CRAQ

High-Throughput Chain Replication for Read-Mostly Workloads

Jeff Terrace & Mike Freedman
Presented by Abrahim Ladha

Object Storage (Put/Get)

- Given an identifier, you can query an object
- Many websites don’t need complex joins
- Better scaling and speed for these services
- They achieve this with Eventual Consistency instead of Strong Consistency

Eventual Consistency

Manager

Replicas

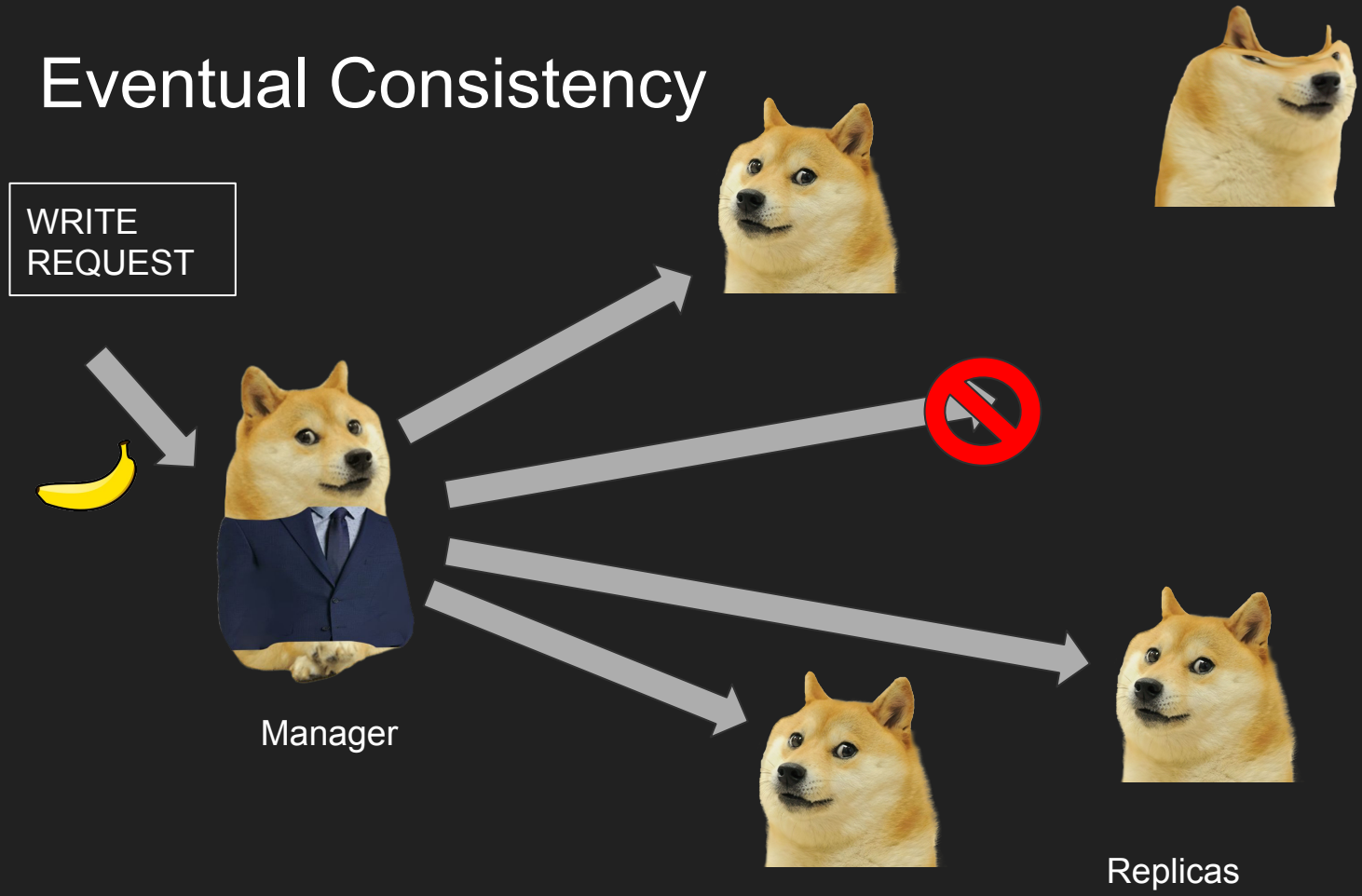
Eventual Consistency

WRITE REQUEST

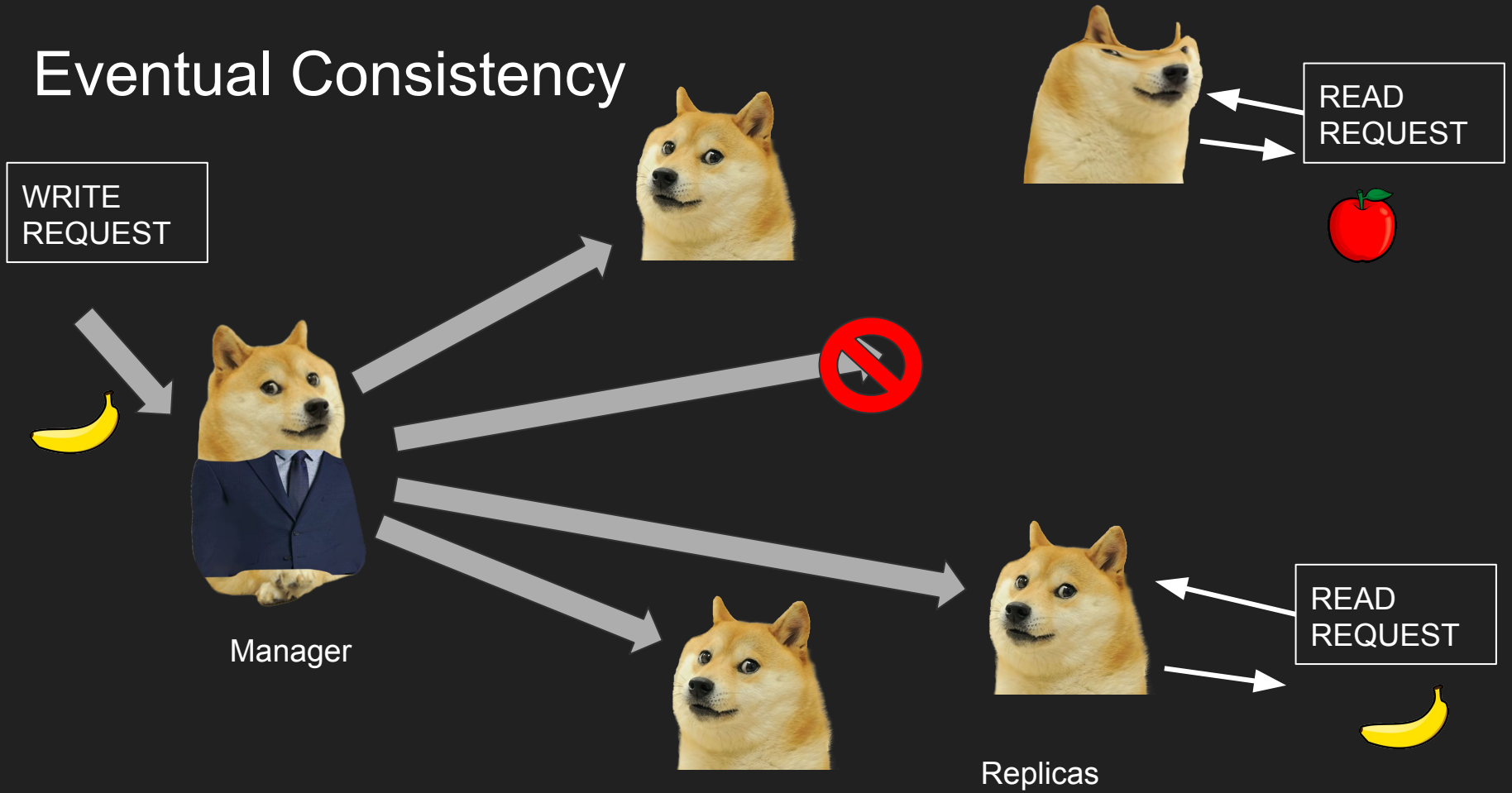
Manager

Replicas

Eventual Consistency



Eventual Consistency



Eventual Consistency

- Writes are ordered after commit
- Reads can be out of order, even stale.
- Lets object storage systems scale very well, with high throughput
- Hard for programmer to know what to do in failure, or detect sta
- Two successive reads could go back in time.

