Exploiting Nil-Externality for Fast Replicated Storage

Artifacts Available acm V1.1

Artifacts Evaluated acm Functional V1.1

Results Reproduced acm V1.1

Aishwarya Ganesan, Ramnatthan Alagappan (***VMware Research***)

Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau (*University of Wisconsin – Madison*)

Interfaces

> "Defining interfaces is the most important part of system design"
>
> – Butler Lampson, Hints for Computer System Design

Well-designed interfaces lead to desirable properties

- idempotent interfaces make failure recovery simple [Sandberg, 1986]
- commutative interfaces enable scalable implementation [Clements et al., 2013]

Do some storage interfaces enable higher performance than others?

Nil-Externality

Nil-externalizing (or nilext) interface

can modify storage system state in any way
but does not externalize its effects or state immediately

A system can defer executing a nilext operation, improving performance

Nilext interfaces are prevalent in storage systems

all updates are nilext in key-value stores such as RocksDB and LevelDB
Twemcache production traces reveal in 80% clusters, 90% updates are nilext

This Work

In this paper, we exploit nilext interfaces for fast replicated storage

Current replication protocols are oblivious to storage interfaces

- involve expensive coordination to order requests
- updates incur two roundtrips

We build Skyros, a nilext-aware replication protocol

Key insight: defer coordination until state is externalized

- complete nilext updates in one roundtrip

Skyros offers linearizability and achieves up to 3x lower latencies compared to Paxos (w/ batching)

Outline

Introduction

➡ Strongly consistent storage background

Nilext-aware replication

Evaluation

Strongly Consistent Storage Systems

A standard approach to building strongly consistent storage

Local Storage System

RocksDB

levelDB

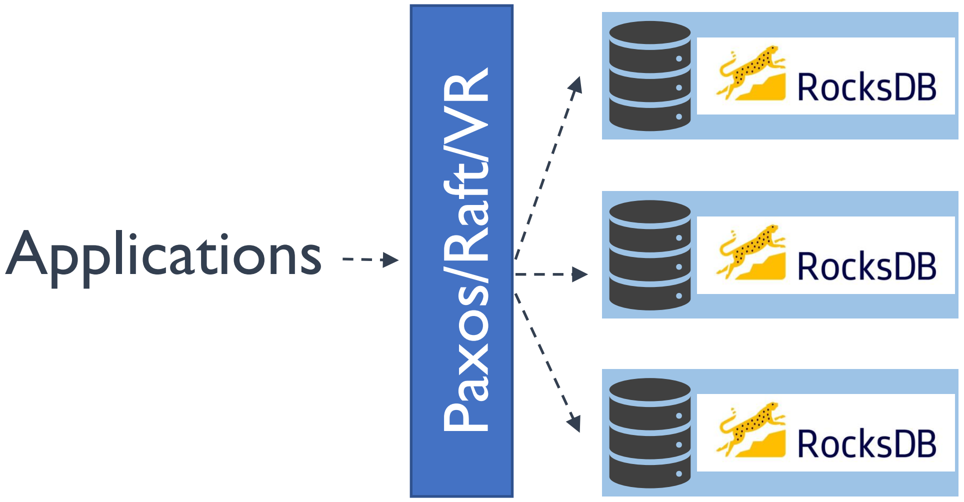
+

Replication Protocol

Paxos

raft

Replicated Storage

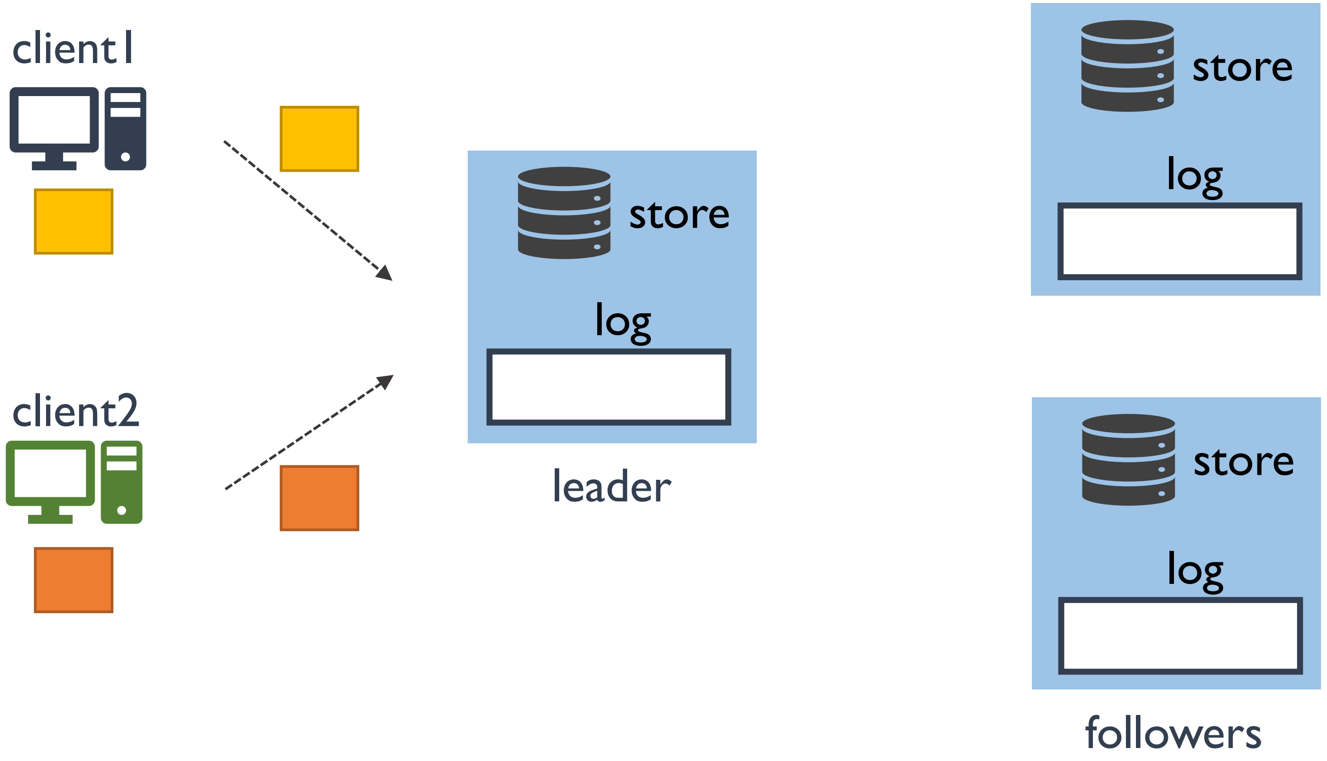


Replicas execute same operations in same order – ensures linearizability

Examples: ZippyDB (Paxos-replicated RocksDB), Harp (VR-replicated FS)

