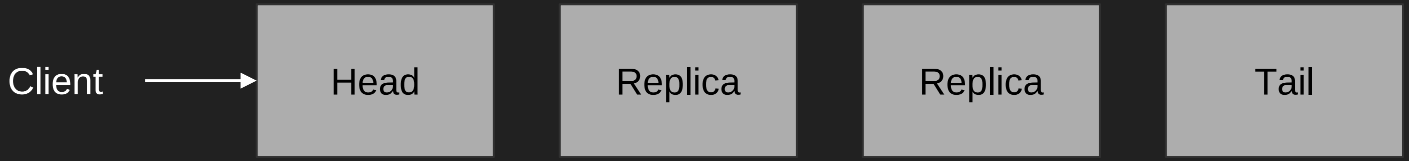
Chain Replication

Background

Invented in 2004, chain replication is a very interesting way to achieve both strong consistency as well as availability, while still maintaining decently high throughput!

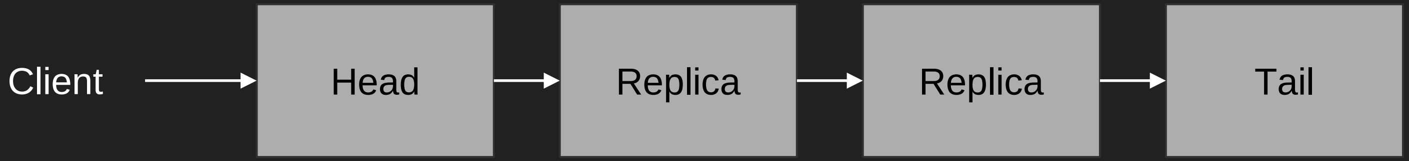
It can be modified, using ideas from the CRAQ paper, in order to provide even better read throughput.

Chain Replication Writes



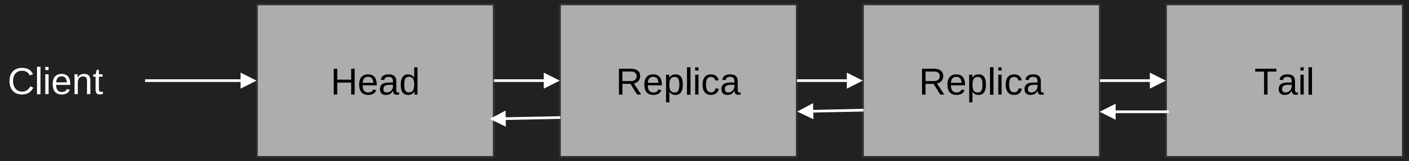
1) Client performs write to head of the list

Chain Replication Writes



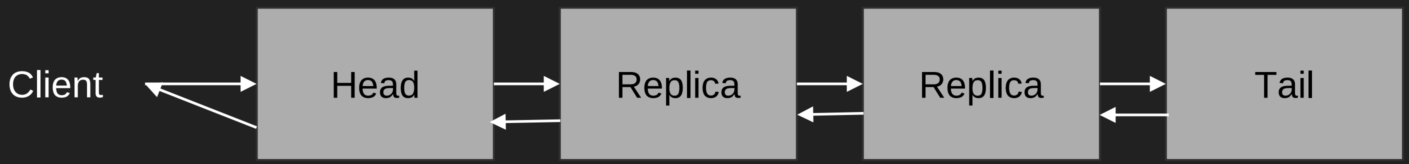
1) Client performs write to head of the list
2) Write is passed through the chain of replicas until it reaches the tail

Chain Replication Writes



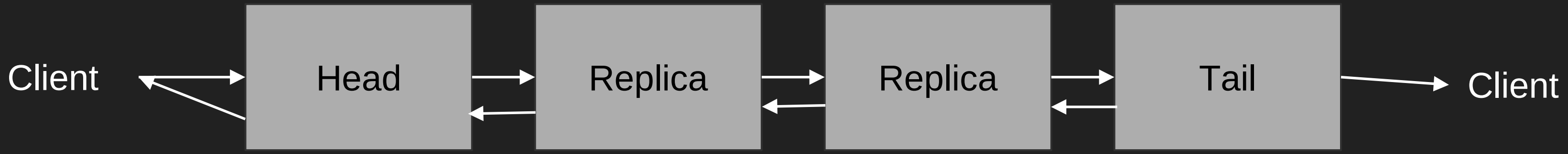
1) Client performs write to head of the list
2) Write is passed through the chain of replicas until it reaches the tail
3) Acknowledgements passed back from the tail all the way to the head

