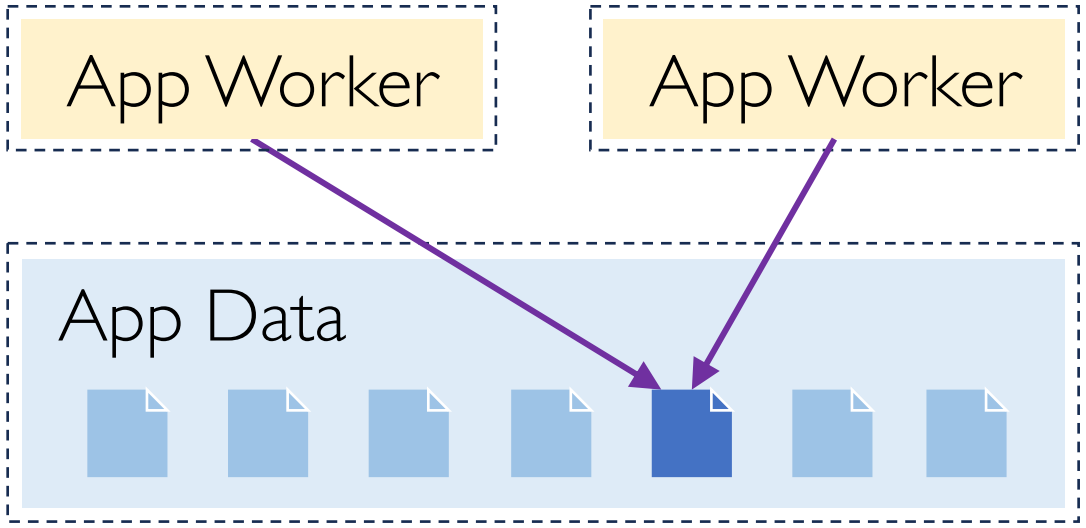
Fast and Scalable In-network Lock Management using *Lock Fission*

Hanze Zhang, Ke Cheng, Rong Chen, Haibo Chen

Institute of Parallel and Distributed Systems (IPADS)

Shanghai Jiao Tong University

Locking is Vital for Concurrency Control



Serialize

conflicting accesses

towards

shared data

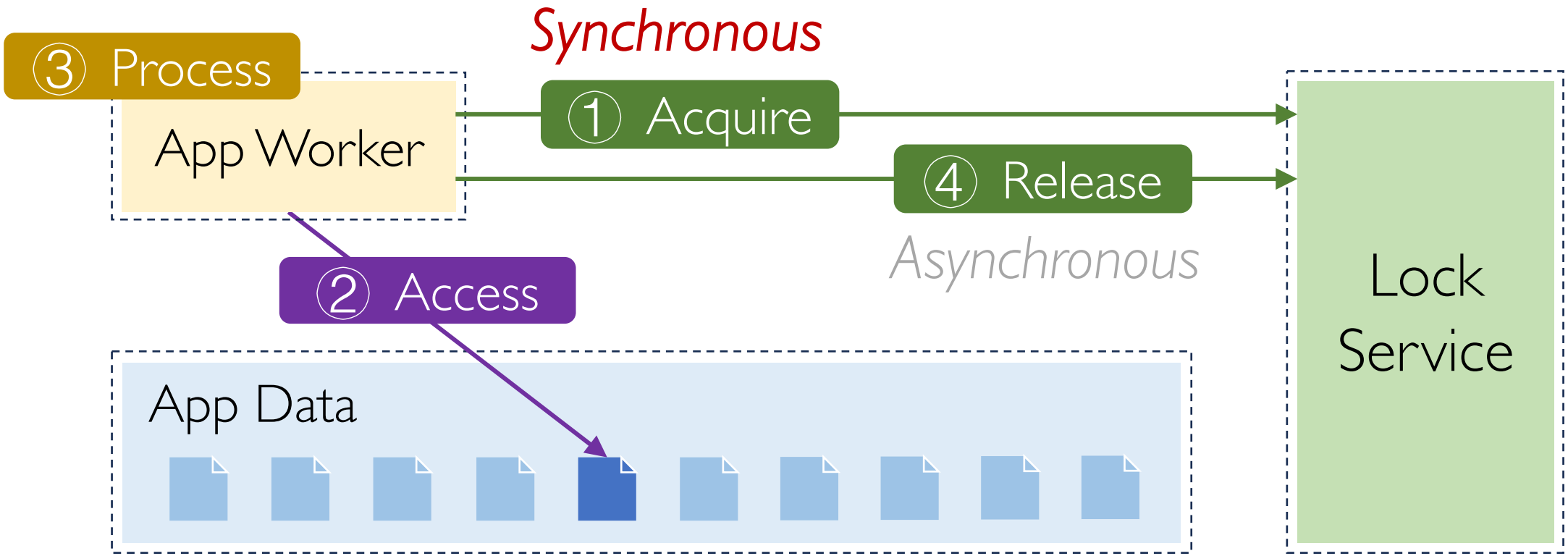
Typical scenarios:

PolarDB

hadoop HDFS

druid

Typical Use Case of Distributed Locks



Two Trends in Distributed Apps

	Workload	Exec. Time	Data Scale	Reference
Txn. Processing	TPC-C/TATP	7/2.8 µs	160M rows	DrTM, FaSST, PolarDB
File System	Read/Write/Mkdir	1/10/20 µs	10B files	Assise, Octopus, Tectonic
Key-value Store	Search/Insert	8/15 µs	250M keys	XStore, RACE, Redis

Low execution time

Microsecond-scale
execution time becomes common

Large data scale

Shared data scale is growing to
Near-billion-level

Two Trends in Distributed Apps

	Workload	Exec. Time	Data Scale	Reference
Txn. Processing	TPC-C/TATP	7/2.8 µs	160M rows	DrTM, FaSST, PolarDB

Distributed lock should be *fast on acquisition* and *scalable on the number of locks*

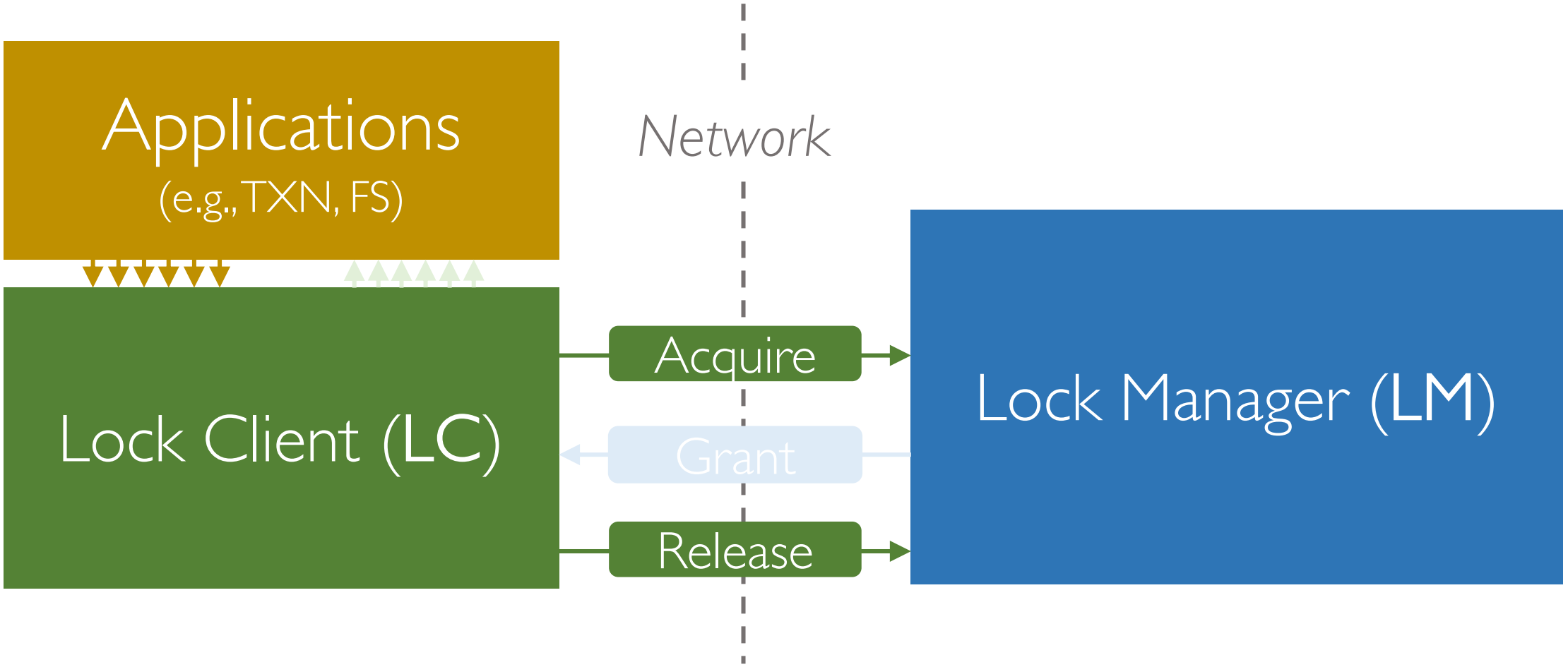
Low execution time

Microsecond-scale
execution time becomes common

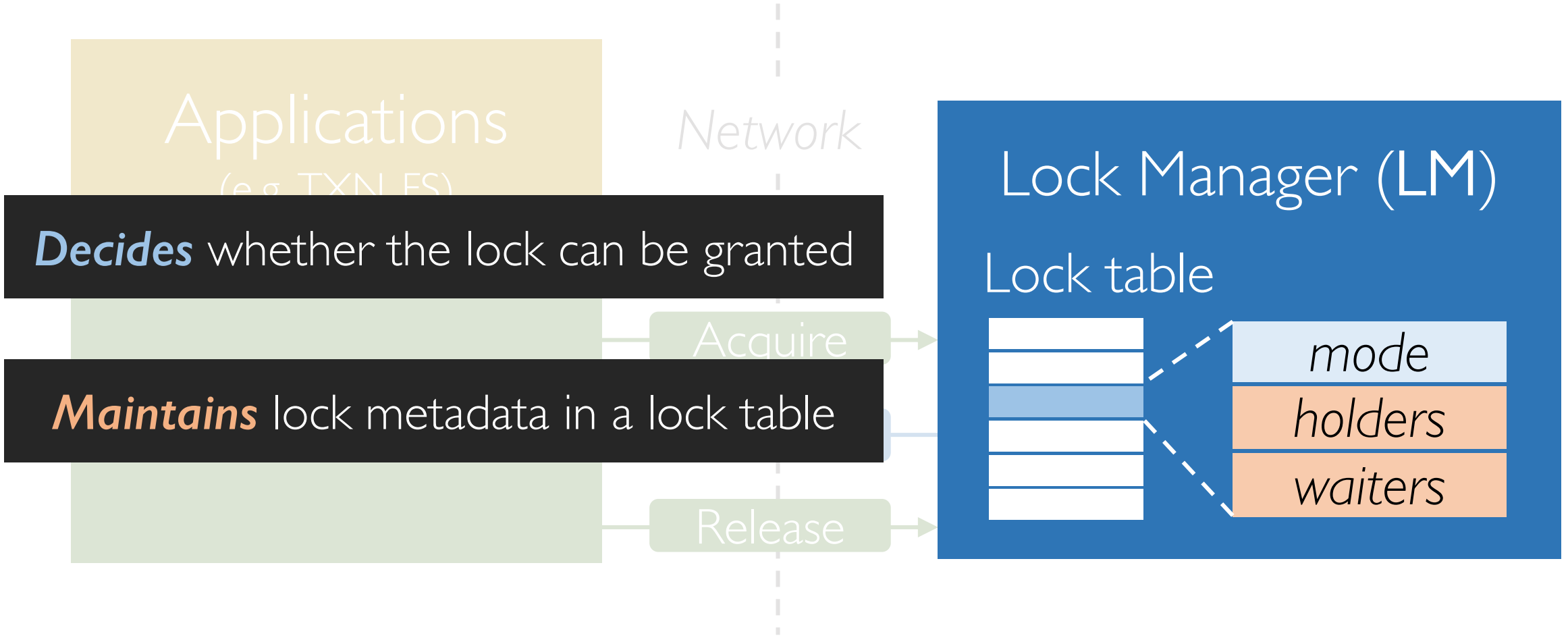
Large data scale

Shared data scale is growing to
Near-billion-level

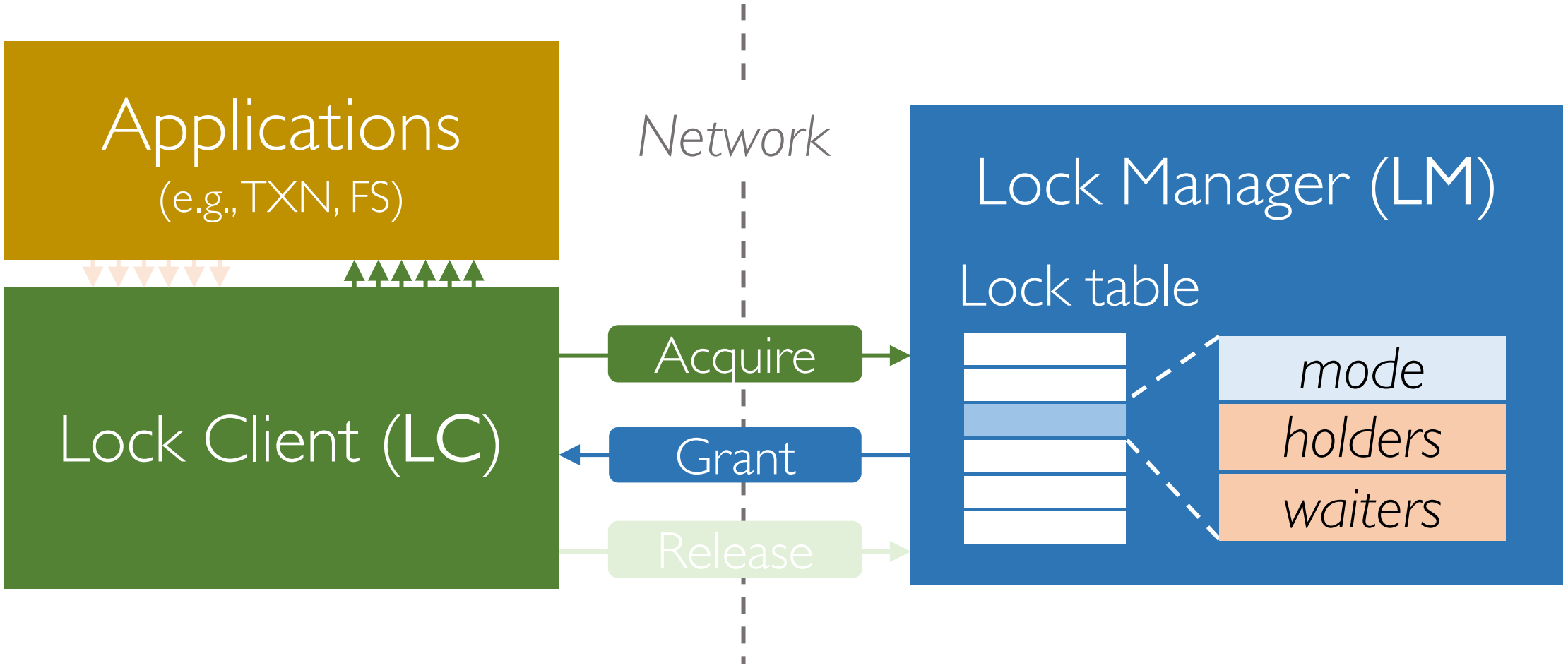
Distributed Lock Service Architecture



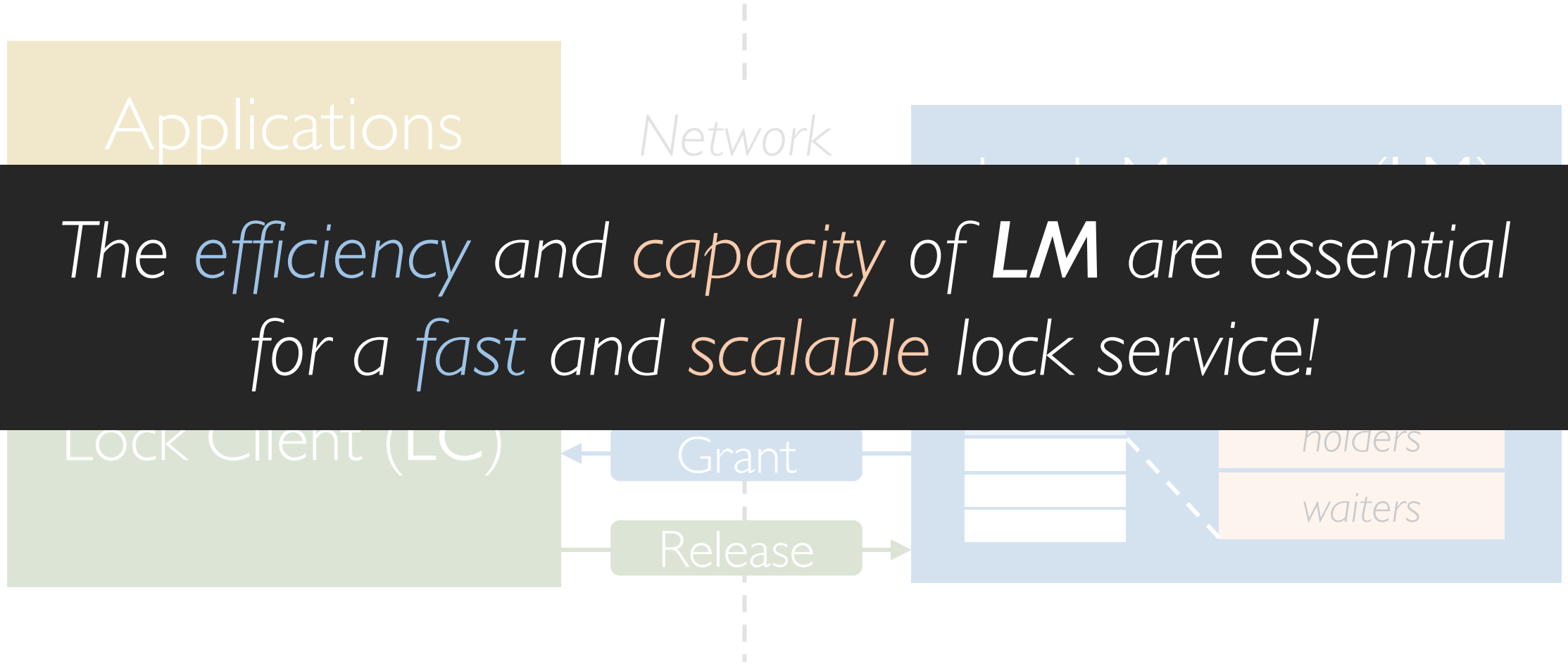
Distributed Lock Service Architecture



Distributed Lock Service Architecture

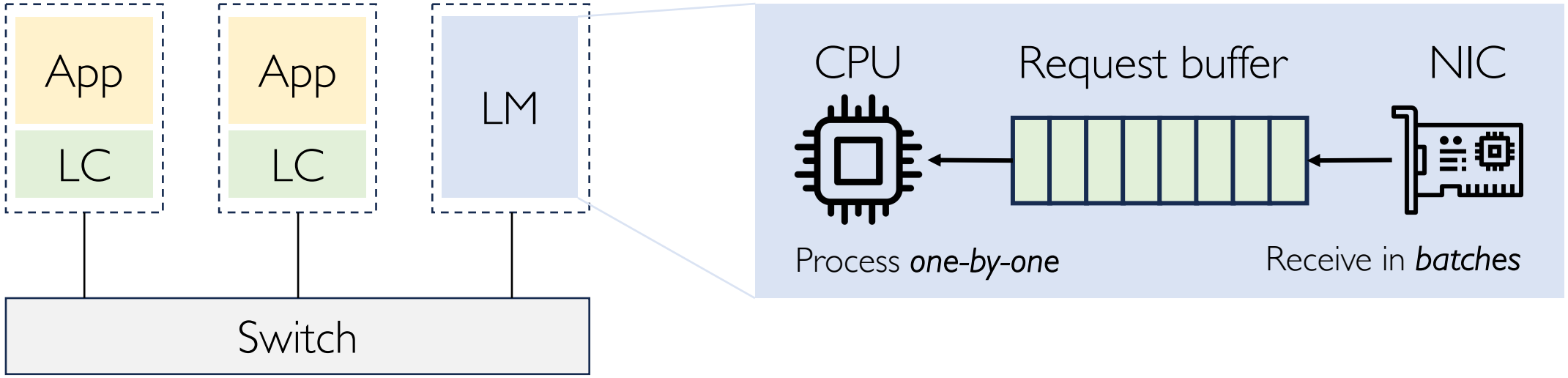


Distributed Lock Service Architecture



*The efficiency and capacity of **LM** are essential for a fast and scalable lock service!*

