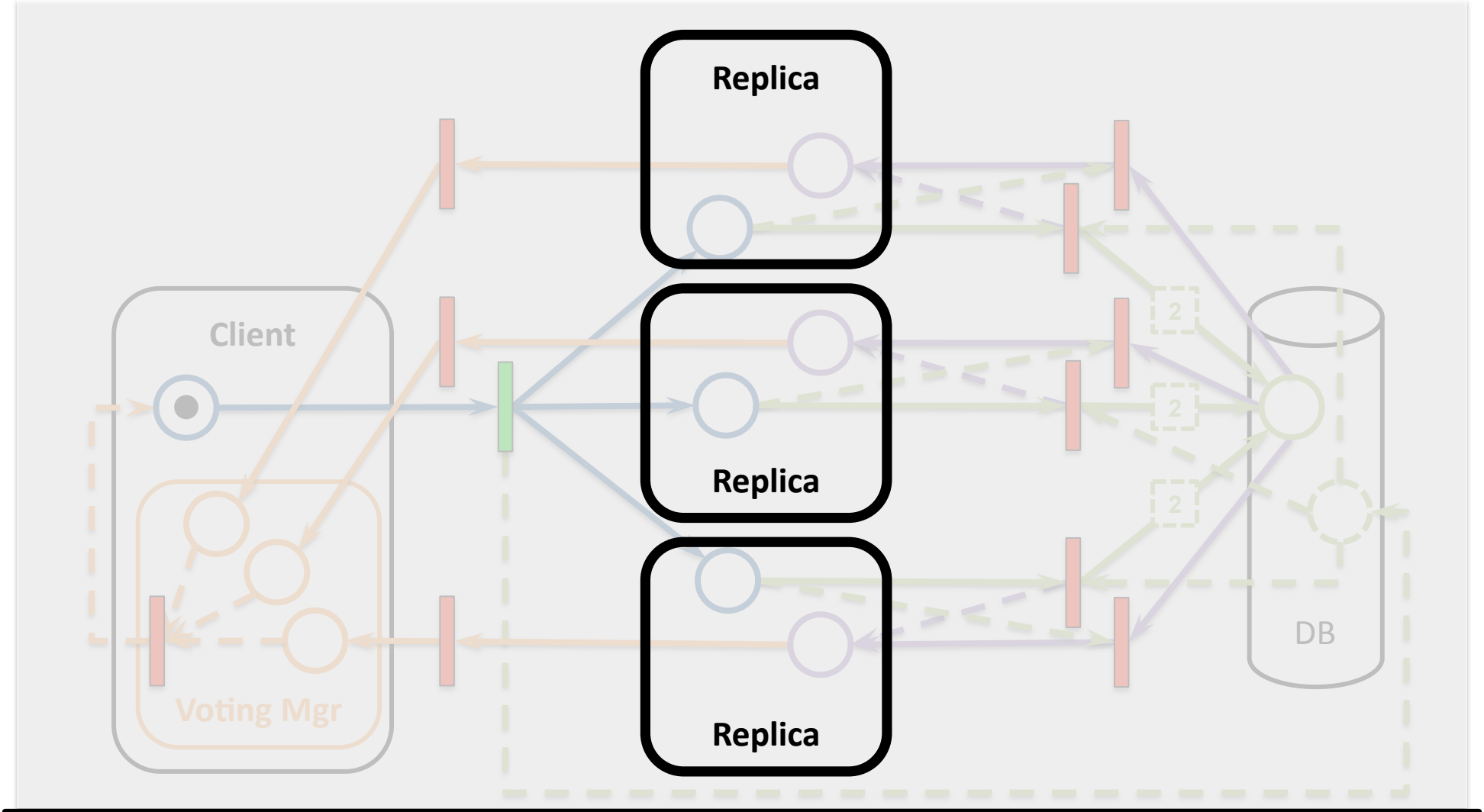
5. Voting Manager Collates



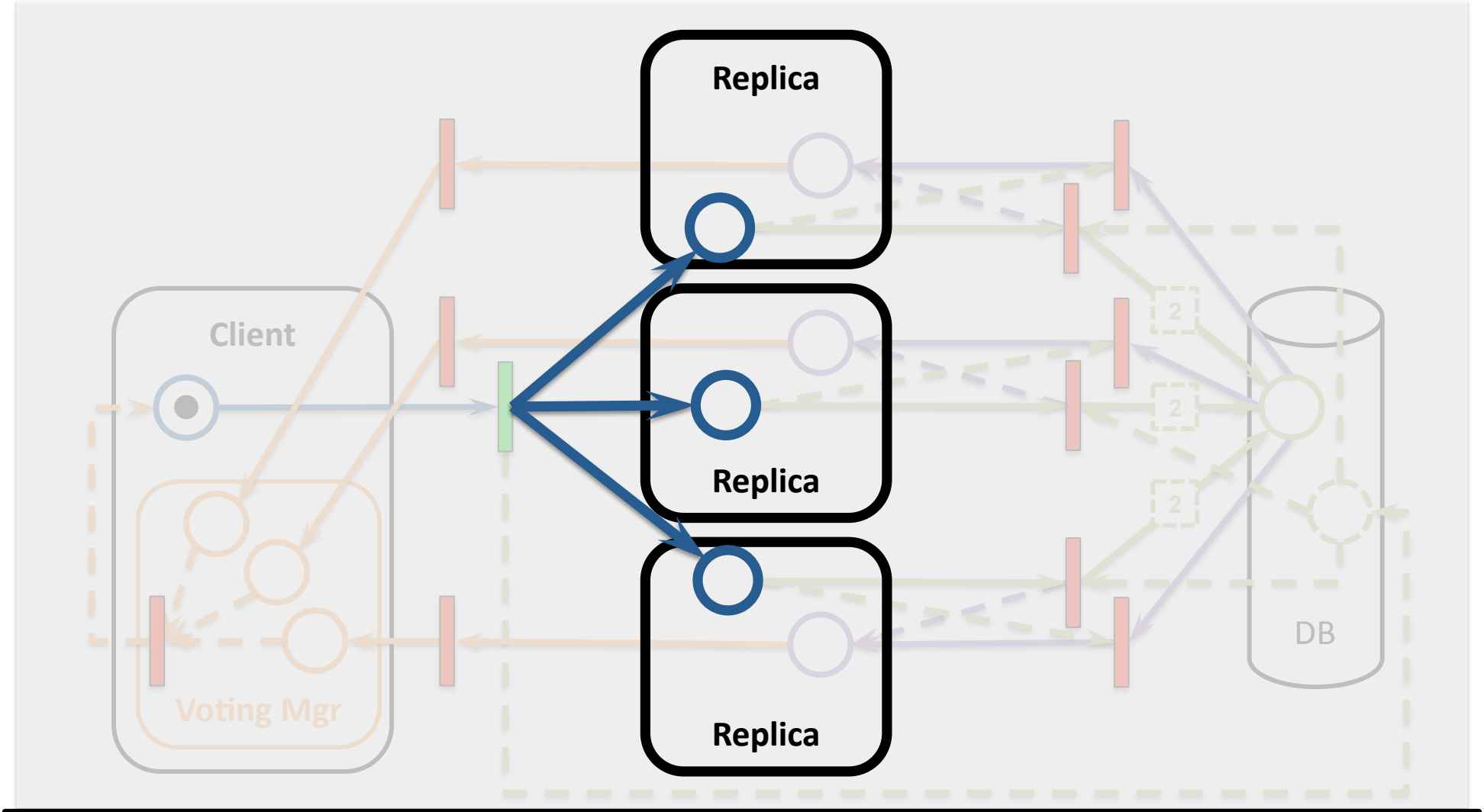
