Fault-Tolerant State Machine Replication

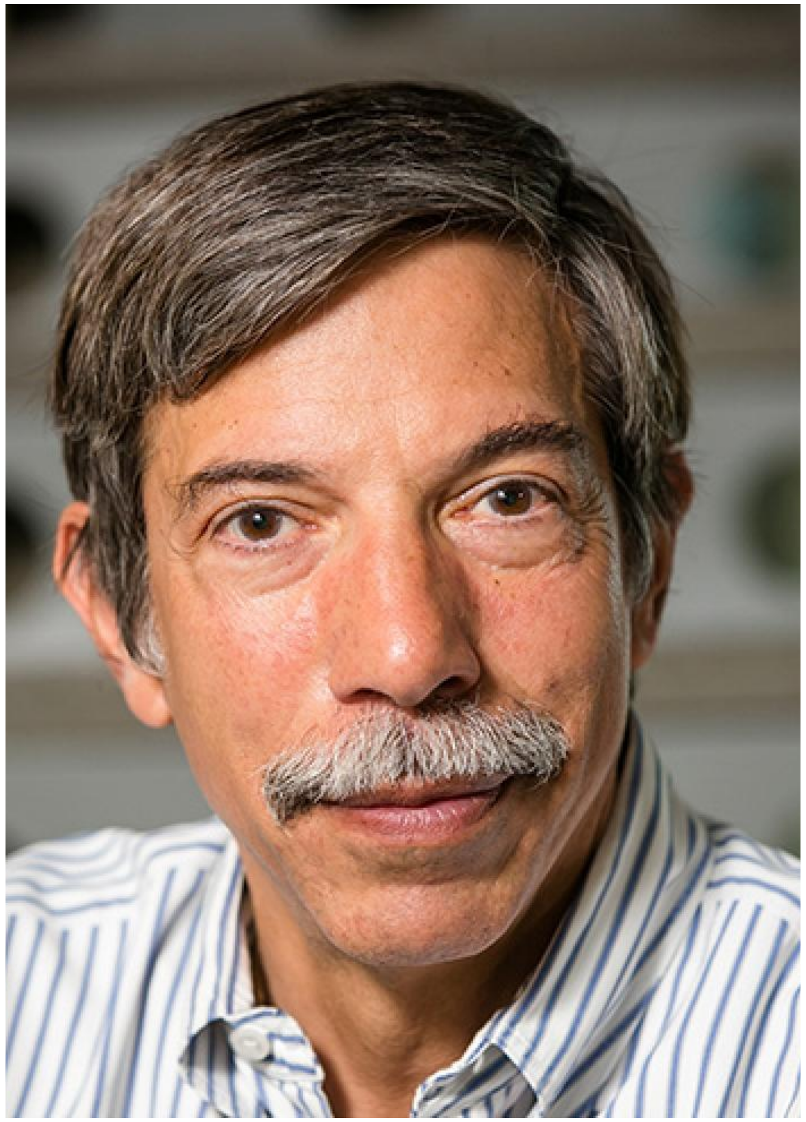
Chinasa T. Okolo

Slides borrowed from Hakim Weatherspoon and Drew Zagieboylo

Authors

Fred Schneider

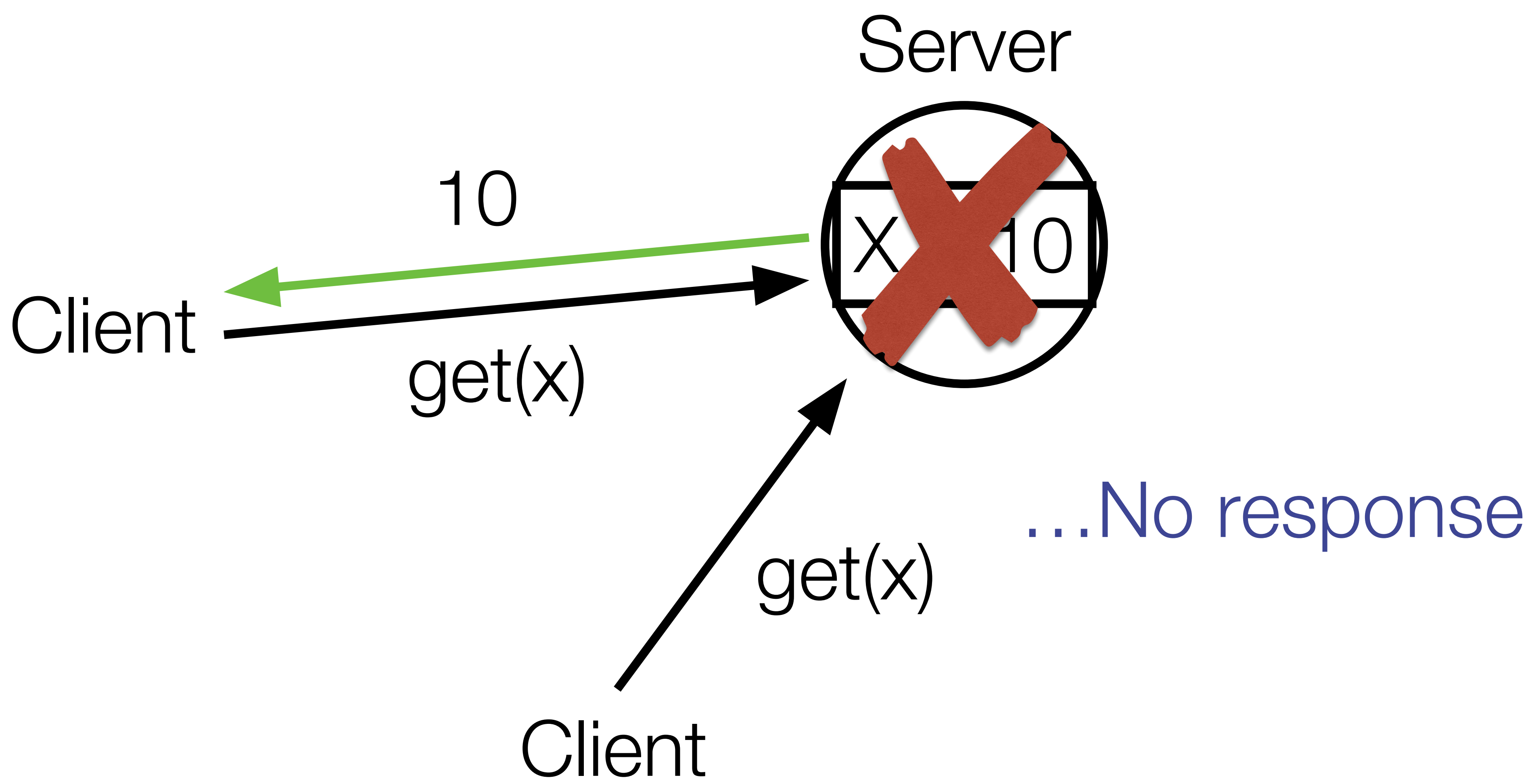
- Samuel B. Eckert Professor of Computer Science
- AAAS, ACM, and IEEE Fellow
- Concurrent and distributed systems for high-integrity and mission-critical settings



