LFS, DISTRIBUTED SYSTEMS

Shivaram Venkataraman

CS 537, Spring 2019

ADMINISTRIVIA

Project 5: Due April 29. Last Project!

Project 4a, 4b grading update

Regrades status

Peer mentors for next semester! https://forms.gle/h7zXQidTP4QxiwVD8

COURSE FEEDBACK

https://aefis.wisc.edu

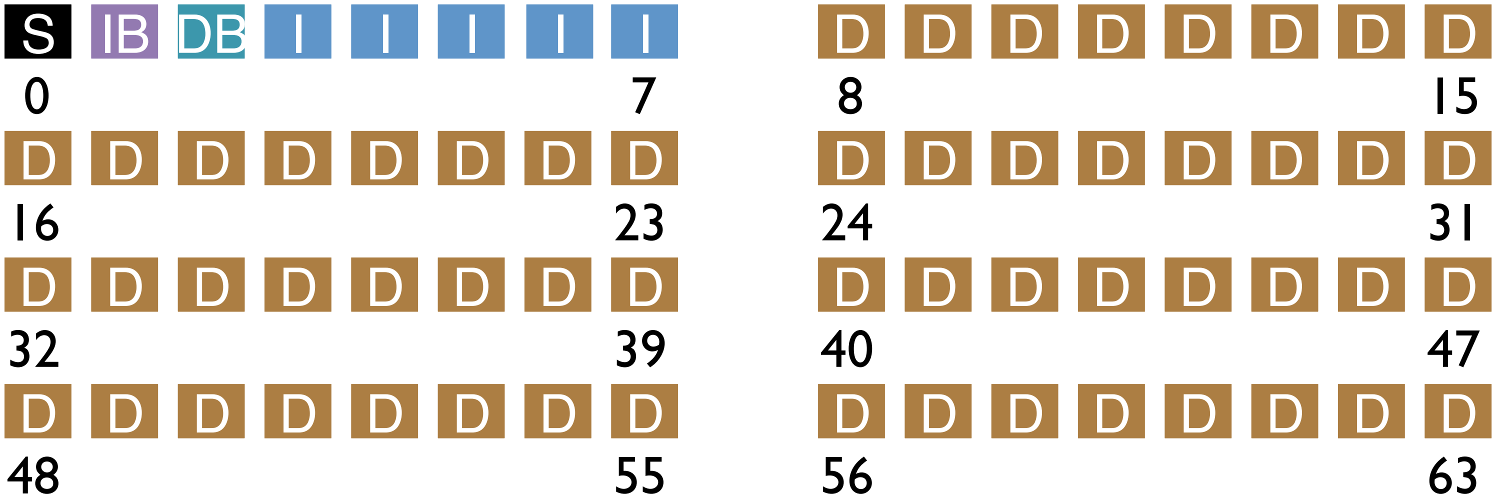
AGENDA / LEARNING OUTCOMES

How to design a filesystem that performs better for small writes?

What are the design principles for systems that operate across machines?

RECAP

FS STRUCTS


S
IB
DB
I
I
I
I
I
0
7
D D D D D D D D
8
15
D D D D D D D D
16
23
D D D D D D D D
24
31
D D D D D D D D
32
39
D D D D D D D D
40
47
D D D D D D D D
48
55
D D D D D D D D
56
63

CRASH CONSISTENCY SUMMARY

Crash consistency: Important problem in filesystem design!

Two main approaches

FSCK:

- Fix file system image after crash happens
- Too slow and only ensures consistency

Journaling

- Write a transaction before in-place updates
- Checksum, batching
- Ordered journal avoids data writes

LOG STRUCTURED FILE SYSTEM (LFS)

