Fault-Tolerance, Fast and Slow: Exploiting Failure Asynchrony in Distributed Systems

Ramnatthan Alagappan, Aishwarya Ganesan, Jing Liu,
Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau

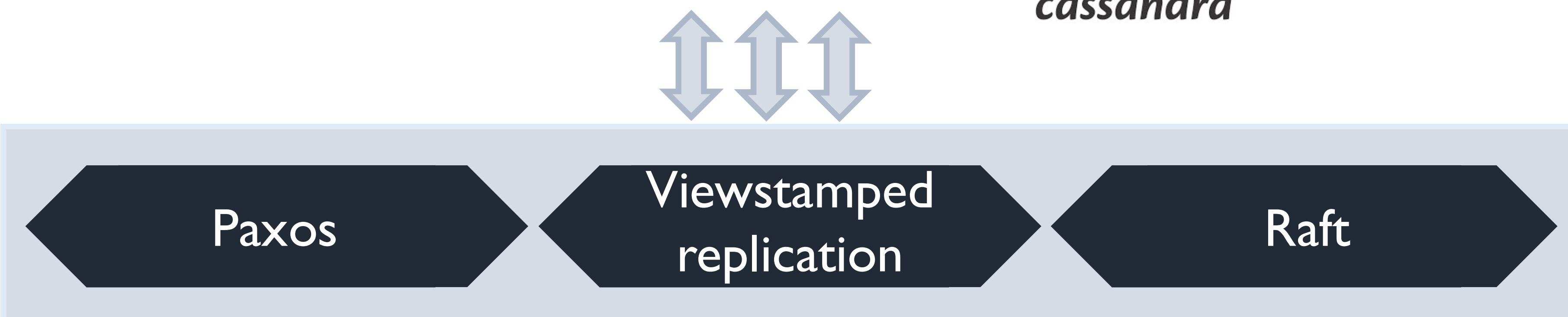
WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Replication Protocols

Replication Protocols

Foundation upon which datacenter systems are built

GFS
Colossus
BigTable



The Two Different Worlds of Replication

The Two Different Worlds of Replication

World-1

World-2

The Two Different Worlds of Replication

How and where to store system state?

World-1

World-2

The Two Different Worlds of Replication

How and where to store system state?

World-1

Disk-durable

synchronously persist updates to disks

World-2

The Two Different Worlds of Replication

How and where to store system state?

World-1

Disk-durable

synchronously persist updates to disks

World-2

Memory-durable

buffer updates only in volatile memory

The Two Different Worlds of Replication

How and where to store system state?

World-1	World-2
Disk-durable	**Memory-durable**
synchronously persist updates to disks	buffer updates only in volatile memory
Paxos, Raft [ATC '14], ZAB [DSN '11], Gaios [NSDI '11], ZooKeeper, etcd, LogCabin …	

The Two Different Worlds of Replication

How and where to store system state?

World-1	World-2
Disk-durable	**Memory-durable**
synchronously persist updates to disks	buffer updates only in volatile memory
Paxos, Raft [ATC '14], ZAB [DSN '11], Gaios [NSDI '11], ZooKeeper, etcd, LogCabin …	Viewstamped replication, NOPaxos [OSDI '16], SpecPaxos [NSDI '15] …

The Two Different Worlds of Replication

How and where to store system state?

World-1

Disk-durable

synchronously persist updates to disks

Paxos, Raft [ATC '14], ZAB [DSN '11], Gaios [NSDI '11], ZooKeeper, etcd, LogCabin …

World-2

Memory-durable

buffer updates only in volatile memory

Viewstamped replication, NOPaxos [OSDI '16], SpecPaxos [NSDI '15] …

Neither approach is ideal: reliable *or* performant

The Two Different Worlds of Replication

How and where to store system state?

World-1

Disk-durable

synchronously persist updates to disks

Paxos, Raft [ATC '14], ZAB [DSN '11], Gaios [NSDI '11], ZooKeeper, etcd, LogCabin …

safe but suffer from poor performance

World-2

Memory-durable

buffer updates only in volatile memory

Viewstamped replication, NOPaxos [OSDI '16], SpecPaxos [NSDI '15] …

Neither approach is ideal: reliable *or* performant

The Two Different Worlds of Replication

How and where to store system state?

World-1

Disk-durable

synchronously persist updates to disks

Paxos, Raft [ATC '14], ZAB [DSN '11], Gaios [NSDI '11], ZooKeeper, etcd, LogCabin …

safe but suffer from poor performance

World-2

Memory-durable

buffer updates only in volatile memory

Viewstamped replication, NOPaxos [OSDI '16], SpecPaxos [NSDI '15] …

performant but risk unsafety or unavailability

Neither approach is ideal: reliable *or* performant

Can a protocol provide strong reliability while maintaining high performance?

SAUCR:
Situation-Aware Updates and Crash Recovery

SAUCR:
Situation-Aware Updates and Crash Recovery

Simple insight: dynamically (based on the situation) decide how to commit updates

SAUCR: Situation-Aware Updates and Crash Recovery

Simple insight: dynamically (based on the situation) decide how to commit updates

- ➡ with many or all nodes up, buffer in memory – fast mode
- ➡ with failures, if only bare majority up, flush to disk – slow mode

SAUCR:
Situation-Aware Updates and Crash Recovery

Simple insight: dynamically (based on the situation) decide how to commit updates

- ➡ with many or all nodes up, buffer in memory – fast mode
- ➡ with failures, if only bare majority up, flush to disk – slow mode

Strong reliability while maintaining high performance

Simultaneity of Failures

SAUCR's effectiveness depends upon simultaneity of failures

Simultaneity of Failures

SAUCR's effectiveness depends upon simultaneity of failures

- ➡ independent and non-simultaneous correlated (gap of a few milliseconds to a few seconds)
  - ➡ can react and switch from fast to slow mode
  - ➡ preserves durability and availability

Simultaneity of Failures

SAUCR's effectiveness depends upon simultaneity of failures

- independent and non-simultaneous correlated (gap of a few milliseconds to a few seconds)
  - can react and switch from fast to slow mode
  - preserves durability and availability
- many truly simultaneous correlated
  - no gap and so cannot react
  - remain unavailable

Simultaneity of Failures

SAUCR's effectiveness depends upon simultaneity of failures

- independent and non-simultaneous correlated (gap of a few milliseconds to a few seconds)
  - can react and switch from fast to slow mode
  - preserves durability and availability
- many truly simultaneous correlated
  - no gap and so cannot react
  - remain unavailable
- however, existing data hints they are extremely rare – the Non-Simultaneity Conjecture

Results

Results

Implemented in ZooKeeper

Results

Implemented in ZooKeeper

SAUCR improves reliability compared to memory-durable systems

- ➡ durable and available in 100s of crash scenarios
- ➡ memory-durable loses data or becomes unavailable

Results

Implemented in ZooKeeper

SAUCR **improves reliability** compared to memory-durable systems

- ➡ durable and available in 100s of crash scenarios
- ➡ memory-durable loses data or becomes unavailable

Improvements at **no** or **little cost**

- ➡ overheads within **0%-9%** of memory-durable systems

Results

Implemented in ZooKeeper

SAUCR improves reliability compared to memory-durable systems

- ➡ durable and available in 100s of crash scenarios
- ➡ memory-durable loses data or becomes unavailable

Improvements at no or little cost

- ➡ overheads within 0%-9% of memory-durable systems

Compared to disk-durable

- ➡ slight reduction in availability in extremely rare cases
- ➡ improves performance – 2.5x on SSDs, 100x on HDDs