Outline

- Motivation
- State Machine Replication Approach
- Implementation
- Fault Tolerance
- Chain Replication
- Conclusions

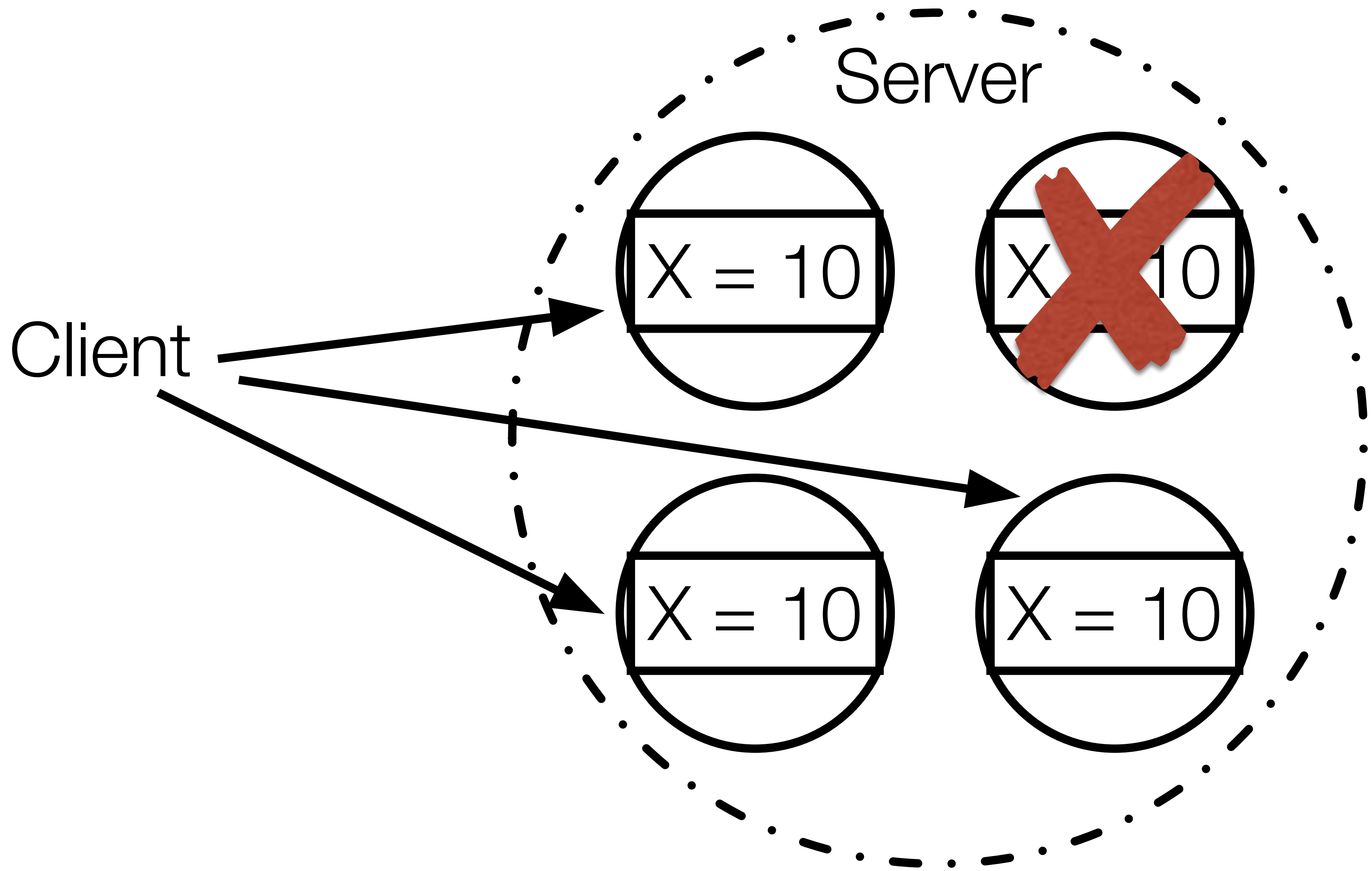
Motivation



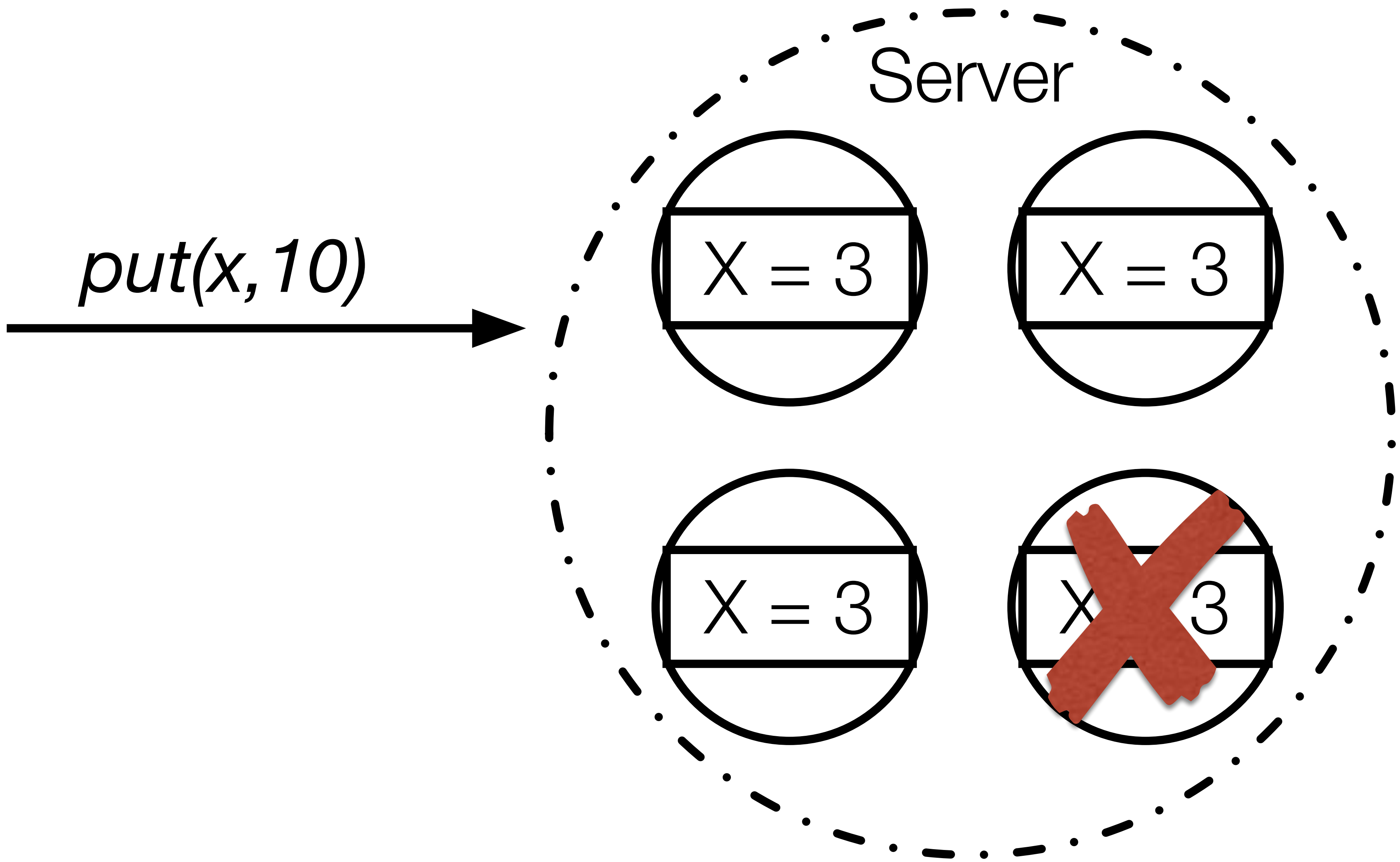
Motivation

- Need replication for fault tolerance
- **What happens in scenarios without replication?**
  - Storage - Disk Failure
  - Web service - Network failure
- **Be able to reason about failure tolerance**
  - How badly can things go wrong and have our system continue to function?

