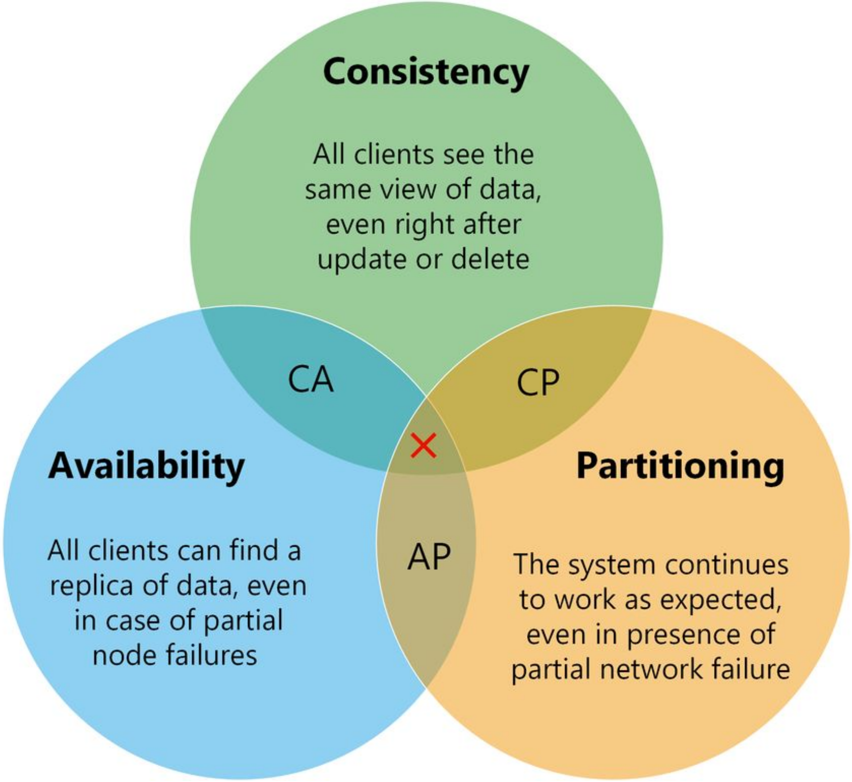
# State Machine Replication

Jacqueline Wen

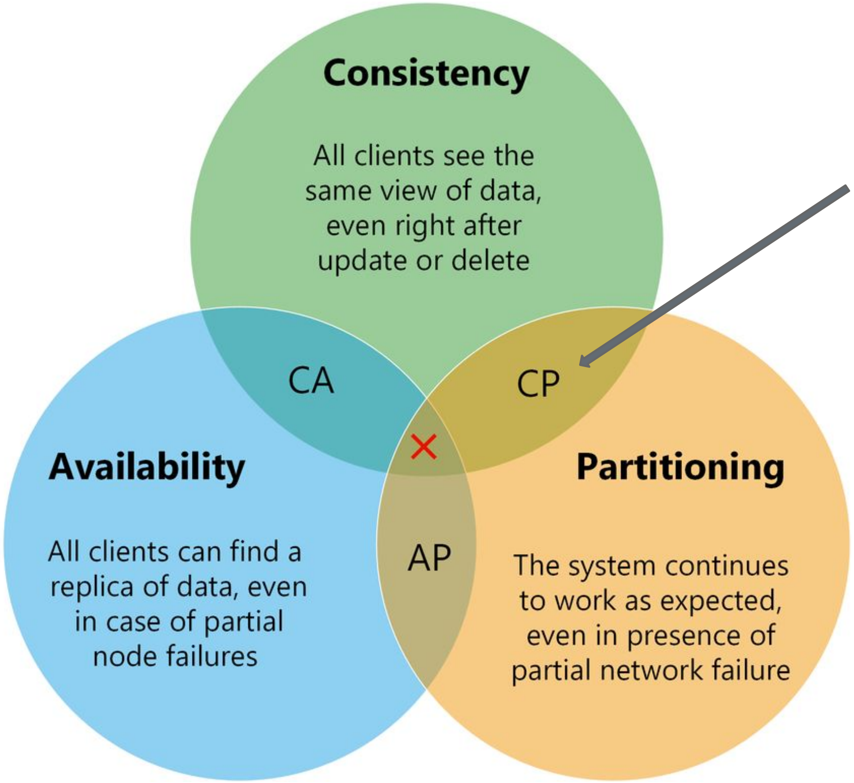
"A distributed system is one in which the failure of a computer didn't even know existed can render your own computer unusable"

–Leslie Lamport

# CAP Theorem


Consistency
All clients see the same view of data, even right after update or delete
CA
CP
Availability
All clients can find a replica of data, even in case of partial node failures
AP
Partitioning
The system continues to work as expected, even in presence of partial network failure