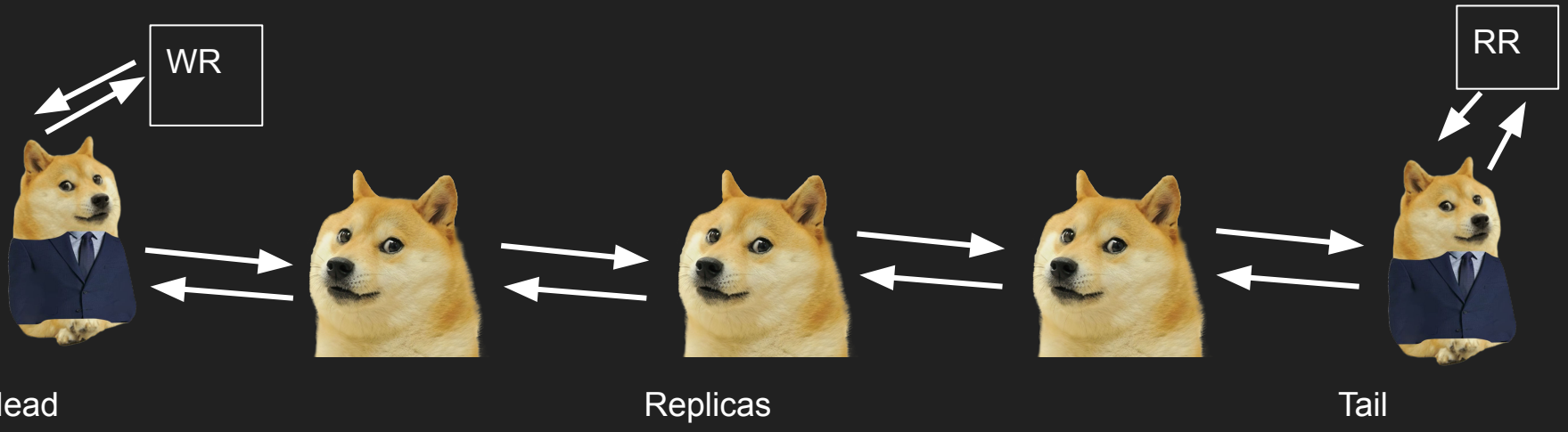
Strong Consistency

- Instead of the manager just sending out the messages, the manager and the replicas perform an agreement protocol, something like two phase commit.
- Reads and Writes strictly ordered
- Easier programming.
- Expensive to implement
- Does not scale as well
- Potential availability sacrifice

Chain Replication [van Renesse & Schneider, OSDI 2004]

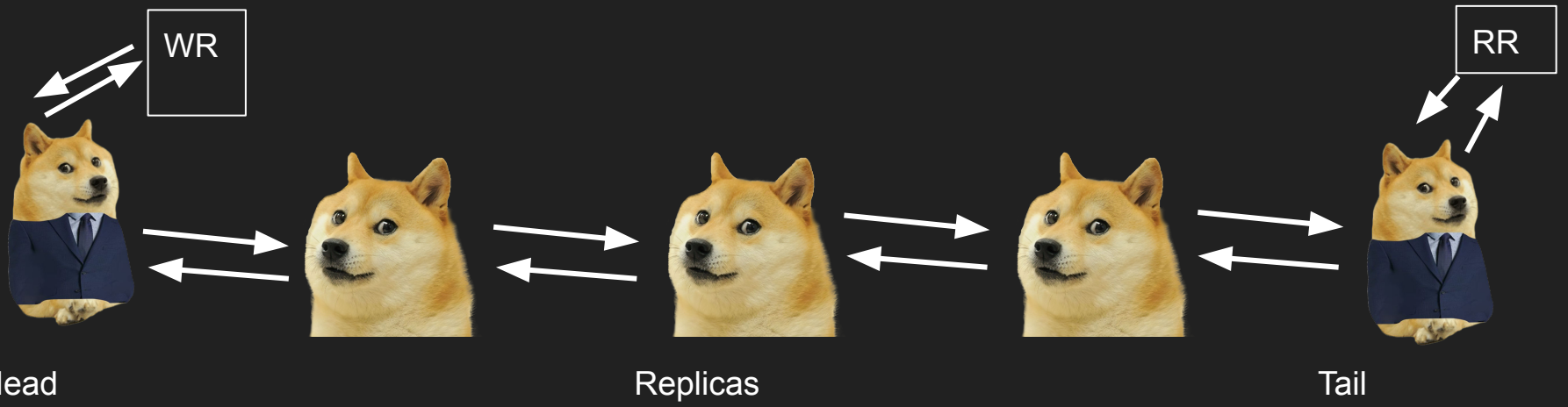
- Writes all go to the head
- Propagate serially
- Once it reaches tail, it is committed
- Acknowledgement is propagated back to the head

- Read requests all go to the head
- This gives us strong consistency
- Tail decides ordering of many reads

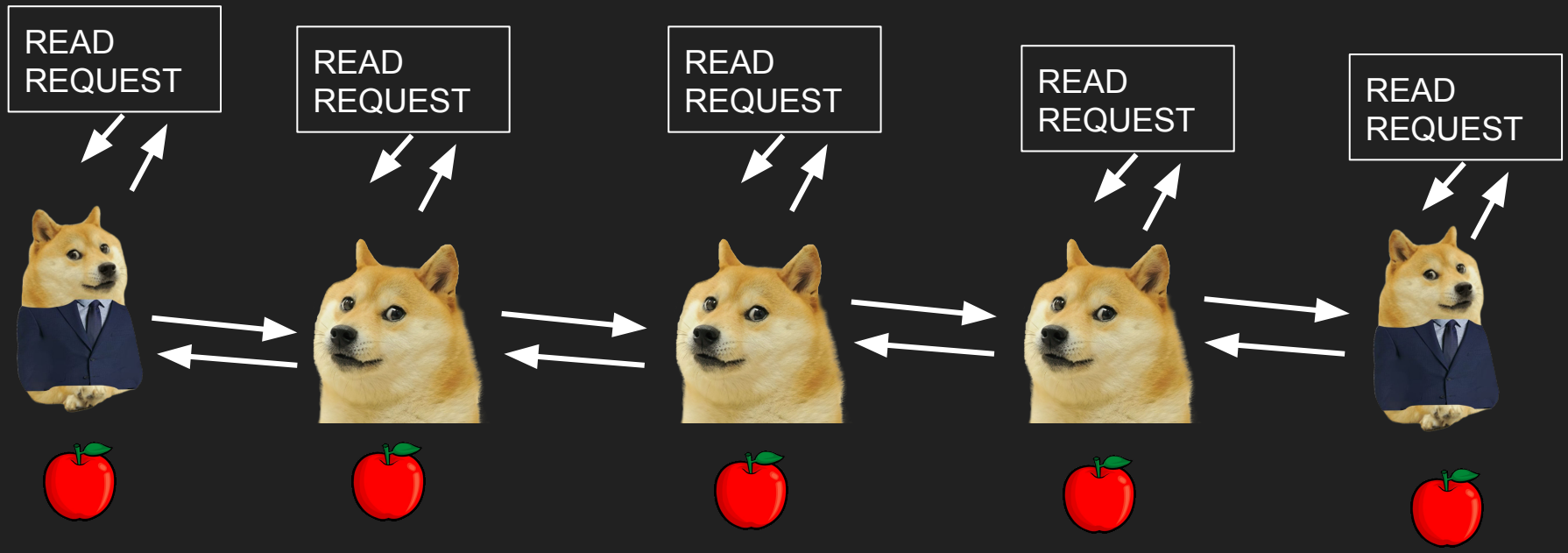


Chain Replication [van Renesse & Schneider, OSDI 2004]

- Strong Consistency and replication is simple.
- Increased write throughput because writes are pipelined down the chain
- Low read throughput because all reads have to go to the tail
- CRAQ increases the read throughput
- Motivation from industry: Many applications are very read heavy

