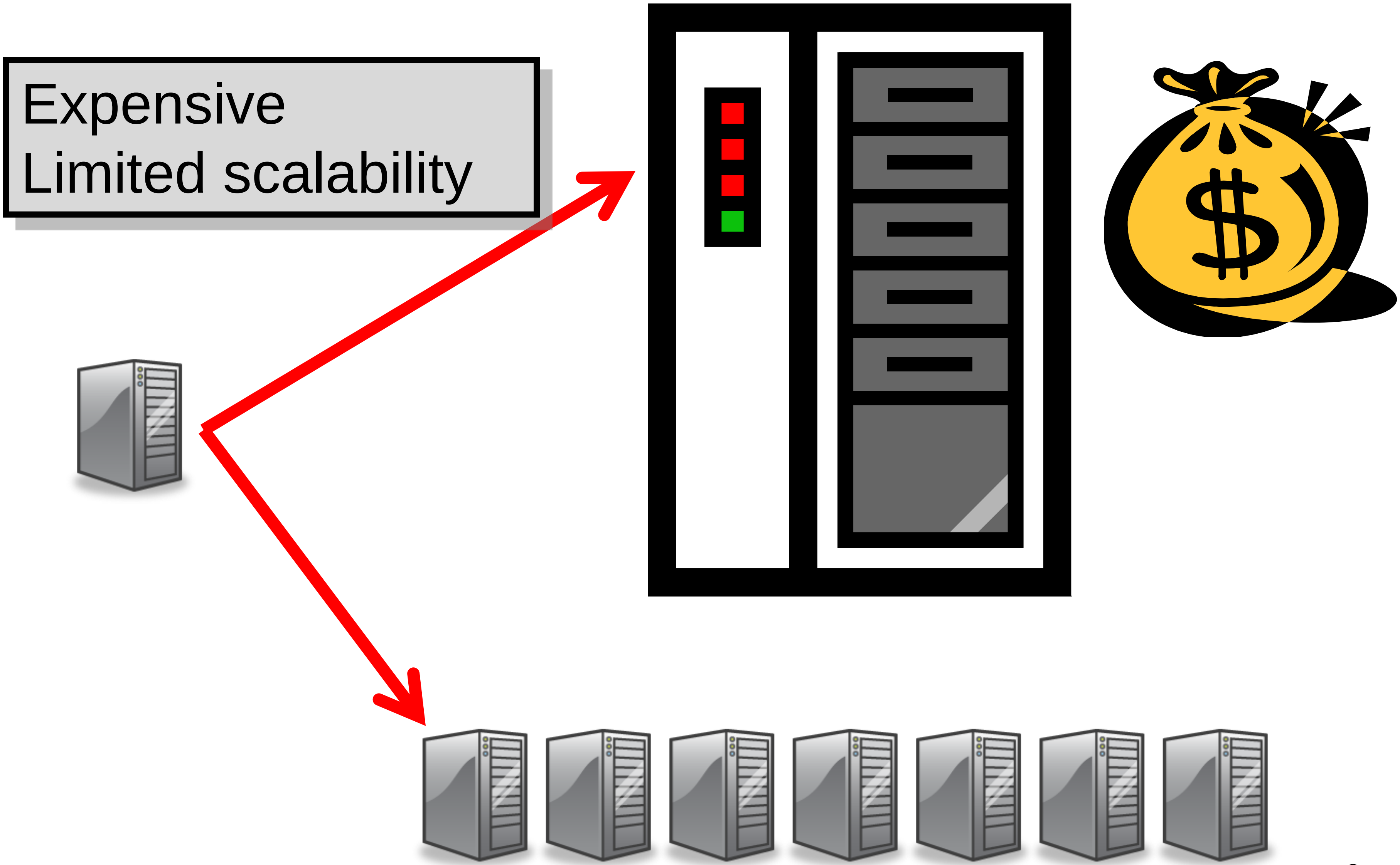
Database Replication in Tashkent

CSEP 545 Transaction Processing

Sameh Elnikety

Replication for Performance



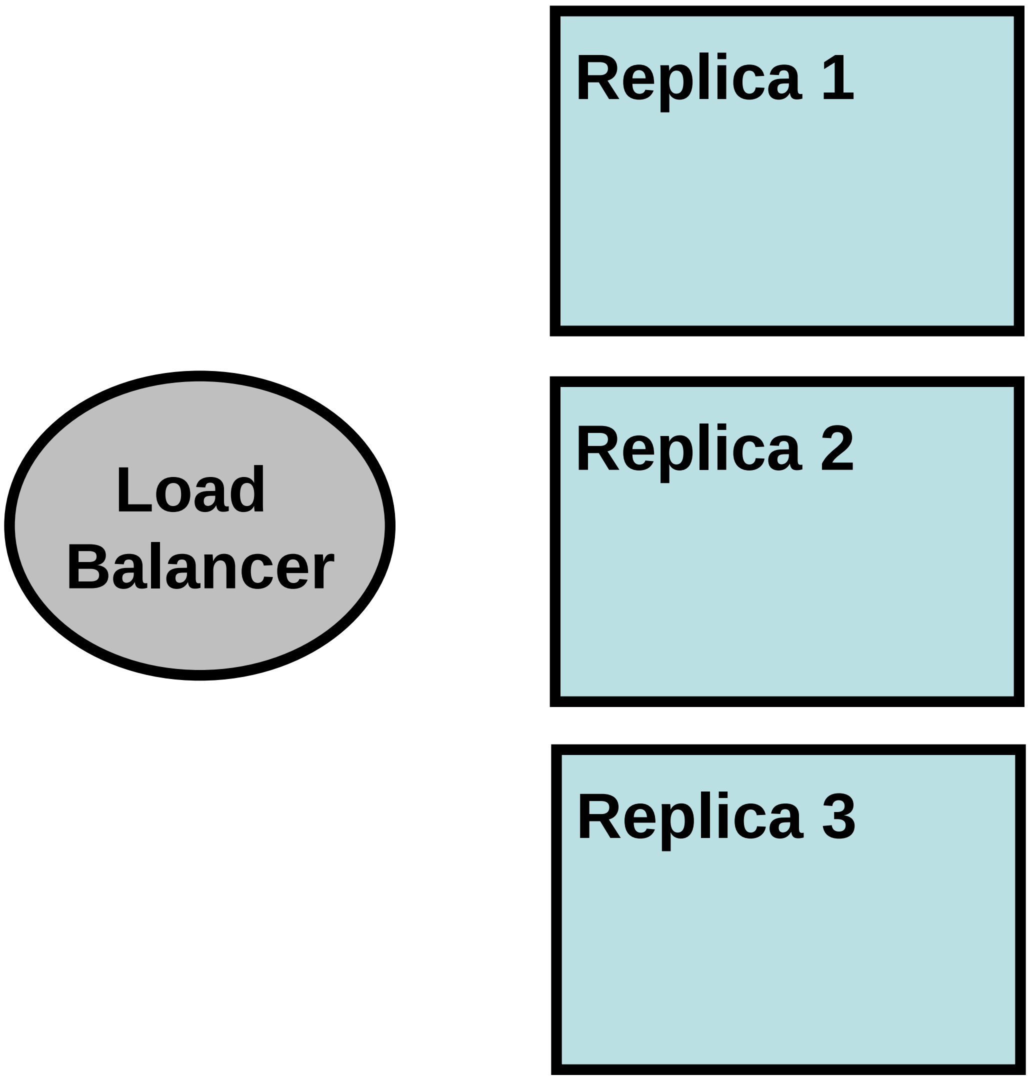
DB Replication is Challenging

- Single database system
  - Large, persistent state
  - Transactions
  - Complex software
- Replication challenges
  - Maintain consistency
  - Middleware replication

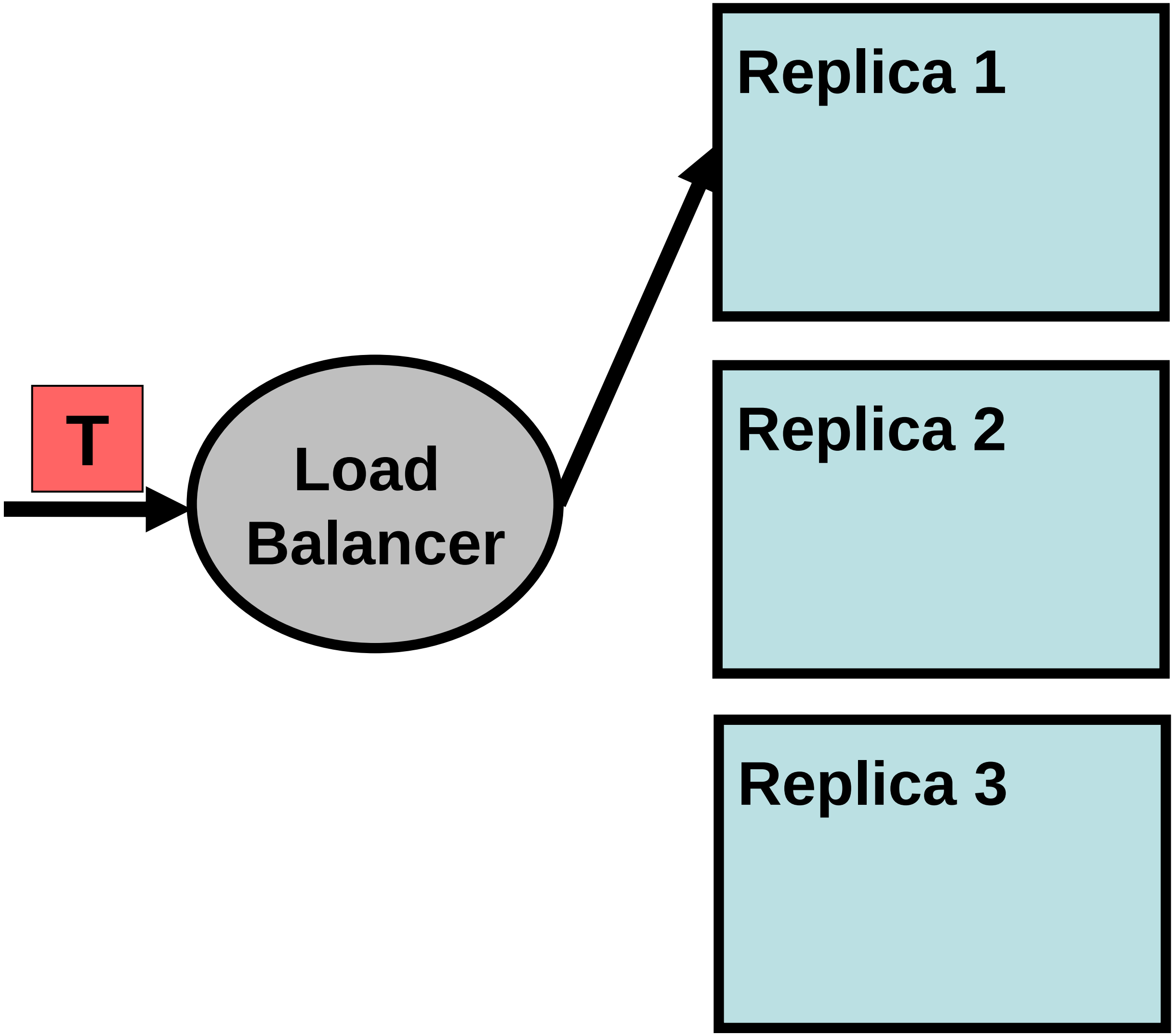
Background

Standalone DBMS

Background

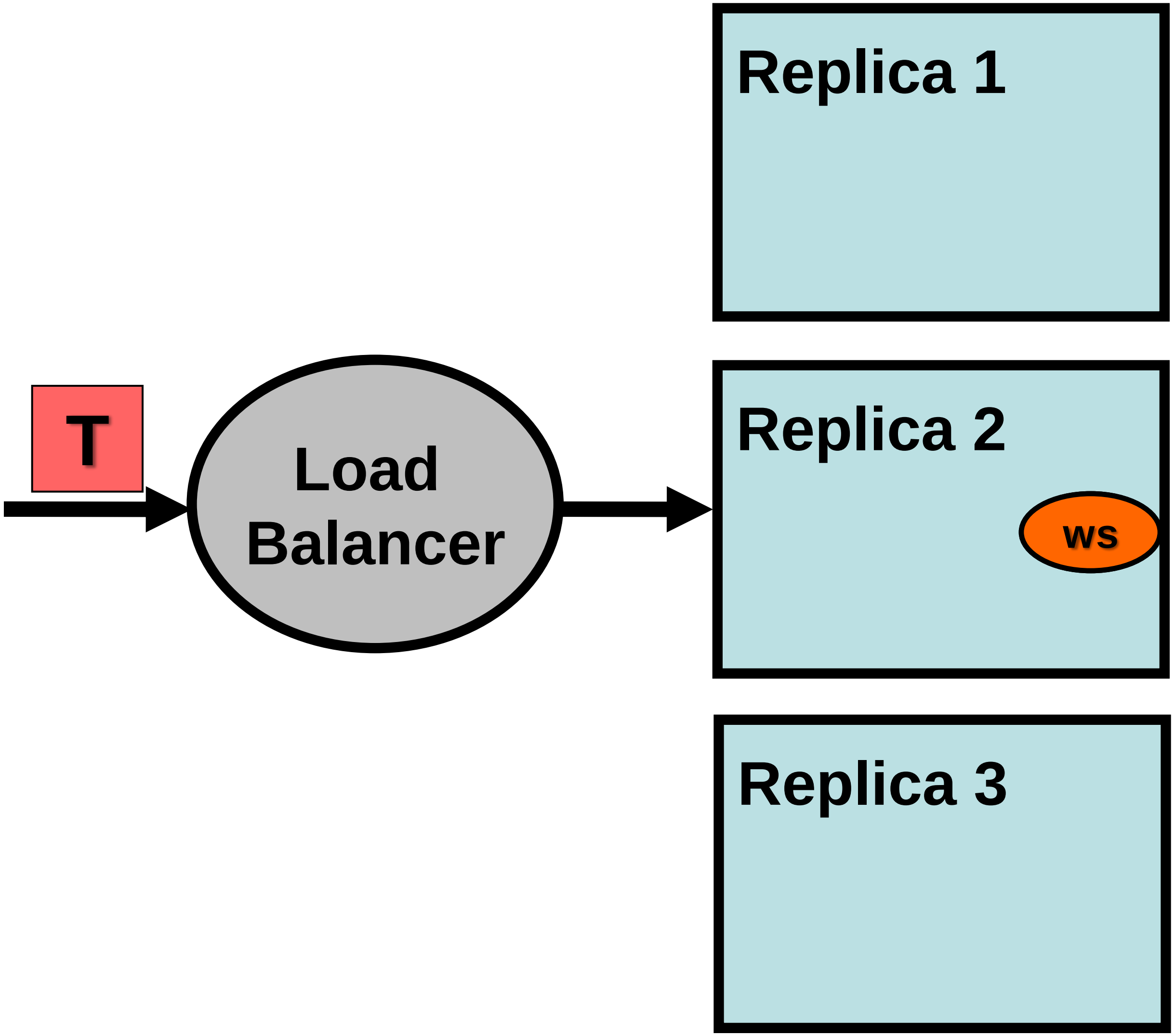


Read Tx



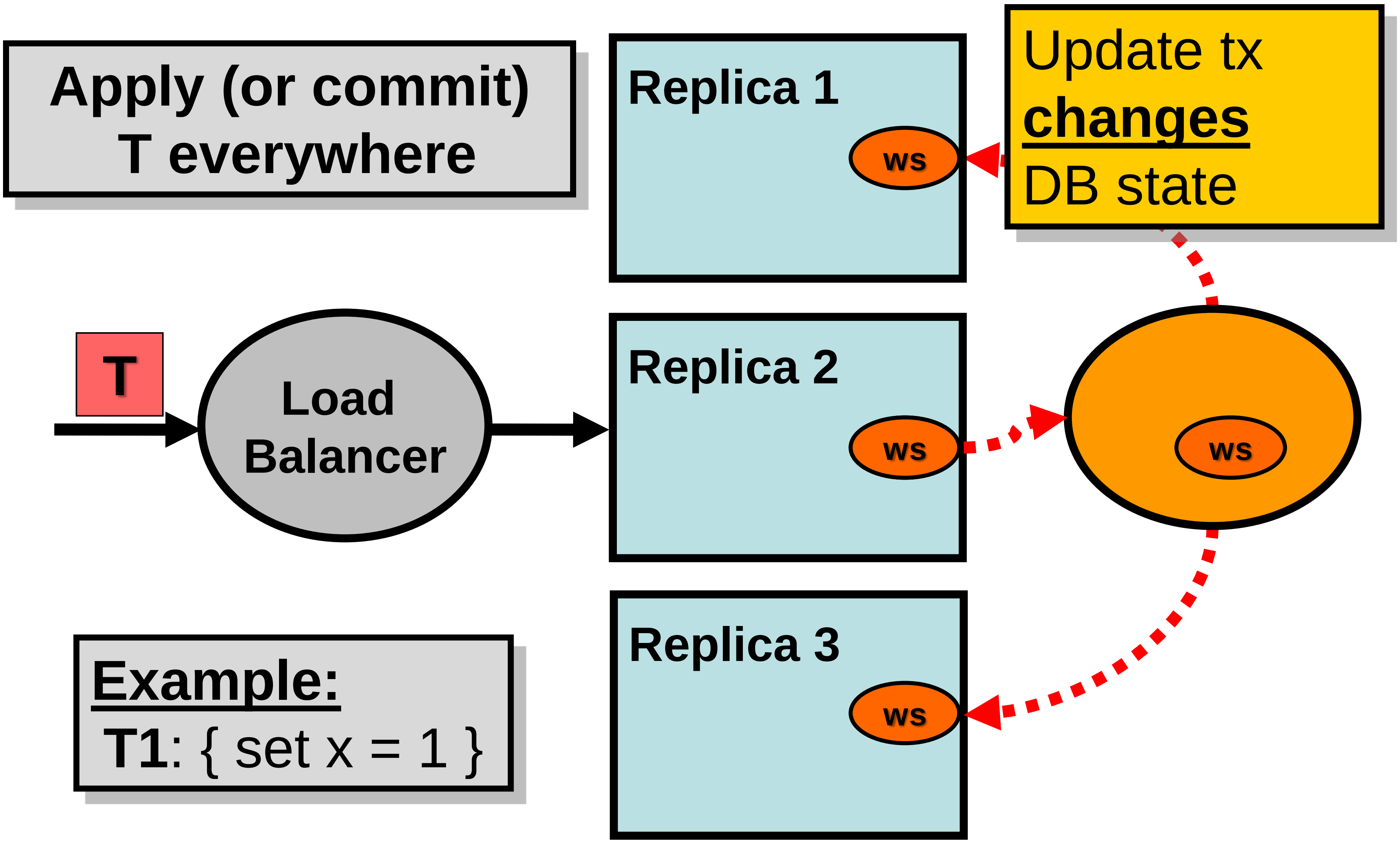
Read tx does **not** change DB state

Update Tx 1/2



Update tx **changes** DB state

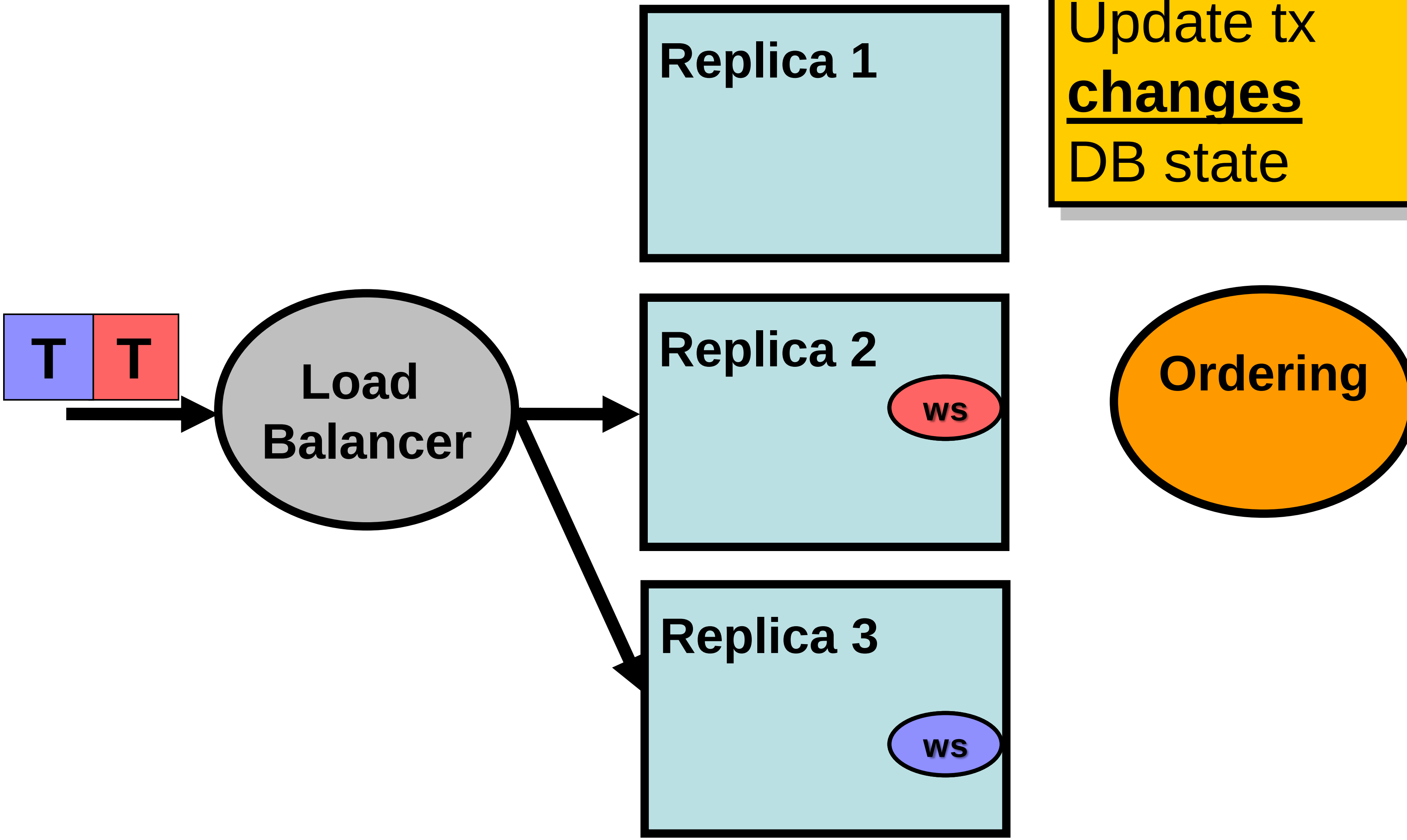
Update Tx 1/2



Apply (or commit)
T everywhere
Replica 1
ws
Update tx
changes
DB state
T
Load
Balancer
Replica 2
ws
ws
Replica 3
ws
Example:
T1: { set x = 1 }

Update Tx 2/2

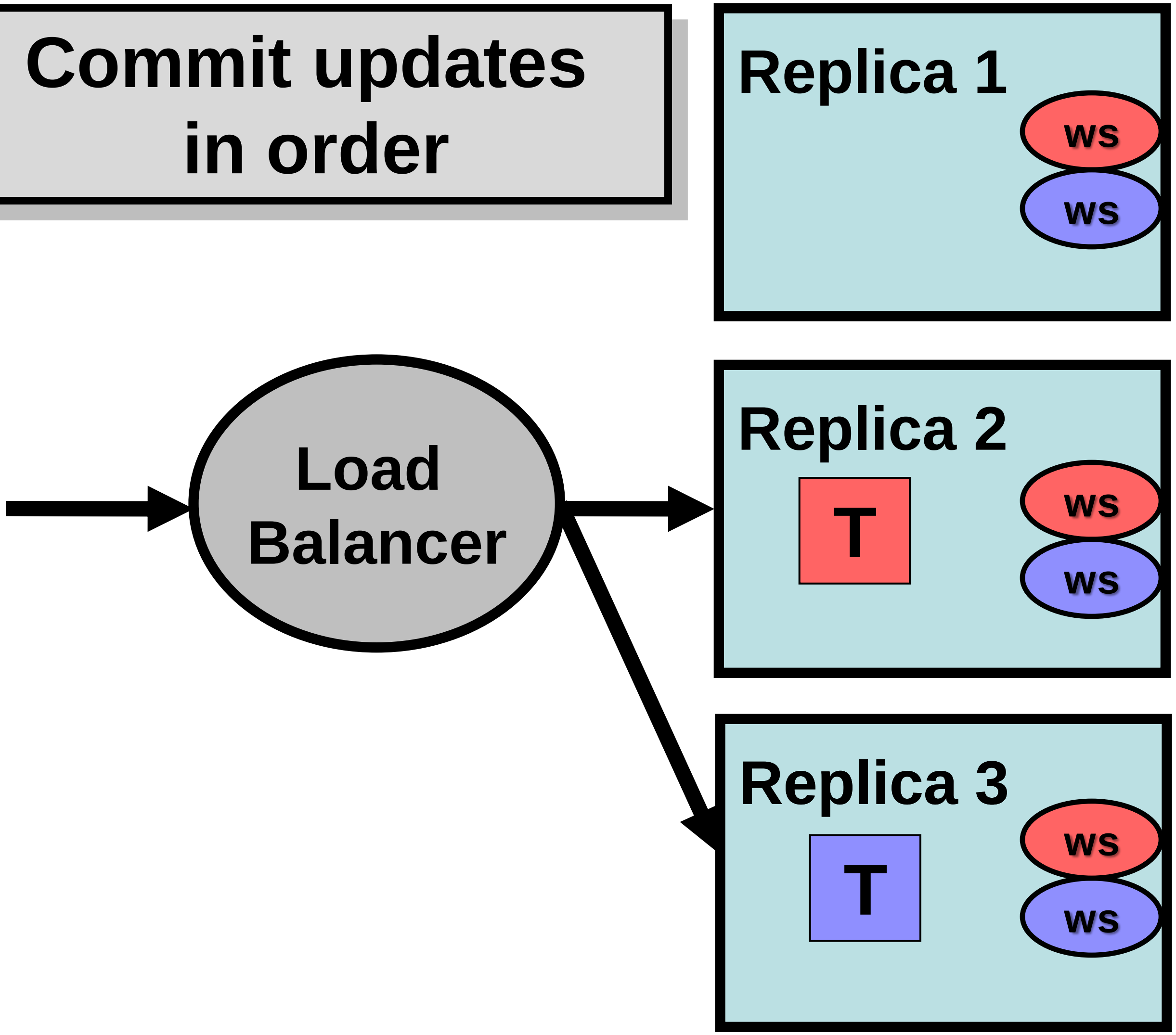
Update tx
changes
DB state

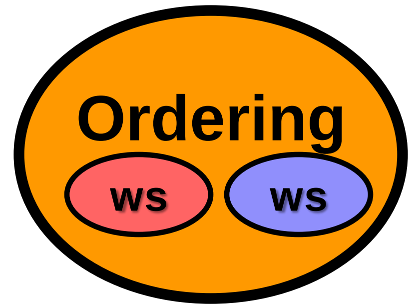


Update Tx 2/2

Commit updates in order

Update tx **changes** DB state



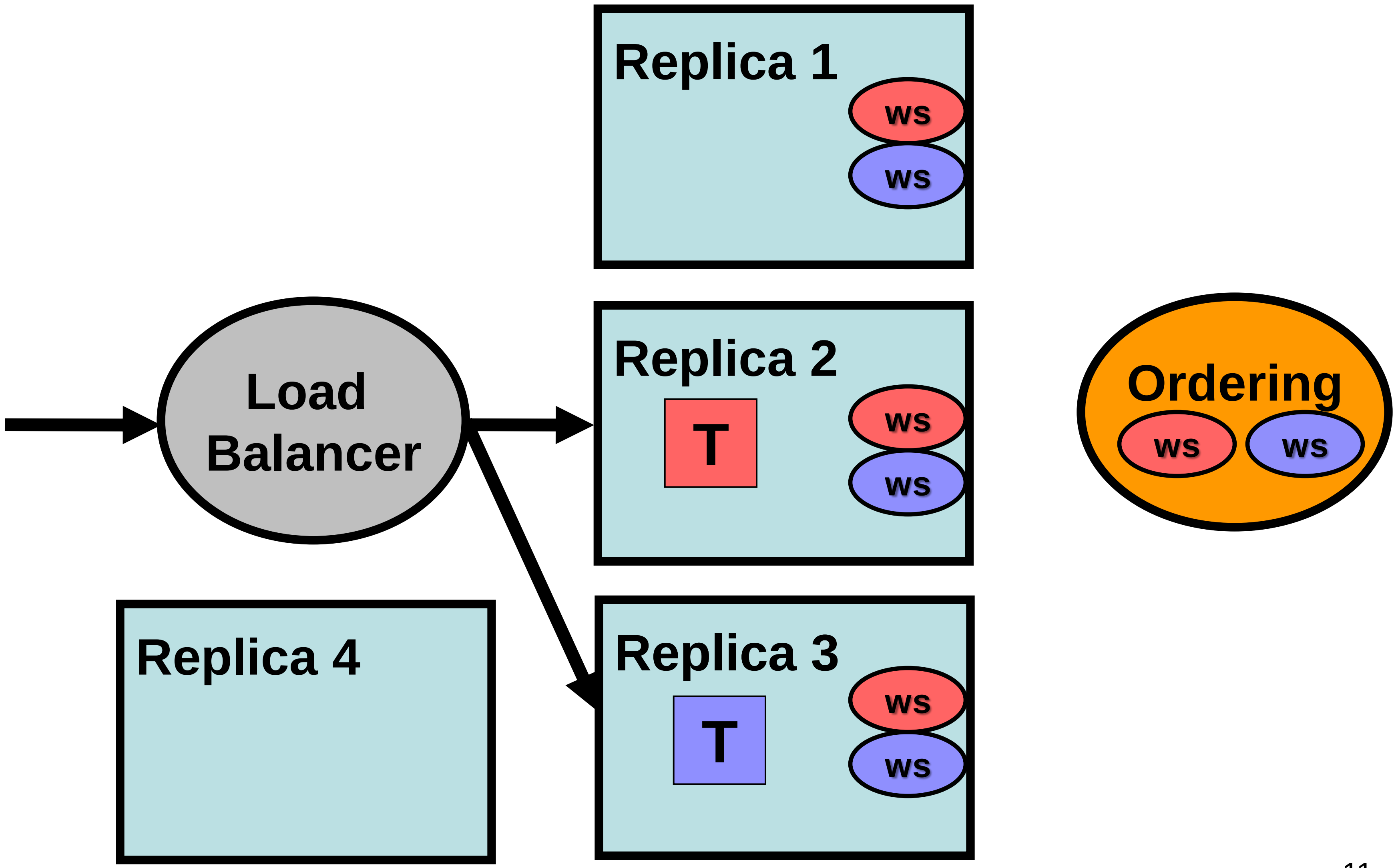


Example:

T1: { set x = 1 }

T2: { set x = 7 }

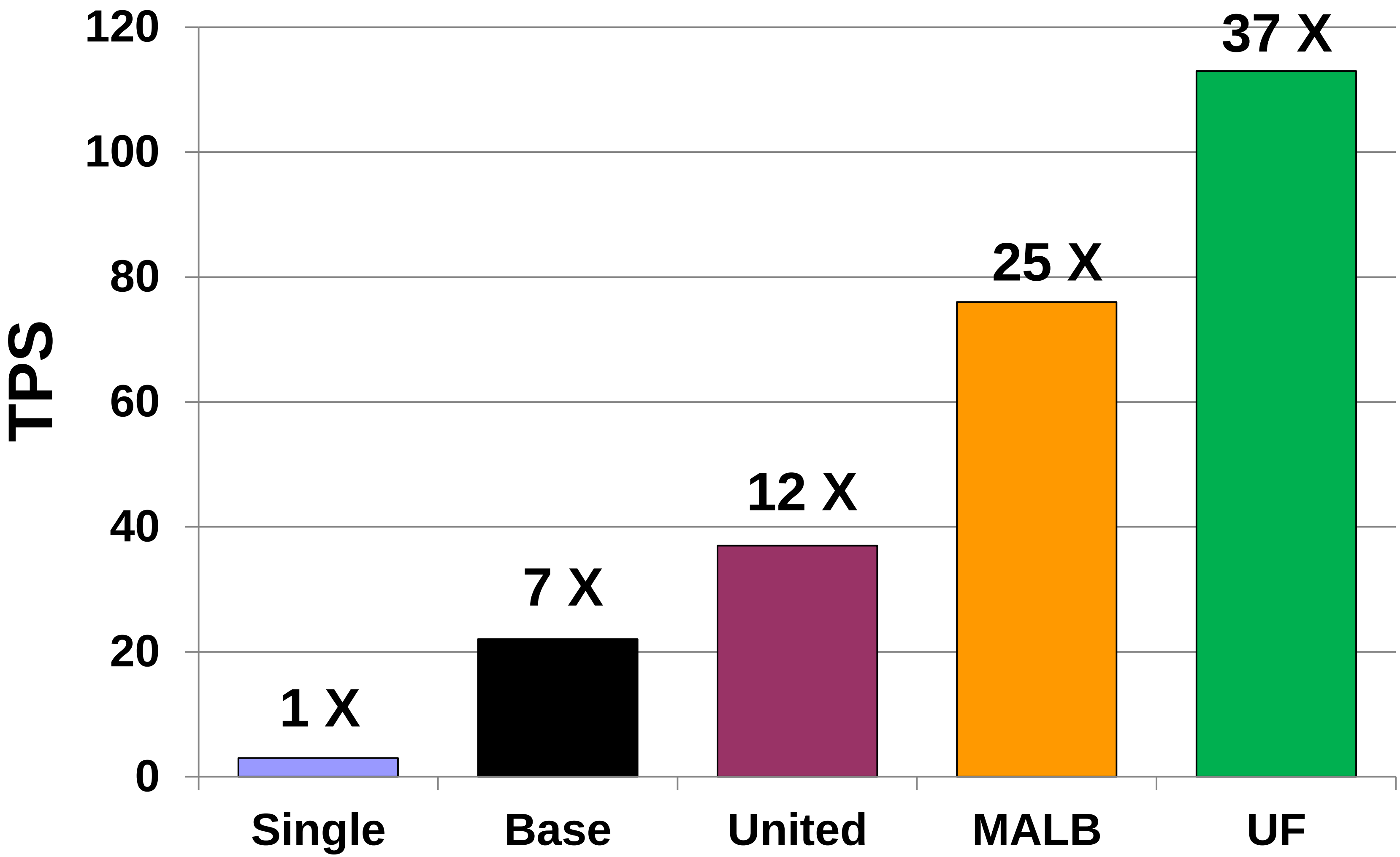
Sub-linear Scalability Wall



This Talk

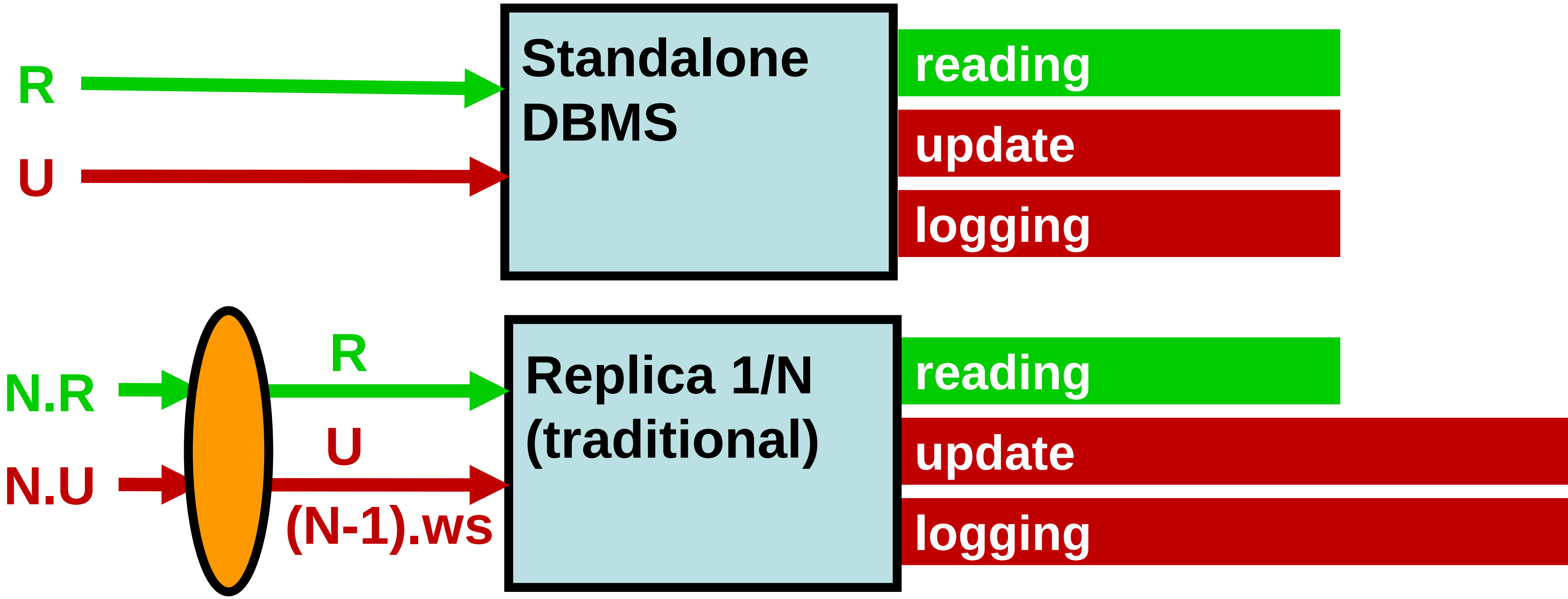
- General scaling techniques
  - Address fundamental bottlenecks
  - Synergistic, implemented in middleware
  - Evaluated experimentally

Super-linear Scalability

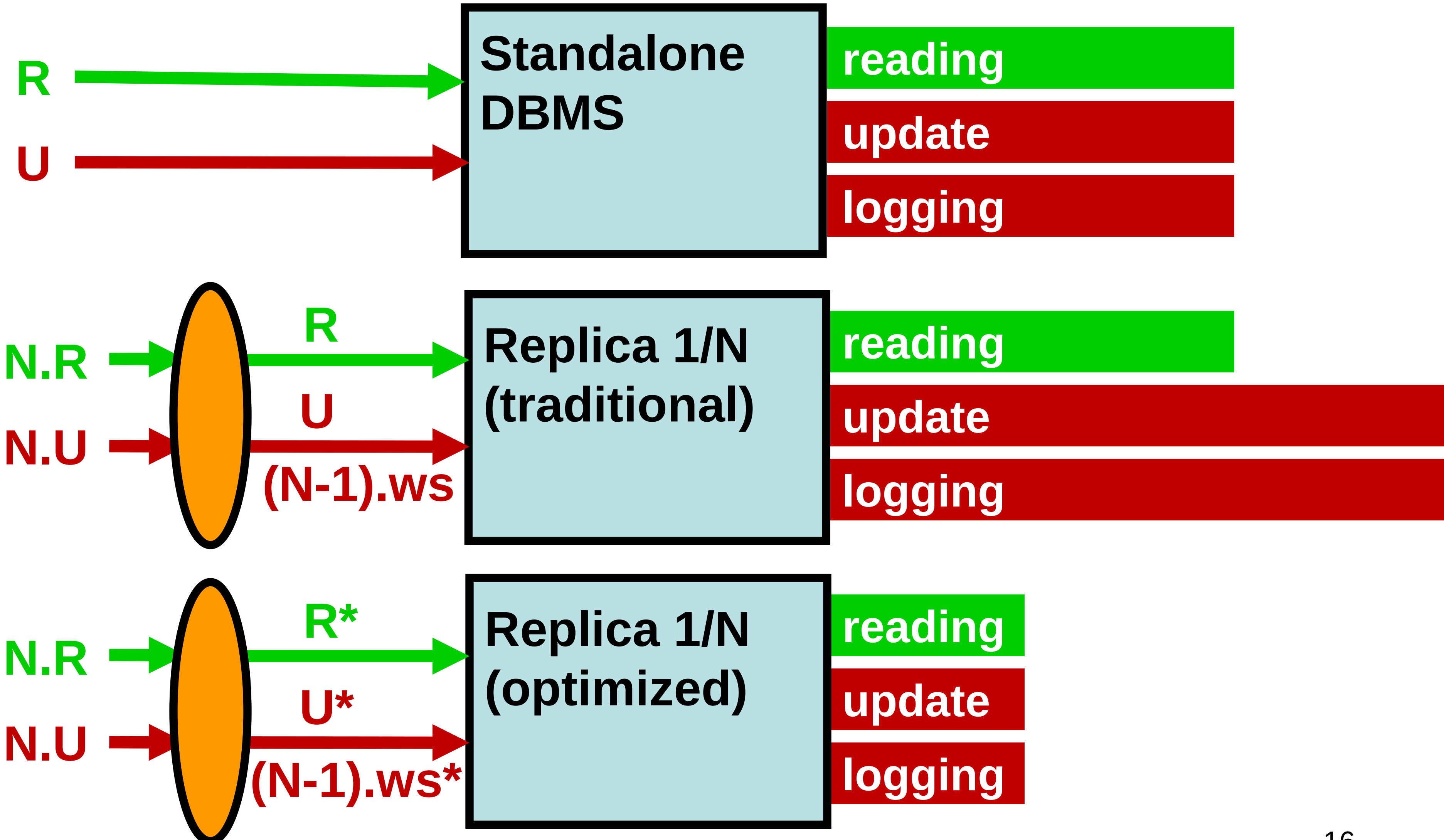


Big Picture: Let’s Oversimplify

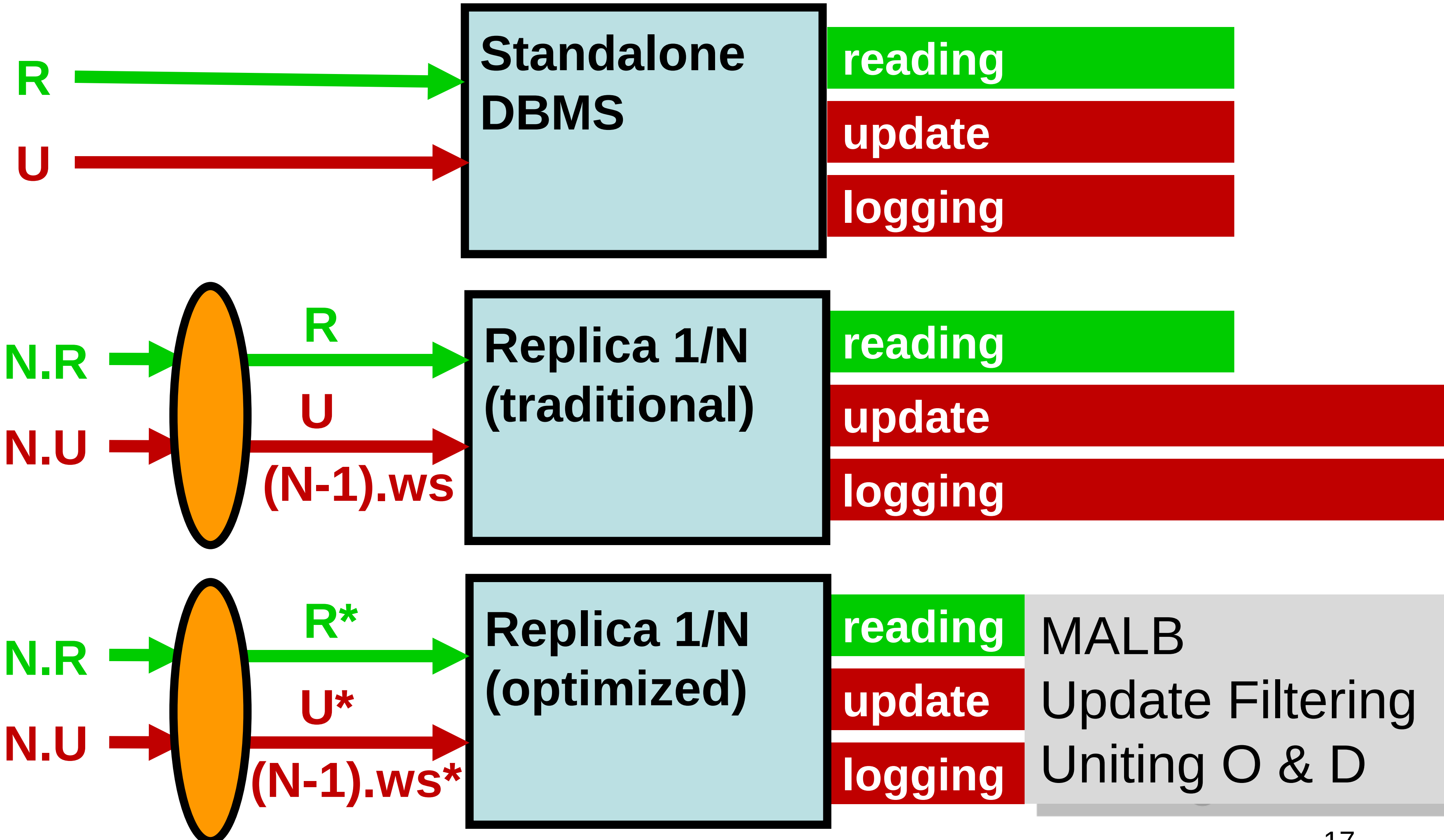
Big Picture: Let's Oversimplify



Big Picture: Let's Oversimplify



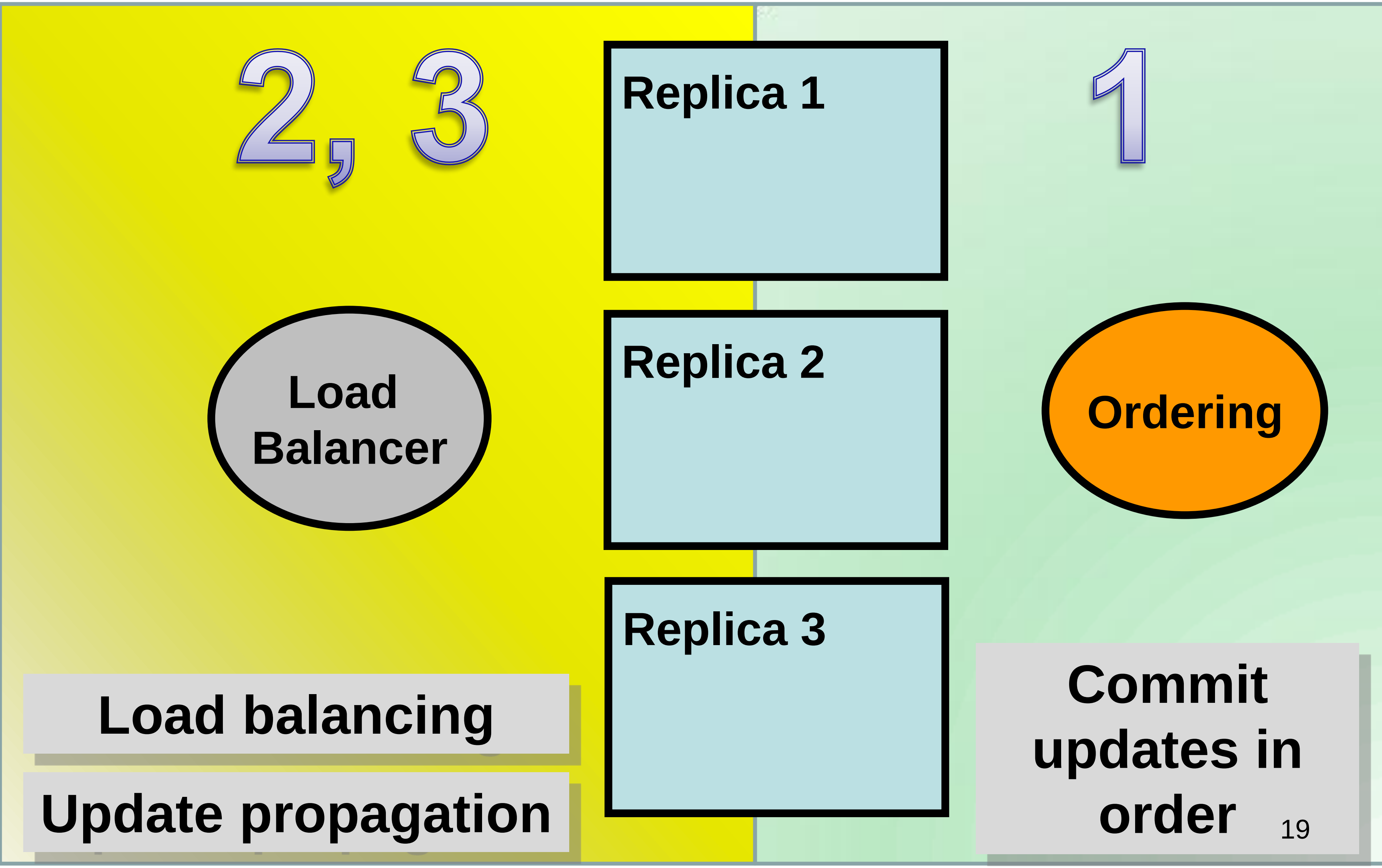
Big Picture: Let's Oversimplify



Key Points

1. **Commit updates in order**
   - Perform serial synchronous disk writes
   - Unite ordering and durability
2. **Load balancing**
   - Optimize for equal load: memory contention
   - MALB: optimize for in-memory execution
3. **Update propagation**
   - Propagate updates everywhere
   - Update filtering: propagate to where needed

Roadmap



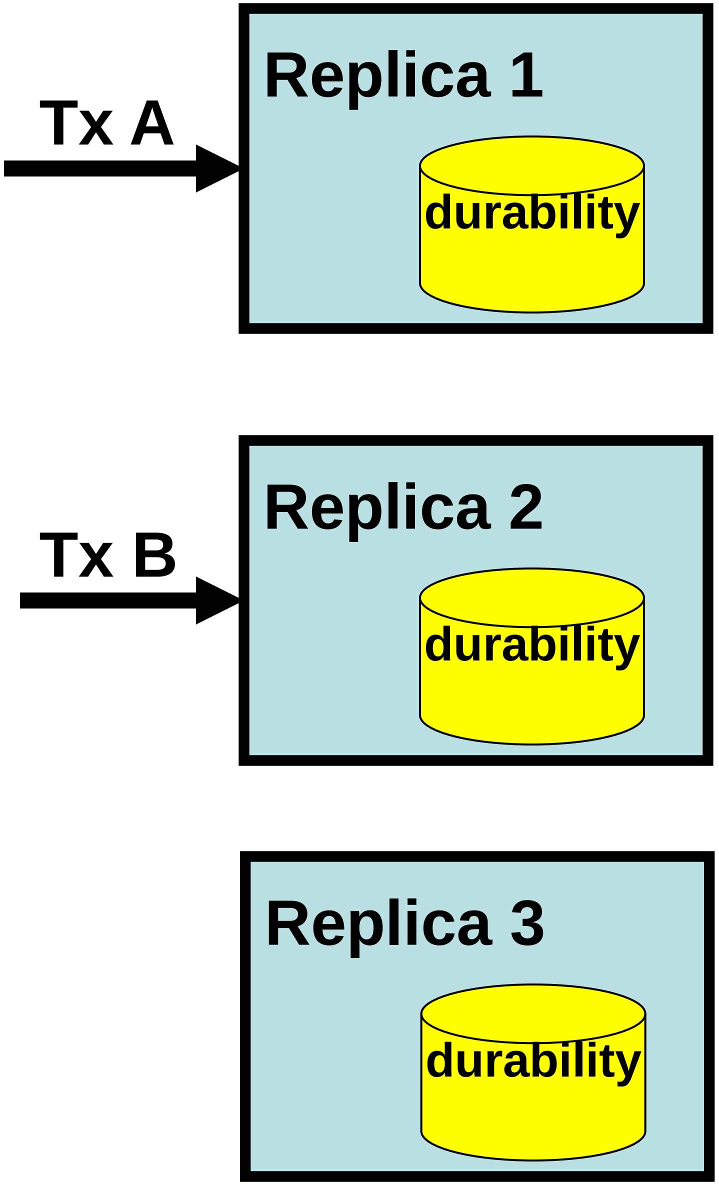
2, 3
1
Replica 1
Replica 2
Replica 3
Load Balancer
Ordering
Load balancing
Update propagation
Commit updates in order

Key Idea

- **Traditionally:**
  - Commit ordering and durability are separated

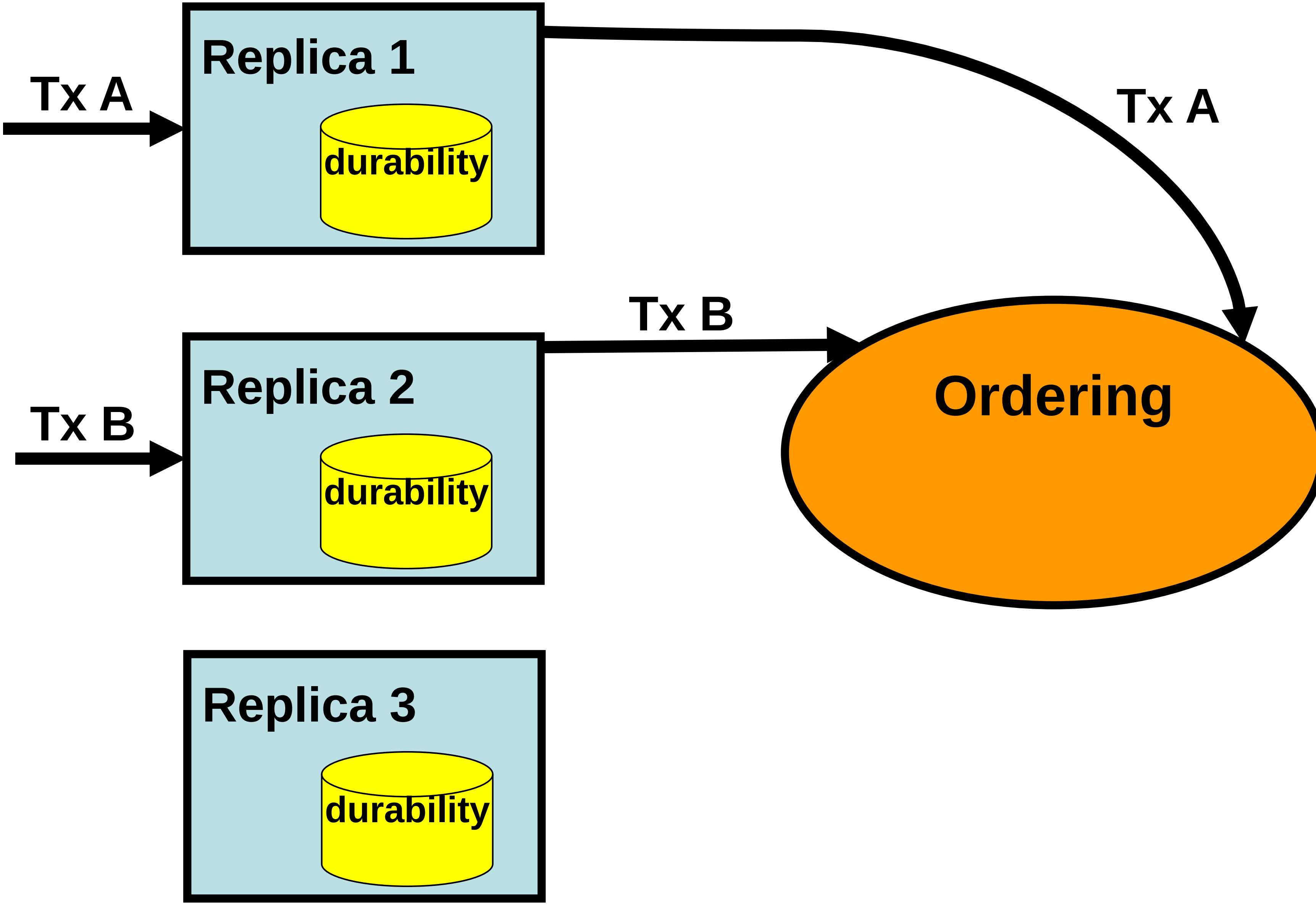
- **Key idea:**
  - Unite commit ordering and durability

