NFS

Shivaram Venkataraman

CS 537, Spring 2020

ADMINISTRIVIA

AEFIS feedback

Optional project

Final exam details

No discussion this week!

AGENDA / LEARNING OUTCOMES

How to design a distributed file system that can survive partial failures?

What are consistency properties for such designs?

RECAP

DISTRIBUTED FILE SYSTEMS

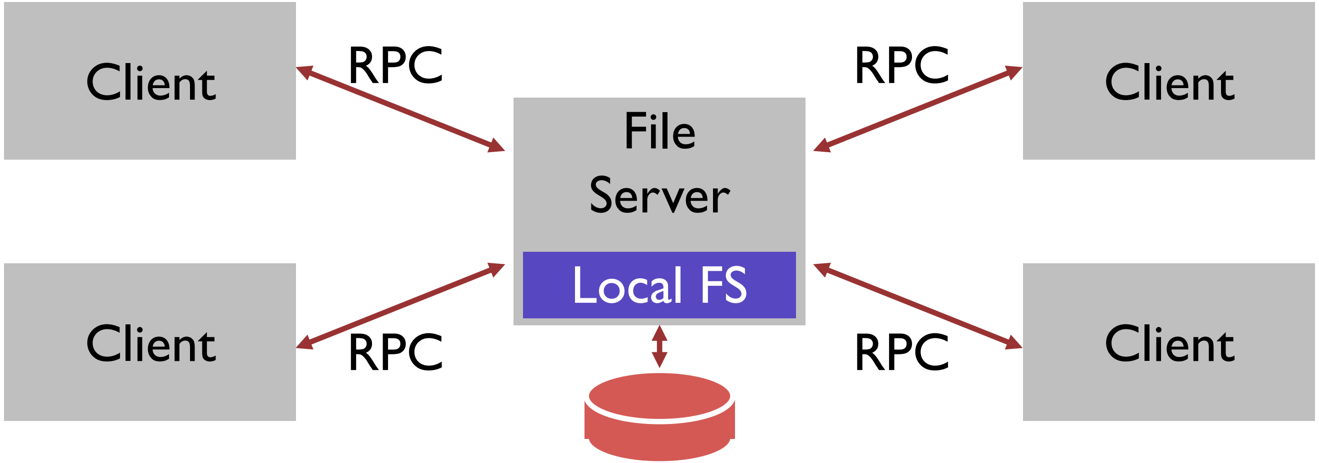
Local FS: processes on same machine access shared files

Network FS: processes on different machines access shared files in same way

Goals

- Transparent access
- Fast + simple crash recovery
- Reasonable performance?

NFS ARCHITECTURE



STRATEGY 1

Attempt: Wrap regular UNIX system calls using RPC

open() on client calls open() on server

open() on server returns fd back to client

read(fd) on client calls read(fd) on server

read(fd) on server returns data back to client

```
int fd = open(“foo”, O_RDONLY);
read(fd, buf, MAX);
…
read(fd, buf, MAX);
```

Server crash!

STRATEGY 2: PUT ALL INFO IN REQUESTS

“Stateless” protocol: server maintains no state about clients

Need API change. One possibility:

pread(char *path, buf, size, offset);

pwrite(char *path, buf, size, offset);

Specify path and offset each time. Server need not remember anything from clients.