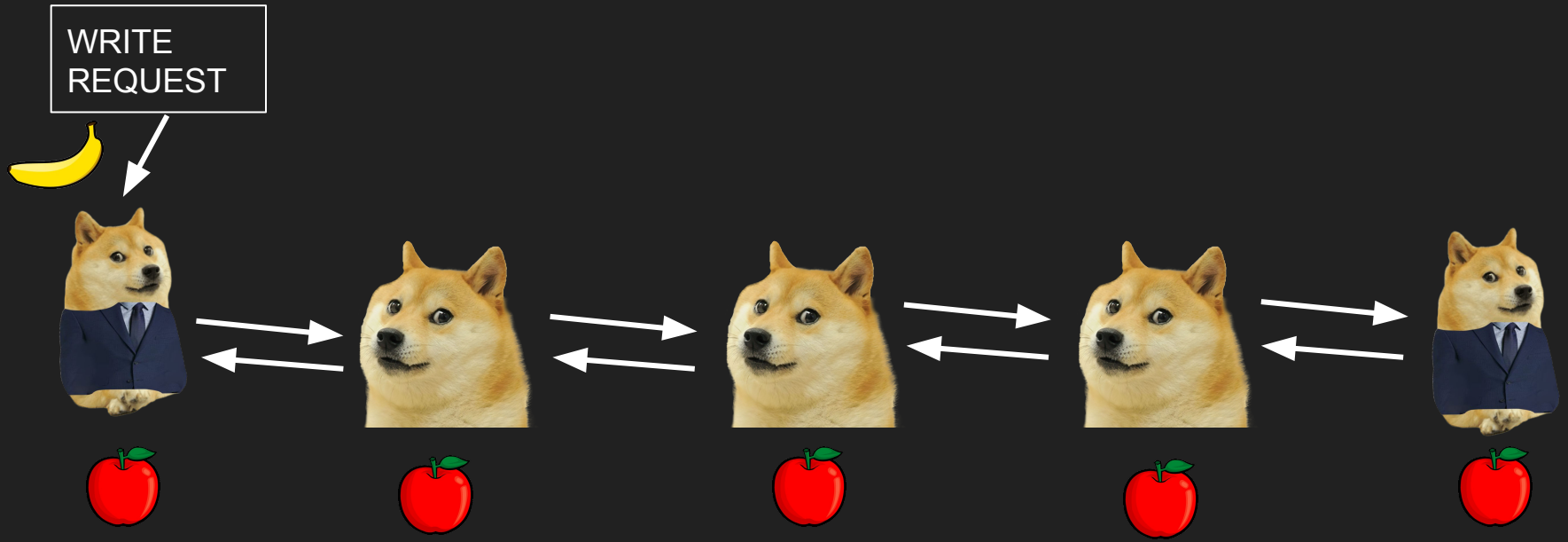


Clean Reads

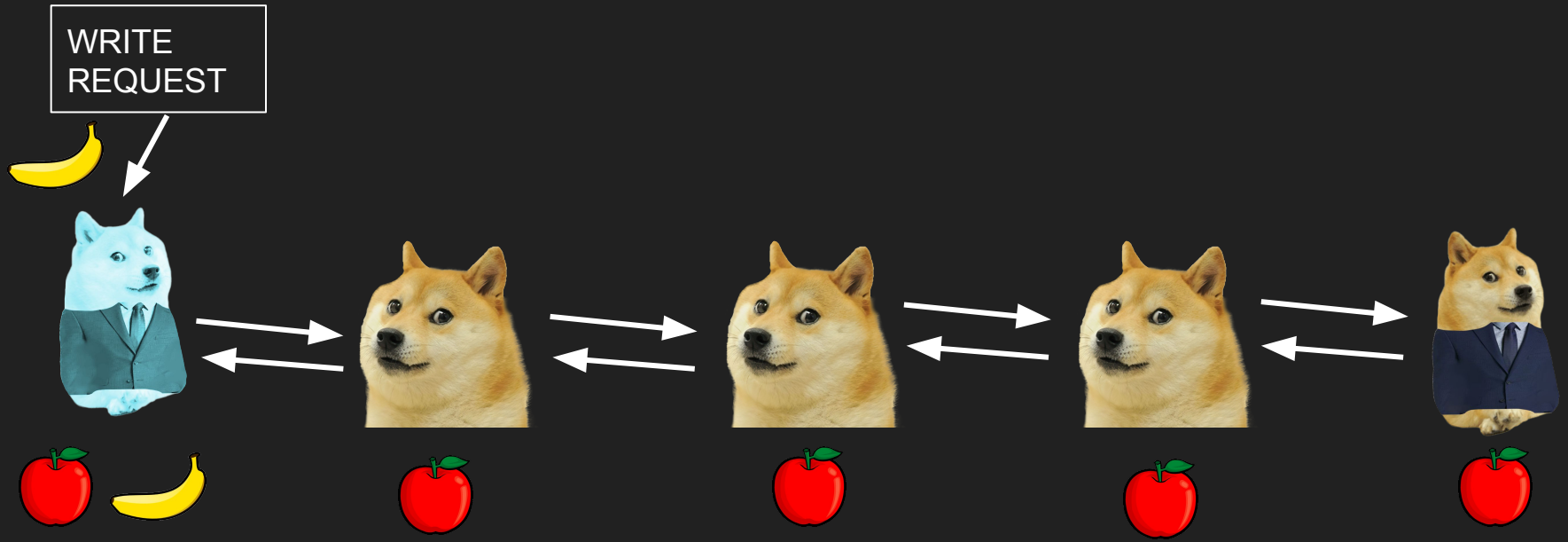


- Two states per object, clean, and dirty
- If object is clean, then we can read anywhere!