All Replicas Must Agree

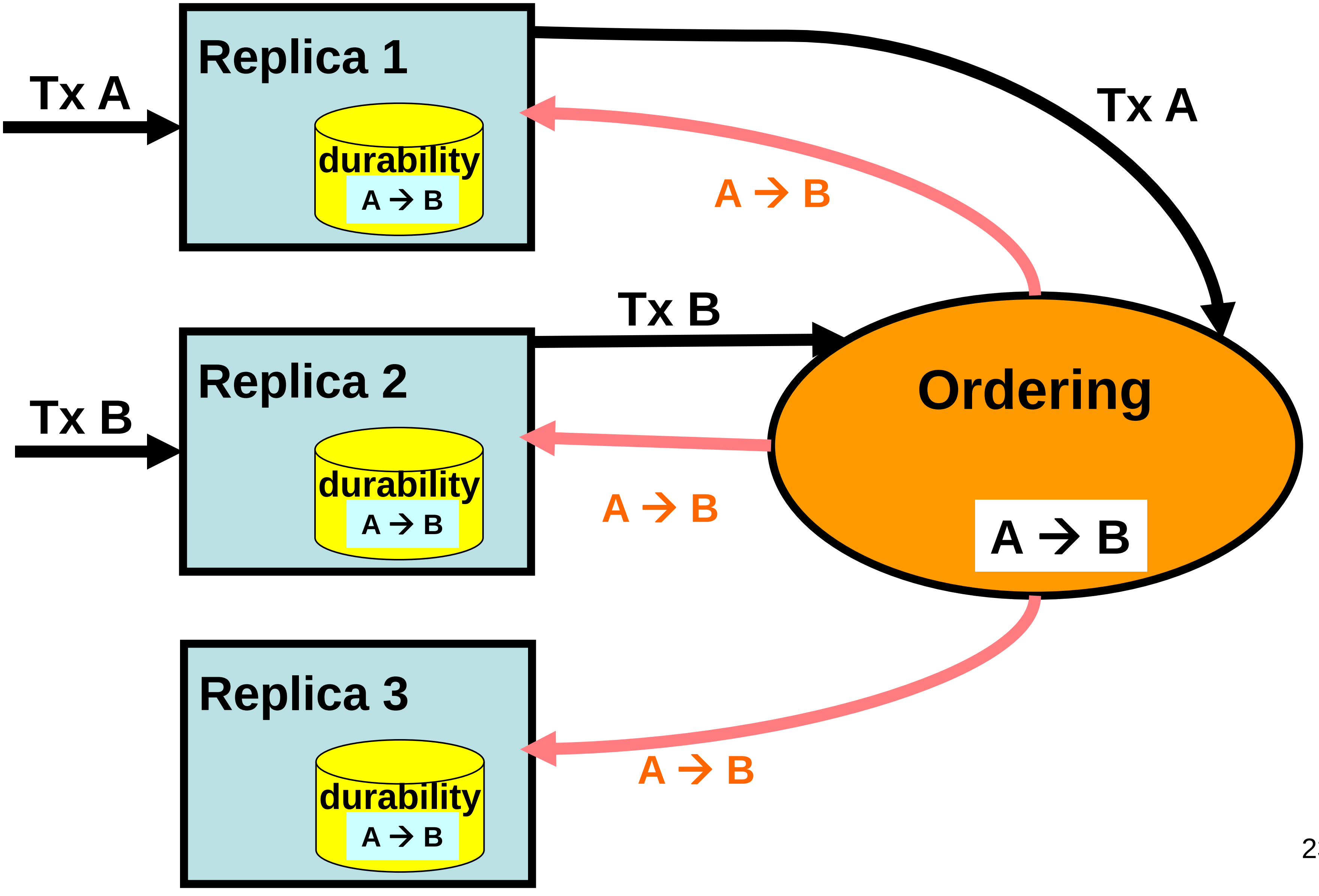


- All replicas agree on
  - which update tx commit
  - their commit order
- Total order
  - Determined by middleware
  - Followed by each replica

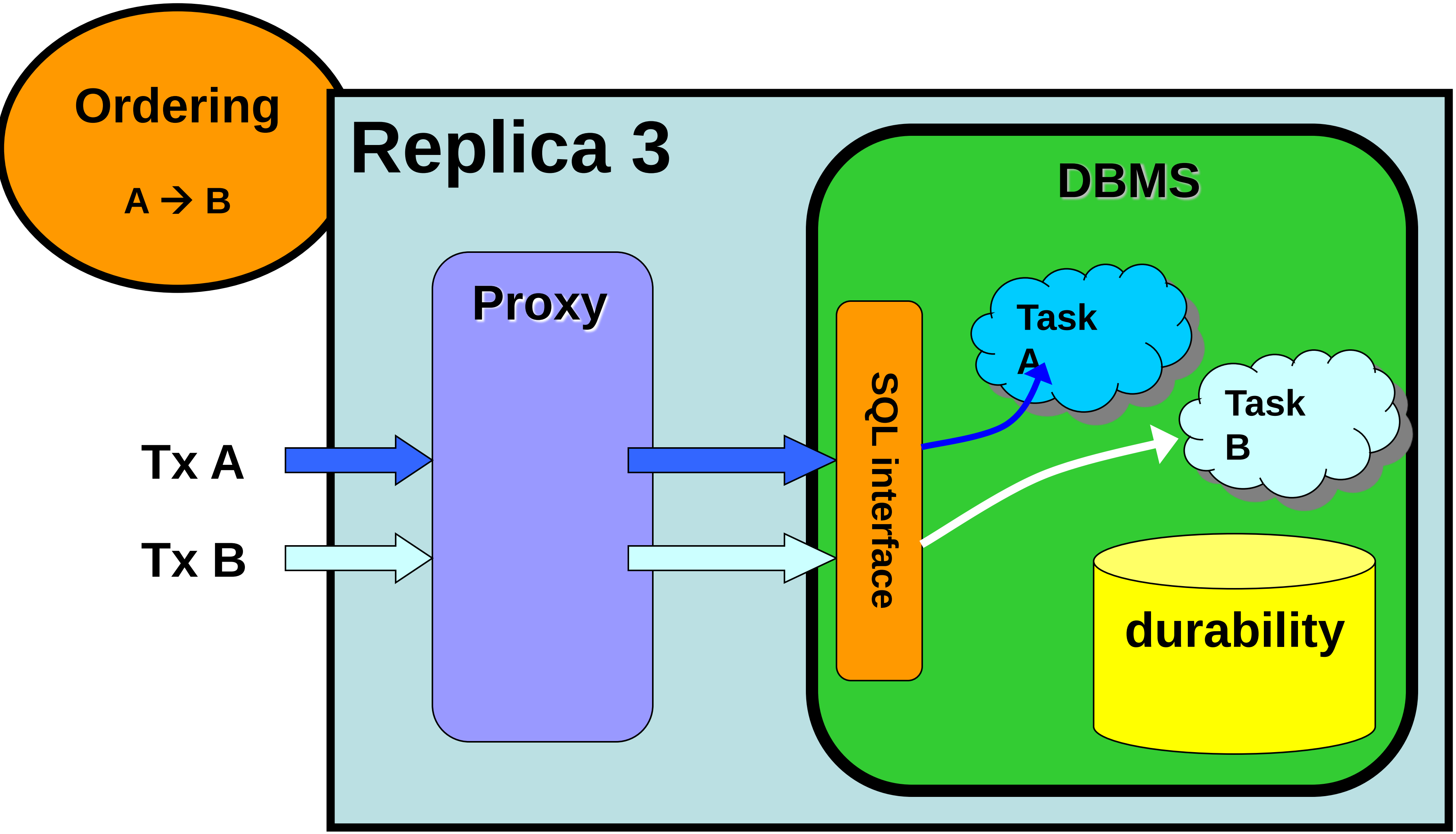
Order Outside DBMS



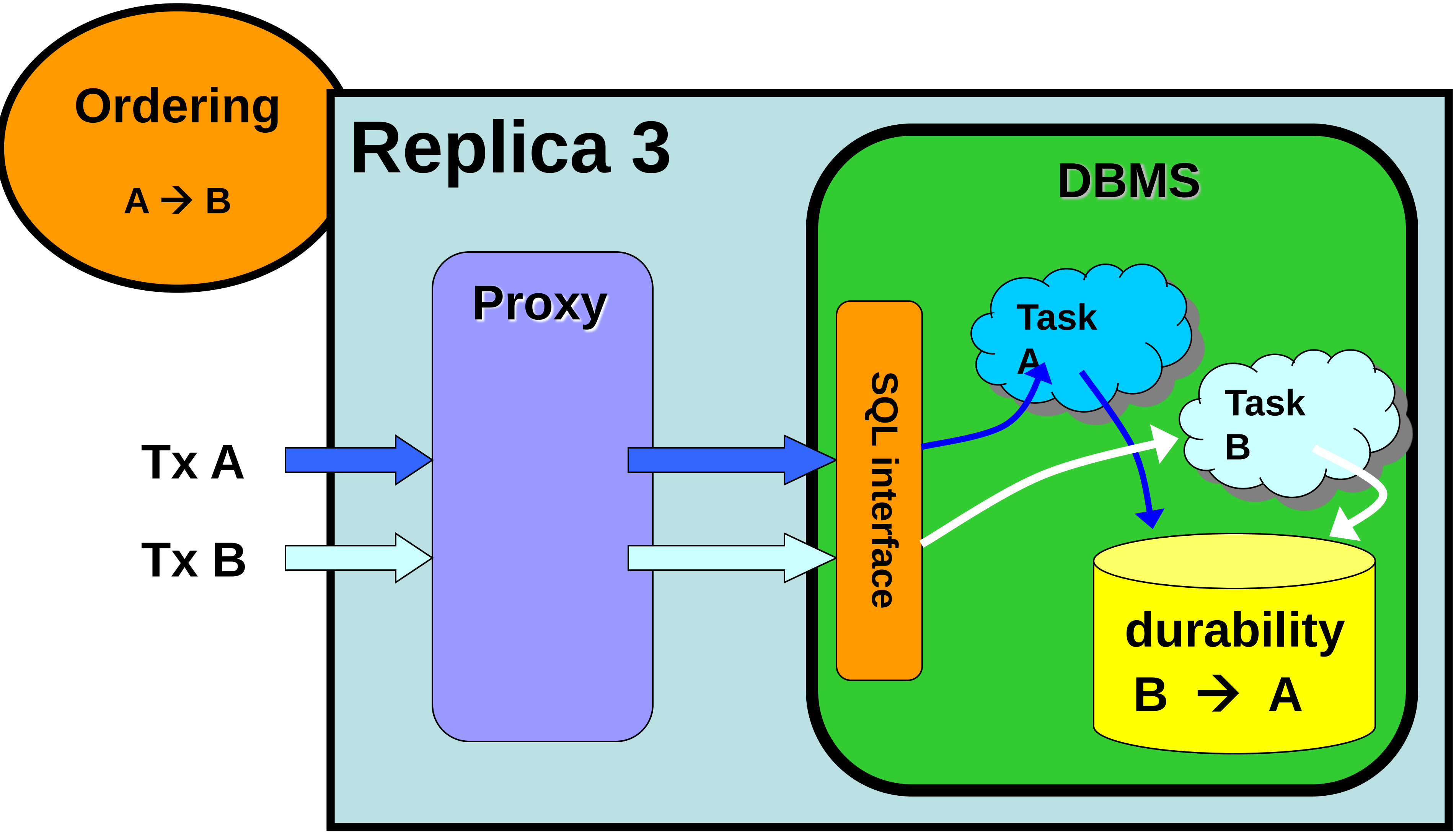
Order Outside DBMS



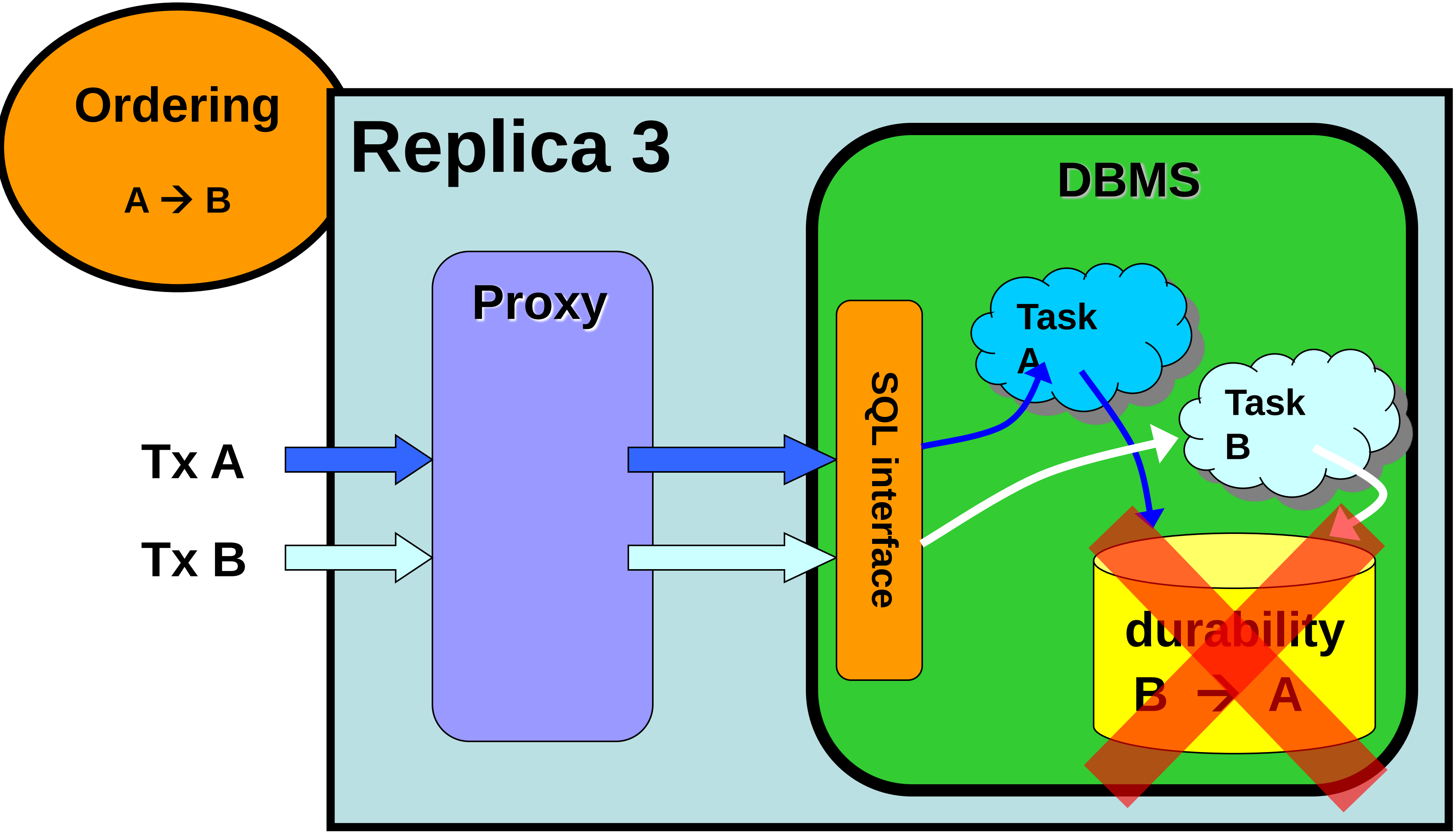
Enforce External Commit Order



Enforce External Commit Order

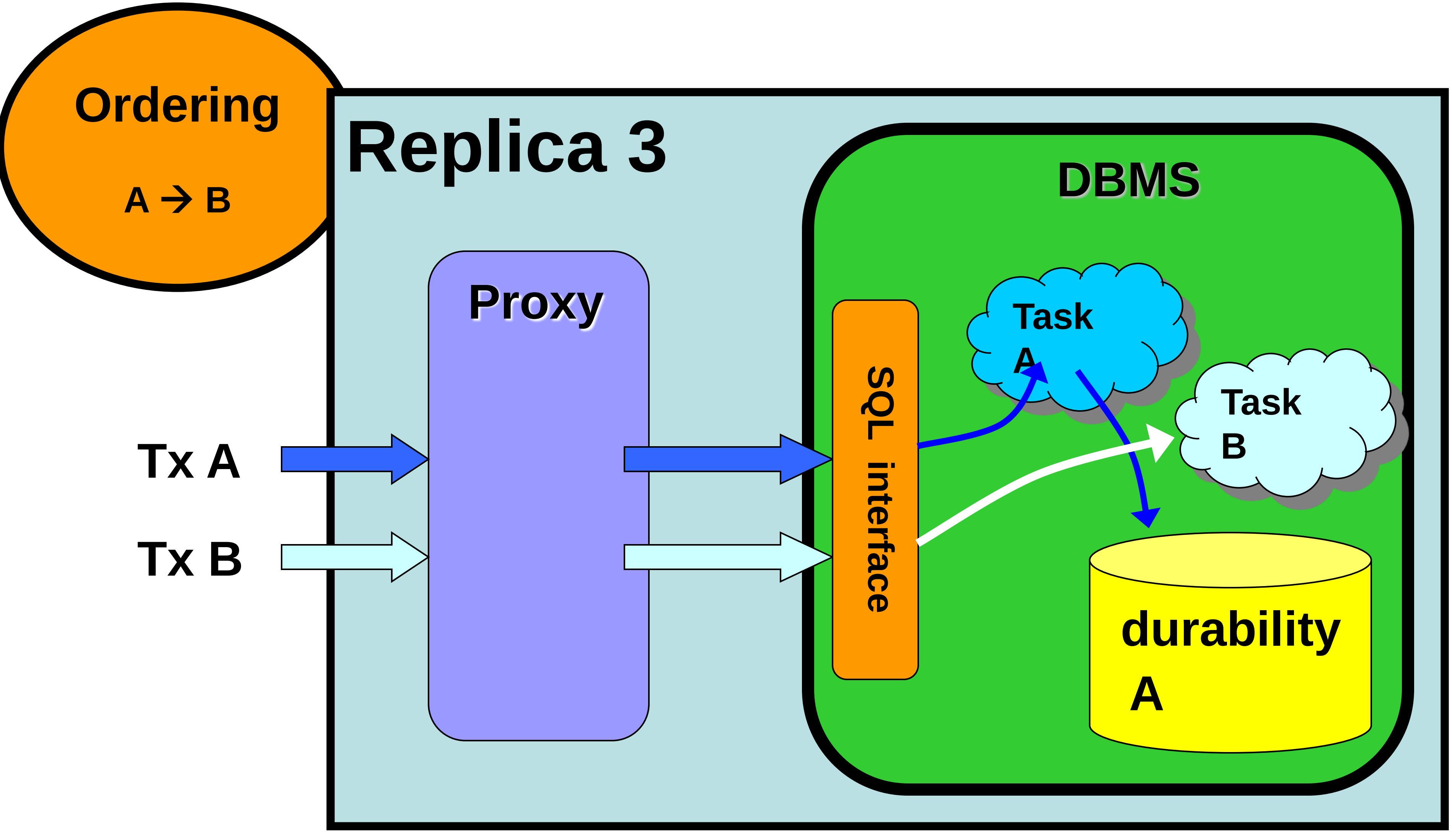


Enforce External Commit Order

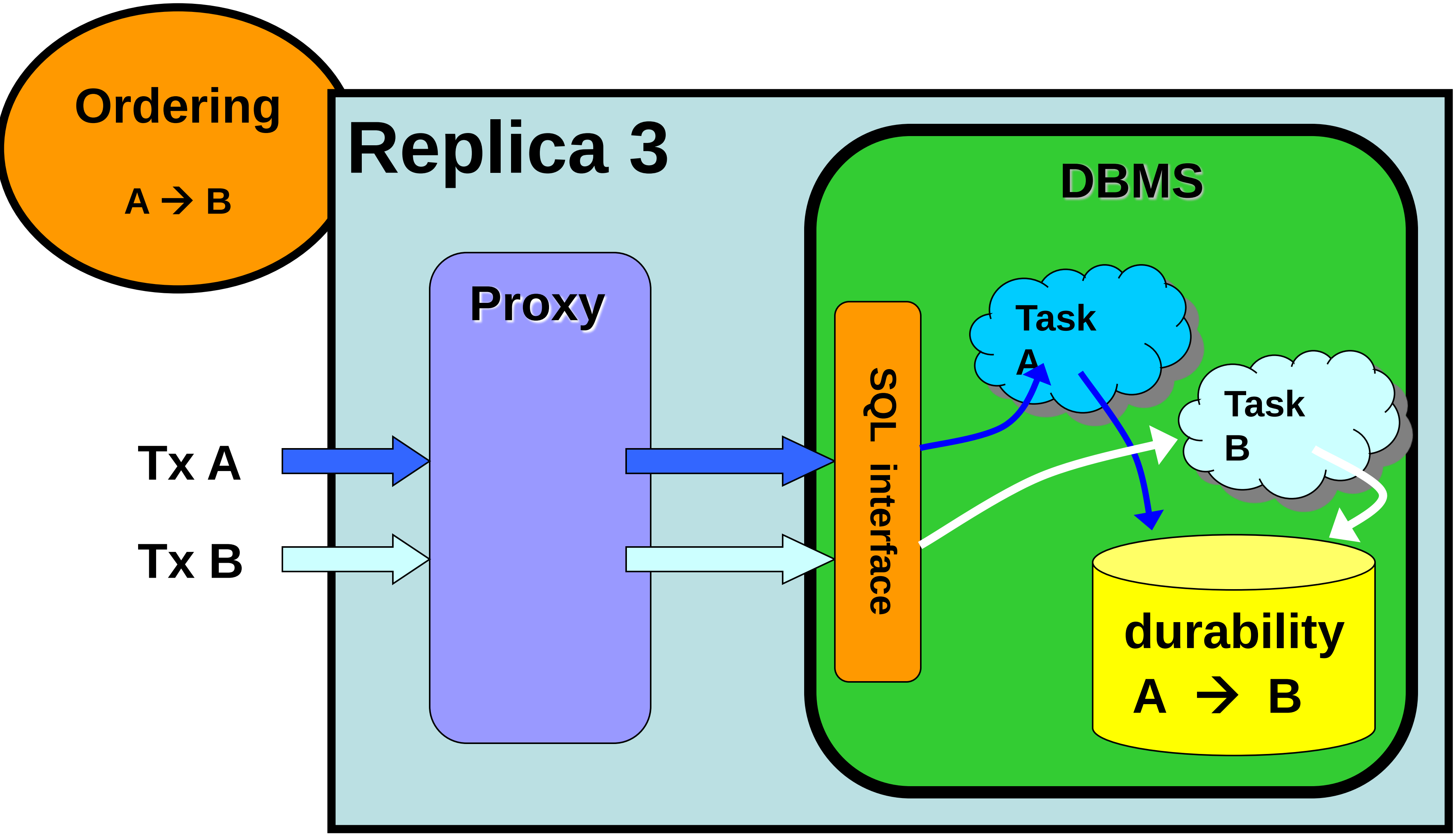


Cannot commit A & B concurrently!