Outline

Introduction

Distributed updates and crash recovery

- disk-durable protocols
- memory-durable protocols

Situation-aware updates and crash recovery

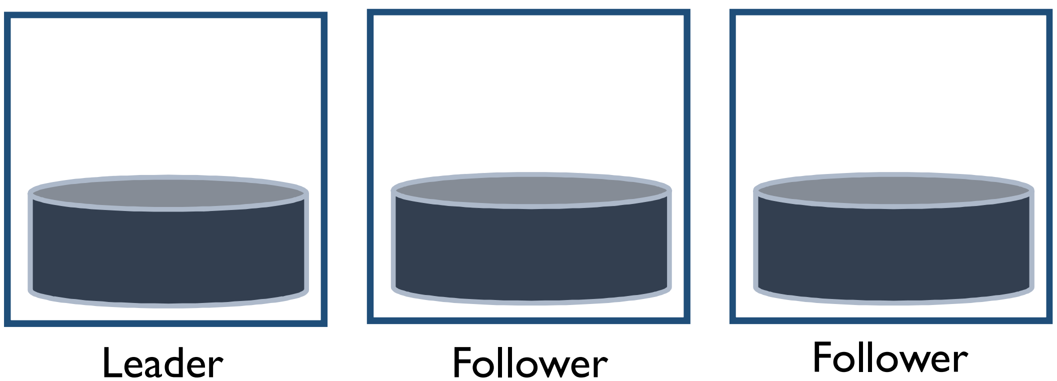
Results

Summary and conclusion

Disk-Durable Protocols

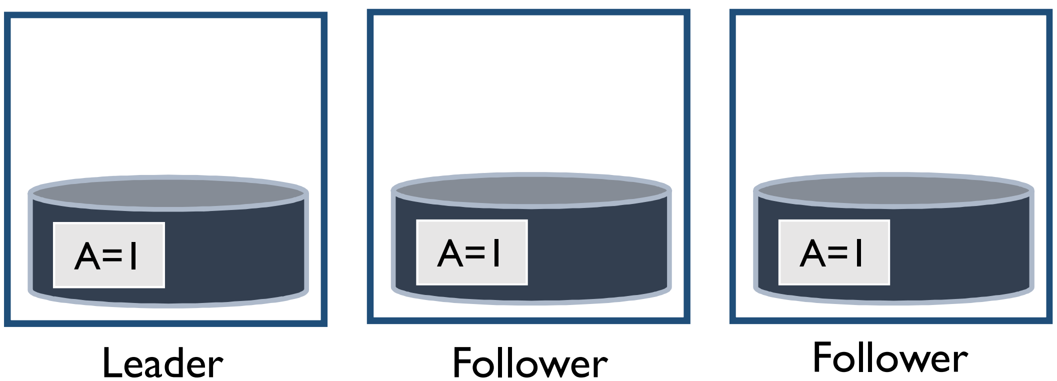
Disk-Durable Protocols

Update



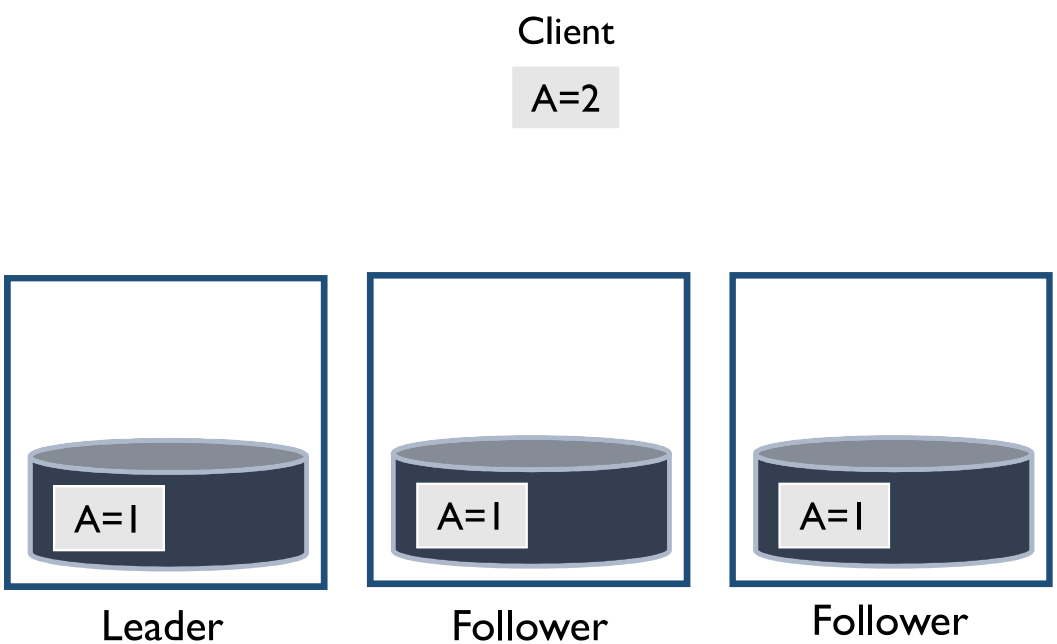
Disk-Durable Protocols

Update



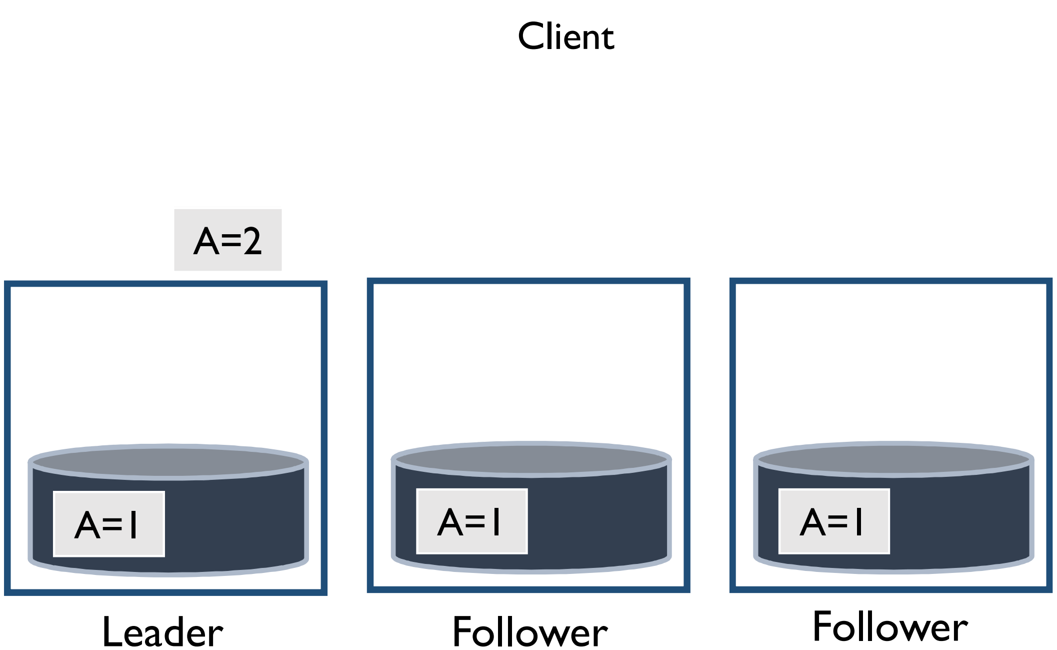
Disk-Durable Protocols

Update



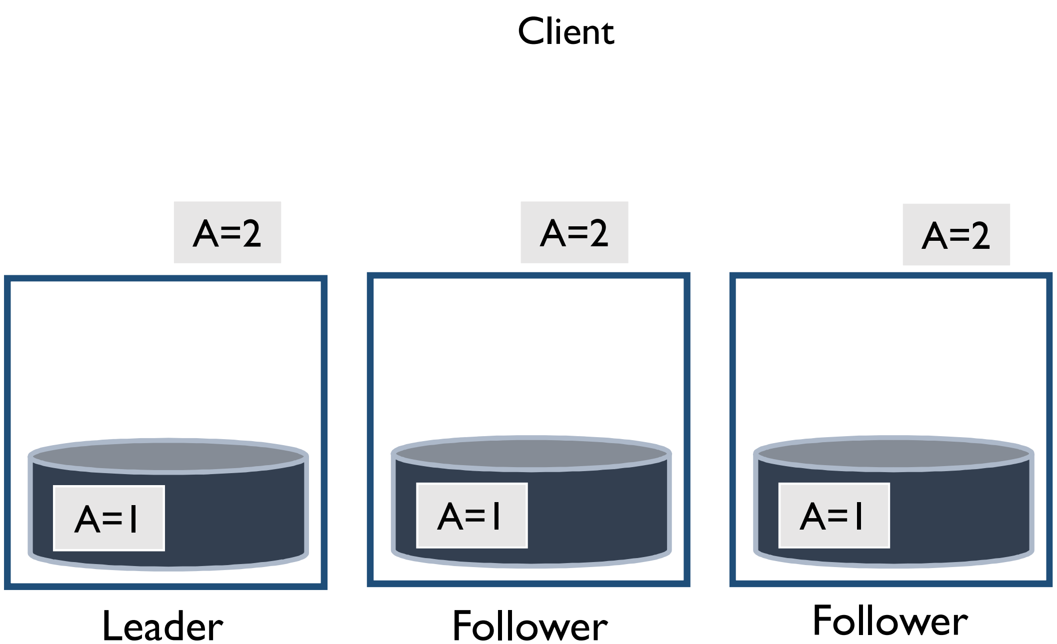
Disk-Durable Protocols

Update



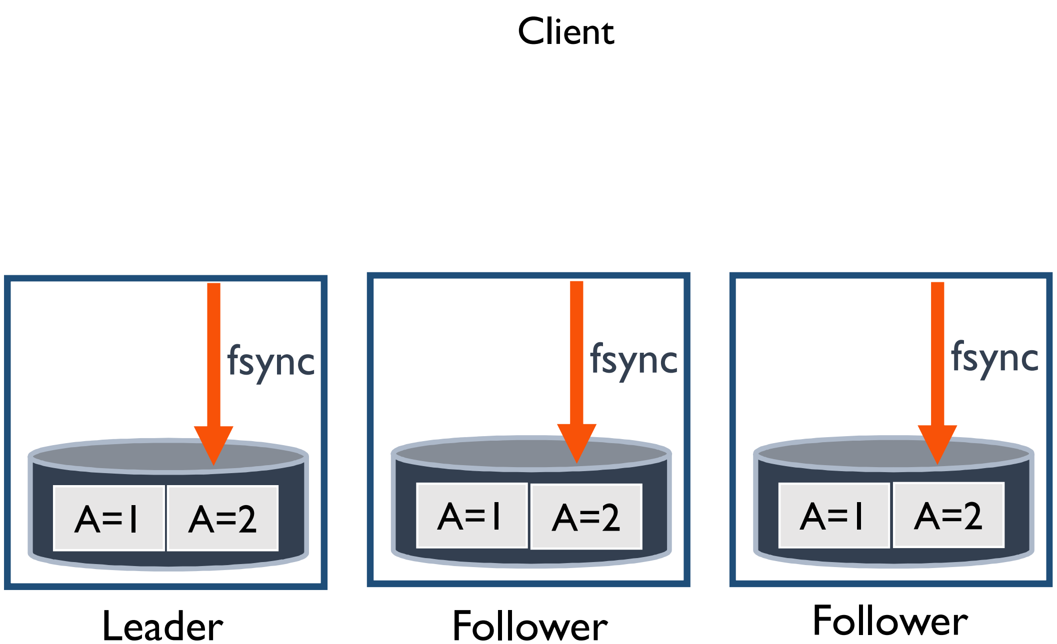
Disk-Durable Protocols

Update

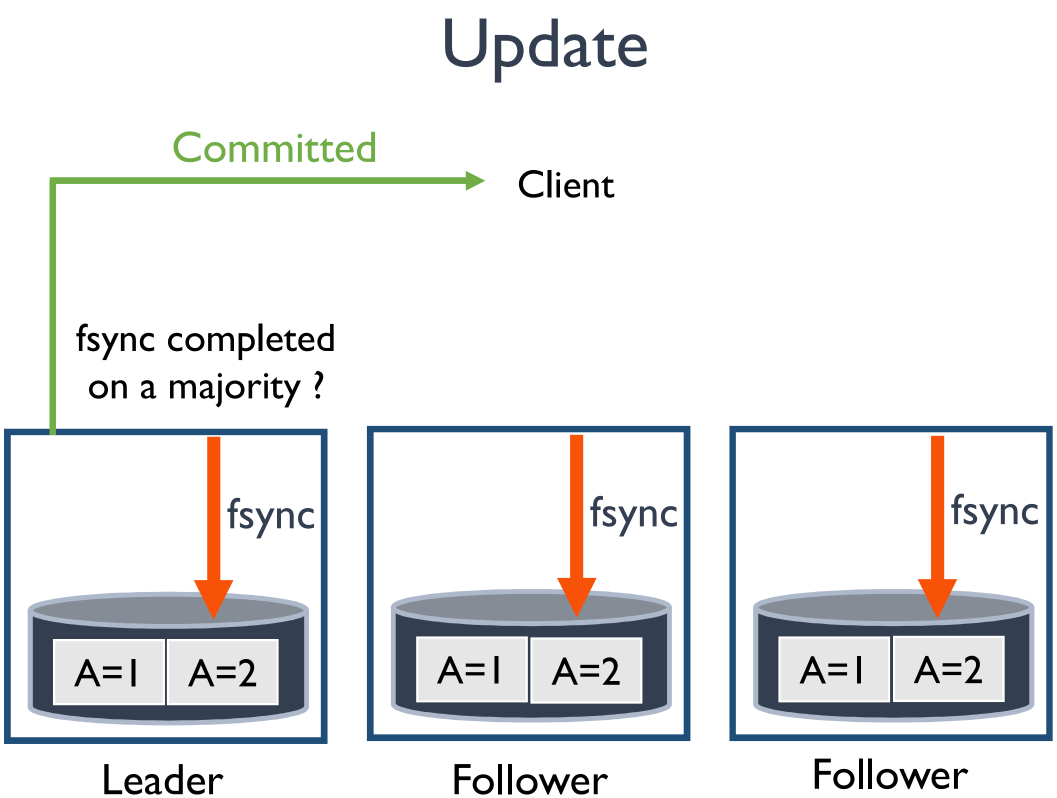


Disk-Durable Protocols

Update

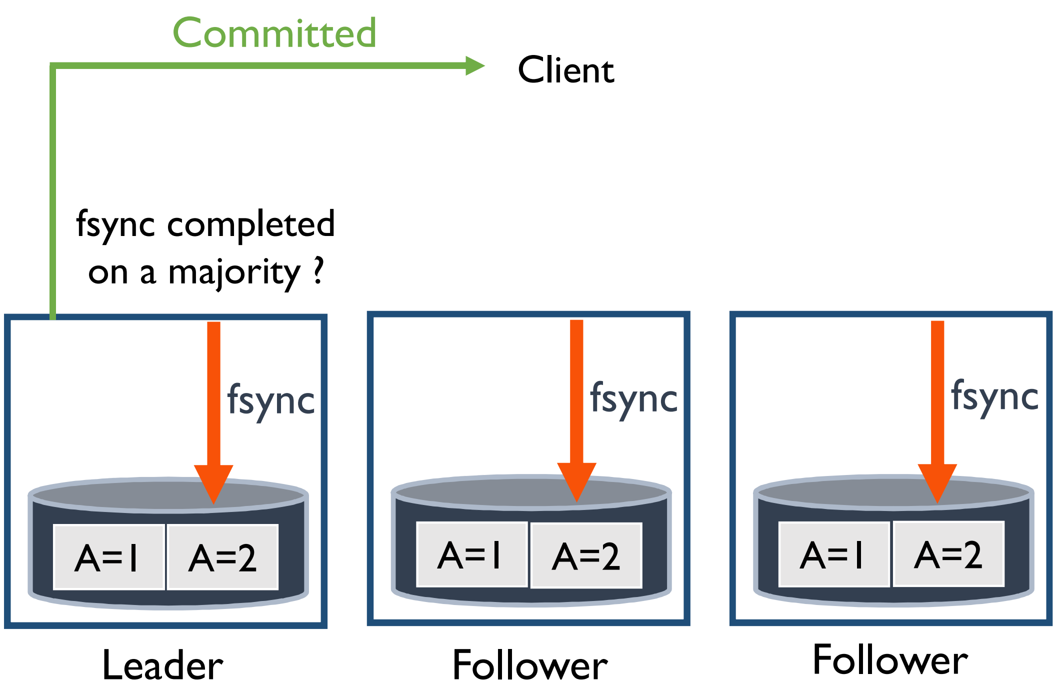


Disk-Durable Protocols



Disk-Durable Protocols

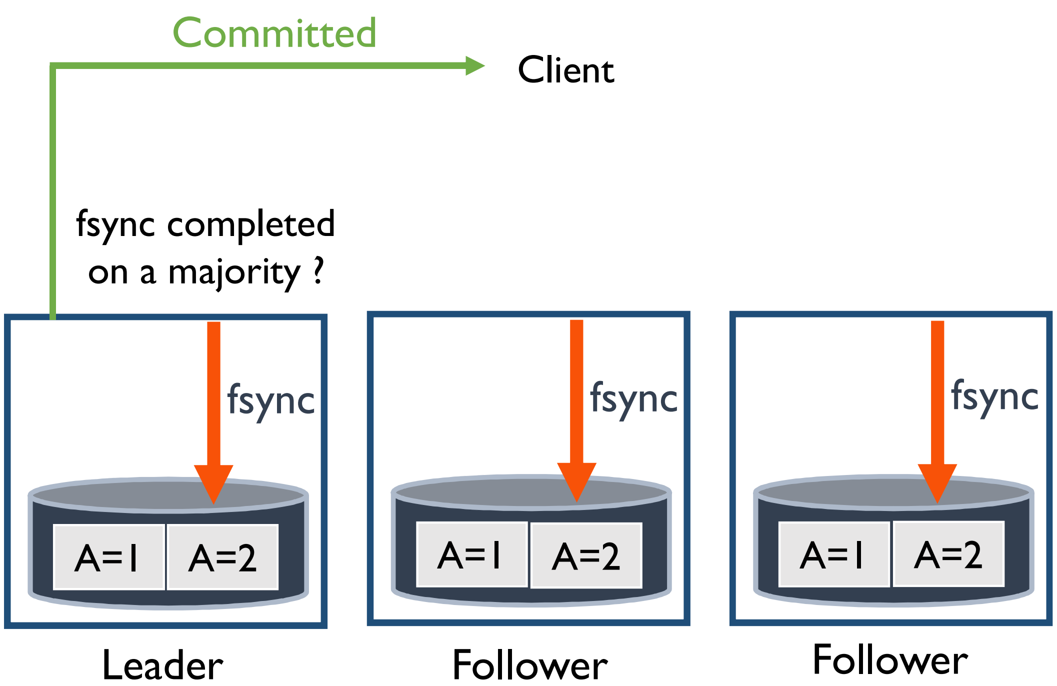
Update



Recovery

Disk-Durable Protocols

Update

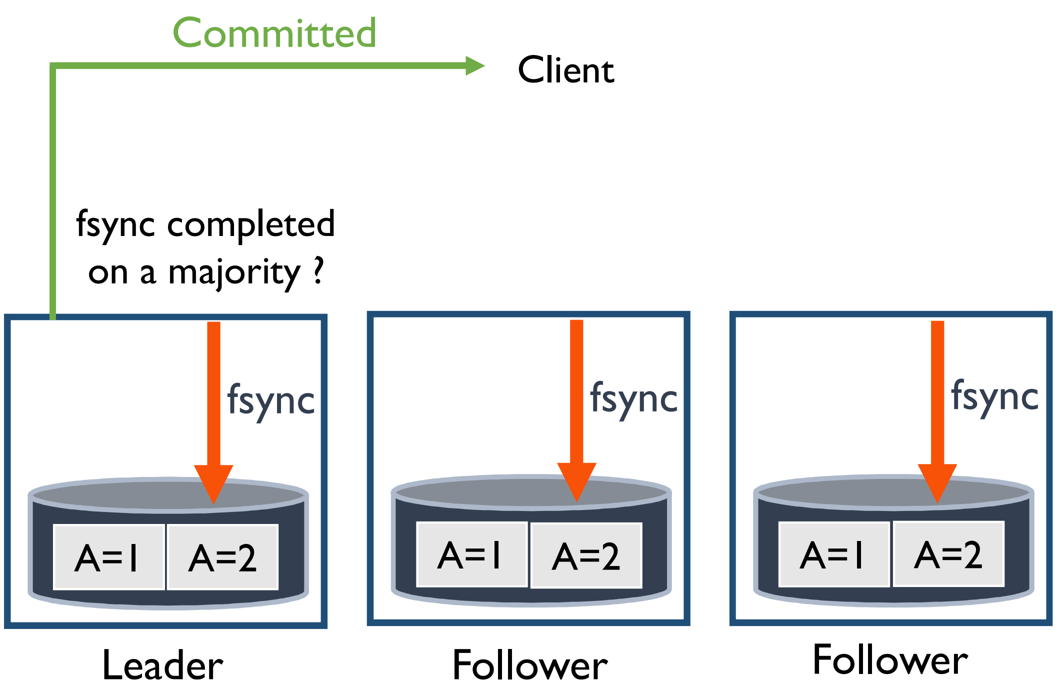


Recovery

if ack'd anyone, data on disk – safe

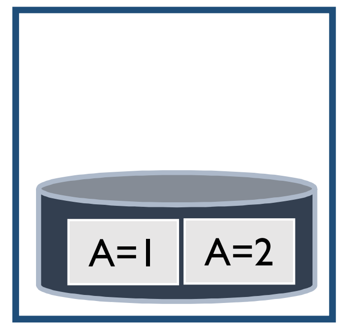
Disk-Durable Protocols

Update



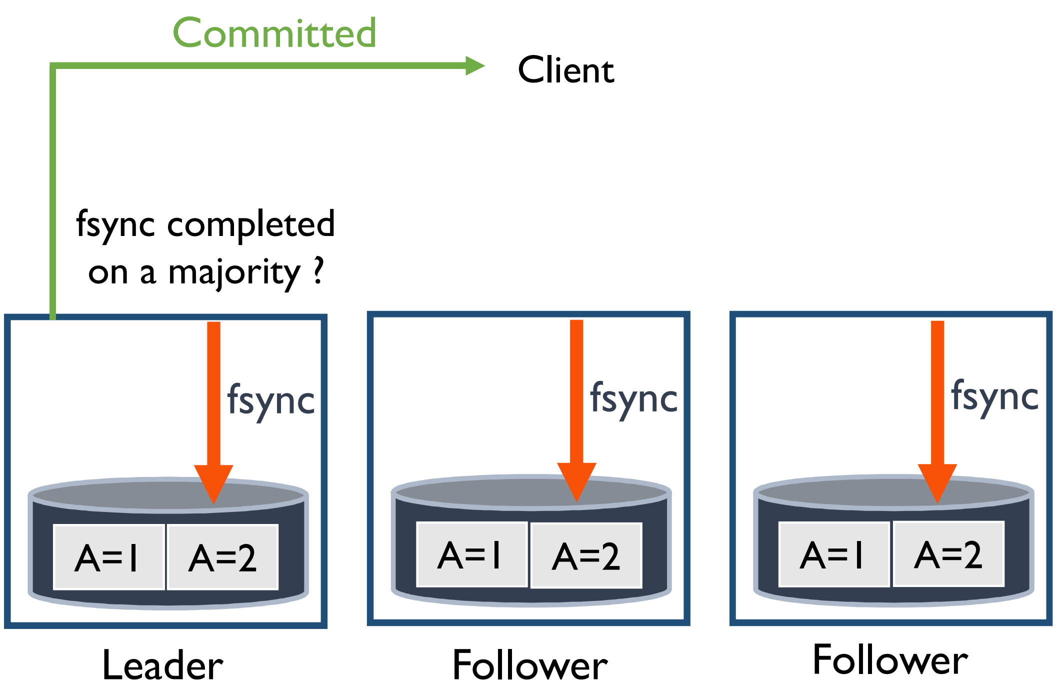
Recovery

if ack'd anyone, data on disk – safe



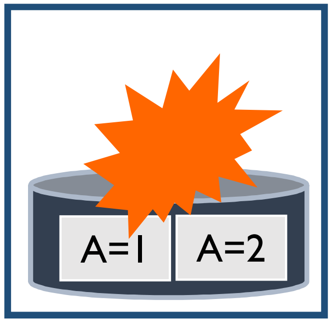
Disk-Durable Protocols

Update



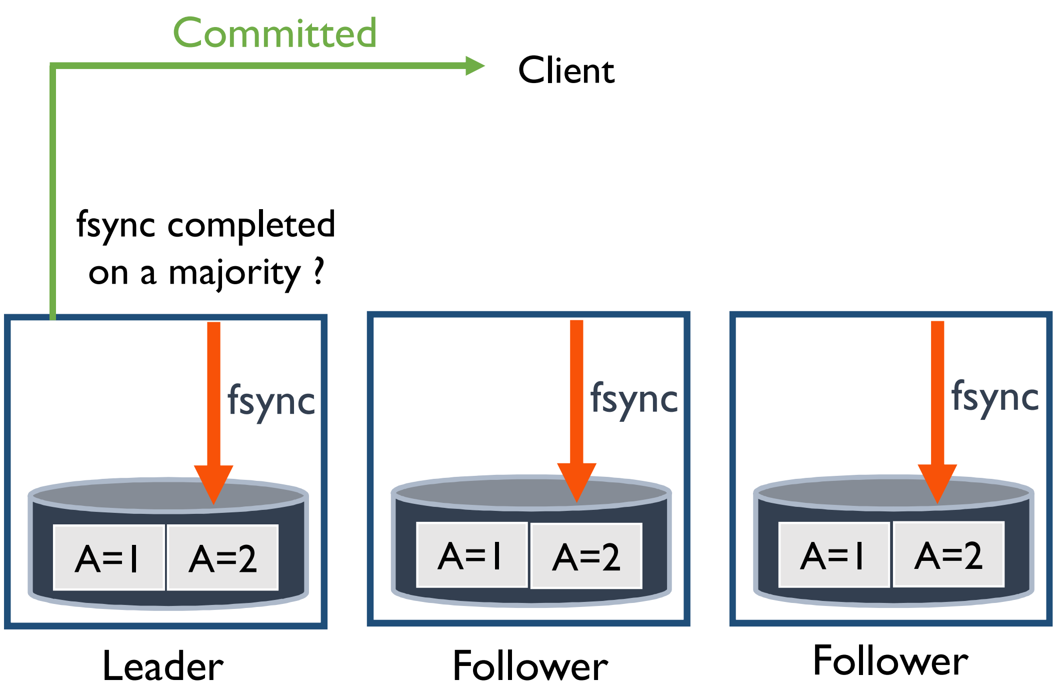
Recovery

if ack'd anyone, data on disk – safe



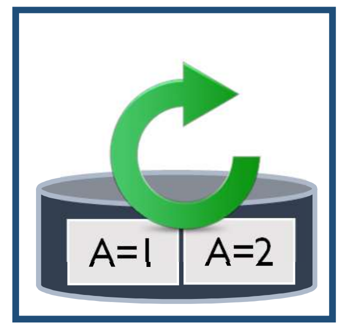
Disk-Durable Protocols

Update



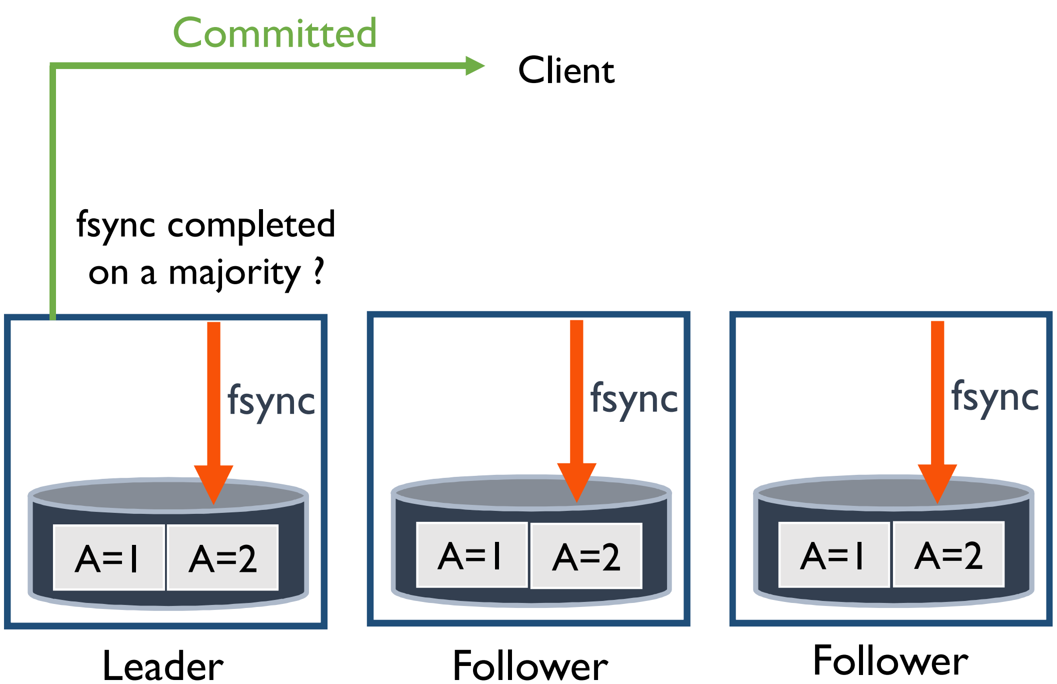
Recovery

if ack'd anyone, data on disk – safe



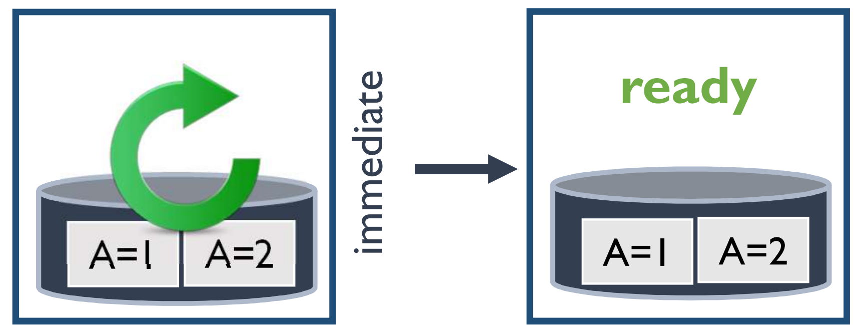
Disk-Durable Protocols

Update



Recovery

