CS202 (003): Operating Systems
File System IV

Instructor: Jocelyn Chen

Most of the materials covered in this slide come from the lecture notes of Mike Walfish's CS202

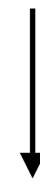
Last Time

Journaling — redo logging (used by ext3 & ext4)

FS computes what would change due to a operation

FS computes where in the log it can write this transaction, and writes a transactions begin record . Do not have to wait on this.

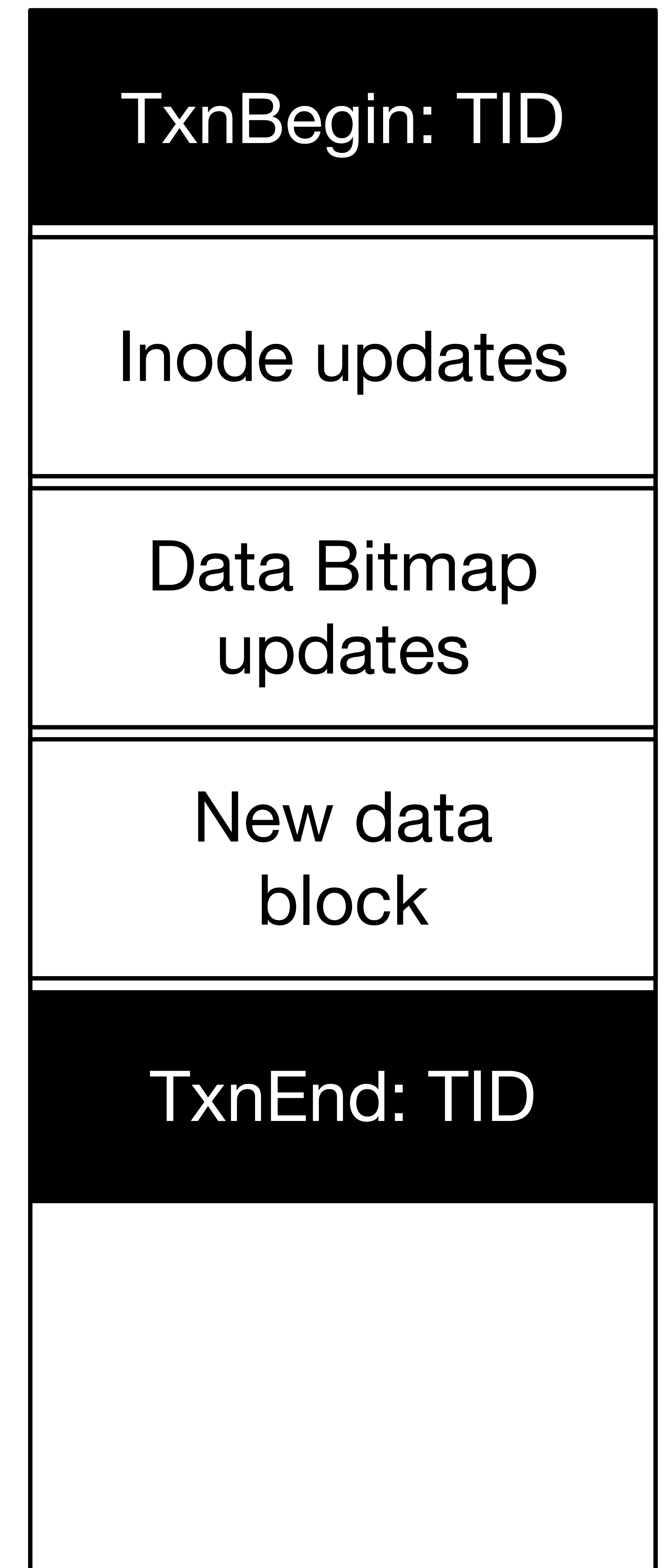
FS writes a record(s) with all the changes it computed in step 1. FS **must wait** for changes and TxnBegin to finish written to the disk



Once step 3 finishes, the system writes a transaction end record

Once the TxnEnd has been written, the FS asynchronously performs the actual FS changes

"checkpointing"



ext3 journal layout

Journaling — crash recovery of redo logging

High-level idea:
read through the logs, find **committed operations** and apply them

How to check whether ops are committed? Look at TxnBegin and TxnEnd!
It is safe to apply the same redo log multiple times

FS starts scanning from the **beginning of the log**

↓

Every time it finds a TxnBegin entry, it looks for the corresponding TxnEnd entry

