Leases and Cache Coherence

Leases

Lease - a time-limited right to do something

- can be renewed
- unlike Paxos, depends on loosely synchronized clocks

Lease fault tolerance

- if lease holder or network fails, wait for lease to expire
- plus epsilon to account for clock drift
- hand lease to someone new

Paxos as Lease Server

Paxos group as fault tolerant view server

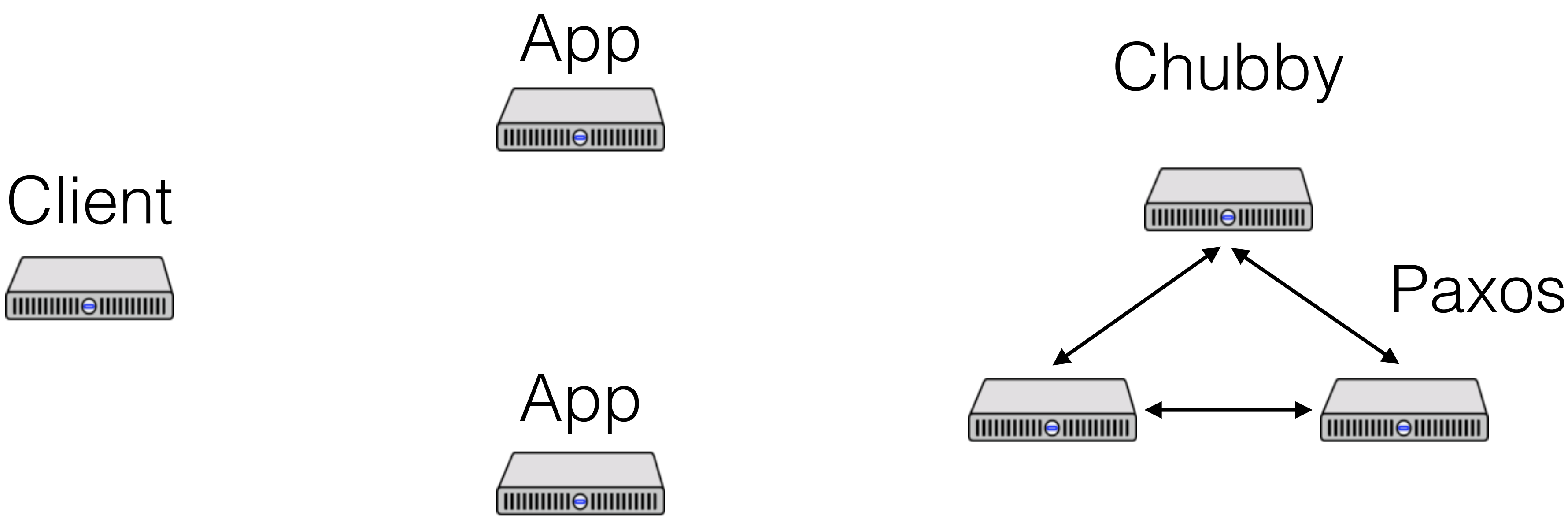
- grant lease to primary
- primary serves requests
- revoke lease if not renewed
- grant lease to new primary

Design pattern used in GFS, BigTable, …

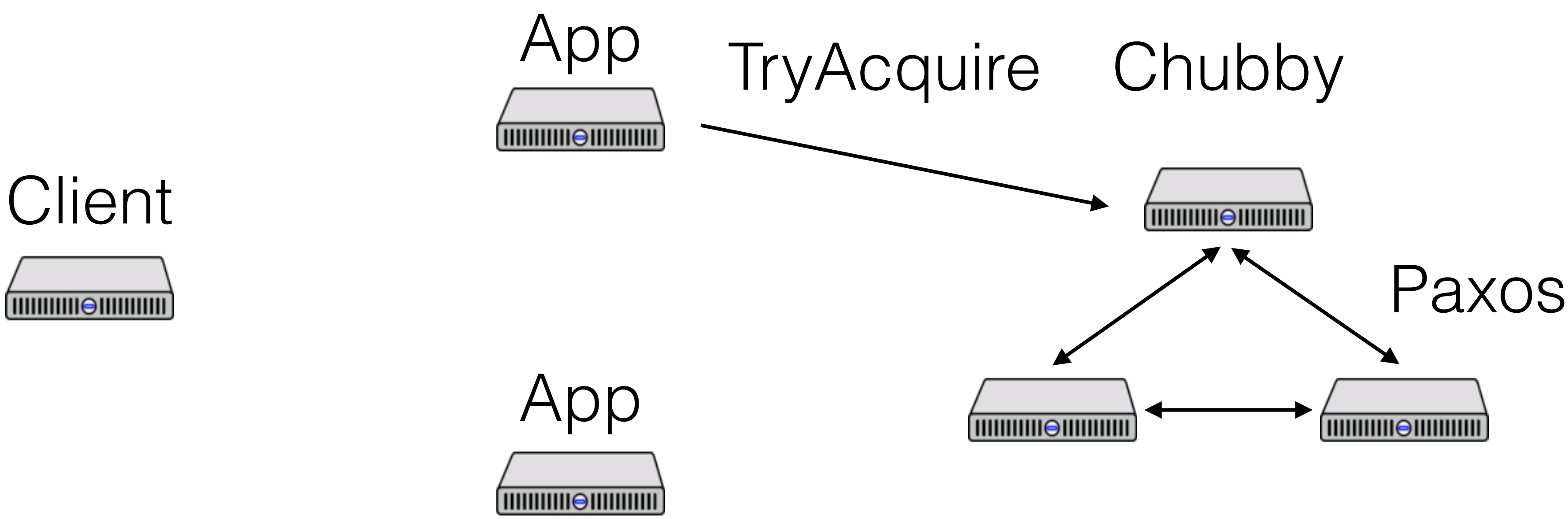
Primary election in Chubby, Zookeeper

```
x = Open("/BigTable/primary")
if (TryAcquire(x) == success) {
  // I'm the primary, tell everyone
  SetContents(x, my-address)
} else {
  // I'm not the primary, find out who is
  primary = GetContents(x)
  // also set up notifications
  // in case the primary changes
}
```

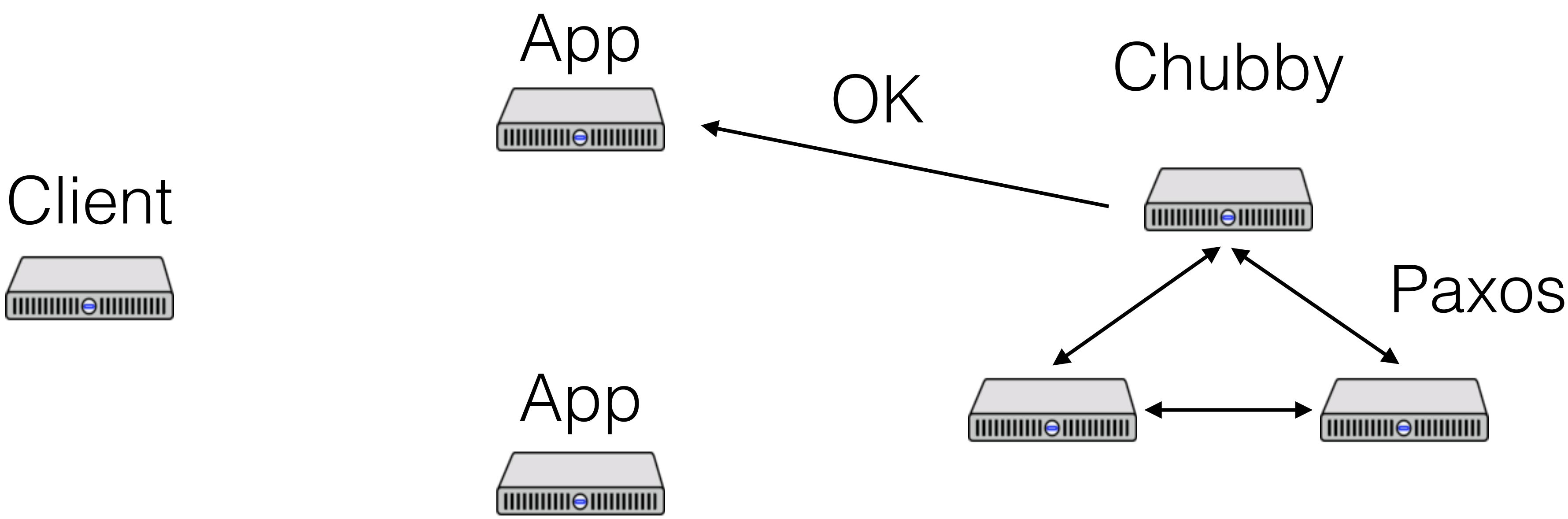
Example



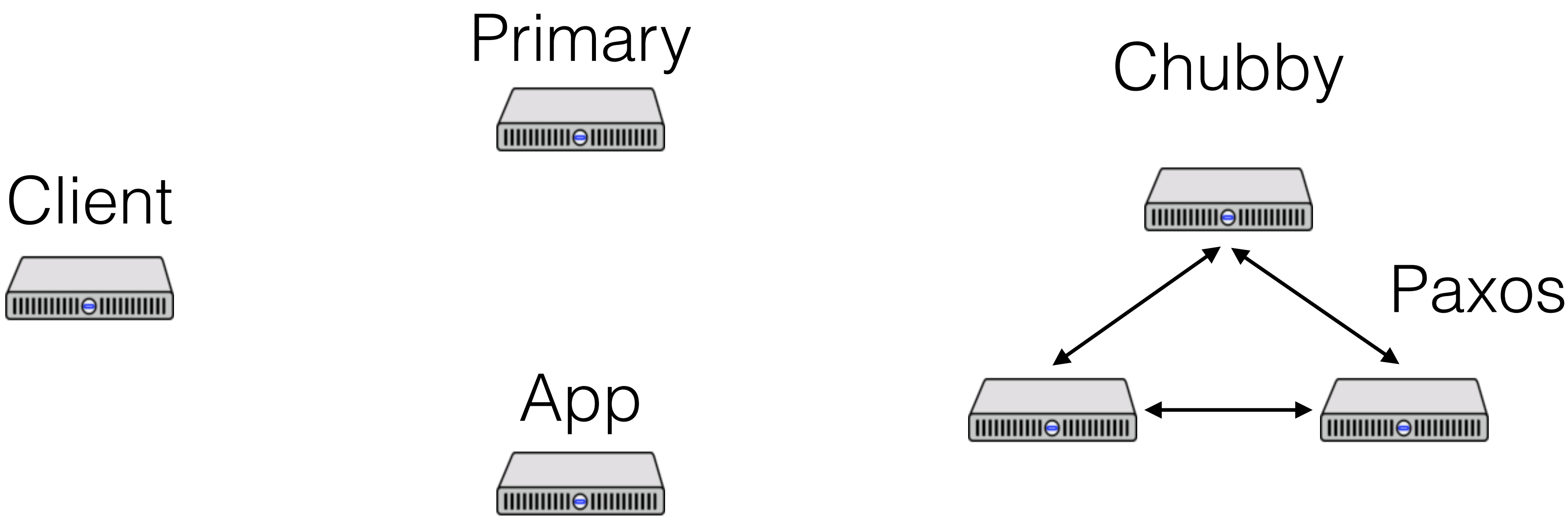
Example



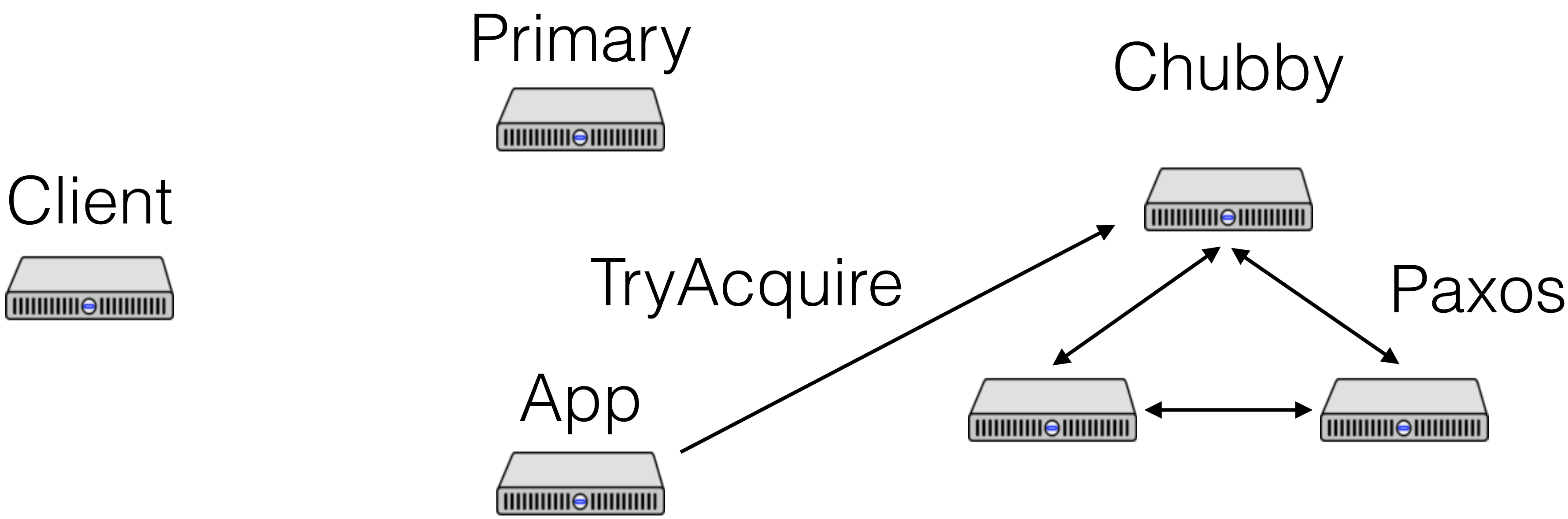
Example



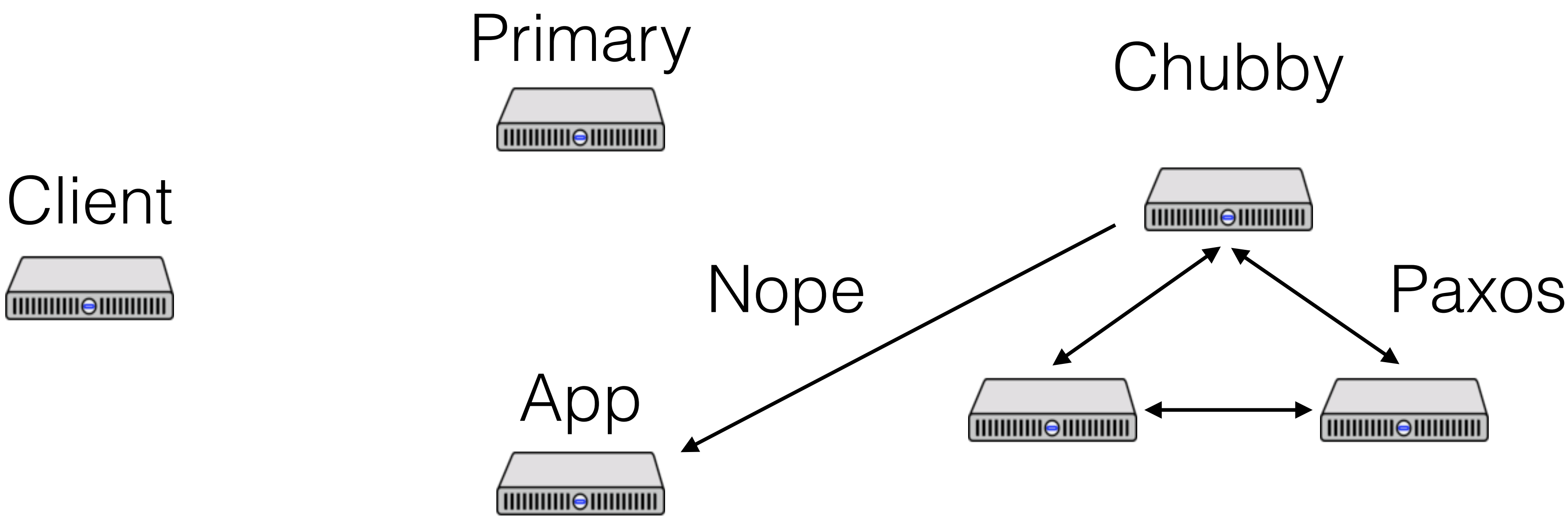
Example



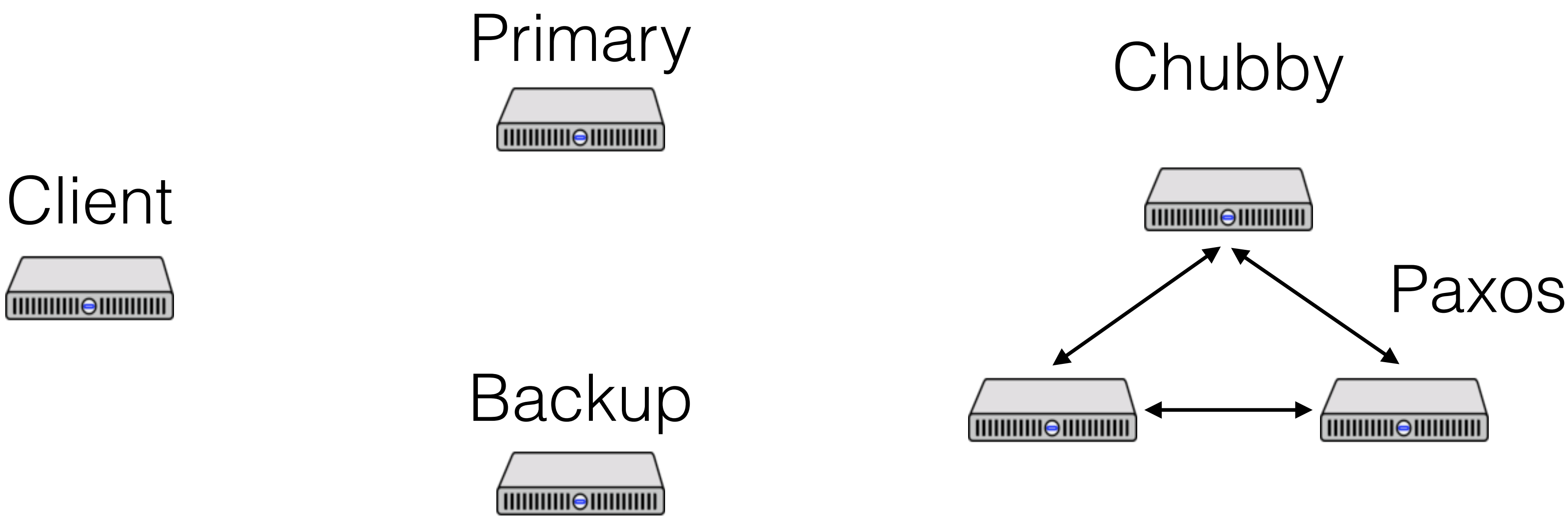
Example



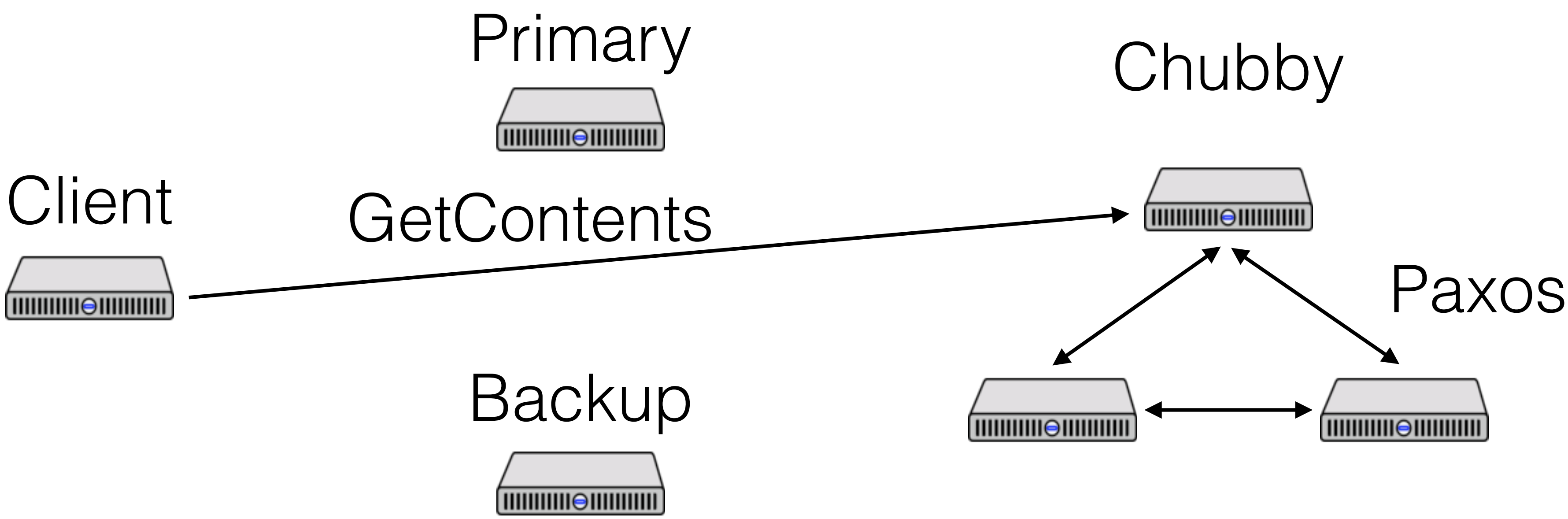
Example



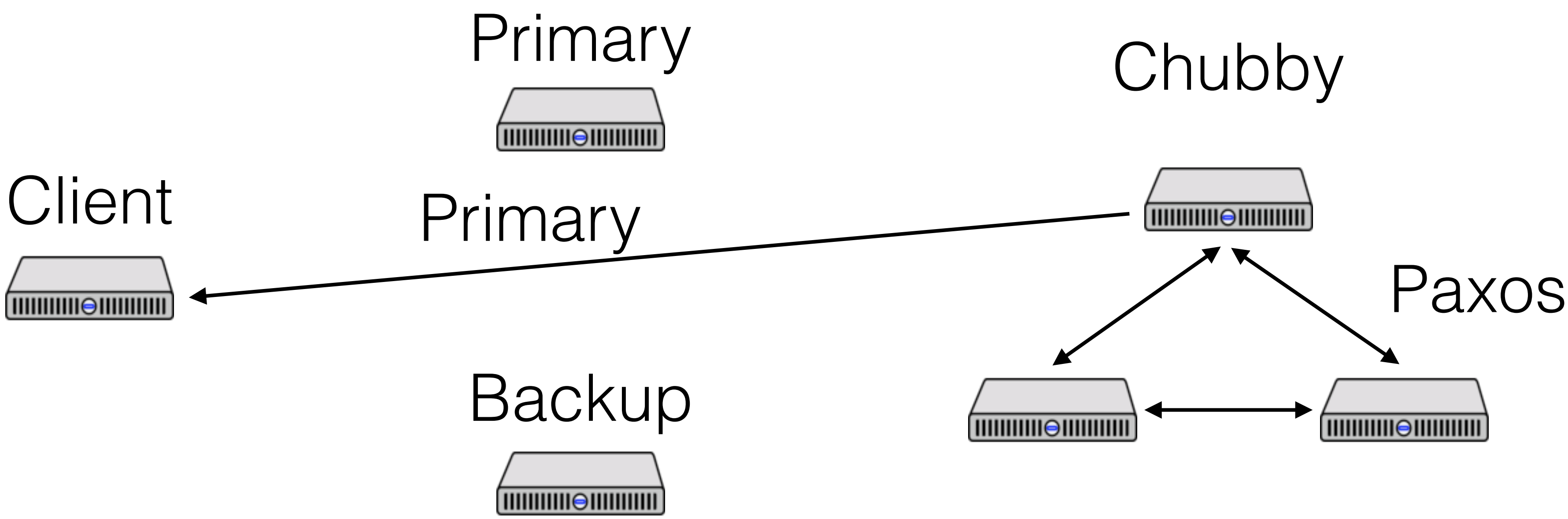
Example



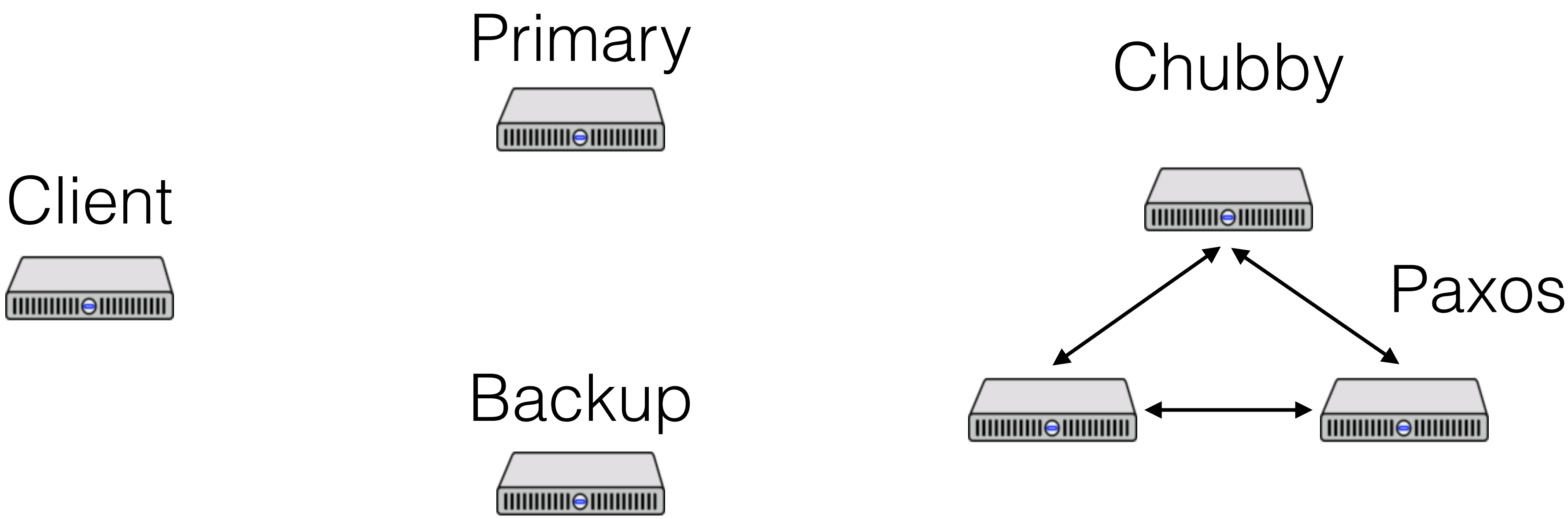
Example



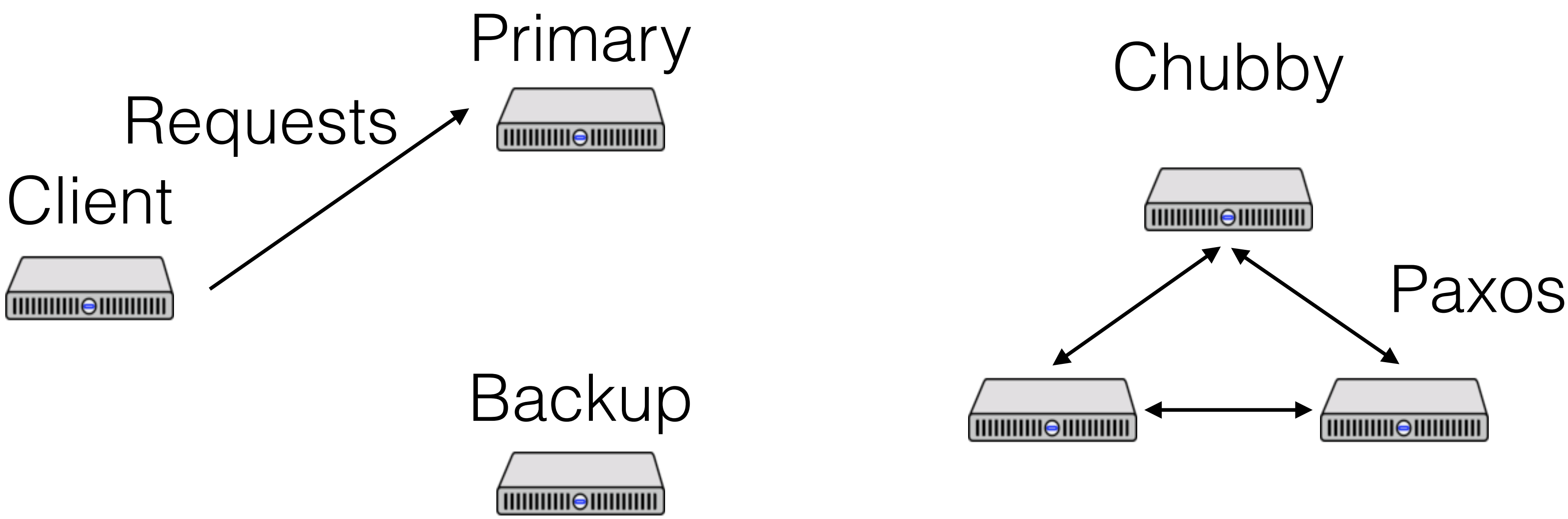
Example



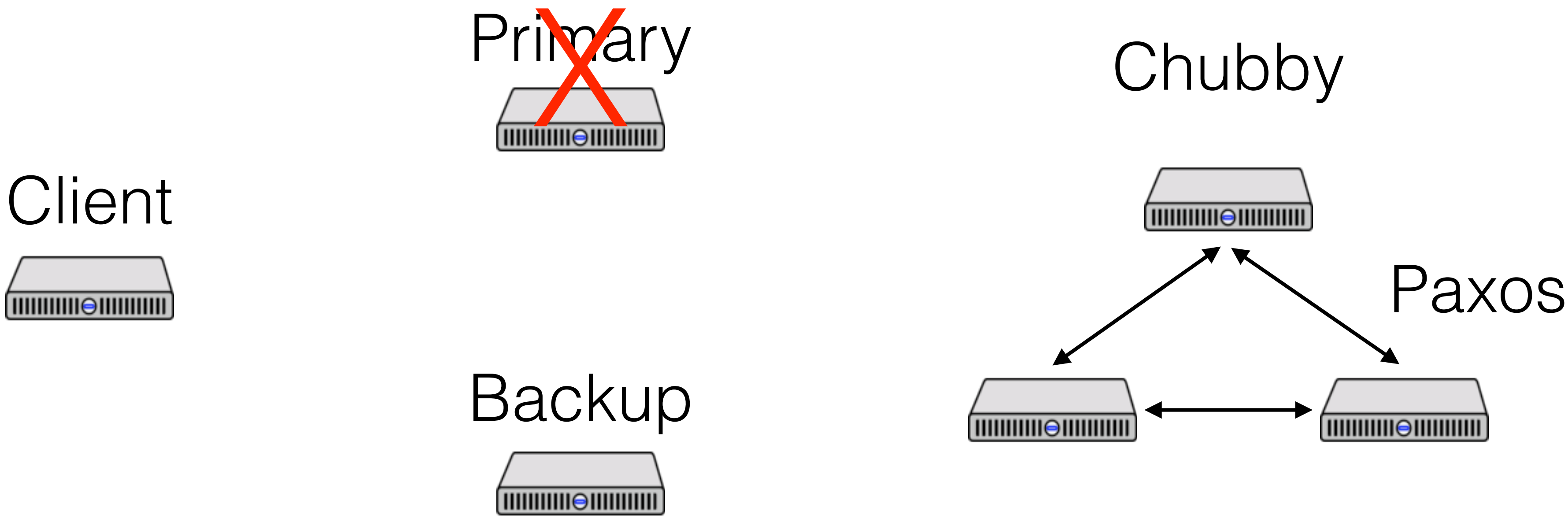
Example



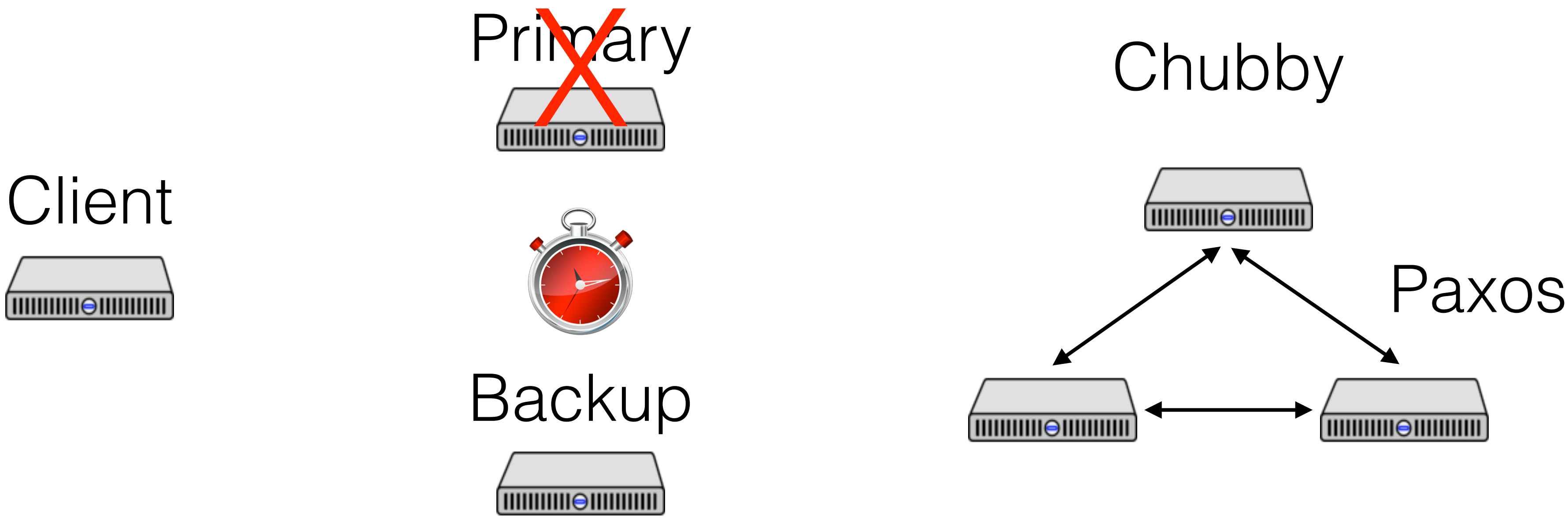
Example



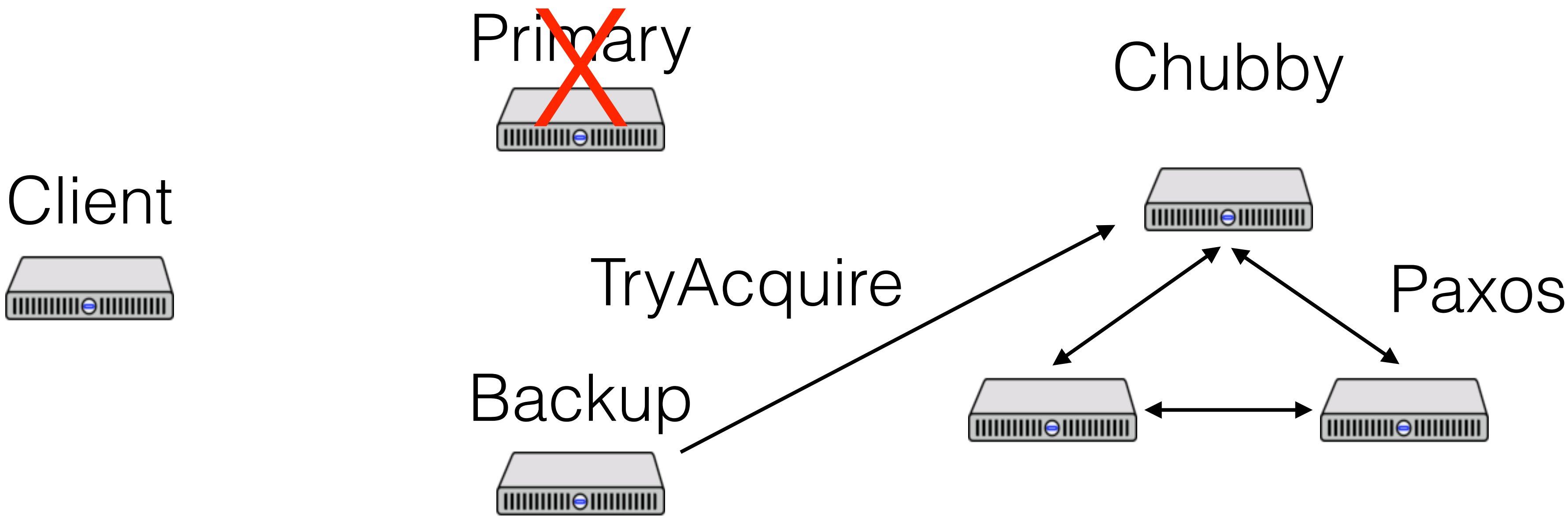
What if Primary Fails?