Chain Replication Writes



1) Client performs write to head of the list
2) Write is passed through the chain of replicas until it reaches the tail
3) Acknowledgements passed back from the tail all the way to the head
4) One head receives the acknowledgement, it alerts client that write was successful

Chain Replication Reads



All reads are made from the tail, which ensures strong consistency! Recall that all writes are acknowledged as successful once they reach the tail node and propagate back, so the tail should have up to date data!

Chain Replication Failover

A chain replication schema generally uses an external coordination service (using some sort of fault tolerant consensus algorithm) to detect node failures:

- If a head is believed to have failed, promote the next replica to head
  - May lead to some lost uncommitted writes from the head
- If the tail is believed to have failed, start executing reads from the replica behind it in the list
- If any middle node has failed, remove it from the chain
  - Requires each node keeping track of which writes that it has processed so that the node behind the failed middle node can update the crashed node’s successor

Chain Replication Analysis

Pros:

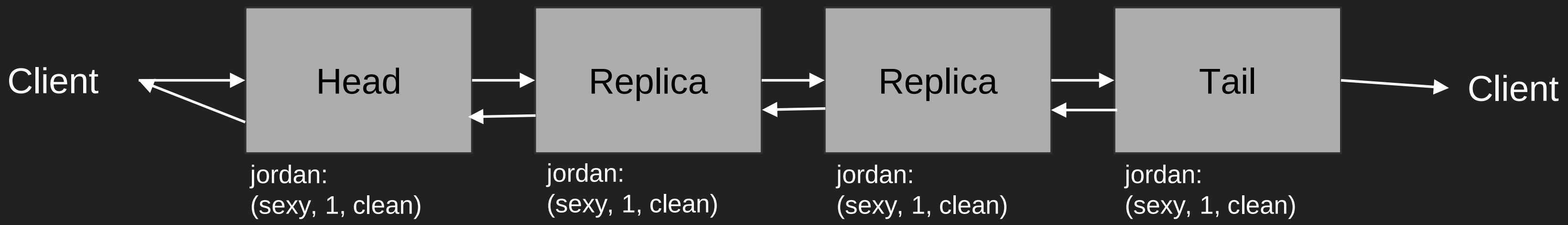
- Strong consistency/linearizability achieved through a total ordering of writes
- Writes less expensive than single leader replication because head only has to replicate them to one node, as opposed to potentially many (same happens in consensus algorithms)

Cons:

- Read throughput is limited to just one node (unless sharded)
- One slow node in the chain can cripple write speeds (still not very fault tolerant, even if the node will eventually be removed from the chain)

CRAQ

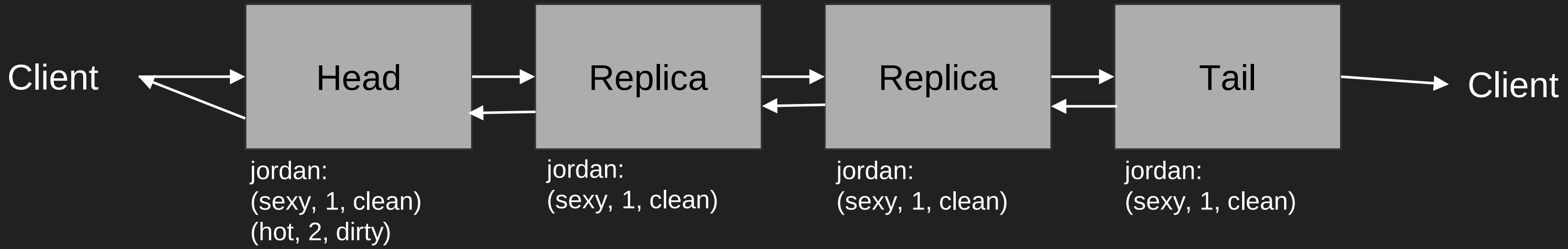
Modification of chain replication to increase read throughput while ensuring strong consistency!