Petri Net Model Analysis

Determining Key Design Elements

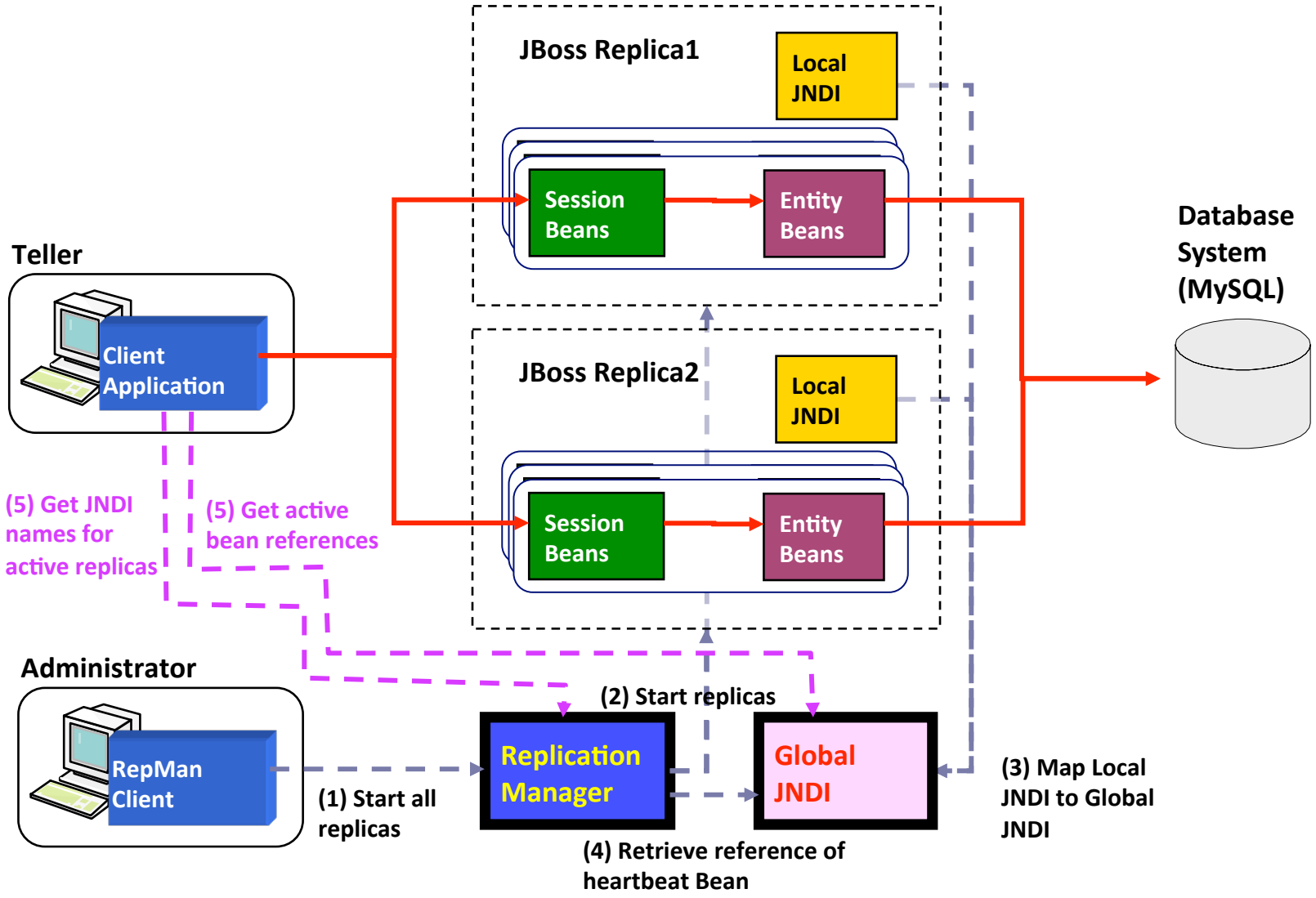
