DISTRIBUTED SYSTEMS

Shivaram Venkataraman

CS 537, Spring 2019

ADMINISTRIVIA

Project 5: Due April 29. Last Project!

Final Exam: Everything before the last lecture

till Tue 4/30

Discussion today: Worksheet on topics after midterm

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

AGENDA / LEARNING OUTCOMES

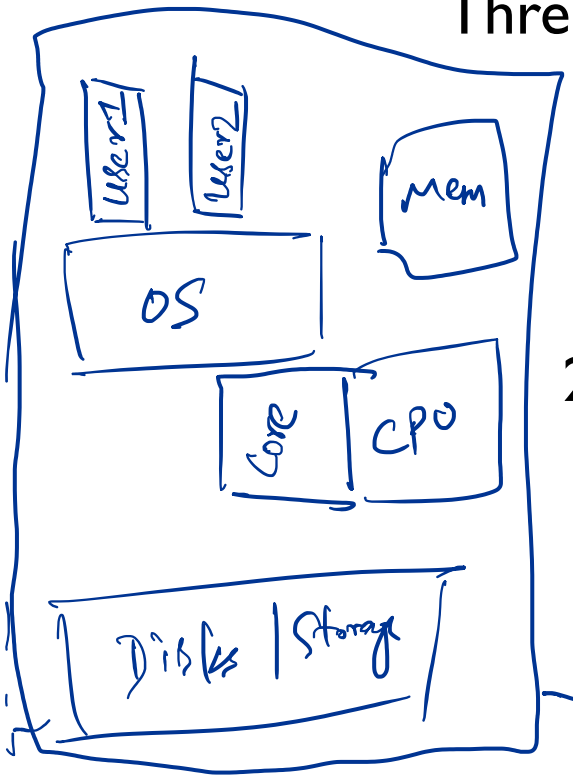
What are the design principles for systems that operate across machines?

How to handle partial failures?

RECAP

OPERATING SYSTEMS: THREE EASY PIECES

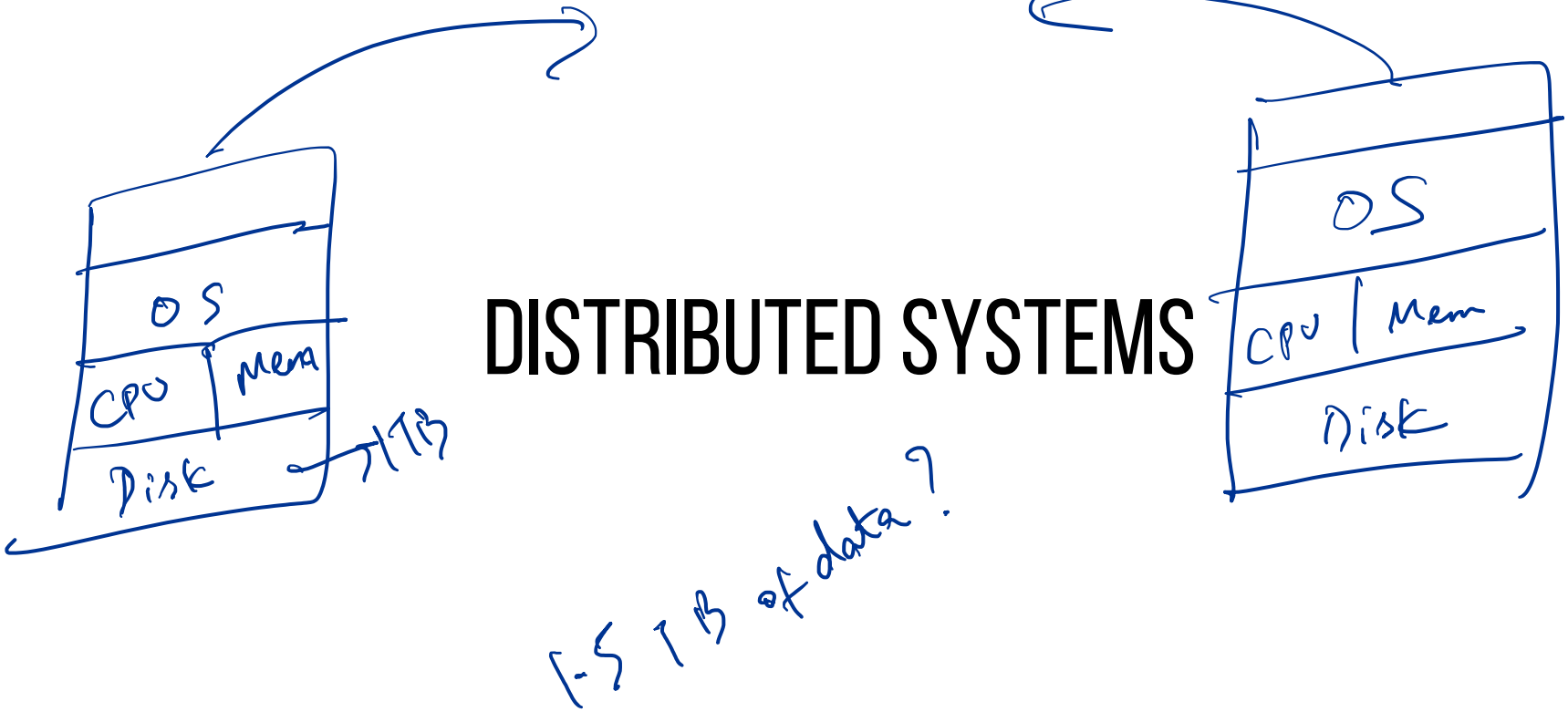
Three conceptual pieces



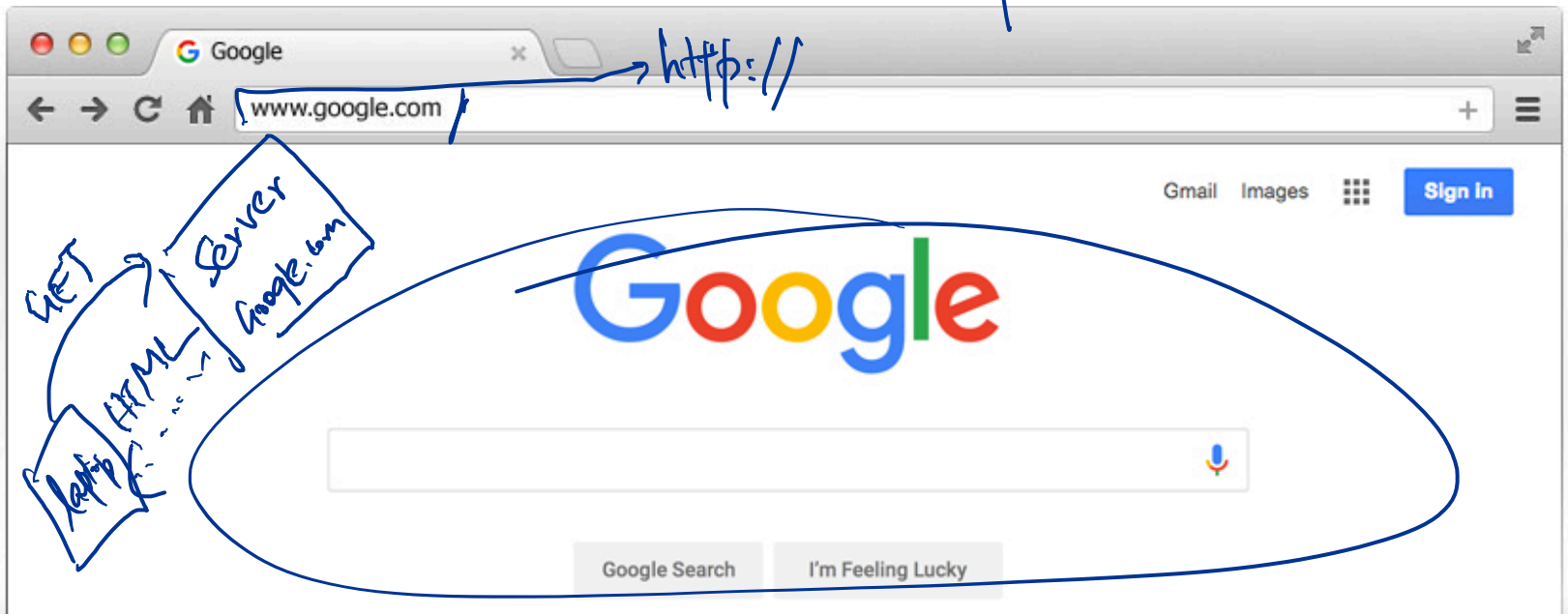
1. Virtualization → CPU, Mem
   - Processes → Scheduling
   - Addr. spaces → Page table
2. Concurrency → Locks, CV, semaphore
3. Persistence → Data on hard disks
   - Impl → FFS, LFS
   - FS → Crash consistency

Server

DISTRIBUTED SYSTEMS



HOW DOES GOOGLE SEARCH WORK?



