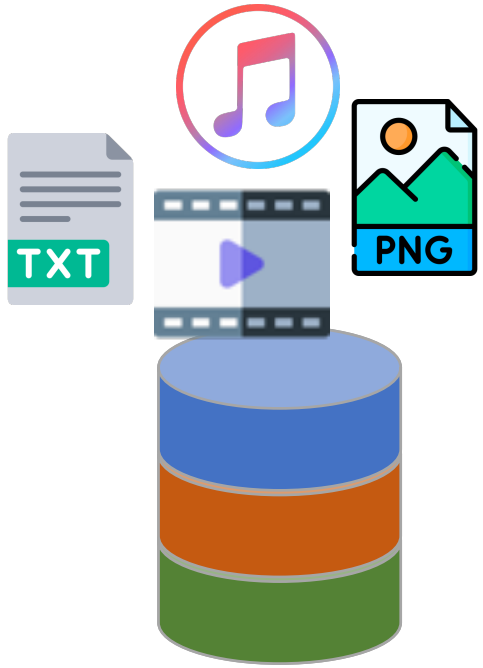
Hydra:
Serialization-Free Network Ordering for Strongly Consistent Distributed Applications

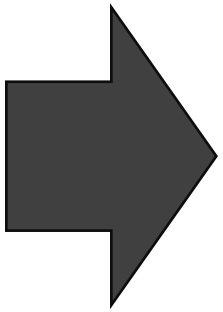
Inho Choi[1], Ellis Michael[2], Yunfan Li[1], Dan R. K. Ports[3], and Jialin Li[1]

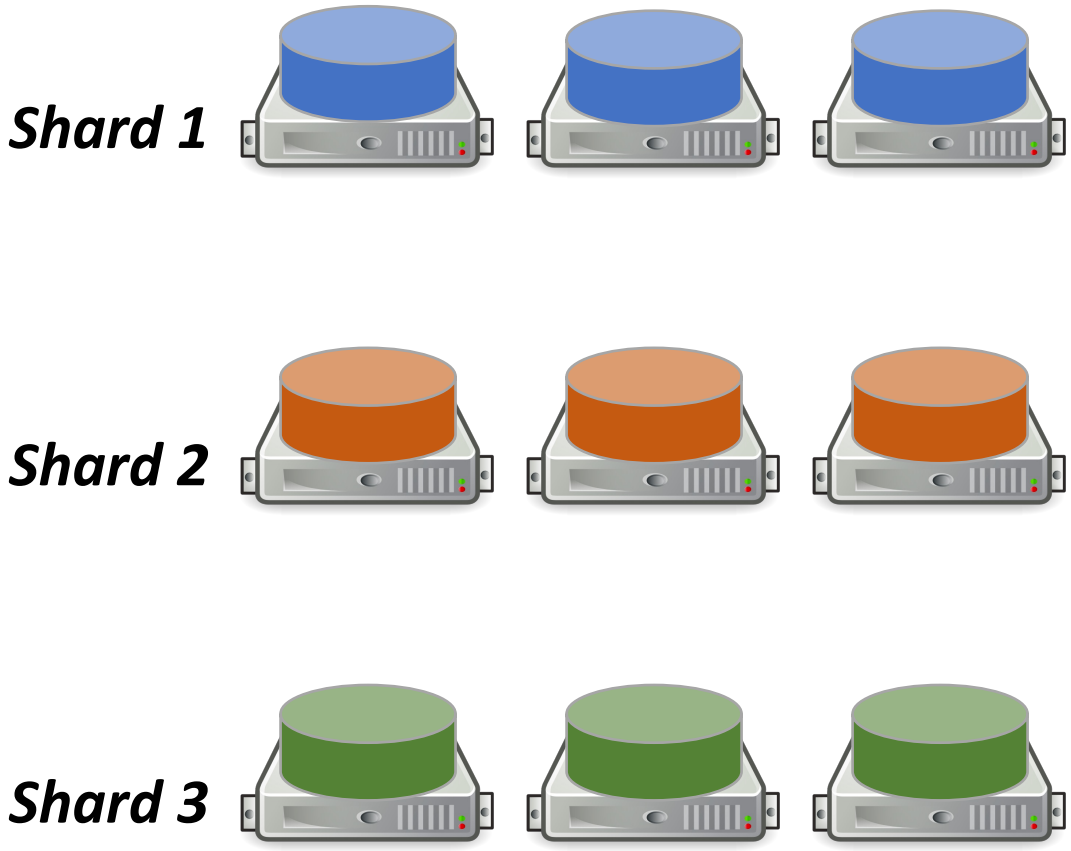
[1]*National University of Singapore,* [2]*University of Washington,* [3]*Microsoft Research*

Scalability by *Sharding*
Fault Tolerance by *Replication*

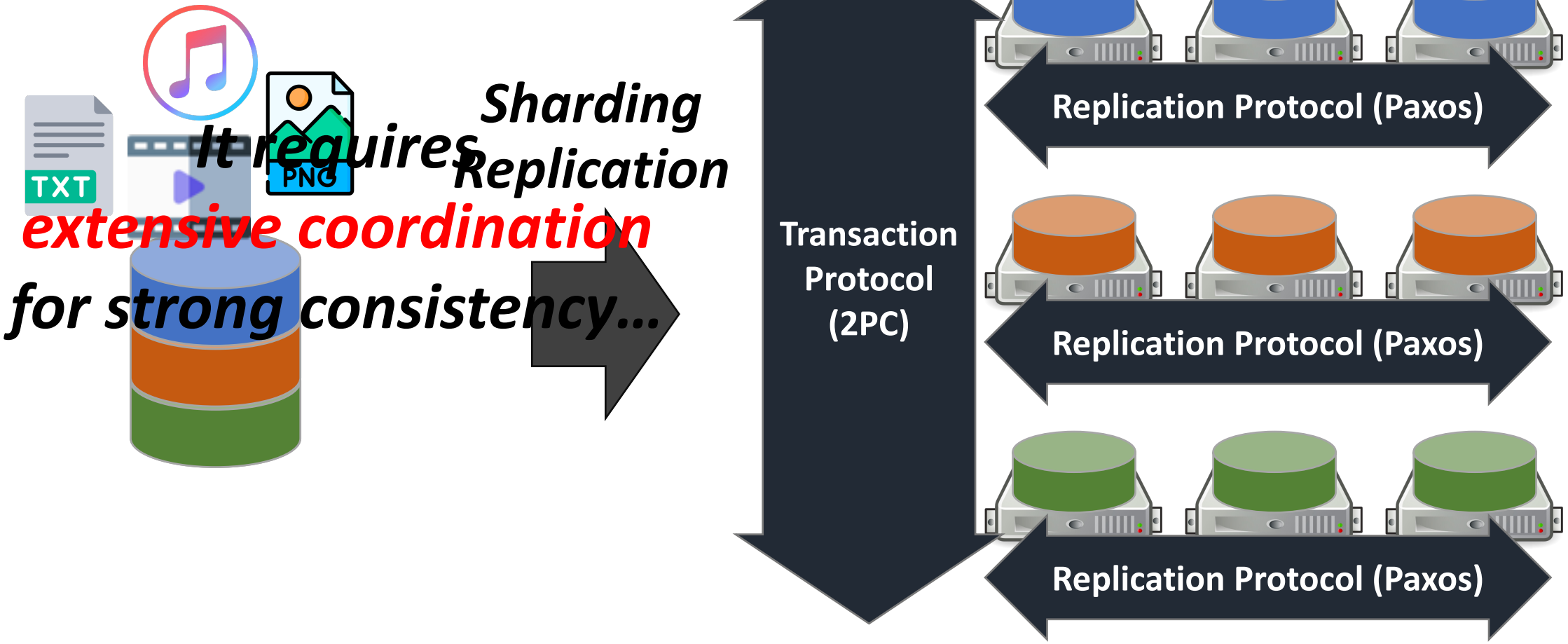


Sharding
Replication





Scalability by *Sharding*
Fault Tolerance by *Replication*



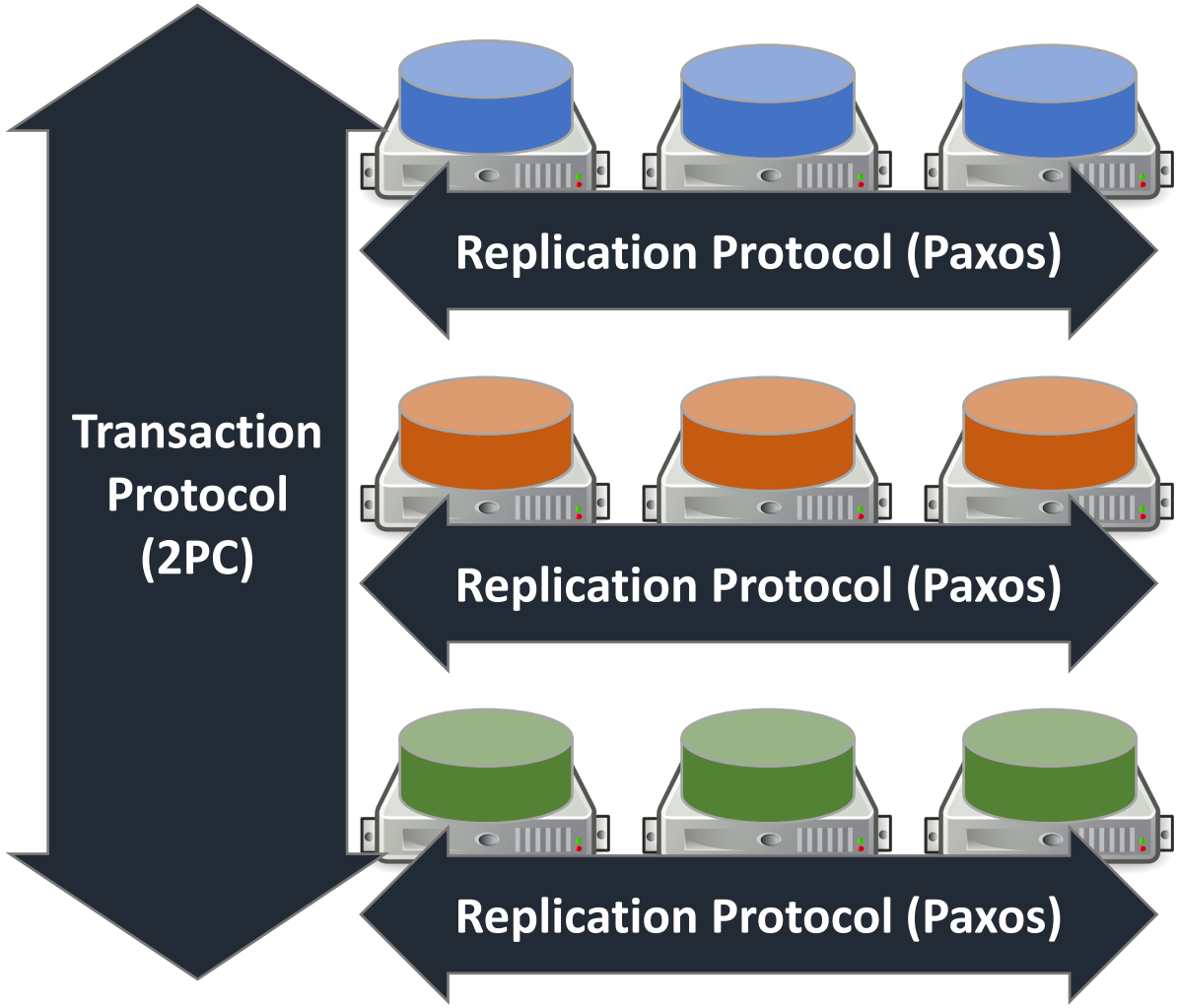
Network Ordering to Eliminate Coordination

Network

Sequencer
(e.g., Programmable Switch)

Consistent Ordering

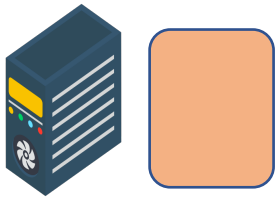
Related Works:
NOPaxos [OSDI '16], **Eris** [SOSP '17]

