Consistency in Distributed Storage Systems

Mihir Nanavati
March 4th, 2016

Today

- Overview of distributed storage systems
- CAP Theorem

About Me

- Virtualization/Containers, CPU microarchitectures/Caches, Network Stacks
- Last year and a half: high speed storage systems (Last 24 hours: Powerpoint)
- Don’t believe everything I say – built (at most) one distributed system more than you!

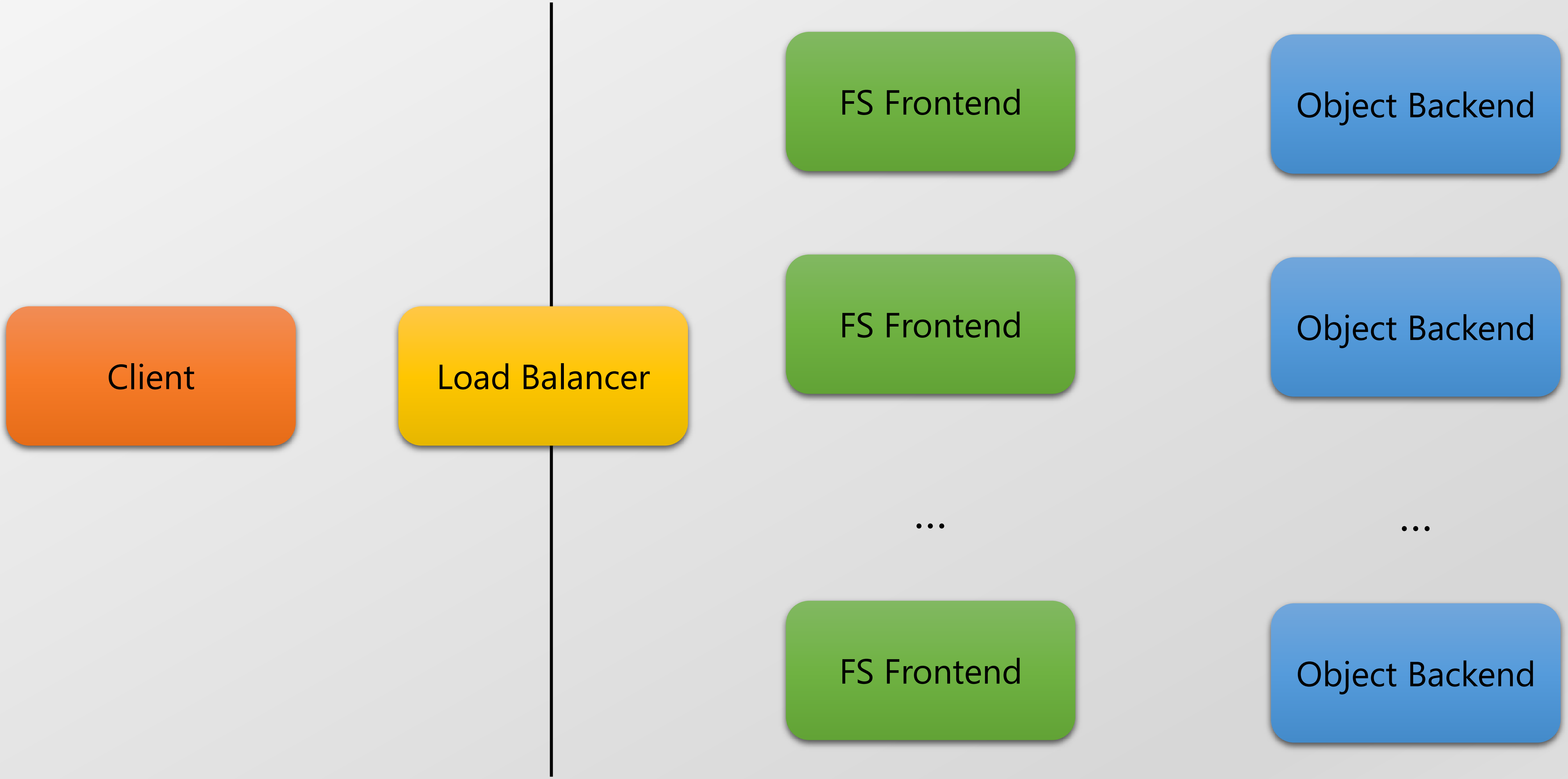
On Systems

- “Point in time” designs
- Balance tradeoffs
- What is old is new again?

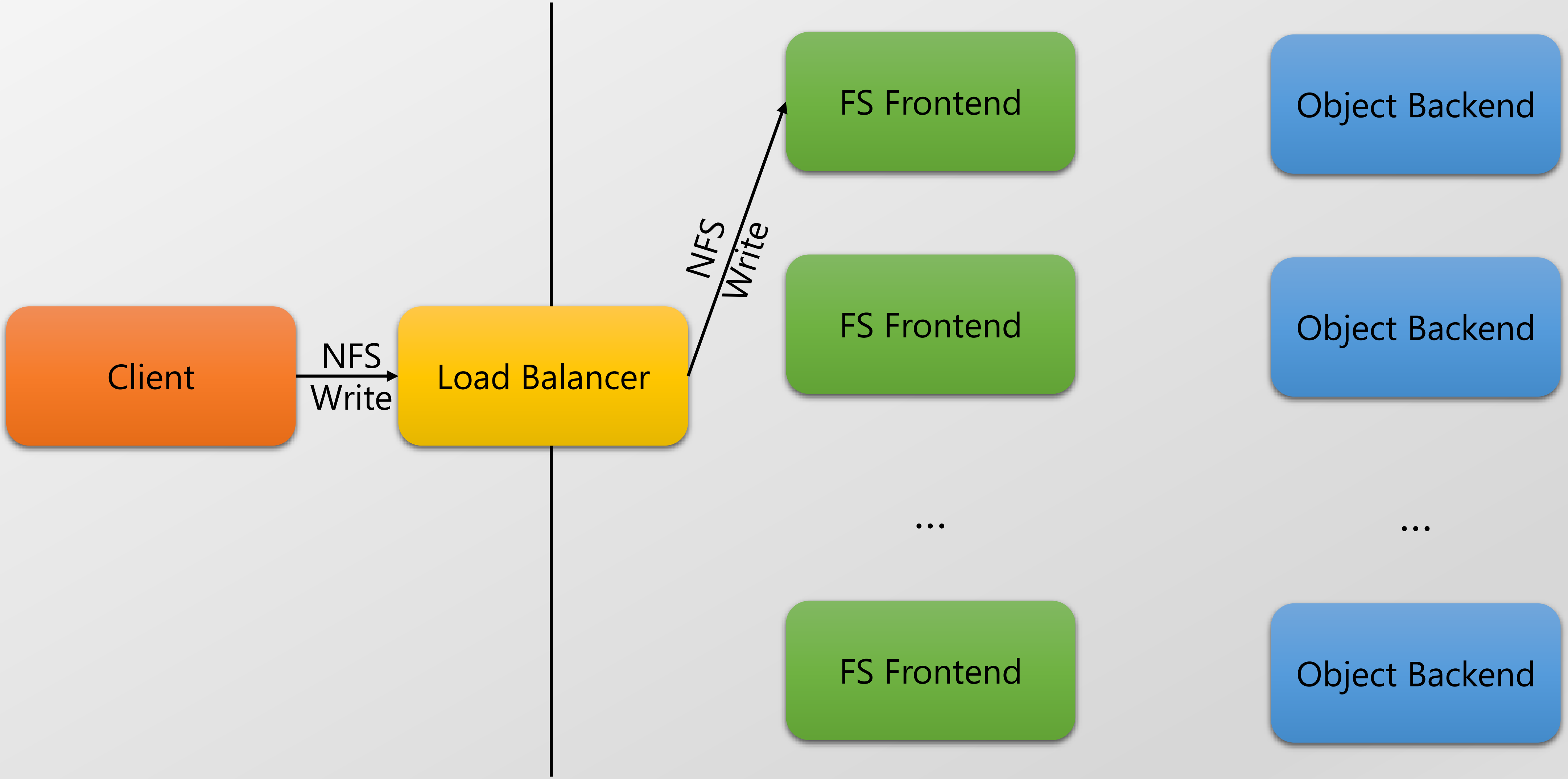
Why Distributed Storage Systems?

- Performance
- Capacity
- Reliability

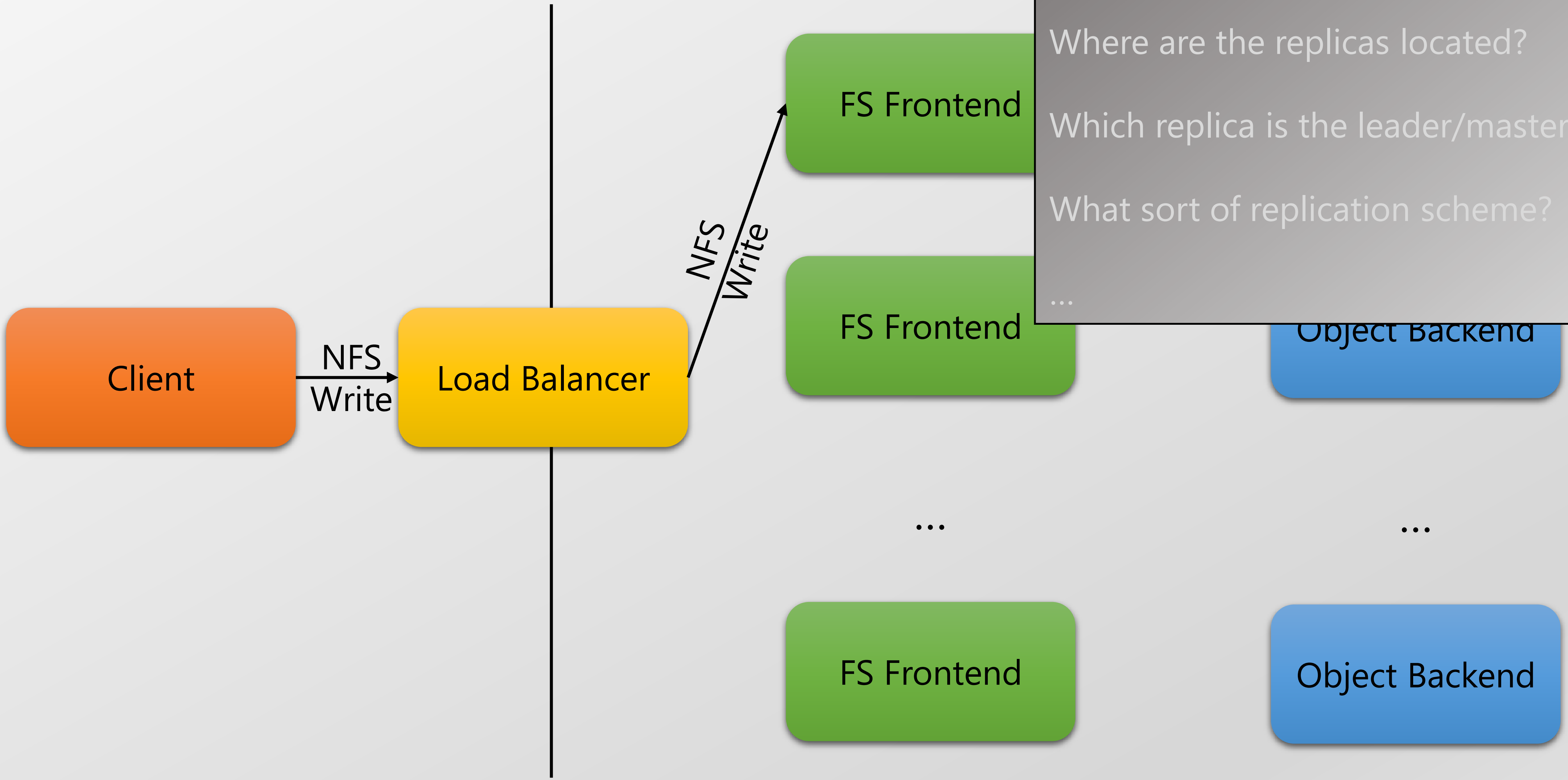
Architecture



Architecture



Architecture



What does this file map to?

How many replicas does it have?