Element 1: Replication Manager



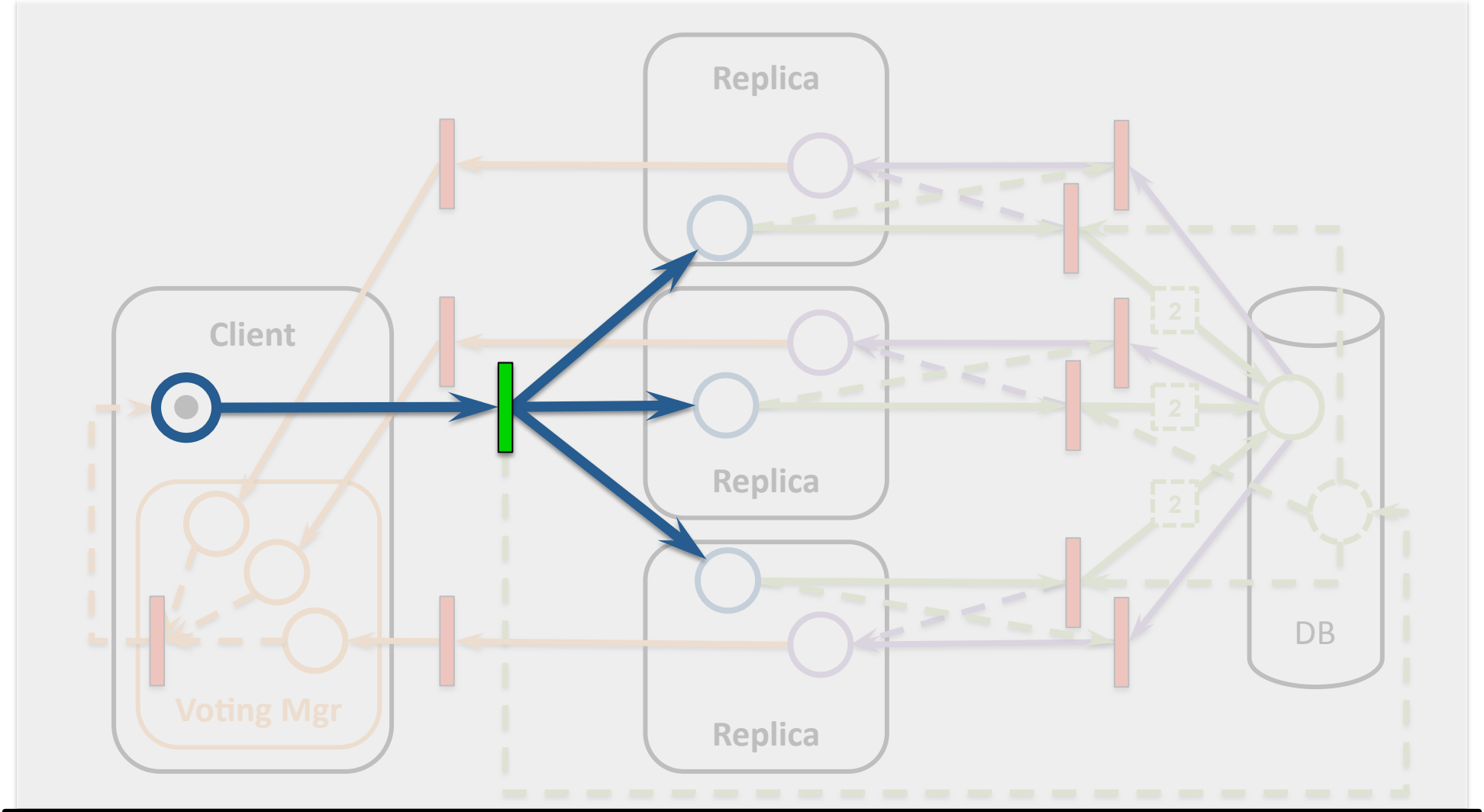
Element 2: Global