High Throughput Replication with Integrated Membership Management

Pedro Fouto, Nuno Preguiça, João Leitão

Presented By: Aman, Niranjan, Hrishabh

Adapted from authors USENIX ATC 2022 Slides

17 October, 2025

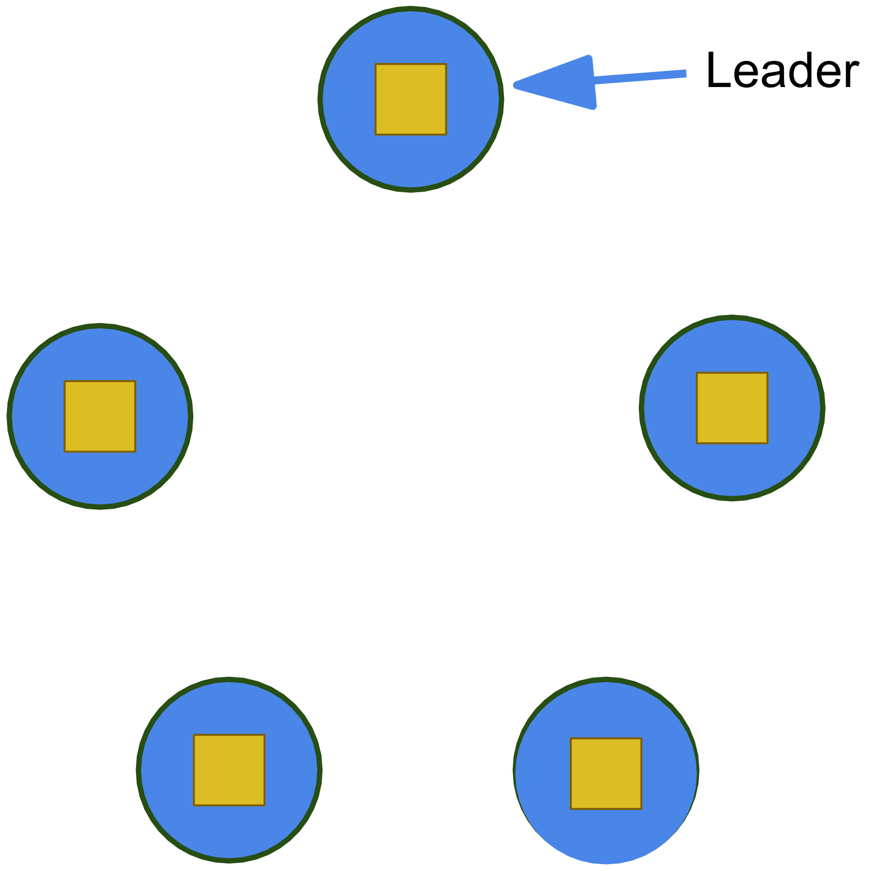
Motivation: Consensus and SMR

- Building blocks of numerous practical replication systems
- Their performance is critical
- Many alternatives have been designed
- Two very relevant ones:
    - (Multi-)Paxos
    - Chain Replication

Motivation: Consensus and SMR

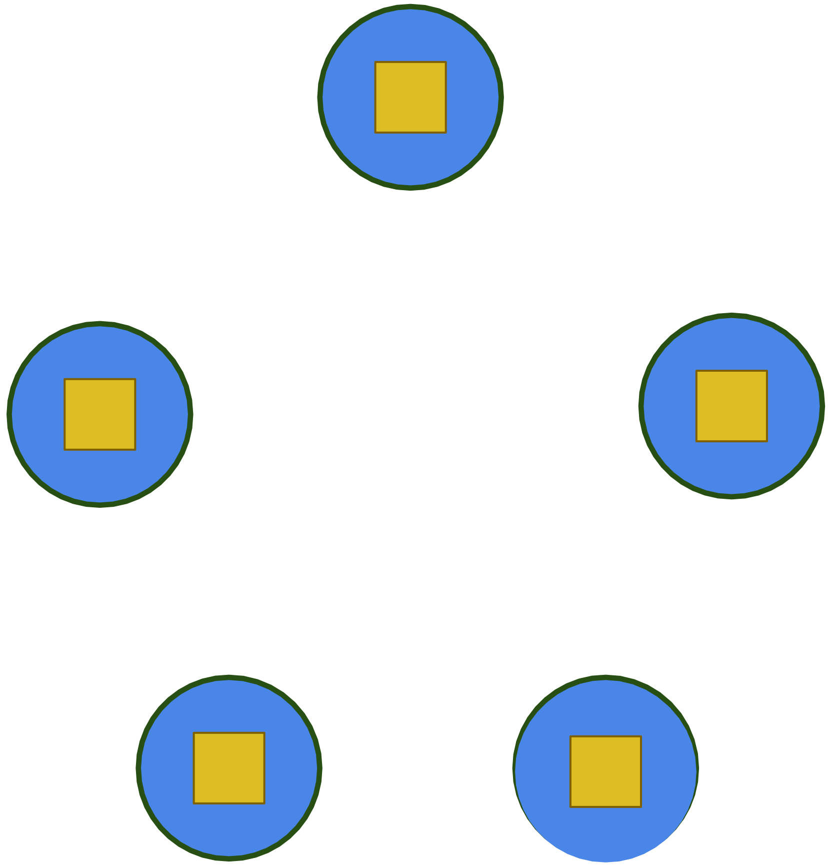
- Building blocks of numerous practical replication systems
- Their performance is critical
- Many alternatives have been designed
- Two very relevant ones:
  - **(Multi-)Paxos**
  - Chain Replication

Multi-Paxos



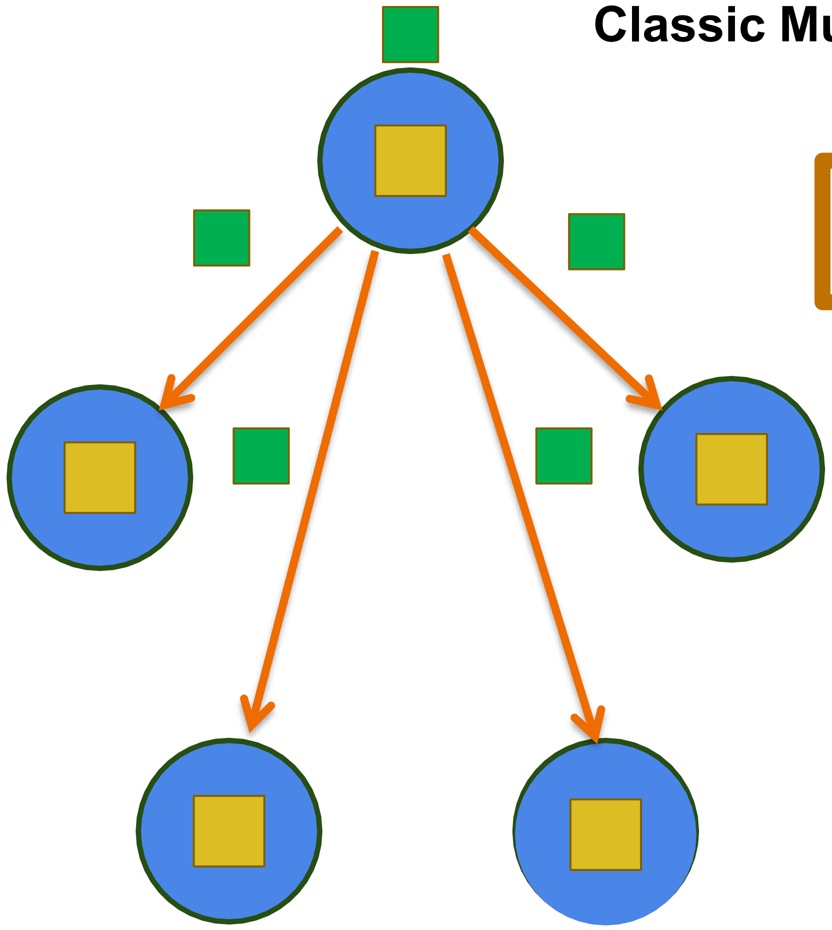
Multi-Paxos

Classic Multi-Paxos



Multi-Paxos

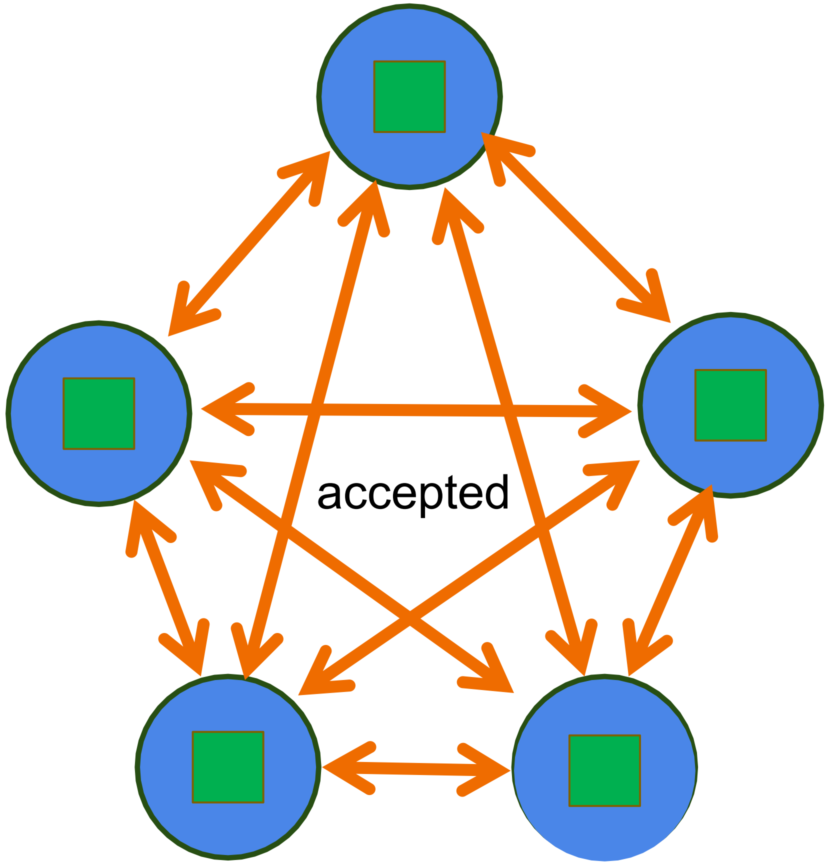
Classic Multi-Paxos



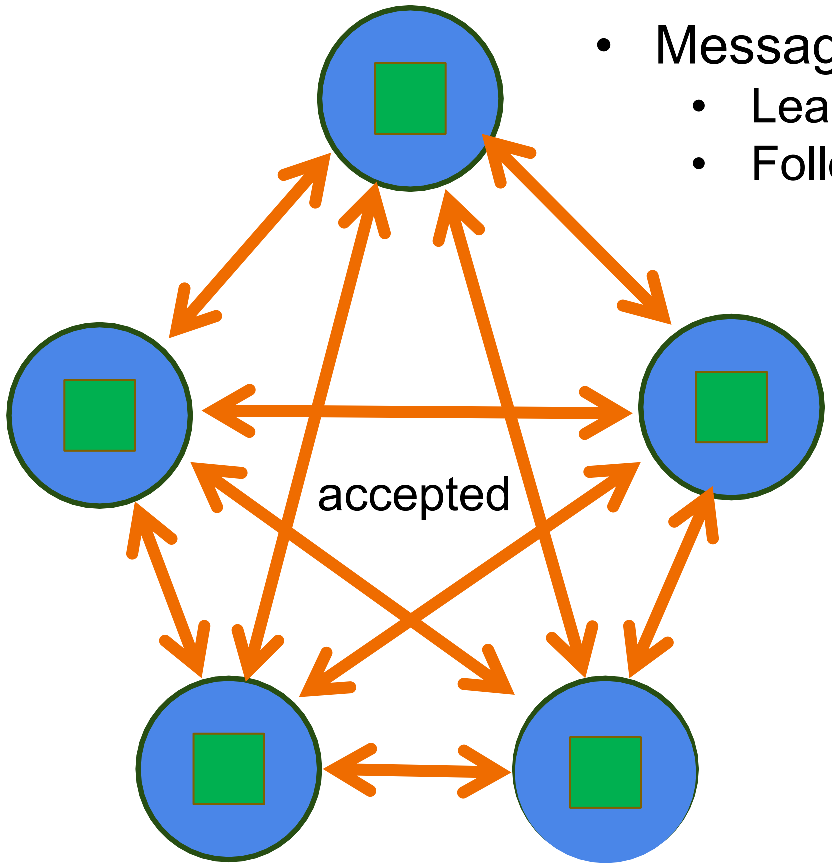
Skipping the first phase

Multi-Paxos

Classic Multi-Paxos



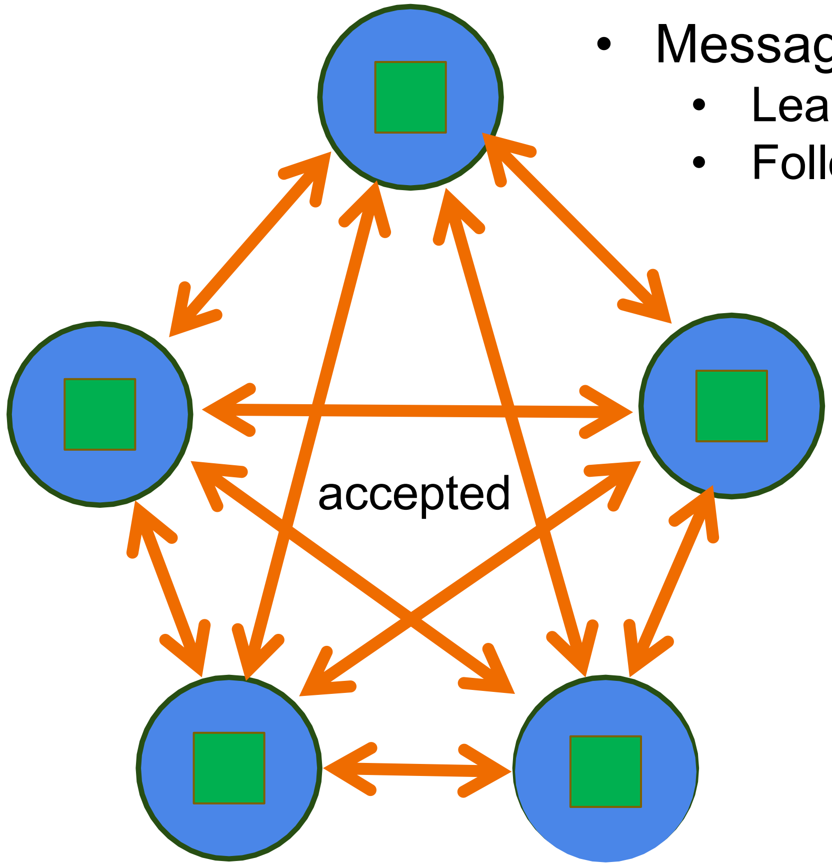
Multi-Paxos



Classic Multi-Paxos

- Message complexity:
  - Leader: O(n)
  - Followers: O(n)

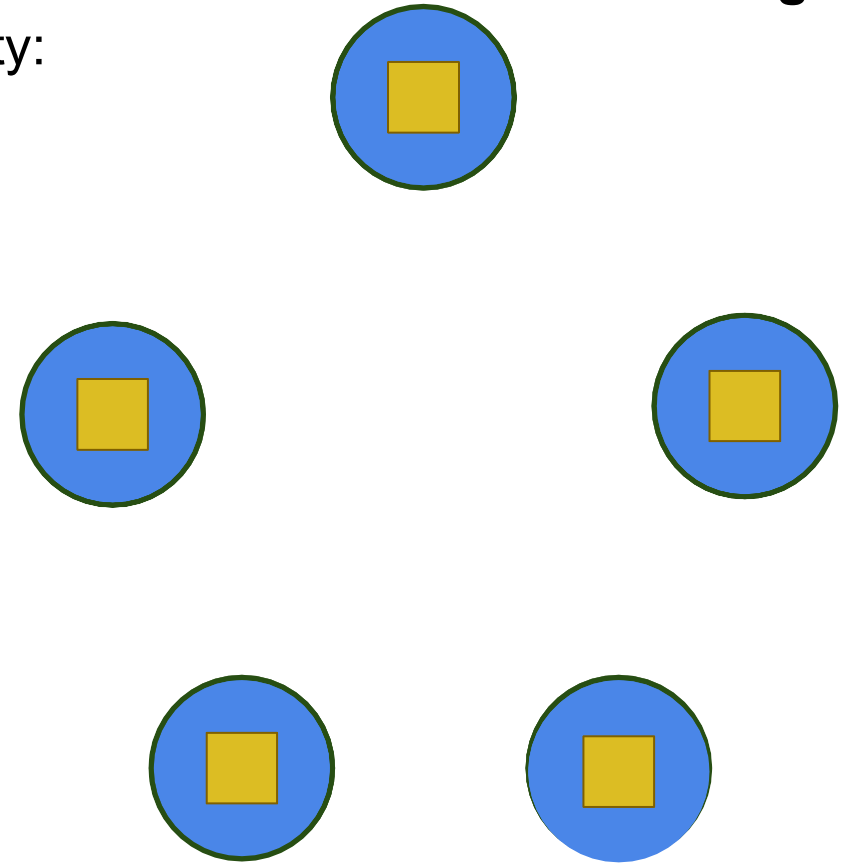
Multi-Paxos



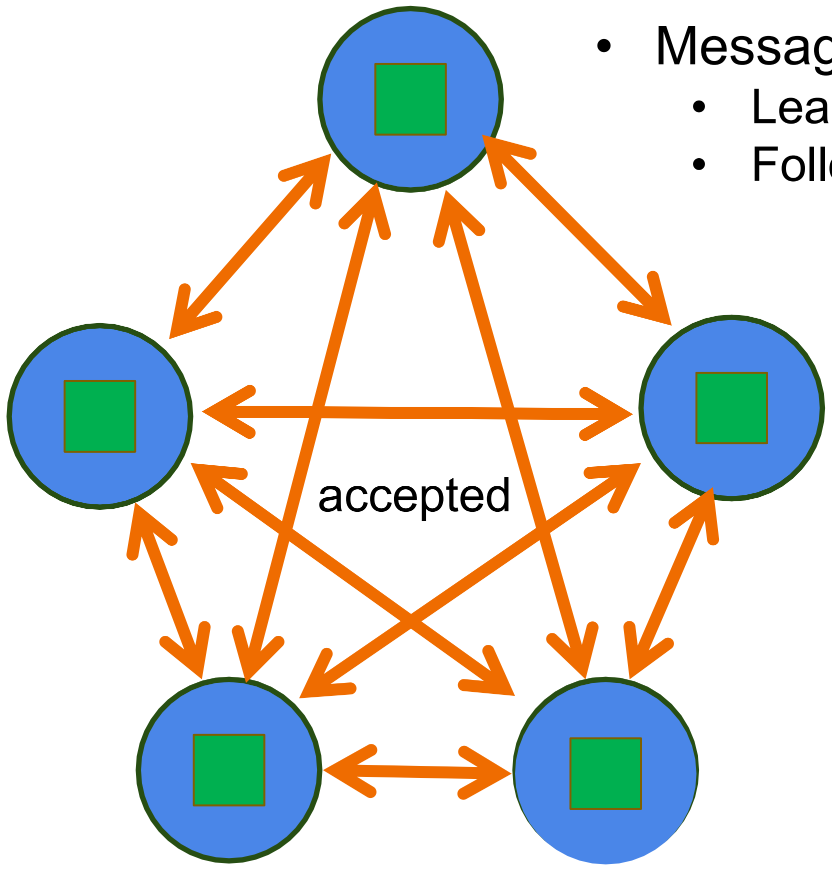
Classic Multi-Paxos

- Message complexity:
  - Leader: O(n)
  - Followers: O(n)

Distinguished Learner

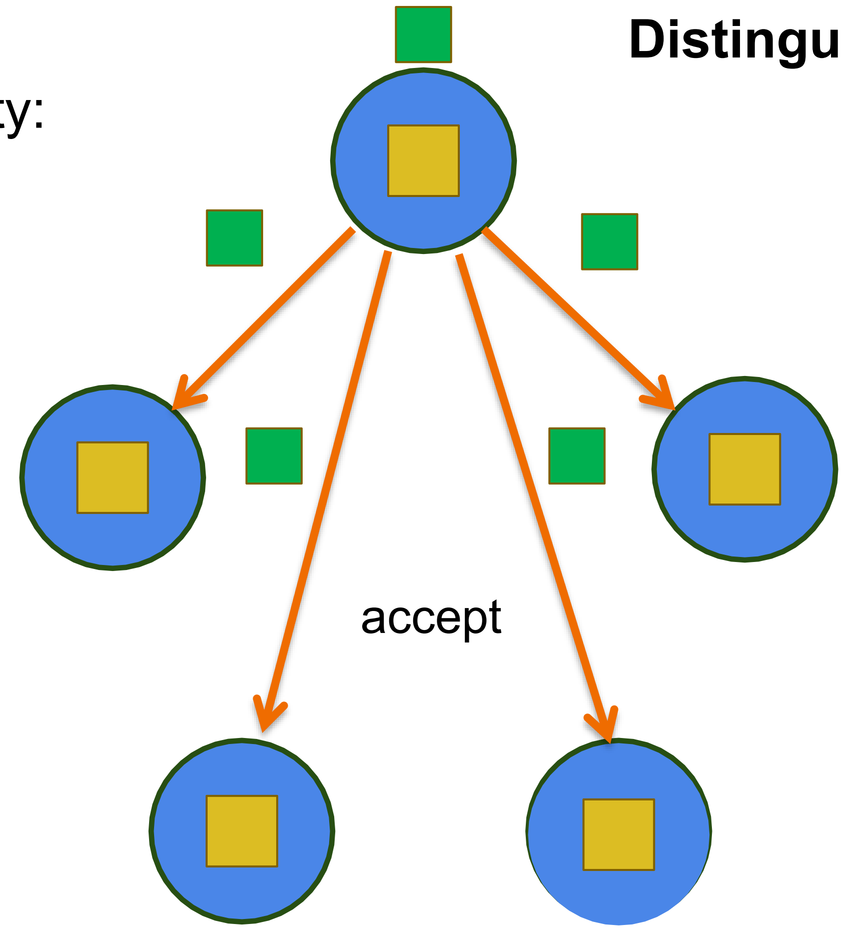


Multi-Paxos



Classic Multi-Paxos

- Message complexity:
  - Leader: O(n)
  - Followers: O(n)

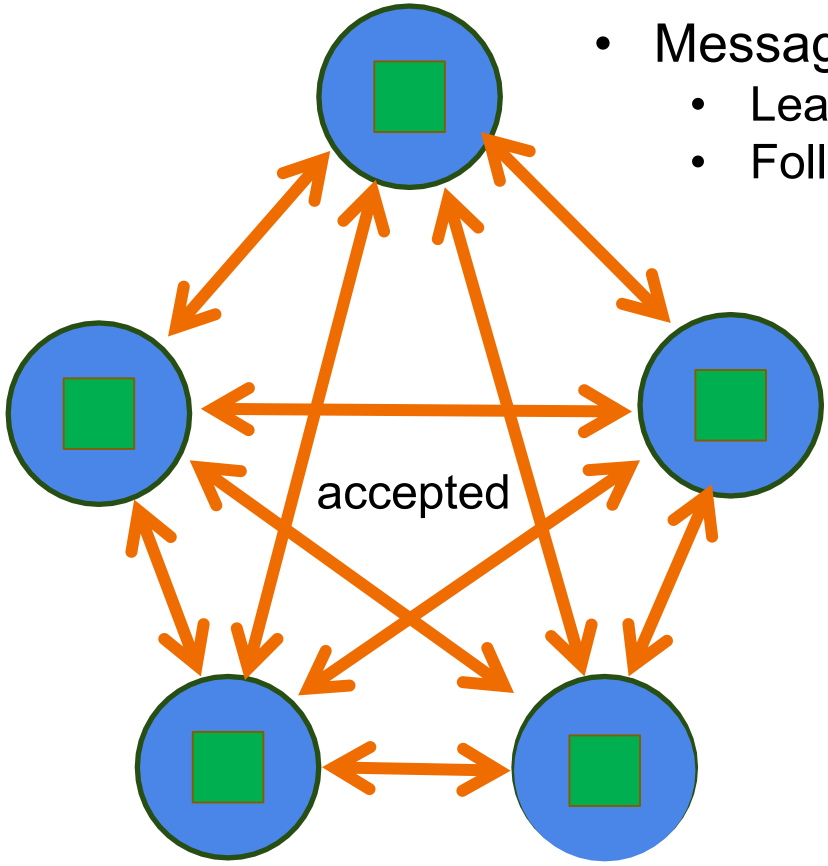


Distinguished Learner

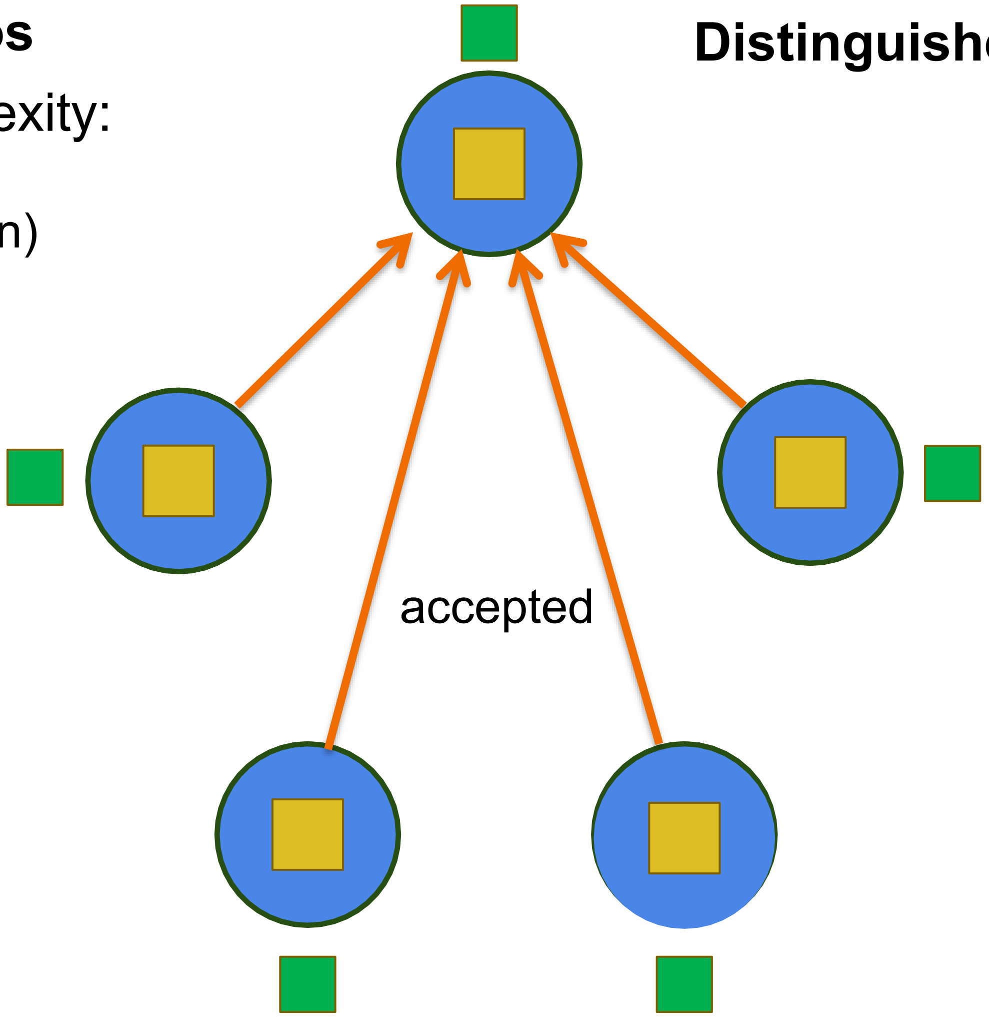
Multi-Paxos

Classic Multi-Paxos

- Message complexity:
  - Leader: O(n)
  - Followers: O(n)



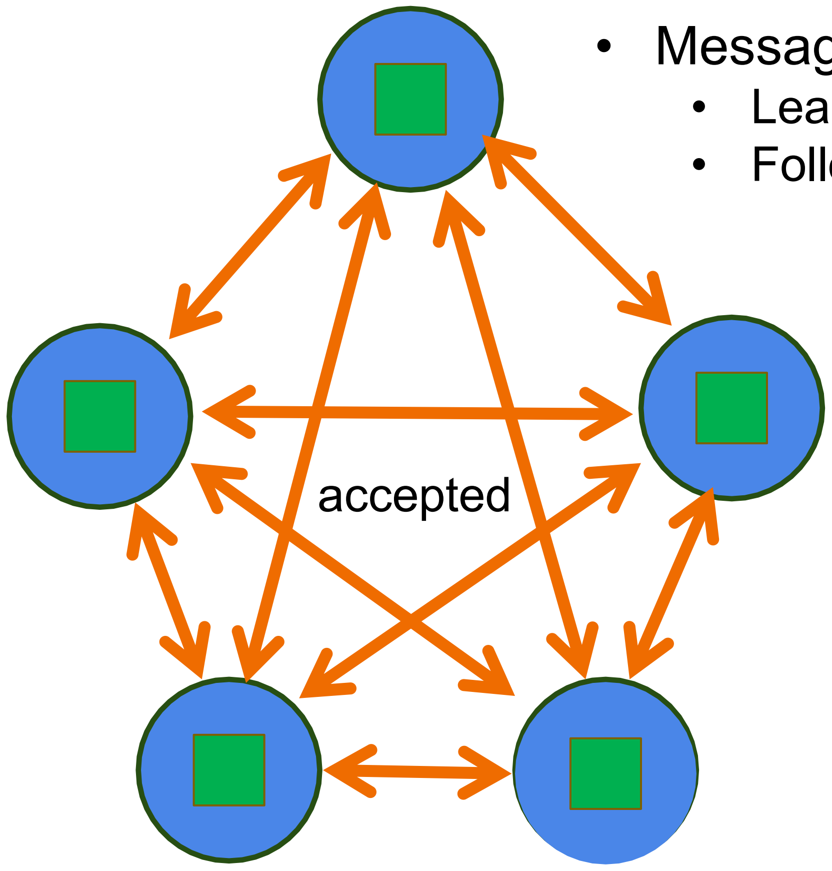
Distinguished Learner



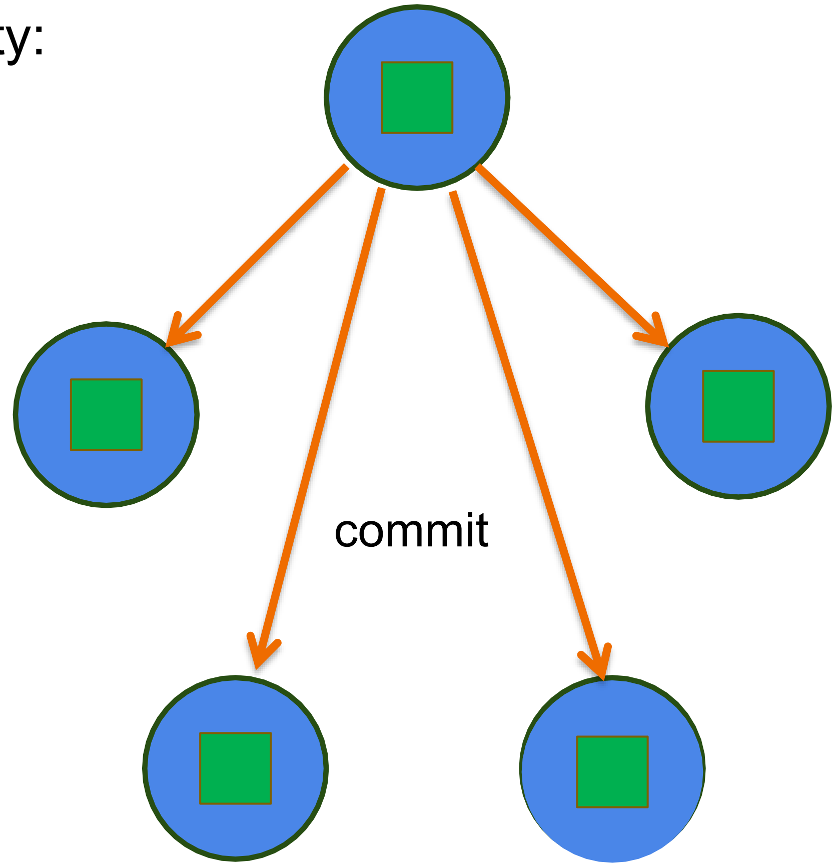
Multi-Paxos

Classic Multi-Paxos

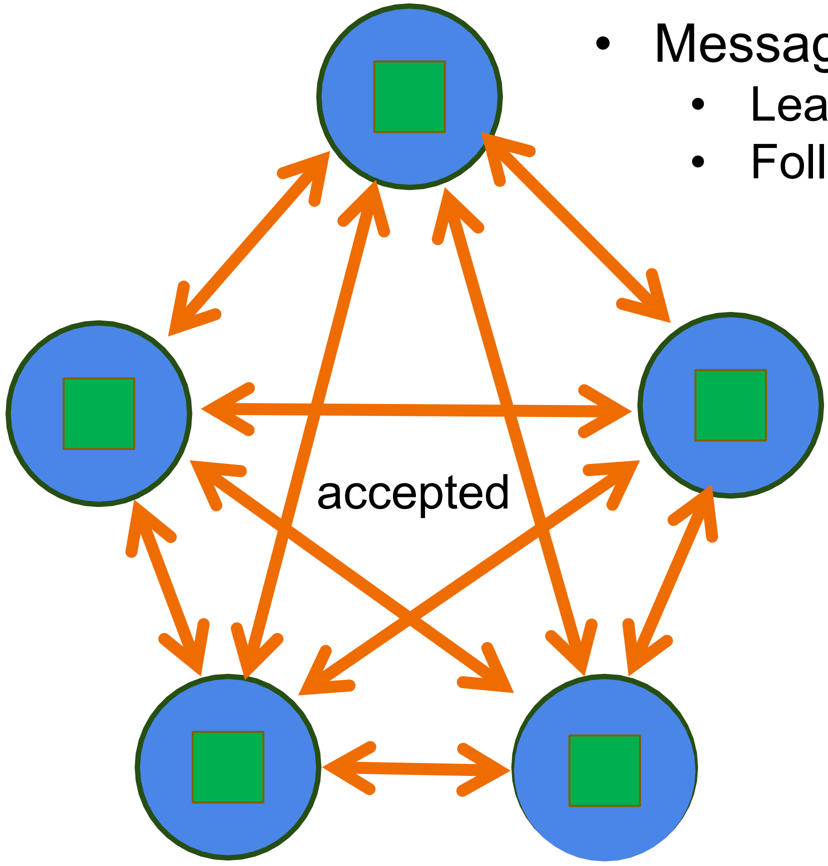
- Message complexity:
  - Leader: O(n)
  - Followers: O(n)



Distinguished Learner

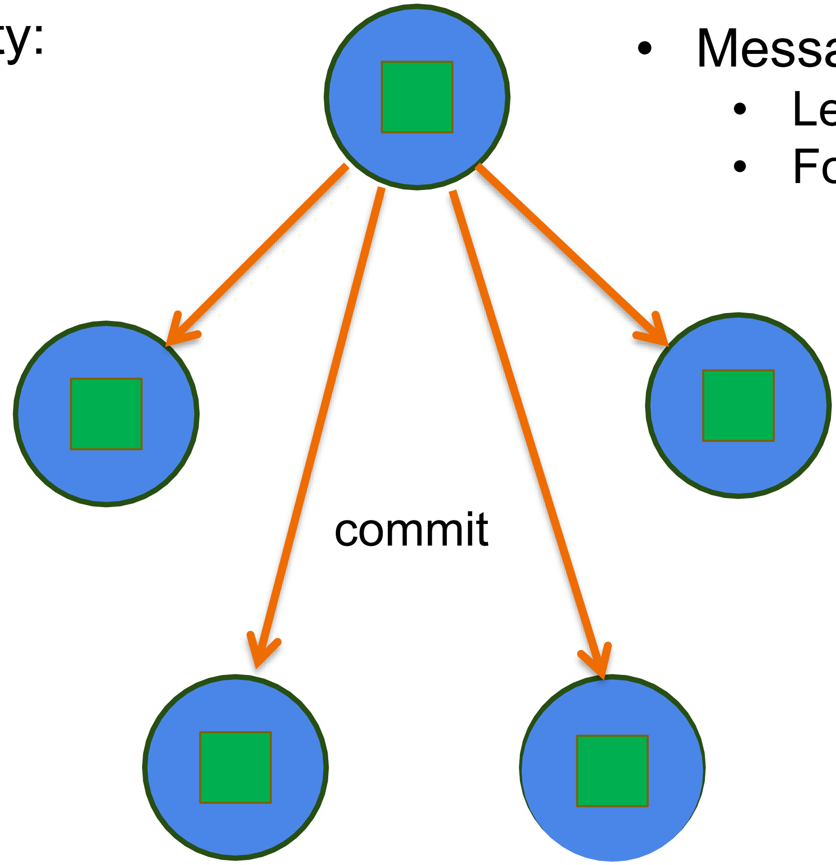


Multi-Paxos



Classic Multi-Paxos

- Message complexity:
  - Leader: O(n)
  - Followers: O(n)



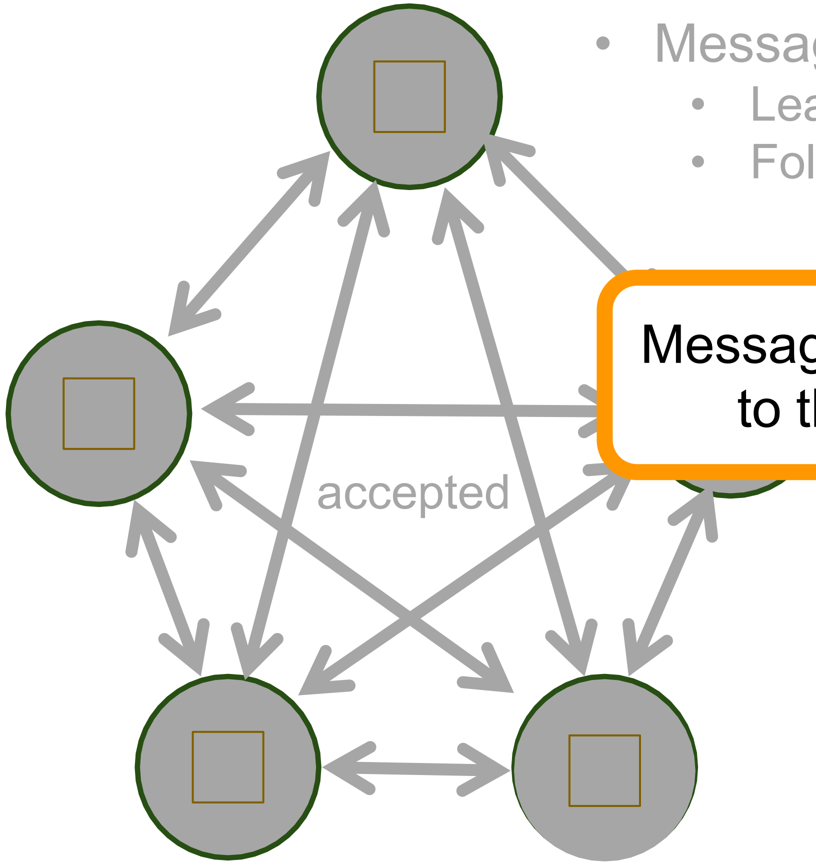
Distinguished Learner

- Message complexity:
  - Leader: O(n)
  - Followers: O(1)

Multi-Paxos

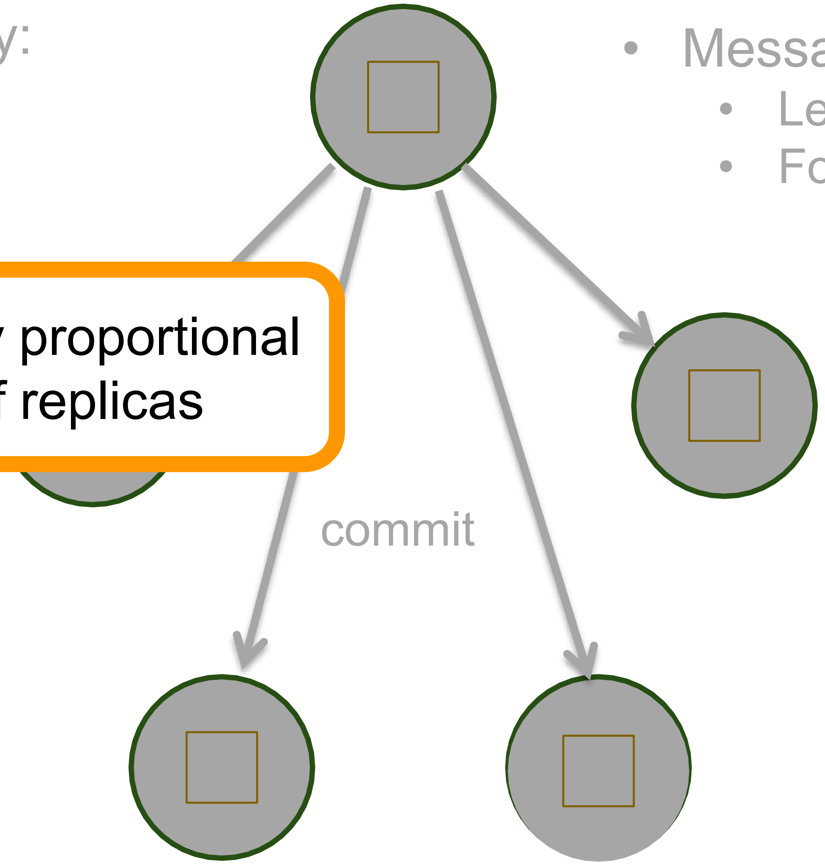
Classic Multi-Paxos

- Message complexity:
  - Leader: O(n)
  - Followers: O(n)



Distinguished Learner

- Message complexity:
  - Leader: O(n)
  - Followers: O(1)



Message complexity proportional to the number of replicas

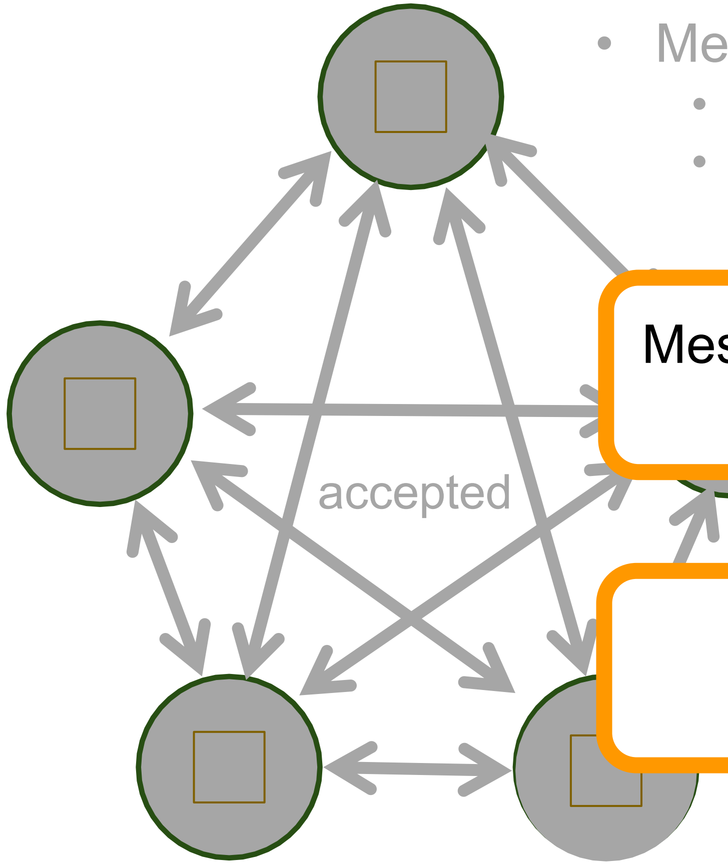
Multi-Paxos

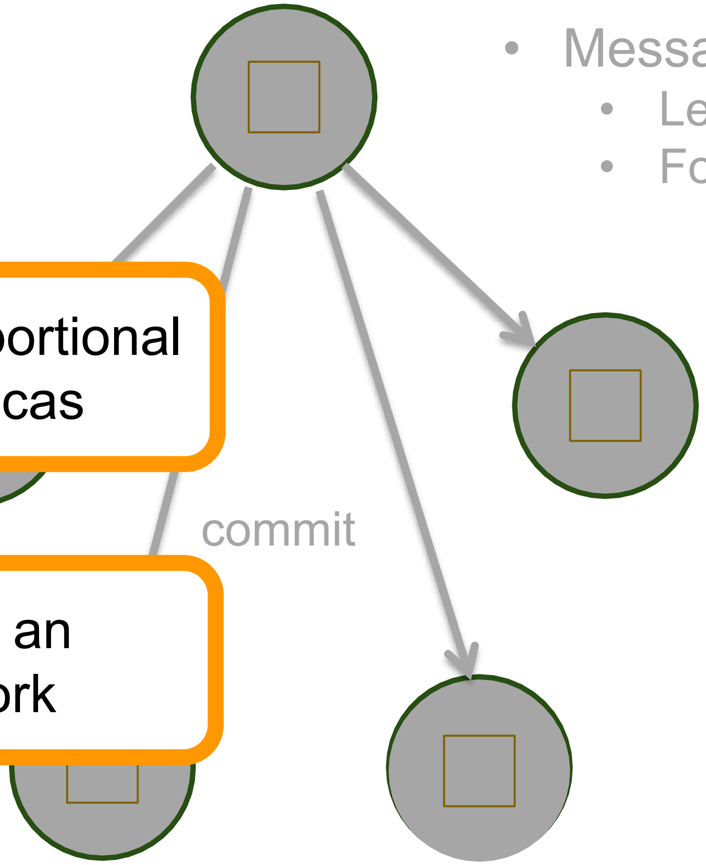
Classic Multi-Paxos

- Message complexity:
  - Leader: O(n)
  - Followers: O(n)

Distinguished Learner

- Message complexity:
  - Leader: O(n)
  - Followers: O(1)





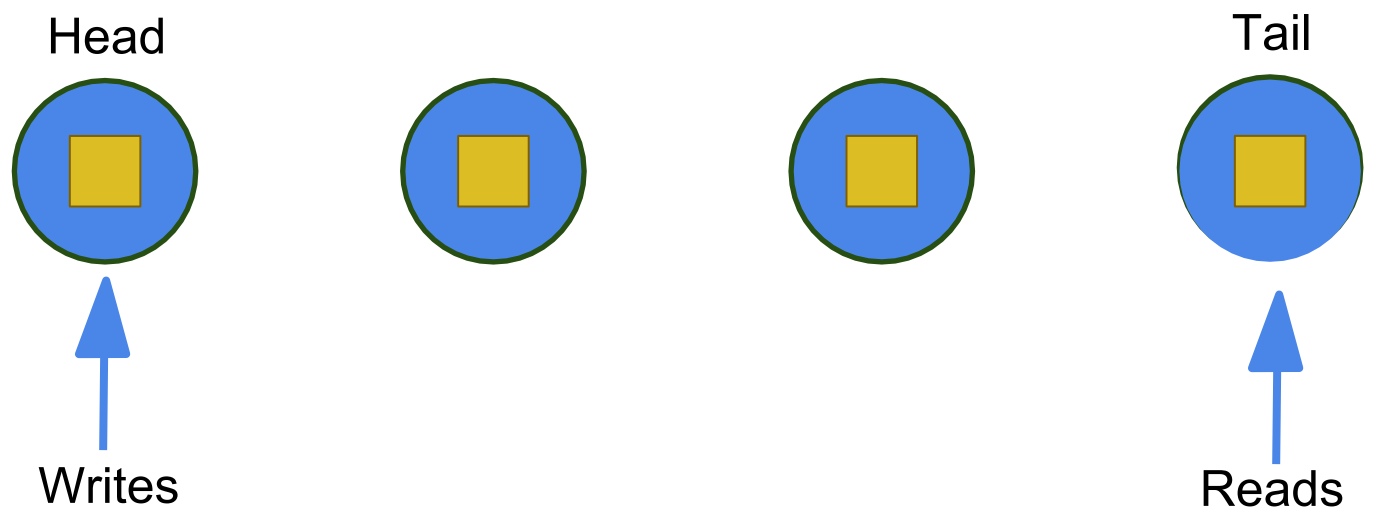
Message complexity proportional to the number of replicas

Guarantees safety in an asynchronous network

Motivation: Consensus and SMR

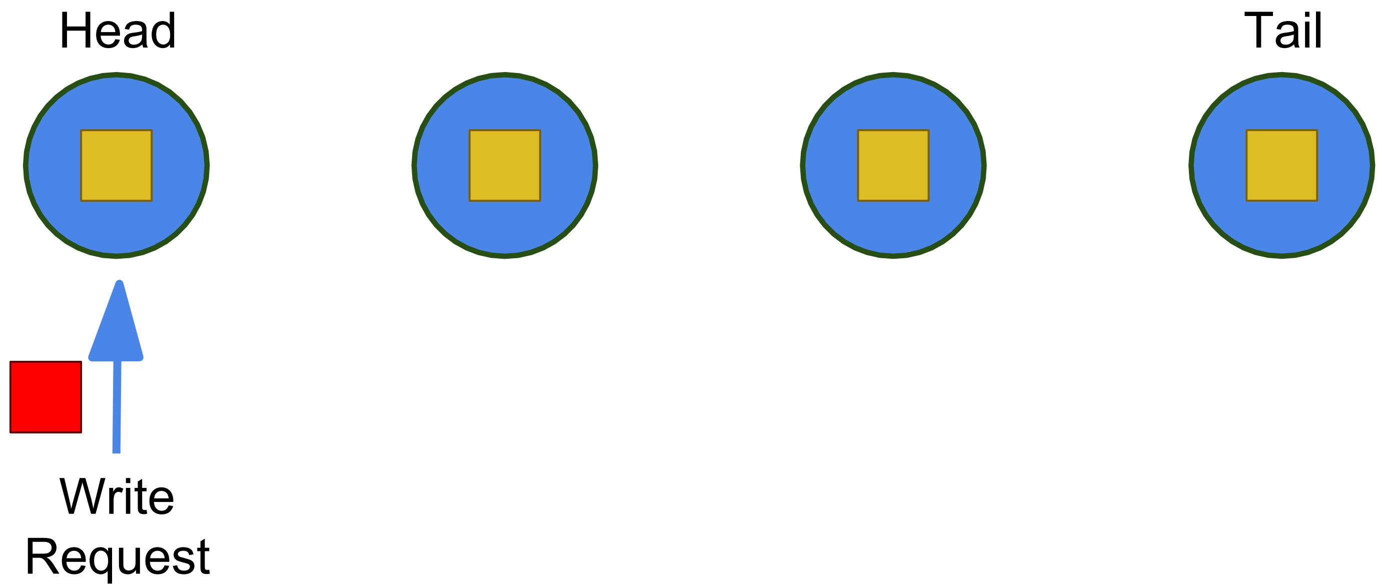
- Building blocks of numerous practical replication systems
- Their performance is critical
- Many alternatives have been designed
- Two very relevant ones:
  - (Multi-)Paxos
  - **Chain Replication**

Chain Replication (CR)

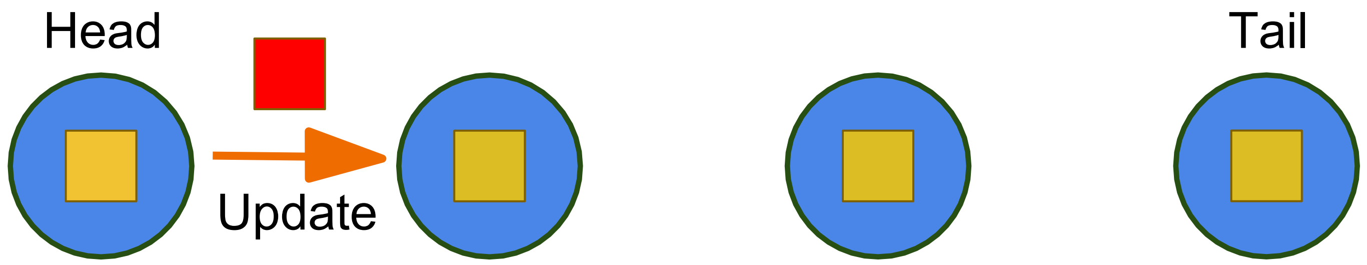


🟨 Any object.
Eg: Key-Value Store: X = 1

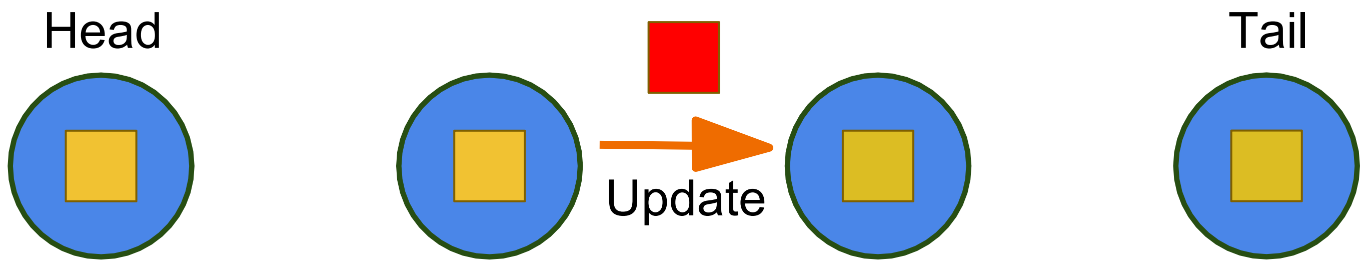
Chain Replication (CR)



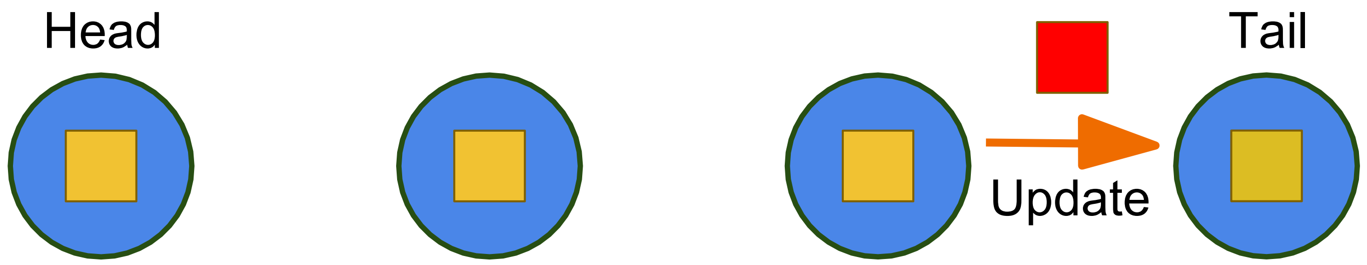
Chain Replication (CR)



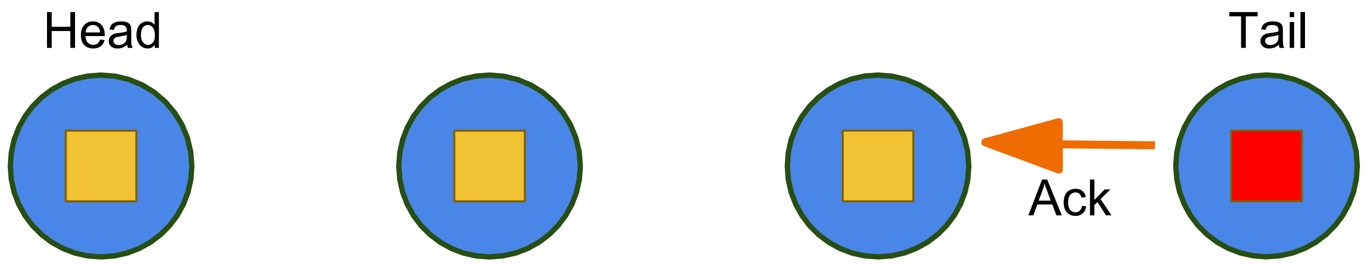
Chain Replication (CR)



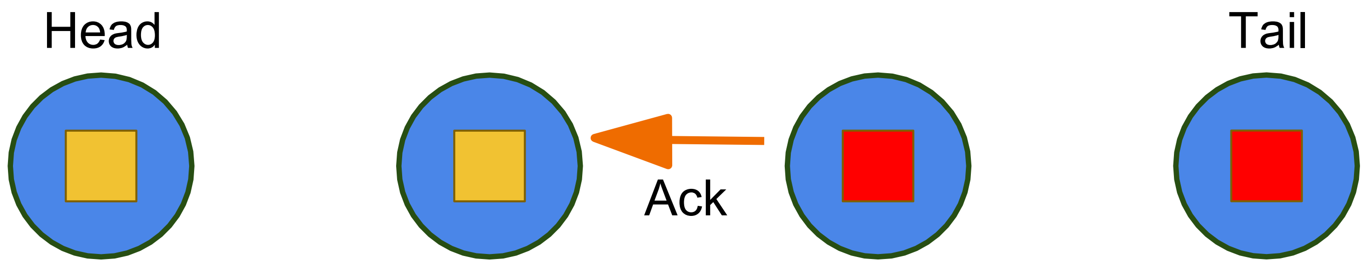
Chain Replication (CR)



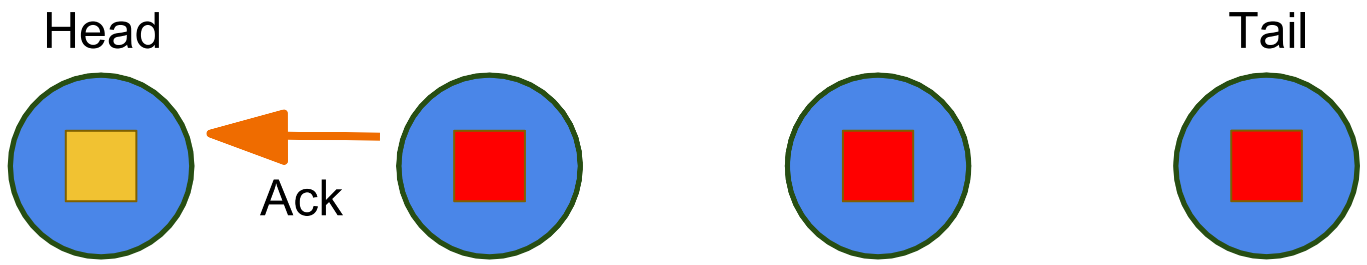
Chain Replication (CR)



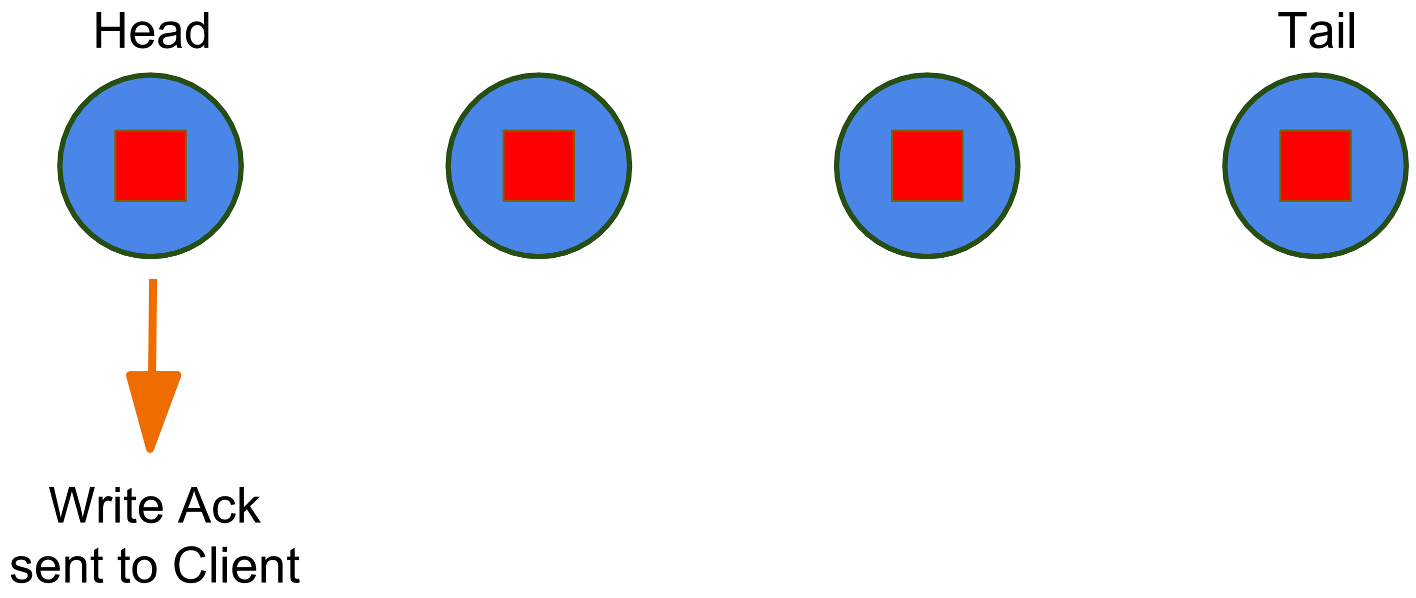
Chain Replication (CR)



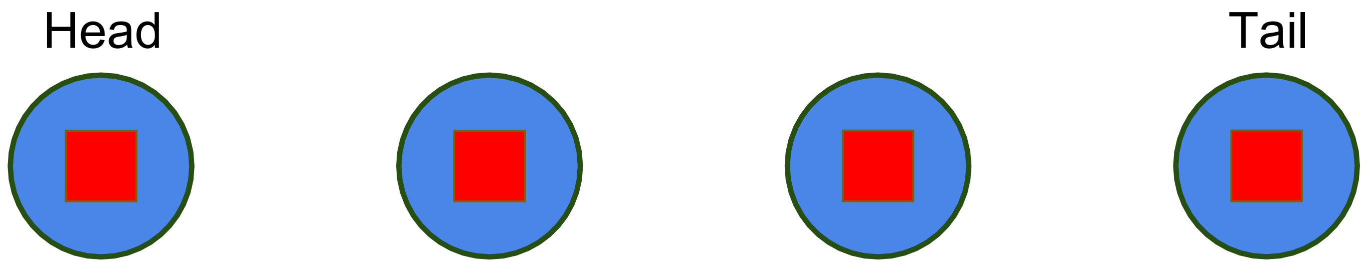
Chain Replication (CR)



Chain Replication (CR)

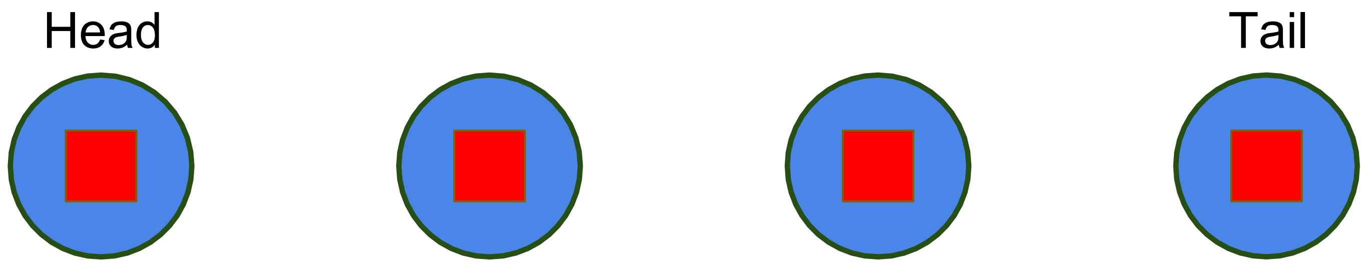


Chain Replication (CR)



- Message complexity:
    - All replicas: O(1)

Chain Replication (CR)



- Message complexity:
  - All replicas: O(1)