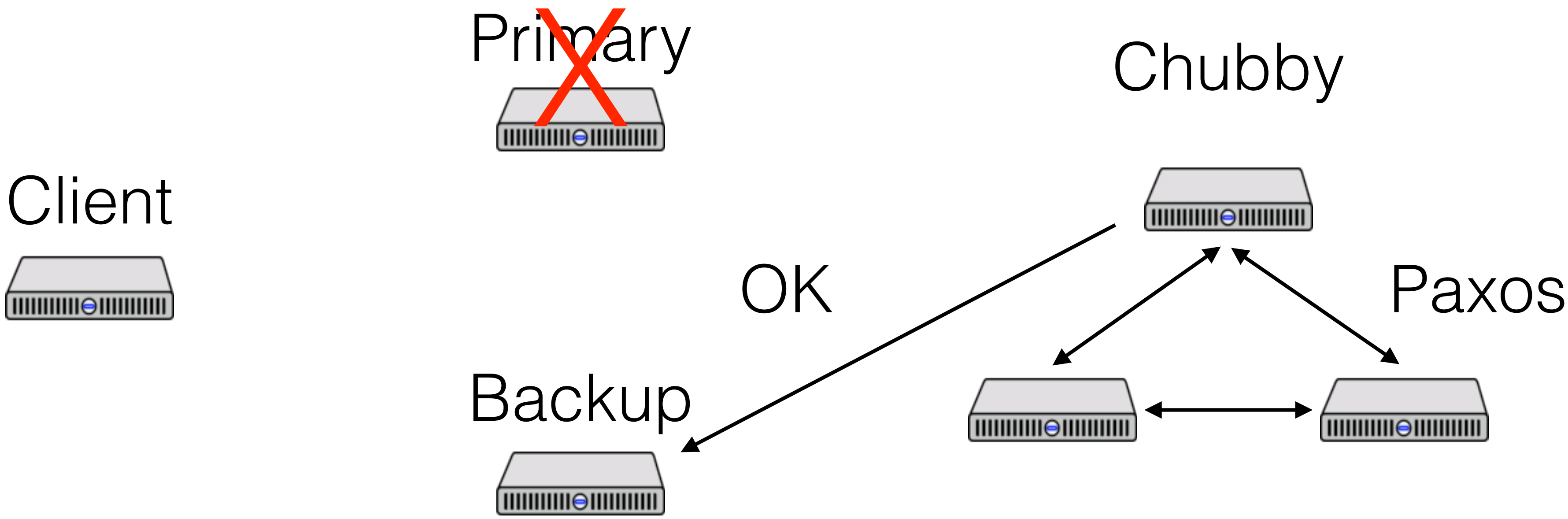
What if Primary Fails?



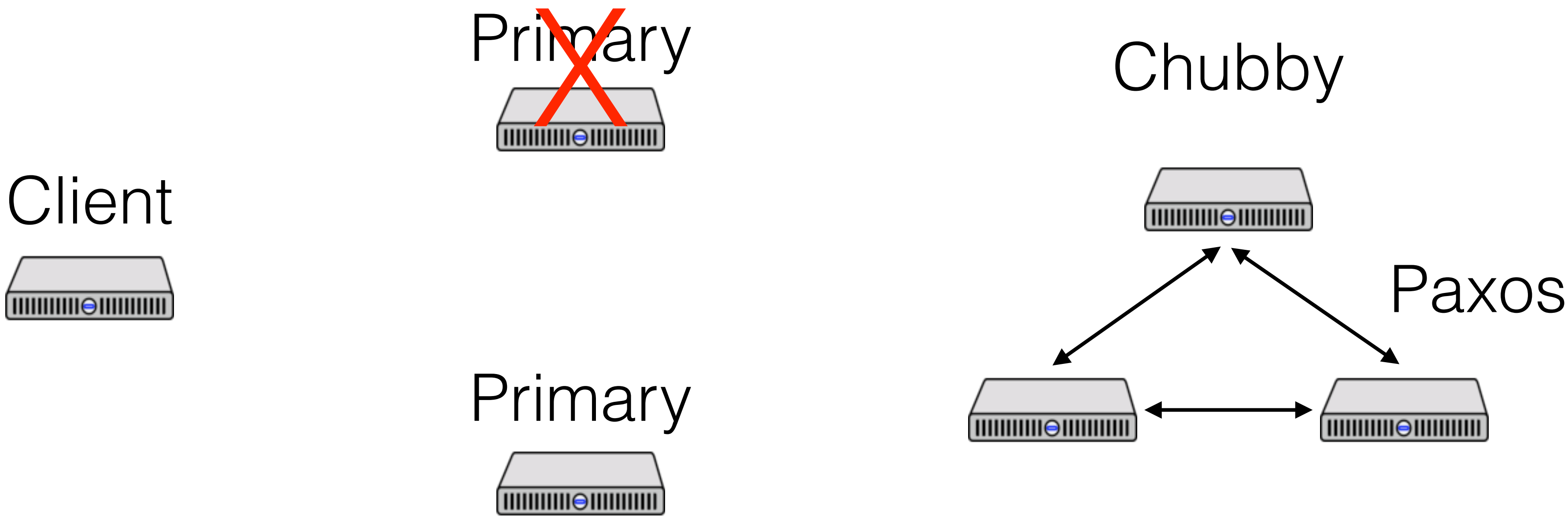
What if Primary Fails?



What if Primary Fails?



What if Primary Fails?



Primary Backup With Leases

What if the old primary didn't crash?

Client sends request to old primary

What keeps old primary from performing op?

Primary Backup With Leases

What if the old primary didn't crash?

Client sends request to old primary

What keeps old primary from performing op?

Old primary demotes itself if it doesn't renew lease

Primary Backup with Leases

No possibility of split brain

Reads can occur at the primary!

- no need to talk to backup

Writes can be logged to storage layer

- on failure, new primary reads latest changes from storage layer
- backup is optimization to speed recovery

Fault Tolerant Caching with Leases

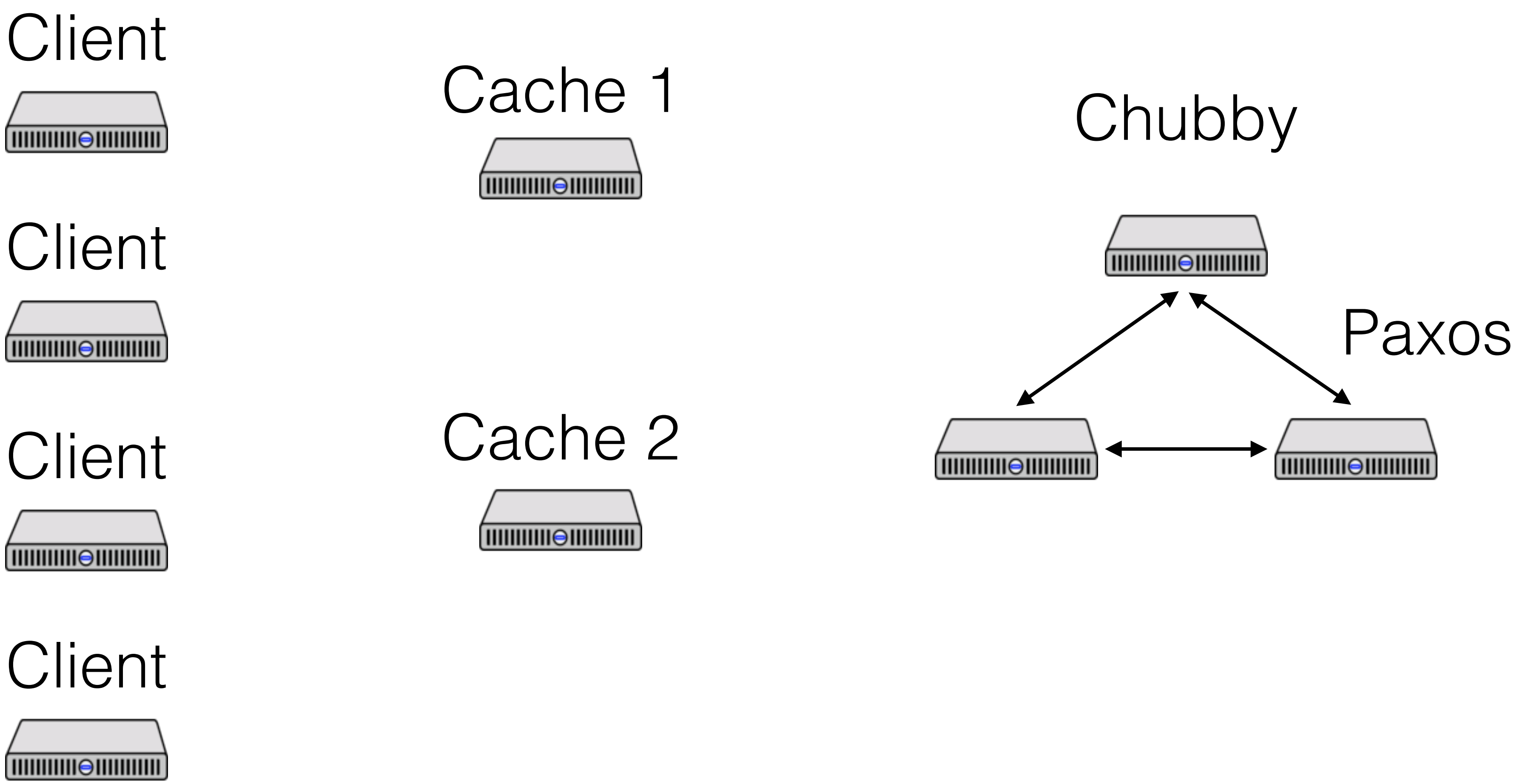
Linearizability with caches is another use of leases

Cache obtains lease (ex: read-only)

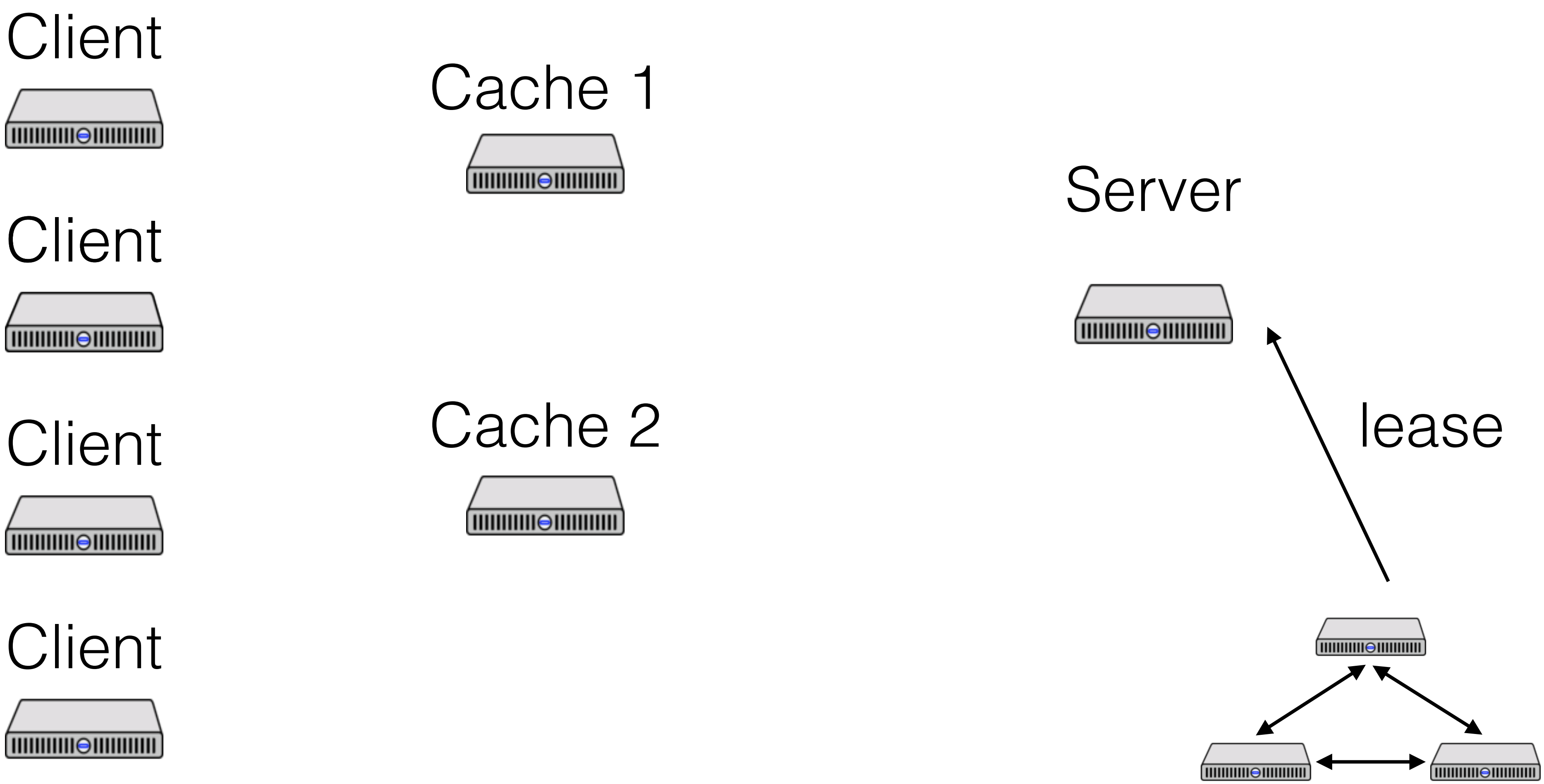
No one can modify data until lease expires or is revoked

