Hello!

DISTRIBUTED SYSTEMS

Shivaram Venkataraman

CS 537, Fall 2024

ADMINISTRIVIA

Project 5 grades → Piazza

Project 6 deadline → Friday

HelioCampus feedback ~ 15%

AGENDA / LEARNING OUTCOMES

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

RECAP

MULTI QUEUE SCHEDULING

Single Queue

CPU — lock, pick up next, unlock

(RR)

Maintain a queue per CPU
Within each queue, use existing scheduling algorithms

Scalability – Good!

Cache affinity – Good!

Load balance – How?

LINUX: COMPLETELY FAIR SCHEDULER (CFS)

Similar approach to stride scheduler (remember P4!?)

Goal: Divide a CPU using the concept of virtual runtime (vruntime)

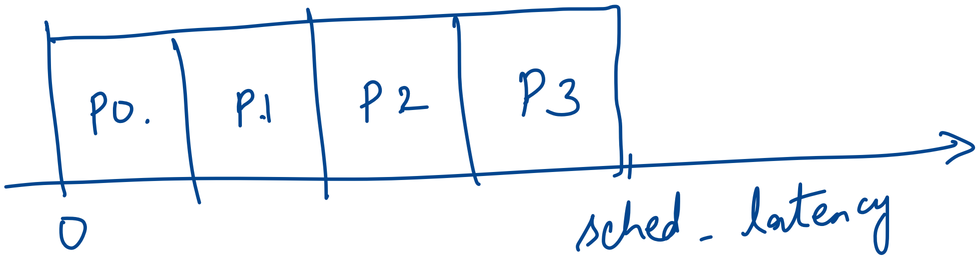
Approach: Pick the process which has the lowest vruntime

When to switch processes:

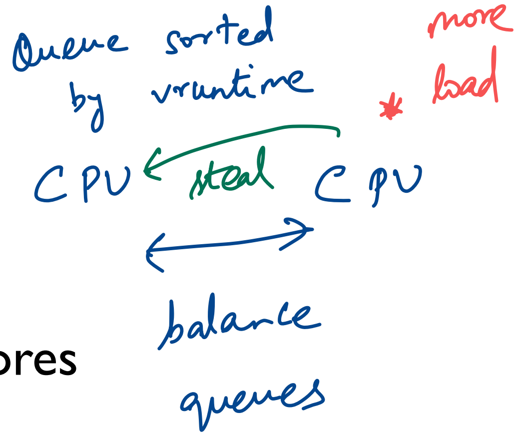
sched_latency – how long before switch ("fairness window")

min_granularity

niceness or priority



CFS: LOAD BALANCING

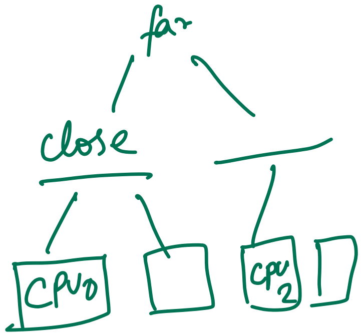


Periodically (e.g., 4ms) steal work from other cores

When stealing work, even out the load between the two cores

Topology awareness while work stealing

- Try to steal work more frequently from cores that are "close" vs. cores that are "remote" (e.g., on a remote NUMA node)



Load difference is small (less than 25% in practice), no load balancing closer to CPU 0 gets more threads