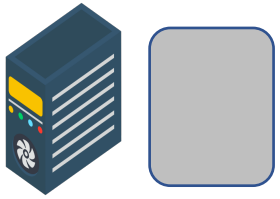


Network Ordering to Eliminate Coordination

Network Ordering to Eliminate Coordination

Senders

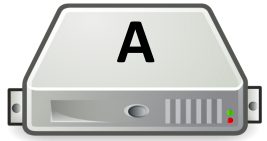


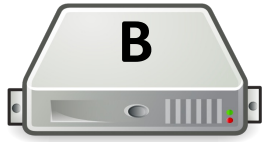


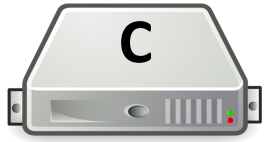
Sequencer

Counter
0

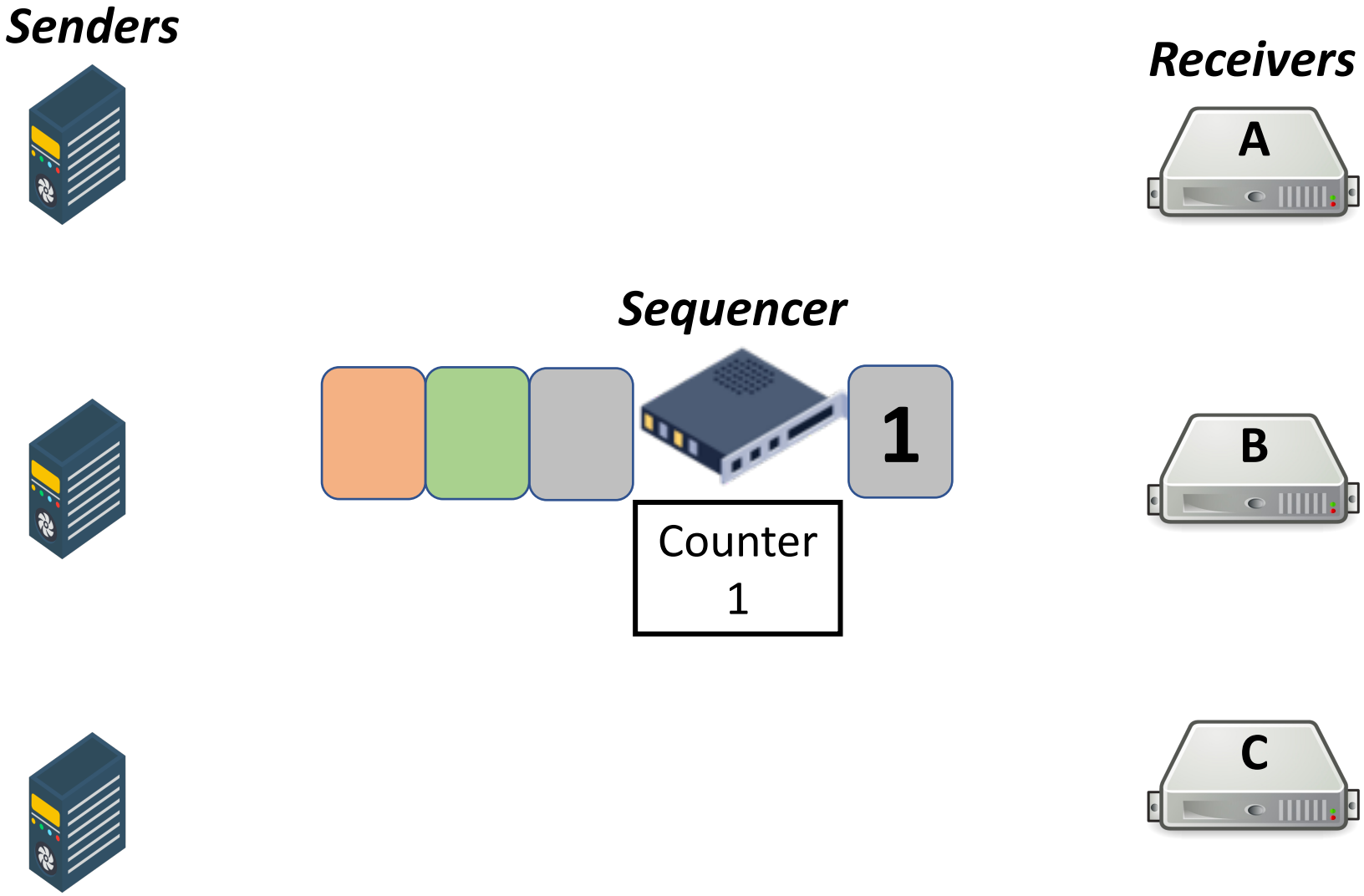
Receivers



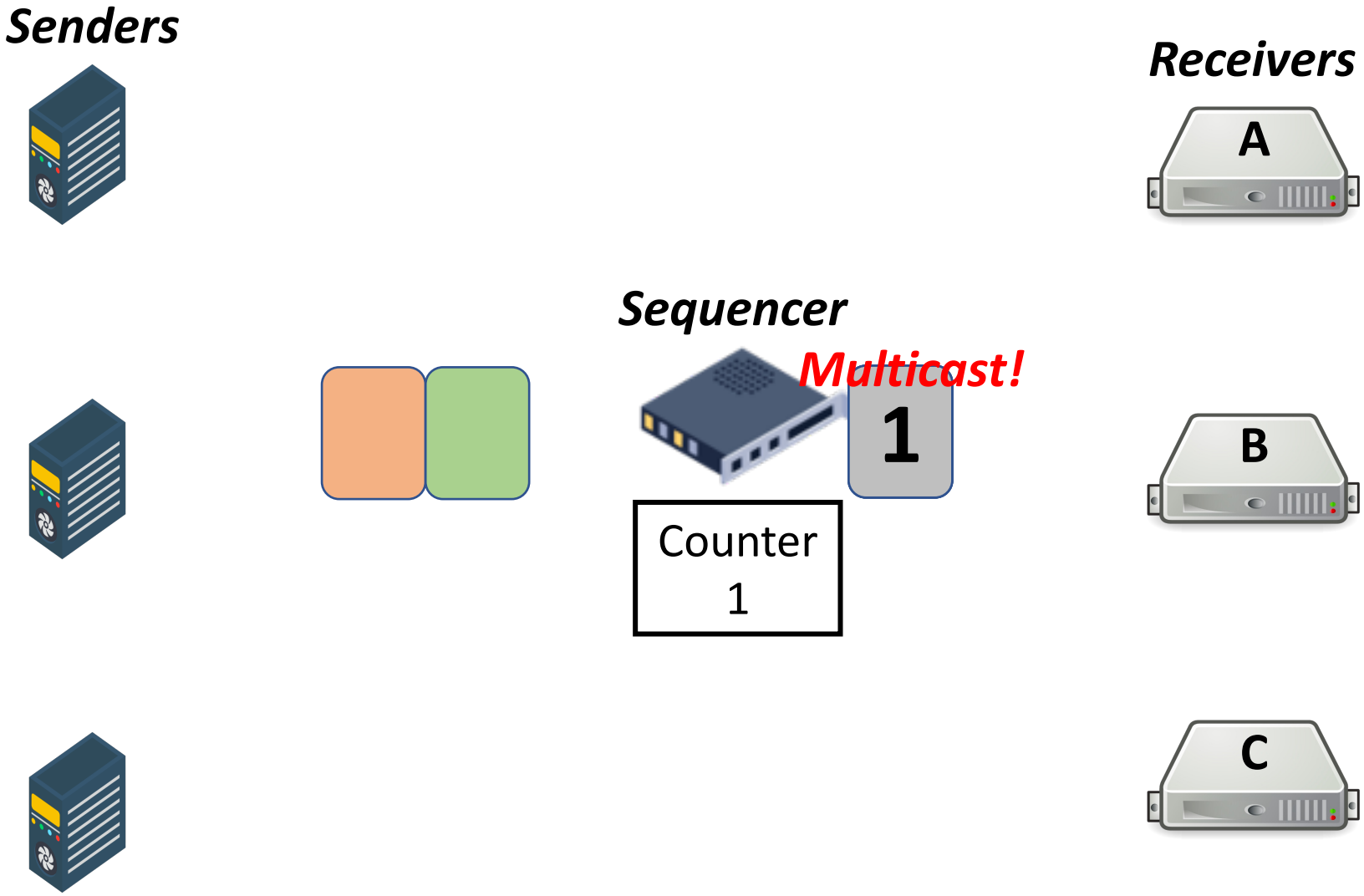




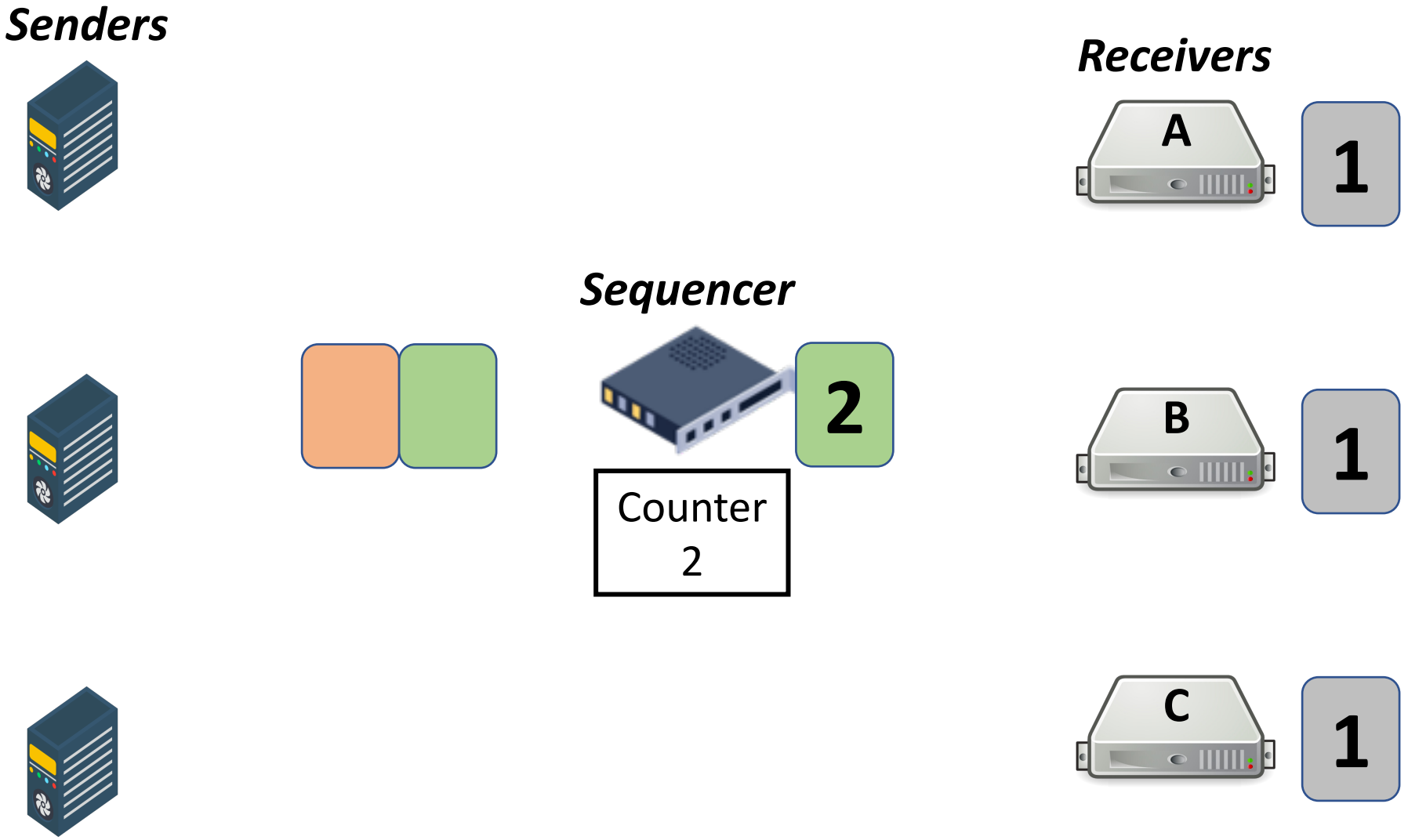
Network Ordering to Eliminate Coordination



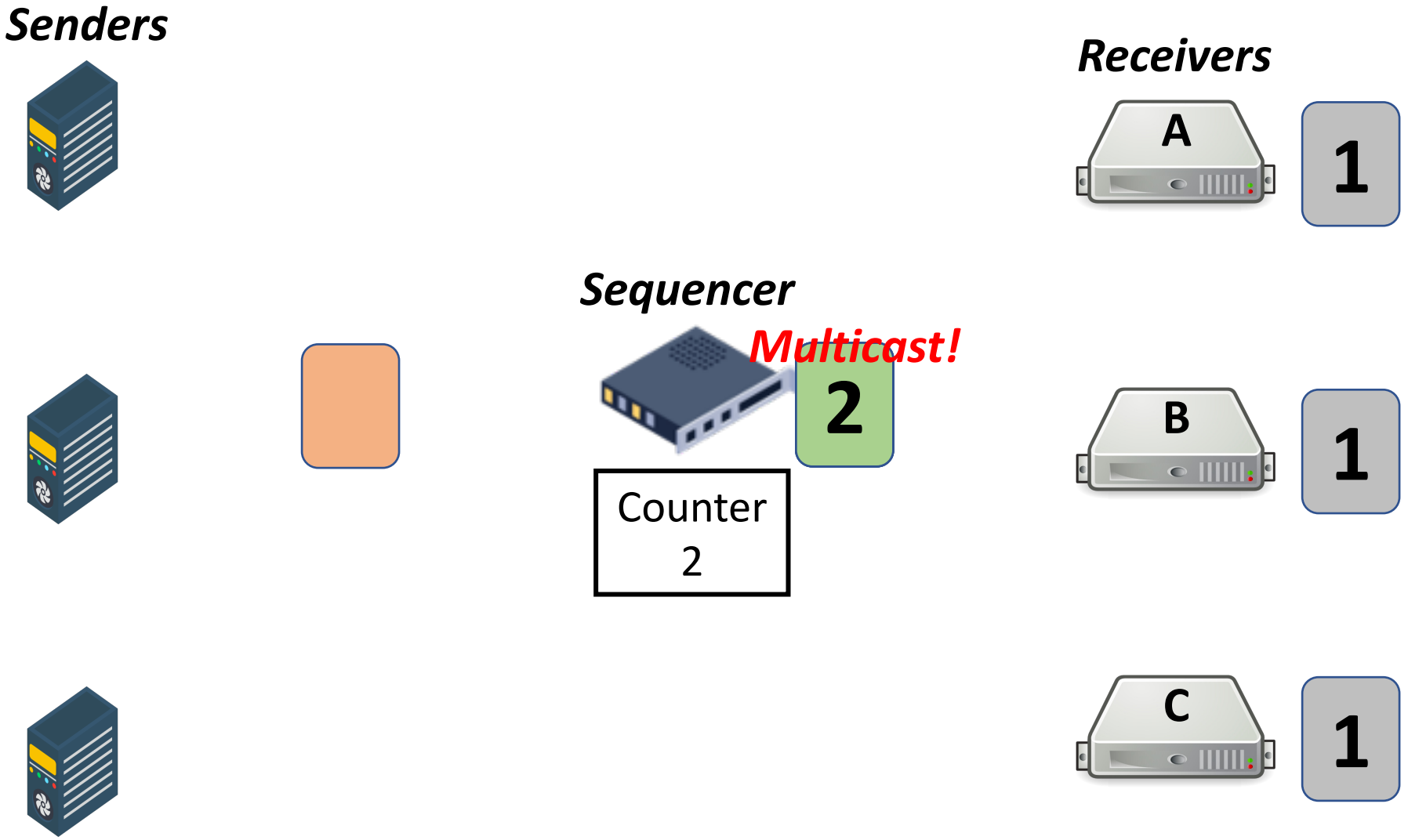
Network Ordering to Eliminate Coordination



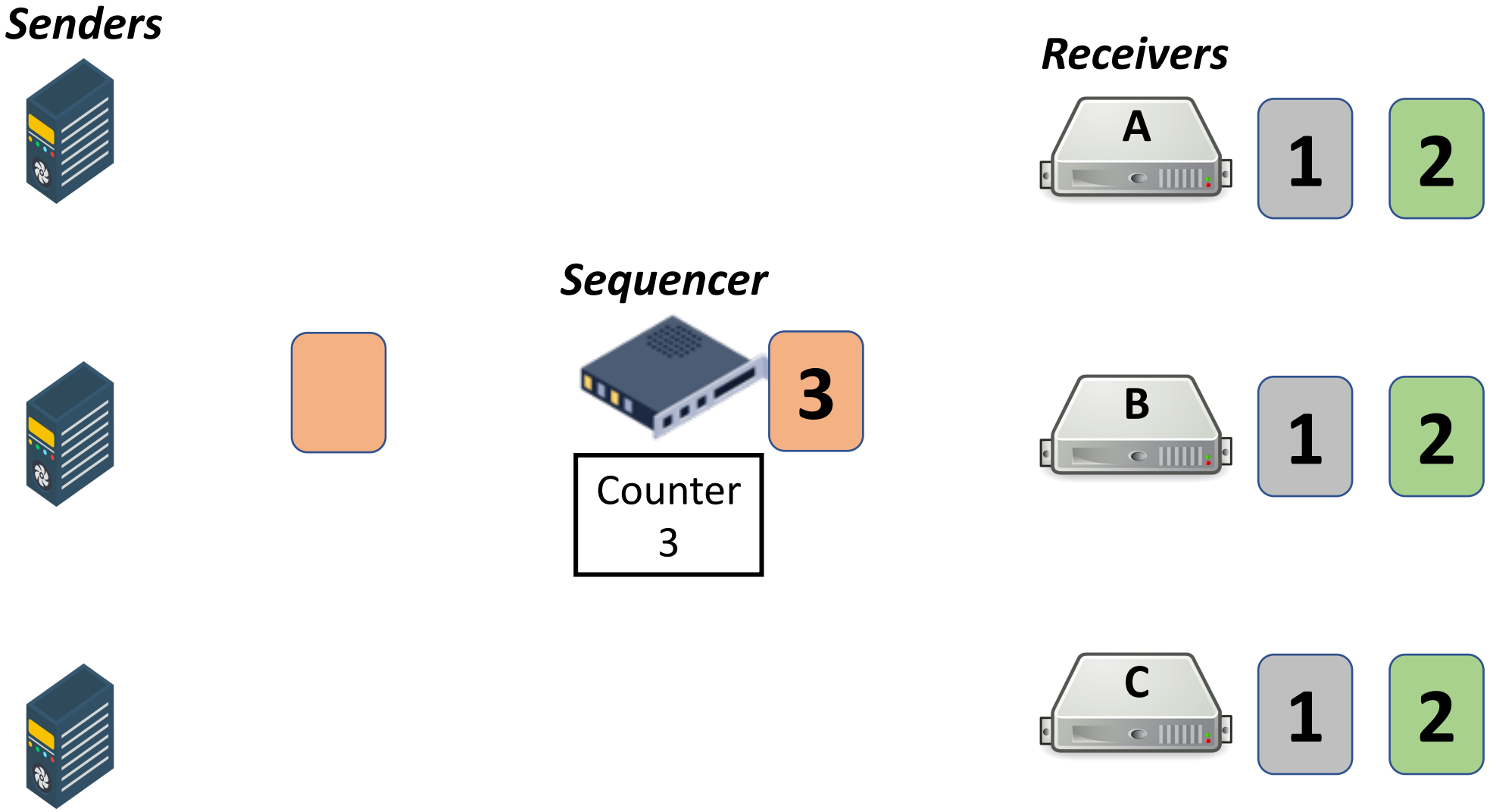
Network Ordering to Eliminate Coordination



