[537] AFS

Chapter 49

Tyler Harter

12/03/14

File-System Case Studies

Local
- **FFS**: Fast File System
- **LFS**: Log-Structured File System

Network
- **NFS**: Network File System
- **AFS**: Andrew File System

File-System Case Studies

Local
- **FFS**: Fast File System
- **LFS**: Log-Structured File System

Network
- **NFS**: Network File System
- **AFS**: Andrew File System [today]

NFS Review

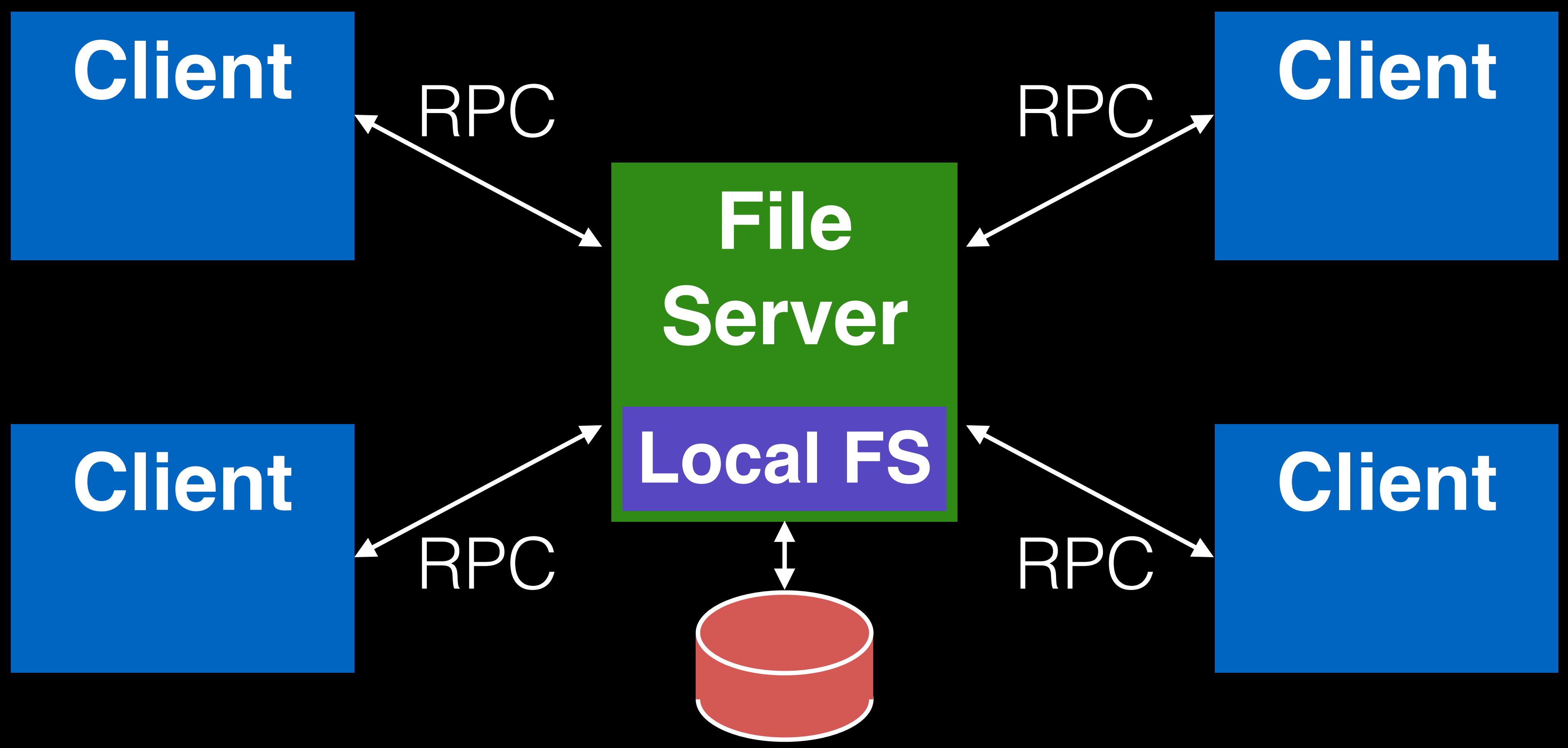
NFS

Export local FS to network
 - many machines may mount

Goal: fast/simple crash recovery

Transparent access

NFS Arch


Client
Client
Client
Client
RPC
RPC
RPC
RPC
File Server
Local FS

If at first you don't succeed,
and you're **stateless** and **idempotent**,
then try, try again.

Idempotent

Applying f() once or N>1 times has same result.

Why is retry hard if we’re not idempotent?

Idempotent

Applying f() once or N>1 times has same result.

Why is retry hard if we're not idempotent?

Retry may cause the operation to run multiple times, resulting in wrong state.

E.g., stupid e-commerce sites that double charge if you "click back or refresh" aren't idempotent.

Stateless

Server still keeps state! Just not about clients.

E.g., we don't have an "open" call for NFS.

Why is retry hard if we're not stateless?

Stateless

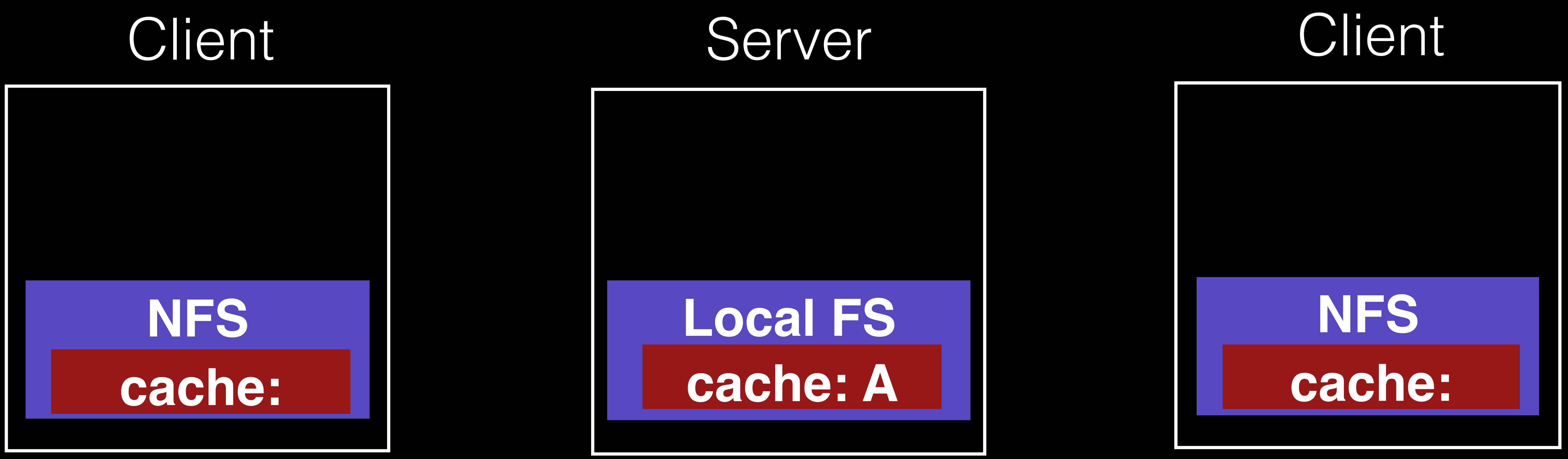
Server still keeps state! Just not about clients.

E.g., we don't have an "open" call for NFS.

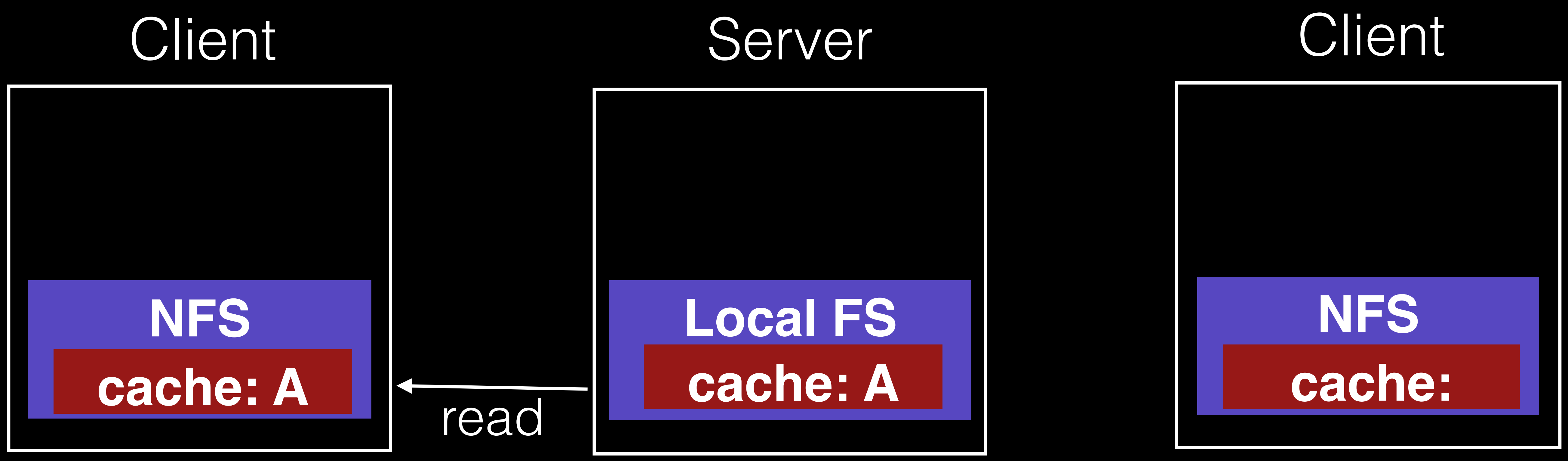
Why is retry hard if we're not stateless?

If server crashes, retried requests don't have any context. E.g., what does "read from fd 5" mean?

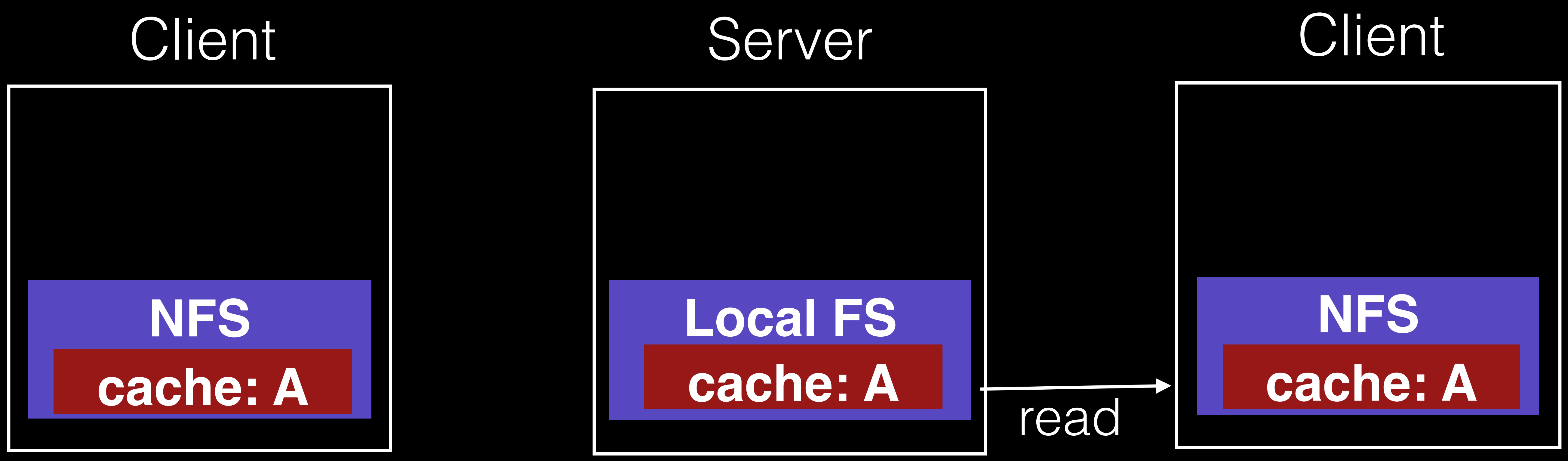
Cache Consistency



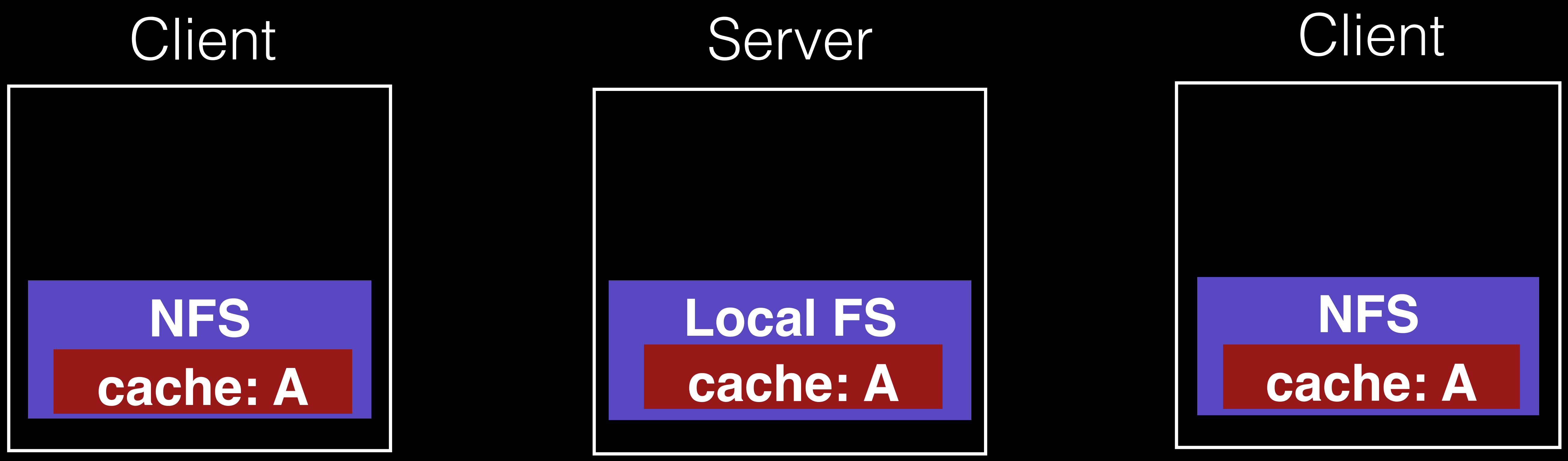
Cache Consistency



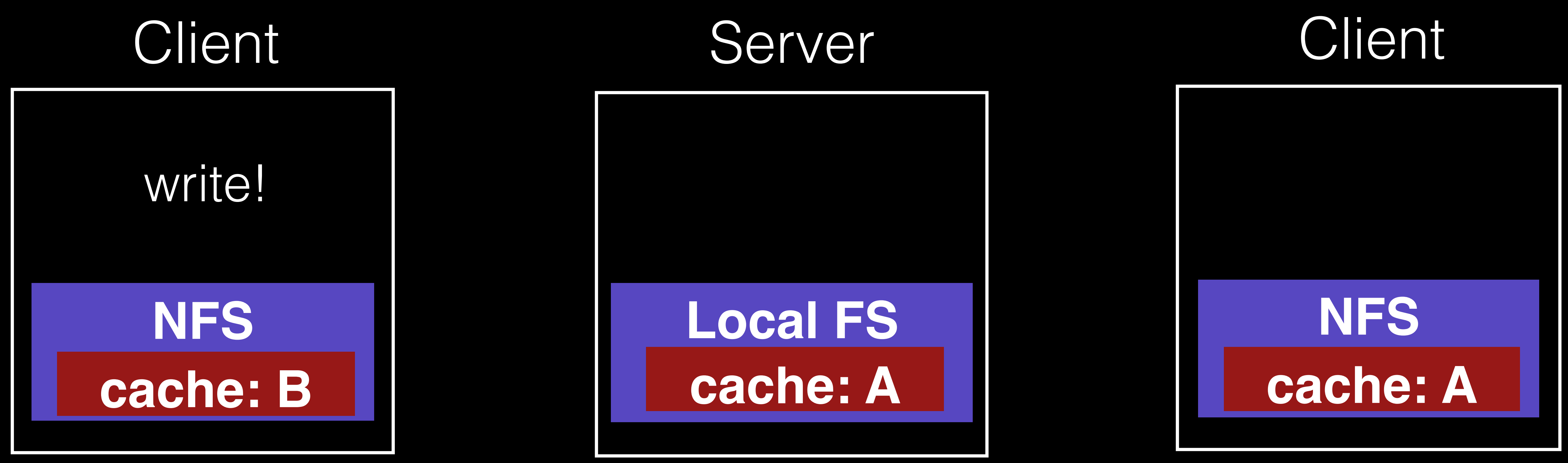
Cache Consistency



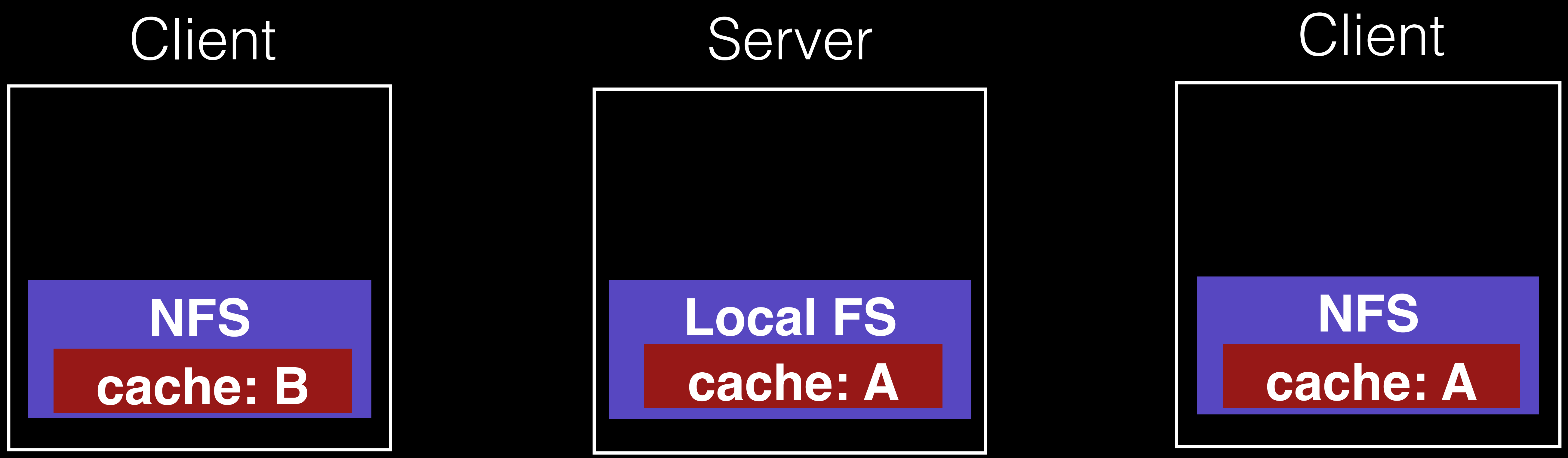
Cache Consistency



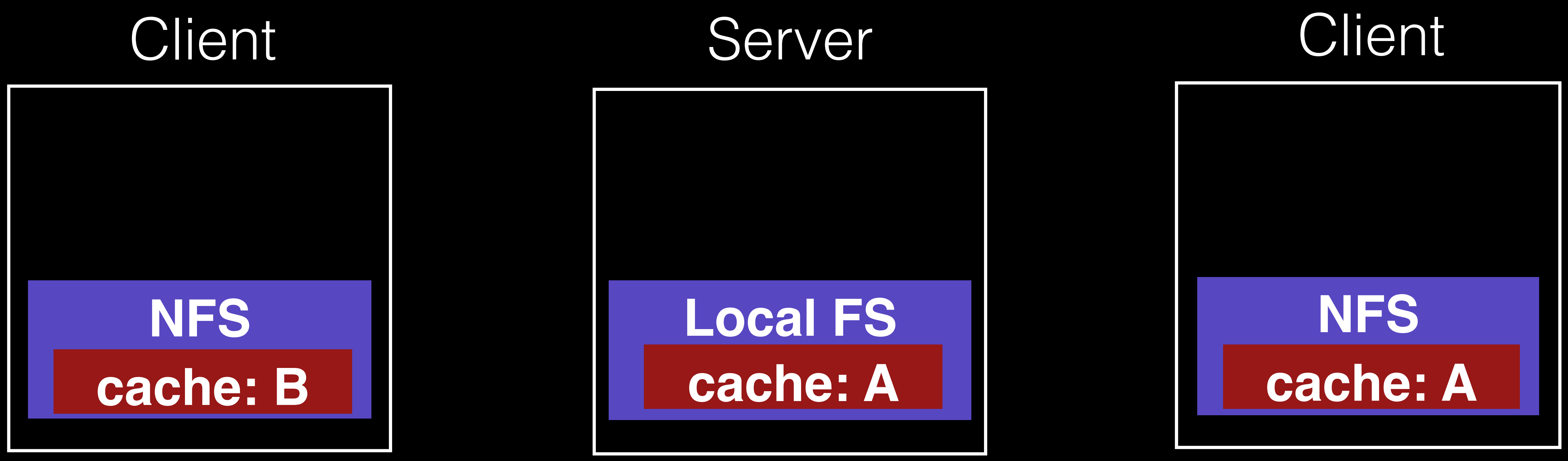
Cache Consistency



Cache Consistency

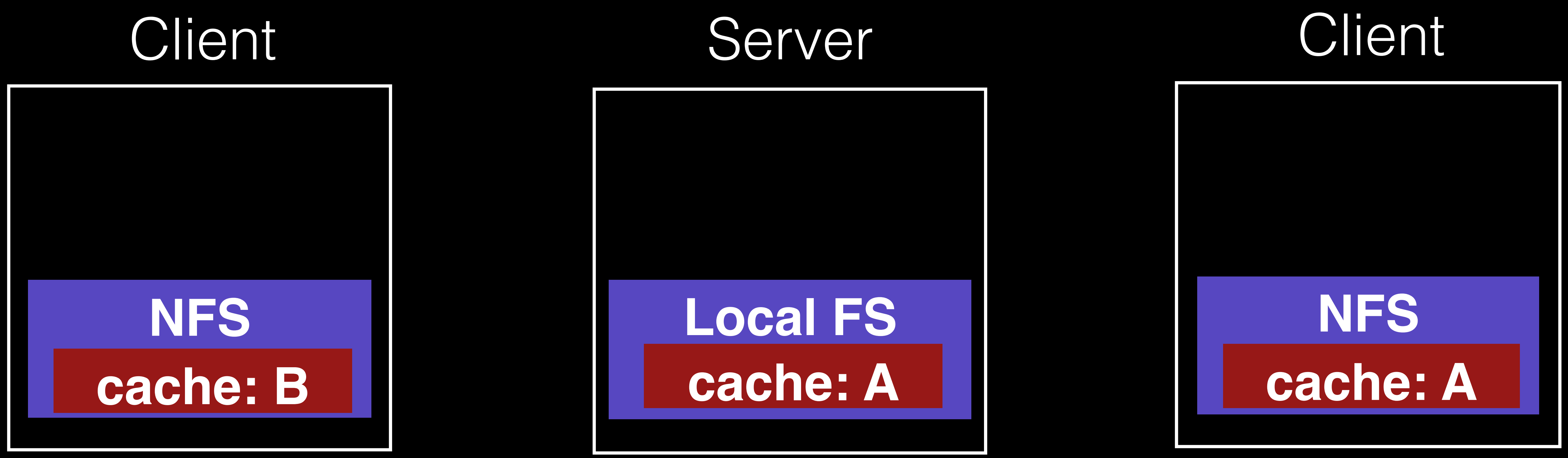


Cache Consistency

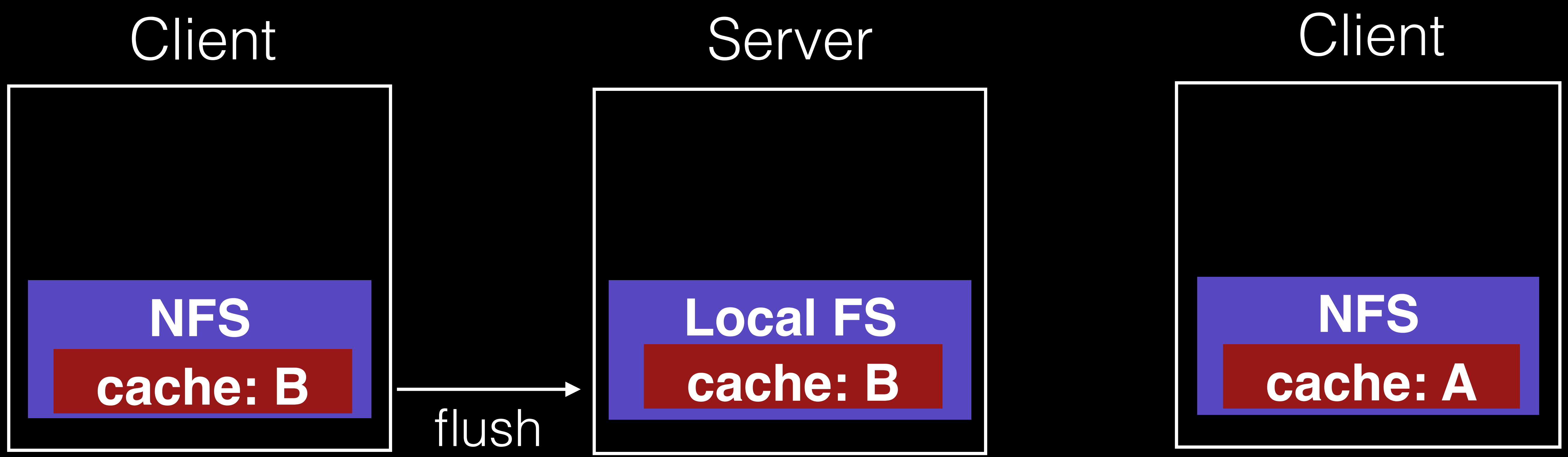


"Update Visibility" problem: server doesn't have latest.