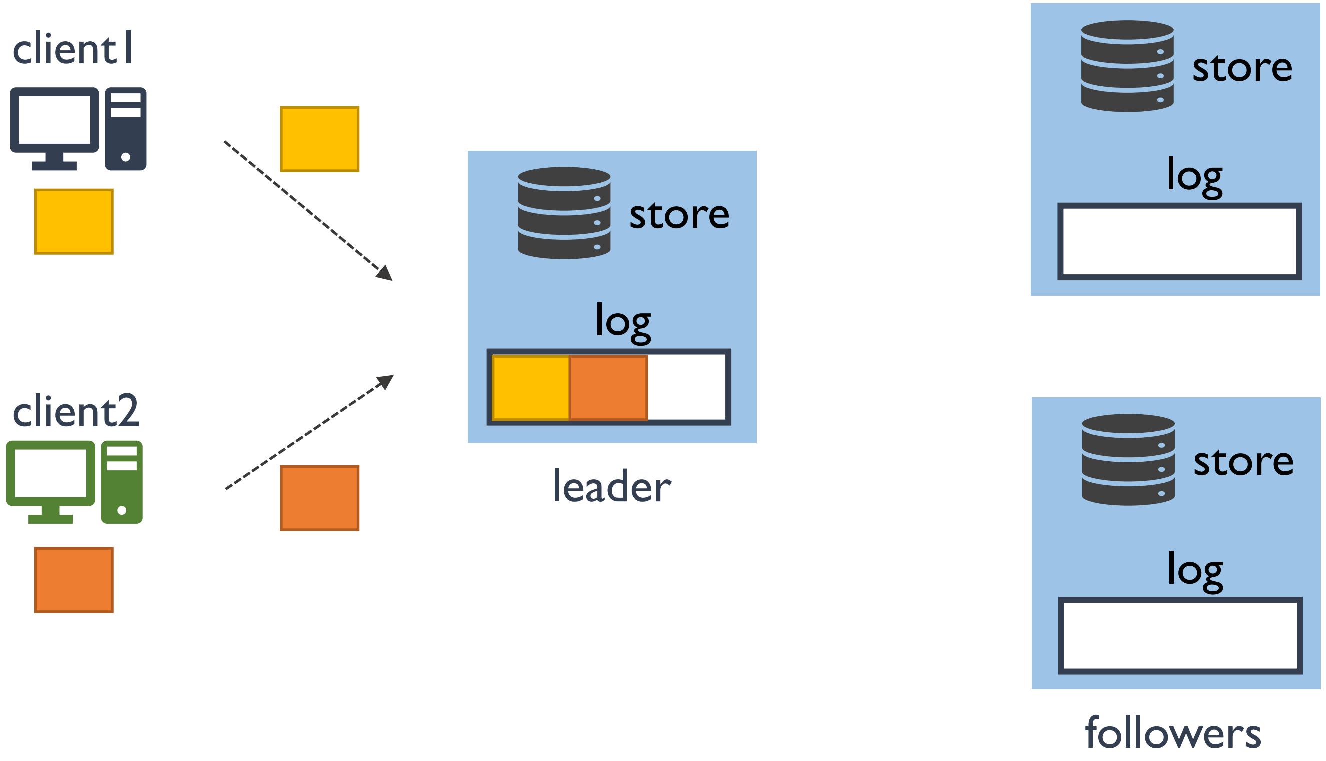
Ordering is Expensive

Several steps to update replicated data



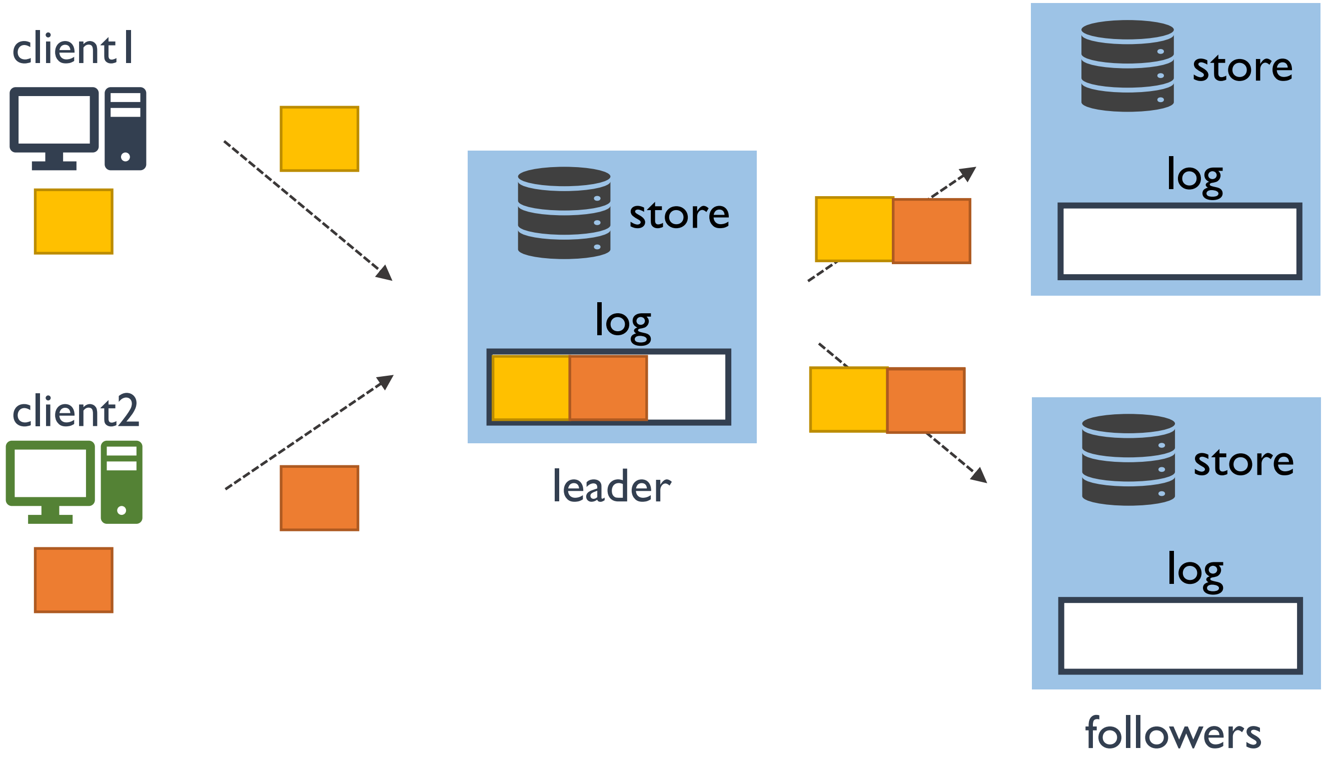
Ordering is Expensive

Several steps to update replicated data



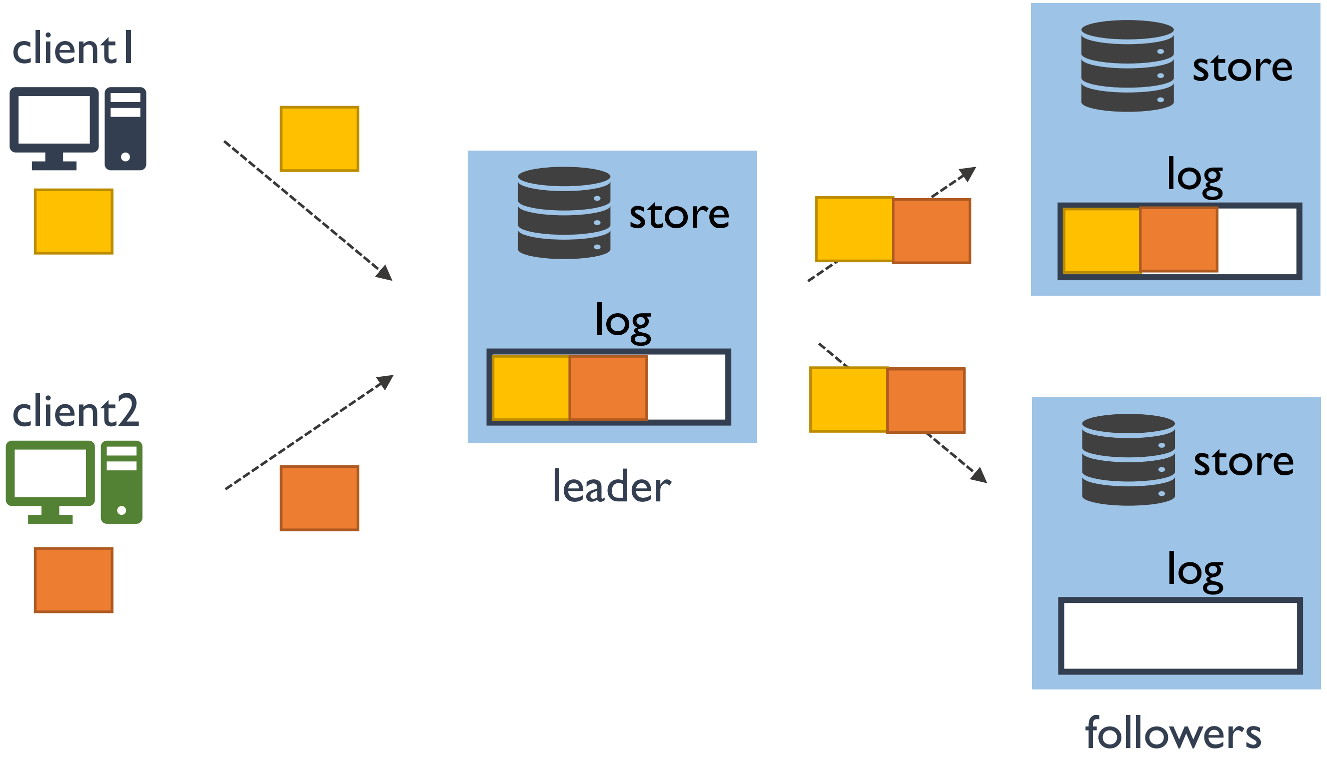
Ordering is Expensive

Several steps to update replicated data



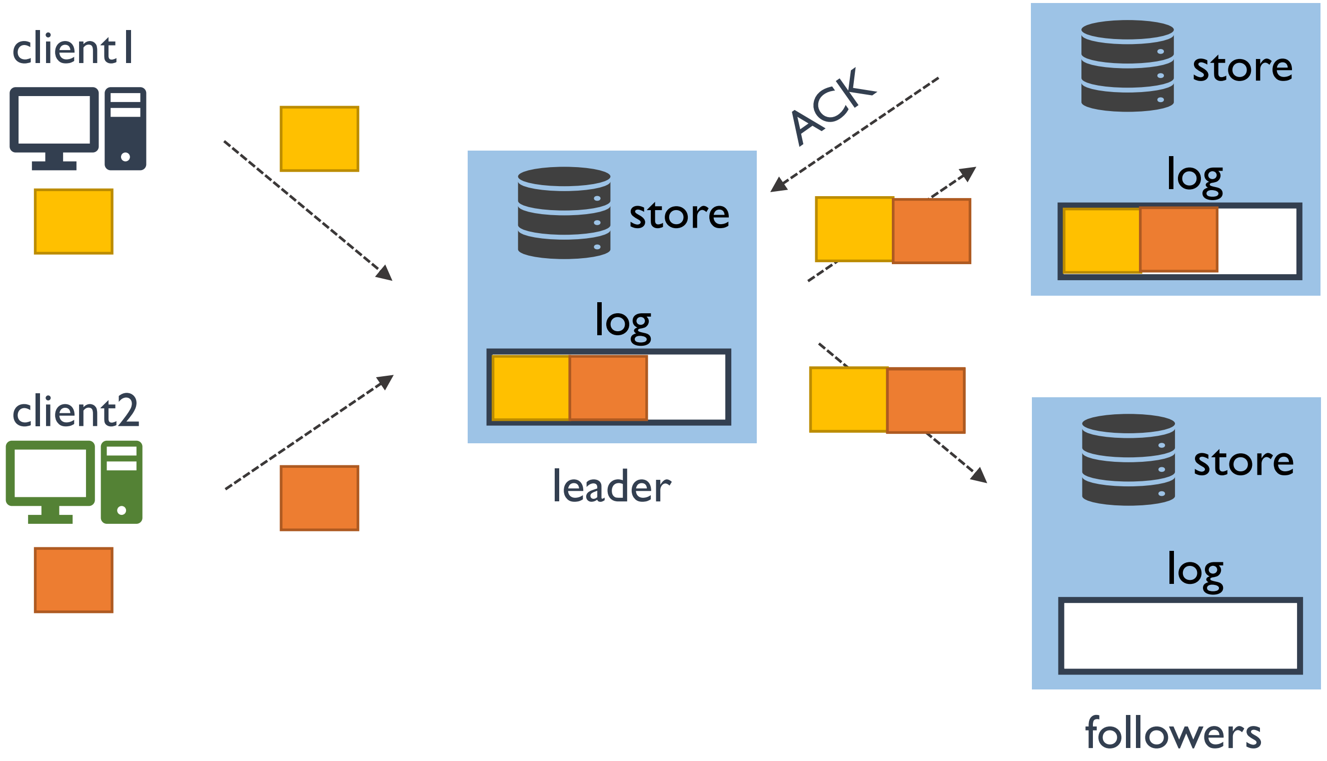
Ordering is Expensive

Several steps to update replicated data



Ordering is Expensive

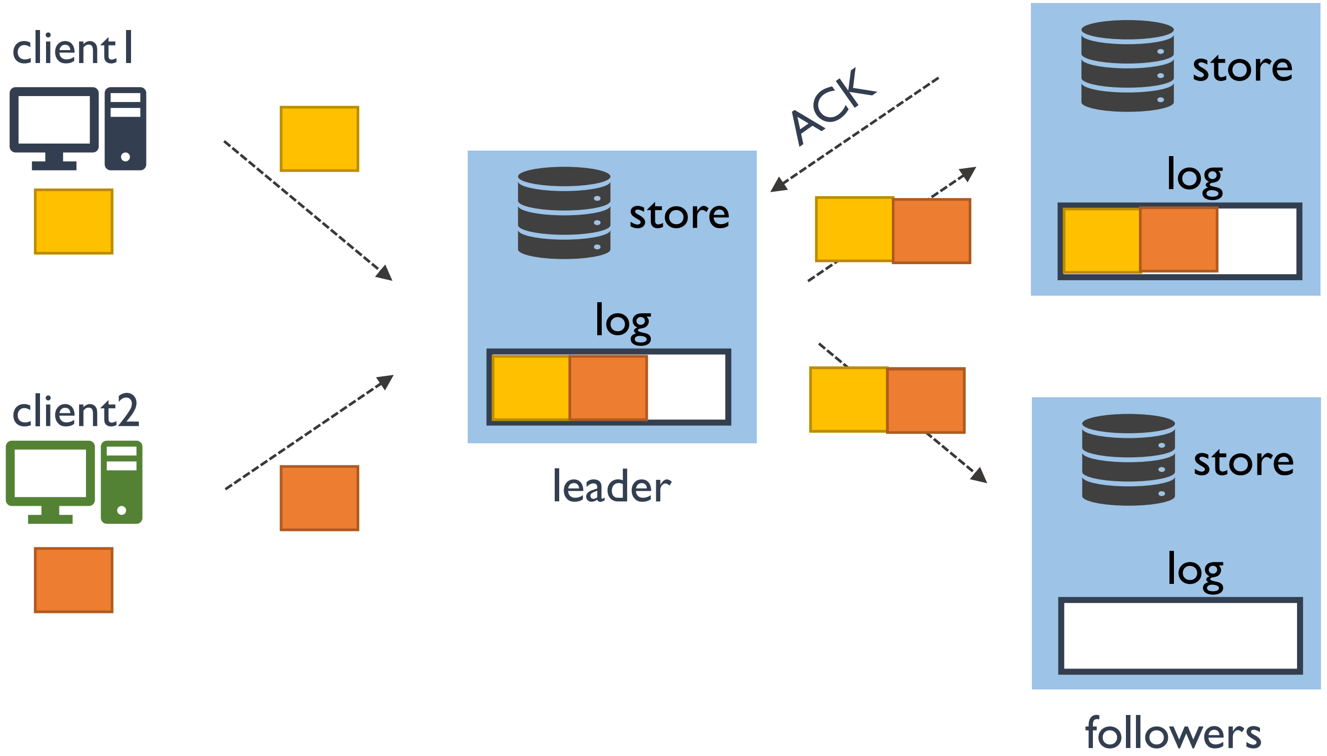
Several steps to update replicated data



Ordering is Expensive

Several steps to update replicated data

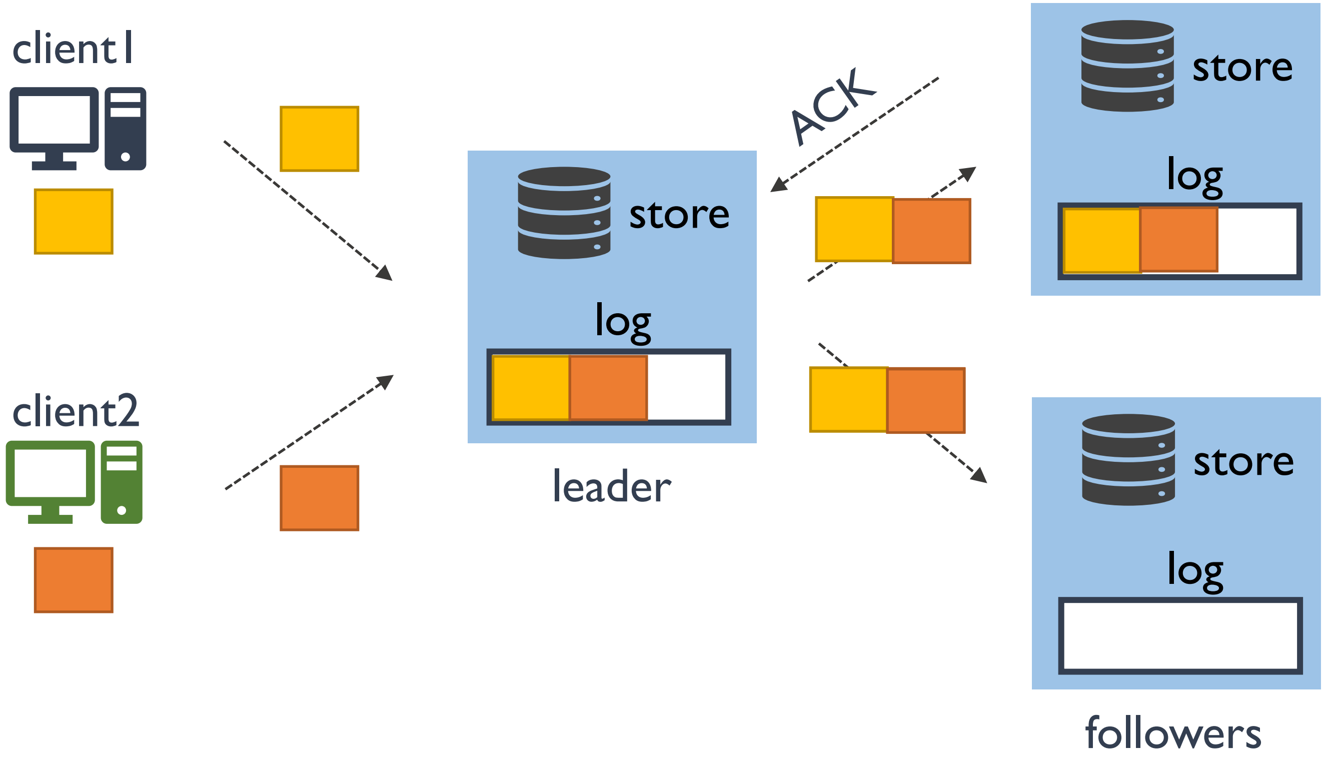
durability: update will not be lost once majority ack



