Scalable Consistency in Scatter

A Distributed Key-Value Storage System

Lisa Glendenning
Ivan Beschastnikh
Arvind Krishnamurthy
Thomas Anderson

University of Washington

Supported by NSF CNS-0963754

SOSP October 2011

Internet services depend on distributed key-value stores

Scatter:
Goals

✓ linearizable consistency semantics

✓ scalable in a wide area network

✓ high availability

✓ performance close to existing systems

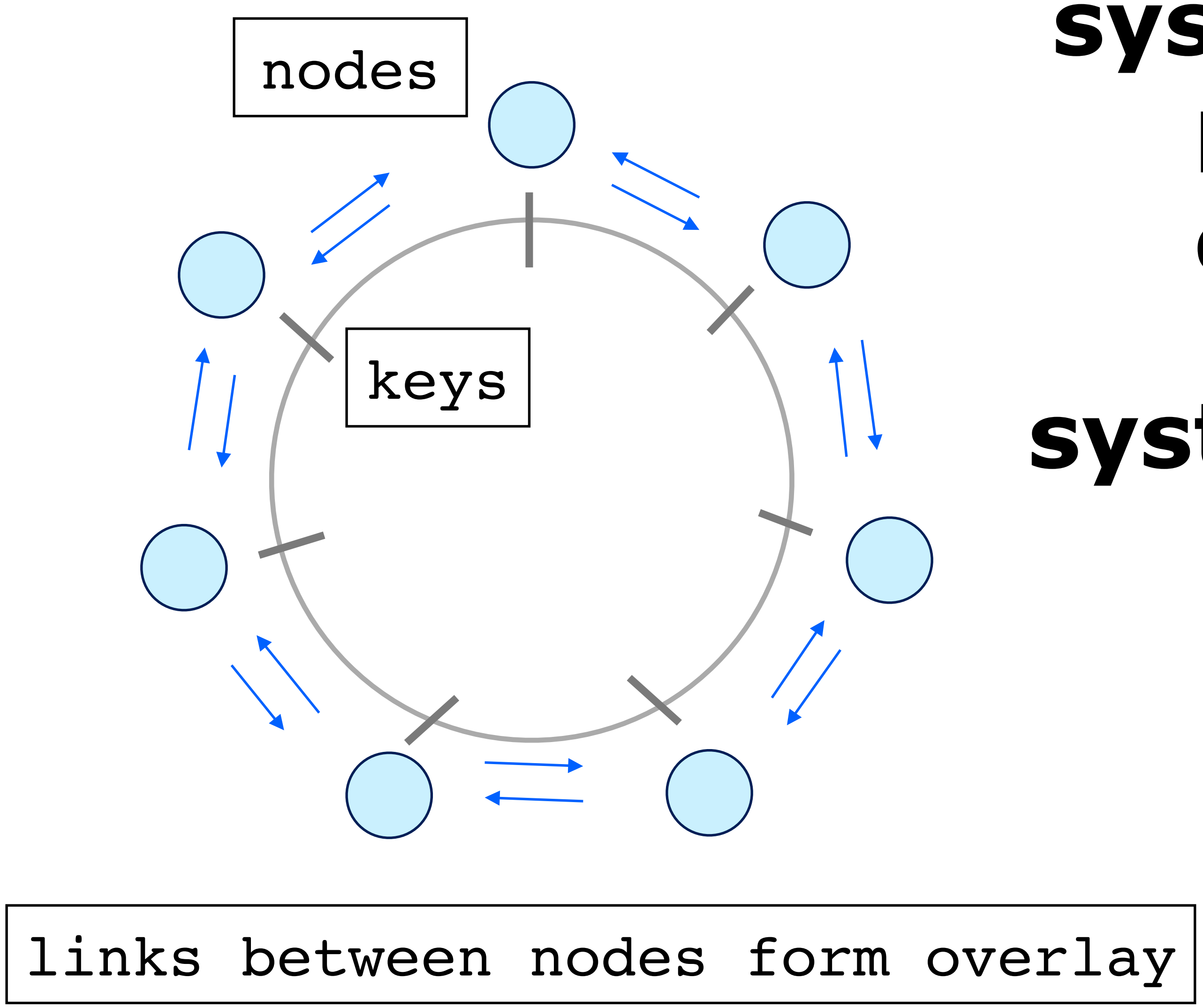
Scatter: Approach

combine ideas from:

scalable peer-to-peer systems	consistent datacenter systems
✓ distributed hash table	✓ consensus
✓ self-organization	✓ replication
✓ decentralization	✓ transactions

Distributed Hash Tables: Background

core functionality: partition and assign keys to nodes



system structure:

knowledge of system state is distributed among all nodes

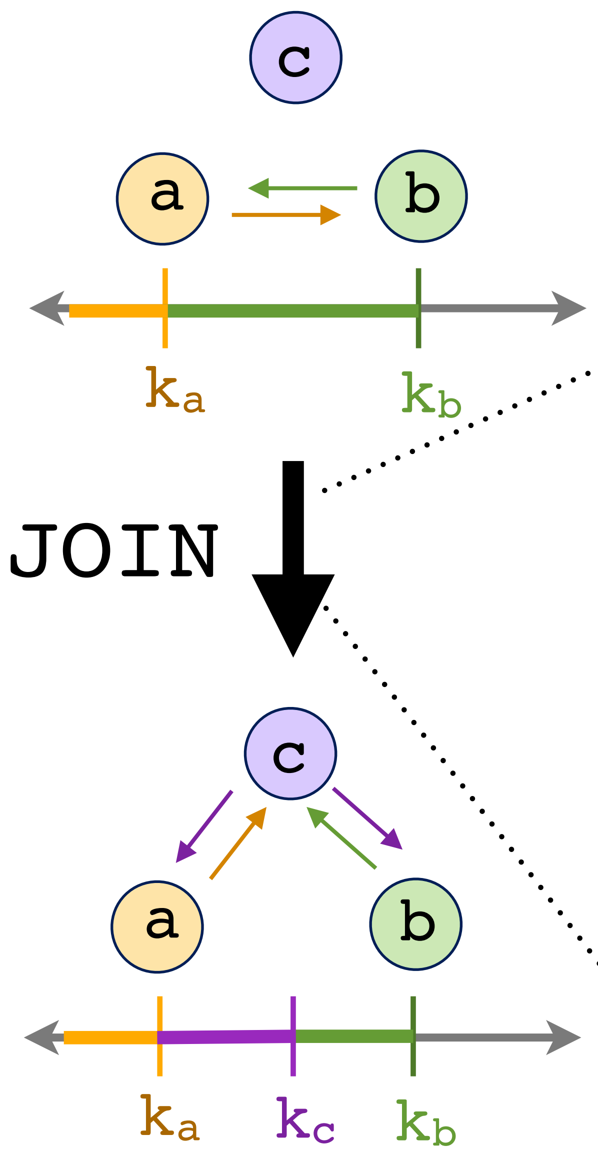
system management:

nodes coordinate locally to respond to churn, e.g.,

- give keys to new nodes
- take over keys of failed nodes

Distributed Hash Tables: Faults Cause Inconsistencies

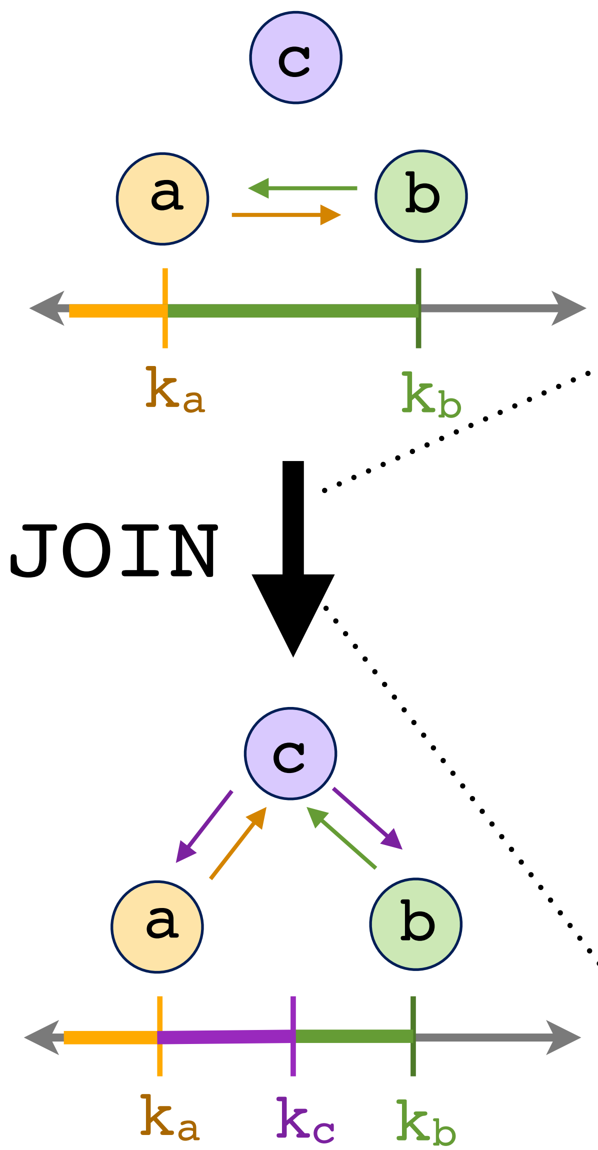
Example: c joins between a and b




```
c.pred = a
c.succ = b
a.succ = c
b.pred = c
b.keys = (kc,kb]
c.keys = (ka,kc]
```

Distributed Hash Tables: Faults Cause Inconsistencies

Example: c joins between a and b




```
c.pred = a
c.succ = b
a.succ = c
b.pred = c
b.keys = (kc,kb]
c.keys = (ka,kc]
```

what could go wrong?

FAULT	OUTCOME
communication fault between b and c	both b and c claim ownership of $(k_a, k_c]$
c fails during operation	no node claims ownership of $(k_a, k_c]$
communication fault between a and c	routes through a skip over c

Distributed Hash Tables: Weak Atomicity Causes Anomalies

DHTs use ad-hoc protocols to add and remove nodes

what happens if...

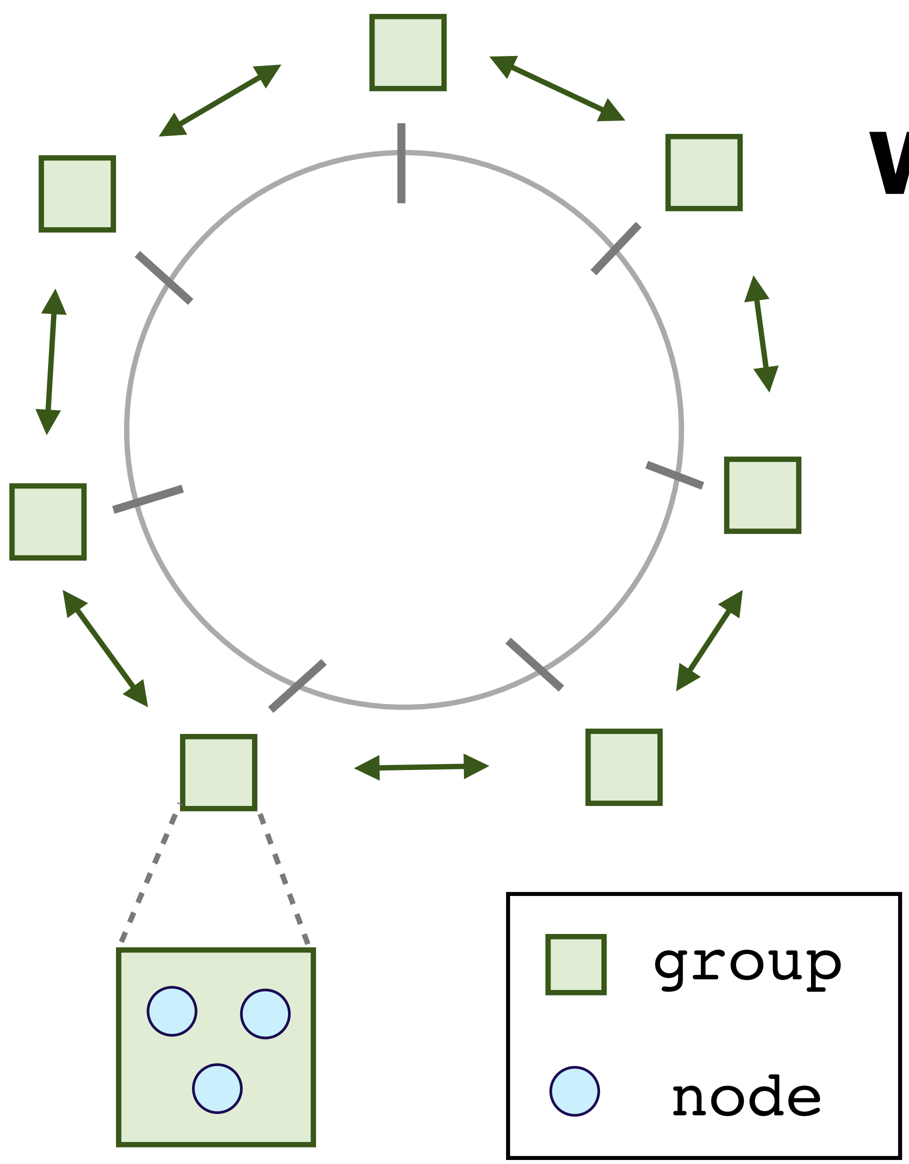
- two nodes join at the same place at the same time
- two adjacent nodes leave at the same time
- during a node join the predecessor leaves
- one node mistakenly thinks another node has failed

...

Scatter: Design Overview

How is Scatter different?

use *groups* as building blocks instead of nodes



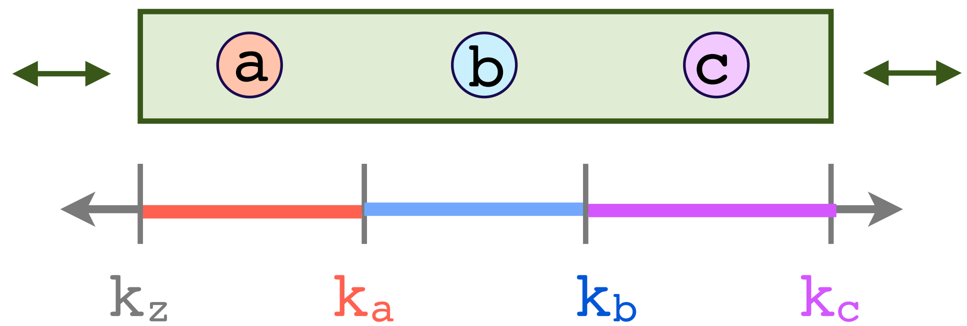
What is a group?

set of nodes that cooperatively manage a key-range

What does this give us?

