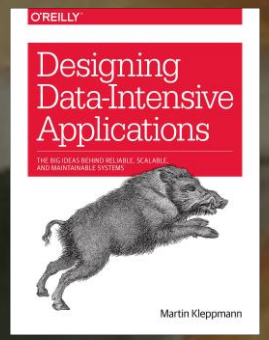
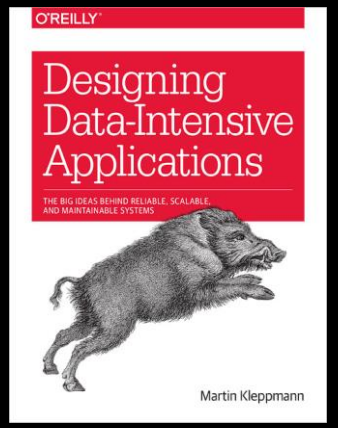
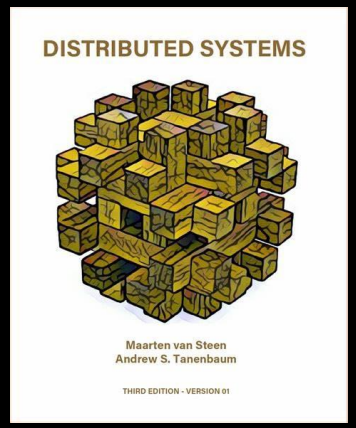
MSFT Sys Meetup



http://

Freely accessible resources





[Resources | MSFT-System-Meetup (microsoft-distributed-system-meetup.github.io)]

Company Privacy

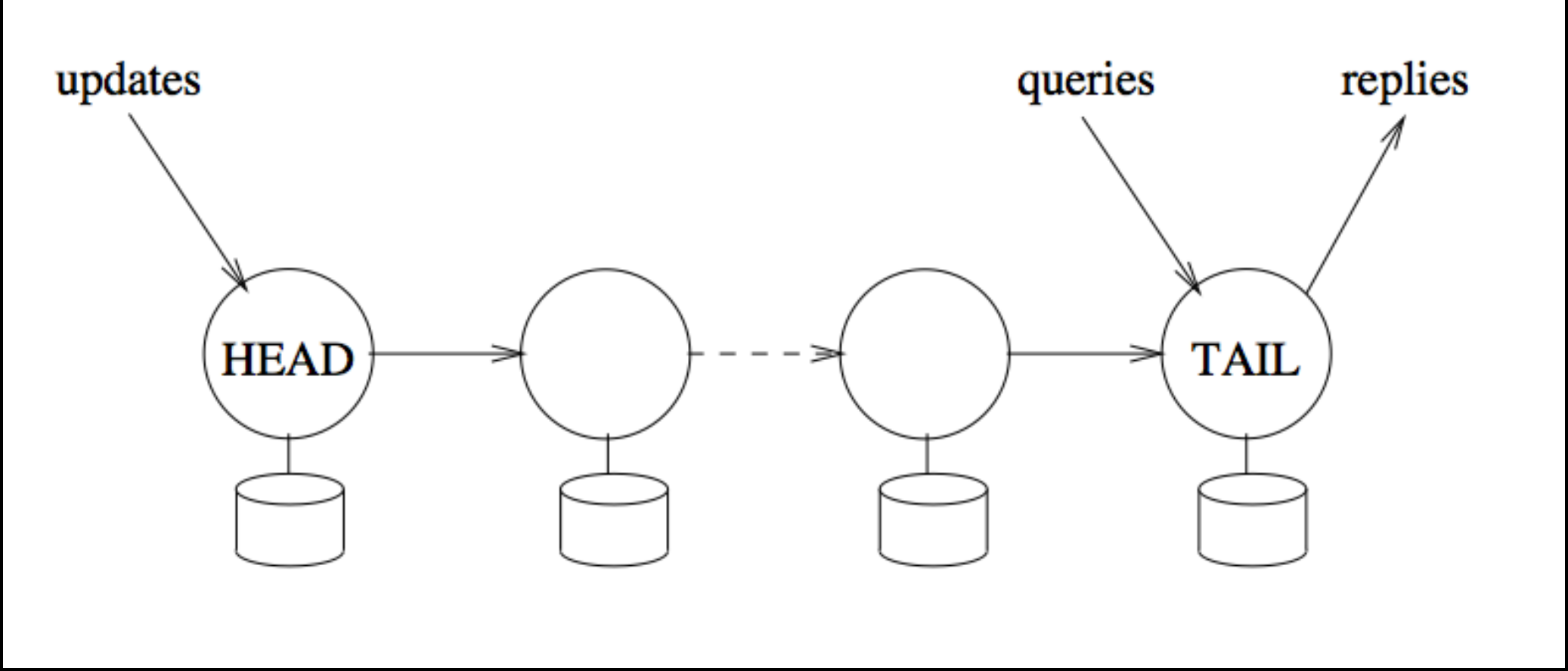
Topic Covered

- What is chain replication
- **Chain replication protocol**
- **How chain replication works**
- Performance
- **Why chain replication**

What is chain replication: Replication

- Replication Approaches:
  - State transfer
    - Memory & CPU & I/O devices
  - Replicated state machine
    - Operations (Deterministic & Non-deterministic)
- Questions in replication
  - What state to replicate?
  - Does primary have to wait for backup?
  - When to cut over to backup?
  - How to bring a replacement backup up to speed?

What is chain replication: Overview



What is chain replication: Overview

State is:

$Hist_{objID}$: **update request sequence**

$Pending_{objID}$: **request set**

Transitions are:

T1: Client request r arrives:

$Pending_{objID} := Pending_{objID} \cup \{r\}$

T2: Client request $r \in Pending_{objID}$ ignored:

$Pending_{objID} := Pending_{objID} - \{r\}$

T3: Client request $r \in Pending_{objID}$ processed:

