Managing Soft-errors in Transactional Systems

<u>Mohamed Mohamedin</u>, Roberto Palmieri, and Binoy Ravindran

Virginia Tech

USA

{mohamedin,robertop,binoy}@vt.edu

DPDNS'14

What are Soft-errors?

- Transient faults that may happen anytime during application execution
- Caused by physical phenomena (e.g., cosmic particle strikes, electric noise)
- E.g., Soft-error can cause a single bit in a CPU register to flip causing transient failures

Do soft-errors represent a problem?

- Soft-errors are:
  - Random: Can occur anytime
  - Undetectable: No hardware interrupt is triggered
  - Corrupting: Can silently corrupt program data or crash the program

Soft-errors effect

CPU mathematical operation

101011101011

+

100010101001

=

1001110010100

Soft-errors effect

CPU mathematical operation

101011101011

+

100010101001

=

1001110010100

But if a soft-error happened

101011101011

+

100010101001

=

1001110010100

Soft-errors effect

CPU mathematical operation

101011101011

+

100010101001

=

1001110010100

But if a soft-error happened

101011101011

+

000010101001

=

1001110010100

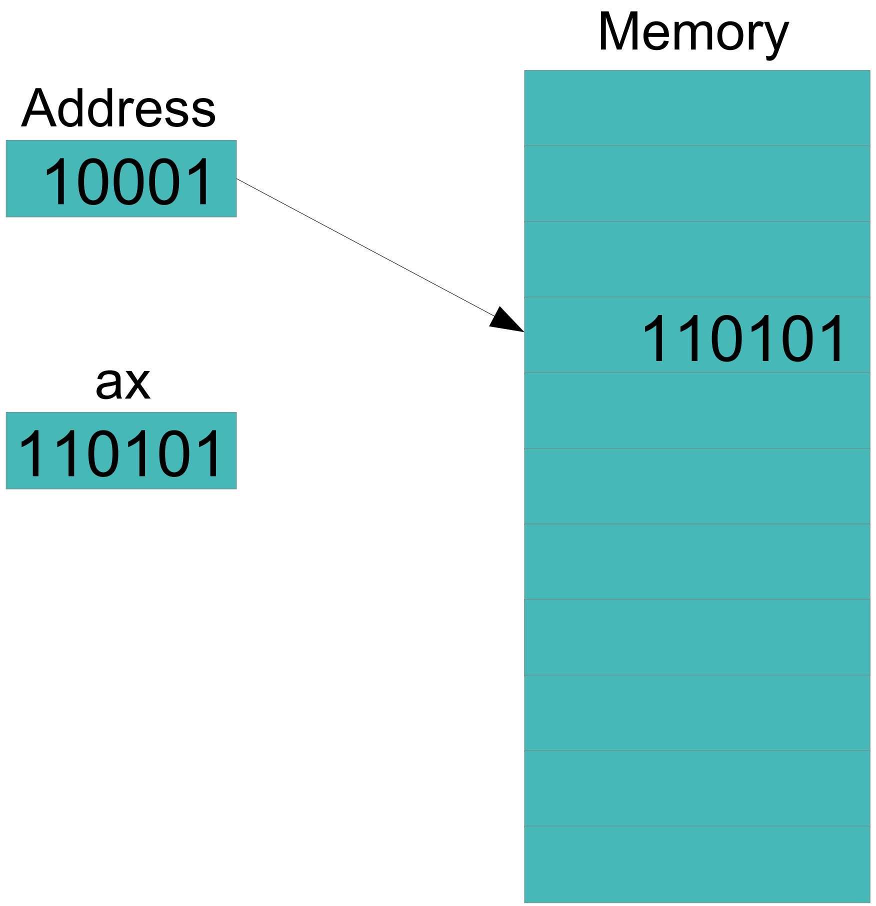
Soft-errors effect

CPU mathematical operation	But if a soft-error happened
101011101011	101011101011
+	+
100010101001	000010101001
=	=
1001110010100	0101110010100

