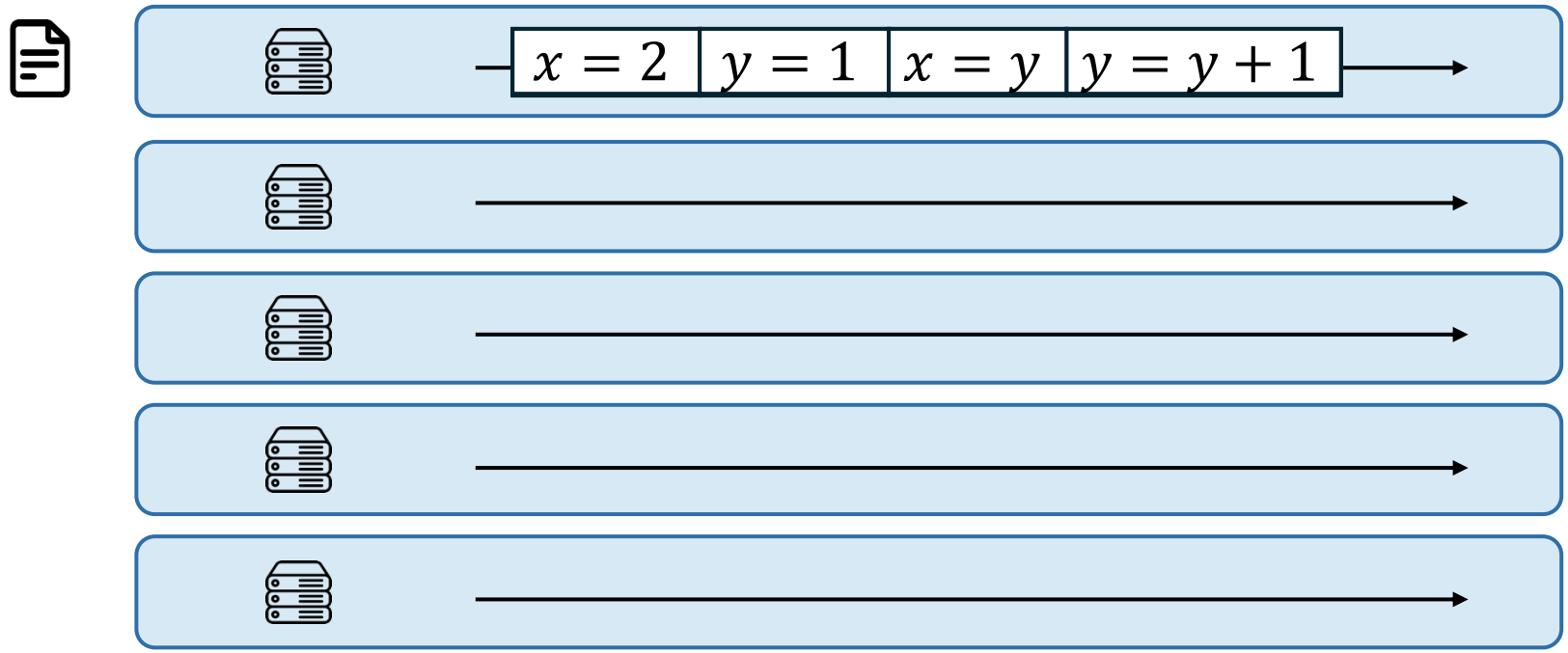
Jetpack

Consensus Made Generally Fast

Ze Tang[1], Zihao Zhang[1], Weihai Shen[1], Jicheng Shi[2], Shuai Mu[1]

[1]Stony Brook University [2]DatenLord

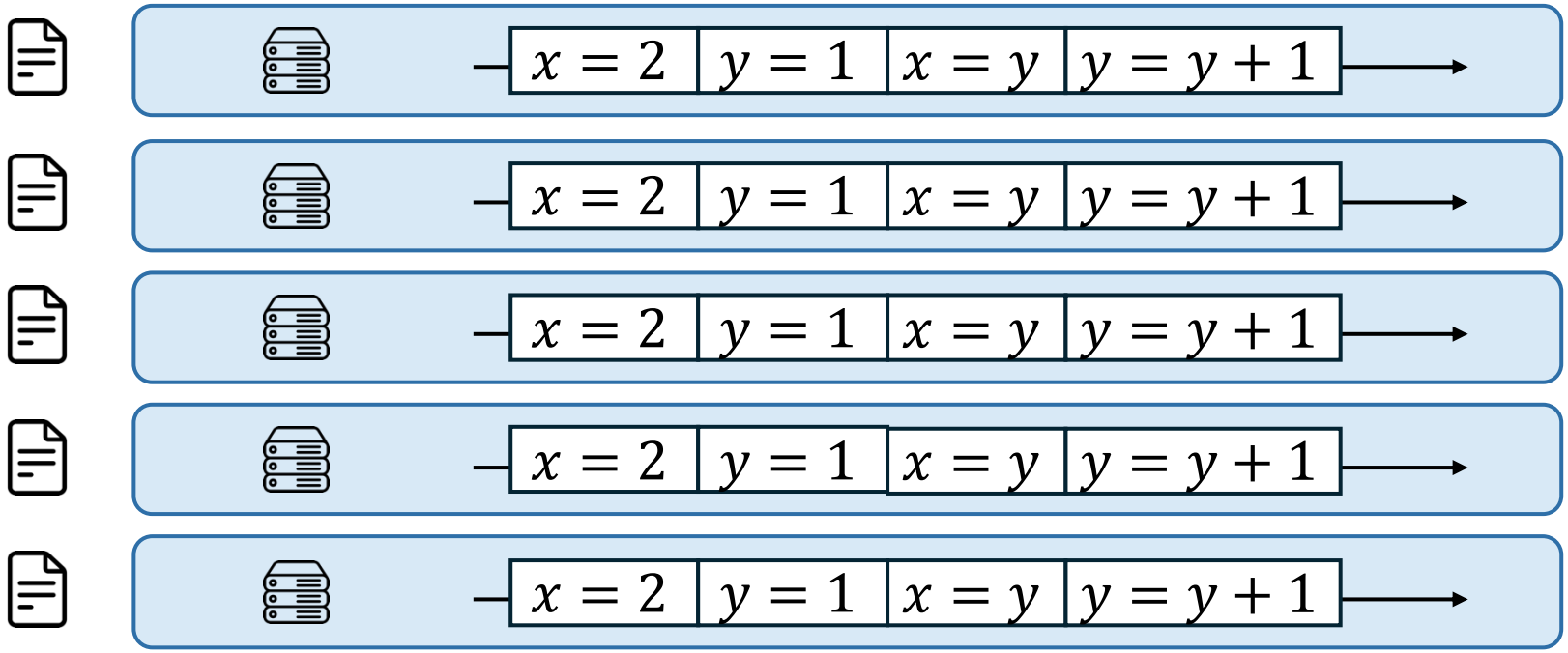
Consensus



Consensus



Leader

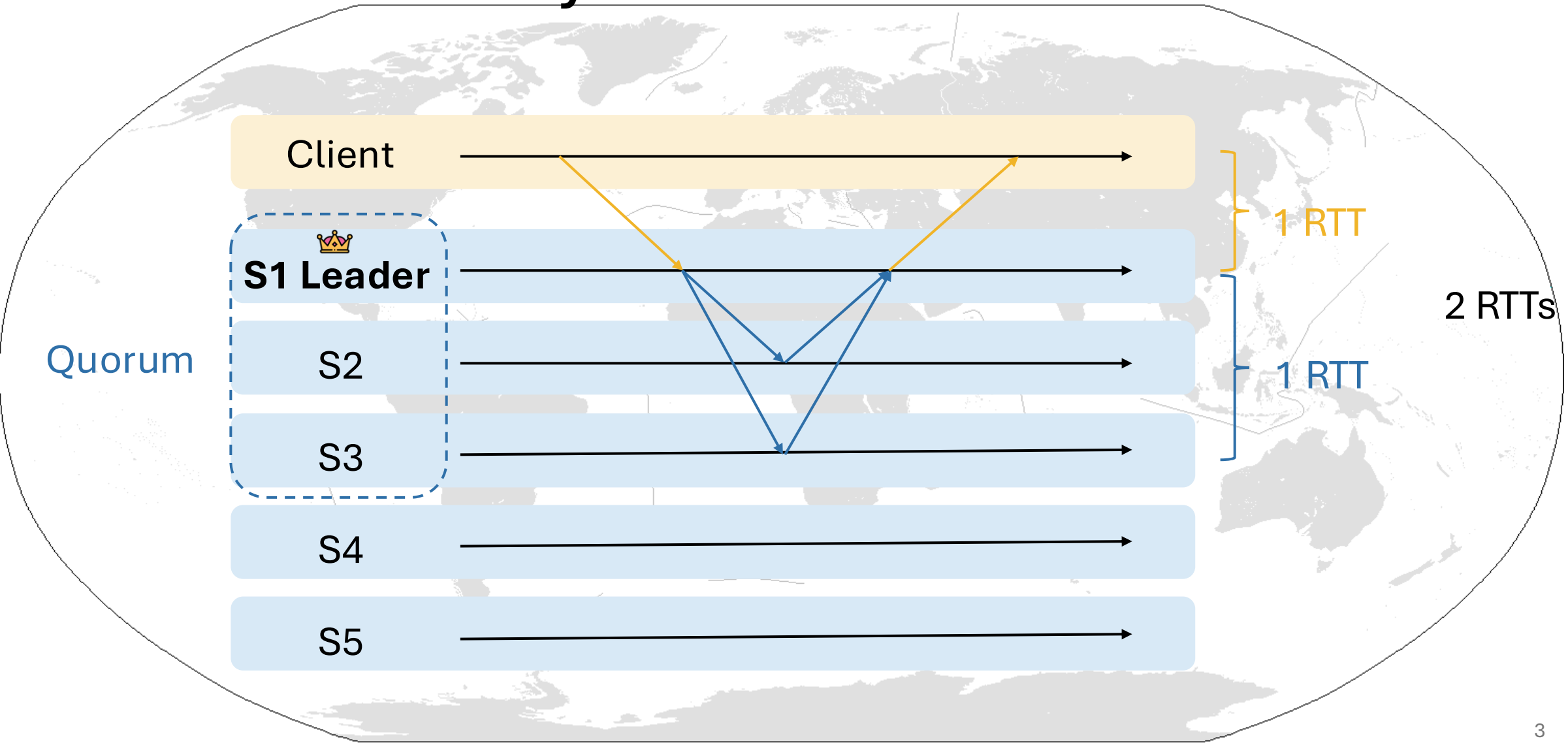


Consistency ✓

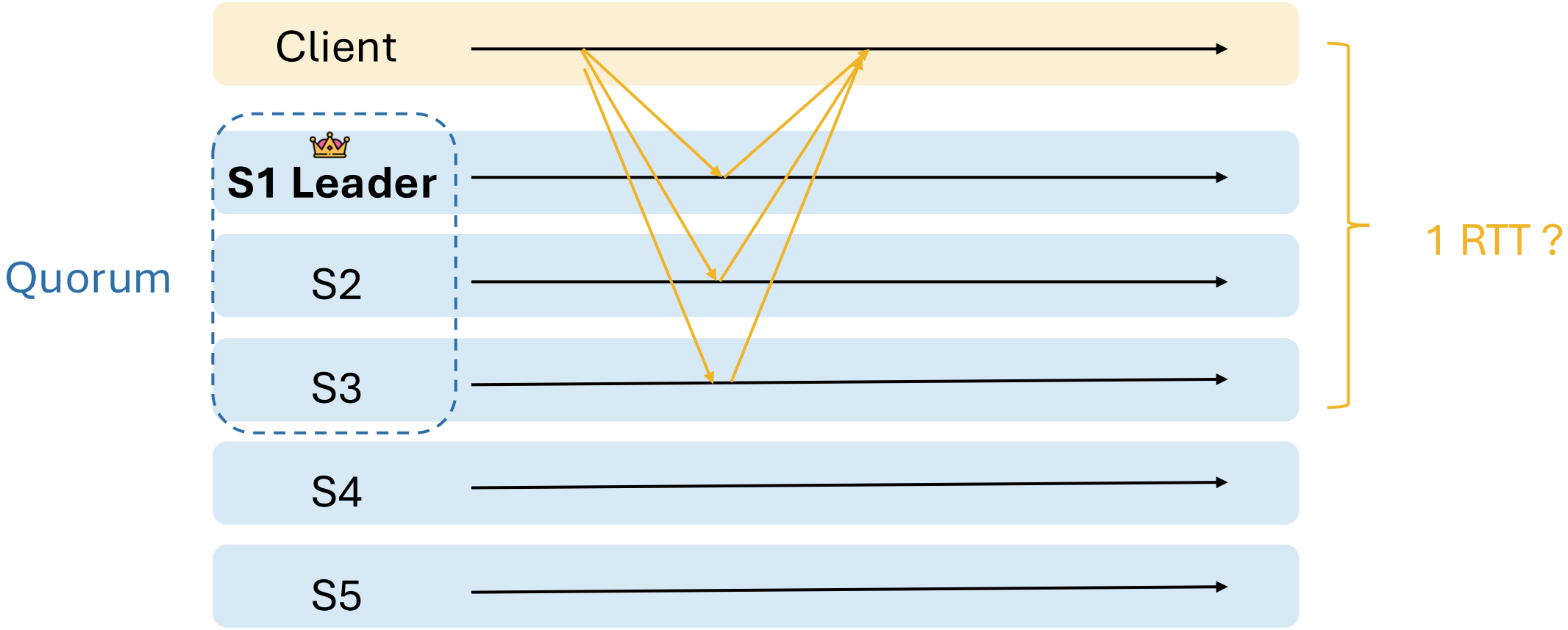
Fault tolerance ✓

Latency ?

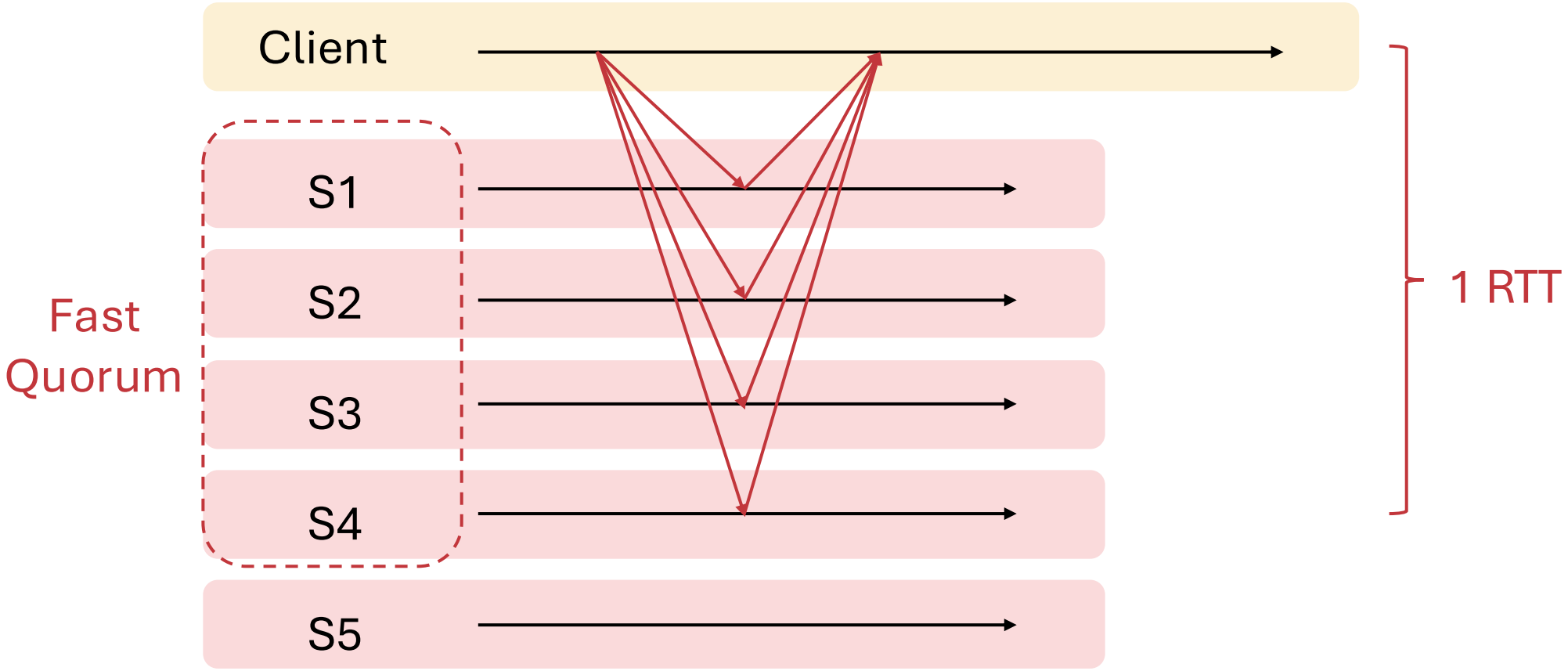
2 RTTs Latency



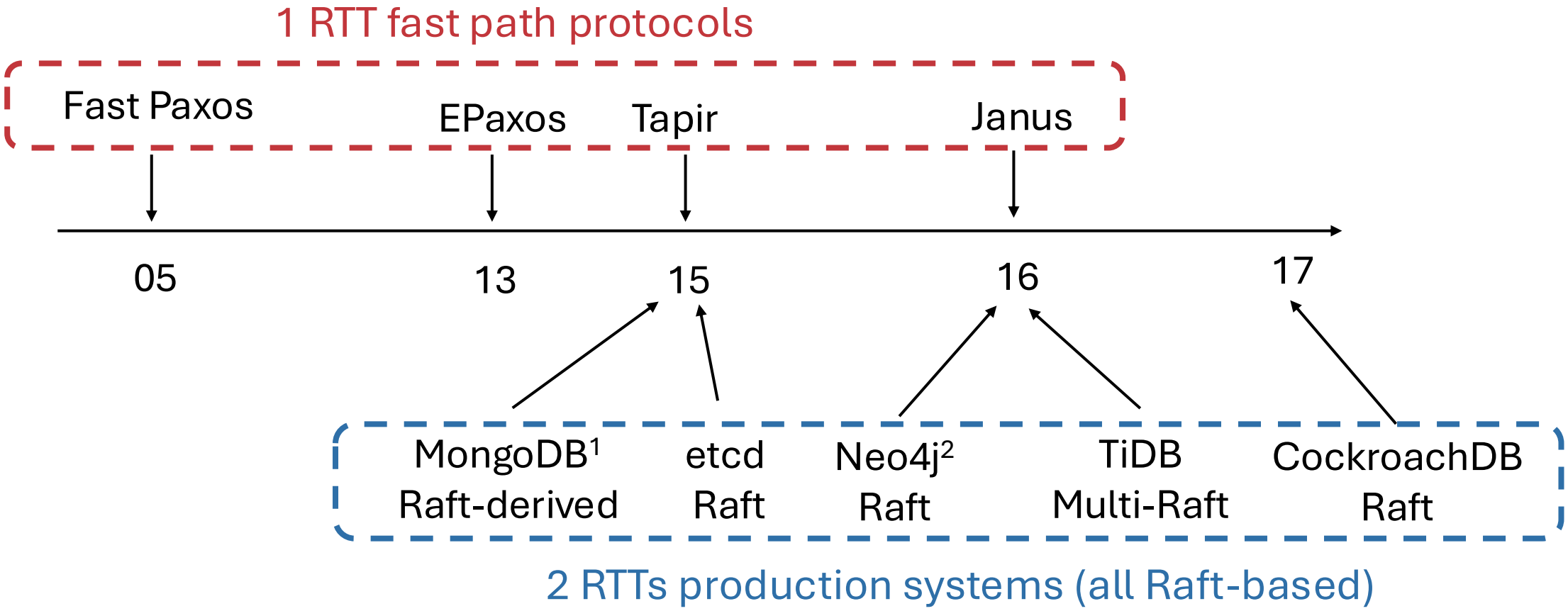
2 RTTs Latency



Fast Paxos, Lamport 2005



A huge gap between two systems



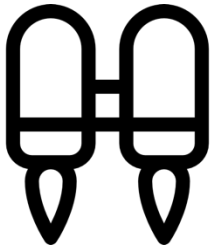
[1]MongoDB's consensus version was first released in 2015
[2]Neo4j's consensus version was first released in 2016

Can we develop a general fast path to benefit all?