Once lease expires, ok for server to change

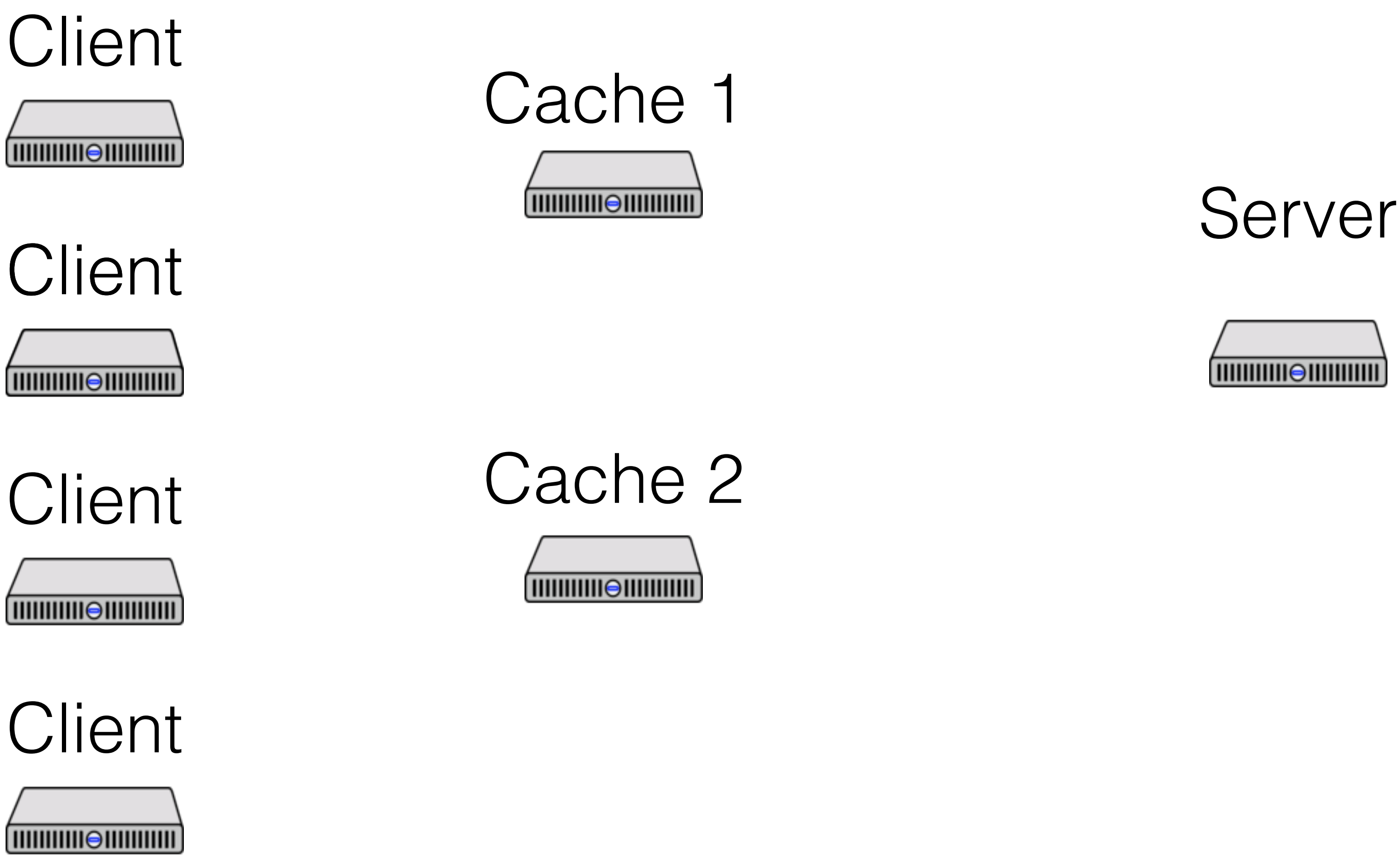
Caching With Leases



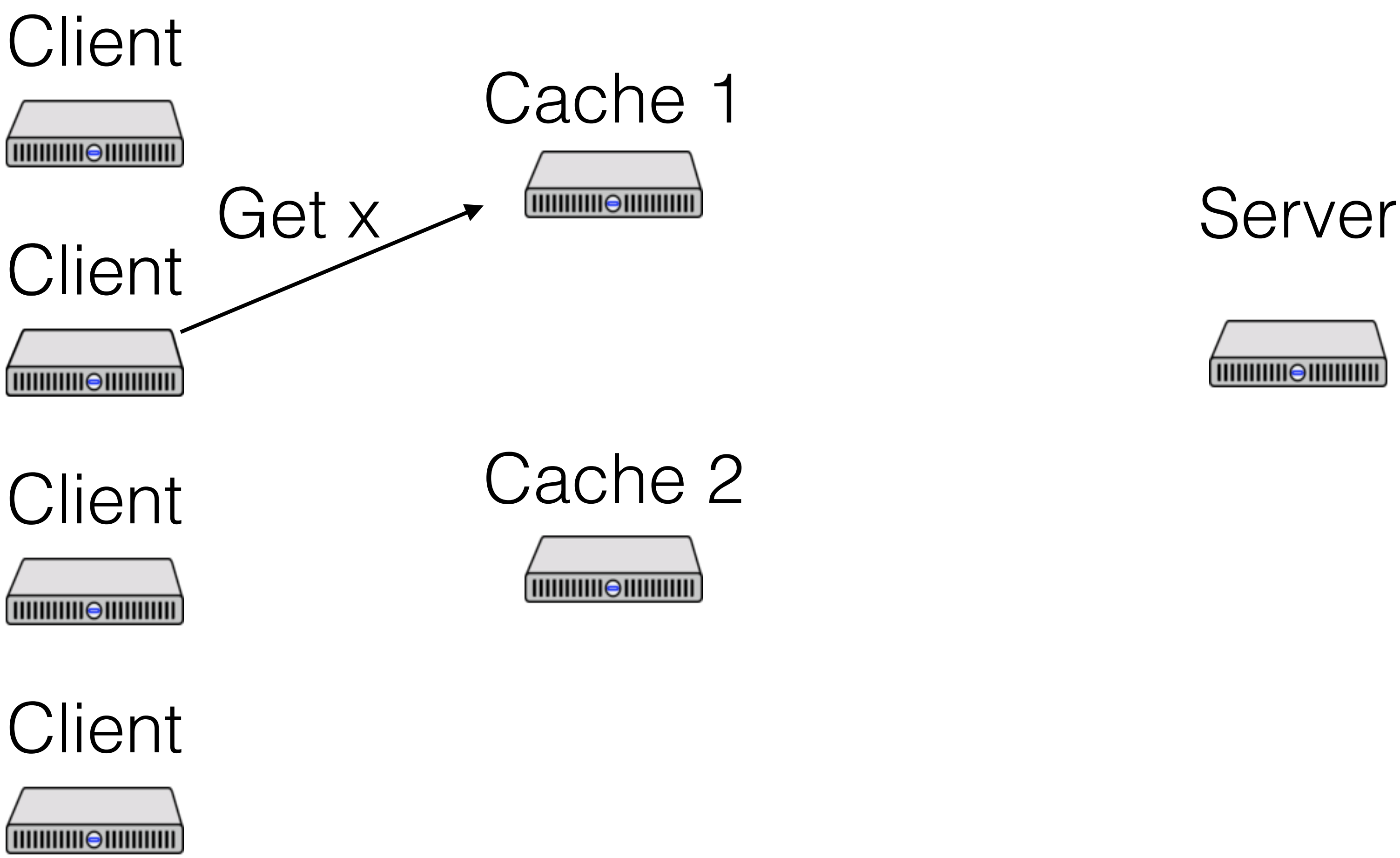
Caching With Leases



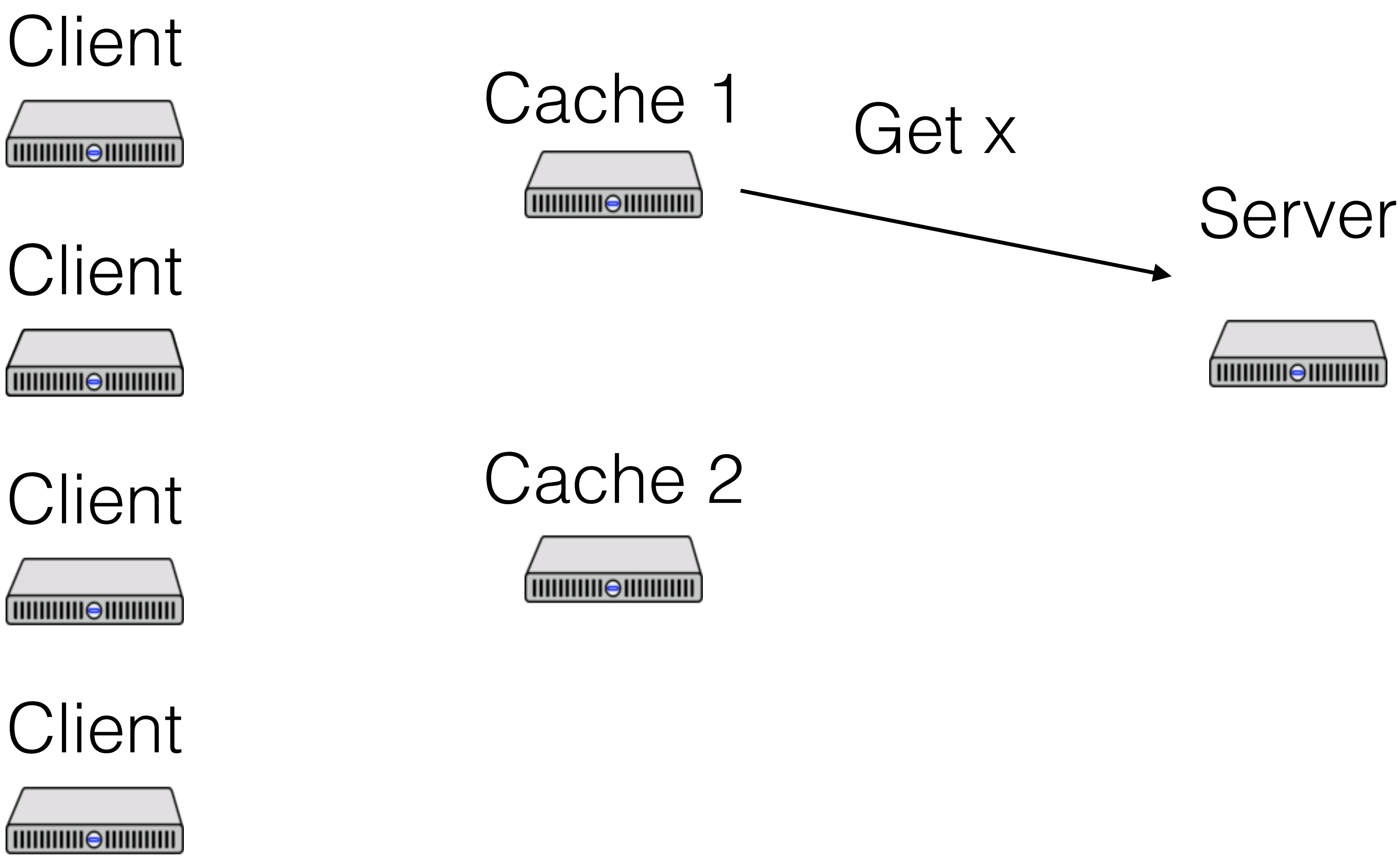
Caching With Leases



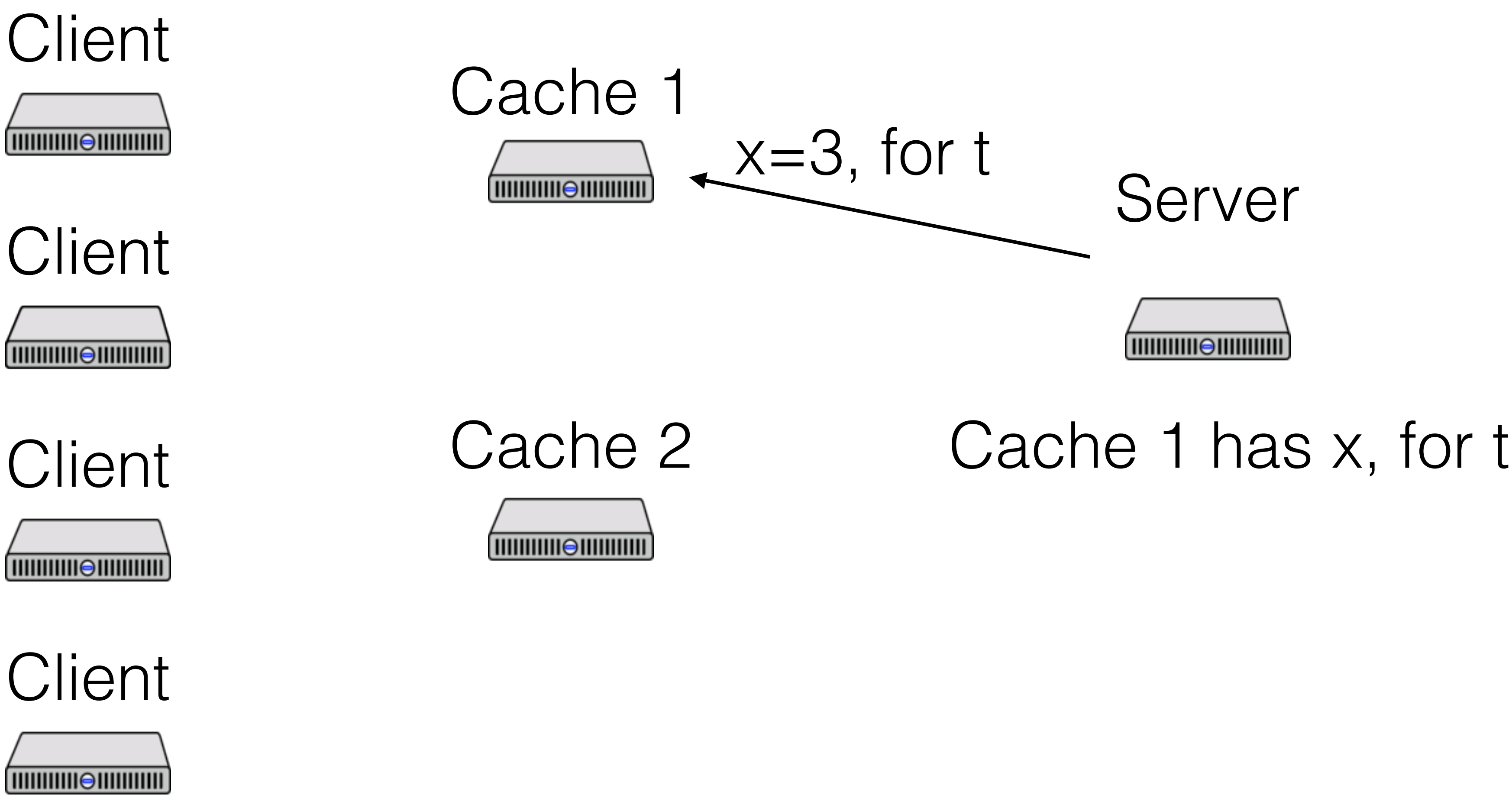
Caching With Leases



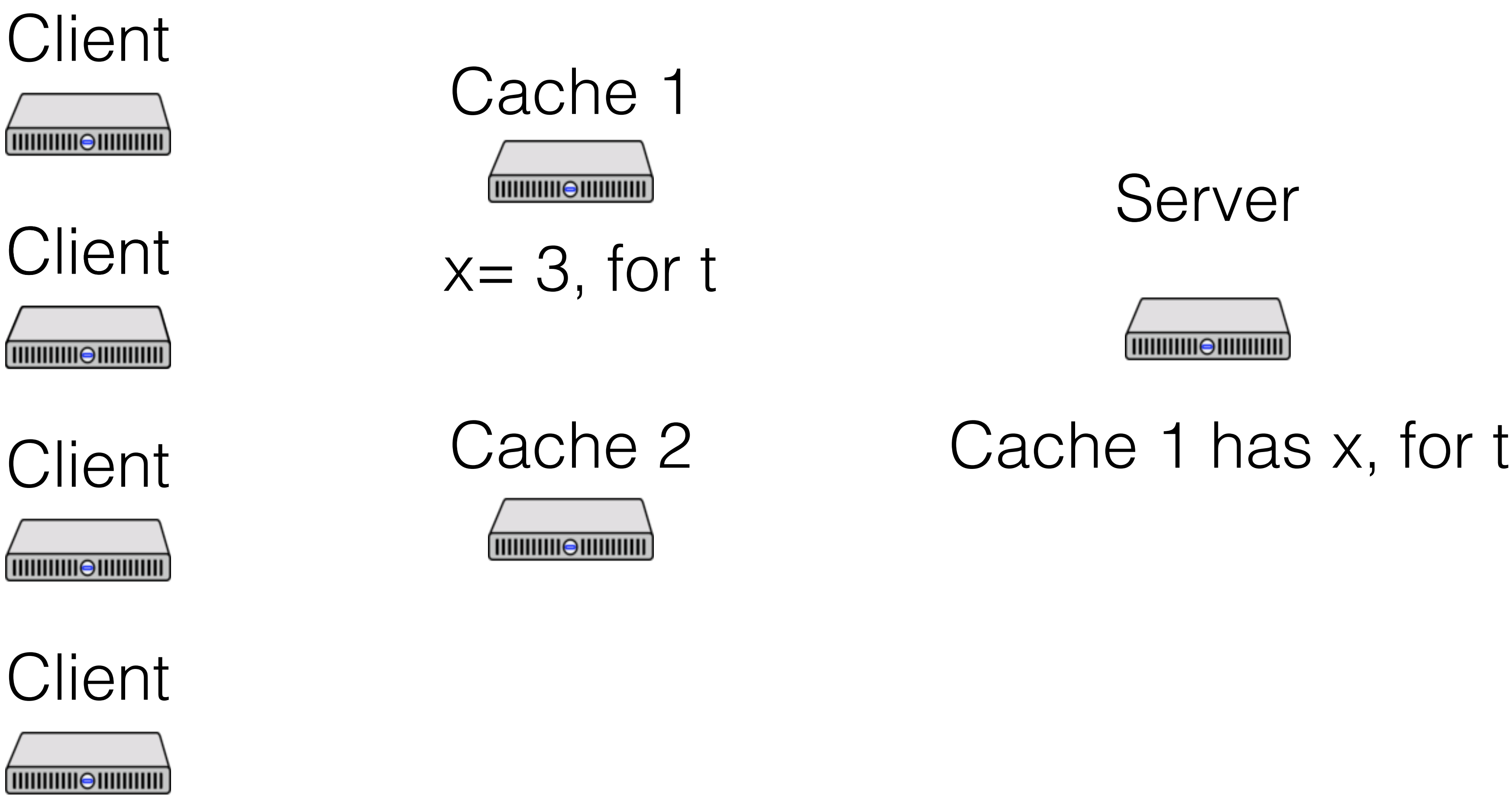
Caching With Leases



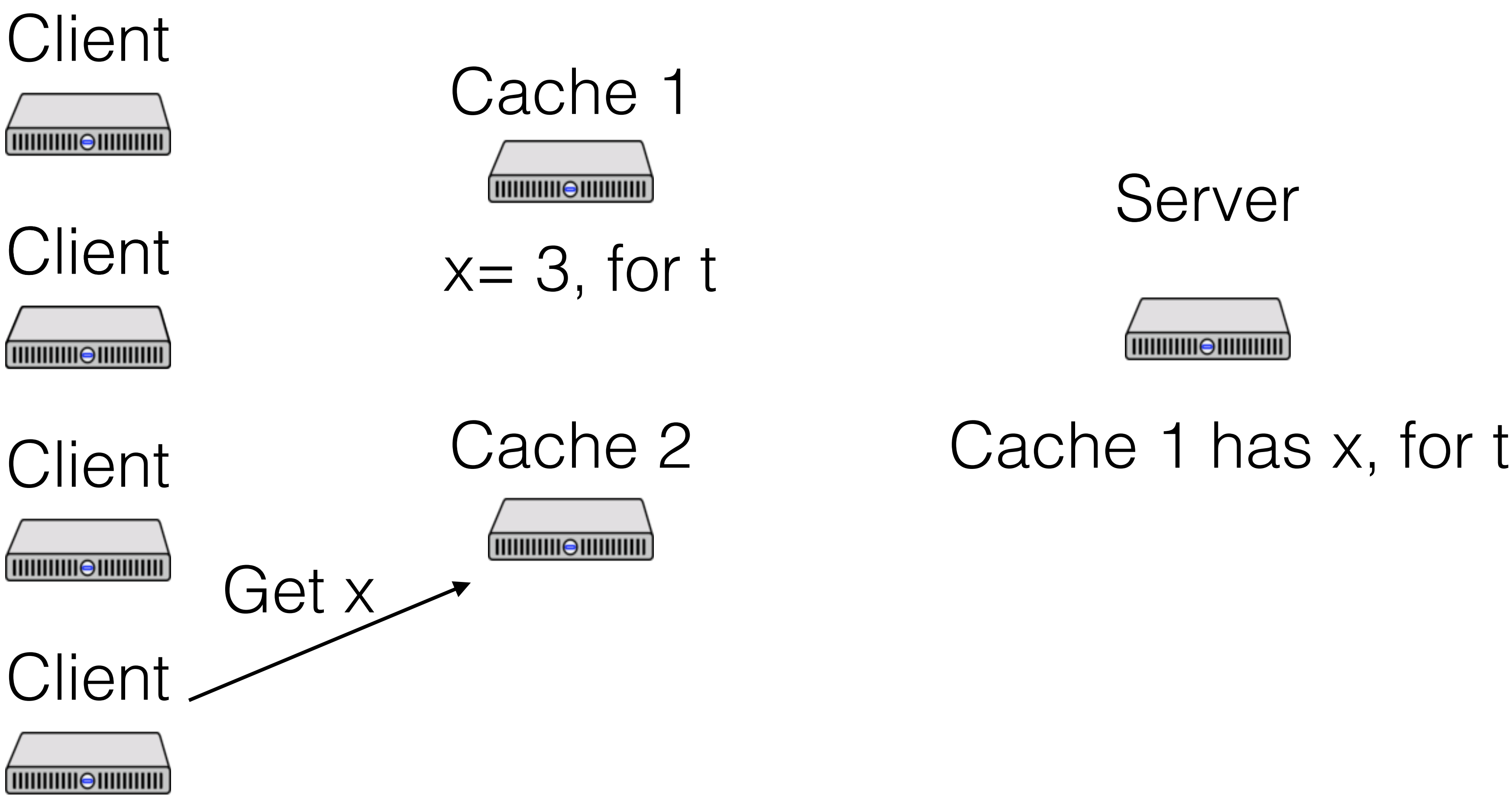
Caching With Leases



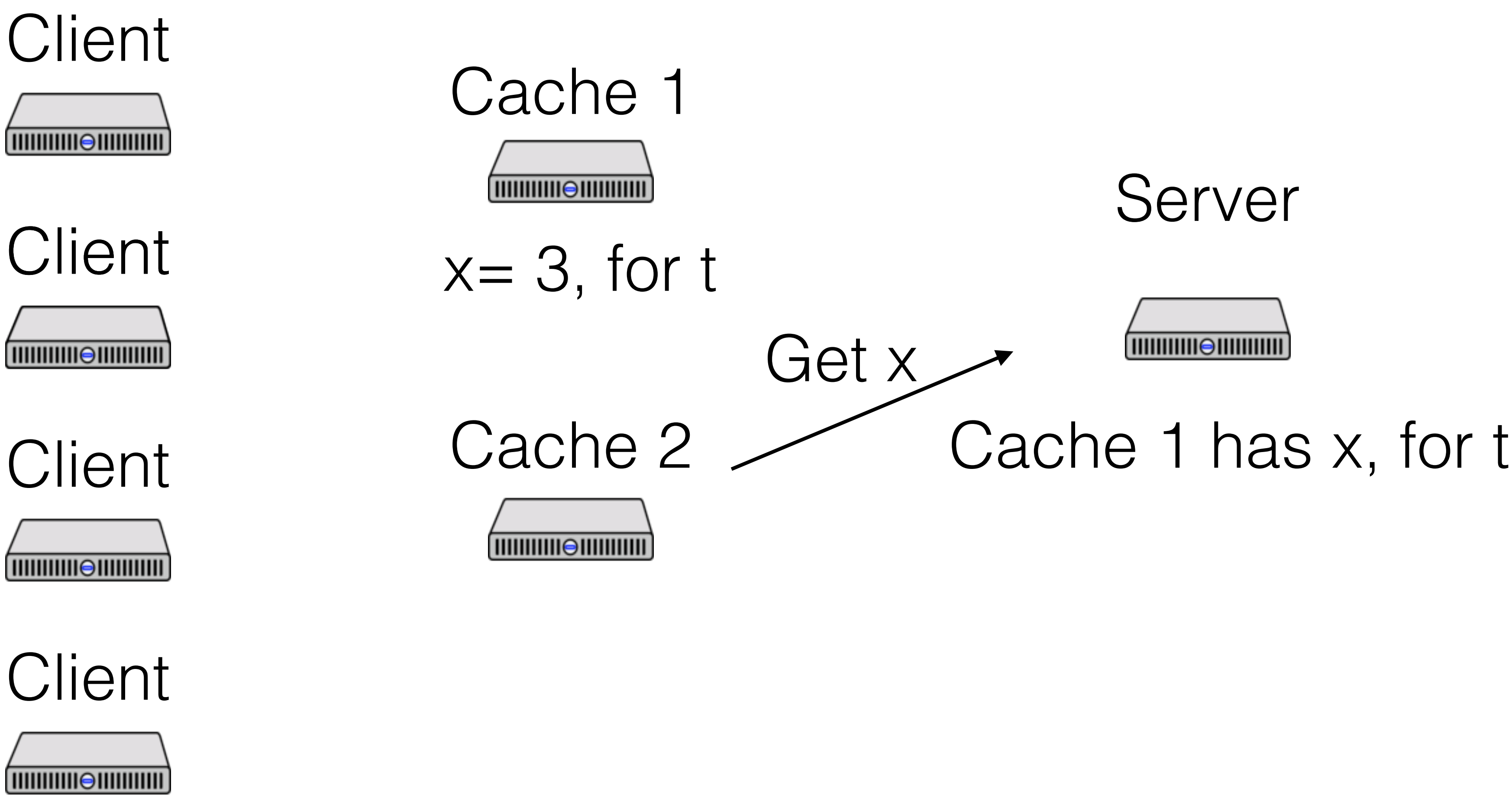
Caching With Leases



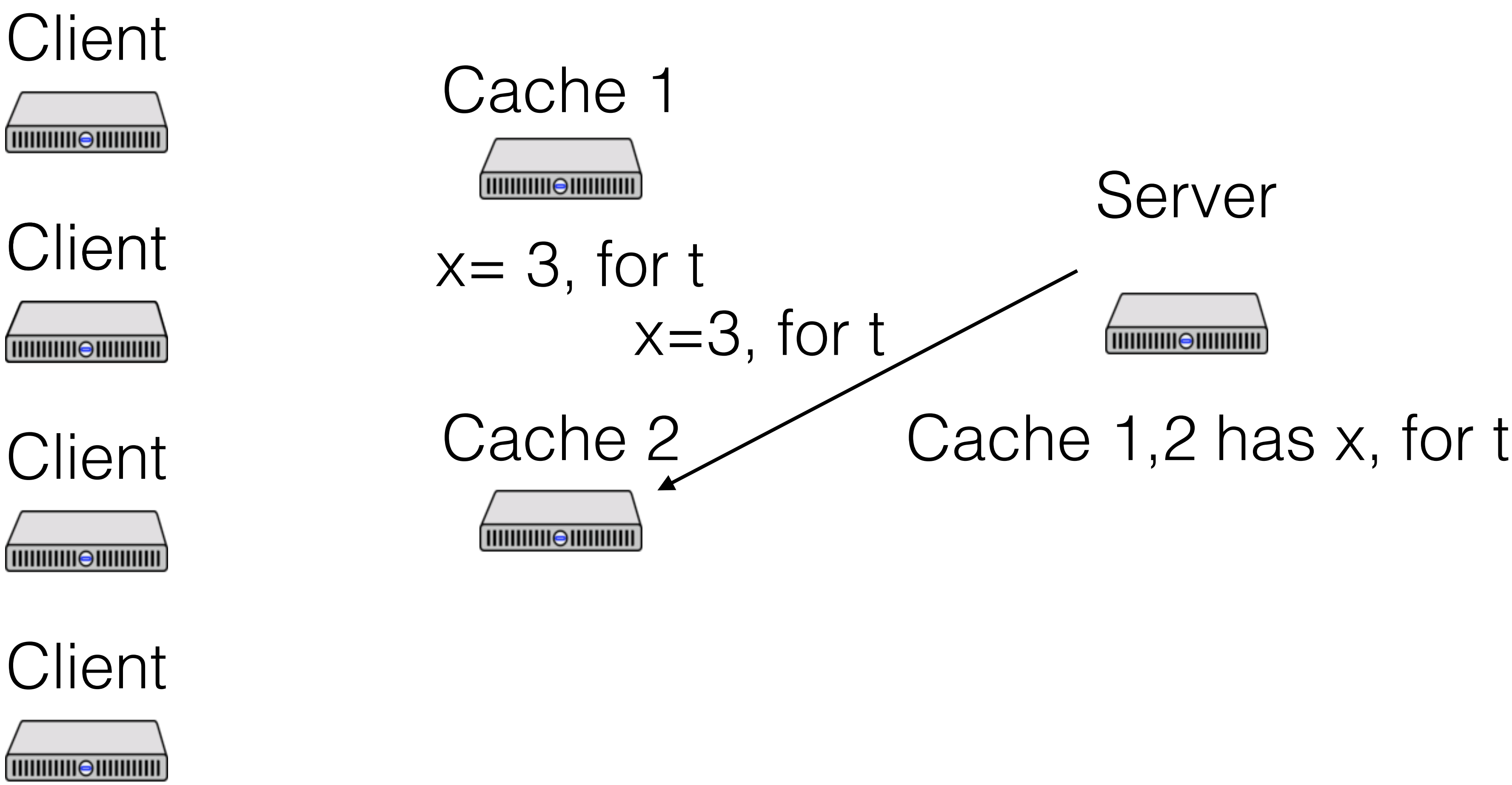
Caching With Leases



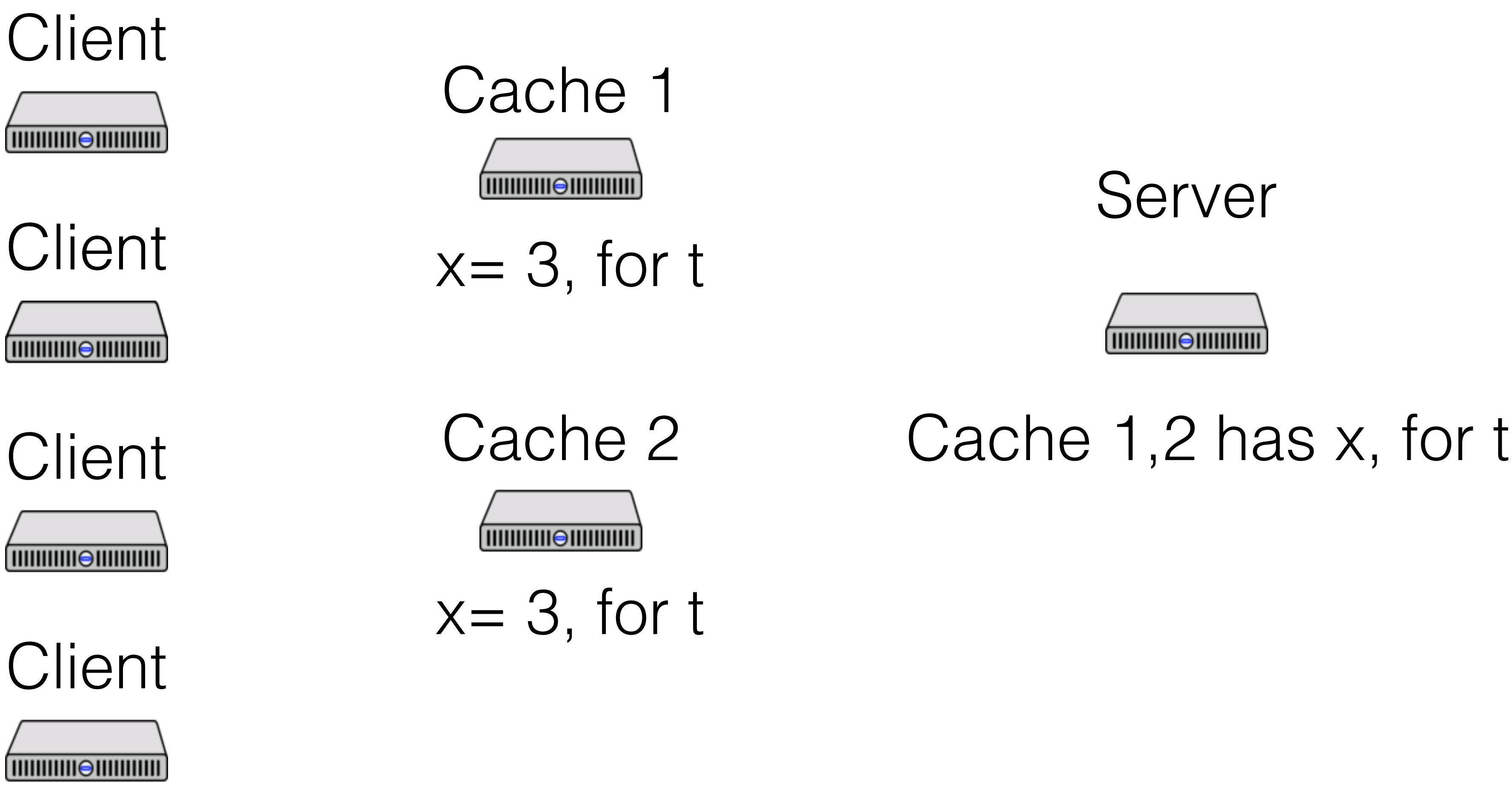
Caching With Leases



Caching With Leases



Caching With Leases



Caching With Leases

Why give out cache leases with same values of t?

Why give out cache leases with different values of t?

Caching With Leases

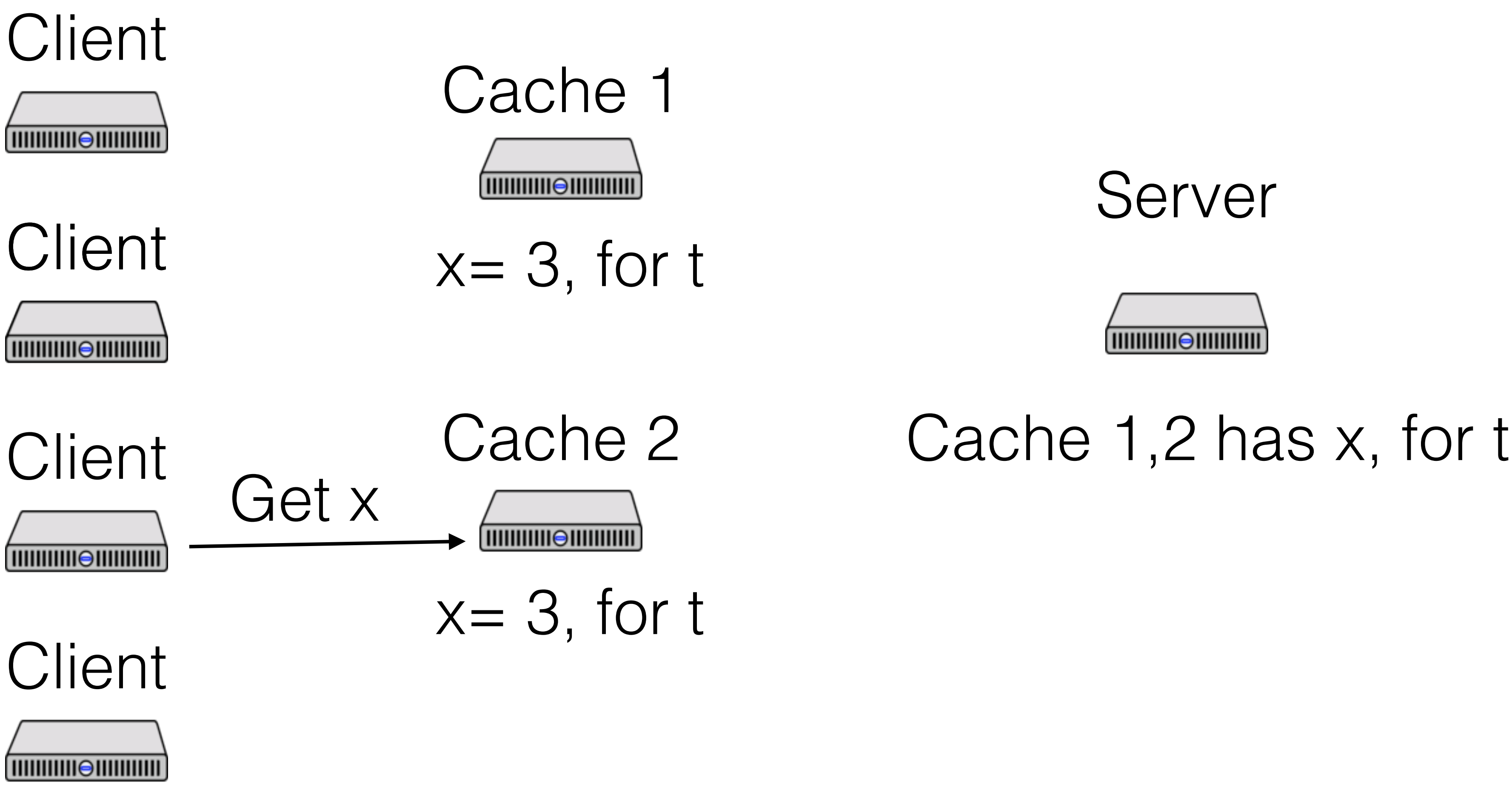
Why give out cache leases with same values of t?

- less state at server
- can reclaim leases at same time

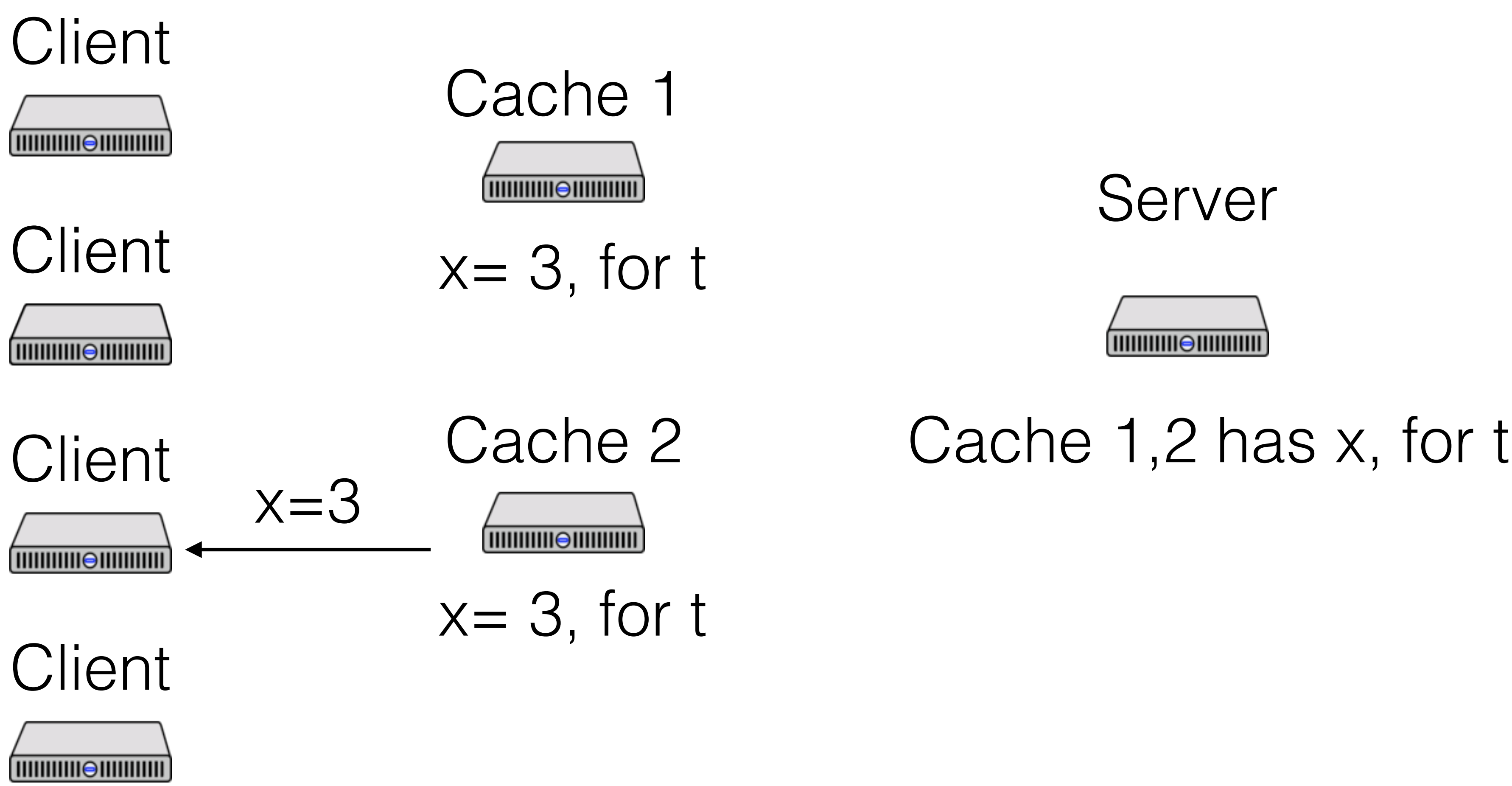
Why give out cache leases with different values of t?

- caches all ask for new lease at same time

Caching With Leases



Caching With Leases



Caching With Leases

Can clients cache values too?

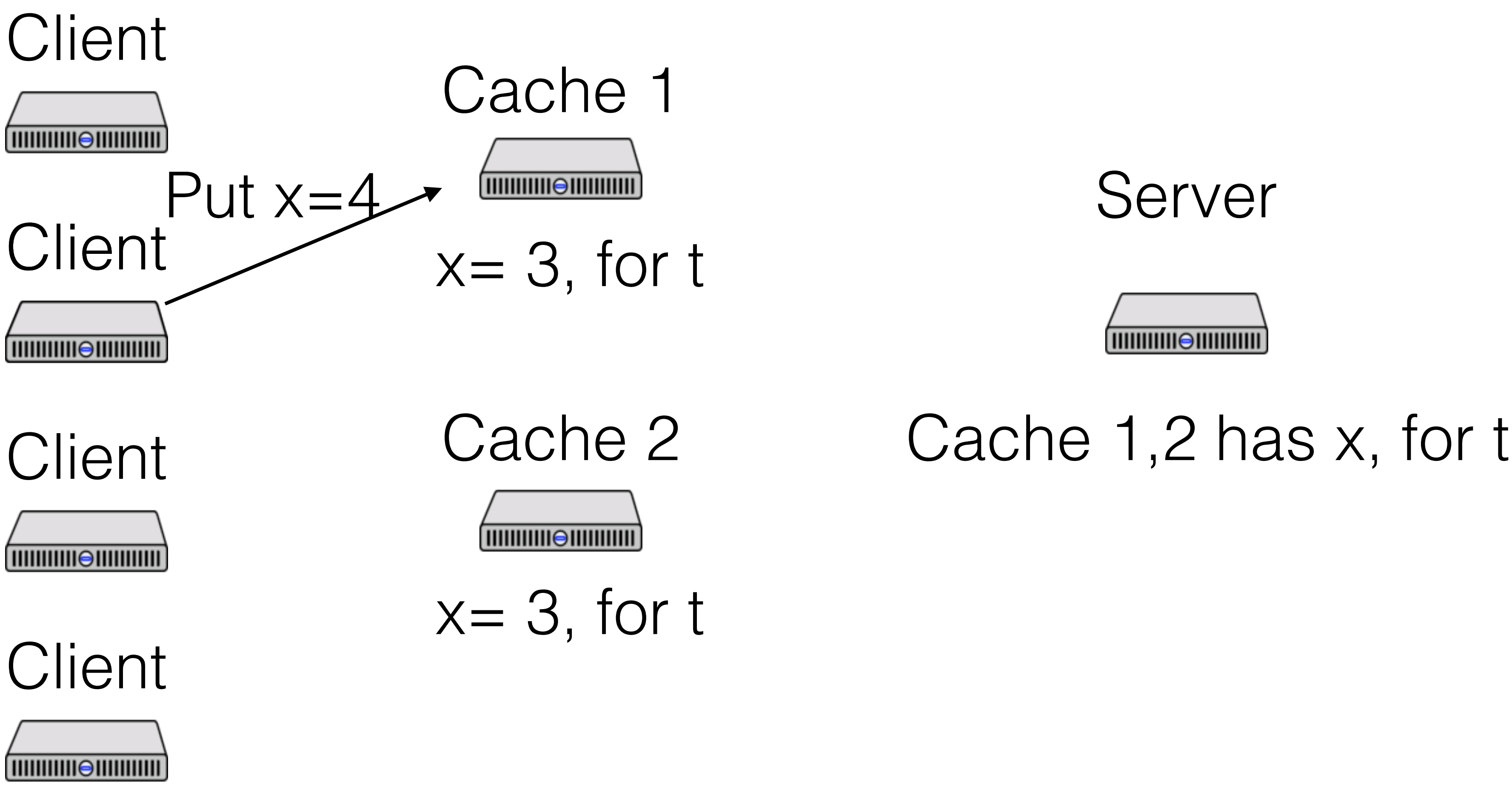
Caching With Leases

Can clients cache values too?

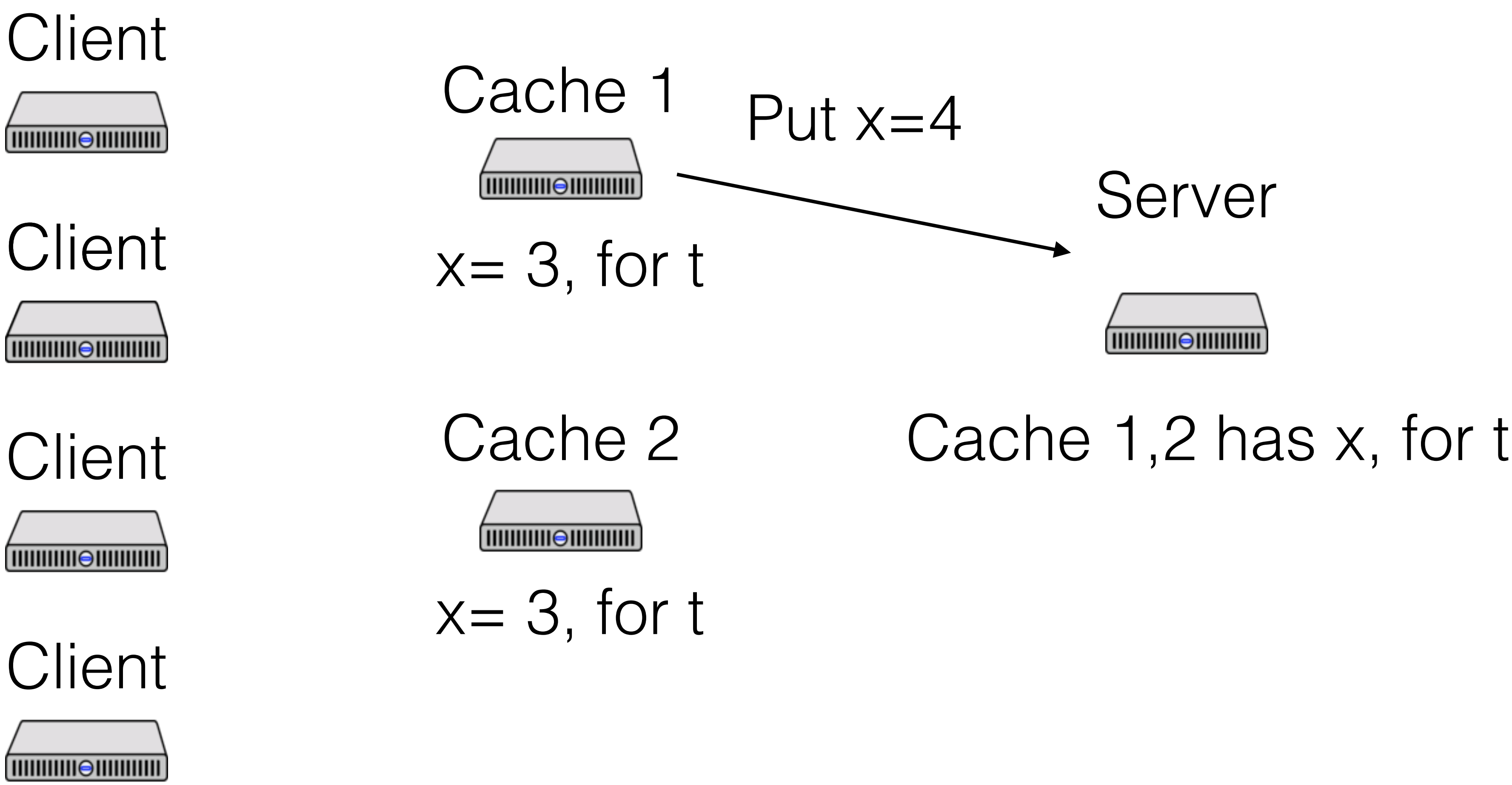
Yes! Leases can be delegated.

Caches keep track as to which clients have which data.

