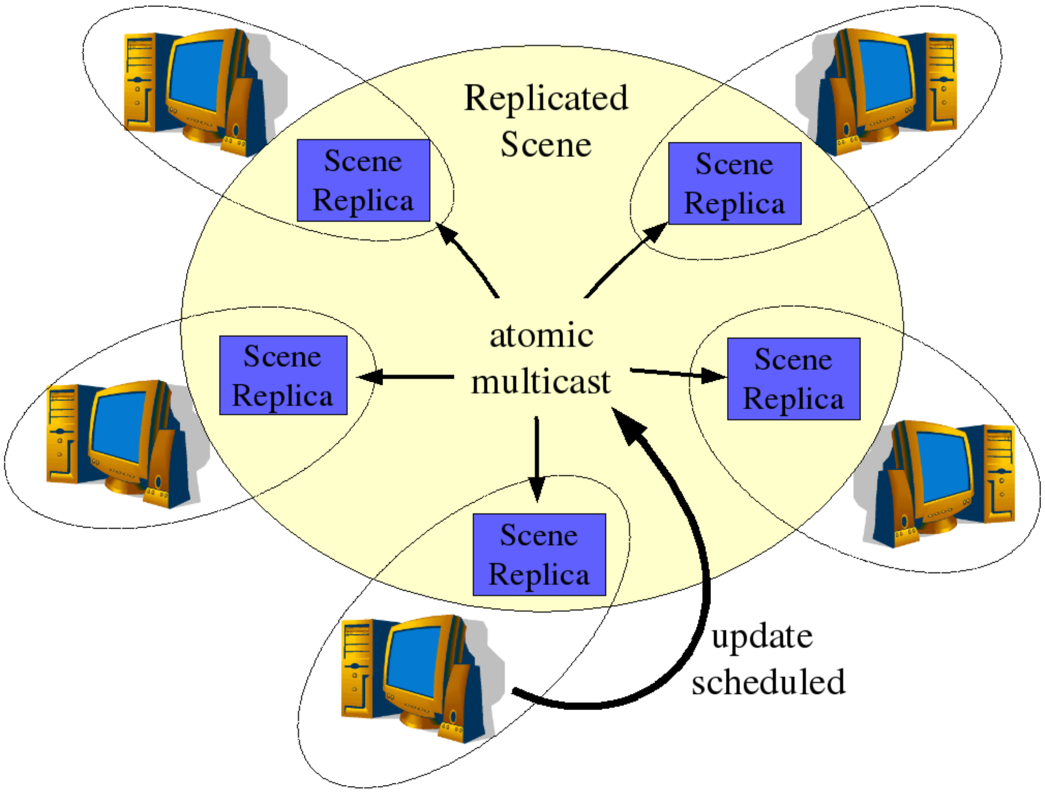
Replication



https://en.wikipedia.org/wiki/File:Active_Replica

Outline

- Data replication
- Replication techniques
- Chain replication
- Gossiping

DATA REPLICATION

Why Replicate?

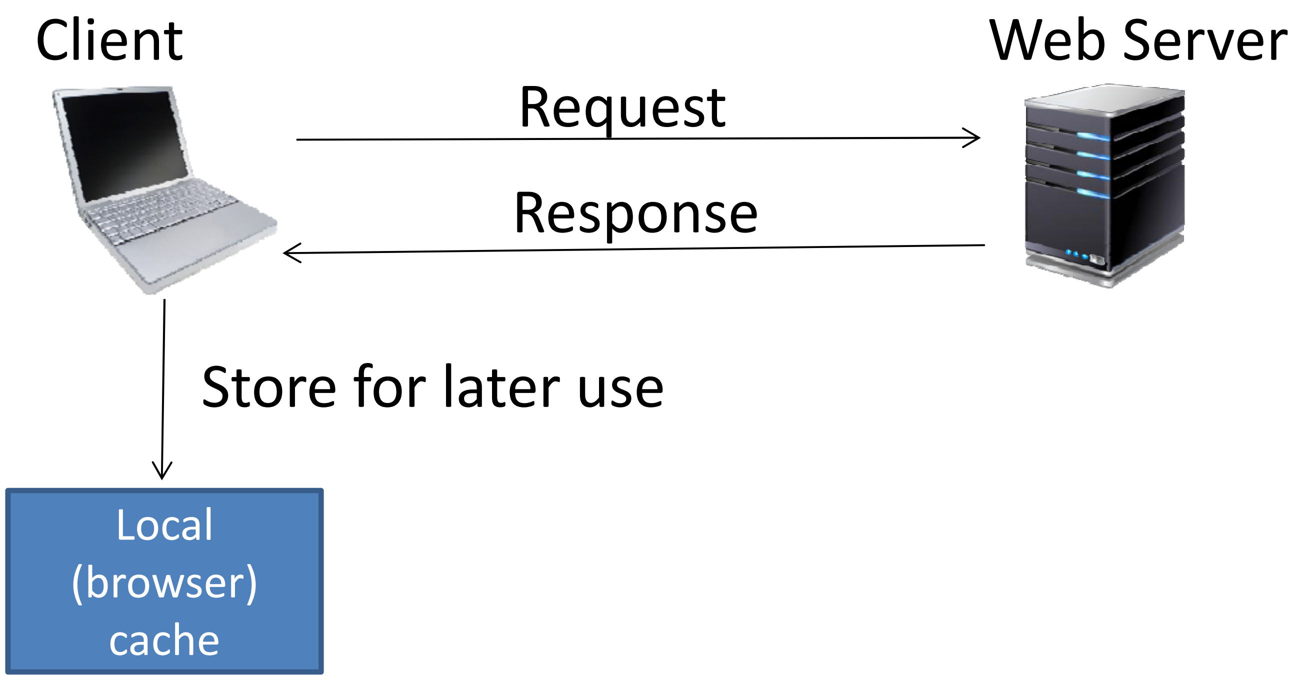
- Performance
- High availability
- Fault tolerance

Today, we look at **data replication**;

Computation replication is another concept (cf. MapReduce)

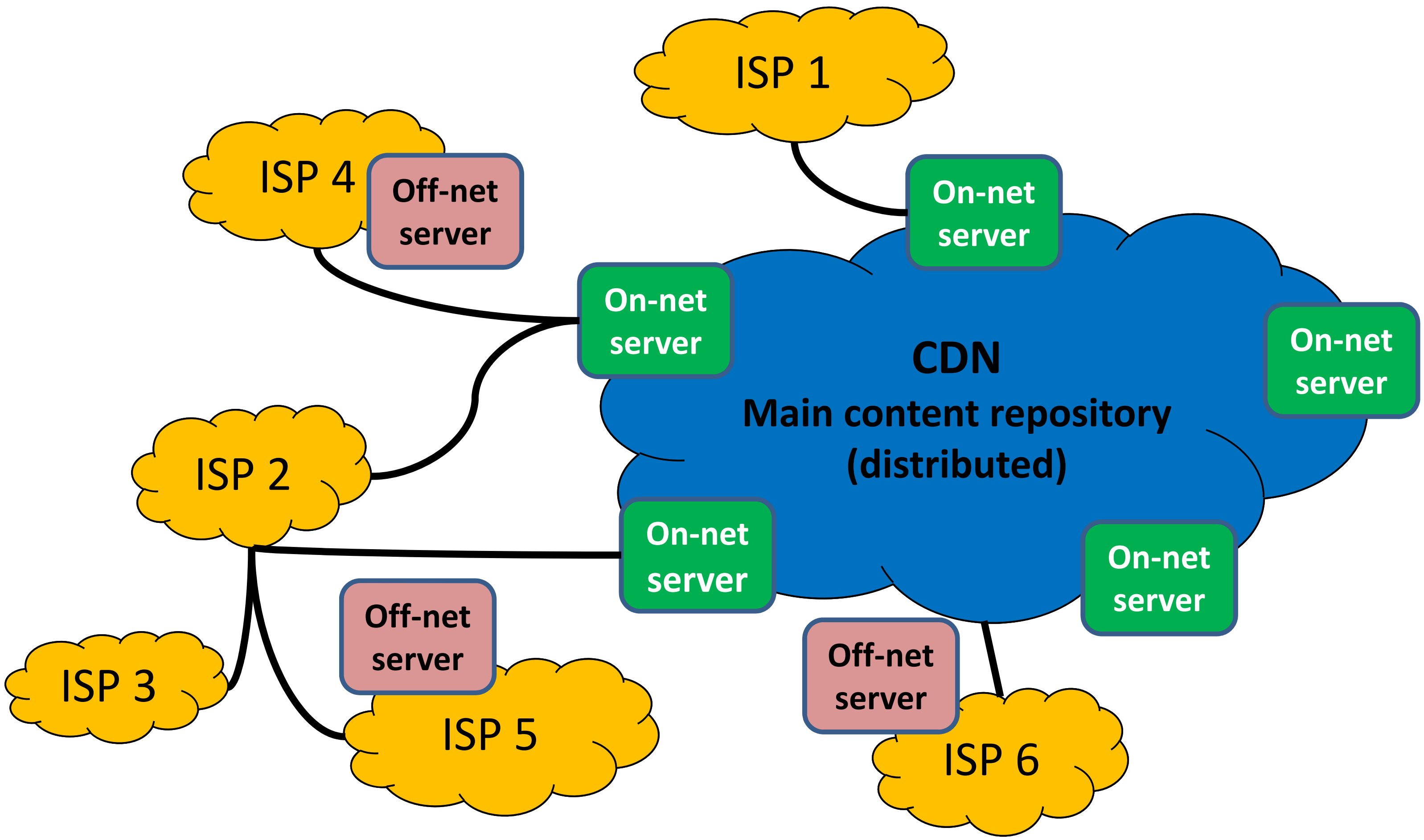
Performance

Caching data at **browsers** and **proxy servers**



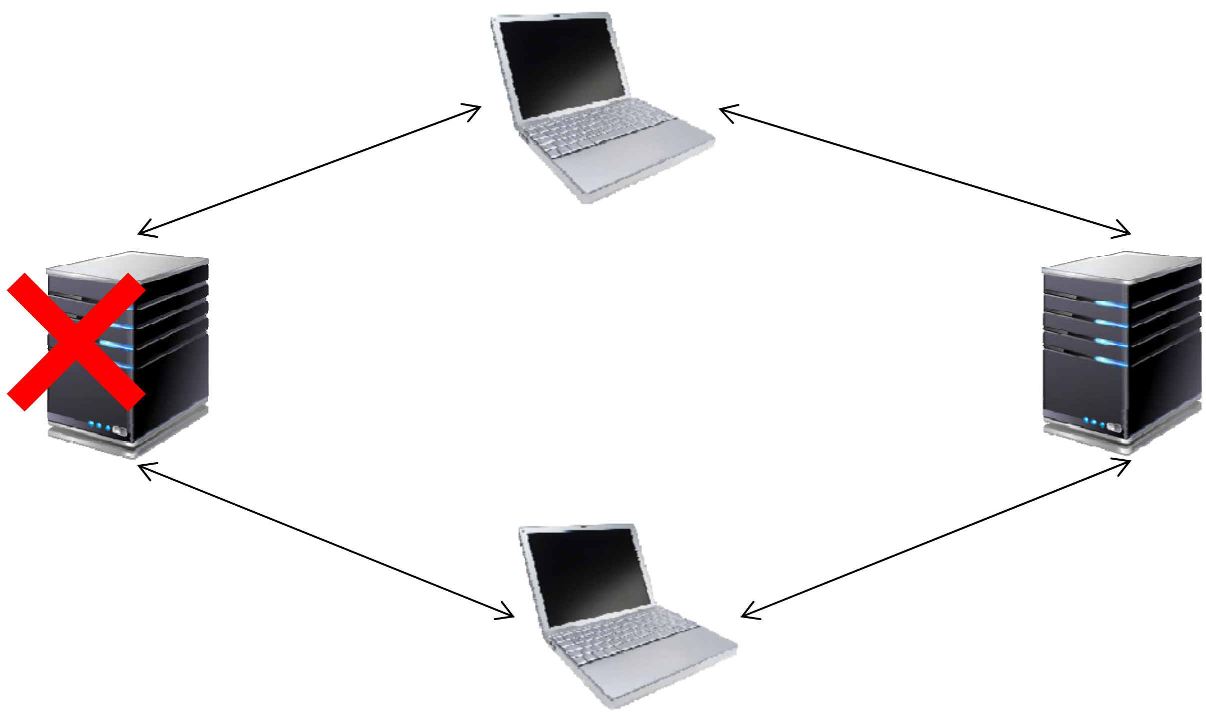
Typical (video) CDN

(Content Delivery Network)



High Availability

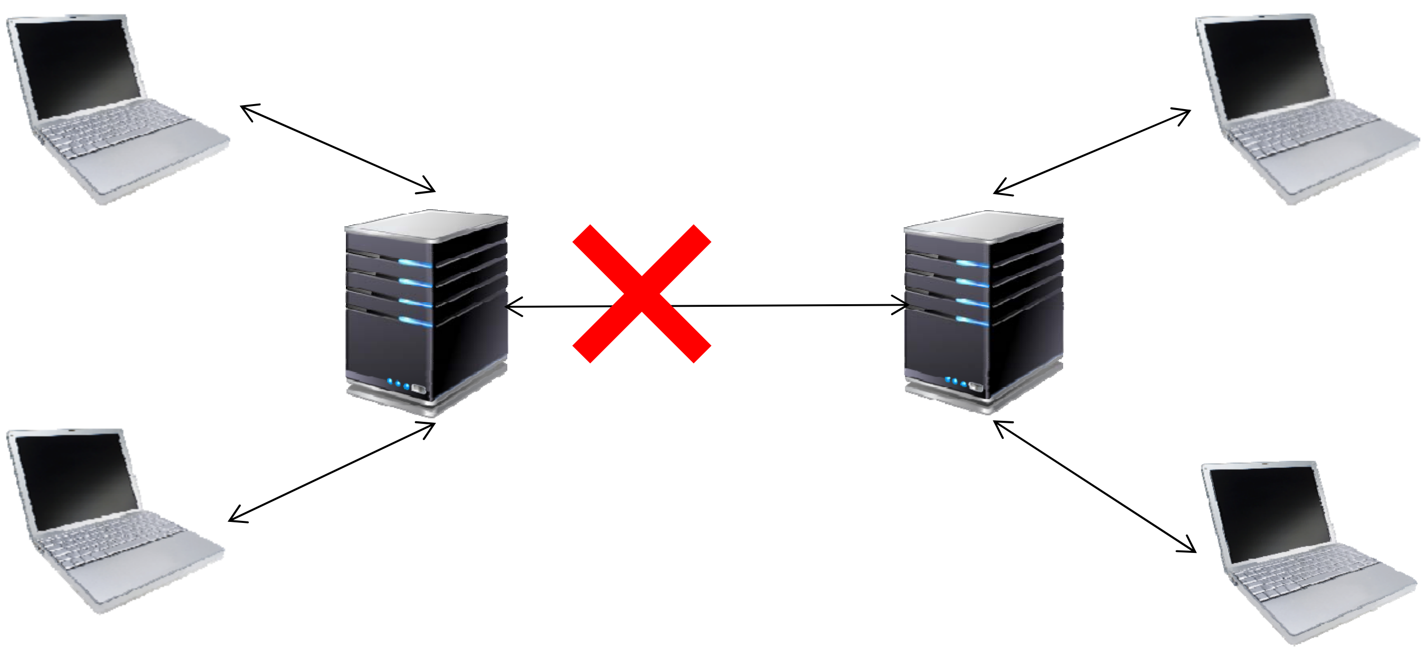
Upon crashes, data offered/retrieved by/from replica



[1] The availability of a service by replicating its servers would grow.

High Availability: Network Partitioning

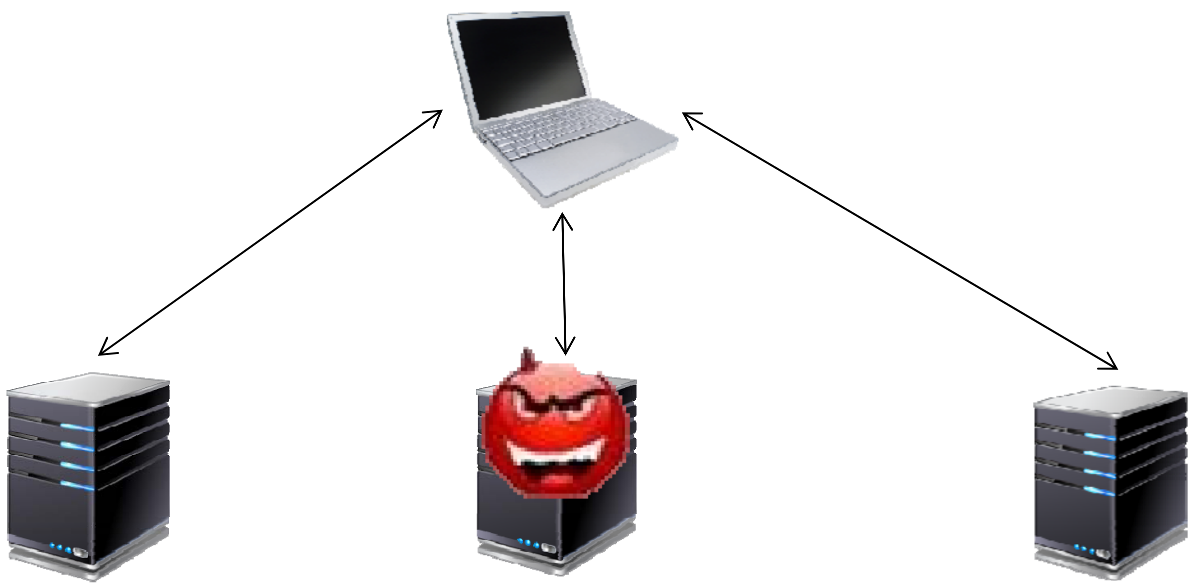
Upon network partition, data
available to clients from within partition



Partition tolerance: A system that continues to operate in face of network partitions

High Availability: Fault Tolerance

Providing reliable service in face of faulty servers
(not just crashes, but also arbitrary failures!)



“Cost” of Replication

- Not just cost of **storing** additional copies of data
  - I.e., additional bandwidth, number of messages exchanged, higher latency (i.e., response time), complexity of code, etc.

- Cost to **keep replicas up to date** in face of updates

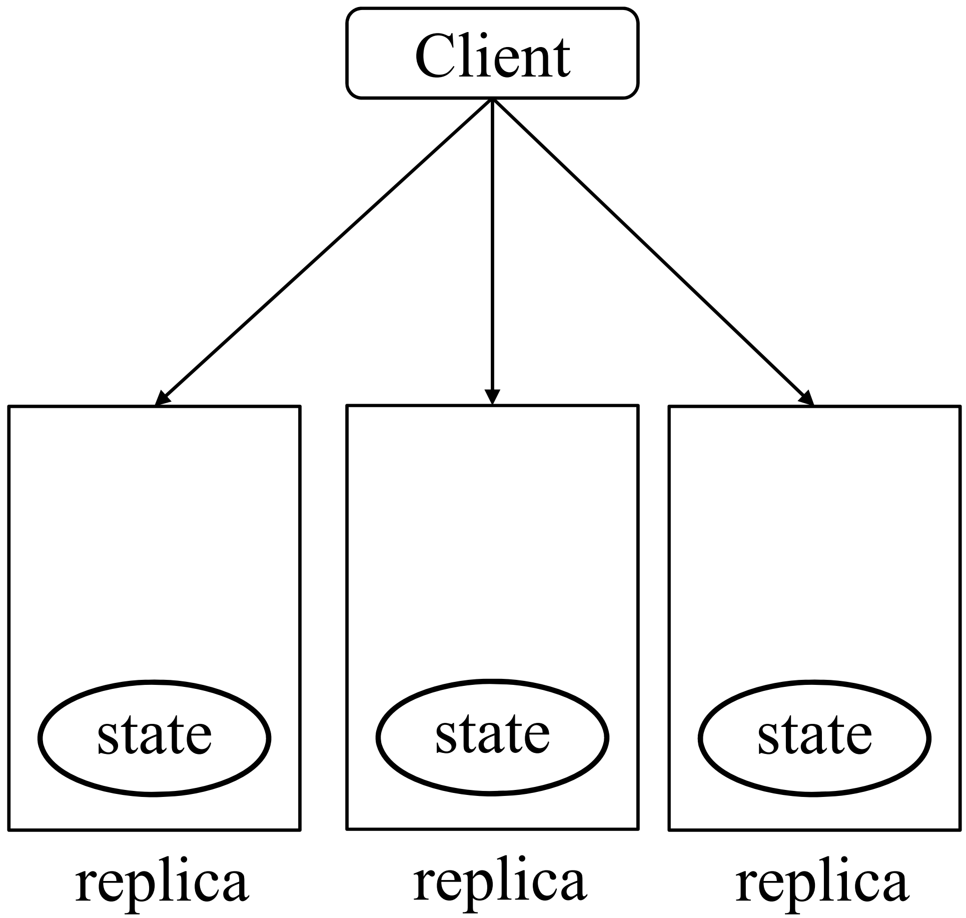
- *How to deal with **stale** (out-of-date) **data at replicas?***

REPLICATION TECHNIQUES

UPDATE PROCESSING

Replication Techniques

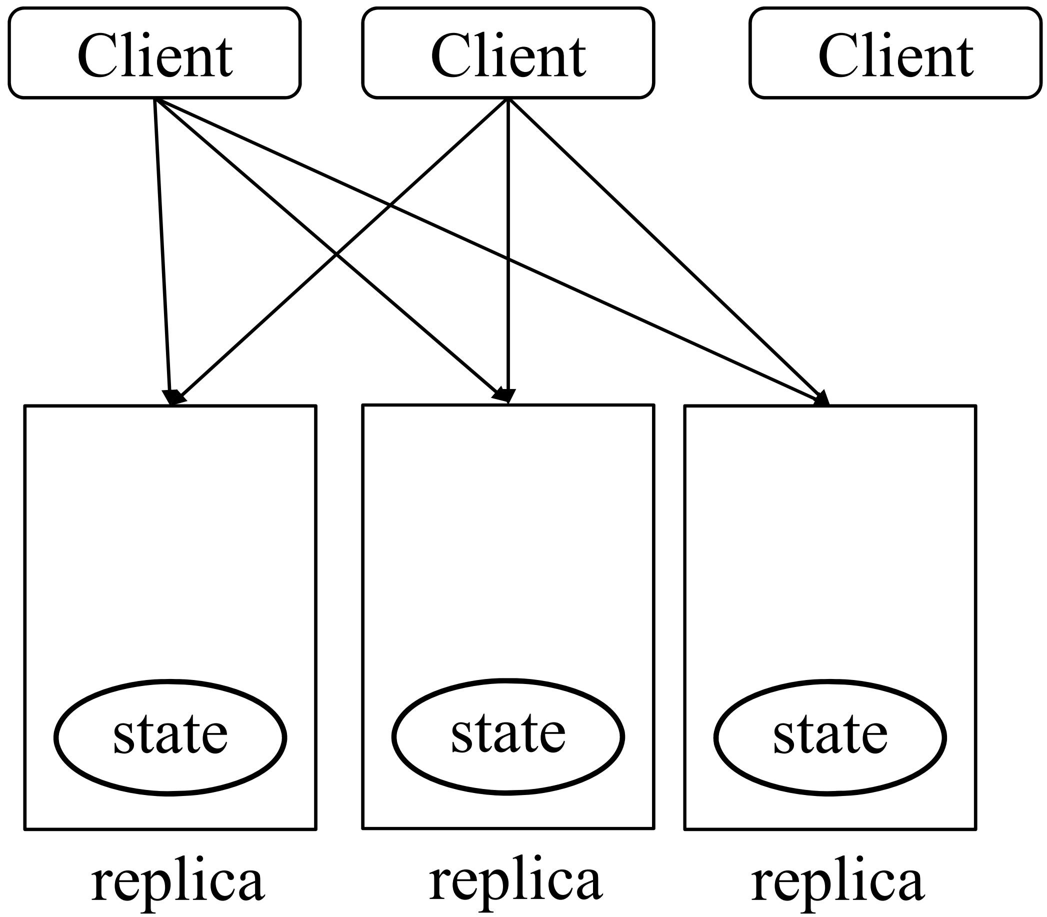
- **Active** replication
- Requests can be reads or writes
- Processes can crash



Replication Techniques

- **Active** replication

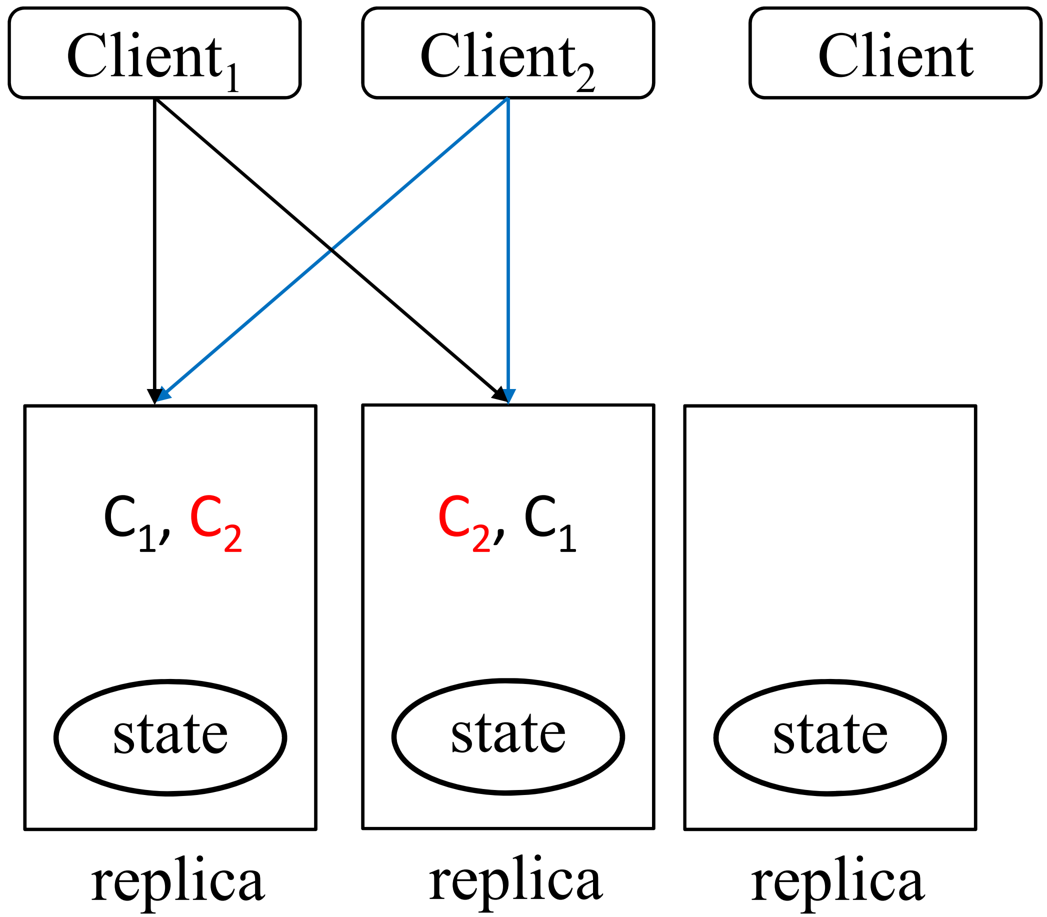
- Clients send requests to every replica



Replication Techniques

- **Active** replication

- Replicas may diverge



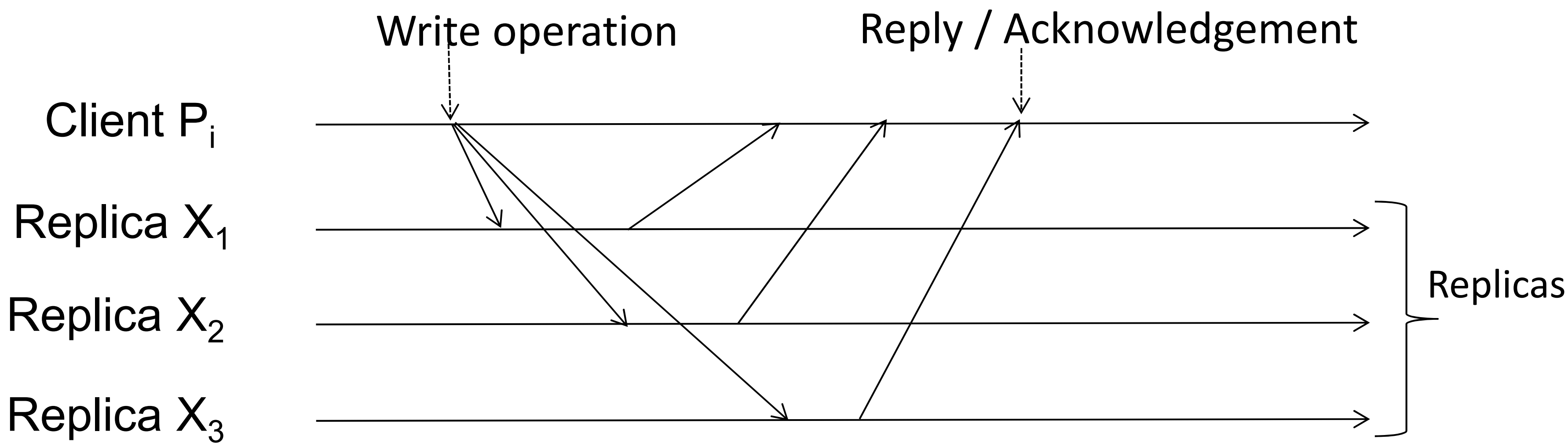
Active Replication

- Requires **total order broadcast** to guarantee each replica receives
  - **all** requests (from **all clients**)
  - **in the same order**