Pros? Server can crash and reboot transparently to clients

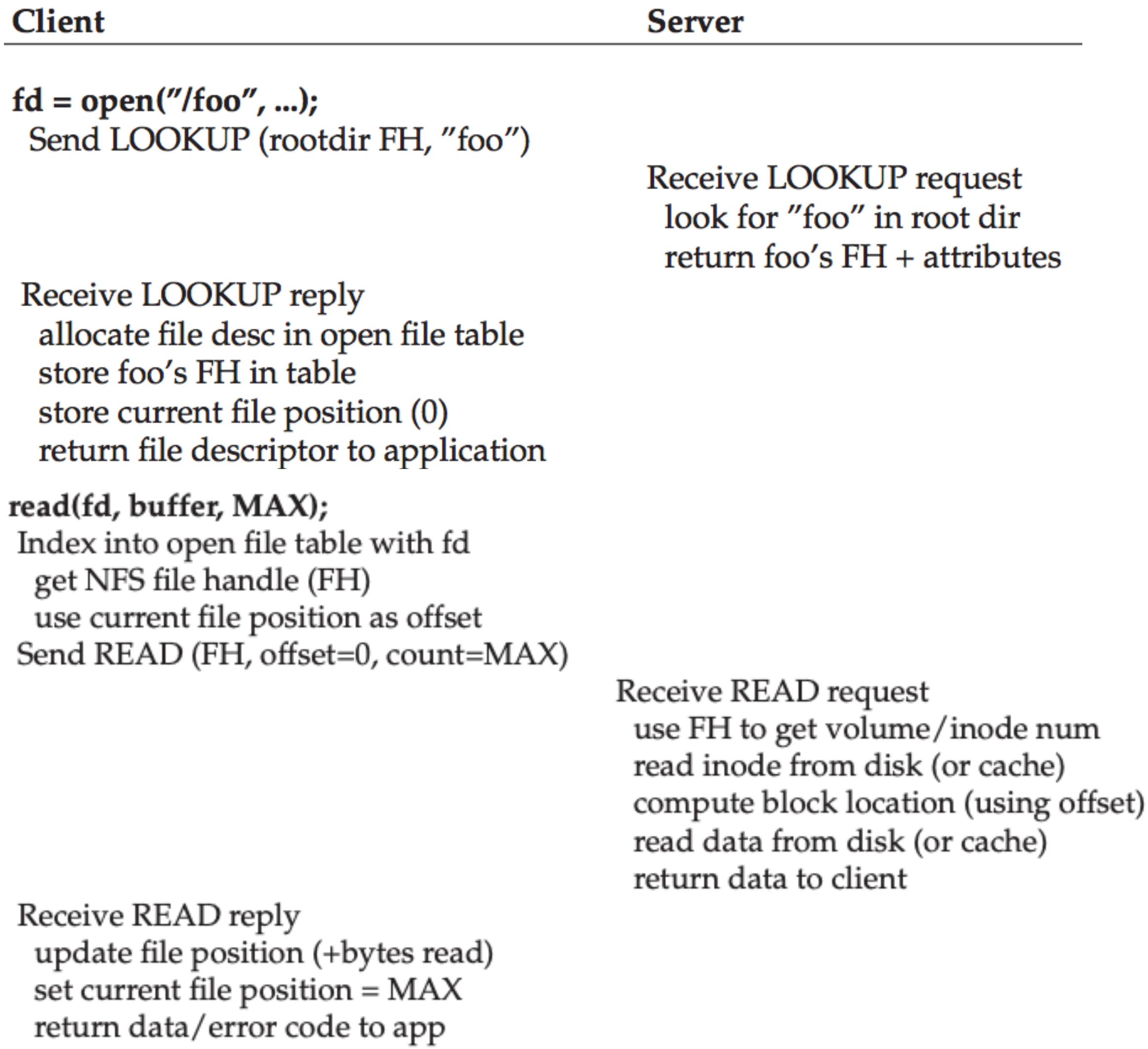
Cons? Too many path lookups.