# CAP Theorem

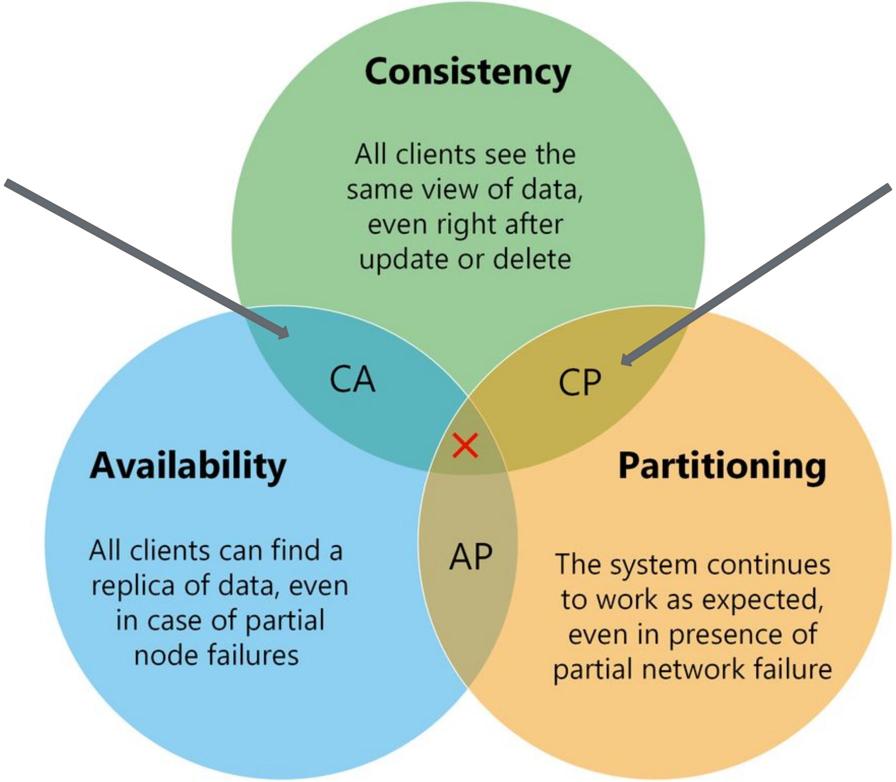



Can guarantee consistency

# CAP Theorem

Best effort… without partitions

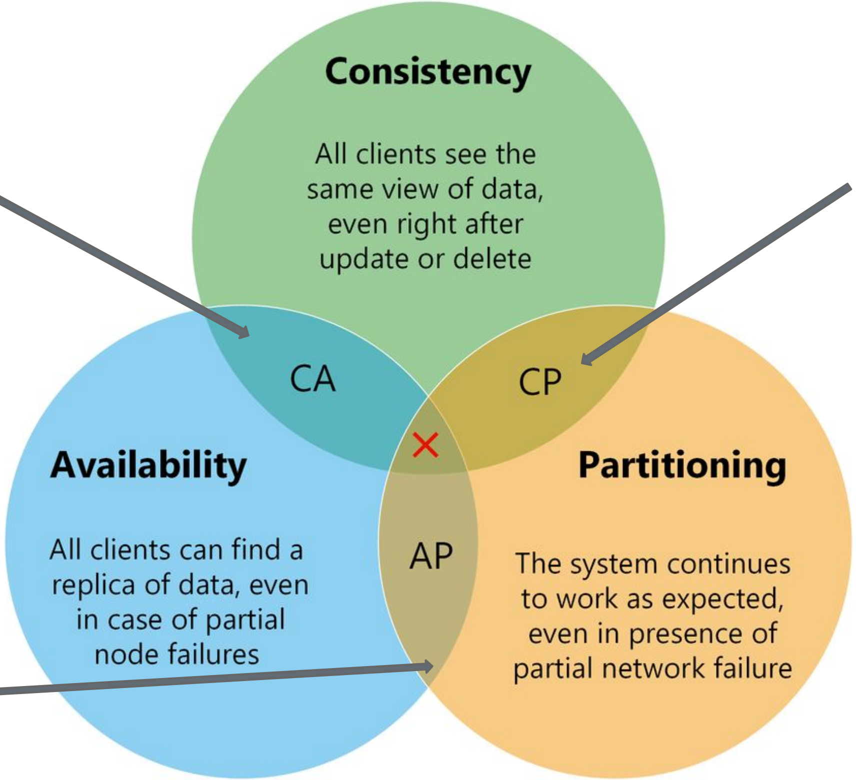
Can guarantee consistency



# CAP Theorem

Best effort… without partitions

Can guarantee consistency




Only availability on your side of partition…

# Timeline of papers

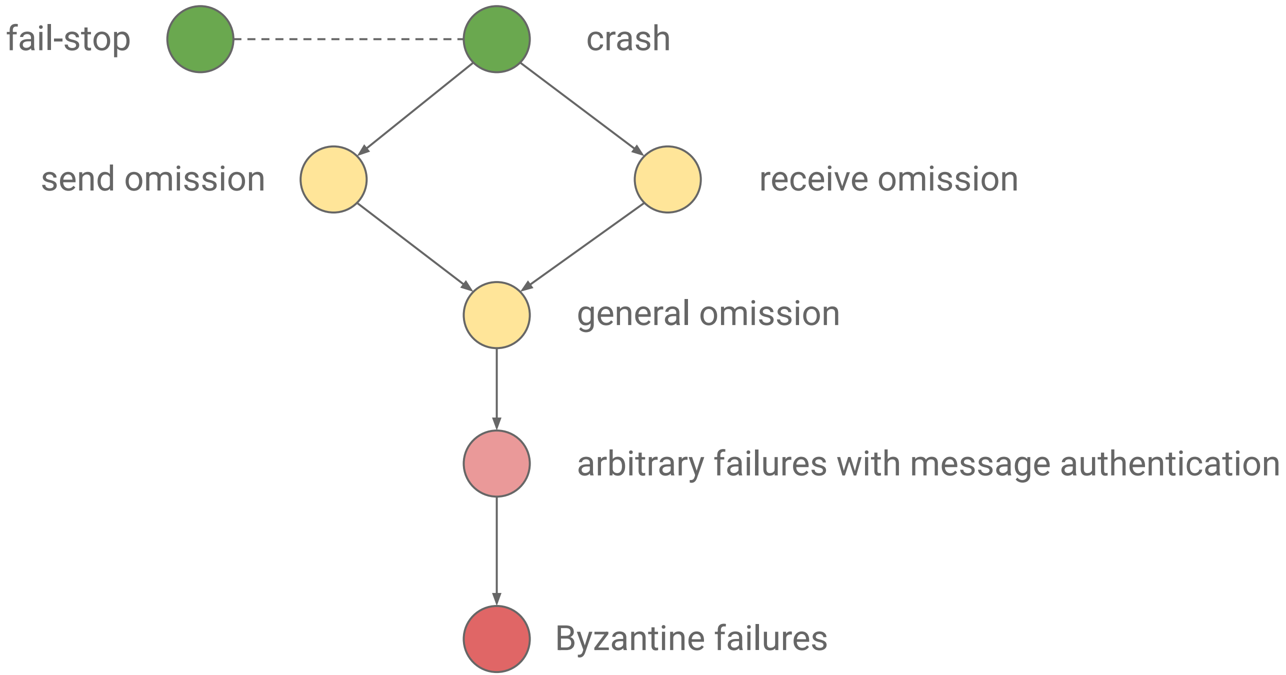
- “Time, clocks, and the ordering of events in a distributed system” (Leslie Lamport, 1978)
- “The Byzantine Generals Problem” (Leslie Lamport, 1984)
- “Implementing fault-tolerant services using the state machine approach: A Tutorial” (Fred Schneider, 1990)
- “The Part-Time Parliament” (Leslie Lamport, 1998[?])
- “Chain replication for supporting high throughput and availability” (Robbert van Renesse + Fred Schneider, 2004)

# Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial

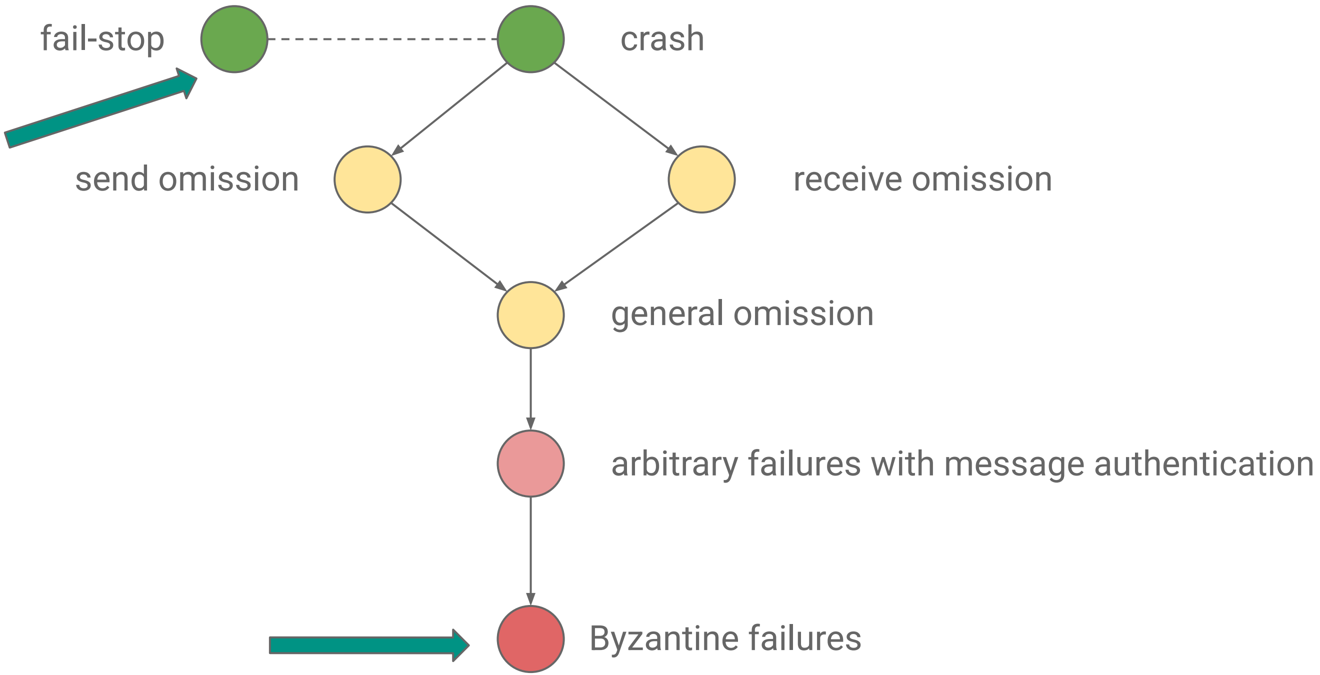
# Fred Schneider

Gates Hall 422

# Failure modes

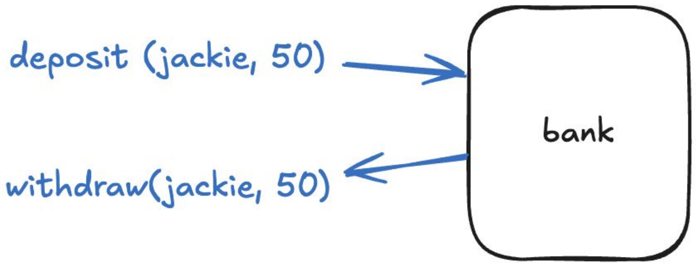


# Failure modes

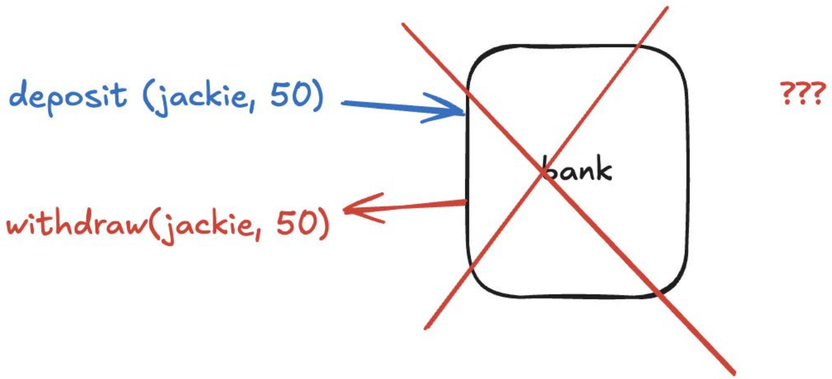



cr. 5414 slides

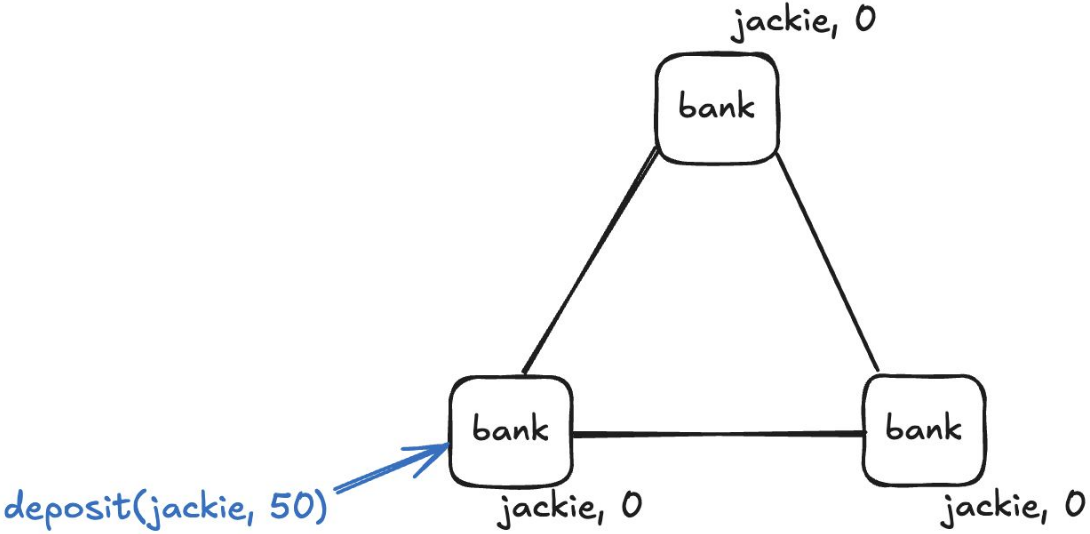
# Bank example: single server



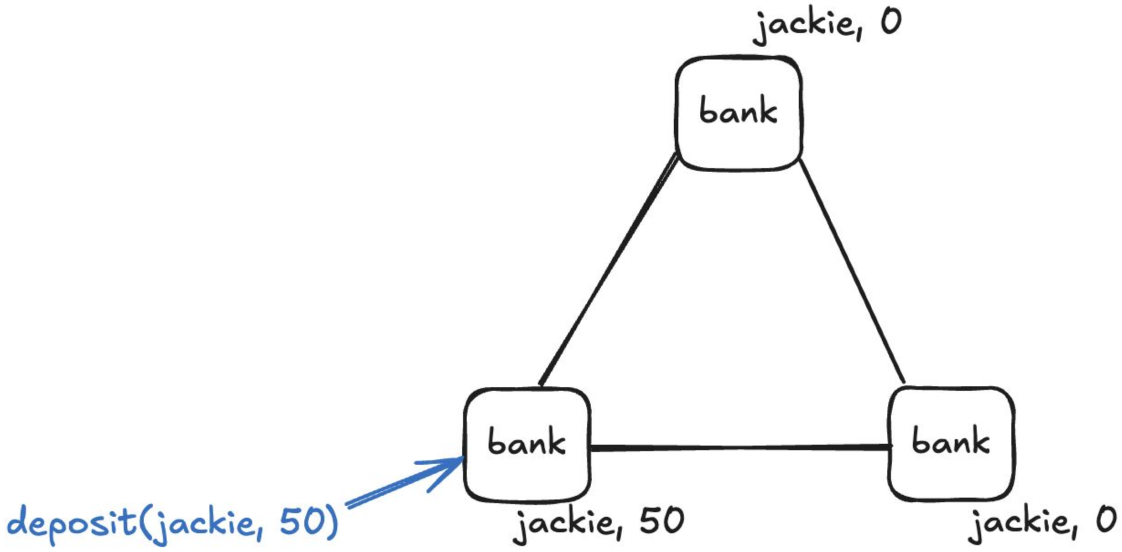
# Bank example: single server



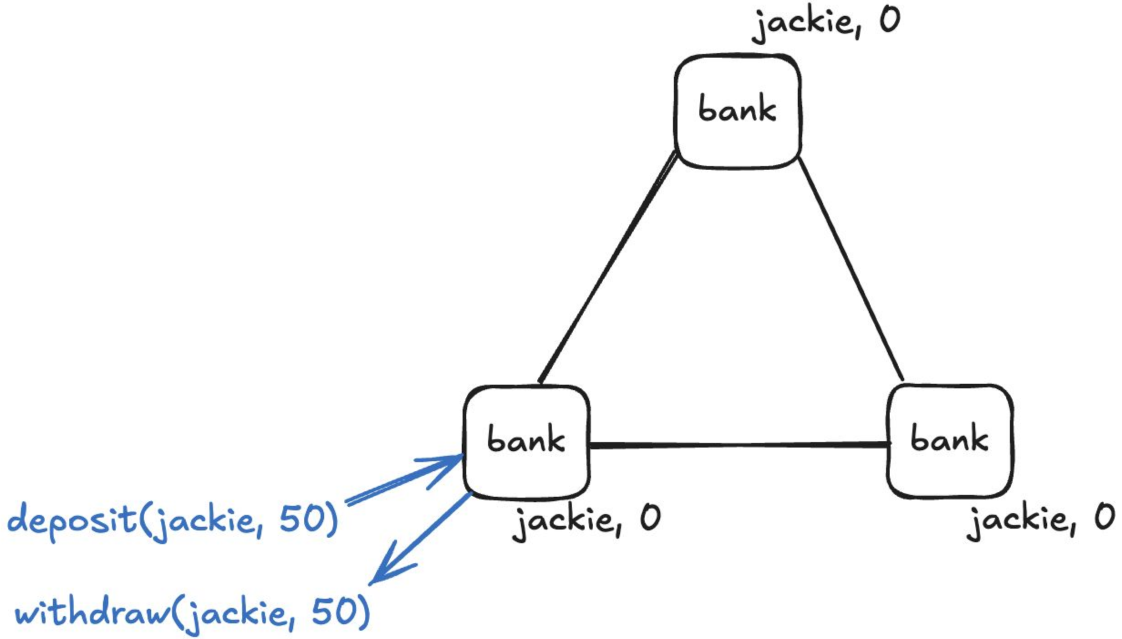
# Bank example: multiple servers



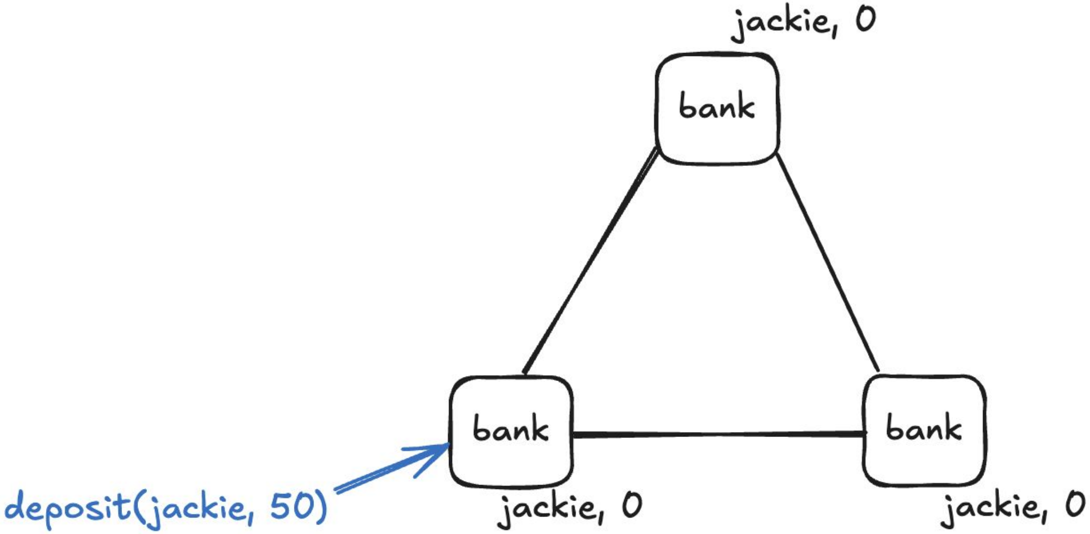
# Bank example: multiple servers



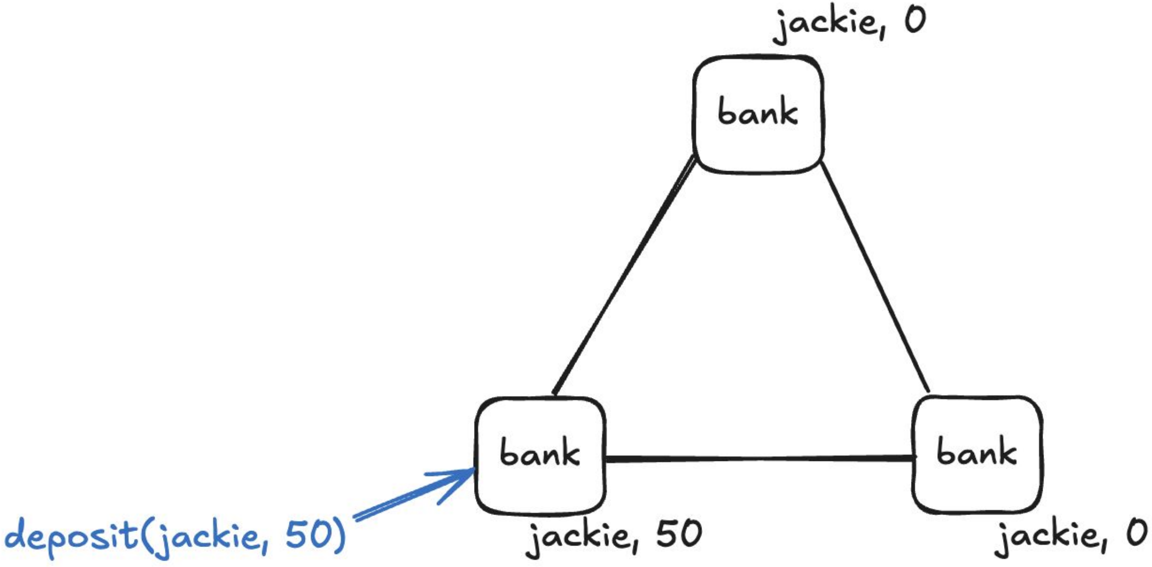
# Bank example: multiple servers



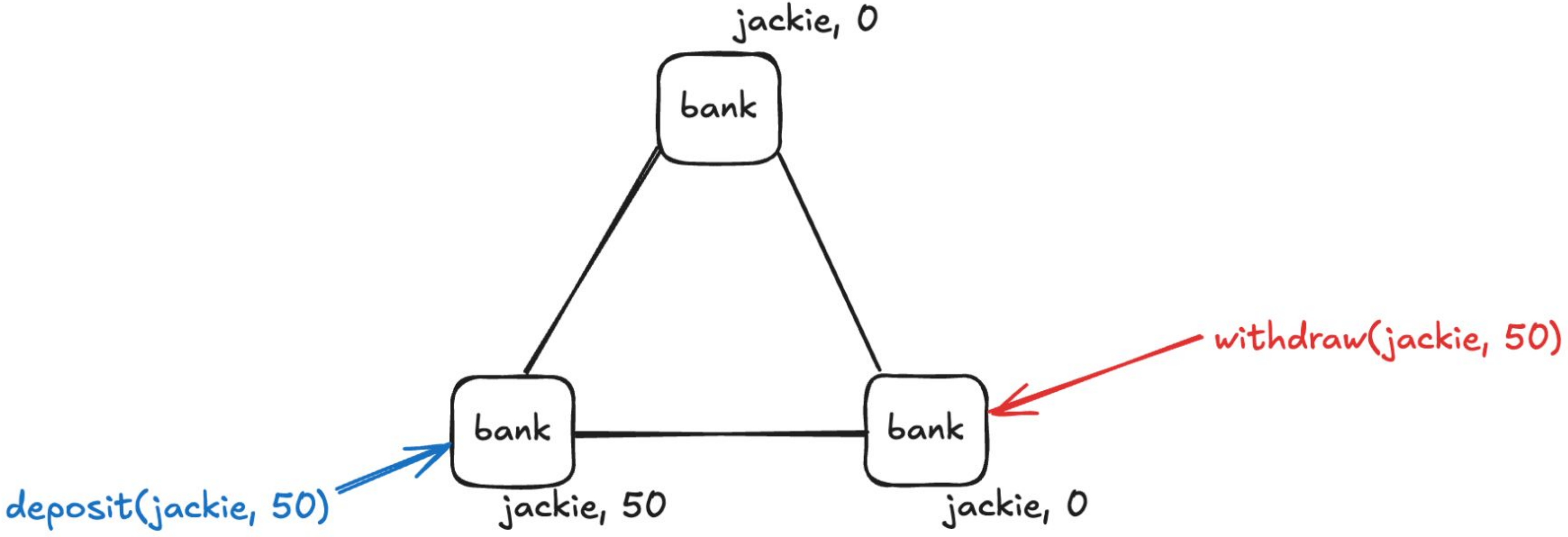
