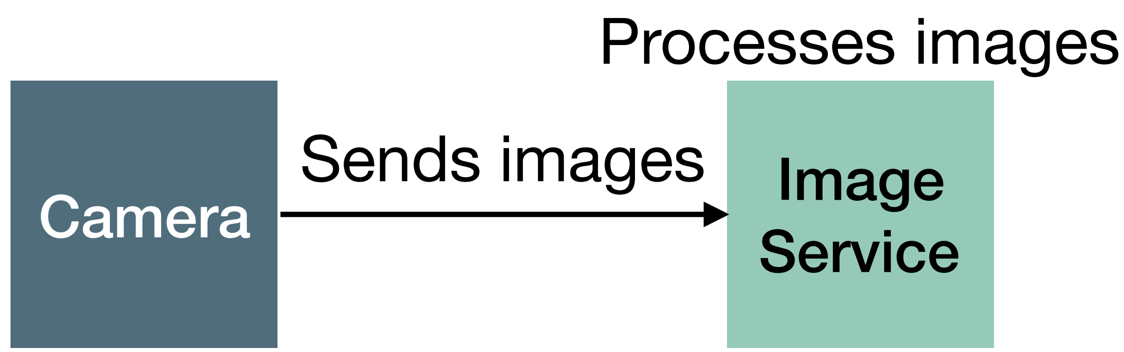
Async Programming & Networking

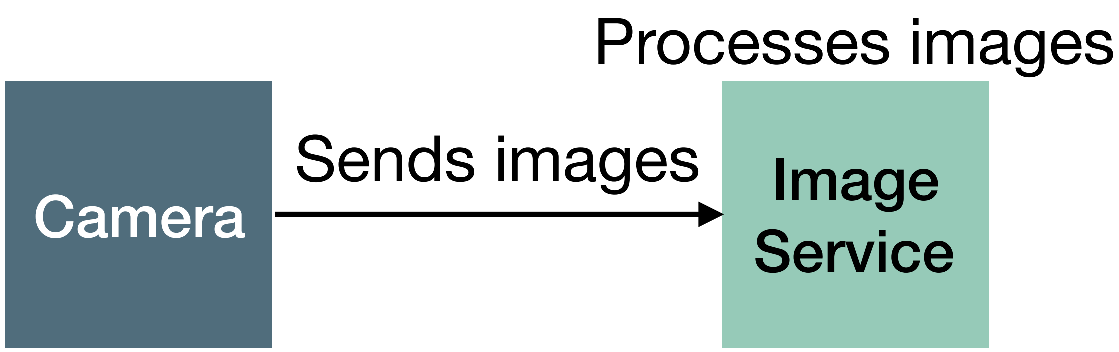
CS 475, Spring 2018
Concurrent & Distributed Systems

GEORGE MASON UNIVERSITY

Review: Resource Metric

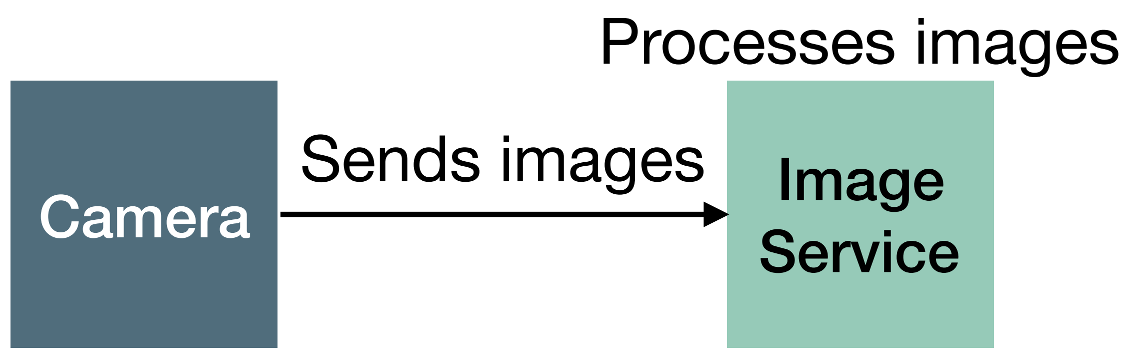


Review: Resource Metric



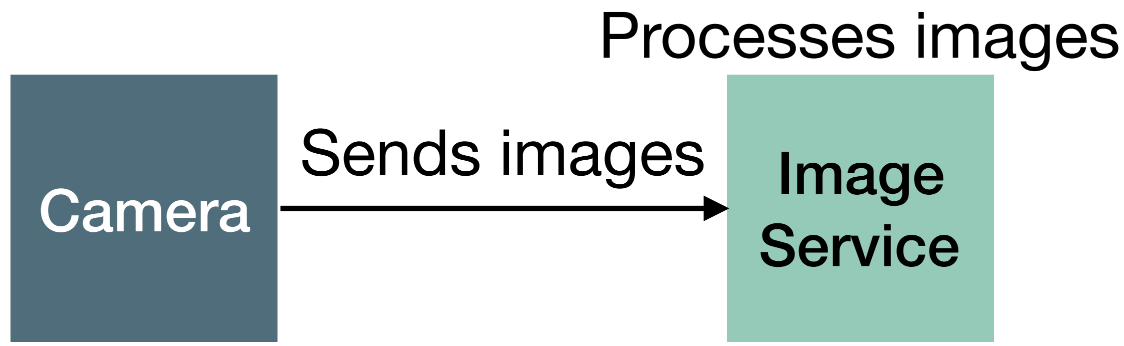
- Say, capacity is measured in terms of processor cycles

Review: Resource Metric



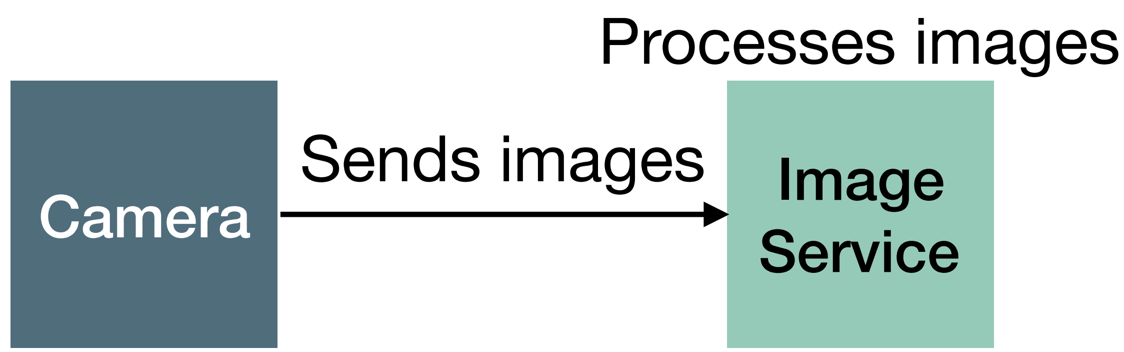
- Say, capacity is measured in terms of processor cycles
- Workload might be processing a single image

Review: Resource Metric



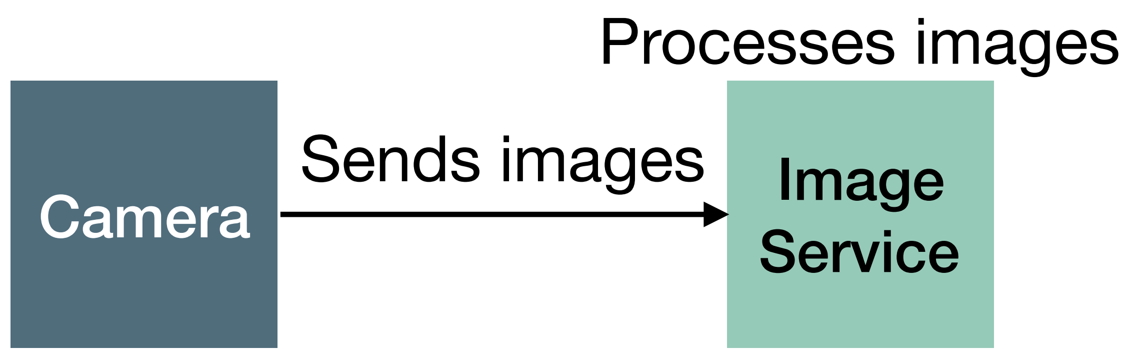
- Say, capacity is measured in terms of processor cycles
- Workload might be processing a single image
- Utilization could 10% -> 90% of processor is unused

Review: Resource Metric



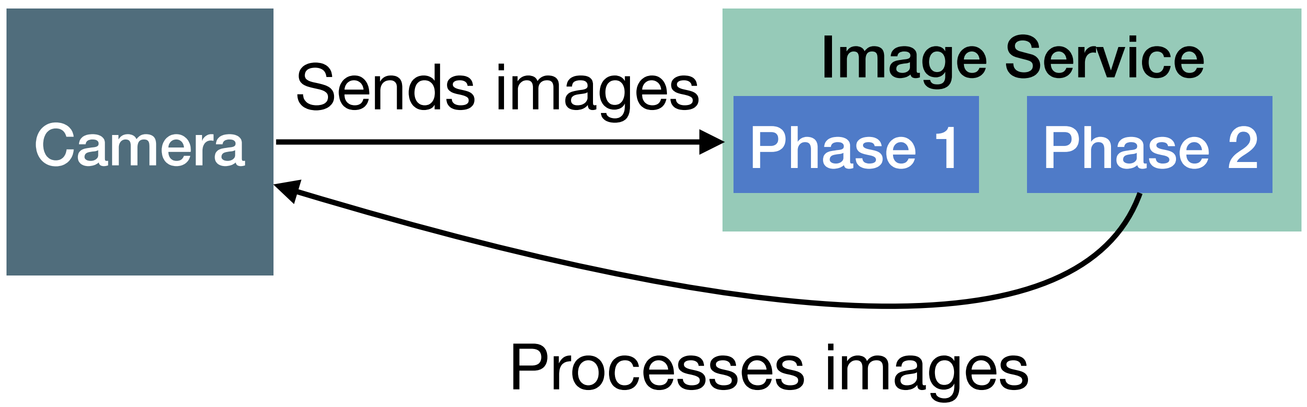
- Say, capacity is measured in terms of processor cycles
- Workload might be processing a single image
- Utilization could 10% -> 90% of processor is unused
- Overhead could go to the OS, perhaps 5% of the CPU going to OS bookkeeping

Review: Resource Metric



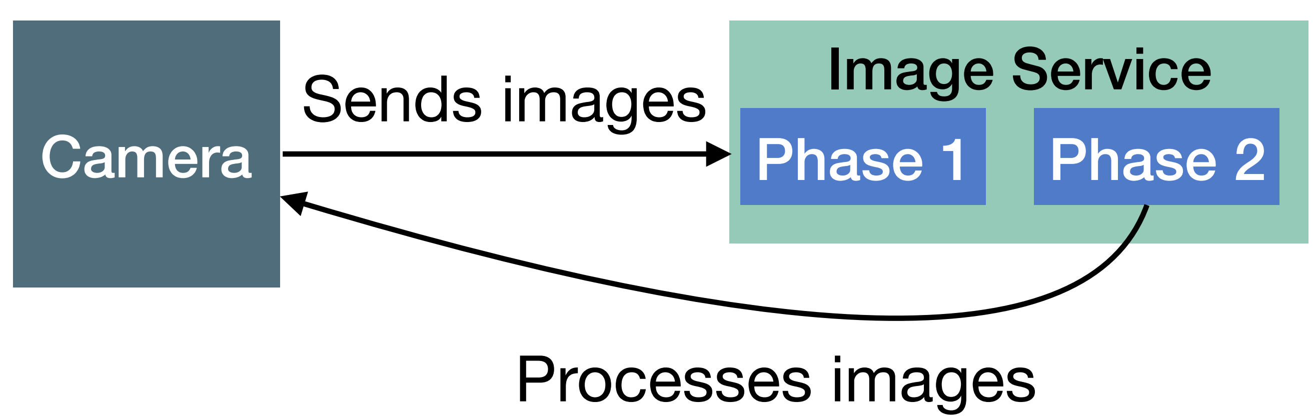
- Say, capacity is measured in terms of processor cycles
- Workload might be processing a single image
- Utilization could 10% -> 90% of processor is unused
- Overhead could go to the OS, perhaps 5% of the CPU going to OS bookkeeping
- Useful work would then be 5%

Review: Latency



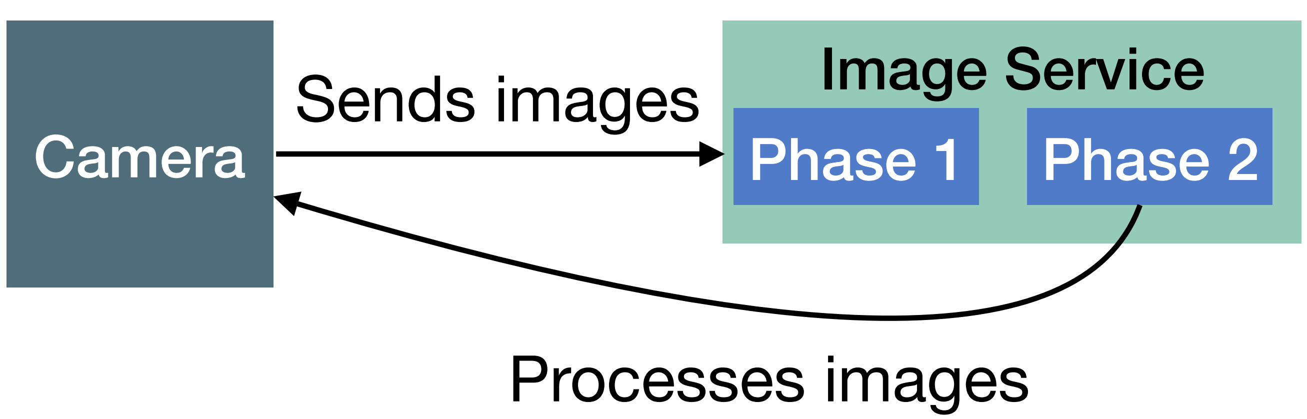
Review: Latency

- In client/server model, latency is simply: time between client sending request and receiving response



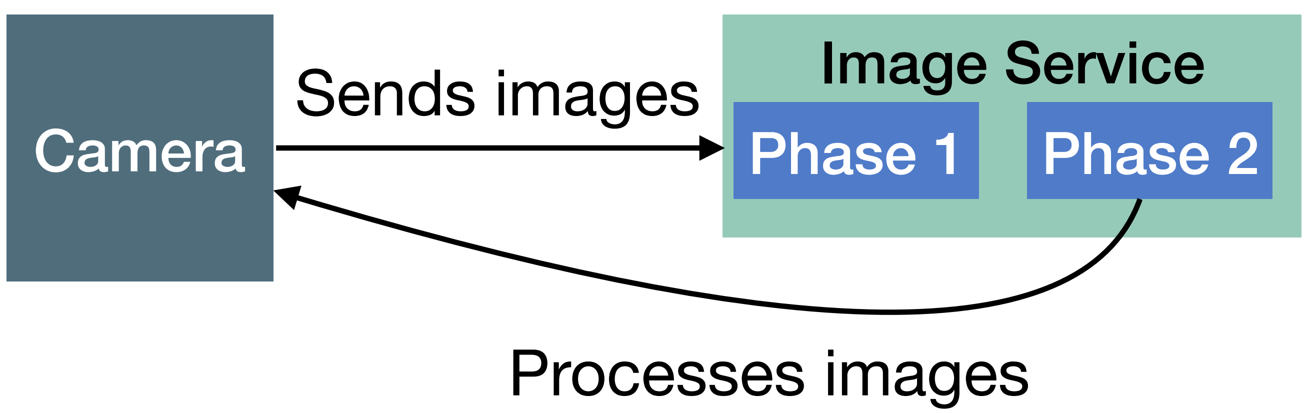
Review: Latency

- In client/server model, latency is simply: time between client sending request and receiving response
- What contributes to latency?



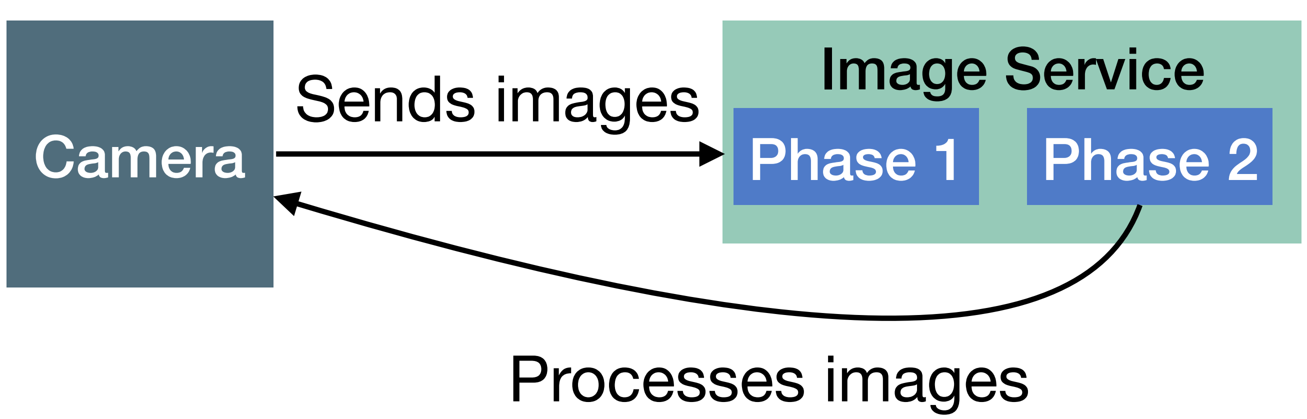
Review: Latency

- In client/server model, latency is simply: time between client sending request and receiving response
- What contributes to latency?
  - Latency sending the message



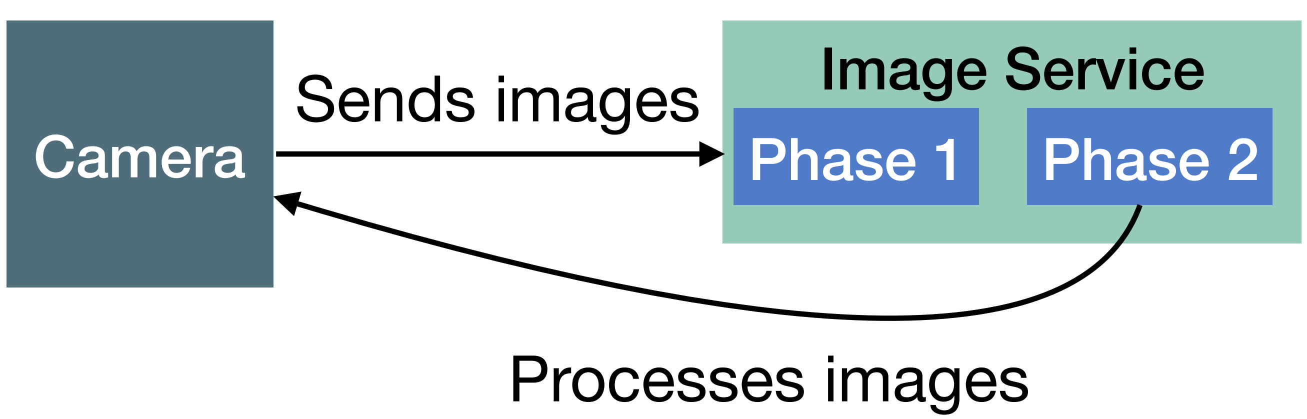
Review: Latency

- In client/server model, latency is simply: time between client sending request and receiving response
- What contributes to latency?
  - Latency sending the message
  - Latency processing the message



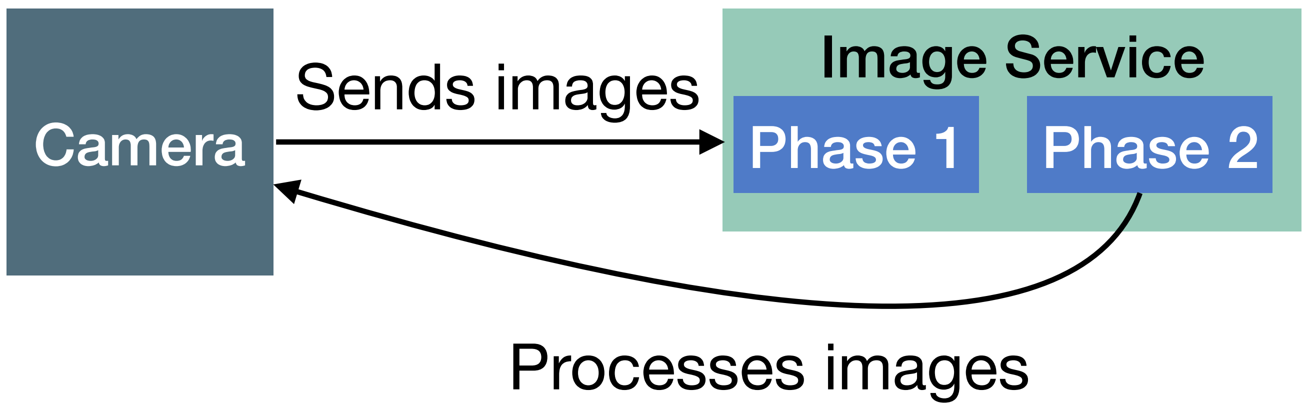
Review: Latency

- In client/server model, latency is simply: time between client sending request and receiving response
- What contributes to latency?
  - Latency sending the message
  - Latency processing the message
  - Latency sending the response