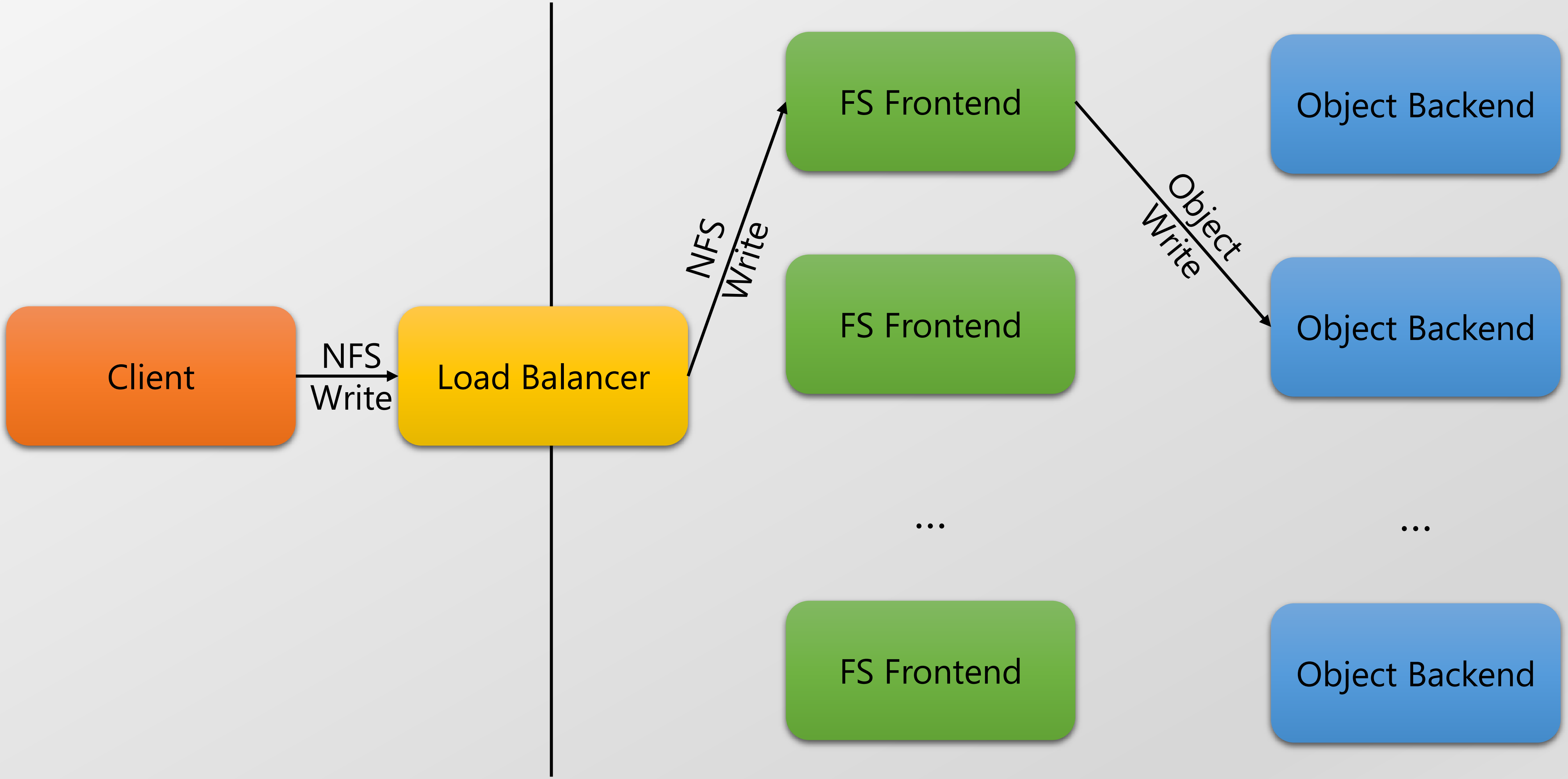
Where are the replicas located?

Which replica is the leader/master?

What sort of replication scheme?

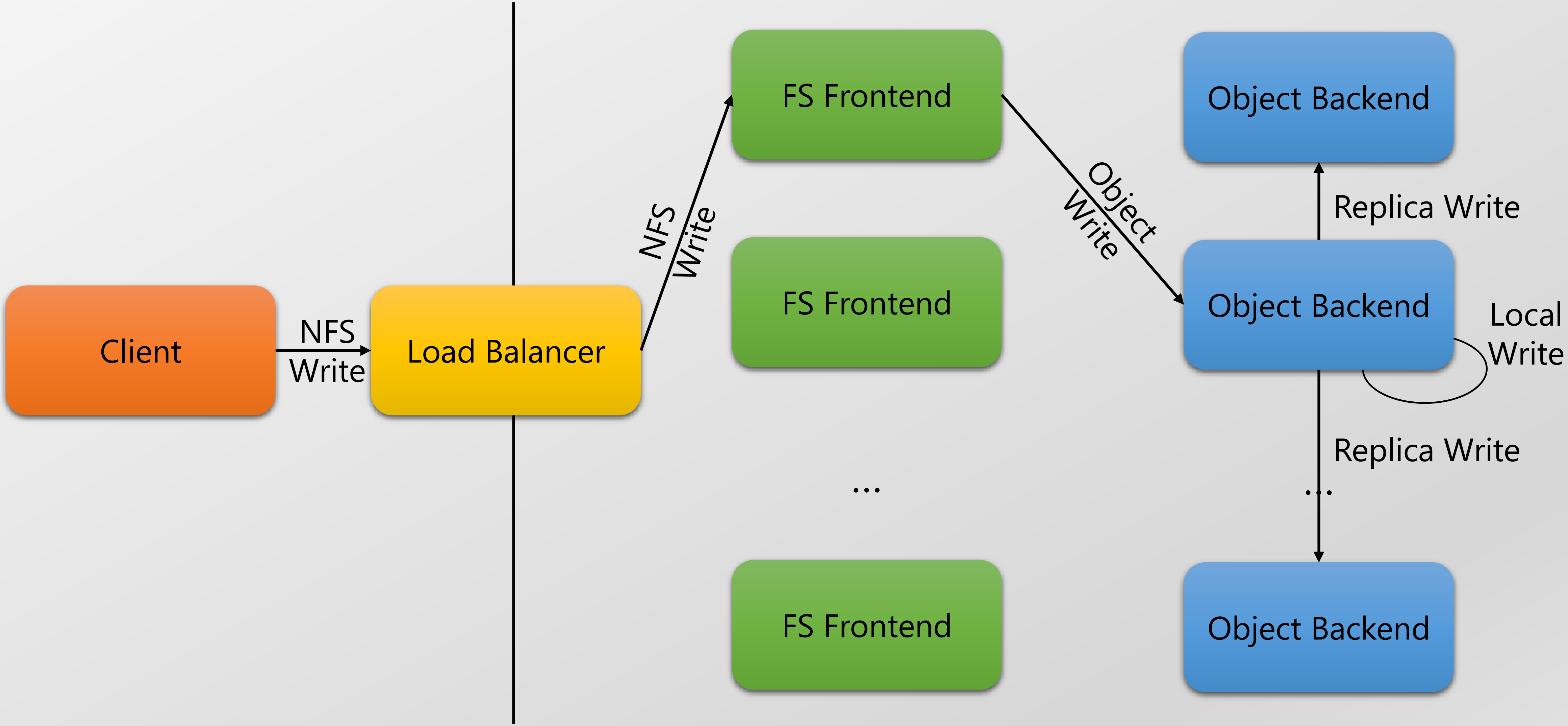
…

Architecture

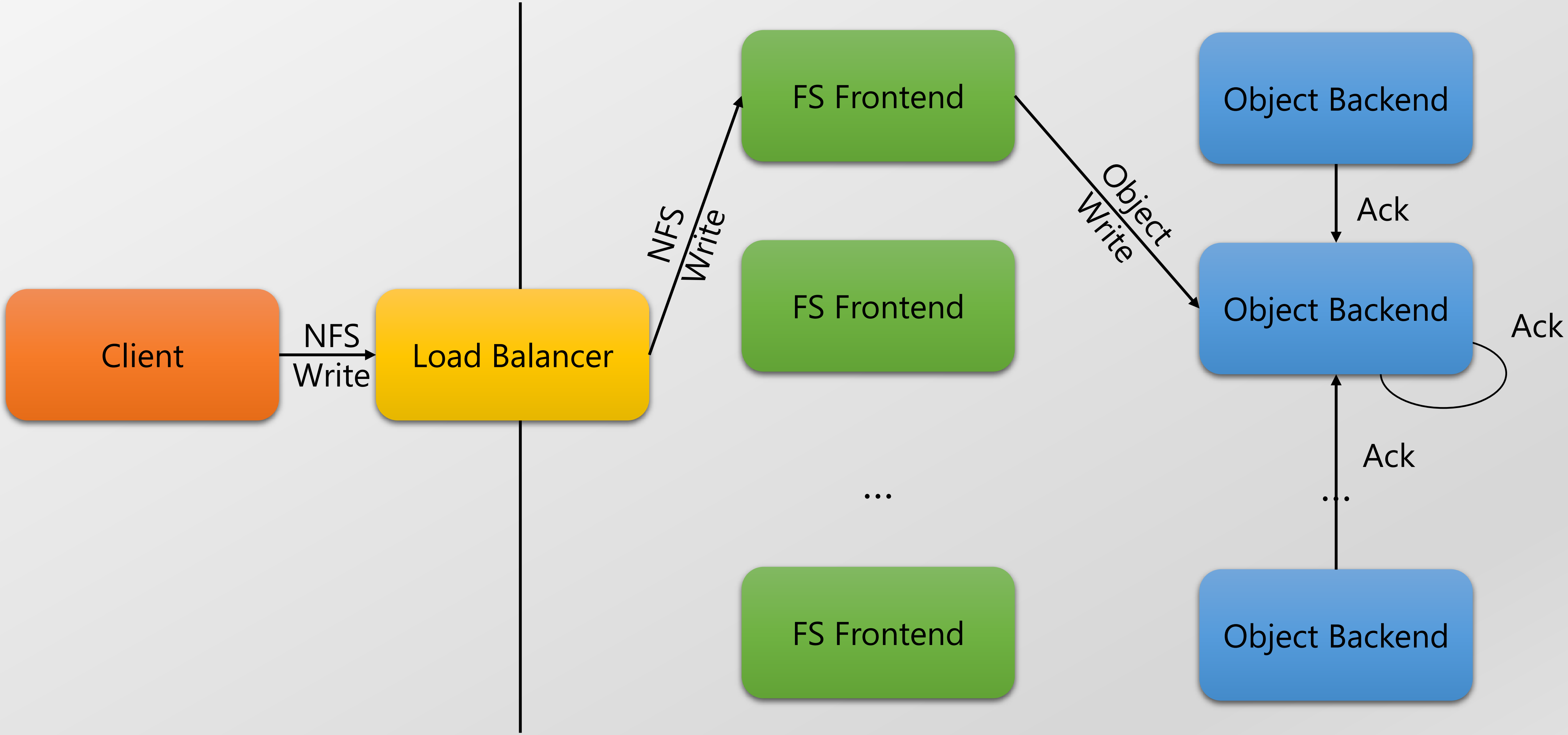


Architecture

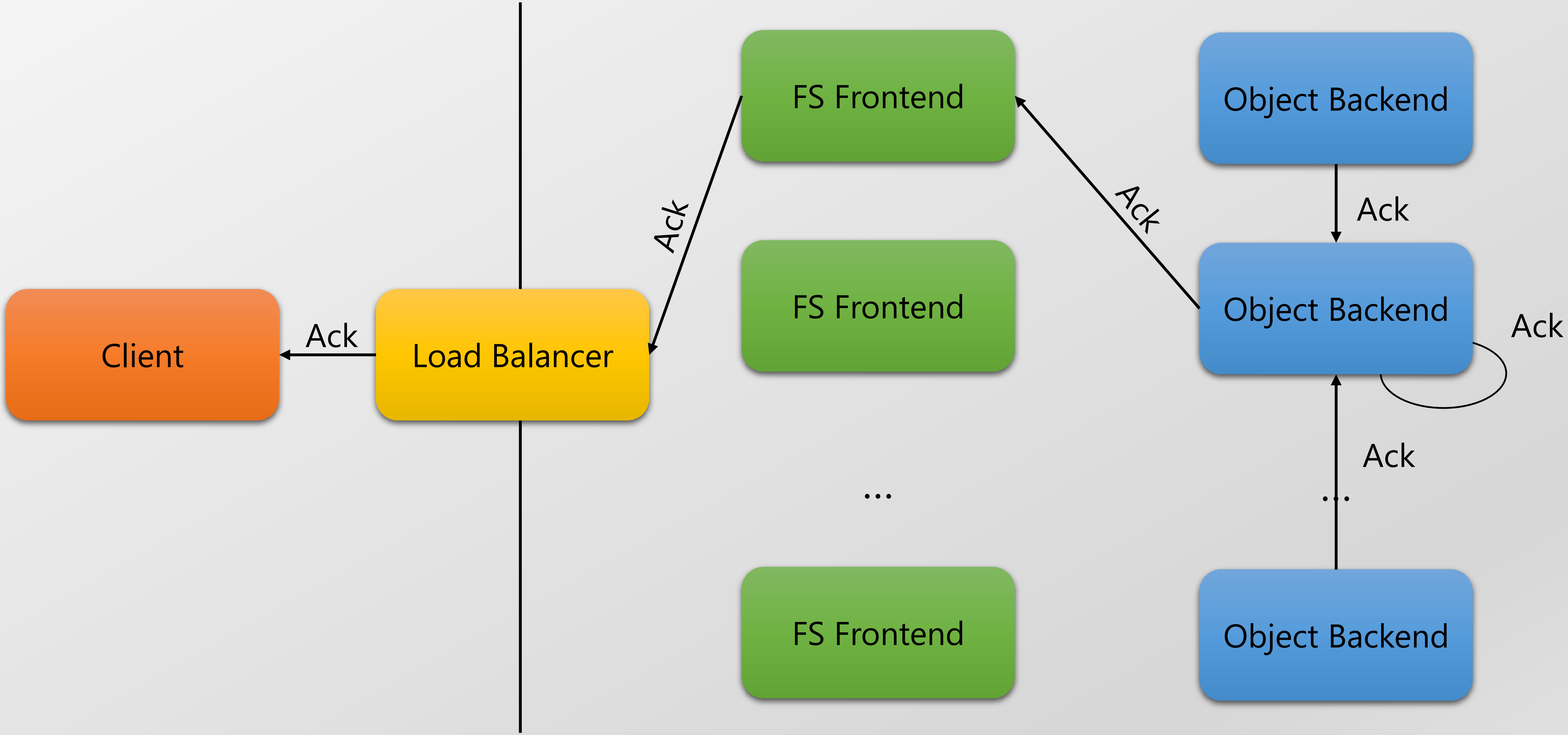
Subtleties with in-parallel writes!