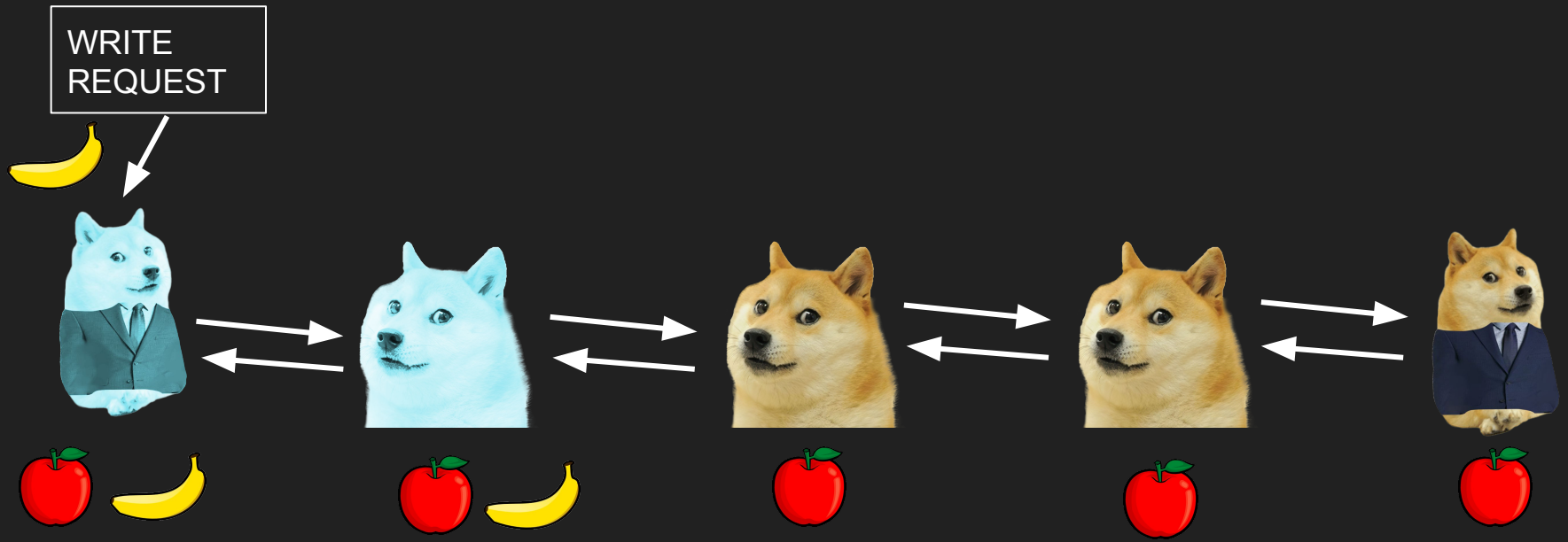
Writes



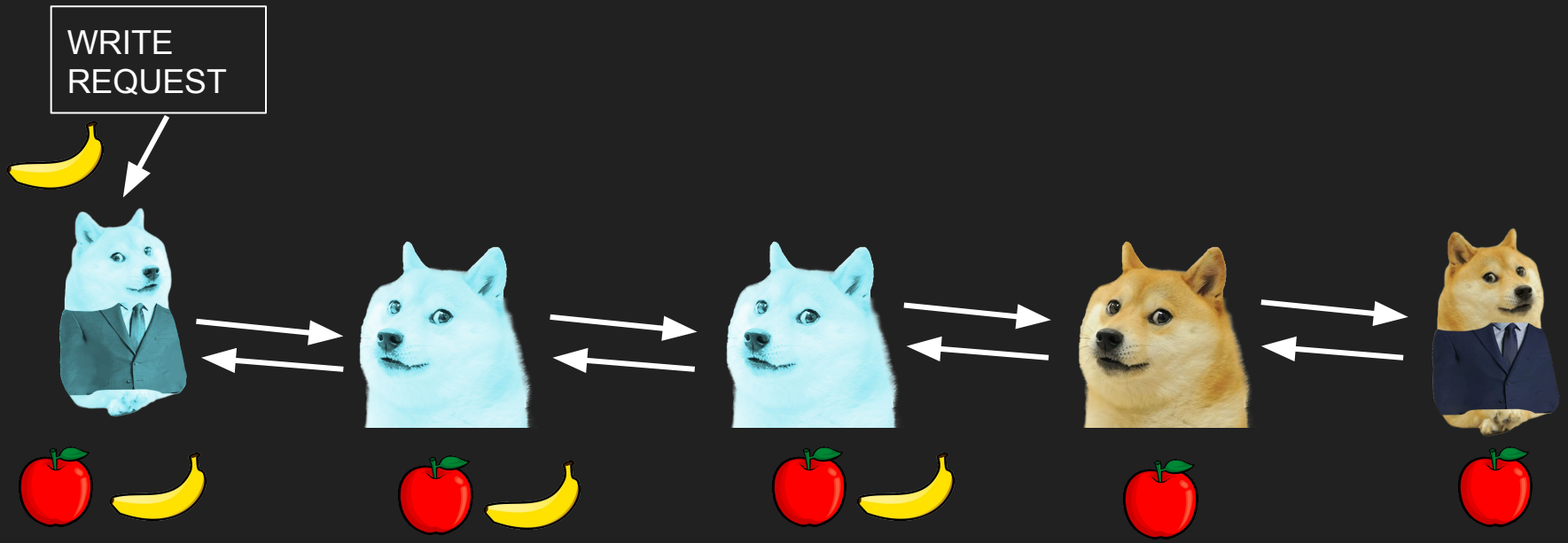
Writes



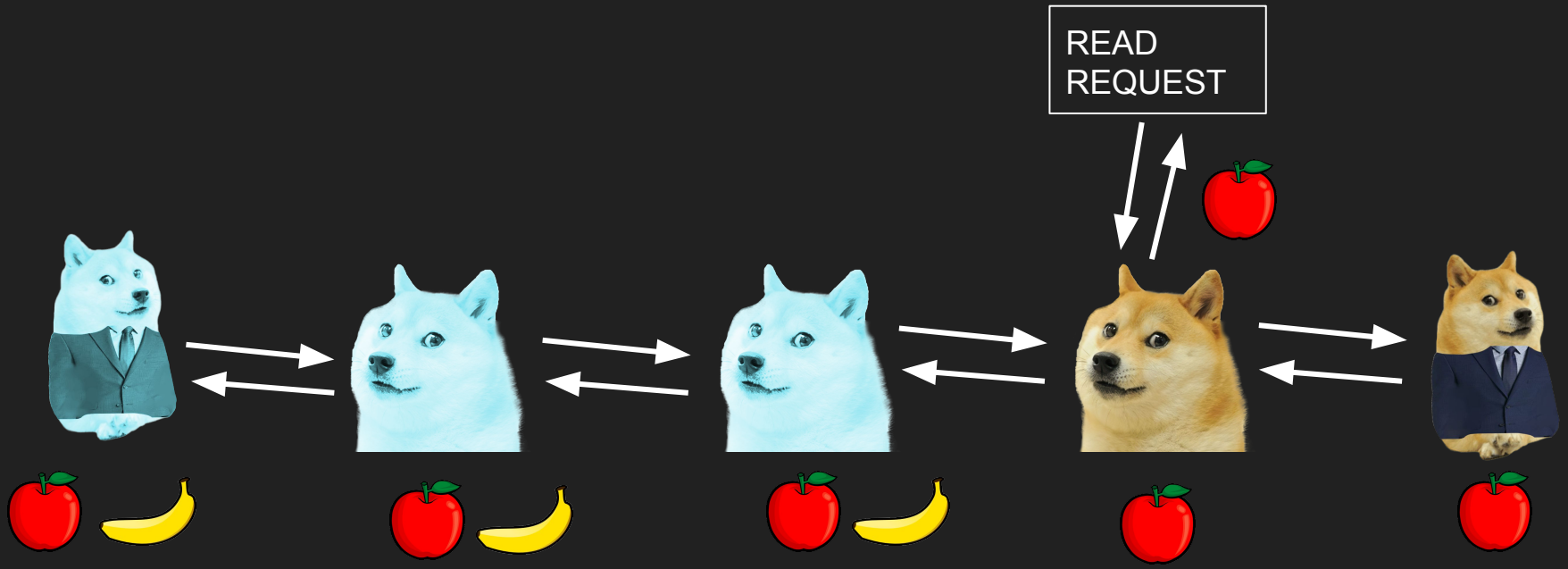
Writes



Writes

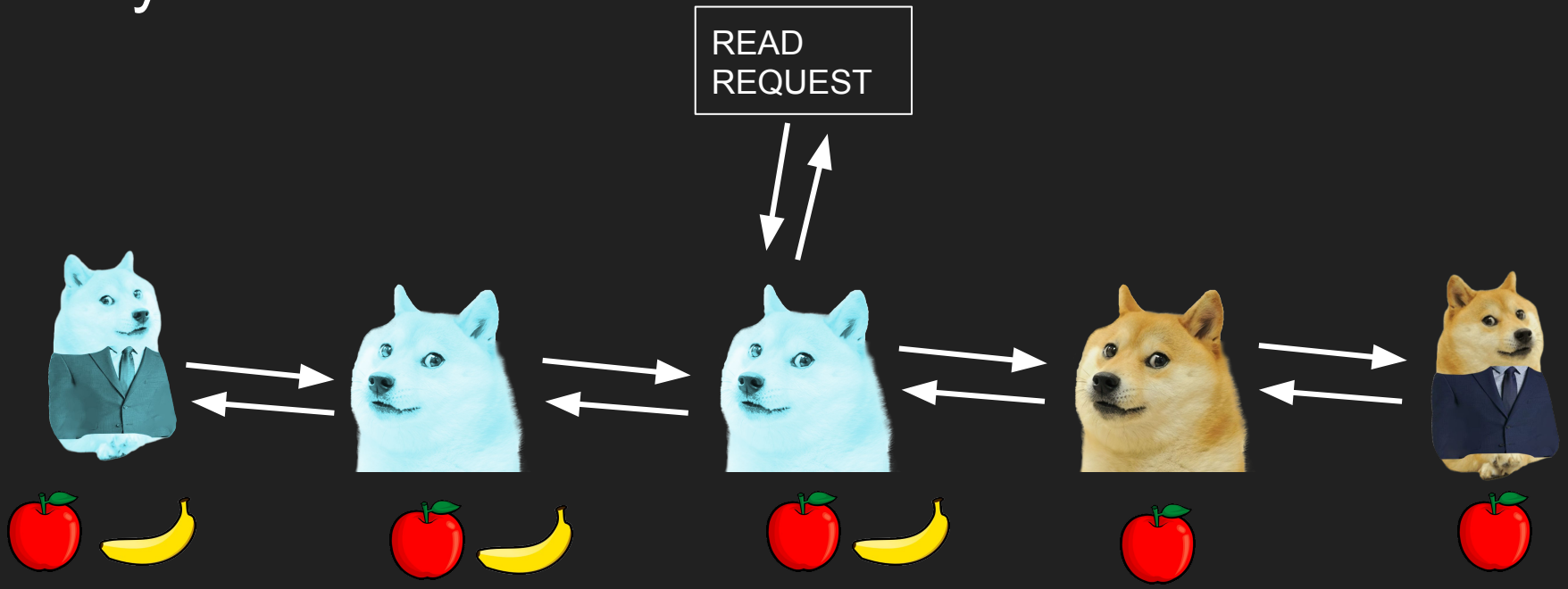


Clean Read



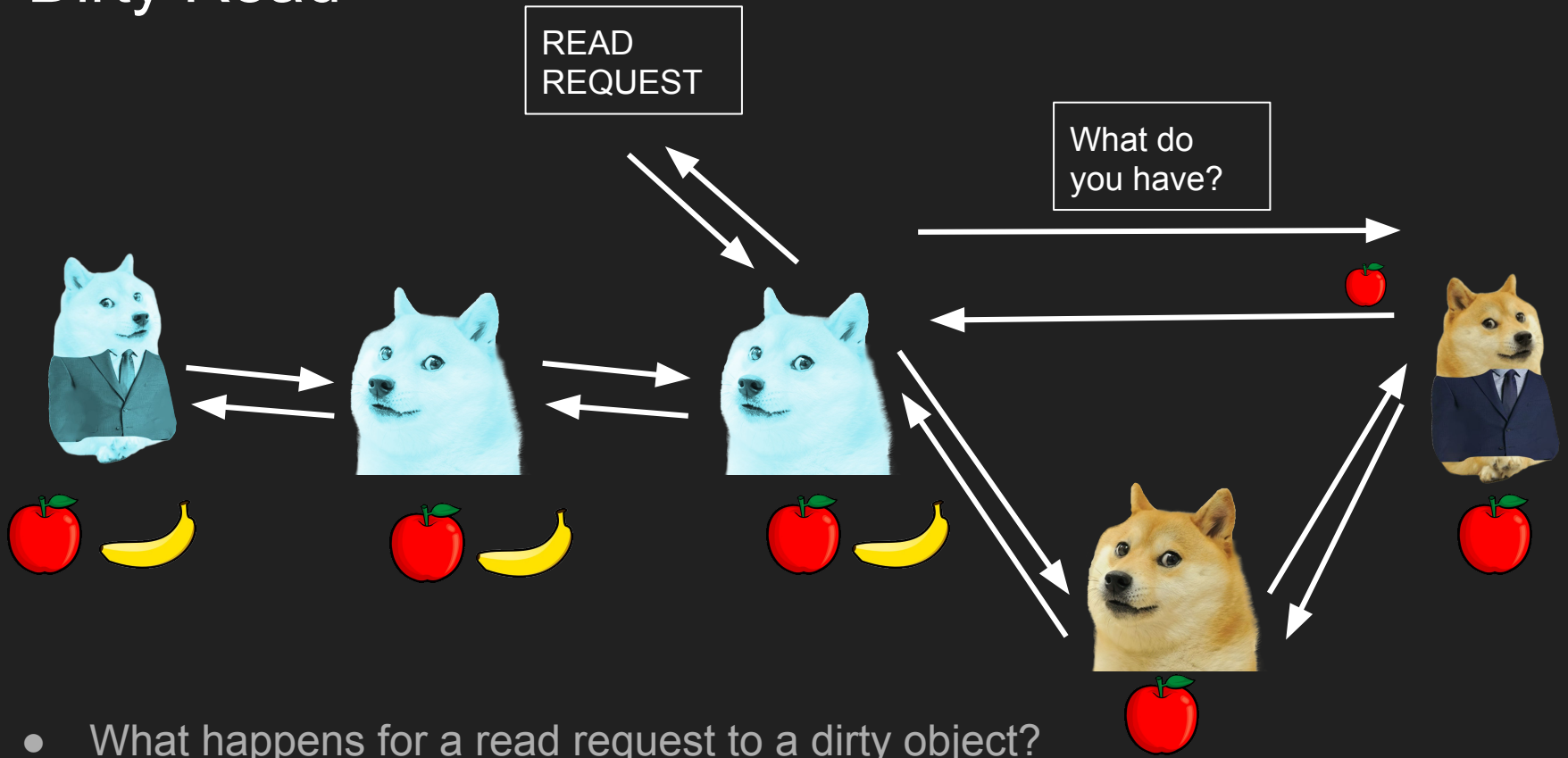