Ordering is Expensive

Several steps to update replicated data

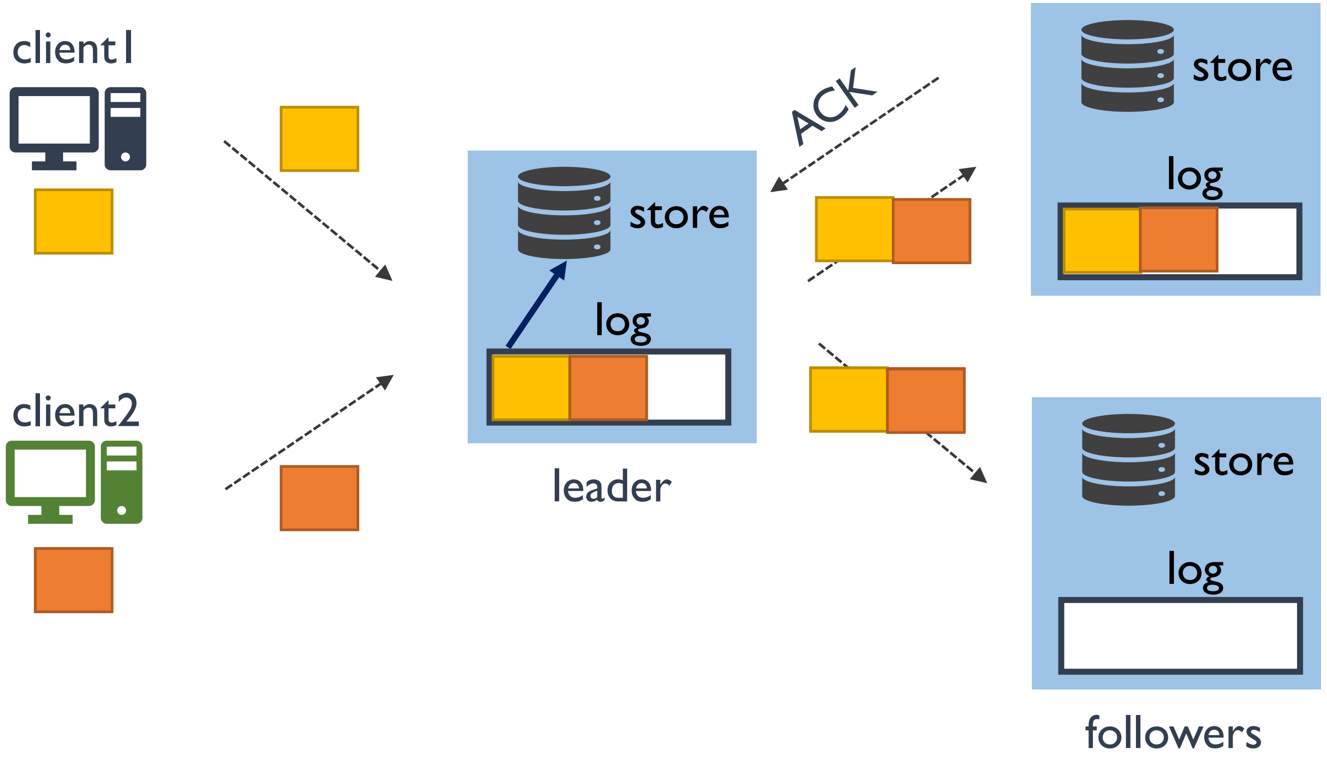
- durability: update will not be lost once majority ack
- ordering: update order agreed upon by replicas



Ordering is Expensive

Several steps to update replicated data

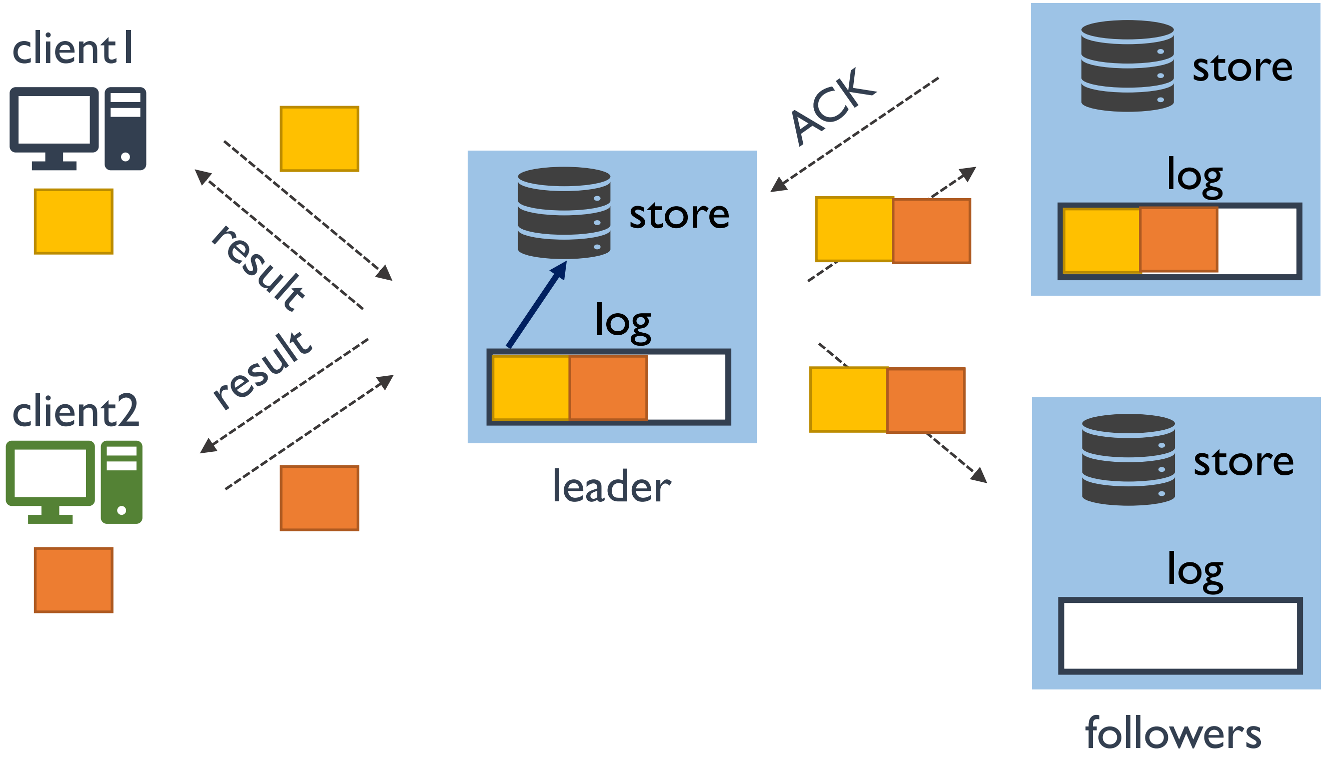
- durability: update will not be lost once majority ack
- ordering: update order agreed upon by replicas
- execution: apply updates to store



Ordering is Expensive

Several steps to update replicated data

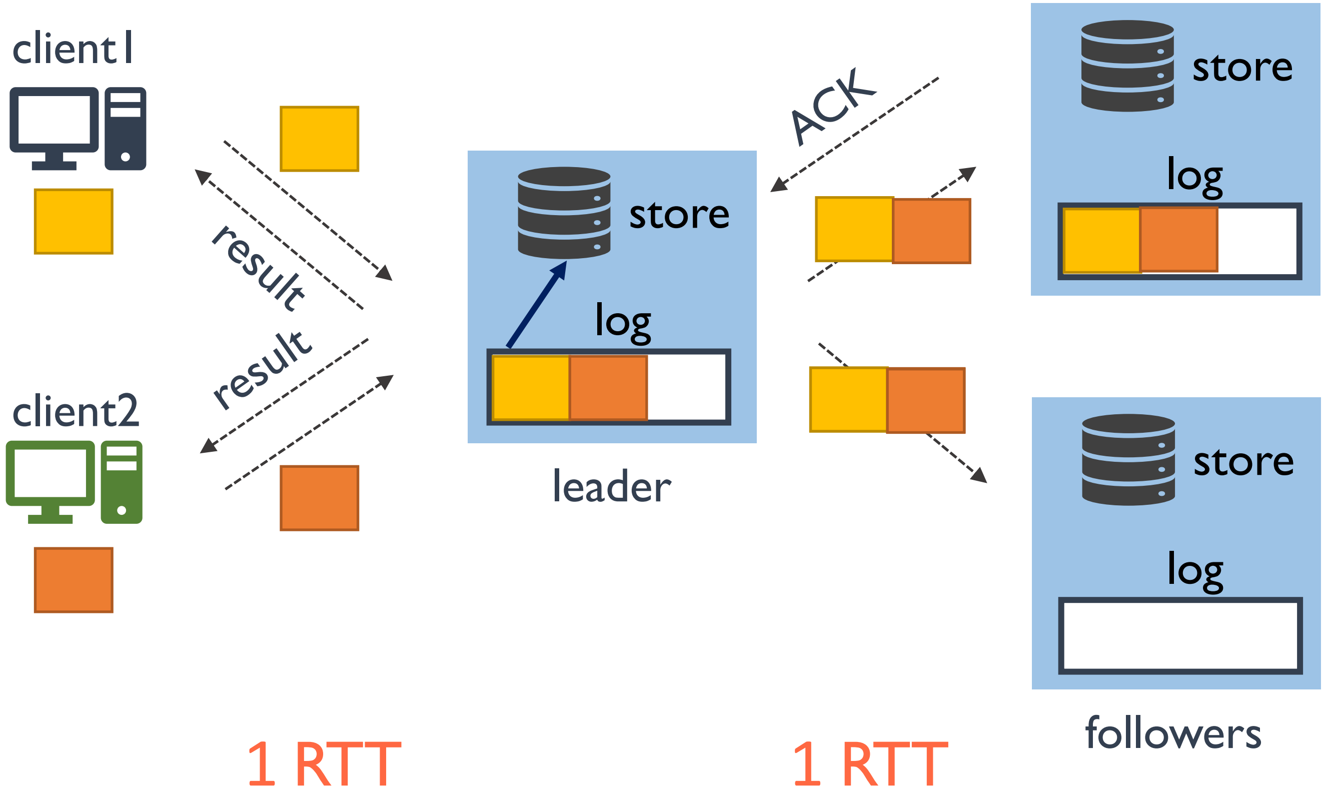
- durability: update will not be lost once majority ack
- ordering: update order agreed upon by replicas
- execution: apply updates to store



Ordering is Expensive

Several steps to update replicated data

- durability: update will not be lost once majority ack
- ordering: update order agreed upon by replicas
- execution: apply updates to store



Multi round-trip agreement

Network roundtrips critical for application performance

Outline

Introduction

Strongly consistent storage background

➡ Nilext-aware replication

Evaluation

Nilext Interfaces

A nil-externalizing or nilext interface

- may modify state in any way: blind write, or read-modify-write
- does not externalize storage-system state
  - does not return an execution result or an execution error
- usually returns an ack

Example: `Put` interface in KV API

- does not return execution result (only an ack)
- does not return execution error (e.g., by checking if key is already present)

Nilext Interfaces are Prevalent

All updates are nilext in key-value stores (e.g., RocksDB, LevelDB) built upon write-optimized structures such as LSM and B^e-trees

Interface	Nilext?	
Put, Write(multi-put)	Yes	No error if key(s) already present
Delete	Yes	No error if key absent – insert tombstone
Merge (RMW)	Yes	Not applied immediately – insert message specifying how to modify value
Get	No	Returns value or error

avoid query before update

[Bender et al., 2015]

Some systems have a mix of nilext and non-nilext interfaces (e.g., Memcached)

Real-world traces show most updates are nilext

- 90% updates are nilext in 80% clusters (Twemcache production traces)
- more analysis in the paper …

Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

store
store
store

Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

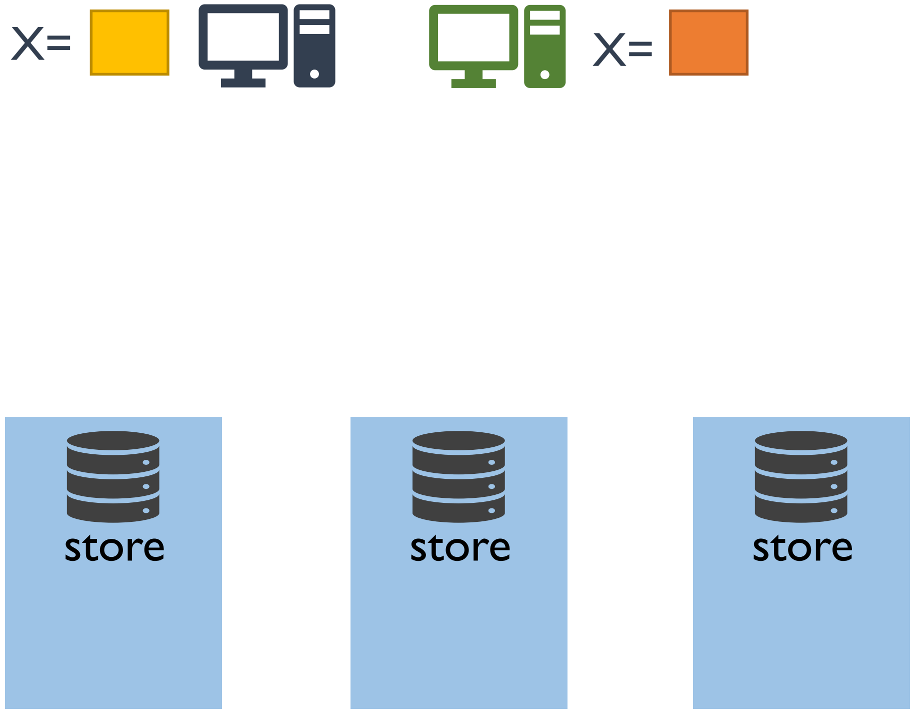
1 Durability without coordination

Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1 Durability without coordination

clients send directly to replicas

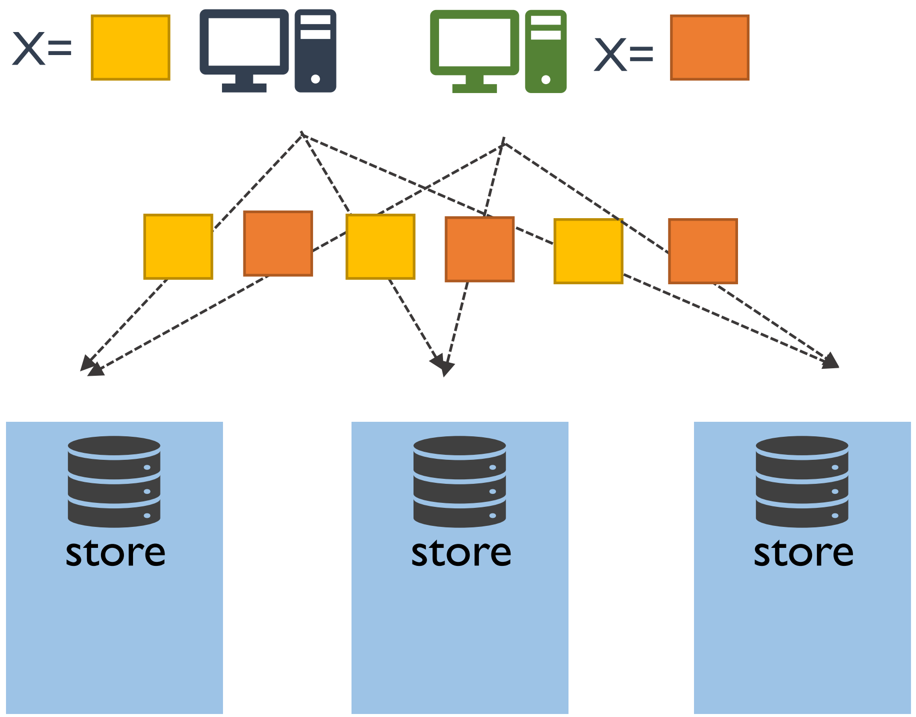


Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1 Durability without coordination