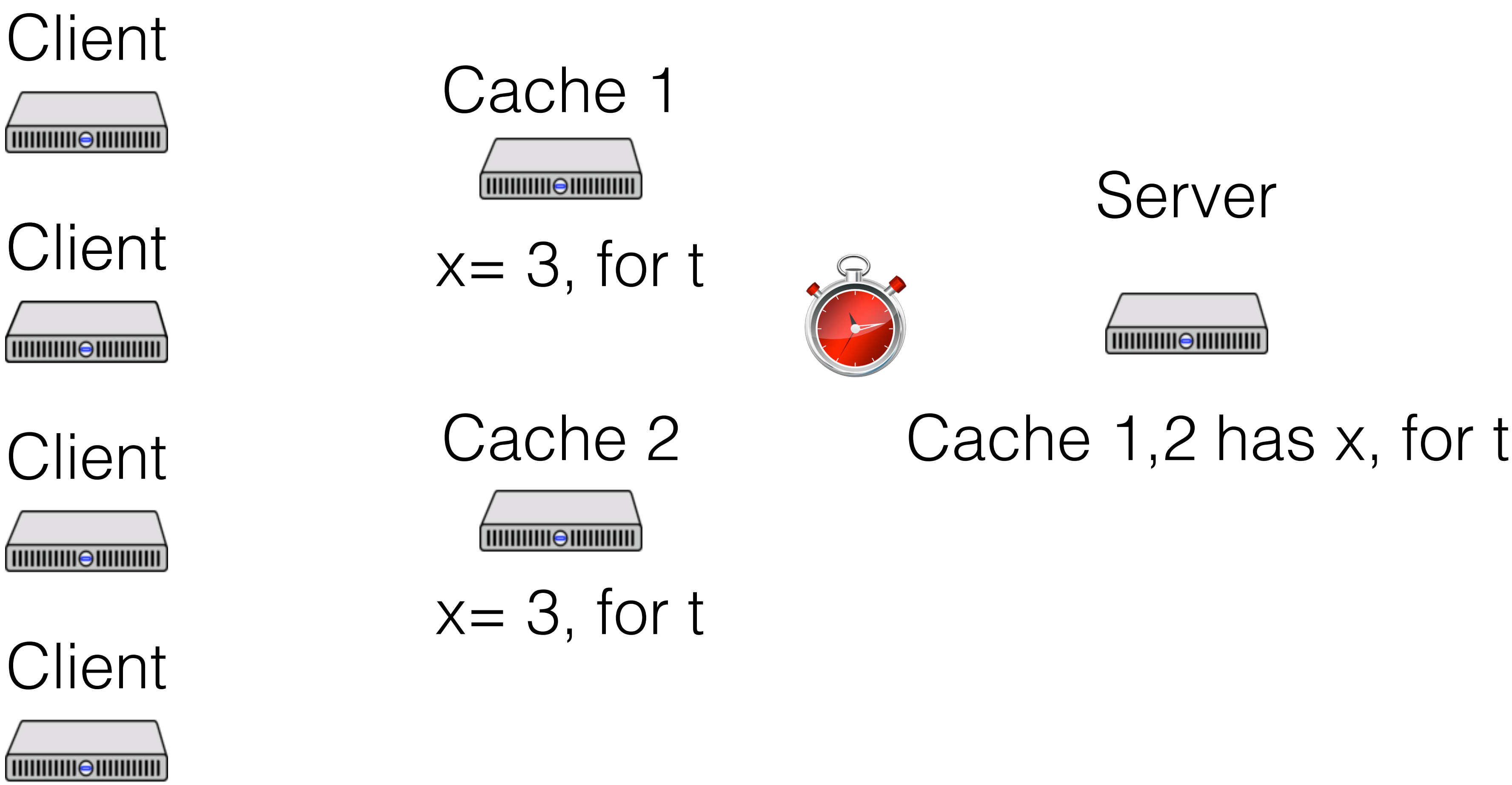
Caching With Leases



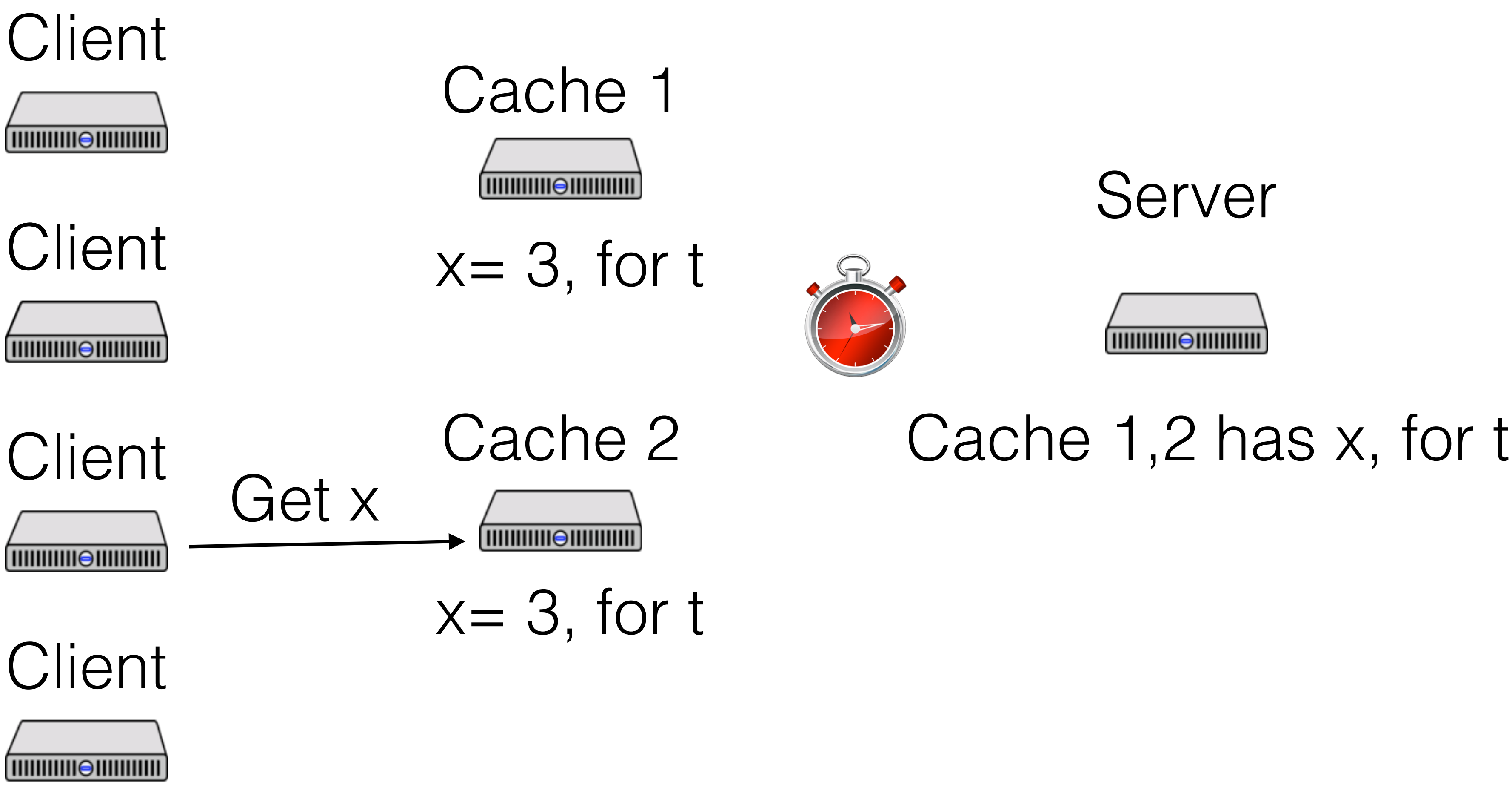
Caching With Leases



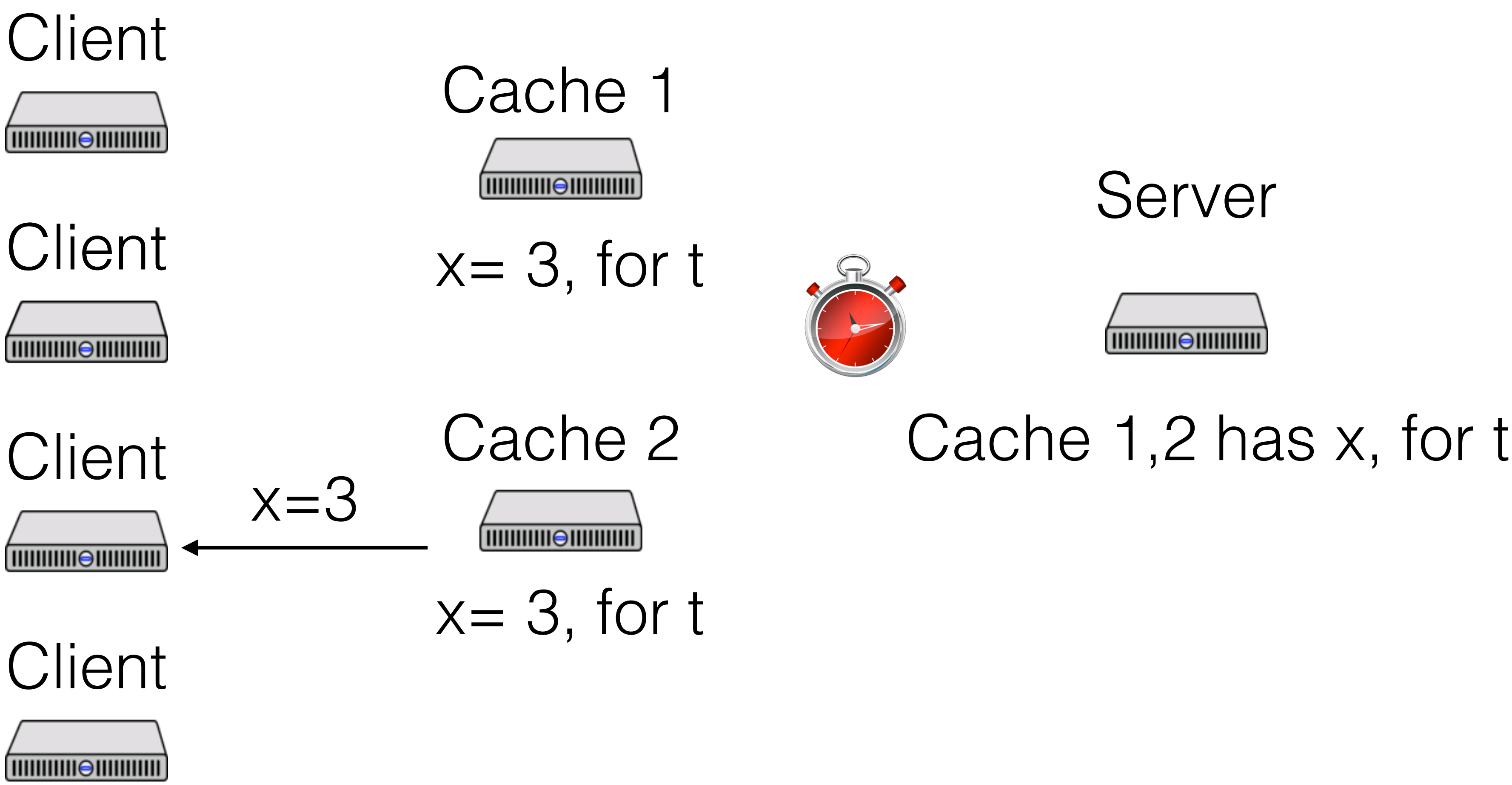
Caching With Leases



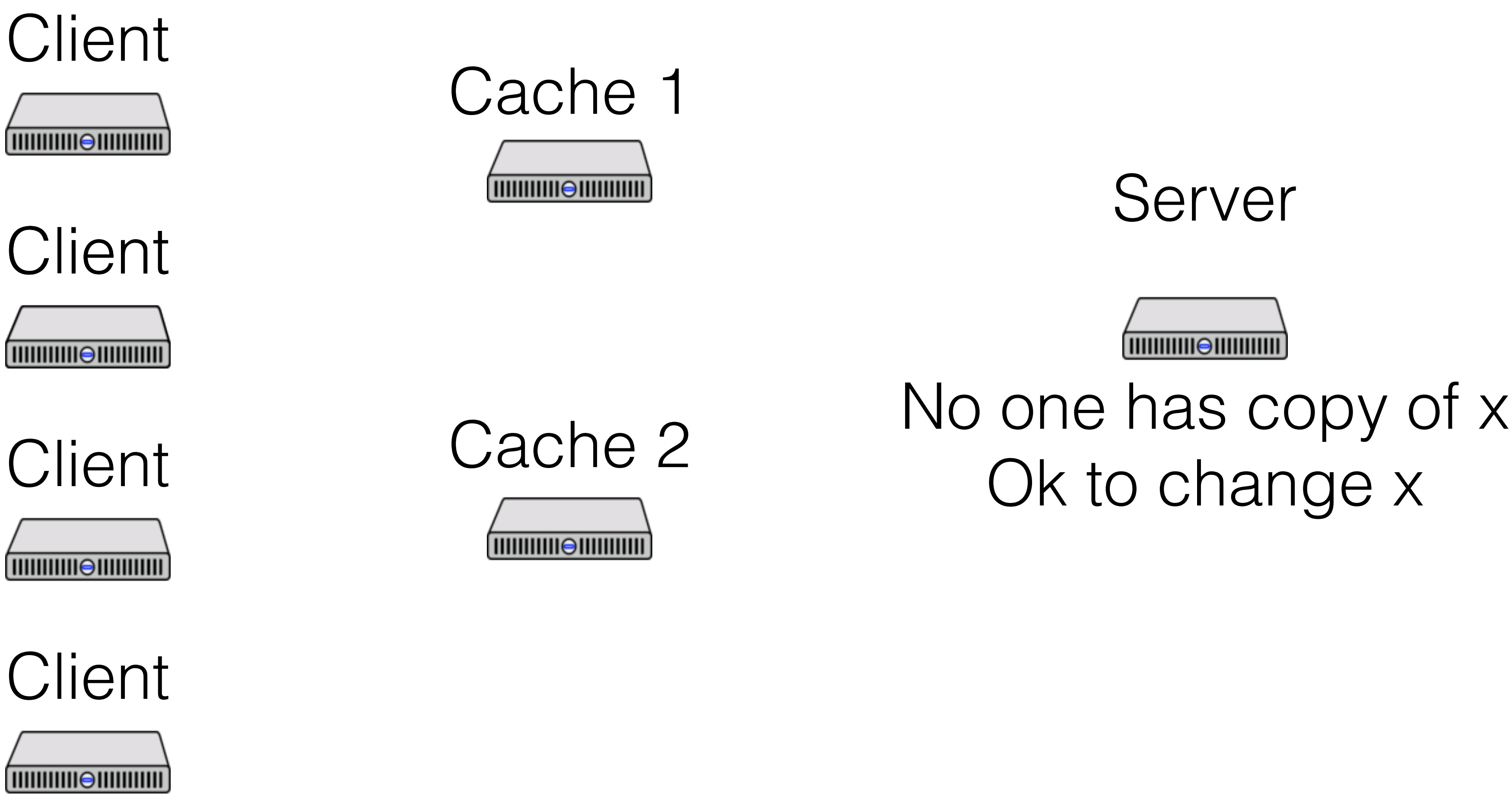
Caching With Leases



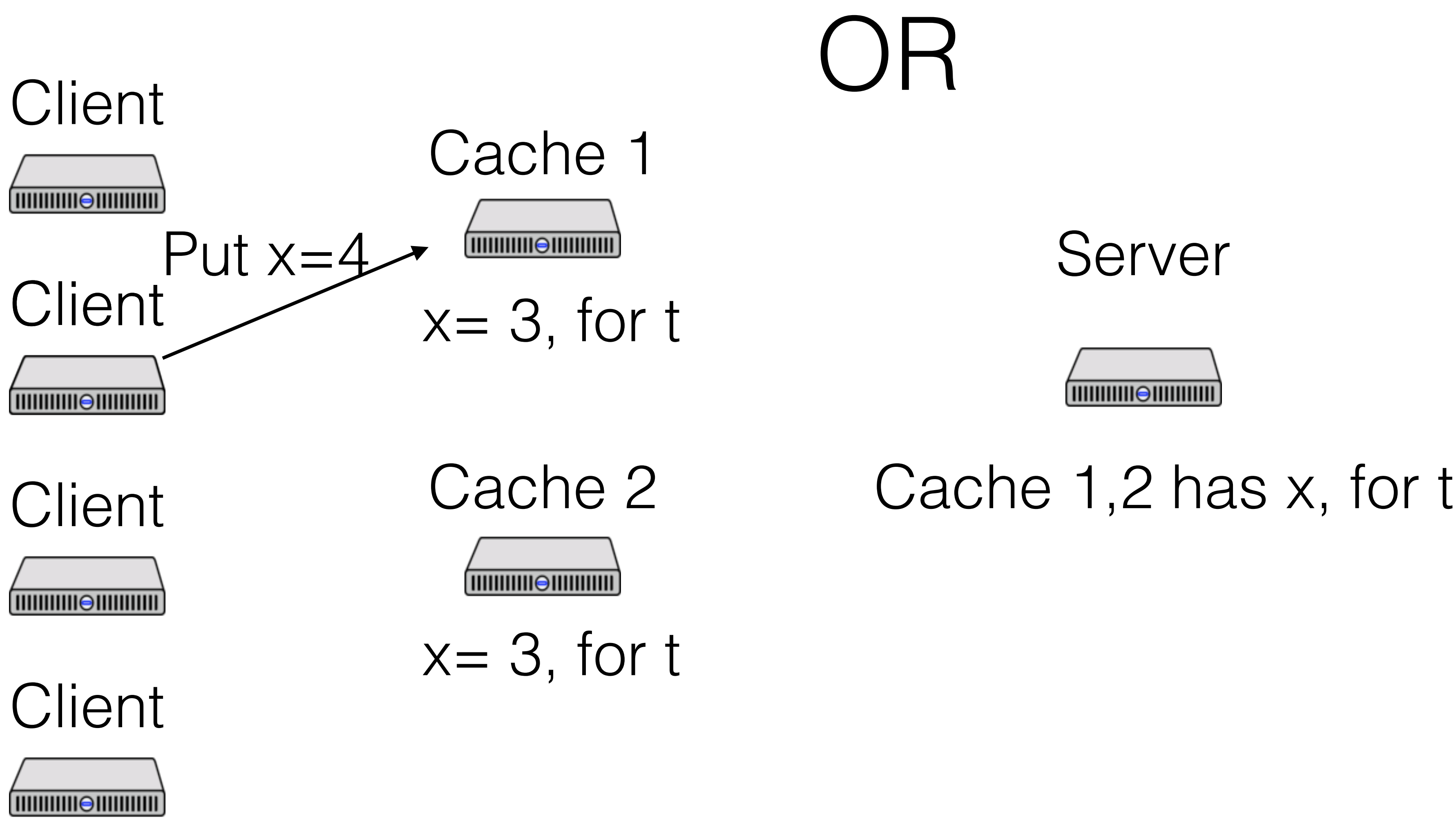
Caching With Leases



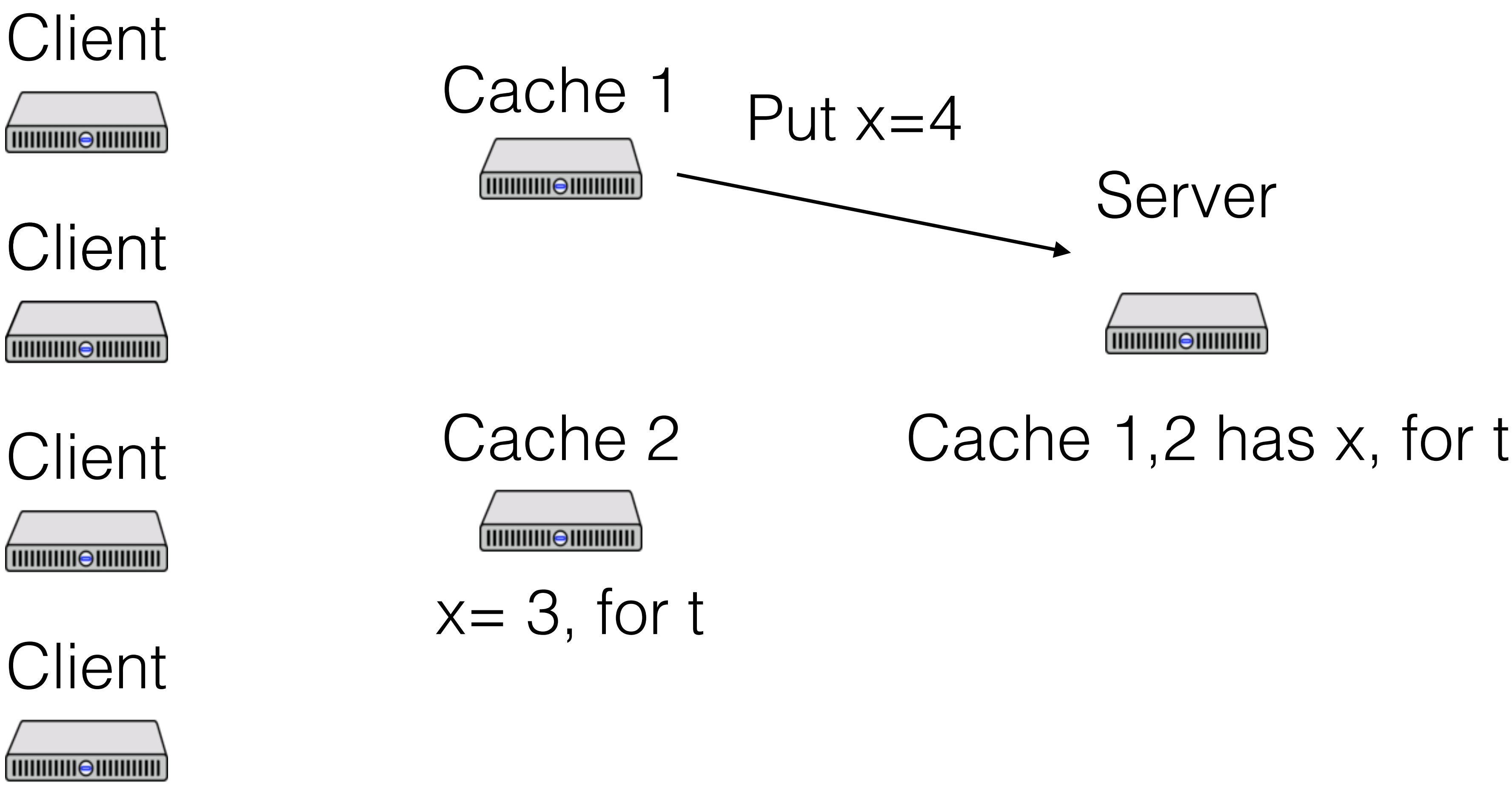
Caching With Leases



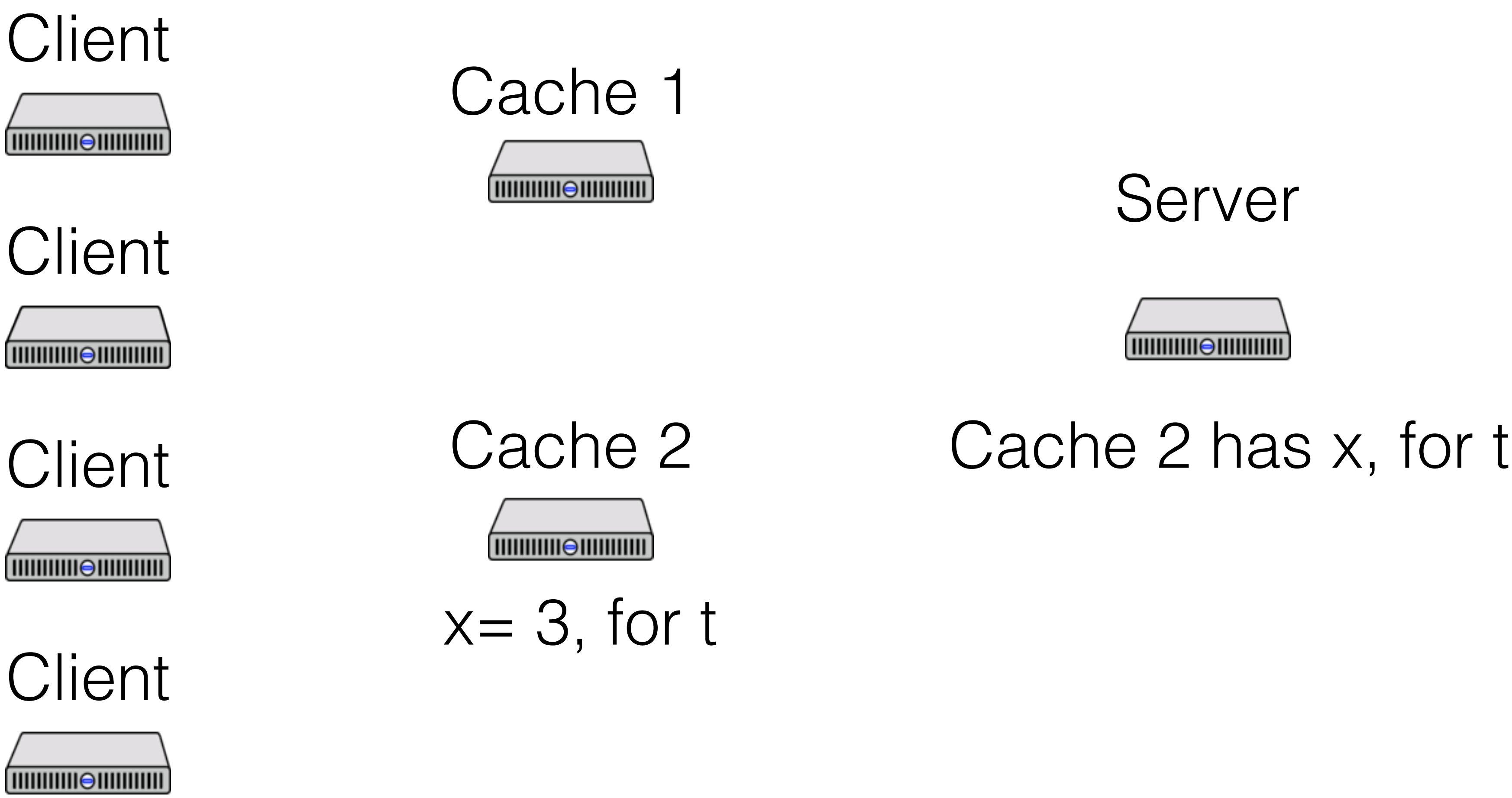
Caching With Leases



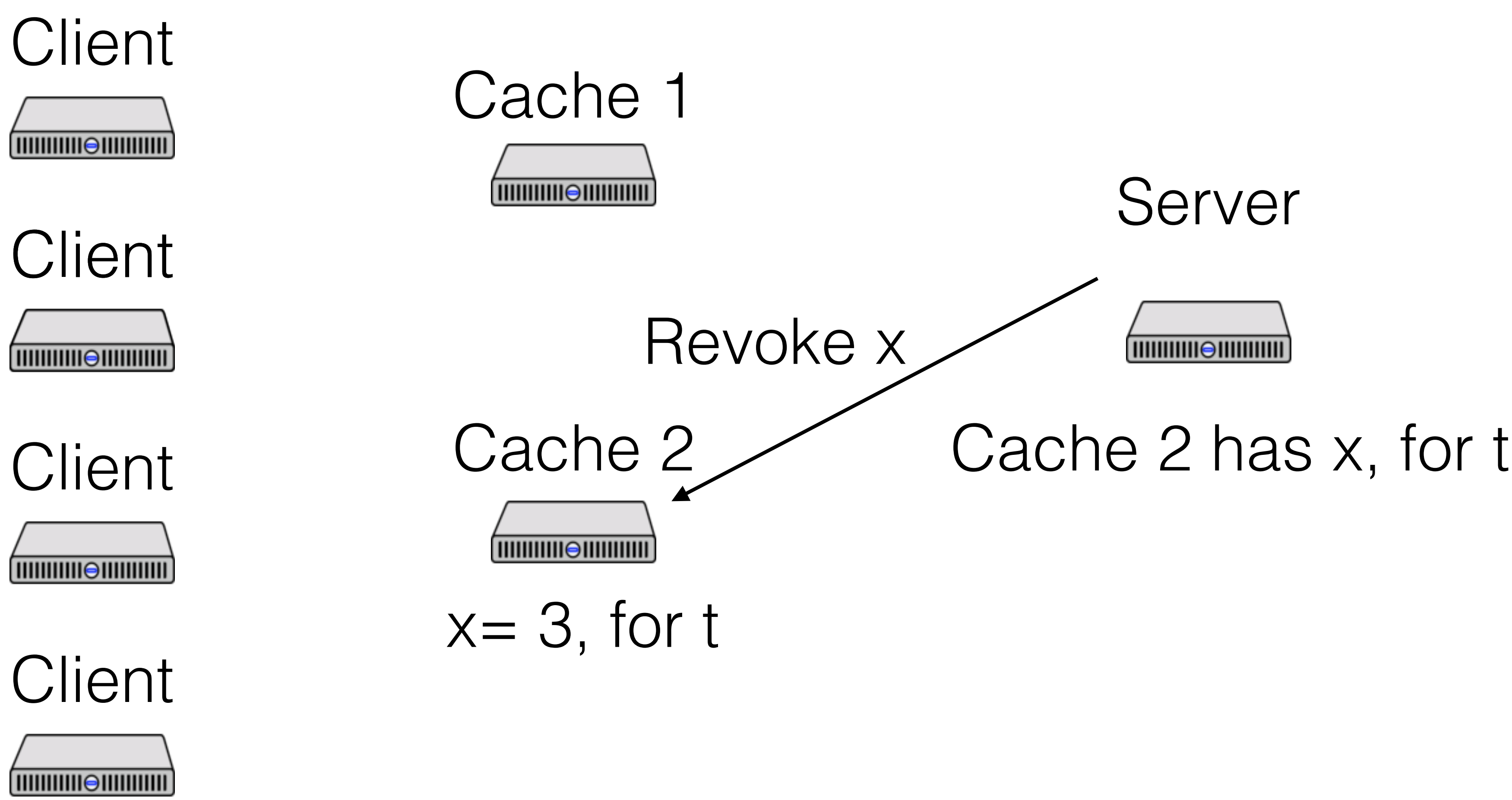
Caching With Leases



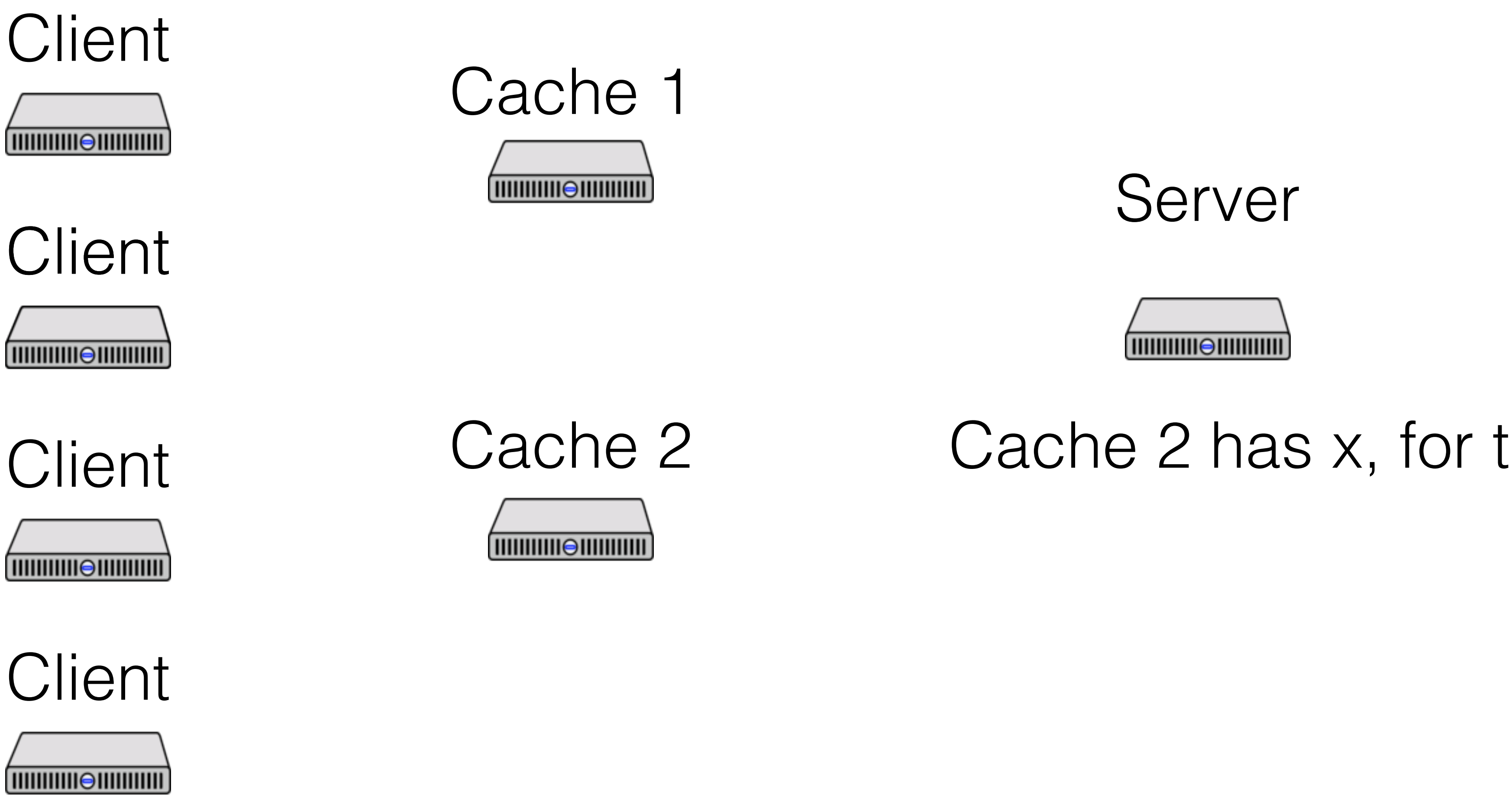
Caching With Leases



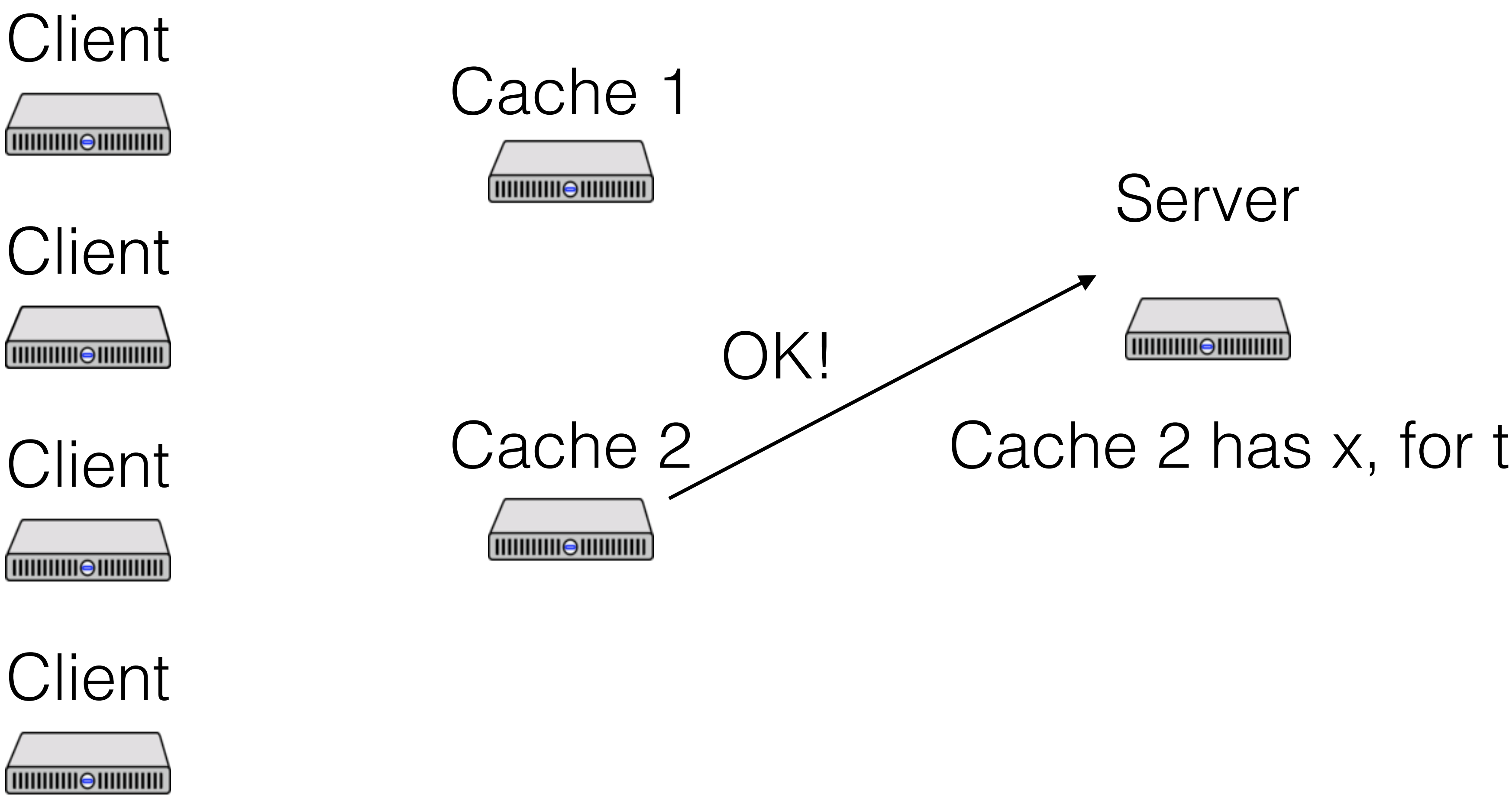
Caching With Leases



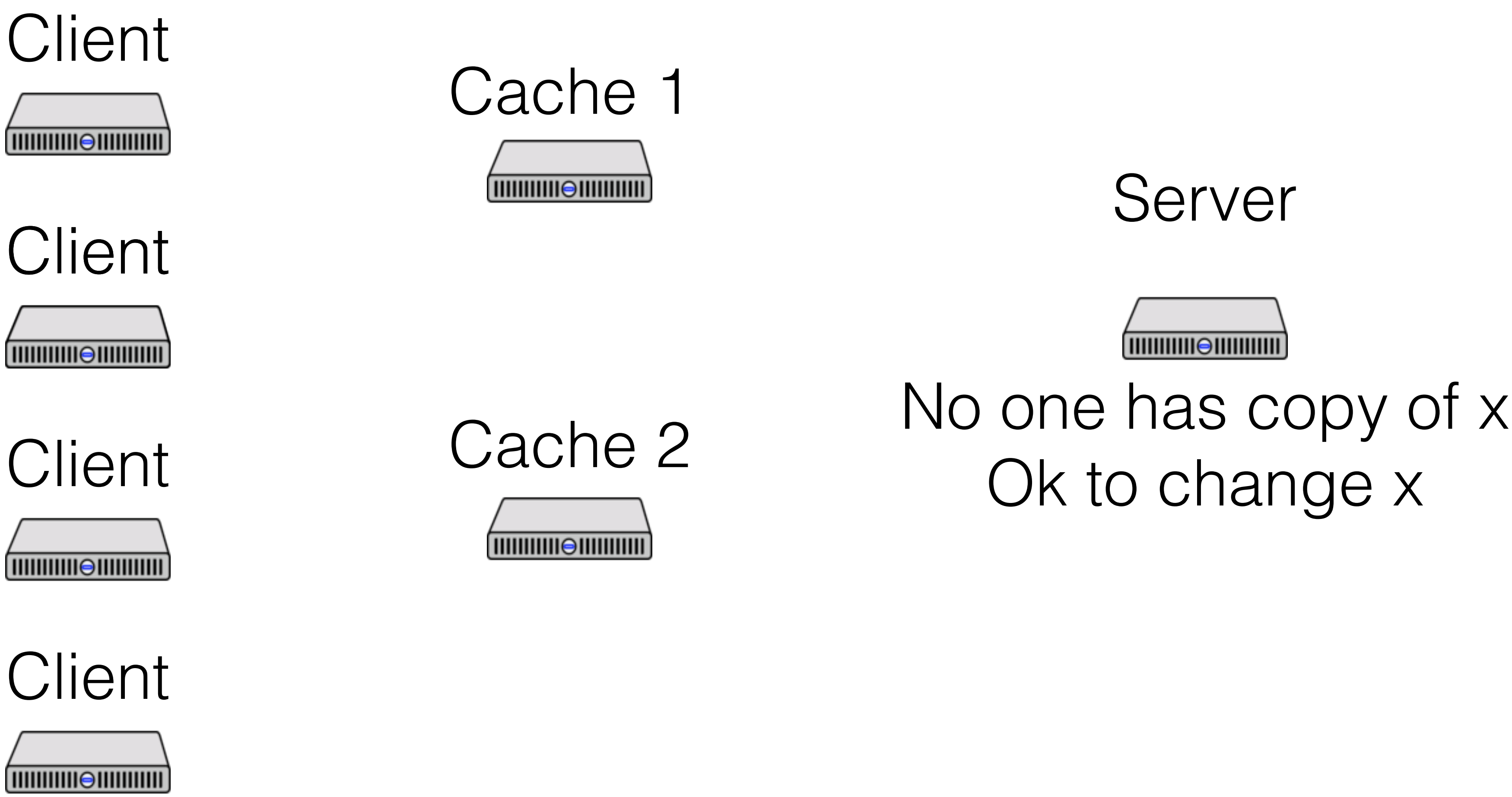
Caching With Leases



Caching With Leases



Caching With Leases



Caching With Leases

Why can’t we leave the old value on cache 1 while we shoot down other copies?

Why can’t we just update the old value on cache 1 and then shoot down the other copies?

Caching With Leases

Why can't we leave the old value on cache 1 while we shoot down other copies?

Why can't we just update the old value on cache 1 and then shoot down the other copies?

Linearizability: as if there is only one copy

- implement by having only one copy for updates
- many copies ok when no one is updating