- If read request is made to clean replica, then it can respond

Dirty Read



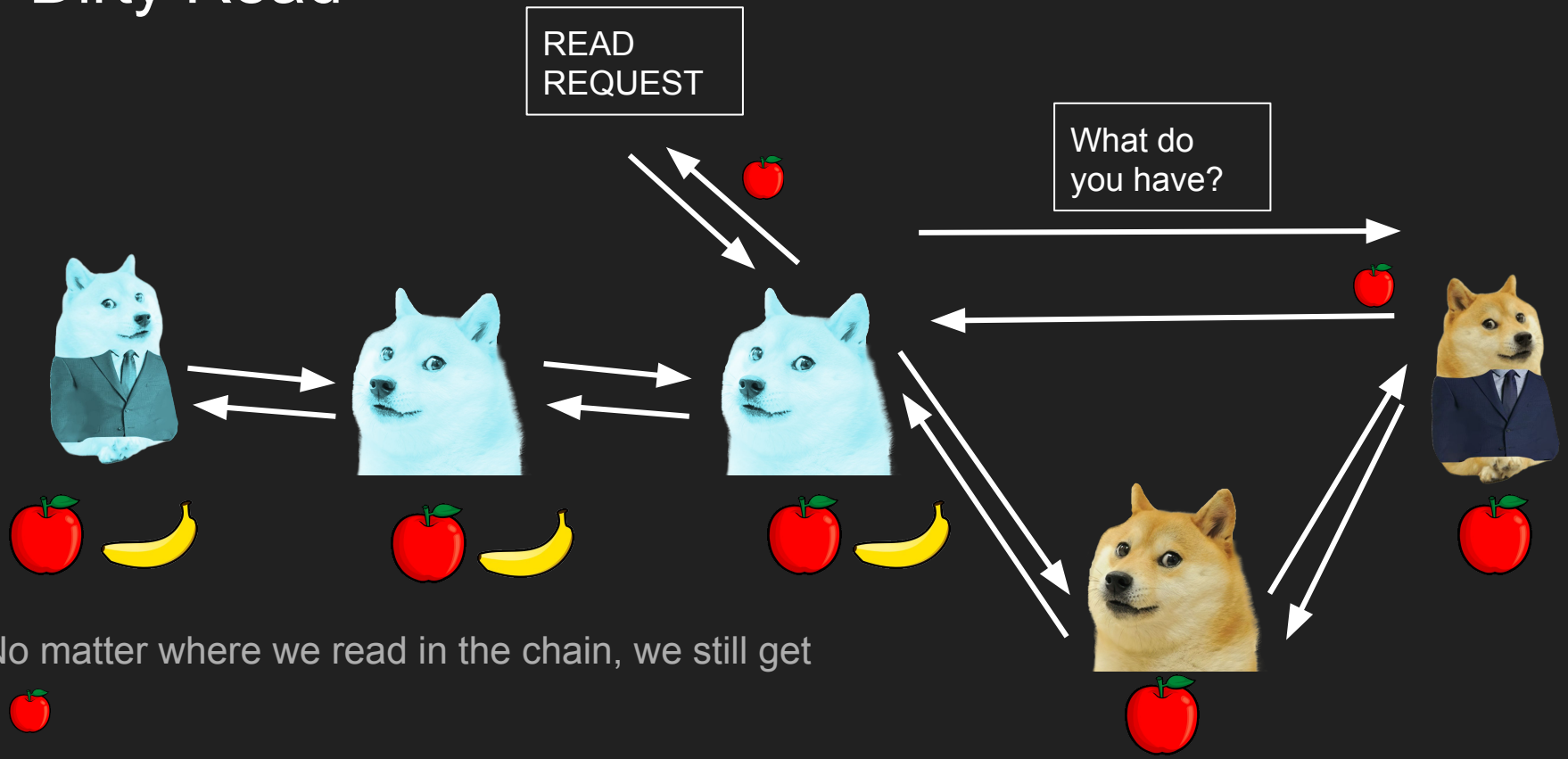
- What happens for a read request to a dirty object?

Dirty Read



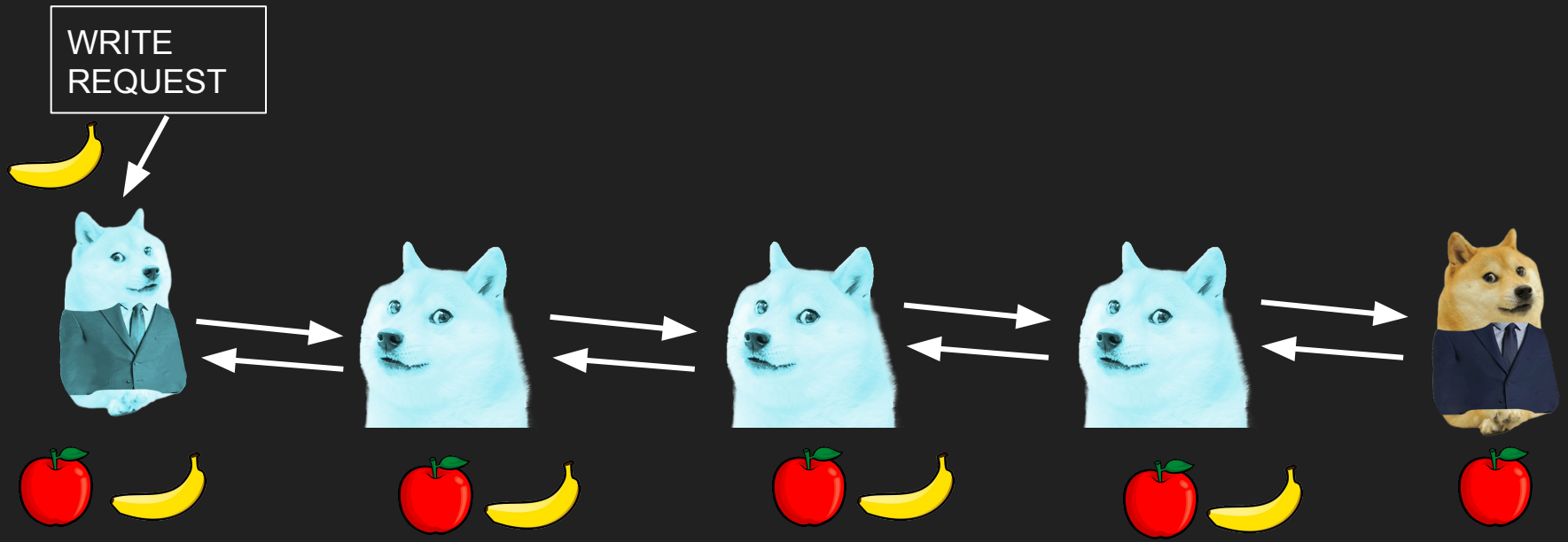
- What happens for a read request to a dirty object?

