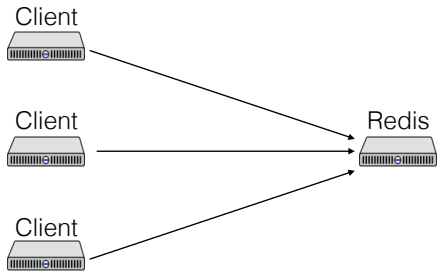
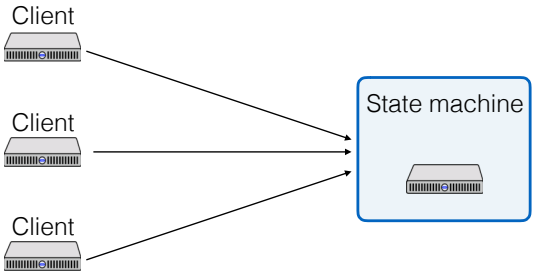
Memory Consistency Models

Tom Anderson (+ Doug Woos)

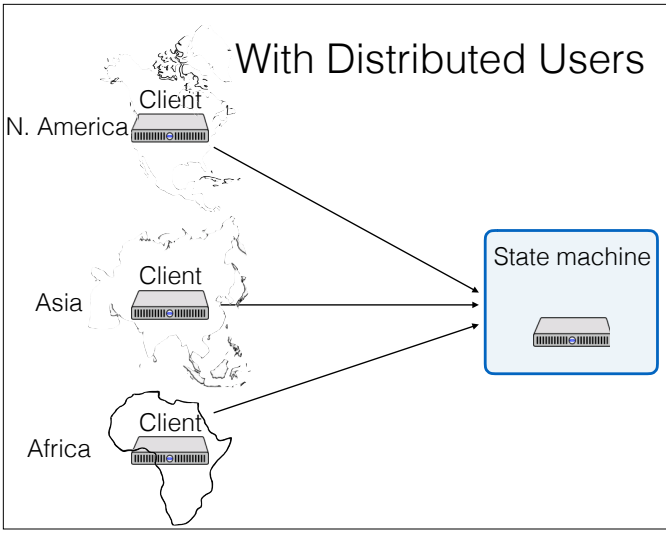
A Central Store



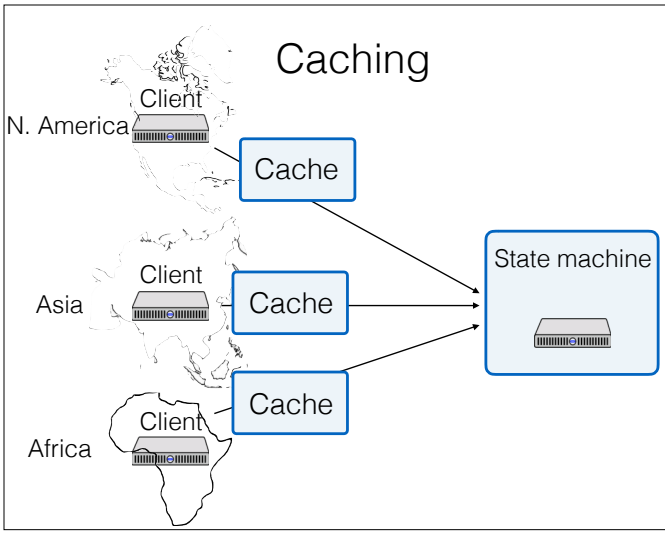
A Replicated Central Store



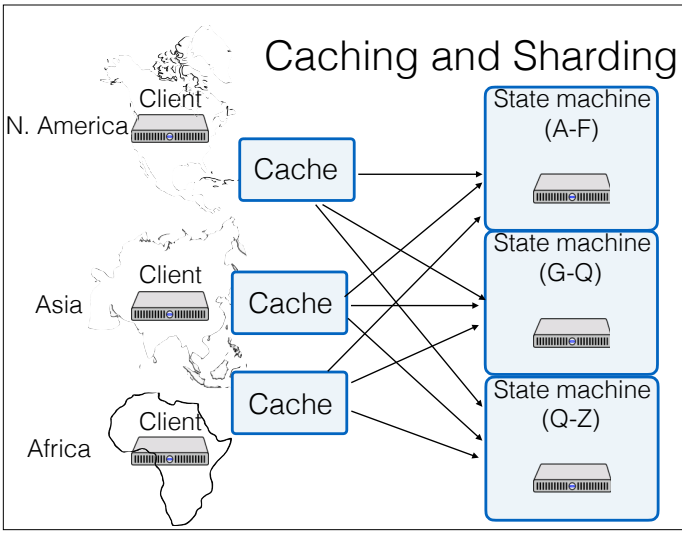
With Distributed Users



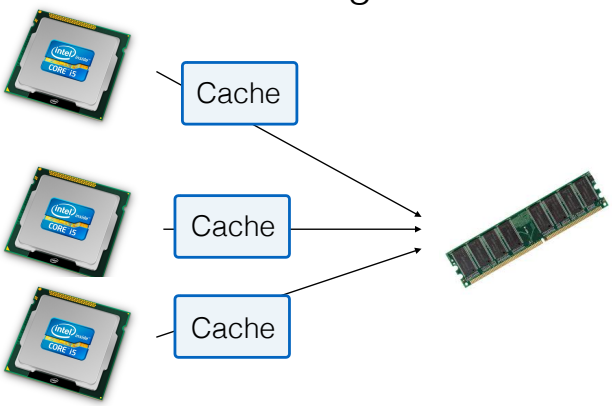
Caching



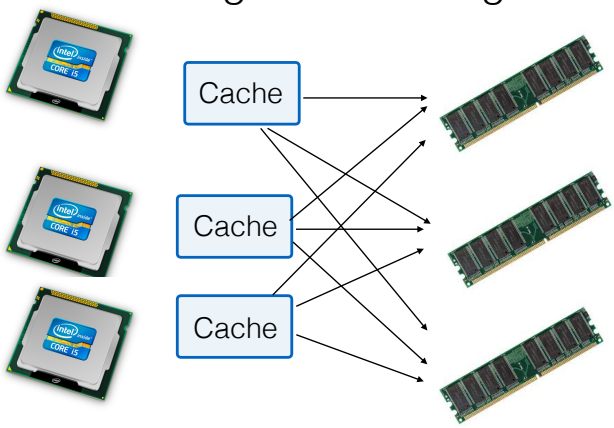
Caching and Sharding



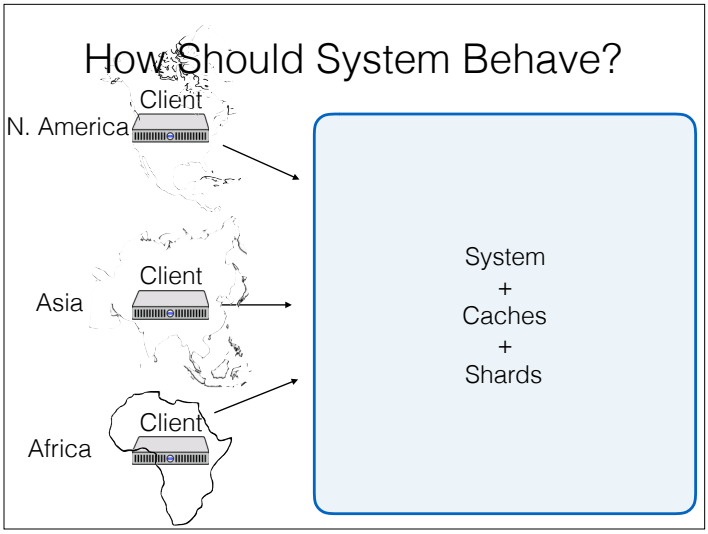
Caching



Caching and Sharding



How Should System Behave?



Intuition

Want system to act like a single central store

- As if a single copy

Caches and shards are an implementation decision, which shouldn't affect client-visible behavior

Terminology

Anomaly: some sequence of operations (reads and writes) that wouldn't be allowed

- By the single store abstraction

Classes of memory consistency model:

- Strong consistency: no anomalies allowed
- Weak consistency: could have anomalies
- Eventual consistency: anomalies are "temporary"

Why different models?

