DISTRIBUTED SYSTEMS

Shivaram Venkataraman

CS 537, Spring 2023

ADMINISTRIVIA

Project 6 grades

Project 7

Project 8 – Extra credit (4%)

Midterm 3 conflicts

AEFIS feedback

AGENDA / LEARNING OUTCOMES

What are some basic building blocks for systems that span across machines?

RECAP

SSD OPERATIONS

Read a page: Retrieve contents of entire page (e.g., 4 KB)

- Cost: 25—75 microseconds
- Independent of page number, prior request offsets

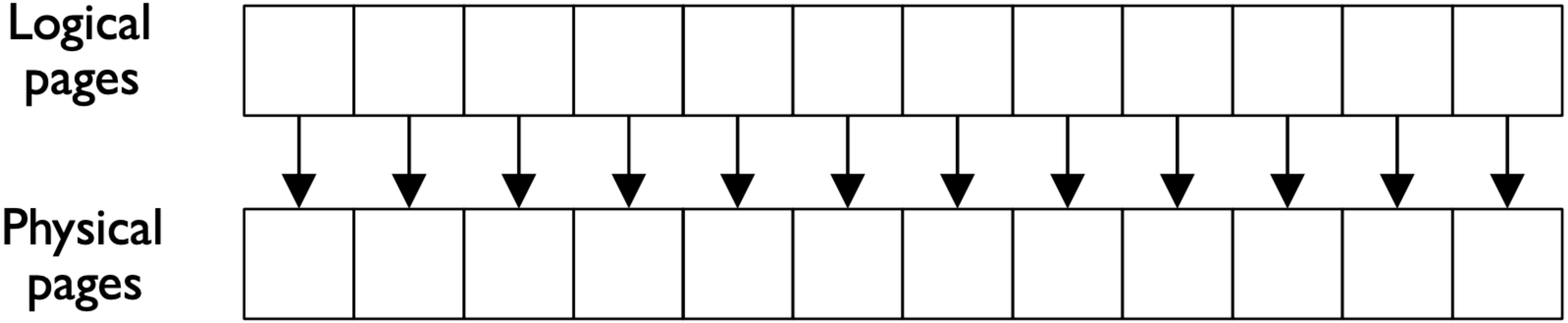
Erase a block: Resets each page in the block to all 1s

- Cost: 1.5 to 4.5 milliseconds
- Much more expensive than reading!
- Allows each page to be written

Program (i.e., write) a page: Change selected 1s to 0s

- Cost is 200 to1400 microseconds
- Faster than erasing a block, but slower than reading a page

FTL: DIRECT MAPPING

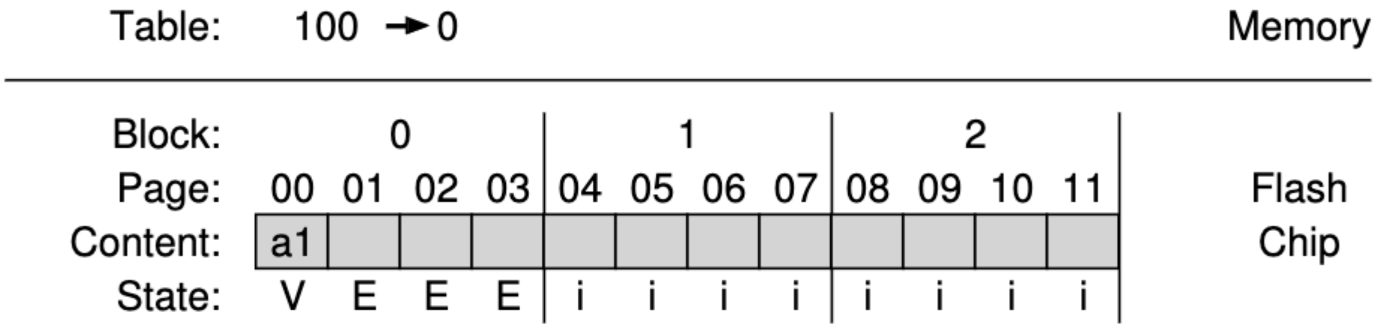


Cons?