clients send directly to replicas

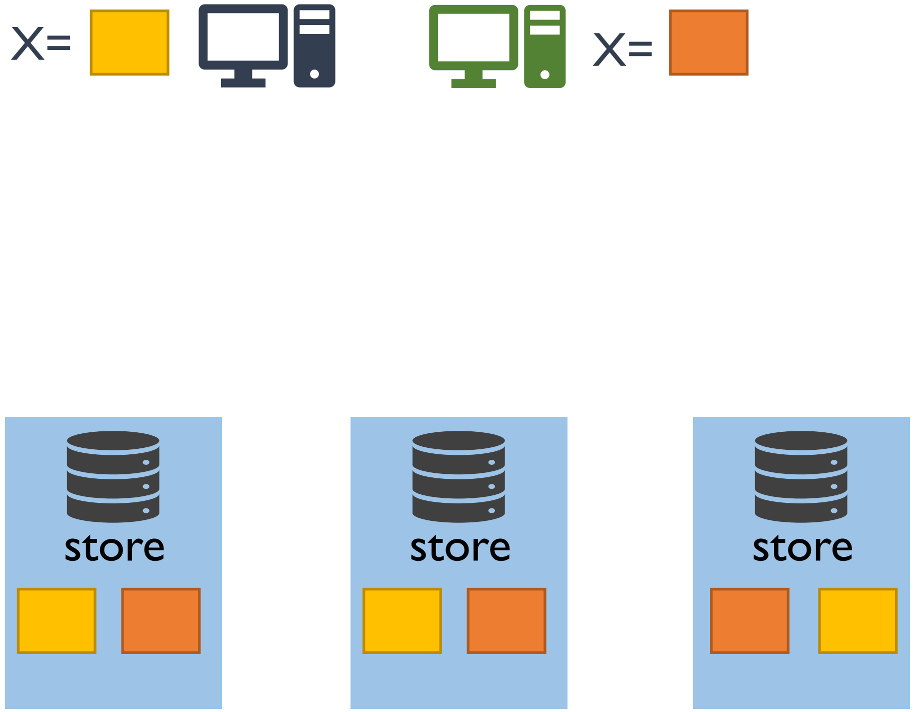


Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1 Durability without coordination

clients send directly to replicas



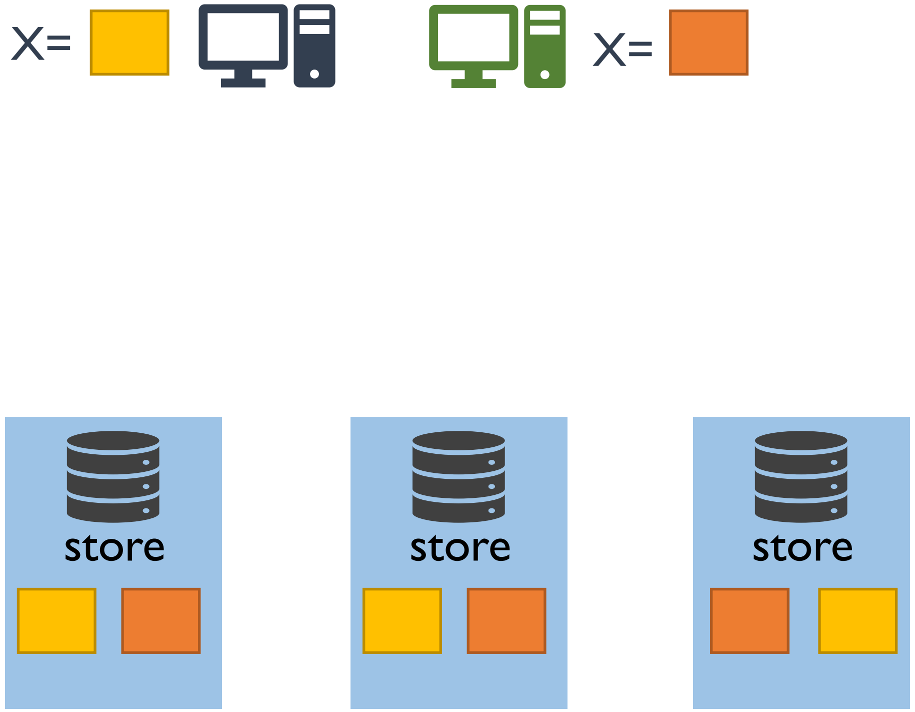
Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1 Durability without coordination

clients send directly to replicas

2 Defer ordering (and execution) if nilext



Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

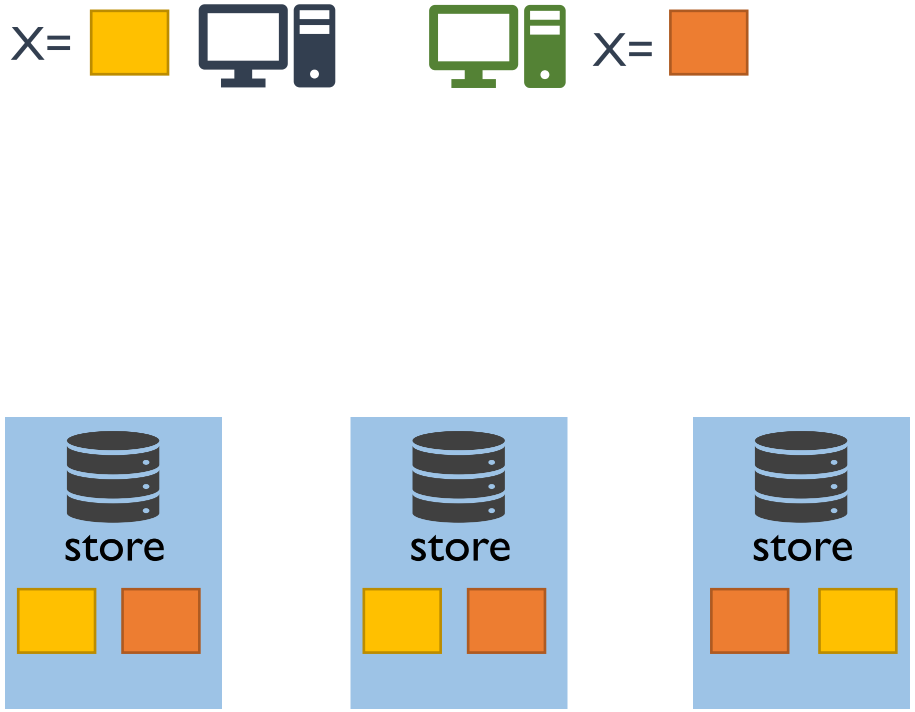
1 Durability without coordination

clients send directly to replicas

2 Defer ordering (and execution) if nilext

nilext update does not externalize state

defer nilext update → 1 RTT completion



Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

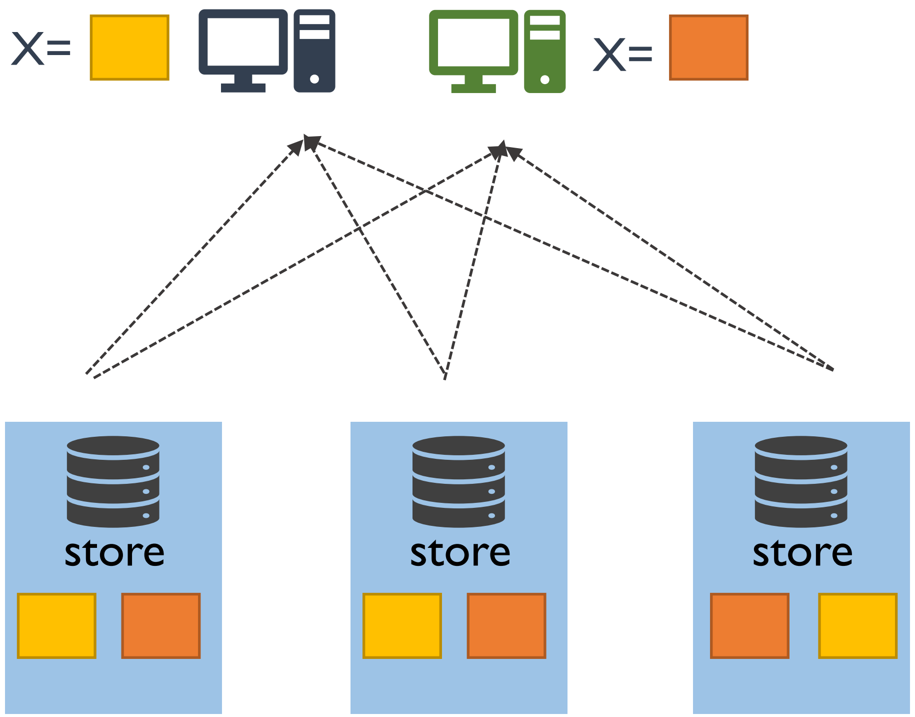
1. Durability without coordination

   clients send directly to replicas

2. Defer ordering (and execution) if nilext

   nilext update does not externalize state

   defer nilext update → 1 RTT completion



Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1. Durability without coordination

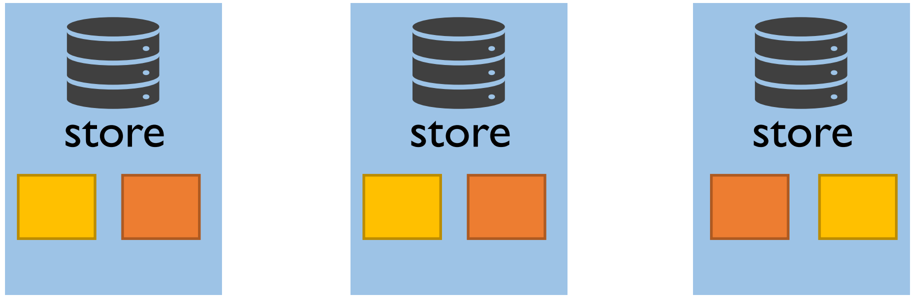
   clients send directly to replicas

2. Defer ordering (and execution) if nilext

   nilext update does not externalize state

   defer nilext update → 1 RTT completion

3. Non-nilext operations externalize state



Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1. Durability without coordination

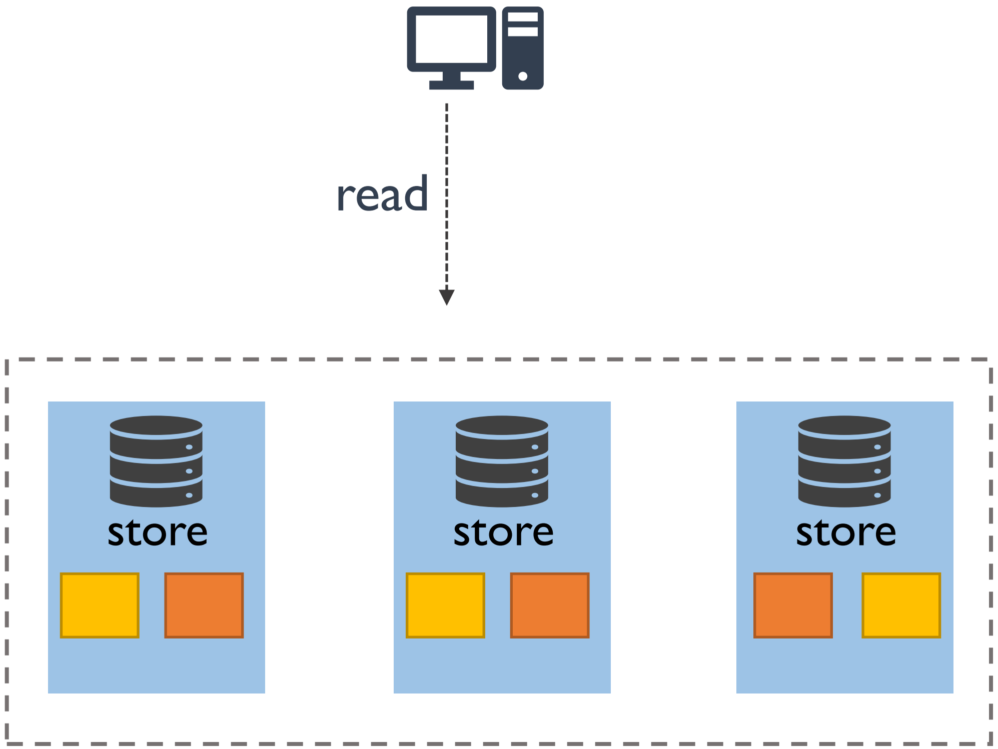
   clients send directly to replicas

2. Defer ordering (and execution) if nilext

   nilext update does not externalize state

   defer nilext update → 1 RTT completion

3. Non-nilext operations externalize state



Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1. Durability without coordination