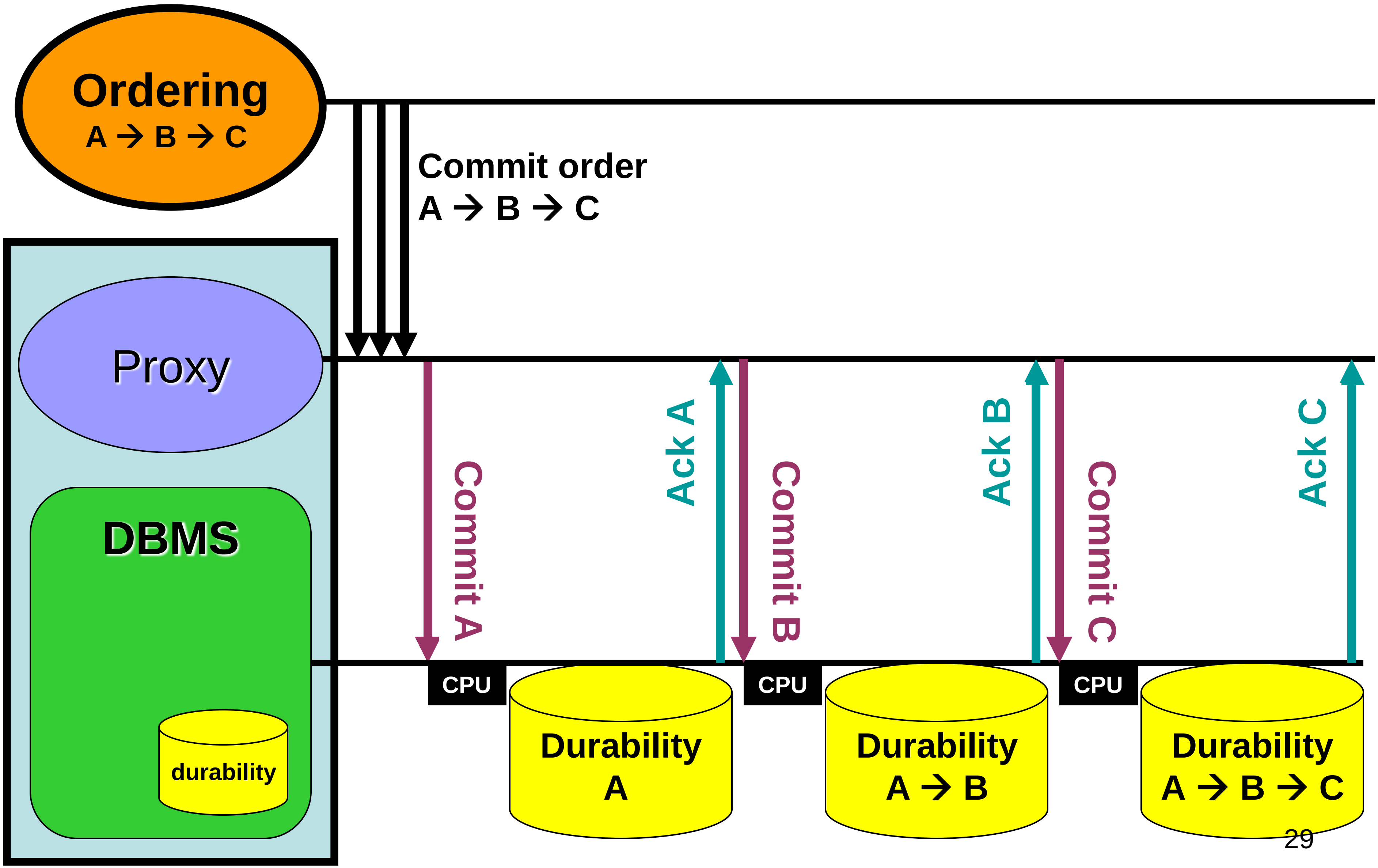
Enforce Order = Serial Commit



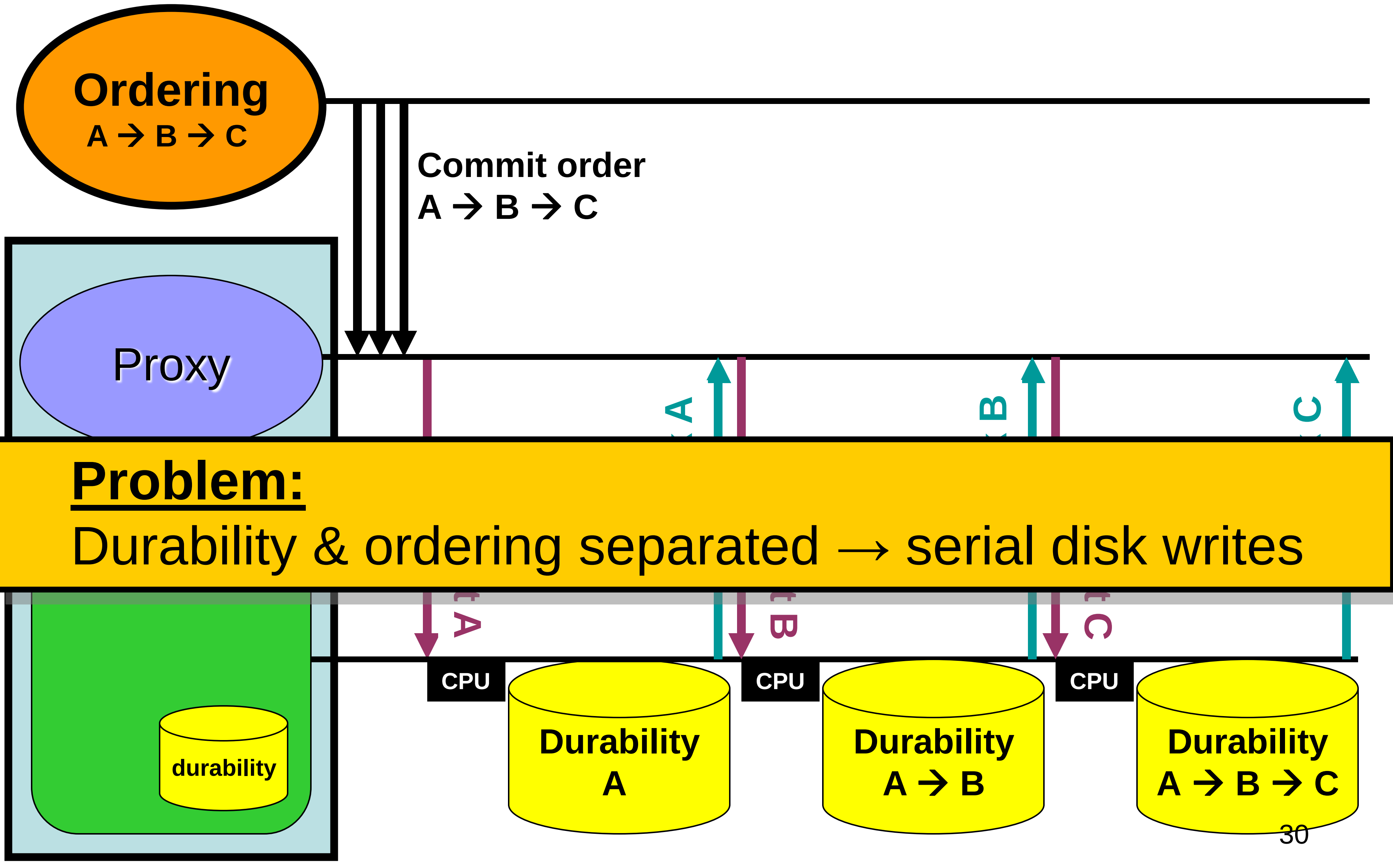
Enforce Order = Serial Commit



Commit Serialization is Slow



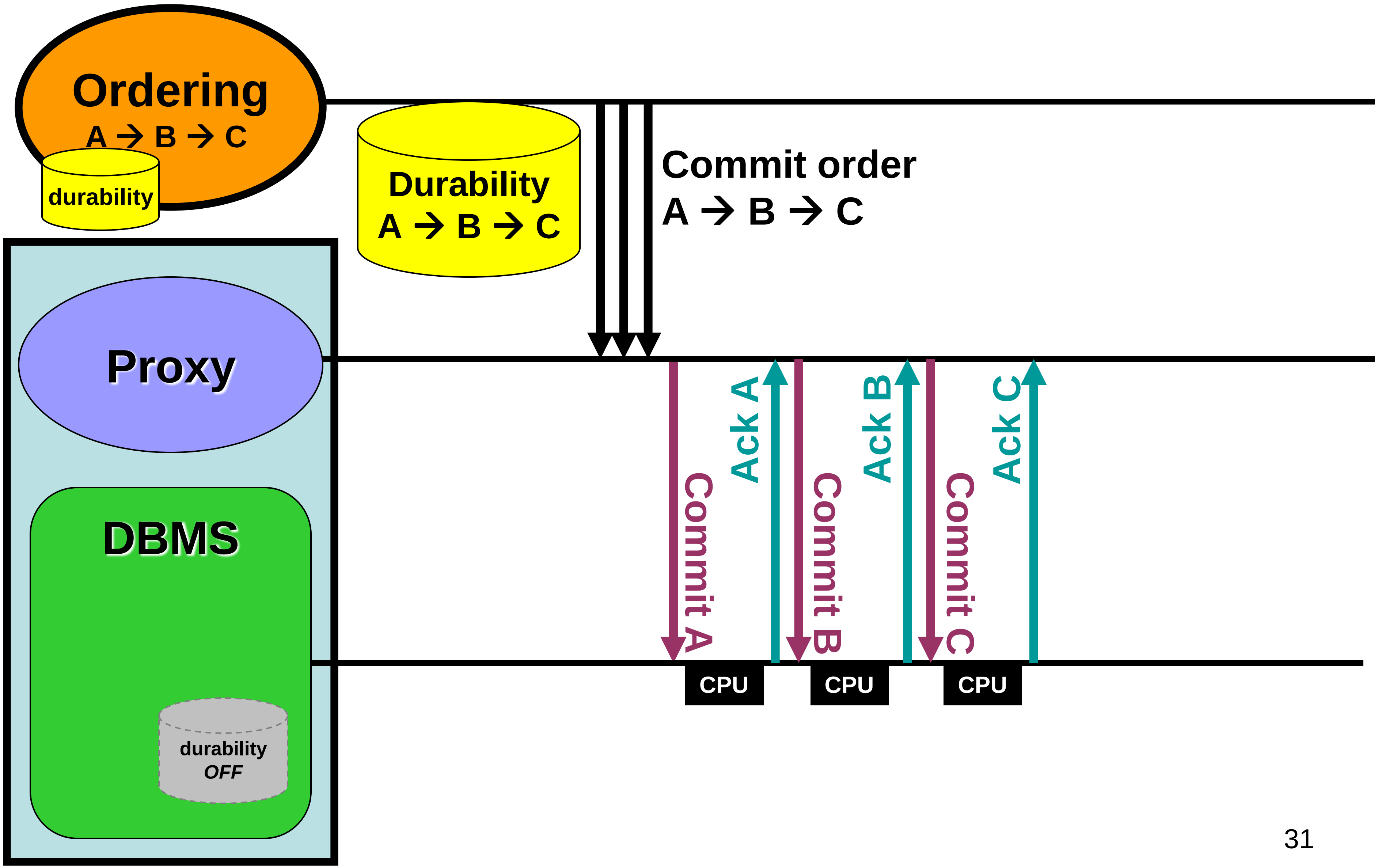
Commit Serialization is Slow



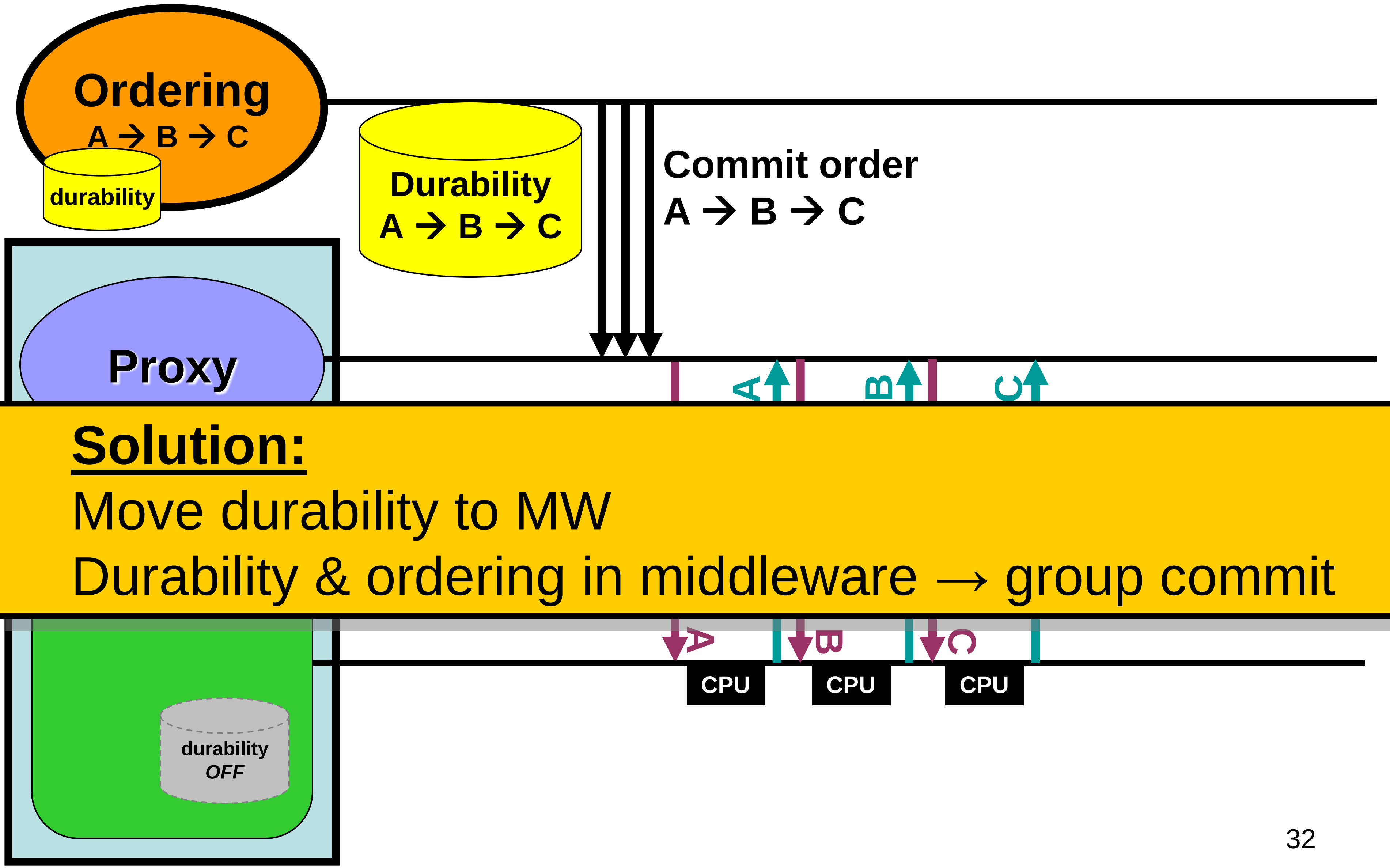
Problem:

Durability & ordering separated → serial disk writes

Unite D. & O. in Middleware



Unite D. & O. in Middleware



Solution:

Move durability to MW

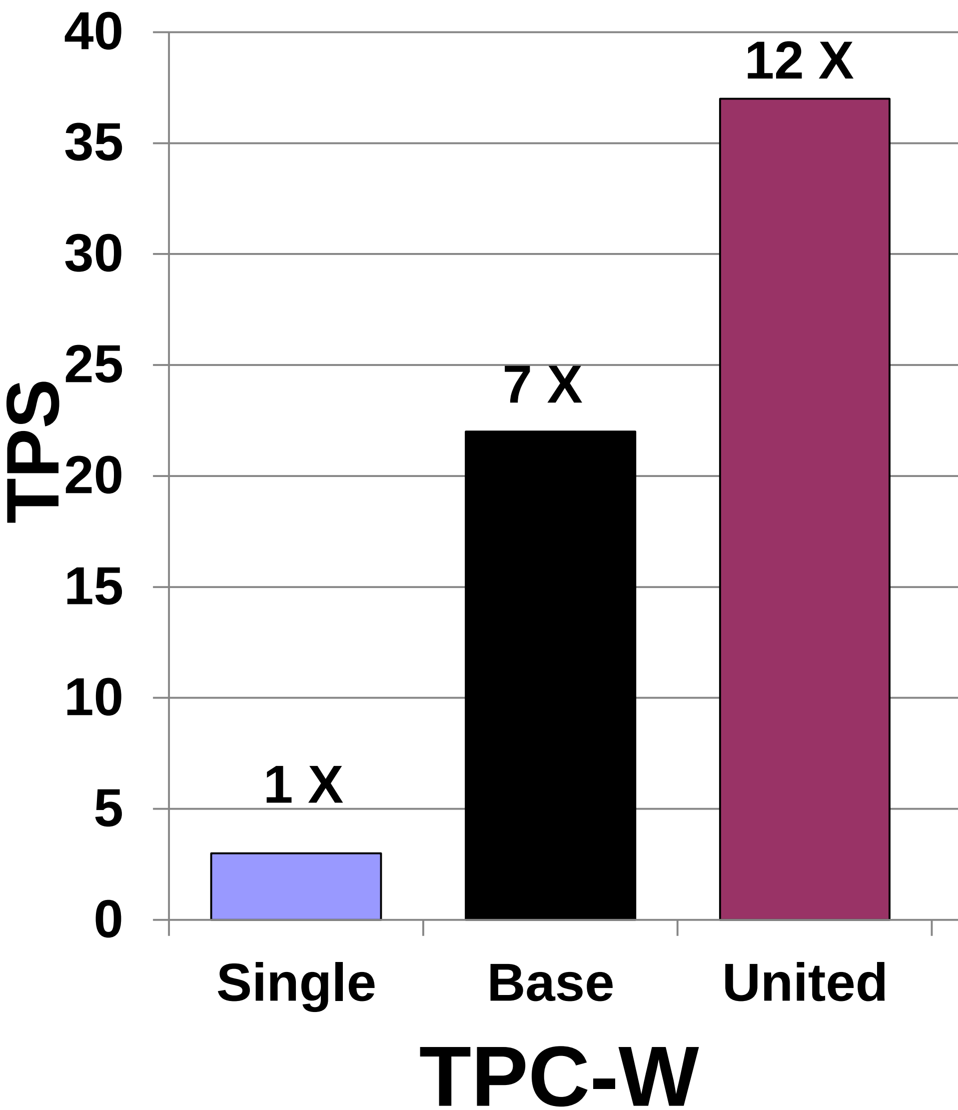
Durability & ordering in middleware → group commit

Implementation: Uniting D & O in MW

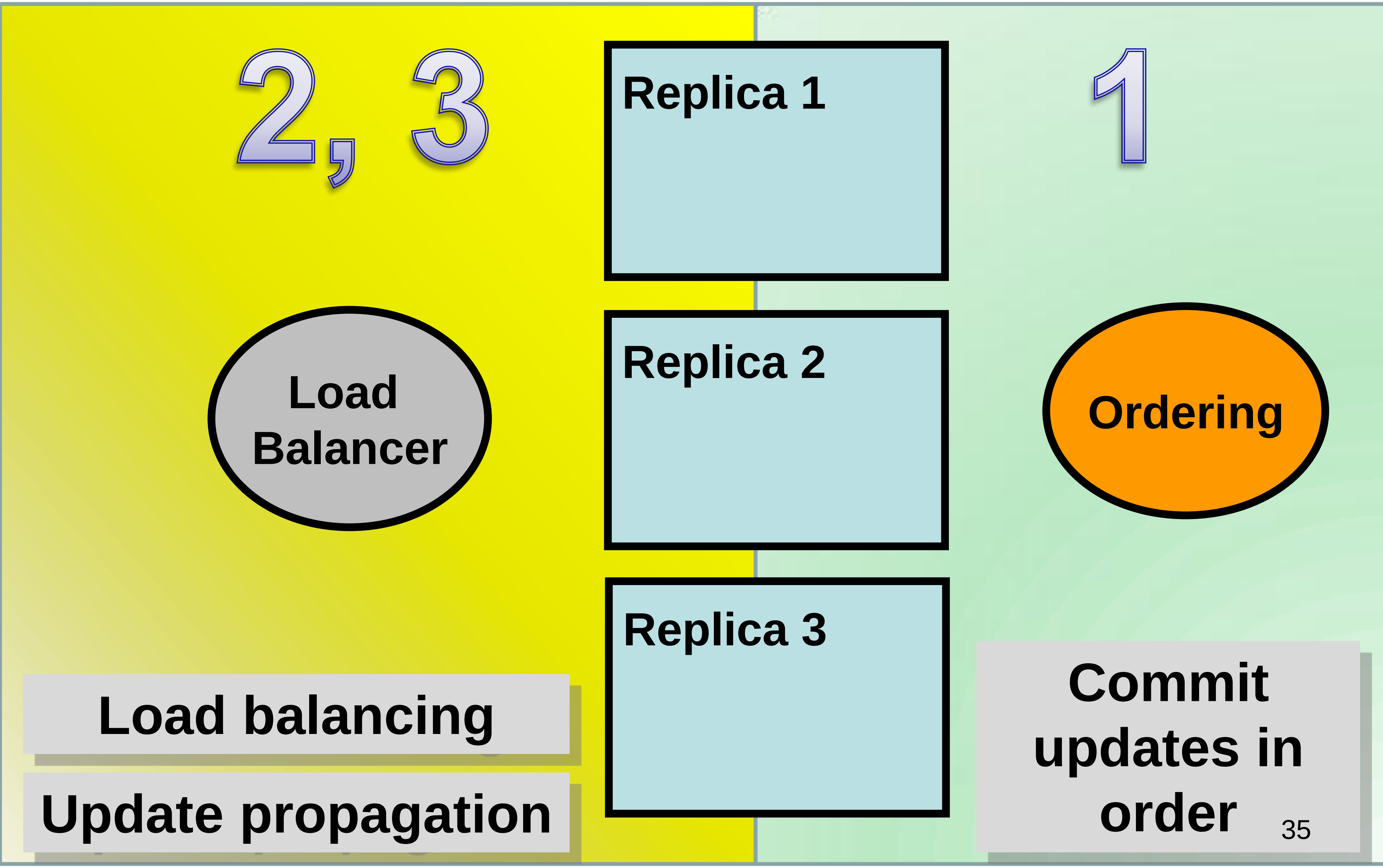
- Middleware logs tx effects
  - Durability of update tx
    - Guaranteed in middleware
    - Turn durability off at database
- Middleware performs durability & ordering
  - United → group commit → fast
- Database commits update tx serially
  - Commit = quick main memory operation

Uniting Improves Throughput

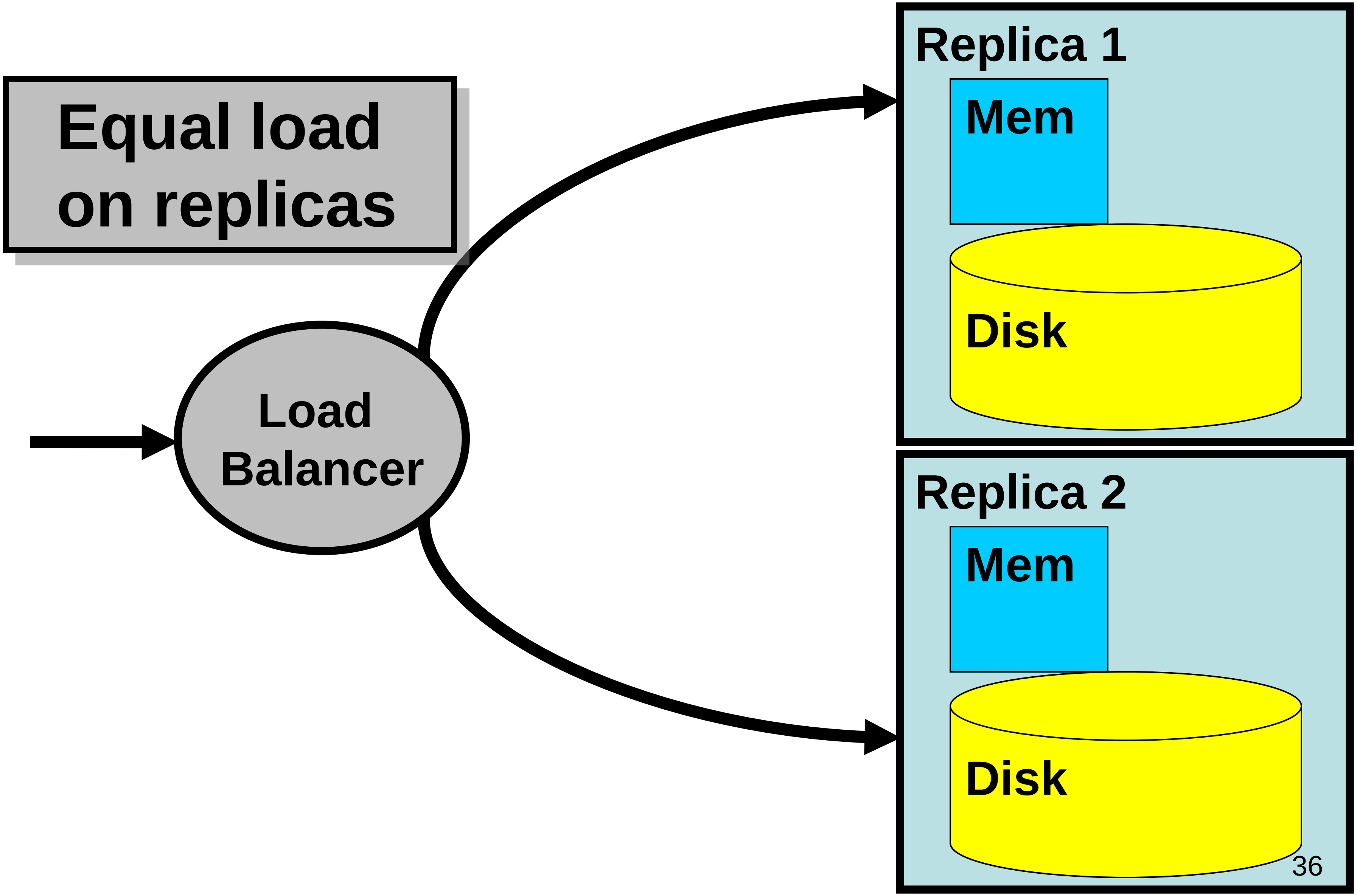
- **Metric**
  - Throughput
- **Workload**
  - TPC-W Ordering (50% updates)
- **System**
  - Linux cluster
  - PostgreSQL
  - 16 replicas
  - Serializable exec.



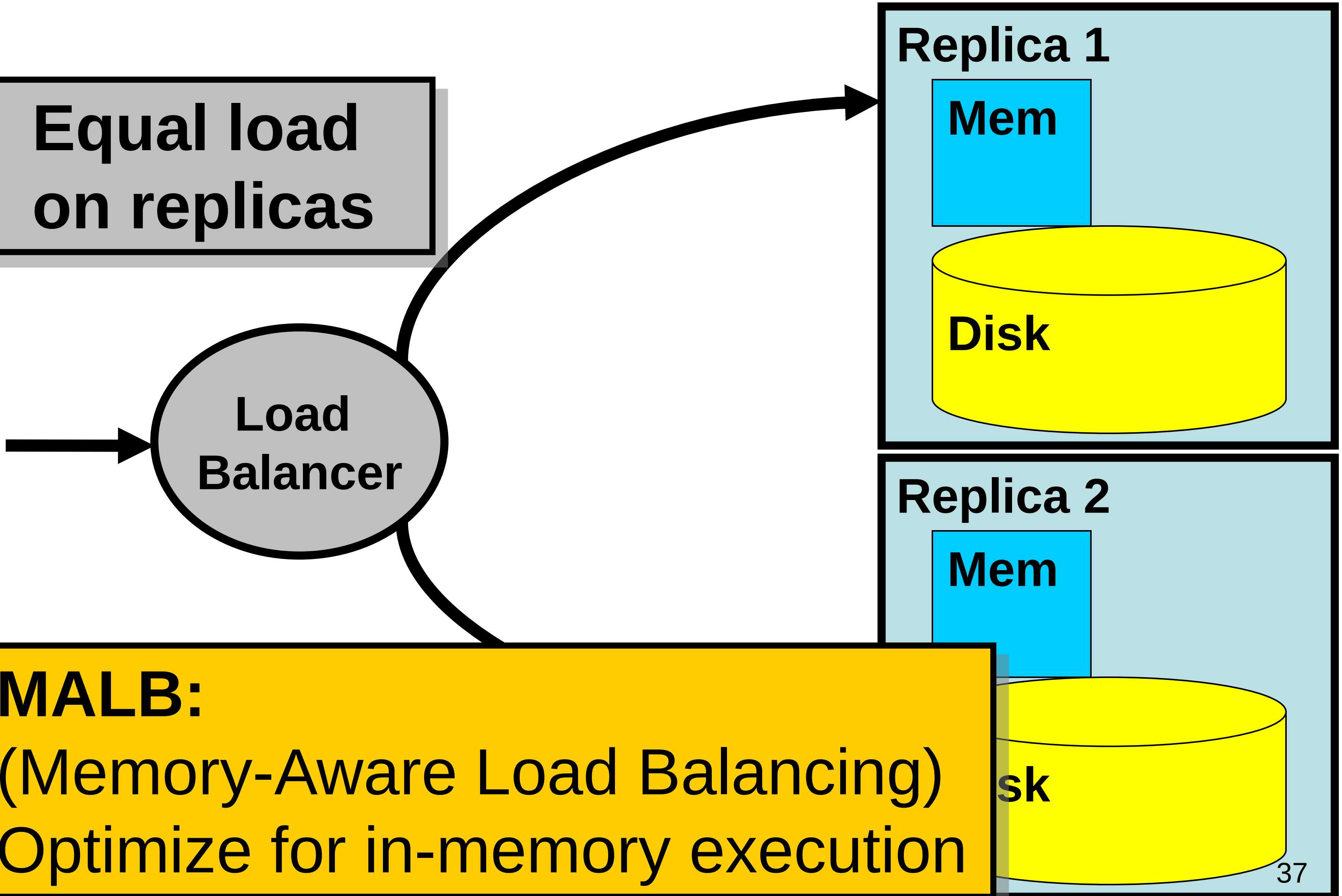
Roadmap


2, 3
Replica 1
1
Load Balancer
Replica 2
Ordering
Replica 3
Load balancing
Update propagation
Commit updates in order

Key Idea


Equal load
on replicas
Load
Balancer
Replica 1
Mem
Disk
Replica 2
Mem
Disk

Key Idea



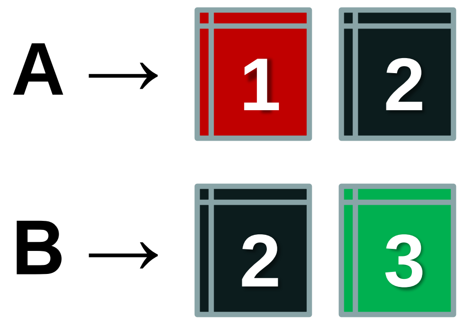
Equal load on replicas

MALB:
(Memory-Aware Load Balancing)
Optimize for in-memory execution

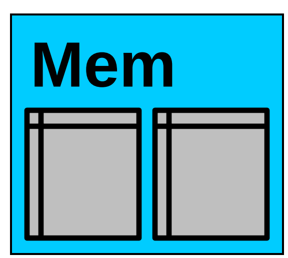
How Does MALB Work?

