Chain Replication



COS 316: Principles of Computer System Design
Lecture 15

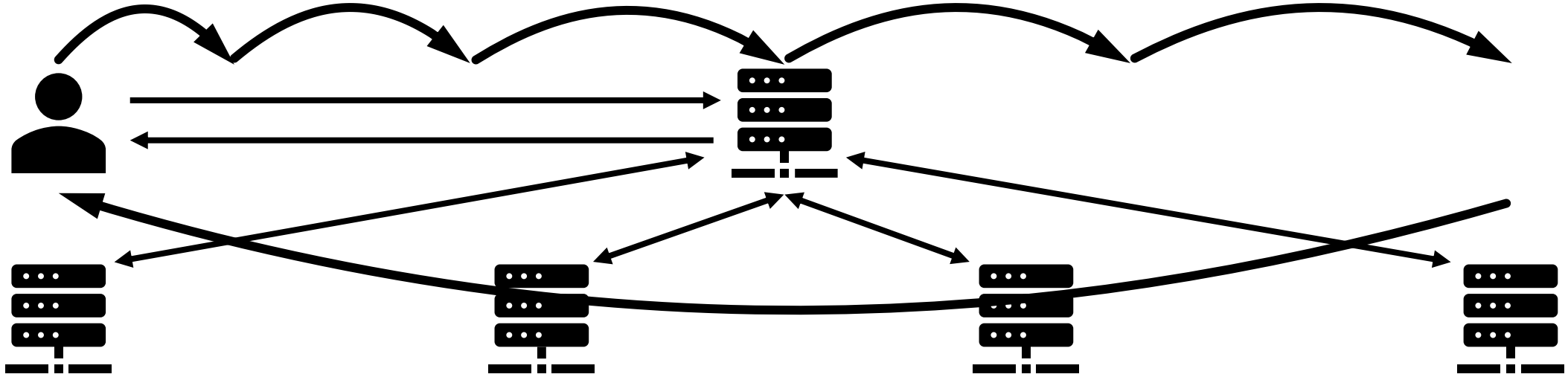
Wyatt Lloyd

Some slides developed by Nicolaas Kaashoek

Chain Replication

An extension to primary-backup replication

Rather than one primary, arrange servers in a *chain*



Chain Replication: Object Histories

Rather than a single, shared log, maintain history for object

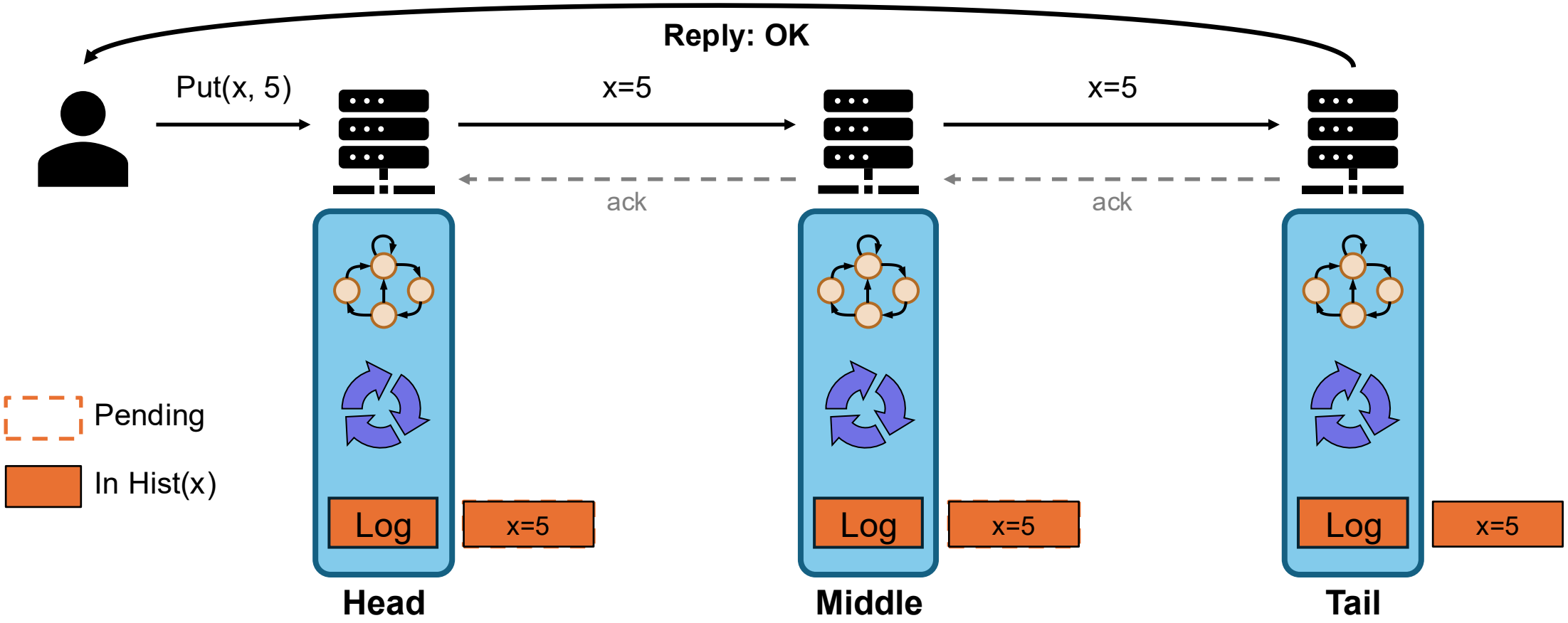
- Hist(obj) = the set of updates for obj processed by tail
- Pend(obj) = the outstanding updates to obj *not yet processed by the tail*