However, it has its limitations…

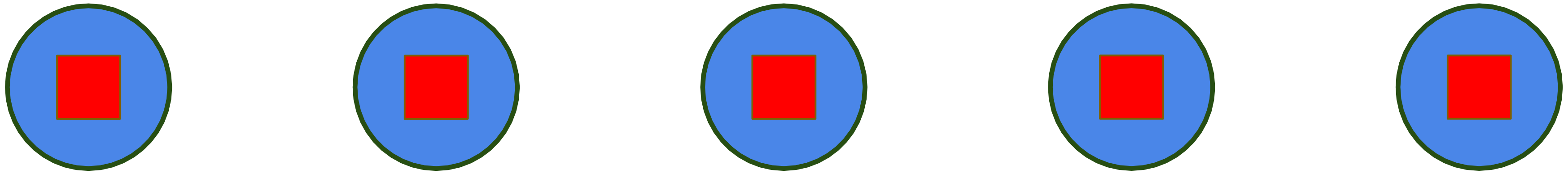
Motivation: Consensus and SMR

- Building blocks of numerous practical replication systems
- Their performance is critical
- Many alternatives have been designed
- Two very relevant ones:
    - (Multi-)Paxos
    - Chain Replication
- Two aspects are often overlooked:
    - **Performant linearizable reads**
    - Membership management and reconfiguration

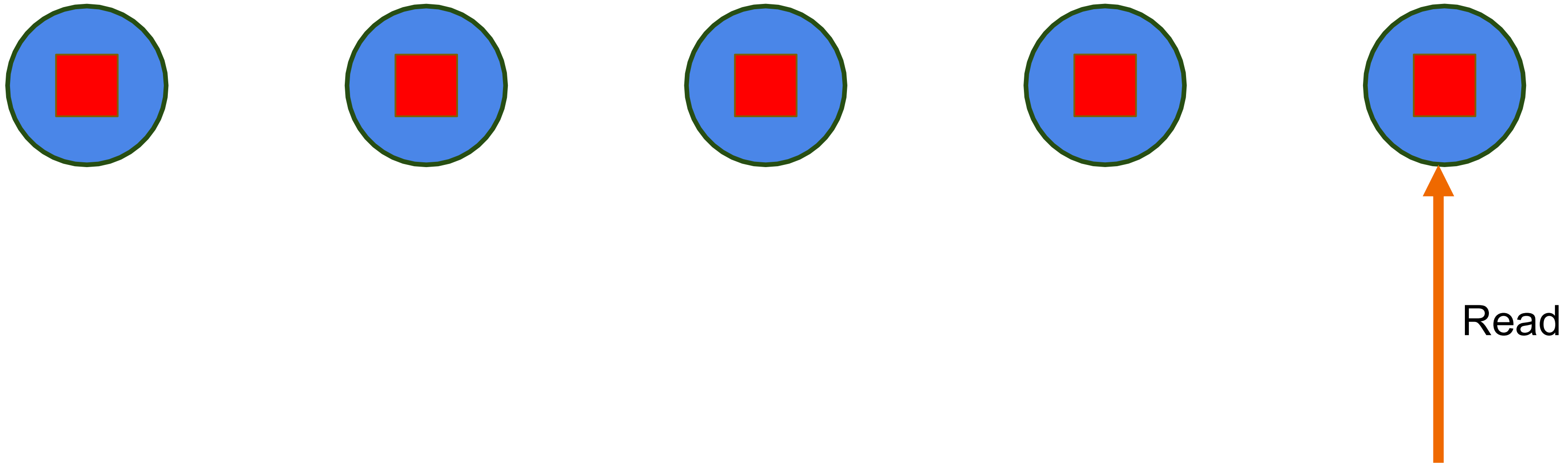
Motivation: Linearizable Reads

- Some existing solutions assume a synchronous model (e.g. Chain Replication)
- Dealing with asynchrony is complicated. One can:
    - Relax consistency (e.g. ZooKeeper)
    - Add extra (costly) steps to write operations
    - Synchronize with other replicas when reading

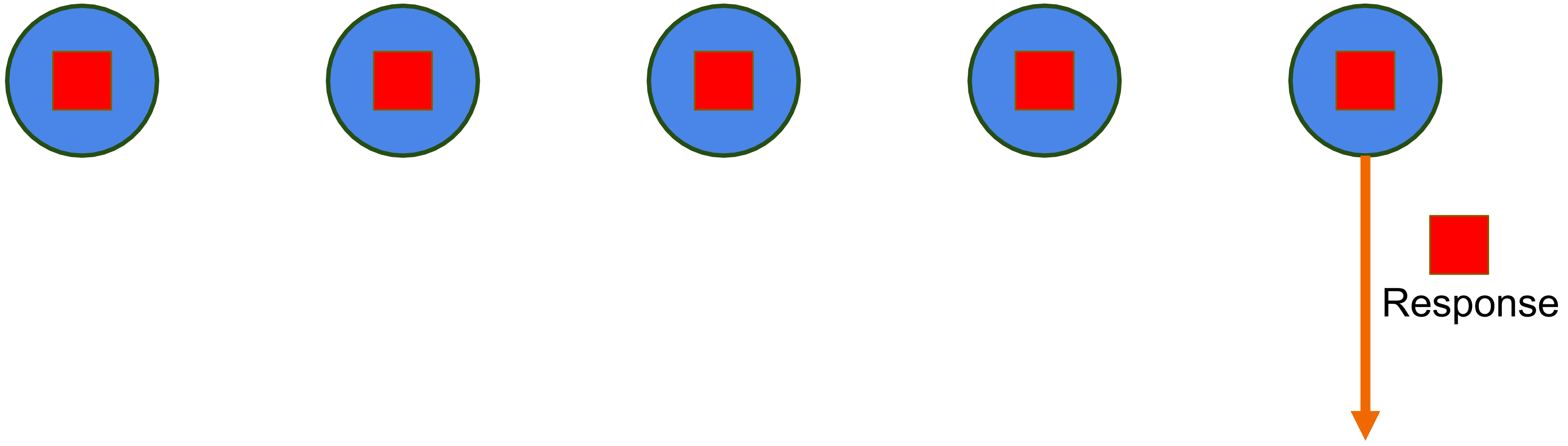
Motivation: Linearizable Reads



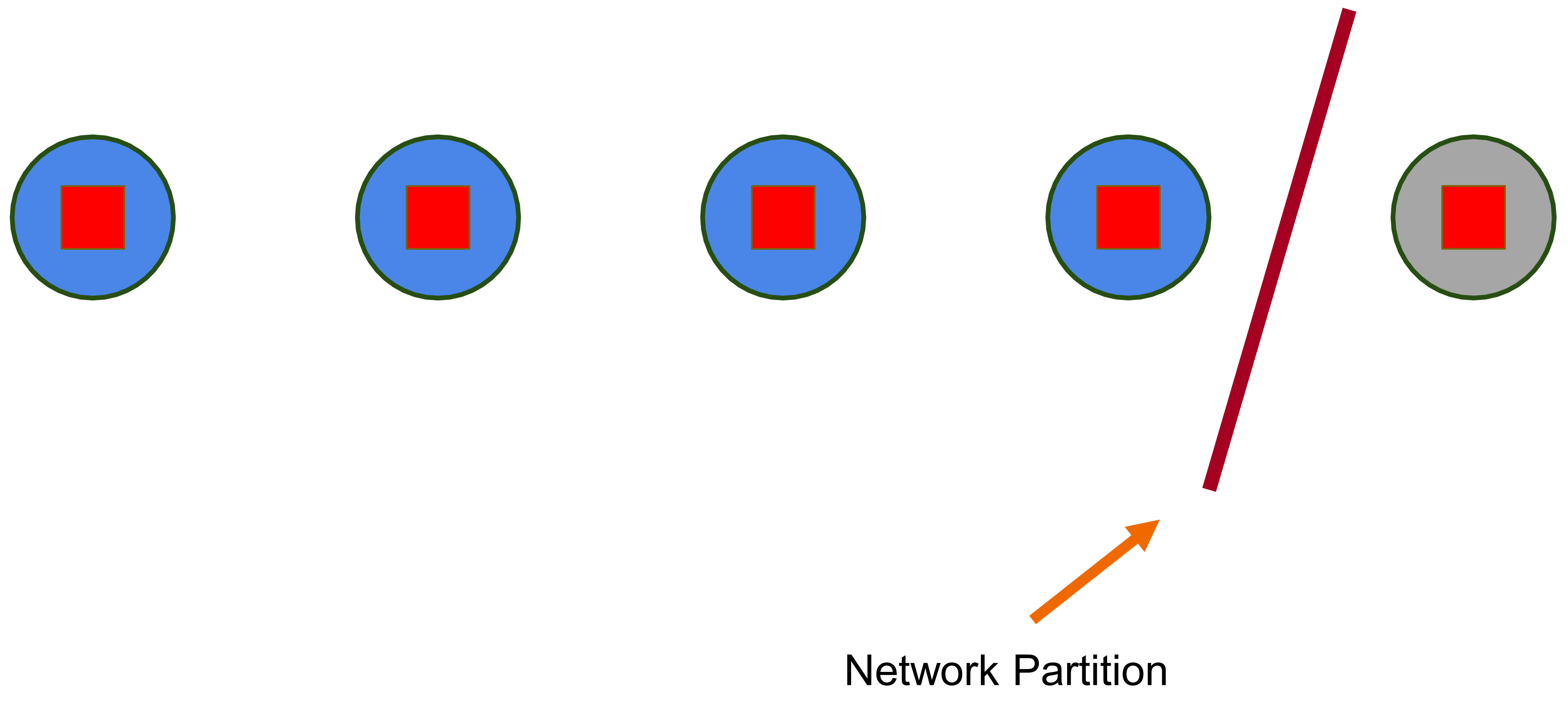
Motivation: Linearizable Reads



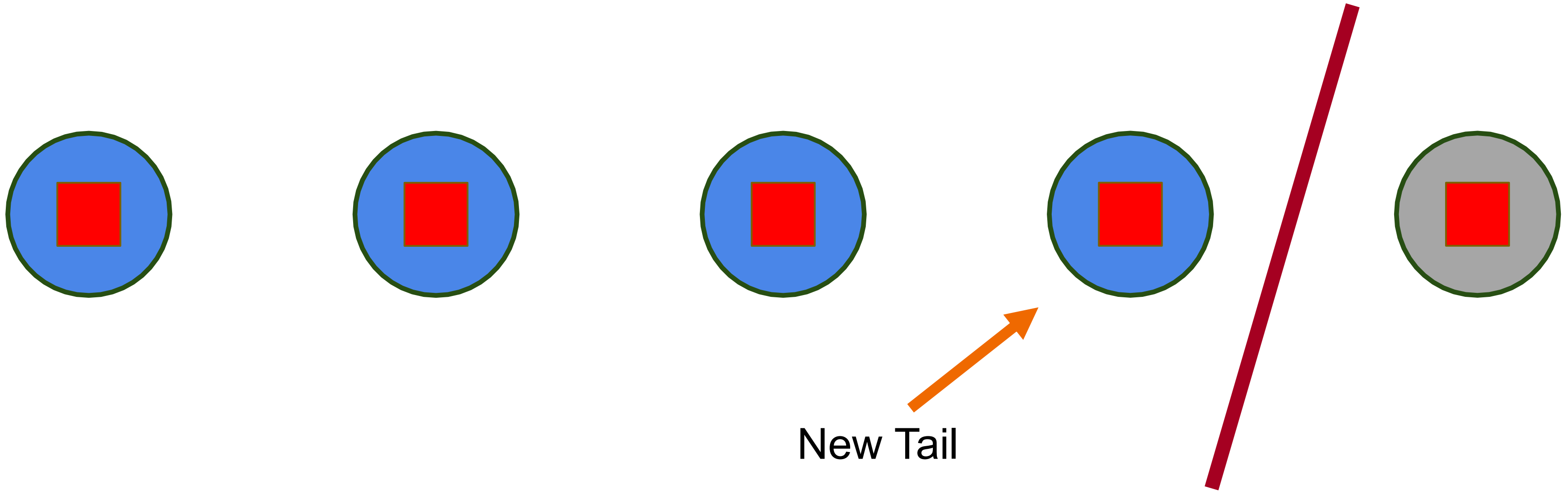
Motivation: Linearizable Reads



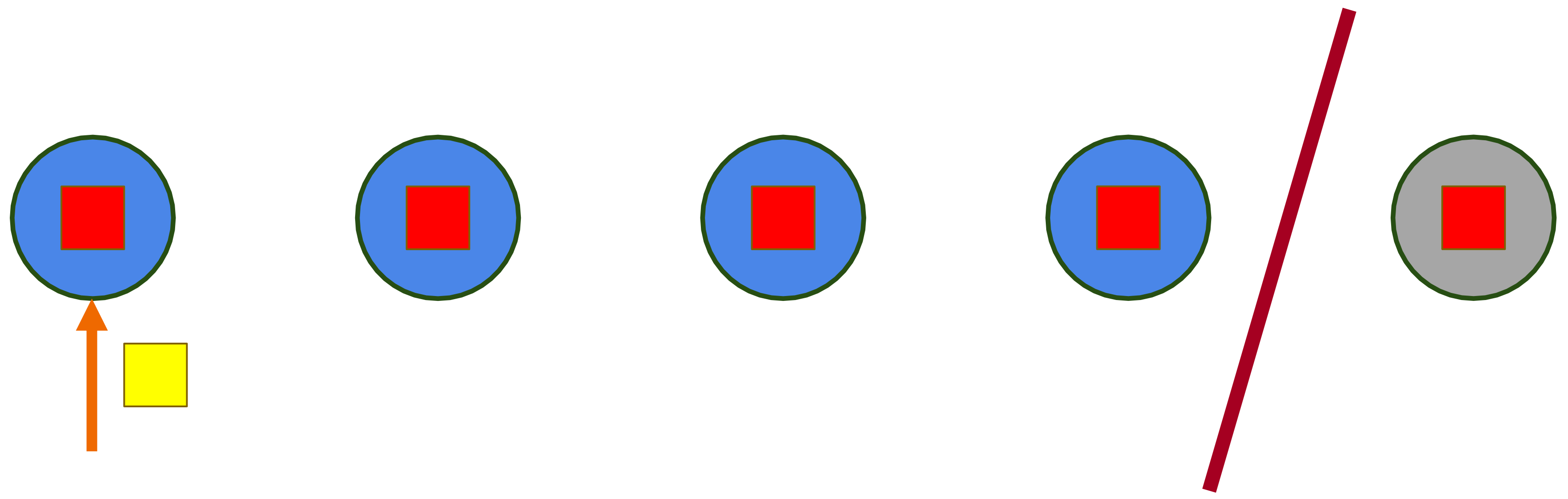
Motivation: Linearizable Reads



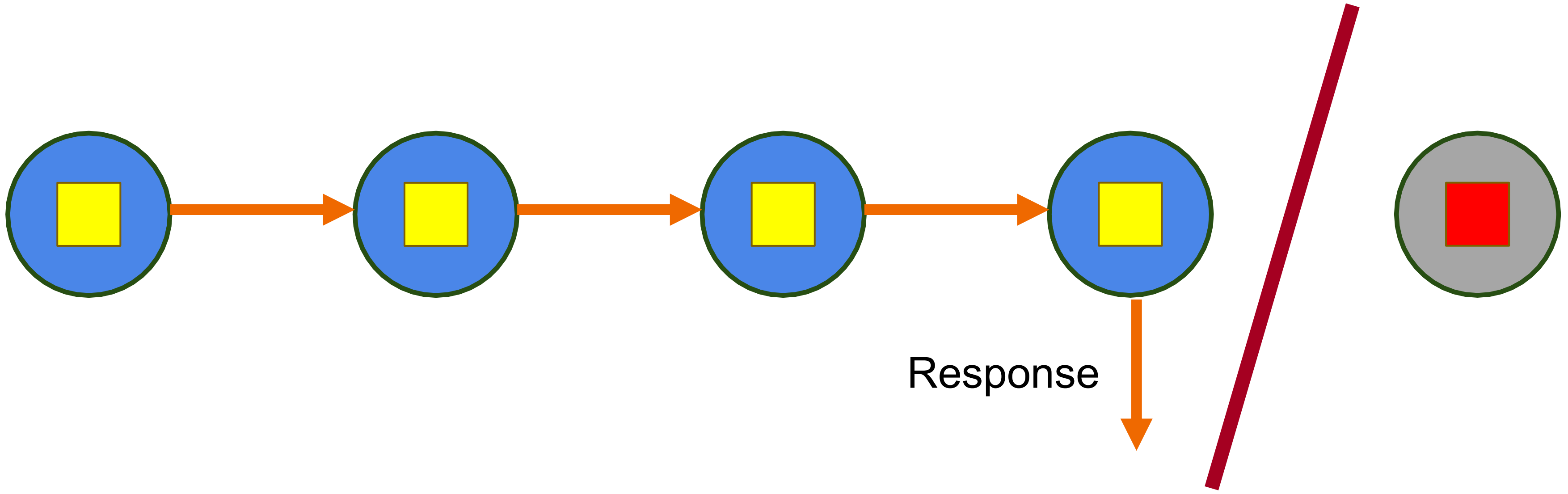
Motivation: Linearizable Reads



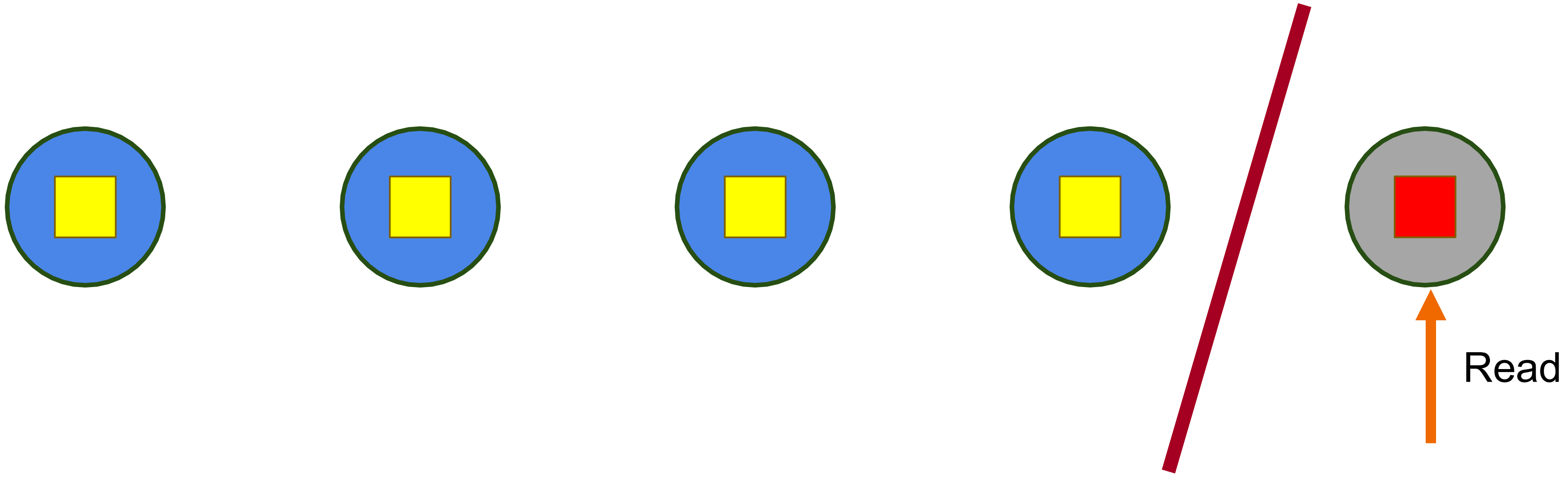
Motivation: Linearizable Reads



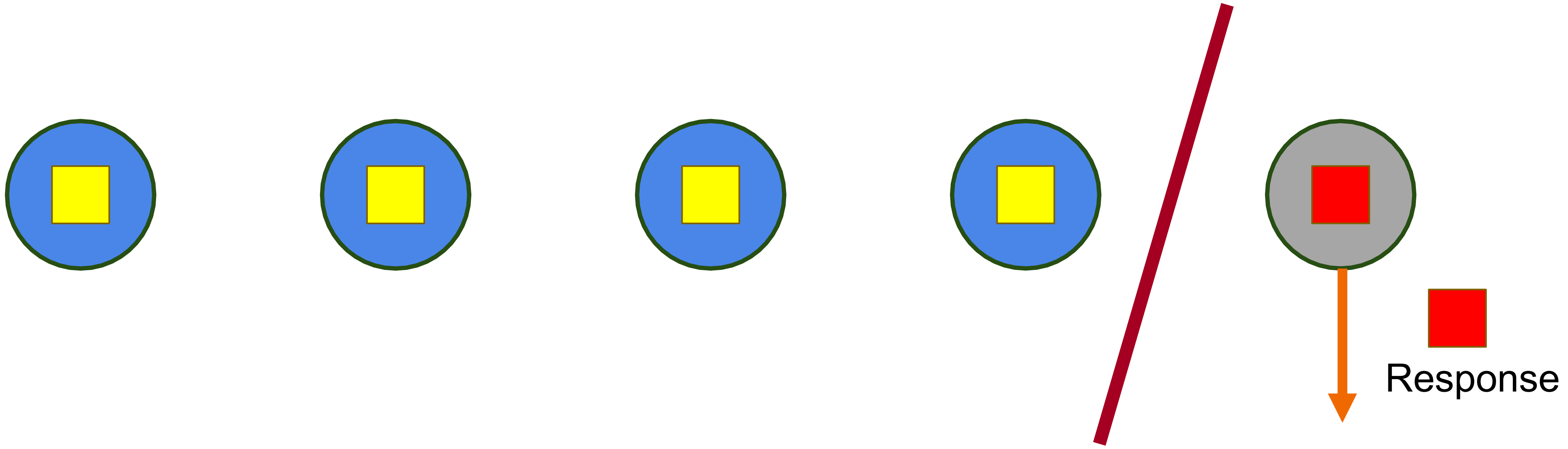
Motivation: Linearizable Reads



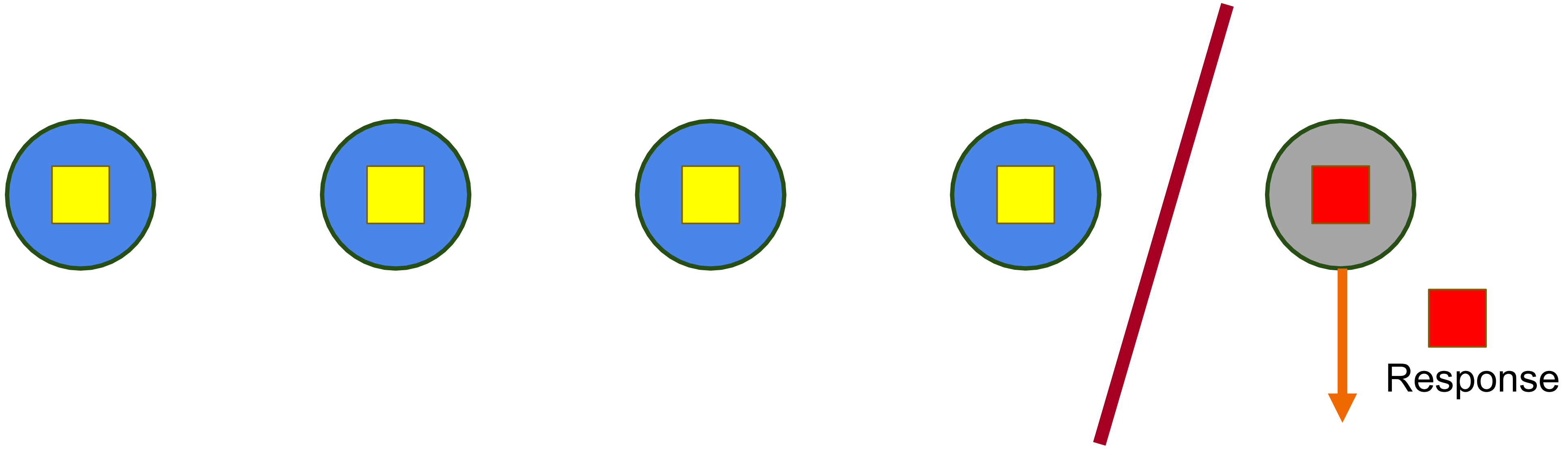
Motivation: Linearizable Reads



Motivation: Linearizable Reads



Motivation: Linearizable Reads



Breaks linearizability in an asynchronous network model

Motivation: Linearizable Reads

- Some existing solutions assume a synchronous model (e.g. Chain Replication)

- Dealing with asynchrony is complicated. One can:
  - **Relax consistency** (e.g. ZooKeeper)
  - Add **extra (costly) steps** to write operations
  - **Synchronize** with other replicas when reading

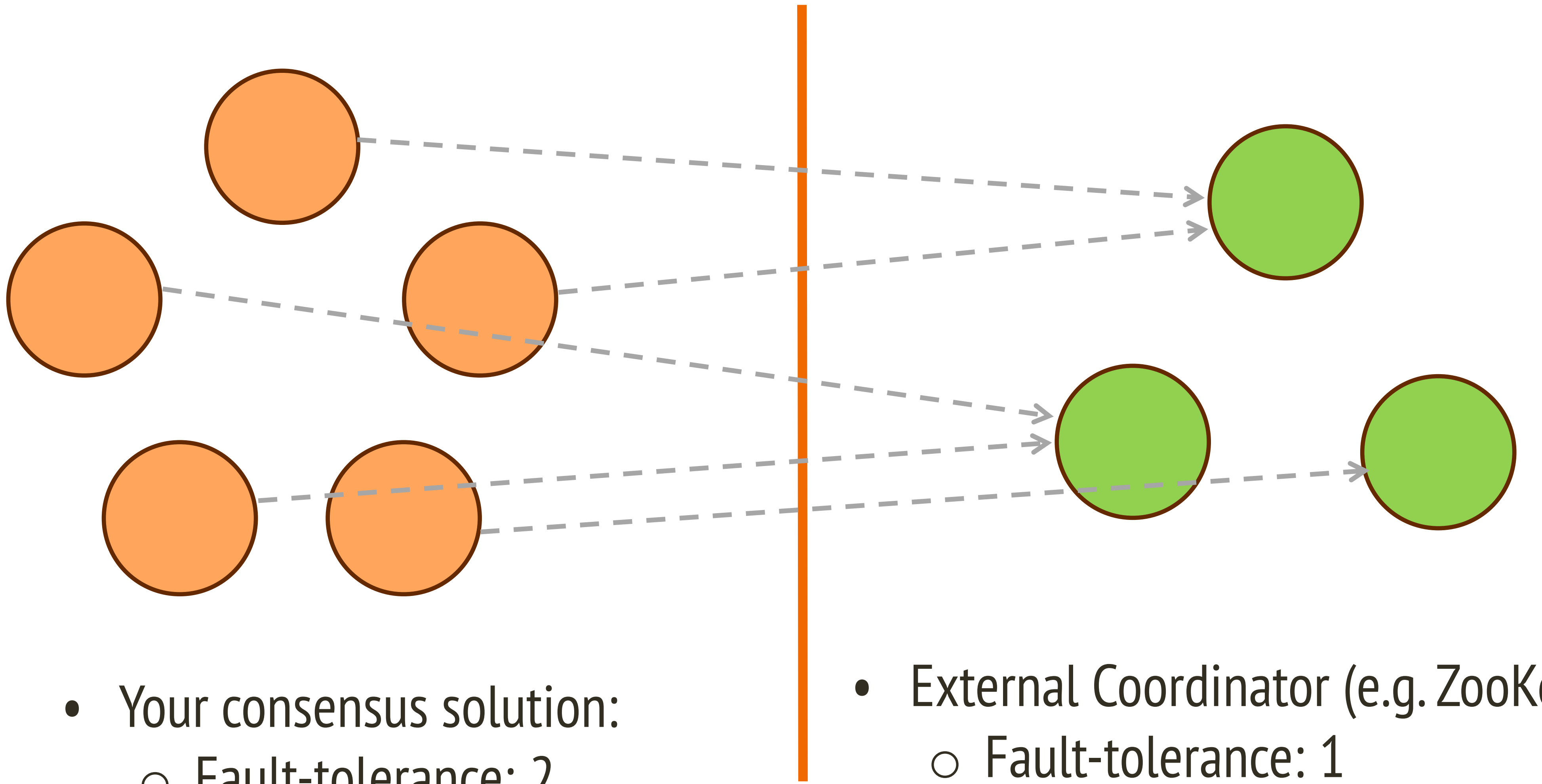
Motivation: Consensus and SMR

- Building blocks of numerous practical replication systems
- Their performance is critical
- Many alternatives have been designed
- Two very relevant ones:
  - (Multi-)Paxos
  - Chain Replication
- Two aspects are often overlooked:
  - Performant linearizable reads
  - **Membership management and reconfiguration**

Motivation : Membership Management

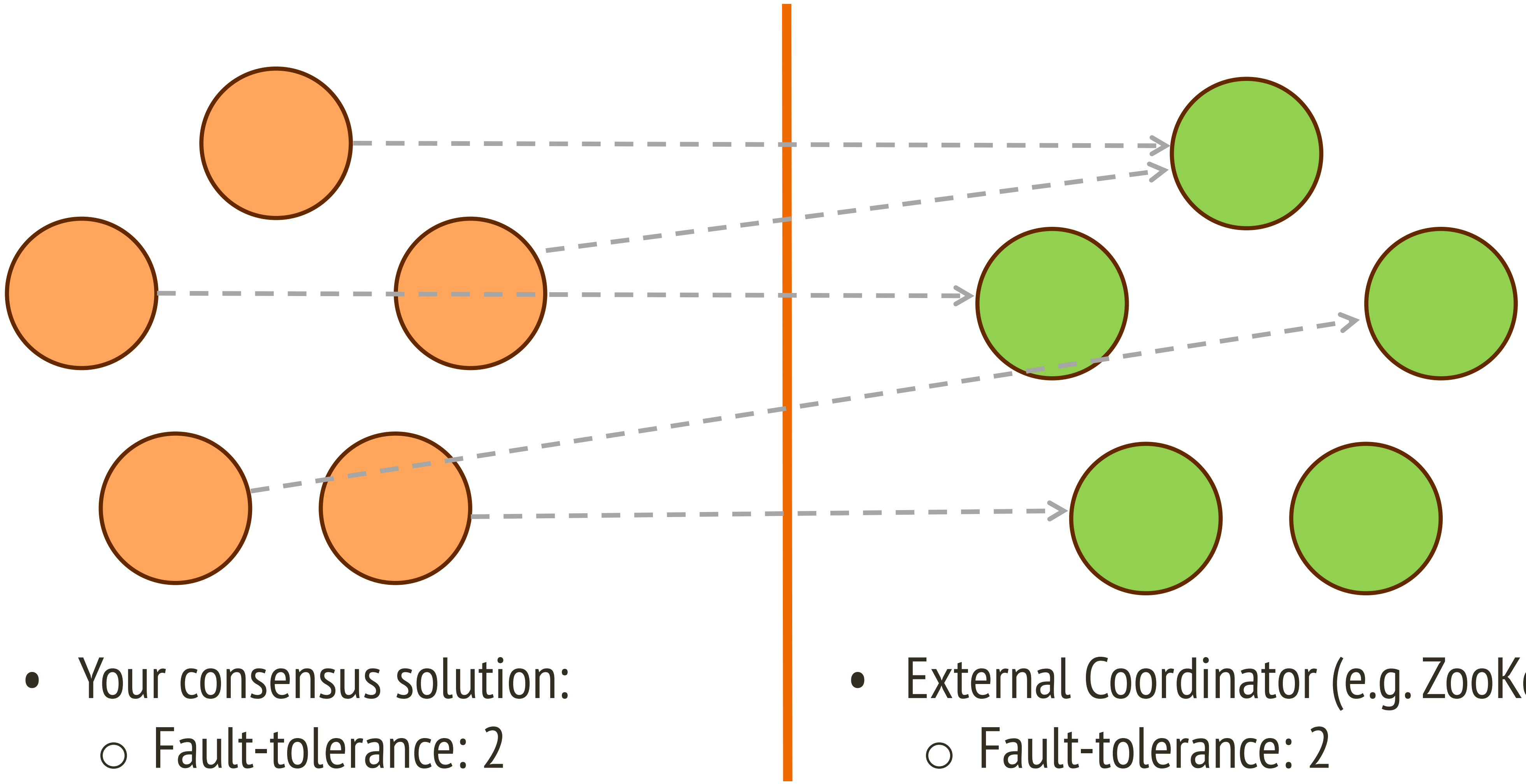
- Most consensus solutions overlook membership management
- Often assume that using an external coordinator service (e.g. ZooKeeper) is trivial and the best solution
- This is not the case:
  - Fault-tolerance becomes complex
  - Complex (and redundant) integration with consensus
  - More vulnerable to partial network partitions

Motivation : Membership Management



- Your consensus solution:
  - Fault-tolerance: 2
- External Coordinator (e.g. ZooKeeper)
  - Fault-tolerance: 1

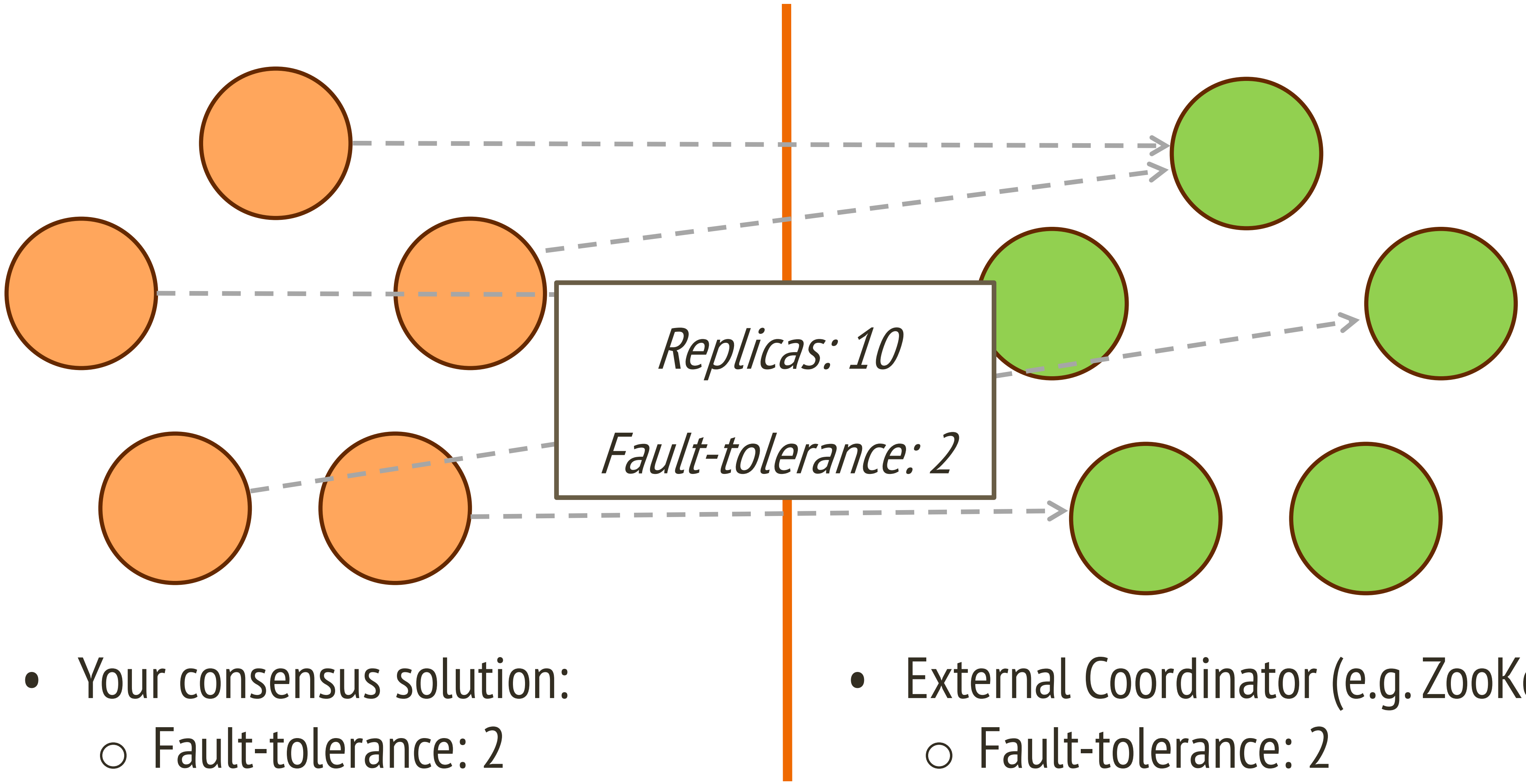
Motivation : Membership Management



- Your consensus solution:
  - Fault-tolerance: 2

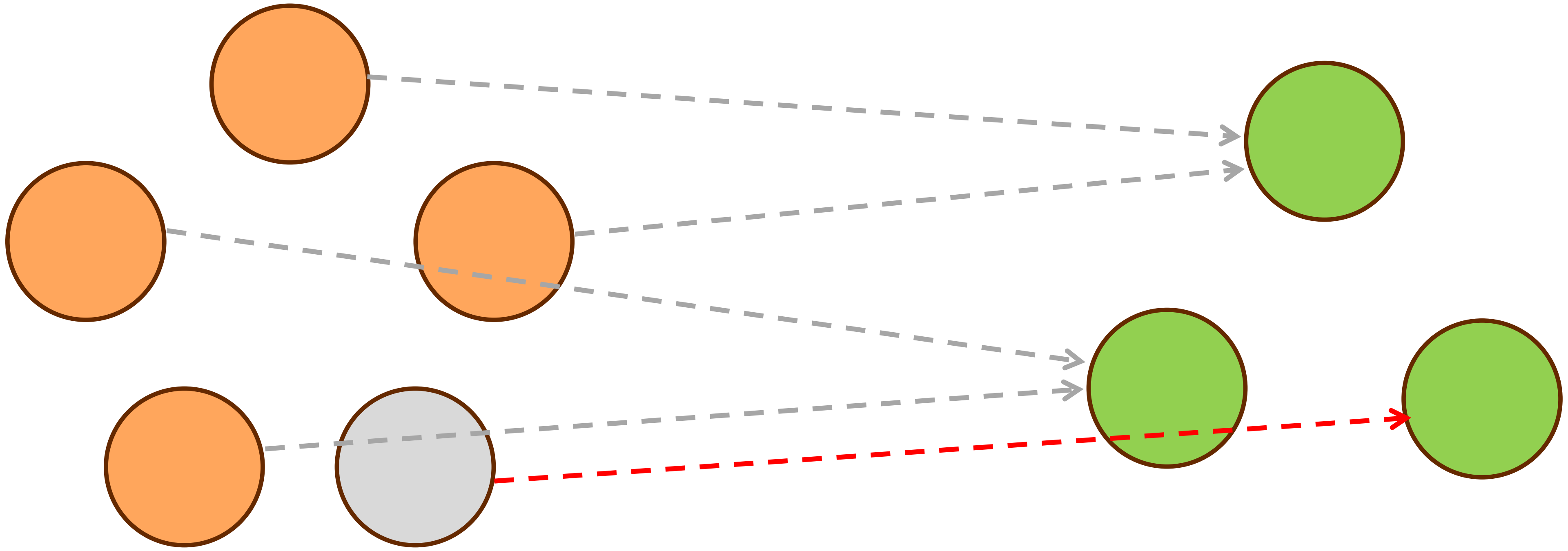
- External Coordinator (e.g. ZooKeeper)
  - Fault-tolerance: 2

Motivation : Membership Management

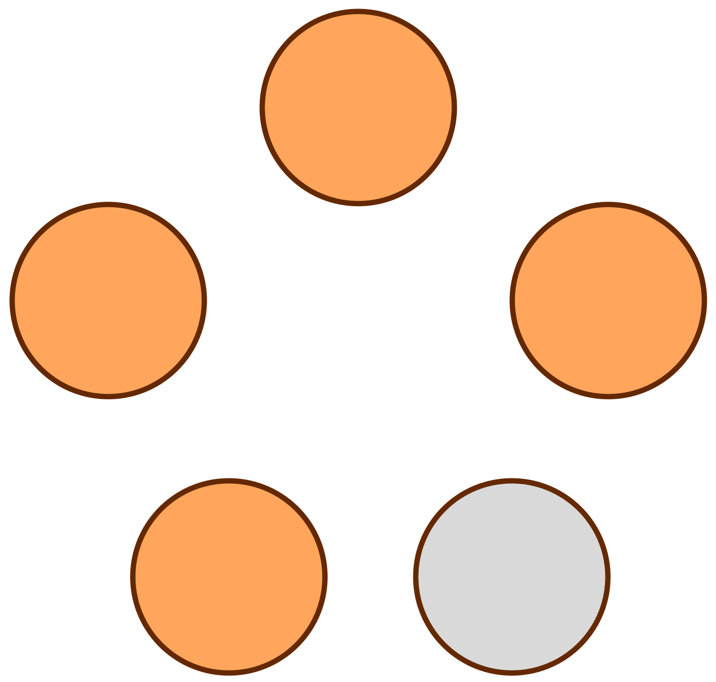


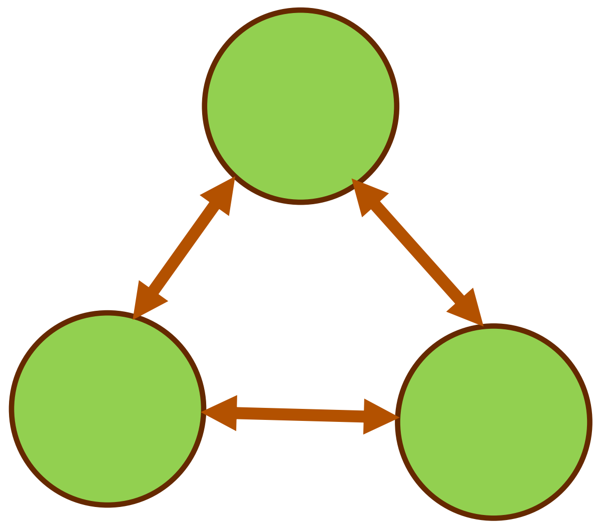
- Your consensus solution:
  - Fault-tolerance: 2
- External Coordinator (e.g. ZooKeeper)
  - Fault-tolerance: 2

Motivation : Membership Management



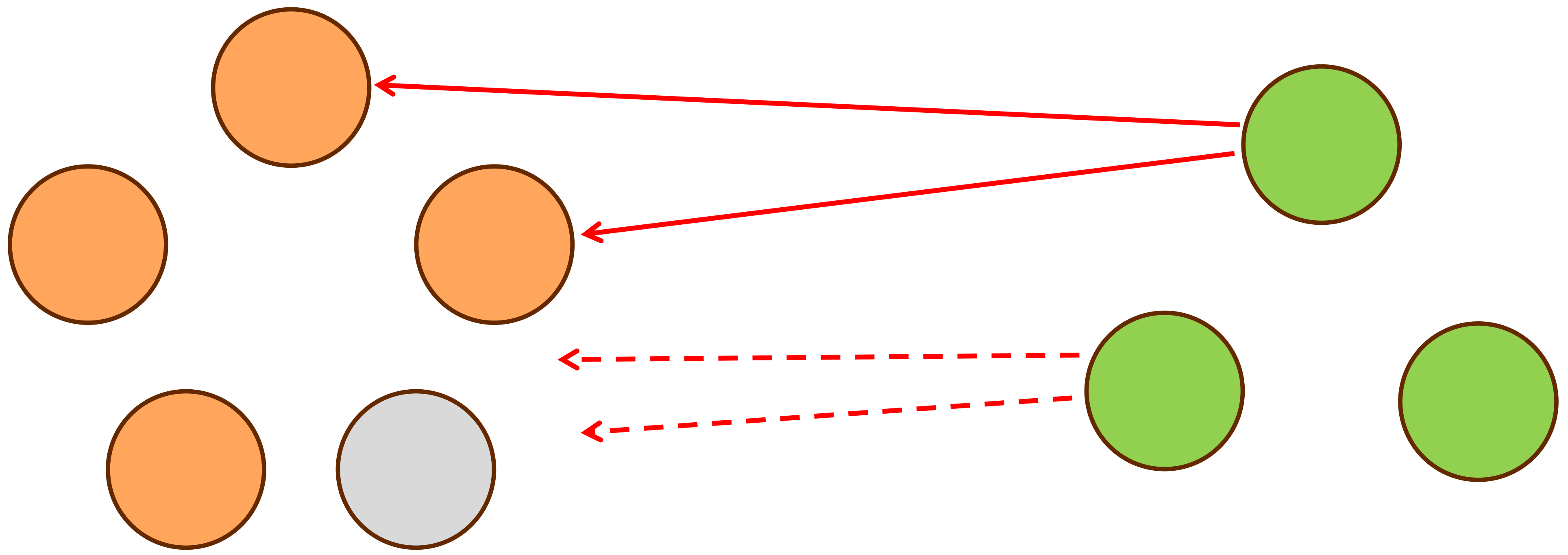
Motivation : Membership Management





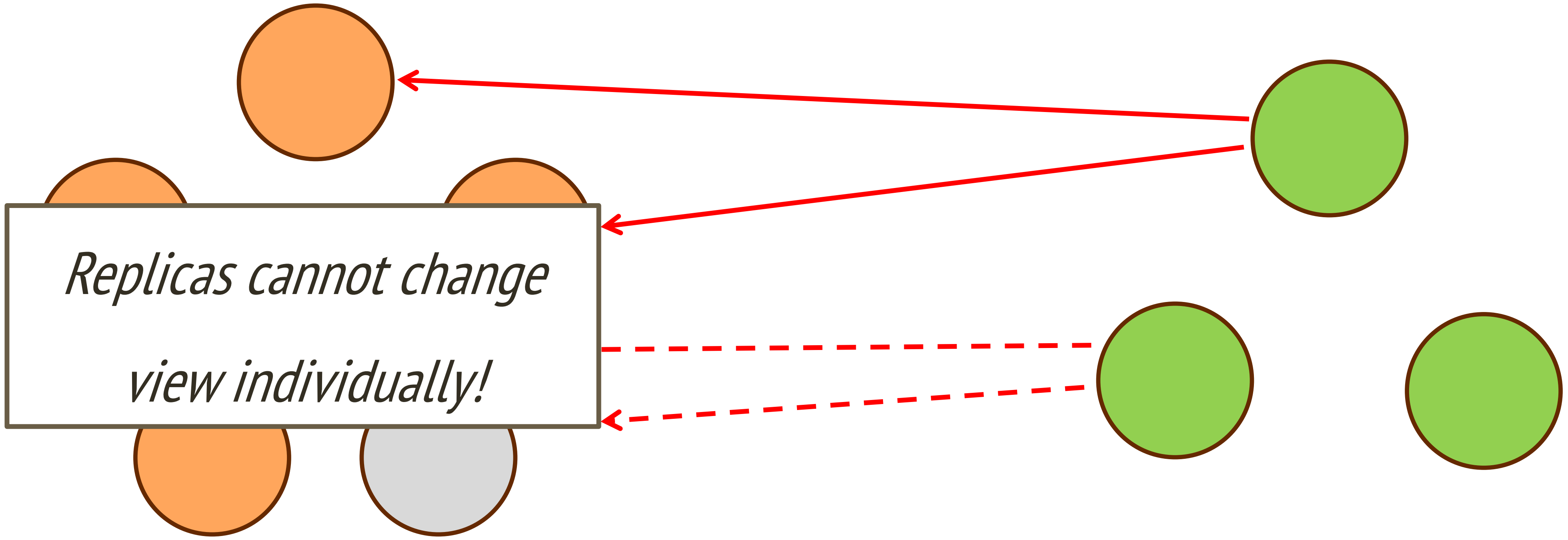
Consensus round to decide removal

Motivation : Membership Management

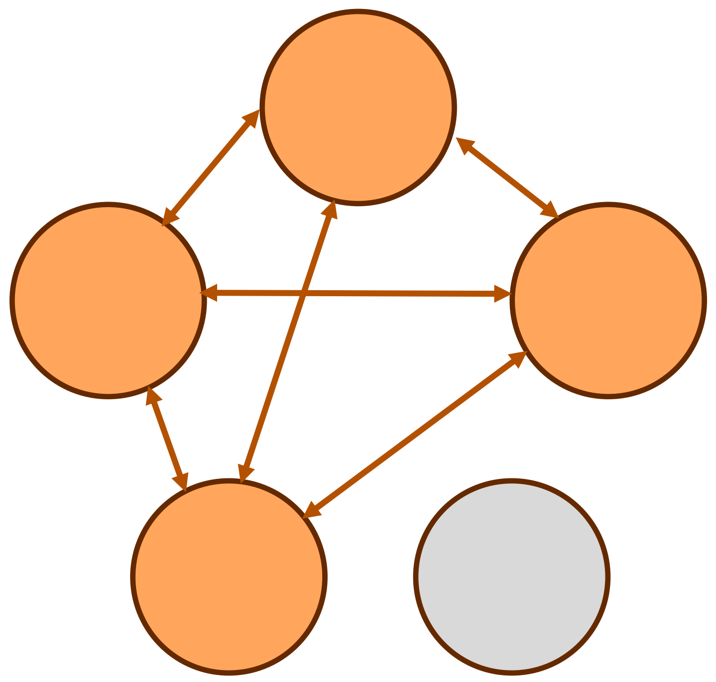


Asynchronous design propagation to consensus replicas

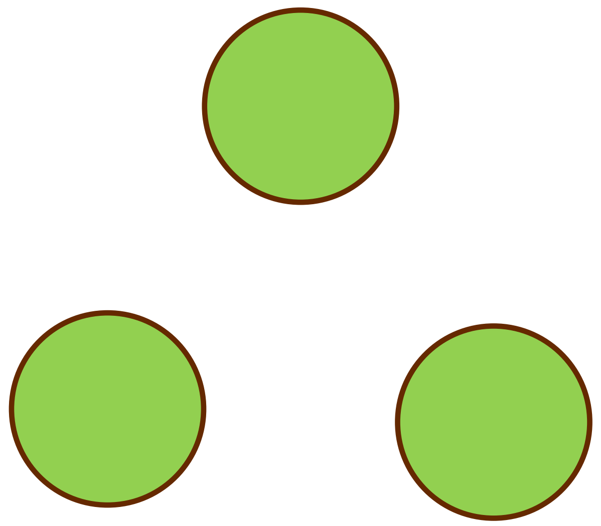
Motivation : Membership Management



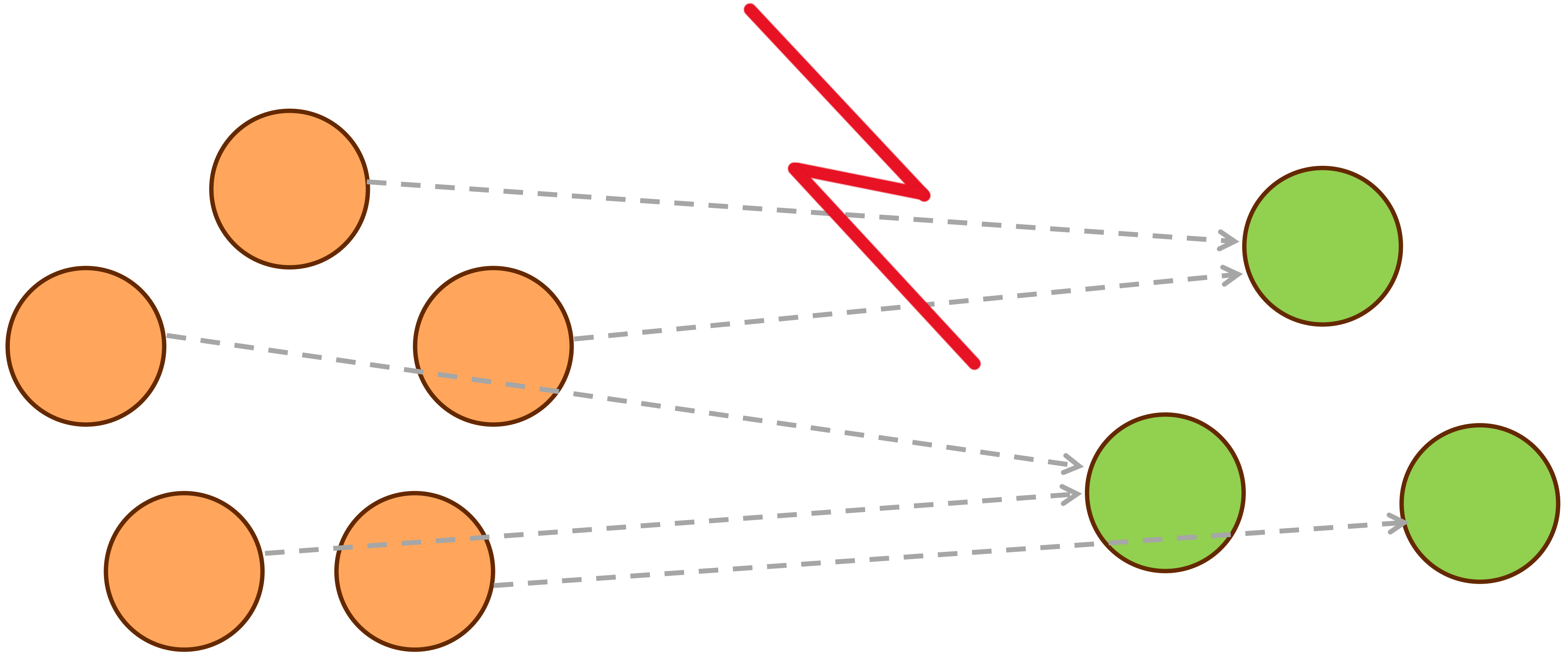
Motivation : Membership Management



Another redundant consensus round is required

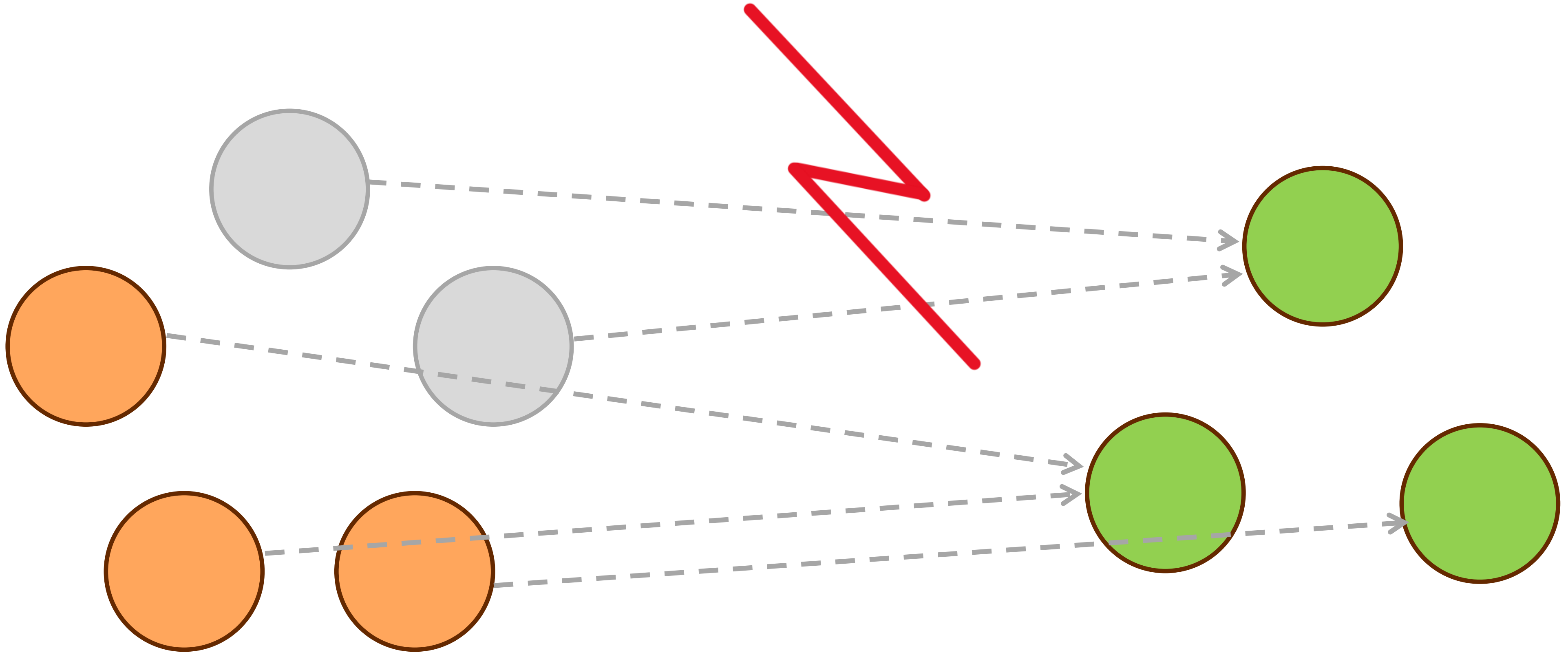


Motivation : Membership Management



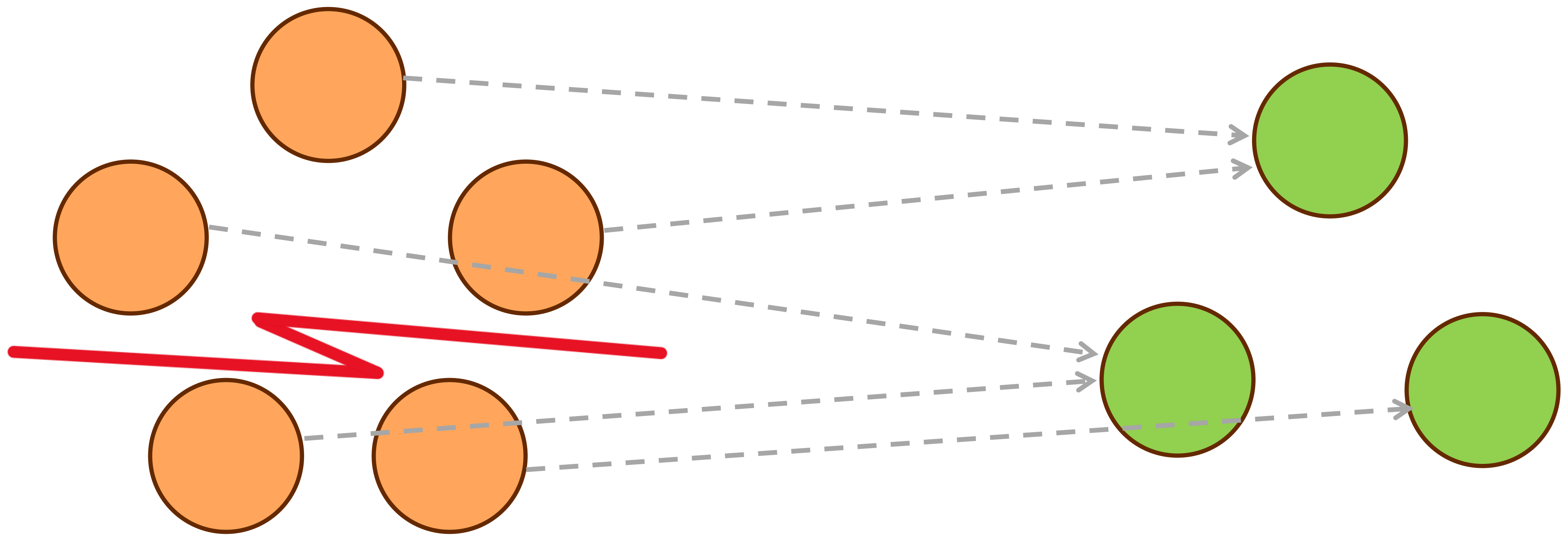
Partition between coordinator and consensus replicas

Motivation : Membership Management



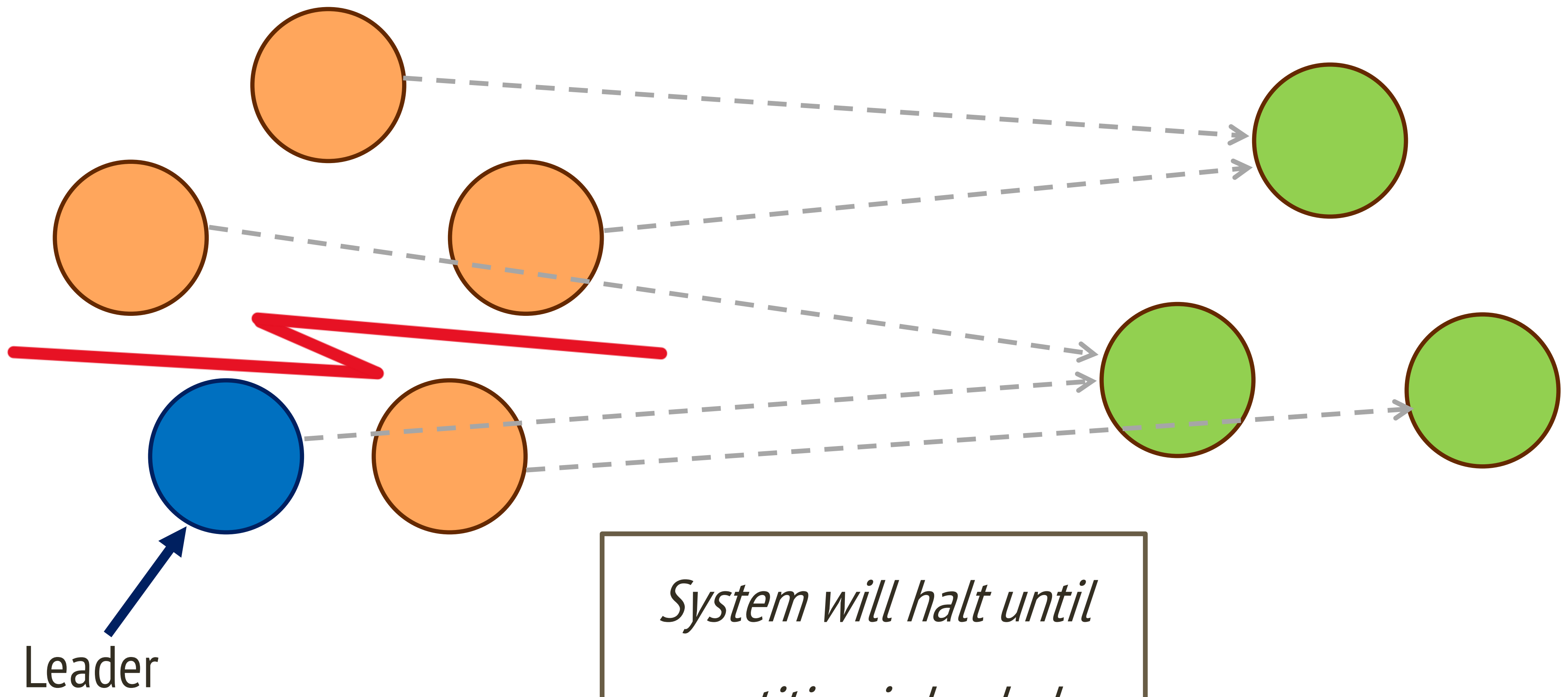
Correct replicas will be removed

Motivation : Membership Management



Partition between consensus replicas

Motivation : Membership Management



System will halt until partition is healed

Proposal : ChainPaxos

Novel consensus algorithm:

- Combining the best properties of Multi-Paxos and Chain Replication
  - Correction in an asynchronous network
  - Constant message complexity
- Going beyond existing solutions:
  - Maximizing throughput of both read and write operations
  - Providing local linearizable reads in any replica
  - Integrated reconfiguration and fault-tolerance

State of ChainPaxos Nodes

1

chain : *array of nodes*
self : *node*
c_{nextok} : *node*
$c_{sleader}$: *node*
marked : *set of node*