Database

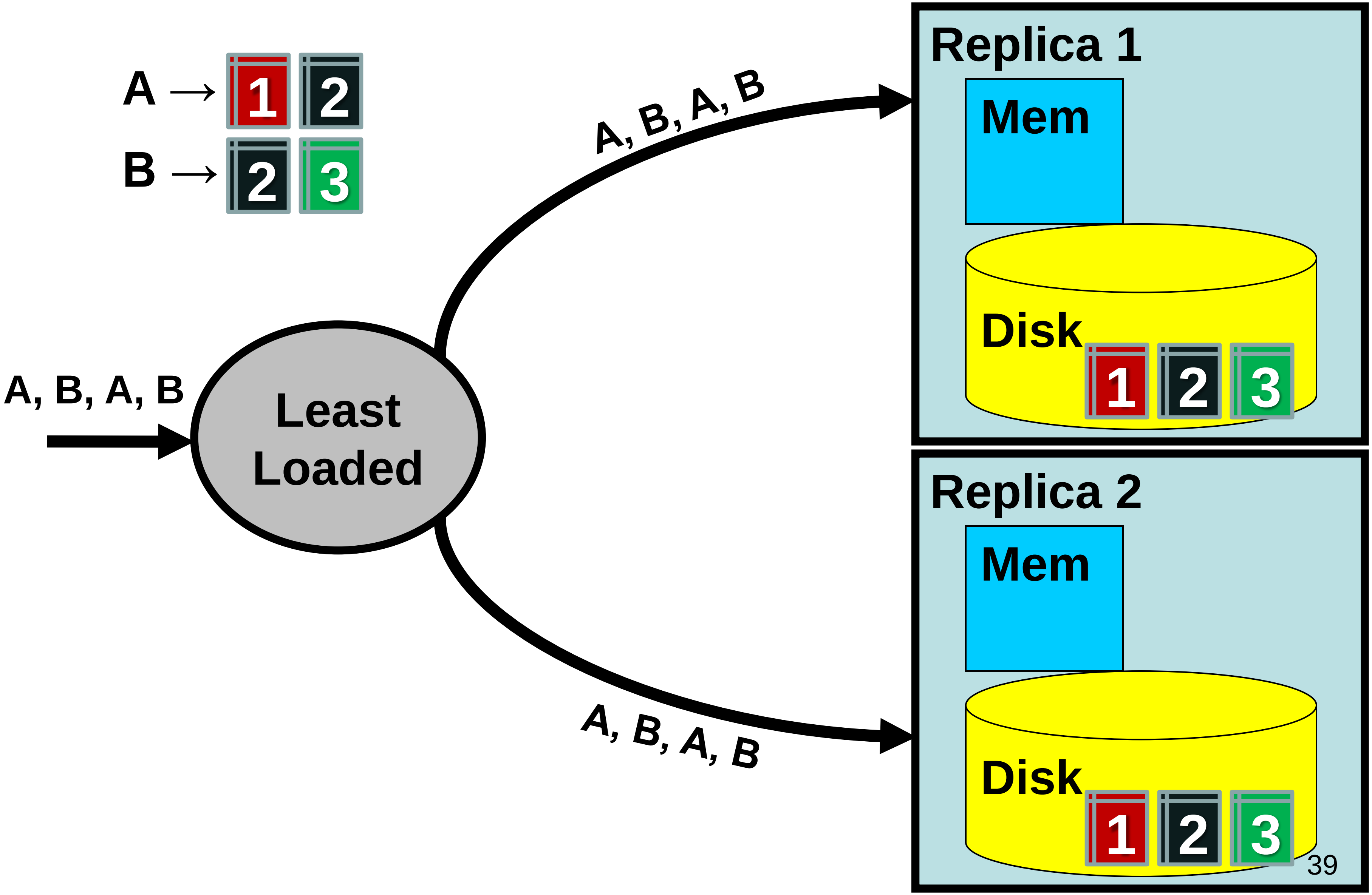
Workload



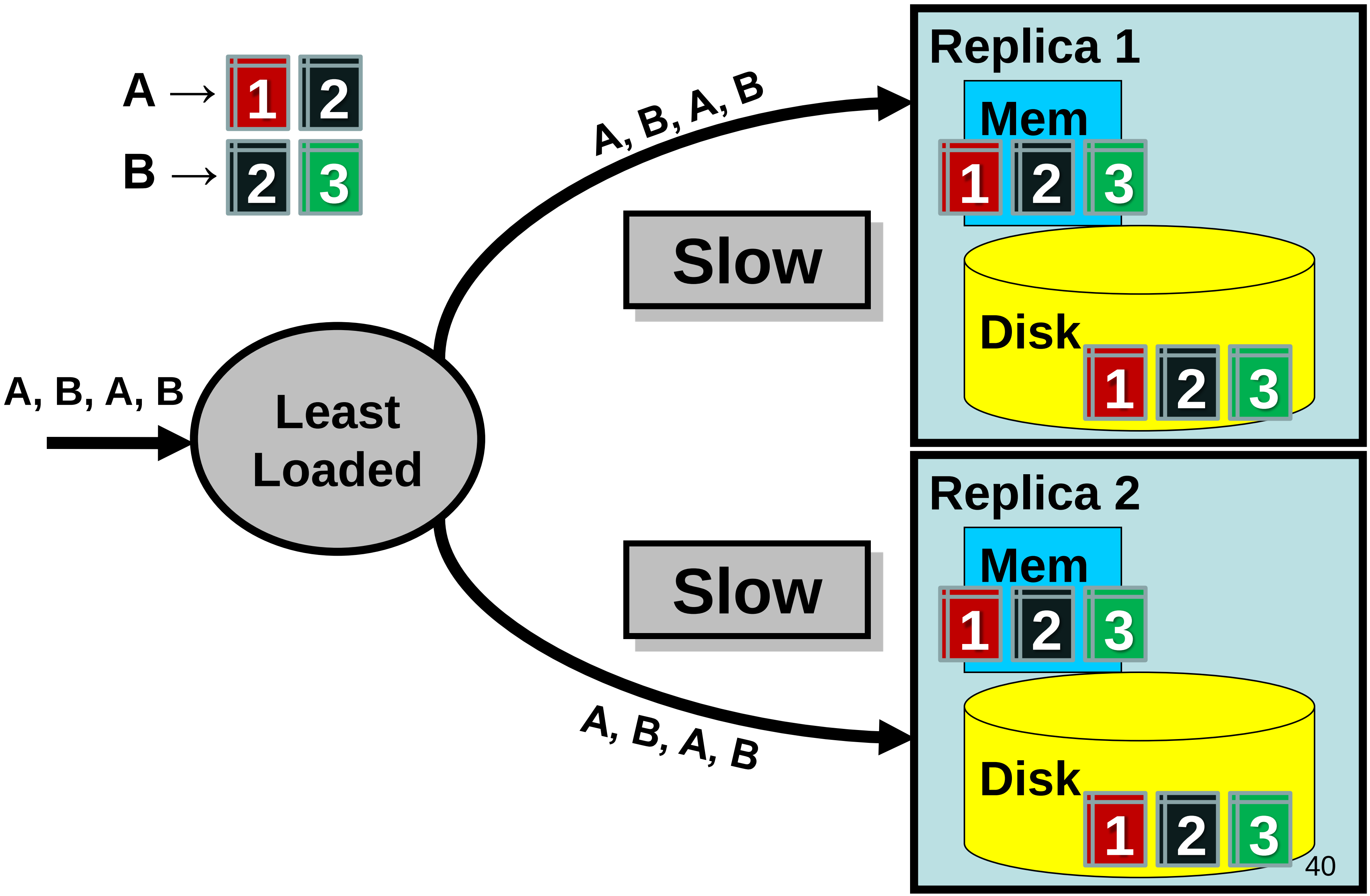
Memory



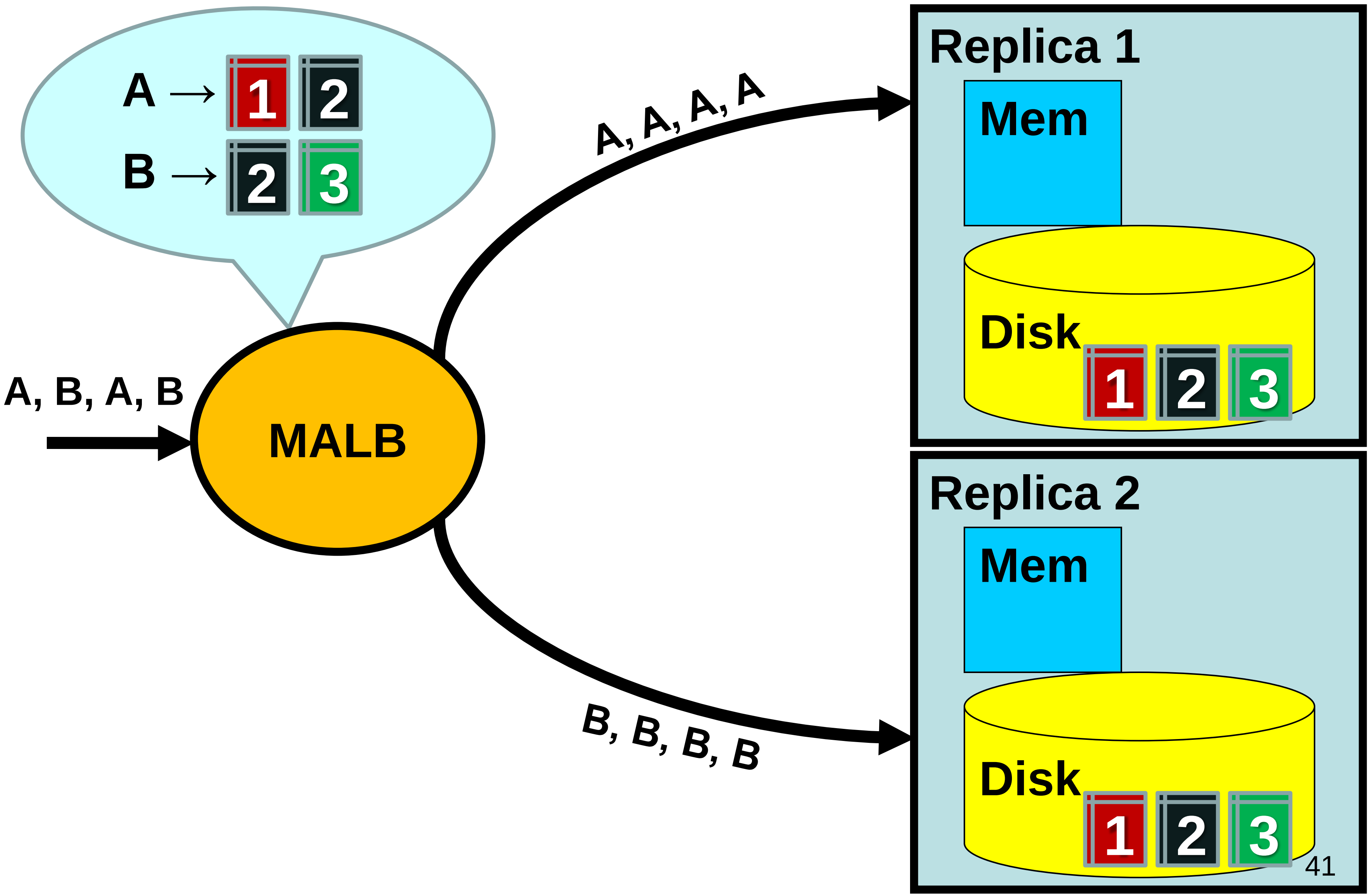
Read Data From Disk



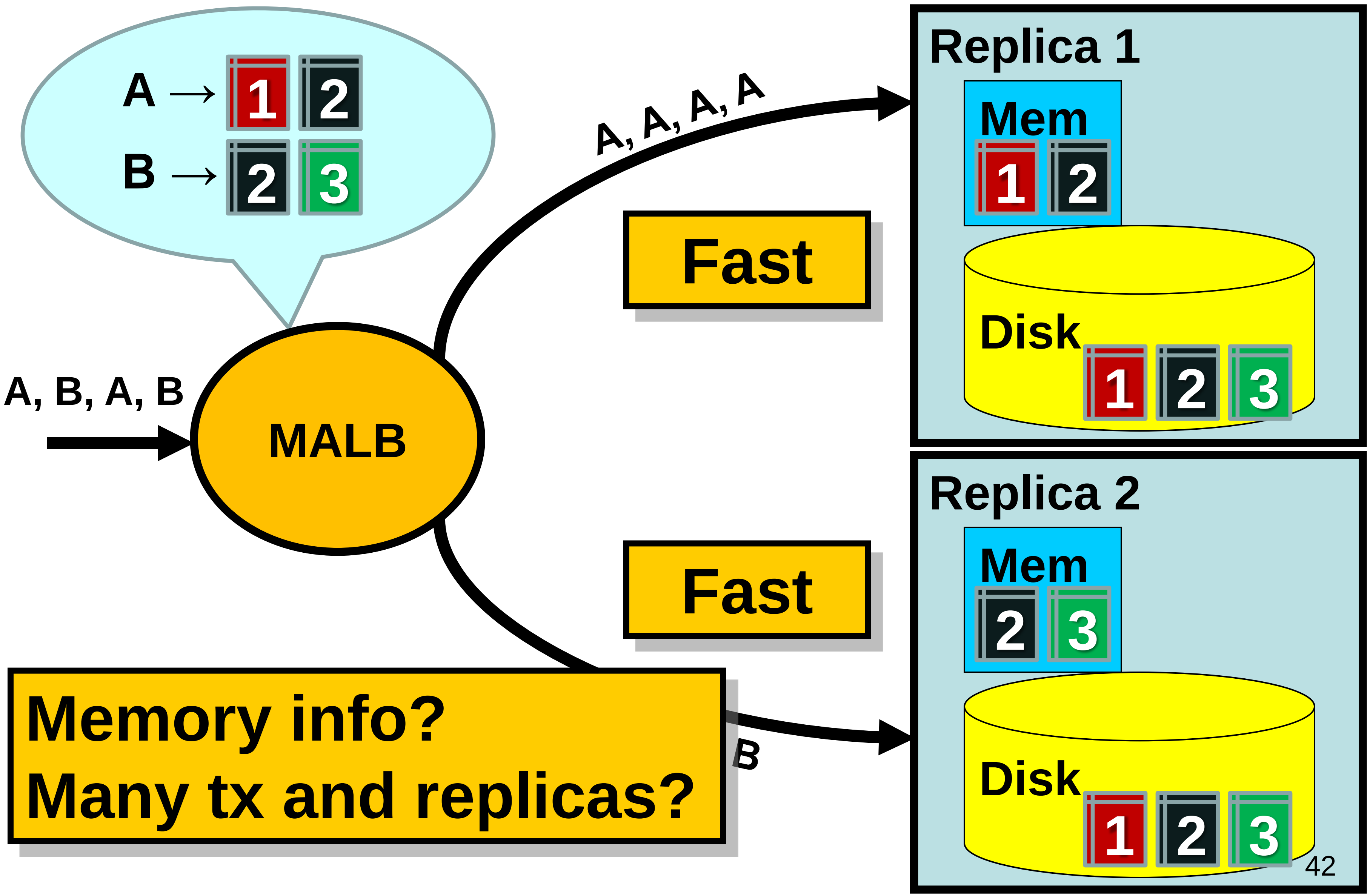
Read Data From Disk



Data Fits in Memory



Data Fits in Memory



Memory info?
Many tx and replicas?

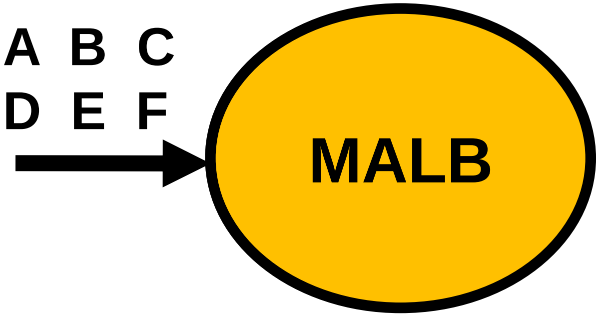
Estimate Tx Memory Needs

- **Exploit tx execution plan**
  - Which tables & indices are accessed
  - Their access pattern
    - Linear scan, direct access
- **Metadata from database**
  - Sizes of tables and indices

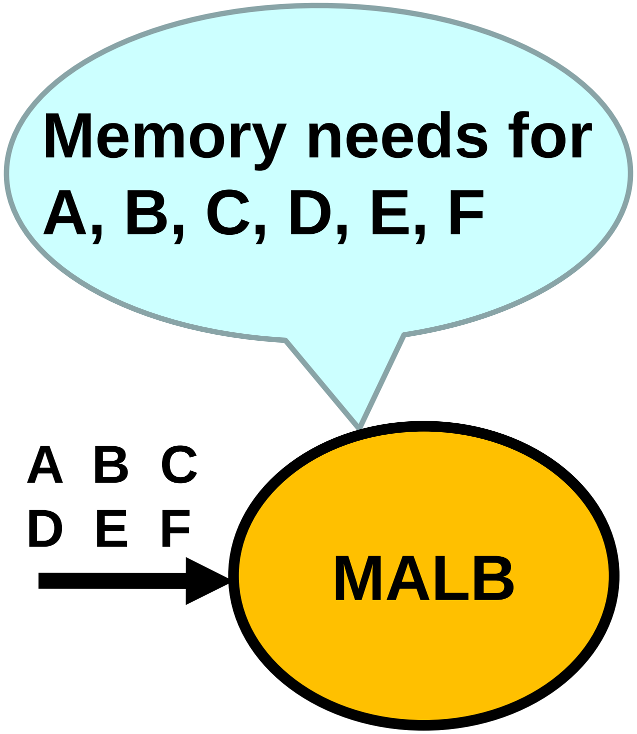
Grouping Transactions

- **Objective**
  – Construct tx groups that fit together in memory
- **Bin packing**
  – Item: tx memory needs
  – Bin: memory of replica
  – Heuristic: Best Fit Decreasing
- **Allocate replicas to tx groups**
  – Adjust for group loads

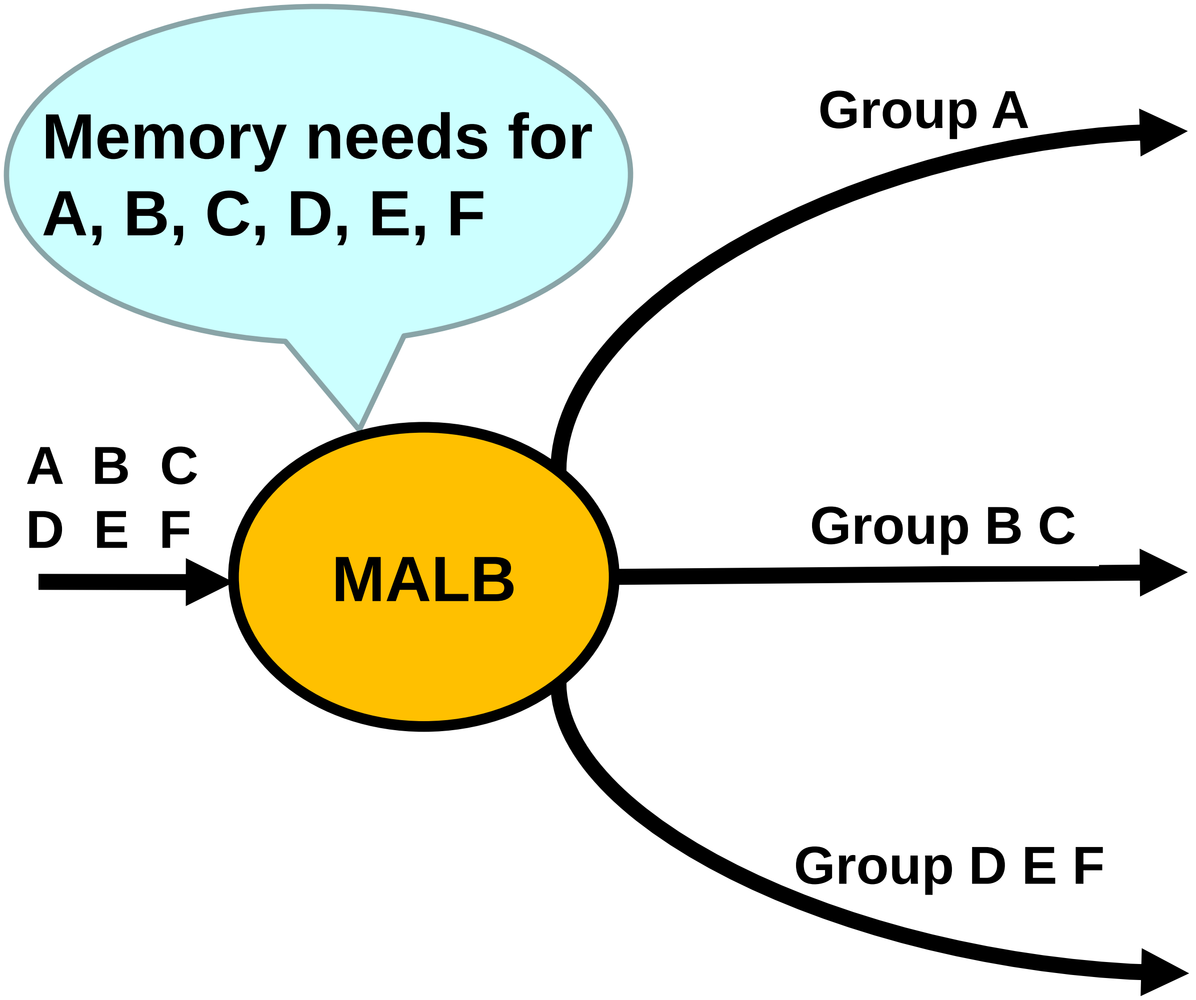
MALB in Action



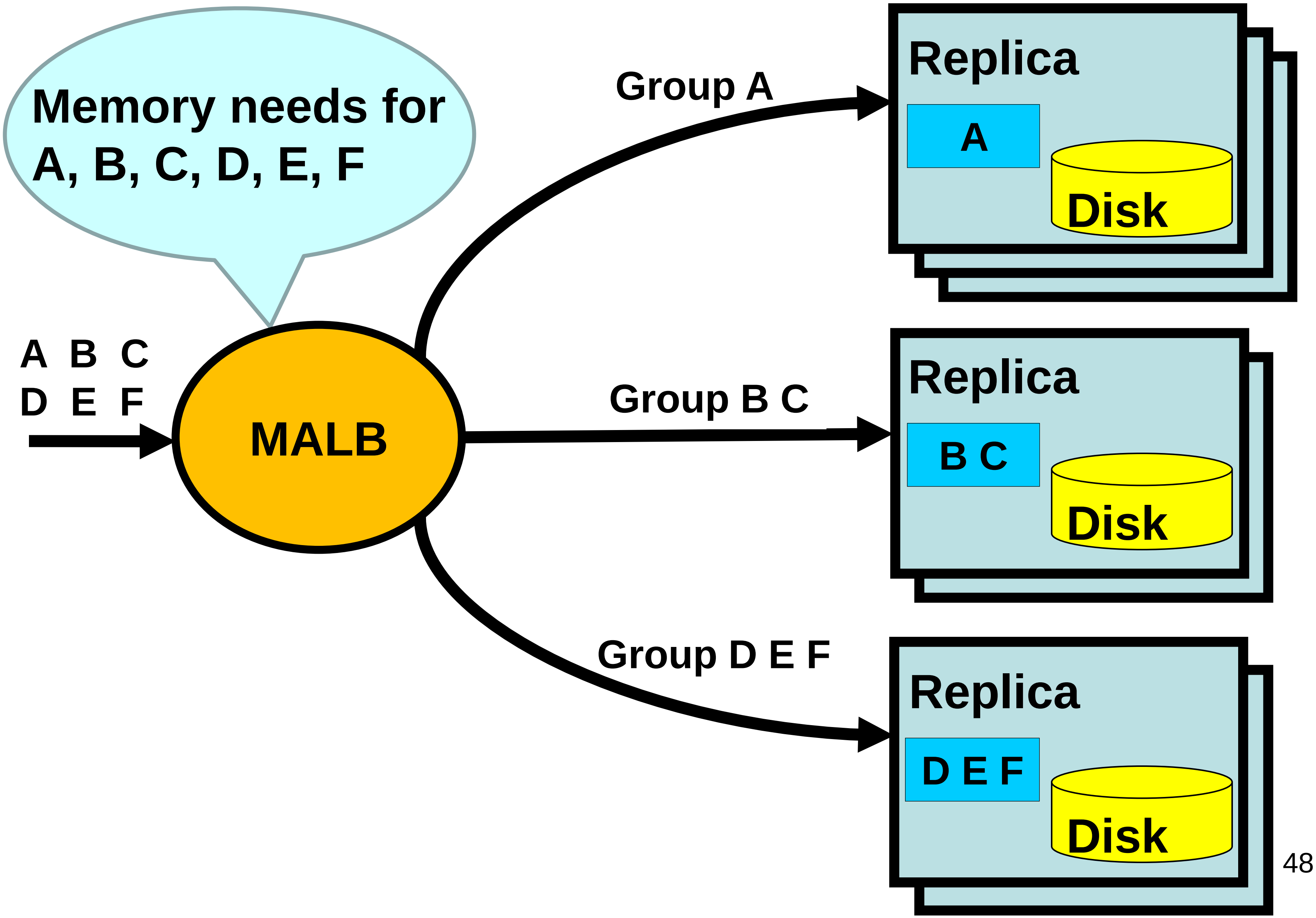
MALB in Action



MALB in Action



MALB in Action



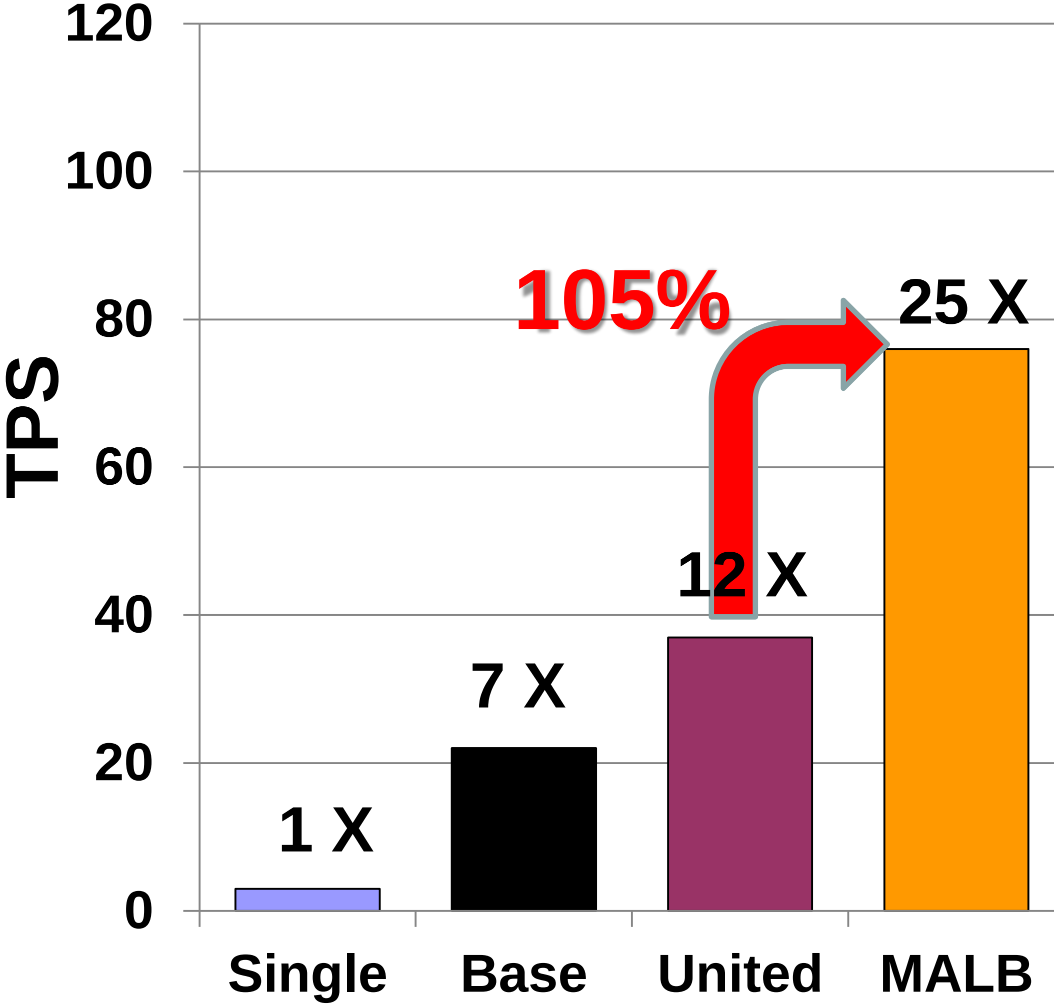
MALB Summary

- **Objective**
  – Optimize for in-memory execution
- **Method**
  – Estimate tx memory needs
  – Construct tx groups
  – Allocate replicas to tx groups