LFS PERFORMANCE GOAL

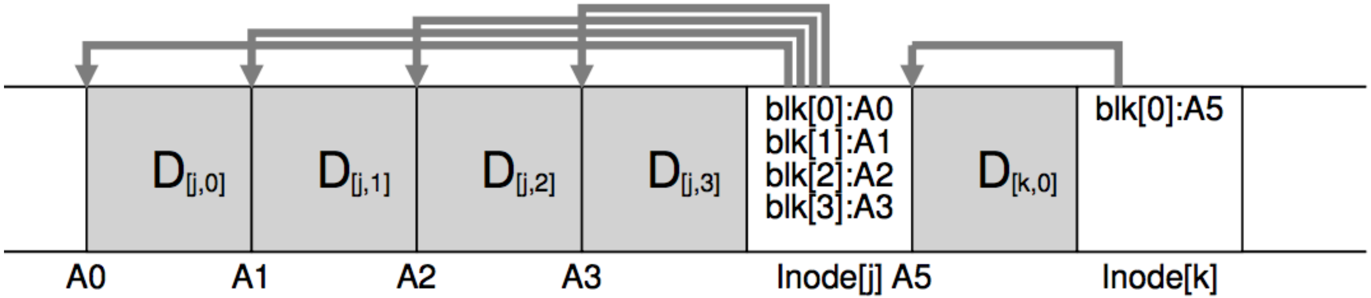
Motivation:

- Growing gap between sequential and random I/O performance
- RAID-5 especially bad with small random writes

Idea: use disk purely sequentially

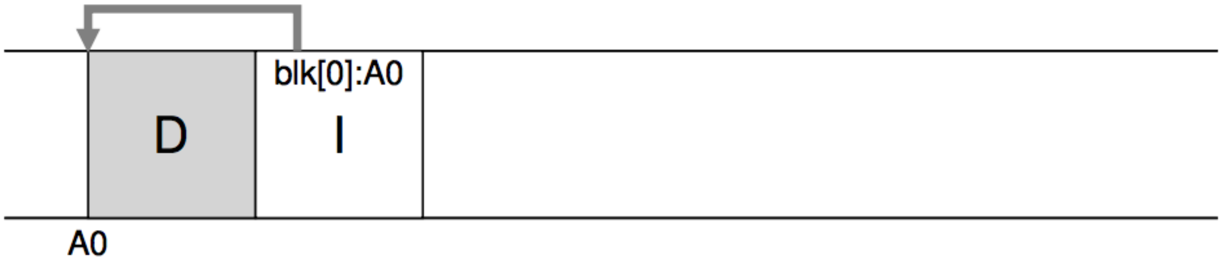
Design for writes to use disk sequentially – how?

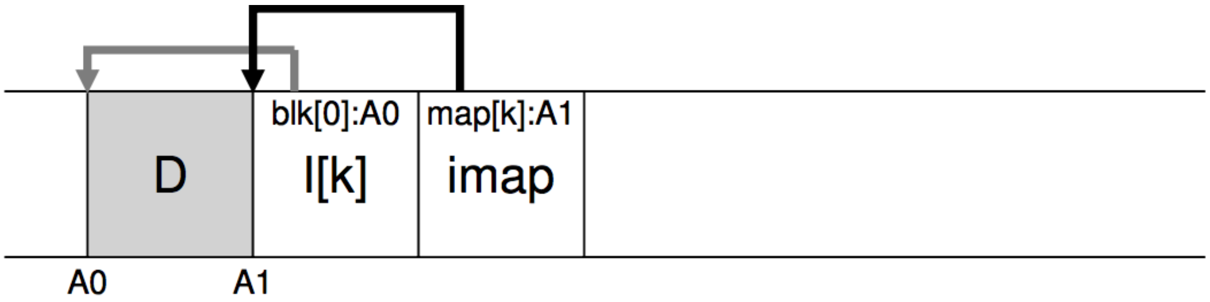
WRITES



blk[0]:A0
blk[1]:A1
blk[2]:A2
blk[3]:A3
blk[0]:A5
$D_{[j,0]}$
$D_{[j,1]}$
$D_{[j,2]}$
$D_{[j,3]}$
$D_{[k,0]}$
A0
A1
A2
A3
Inode[j] A5
Inode[k]