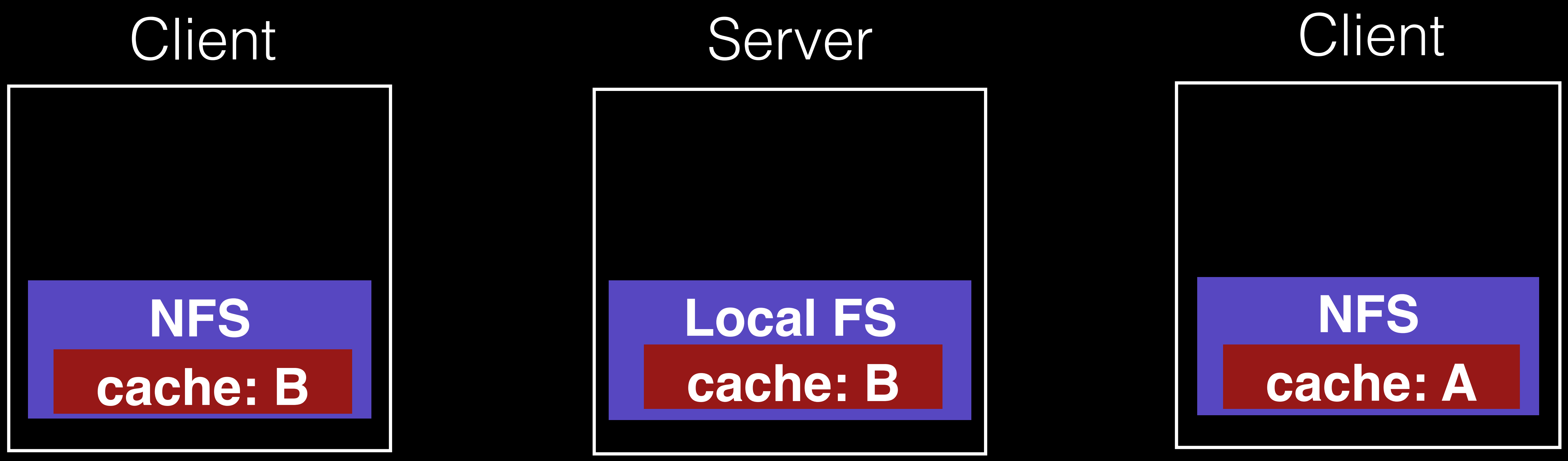
Cache Consistency



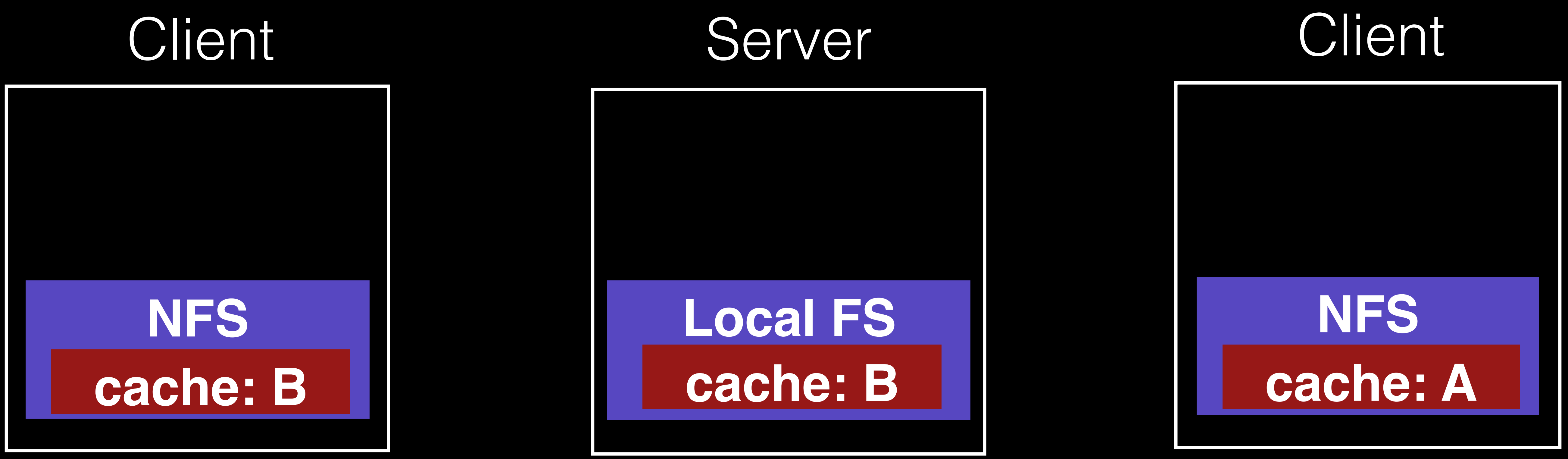
Cache Consistency



Cache Consistency

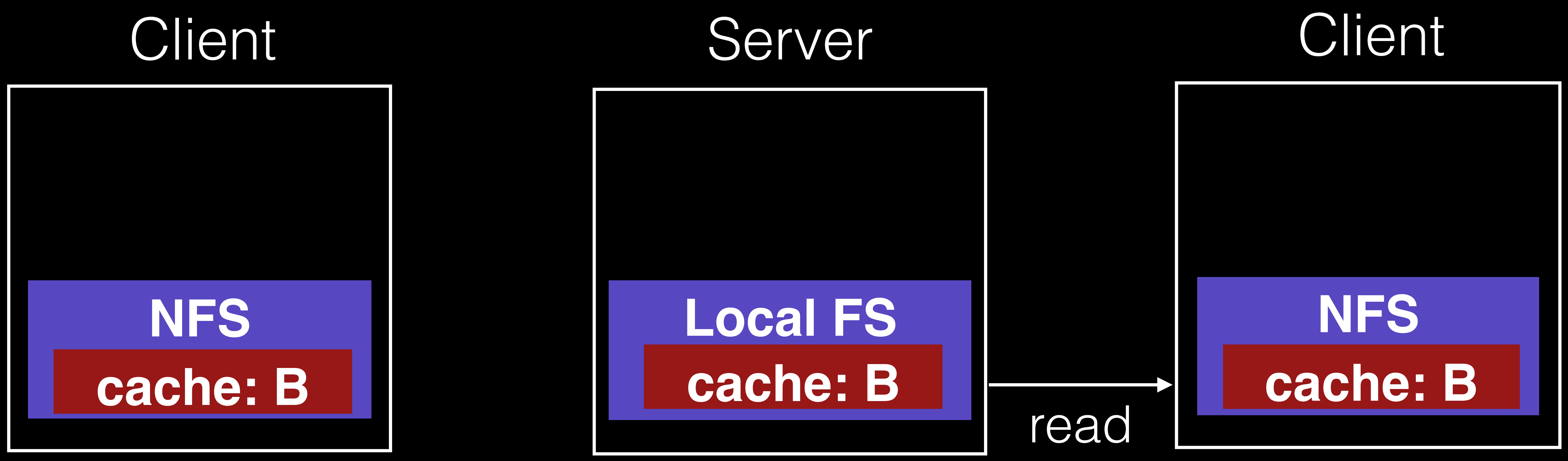


Cache Consistency



“Stale Cache” problem: client doesn’t have latest.

Cache Consistency



NFS

Update visibility: flush buffer on close (or sooner)

Stale cache: check before use (if expired).

No flock.

May often have weird behavior.

Andrew File System

AFS Goals

Primary goal: scalability! (many clients per server)

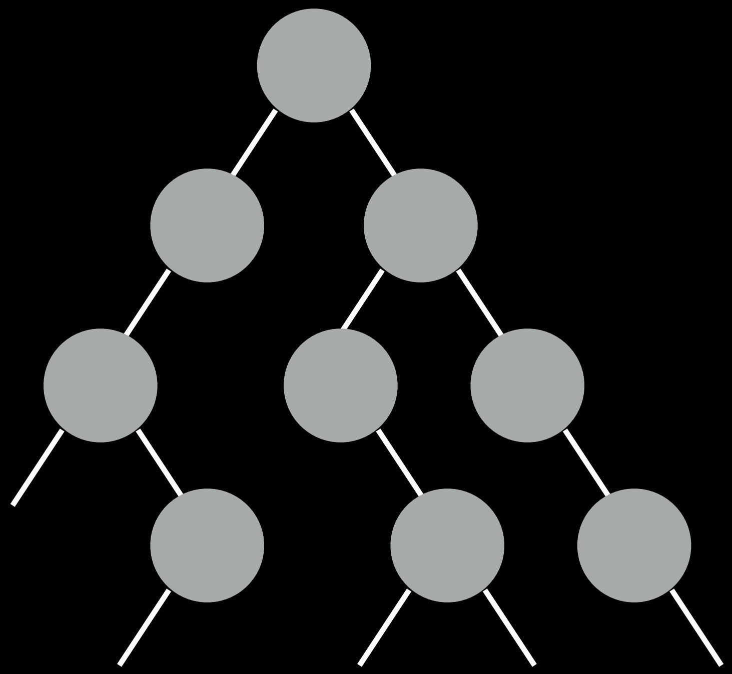
More reasonable semantics for concurrent file access.

Not good about handling some failure scenarios.

AFS Design

NFS: export local FS

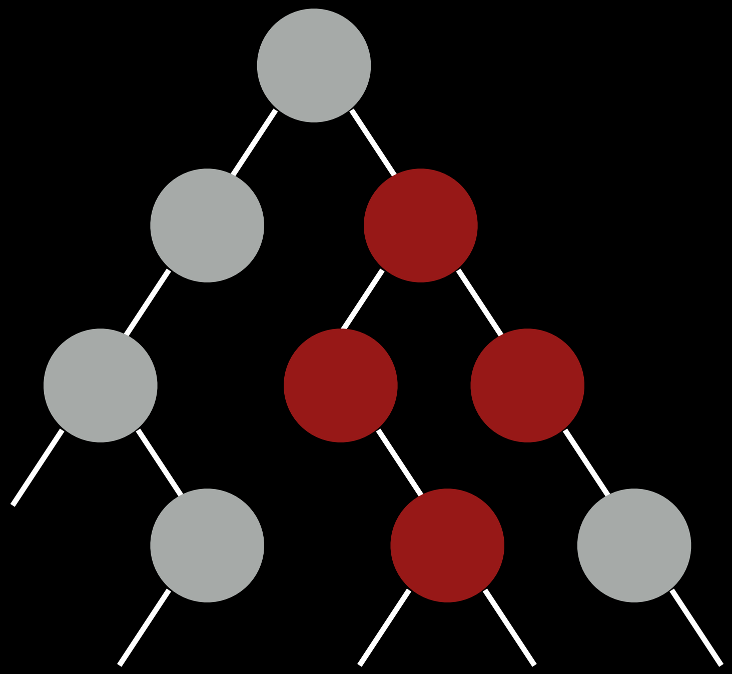
AFS: present big file tree, store across many machines.



AFS Design

NFS: export local FS

AFS: present big file tree, store across many machines.

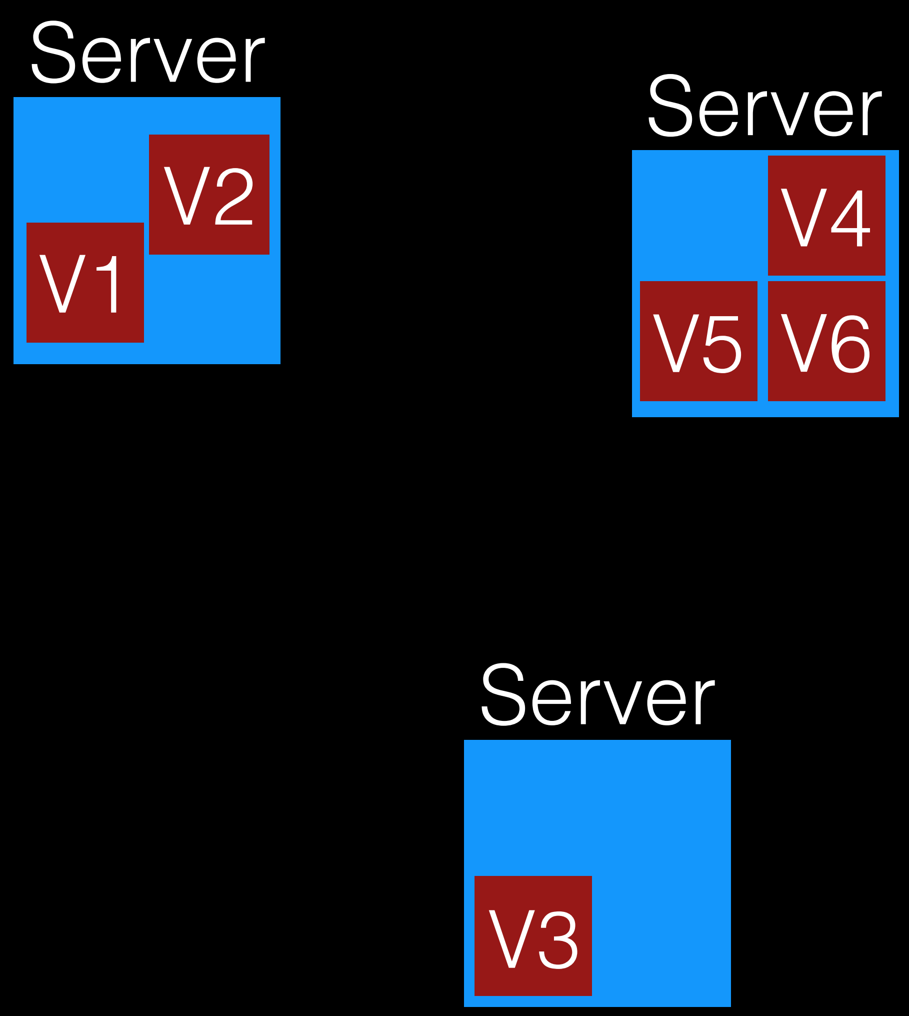


Break tree into "volumes."
I.e., partial sub trees.

Viewing Volumes

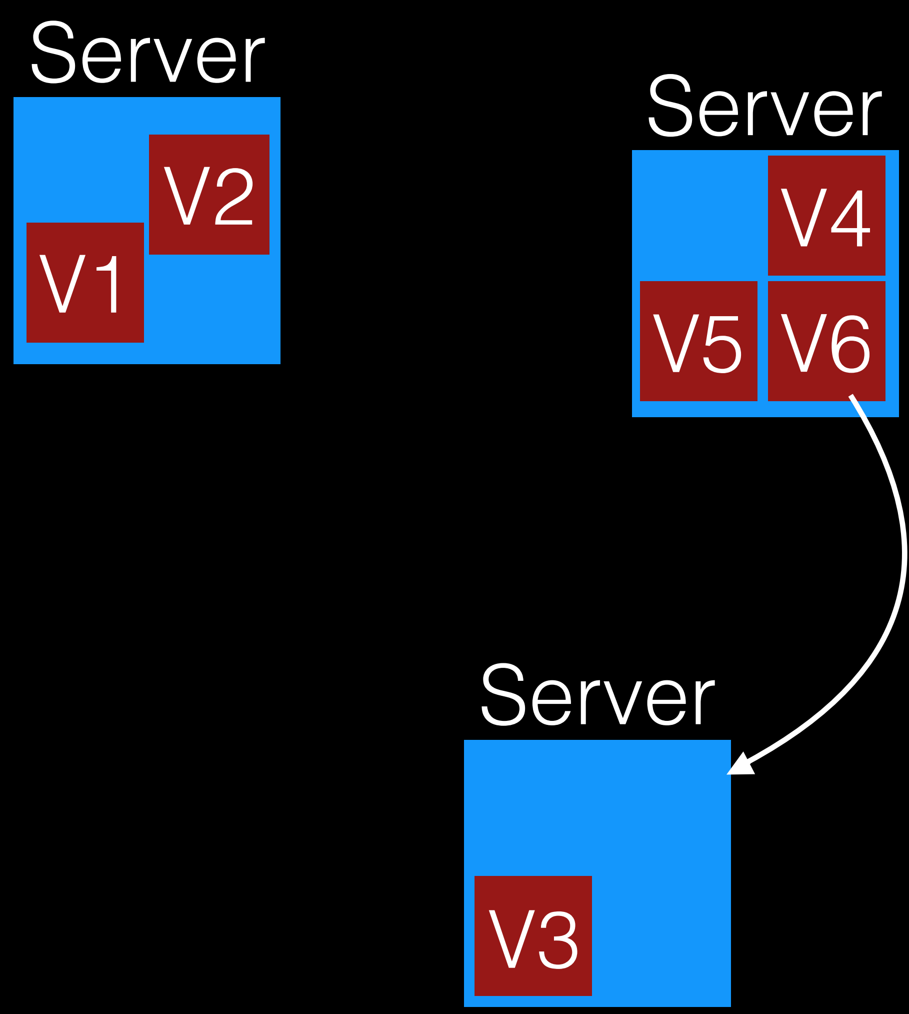
```
[harter@egg] (3)$ pwd
/u/h/a/harter
[harter@egg] (4)$ fs lq
Volume Name          Quota       Used %Used   Partition
u.harter         100000000   12964328   13%         76%
[harter@egg] (5)$ cd /p/wind/
[harter@egg] (6)$ fs lq
Volume Name          Quota       Used %Used   Partition
p.wind.root      100000000    1000208    1%          0%
```

Arch



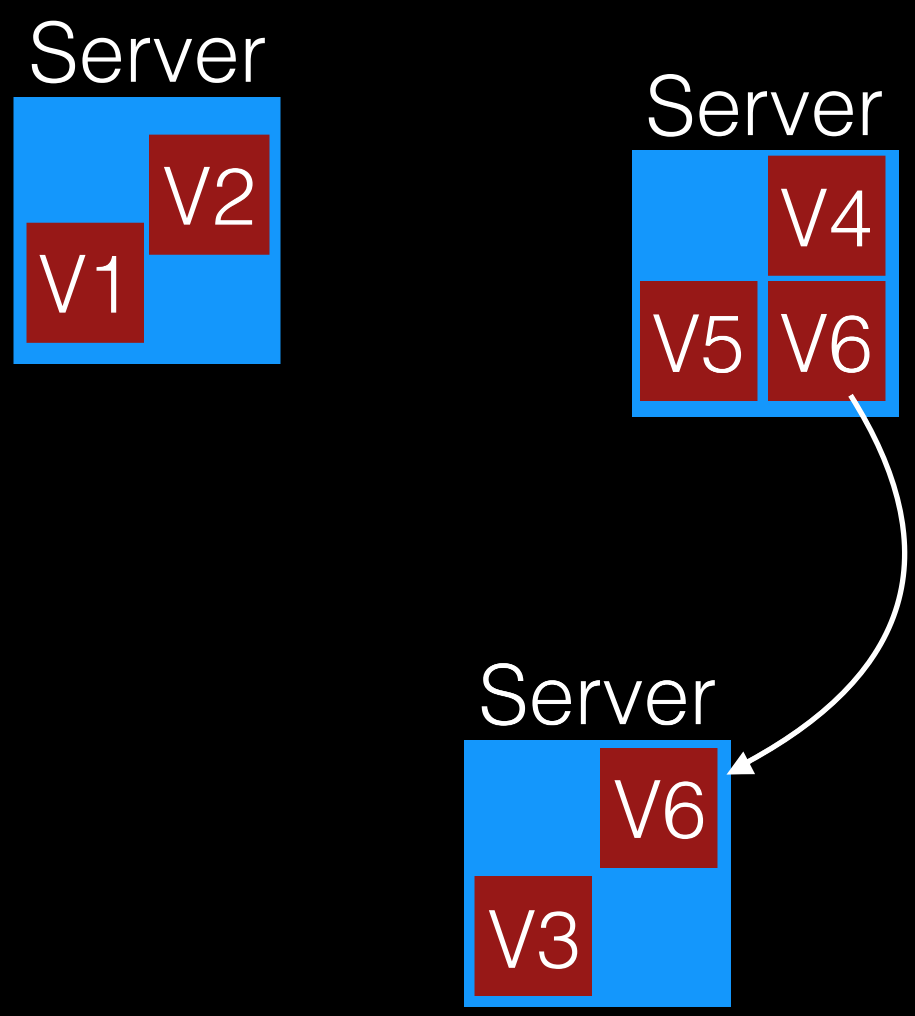
collection of servers store different volumes that together make up file tree.

Arch



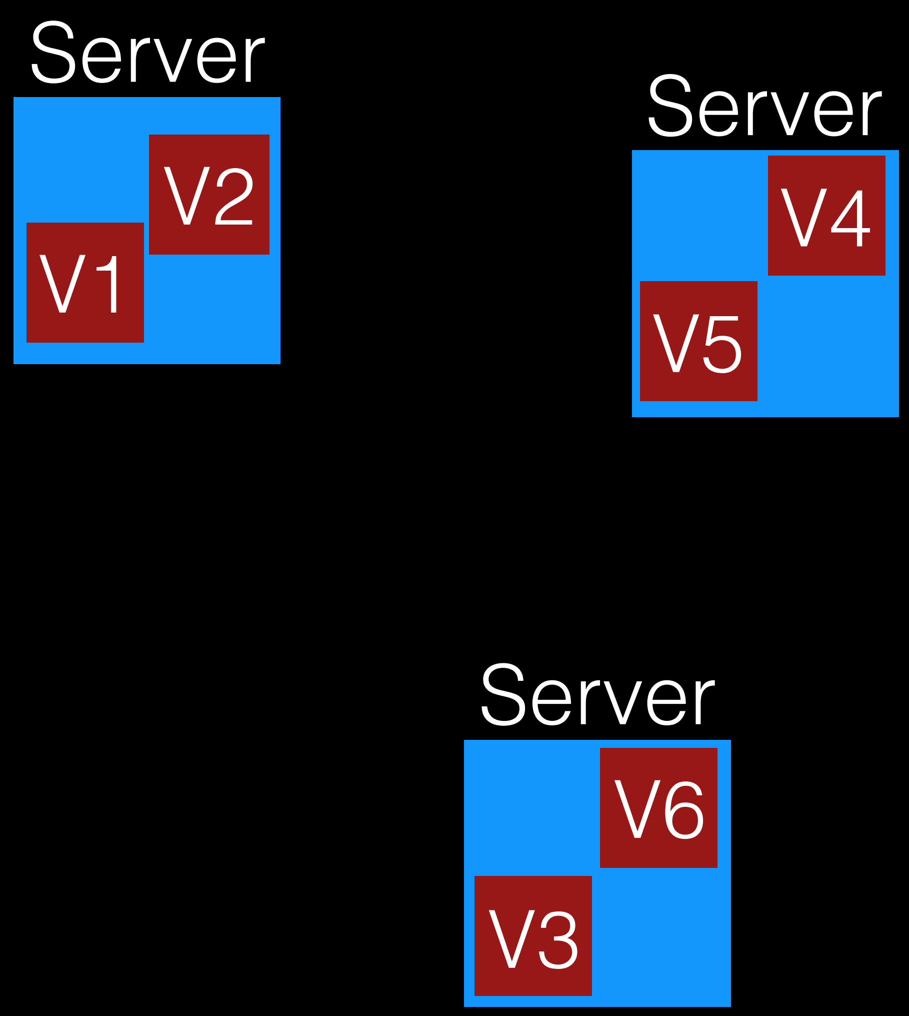
volumes may be moved by an administrator.

Arch



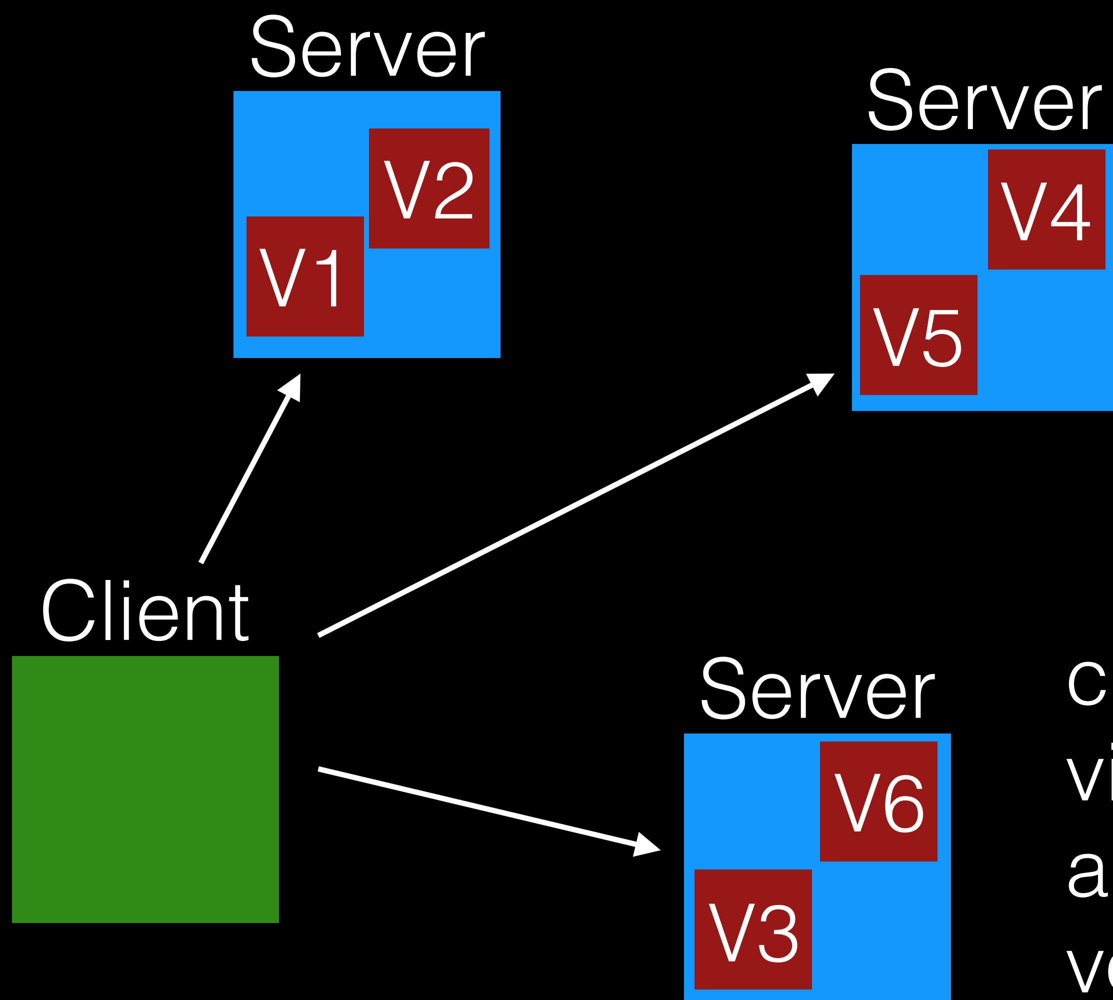
volumes may be moved by an administrator.