↓

If matching (TxnBegin, TxnEnd) found, FS checkpoints the changes

↓

Recovery is completed once the entire log is scanned

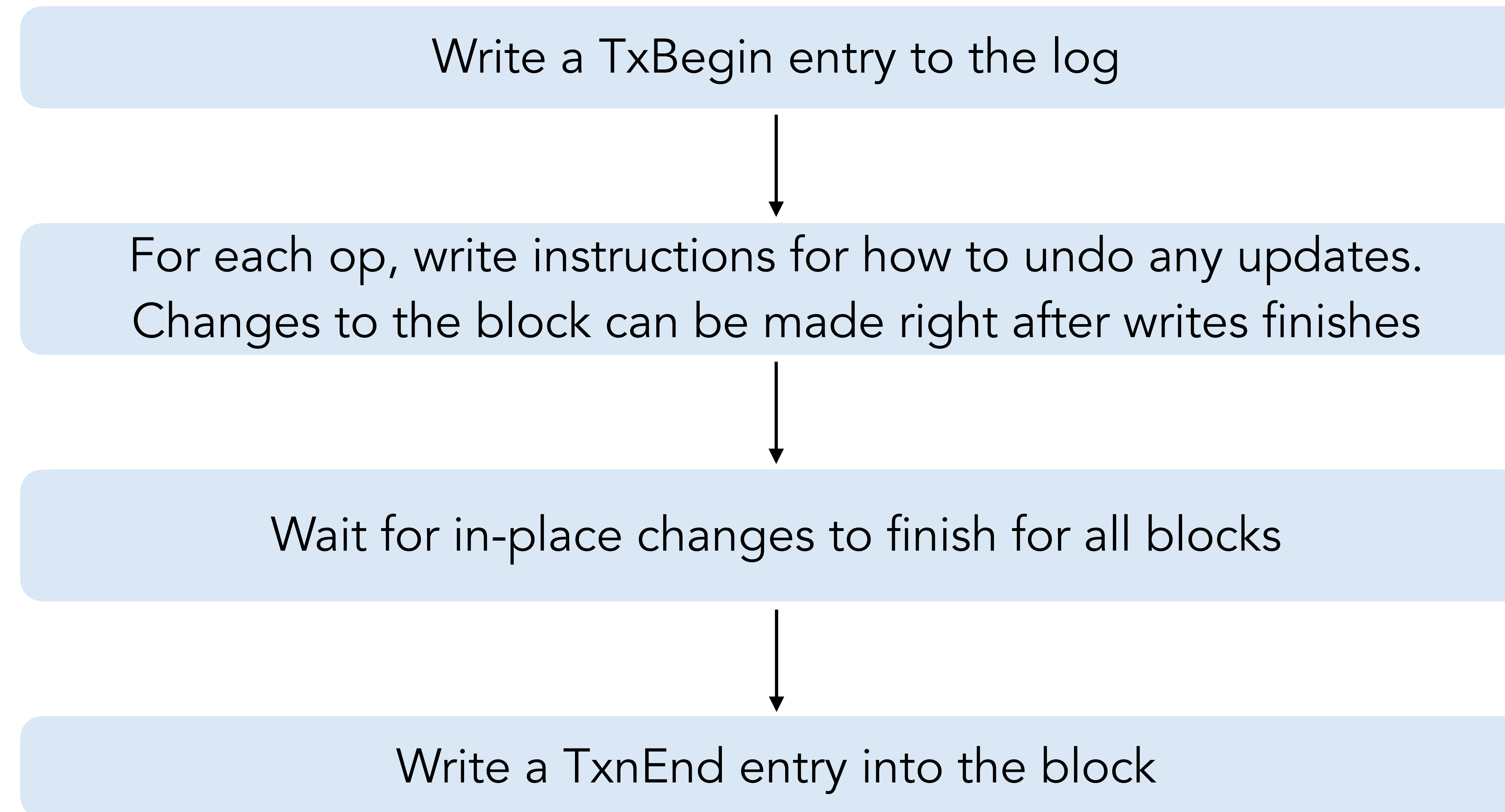
What to log?

