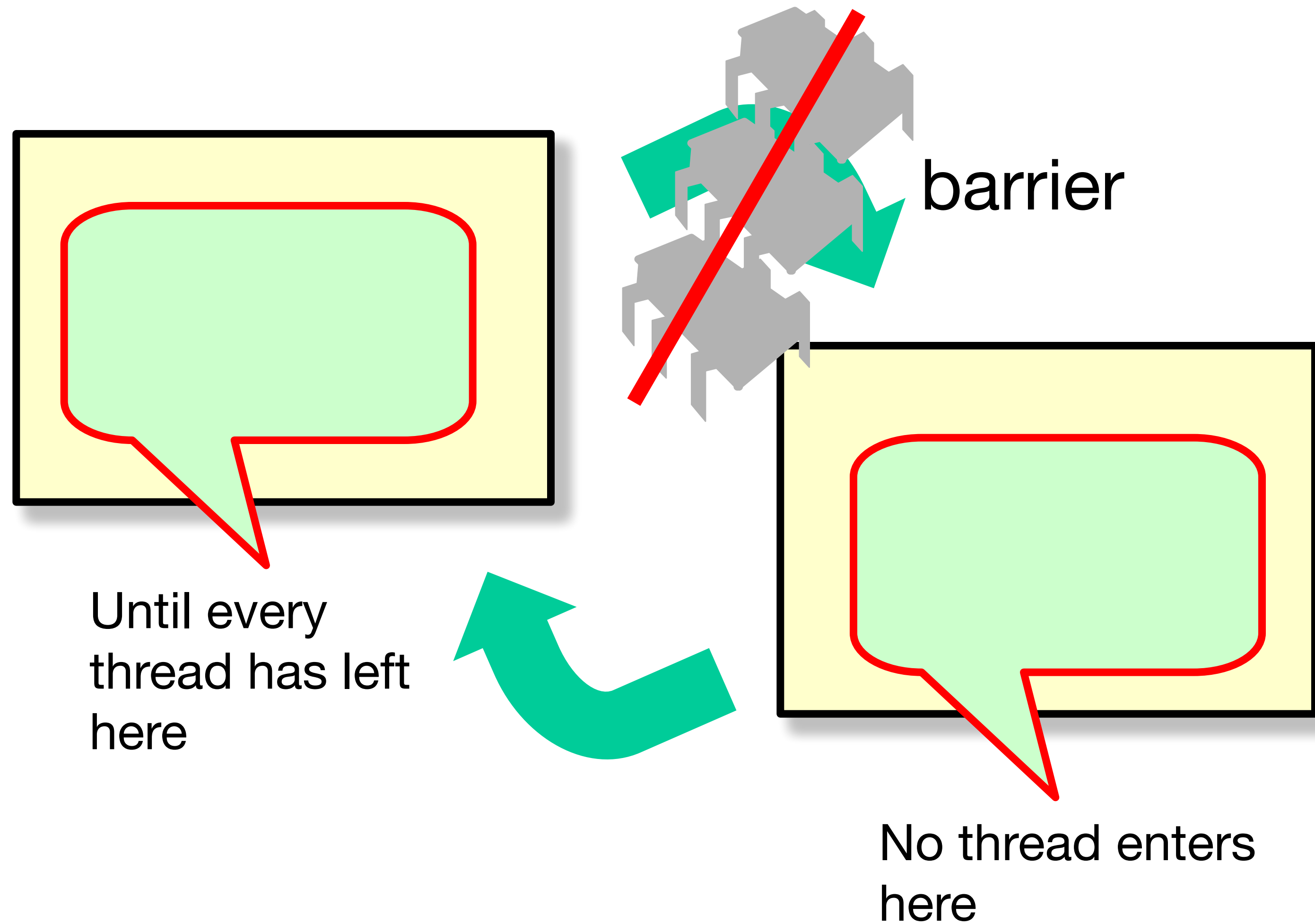


# Networks

CS 475, Spring 2019  
Concurrent & Distributed Systems

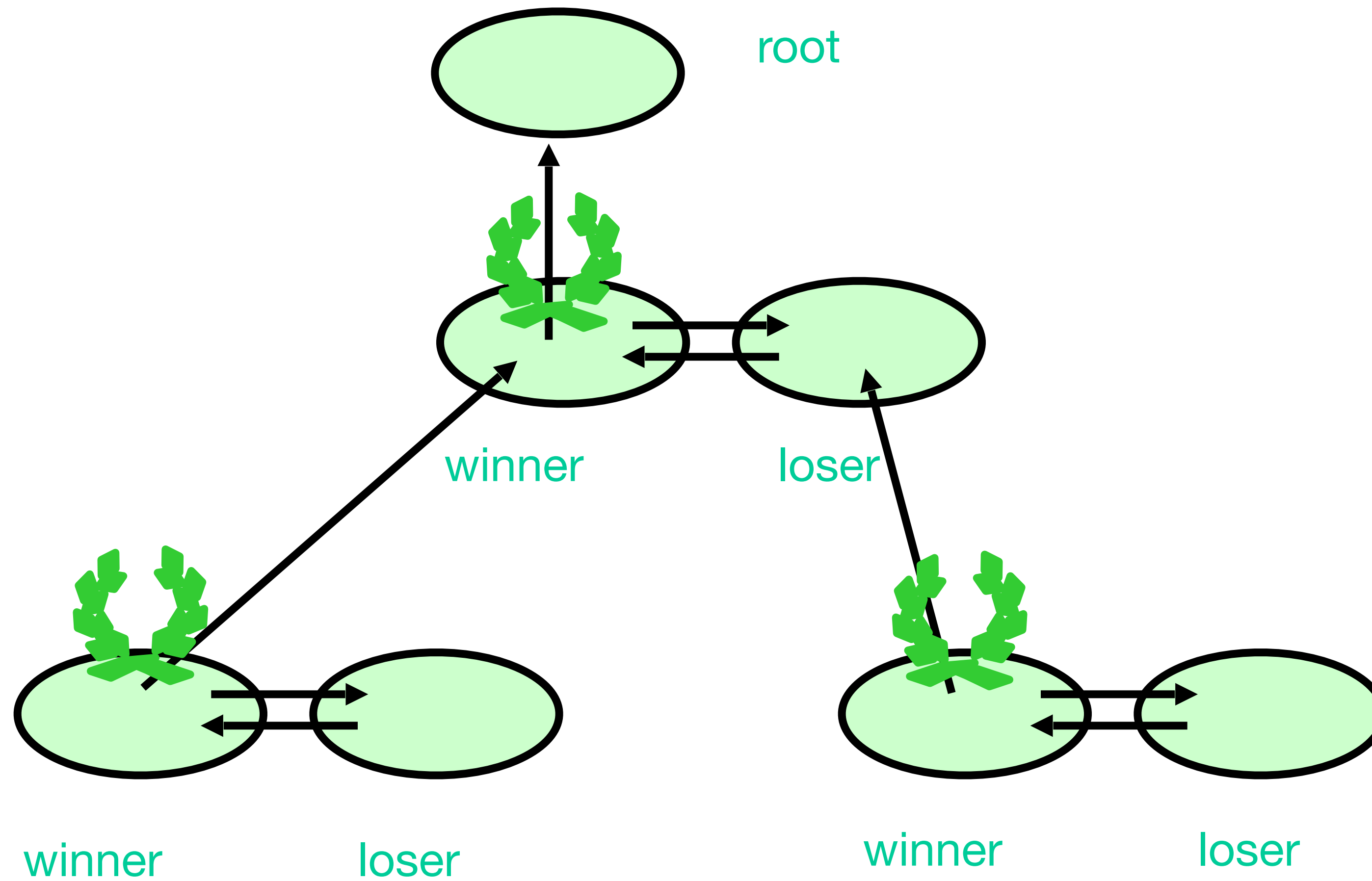
# Barrier Synchronization



# Duality

- Dual to mutual exclusion
  - Include others, not exclude them
- Same implementation issues
  - Interaction with caches ...
    - Invalidation?
    - Local spinning?

# Tournament Tree Barriers



# So...How will we make use of multicores?

Back to Amdahl's Law:

$$\text{Speedup} = 1/(\text{ParallelPart}/N + \text{SequentialPart})$$

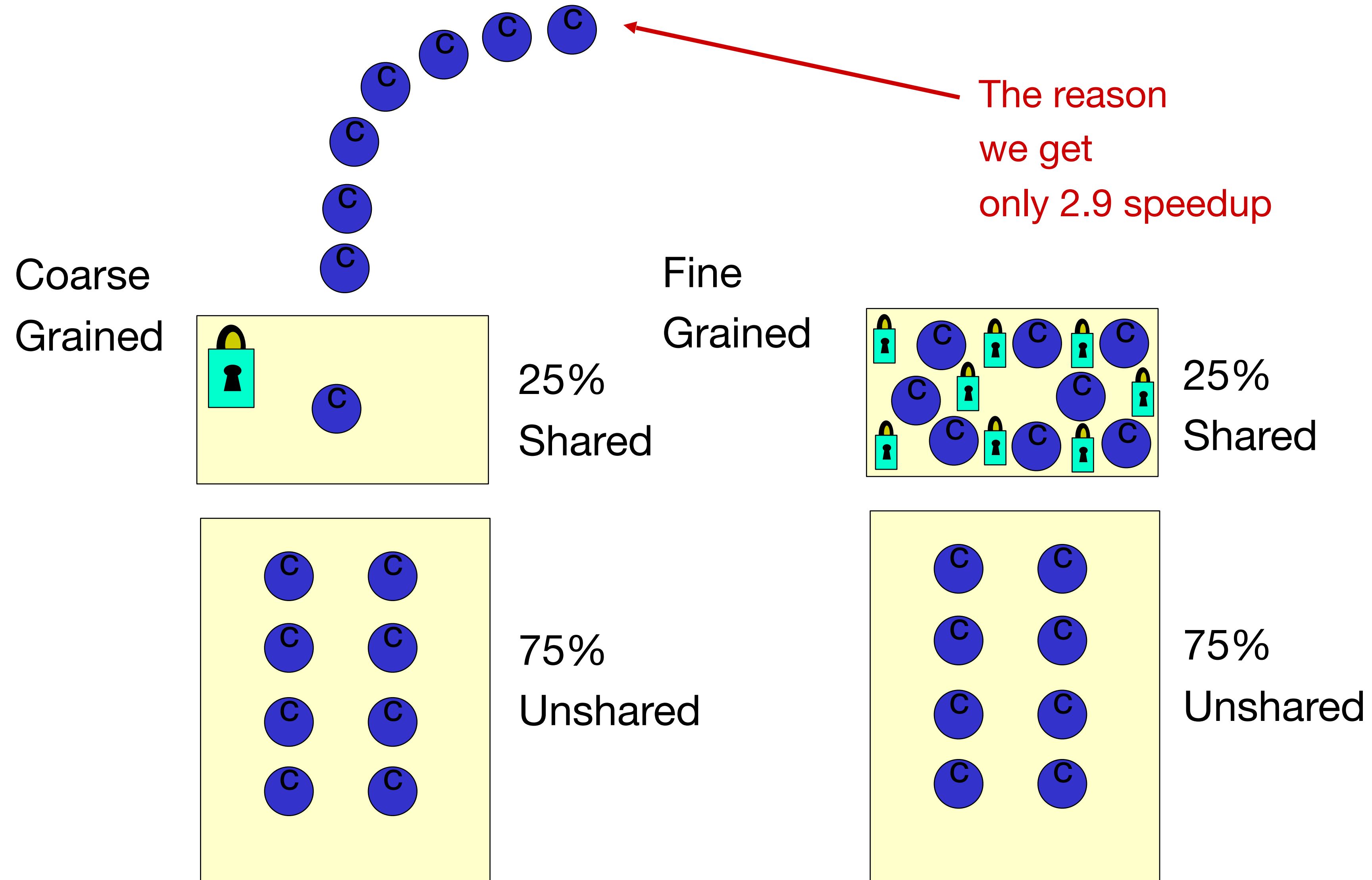
Pay for  $N = 8$  cores

$$\text{SequentialPart} = 25\%$$

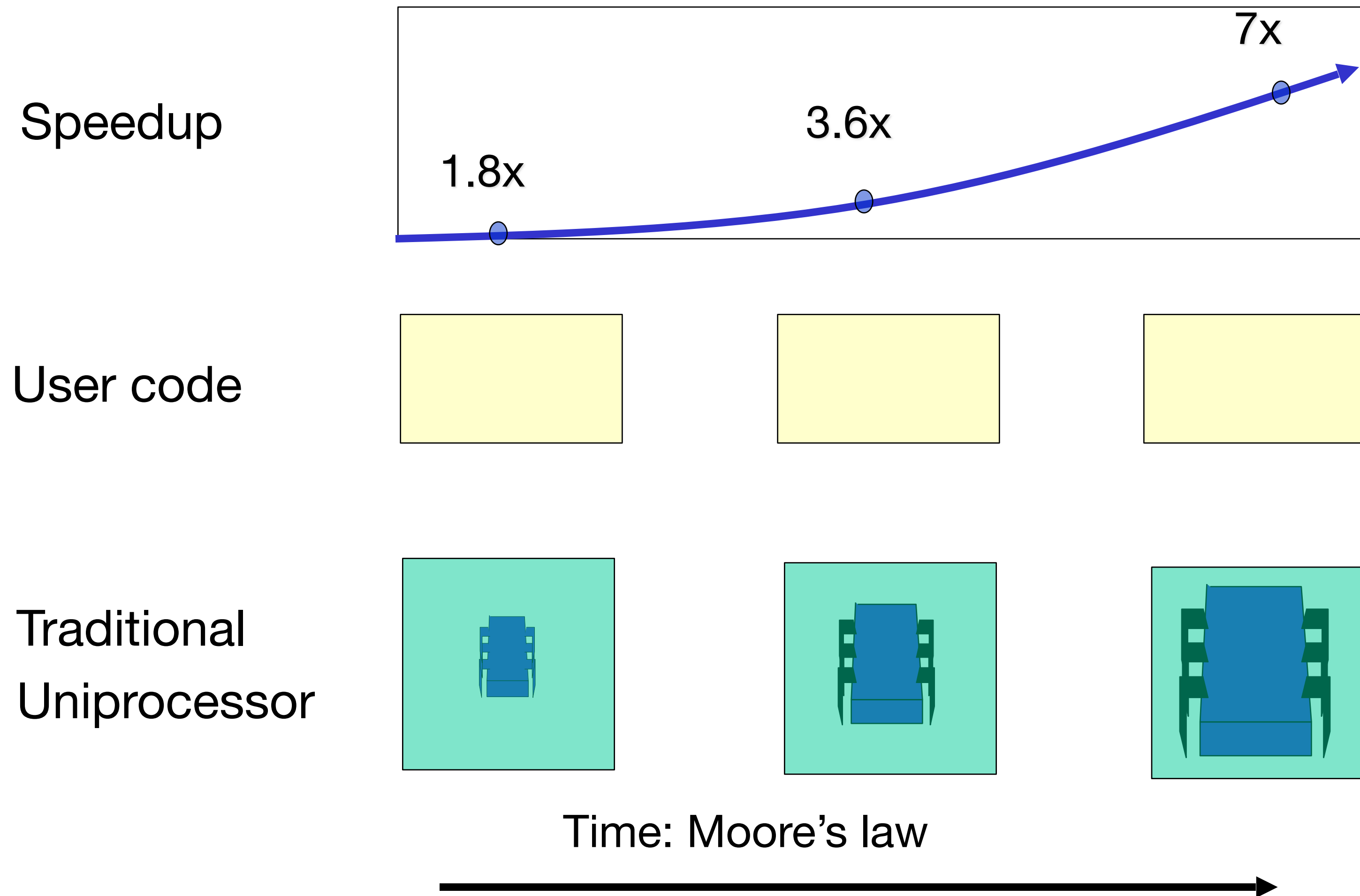
Speedup = only 2.9 times!

Must parallelize applications on a very fine grain!

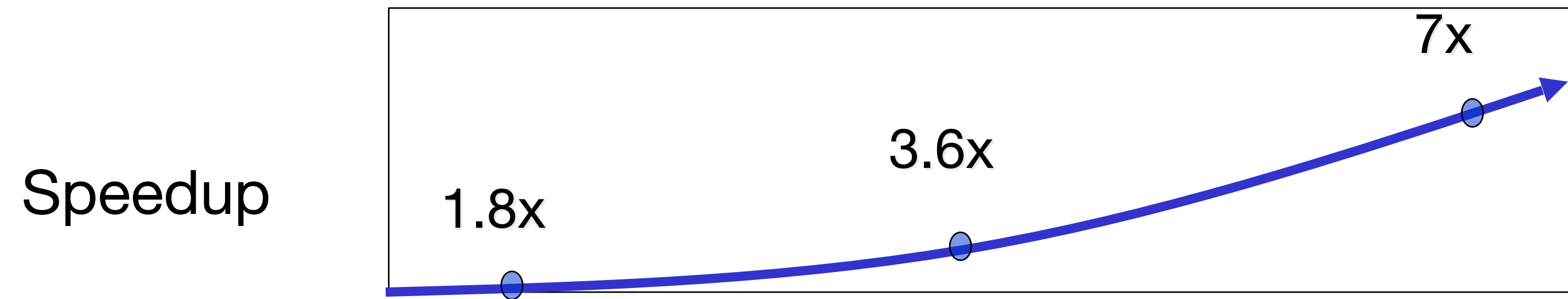
# Need Fine-Grained Locking



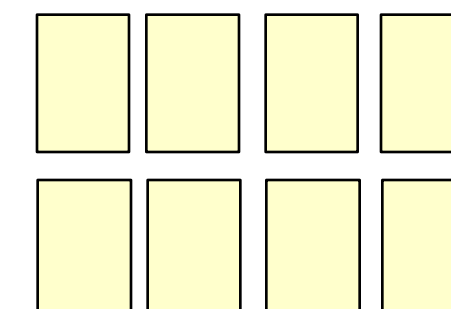
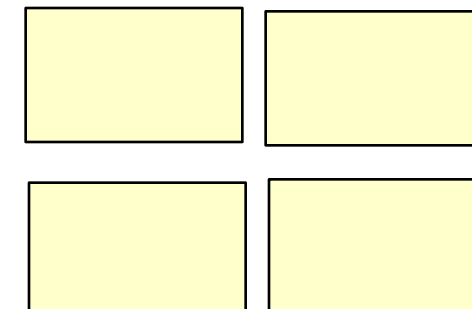
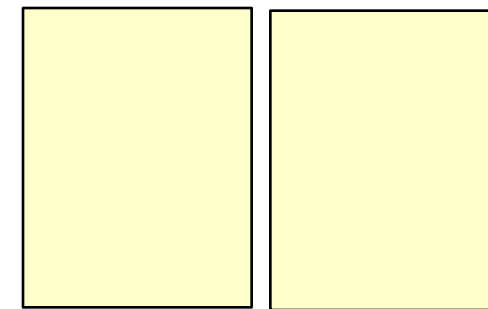
# Traditional Scaling Process



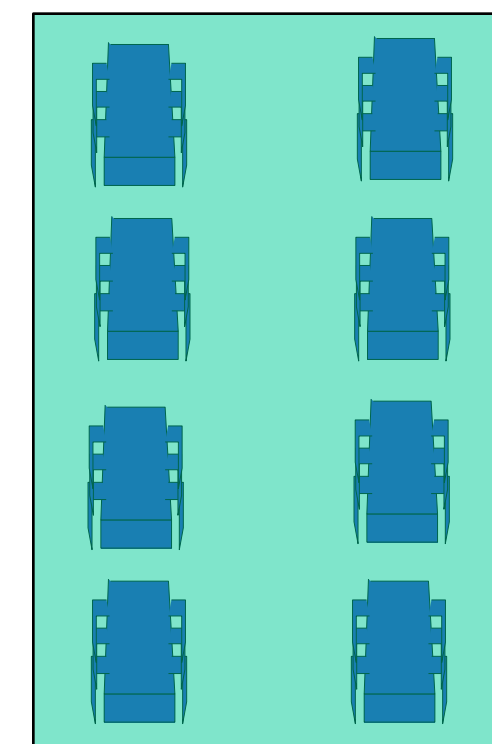
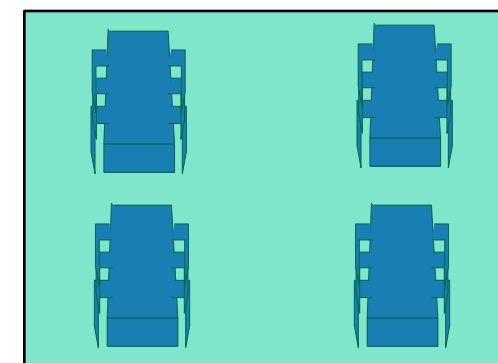
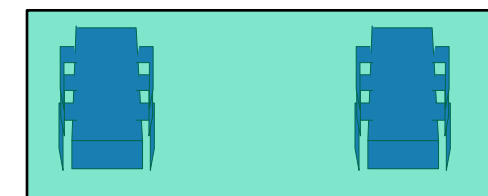
# Multicore Scaling Process



User code



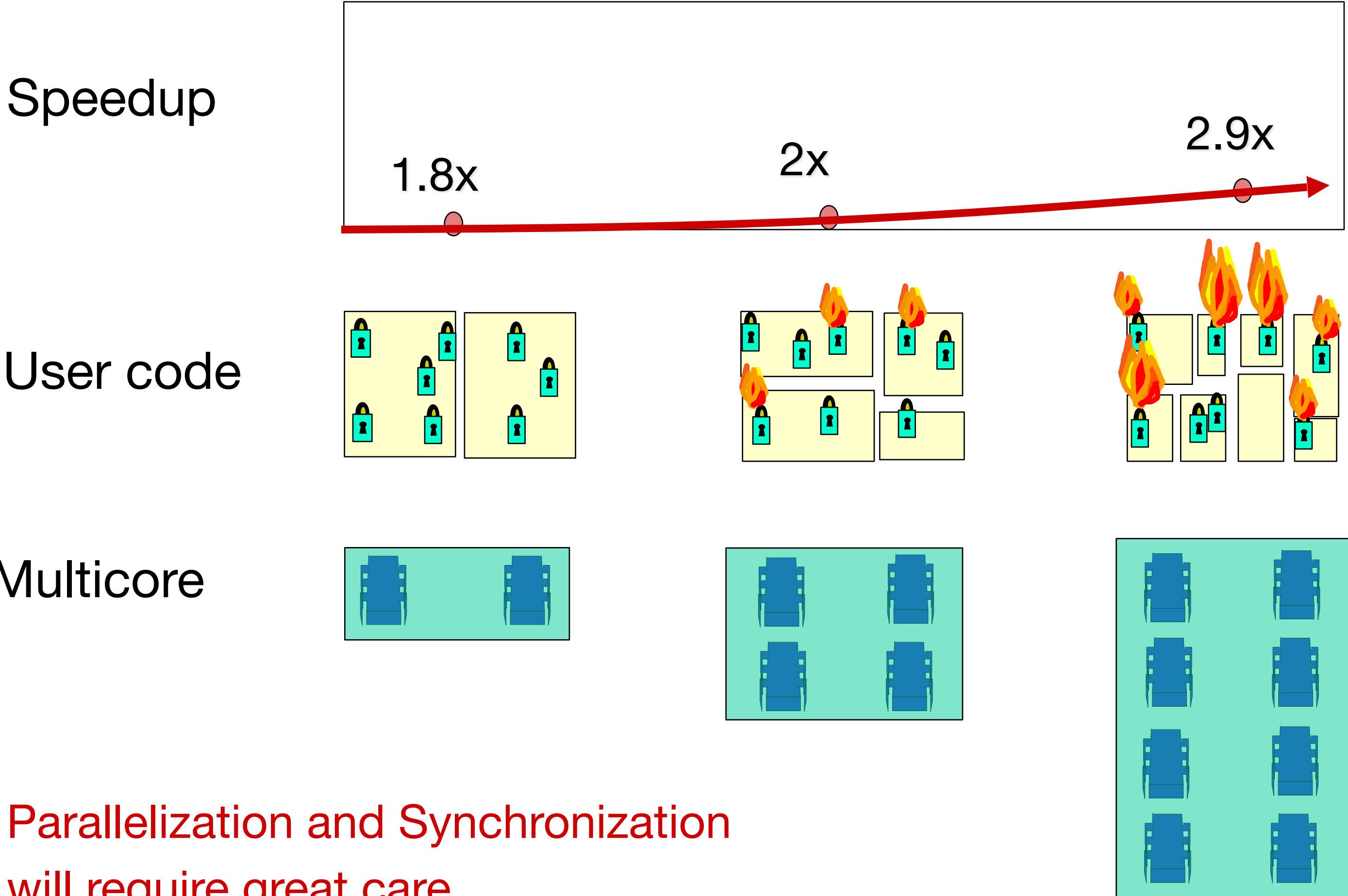
Multicore



As noted, not so simple...