1) System starts in a consistent state

CRAQ

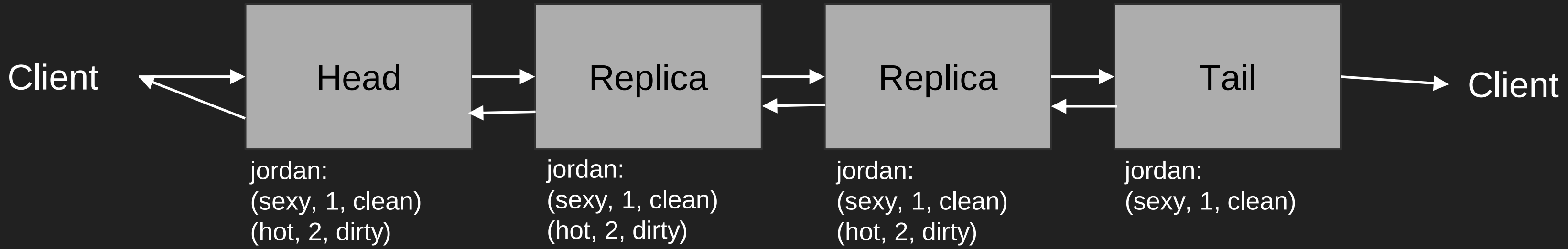
Modification of chain replication to increase read throughput while ensuring strong consistency!



1) System starts in a consistent state
2) Client performs write key: jordan, value: hot

CRAQ

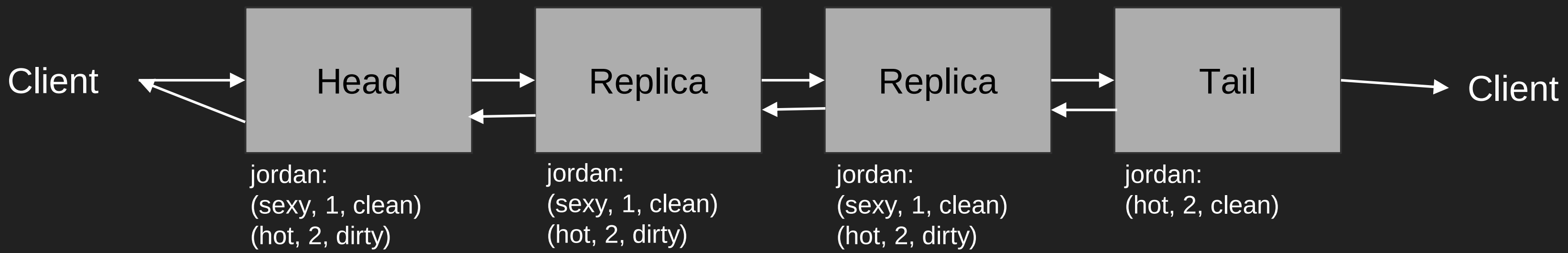
Modification of chain replication to increase read throughput while ensuring strong consistency!



1) System starts in a consistent state
2) Client performs write key: jordan, value: hot, each node increases version for key jordan
3) Until each replica gets an acknowledgement for the write, it holds in with value dirty

CRAQ

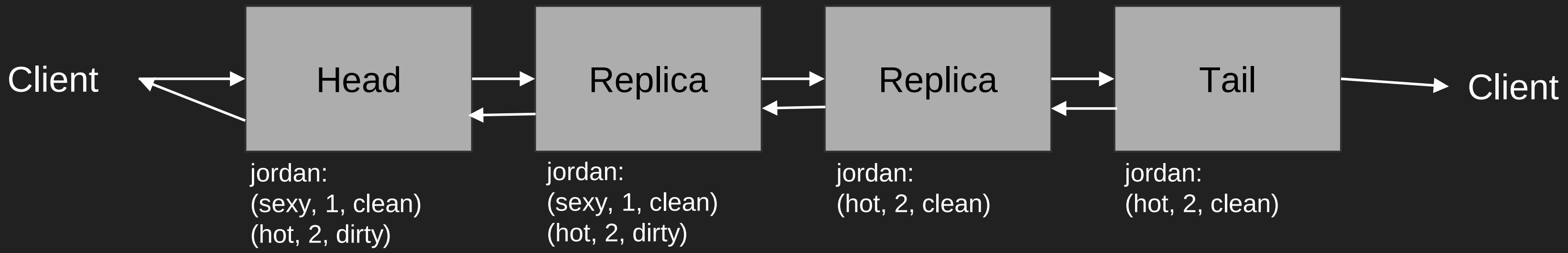
Modification of chain replication to increase read throughput while ensuring strong consistency!