# Bank example: multiple servers



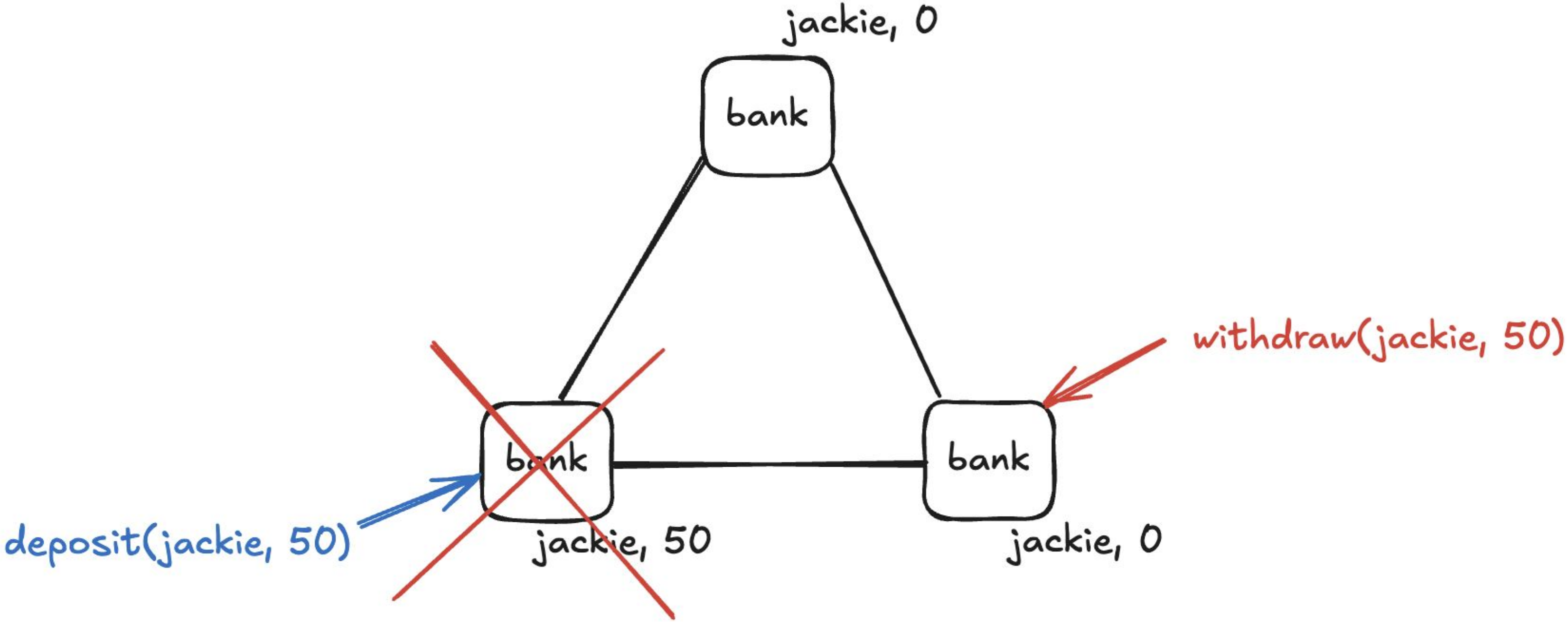
# Bank example: multiple servers



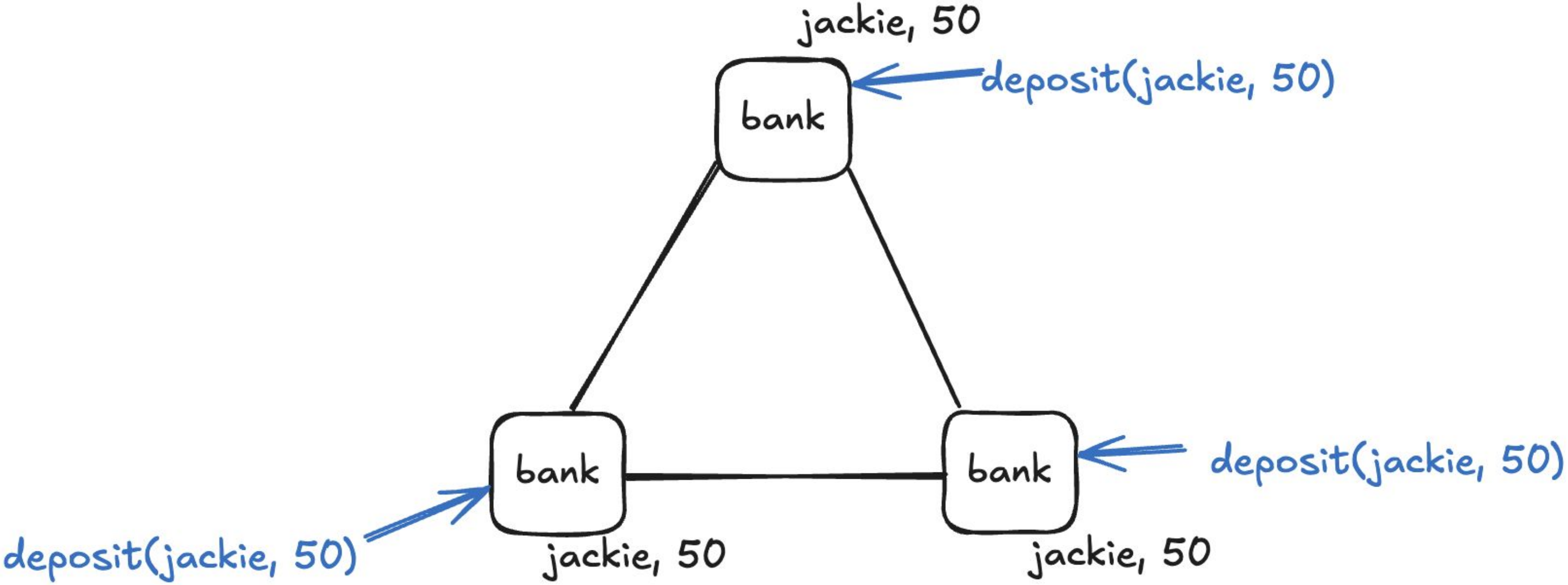
# Bank example: multiple servers



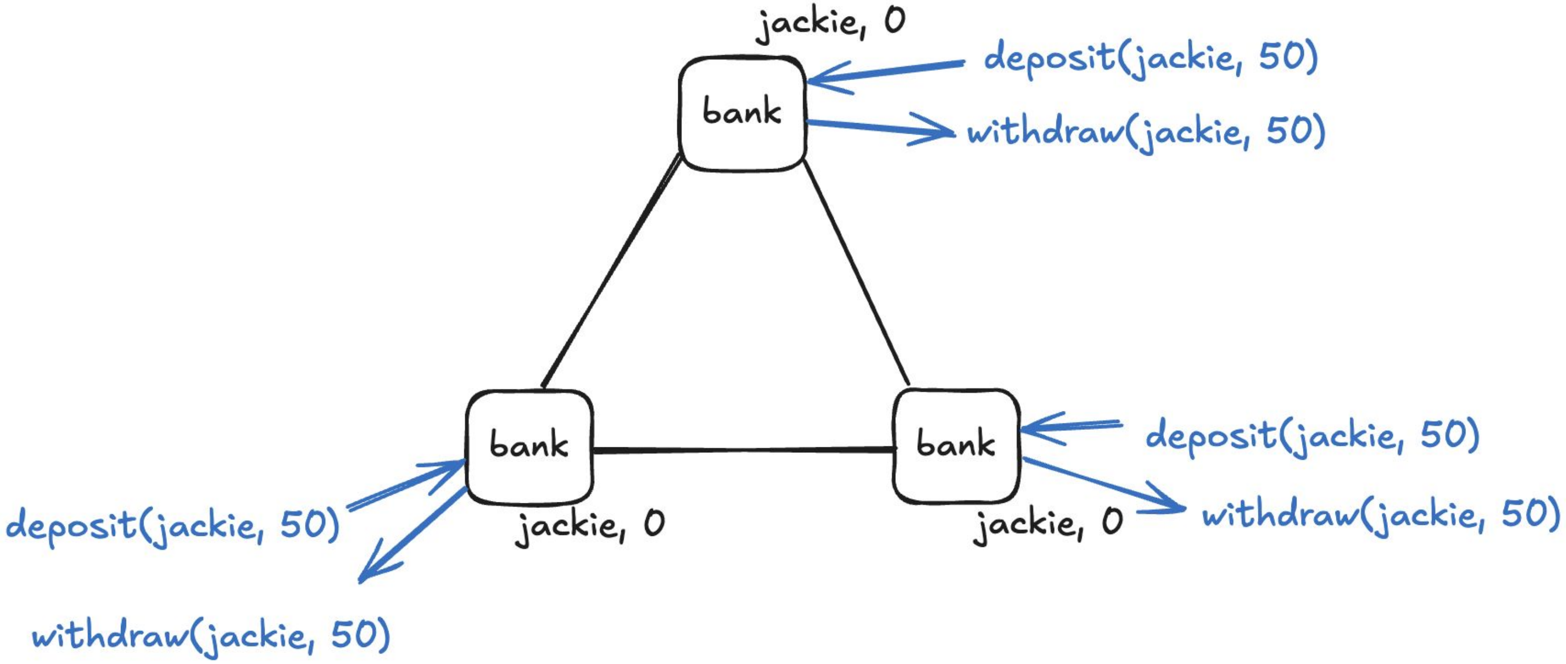
# Bank example: multiple servers



# Bank example: multiple servers (ideally)



# Bank example: multiple servers (ideally)



# Observation

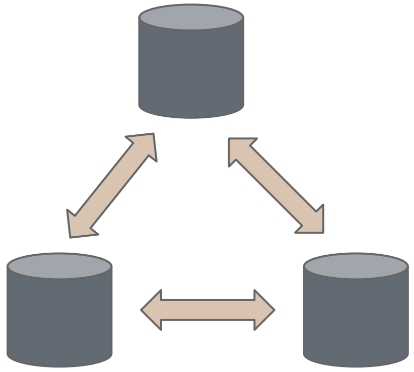
- A replica is just a FSM
  - Ex. if deposit 50, add 50 to balance
  - Ex. if withdraw 50, subtract 50 from balance
- Replicas have deterministic transitions
- → if we have the same transactions in FSM, by definition we will have same result
  - Consensus!