Caching with Invalidation

Cache obtains lease (read-only)

No one can modify data until lease expires or is revoked

Server gets update

Forwards invalidation (revoke) to every node with copy

Wait for response from all (or timeout)

OK to proceed with change

Terminology

Cache coherence: keeping caches up to date

- can be linearizable, or weaker semantics

Write through: caches hold read-only data

- write sent to store, store revokes copies

Write back: caches can hold read-only or modified data

- write to cache, cache asks store to revoke
- subsequent writes faster

MSI

Three cache states:

- **M**odified: this is the only copy, it's dirty
- **S**hared: this is one of many copies, it's clean
- **I**nvalid

Allowed states between pairs of caches:

	M	S	I
M			✓
S		✓	✓
I	✓	✓	✓

Write Back Fault Tolerance?

Write back: caches can hold modified data

What happens when cache fails? Lose data?

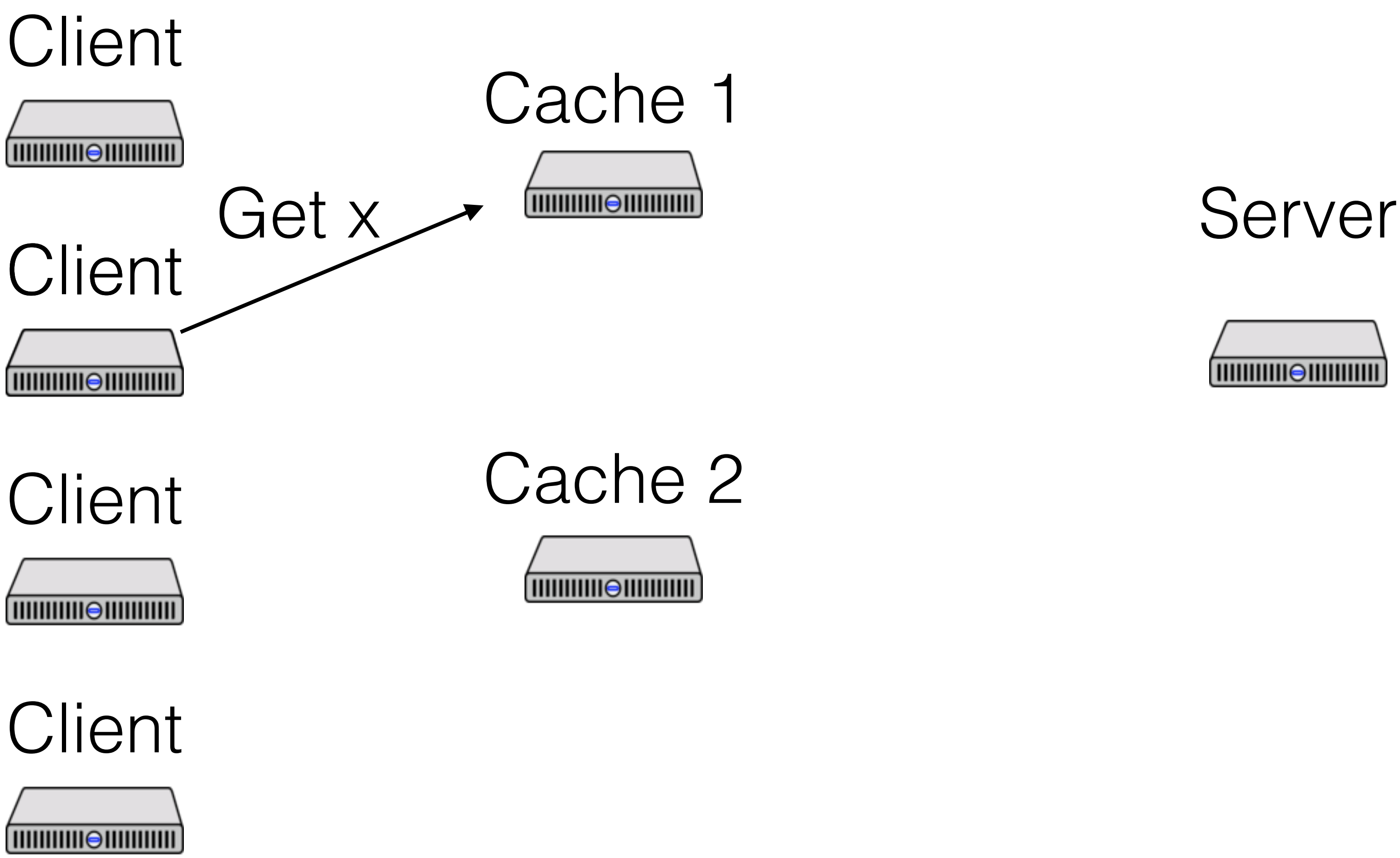
Option 1: checkpoint/restart if any cache fails

- appropriate for background computations
- CPU cache coherence is write-back

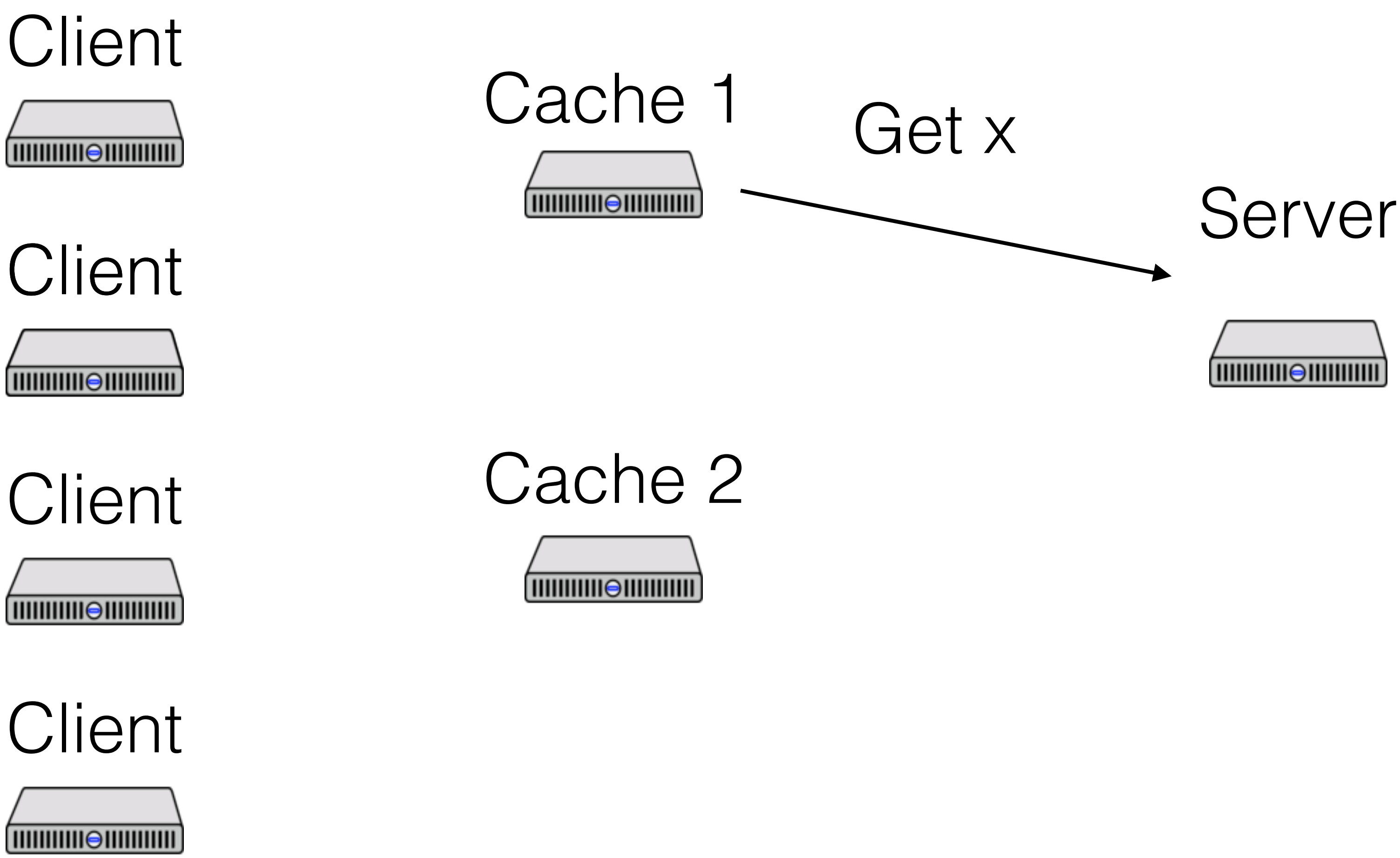
Option 2: log local changes to replicas

- identical to lease to a primary (primary logs changes), except fine-grained leases