1) System starts in a consistent state
2) Client performs write key: jordan, value: hot, each node increases version for key jordan
3) Until each replica gets an acknowledgement for the write, it holds in with value dirty
4) Once write reaches the tail, the tail marks it as clean locally and acknowledges the write

CRAQ

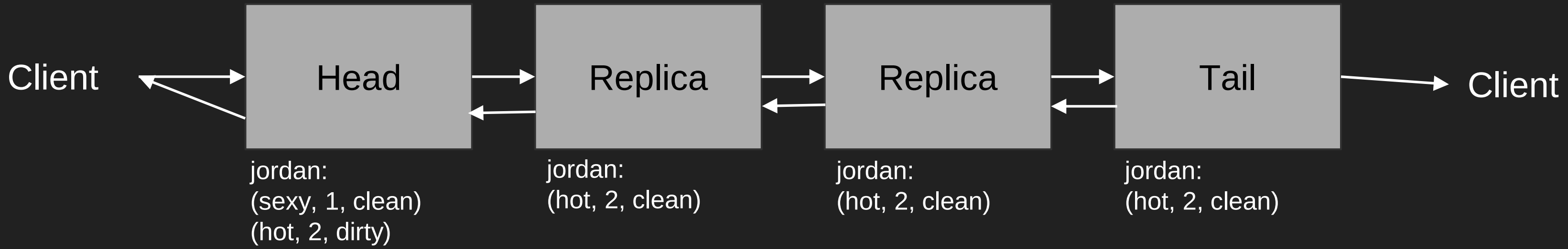
Modification of chain replication to increase read throughput while ensuring strong consistency!



1) System starts in a consistent state
2) Client performs write key: jordan, value: hot, each node increases version for key jordan
3) Until each replica gets an acknowledgement for the write, it holds in with value dirty
4) Once write reaches the tail, the tail marks it as clean locally and acknowledges the write
5) As acknowledgements are passed back through, each replica marks the dirty write to clean and deletes the old version of the write

CRAQ

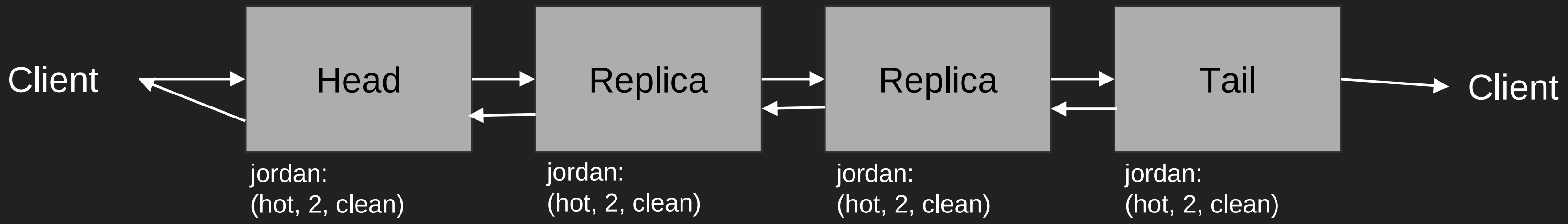
Modification of chain replication to increase read throughput while ensuring strong consistency!



1) System starts in a consistent state
2) Client performs write key: jordan, value: hot, each node increases version for key jordan
3) Until each replica gets an acknowledgement for the write, it holds in with value dirty
4) Once write reaches the tail, the tail marks it as clean locally and acknowledges the write
5) As acknowledgements are passed back through, each replica marks the dirty write to clean and deletes the old version of the write