2

np_{leader} : *int*
inst : *map int × PaxosInst*

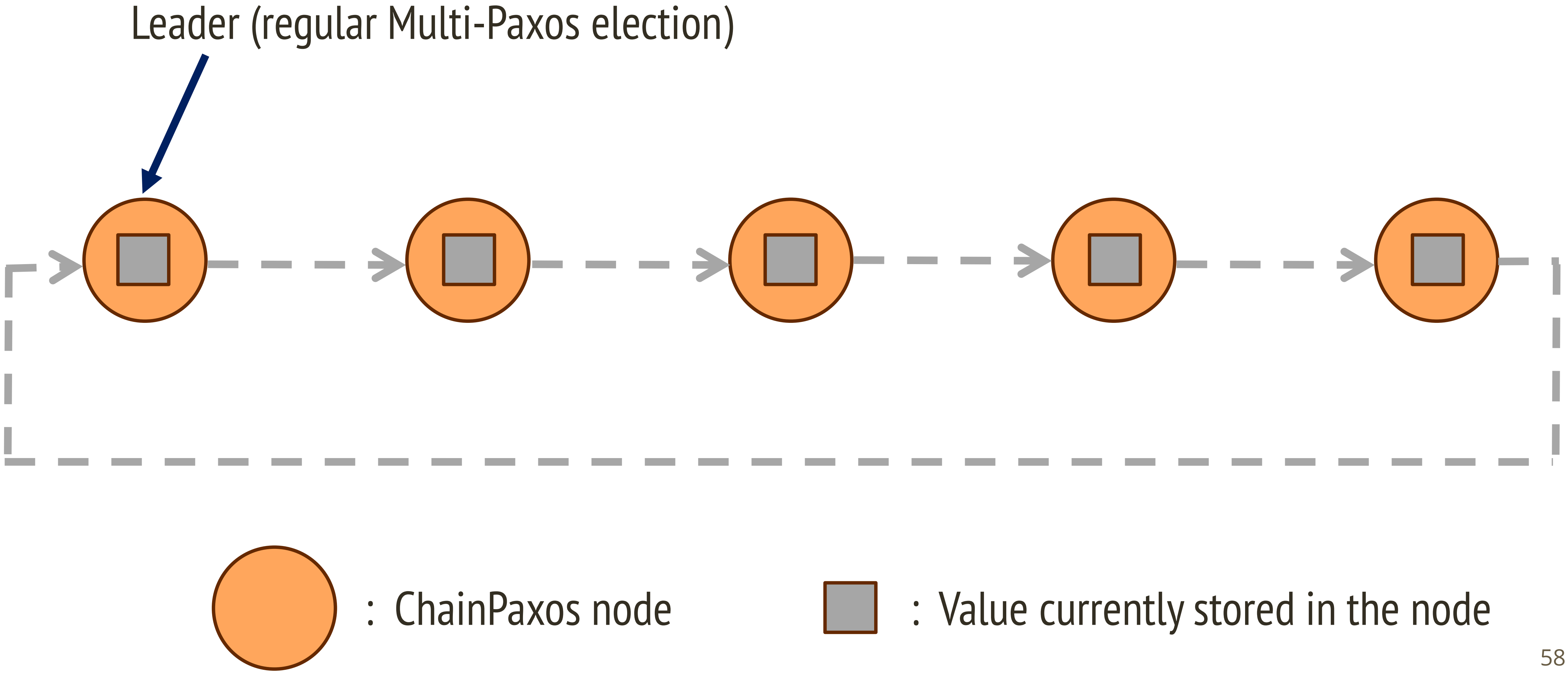
3

submitted : *set of requests*
pending : *set of requests*

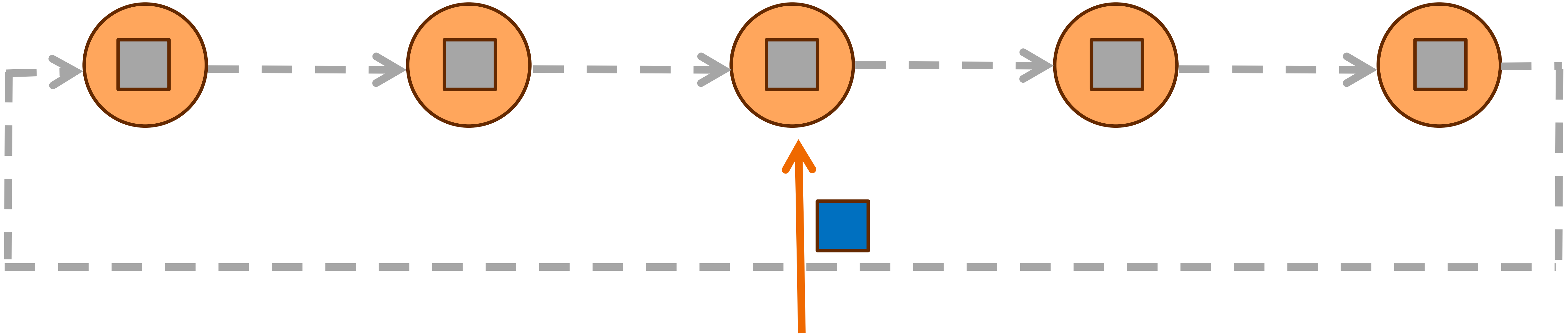
4

max_{ack} : *int*
max_{acpt} : *int*
amLeader : *bool*

ChainPaxos : Write Path

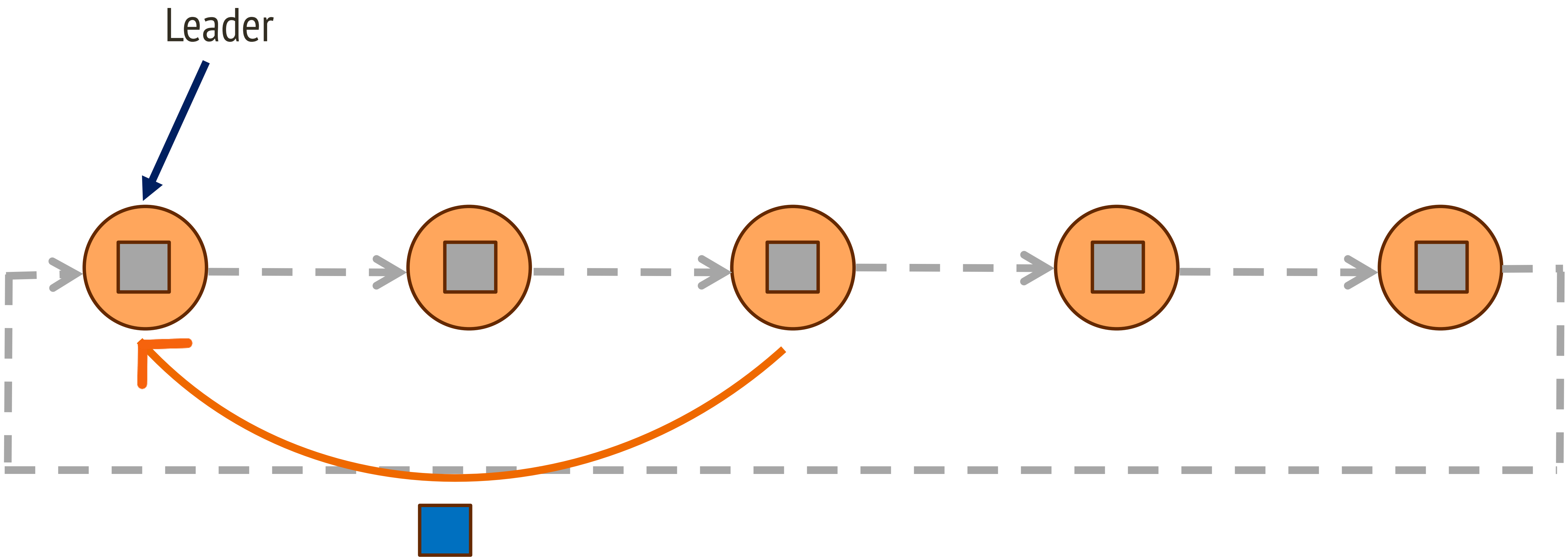

Leader (regular Multi-Paxos election)
: ChainPaxos node
: Value currently stored in the node

ChainPaxos : Write Path

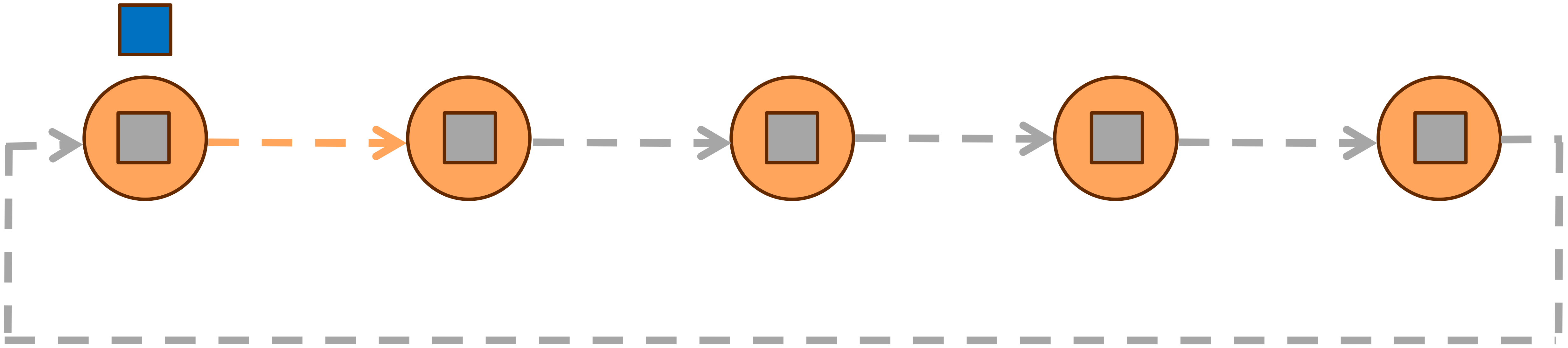


🟦 : New value to be written

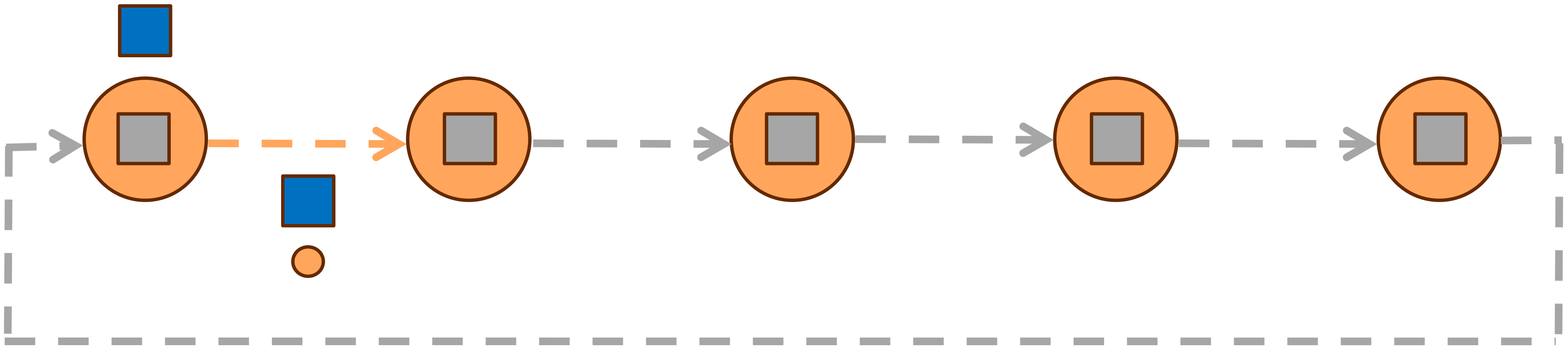
ChainPaxos : Write Path



ChainPaxos : Write Path



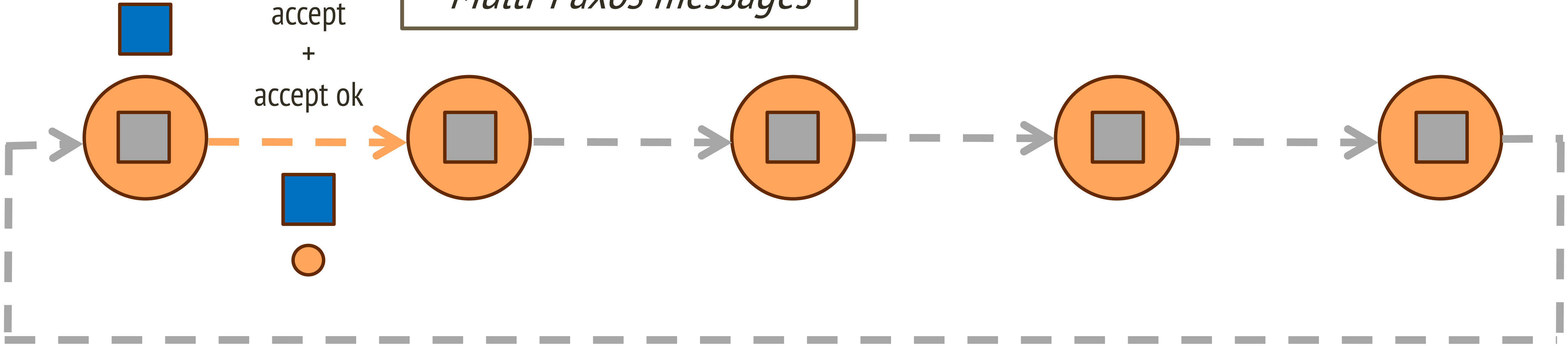
ChainPaxos : Write Path



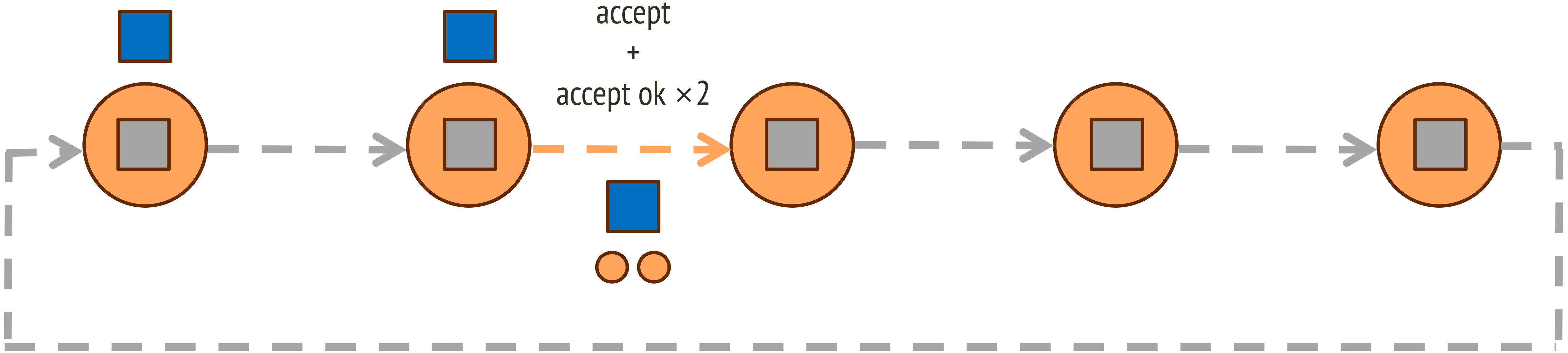
ChainPaxos : Write Path

Encapsulate multiple

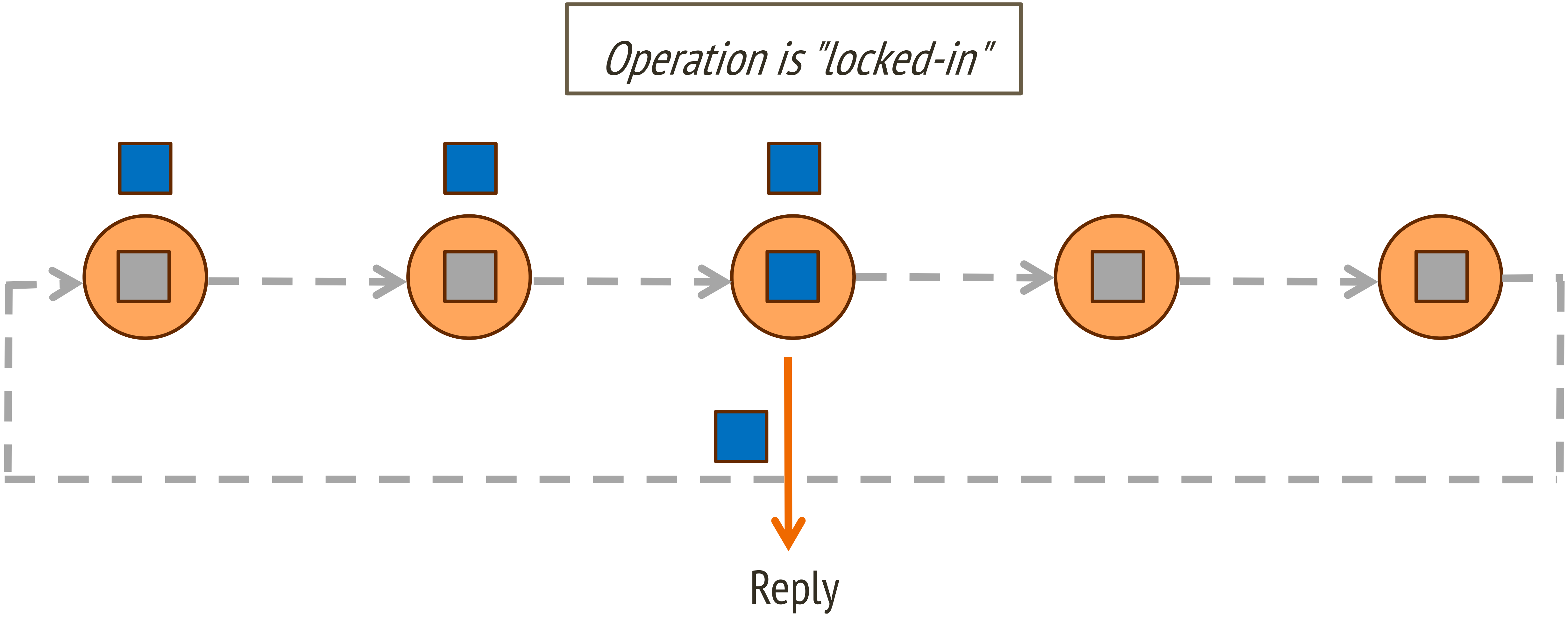
Multi-Paxos messages



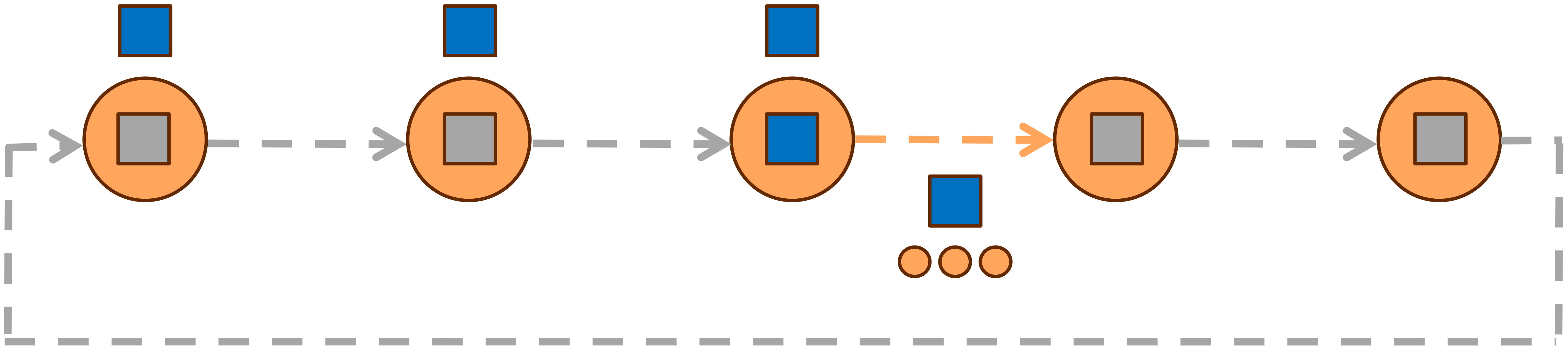
ChainPaxos : Write Path



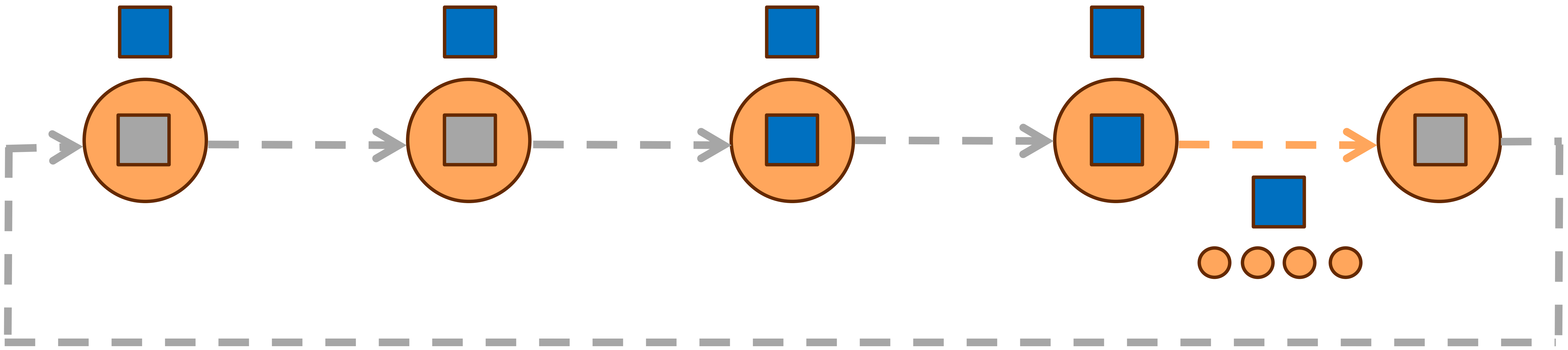
ChainPaxos : Write Path



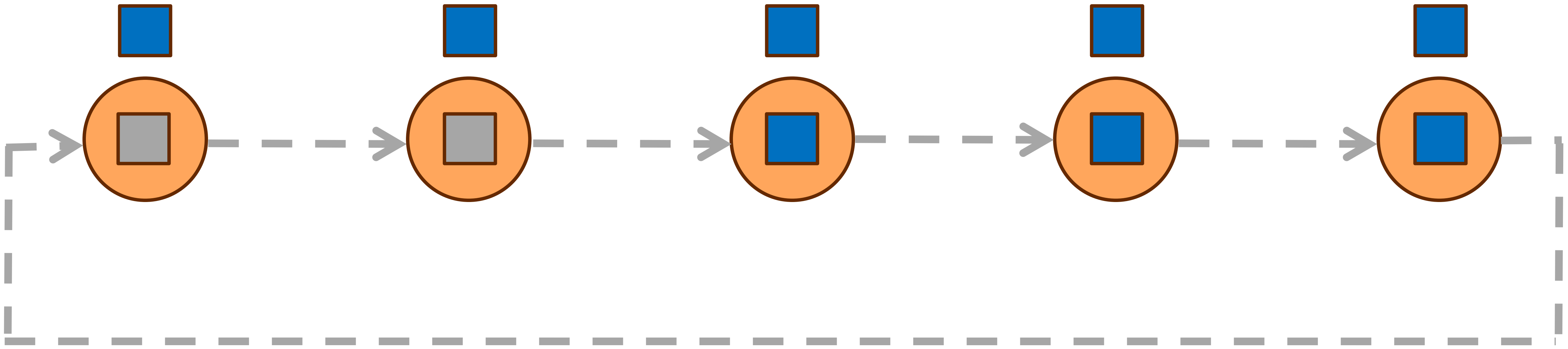
ChainPaxos : Write Path



ChainPaxos : Write Path



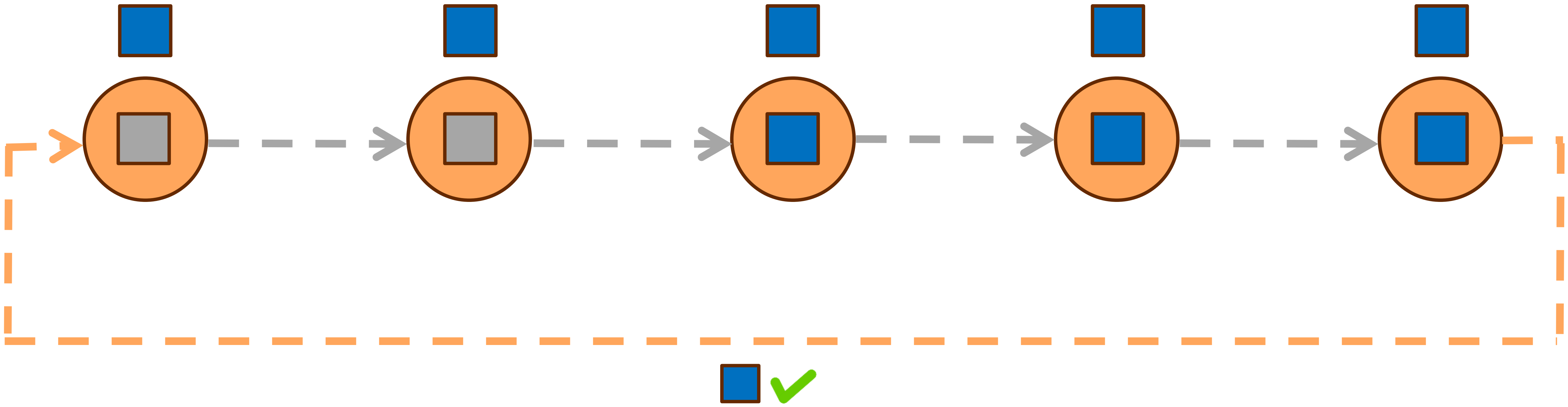
ChainPaxos : Write Path



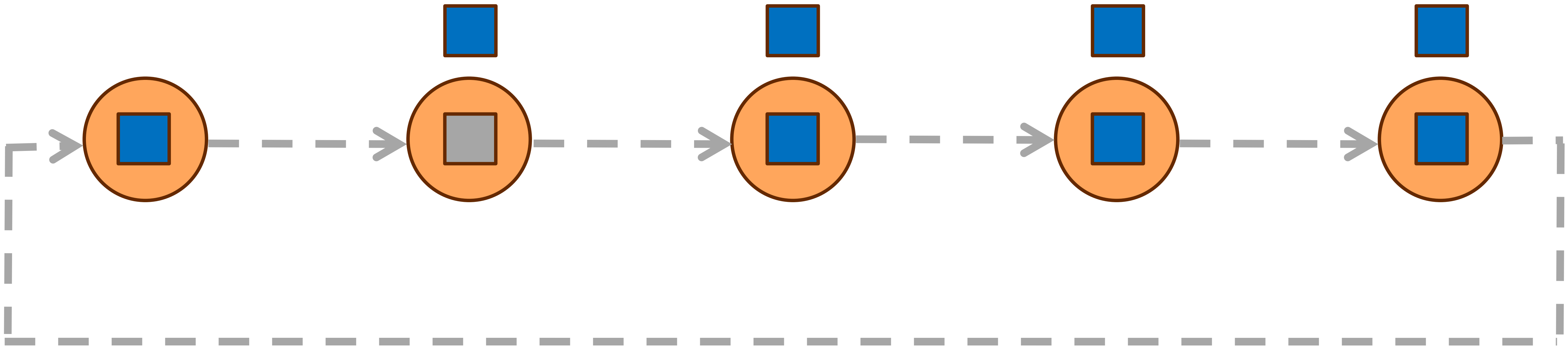
Need to garbage collect +

execute on the first replicas

ChainPaxos : Write Path

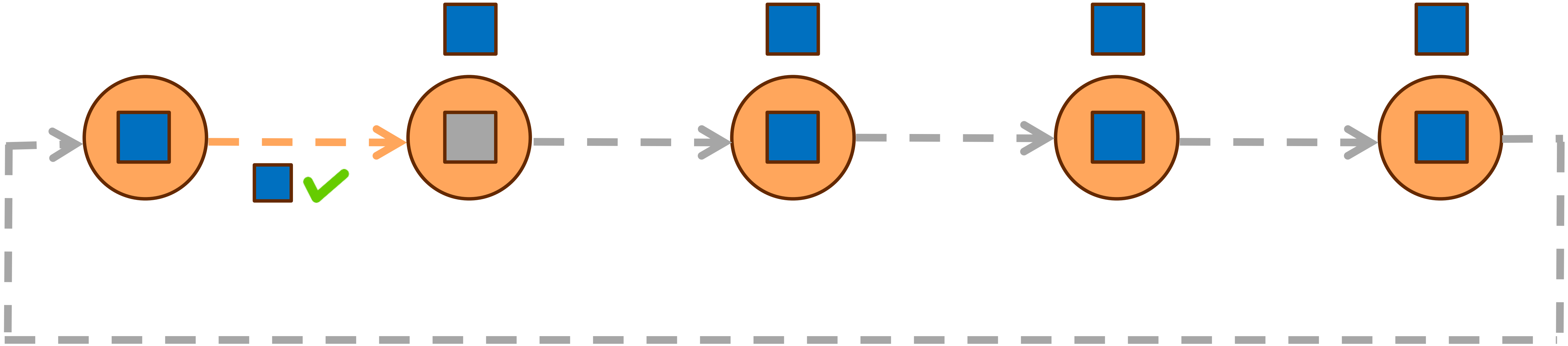


ChainPaxos : Write Path



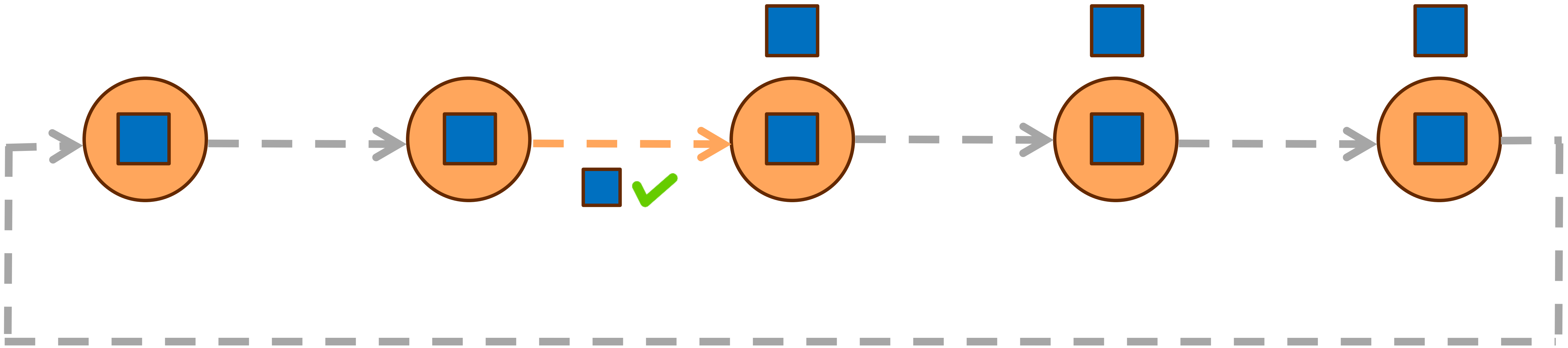
ChainPaxos : Write Path

*Ack is "piggybacked" along with next write**

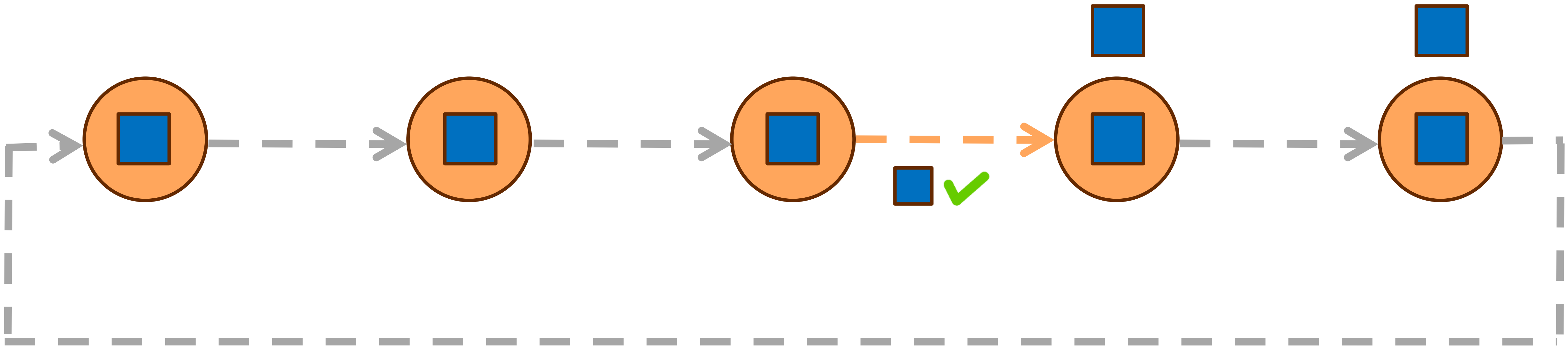


*Next write not shown for ease of representation

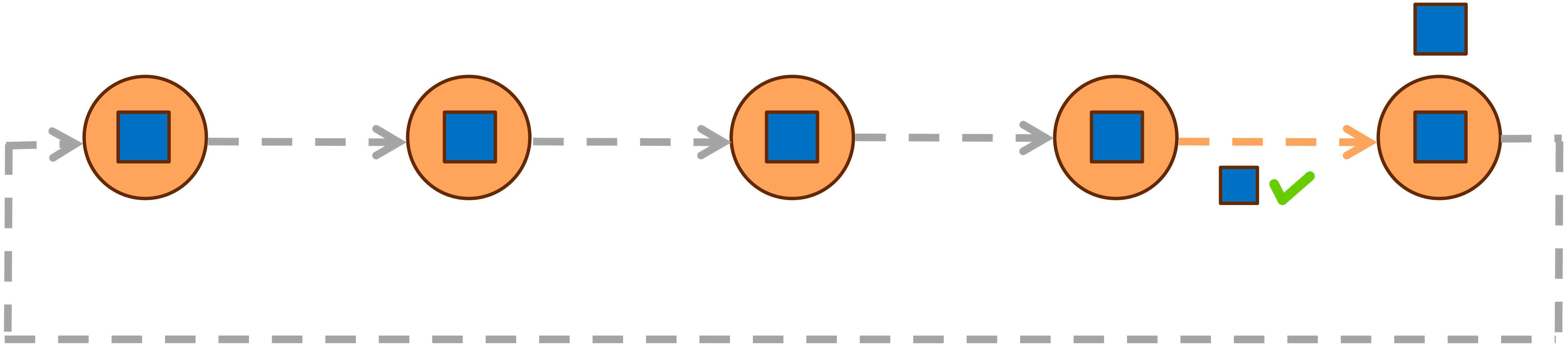
ChainPaxos : Write Path



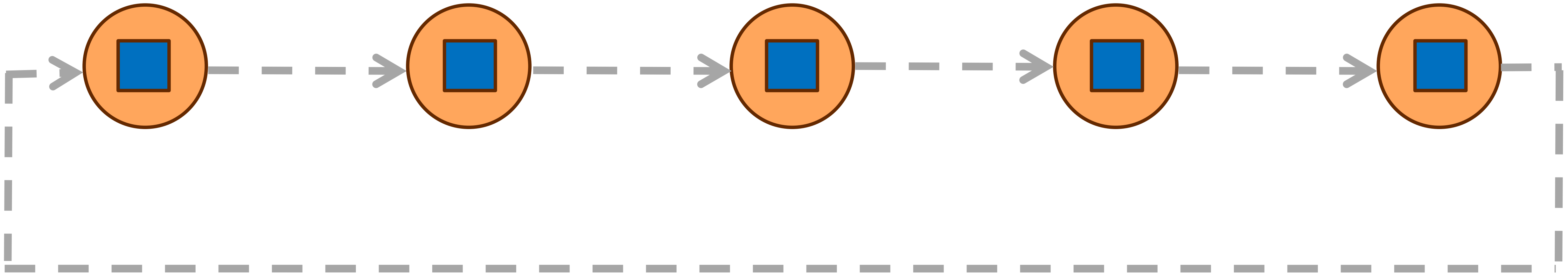
ChainPaxos : Write Path



ChainPaxos : Write Path



ChainPaxos : Write Path



Local Linearizable Reads

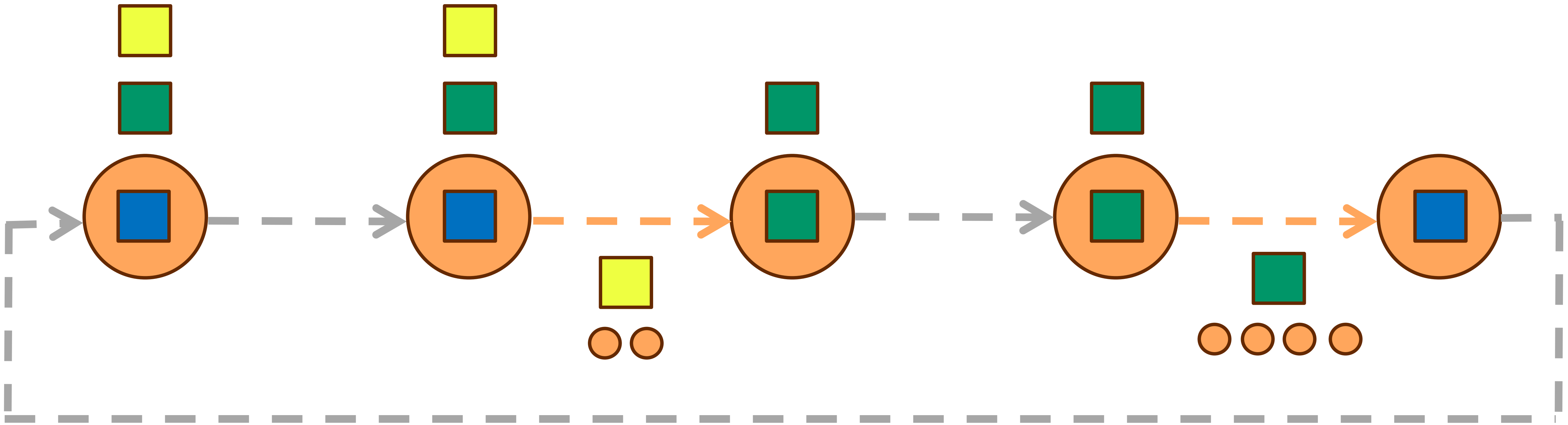
Requirements for Linearizability:

- The result of a read must contain **all writes that completed** before it started
- The result of a read must contain the result of **all reads that completed** before it started

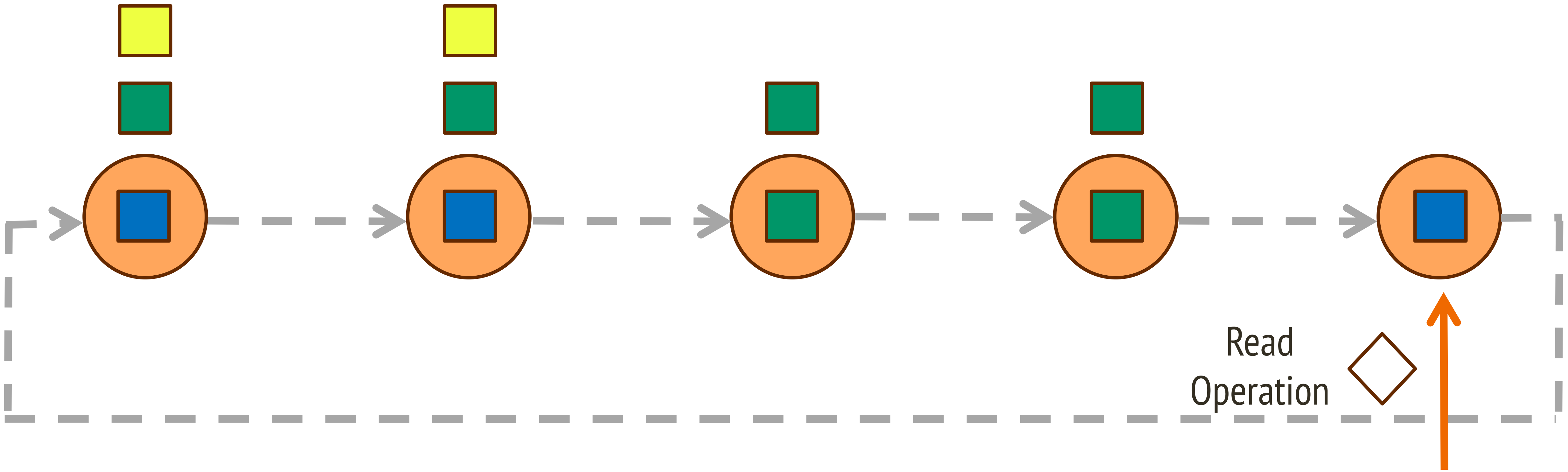
Challenge:

- Read from **any replica**
- **No extra communication** steps

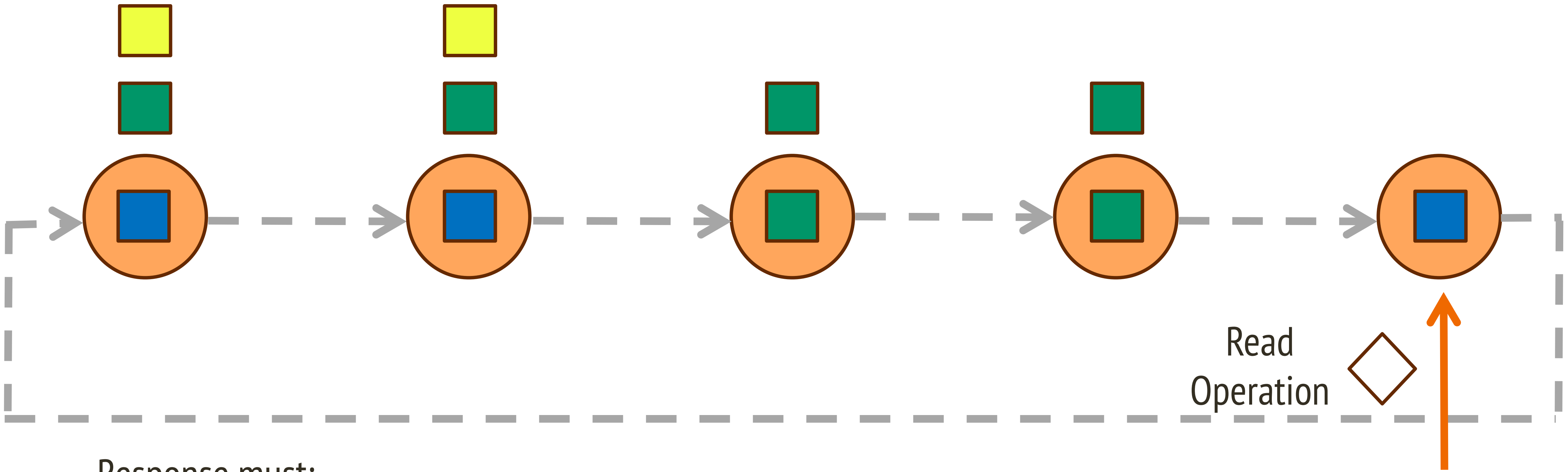
Local Linearizable Reads



Local Linearizable Reads



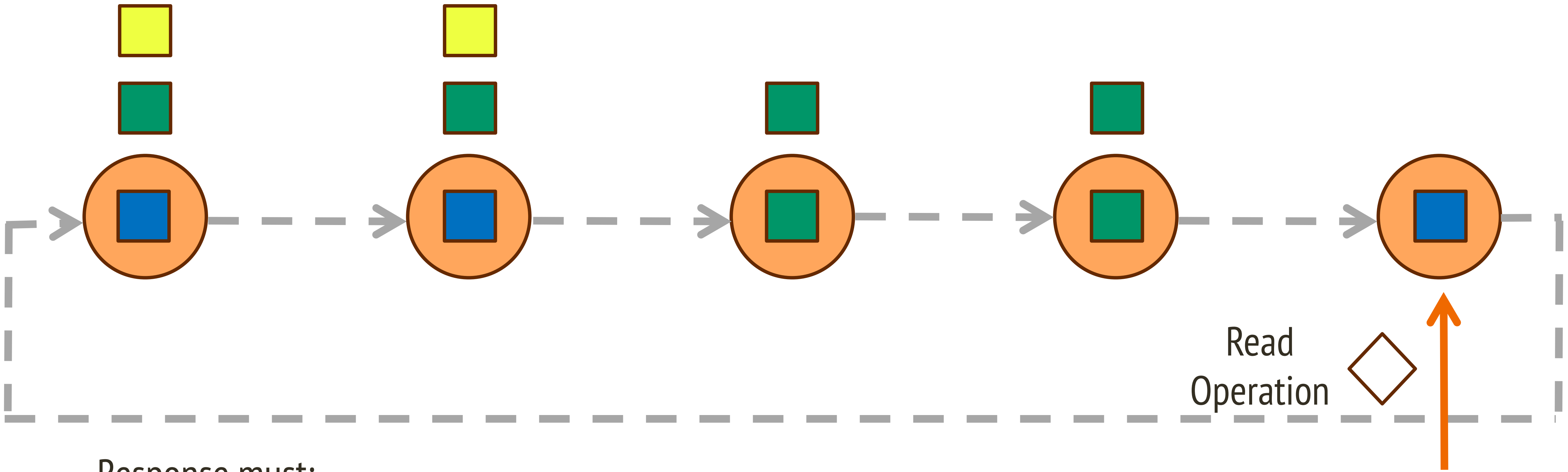
Local Linearizable Reads



Response must:

- Contain all completed writes
- Contain all completed reads

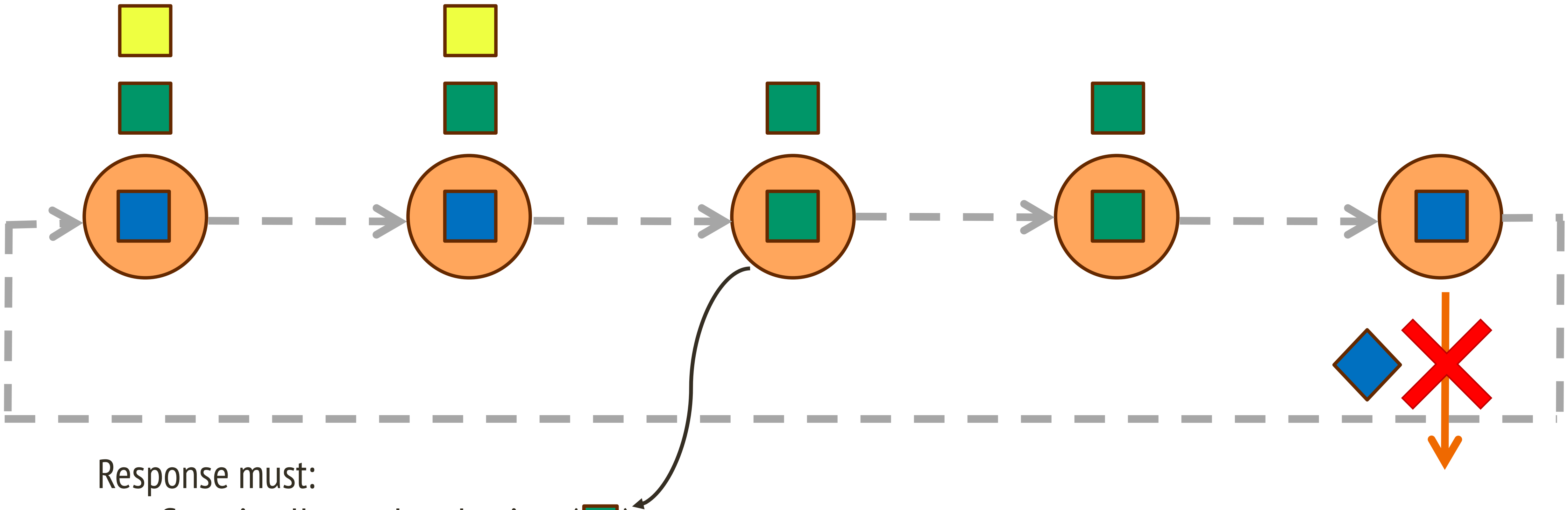
Local Linearizable Reads



Response must:

- Contain all completed writes
- Contain all completed reads

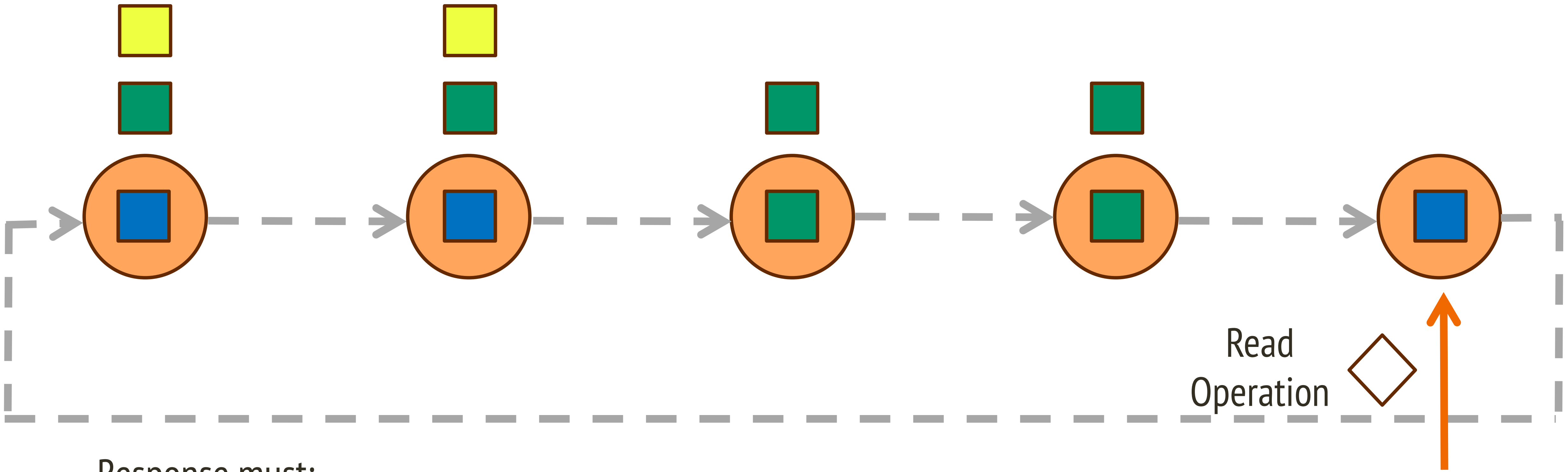
Local Linearizable Reads



Response must:

- Contain all completed writes (🟩)
- Contain all completed reads

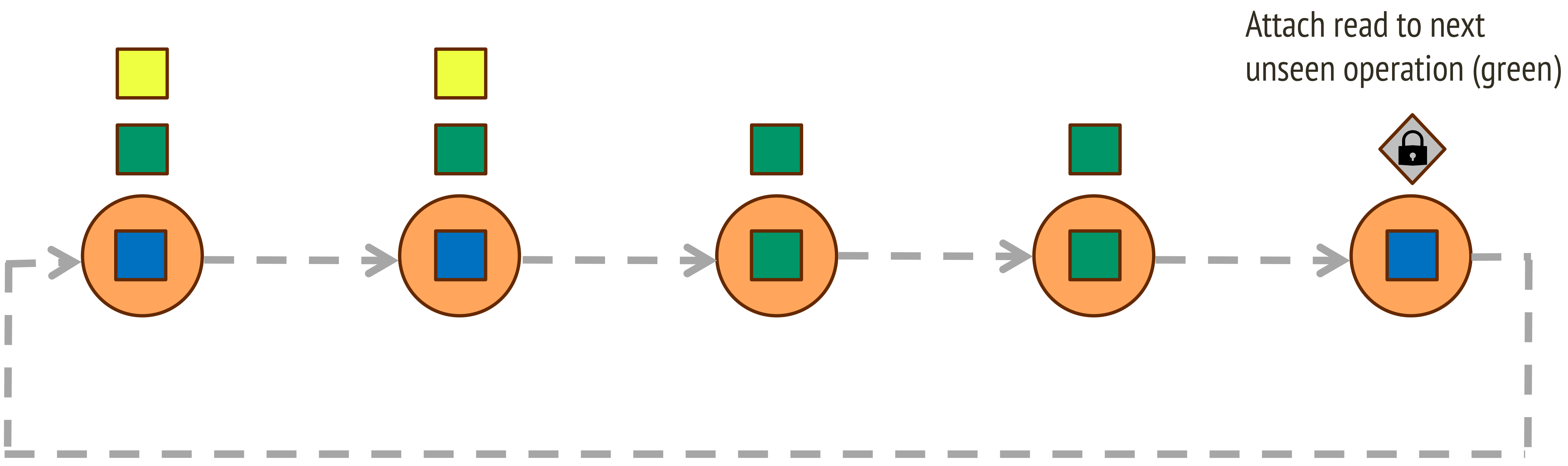
Local Linearizable Reads



Response must:

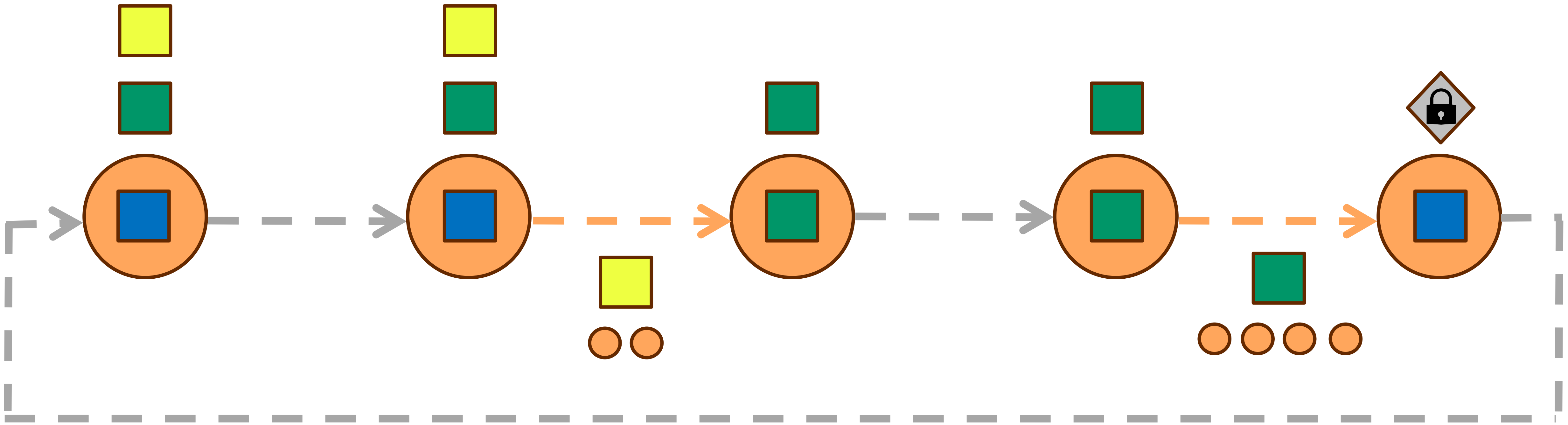
- Contain all completed writes (🟩)
- Contain all completed reads