WHAT IS A DISTRIBUTED SYSTEM?

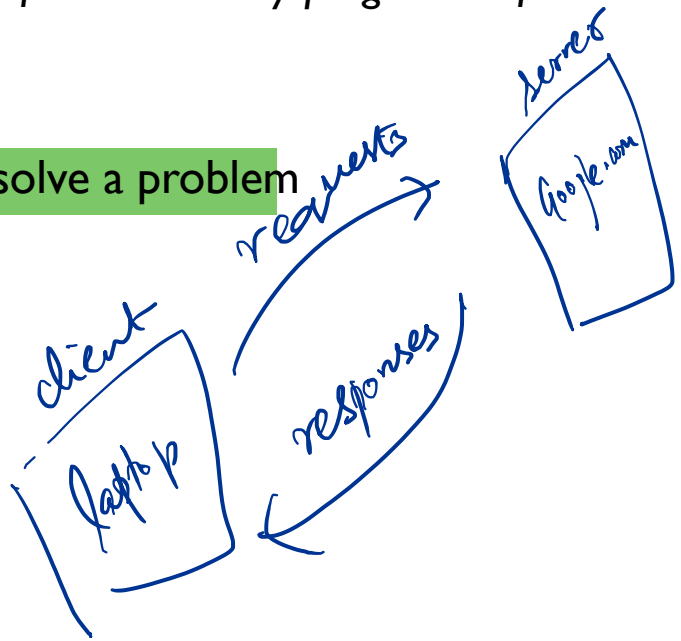
A distributed system is one where a machine I've never heard of can cause my program to fail.

— *Leslie Lamport* → 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



WHY GO DISTRIBUTED?

- More computing power
- More storage capacity
- Fault tolerance
- Data sharing ≡ P2P file transfers | downloads

1. MC failure
2. Link failure
3. Router failure
4. Packet errors / drops

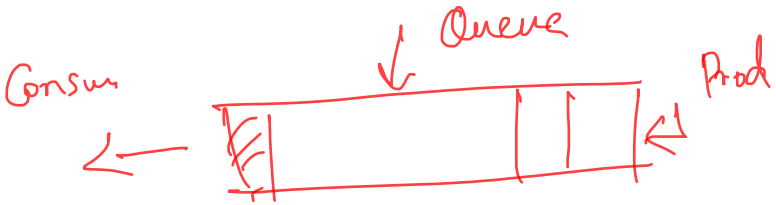
NEW CHALLENGES

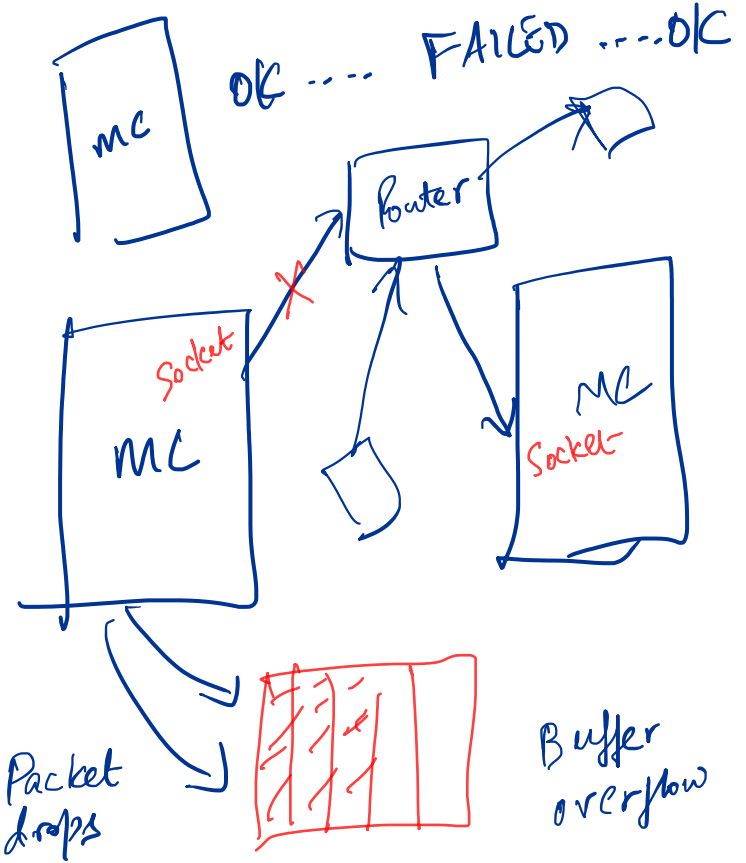
System failure: need to worry about partial failure

Communication failure: links unreliable

- bit errors
- packet loss
- node/link failure

Why are network sockets less reliable than pipes?





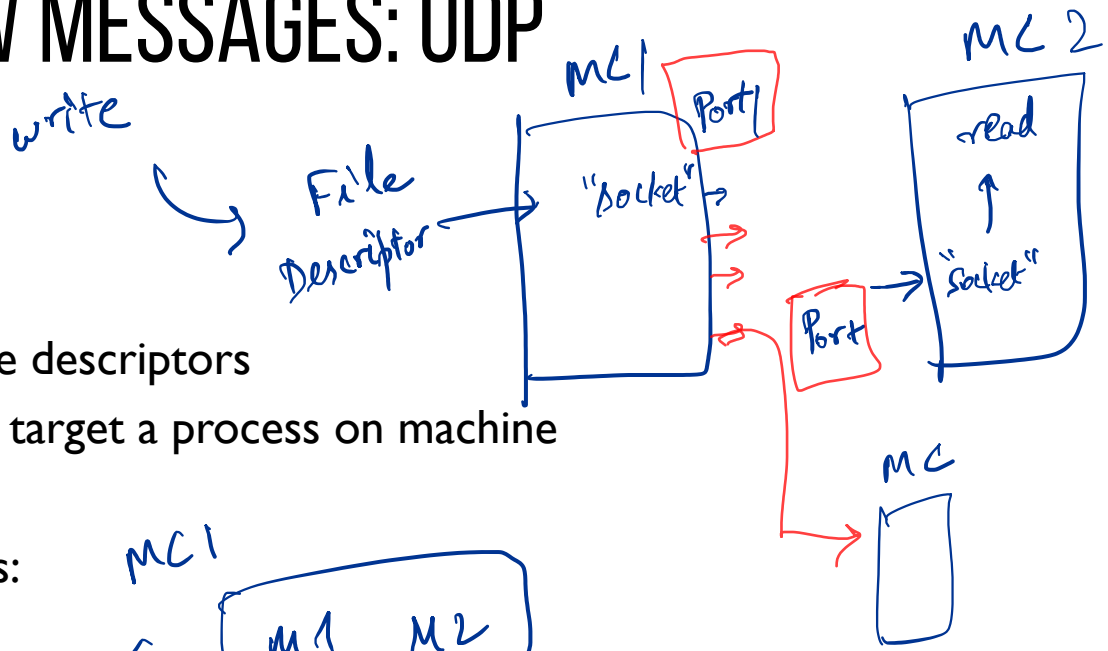
COMMUNICATION OVERVIEW

Raw messages: UDP

Reliable messages: TCP

Remote procedure call: RPC

RAW MESSAGES: UDP



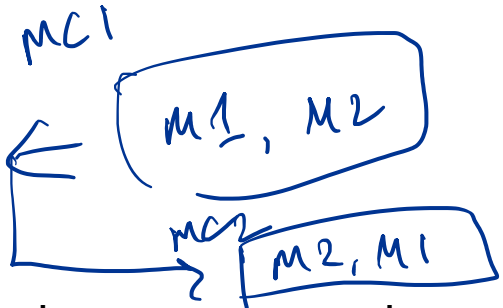
UDP : User Datagram Protocol

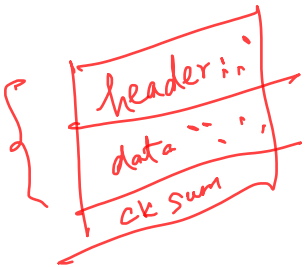
API:

- 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

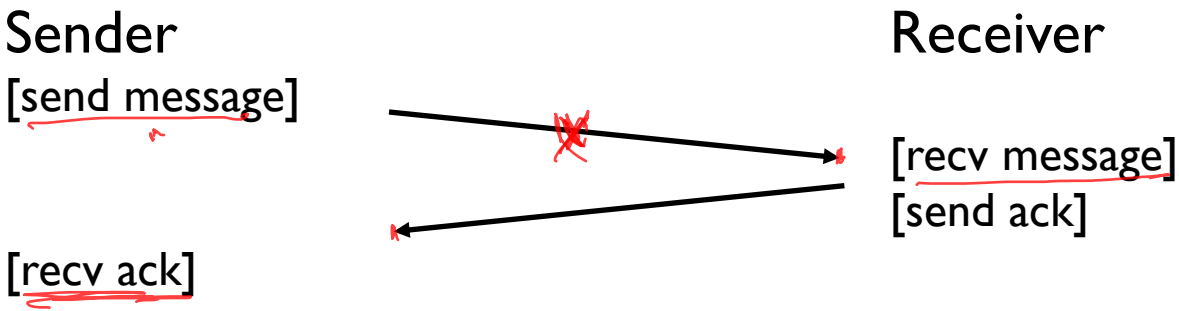
TCP: Transmission Control Protocol

Using software to build

reliable logical connections over unreliable physical connections

TECHNIQUE #1: ACK

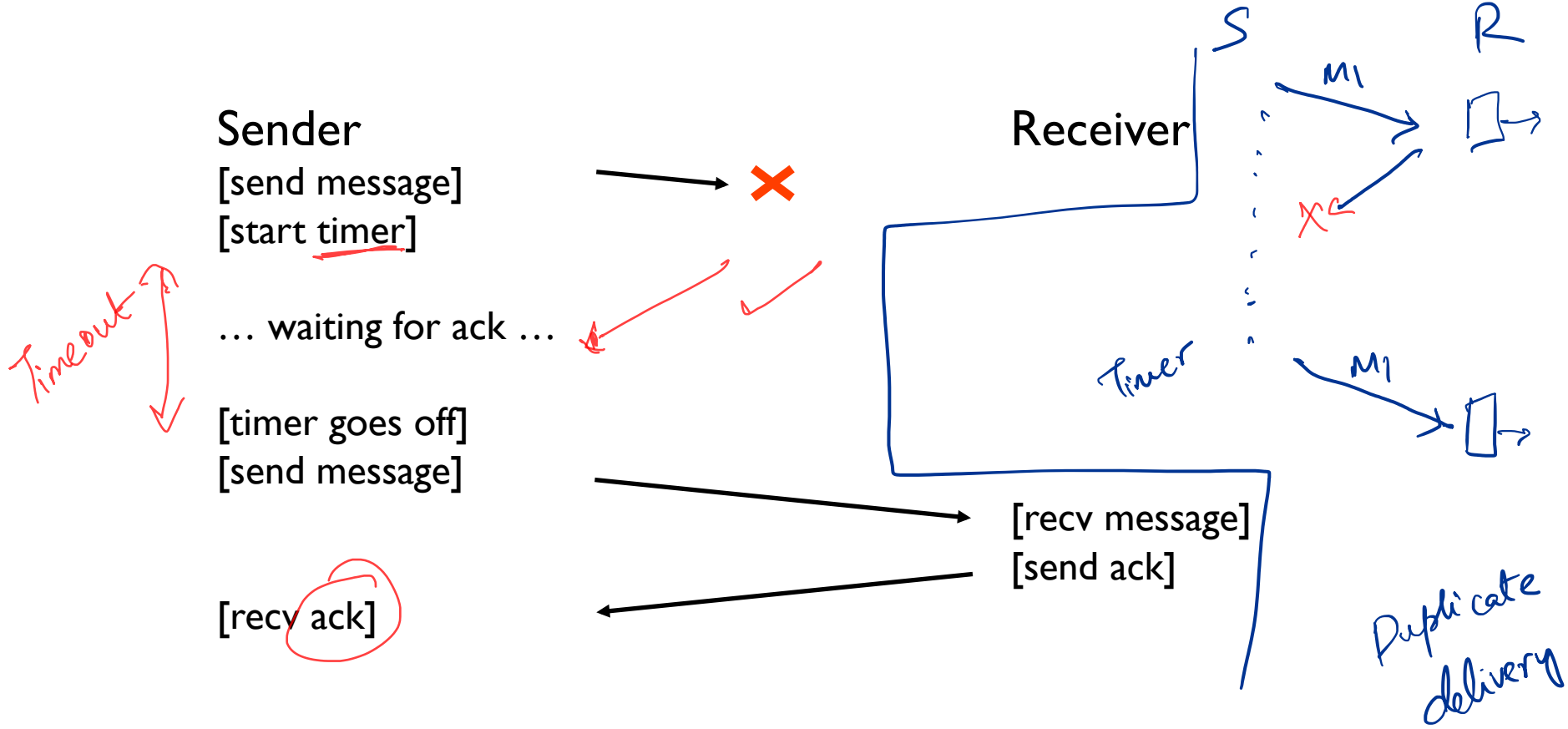
Acknowledgement



Ack: Sender knows message was received

What to do about message loss?

TECHNIQUE #2: TIMEOUT



TIMEOUT

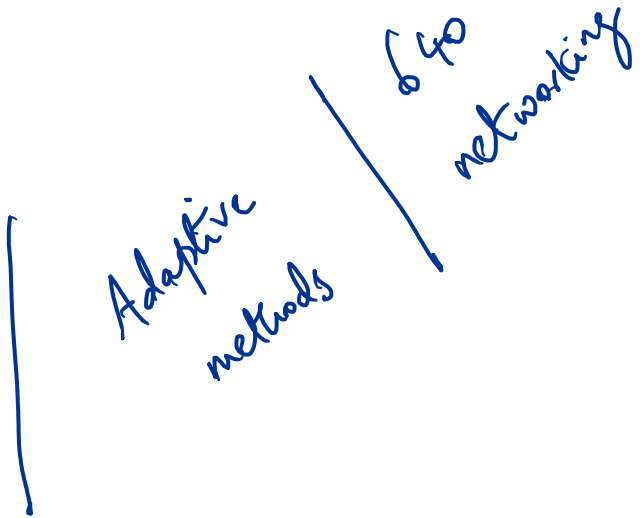
How long to wait?

Too long?

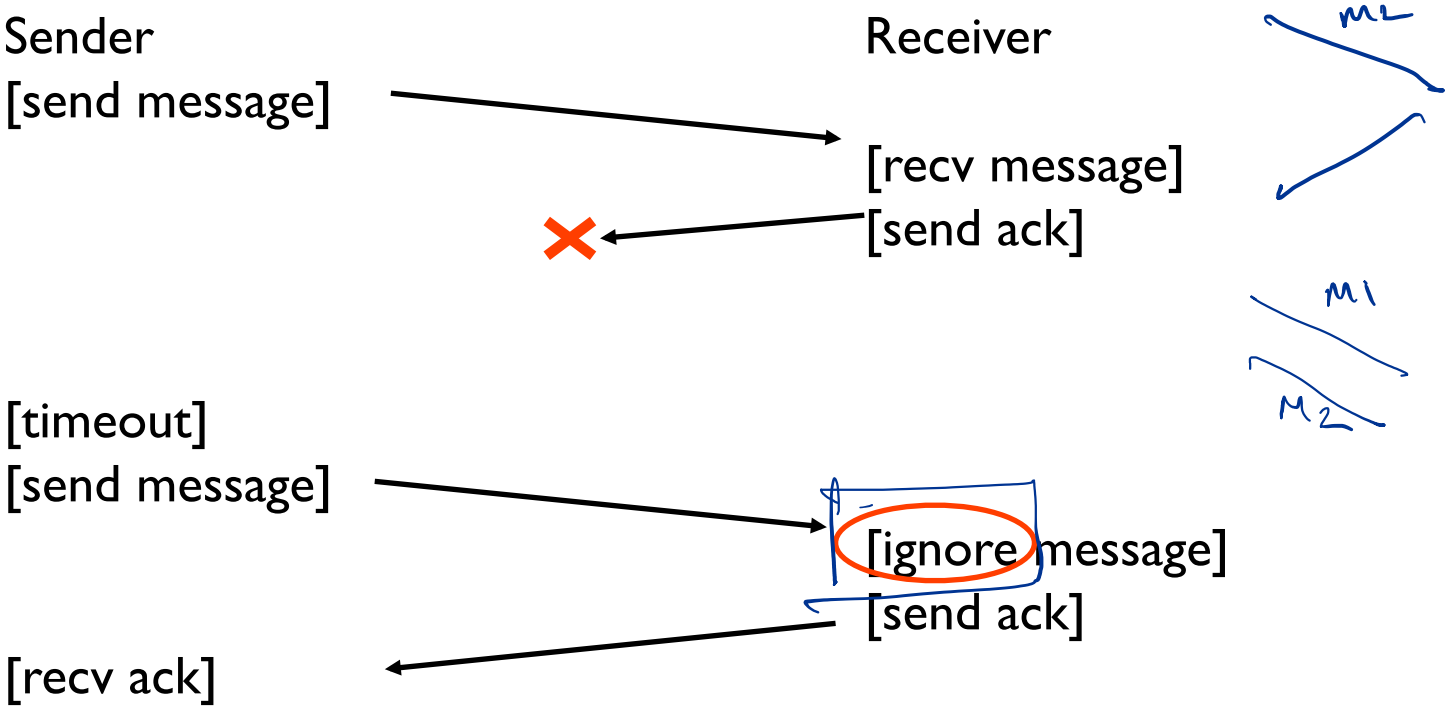
- System feels unresponsive

Too short?