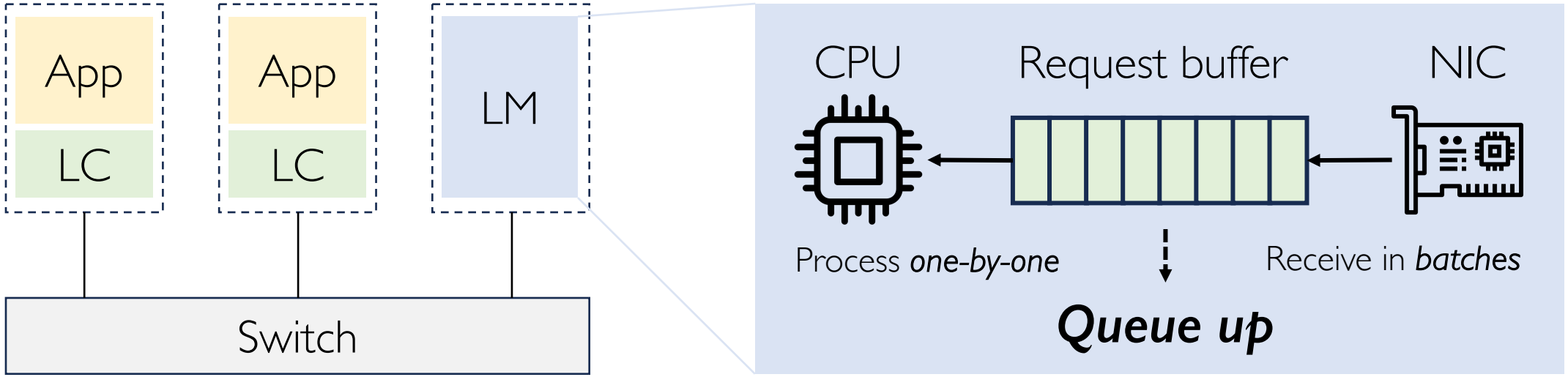
Existing Lock Manager Designs



SrvLock

Handle lock requests with
dedicated servers

Existing Lock Manager Designs



SrvLock

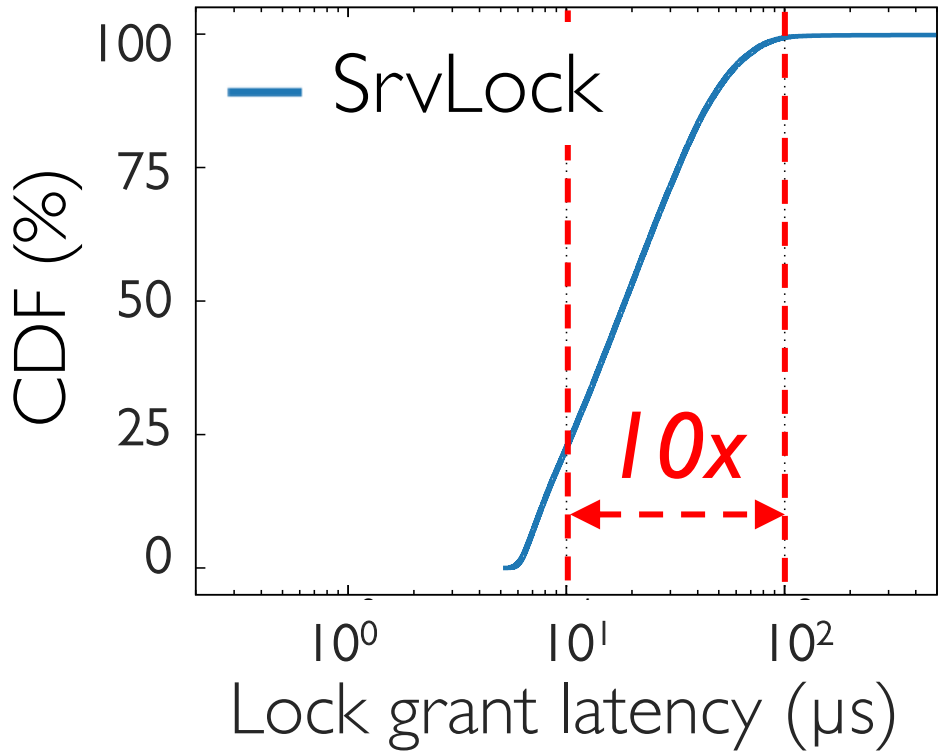
Handle lock requests with
dedicated servers

Scalable on lock number

Slow due to high queueing delay

Performance Issues of Existing LMs

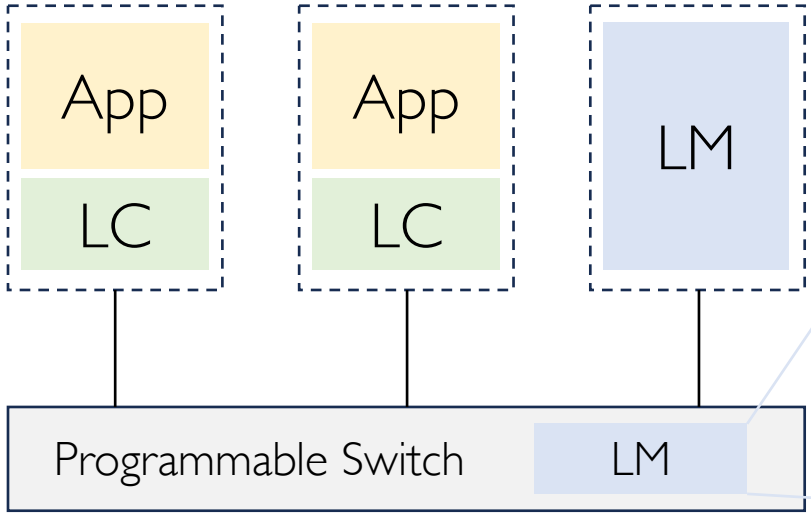
#1 Large ***latency variance*** due to queueing delay

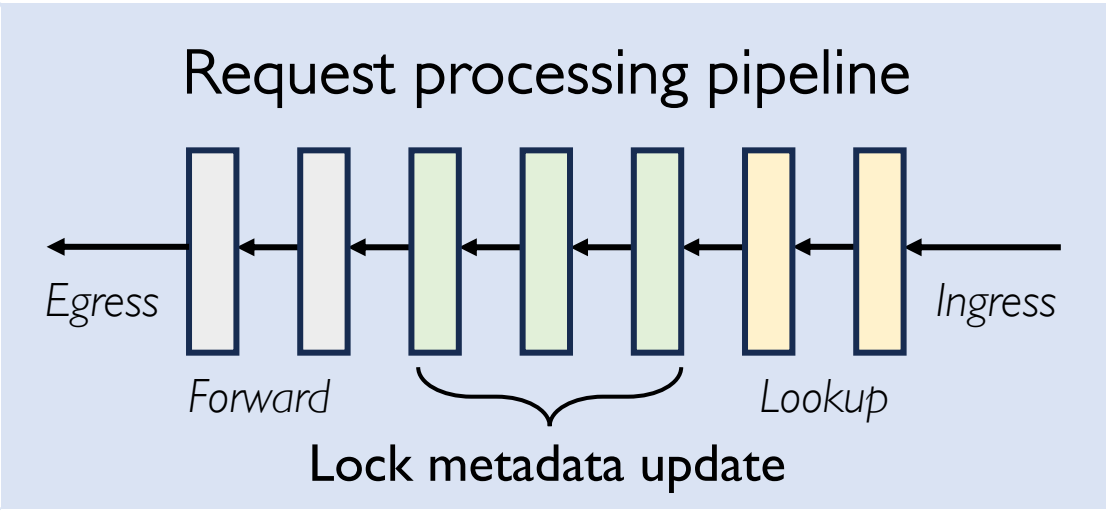


Latency variance (10^{th} ~ 99^{th})

Update-heavy	Uniform	17.26x
	Zipfian	2.48x
Read-mostly	Uniform	19.08x
	Zipfian	12.34x
Read-only	Uniform	20.16x
	Zipfian	18.96x

Existing Lock Manager Designs



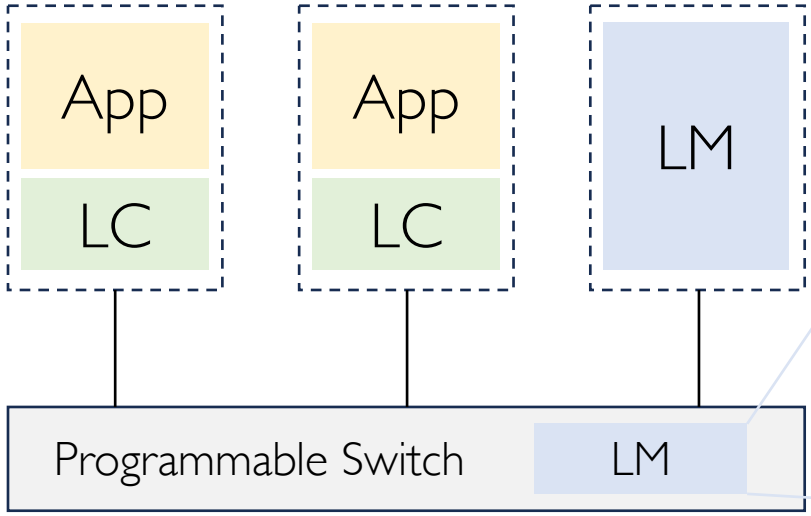


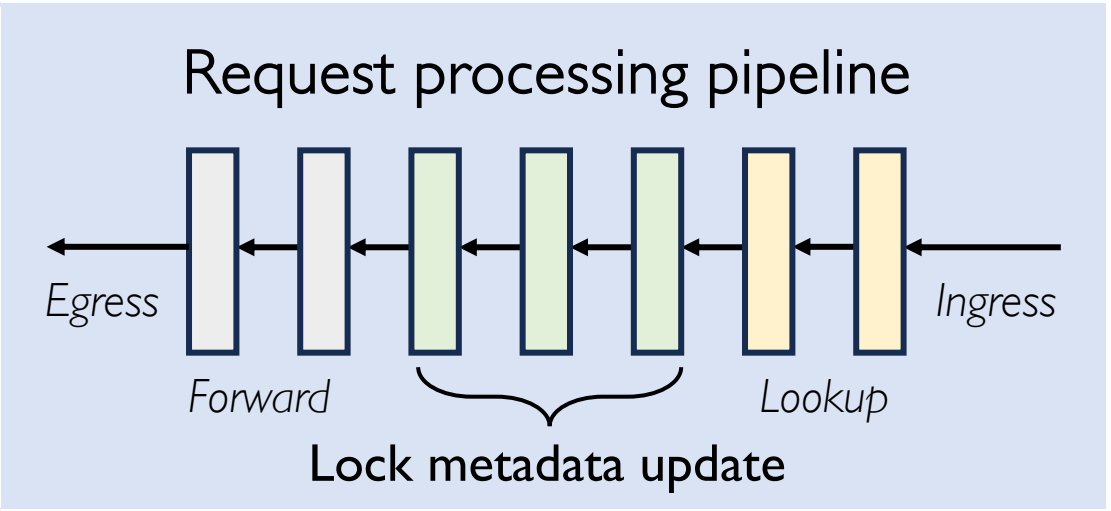
NetLock

(SIGCOMM'20)

Fast-path lock manager
on **programmable switch**

Existing Lock Manager Designs





NetLock

(SIGCOMM'20)

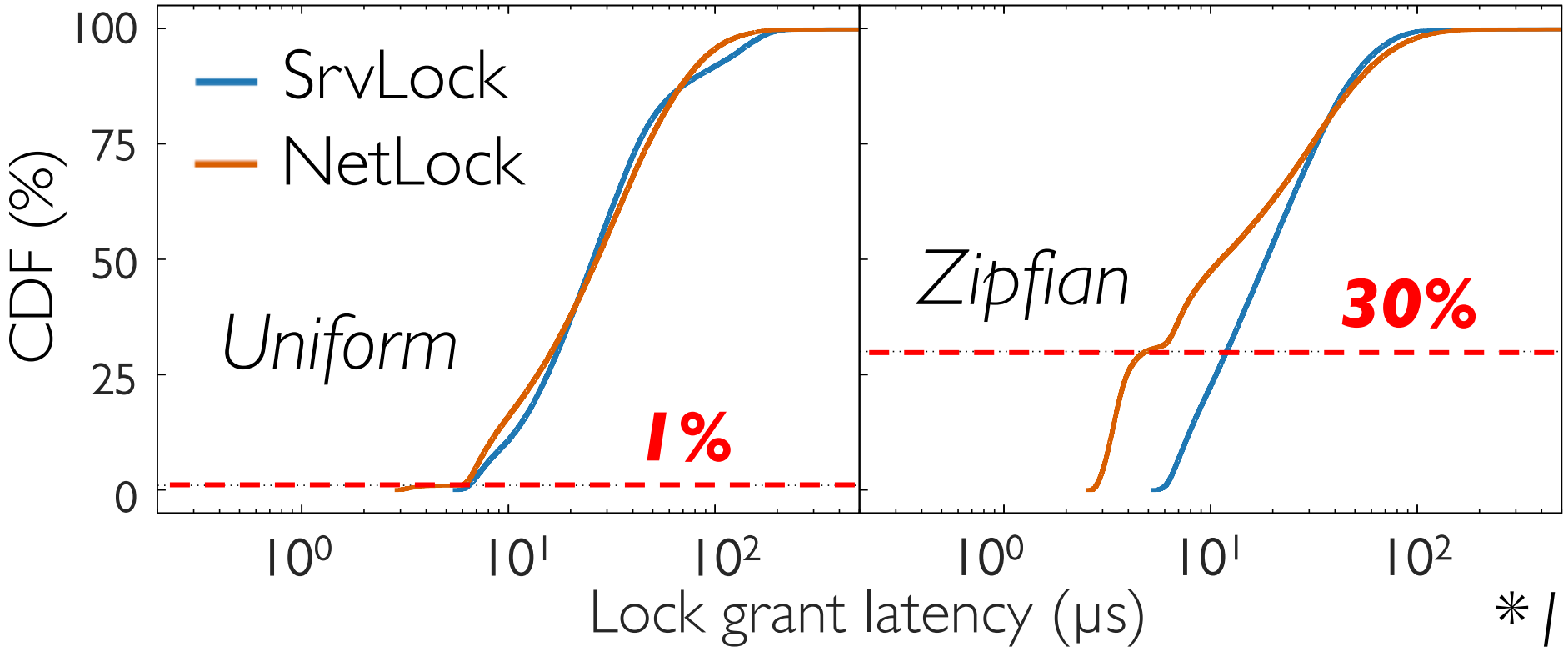
Fast-path lock manager
on **programmable switch**

Fast (*near-zero* queueing delay)

Unscalable due to small memory

Performance Issues of Existing LMs

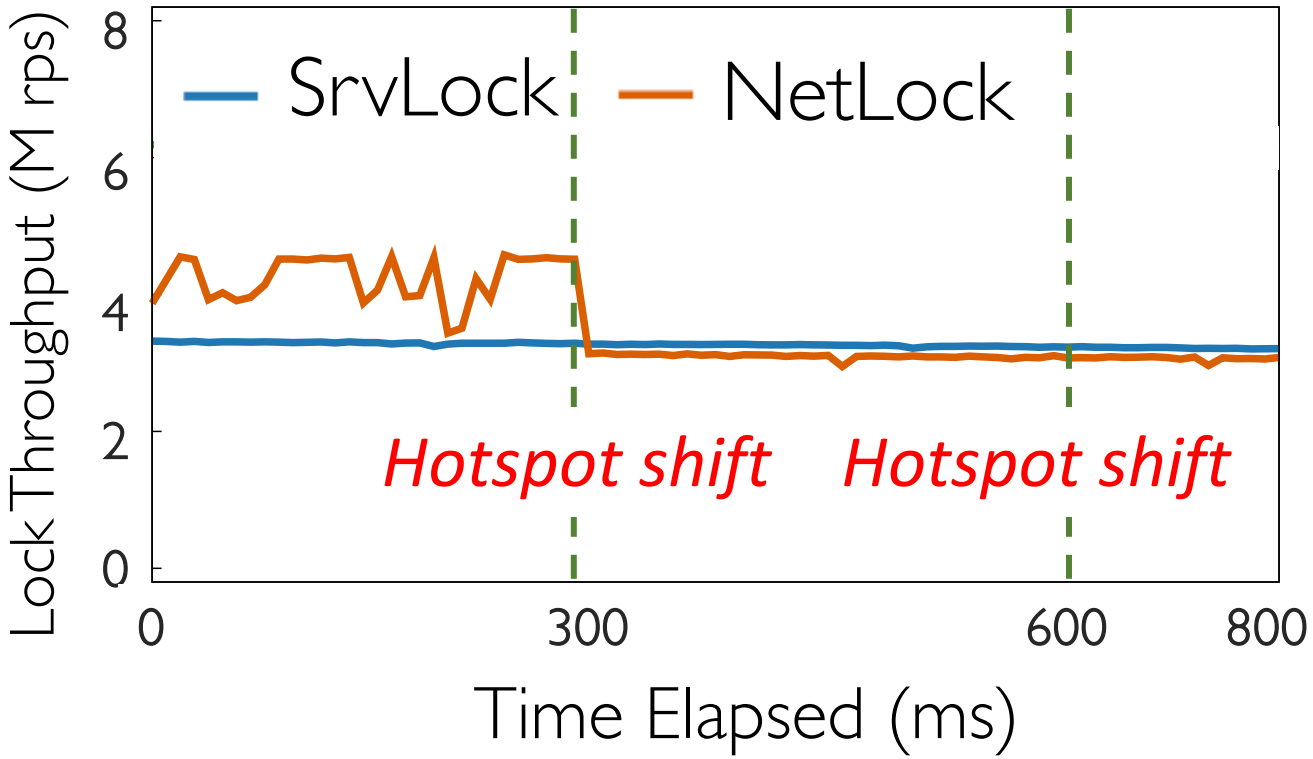
#2 *Limited acceleration* due to poor scalability on #locks