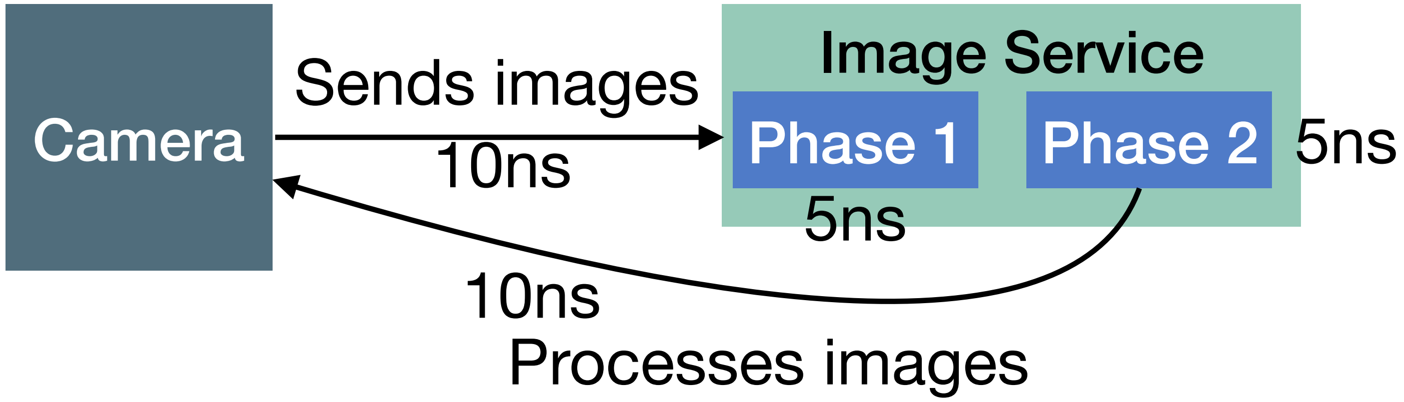
Review: Latency

- In client/server model, latency is simply: time between client sending request and receiving response
- What contributes to latency?
  - Latency sending the message
  - Latency processing the message
  - Latency sending the response
- Adding pipelined components -> latency is cumulative



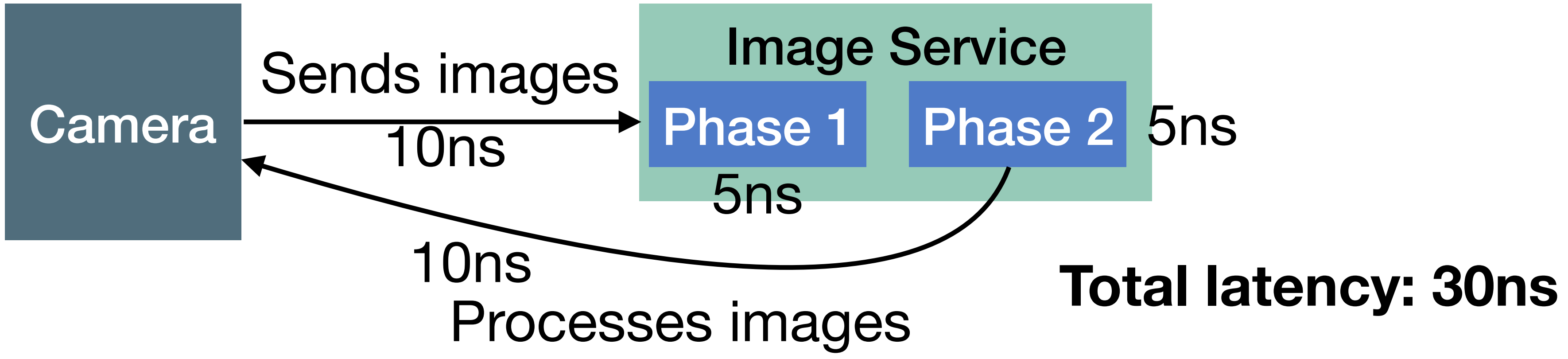
Review: Latency

- In client/server model, latency is simply: time between client sending request and receiving response
- What contributes to latency?
  - Latency sending the message
  - Latency processing the message
  - Latency sending the response
- Adding pipelined components -> latency is cumulative



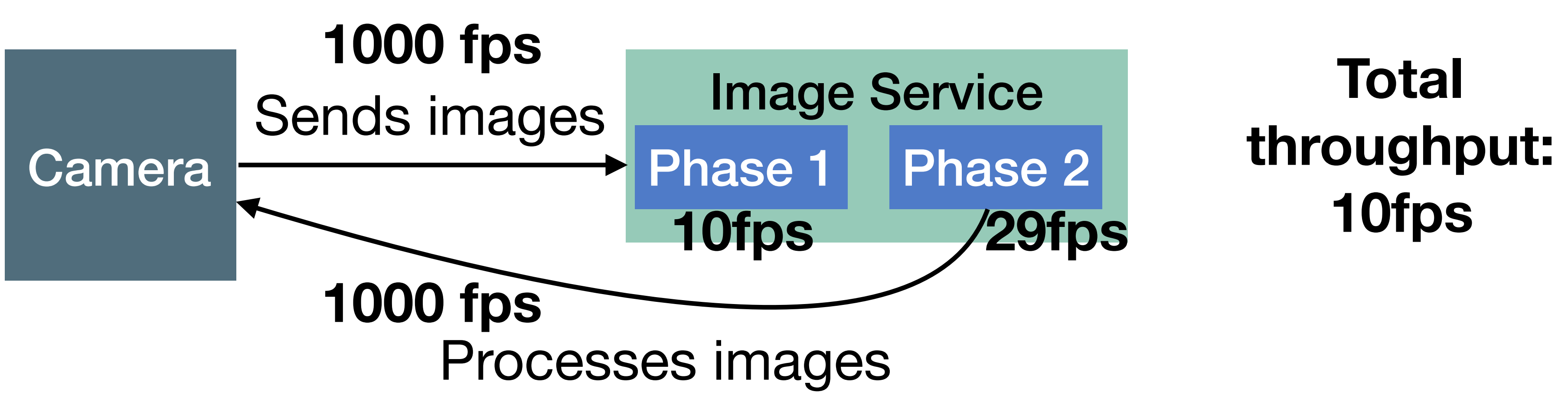
Review: Latency

- In client/server model, latency is simply: time between client sending request and receiving response
- What contributes to latency?
  - Latency sending the message
  - Latency processing the message
  - Latency sending the response
- Adding pipelined components -> latency is cumulative



Review: Throughput

- Measure of the rate of useful work done for a given workload
- Example:
  - Throughput is camera frames processed/second
- When adding multiple pipelined components -> throughput is the minimum value



Review: Designing for Performance

- Measure system to find which aspect of performance is lacking (throughput or latency)
- Measure each component to identify bottleneck
- Identify if fixing that bottleneck will realistically improve system performance
- Measure improvement
- Repeat

Review: Streams

- Java 8 introduced the concept of **Streams**
- A stream is a sequence of objects
- Streams have functions that you can perform on them, which are (mostly) **non-interfering** and **stateless**
  - Non-interfering: Does not modify the actual stream
  - Stateless: Each time the function is called on the same data, get same result
- Example:

```
IntStream.range(1, 1000000) //Generate a stream of all ints 1 - 1m
.filter(x -> isPrime(x)) //Retain only values that pass some expensive isPrime function
.forEach(System.out::println); //For each value returned by filter, print it
```

Review: Streams

Review: Streams

```
.forEach(System.out::println); //For each value returned by filter, print
it
```

Review: Streams

- Why use the stream interface instead of

```
System.out.println(x);
```

Review: Streams

- Why use the stream interface instead of

- Who wants to write the parallel version of this?

```
.forEach(System.out::println); //For each value returned by filter, print
it
```

Review: Streams

- Why use the stream interface instead of
- Who wants to write the parallel version of this?
- The magic works as long as isPrime is stateless!

Announcements

- Reminder: HW2 is out
  - http://www.jonbell.net/gmu-cs-475-spring-2018/homework-2/
- Today: Networking Basics
- Reading: Saltzer Ch 7.1, 7.2

Streams under the hood

- Just adding more parallel() doesn't always make it faster! (see: law of leaky abstractions)
- There is some overhead to how a parallel operation occurs
- Internally, Java keeps a pool of **worker threads** (rather than make new threads for each parallel task)
- Streams use a special kind of pool, called a **ForkJoinPool**

Fork/Join Programming

- **Special kind of task - fork() defines how to create subtasks, join() defines how to combine the results**
- **Similar to map/reduce, but not distributed**
- **For streams:**
  - **Fork** a task into subtasks for many threads to work on
  - **Join** the results together

Fork/Join Programming

- Obligatory array sum example

```
class Sum extends RecursiveTask<Long> {
    static final int SEQUENTIAL_THRESHOLD = 5000;

    int low;
    int high;
    int[] array;

    Sum(int[] arr, int lo, int hi) {
        array = arr;
        low   = lo;
        high  = hi;
    }

    protected Long compute() {
        if(high - low <= SEQUENTIAL_THRESHOLD) {
            long sum = 0;
            for(int i=low; i < high; ++i)
                sum += array[i];
            return sum;
        } else {
            int mid = low + (high - low) / 2;
            Sum left  = new Sum(array, low, mid);
            Sum right = new Sum(array, mid, high);
            left.fork();
            long rightAns = right.compute();
            long leftAns  = left.join();
            return leftAns + rightAns;
        }
    }

    static long sumArray(int[] array) {
        return ForkJoinPool.commonPool().invoke(new Sum(array,0,array.length));
    }
}
```

(Fork/Join Demo)

Promise

- What if we want to run some task, and do stuff while we are waiting for it to be done?
- You COULD do it with a complicated combination of **synchronized**, **wait**, and **notify**
- You can use the **Promise** abstraction instead
  - Called a **CompletableFuture** in Java 8

```
CompletableFuture<String> future = CompletableFuture.supplyAsync(() -> {
    try {
        TimeUnit.SECONDS.sleep(1);
    } catch (InterruptedException e) {
        throw new IllegalStateException(e);
    }
    return "Result of the asynchronous computation";
});
// Block and get the result of the Future
String result = future.get();
System.out.println(result);
```

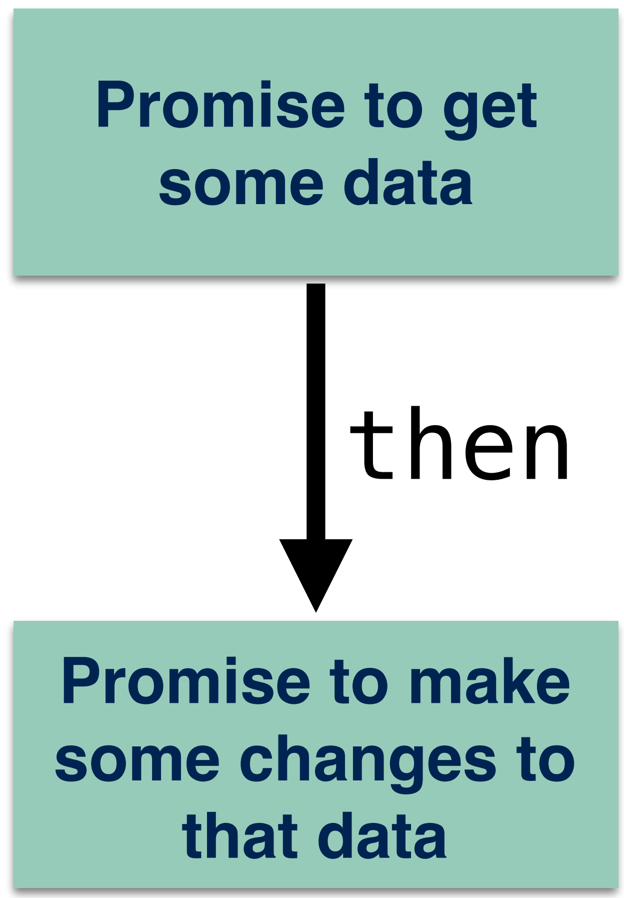
Promise Use-Cases

- Any case where you need to have multiple things happen in the background, but care about the result, and care about them happening in some order
- Asynchronous I/O
  - Read data from a web service
  - Then process it
  - Then save it to a file