Soft-errors effect

Memory store

e.g., mov [address], ax

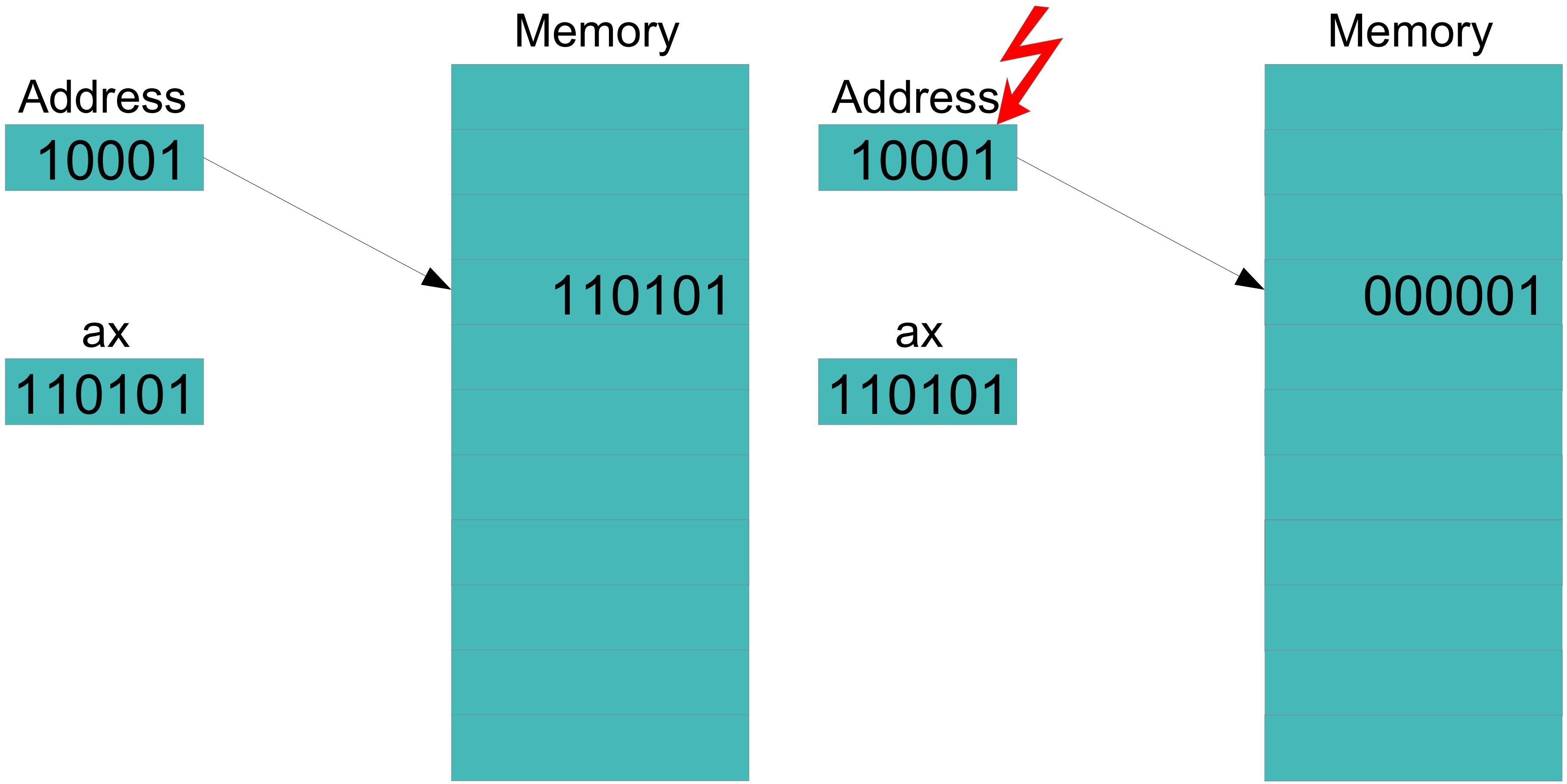


Soft-errors effect

Memory store

e.g., mov [address], ax

But if a soft-error happened

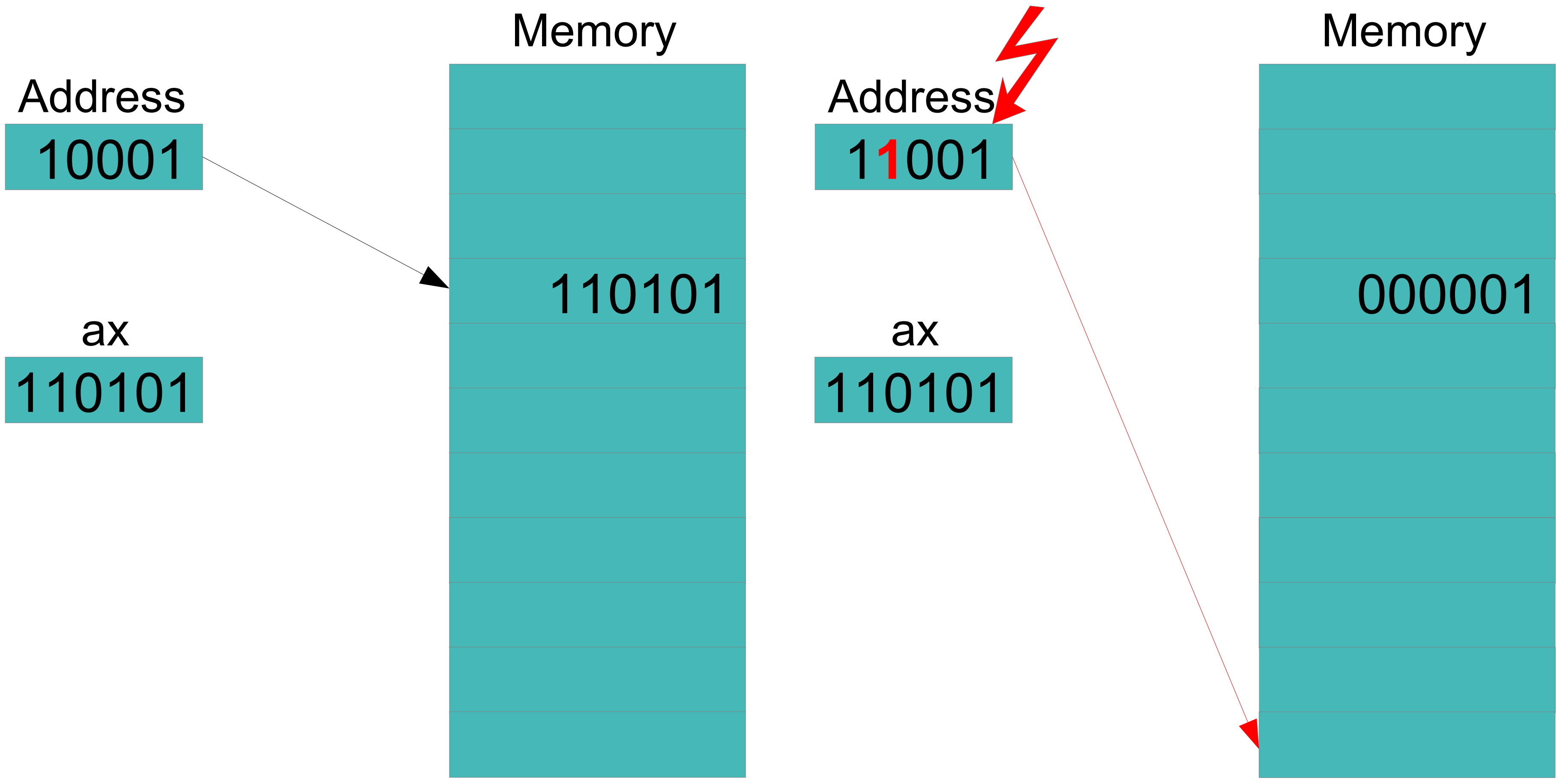


Soft-errors effect

Memory store

e.g., mov [address], ax

But if a soft-error happened

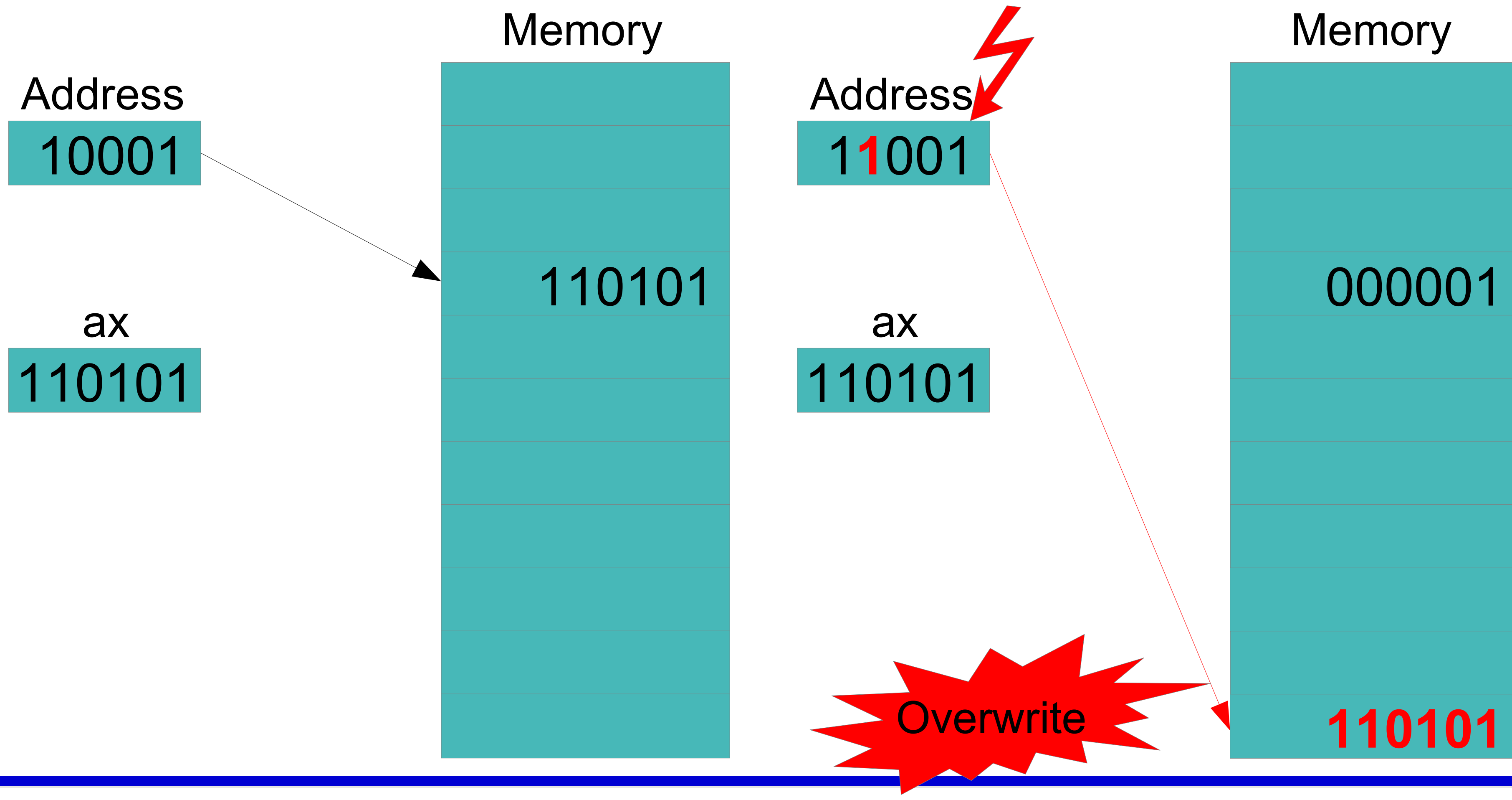


Soft-errors effect

Memory store

e.g., mov [address], ax

But if a soft-error happened

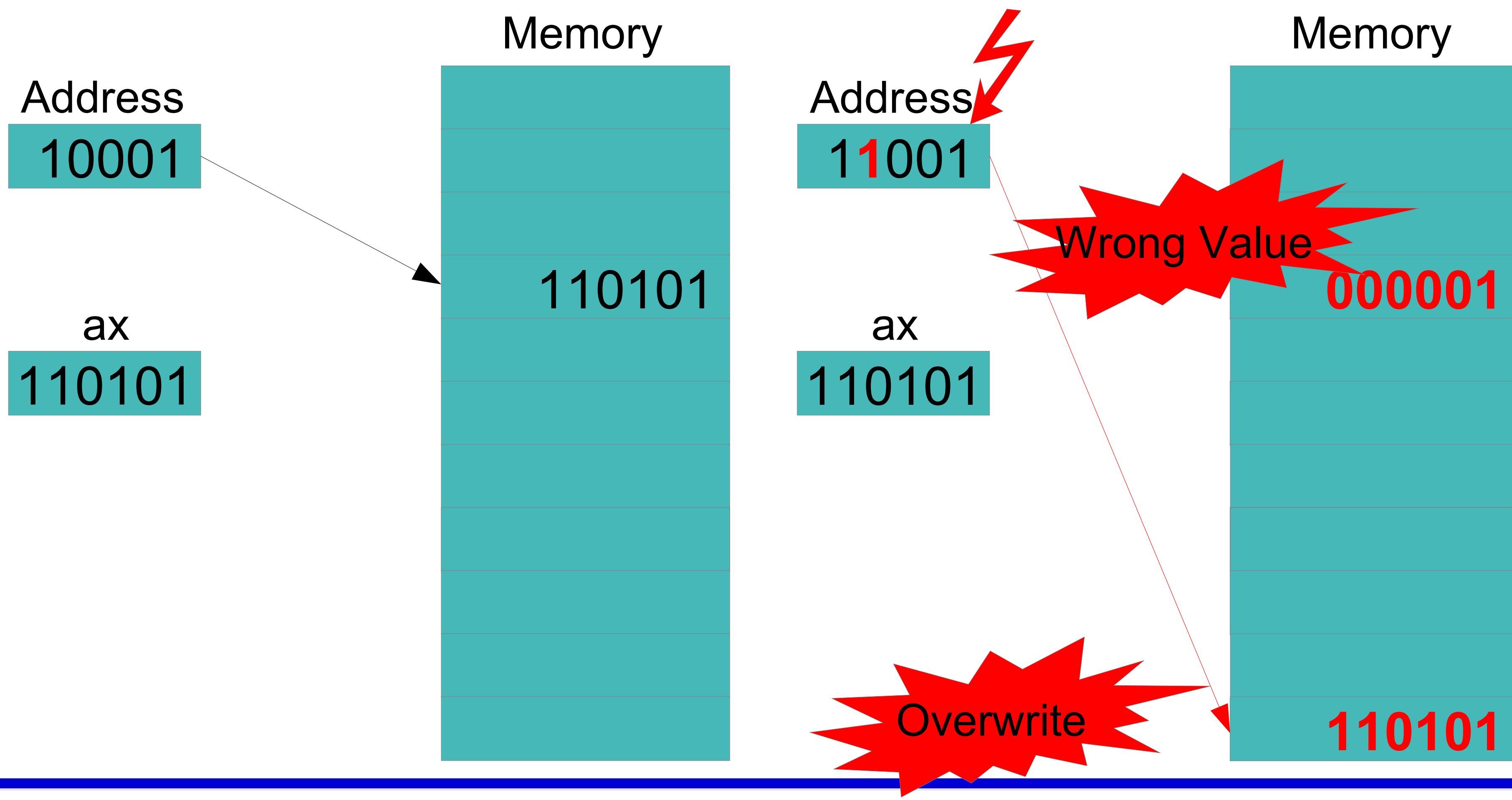


Soft-errors effect

Memory store

e.g., mov [address], ax

But if a soft-error happened

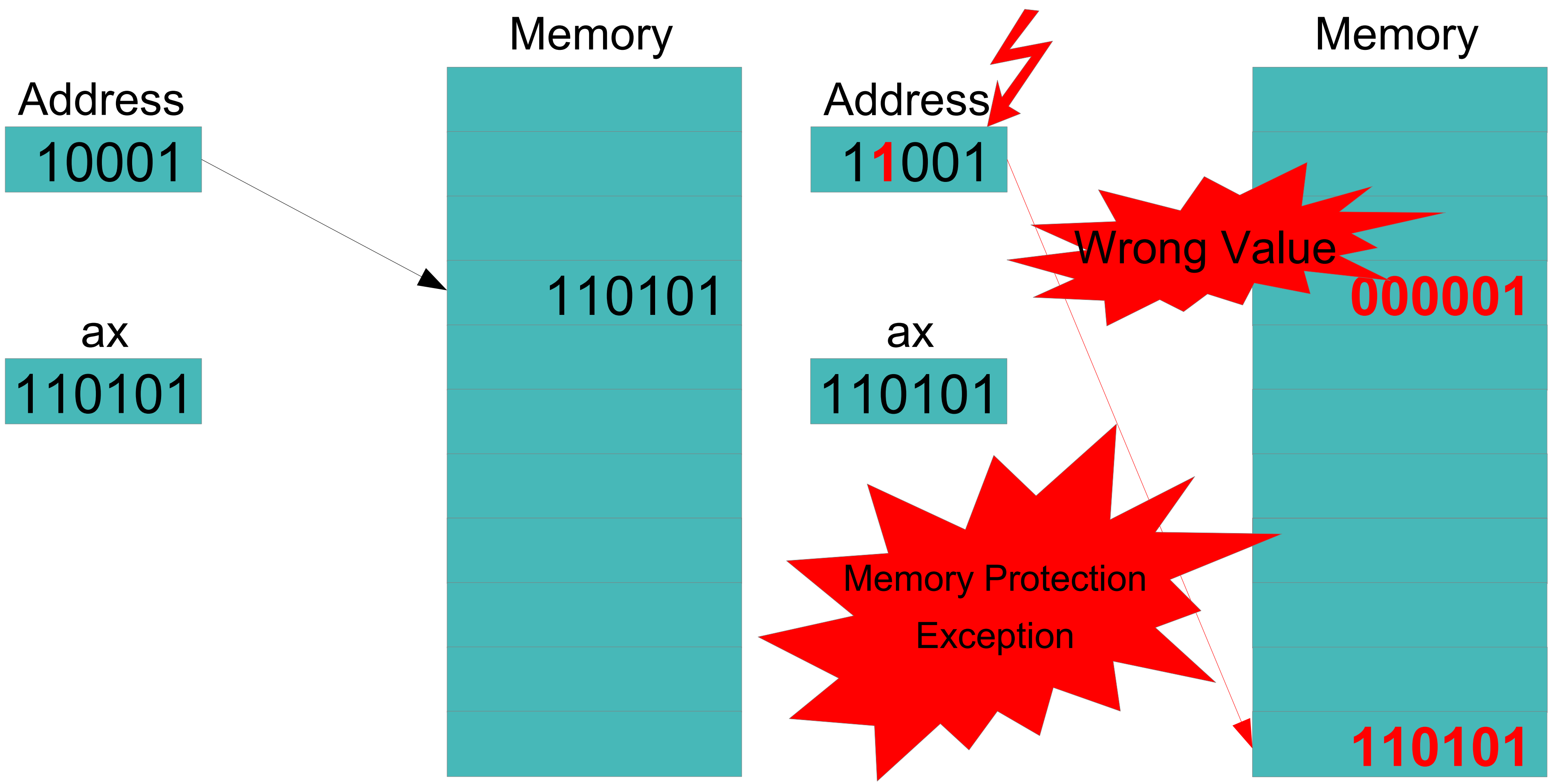


Soft-errors effect

Memory store

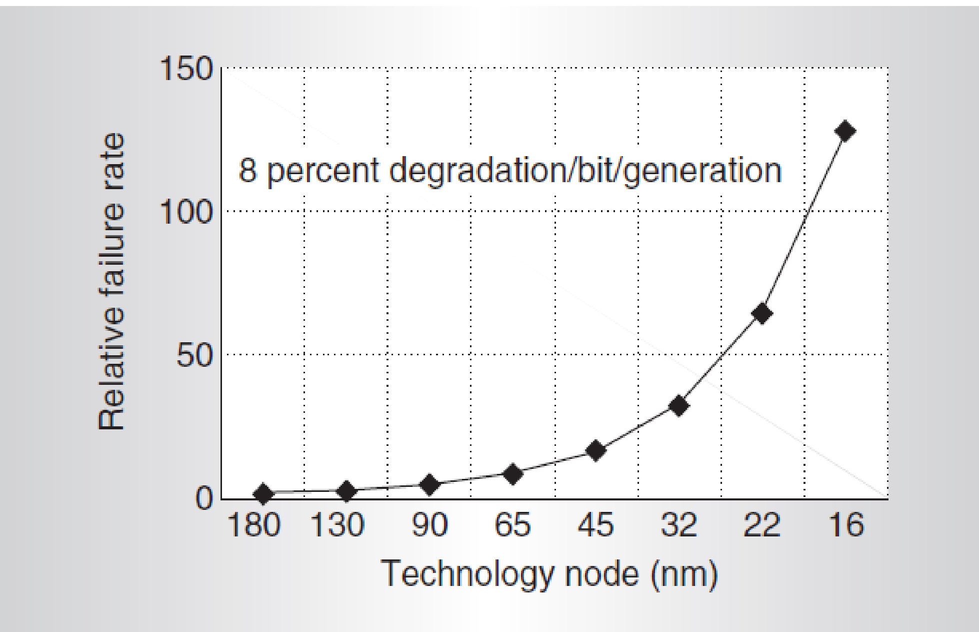
e.g., mov [address], ax

But if a soft-error happened

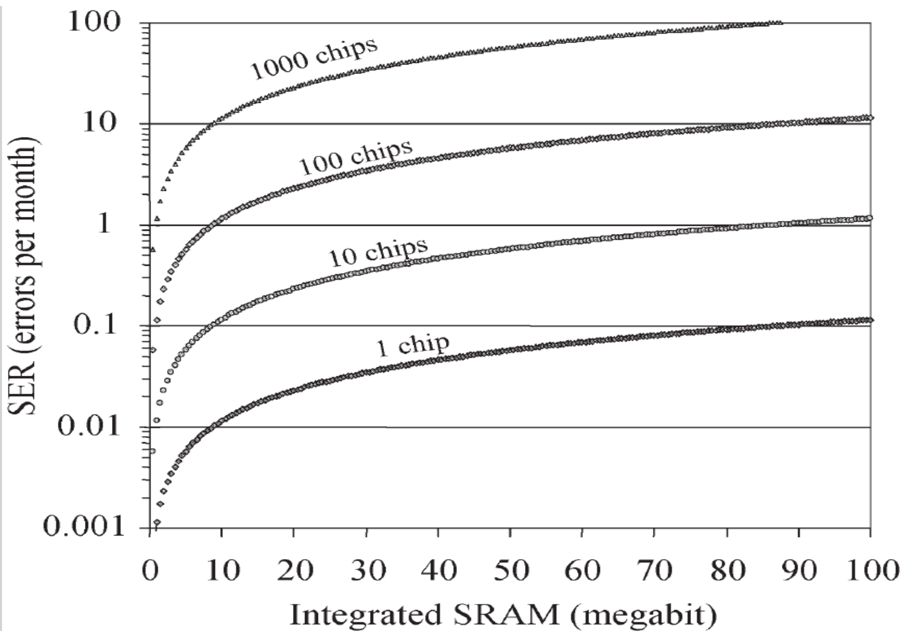


Soft-errors in multicore architectures

- Soft-errors rate is growing in the current and emerging multicore architectures
  - Smaller transistors (e.g., Intel Haswell uses 22nm)
  - More components on same chip (e.g., more cores)



Soft-error failure-in-time of a chip [1]



SER as a function of the number of chips [2]

How to tolerate soft-errors?