Goals 1 - general 1 RTT fast path



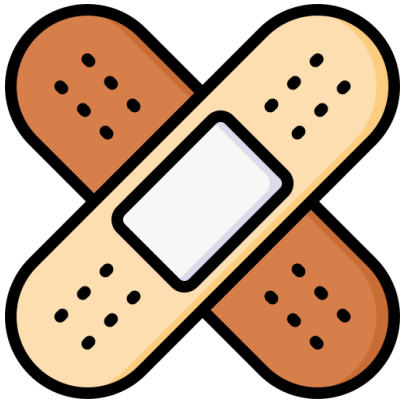
And many more…

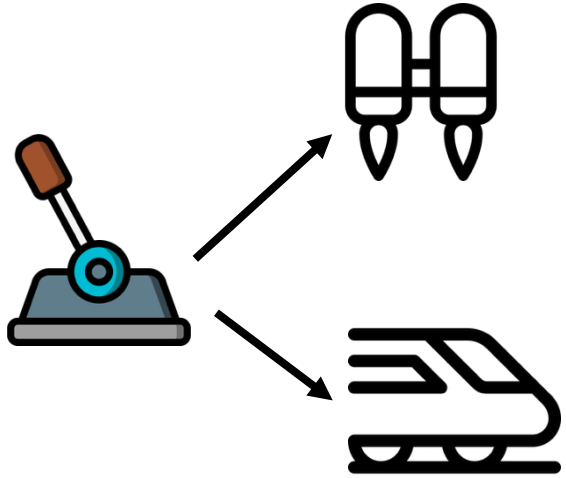


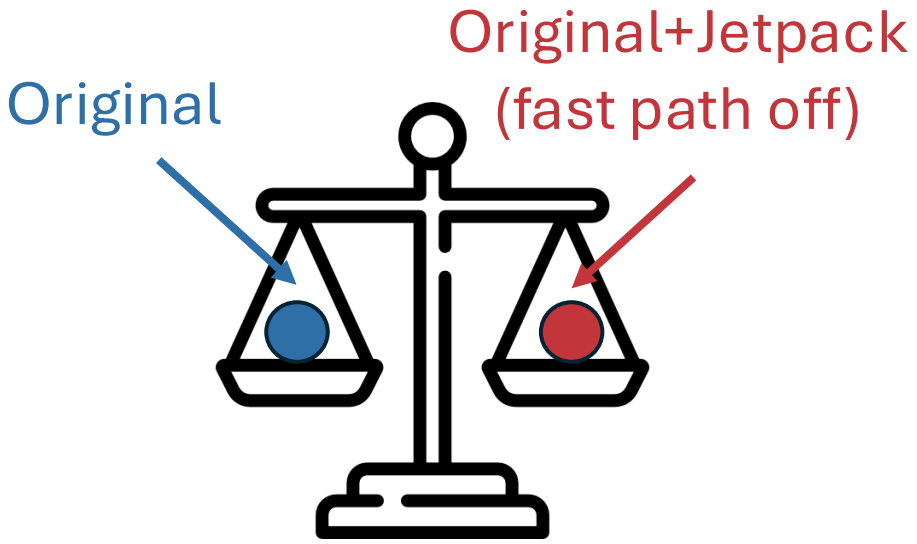


And many more…

Goal: a practically meaningful plugin approach





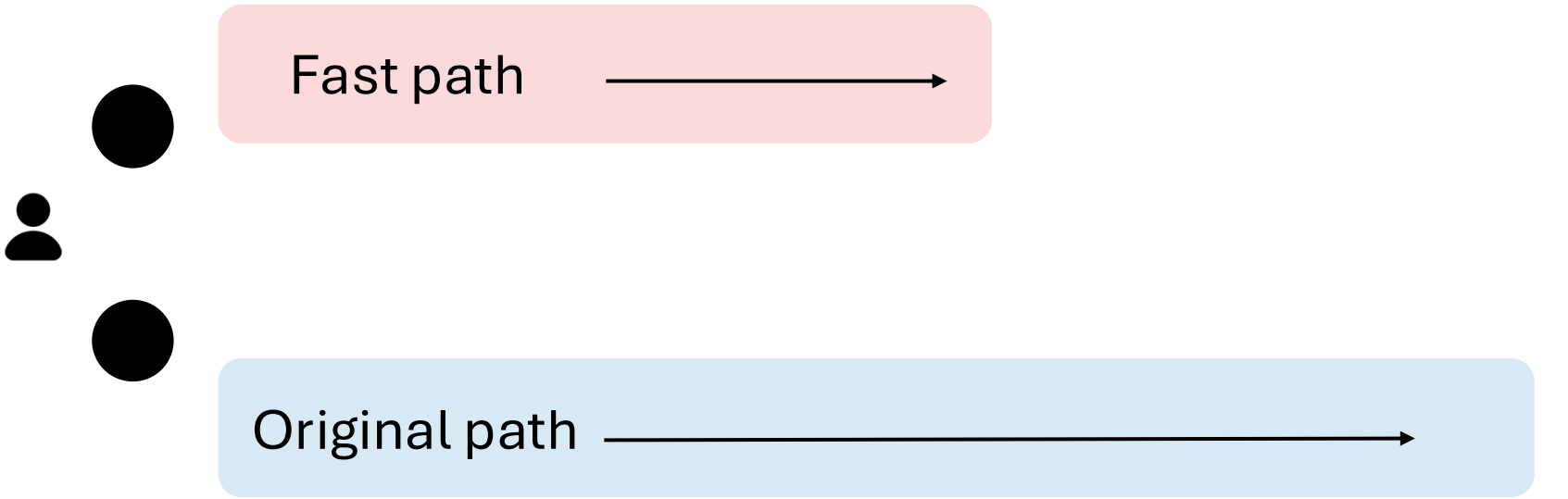


(1) Small changes

(2) Client choose path freely

(3) Low overhead

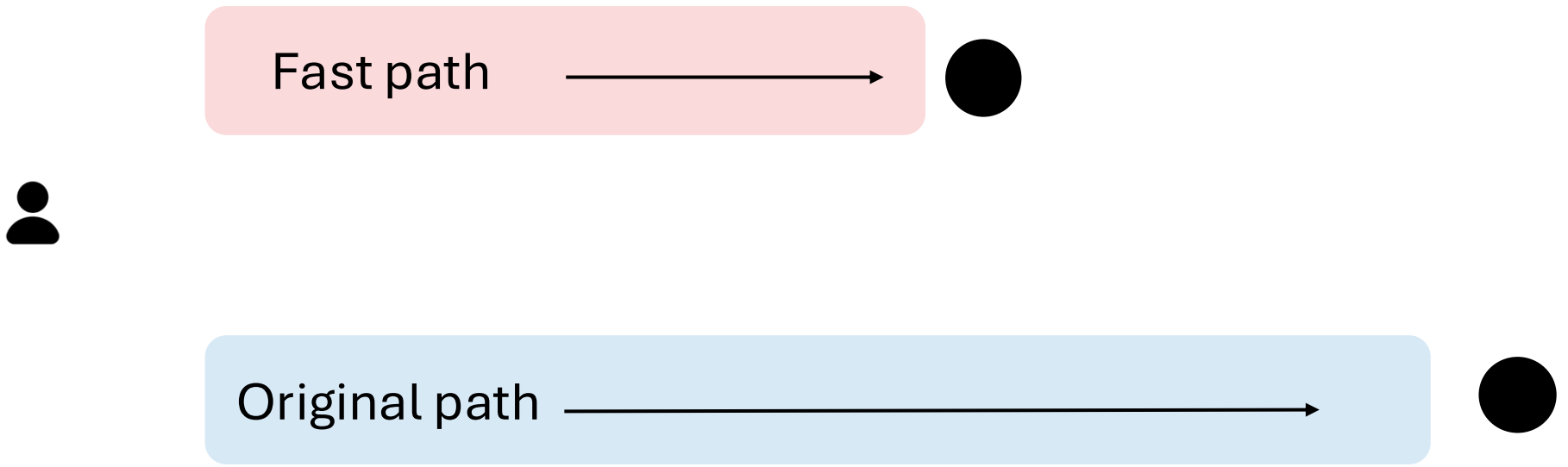
Intuition



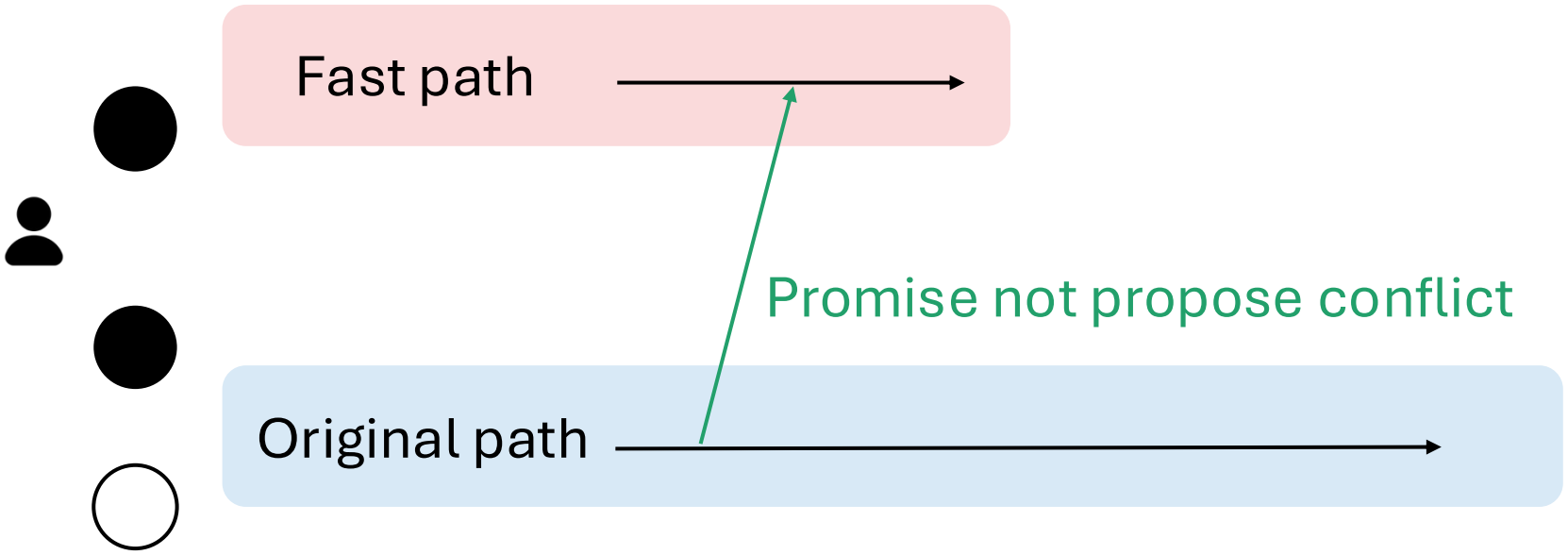
1 RTT
without conflicts

2 RTTs
Handles conflicts
original features

Intuition


Fast path
Original path
1 RTT
without conflicts
2 RTTs
Handles conflicts
original features