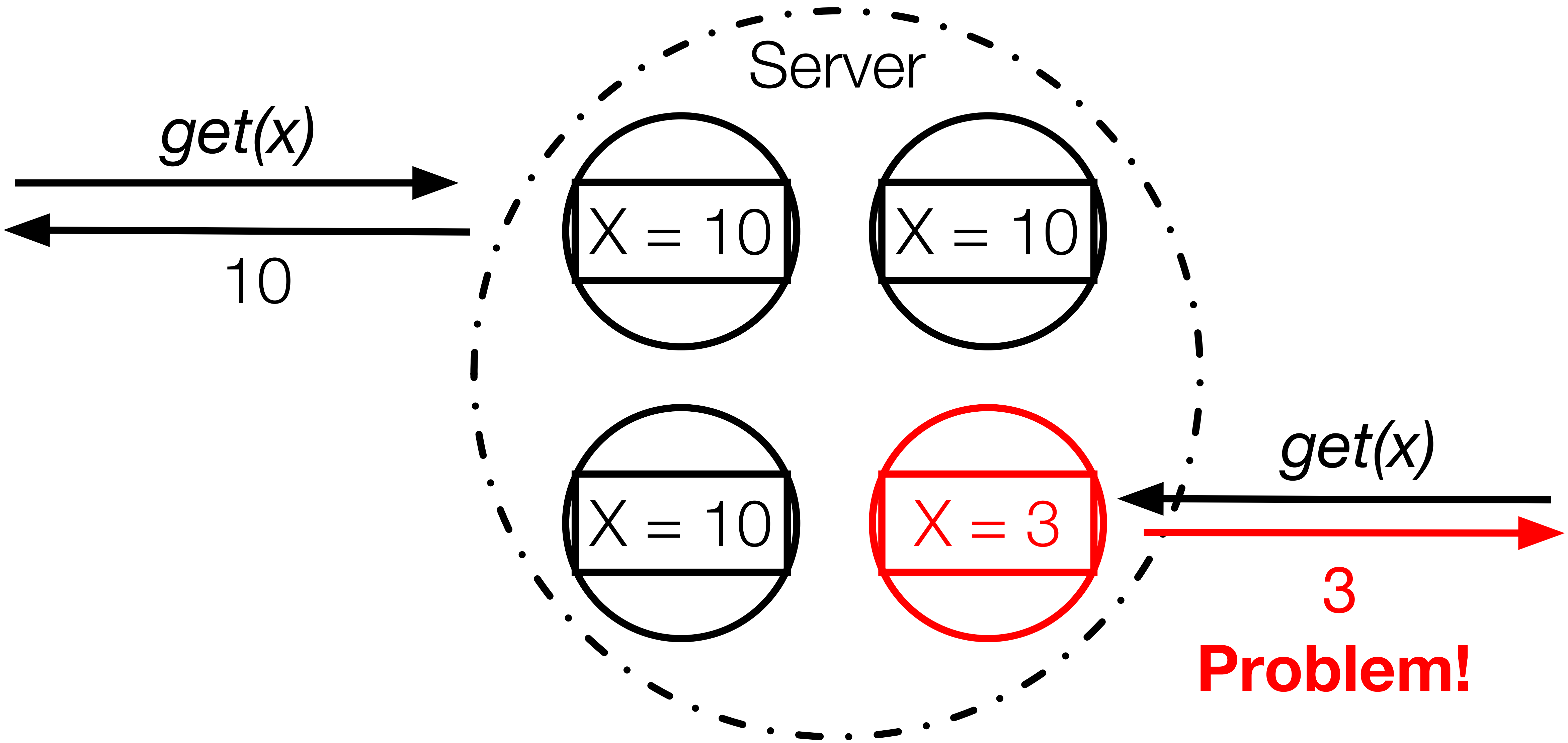
Motivation



Motivation



Motivation



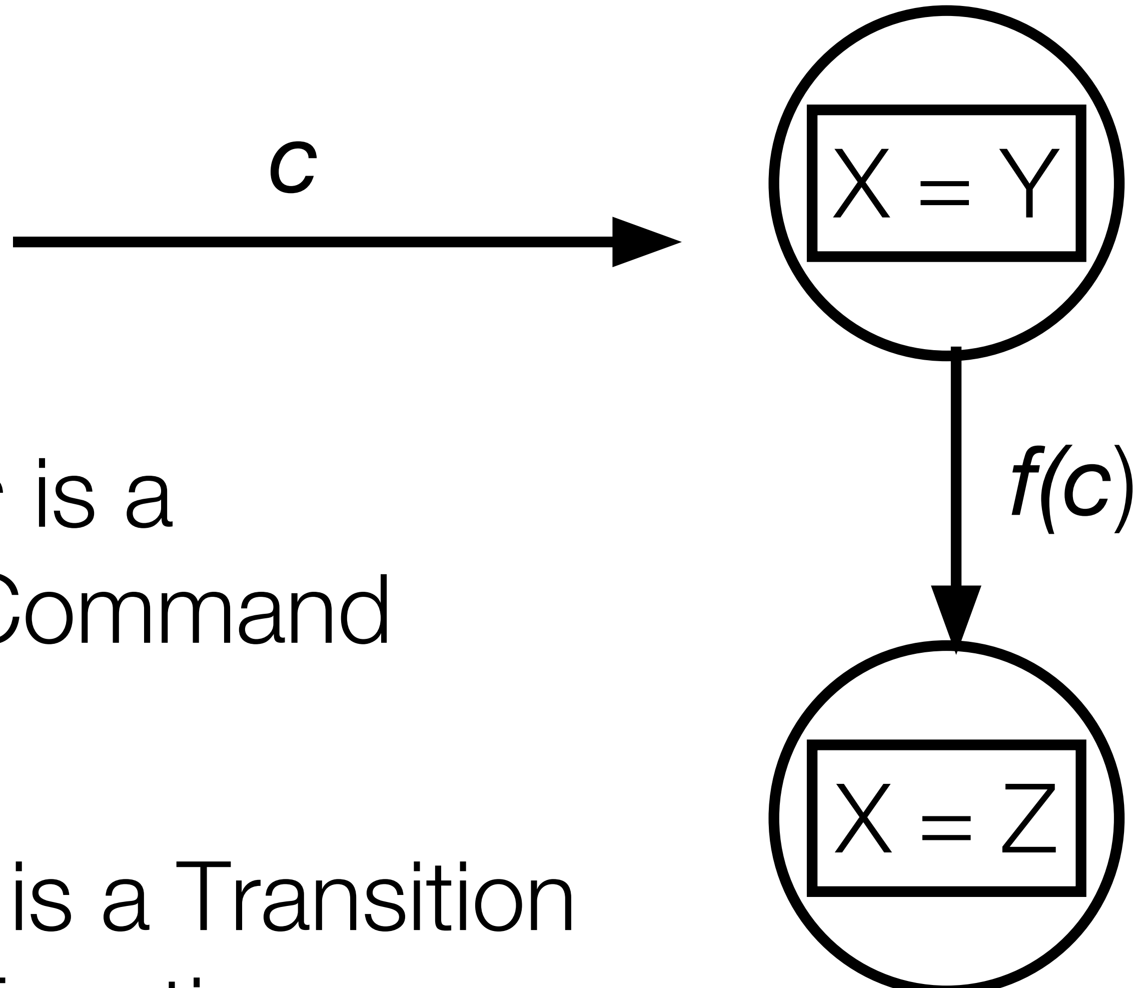
Problem

How can we ensure that all replicas are in the same state all of the time?

Outline

- Motivation
- State Machine Replication Approach
- Implementation
- Fault Tolerance
- Chain Replication
- Conclusions

State Machines



- *c* is a Command
- *f* is a Transition Function

State Machine Coding

- State machines are procedures
- Client calls procedure
- Avoid loops
- Flexible structure

State Machine Replication

- Each starts in the same initial state
- Executes the same requests
- Requires <u>consensus</u> to execute in same order
- Deterministic, each will do the exact same thing
- Produce the same output

State Machine Replication

All non faulty servers need:

- Agreement
  - Every replica needs to accept the same set of requests
- Order
  - All replicas process requests in the same relative order

Outline

- Motivation
- State Machines
- Implementation
- Fault Tolerance
- Chain Replication
- Conclusions

Implementation

Agreement

- Transmitter proposes a request; if it is non-faulty all servers will accept that request
- Transmitter can be client or server
- Client or Server can propose the request

Implementation

Agreement

- IC1: All non-faulty processors agree on the same value
- IC2: If transmitter is non-faulty, agree on its value

Ordering

"The Order requirement can be satisfied by assigning unique identifiers to requests and having state machine replicas process requests according to a total ordering relation on these unique identifiers."

Implementation

- **Order**
  - Assign unique ids to requests and process them in ascending order.
  - How do we assign unique ids in a distributed system?

Implementation
Client Generated IDs

Ordering via clocks

- Logical Clocks
- Synchronized Clocks

- Ideas from last class! [Lamport 1978]

Can the replicas generate unique identifiers?