**1 million locks*

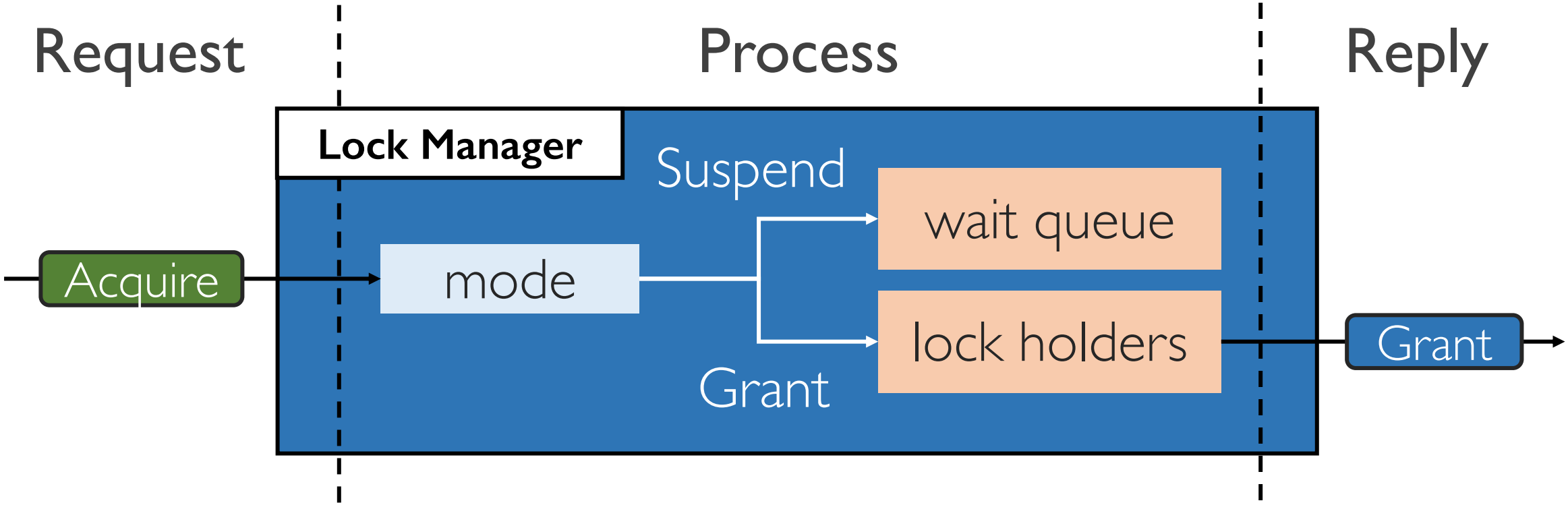
Performance Issues of Existing LMs

#3 High ***workload sensitivity*** due to static profiling

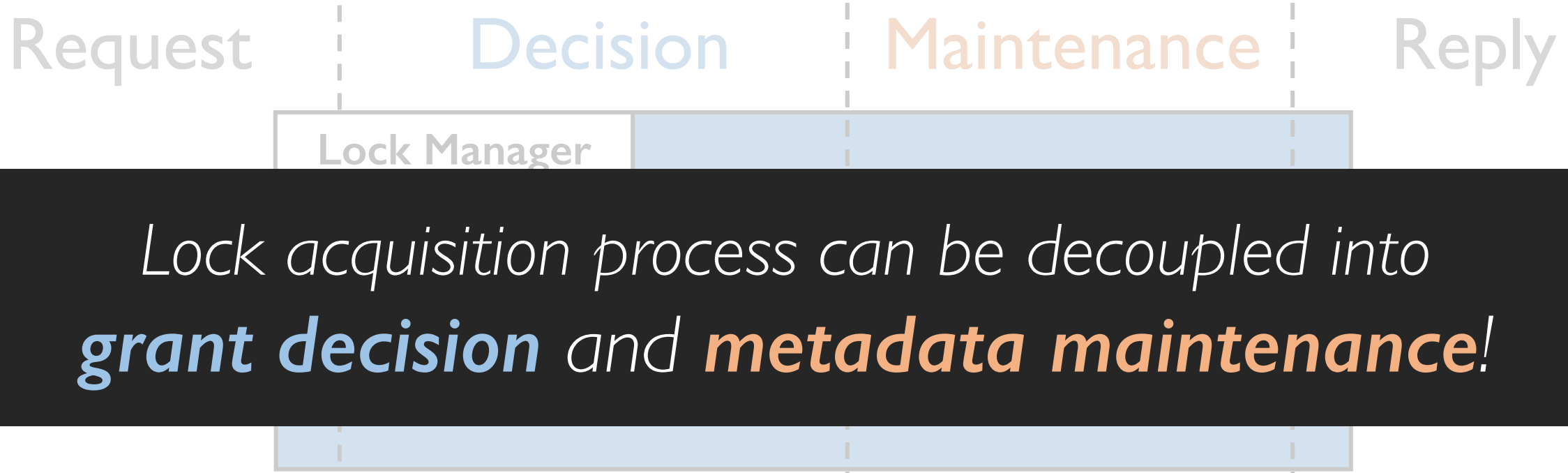


**1 million locks, 50% requests to 2500 hot locks*

Revisit the Lock Acquisition Process

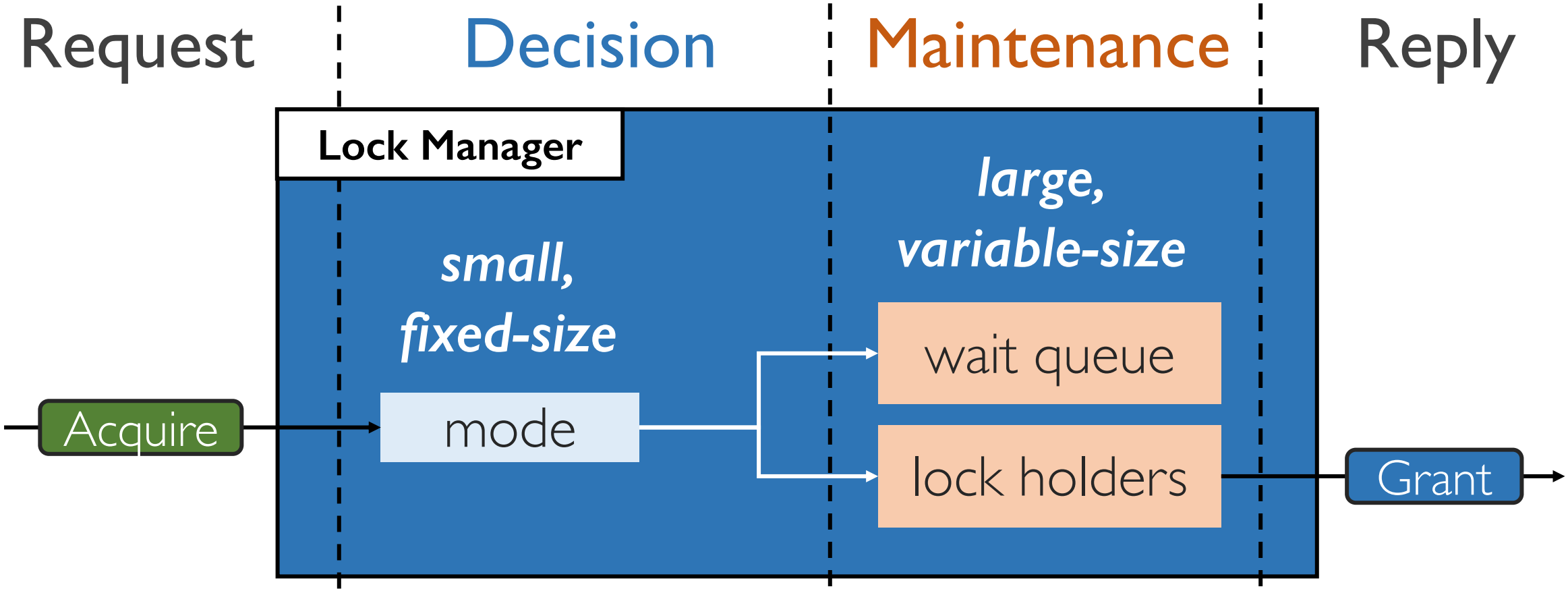


Our Key Insight

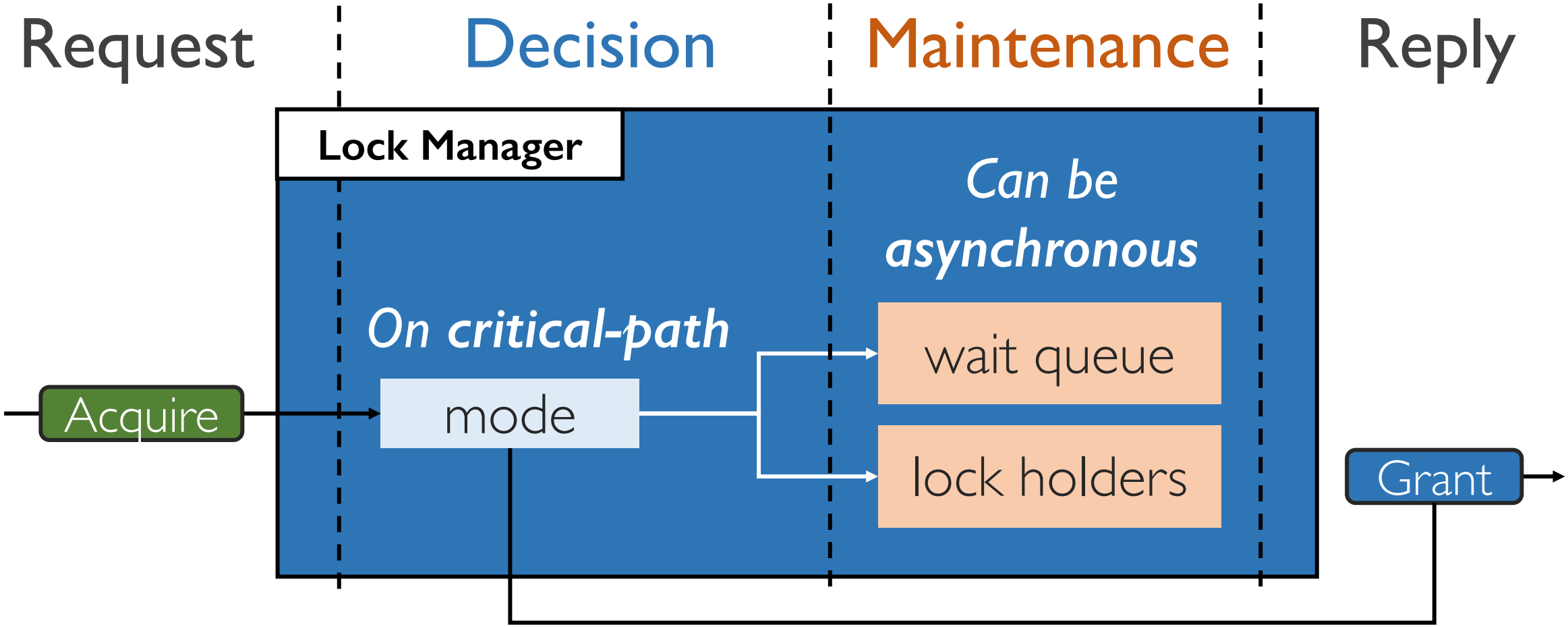


Lock acquisition process can be decoupled into ***grant decision*** *and* ***metadata maintenance****!*