Write amplification
No wear-leveling

FTL: LOG-BASED MAPPING

Idea: Treat the physical blocks like a log



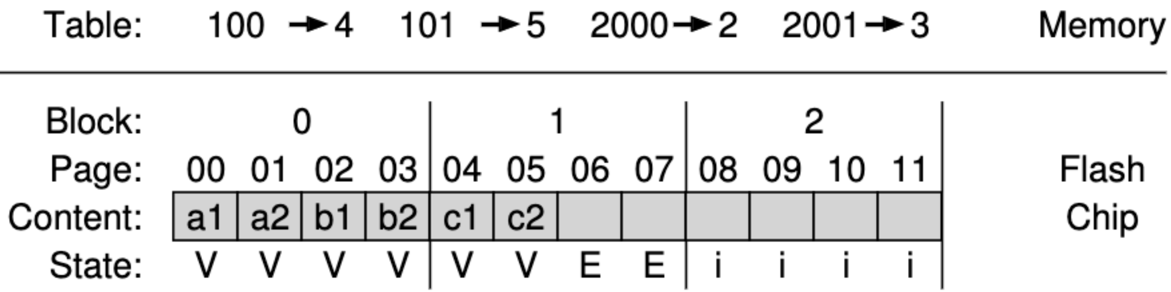
GARBAGE COLLECTION

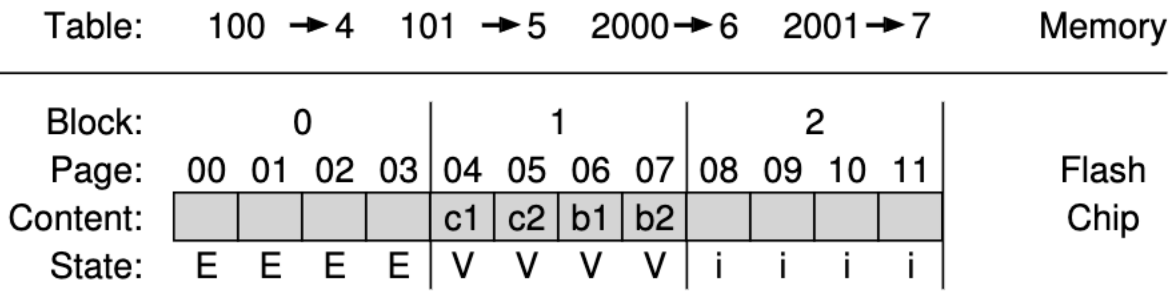
Steps:

Read all pages in physical block

Write out the alive entries to the end of the log

Erase block (freeing it for later use)

