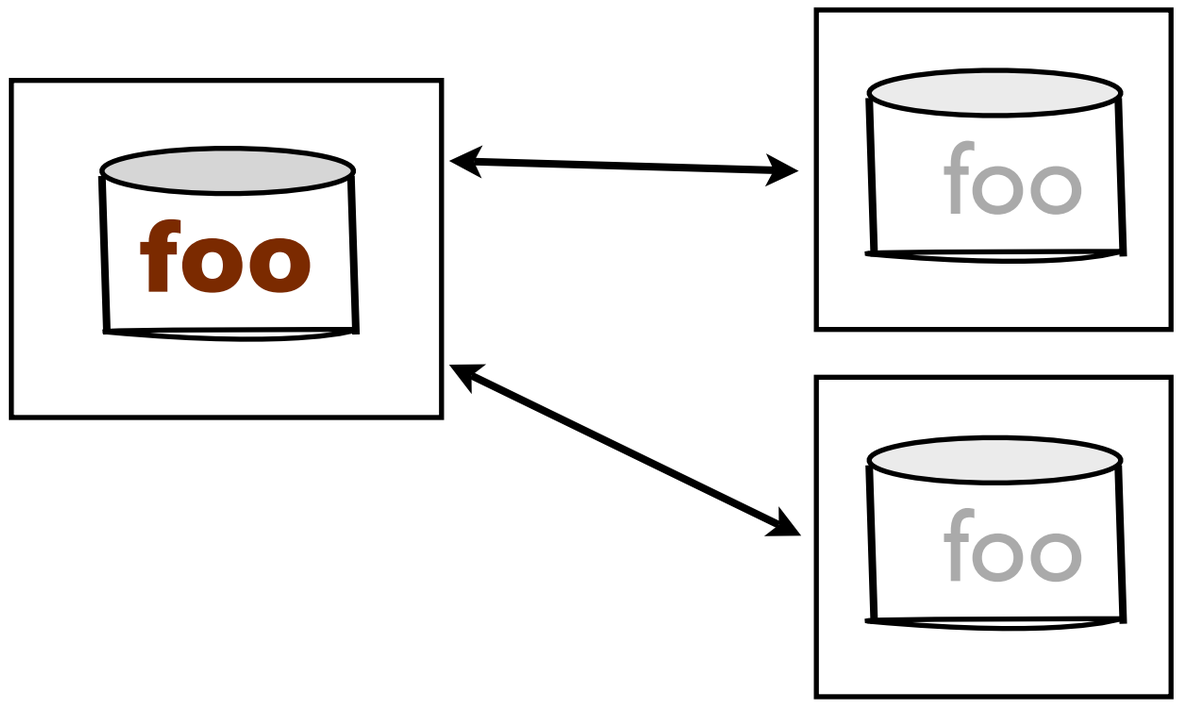
Distributed File Systems

NFS 3 vs. Sprite FS

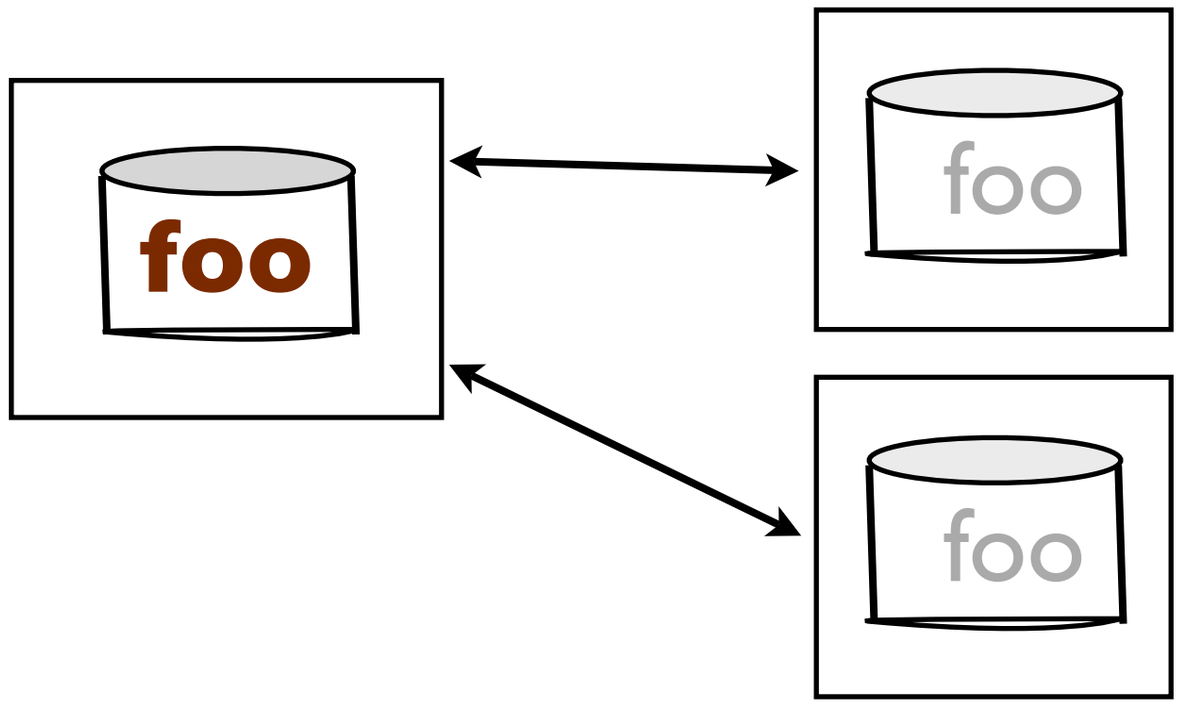
Stateful vs. Stateless

Intro: Distributed File Systems



- Client - Server model : clients use server resources
- Files of a remote computer mounted locally (transparent)
- Mobility of files (transparent), one spacey central server.
- E.g: Lab machines mount home dir from allspice.