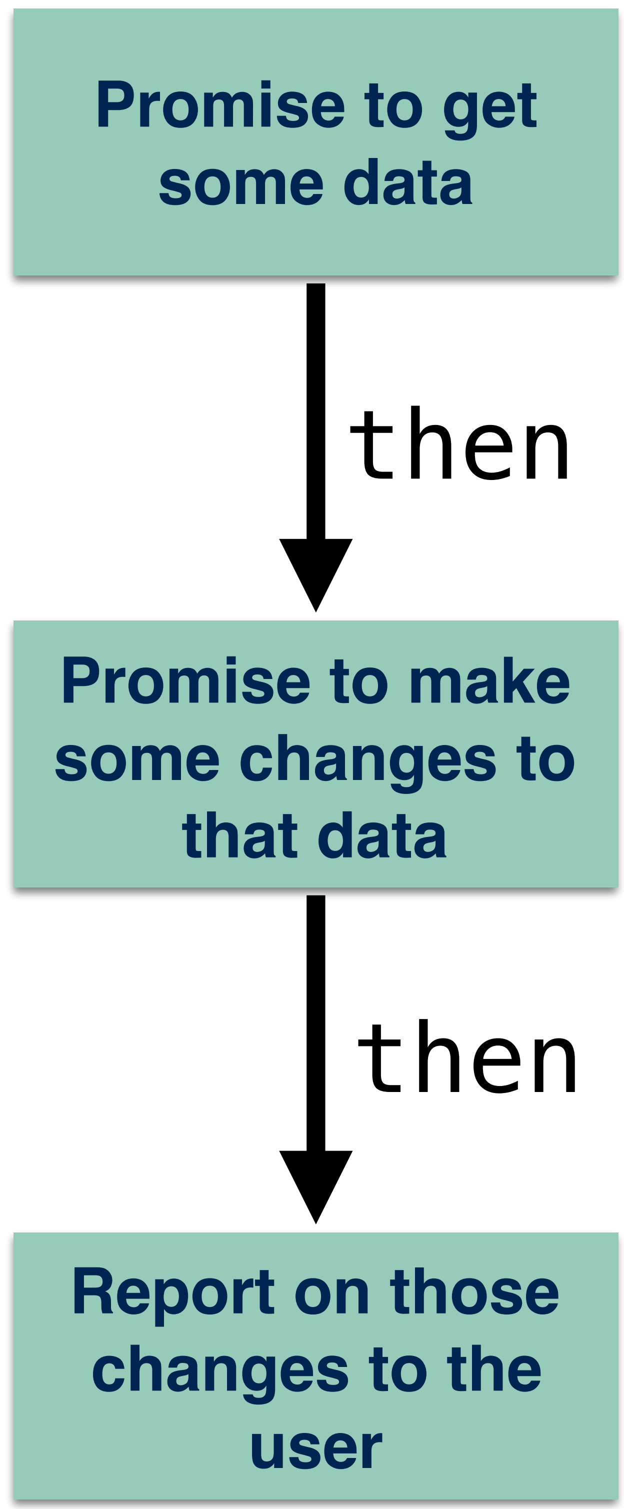
Chaining Promises

Promise to get some data

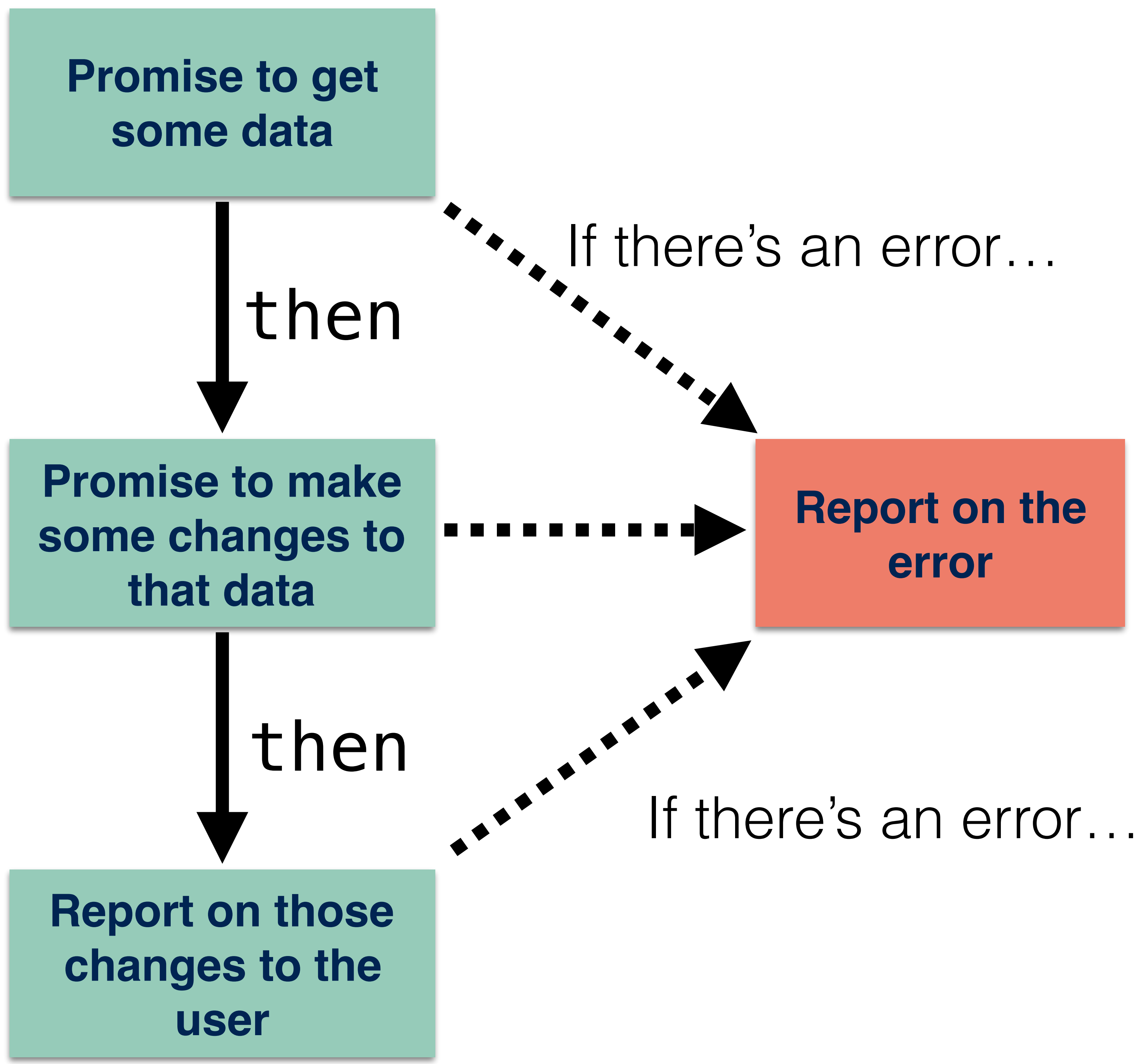
Chaining Promises



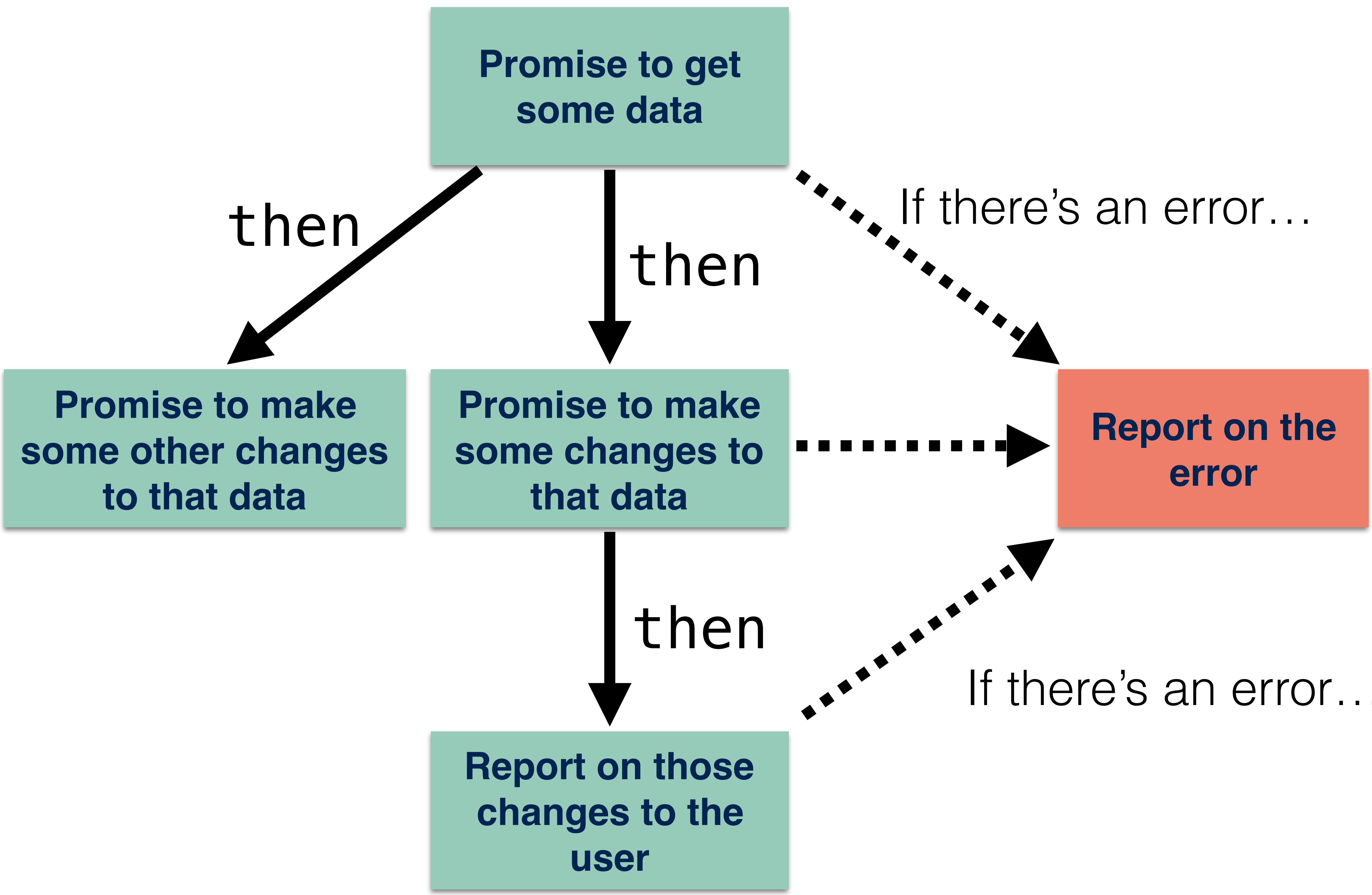
Chaining Promises



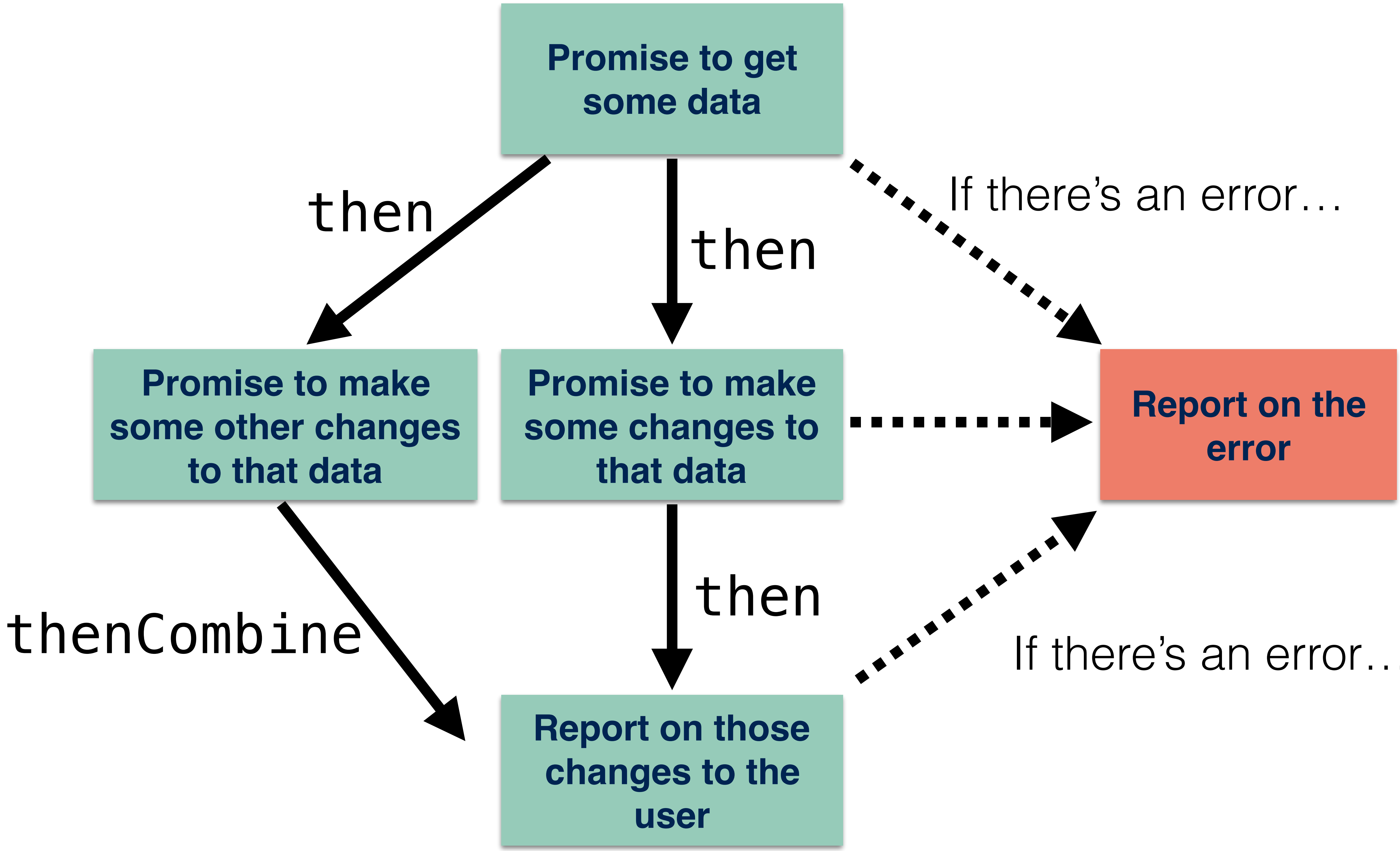
Chaining Promises



Chaining Promises



Chaining Promises



Promises

- Catch errors by providing a callback function for **exceptionally** (called when an exception occurs in any of those threads
- API: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html

(Promises Demo)

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

Networks as Abstractions

- Stuff goes in, stuff goes out?

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)

Networks as Abstractions

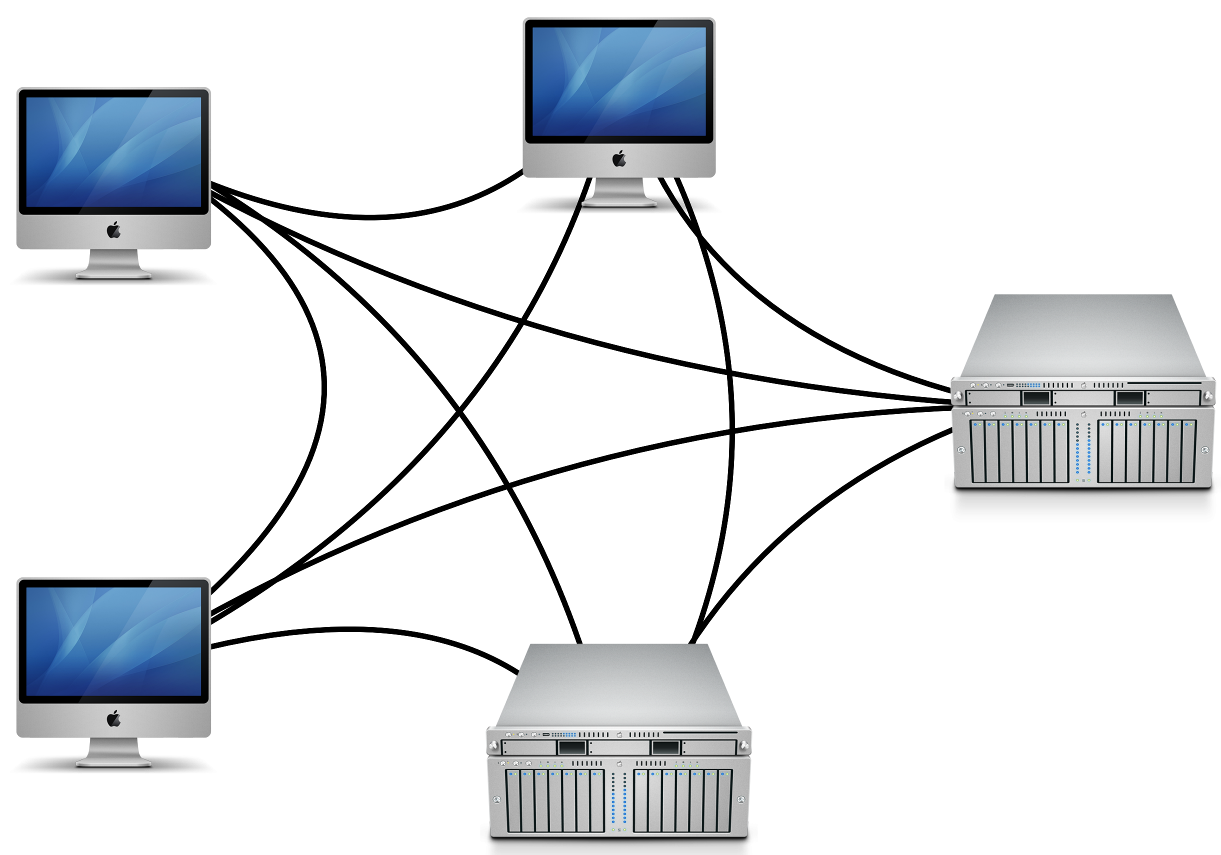
- Stuff goes in, stuff goes out?
- Not a perfect abstraction, because:
  - Speed of light (1 foot/nanosecond)
  - Communication links exist in uncontrolled/hostile environments
  - Communication links may be bandwidth limited (tough to reach even 100MB/sec)
- In contrast to a single computer, where:
  - 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

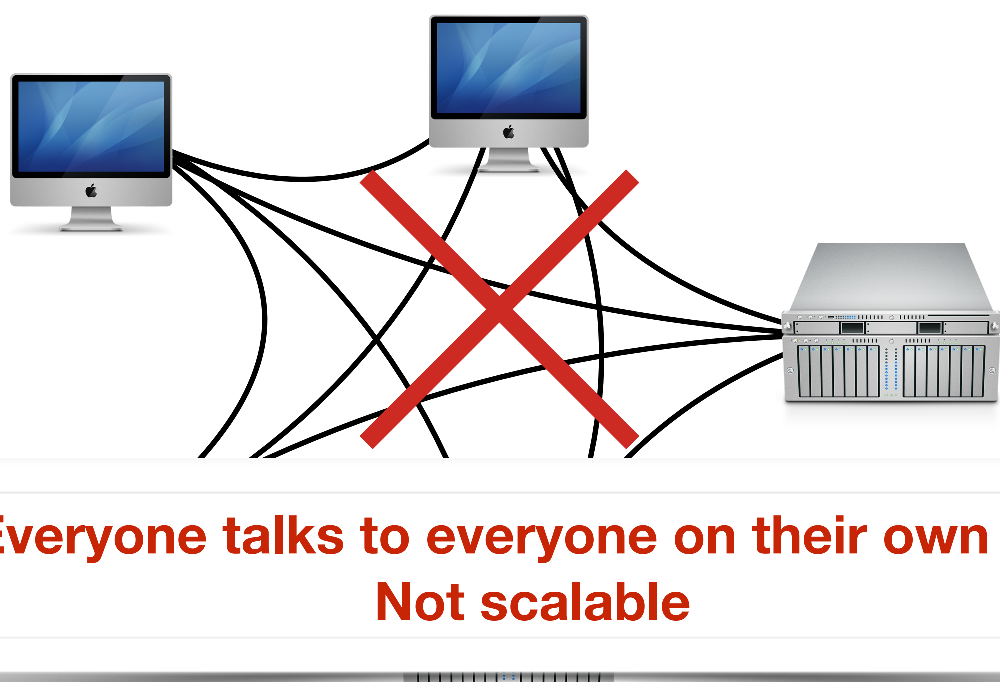
Networks are Shared

- 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



Networks are Shared

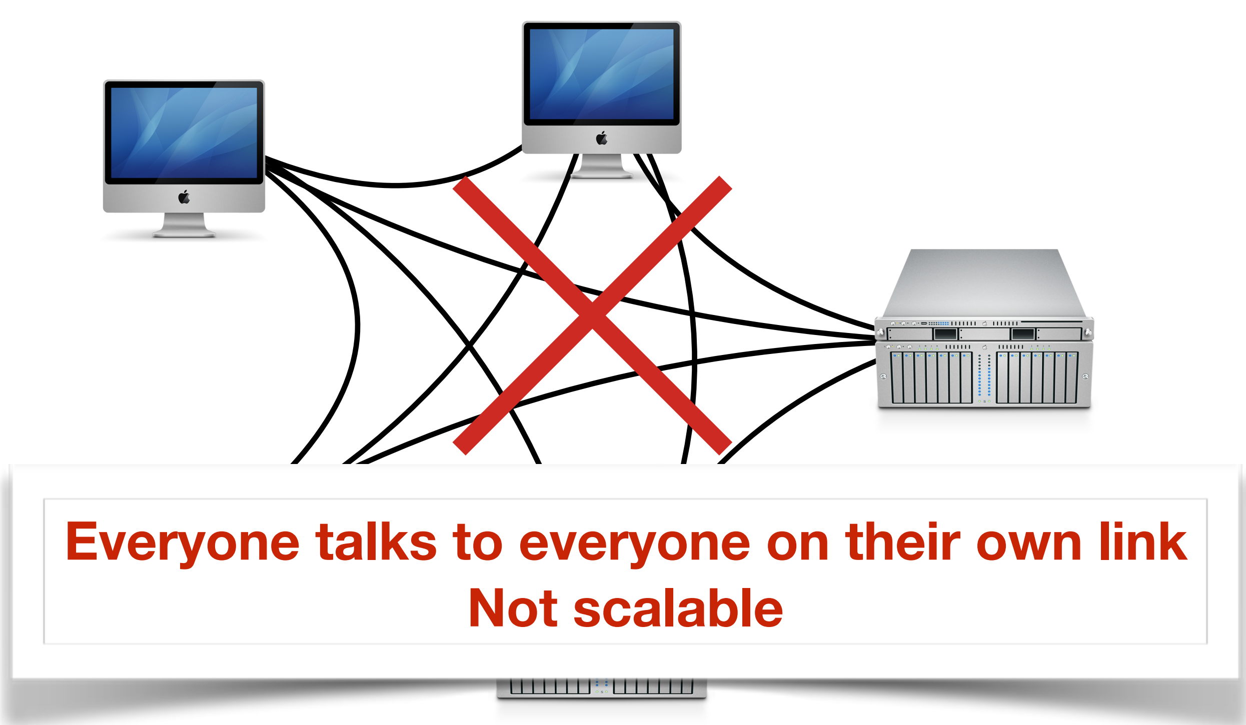
- With processes, we considered how one CPU could be shared between multiple programs running at once
- With networks, communication links are probably shared even more widely