$Pending_{objID} := Pending_{objID} - \{r\}$

if $r = \mathsf{query}(objId, opts)$ **then**

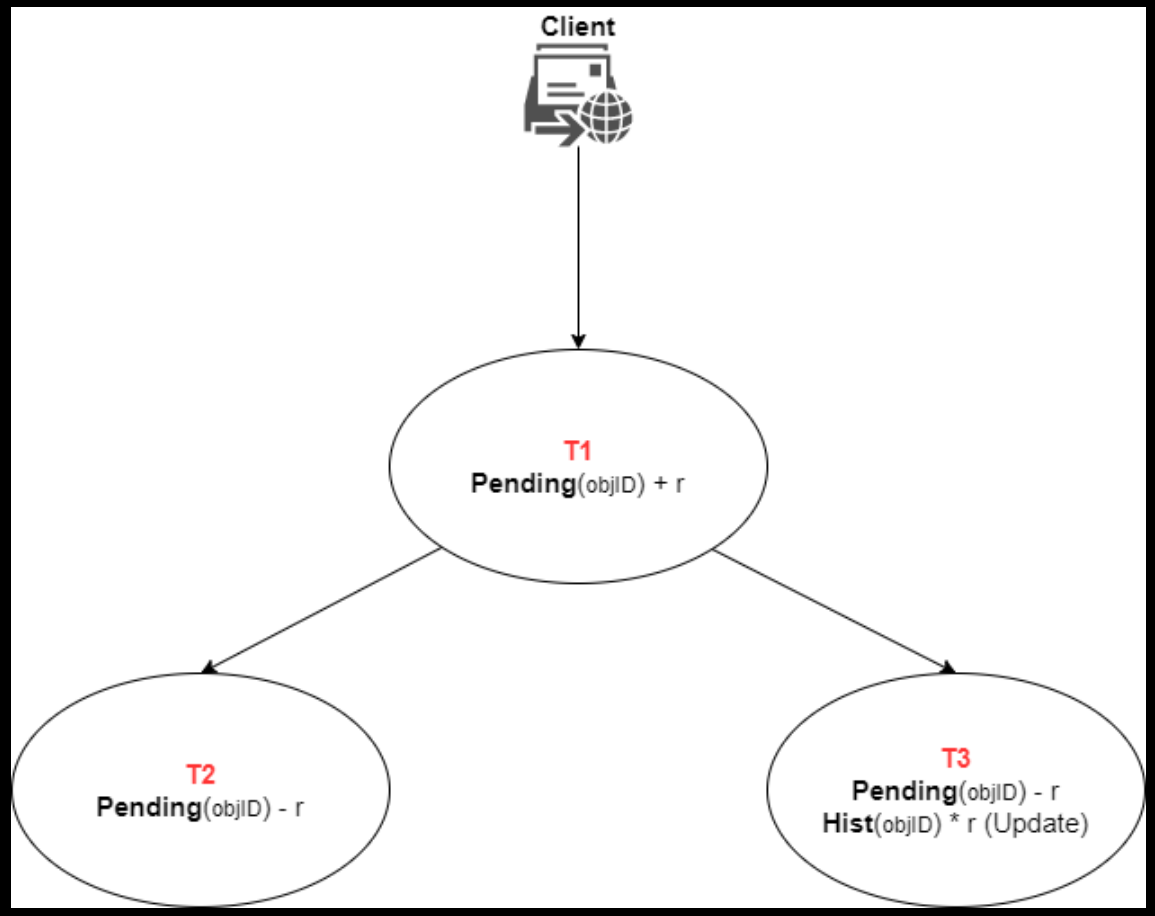
reply according options $opts$ based on $Hist_{objID}$

else if $r = \mathsf{update}(objId, newVal, opts)$ **then**

$Hist_{objID} := Hist_{objID} \cdot r$

reply according options $opts$ based on $Hist_{objID}$

Figure 1: Client's View of an Object.

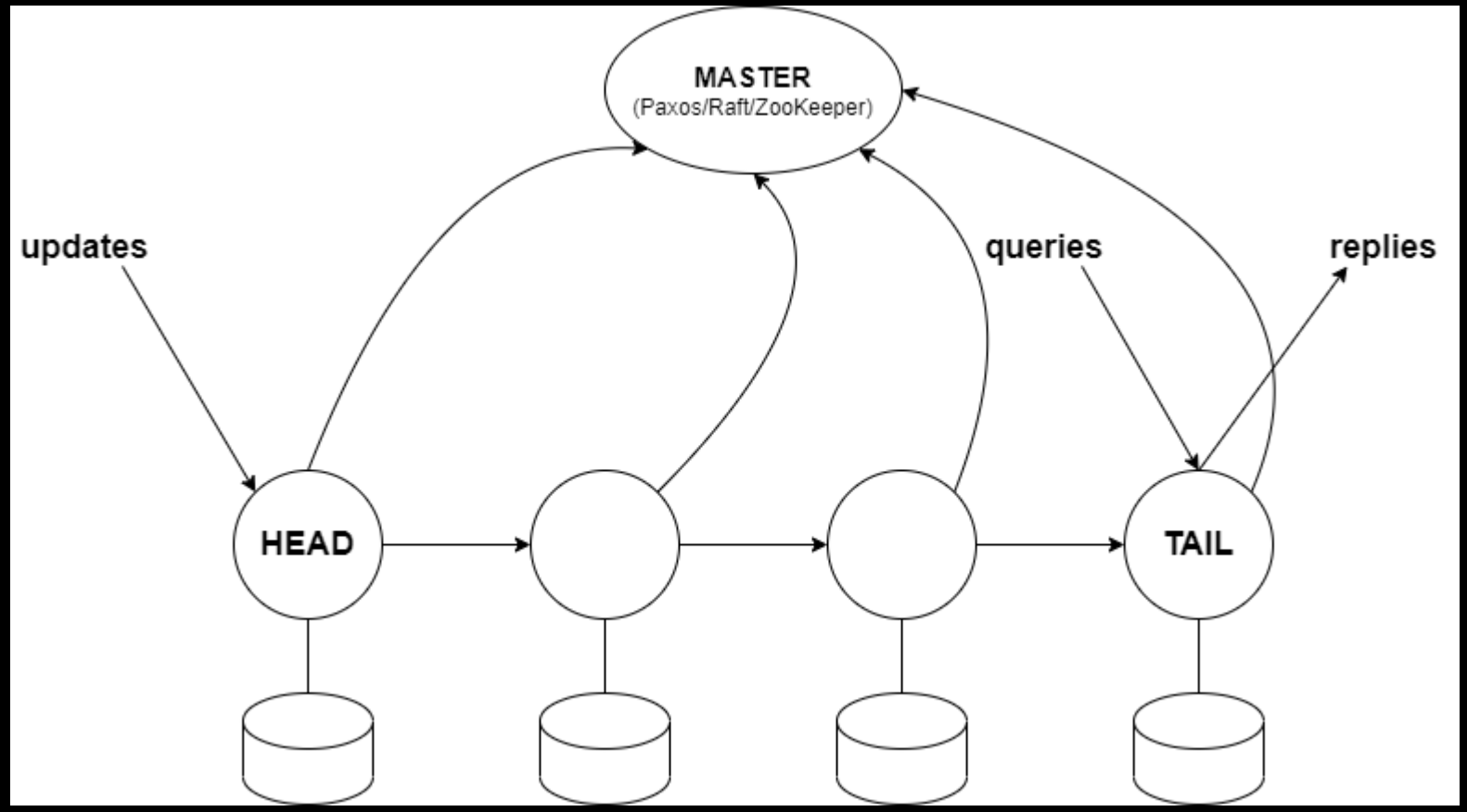


Chain replication protocol

- $Hist_{objID}$
  - The value of $Hist_{objID}$ stored by tail T of the chain (Tail)
- $Pending_{objID}$
  - Defined to be the set of client requests received by any server in the chain and not yet processed by the tail. *(Header)*
- *Request processing*
  - Clients send requests to either the head (update) or the tail (query).
  - Reply are all generated by tail (Query + Update)