- Restart the application!
  - It may not crash!
  - Not suitable for critical business applications
    - We need to maintain availability/reliability constraints
- Hardware
  - High end systems
  - Expensive
- Replication
  - Multiple isolated copies of the application data
  - Fully mask faults
  - But, it is designed for distributed system

Motivation

- Apply the same distributed replication mechanisms in centralized multicore systems
- Is that enough?
  - Significantly degraded performance
  - Expensive

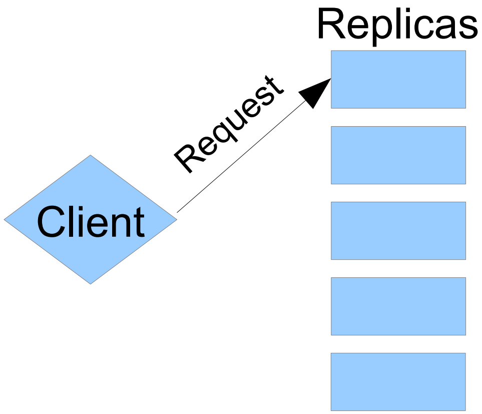
Byzantine faults

- Byzantine Faults are arbitrary faults
  - Omission faults
  - Commission faults
- Soft-errors can be categorized as Byzantine Faults
- Byzantine fault-tolerant systems are usually based on state-machine replication

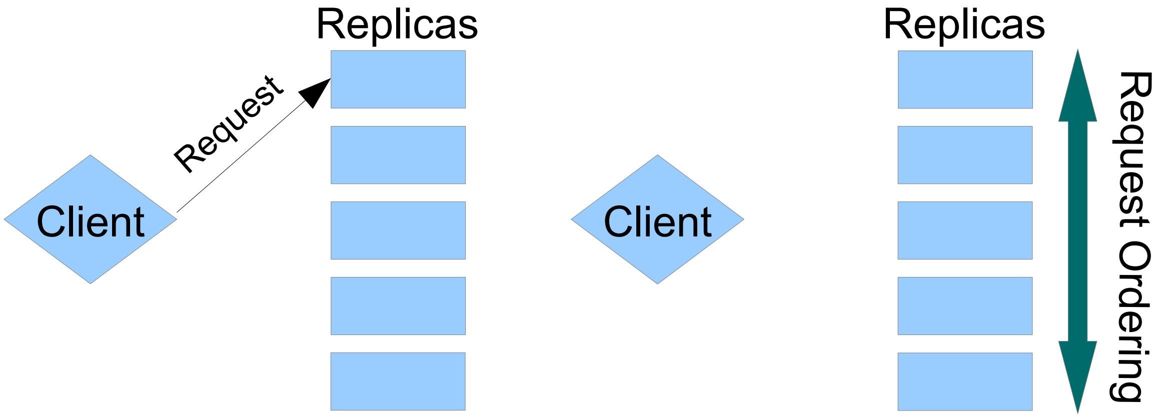
Byzantine fault-tolerant (BFT) systems

- System clients + Multiple replicas (servers)
  - Requests sent by clients are totally ordered.
  - All replicas execute the requests in the same order independently
  - Client receives a reply from each replica
  - Different reply means an error has occurred
- Require $3f+1$ replica to tolerate f faults
- Target arbitrary faults and malicious activities

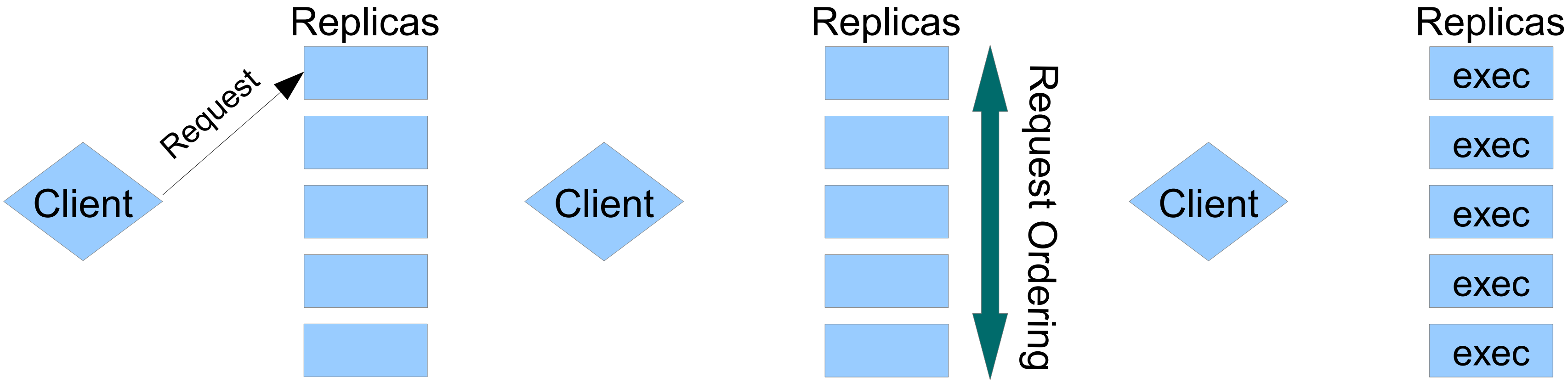
BFT Systems



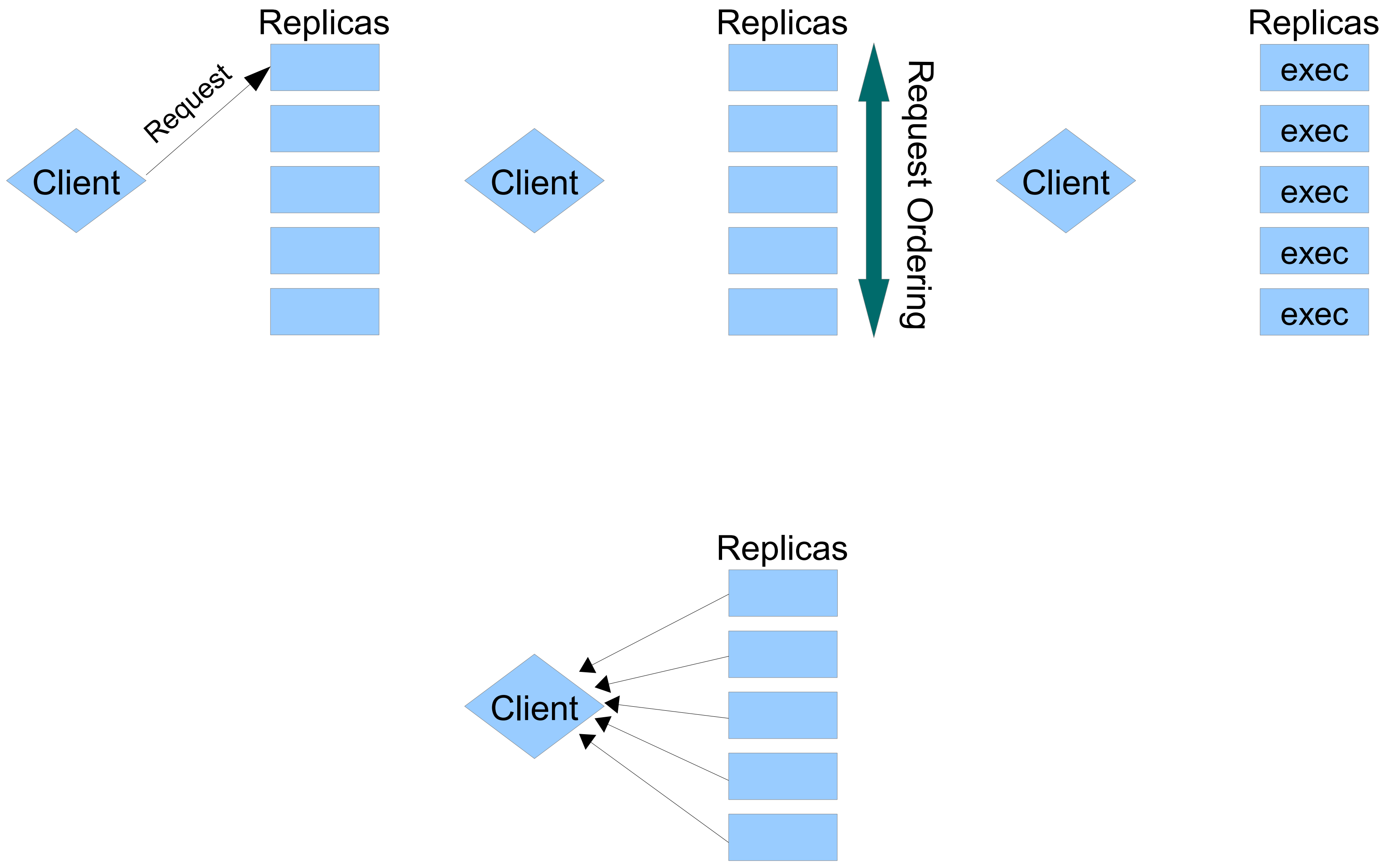
BFT Systems


Replicas
Client
Request
Replicas
Client
Request Ordering

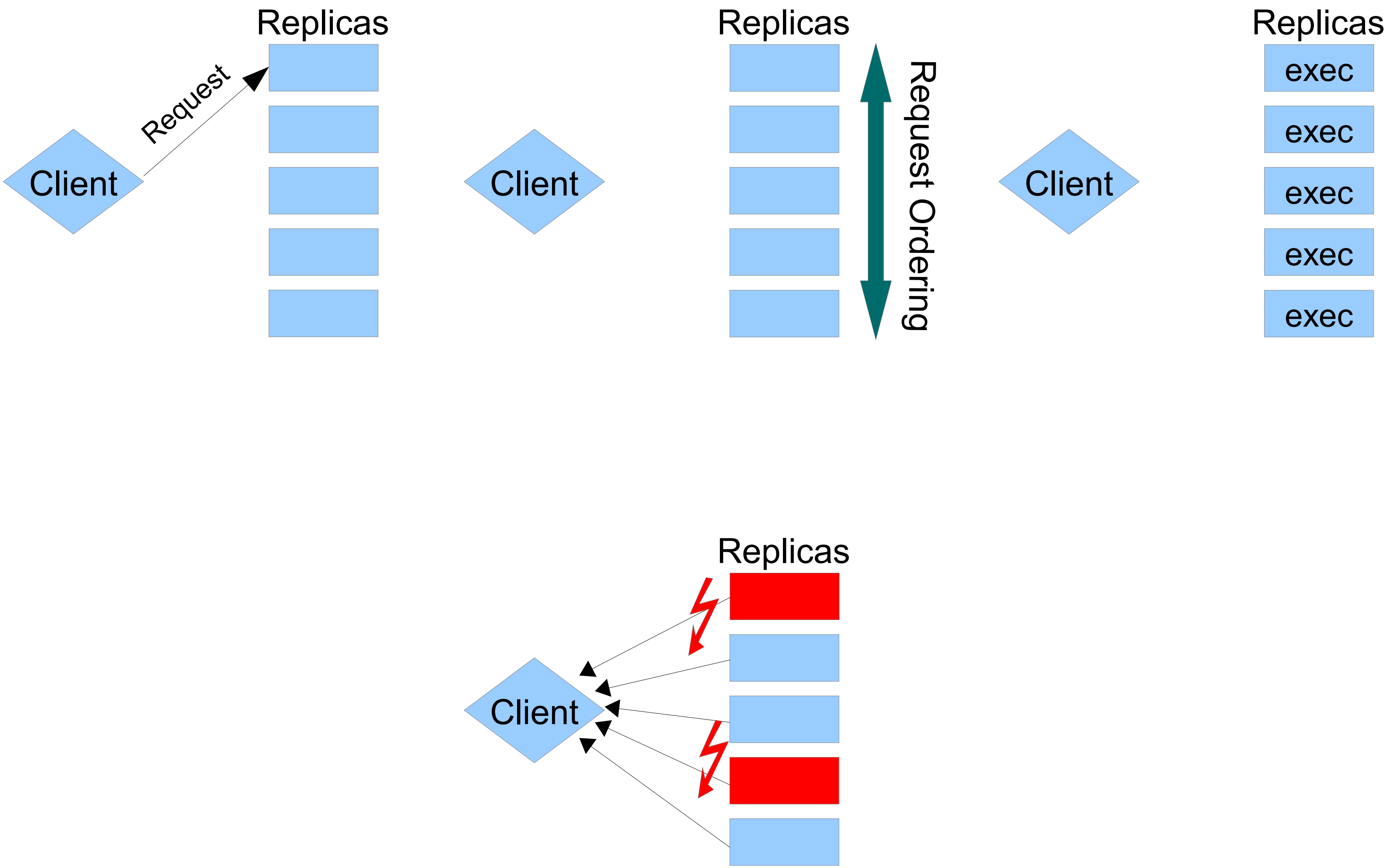
BFT Systems


Replicas
Client
Request
Replicas
Client
Request Ordering
Replicas
Client
exec
exec
exec
exec
exec