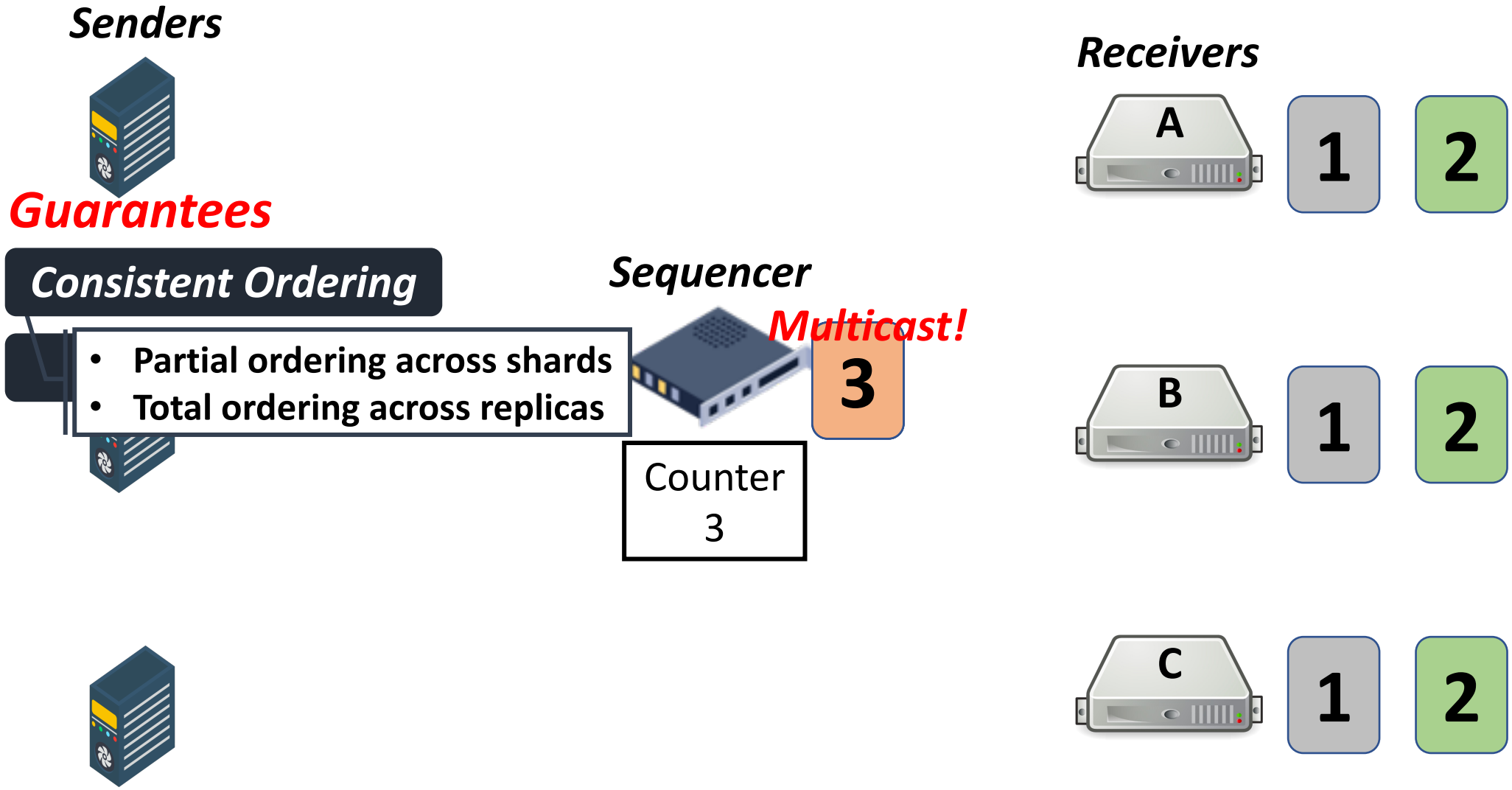
Network Ordering to Eliminate Coordination



Network Ordering to Eliminate Coordination



Network Ordering to Eliminate Coordination



Network Ordering to Eliminate Coordination

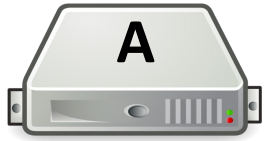
Guarantees

Consistent Ordering

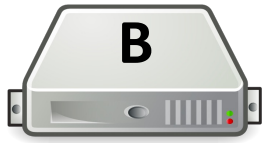
Drop Detection

Sequencer

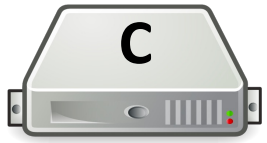
Receivers



1 2 3

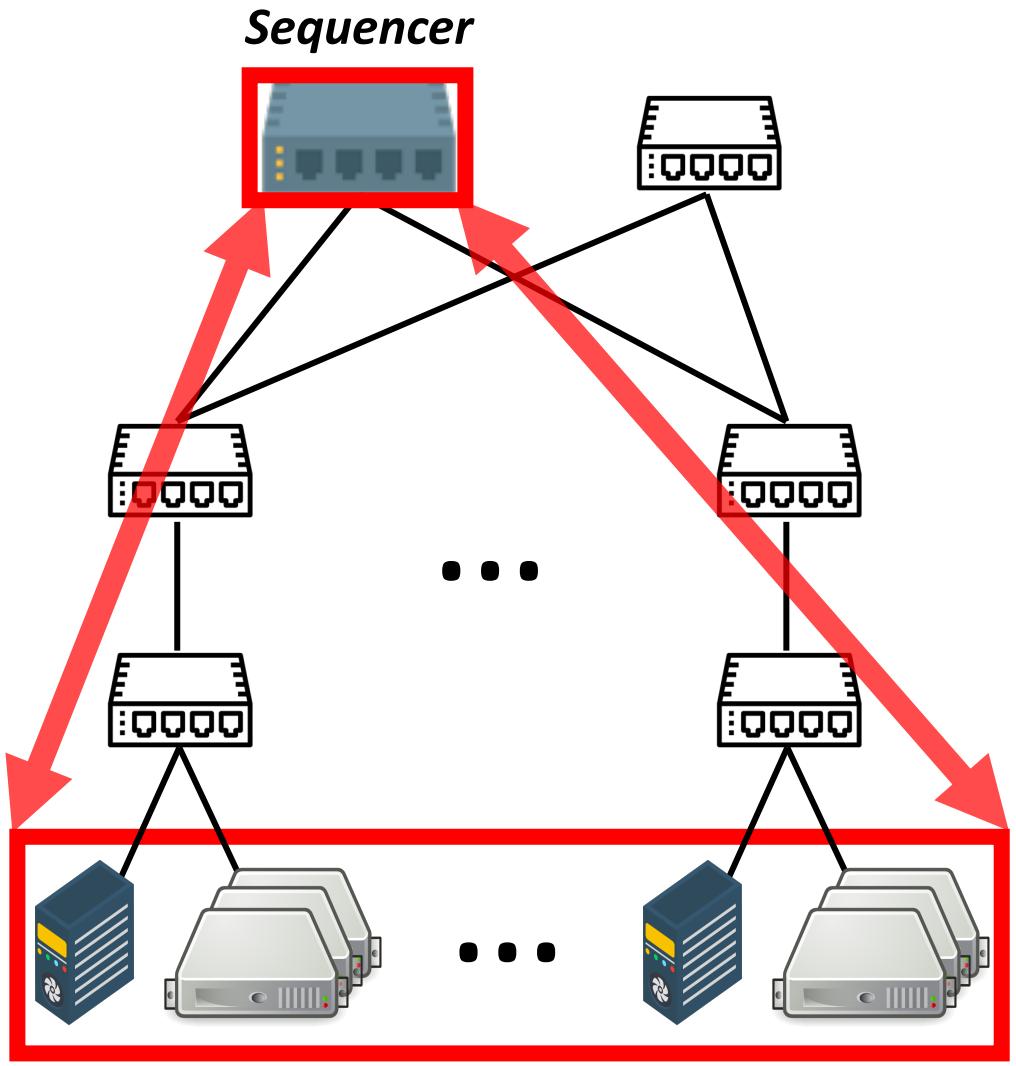


1 2 3



1 2 3

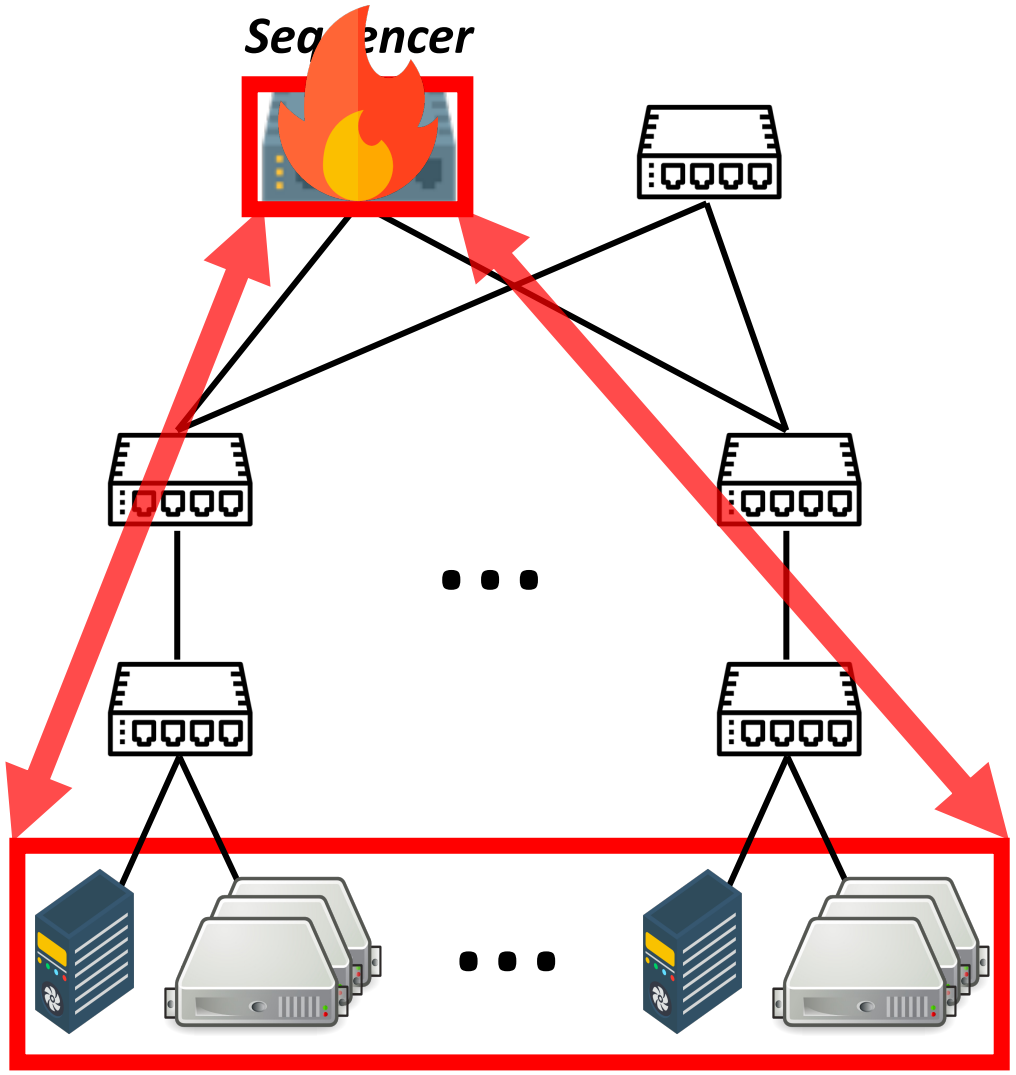
Drawbacks due to the Single Sequencer



All request traffic
must go through the *single sequencer*
(Network Serialization)

☒ ***Network load imbalance ⇒ high latency***
☒ ***Sequencer scalability bottleneck***
☒ ***Prolonged sequencer failover***

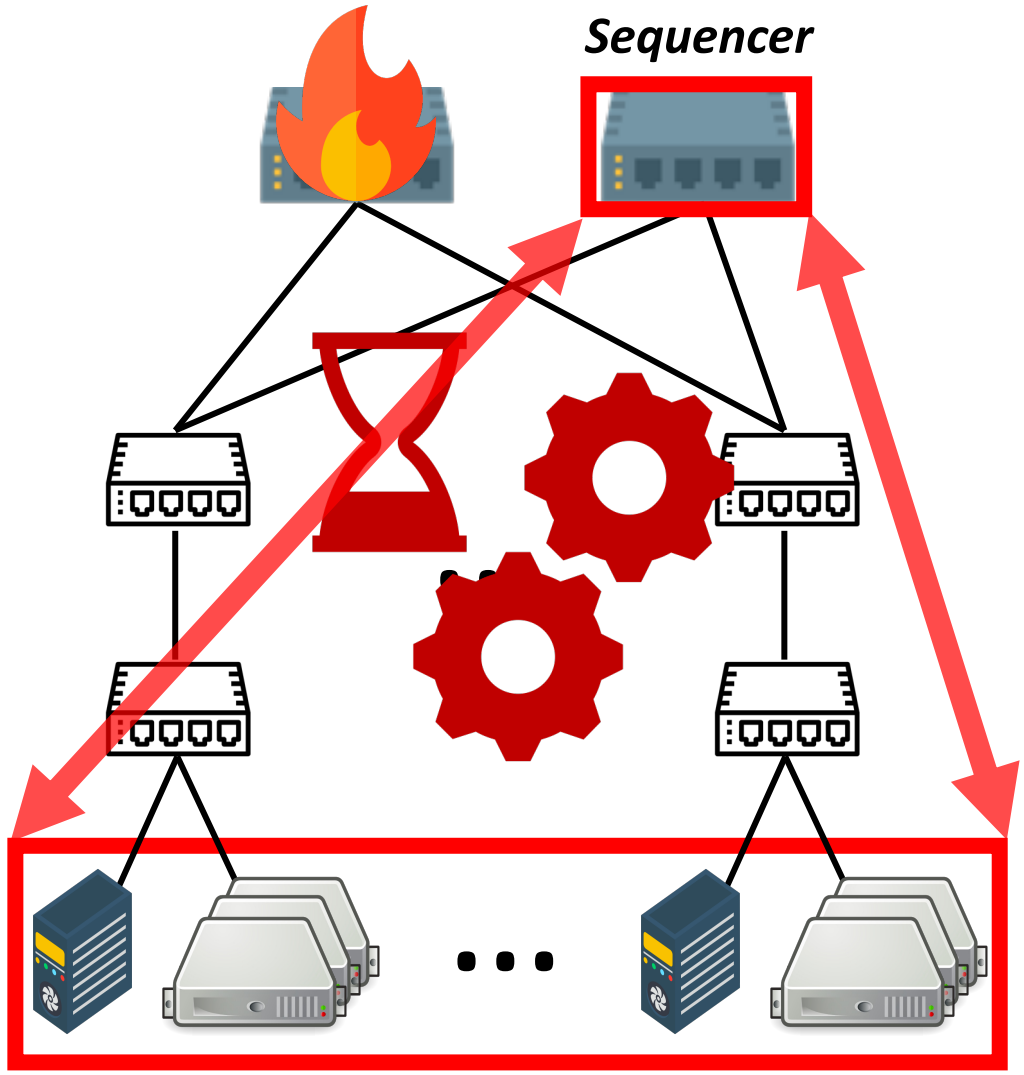
Drawbacks due to the Single Sequencer



All request traffic
must go through the *single sequencer*
(Network Serialization)

☒ ***Network load imbalance ⇒ high latency***
☒ ***Sequencer scalability bottleneck***
☒ ***Prolonged sequencer failover***

Drawbacks due to the Single Sequencer



All request traffic
must go through the *single sequencer*
(Network Serialization)

☒ ***Network load imbalance ⇒ high latency***
☒ ***Sequencer scalability bottleneck***
☒ ***Prolonged sequencer failover***

Can we achieve network ordering ***without serialization***?