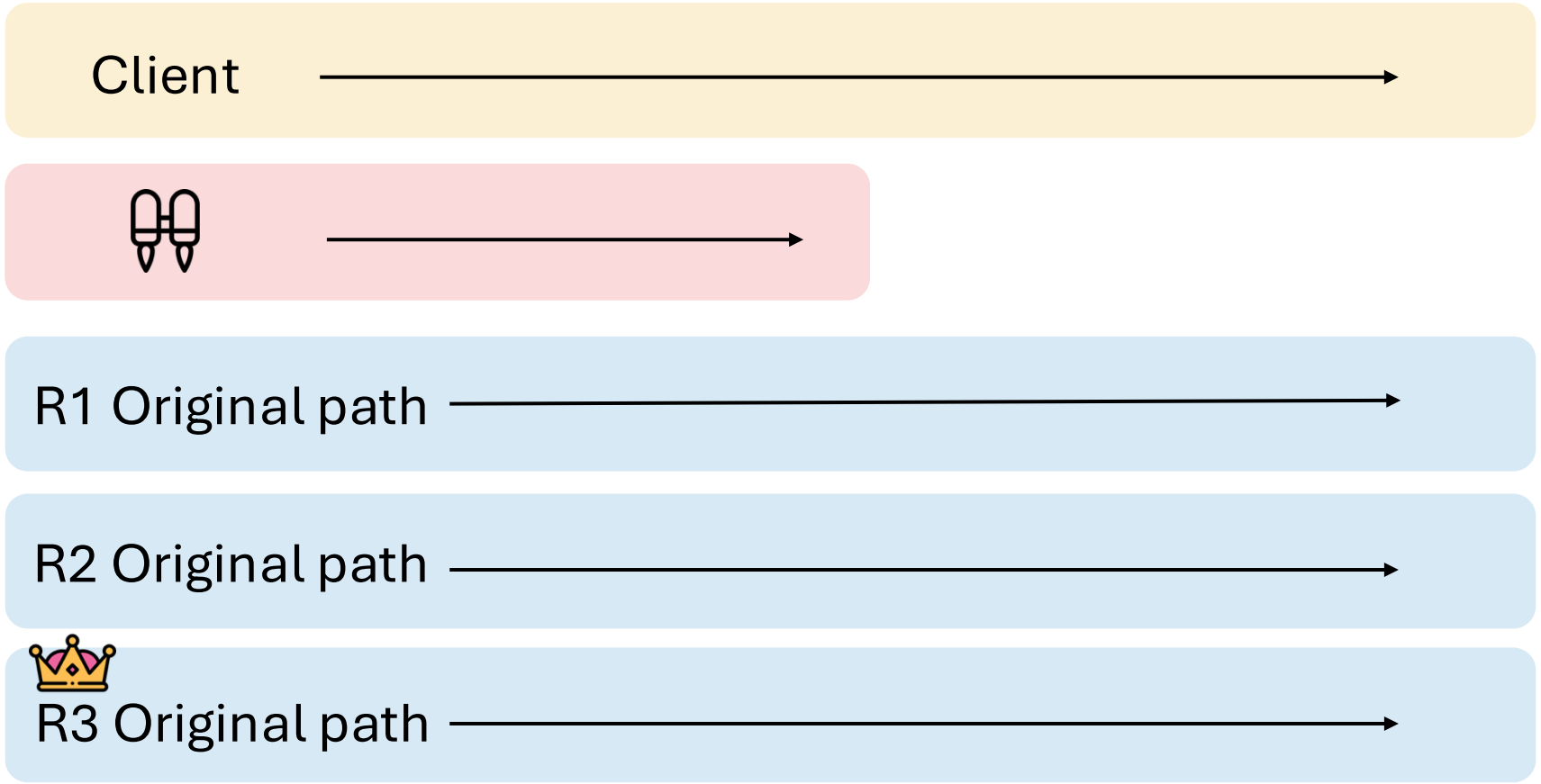
Intuition



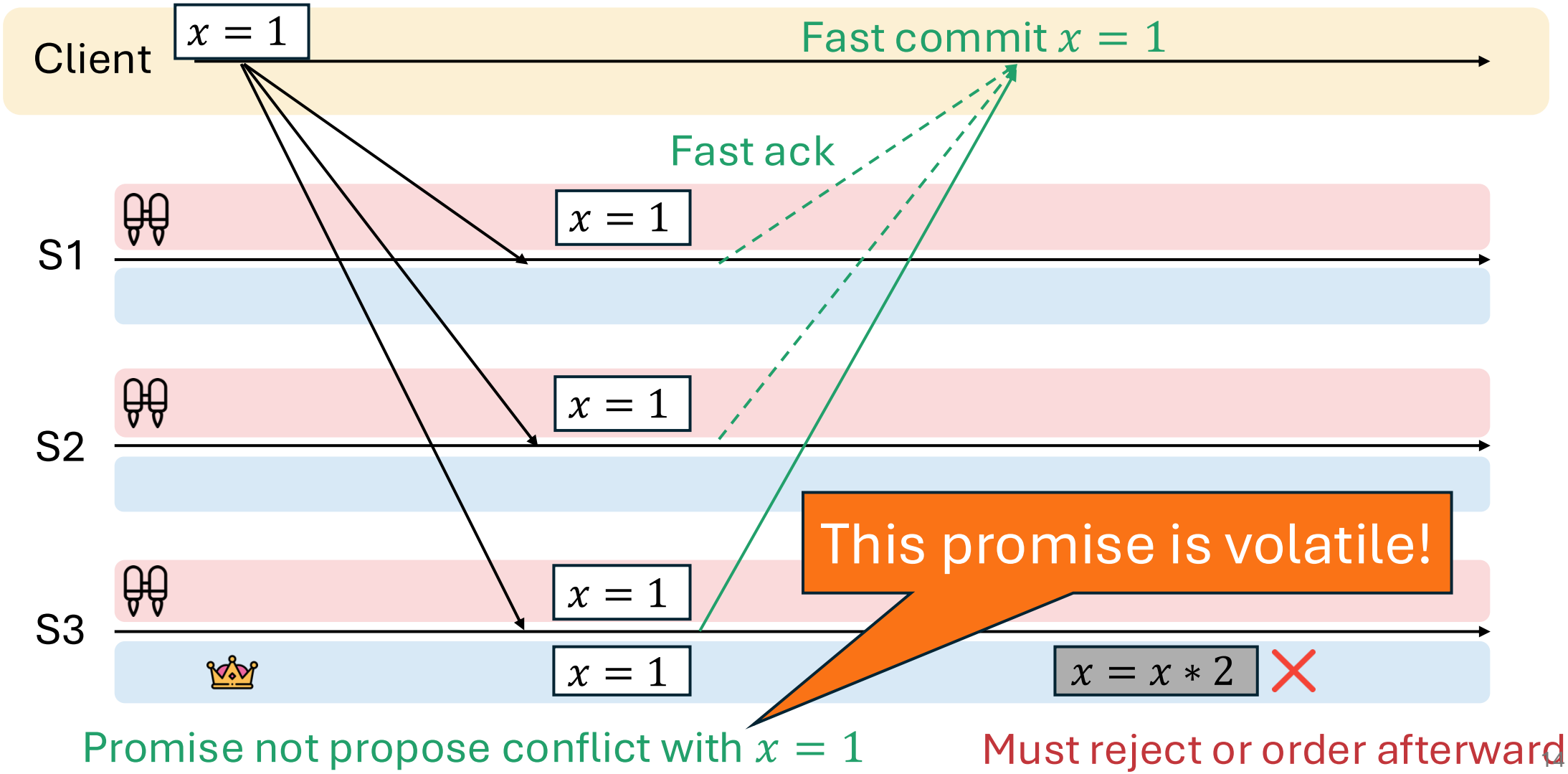
1 RTT
without conflicts

2 RTTs
Handles conflicts
original features

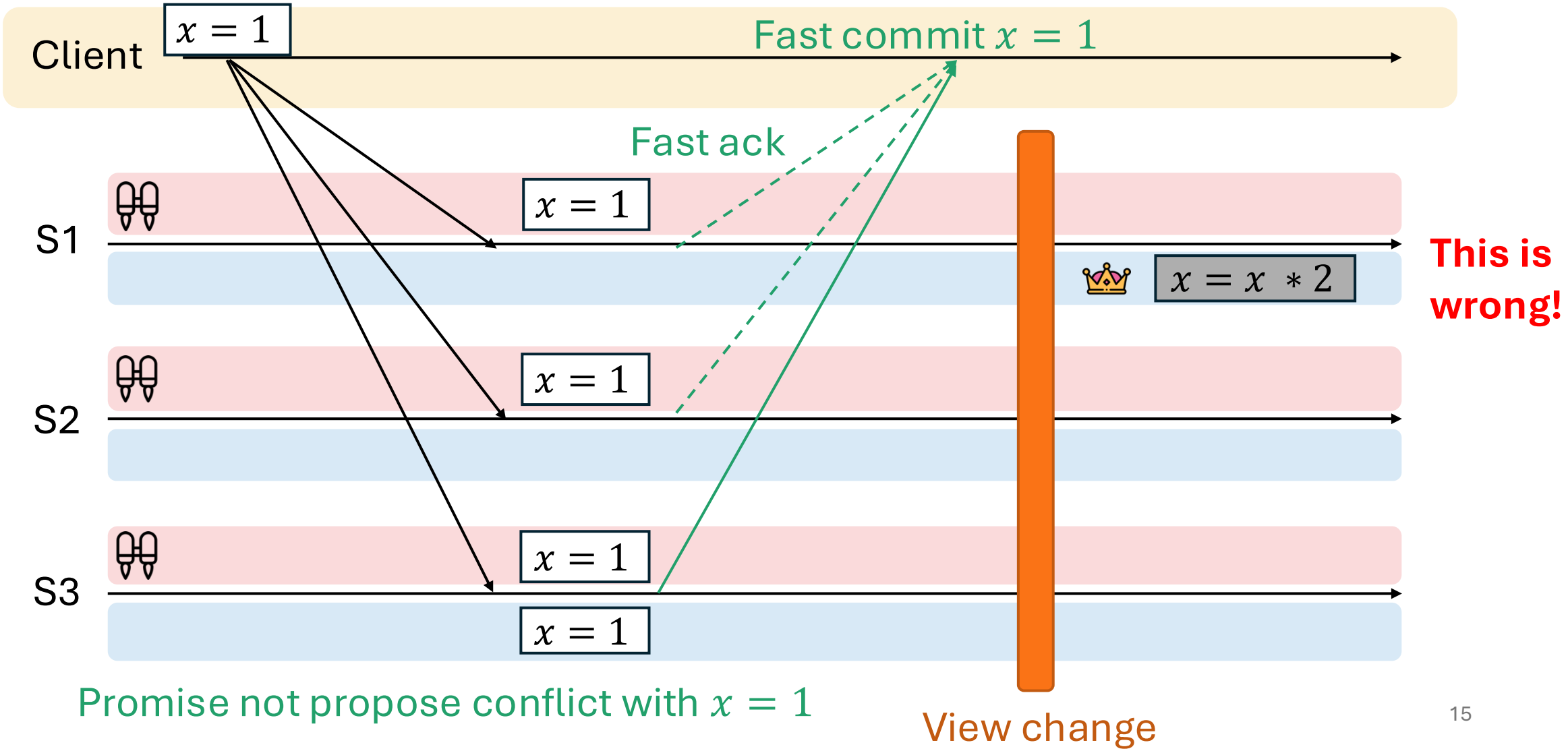
Idea: Original path make proposal promise



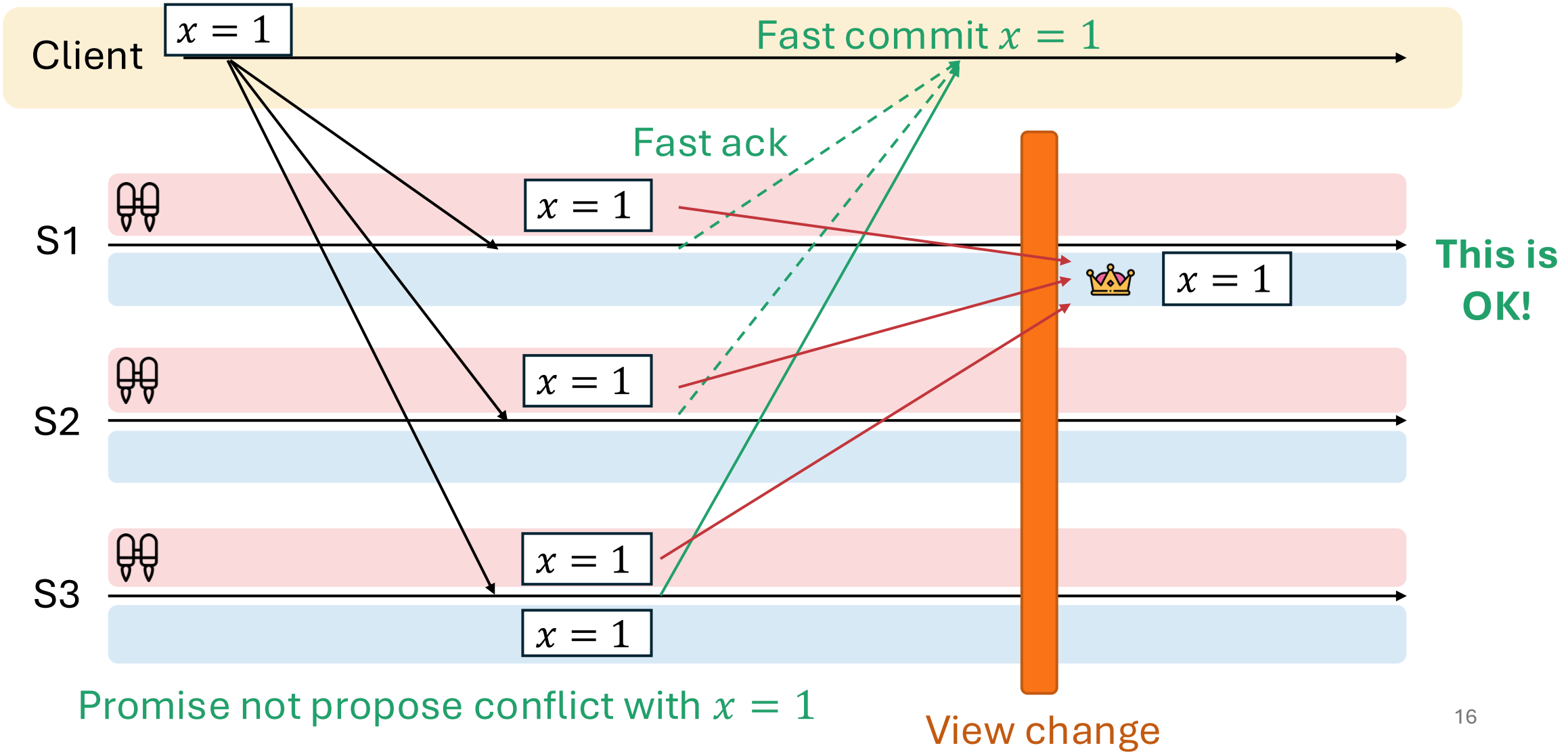
Idea: Original path make proposal promise