STRATEGY 3: FILE HANDLES

```
fh = open(char *path);
pread(fh, buf, size, offset);
pwrite(fh, buf, size, offset);
```

File Handle = <volume ID, inode #, **generation #**>

Opaque to client (client should not interpret internals)



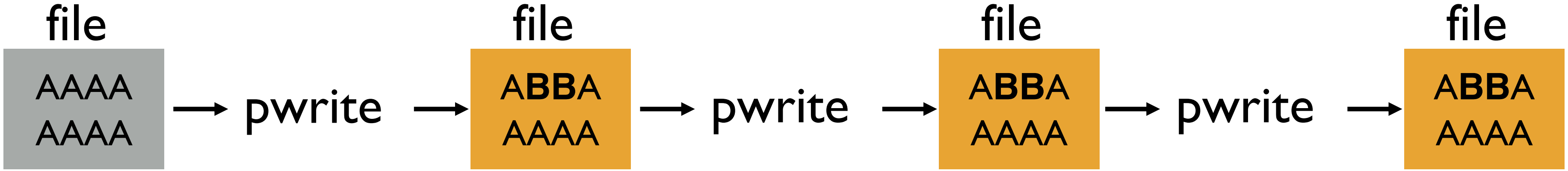
Client	Server
fd = open("/foo", ...);	
Send LOOKUP (rootdir FH, "foo")	
	Receive LOOKUP request look for "foo" in root dir return foo's FH + attributes
Receive LOOKUP reply allocate file desc in open file table store foo's FH in table store current file position (0) return file descriptor to application	
read(fd, buffer, MAX);	
Index into open file table with fd get NFS file handle (FH) use current file position as offset Send READ (FH, offset=0, count=MAX)	
	Receive READ request use FH to get volume/inode num read inode from disk (or cache) compute block location (using offset) read data from disk (or cache) return data to client
Receive READ reply update file position (+bytes read) set current file position = MAX return data/error code to app	

CAN NFS PROTOCOL INCLUDE APPEND?

```
fh = open(char *path);
pread(fh, buf, size, offset);
pwrite(fh, buf, size, offset);

append(fh, buf, size);
```

PWRITE VS APPEND

pwrite(file, “BB”, 2, 2);



file
AAAA
AAAA
→ pwrite →
file
ABBA
AAAA
→ pwrite →
file
ABBA
AAAA
→ pwrite →
file
ABBA
AAAA

append(file, “BB”);

IDEMPOTENT OPERATIONS

Solution: Design API so no harm to executing function more than once

If f() is idempotent, then:

f() has the same effect as f(); f(); … f(); f()

```
int fd = open(“foo”, O_RDONLY);
read(fd, buf, MAX);
write(fd, buf, MAX);
…
```

Server crash!

WHAT OPERATIONS ARE IDEMPOTENT?