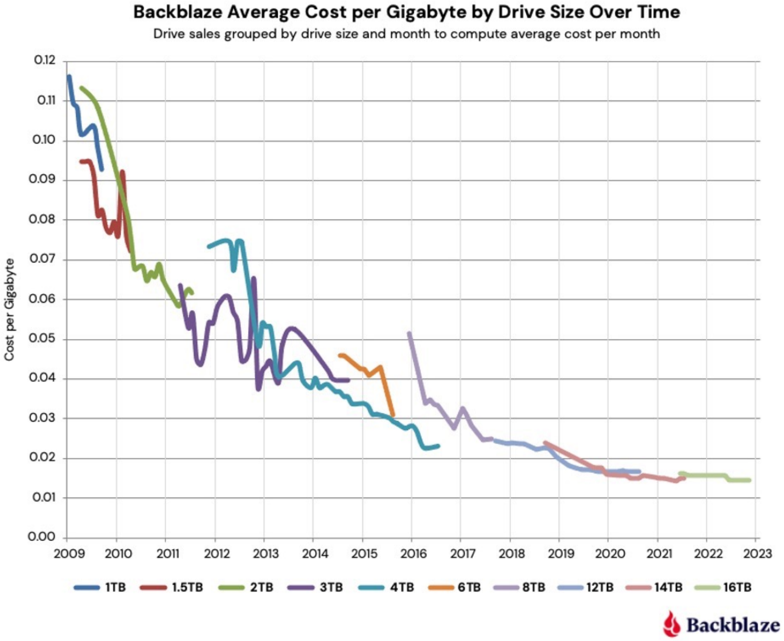


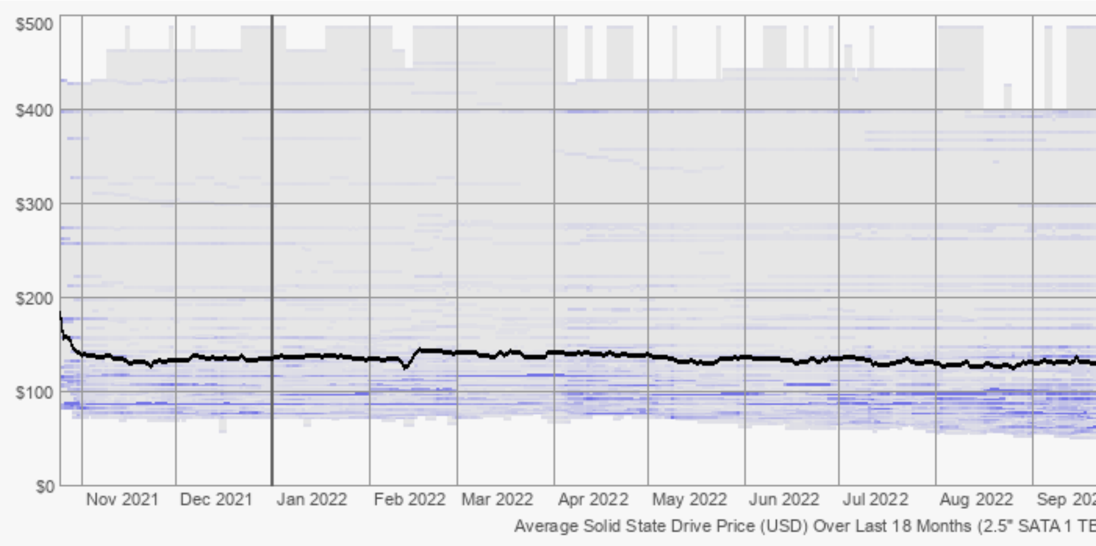


SSD VS HDD PERFORMANCE

Device	Random Reads (MB/s)	Random Writes (MB/s)	Sequential Reads (MB/s)	Sequential Writes (MB/s)
Samsung 840 Pro SSD	103	287	421	384
Seagate 600 SSD	84	252	424	374
Intel SSD 335 SSD	39	222	344	354
Seagate Savvio 15K.3 HDD	2	2	223	223

SSD VS HDD COST





~1.5 cents / GB

1TB ~ $150 on average
~15 cents / GB

PERSISTENCE SUMMARY

Managing I/O devices is a significant part of OS!
Disk drives: storage media with specific geometry
SSDs: Pages, Blocks

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

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 one machine working together to solve a problem

Examples:

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

WHY GO DISTRIBUTED?

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

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

NOT A QUIZ?