- nodes within a group act as a single entity
- a group is much less likely to fail than an individual node
- distributed transactions for operations involving multiple groups

Scatter: Group Anatomy



- group replicates all state among members with Paxos

```
nodes = {a,b,c}
keys = (kz,kc]
values = {...}
```

- changes to group membership are Paxos reconfigurations:
  - include new nodes
  - exclude failed nodes

- key-range further partitioned among nodes of group for performance

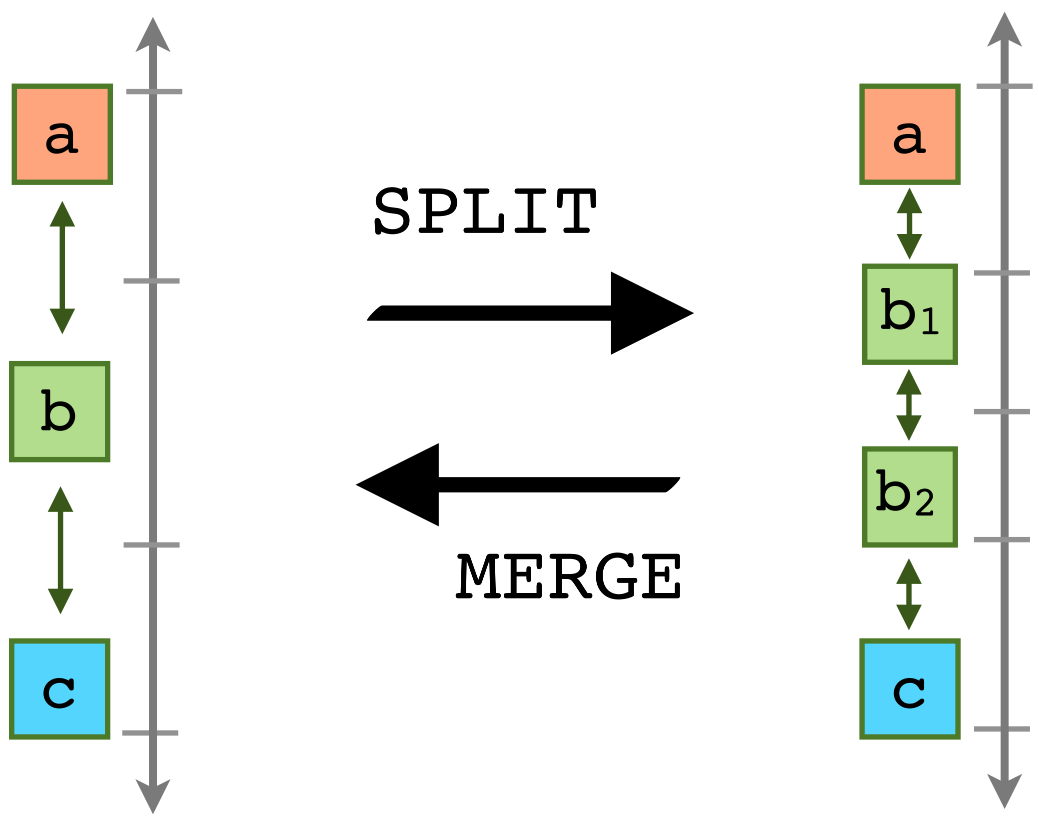
```
a.keys = (kz,ka]
b.keys = (ka,kb]
c.keys = (kb,kc]
```

- each node orders client operations on its keys

Scatter: Self-Reorganization

<u>some problems can't be handled within a single group</u>

- small groups are at risk of failing
- large groups are slow
- load imbalance across groups



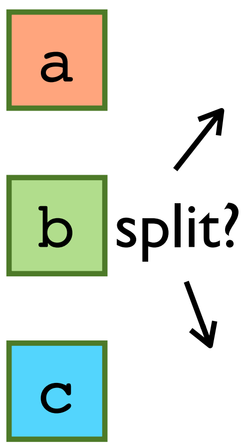
multi-group operations:

- *merge* two small groups into one
- *split* one large group into two
- *rebalance* keys and nodes between groups