Of course!

Implementation
Replica Generated IDs

- 2 Phase ID generation
  - Every replica proposes a *candidate*
  - One candidate is chosen and agreed upon by all replicas

Implementation
Replica Generated IDs

- When do we know a candidate is ***stable?***
  - A candidate is ***accepted***
  - No other pending requests with smaller candidate ids

Stability Testing

- Stability tests for logical and synchronized clocks?
- **Disadvantages**
  - Stability tests require all nodes to communicate
    - Logical: stabilizing requests
    - Synchronized: clock synchronization

Outline

- Motivation
- State Machines
- Implementation
- Fault Tolerance
- Chain Replication
- Conclusions

When does behavior become faulty?

When it's no longer consistent with specification!

Fault Tolerance

- **Fail-Stop**
  - A faulty server can be detected as faulty
- **Crash Failures**
  - Server can stop responding without notification (subset of Byzantine)
- **Byzantine**
  - Faulty servers can do arbitrary, perhaps malicious things

Fault Tolerance

- **Fail-Stop Tolerance**
  - To tolerate *t* failures, need *t+1* servers.
  - As long as 1 server remains, we're OK!
  - Only need to participate in protocols with other *live* servers

Fault Tolerance

Byzantine Failures

To tolerate *t* failures, need *2t + 1* servers

- Protocols now involve votes
  - Can only trust server response if the majority of servers say the same thing
- *t + 1* servers need to participate in replication protocols

Takeaways

- Can represent ***deterministic*** distributed system as *Replicated State Machine*
- Each replica reaches the same conclusion about the system ***independently***
- Formalizes notions of fault-tolerance in *SMR*

Discussion

- Why is State Machine Replication so important?
- What is the best case scenario in terms of replications for fault tolerance?
- Is the state machine approach still feasible?