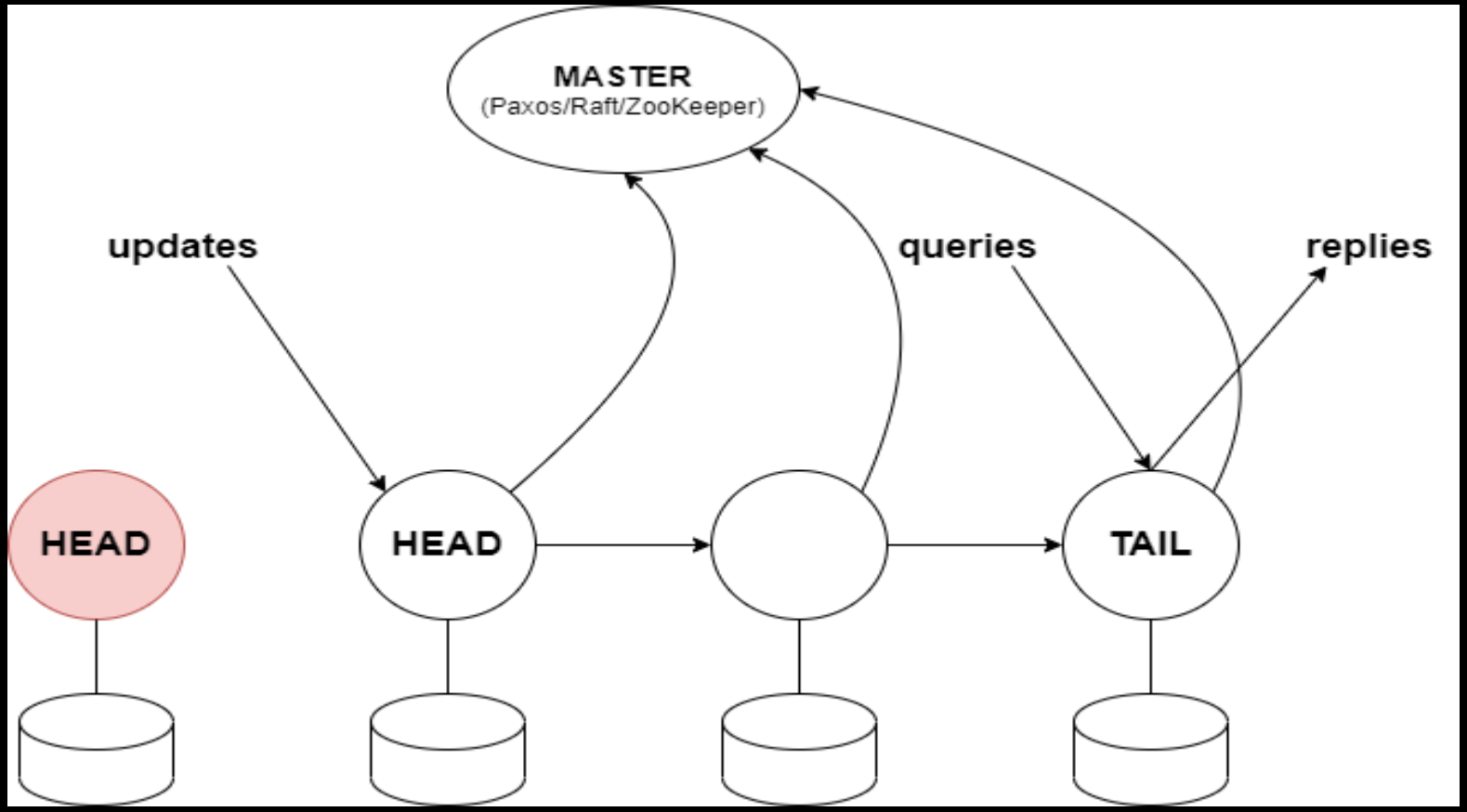
Chain replication protocol -- Failure

- Detects failures of servers
- Inform each server in the chain of its new predecessor or new successor in the new chain
- Inform clients which server is the head and which is the tail of the chain