How can we get the current state of an object? Apply all updates in the history.

Out of scope: Trimming history, checkpointing, etc

Chain Replication: Updates

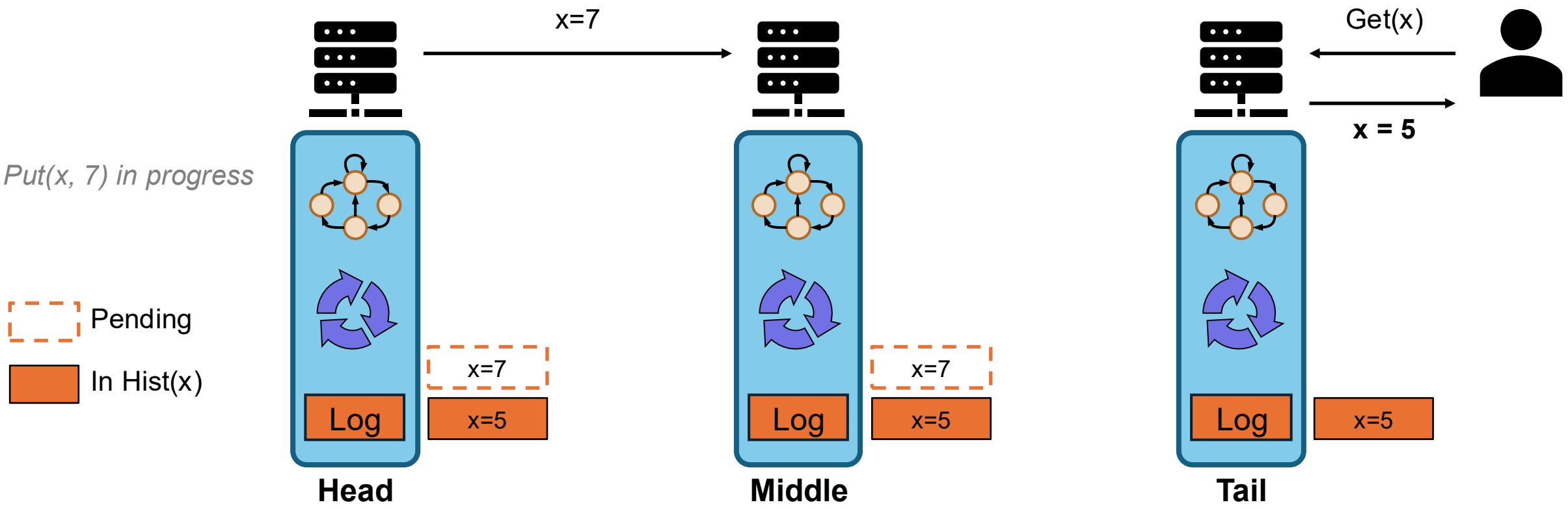
All updates go to the head of the chain



Chain Replication: Read Requests

Read requests go to the tail

- Tail only returns updates in Hist(x): x = 5, never the pending x = 7



Chain Replication: Consistency

Does Chain Replication maintain Linearizability? Yes!