Naming Service



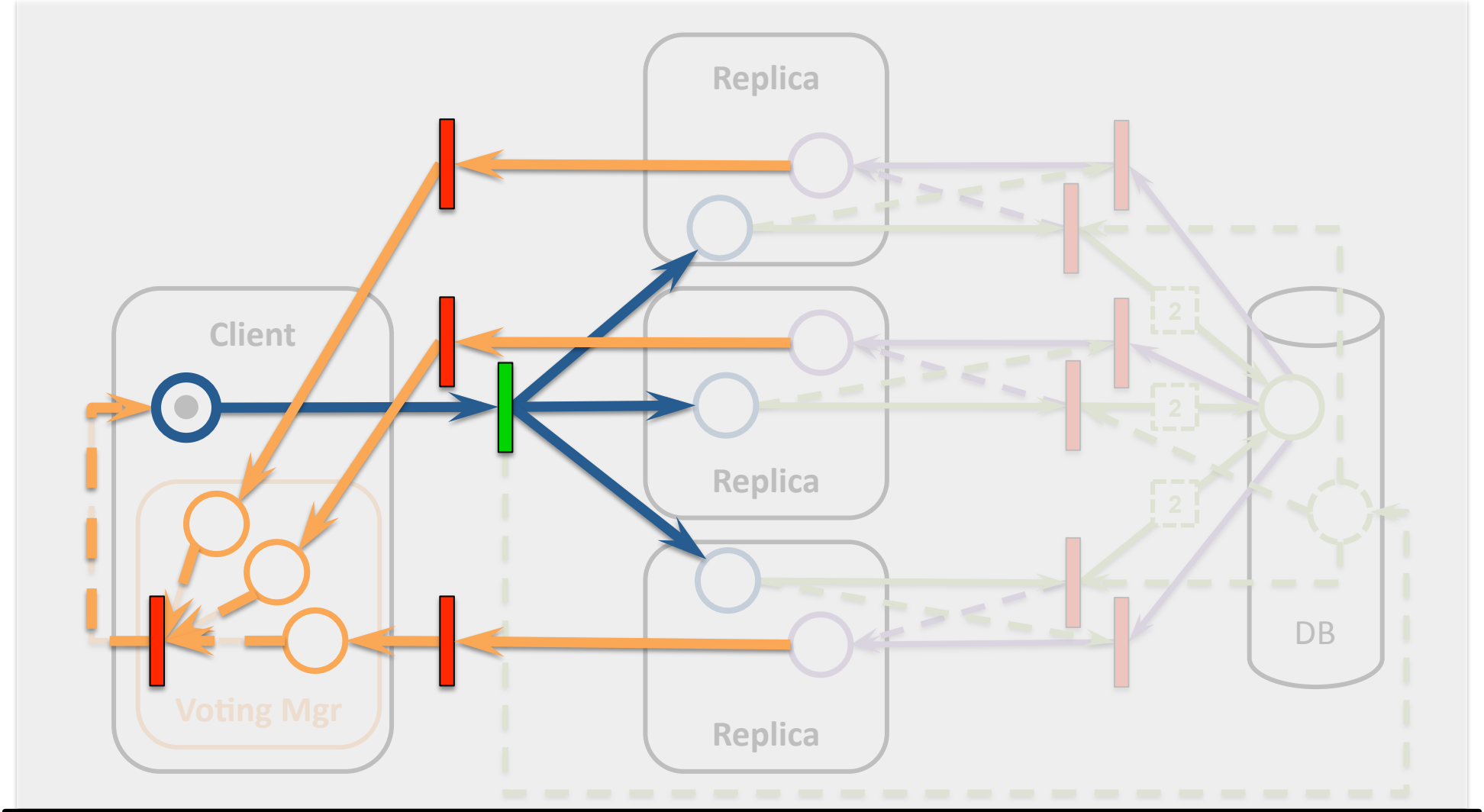
Implementation Schematic: *Elements 1 & 2*