Logging can double the amount of data written to the disk
Ext3 and 4 allows user to choose what to log

Default: metadata only (assuming people are fine with data loss after crash)
Can change to force data to be logged w/ metadata

Journaling — undo logging

(Not used in isolation by any file system)



all changes have been written to the actual FS data structures

Journaling — crash recovery from undo logging

Scan to find all uncommitted transactions from **the end of the log**

↓

For each such transaction, check whether undo entry is valid (checksum)

↓

Apply all valid undo entries found

disk back to a consistent state

Benefits

Changes can be checkpoints to disk as soon as the undo log has been updated — useful when the amount of buffer cache is low

Disadvantages

A transaction is not committed until all dirty blocks have been flushed to their in-place targets

Redo logging vs. Undo logging

Benefits

A transaction can commit without all in-place updates (writes to actual disk locations) being completed — useful when in-place updates might be scattered all over the disk

Disadvantages

A transaction's dirty blocks need to be kept in the buffer-cache until the transaction commits and all of the associated journal entries have been flushed to disk.
This might increase memory pressure.

Benefits

Changes can be checkpoints to disk as soon as the undo log has been updated — useful when the amount of buffer cache is low

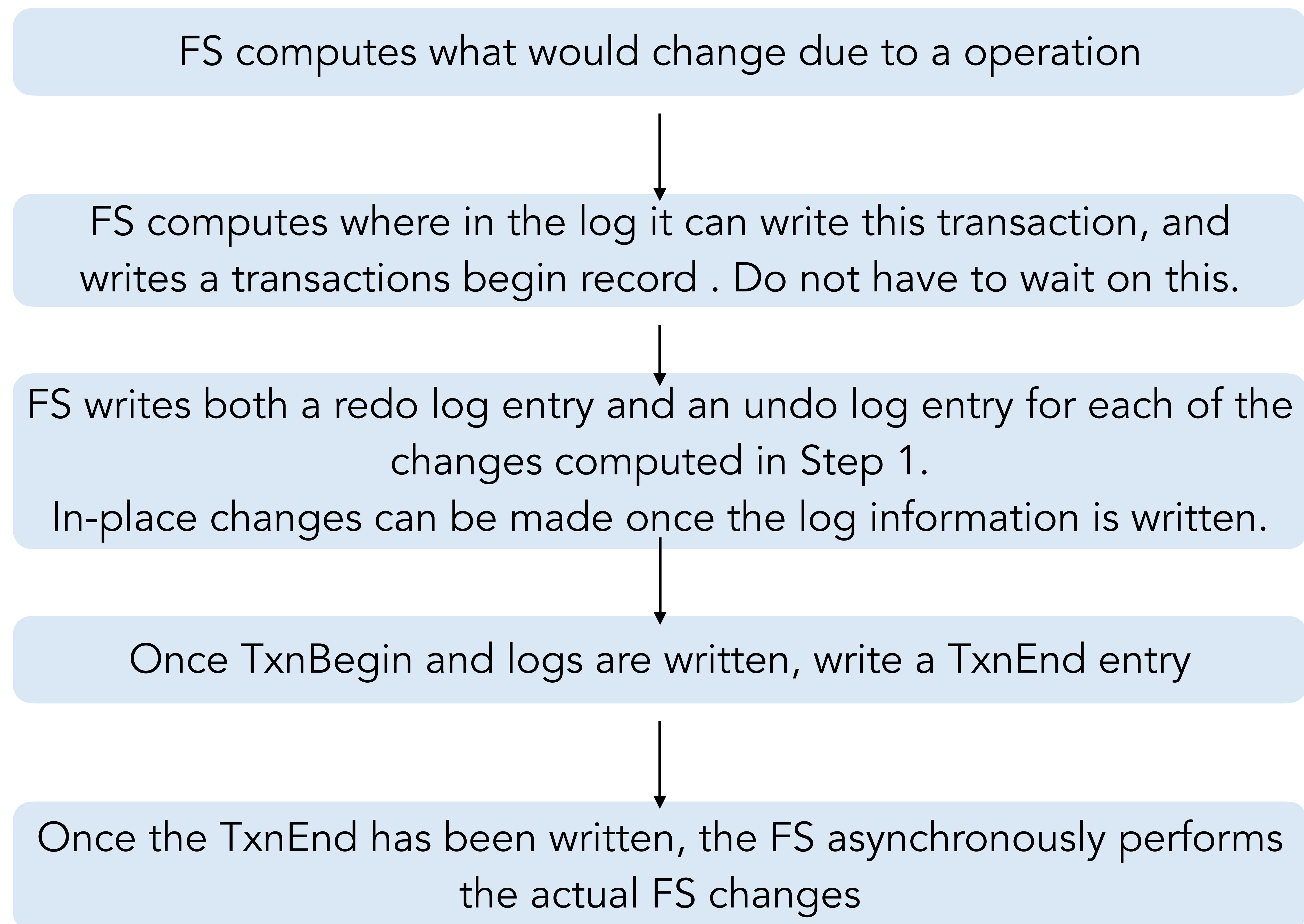
Disadvantages

A transaction is not committed until all dirty blocks have been flushed to their in-place targets