All request traffic must go through the *single sequencer*

(*Network Serialization*)

☒ *Network load imbalance ⇒ high latency*

☒ *Sequencer scalability bottleneck*

☒ *Prolonged sequencer failover*

Hydra

Use multiple sequencers concurrently

(*Serialization-Free*)

✅ *Network load balancing ⇒ low latency* (e.g., 13x)

✅ *Higher scalability beyond a single sequencer*

✅ *Faster sequencer failover* (e.g., 5x)

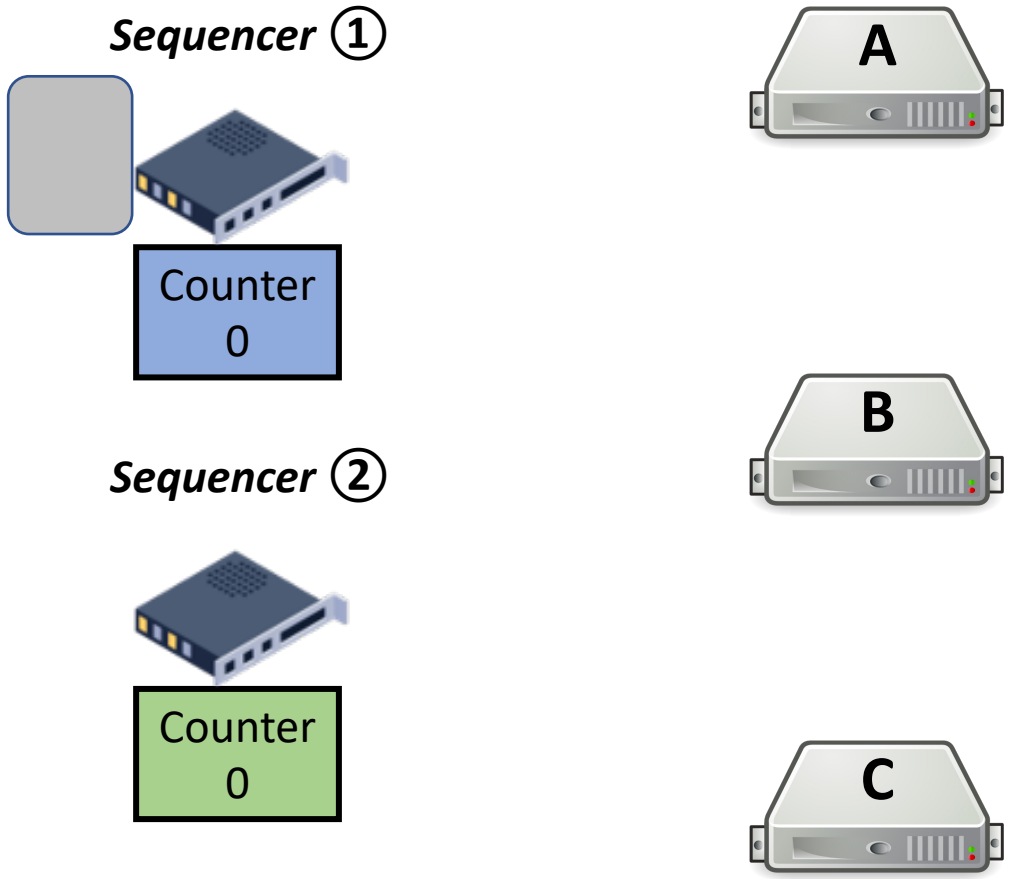
Outline

1. Introduction
2. Hydra Network Primitive
3. Handling Network Anomalies
4. Evaluation

Multi-Sequencer Challenge:
Provide the same guarantees

Guarantees

- ***Consistent Ordering***
- ***Drop Detection***

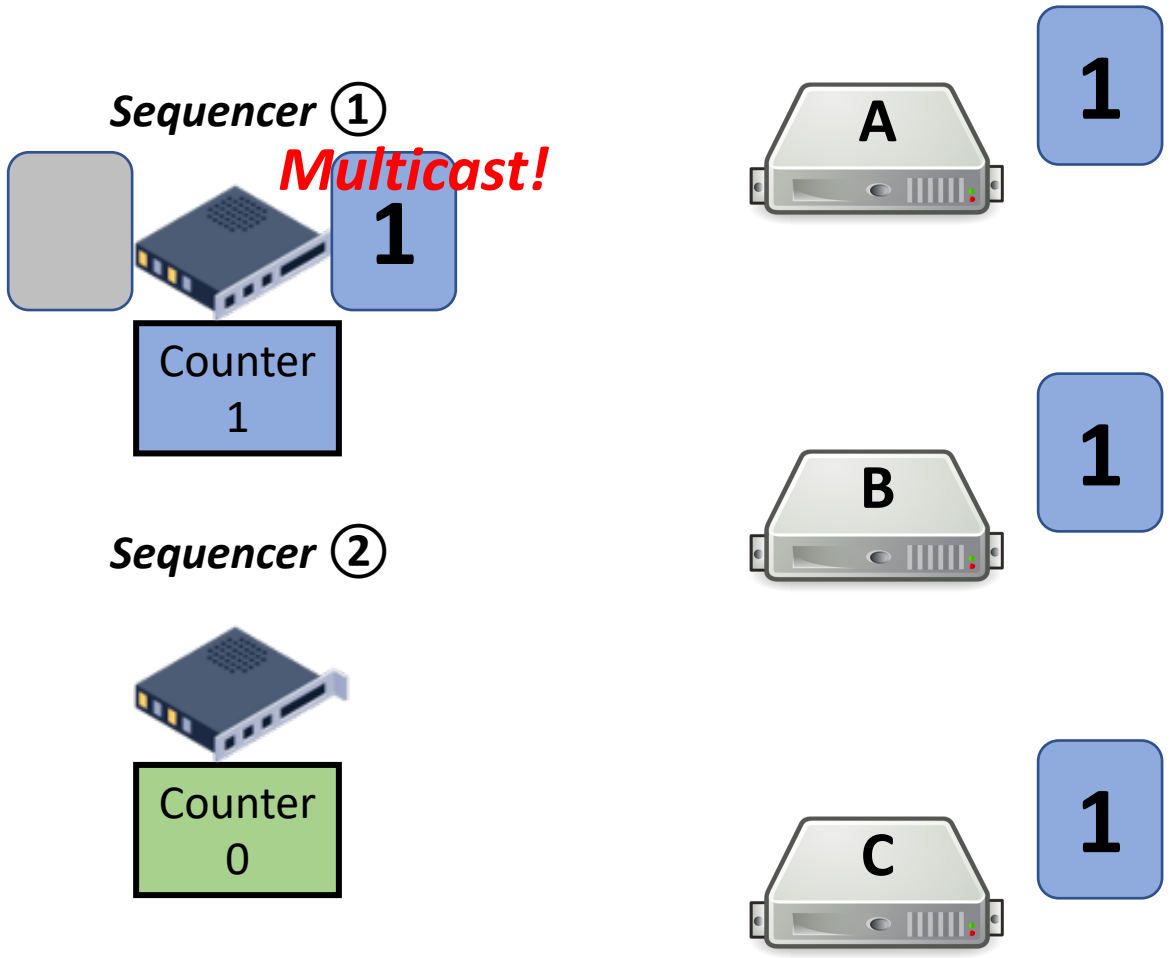


Multi-Sequencer Challenge:

Provide the same guarantees

Guarantees

- ***Consistent Ordering***
- ***Drop Detection***

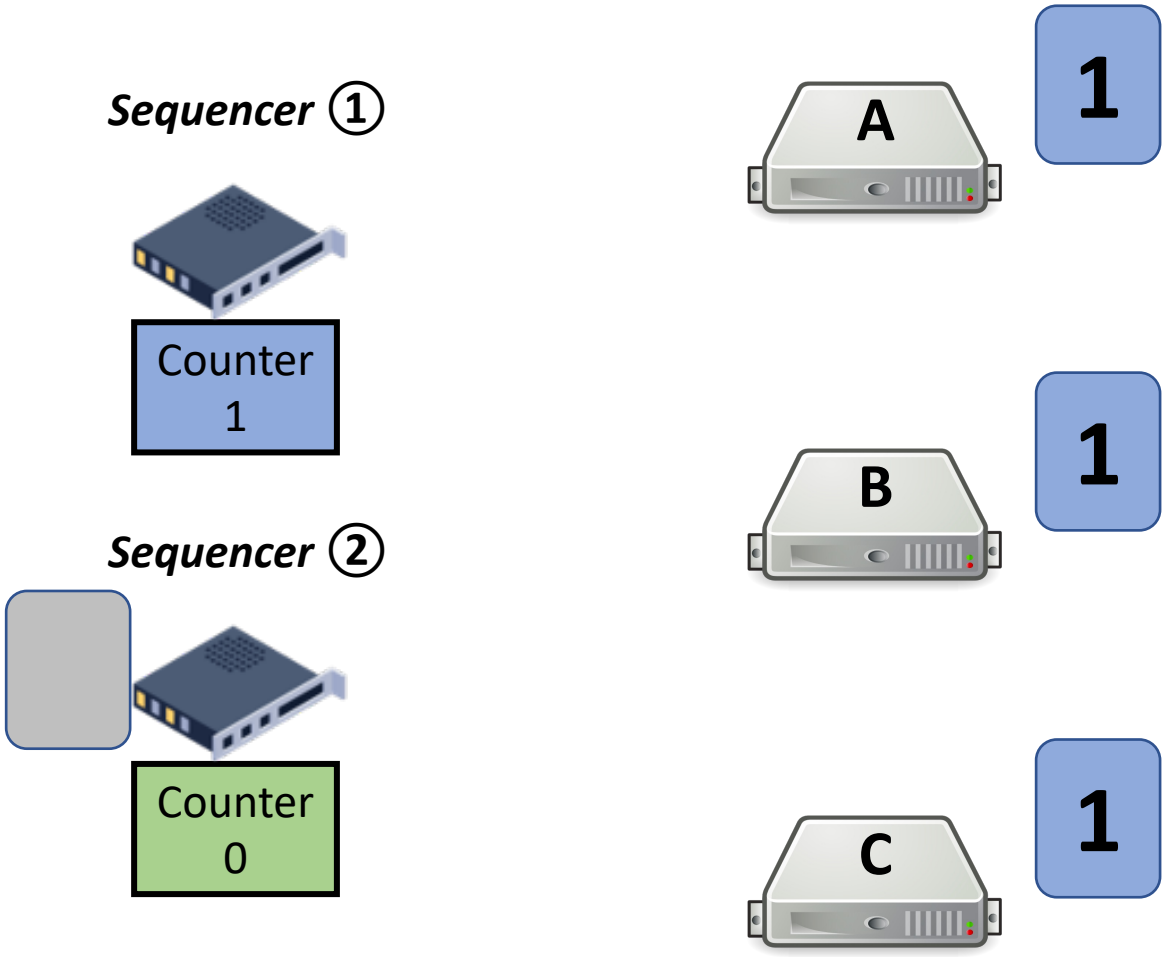


Multi-Sequencer Challenge:
Provide the same guarantees

Guarantees

Consistent Ordering

Drop Detection



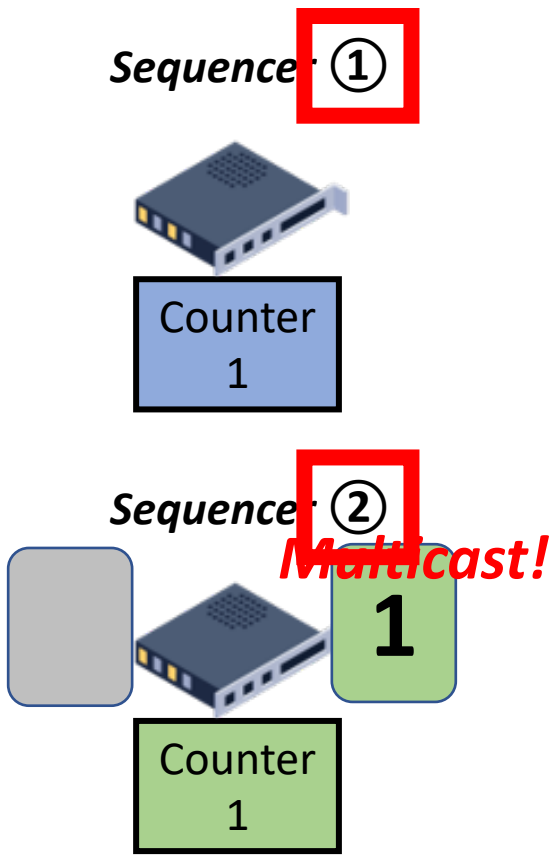
Multi-Sequencer Challenge:

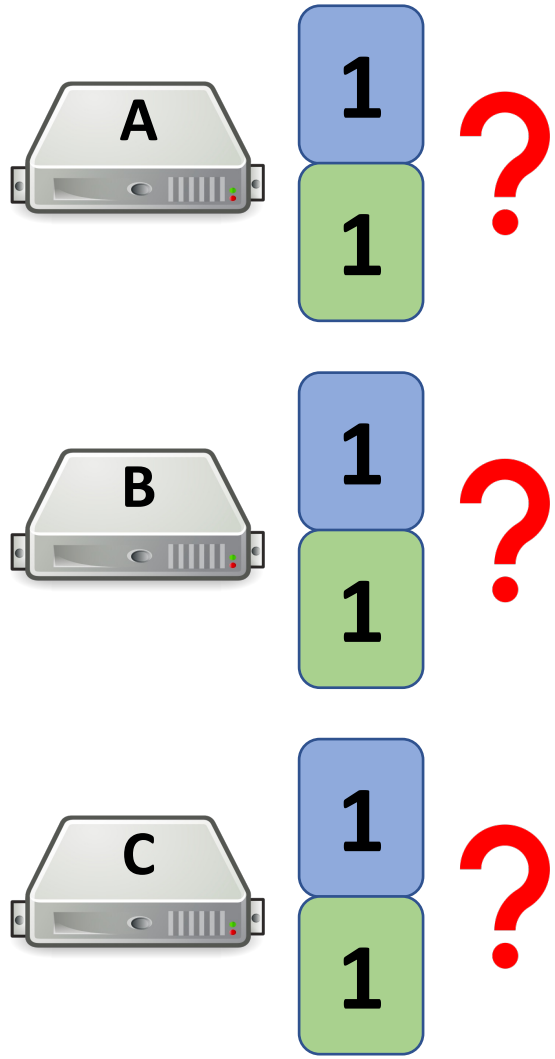
Provide the same guarantees

Guarantees

~~***Consistent Ordering***~~

Drop Detection



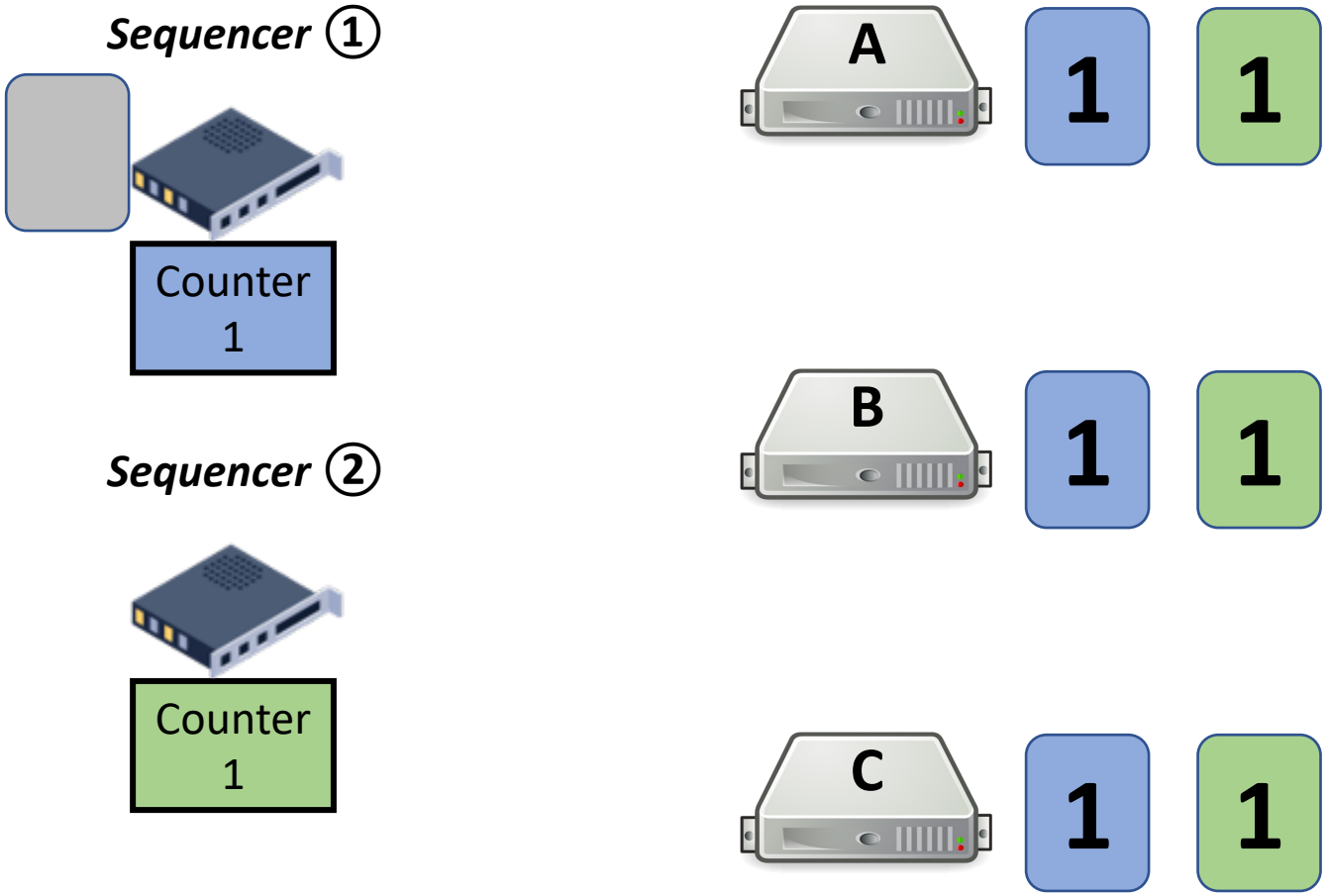


Multi-Sequencer Challenge:
Provide the same guarantees

Guarantees

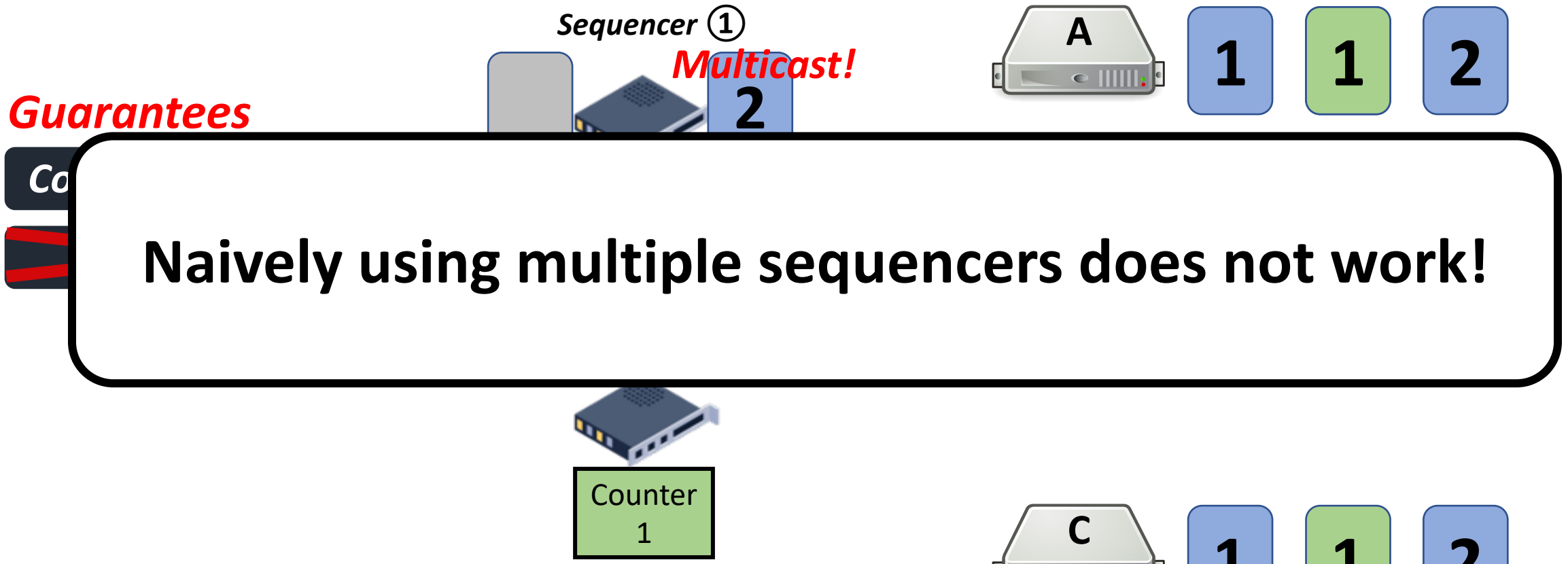
Consistent Ordering

Drop Detection

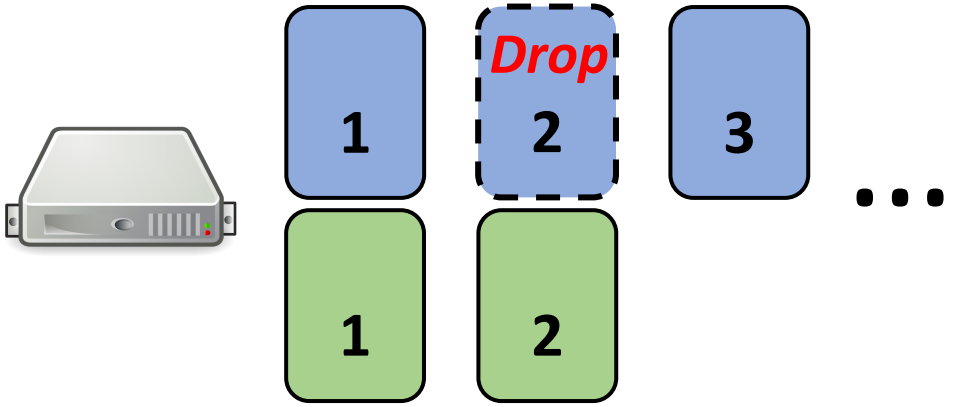


Multi-Sequencer Challenge:

Provide the same guarantees



Solution: Combine sequence number with *physical clock*

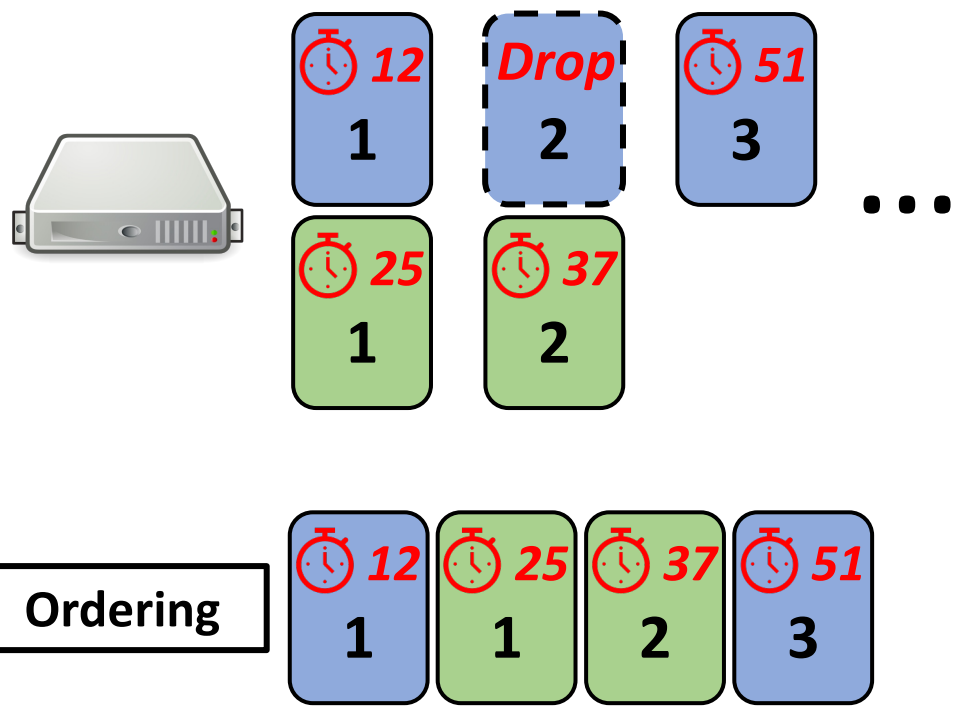


Sequence Number → ***Drop Detection***

Physical Clock Timestamp → ***Consistent Ordering***

- Consistent across receivers
- Strictly monotonically increasing
- Physical clocks can be (loosely) synchronized across sequencers

Solution: Combine sequence number with *physical clock*

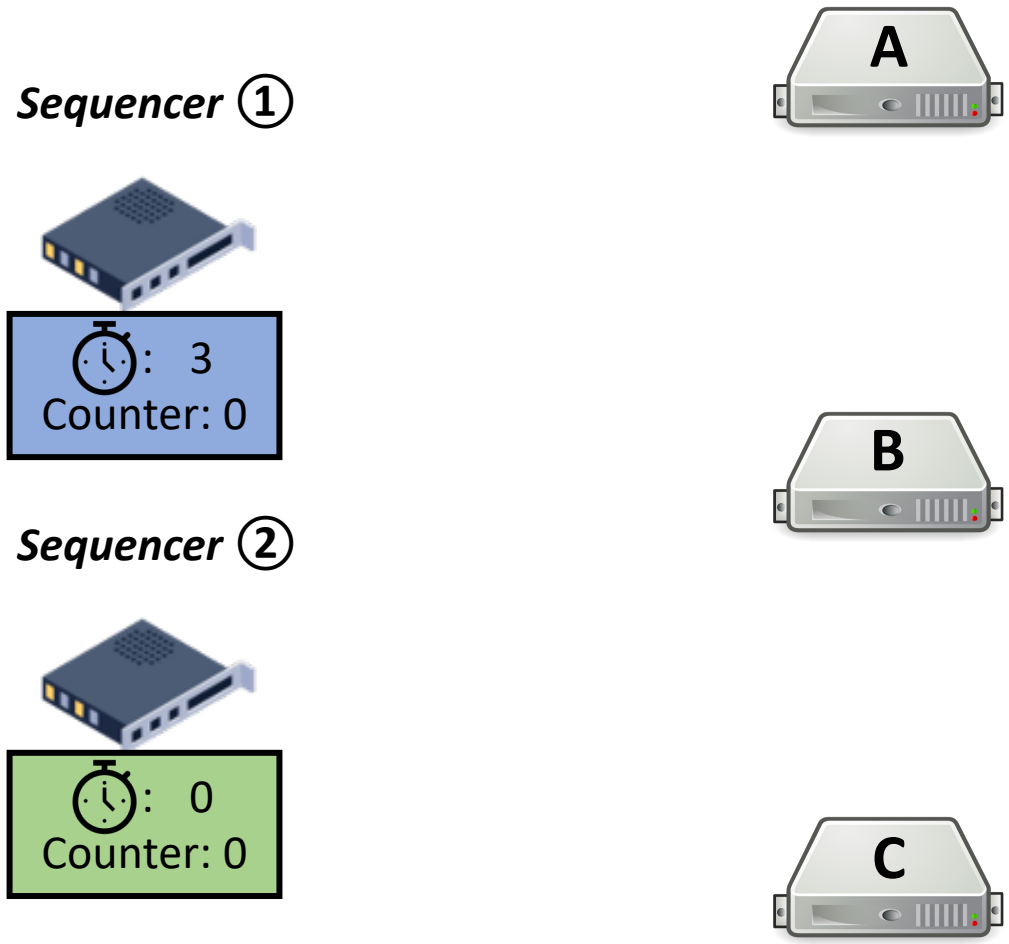


Sequence Number → ***Drop Detection***

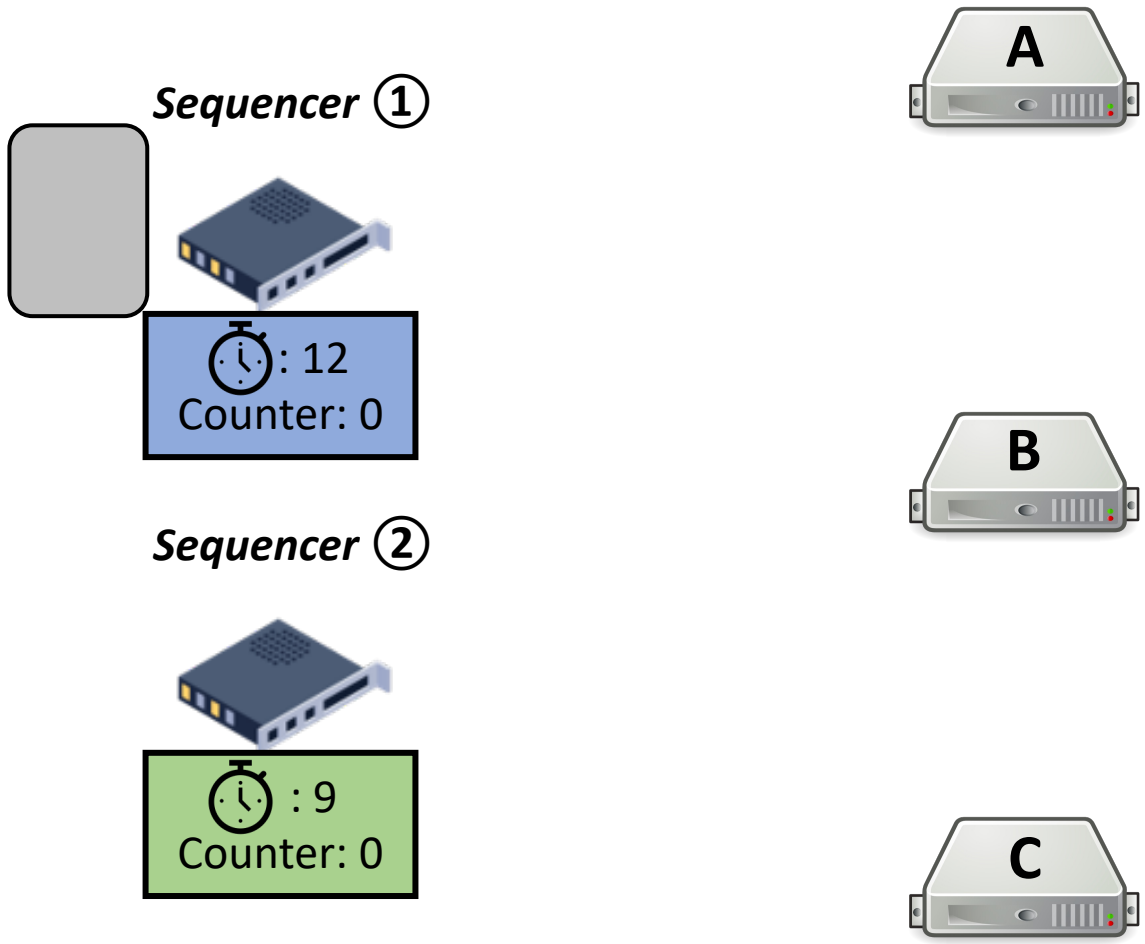
Physical Clock Timestamp → ***Consistent Ordering***