IMAP EXPLAINED





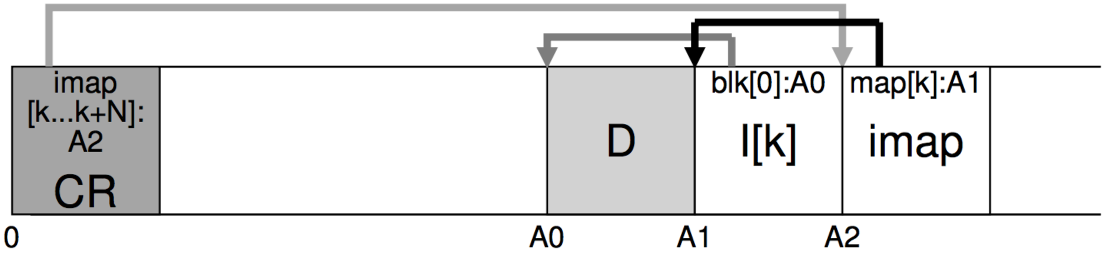
CHECKPOINT REGION

How do we find the imap, given pieces of it are also spread across the disk?

Checkpoint Region (CR):

- fixed region at say start of the disk
- pointers to the latest pieces of the inode map
- Updated every 30s or so, performance is not affected

READING IN LFS



1. Read the Checkpoint region
2. Read all imap parts, cache in mem
3. To read a file:
    1. Lookup inode location in imap
    2. Read inode
    3. Read the file block

BUNNY 20

https://tinyurl.com/cs537-sp19-bunny19

You are given the traffic stream of writes to disk performed by LFS.
Before these writes, you can assume the file system only had a root directory
You can also assume that a single inode takes up an entire block.

(a) Segment written starting at disk address 100, in a segment of size 4:

```
block 100: [("." 0), (".." 0), ("foo" 1)]    // a data block
block 101: [size=1,ptr=100,type=d]           // an inode
block 102: [size=0,ptr=-,type=r]             // an inode
block 103: [imap: 0->101,1->102]             // a piece of the imap
```

What file system operation(s) led to this segment write?

BUNNY 20

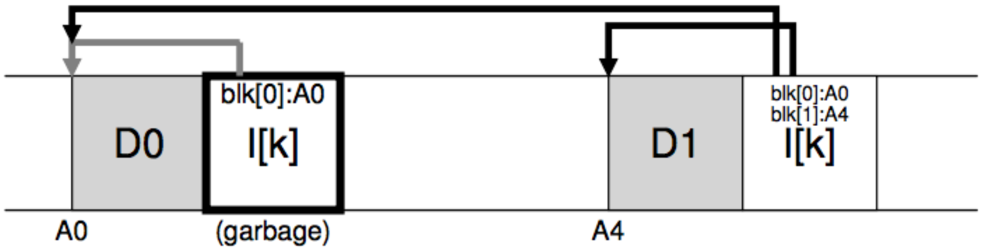
https://tinyurl.com/cs537-sp19-bunny19

(b) Segment written to disk address 104, in a segment of size 4:

```
block 104: [SOME DATA]                        // a data block
block 105: [SOME DATA]                        // a data block
block 106: [size=2,ptr=104,ptr=105,type=r]    // an inode
block 107: [imap: 0->101,1->106]              // a piece of the imap
```

What file system operation(s) led to this segment write?

GARBAGE COLLECTION



WHAT TO DO WITH OLD DATA?

Old versions of files → garbage

Approach 1: garbage is a feature!

- Keep old versions in case user wants to revert files later
- Versioning file systems
- Example: Dropbox

Approach 2: garbage collection

Garbage Collection

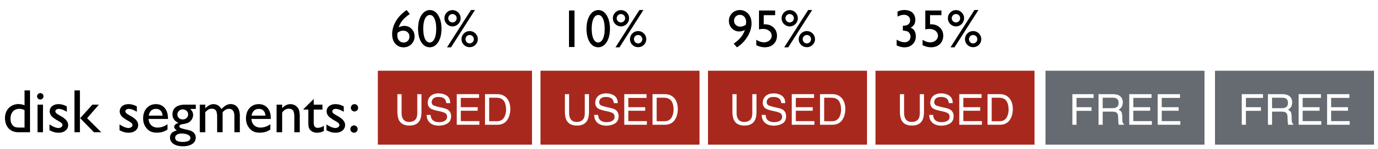
Need to reclaim space:

1. When no more references (any file system)
2. After newer copy is created (COW file system)

LFS reclaims **segments** (not individual inodes and data blocks)

- Want future overwites to be to sequential areas
- Tricky, since segments are usually partly valid

GARBAGE COLLECTION



compact 2 segments to one

When moving data blocks, copy new inode to point to it
When move inode, update imap to point to it

Garbage Collection

General operation:

Pick M segments, compact into N (where N < M).