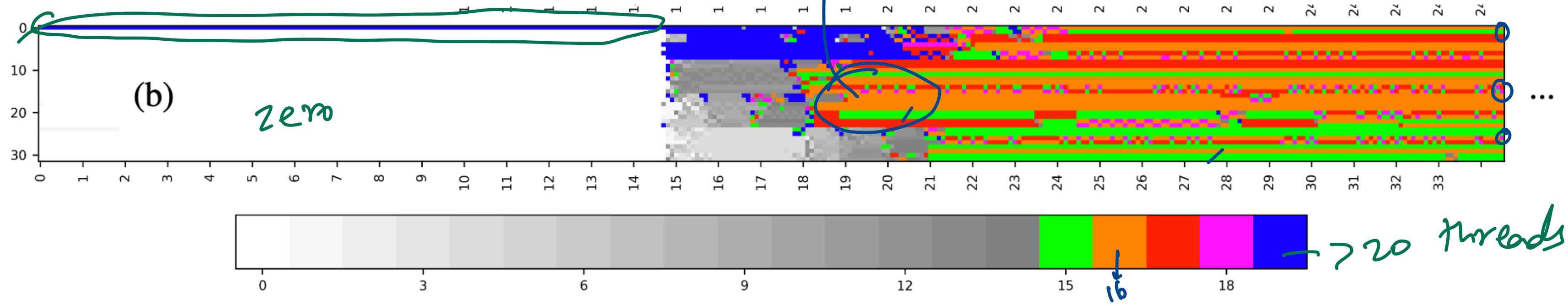


Figure 6: Number of threads per core over time on (a) ULE and (b) CFS. Each line represents a core (32 in total), time passes on the x-axis (in seconds), and colors represent the numbers of threads on the core. Thread counts below 15 are represented in shades of grey. Threads are pinned on core 0 for the first 14.5 seconds of the execution.

Start 512 spinning threads on core 0. Let load balancer work

Pinning: force the OS to run thread on a core

Unpin: OS can make decisions to move thread

→ Not perfectly load balanced

512 / 32 = 16 threads per core

ULE (BSD Scheduler)

MLFQ approach

Three queues per core: interactive, batch and idle threads

Calculate interactivity based on last 5 seconds

Inside a queue, sort threads by priority (niceness level)

First search interactive queue, then batch queue.

ULE (BSD SCHEDULER)

Periodic load balancing

Aims to even out the number of threads per core (not load)

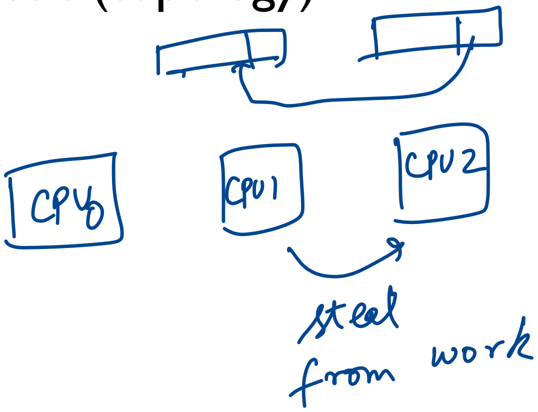
↳ CFS

Choosing a core for a newly created thread: affinity heuristic (topology)

Periodic load balancing only by core 0.

a thread from the most loaded core, the (donor)

to the less loaded core, the (receiver)



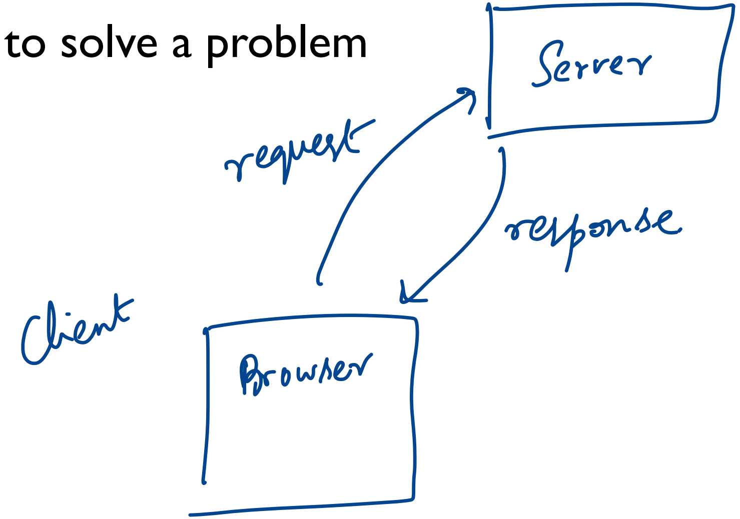
centralized approach

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 → Turing award

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



Examples:

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

of www

WHY GO DISTRIBUTED?

→ want to get some data which is not on your machine

→ reliability → lab machines

→ scale up → processor power! 32 x 10 → faster
→ storage . . .

WHY GO DISTRIBUTED?