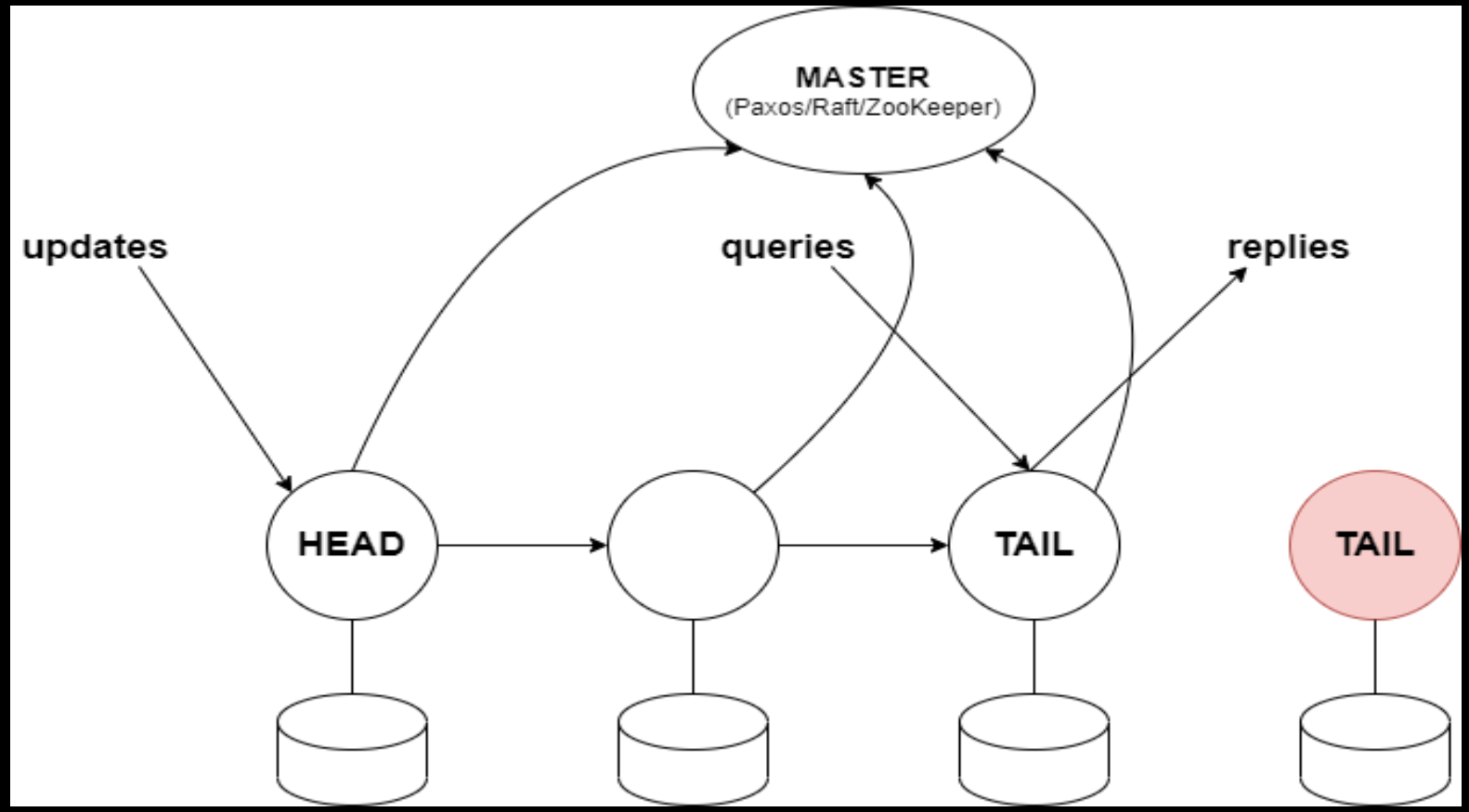
Chain replication protocol: Failure of Head

- $Pending_{objID}$
- Requests forwarded will be kept in new head.
- Requests haven't been forwarded to a successor will be ignored.



Chain replication protocol: Failure of Tail

- $Pending_{objID}$ & $Hist_{objID}$
- Decreased PendingobjID
- Increased HistobjID
- A request has been processed by T- (new Tail) and failed in forward to T (Tail). Will new tail send the result to the client?



Chain replication protocol: Failure of other servers

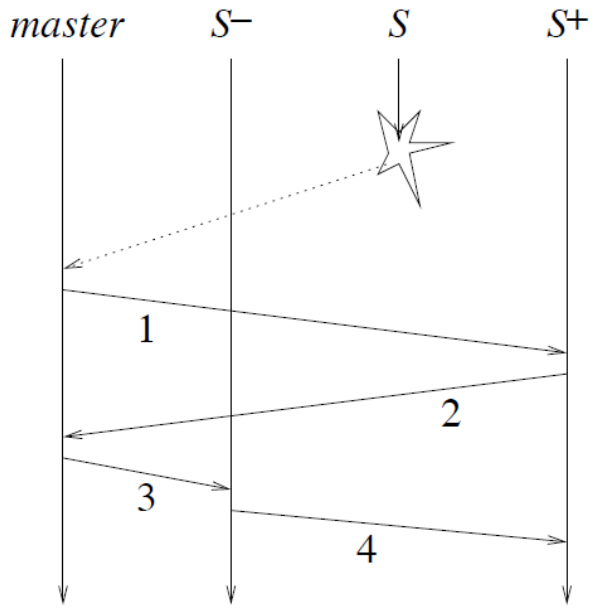
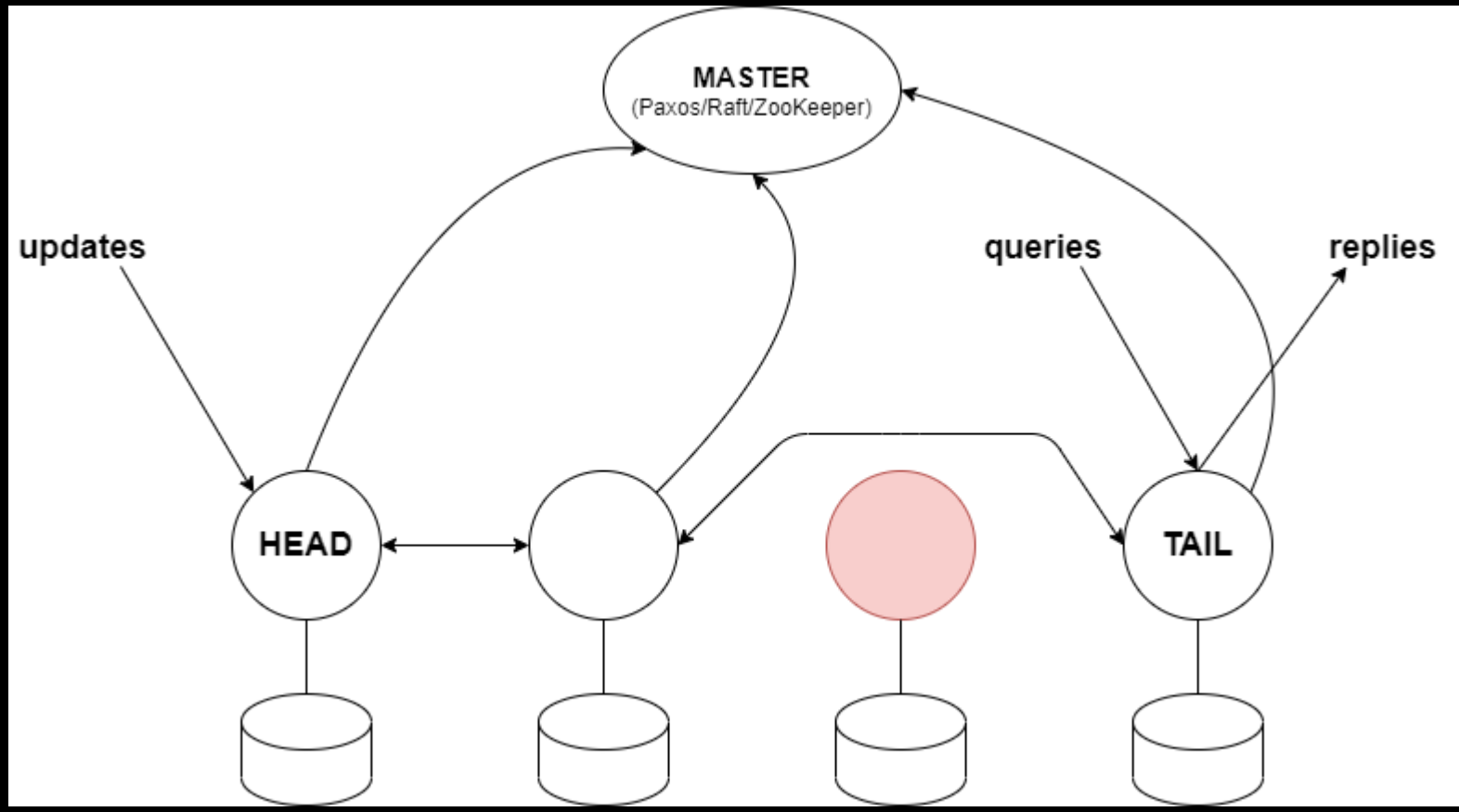