The view change hazard



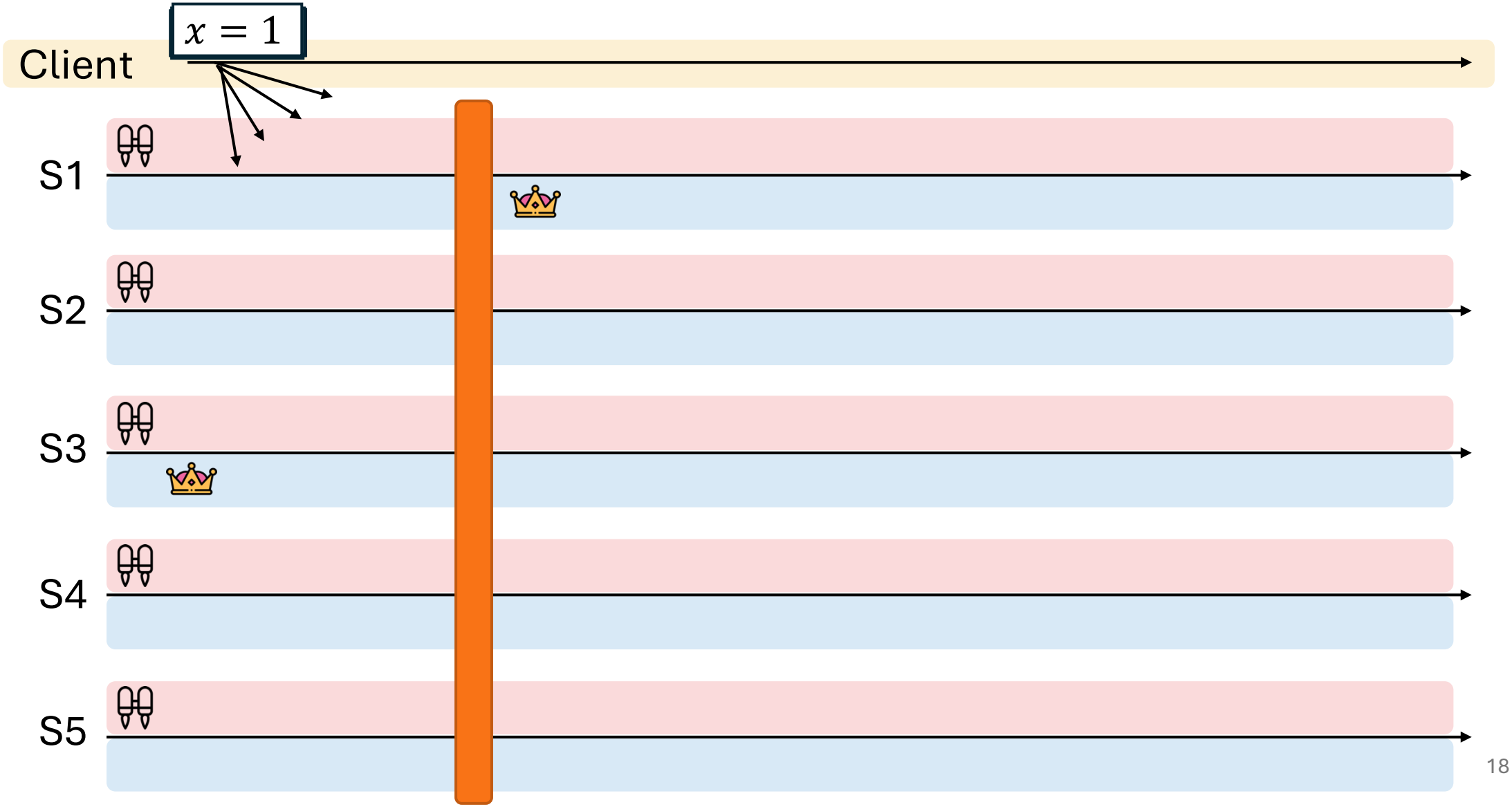
General solution



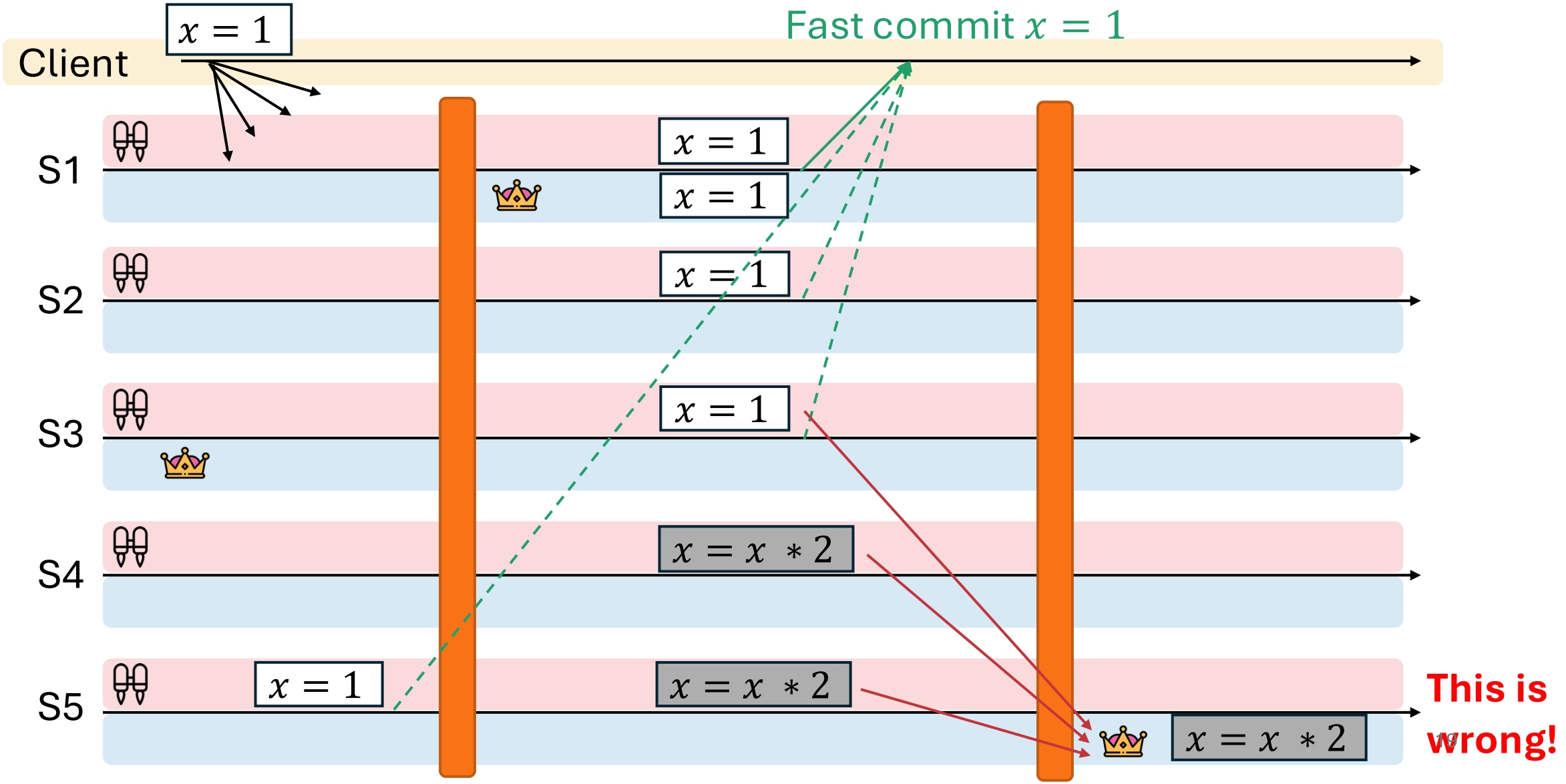
Two concreate corner cases

- Case 1: Silent Loss
- Case 2: Broken Order

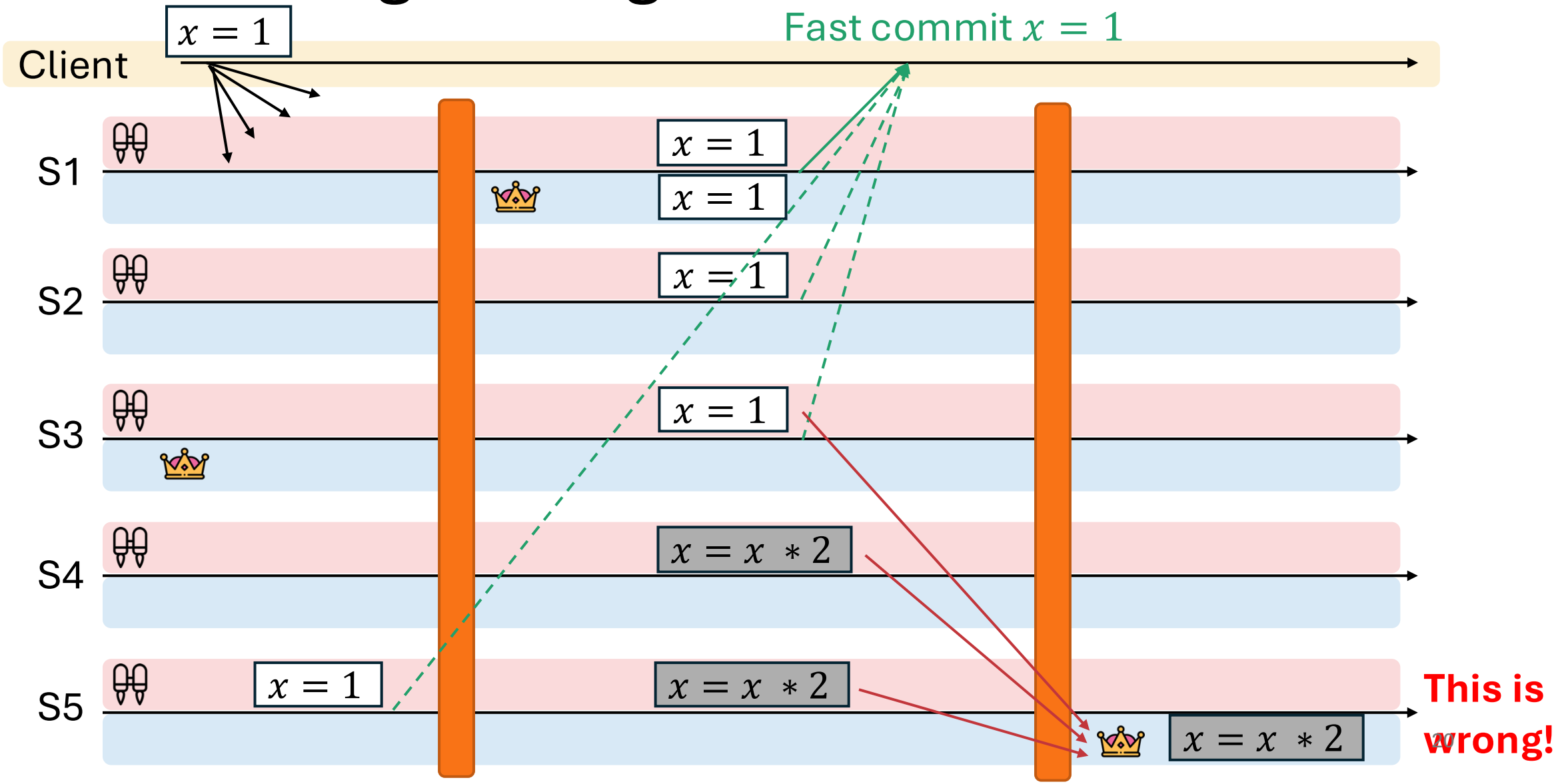
Case 1 Silent Loss



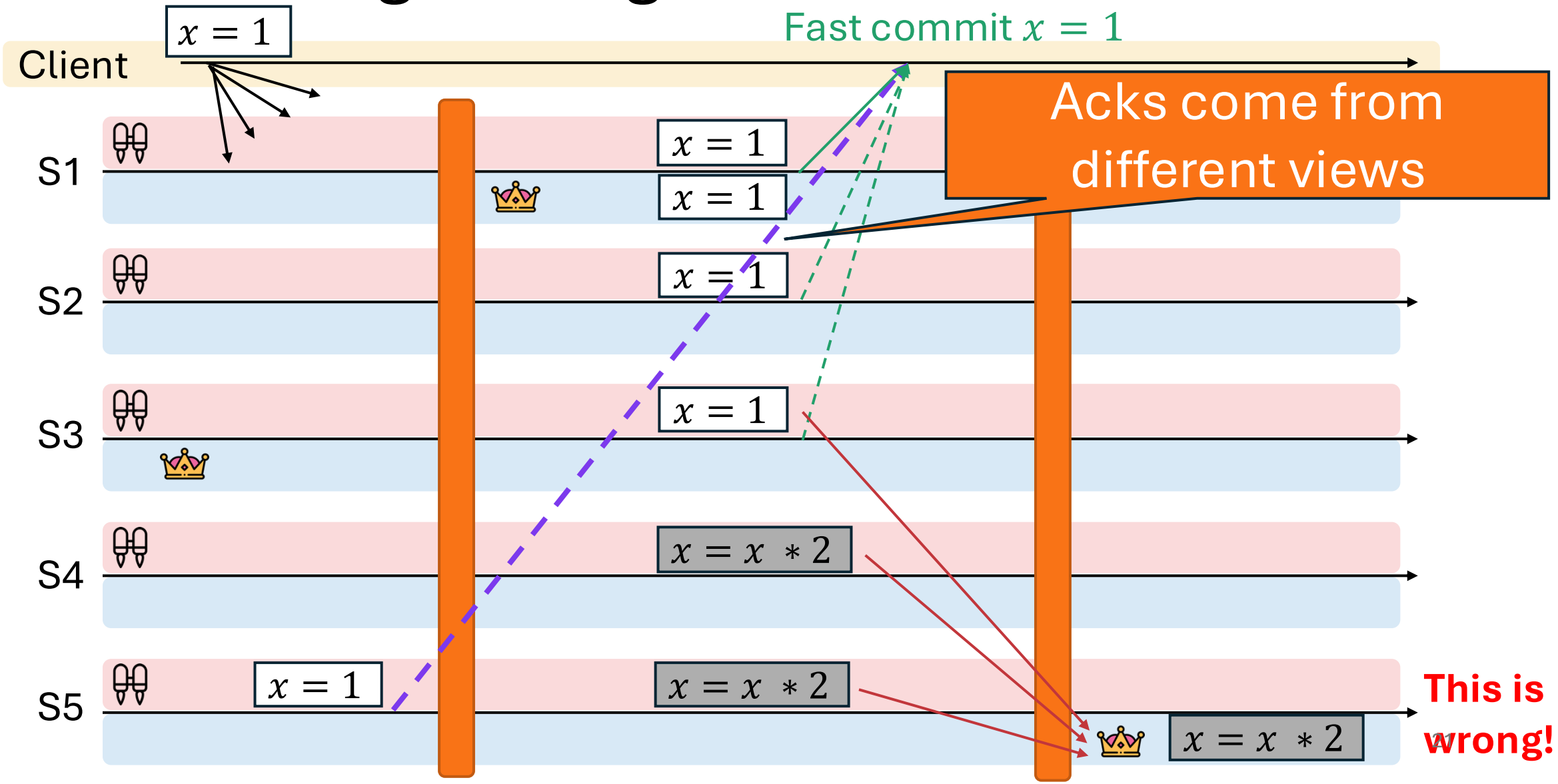
Case 1 Silent Loss



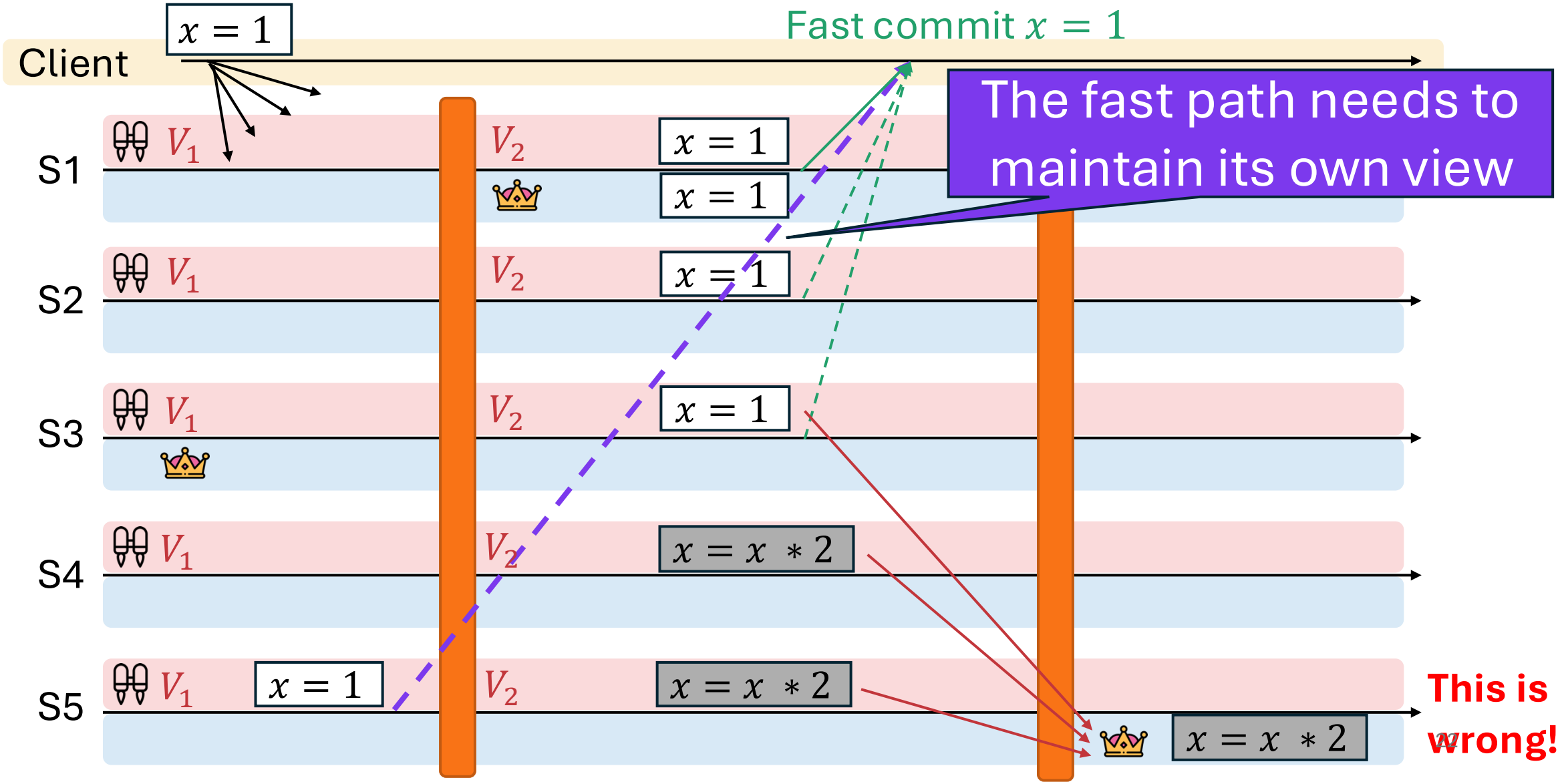
When did it go wrong?



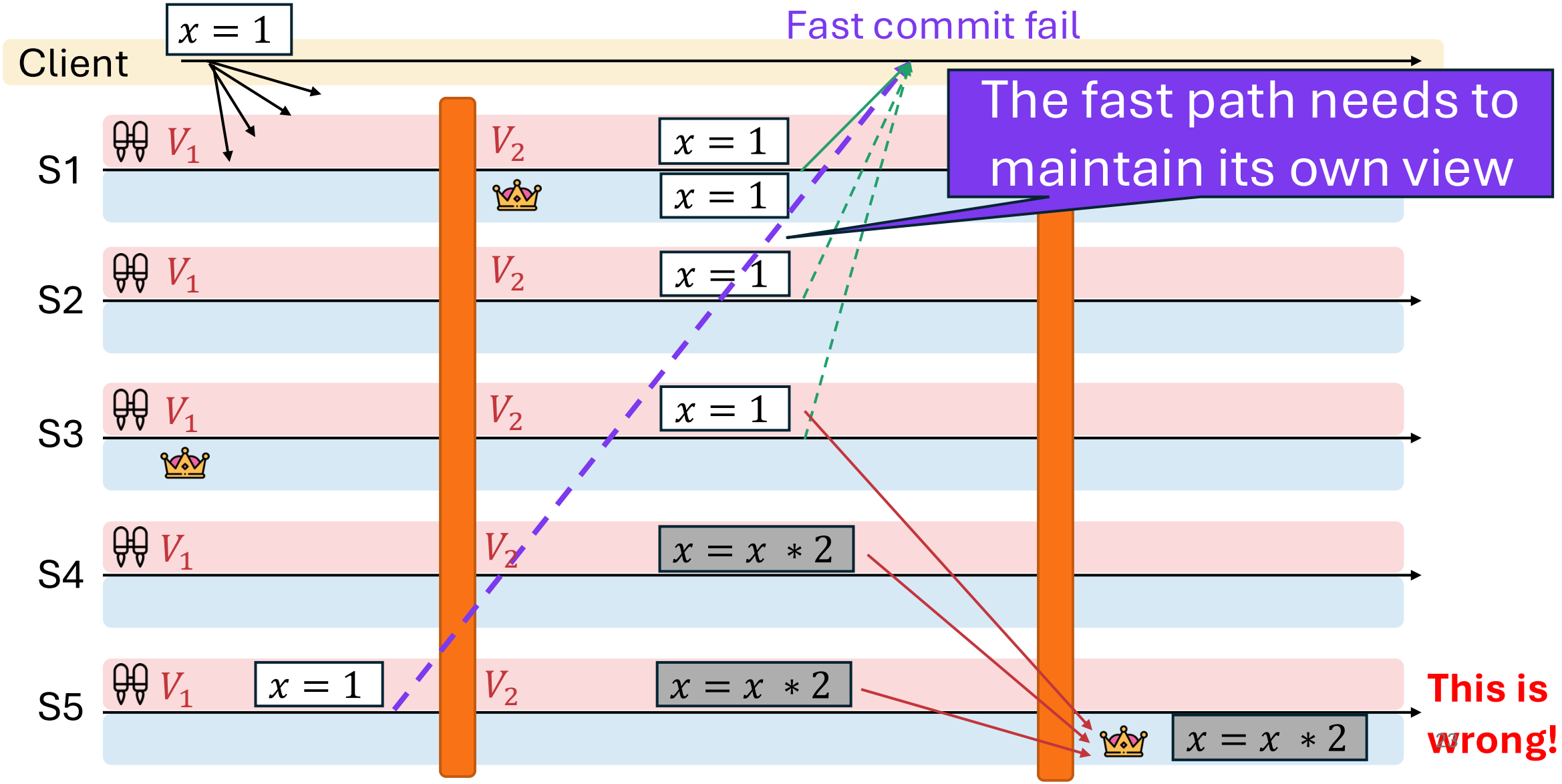
When did it go wrong?



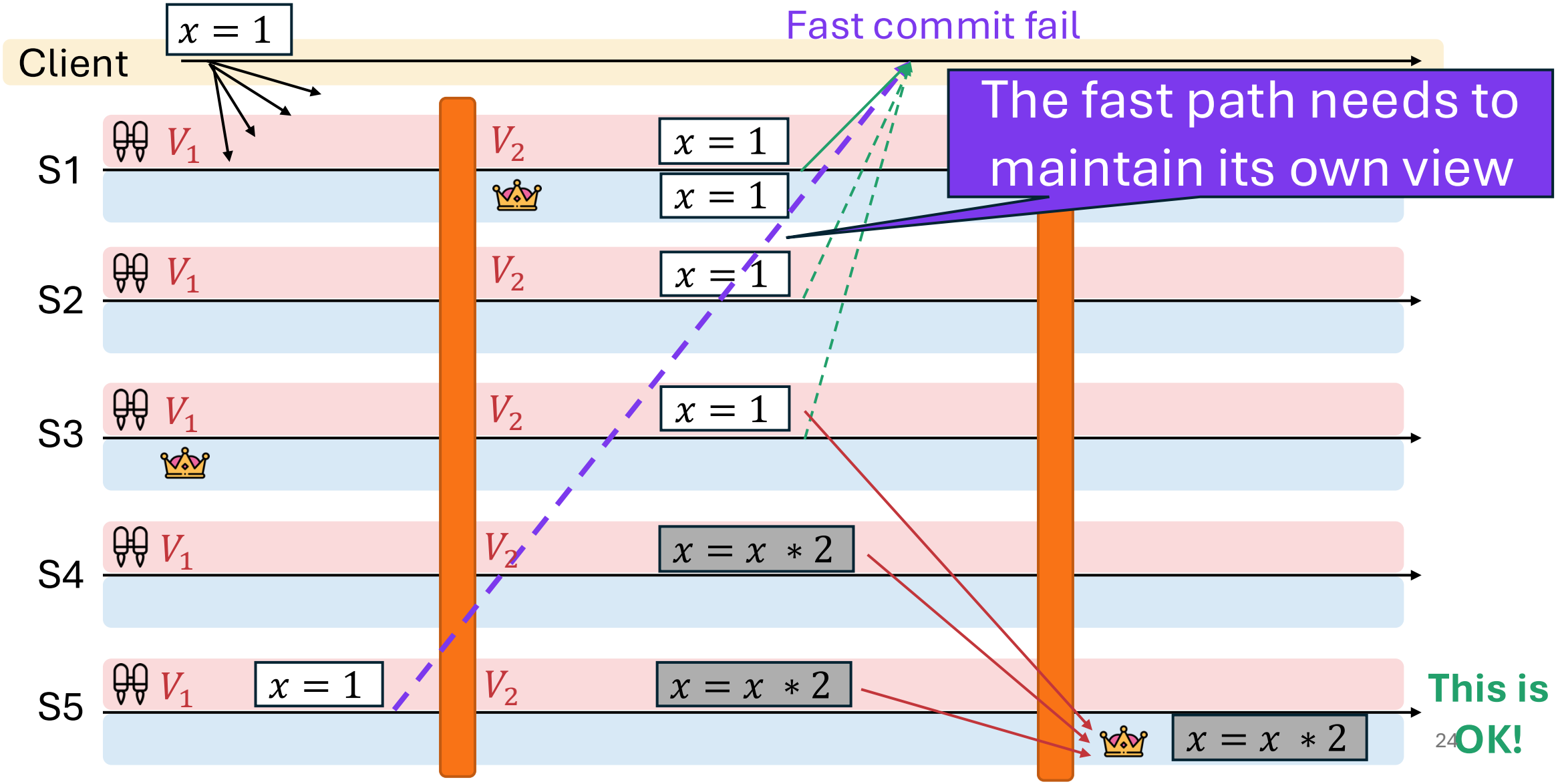
How can we solve this?