distributed transactions coordinated locally by groups

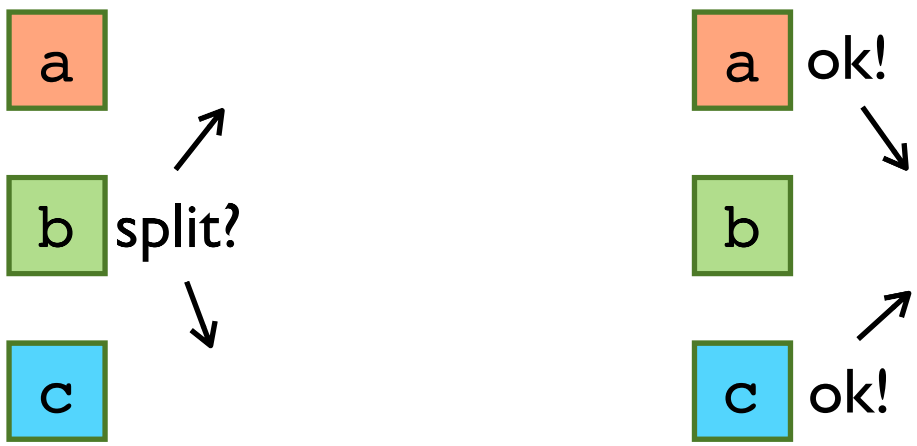
Example: Group Split

2PC



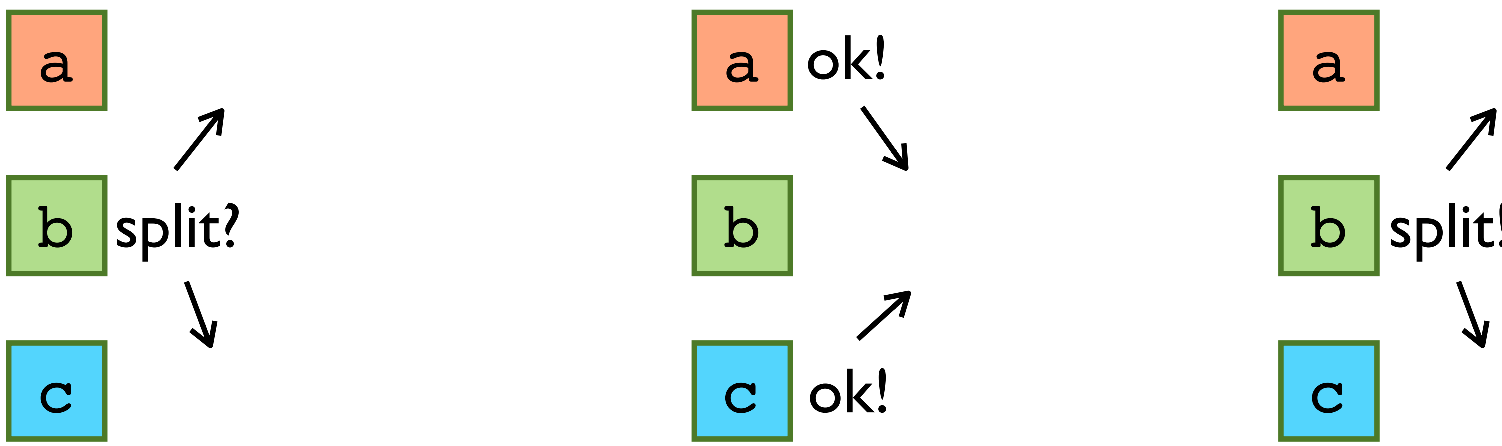
Example: Group Split

2PC



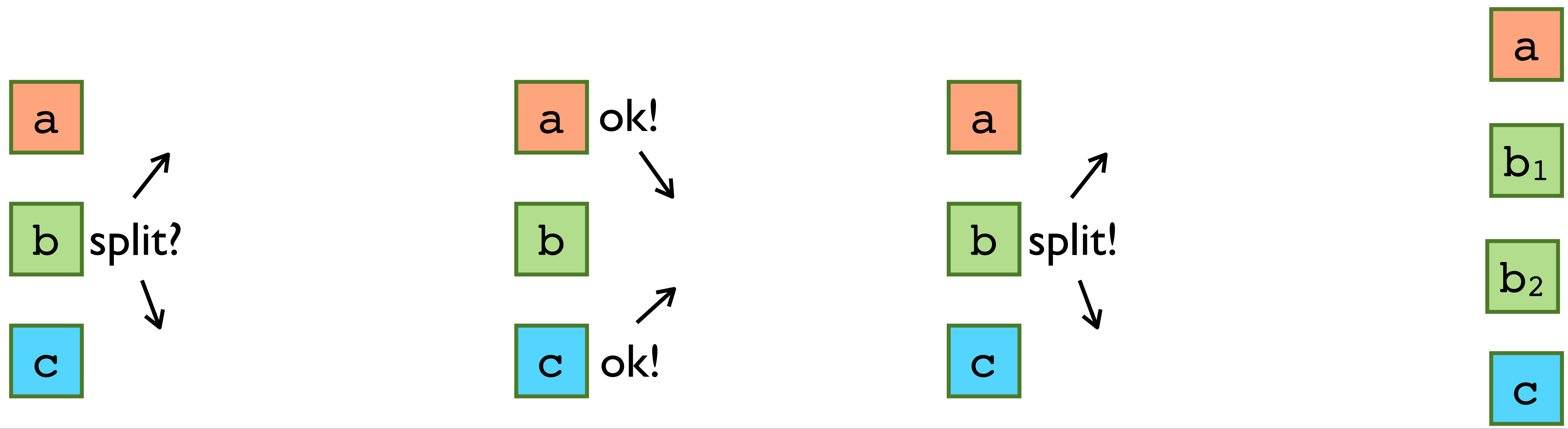
Example: Group Split

2PC



Example: Group Split

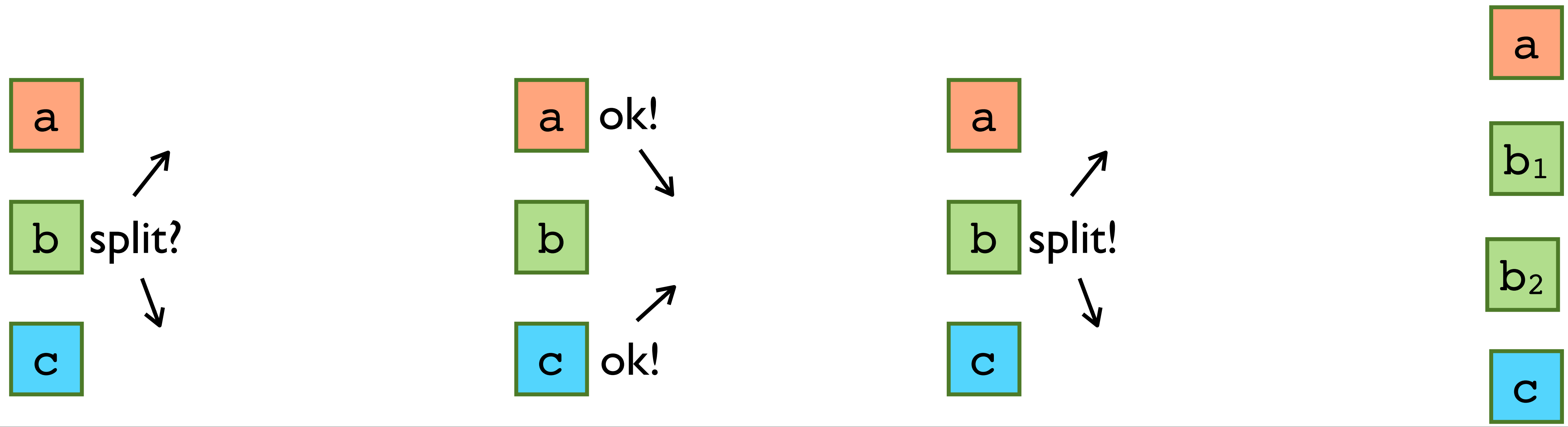
2PC

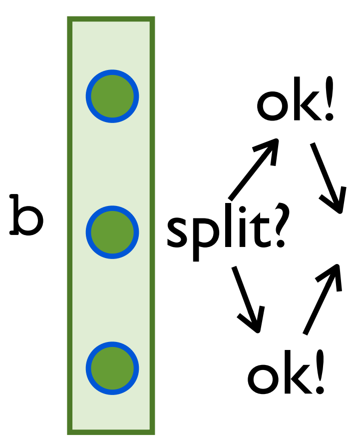


11

Example: Group Split

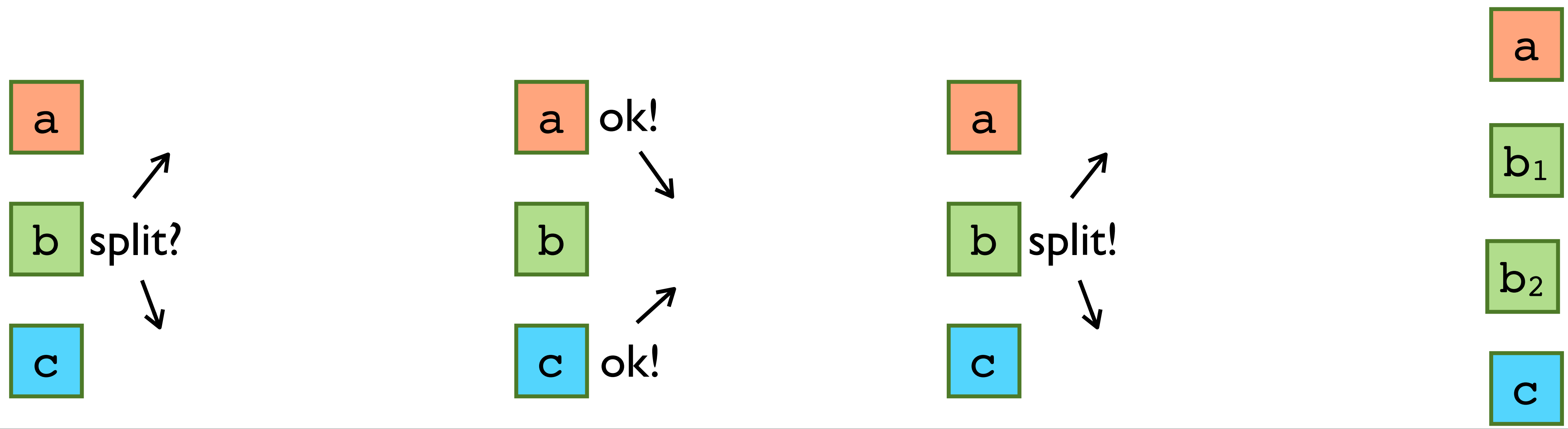
2PC