More computing power

More storage capacity

Fault tolerance

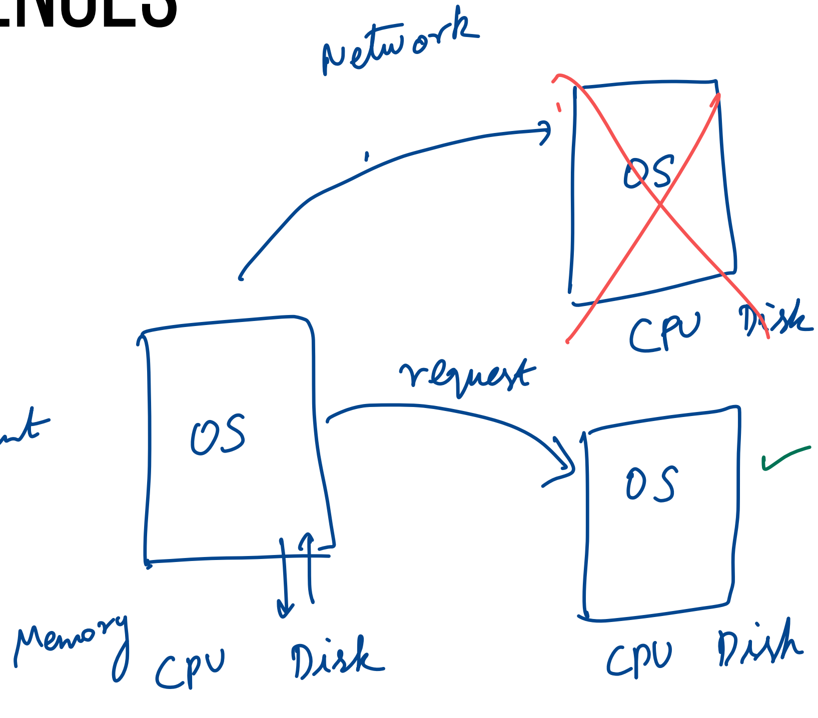
Data sharing

NEW CHALLENGES

System failure: need to worry about partial failure

Communication failure: links unreliable

- bit errors
- packet loss → message from client to server ↳ part of message is lost
- node/link failure ↓ subset of machines / links that can fail



COMMUNICATION OVERVIEW

Raw messages: UDP

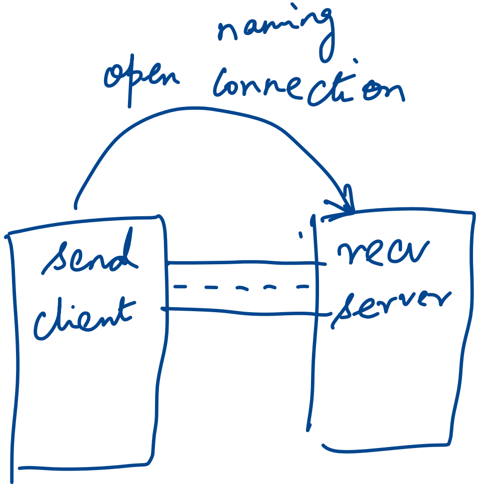
Reliable messages: TCP

Remote procedure call: RPC

Networking

640

RAW MESSAGES: UDP



UDP : User Datagram Protocol

Abstraction (analogous to file)

API:

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

Provide minimal reliability features:

partial failures

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

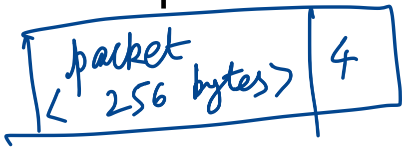
write 1
write 2

read 2
read 1
read 1

duplicate

Checksum

Journalling



RAW MESSAGES: UDP

Advantages

- Lightweight → Protocol doesnt add overhead
- Some applications make better reliability decisions themselves (e.g., video conferencing programs)