- Like client-server, “natural” to think about

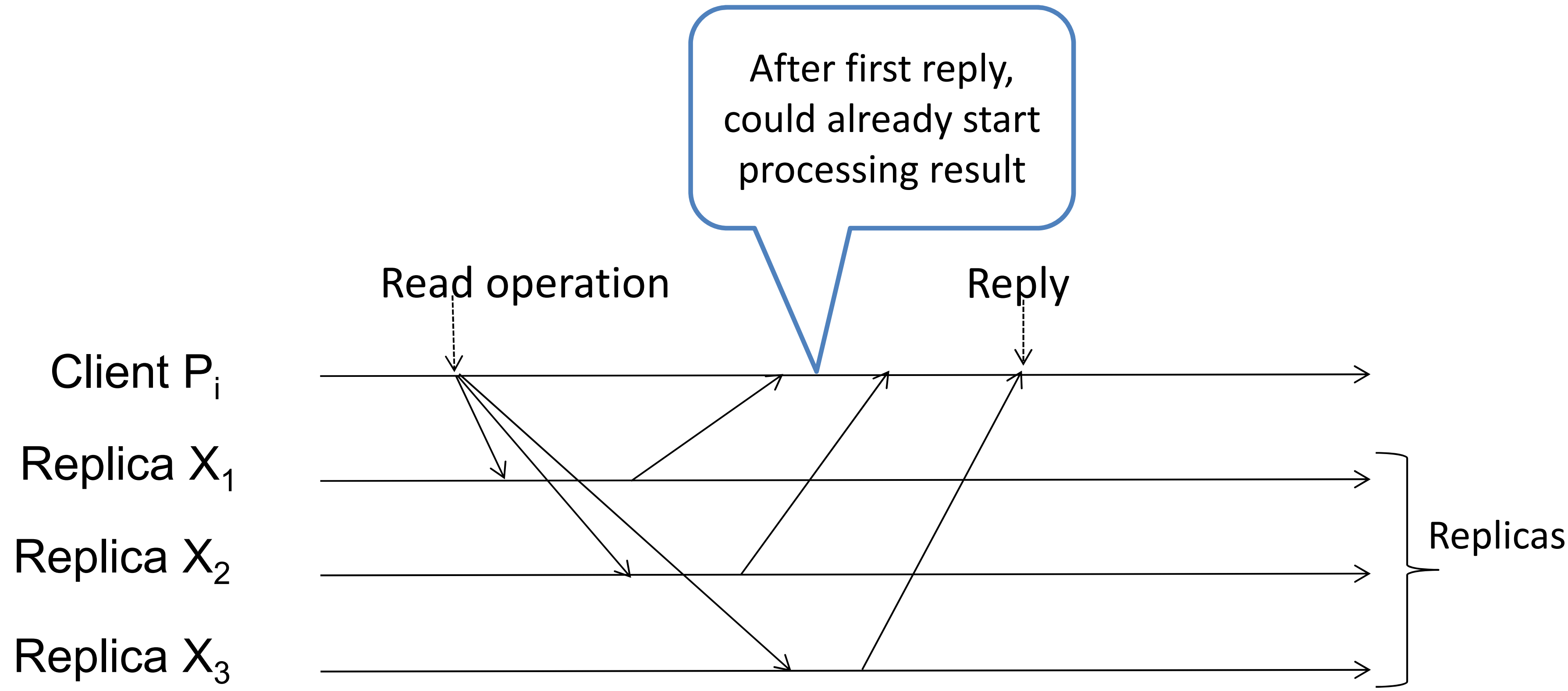
- Fast to get a response, first result received, unless Byzantine failure assumptions
  - Majority result

Active Replication



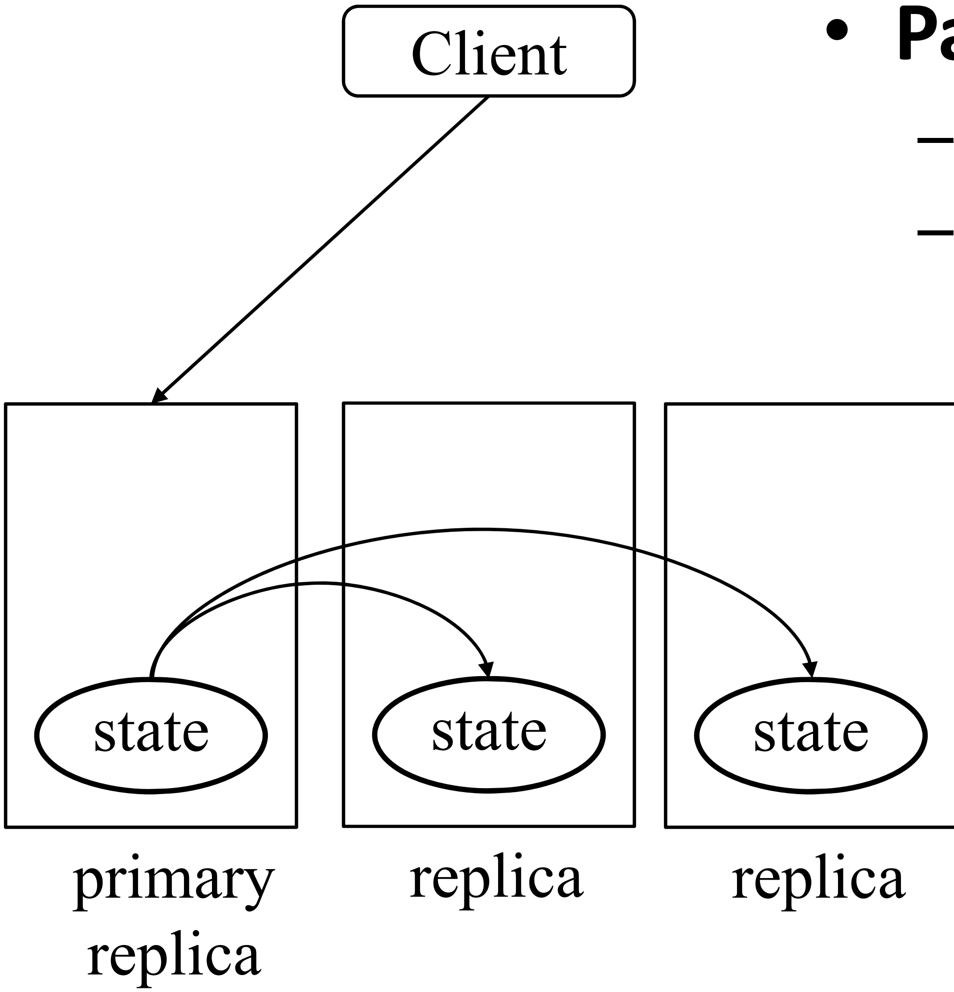
Configuration service:
- Failure detection
- Configuration management

Active Replication



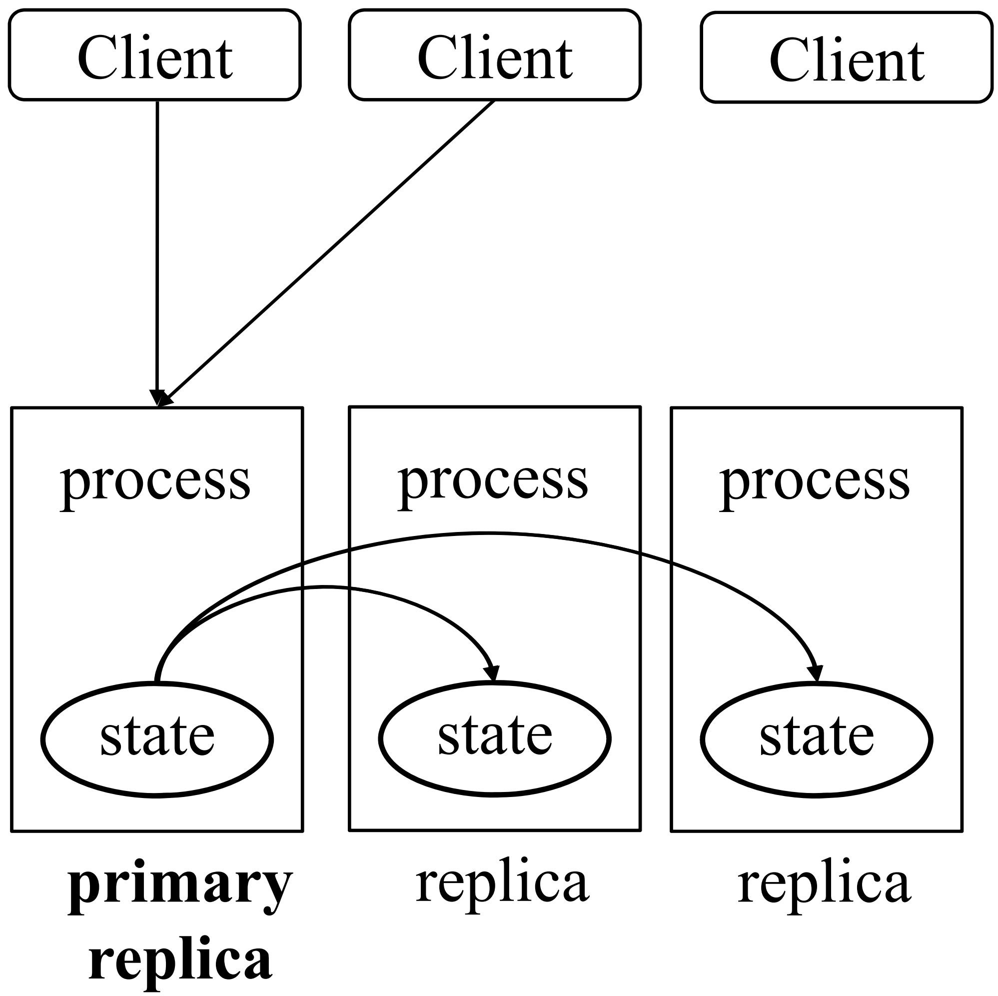
Alternative: Read from a quorum

Replication Techniques



- **Passive** replication
  - **Primary-backup**
  - Multi-primary

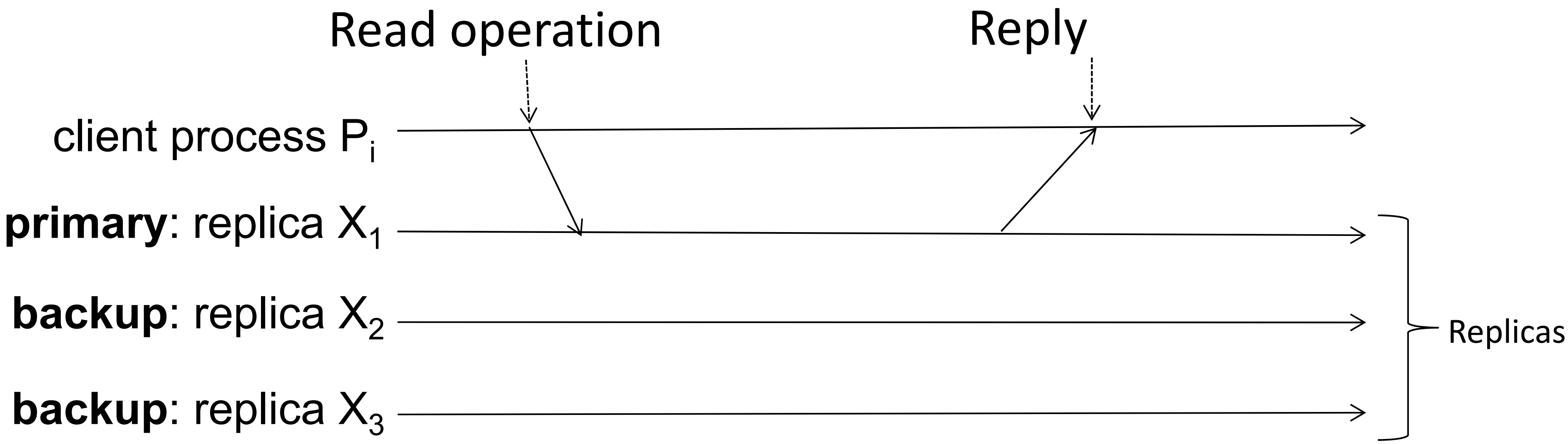
Replication Techniques



Passive Replication: Primary-Backup

- A replica is chosen to be the **primary** (leader election)
- **Primary**
  - Receives invocations from clients
  - Executes requests and sends back replies
  - Replicates the state to other replicas
- **Backup**
  - Interacts with primary only
  - Used to replace primary when it crashes (leader election)
- Called **eager replication** if replication is performed **within request boundary** (e.g., before the reply is sent)

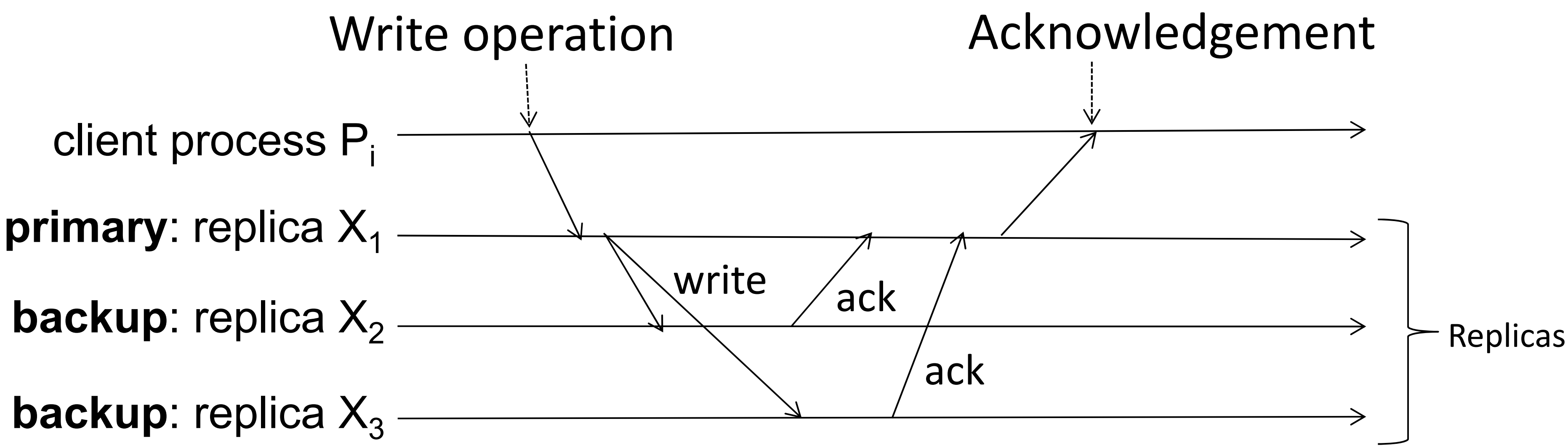
Primary-Backup Scenario



Configuration service:
- Failure detection
- Configuration management

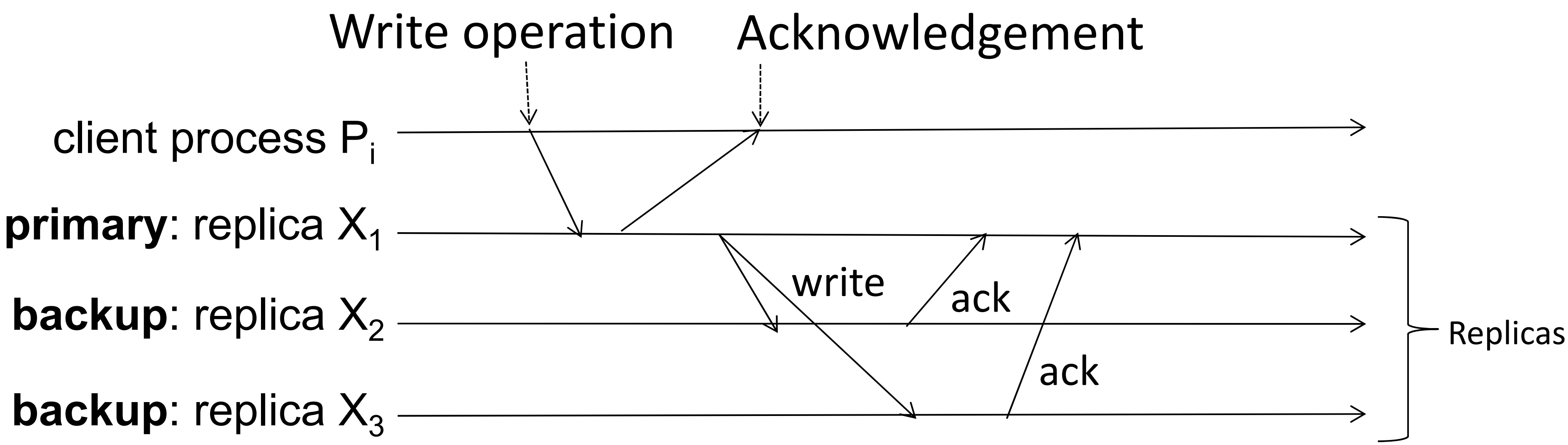
Eager replication

Primary-Backup Scenario



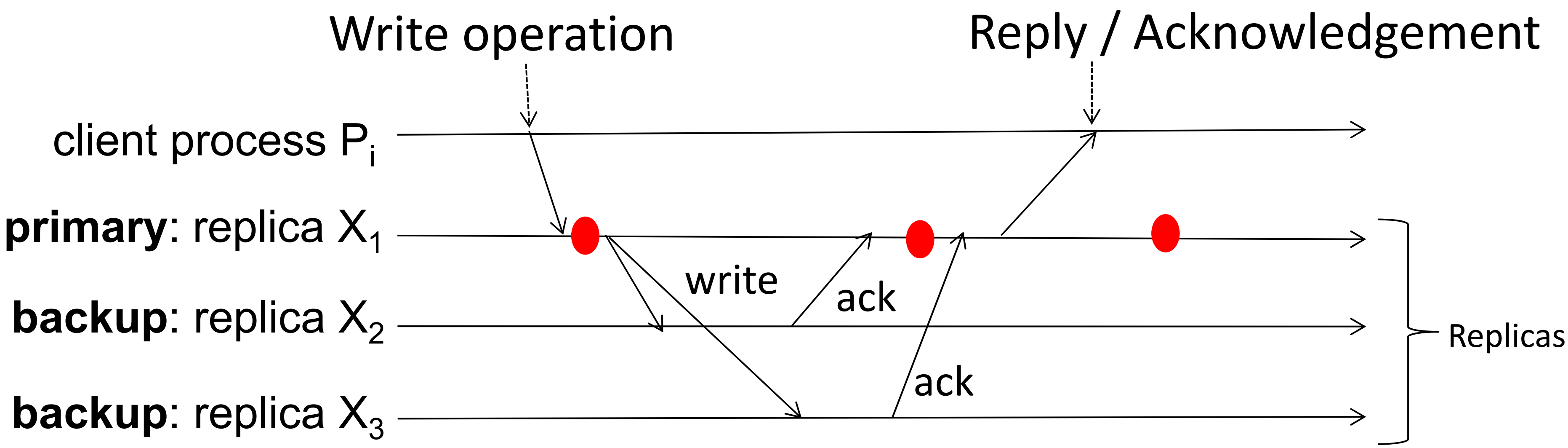
Lazy replication

Primary-Backup Scenario



- Writes are propagated asynchronously after primary acknowledges update to client
- Replicas may diverge
- Requires additional mechanism to deal with primary crashing

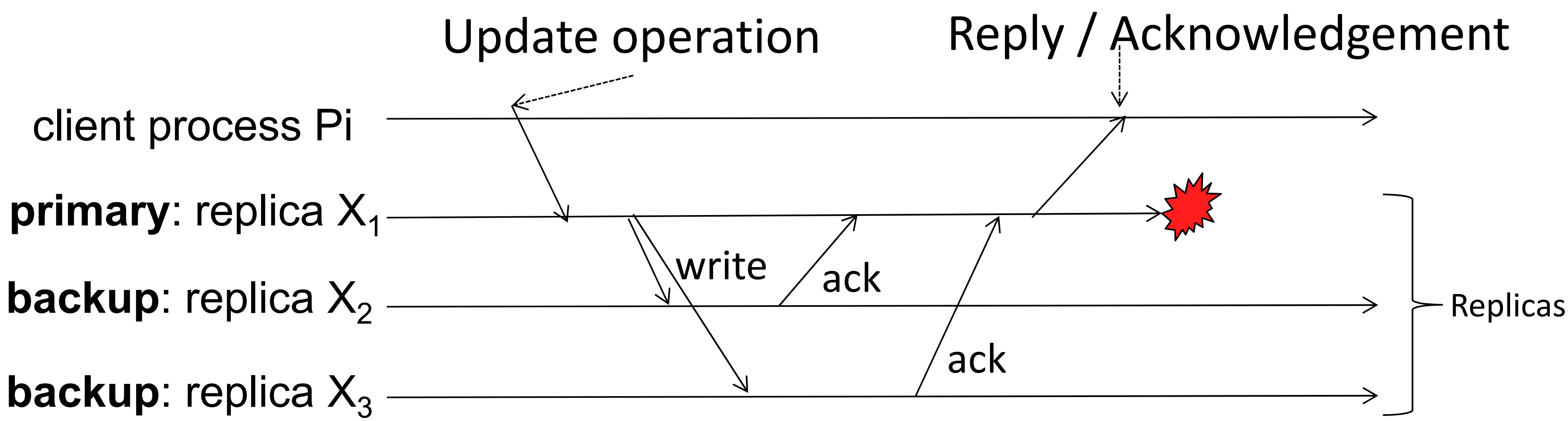
Primary-Backup: Presence of Failures



- In all cases, a new **primary is elected** from among the backups

Scenario 1

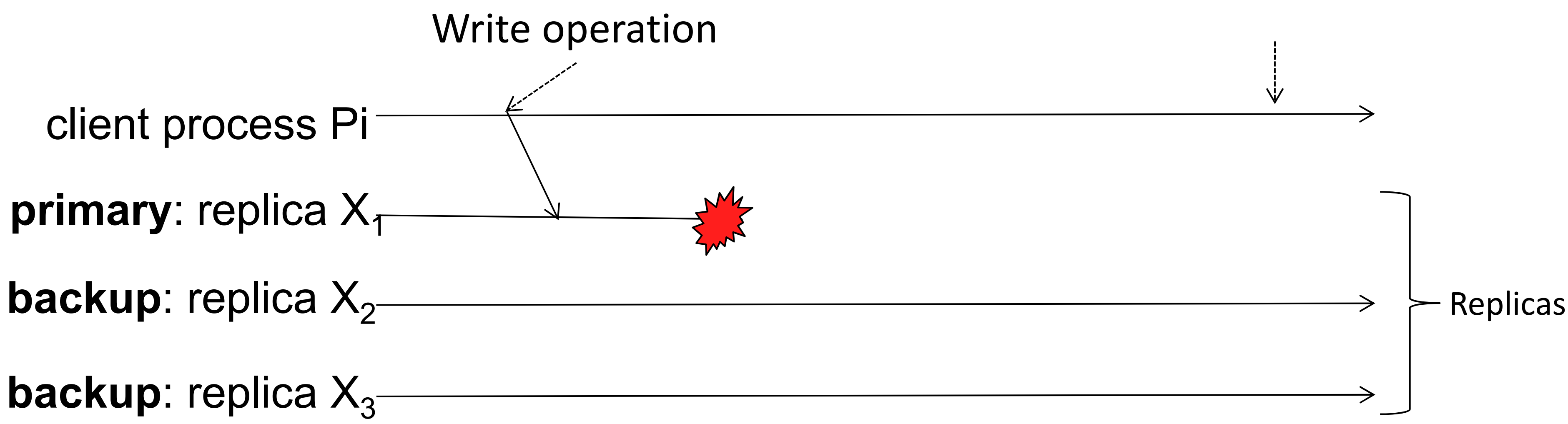
Primary fails after client receives reply



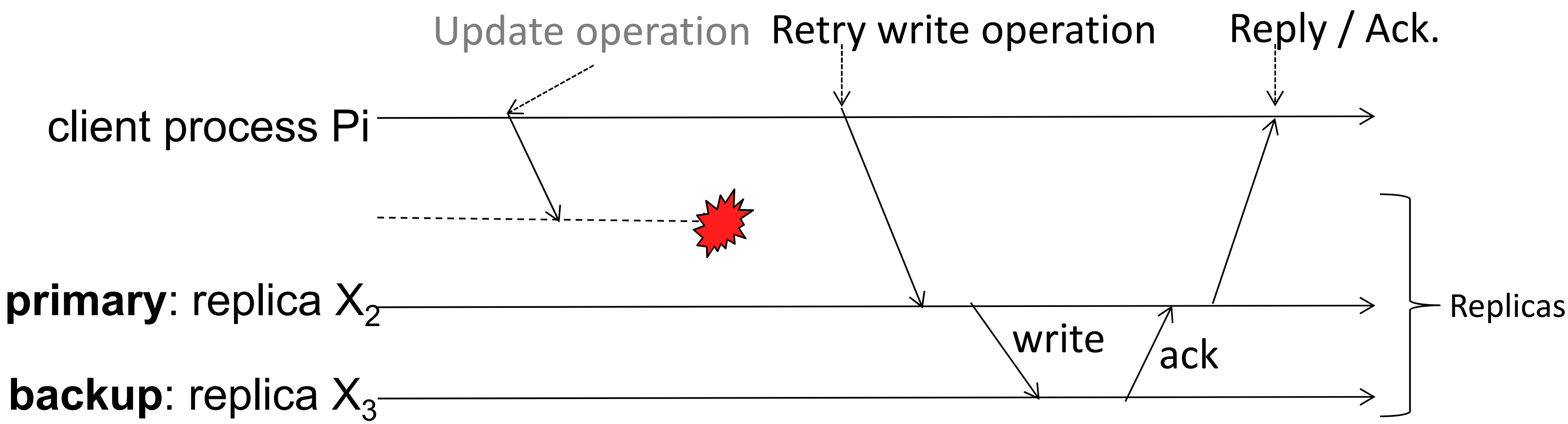
- New primary is elected

Scenario 2

Primary fails before propagating updates



Scenario 2



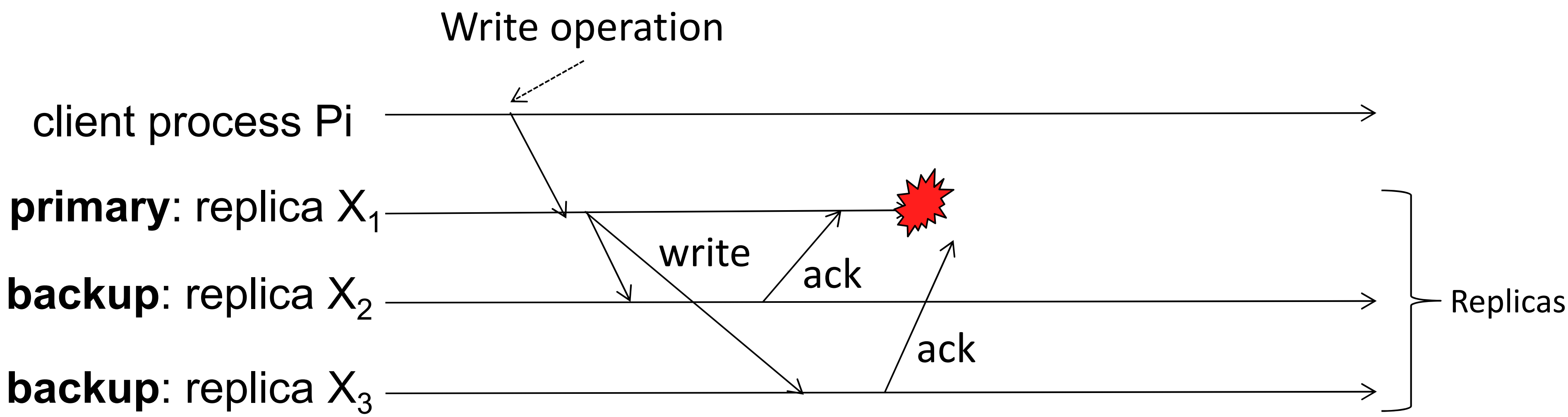
Configuration service:

- Failure detection
- Leader election
- Configuration management

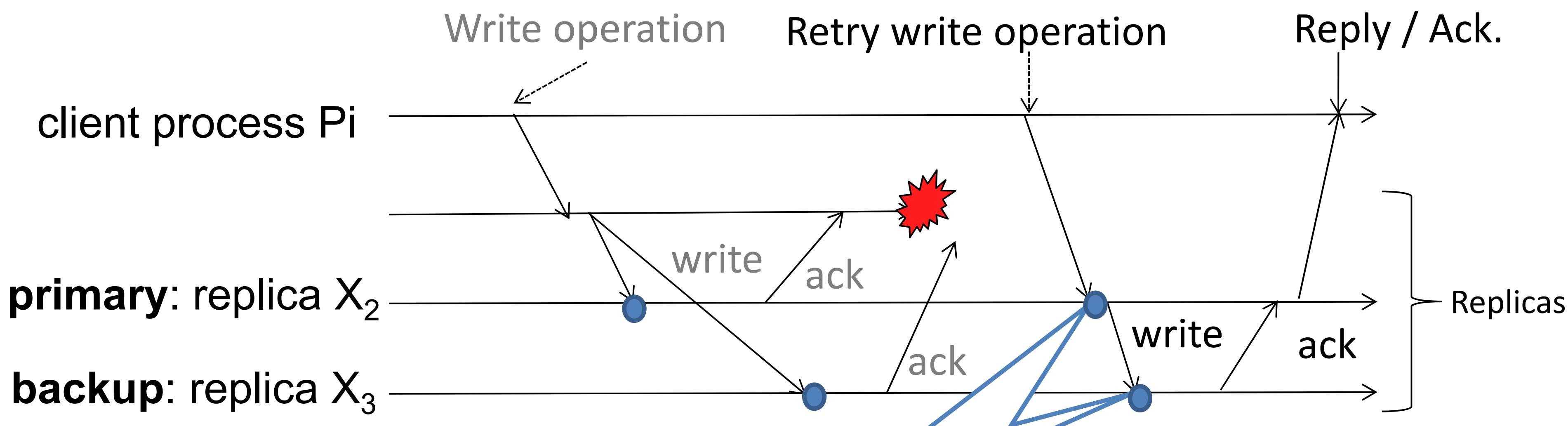
- Timeout mechanism at client triggers retry
- Retry could fail, if against old primary
- Check configuration service for new leader, retry