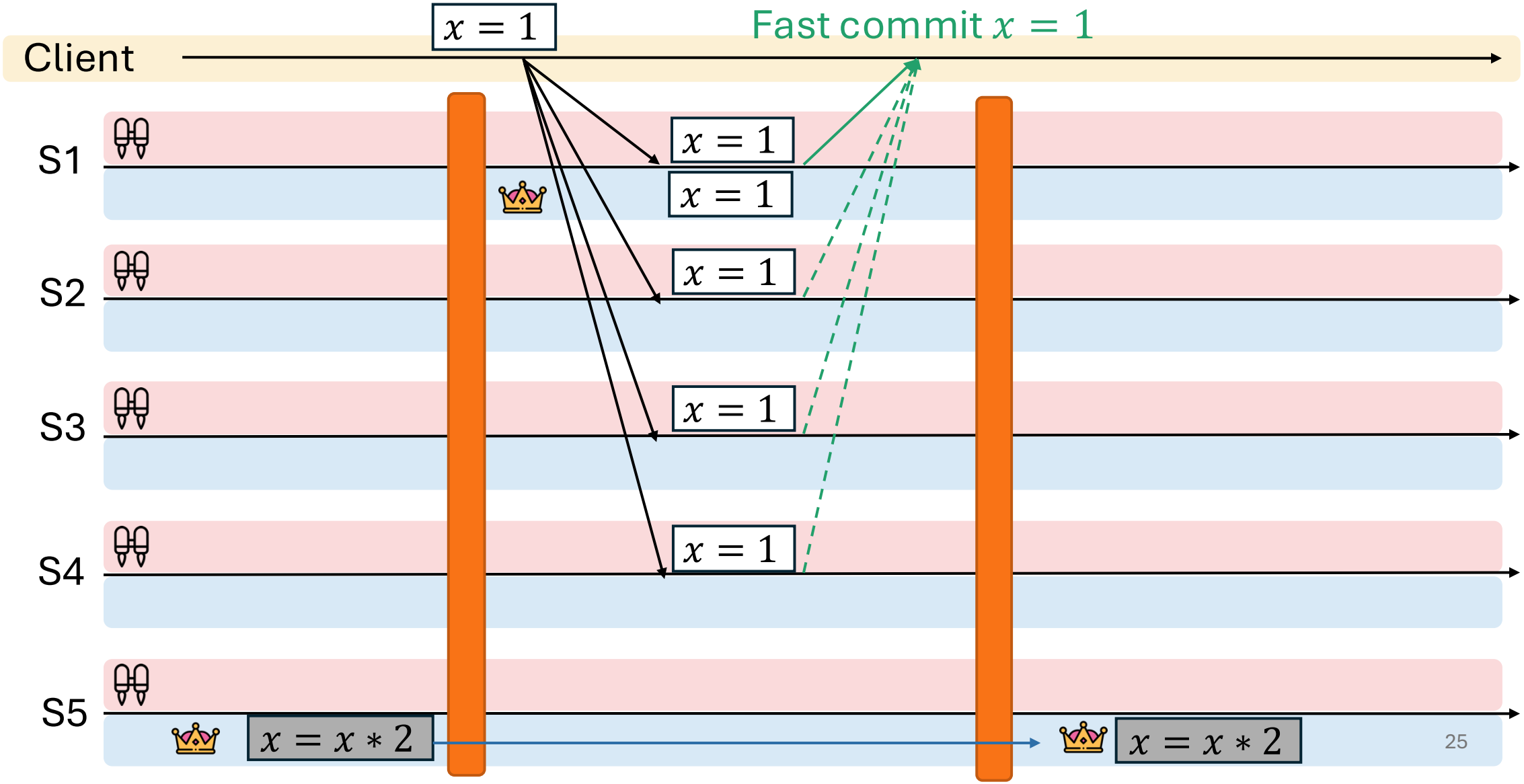
How can we solve this?



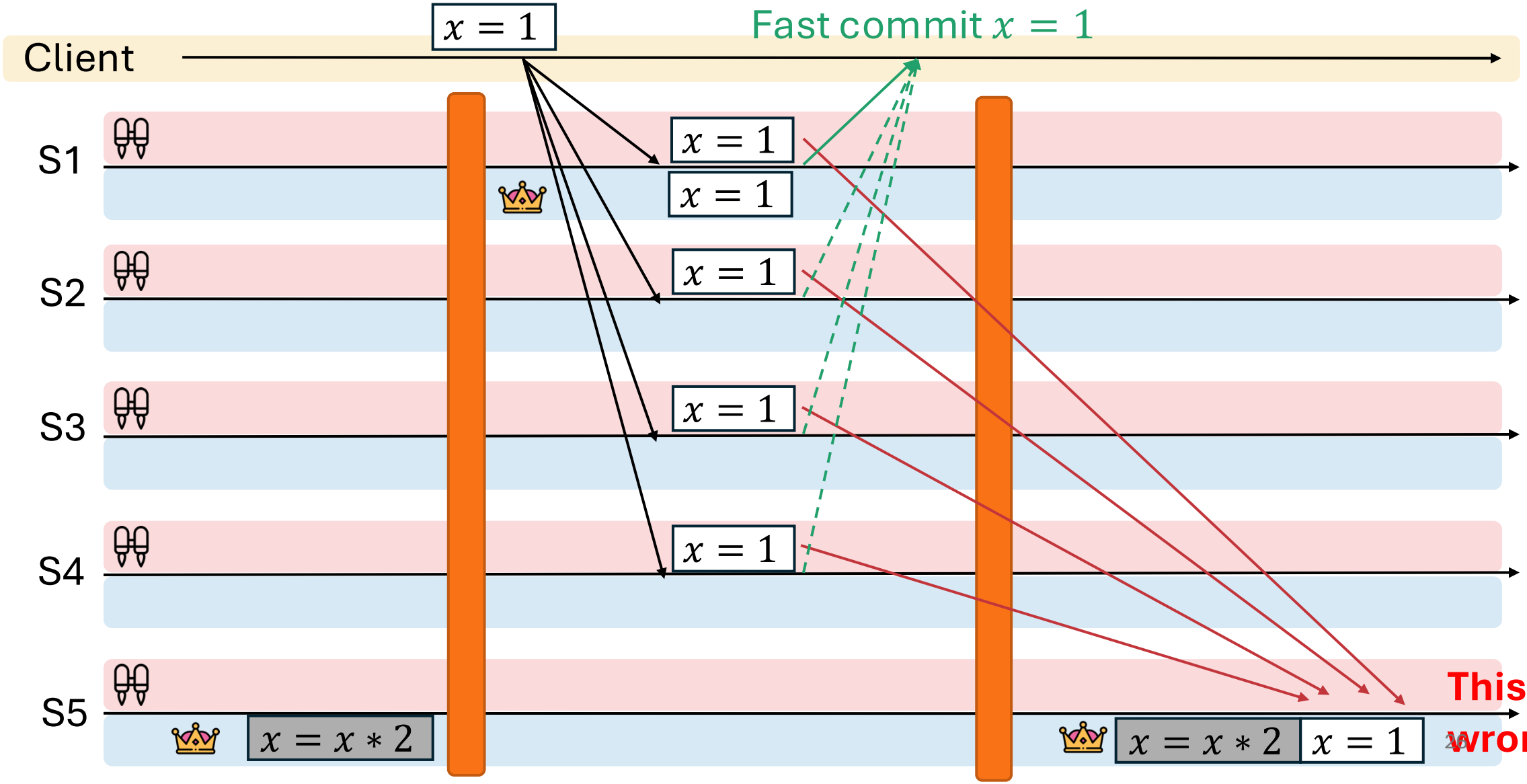
How can we solve this?



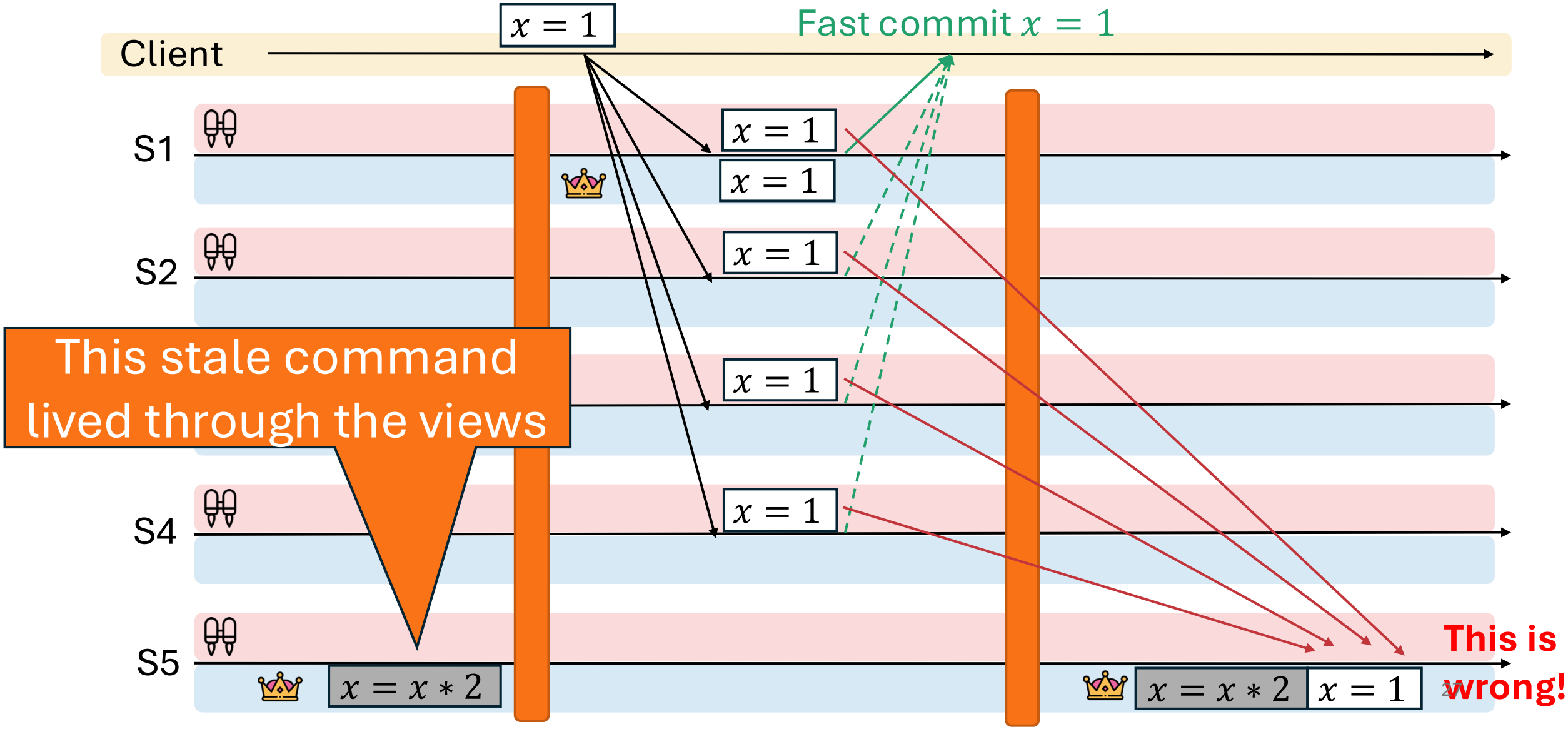
Case 2 Broken Order



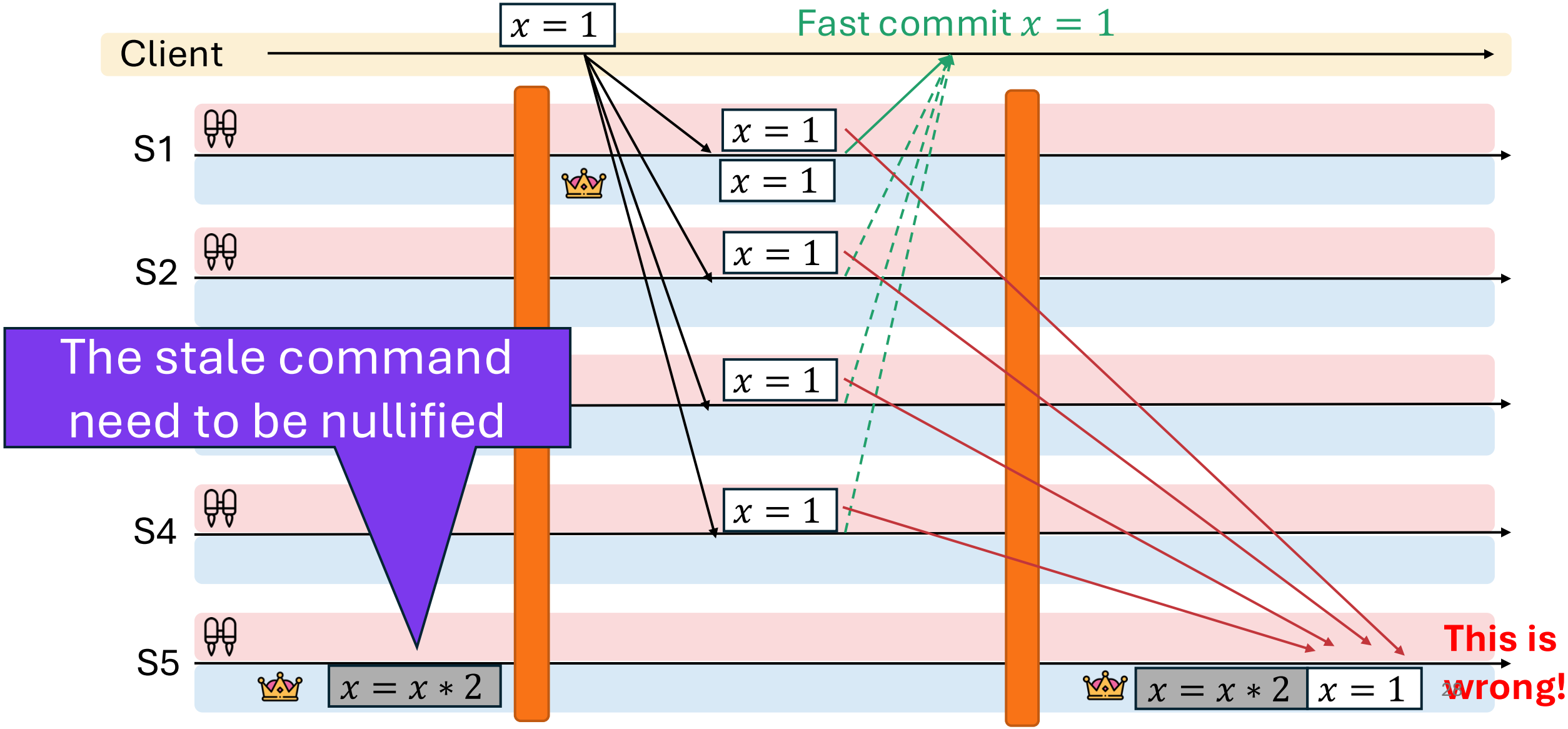
Case 2 Broken order



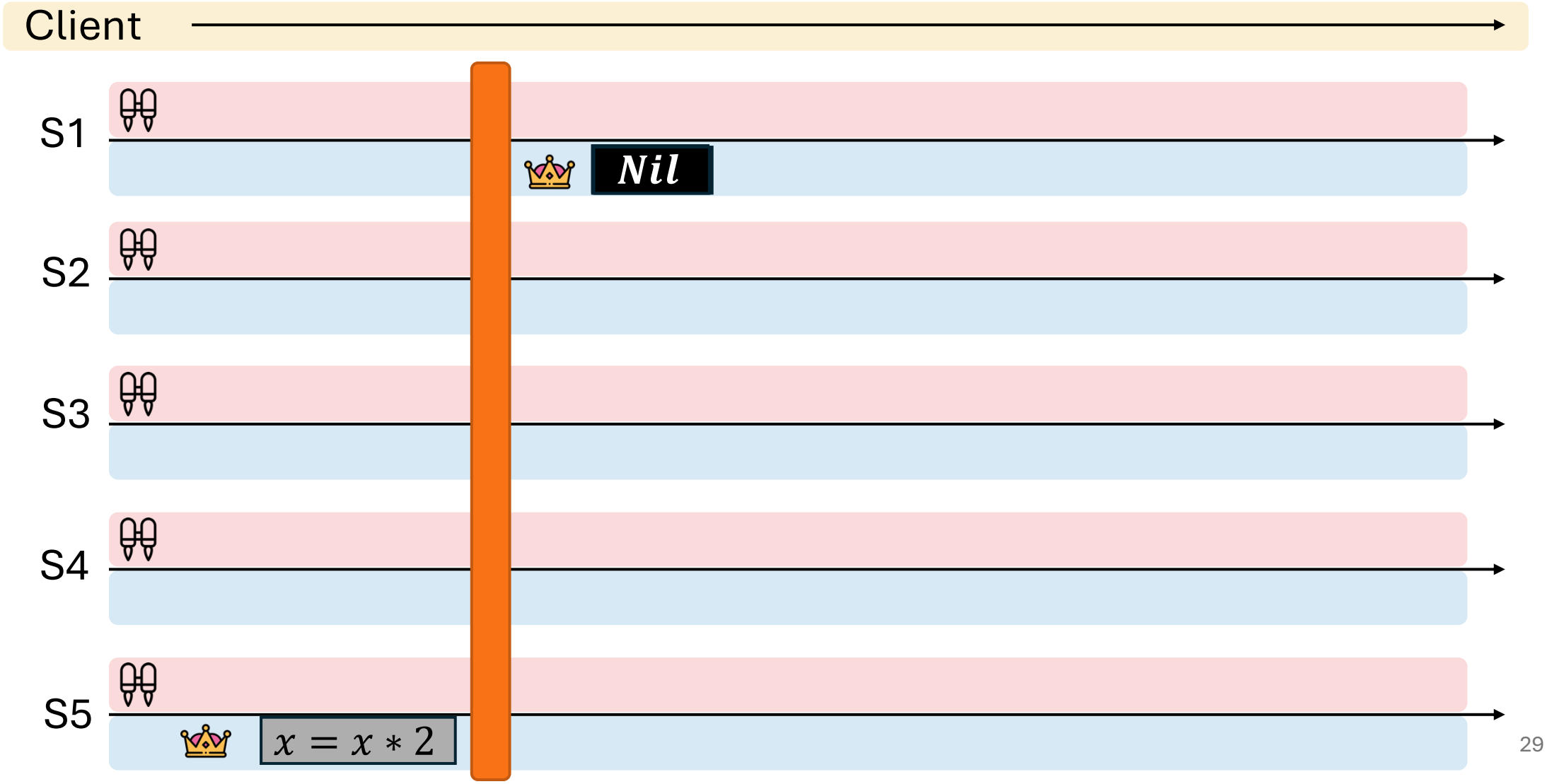
When did it go wrong?