Local Linearizable Reads

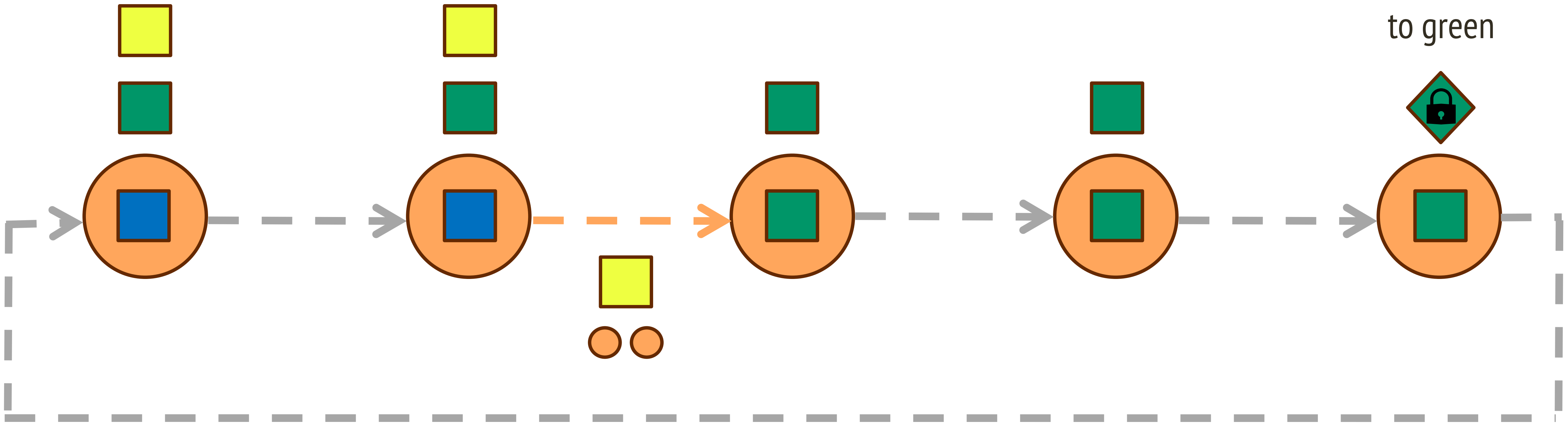


: Waiting Read

Local Linearizable Reads

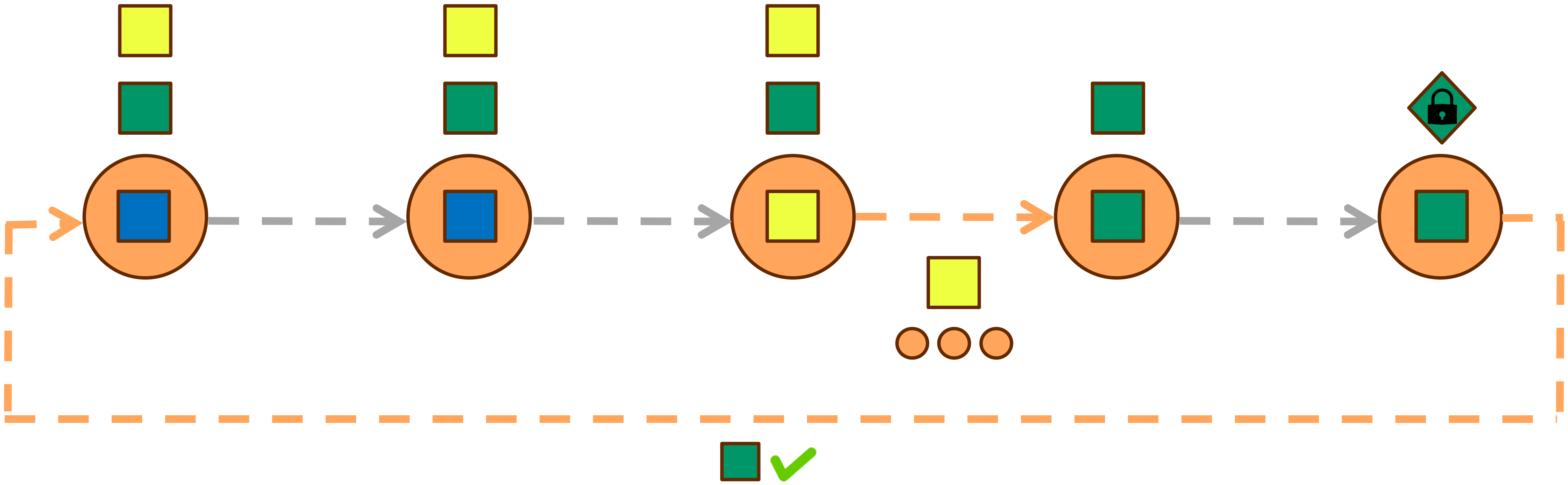


Local Linearizable Reads

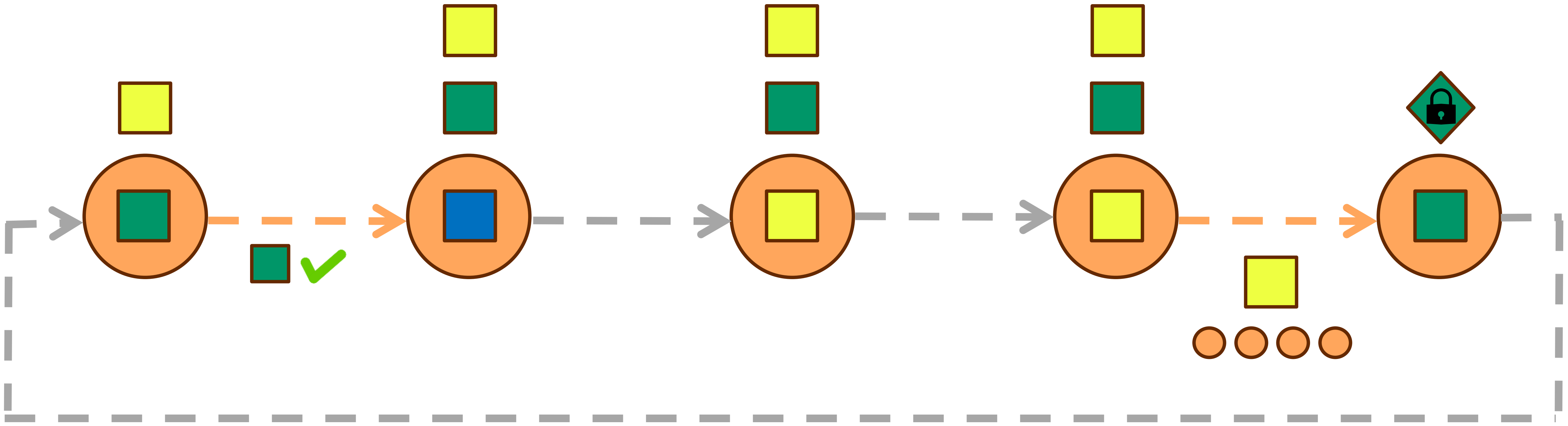


Wait until green ack goes around chain

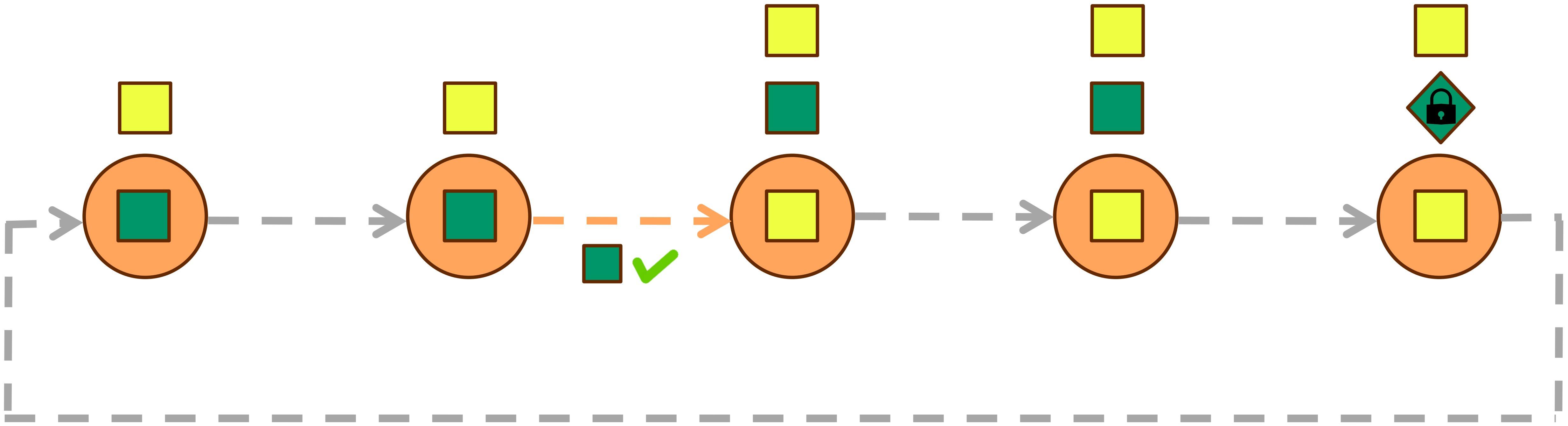
Local Linearizable Reads



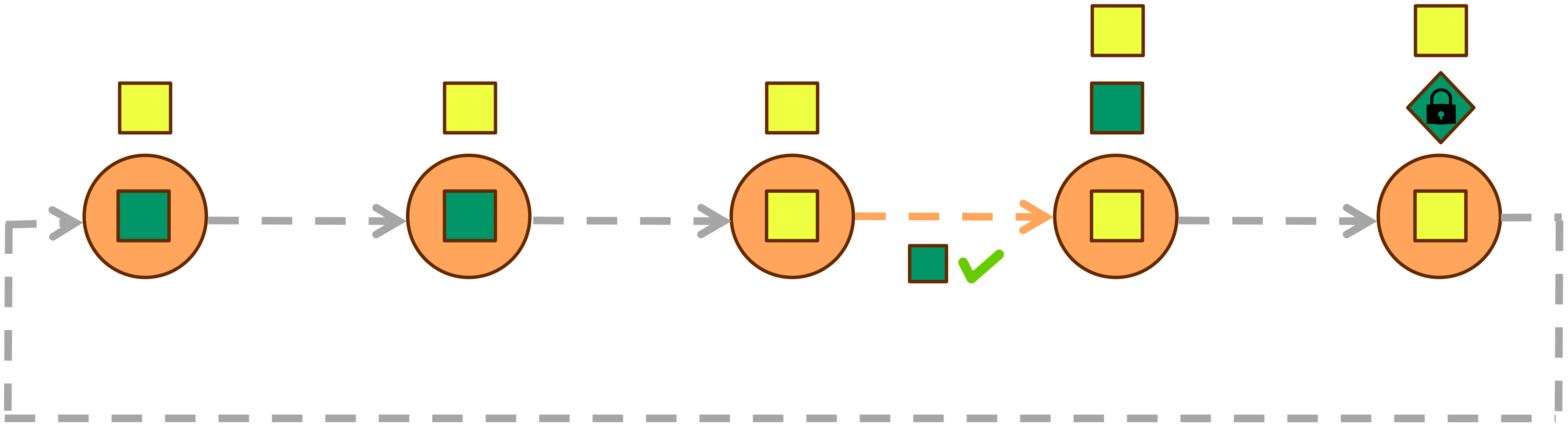
Local Linearizable Reads



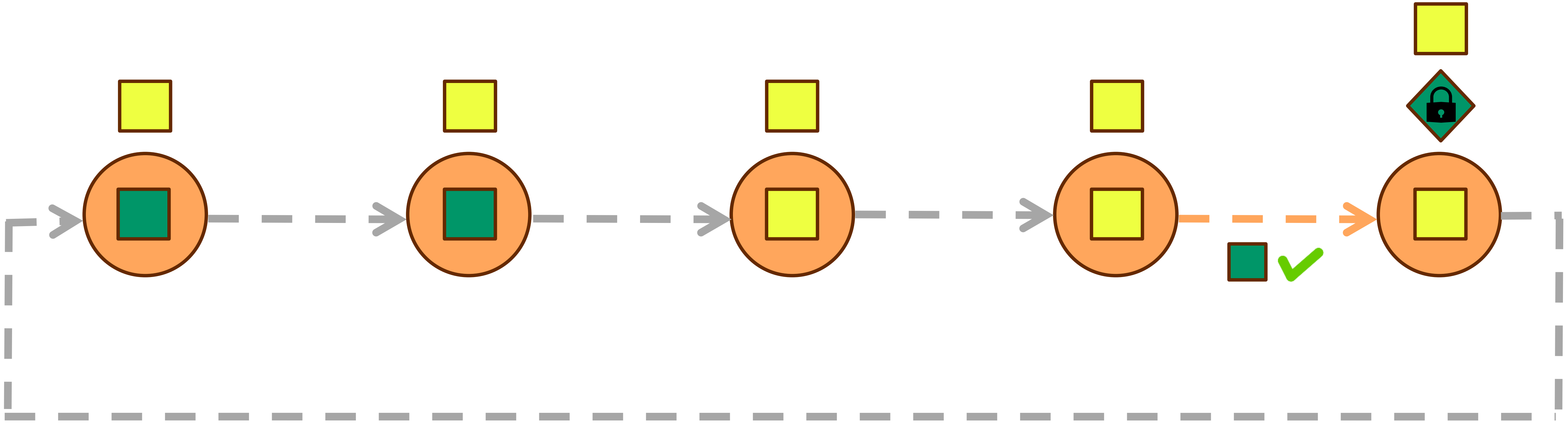
Local Linearizable Reads



Local Linearizable Reads

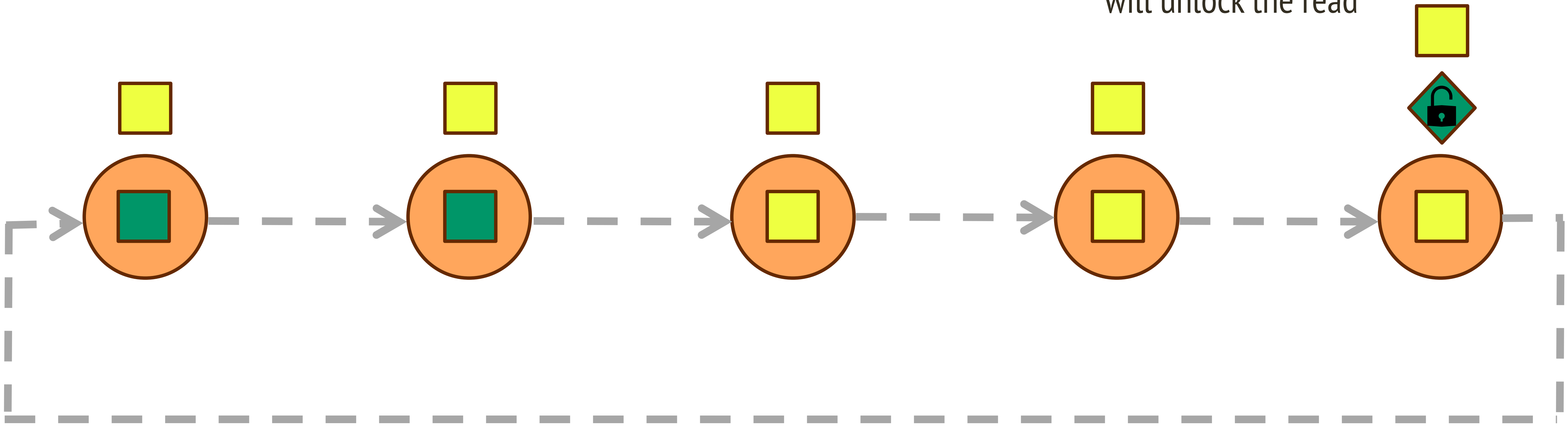


Local Linearizable Reads

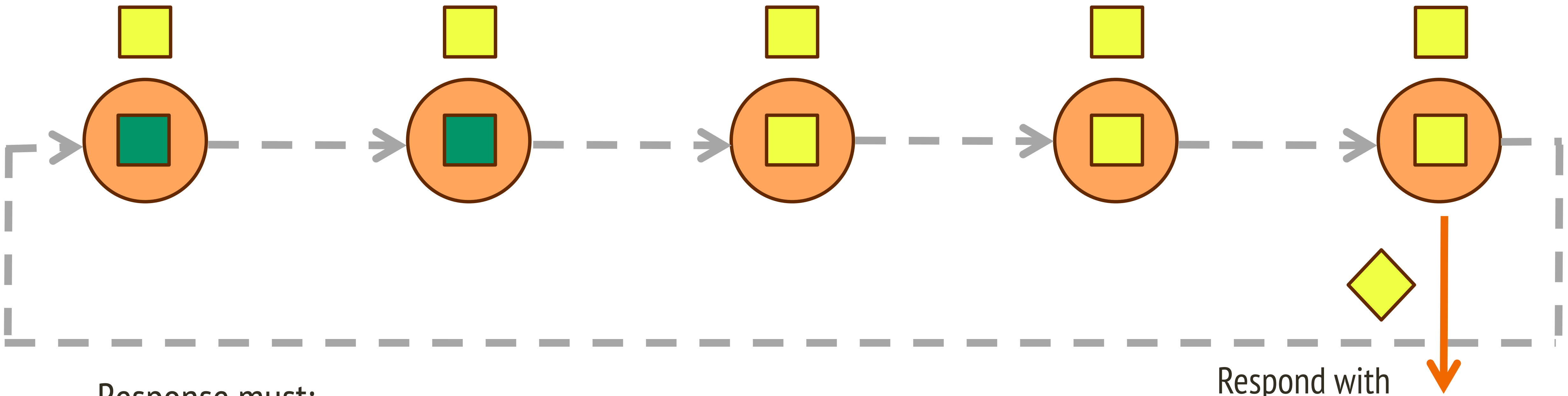


Local Linearizable Reads

Acknowledge of green will unlock the read



Local Linearizable Reads



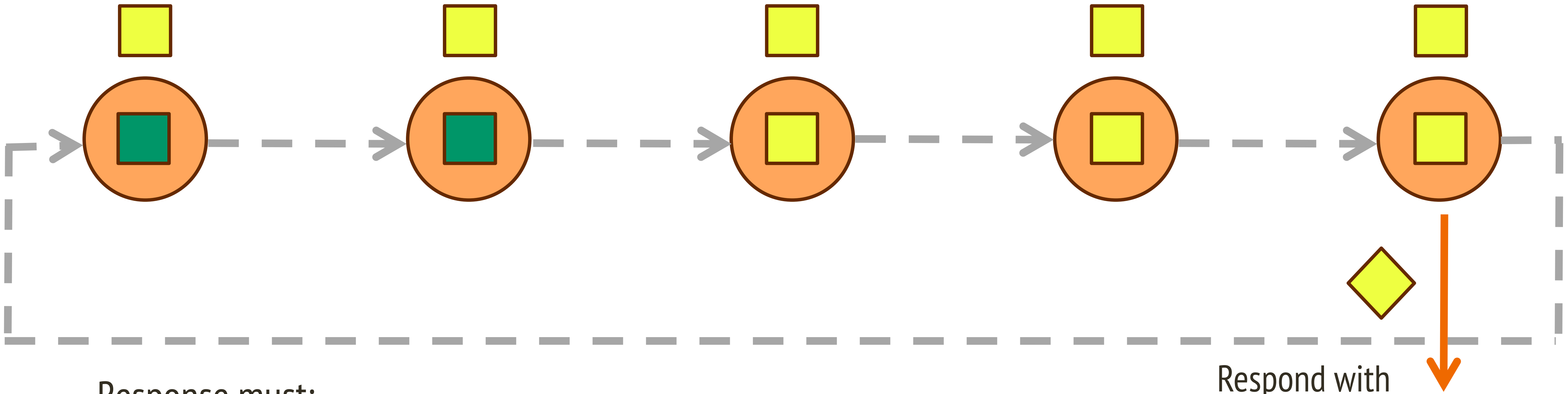
Response must:

- Contain all completed writes (🟩)
- Contain all completed reads

Local Linearizable Reads

Stronger than required (🟨)

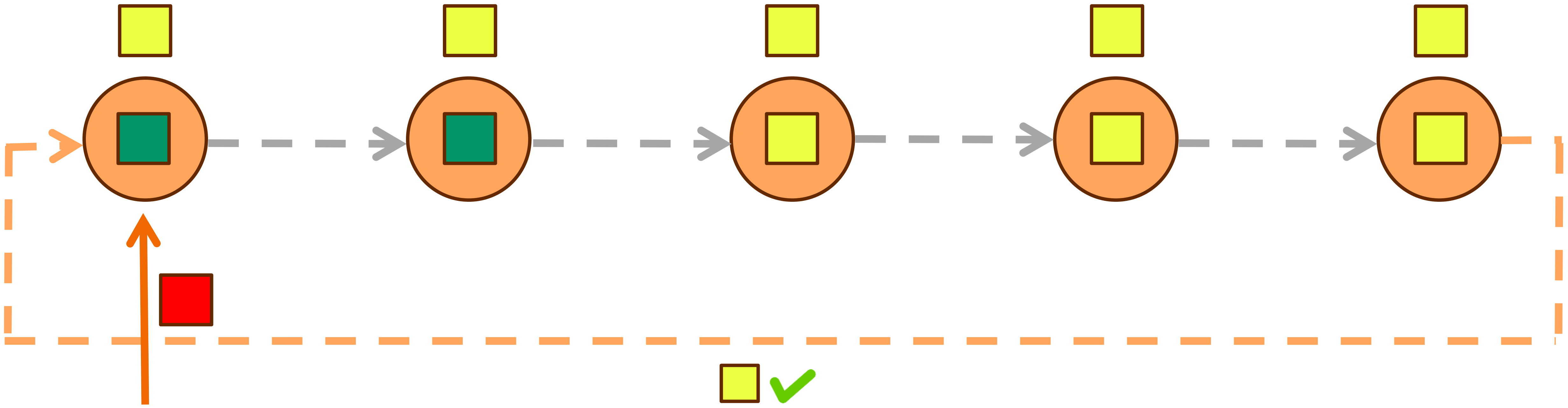
Best we can do without extra communication



Response must:

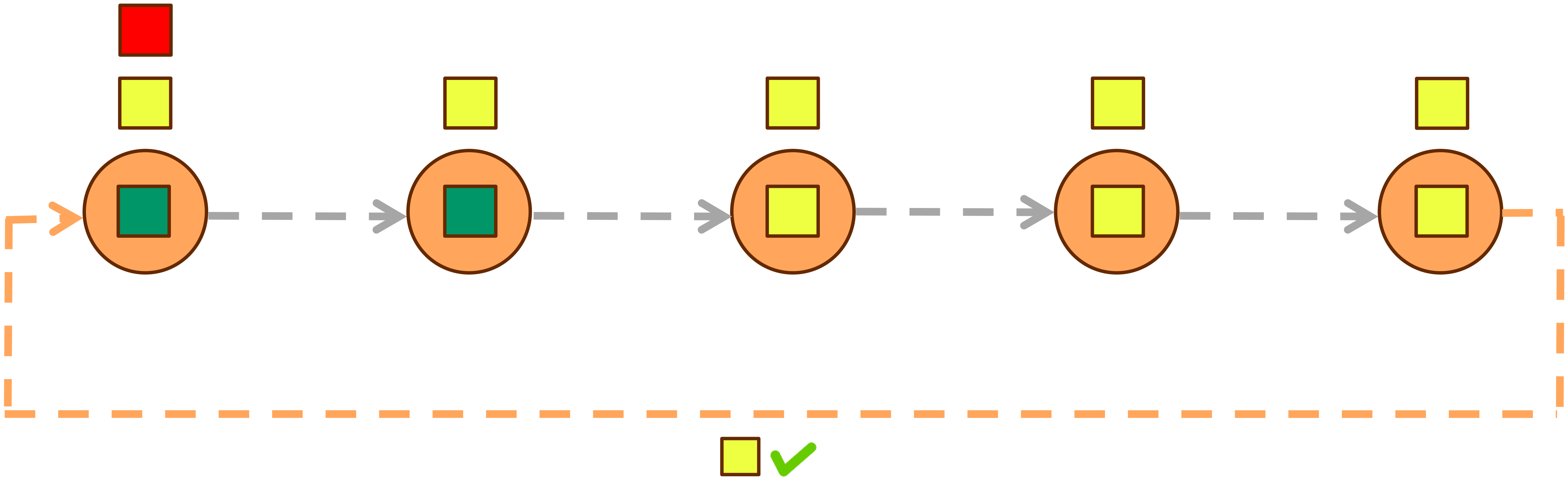
- Contain all completed writes (🟩)
- Contain all completed reads

Local Linearizable Reads

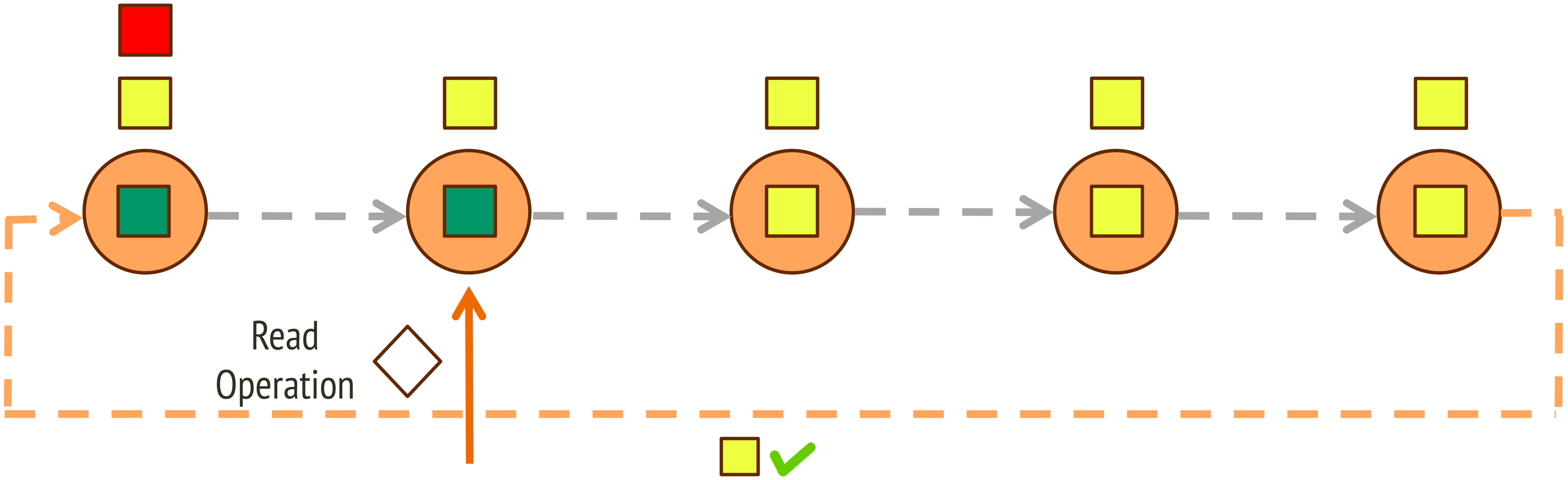


🟥 : New value to be written

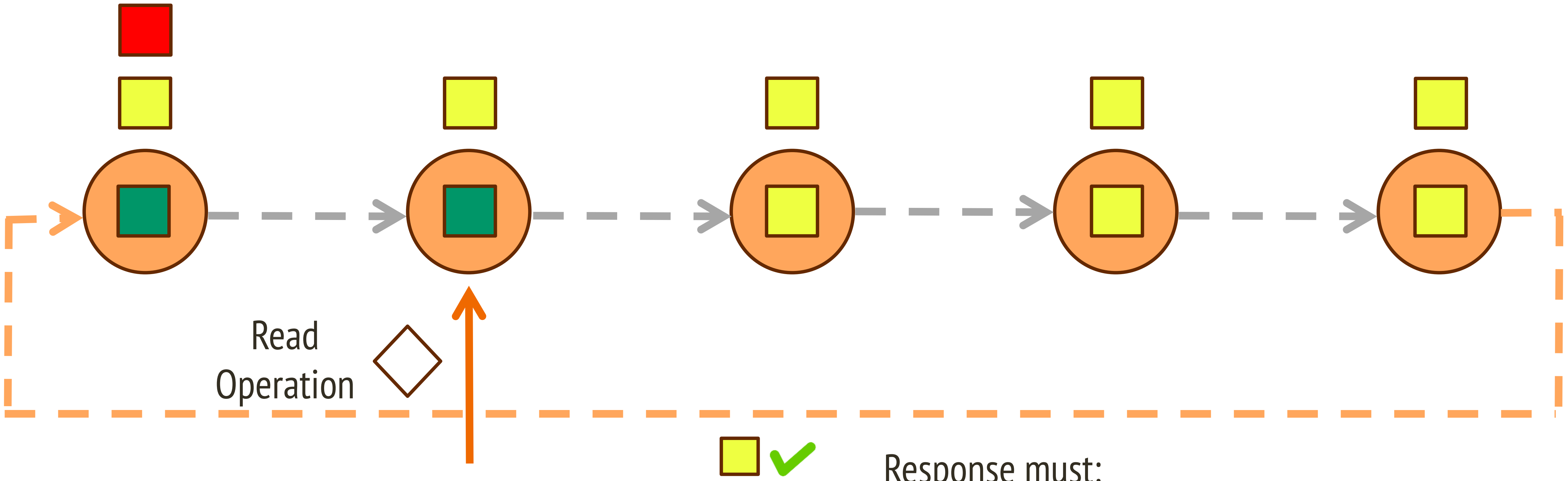
Local Linearizable Reads



Local Linearizable Reads

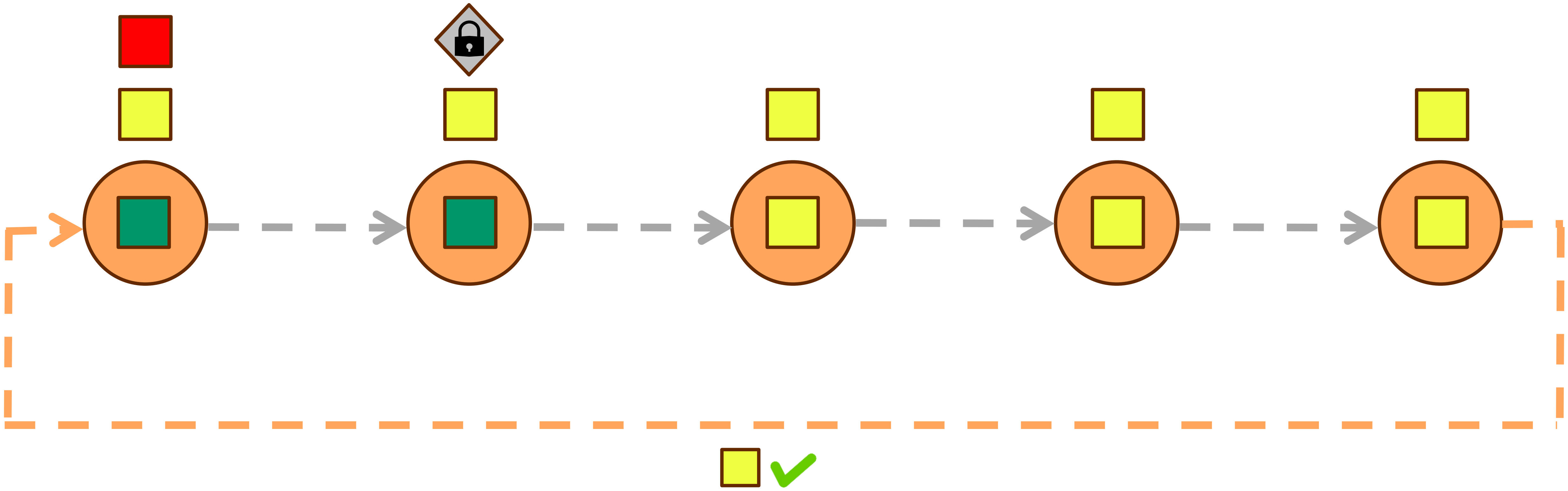


Local Linearizable Reads

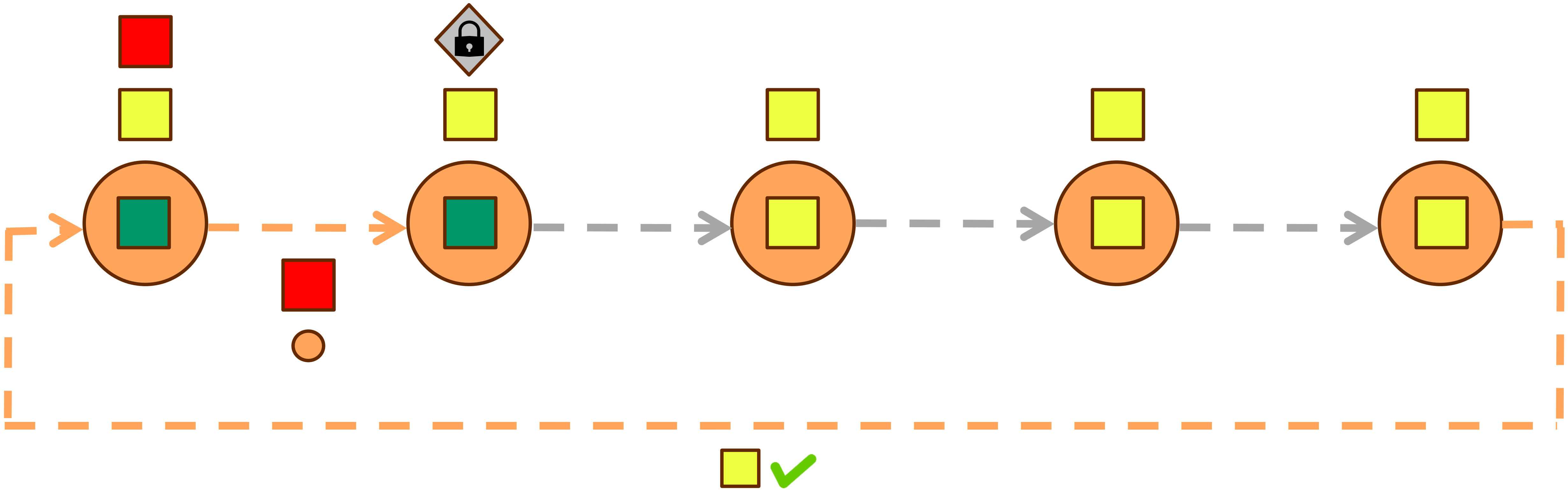


Response must:
- Contain all completed writes
- Contain all completed reads (🟨)

Local Linearizable Reads

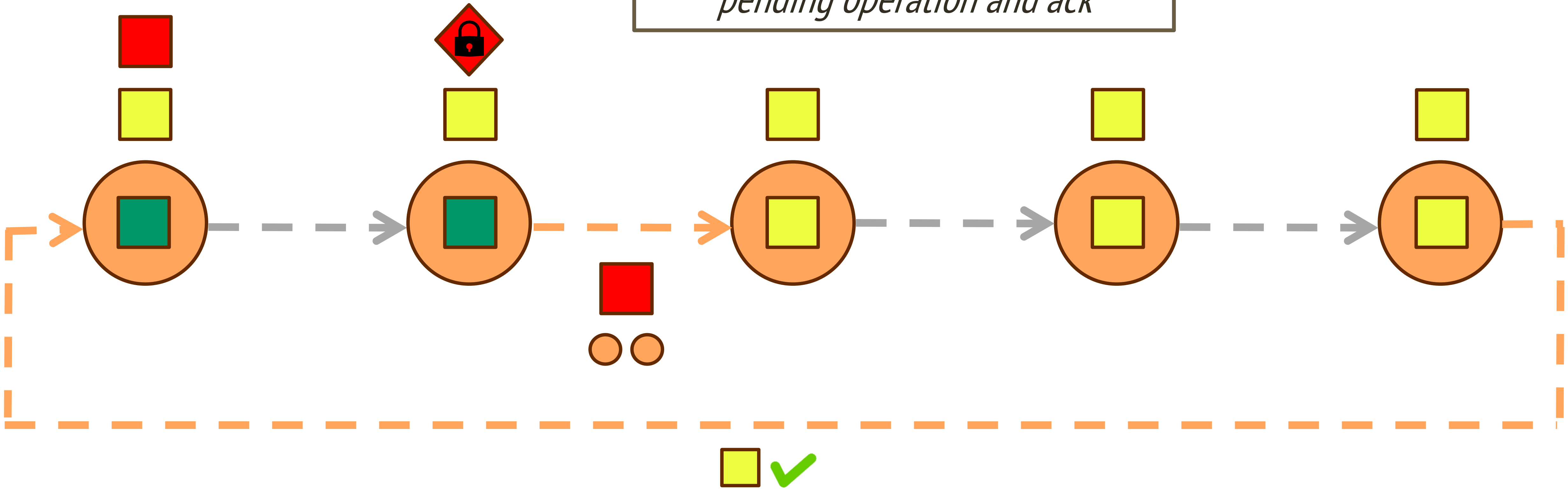


Local Linearizable Reads

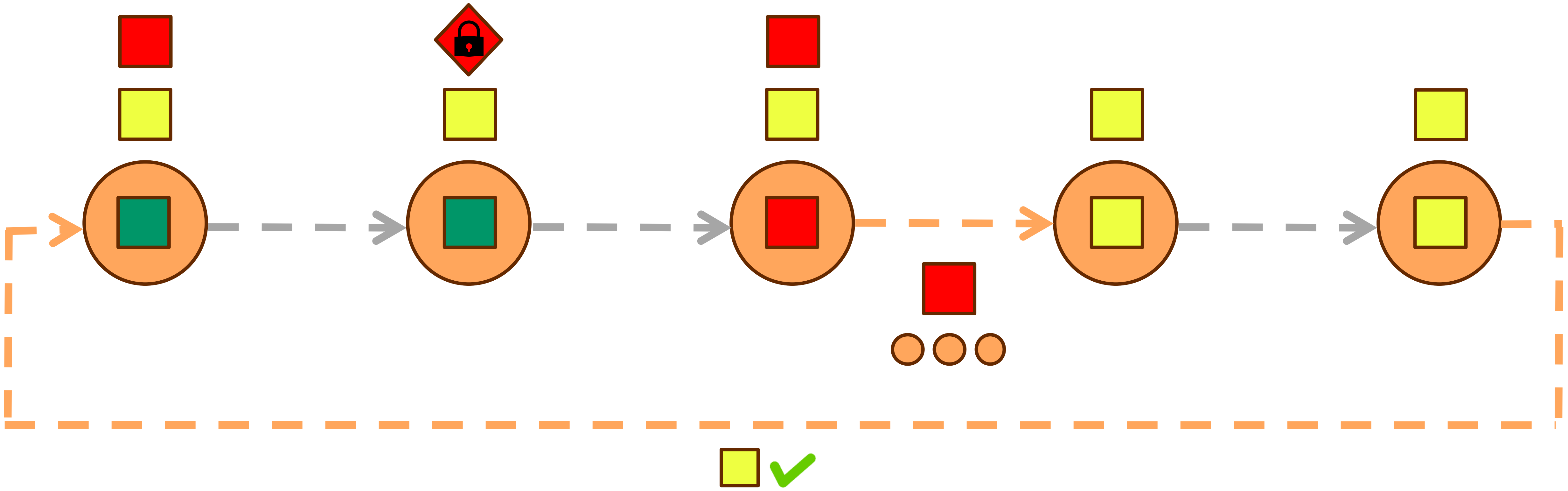


Local Linearizable Reads

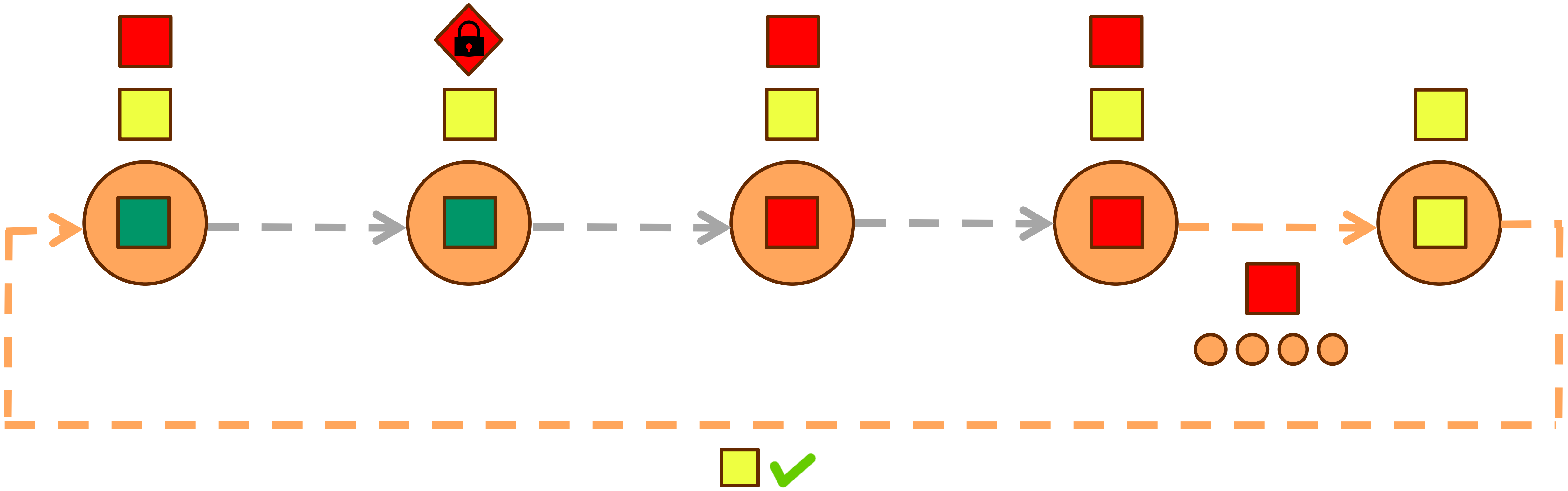
Red operation will "push" every pending operation and ack



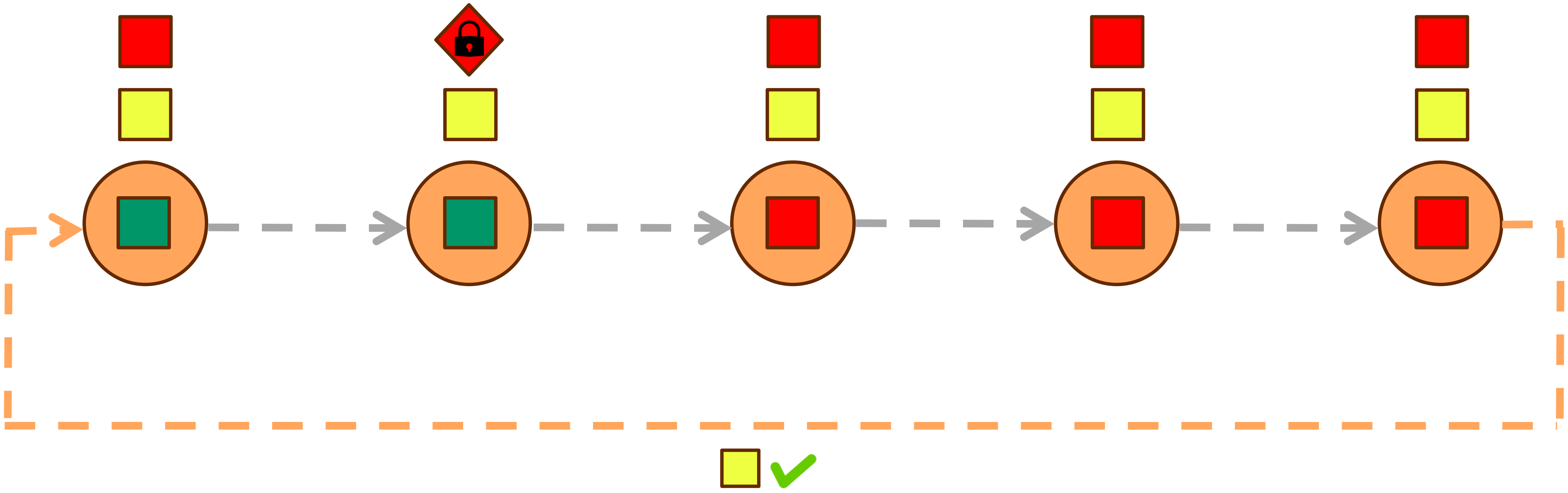
Local Linearizable Reads



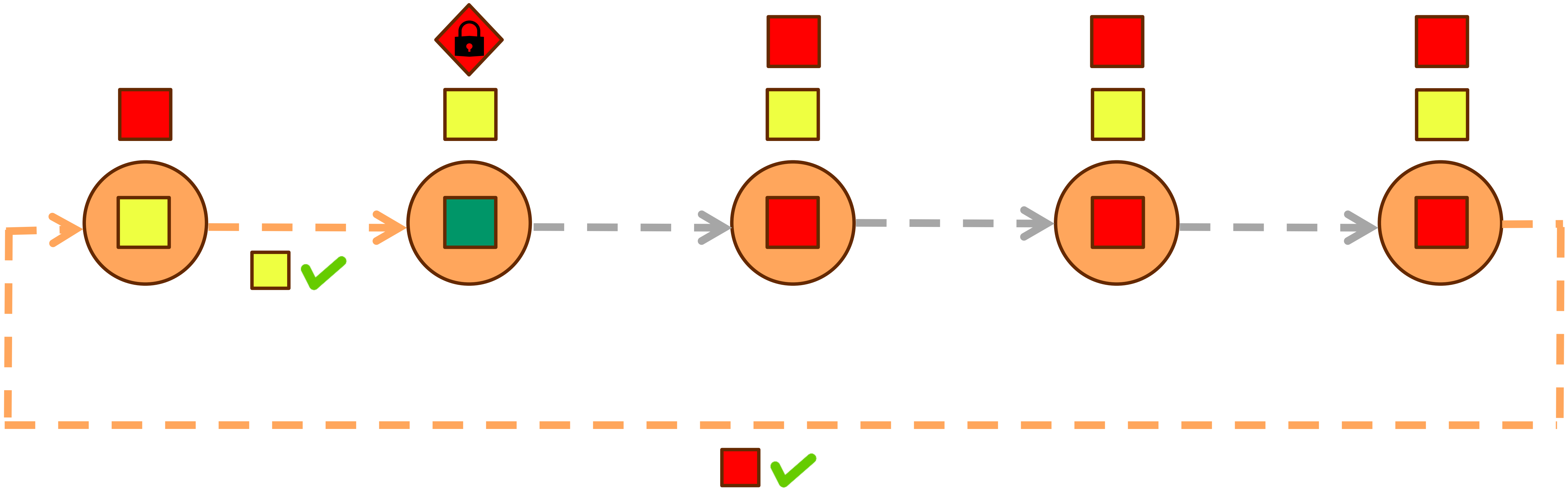
Local Linearizable Reads



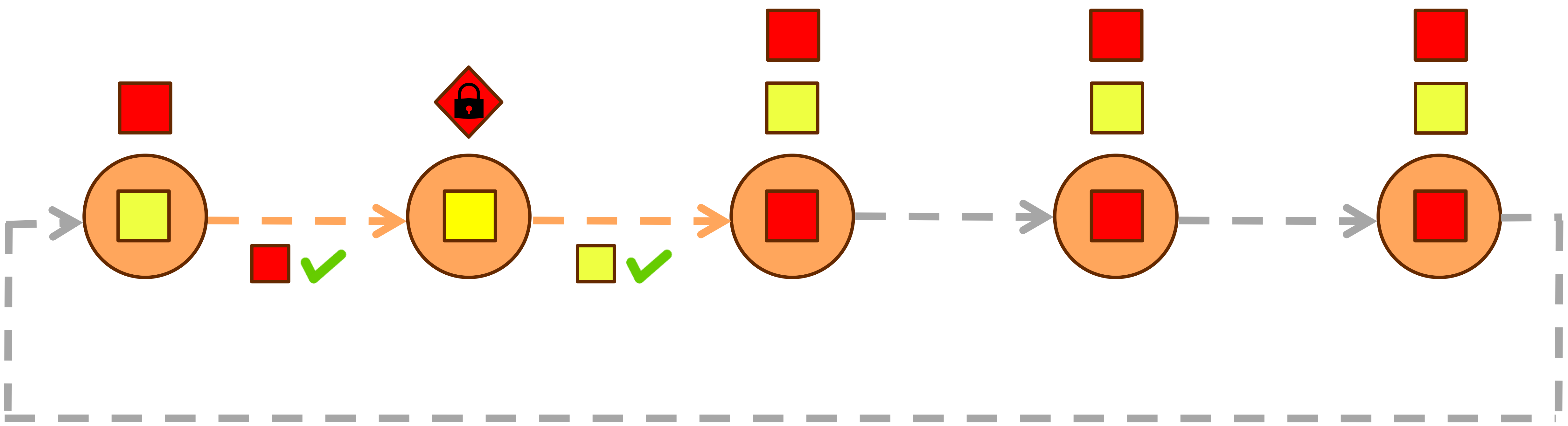
Local Linearizable Reads



Local Linearizable Reads



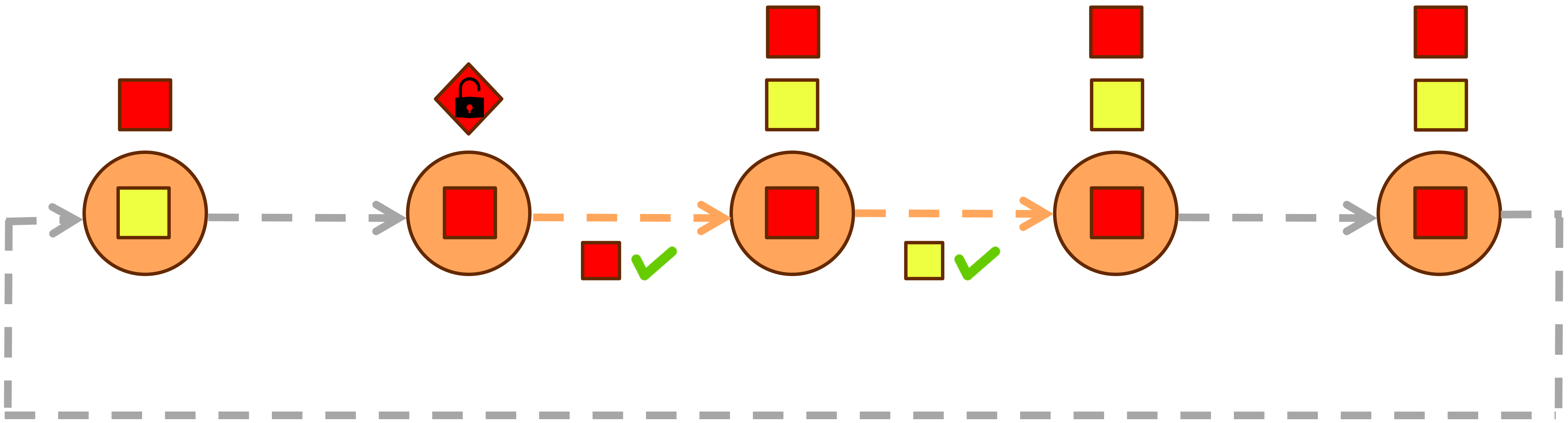
Local Linearizable Reads



Response must:

- Contain all completed writes
- Contain all completed reads (🟨)

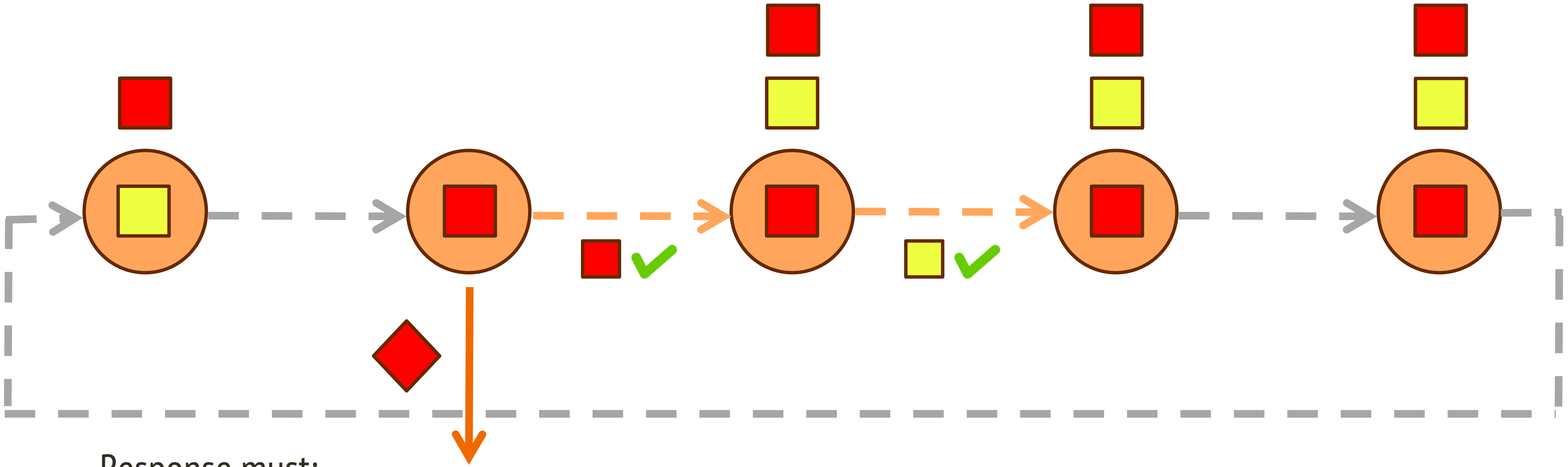
Local Linearizable Reads



Response must:

- Contain all completed writes
- Contain all completed reads (🟨)

Local Linearizable Reads



Response must:

- Contain all completed writes
- Contain all completed reads (🟨)

Local Linearizable Reads: Summary

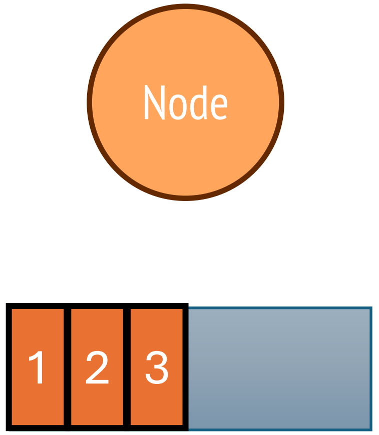
- Local read operations do not break linearizability
  - Ensures that all previously **completed reads and writes** are visible

- **No additional communication** steps are required
  - More **conservative** than required, but **unavoidable without coordination**

- **Periodic No-Op** messages ensure that reads are replied to during low write workloads

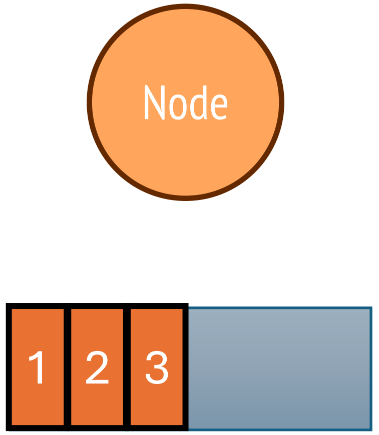
In Depth: The Log Representation

Log contains the ordered list of all operations seen



In Depth: The Log Representation

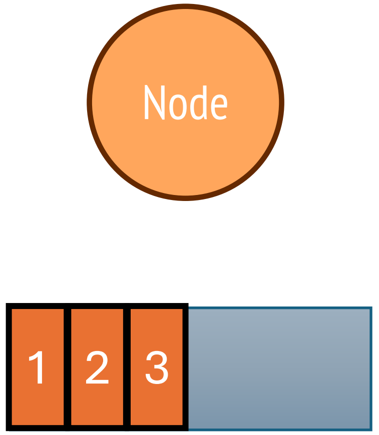
Log contains the ordered list of all operations seen



Each Node/Replica has its own view of the log

In Depth: The Log Representation

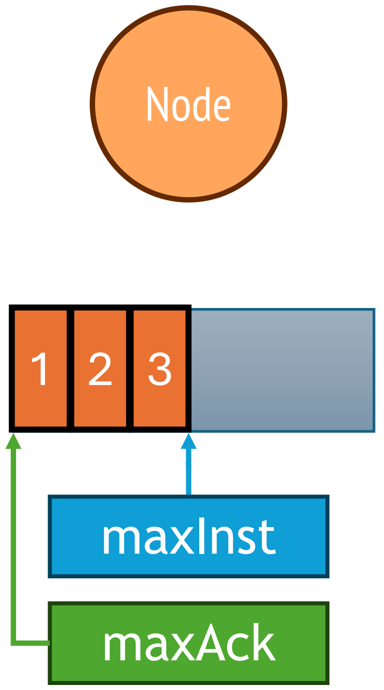
Log contains the ordered list of all operations seen



Each Node/Replica has its own view of the log

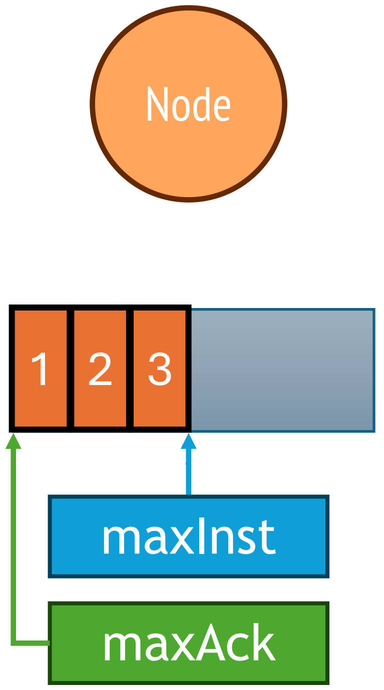
Each Instance writes to a new log index

In Depth: The Log Representation



In Depth: The Log Representation

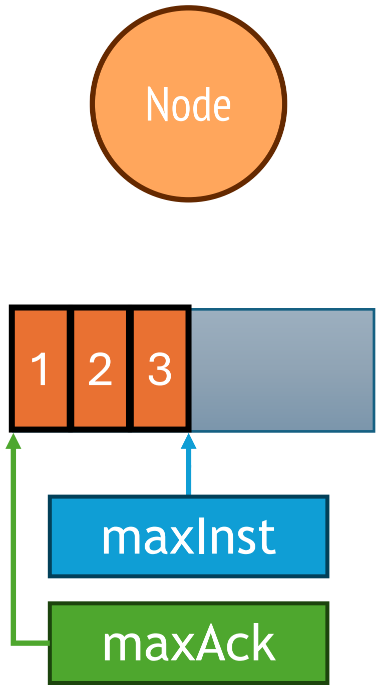
maxInst keeps track of replica's max seen instance



In Depth: The Log Representation

maxInst keeps track of replica's max seen instance

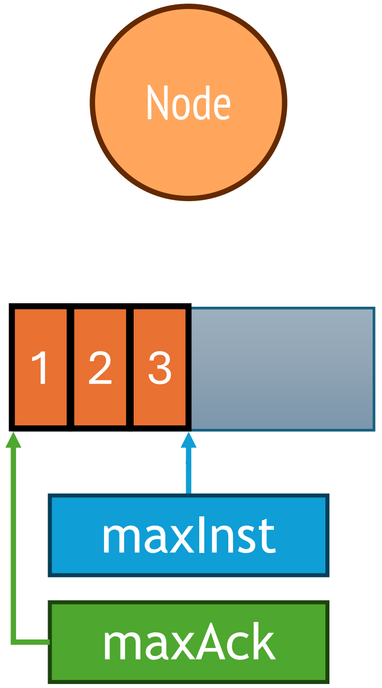
maxInst is essentially the length of the replica's log



In Depth: The Log Representation

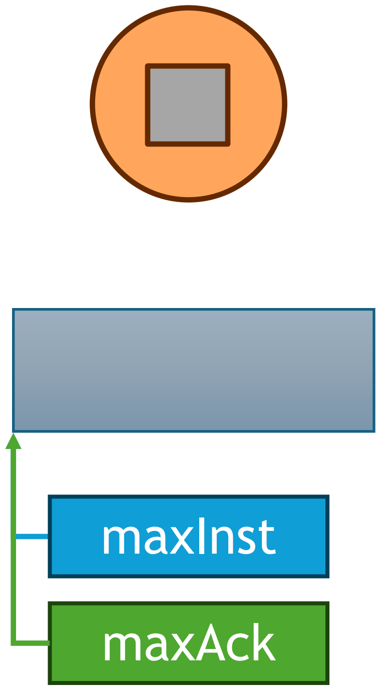
maxInst keeps track of replica's max seen instance