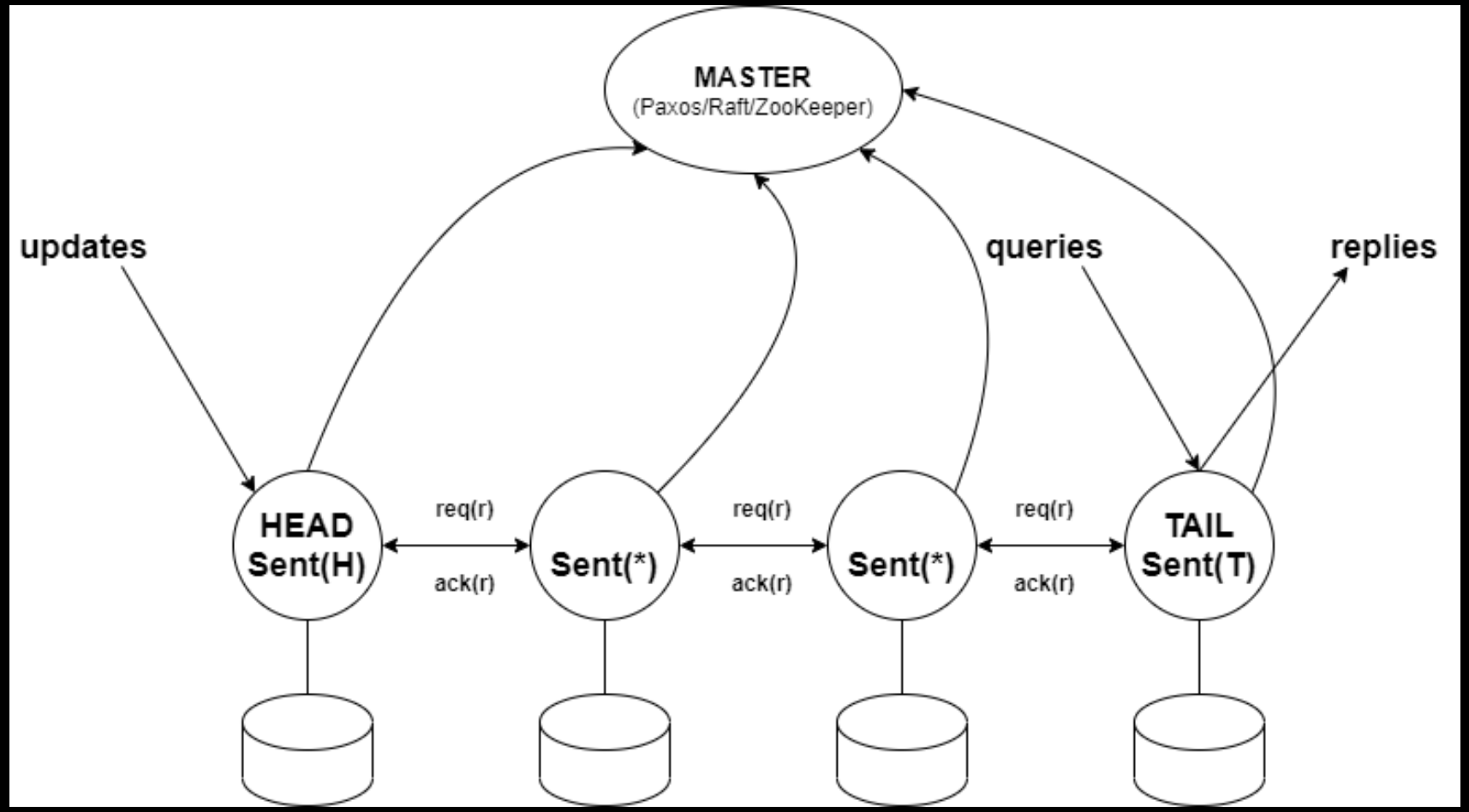
Figure 3: Space-time diagram for deletion of internal replica.