Experimental Evaluation

- **Implementation**
  - No change in consistency
  - Still middleware
- **Compare**
  - **United**: efficient baseline system
  - **MALB**: exploits working set information
- **Same environment**
  - Linux cluster running PostgreSQL
  - Workload: TPC-W Ordering (50% update txs)

MALB Doubles Throughput

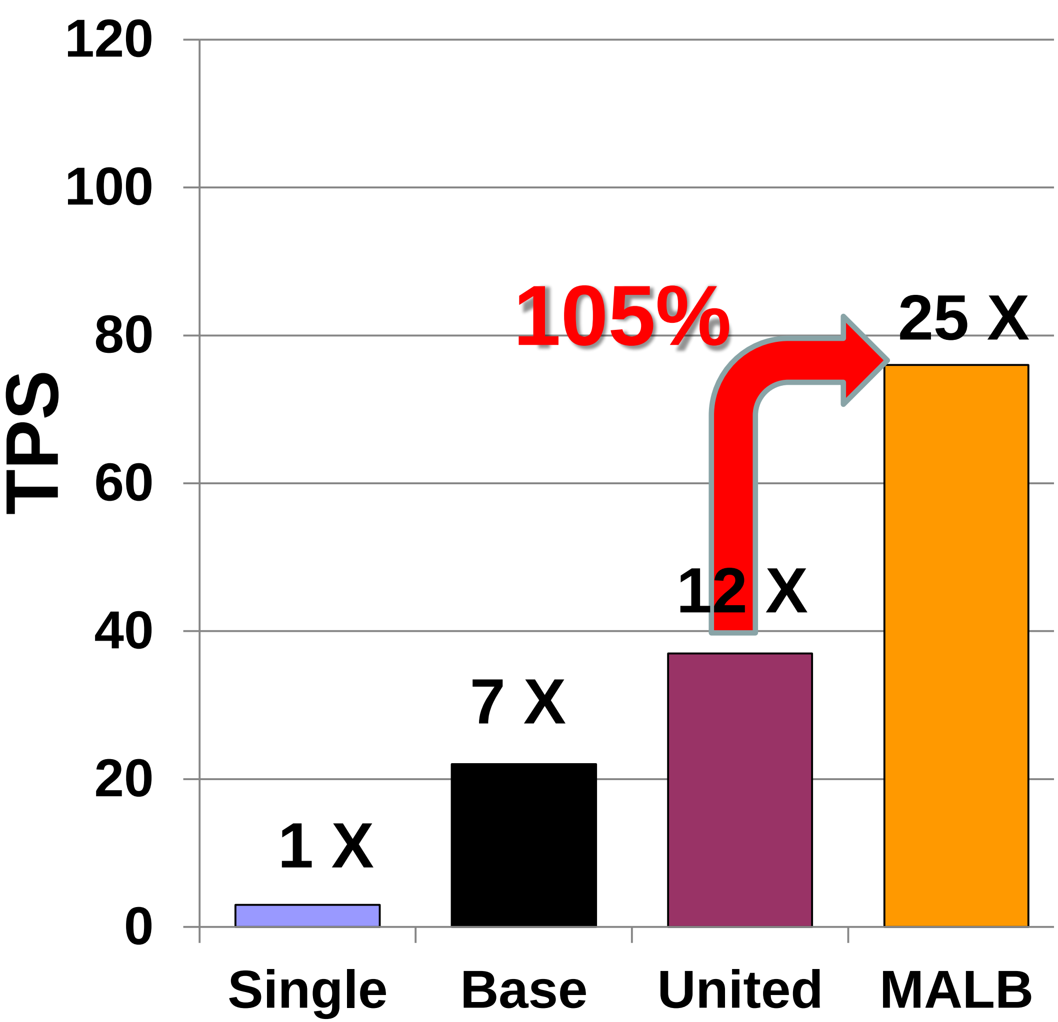


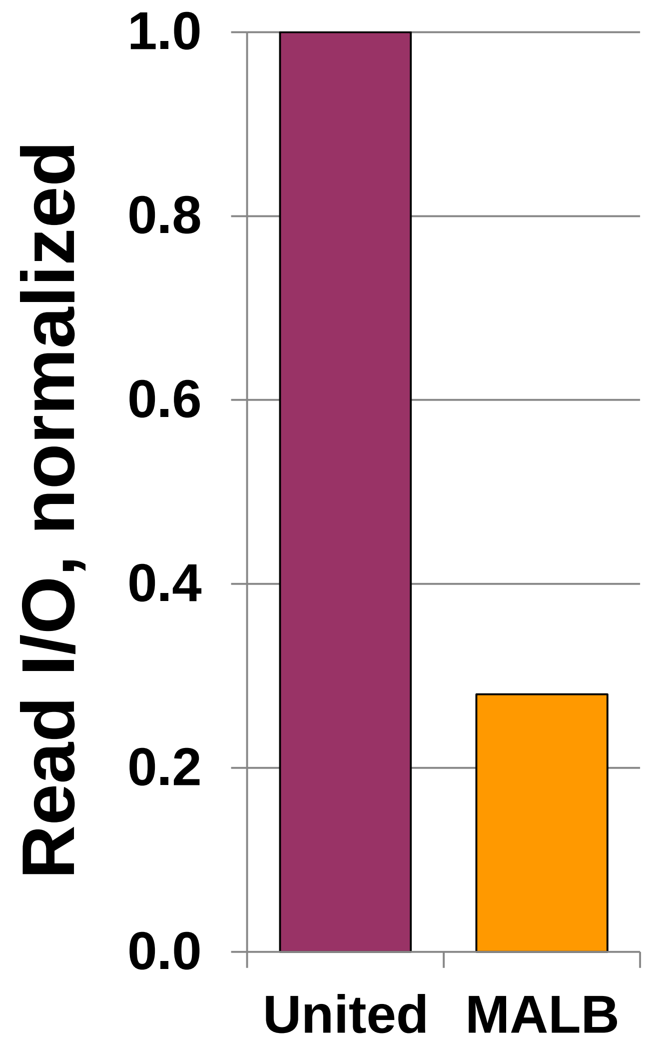
TPC-W

Ordering

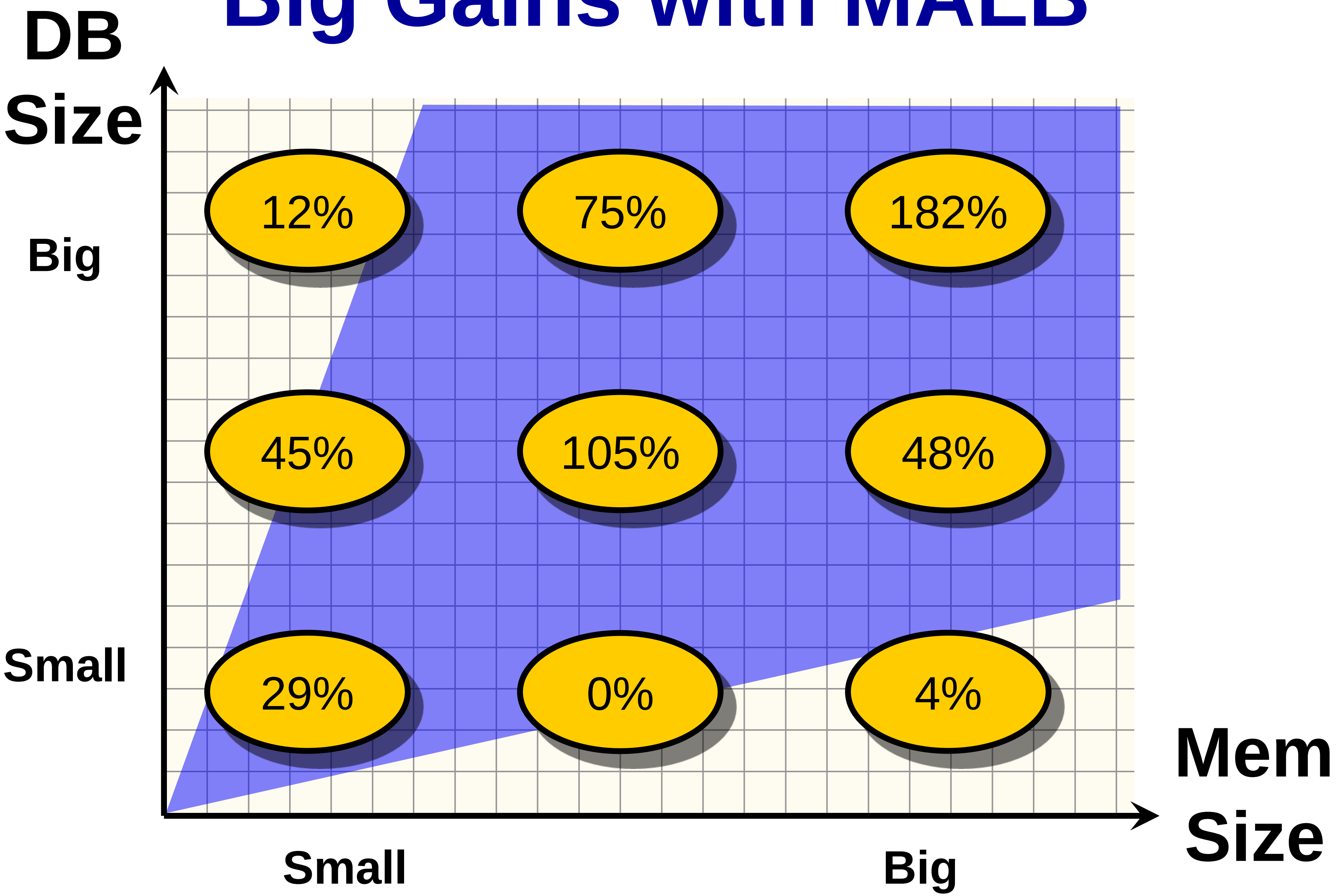
16 replicas

MALB Doubles Throughput

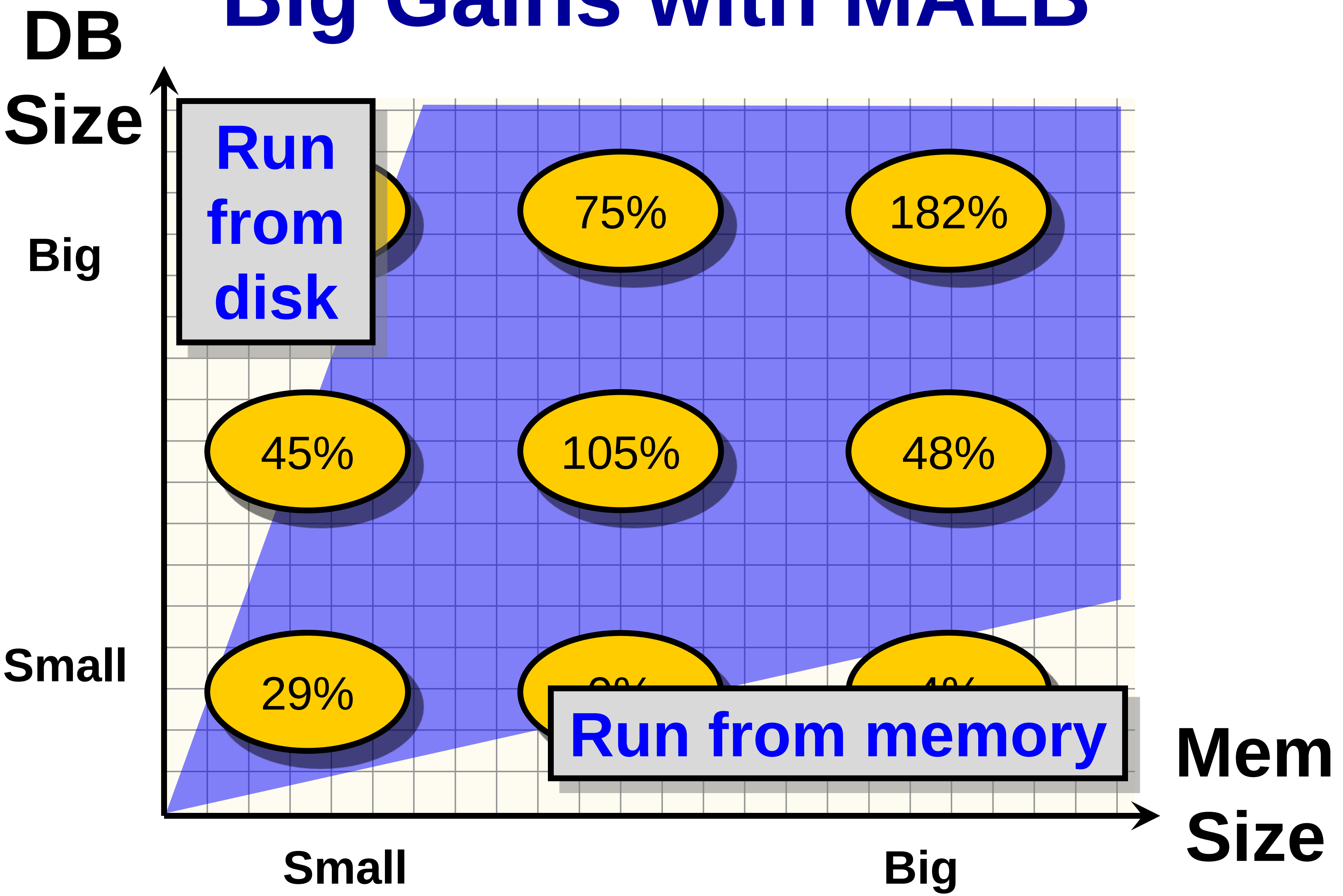




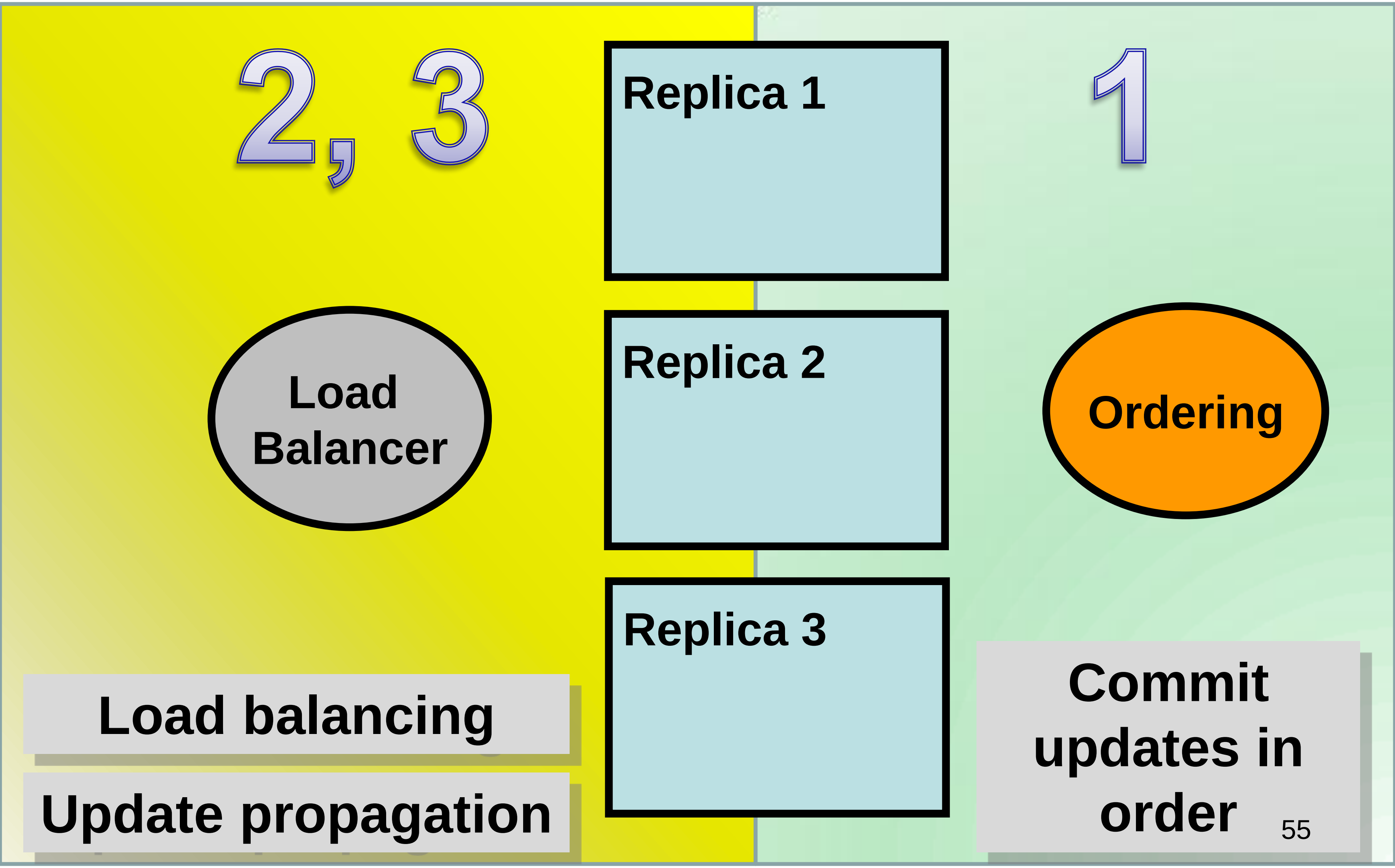
Big Gains with MALB



Big Gains with MALB



Roadmap

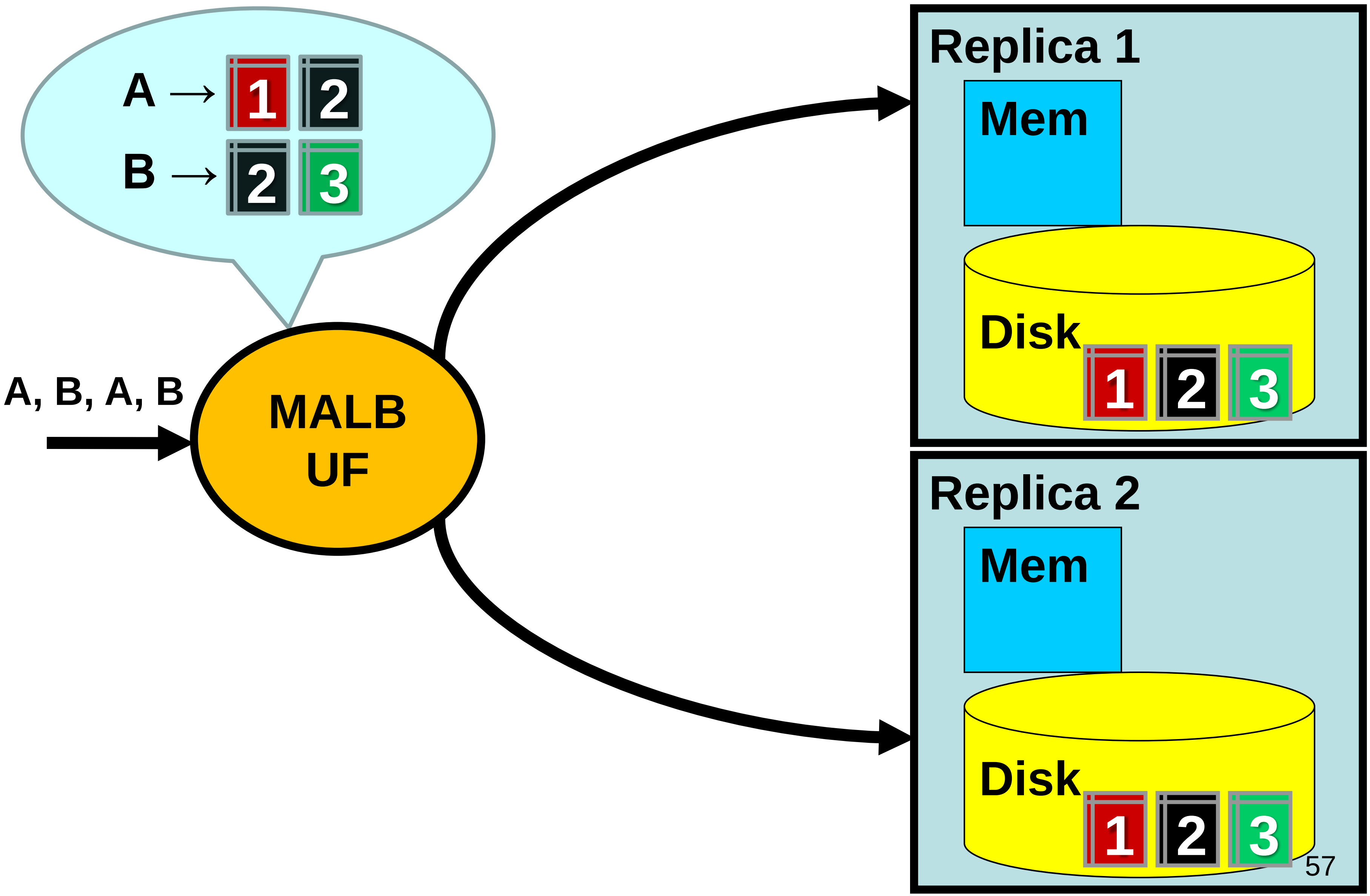


2, 3
1
Replica 1
Replica 2
Replica 3
Load Balancer
Ordering
Load balancing
Update propagation
Commit updates in order

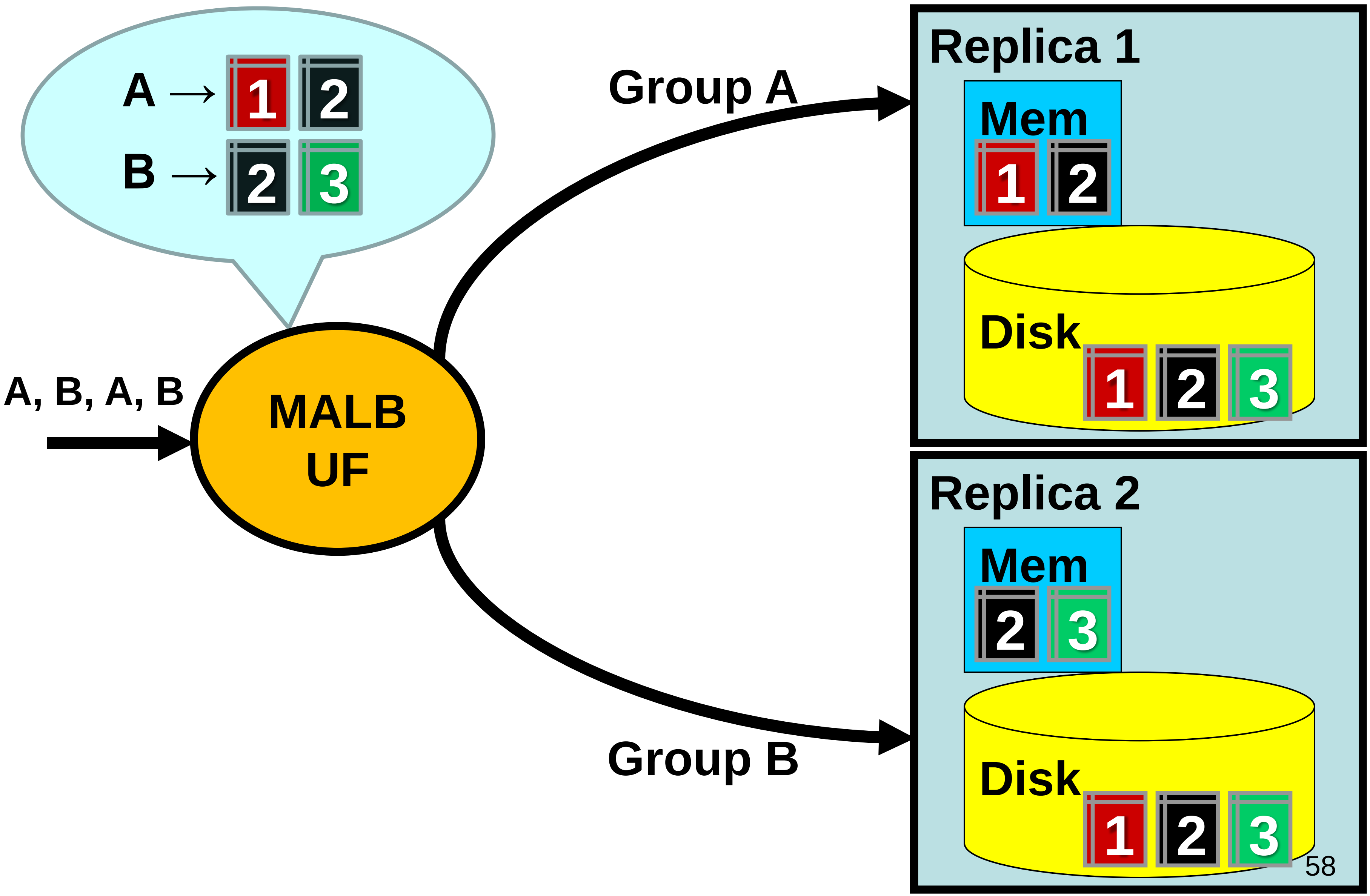
Key Idea

- **Traditional:**
  - Propagate updates everywhere
- **Update Filtering:**
  - Propagate updates to where they are needed