maxInst is essentially the length of the replica's log

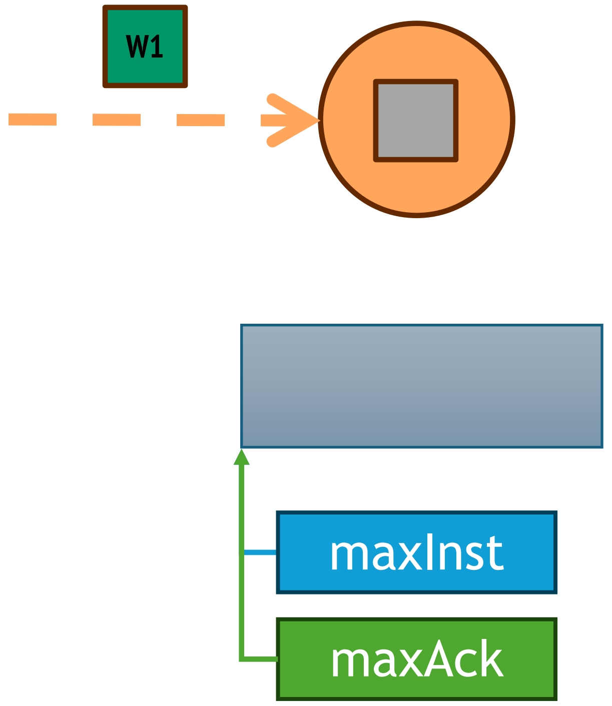


maxAck keeps track of replica's max seen acknowledgment

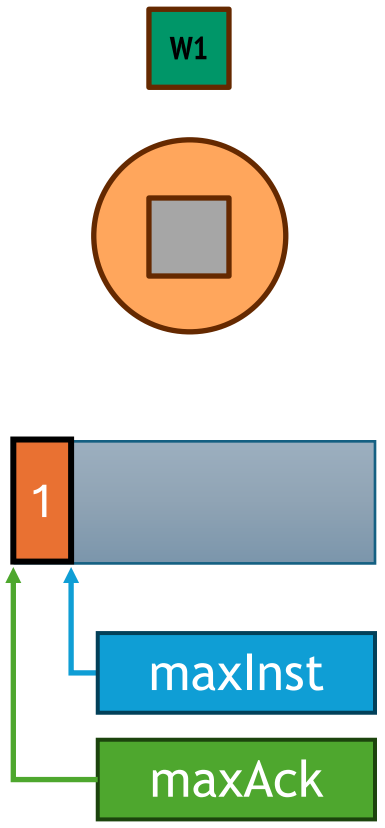
In Depth: The Log Representation



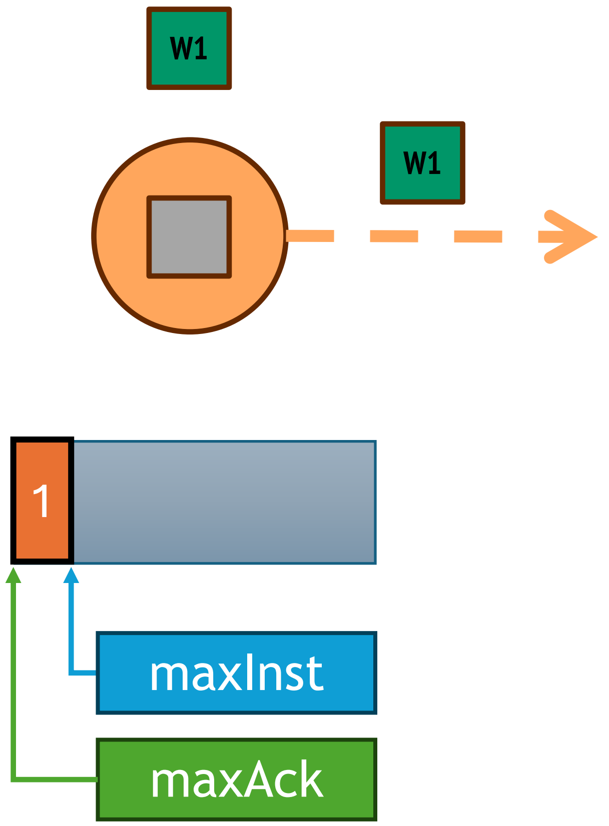
In Depth: The Log Representation



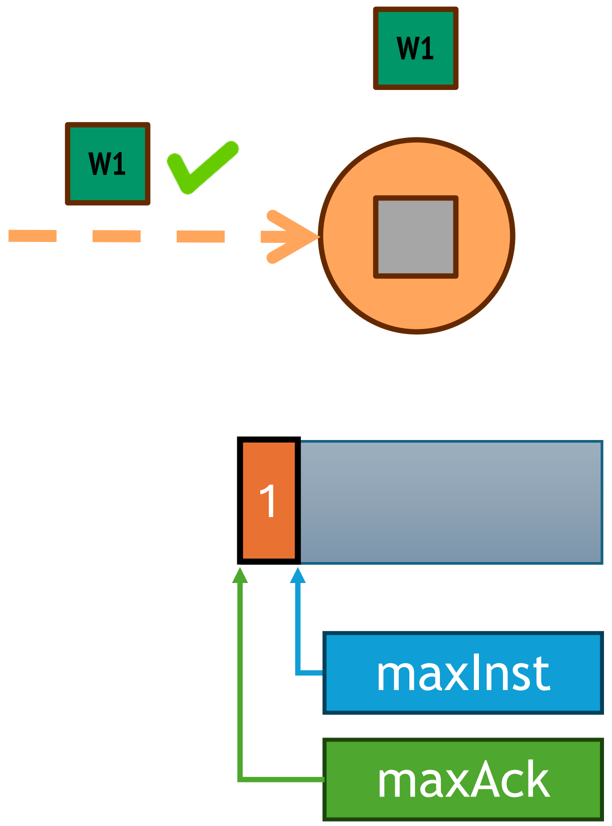
In Depth: The Log Representation



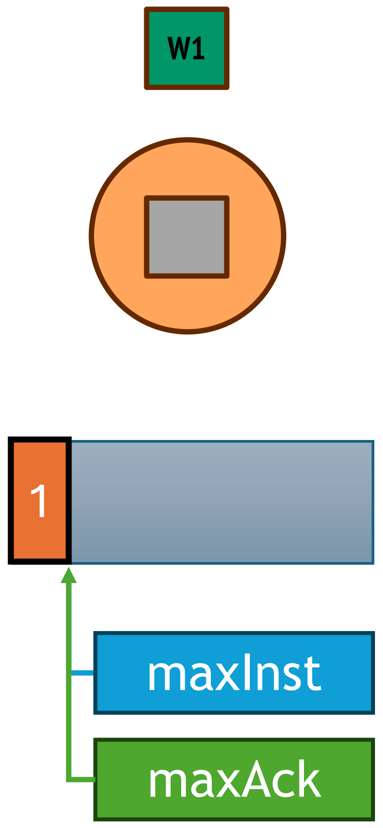
In Depth: The Log Representation



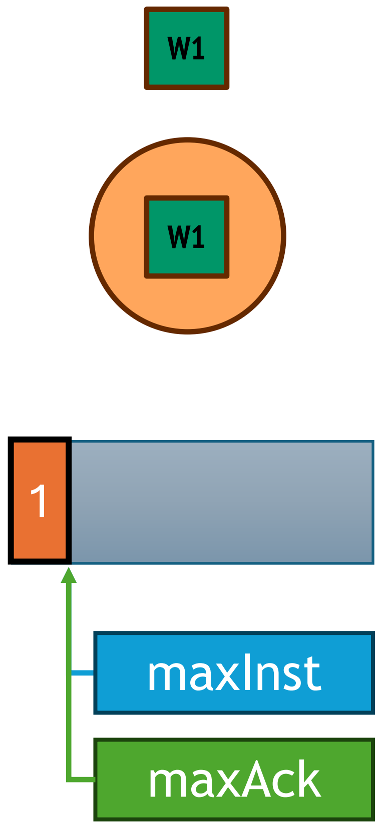
In Depth: The Log Representation



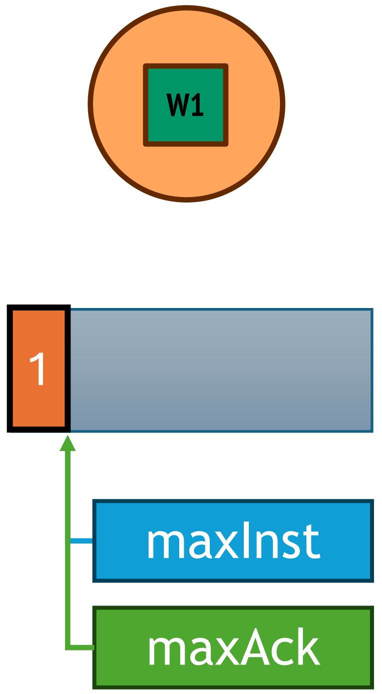
In Depth: The Log Representation



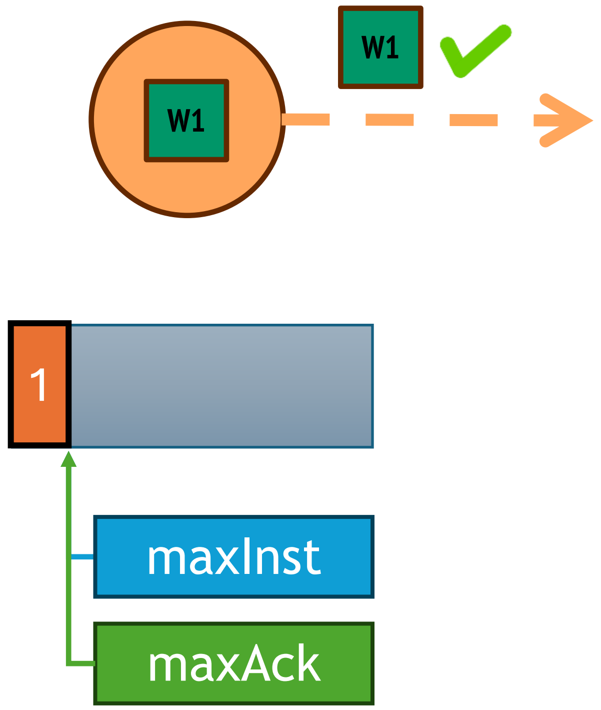
In Depth: The Log Representation



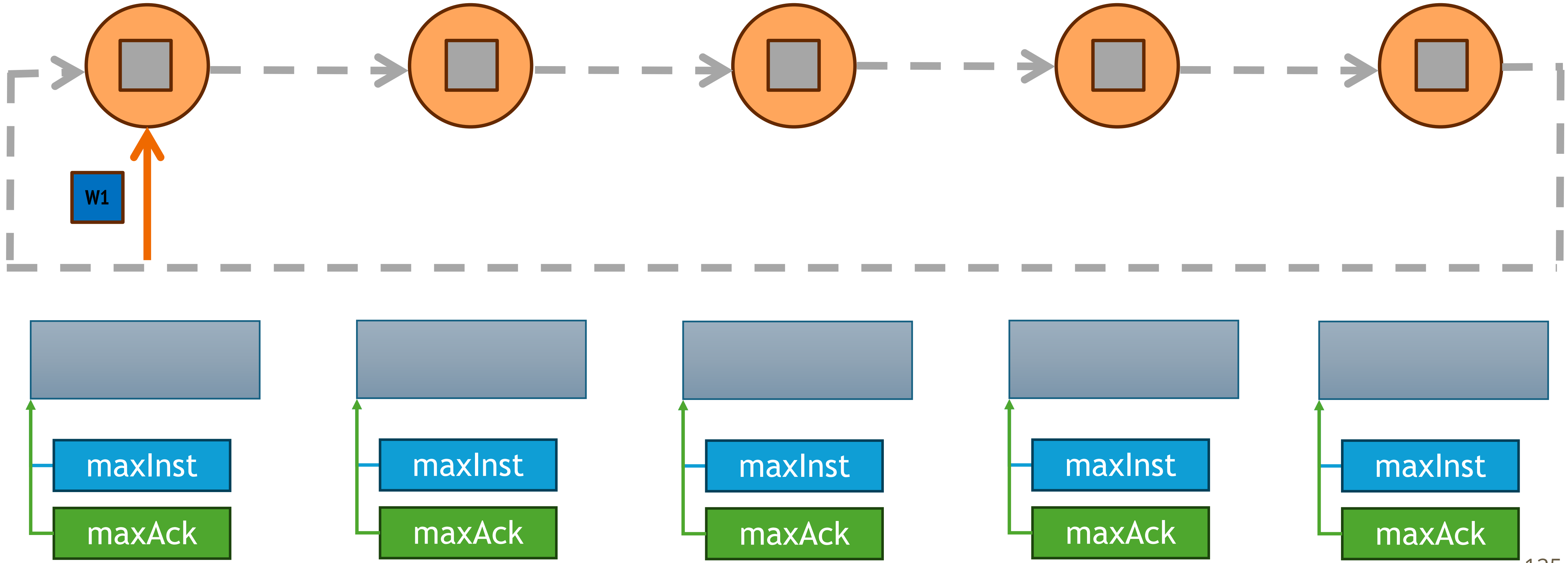
In Depth: The Log Representation



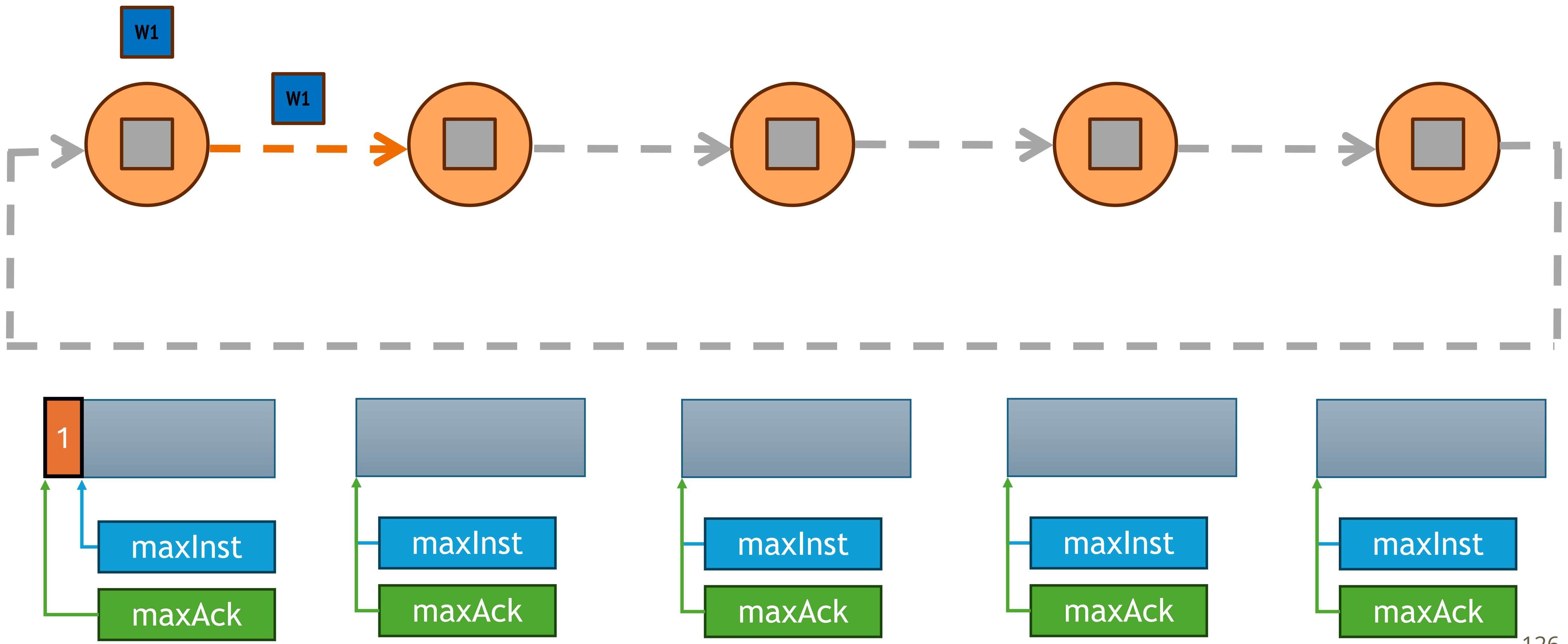
In Depth: The Log Representation



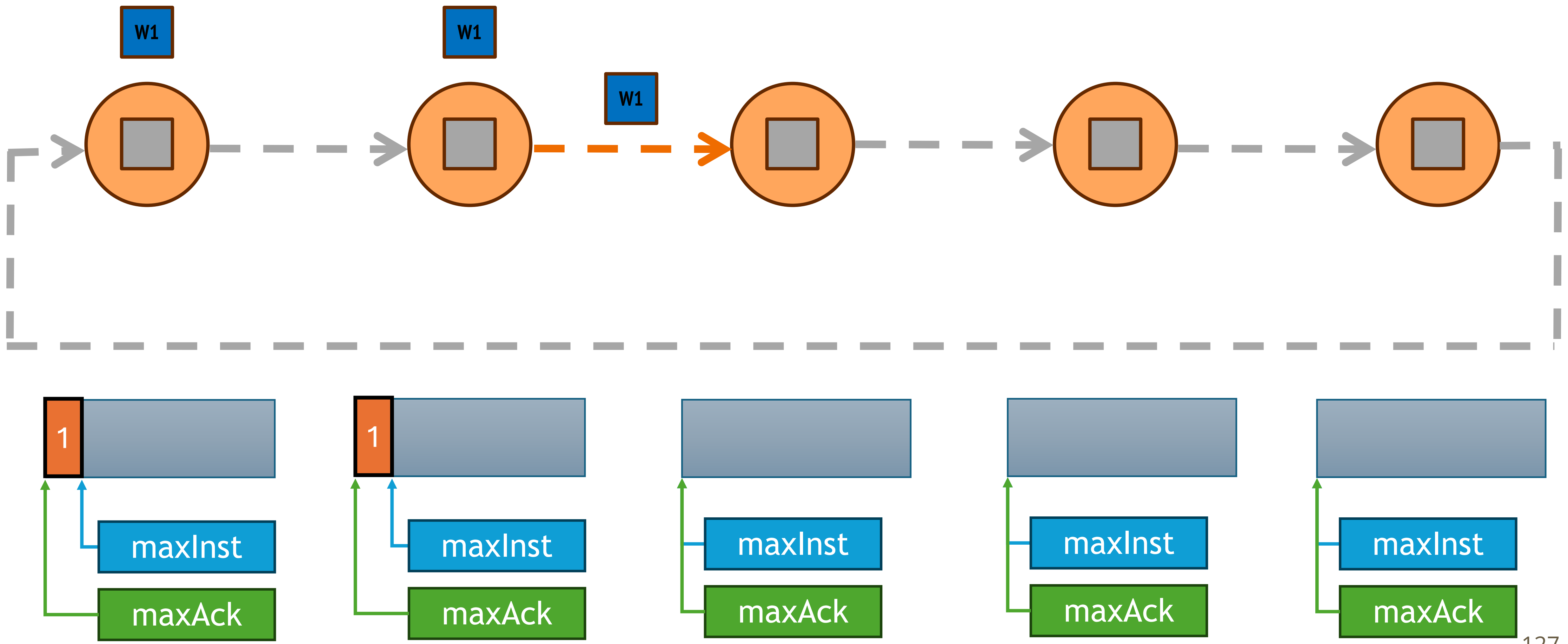
In Depth: Writes



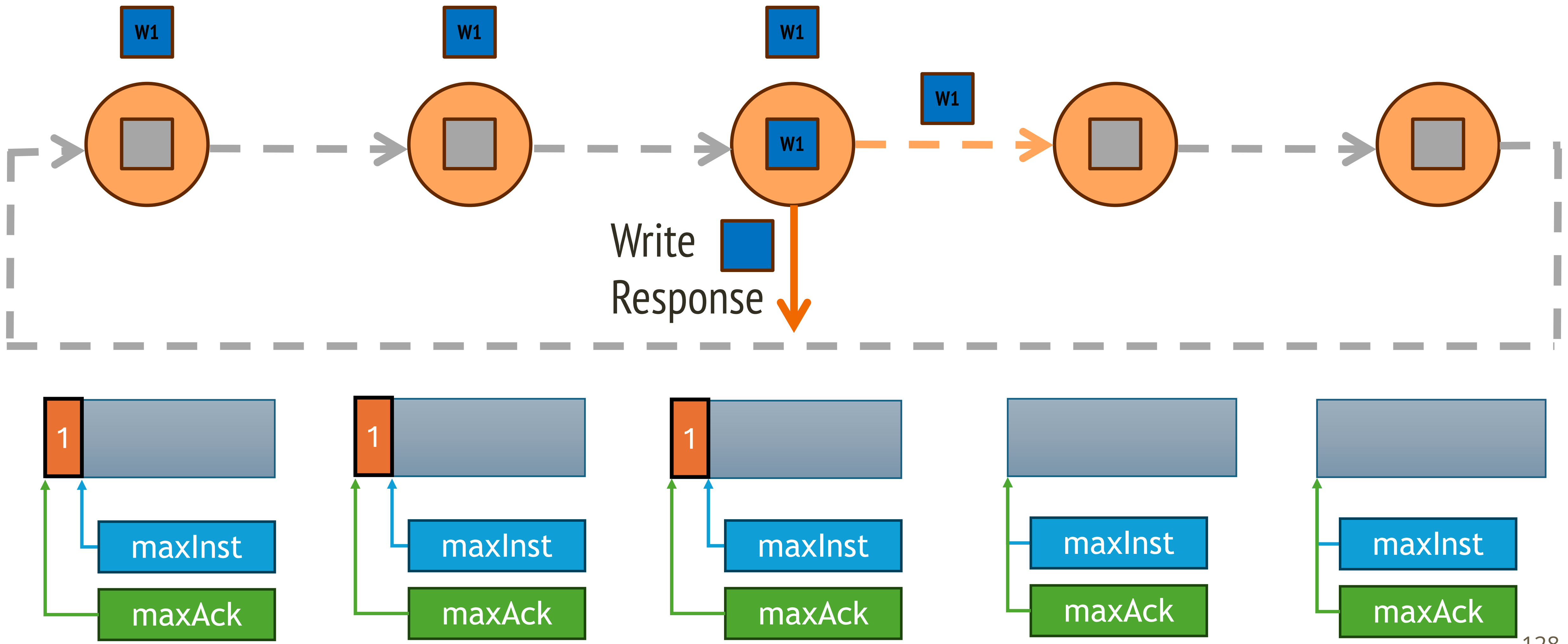
In Depth: Writes



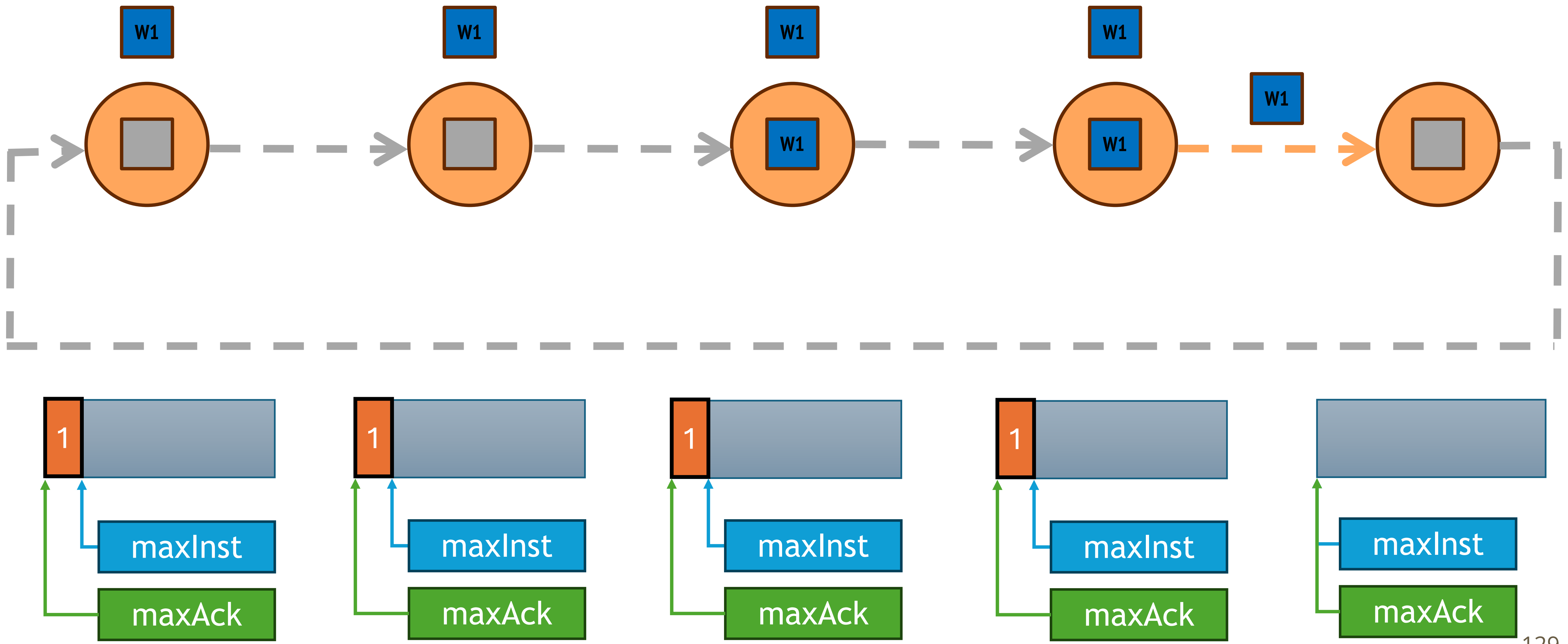
In Depth: Writes



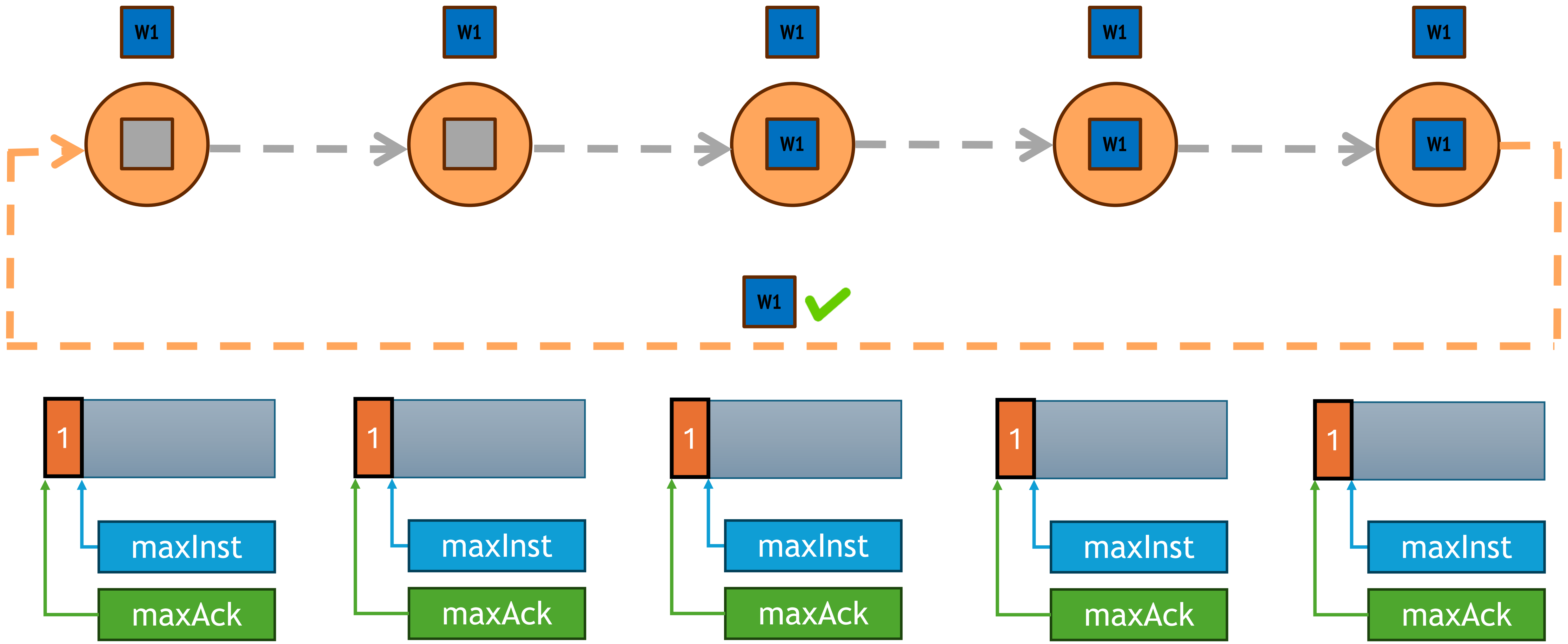
In Depth: Writes



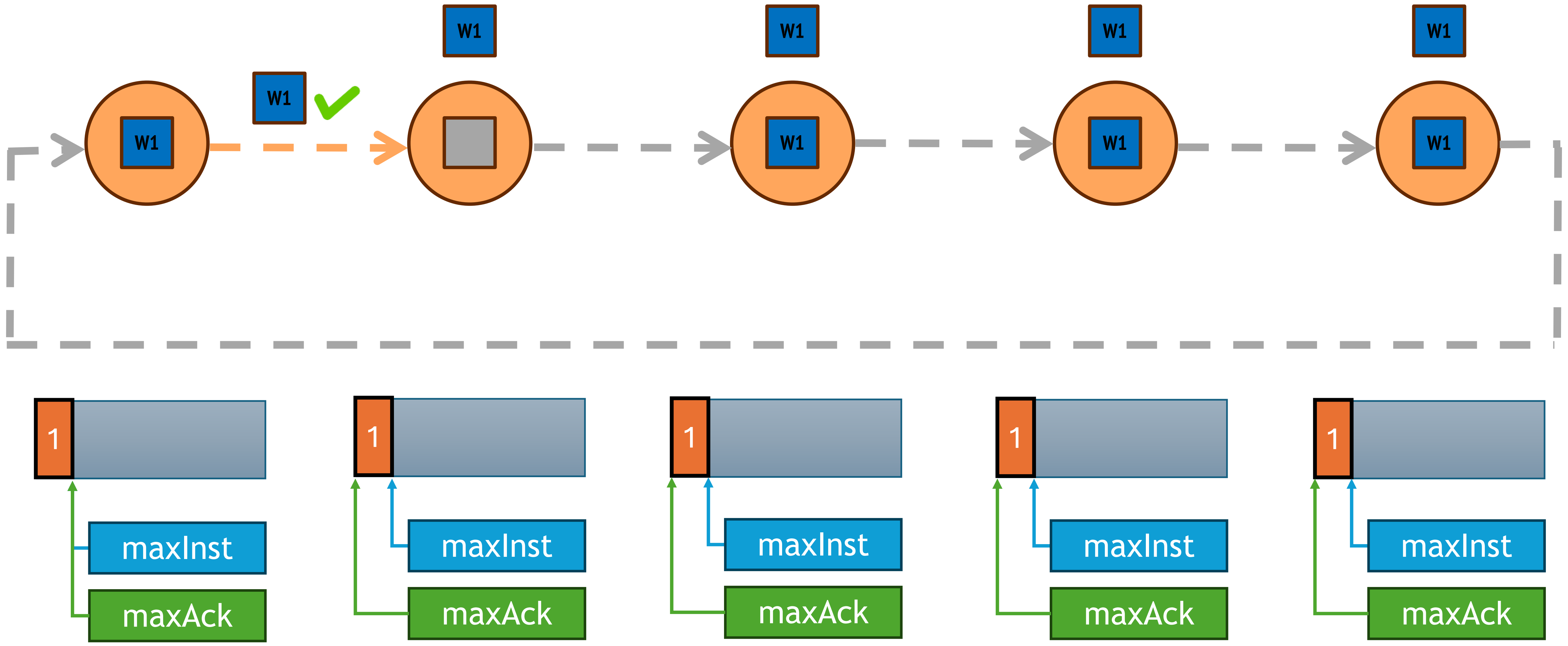
In Depth: Writes



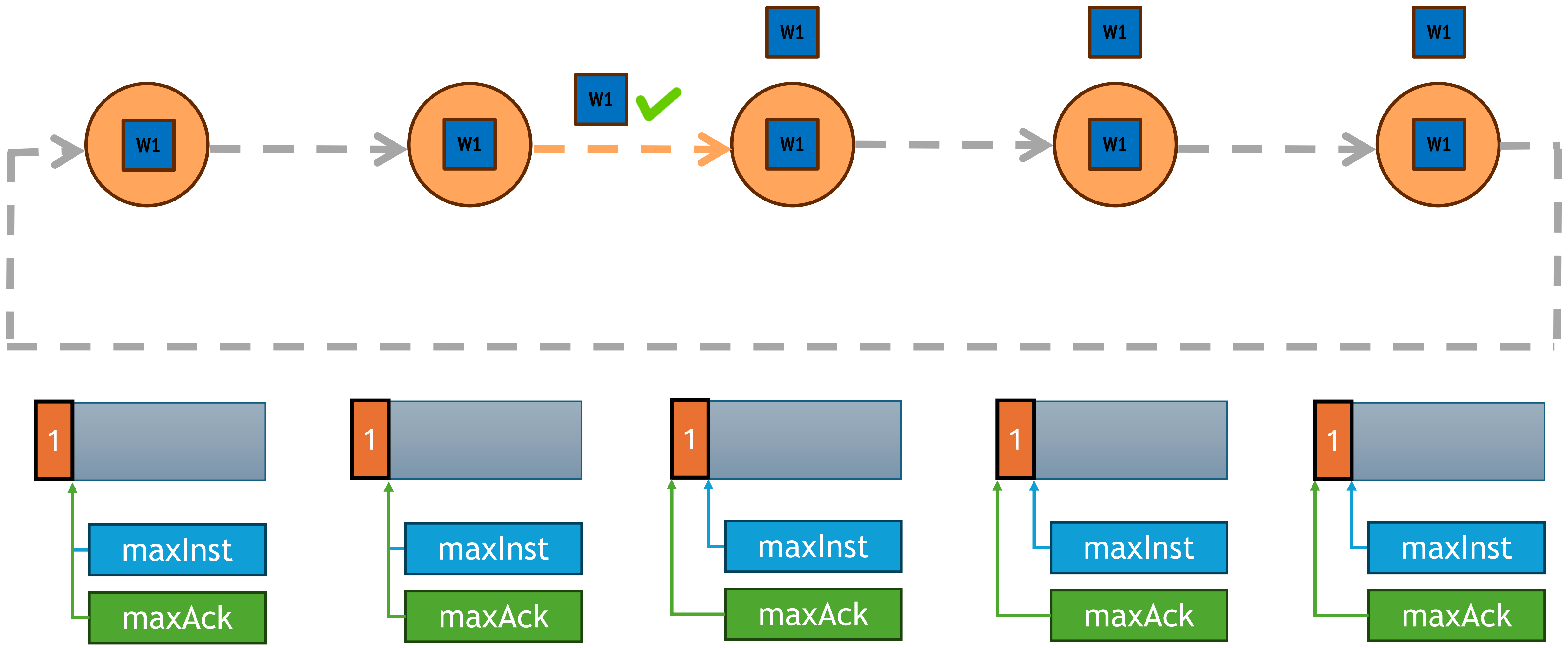
In Depth: Writes



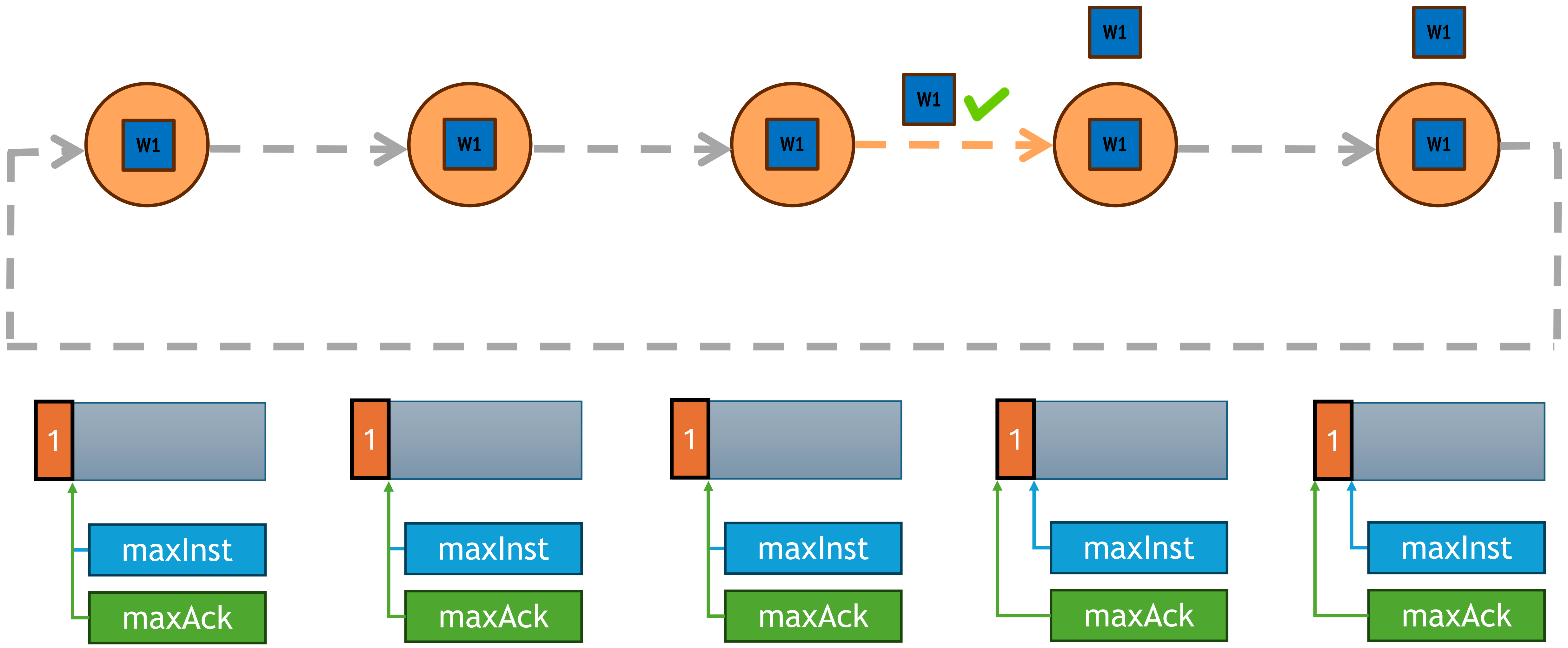
In Depth: Writes



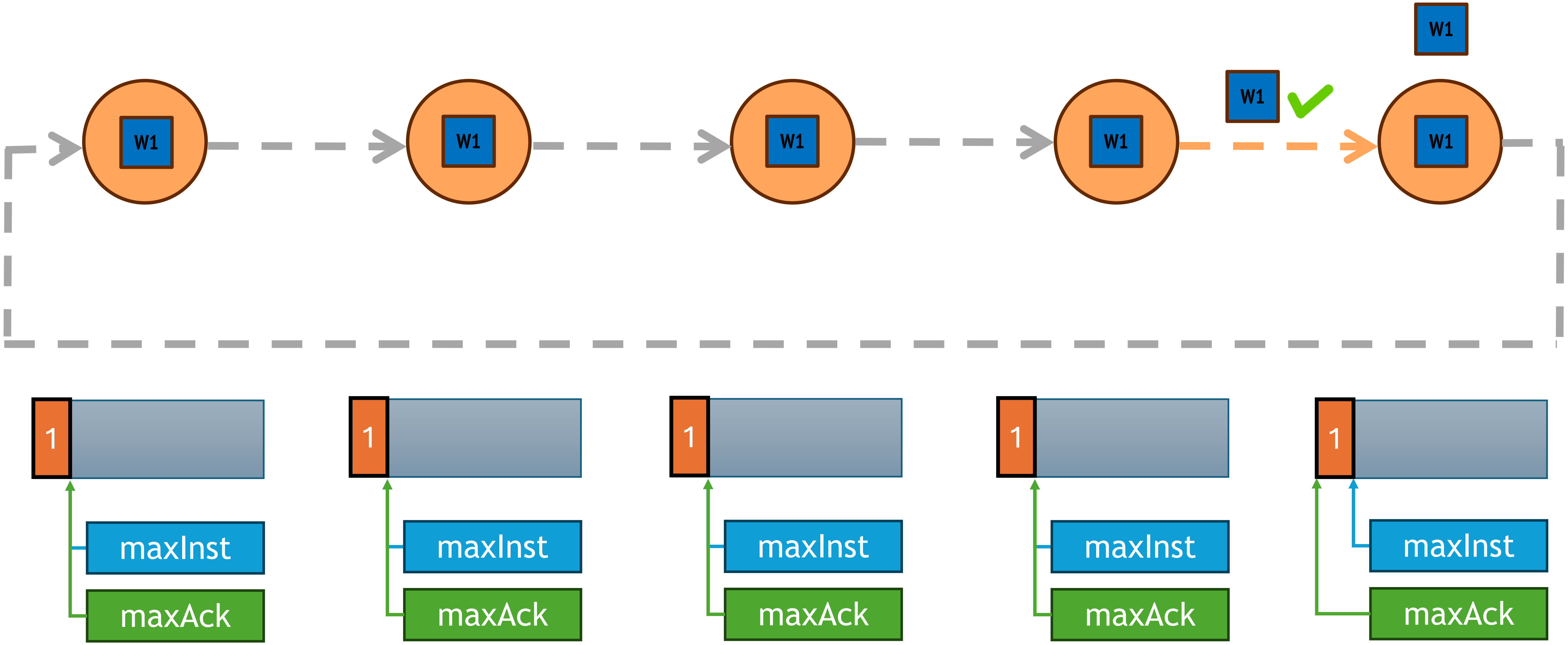
In Depth: Writes



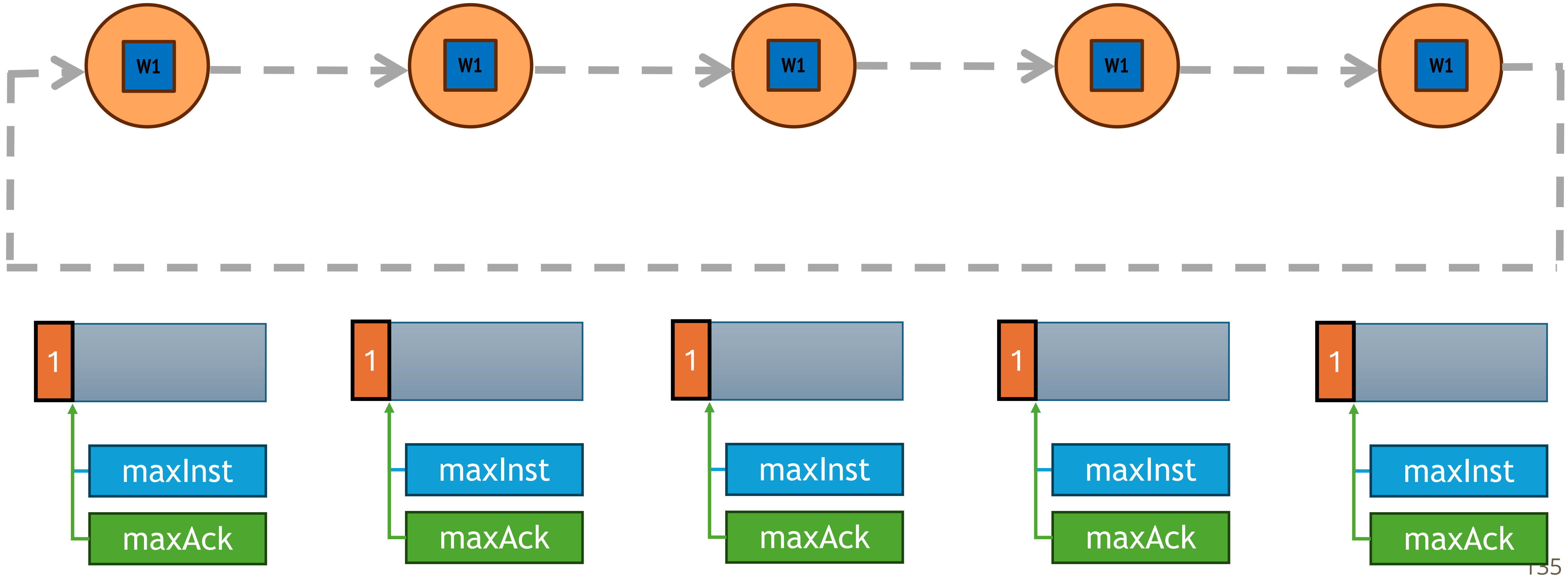
In Depth: Writes



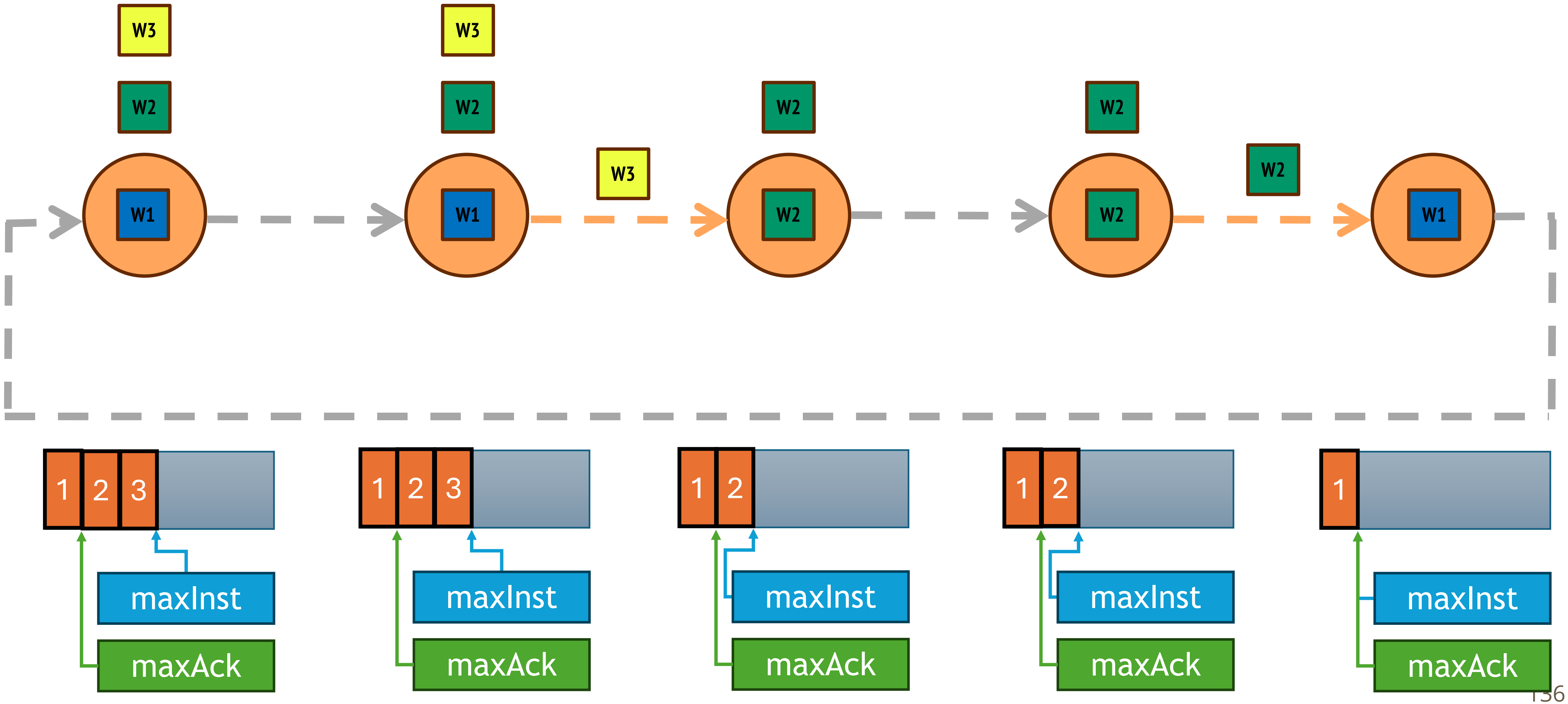
In Depth: Writes



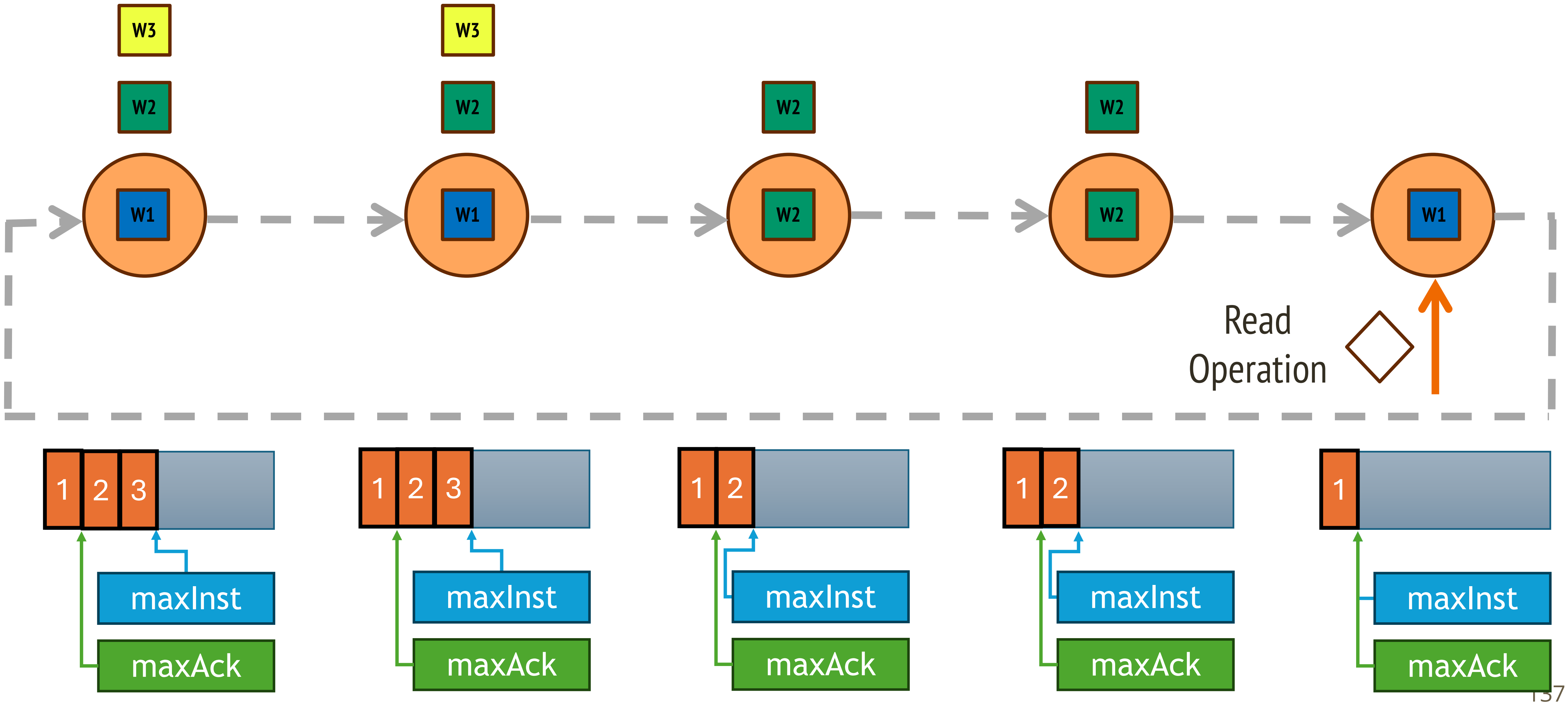
In Depth: Writes



In Depth: Reads



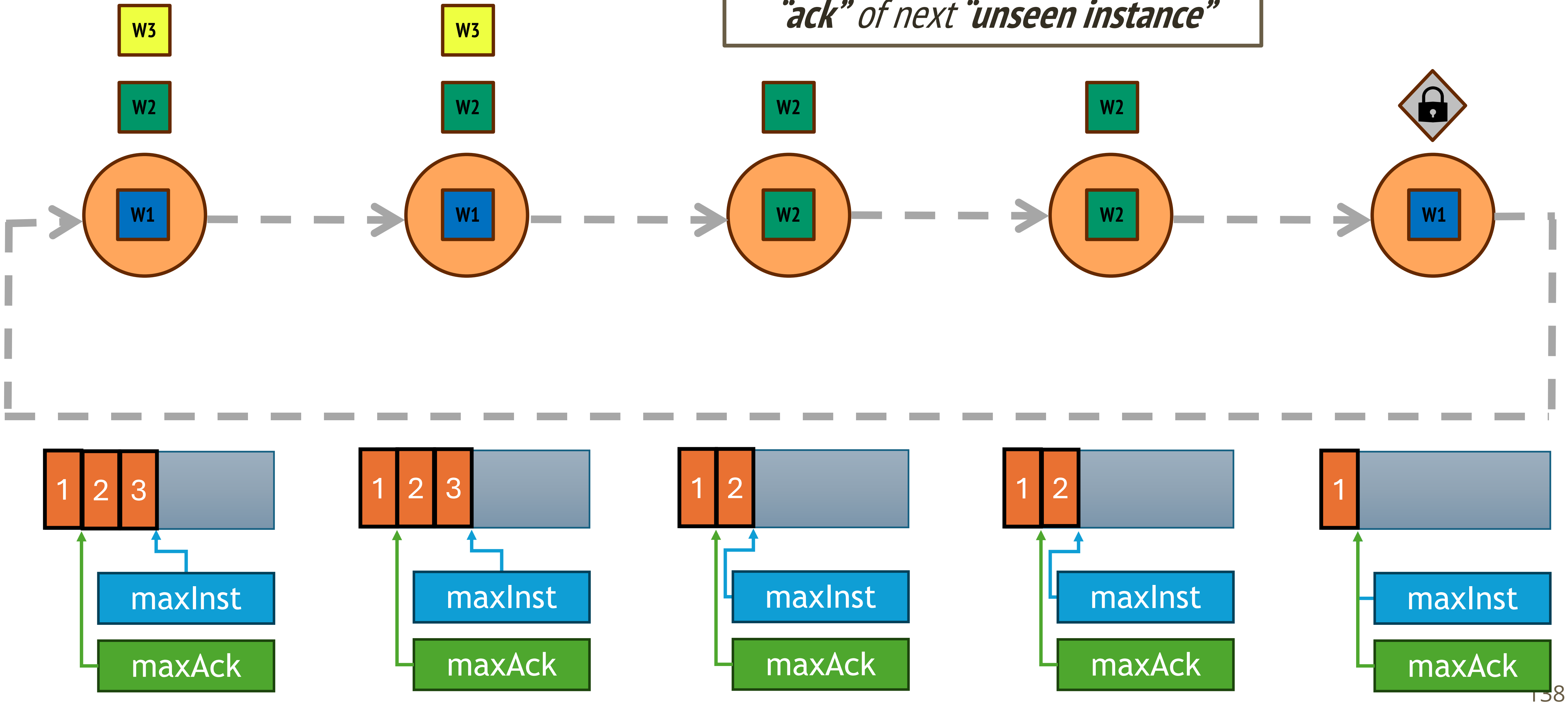
In Depth: Reads



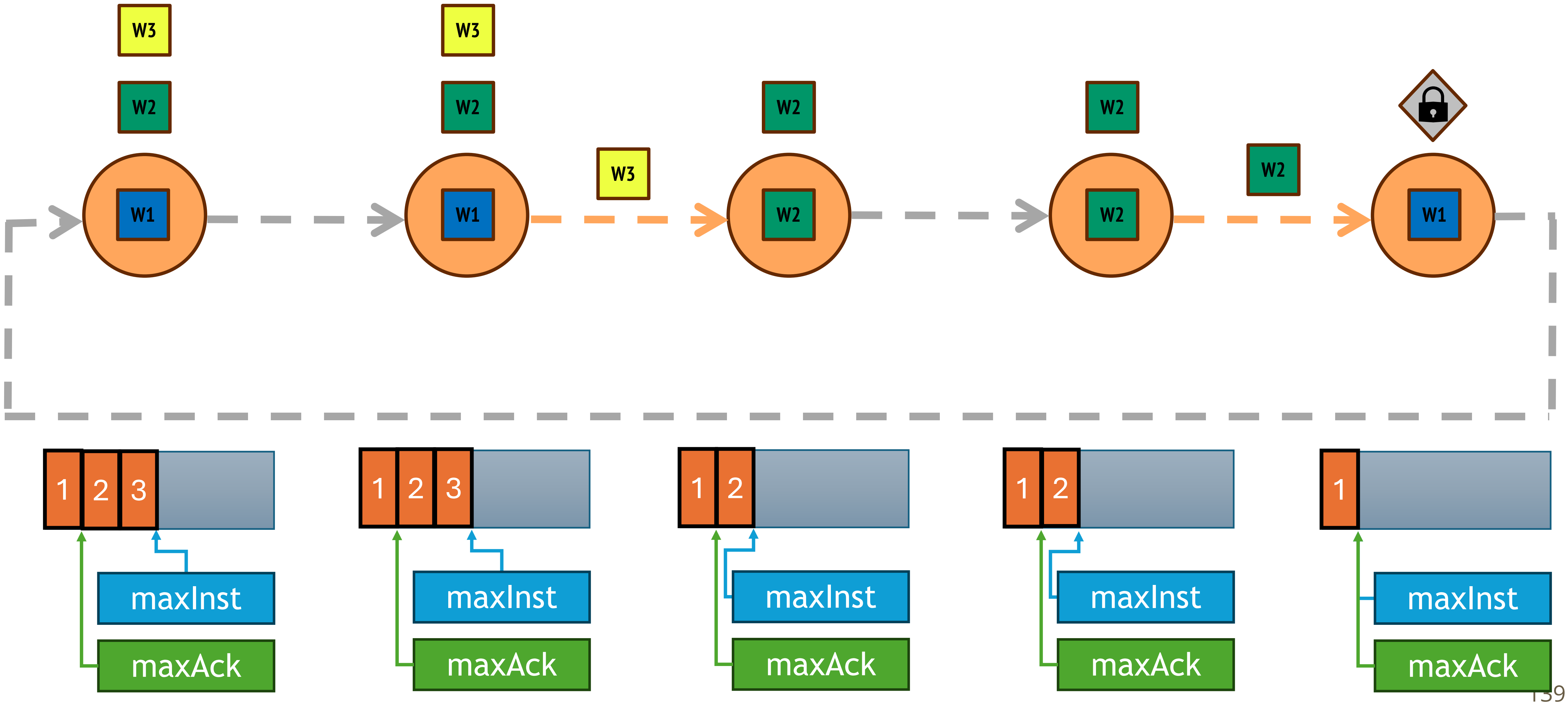
In Depth: Reads

Reads wait for

*"**ack**" of next "**unseen instance**"*

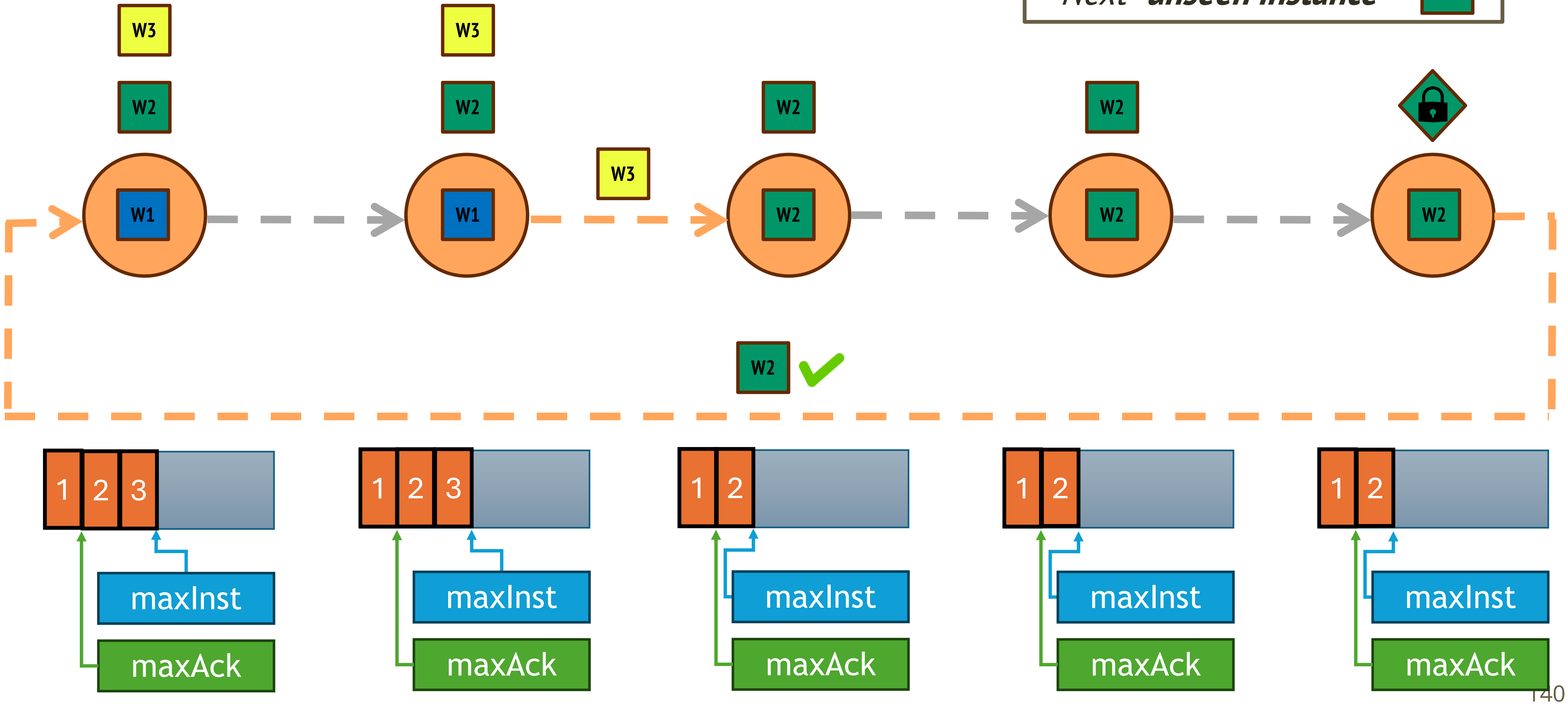


In Depth: Reads

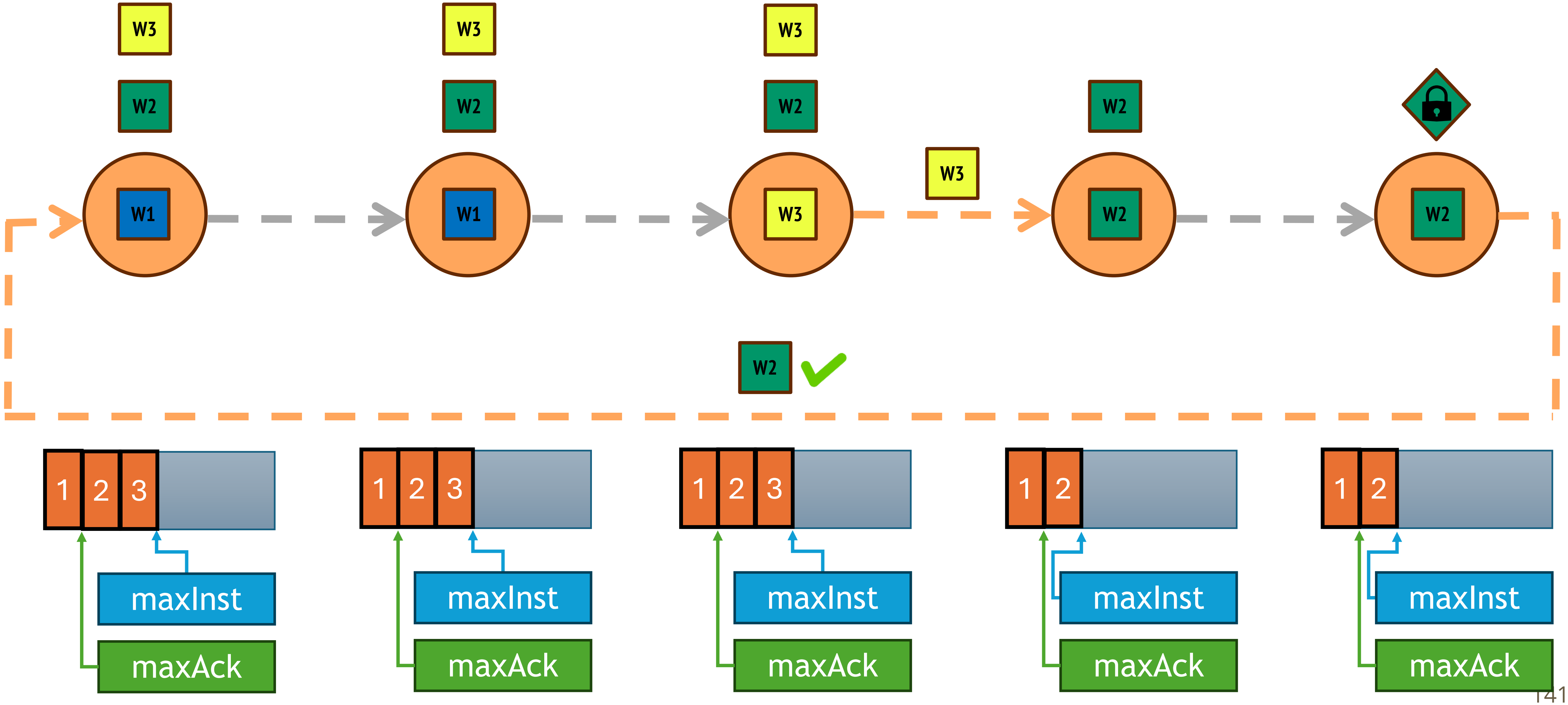


In Depth: Reads

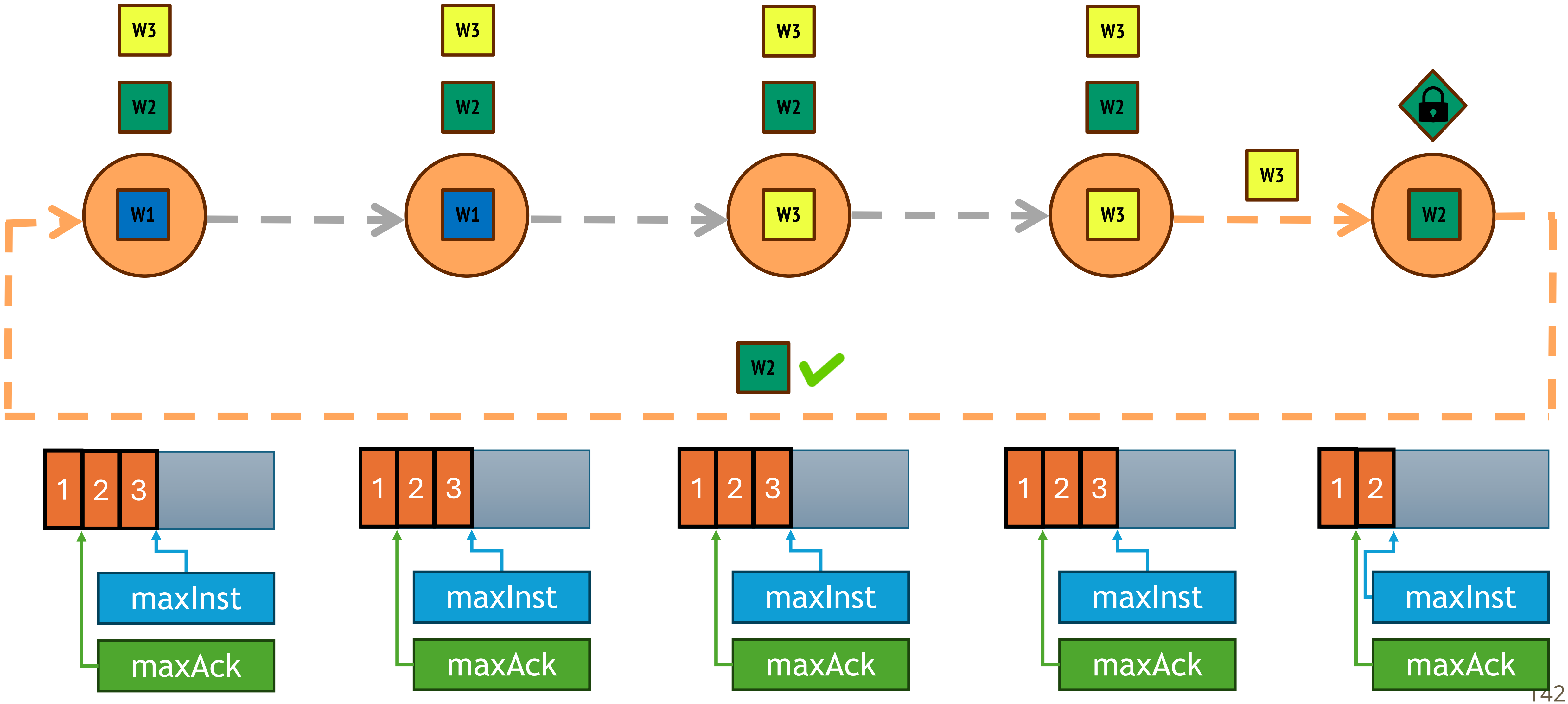
Next "unseen instance" - W2



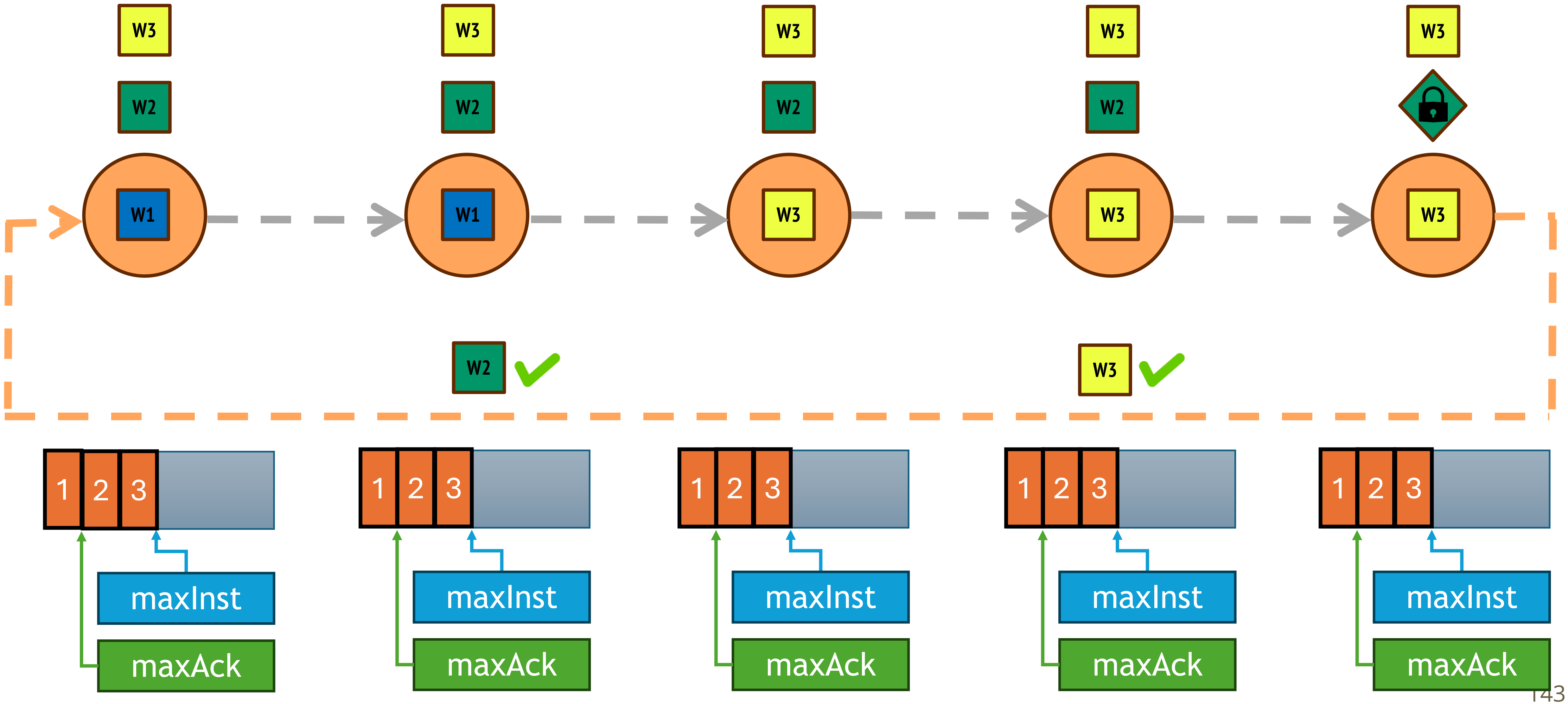
In Depth: Reads



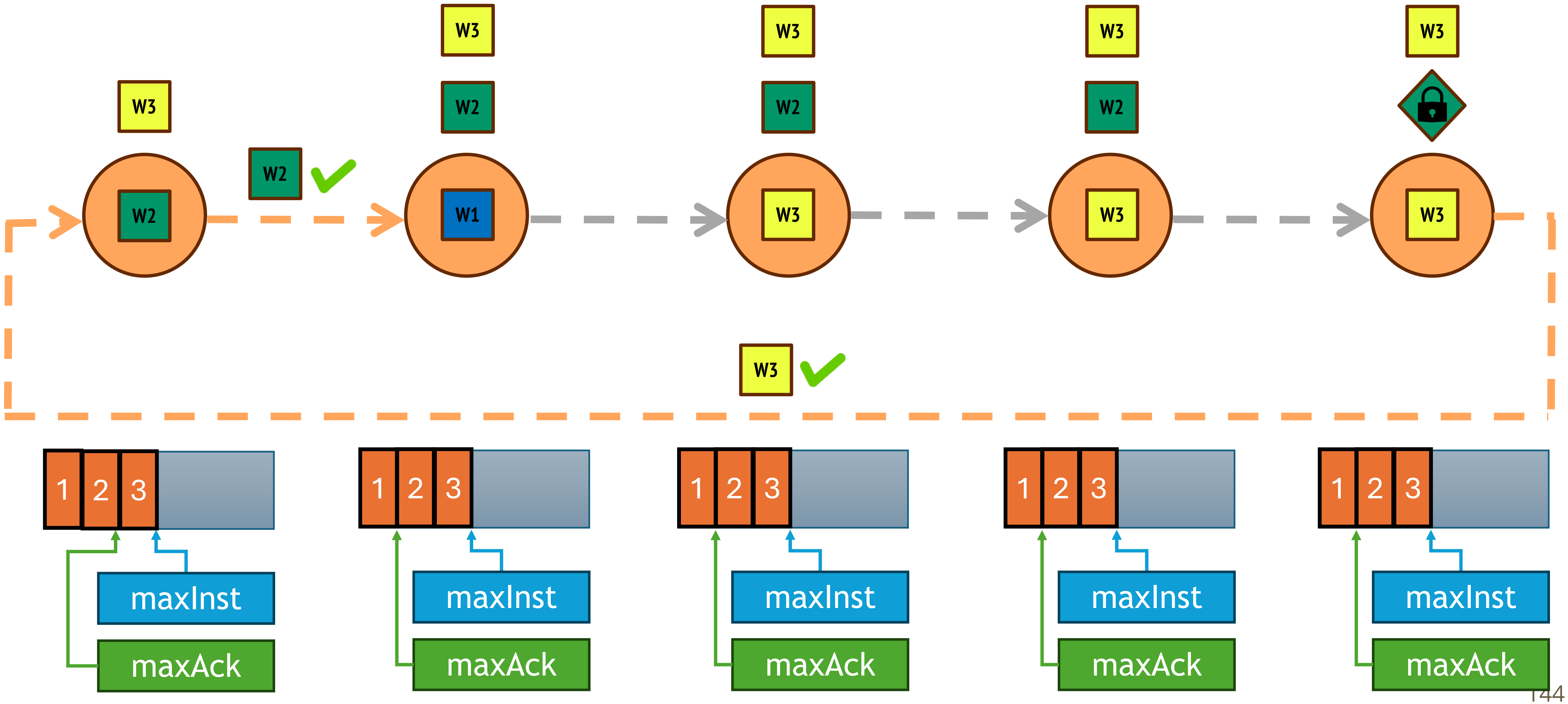
In Depth: Reads



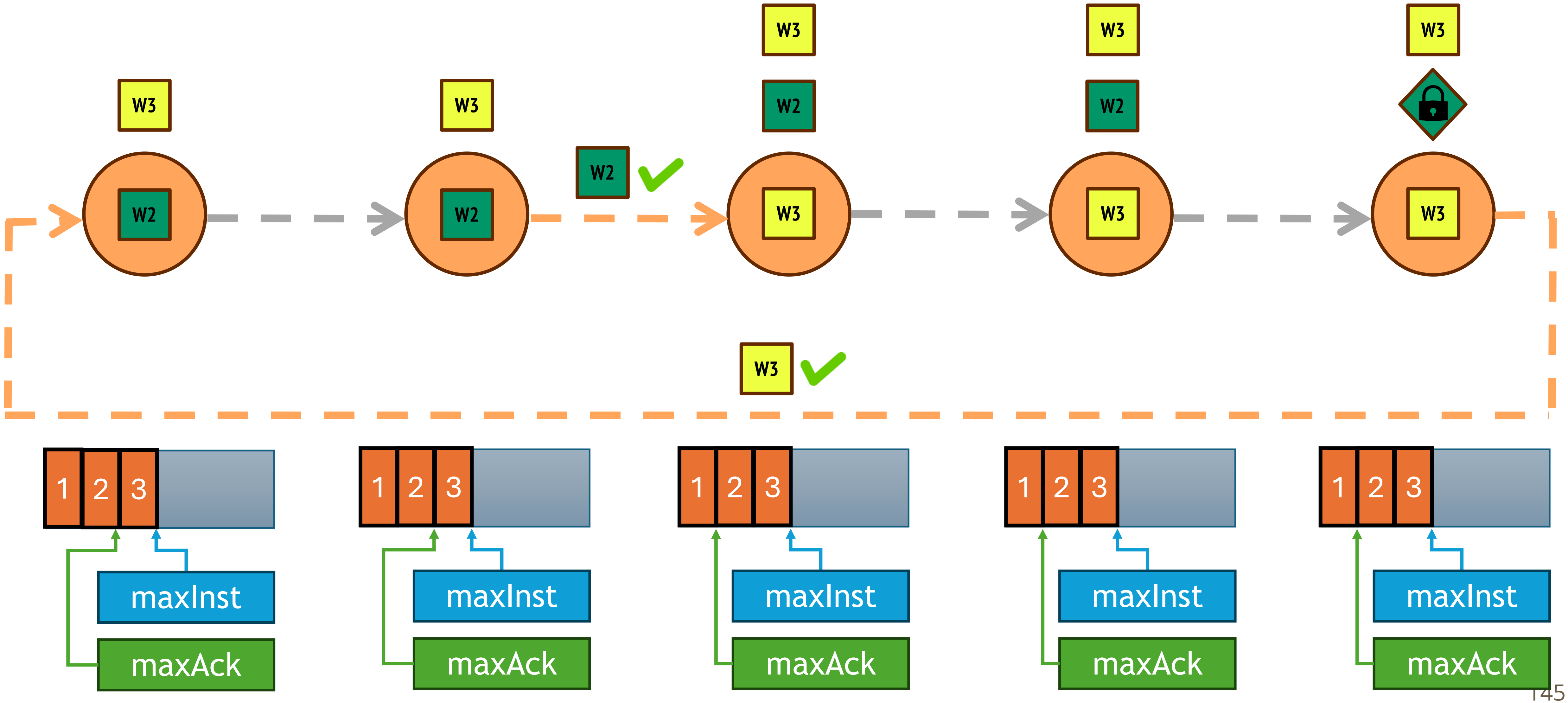
In Depth: Reads



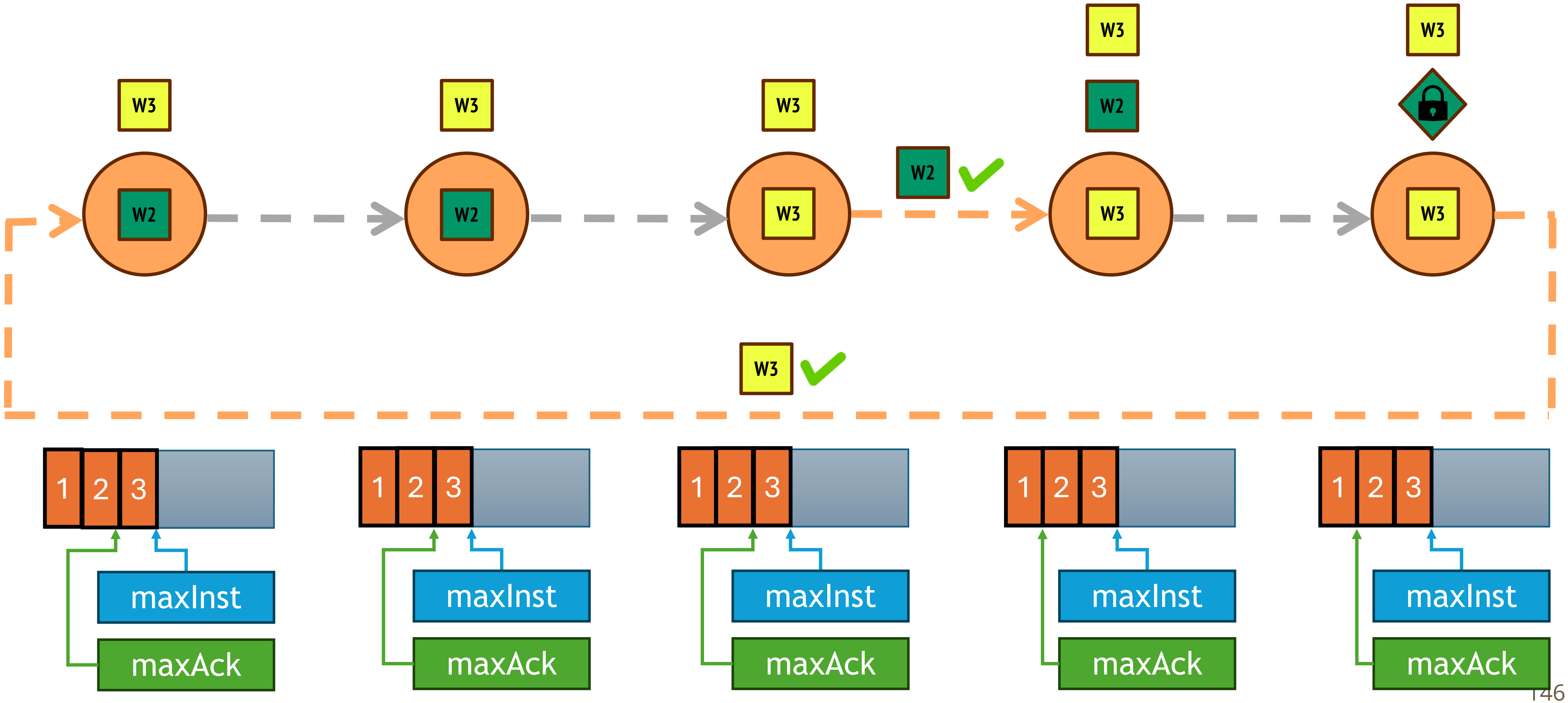
In Depth: Reads



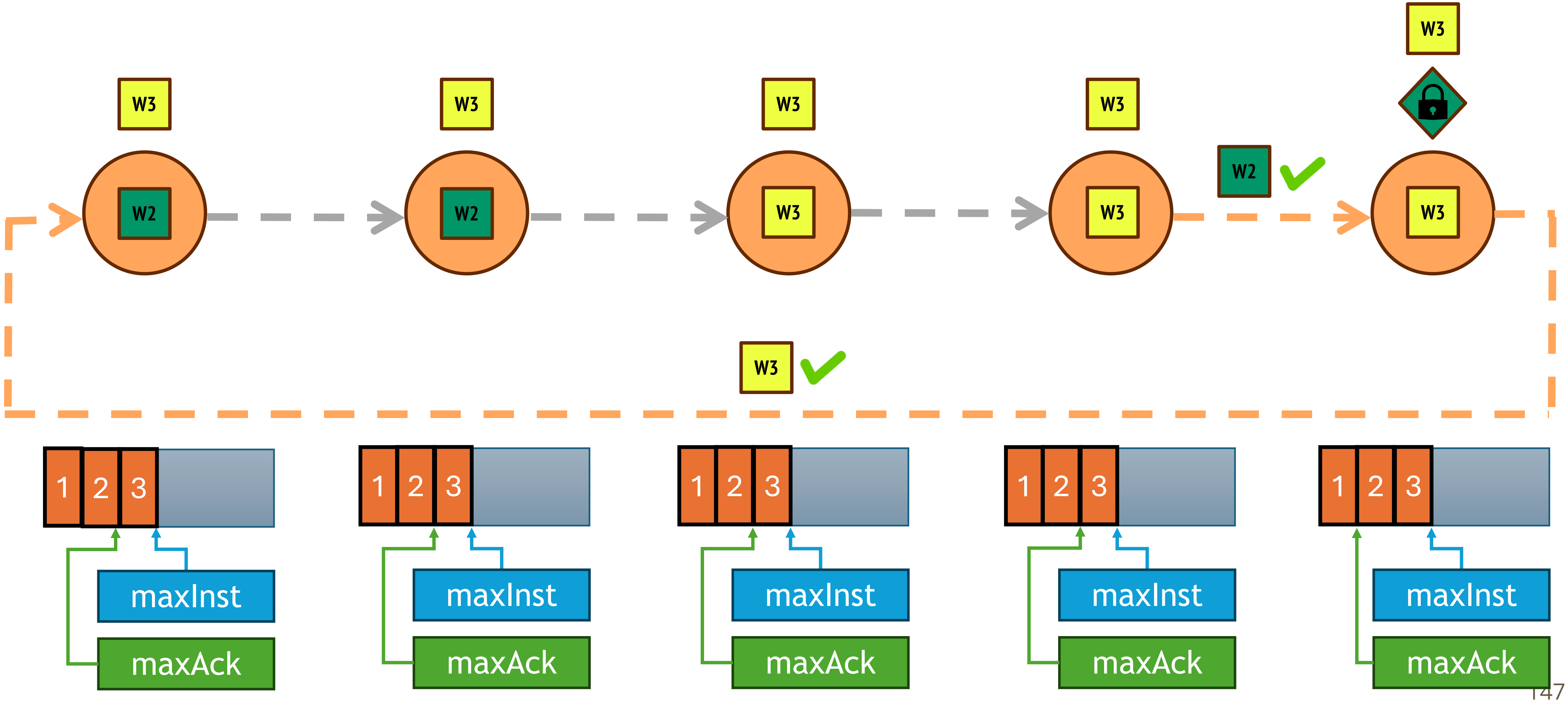
In Depth: Reads



In Depth: Reads

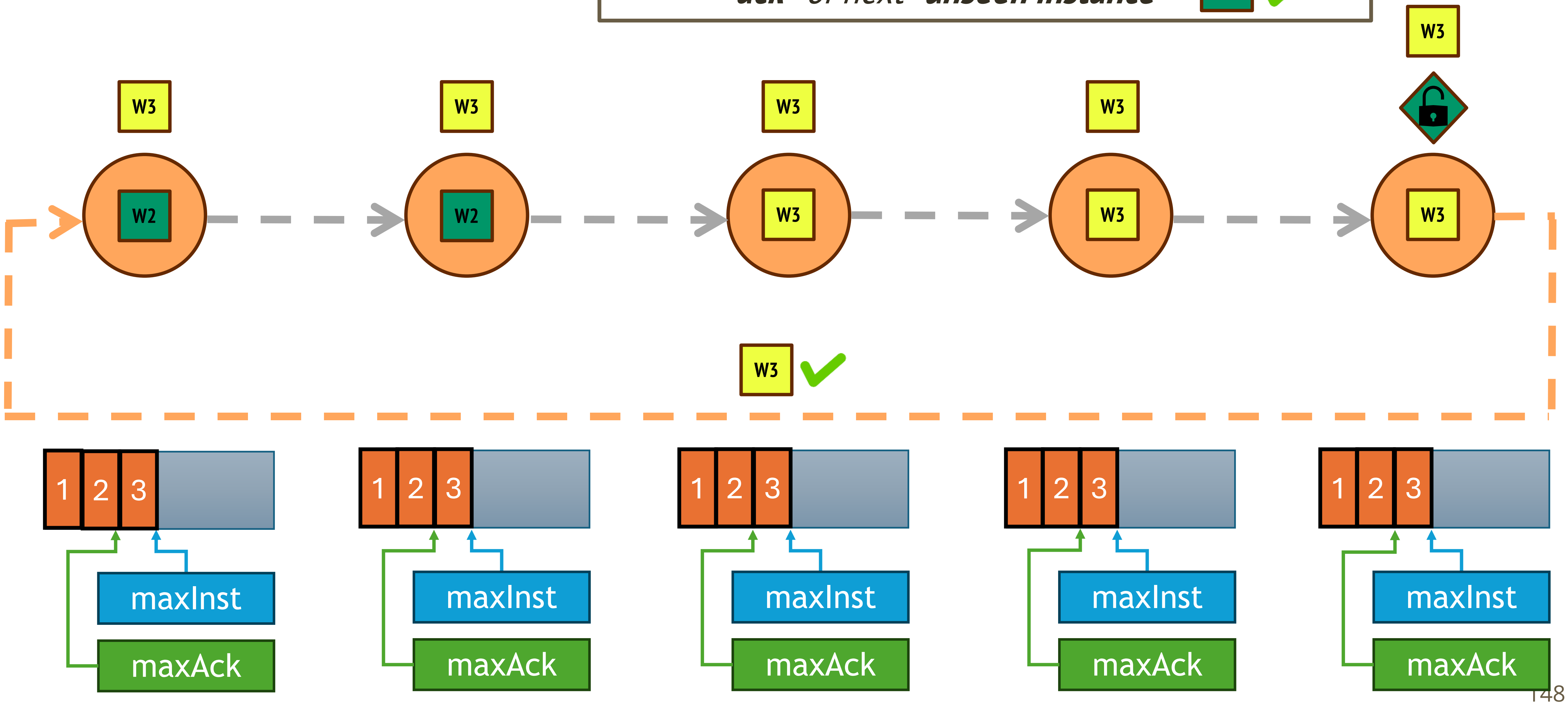


In Depth: Reads

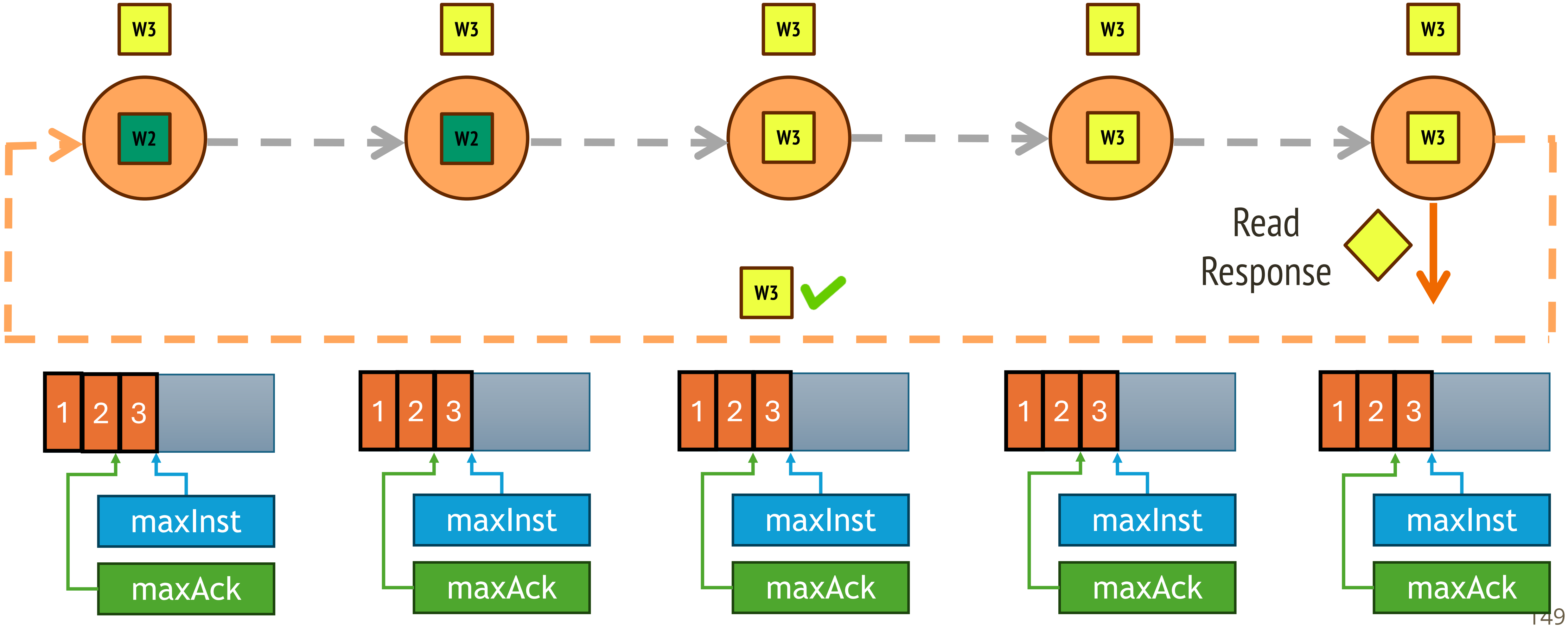


In Depth: Reads

"ack" of next "unseen instance" - W2 ✓