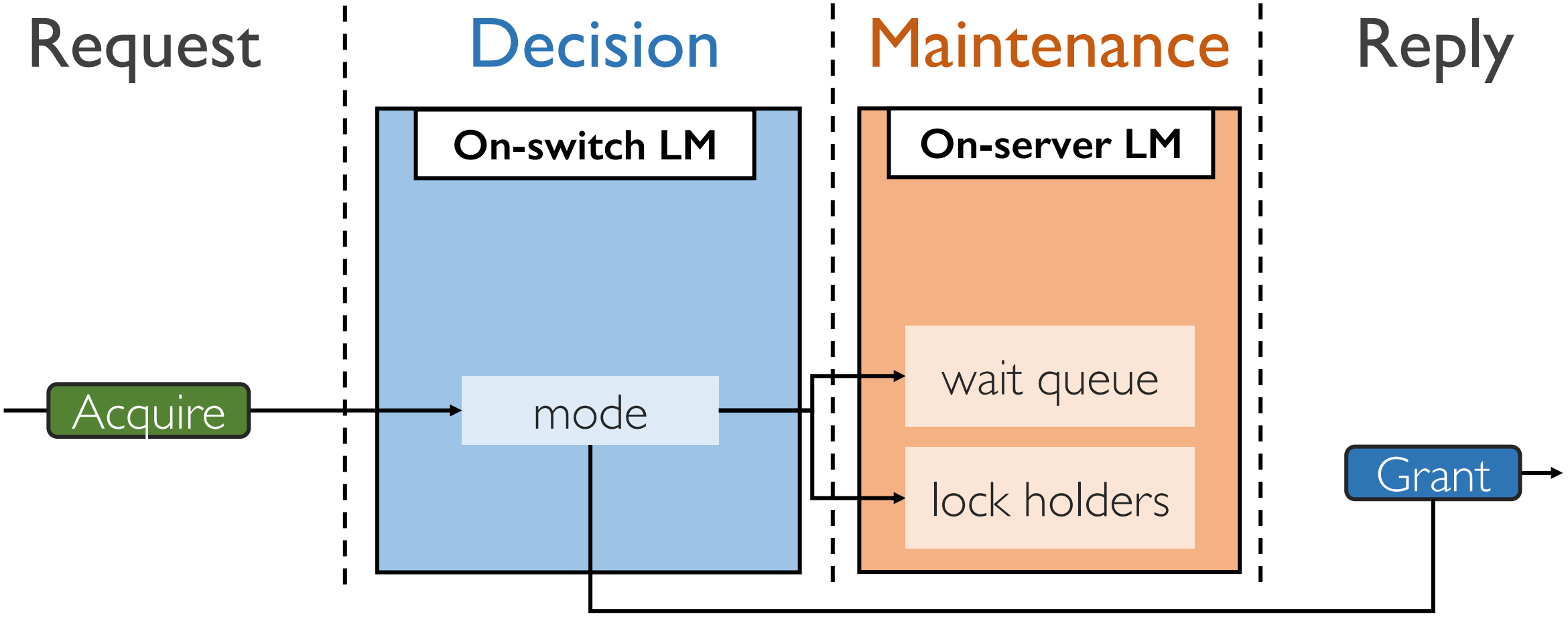
Our Key Insight



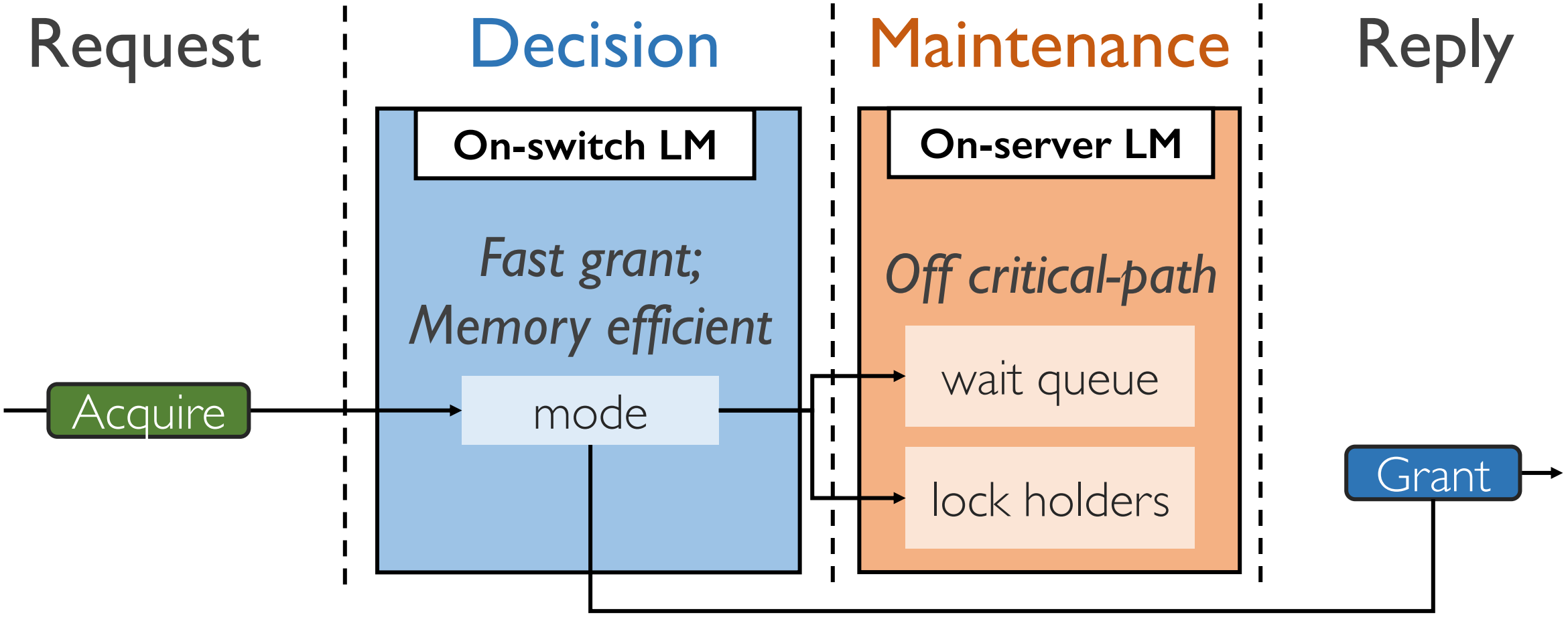
Our Key Insight



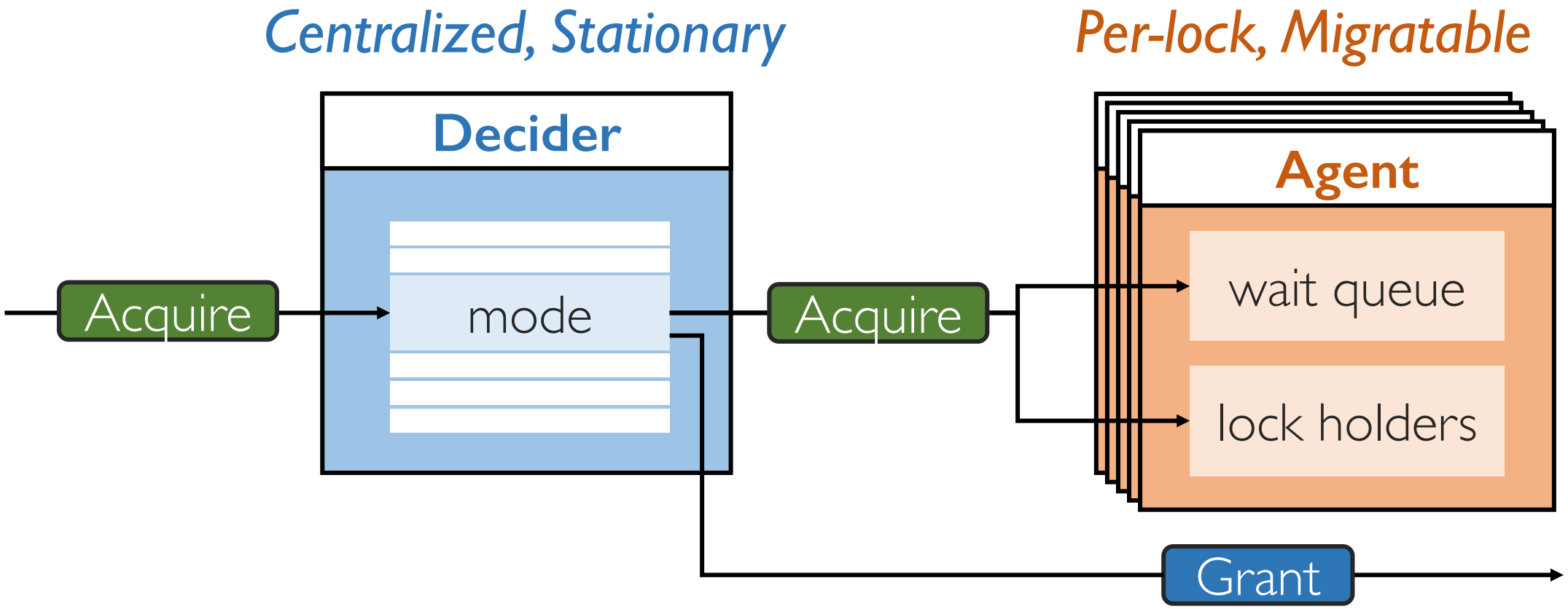
Key Idea: Lock Fission



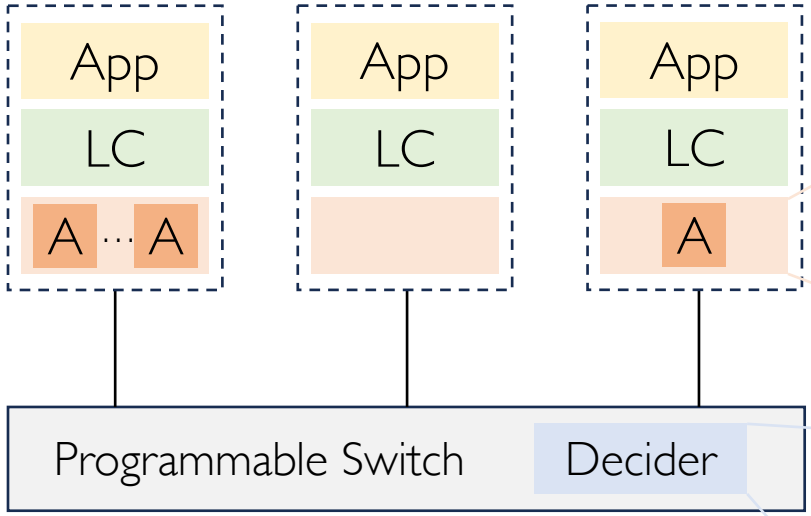
Key Idea: Lock Fission

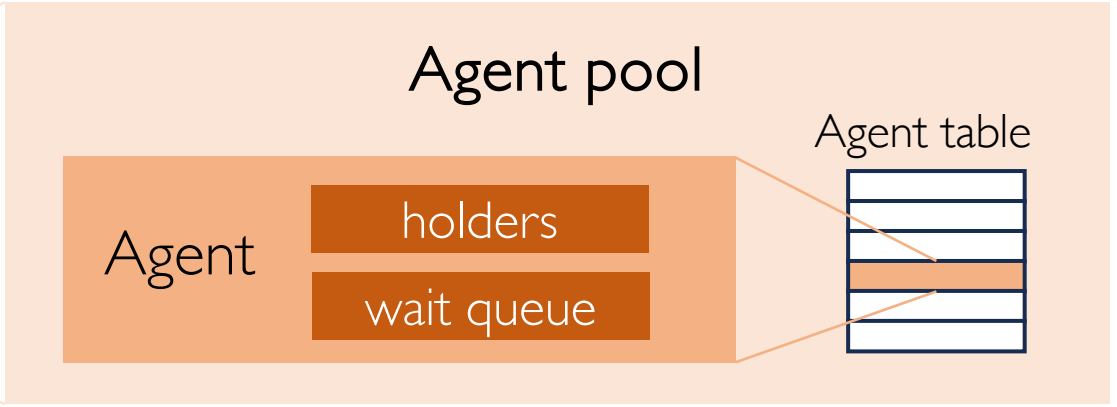


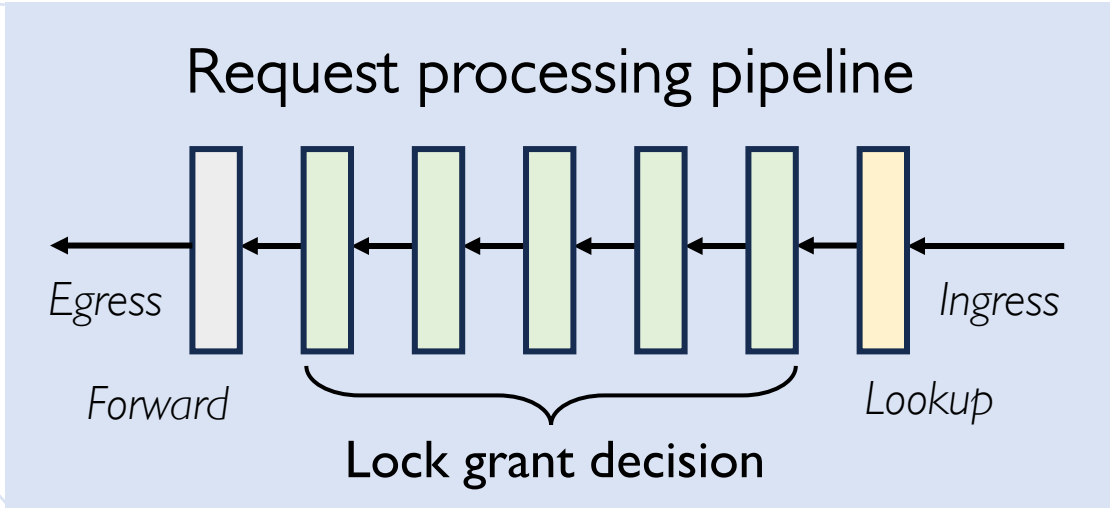
Key Idea: Lock Fission



Our System: FissLock



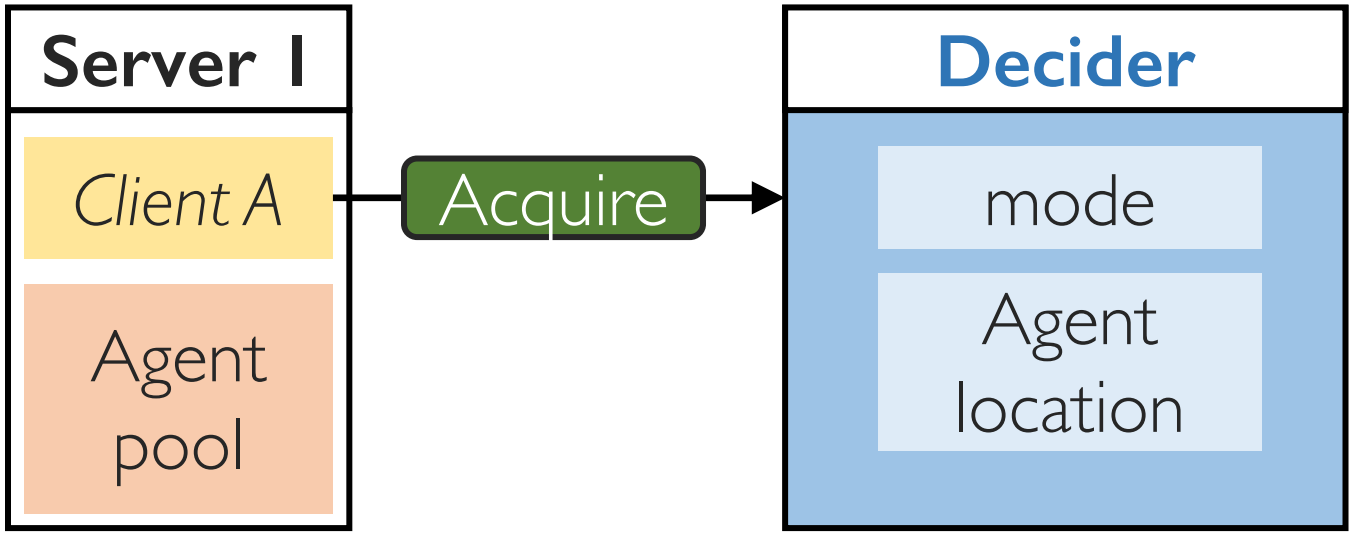




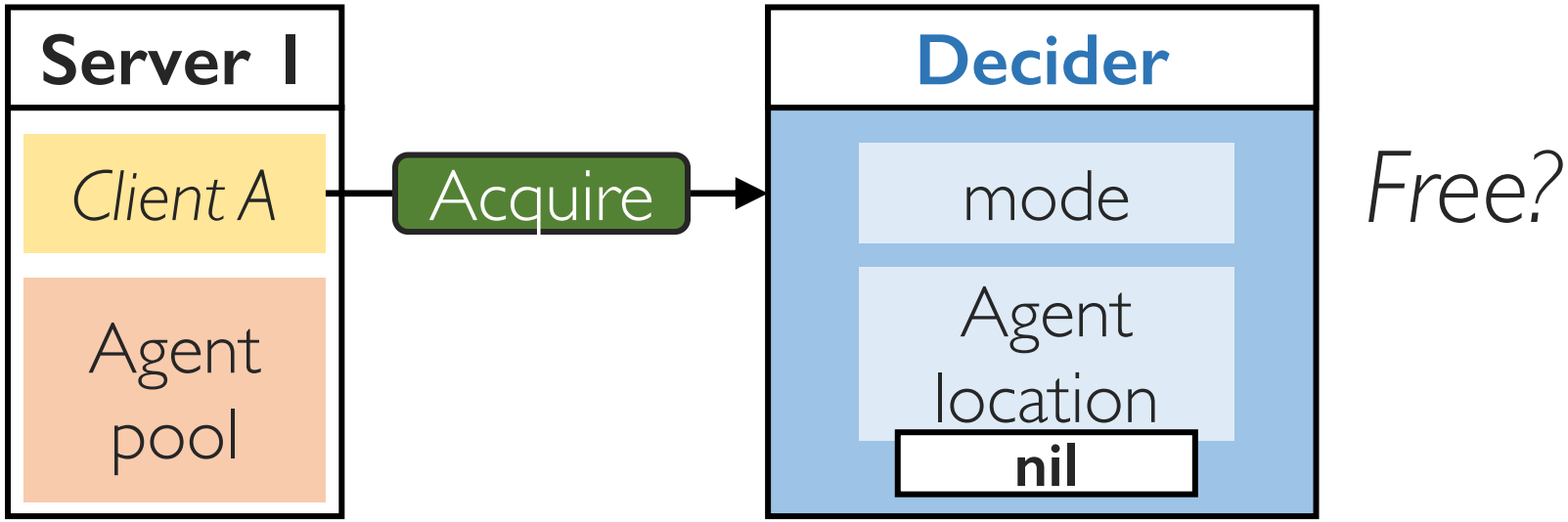
FissLock

On-switch decider and
On-server agents

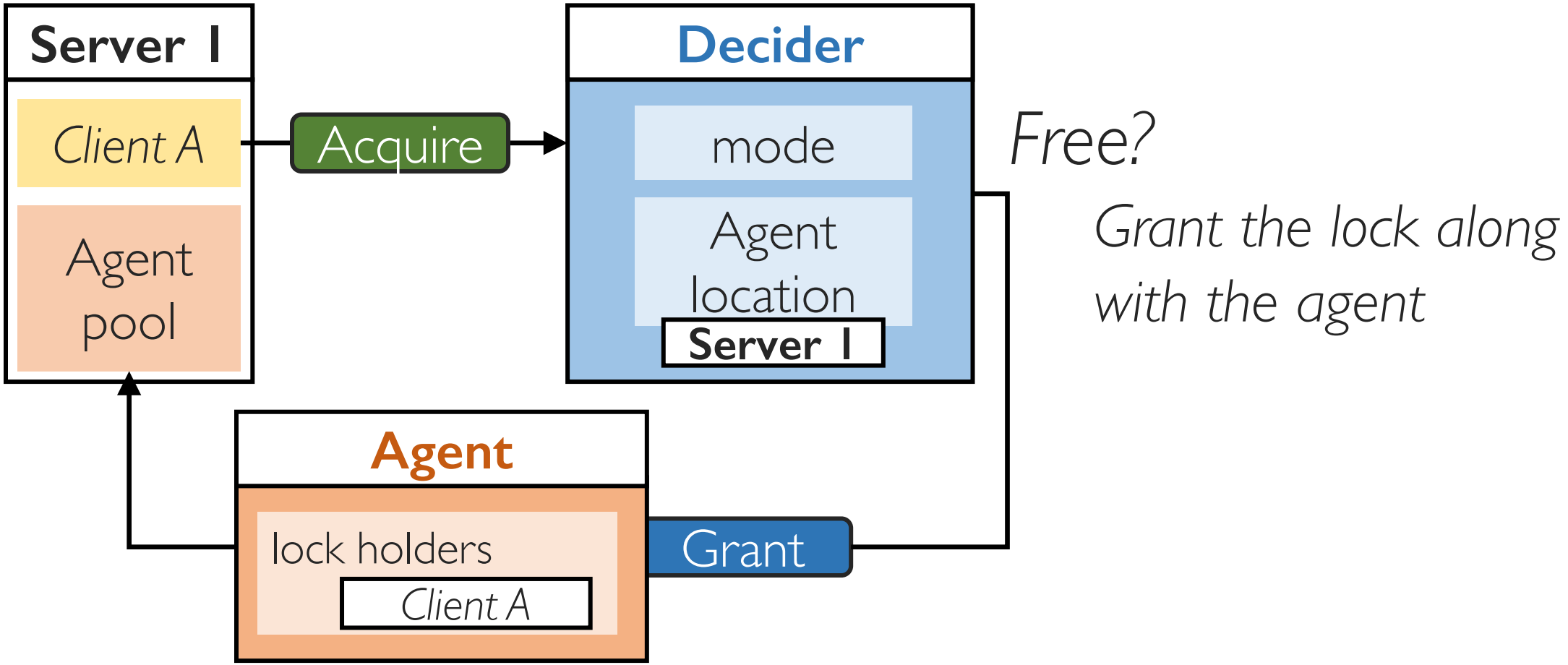
Lock Acquisition Workflow



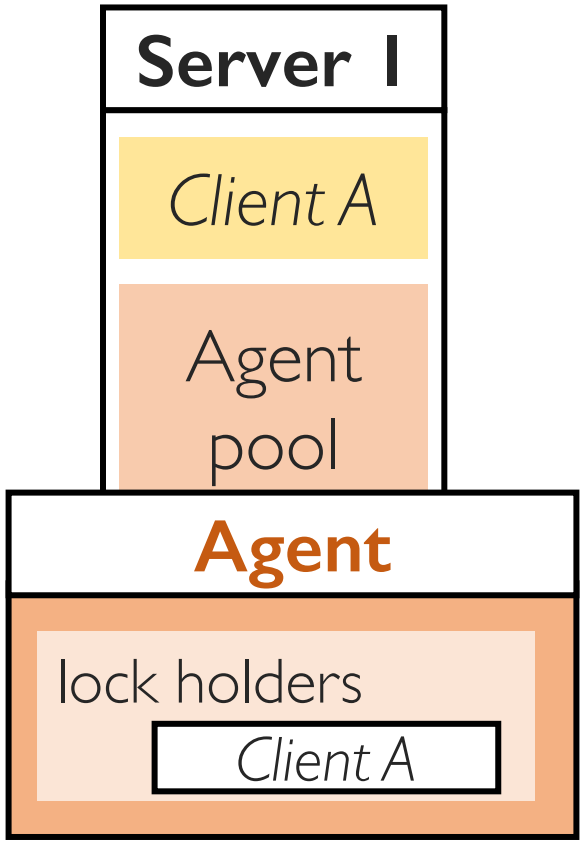
Lock Acquisition Workflow

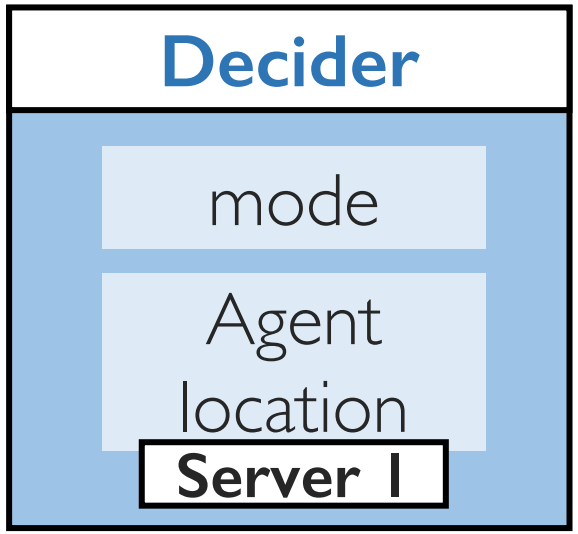


Lock Acquisition Workflow



Lock Acquisition Workflow

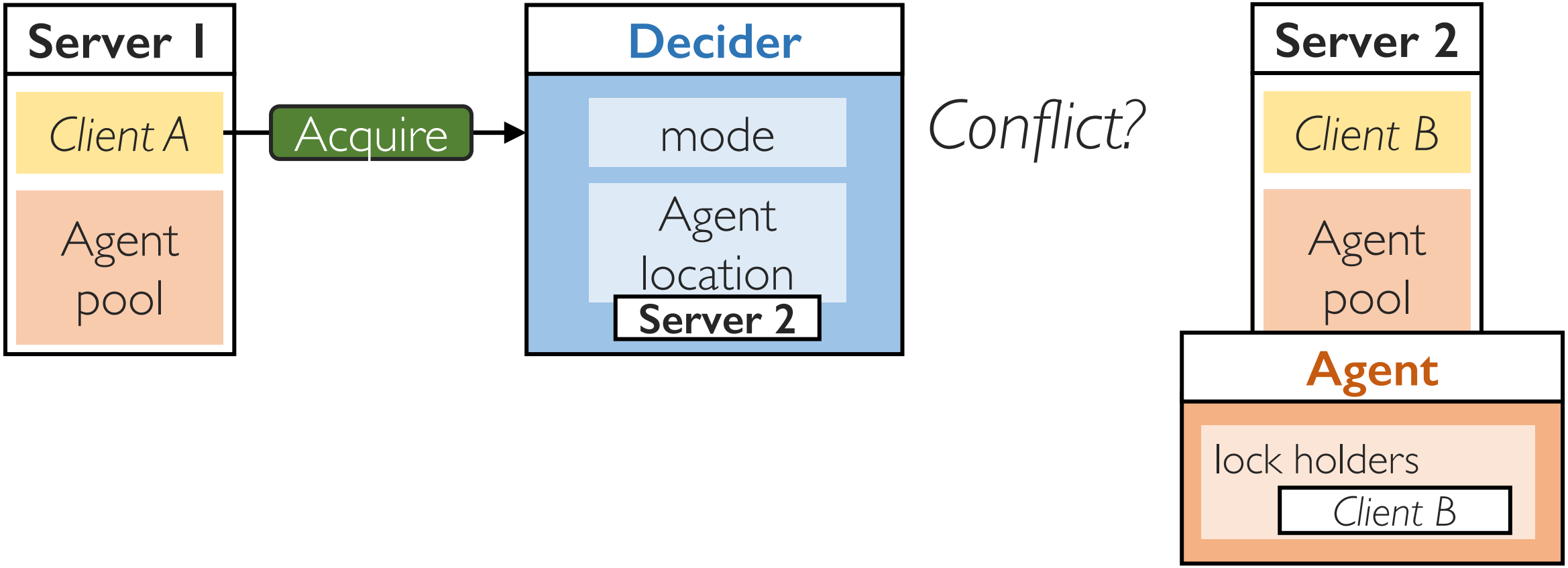




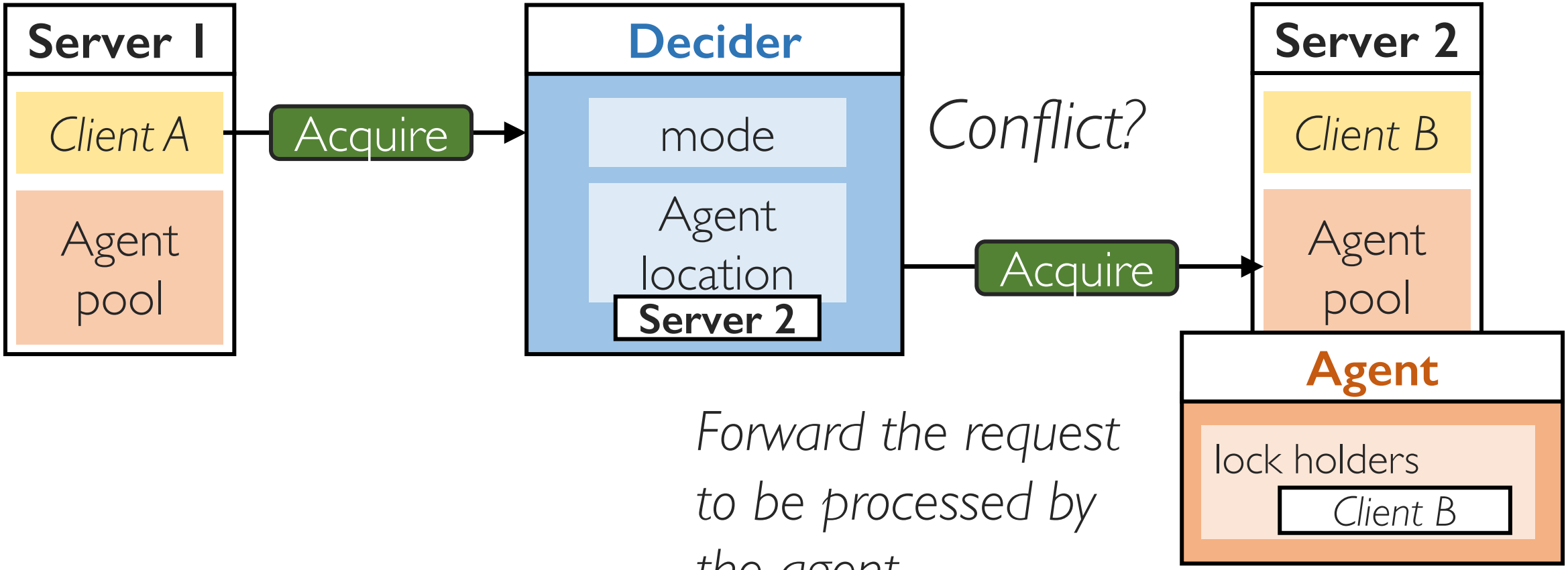
Free?