Arch



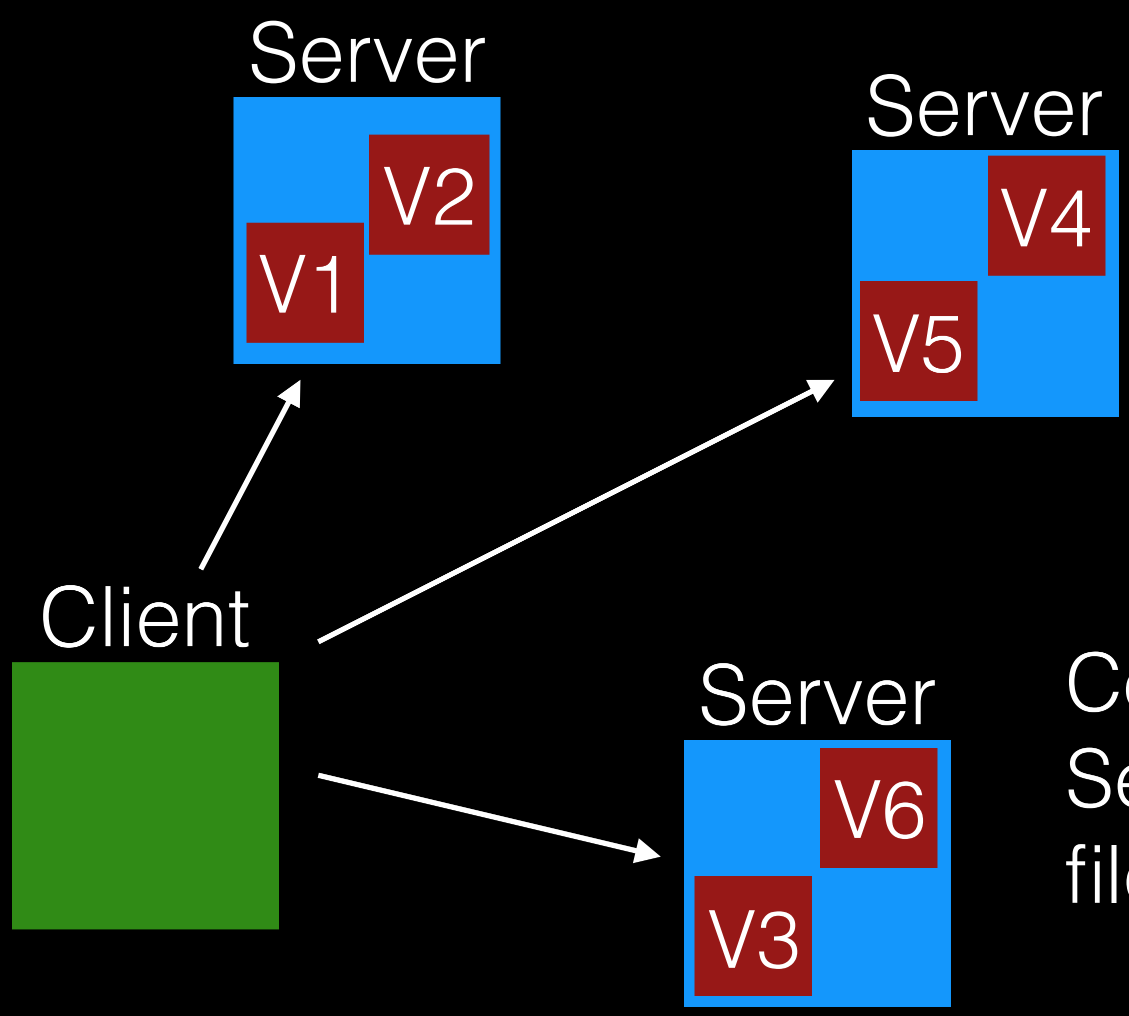
volumes may be moved by an administrator.

Arch



client library gives seamless view of file tree by automatically finding write volumes.

Arch



Communication via RPC.
Servers store data in local file systems.

Outline

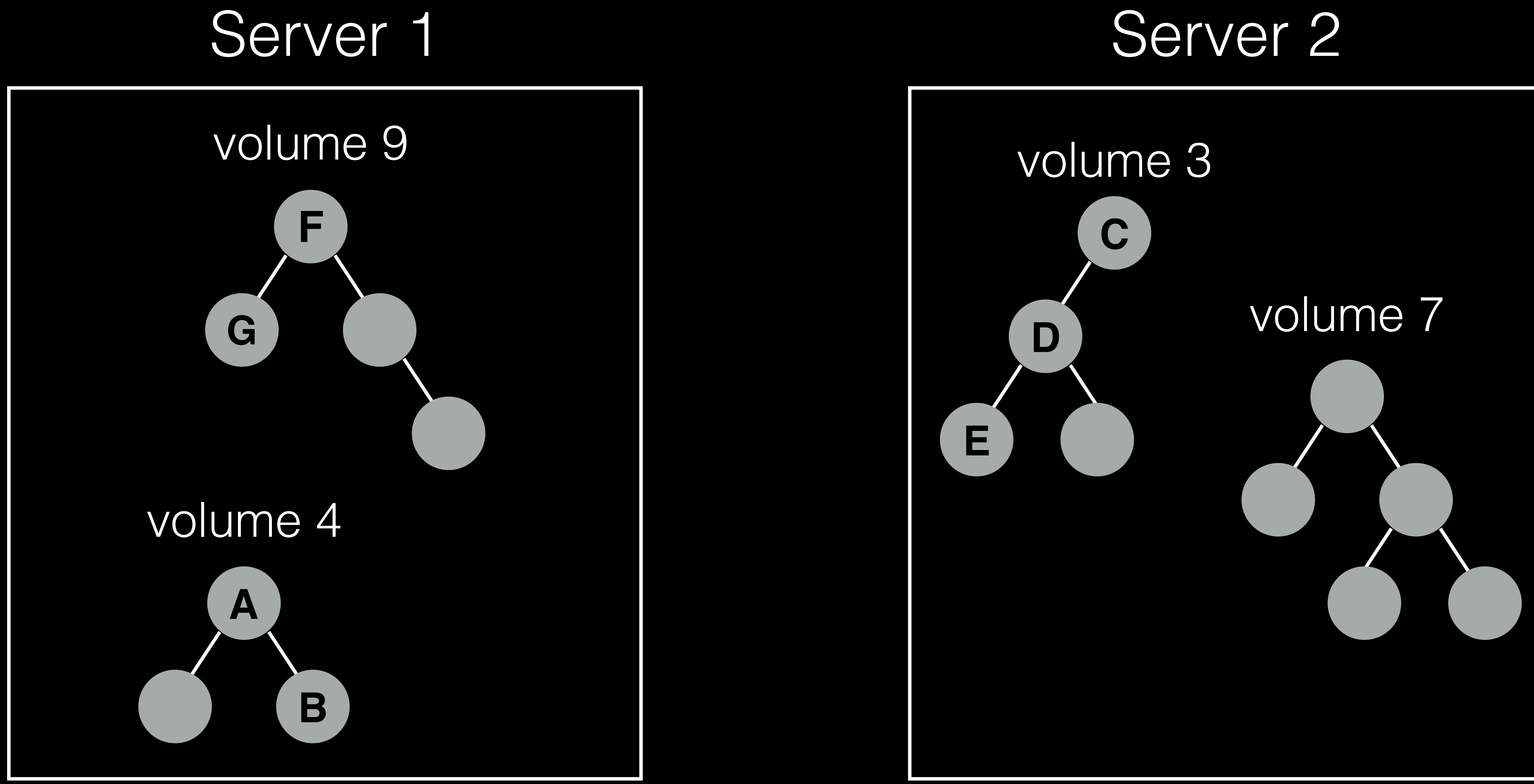
- Volume management
- Cache management
- Name resolution
- Process structure
- Local-storage API
- File locks.

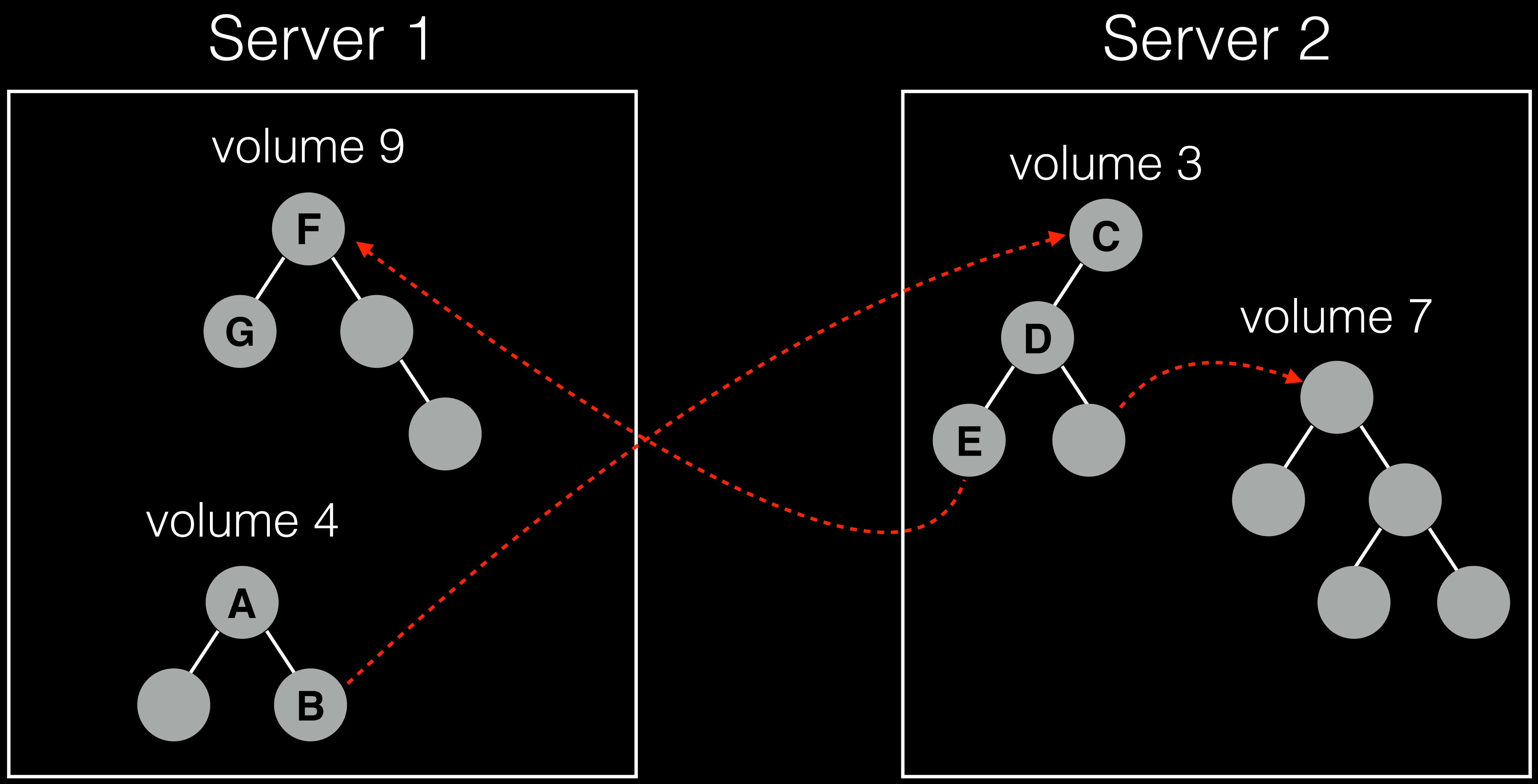
Volume Glue

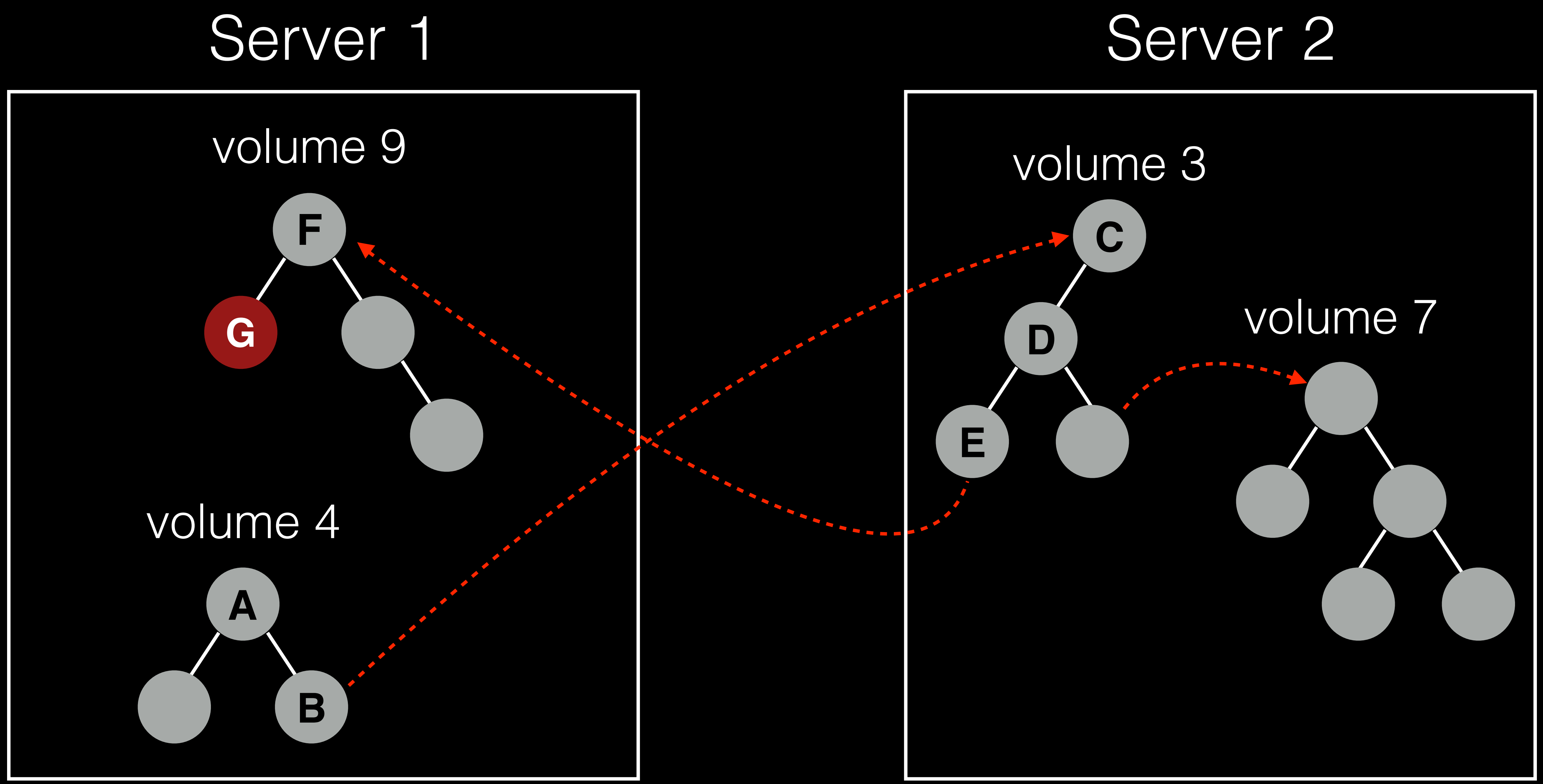
Volumes should be glued together into a seamless file tree.

Volume is a partial subtree.

Volume leaves may point to other volumes.


Server 1
volume 9
F
G
volume 4
A
B
Server 2
volume 3
C
D
E
volume 7


Server 1
Server 2
volume 9
F
G
volume 4
A
B
volume 3
C
D
E
volume 7



open A/B/C/D/E/F/G

Volume Database

Given a volume name, how do we know what machine stores it?

Maintain volume database mapping volume name to locations.

Replicate to every server.
 - clients can ask any server they please

Volume Movement

What if we want to migrate a volume to another machine?

Steps:

- copy data over
- update volume database

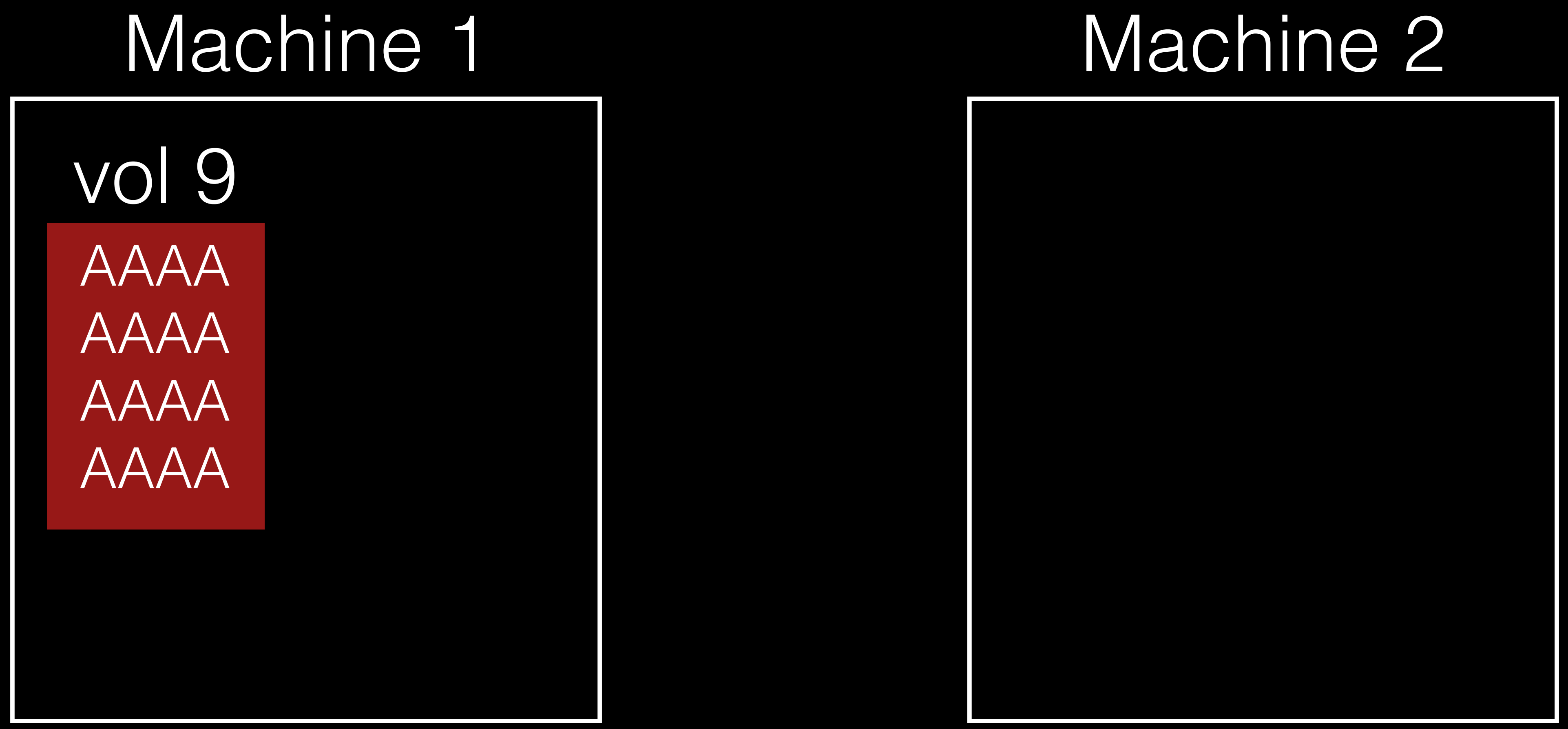
Volume Movement

What if we want to migrate a volume to another machine?

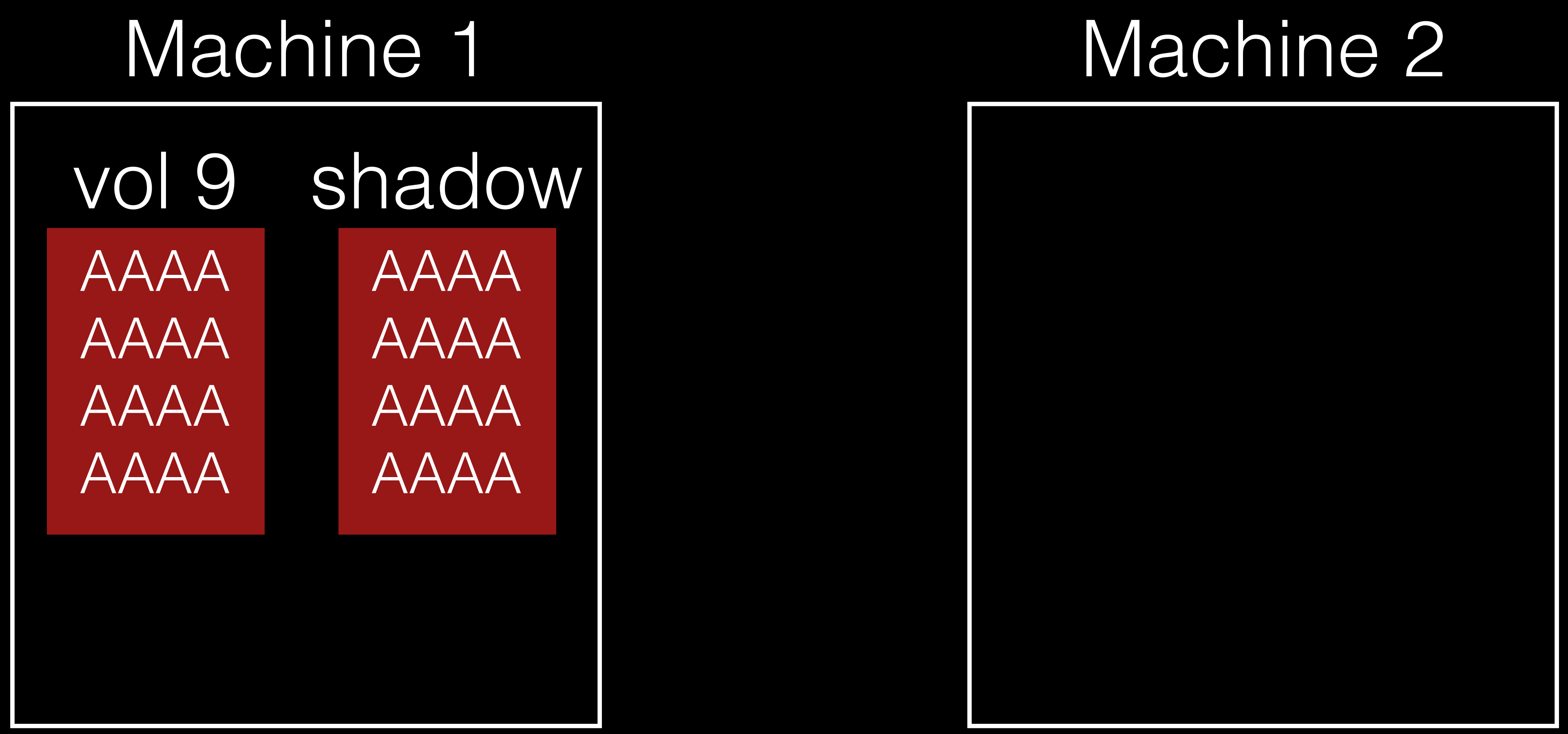
Steps:
- copy data over ← don't want to halt I/O during
- update volume database

What about updates during movement?