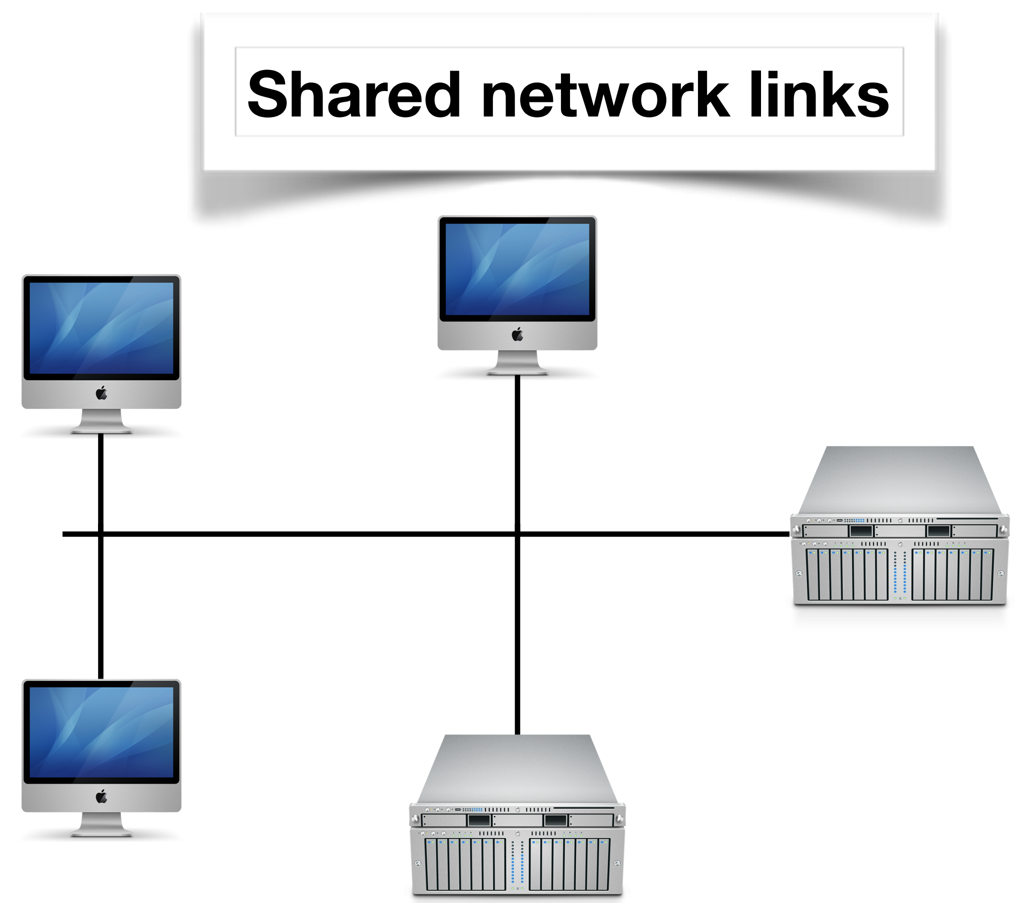


Everyone talks to everyone on their own link
Not scalable

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

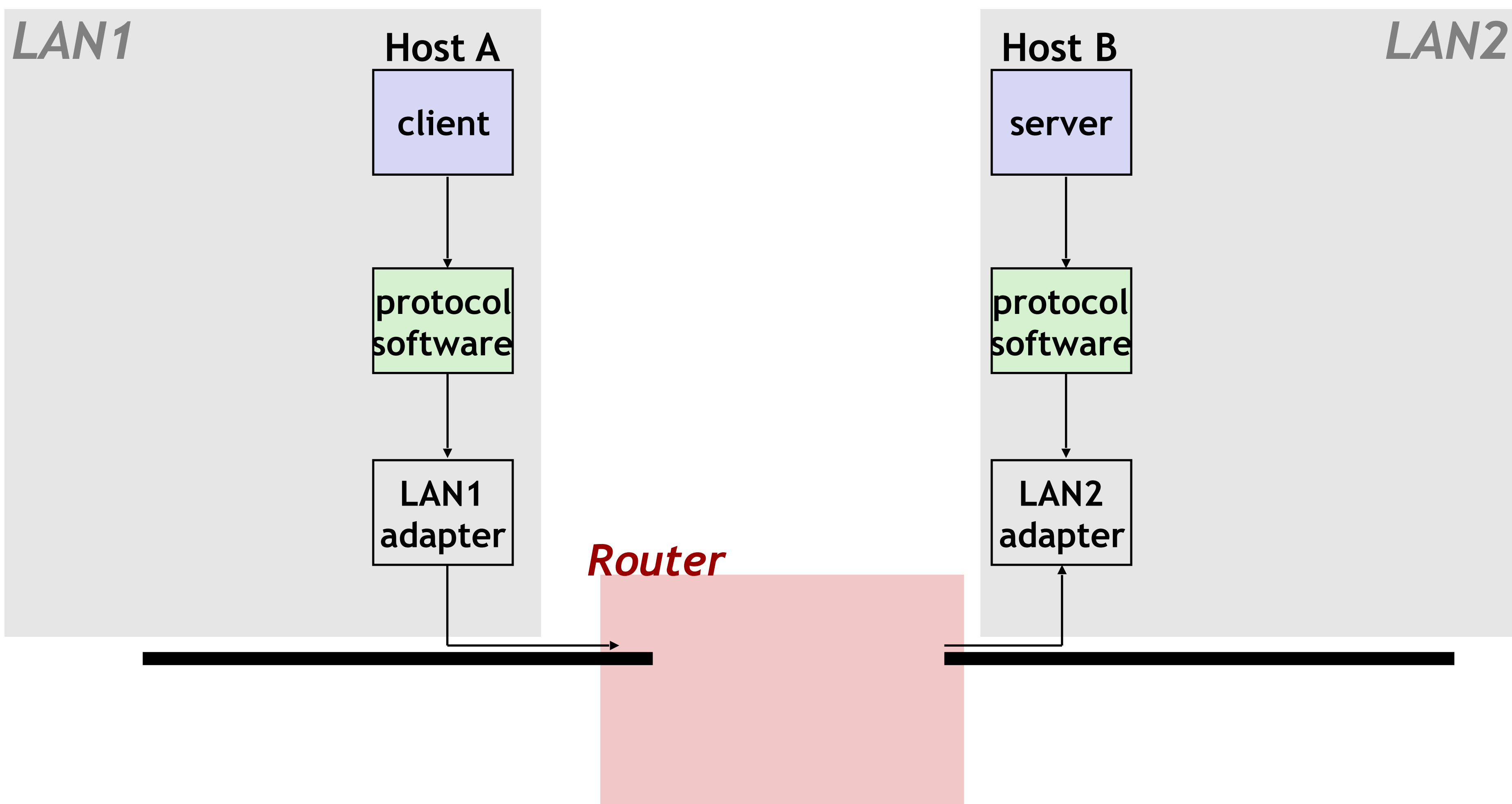




Network as Abstractions

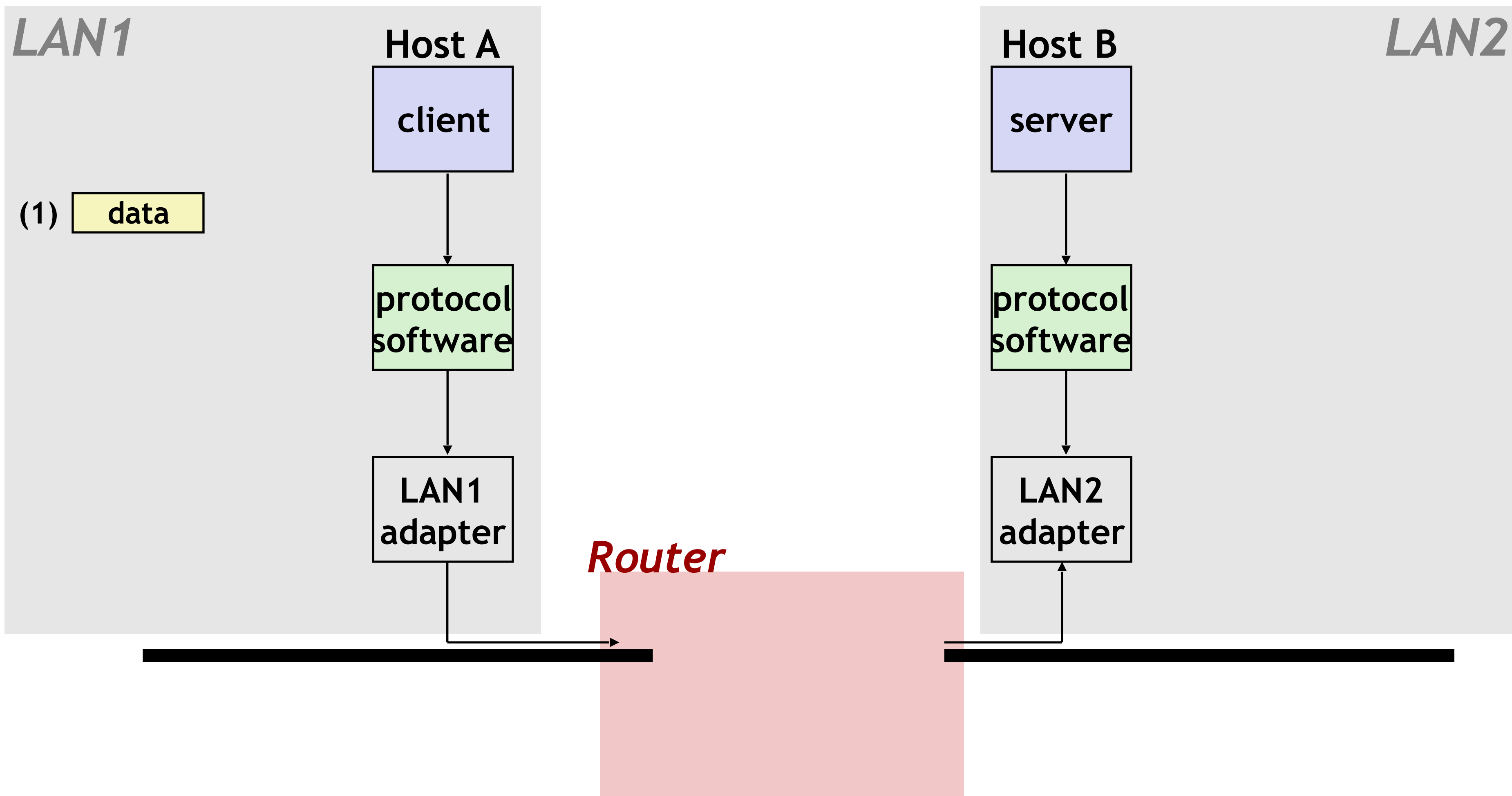
- What do we send, what gets received?
- At the lowest level, we call what gets sent **frames**
- Each frame is limited in size
    - Ethernet: max 1522 **bytes**
- Frame is packed with source/destination info into a **packet**
- Network knows what to do with packets to get them to their destination

Networks as Abstractions



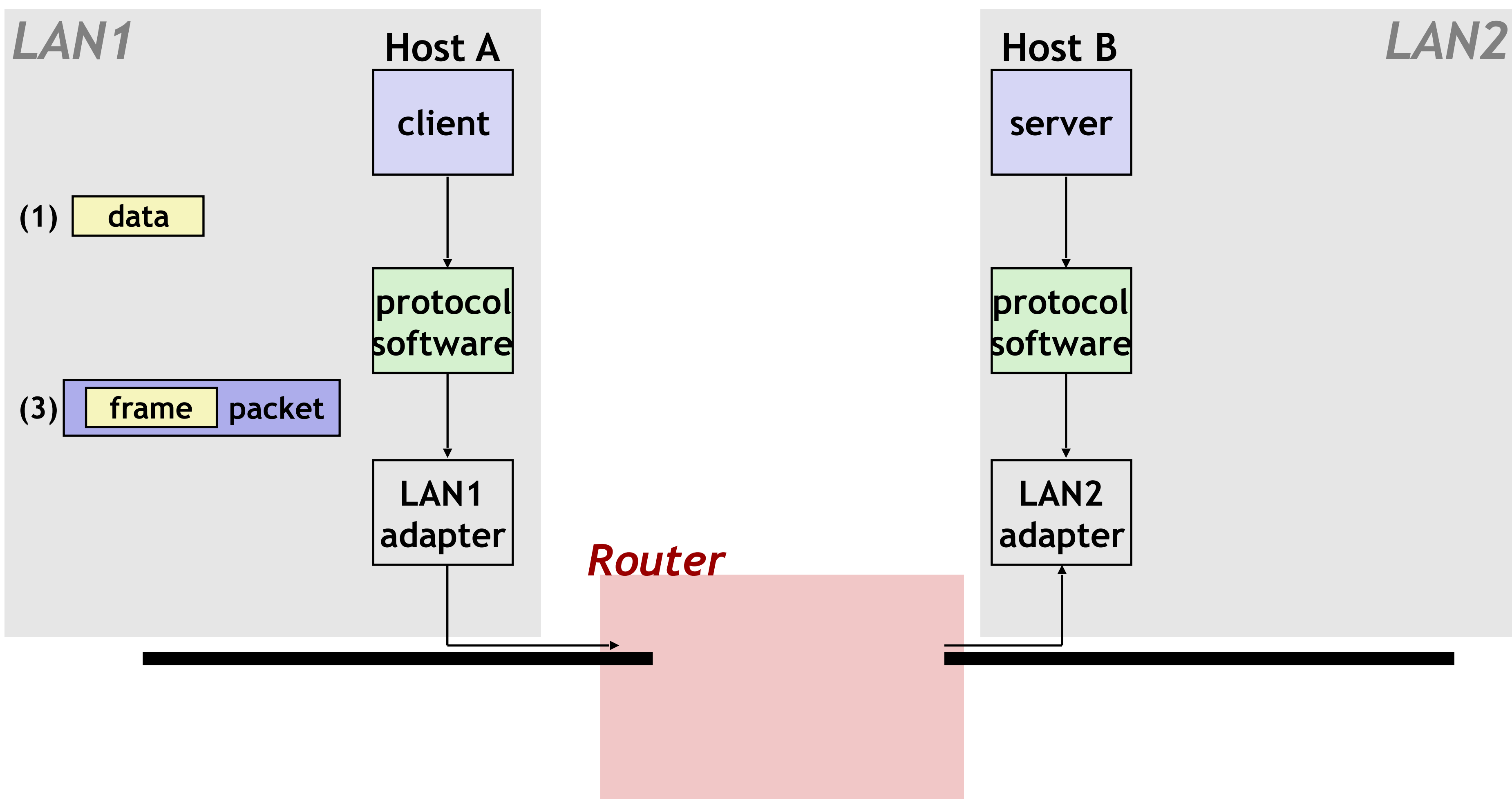
PH: Internet packet header
FH: LAN frame header

Networks as Abstractions



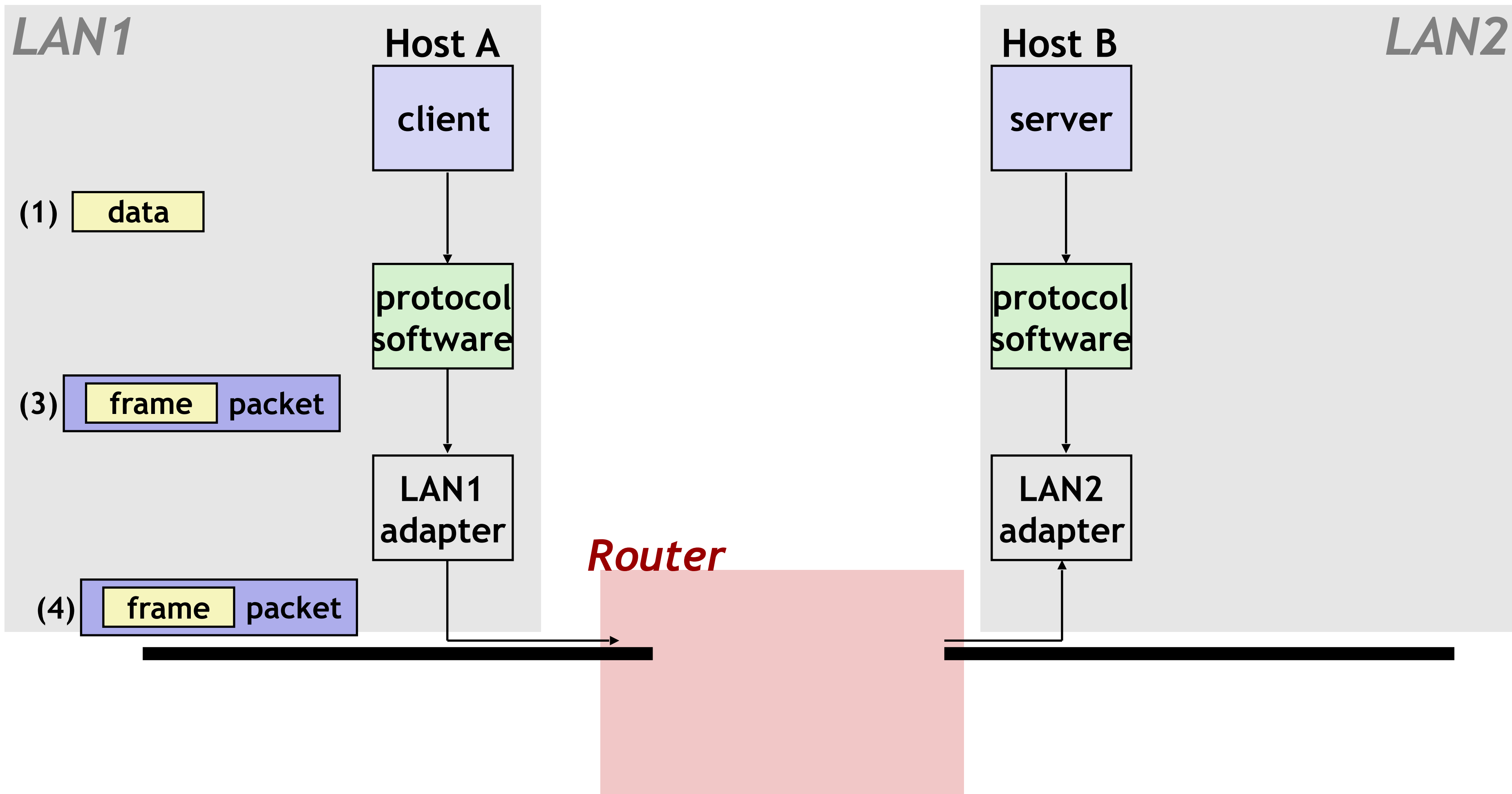
PH: Internet packet header
FH: LAN frame header

Networks as Abstractions



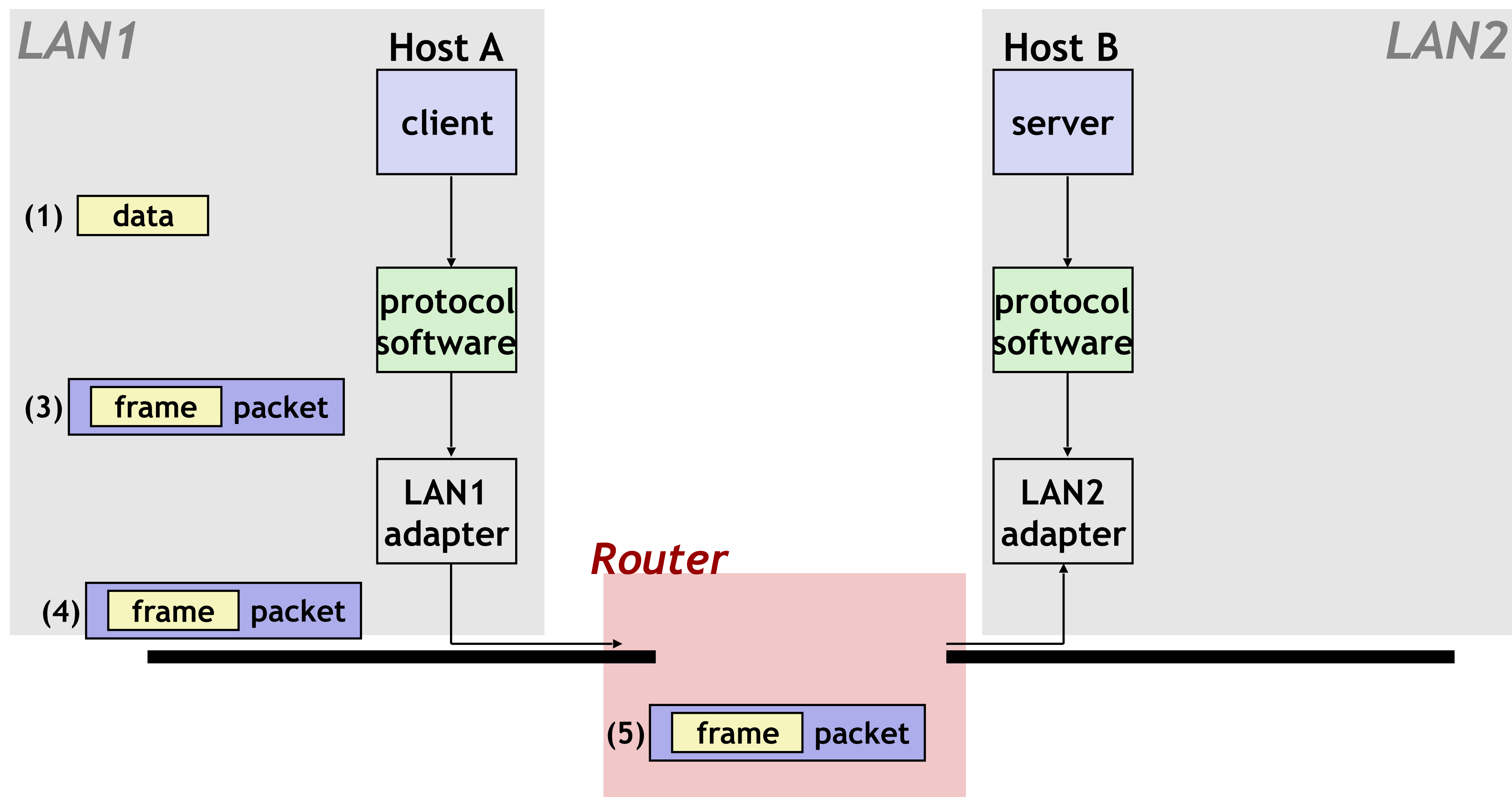
PH: Internet packet header
FH: LAN frame header

Networks as Abstractions



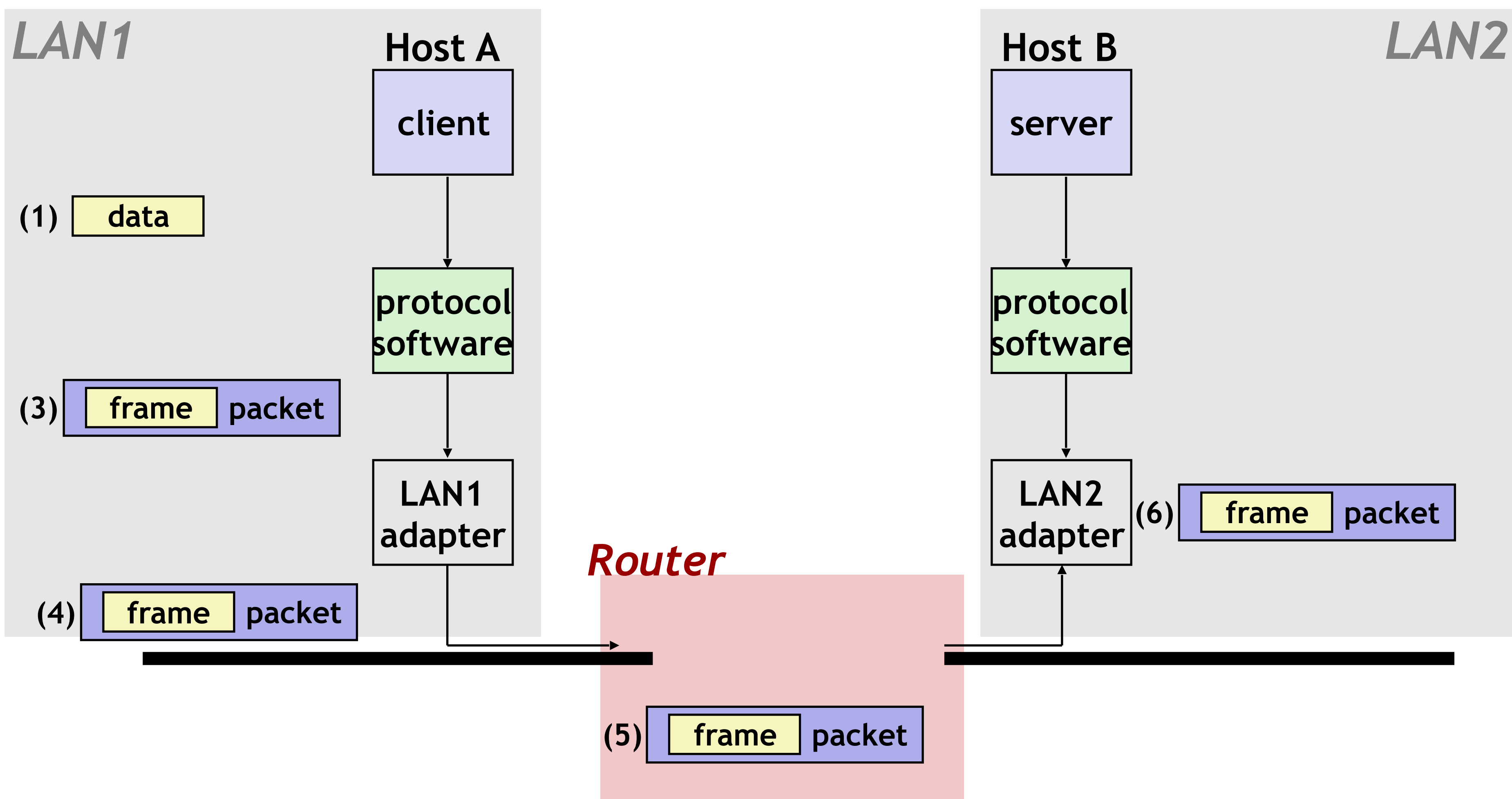
PH: Internet packet header
FH: LAN frame header