Dirty Read

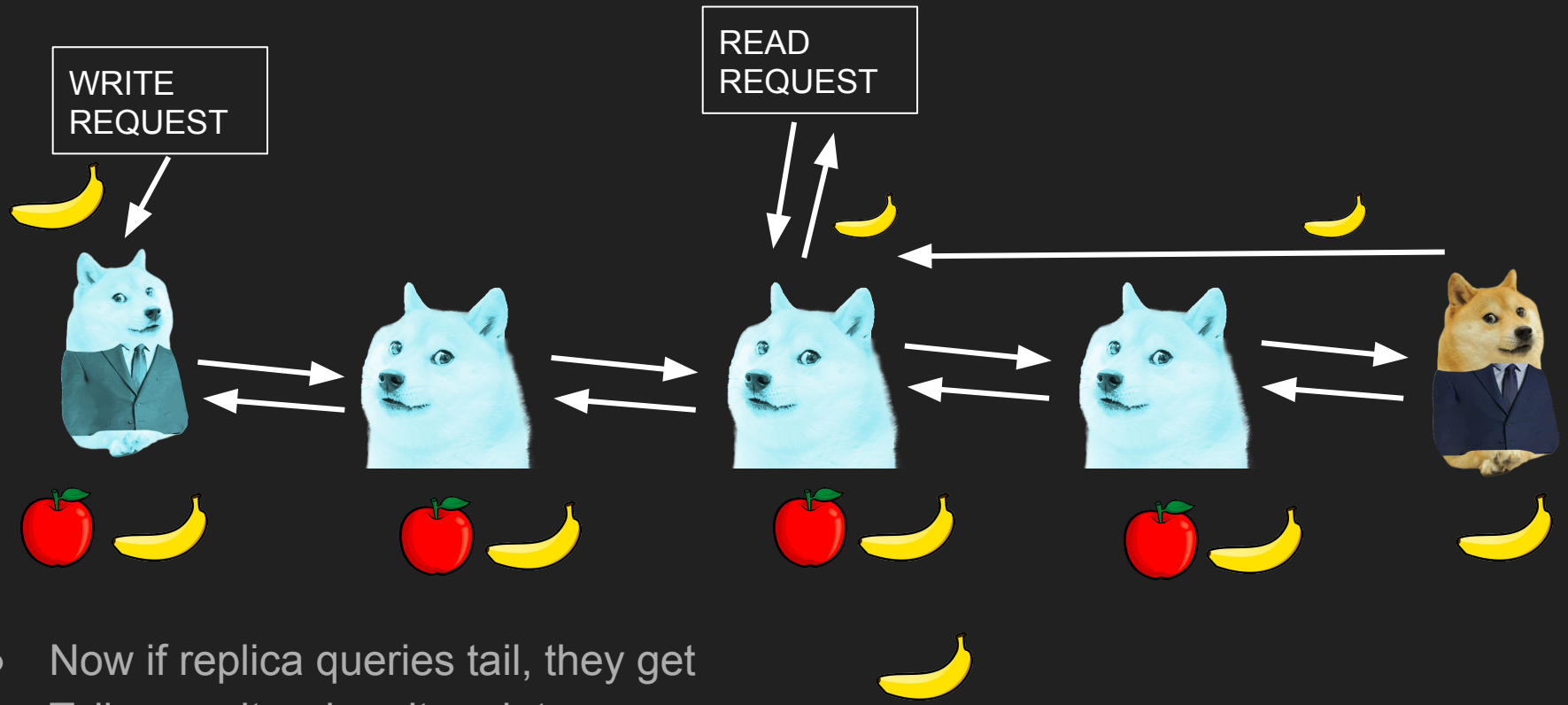


No matter where we read in the chain, we still get

Writes

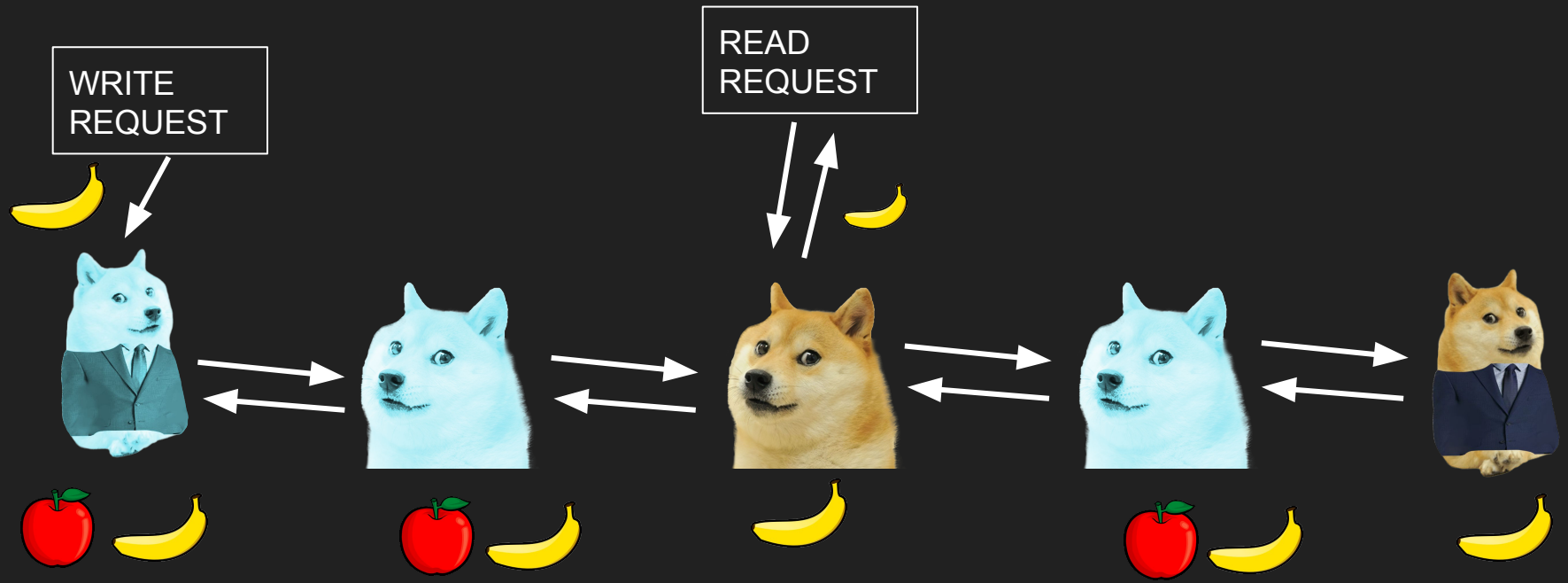


Writes



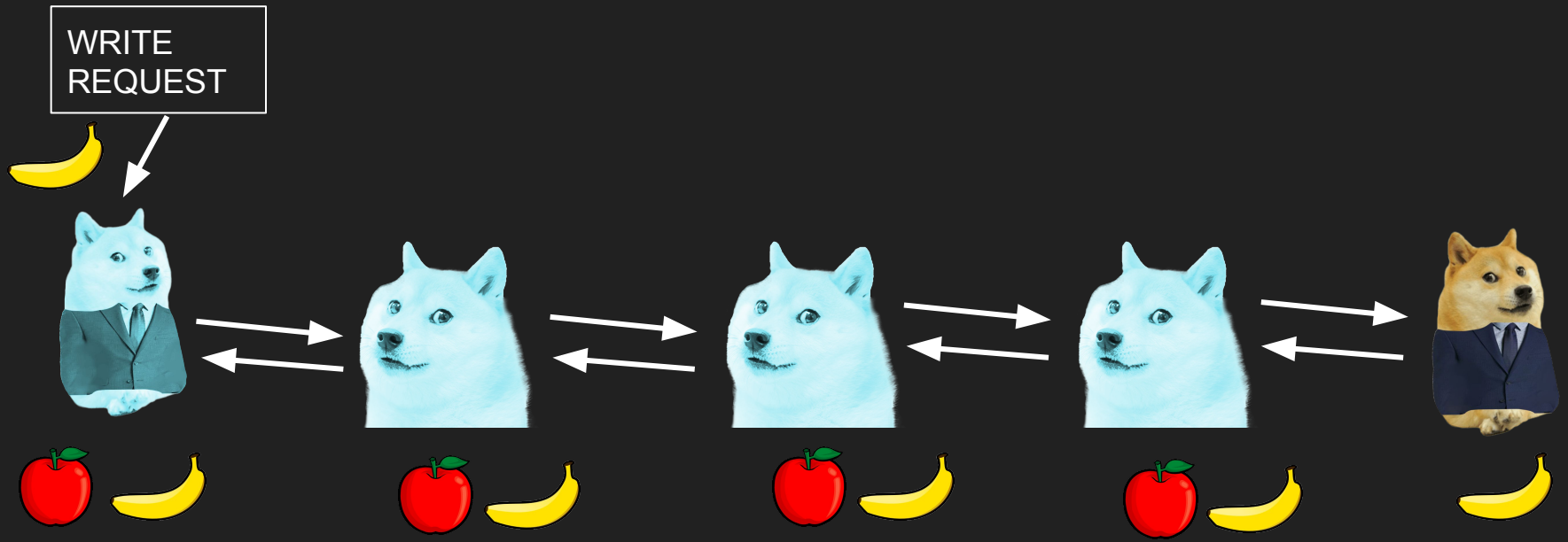
- Now if replica queries tail, they get
- Tail commits when it updates