Scenario 3

Primary fails before receiving all write acknowledgements



Scenario 3

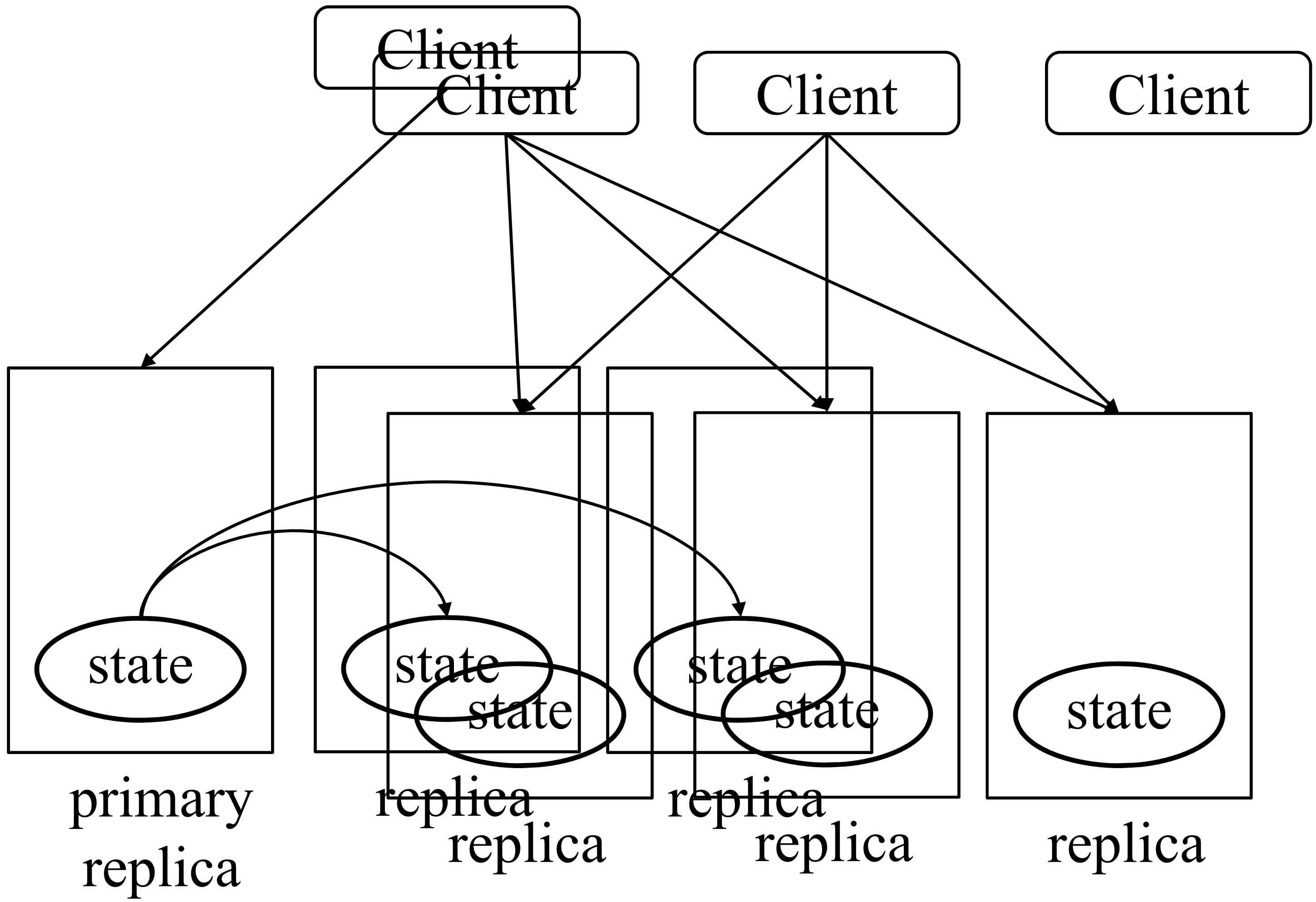


Configuration service:
- Failure detection
- Leader election
- Configuration

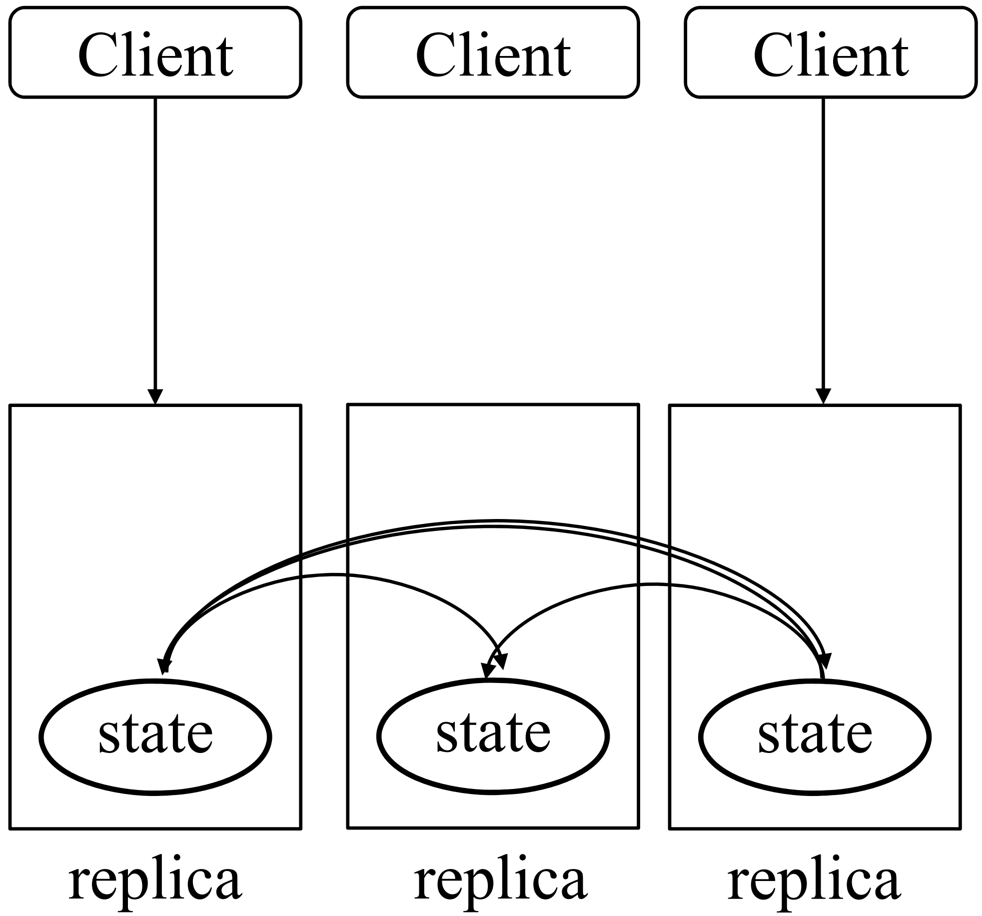
Use WAL at replicas to differentiate between Scenario 2 and 3 (**avoid updating twice**)

- Timeout and retry write
- Persistent log at replicas
- Configuration service

Multi-primary Replication (MPR)



Multi-primary Replication (MPR)



Multi-primary Replication (MPR)

- Primary-backup is **not scalable**, since only a single process handles client requests
  - Inefficient use of replica resources
- Multi-primary solves issue by allowing every replica to handle client requests
  - Replicas have to figure out how to order requests (e.g., **using consensus**)
- If replication is **eager**, processes have to **agree on order of operations** before they execute any command and respond to clients
  - Can be slow since it locks processes

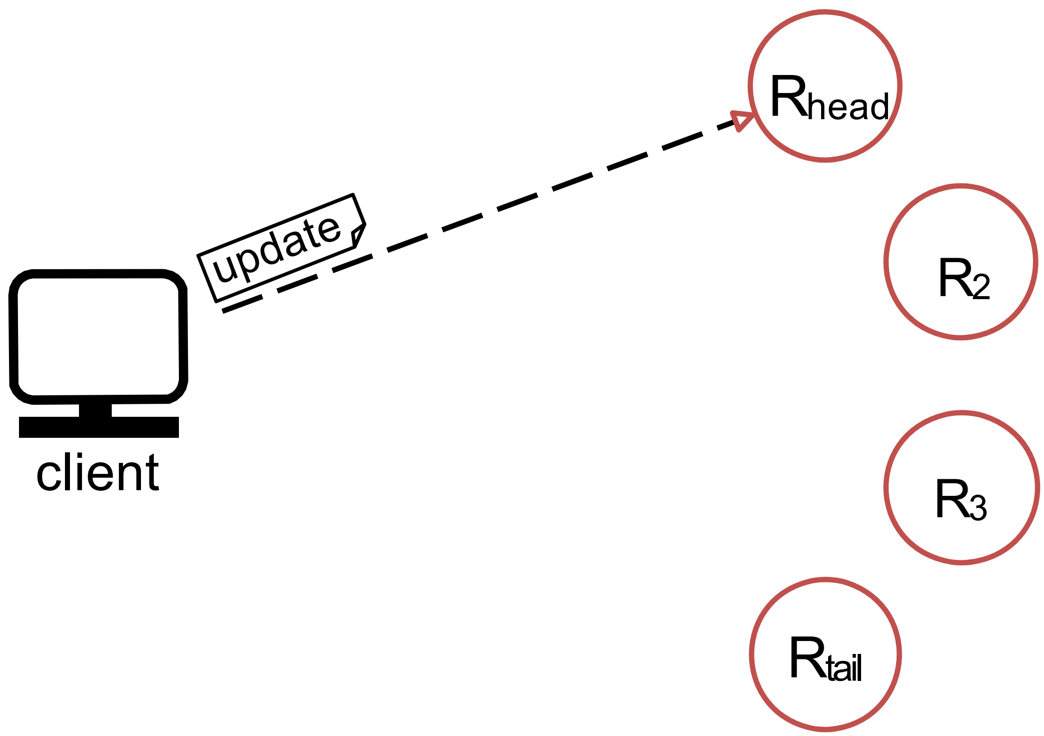
Optimistic Lazy MPR

- To improve response times, **replication** is often **done lazily**
  - Replica first executes locally and returns a response to client right away
  - Replicas asynchronously propagate updates they made

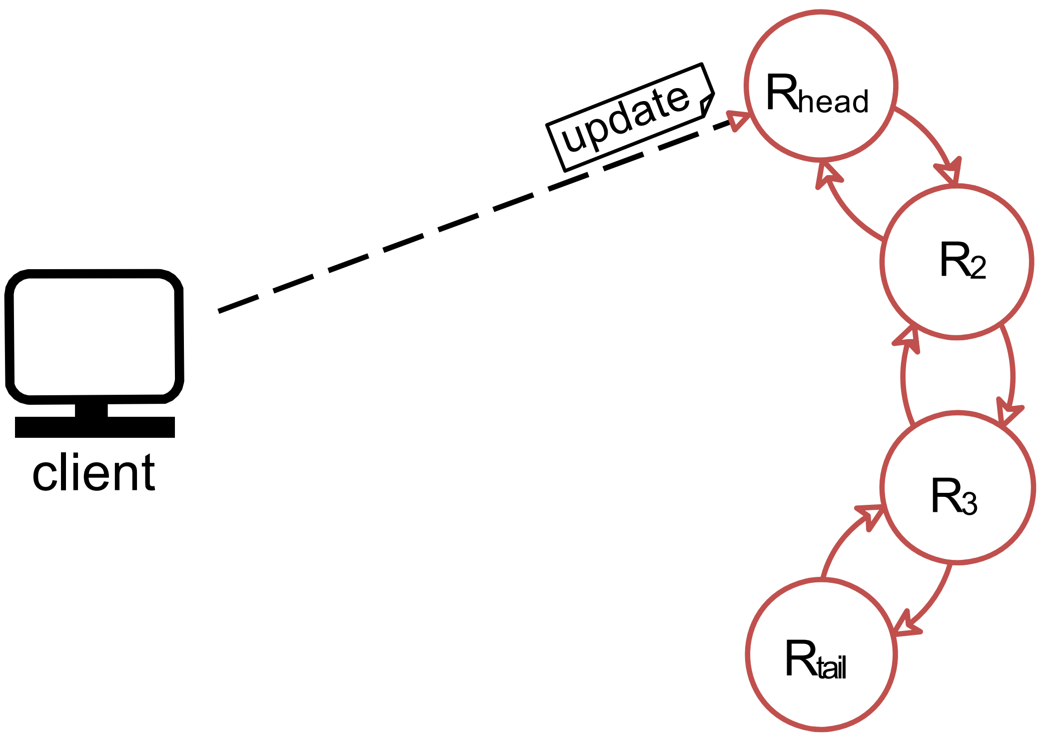
- Also called **optimistic replication**
  - Replicas may diverge, which can introduce inconsistencies, aborts, and rollbacks

CHAIN REPLICATION

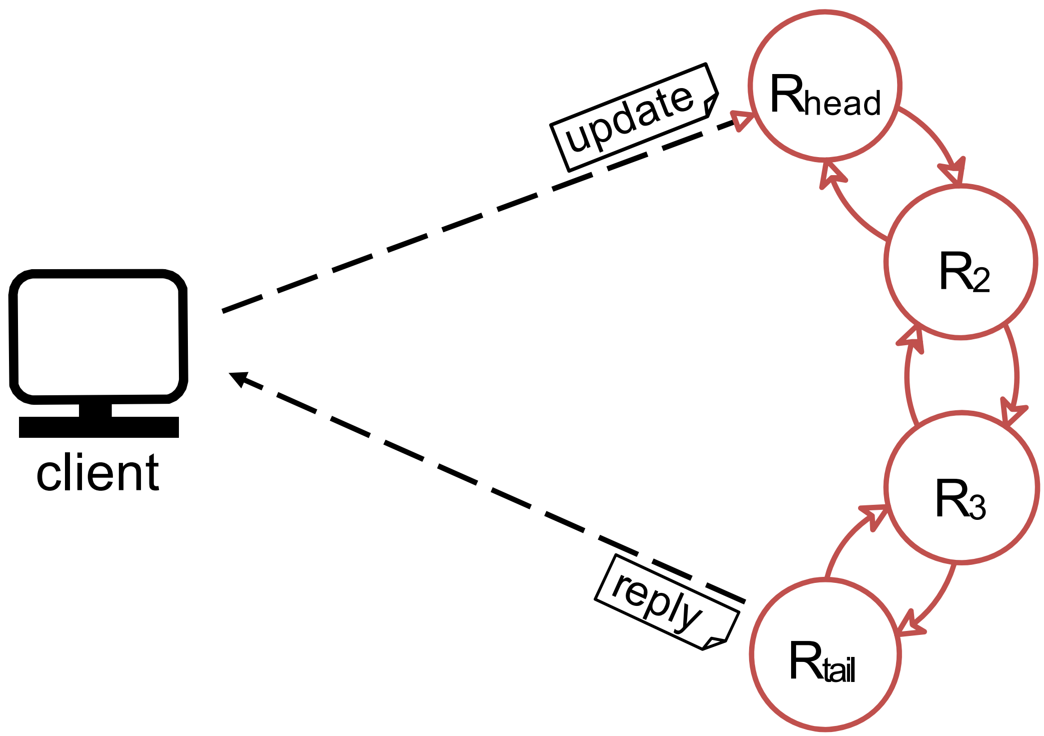
Chain Replication



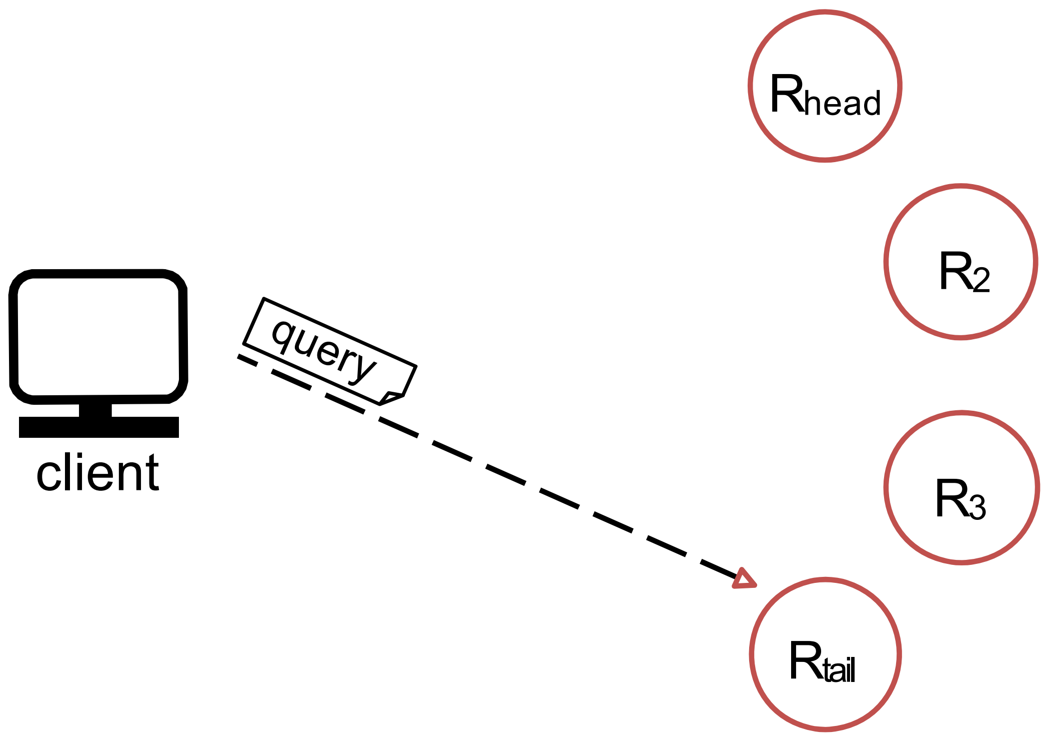
Chain Replication



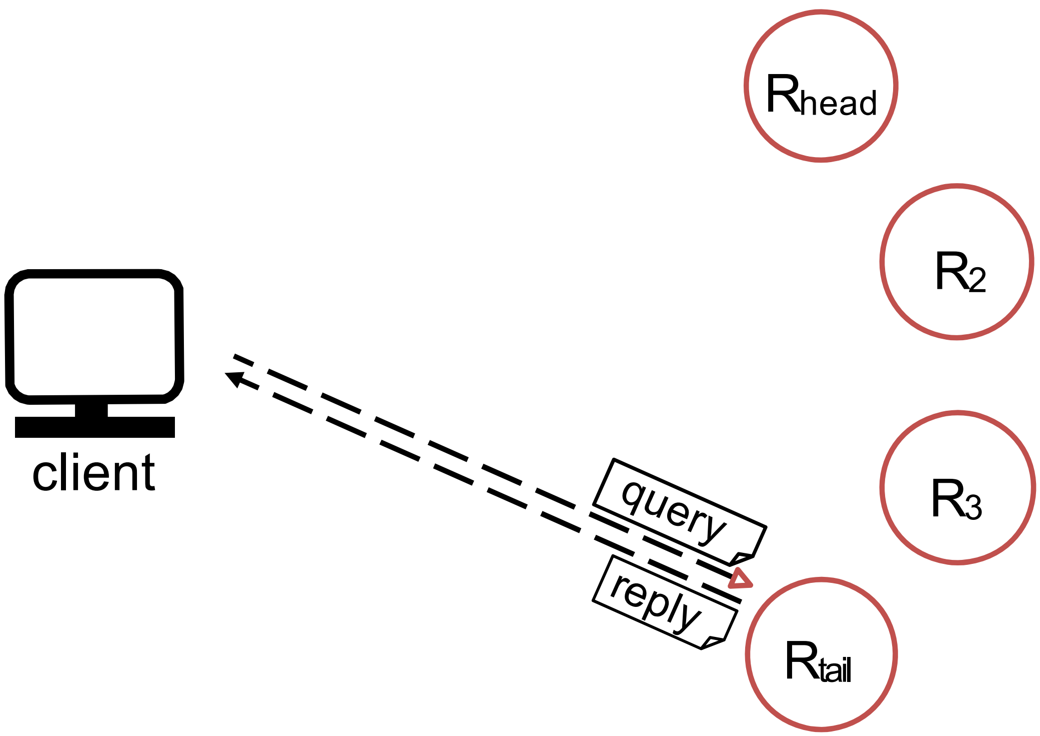
Chain Replication



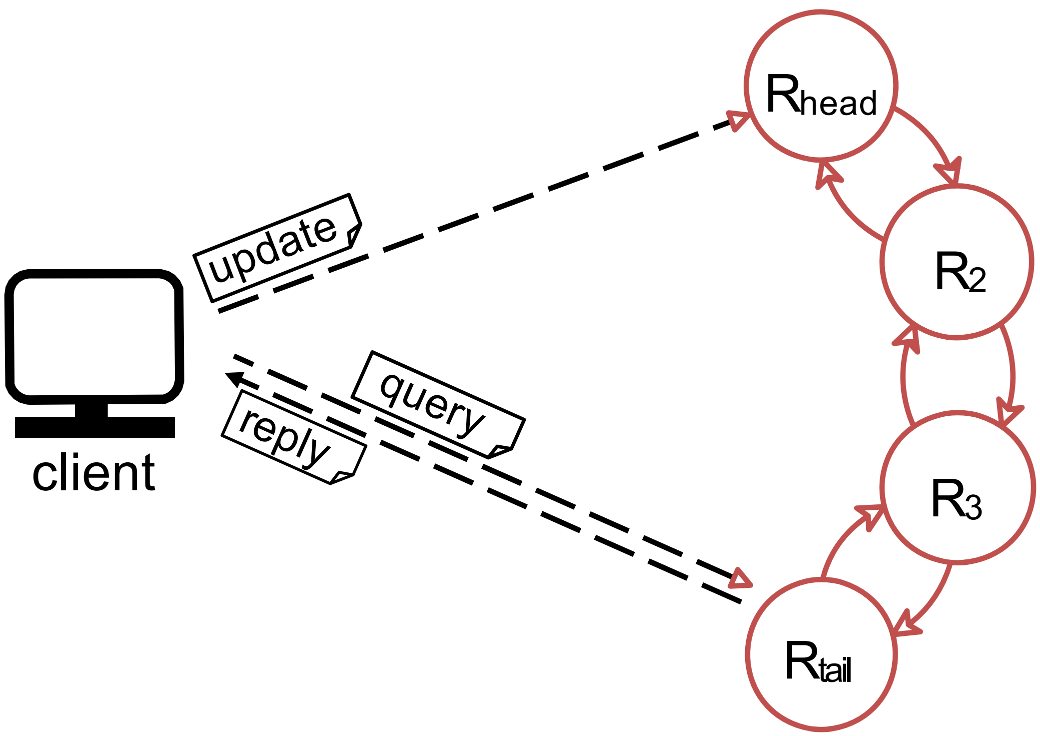
Chain Replication



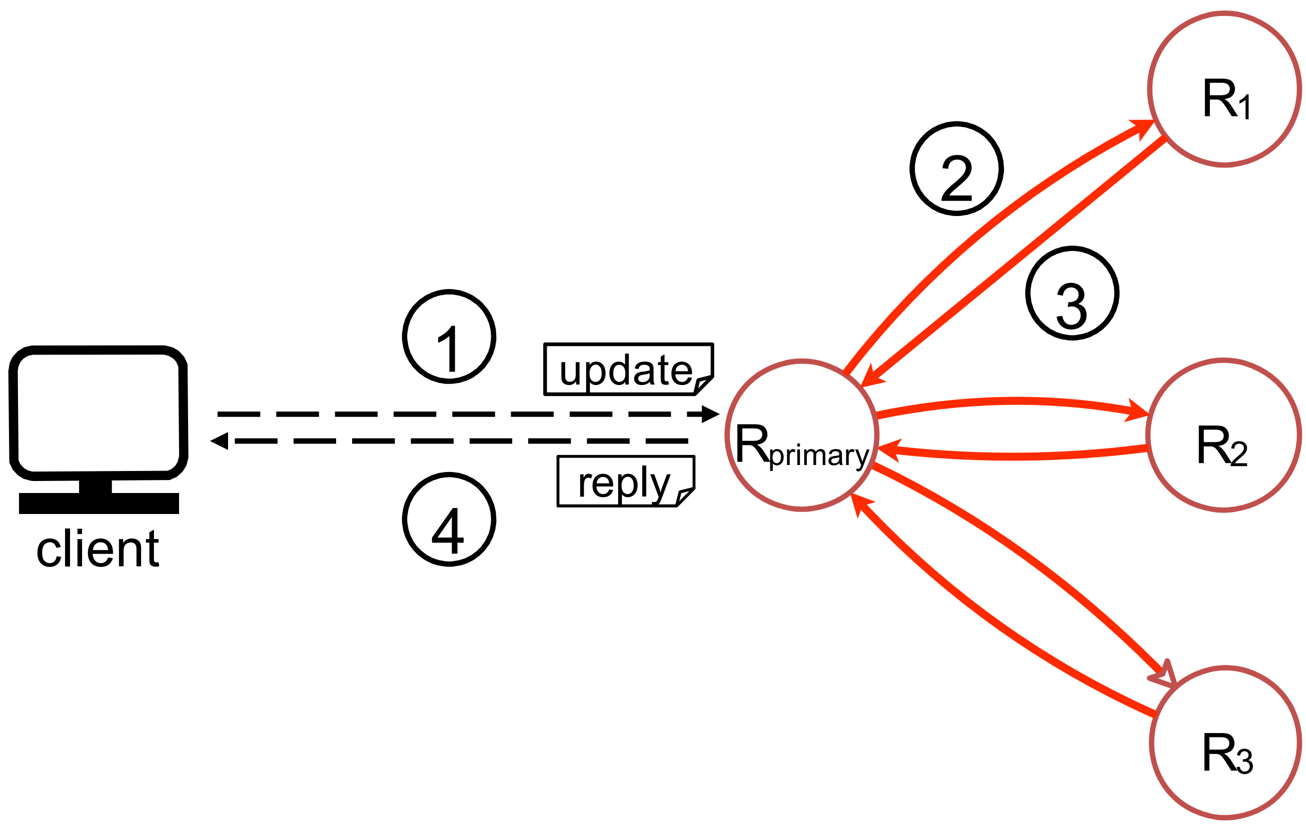
Chain Replication



Chain Replication

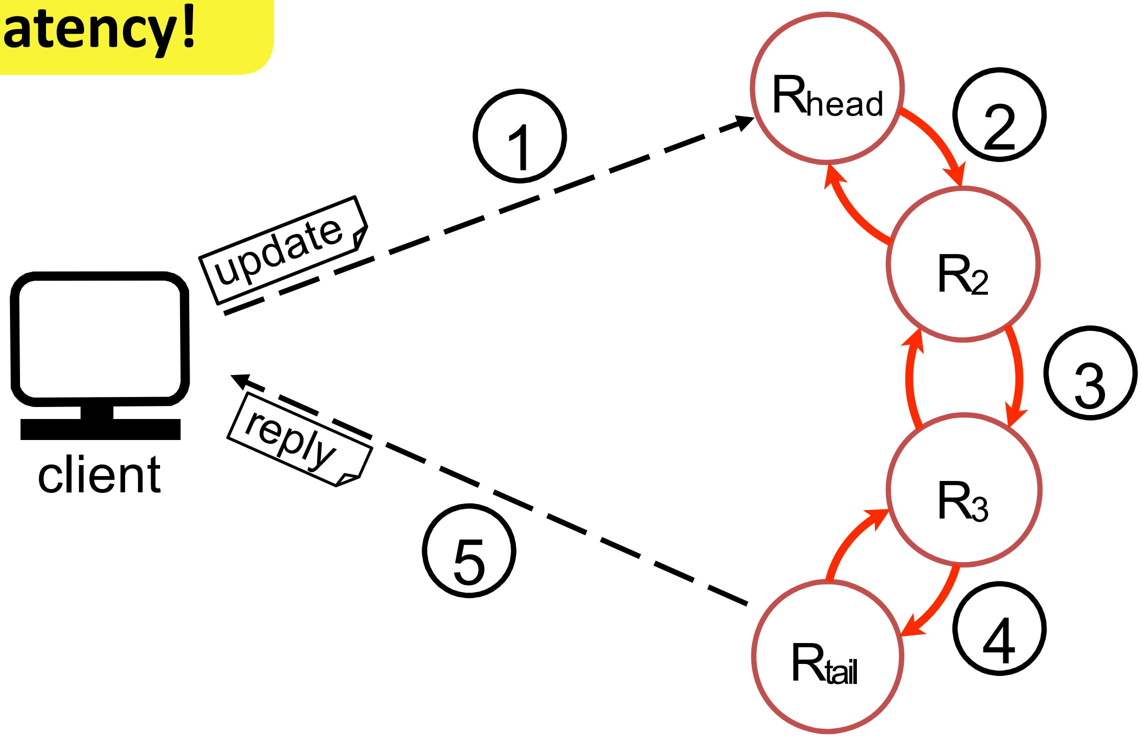


Primary-Backup Replication

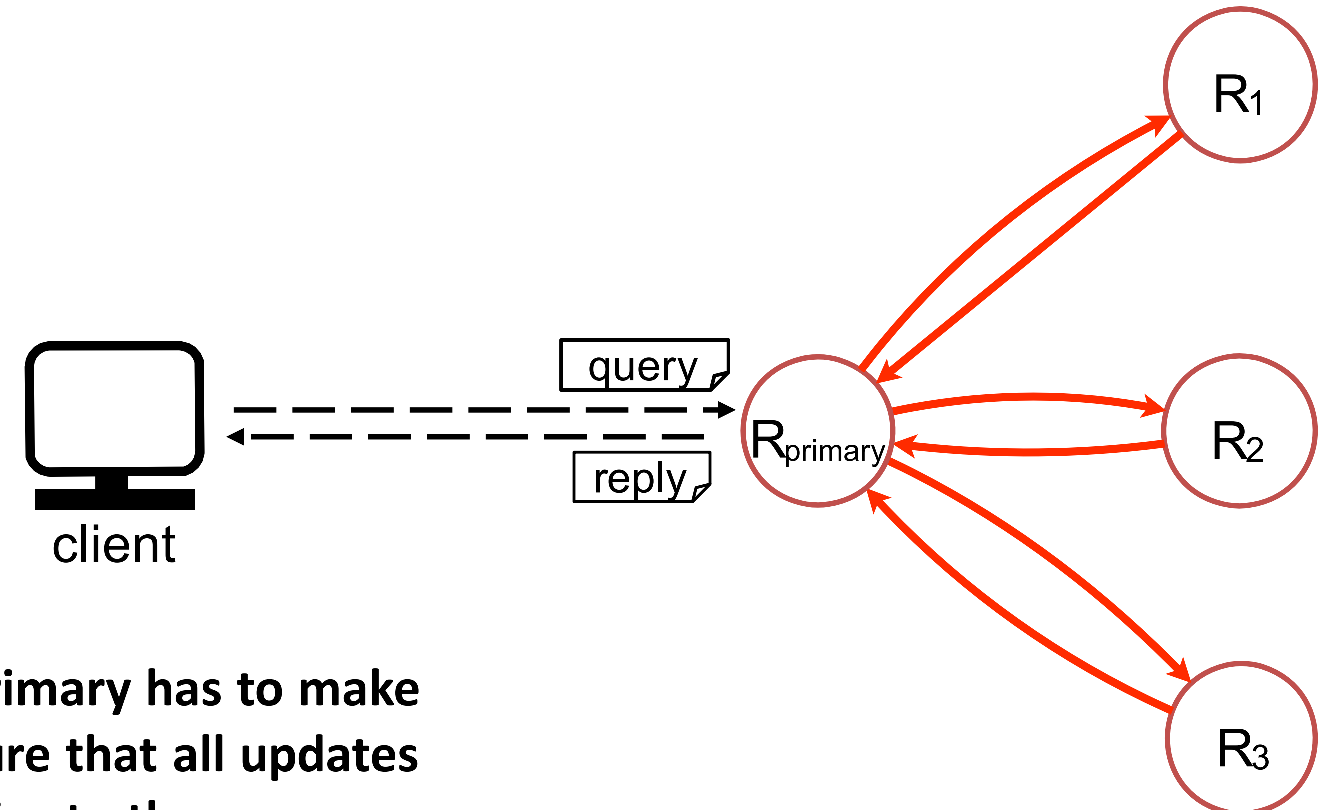


Chain Replication

Higher latency!