Networks as Abstractions



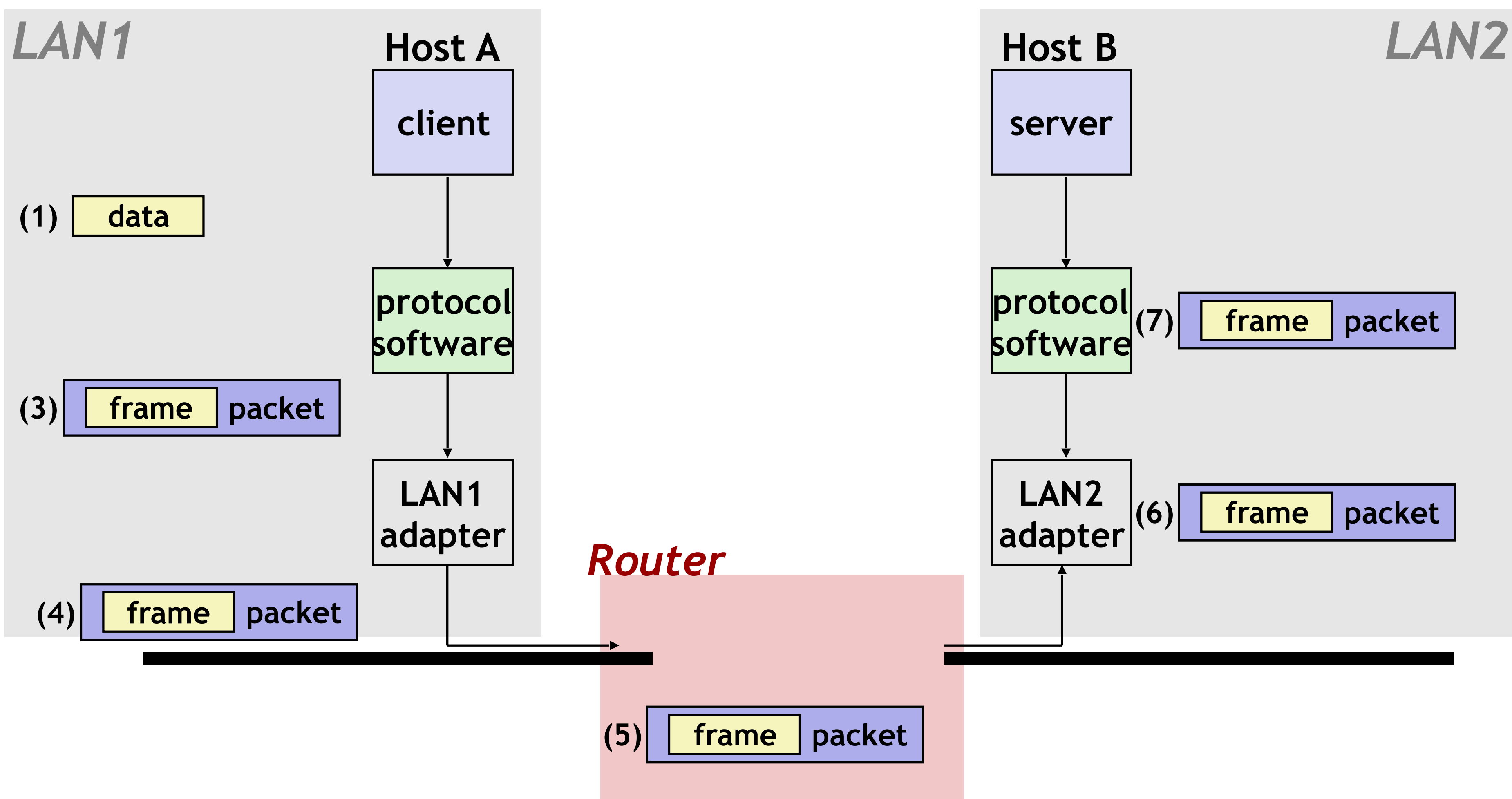
PH: Internet packet header
FH: LAN frame header

Networks as Abstractions



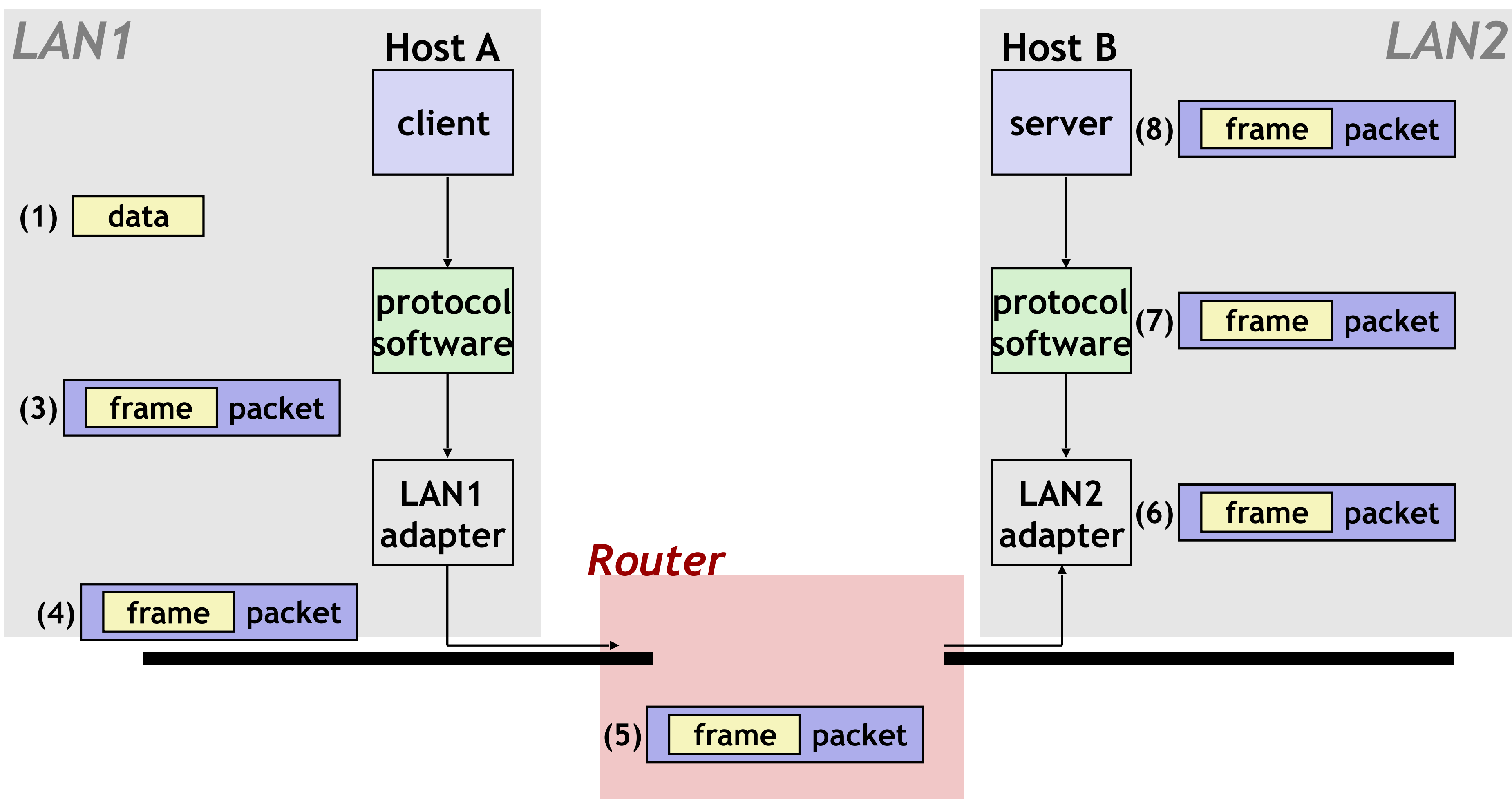
PH: Internet packet header
FH: LAN frame header

Networks as Abstractions



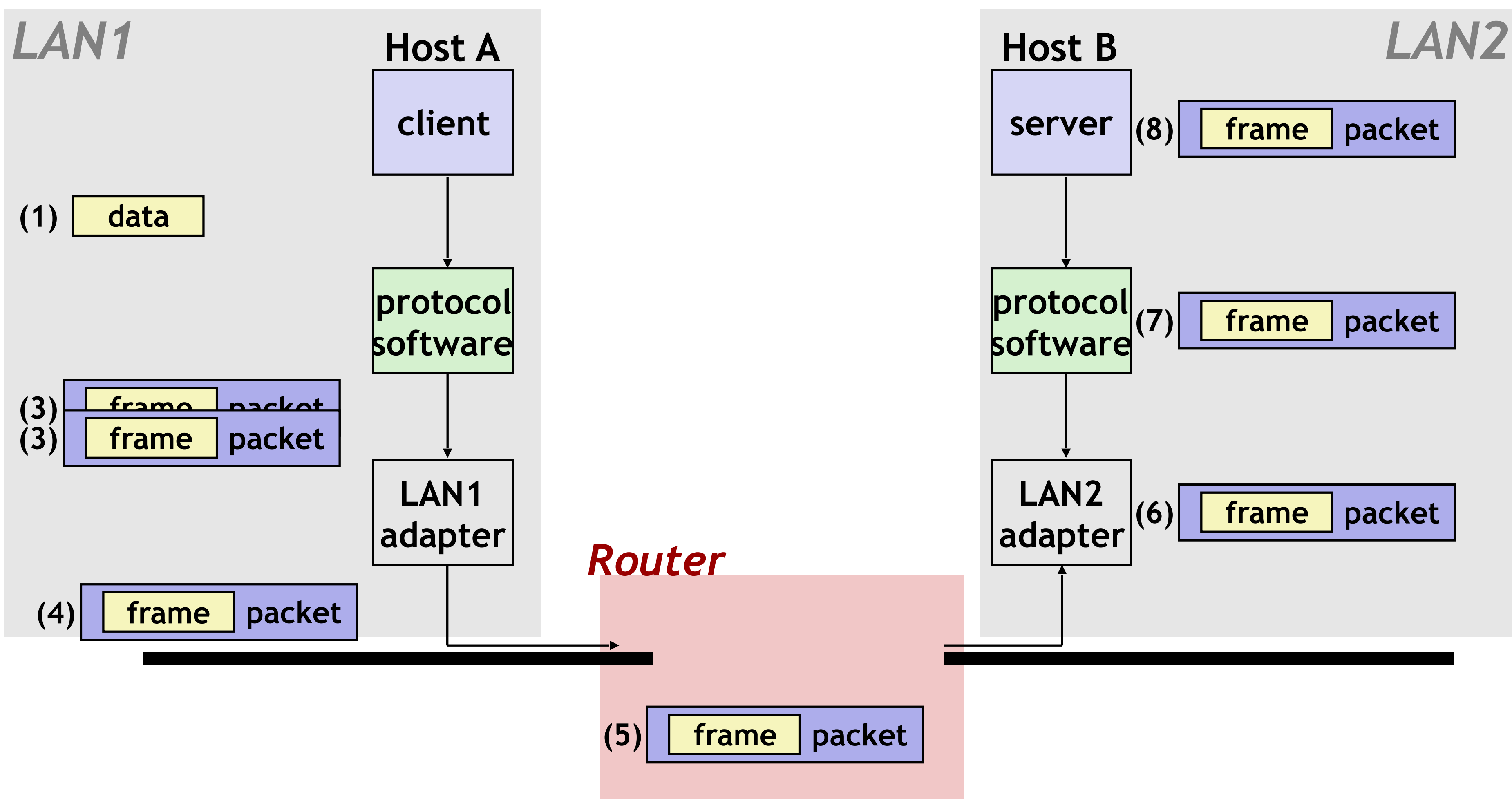
PH: Internet packet header
FH: LAN frame header

Networks as Abstractions



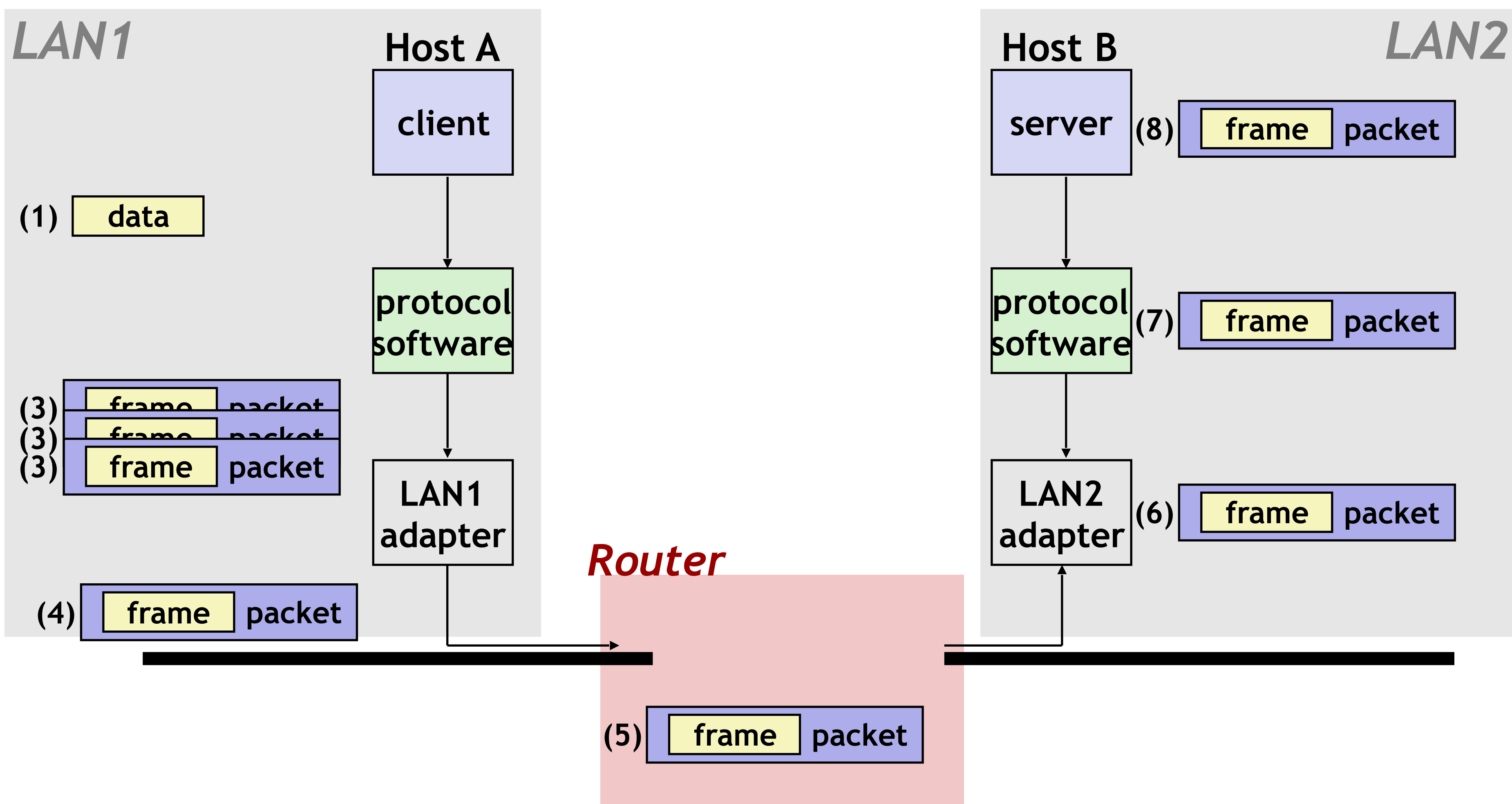
PH: Internet packet header
FH: LAN frame header

Networks as Abstractions



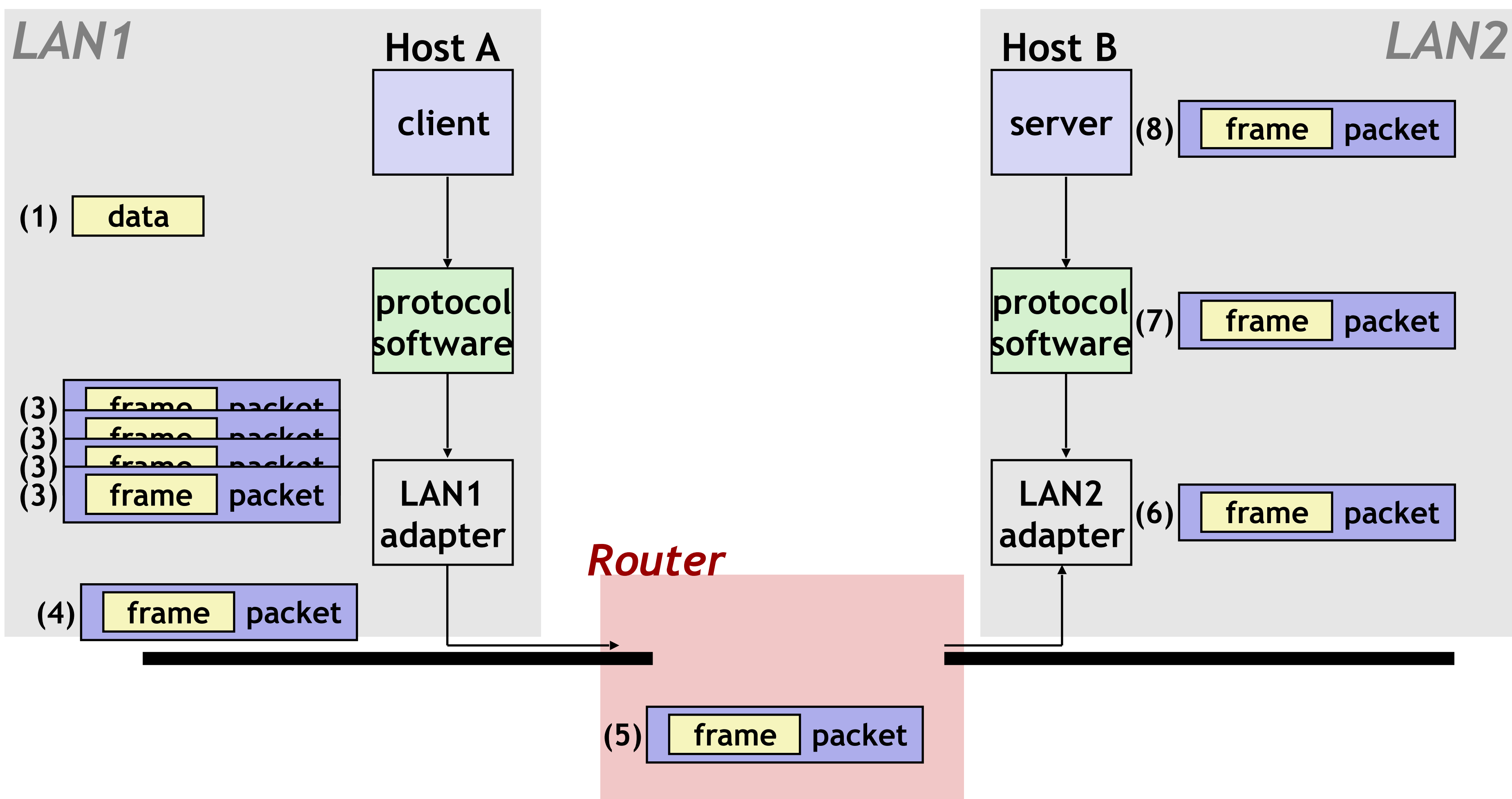
PH: Internet packet header
FH: LAN frame header