Architecture



Architecture



Metadata

- What does this file map to?
- How many replicas does it have?
- Where are the replicas located?
- Which replica is the leader/master?
- What sort of replication scheme?

Location of Replicas

- Directory Service [Petal, GFS]
- Consistent Hashing [Chord, Redis]
- Clients vs Proxy?

Directory Services vs Hashing

- Fine grained placement
- Locality
- Lookup on reads/writes
- Update on data moves

- Statistical load balancing
- Lookups are “free”
- Membership list of nodes
- Update on node churn

Directory Services vs Hashing

- Fine grained placement
- Locality
- Lookup on reads/writes
- **Update on data moves**

- Statistical load balancing
- Lookups are “free”
- Membership list of nodes
- **Update on churn**

Both systems require metadata

Data and Metadata

- Can’t avoid having metadata
- Metadata is also data!
- Different consistency and availability requirements!

Consistency and Availability

Consistency:

- Is there a single “true” value for every object in the system?
- Does a read always return the result of the most recent write, i.e., is stale data ever visible?

Availability: Is data accessible in the face of failures?

CAP Theorem [Brewer]

Pick any two of:

- Sequential Consistency
- Availability
- Partition Tolerance

C in CAP vs C in ACID

- ACID vs BASE

- Different definitions of consistency!
  - CAP’s C(onsistency) = ACID’s A(tomicity) = Visibility to all future operations
  - ACID’s C(onsistency) = Does data follow all schema constraints?

Sequential Consistency

“if client *a* executes operations {*a1*, *a2*, *a3*, ...} and
client *b* executes operations {*b1*, *b2*, *b3*, ...},
then you could create some serialized version
{*a1*, *b1*, *b2*, *a2*, ...} (or whatever)
executed by the clients“

Sequential Consistency

For a single piece of data [Lamport]:

- Appears like a single copy to an outside observer
- Strict ordering of operations from the same client
- Some arbitrary (single) ordering of operating across clients

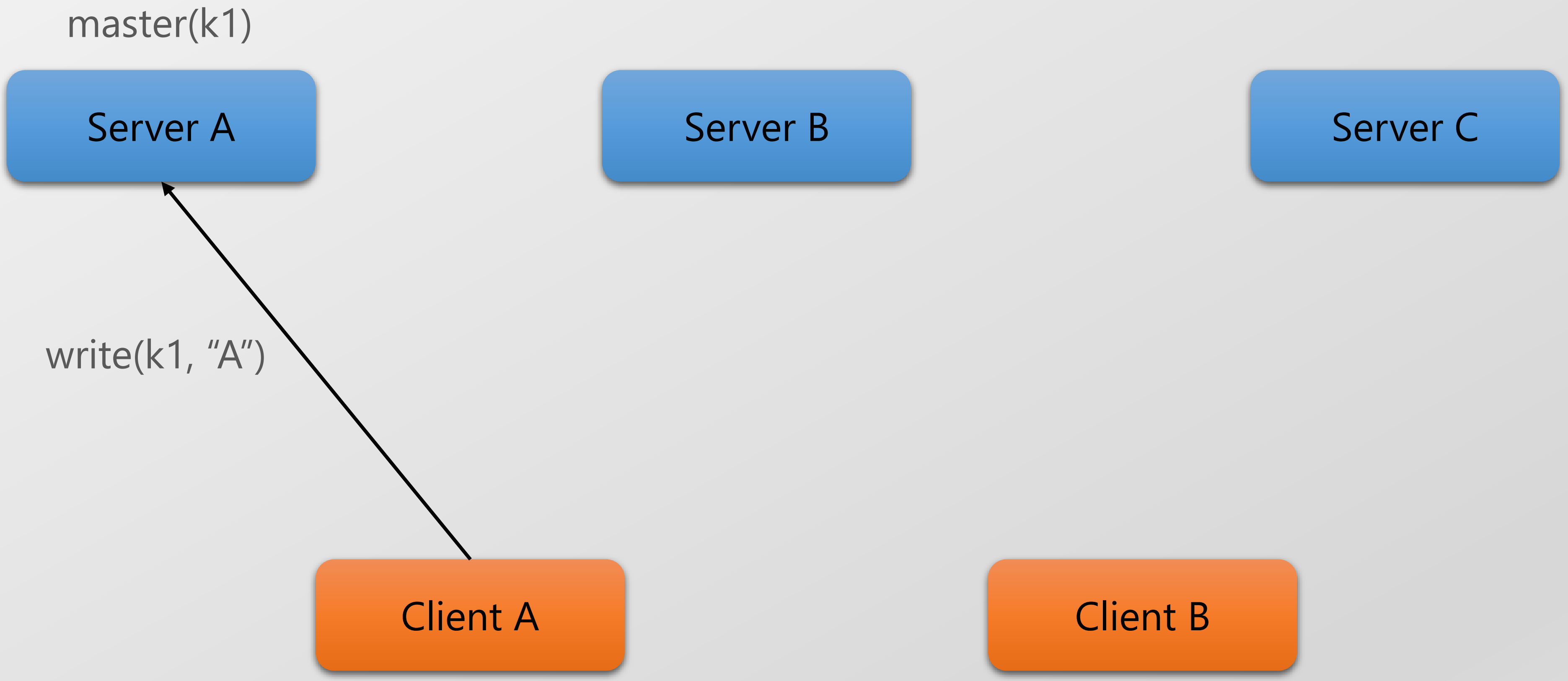
Sometimes referred to as “linearizable” [Herlihy and Wing]

Defeating CAP

Setup

- 3 servers, 2 clients
- 3-way in-memory replication
- Each piece of data has a single master
- Clients send request to master if available, else any other node

Defeating CAP



master(k1)

Server A

Server B

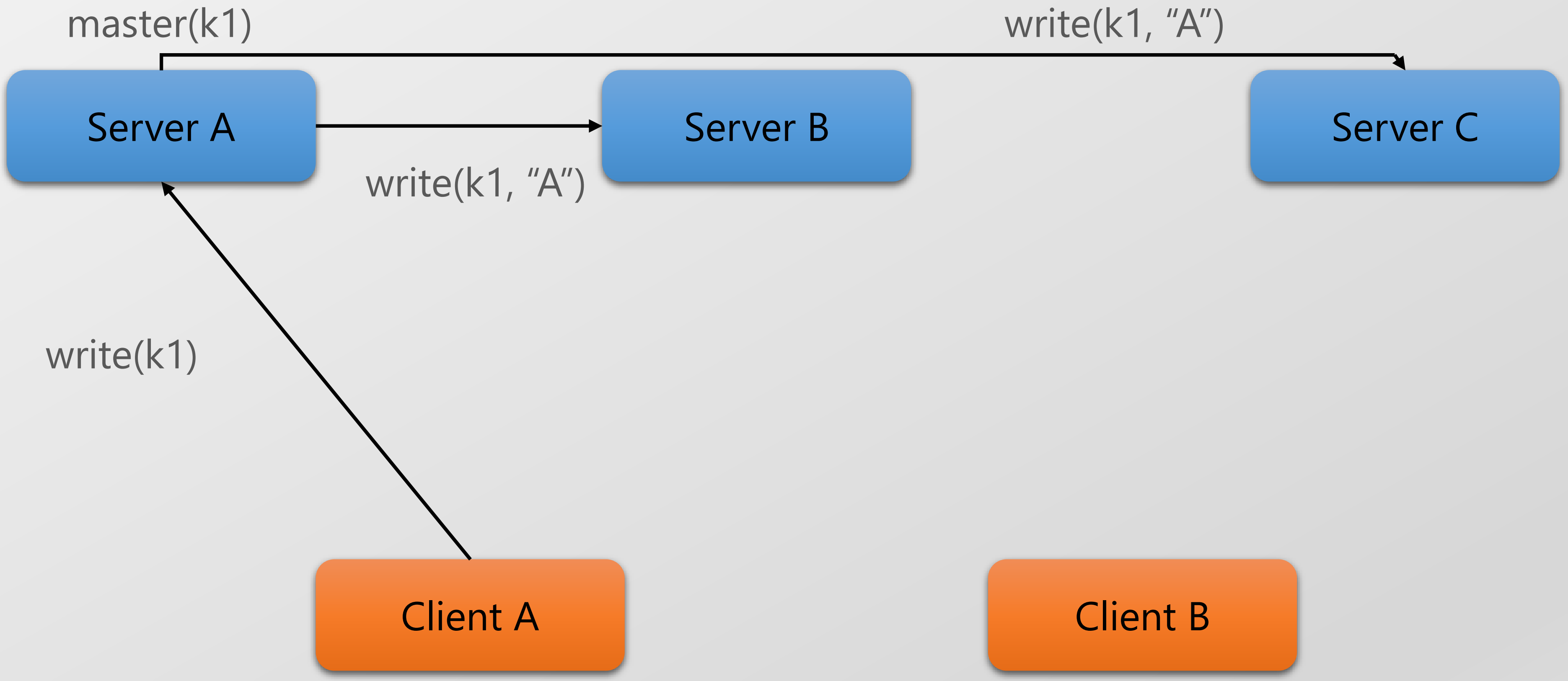
Server C

write(k1, "A")

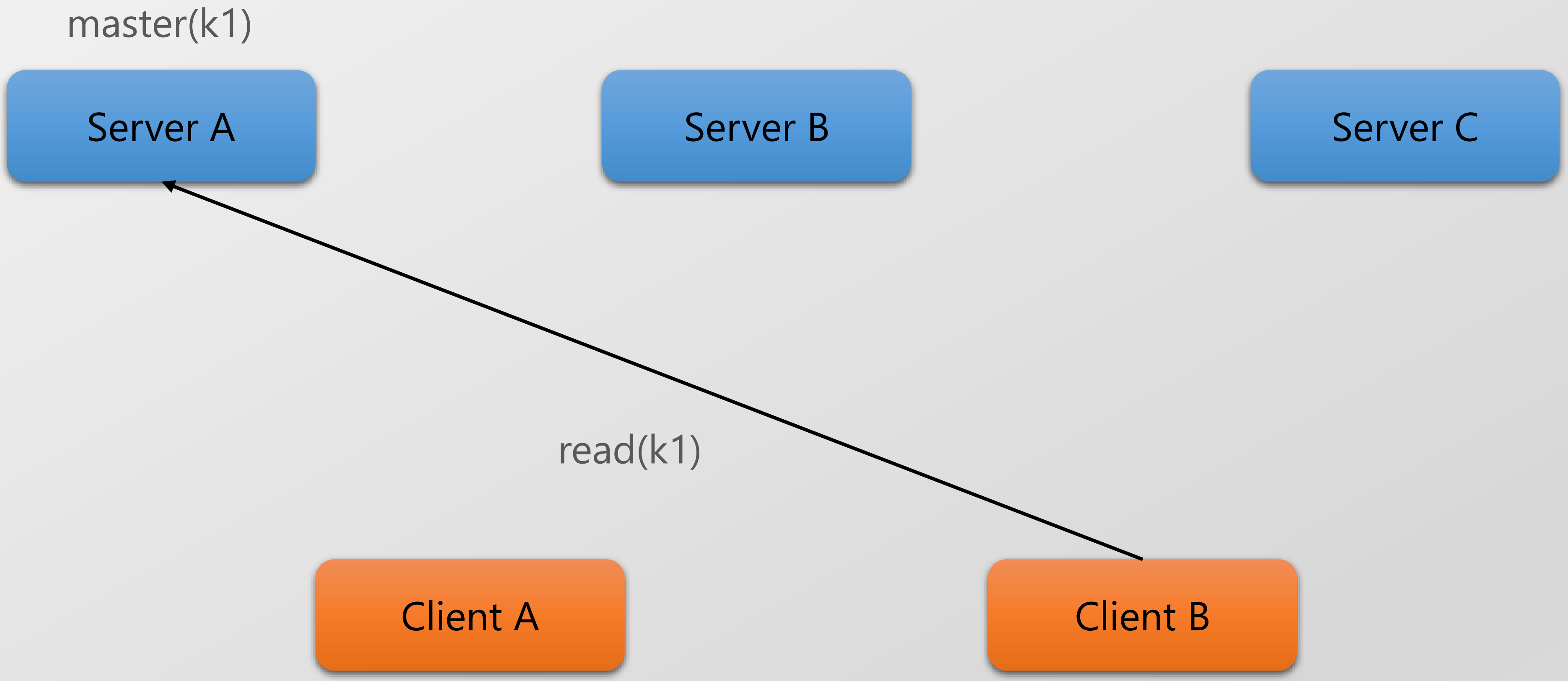
Client A

Client B

Defeating CAP



Defeating CAP



master(k1)