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



LOST ACK PROBLEM



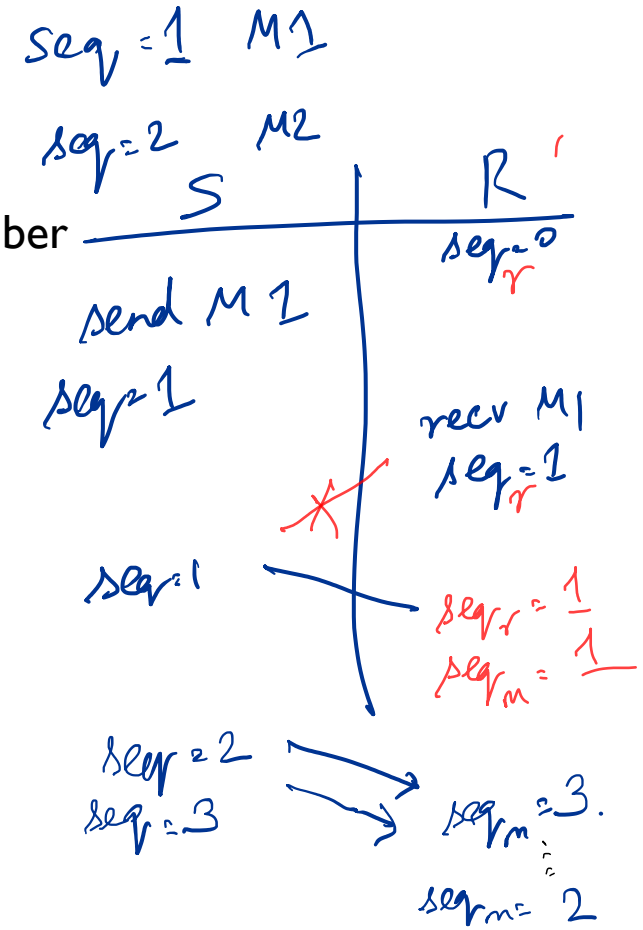
Reliability : exactly once

Ordering : sender, recv see msgs in same order

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 ?

Buffer these messages until K = N+1

TCP

TCP: Transmission Control Protocol

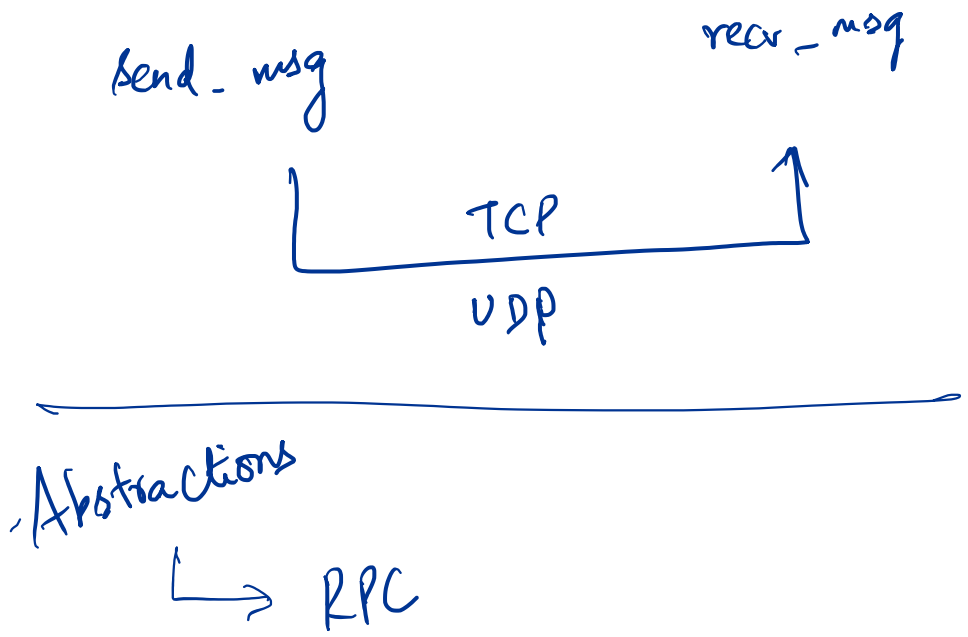
Most popular protocol based on seq nums
Buffers messages so arrive in order
Timeouts are adaptive

COMMUNICATIONS OVERVIEW

Raw messages: UDP

Reliable messages: TCP

Remote procedure call: RPC



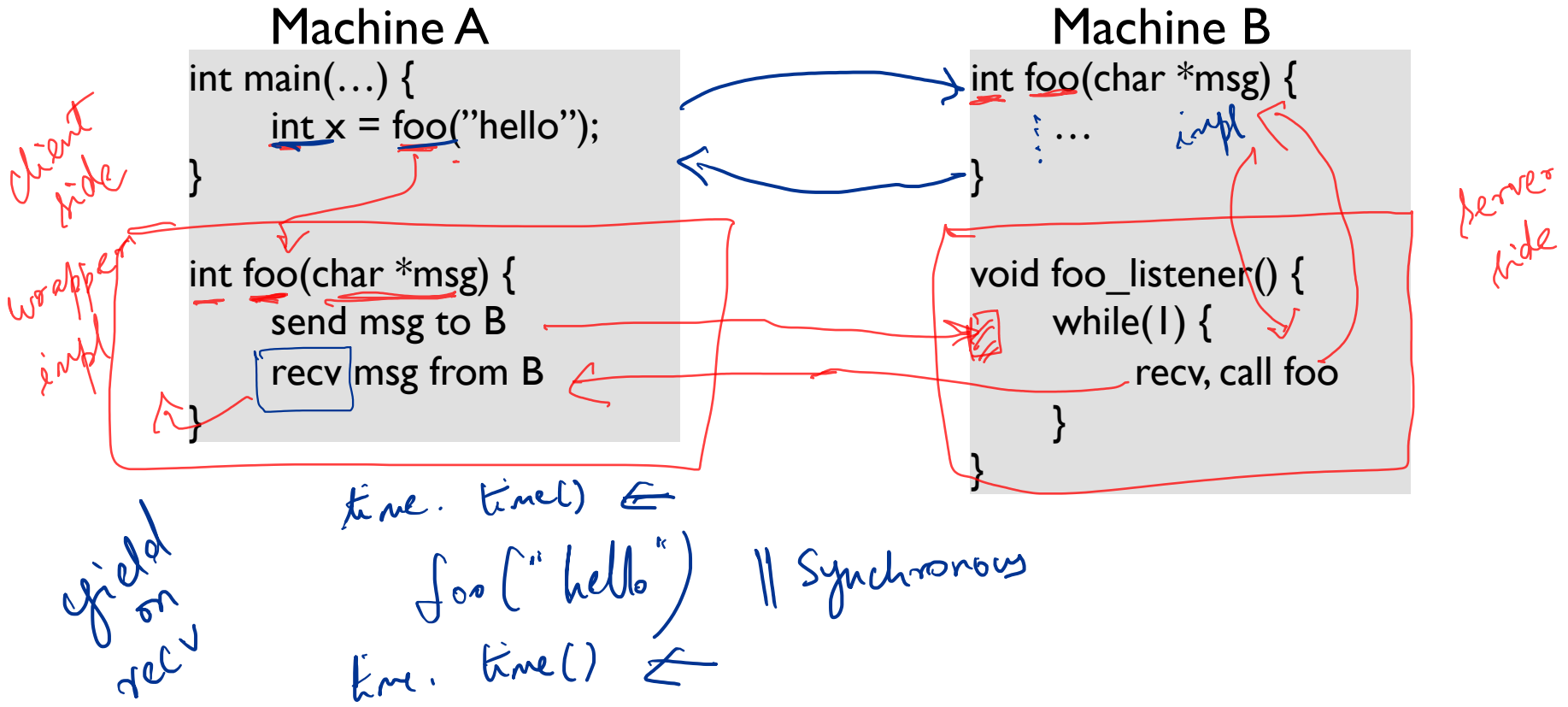
RPC

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
}
}
client side
wrapper impl
server side
impl
yield on recv
time. time()
foo("hello")
Synchronous
time. time()