Mechanism:

How does LFS know whether data in segments is valid?

Policy:

Which segments to compact?

GARBAGE COLLECTION MECHANISM

Is an inode the latest version?

- Check imap to see if this inode is pointed to
- Fast!

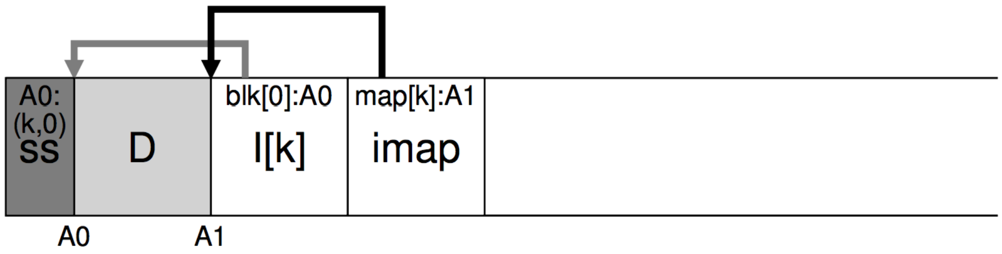
Is a data block the latest version?

- Scan ALL inodes to see if any point to this data
- Very slow!

How to track information more efficiently?

- Segment summary: For every data block in segment, store its inode number (which file) and offset (which block of file)

SEGMENT SUMMARY




```
(N, T) = SegmentSummary[A];

inode = Read(imap[N]);

if (inode[T] == A)
    // block D is alive
else
    // block D is garbage
```

GARBAGE COLLECTION POLICY

General operation:

Pick M segments, compact into N (where N < M).

Mechanism:

Use segment summary, imap to determine liveness

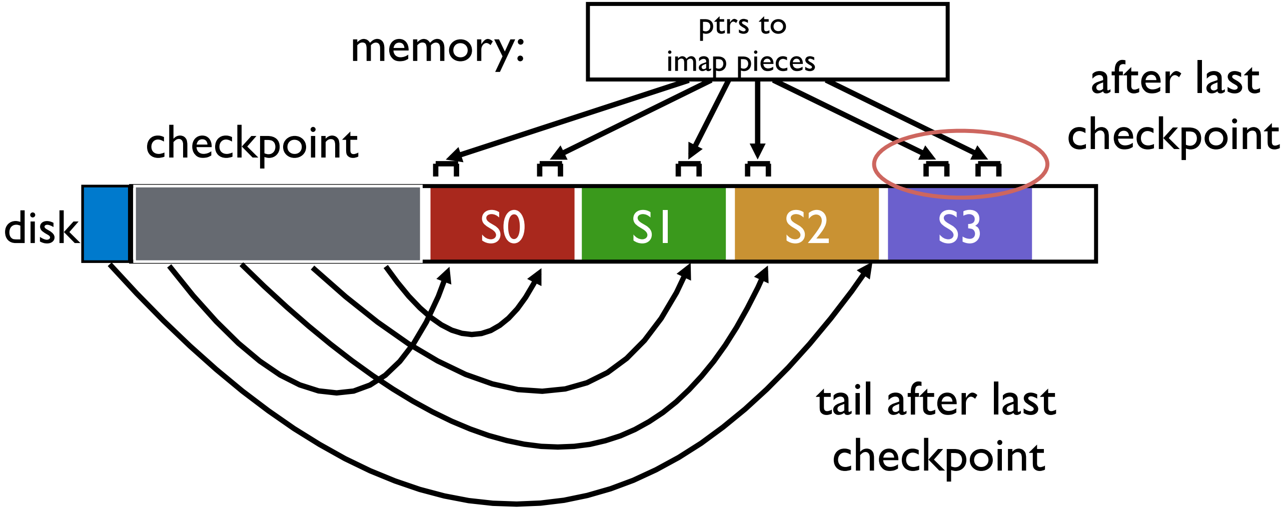
Policy:

Which segments to compact?