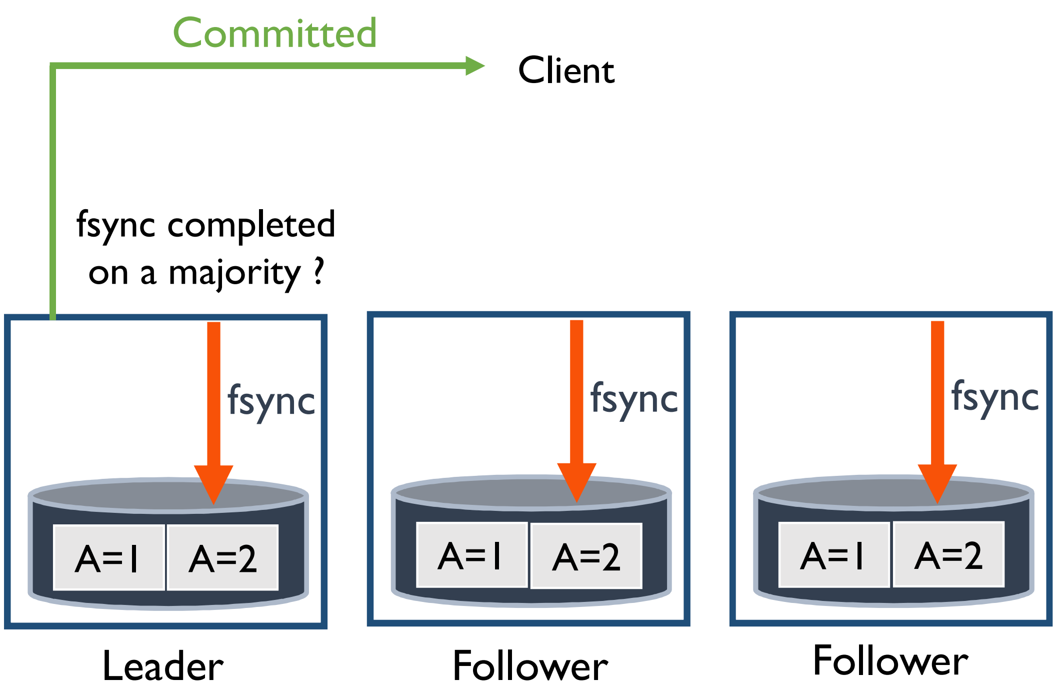
if ack'd anyone, data on disk – safe



recovery: just read from local disk

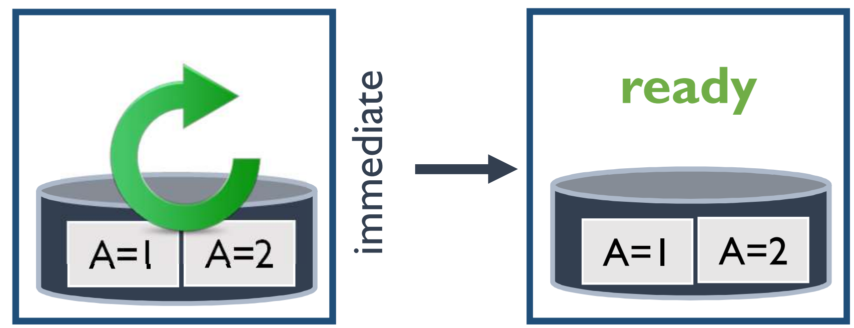
Disk-Durable Protocols

Update

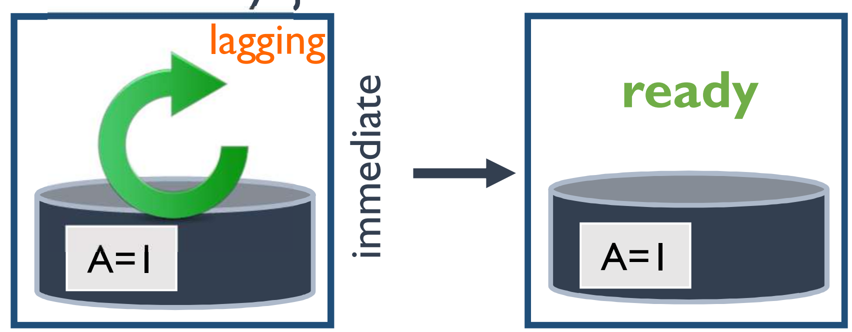


Recovery

if ack'd anyone, data on disk – safe

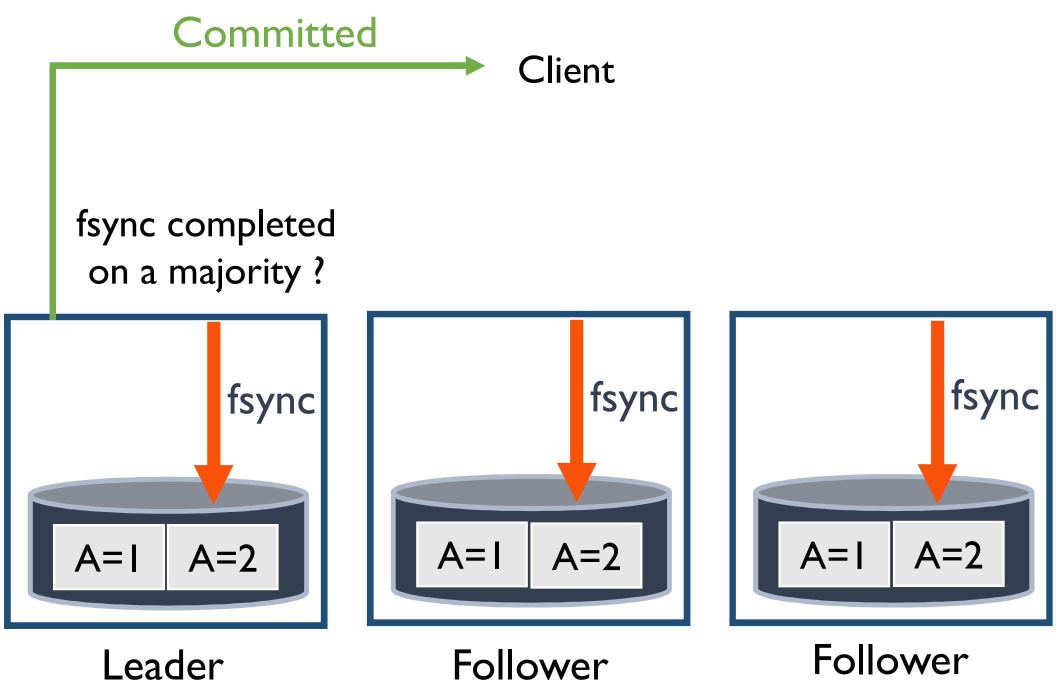


recovery: just read from local disk



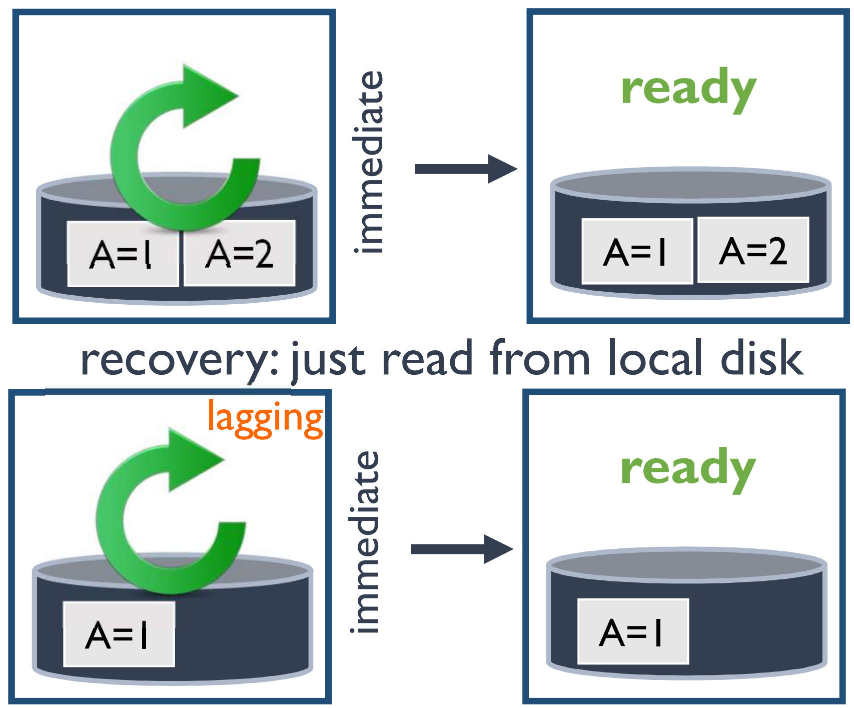
Disk-Durable Protocols

Update



Recovery

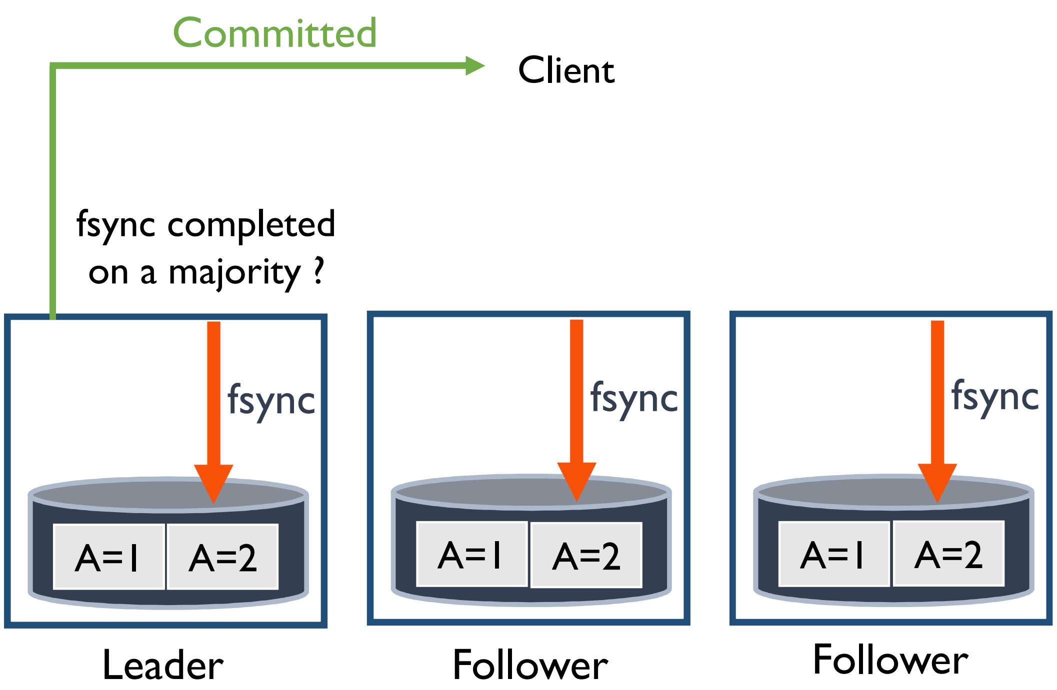
if ack'd anyone, data on disk – safe



Safe and available

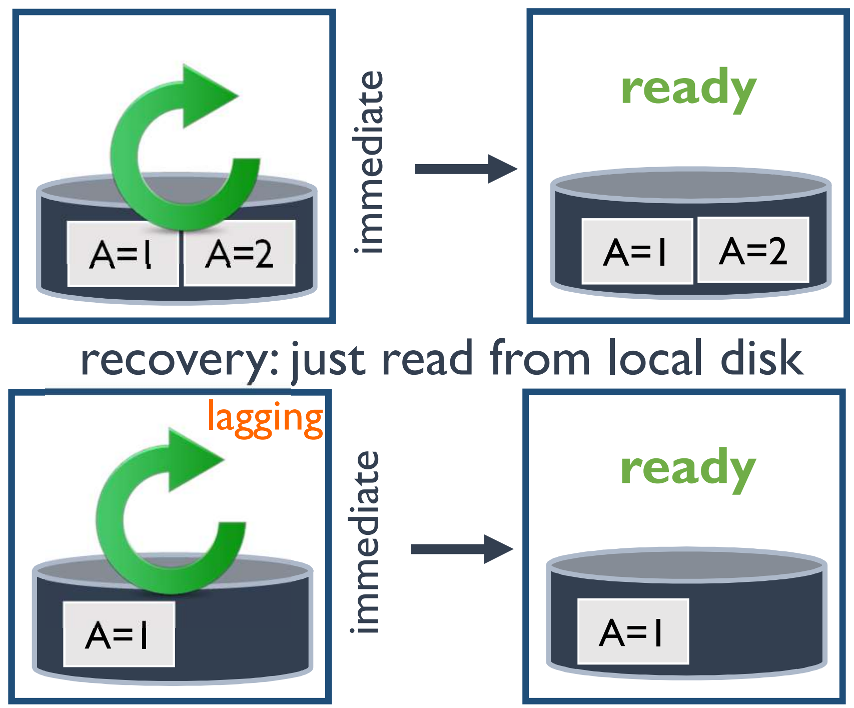
Disk-Durable Protocols

Update



Recovery

if ack'd anyone, data on disk – safe

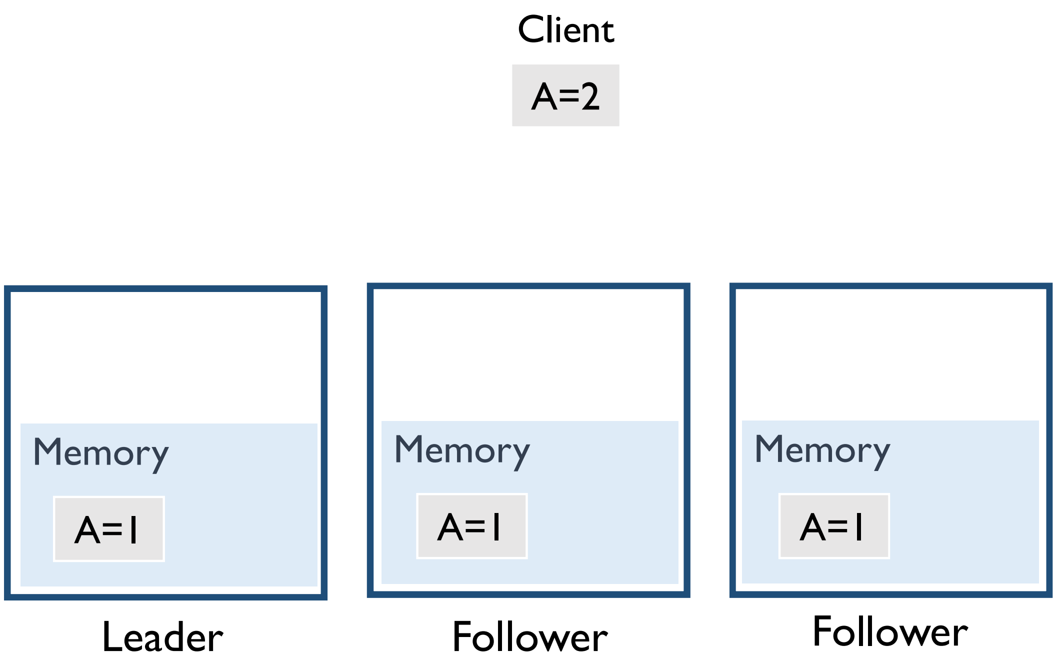


Safe and available

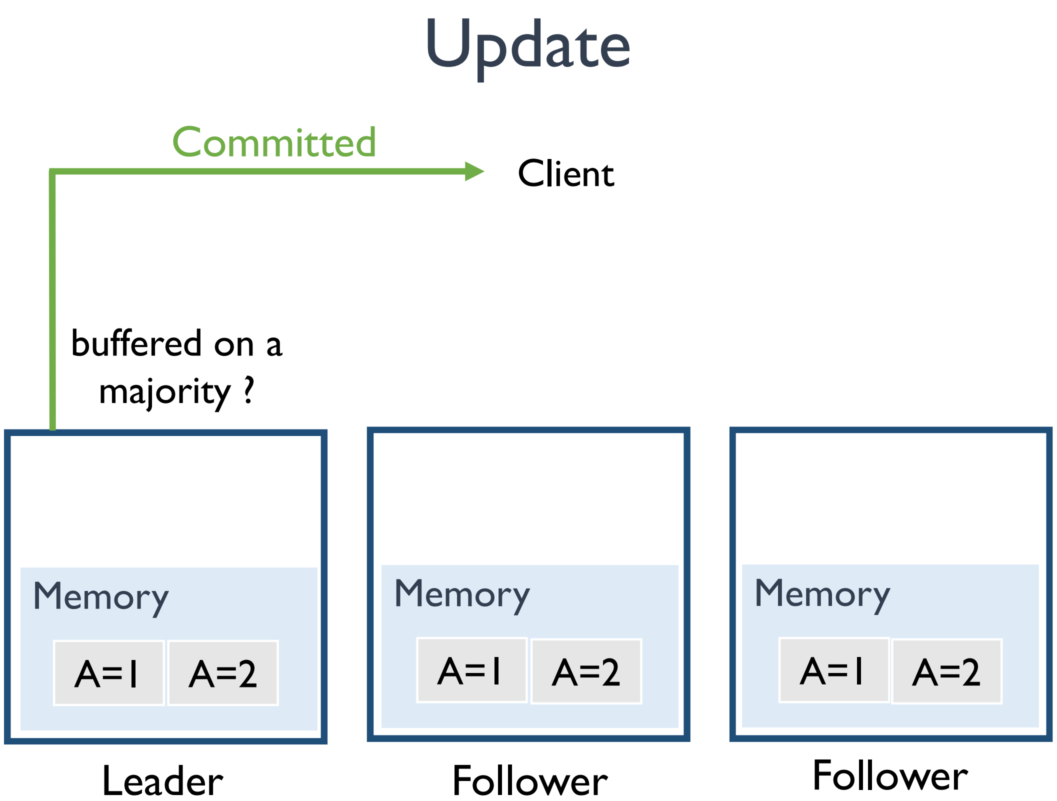
But poor performance due to fsync – 50x on HDDs, 2.5x on SSDs

Memory-Durable Protocols (Oblivious Recovery)

Update

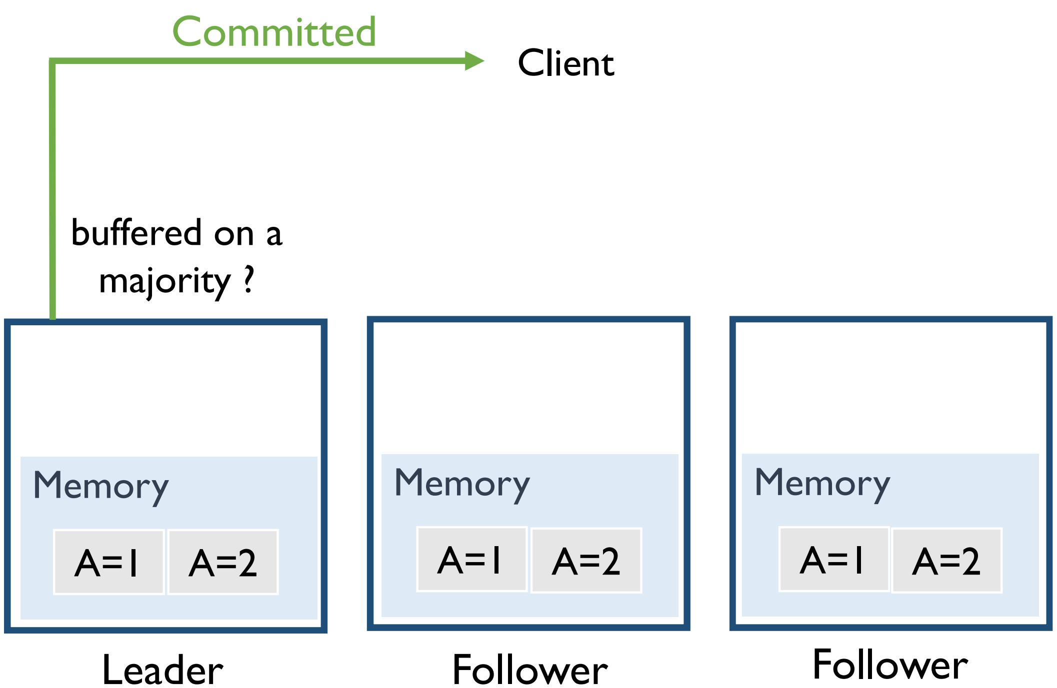


Memory-Durable Protocols (Oblivious Recovery)



Memory-Durable Protocols (Oblivious Recovery)

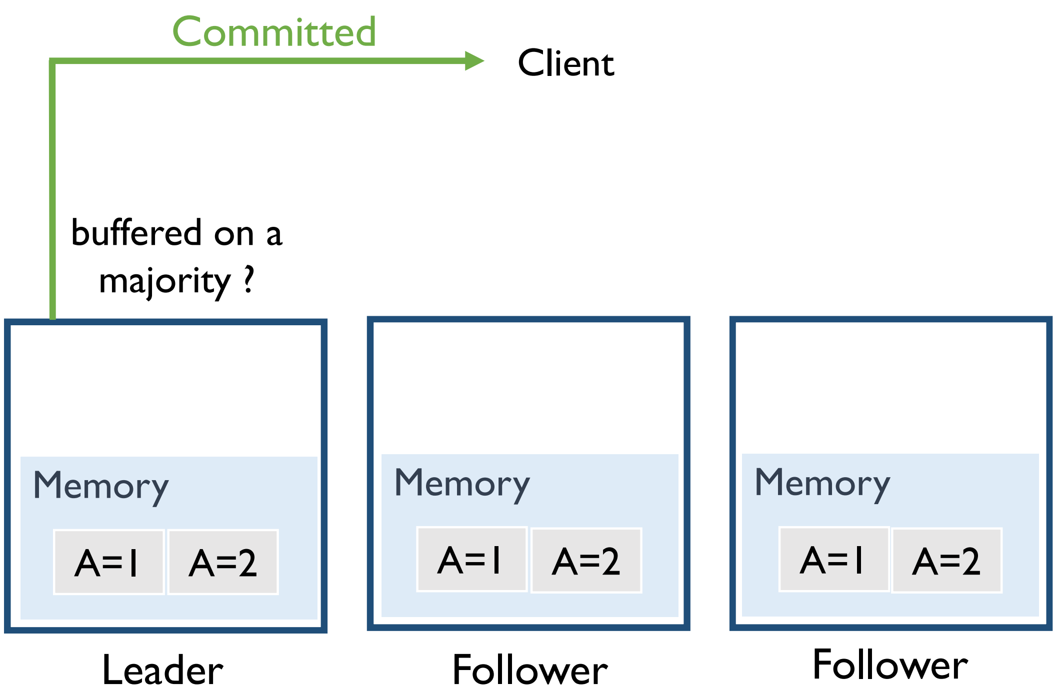
Update



Recovery

Memory-Durable Protocols (Oblivious Recovery)

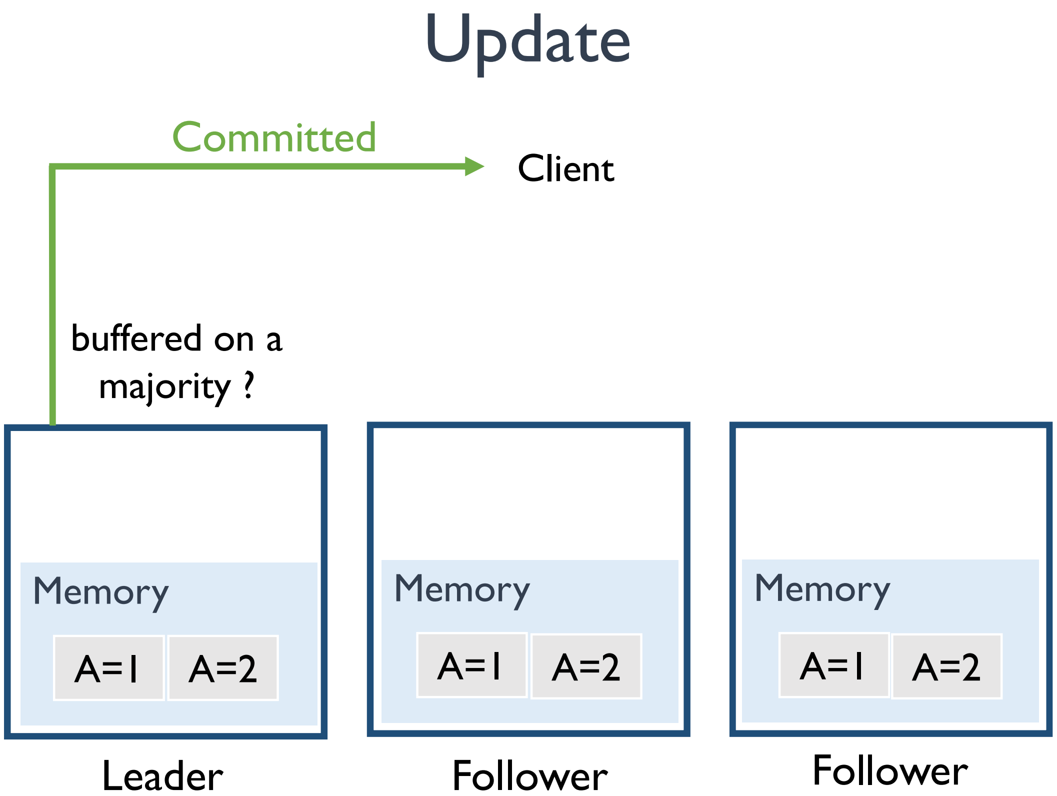
Update

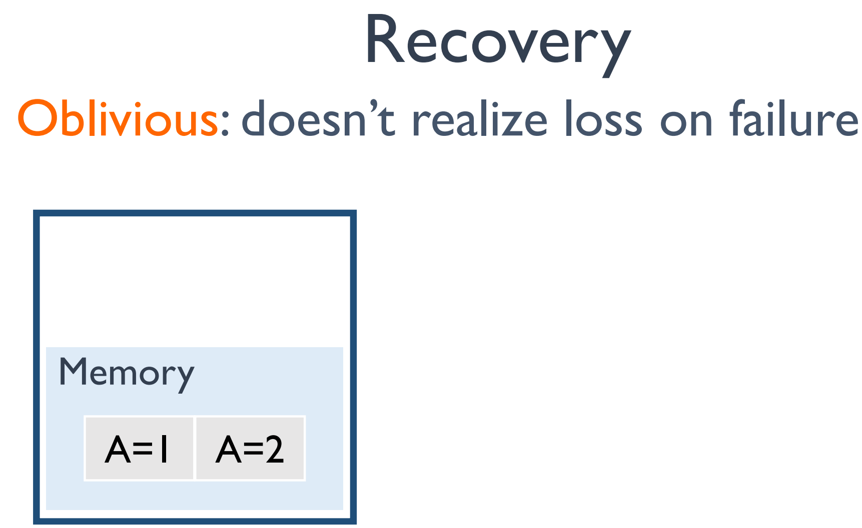


Recovery

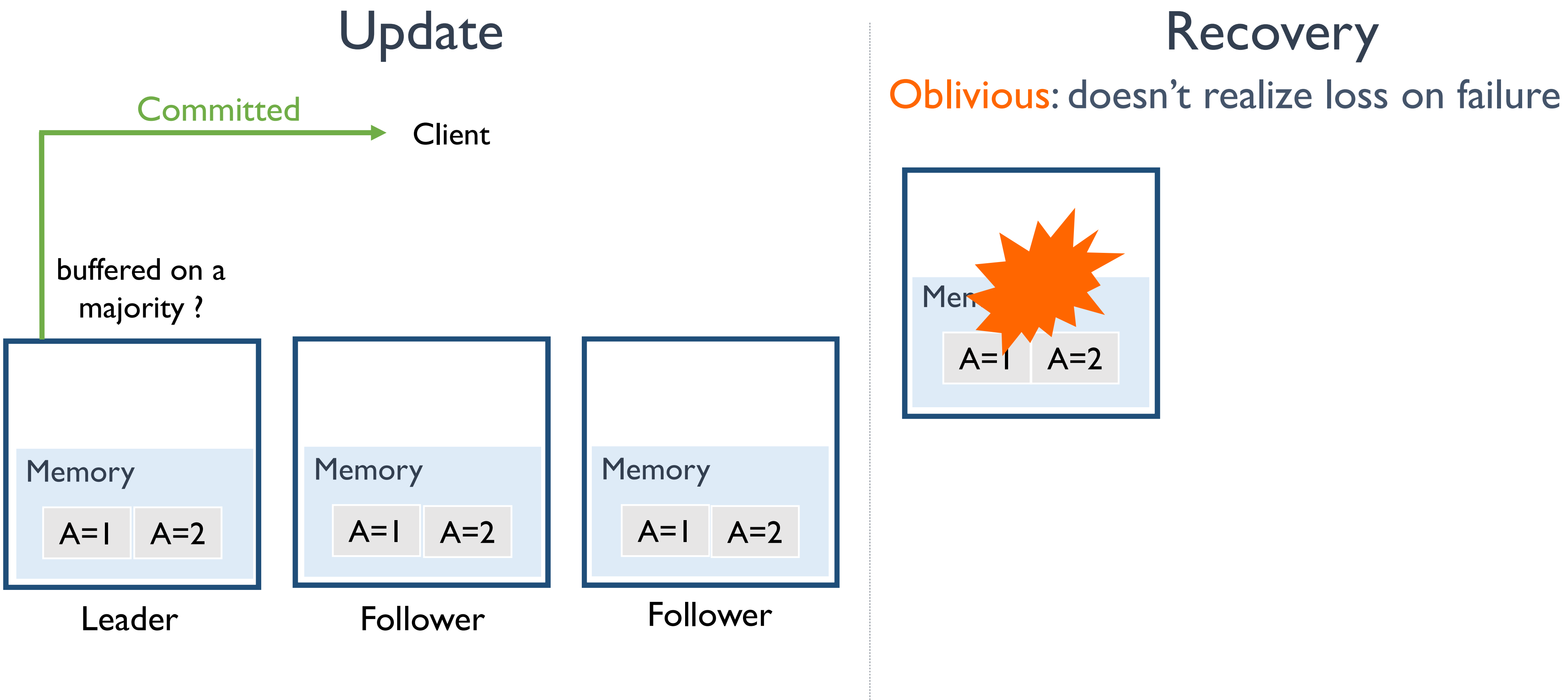
Oblivious: doesn't realize loss on failure

Memory-Durable Protocols (Oblivious Recovery)



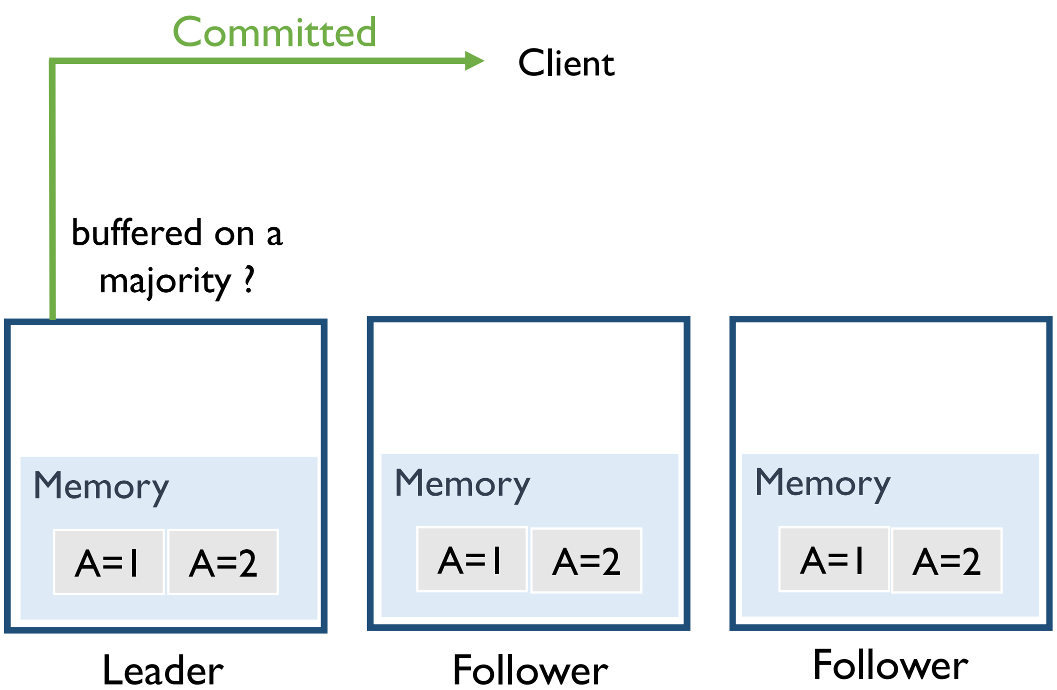


Memory-Durable Protocols (Oblivious Recovery)



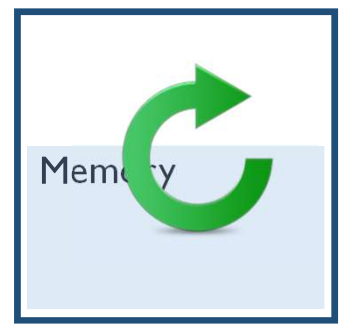
Memory-Durable Protocols (Oblivious Recovery)