Networks as Abstractions



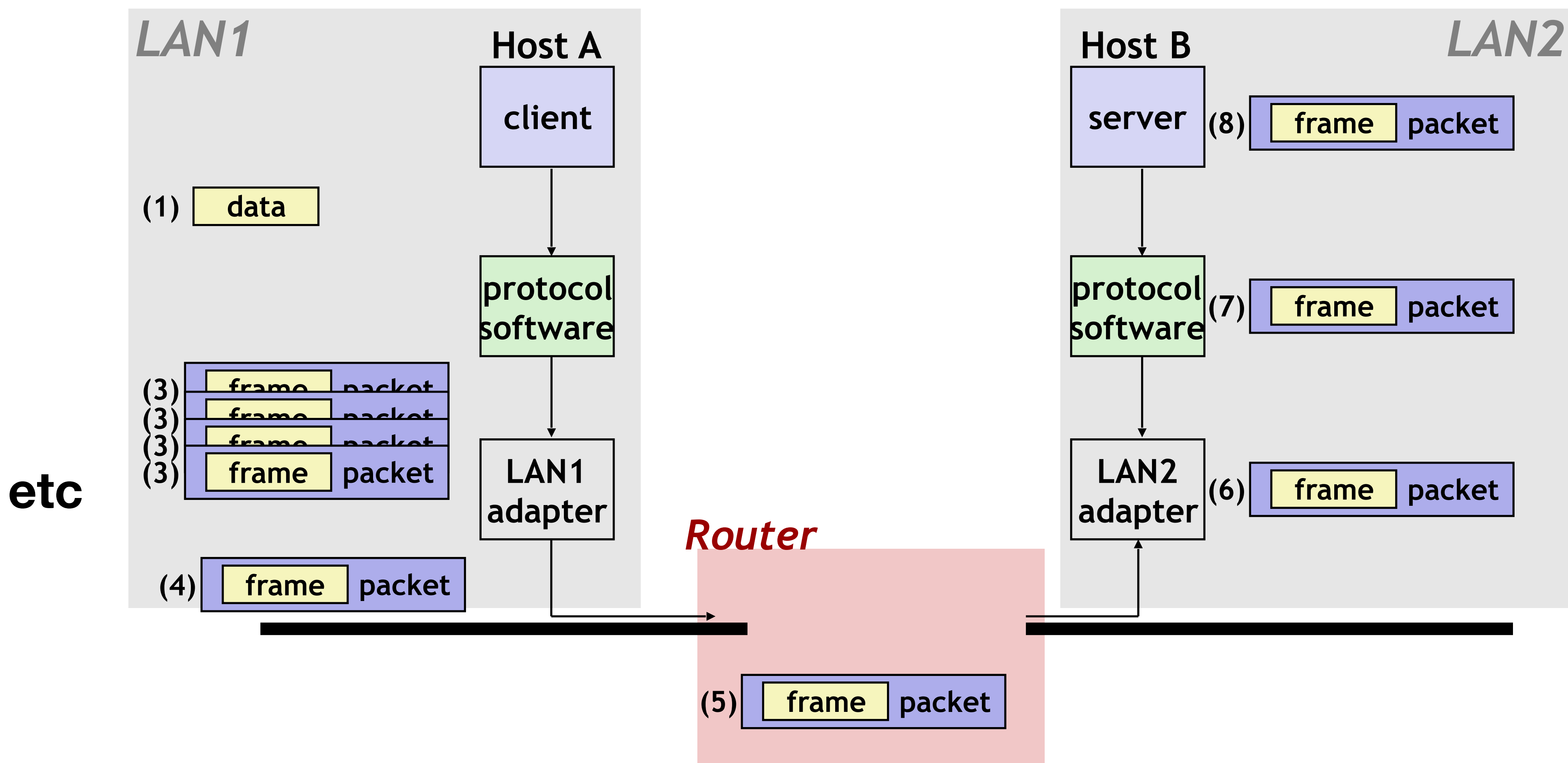
PH: Internet packet header
FH: LAN frame header

Networks as Abstractions



PH: Internet packet header
FH: LAN frame header

Networks as Abstractions



PH: Internet packet header
FH: LAN frame header

Packet Switching Delays

Packet Switching Delays

- As these packets flow through a network and are routed, we might see delays due to:

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)

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)

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)