Combining Redo/Undo Logging (Done by NFTS)

Goal: allow dirty buffers to be flushed as soon as their associated journal entries are written. Transactions are committed as soon as logging is done

Reduce memory pressure when necessary, and have greater flexibility when scheduling disk writes



Journaling — crash recovery from redo+undo logging

Step 1: Redo pass

FS starts scanning from the **beginning of the log**

↓

Every time it finds a TxnBegin entry, it looks for the corresponding TxnEnd entry

↓

If matching (TxnBegin, TxnEnd) found, FS checkpoints the changes

↓

Recovery is completed once the entire log is scanned

Step 2: Undo pass

Scan to find all uncommitted transactions from **the end of the log**

↓

For each such transaction, check whether undo entry is valid (checksum)

↓

Apply all valid undo entries found

disk back to a consistent state

Designed for a time when the same Operating System ran on machines with very little memory (8-32MB), and also on "big-iron" servers with lots of memory (1GB+).
This was an attempt to get the best of both worlds.

RPC (Remote Procedure Call)

A mechanism that allow programs to call procedures on other computers across a network

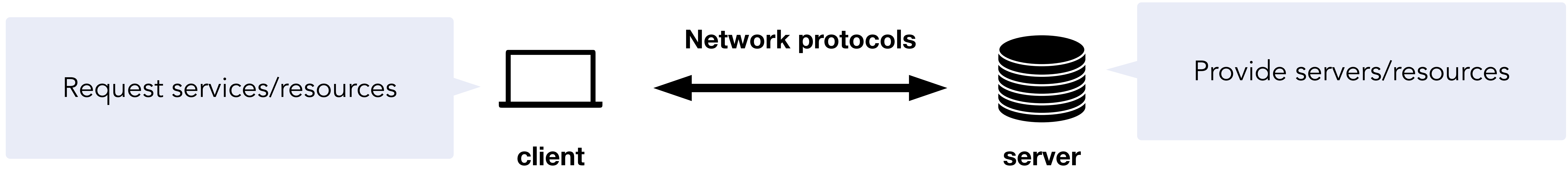
Make remote function calls **appear similar** to local ones

Access remote services/resources
without worrying about distributed/network issues
But, more things might go wrong
(failures, network latency, distributed transactions)

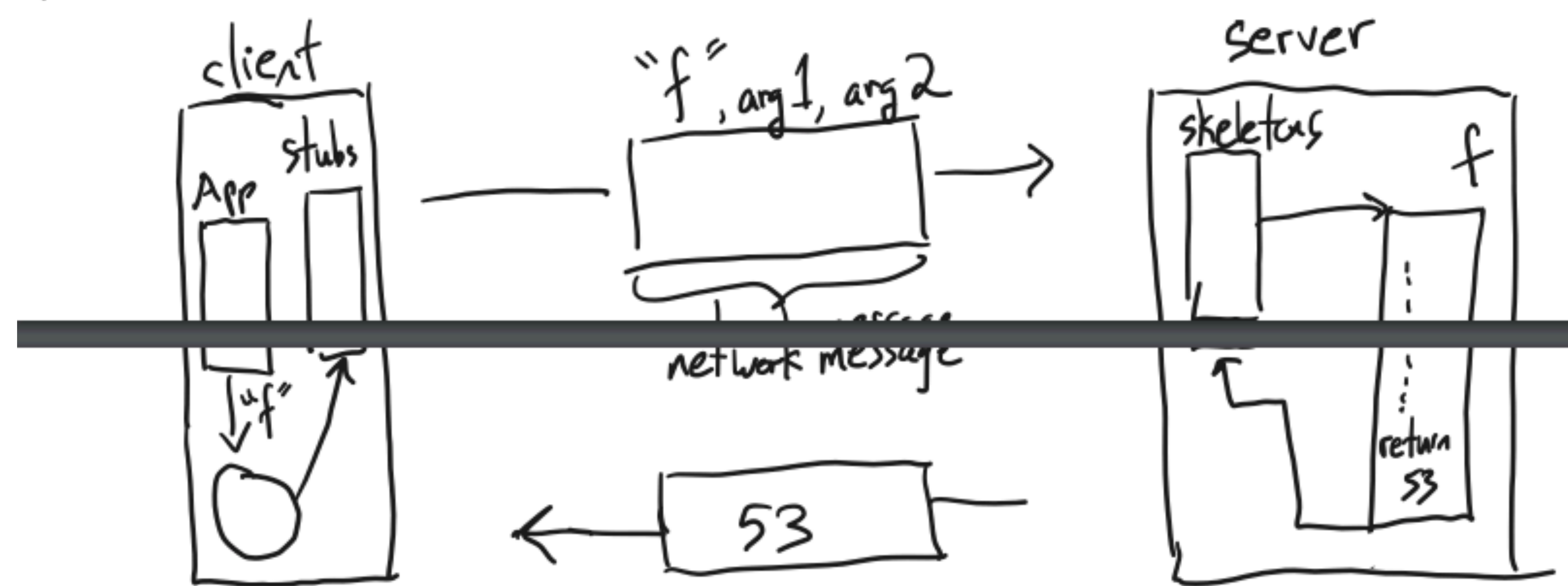
Direct memory access (fast!)
Predictable performance
But, only works w/ local resources