Server A

Server B

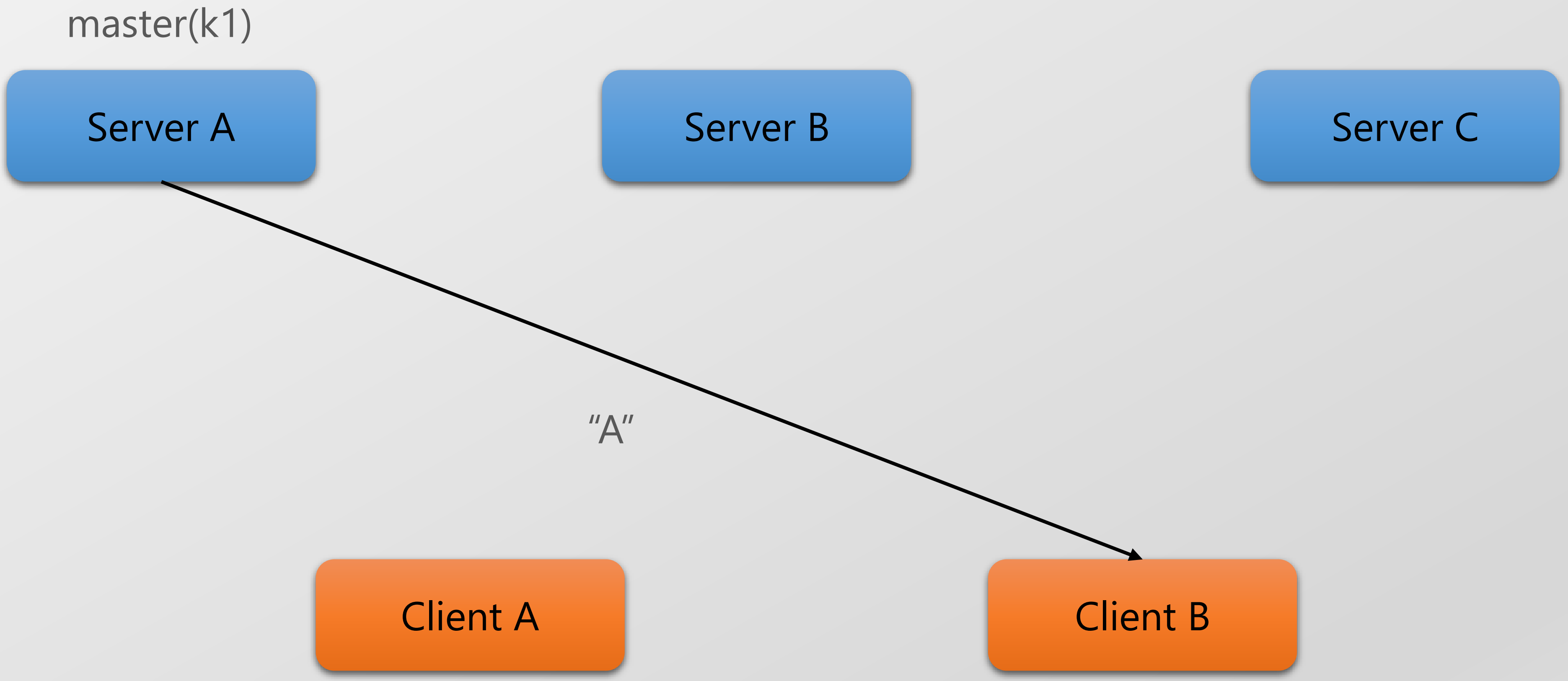
Server C

read(k1)