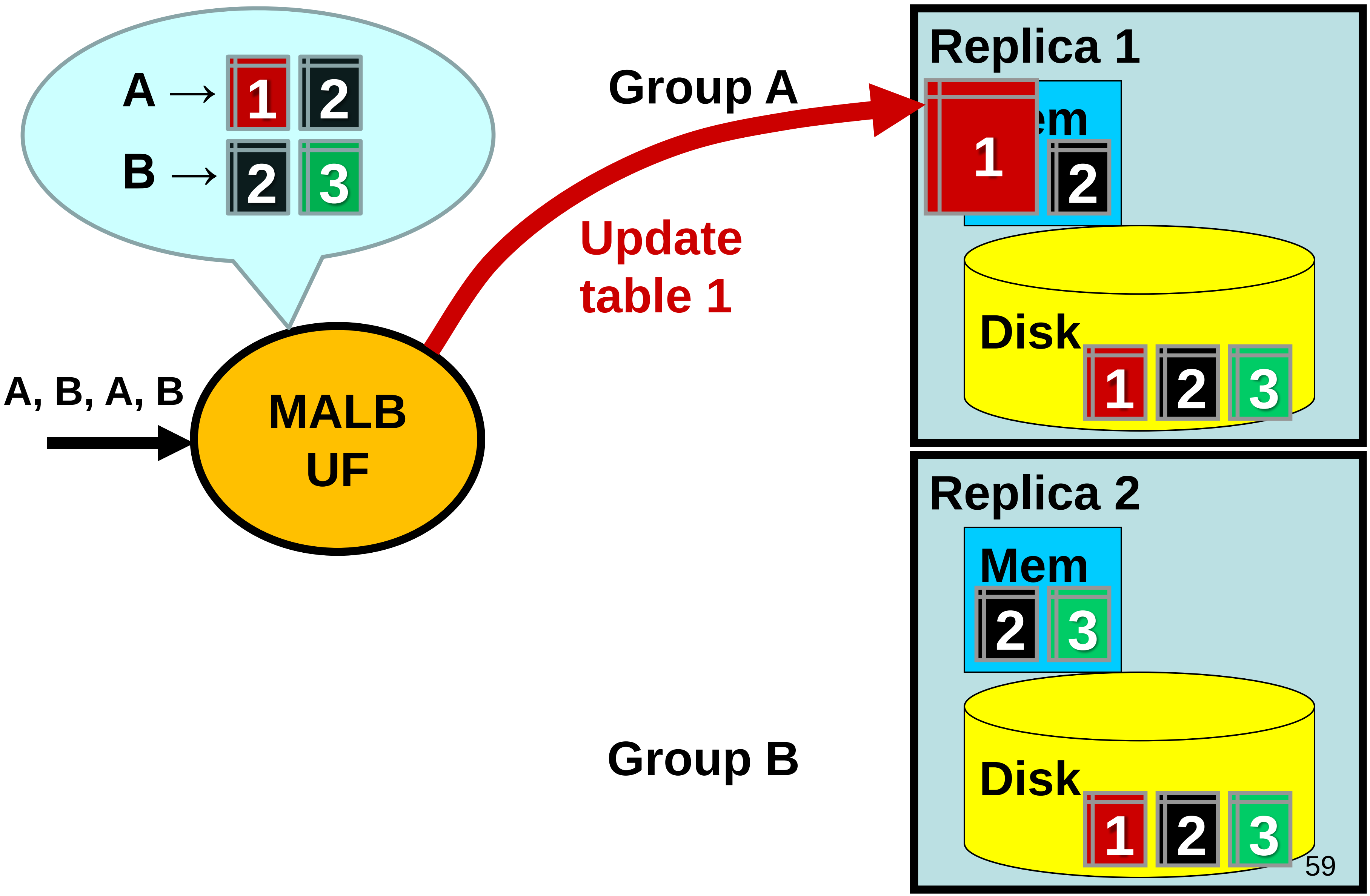
Update Filtering Example



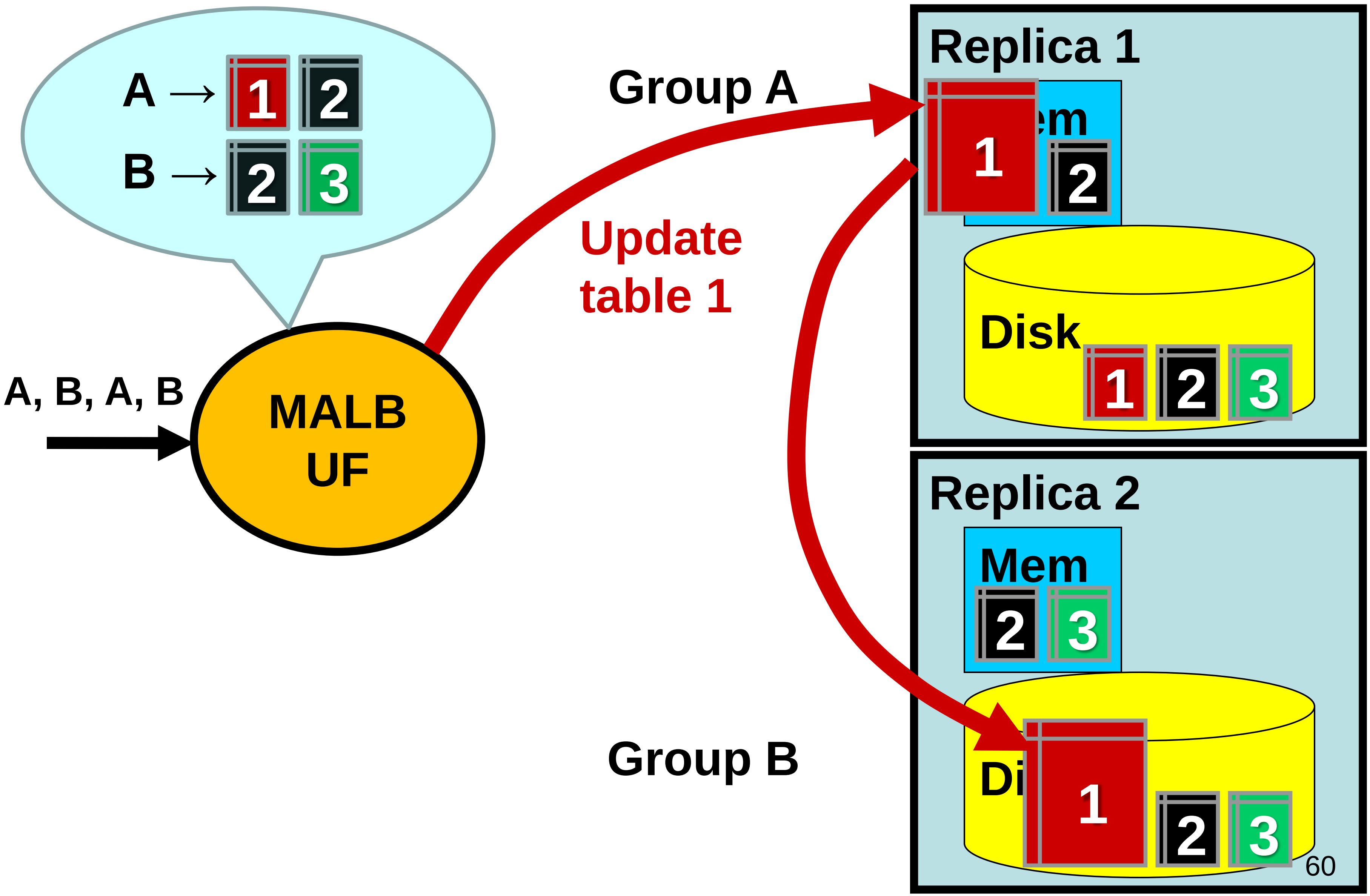
Update Filtering Example



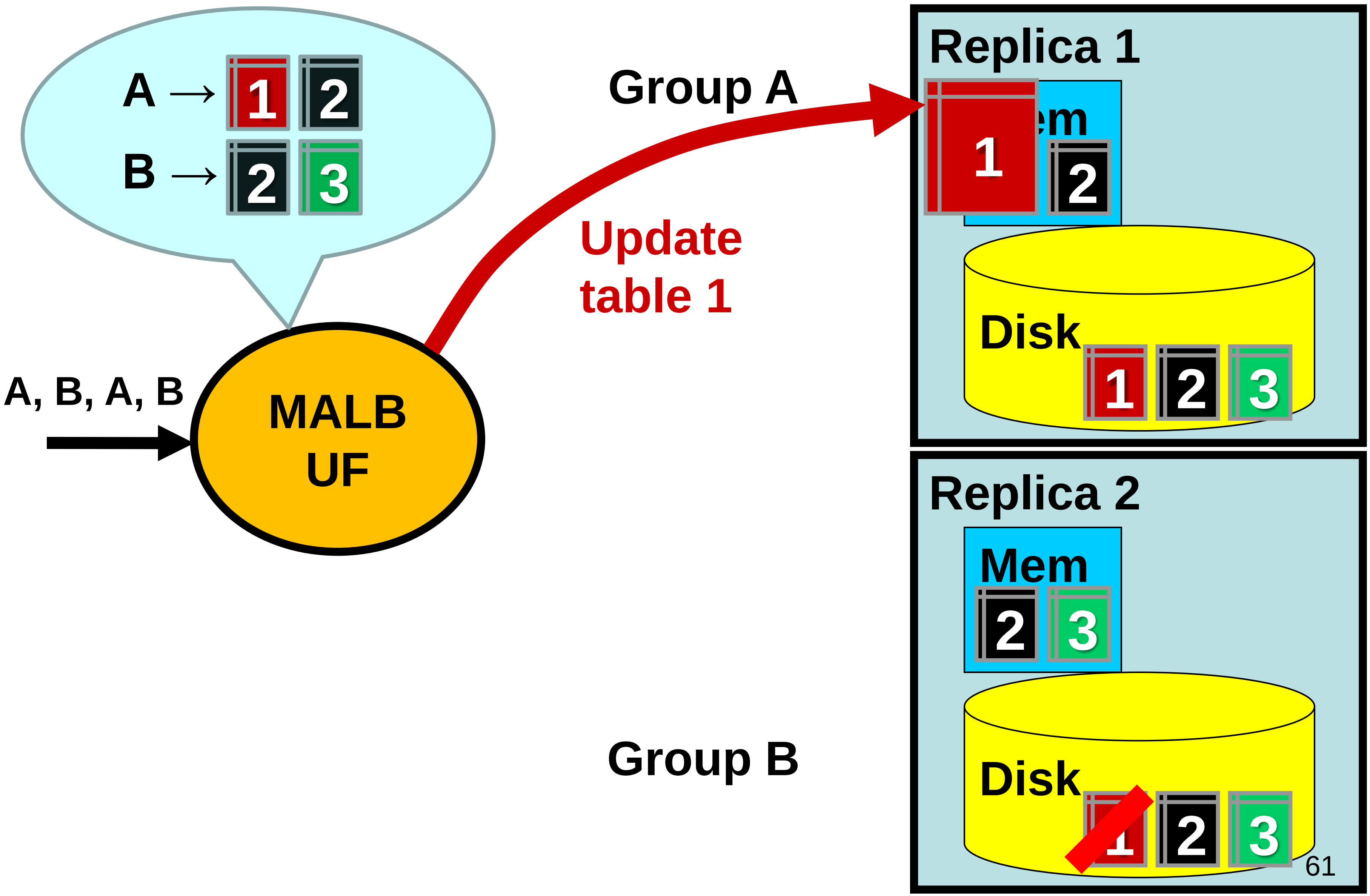
Update Filtering Example



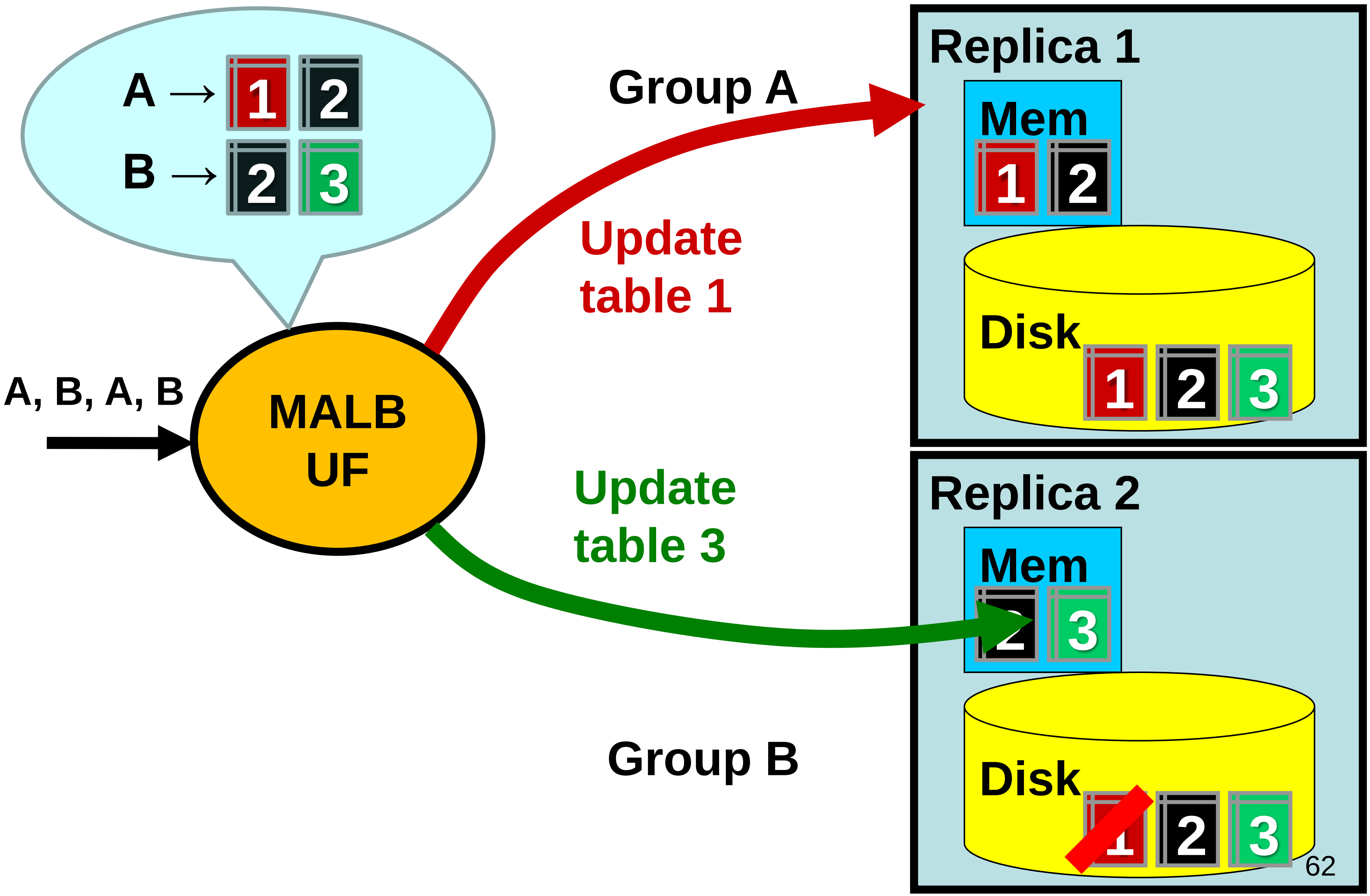
Update Filtering Example



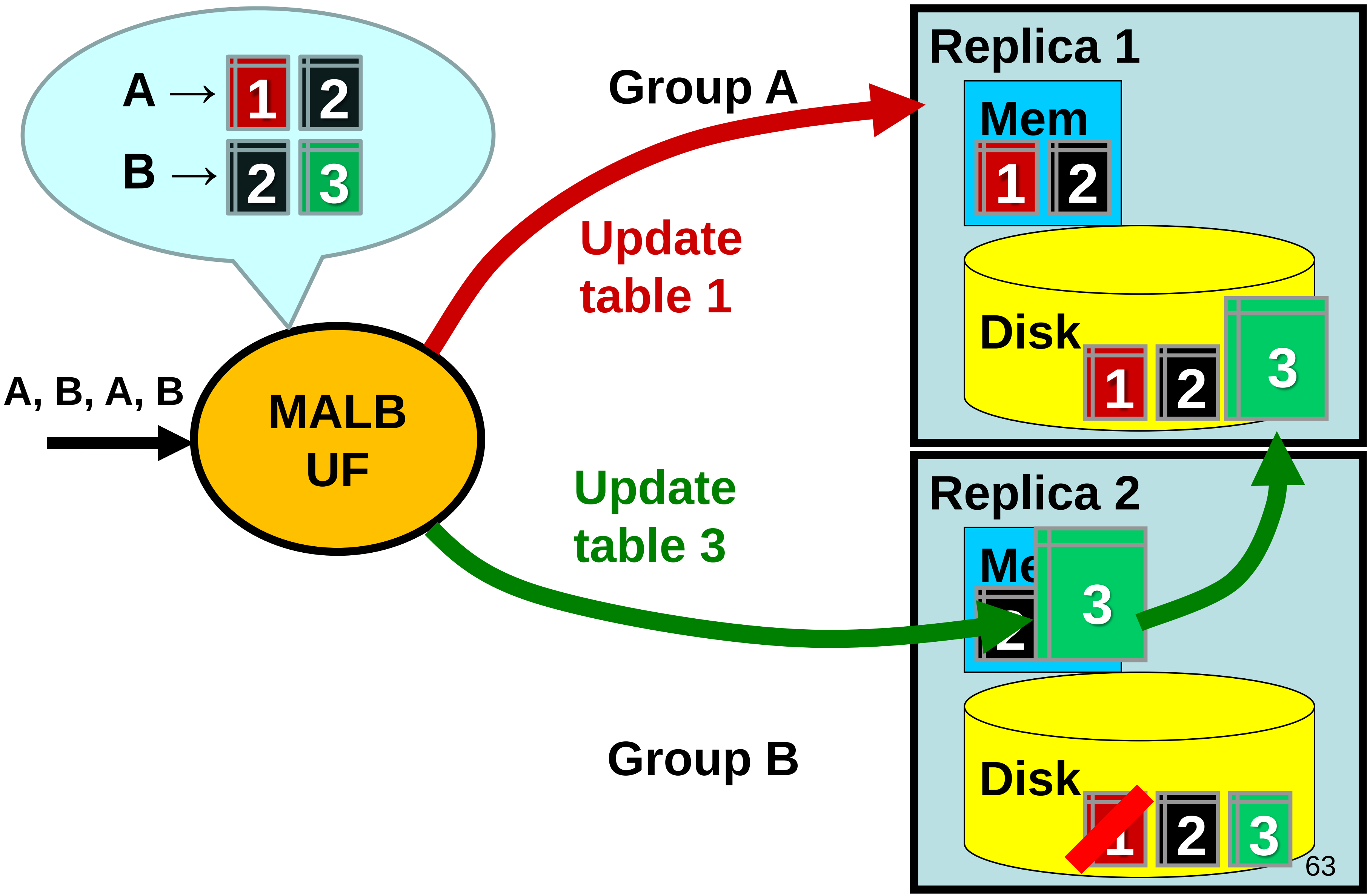
Update Filtering Example



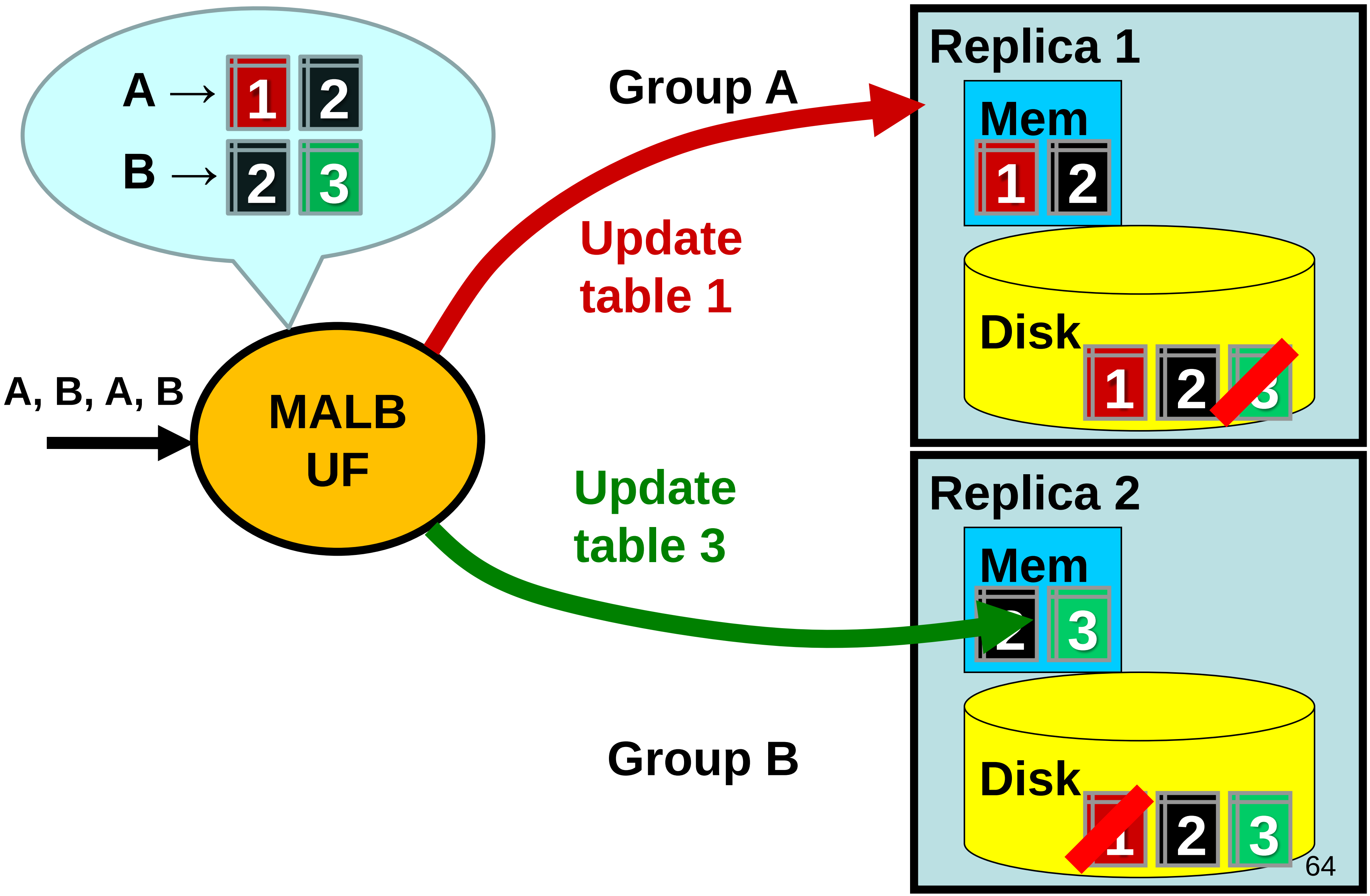
Update Filtering Example



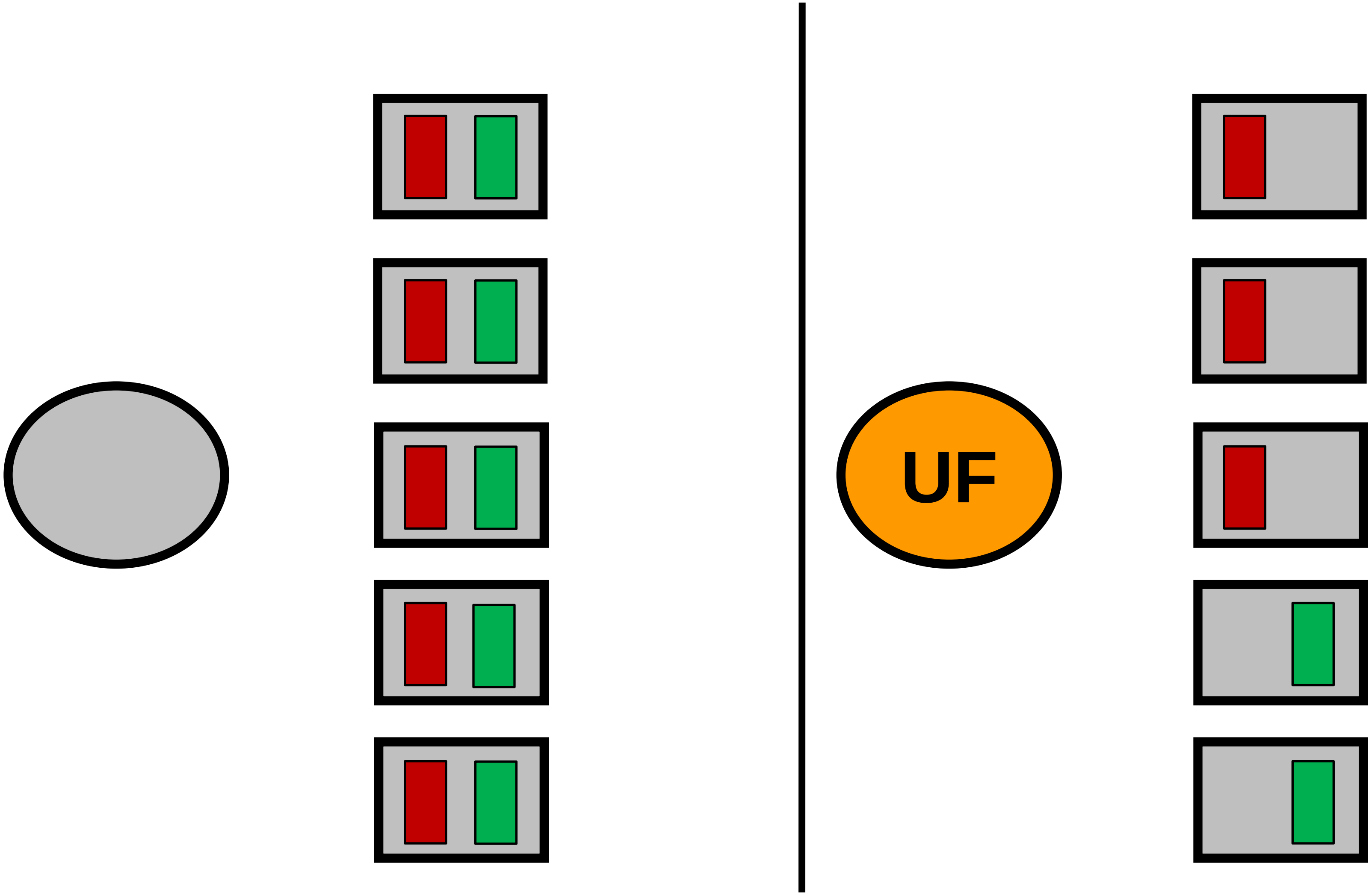
Update Filtering Example



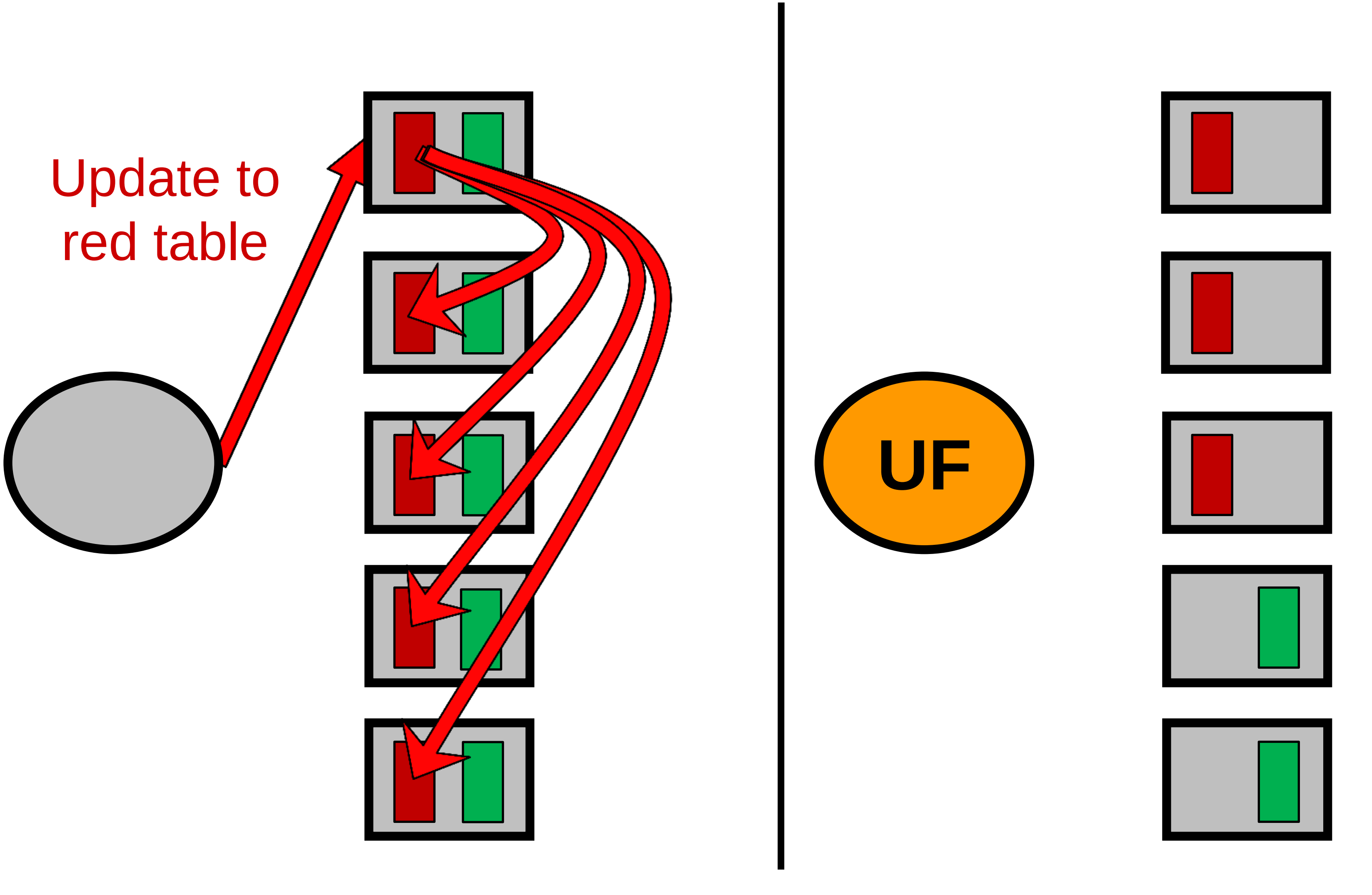
Update Filtering Example



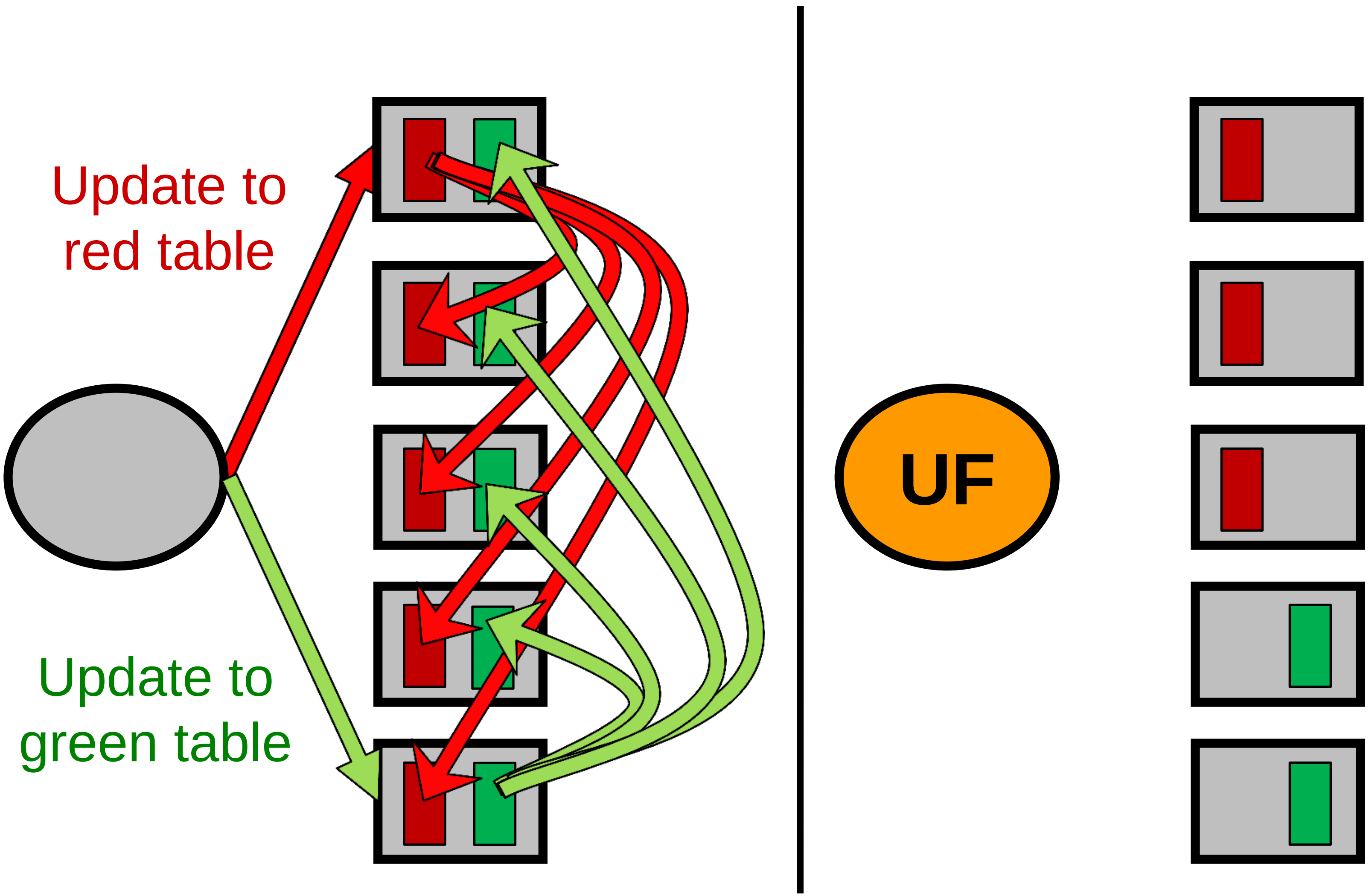
Update Filtering in Action



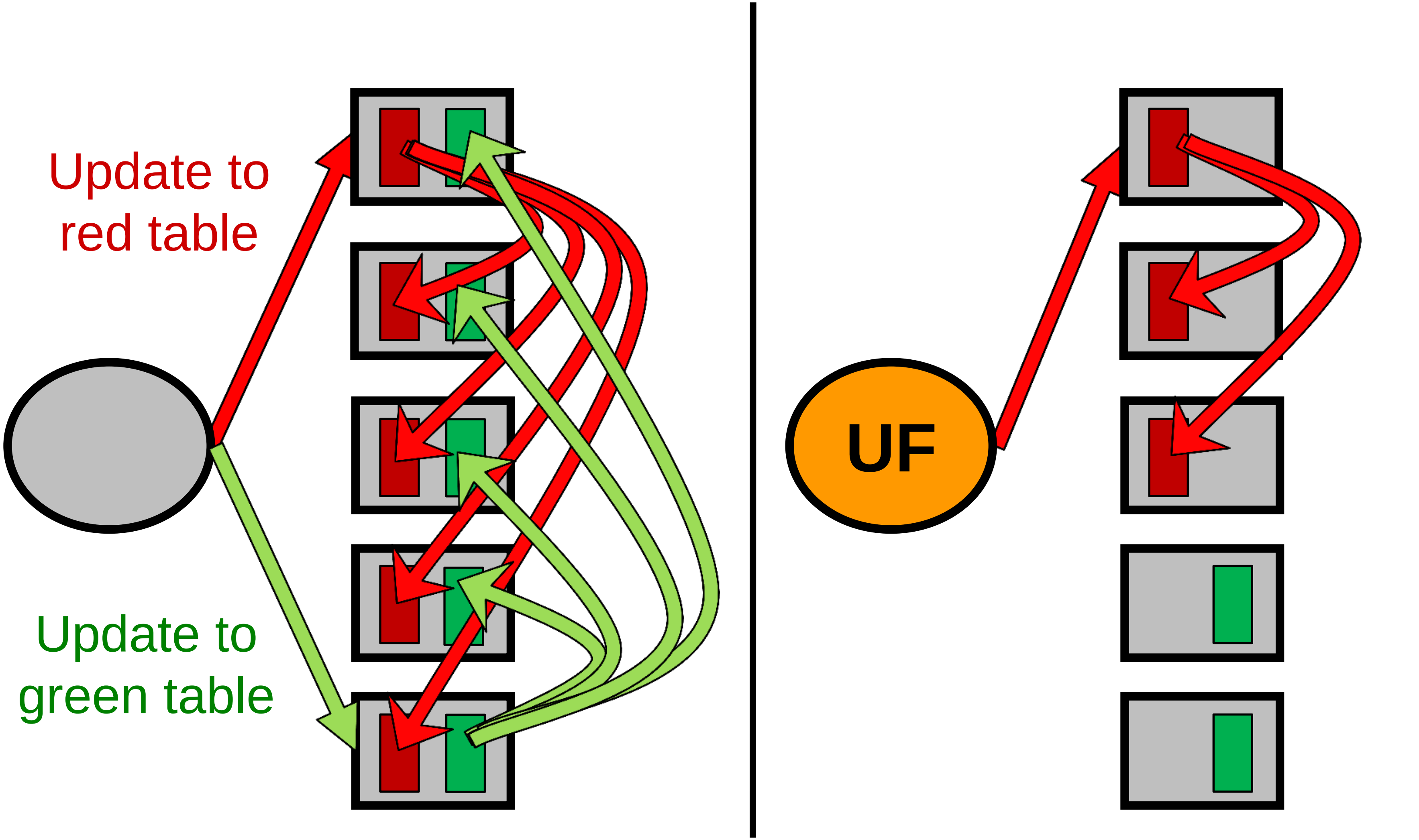
Update Filtering in Action