  They use Network File System protocol (NFS).

Intro: Distributed File Systems



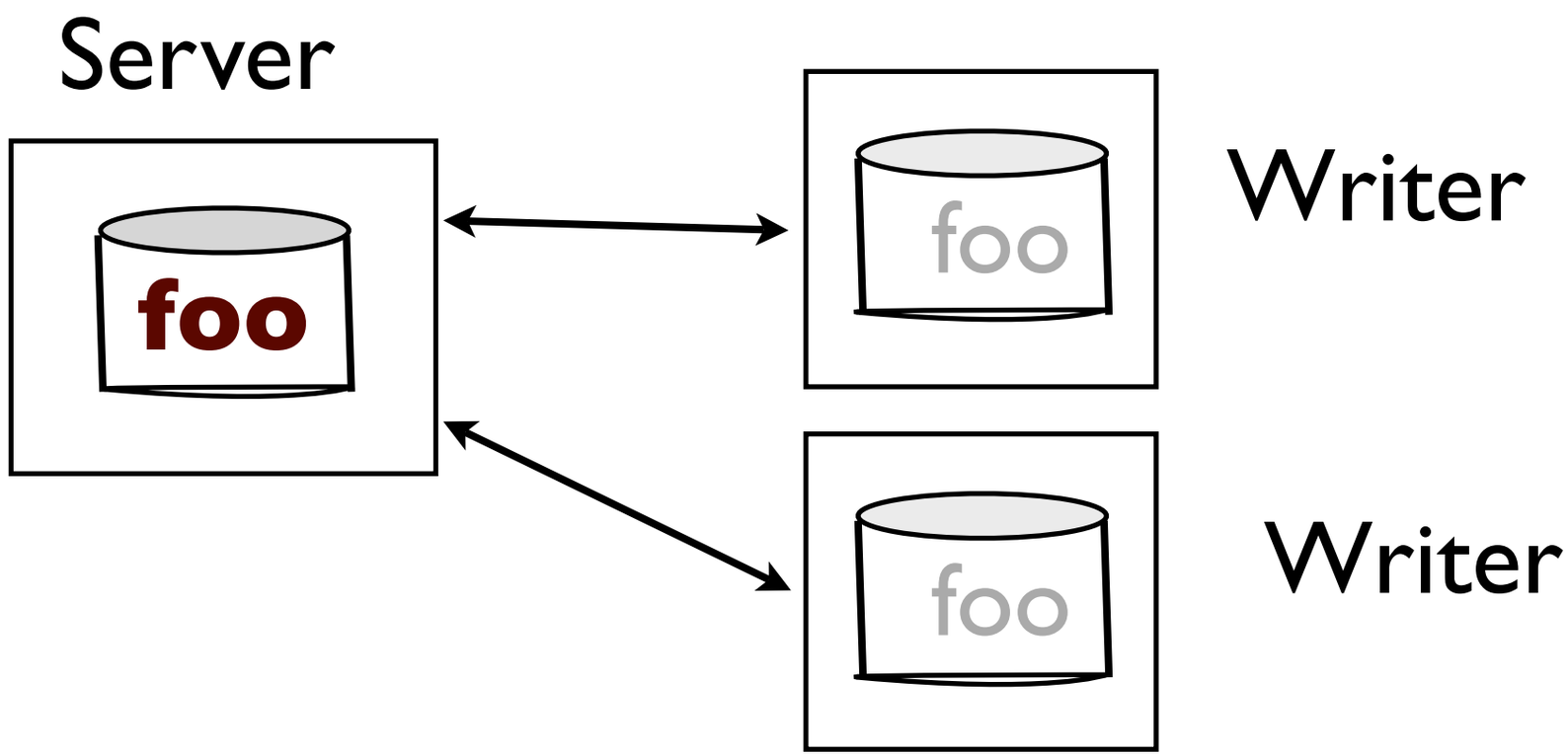
- Multiple users (readers and writers) possibly of the same file
- Client side caching for speed
- A problem with caching : global consistency
- Consistency: Having only one version of a file.
- Unix model: Let the user deal with consistency. (also NFS)