One of the building blocks for client/server systems

Client/server system



Example: web browser/servers, database client/servers, …



Networked file systems

Look like a file system to the application,
but the data potentially stored on another machine
Reads/writes must go over the network

Benefits

Easy to share if files available on multiple machines
Easier to administer servers than clients
Access way more data than fits on your local disk
Network + remote buffer cache faster than local disk (in certain cases)

Disadvantages

Network + remote disk slower than local disk
Network or server fail even when client is still running
Complexity and security issues

NFS: Network File System

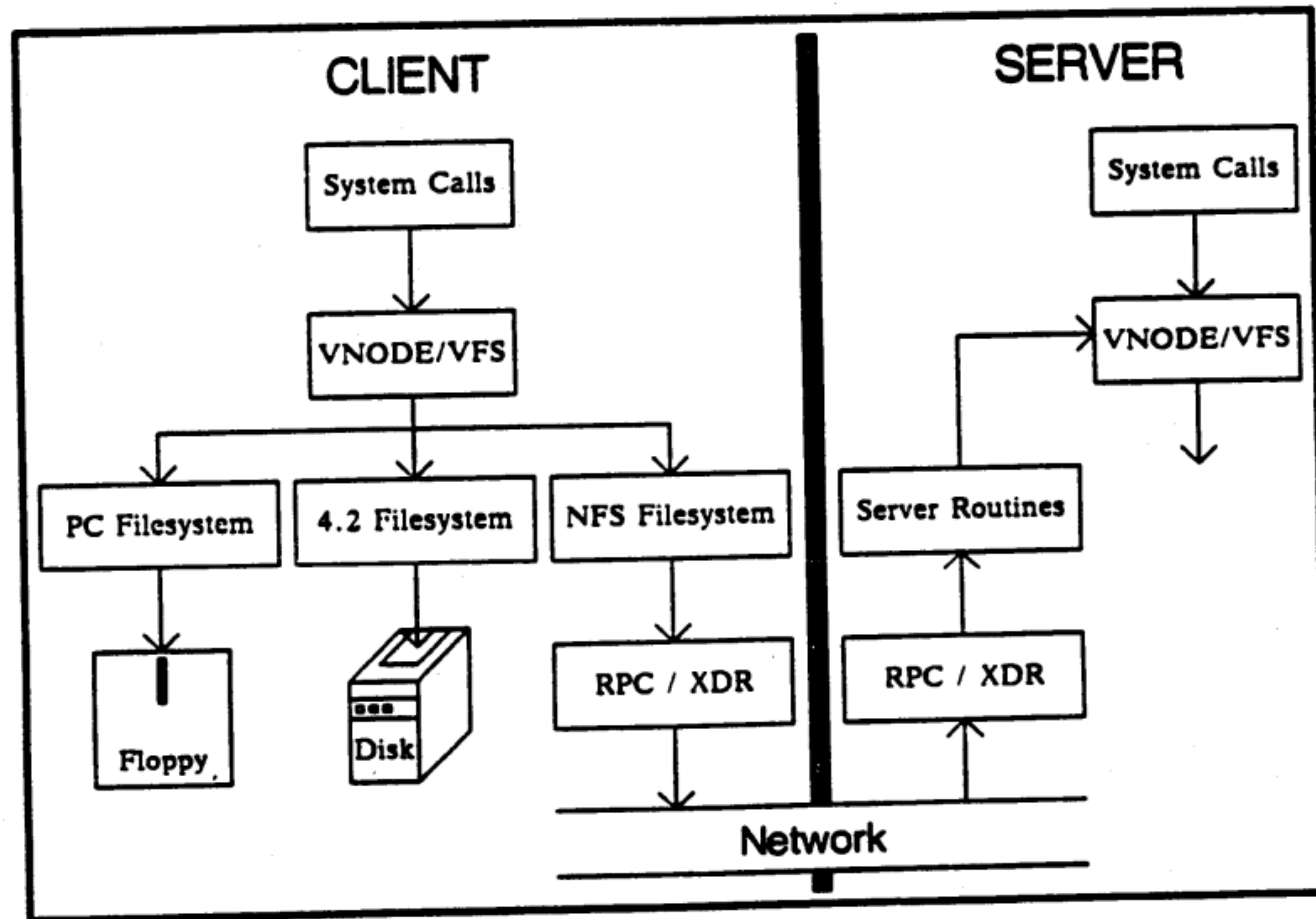


Figure 1

https://web.stanford.edu/class/cs240/readings/nfs.pdf

NFS implements vnode operations through RPC

```
open("/usr/jo/lab1.c", …)

                    Lookup("/usr")

                    fh1 = (FS id, i#, gen#)

                    Lookup(fh1, "jo")

                    fh2 = (FS id, i#, gen#)