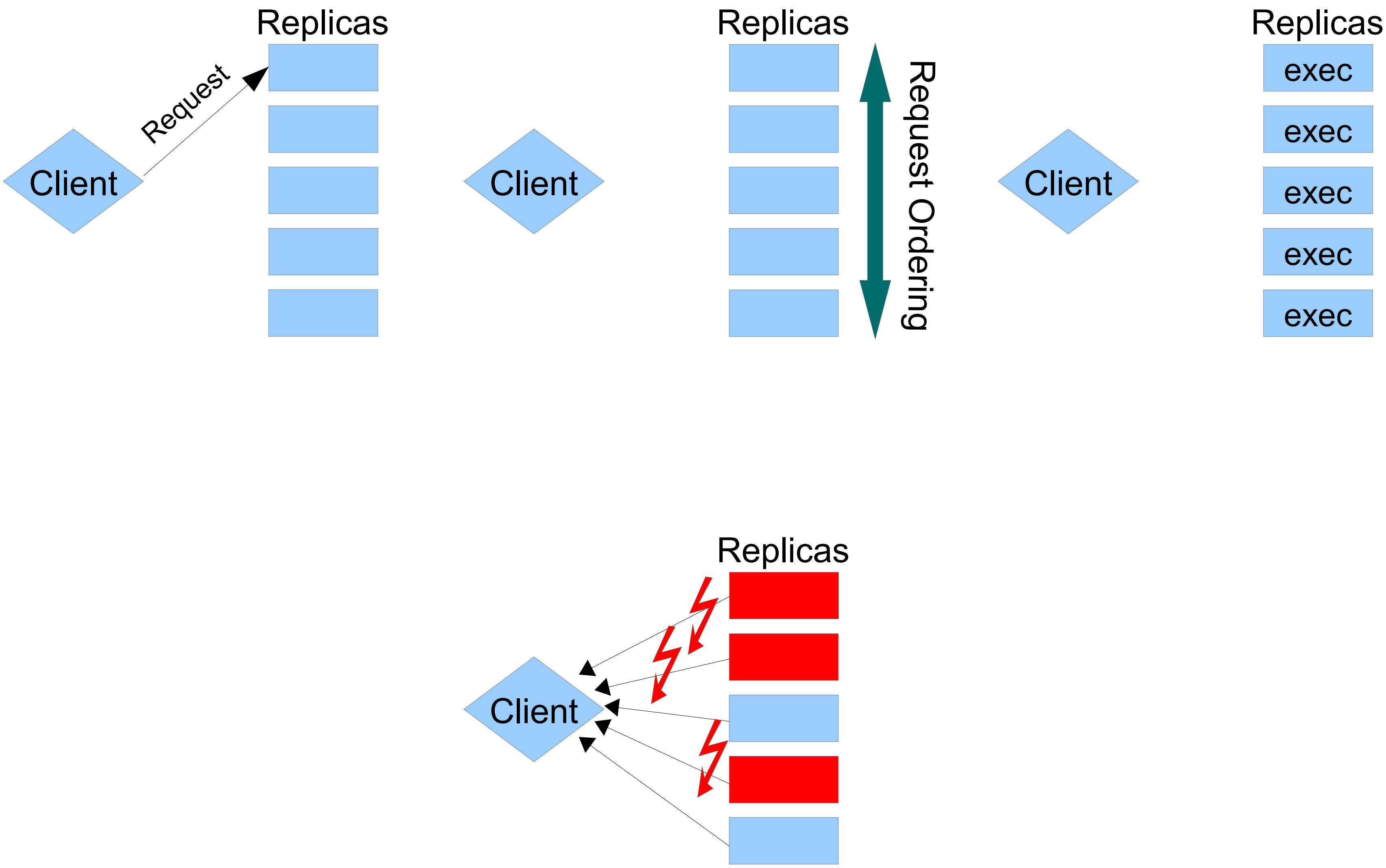
BFT Systems


Replicas
Request
Client
Replicas
Request Ordering
Client
Replicas
exec
exec
exec
exec
exec
Client
Replicas
Client

BFT Systems


Replicas
Request
Client
Replicas
Request Ordering
Client
Replicas
exec
exec
exec
exec
exec
Client
Replicas
Client

BFT Systems



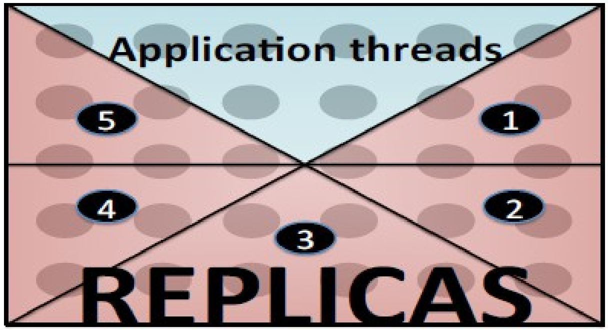
Replicas
Request
Client
Replicas
Request Ordering
Client
Replicas
exec
exec
exec
exec
exec
Client
Replicas
Client

Proposed Solution

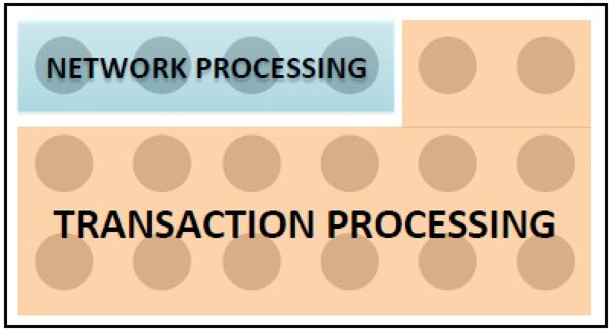
- A state-machine replication-based system customized for centralize systems
  - An Optimized network protocol
    - Decentralized
    - Supports optimistic delivery
  - An innovative concurrency control algorithm
    - Allows concurrent requests execution using STM
    - Preserves a predefined commit order

Proposed Solution

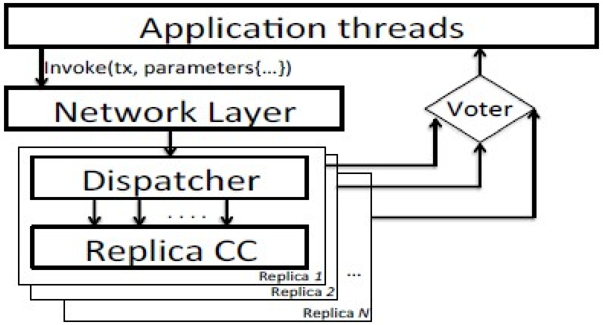
- Partition available resources into replicas and application threads
- ObCC: Ordering-based Concurrency Control.
- Replicas immediately optimistically deliver request
  - Replicas: Start total ordering phase
  - ObCC: Execute request speculatively using STM



(a) System



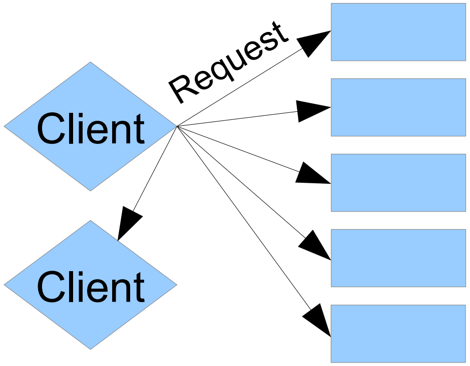
(b) Replica



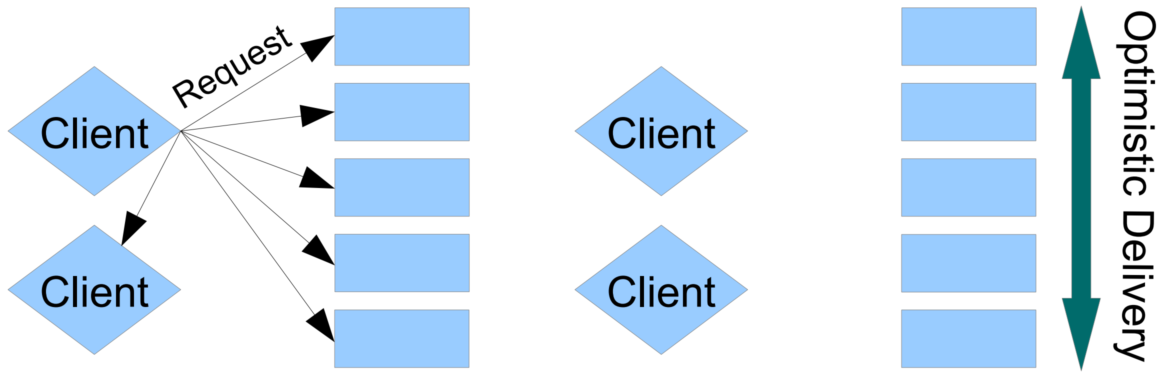
(c) Software

Network Layer

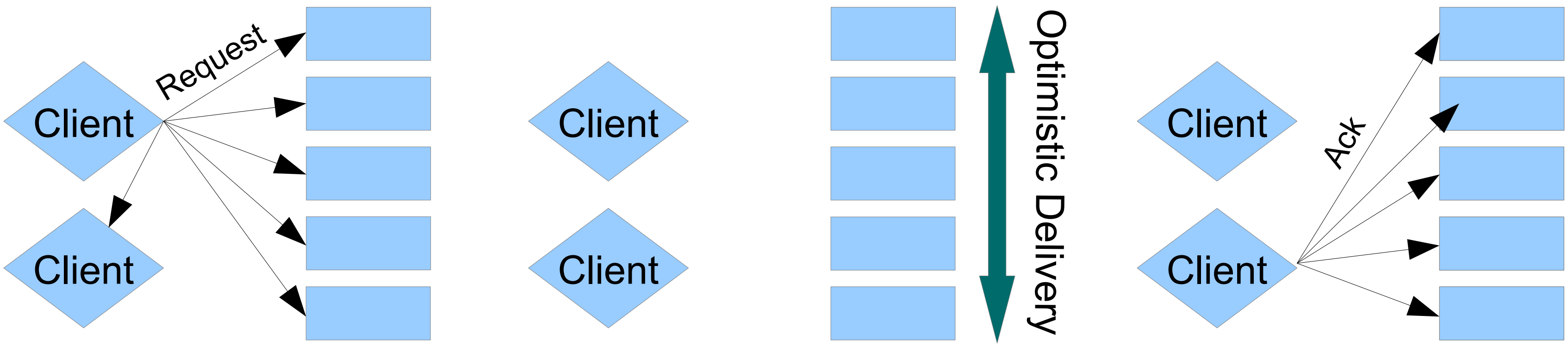
Network Layer



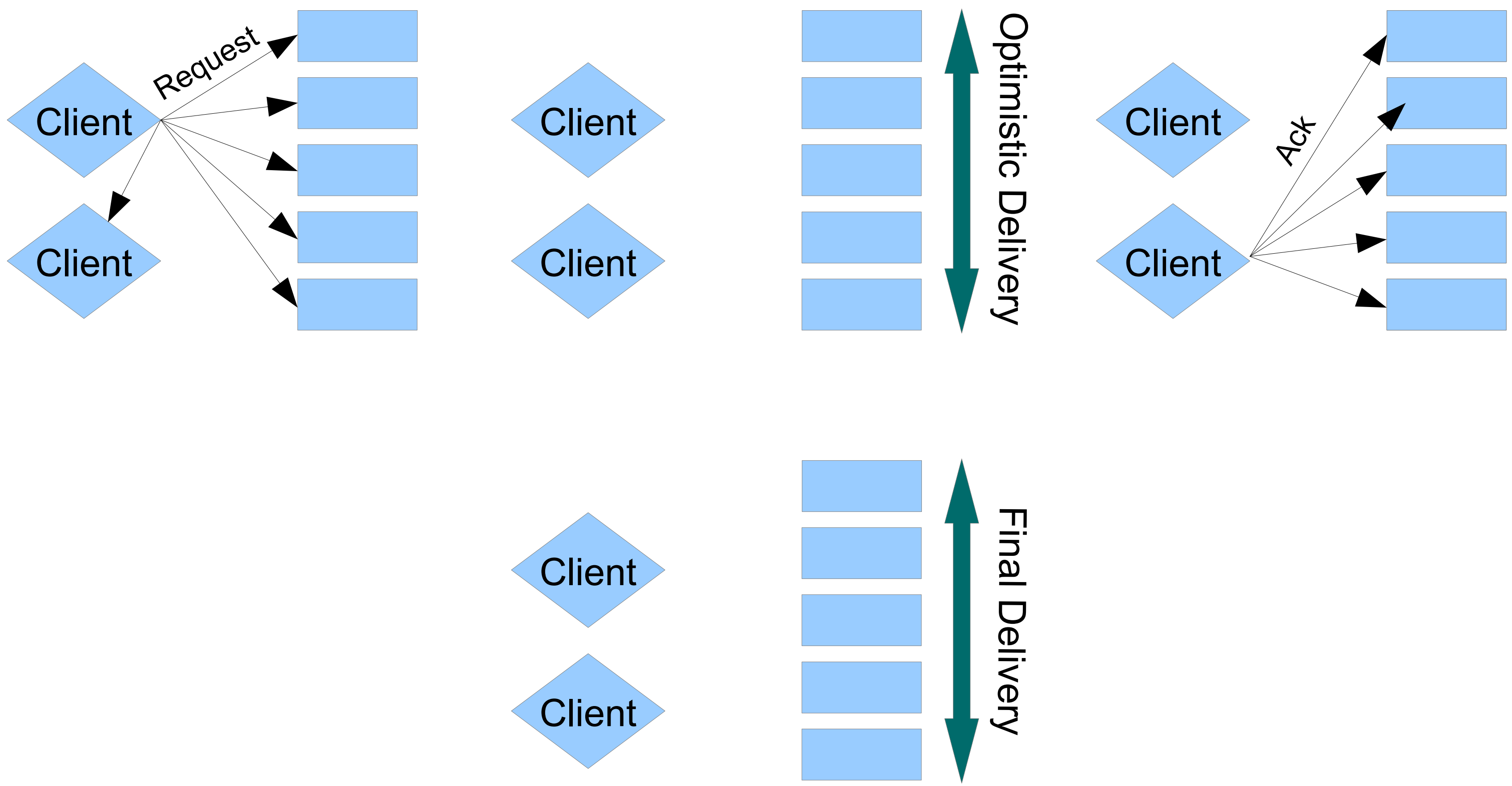
Network Layer



Network Layer



Network Layer



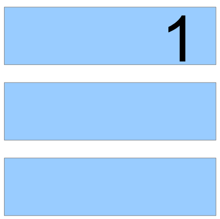
Network Layer

- Decentralized Ordering
- Assumptions
  - Reliable Network
  - Thread FIFO: thread requests are received in the same order
  - Synchronized Clock