# Real-World Scaling Process



# Multicore Programming

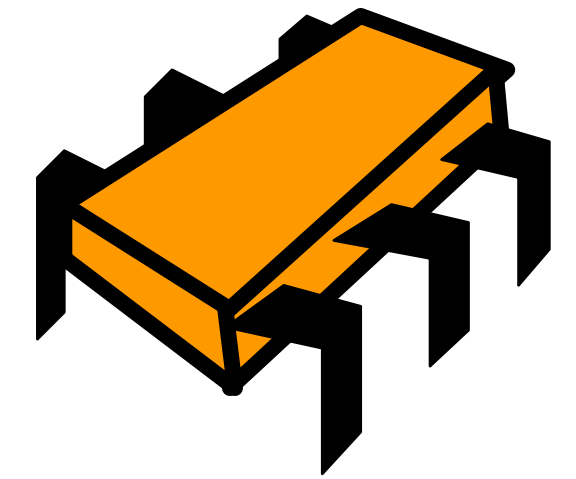
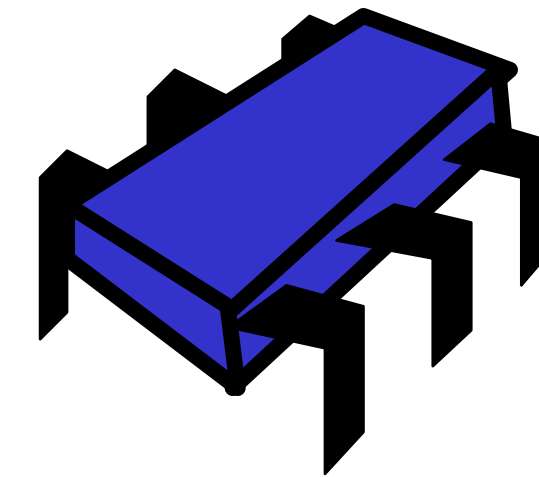
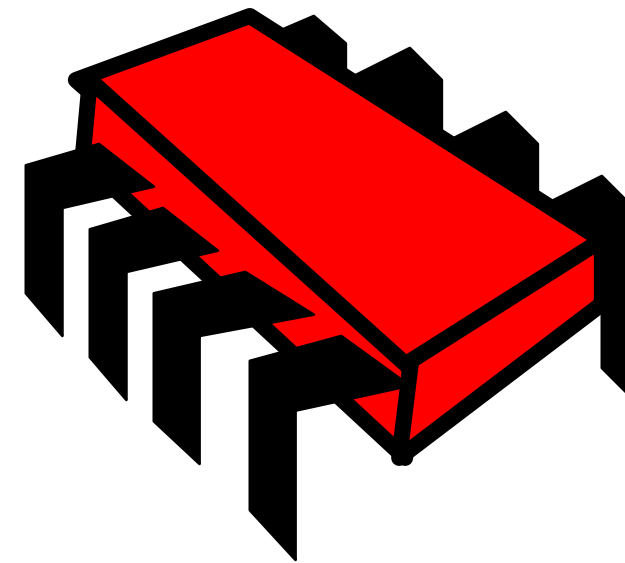
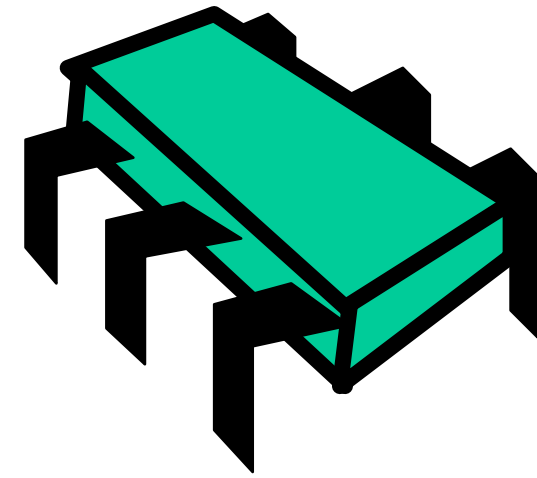
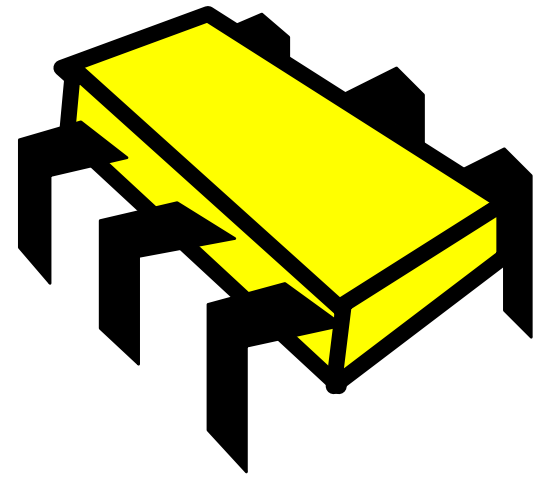
- Nothing is free - if we want to take advantage of multicore machines, we need to work hard at designing our algorithms and systems to be amenable to parallelization
- We are at the dawn of a new era...
  - Still a lot of research and development to be done
- Moving on now: what happens when each CPU is on a different physical machine, connected by a network?

# Today

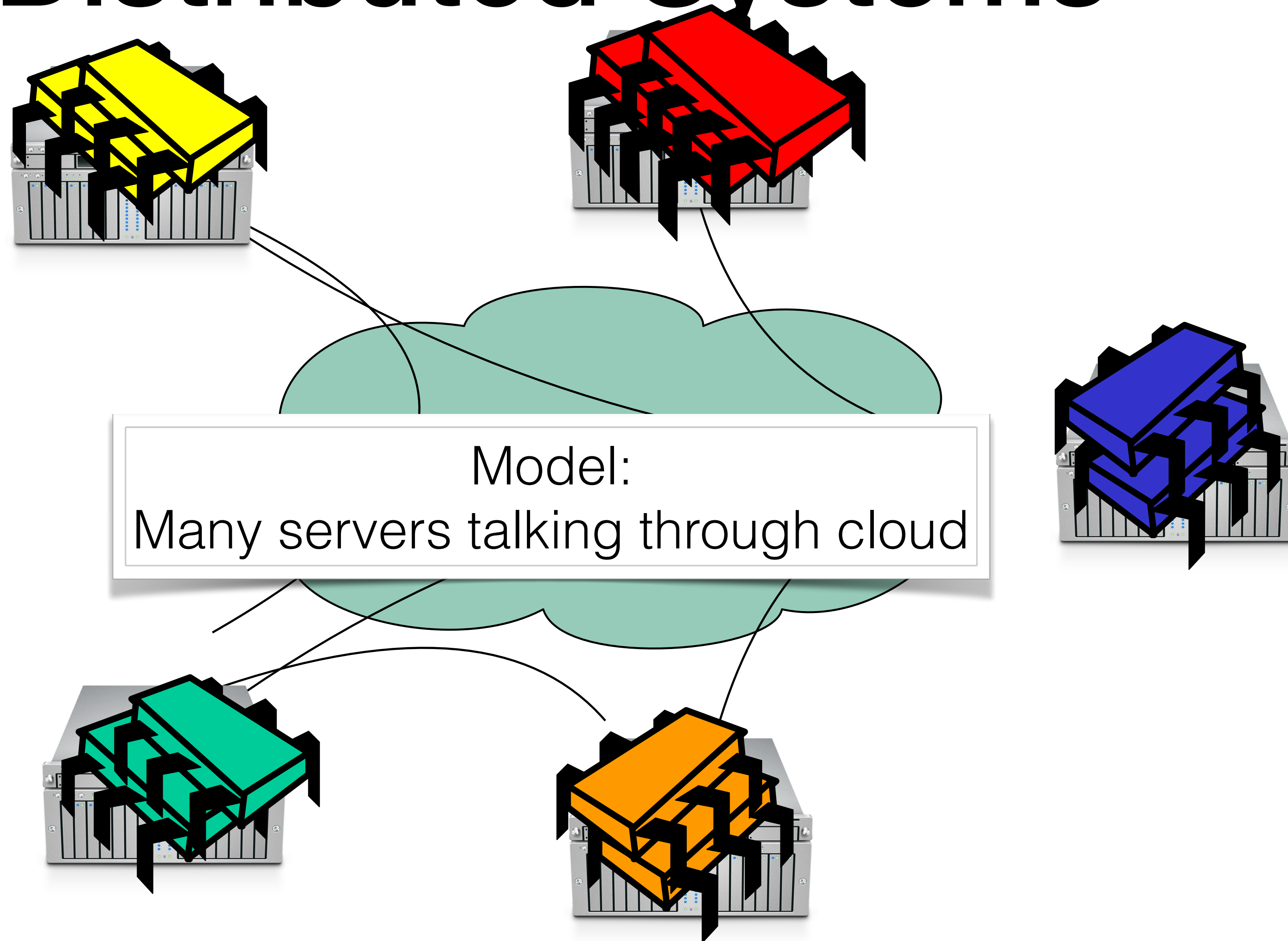
- Gentle introduction to distributed computation
- Computer networks - what do they mean for us?
- Reminders:
  - Midterm
  - HW2

# More Abstractions

- Process + Thread -> one computer
- How can we abstract many computers working together?
- What does that even look like?