   clients send directly to replicas

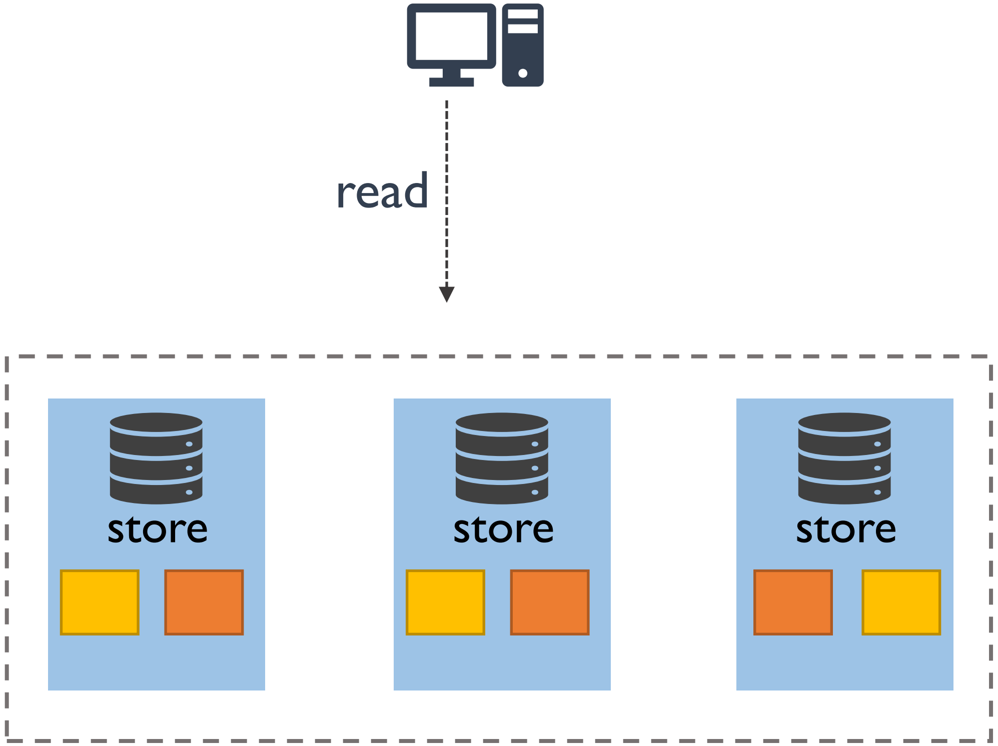
2. Defer ordering (and execution) if nilext

   nilext update does not externalize state

   defer nilext update → 1 RTT completion

3. Non-nilext operations externalize state

   enforce ordering and execution before state is externalized → strong consistency



Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1 Durability without coordination

clients send directly to replicas

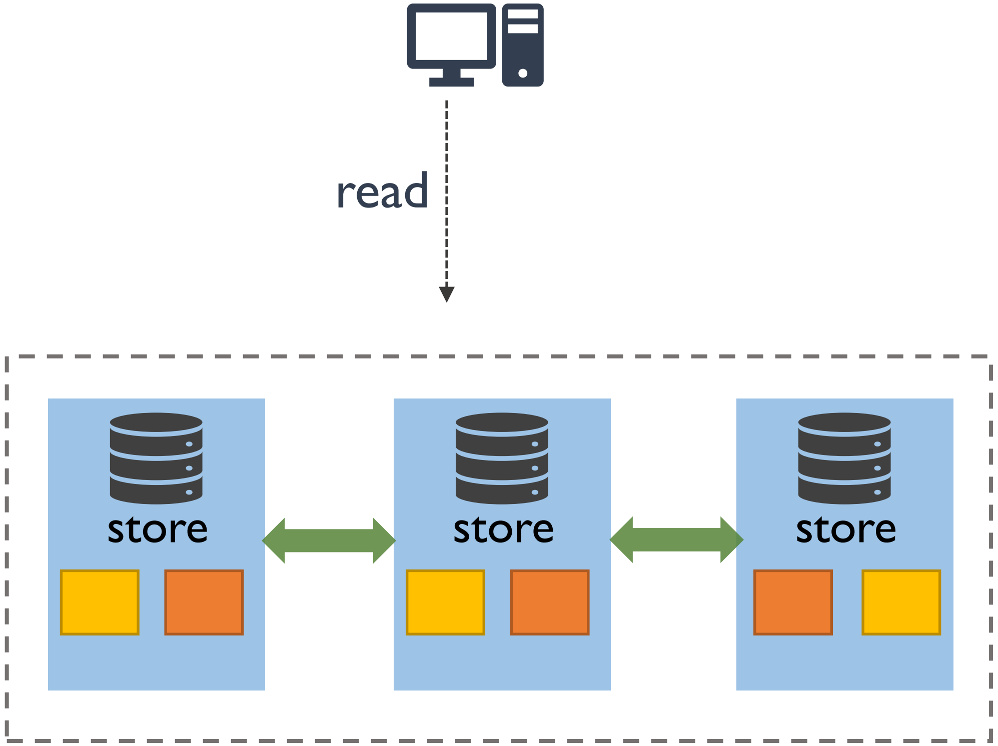
2 Defer ordering (and execution) if nilext

nilext update does not externalize state

defer nilext update → 1 RTT completion

3 Non-nilext operations externalize state

enforce ordering and execution before state is externalized → strong consistency



Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1 Durability without coordination

clients send directly to replicas

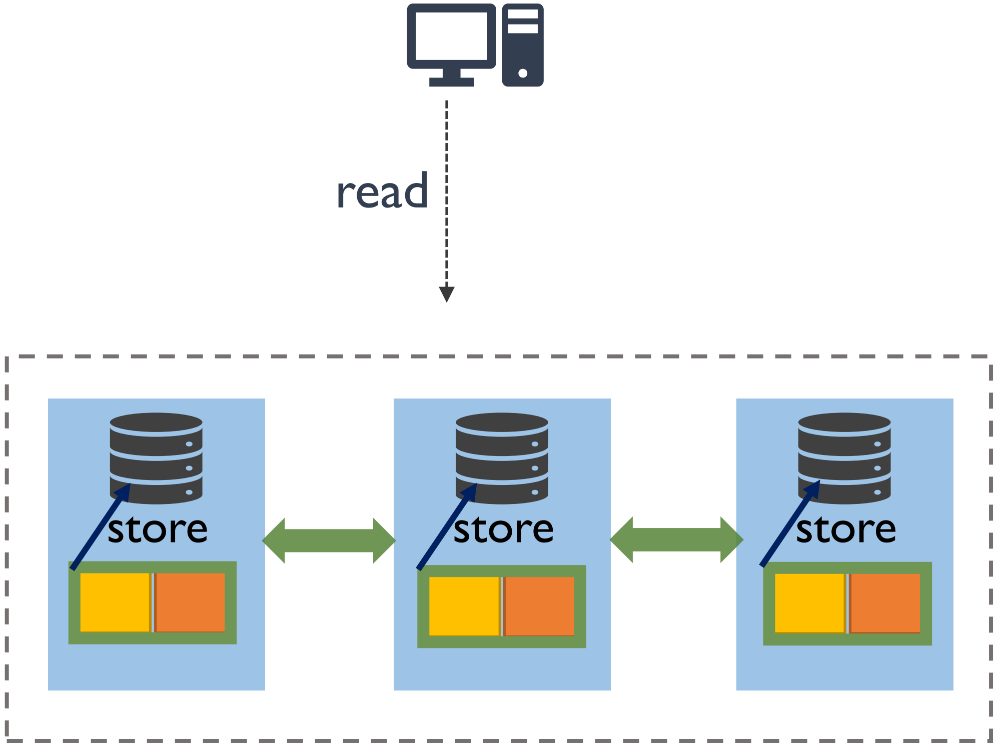
2 Defer ordering (and execution) if nilext

nilext update does not externalize state

defer nilext update → 1 RTT completion

3 Non-nilext operations externalize state

enforce ordering and execution before state is externalized → strong consistency



Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1. Durability without coordination

   clients send directly to replicas

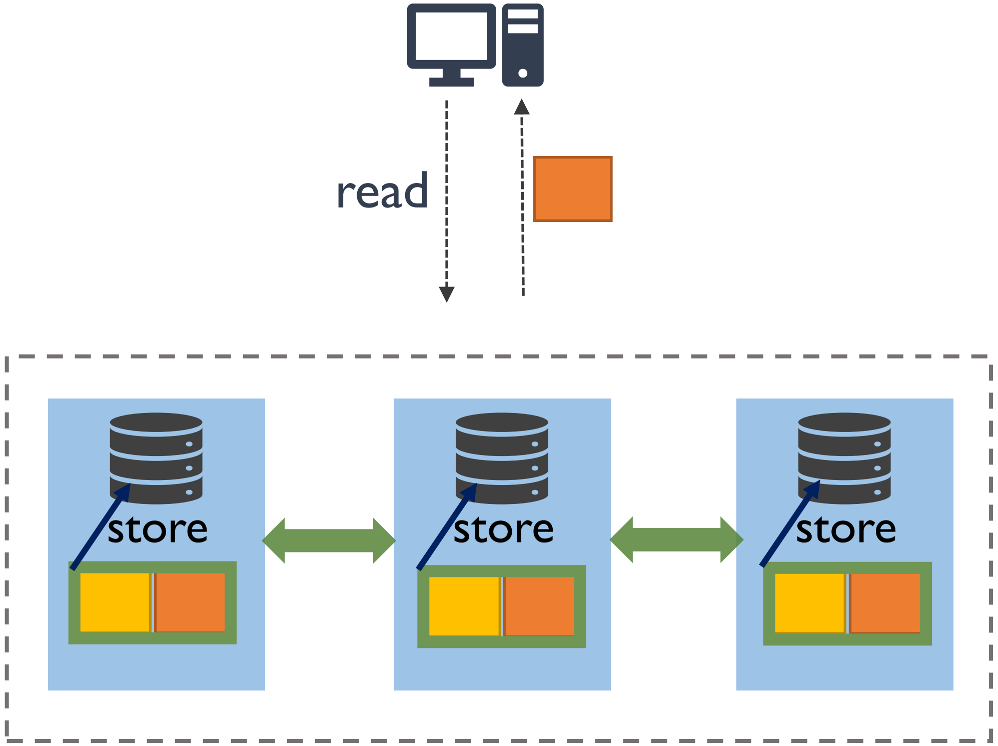
2. Defer ordering (and execution) if nilext

   nilext update does not externalize state

   defer nilext update → 1 RTT completion

3. Non-nilext operations externalize state

   enforce ordering and execution before state is externalized → strong consistency



Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1. Durability without coordination

   clients send directly to replicas

2. Defer ordering (and execution) if nilext

   nilext update does not externalize state

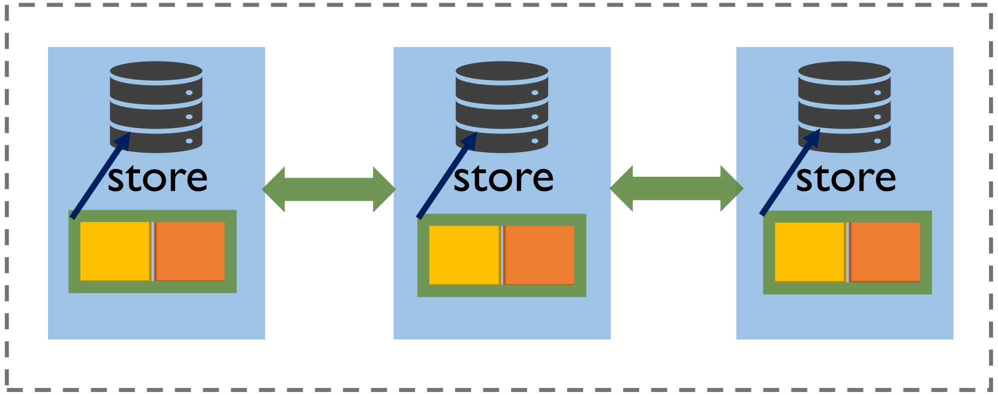
   defer nilext update → 1 RTT completion

3. Non-nilext operations externalize state

   enforce ordering and execution before state is externalized → strong consistency

   order and execute in the background

   common case: updates already executed



Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1 Durability without coordination