Outline

- Motivation
- State Machines
- Implementation
- Fault Tolerance
- Chain Replication
- Conclusions

Chain Replication

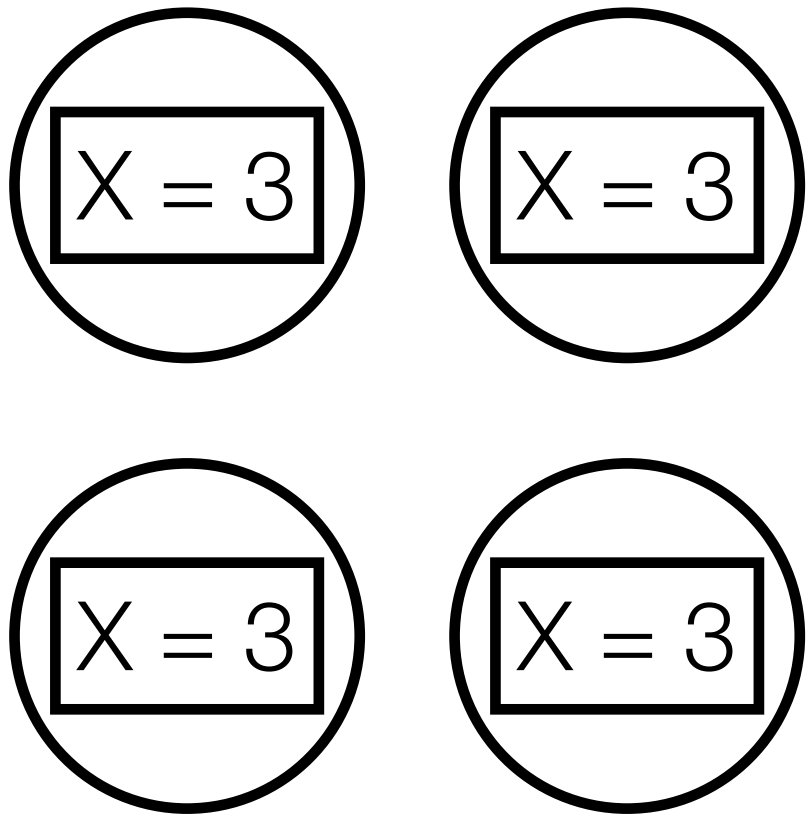
Authors

- Robert Van Renesse
  - Senior Researcher at Cornell
  - ACM Fellow and Ukelele enthusiast
  - Systems and Networking
- Fred Schneider

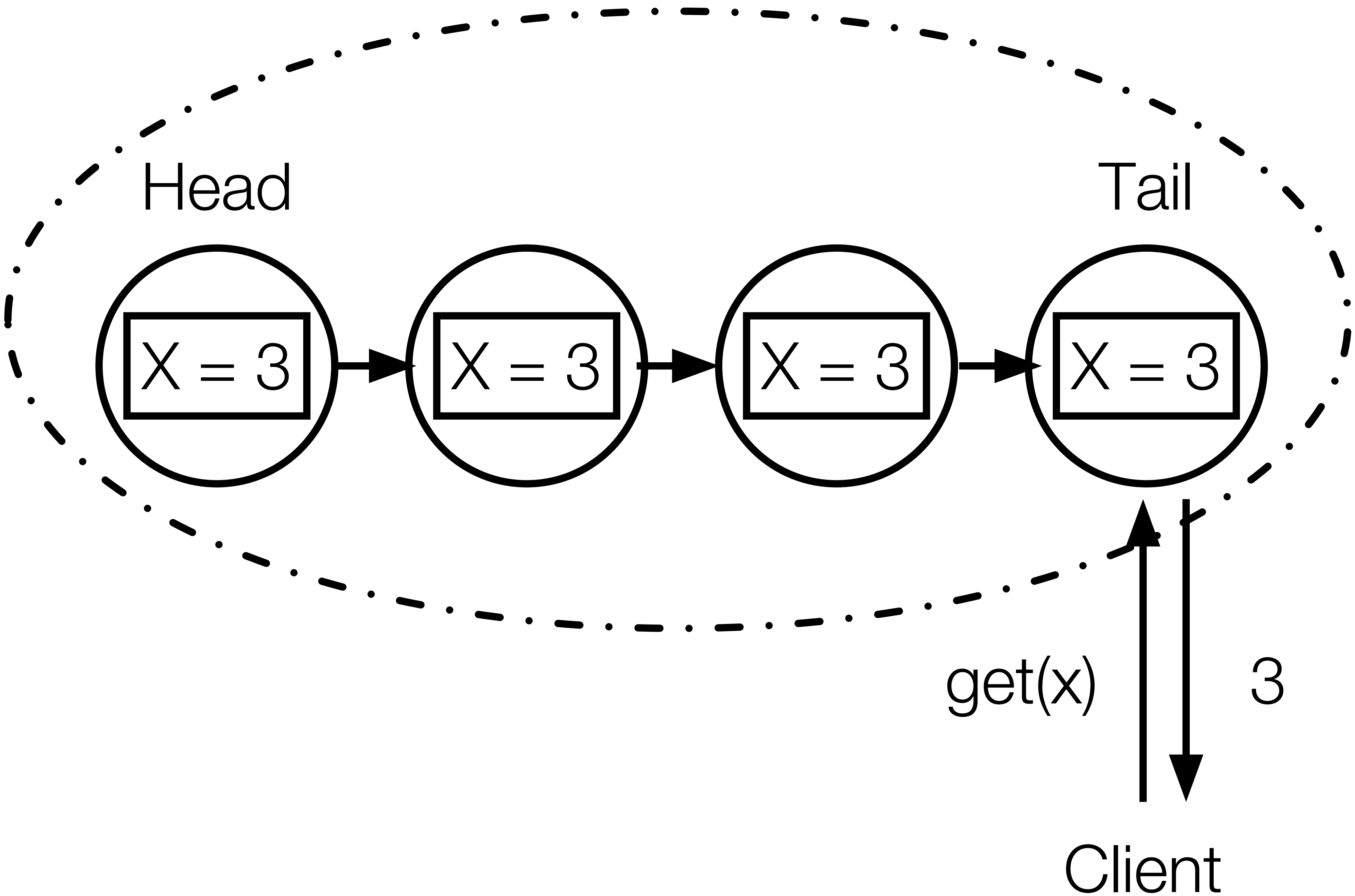
Chain Replication

- Fault Tolerant Storage Service
- Requests:
  - Update(x, y) => set object *x* to value *y*
  - Query(x) => read value of object *x*