Copy

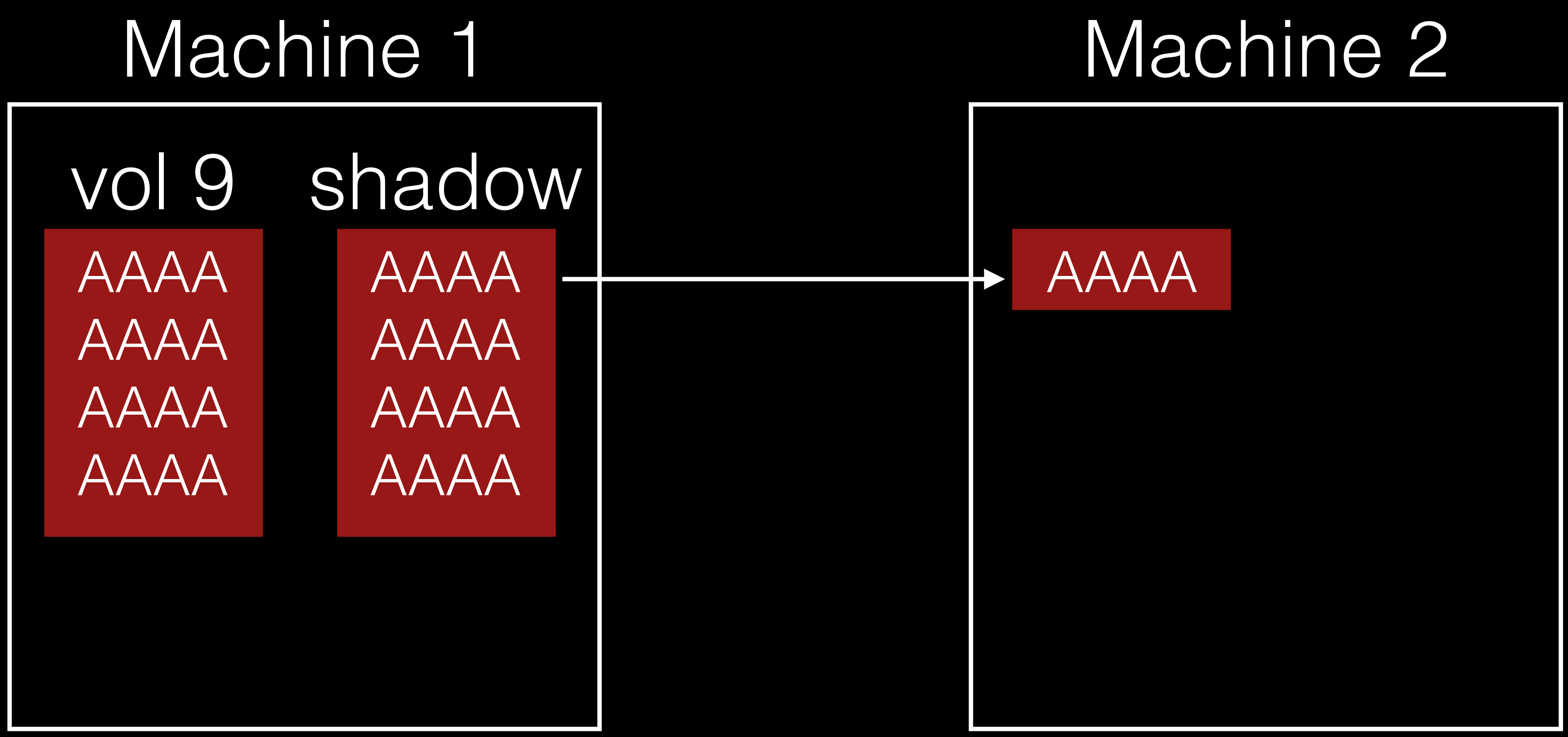


Copy

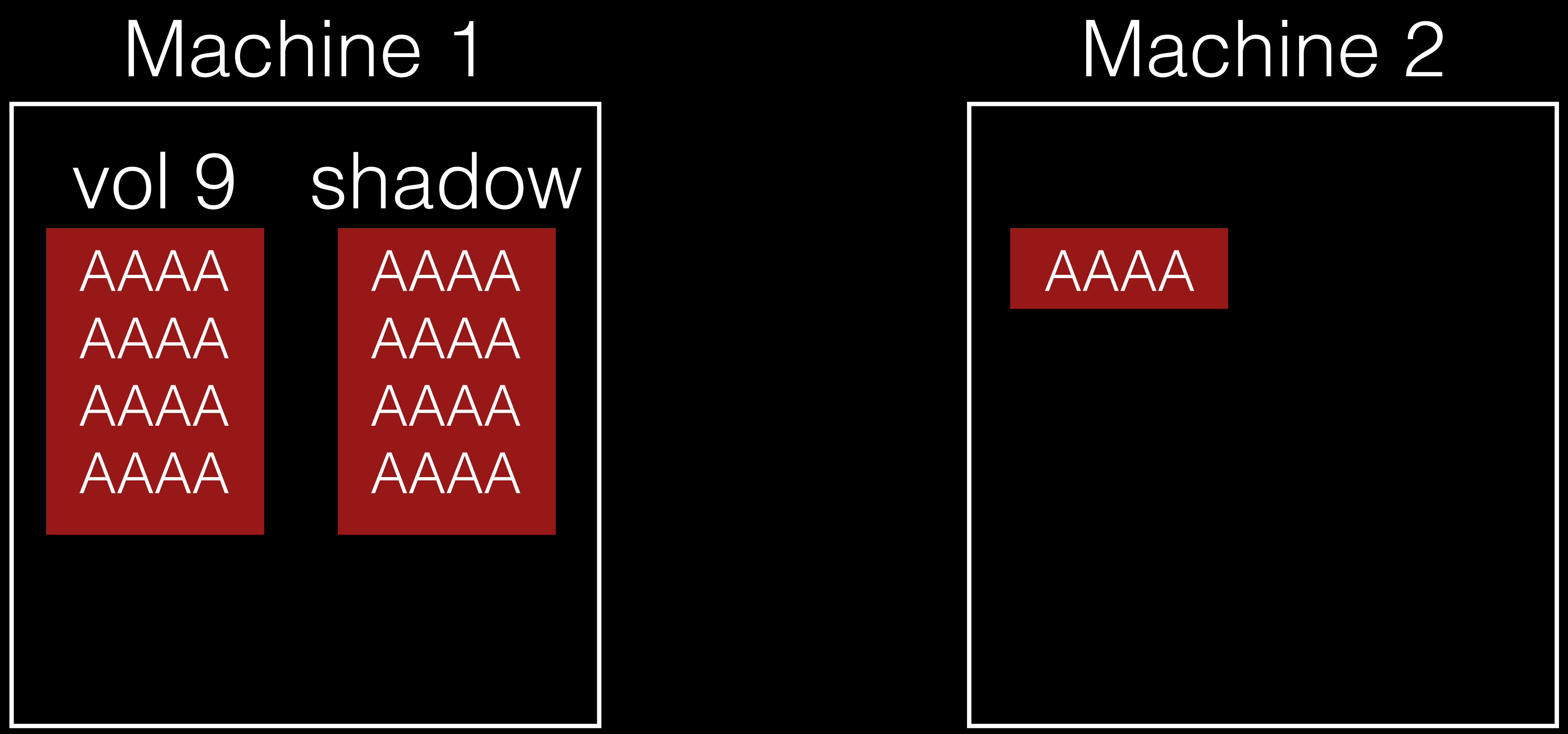


Machine 1
Machine 2
vol 9
shadow
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA

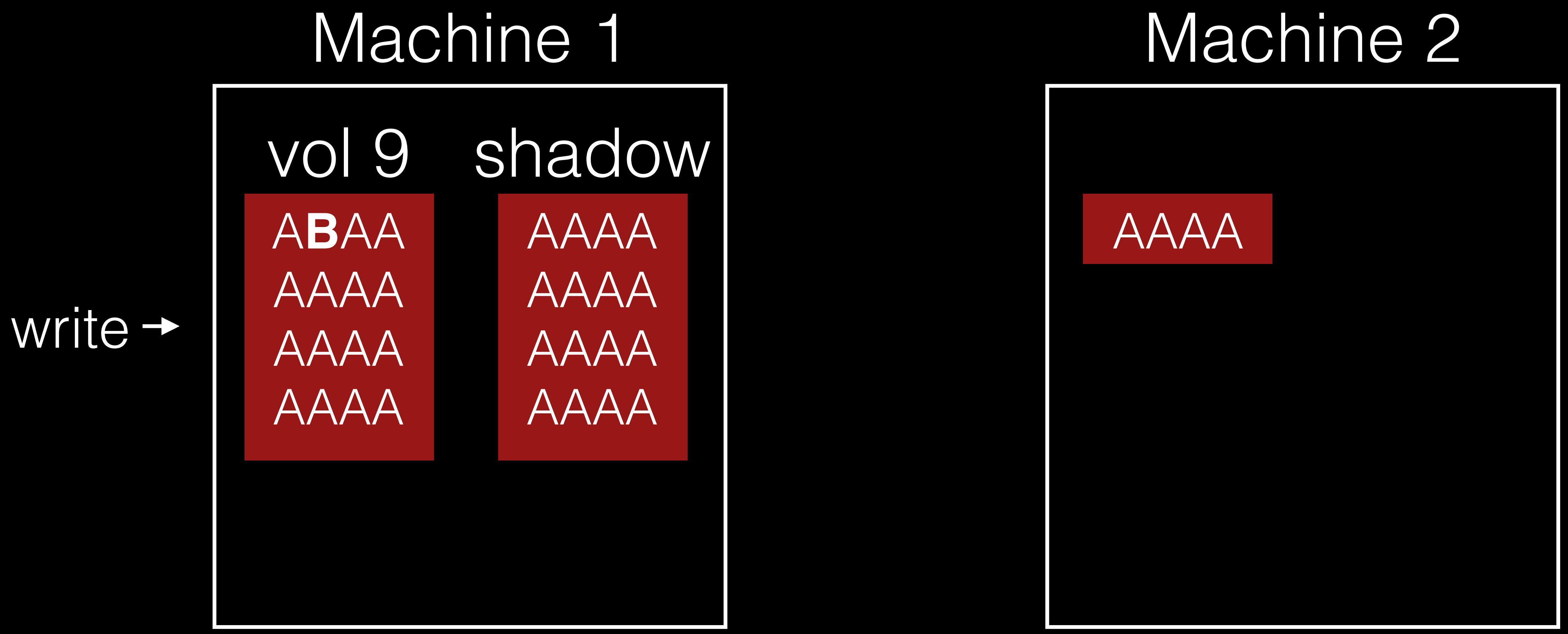
Copy


Machine 1
Machine 2
vol 9
shadow
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA

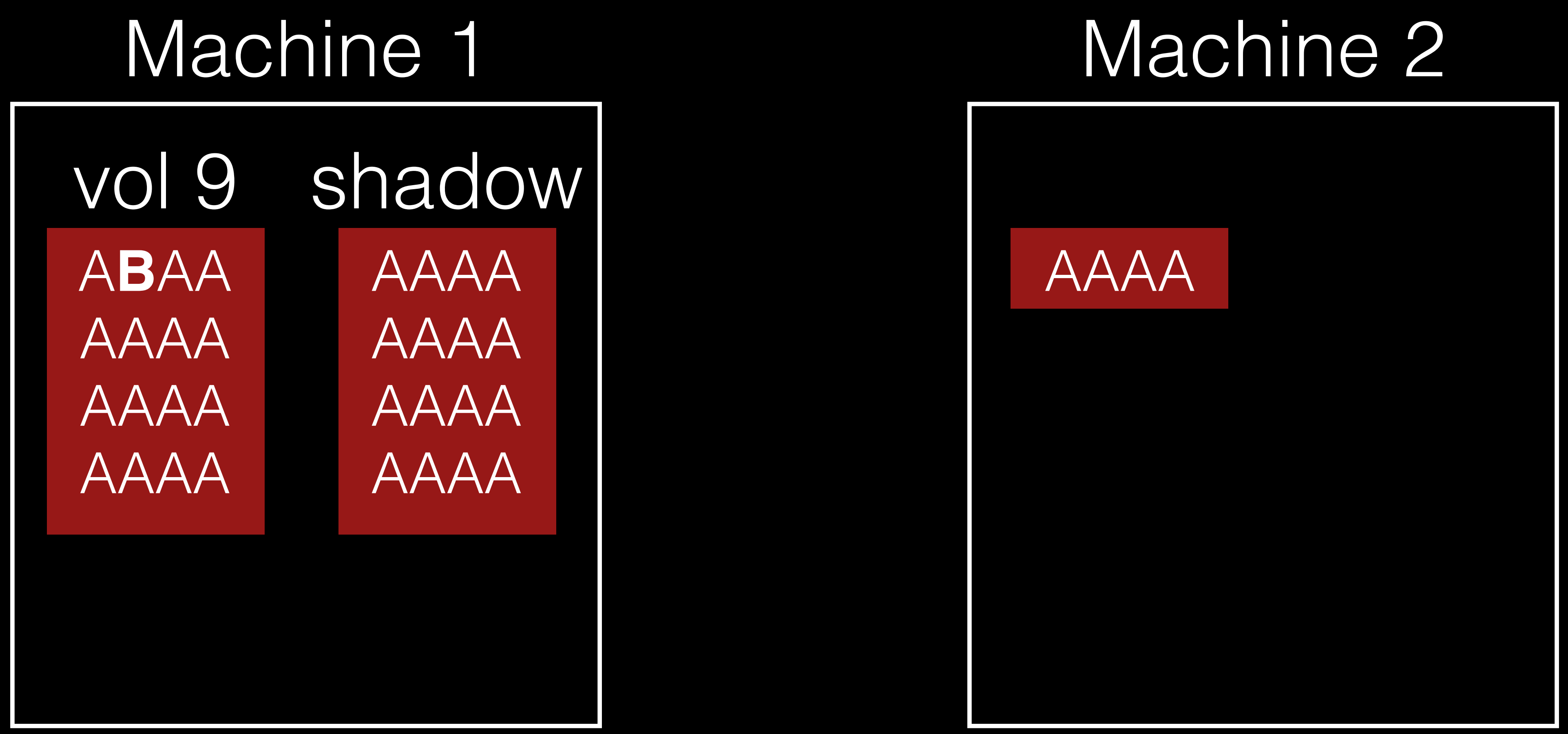
Copy


Machine 1
vol 9
shadow
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
Machine 2
AAAA

Copy

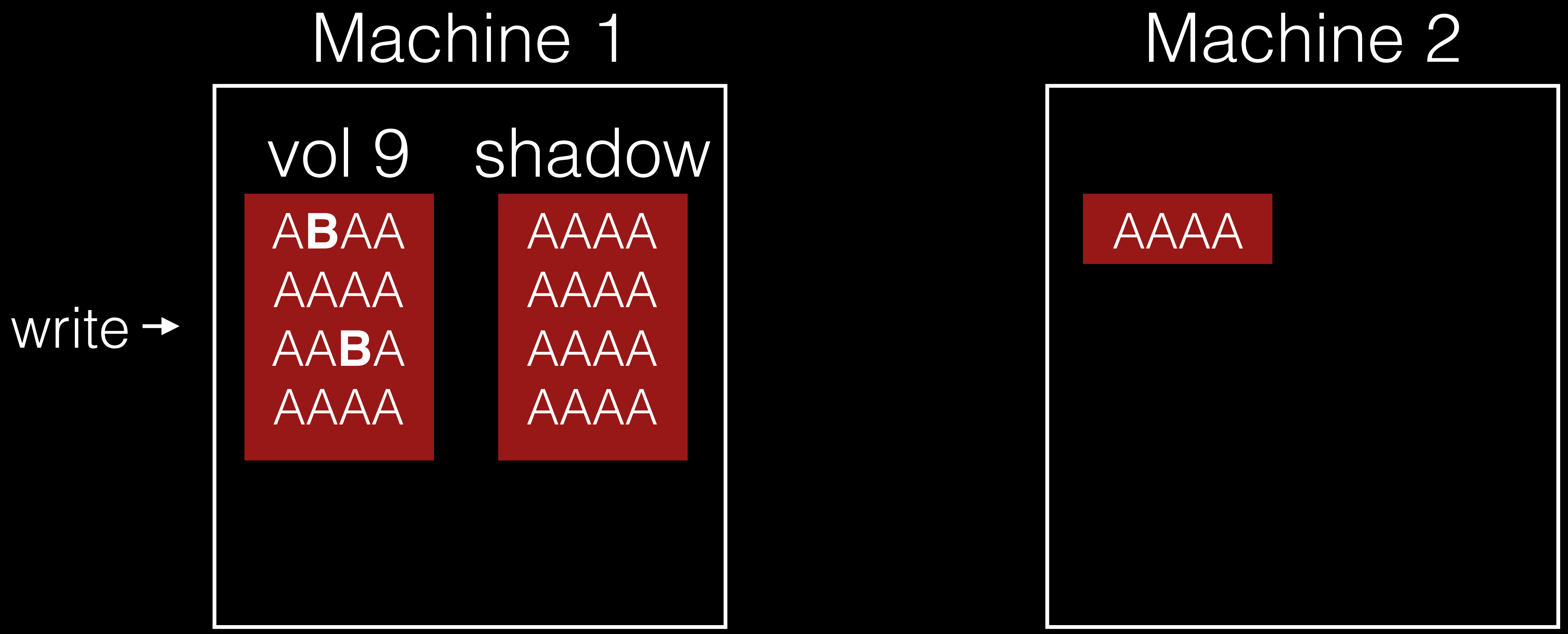

Machine 1
Machine 2
vol 9
shadow
ABAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
write
AAAA

Copy

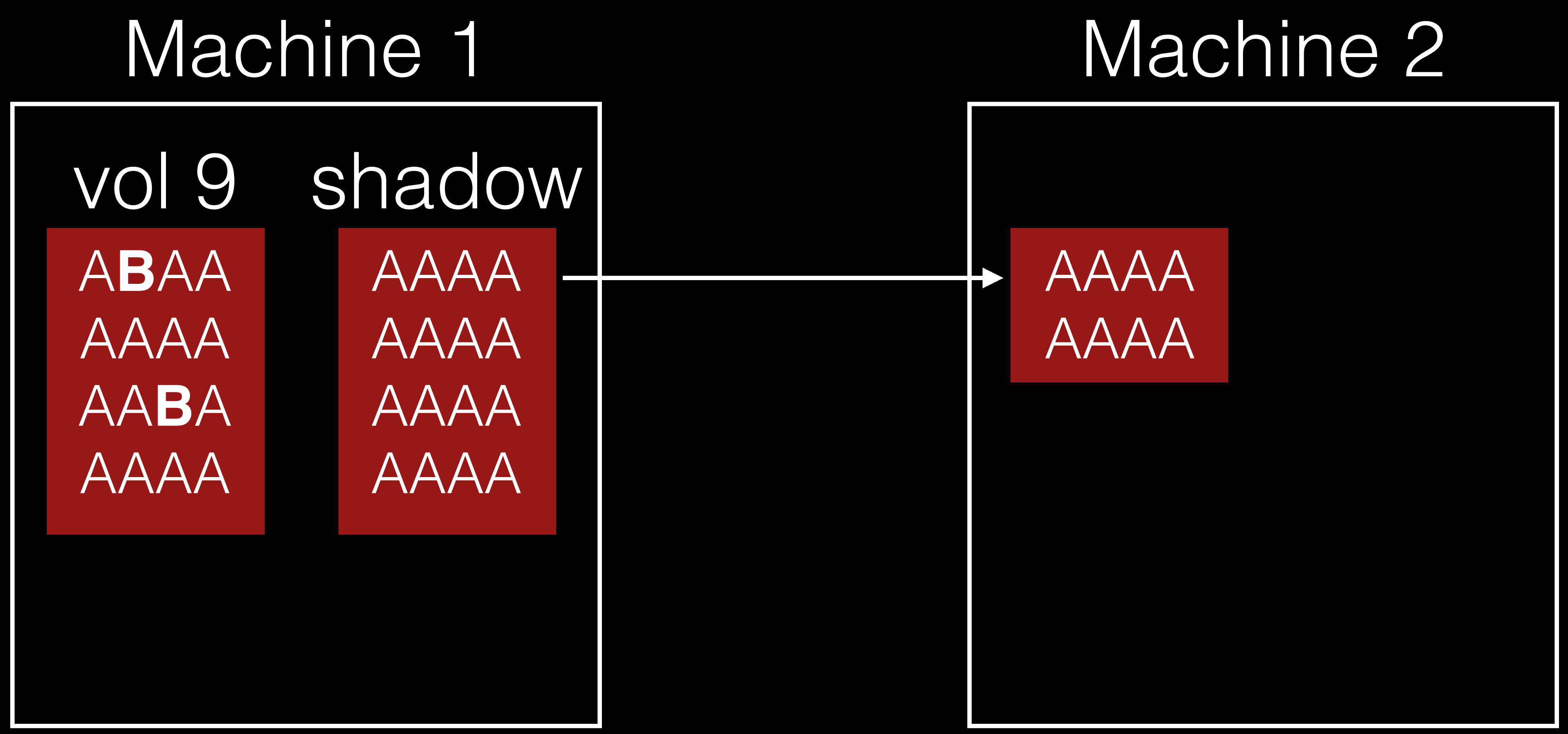


Machine 1
Machine 2
vol 9
shadow
ABAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA

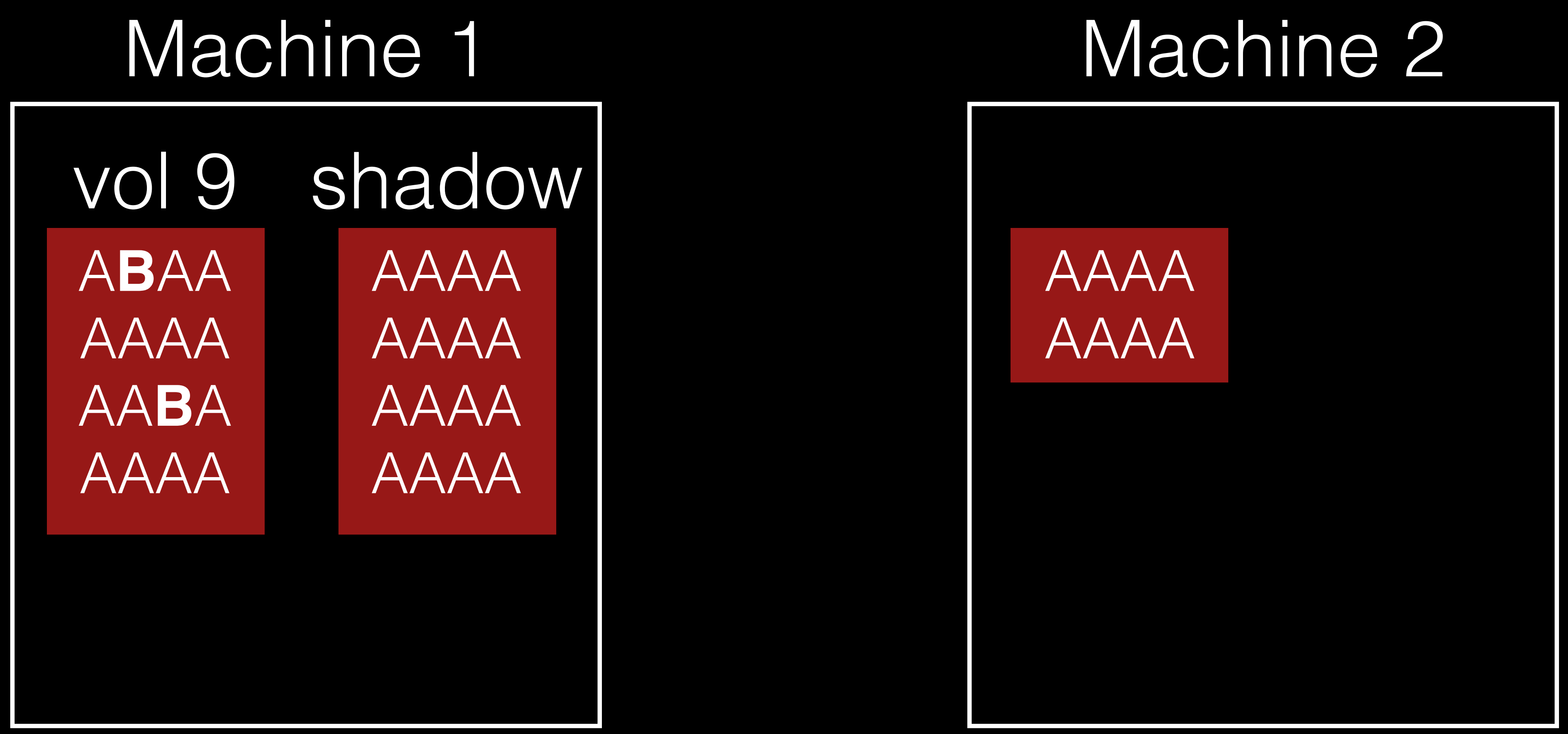
Copy


Machine 1
Machine 2
vol 9
shadow
write
ABAA
AAAA
AABA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA

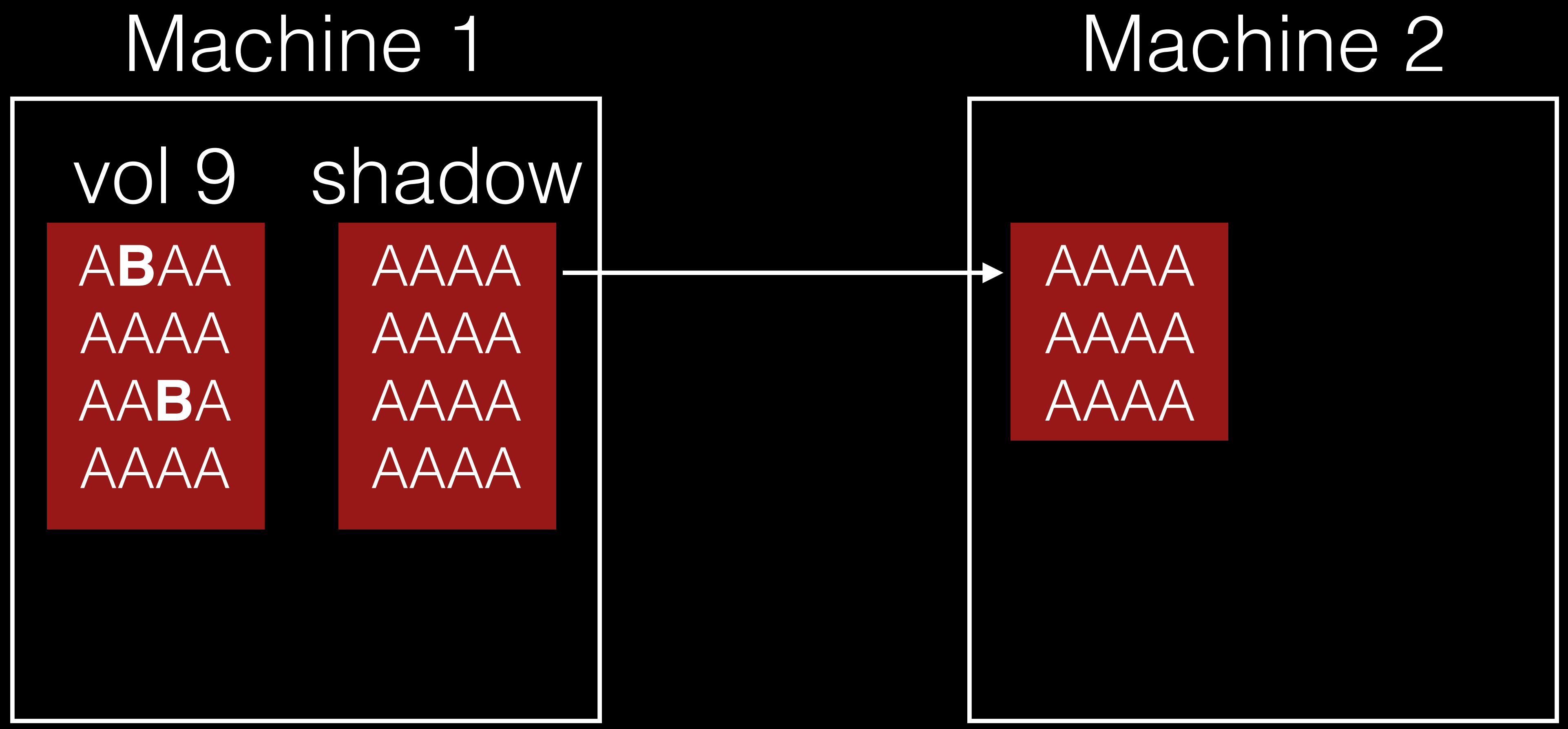
Copy


Machine 1
Machine 2
vol 9
shadow
ABAA
AAAA
AABA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA

Copy

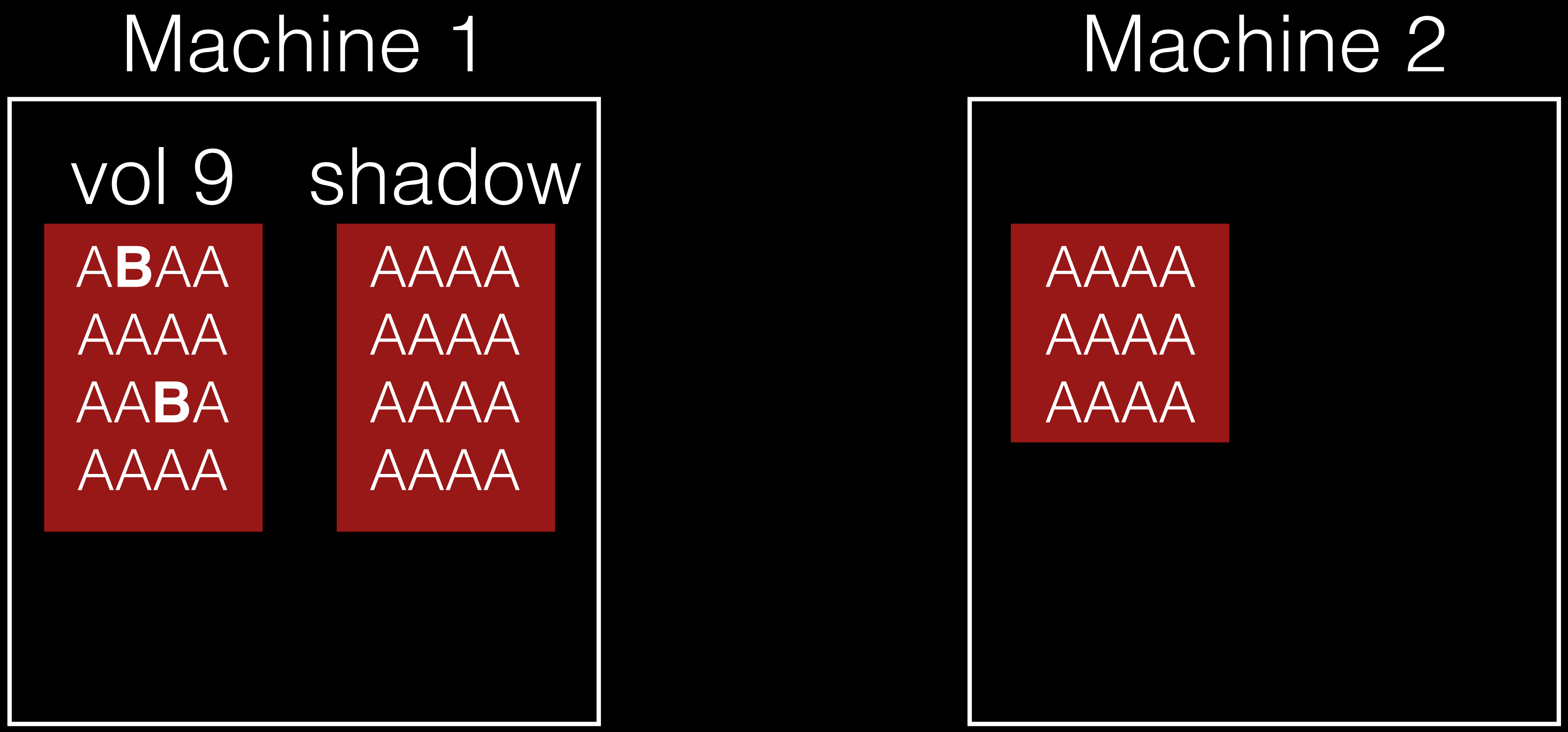

Machine 1
vol 9
shadow
ABAA
AAAA
AABA
AAAA
AAAA
AAAA
AAAA
AAAA
Machine 2
AAAA
AAAA

Copy

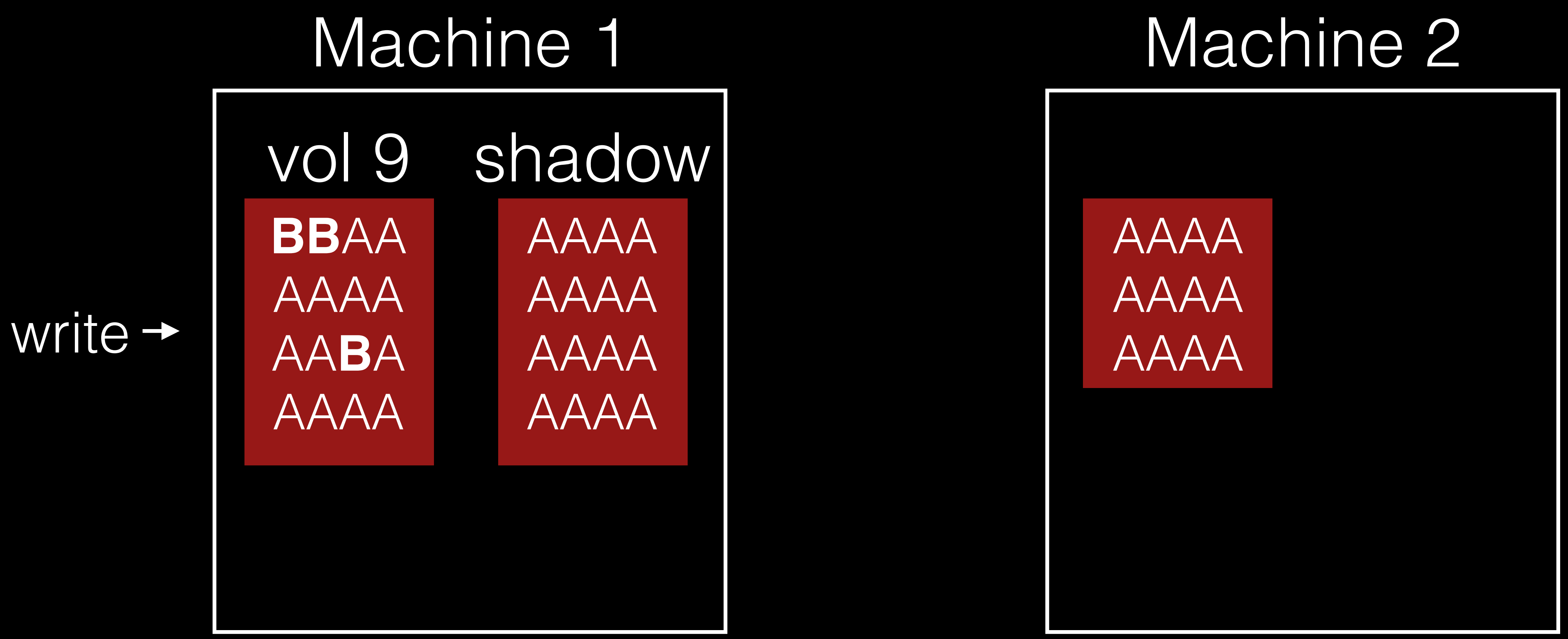


Machine 1
Machine 2
vol 9
shadow
ABAA
AAAA
AABA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA

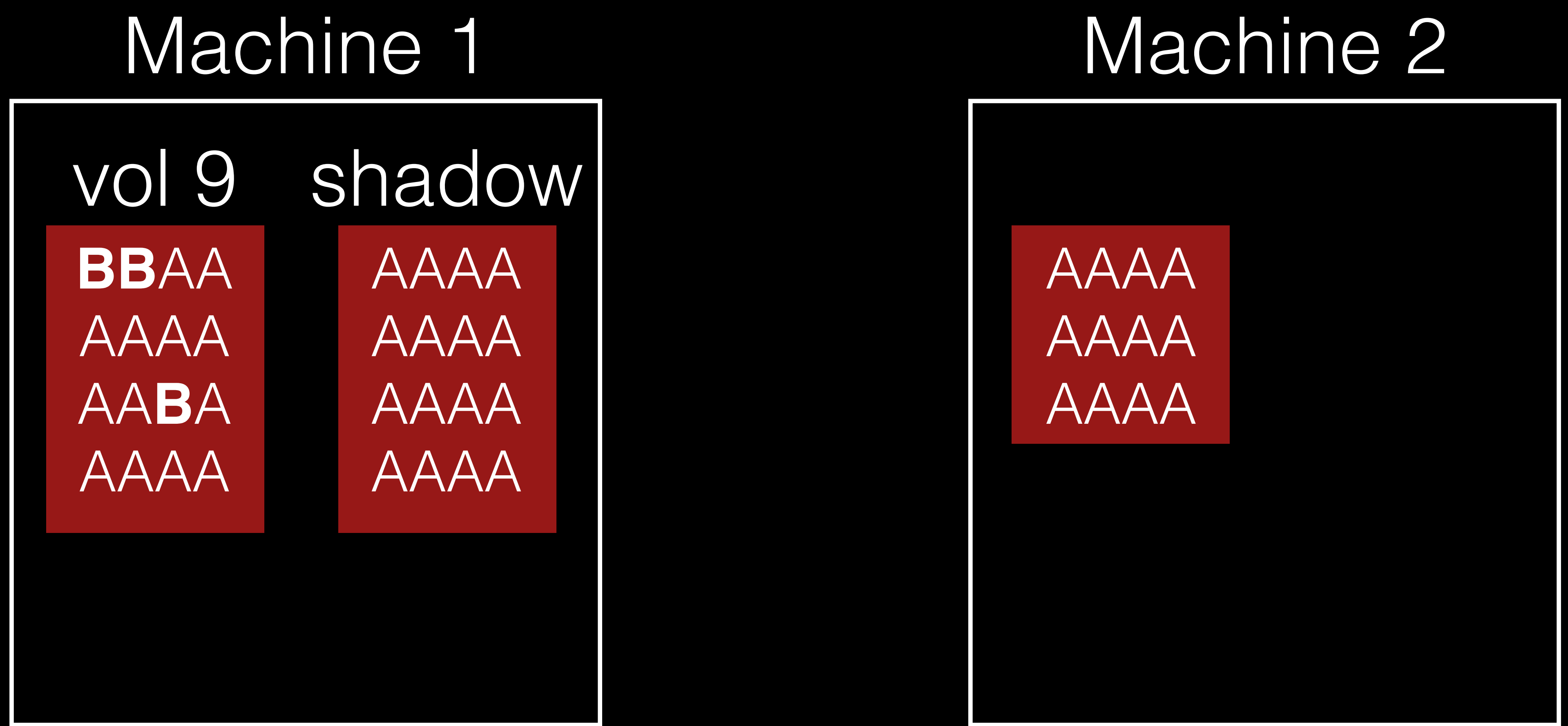
Copy


Machine 1
Machine 2
vol 9
shadow
ABAA
AAAA
AABA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA

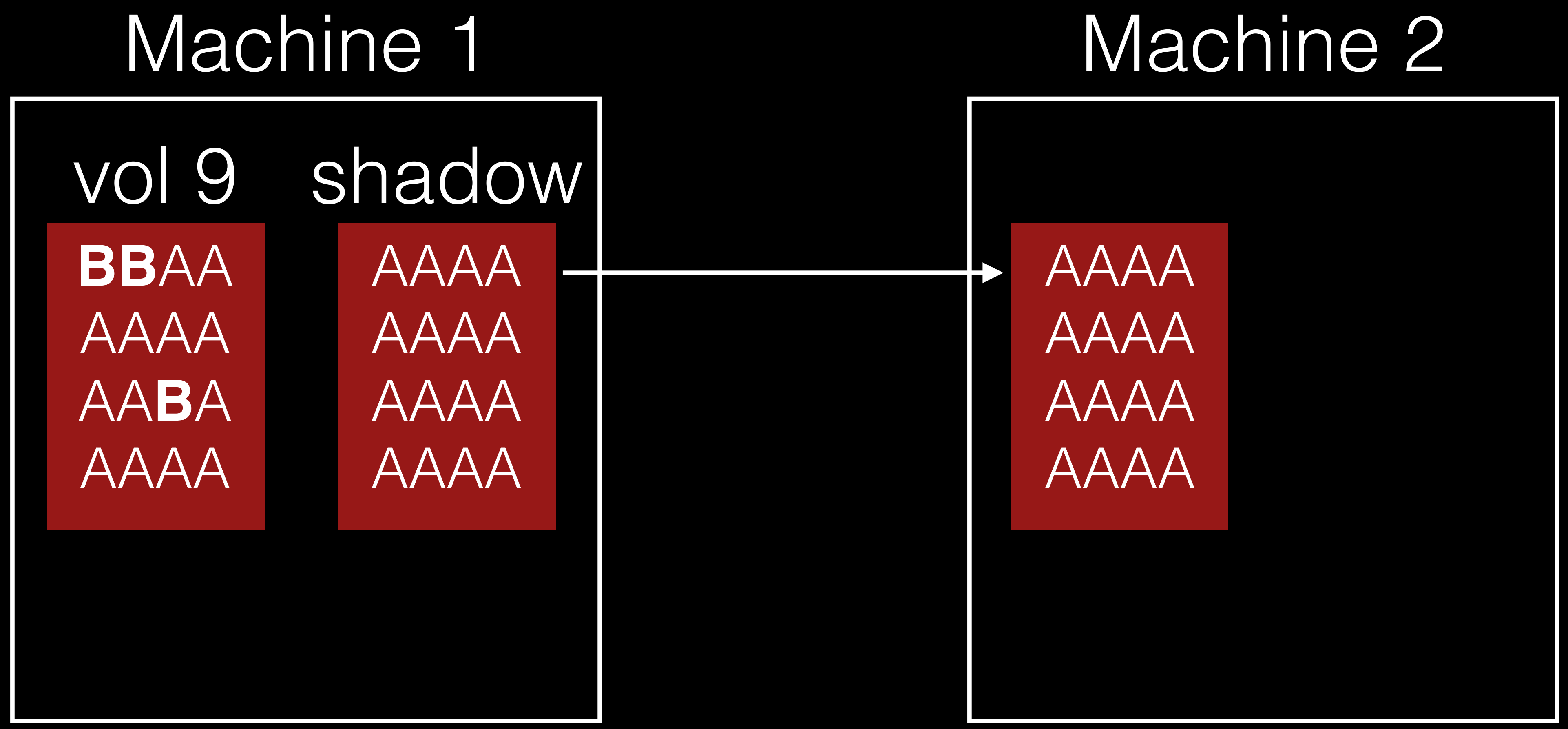
Copy


Machine 1
Machine 2
vol 9
shadow
write
BBAA
AAAA
AABA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA

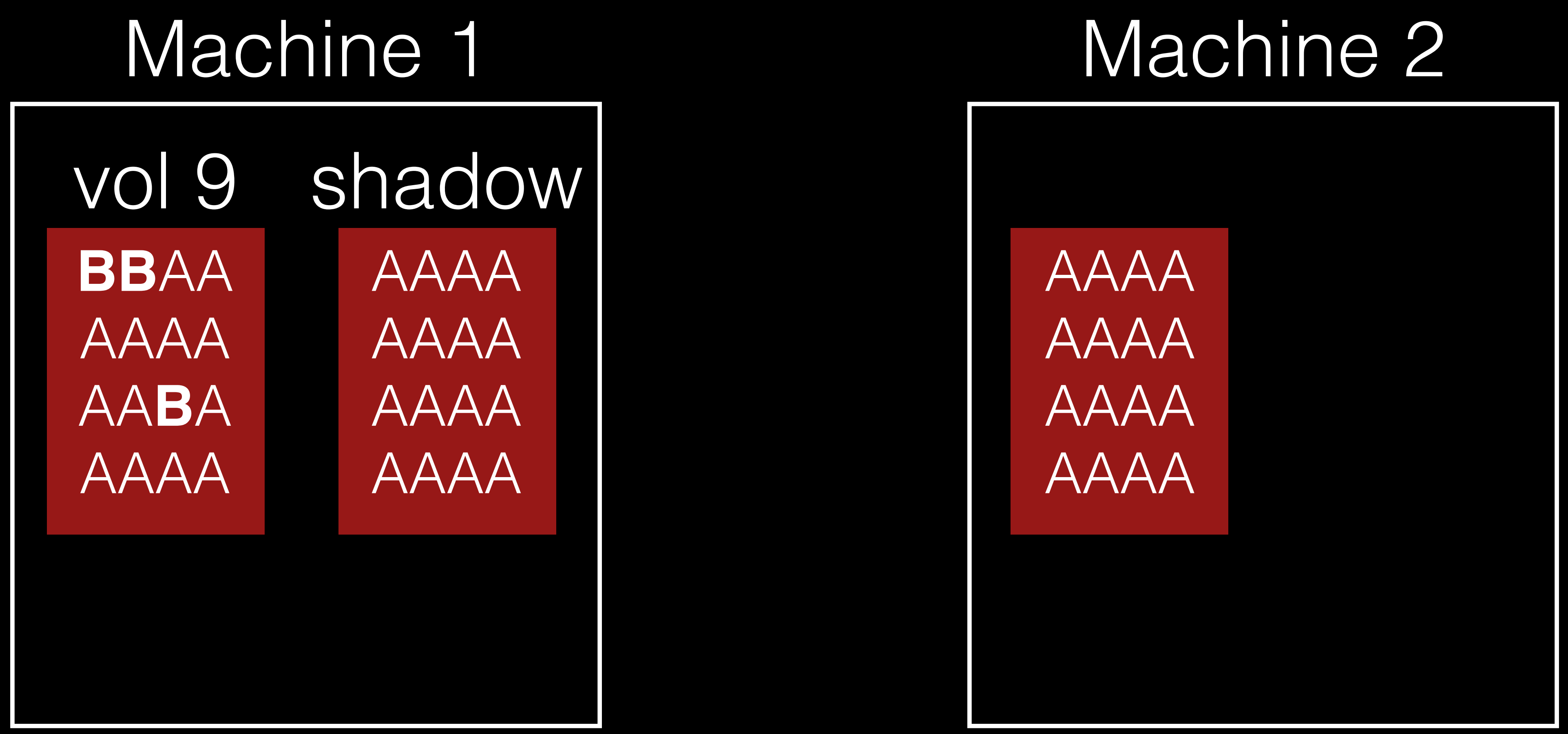
Copy


Machine 1
vol 9
shadow
BBAA
AAAA
AABA
AAAA
AAAA
AAAA
AAAA
AAAA
Machine 2
AAAA
AAAA
AAAA

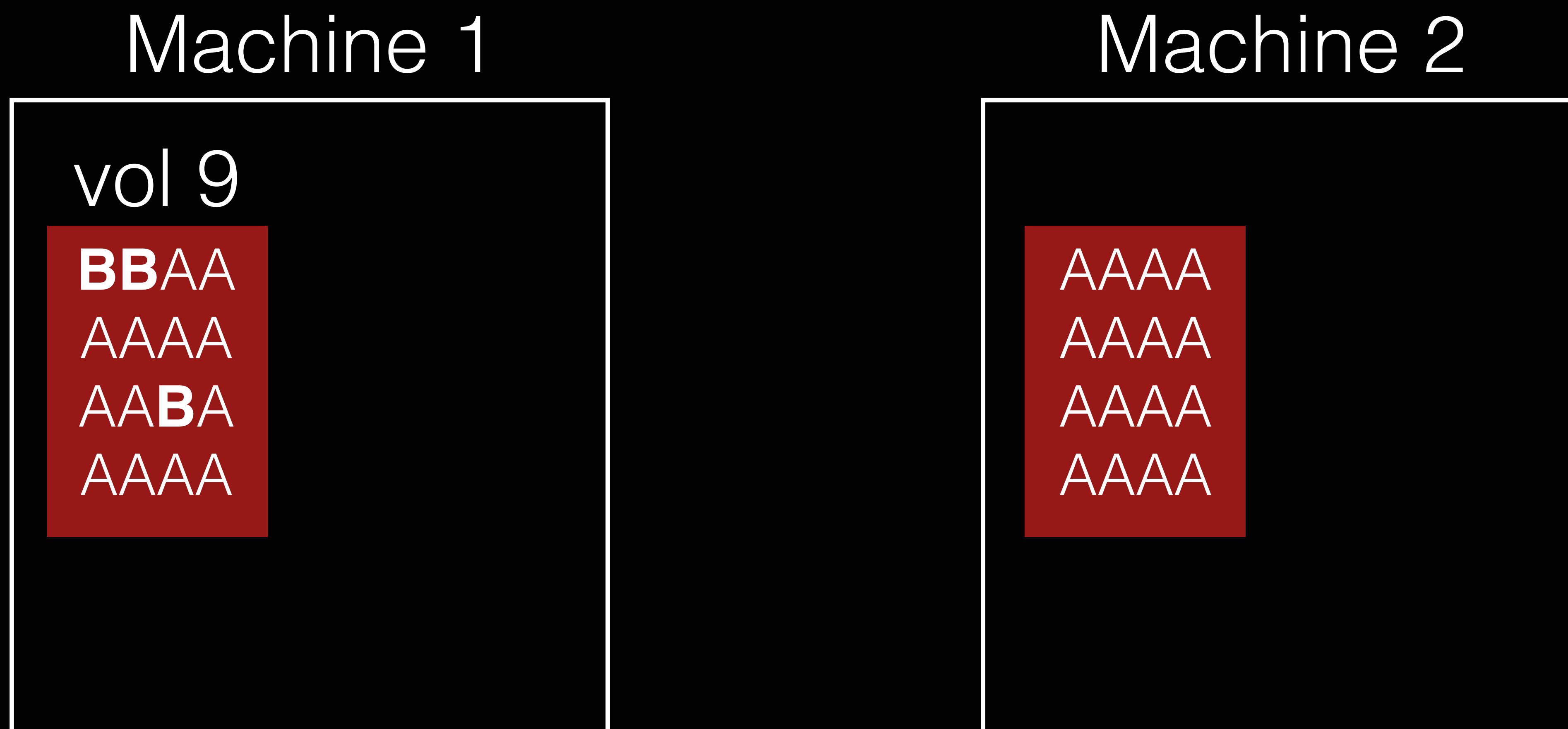
Copy


Machine 1
Machine 2
vol 9
shadow
BBAA
AAAA
AABA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA

Copy

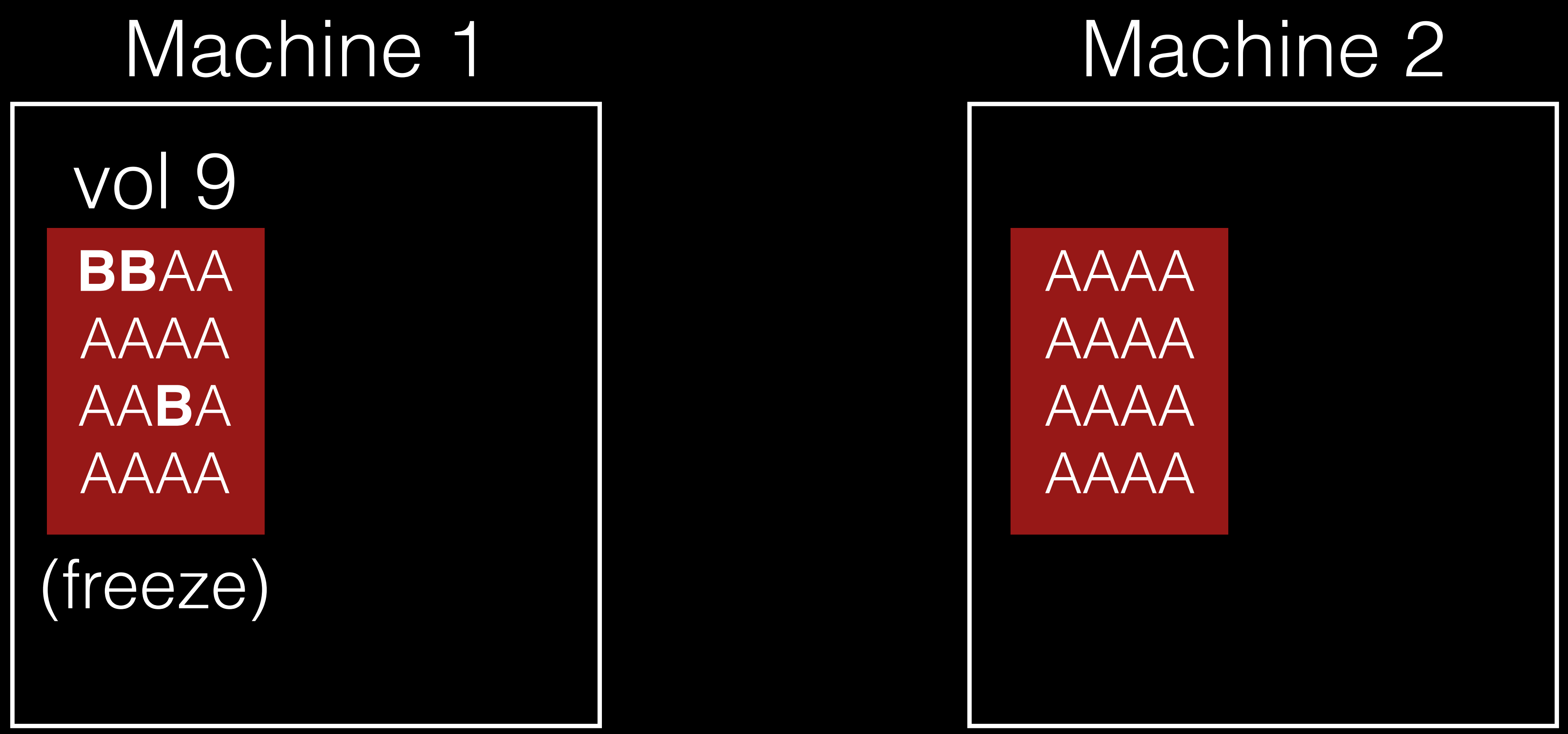

Machine 1
Machine 2
vol 9
shadow
BBAA
AAAA
AABA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA

Copy



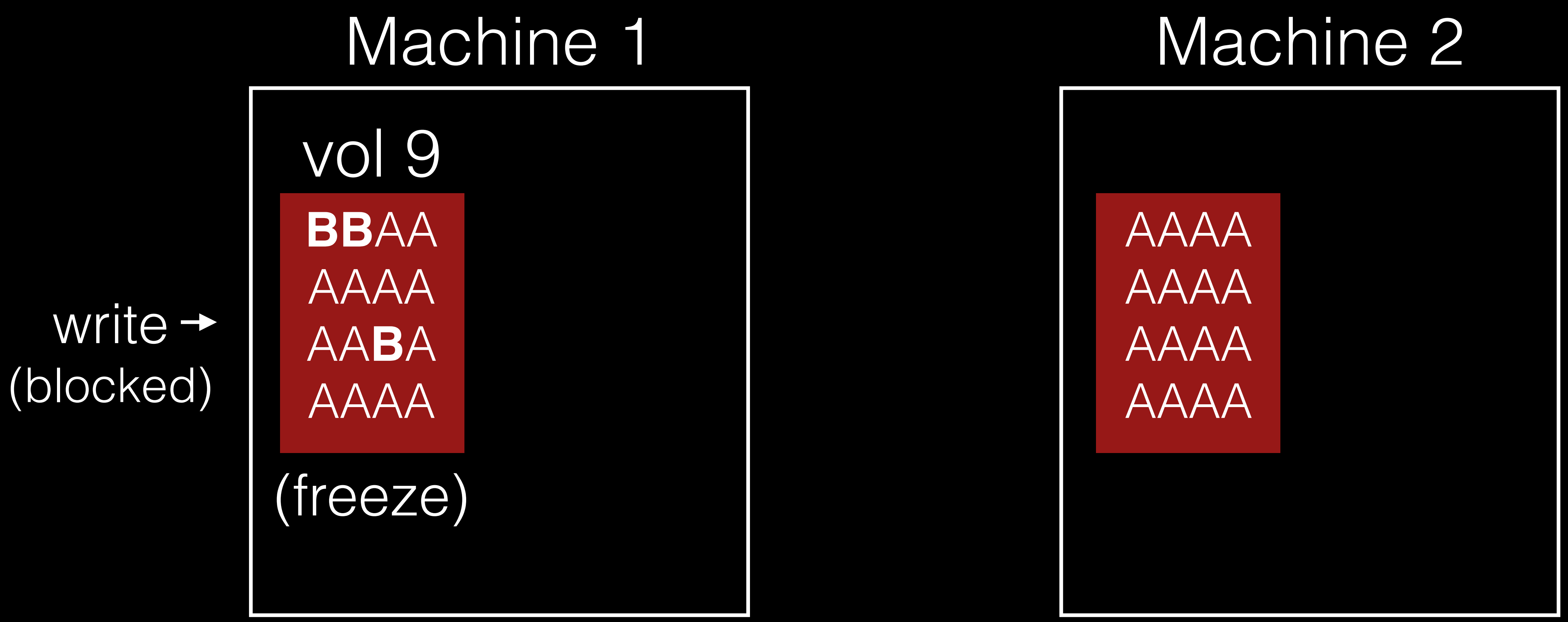
Machine 1
vol 9
BBAA
AAAA
AA**B**A
AAAA
Machine 2
AAAA
AAAA
AAAA
AAAA

Copy

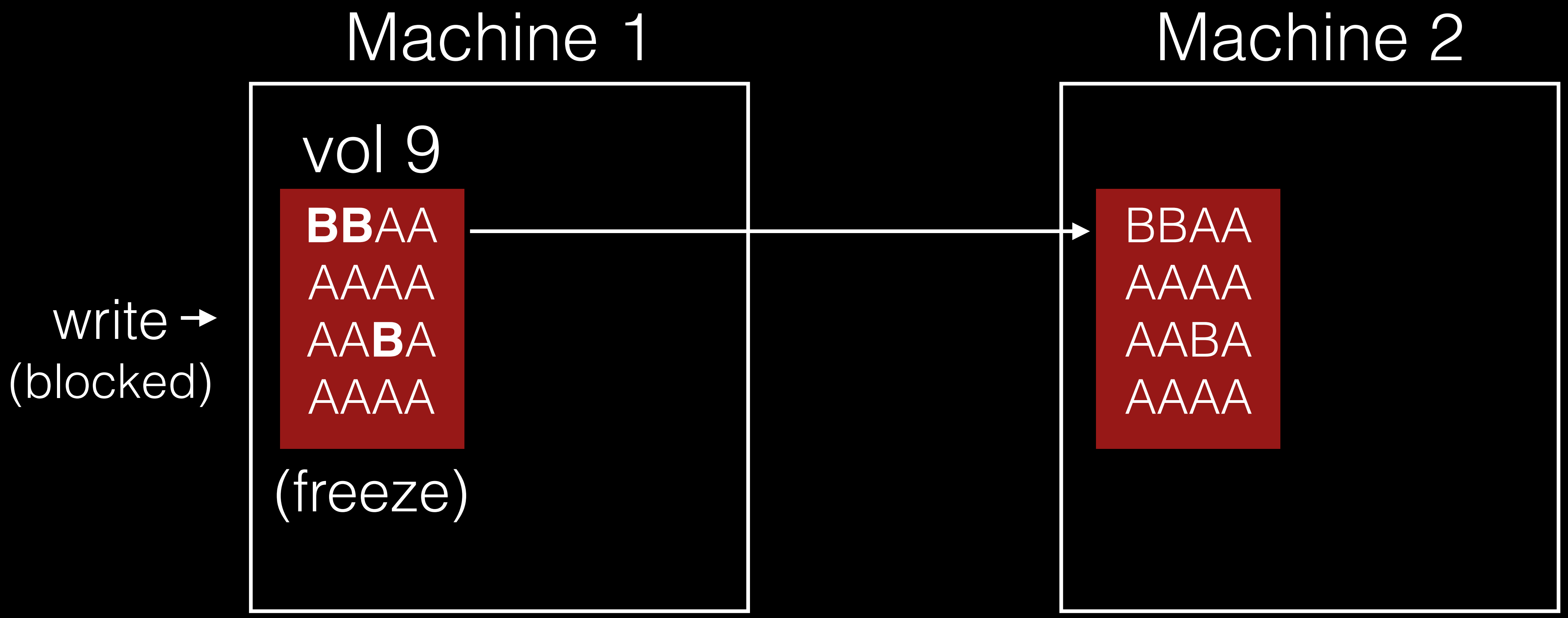


Machine 1
vol 9
BBAA
AAAA
AA**B**A
AAAA
(freeze)
Machine 2
AAAA
AAAA
AAAA
AAAA

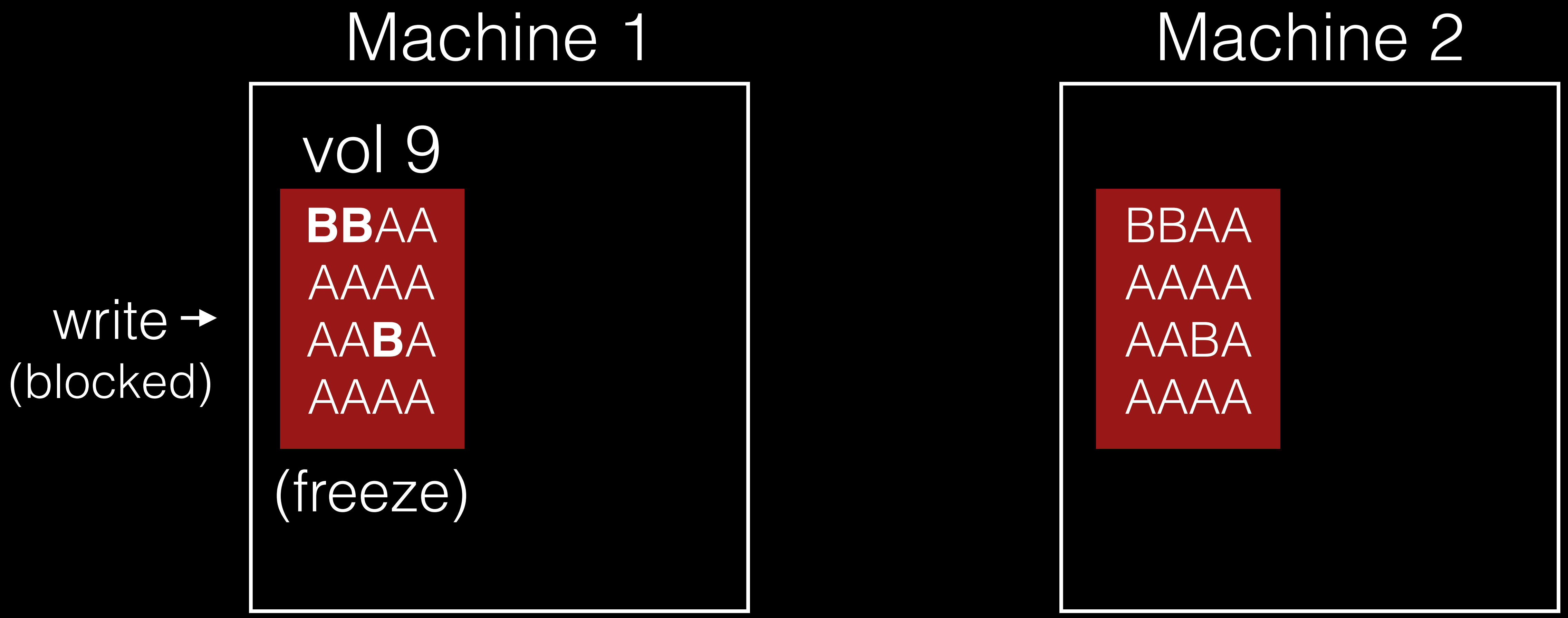
Copy


Machine 1
vol 9
BBAA
AAAA
AABA
AAAA
(freeze)
write →
(blocked)
Machine 2
AAAA
AAAA
AAAA
AAAA

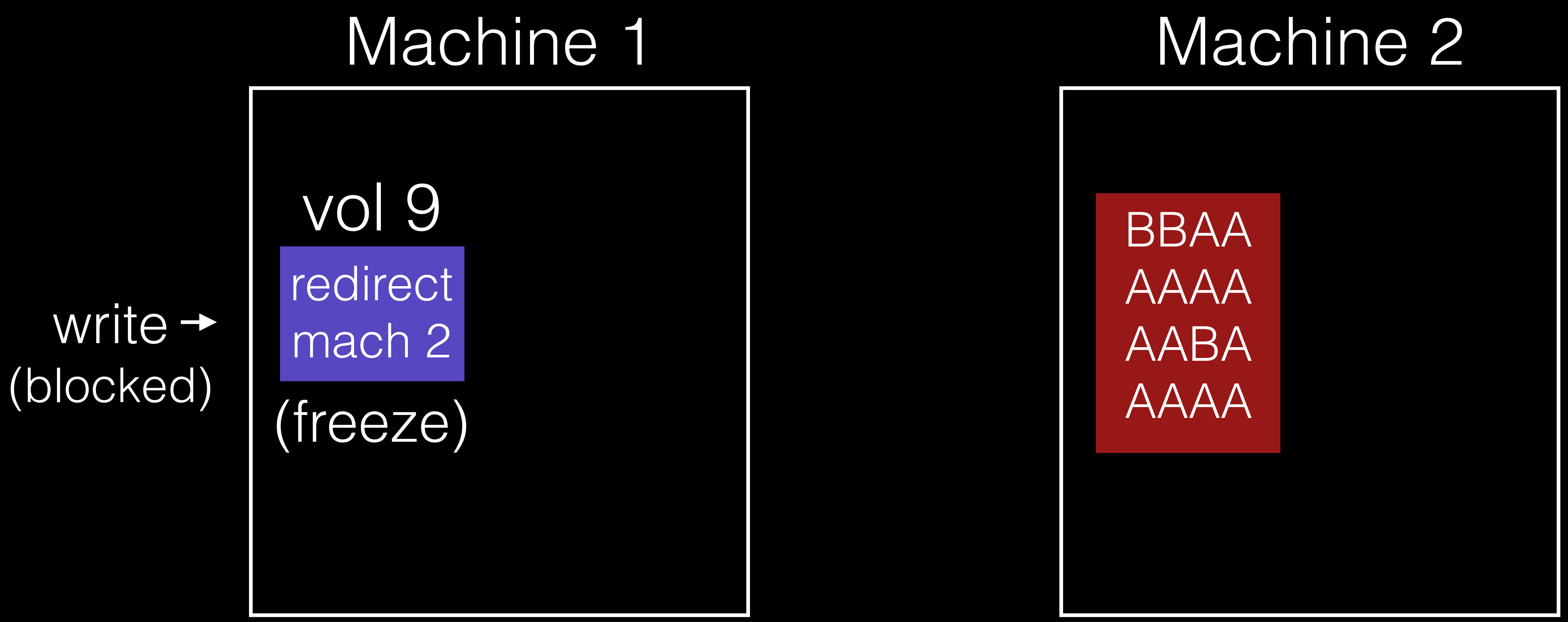
Copy


Machine 1
Machine 2
vol 9
BBAA
AAAA
AA**B**A
AAAA
(freeze)
write
(blocked)
BBAA
AAAA
AABA
AAAA

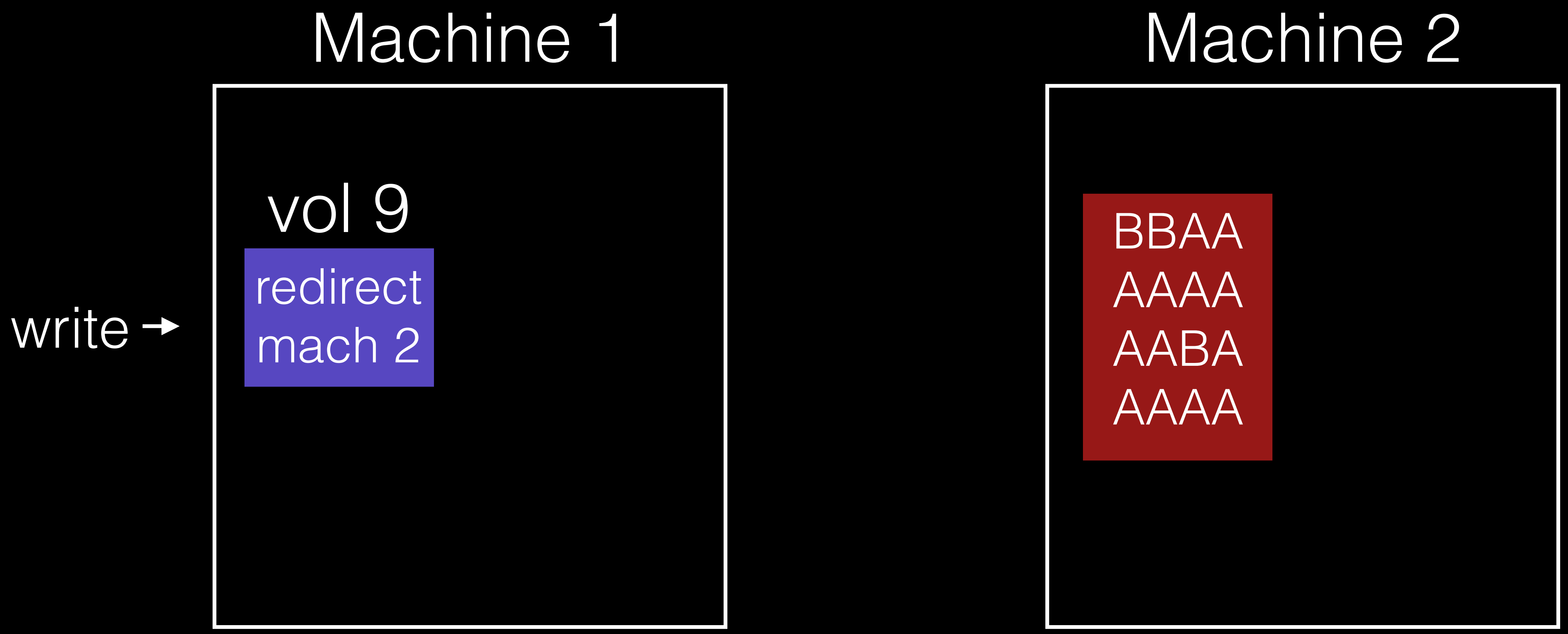
Copy


Machine 1
vol 9
BBAA
AAAA
AABA
AAAA
(freeze)
write
(blocked)
Machine 2
BBAA
AAAA
AABA
AAAA

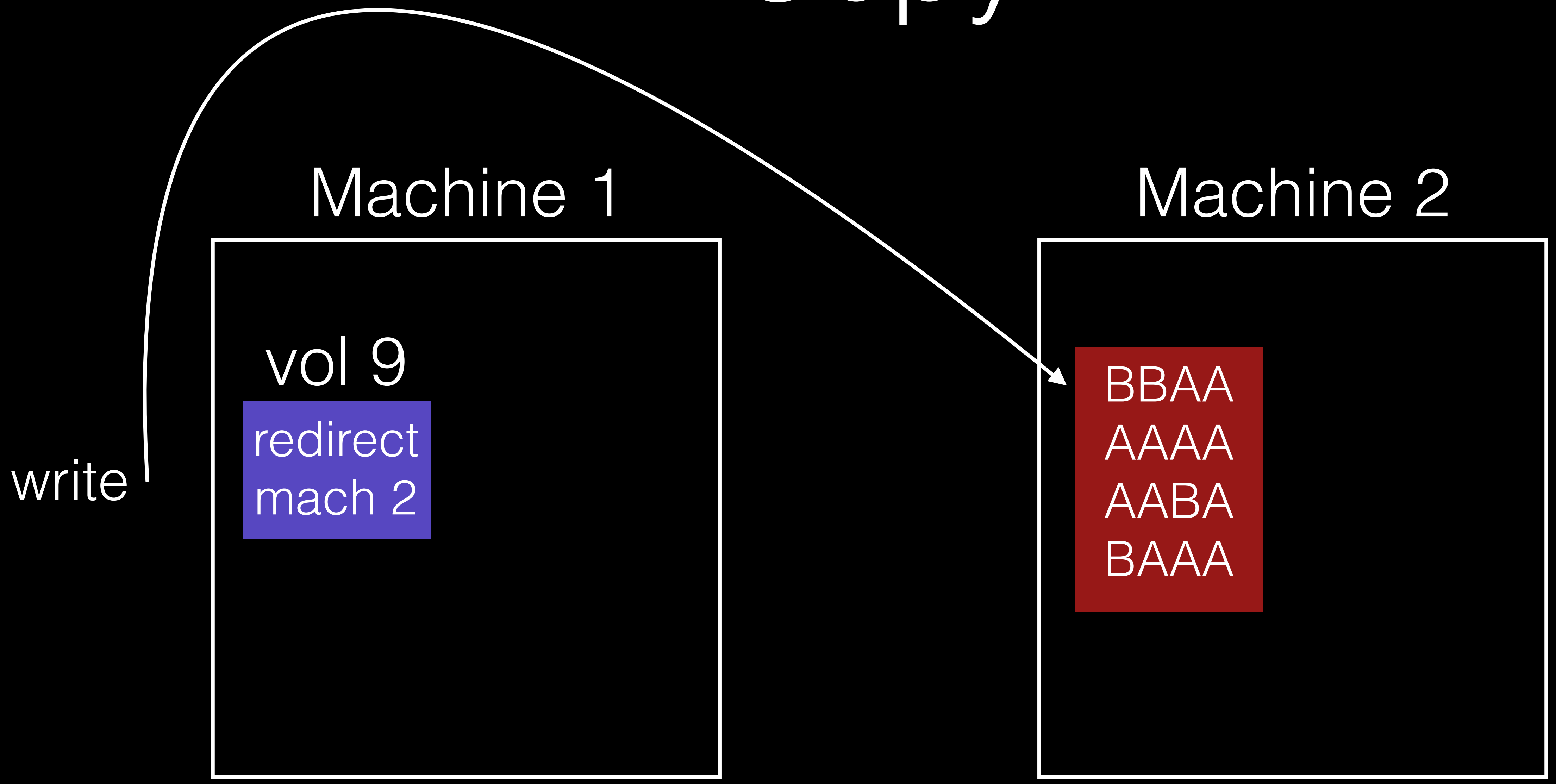
Copy


Machine 1
Machine 2
vol 9
redirect
mach 2
(freeze)
write
(blocked)
BBAA
AAAA
AABA
AAAA

Copy

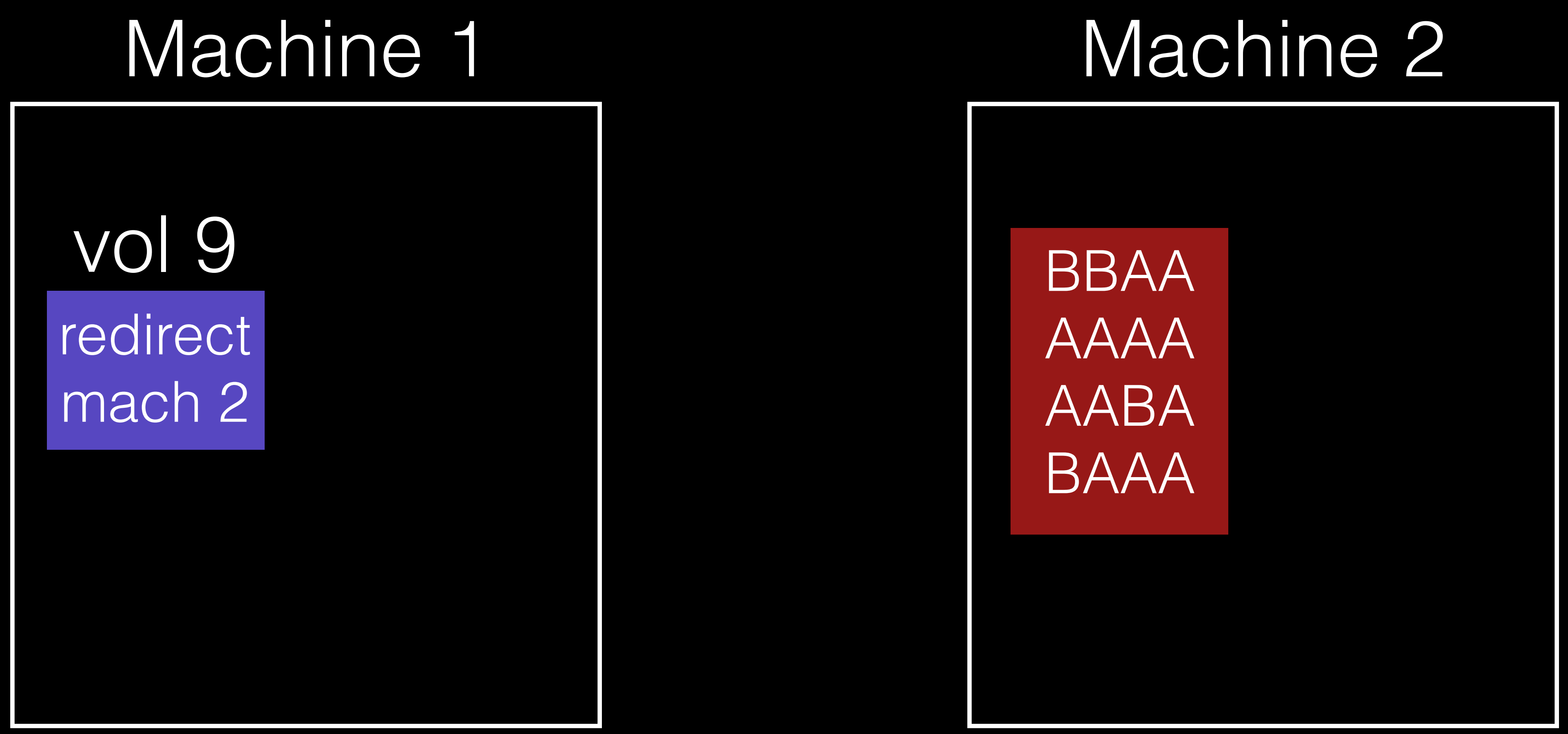

Machine 1
Machine 2
vol 9
write
redirect
mach 2
BBAA
AAAA
AABA
AAAA

Copy



write
Machine 1
vol 9
redirect
mach 2
Machine 2
BBAA
AAAA
AABA
BAAA

Copy



Volume Movement

What if we want to migrate a volume to another machine?

Steps:
- copy data over ← don’t want to halt I/O during
- update volume database

What about updates during movement?