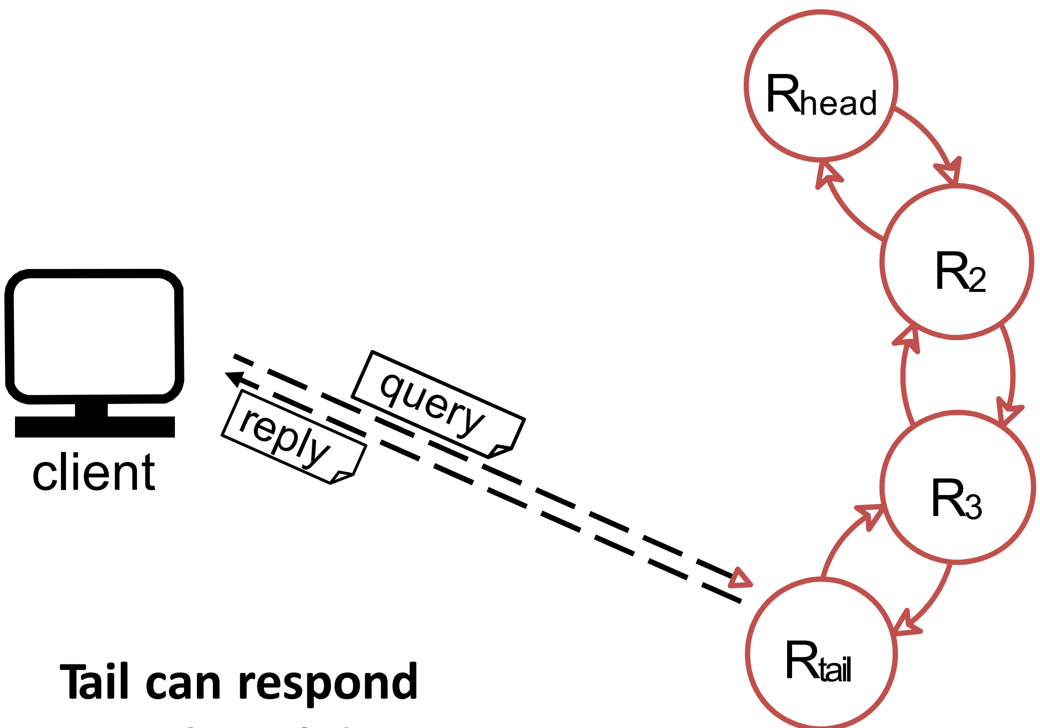


Primary-Backup Replication



Primary has to make sure that all updates prior to the query are completed!

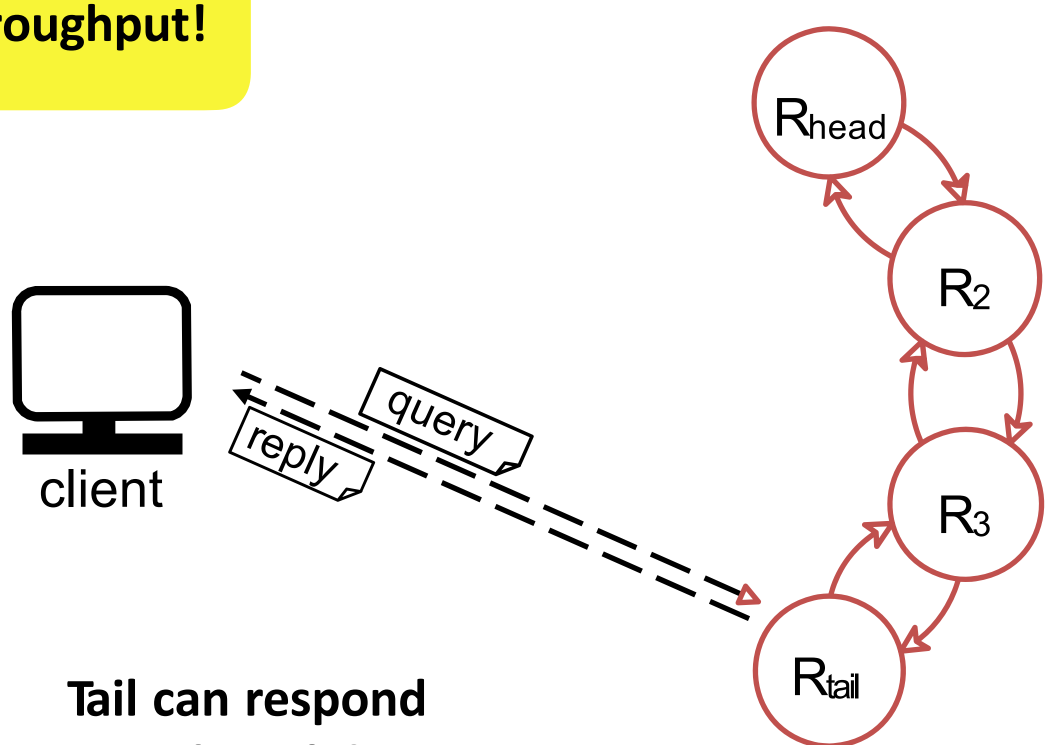
Chain Replication



Tail can respond directly!

Chain Replication

Higher throughput!



Tail can respond directly!

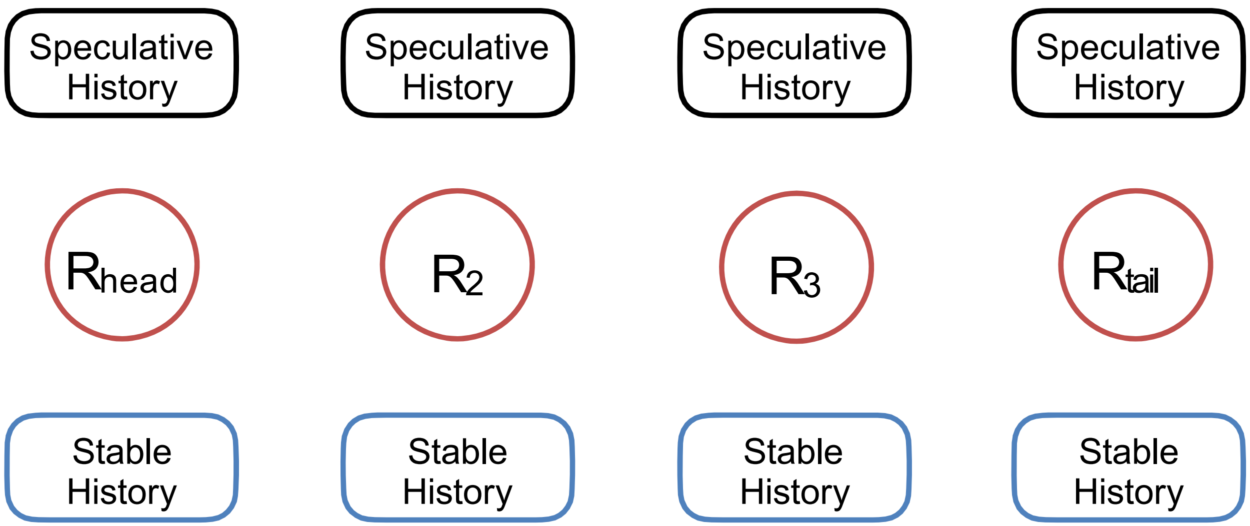
Failures in Chain Replication: $f + 1$

- Need $f + 1$ nodes to tolerate f failures

Chain Replication: Operations

- Chain replication operations:
  - Updates
  - Queries
  - Failures
  - Reconfigurations

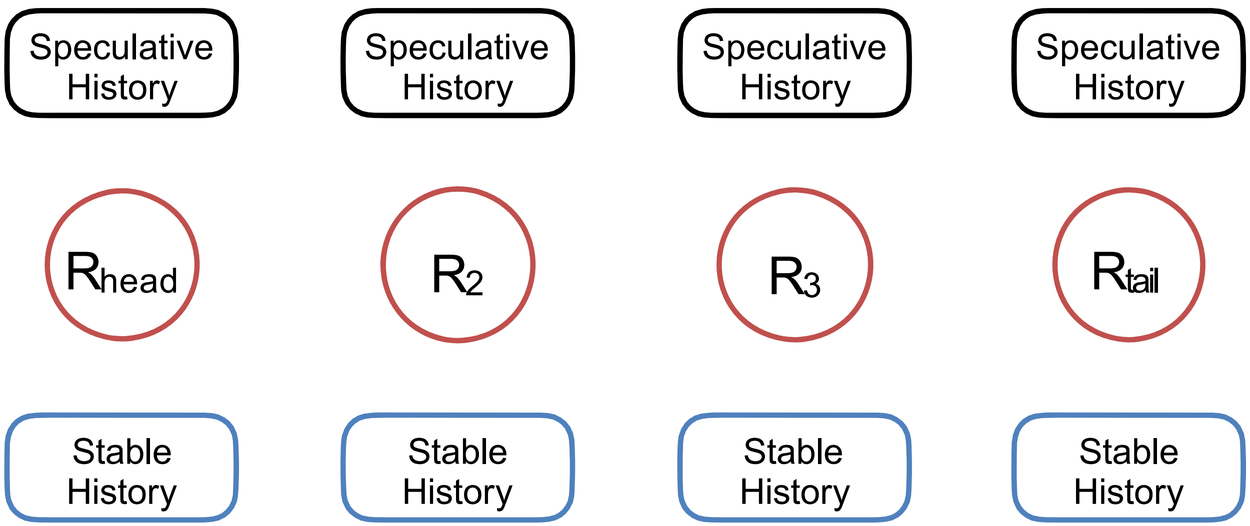
Updates



Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

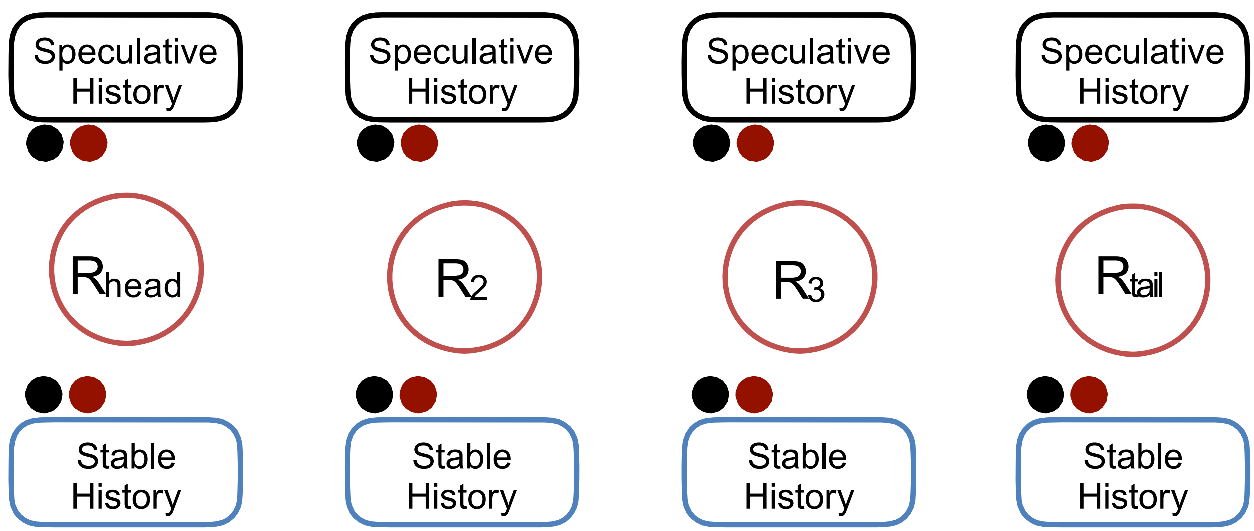
Updates

R_2 is the **predecessor** of R_3

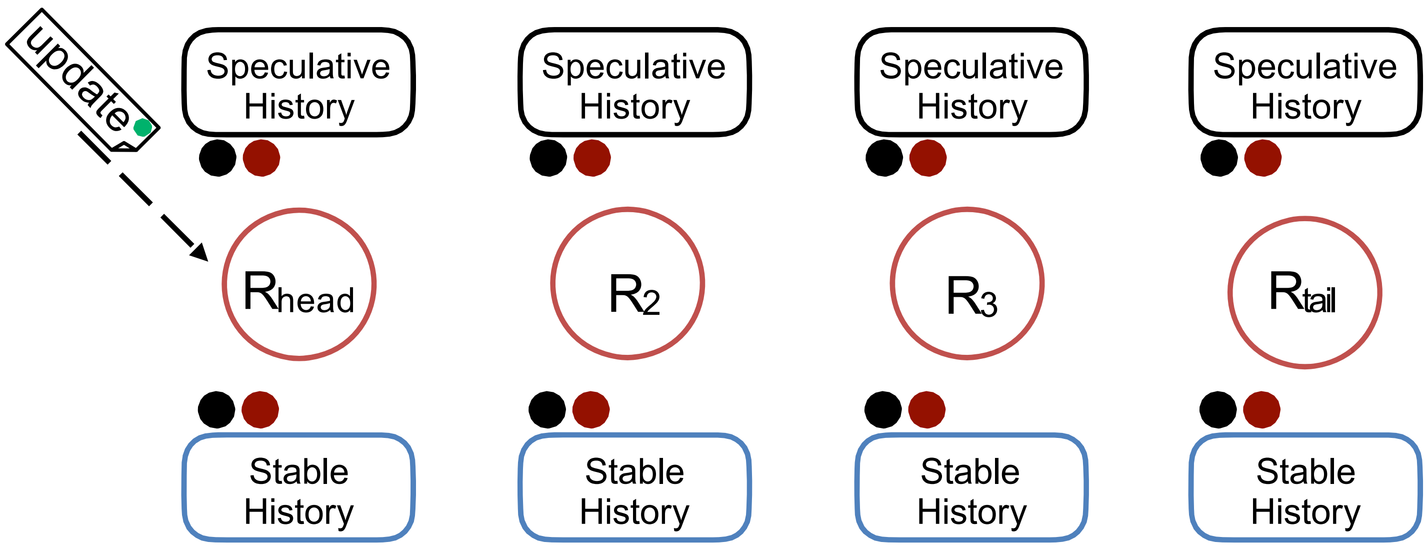


R_3 is the **successor** of R_2

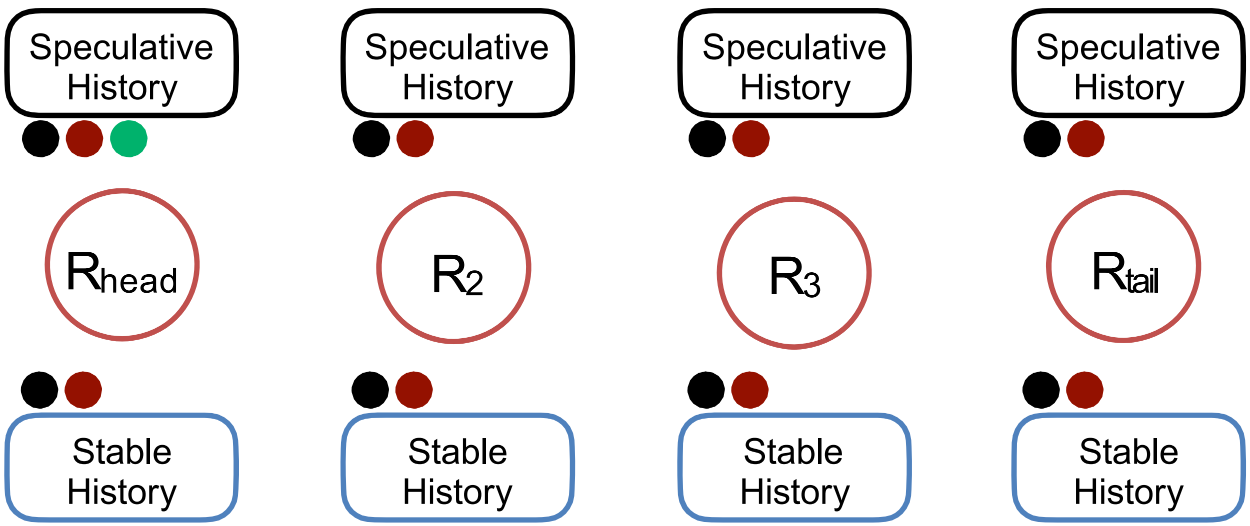
Updates


Speculative History
Speculative History
Speculative History
Speculative History
R_{head}
R_2
R_3
R_{tail}
Stable History
Stable History
Stable History
Stable History

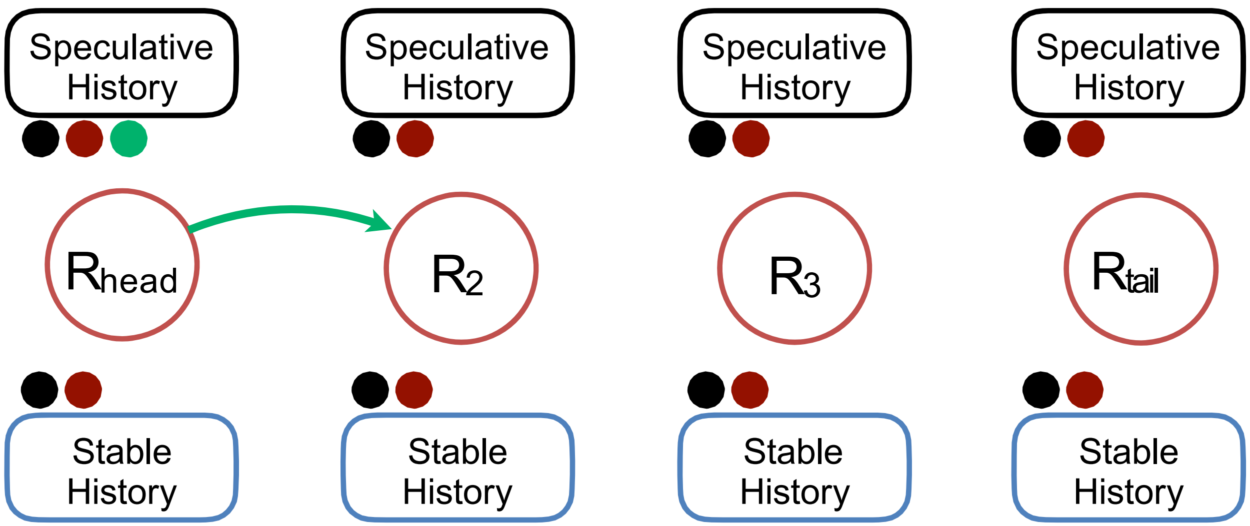
Updates


update
Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

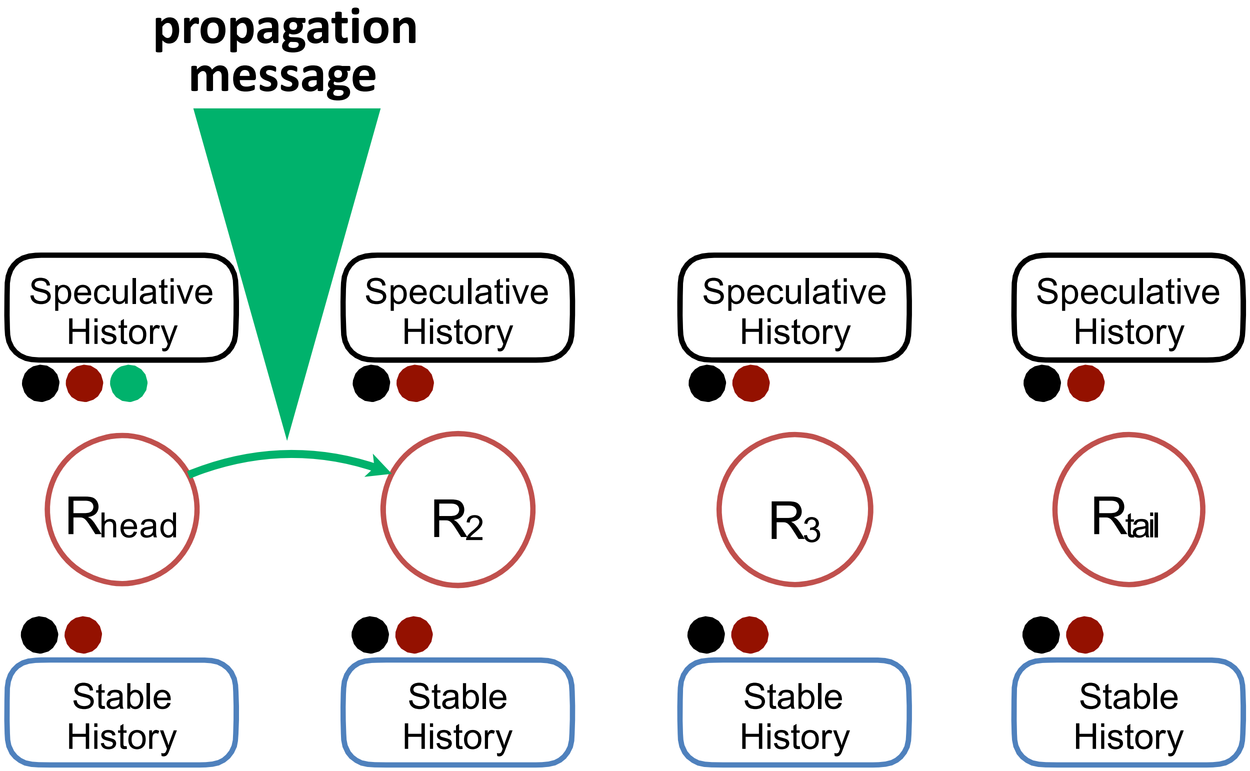
Updates


Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

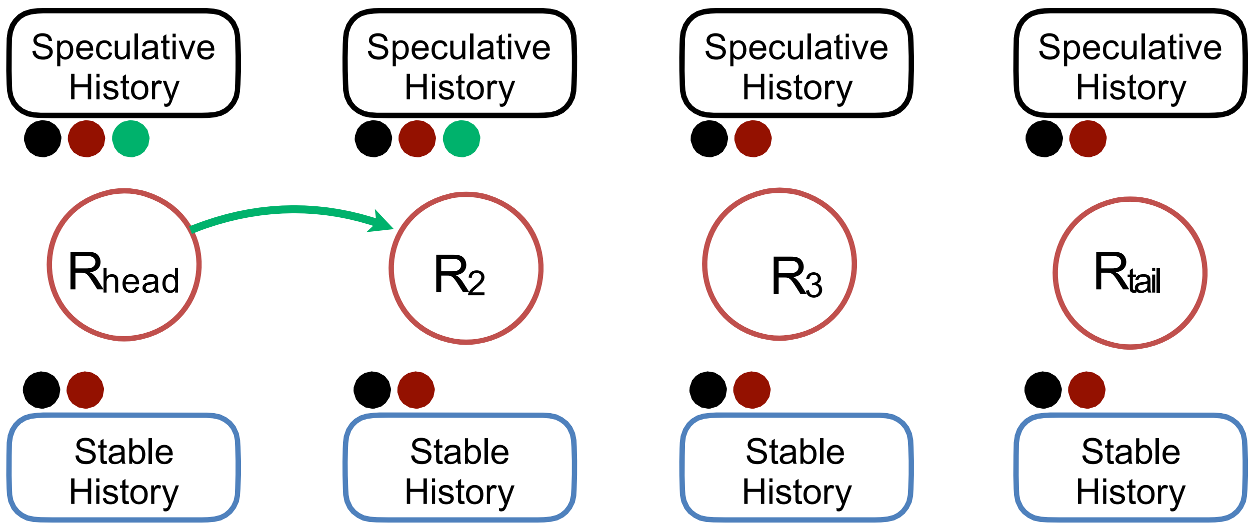
Updates


Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

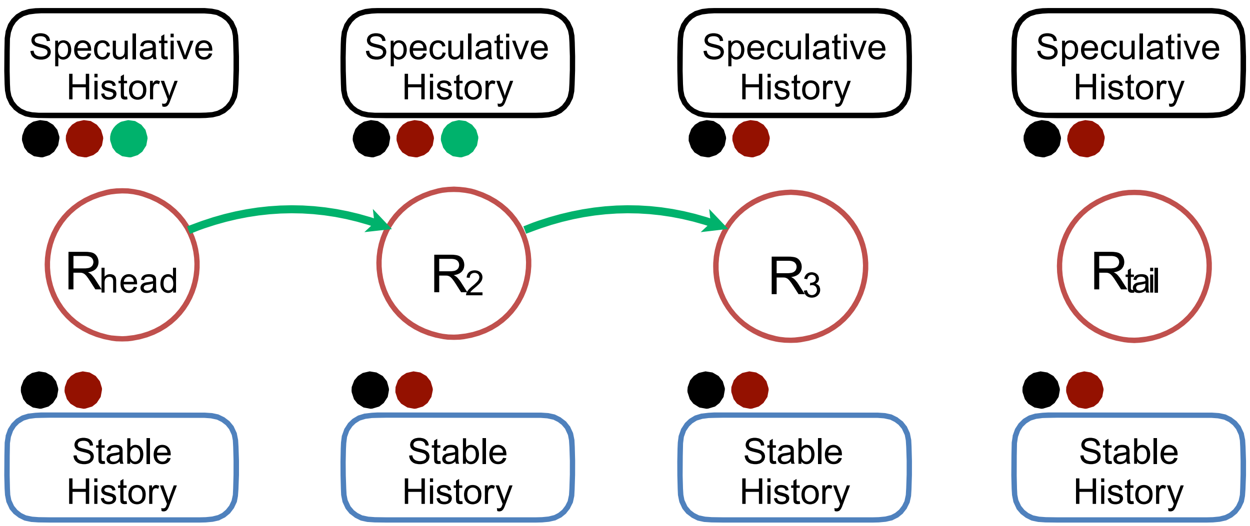
Updates


propagation
message
Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Updates

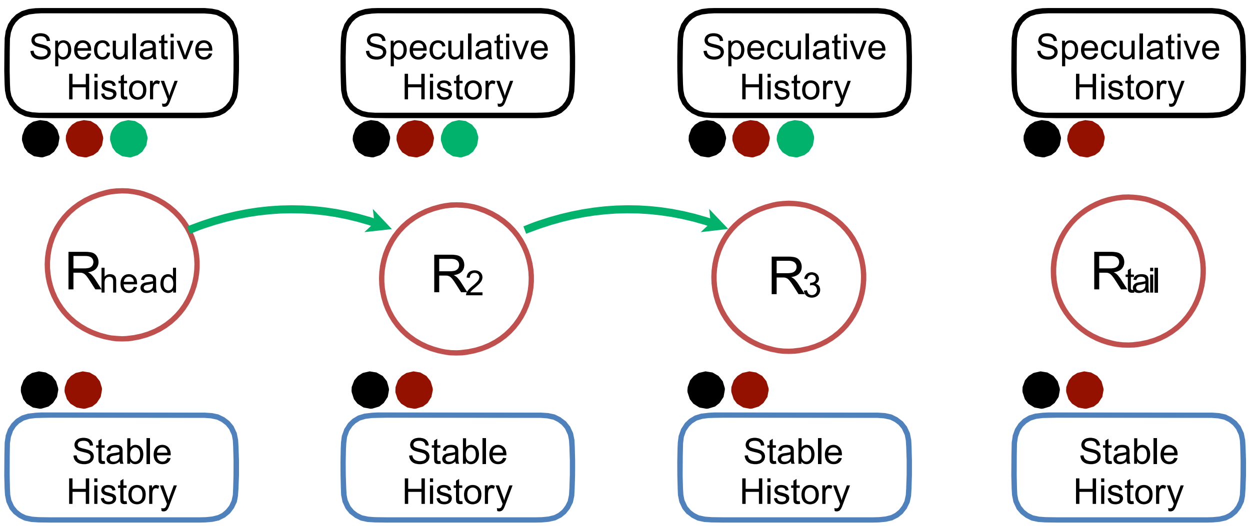

Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Updates

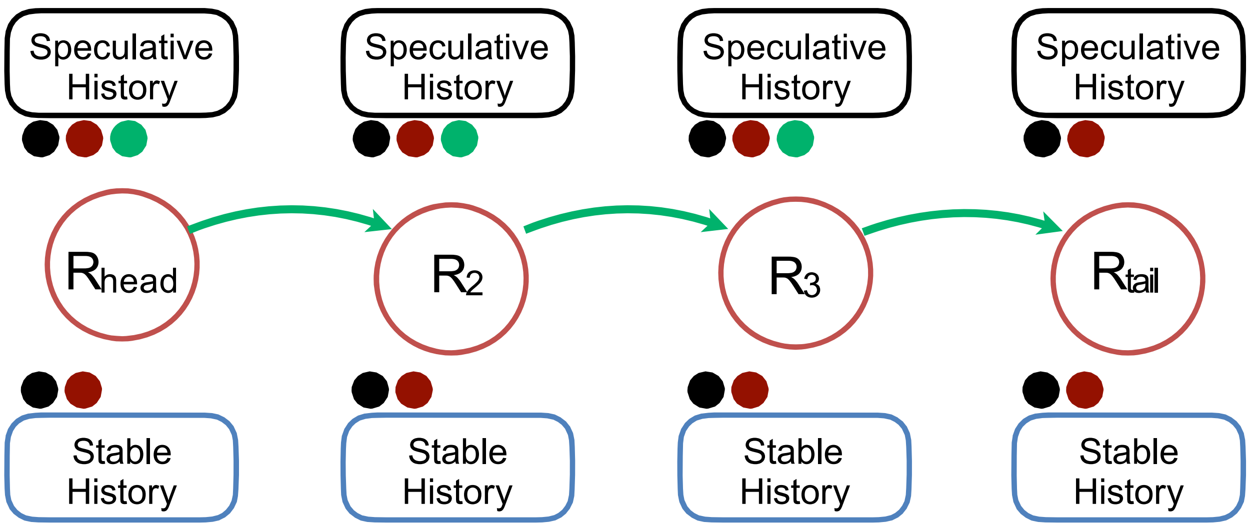


Speculative History
Speculative History
Speculative History
Speculative History
R_{head}
R_2
R_3
R_{tail}
Stable History
Stable History
Stable History
Stable History