clients send directly to replicas

2 Defer ordering (and execution) if nilext

nilext update does not externalize state

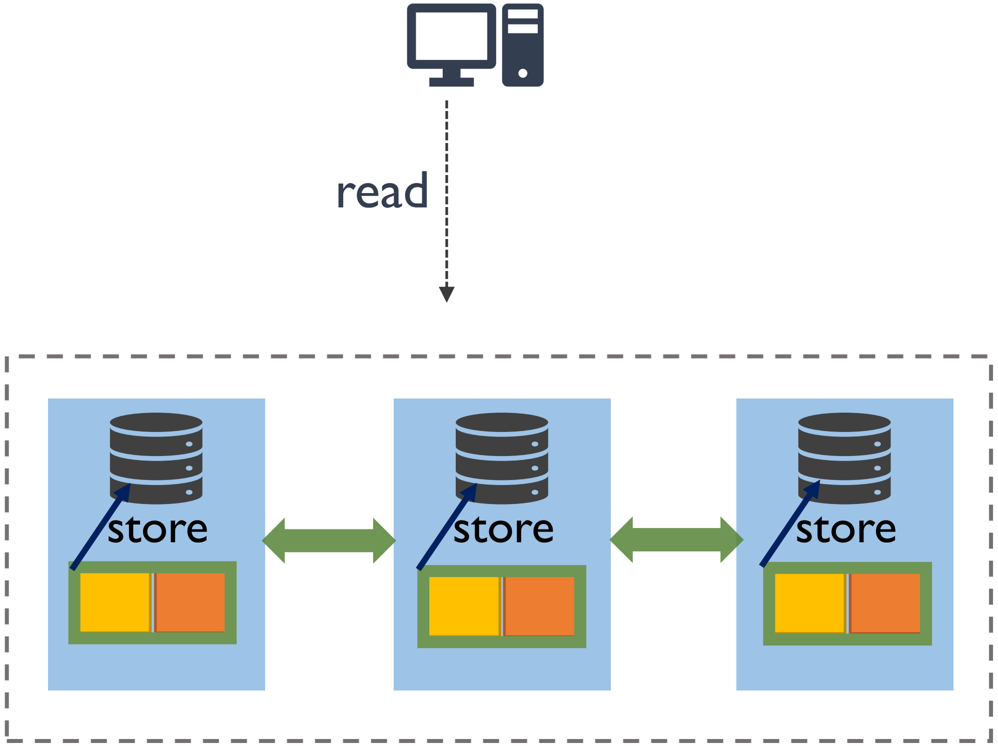
defer nilext update → 1 RTT completion

3 Non-nilext operations externalize state

enforce ordering and execution before state is externalized → strong consistency

order and execute in the background

common case: updates already executed



Exploiting Nil-Externality for Replication: Insights

Problem: coordination for ordering incurs multiple RTTs

1. Durability without coordination

   clients send directly to replicas

2. Defer ordering (and execution) if nilext

   nilext update does not externalize state

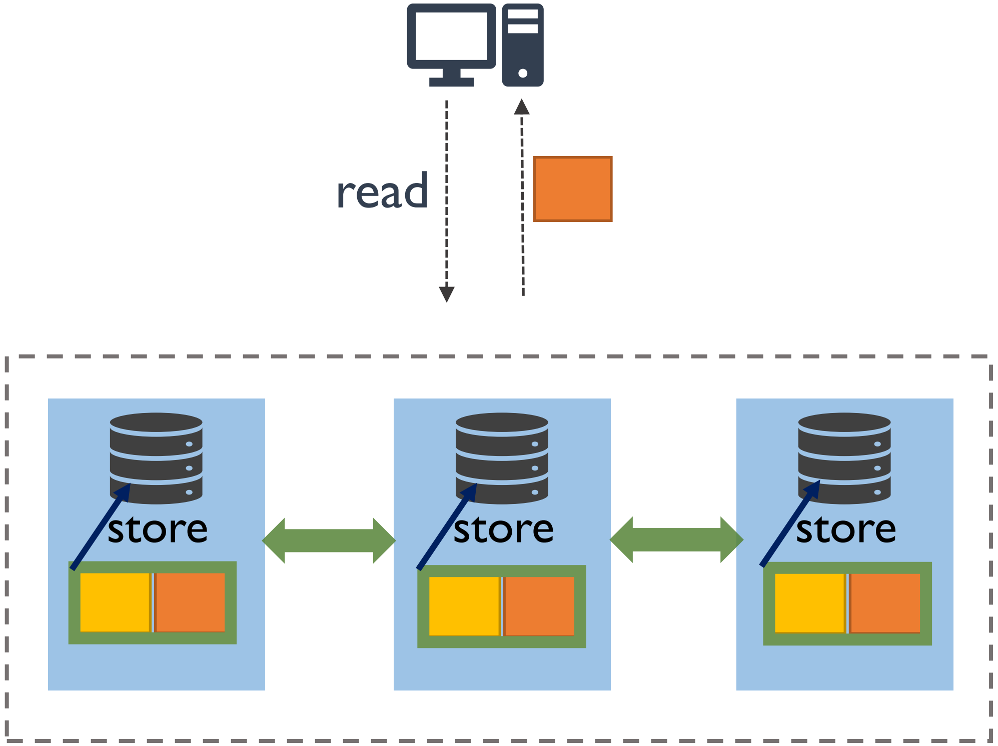
   defer nilext update → 1 RTT completion

3. Non-nilext operations externalize state

   enforce ordering and execution before state is externalized → strong consistency

   order and execute in the background

   common case: updates already executed



Deferring Work in Other Contexts

Defer work until observed has proven beneficial in other contexts

- programming languages [Henderson and Morris, 1976] [Friedman and Wise, 1976]
- file systems [Nightingale et al., 2006]
- databases [Faleiro et al., 2014]

Our work:

- applies this general idea in the context of replication
- identifies an interface-level property in storage systems that enables deferring work

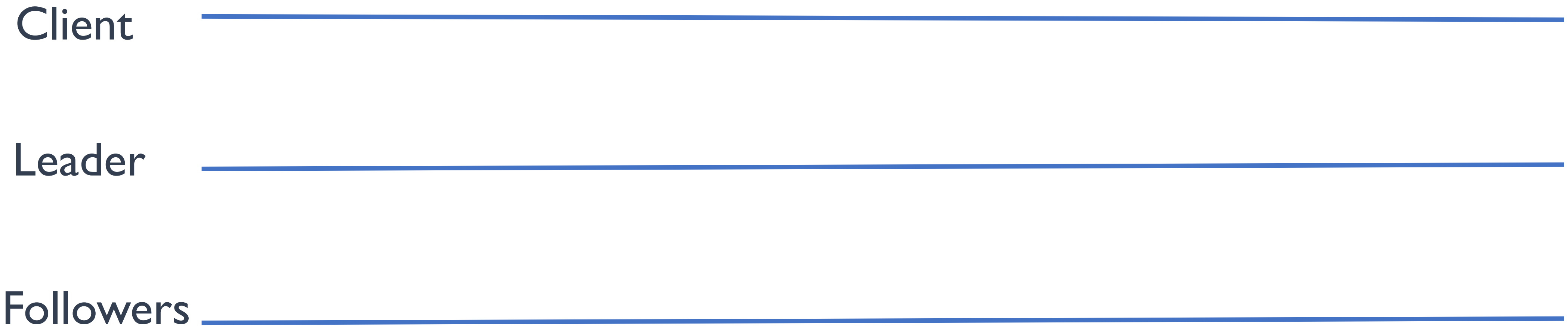
Skyros

Skyros is a new nilext-aware replication protocol

Based on viewstamped replication (VR) [Oki and Liskov, 1988] [Liskov and Cowling, 2012]

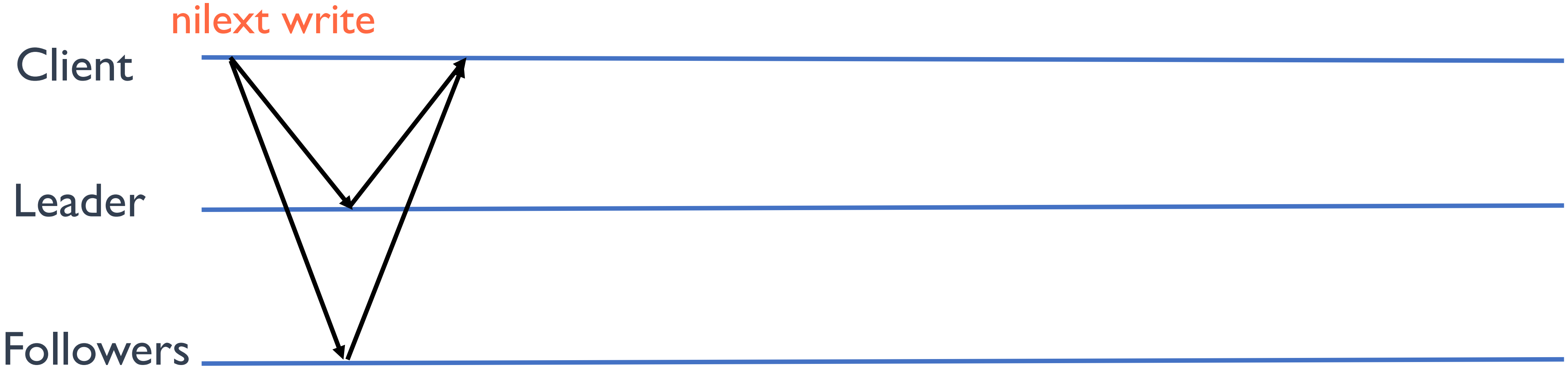
- leader based
- provides linearizability
- available when majority replicas alive

Skyros Overview



Skyros Overview

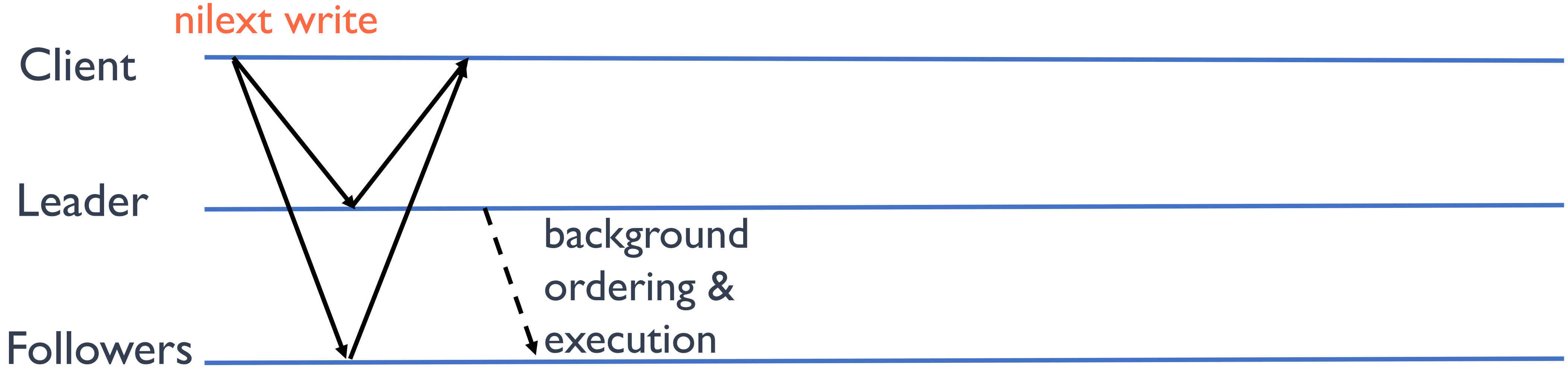
Nilext updates: clients write to replicas directly and make durable in 1 RTT