Course feedback: https://aefis.wisc.edu

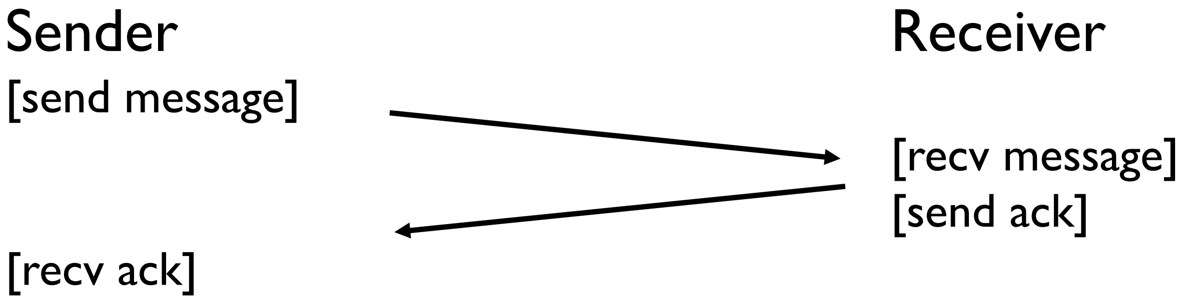
RELIABLE MESSAGES: LAYERING STRATEGY

TCP: Transmission Control Protocol

Using software to build

reliable logical connections over unreliable physical connections

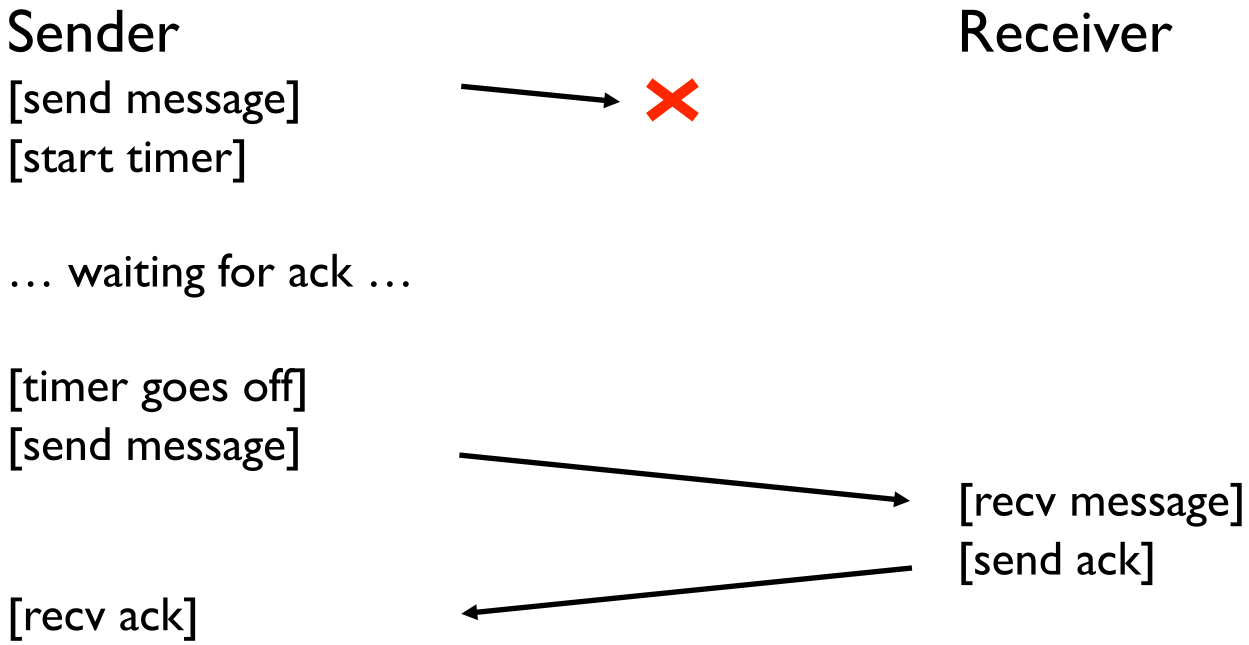
TECHNIQUE #1: ACK



Ack: Sender knows message was received

What to do about message loss?

TECHNIQUE #2: TIMEOUT



TIMEOUT

How long to wait?

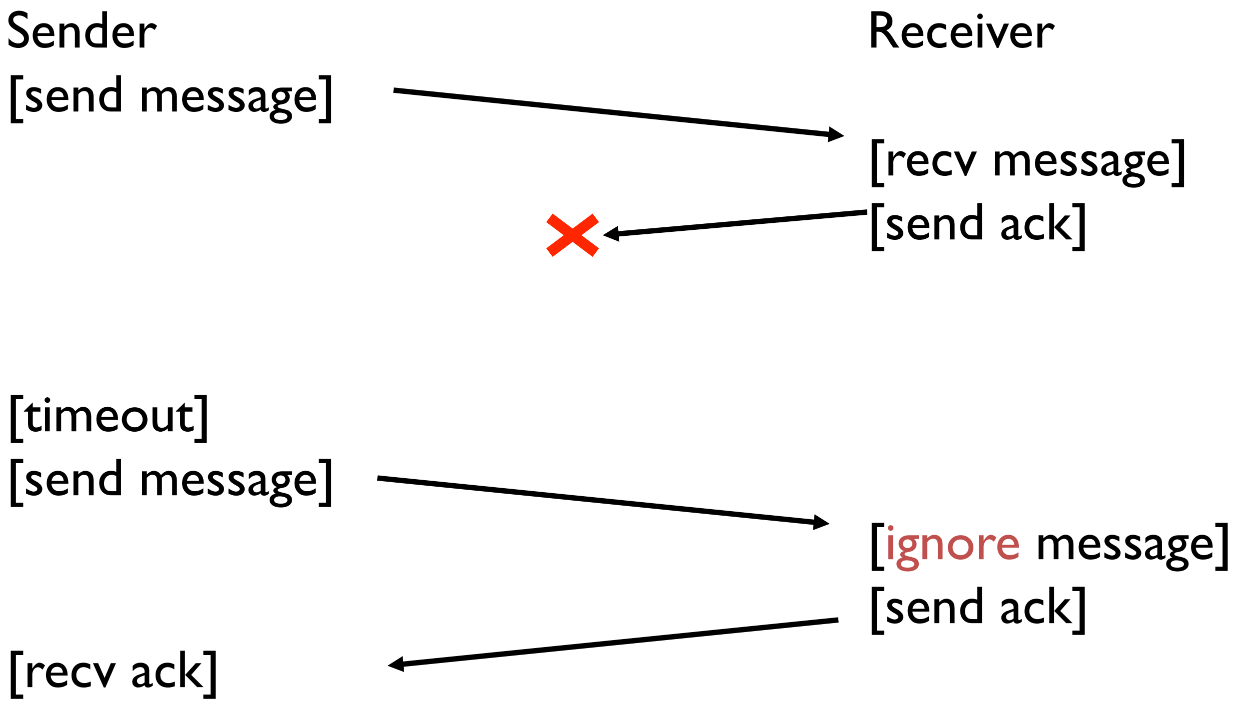
Too long?