Performance

- Consistency requires synchronization/coordination
- Slower to make sure you always return right answer
- Tradeoff: performance vs. how "wrong" or out of date

Availability

- What if client is offline, or network is not working?
- Eventual consistency may be only option

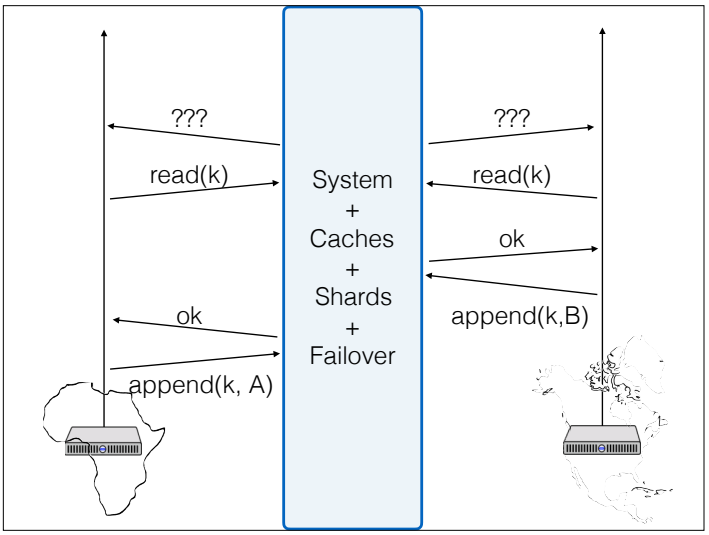
Programmability

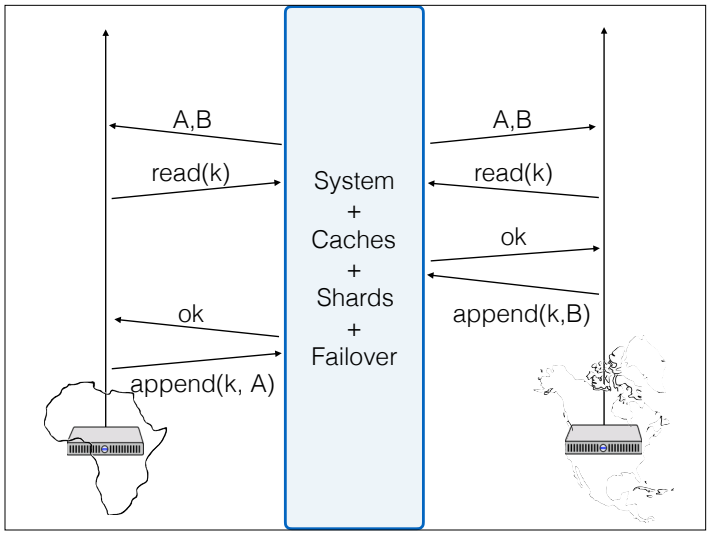
- Weaker models are harder to reason against