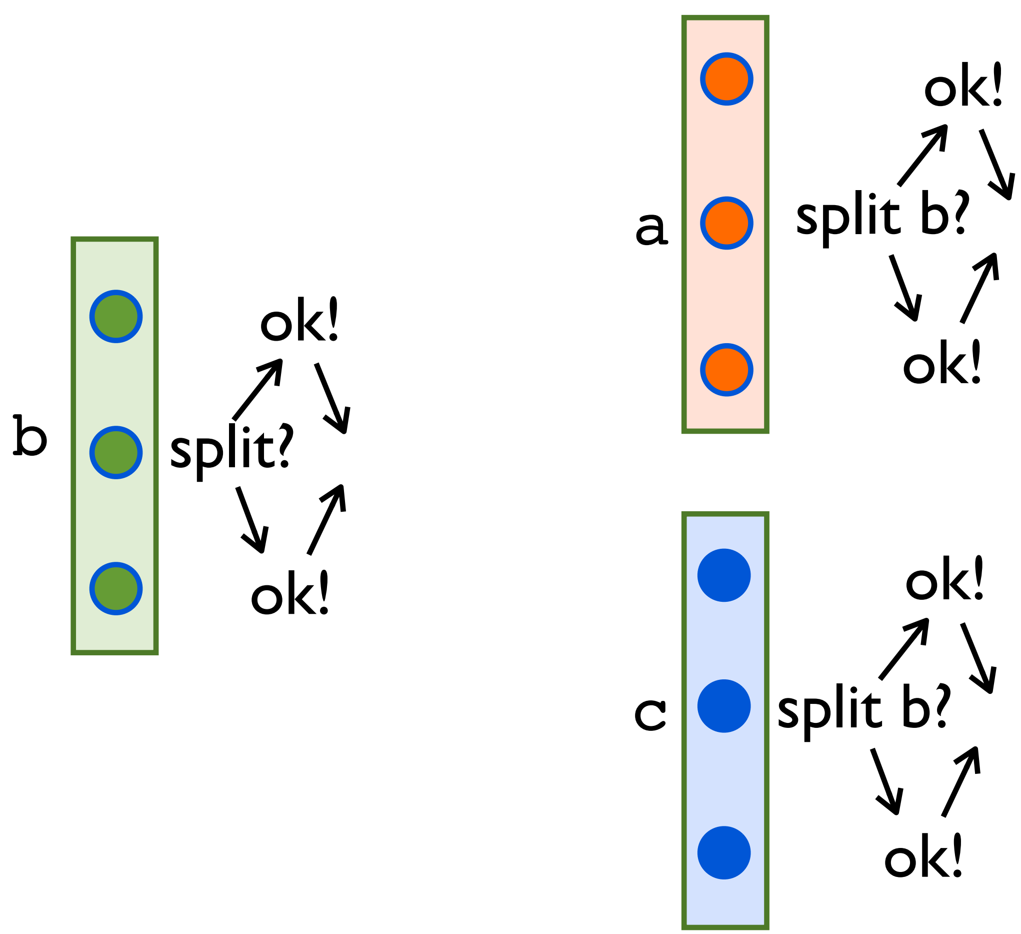




Example: Group Split

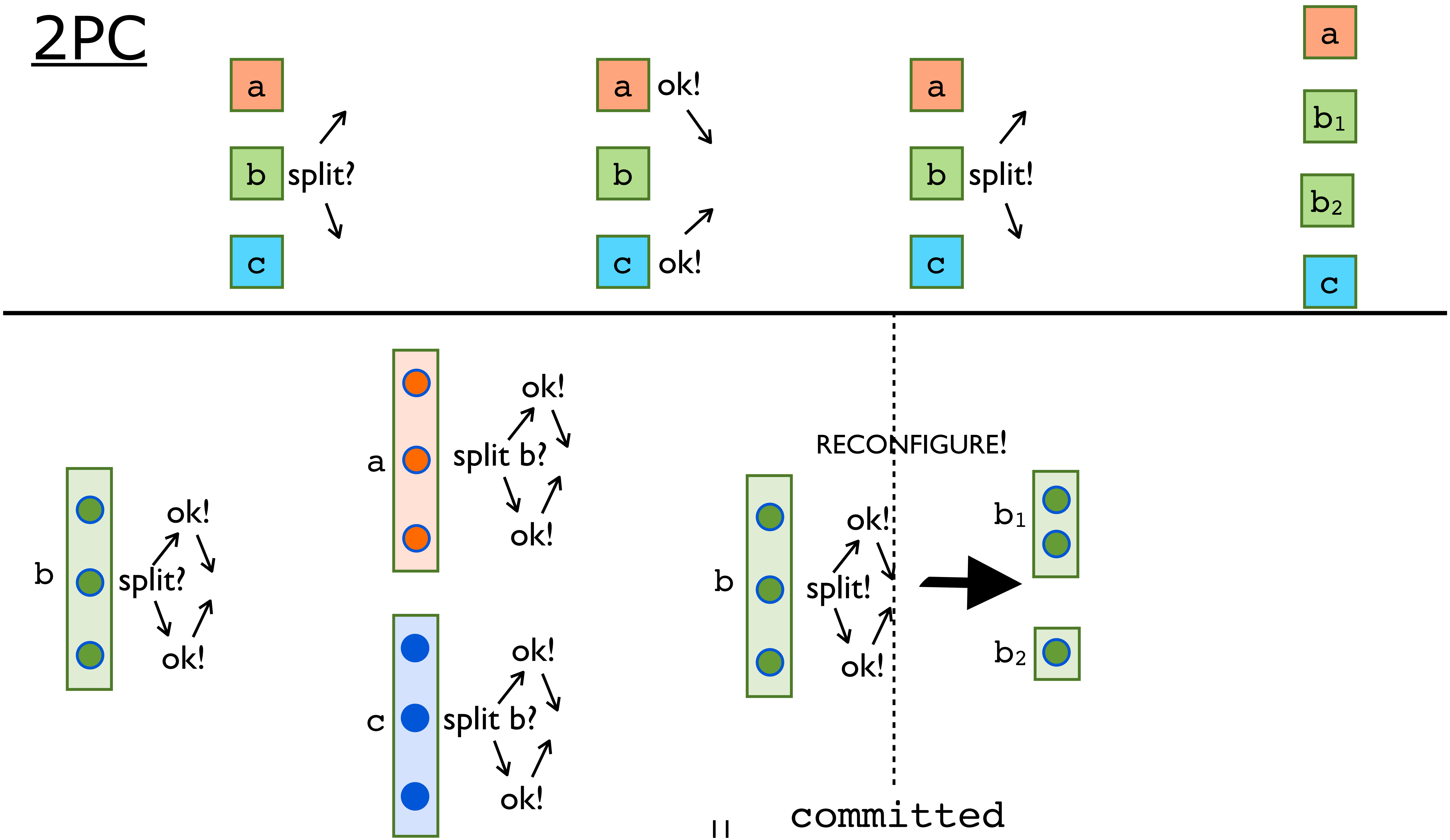
2PC



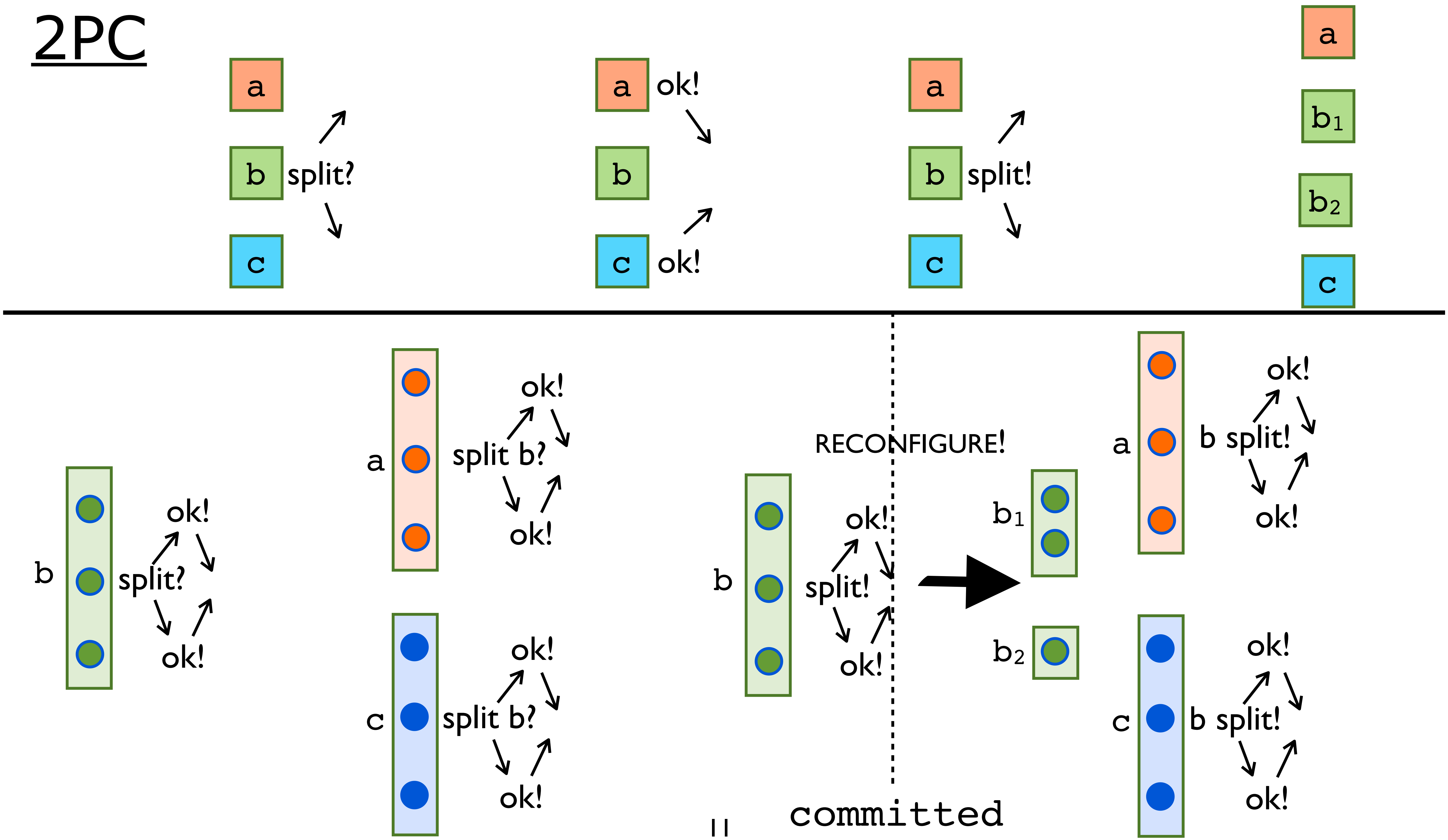


11

Example: Group Split



Example: Group Split



Scatter

- ✓ linearizable consistency semantics
  ...group consensus, transactions
- ✓ scalable in a wide area network
  ...local operations
- ✓ high availability
  ...replication, reconfiguration
- ✓ performance close to existing systems
  ...key partitioning, optimizations