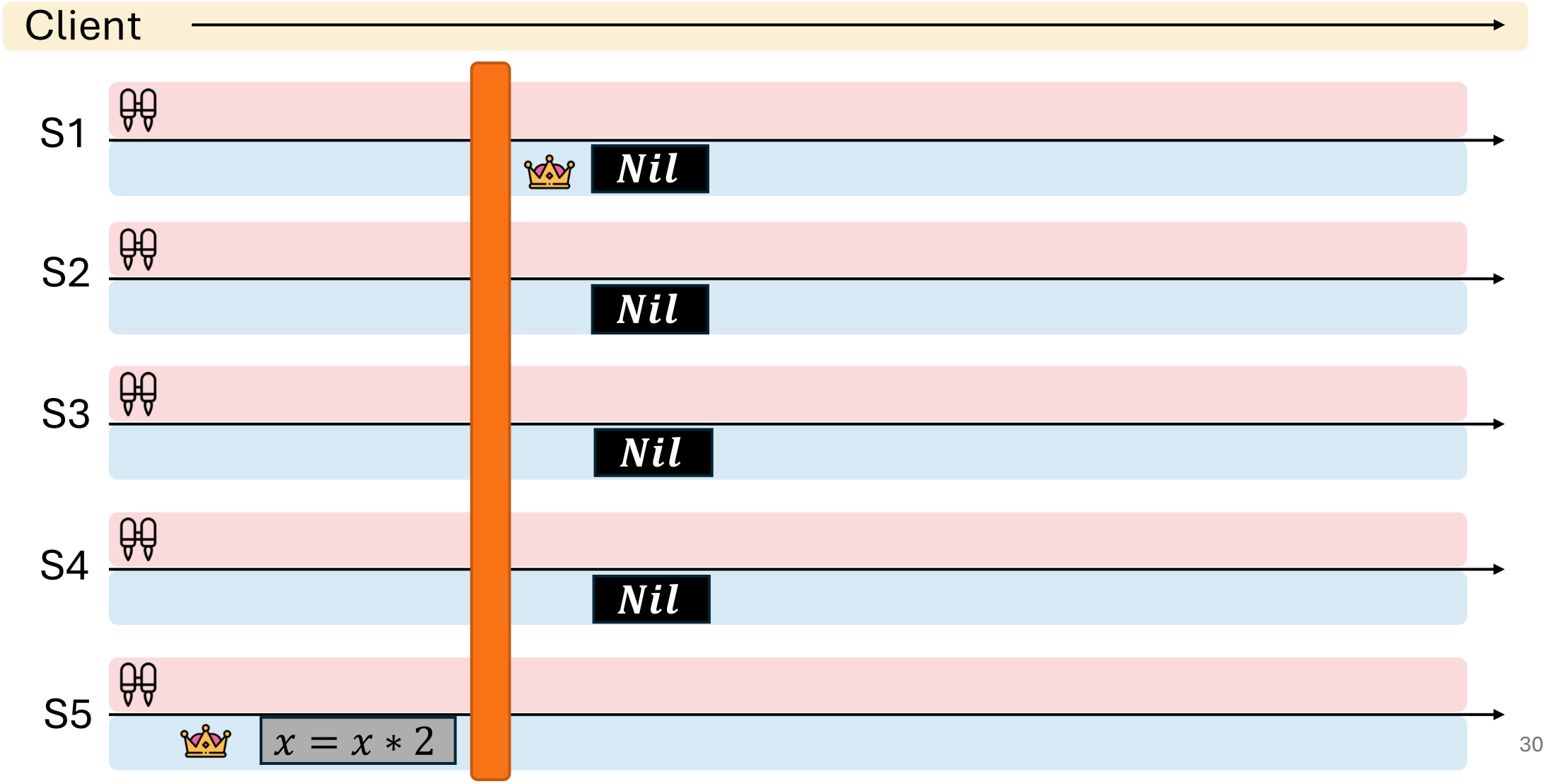
How can we solve this?



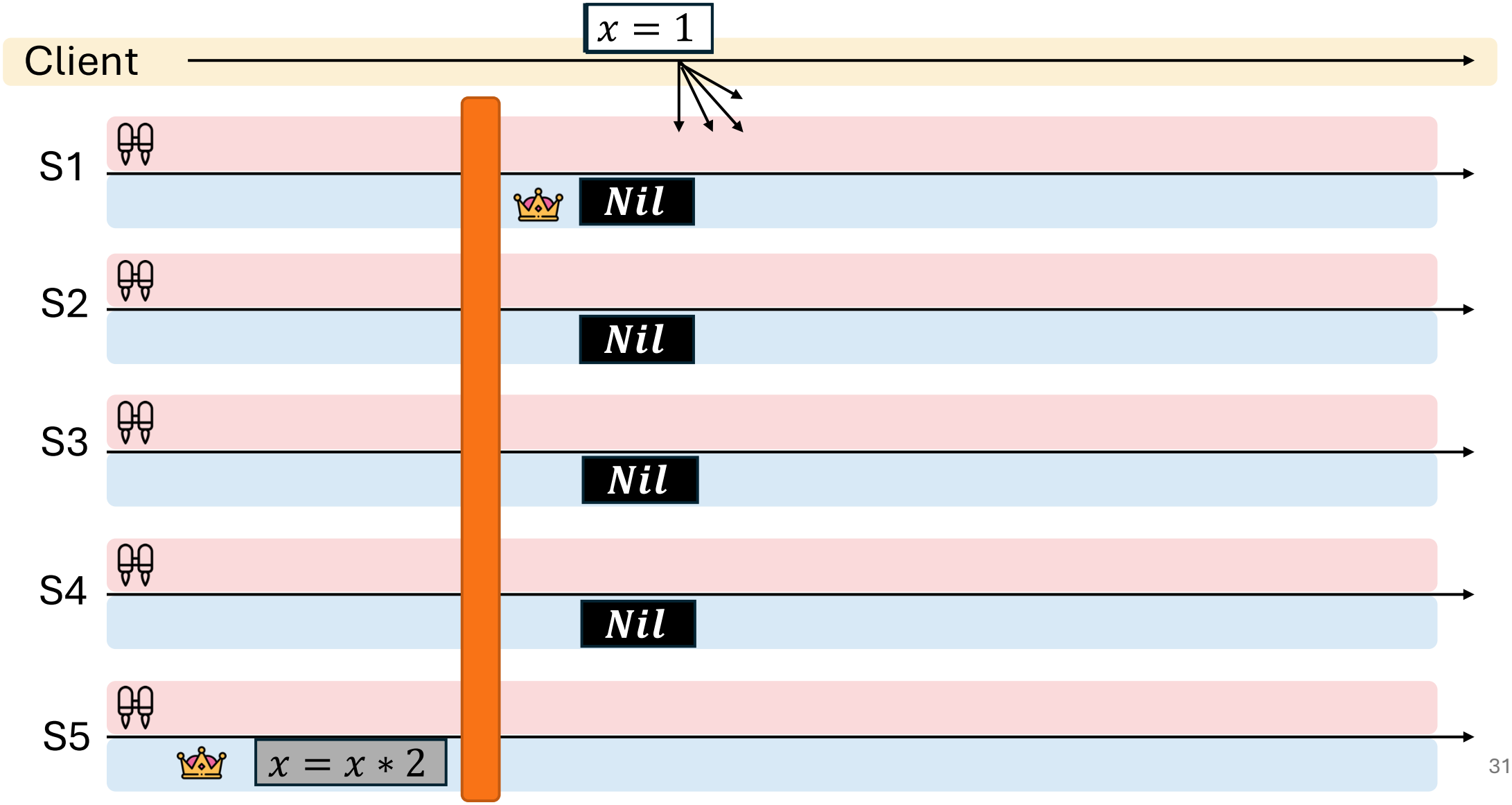
How can we solve this?



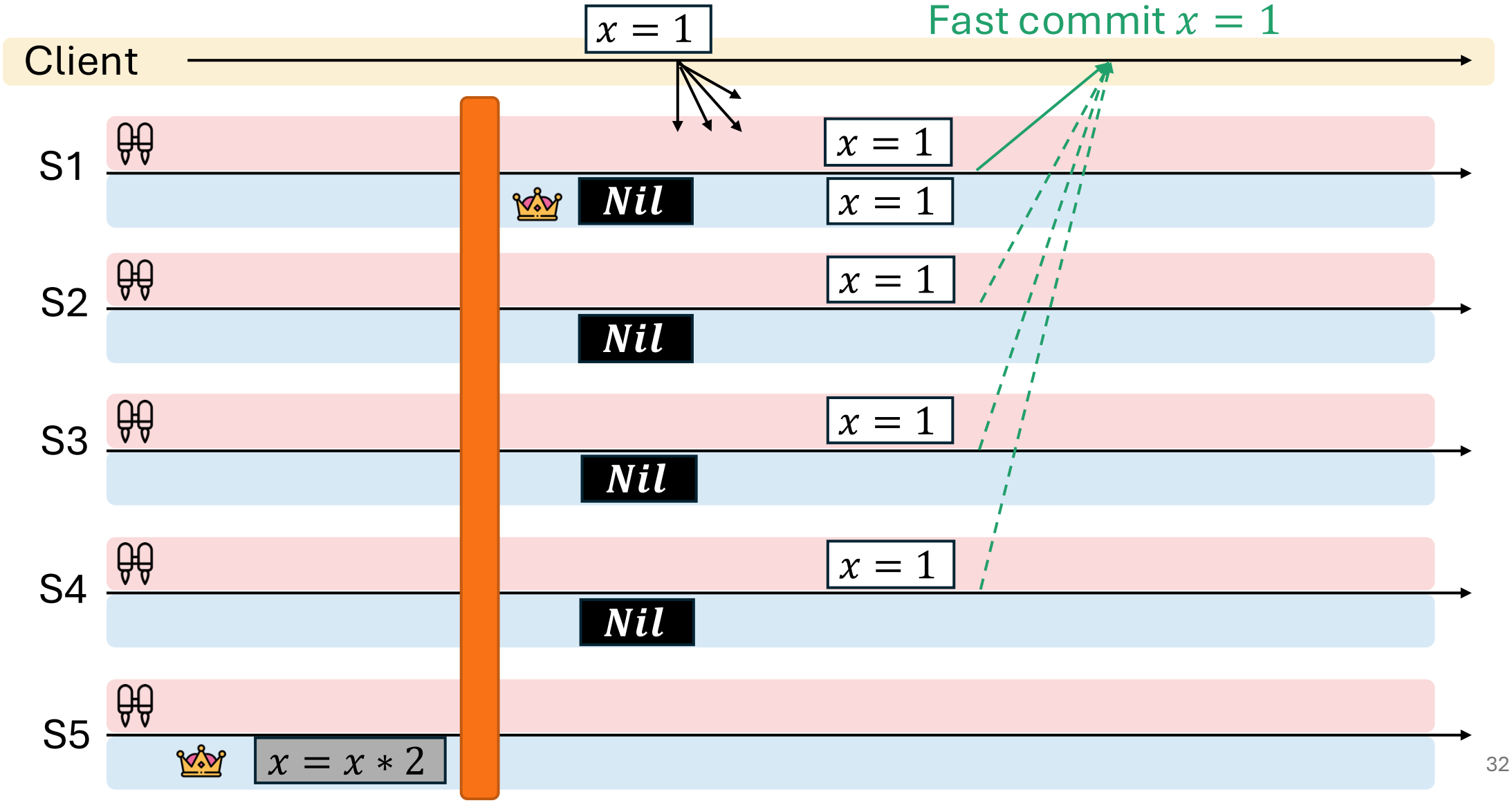
How can we solve this?



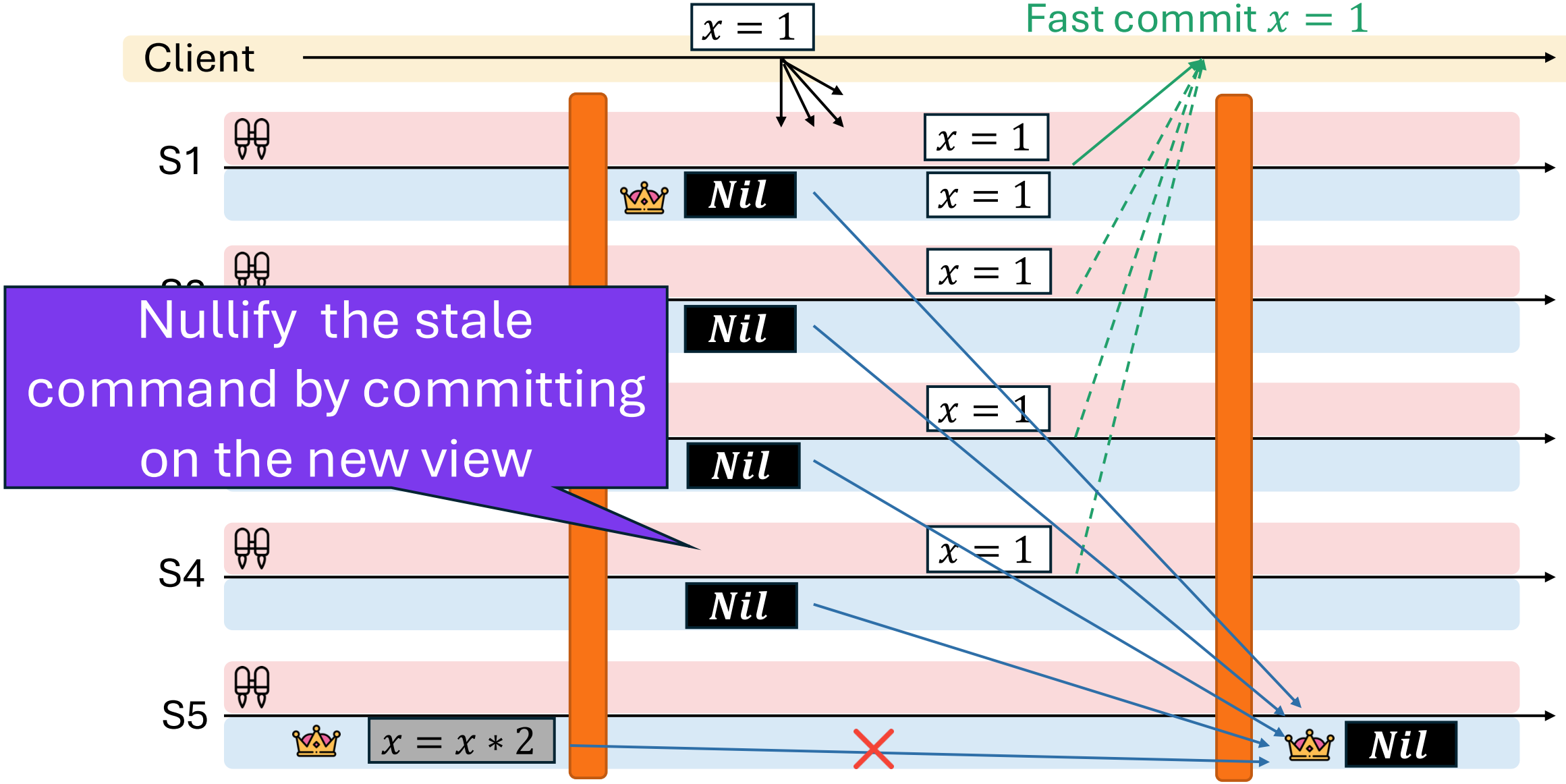
How can we solve this?



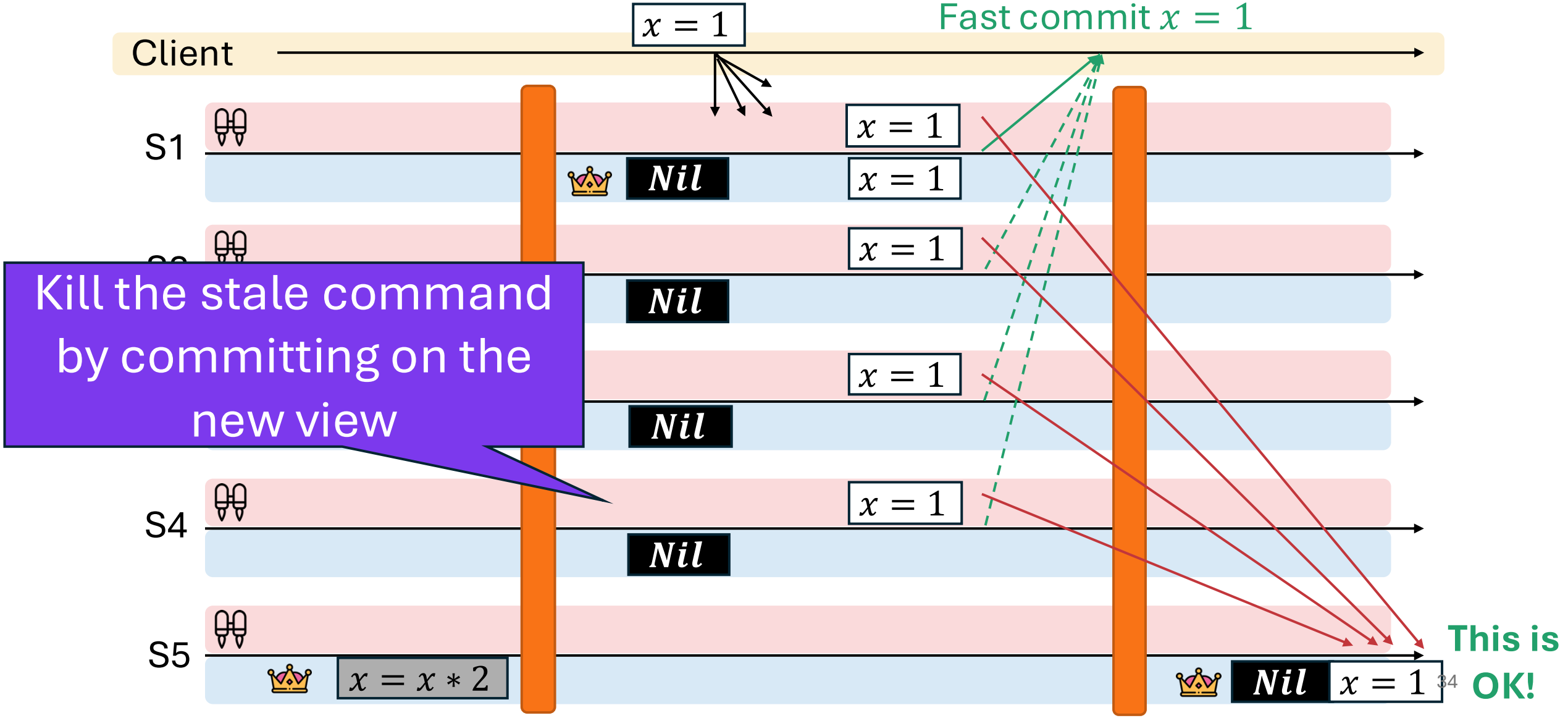
How can we solve this?



How can we solve this?



How can we solve this?