Skyros Overview

Nilext updates: clients write to replicas directly and make durable in 1 RTT

leader orders and executes in background

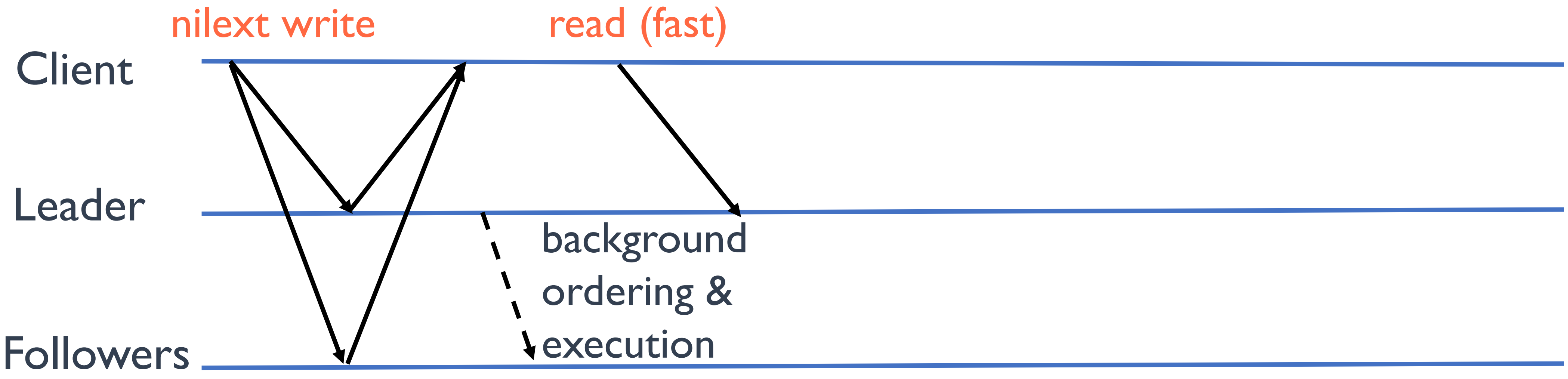


Skyros Overview

Nilext updates: clients write to replicas directly and make durable in 1 RTT

leader orders and executes in background

Reads: at leader; mostly 1 RTT

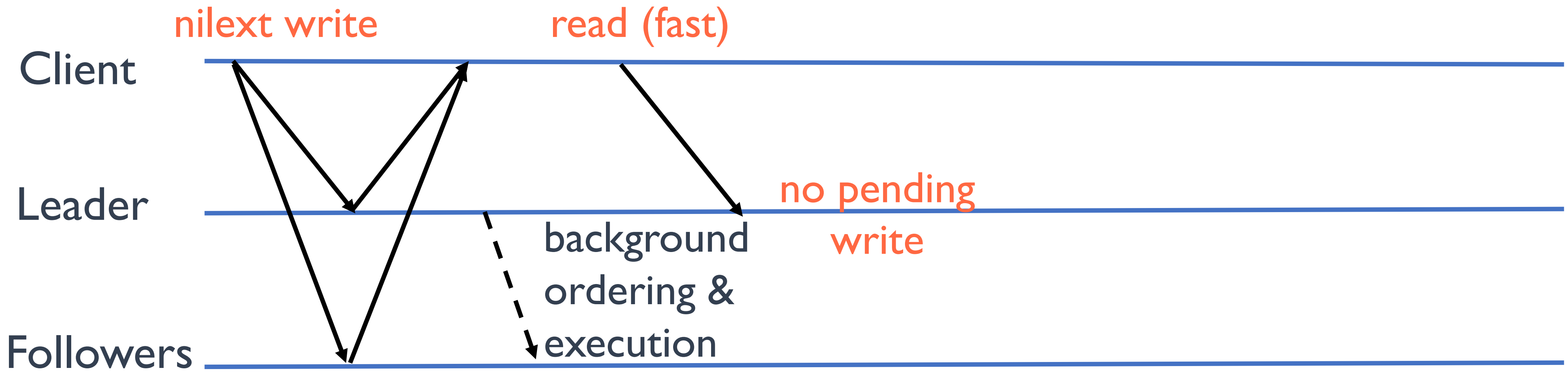


Skyros Overview

Nilext updates: clients write to replicas directly and make durable in 1 RTT

leader orders and executes in background

Reads: at leader; mostly 1 RTT

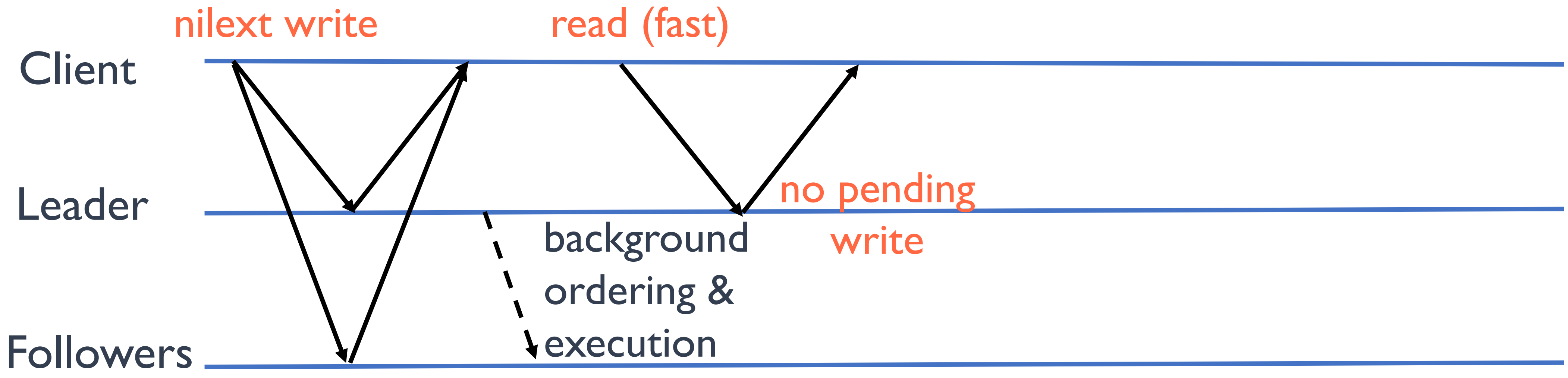


Skyros Overview

Nilext updates: clients write to replicas directly and make durable in 1 RTT

leader orders and executes in background

Reads: at leader; mostly 1 RTT

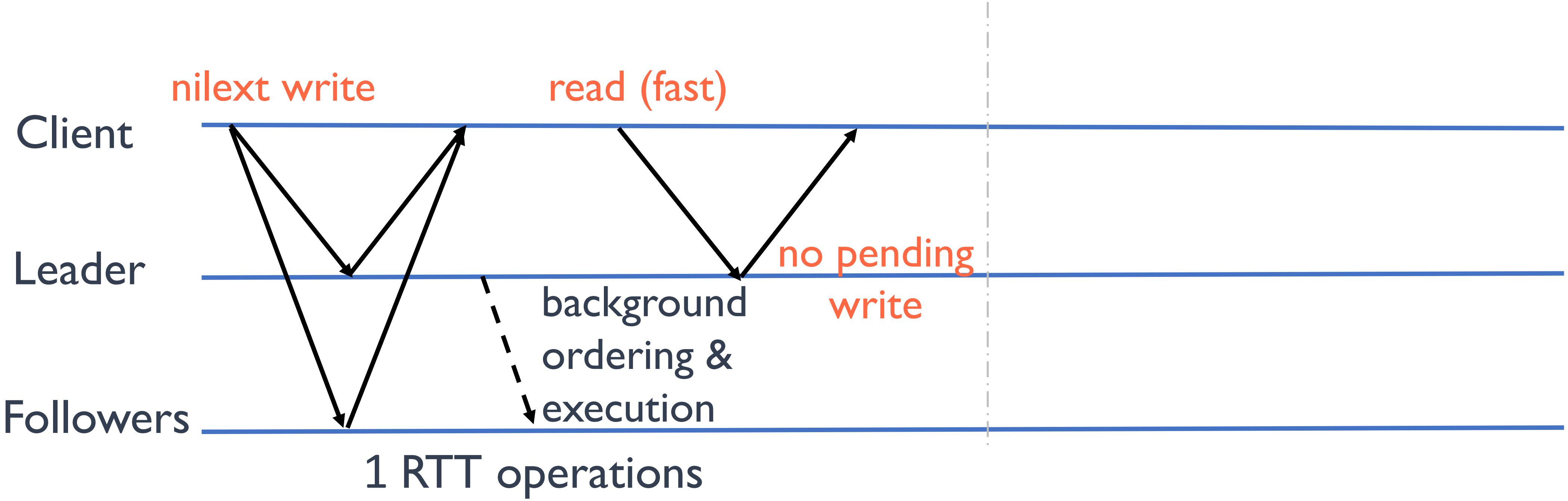


Skyros Overview

Nilext updates: clients write to replicas directly and make durable in 1 RTT

leader orders and executes in background

Reads: at leader; mostly 1 RTT



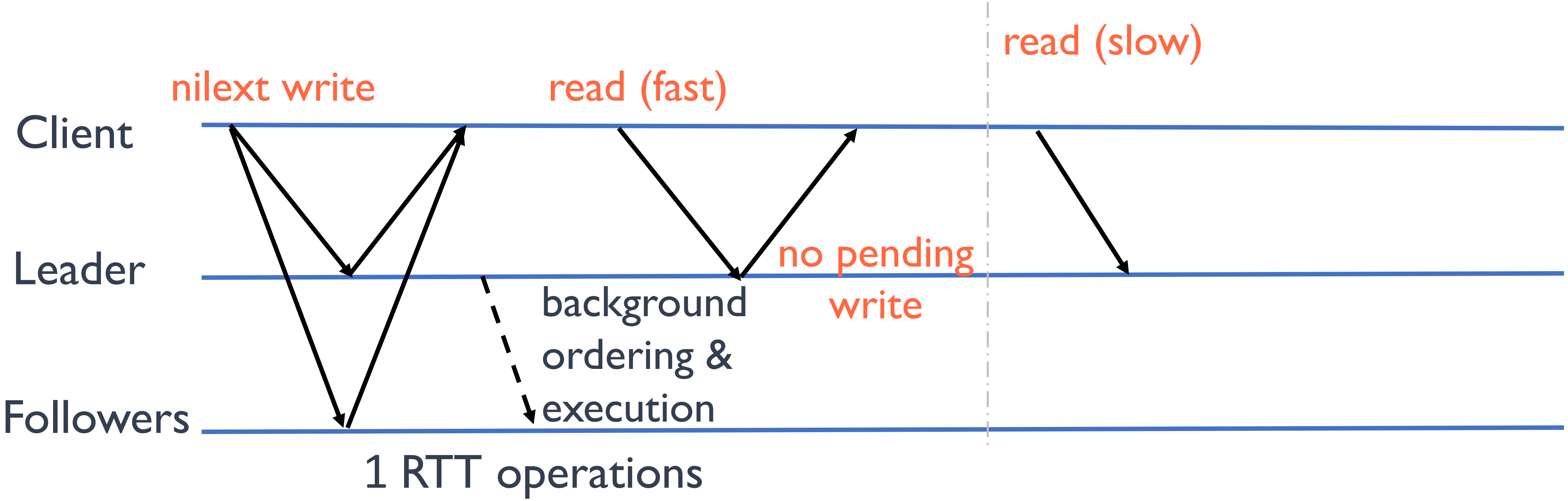
Skyros Overview

Nilext updates: clients write to replicas directly and make durable in 1 RTT

leader orders and executes in background

Reads: at leader; mostly 1 RTT

sometimes 2 RTTs (when they read not-yet ordered updates)



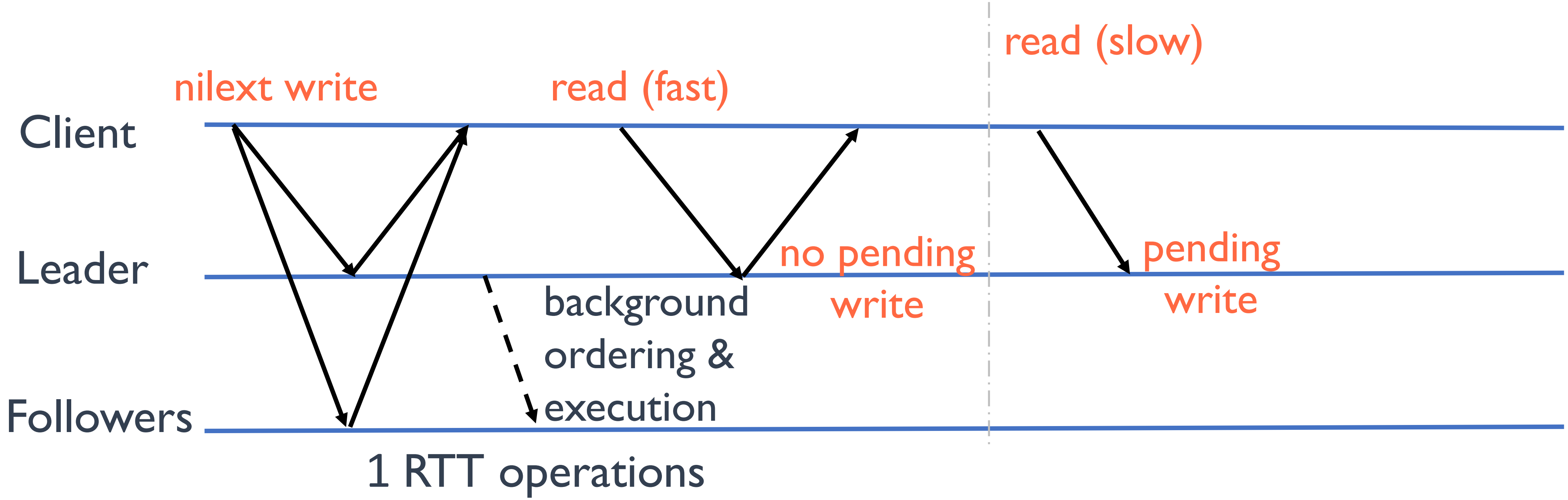
Skyros Overview

Nilext updates: clients write to replicas directly and make durable in 1 RTT

leader orders and executes in background

Reads: at leader; mostly 1 RTT

sometimes 2 RTTs (when they read not-yet ordered updates)



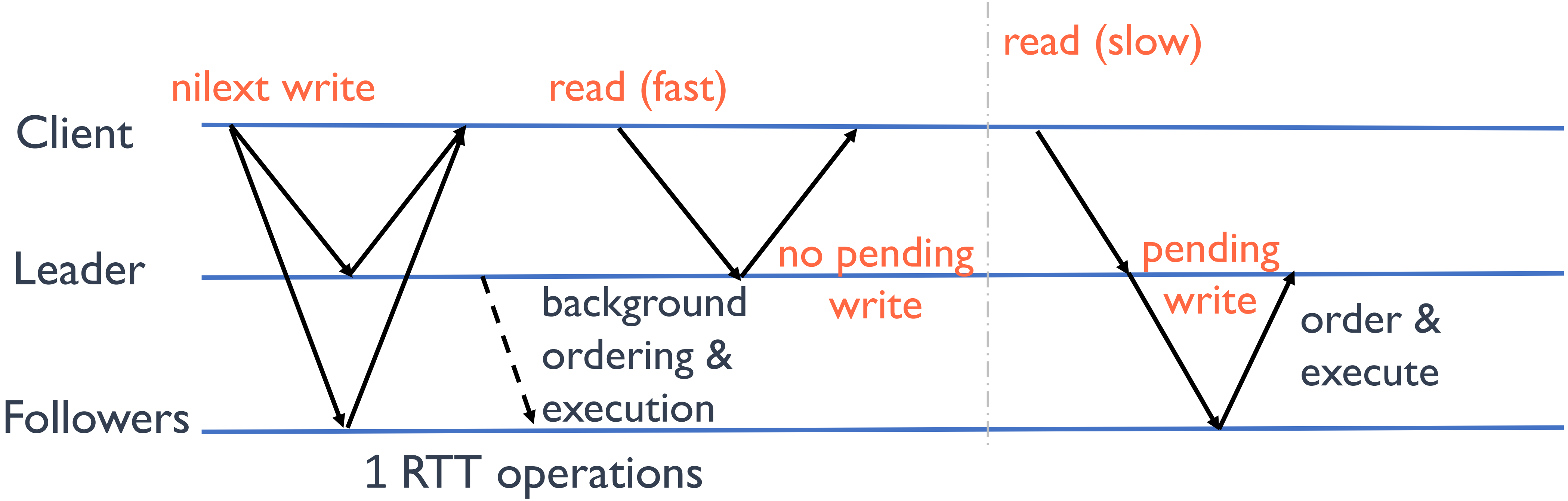
Skyros Overview

Nilext updates: clients write to replicas directly and make durable in 1 RTT

leader orders and executes in background

Reads: at leader; mostly 1 RTT

sometimes 2 RTTs (when they read not-yet ordered updates)