In Depth: Reads



In Depth: Reads

Reads wait for

“ack”** of next **“unseen instance”

In Depth: Reads

Why acks of next unseen instance?

In Depth: Reads

Why acks of next unseen instance?

Why not immediately reply?

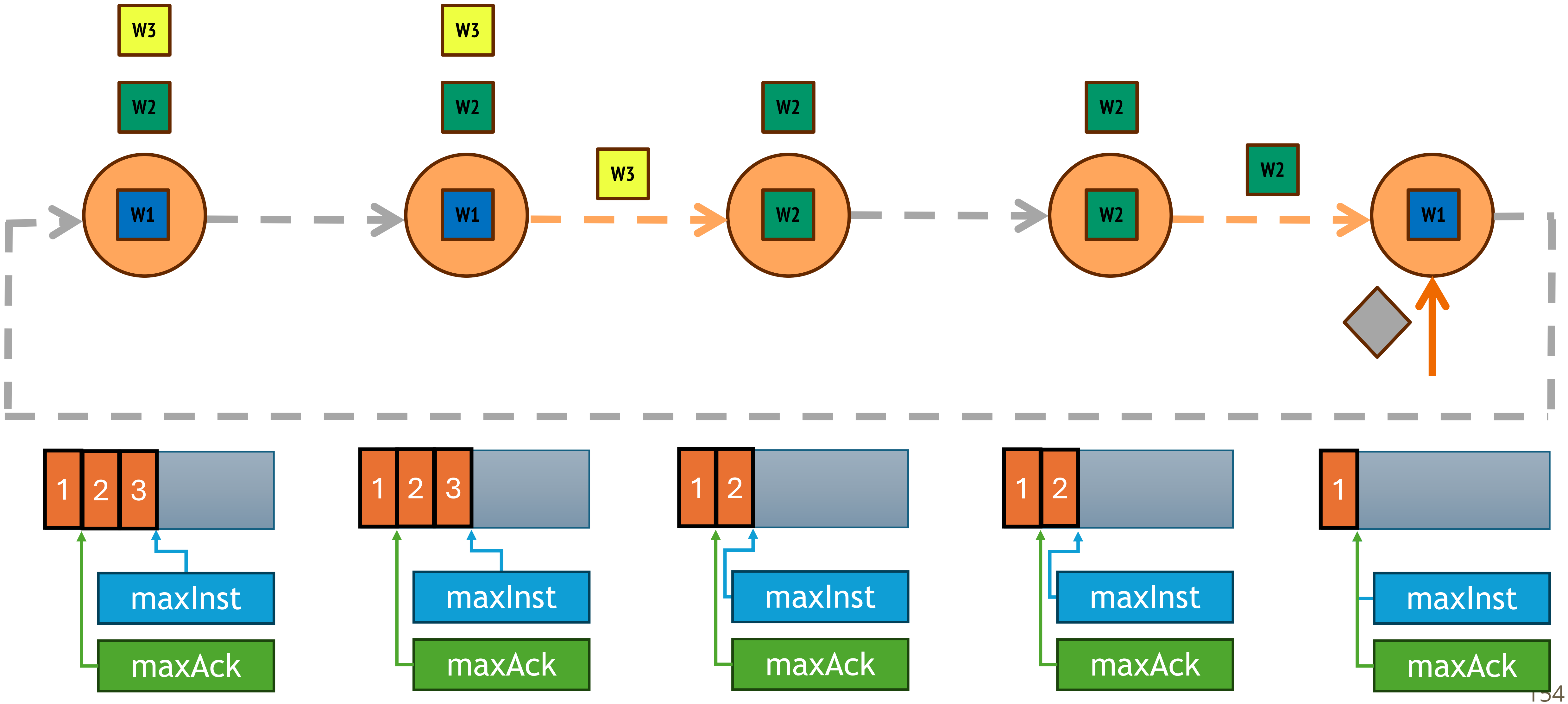
In Depth: Reads

Why acks of next unseen instance?

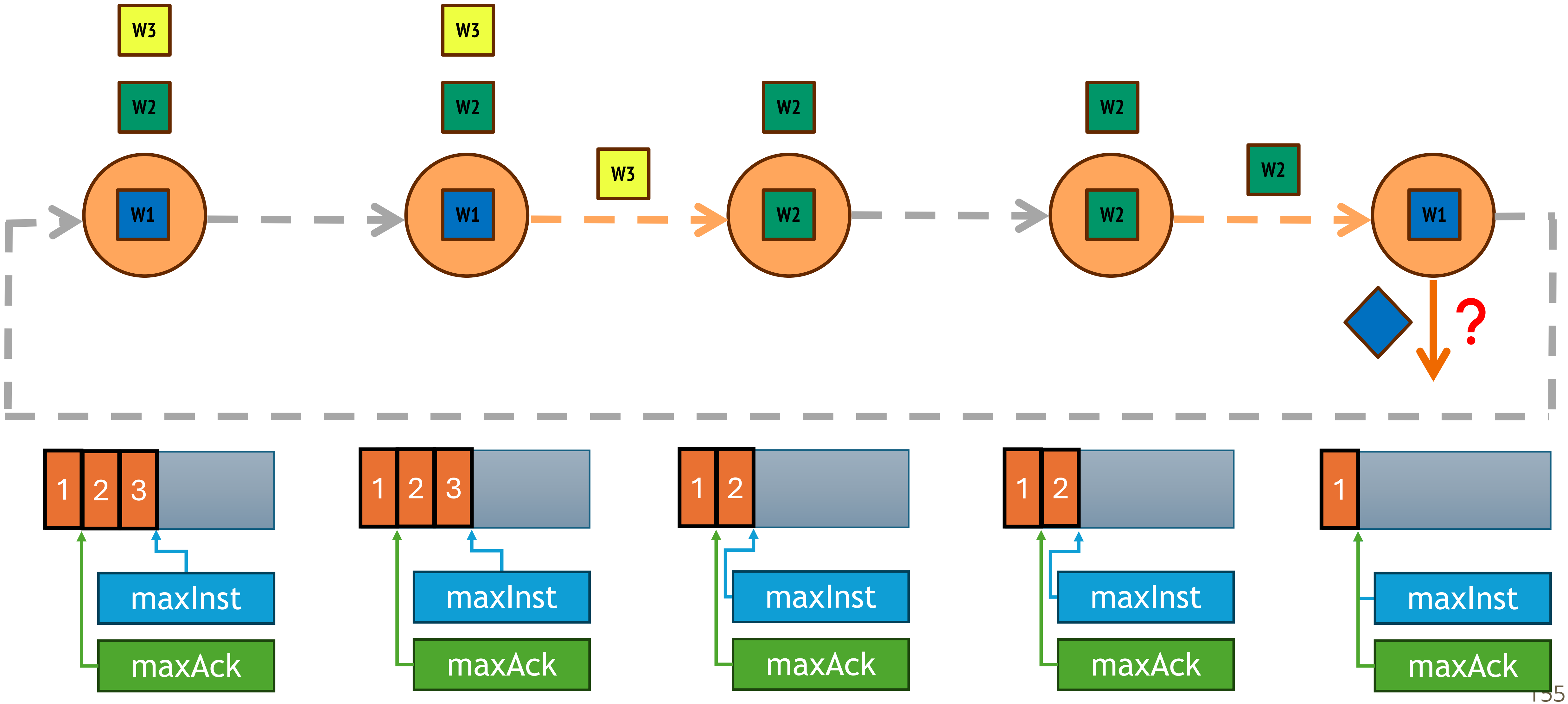
Why not immediately reply?

Replicas cannot be sure whether other replicas have replied to new writes

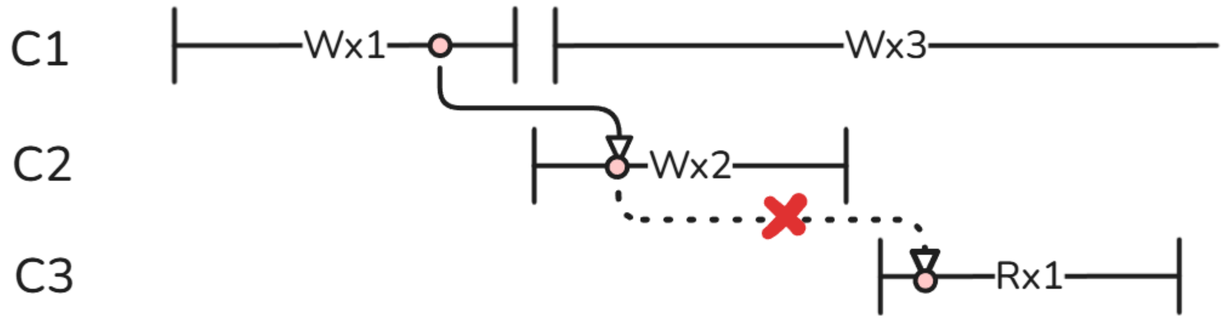
In Depth: Why not immediately reply?

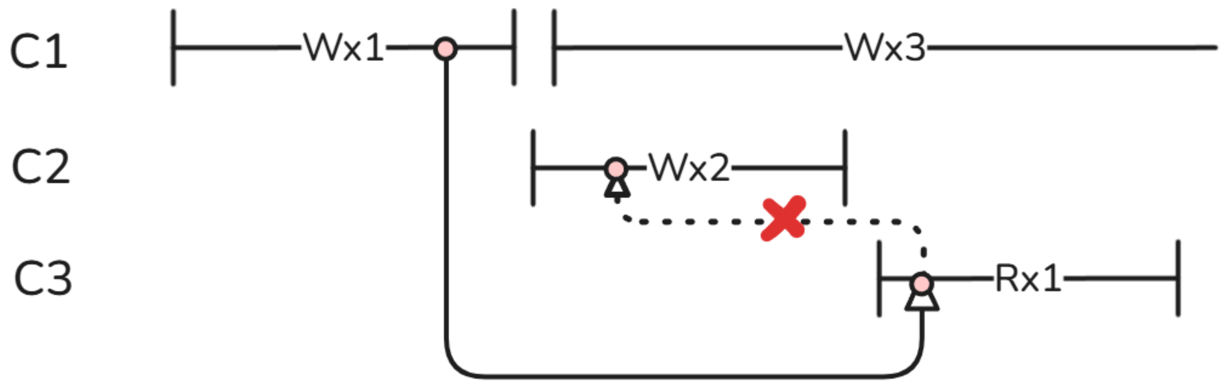


In Depth: Why not immediately reply?



In Depth: Why not immediately reply?





Not Linearizable ✗

In Depth: Reads

Why acks of next unseen instance?

~~*Why not immediately reply?*~~

In Depth: Reads

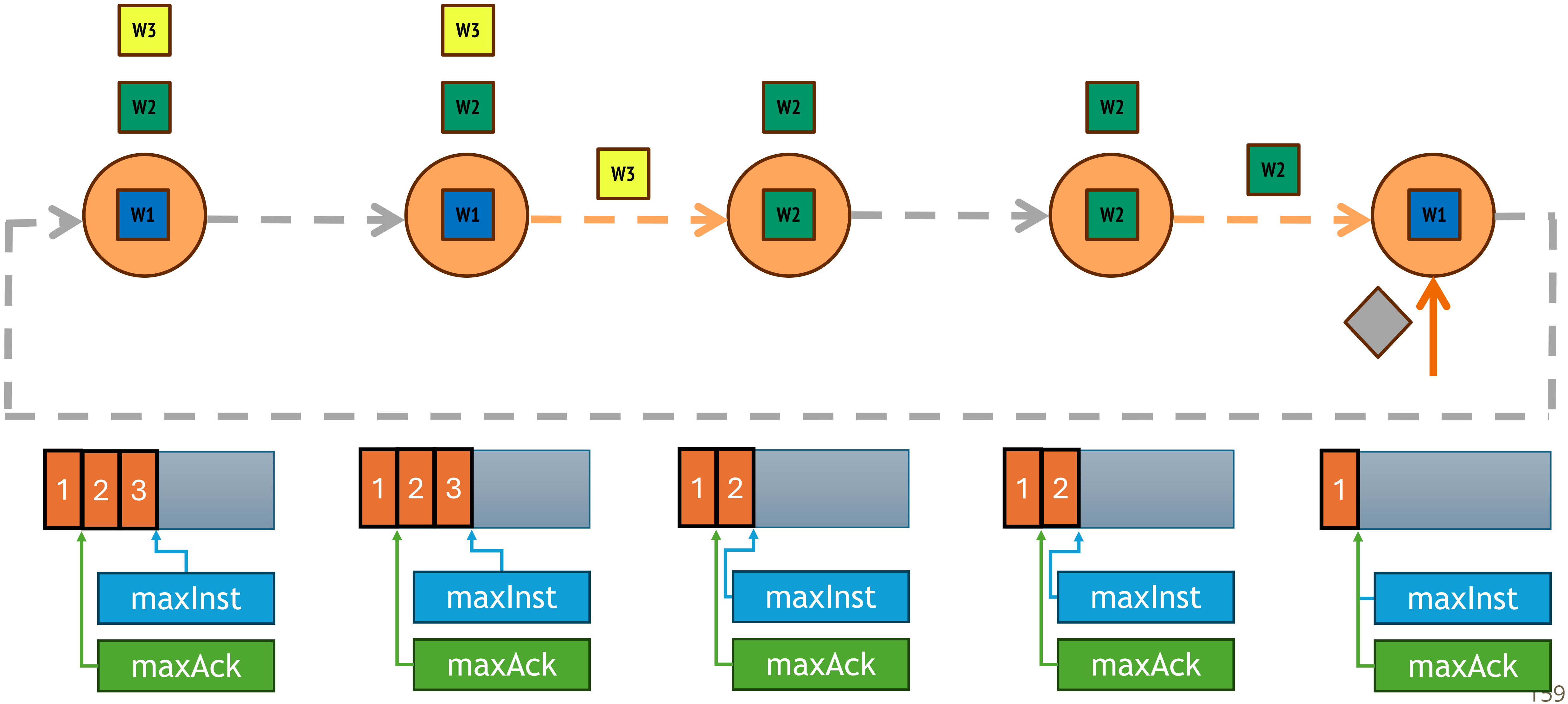
Why acks of next unseen instance?

~~*Why not immediately reply?*~~

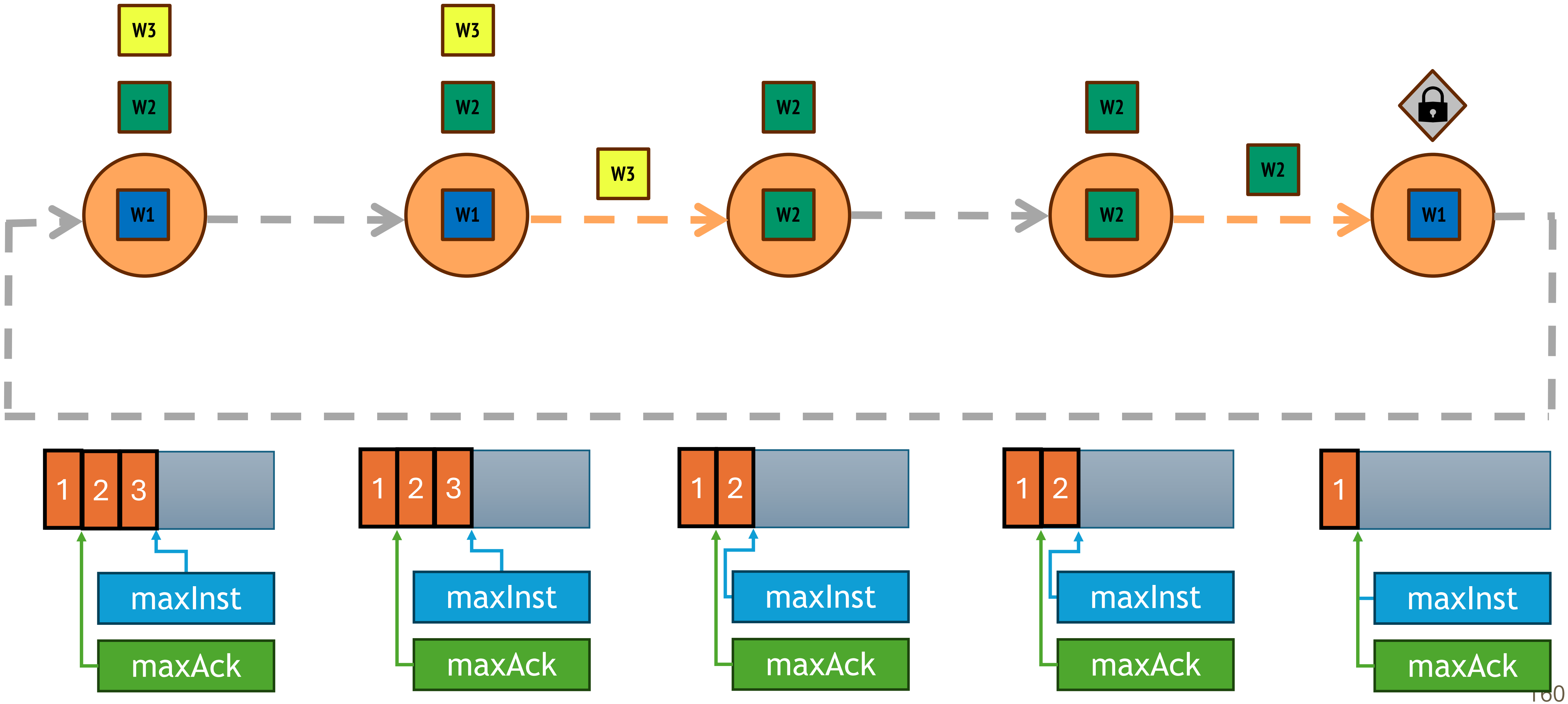
Why not next unseen instance?

Again, replicas cannot be sure whether other replicas have replied to new writes

In Depth: Why not next unseen instance?

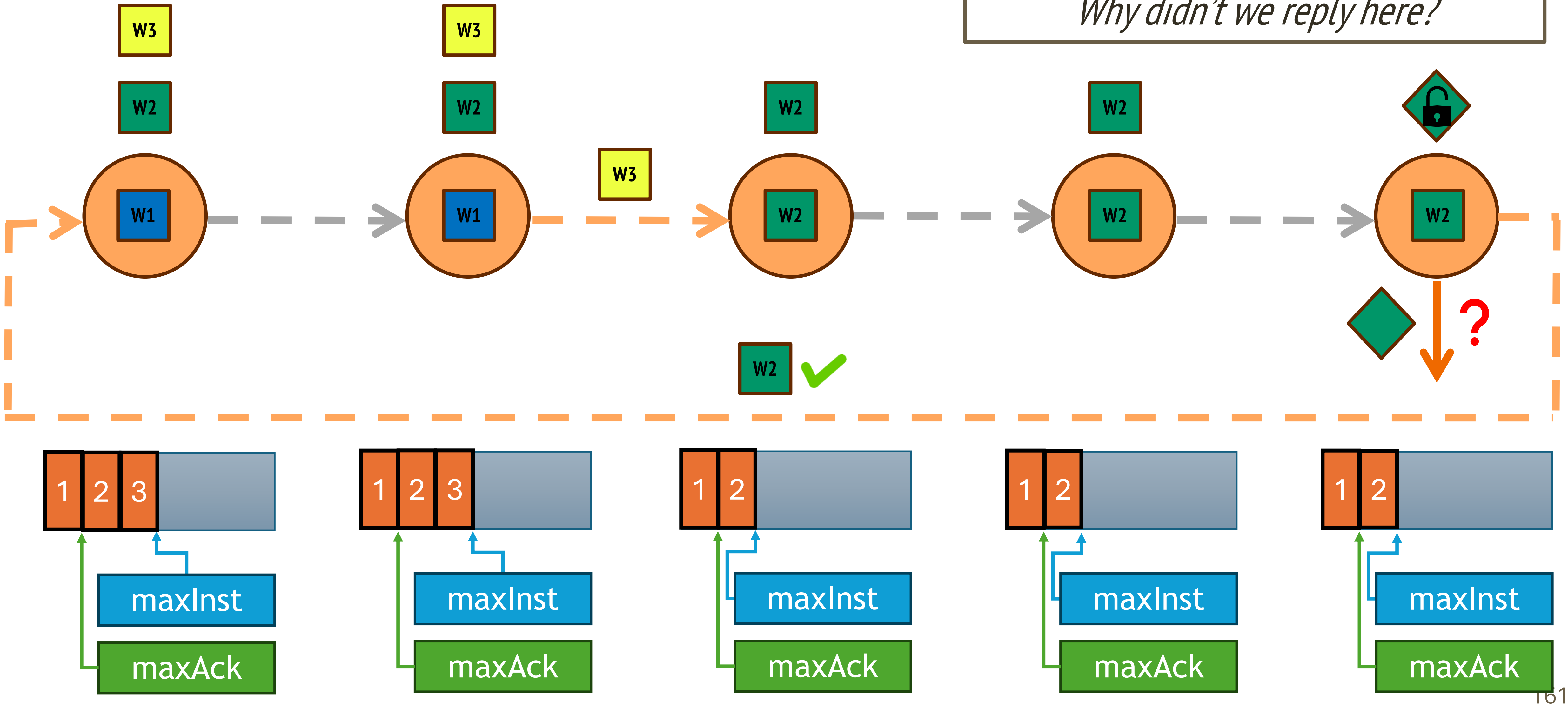


In Depth: Why not next unseen instance?



In Depth: Why not next unseen instance?

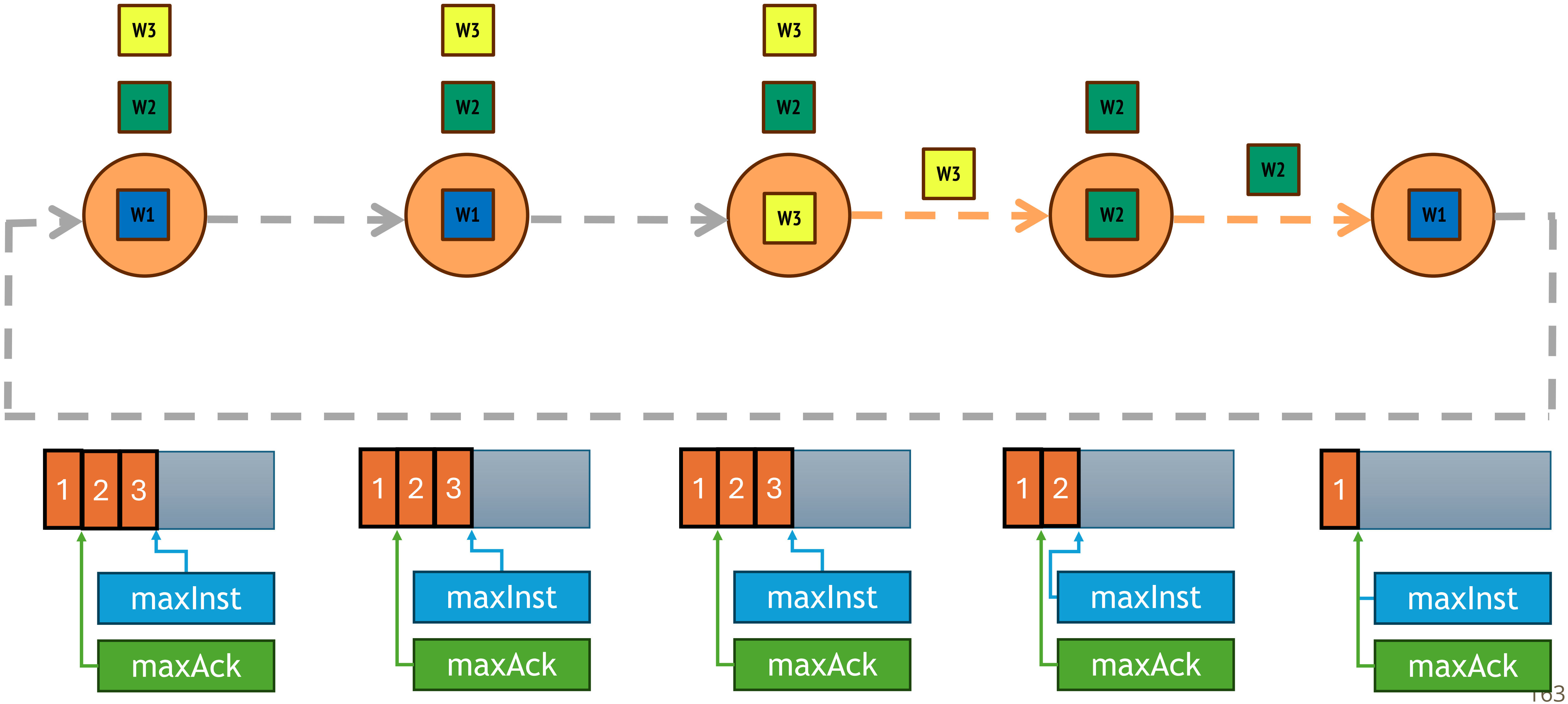
Why didn't we reply here?



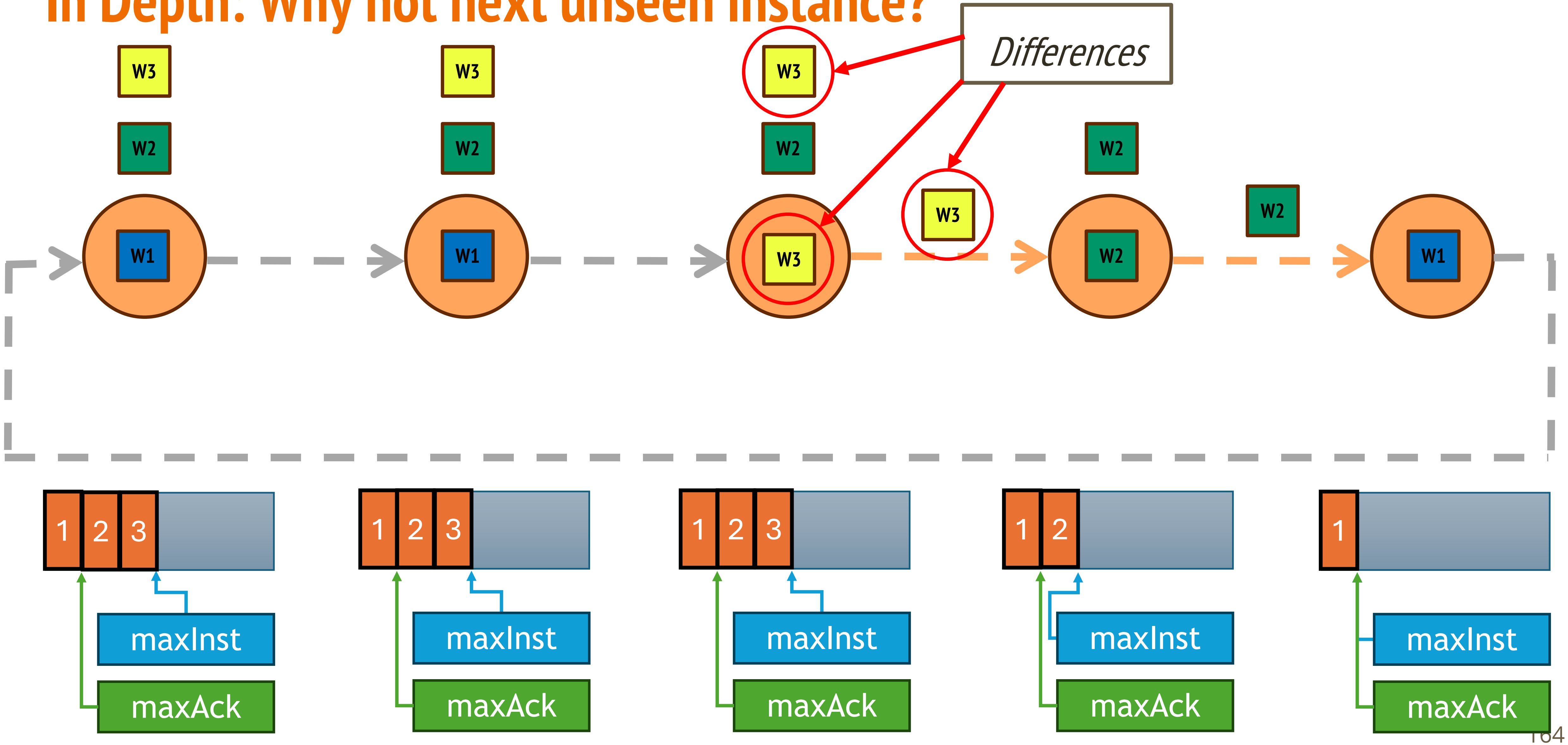
In Depth: Why not next unseen instance?

Consider a slightly different situation of the system …

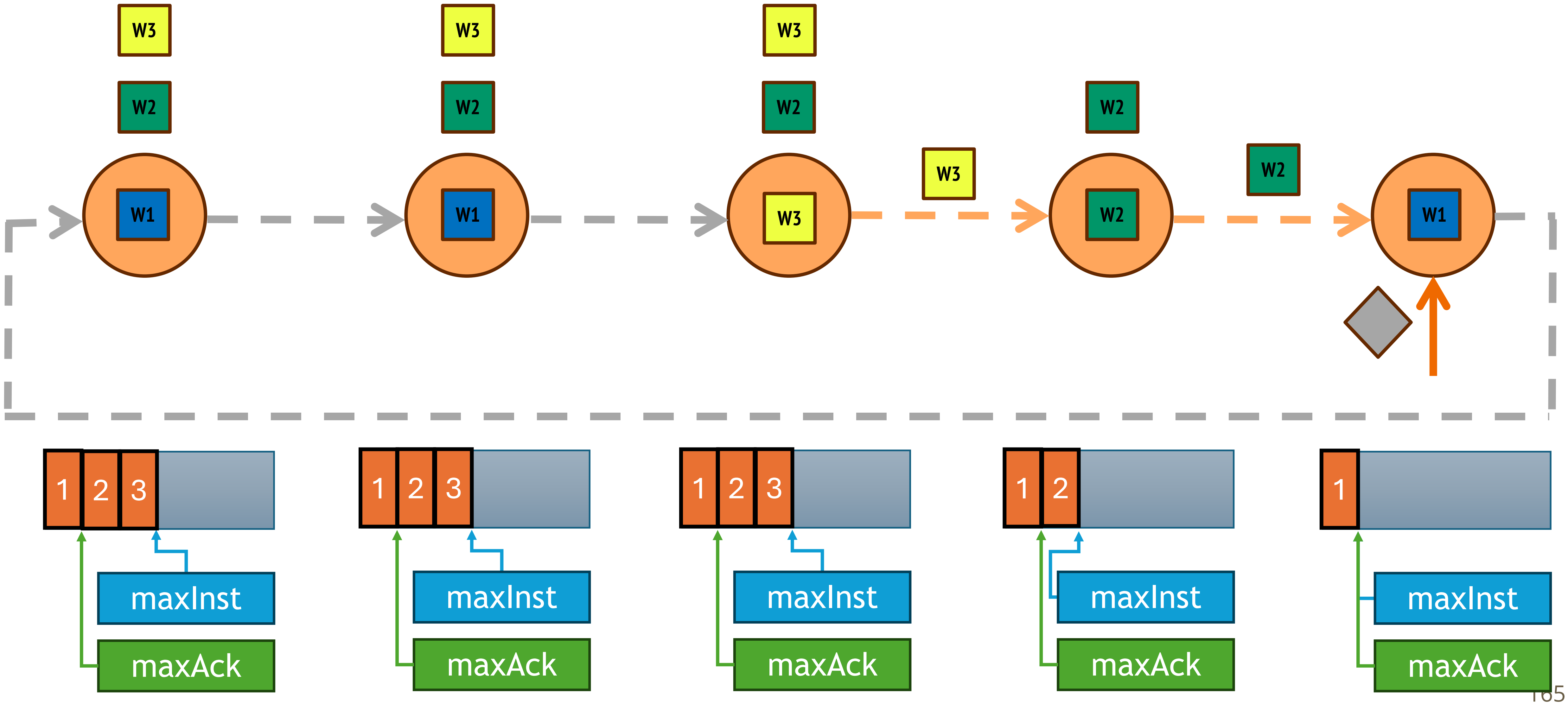
In Depth: Why not next unseen instance?



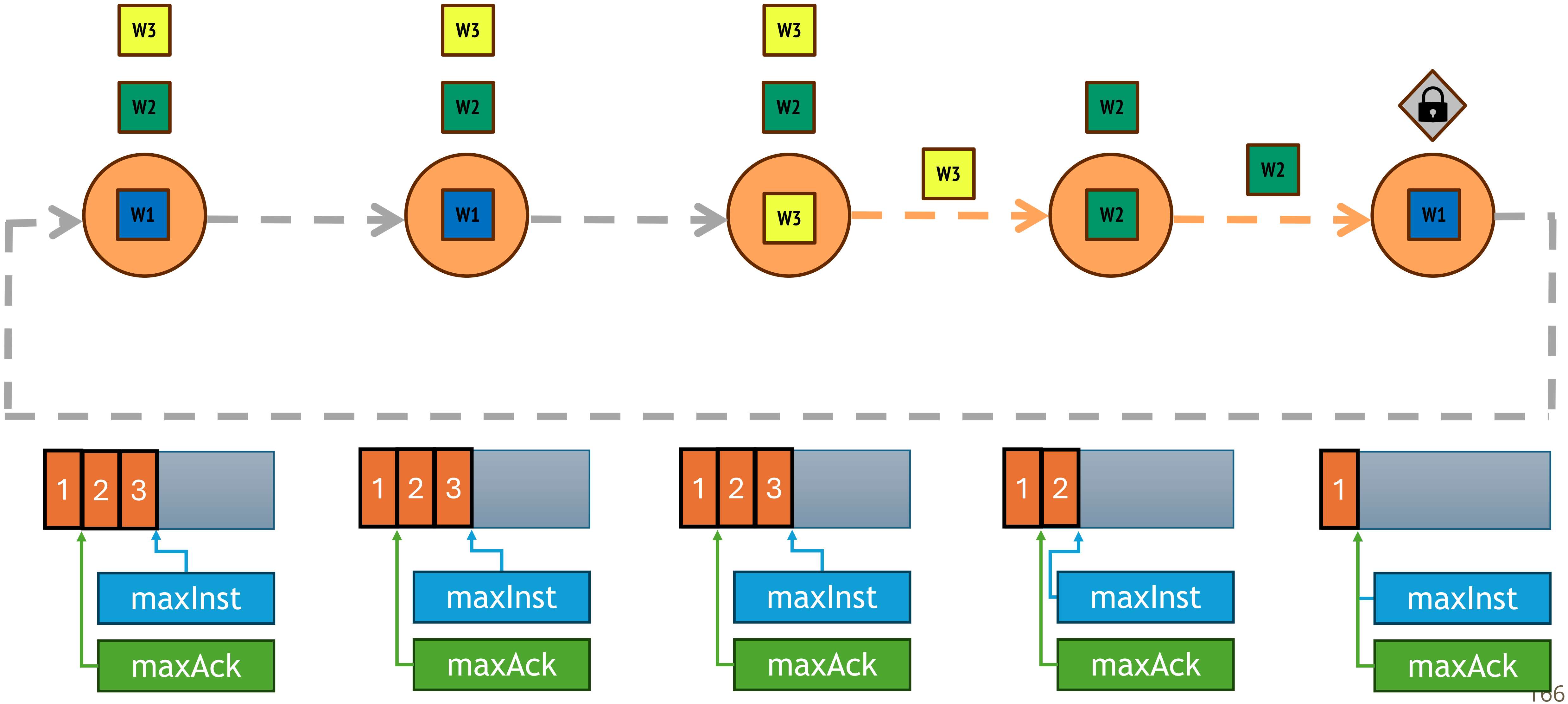
In Depth: Why not next unseen instance?



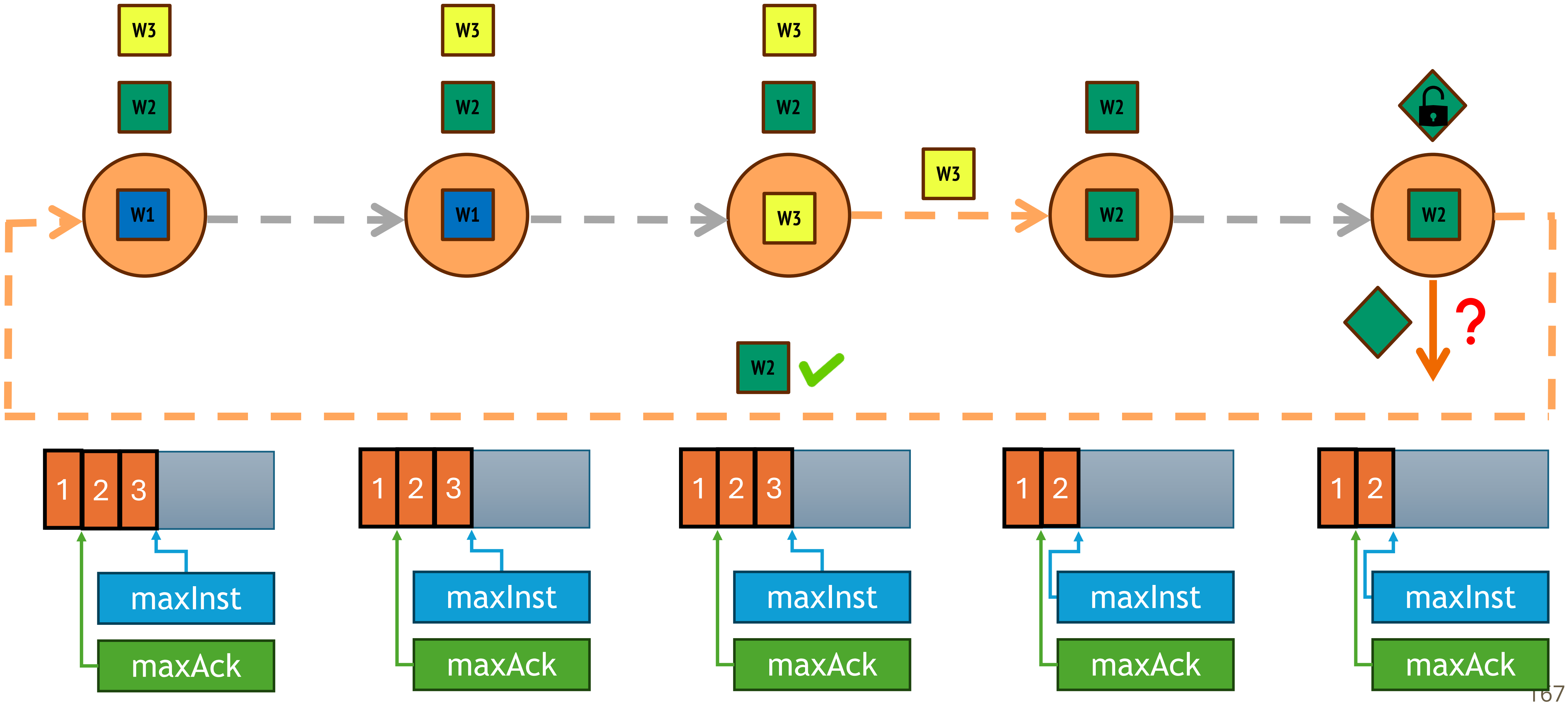
In Depth: Why not next unseen instance?



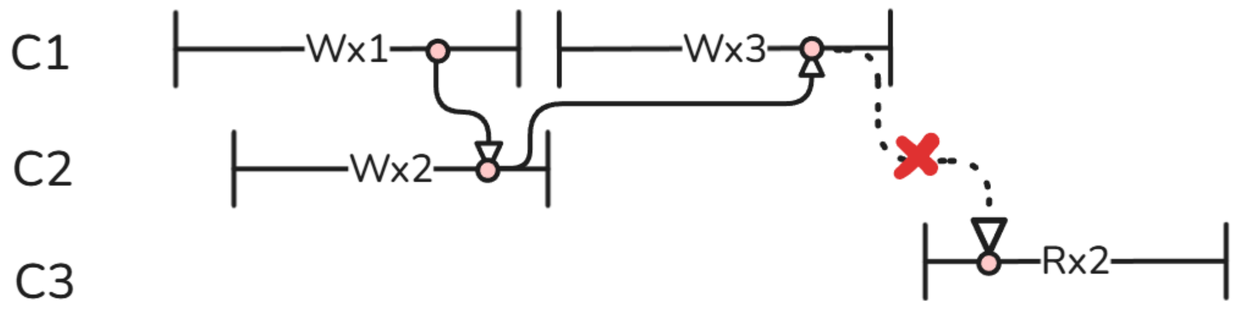
In Depth: Why not next unseen instance?

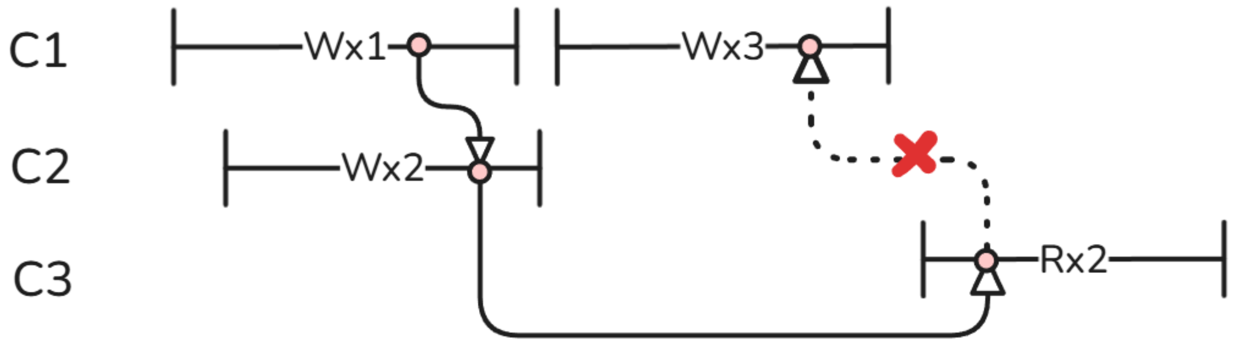


In Depth: Why not next unseen instance?



In Depth: Why not next unseen instance?





Not Linearizable ✗

In Depth: Reads

Why acks of next unseen instance?

~~*Why not immediately reply?*~~

~~*Why not next unseen instance?*~~

In Depth: Reads

Why acks of next unseen instance?

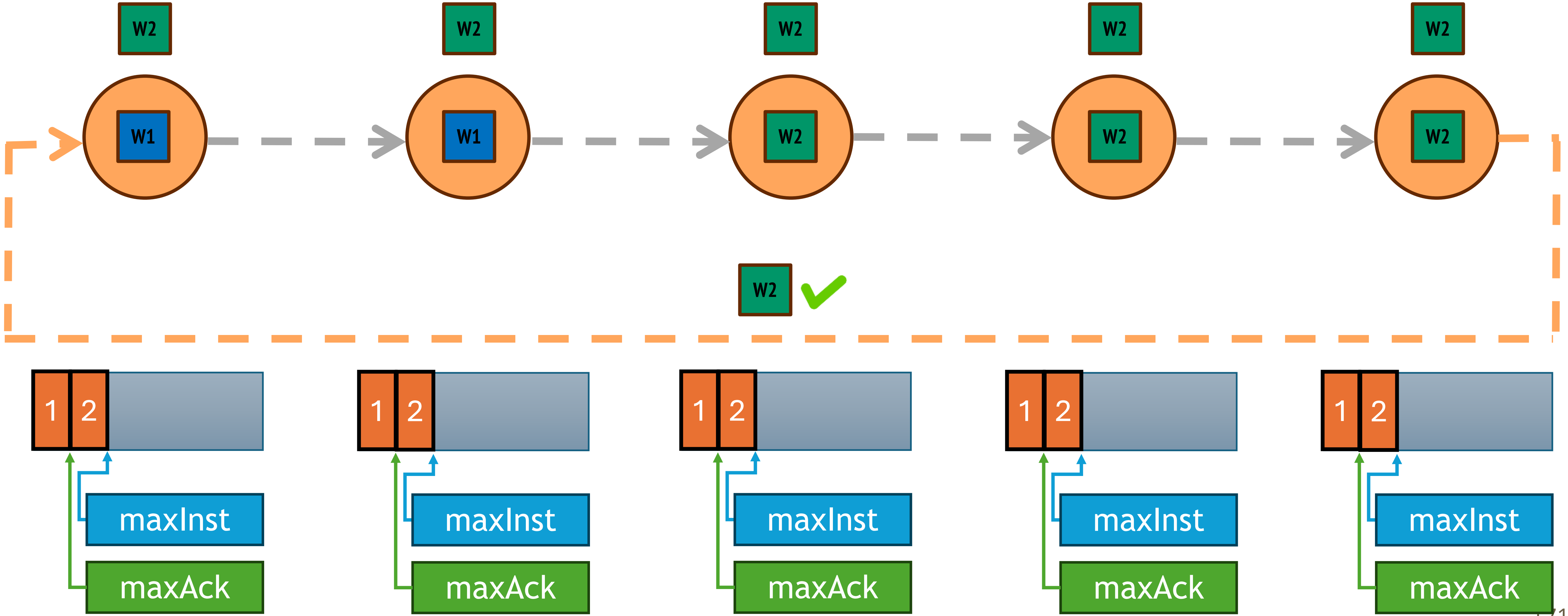
~~*Why not immediately reply?*~~

~~*Why not next unseen instance?*~~

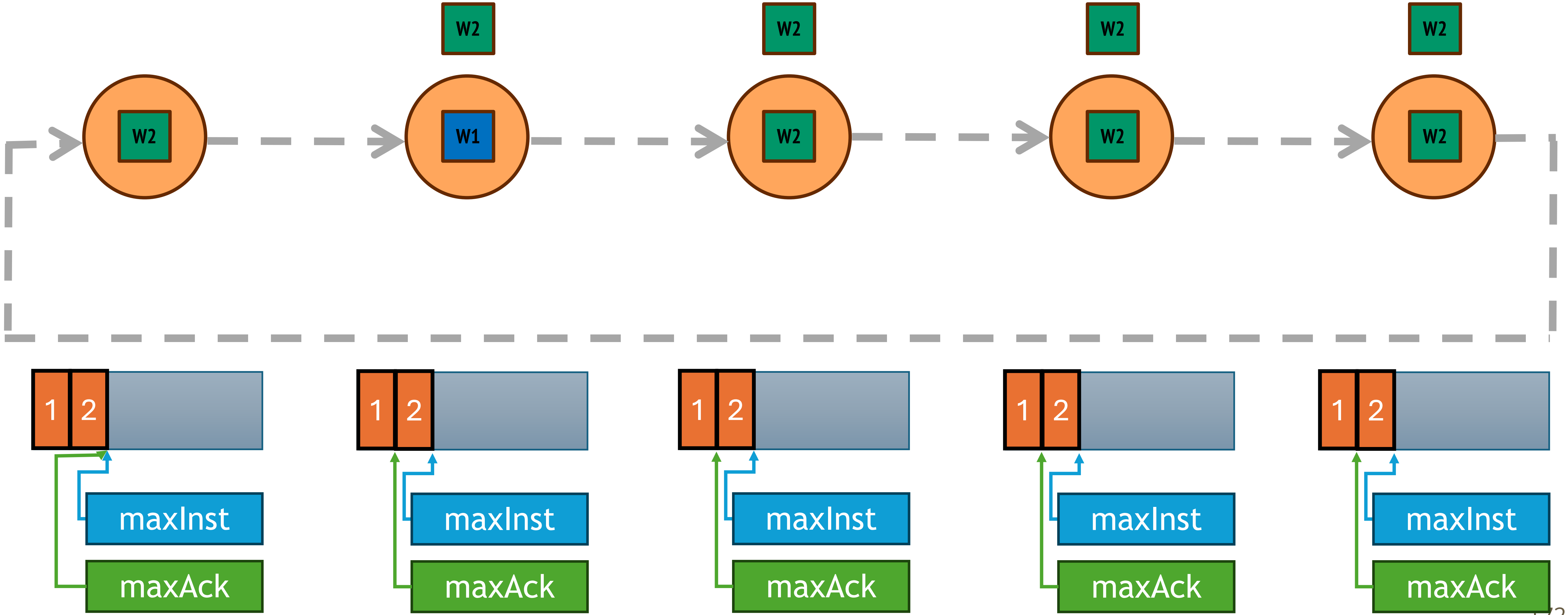
Why not ack of max seen instance?

Slightly more complicated to show, but it still leads to non-linearizable history

In Depth: Why not ack of max seen instance?