Two concreate corner cases

- Case 1: Silent loss
  - Problem: Acks come from a different view
  - Solution: The fast path needs to maintain its own view
- Case 2: Broken order
  - Problem: Stale commands live through views
  - Solution: Nullify stale commands on view change

Evaluation setup

- Protocol testbed[1]
  - Raft: single leader
  - Copilot: dual leader
  - Mencius: multi leader
- Real production systems
  - MongoDB: Raft virant
  - Etcd: Raft
  - Zookeeper: ZAB

[1]Build on DepFast

Evaluation setup

- Protocol testbed[1]
  - Raft: single leader
  - Copilot: duel leader
  - Mencius: multi leader
- Real production systems
  - MongoDB: Raft virant
  - Etcd: Raft
  - Zookeeper: ZAB

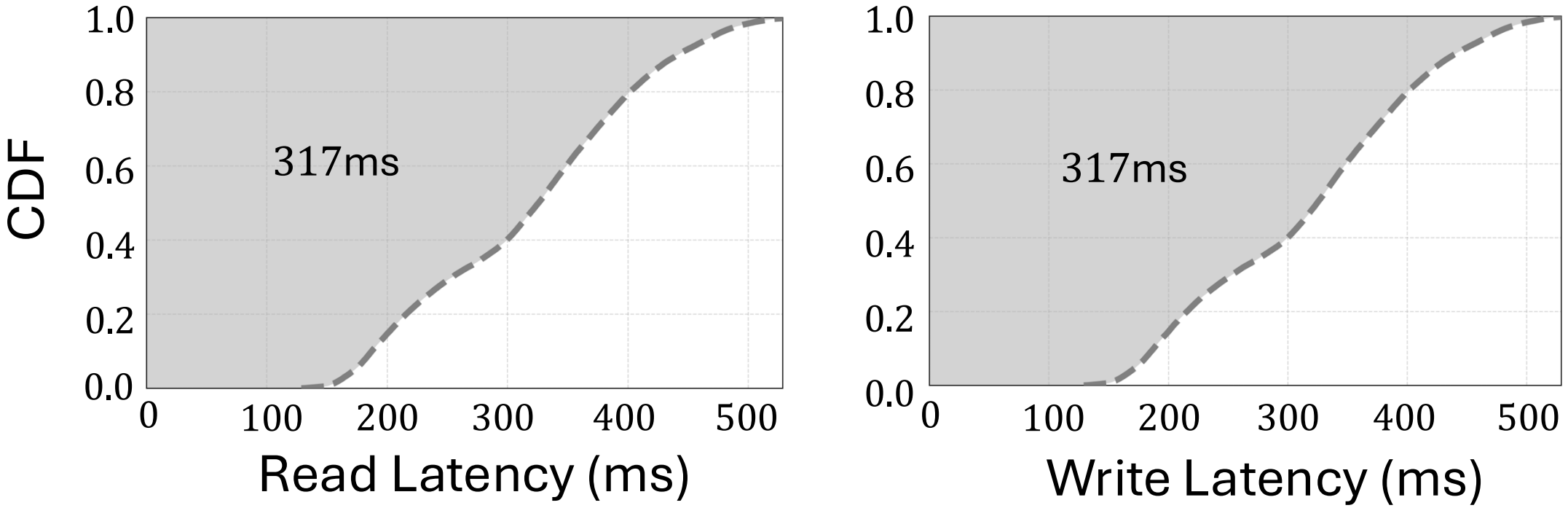
[1]Build on DepFast

Evaluation setup

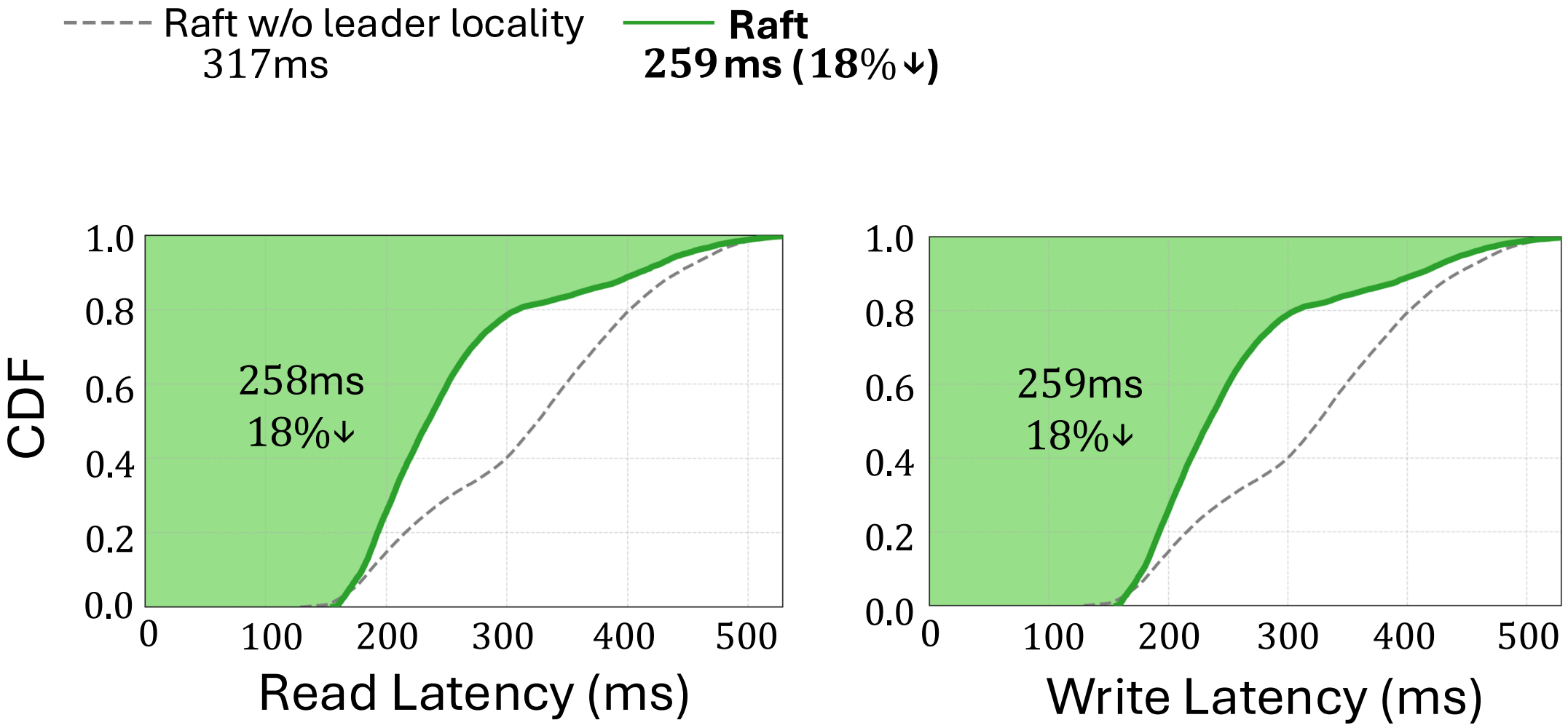
- Testbed & workload
  - AWS
  - 10 datacenters
  - Simulated Facebook's Akkio traces
    - 73/8/9/7/3%
  - YCSB-like 50/50 read/write

Performance of Raft + Jetpack

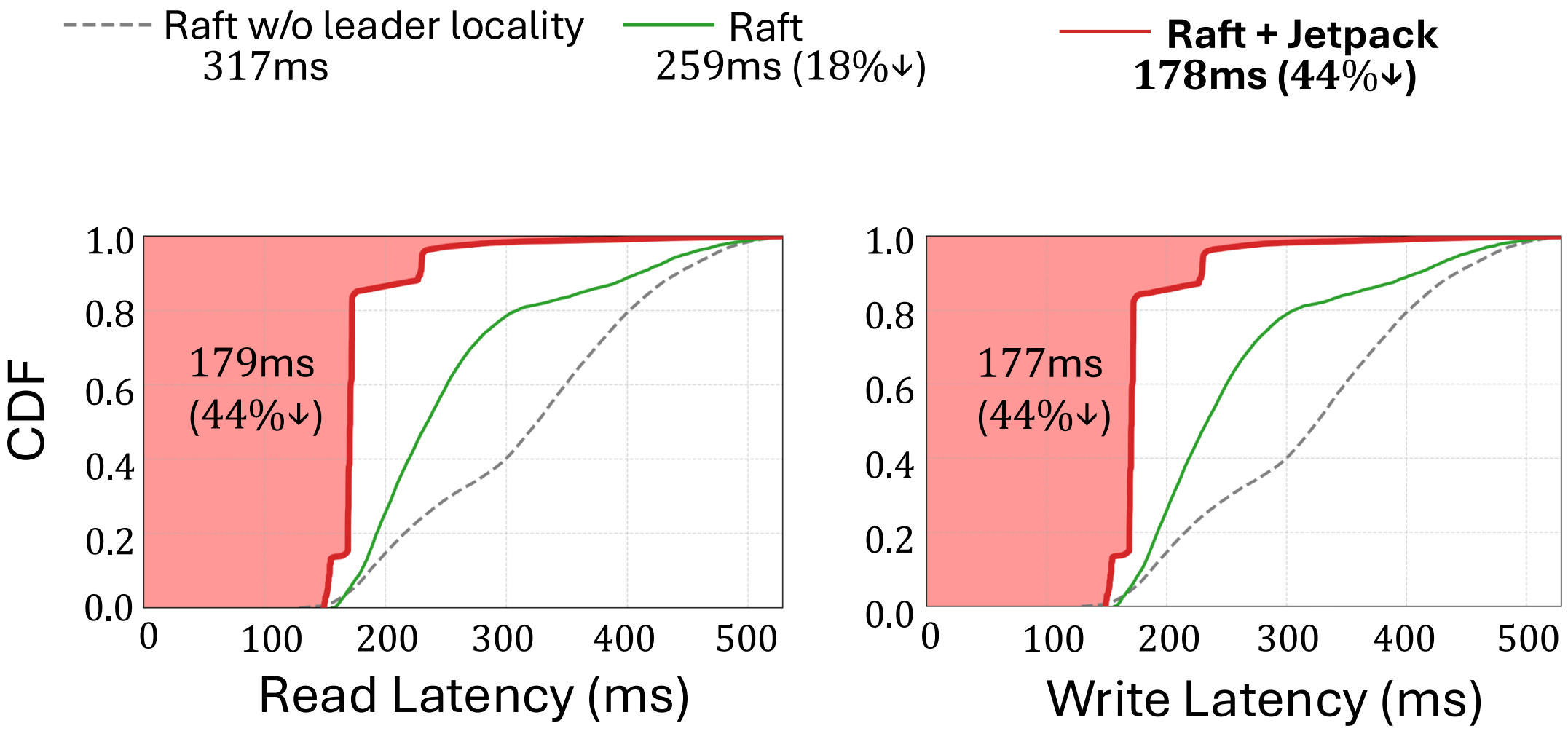
----- **Raft w/o leader locality**
317ms



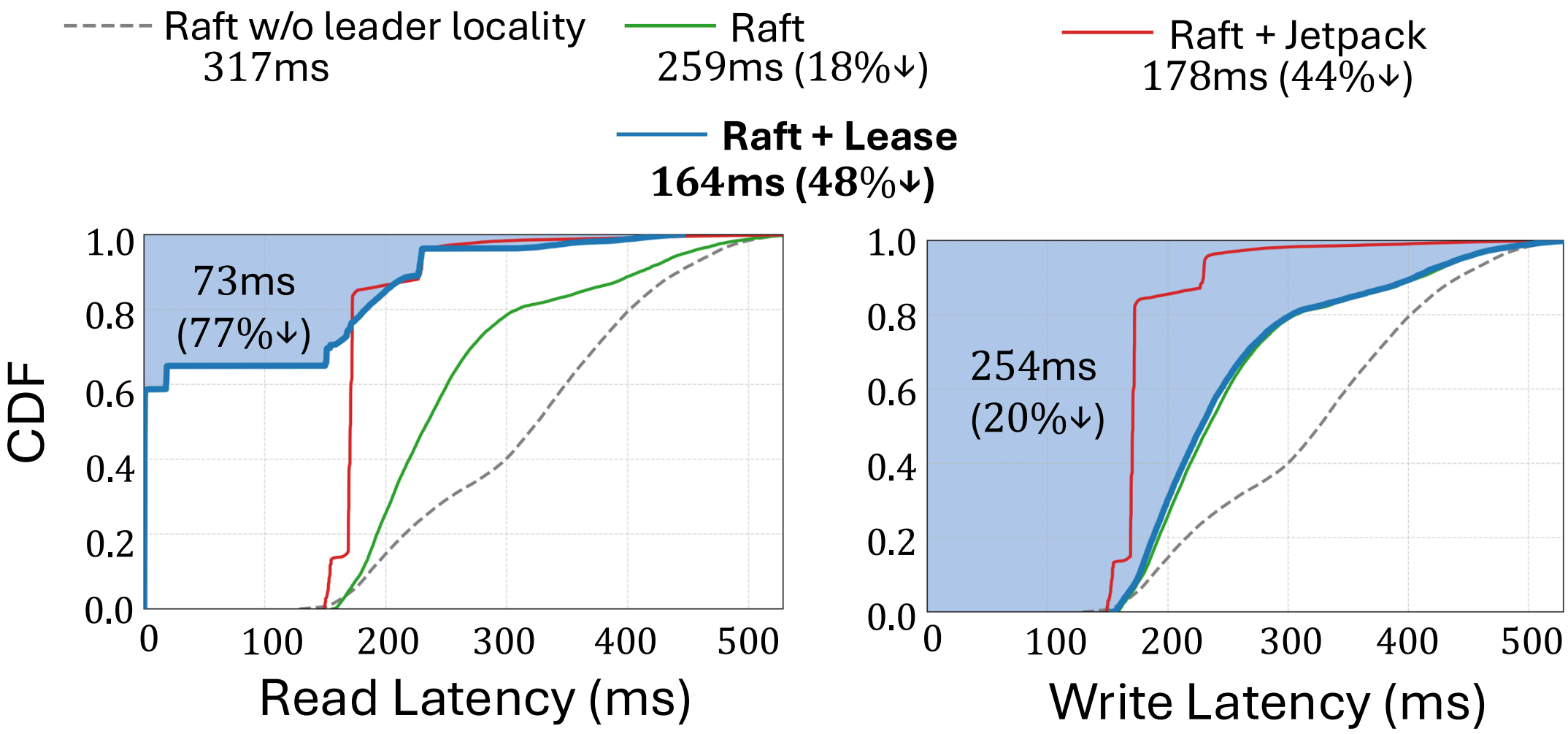
Performance of Raft + Jetpack



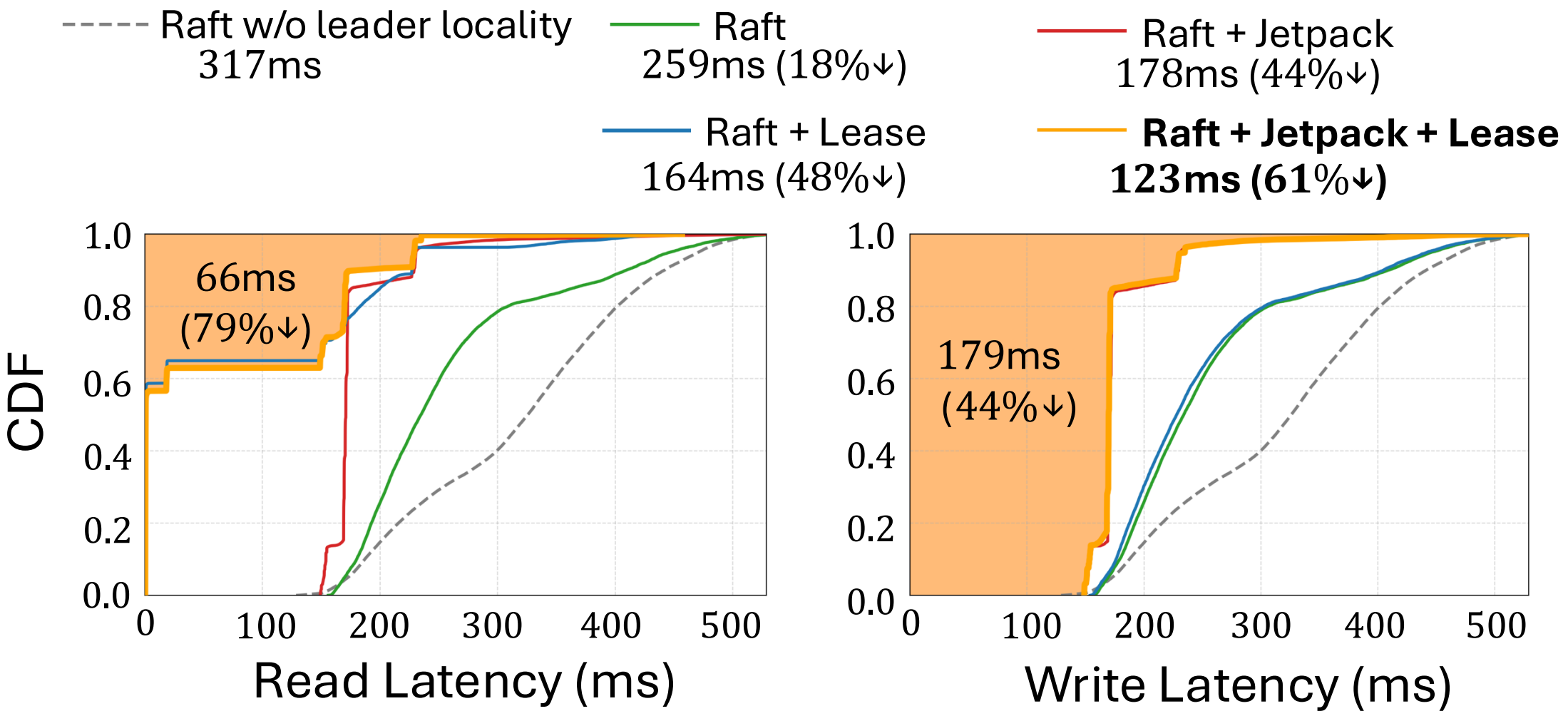
Performance of Raft + Jetpack



Performance of Raft + Jetpack



Performance of Raft + Jetpack



Integration with MongoDB

MongoDB
modified Raft

Fault-Tolerant Replication with Pull-Based Consensus in MongoDB

Siyuan Zhou[1], Shuai Mu[2]
[1]*MongoDB Inc.* [2]*Stony Brook University*

primary-catchup
(new leader pull uncommitted)
+
fast path

Case 2

4.3 Preserving Uncommitted Oplog Entries

After a failover, the uncommitted oplog entries on the previous primary are likely lost. This would be fine for a different system because the clients could retry uncommitted updates. This is, however, an issue for MongoDB because MongoDB supports fast but weak consistency levels that acknowledge writes as soon as they are applied on the primary or just replicated to a fewer number of nodes than a majority. Thus, a failover could cause a large loss of uncommitted writes. Though the clients are not promised durability with weak consistency levels, we still prefer to preserve their uncommitted writes as much as possible.