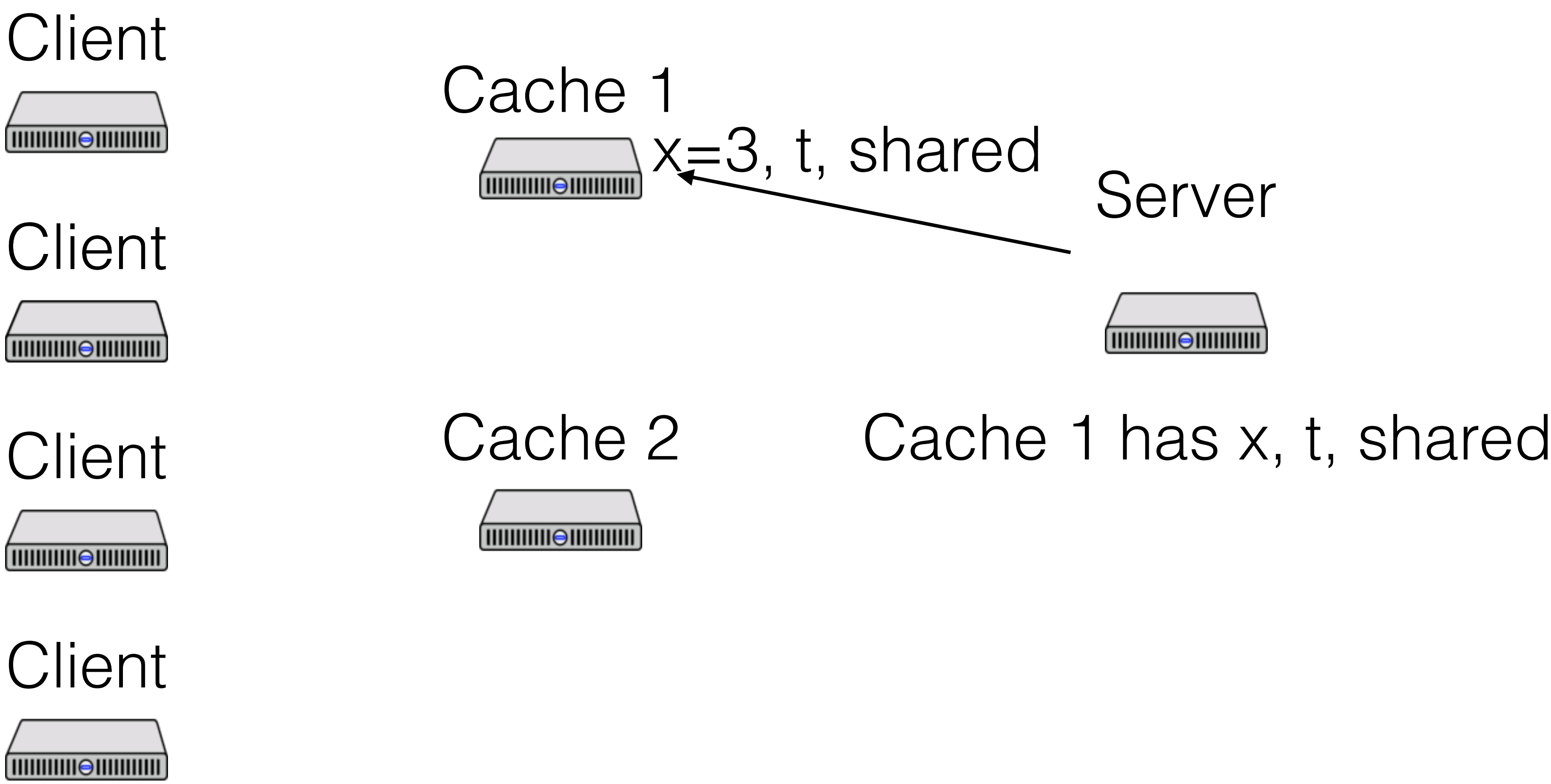
Caching With Leases



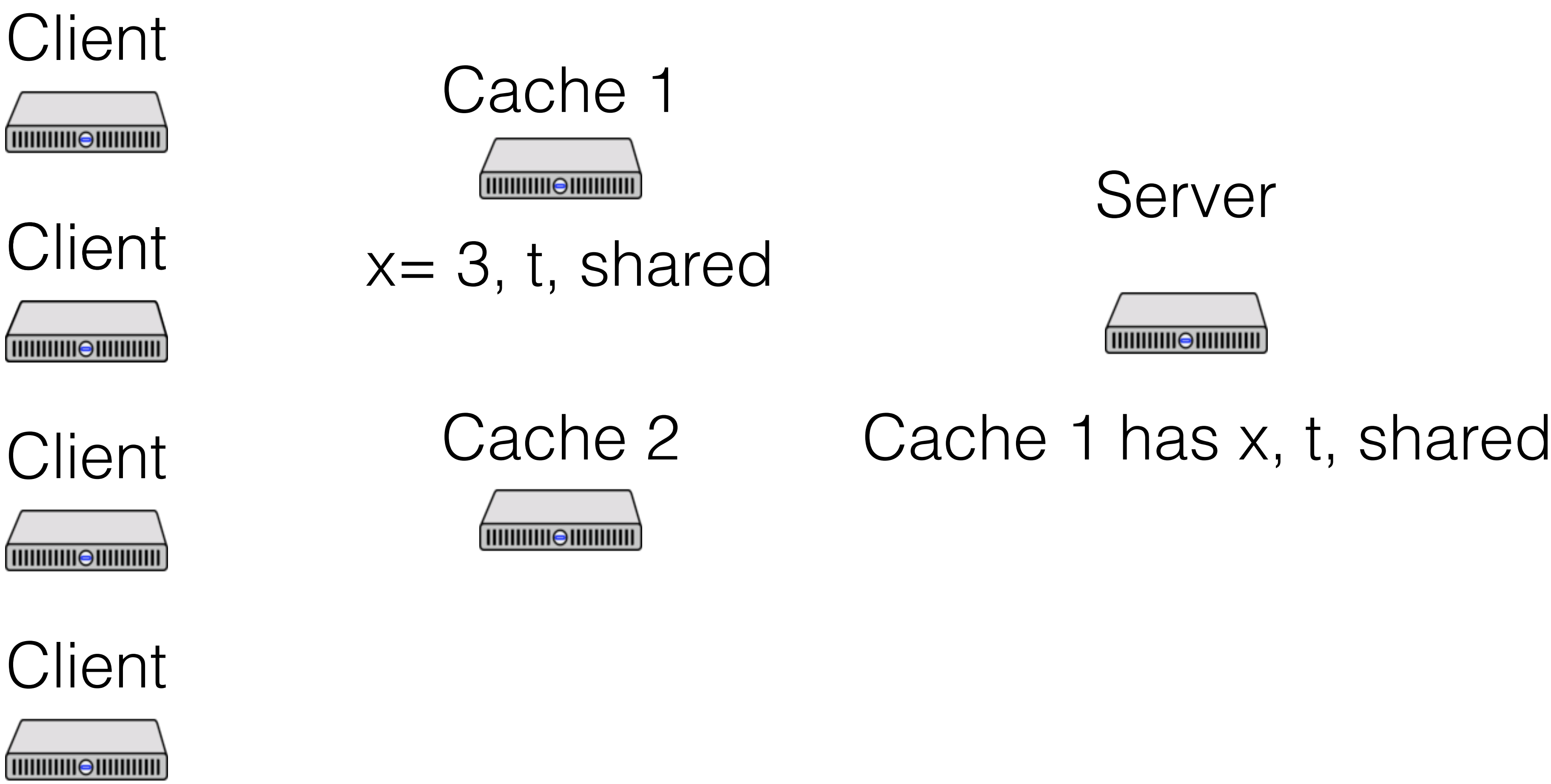
Caching With Leases



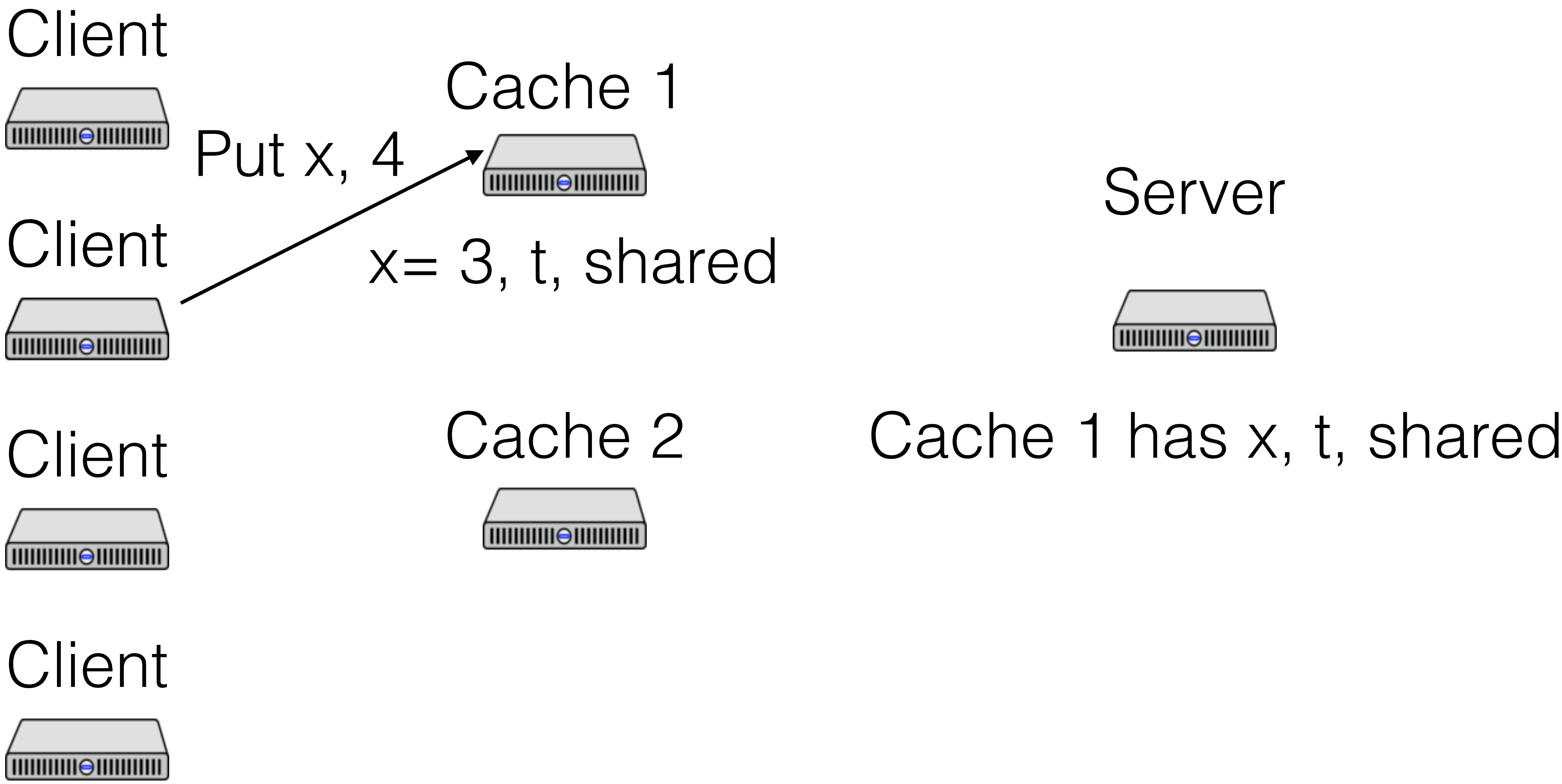
Caching With Leases



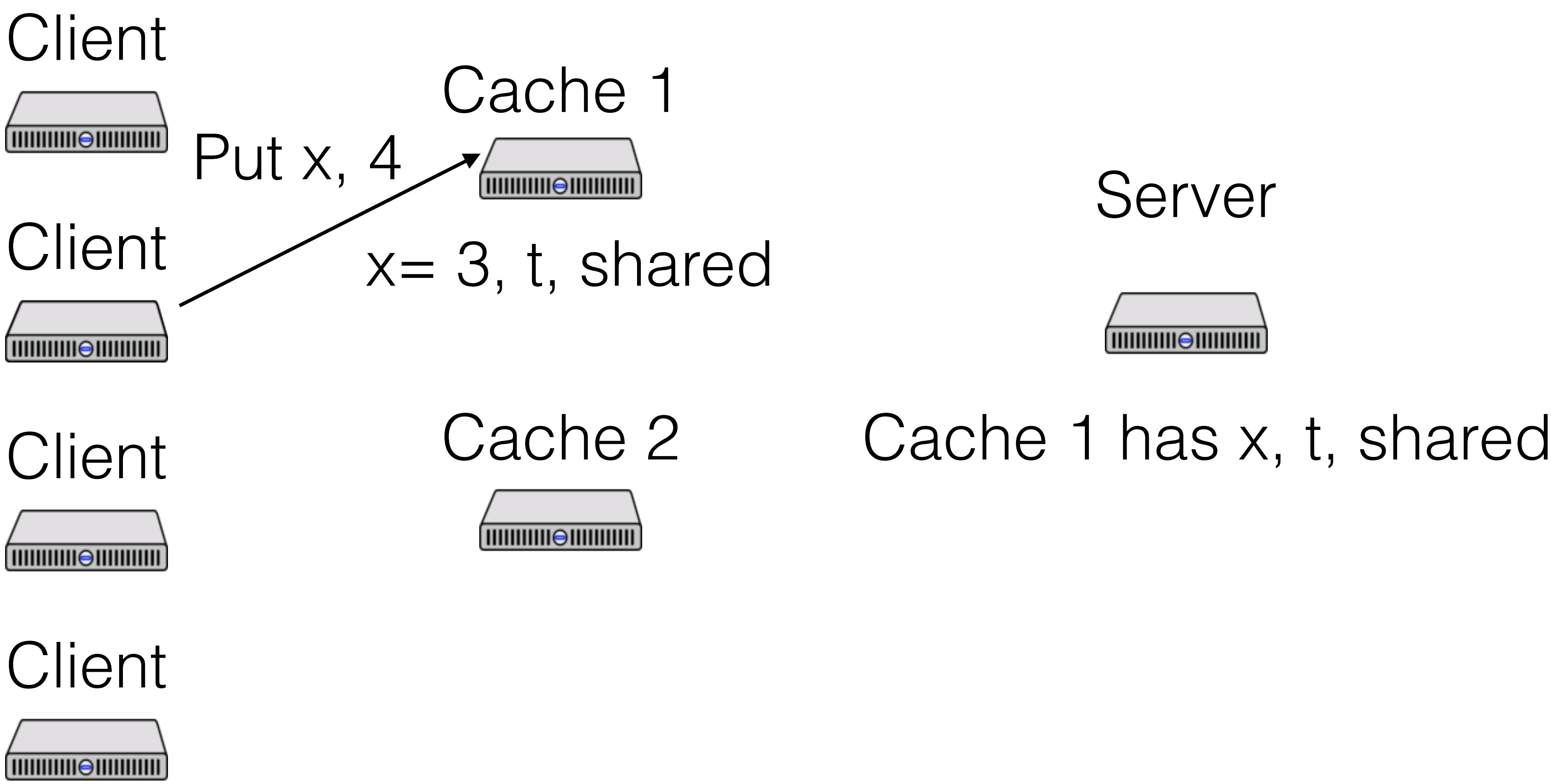
Caching With Leases



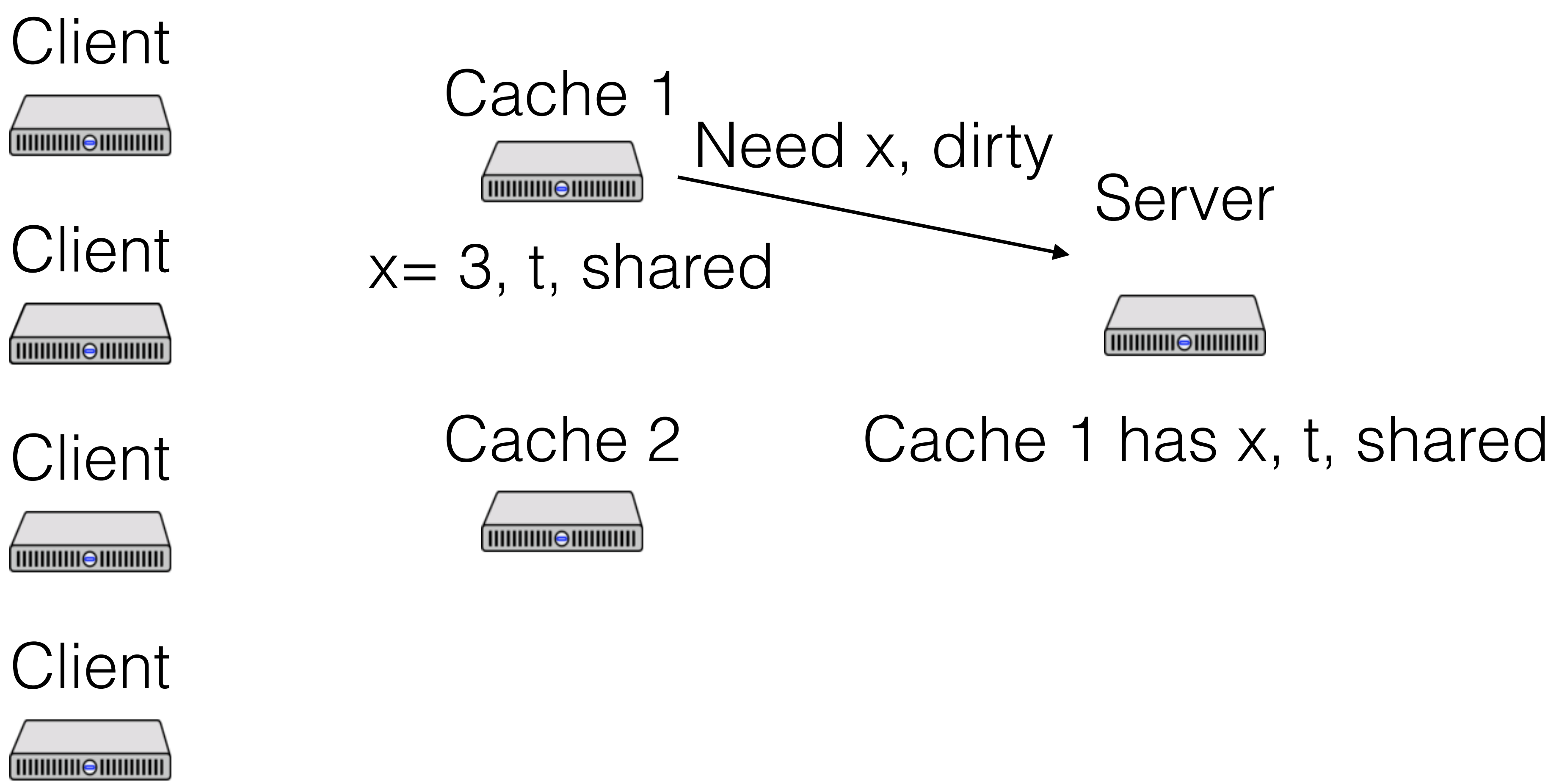
Caching With Leases



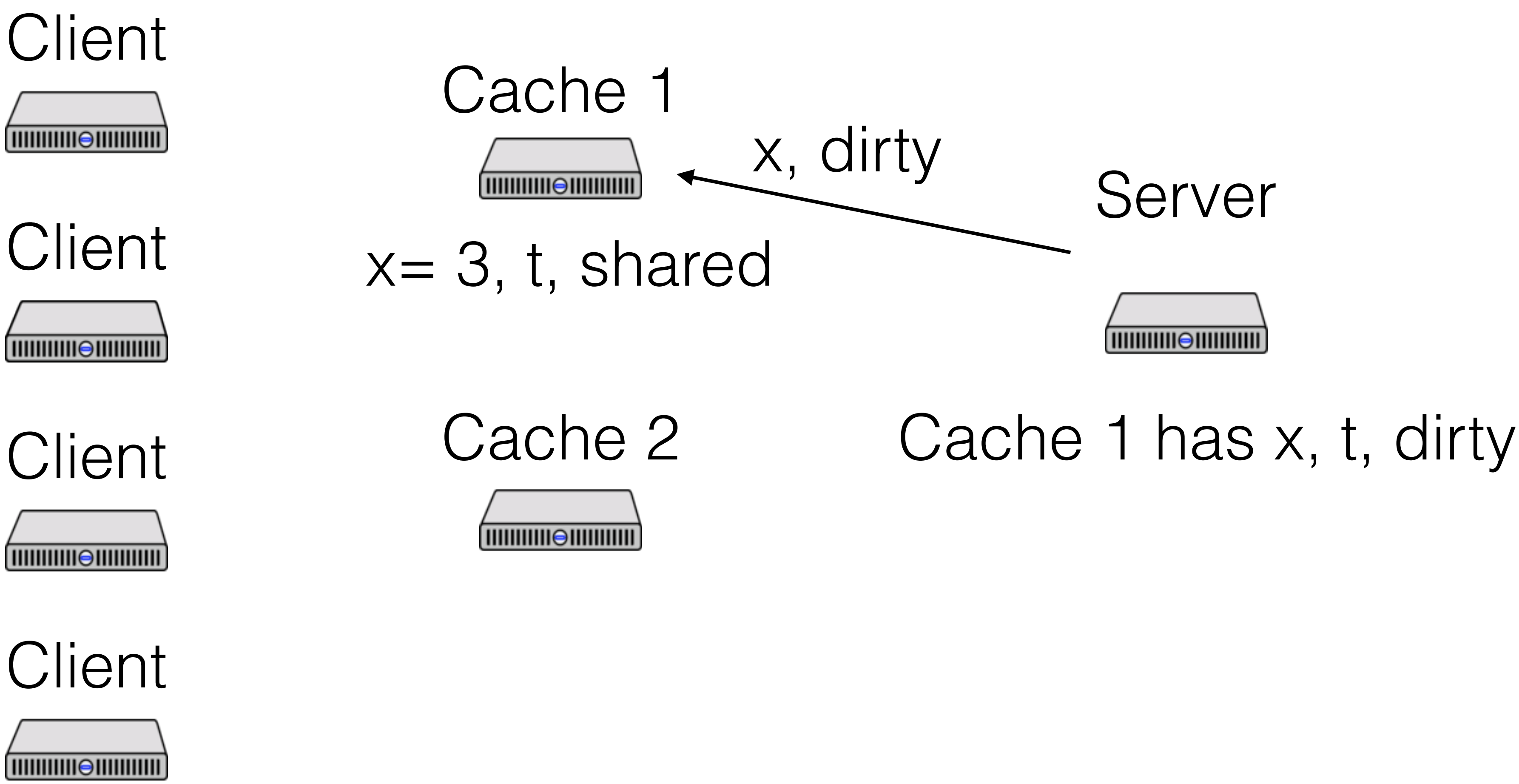
Caching With Leases



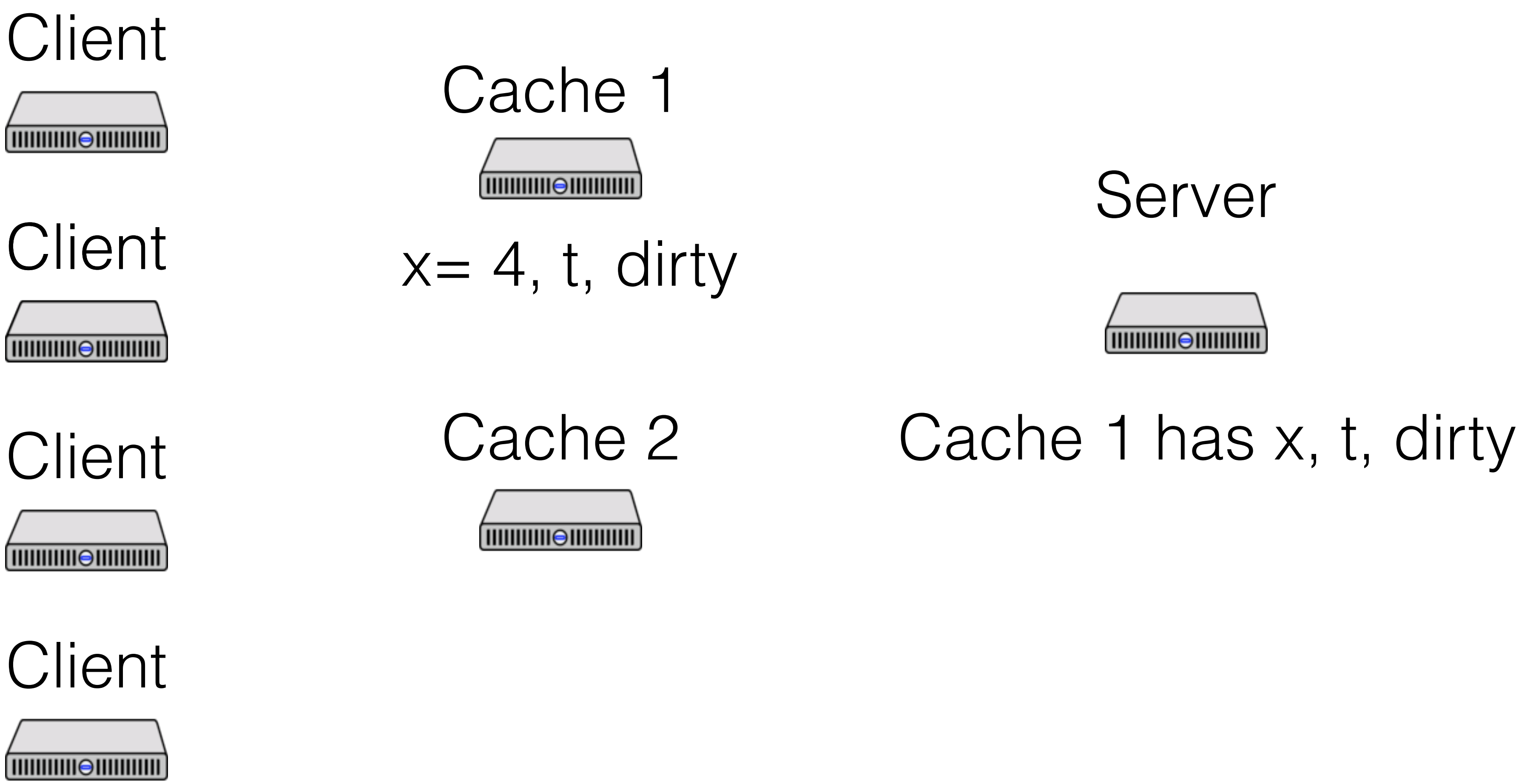
Caching With Leases



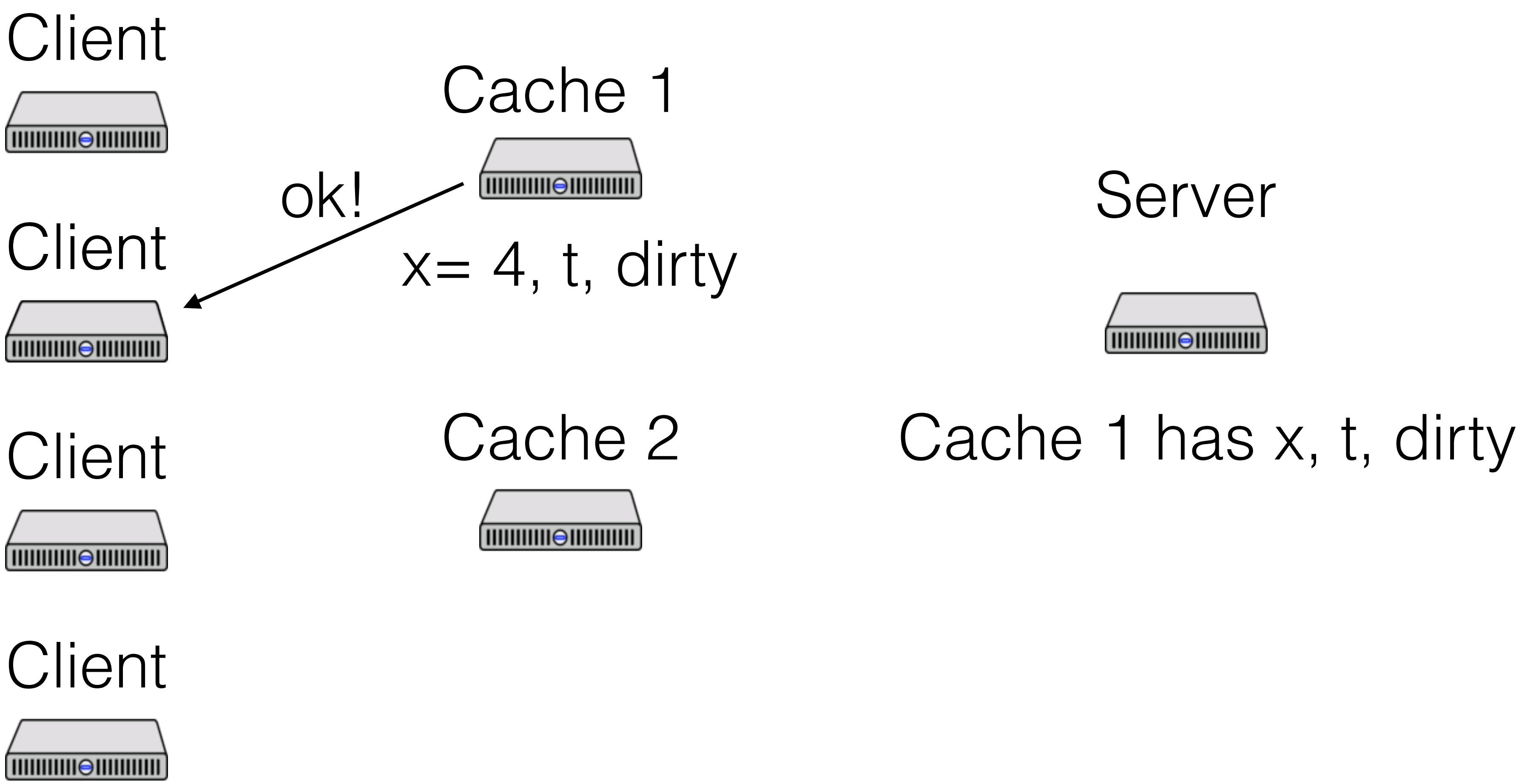
Caching With Leases



Caching With Leases



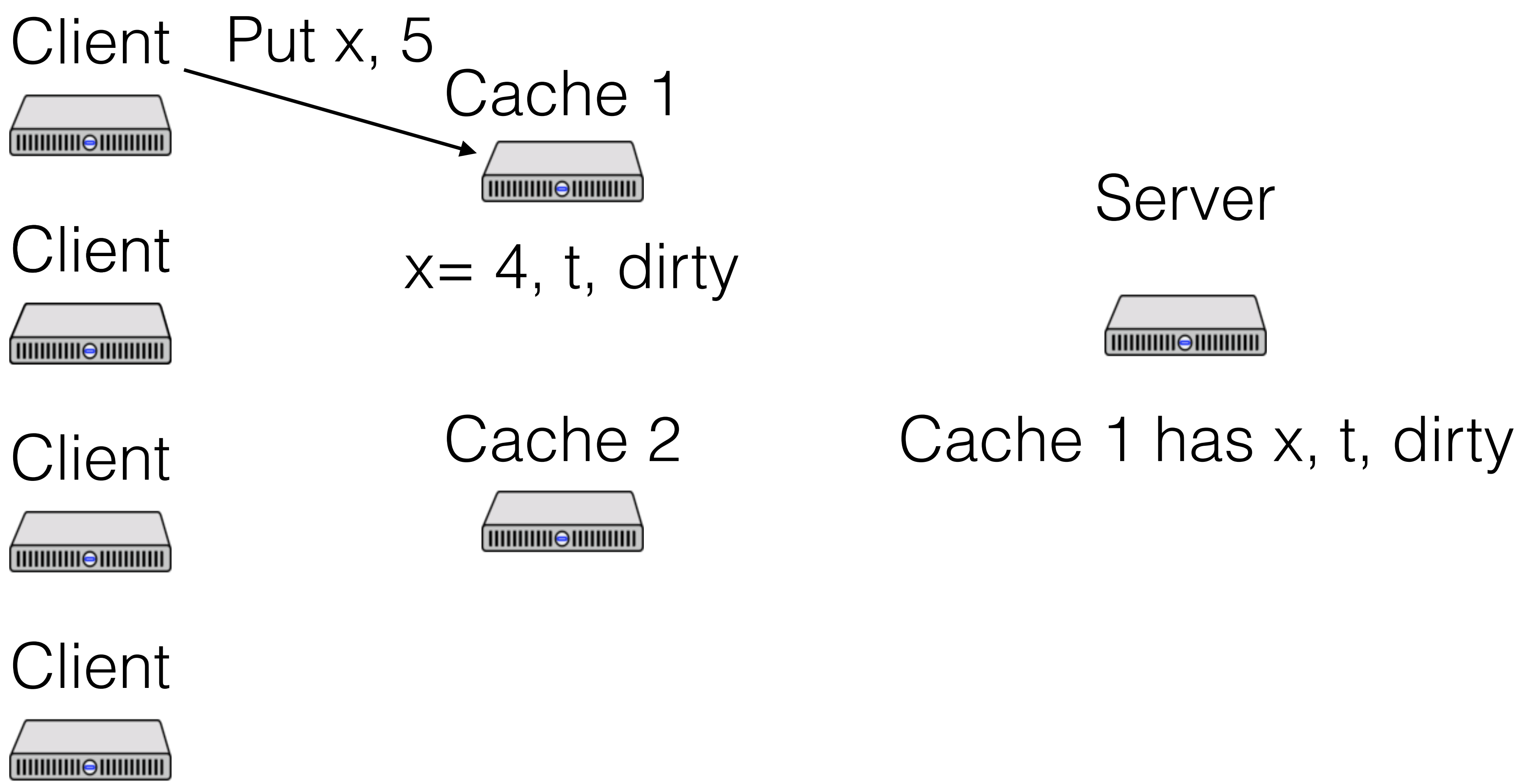
Caching With Leases



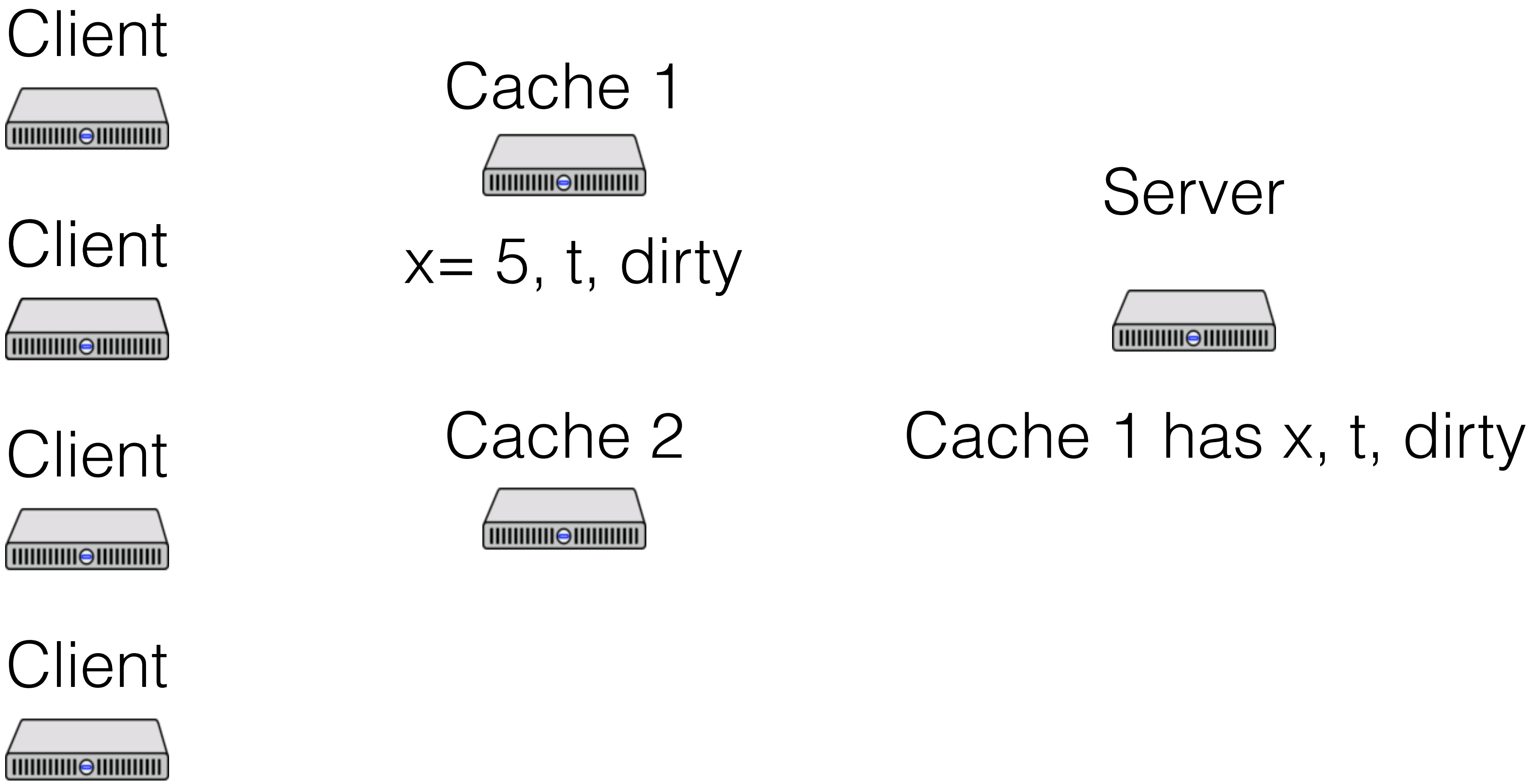
Caching With Leases

Why does cache 1 wait until other copies are revoked and write is applied before returning ok to client?