# Discussion

- What would happen if replicas weren't deterministic?
- What are the limitations of using FSMs for modeling replicas, especially in systems with infinite or highly dynamic states?

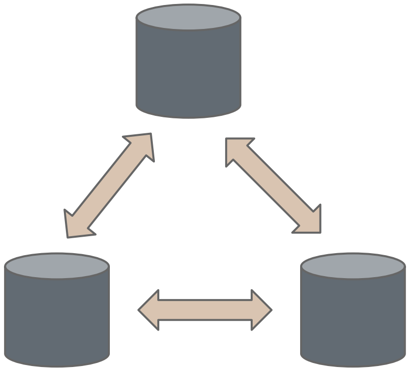
# State machine approach

- REPLICATION!



# State machine replication (SMR) approach

- REPLICATION!
- Replicas are coordinated
  1. **Agreement:** every non faulty state machine replica receives every request
  2. **Order:** every non faulty state machine replica processes its requests in same relative order
  - Each server runs same deterministic state machine, executing same sequence of requests
  - Failures masked



# Replica coordination: agreement

- Any protocol that allows designated processor to disseminate a value to other processes such that
  - IR1 All non faulty processors agree on the same value
  - IR2 If the transmitter is non faulty, then all non faulty processors use its value as the one on which they agree

# Replica coordination: order

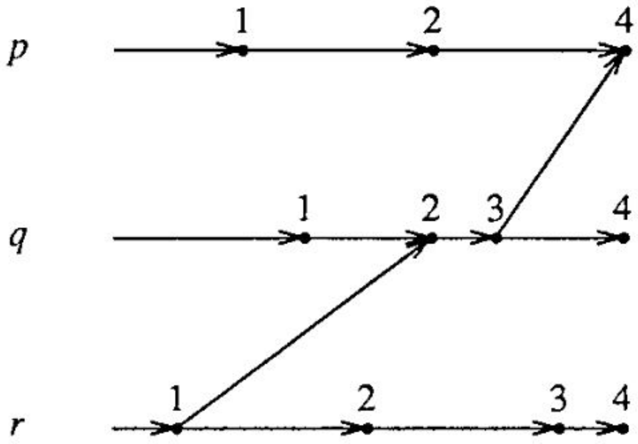
Implementation:

*"Replica next processes the stable request with smallest unique identifier"*

- O1: Requests issued by a single client to a given state machine *sm* are processed by *sm* in the order they were issued
- O2: If the fact that request *r* was made to a state machine *sm* by client *c* could have caused a request *r'* to be made by a client *c'* to *sm*, then *sm* processes *r* before *r'*

# Order: Lamport Clocks

- assume FIFO channels, fail stop failures



# Order: Synchronized Real-Time Clocks

- Assumes approximately synchronized clocks with known bounds on drift and message delay
- Each client tags request with real-time clock value as uid
- O1: clients can’t make > 1 requests on same clock tick
- O2: The clock synchronization bound δ must be less than the minimum message delivery time
  - If clocks are synchronized to within δ and message delay >δ, the timestamps respect causality

# Order: Replica Generated Identifiers

- Replicas themselves propose identifiers during agreement phase
  1. Each replica propose a candidate identifier cuid(sm_i, r) for request r
  2. One candidate is selected as the final uid(r)

UID1: $cuid(sm_i, r) \le uid(r)$.

UID2: If a request $r'$ is seen by replica $sm_i$ after $r$ has been accepted by $sm_i$ then $uid(r) < cuid(sm_i, r')$.

# Discussion

- The paper separates agreement and order. Why is this separation useful?
- The paper notes that order can sometimes be relaxed when requests commute. Any real-world examples where requests may commute? What trade-offs come with exploiting commutativity?
- Why is assigning unique identifiers to requests essential for ordering? What guarantees do these ids need to satisfy?
- The paper introduces ordering based on identifiers generated by the replicas themselves. How does this compare to client-assigned identifiers? What are the advantages and drawbacks?

# Handling outputs

- Ordering and agreement only ensure internal consistency
  - Make sure that outputs also remain correct even if devices fail
- Replicate output devices if outputs go to outside world
  - Each voter collects outputs from all state machine replicas
  - Environment effectively becomes final voter
- Let clients act as voters if outputs returned internally
  - Fail stop: client trusts first response it receives
  - Byzantine: wait for t+1 identical responses

# Tolerating faulty clients

- Replicate clients (with voting)
    - Requests buffered, corresponding commands run only once
- Defensive programming in state machines (restrict commands, add validity checks) so they can’t be corrupted by bad requests

# Reconfiguration

- Remove faulty components and add repaired ones without stopping the service
- Require mechanisms for updating configuration and synchronizing new components with system state

# Discussion

- Can you think of any examples/formats of SMR?
- What are the pros and cons of each of the ordering protocols?

# Chain Replication for Supporting High Throughput and Availability

# Robbert van Renesse + Fred Schneider

Gates Hall 433

Gates Hall 422

Chain replication is a way of implementing state machine replication!

# Strong consistency

**Reads see latest writes**

- **All accesses are seen by all servers in same order**
- **Only one consistent state can be observed**

# High throughput

**Queries look at tail of chain**

# High availability

**without partitions*

**System reconfigures on failures**

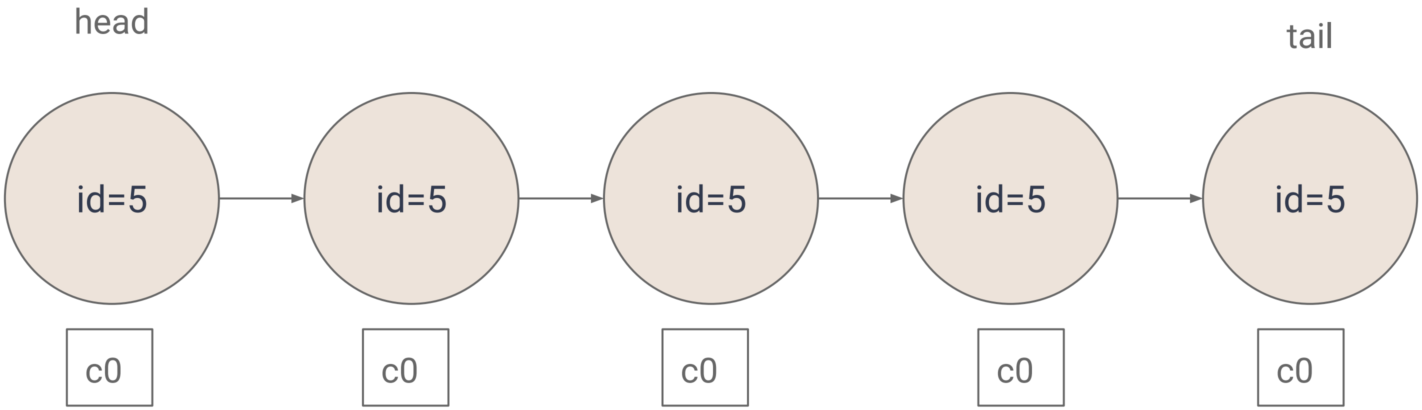
# Two request types

- query(id)
- update(id, val)

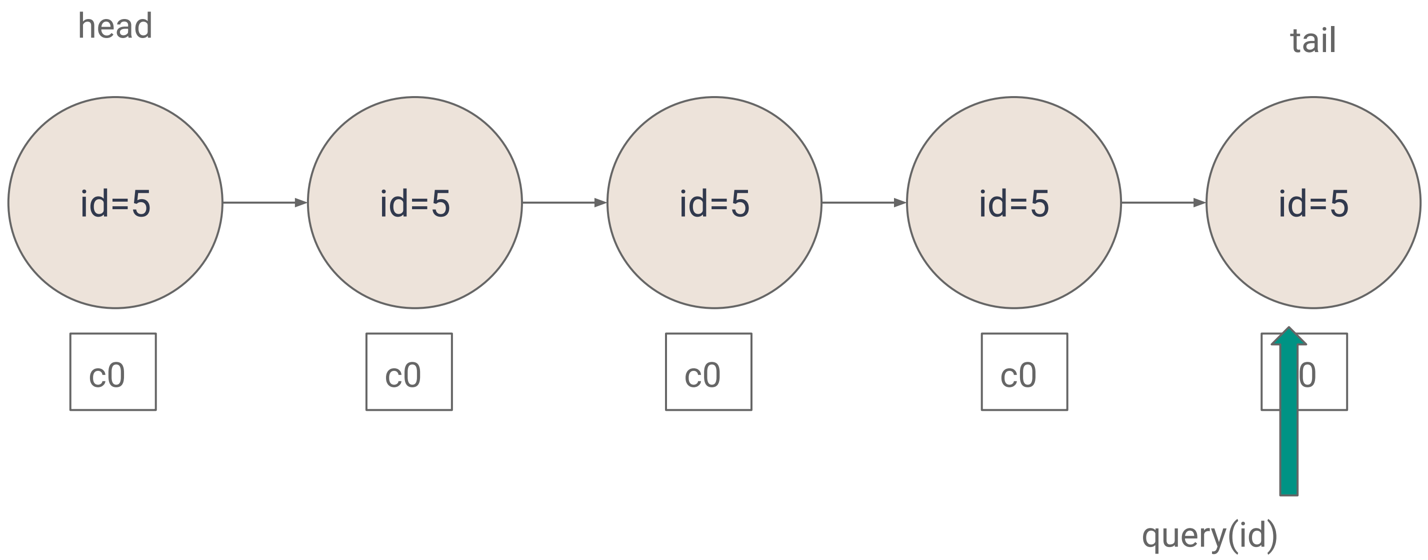
# Assumptions

- Failure method: fail stop
  - Can detect when server fails
- Reliable FIFO channel between servers