Integration with MongoDB

MongoDB

modified Raft

Fault-Tolerant Replication with Pull-Based Consensus in MongoDB

Siyuan Zhou[1], Shuai Mu[2]
[1]*MongoDB Inc.* [2]*Stony Brook University*

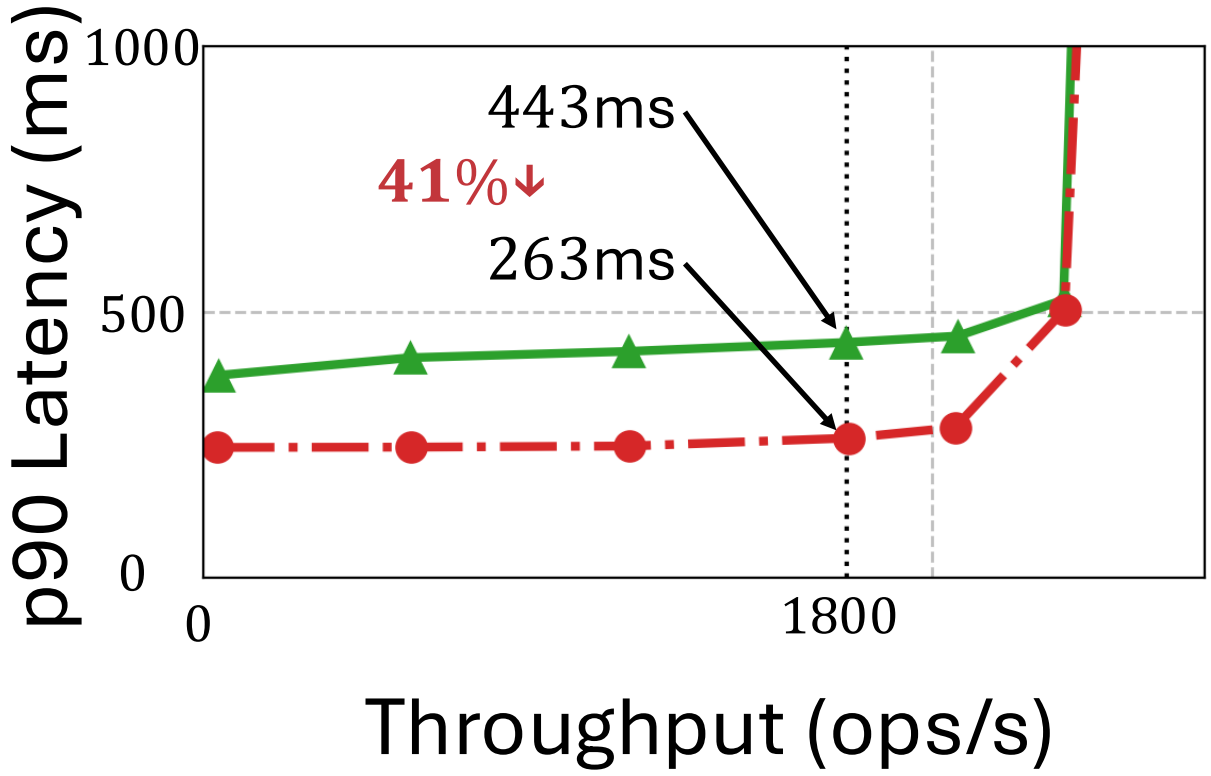
primary-catchup
(new leader pull uncommitted)

+

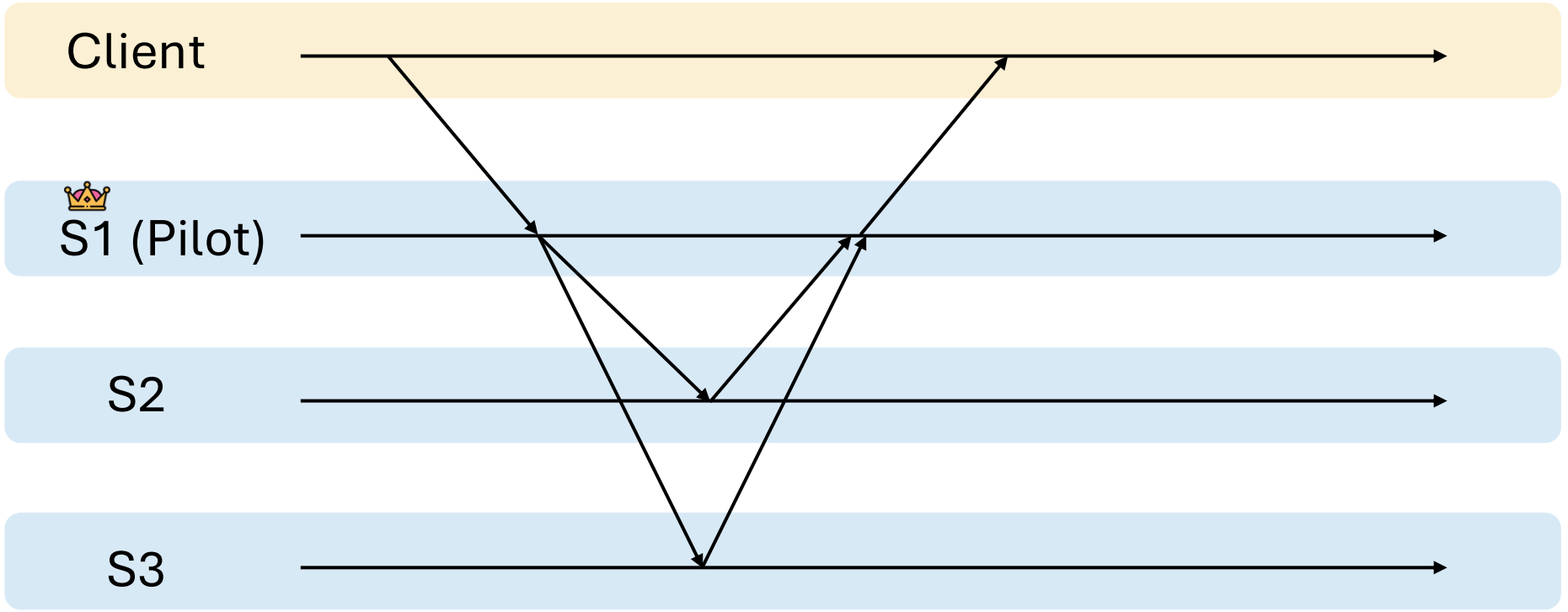
Jetpack

60 loc

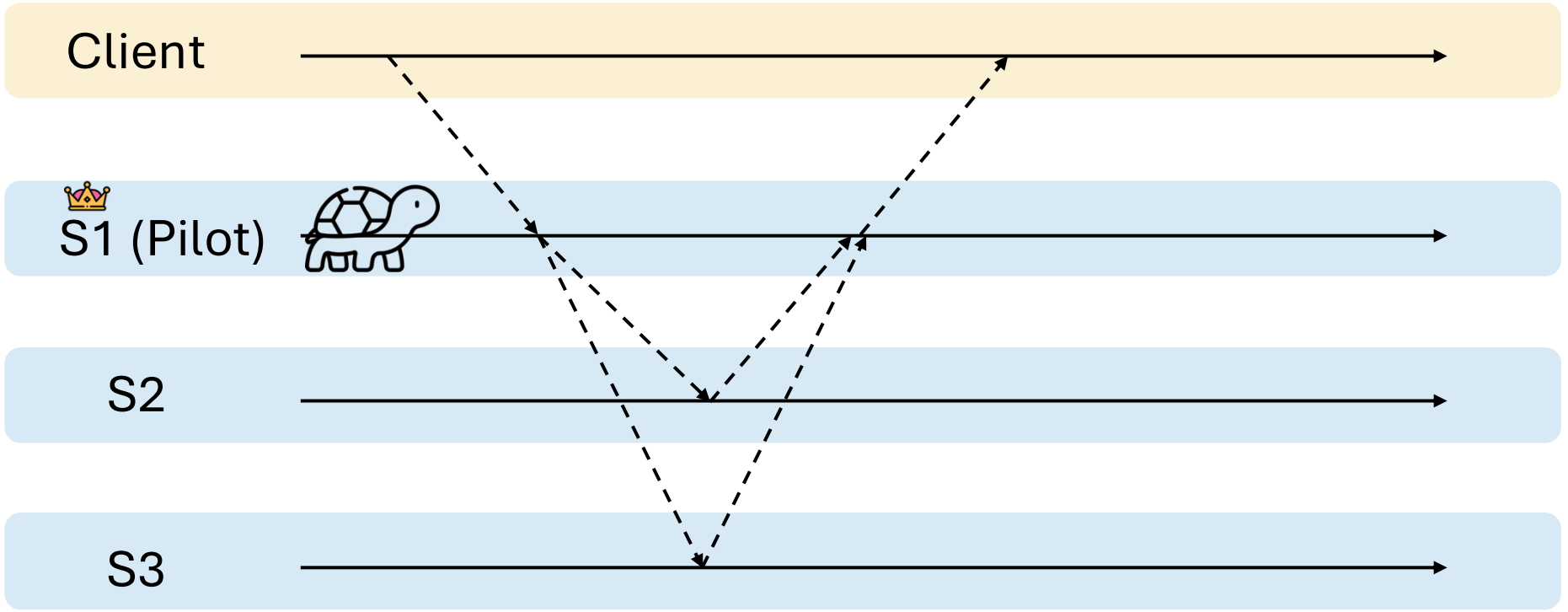
Latency reduction for MongoDB



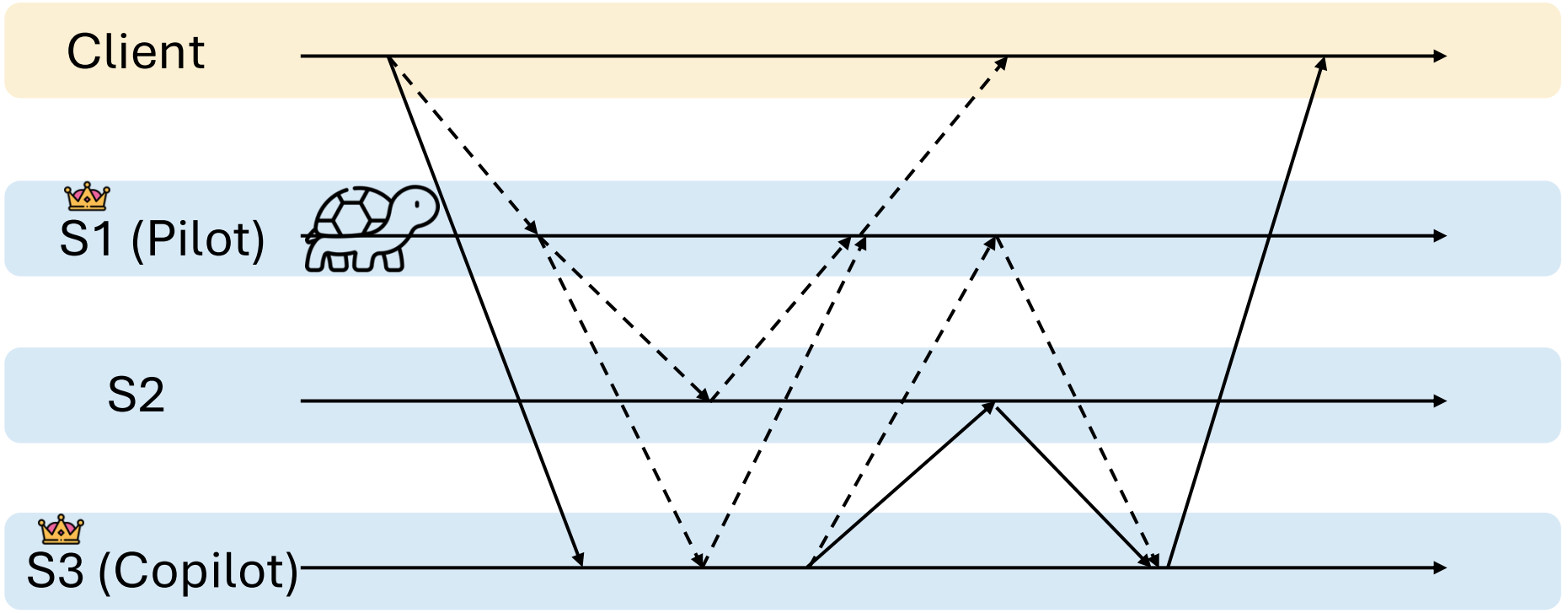
Copilot 1-slowdown-tolerance



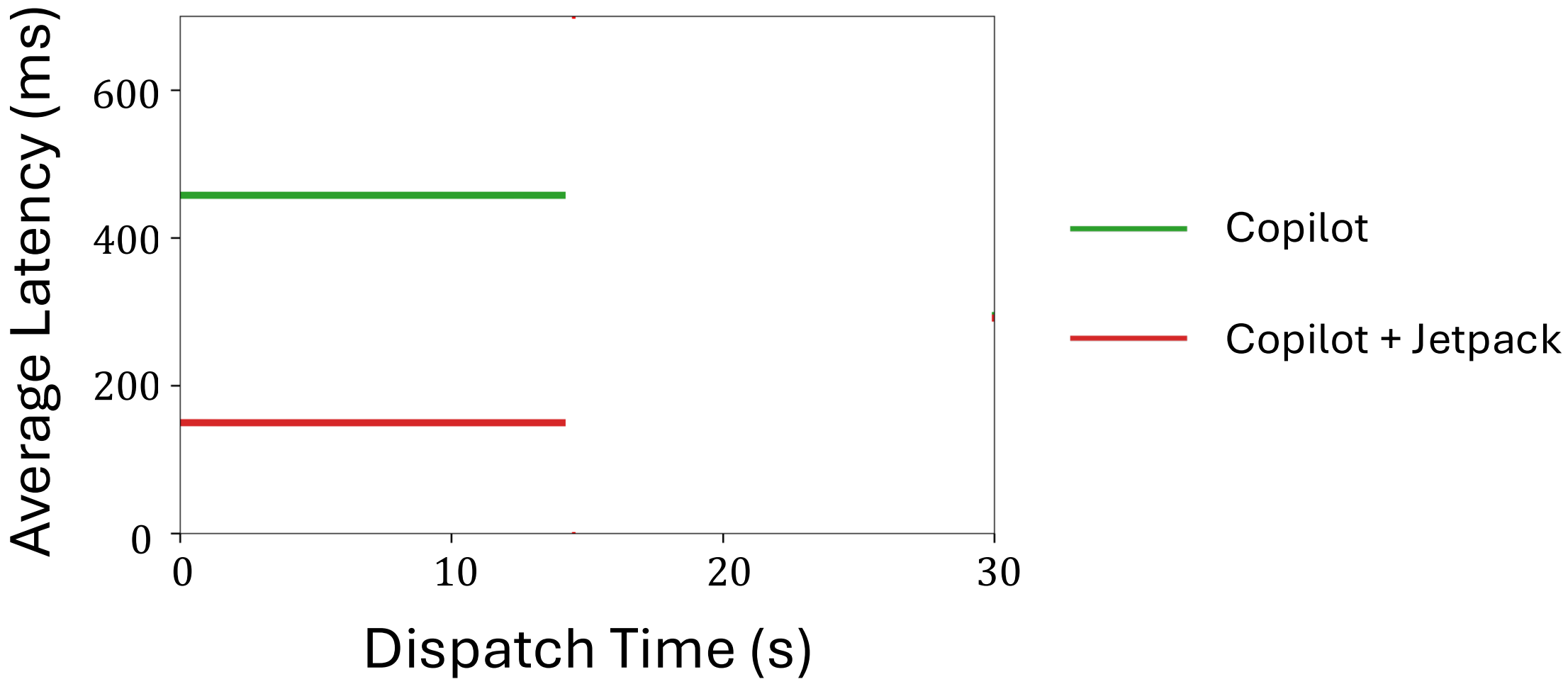
Copilot 1-slowdown-tolerance



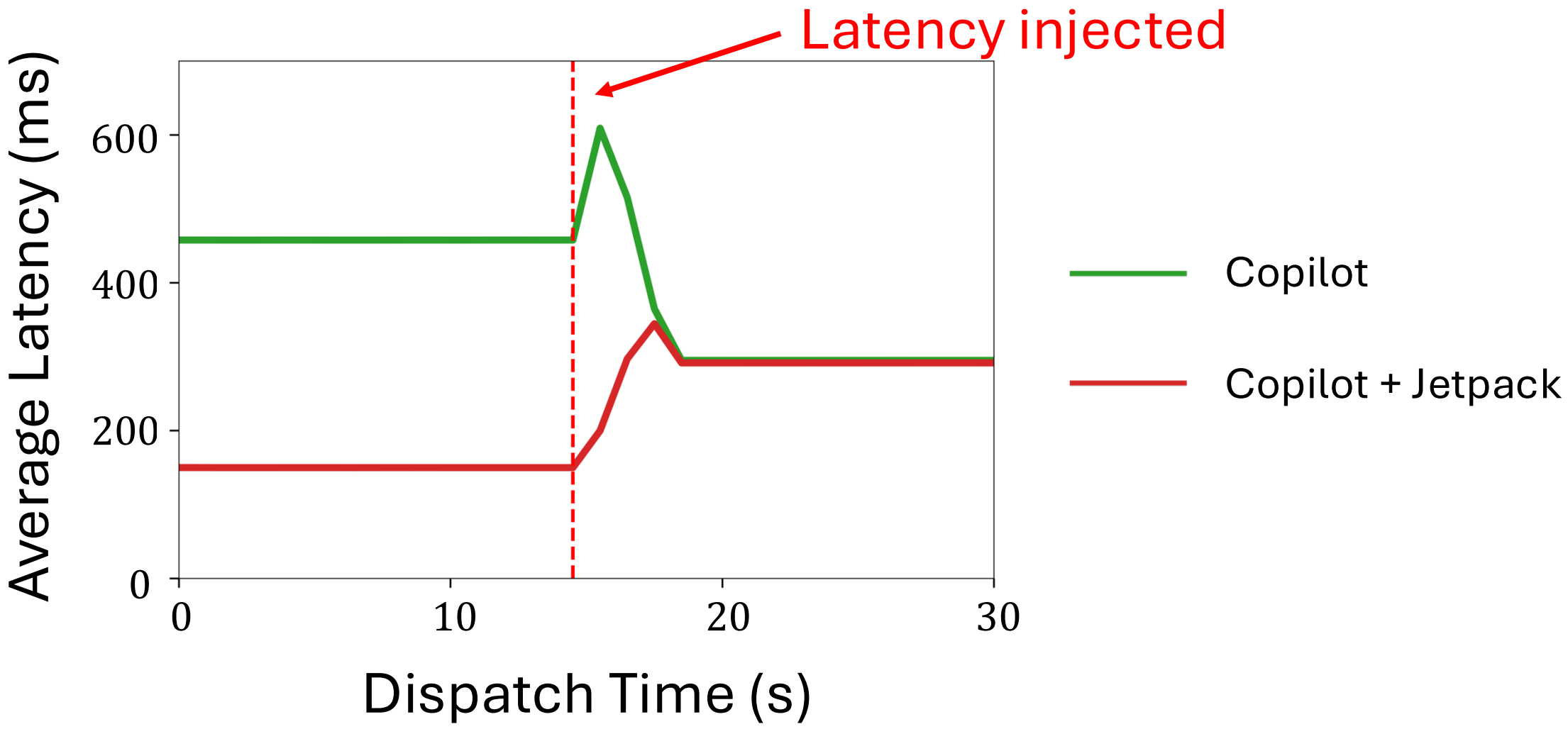
Copilot 1-slowdown-tolerance



Copilot 1-slowdown-tolerance



Copilot 1-slowdown-tolerance



Takeaway

- Jetpack is a general 1 RTT fast path that plugs into existing consensus systems.
  - Run the fast path and the original path concurrently.
  - A promise from the original path keeps the two consistent.

- The View Change Hazard
  - Case 1 Silent Loss
  - Case 2 Broken Order