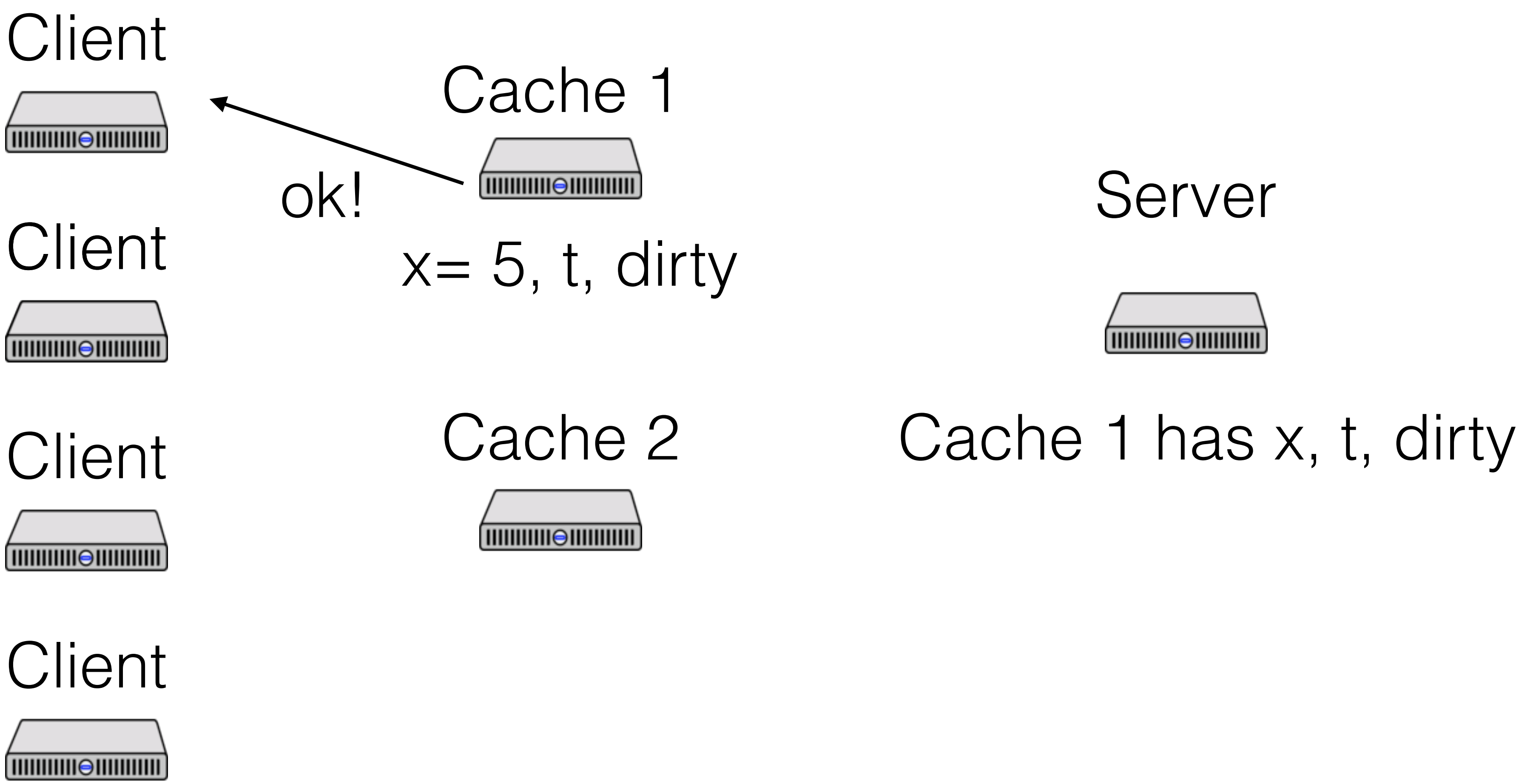
Caching With Leases



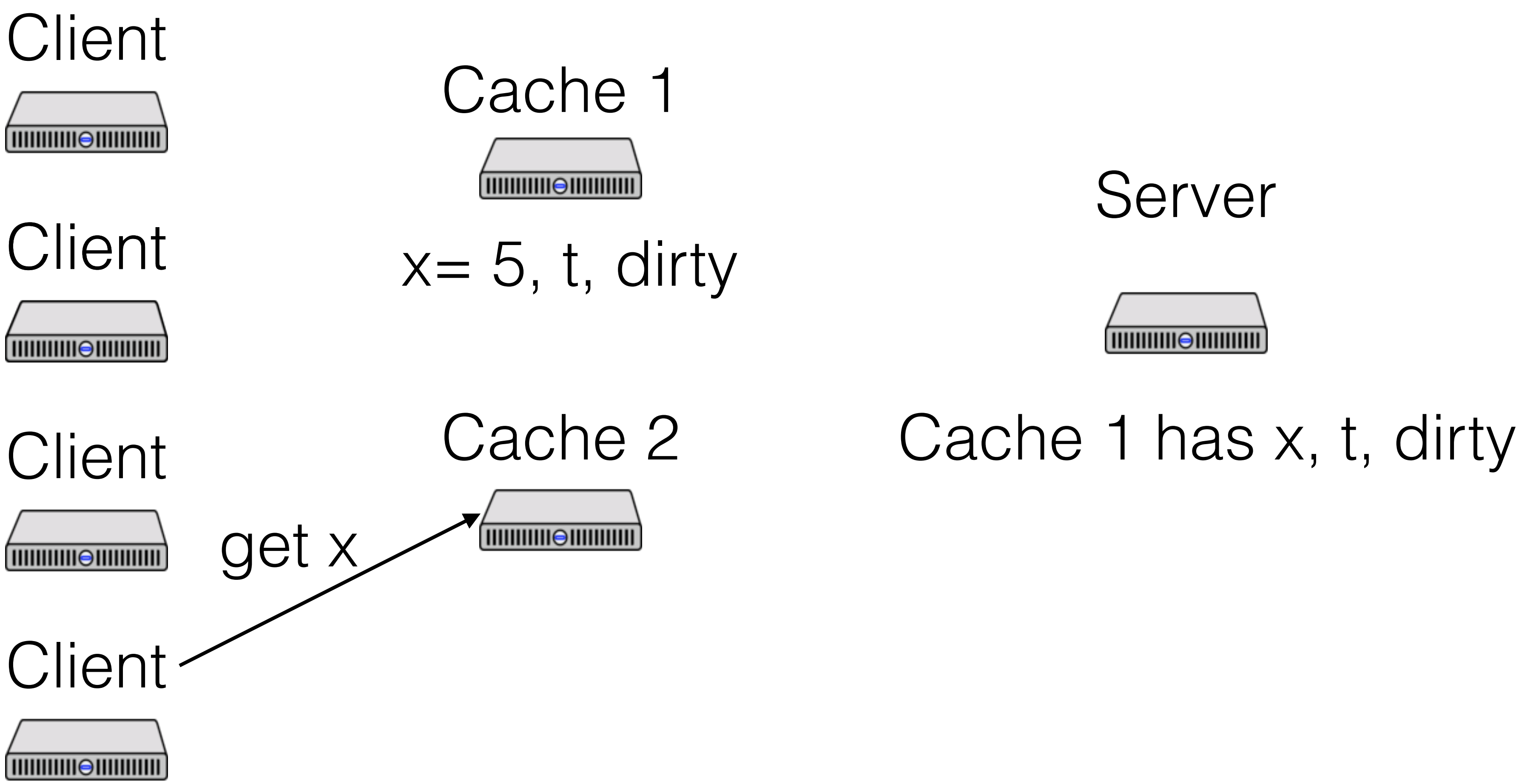
Caching With Leases



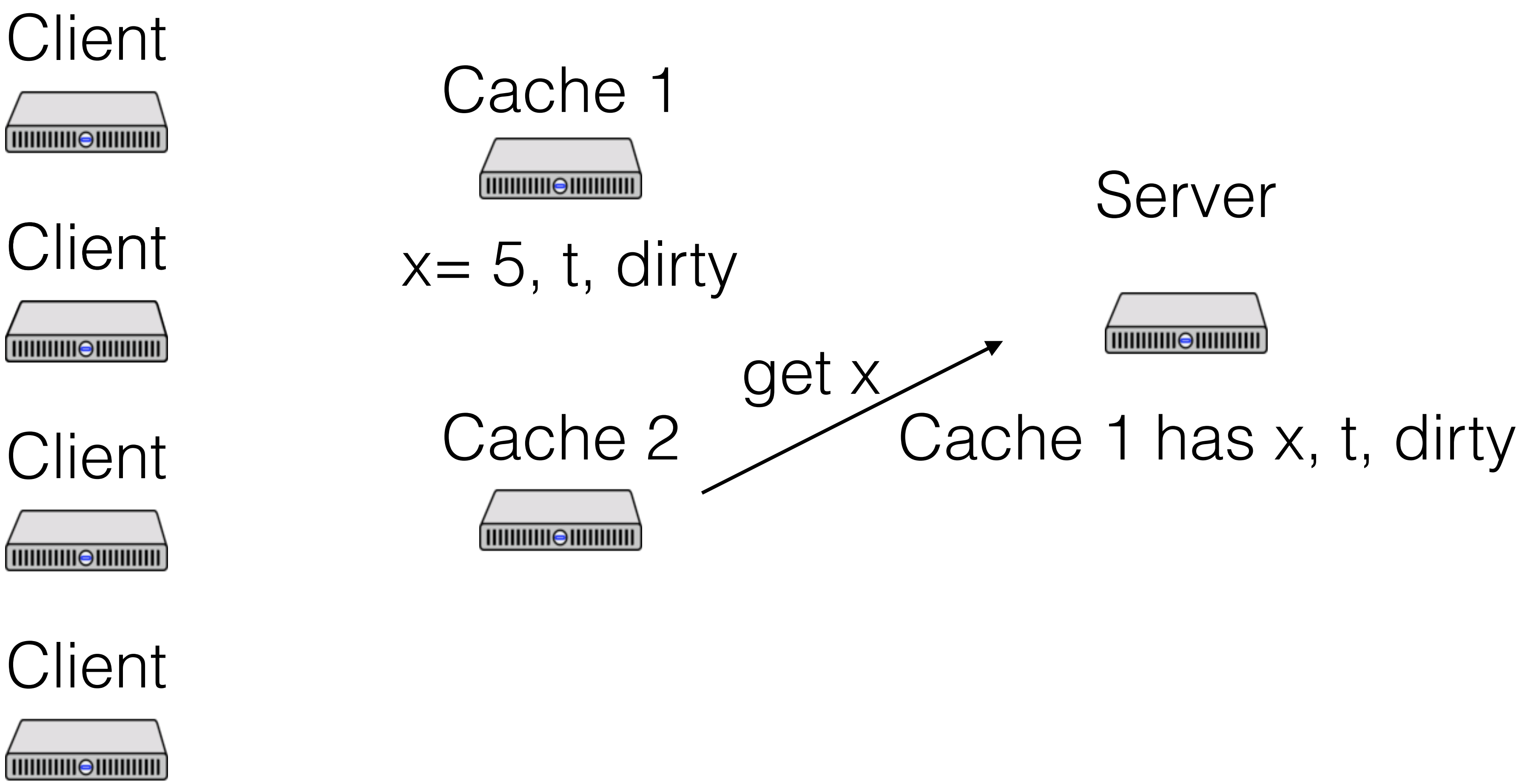
Caching With Leases



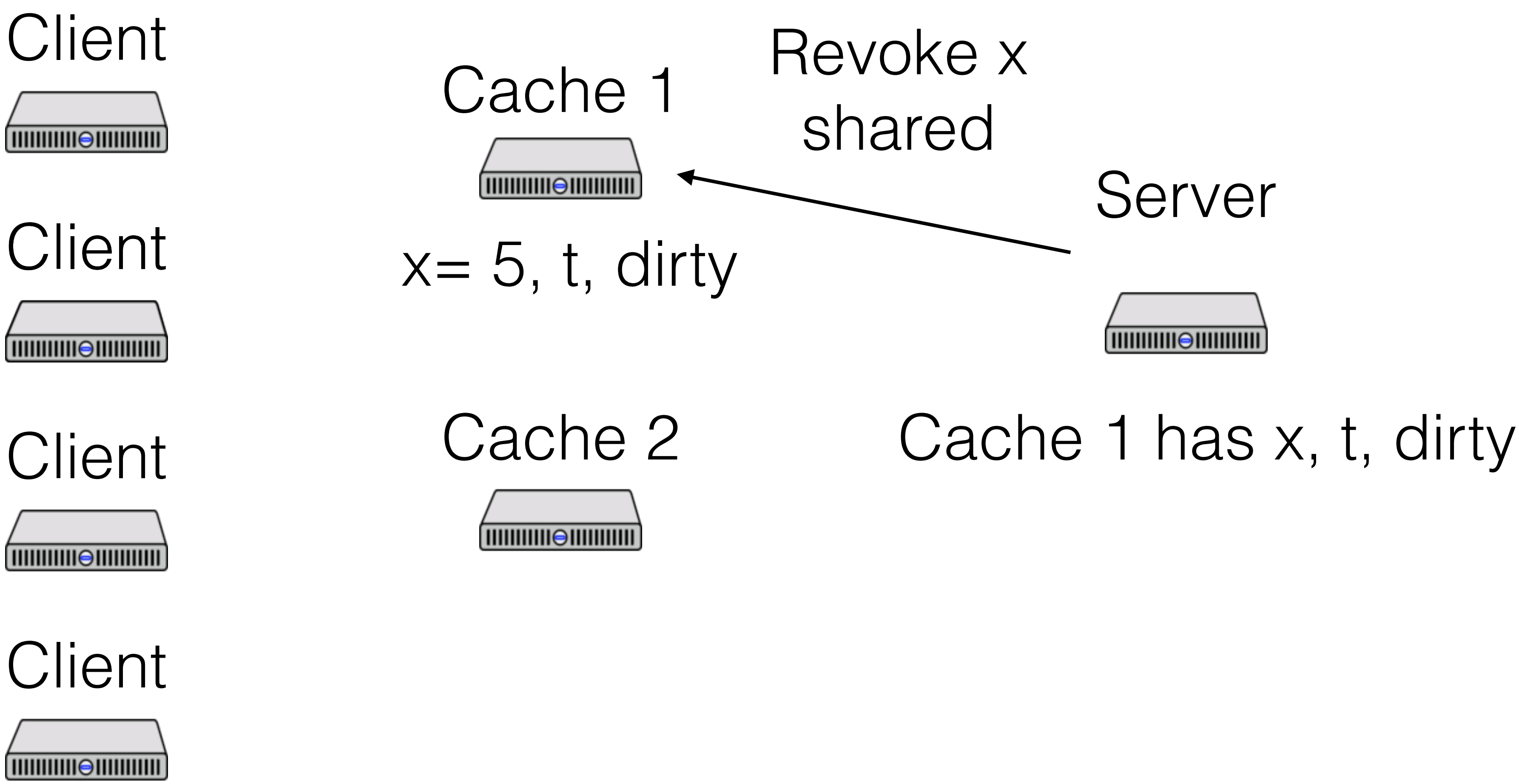
Caching With Leases



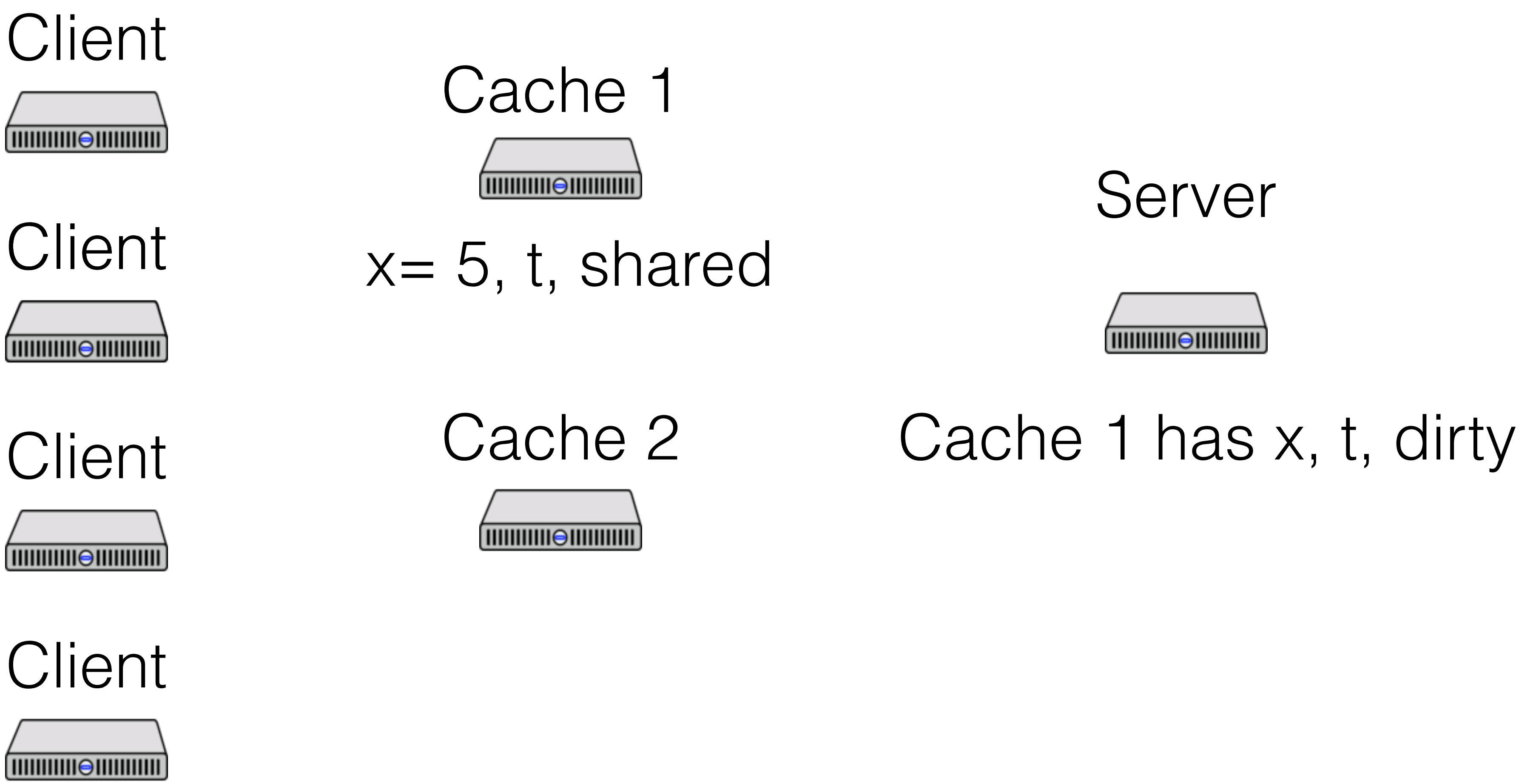
Caching With Leases



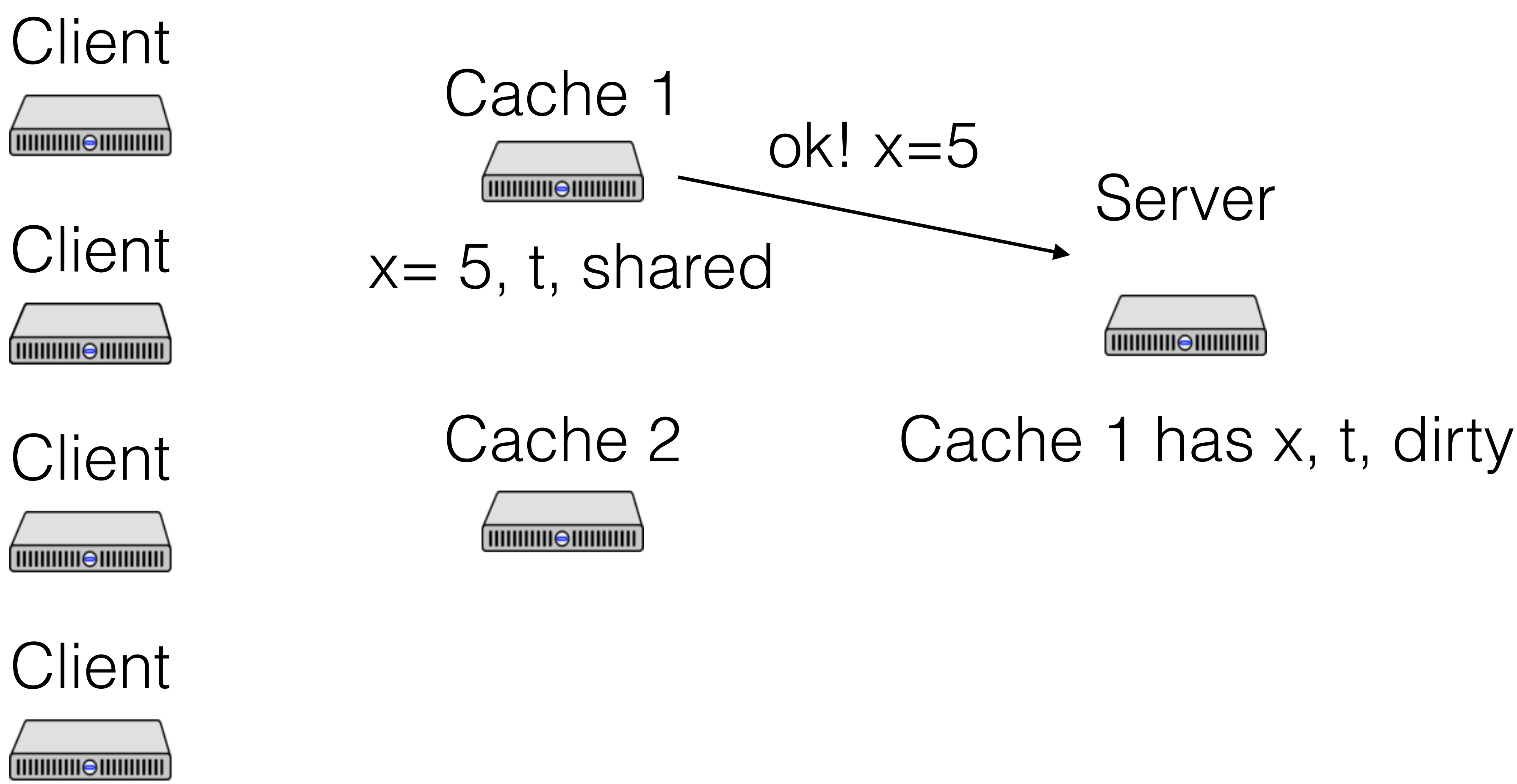
Caching With Leases



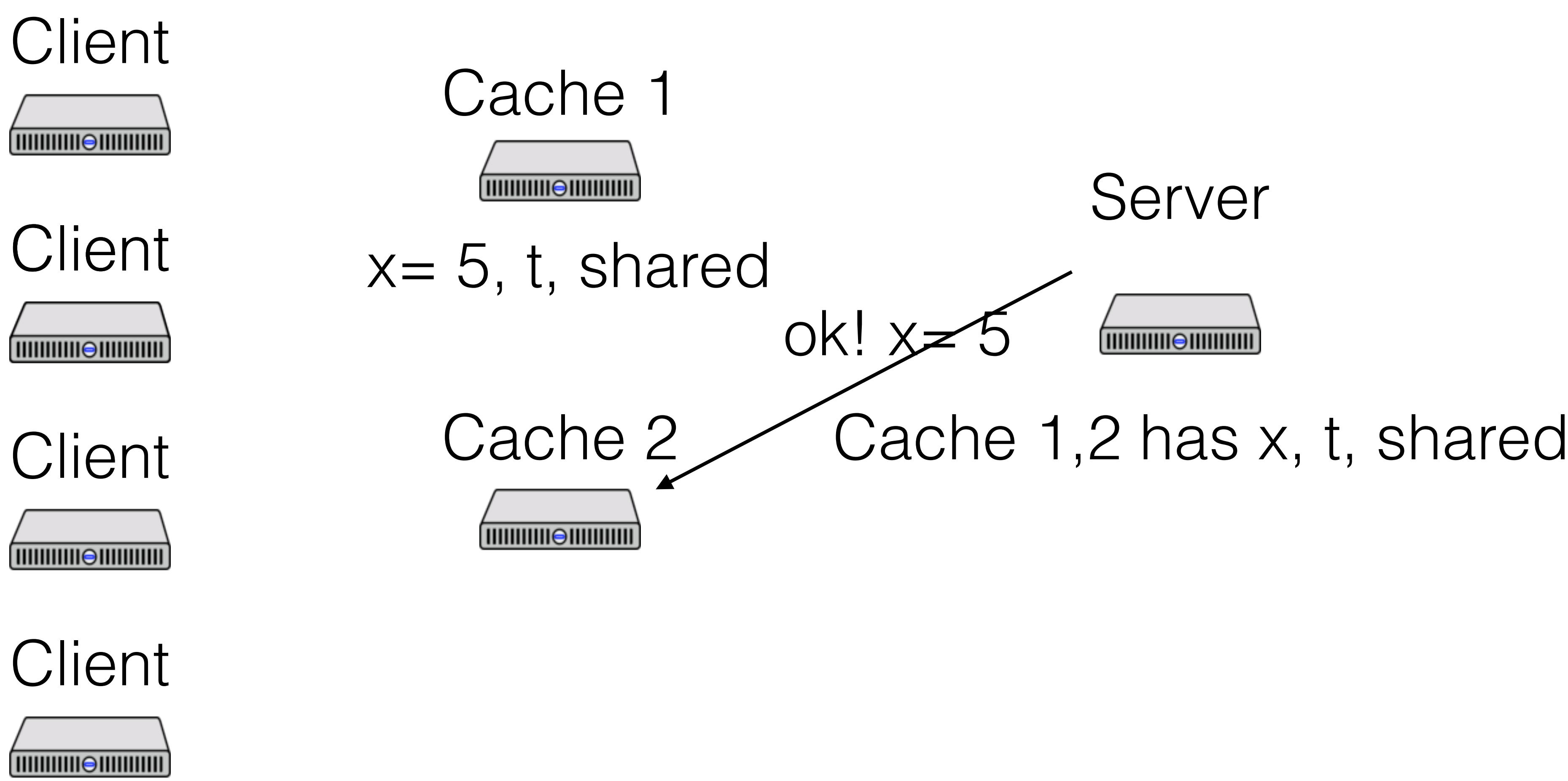
Caching With Leases



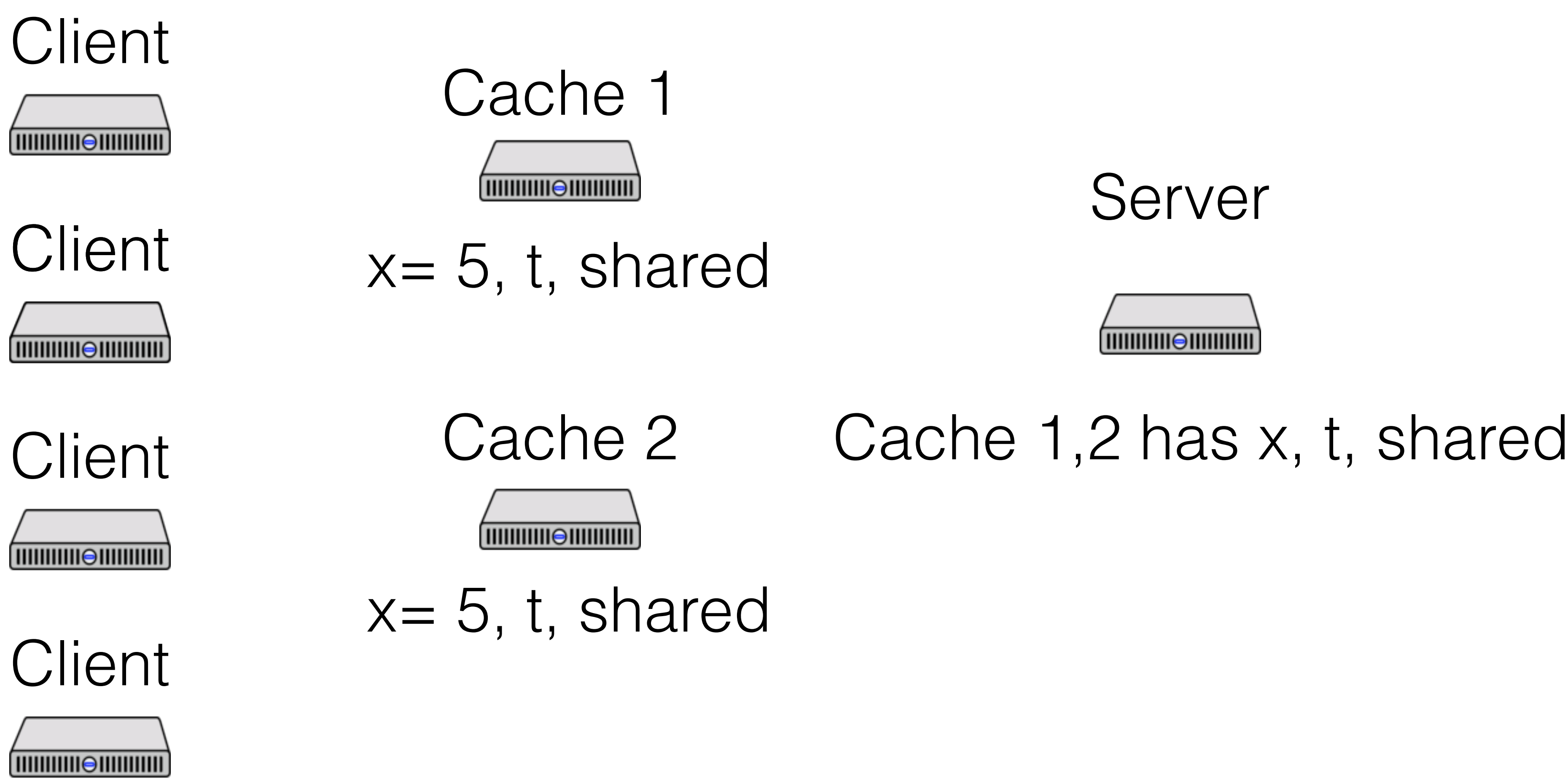
Caching With Leases



Caching With Leases



Caching With Leases



Questions

While a write to x is waiting on invalidations, can other clients read old values of x from their caches?

Questions

While a write to x is waiting on invalidations, can the server perform a read to y != x?

Questions

While a write to x is waiting on invalidations, can the server perform a write (from another cache) to y != x?

Questions

While a write to x is waiting on invalidations, can the server perform a write (from another cache) to y = x?

Write Back Cache Coherence

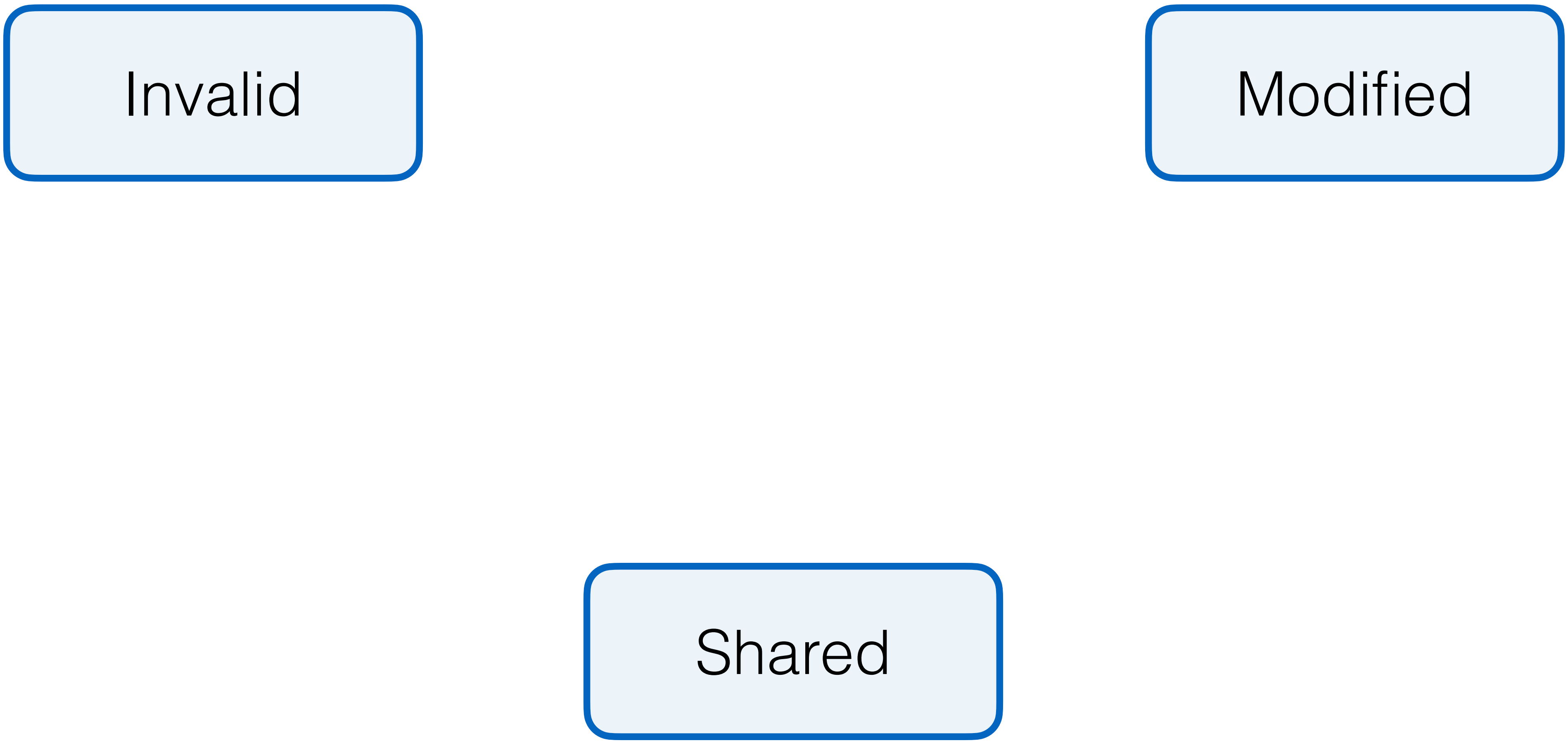
On a write:

- Send invalidations to all caches
- Each cache invalidates, responds (possibly with updated data)
- Wait for all invalidations
- Return

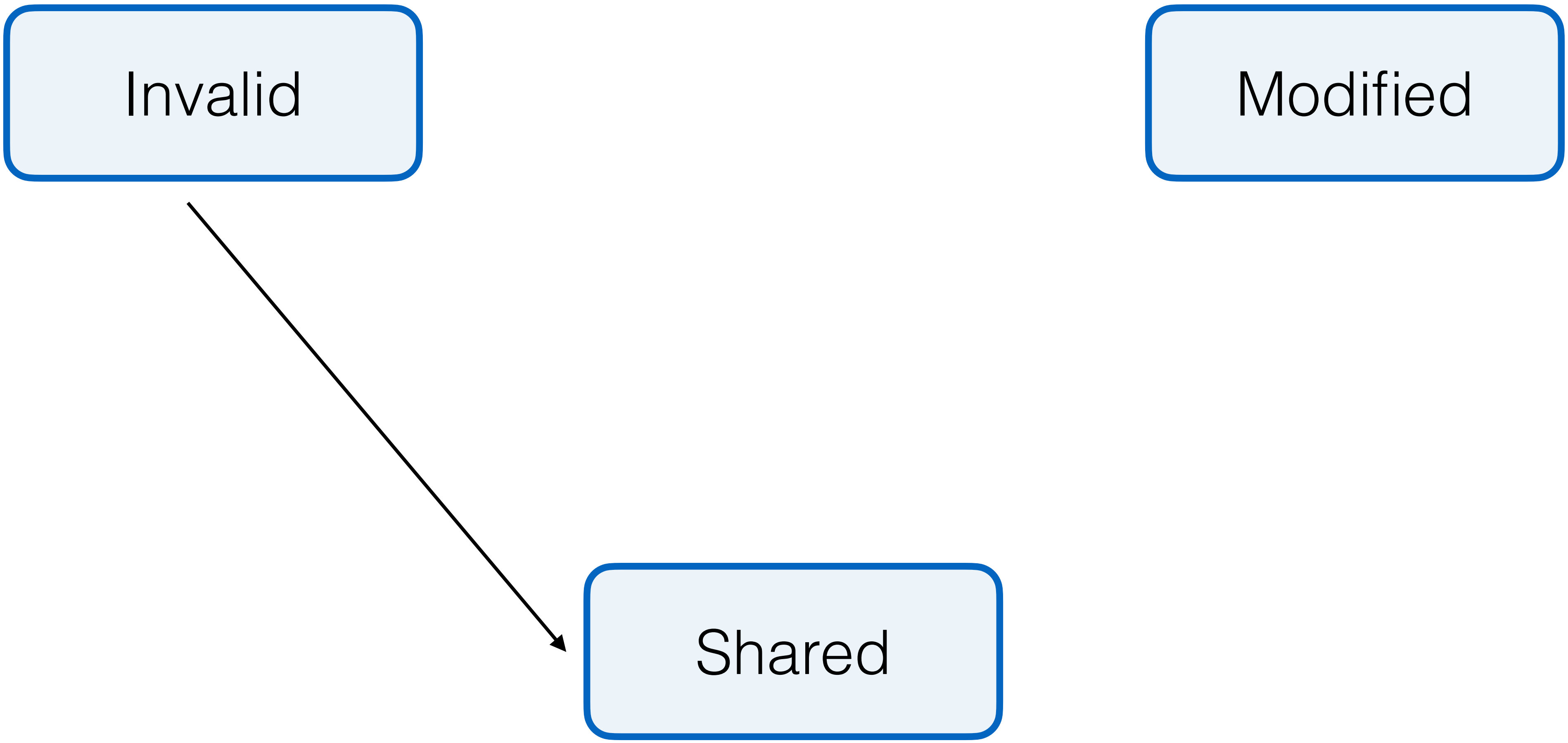
Reads can proceed when there is a local copy