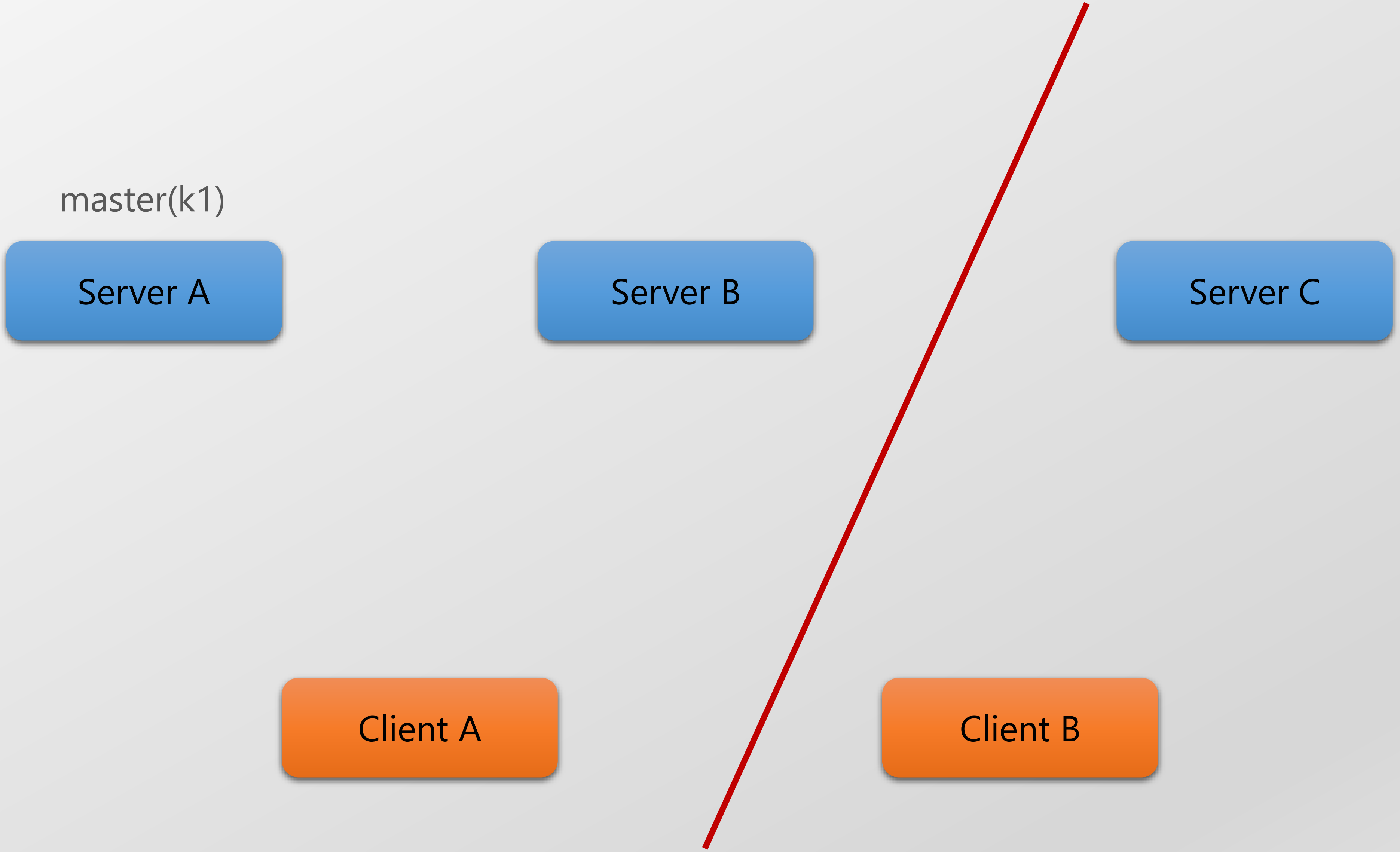
Client A

Client B

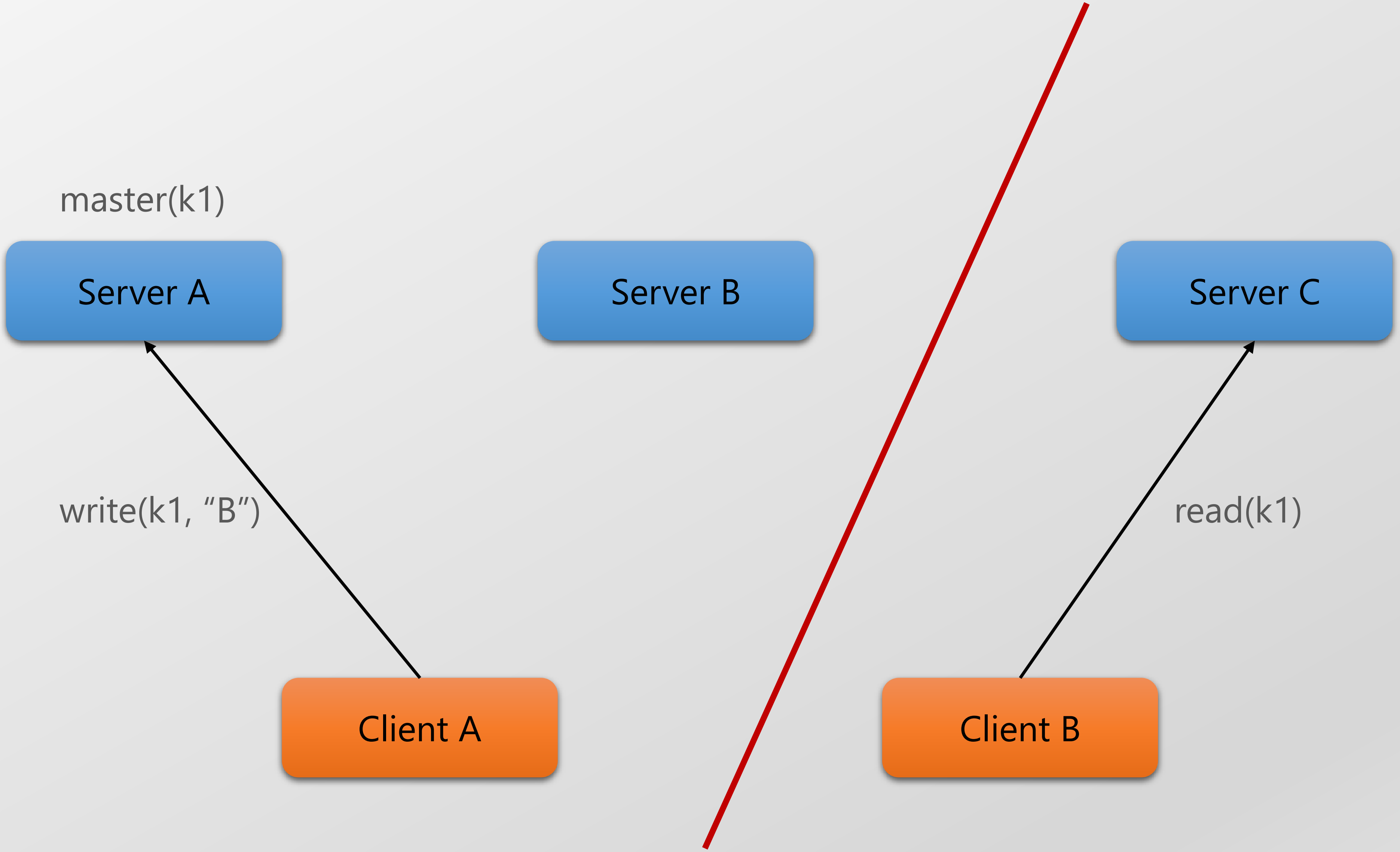
Defeating CAP



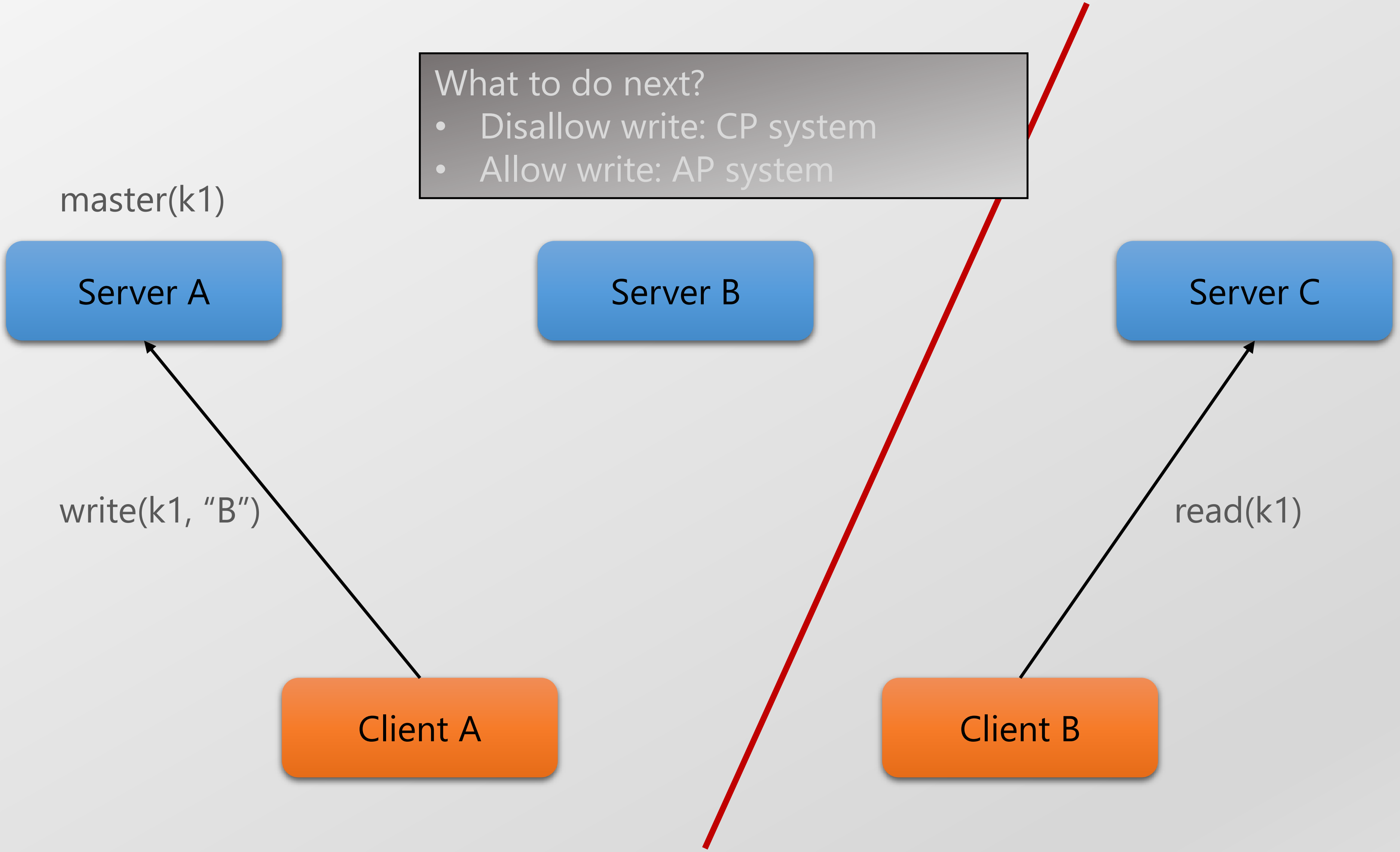
Defeating CAP



Defeating CAP



Defeating CAP



What to do next?

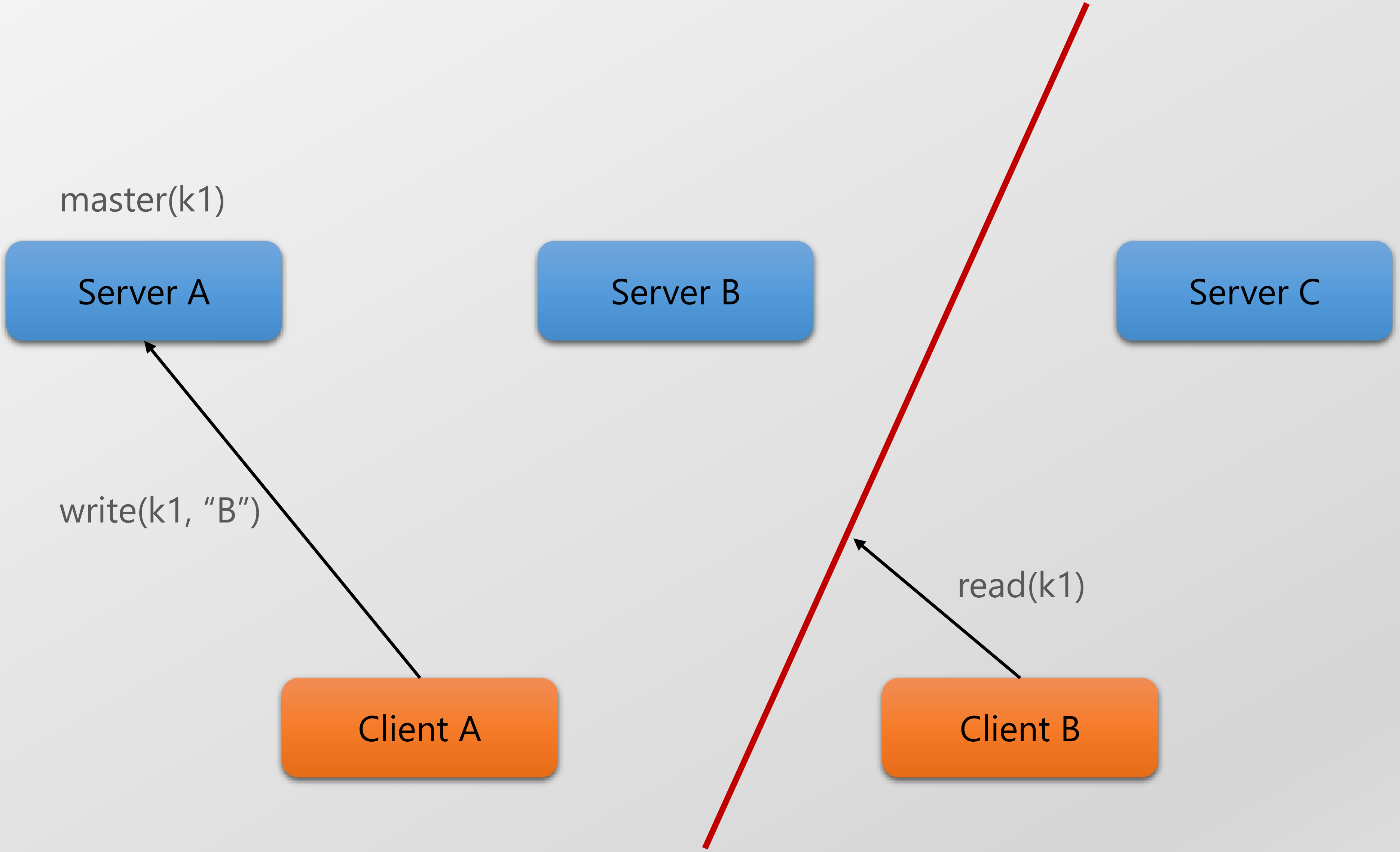
- Disallow write: CP system
- Allow write: AP system

Defeating CAP

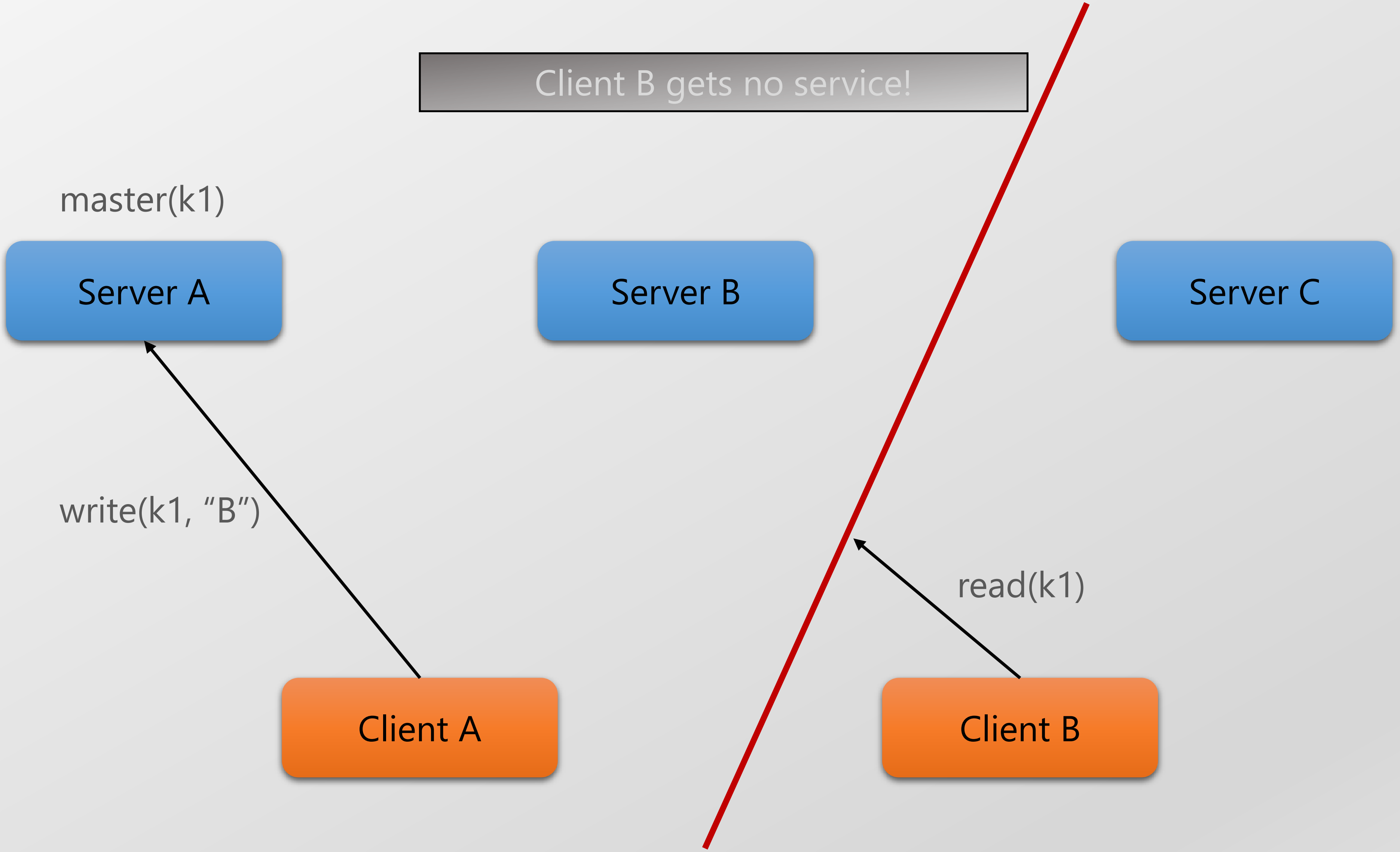
Setup

- 3 servers, 2 clients
- 3-way in-memory replication
- Each piece of data has a single master
- Clients only send request to masters

Defeating CAP



Defeating CAP



Client B gets no service!

Defeating CAP

- Defeating CAP is rather hard!

- Formal proof of impossibility [Gilbert and Lynch]

Strong vs Eventual Consistency

- Extreme ends of the spectrum
- Active research: Causal, Causal+, ….
- NoSQL give up consistency: schema-less, no multi-key transactions, etc.

CAP: Pick any Two?

- Can’t drop P: CP vs AP systems
- Like a prix fixe menu!
- Not all systems are strictly CP or AP!

Dissenters [Michael Stonebreaker, VoltDB]

“In my experience, **network partitions** do not happen **often**. Specifically, they occur **less frequently** than the sum of bohrbugs [deterministic DB crashes], application errors, human errors and reprovisioning events. So it doesn’t much matter what you do when confronted with network partitions. Surviving them will not “move the needle” on availability because higher frequency events will cause global outages. Hence, you are **giving up something (consistency)** and getting nothing in return.”

Network Reliability [Gill, et al.]