Evaluation: Overview

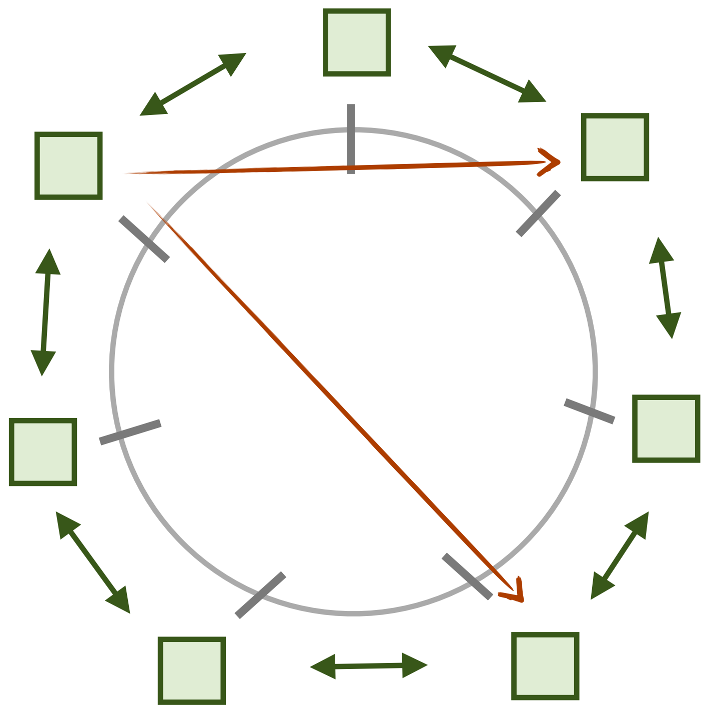
Questions:

1. How robust is Scatter in high-churn peer-to-peer environment?
2. How does Scatter adapt to dynamic workload in datacenter environment?

Comparisons:

Environment	**P2P**	**Datacenter**
Comparison System	OpenDHT	ZooKeeper

Comparison: OpenDHT



Layered OpenDHT's recursive routing on top of Scatter groups

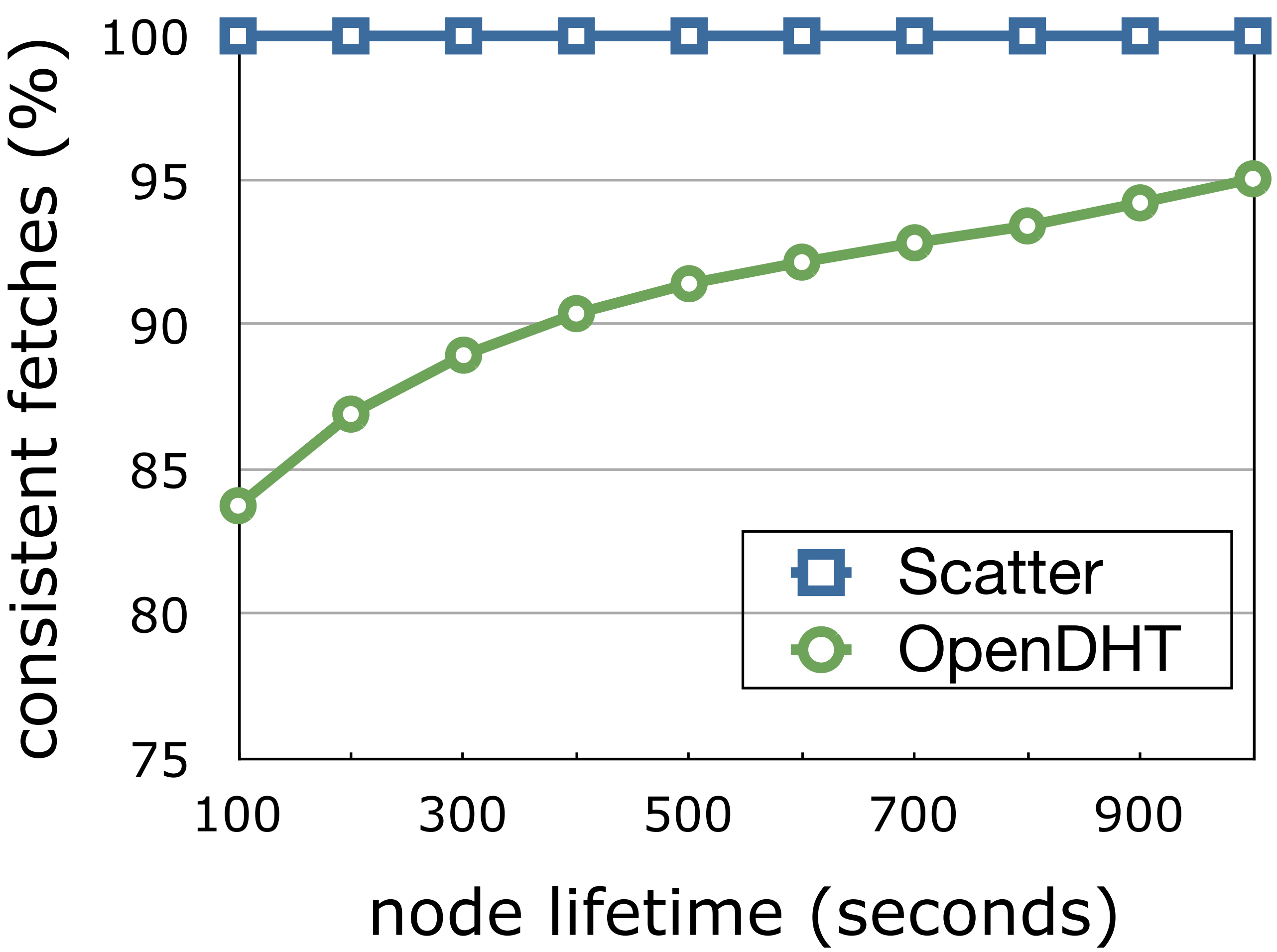
Implemented a Twitter-like application, Chirp