- clean most empty first
- clean coldest (ones undergoing least change)
- more complex heuristics…

CRASH RECOVERY

What data needs to be recovered after a crash? Need imap (lost in volatile memory)



Checkpoint Summary

Checkpoint occasionally (e.g., every 30s)

Upon recovery:

- read checkpoint to find most imap pointers and segment tail
- find rest of imap pointers by reading past tail

What if crash <u>during</u> checkpoint?

CHECKPOINT STRATEGY

Have two checkpoint regions

Only overwrite one checkpoint at a time

Use checksum/timestamps to identify newest checkpoint

Persistence Summary

Managing I/O devices is a significant part of OS!

Disk drives: storage media with specific geometry

Filesystems: OS provided API to access disk

Simple FS: FS layout with SB, Bitmaps, Inodes, Datablocks

FFS: Split simple FS into groups. Key idea: put inode, data close to each other

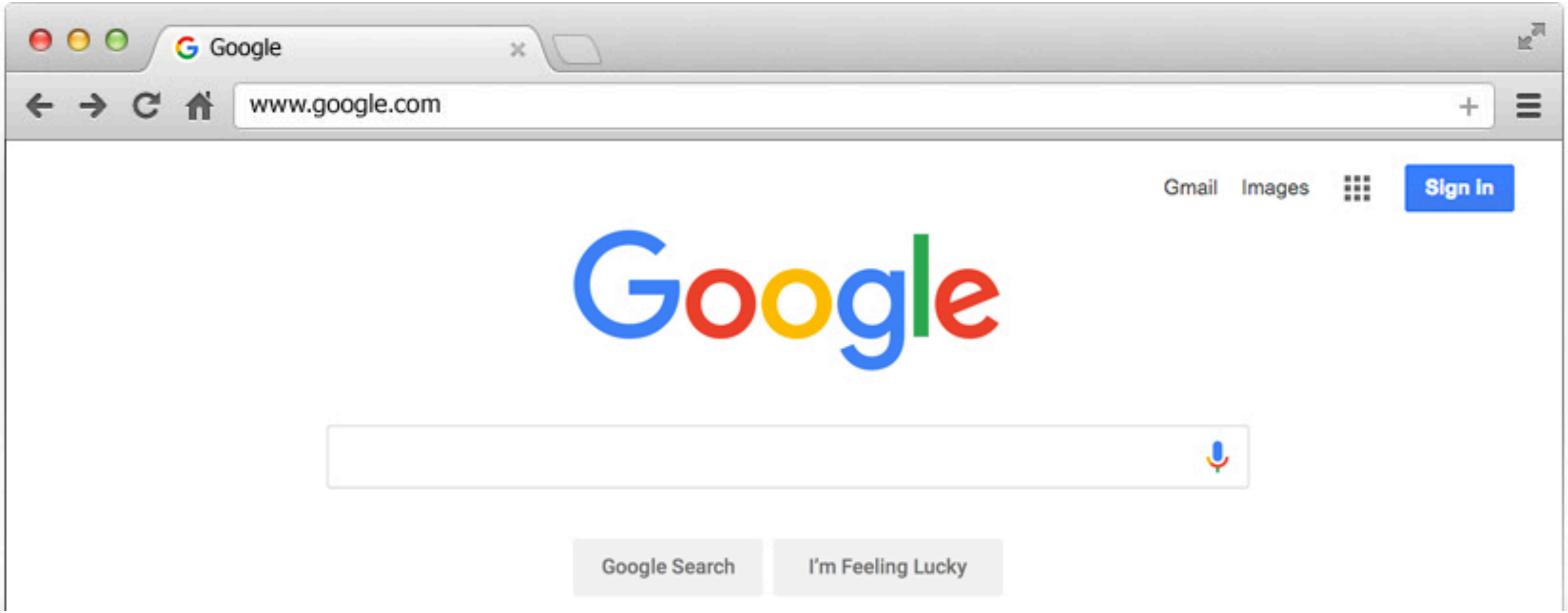
LFS: Puts data where it's fastest to write, hope future reads cached in memory

https://www.eecs.harvard.edu/~margo/papers/usenix95-lfs/supplement/

FSCK, Journaling

DISTRIBUTED SYSTEMS

HOW DOES GOOGLE SEARCH WORK?