# Distributed Systems

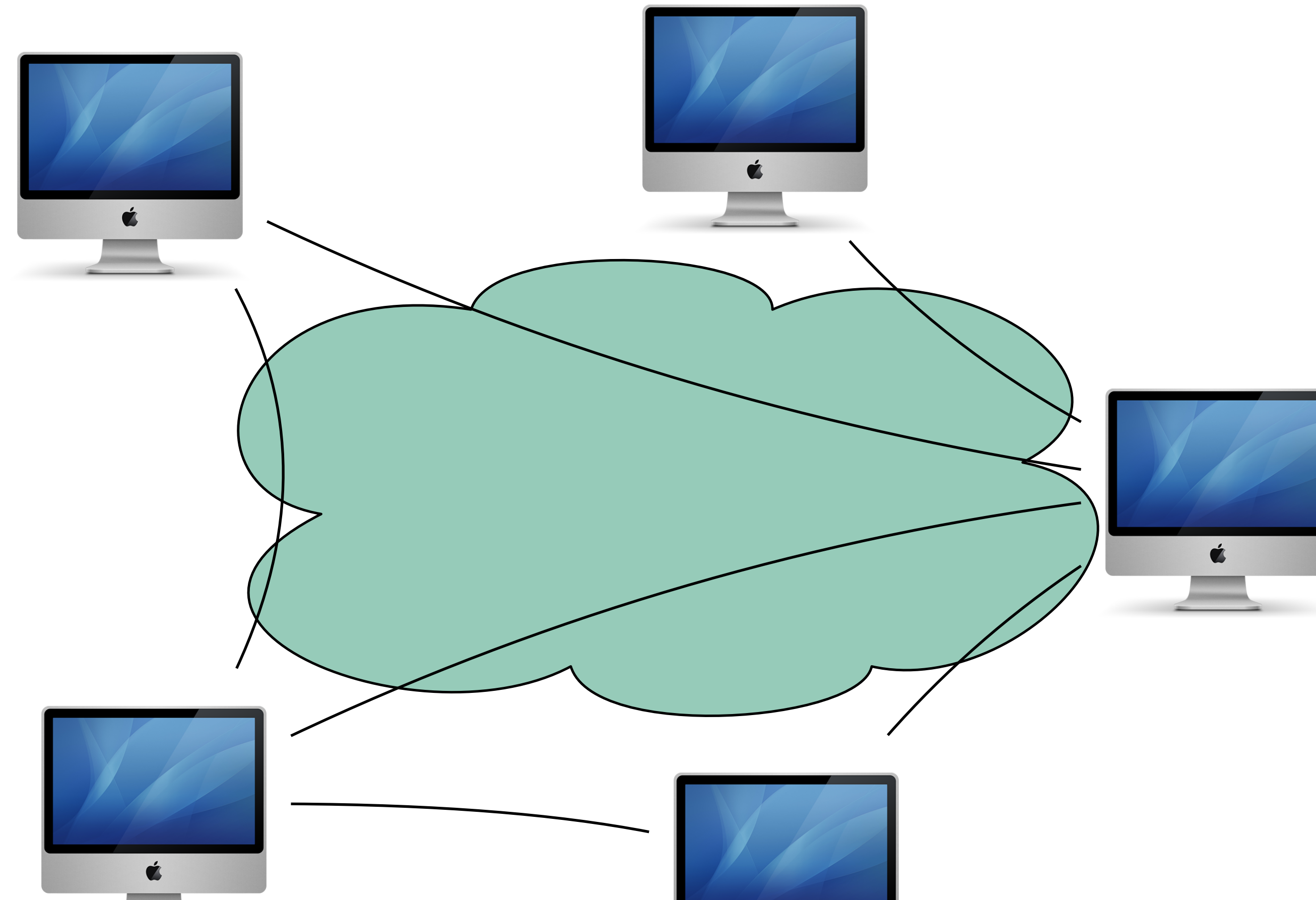


# Distributed Systems



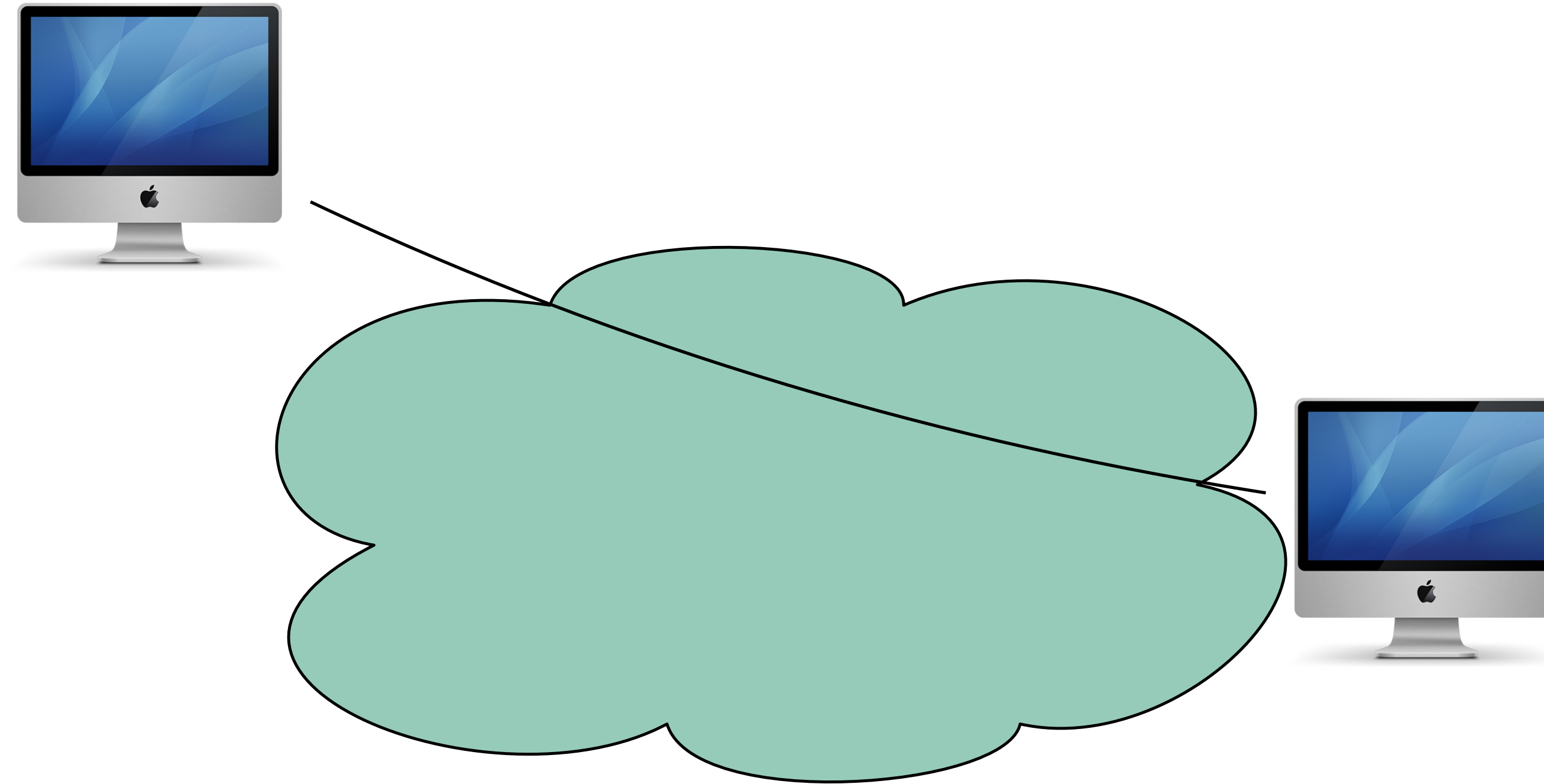
Model:  
Servers and Clients talking through cloud

# Distributed Systems



Model:  
Many clients talking through cloud

# Distributed Systems



Model:  
Two clients talking through cloud



# Why expand to distributed systems?

- Scalability
- Performance
- Latency
- Availability
- Fault Tolerance

“Distributed Systems for Fun and Profit”, Takada

# Distributed Systems Goals

- **Scalability**
- Performance
- Latency
- Availability
- Fault Tolerance

“the ability of a system, network, or process, to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth.”

“Distributed Systems for Fun and Profit”, Takada

# Distributed Systems Goals

- Scalability
- **Performance**
- Latency
- Availability
- Fault Tolerance

“is characterized by the amount of useful work accomplished by a computer system compared to the time and resources used.”

# Distributed Systems Goals

- Scalability
- Performance
- **Latency**
- Availability
- Fault Tolerance

“The state of being latent; delay, a period between the initiation of something and the it becoming visible.”

# Distributed Systems Goals

- Scalability
- Performance
- Latency
- **Availability**
- Fault Tolerance

“the proportion of time a system is in a functioning condition. If a user cannot access the system, it is said to be unavailable.”

Availability = uptime / (uptime + downtime).

Often measured in “nines”

Availability %	Downtime/year
90%	>1 month
99%	< 4 days
99.9%	< 9 hours
99.99%	<1 hour
99.999%	5 minutes
99.9999%	31 seconds

# Distributed Systems Goals

- Scalability
- Performance
- Latency
- Availability
- **Fault Tolerance**

“ability of a system to behave in a well-defined manner once faults occur”

## What kind of faults?

Disks fail

Power supplies fail

Networking fails

Security breached

Power goes out      Datacenter goes offline

# More machines, more problems

- Say there's a 1% chance of having some hardware failure occur to a machine (power supply burns out, hard disk crashes, etc)
- Now I have 10 machines
- Probability(at least one fails) =  $1 - \text{Probability}(\text{no machine fails}) = 1 - (1 - .01)^{10} = 10\%$
- 100 machines -> 63%
- 200 machines -> 87%
- So obviously just adding more machines doesn't solve fault tolerance

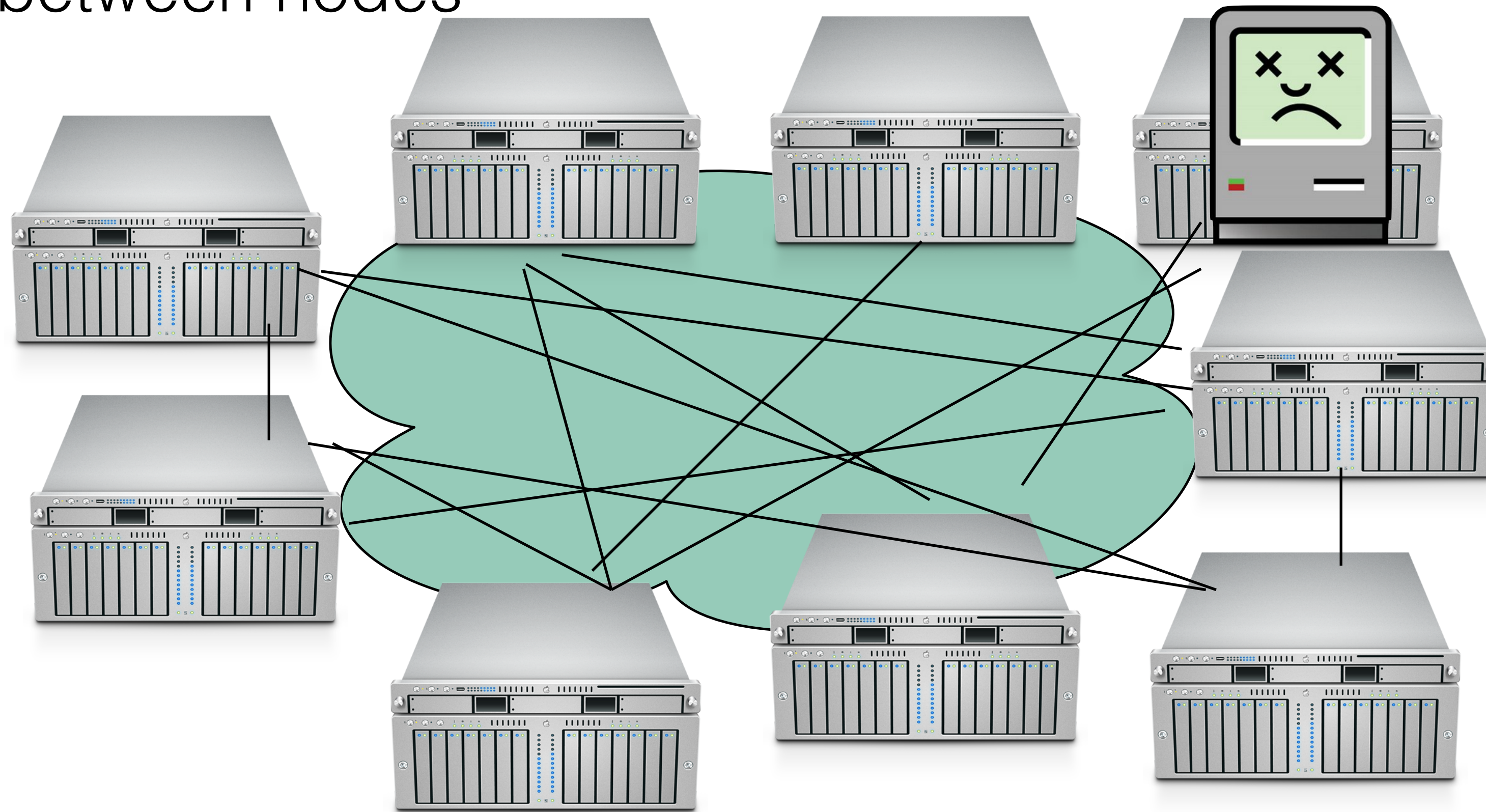
# More machines, more problems

- PLUS, the network may be:
  - Unreliable
  - Insecure
  - Slow
  - Expensive
  - Limited



# Constraints

- Number of nodes
- Distance between nodes



# Networks as Abstractions

- A network consists of communication links
- Networks have several “interesting” properties we will look at
  - Latency
  - Failure modes
- What is the abstraction?