Update

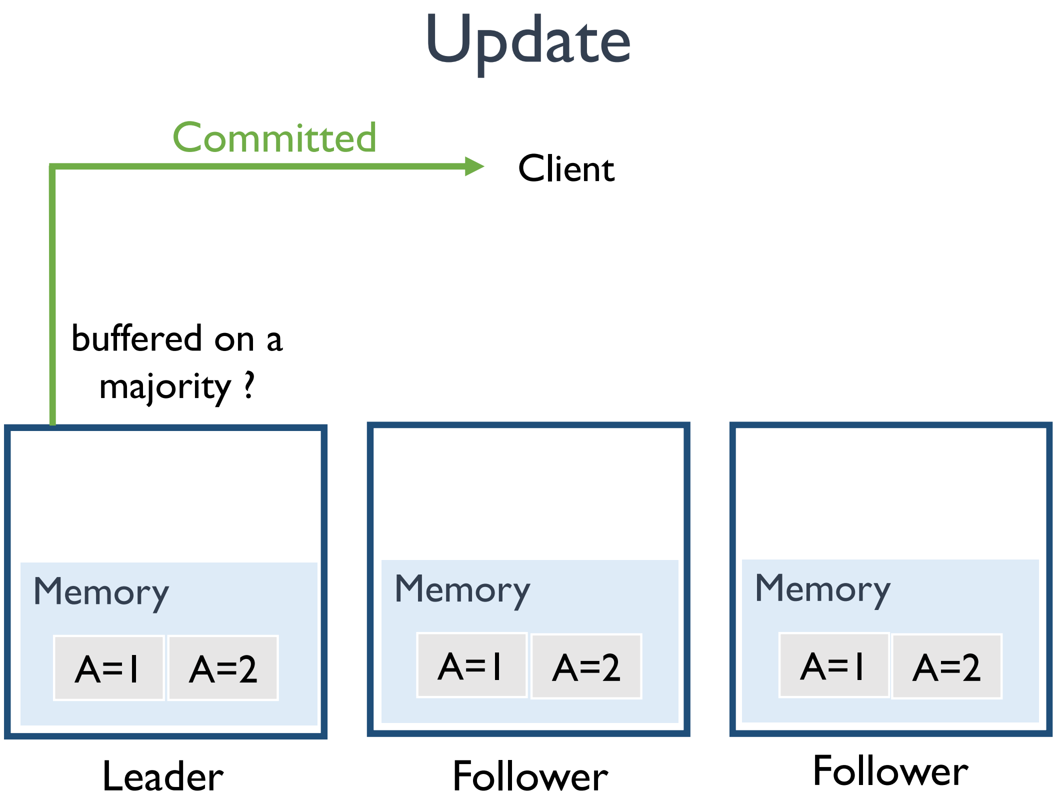


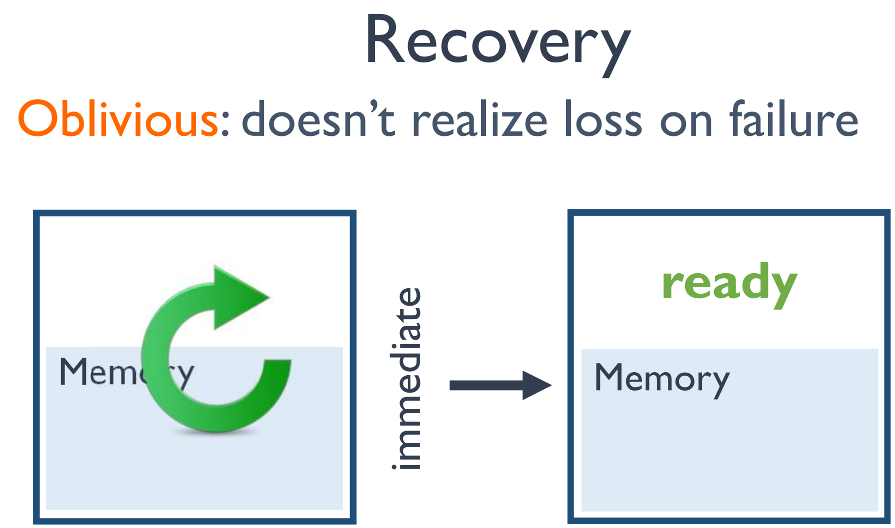
Recovery

Oblivious: doesn't realize loss on failure



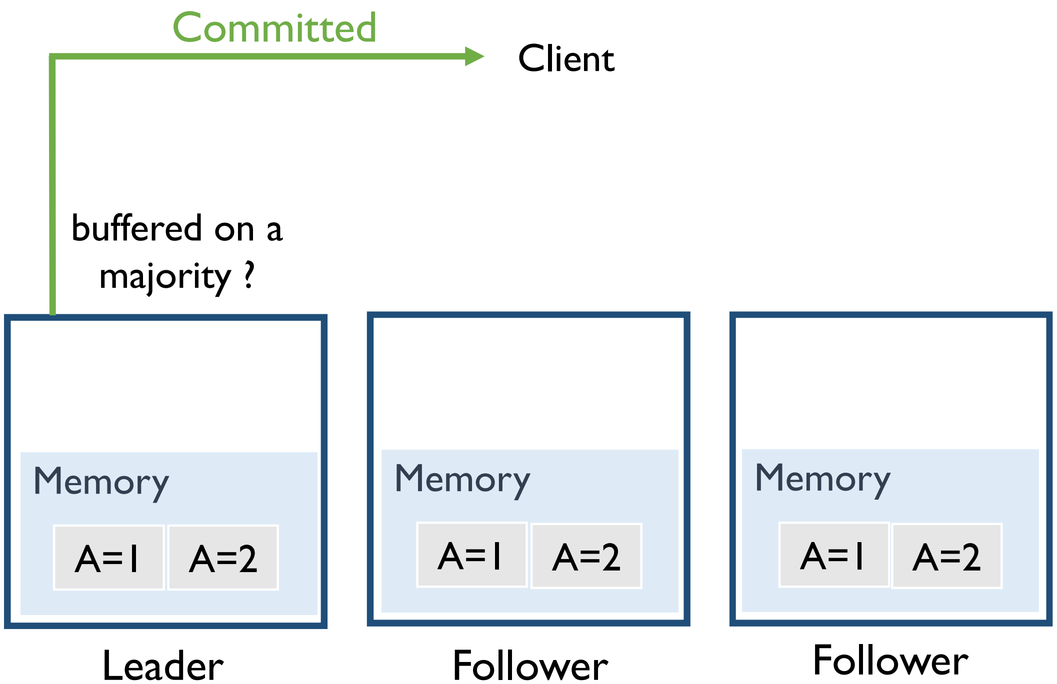
Memory-Durable Protocols (Oblivious Recovery)





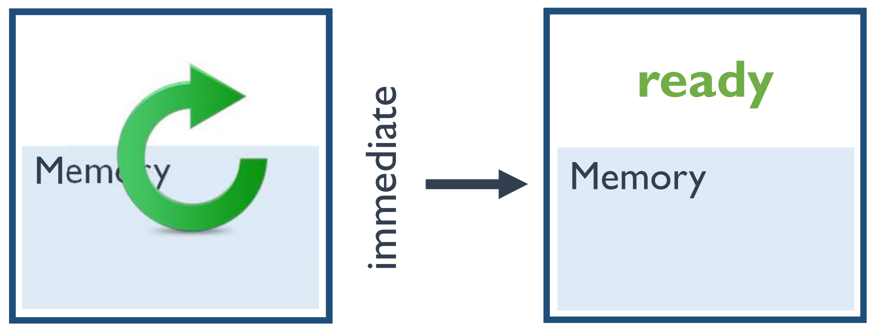
Memory-Durable Protocols (Oblivious Recovery)

Update



Recovery

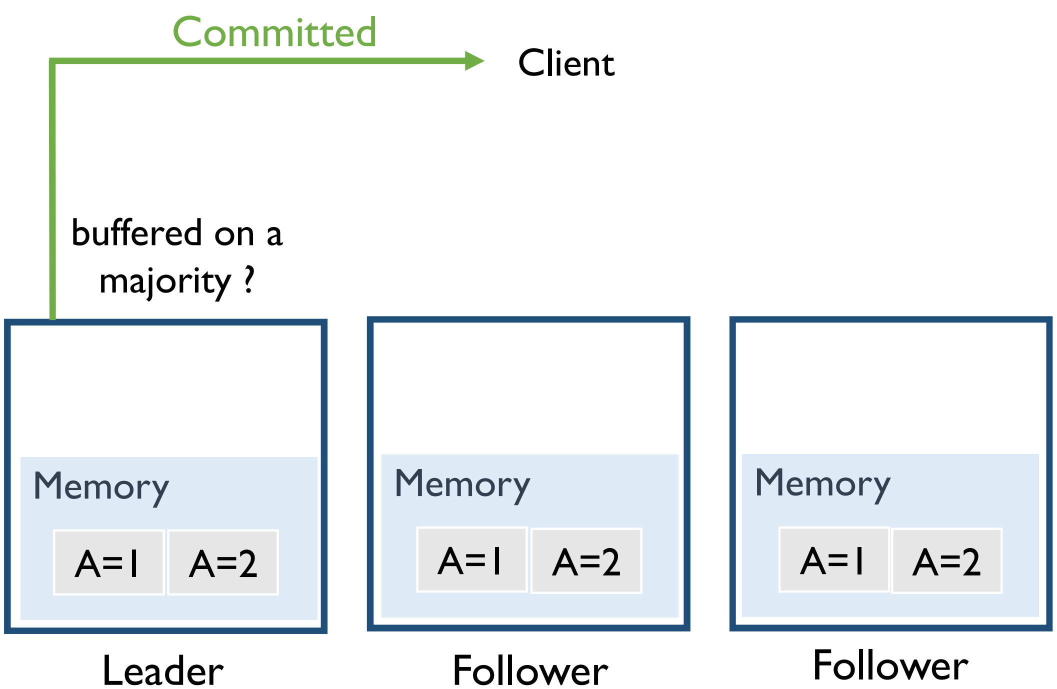
Oblivious: doesn't realize loss on failure



e.g., ZooKeeper with *forceSync* = *false*
practitioners do use this config!

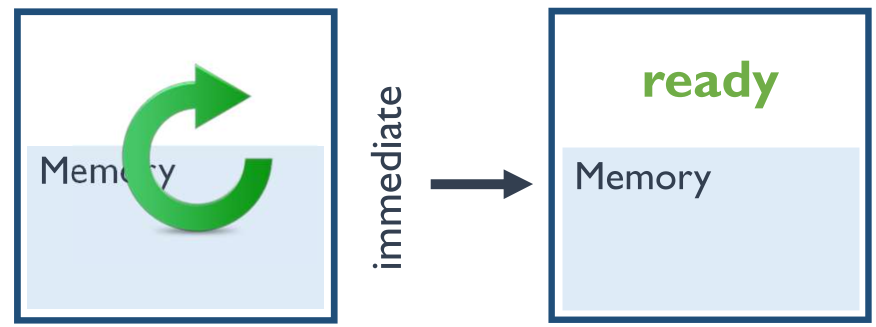
Memory-Durable Protocols (Oblivious Recovery)

Update



Recovery

Oblivious: doesn't realize loss on failure

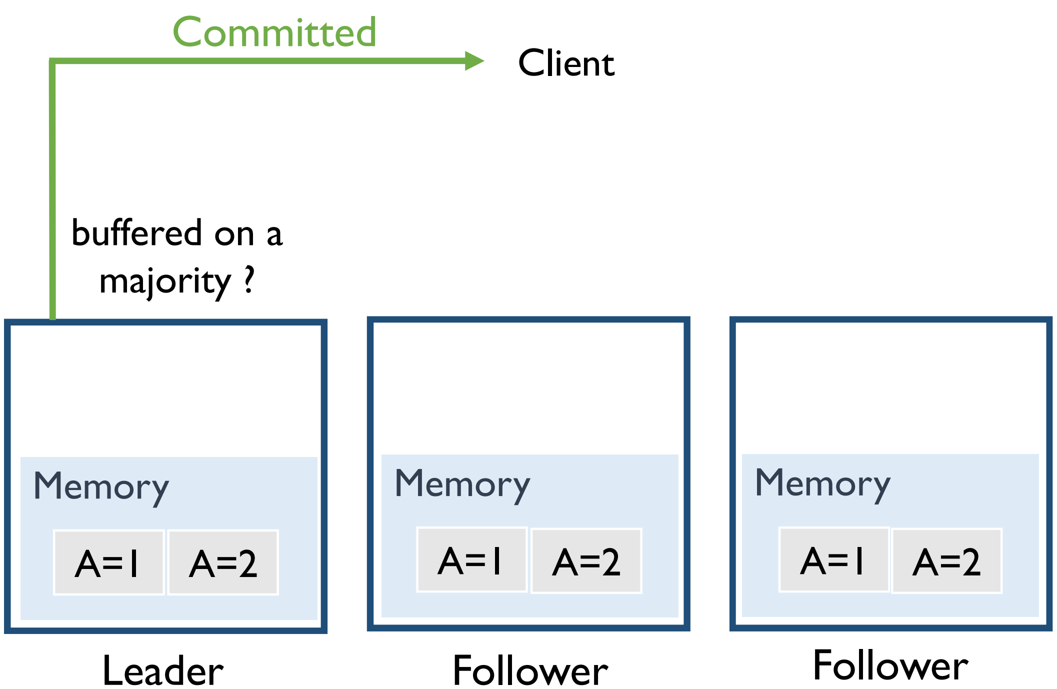


e.g., ZooKeeper with *forceSync* = *false*
practitioners do use this config!

Performant

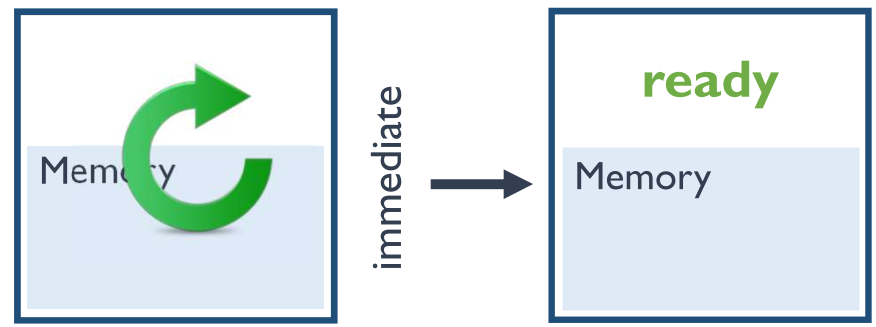
Memory-Durable Protocols (Oblivious Recovery)

Update



Recovery

Oblivious: doesn't realize loss on failure



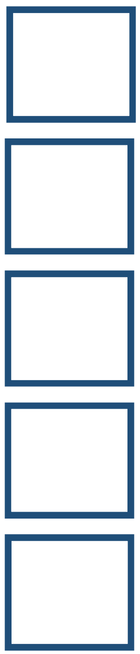
e.g., ZooKeeper with *forceSync* = *false*
practitioners do use this config!