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
        }
}
```

Thread pool [5]

collection of threads

RPC TOOLS

RPC packages help with two components

(1) Runtime library

- Thread pool , load balancing queuing
- Socket listeners call functions on server

(2) Stub generation 1980s

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

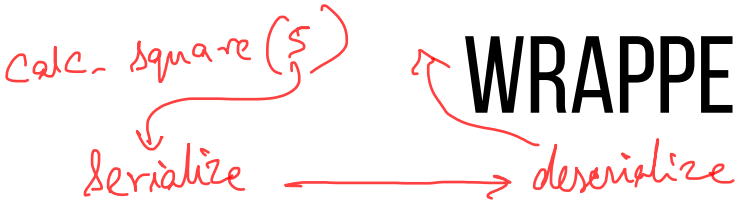
foobar.proto

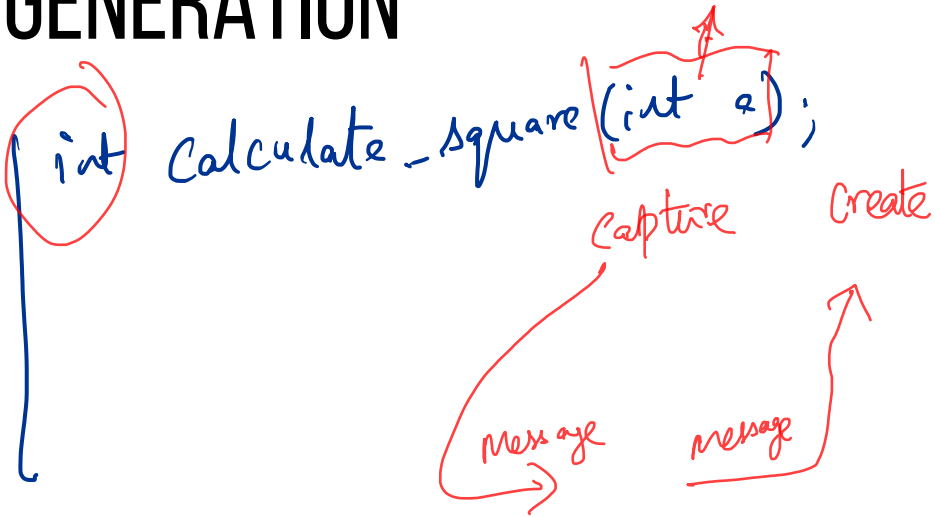
```
service FooBar {
    int foo(string var);
    boolean bar(int var1);
}
```

protogen → client, foobar.h

foobar.proto → server → foobar_server.h

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