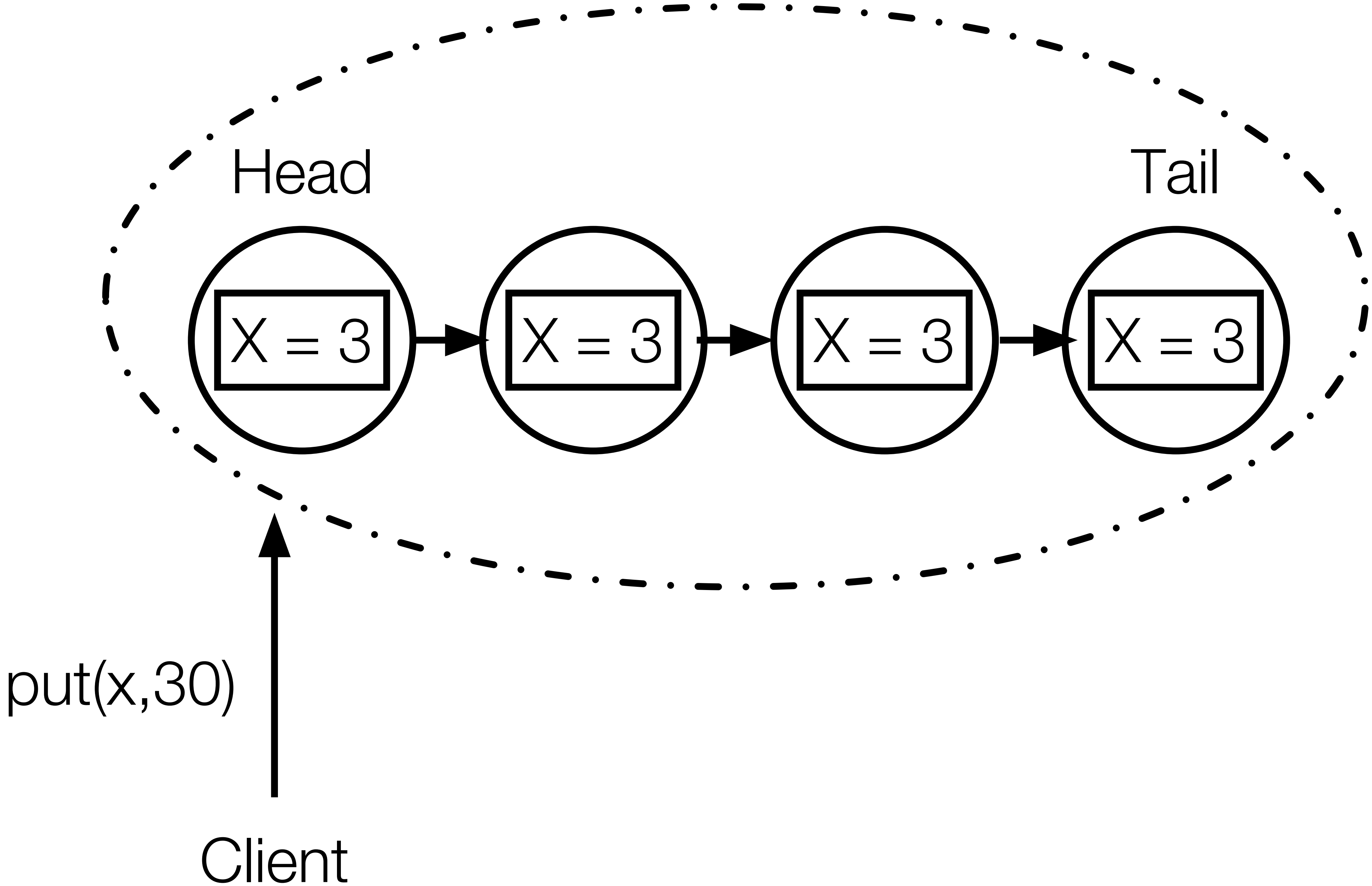
Chain Replication



Chain Replication

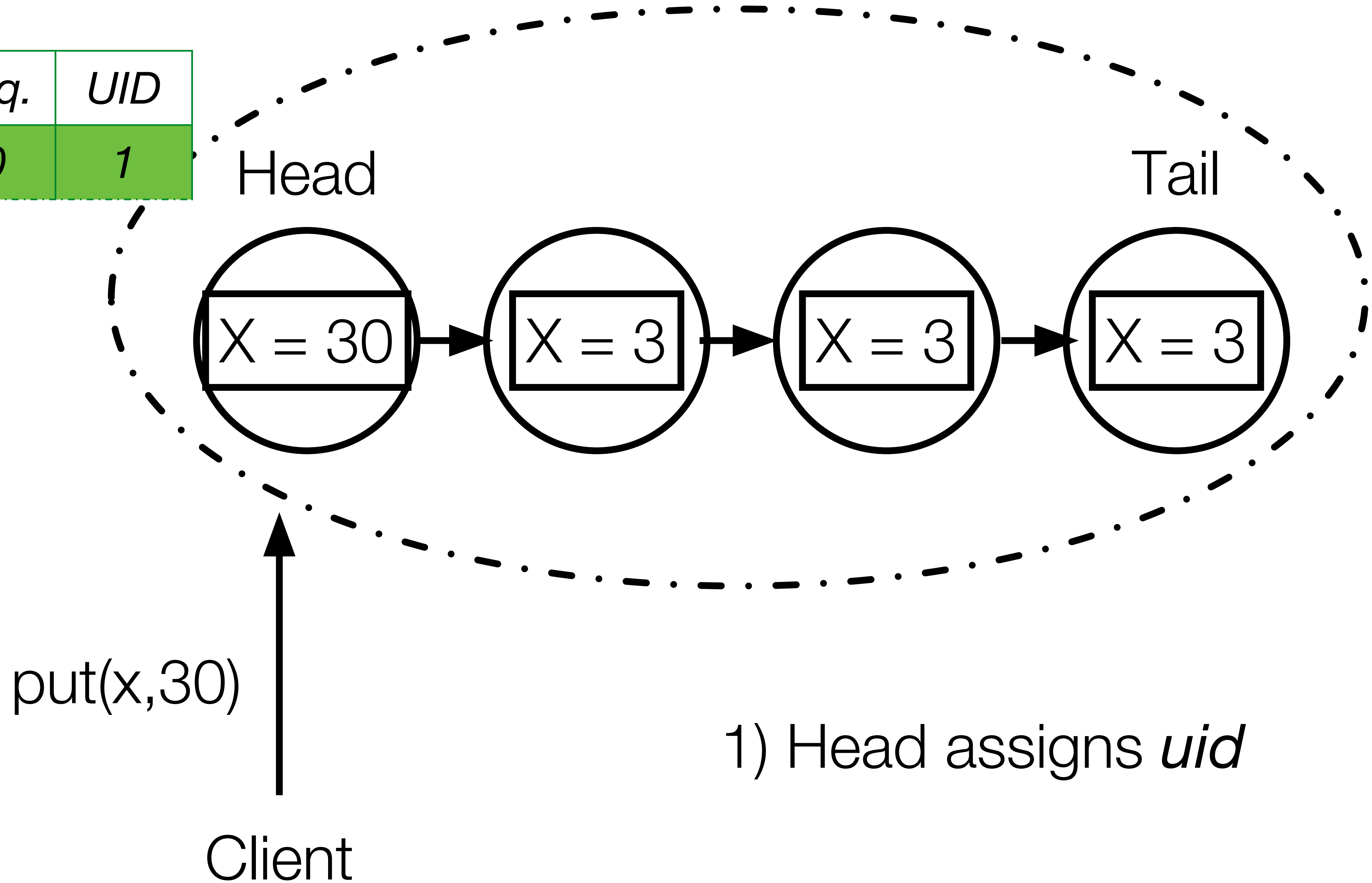


Chain Replication



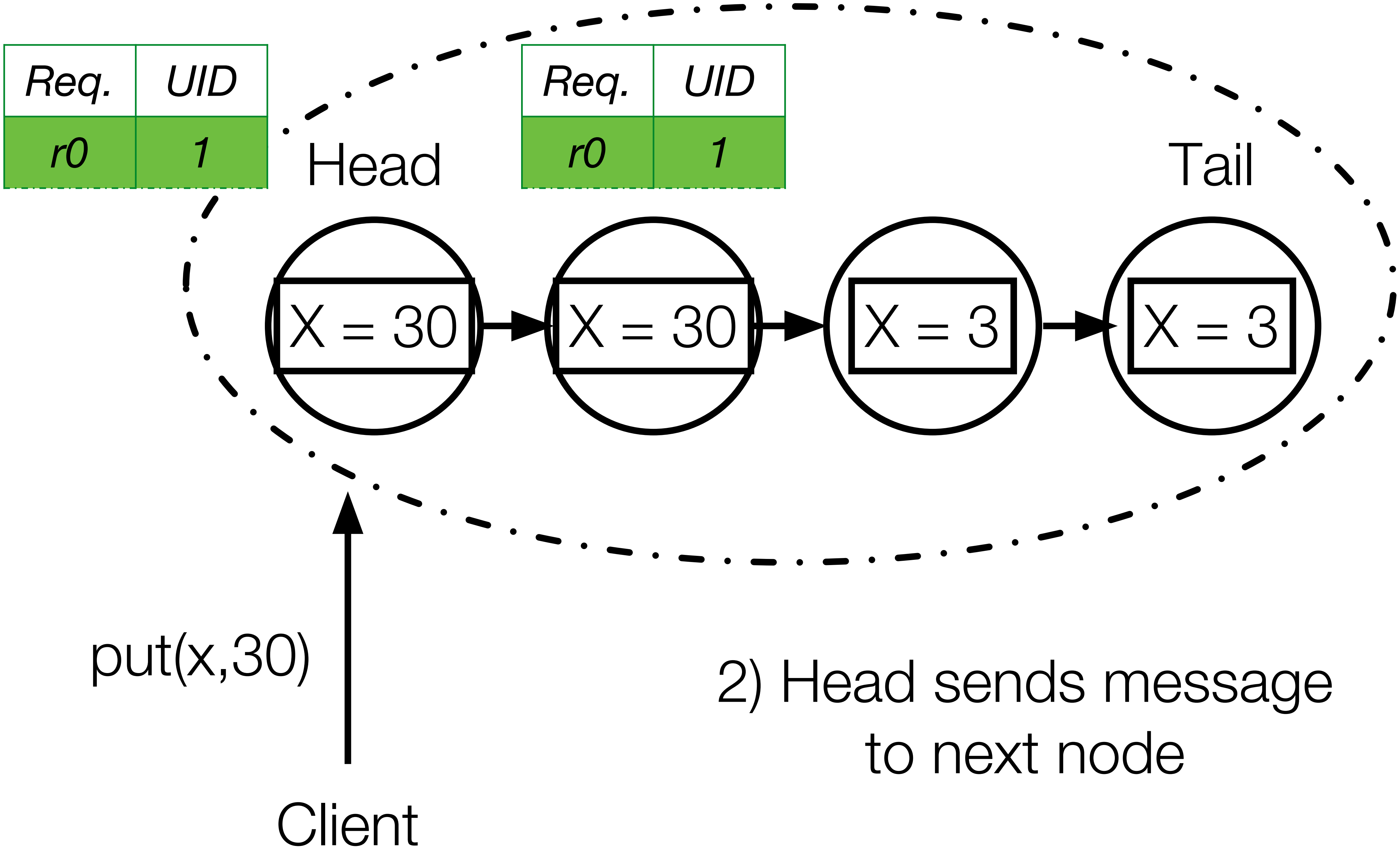
Chain Replication

Req.	UID
r0	*1*



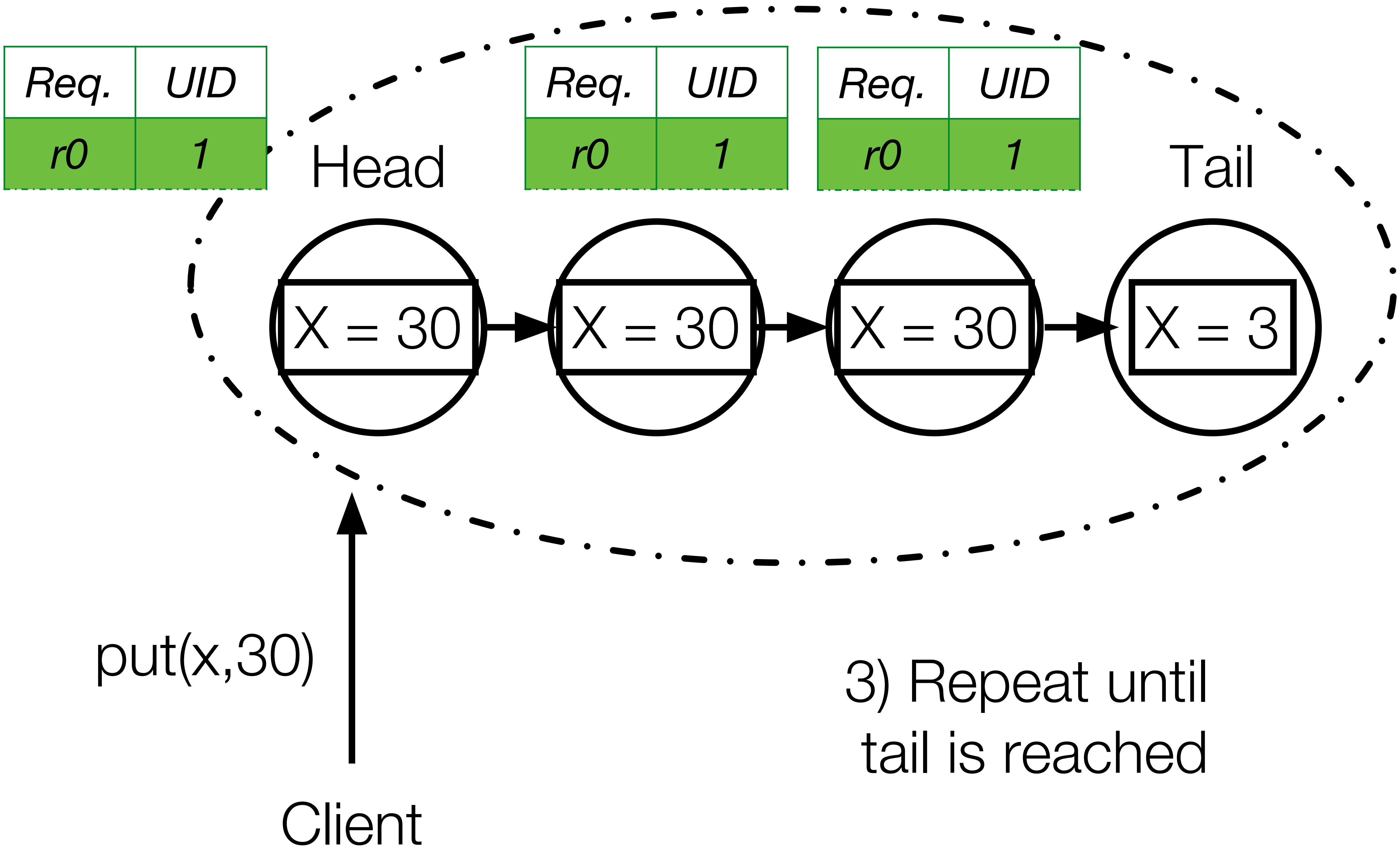
1) Head assigns *uid*

Chain Replication



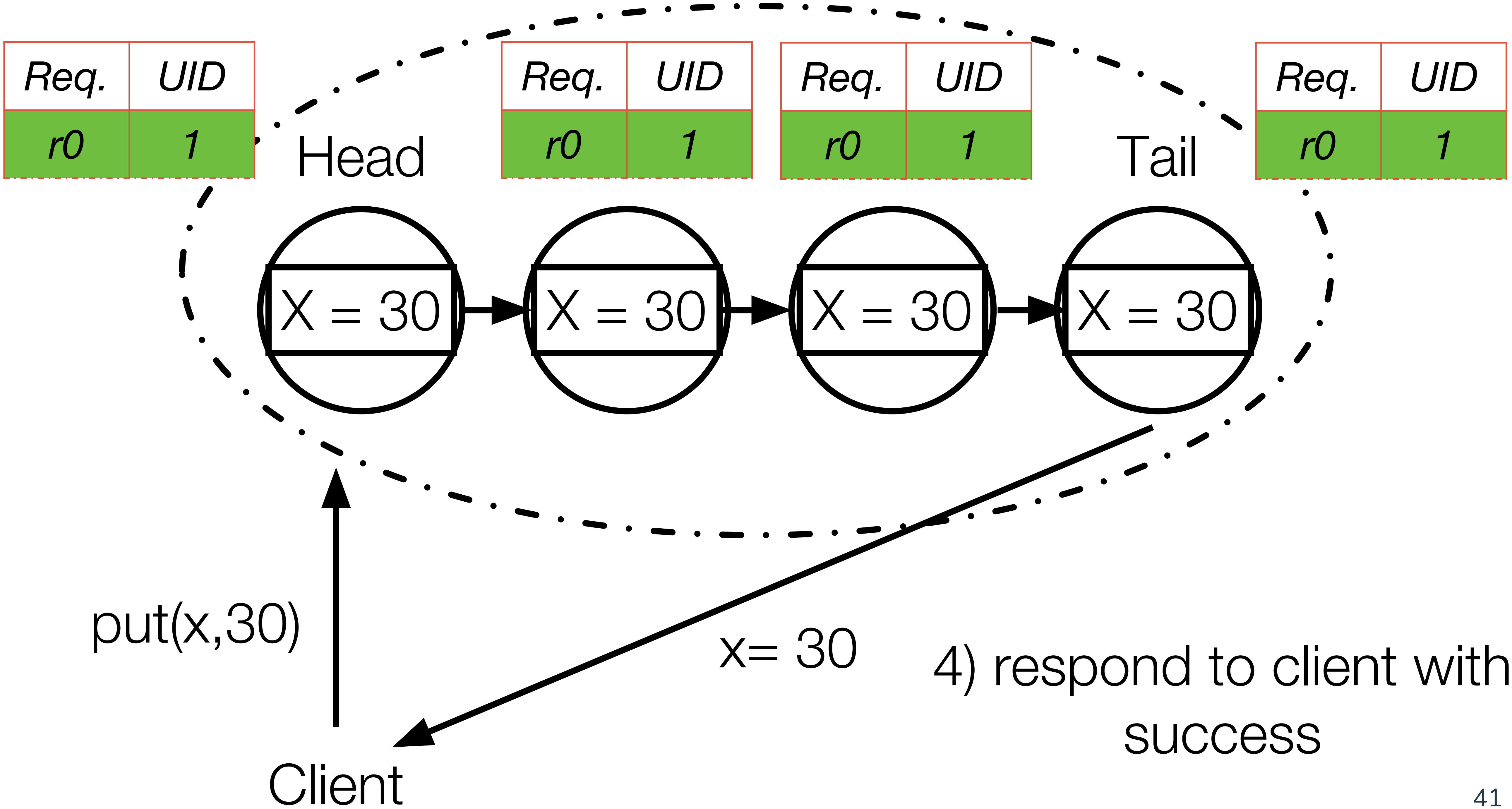
2) Head sends message to next node

Chain Replication



3) Repeat until tail is reached

Chain Replication



4) respond to client with success

Chain Replication Assumptions

- No partition tolerance
- High throughput
- Fail-stop processors
- A universally accessible, failure resistant or replicated Master

Chain Replication

How does Chain Replication implement State Machine Replication?

- *Agreement*