WHAT IS A DISTRIBUTED SYSTEM?

A distributed system is one where a machine I've never heard of can cause my program to fail.
— *Leslie Lamport*

Definition:
More than 1 machine working together to solve a problem

Examples:

- client/server: web server and web client
- cluster: page rank computation

WHY GO DISTRIBUTED?

More computing power

More storage capacity

Fault tolerance

Data sharing

NEW CHALLENGES

System failure: need to worry about partial failure

Communication failure: links unreliable

- bit errors
- packet loss
- node/link failure

Why are network sockets less reliable than pipes?

COMMUNICATION OVERVIEW

Raw messages: UDP

Reliable messages: TCP

Remote procedure call: RPC

RAW MESSAGES: UDP

UDP : User Datagram Protocol

API:

- reads and writes over socket file descriptors
- messages sent from/to ports to target a process on machine

Provide minimal reliability features:

- messages may be lost
- messages may be reordered
- messages may be duplicated
- only protection: checksums to ensure data not corrupted

RAW MESSAGES: UDP

Advantages

- Lightweight
- Some applications make better reliability decisions themselves (e.g., video conferencing programs)

Disadvantages

- More difficult to write applications correctly

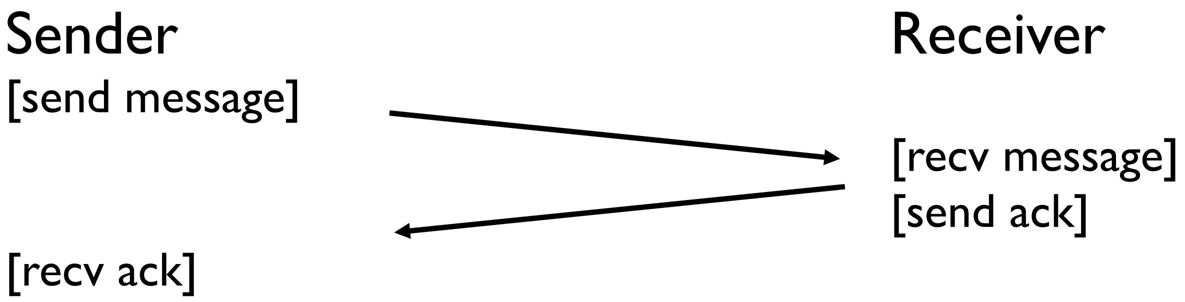
RELIABLE MESSAGES: LAYERING STRATEGY

TCP: Transmission Control Protocol

Using software to build

reliable logical connections over unreliable physical connections

TECHNIQUE #1: ACK



Ack: Sender knows message was received

What to do about message loss?

TECHNIQUE #2: TIMEOUT

Sender

[send message]

[start timer]

… waiting for ack …

[timer goes off]

[send message]

[recv ack]

Receiver

[recv message]

[send ack]

TIMEOUT

How long to wait?