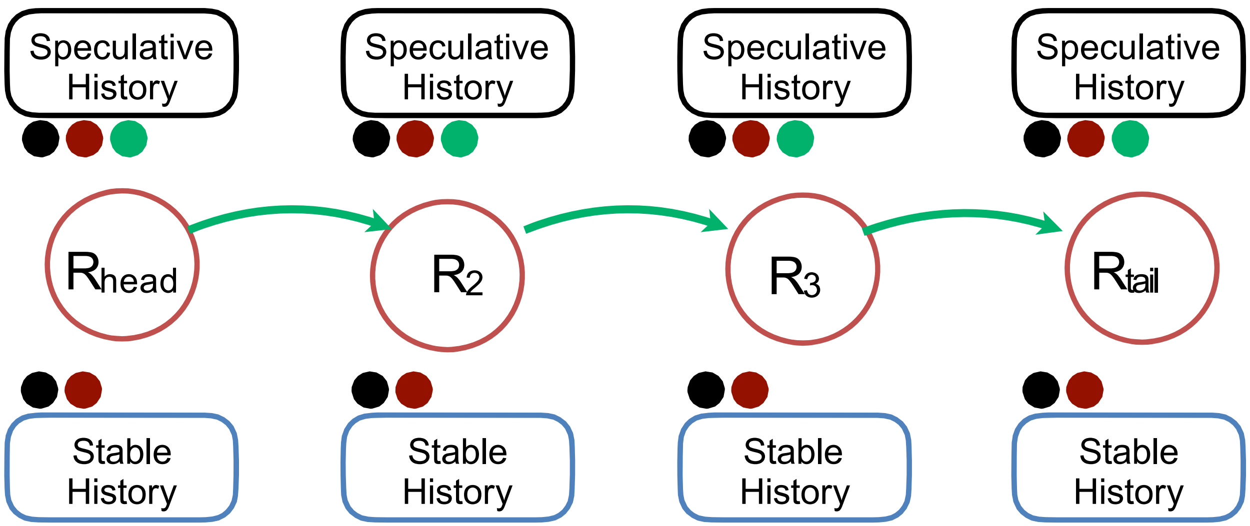
Updates


Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Updates

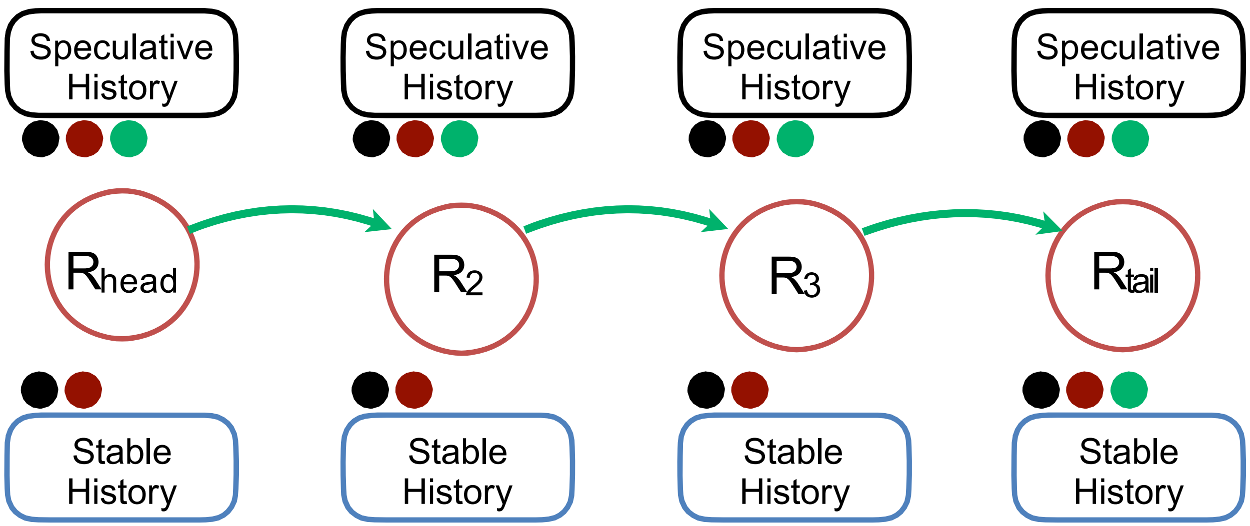

Speculative History
Speculative History
Speculative History
Speculative History
R_{head}
R_2
R_3
R_{tail}
Stable History
Stable History
Stable History
Stable History

Updates



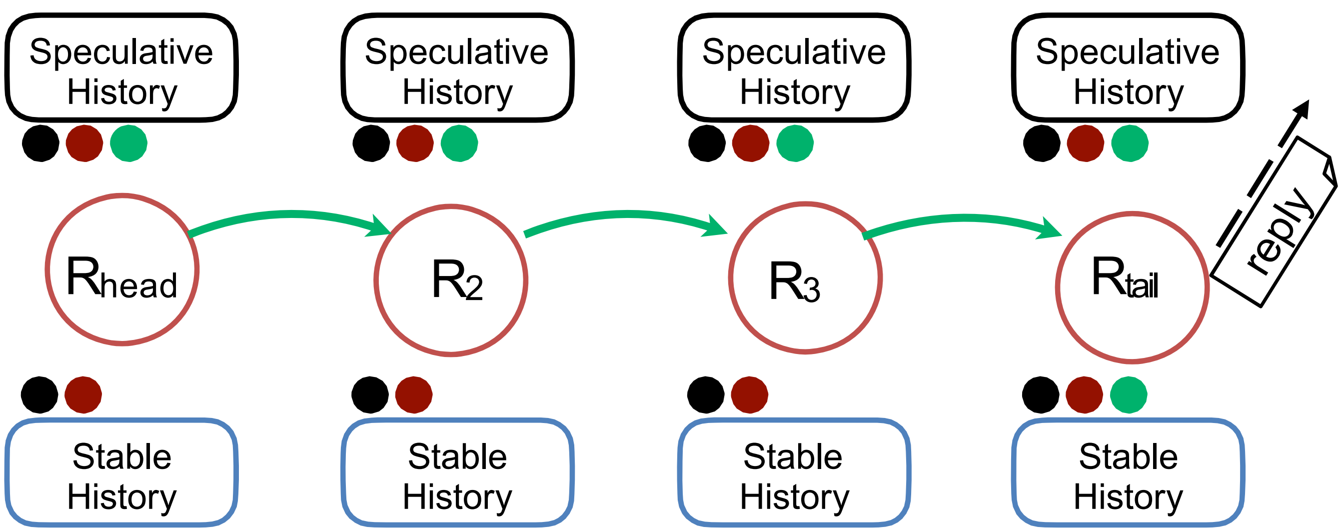
Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Updates



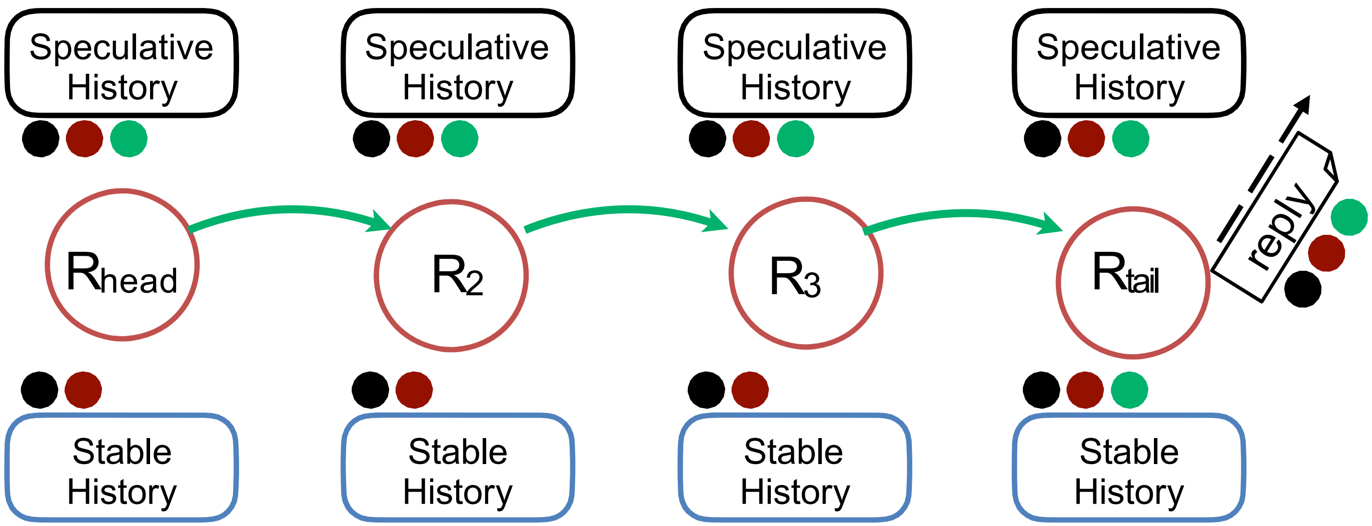
Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Updates



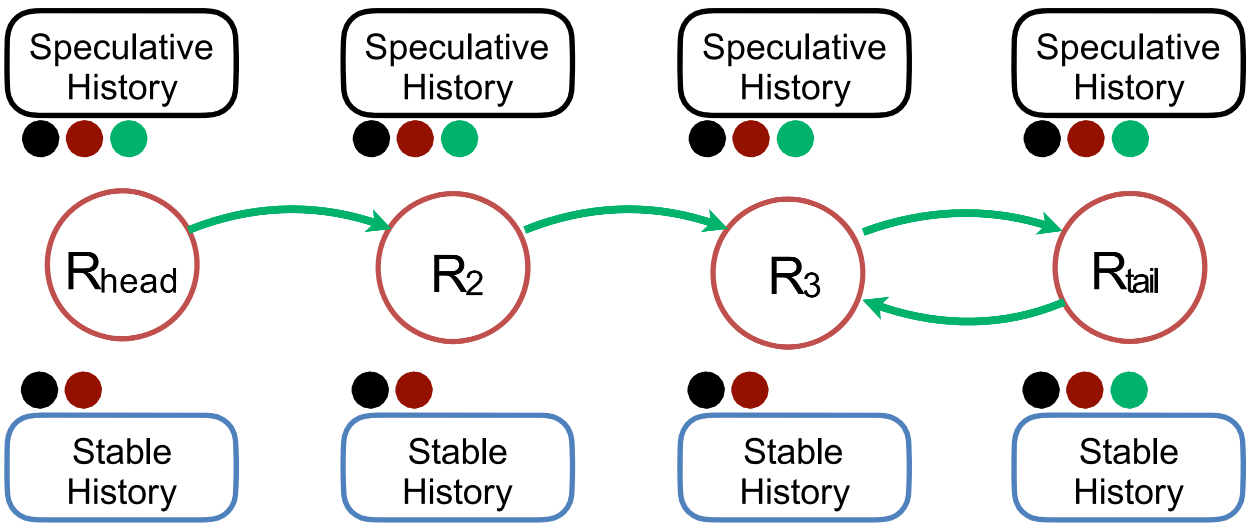
Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
reply
Stable History
Stable History
Stable History
Stable History

Updates

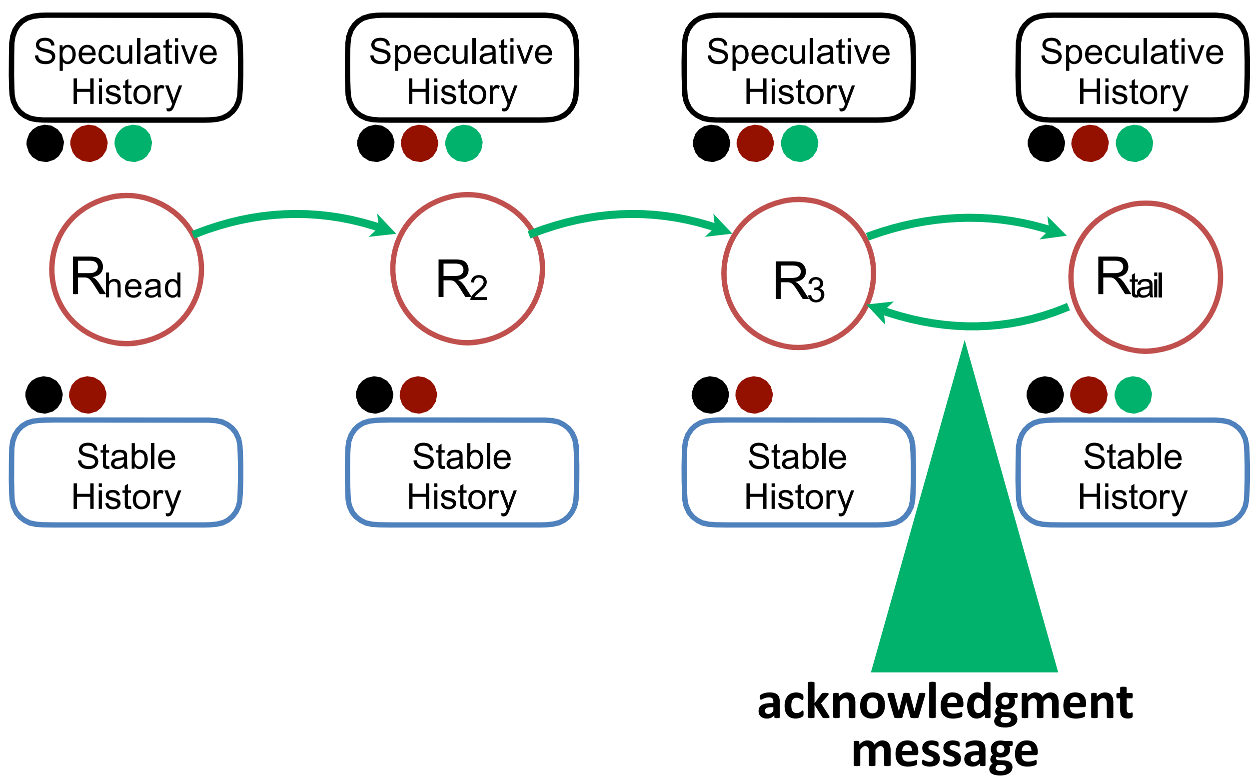


Speculative History
Speculative History
Speculative History
Speculative History
reply
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Updates

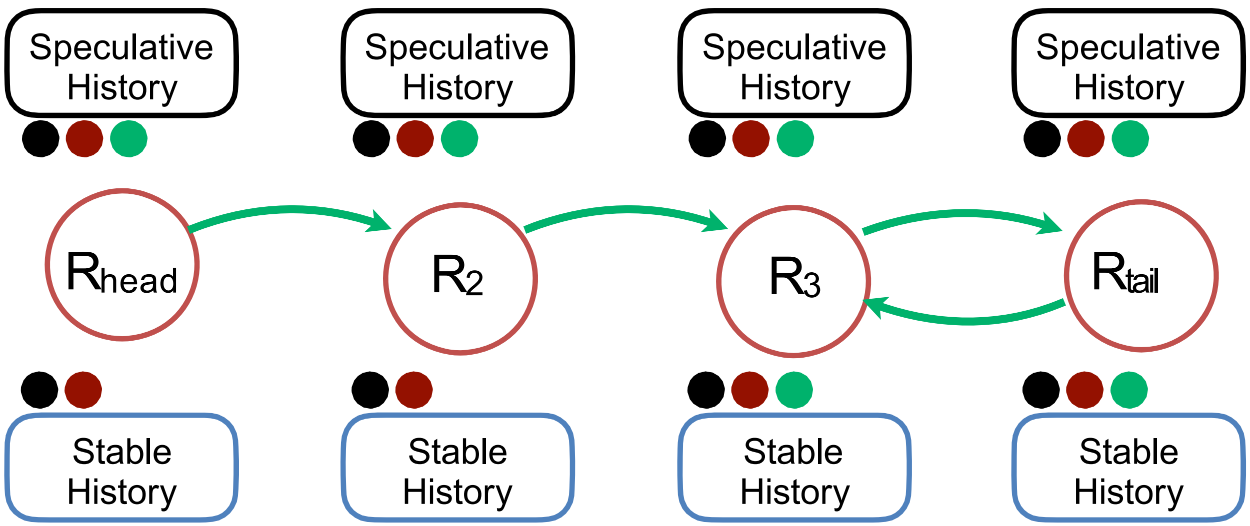

Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Updates

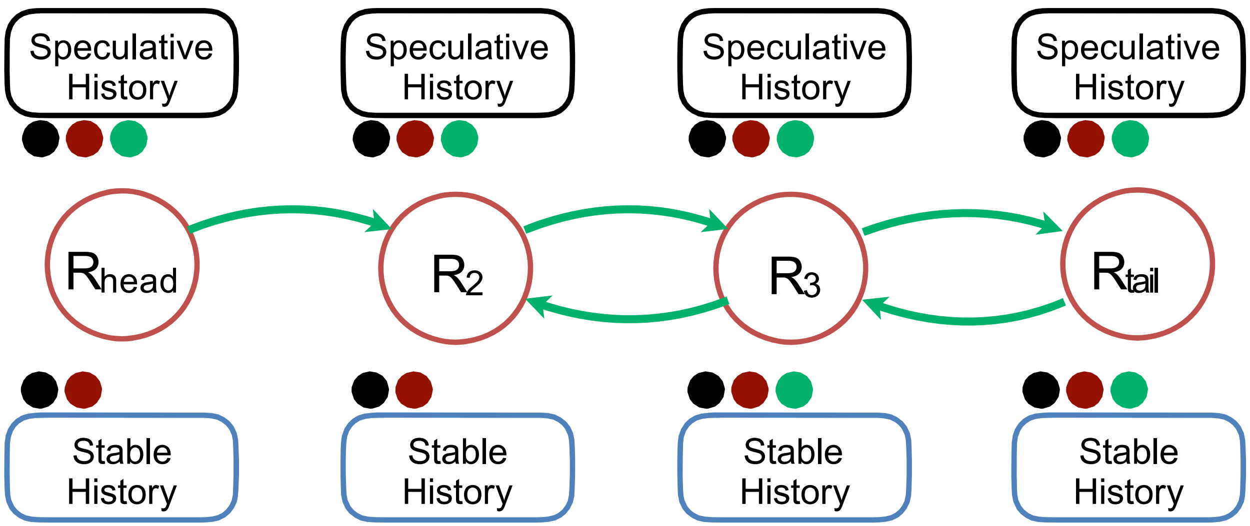


Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History
acknowledgment message

Updates

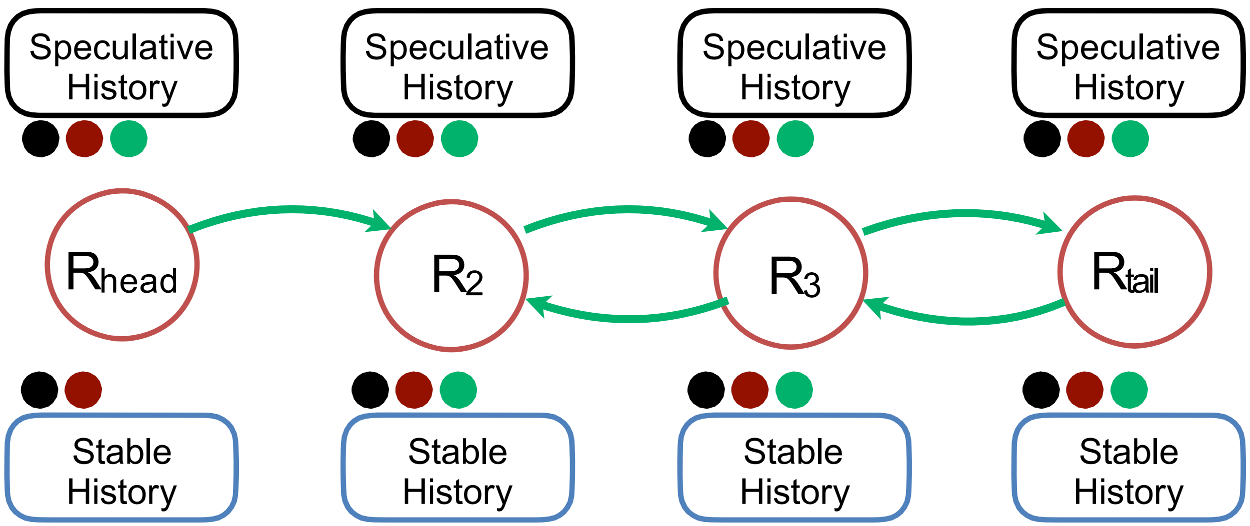

Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Updates



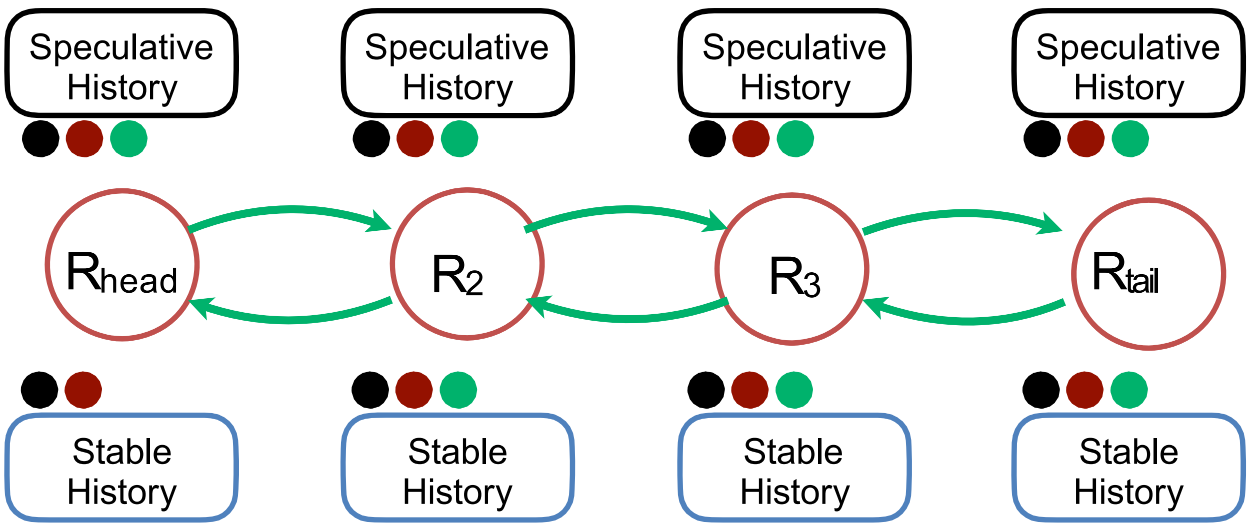
Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Updates

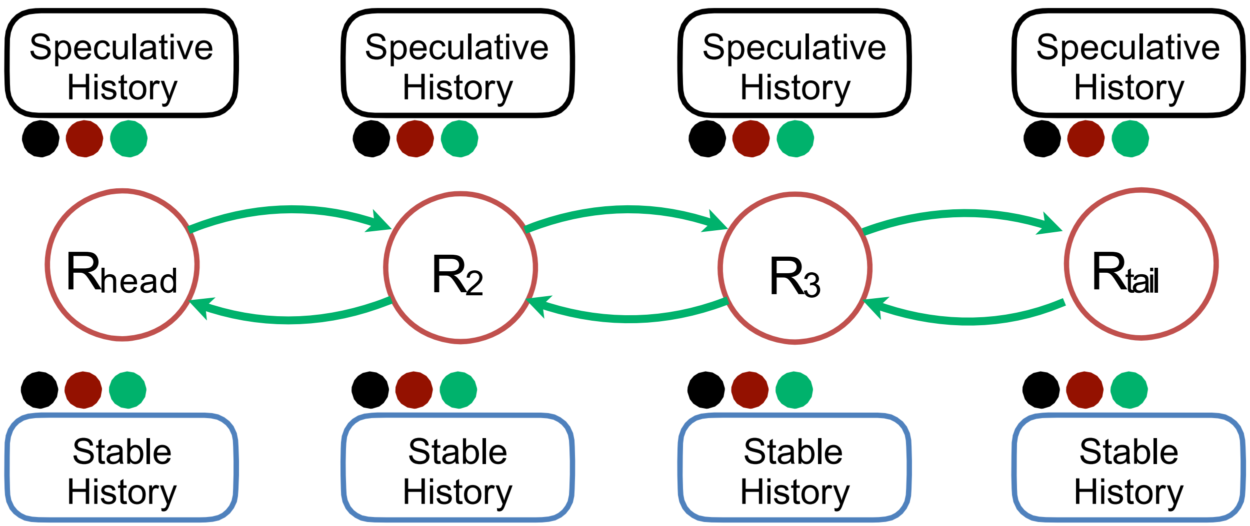


Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Updates

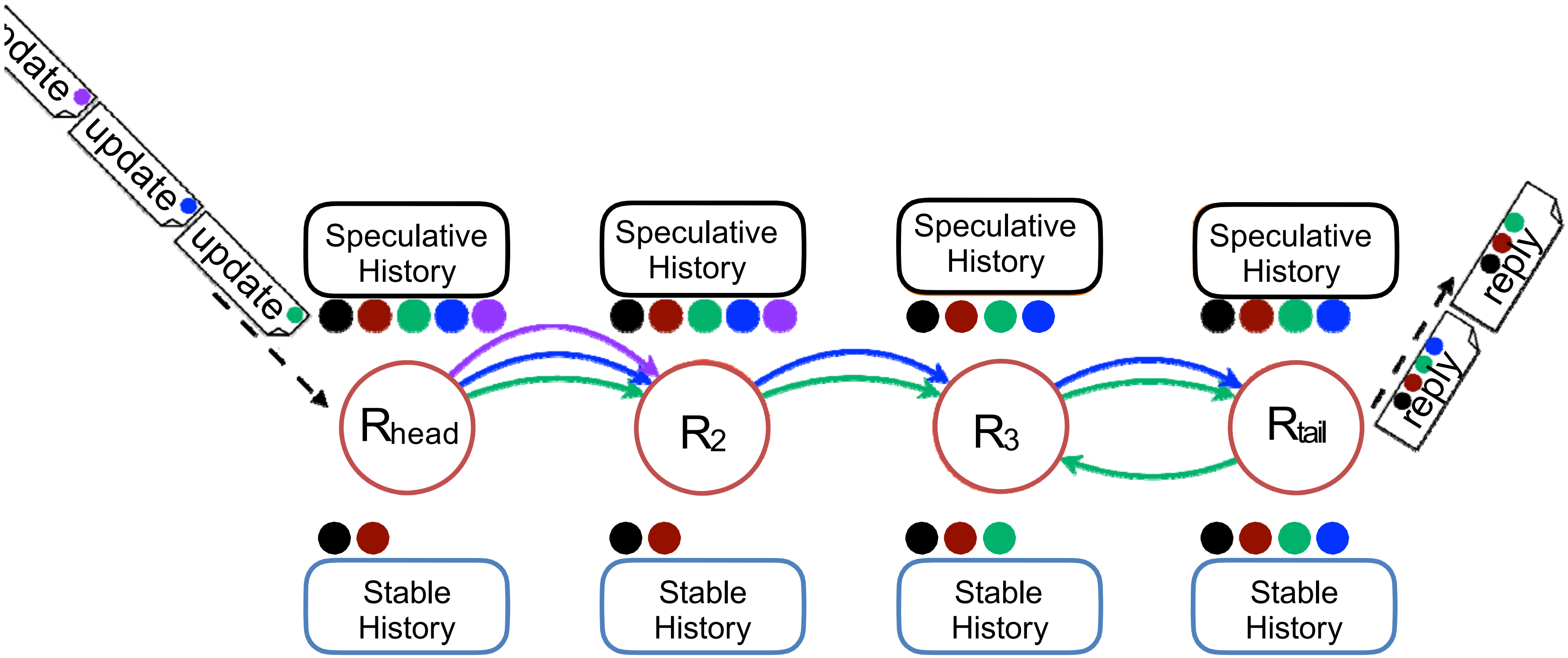

Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Updates



Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

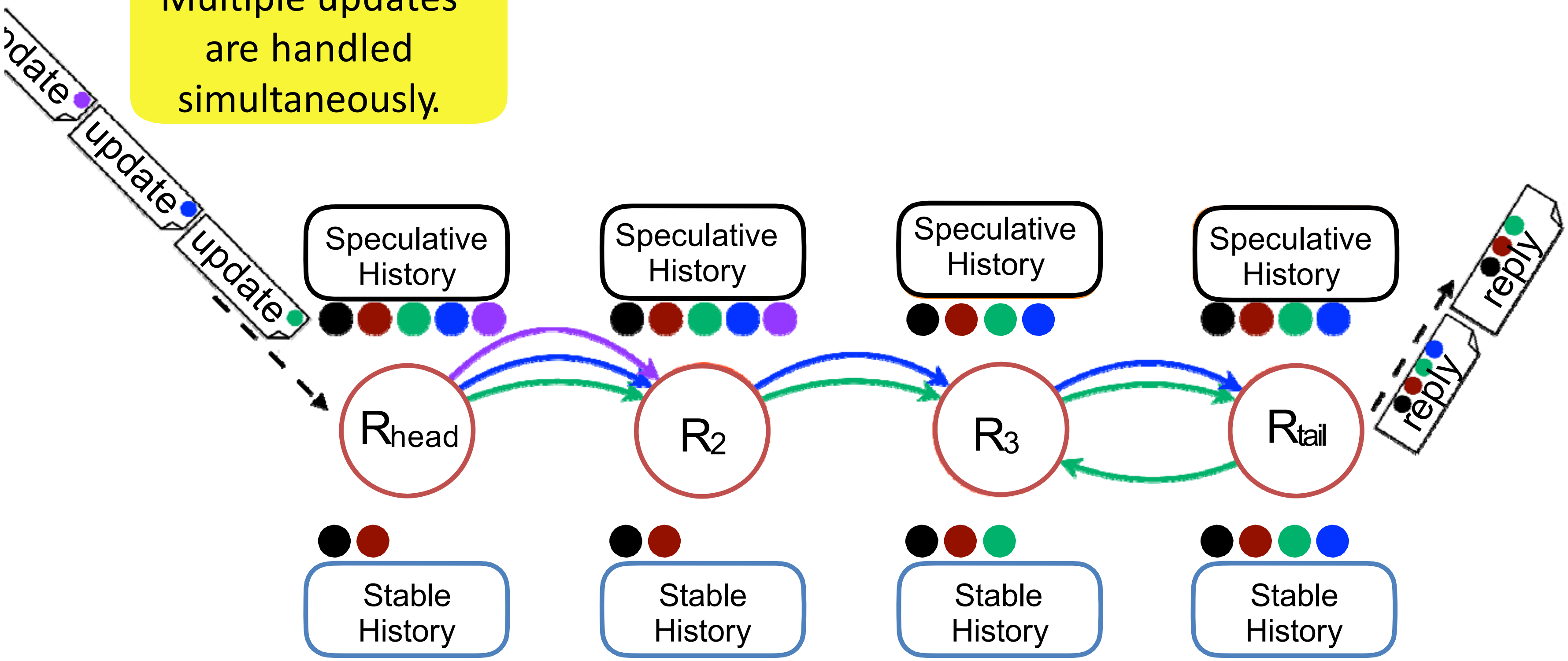
Updates



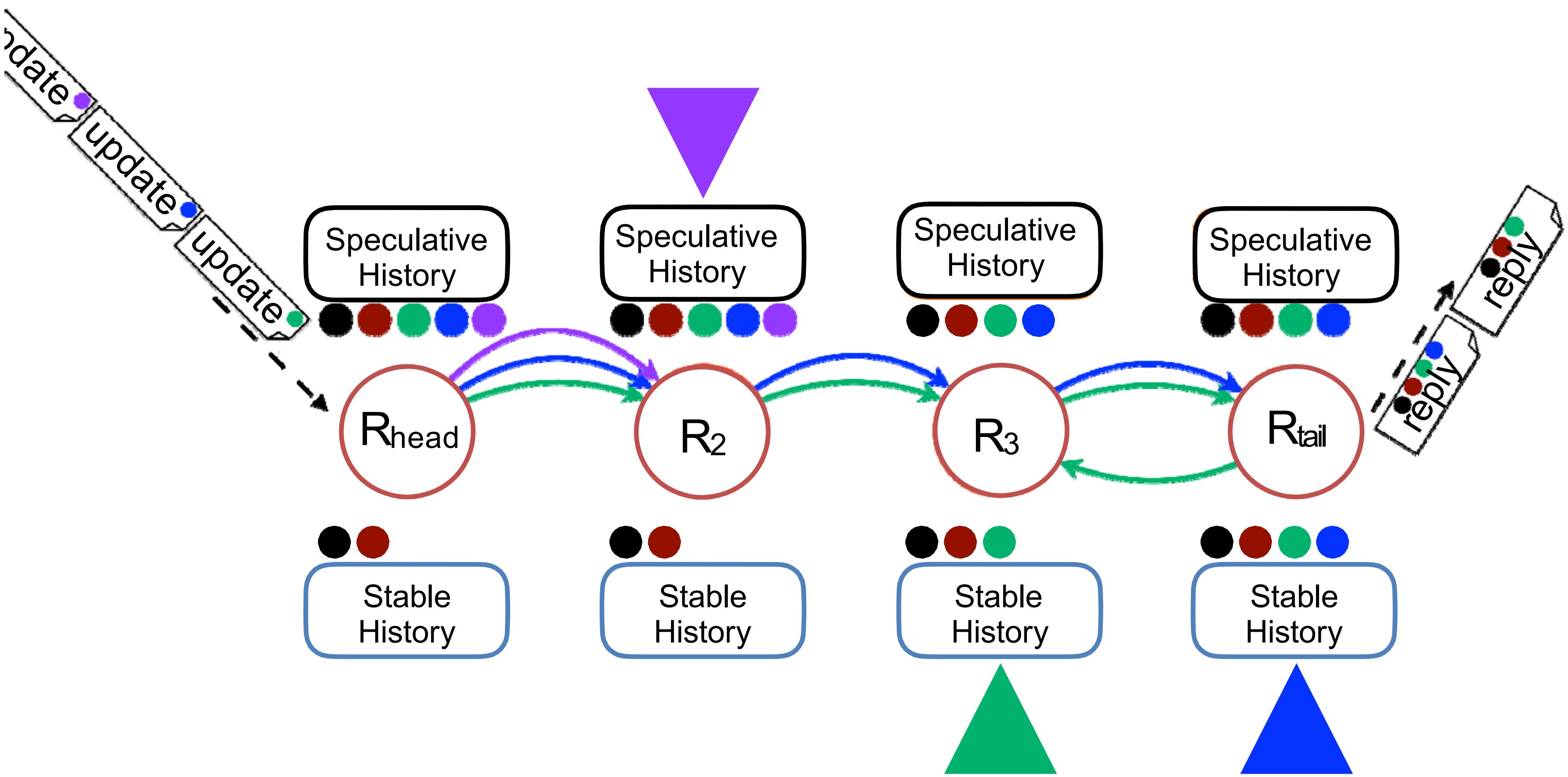
update
update
update
Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
reply
reply
Stable History
Stable History
Stable History
Stable History

Updates

Multiple updates are handled simultaneously.