Idempotent

- any sort of read that doesn't change anything
- pwrite

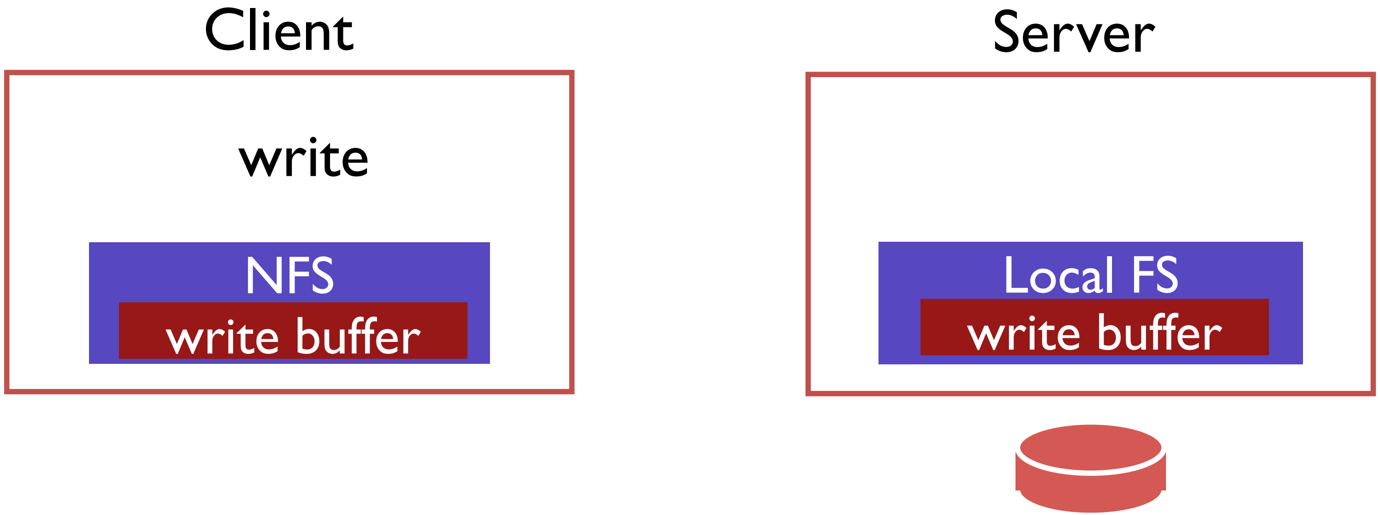
Not idempotent

- append

What about these?

- mkdir
- creat

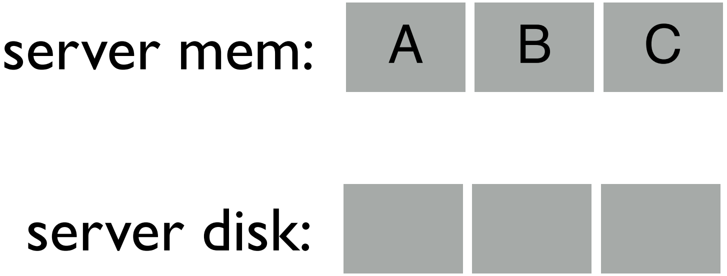
WRITE BUFFERS



Server acknowledges write before write is pushed to disk;
What happens if server crashes?

SERVER WRITE BUFFER LOST

```
client:


 write A to 0
 write B to 1
 write C to 2
```



server mem: A B C

server disk:

server acknowledges write before write is pushed to disk

SERVER WRITE BUFFER LOST

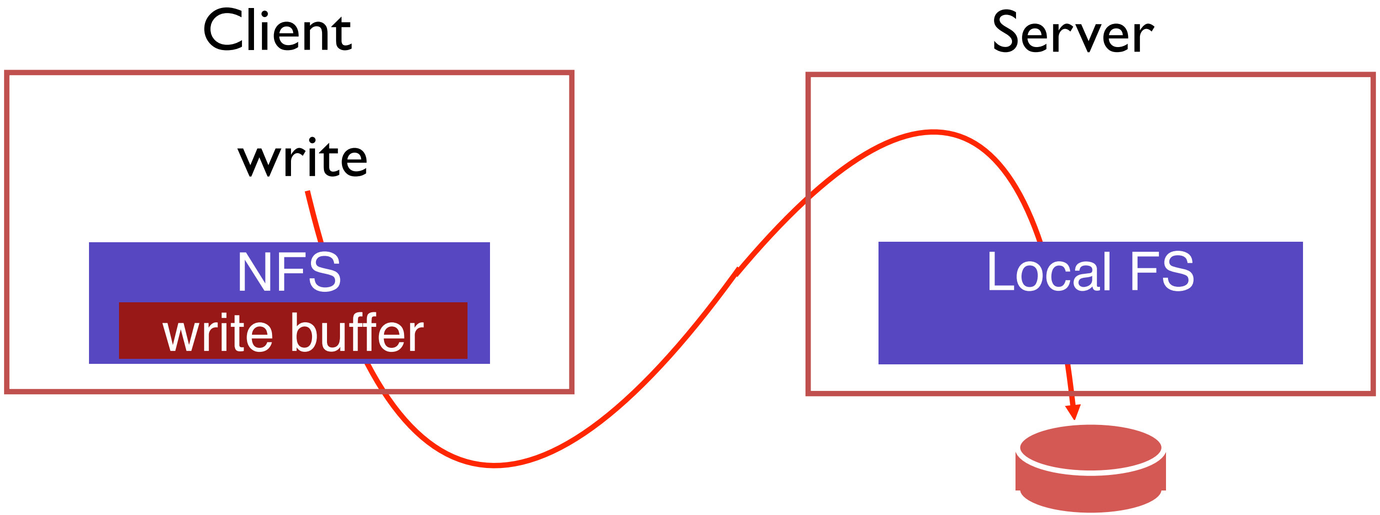
```
Client:

 write A to 0

 write B to 1

 write C to 2

 write X to 0

 write Y to 1

 write Z to 2
```