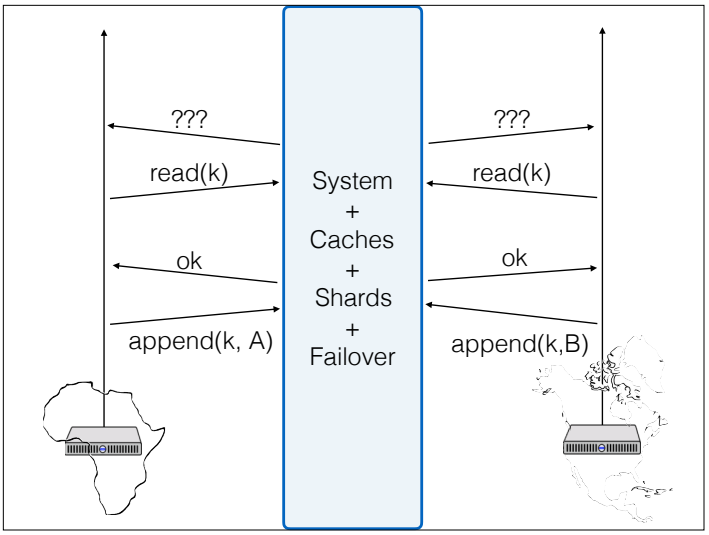
Linearizability

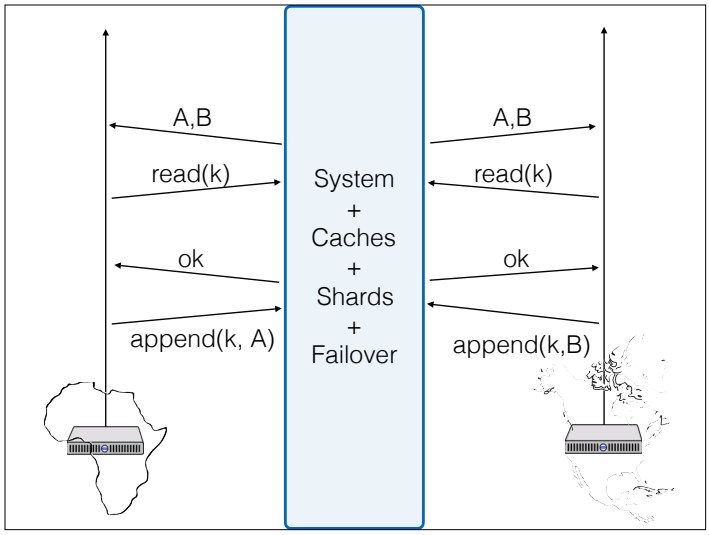
Linearizability or Strict Consistency

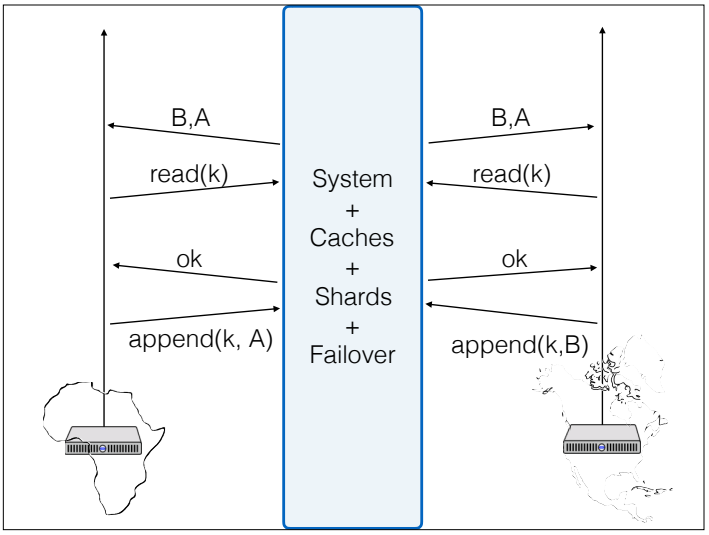
- Equivalent to idealized single store
- Reads always reflect latest write
- Concurrent operations can be executed in any order
- All reads reflect same order of operations