Insight: *the tail is the commit point*

- An update reaches the tail only after every other replica has it
- Reads only go to the tail, so they only see Hist(obj): fully replicated updates, in one order

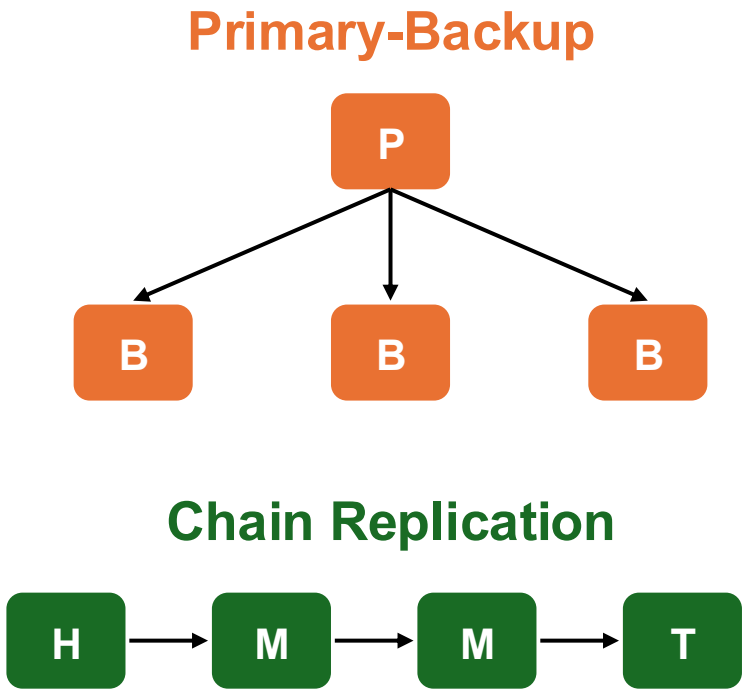
What if we read from the head instead?

- Could return x = 7 while it is still pending…
- …then a later read at the tail returns x = 5: reads go back in time, not linearizable!

CR vs. PB: Performance Setting

Same setting as last lecture: 4 servers in one datacenter

- Client-server RTT: 60ms (30ms each way)
- Server-server RTT: 1ms
- Log append: ~0ms;
- Apply to state machine: ~0ms
- Each server can handle 100 requests/sec



CR vs. PB: Write Latency

What is the latency of a single write?

Primary-backup:

- 30 (to primary) + 0 (log) + 1 (all backups, in parallel) + 0 (apply) + 30 (reply) = **61ms**

Chain replication:

- 30 (client to head) + 0 (head logs)
- 3 hops down the chain, one after another: 3 × (0.5 one-way + 0 log) = 1.5
- 0 (tail applies) + 30 (tail replies to client)
- Total: **61.5ms**: slightly worse, the hops are sequential, not parallel

CR vs. PB: Write Throughput

What is the bottleneck for writes? How many writes/sec?

Primary-backup: primary receives 1 + 3 = 4 requests per write → **25 writes/s**

Chain replication: work is spread down the chain

- Head: 1 (client request)
- Middle: 1 (from predecessor)
- Tail: 1 (from predecessor)
- Bottleneck is 1 units → 100/1 = **100 writes/s**: 4× primary-backup!

Simplification: ignoring acks flowing back up the chain (off the critical path, can be batched)

In-Class Exercises

- Draw throughput-latency graphs to compare CR and PB:
  - When replicas have 80ms RTTs between them (geo-replication)
  - When the throughput bottleneck is the 400 Gbps links in/out of each server and each request include 1GB of data (latency is the same)

Chain Replication: Extra Details

Covered the basics of chain replication, some pieces missing:

- How do we handle failures? Need a *coordinator*
- Head fails: Make the successor the new head
- Tail fails: Make the predecessor the new tail

Fault tolerance: Still f+1!

Some more details in paper!

Asides if we have time

- Sharded Primary-Backup
- CRAQ / 3FS / DeepSeek

Conclusion

Chain replication arranges replicas in a chain: writes enter at the head, reads are served by the tail

The tail is the commit point, so reads at the tail are strongly consistent

Same fault tolerance as primary-backup: f+1 replicas tolerate f failures

Our throughput-latency mental model shows the tradeoff: slightly higher write latency, much higher throughput