server mem:			Z
server disk:	X	B	Z

Problem:
No write failed, but disk state doesn’t match any point in time

Solutions?

WRITE BUFFERS



Don't use server write buffer. Problem: Slow?

Use persistent write buffer (more expensive)

QUIZ 31

https://tinyurl.com/cs537-sp20-quiz31

The only costs to worry about are network costs. Assume "small" messages takes S units of time, whereas a "bigger" message (e.g., size of a block=4KB) takes B units. If a message is larger than 4KB, it takes longer (2B for 8KB).

1. How long does it take to open a 100-block (400 KB) file called /a/b/c.txt for the first time? (assume root directory file handle is already available)

2. How long does it take to read the whole file?

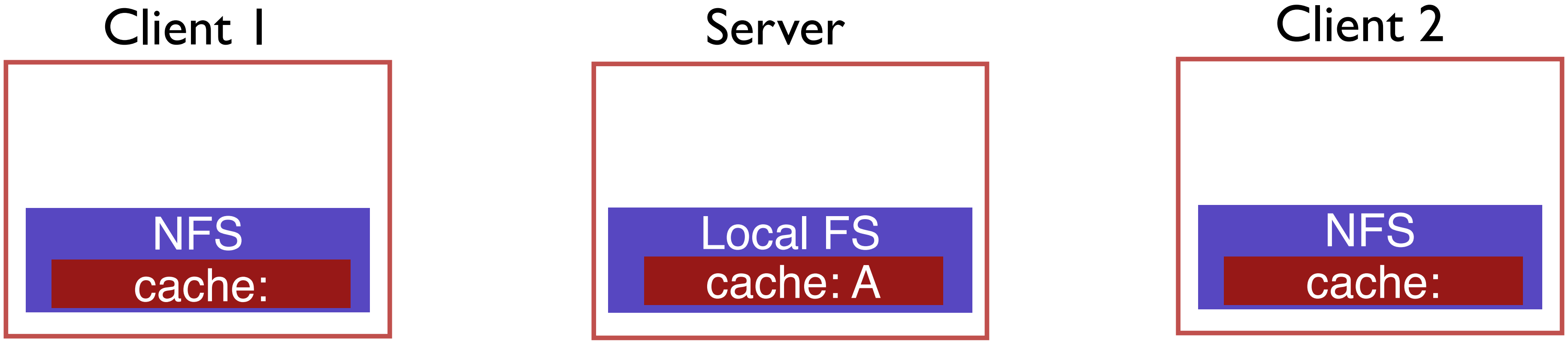
CACHE CONSISTENCY

NFS can cache data in three places:

- server memory
- client disk
- client memory

How to make sure all versions are in sync?

DISTRIBUTED CACHE



Client 1

NFS

cache:

Server

Local FS

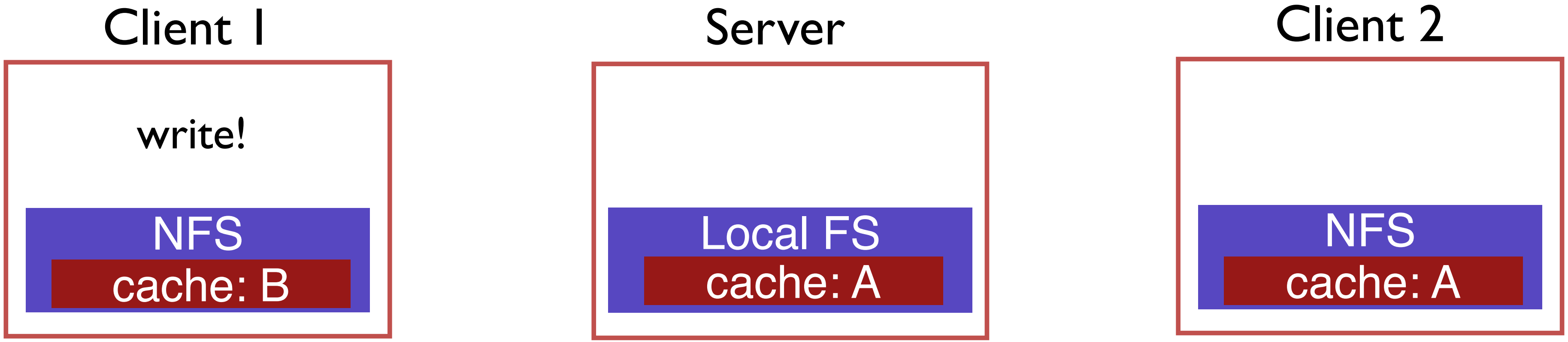
cache: A

Client 2

NFS

cache:

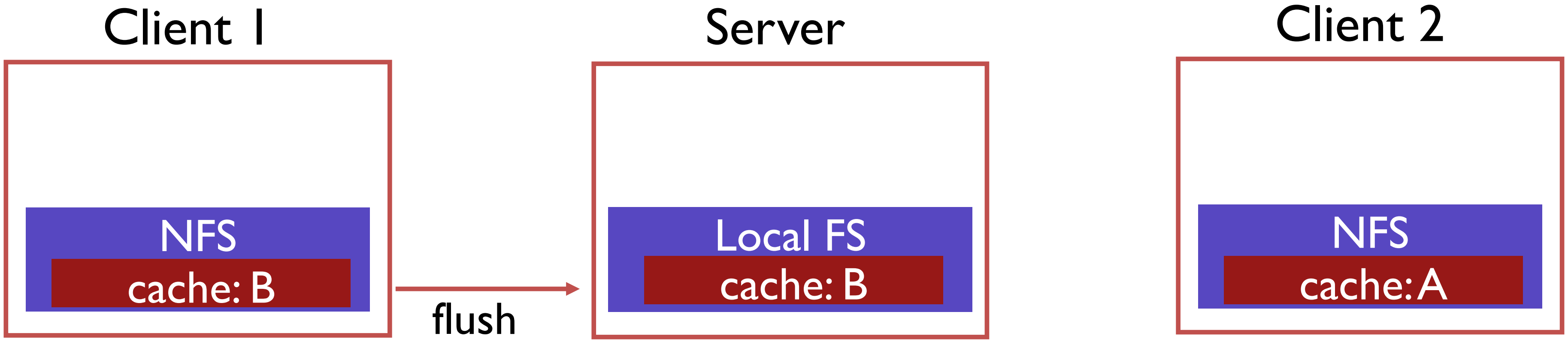
CACHE



“Update Visibility” problem: server doesn’t have latest version

What happens if Client 2 (or any other client) reads data?