Grant the lock along with the agent

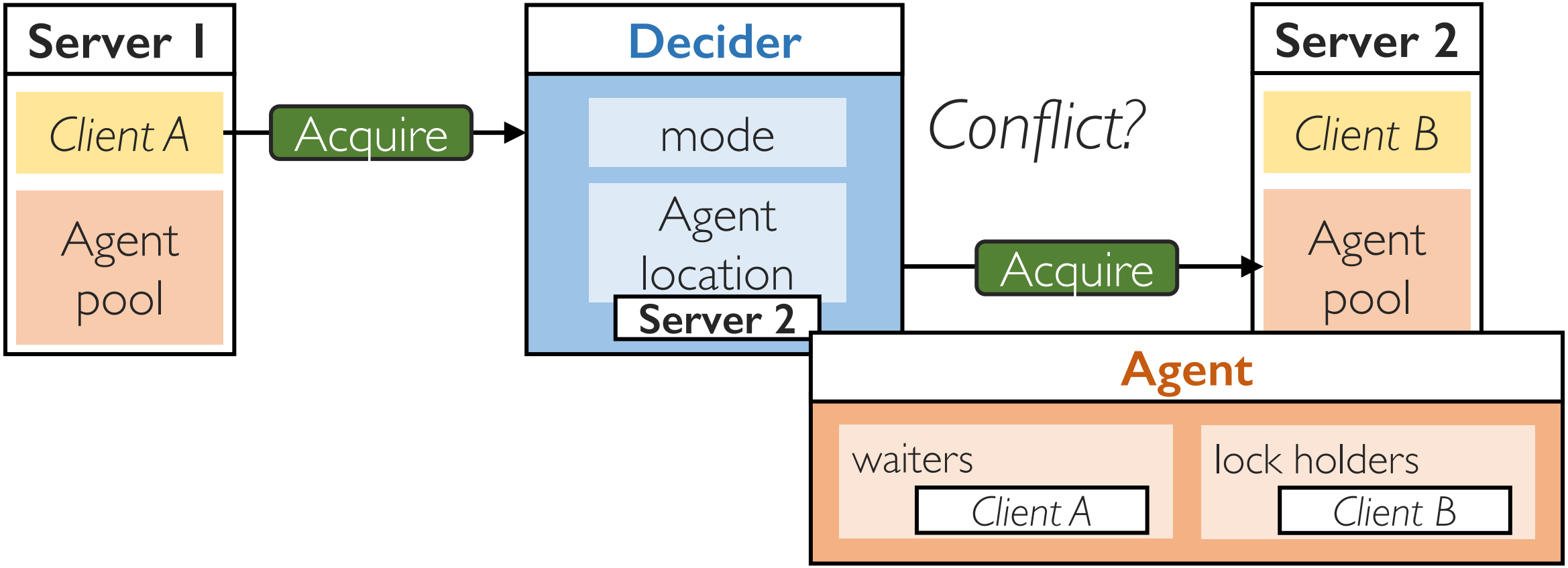
Lock Acquisition Workflow



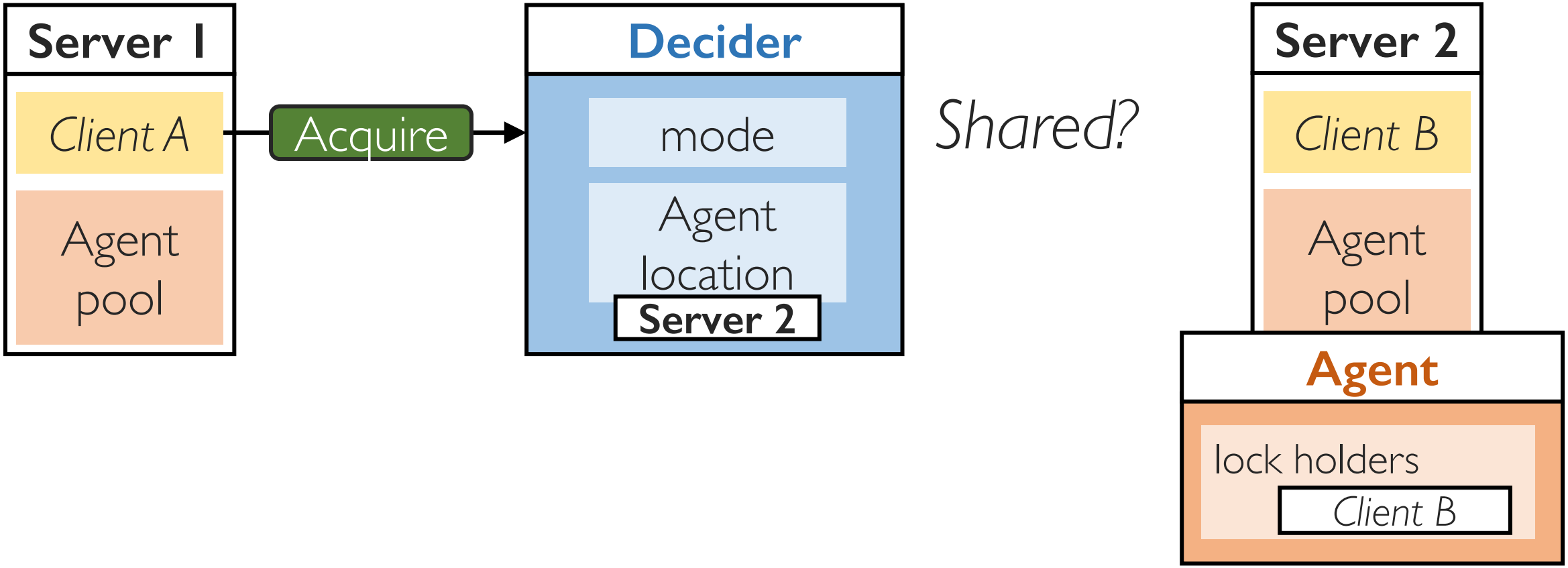
Lock Acquisition Workflow



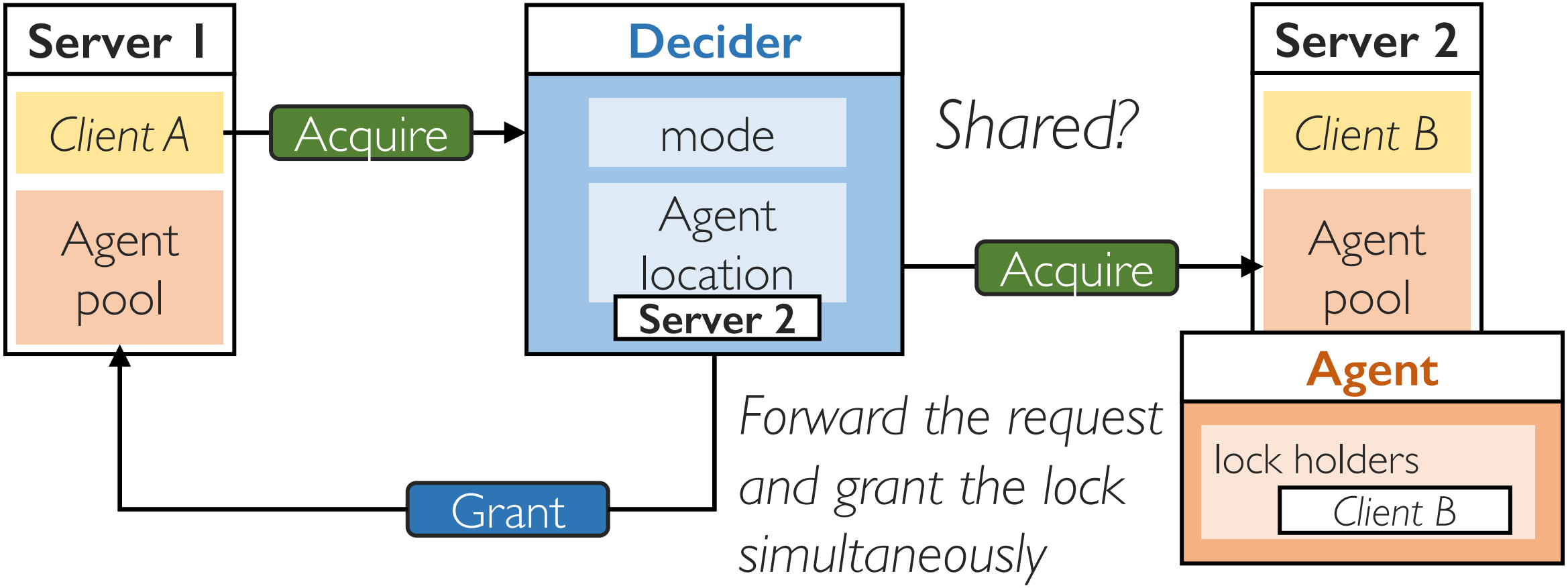
Lock Acquisition Workflow



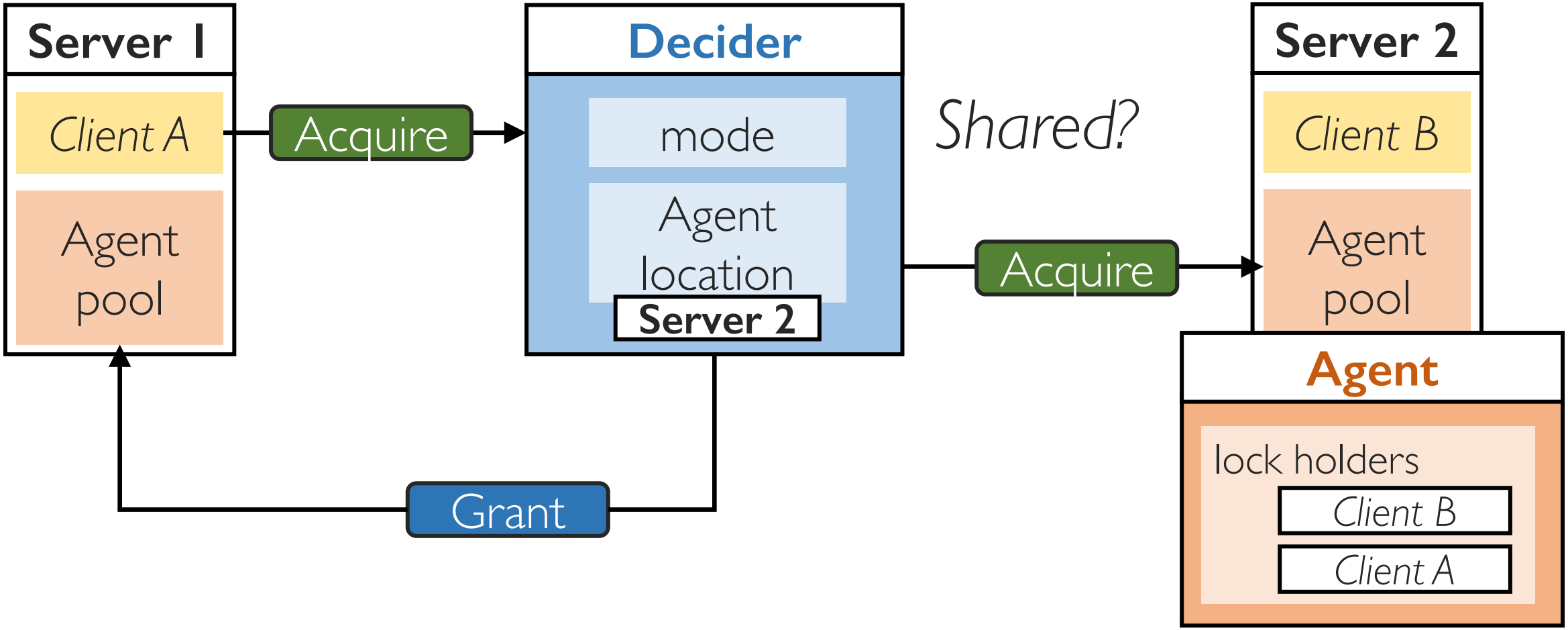
Lock Acquisition Workflow



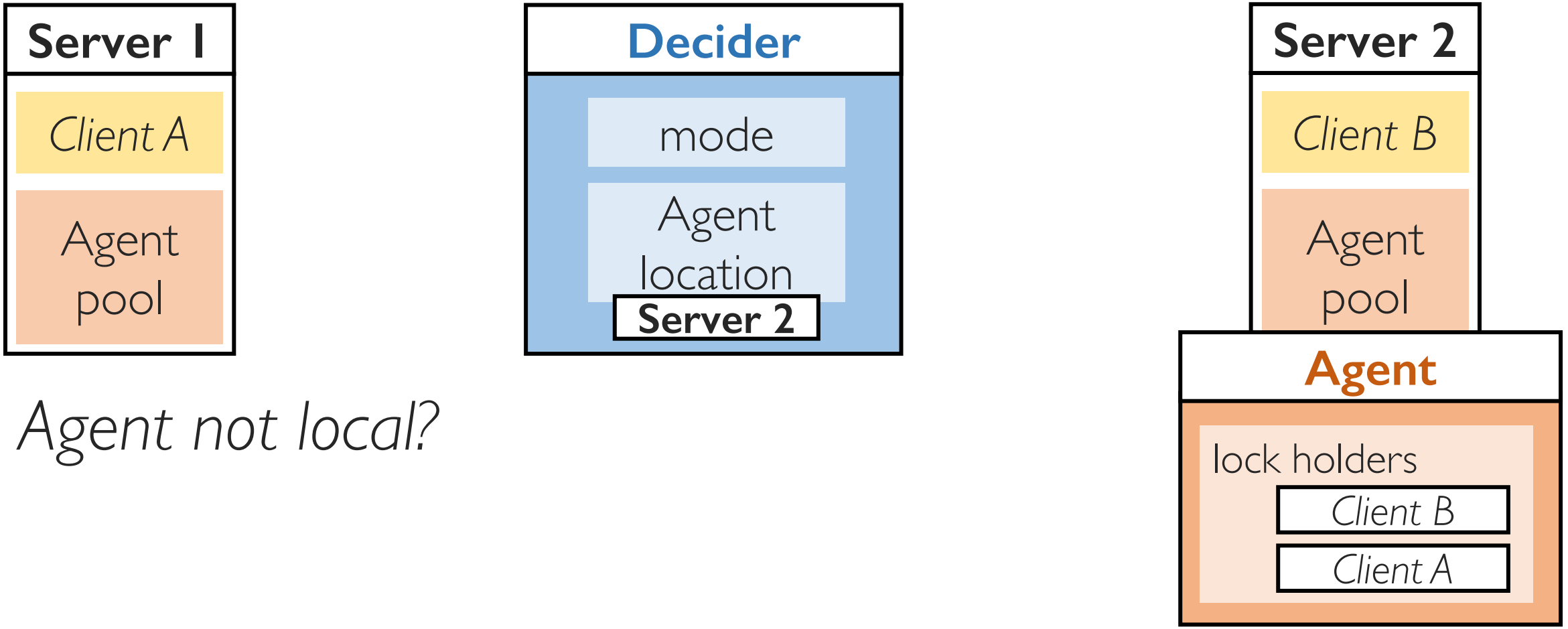
Lock Acquisition Workflow



Lock Acquisition Workflow

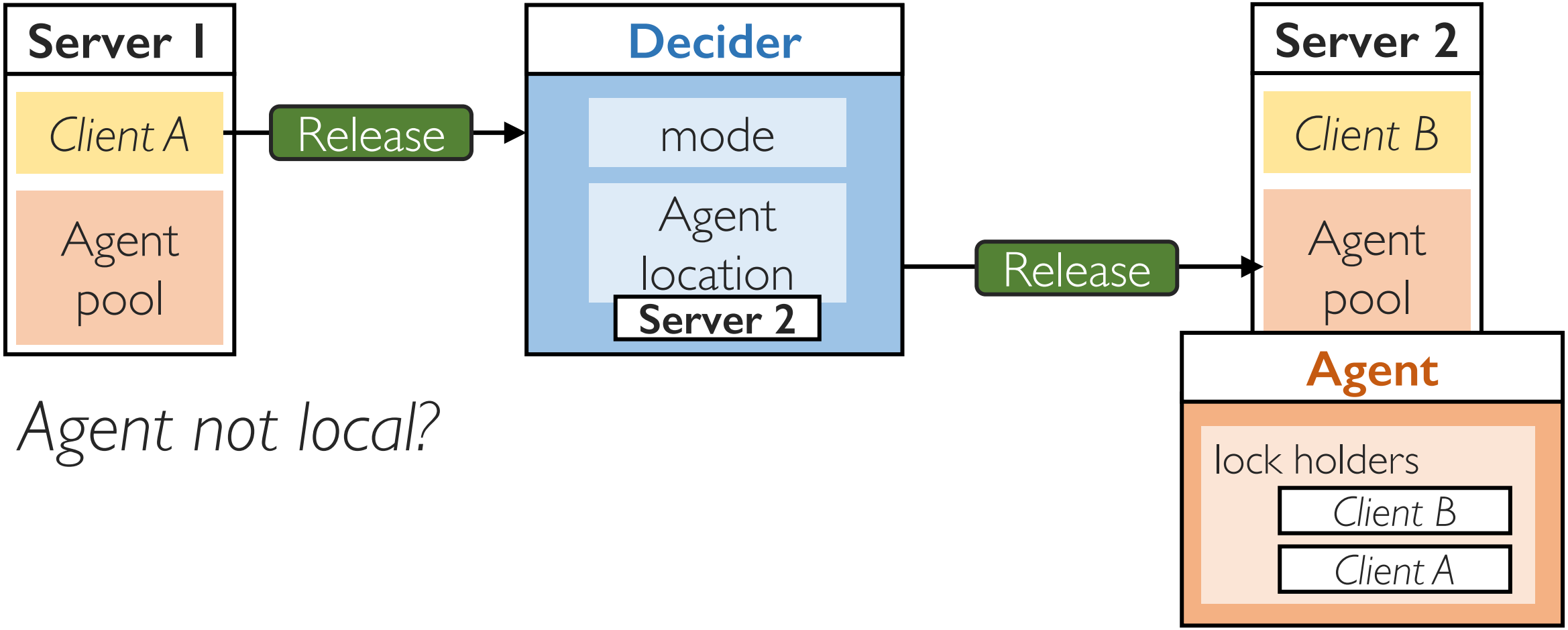


Lock Release Workflow



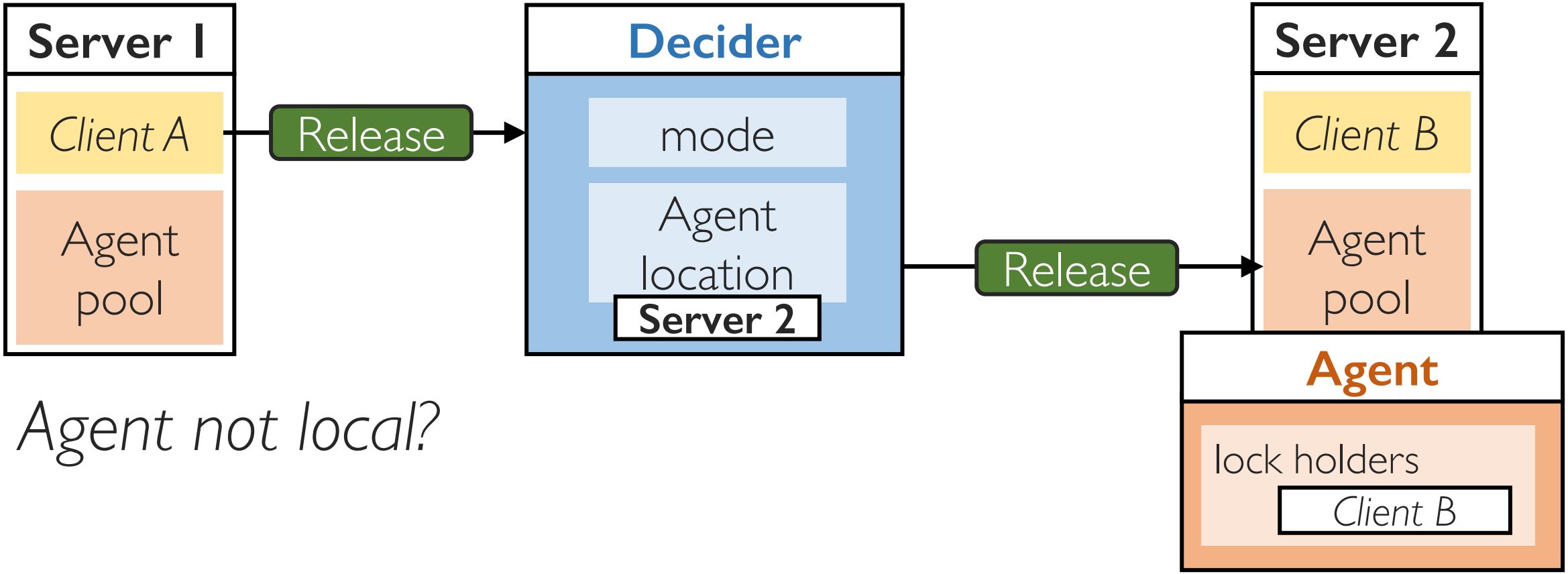
Agent not local?

Lock Release Workflow



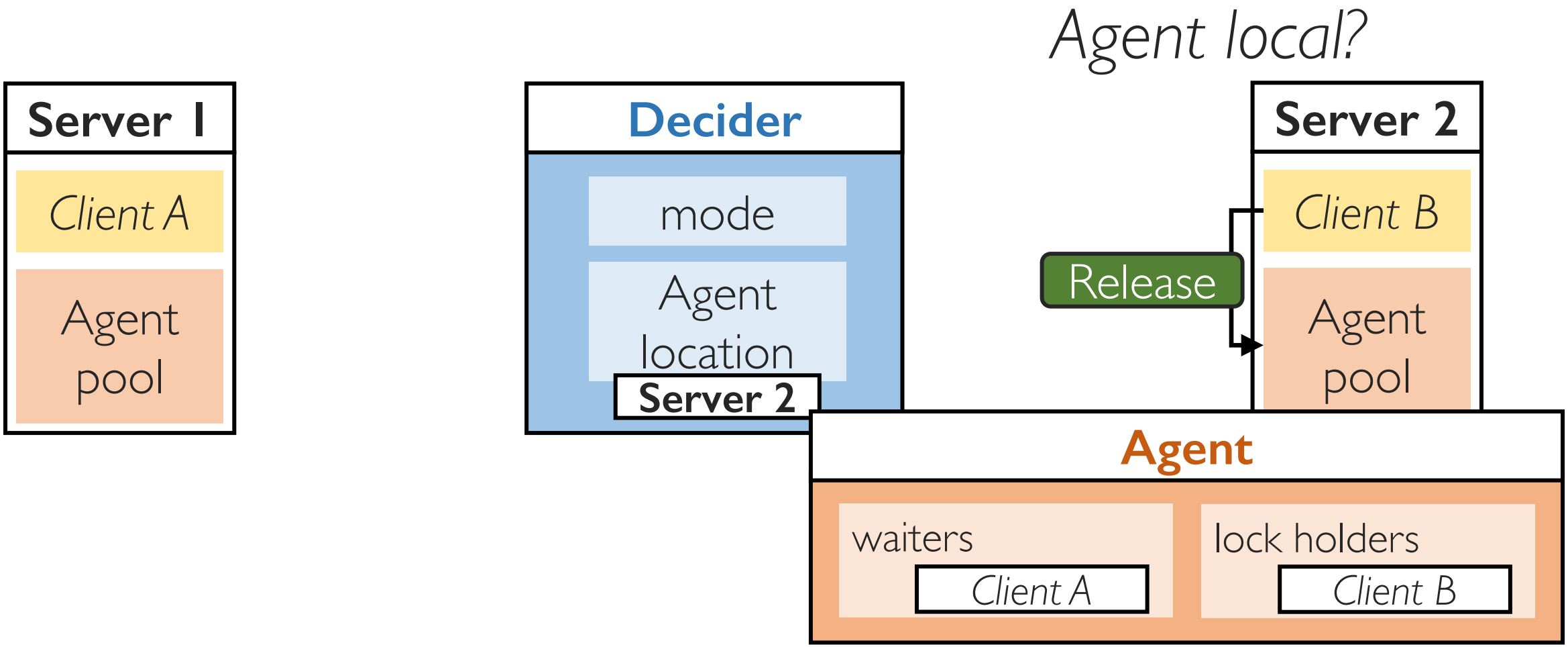
Agent not local?

Lock Release Workflow

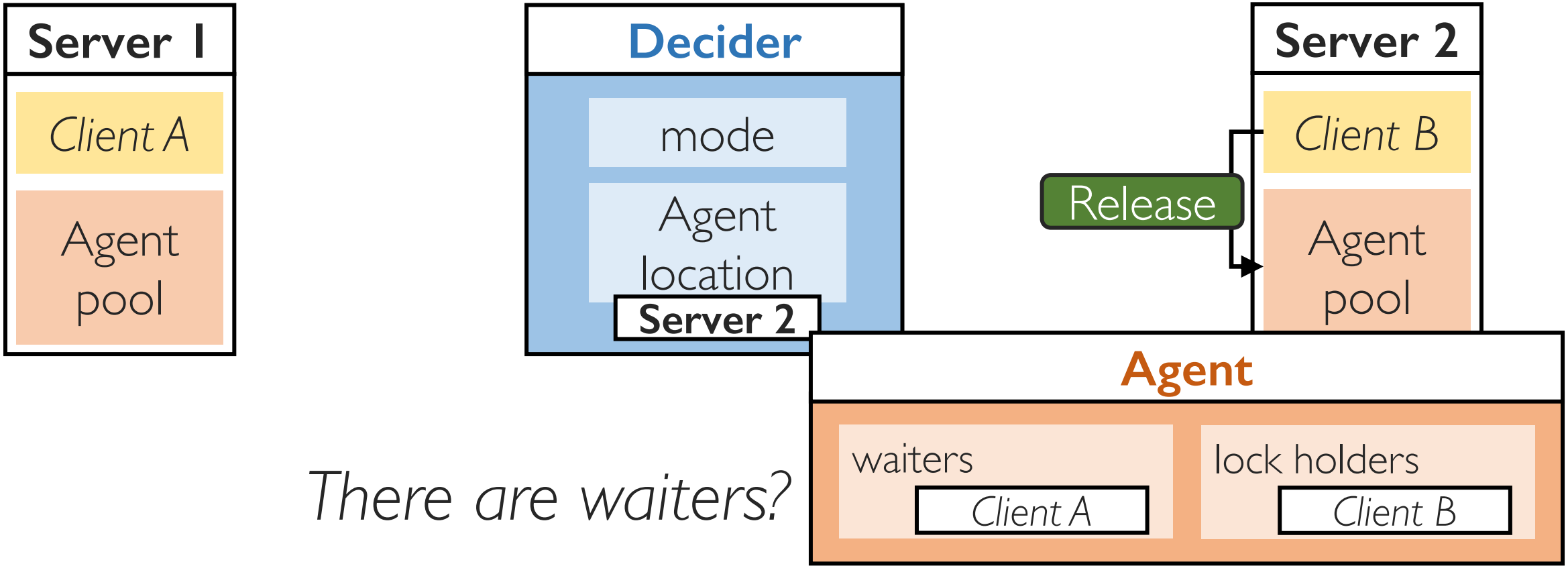


Agent not local?

Lock Release Workflow

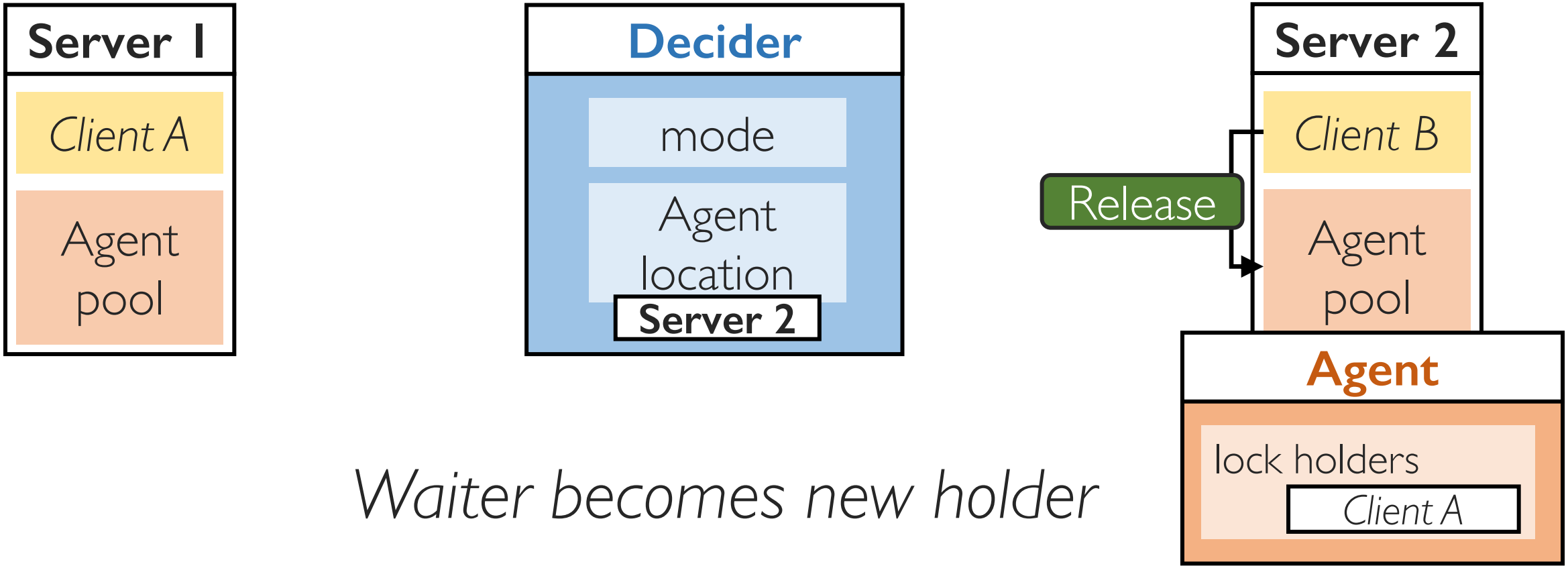


Lock Release Workflow



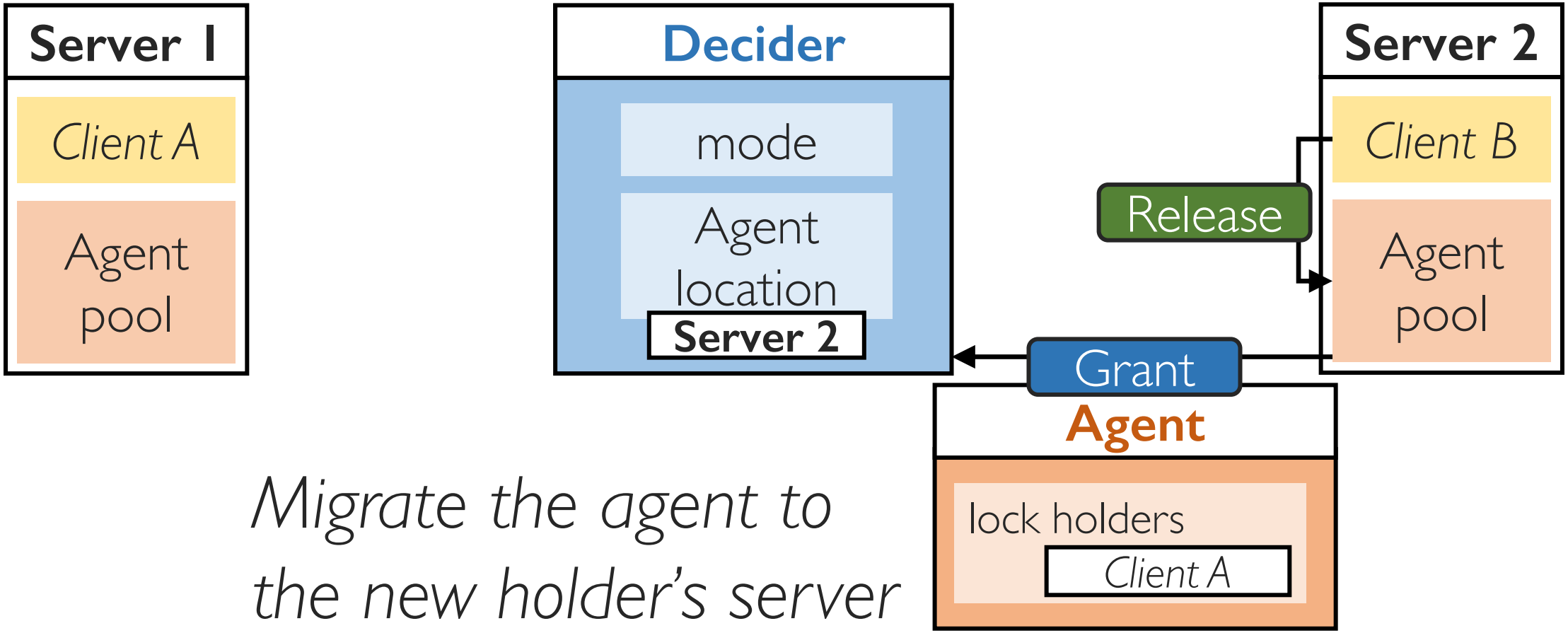
There are waiters?