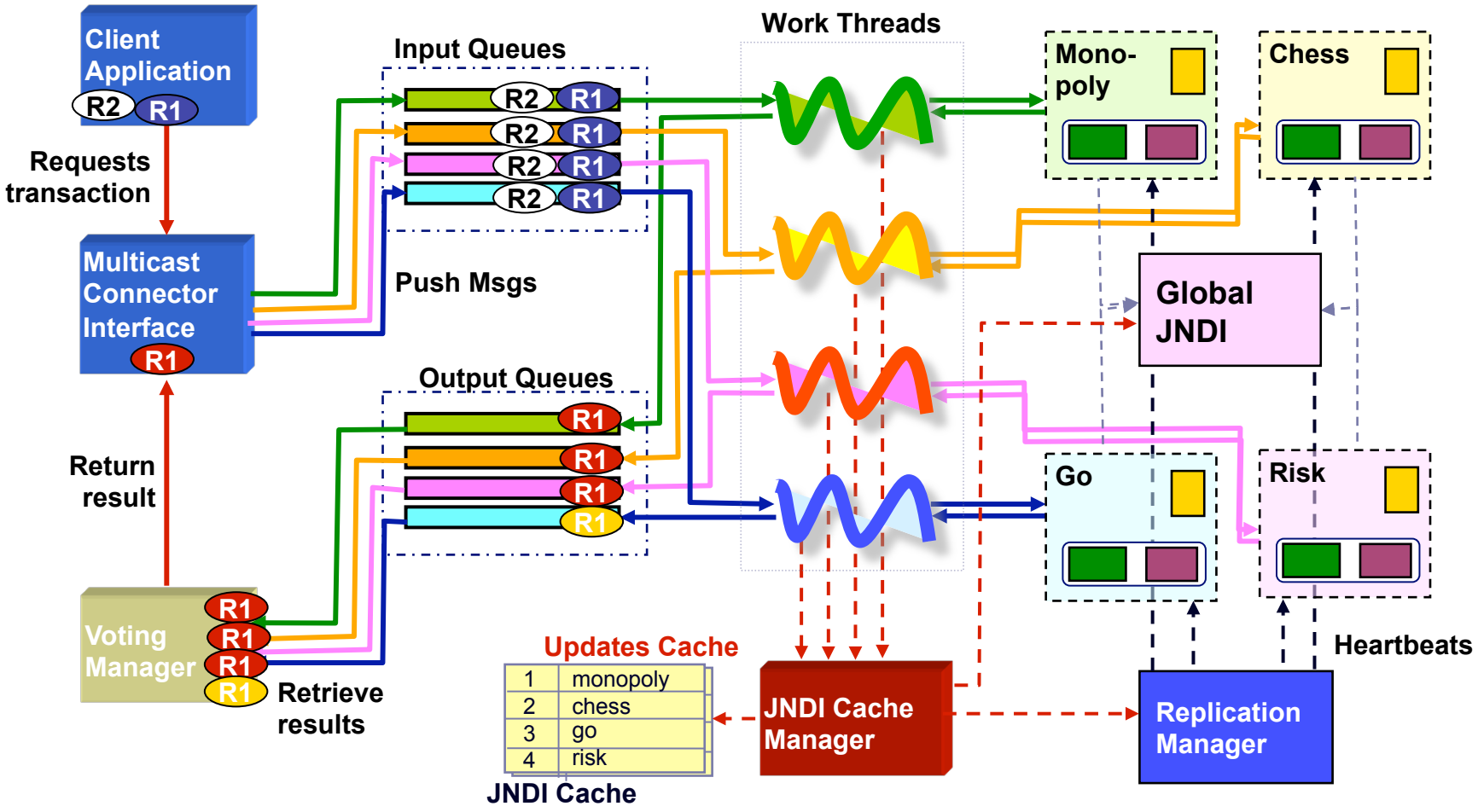
Element 3: Multicast Messaging



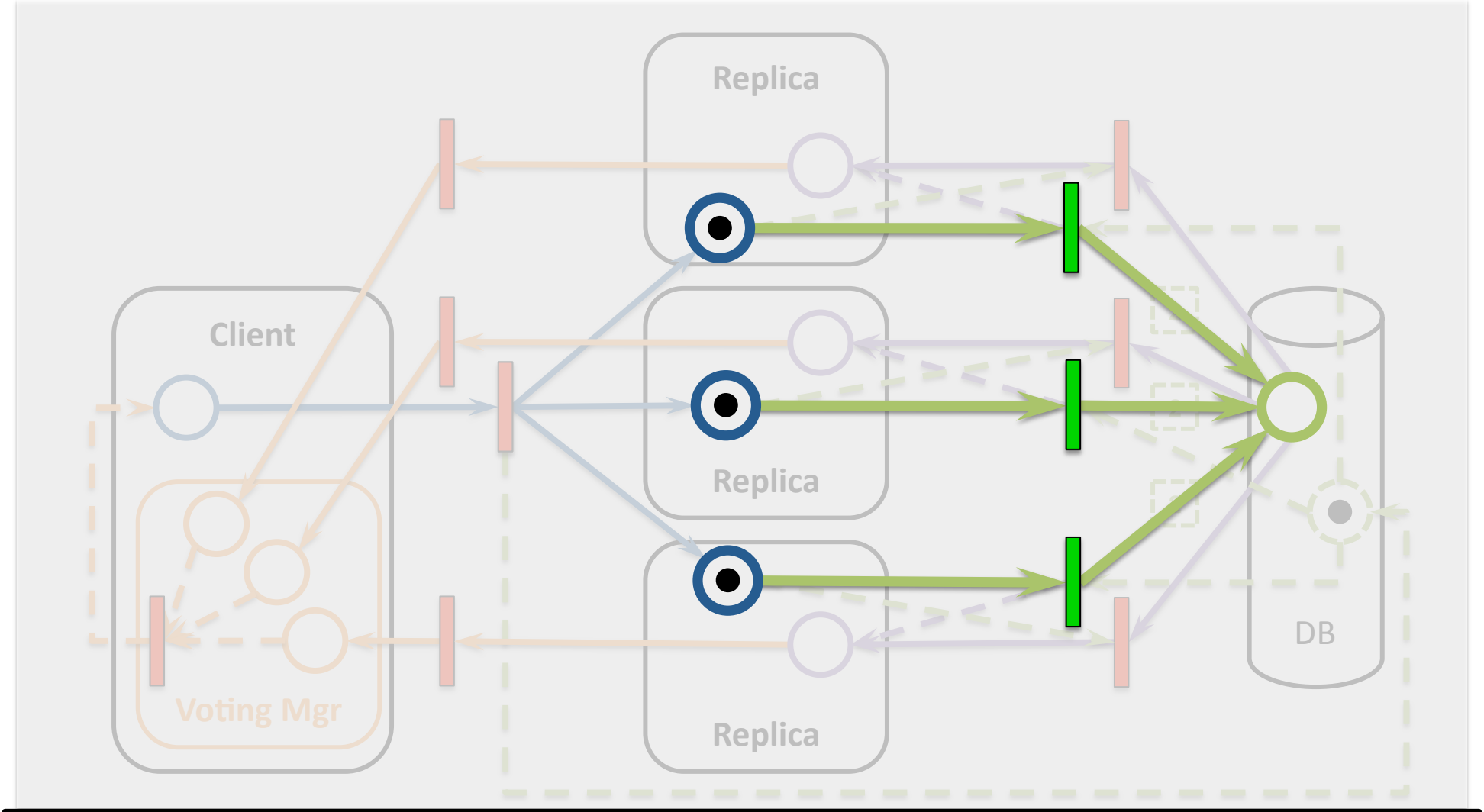