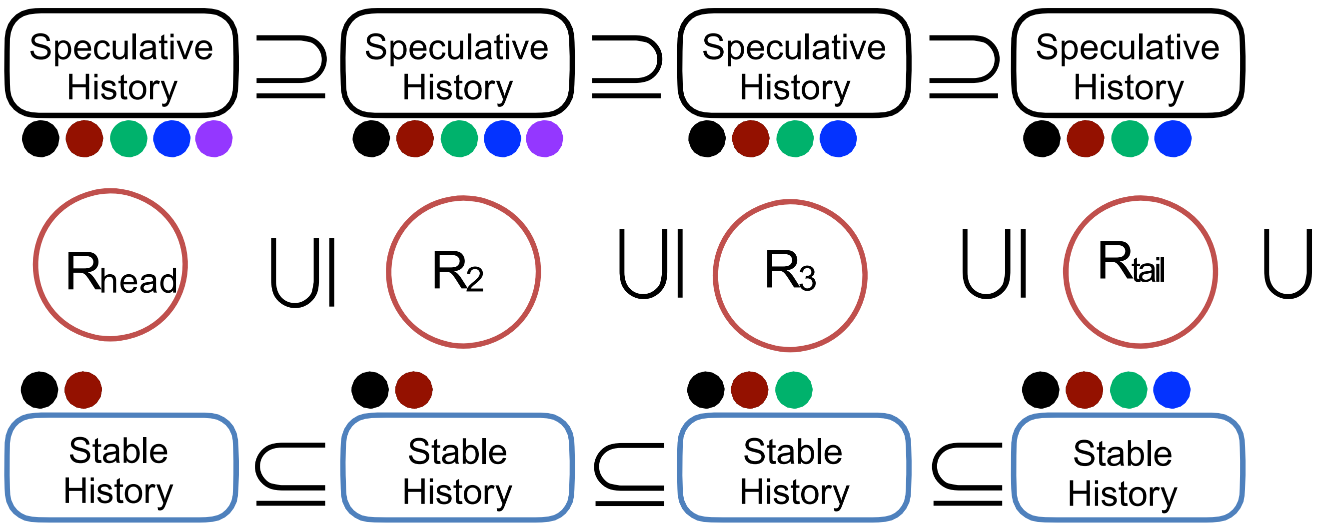


Updates

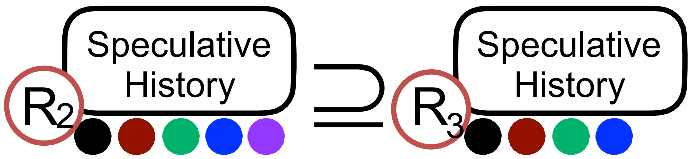


update
update
update
Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
reply
reply
Stable History
Stable History
Stable History
Stable History

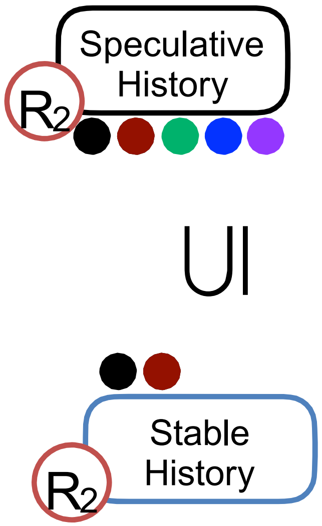
Updates



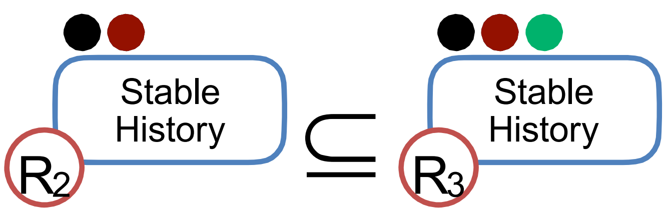
Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History



The speculative history of a node's successor is a **subset** of that node's speculative history.

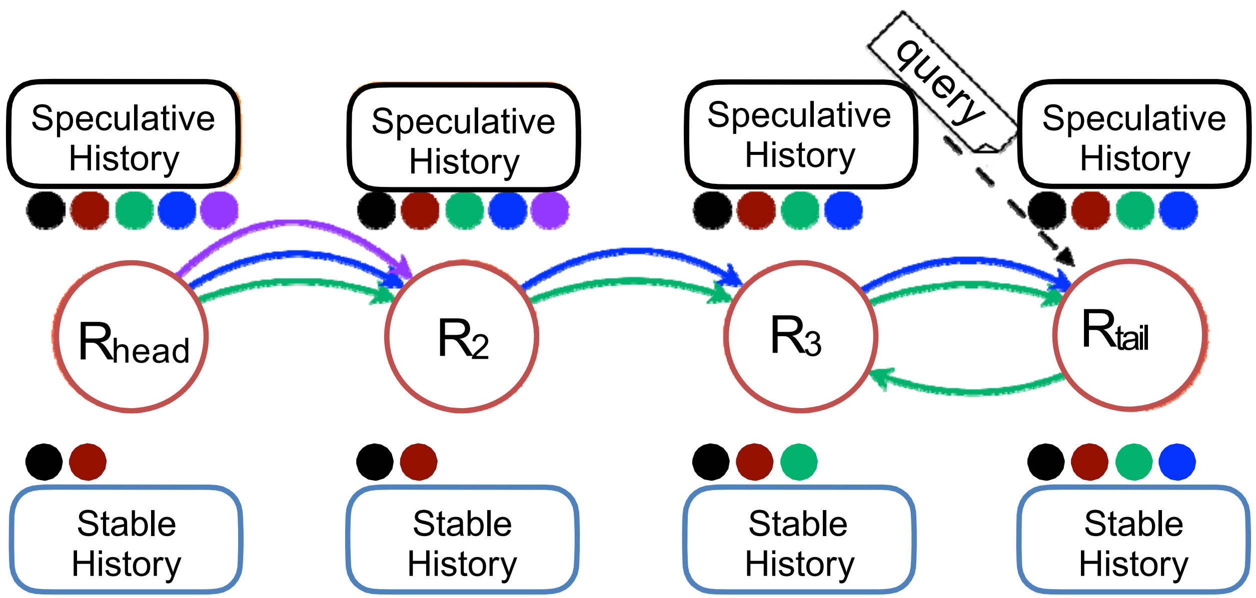


The speculative history of a node is a **superset** of its stable history.



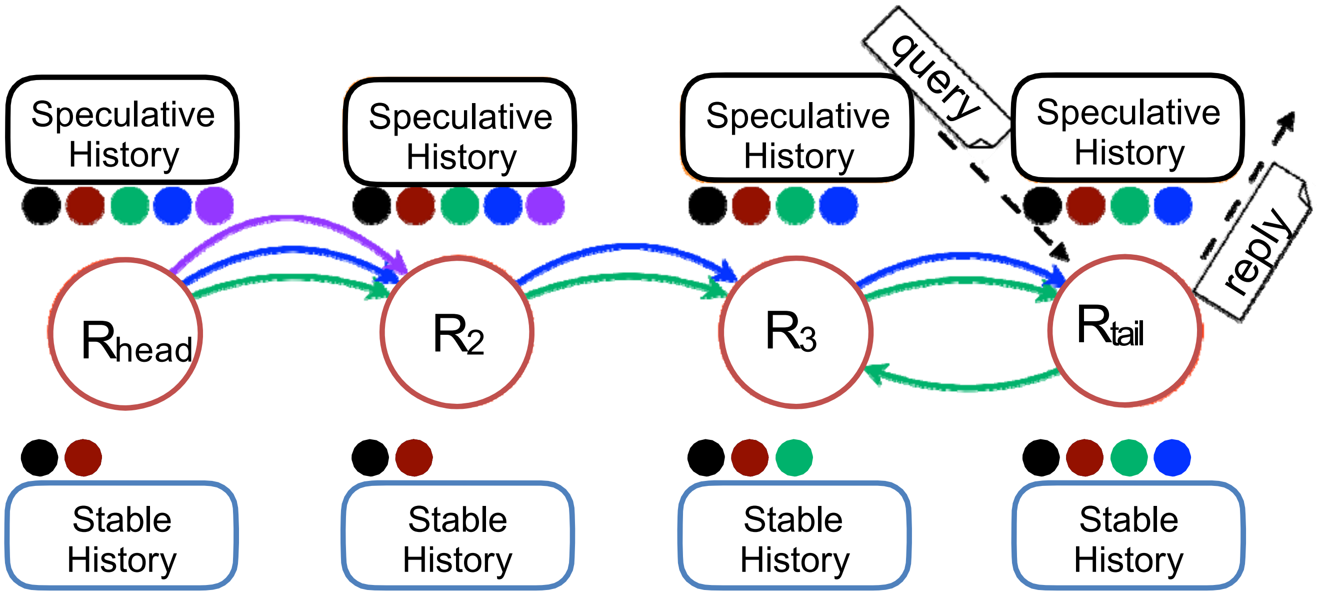
The stable history of a node's successor is a **superset** of that node's stable history.

Queries



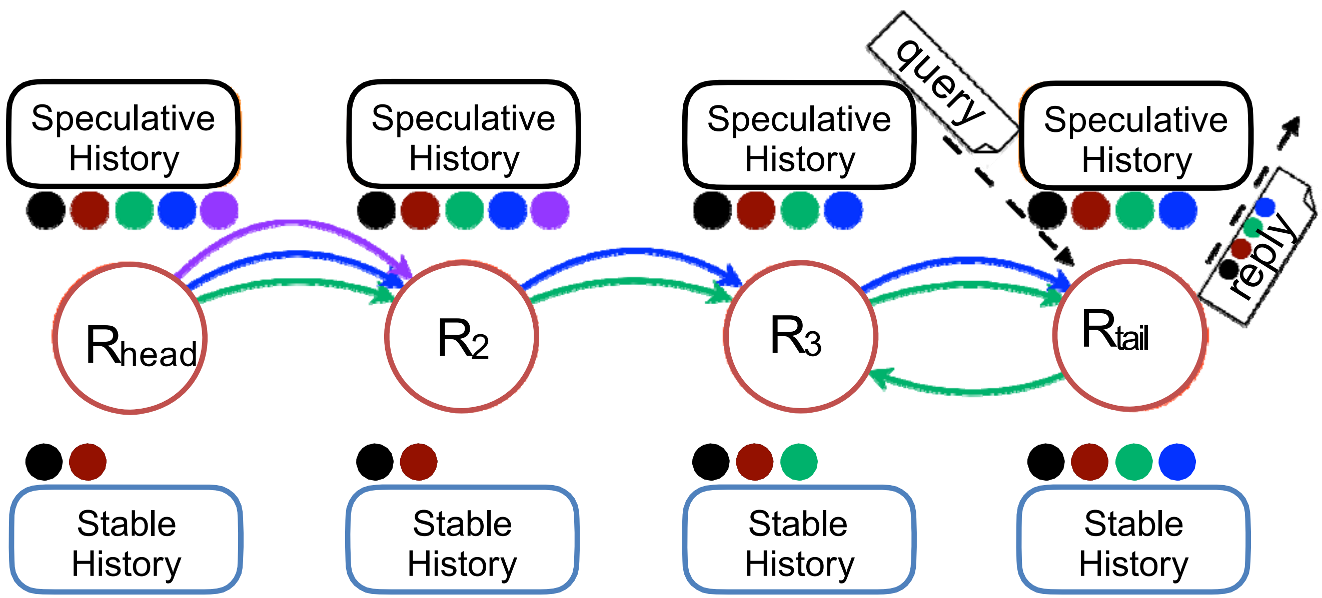
query
Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Queries



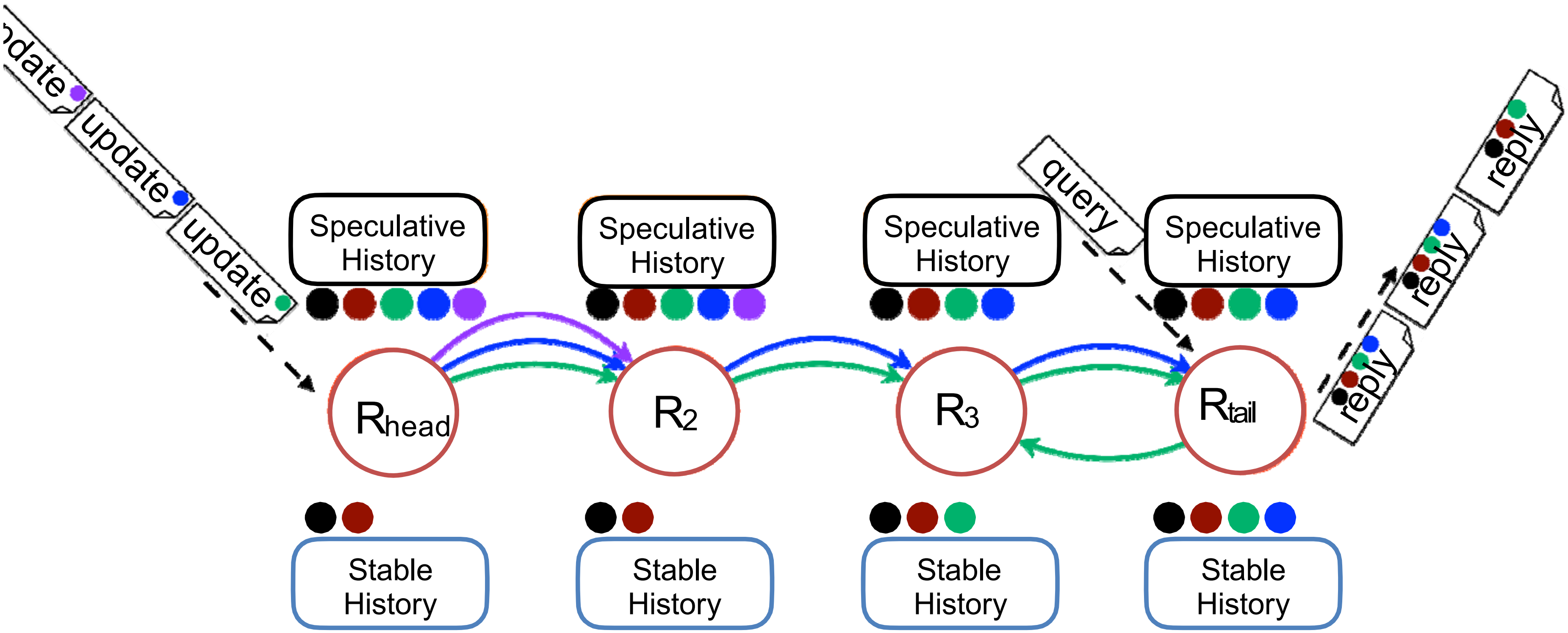
query
Speculative History
Speculative History
Speculative History
Speculative History
reply
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Queries



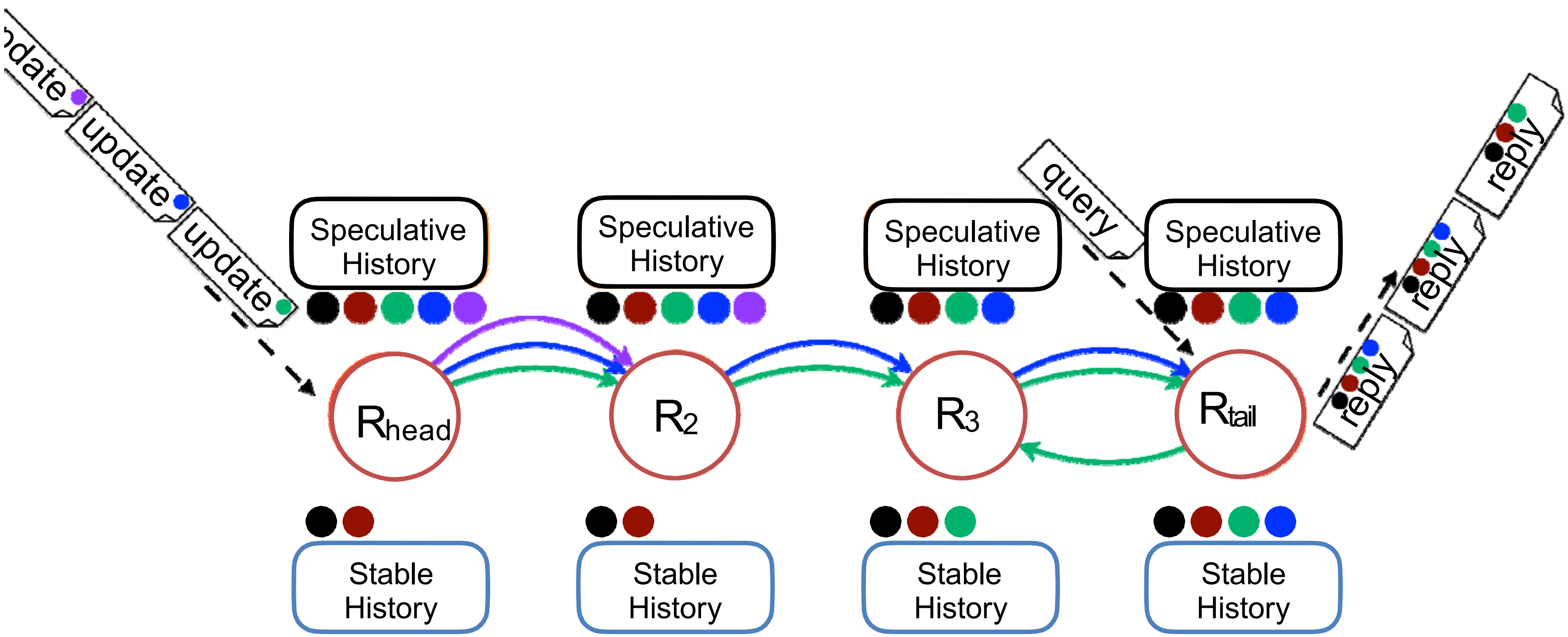
query
Speculative History
Speculative History
Speculative History
Speculative History
reply
Rhead
R2
R3
Rtail
Stable History
Stable History
Stable History
Stable History

Queries



update
update
update
query
Speculative History
Speculative History
Speculative History
Speculative History
Rhead
R2
R3
Rtail
reply
reply
reply
Stable History
Stable History
Stable History
Stable History

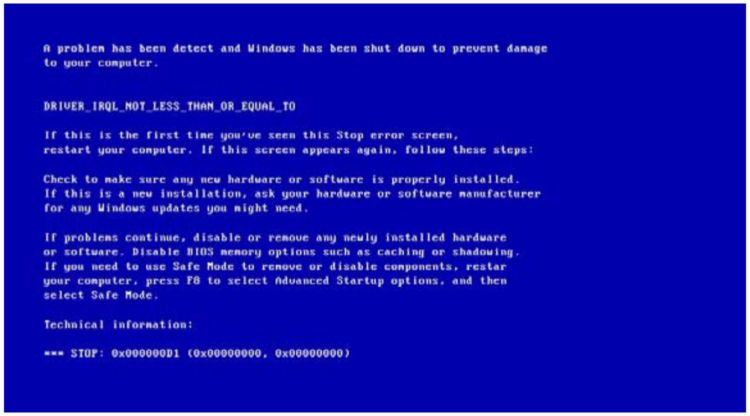
Queries



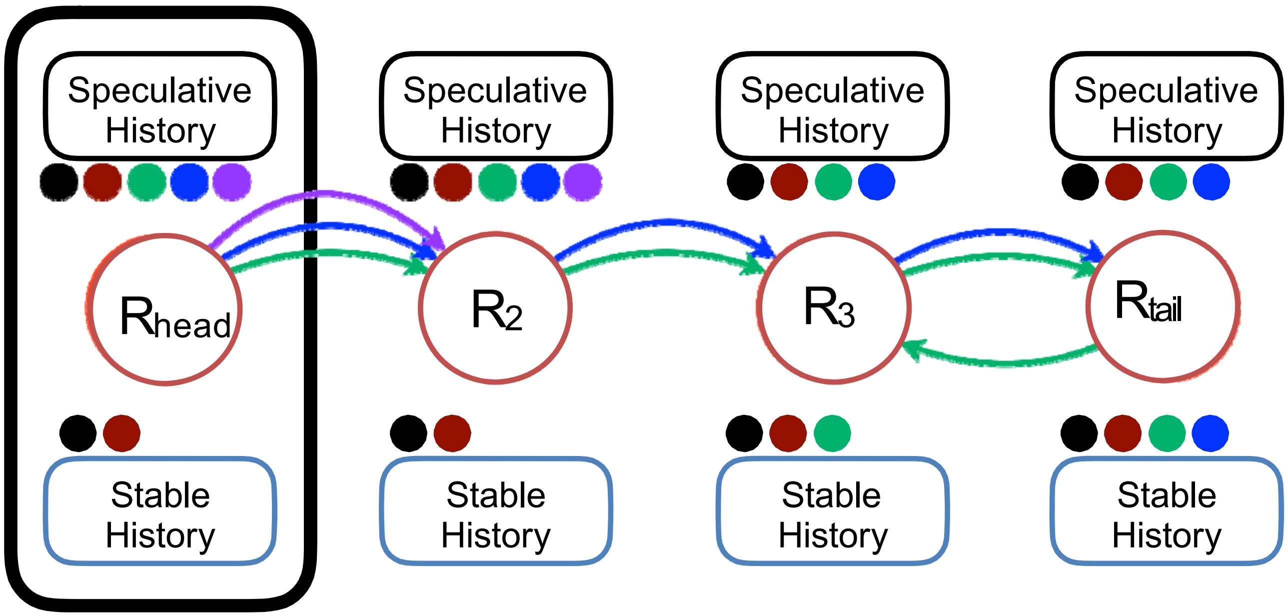
The tail is the point of linearization!