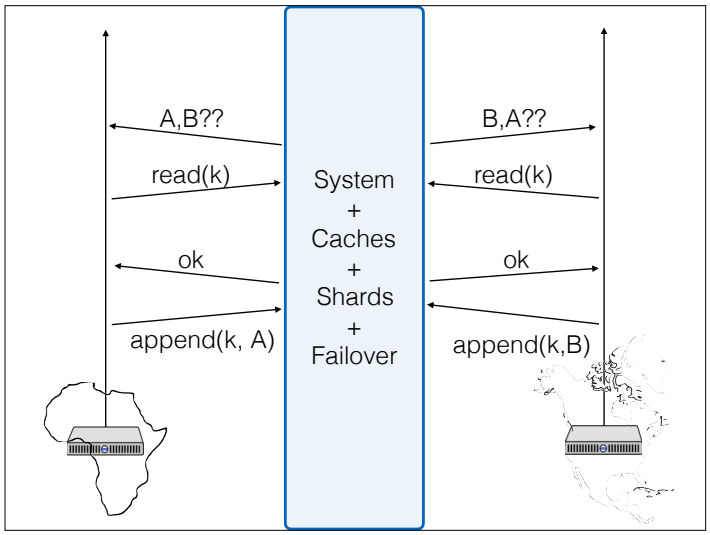


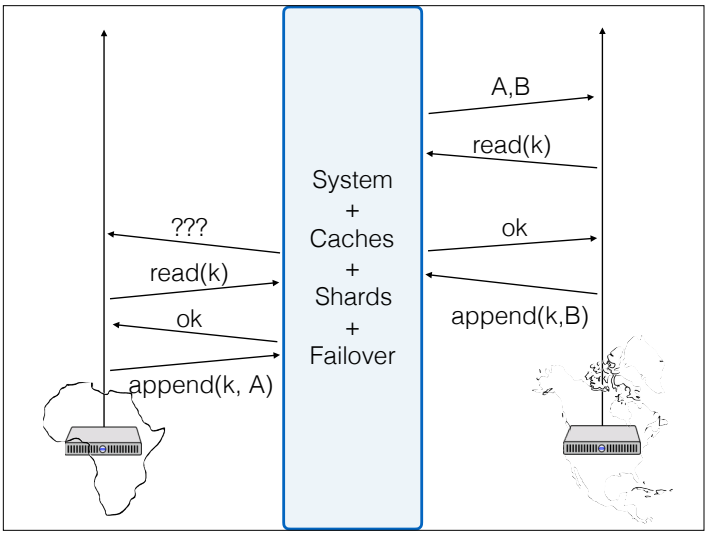


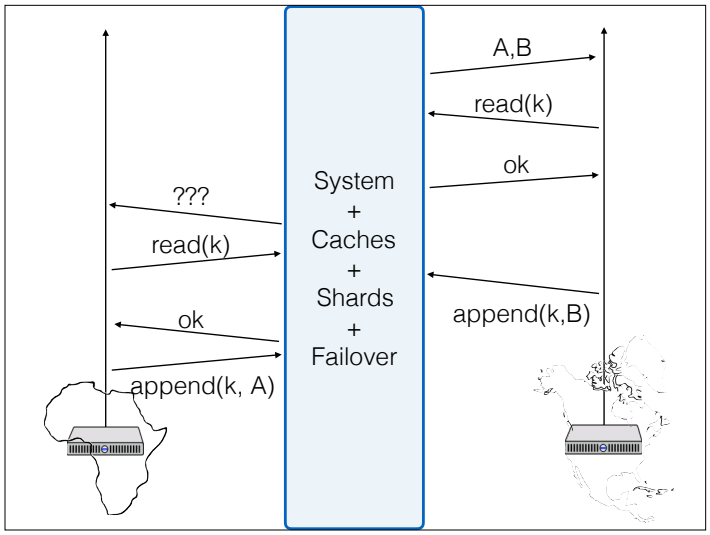


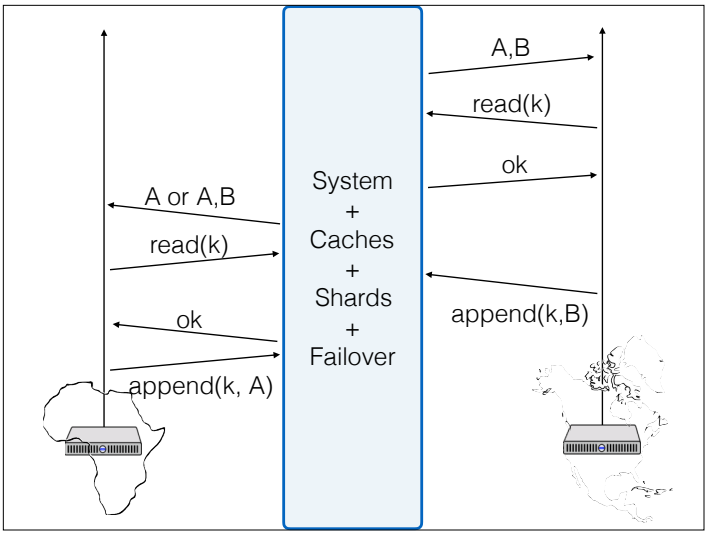


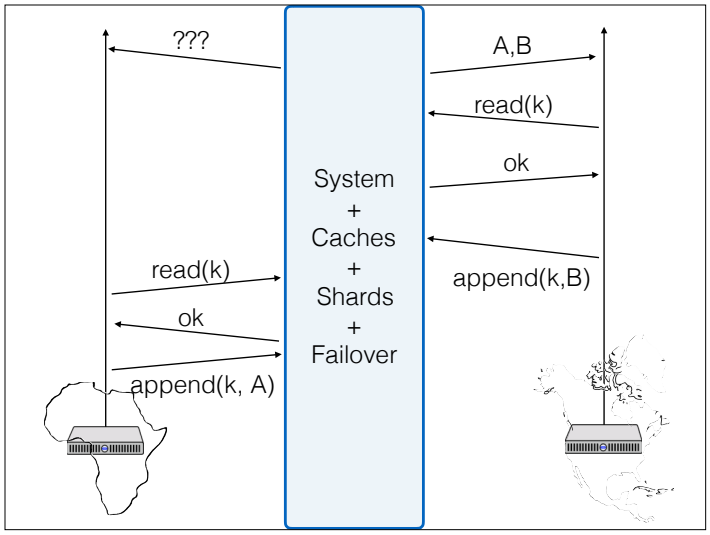


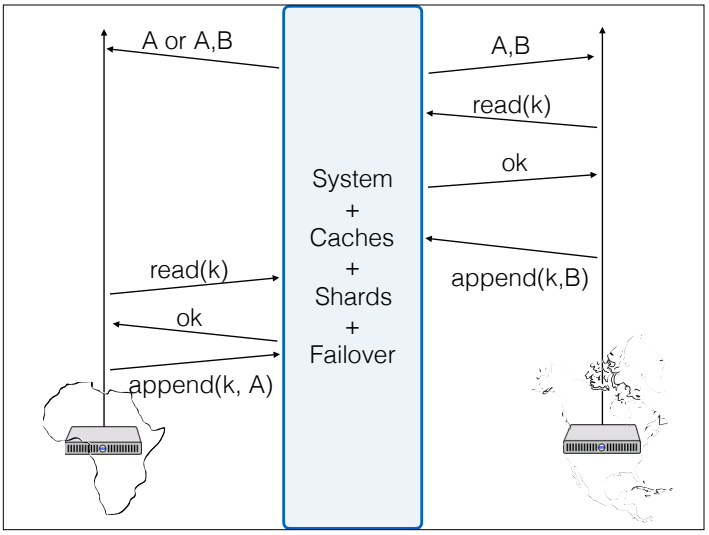

A,B??
read(k)
ok
append(k, A)
System
+
Caches
+
Shards
+
Failover
B,A??
read(k)
ok
append(k,B)


???
read(k)
ok
append(k, A)
System
+
Caches
+
Shards
+
Failover
A,B
read(k)
ok
append(k,B)


???
read(k)
ok