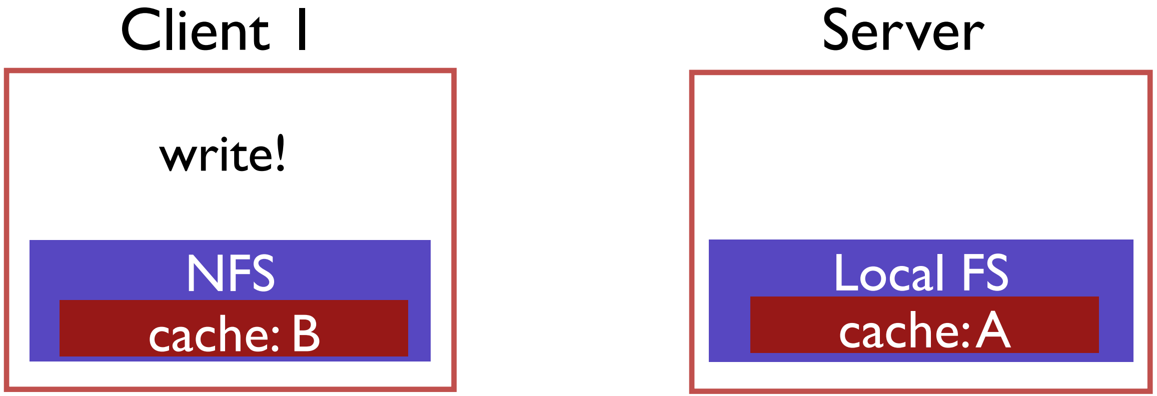
CACHE



“Stale Cache” problem: client 2 doesn’t have latest version

What happens if Client 2 reads data?

PROBLEM 1: UPDATE VISIBILITY



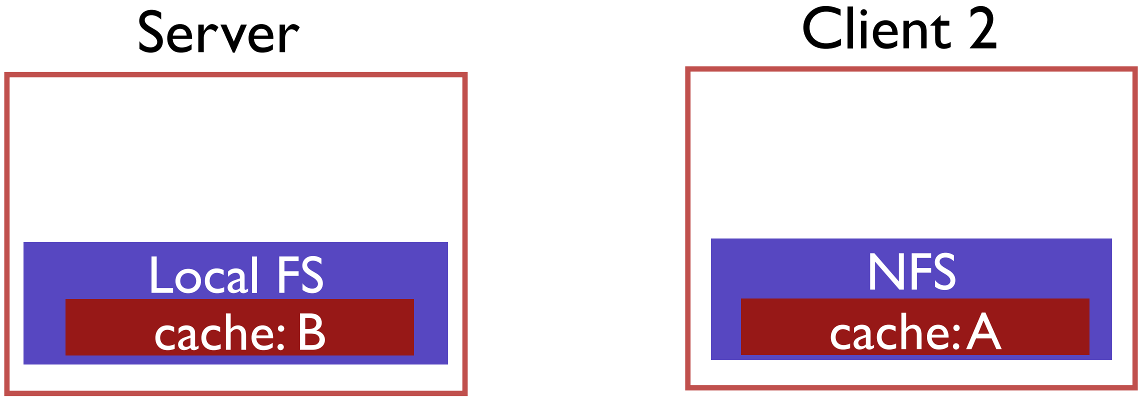
When client buffers a write, how can server (and other clients) see update?

Client flushes cache entry to server

When should client perform flush?