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

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

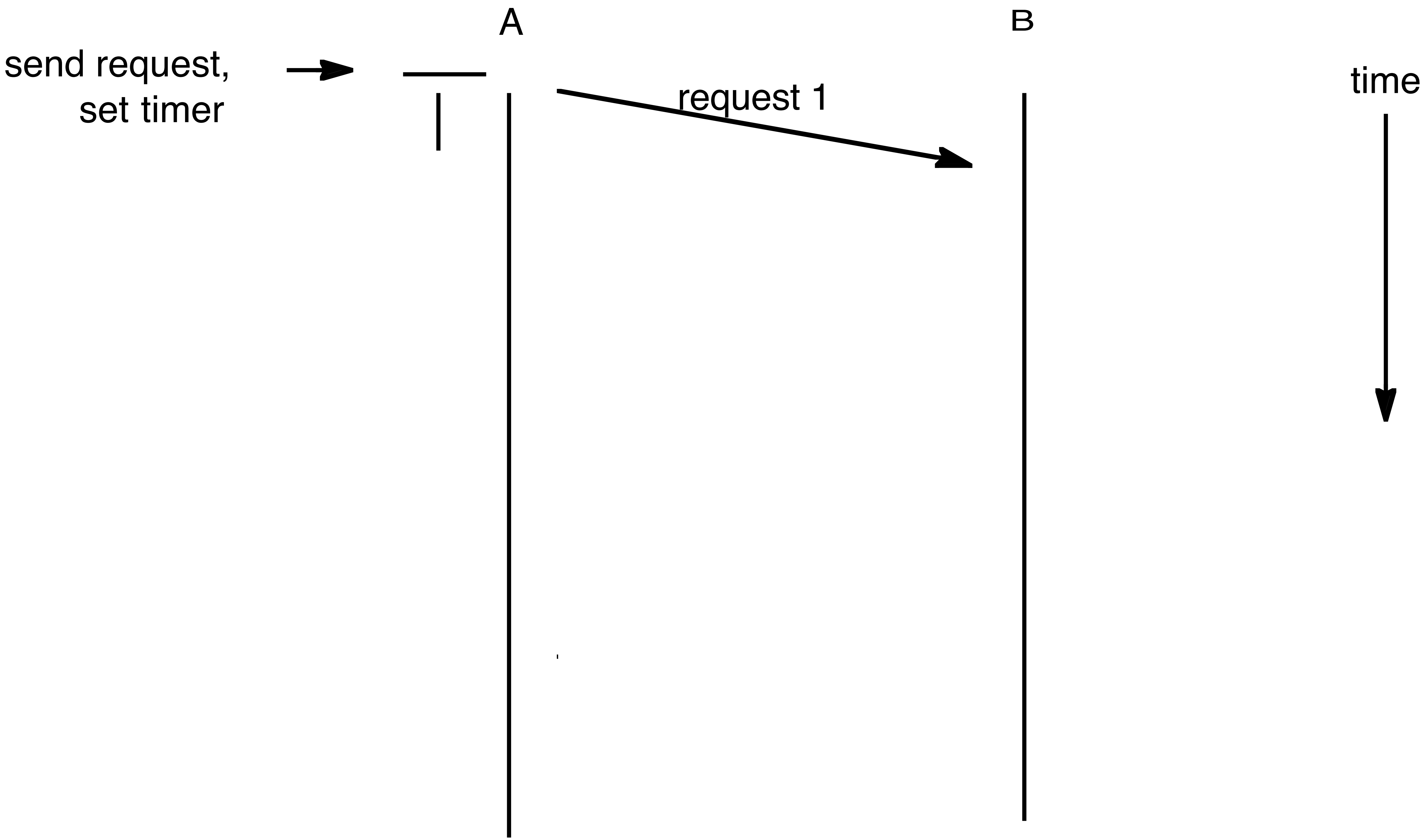


Fig © Saltzer & Kaashoek

Resending Packets

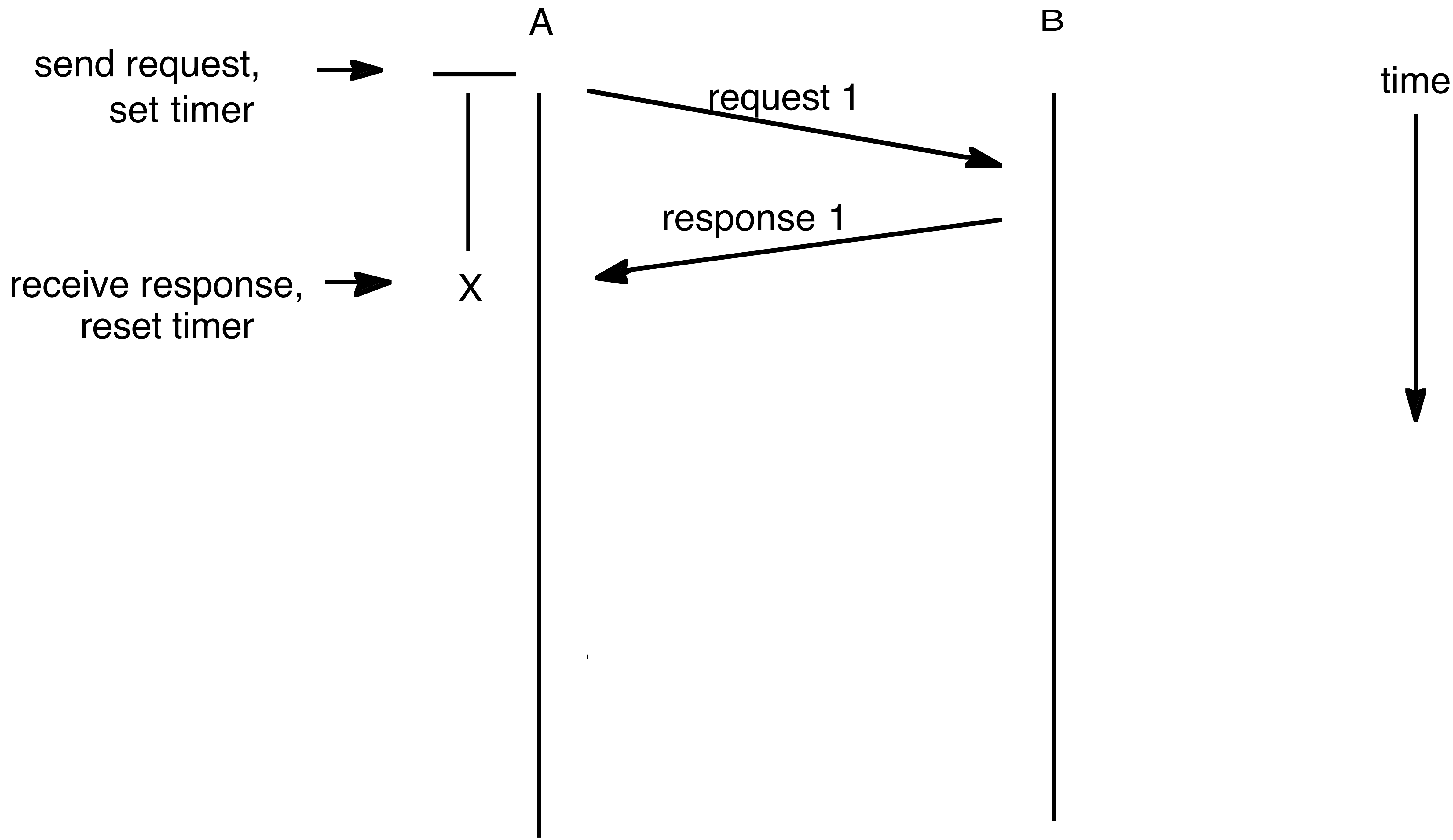


Fig © Saltzer & Kaashoek

Resending Packets

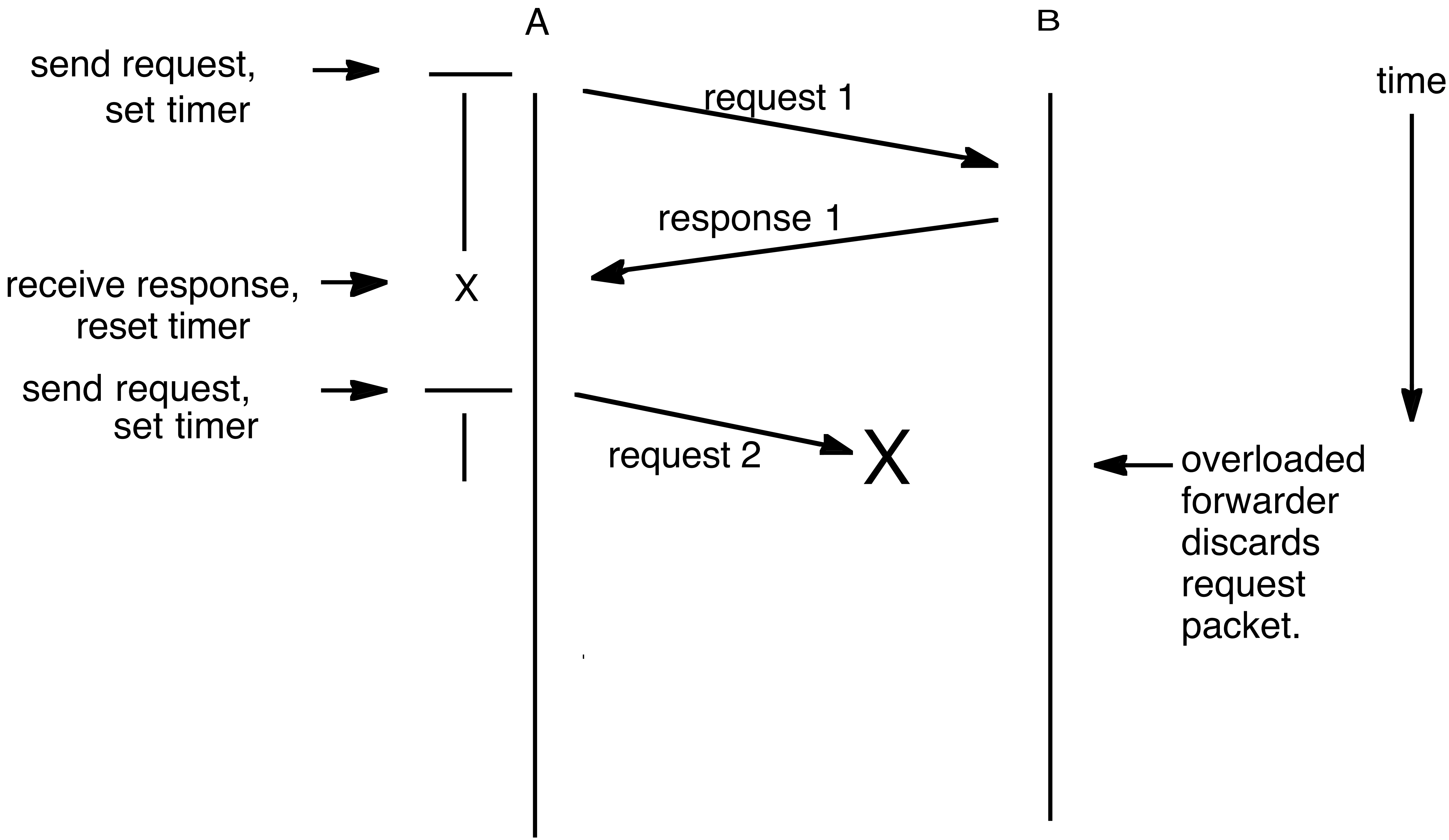


Fig © Saltzer & Kaashoek

Resending Packets

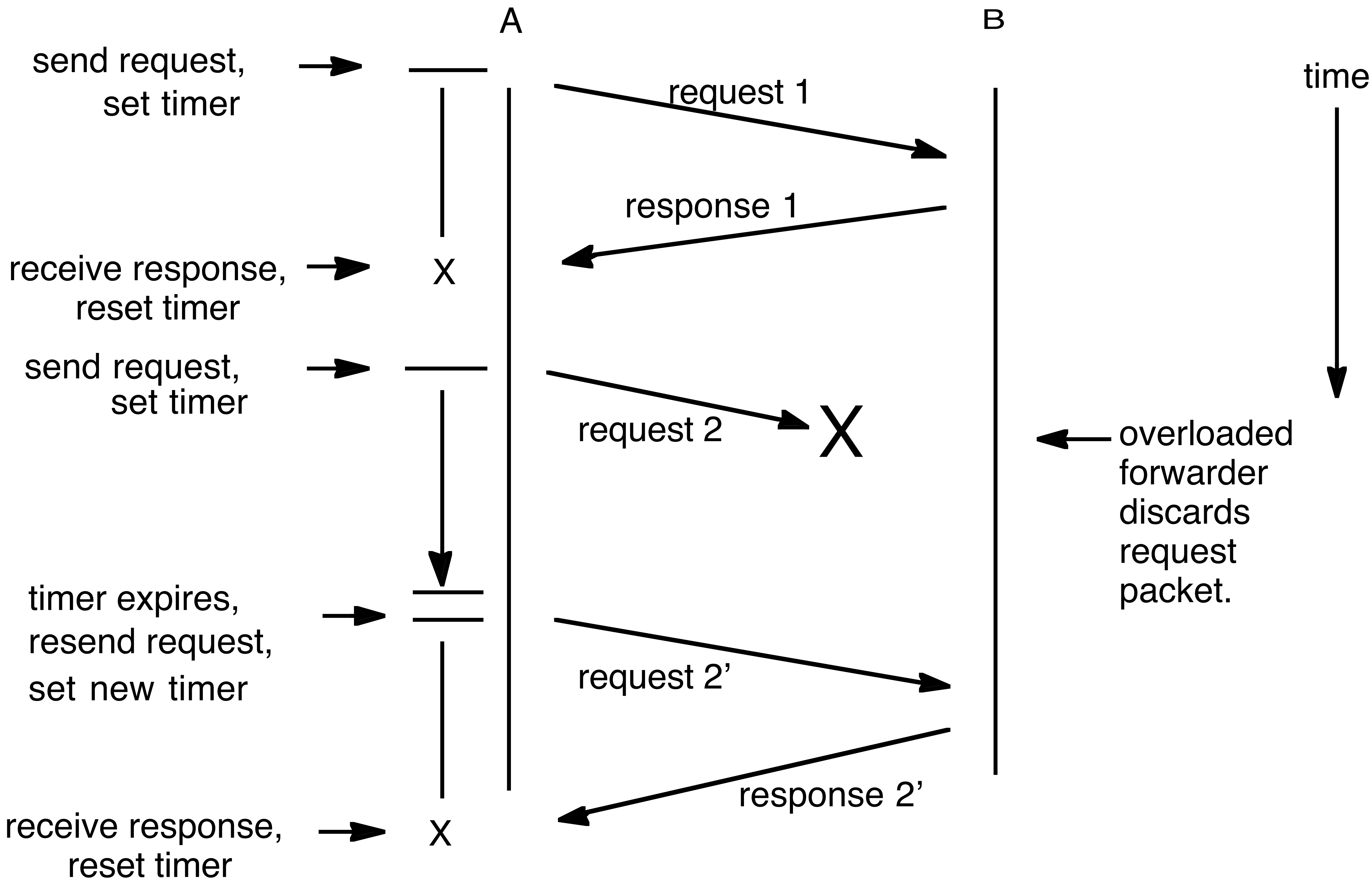


Fig © Saltzer & Kaashoek

Resending Packets

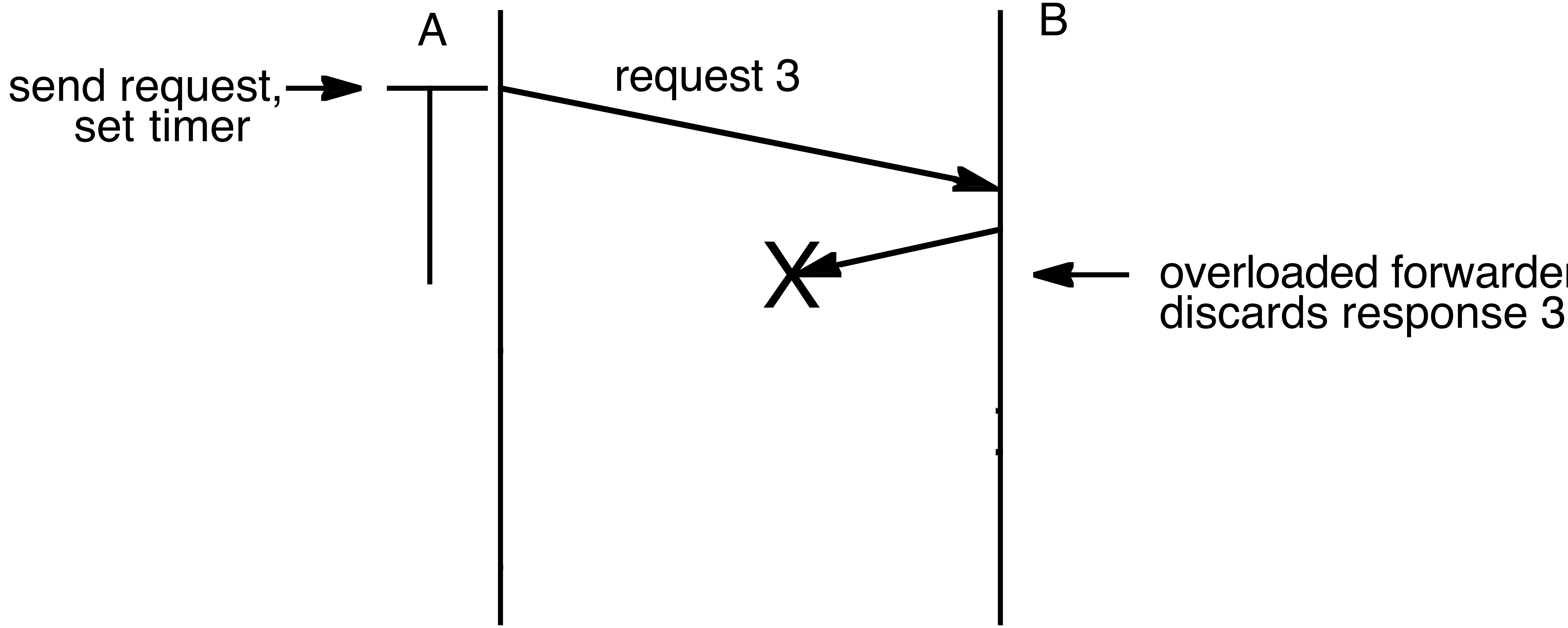


Fig © Saltzer & Kaashoek

Resending Packets

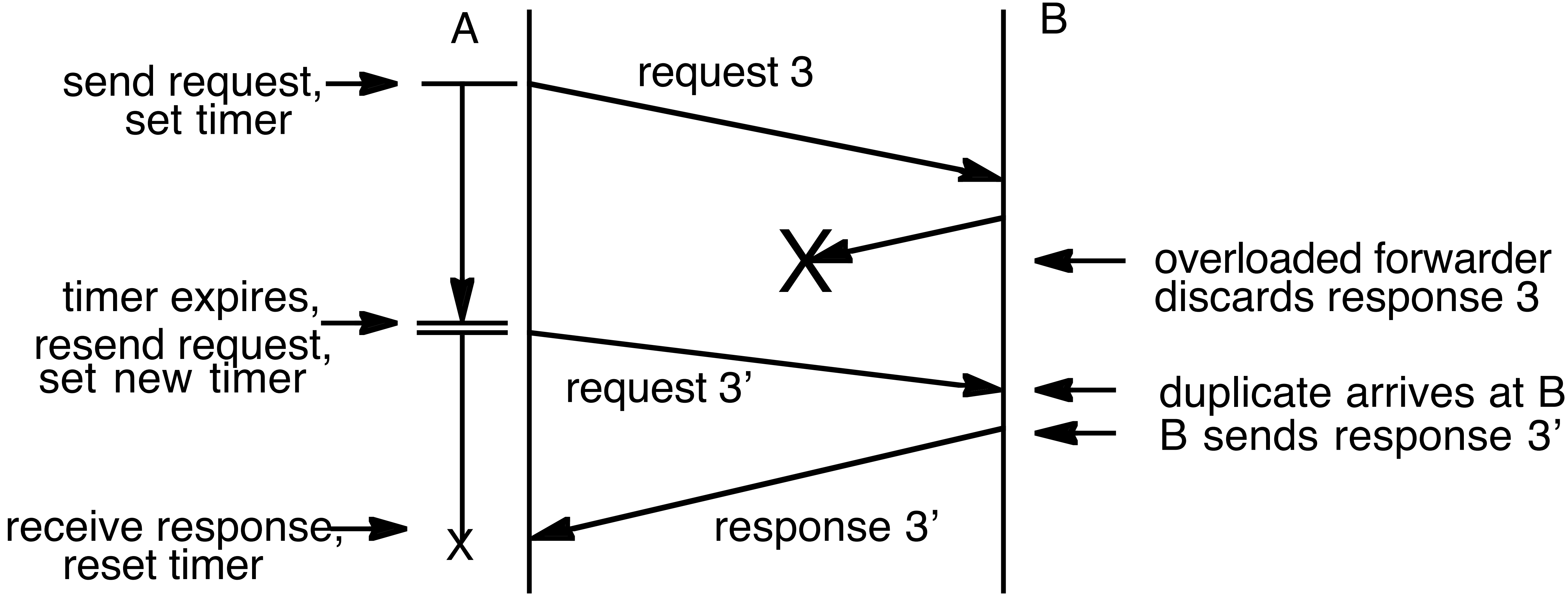


Fig © Saltzer & Kaashoek

Resending Packets

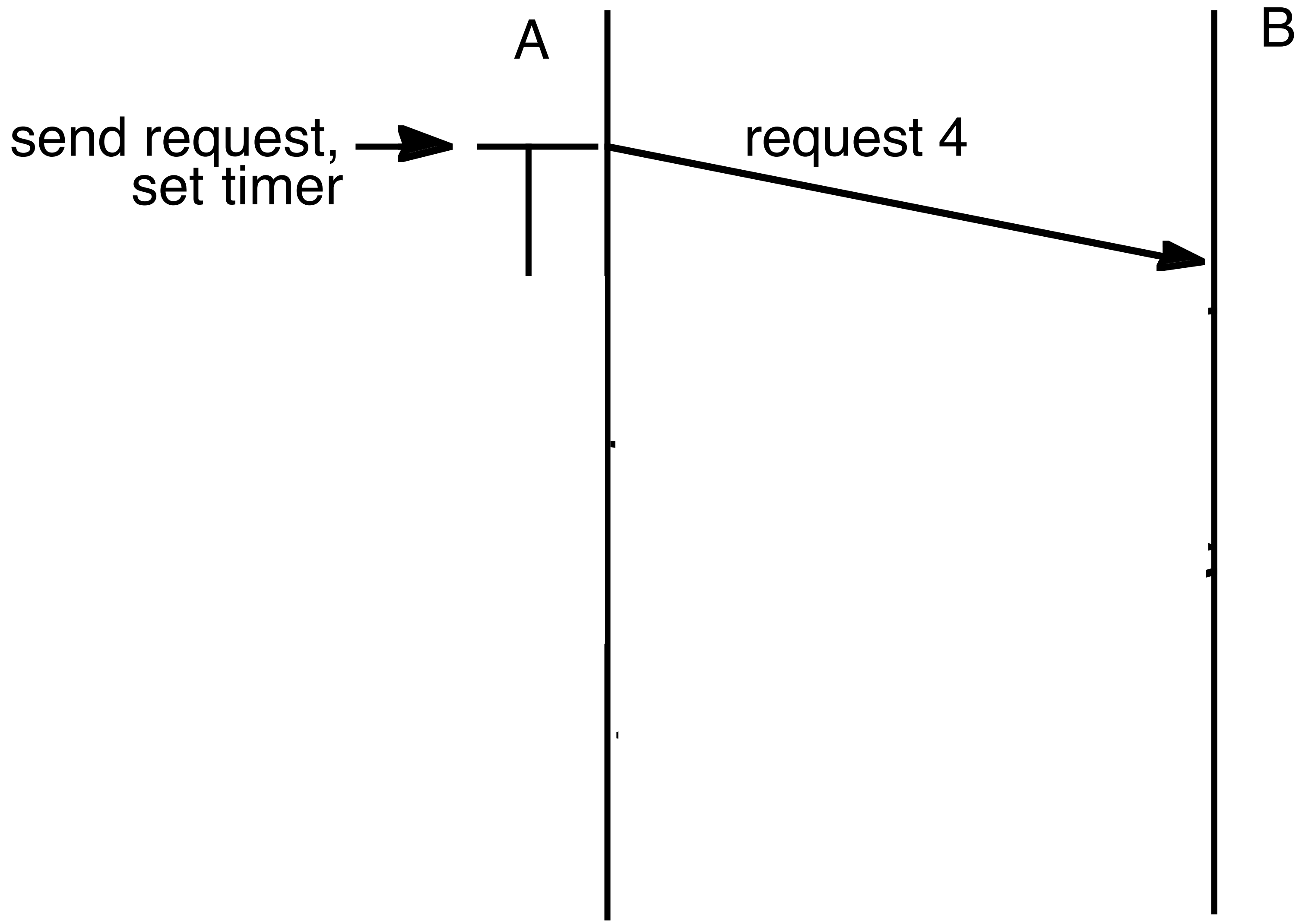


Fig © Saltzer & Kaashoek

Resending Packets

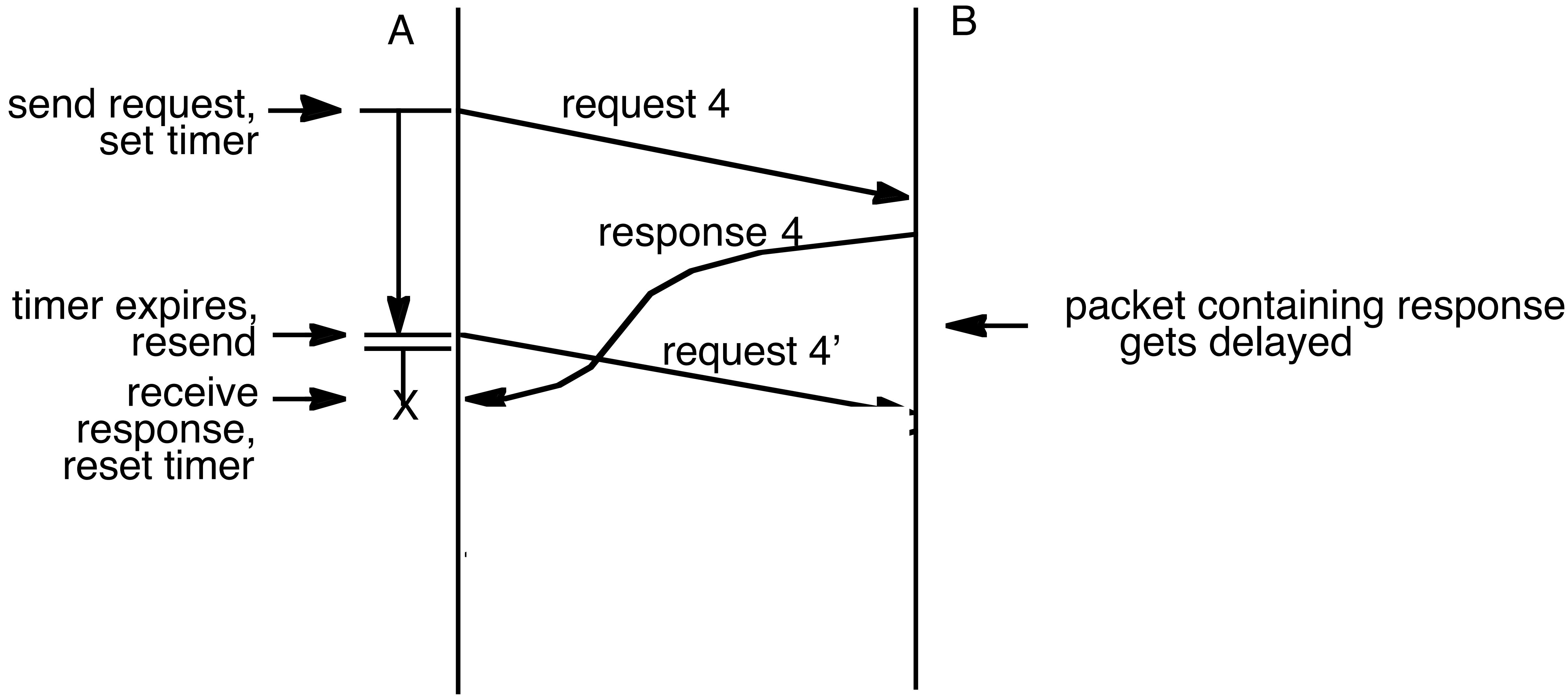


Fig © Saltzer & Kaashoek

Resending Packets

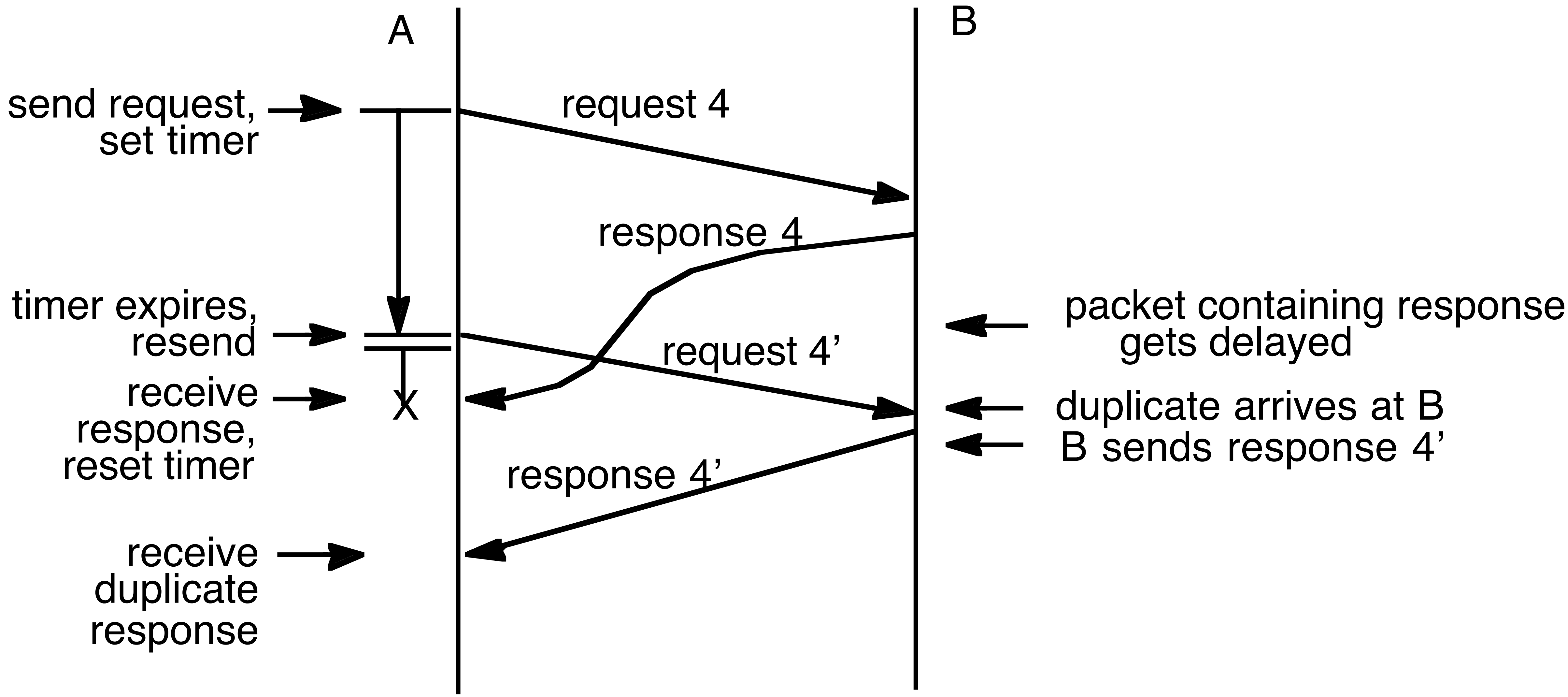


Fig © Saltzer & Kaashoek

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 hide the complexity of what’s below them
- Next class: all RPC

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 hide the complexity of what's below them
- Next class: all RPC

Networks as Abstractions

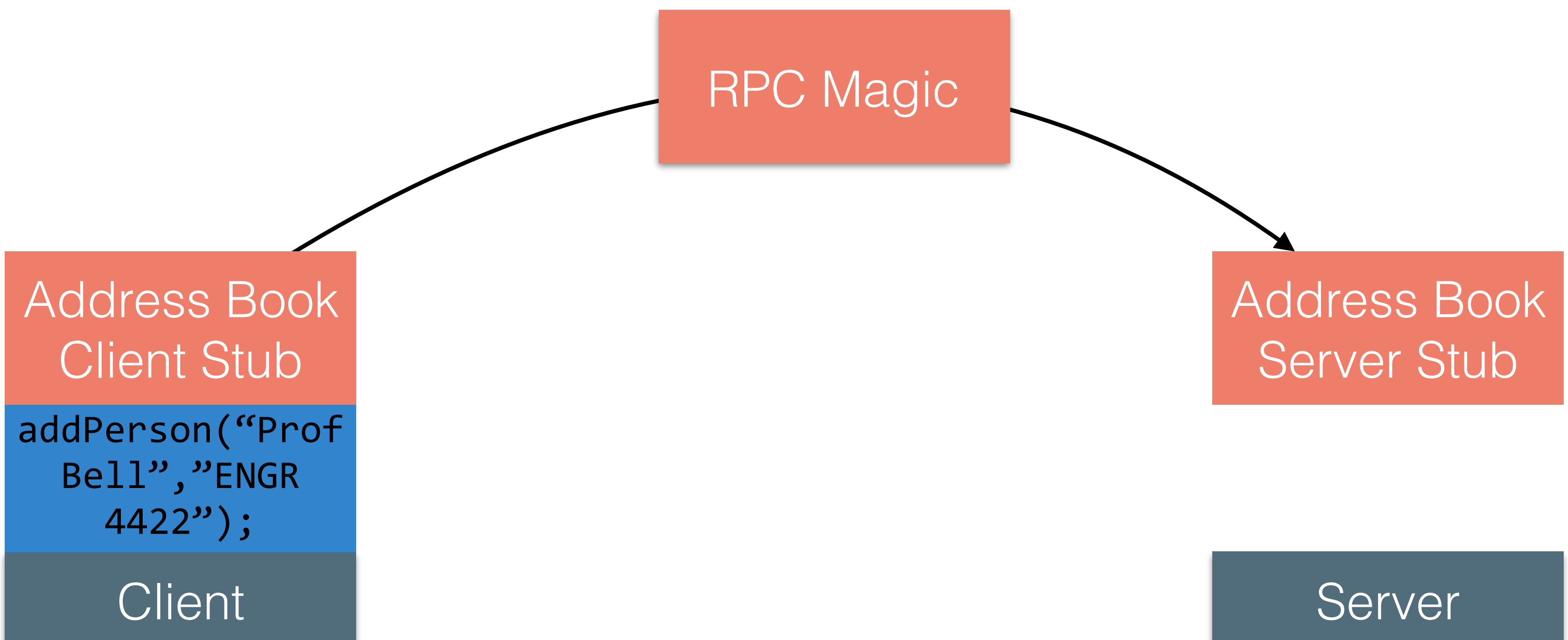
- Obviously, we don't think or care about packets
- We think and care about sending data!
- We want abstractions, like RPC (Remote Procedure Calls)
- Abstractions hide the complexity of what's below them
- Next class: all RPC

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 hide the complexity of what's below them
- Next class: all RPC