Volume Movement

What if we want to migrate a volume to another machine?

Steps:
- copy data over
- update volume database ← what if somebody reads stale?

What about updates during movement?

Volume Movement

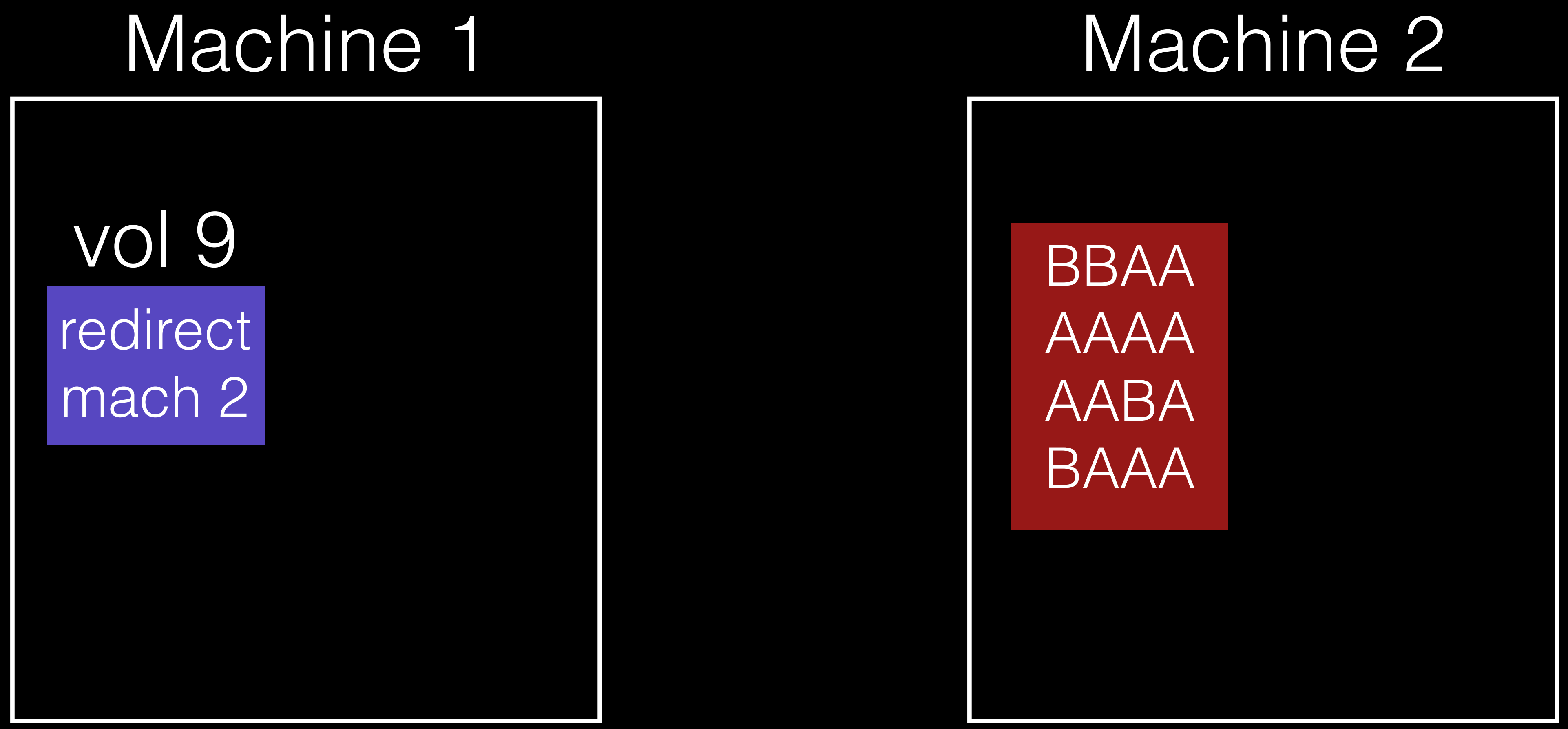
What if we want to migrate a volume to another machine?

Steps:

- copy data over
- update volume database ← what if somebody reads stale? keep forwarding note at old location until all replicas updated

What about updates during movement?

Copy



Outline

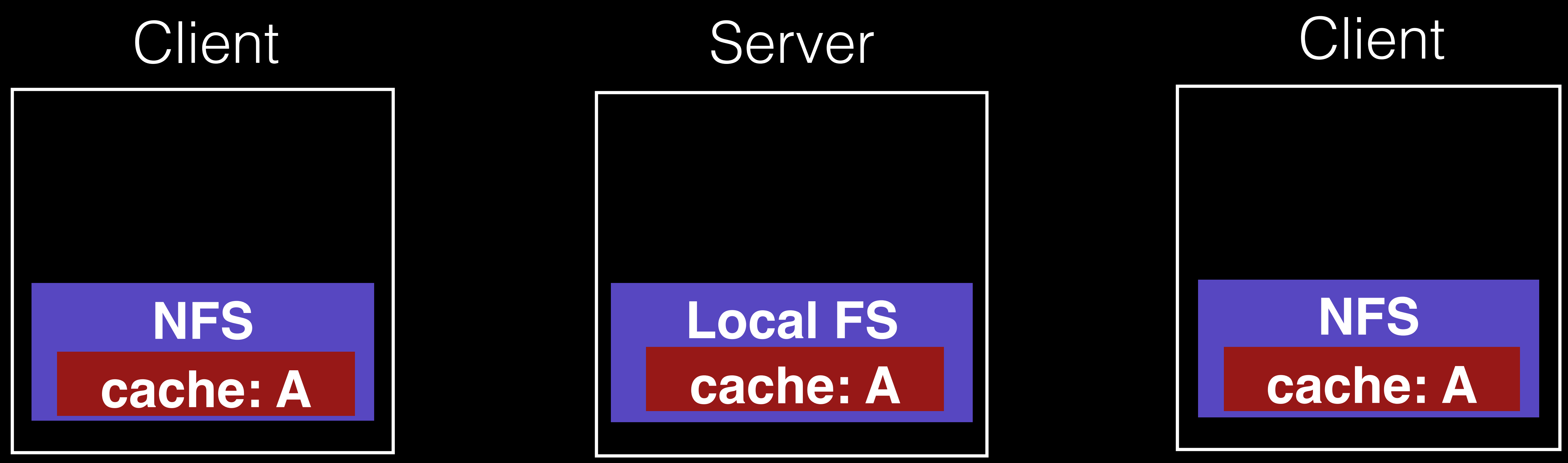
- Volume management
- Cache management
- Name resolution
- Process structure
- Local-storage API
- File locks

Cache Consistency

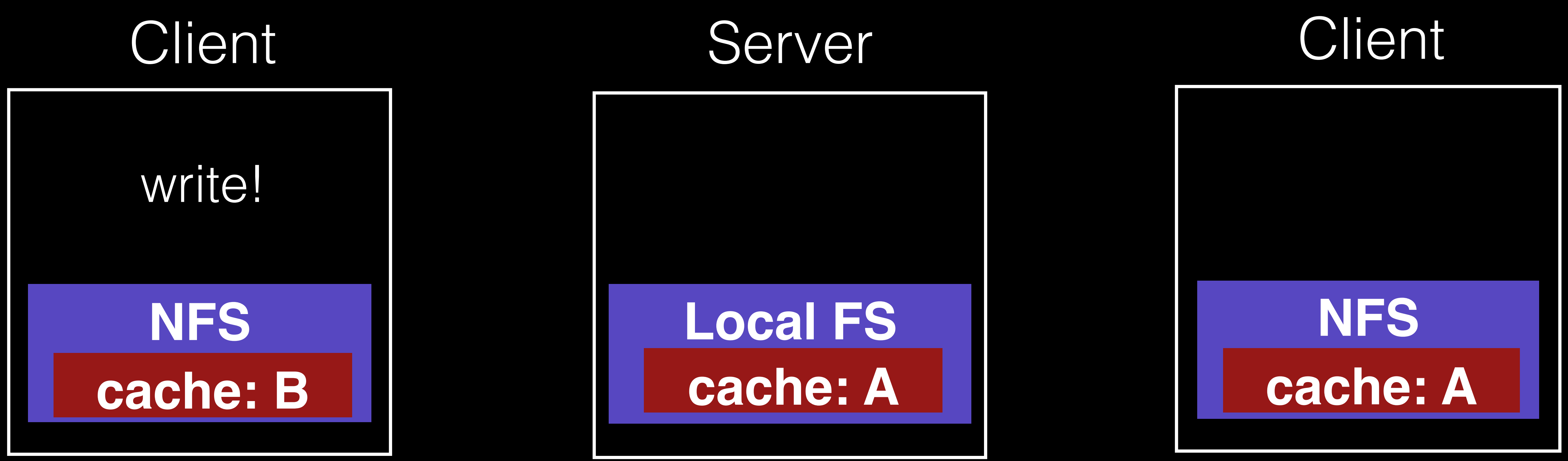
Update visibility

Stale cache

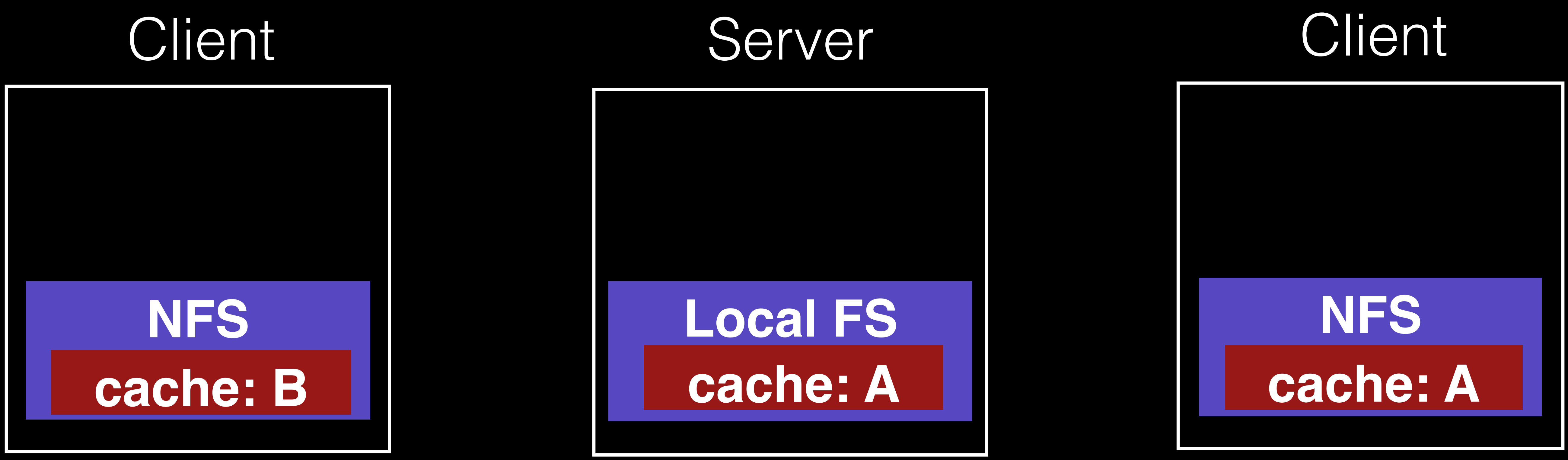
Update Visibility



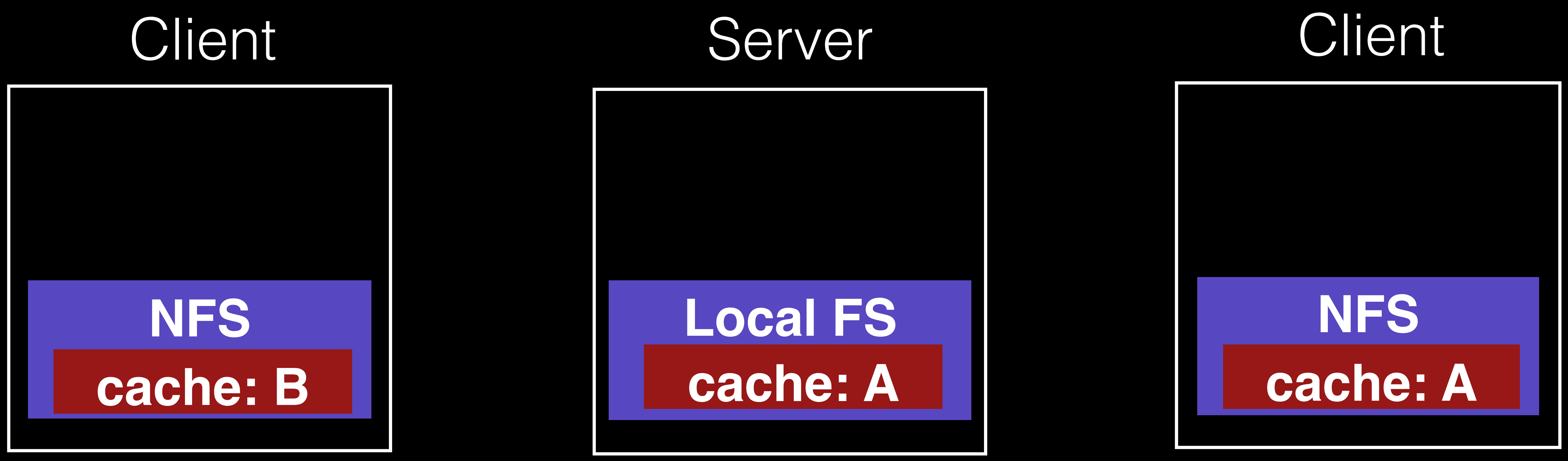
Update Visibility



Update Visibility



Update Visibility



"Update Visibility" problem: server doesn't have latest.

Update Visibility

Clients updates not seen on servers yet.

NFS solution is flush blocks:
- on close()
- when low on memory

Problems
- flushes not atomic (one block at a time)
- two clients flush at once: mixed data

Update Visibility

Clients updates not seen on servers yet.

AFS solution:
- flush on close
- buffer whole files on local disk

Concurrent writes? Last writer (i.e., closer) wins.

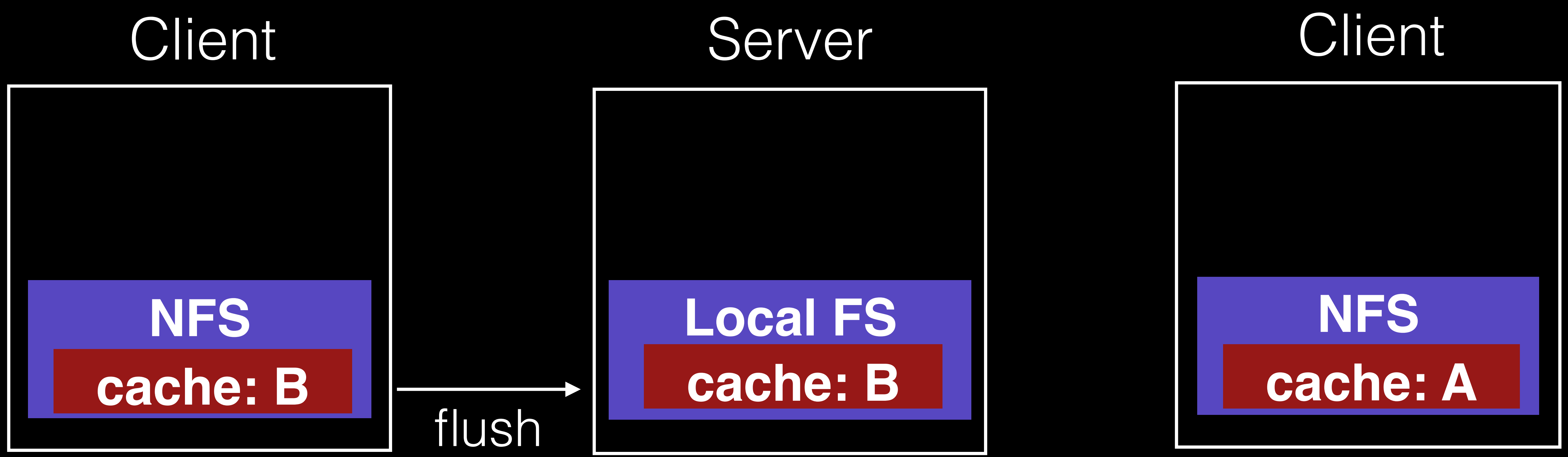
Never get mixed data.

Cache Consistency

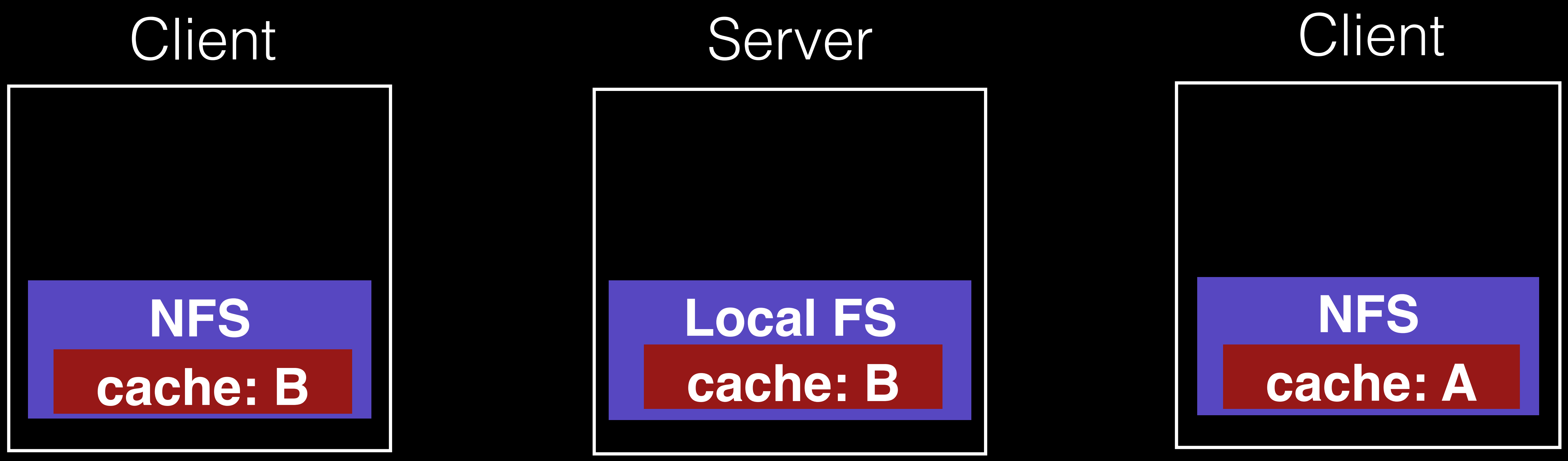
Update visibility

Stale cache

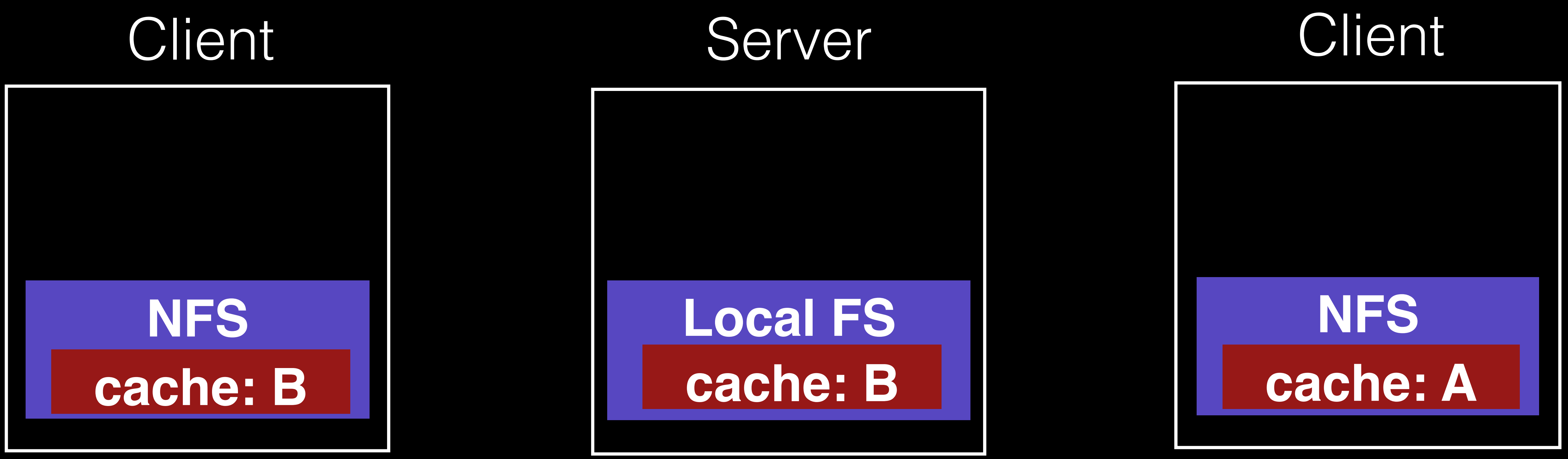
Cache Consistency



Cache Consistency



Cache Consistency



"Stale Cache" problem: client doesn't have latest.

Stale Cache

Clients have old version

NFS rechecks cache entries before using them, assuming a check hasn't been done "recently".

"Recent" is too long: ?

"Recent" is too short: ?

Stale Cache

Clients have old version

NFS rechecks cache entries before using them, assuming a check hasn't been done "recently".

"Recent" is too long: you read old data

"Recent" is too short: server overloaded with stats

Stale Cache

AFS solution: tell clients when data is overwritten.

When clients cache data, ask for “callback” from server.

No longer stateless!

Callbacks

What if client crashes?

What if server runs out of memory?

What if server crashes?

Callbacks

What if client crashes?

What if server runs out of memory?

What if server crashes?

Client Crash

What should client do after reboot?

Option 1: evict everything from cache

Option 2: ???

Client Crash

What should client do after reboot?

Option 1: evict everything from cache

Option 2: recheck before using

Callbacks

What if client crashes?

What if server runs out of memory?

What if server crashes?

Low Server Memory

Strategy: tell clients you are dropping their callback.

What should client do?

Low Server Memory

Strategy: tell clients you are dropping their callback.

What should client do? Mark entry for recheck.

Low Server Memory

Strategy: tell clients you are dropping their callback.

What should client do? Mark entry for recheck.

How does server choose which entry to bump?

Low Server Memory

Strategy: tell clients you are dropping their callback.

What should client do? Mark entry for recheck.

How does server choose which entry to bump?
Sadly, it doesn't know which is most useful.

Callbacks

What if client crashes?

What if server runs out of memory?

What if server crashes?

Server Crashes

What if server crashes?

Server Crashes

What if server crashes?

Option: tell everybody to recheck everything before next read.

Server Crashes

What if server crashes?

Option: tell everybody to recheck everything before next read.

Option: persist callbacks.

Callbacks

What if client crashes?

What if server runs out of memory?

What if server crashes?

AFS paper: “there is a potential for inconsistency if the callback state maintained by a [client] gets out of sync with the [server state]”.

Prefetching

AFS paper notes: "the study by Ousterhout *et al.* has shown that most files in a 4.2BSD environment are read in their entirety."

What are the implications for prefetching policy?

Prefetching

AFS paper notes: “the study by Ousterhout *et al.* has shown that most files in a 4.2BSD environment are read in their entirety.”

What are the implications for prefetching policy?

Aggressively prefetch whole files.

Whole-File Caching

Upon open, AFS fetches whole file (even if it's huge), storing it in local memory or disk.

Upon close, whole file is flushed (if it was written).

Convenient:
- AFS needs to do work for open/close
- reads/writes are local

Outline

- Volume management
- Cache management
- Name resolution
- Process structure
- Local-storage API
- File locks

Why is this Inefficient?

Requests to server:

```
fd1 = open(“/a/b/c/d/e/1.txt”)
fd2 = open(“/a/b/c/d/e/2.txt”)
fd3 = open(“/a/b/c/d/e/3.txt”)
```

Why is this Inefficient?

Requests to server:

```
fd1 = open(“/a/b/c/d/e/1.txt”)
fd2 = open(“/a/b/c/d/e/2.txt”)
fd3 = open(“/a/b/c/d/e/3.txt”)
```

Same inodes and dir entries repeatedly read.
Cache prevent too much disk I/O.
Too much CPU, though.

Solution

Server returns dir entries to client.

Client caches entries, inodes.

Pro: partial traversal is the common case.

Con: first lookup requires many round trips.

Outline

- Volume management
- Cache management
- Name resolution
- Process structure
- Local-storage API
- File locks

Process Structure

For each client, a different process ran on the server.

Context switching costs were high.

Solution: ???

Process Structure

For each client, a different process ran on the server.

Context switching costs were high.

Solution: use threads.

Shared addr space => more useful TLB entries.