Chain replication protocol: Failure of other servers

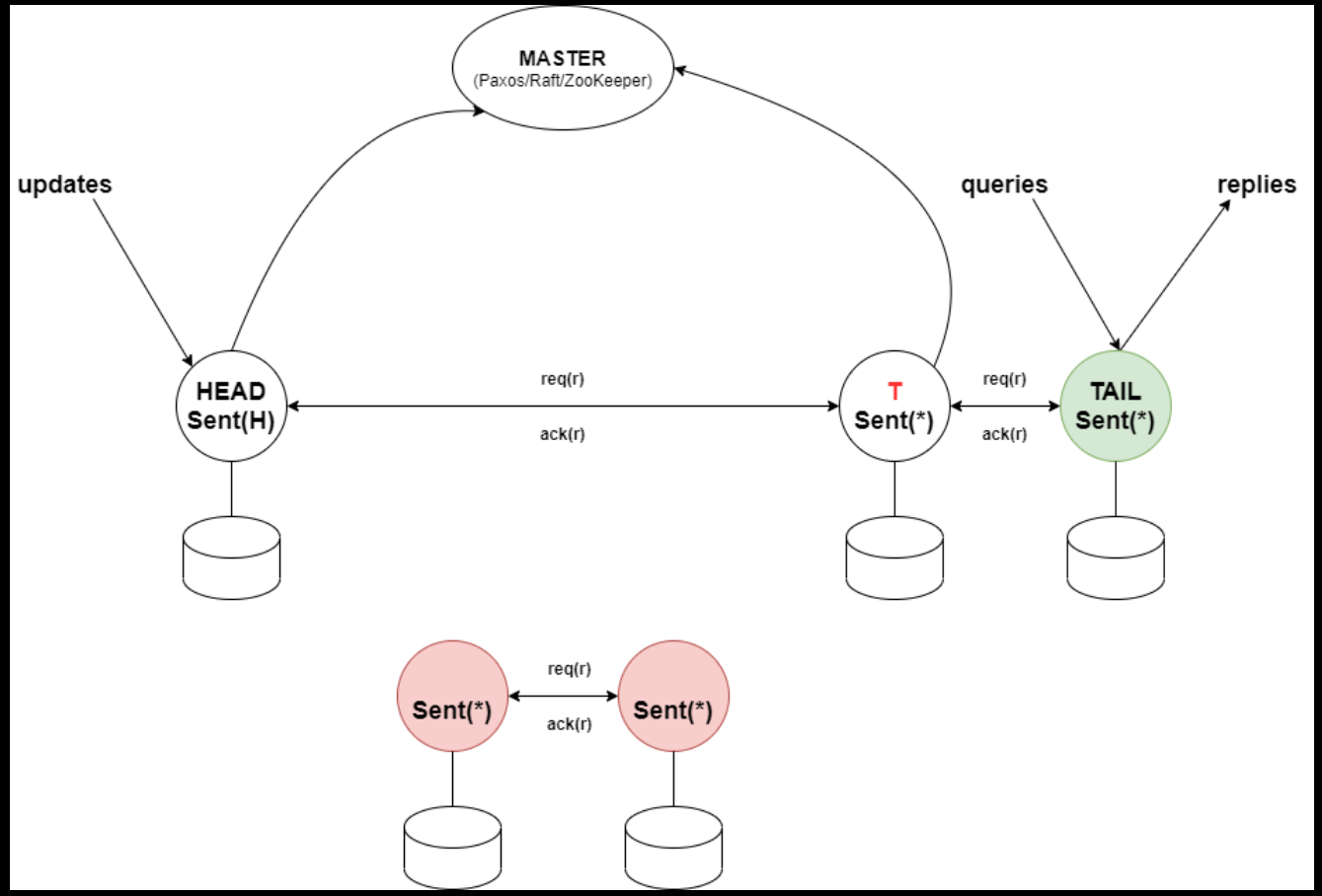
- Send to S+ (using the FIFO link that connects them) those requests in $Hist_{objID}(S-)$ that might not have reached S+.
- Each server i maintains a list $Sent(i)$ of update requests that i has forwarded to some successor but that might not have been processed by the tail.
- Whenever server i forwards an update request r to its successor, server I also appends r to $Sent(i)$. The tail sends an acknowledgement ack(r) to its predecessor when it completes the processing of update request r. And upon receipt ack(r), a server i deletes r from Sent(i) and forwards ack(r) to its predecessor.

Chain replication protocol: Failure of other servers

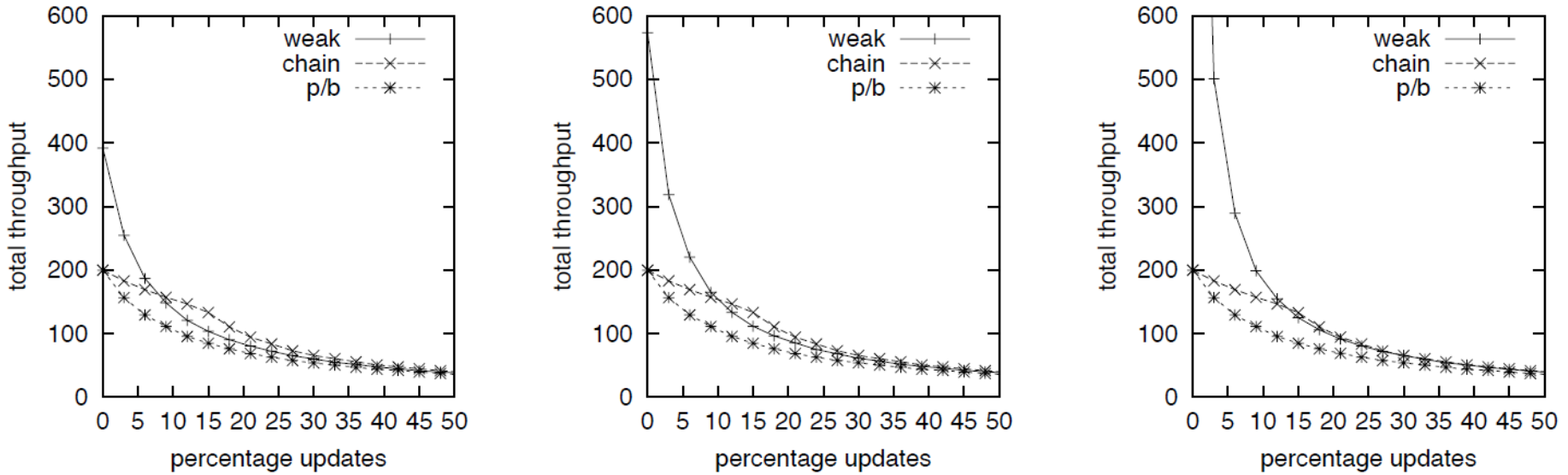


Chain replication protocol: Extending a chain

- Initialize HistobjID (copy from current Tail)
- T is notified that it no longer is the tail.
- Requests in Sent(T) are sent to TAIL
- The master is notified that TAIL is the new tail.
- Notify all clients the new tail.



Performance: Single Chain, No Failures



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

Figure 4: Request throughput as a function of the percentage of updates for various replication management alternatives **chain**, **p/b**, and **weak** (denoting **weak-chain**, and **weak-p/b**) and for replication factors t.