Network Layer Decentralized Ordering

Replicas have a queue for each client

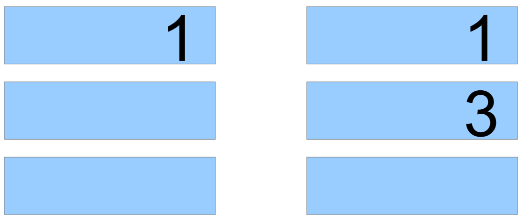
Example on a system with 3 clients



Network Layer Decentralized Ordering

Replicas have a queue for each client

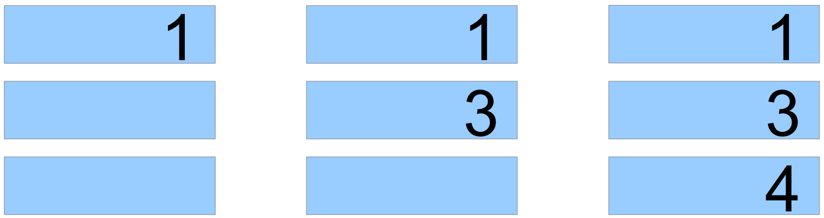
Example on a system with 3 clients



Network Layer Decentralized Ordering

Replicas have a queue for each client

Example on a system with 3 clients



Network Layer Decentralized Ordering

Replicas have a queue for each client

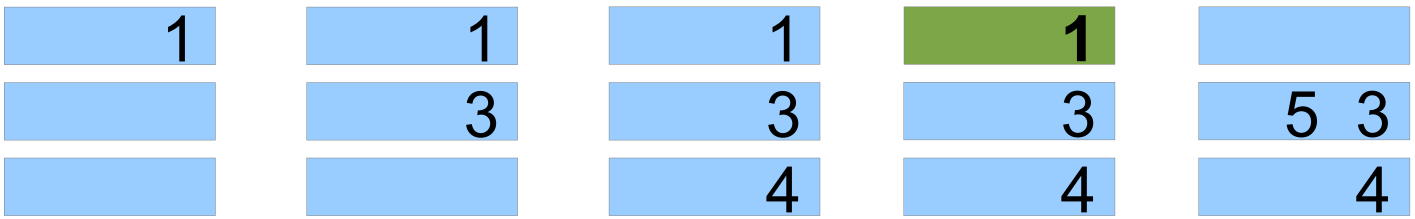
Example on a system with 3 clients

1	1	1	**1**
	3	3	3
		4	4

Network Layer Decentralized Ordering

Replicas have a queue for each client

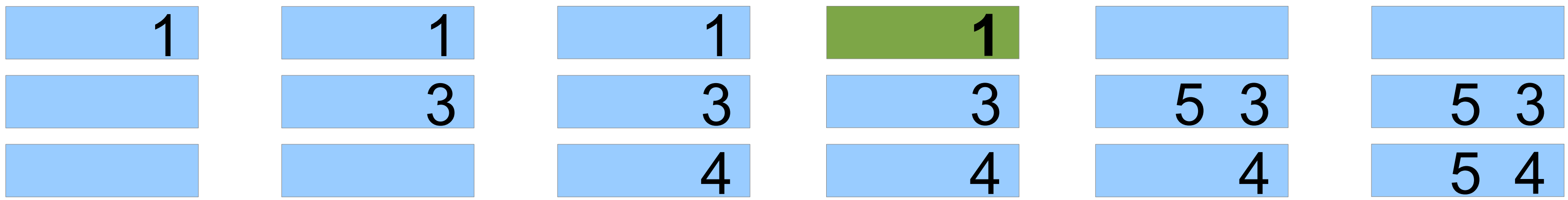
Example on a system with 3 clients



Network Layer Decentralized Ordering

Replicas have a queue for each client

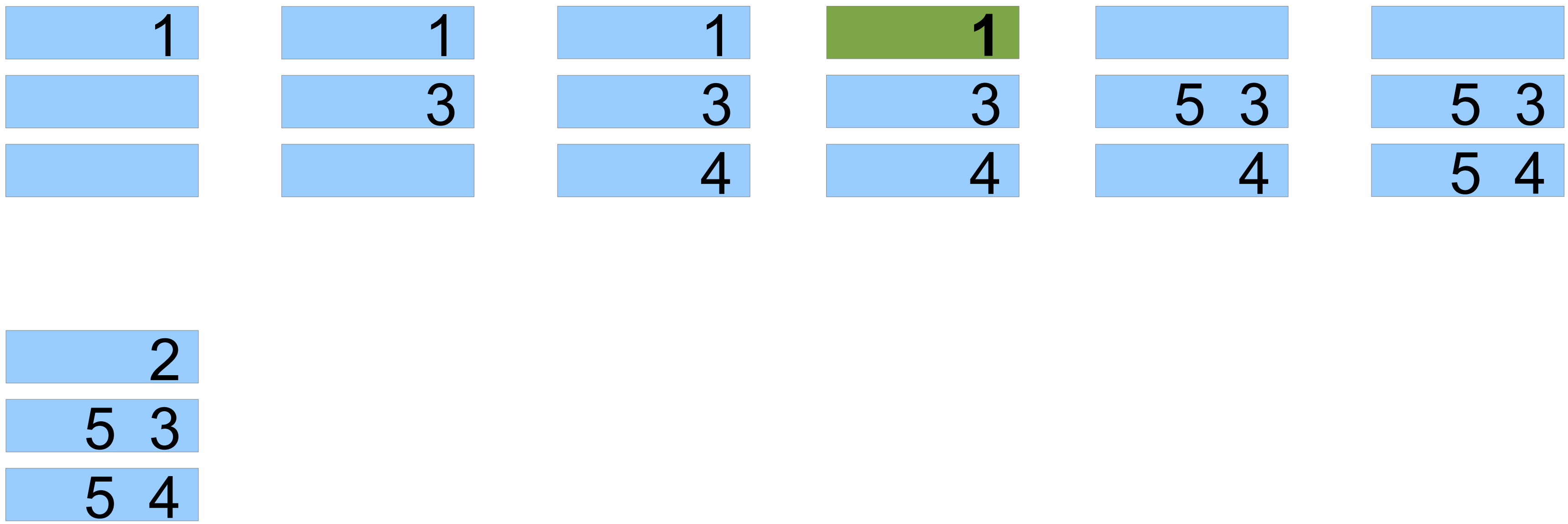
Example on a system with 3 clients