Experimental Setup:

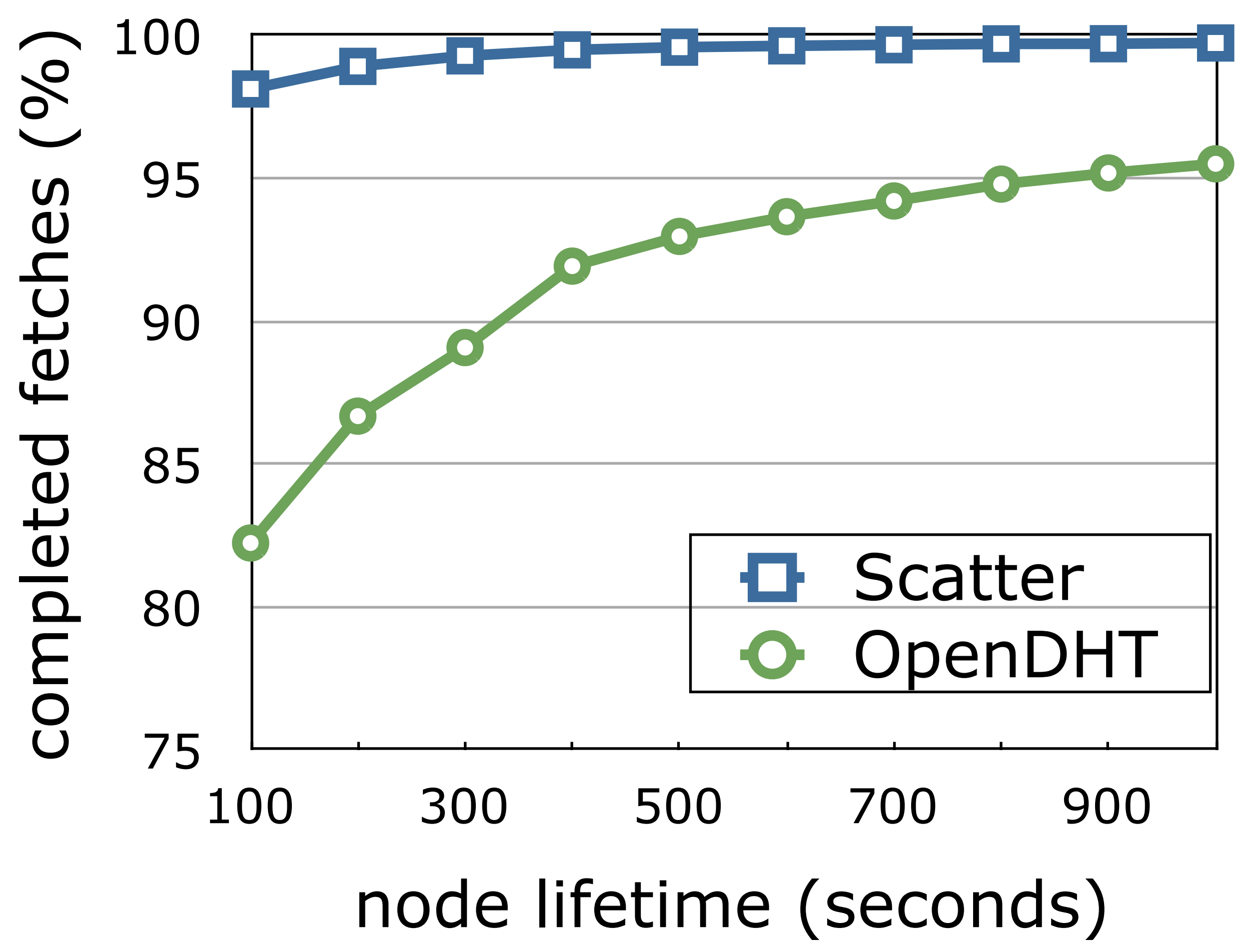
- 840 PlanetLab nodes
- injected node churn at varying rates
- Twitter traces as a workload
- tweets and social network stored in DHT