Outline

Volume management

Cache management

Name resolution

Process structure

Local-storage API

File locks

Which API is faster? More convenient?

```
open(int inode, ...)

open(char *path, ...)
```

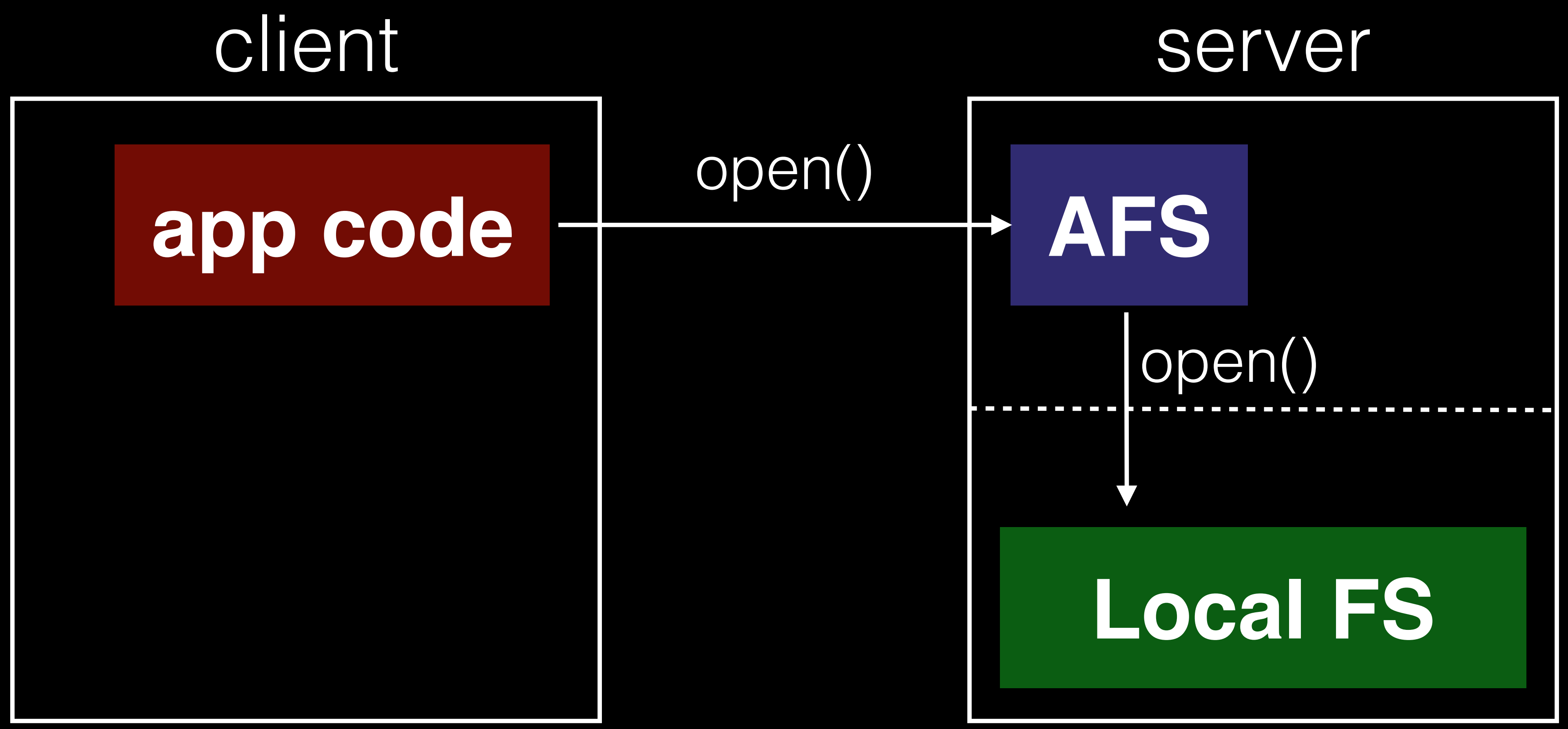
Which API is faster? More convenient?

```
open(int inode, ...)
```

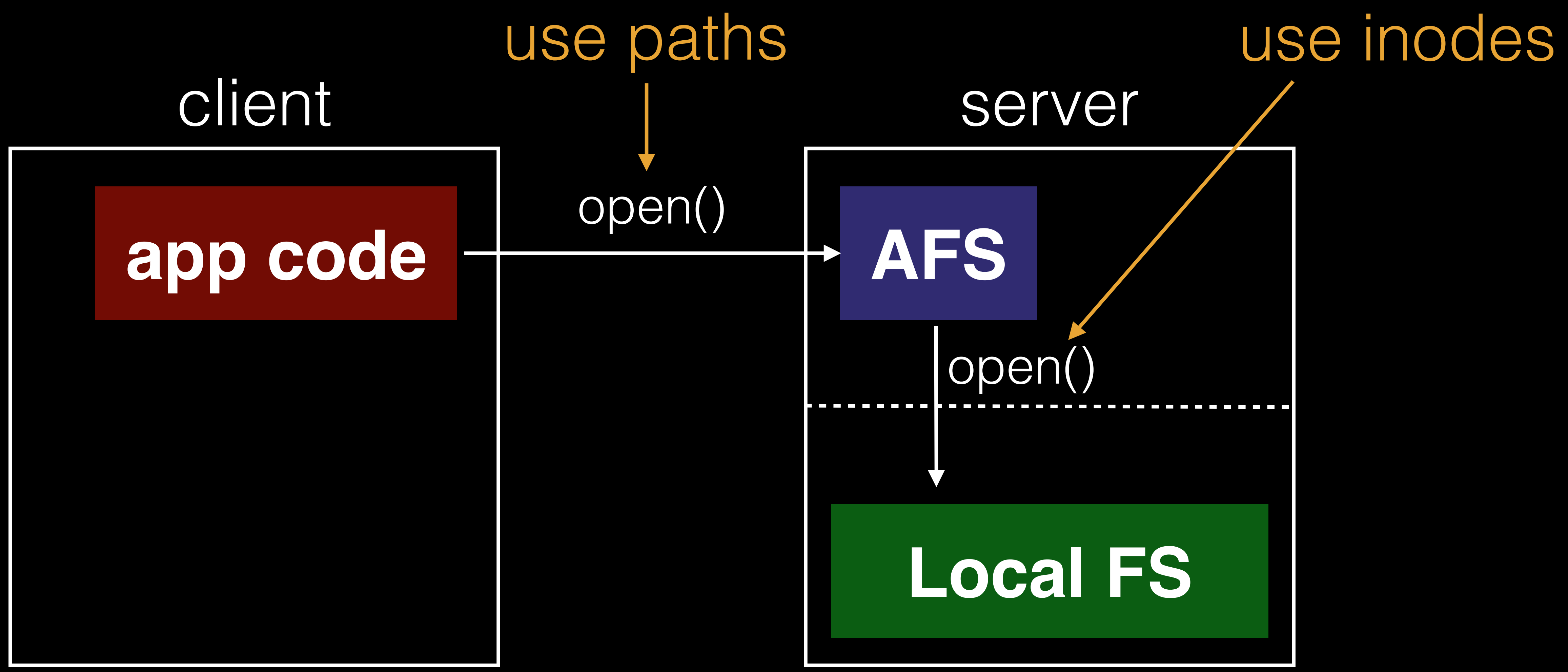
```
open(char *path, ...)
```

Lookup by inodes is faster (no traversal), but less convenient.

Which open API is better?



Which open API is better?



Which API is faster? More convenient?

```
open(int inode, ...)
```

```
open(char *path, ...)
```

Lookup by inodes is faster (no traversal),
but less convenient.

AFS developers added first call so AFS could use it.

Outline

Volume management

Cache management

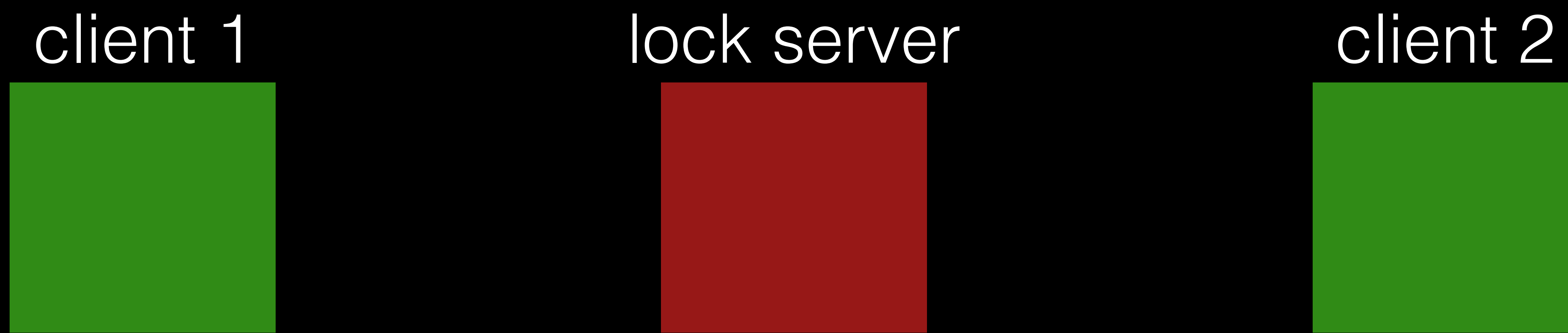
Name resolution

Process structure

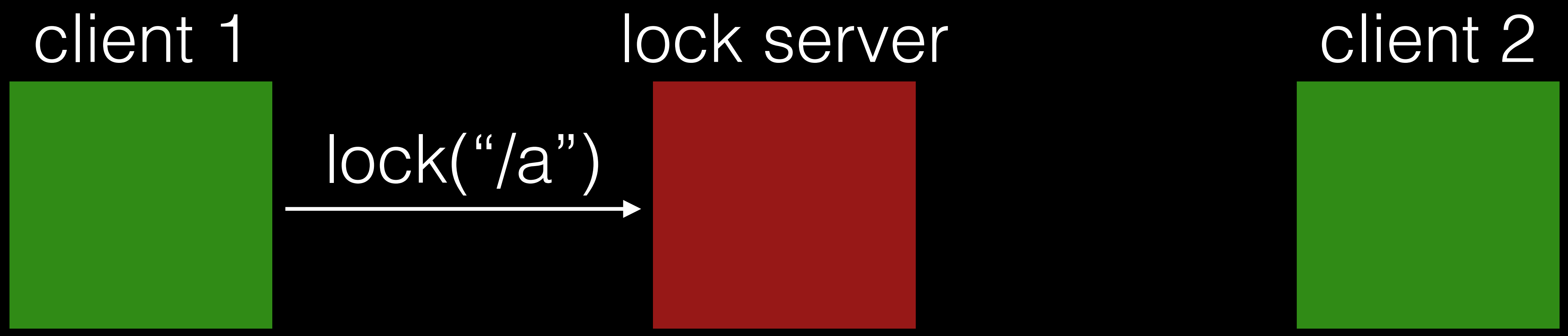
Local-storage API

File locks

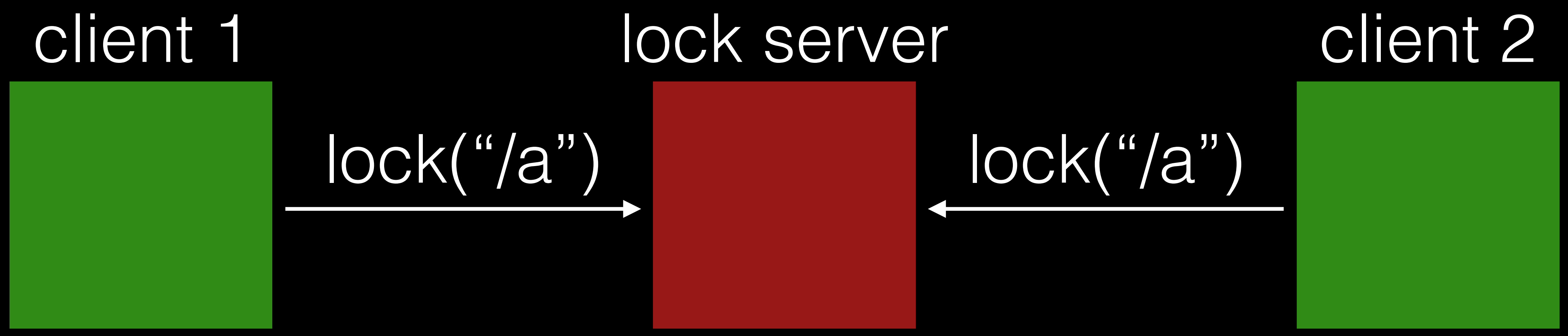
Dedicated Lock Server



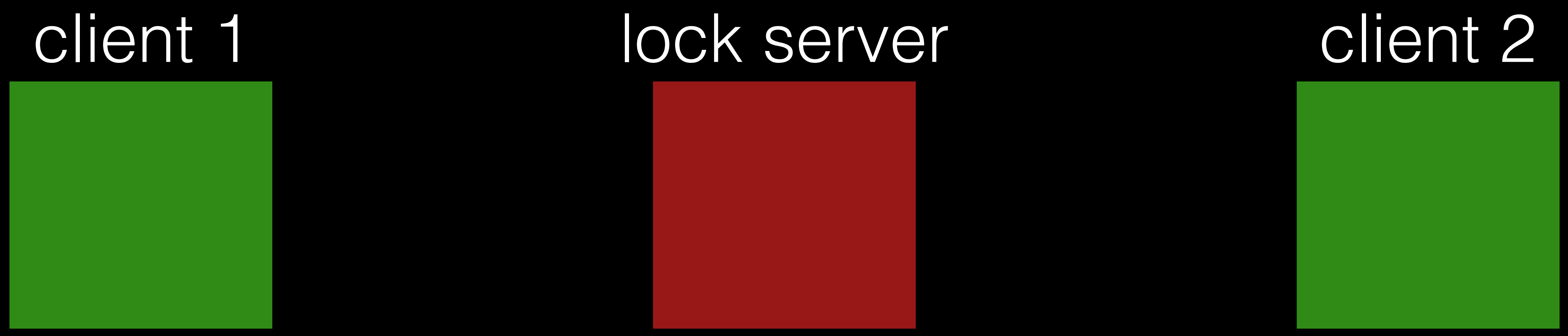
Dedicated Lock Server



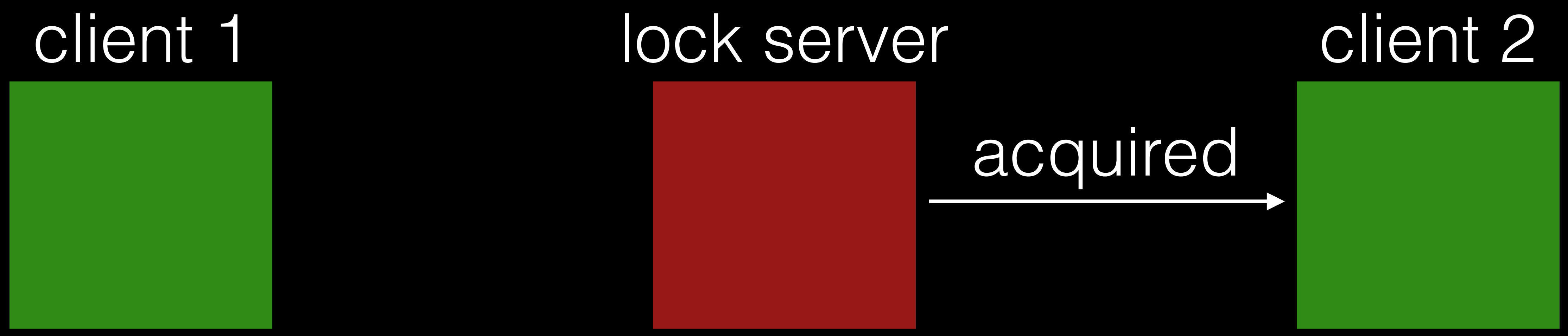
Dedicated Lock Server



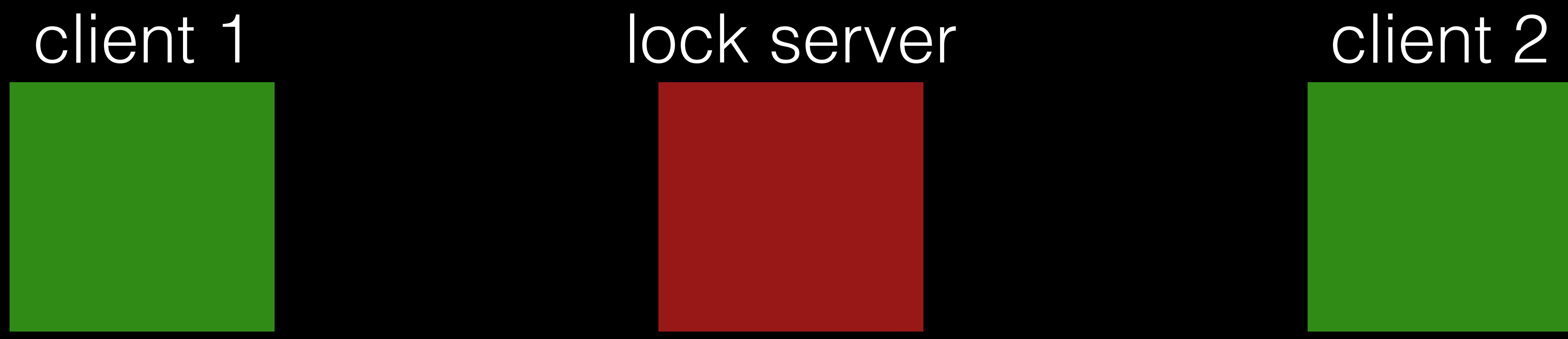
Dedicated Lock Server



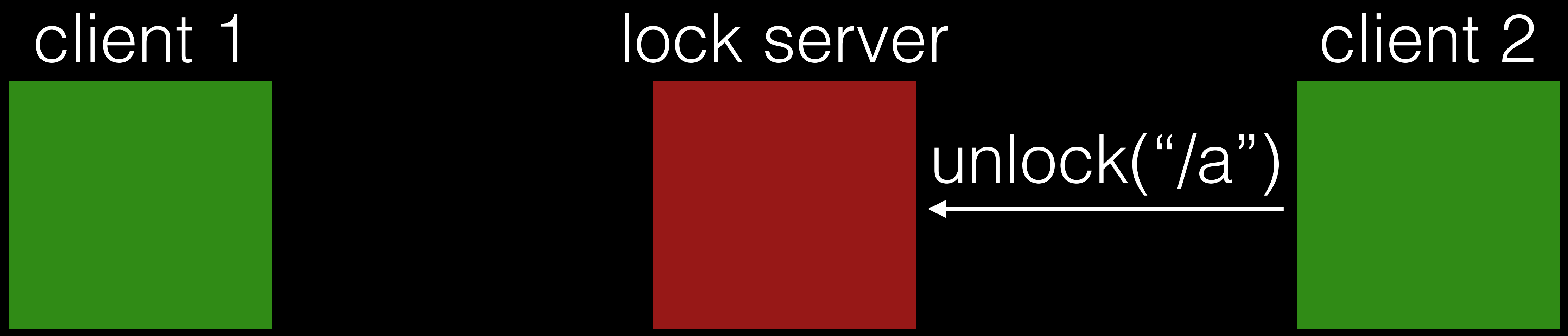
Dedicated Lock Server



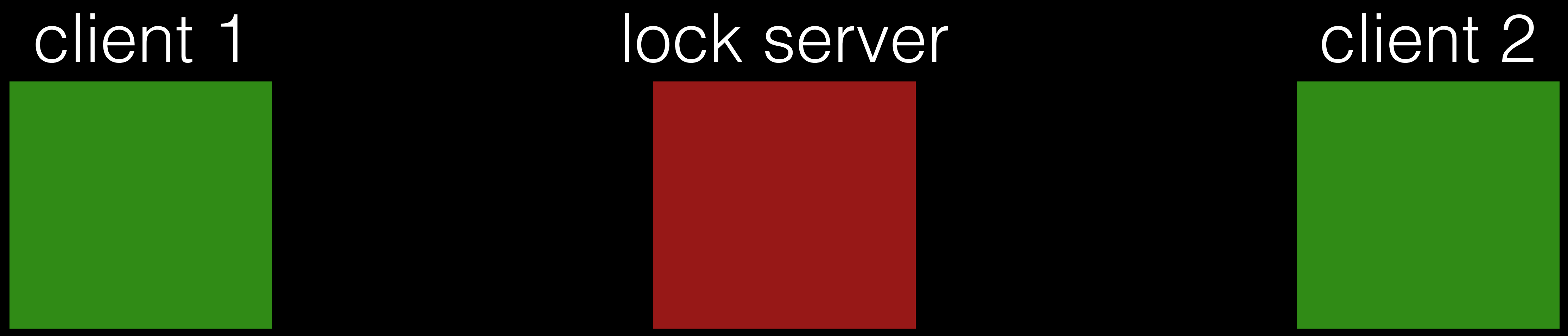
Dedicated Lock Server



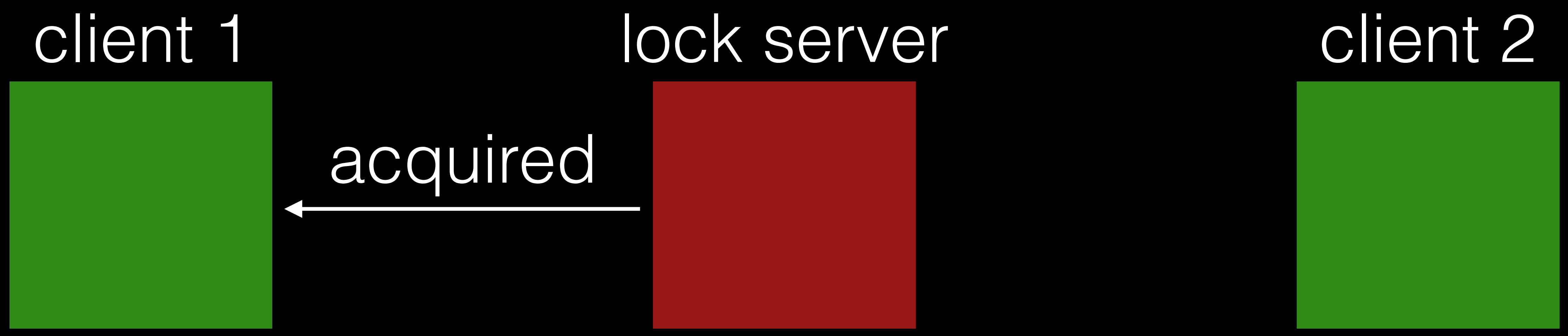
Dedicated Lock Server



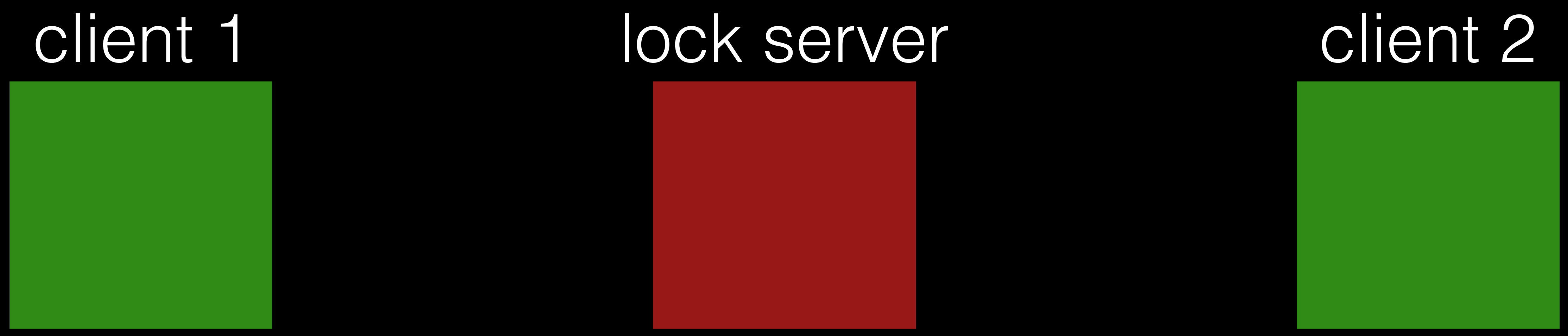
Dedicated Lock Server



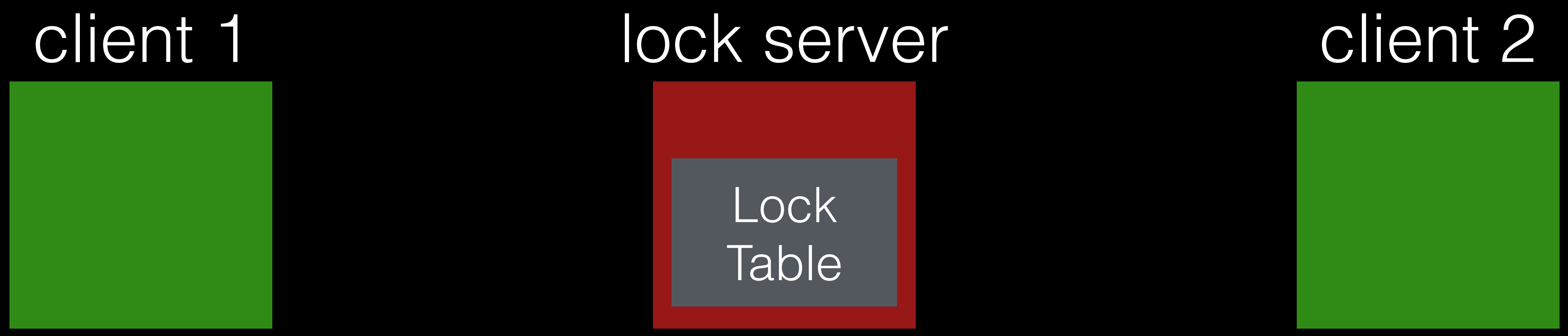
Dedicated Lock Server



Dedicated Lock Server



Dedicated Lock Server



Lock Table

```
void table_lock(char *name) {
   hash_entry_t *entry;
   acquire(guard);
   entry = find_or_create(name);
   release(guard);
   lock(entry->lock);
}

void table_unlock(char *name) {
   hash_entry_t *entry;
   acquire(guard);
   entry = find_or_create(name);
   release(guard);
   unlock(entry->lock);
}
```

Lock Table

```
void table_lock(char *name) {
   hash_entry_t *entry;
   acquire(guard);
   entry = find_or_create(name);
   release(guard);
   lock(entry->lock);
}

void table_unlock(char *name) {
   hash_entry_t *entry;
   acquire(guard);
   entry = find_or_create(name);
   release(guard);
   unlock(entry->lock);
}
```

expose these with RPCs

Outline

Volume management

Cache management

Name resolution

Process structure

Local-storage API

File locks

Summary

Multi-step copy and forwarding make volume migration fast and consistent.

Workload drives design: whole-file caching.

State is useful for scalability, but makes consistency hard.

Announcements

p5a and p5b due Dec 12.

Office hours today, at 1pm, in office.

Thursday discussion held this week.

New: can drop 1 sub project.