Network Layer Decentralized Ordering

Replicas have a queue for each client

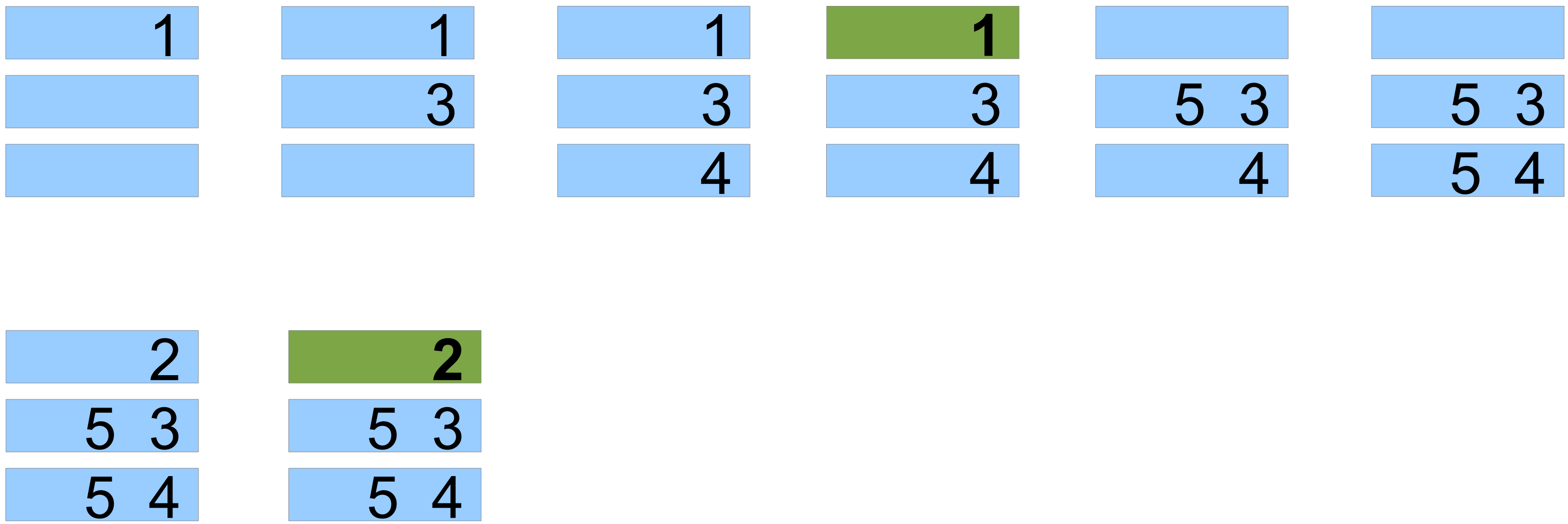
Example on a system with 3 clients



Network Layer Decentralized Ordering

Replicas have a queue for each client

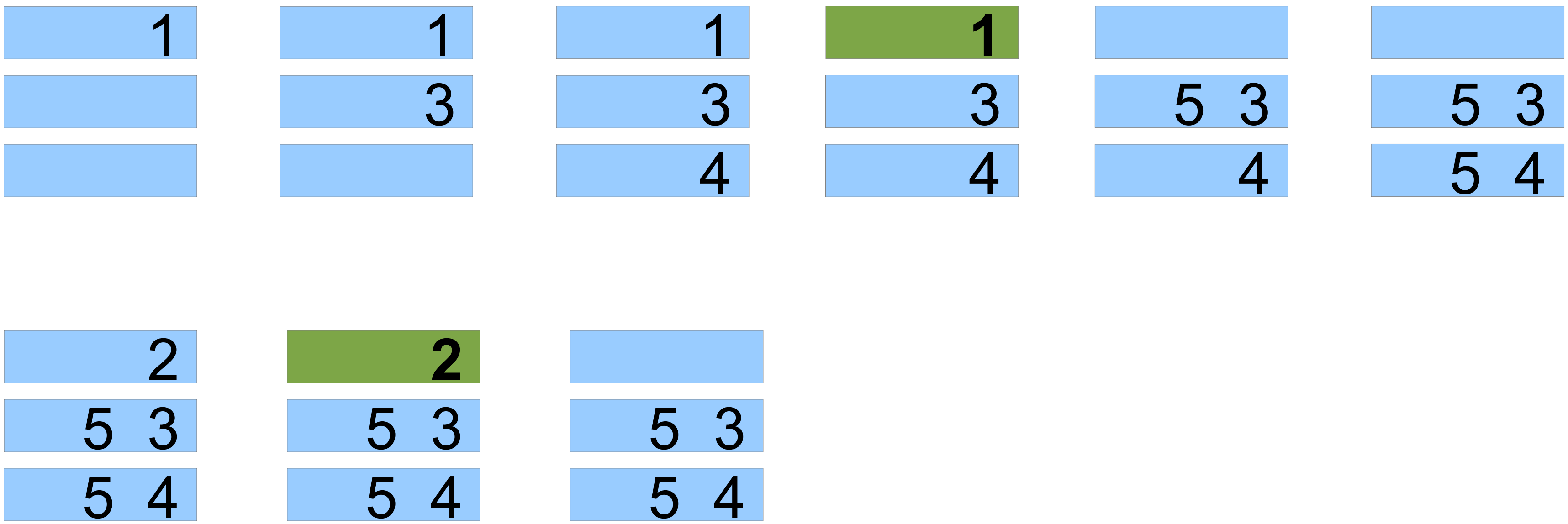
Example on a system with 3 clients



Network Layer Decentralized Ordering

Replicas have a queue for each client

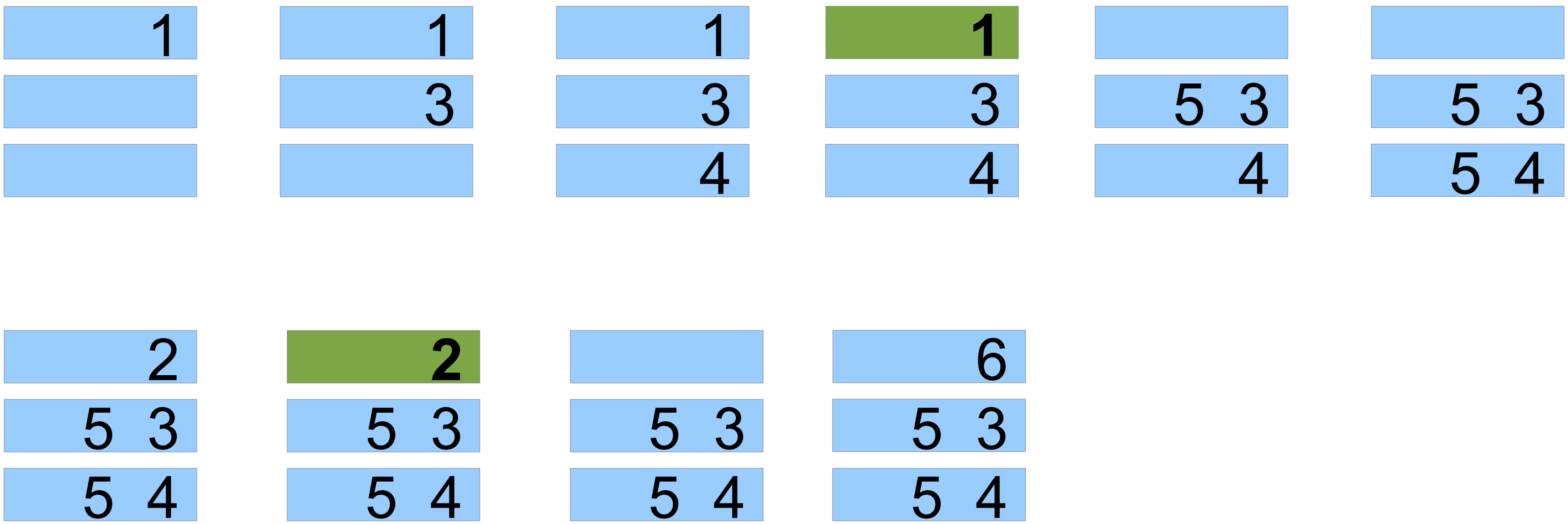
Example on a system with 3 clients



Network Layer Decentralized Ordering

Replicas have a queue for each client

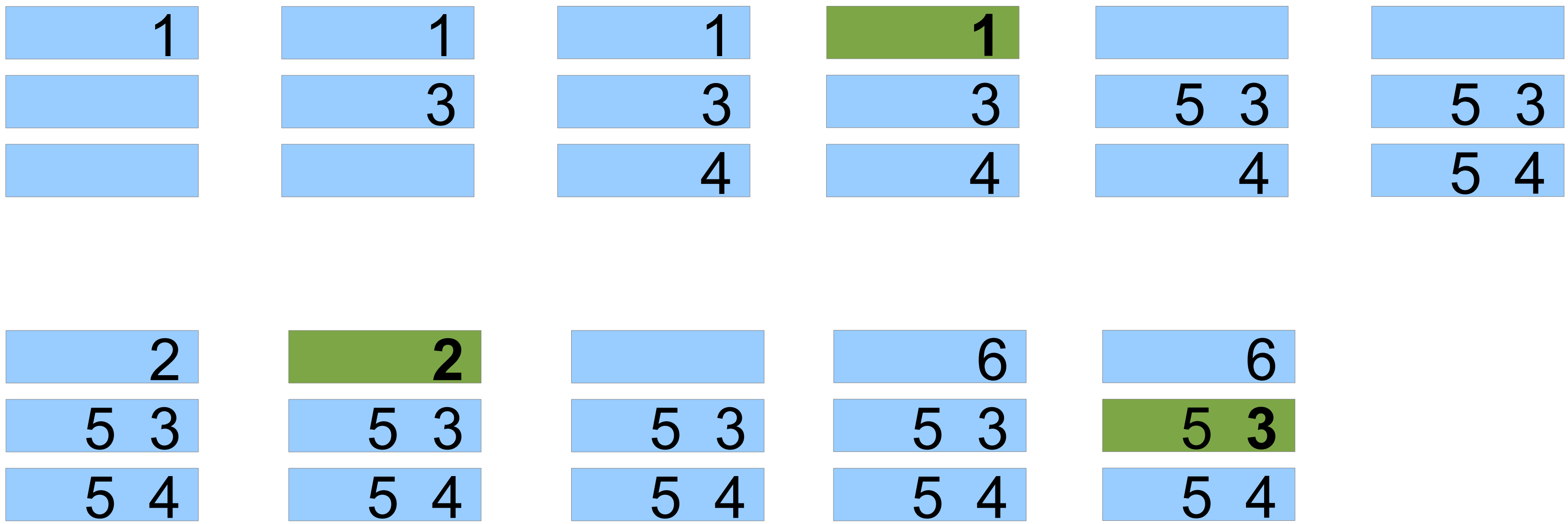
Example on a system with 3 clients



Network Layer Decentralized Ordering

Replicas have a queue for each client

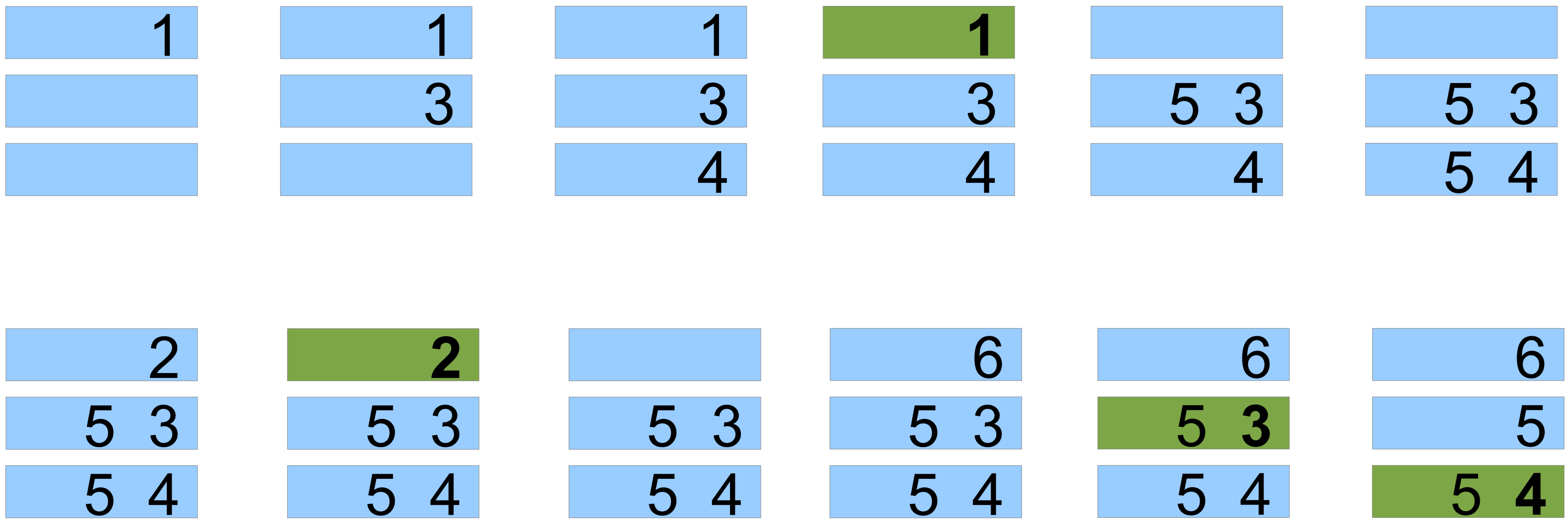
Example on a system with 3 clients



Network Layer Decentralized Ordering

Replicas have a queue for each client

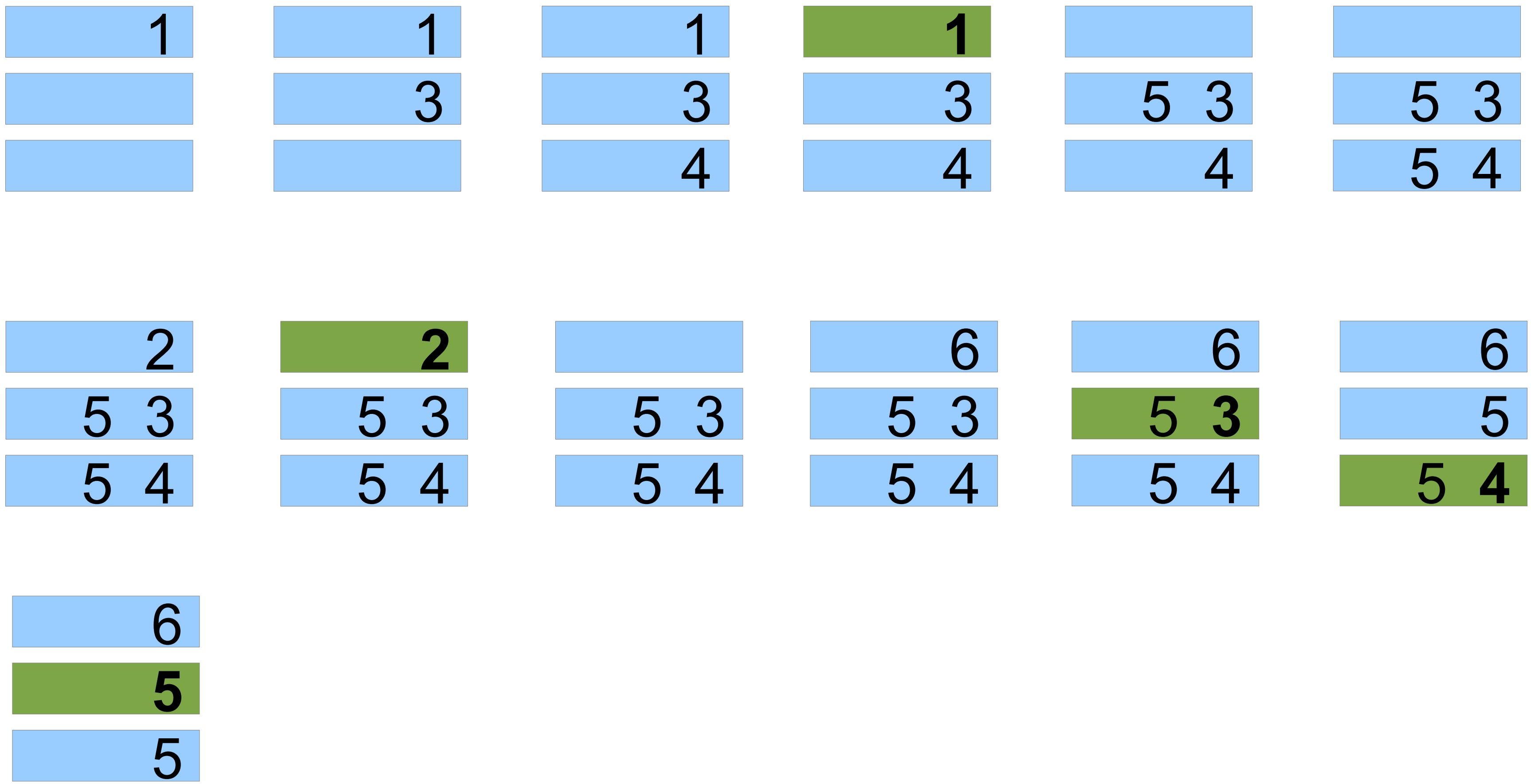
Example on a system with 3 clients