- Consistent across receivers
- Strictly monotonically increasing
- Physical clocks can be (loosely) synchronized across sequencers

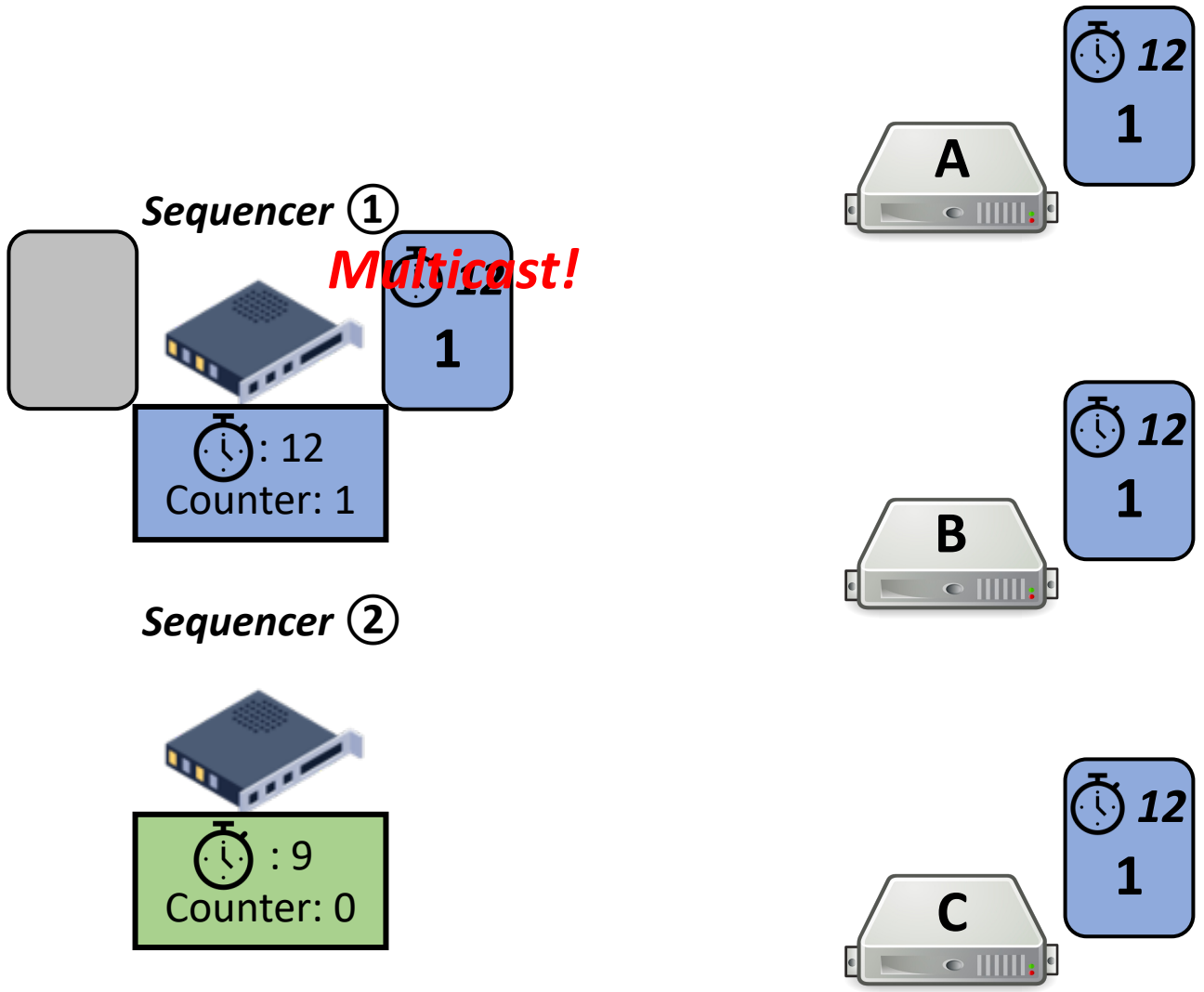
Hydra Network Primitive



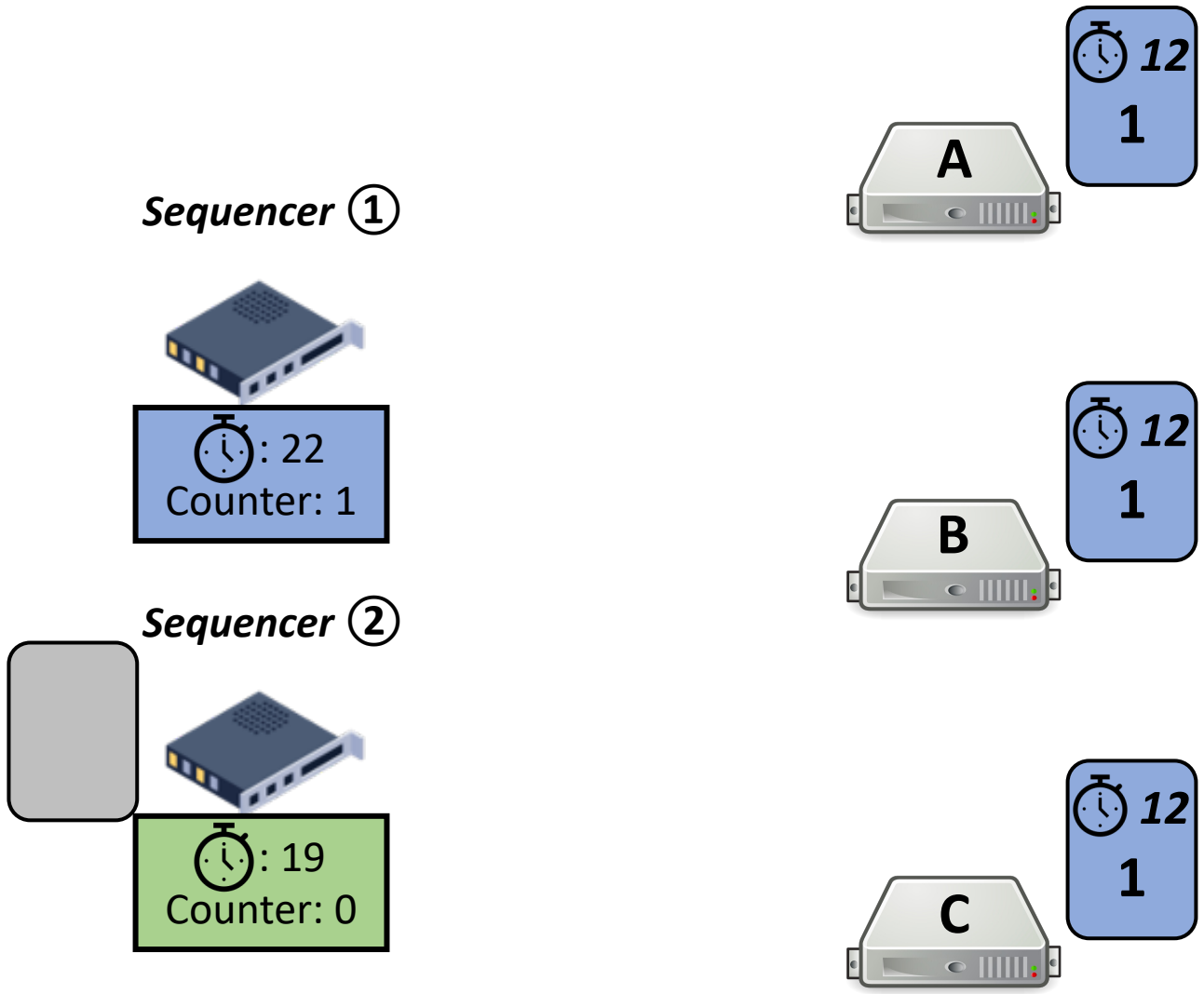
Hydra Network Primitive - Sequencers



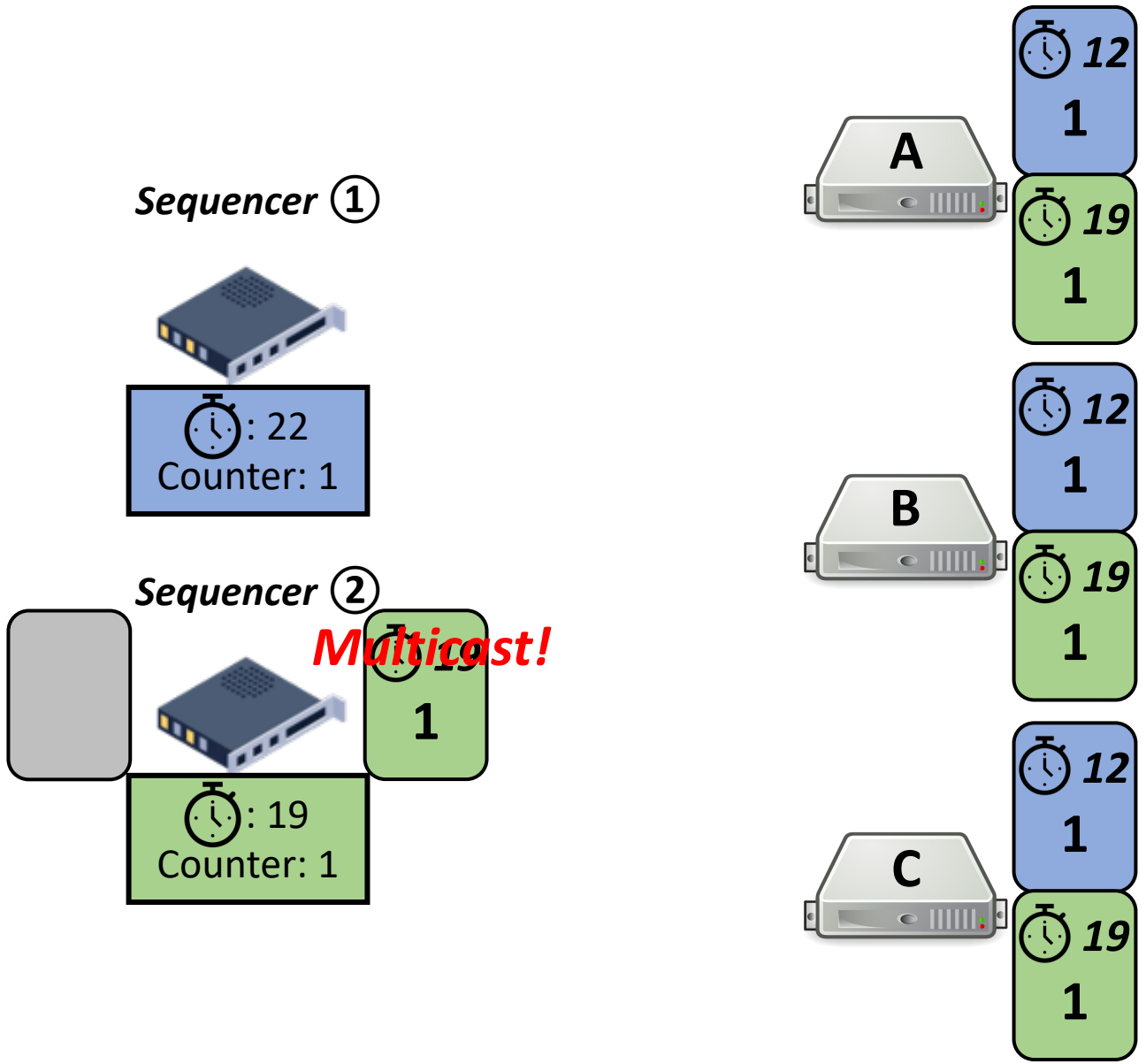
Hydra Network Primitive - Sequencers



Hydra Network Primitive - Sequencers



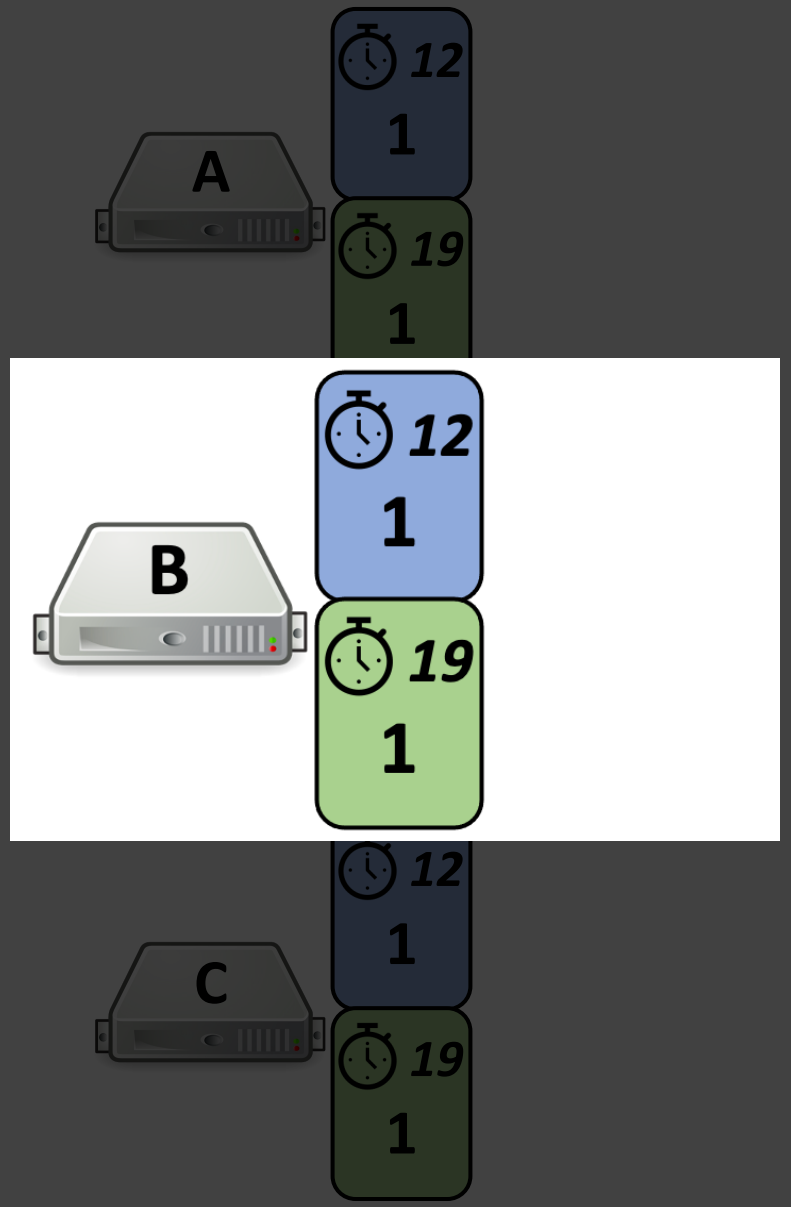