Performance: Multiple Chain, No Failures

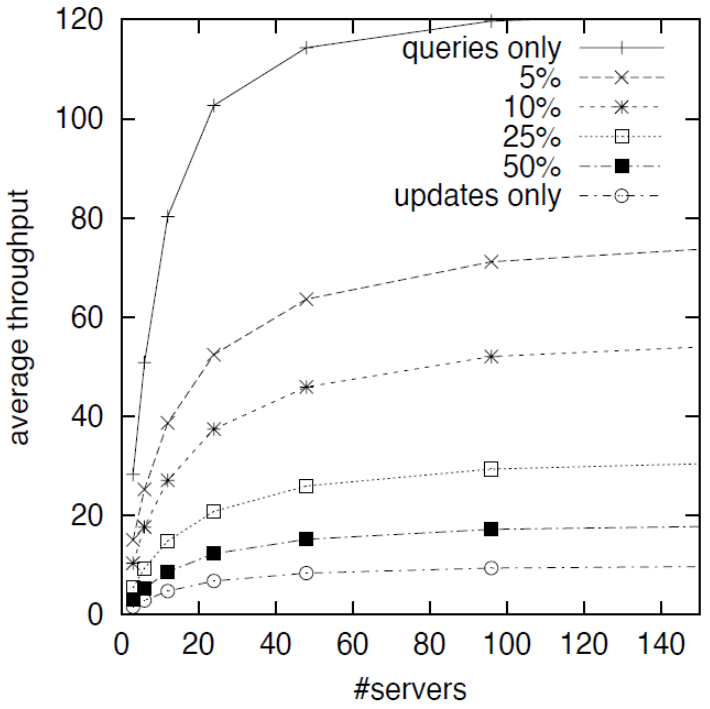
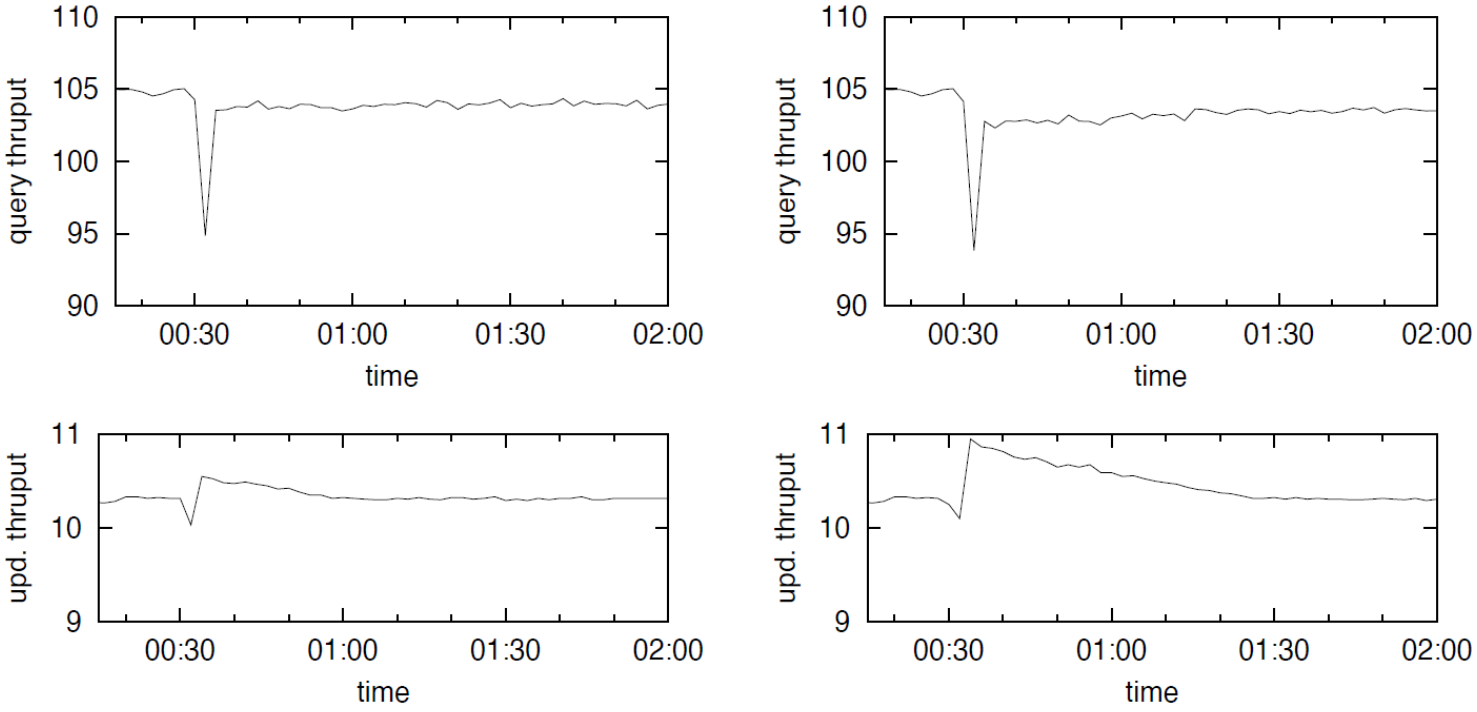


Figure 5: Average request throughput per client as a function of the number of servers for various percentages of updates.

Performance: With Failures

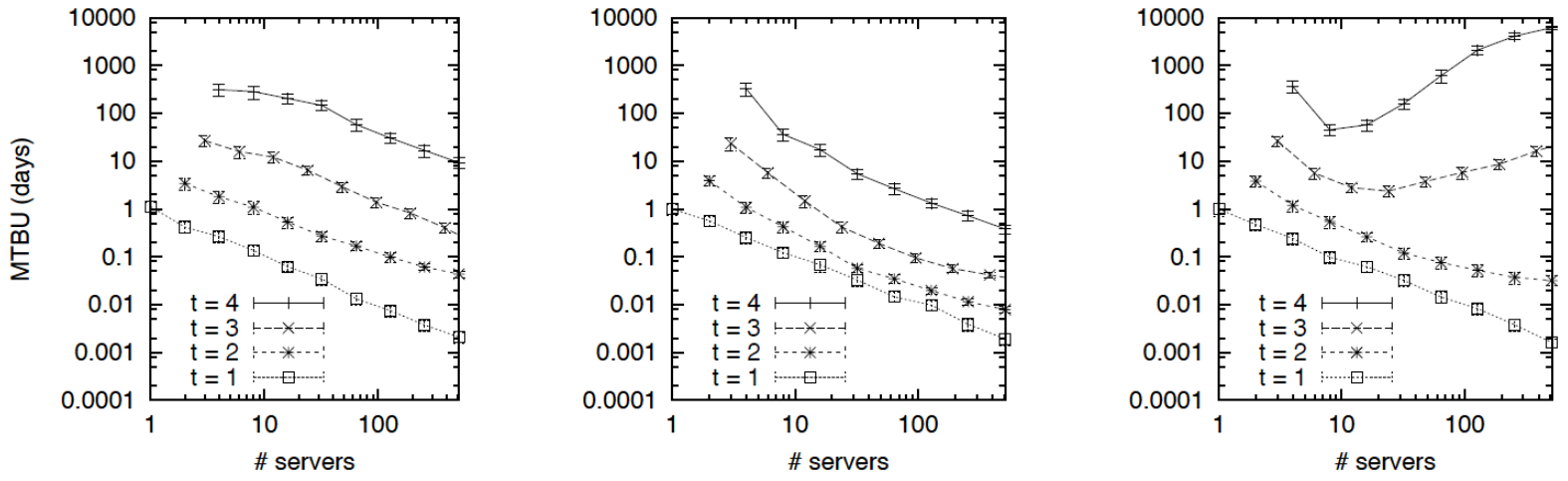


(a) one failure

(b) two failures

Figure 6: Query and update throughput with one or two failures at time 00:30.