Writes

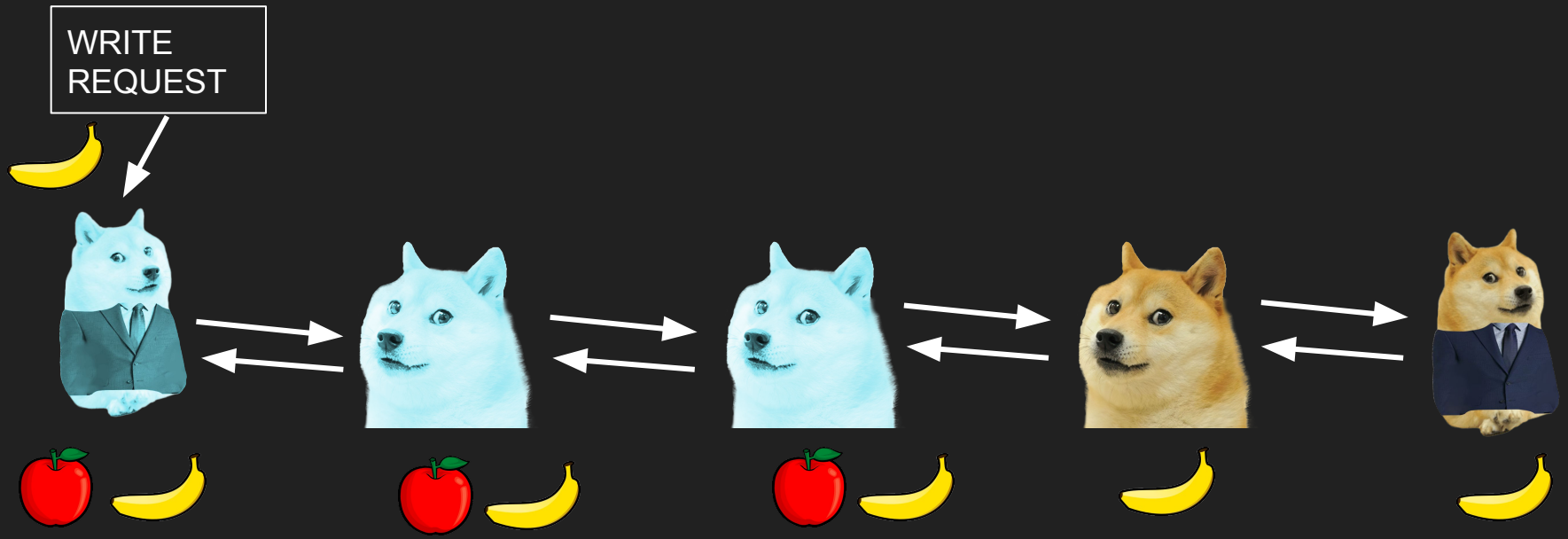


- Lazy model: Only update replica on read
- Optimization: Propagate backwards to update earlier

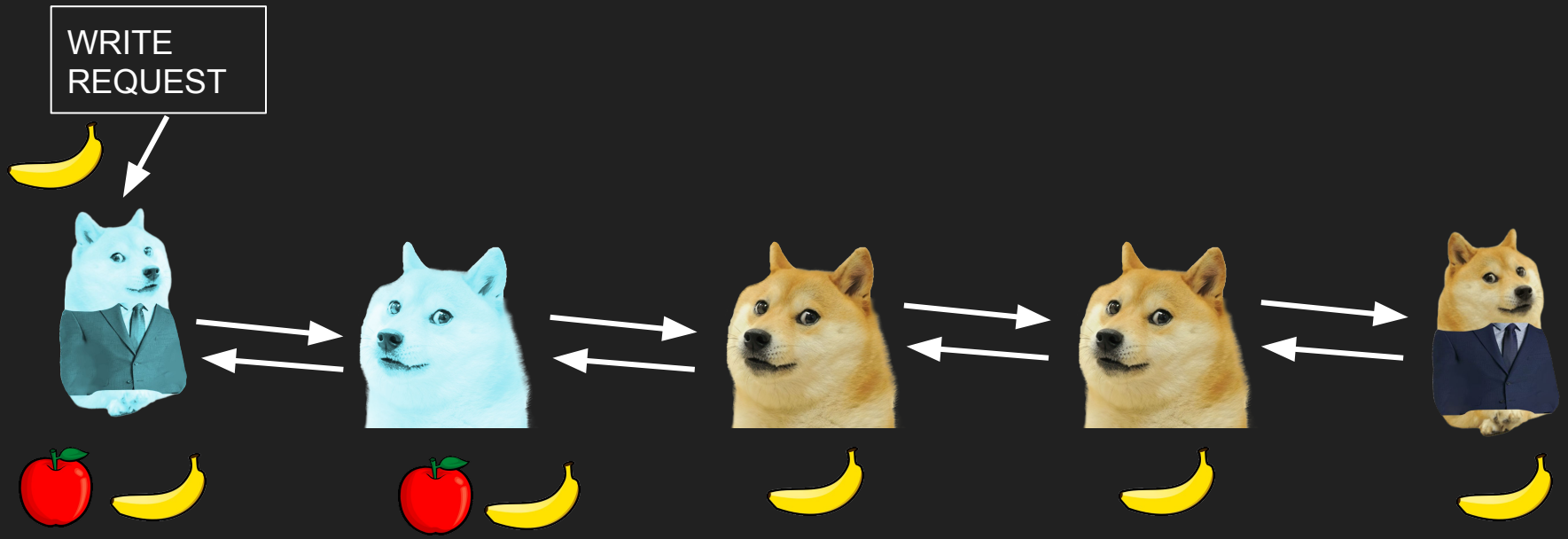
Writes



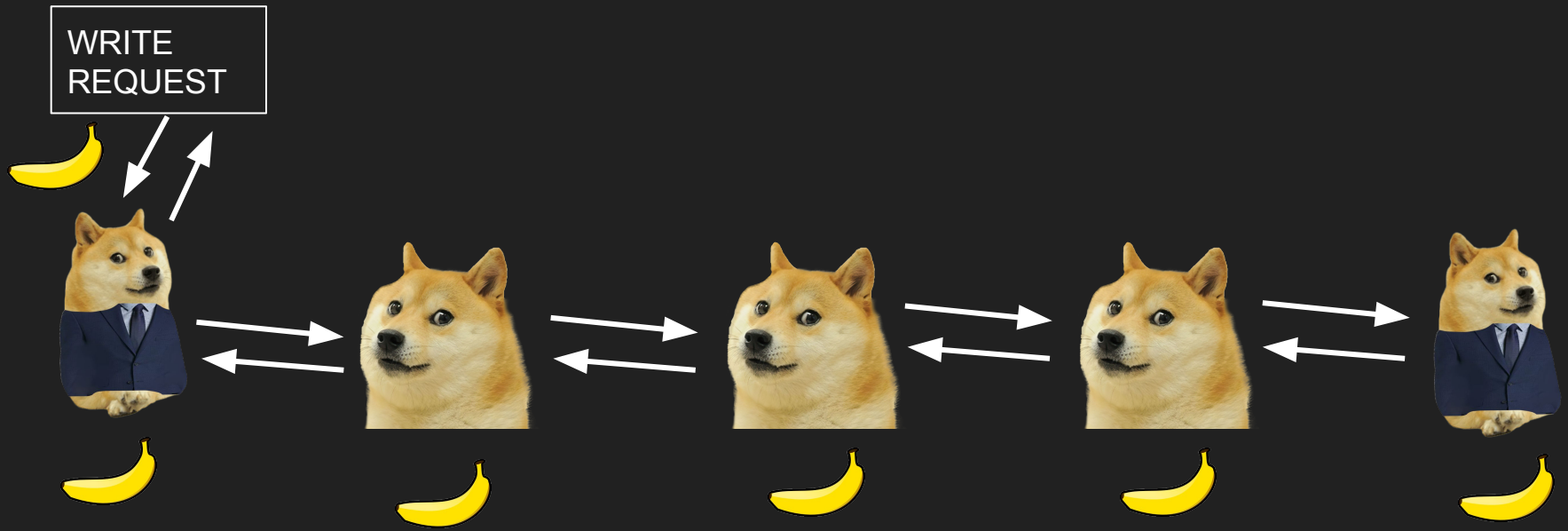
Writes



Writes



Writes



- Once head updates, can respond to write request

Optimizations

- Each chain forms a group to multicast, tail sends acknowledgement to whole chain
- Suppose we have a large write. The head can propagate it to entire chain, each replica stores and marks it as temporary, then head propagates small message to mark as ready