Performant

But can lead to data loss

Data Loss Example in Oblivious Approach

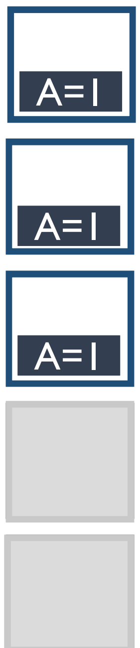
Data Loss Example in Oblivious Approach



Data Loss Example in Oblivious Approach

A=1 committed

Data Loss Example in Oblivious Approach



A=1 committed
two nodes slow or failed

Data Loss Example in Oblivious Approach

A=1 committed
two nodes slow or failed

Data Loss Example in Oblivious Approach

A=1 committed
two nodes slow or failed

crashes

Data Loss Example in Oblivious Approach

A=1 committed
two nodes slow or failed

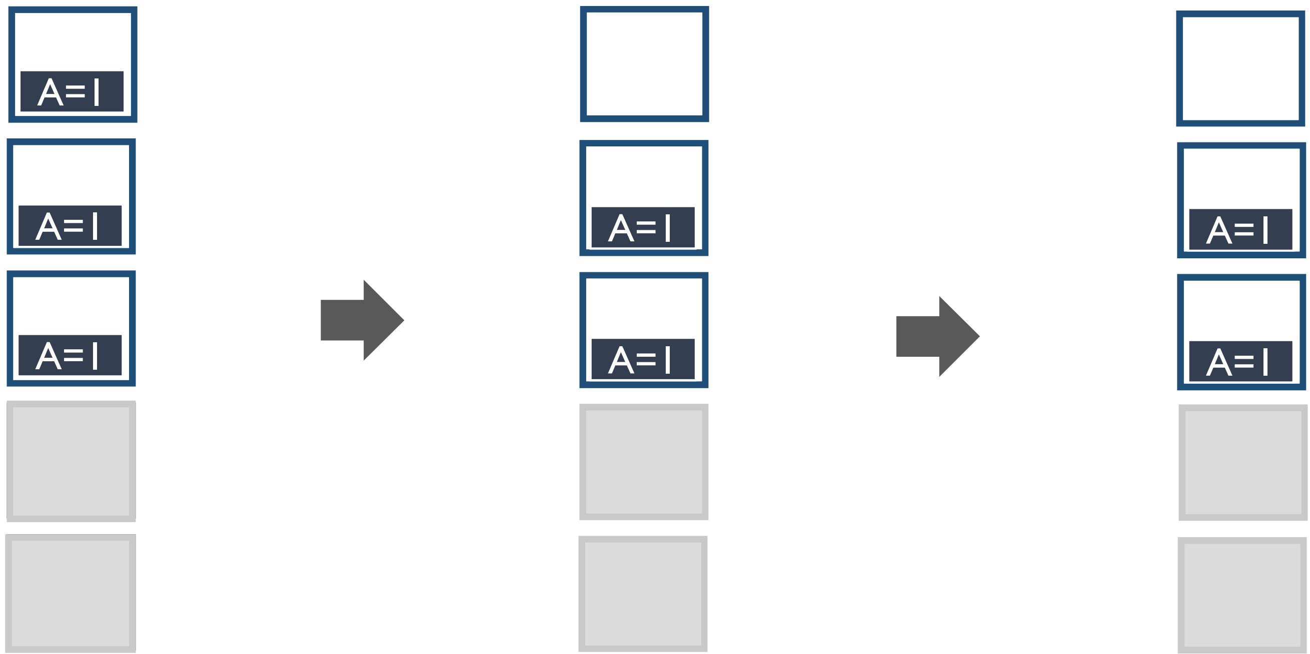
crashes, recovers

Data Loss Example in Oblivious Approach

A=1 committed
two nodes slow or failed

crashes, recovers
loses its data
but oblivious:
immediately joins

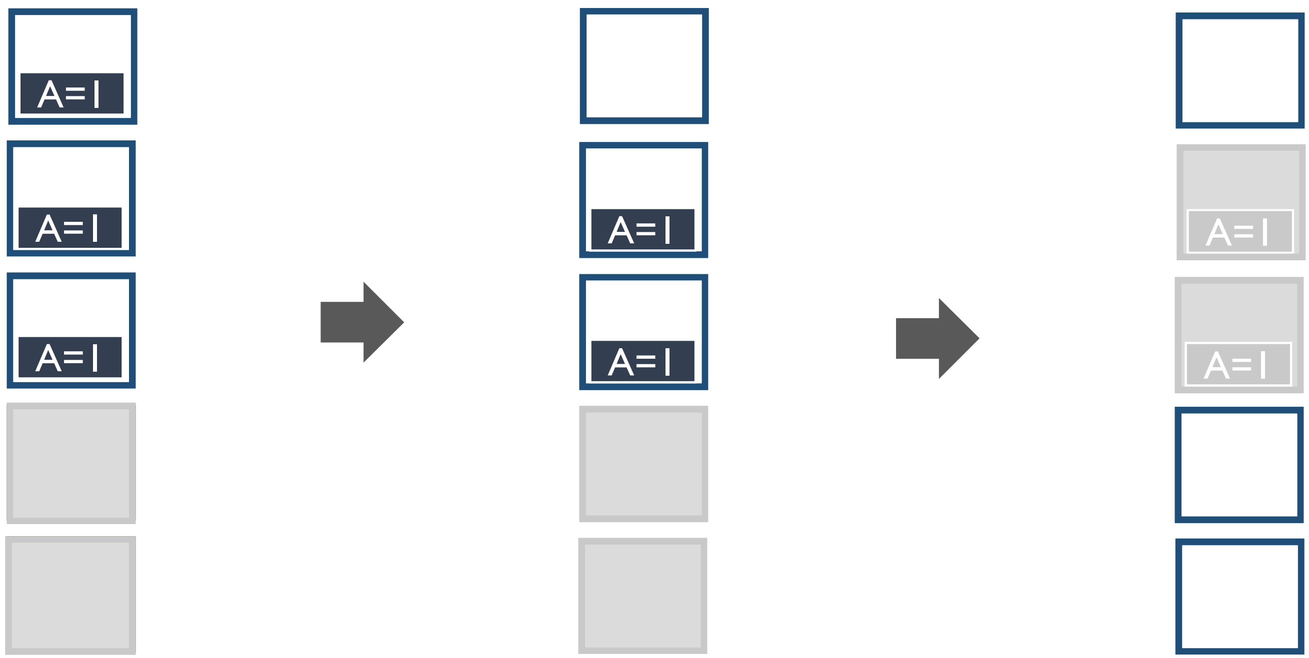
Data Loss Example in Oblivious Approach



A=1 committed
two nodes slow or failed

crashes, recovers
loses its data
but oblivious:
immediately joins

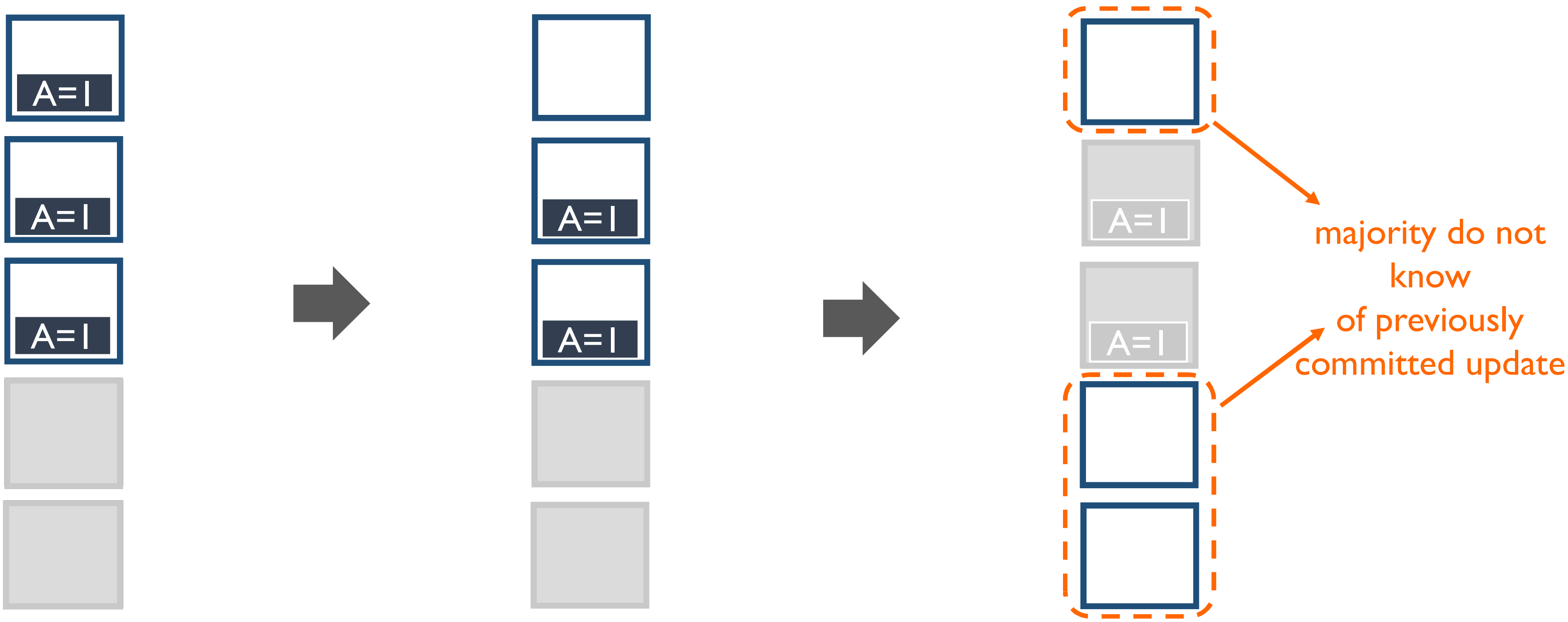
Data Loss Example in Oblivious Approach



A=1 committed
two nodes slow or failed

crashes, recovers
loses its data
but oblivious:
immediately joins

Data Loss Example in Oblivious Approach



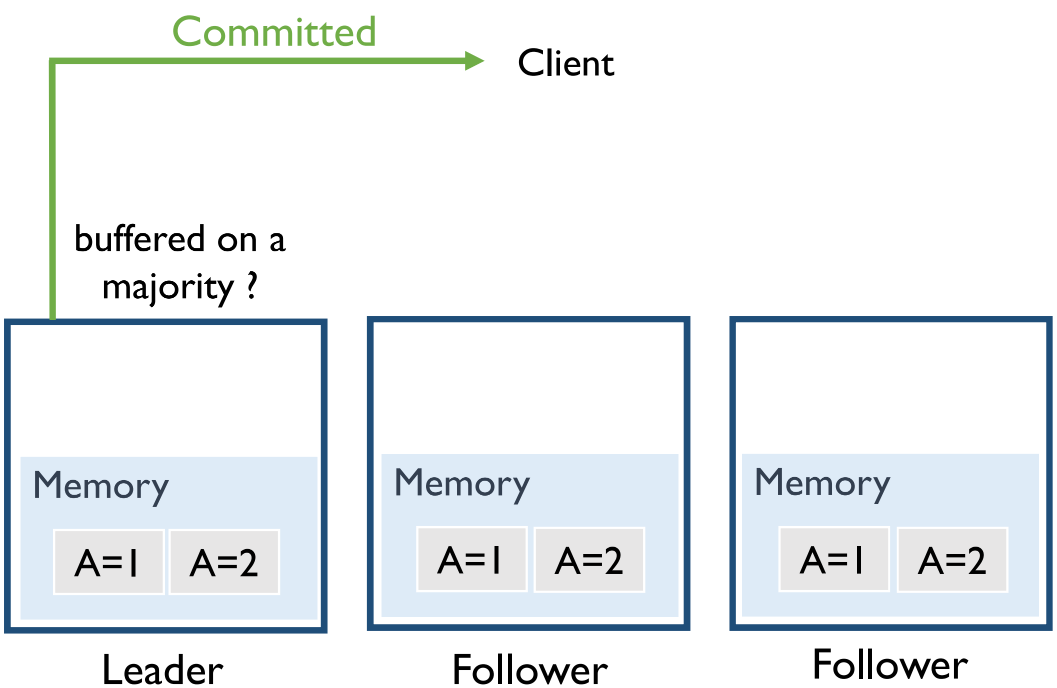
A=1 committed
two nodes slow or failed

crashes, recovers
loses its data
but oblivious:
immediately joins

lagging nodes along with recovered node form majority;
lose committed update

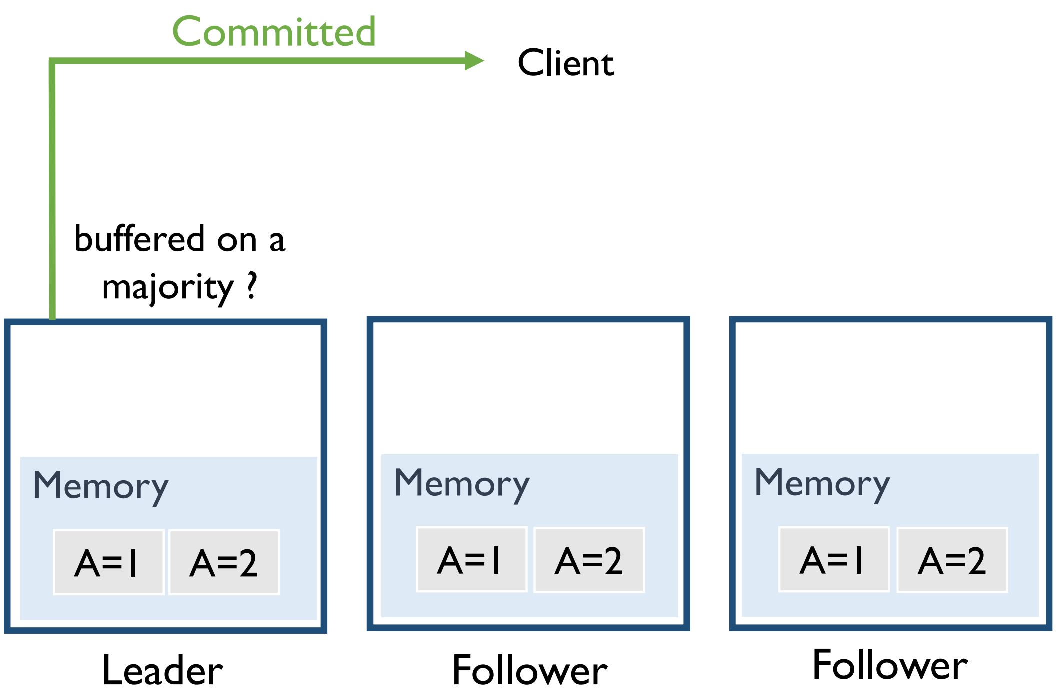
Memory-Durable Protocols (Loss-Aware Recovery)

Update



Memory-Durable Protocols (Loss-Aware Recovery)

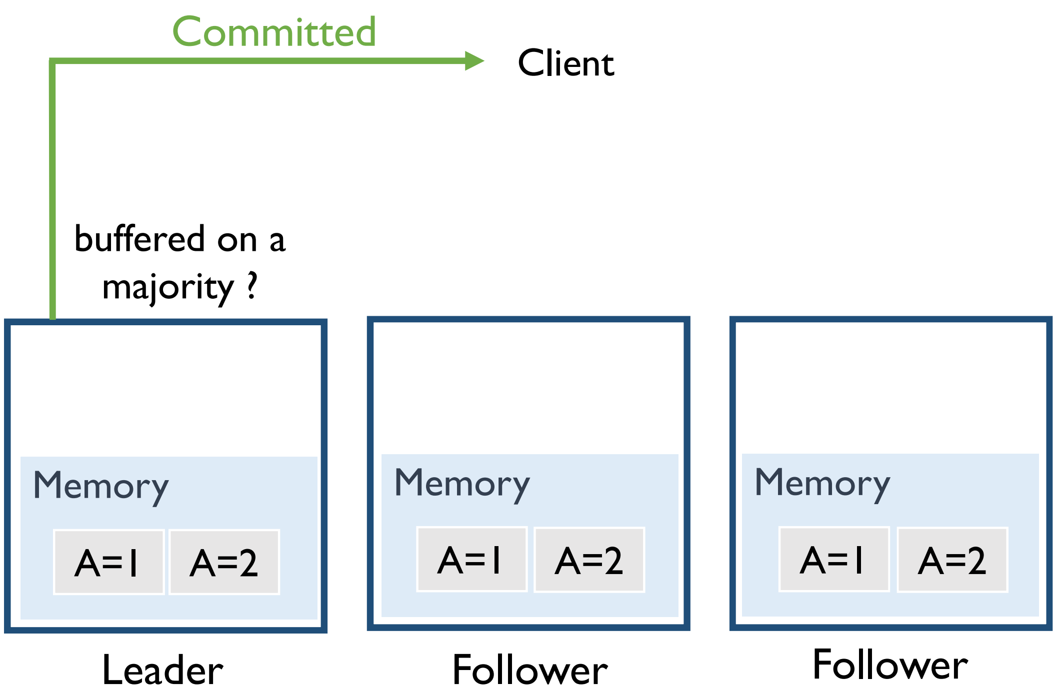
Update



Recovery

Memory-Durable Protocols (Loss-Aware Recovery)

Update

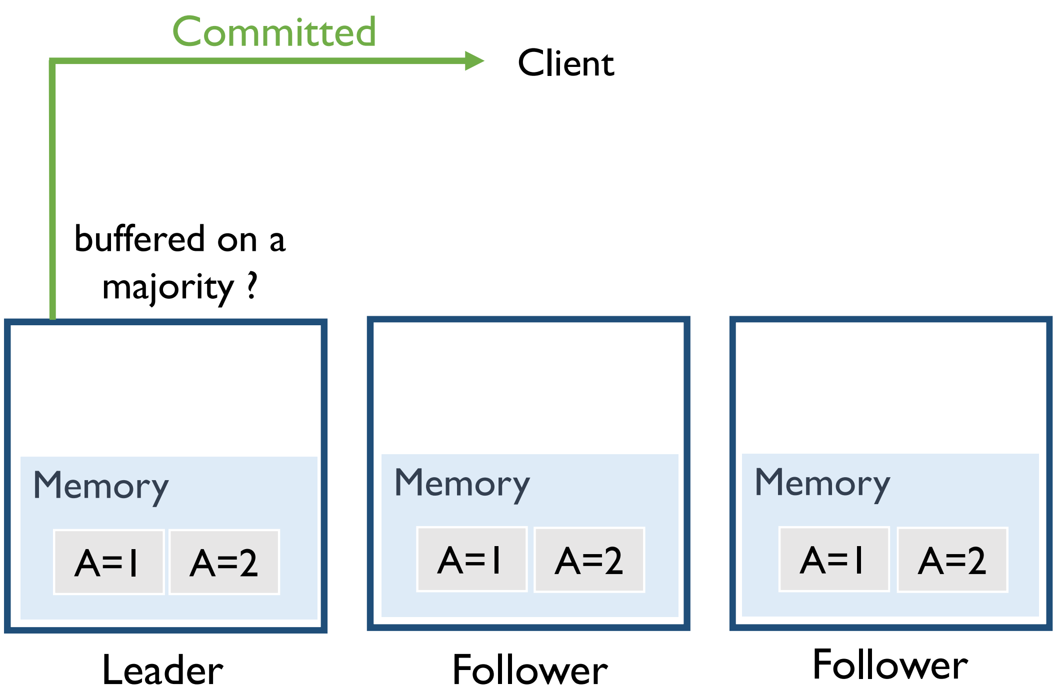


Recovery

Loss-aware: realizes loss, waits for majority

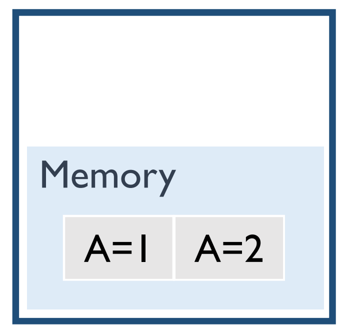
Memory-Durable Protocols (Loss-Aware Recovery)

Update



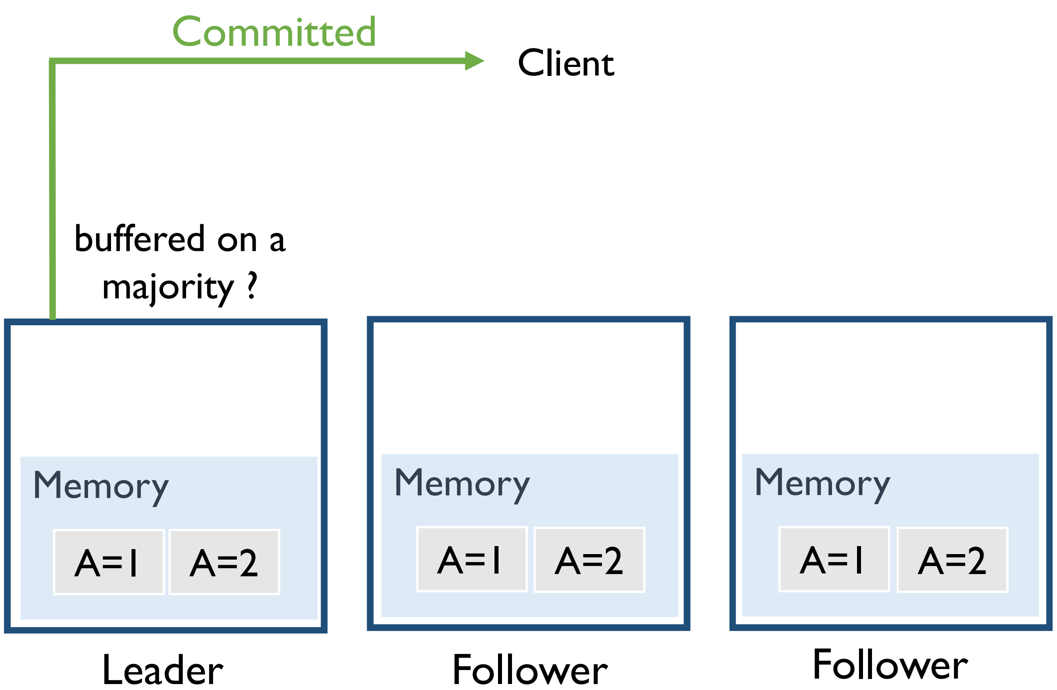
Recovery

Loss-aware: realizes loss, waits for majority



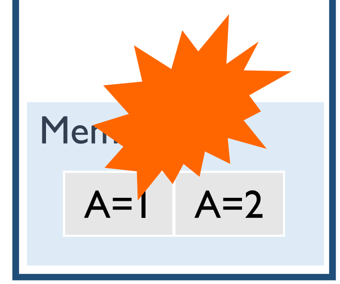
Memory-Durable Protocols (Loss-Aware Recovery)

Update



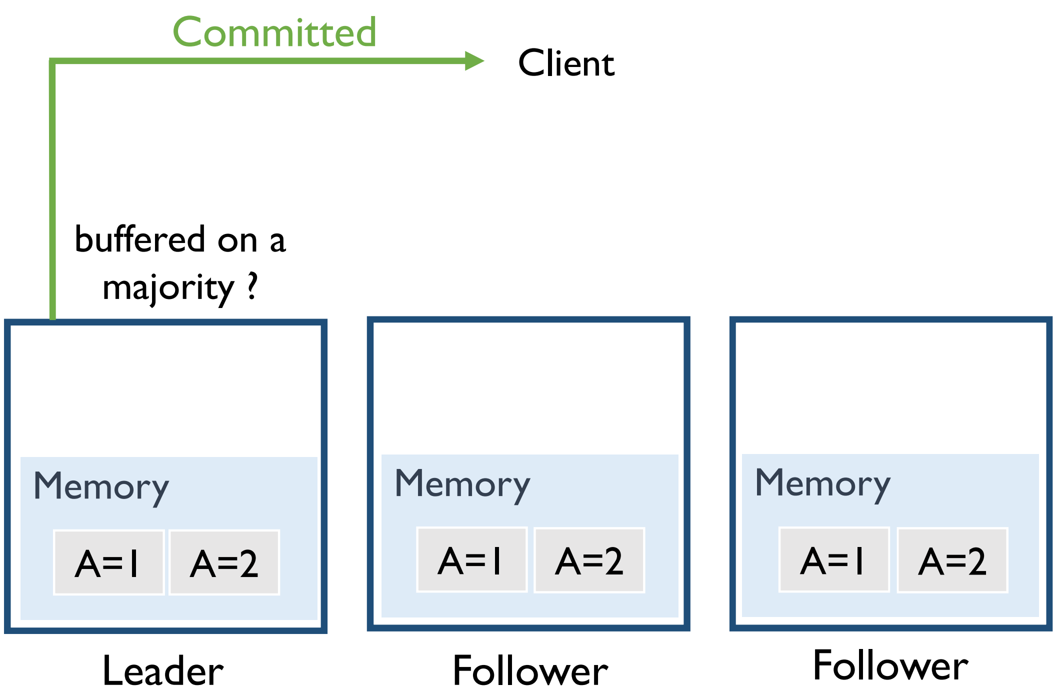
Recovery

Loss-aware: realizes loss, waits for majority



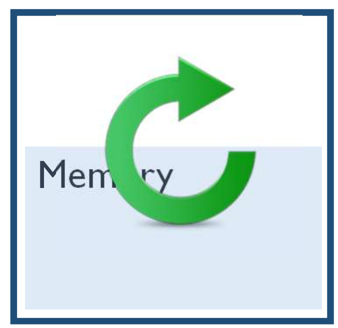
Memory-Durable Protocols (Loss-Aware Recovery)

Update



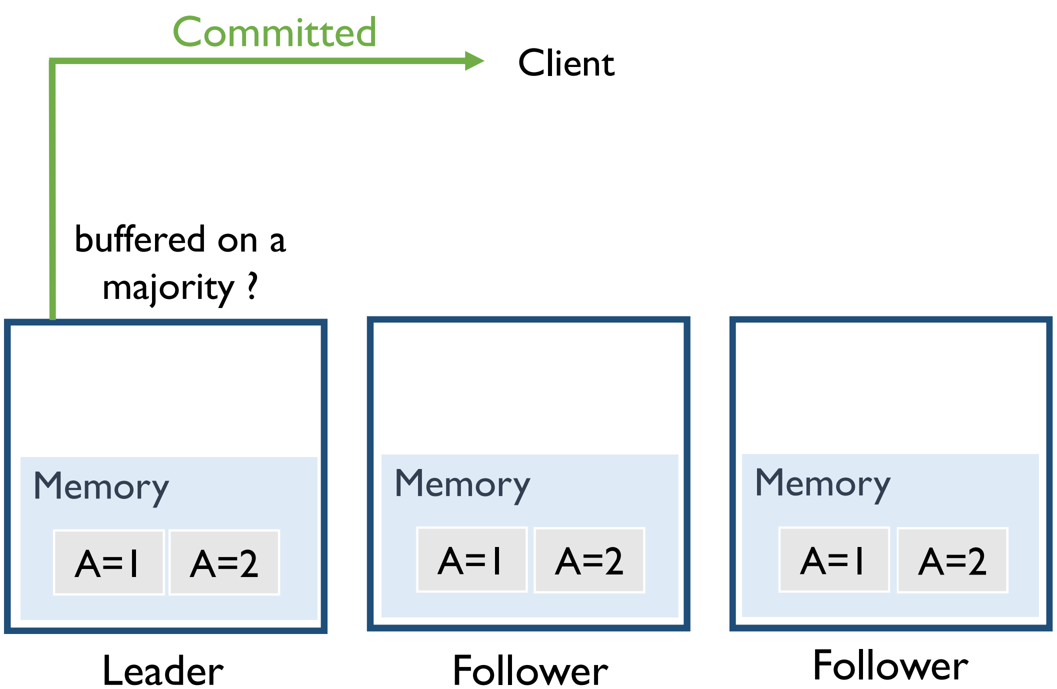
Recovery

Loss-aware: realizes loss, waits for majority



Memory-Durable Protocols (Loss-Aware Recovery)

Update

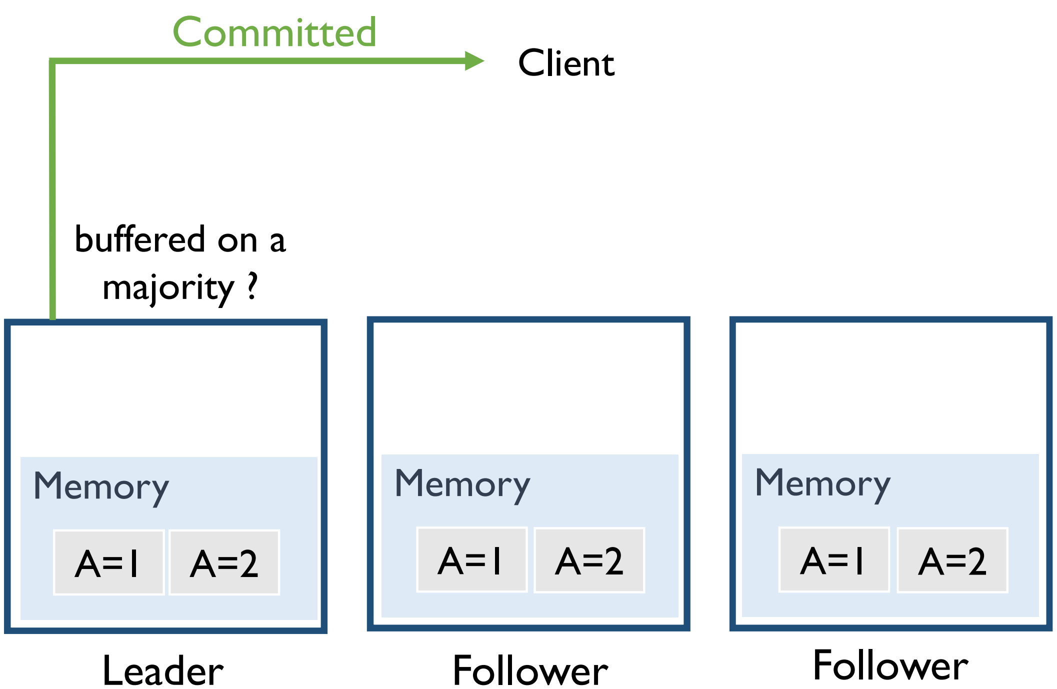


Recovery

Loss-aware: realizes loss, waits for majority

Memory-Durable Protocols (Loss-Aware Recovery)

Update

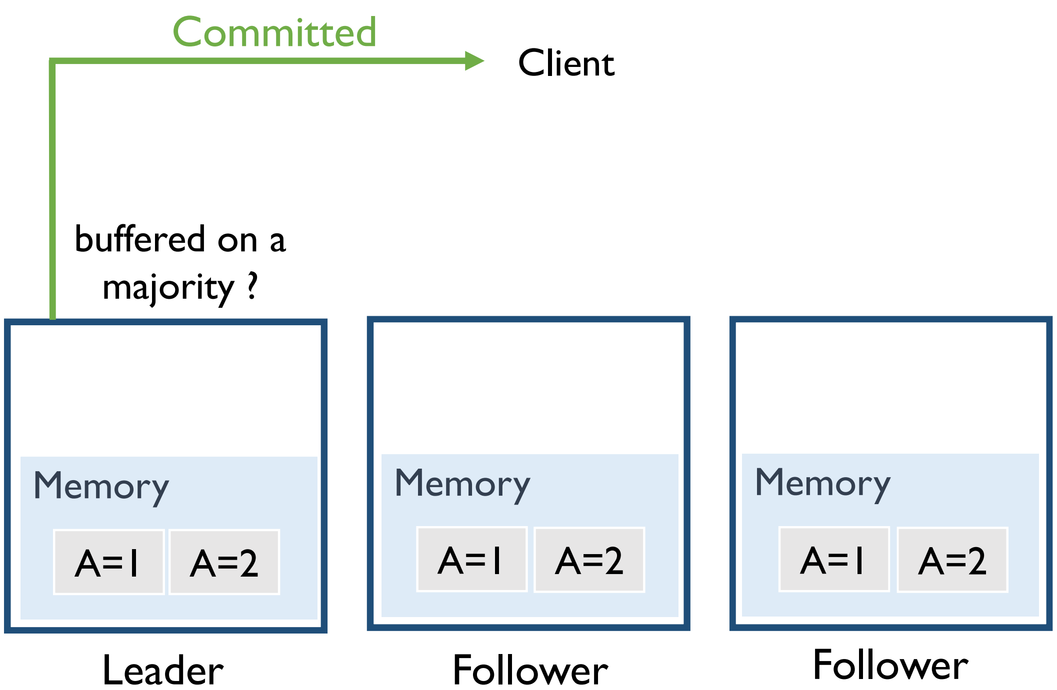


Recovery

Loss-aware: realizes loss, waits for majority

Memory-Durable Protocols (Loss-Aware Recovery)