Hydra Network Primitive - Sequencers



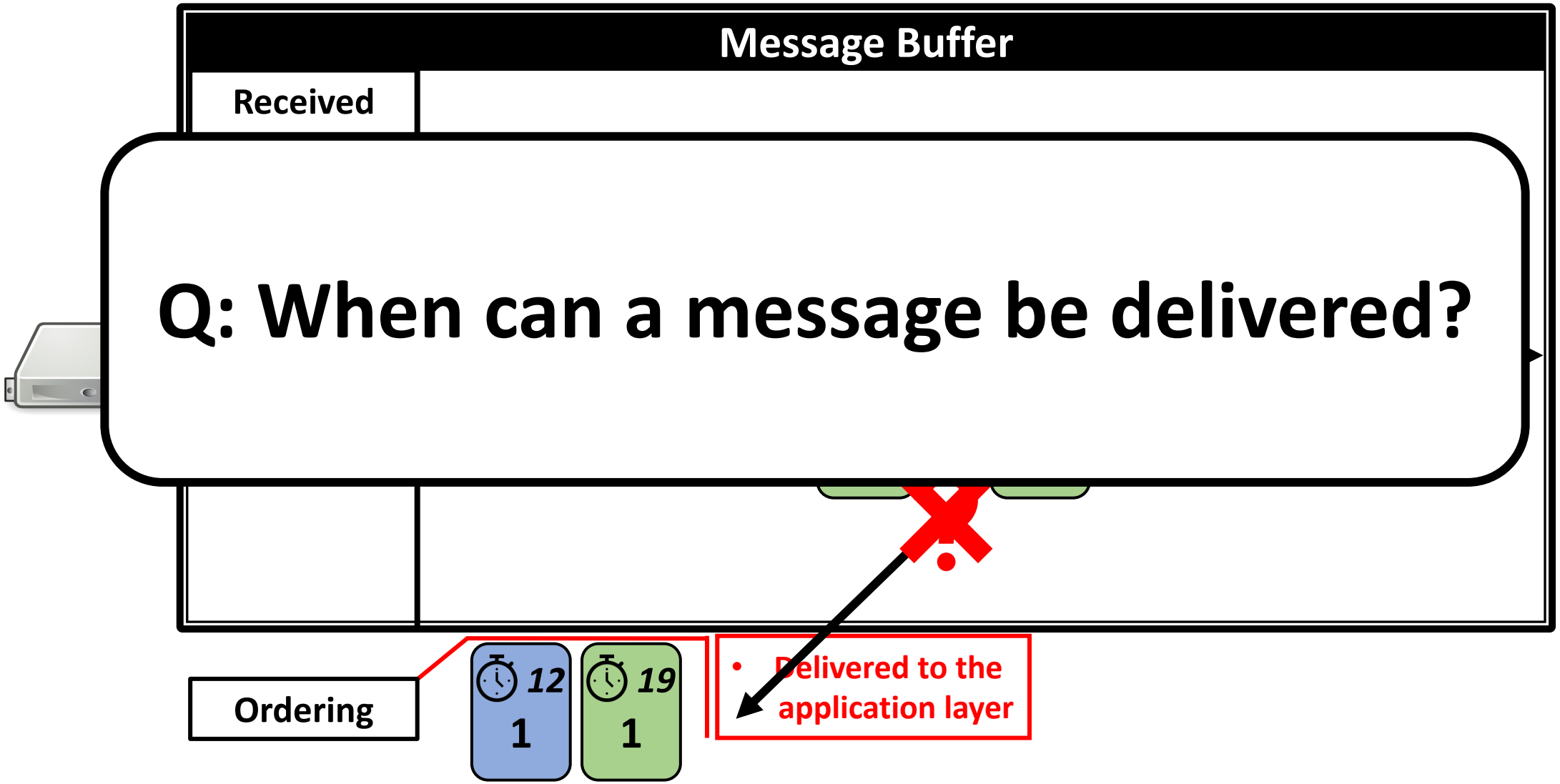
Hydra Network Primitive - Sequencers

Sequencer ①

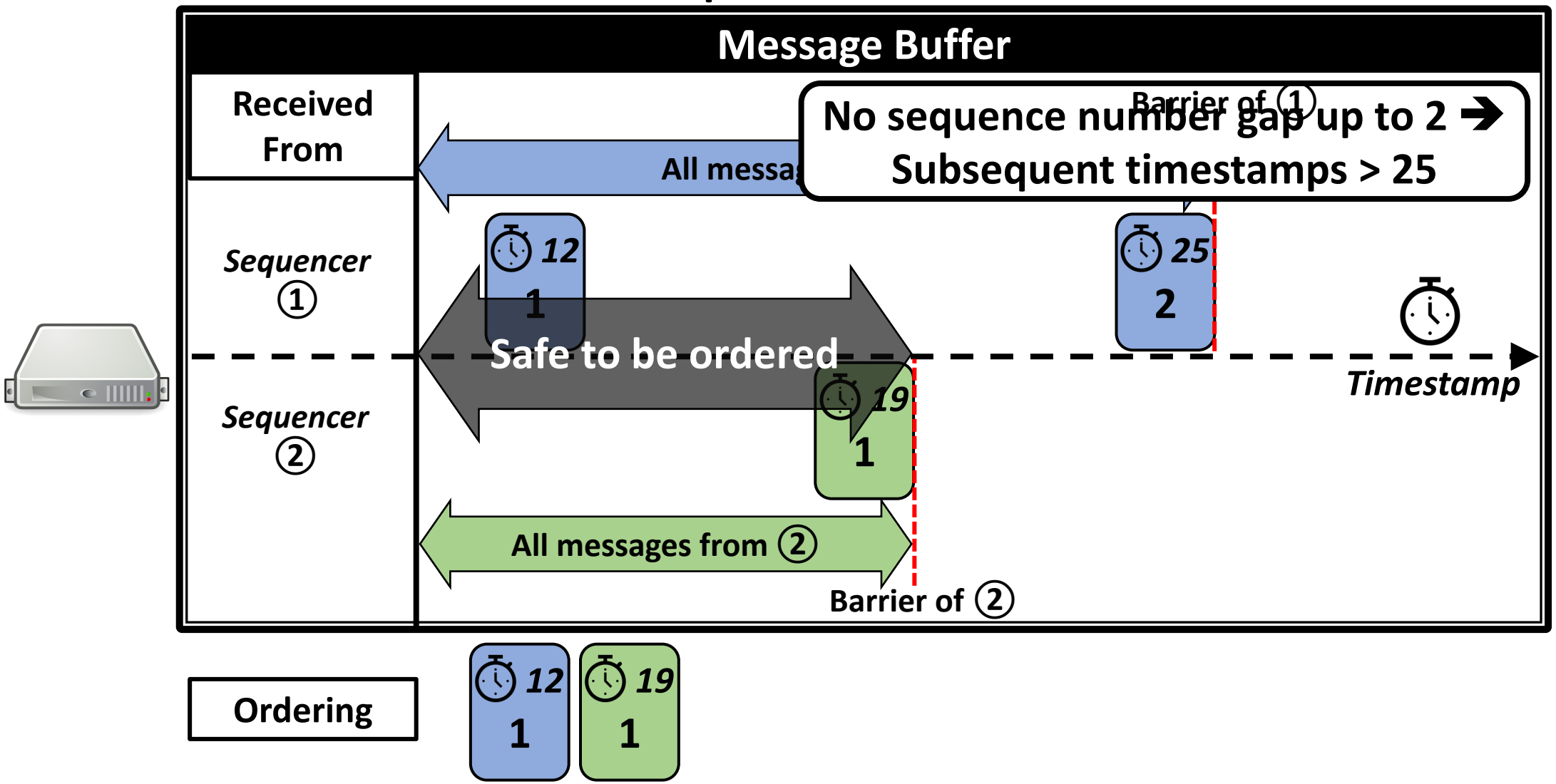
Sequencer ②



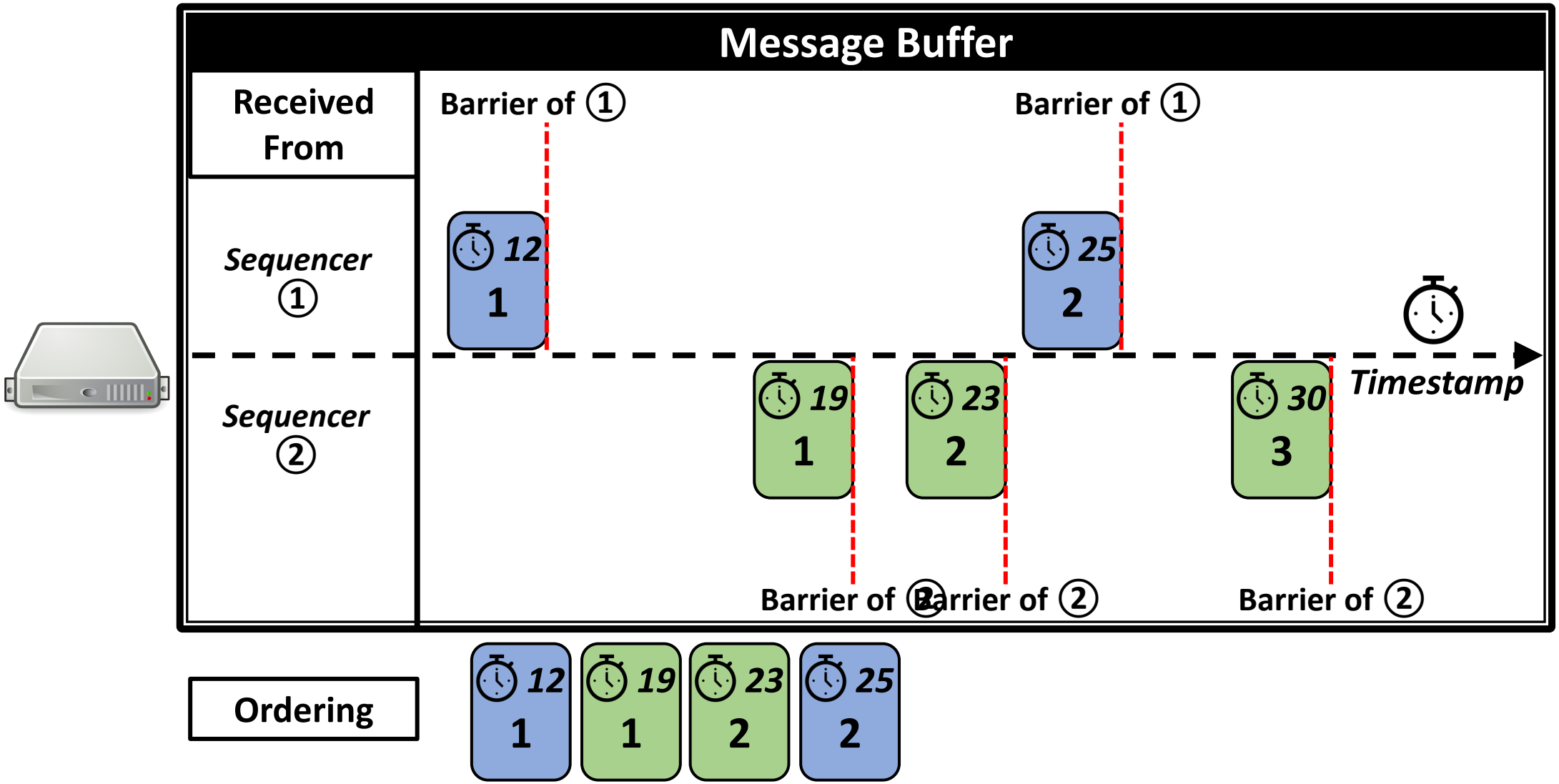
Hydra Network Primitive - Receivers



A: Once ALL messages with lower timestamp have been received



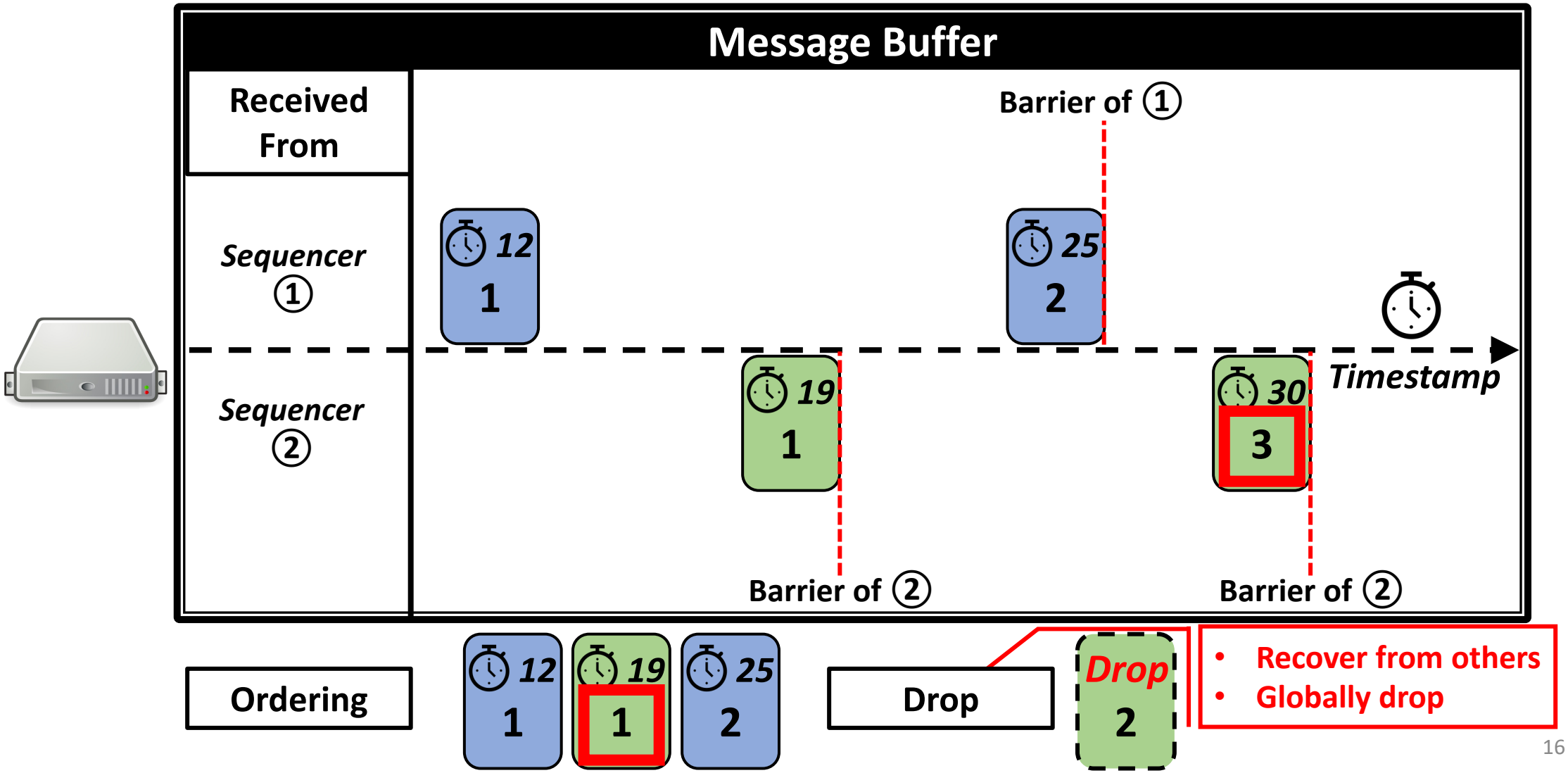
Deliver messages up to *the minimum barrier*