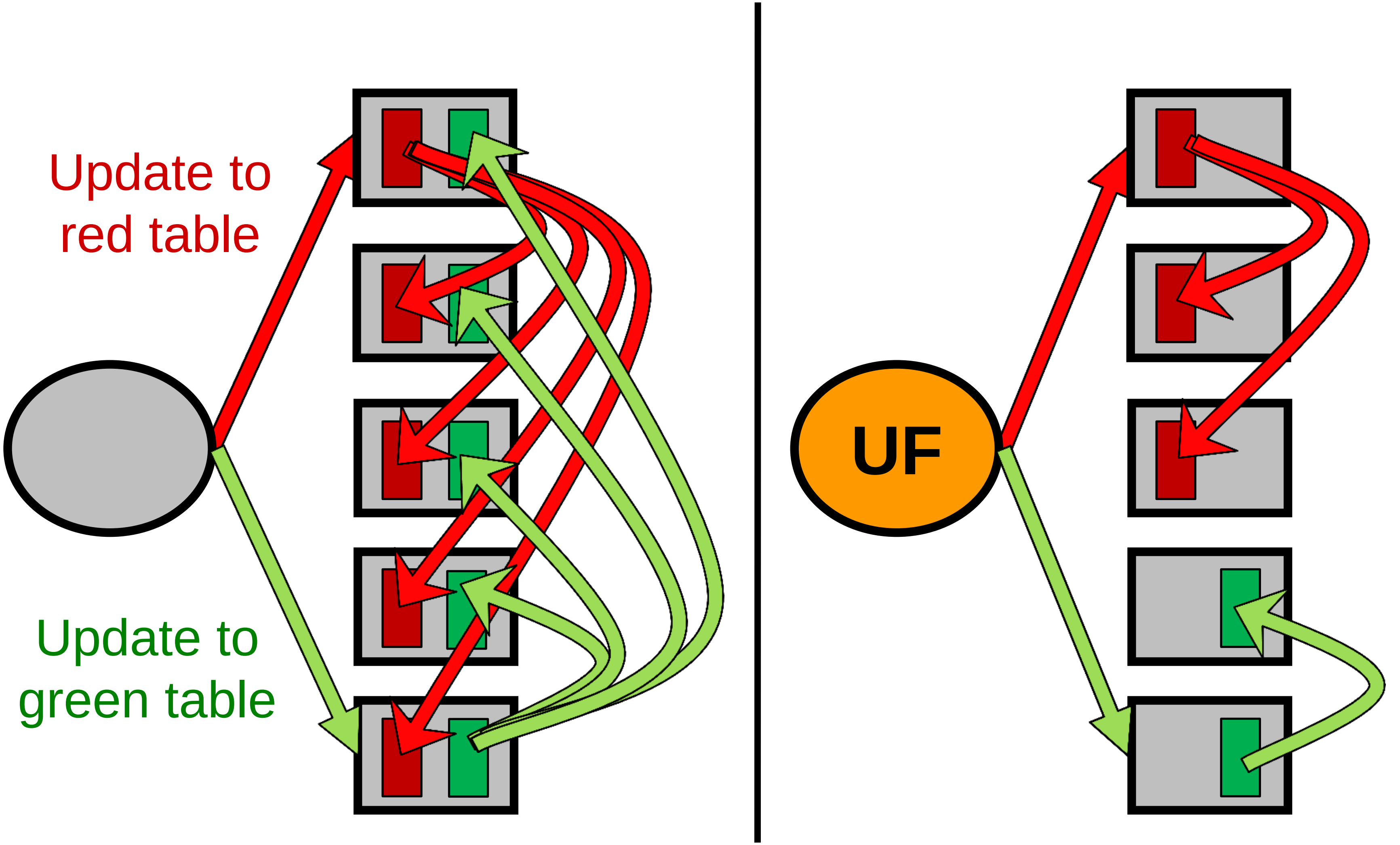
Update Filtering in Action



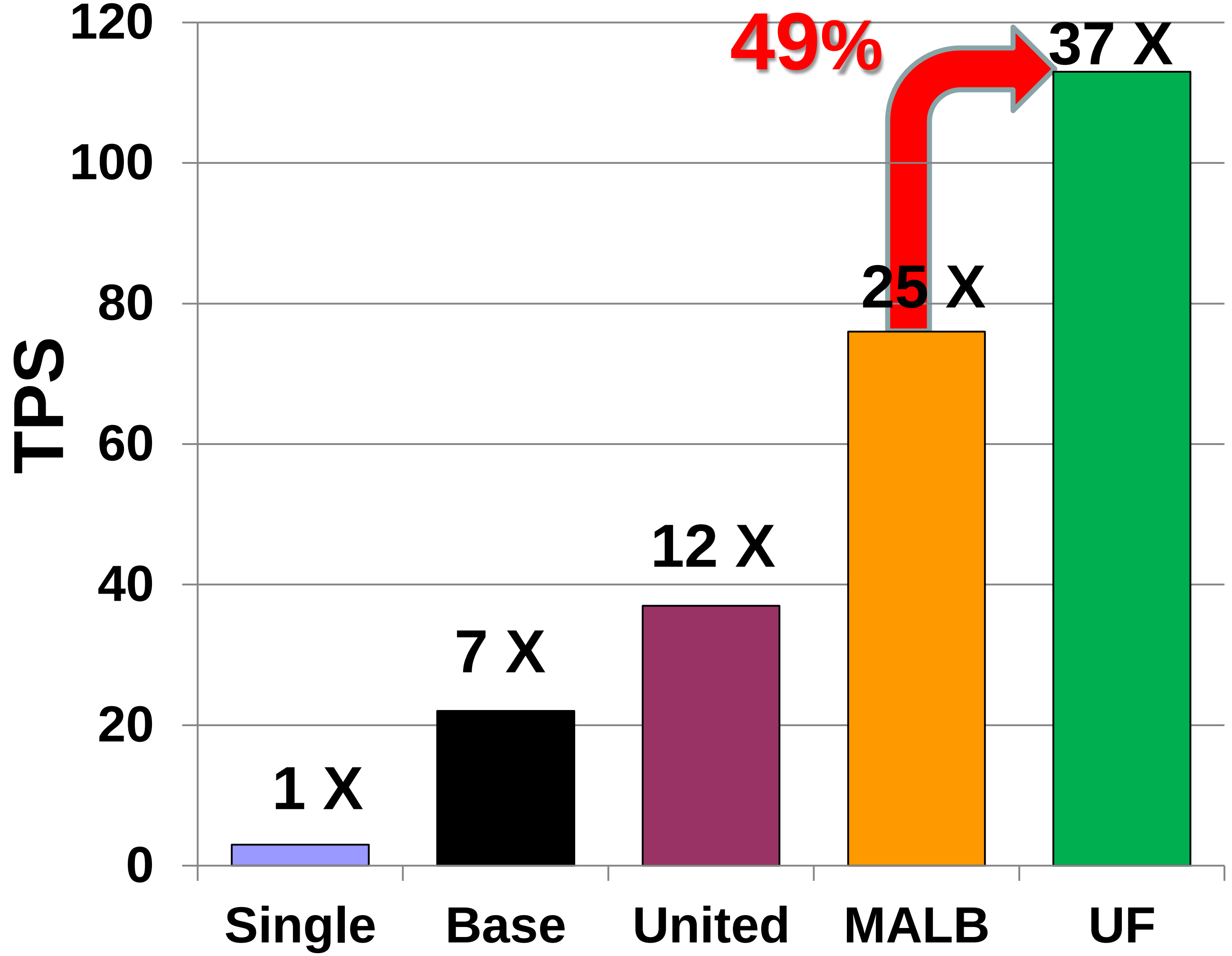
Update Filtering in Action



Update Filtering in Action



MALB+UF Triples Throughput

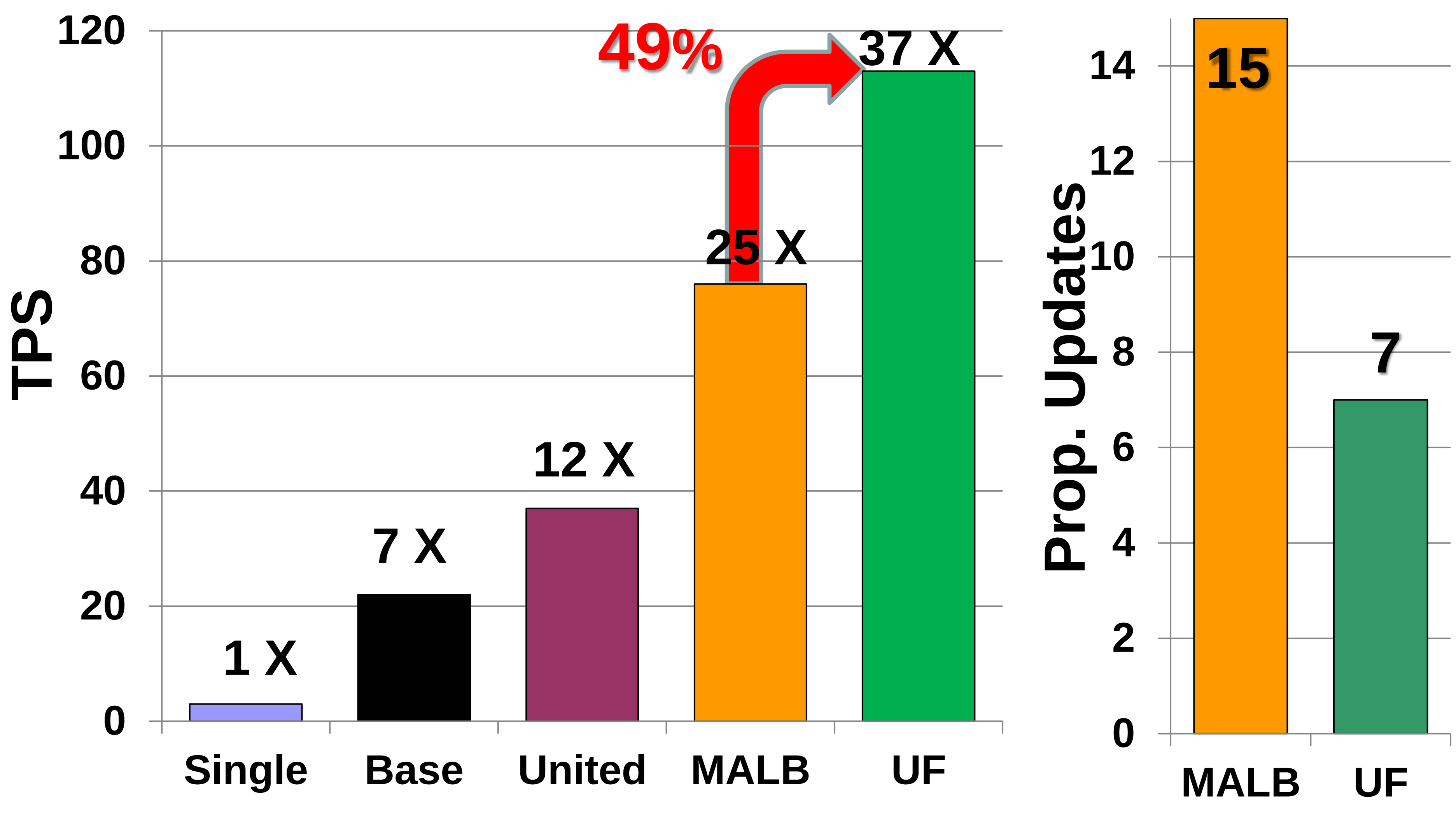


TPC-W

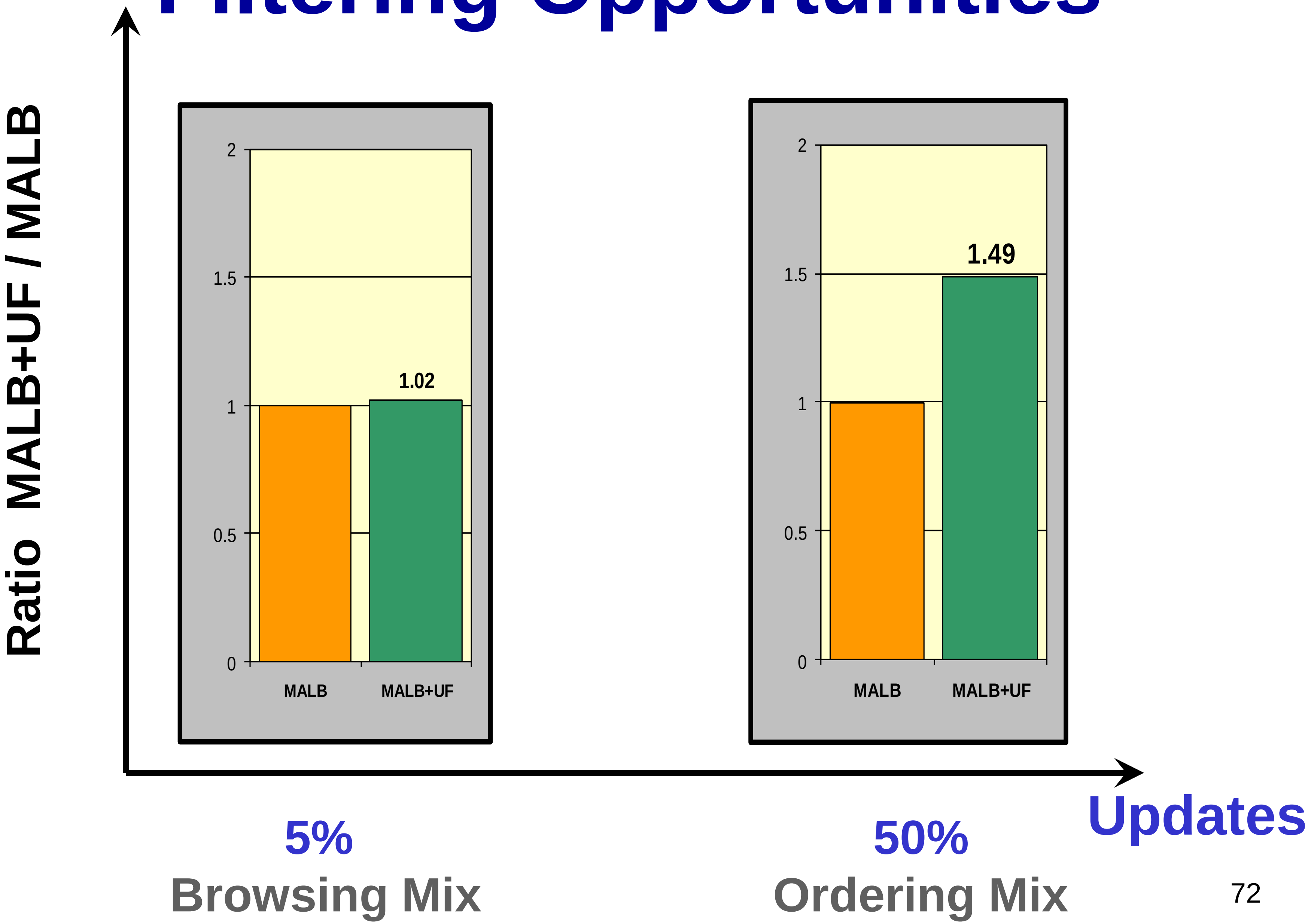
Ordering

16 replicas

MALB+UF Triples Throughput



Filtering Opportunities



Conclusions

1. **Commit updates in order**
   – Perform serial synchronous disk writes
   – Unite ordering and durability
2. **Load balancing**
   – Optimize for equal load: memory contention
   – MALB: optimize for in-memory execution
3. **Update propagation**
   – Propagate updates everywhere
   – Update filtering: propagate to where needed