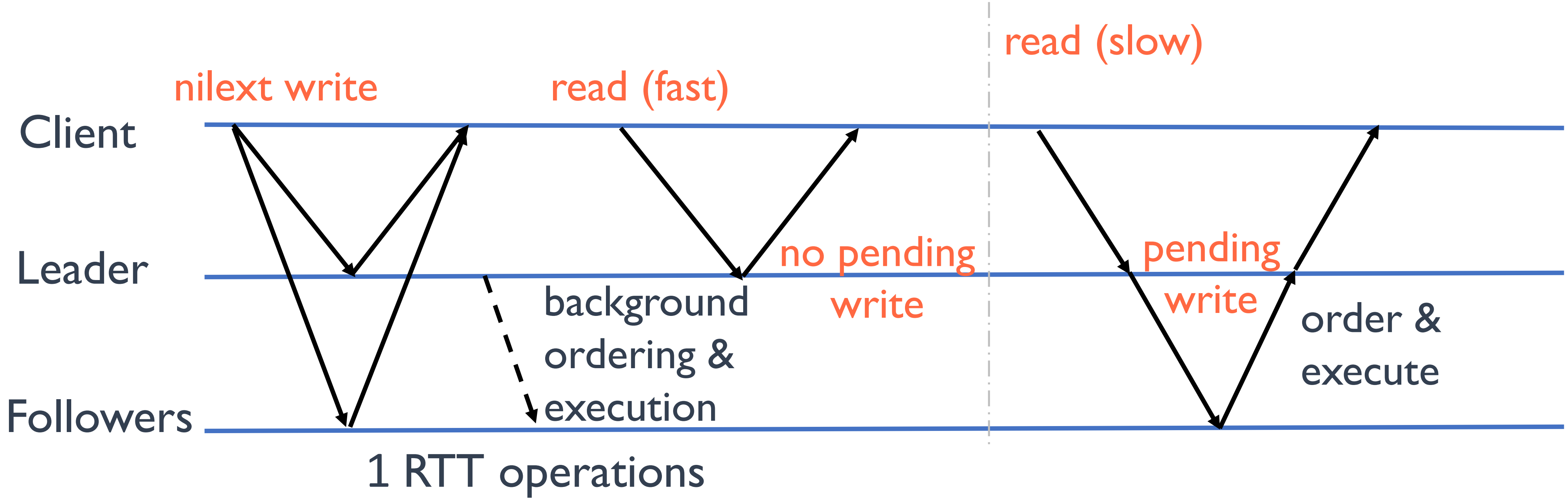
Skyros Overview

Nilext updates: clients write to replicas directly and make durable in 1 RTT

leader orders and executes in background

Reads: at leader; mostly 1 RTT

sometimes 2 RTTs (when they read not-yet ordered updates)



Skyros Overview

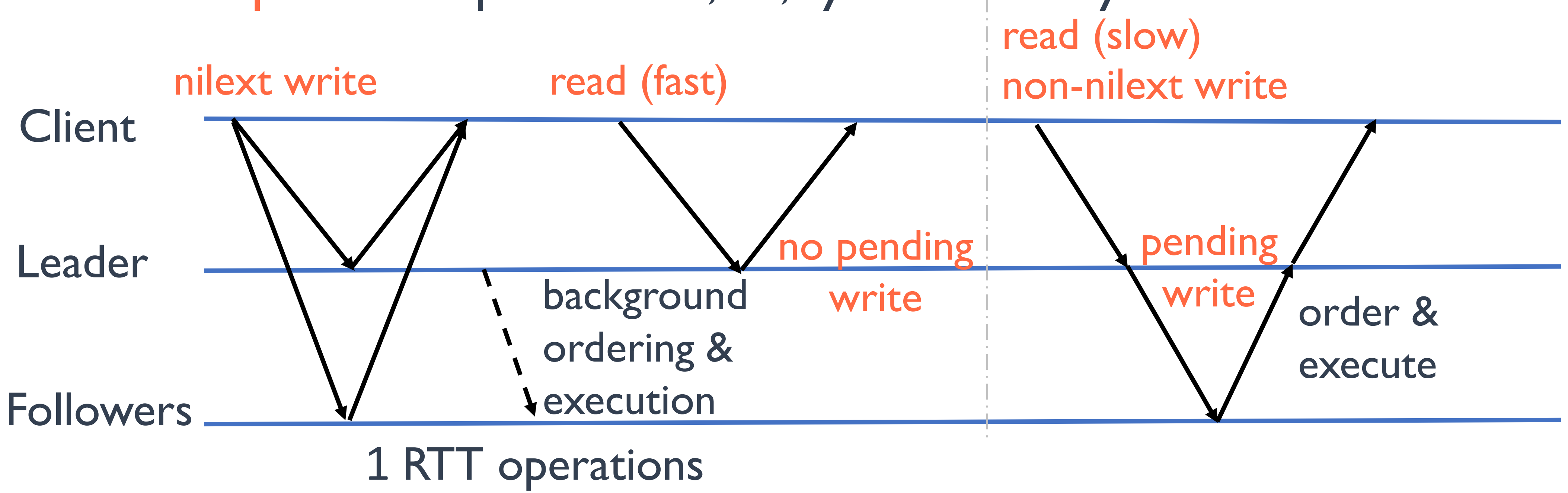
Nilext updates: clients write to replicas directly and make durable in 1 RTT

leader orders and executes in background

Reads: at leader; mostly 1 RTT

sometimes 2 RTTs (when they read not-yet ordered updates)

Non-nilext updates: expose state; so, synchronously order



Skyros Overview

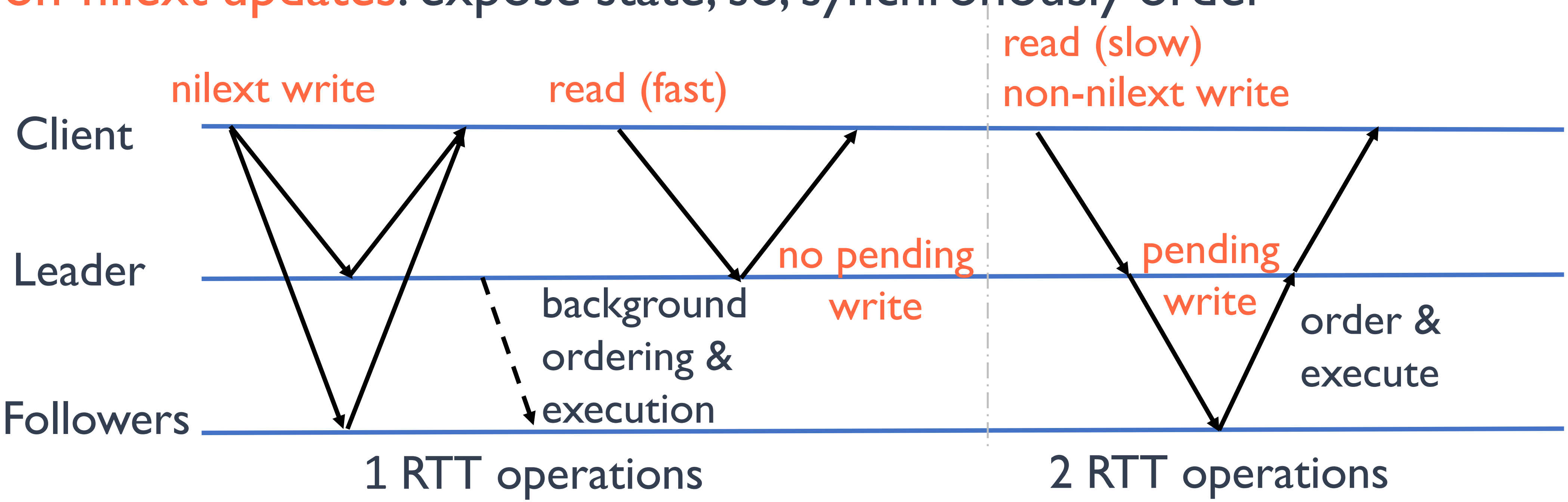
Nilext updates: clients write to replicas directly and make durable in 1 RTT

leader orders and executes in background

Reads: at leader; mostly 1 RTT

sometimes 2 RTTs (when they read not-yet ordered updates)

Non-nilext updates: expose state; so, synchronously order



Skyros Overview

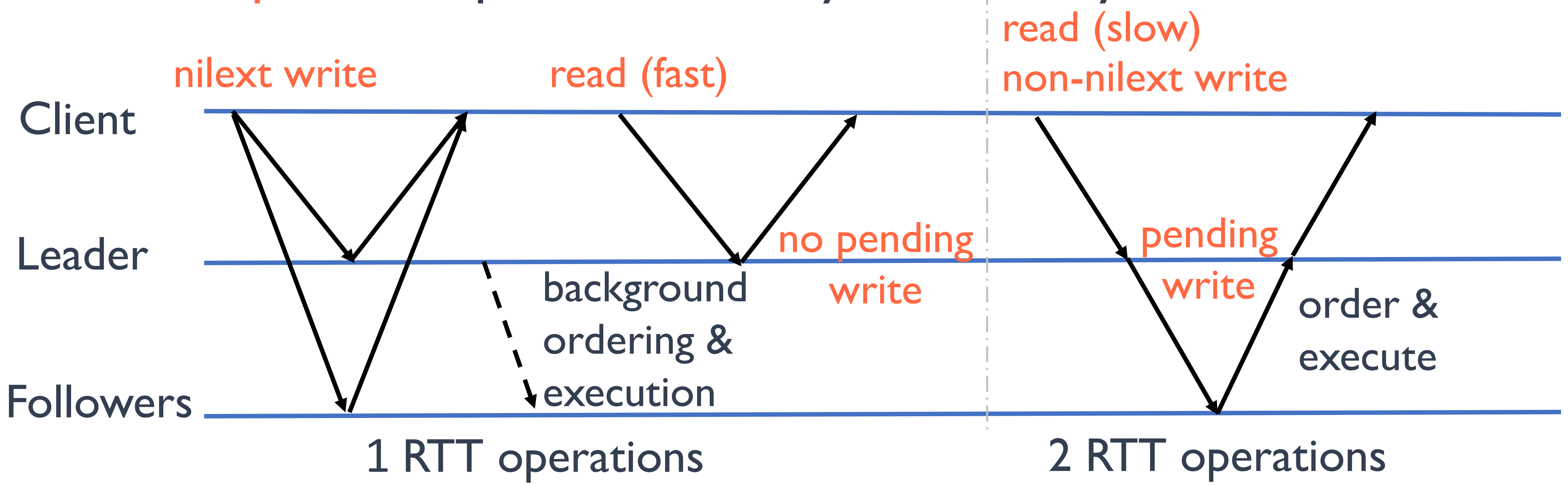
Nilext updates: clients write to replicas directly and make durable in 1 RTT

leader orders and executes in background

Reads: at leader; mostly 1 RTT

sometimes 2 RTTs (when they read not-yet ordered updates)

Non-nilext updates: expose state; so, synchronously order



Real-world traces show fast case is common

Skyros Design

Skyros uses several techniques in its design

- durability log and supermajority quorums to complete nilext writes in one RTT
- ordering-and-execution check to serve reads mostly in one RTT
- DAG-based order-resolution to reconstruct linearizable order during view changes
- a variant that exploits commutativity [Lamport, 2004] in addition to nil-externality to quickly commit non-nilext updates

Please see paper …

Outline

Introduction

Strongly consistent storage background

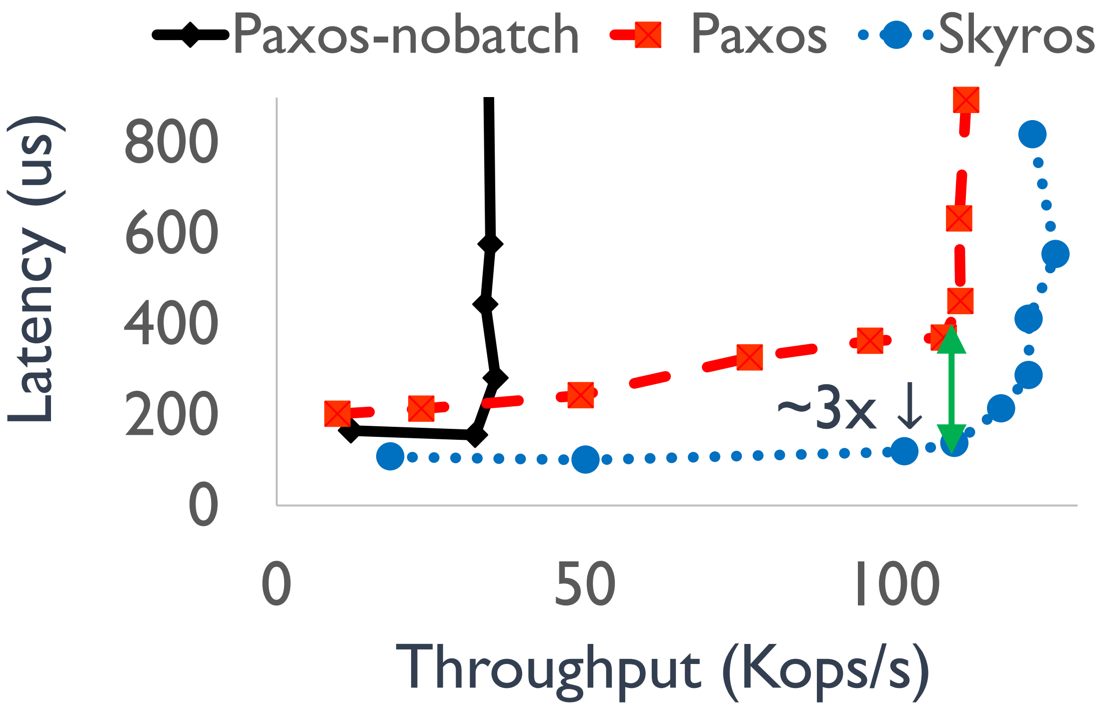
Nilext-aware replication

➡ Evaluation

What are the Benefits of Exploiting Nil-Externality?

Workload: nilext-only updates; vary number of clients

Compare Skyros with Paxos-nobatch and Paxos (with batching, default)



Significant reduction in latency over Paxos w/ batching

More in the paper …

Microbenchmarks varying many factors

- outperforms Paxos in most cases
- at extremes, performs as well as Paxos

Write-heavy YCSB workloads: up to 2x lower latencies

Read-heavy workloads: 70% lower p99 latency

Compare with Curp, a commutative protocol [Park and Ousterhout, 2019]

- 2.7x lower p99 latency for write-only workload

Concluding Thoughts

We identify nil-externality, a property prevalent in storage systems

Skyros, a new replication protocol

- defers coordination until state is externalized
- improves performance for a range of workloads while providing linearizability

Paying attention to what is observable to external clients is key

Useful to exploit properties of an underlying layer

Thank you!

Aishwarya Ganesan (aishwaryag@vmware.com) & Ramnatthan Alagappan (ralagappan@vmware.com)
are on the academic job market this year