```
int calc_square (int * a) {

}
```

(struct linked_list* head)

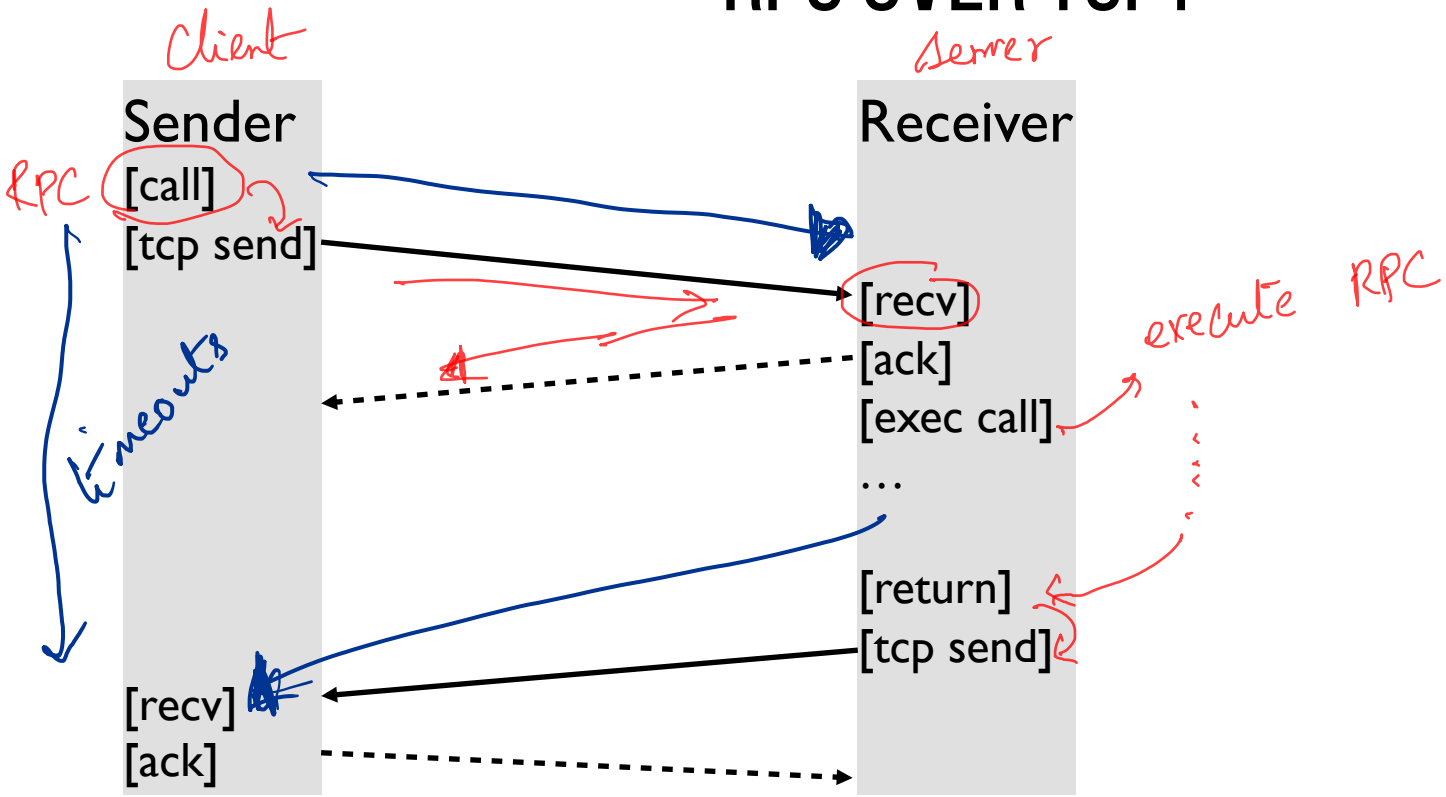
WRAPPER GENERATION: POINTERS

Why are pointers problematic?

Address passed from client not valid on server

Solutions? Smart RPC package: follow pointers and copy data

RPC OVER TCP?


Client
Server
Sender
Receiver
RPC
[call]
[tcp send]
[recv]
[ack]
[exec call]
...
execute RPC
timeouts
[return]
[tcp send]
[recv]
[ack]

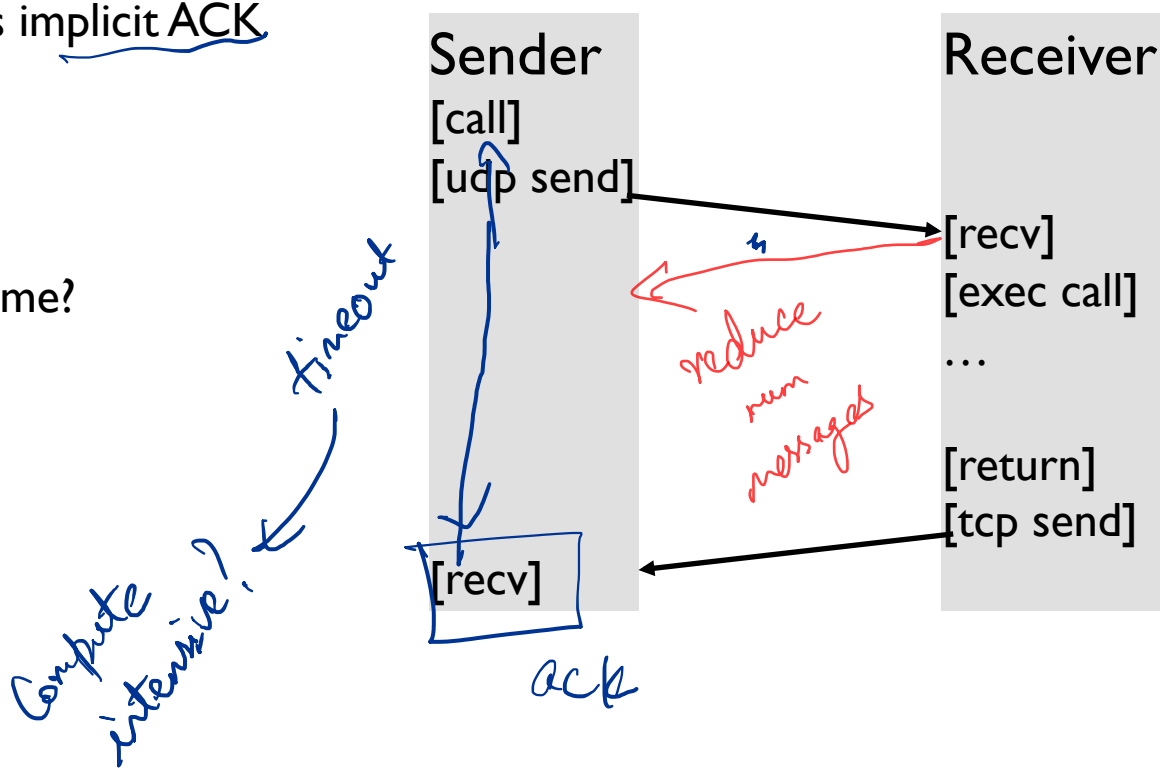
RPC OVER UDP

Strategy: use function return as implicit ACK

Piggybacking technique

What if function takes a long time?

then send a separate ACK



BREAK! NO QUIZ!

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

DISTRIBUTED SYSTEMS IN PRACTICE

DISTRIBUTED SYSTEMS

Classic systems, algorithms

Email

Grapevine: An exercise in distributed computing
Andrew Birrell Roy Levin Roger M. Needham Mike Schroeder
Communications of the ACM | April 1982, Vol 25

CS 739

Time, Clocks, and the Ordering of Events in a Distributed System
L. Lamport
Communications of the ACM
Vol. 21, No. 7 (July 1978)

GOOGLE 1997

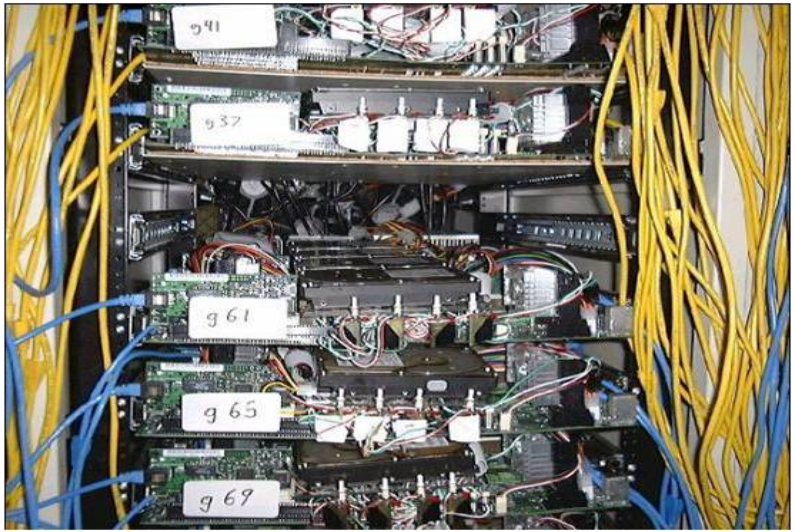
DATA, DATA, DATA

"…Storage space must be used efficiently to store indices and, optionally, the documents themselves. The indexing system must process hundreds of gigabytes of data efficiently…"

The Anatomy of a Large-Scale Hypertextual Web Search Engine

Sergey Brin and Lawrence Page

GOOGLE 2001



Commodity CPUs

Lots of disks

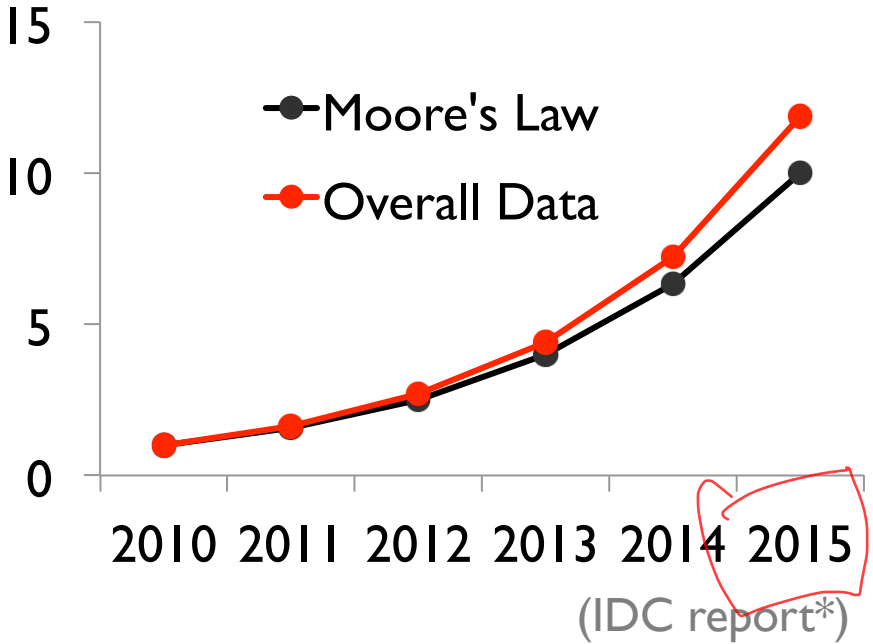
Low bandwidth network

Cheap !

DATACENTER EVOLUTION

Facebook's daily logs: 60 TB

Google web index: 10+ PB