- Intra-datacenter links: Median downtime of 10 minutes/year
- Inter-datacenter (WAN) links: 24-72 minutes per year
- Is a node failure a network partition?

CAP: What to Pick?

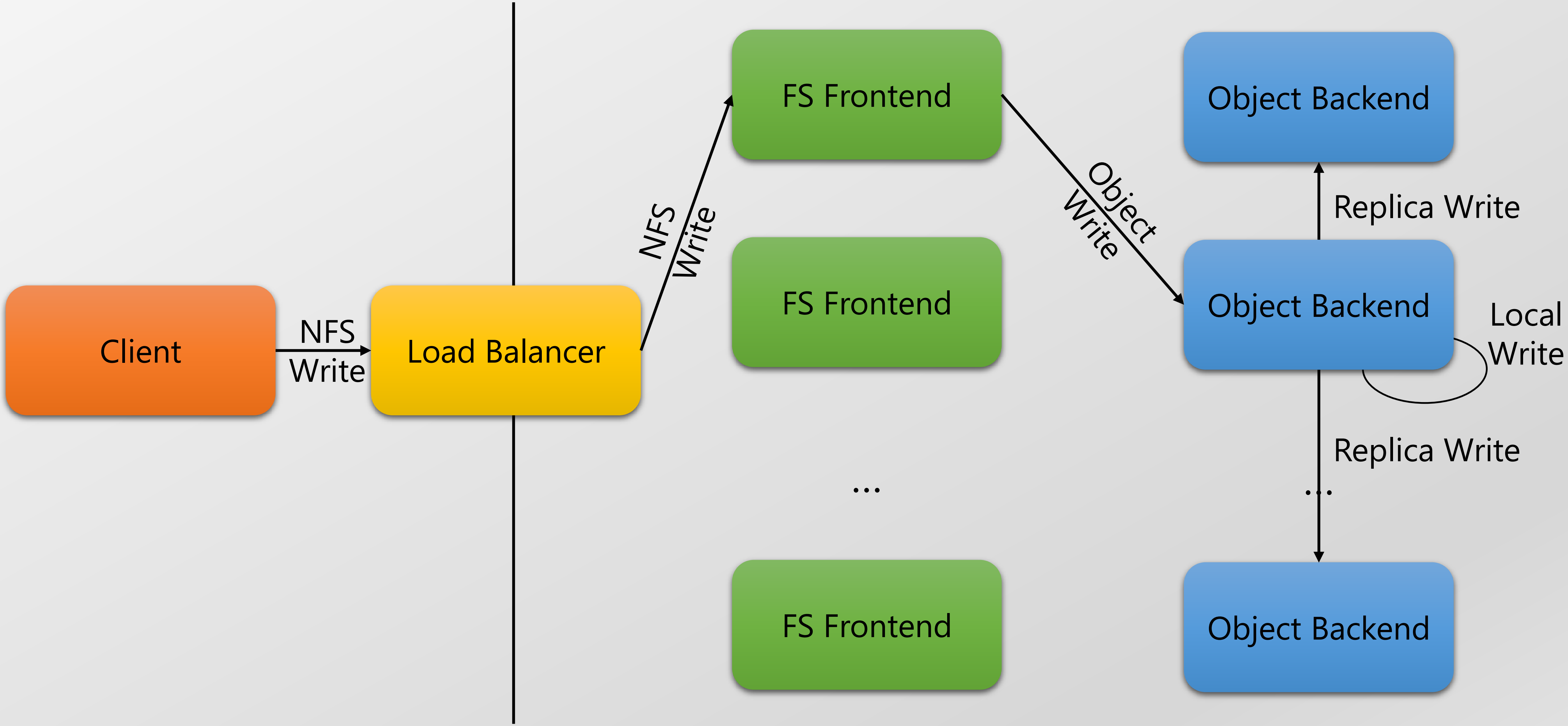
- No universally true guidelines
- Depends on application constraints
- Is the client within our control?

Consistency in Memory vs Non-volatile Storage

- Post crash semantics

Consistency in Durable Storage

Subtleties with in-parallel writes!



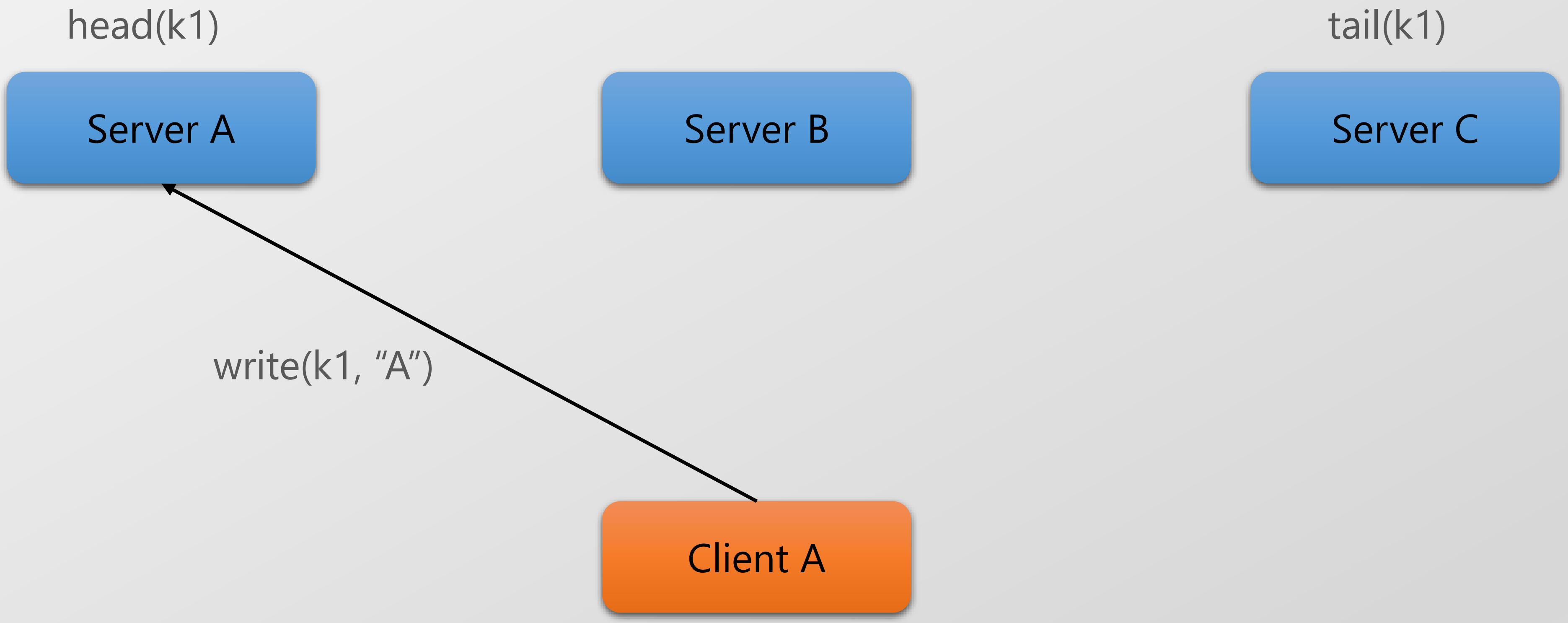
Consistency in Durable Storage

- In-place writes: Risk corruption
- Overwrite-allocate: Metadata updates
- Concurrency?

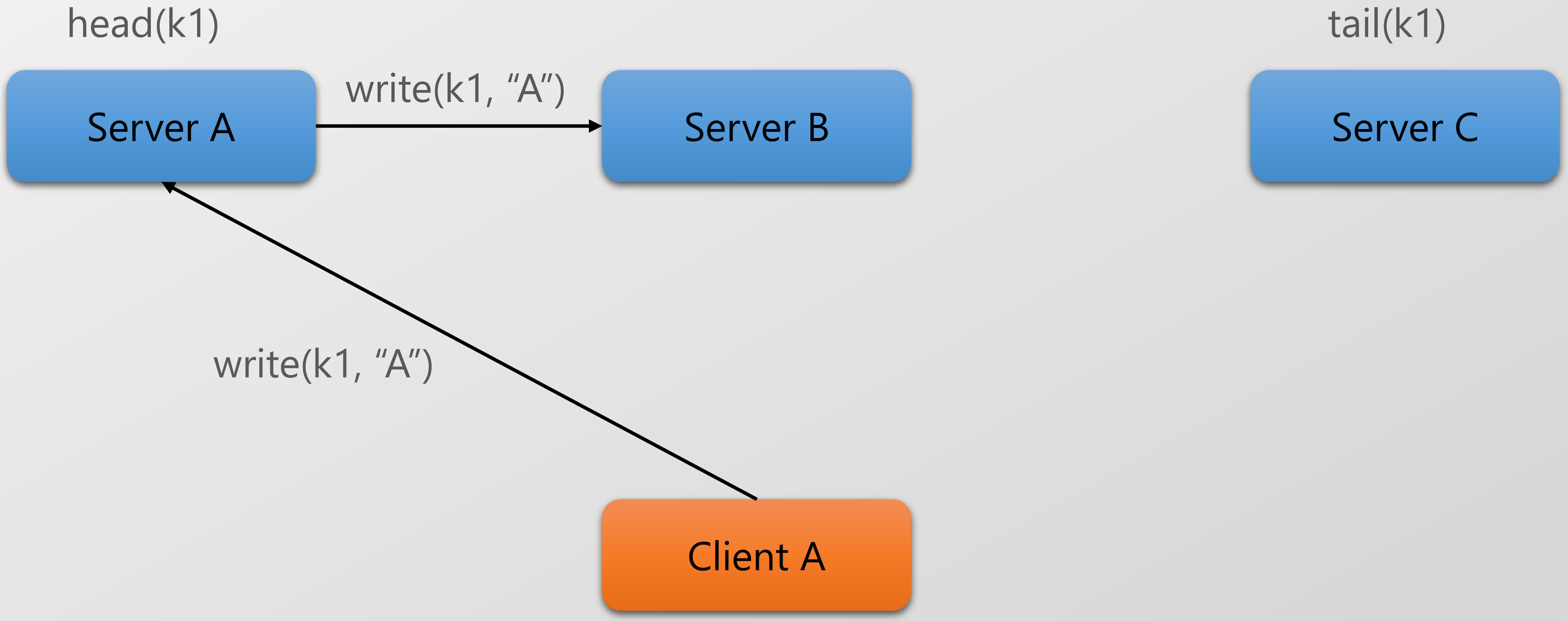
Chain Replication [Renesse and Schneider]

- Sequential writes → No data corruption
- Writes to go head of a chain; reads to the tail