Update



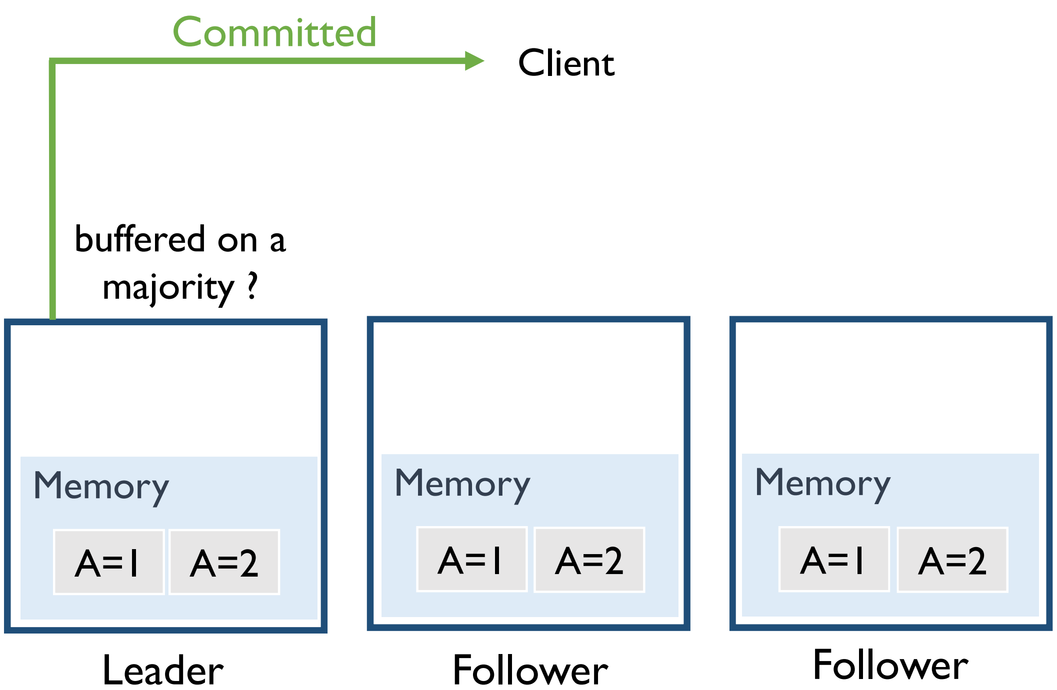
Recovery

Loss-aware: realizes loss, waits for majority

e.g., Viewstamped replication

Memory-Durable Protocols (Loss-Aware Recovery)

Update



Recovery

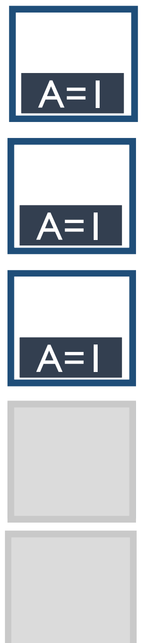
Loss-aware: realizes loss, waits for majority

e.g., Viewstamped replication

Avoids loss (unlike oblivious) but can lead to unavailability

Unavailability Example in Loss-Aware Approach

Unavailability Example in Loss-Aware Approach



A=1 committed
two nodes crashed

Unavailability Example in Loss-Aware Approach

A=1 committed
two nodes crashed

Unavailability Example in Loss-Aware Approach

A=1 committed
two nodes crashed

crashes

Unavailability Example in Loss-Aware Approach

A=1 committed
two nodes crashed

crashes, recovers

Unavailability Example in Loss-Aware Approach

A=1 committed
two nodes crashed

crashes, recovers
cannot collect
majority responses
although majority up – unavailable

Unavailability Example in Loss-Aware Approach

A=1 committed
two nodes crashed

crashes, recovers
cannot collect
majority responses
although majority up – unavailable

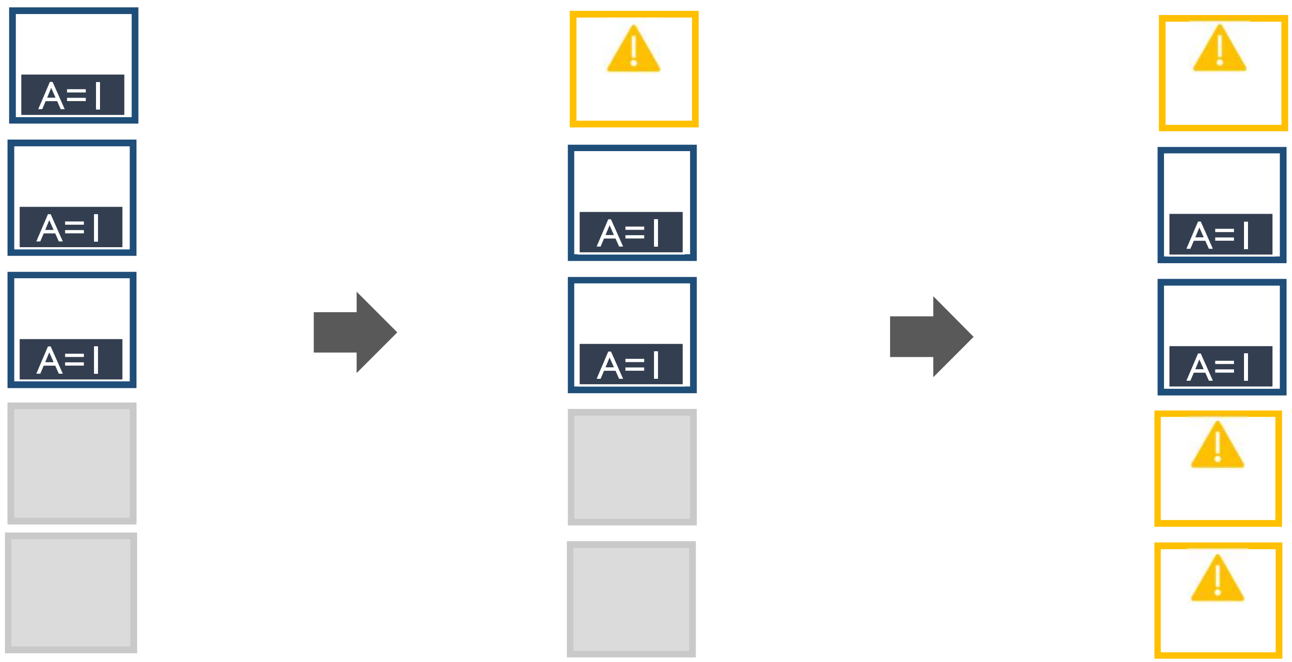
Unavailability Example in Loss-Aware Approach

A=1 committed
two nodes crashed

crashes, recovers
cannot collect
majority responses
although majority up – unavailable

failed nodes recover

Unavailability Example in Loss-Aware Approach



A=1 committed
two nodes crashed

crashes, recovers
cannot collect
majority responses
although majority up – unavailable

failed nodes recover
stay in recovering
unavailable even after
all nodes recover

Outline

Introduction

Distributed updates and crash recovery

Situation-aware updates and crash recovery

- SAUCR insights, guarantees, and overview
- situation-aware updates
- situation-aware crash recovery

Results

Summary and conclusion

SAUCR Intuition and Insight

SAUCR Intuition and Insight

Existing protocols are static in nature: do not adapt to failures

SAUCR Intuition and Insight

Existing protocols are static in nature: do not adapt to failures

always

Memory-durable

buffer even with many failures

poor reliability

SAUCR Intuition and Insight

Existing protocols are **static in nature**: do not adapt to failures

always	always
Memory-durable	**Disk-durable**
buffer even with many failures	persist even when no failures
poor reliability	poor performance

SAUCR Intuition and Insight

Existing protocols are static in nature: do not adapt to failures

always	always
Memory-durable	**Disk-durable**
buffer even with many failures	persist even when no failures
poor reliability	poor performance

Insight: reacting to failures and adapting to situation can achieve reliability and performance

SAUCR Intuition and Insight

Existing protocols are static in nature: do not adapt to failures

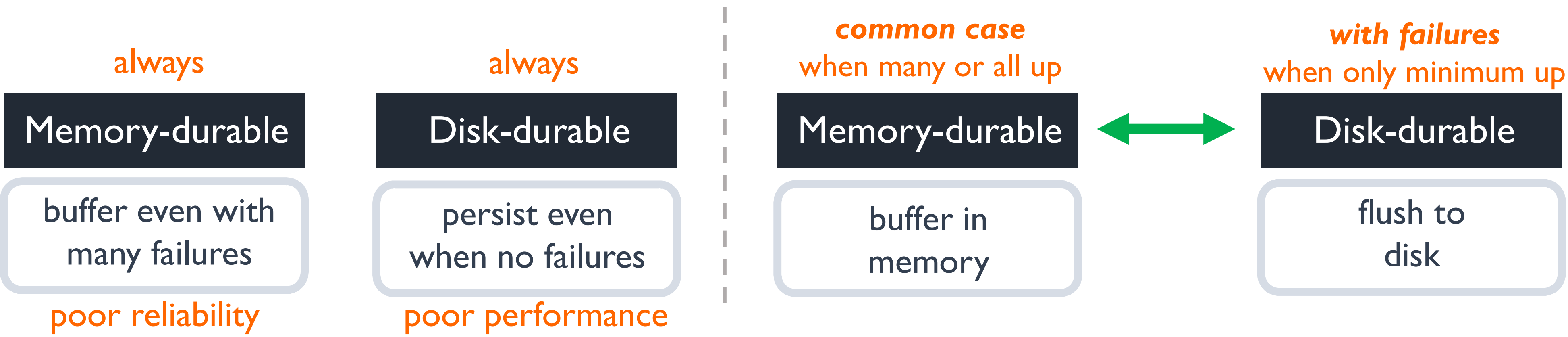
always	always	***common case*** when many or all up
Memory-durable	Disk-durable	Memory-durable
buffer even with many failures	persist even when no failures	buffer in memory
poor reliability	poor performance	

Insight: reacting to failures and adapting to situation can achieve reliability and performance

- ➡ when no or few failures could buffer in memory

SAUCR Intuition and Insight

Existing protocols are static in nature: do not adapt to failures



Insight: reacting to failures and adapting to situation can achieve reliability and performance

➡ when no or few failures could buffer in memory

➡ when failure arise, flush

Guarantees Depend upon Simultaneity of Failures

Guarantees Depend upon Simultaneity of Failures

With non-simultaneous, gap exists, SAUCR can react and ensures durability

Guarantees Depend upon Simultaneity of Failures

With non-simultaneous, gap exists, SAUCR can react and ensures durability

➡ independent: likelihood of many nodes failing together is negligible

Guarantees Depend upon Simultaneity of Failures

With non-simultaneous, gap exists, SAUCR can react and ensures durability

- ➡ independent: likelihood of many nodes failing together is negligible
- ➡ correlated: many nodes fail together
  - ➡ although many nodes fail, not necessarily simultaneous; most cases, non-simultaneous

Guarantees Depend upon Simultaneity of Failures

With non-simultaneous, gap exists, SAUCR can react and ensures durability

- ➡ independent: likelihood of many nodes failing together is negligible
- ➡ correlated: many nodes fail together
  - ➡ although many nodes fail, not necessarily simultaneous; most cases, non-simultaneous

With simultaneous correlated, no gap, SAUCR cannot react, unavailable

Guarantees Depend upon Simultaneity of Failures

With non-simultaneous, gap exists, SAUCR can react and ensures durability

- independent: likelihood of many nodes failing together is negligible
- correlated: many nodes fail together
  - although many nodes fail, not necessarily simultaneous; most cases, non-simultaneous

With simultaneous correlated, no gap, SAUCR cannot react, unavailable

We conjecture they are extremely rare: a gap exists between failures

- correlated but a few seconds apart [Ford et al., OSDI '10]
- analysis reveals a gap of 50 ms or more almost always

Guarantees Depend upon Simultaneity of Failures

With non-simultaneous, gap exists, SAUCR can react and ensures durability

- independent: likelihood of many nodes failing together is negligible
- correlated: many nodes fail together
  - although many nodes fail, not necessarily simultaneous; most cases, non-simultaneous

With simultaneous correlated, no gap, SAUCR cannot react, unavailable

We conjecture they are extremely rare: a gap exists between failures

- correlated but a few seconds apart [Ford et al., OSDI '10]
- analysis reveals a gap of 50 ms or more almost always

Most cases: any no. of independent and non-simultaneous correlated – same as disk-durable
Rare cases: more than a majority crash truly simultaneously – remain unavailable

SAUCR Overview

SAUCR Overview

Updates

- when more than a majority up, buffer updates in memory – fast mode
  - e.g., 4 or 5 nodes up in a 5-node cluster

SAUCR Overview

Updates

- when more than a majority up, buffer updates in memory – fast mode
    - e.g., 4 or 5 nodes up in a 5-node cluster
- When nodes fail and only a bare majority alive, flush to disk – slow mode
    - e.g., only 3 nodes up in a 5-node cluster

SAUCR Overview

Updates

- when more than a majority up, buffer updates in memory – fast mode
  - e.g., 4 or 5 nodes up in a 5-node cluster
- When nodes fail and only a bare majority alive, flush to disk – slow mode
  - e.g., only 3 nodes up in a 5-node cluster

Crash Recovery

- when a node recovers from a crash, it recovers its data
  - either from its disk (if crashed in slow mode)
  - or from other nodes (if crashed in fast mode)

Situation-Aware Updates

Situation-Aware Updates

all nodes up

Situation-Aware Updates

all nodes up
fast mode -
buffer updates

Situation-Aware Updates

all nodes up
fast mode -
buffer updates

4 nodes up
(more than majority)

Situation-Aware Updates

all nodes up
fast mode -
buffer updates

4 nodes up
(more than majority)
remain in
fast mode

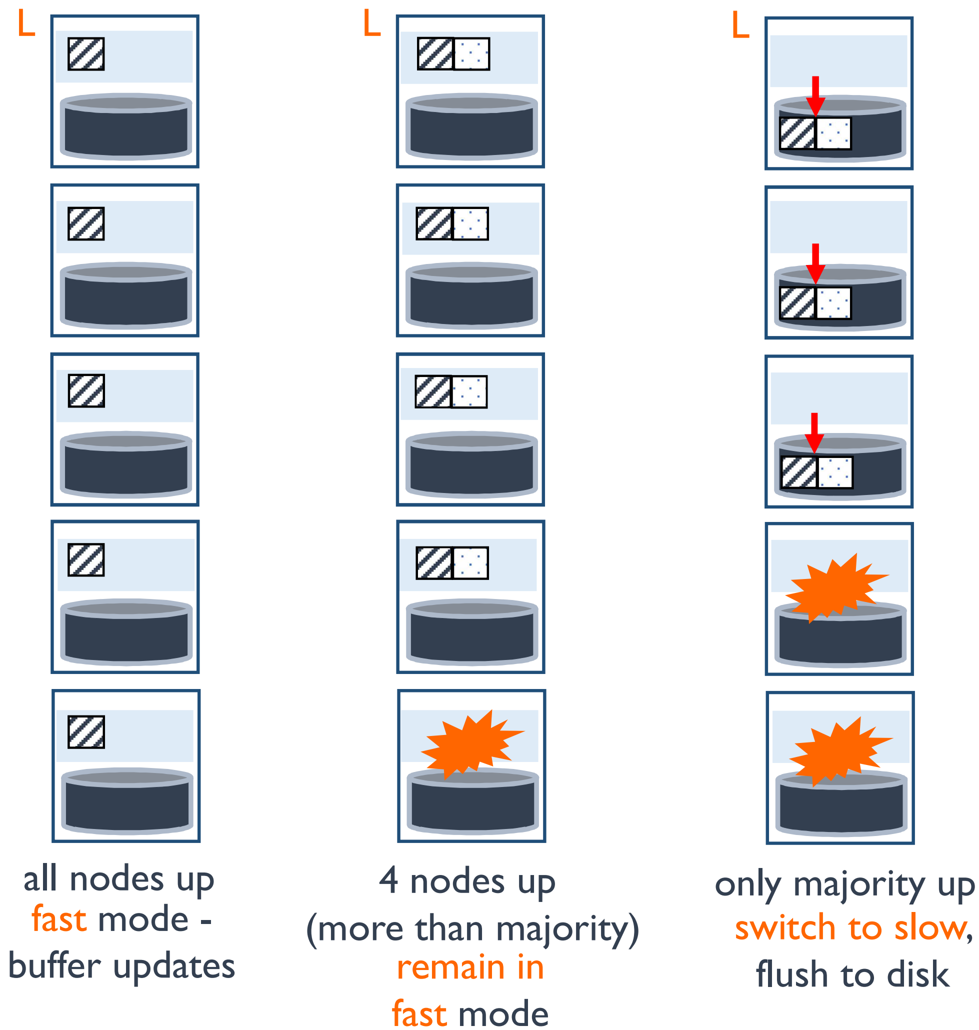
Situation-Aware Updates

all nodes up
fast mode -
buffer updates

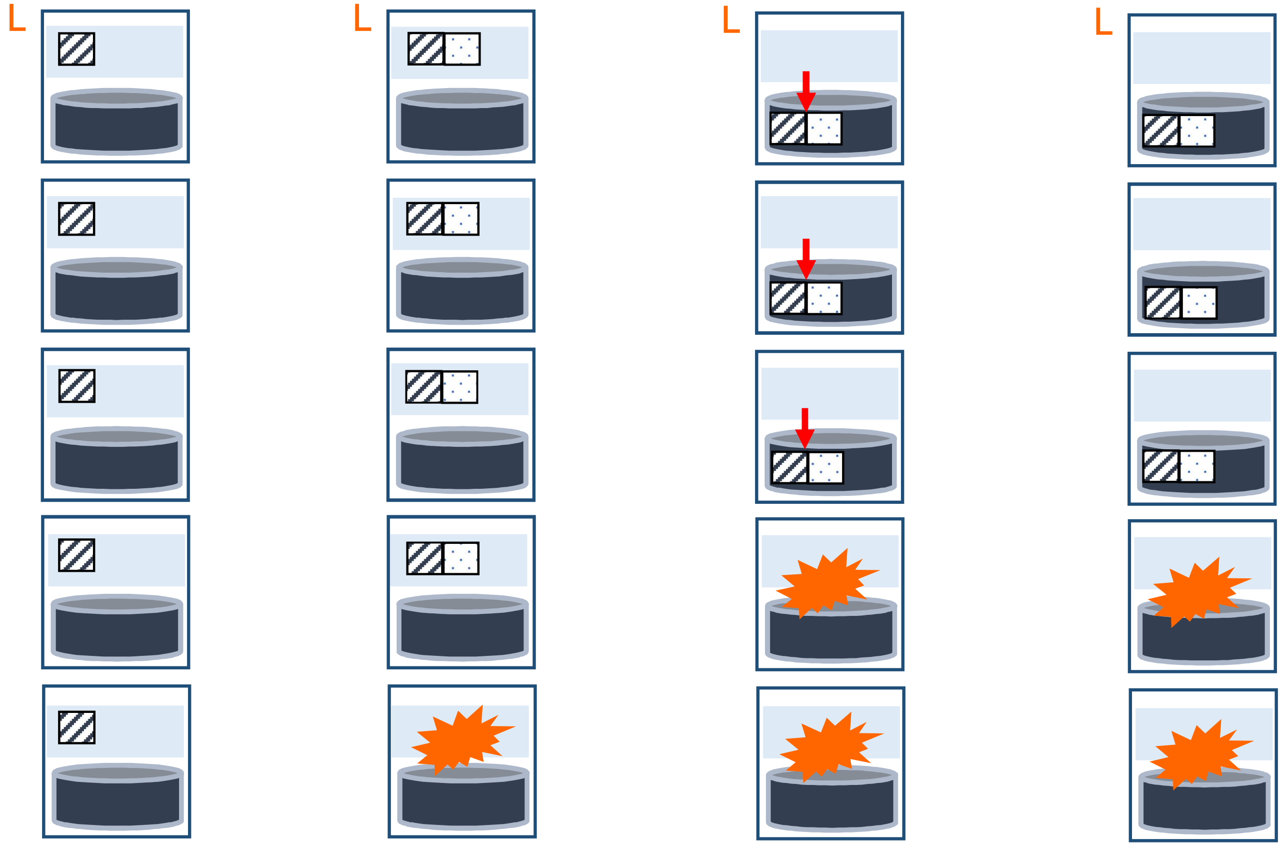
4 nodes up
(more than majority)
remain in
fast mode