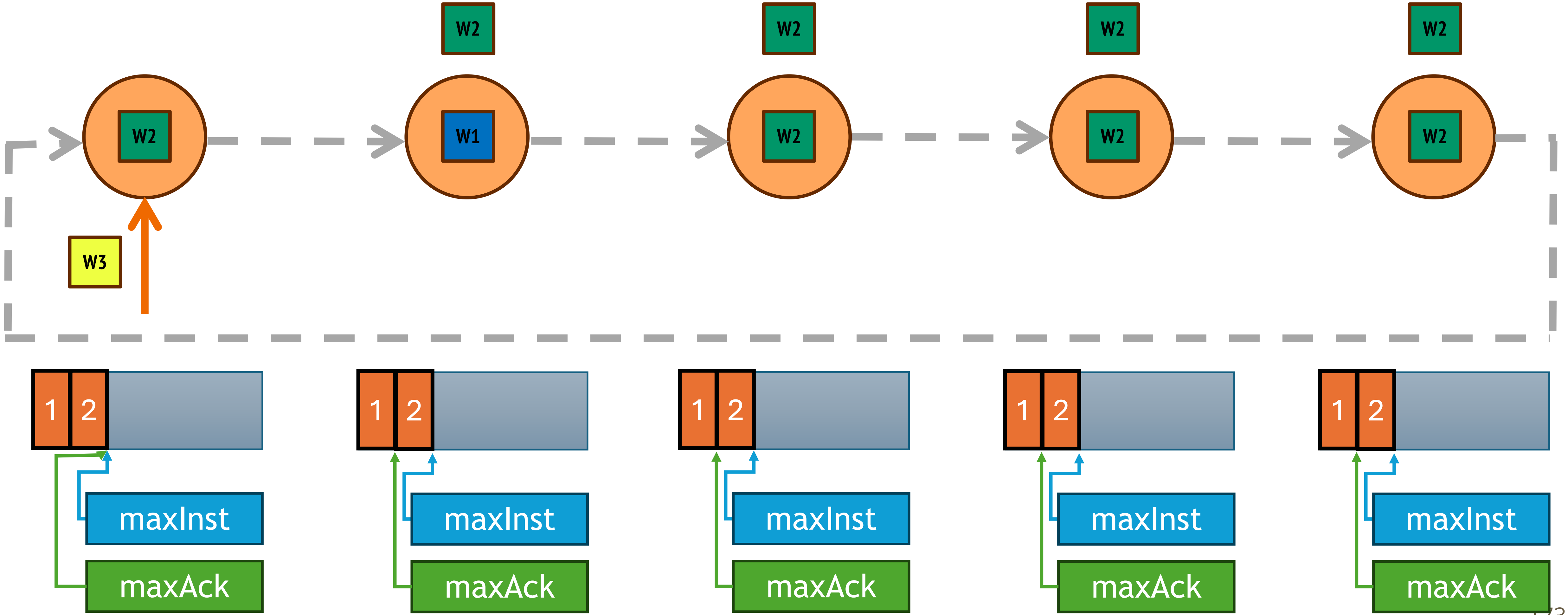


In Depth: Why not ack of max seen instance?



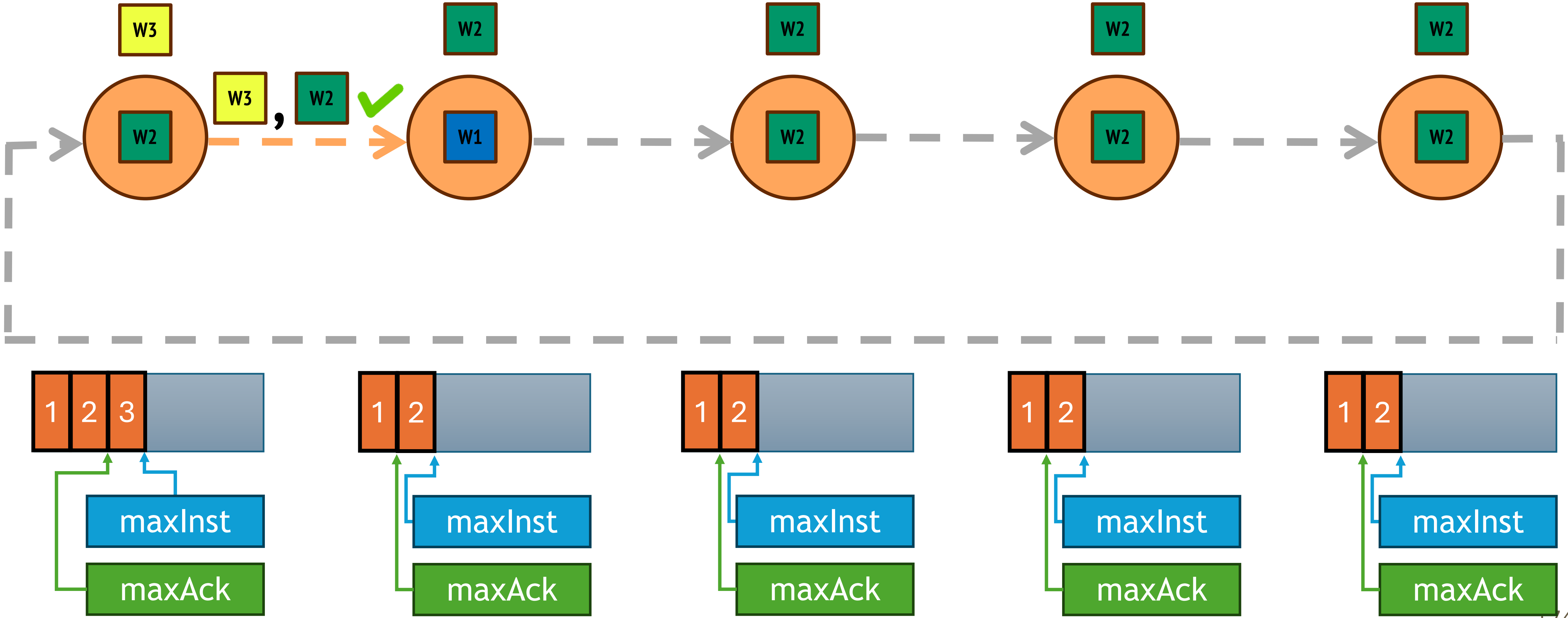
In Depth: Why not ack of max seen instance?

Ack is "piggybacked" along with next write

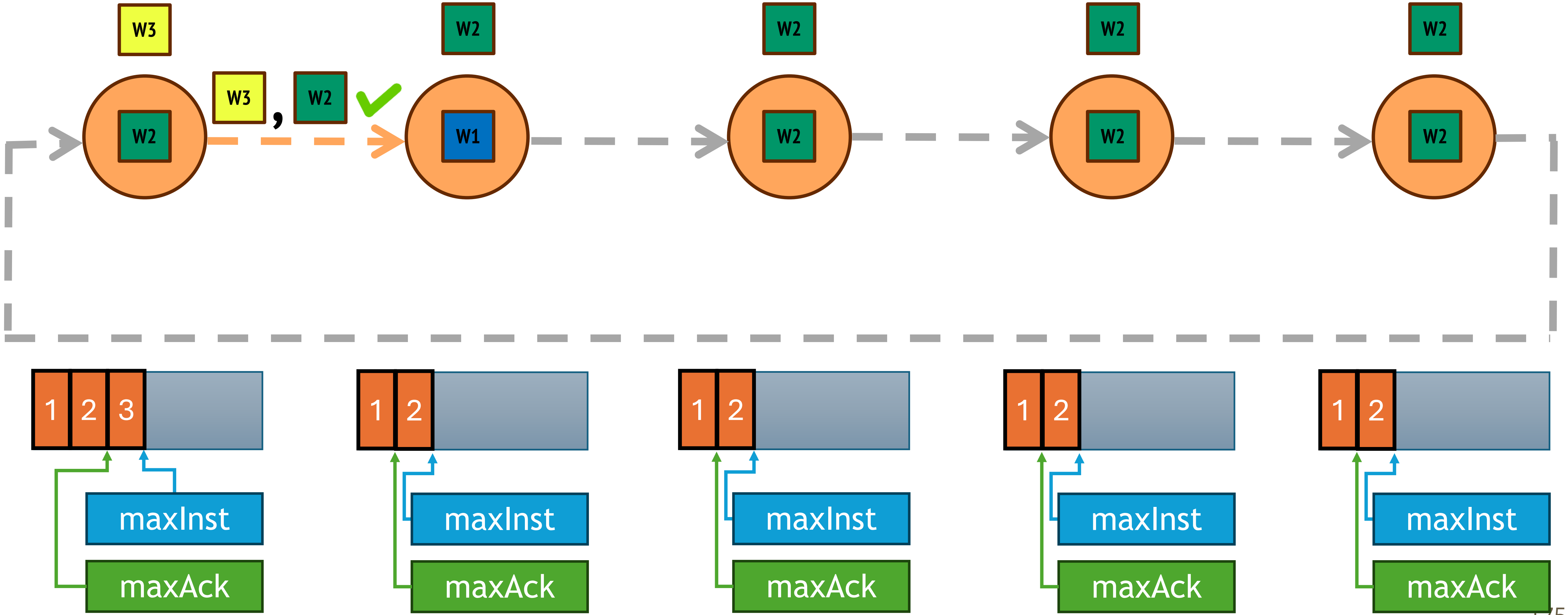


In Depth: Why not ack of max seen instance?

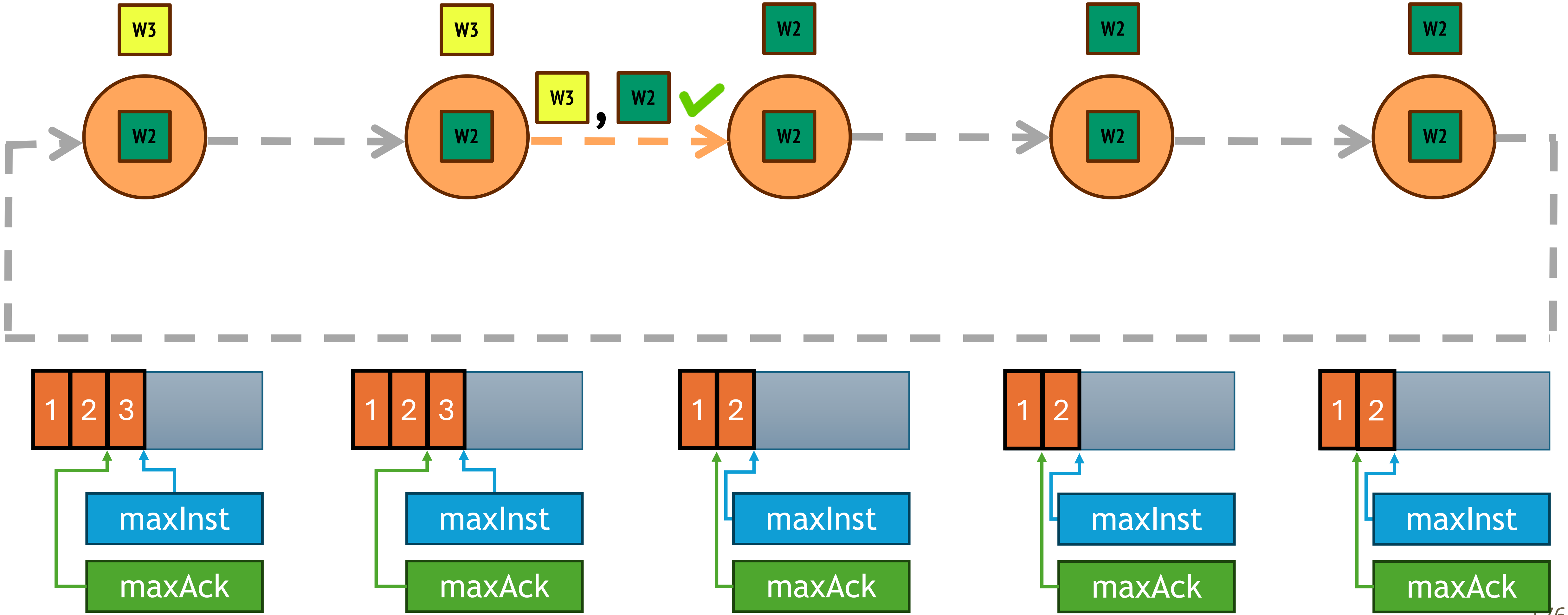
Ack is "piggybacked" along with next write



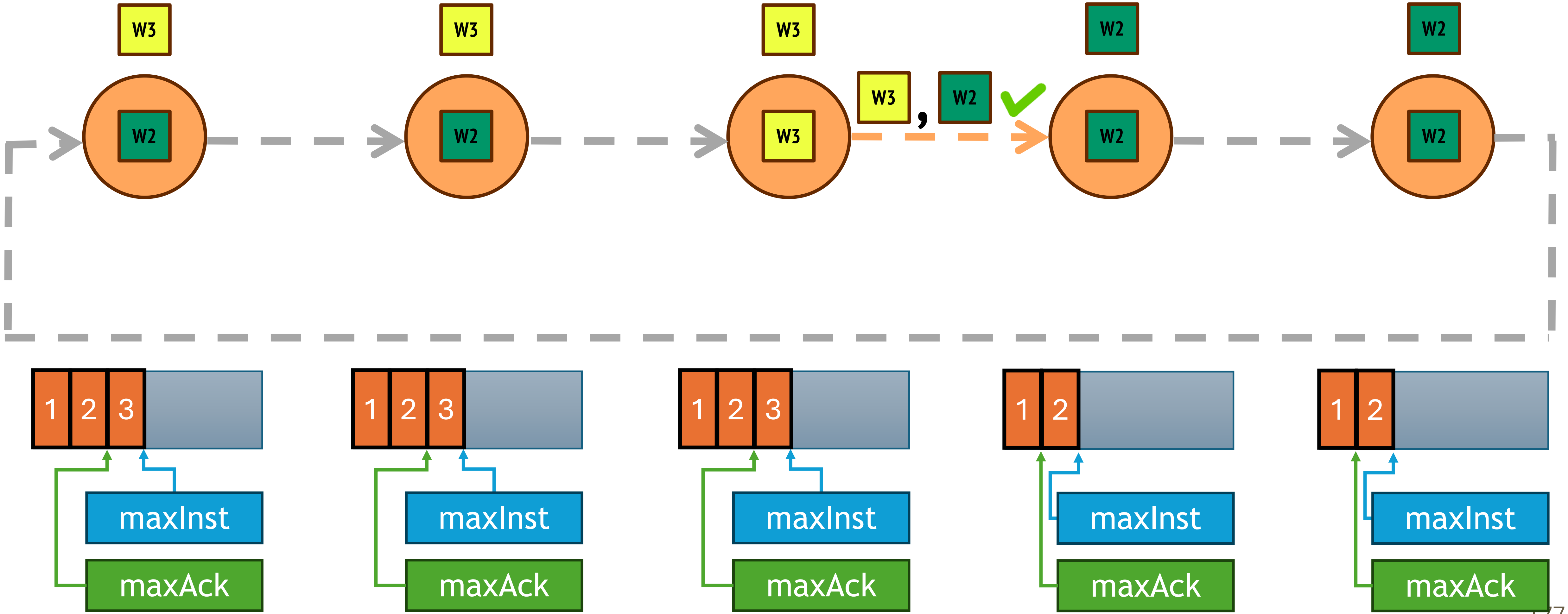
In Depth: Why not ack of max seen instance?



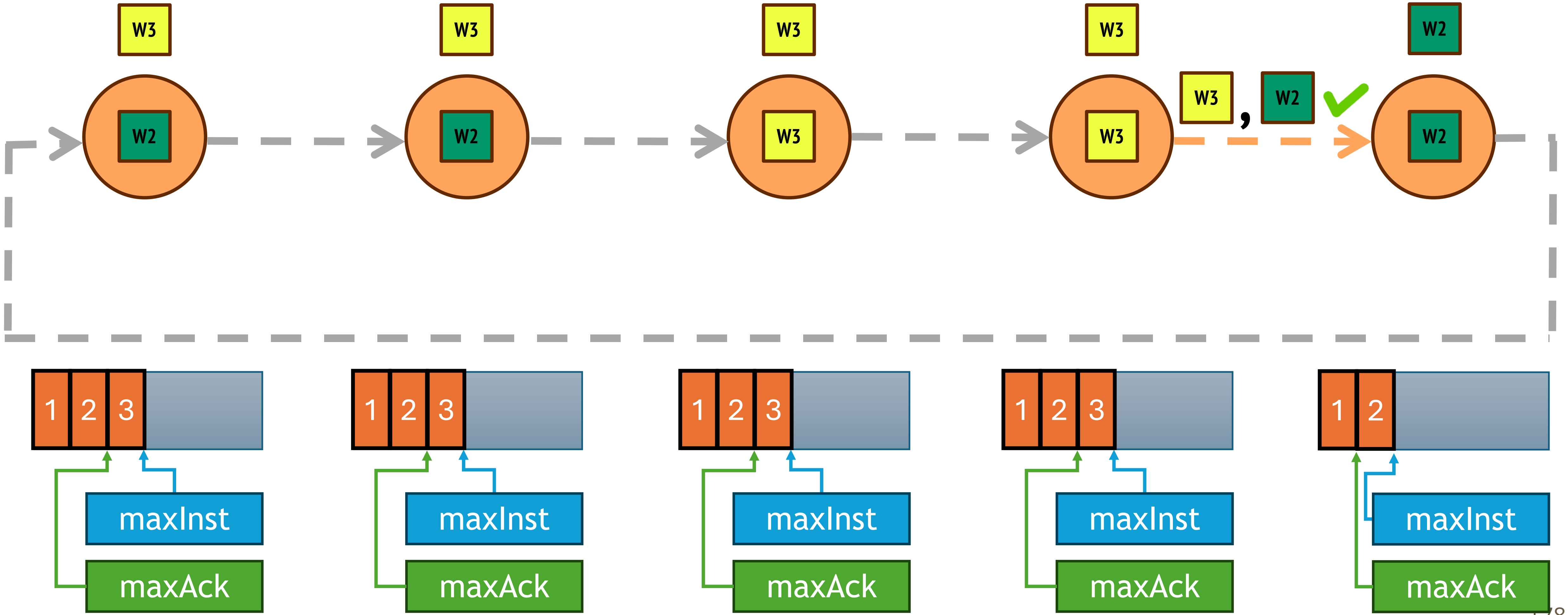
In Depth: Why not ack of max seen instance?



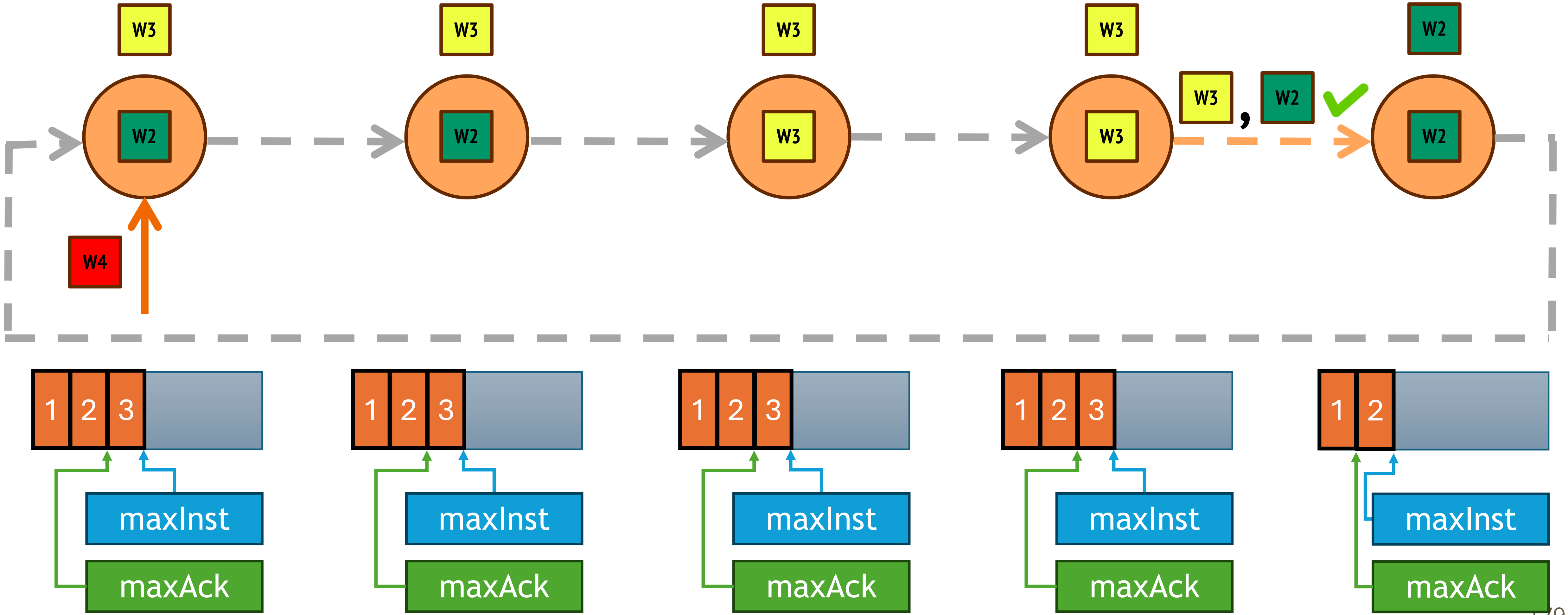
In Depth: Why not ack of max seen instance?



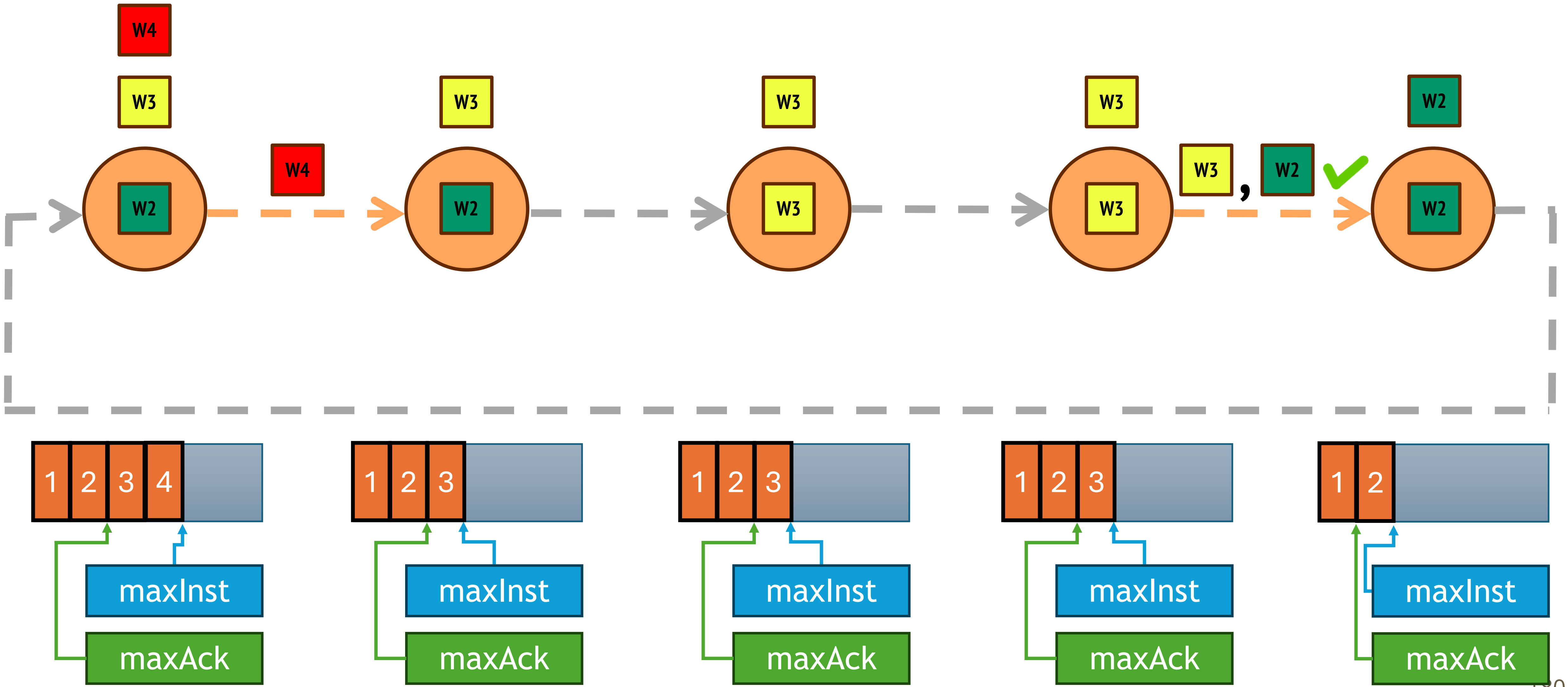
In Depth: Why not ack of max seen instance?



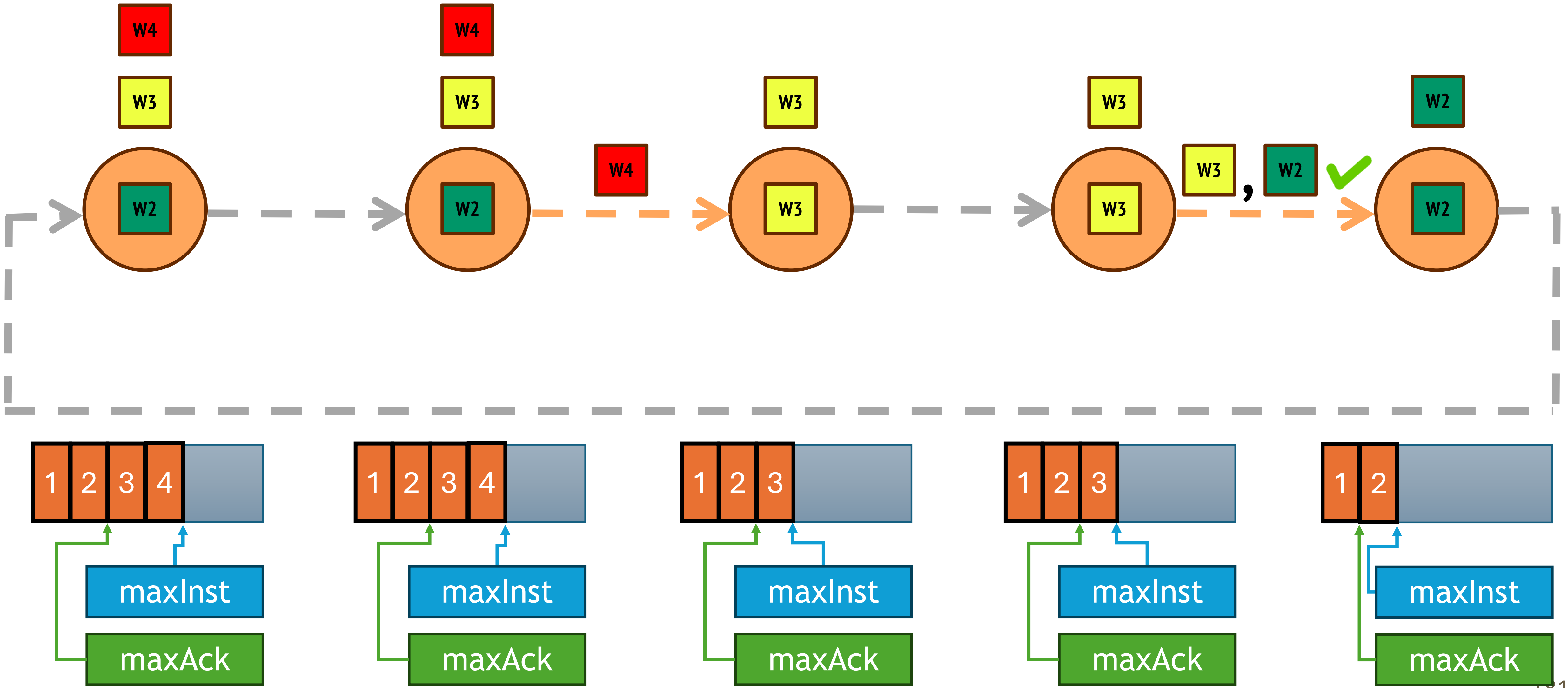
In Depth: Why not ack of max seen instance?



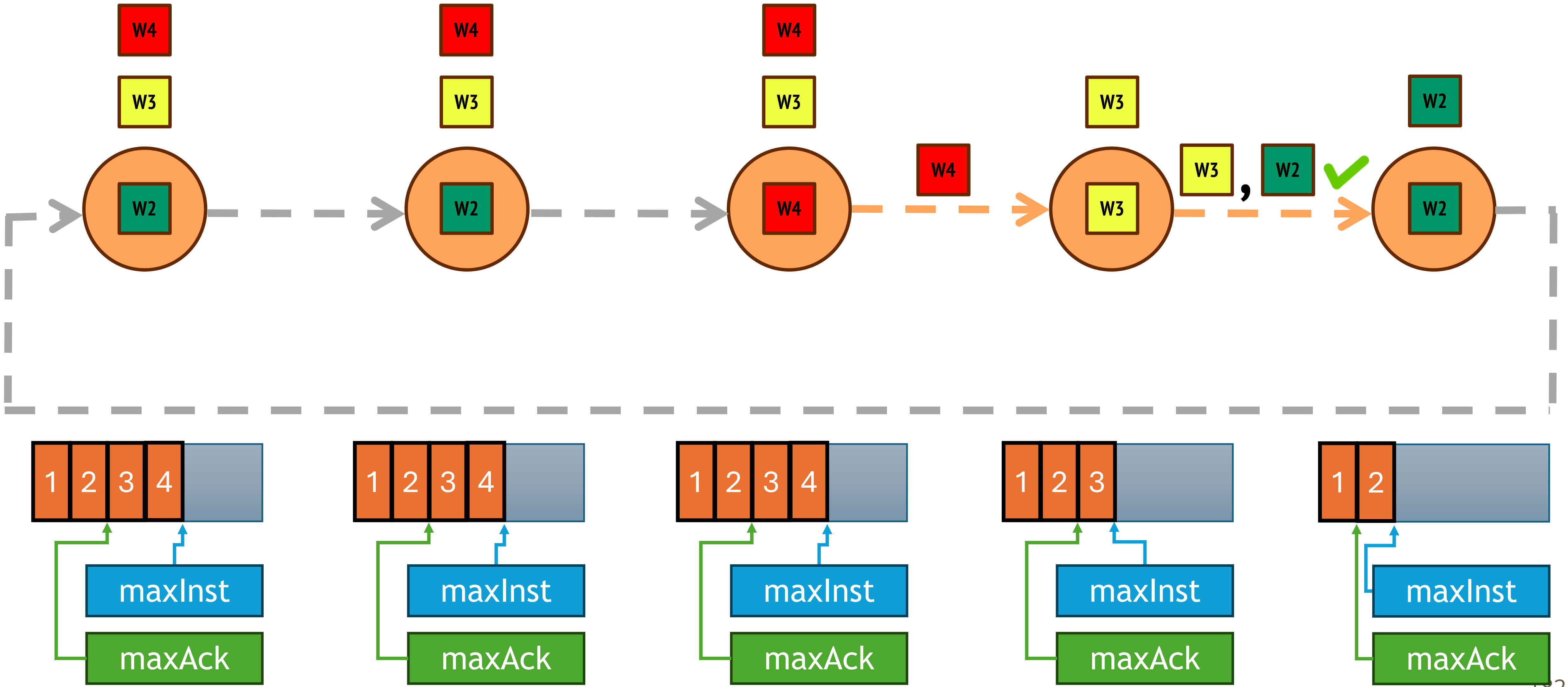
In Depth: Why not ack of max seen instance?



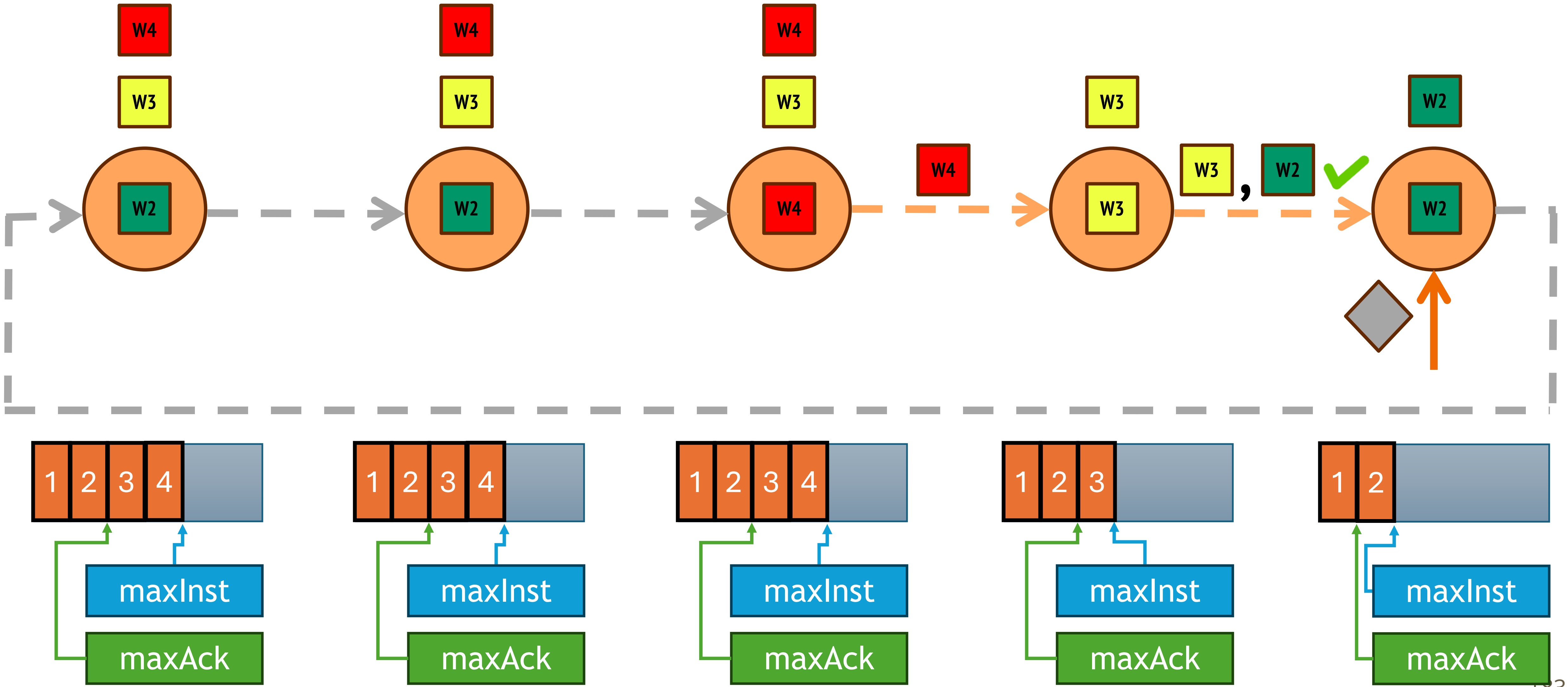
In Depth: Why not ack of max seen instance?



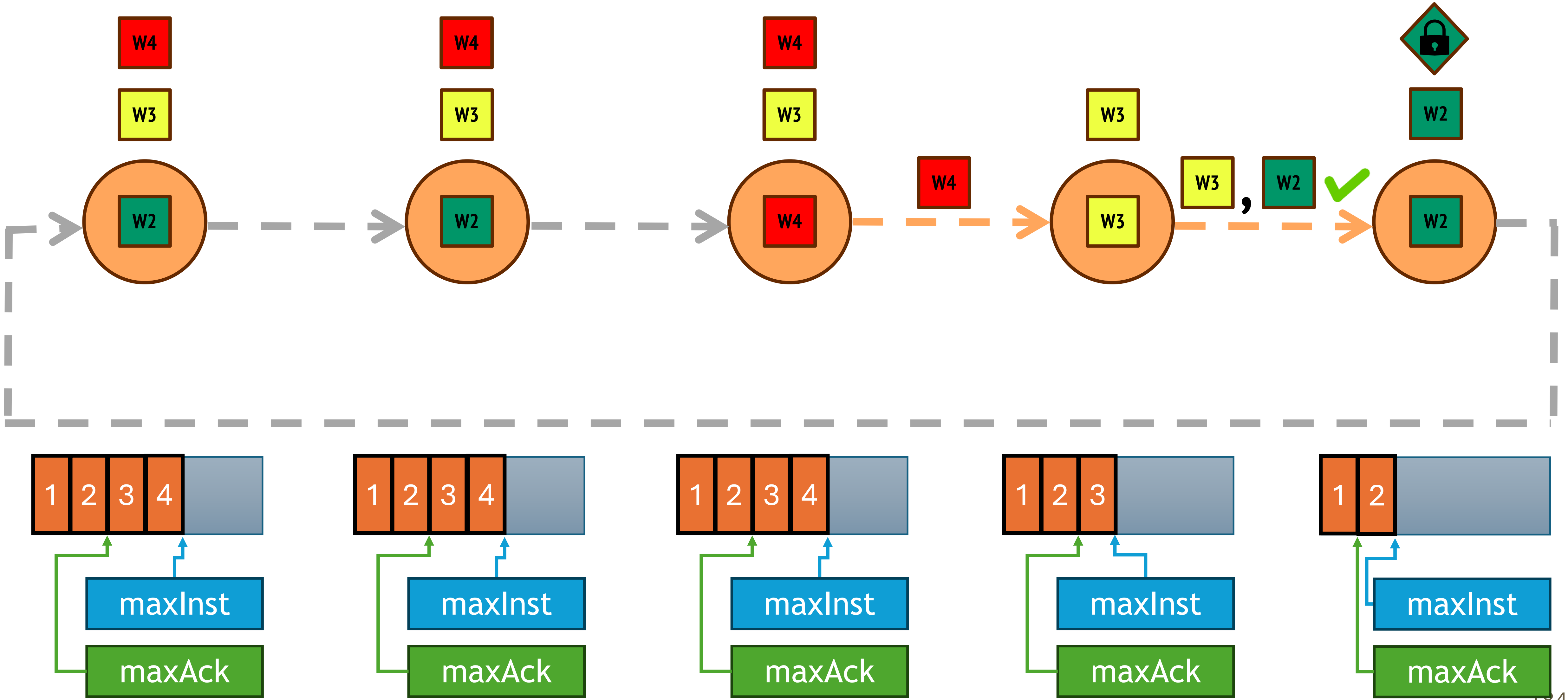
In Depth: Why not ack of max seen instance?



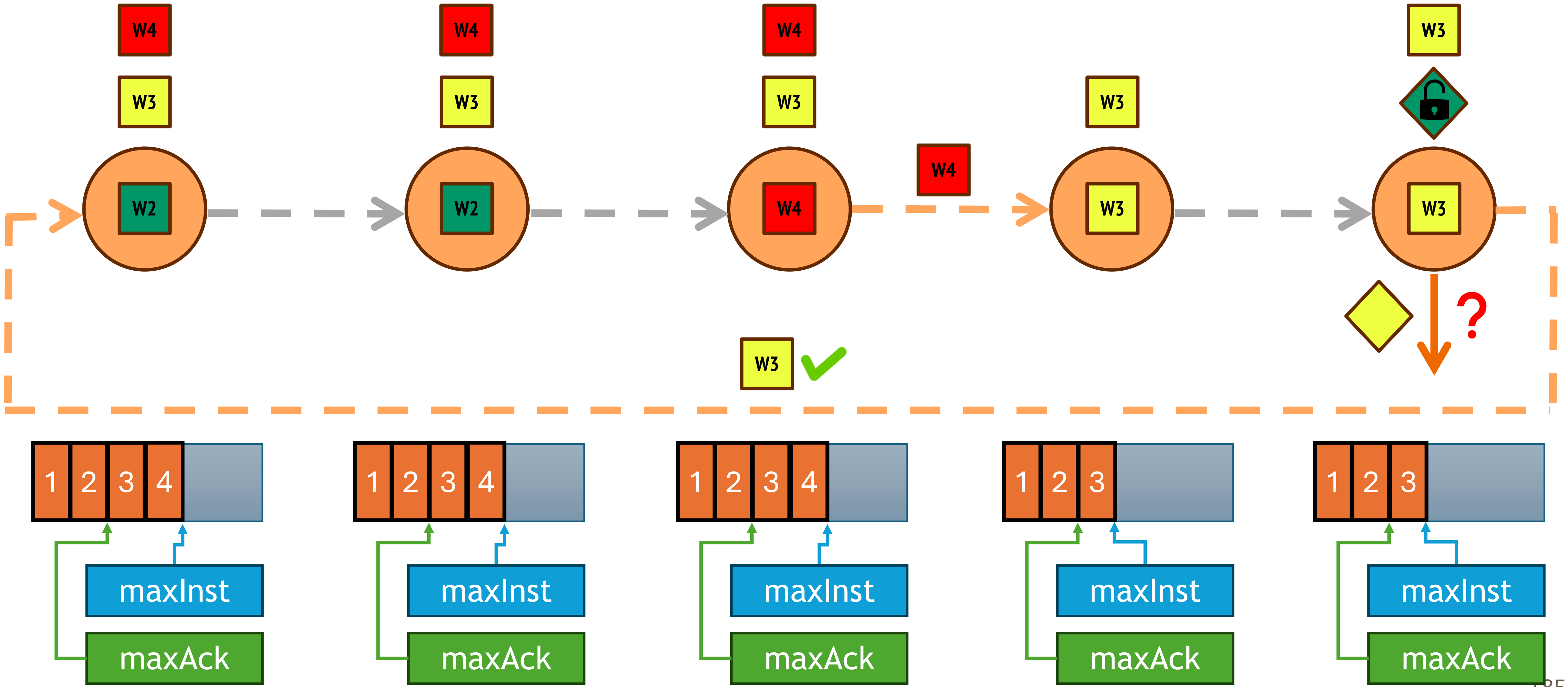
In Depth: Why not ack of max seen instance?



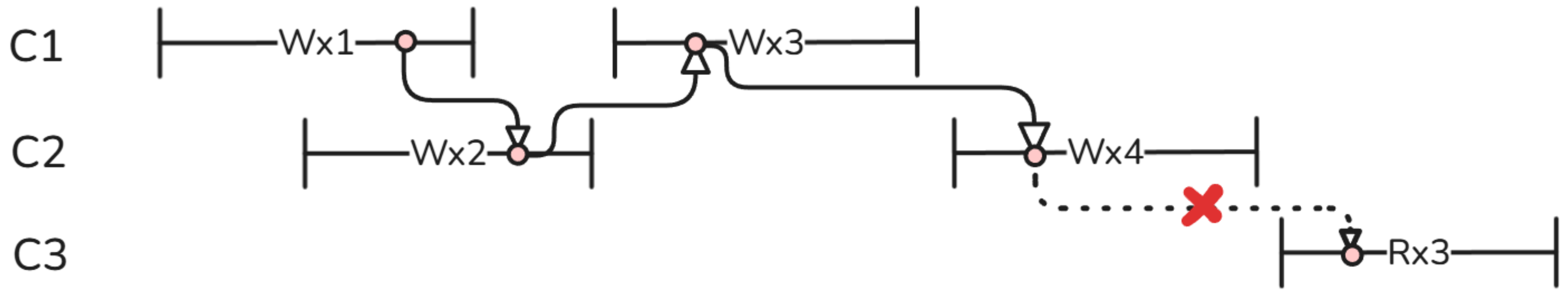
In Depth: Why not ack of max seen instance?

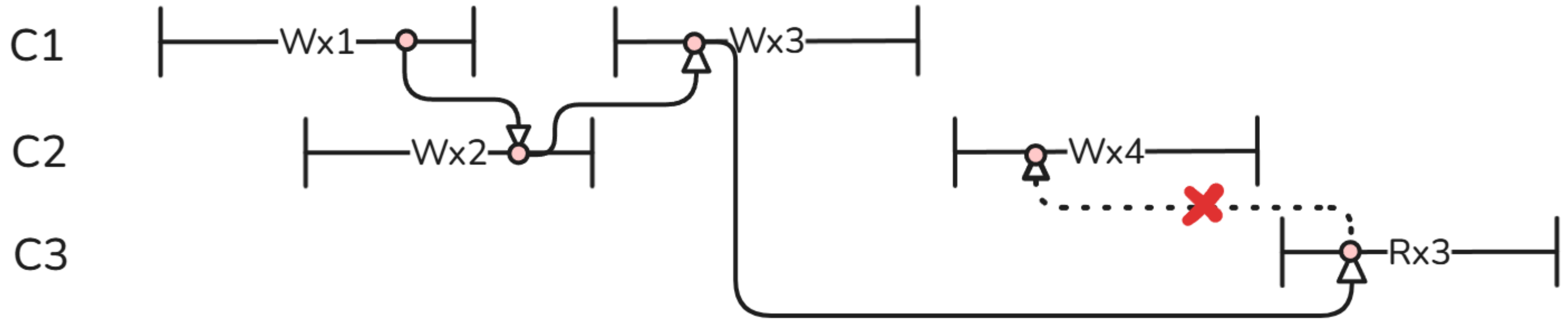


In Depth: Why not ack of max seen instance?



In Depth: Why not ack of max seen instance?





Not Linearizable ✗

In Depth: Reads

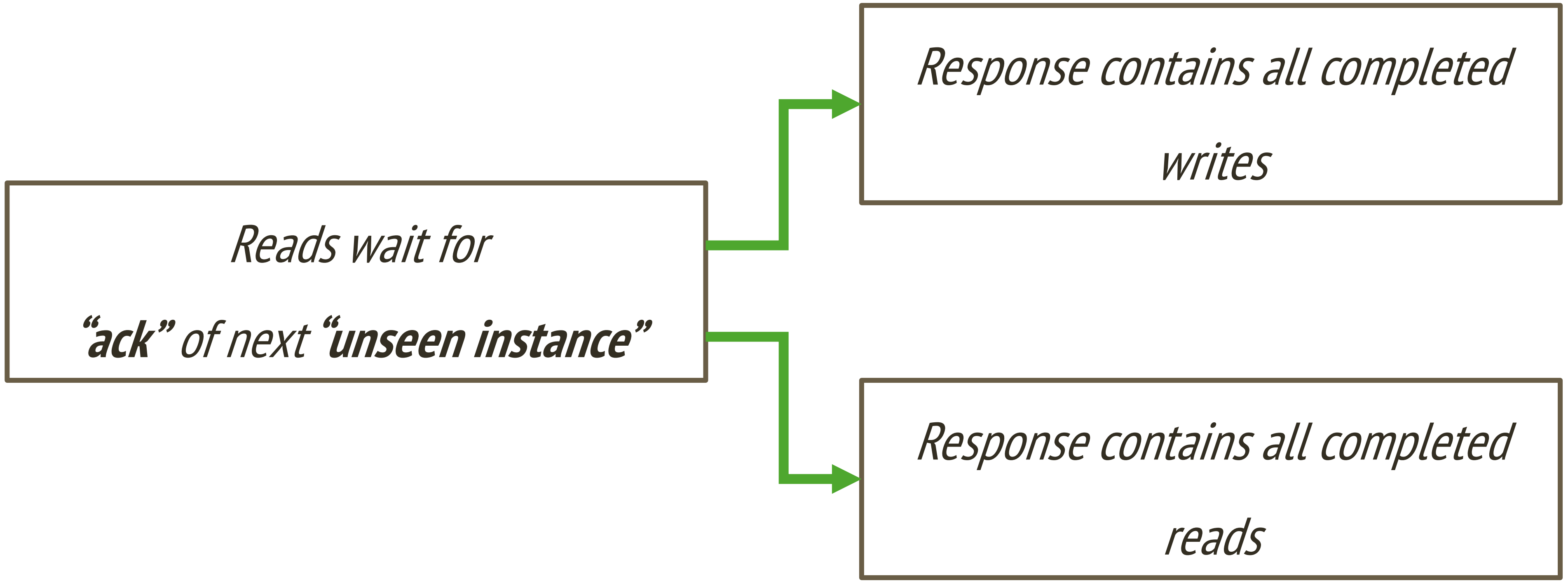
Why acks of next unseen instance?

~~*Why not immediately reply?*~~

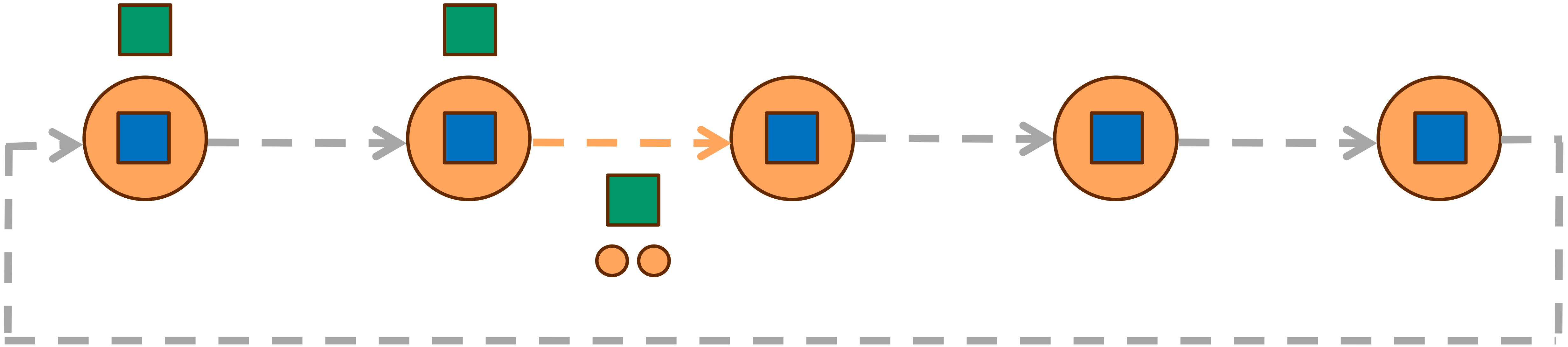
~~*Why not next unseen instance?*~~

~~*Why not ack of max seen instance?*~~

In Depth: Reads

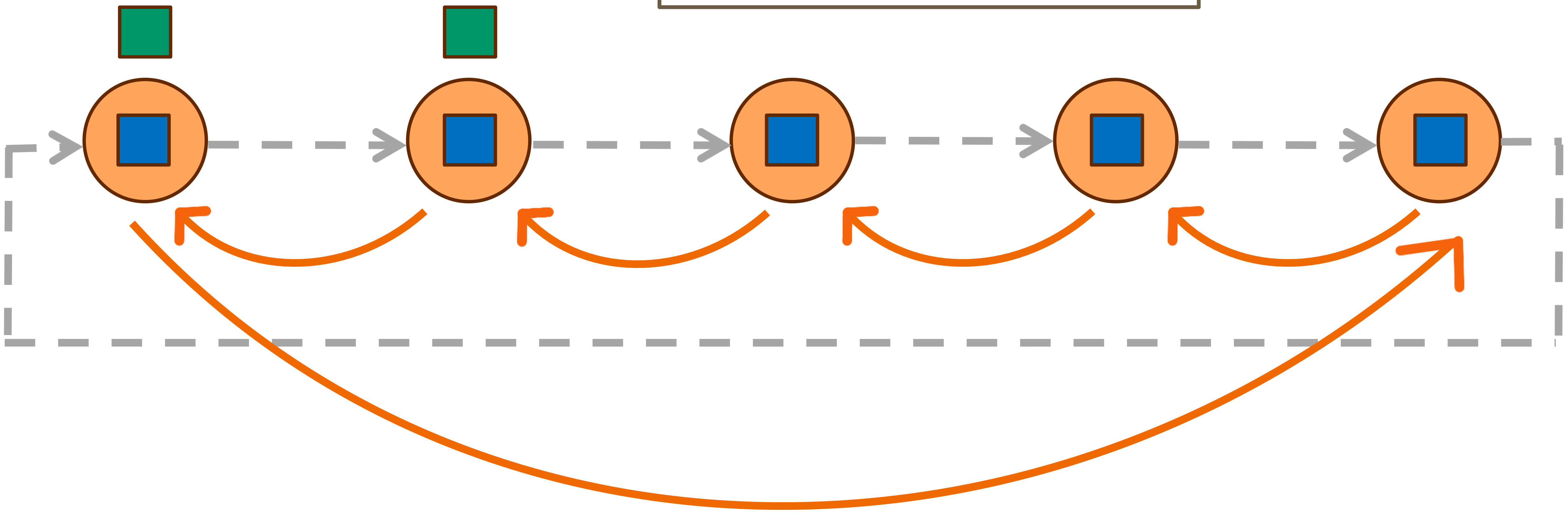

Reads wait for
*“**ack**” of next “**unseen instance**”*
Response contains all completed writes
Response contains all completed reads

Membership Management: Removal

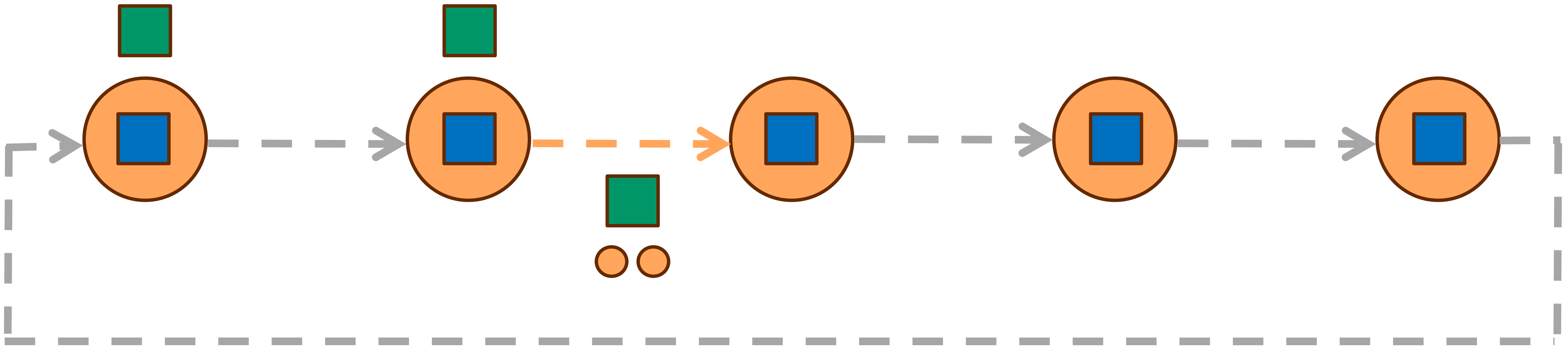


Membership Management: Removal

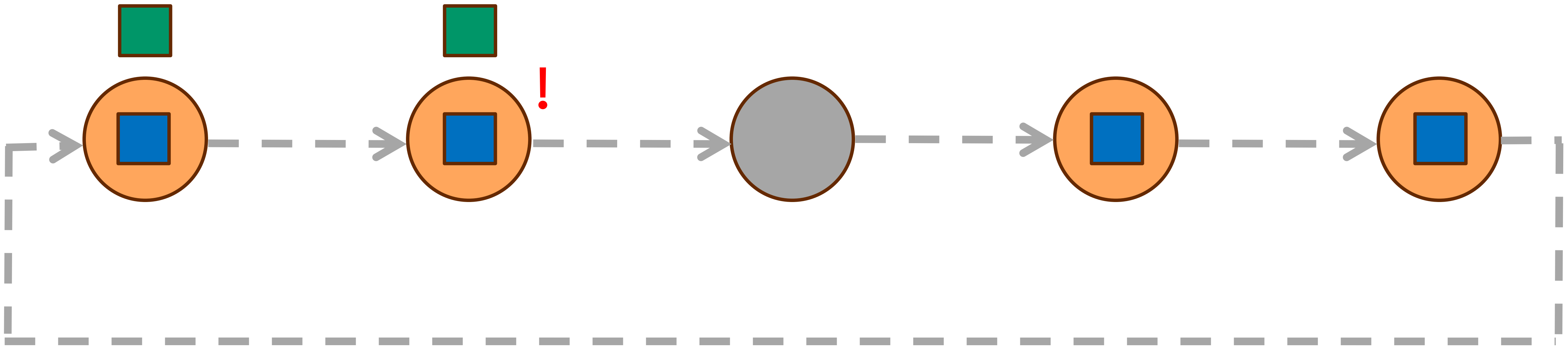
Replicas send "keep-alive" messages to previous replica