Lock Release Workflow



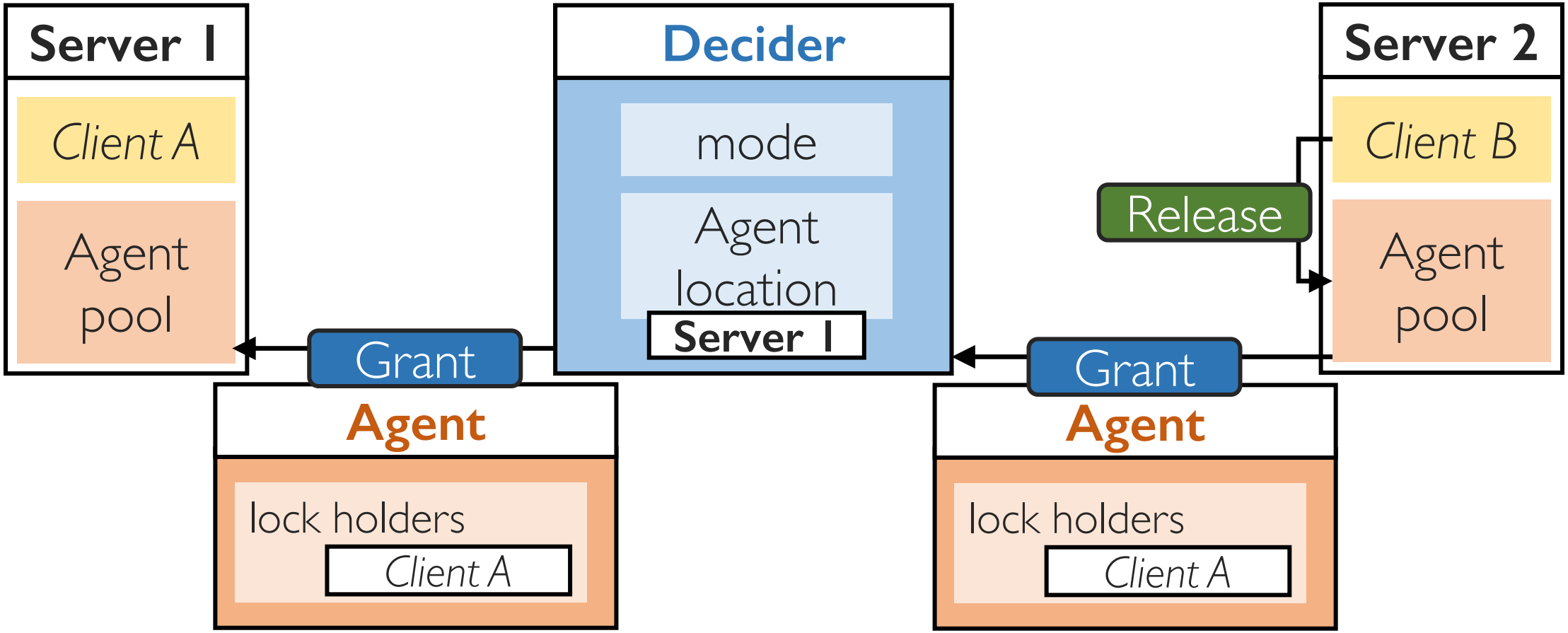
Waiter becomes new holder

Lock Release Workflow

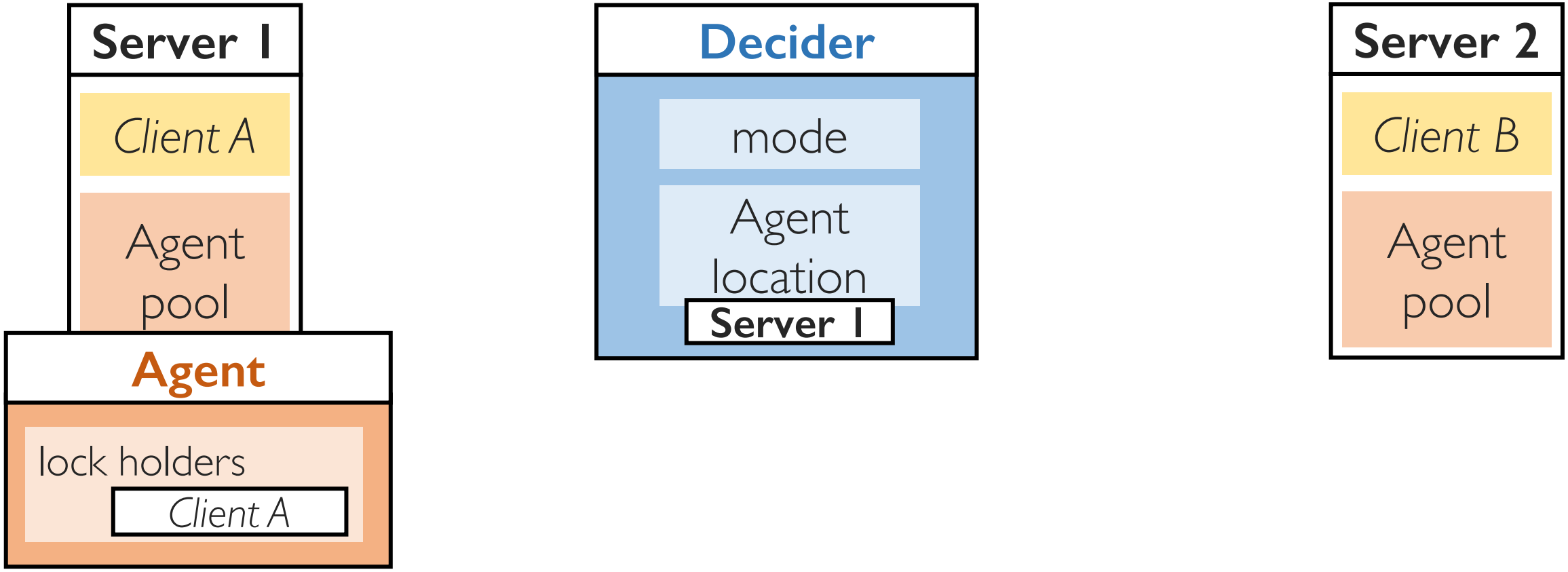


Migrate the agent to the new holder's server

Lock Release Workflow

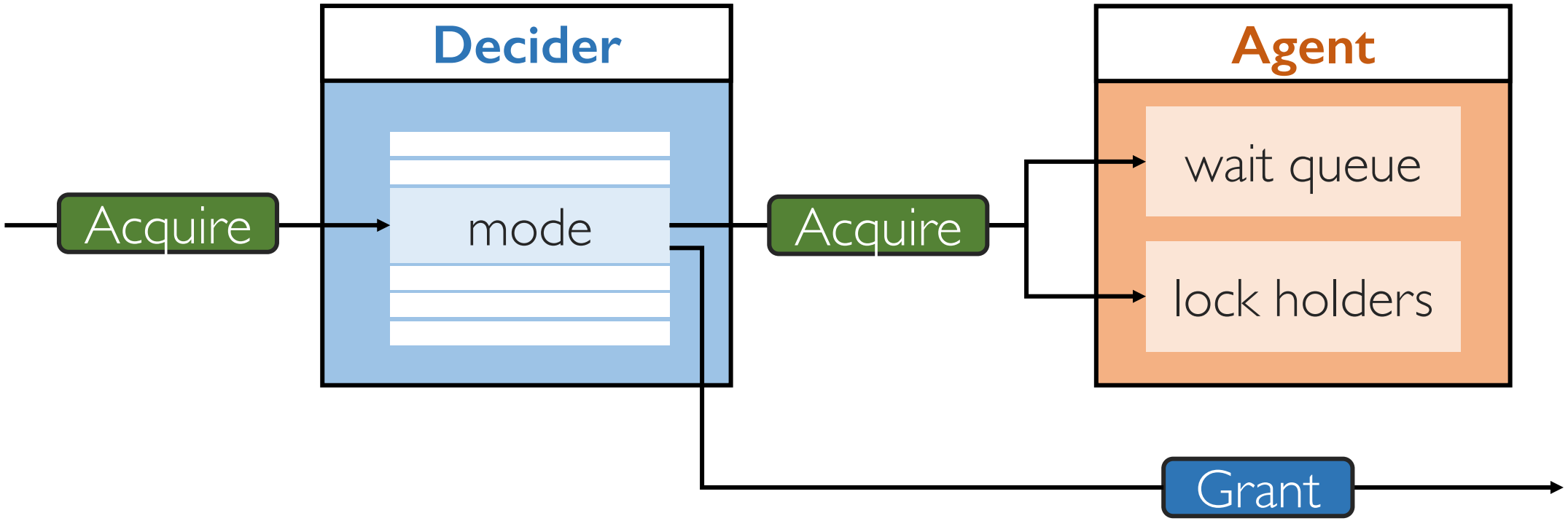


Lock Release Workflow



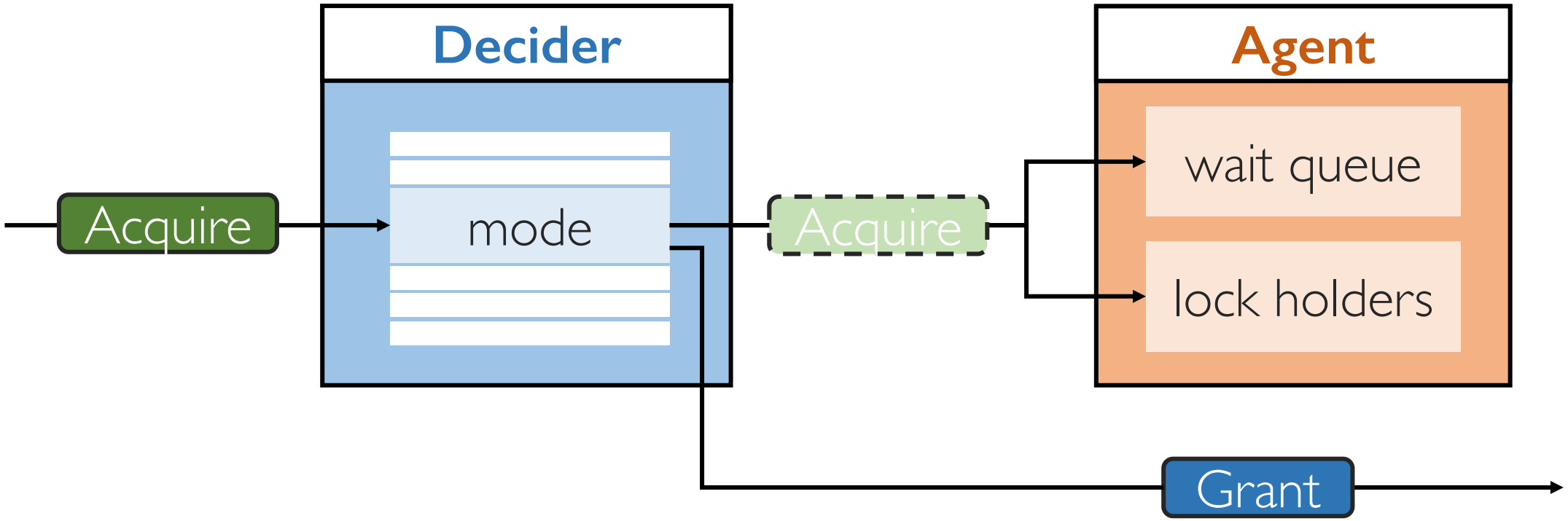
Challenge: Distributed Lock Operations

How to ensure consistency between decider and agent?



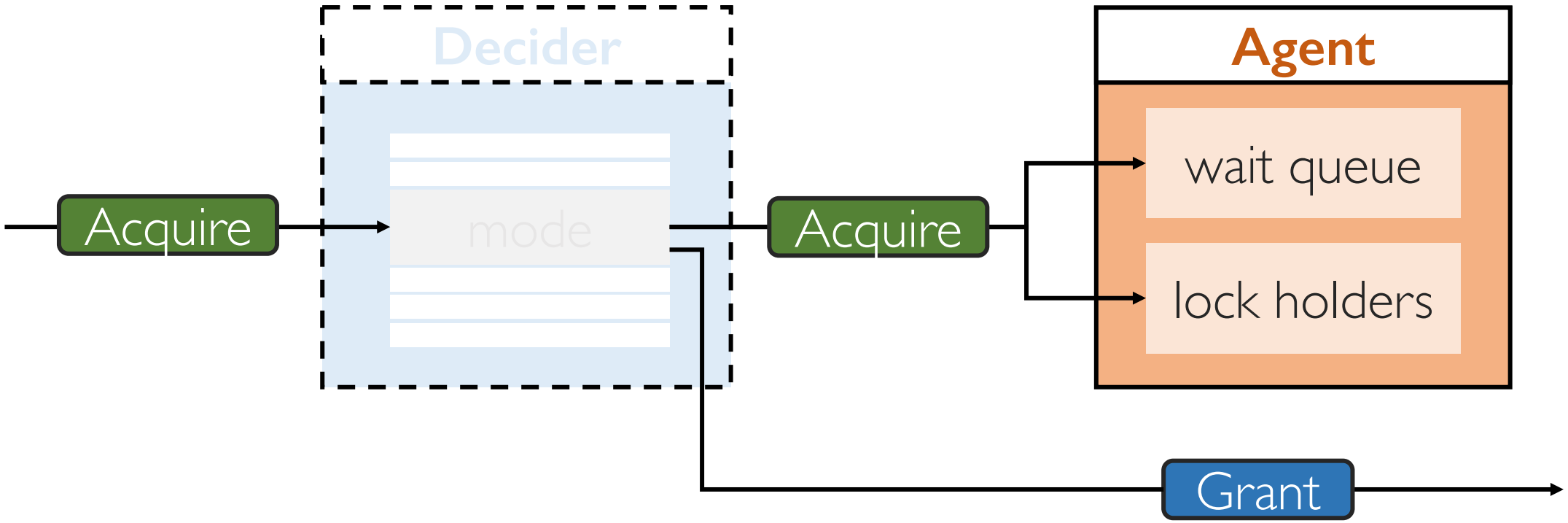
Challenge: Distributed Lock Operations

How to handle lost, reordered, and delayed packets?



Challenge: Distributed Lock Operations

How to handle switch, server, or simultaneous failures?



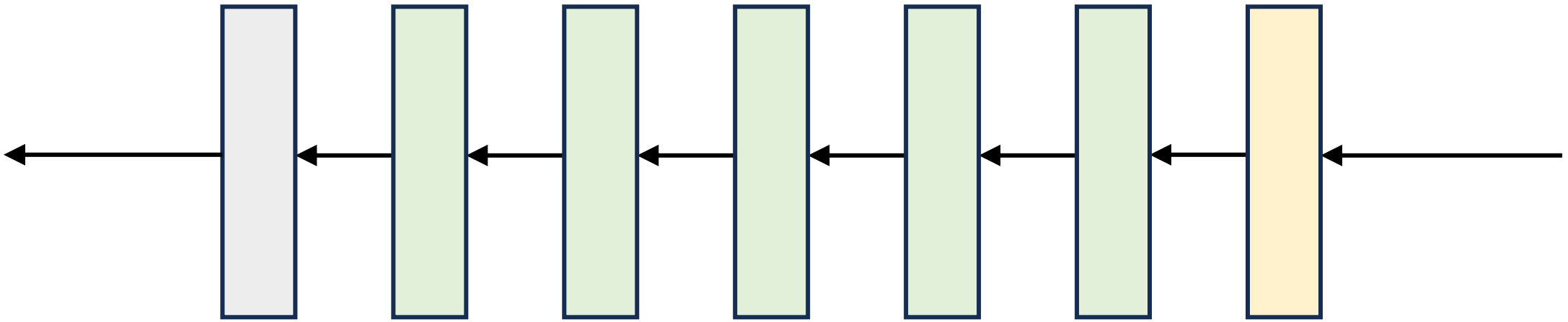
Challenge: Distributed Lock Operations

A new "Lock Fission Protocol" handles them all!

(see details in our paper §4)

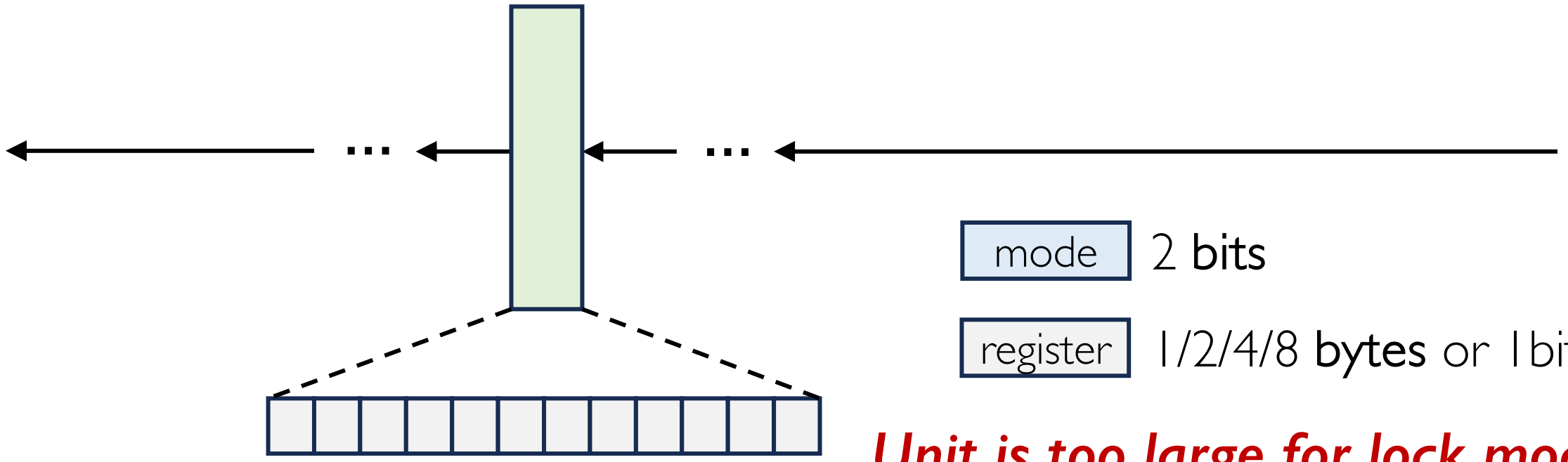
On-switch Decider Design

How to maximize the utilization of limited switch memory (<10MB)?



On-switch Decider Design

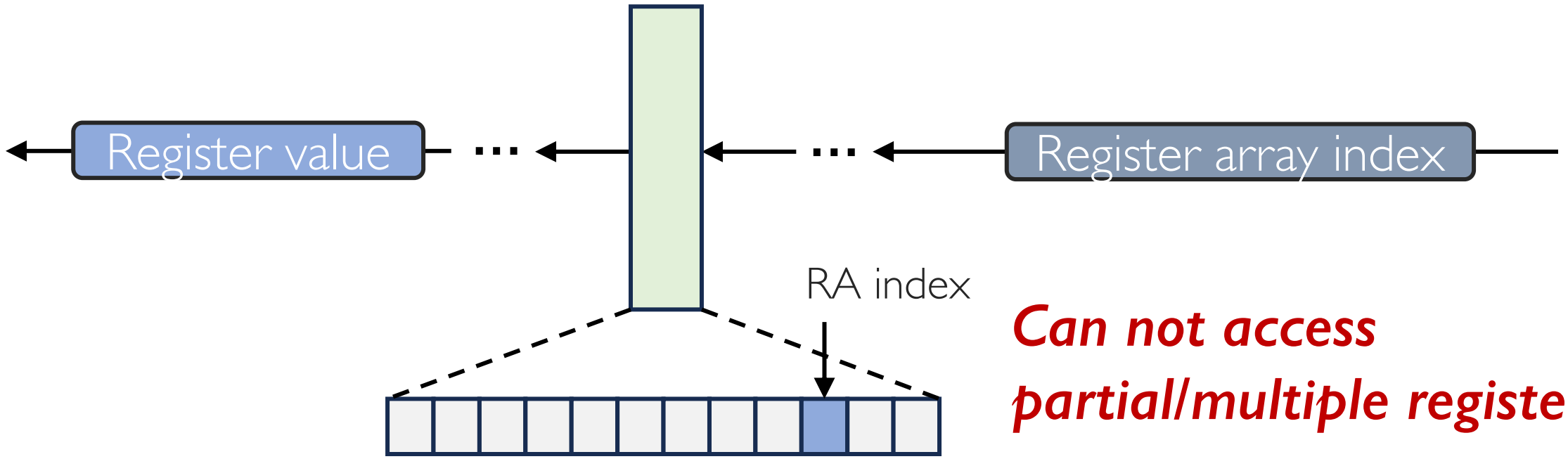
Switch memory is organized as fixed-size registers



Unit is too large for lock mode!

On-switch Decider Design

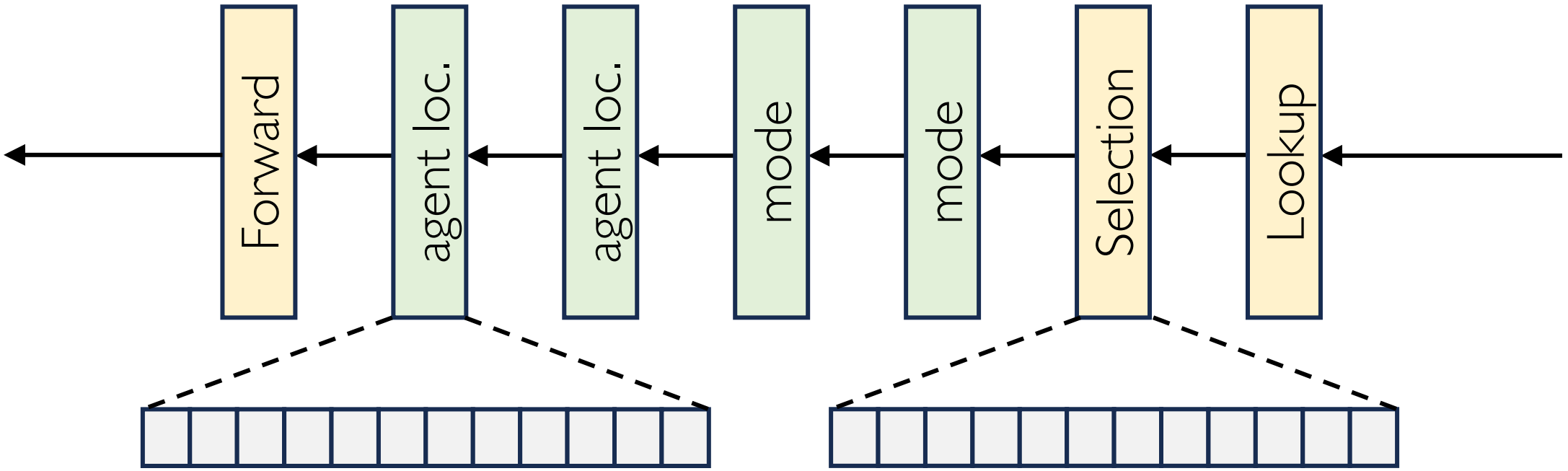
Each pipeline stage allows accessing exactly one register



Can not access partial/multiple registers!