Element 4: Voting Policies



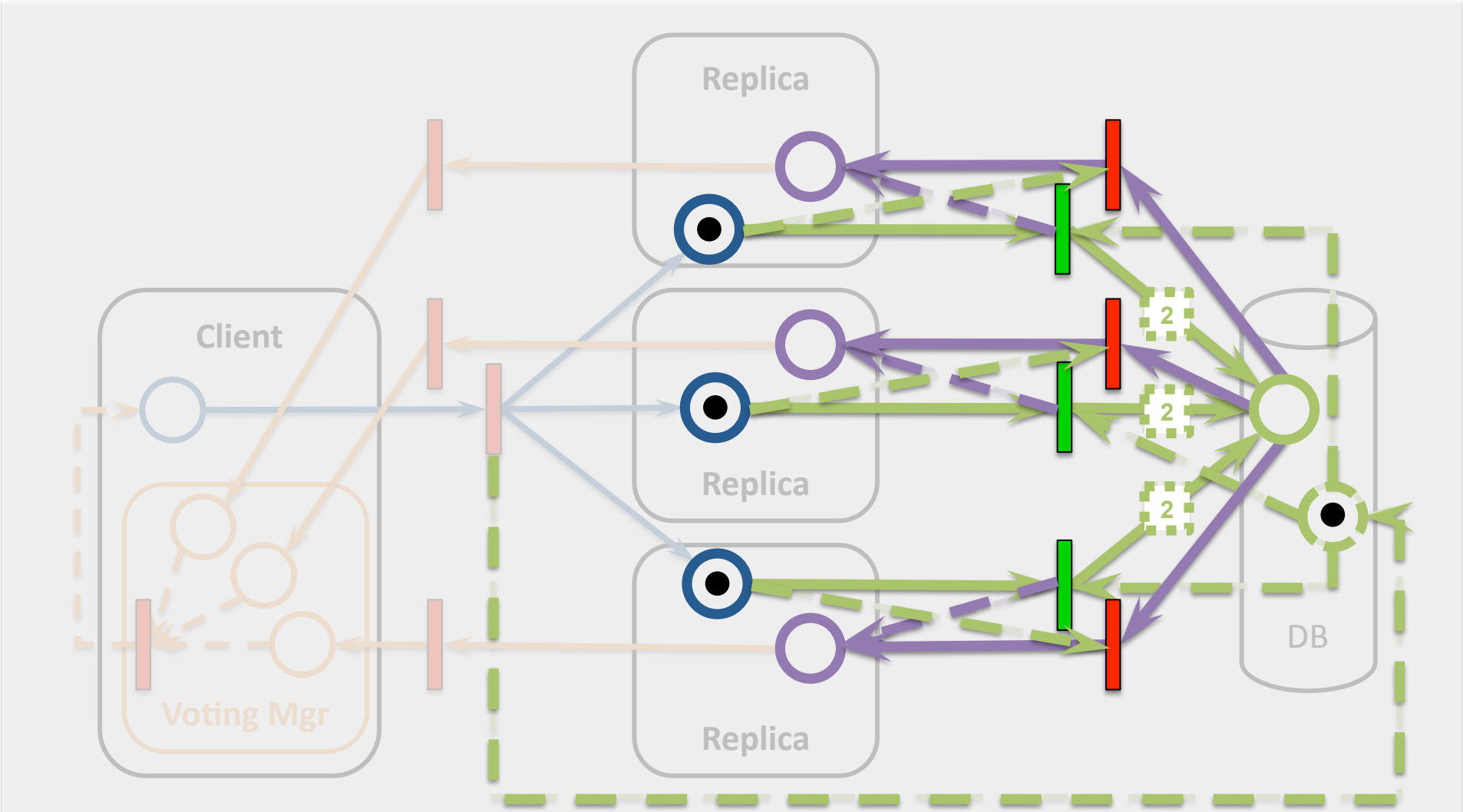
Implementation Schematic: *Elements 3 & 4*