  - Only ***Update*** modifies state, can ignore ***Query***
  - Client always sends ***update*** to ***Head***. ***Head*** propagates request down chain to ***Tail***.
  - Everyone accepts the request!

Chain Replication

How does Chain Replication implement State Machine Replication?

- *Order*
  - Unique IDs generated implicitly by ***Head***'s ordering
  - FIFO order preserved down the chain
  - Tail interleaves ***Query*** requests

Chain Replication Fault Tolerance

- Trusted Master
  - *Fault-tolerant state machine*
  - Trusted by all replicas
  - Monitors all replicas & issues commands

Chain Replication Fault Tolerance

- **Head Fails**
  - *Master* assigns 2nd node as Head
- **Intermediate Node Fails**
  - *Master* coordinates chain link-up
- **Tail Fails**
  - *Master* assigns 2nd to last node as Tail

Outline

- Motivation
- State Machines
- Implementation
- Fault Tolerance
- Chain Replication
- Conclusions

Conclusions

- Implements the “exercise left to the reader” hinted at by Lamport’s paper
- Provides ***some*** of the concrete details needed to actually implement this idea
  - But still a fair number of details in real implementations that would need to be considered
  - Chain replication illustrates a “simple” example with fully concrete details
- A key contribution that bridges the gap between academia and practicality for SMR

Chain Replication Discussion

- Comparison to other primary/backup protocols?
- What are the tradeoffs of Chain Replication?
  - Latency
  - Consistency
- Any thoughts on the Trusted Master system?