append(k, A)
System
+
Caches
+
Shards
+
Failover
A,B
read(k)
ok
append(k,B)


A or A,B
read(k)
ok
append(k, A)
System
+
Caches
+
Shards
+
Failover
A,B
read(k)
ok
append(k,B)


???
read(k)
ok
append(k, A)
System
+
Caches
+
Shards
+
Failover
A,B
read(k)
ok
append(k,B)


A or A,B
read(k)
ok
append(k, A)
System
+
Caches
+
Shards
+
Failover
A,B
read(k)
ok
append(k,B)

Serializability

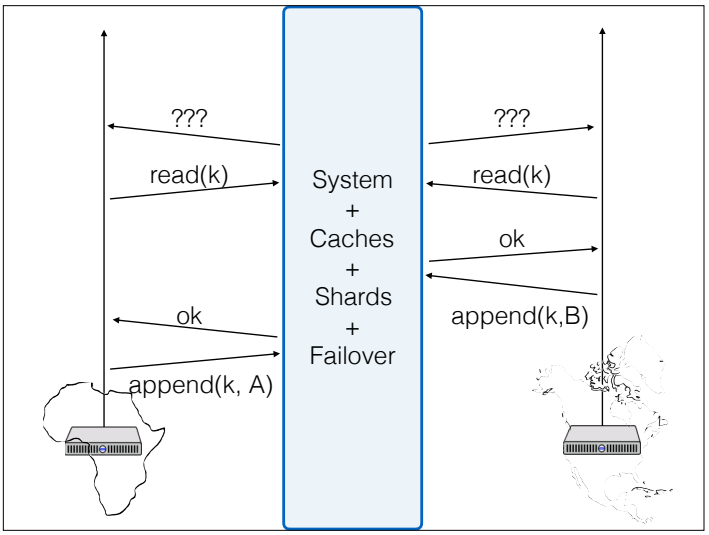
Serializability or Sequential Consistency