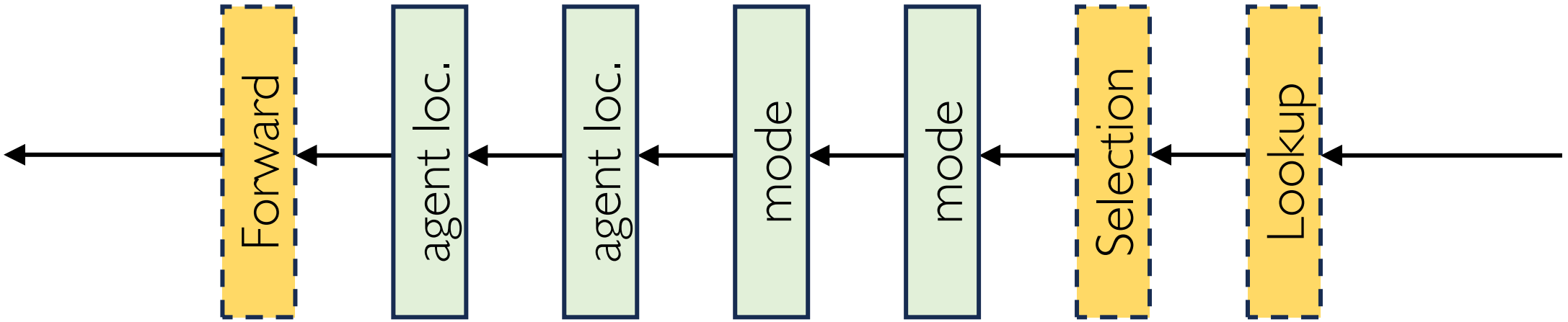
On-switch Decider Design

There is no shared memory across pipeline stages



On-switch Decider Design

There is no shared memory across pipeline stages



Memory used by control stages (e.g., forward) are wasted!

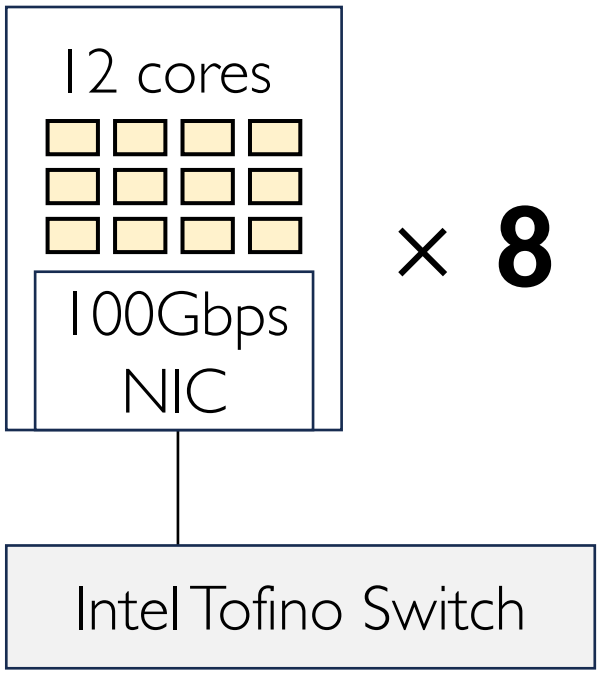
On-switch Decider Design

A pipeline design that supports over 1.5M locks!

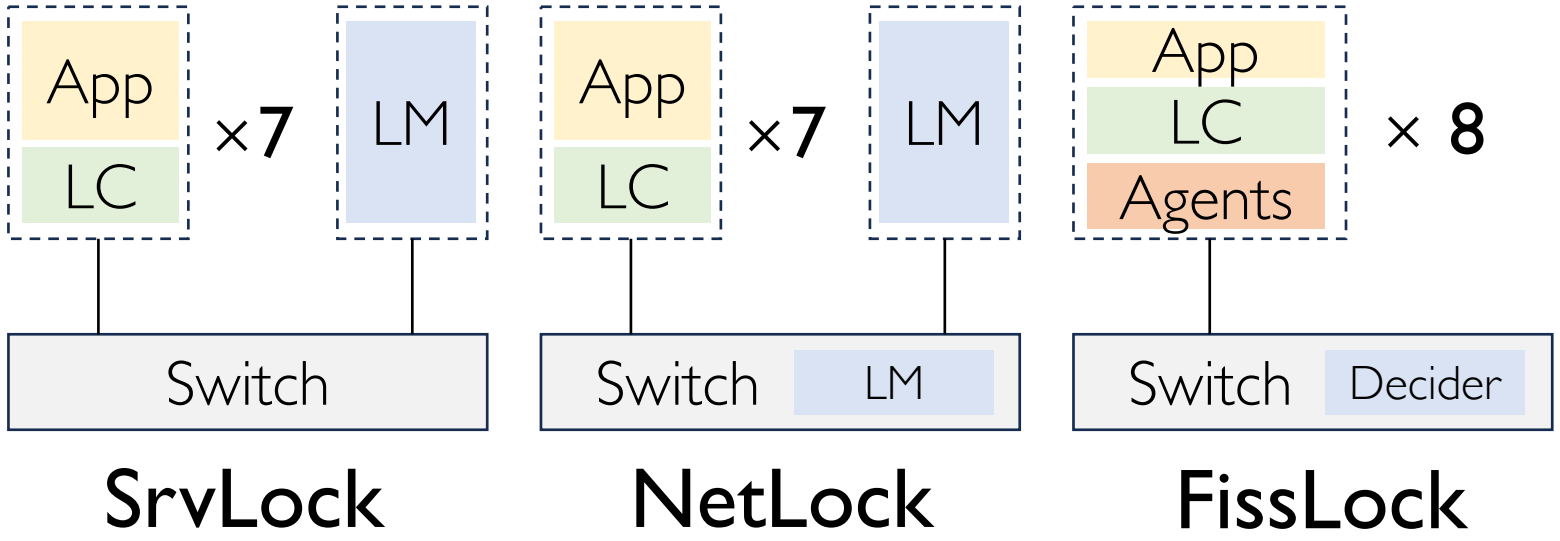
(see details in our paper §5.2)

Evaluation Setup

Testbed

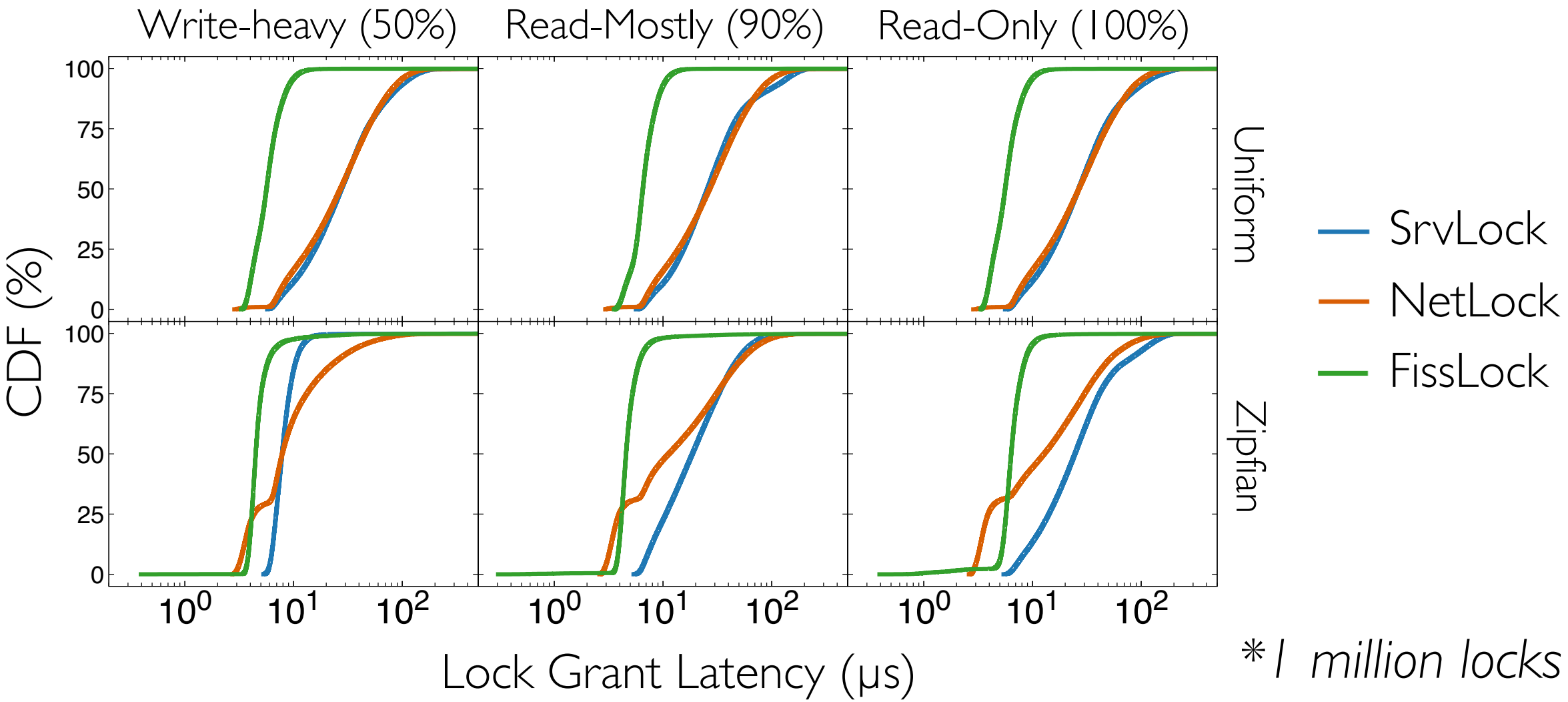


Comparing targets



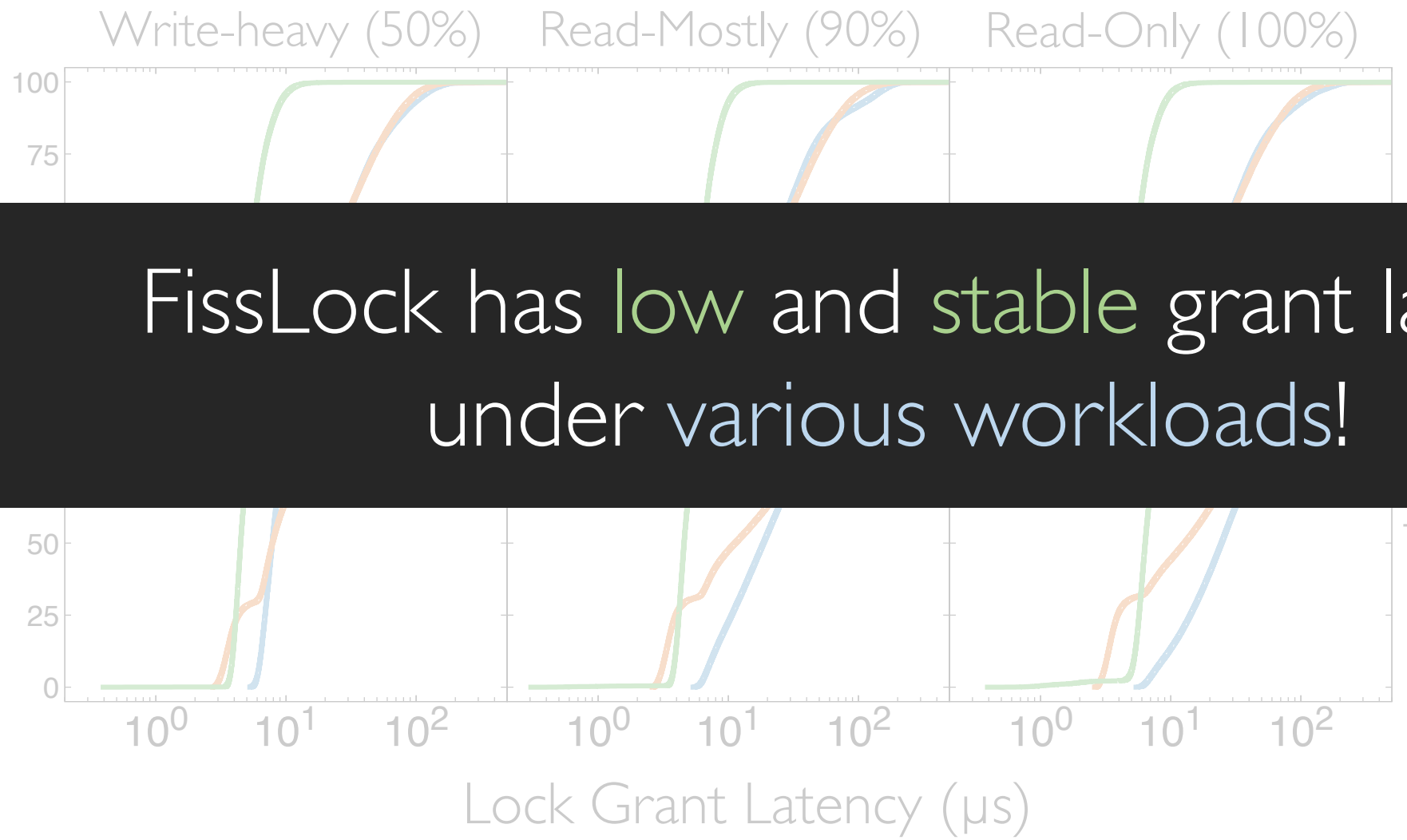
**All systems use 80 cores in total for LC and 8 cores for LM / Agents*

Microbenchmark: Is FissLock Fast?



*1 million locks

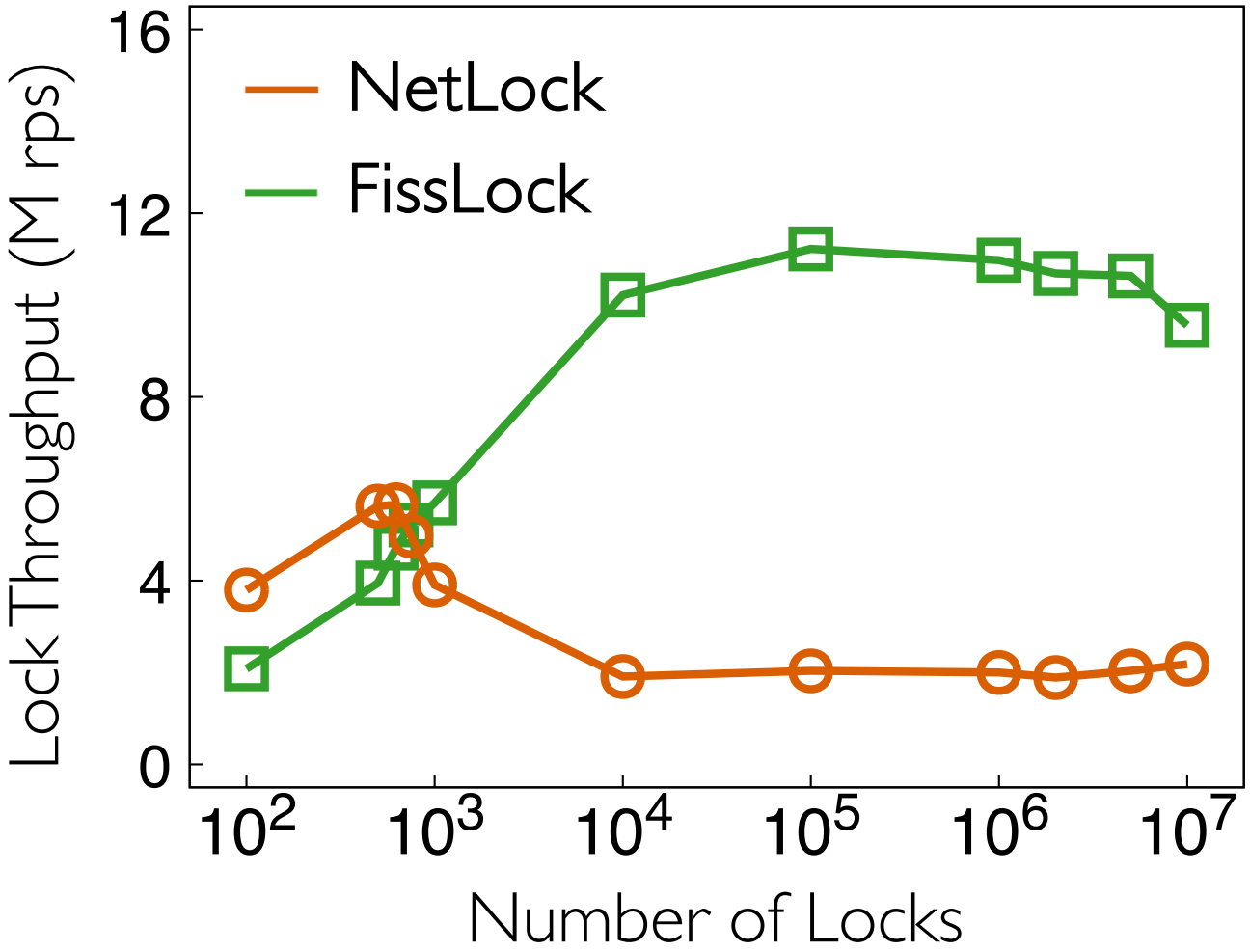
Microbenchmark: Is FissLock Fast?



FissLock has low and stable grant latency under various workloads!

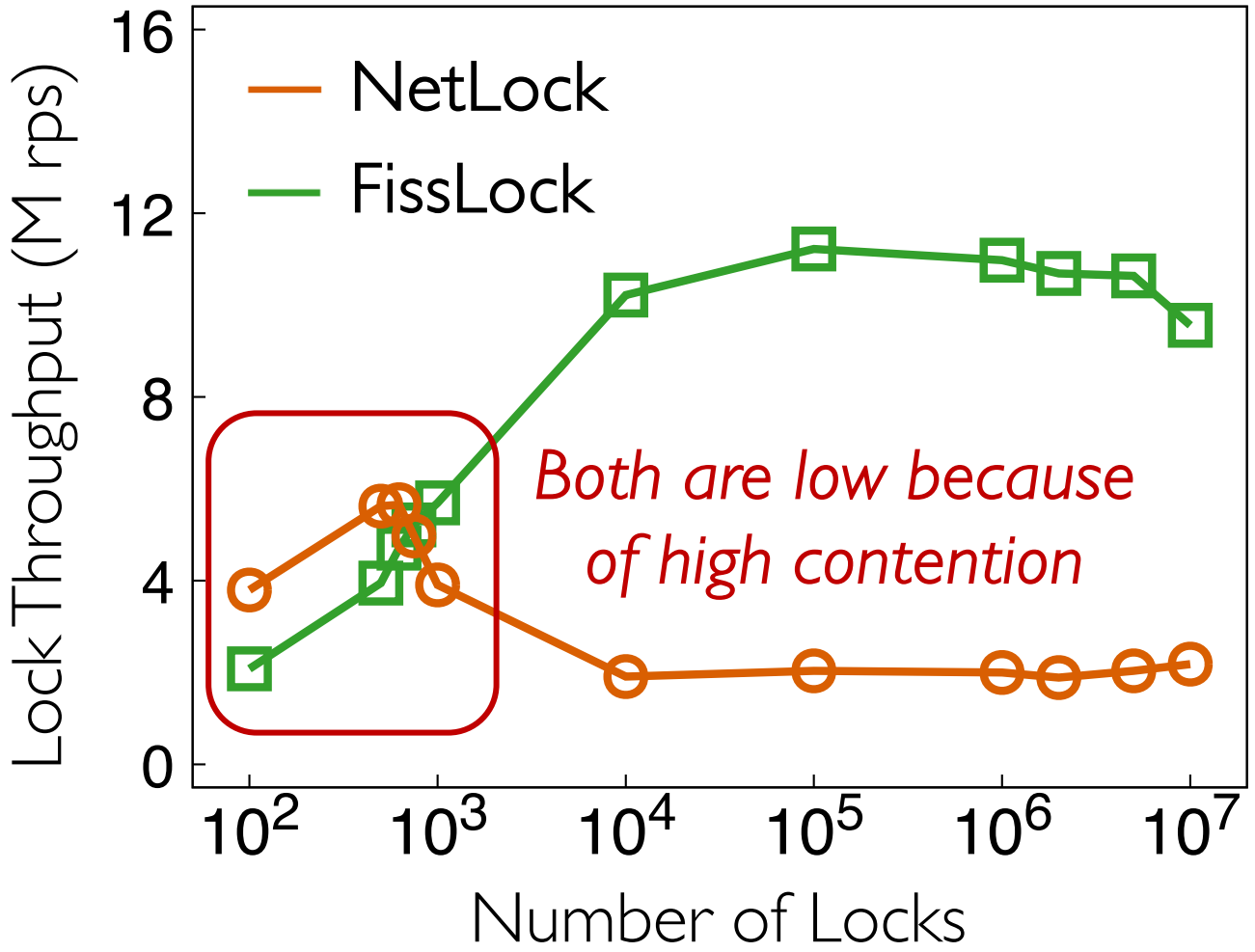
*1 million locks

Microbenchmark: Is FissLock Scalable?