(IDC report*)

CS 744 Big Data Systems

DATACENTER EVOLUTION

Google data centers in The Dalles, Oregon

DATACENTER EVOLUTION

Capacity:

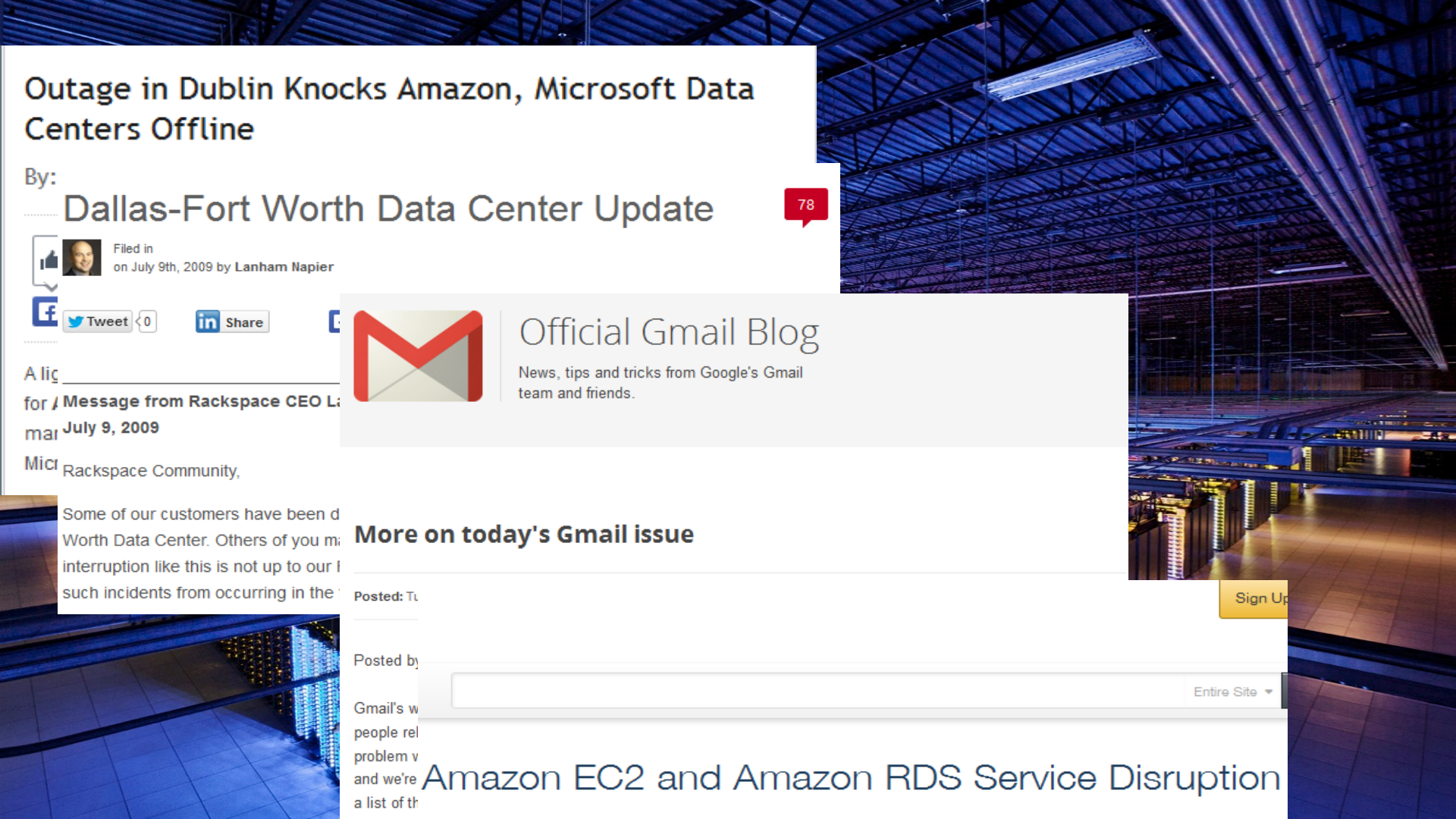
~10000 machines

Bandwidth:

12-24 disks per node

Latency:

256GB RAM cache



Outage in Dublin Knocks Amazon, Microsoft Data Centers Offline

By:

Dallas-Fort Worth Data Center Update

78

Filed in
on July 9th, 2009 by **Lanham Napier**

Tweet 0 Share

A lig
for A
mar
Micr

Message from Rackspace CEO La
July 9, 2009

Rackspace Community,

Some of our customers have been d
Worth Data Center. Others of you ma
interruption like this is not up to our h
such incidents from occurring in the

Official Gmail Blog

News, tips and tricks from Google's Gmail team and friends.

More on today's Gmail issue

Posted: Tu

Sign Up

Posted by

Gmail's w
people rel
problem v
and we're
a list of th

Entire Site

Amazon EC2 and Amazon RDS Service Disruption

The Joys of Real Hardware

Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures
slow disks, bad memory, misconfigured machines, flaky machines, etc.

Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

JEFF DEAN @ GOOGLE

MAPREDUCE

PROGRAMMING MODEL

Data type: Each record is (key, value)

Map function:

$$(K_{in}, V_{in}) \rightarrow \text{list}(K_{inter}, V_{inter})$$

Reduce function:

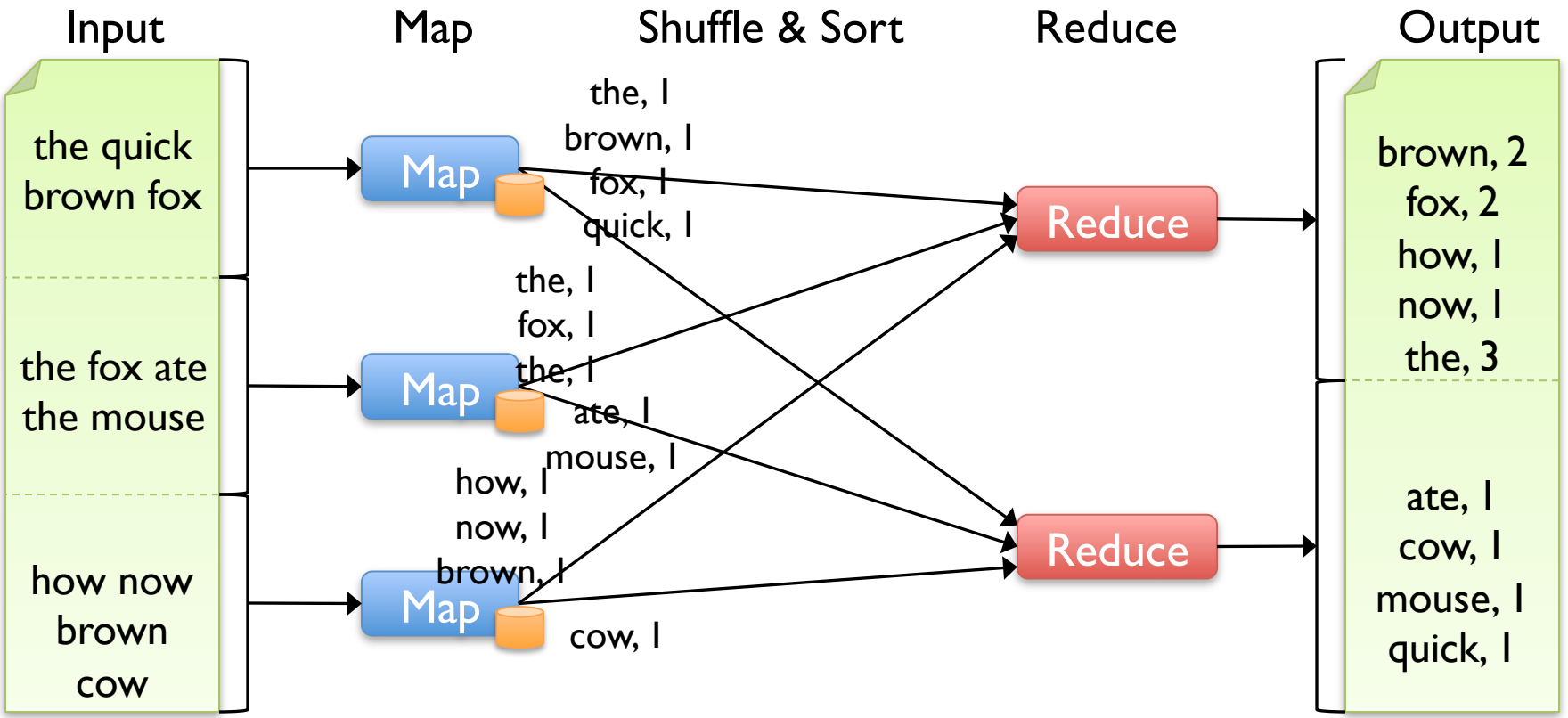
$$(K_{inter}, \text{list}(V_{inter})) \rightarrow \text{list}(K_{out}, V_{out})$$

EXAMPLE: WORD COUNT

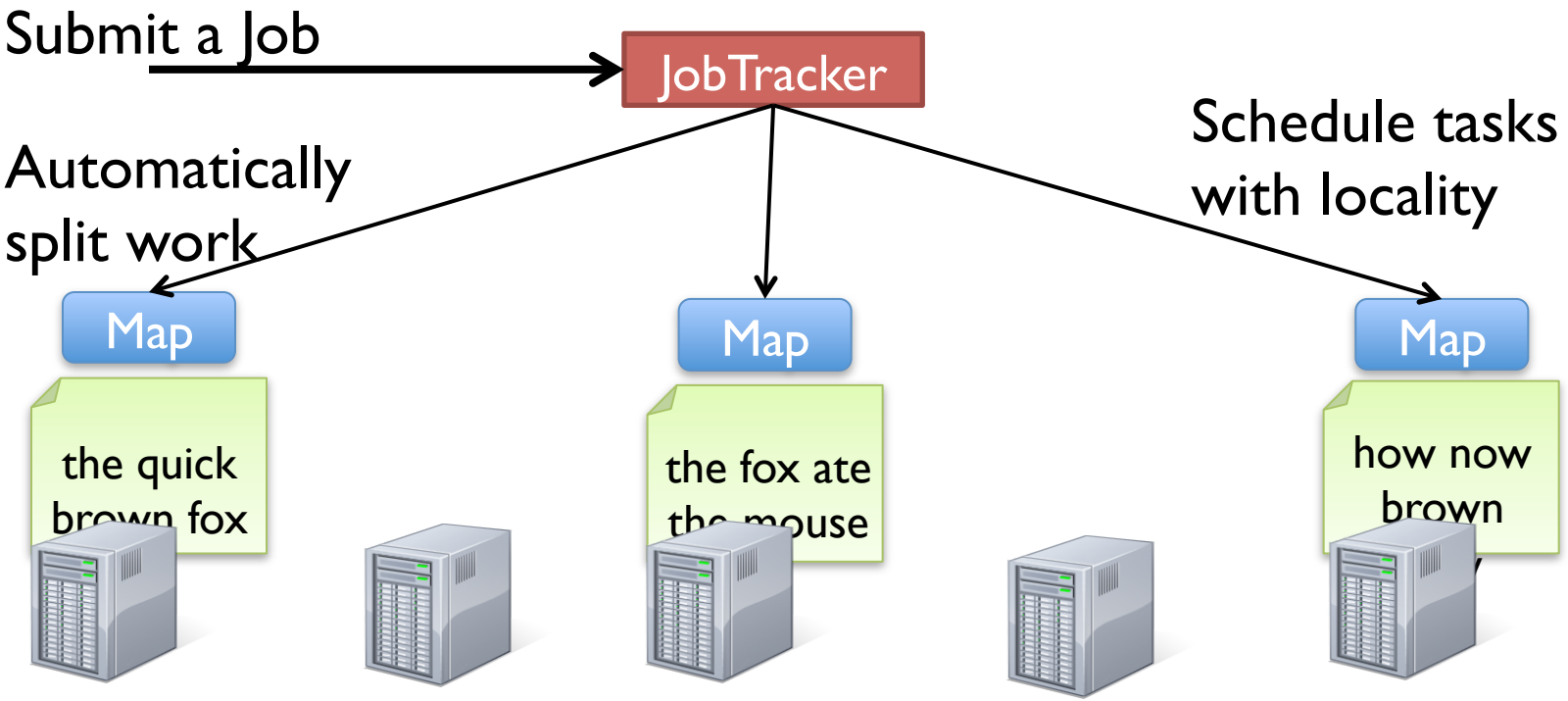
```
def mapper(line):
    for word in line.split():
        output(word, 1)