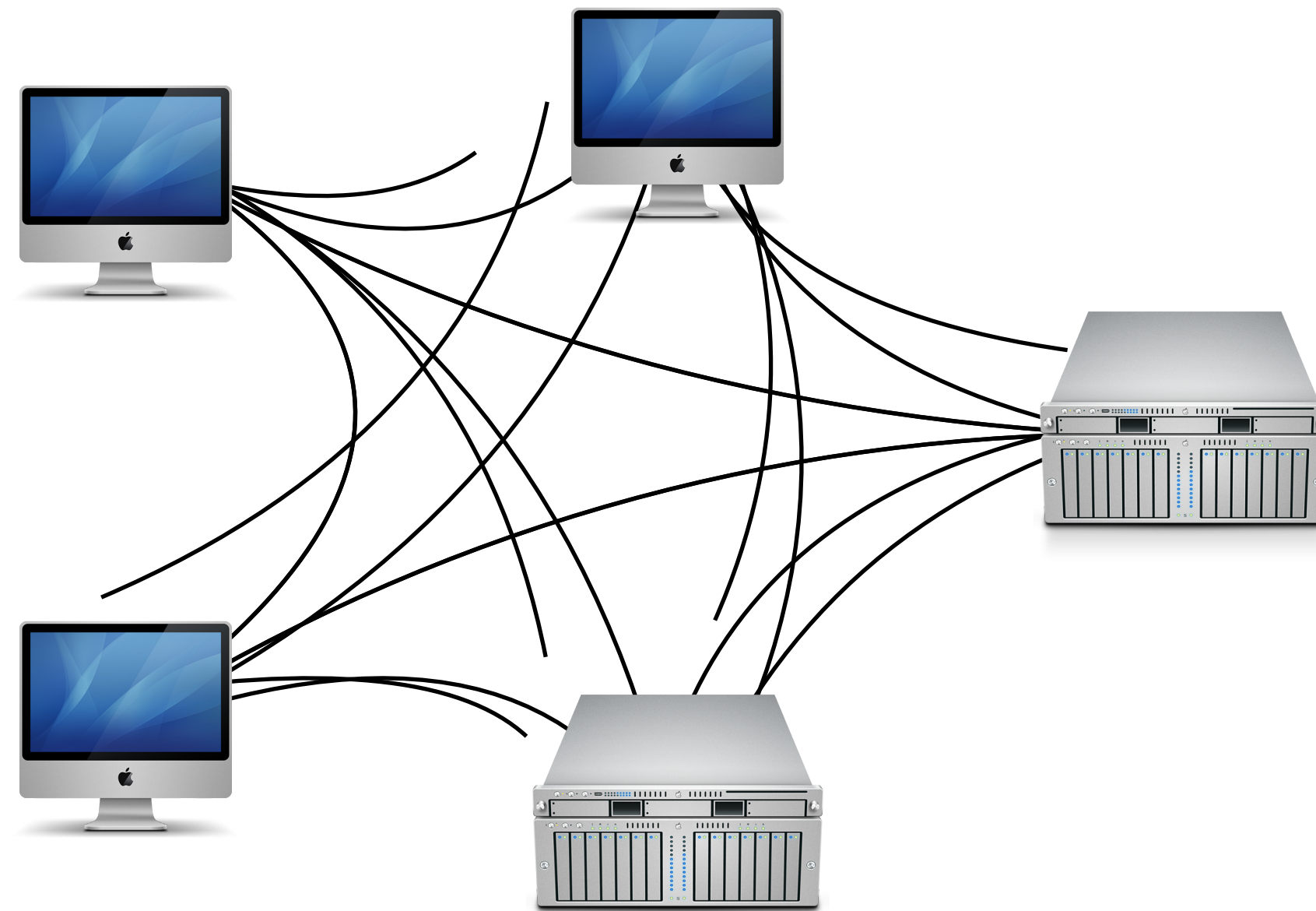


# Networks as Abstractions

- Stuff goes in, stuff goes out?
- Not a perfect abstraction, because:
  - Speed of light (1 foot/nanosecond)
  - Communication links exist in uncontrolled/hostile environments
  - Communication links may be bandwidth limited (tough to reach even 100MB/sec)
- In contrast to a single computer, where:
  - Distances are measured in mm, not feet
  - Physical concerns can be addressed all at once
  - Bandwidth is plentiful (easily GB/sec)

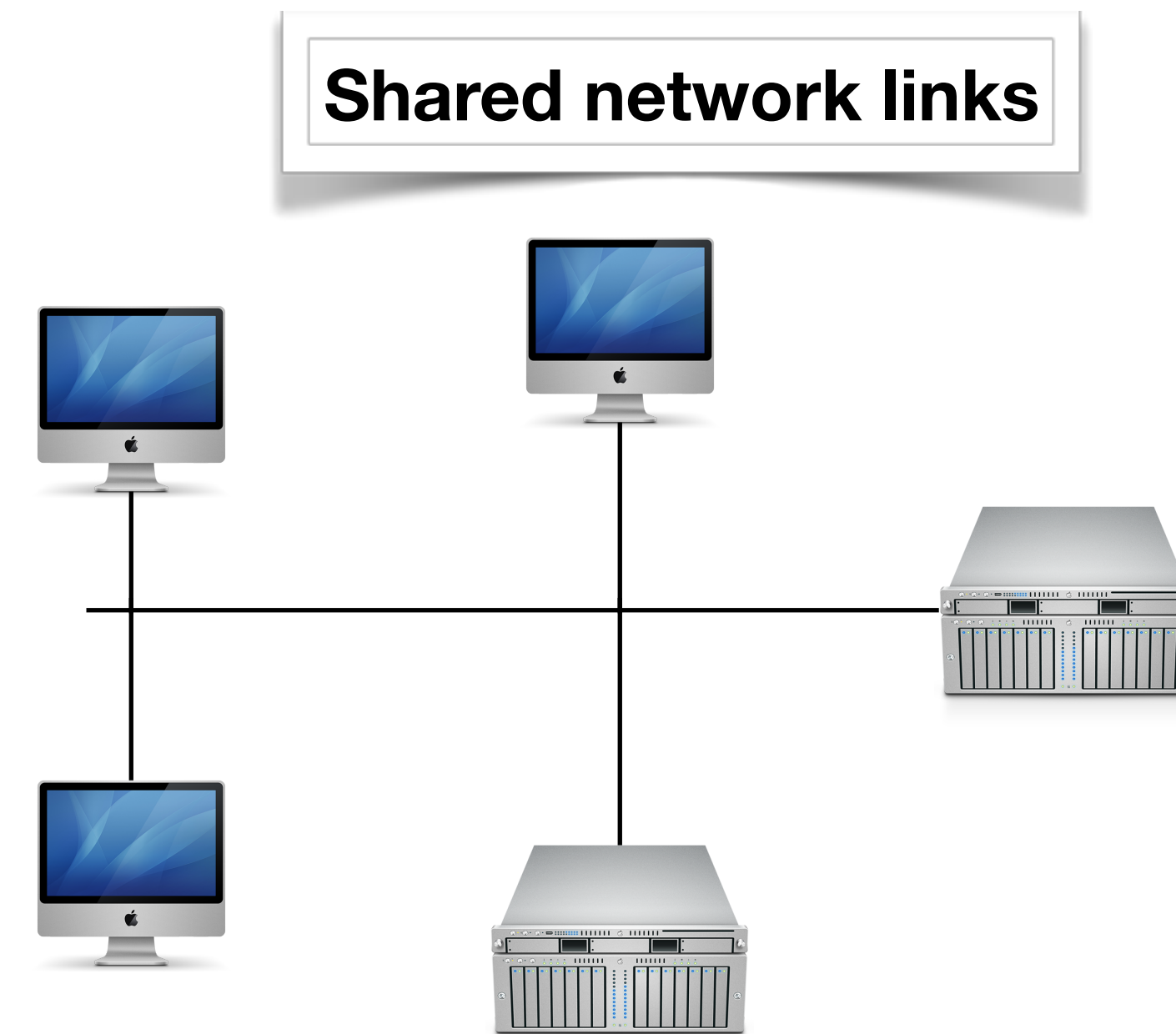
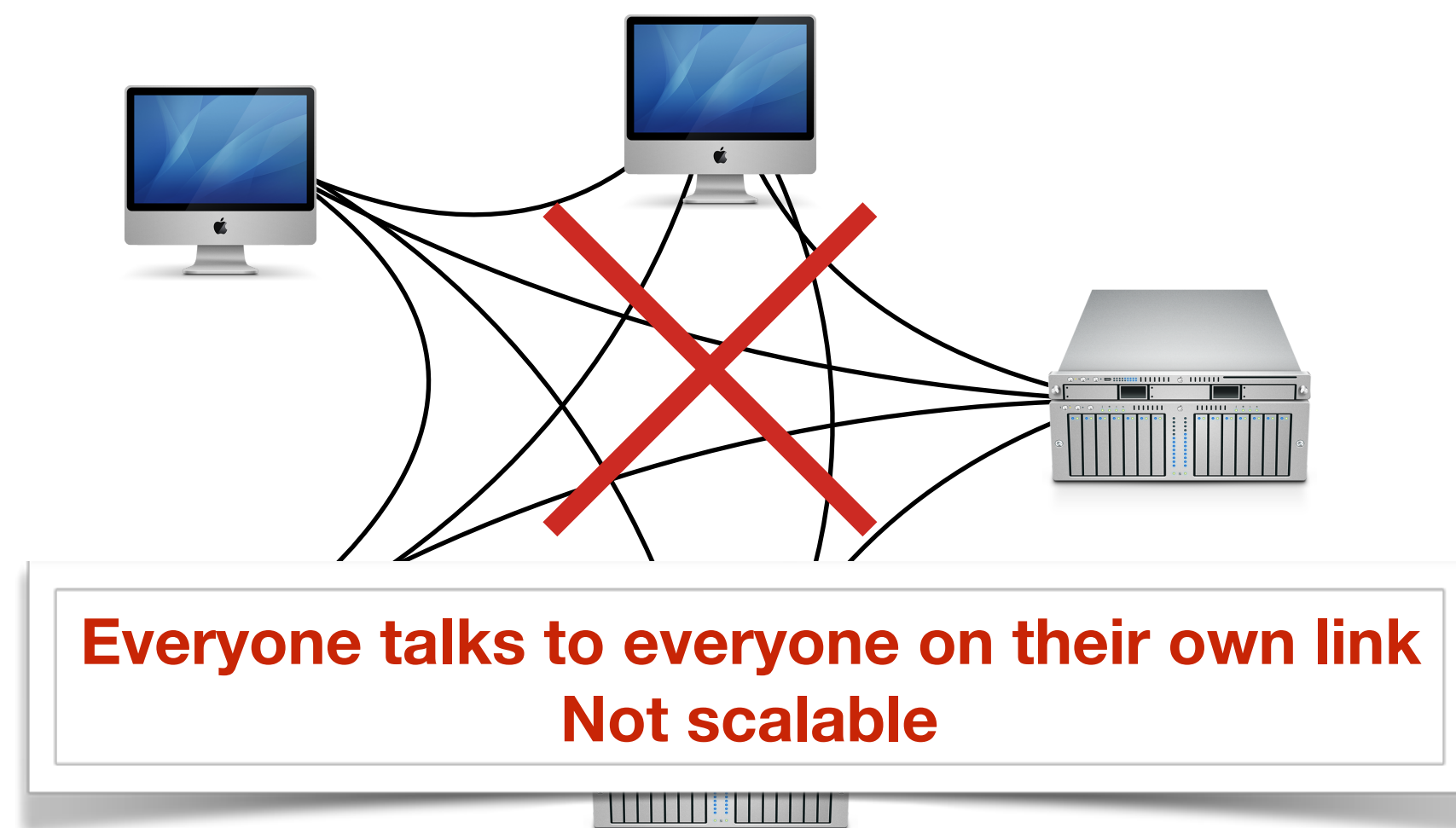
# Networks are Shared