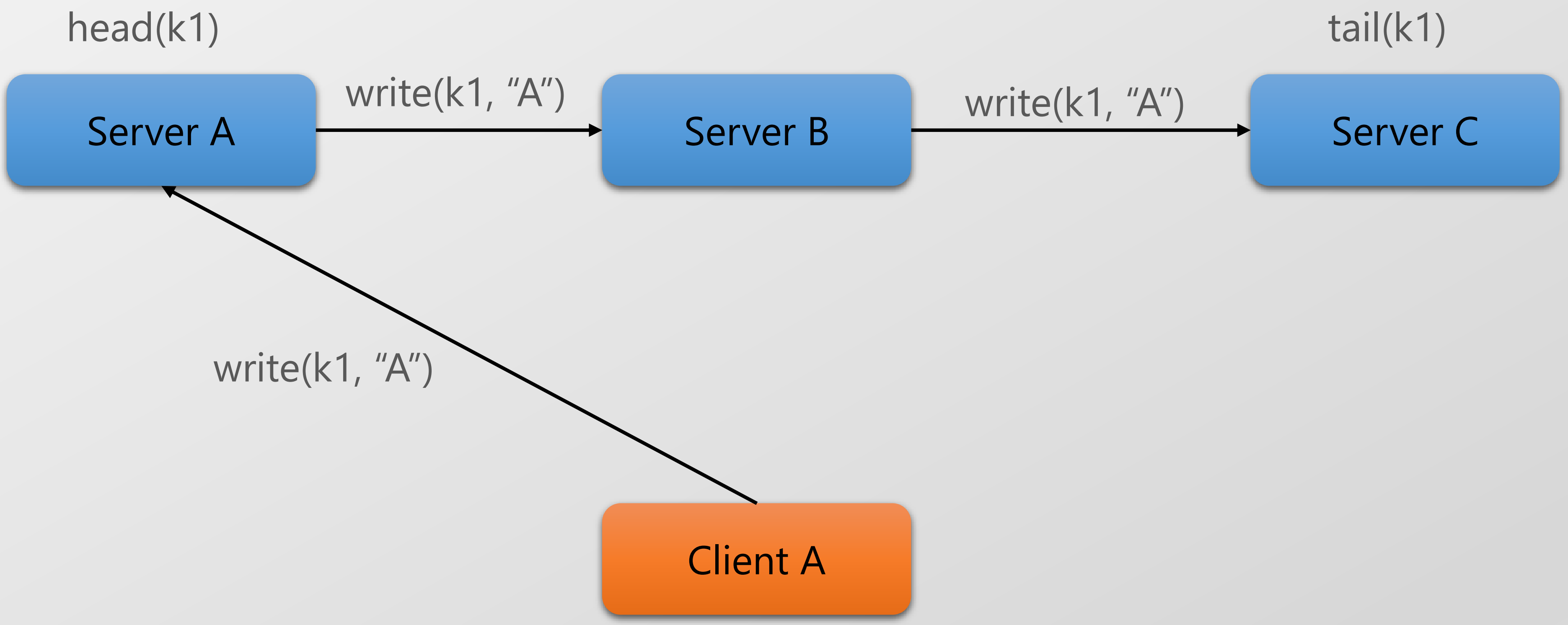
Chain Replication



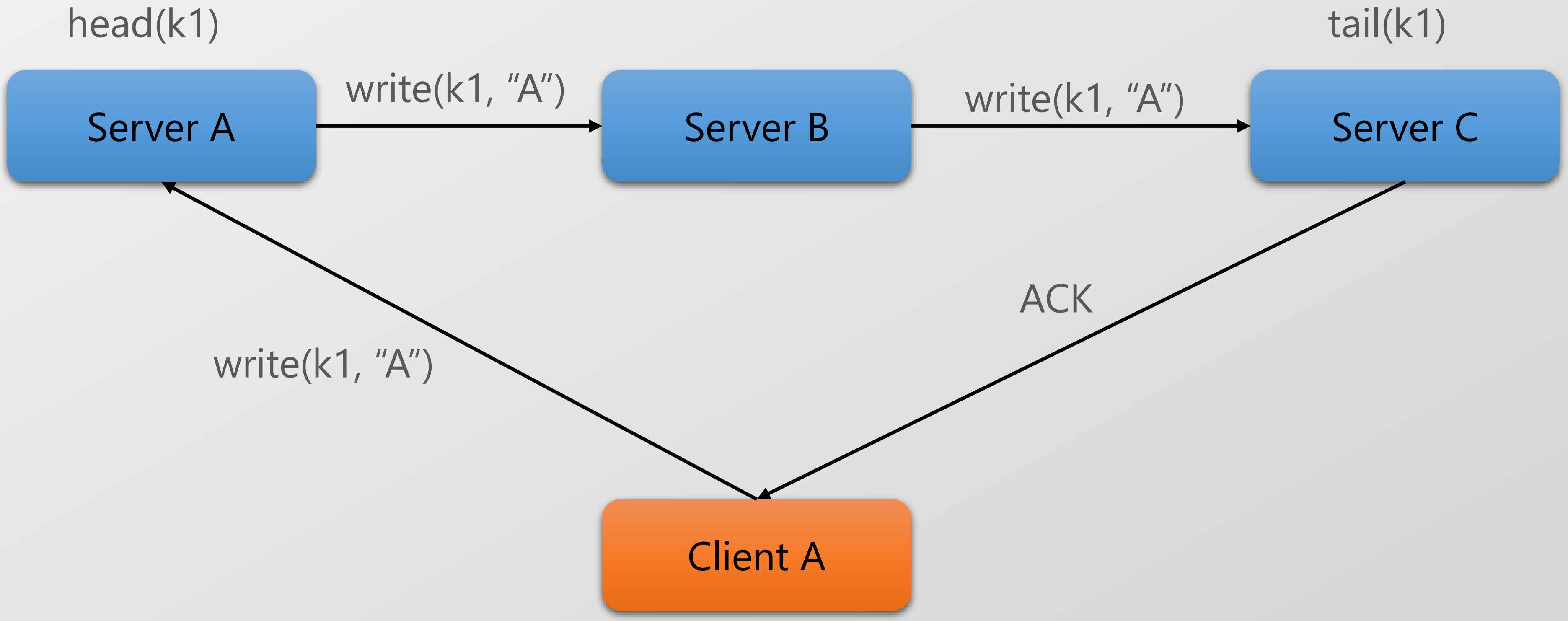
Chain Replication



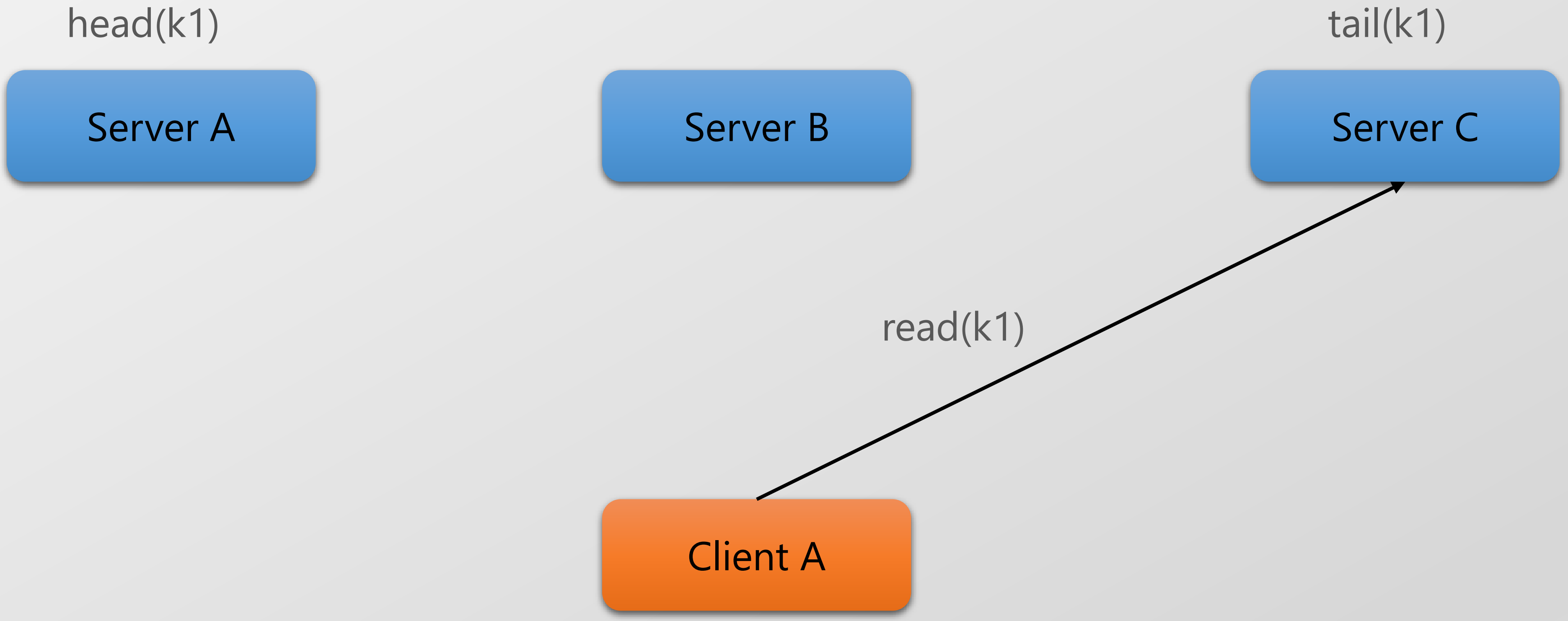
Chain Replication



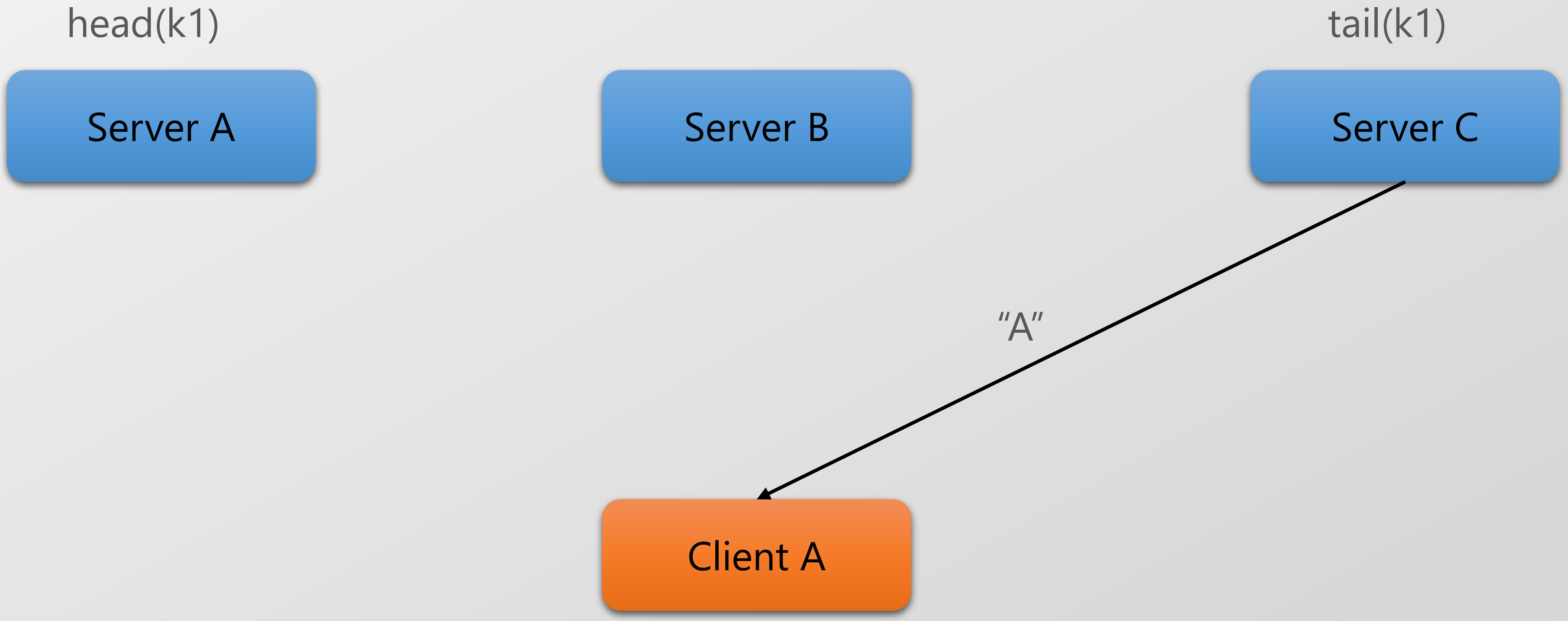
Chain Replication



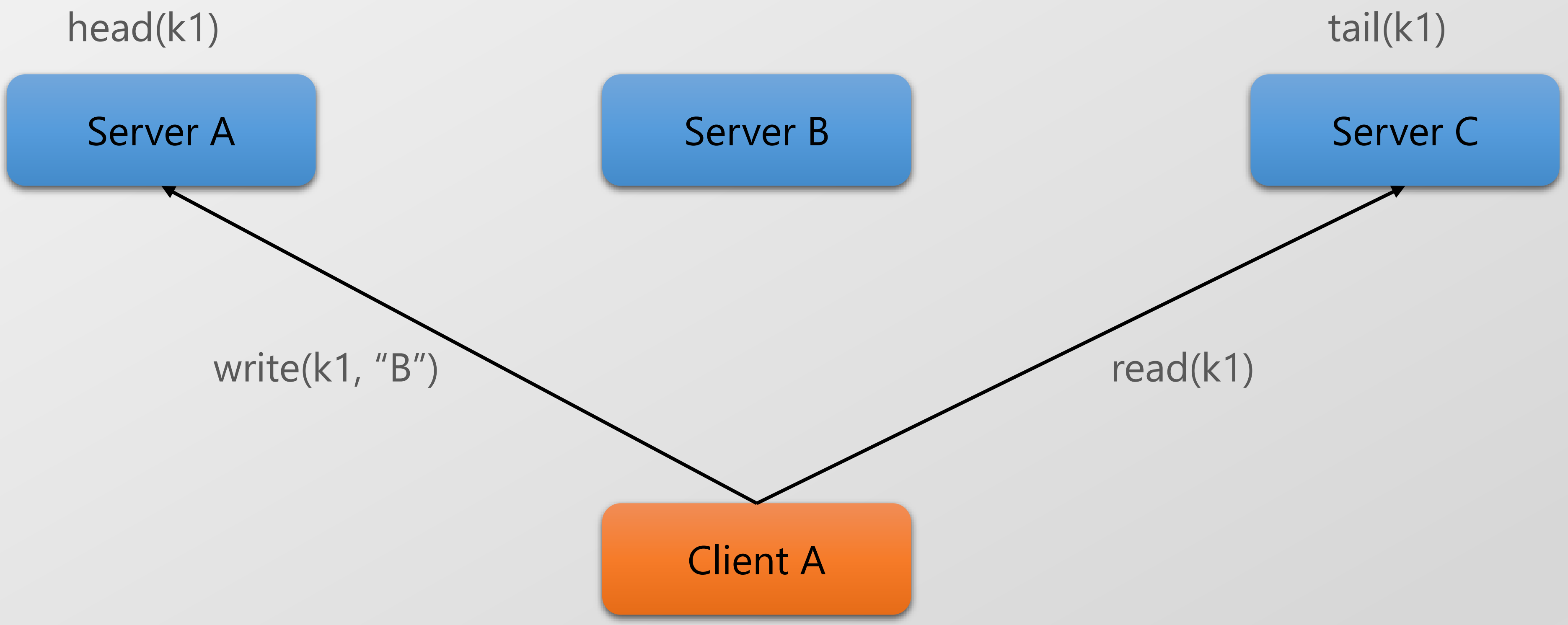
Chain Replication



Chain Replication



Chain Replication



head(k1)

tail(k1)