Network Layer Decentralized Ordering

Replicas have a queue for each client

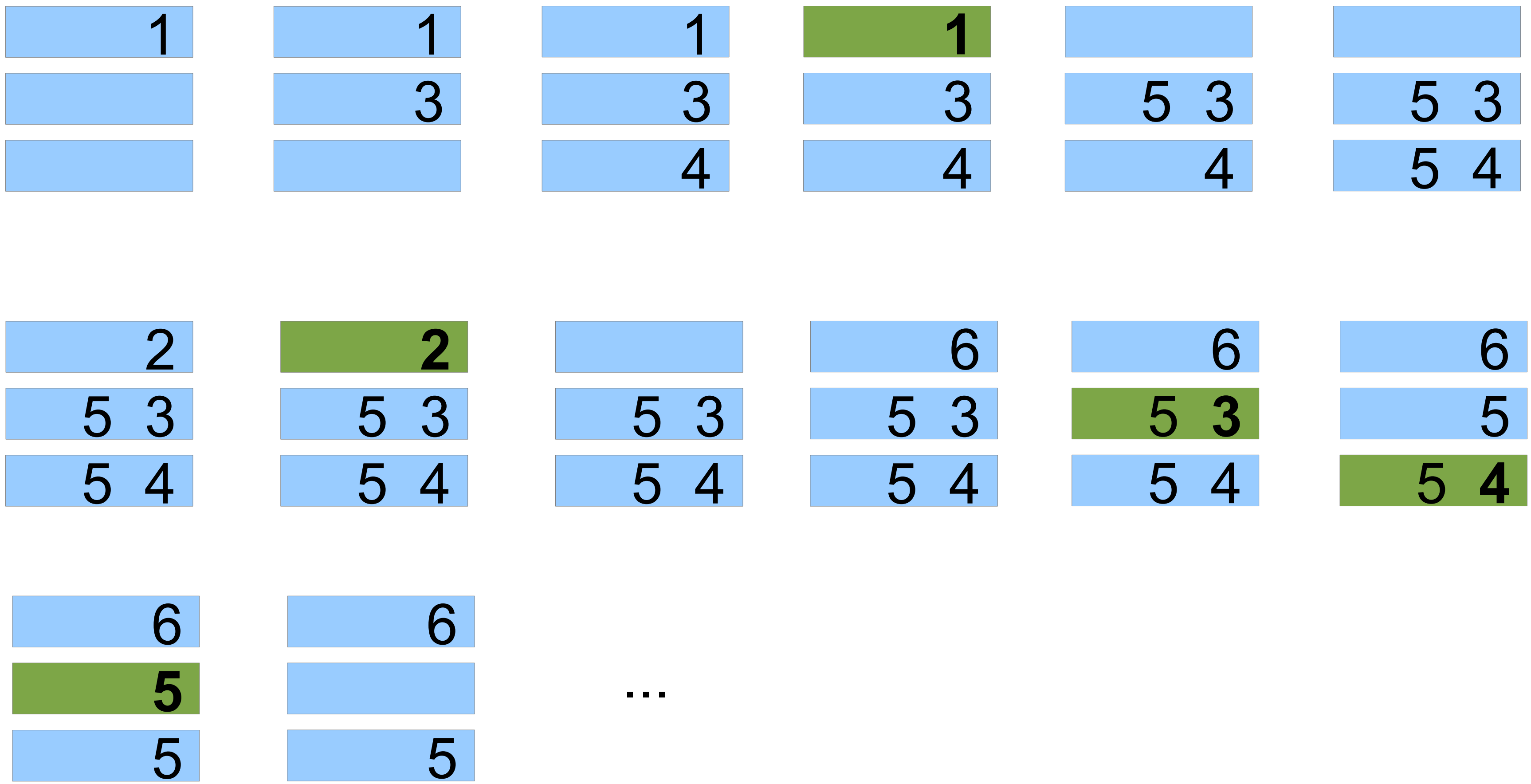
Example on a system with 3 clients



Network Layer Decentralized Ordering

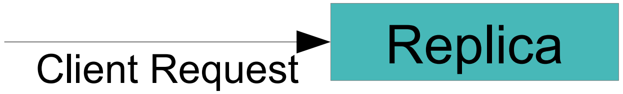
Replicas have a queue for each client

Example on a system with 3 clients

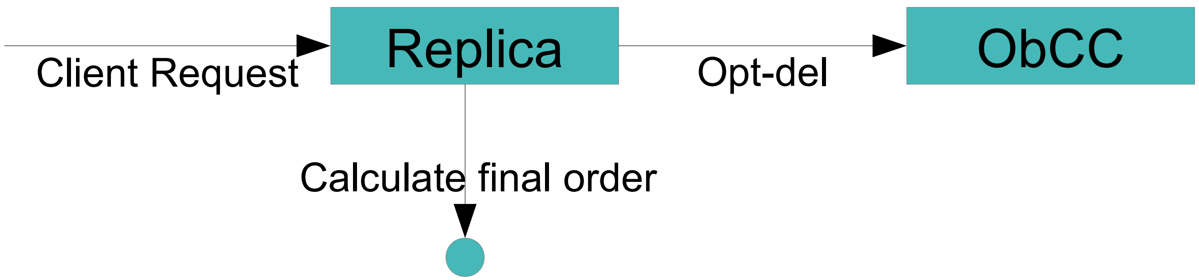


Concurrency Control

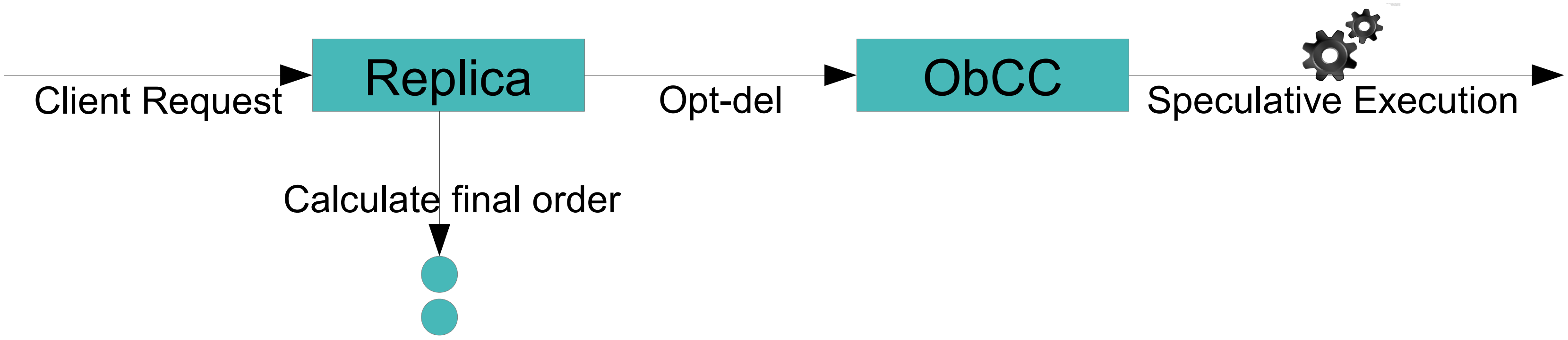
Concurrency Control



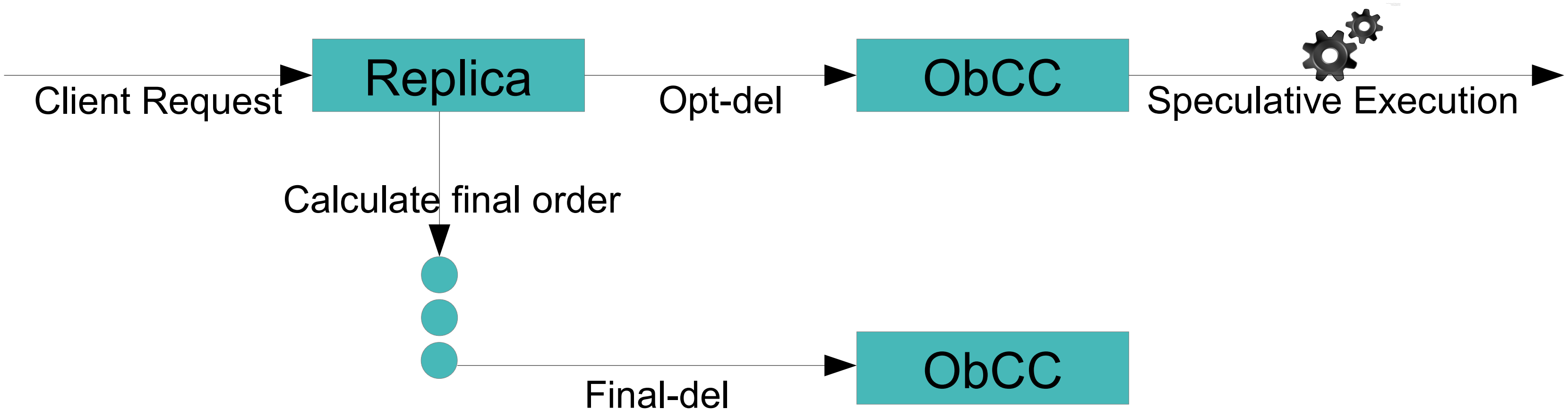
Concurrency Control



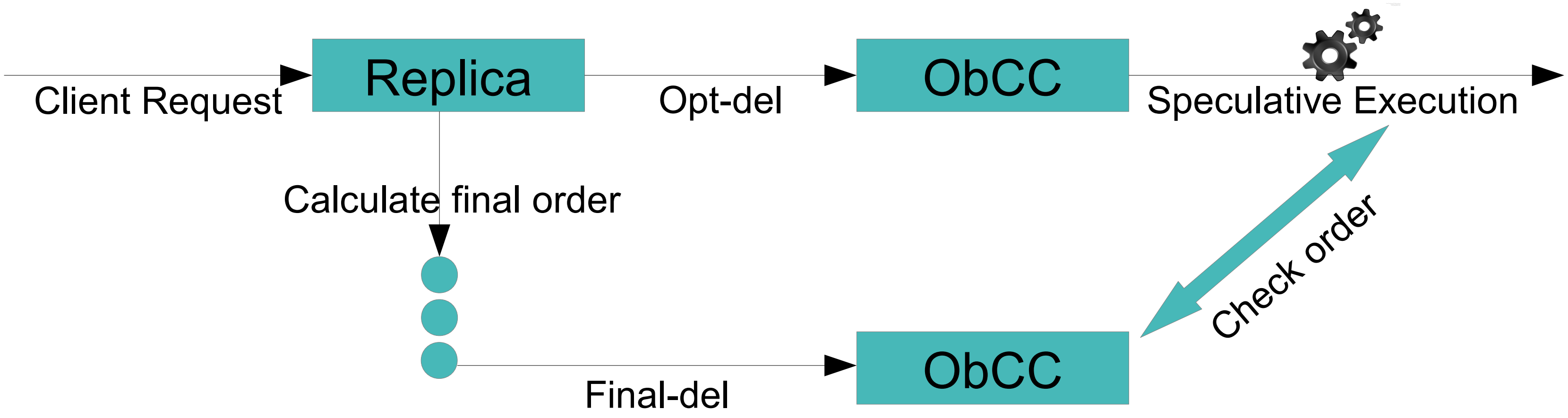
Concurrency Control



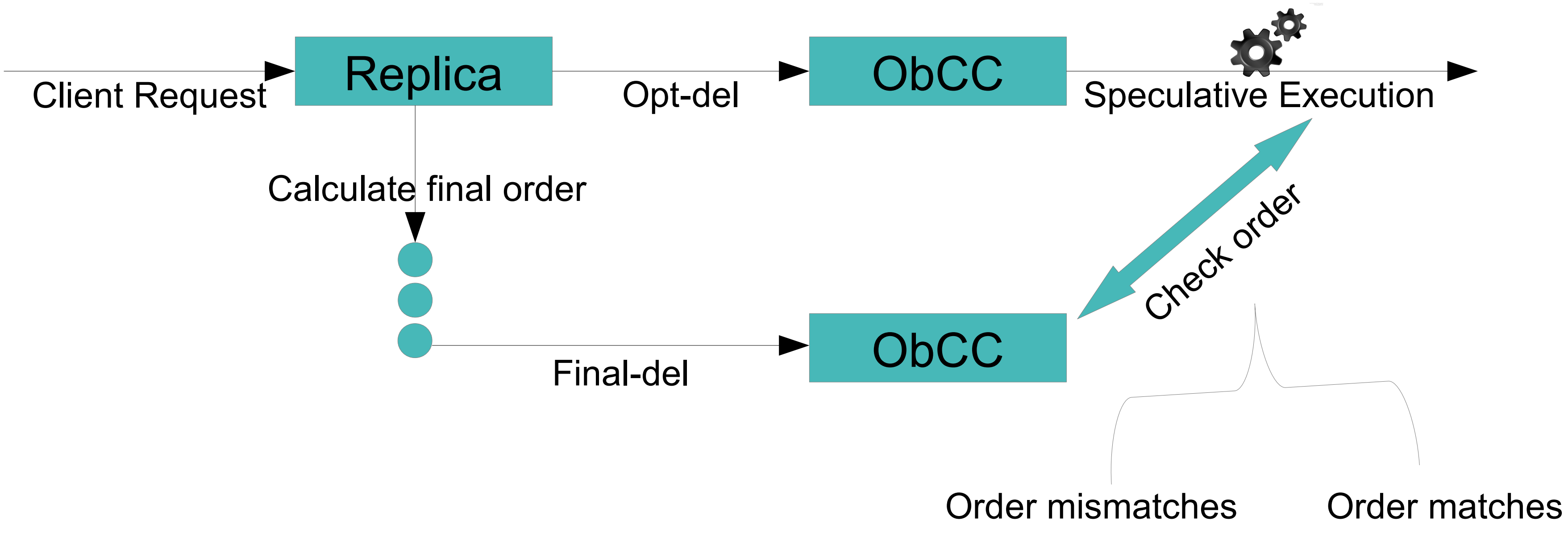
Concurrency Control



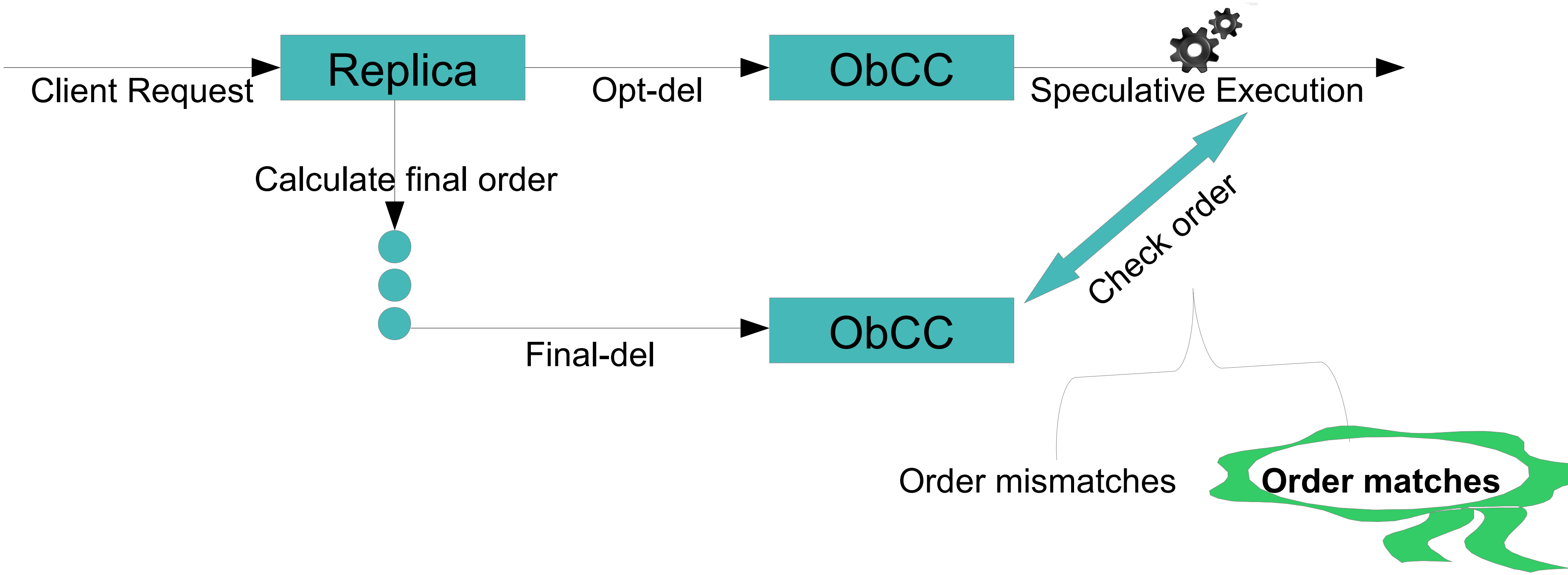
Concurrency Control



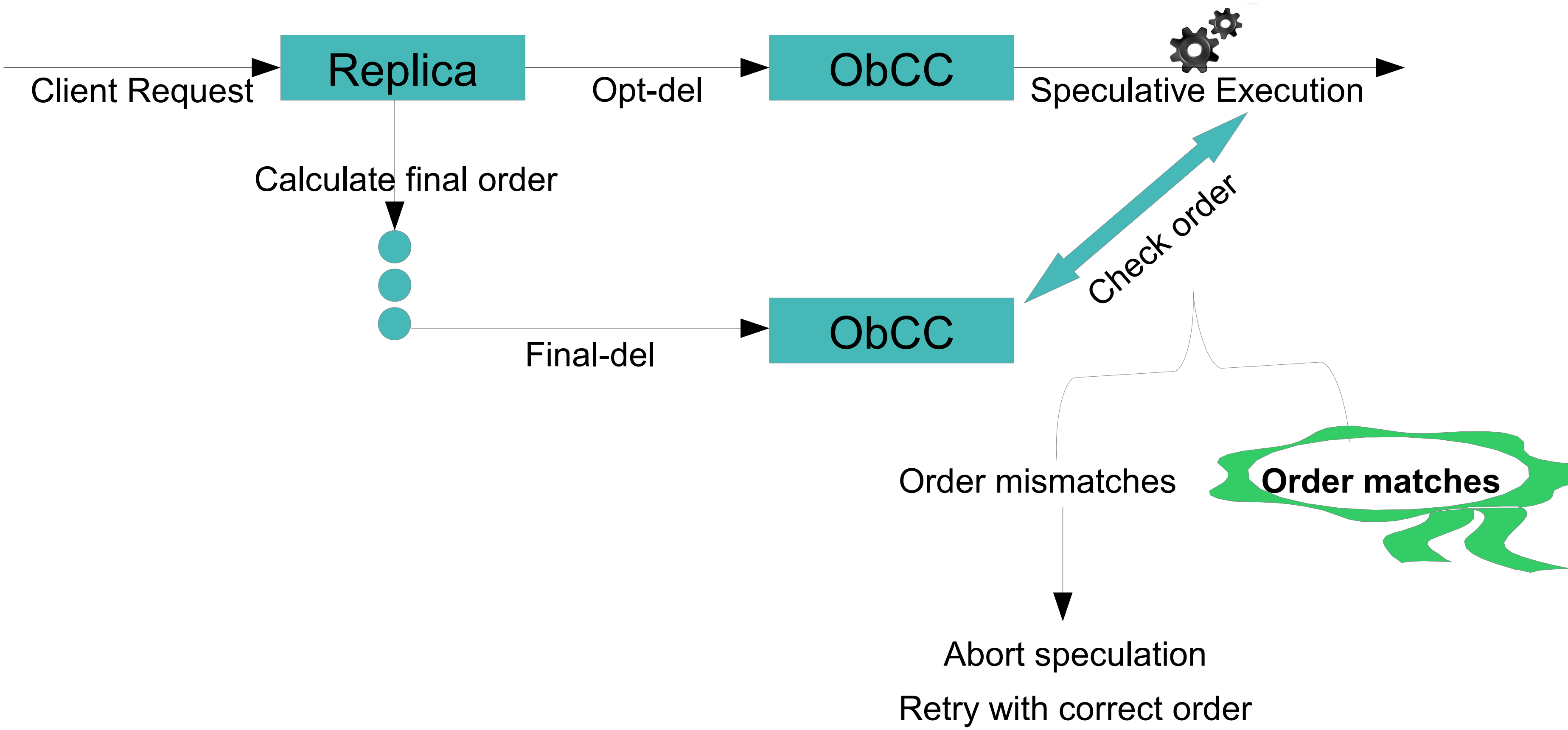
Concurrency Control



Concurrency Control



Concurrency Control



Concurrency Control

- More concurrency
  - Run multiple requests concurrently
    - Conflicts?
    - Order?