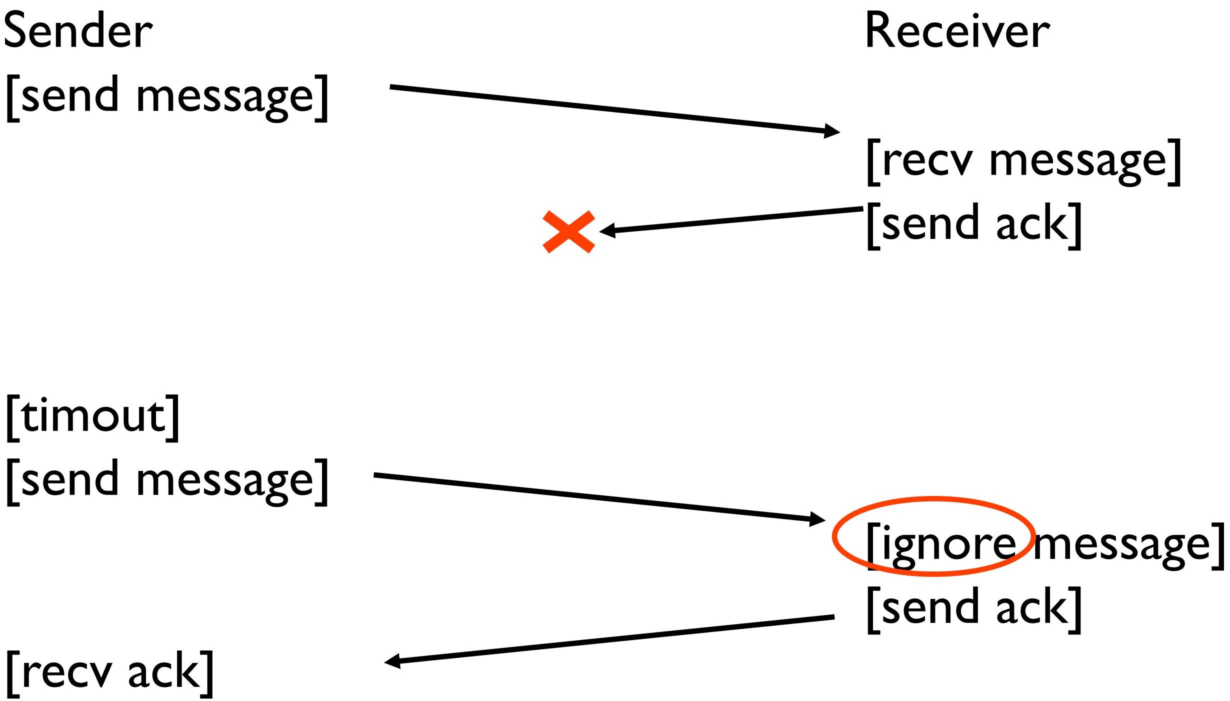
Too long?

- System feels unresponsive

Too short?

- Messages needlessly re-sent
- Messages may have been dropped due to overloaded server. Resending makes overload worse!

LOST ACK PROBLEM



SEQUENCE NUMBERS

Sequence numbers

- senders gives each message an increasing unique seq number
- receiver knows it has seen all messages before N

Suppose message K is received.

- if K <= N, Msg K is already delivered, ignore it
- if K = N + 1, first time seeing this message
- if K > N + 1 ?

TCP

TCP: Transmission Control Protocol

Most popular protocol based on seq nums

Buffers messages so arrive in order

Timeouts are adaptive

COMMUNICATIONS OVERVIEW

Raw messages: UDP

Reliable messages: TCP

Remote procedure call: RPC

RPC

Remote **P**rocedure **C**all

What could be easier than calling a function?

Approach: create wrappers so calling a function on another machine feels just like calling a local function!

RPC

Machine A

```
int main(…) {
	int x = foo(”hello”);
}

int foo(char *msg) {
	send msg to B
	recv msg from B
}
```

Machine B

```
int foo(char *msg) {
	…
}

void foo_listener() {
	while(1) {
		recv, call foo
	}
}
```

RPC

Machine A

```
int main(…) {
        int x = foo(”hello”);
}

int foo(char *msg) {
        send msg to B
        recv msg from B
}
```

client wrapper

Machine B

```
int foo(char *msg) {
        …
}

void foo_listener() {
        while(1) {
                recv, call foo
        }
}
```

server wrapper

RPC TOOLS

RPC packages help with two components

(1) Runtime library

- Thread pool
- Socket listeners call functions on server

(2) Stub generation

- Create wrappers automatically
- Many tools available (rpcgen, thrift, protobufs)

WRAPPER GENERATION

Wrappers must do conversions:

- client arguments to message
- message to server arguments
- convert server return value to message
- convert message to client return value

Need uniform endianness (wrappers do this)