Order requests carefully at server, avoid deadlock

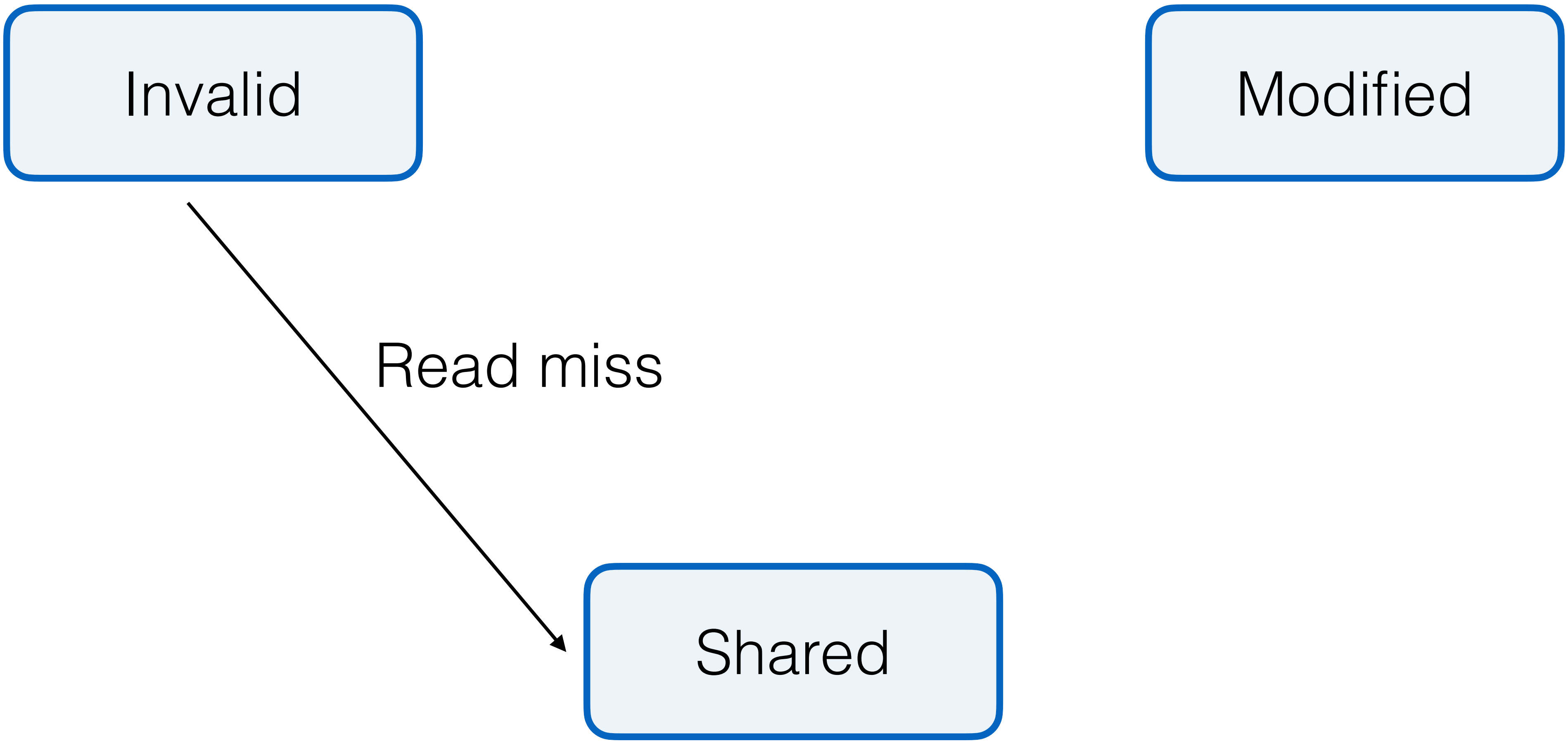
MSI



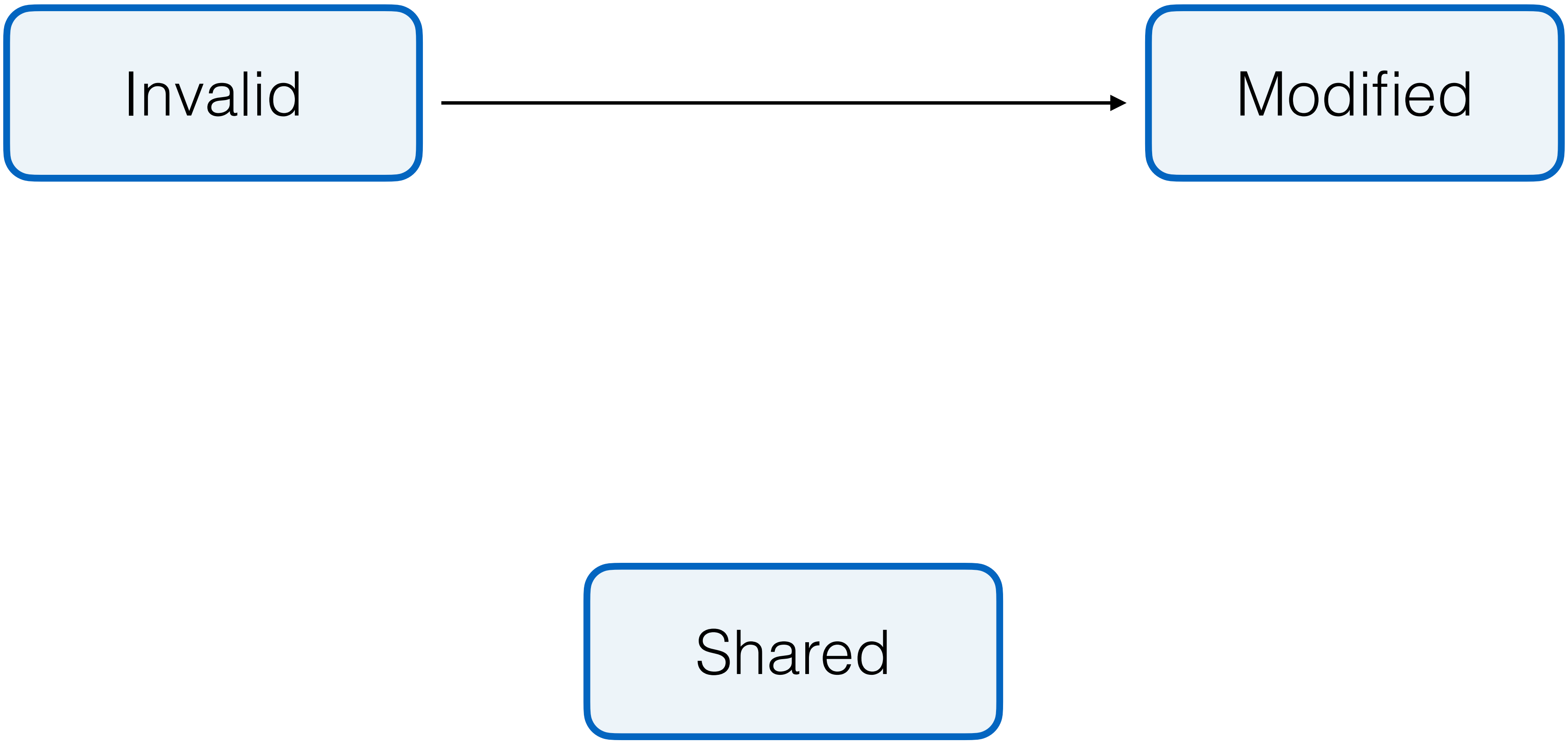
MSI



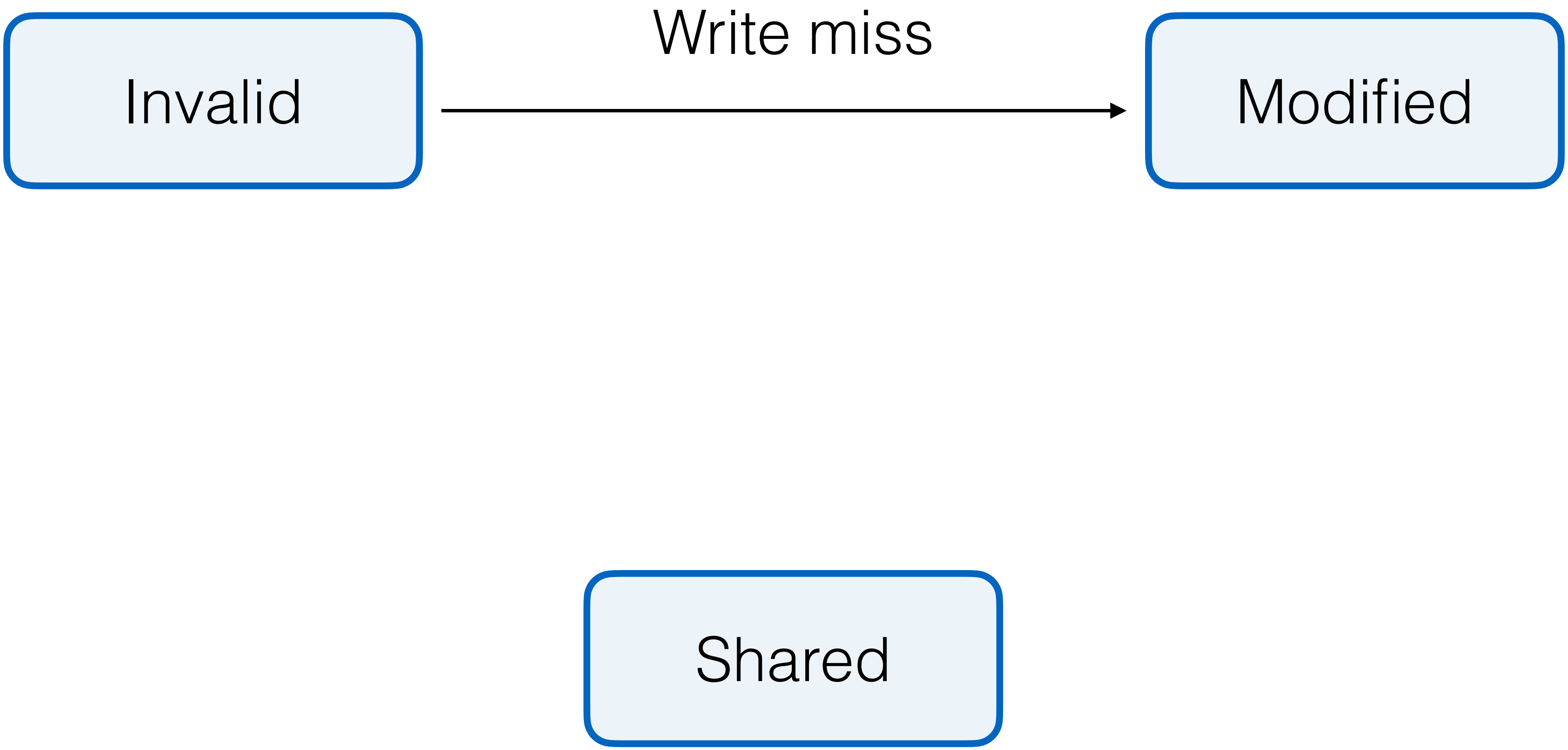
MSI



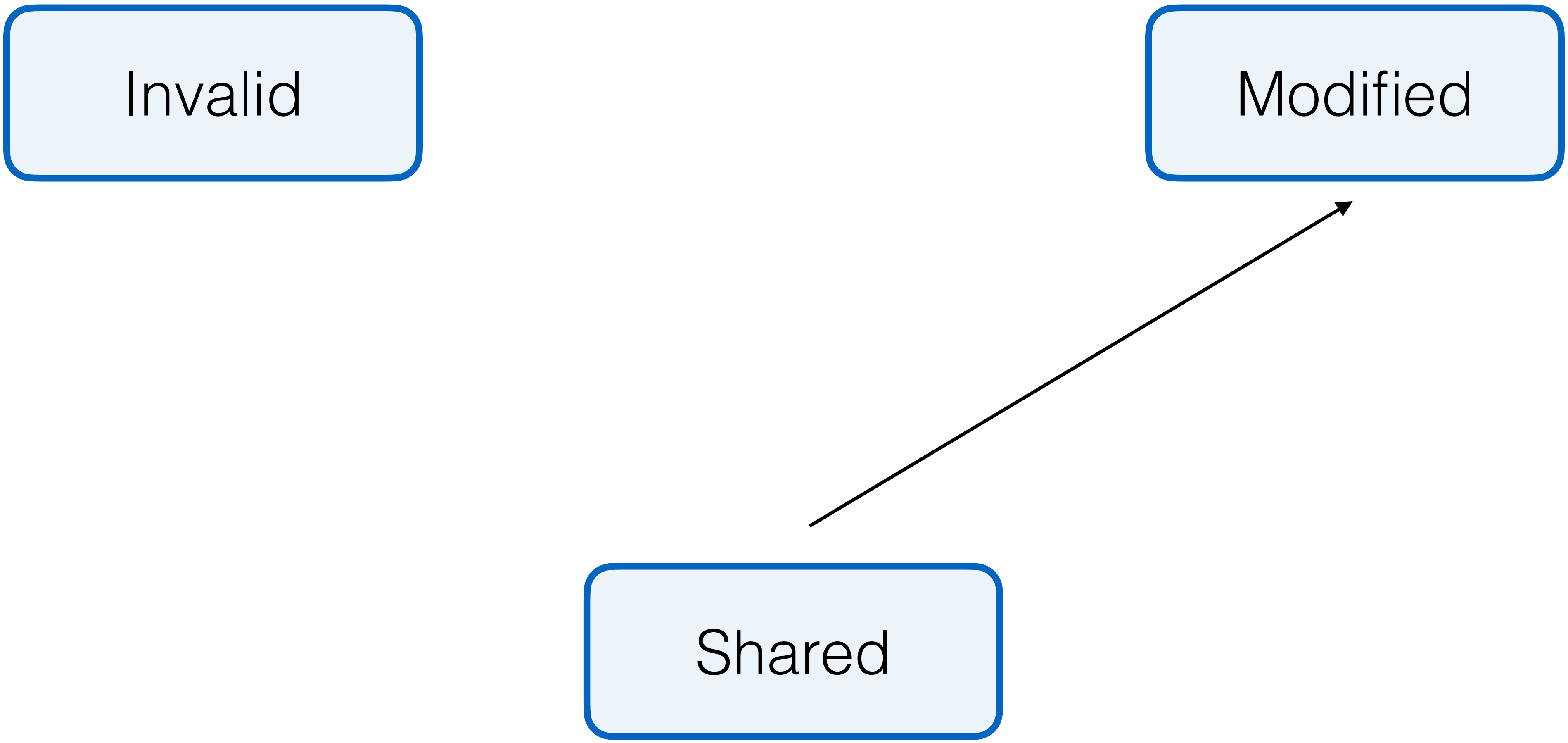
MSI



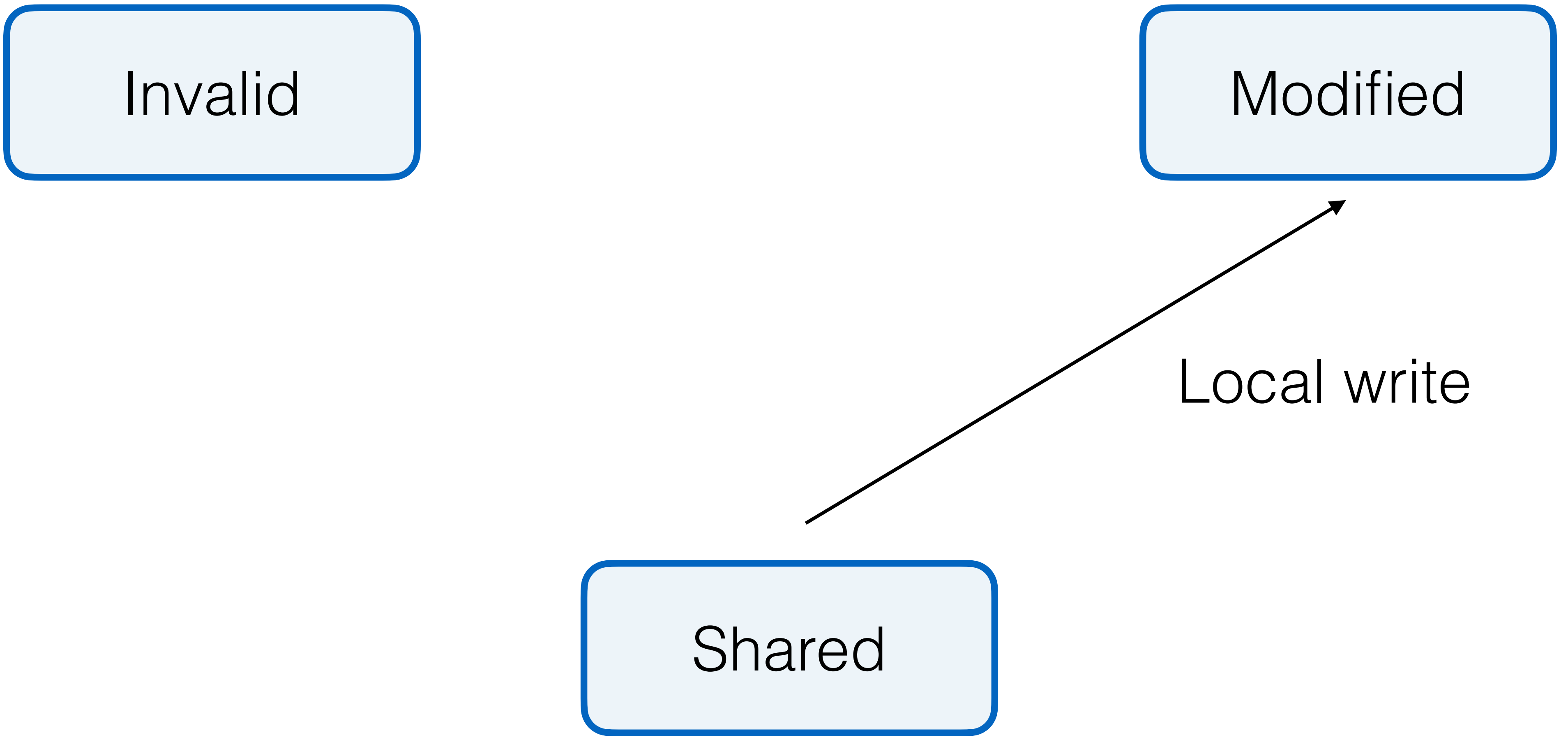
MSI



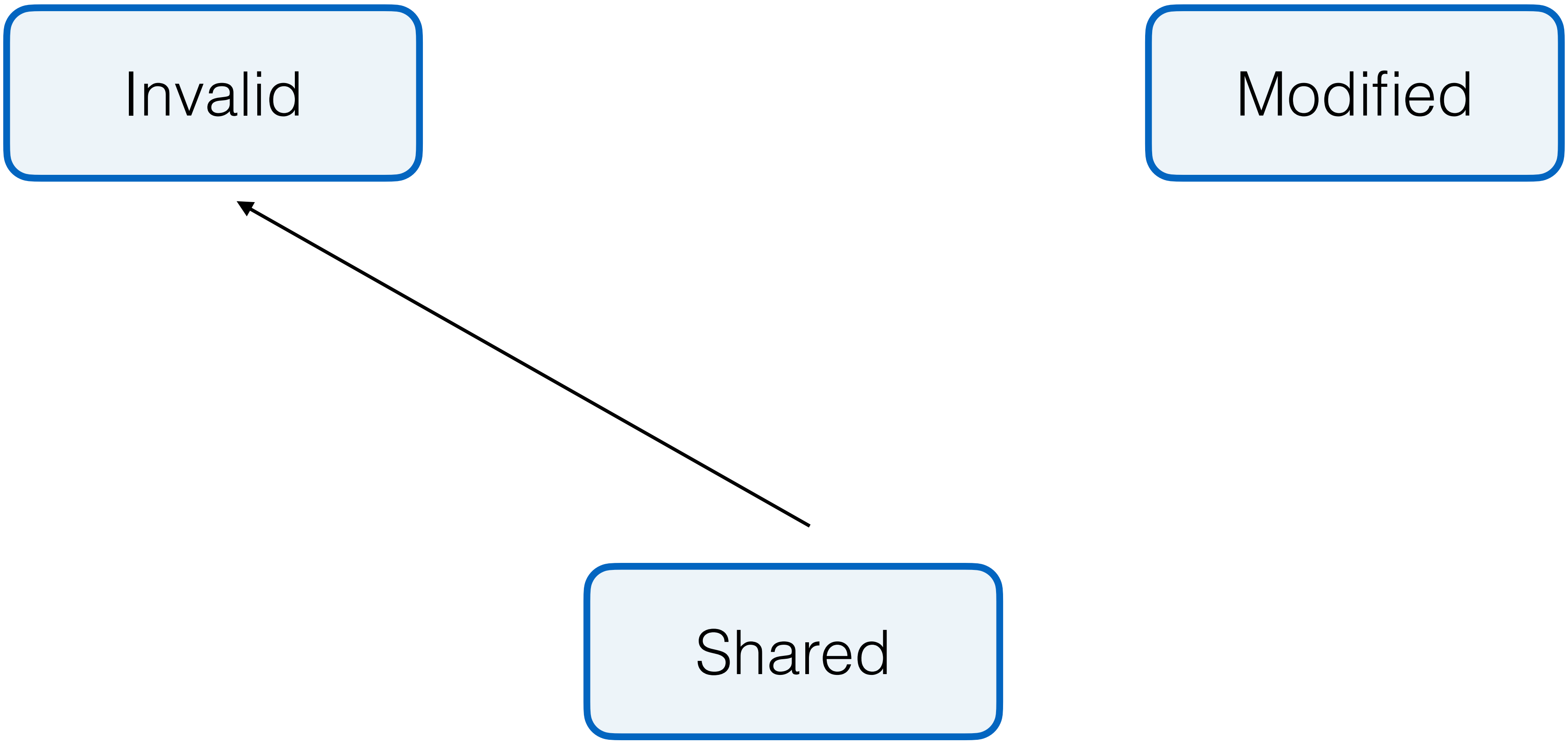
MSI



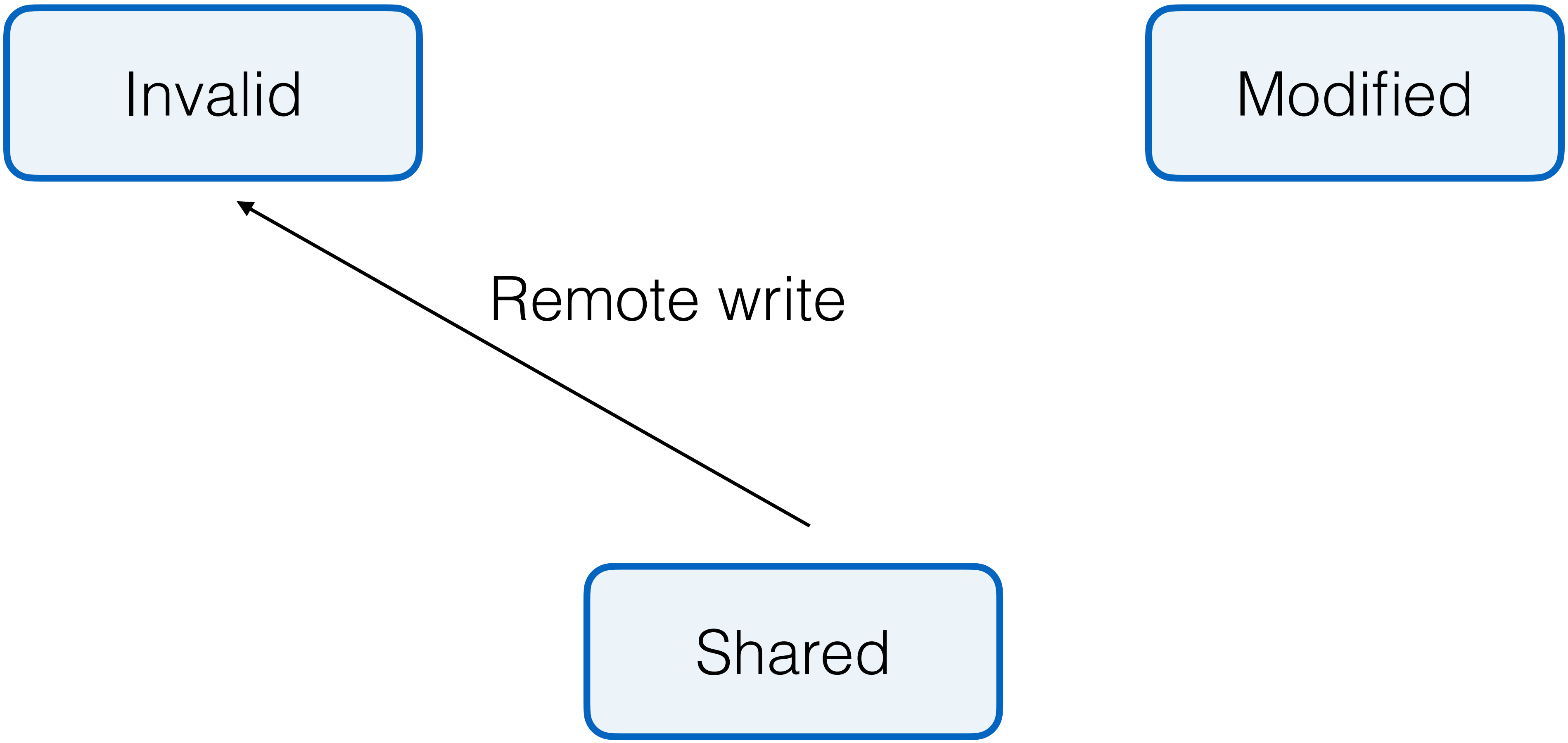
MSI



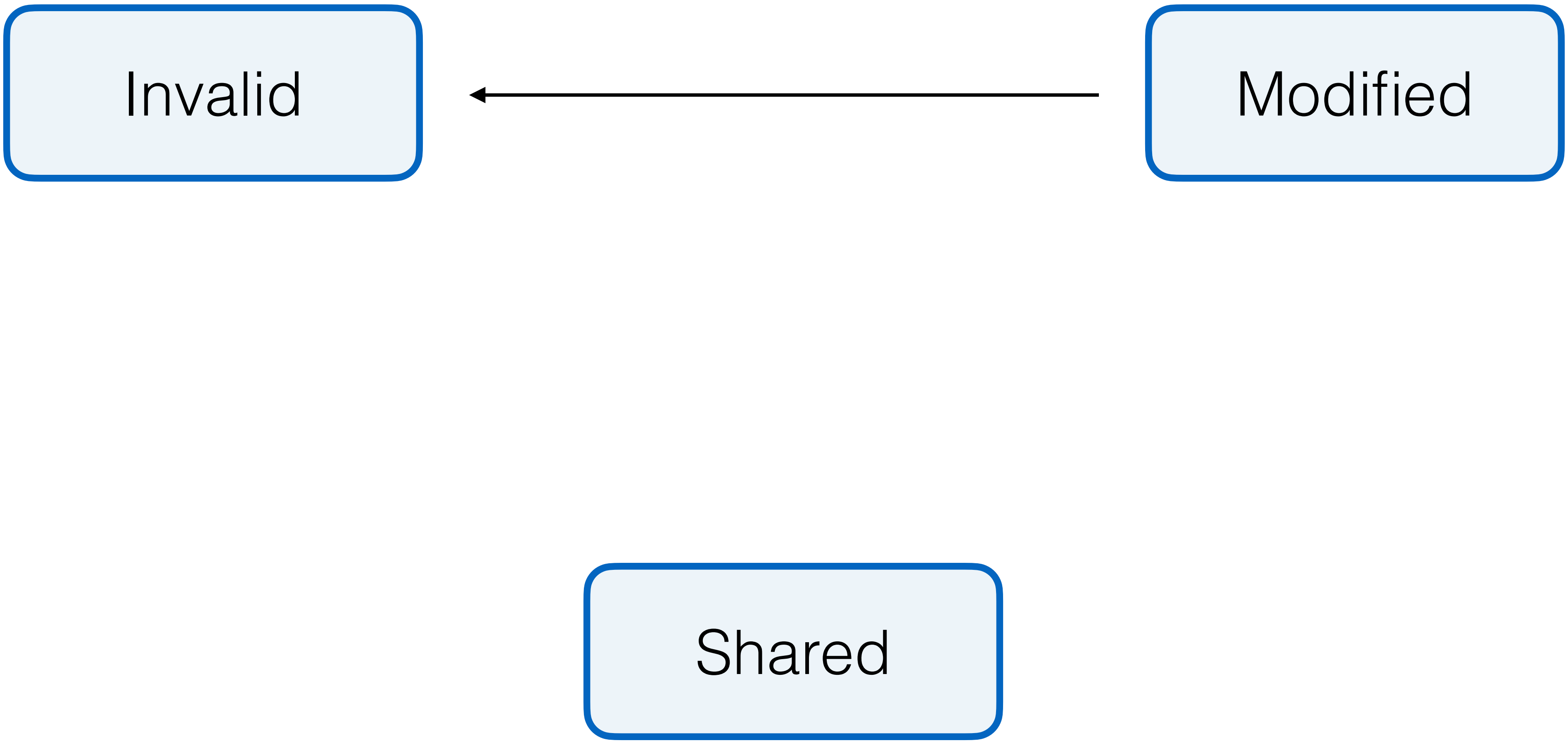
MSI



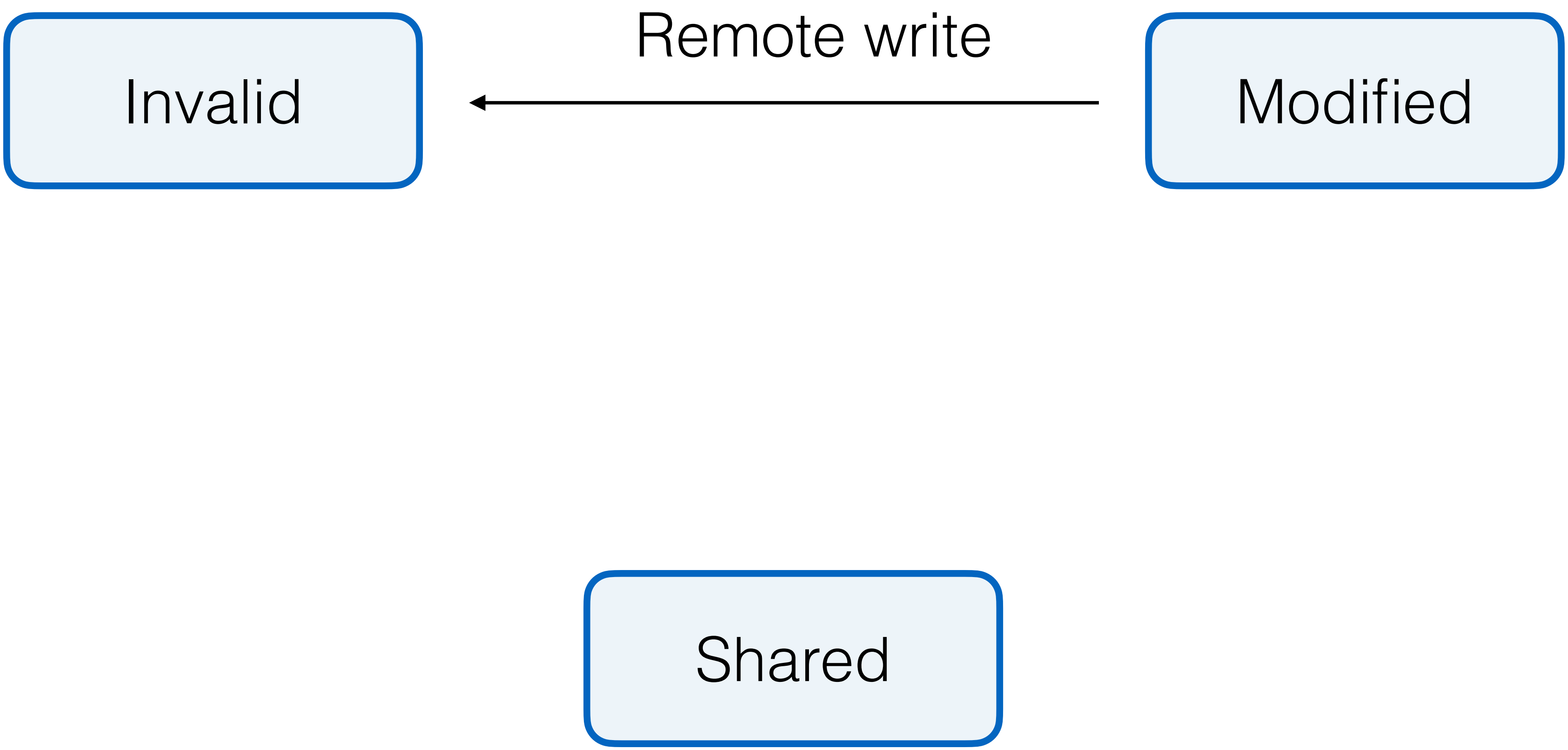
MSI



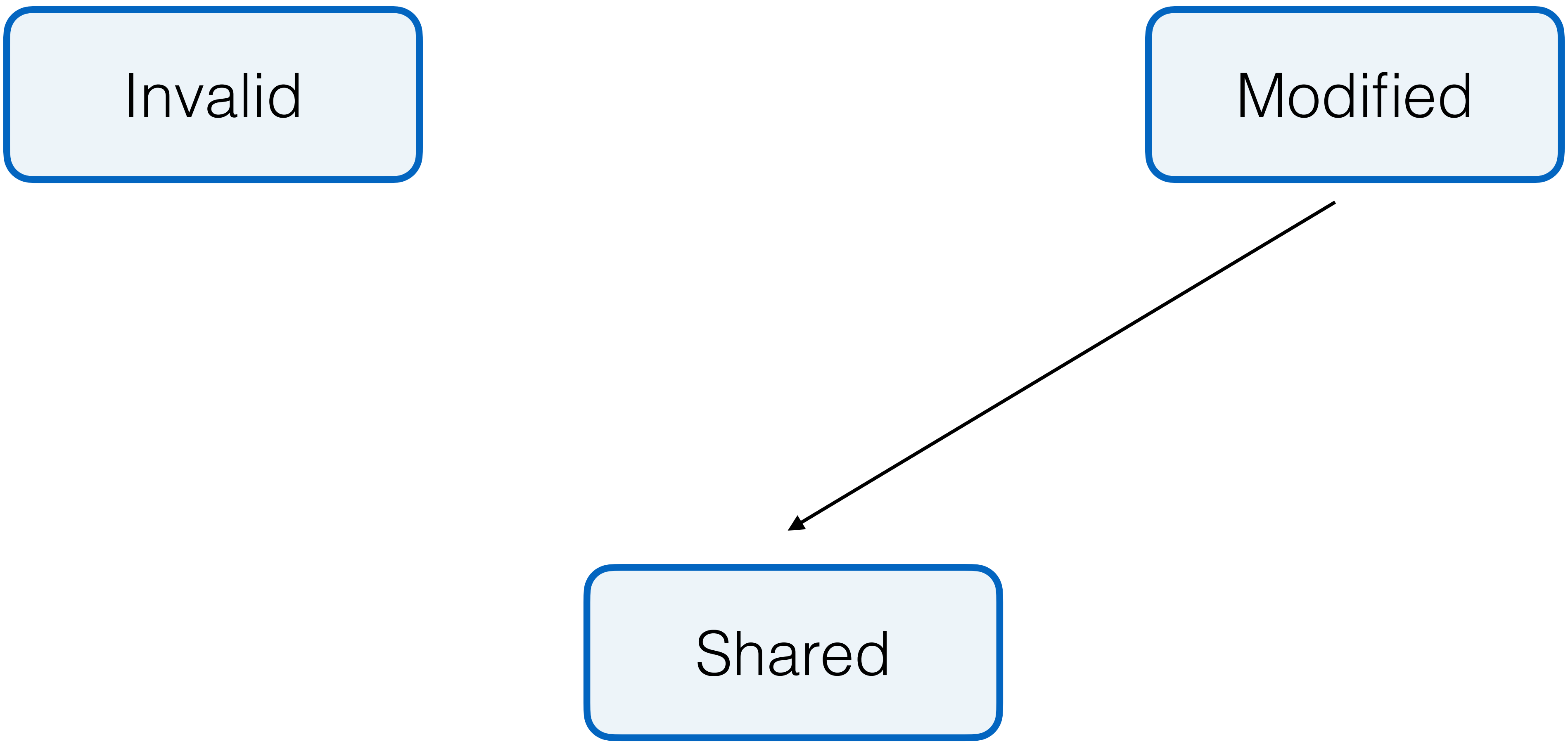
MSI



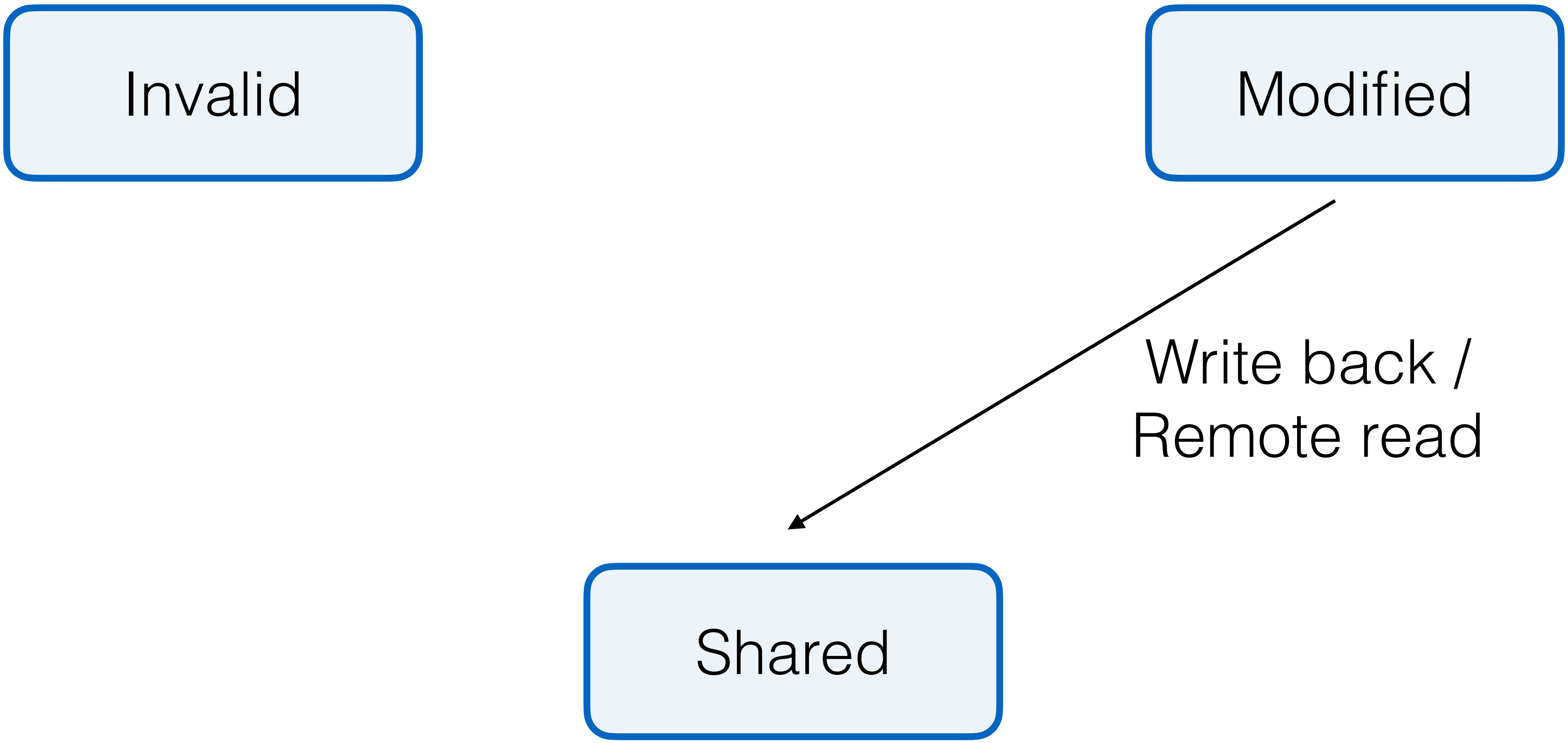
MSI



MSI



MSI



MESI

Motivation:

- Common pattern: read, then write
- MSI inefficient when doing a read and then a write
- If no one else has a copy, can "claim" it with the read

Four cache states:

- **M**odified: this is the only copy, it's dirty
- **E**xclusive: this is the only copy, it's clean
- **S**hared: this is one of many copies, it's clean
- **I**nvalid

MESI allowed states

	M	E	S	I
M				✓
E				✓
S			✓	✓
I	✓	✓	✓	✓

False Sharing

Expensive to keep track of MESI for every memory location

Instead, coarse-grained record-keeping

- CPUs: cache line granularity
- File systems: file/file block granularity

What if two clients try to modify different memory locations in same block, concurrently?

- Cache line can only be “dirty" in one at a time
- Correct behavior, but slow

Atomic Read-Modify-Write

RMW needed to implement spinlocks and other sync

Request cache line exclusive/modified

Delay concurrent remote read/write misses until entire operation completes

Multi-key Transactions

Often want to read/modify multiple keys atomically

Acquire cache lines in MESI state

If remote miss during transaction

- Abort, erase modifications, and try again
- Or delay until done

If reach end of transaction without remote miss

- Success!

Weak leases

Cache valid until lease expires

Allow writes, other reads simultaneously

Semantics?

Weak leases

Examples: NFS, DNS, web browsers

Advantages

- Stateless at server (don’t care who is caching)
- Reads, writes always processed immediately

Disadvantages

- Consistency model (!!!)
- Overhead of revalidations
- Synchronized revalidations