only majority up

Situation-Aware Updates



Situation-Aware Updates

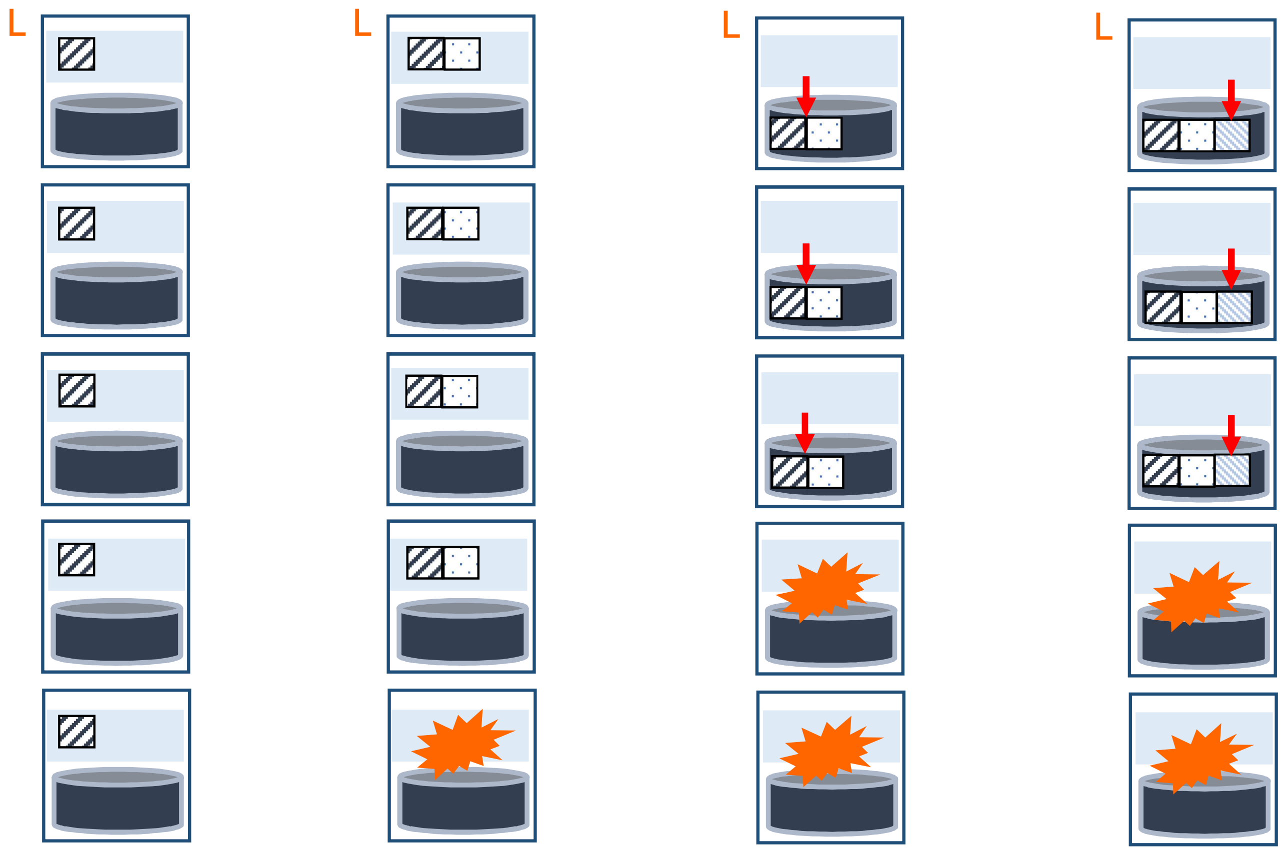


all nodes up
fast mode -
buffer updates

4 nodes up
(more than majority)
remain in
fast mode

only majority up
switch to slow,
flush to disk

Situation-Aware Updates



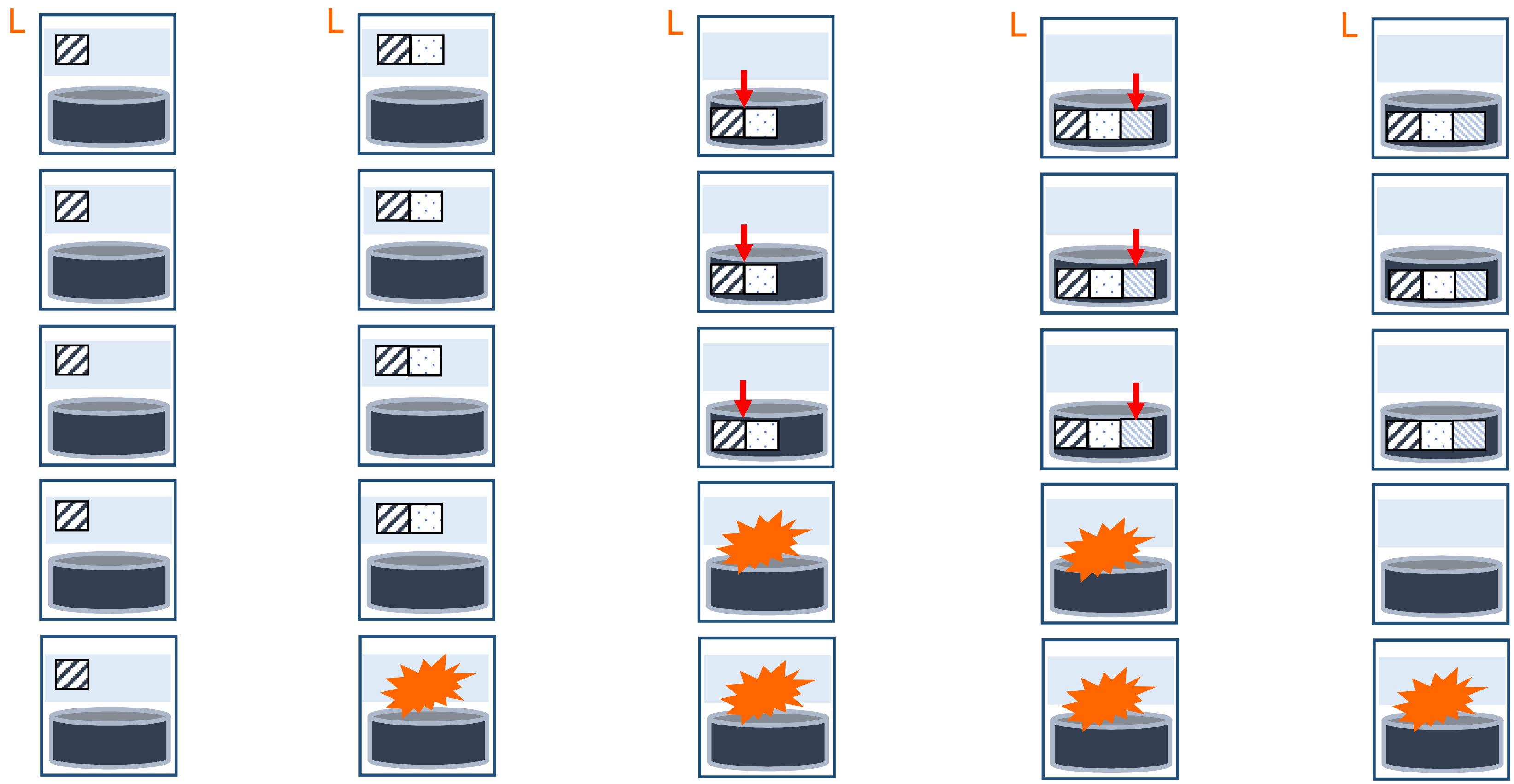
all nodes up
fast mode -
buffer updates

4 nodes up
(more than majority)
remain in
fast mode

only majority up
switch to slow,
flush to disk

commit
subsequent
updates in slow
mode

Situation-Aware Updates



all nodes up
fast mode -
buffer updates

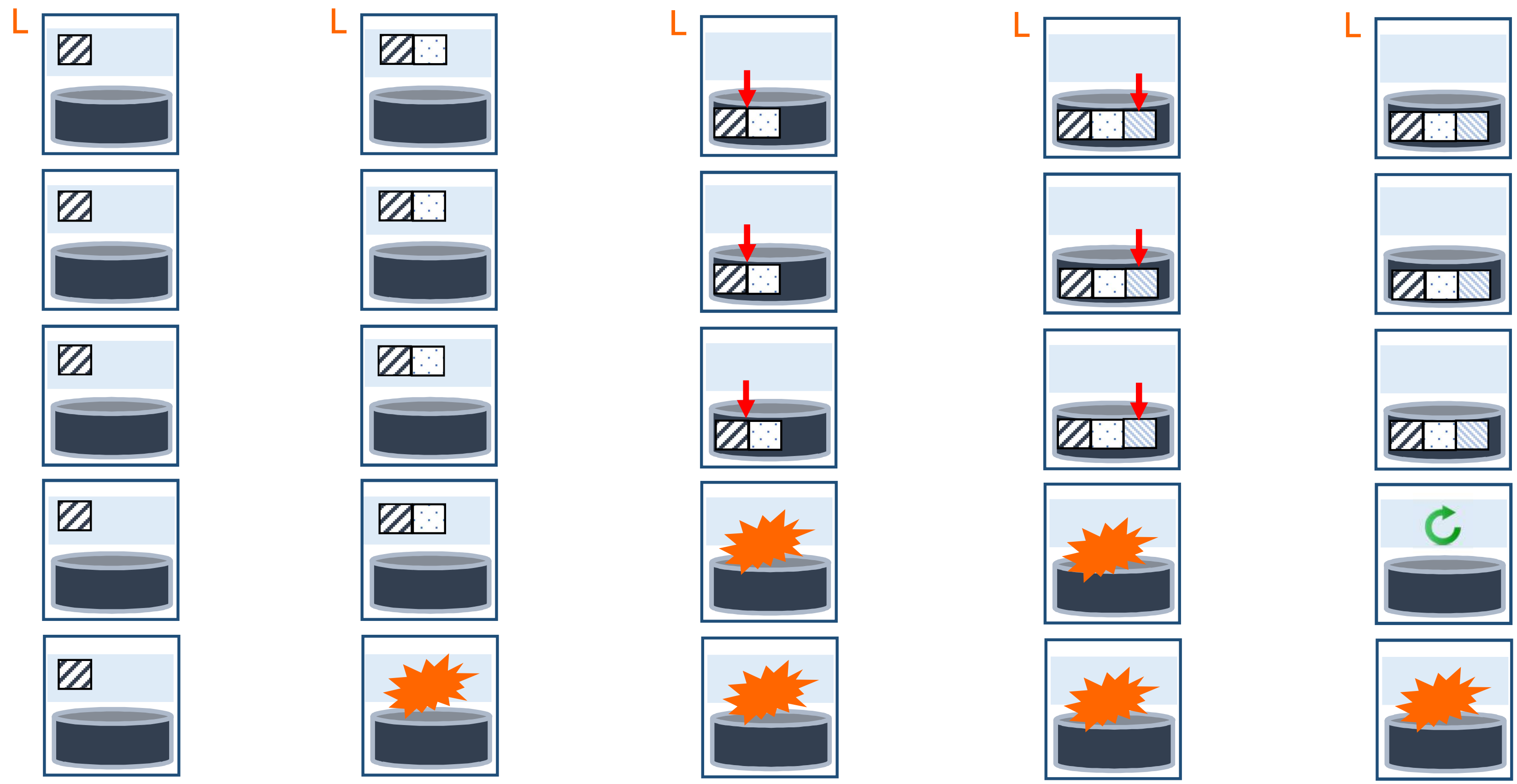
4 nodes up
(more than majority)
remain in
fast mode

only majority up
switch to slow,
flush to disk

commit
subsequent
updates in slow
mode

one node recovers
and catches up;

Situation-Aware Updates



all nodes up
fast mode -
buffer updates

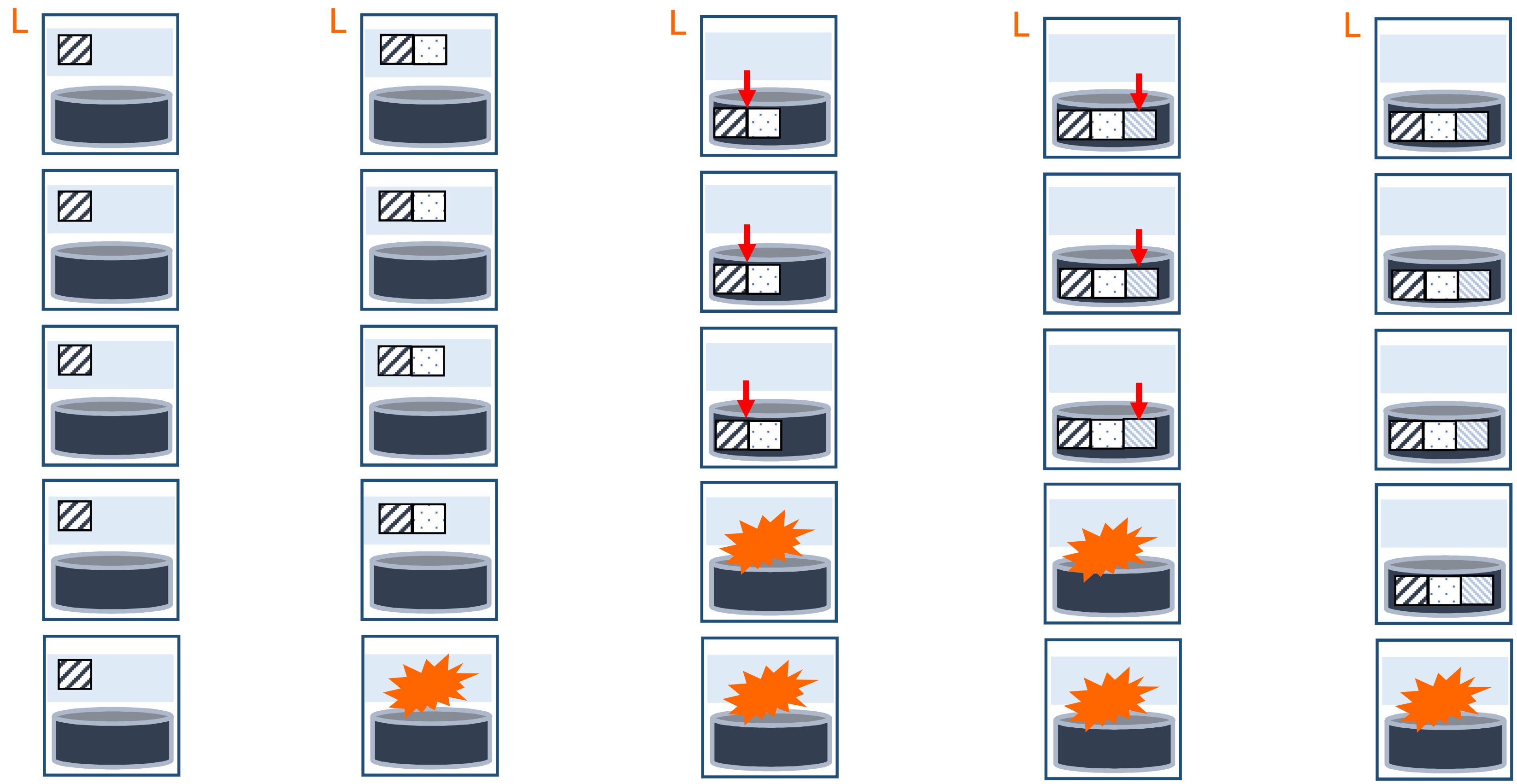
4 nodes up
(more than majority)
remain in
fast mode

only majority up
switch to slow,
flush to disk

commit
subsequent
updates in slow
mode

one node recovers
and catches up;

Situation-Aware Updates



all nodes up
fast mode -
buffer updates

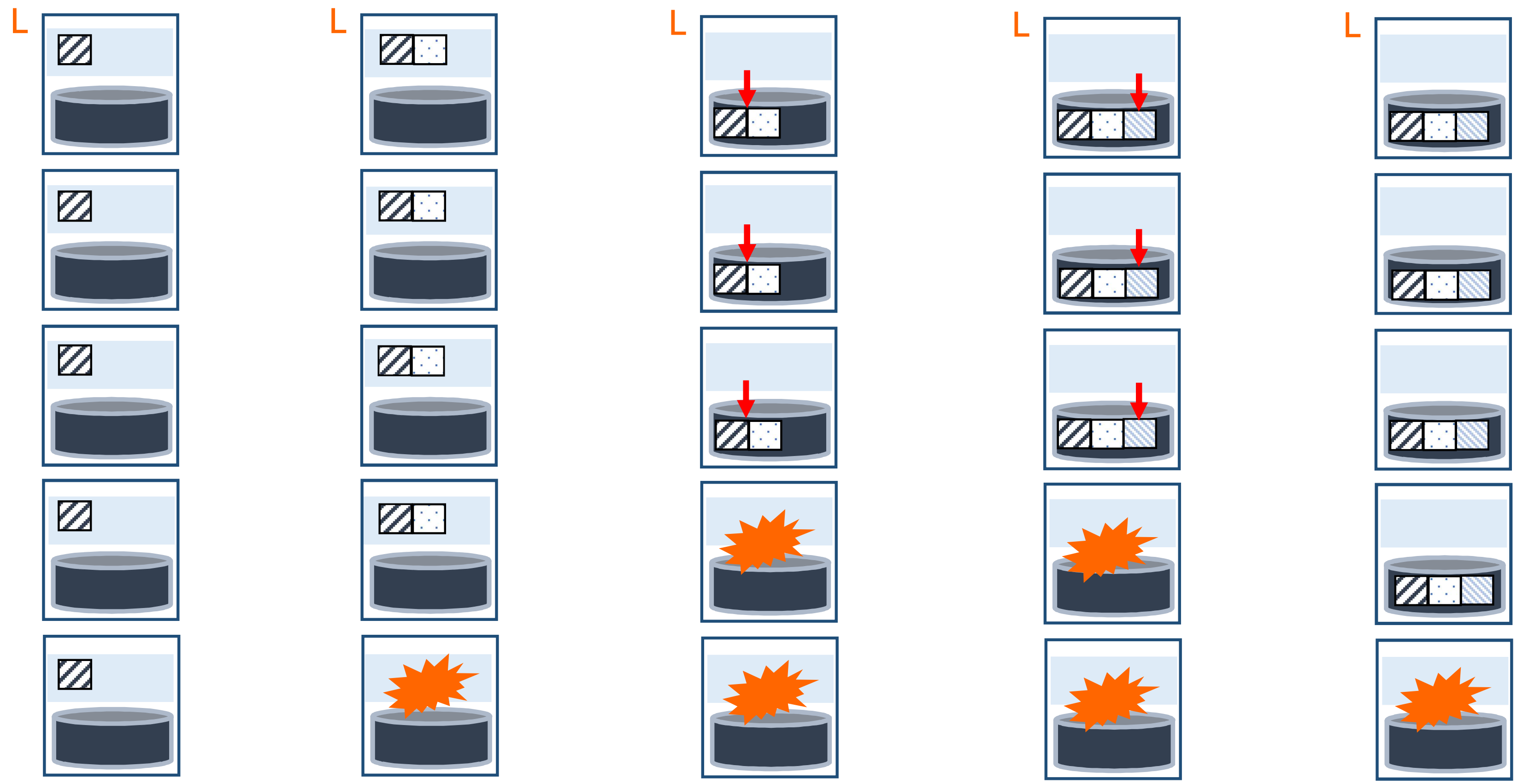
4 nodes up
(more than majority)
remain in
fast mode

only majority up
switch to slow,
flush to disk

commit
subsequent
updates in slow
mode

one node recovers
and catches up;

Situation-Aware Updates



all nodes up
fast mode -
buffer updates

4 nodes up
(more than majority)
remain in
fast mode

only majority up
switch to slow,
flush to disk

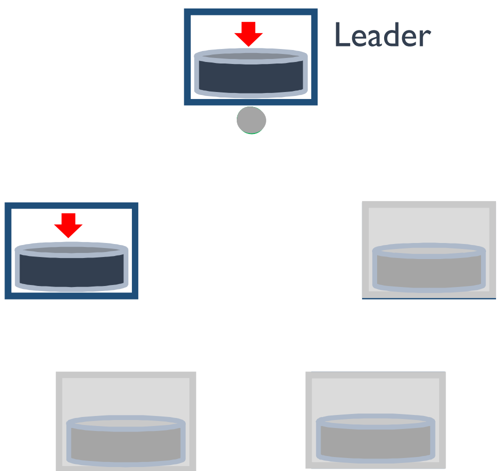
commit
subsequent
updates in slow
mode

one node recovers
and catches up;
switch to fast

Failure Reaction

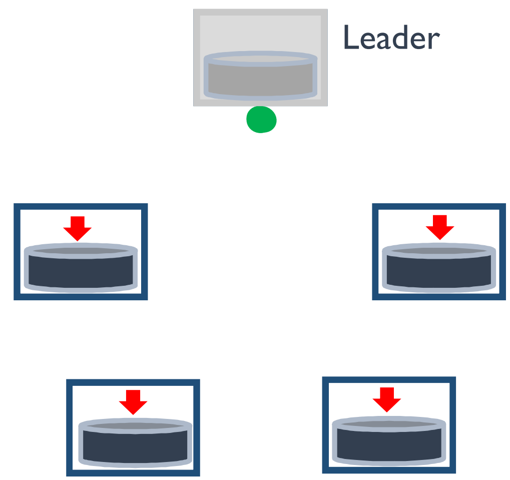
Basic failure-detection mechanism: heartbeats

Follower failures



remain in fast mode
switch to slow mode
steps down

Leader failures



on a missing heartbeat, followers flush to disk

Challenges: too many packets, spurious elections, too much data to flush

Techniques in the paper …

Result: can react to failures even when they are only a few milliseconds apart, preserving durability and availability

Mode-Aware Crash Recovery

Mode-Aware Crash Recovery

Disk-durable: always recover from disk

Memory-durable: always recover from other nodes (loss-aware)

Mode-Aware Crash Recovery

Disk-durable: always recover from disk

Memory-durable: always recover from other nodes (loss-aware)

Mode-Aware Crash Recovery

Disk-durable: always recover from disk

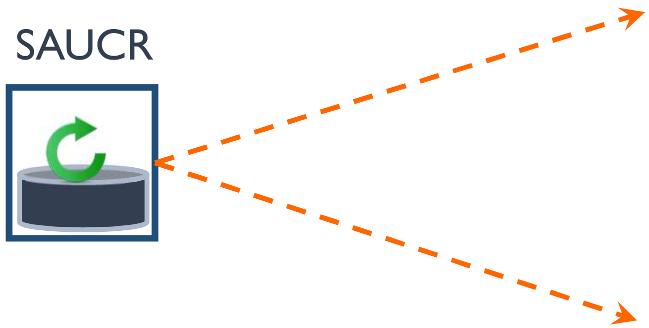
Memory-durable: always recover from other nodes (loss-aware)

SAUCR

Mode-Aware Crash Recovery

Disk-durable: always recover from disk

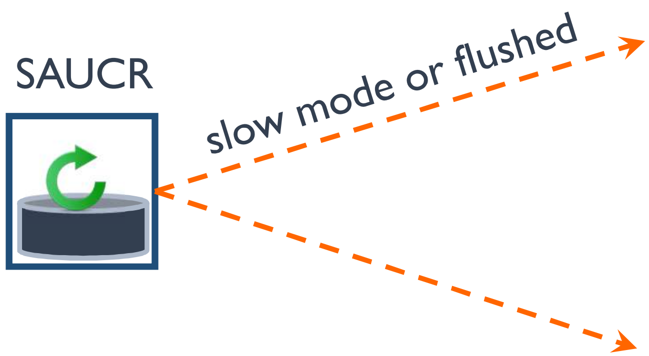
Memory-durable: always recover from other nodes (loss-aware)



Mode-Aware Crash Recovery

Disk-durable: always recover from disk

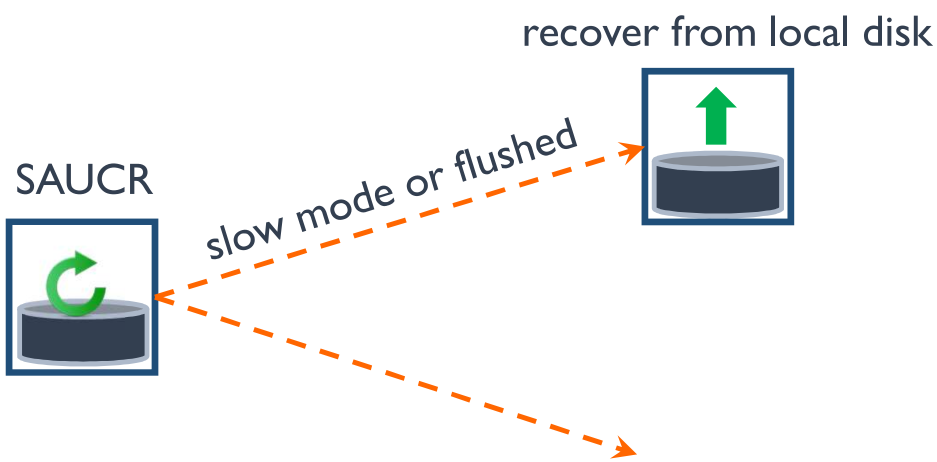
Memory-durable: always recover from other nodes (loss-aware)



Mode-Aware Crash Recovery

Disk-durable: always recover from disk

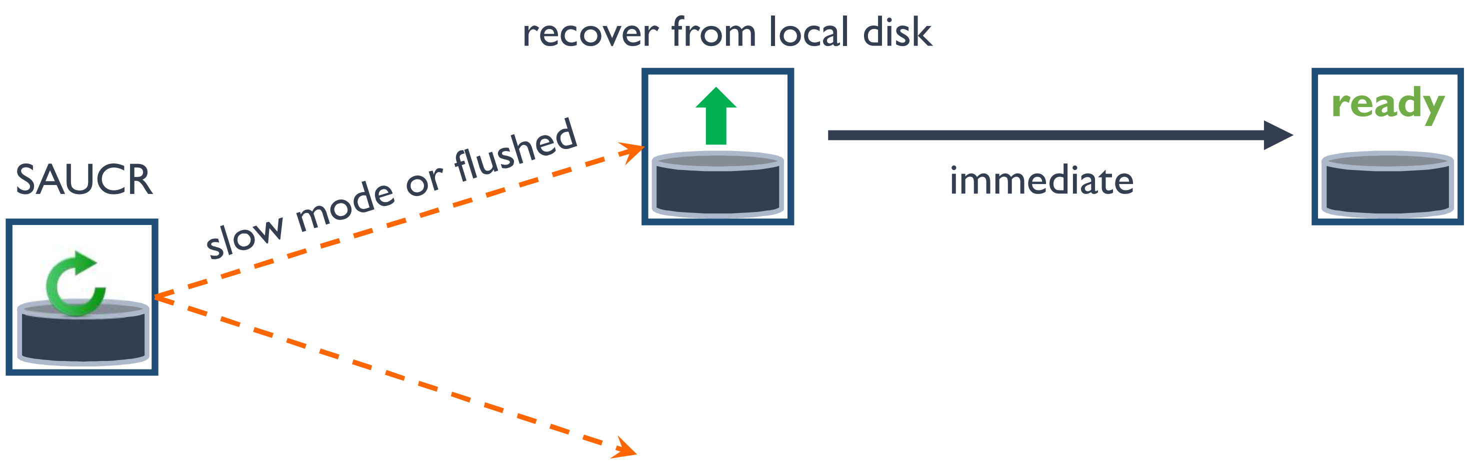
Memory-durable: always recover from other nodes (loss-aware)