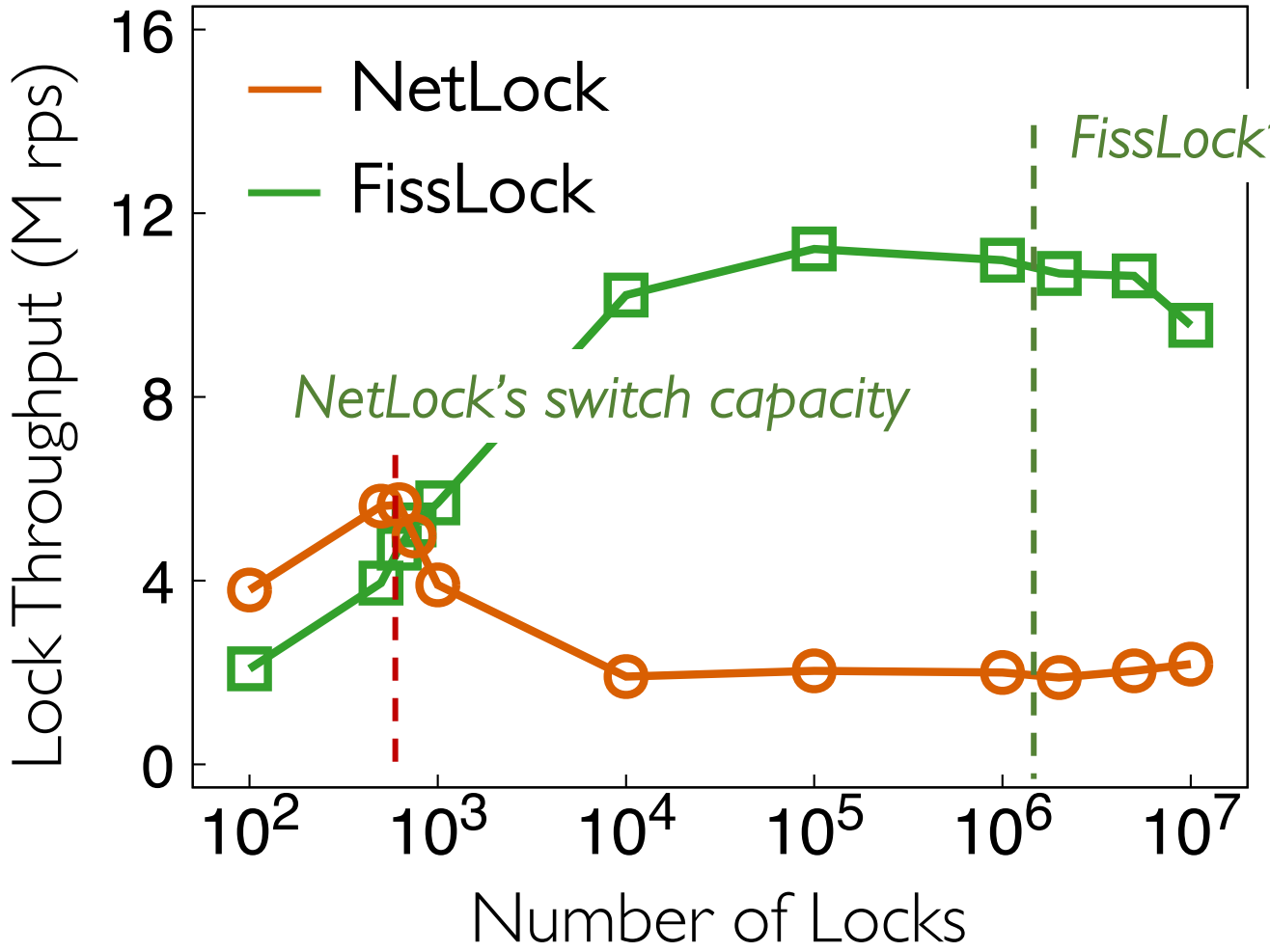
**Uniform RM workload with 160 clients*

Microbenchmark: Is FissLock Scalable?



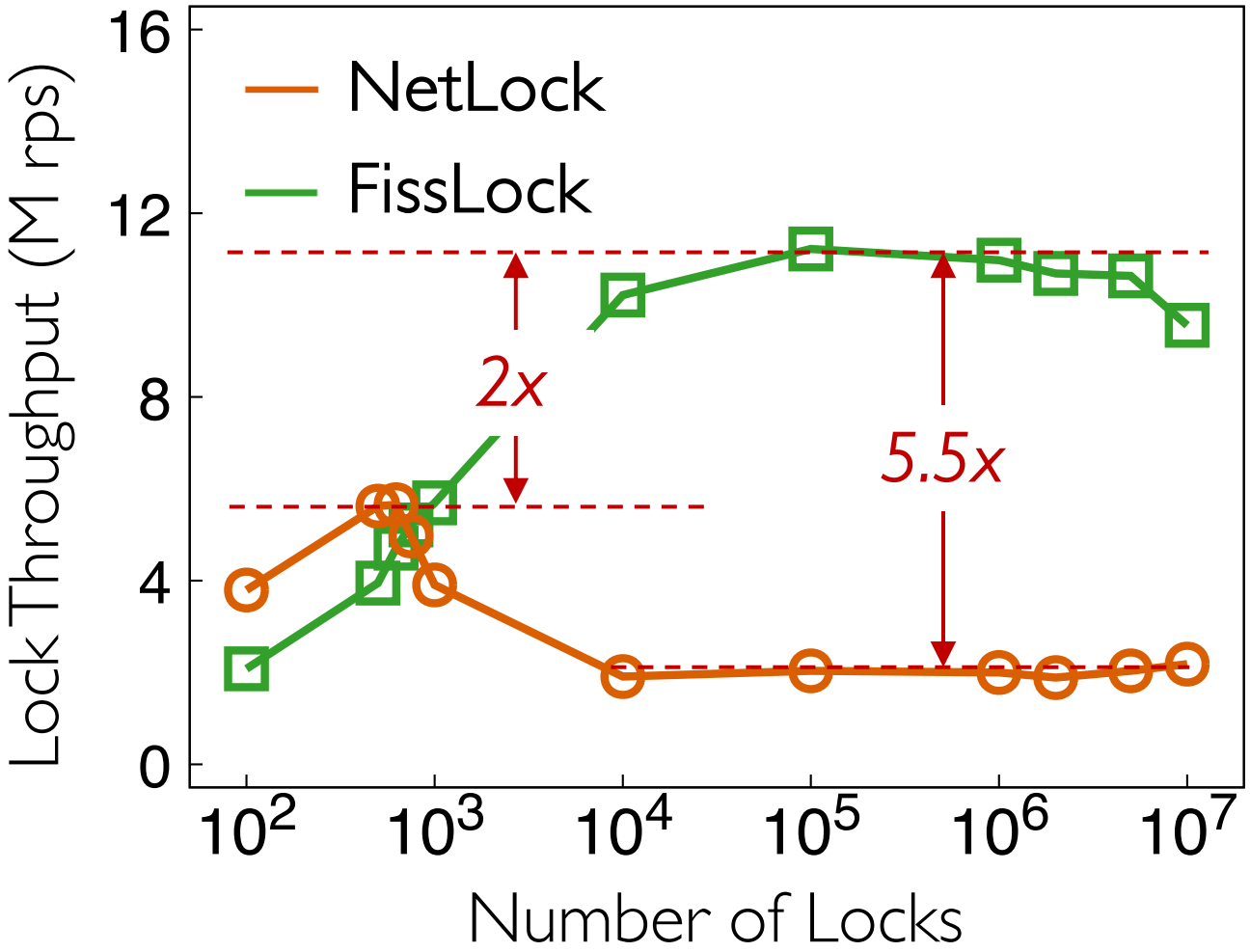
*Uniform RM workload with 160 clients

Microbenchmark: Is FissLock Scalable?



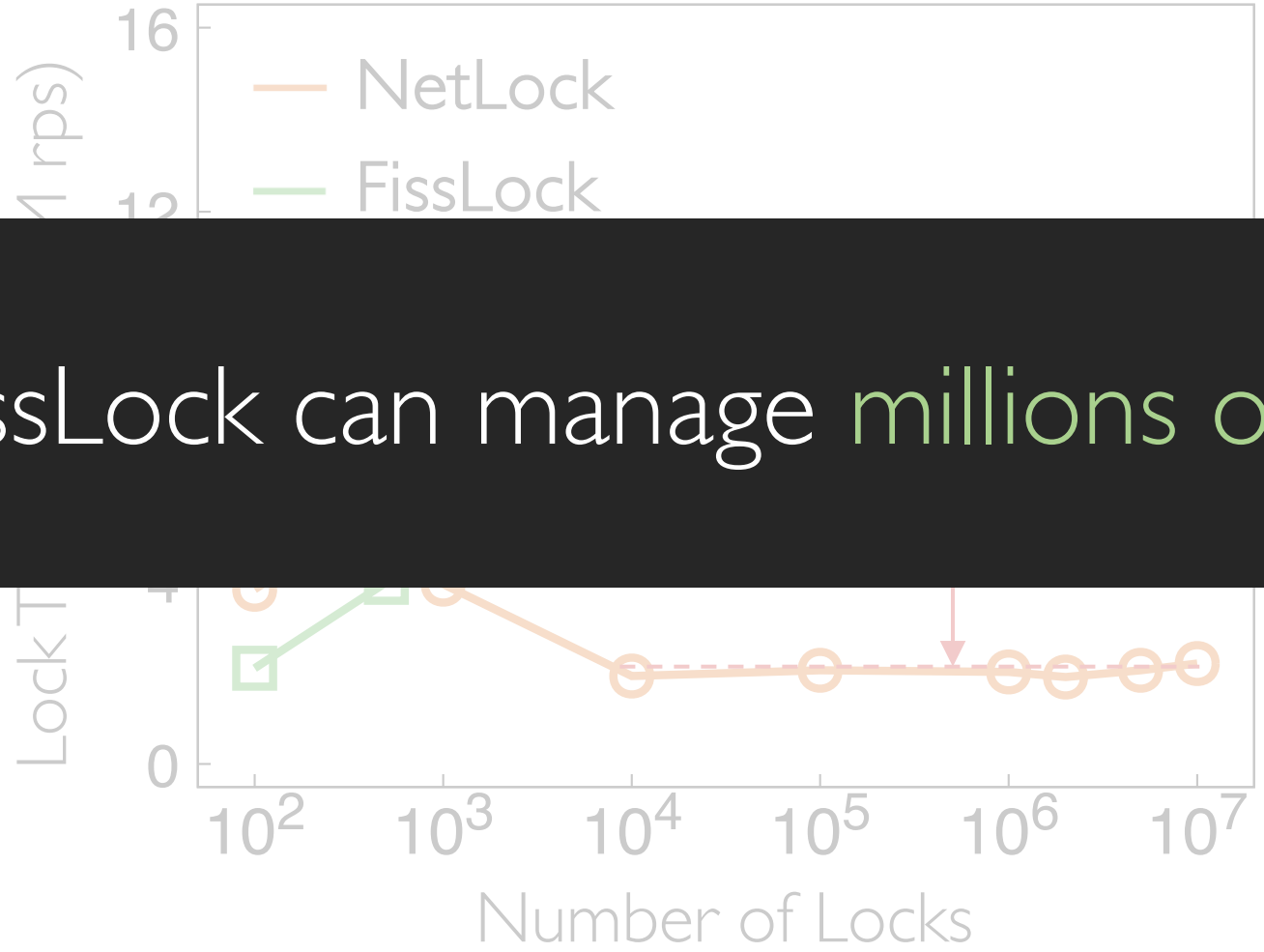
**Uniform RM workload with 160 clients*

Microbenchmark: Is FissLock Scalable?



**Uniform RM workload with 160 clients*

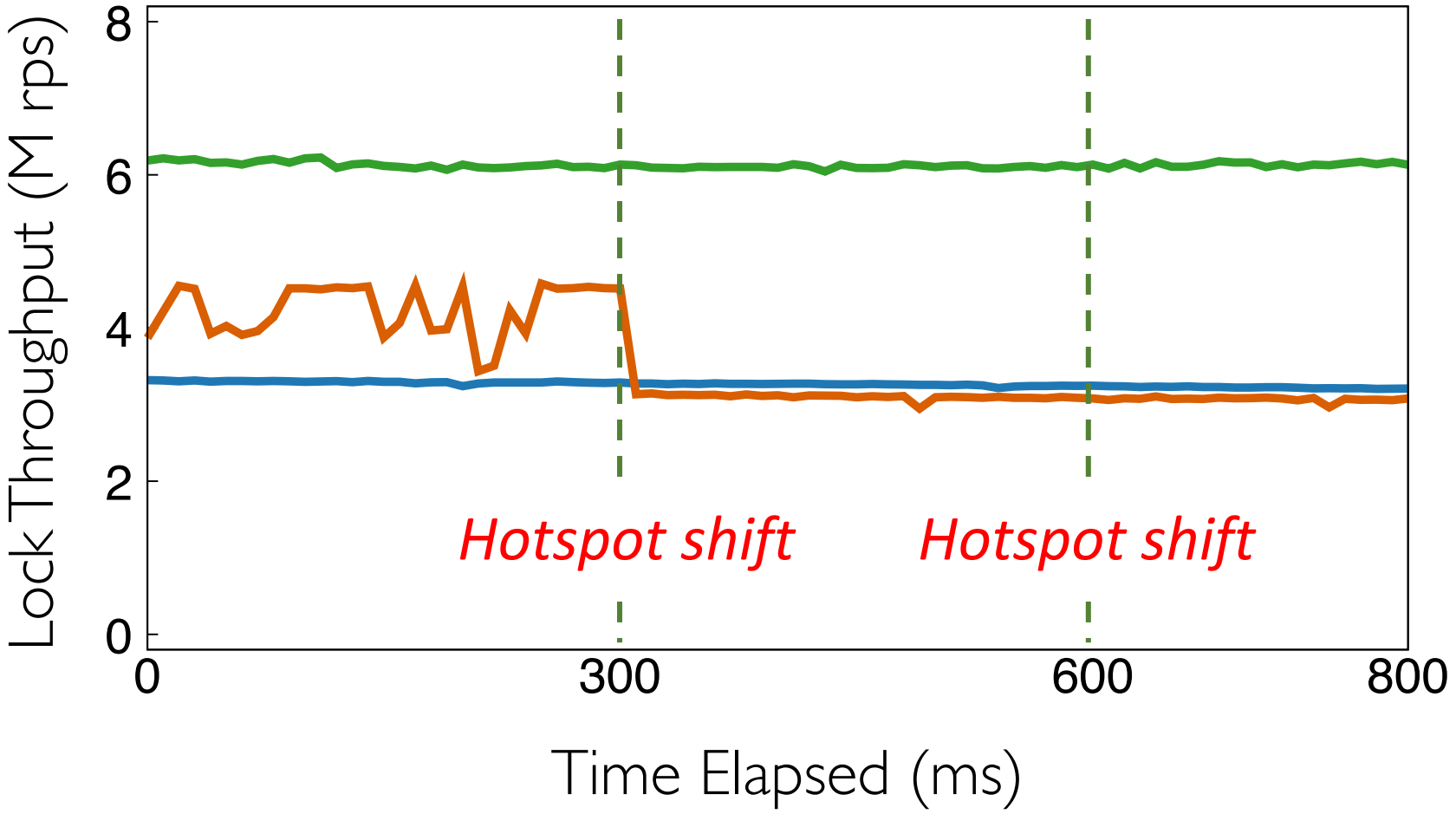
Microbenchmark: Is FissLock Scalable?



FissLock can manage millions of locks efficiently!

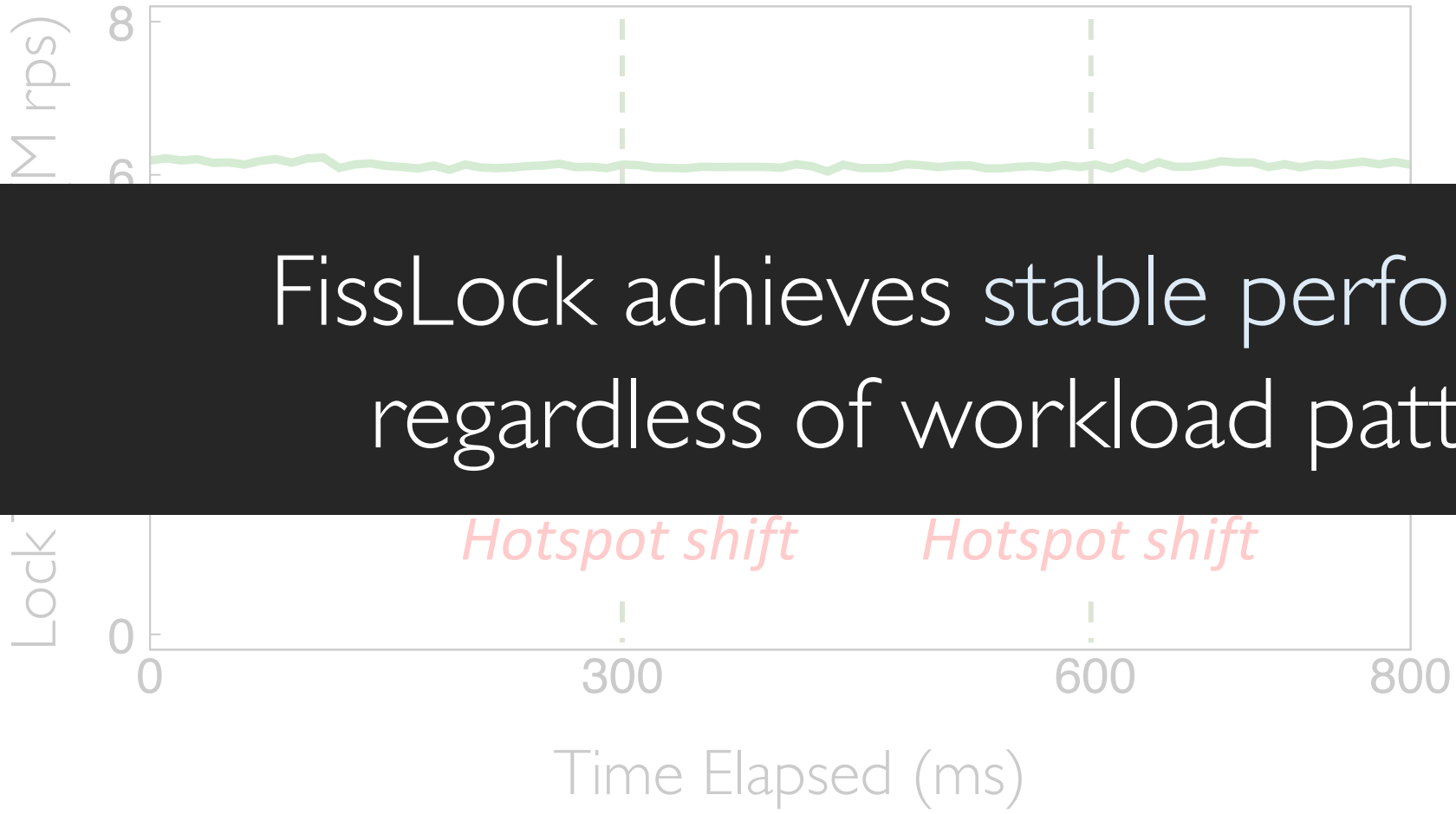
*Uniform RM workload with 160 clients

Microbenchmark: Is FissLock Stable?



*1 million locks
50% requests to
2500 hot locks

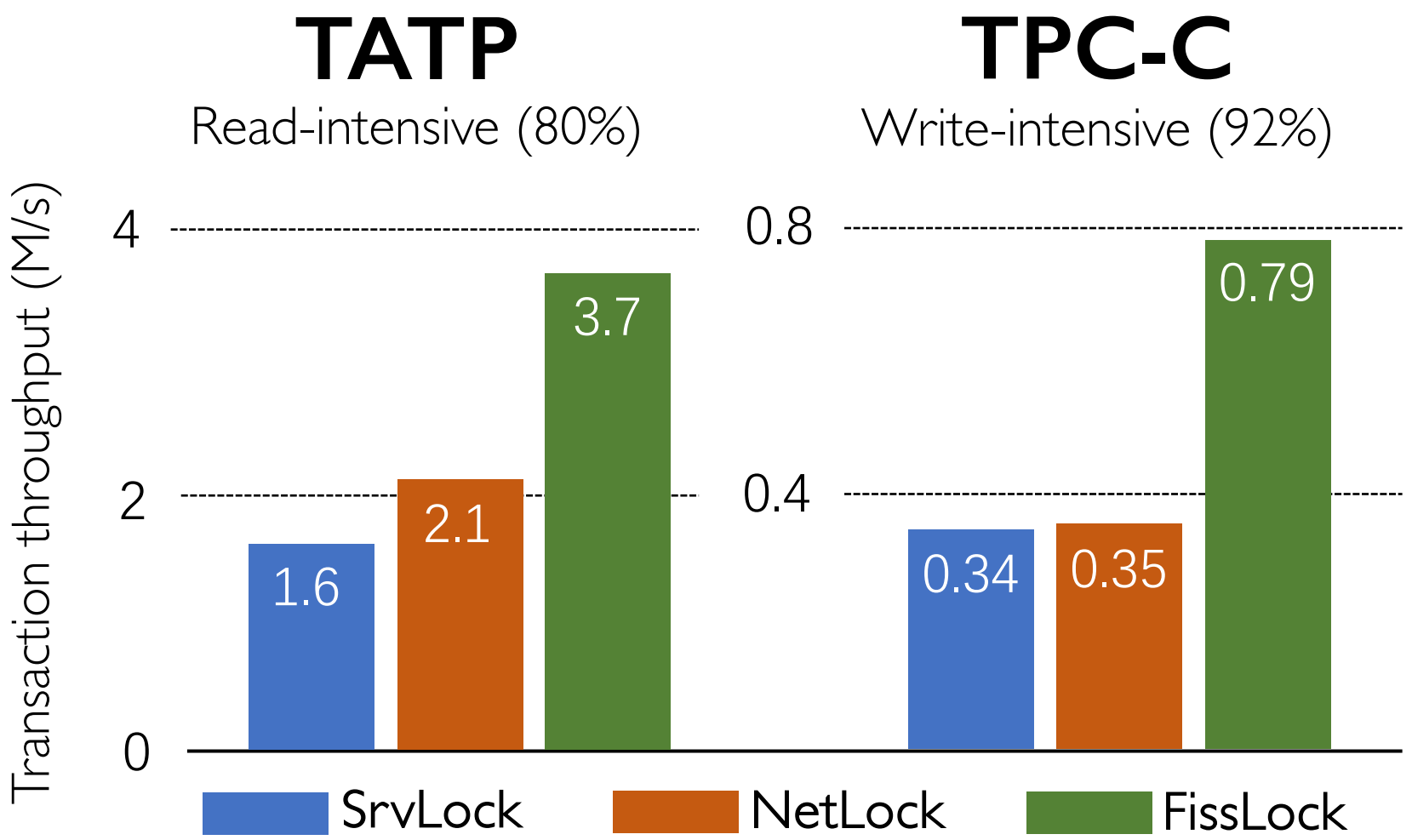
Microbenchmark: Is FissLock Stable?



FissLock achieves stable performance regardless of workload patterns!

*1 million locks
50% requests to
2500 hot locks

Transaction Benchmark



Conclusion

- Distributed lock services need to be *fast* and *scalable*
- Key technique: ***lock fission*** (decouple *grant decision* and *data maintenance*)
- Challenges:
  - Distributed lock operations (§4 *lock fission protocol*)
  - Memory-efficient on-switch design (§5.2 *on-switch decider*)
- Evaluation: 90% tail latency cut, 2x transaction throughput boost,

  scales to millions of locks