Server A

Server B

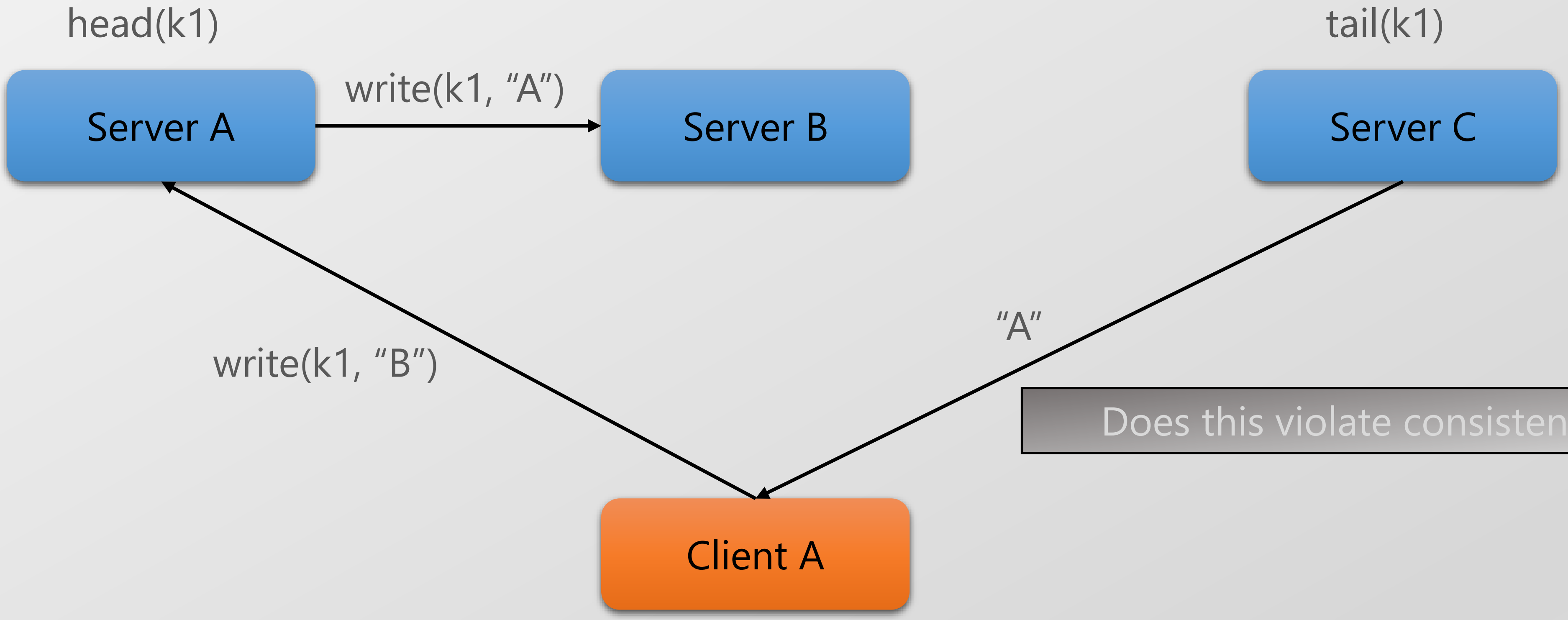
Server C

write(k1, “B”)

read(k1)

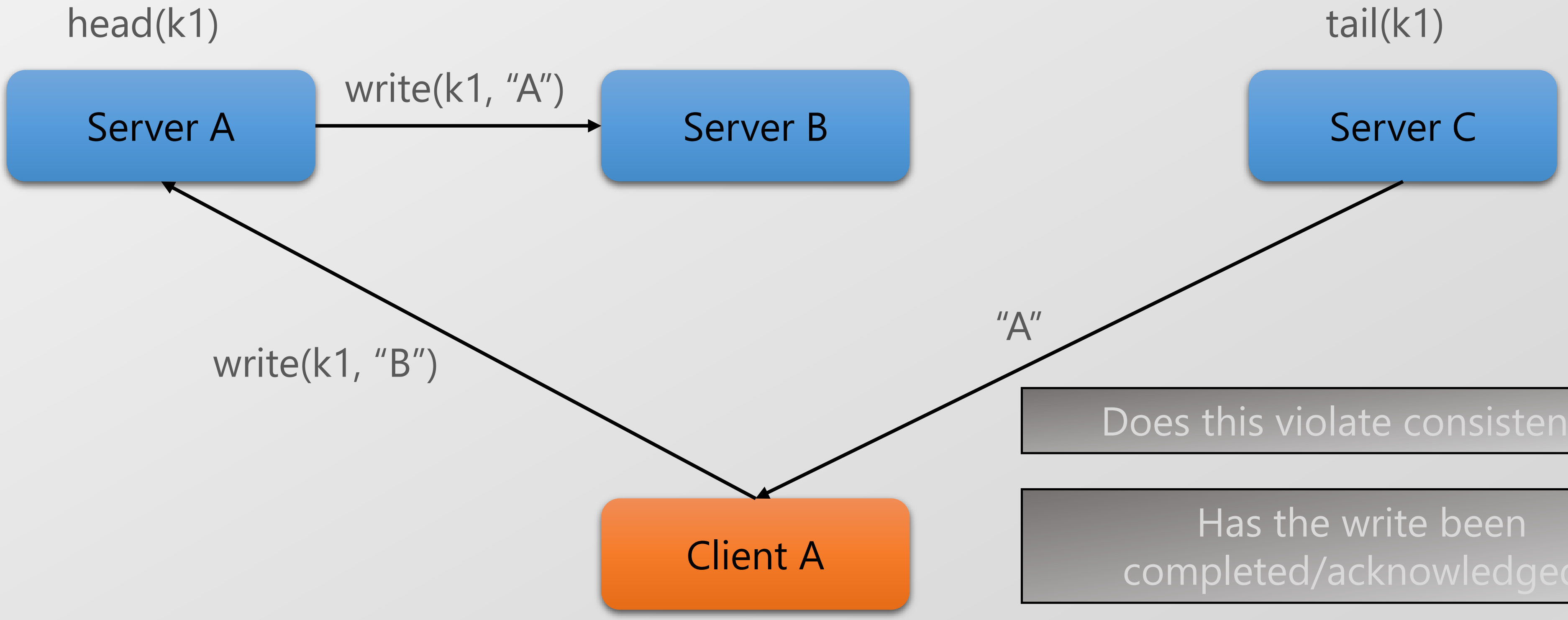
Client A

Chain Replication



Does this violate consistency?

Chain Replication



Does this violate consistency?

Has the write been completed/acknowledged?

Chain Replication

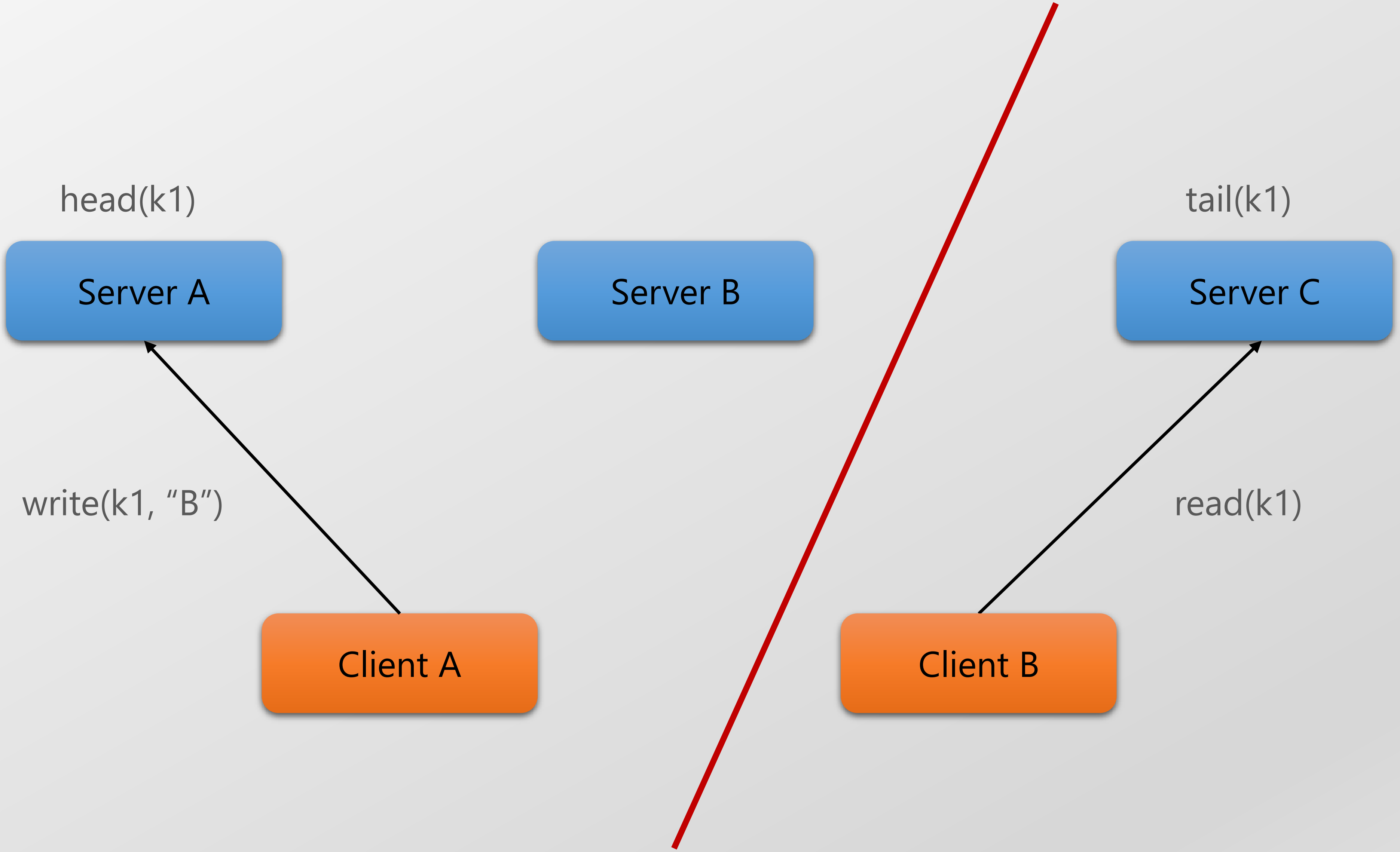
- In-place writes without data corruption, i.e. durability
- CP or AP?

Chain Replication

- In-place writes without data corruption, i.e. durability
- CP or AP?

The approach is intended for supporting large-scale storage services that exhibit high throughput and availability without sacrificing strong consistency guarantees

What about …


head(k1)
tail(k1)
Server A
Server B
Server C
write(k1, “B”)
read(k1)
Client A
Client B

Chain Replication

- In-place writes without data corruption, i.e. durability
- CP or AP?

Our chain replication does not offer graceful handling of partitioned operation, trading that instead for supporting all three of: high performance, scalability, and strong consistency.

Workarounds

- Consistency hard to reason about!
- Send all requests via the master?

Pros and Cons

- In-place writes → Reduce metadata updates
- Sequential writes → Latency amplification

Consistency *and* Availability? [Escriva and Gun Sirer]

What CAP is simplified to:

- You must give up consistency or availability at all times

What the CAP theorem really says:

- If you cannot limit the number of faults
- and requests can be directed to any server
- and you insist on serving every request
- then you cannot possibly be consistent

Limit failures or kind of partition?

Conclusions

- Distributed Storage needs to reason about consistency and availability
- CAP is an analysis tool, not a design principle!
- Within bounds of failures, all three are possible
- Implementation details matter too!