Mode-Aware Crash Recovery

Disk-durable: always recover from disk

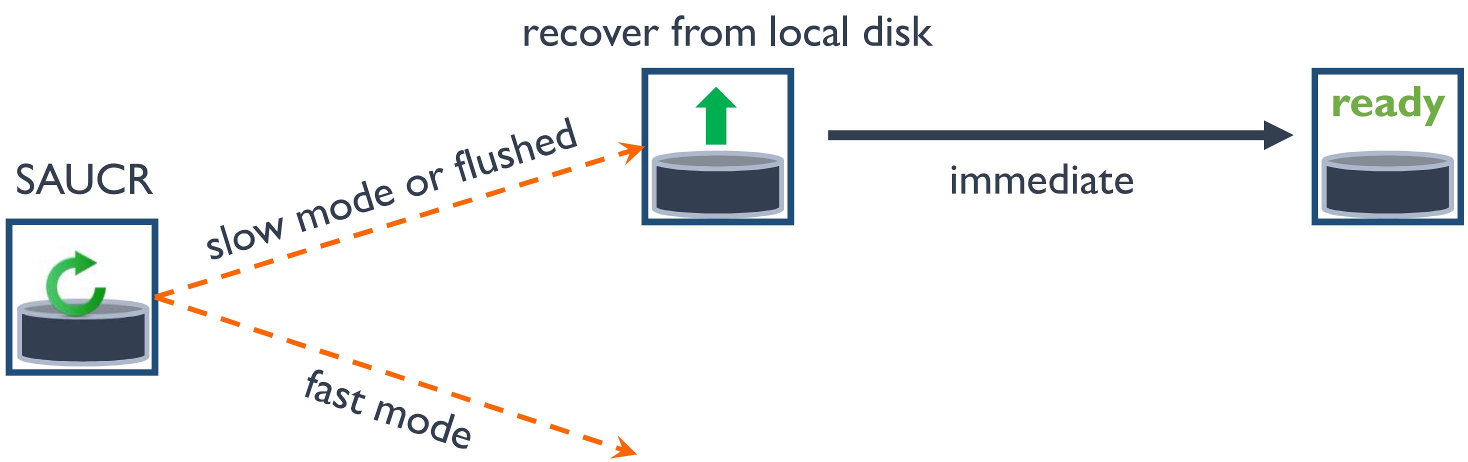
Memory-durable: always recover from other nodes (loss-aware)



Mode-Aware Crash Recovery

Disk-durable: always recover from disk

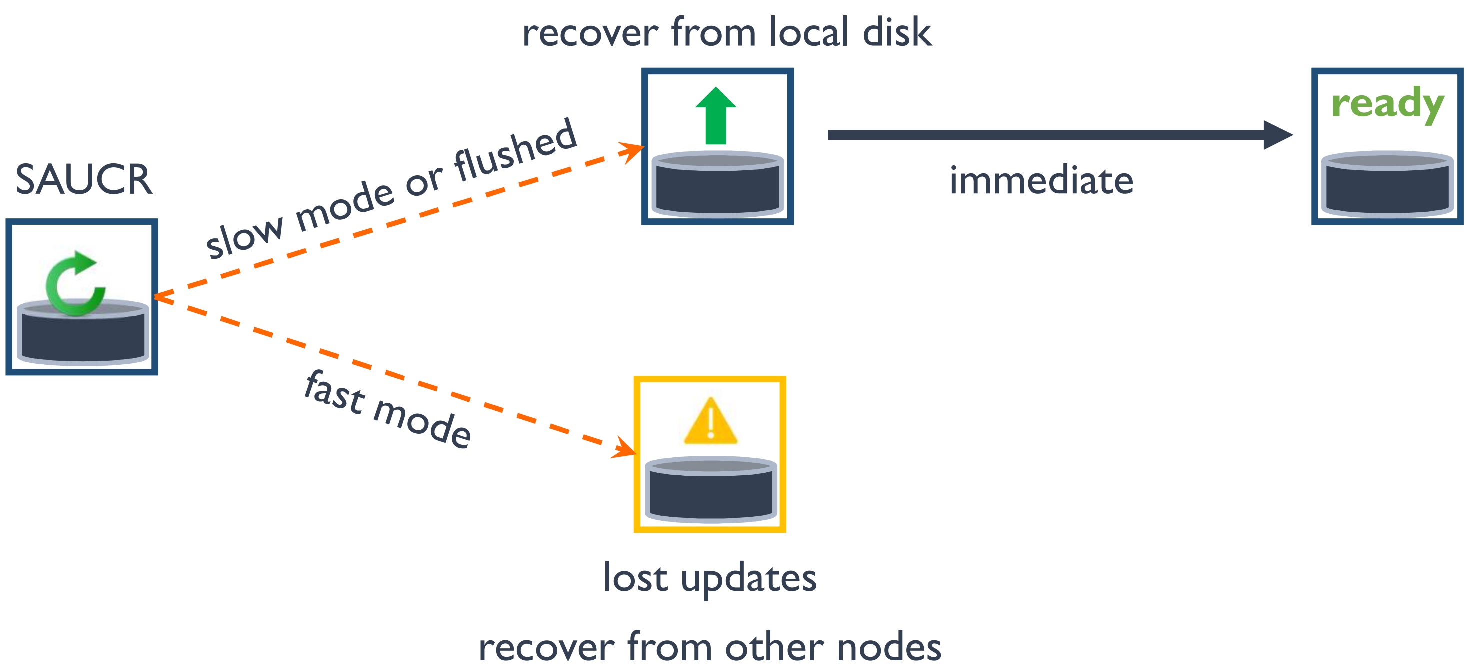
Memory-durable: always recover from other nodes (loss-aware)



Mode-Aware Crash Recovery

Disk-durable: always recover from disk

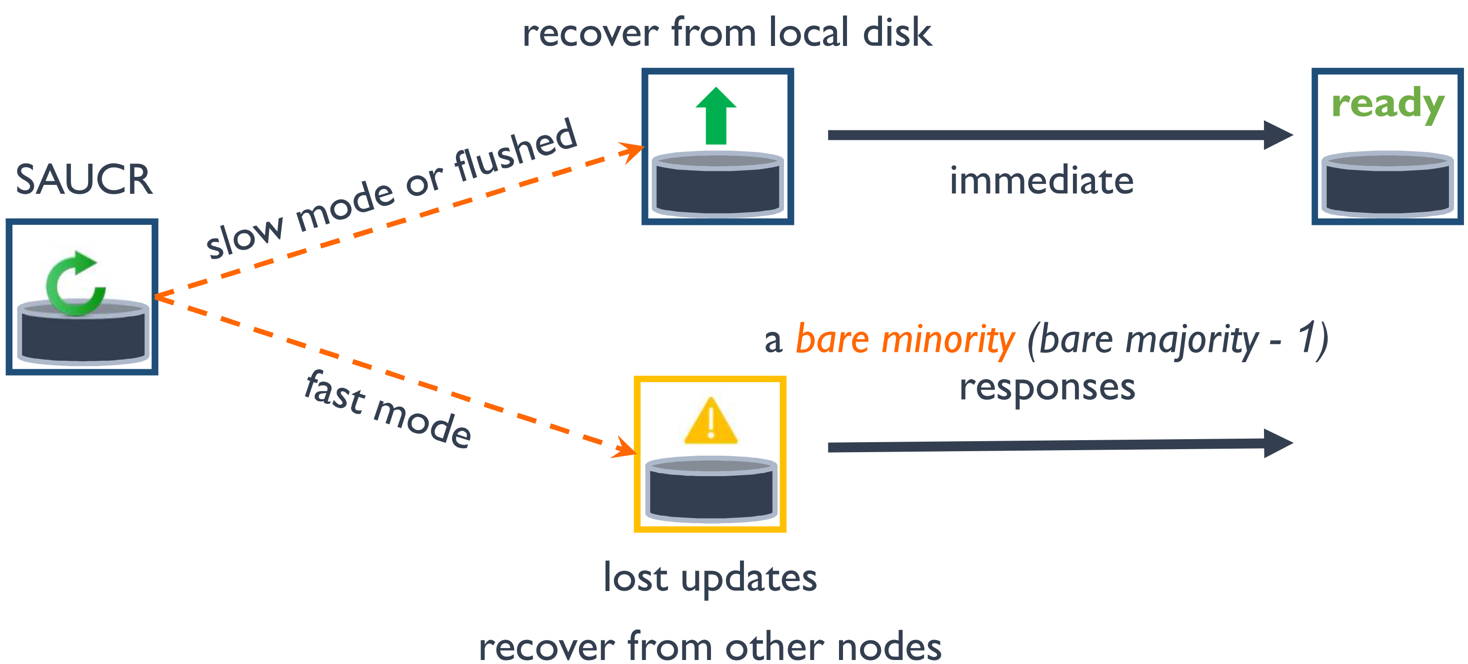
Memory-durable: always recover from other nodes (loss-aware)



Mode-Aware Crash Recovery

Disk-durable: always recover from disk

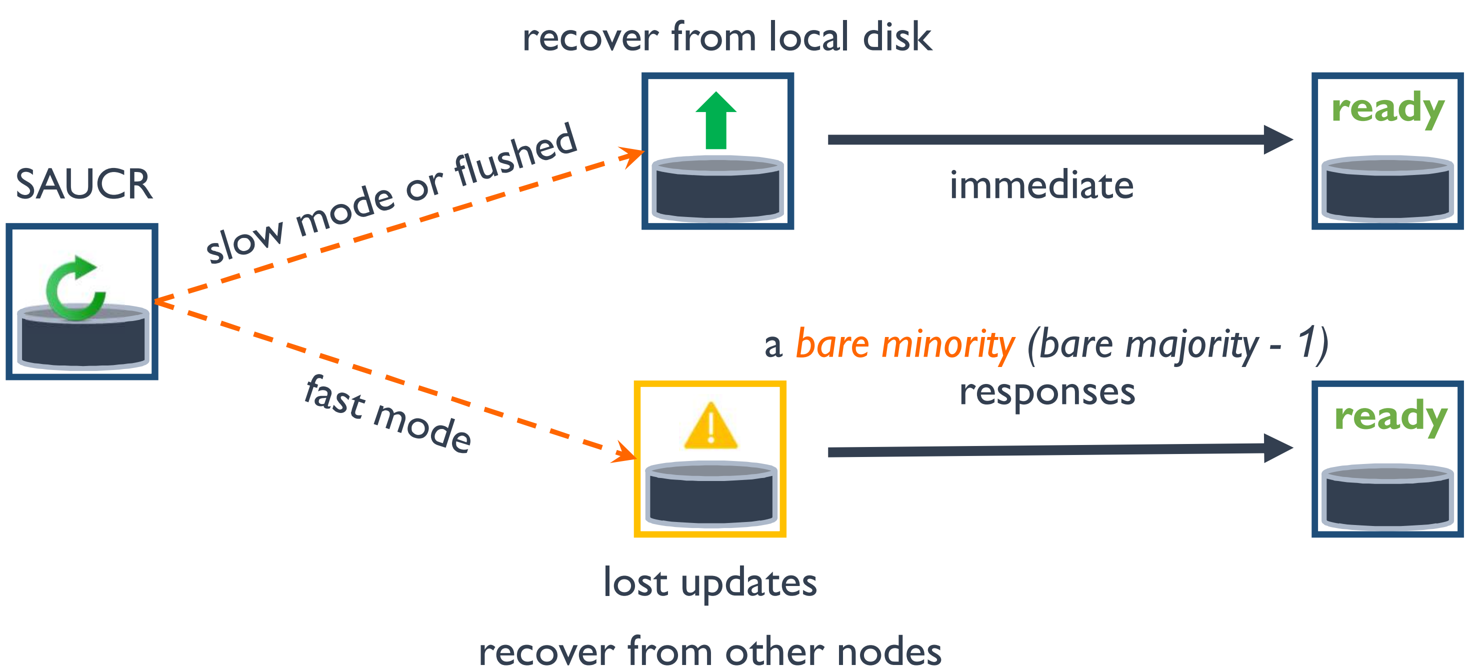
Memory-durable: always recover from other nodes (loss-aware)



Mode-Aware Crash Recovery

Disk-durable: always recover from disk

Memory-durable: always recover from other nodes (loss-aware)



Intuition for why SAUCR's recovery is safe

Intuition for why SAUCR's recovery is safe

Assume update-A committed, S1 recovers and has seen A before crash

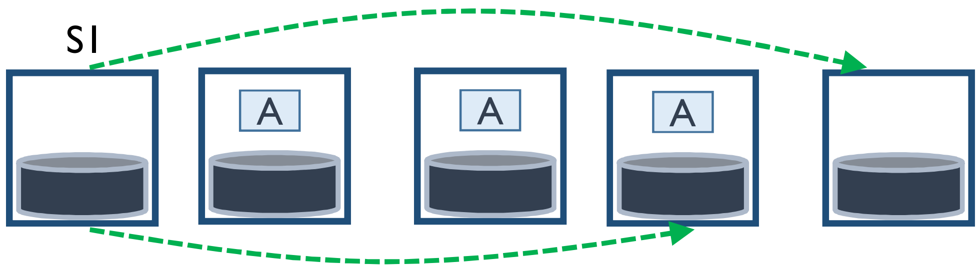
Intuition for why SAUCR's recovery is safe

Assume update-A committed, S1 recovers and has seen A before crash

Safety condition: update-A must be recovered

Intuition for why SAUCR's recovery is safe

Assume update-A committed, S1 recovers and has seen A before crash

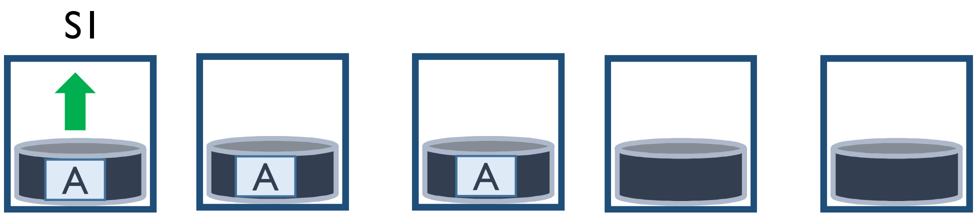


Safety condition: update-A must be recovered

If A was committed in fast mode, then at least one in any bare minority must contain update-A

Intuition for why SAUCR's recovery is safe

Assume update-A committed, S1 recovers and has seen A before crash



Safety condition: update-A must be recovered

If A was committed in fast mode, then at least one in any bare minority must contain update-A

If update-A was committed in slow mode, S1 recovers from disk

Intuition for why SAUCR's recovery is safe

Assume update-A committed, S1 recovers and has seen A before crash

Safety condition: update-A must be recovered

If A was committed in fast mode, then at least one in any bare minority must contain update-A

If update-A was committed in slow mode, S1 recovers from disk

Proof sketch in the paper …

Outline

Introduction

Distributed updates and crash recovery

Situation-aware updates and crash recovery

Results

Summary and conclusion

Evaluation

We implement in SAUCR in ZooKeeper

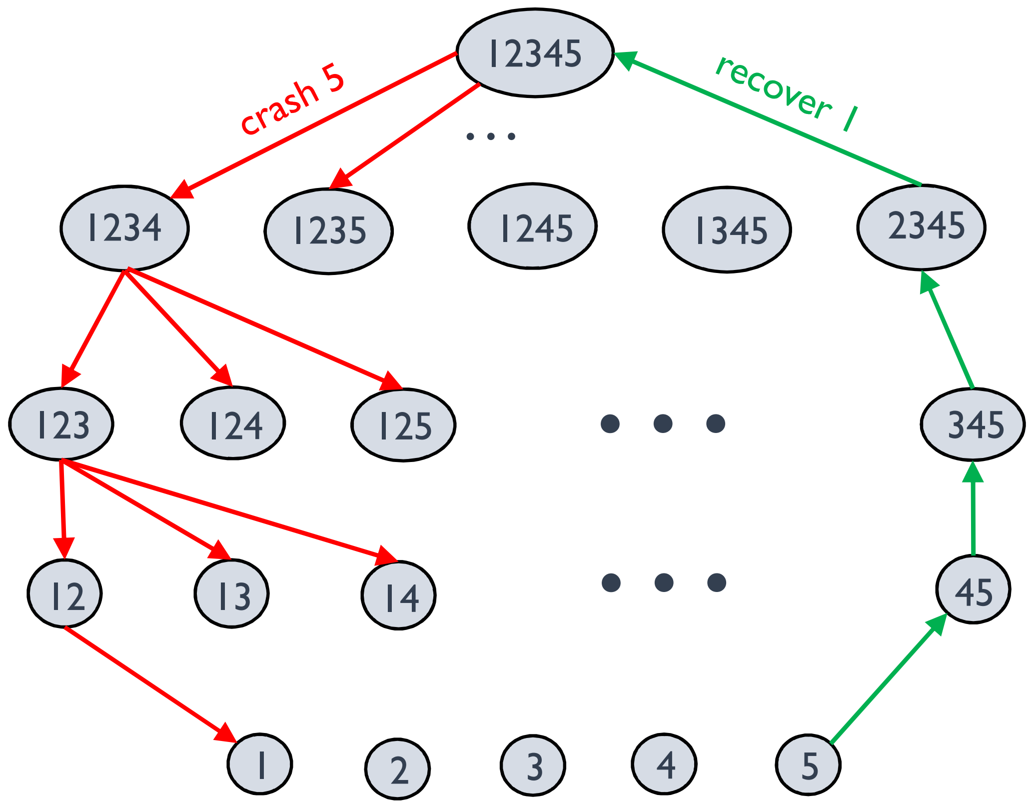
Compare SAUCR's reliability and performance against

- ➡ disk-durable ZooKeeper (forceSync = true)
- ➡ memory-durable ZooKeeper (forceSync = false)
- ➡ viewstamped replication (ideal model)

Reliability Testing

Cluster crash-testing framework
Generates cluster-state sequences

How it works?
Please see our paper…

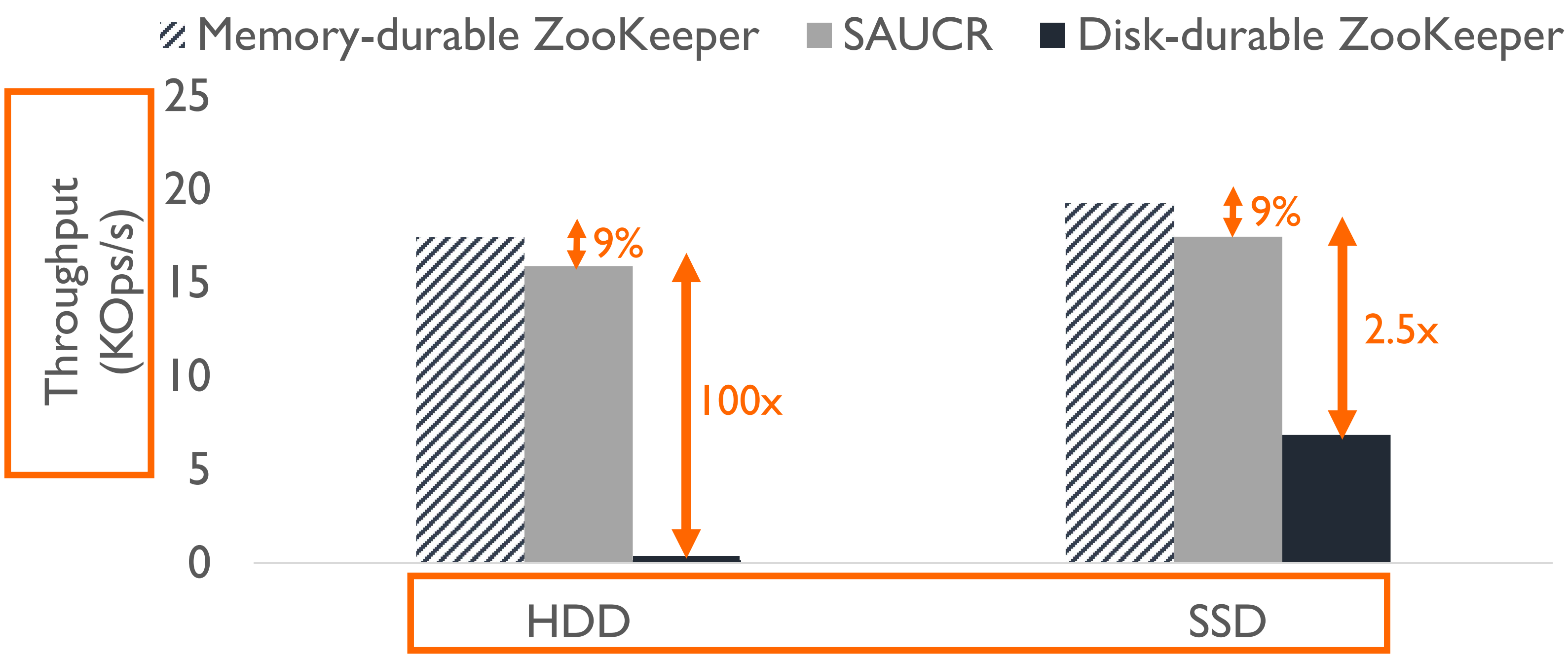


Reliability Results

	Non-Simultaneous			Simultaneous		
Systems	**Correct**	**Unavailable**	**Data loss**	**Correct**	**Unavailable**	**Data loss**
memory-durable zookeeper	703	0	561	703	0	561
viewstamped replication	217	1047	0	217	1047	0
disk-durable zookeeper	1264	0	0	1264	0	0
SAUCR	1264	0	0	1200	64	0

non-simultaneous: gap of 50 ms, simultaneous: no gap
memory-durable zookeeper silently loses data
viewstamped replication leads to permanent unavailability
SAUCR reacts to non-simultaneous – durable and available
other systems behave the same as non-simultaneous cases
simultaneous: SAUCR by design remains unavailable in some cases

Macro-benchmark Performance: YCSB-load



Compared to disk-durable, both memory-durable and SAUCR are faster

SAUCR's performance matches memory-durable ZooKeeper

- within 9% of memory-durable Zookeeper even for write-intensive workloads
- overheads because SAUCR writes to one additional node

Summary

Replication protocols are an important foundation

- need to be performant, yet also provide high reliability

Dichotomy: disk-durable vs. memory-durable protocols

- unsavory choices: either performant or reliable

SAUCR – situation-aware updates and crash recovery

- provides both high performance and reliability

Conclusions

Paying careful attention to how failures occur

- ➡ can find approaches that provide both performance and reliability
- ➡ more data from real-world deployments?

Hybrid approach – an effective systems-design technique – applicable to distributed updates and recovery too

- ➡ worthwhile to look at other important protocols/systems where we make similar two-ends-of-the-spectrum tradeoffs?

Thank you!

Poster #6