NFS solution: flush on fd close

PROBLEM 2: STALE CACHE

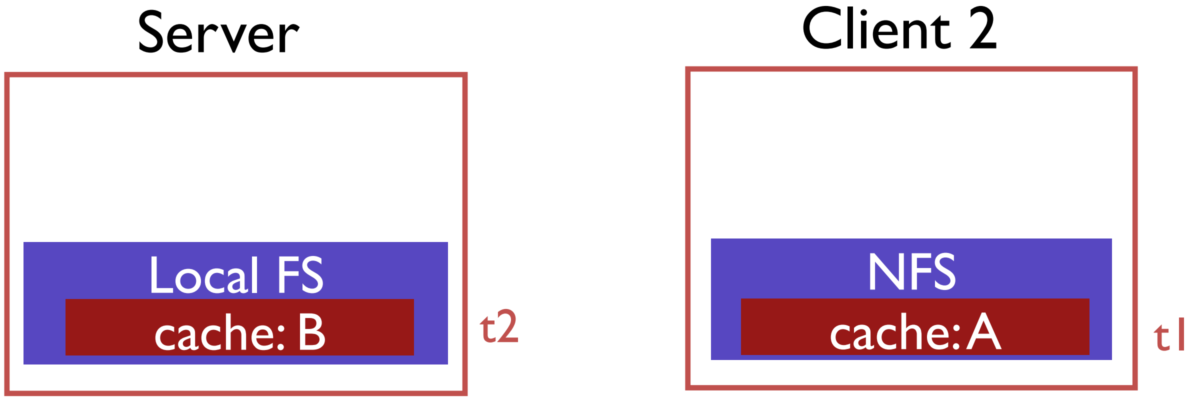


Problem: Client 2 has stale copy of data; how can it get the latest?

NFS solution:

- Clients recheck if cached copy is current before using data

STALE CACHE SOLUTION



Client cache records time when data block was fetched (t1)

Before using data block, client does a STAT request to server

- get's last modified timestamp for this file (t2) (not block…)
- compare to cache timestamp
- refetch data block if changed since timestamp (t2 > t1)

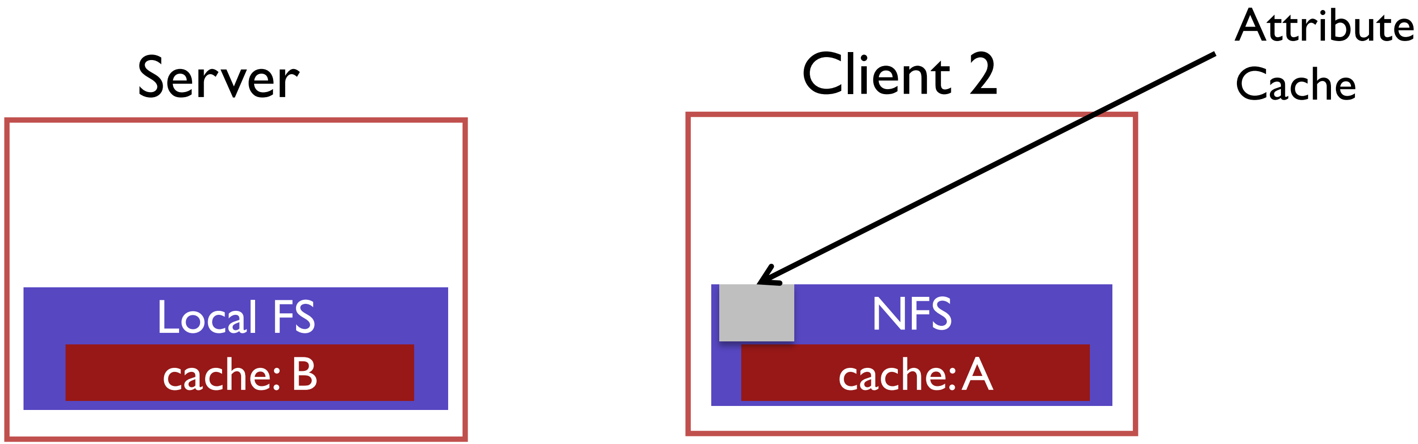
MEASURE THEN BUILD

NFS developers found `stat` accounted for 90% of server requests

Why?

Because clients frequently recheck cache

REDUCING STAT CALLS



Solution: cache results of `stat` calls

Partial Solution:

Make stat cache entries expire after a given time

(e.g., 3 seconds) (discard t2 at client 2)

What is the consequence?

NFS SUMMARY

NFS handles client and server crashes very well; robust APIs that are:

- stateless: servers don't remember clients
- idempotent: doing things twice never hurts

Caching and write buffering is harder, especially with crashes

Problems:

– Consistency model is odd (client may not see updates until 3s after file closed)

– Scalability limitations as more clients call stat() on server

NEXT STEPS

Next class: Review, Looking forward

Optional project due Wed

AEFIS feedback