Conversion is called marshaling/unmarshaling, or serializing/deserializing

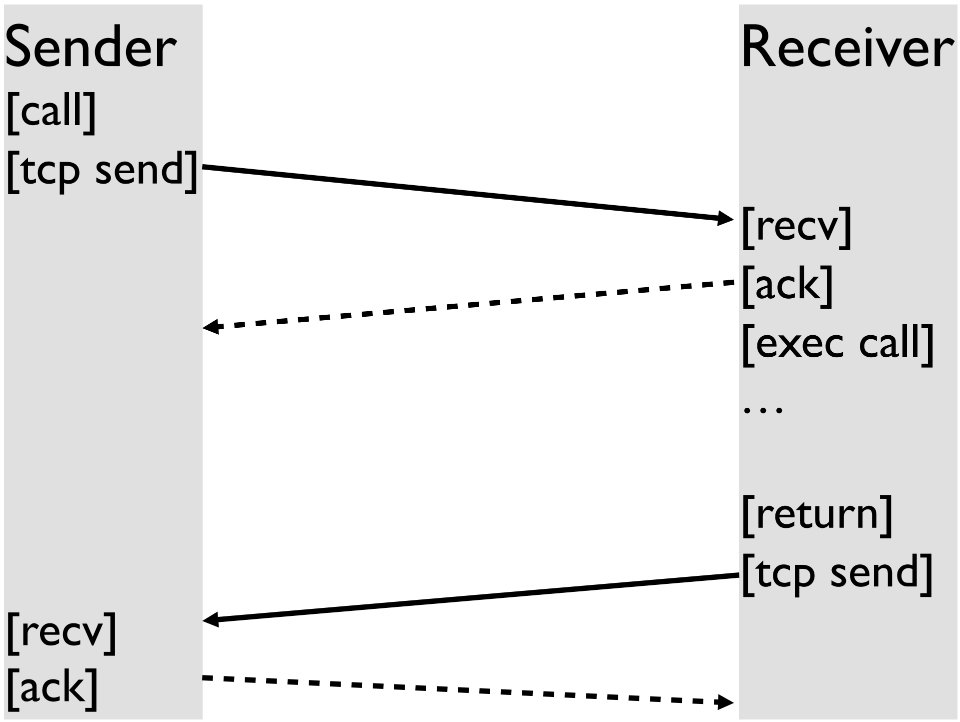
WRAPPER GENERATION: POINTERS

Why are pointers problematic?

Address passed from client not valid on server

Solutions? Smart RPC package: follow pointers and copy data

RPC OVER TCP?



Sender
[call]
[tcp send]
[recv]
[ack]
Receiver
[recv]
[ack]
[exec call]
…
[return]
[tcp send]

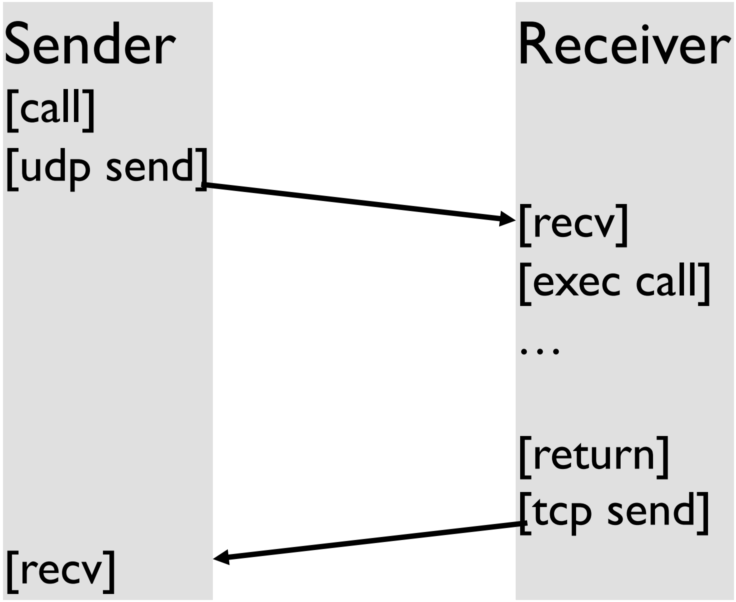
RPC OVER UDP

Strategy: use function return as implicit ACK

Piggybacking technique

What if function takes a long time?

then send a separate ACK



NEXT STEPS

Next class: Distributed NFS

Discussion this week: Worksheet and review, Q&A for P5