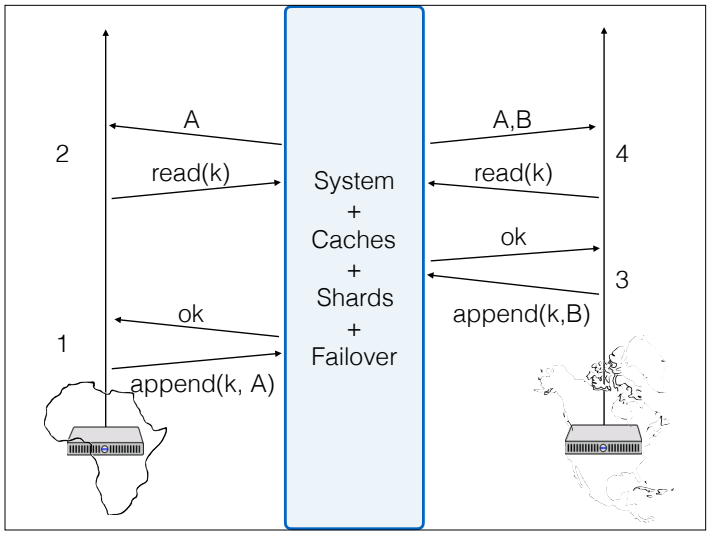
- Execution equivalent to *some* serial interleaving at a single store
- Plus each node's operations done in order

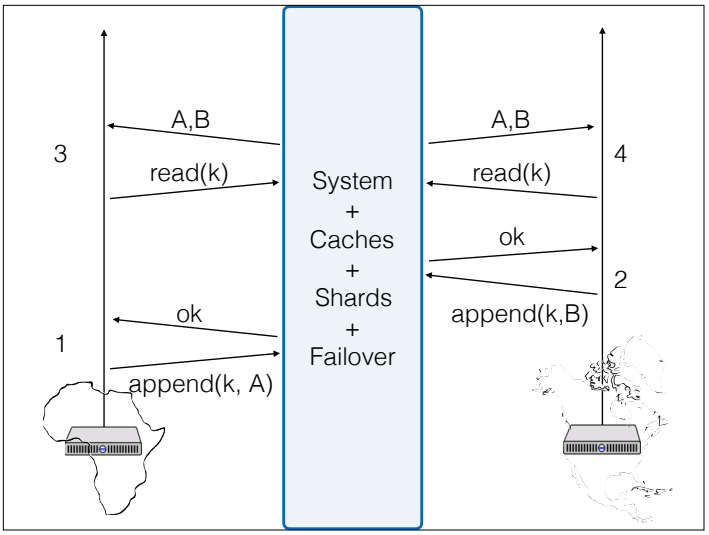
Linearizability, but *without* real time constraint

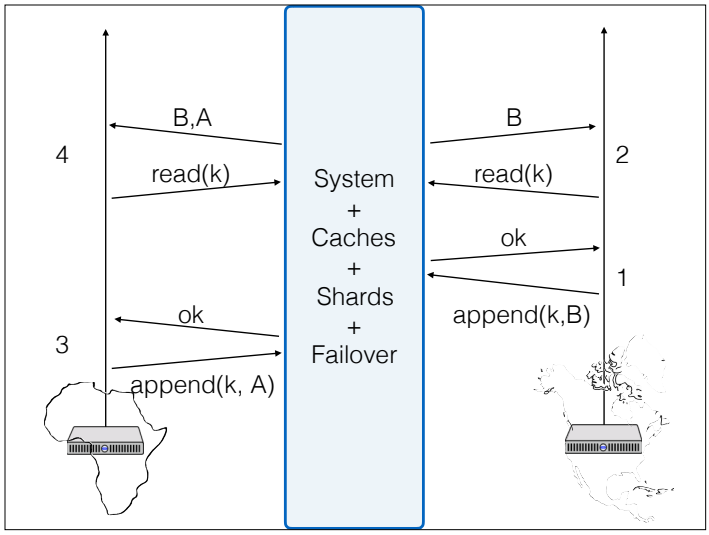
Allows optimizations:

- SQL query optimization
- Asynchronous writes to processor caches









Eventual Consistency

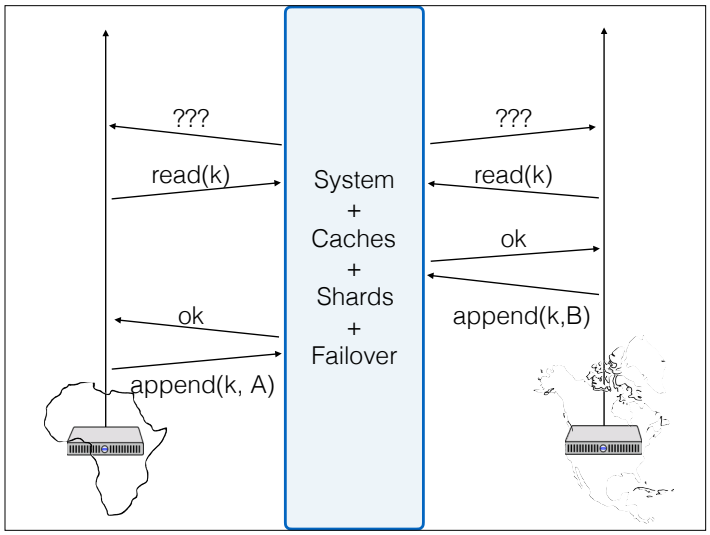
Read Your Writes

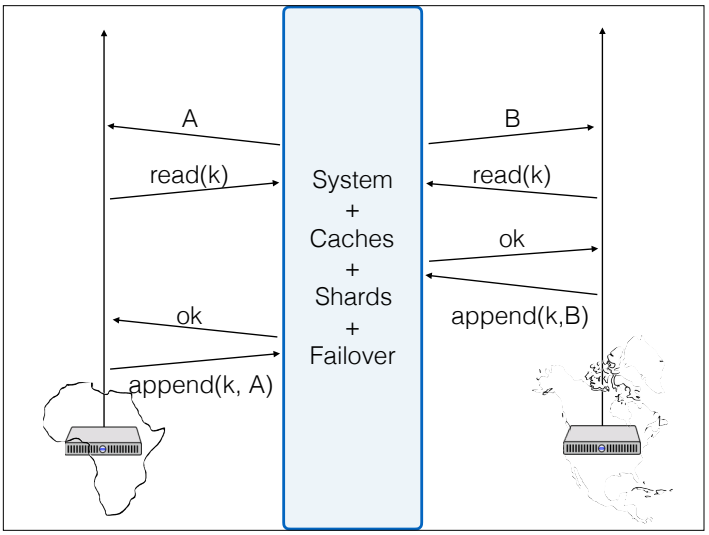
- Clients will always see their own writes

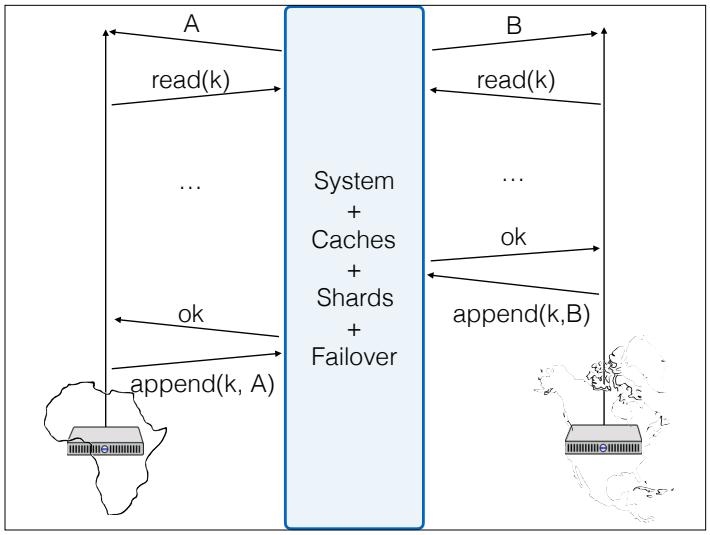
+ Eventual Consistency

- Clients will eventually see everyone's writes
- And eventually the order will be consistent
- Facebook model, approximately

Q: Can you test if a system is eventually consistent?







Snapshot Reads

Reads can be stale

Reads in same transaction need to be consistent

- Taken from the same global state

For example, compute sum of all bank accounts

- With concurrent transfers between accounts

Next couple of lectures

How to implement (various types of) consistency

> "There are only two hard things in Computer Science: cache invalidation and naming things."
> — Phil Karlton

If we cache and shard data, how do we make sure a read reflects writes by other clients?

- While being fast and scalable and available