- With processes, we considered how one CPU could be shared between multiple programs running at once
- With networks, communication links are probably shared even more widely



# Networks are Shared

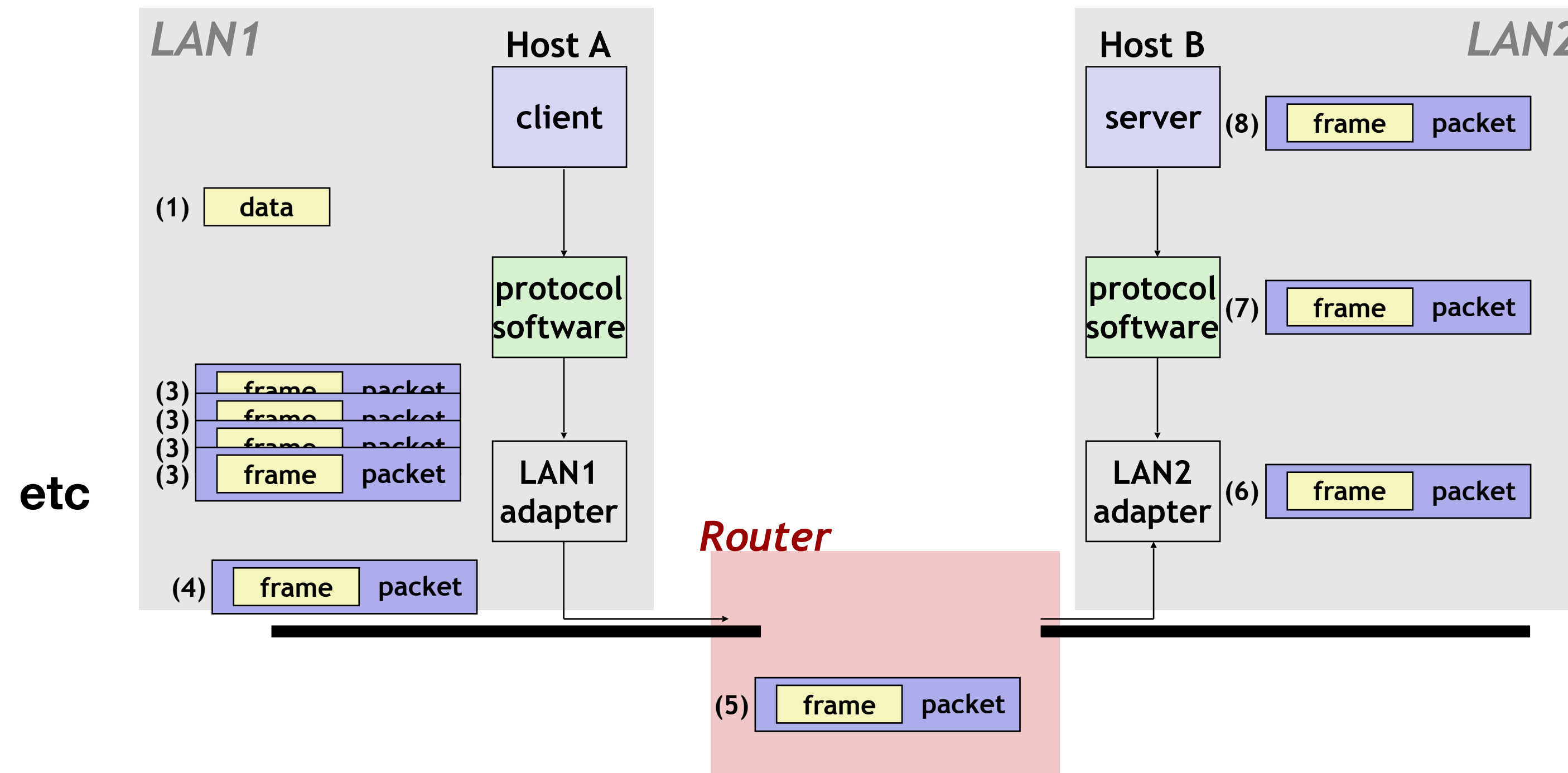
- With processes, we considered how one CPU could be shared between multiple programs running at once
- With networks, communication links are probably shared even more widely



# Network as Abstractions

- What do we send, what gets received?
- At the lowest level, we call what gets sent **frames**
- Each frame is limited in size
  - Ethernet: max 1522 **bytes**
- Frame is packed with source/destination info into a **packet**
- Network knows what to do with packets to get them to their destination

# Networks as Abstractions



PH: Internet packet header  
FH: LAN frame header

# Packet Switching Delays

- As these packets flow through a network and are routed, we might see delays due to:
  - Propagation (traveling across the link, speed of light, etc)
  - Transmission delay (big packets take longer to transmit)
  - Processing delay (once switch sees packet, might be slow to process)
  - Queuing delay (link might be busy)



# Packet Loss

- Some packets could be delayed, others might never reach their target, due to:
  - Buffers overflowing (e.g. on switch)
- Networks are usually considered **best-effort**
  - Aka third-class mail
  - We'll try to get your packet there, but if it doesn't, sorry.
- Solved by requiring recipient to send a confirmation message was received
  - If no confirmation received, assume didn't get sent
- What happens to duplicates?
  - Each message includes a unique ID, can be discarded if duplicate received

# Resending Packets

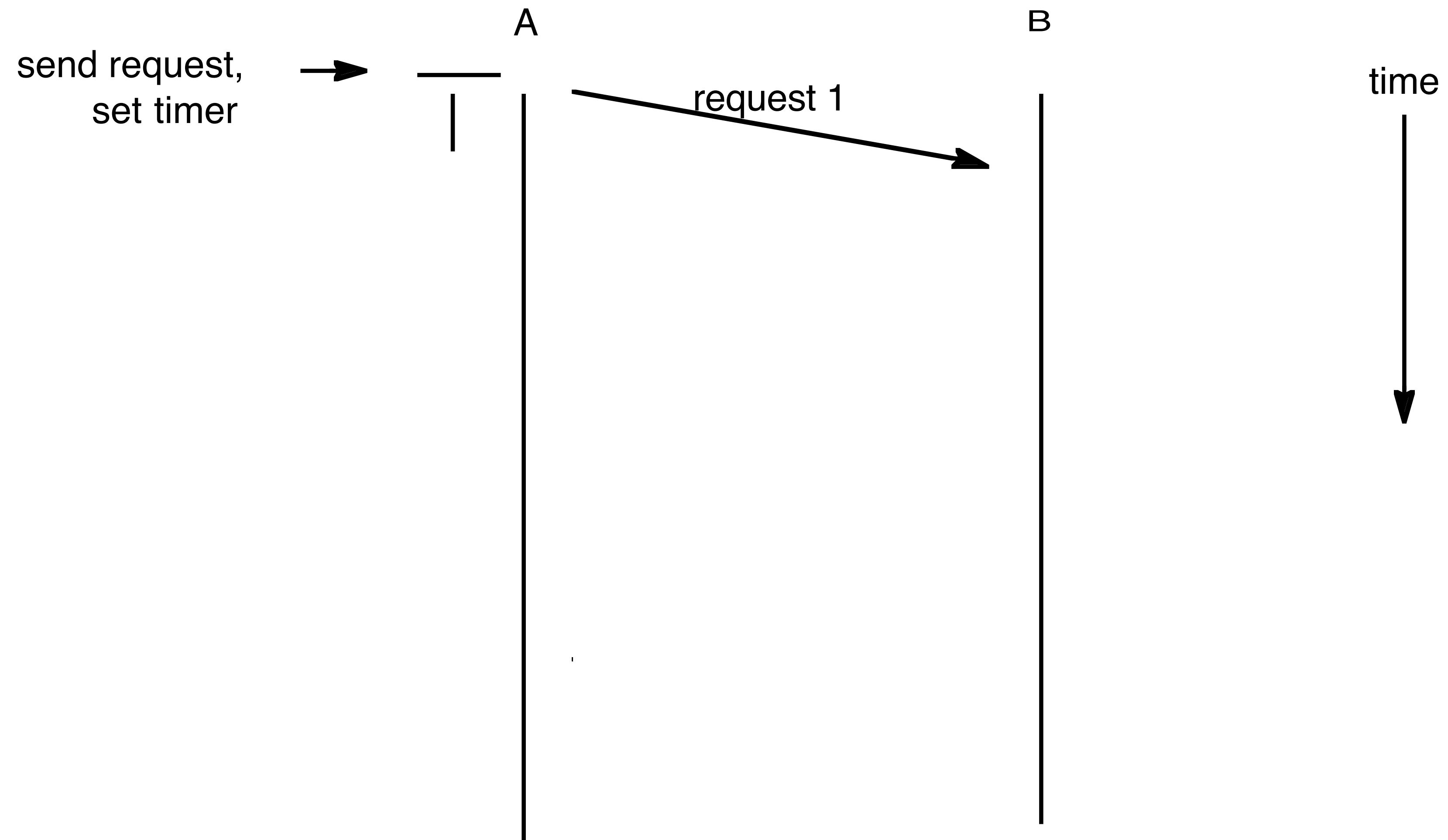
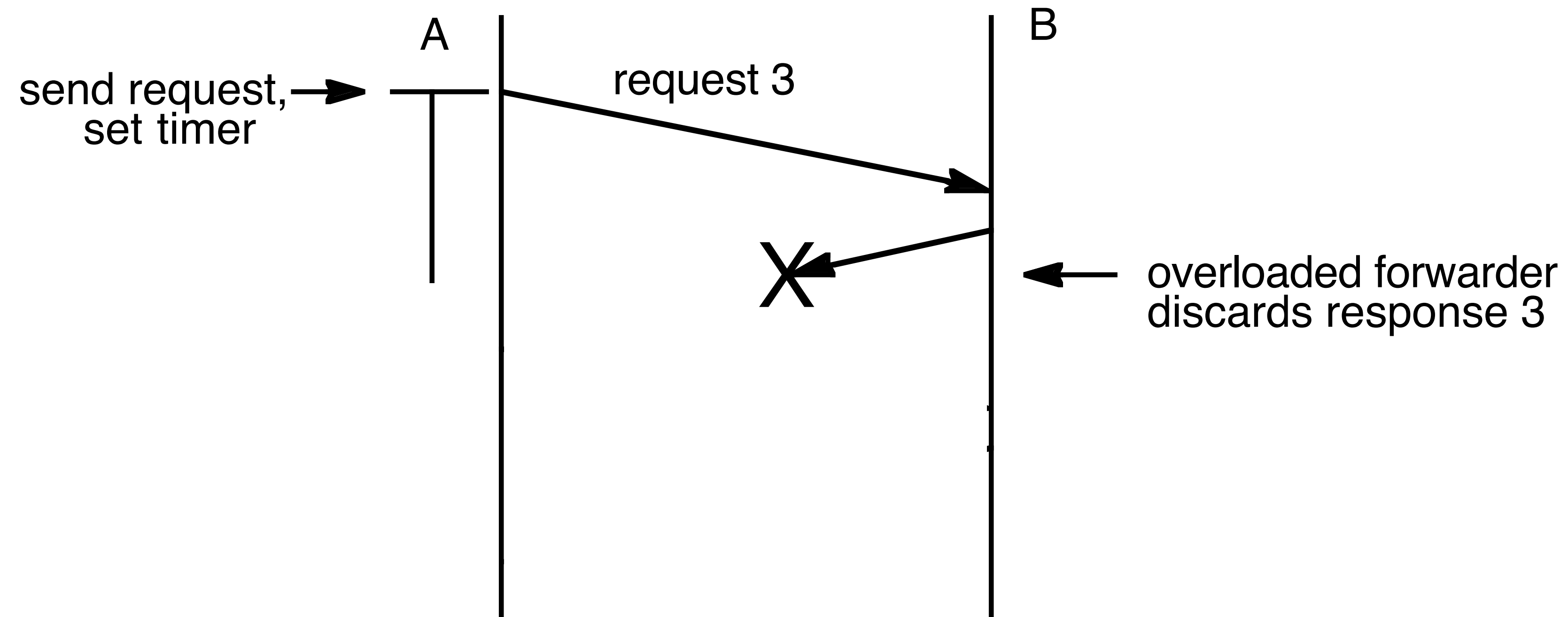
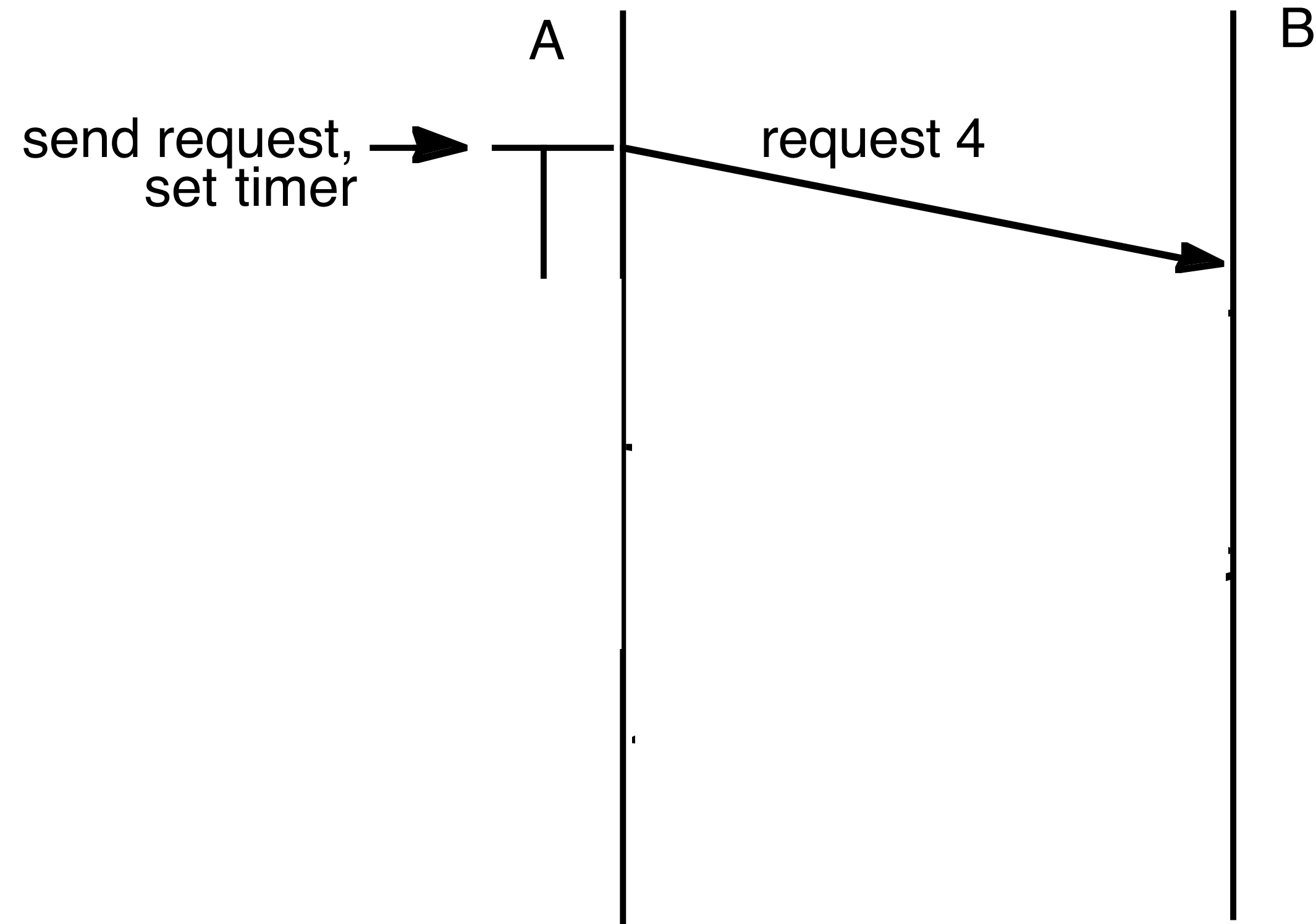