Membership Management: Removal

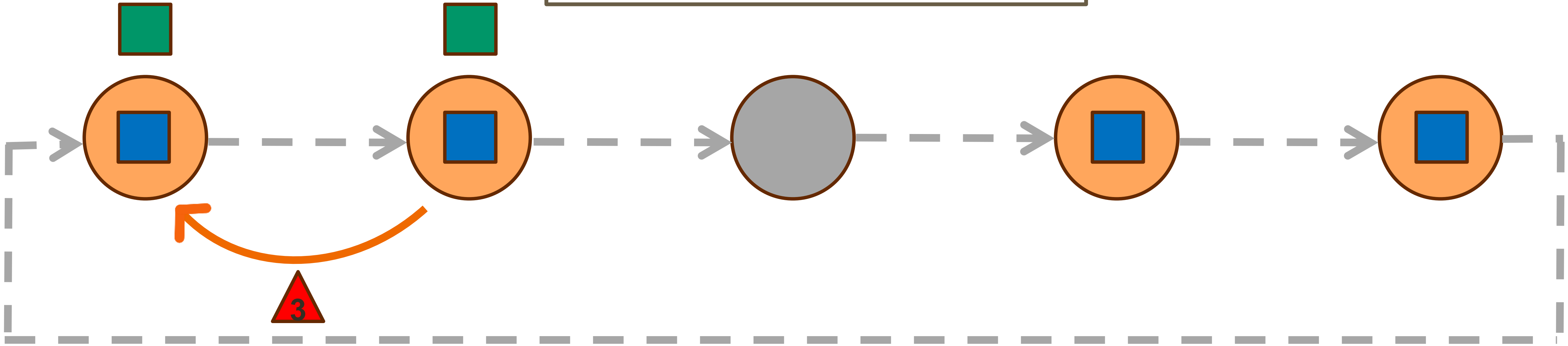


Membership Management: Removal



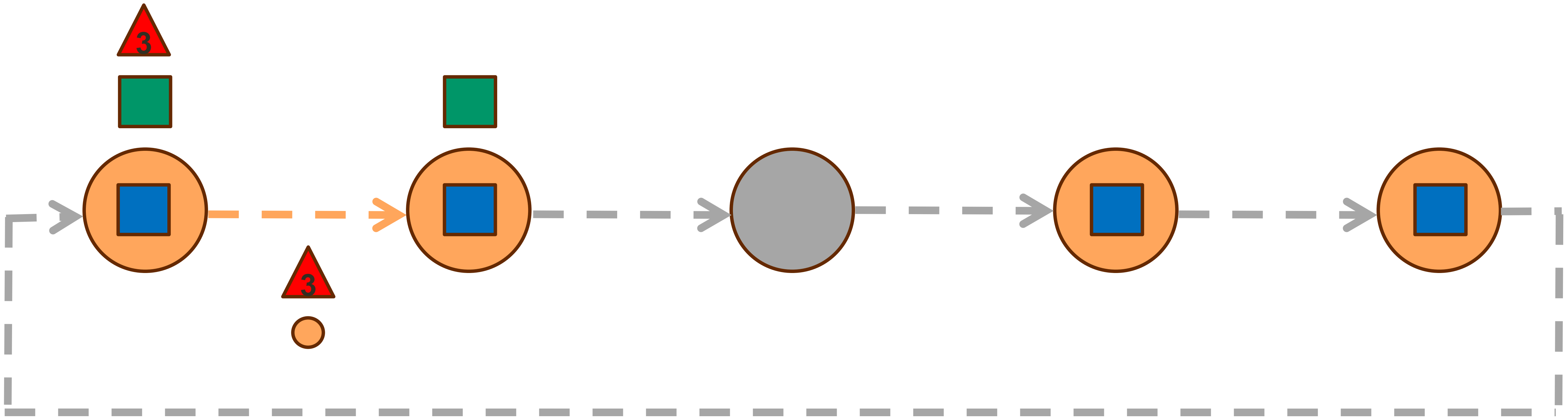
Membership Management: Removal

Remove requests are handled like regular write operations



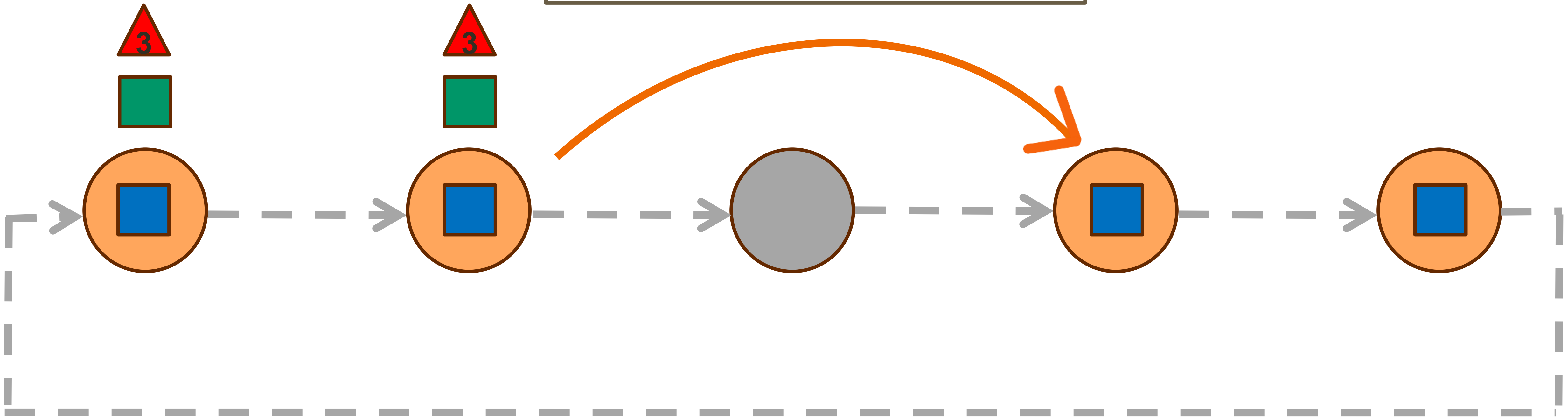
: Remove i'th replica request

Membership Management: Removal

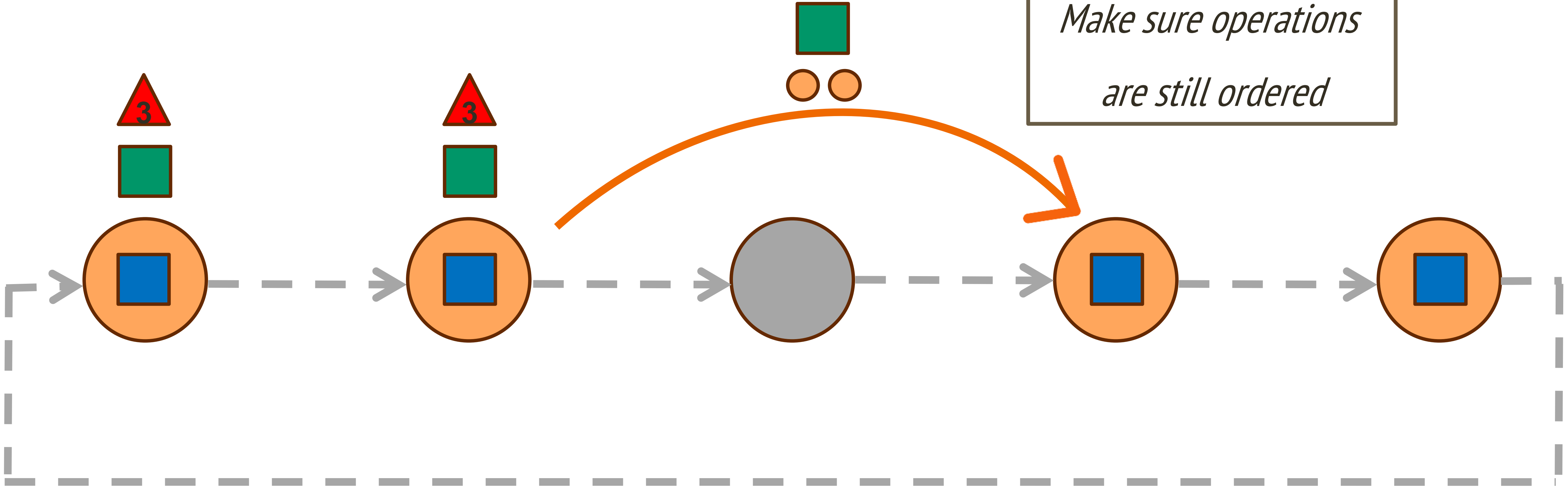


Membership Management: Removal

Skip suspected replica

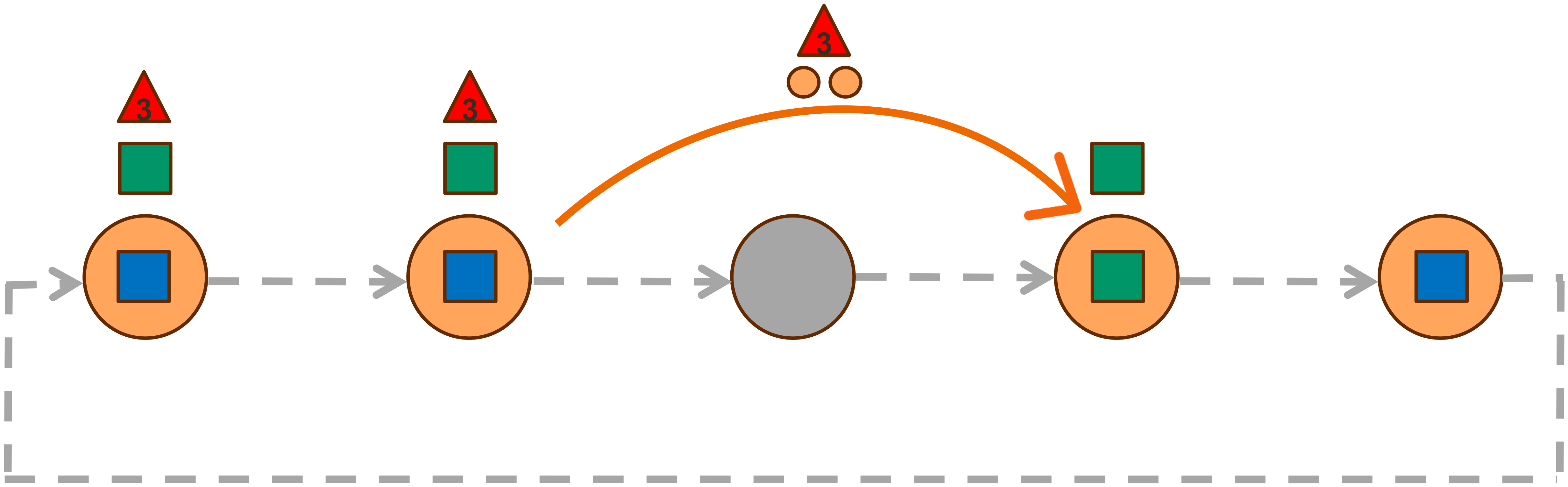


Membership Management: Removal



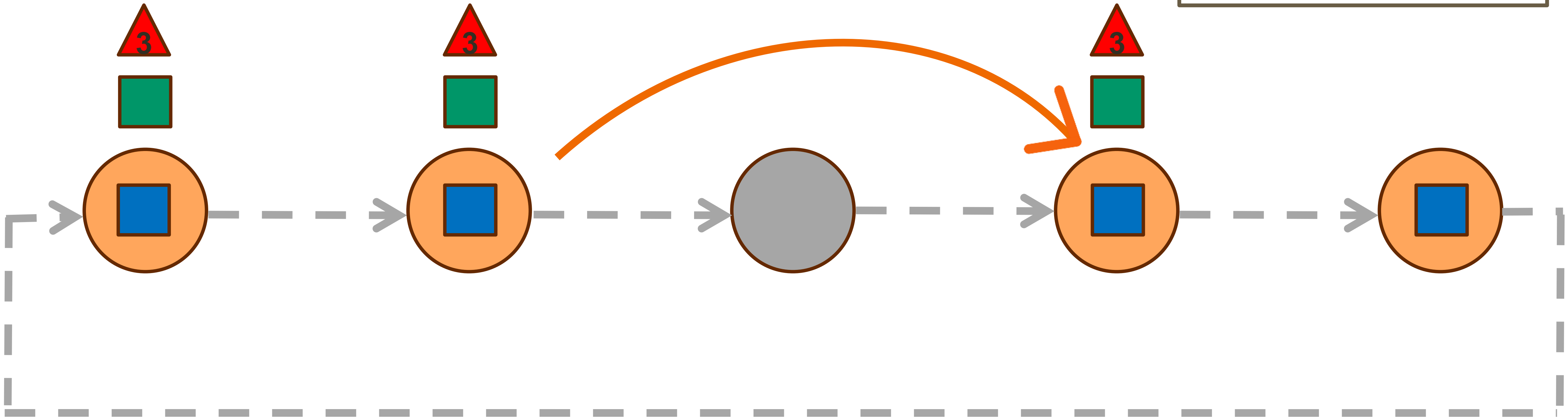
Make sure operations are still ordered

Membership Management: Removal

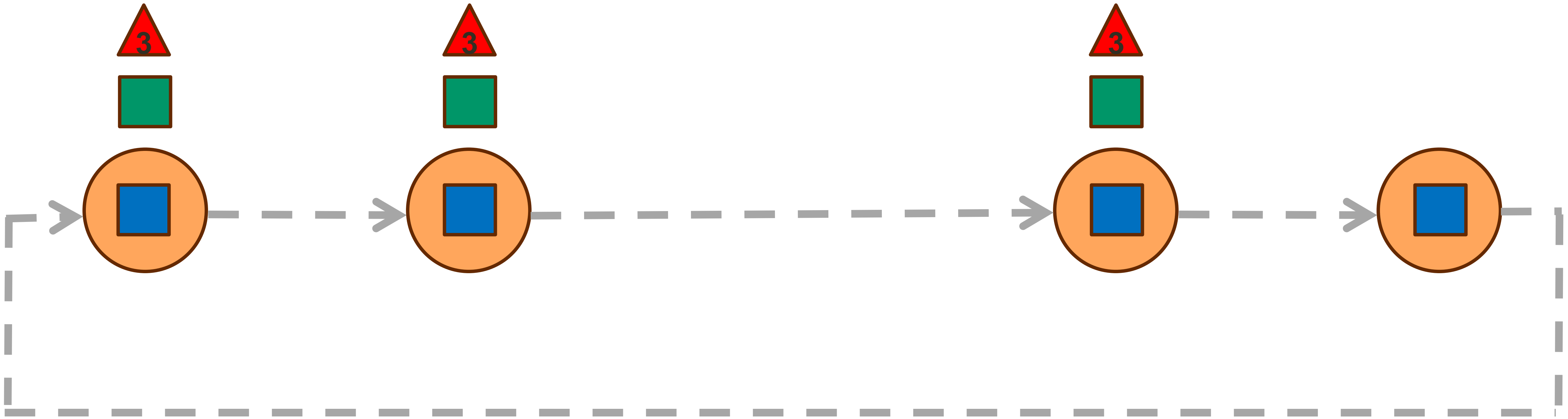


Membership Management: Removal

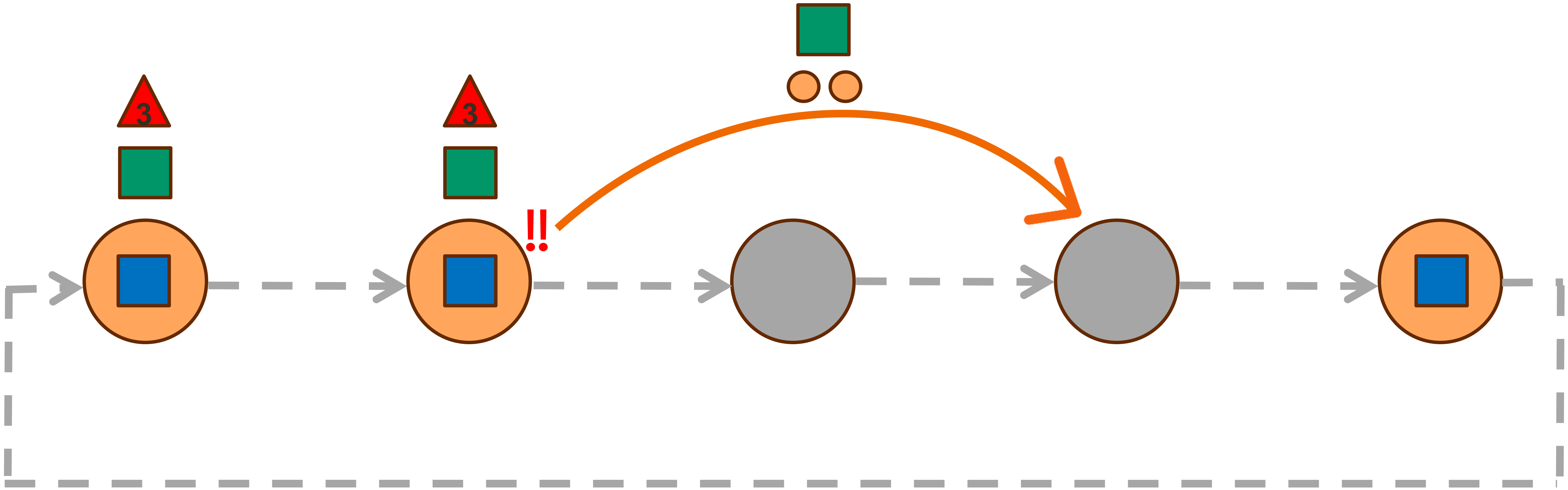
Removal is now decided



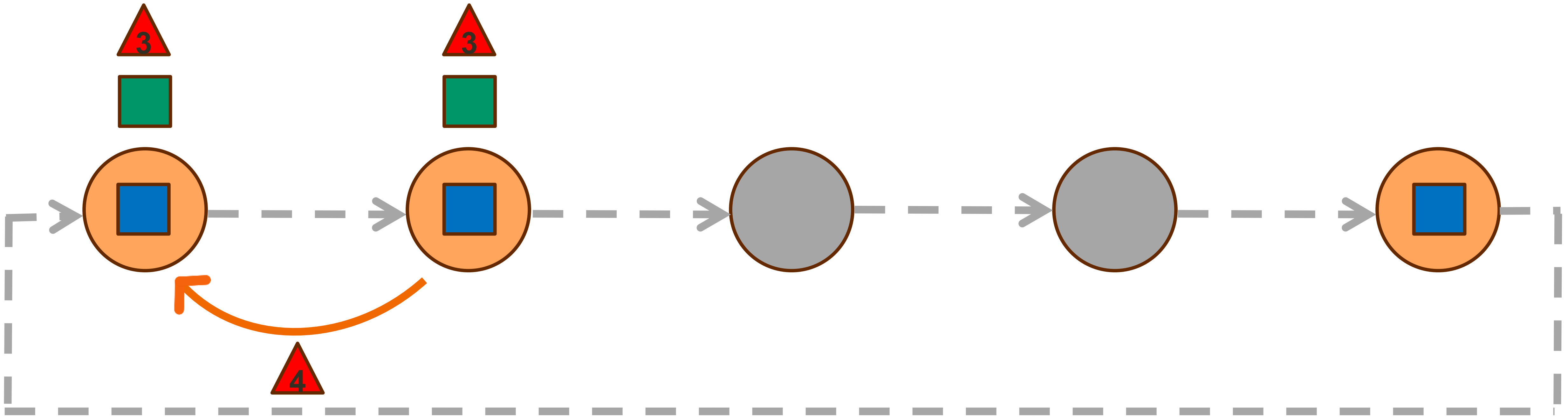
Membership Management: Removal



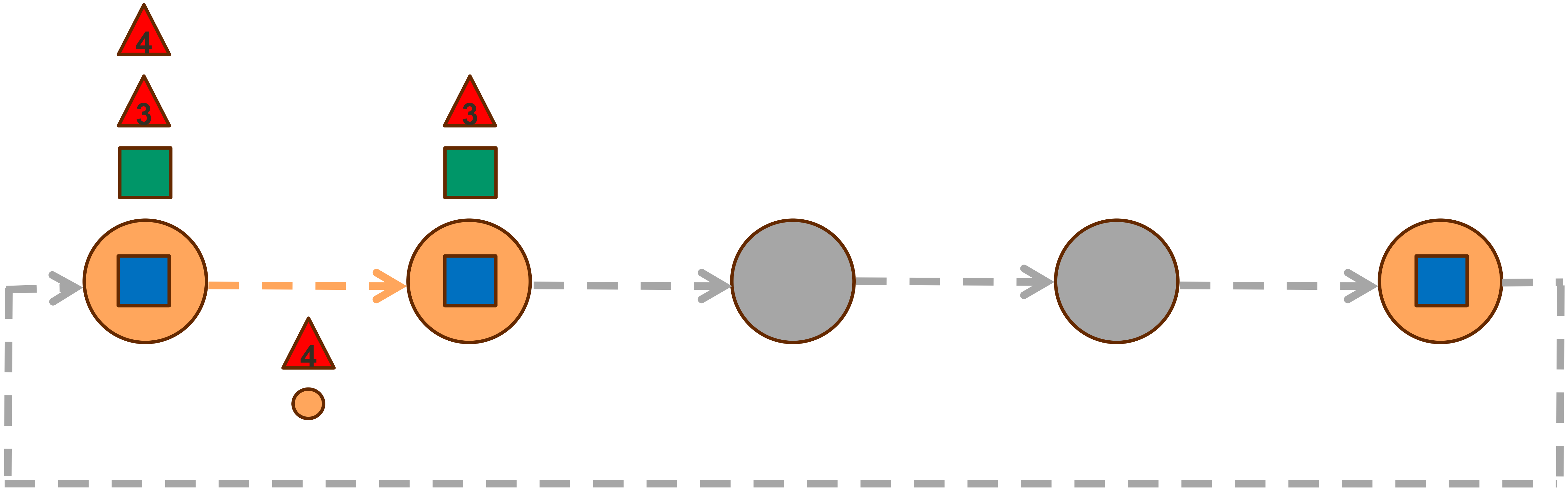
Membership Management: Removal



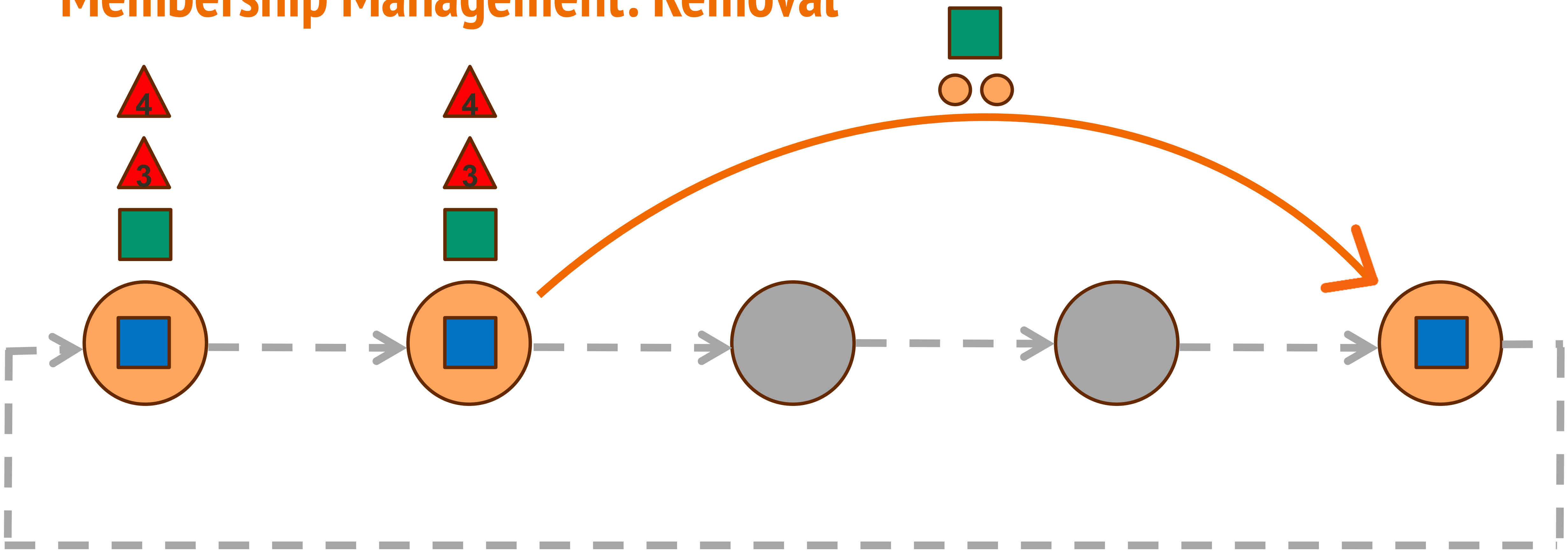
Membership Management: Removal



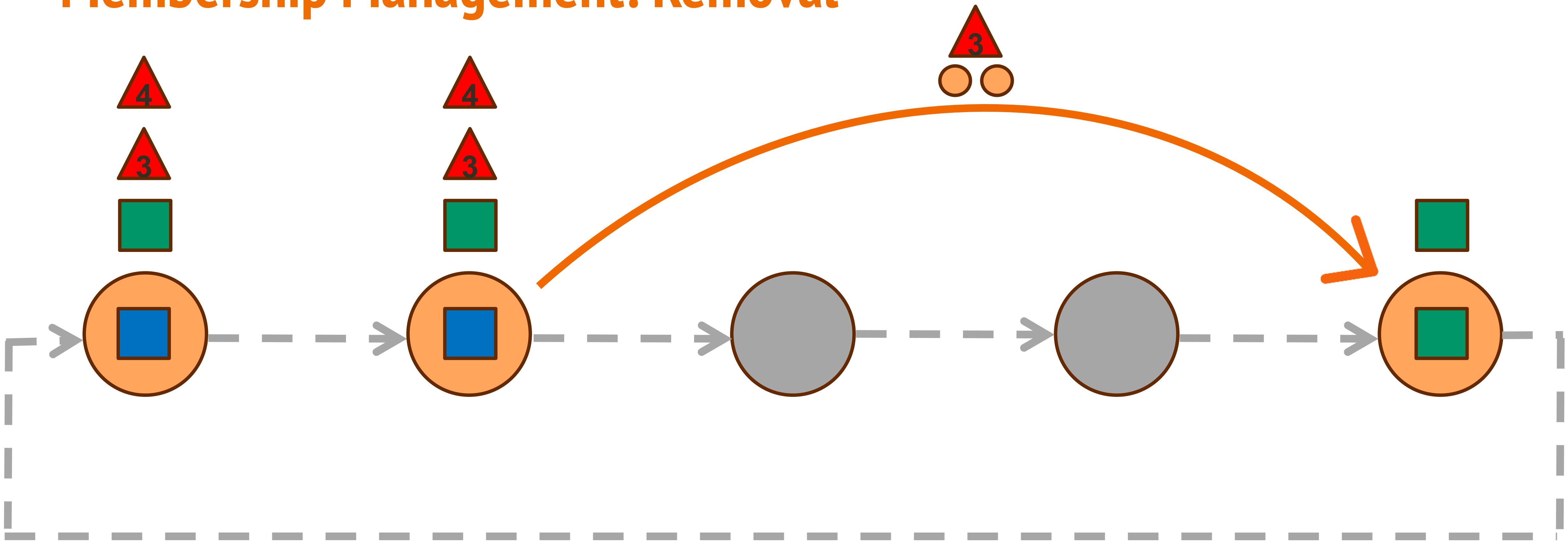
Membership Management: Removal



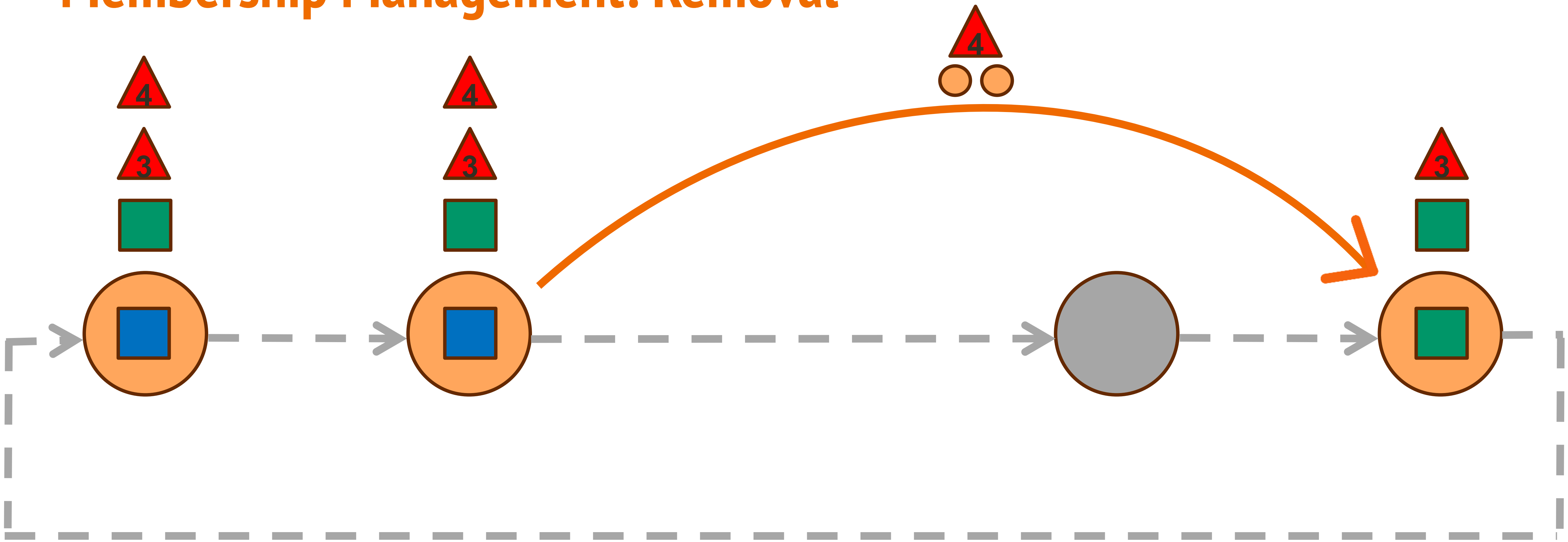
Membership Management: Removal



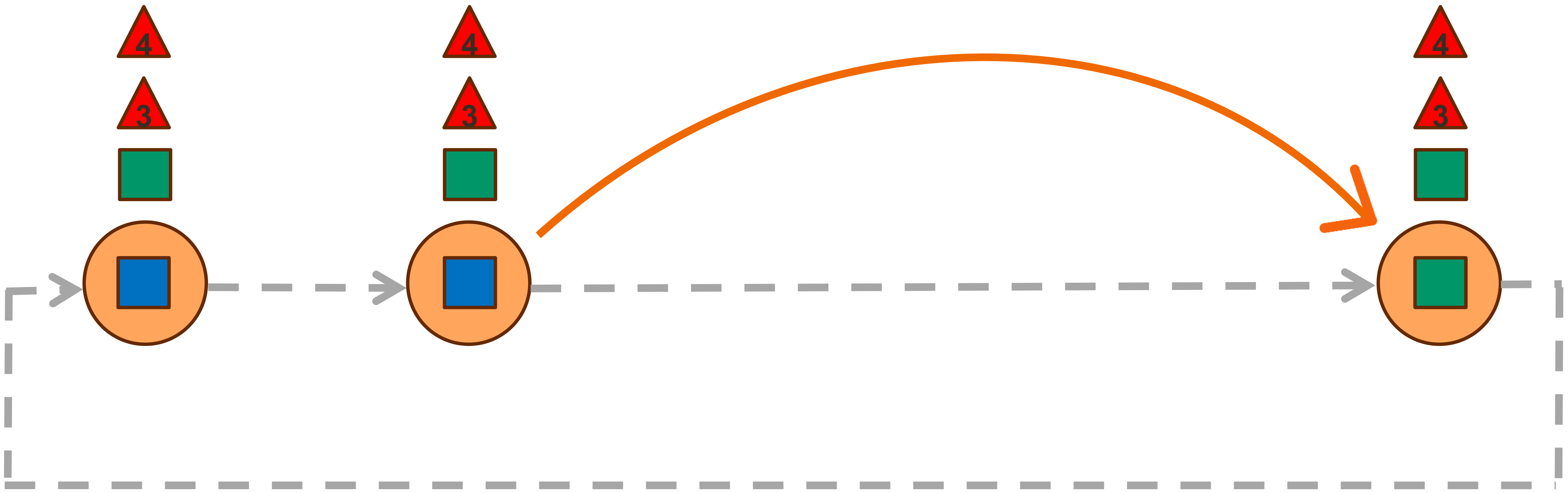
Membership Management: Removal



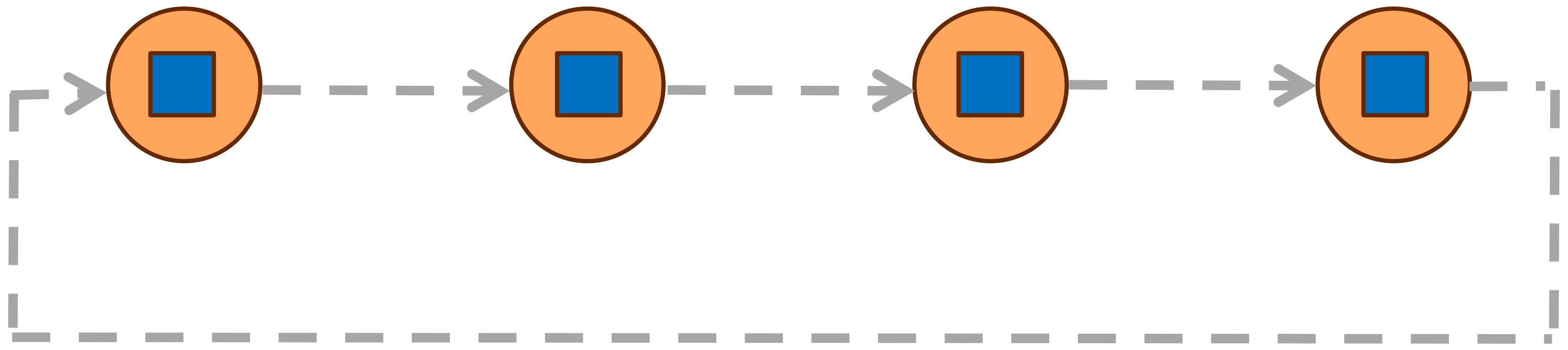
Membership Management: Removal



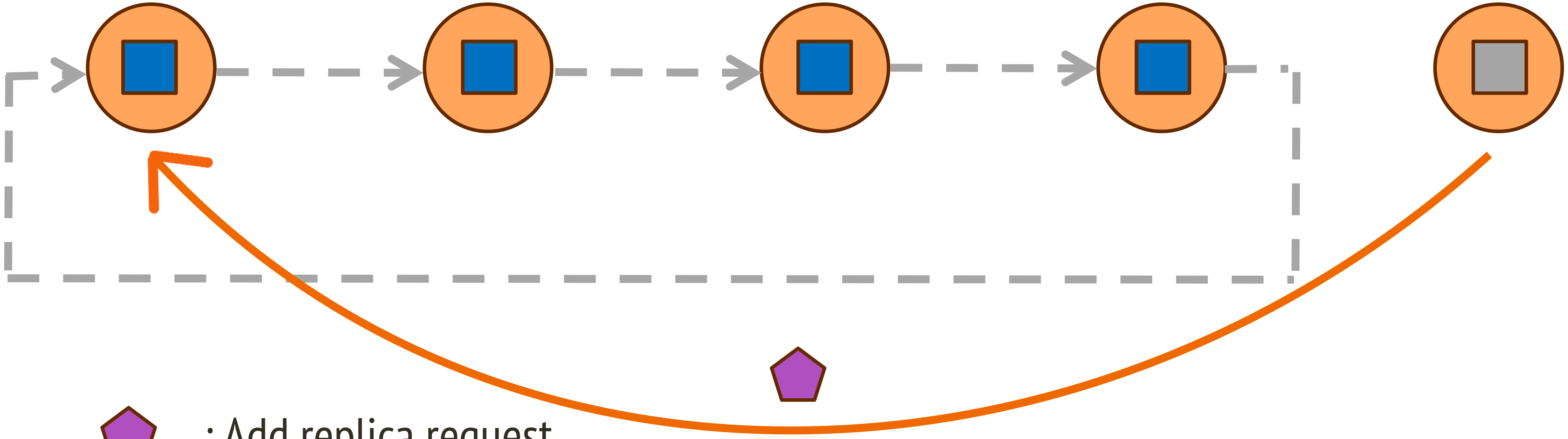
Membership Management: Removal



Membership Management: Addition

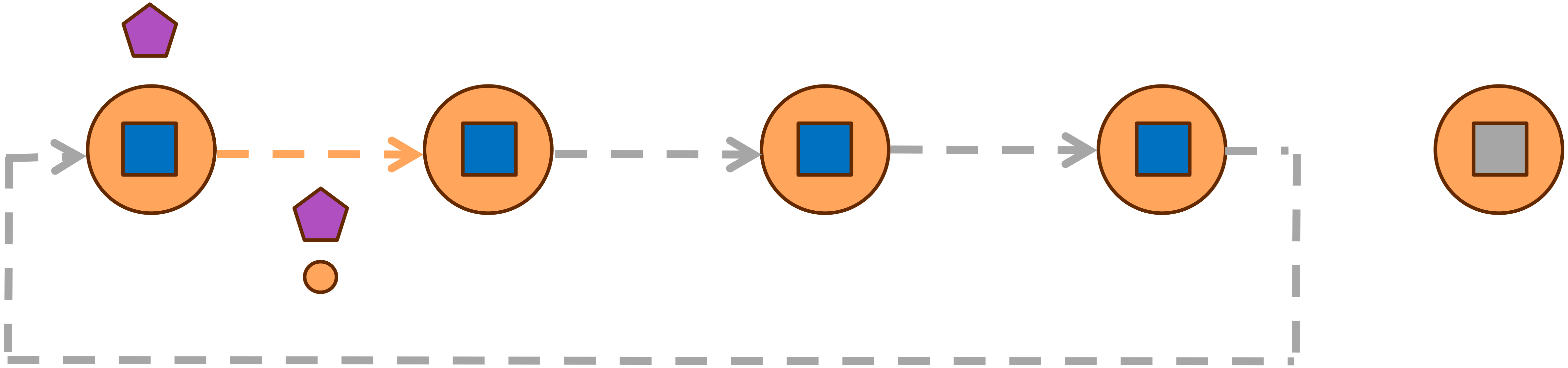


Membership Management: Addition

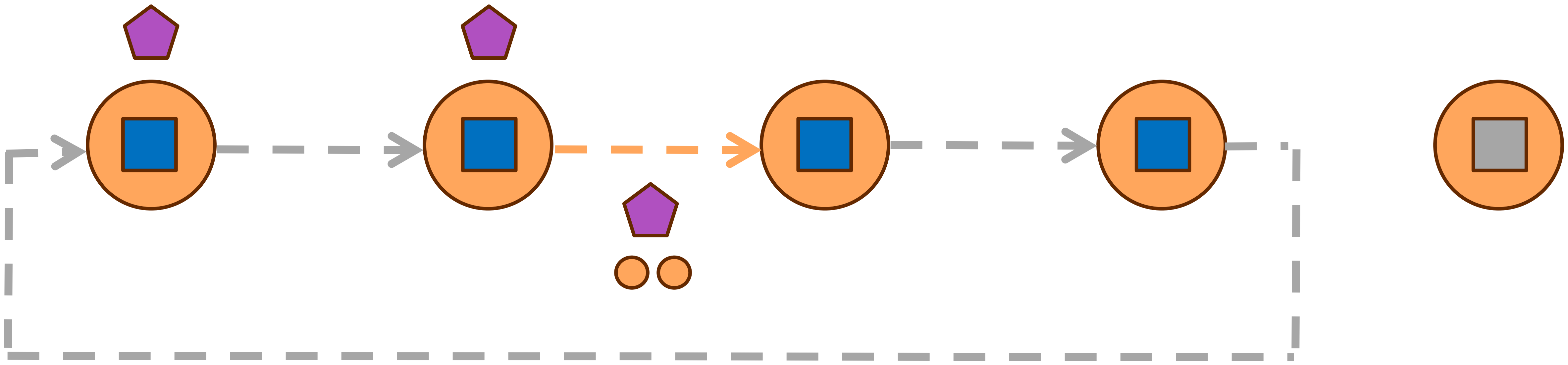


⬟ : Add replica request

Membership Management: Addition

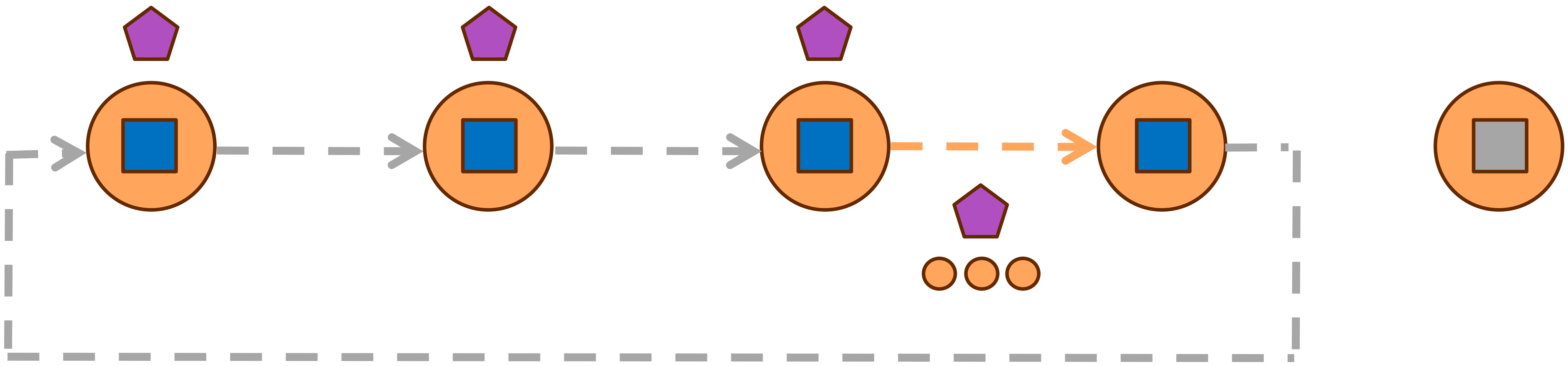


Membership Management: Addition

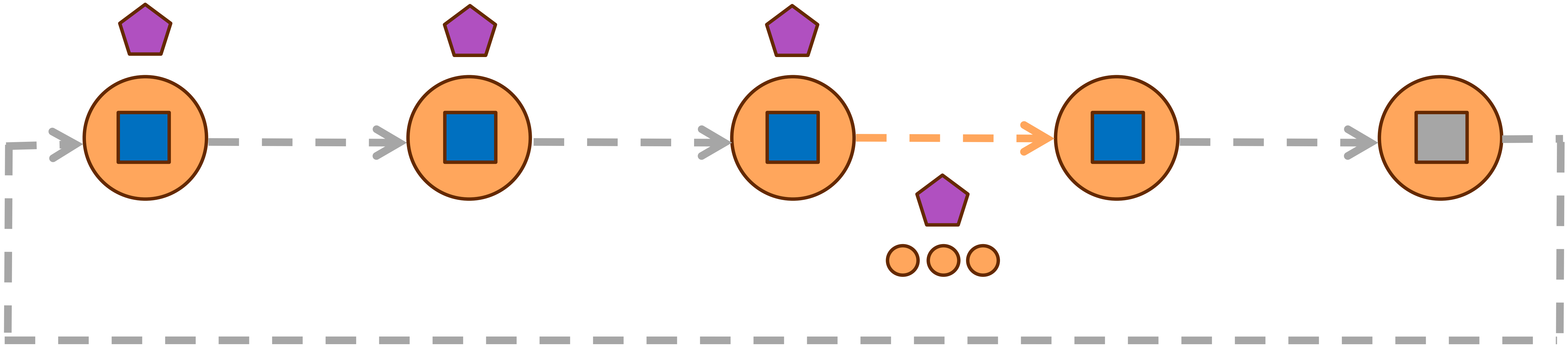


Membership Management: Addition

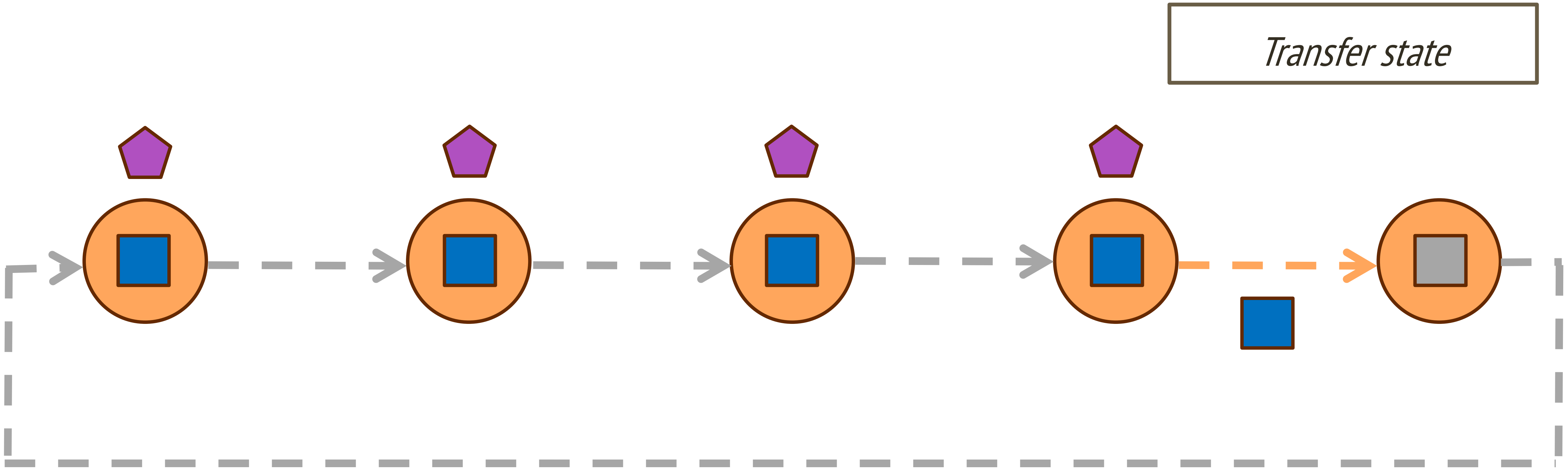
Addition is now decided



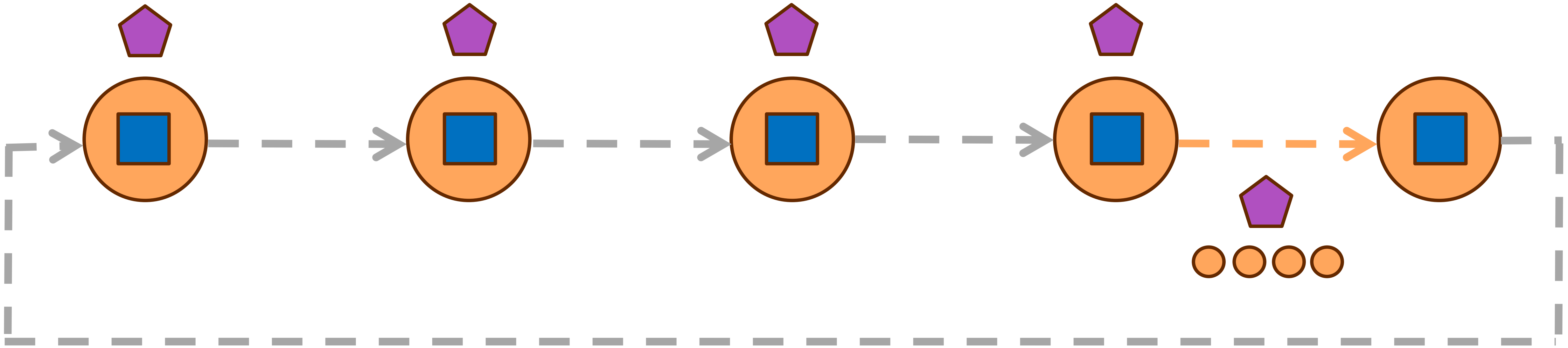
Membership Management: Addition



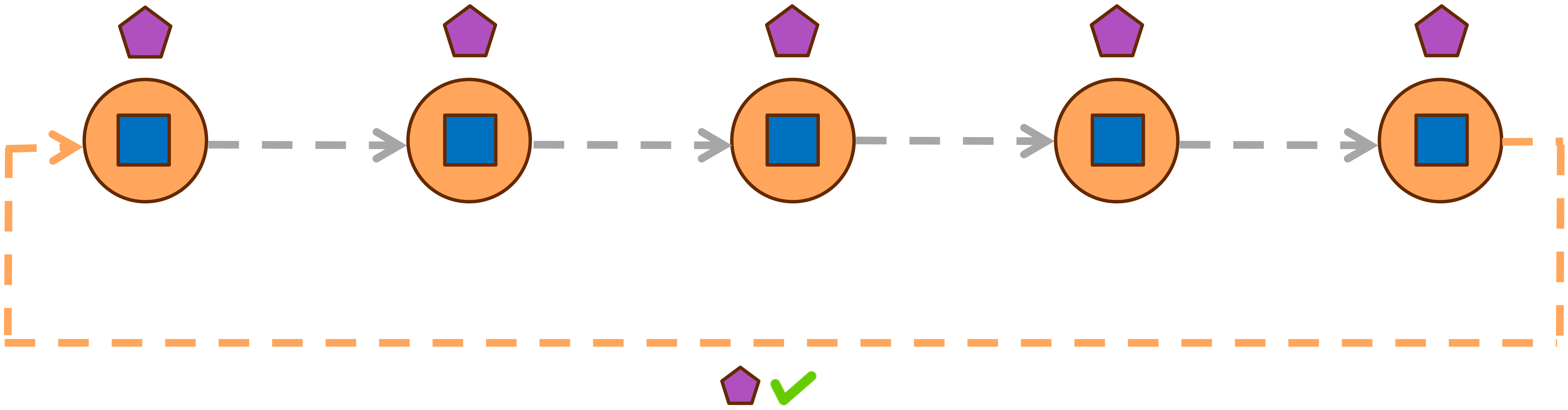
Membership Management: Addition



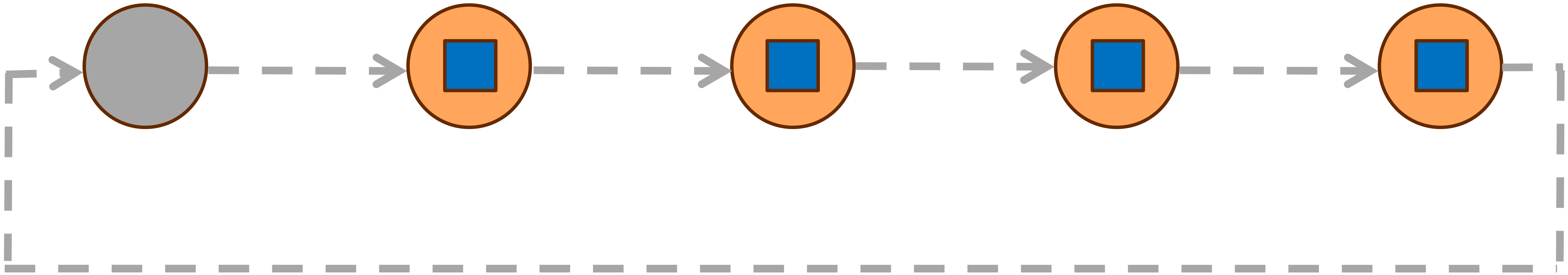
Membership Management: Addition



Membership Management: Addition

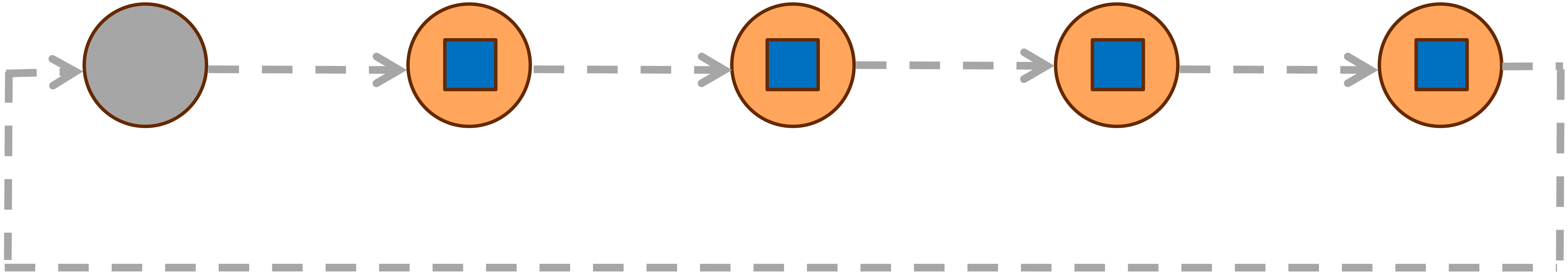


Leader Failure

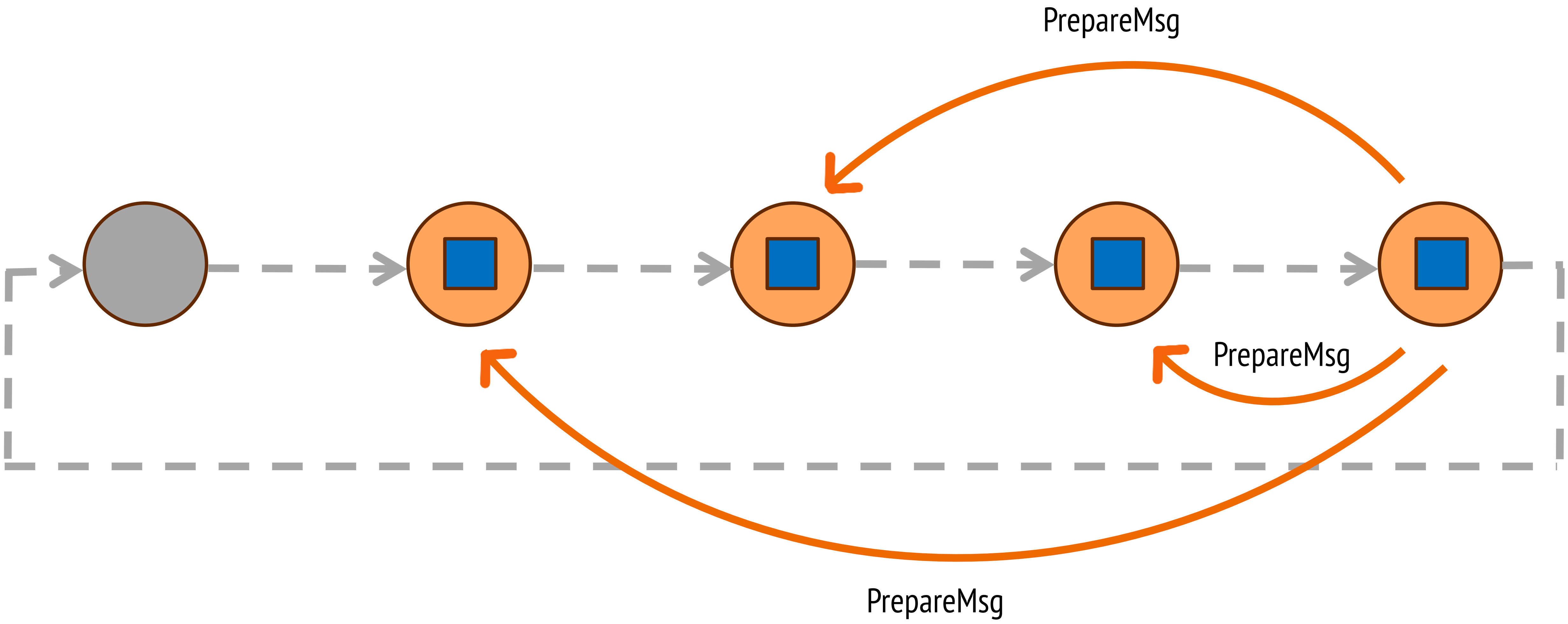


Leader Failure

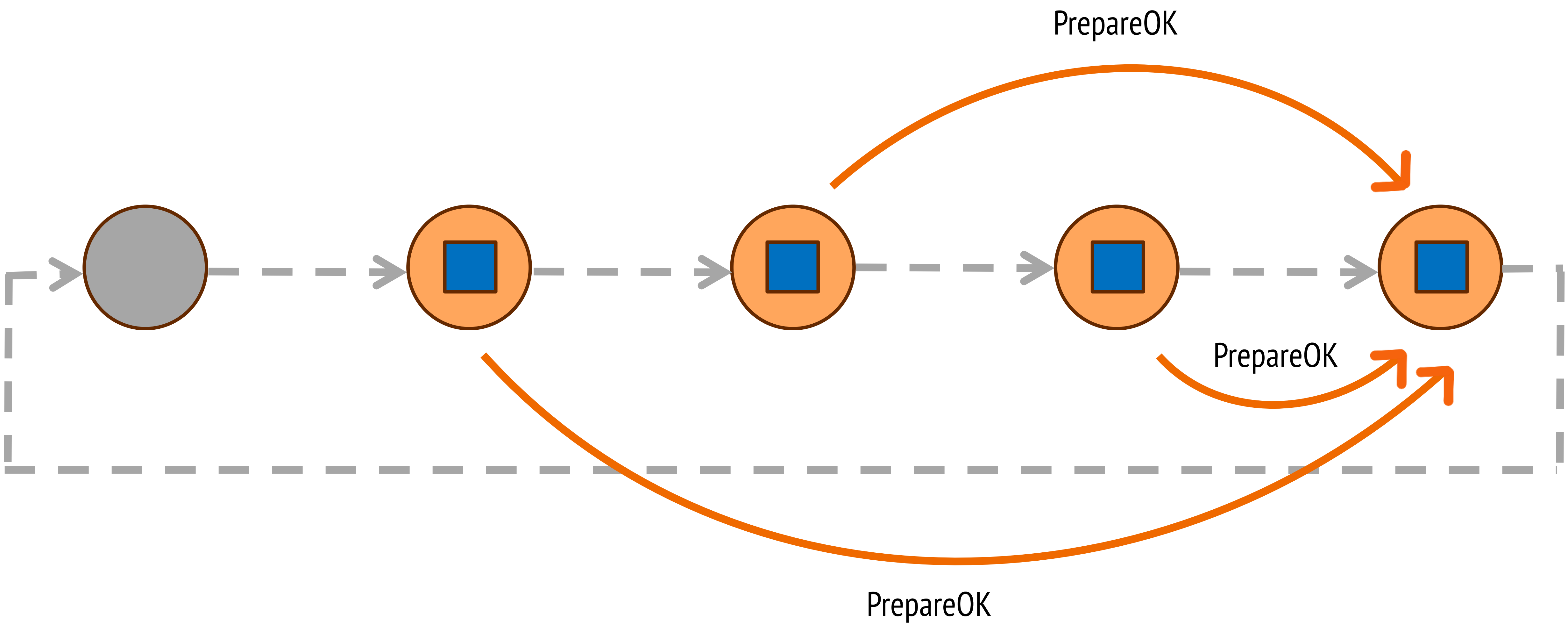
Run Paxos Leader Election



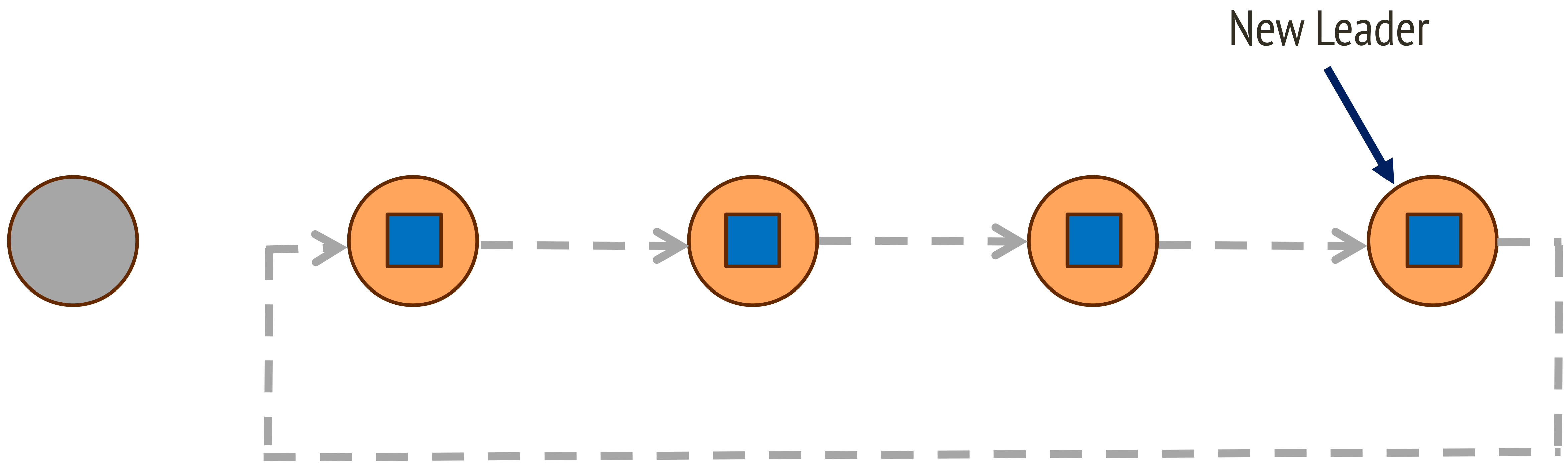
Leader Failure



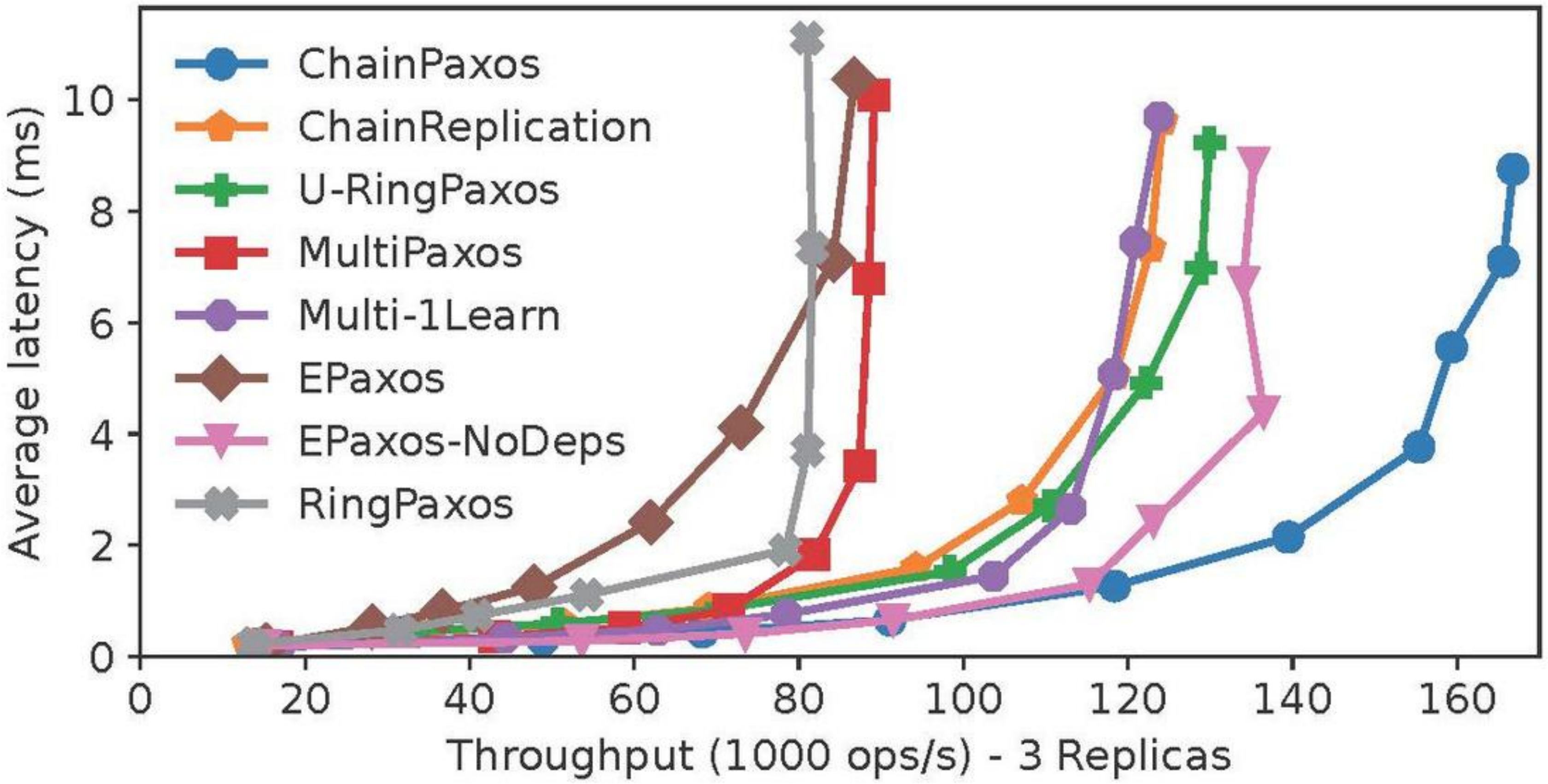
Leader Failure



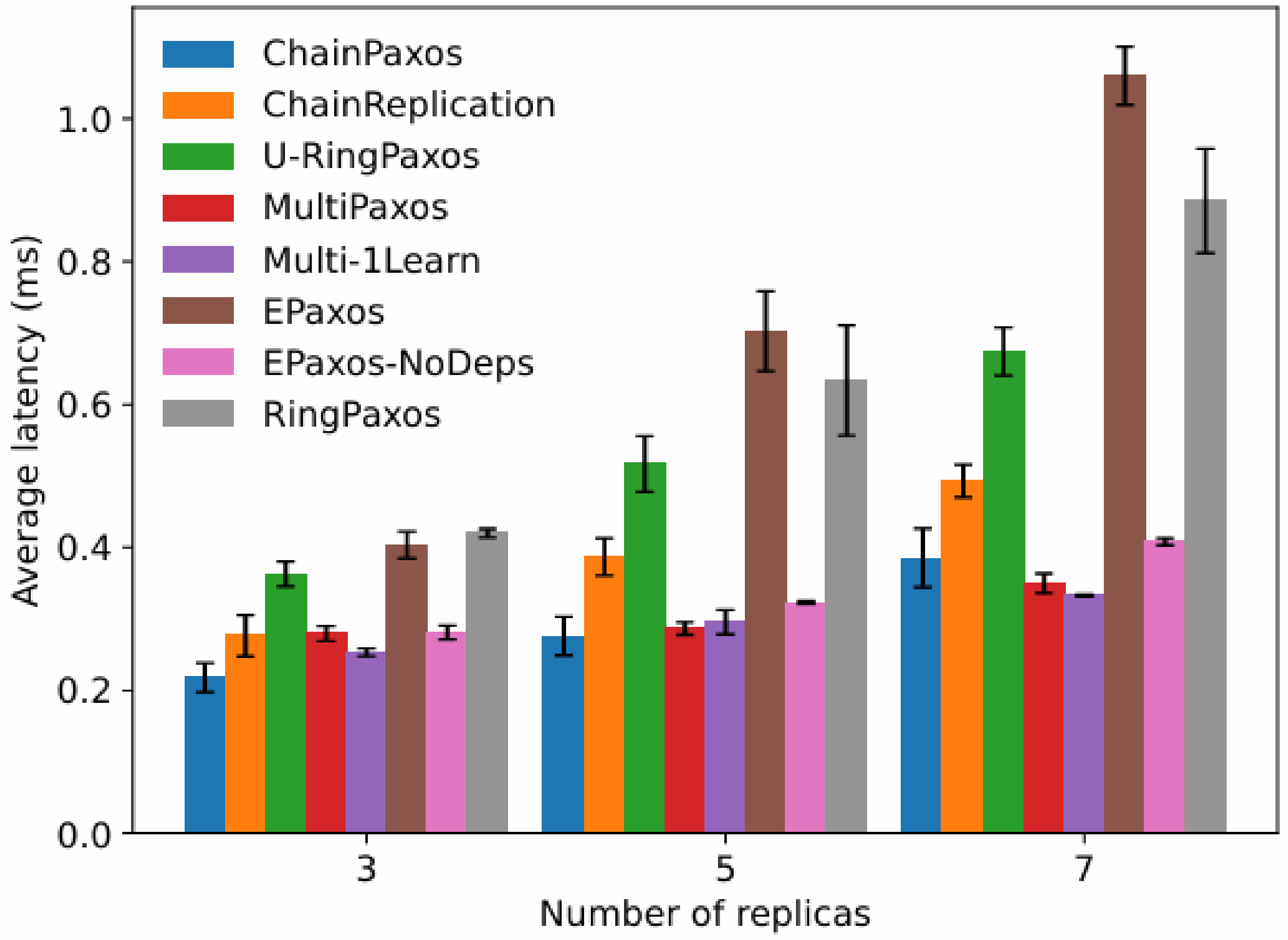
Leader Failure



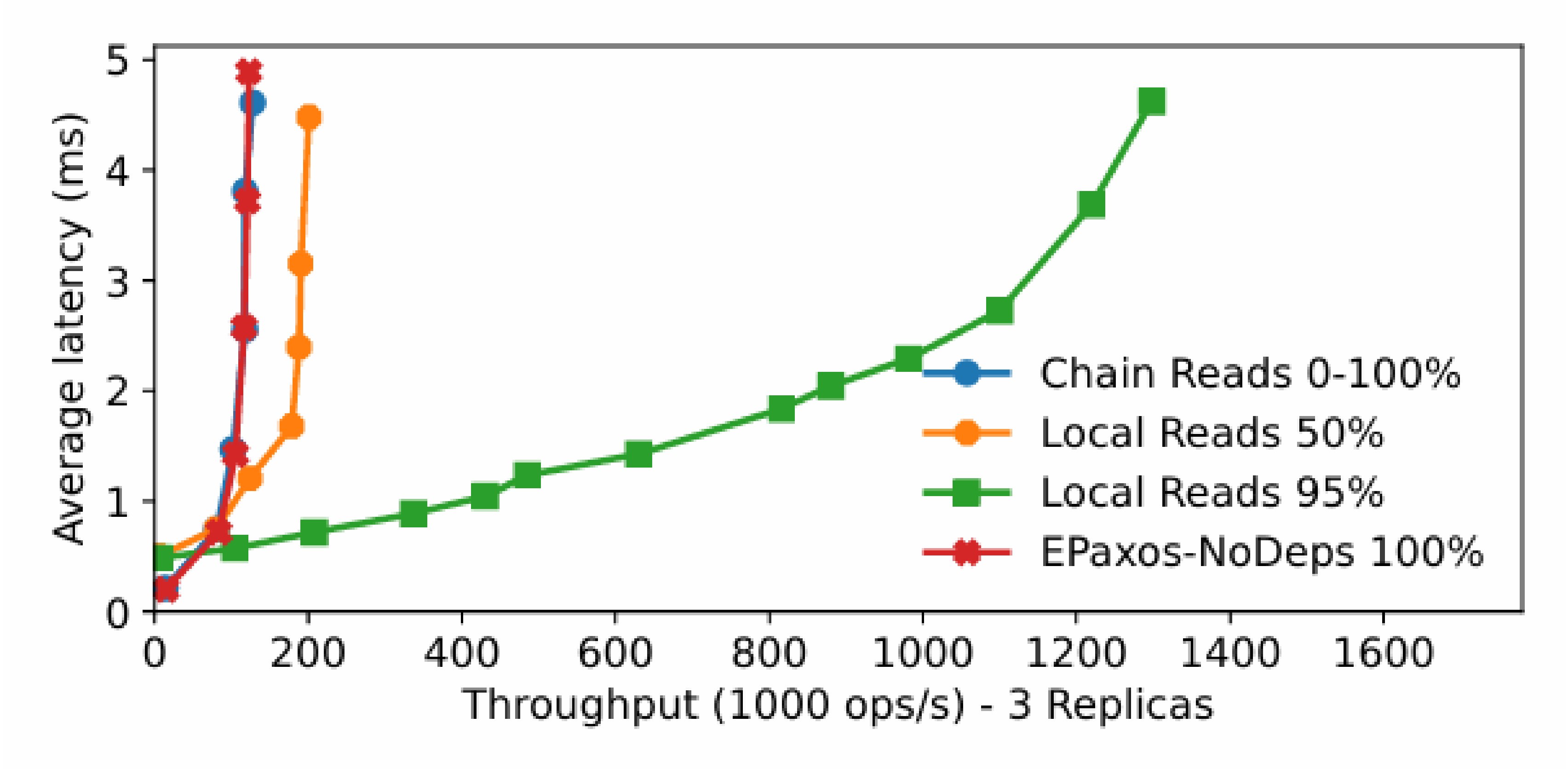
Experiment Results: CPU Bottleneck



Experiment Results: Latency Test



Experiment Results: Local Read Test



References

- Fouto, P., Preguiça, N., & Leitão, J. (2022). High Throughput Replication with Integrated Membership Management. USENIX Annual Technical Conference (USENIX ATC 22), 575–592.

Thank you