Element 5: Atomic Transactions



Element 6: Idempotent Operations



Sequence Numbers

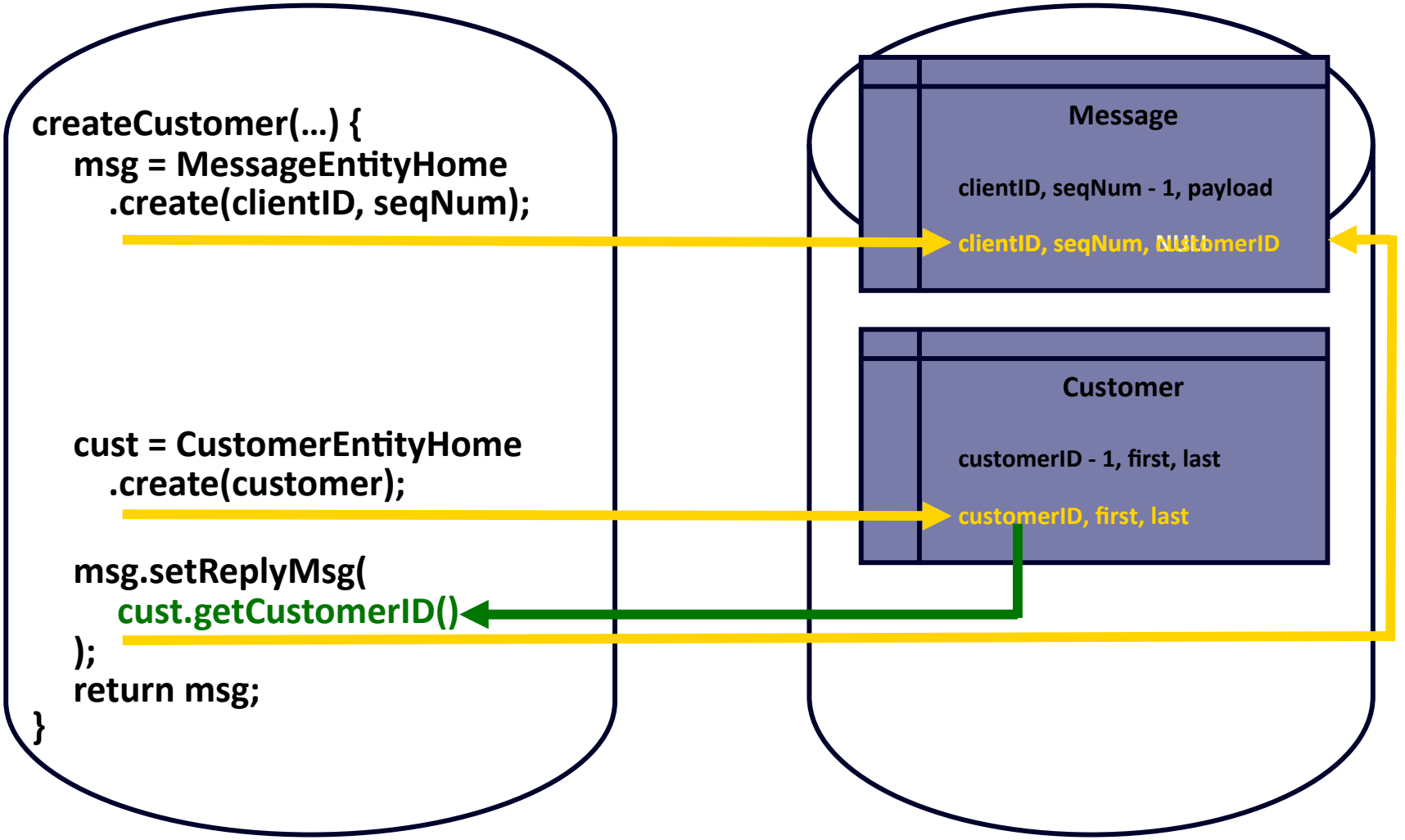
- Clients tag each request with a sequence number
  (in addition to their client IDs)