Comparison: OpenDHT

Consistency



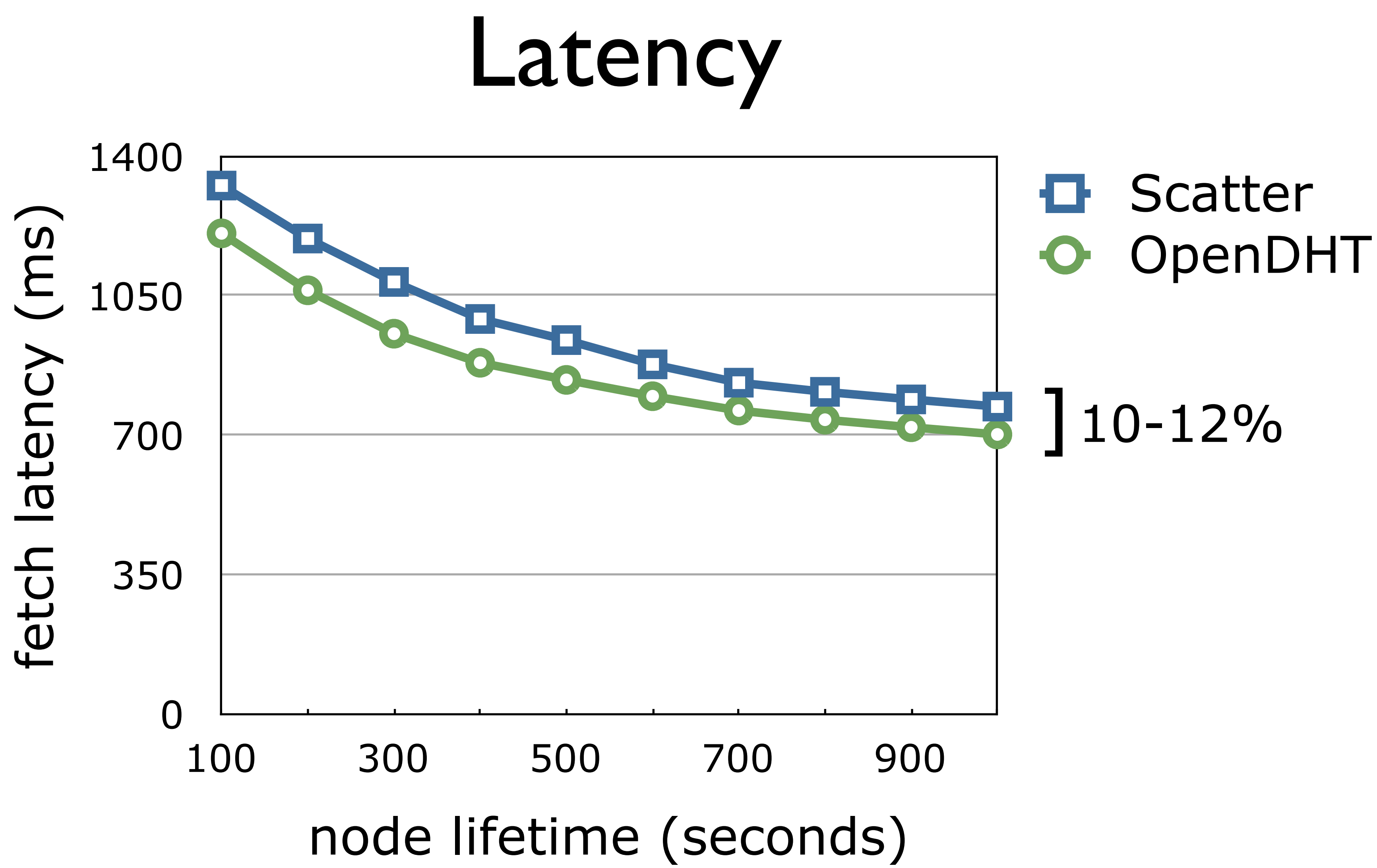
Availability



Scatter has zero inconsistencies and high availability even under churn

15

Comparison: OpenDHT



Scalable consistency is cheap

16

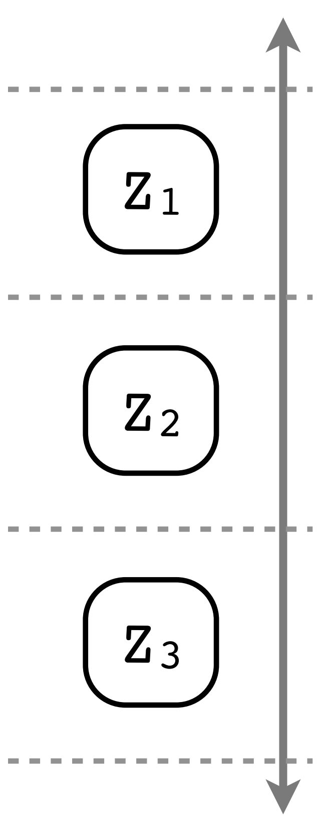
Comparison: Replicated ZooKeeper

ZooKeeper:

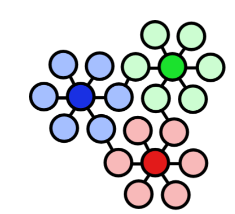
small-scale, centralized coordination service

Replicated ZooKeeper:

statically partitioned global key-space to multiple, isolated ZooKeeper instantiations

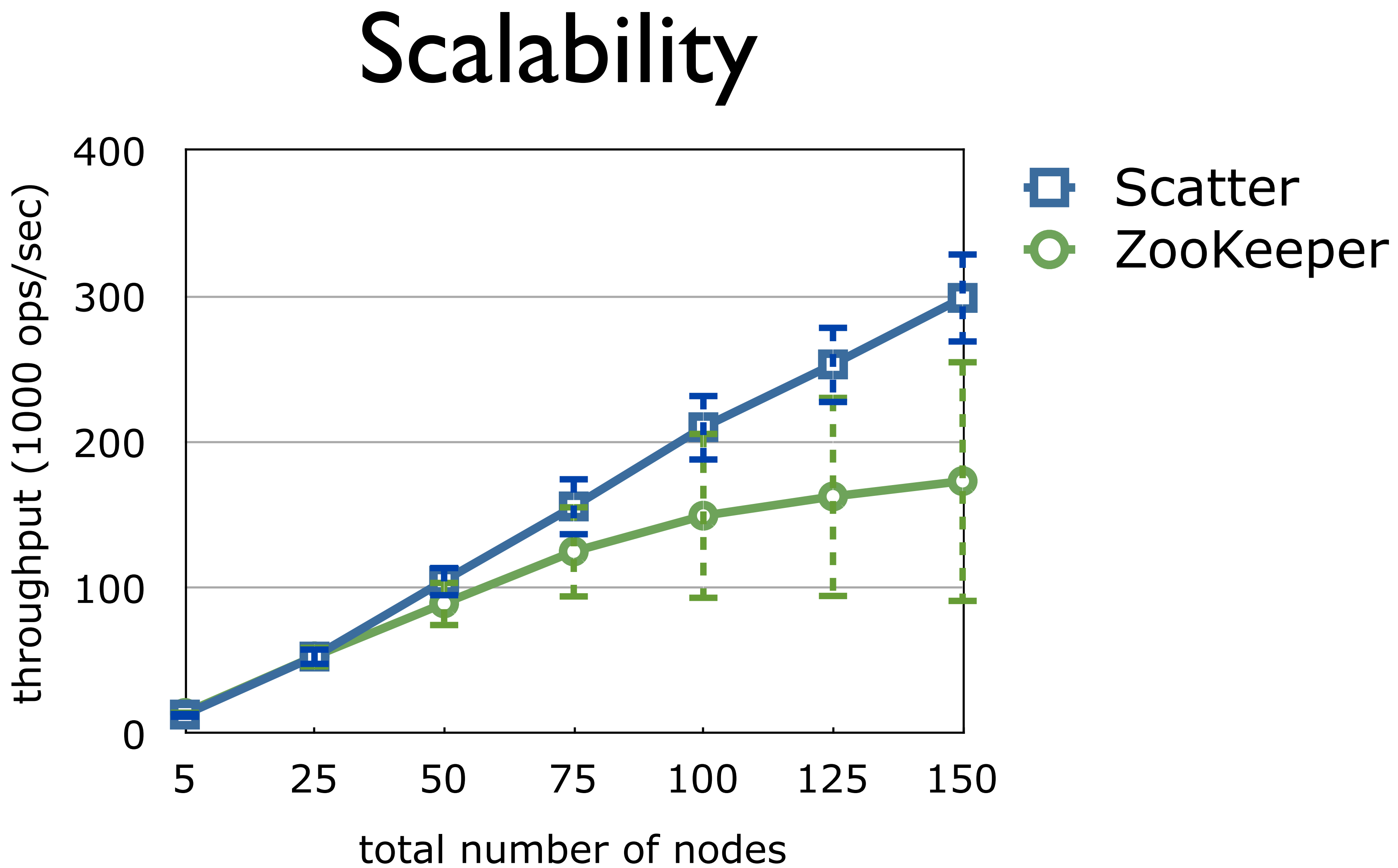


Experimental Setup:



- testbed: Emulab
- varied total number of nodes
- no churn
- same Chirp workload

Comparison: Replicated ZooKeeper



Dynamic partitioning adapts to changes in workload

Scatter: Summary

- ✓ consensus groups of nodes as fault-tolerant building blocks
- ✓ distributed transactions across groups to repartition the global key-space
- ✓ evaluation against OpenDHT and ZooKeeper shows strict consistency, linear scalability, and high availability