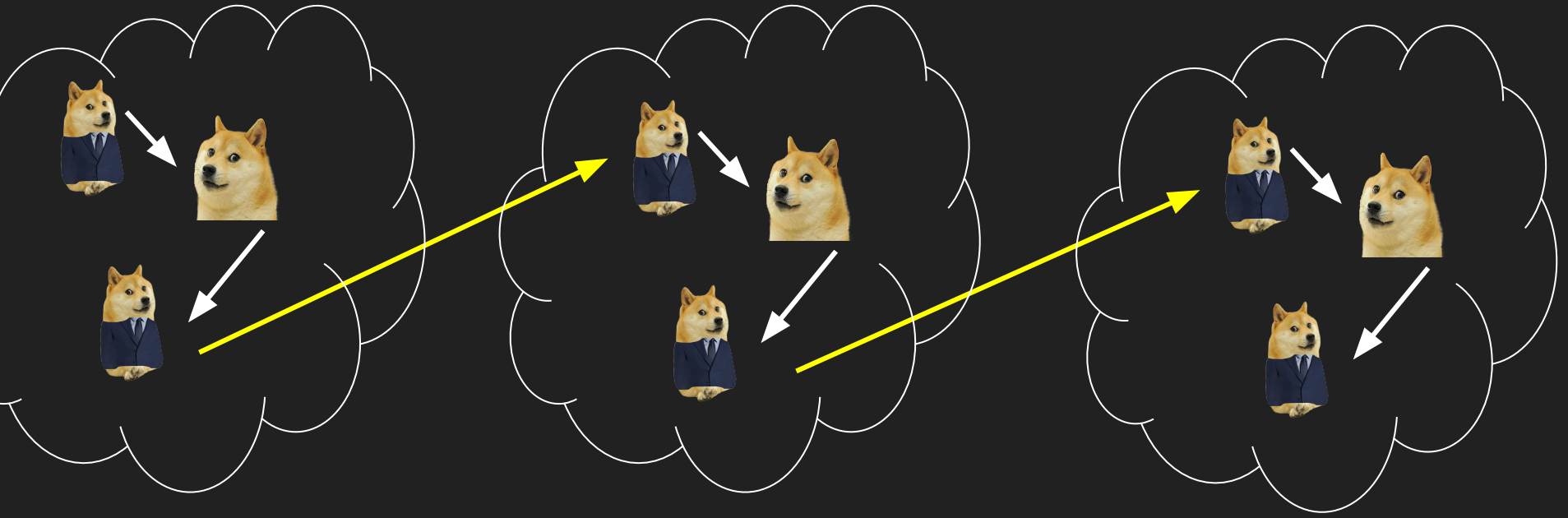
From Chain Replication

- Maintains strong consistency
- Simple to replicate
- Increases write throughput

Added by CRAQ

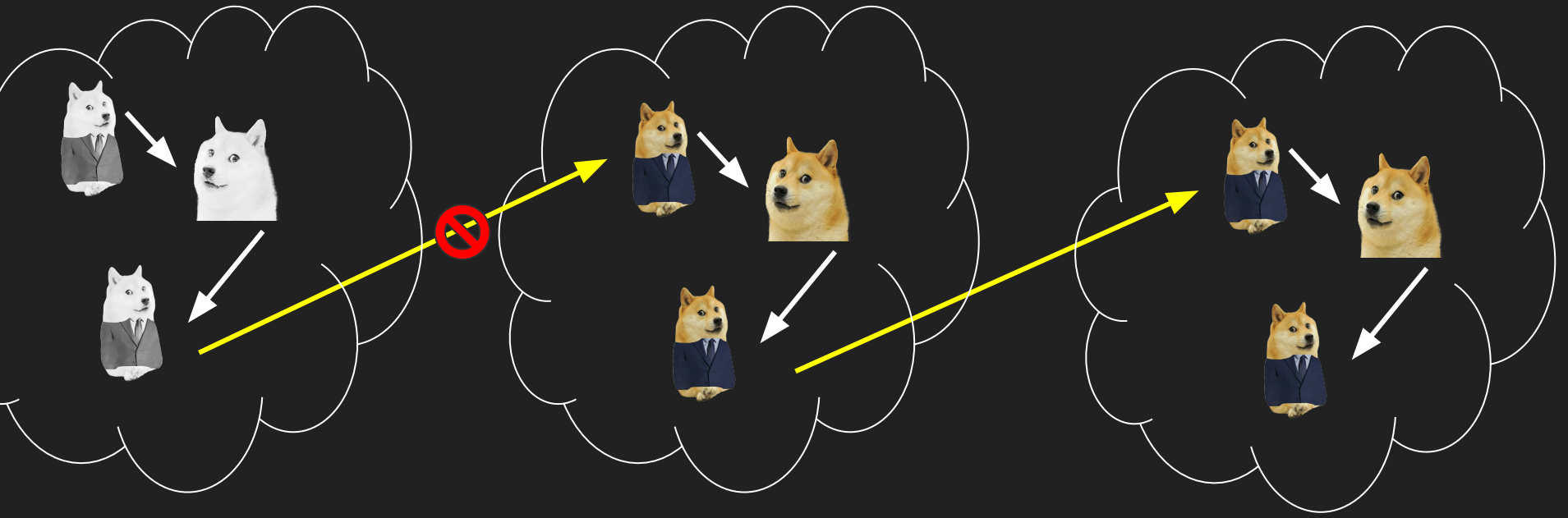
- Read throughput scales for read mostly workloads
  - If chain is mostly read, then throughput should scale linearly with the chain length
- Supports Eventual Consistency when strong consistency isn’t needed

Multi Datacenter CRAQ



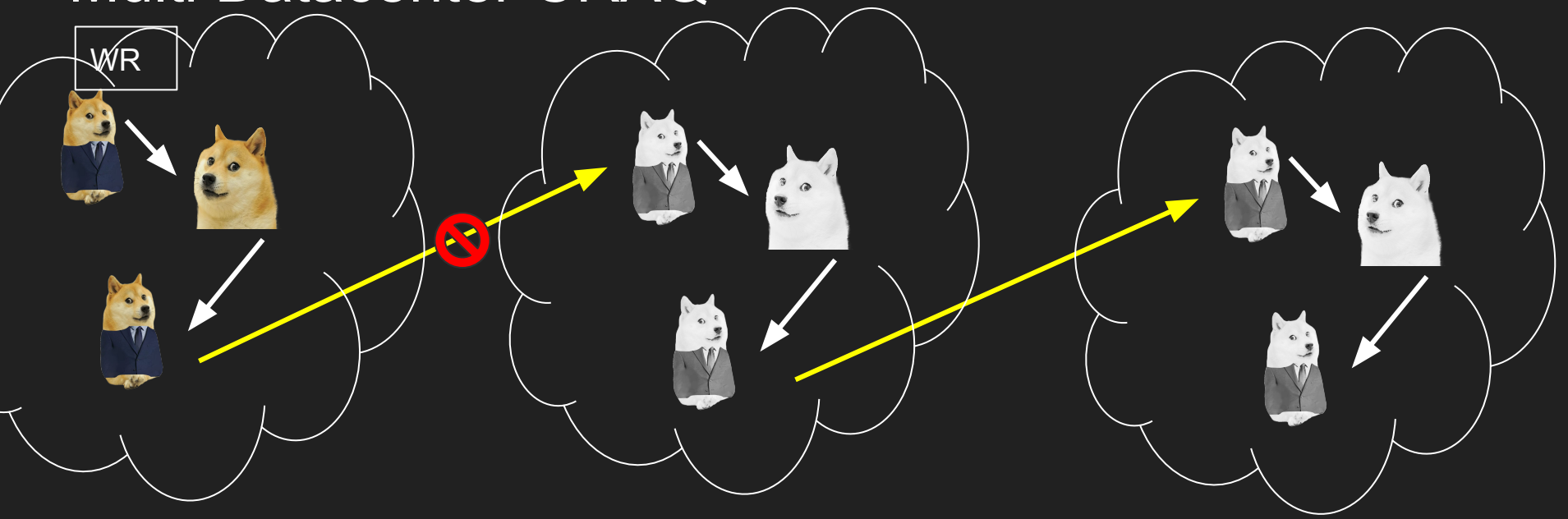
- Writes must still go to global head
- Dirty read queries local tail instead
  - If the local tail is dirty, then it has to contact the global tail
- Maybe some objects are only relevant to some datacenters

Multi Datacenter CRAQ



- If no current writes to D1, then D1 turns off
- Local head of D2 is now global head

Multi Datacenter CRAQ



- If there was a write to the D1, then actually D2, D3 turn off
- Local tail of D1 is now global tail