def reducer(key, values):
    output(key, sum(values))
```

WORD COUNT EXECUTION



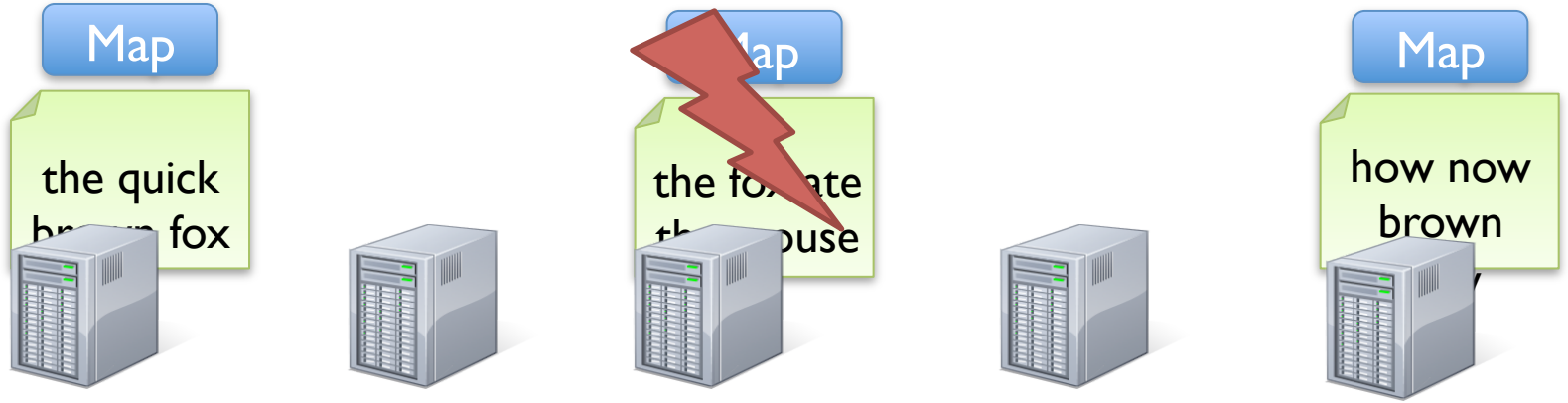
WORD COUNT EXECUTION



FAULT RECOVERY

If a task crashes:

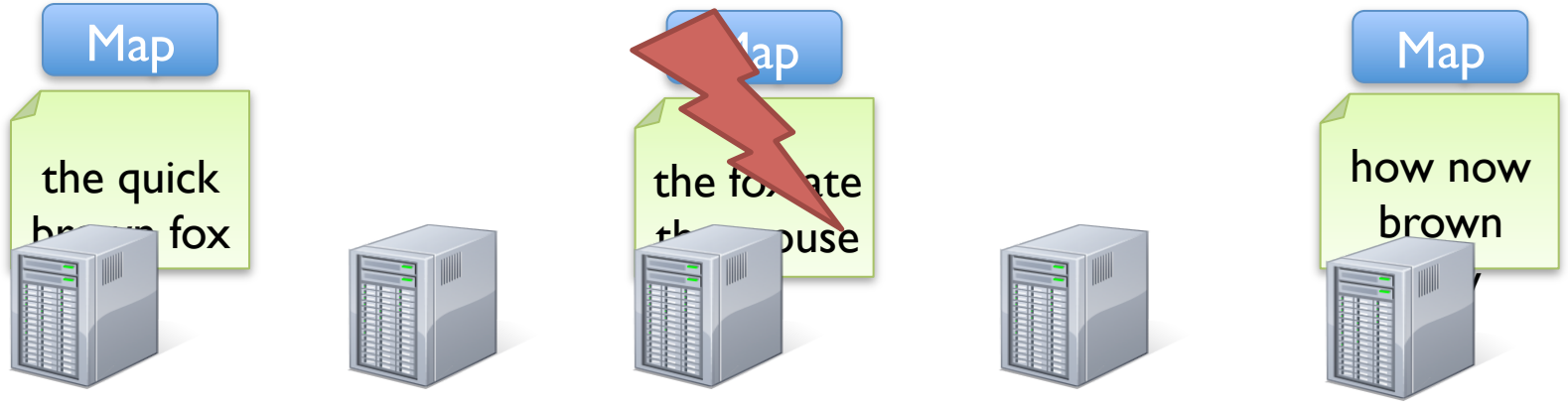
- Retry on another node
- If the same task repeatedly fails, end the job



FAULT RECOVERY

If a task crashes:

- Retry on another node
- If the same task repeatedly fails, end the job



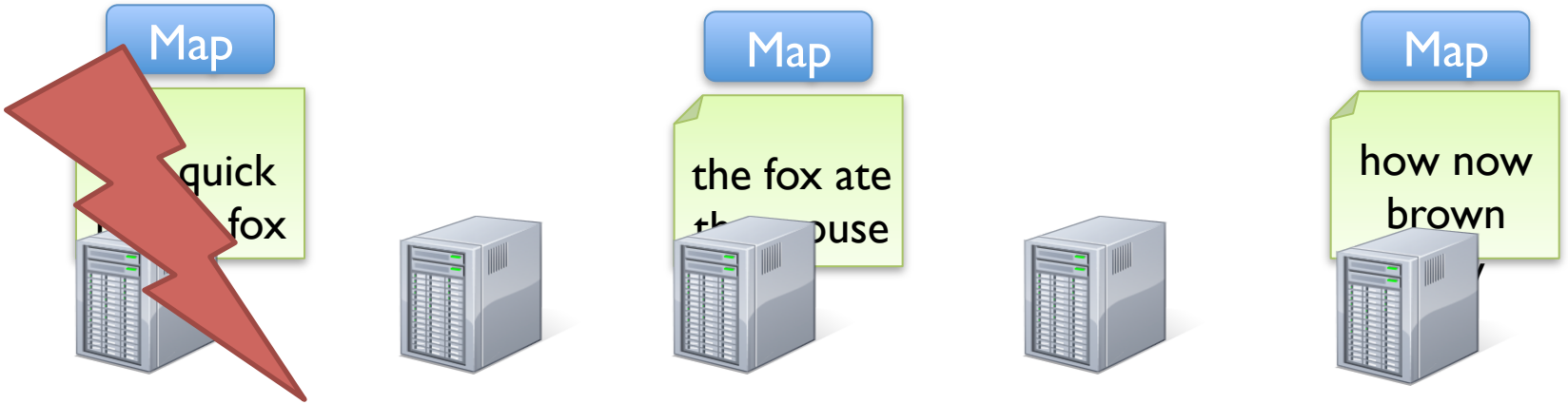
Requires user code to be **deterministic**

FAULT RECOVERY

If a node crashes:

- Relaunch its current tasks on other nodes

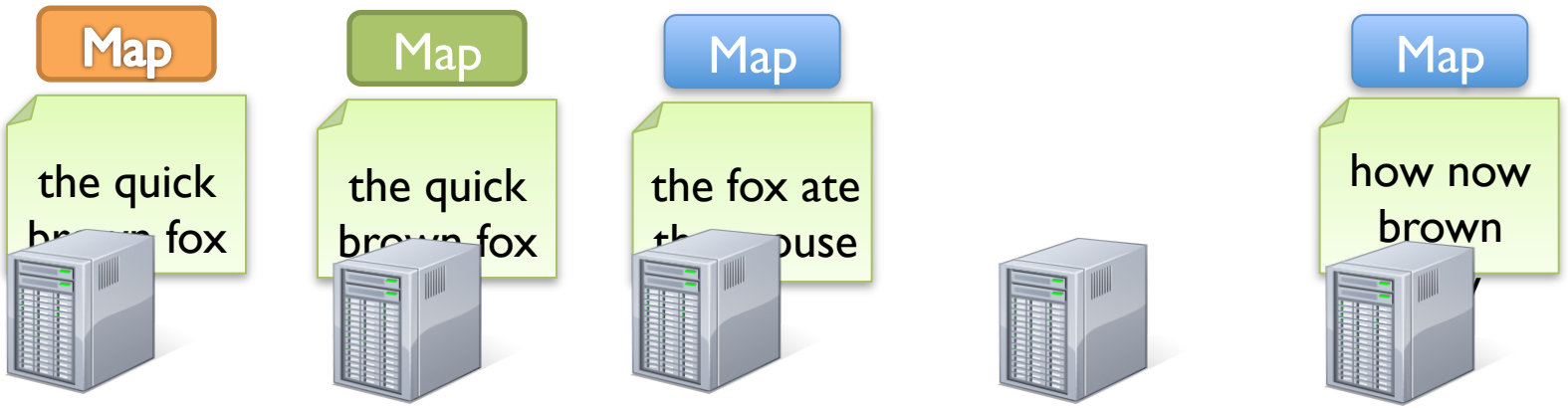
What about task inputs ? File system replication



FAULT RECOVERY

If a task is going slowly (straggler):

- Launch second copy of task on another node
- Take the output of whichever finishes first



NEXT STEPS

Next class: Distributed Filesystem(NFS)

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