Fig © Saltzer & Kaashoek

# Resending Packets



# Resending Packets

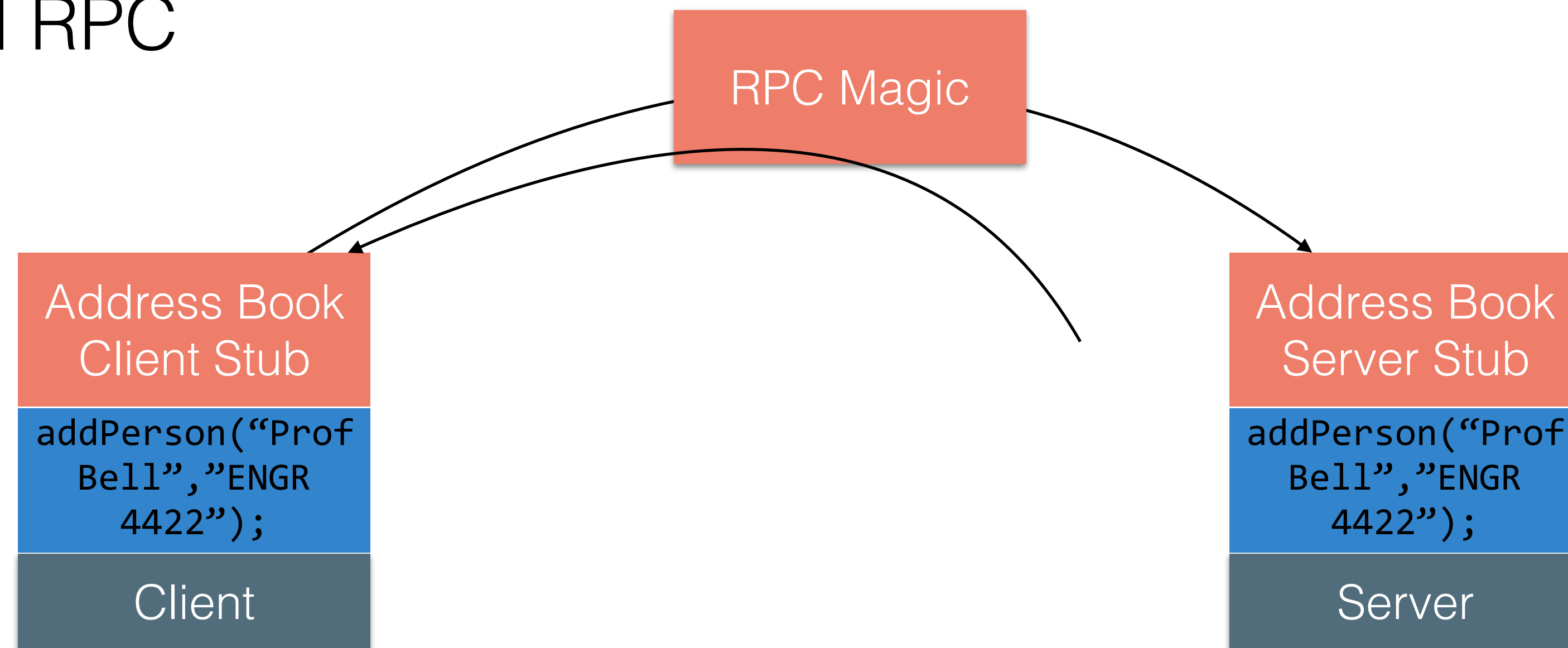


# Resending Packets

- That ID is **really** important to put on the packets!
- Note: it works, but can result in **lots** of duplicate packets sent back and forth
- Also, note: no guarantee that packets are delivered in order!

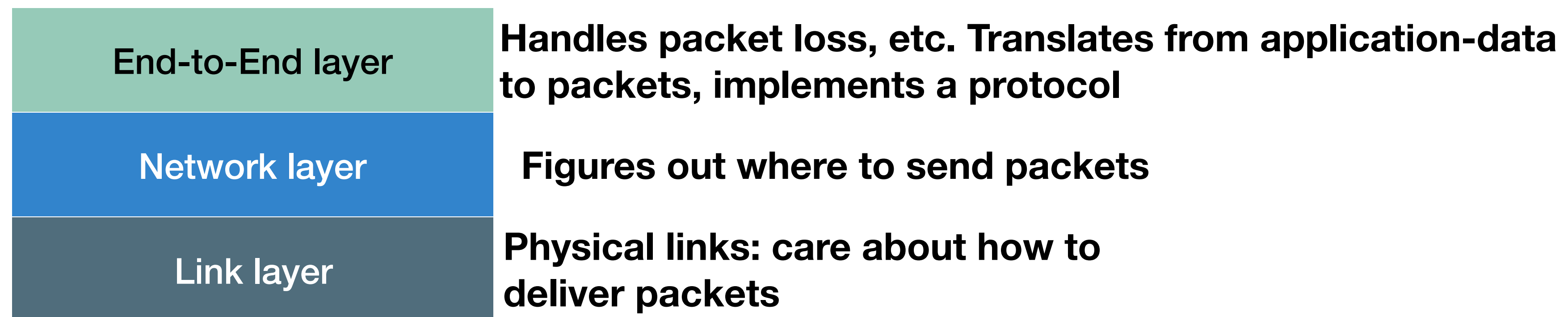
# Networks as Abstractions

- Obviously, we don't think or care about packets
- We think and care about sending data!
- We want abstractions, like RPC (Remote Procedure Calls)
- Abstractions (try to) hide the complexity of what's below them
- Next class: all RPC



# 3 Layer Abstraction

- The typical network abstraction model has 7 layers
- Take CS 455 if you want to know more about these
- We'll think about 3 abstraction layers, and really focus on the top one



# Transport Protocols

- Anything in the end-to-end layer is likely built on top of some lower level protocol (**more** abstractions)
- TCP, or UDP
- Data integrity (checksumming)
- Ordering control
- Flow control  
(not worrying about congestion)