                    Lookup(fh2, "lab1.c")

                    fh3 = (FS id, i#, gen#)

  write(fd, buf, sz);

                    Write(fh3, offset, data, size)

                    return code
```

Why not embed file name in file handle?

(file names can change; would mess everything up. client needs to use an identifier that's invariant across such renames.)

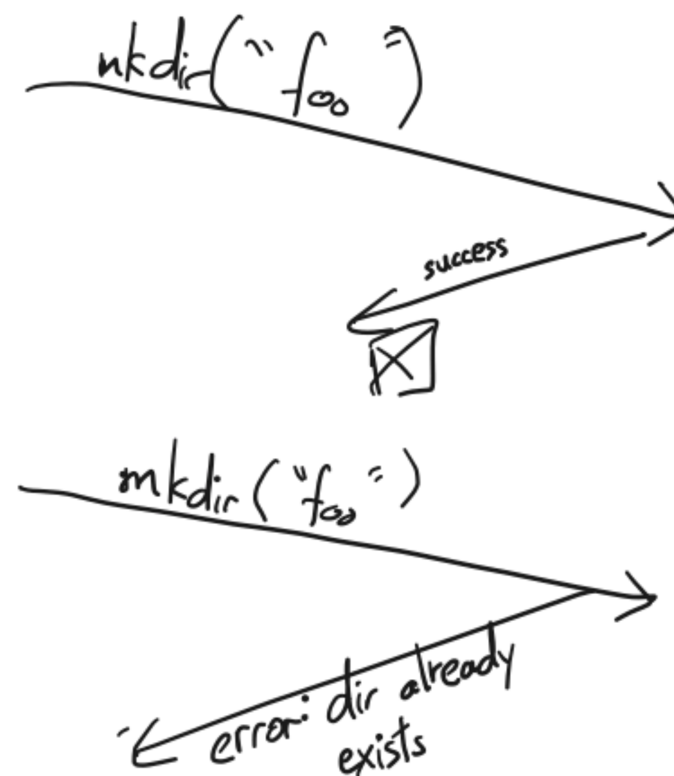
How does client know what file handle to send?

(stored with vnode)

Statelessness of NFS

Every network protocol request contains all of the information needed to carry out that request, without relying on anything remembered from previous protocol requests.

Are all NFS operations idempotent?
(i.e., performing the op multiple times has the same effect as performing it once)



Benefits	simplifies implementation, failure recovery
Disadvantages	mess up w/ traditional unix semantics

Transparency and non-traditional Unix semantics

Transparency requires that the system calls **mean** the same things