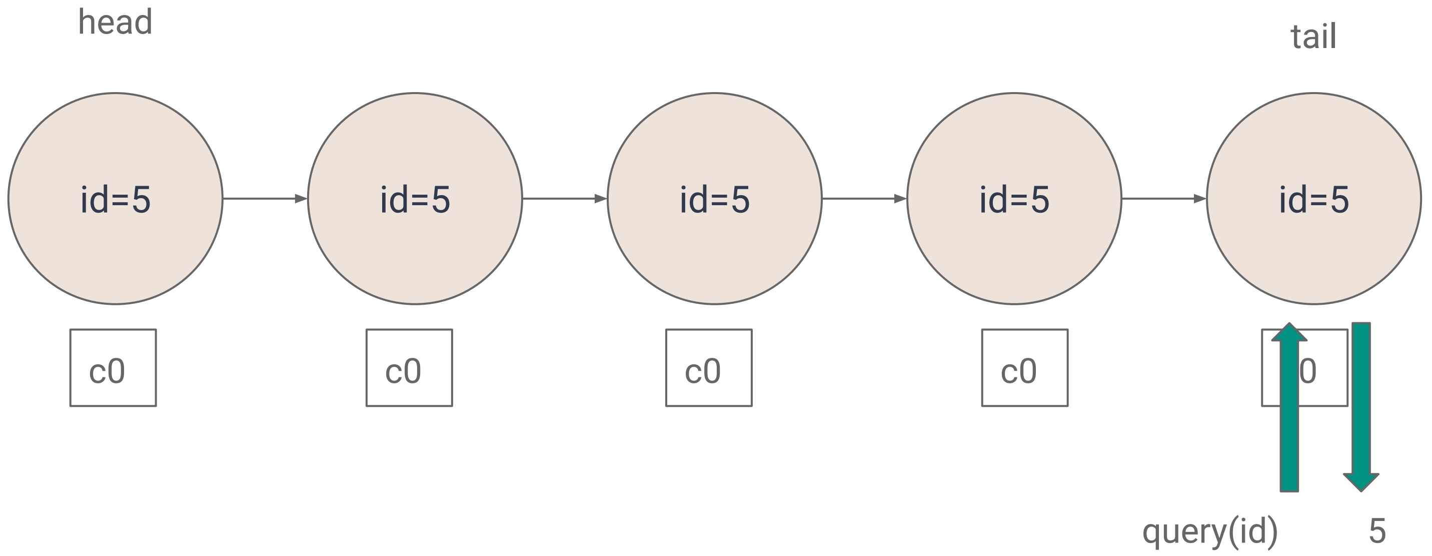
# query(id)



# query(id)

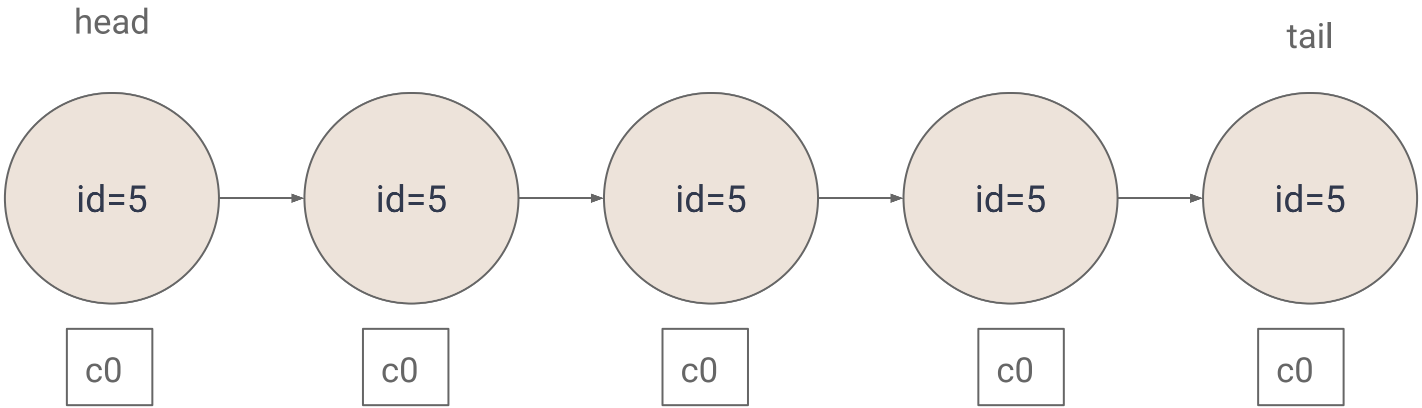


# query(id)

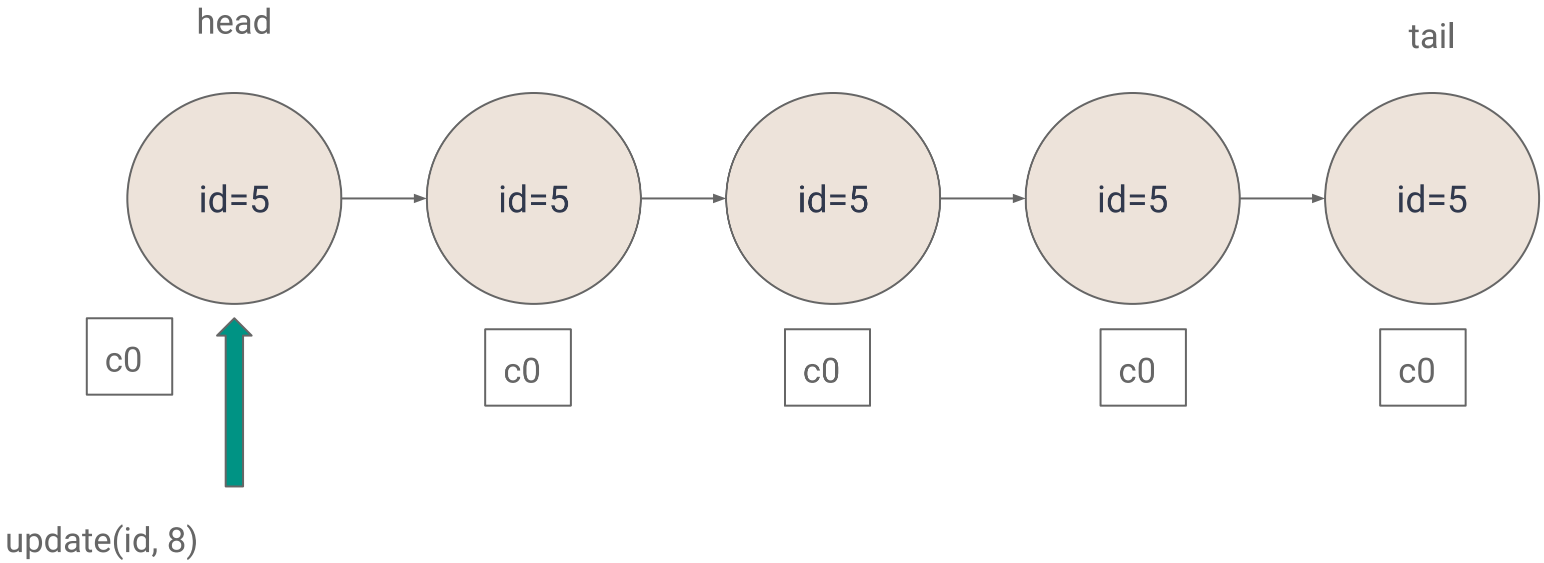


# update(id, val)

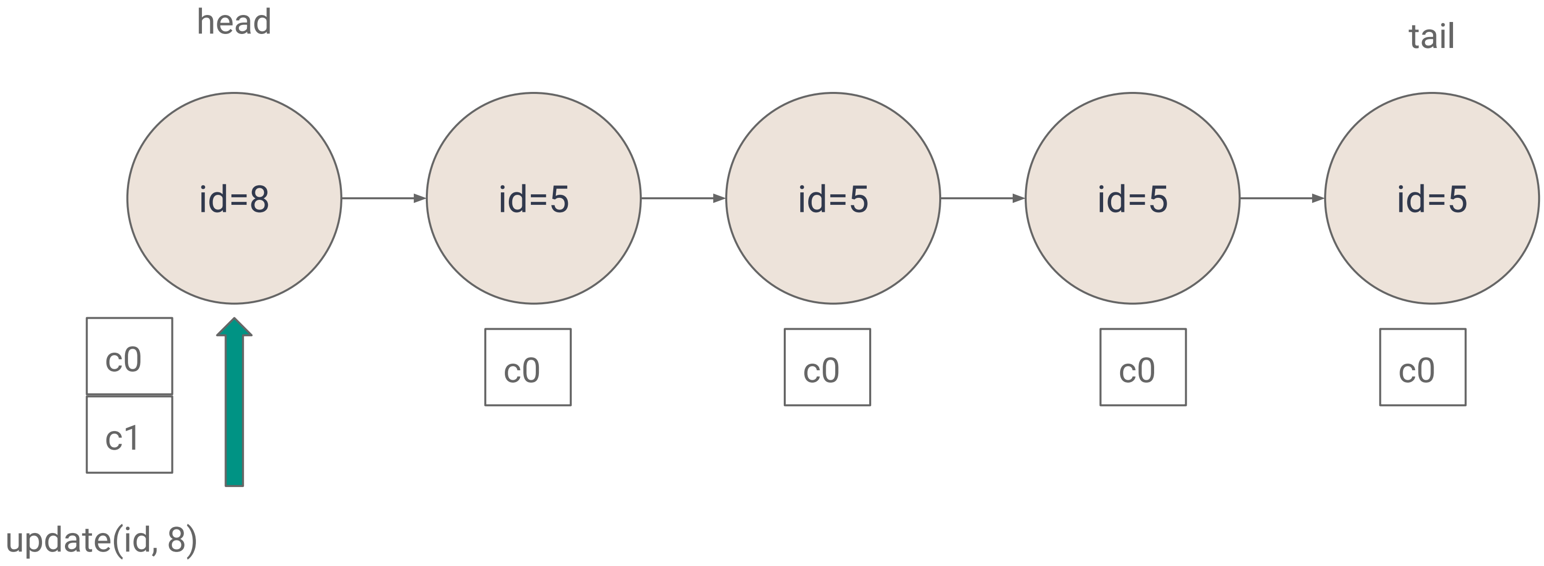
update(id, 8)