Concurrency Control

- Conflict detection and resolution
  - Two threads accessing same object and one access is write
    - Resolution: Thread that precedes wins
- Uses encounter time write-locks
  - Writing to a locked object
    - Conflict
  - Reading locked object
    - Conflict?

Concurrency Control

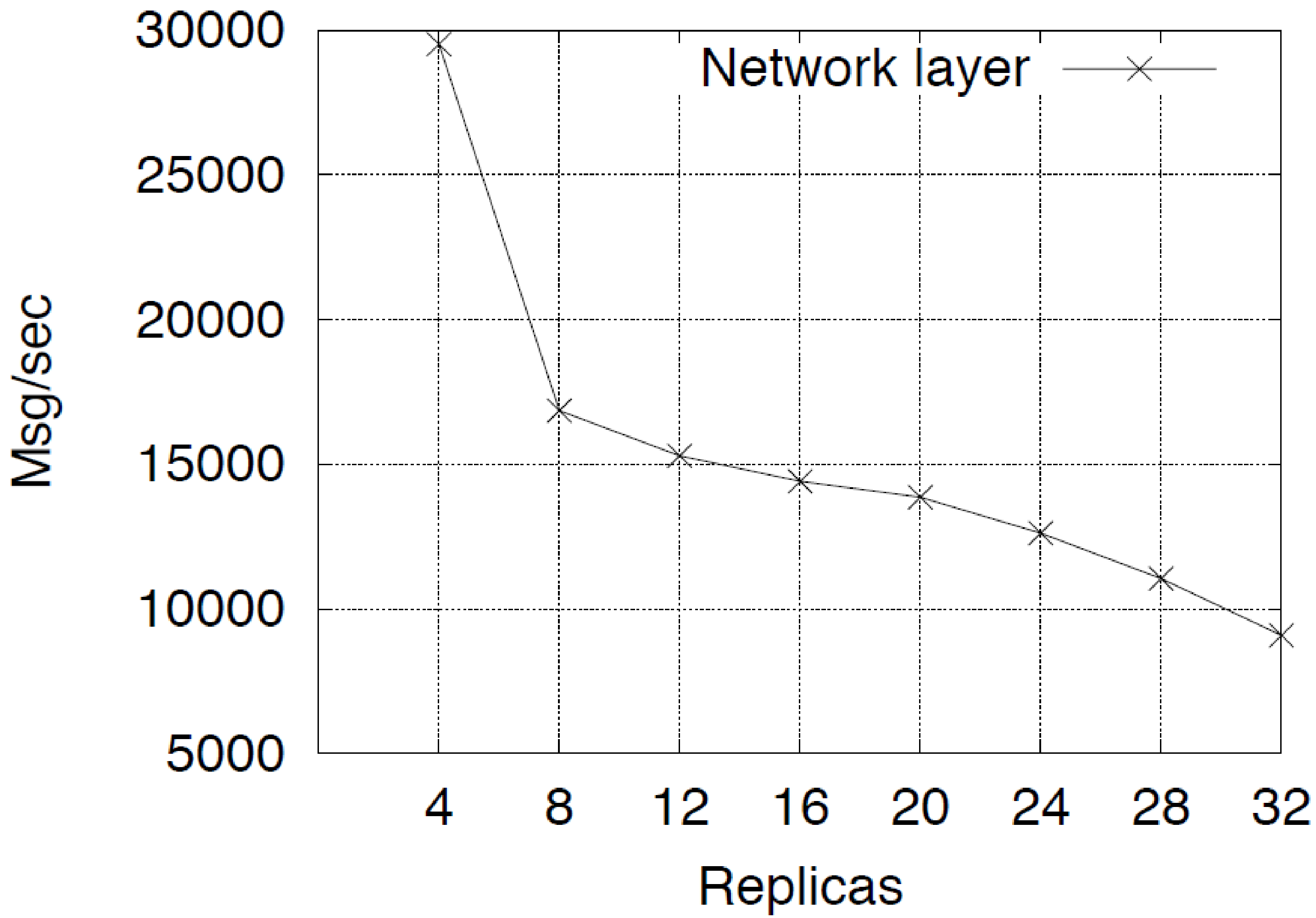
- Another enhancement: Committer mode
  - Minimal instrumentation/overhead
  - Guaranteed to commit

Evaluation

- System is implemented in C++
- Concurrency control implemented on top of RSTM [17]
- Testbed: 36-core Tilera TILEGx cooprocessor
  - 1.0 GHz clock speed
  - 8 GB DDR3 memory
  - Message-passing (iMesh 2D on-chip network)

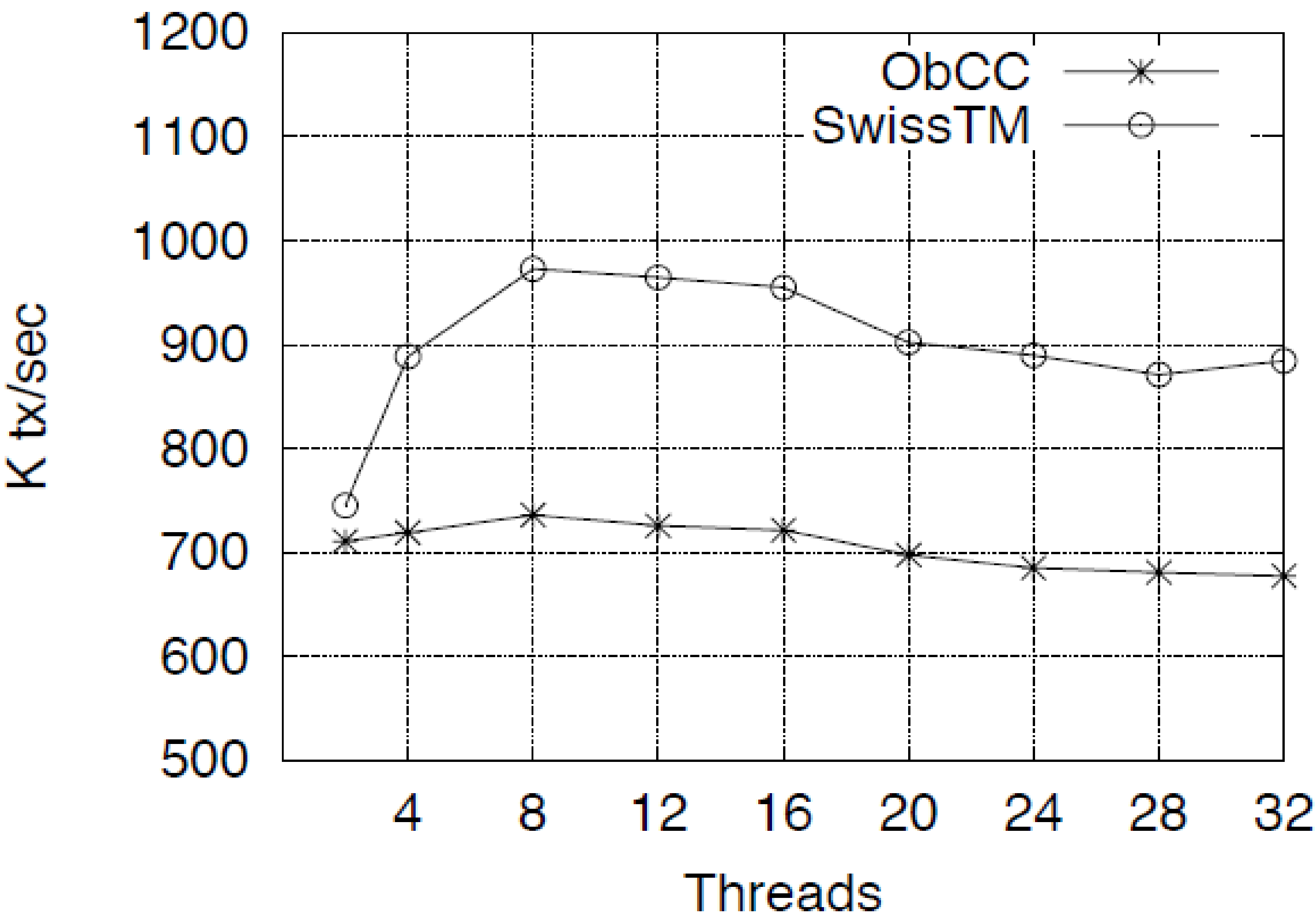
Evaluation: Network Layer

- Good performance for small number of replicas (4-8)



Evaluation: Concurrency Control

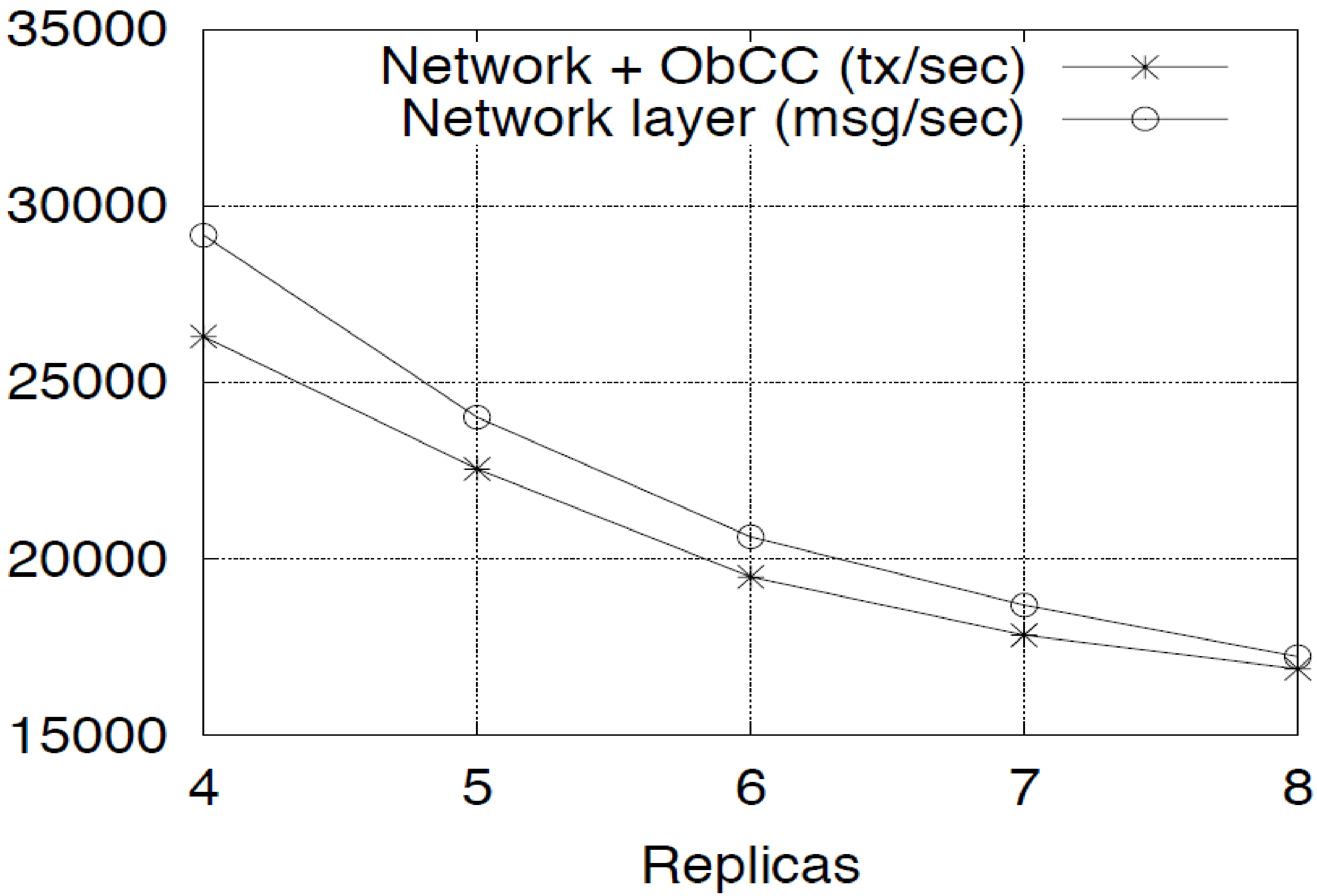
- Overhead of ordered commit is about 25%



Hash-Set

Evaluation: Integration

- System performance is bound by network performance
- Limited gap



Hash-Set

Conclusion

- Active replication is a good candidate for solving soft-errors
  - Fully masks errors
  - Reasonable overhead

- Future Work:
  - Optimizing System components
    - Reducing network layer overhead
    - Increase requests execution concurrency
  - Trying different architectures
    - Message-passing vs. shared-bus

Questions

Thank you!