NFS 3

- Stateless
- Client writes back immediately.(write through)
- Client pings back to check state of file
    a. Local cache is current => continue
    b. Local cache is old => invalidate

NFS 3

Example: shared file **foo**



Write through Periodic checks

Consistency Guarantees?

No!

Stateless write through policy

-	+
1. No consistency guarantee 2. Network traffic (often wasteful: single R/W, temp files) 3. Server bottle-neck 4. Writer is blocked until completion	1. Simplicity 2. Crash recovery (NFS Clients could still lose data since it isn't strict write through)

Stateless write through policy

Network Traffic:

- Writethrough every few secs, ~30 secs.
- Unnecessary for temp files or for singly shared files.
- Expensive: Computationally, network traffic
- Traffic causes server bottle-neck

Speed:

- Slowed down by network and,
- synchronous write: To guarantee write before proceeding.

(NFS doesn't do this! = problems with consistency/ data loss)

Stateless write through policy

Consistency:

File writes can occur in the 30 sec gap.

UNIX's level of consistency

Fault Tolerance:

As good as UNIX's.

Little data is lost.

Spritely NFS

- Stateful : maintains a state of all open files.

- Open/Close() calls give server information:
  read/write mode, keep track of number of clients, versions of files

- Callback : Server can issue calls to clients for sake of consistency.