Failures

- Head failure
- Middle node failure
- Tail failure

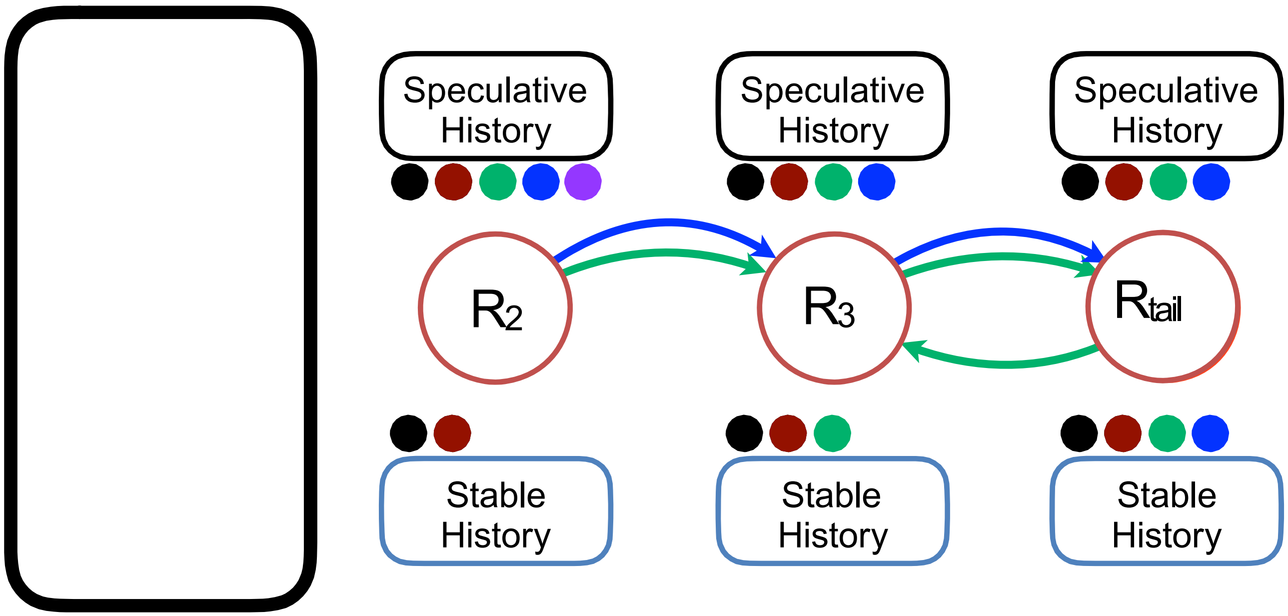


Head Failure I



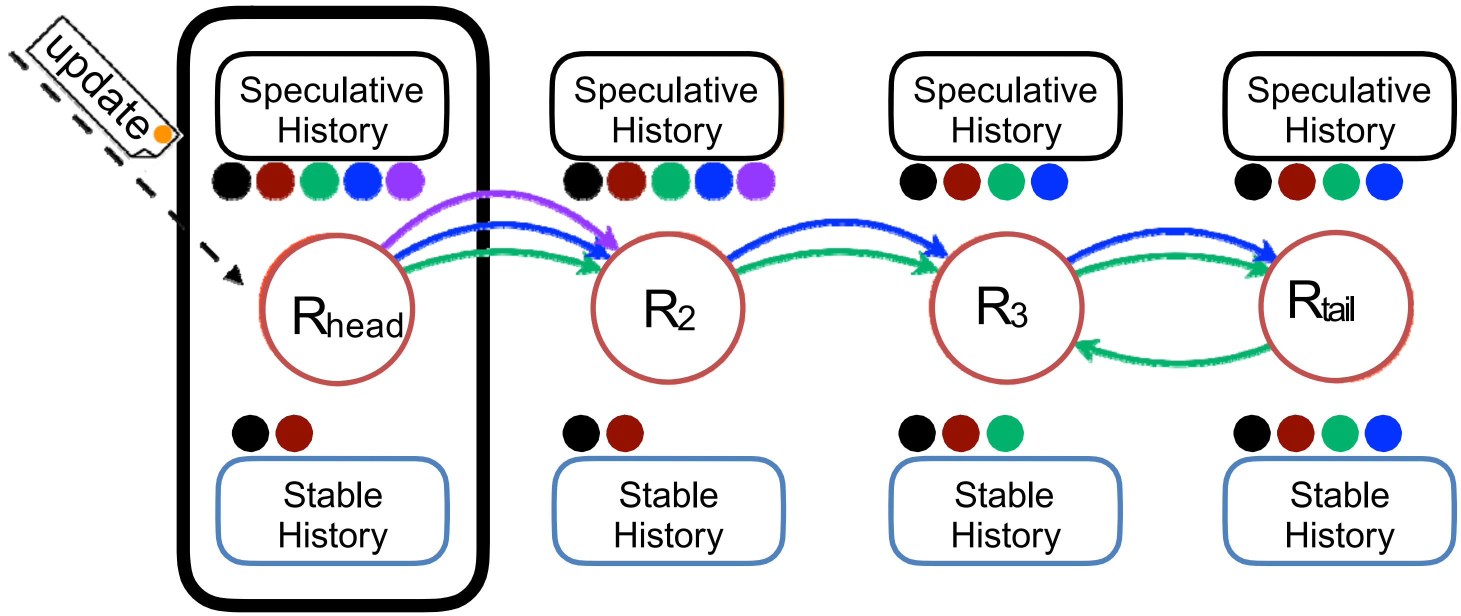
Head Failure II

R_2 becomes new head



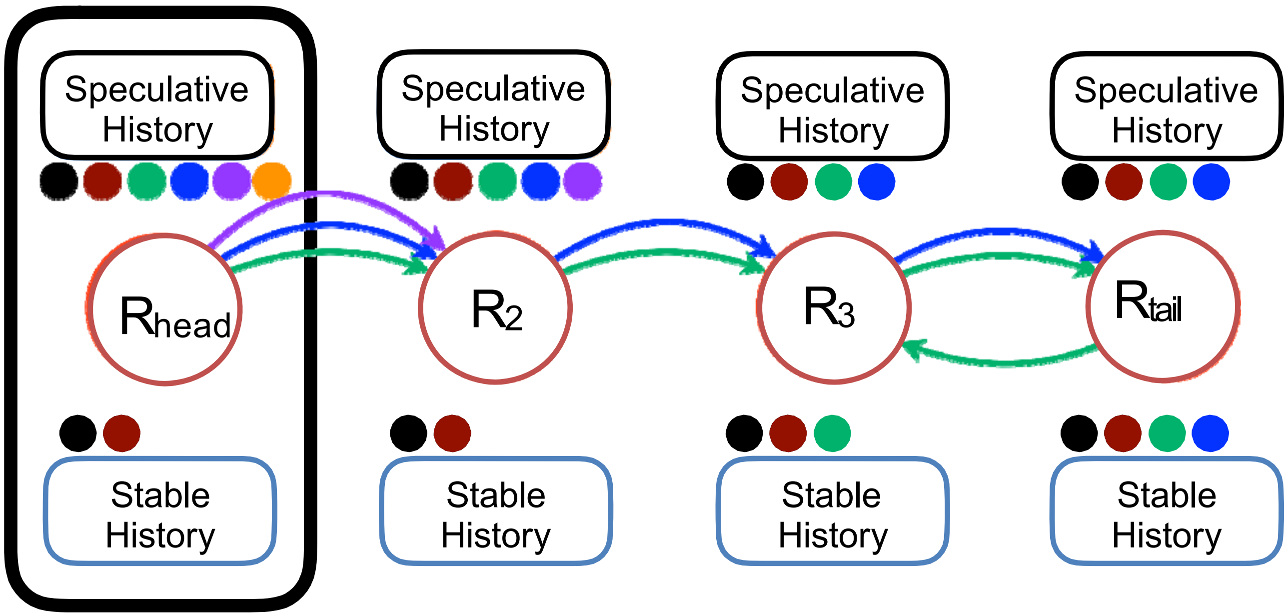
Head Failure I

In-flight, non-propagated updates



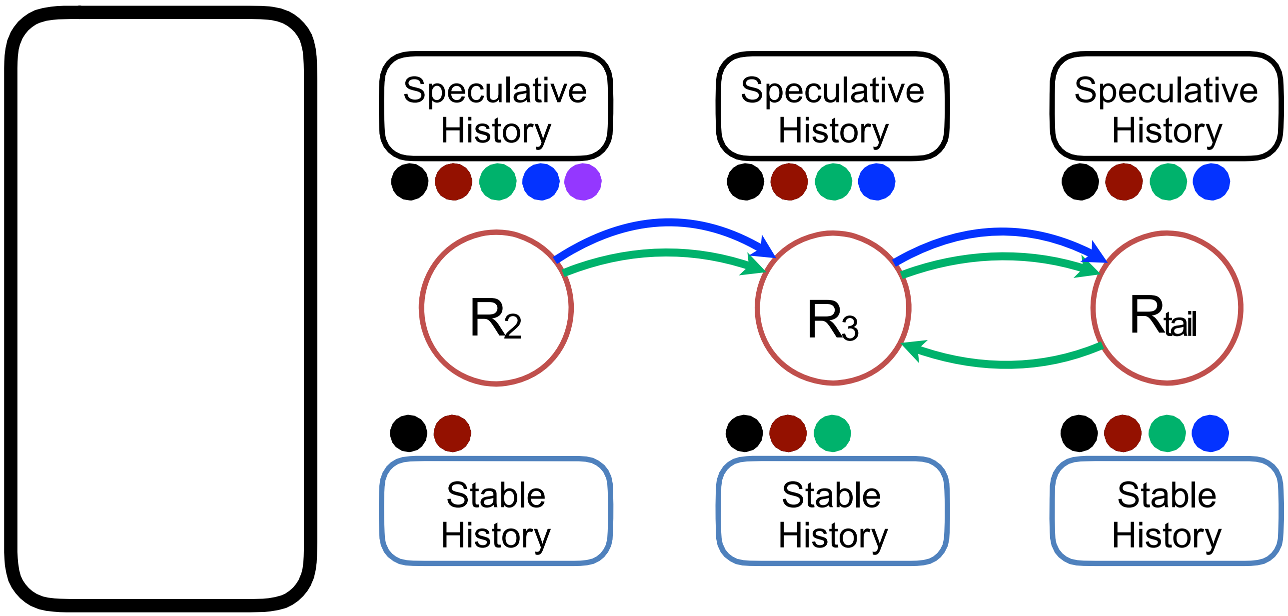
Head Failure II

In-flight, non-propagated updates



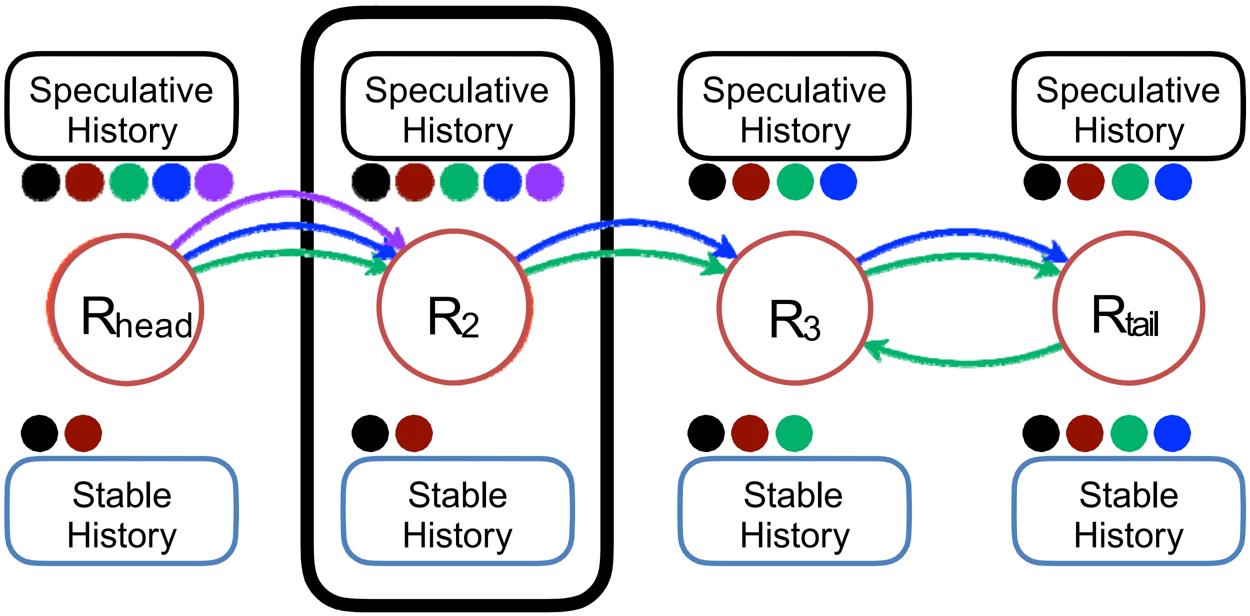
Head Failure III

In-flight, non-propagated updates

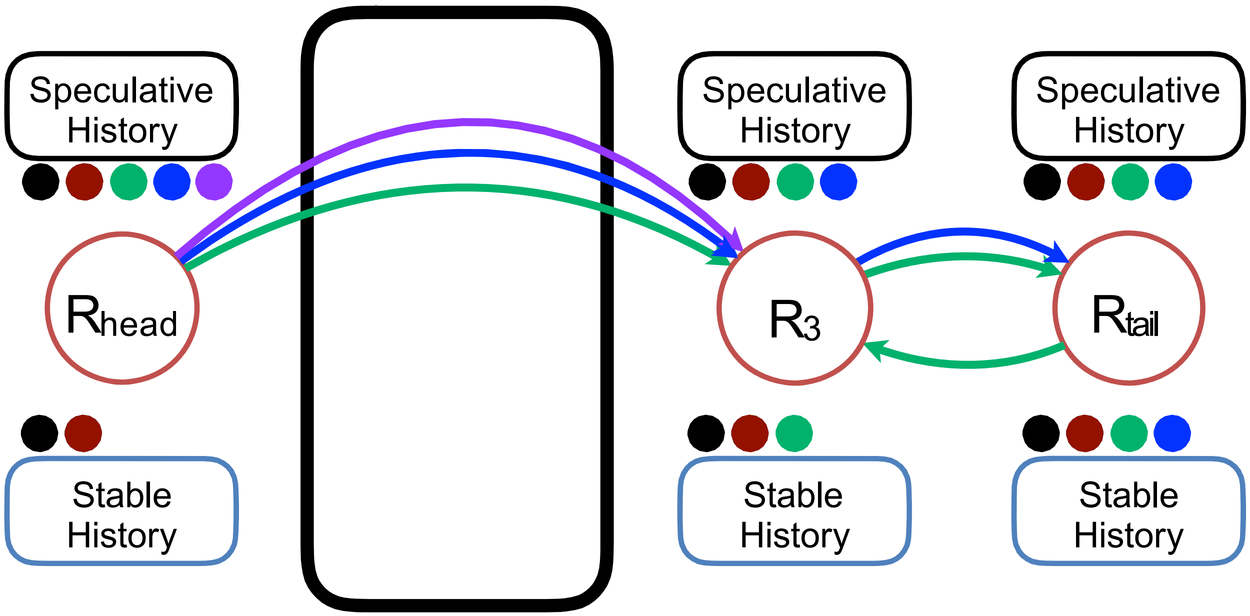


Client would not receive a reply, timeout, and retry

Middle Node Failure I

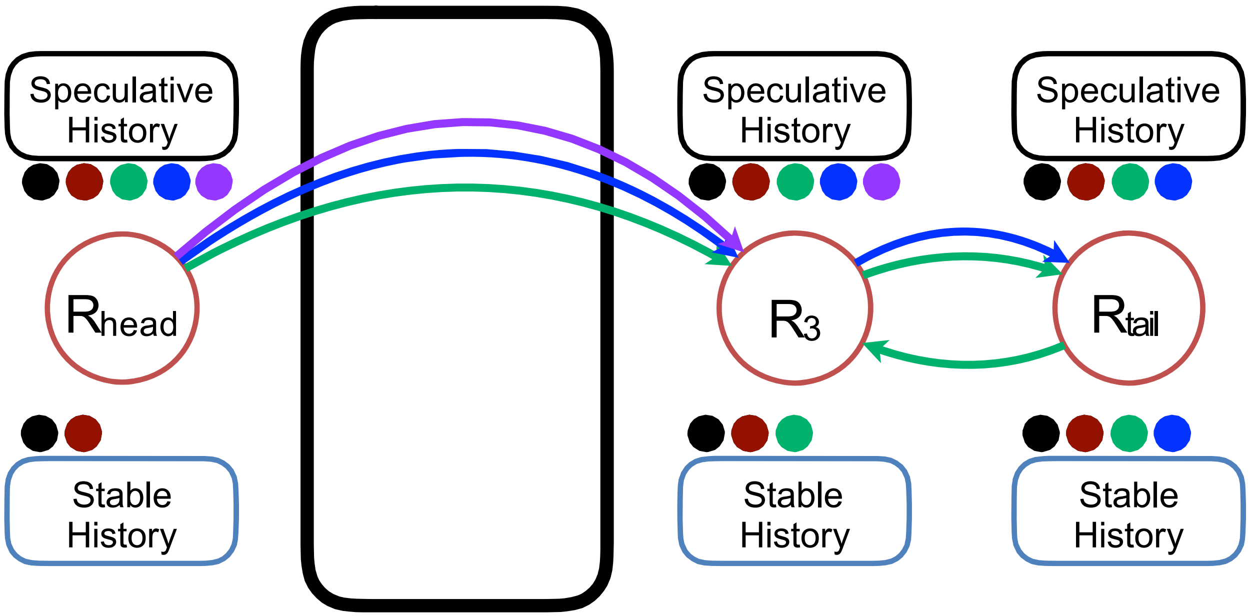


Middle Node Failure II



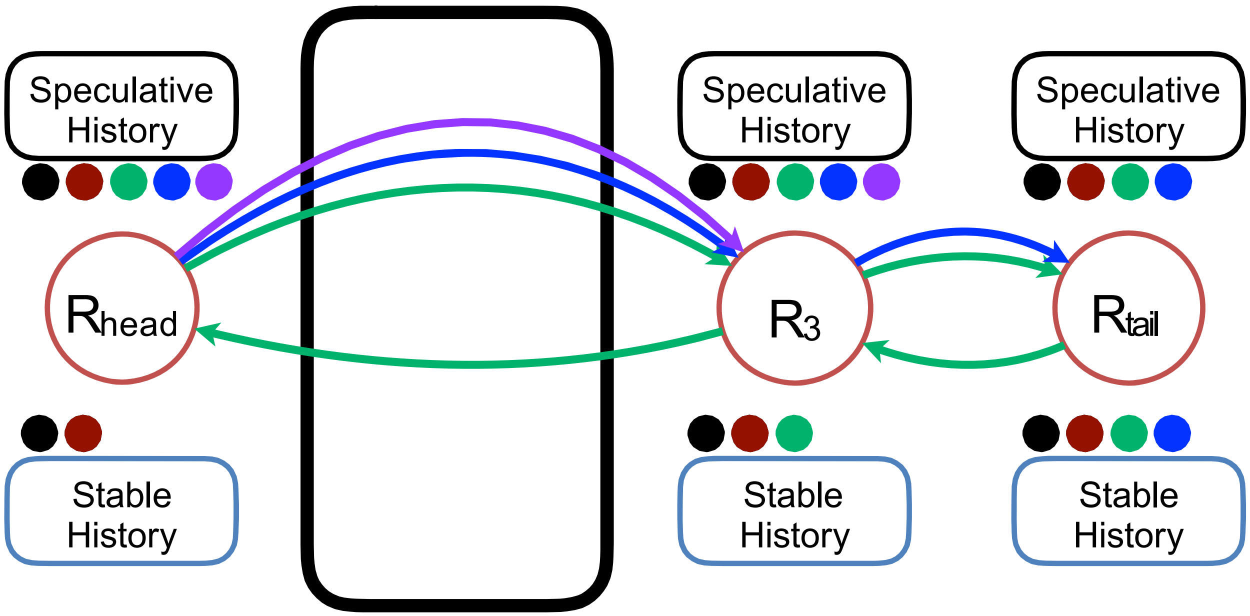
Predecessor needs to talk to failed node's successor

Middle Node Failure III

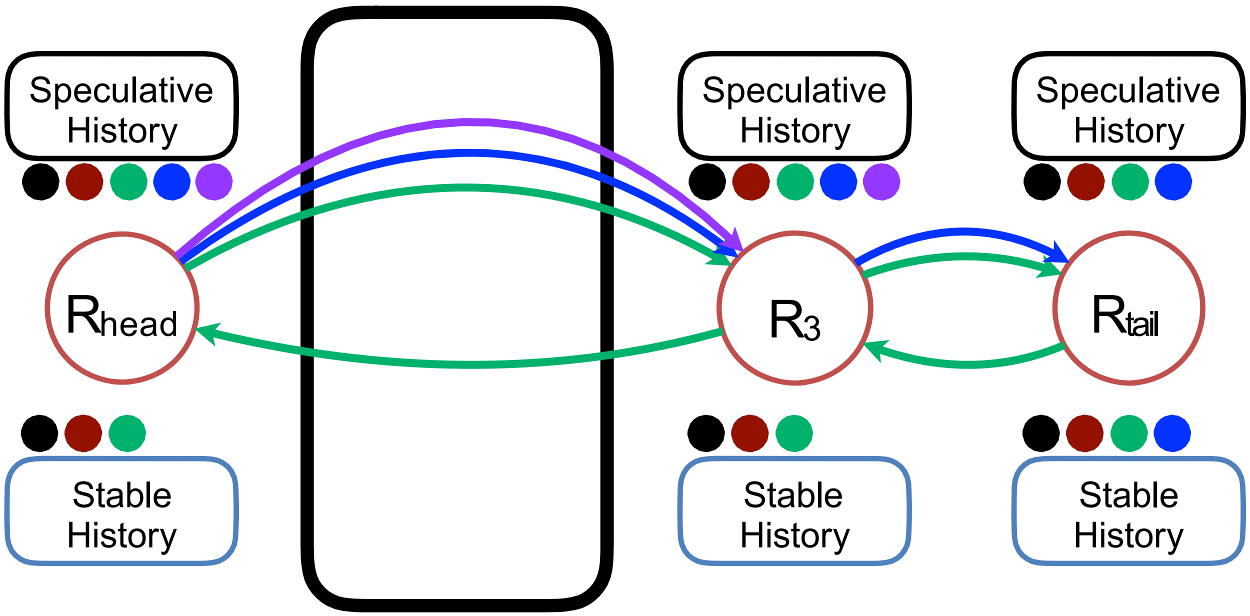


Predecessor propagates update to new successor

Middle Node Failure IV

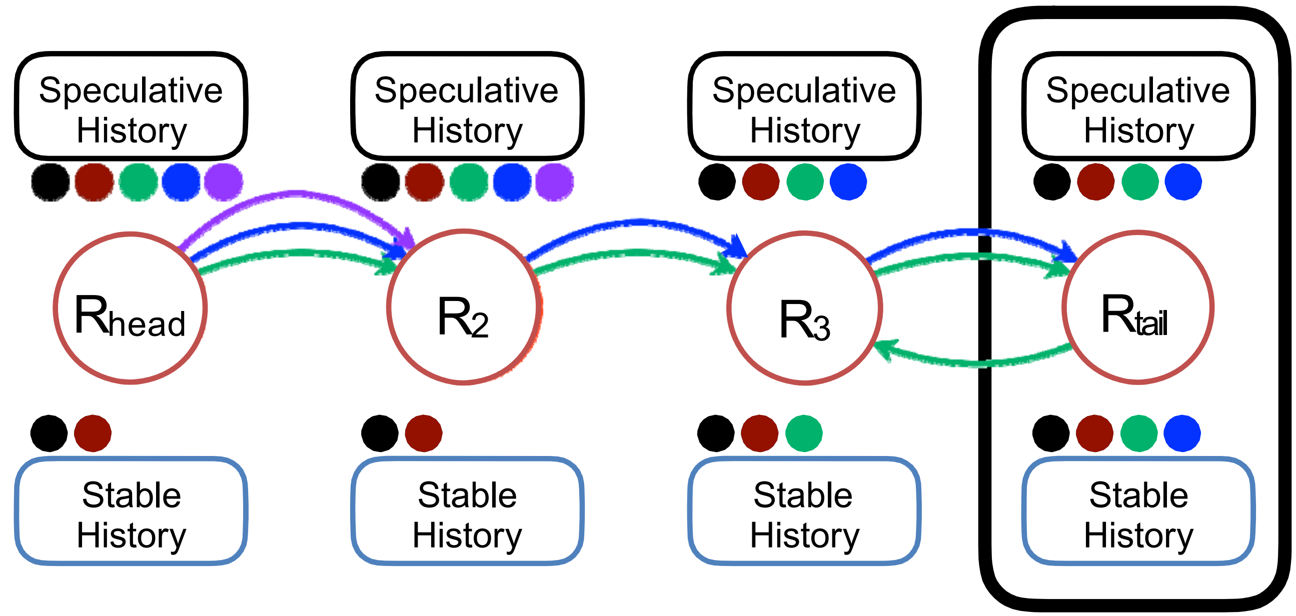


Middle Node Failure V

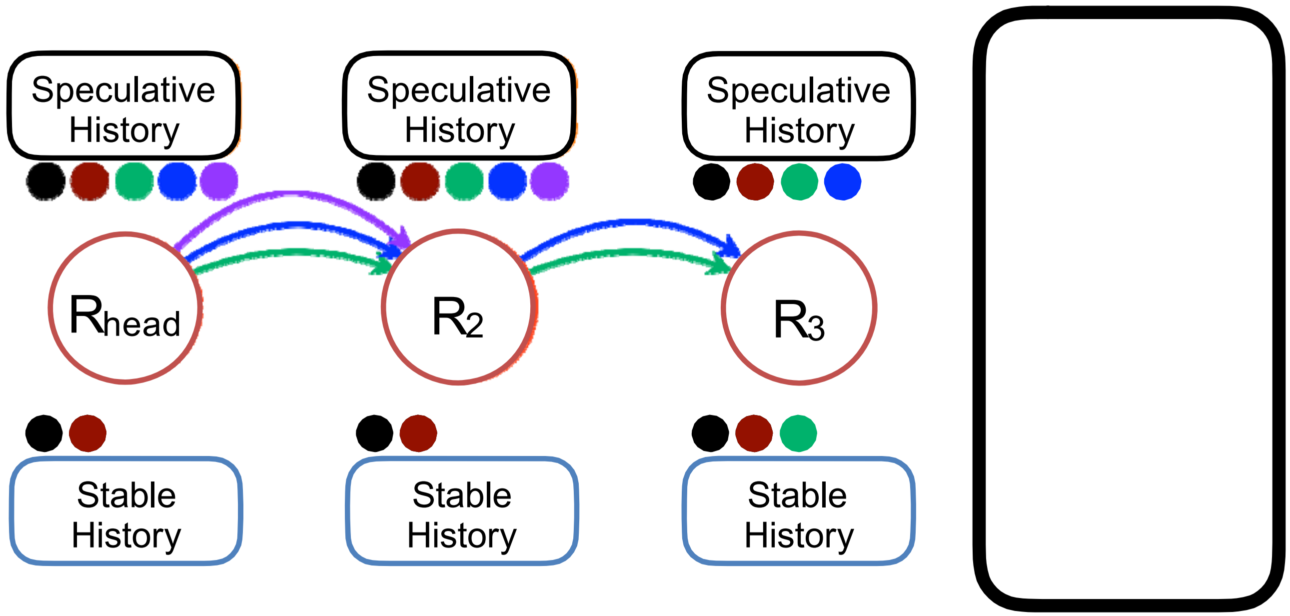


Successor propagates in-flight acknowledgements to new predecessor

Tail Failure I

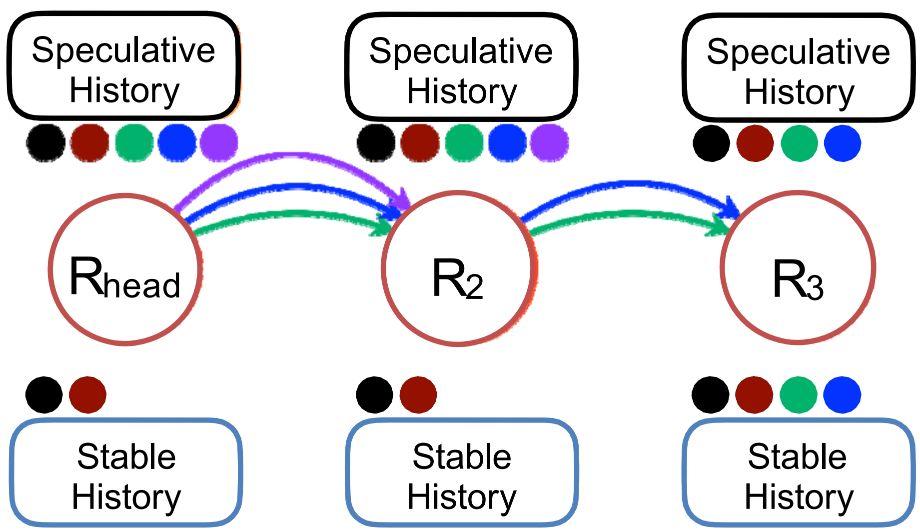


Tail Failure II



R_3 becomes new tail

Tail Failure III



R_3 flushes its speculative history s.t. stable equals speculative history again

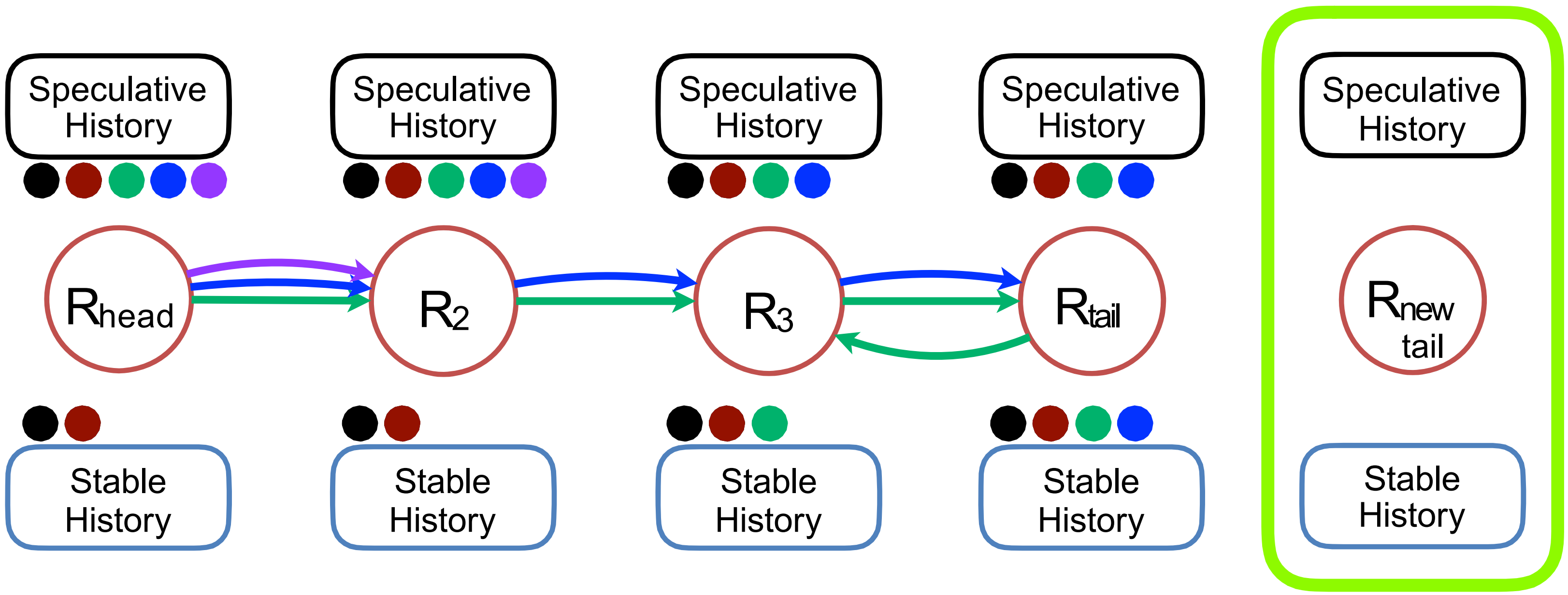
Configuration changes

- Adding a new node

Adding a New Node I

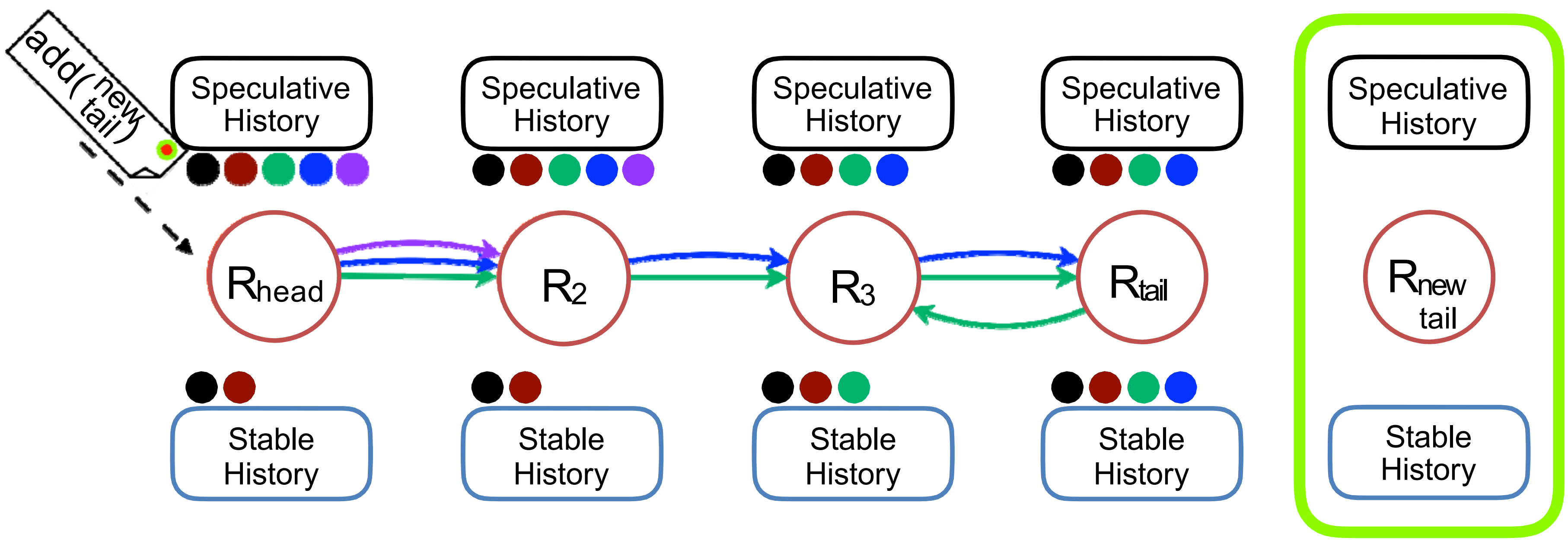
A Configuration Change

Adding an initially empty node



Adding a New Node II

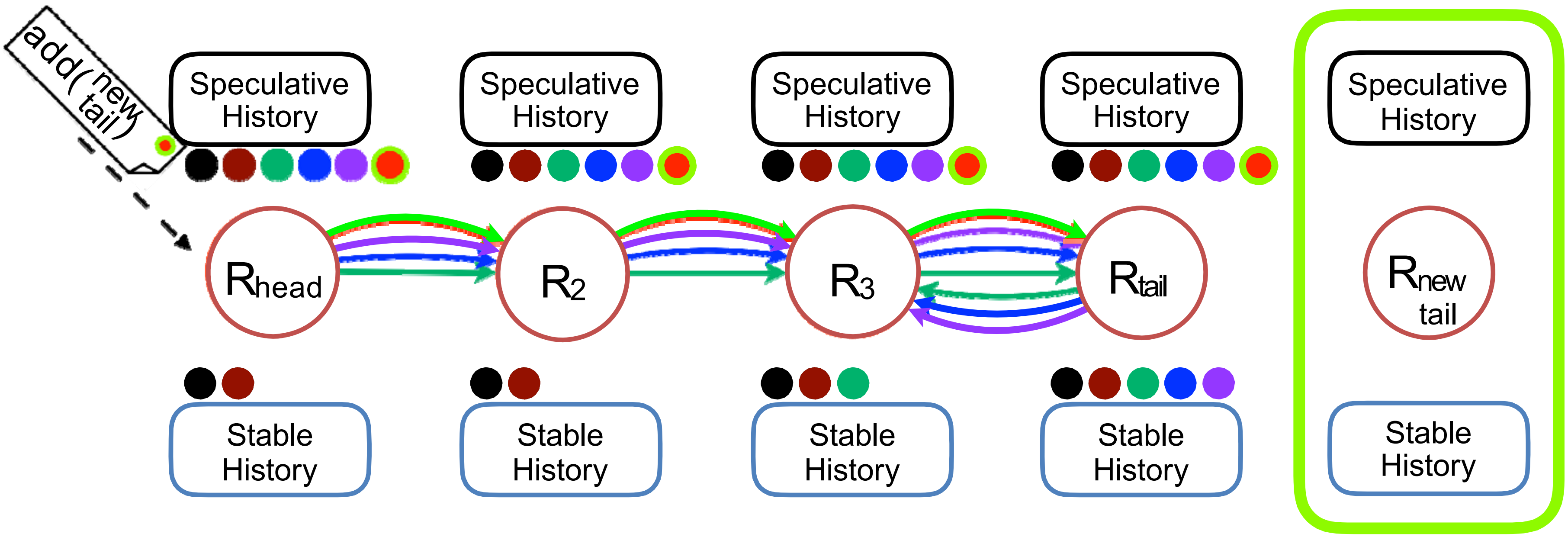
A Configuration Change



- New nodes are added to chain with special **configuration updates**, added to histories: **add(nodeid)**
- Entire chain is build in this manner