Performance: Volume placement strategies

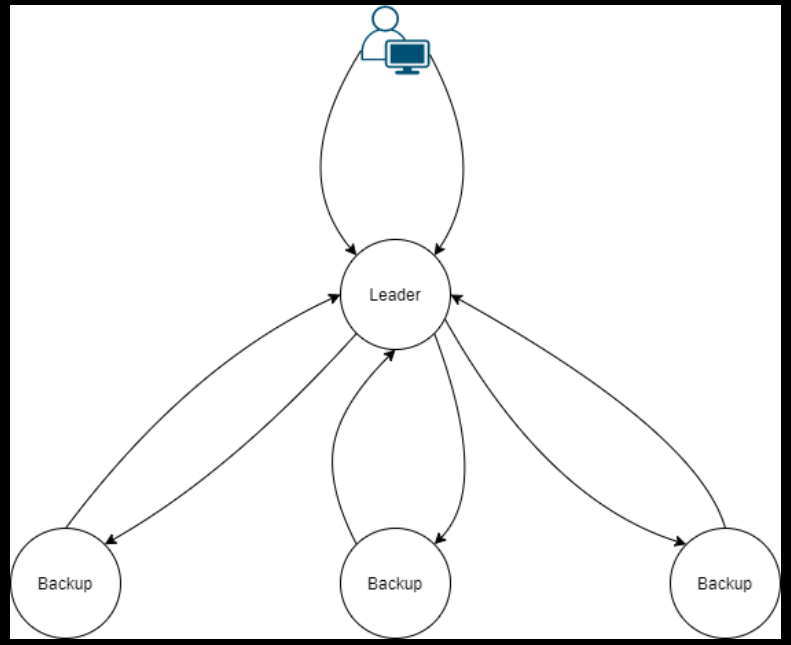


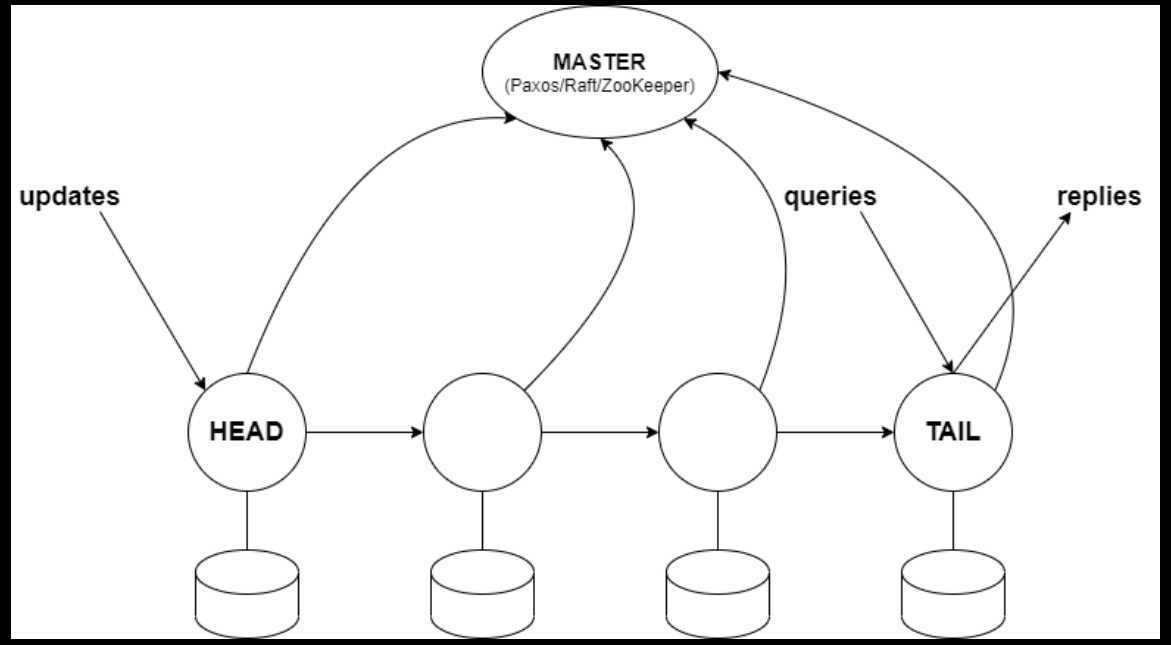
(a) **ring** (b) **rndseq** (c) **rndpar**

Figure 7: The MTBU and 99% confidence intervals as a function of the number of servers and replication factor for three different placement strategies: (a) DHT-based placement with maximum possible parallel recovery; (b) random placement, but with parallel recovery limited to the same degree as is possible with DHTs; (c) random placement with maximum possible parallel recovery.

Open discussion: Why chain replication?

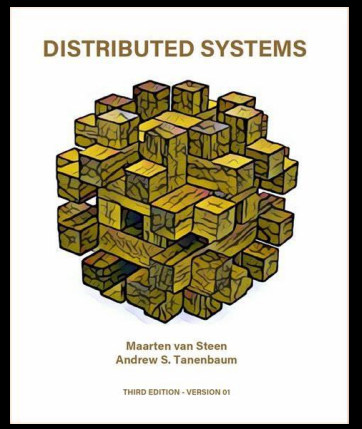
- Strong consistency VS. High throughput and high availability.
- Chain replication VS. Raft / ZooKeeper

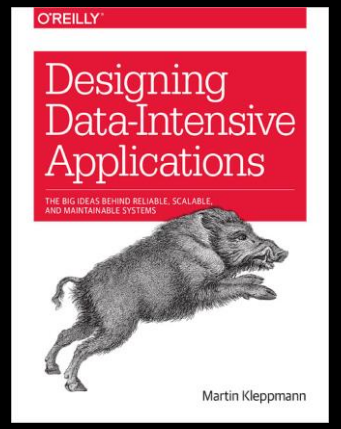




Questions?

Freely accessible resources





[Code]

[Zoom]

[Course]

[DDIA (O'Reilly)]

[Distributed System 3rd edition]

Calendar: https://docs.google.com/spreadsheets/d/1RsbGpq1cwNSmYn5hcmT8Hv5O4qssl2HXsTcG82RHVQk/edit?usp=sharing

(Internal) Teams: g078pwd

(Public) Discord

(Public) WeChat: add mossaka or Lin1991Wen

Notion: https://www.notion.so/invite/cd6df70a94e7f67f6d21f4c509783d3c9cfd0e69

YouTube: https://www.youtube.com/playlist?list=PL1voNxn5MODMJxAZVvgFHZ0jZ-fuSut68