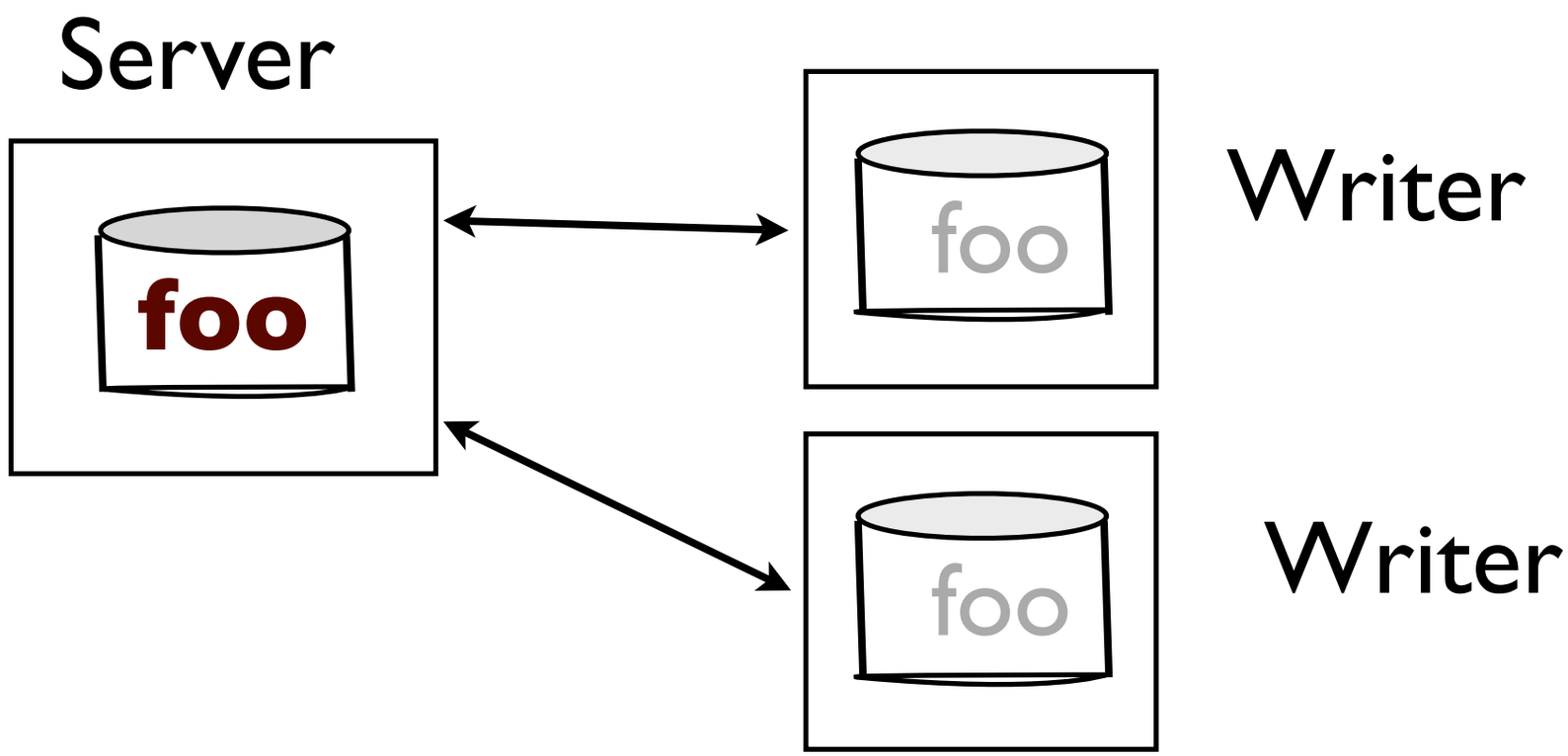
Spritely NFS

Having file state, SNFS can improve efficiency.

- Eliminate useless write-through:
  Unless write shared, no write-through
- Version number: During open, refresh local cache only if current version is old
- Guaranteed consistency through call-backs and version checking.

SNFS 3

Example: shared file **foo**



Conditional caching/ Writethrough

SNFS 3

State Transitions

Example: shared file **foo**

From State	To State	When	Caching	Callback
Closed	1 reader	Open for R	Enabled	None
Closed	1 writer	Open for W	Enabled	None
1 reader	Write shared	Open for W by new	Disabled	Invalidate
1 reader	Multi Readers	Open for R by new	Enabled	None
1 writer	Write shared	Open for R/W by new	Disabled	Write-back and invalidate

Cachable/ Uncachable files

Performance

Which is faster?

NFS	**SNFS**
Read/ Scan (one less RPC)	Write, Temp files (make)

Andrew Benchmark: SNFS ~ 2x faster

Which is less work?

SNFS: Fewer RPCs over the life of a file
Delayed write allows parallelism
No significant increase in computation

Fault Tolerance

NFS	**SNFS**
Easy recovery, not much loss because of write-through	None implemented complex slow down However, consistency can be maintained

NFS 4

- Stateful
- File locking (required for consistency)
- Delayed write, Open & Close.
- Other enhancements:
  RPC bundle (compound procedure)
- Lease: delegation of open/close/locking