Disadvantages

- More difficult to write applications correctly

video and video conferencing

→ low latency

→ skip frames
↳ closer to live action

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

QUIZ 22

1. Why is cache affinity important?

→ expensive → option 3

2. Order the following processes by priority, from highest to lowest.

A: +5
B: -3
C: -10
D: +19

C, B, A, D as highest to lowest

3. What is an advantage of single queue (SQMS) scheduling?

↳ simpler that MQMS

4. What is an advantage of multiple queue (MQMS) scheduling?

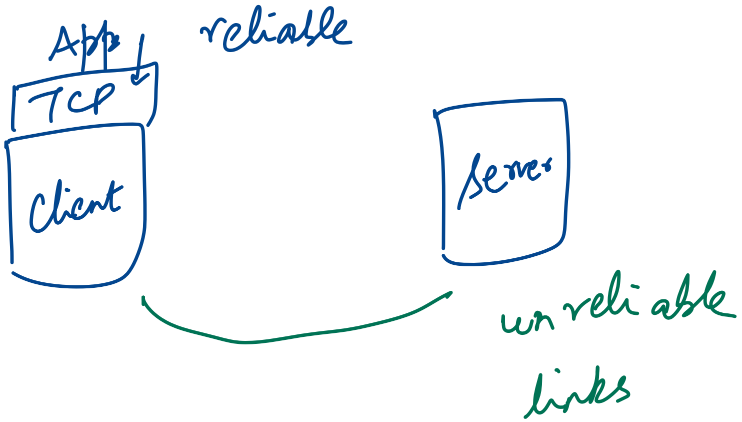
↳ scales better as we add more processors.

RELIABLE MESSAGES: LAYERING STRATEGY

TCP: Transmission Control Protocol

Using software to build

reliable logical connections over unreliable physical connections

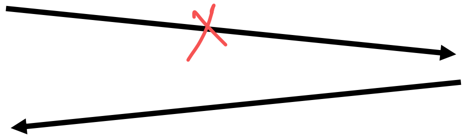


TECHNIQUE #1: ACK

Sender

[send message]

[recv ack]



Receiver

[recv message]

[send ack]

client knows server got this message

if server didn't get the message, I will retry sending it

Ack: Sender knows message was received

What to do about message loss?

TECHNIQUE #2: TIMEOUT

Sender

[send message]

[start timer]

timeout value

… waiting for ack …

[timer goes off]

[send message]

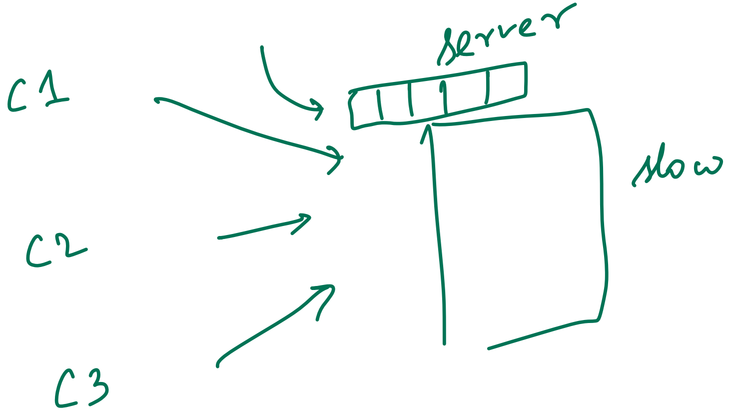
[recv ack]

Receiver

[recv message]

[send ack]

TIMEOUT



How long to wait?