- 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”);
}
```

client wrapper

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

Machine B

```
int foo(char *msg) {
	…
}
```

server wrapper

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

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

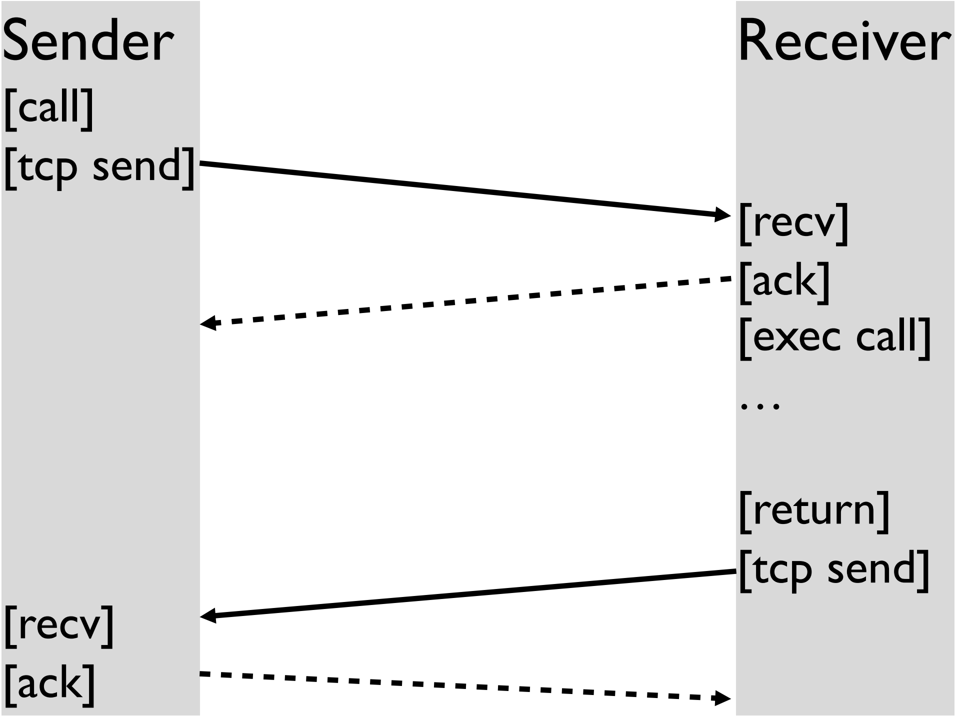
WRAPPER GENERATION: POINTERS

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]

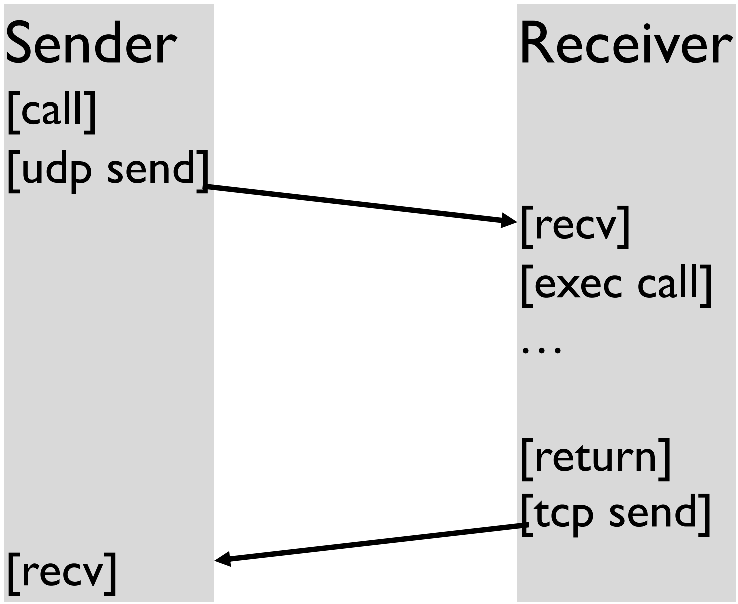
RPC OVER UDP

Strategy: use function return as implicit ACK

Piggybacking technique

What if function takes a long time?

then send a separate ACK



NEXT STEPS

Distributed Filesystems