Client ID	Seq #	Request Message Payload

- Data nodes store the responses in a results table, keyed on the sequence number (and client ID)
- Server nodes returns responses with the same sequence number (and client ID)

Client ID	Seq #	Response Message Payload

First Replica Processing a Request



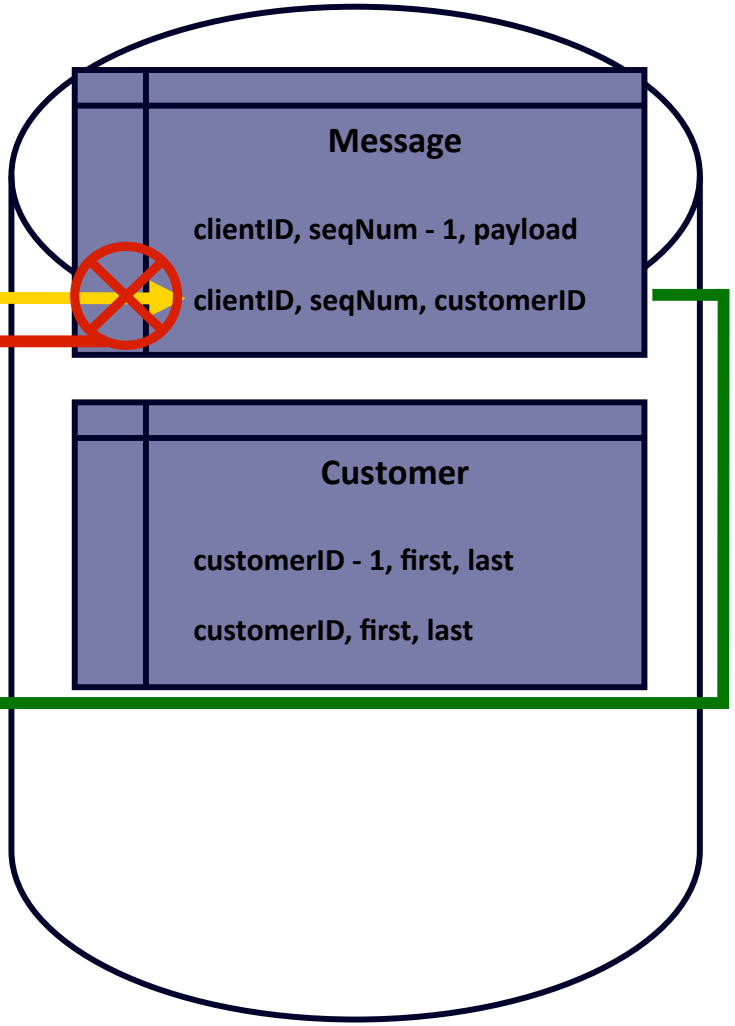
Next Replicas Reprocessing a Request

```
createCustomer(…) {
    msg = MessageEntityHome
       .create(clientID, seqNum);

    catch (CreateException ce) {
       if (ce instanceof
          DuplicateKeyException ||
          ce.getMessage().indexOf(
             "SQLException: Duplicate"
          ) != -1) {

       msg = MessageEntityHome
          .findByKey(clientID, seqNum);

       return msg;
    }
}
```



Conclusion

Summary and Lessons Learned

Summary

- Project Context: Background, Pre-selections, and Baseline
- Informal Sketching, Petri Net Modeling, and Refinement
  - Key Design Elements
    - Replication Manager
    - Global Naming Service
    - Multicast Messaging
    - Voting Policies
    - Atomic Transactions
    - Idempotent Operations

"*Our*" Lessons Learned

- The formal model
  - allowed for the early identification of active replication key elements for a three-tiered client-server system;
  - paved the way for a downstream design and implementation of those elements.
- The methodology enabled the engineering of an end system with a good balance in fault-tolerance, real-time responsiveness, and performance.

Questions

Gabriel L. Zenarosa

gzen@cs.cmu.edu