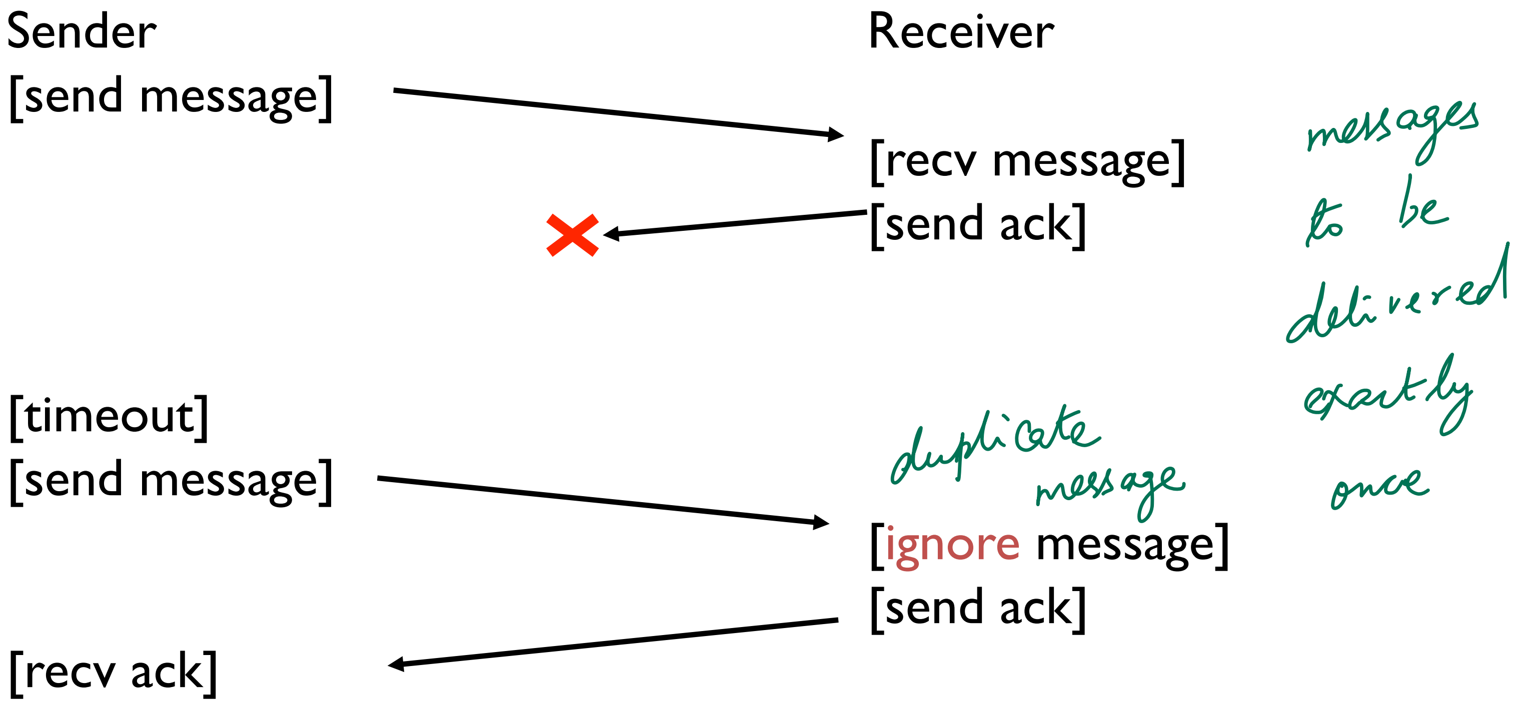
Too long?

- 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

Duplication and Ordering

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 ?
  - ↳ buffer this message

client
send 1
send 2
send 3

server
recv 1
× recv 1 ignore
recv 3
× missing message 2
→ buffer 3

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

send

recv

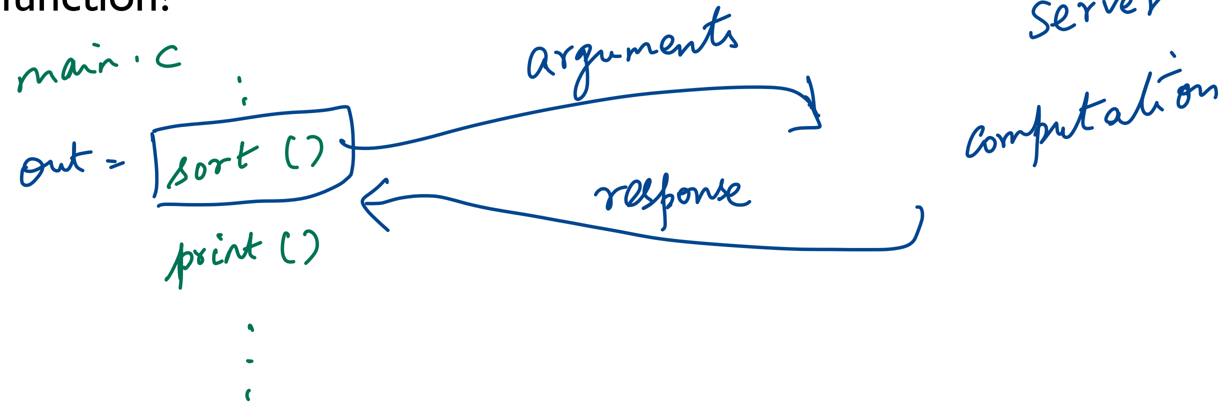
API

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

client — Machine A

```
int main(...) {
        int x = foo("hello");
}

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

server — Machine B

```
int foo(char *msg) {
        ...
}

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

wrapper functions

same type signature

impl of foo

listener parses call foo sends return value

serialize arguments to foo & send a message "run foo on hello"