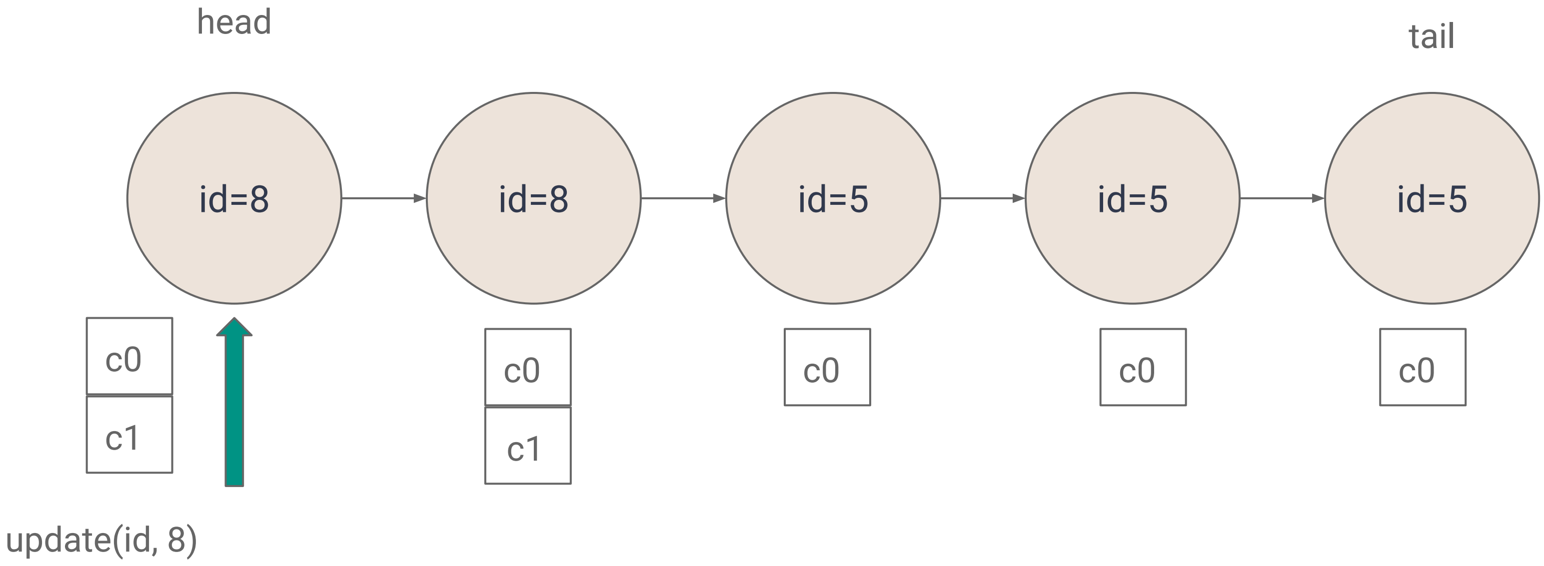
# update(id, val)



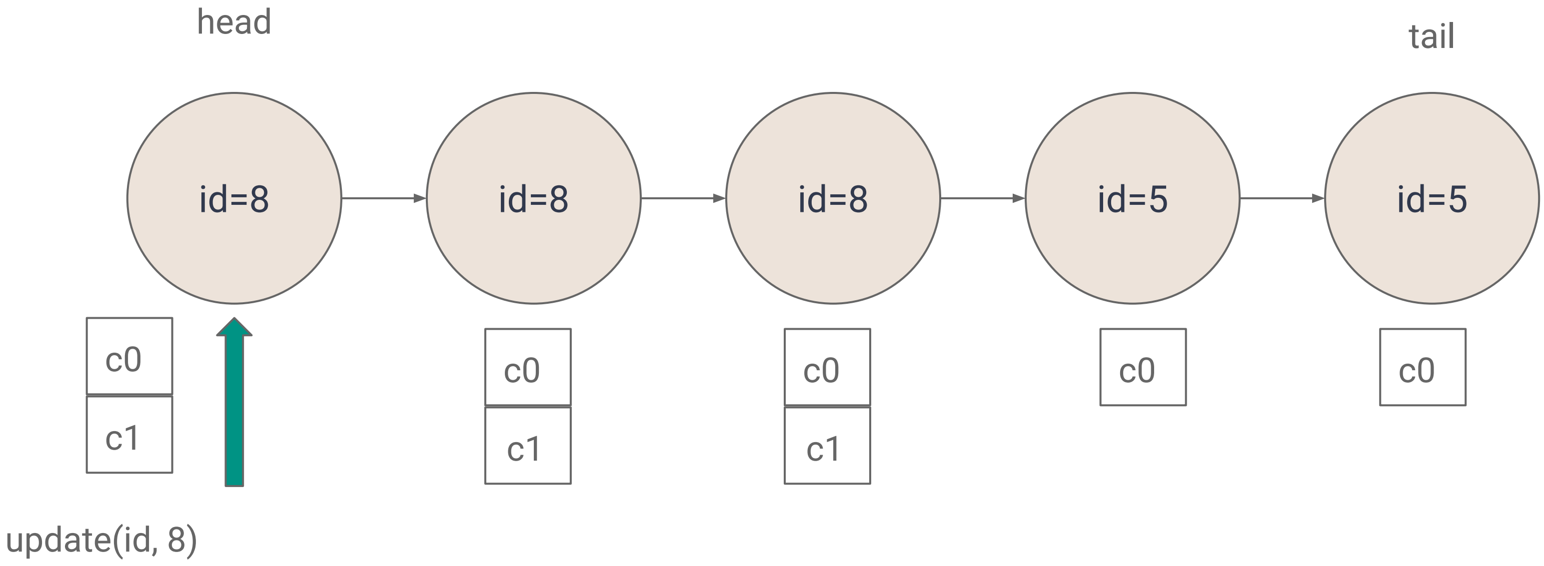
# update(id, val)



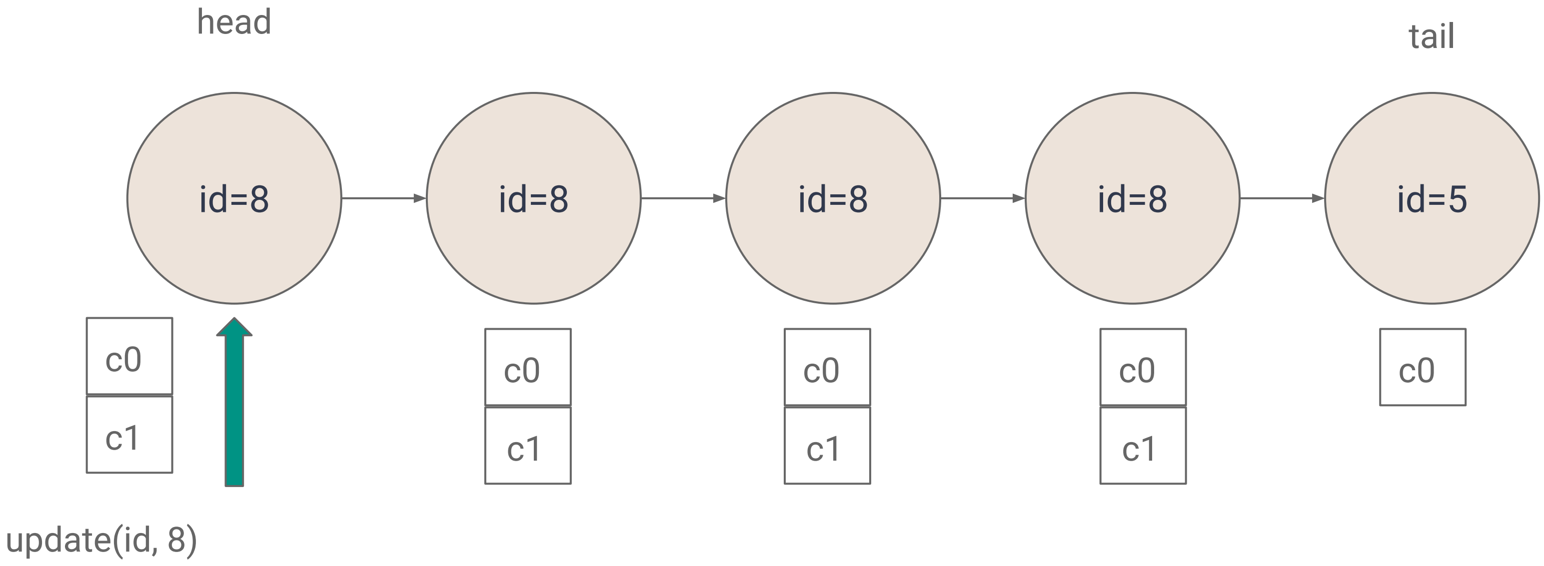
# update(id, val)