Outline

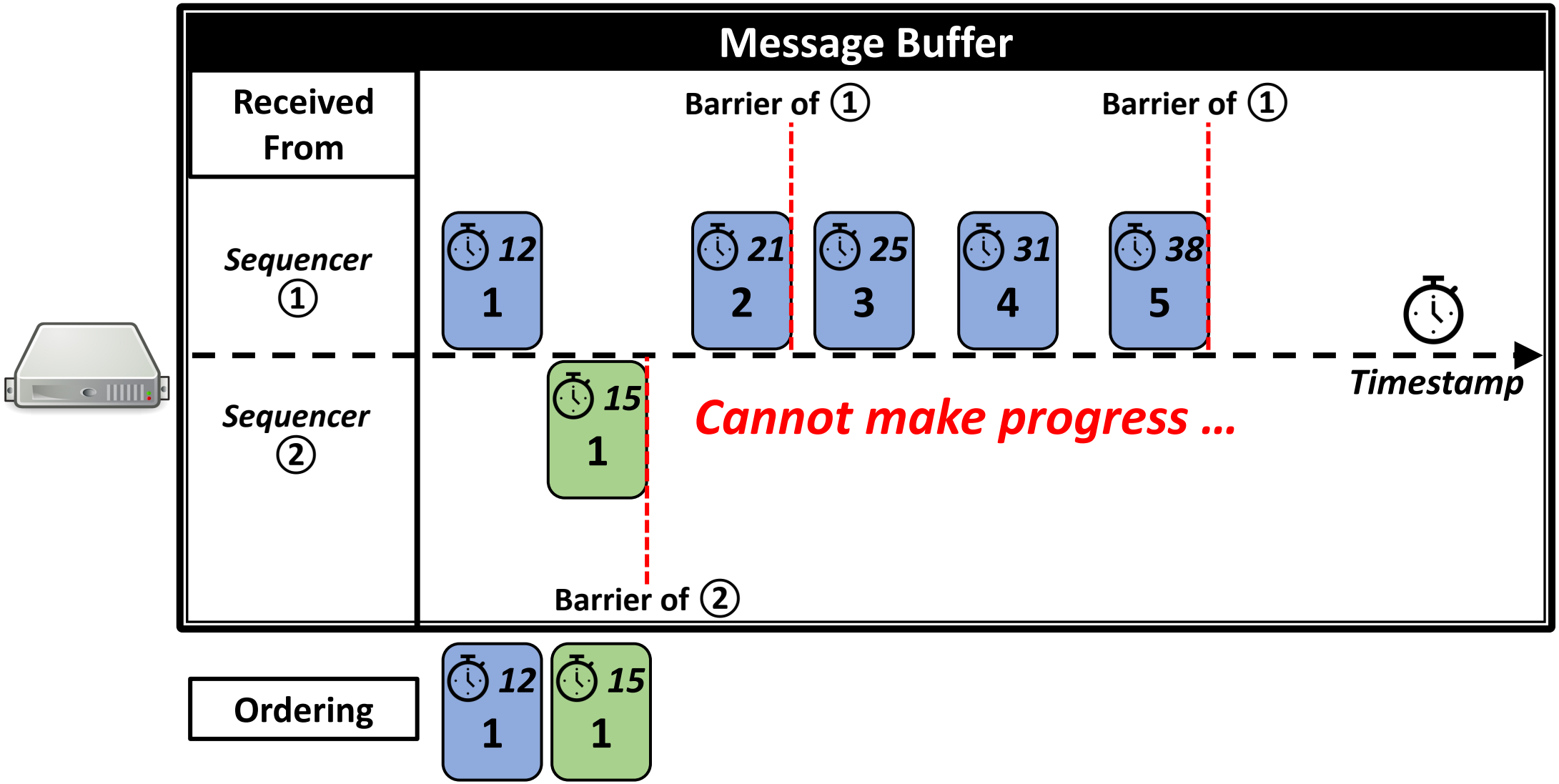
1. Introduction
2. Hydra Network Primitive
3. Handling Network Anomalies
4. Evaluation

How to handle messages drops?

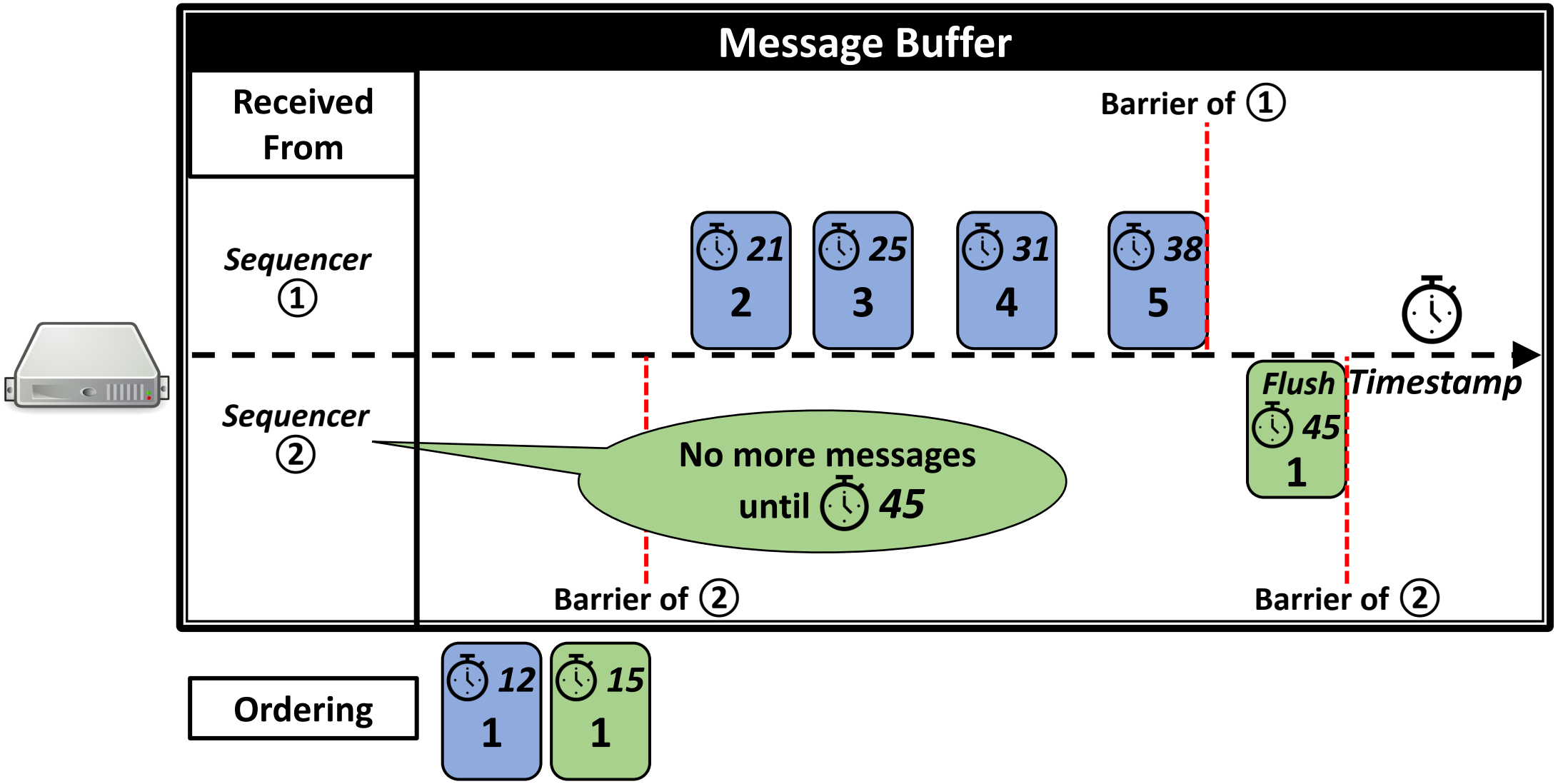


What happens if a sequencer is idle?

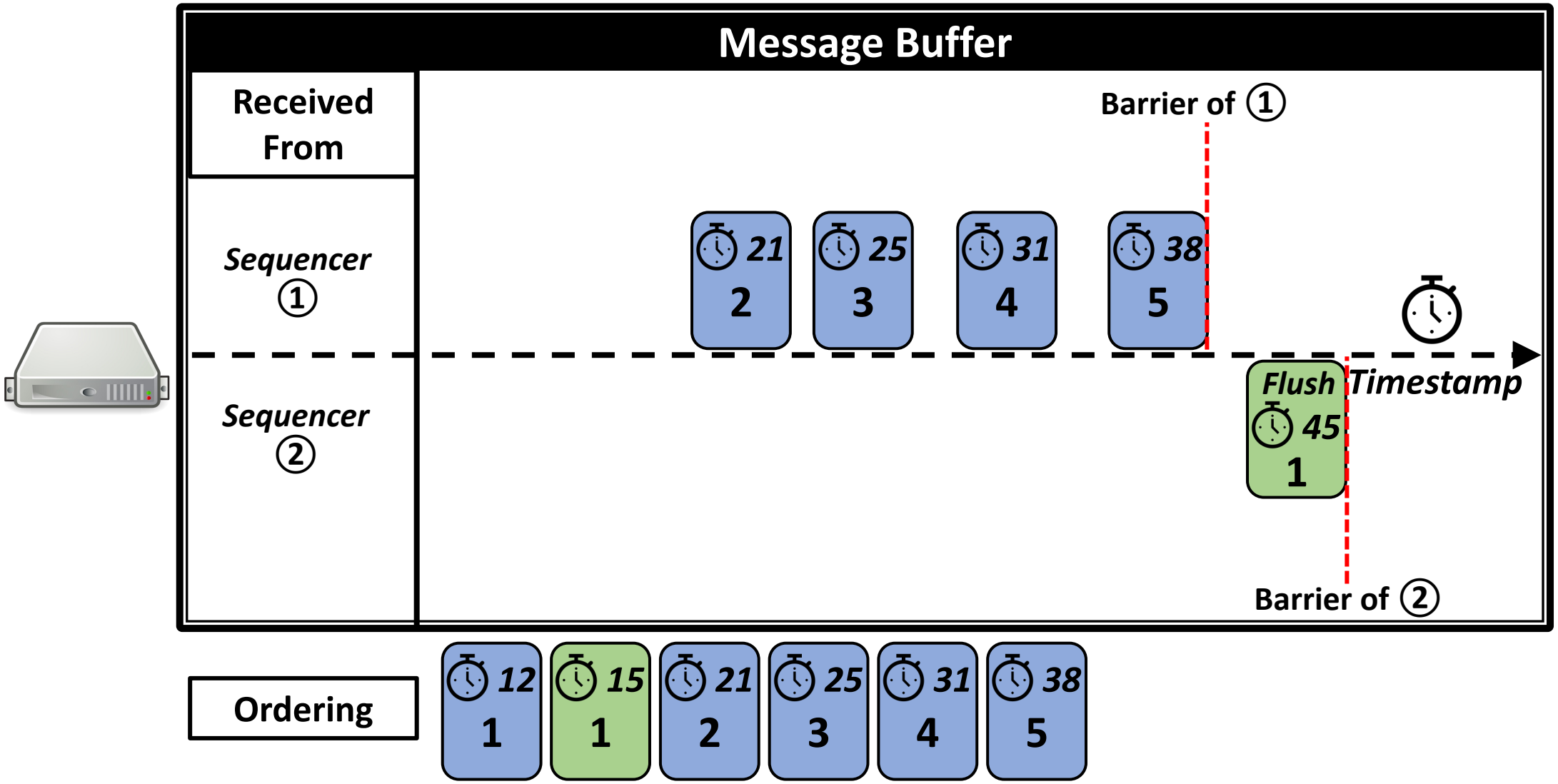
Does not receive any messages for a while



Flush Message from Sequencers



Flush Message from Sequencers



More Discussions

Flush message optimizations

- Receiver-side solicitation
- In-network aggregation

Adding or removing sequencers

Congestion-aware routing

Outline

1. Introduction
2. Hydra Network Primitive
3. Handling Network Anomalies
4. Evaluation

Low Latency through Network Load Balancing

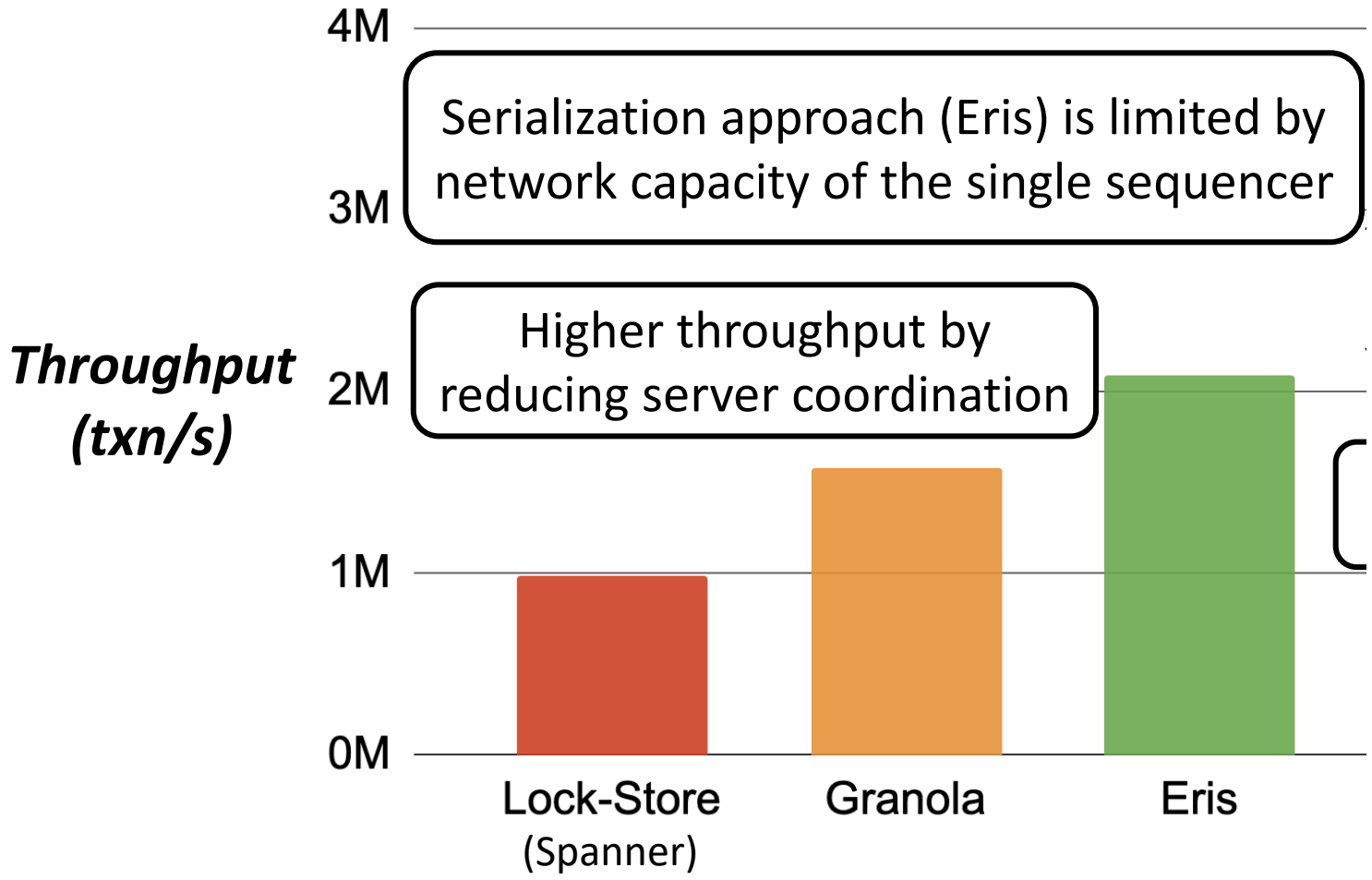
Transactional Database Performance

Experimental Setup

- **HydraTxn**: transactional key-value store using Hydra
- 15 physical shards (3 replicas per shard), virtual shards
- Limited sequencer switch capacity (details in the paper)
- Workload: YCSB+T *

* *Benchmarking cloud serving systems with YCSB* [SoCC '10]

Scales beyond a single sequencer



Spanner: Google's Globally-Distributed Database [OSDI '12]
Granola: Low-Overhead Distributed Transaction Coordination [ATC '12]
Eris: CoordinationFree Consistent Transactions Using In-Network Concurrency Control [SOSP '17]

More Evaluations

- Throughput of Hydra ***scales linearly with the number of sequencers***
- Hydra ***reduces sequencer failover time by 5x***
- Performance of Hydra is ***resilient to moderate levels of packet drops or clock skews*** across sequencers

  Do not affect system correctness
- Hydra ***significantly reduces flush message overhead*** using various optimizations

Summary

- Existing network ordering approaches pose serious drawbacks, due to ***in-network serialization***

- Hydra combines ***sequence number and physical clock*** of sequencers

- Result: ***serialization-free*** network ordering with significantly
  - ✅ ***Better network-level load balancing***
  - ✅ ***Higher sequencer scalability***
  - ✅ ***Faster sequencer failover***

Contact:
inhochoi@comp.nus.edu.sg