Adding a New Node

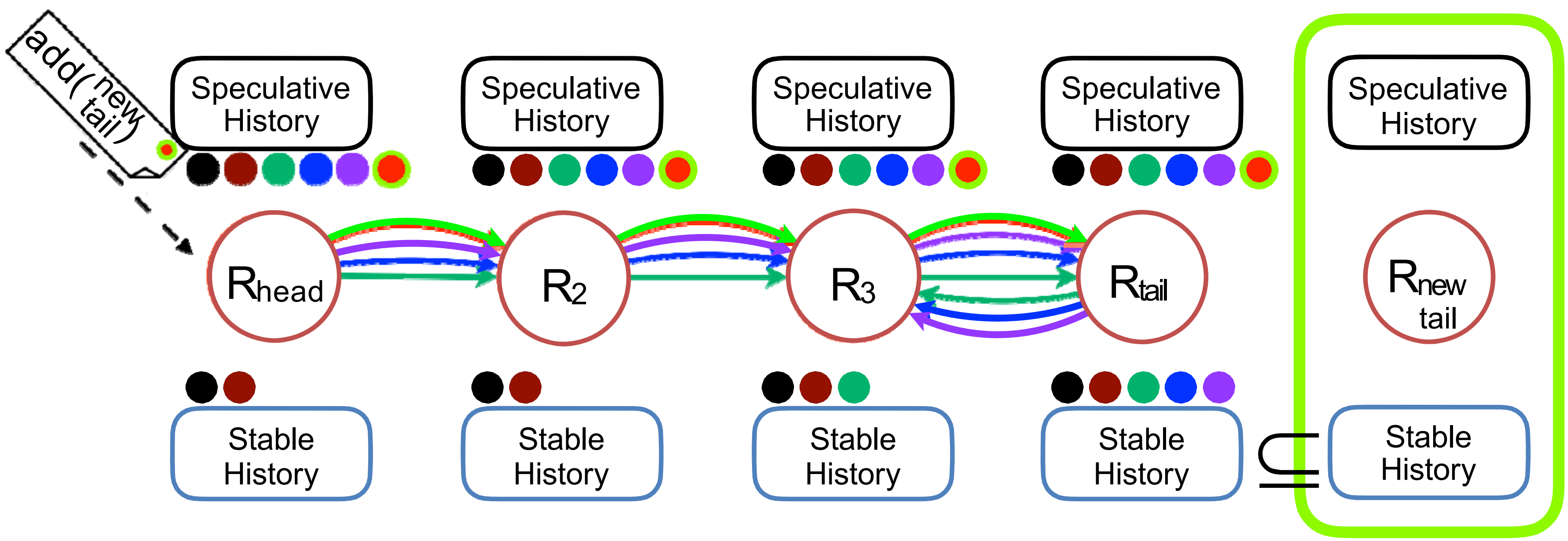
Inferring Configuration



- By looking at **order of these updates**, a node can **determine configuration of chain**
- Old tail discovers it no longer is the tail (via receipt of ●)

Adding a New Node I

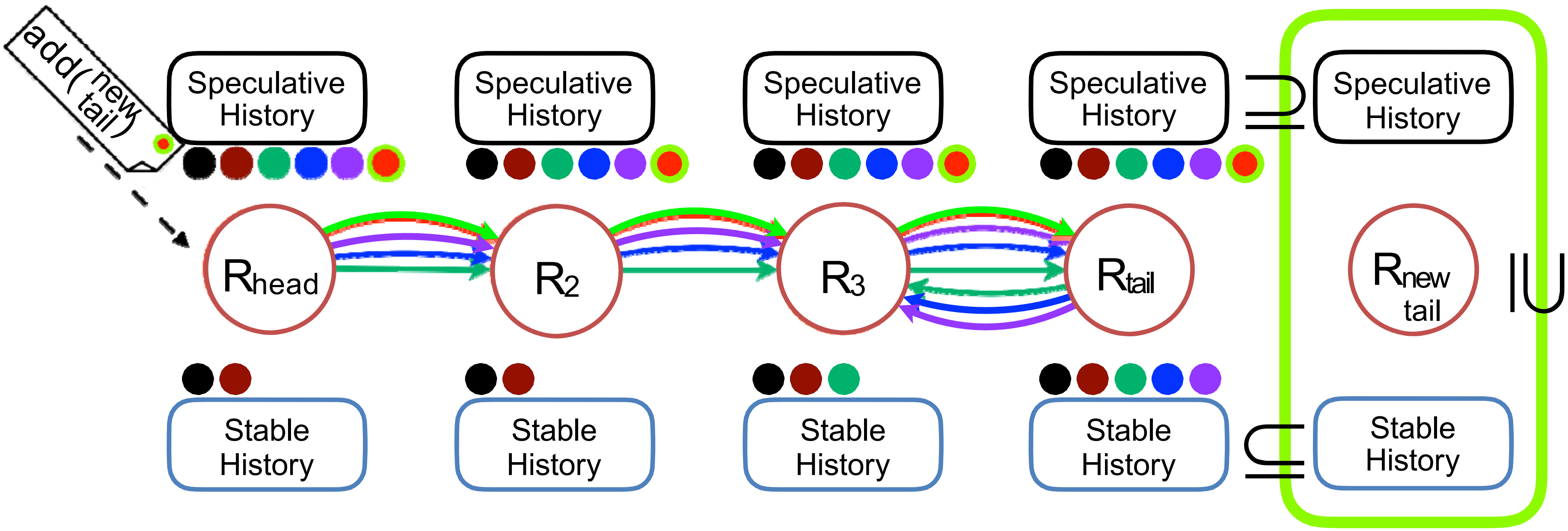
Relationship Among Histories



- **Stable history** of new tail should be **superset of stable history of old tail**

Adding a New Node II

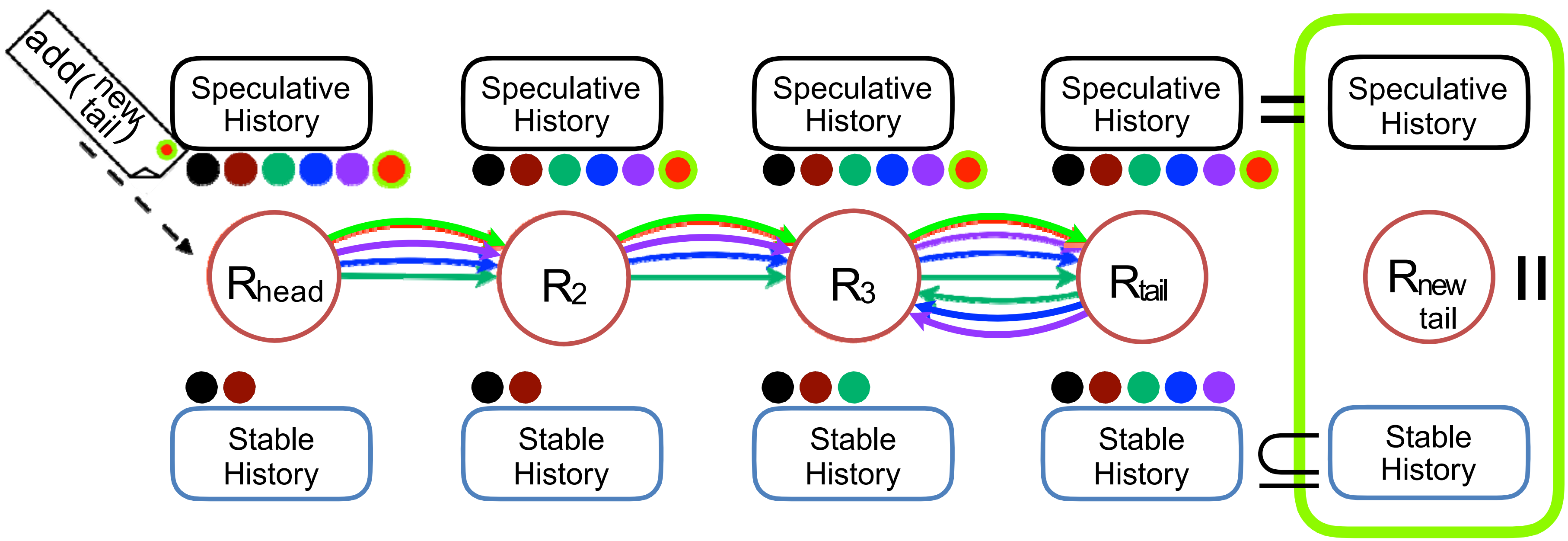
Relationship Among Histories



- **Speculative history** of new tail should be a **superset** of **its stable history**
- speculative and stable histories of new tail should be equal to the speculative history of tail
- old tail should not answer to queries when the new tail should.

Adding a New Node III

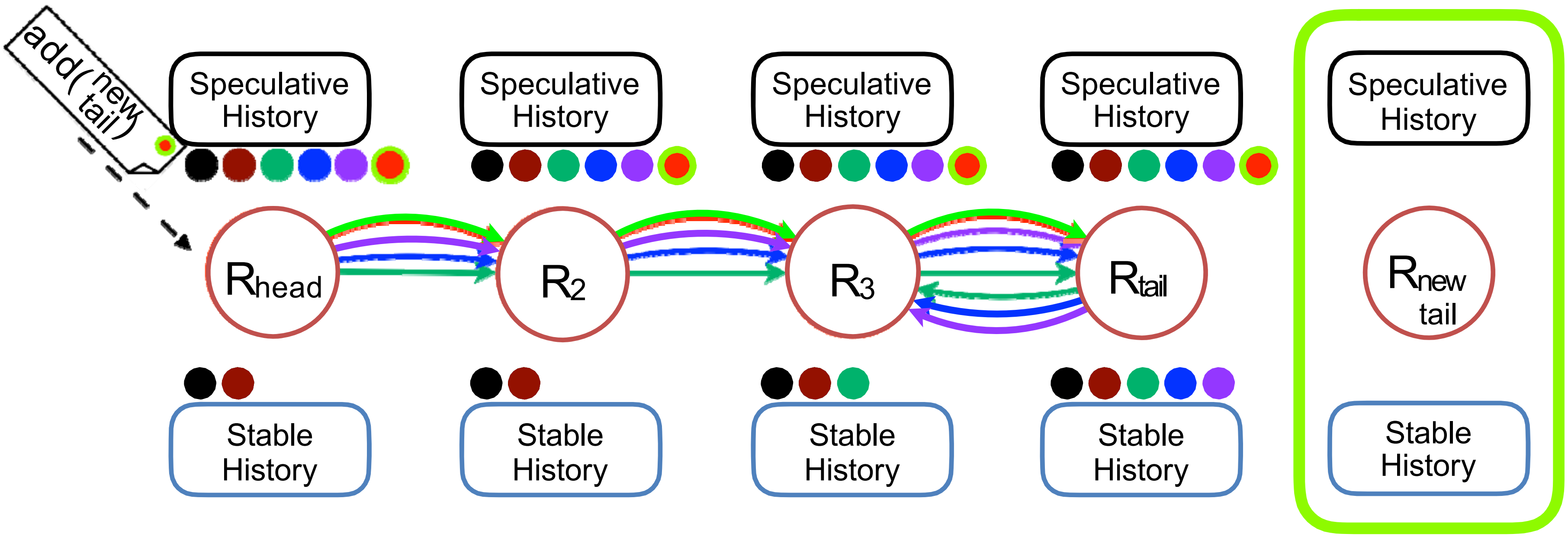
Relationship Among Histories



- **Speculative** and **stable histories** of new tail should become **equal** to the **speculative history of old tail**

Adding a New Node IV

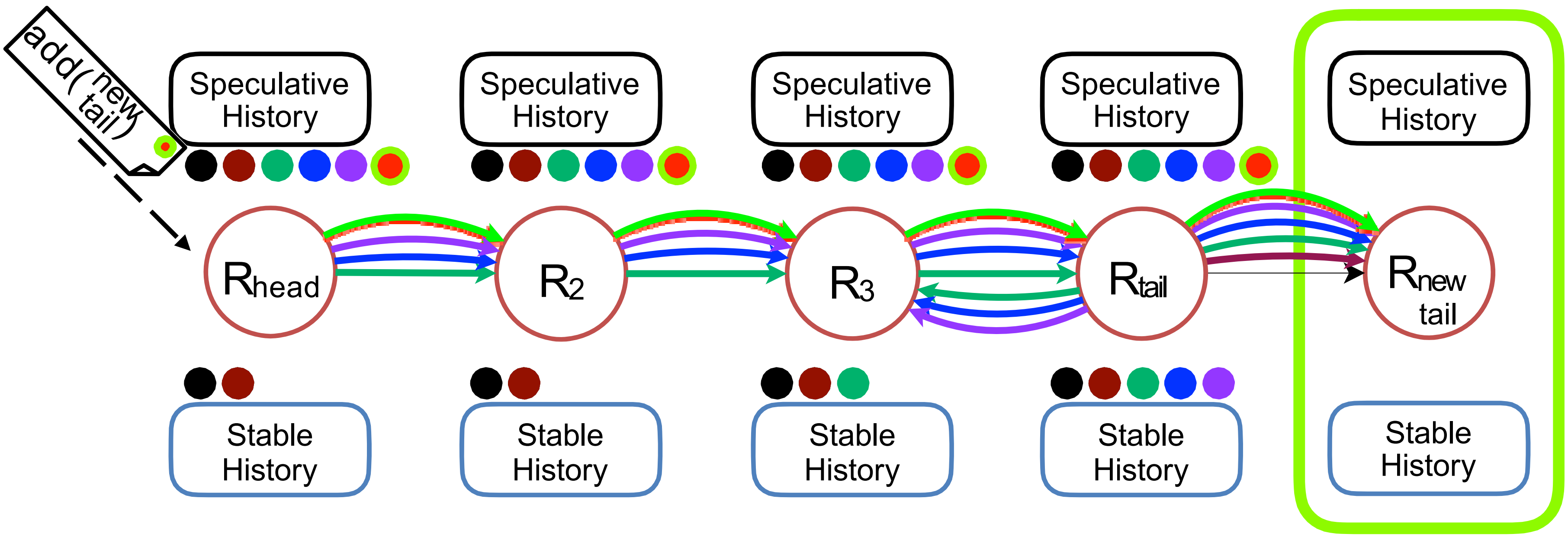
Relationship Among Histories



- **Old tail** should not answer to queries when the new tail does

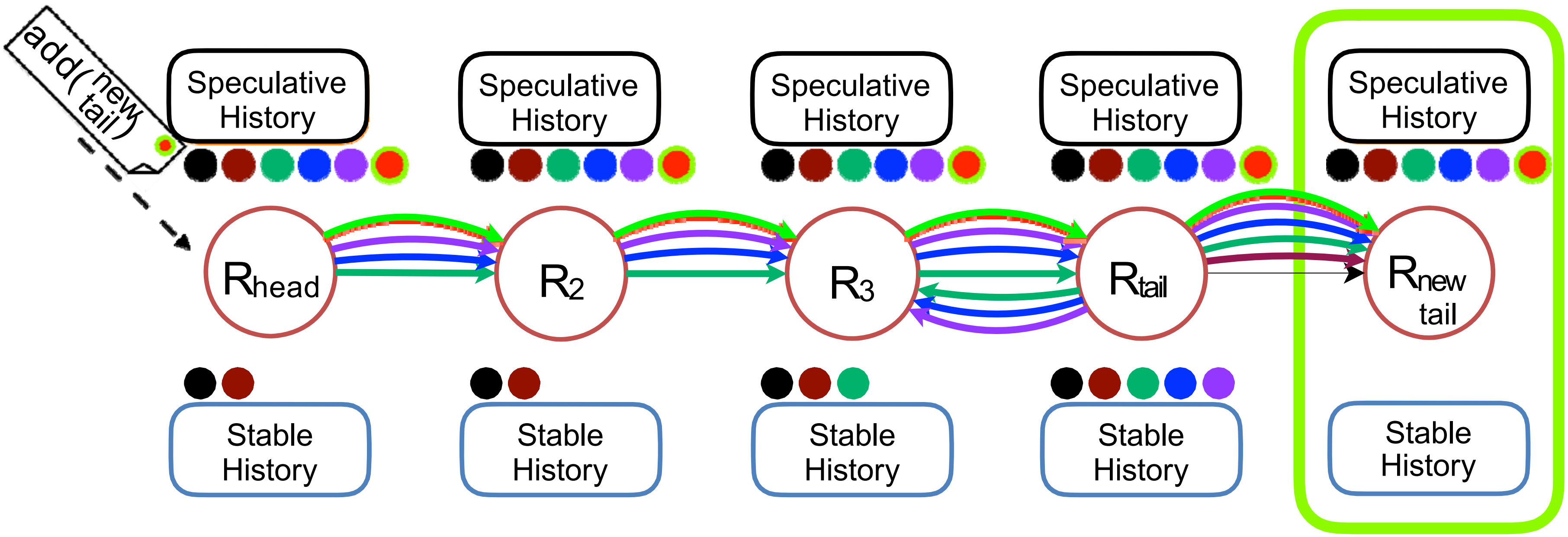
Adding a New Node

Flush History to New Tail



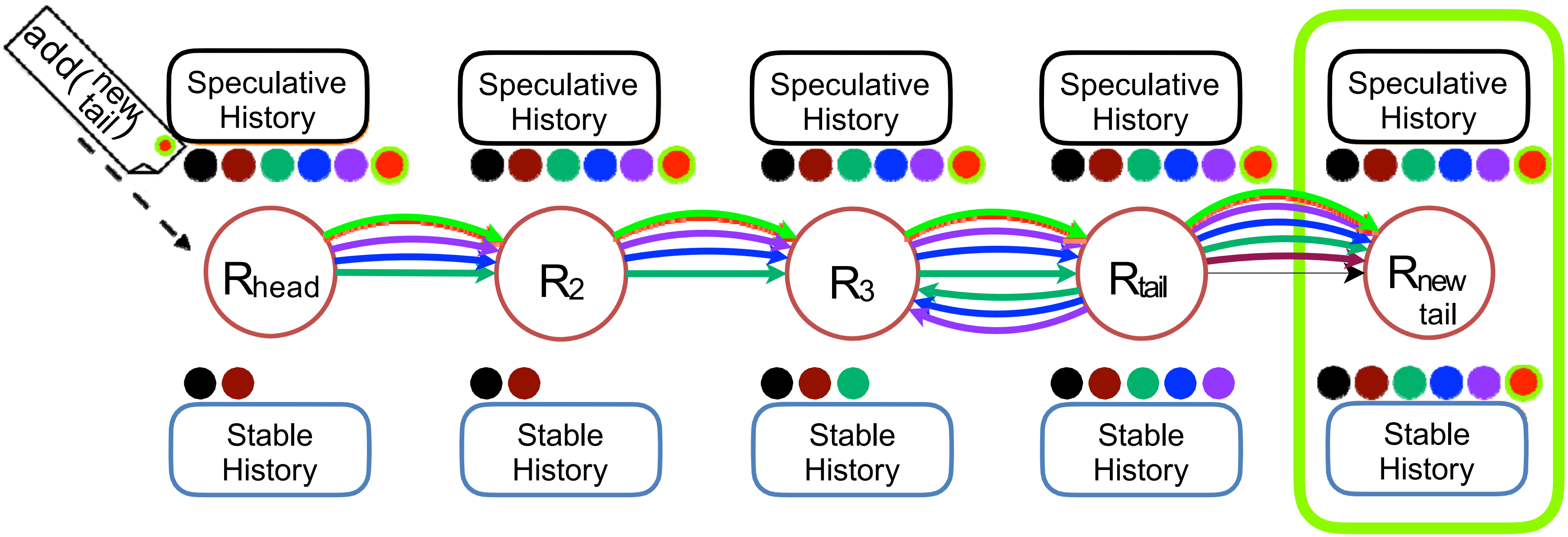
Adding a New Node

Flush History to New Tail



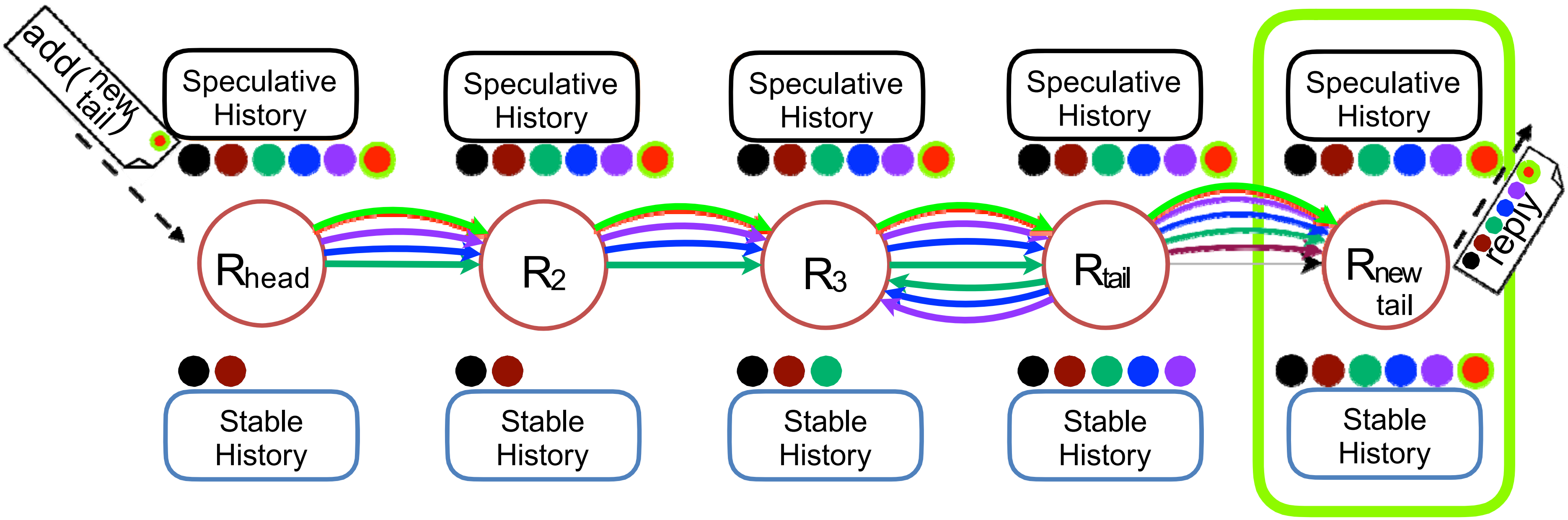
Adding a New Node

Copy Speculative onto Stable History



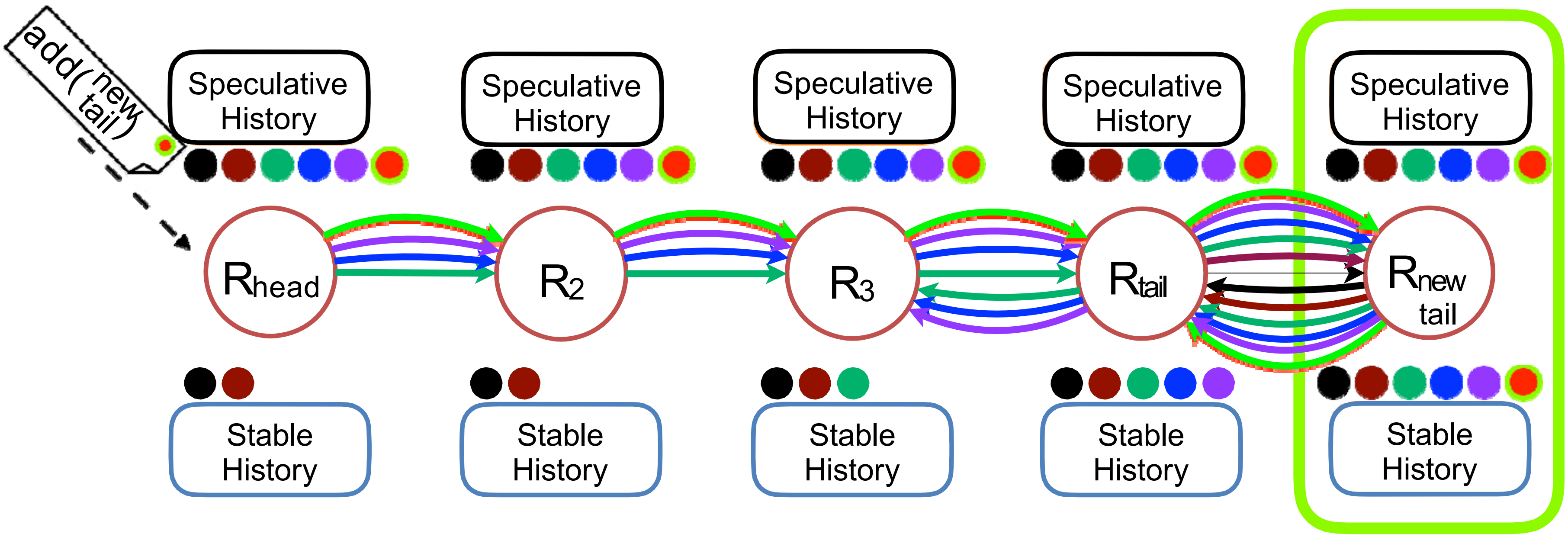
Adding a New Node

Respond to Queries and Acknowledge Updates



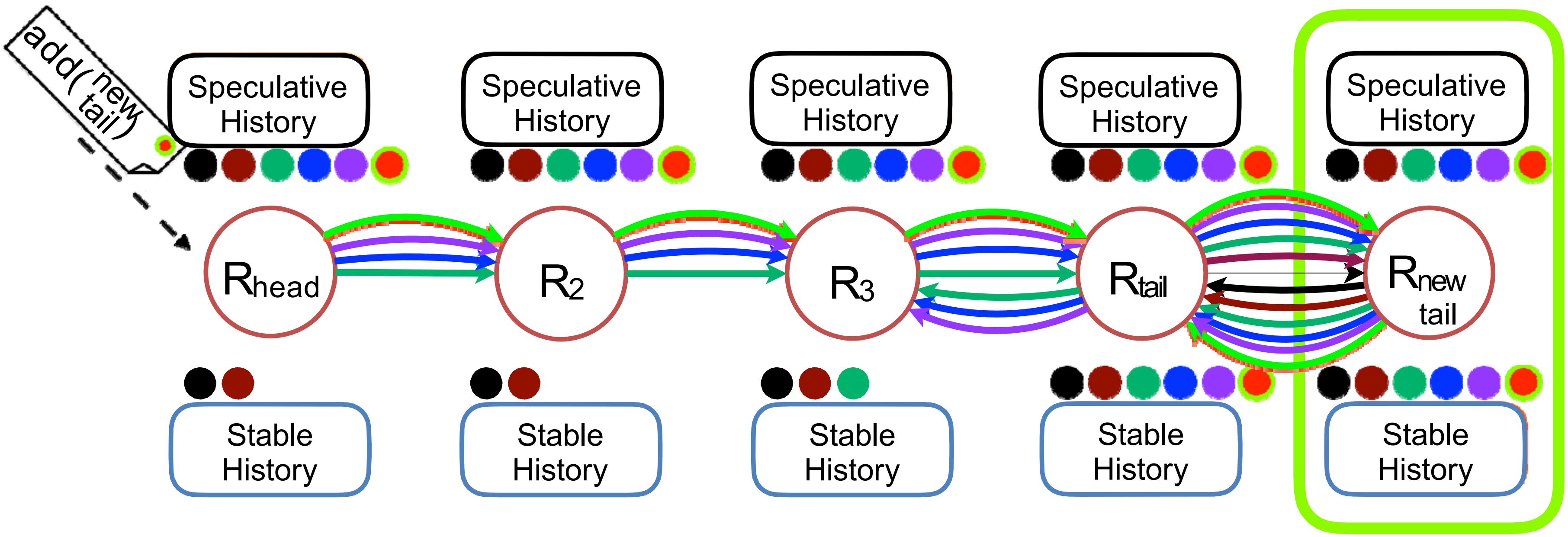
Adding a New Node I

Propagate Acknowledgements



Adding a New Node II

Propagate Acknowledgements



GOSSIPING

Gossiping protocols

- Disseminate information in **incremental manner**
  – Avoid overloading processes with heavy broadcast messages
  – Drawback is longer propagation time for information
- Each node maintains a **partial view** of other nodes
- During each gossip round, each node chooses **random** nodes from its view to exchange information with
  – Application data (e.g., current state)
  – Its partial view
- Nodes update their state and partial view based on the information received
- Gossiping happens **periodically** and **non-deterministically**
- Used in Cassandra for propagating status of each node and metadata

Lazy Replication Using Gossiping

- Replicas gossip about operations processed
- Replicas **reconcile** (compare) their operation logs and each apply any operations not yet seen
- Former step is highly application dependent
- Assumes updates can be applied in any order
- If system processes no more operations. then each replica **eventually** converges to the same state by gossiping enough times