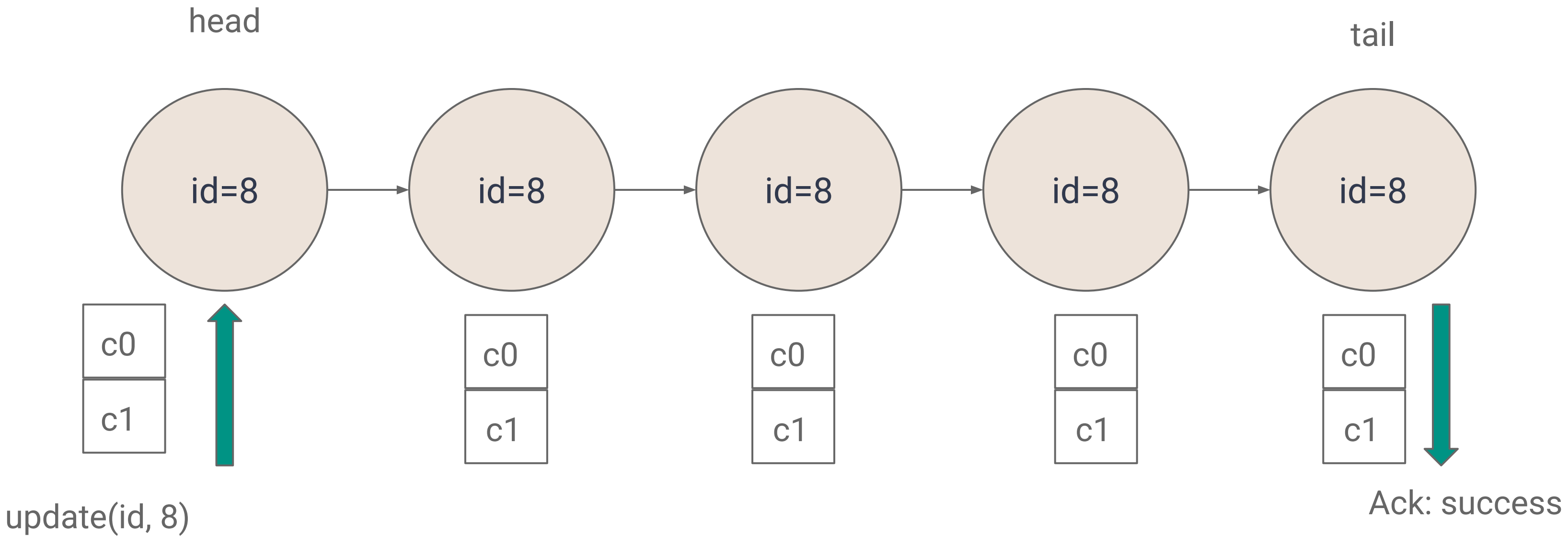
# update(id, val)



# update(id, val)



# update(id, val)



# Master service

- Detects server failures (max t failures)
- Informs server about new predecessor/successor (in new chain when server fails)
- Tells clients which server is head/tail of chain

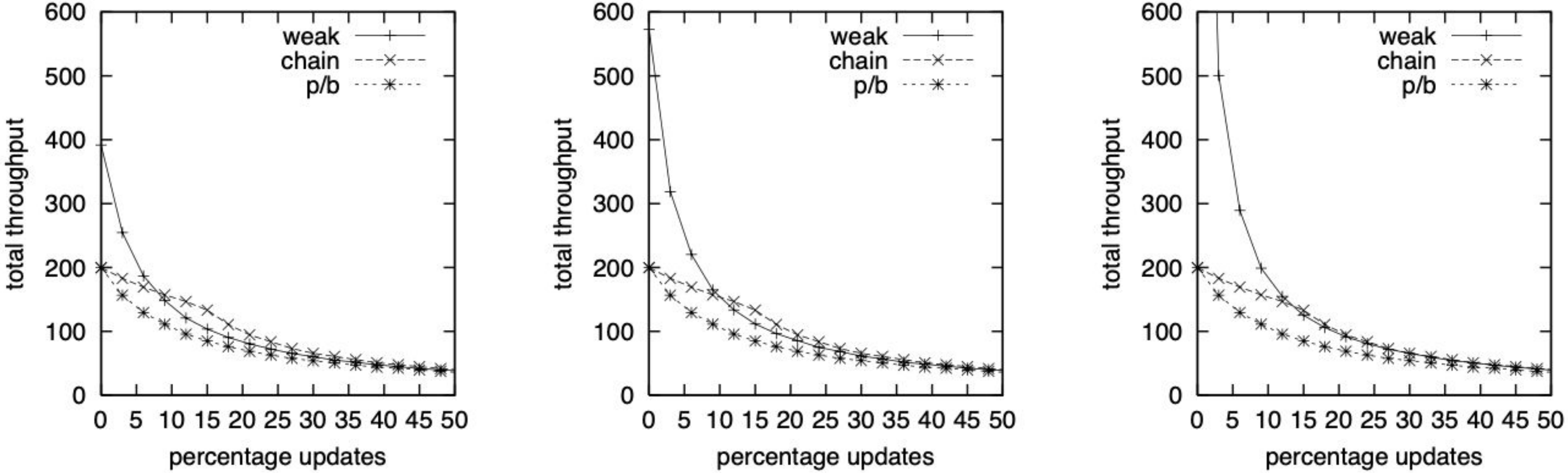
# Fault tolerance

- Head failure: second server in chain is new head
- Tail failure: predecessor of tail is new tail
- Middle failure: link around failed server in chain
- Extending chain: add new server to end of chain

# Metrics

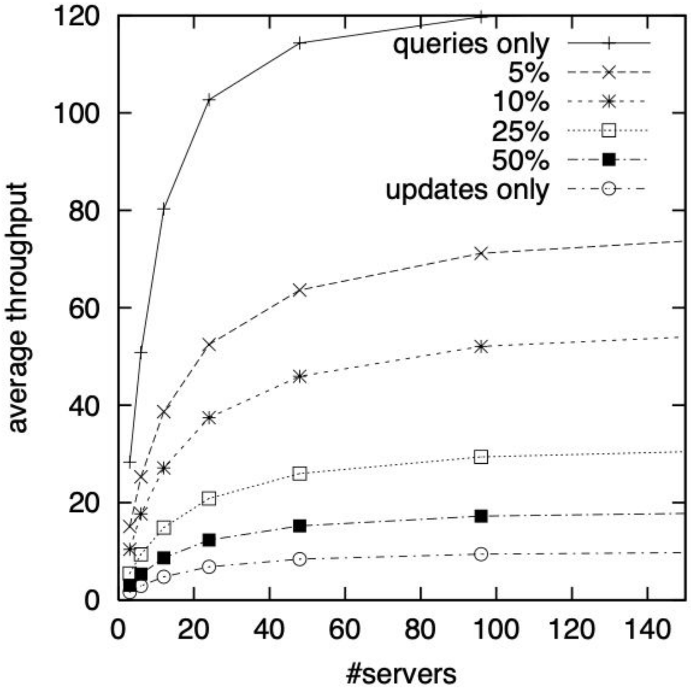
- Chain: chain replication
- p/b: primary backup
- weak-chain: chain replication modified so query request goes to any random server
- weak-p/b: primary backup modified so query request goes to any random server

# Single Chain, No Failures

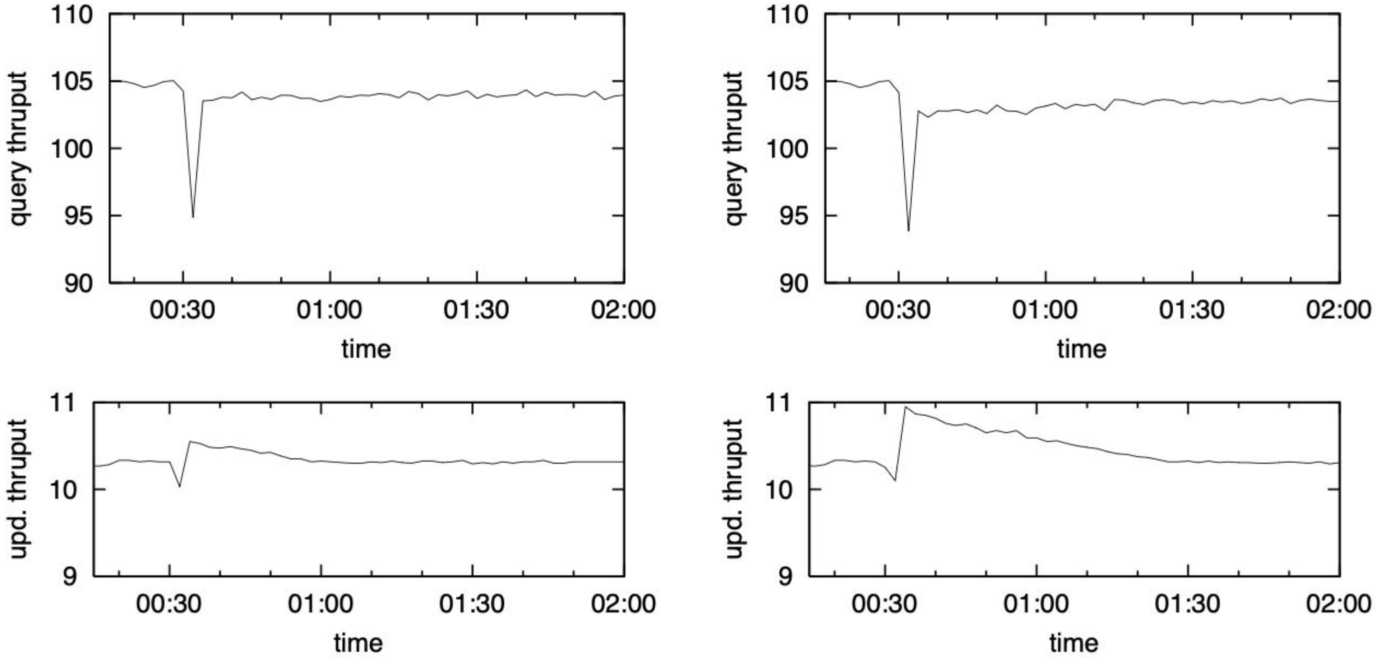



(a) $t = 2$ (b) $t = 3$ (c) $t = 10$

# Multiple chains, no failures



# Throughput with failures




(a) one failure (b) two failures

# Discussion

- Other than not supporting Byzantine failures, are there any other downsides of chain replication?
- Why is chain replication still widely used in industry?