CRAQ

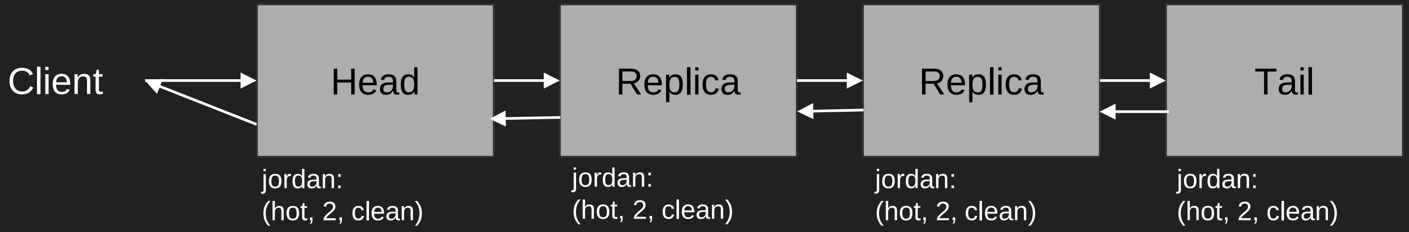
Modification of chain replication to increase read throughput while ensuring strong consistency!



1) System starts in a consistent state
2) Client performs write key: jordan, value: hot, each node increases version for key jordan
3) Until each replica gets an acknowledgement for the write, it holds in with value dirty
4) Once write reaches the tail, the tail marks it as clean locally and acknowledges the write
5) As acknowledgements are passed back through, each replica marks the dirty write to clean and deletes the old version of the write

CRAQ

Modification of chain replication to increase read throughput while ensuring strong consistency!

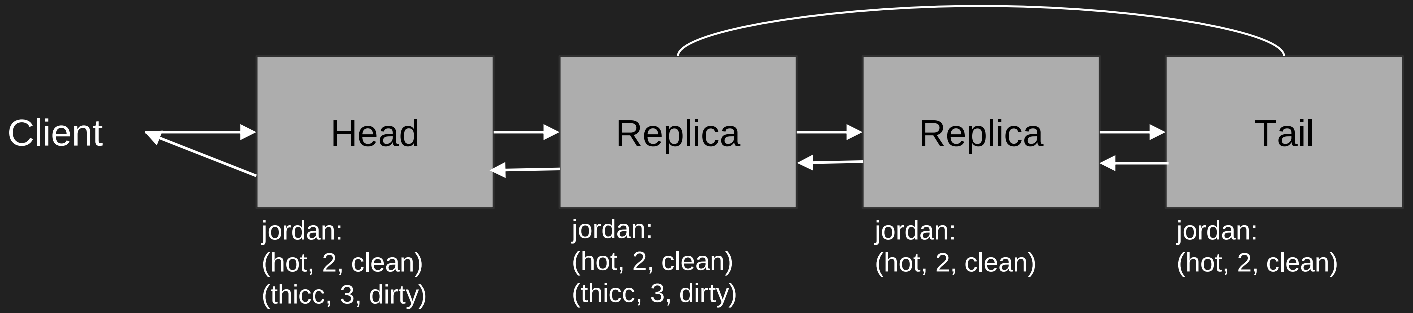


Reads: If read(k) goes to a replica where the highest version number available of key k has a clean label, the replica can safely return the corresponding value!

Read(jordan) from replica 2 = hot

CRAQ

Modification of chain replication to increase read throughput while ensuring strong consistency!



Note: Can't just return the highest clean value from each replica as it is possible the tail has committed the dirty value and the replica hasn't yet received an acknowledgement!!

Reads: If read(k) goes to a replica where the highest version number available of key k has a dirty label, the replica must first ask the tail what the current version number of the key is, and then respond accordingly.

Read(jordan) from replica 2 = hot

CRAQ Evaluated

In mostly read heavy workloads, the majority of reads will be clean and as a result read throughput should be scaled linearly to the number of replicas!

Even if there are a lot of writes, this can still improve read performance because the tail just returning version numbers is likely less expensive over the network than returning the full value of a key (especially if we are storing things like large files).

Ultimately, CRAQ still suffers from the same problems as chain replication in a single node acting as a bottleneck for the whole chain.

Chain Replication Conclusion

Chain replication is interesting in that it allows strong consistency while also displaying some degree of fault tolerance. While it is not the fastest replication scheme, it successfully reduces a lot of load on each node in the schema by limiting the amount of network calls that they make. CRAQ is a very interesting modification to chain replication that significantly increases write throughput!

Ultimately, it probably will not be overly useful to know about chain replication in a systems design interview, but you will look smart if you can mention it if strong consistency ever comes up!

https://www.youtube.com/watch?v=1uUcW-Mqg5o&ab_channel=MIT6.824%3ADistributedSystems

https://timilearning.com/posts/mit-6.824/lecture-9-craq/