UDP	✓	TCP	✓
UDP	✗	TCP	✓
UDP	✗	TCP	✓



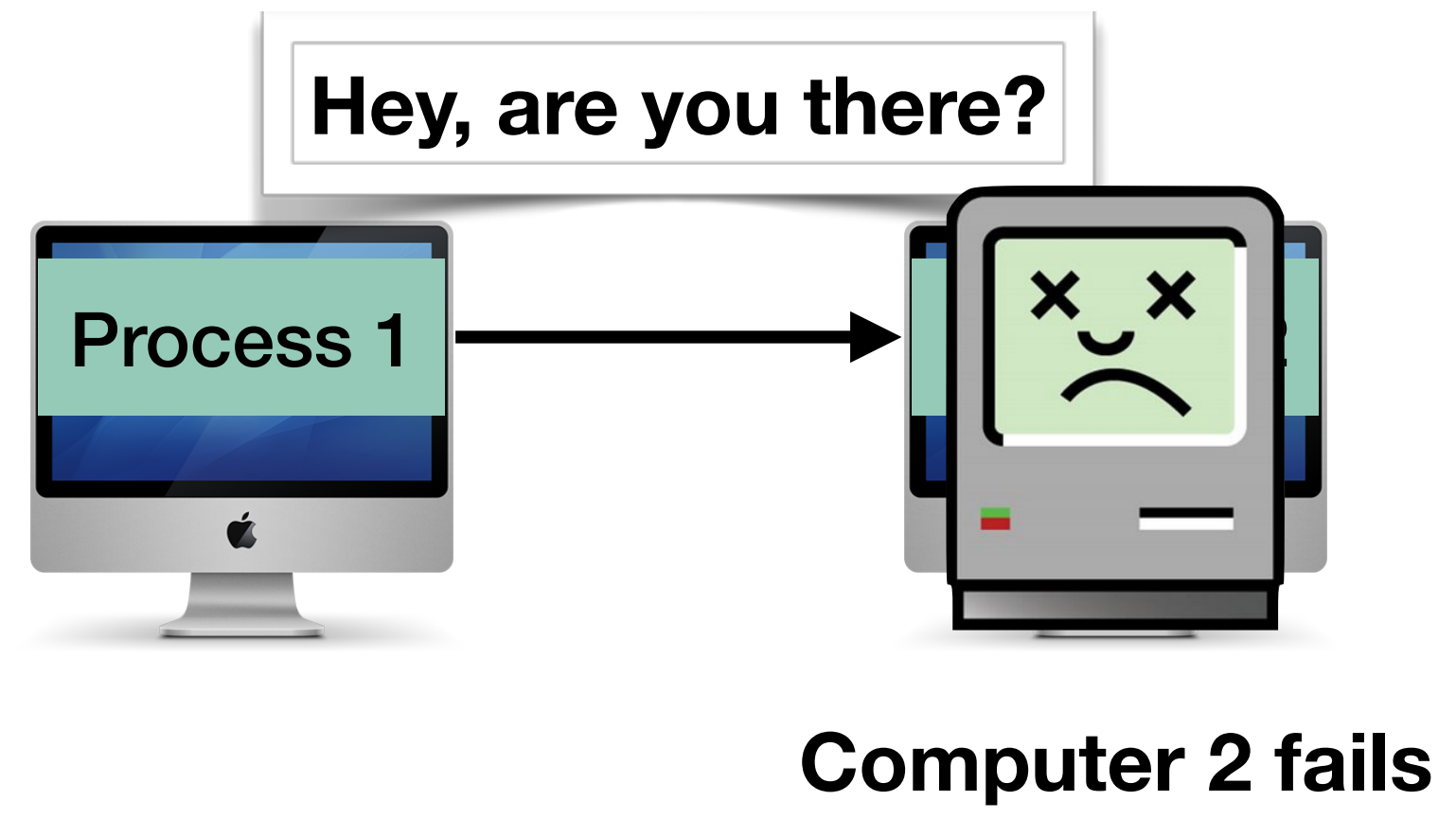
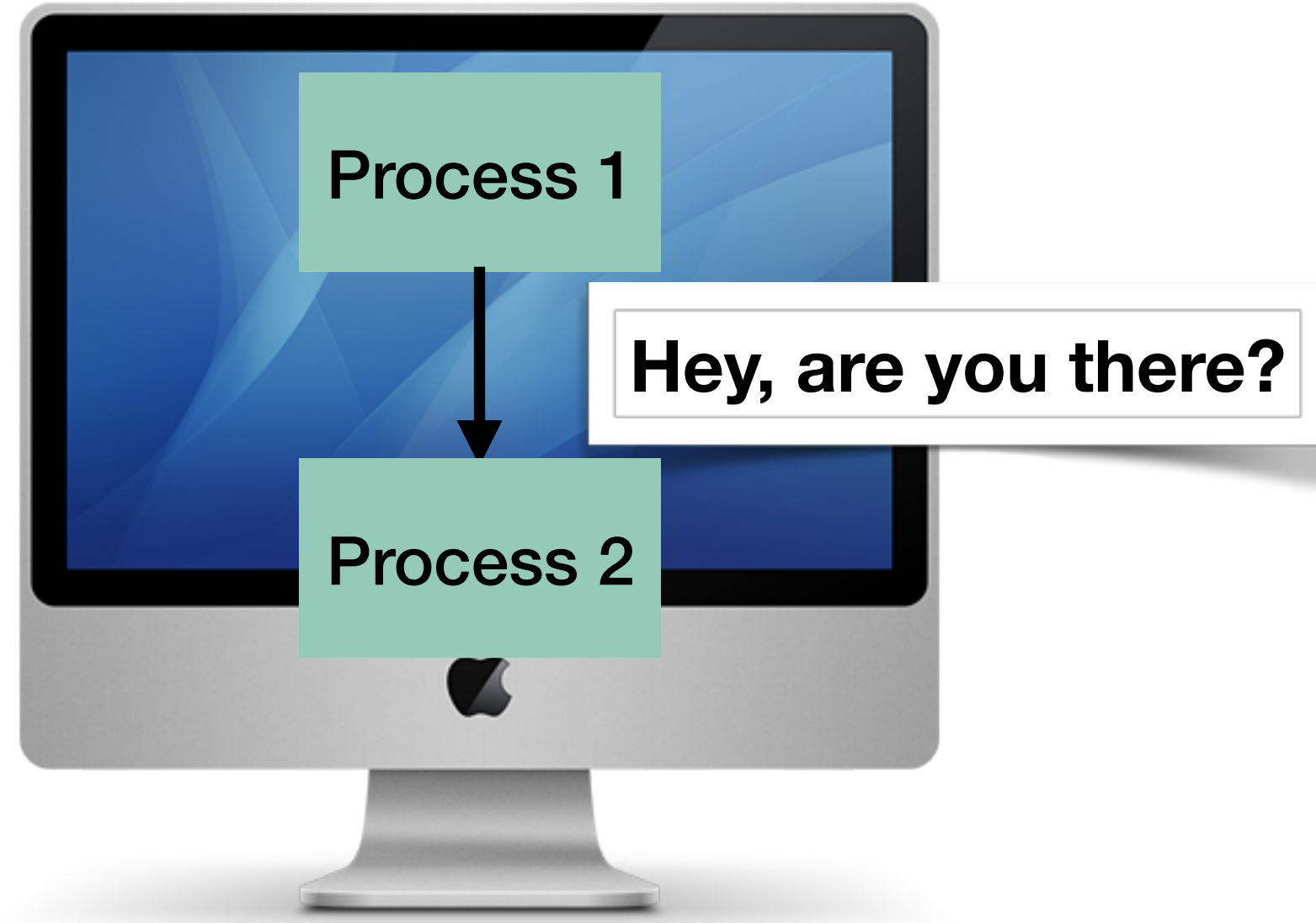
# Reminder: Leaky Abstractions

- From this lecture, you should have found out that networks:
  - Can vary in
    - Data rates
    - Propagation, transmission, queuing and processing delays
  - Traverse hostile environments and may corrupt data or stop working
  - Even best-effort networks have:
    - Variable delays, transmission rates, can discard packets, duplicate packets, have a maximum packet length, can reorder packets
- Even if using TCP, this can still show up!
  - Messages might still arrive late

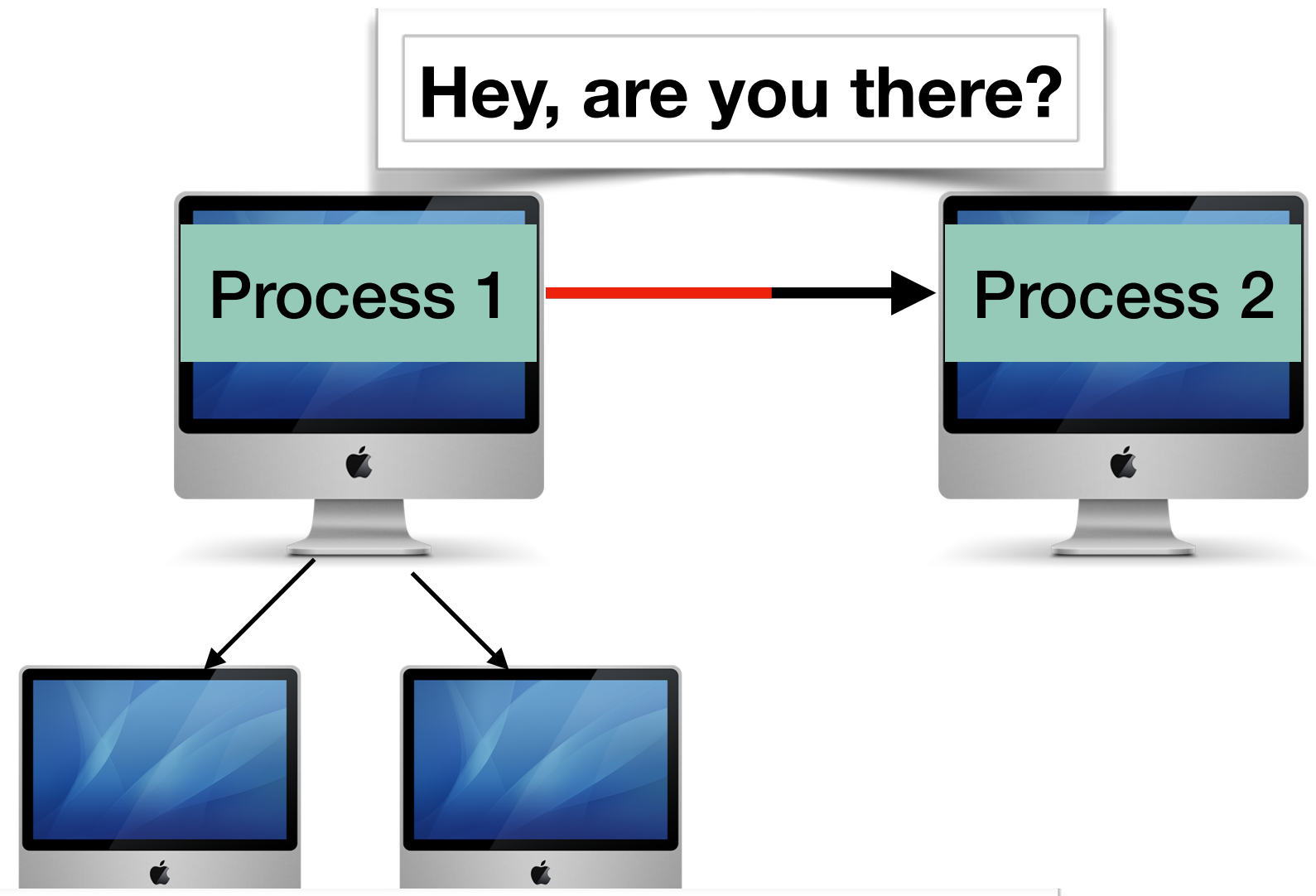
# Sockets as an Abstraction

- Simplest way to build our end-to-end layer is using a **socket**, which gives us an interface to TCP or UDP
- Socket looks **just** like reading/writing to a file (e.g. file descriptor in C, InputStream in Java)
- Sockets are identified by:
  - IP address - identifies the device on the network
  - Port number - identifies the application on the device

# Preview for Next Class



**Spoiler alert: You can not tell the difference in a distributed system between a computer failing and network being arbitrarily slow!**



...well, I can still talk to these guys so I guess internet is ok

# This work is licensed under a Creative Commons Attribution-ShareAlike license

- This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/>
- You are free to:
  - Share — copy and redistribute the material in any medium or format
  - Adapt — remix, transform, and build upon the material
  - for any purpose, even commercially.
- Under the following terms:
  - Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.