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) {
    ...
}
```

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

server wrapper

to remote function

Pointers? → what it points to? "Smart" ptrs

part of RPC library

How do you serialize function arguments? → bytes stream → deserialize → "hello"

RPC TOOLS

· Protocol Buffers

RPC packages help with two components

(1) Runtime library

- Thread pool → each RPC on a diff thread etc.
- Socket listeners call functions on server

(2) Stub generation

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

int foo (string)
↓
client wrapper automatically

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

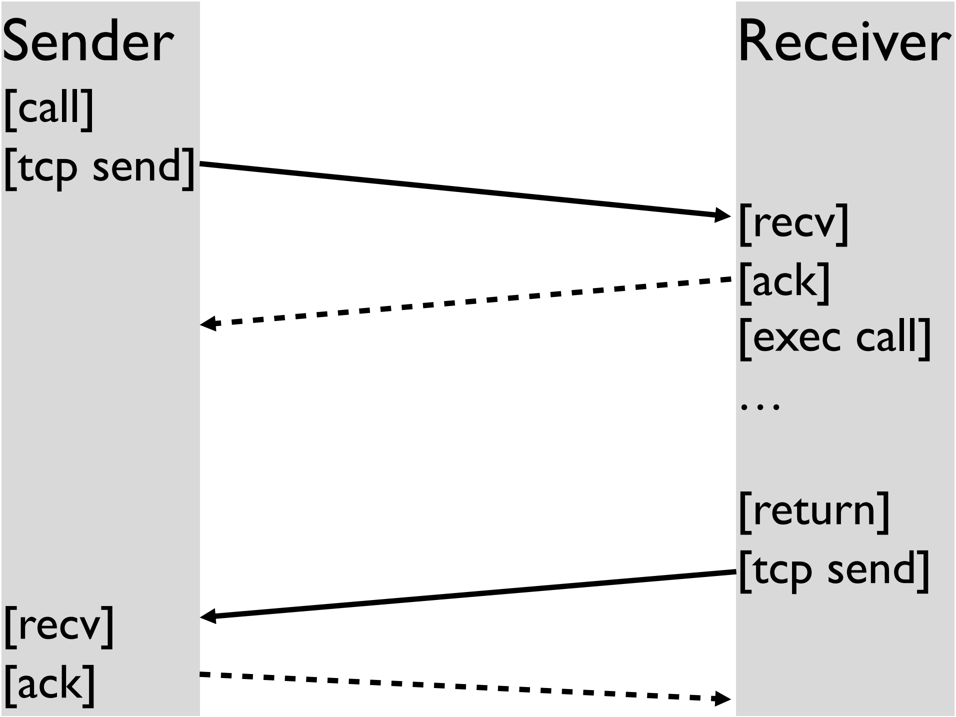
WRAPPER GENERATION: POINTERS

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]

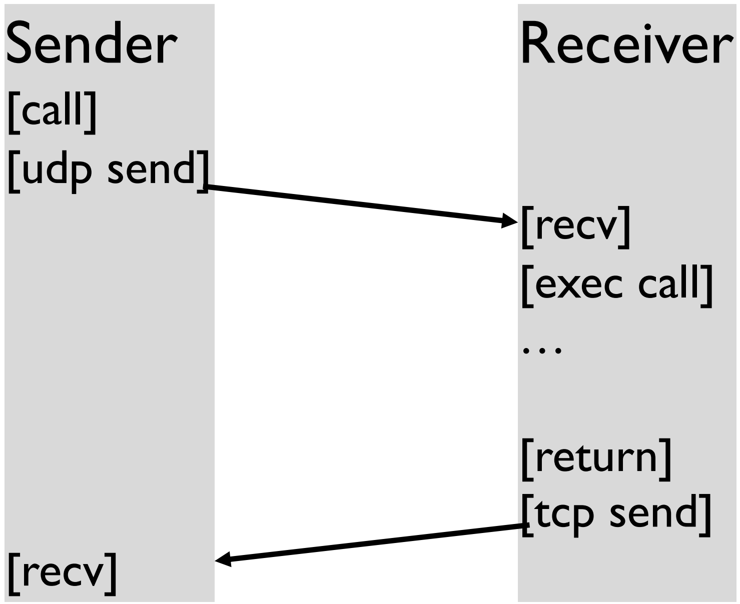
RPC OVER UDP

Strategy: use function return as implicit ACK

Piggybacking technique

What if function takes a long time?

then send a separate ACK



NEXT STEPS

Review for Midterm 3

Last lecture!