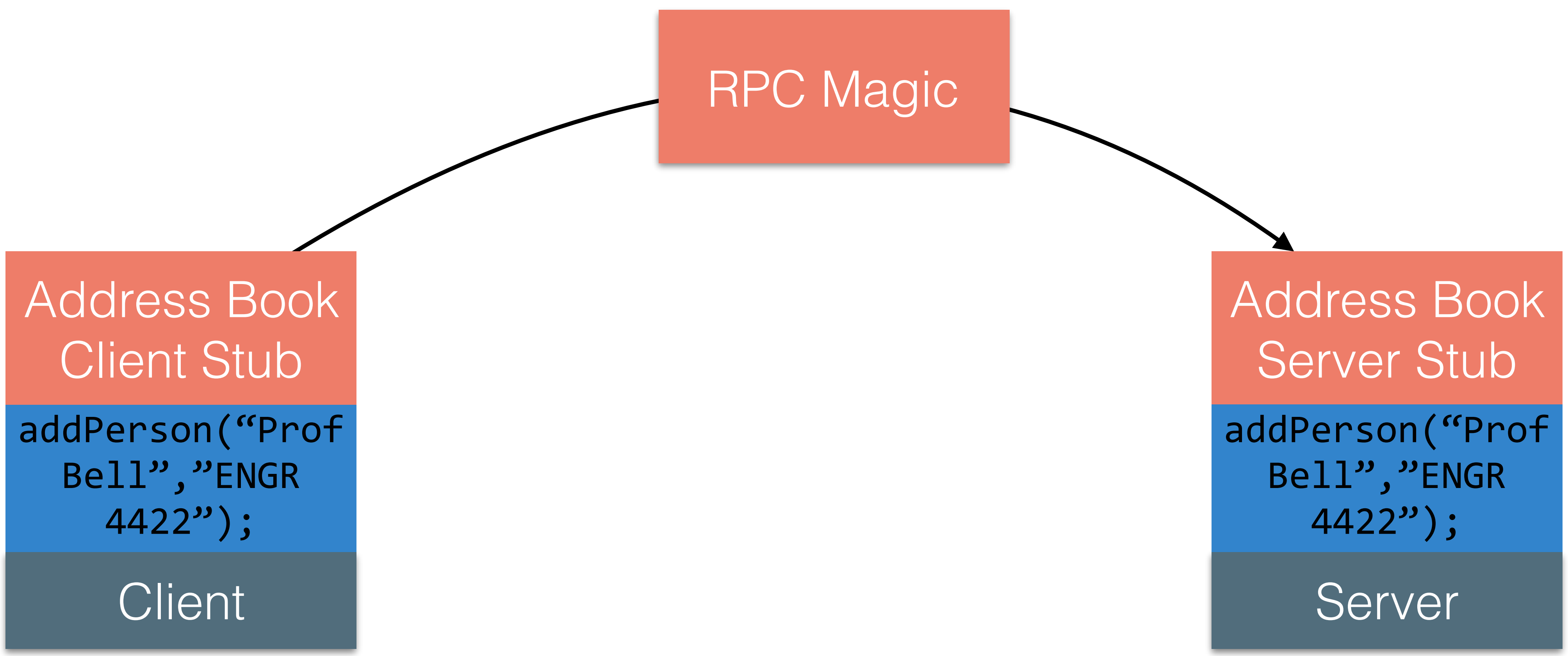
Networks as Abstractions

- 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 hide the complexity of what’s below them
- Next class: all RPC



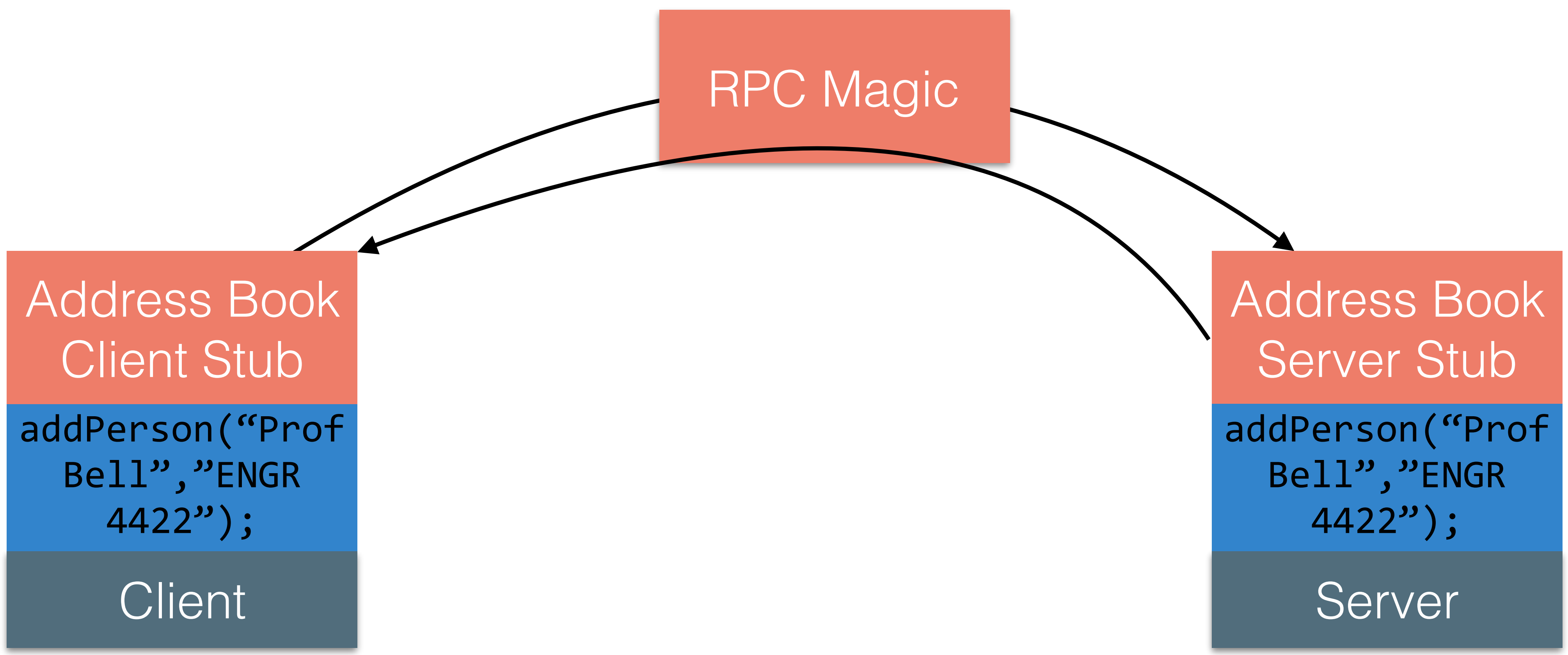
Networks as Abstractions

- 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 hide the complexity of what's below them
- Next class: all RPC



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 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

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

Link layer

Physical links: care about how to deliver packets

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

Network layer	**Figures out where to send packets**
Link layer	**Physical links: care about how to deliver packets**

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

End-to-End layer	**Handles packet loss, etc. Translates from application-data to packets, implements a protocol**
Network layer	**Figures out where to send packets**
Link layer	**Physical links: care about how to deliver packets**

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)

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) **UDP** ✅ **TCP** ✅
- Ordering control
- Flow control
  (not worrying about congestion)

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) **UDP** ✅ **TCP** ✅
- Ordering control **UDP** ❎ **TCP** ✅
- Flow control
  (not worrying about congestion)

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) **UDP** ✅ **TCP** ✅
- Ordering control **UDP** ❎ **TCP** ✅
- Flow control **UDP** ❎ **TCP** ✅
  (not worrying about congestion)

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!

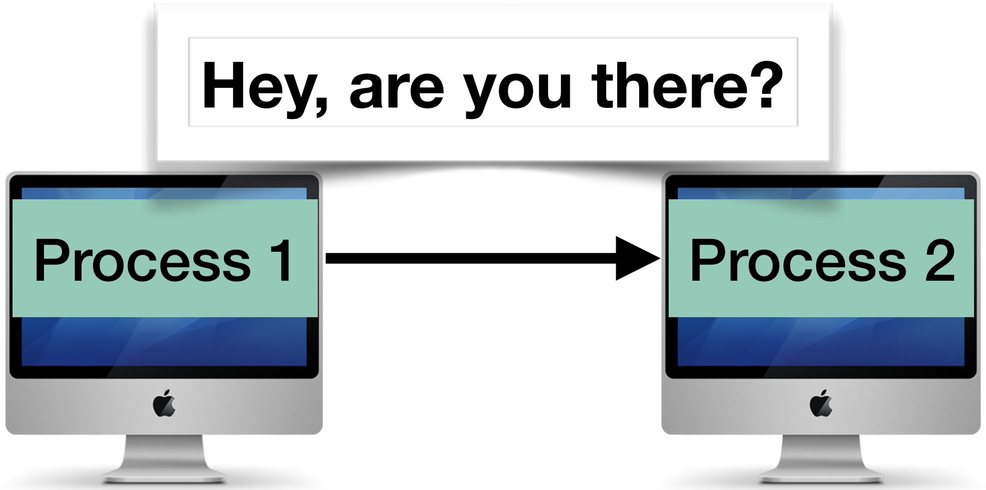
Preview for Next Class

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

Preview for Next Class

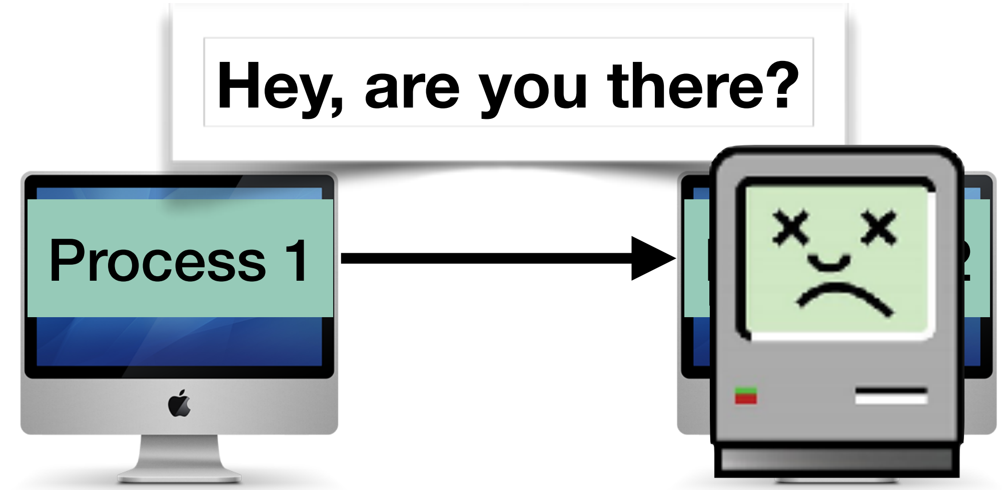
Spoiler alert: You can not tell the difference in a distributed system between a computer failing and network being arbitrarily slow!

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!

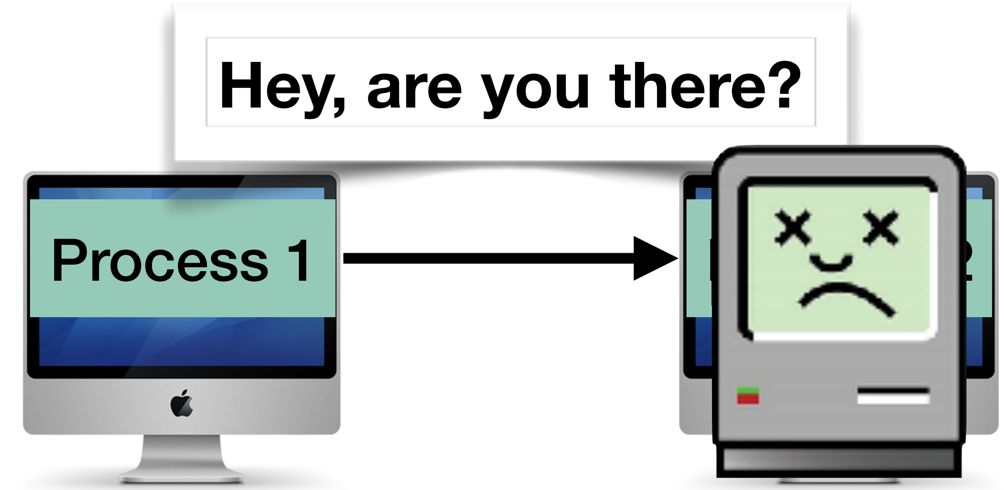
Preview for Next Class



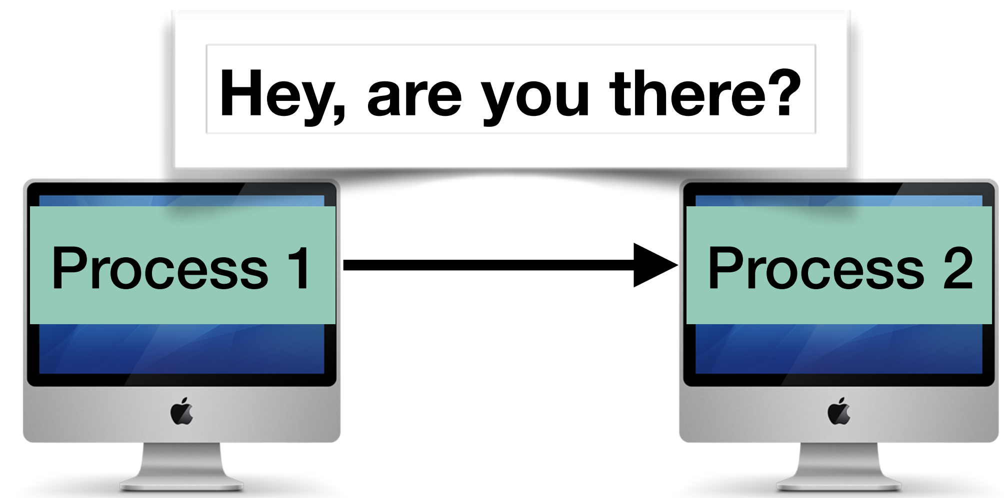
Computer 2 fails

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

Preview for Next Class

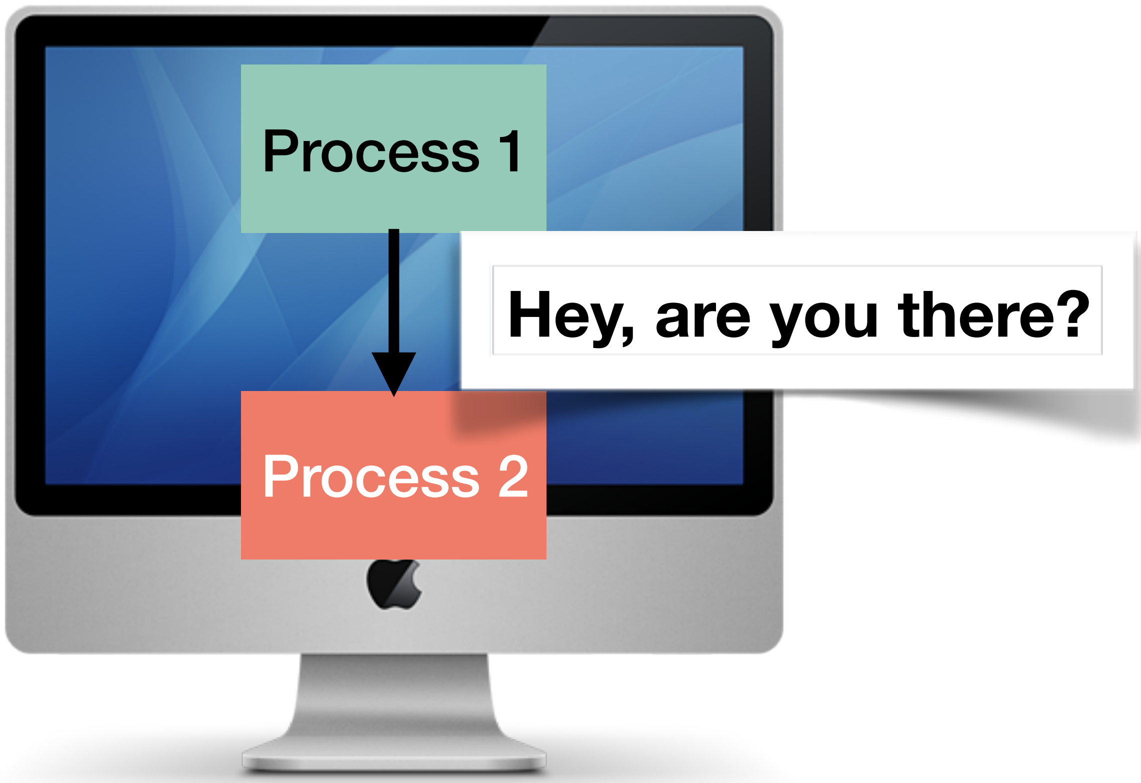


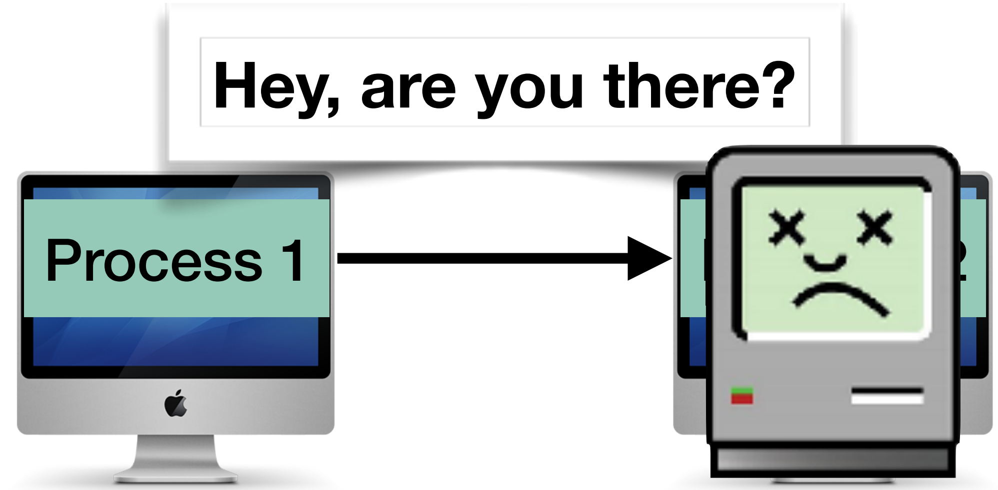
Computer 2 fails



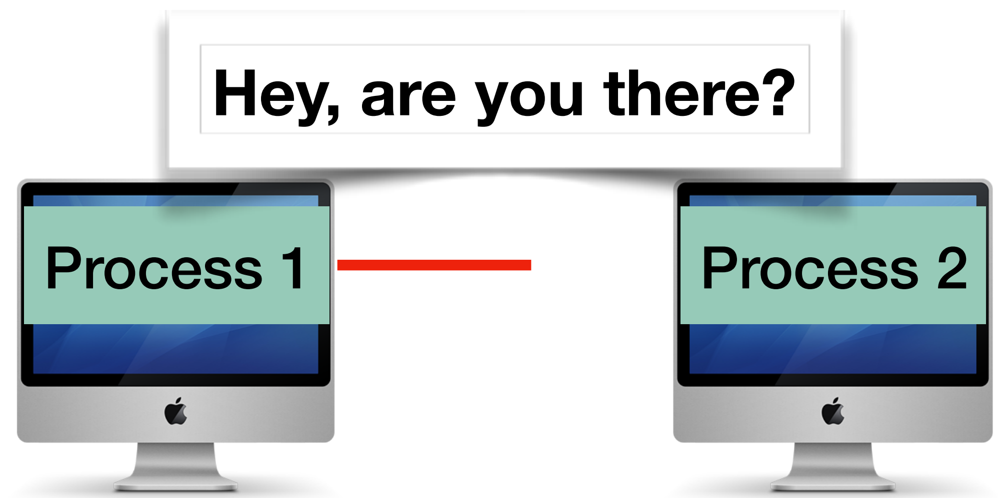
Spoiler alert: You can not tell the difference in a distributed system between a computer failing and network being arbitrarily slow!

Preview for Next Class



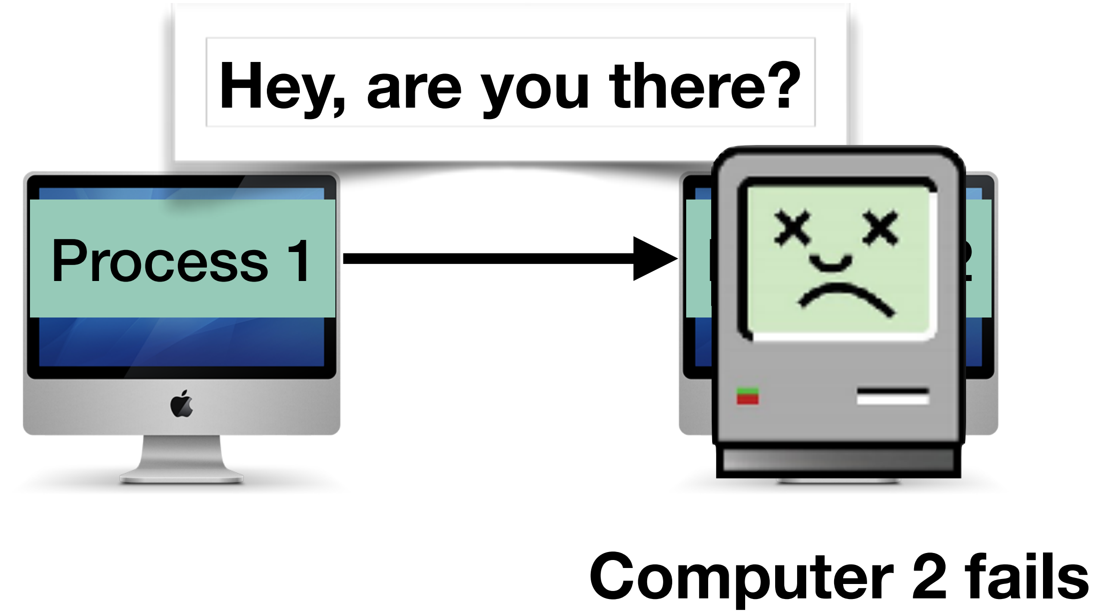


Computer 2 fails

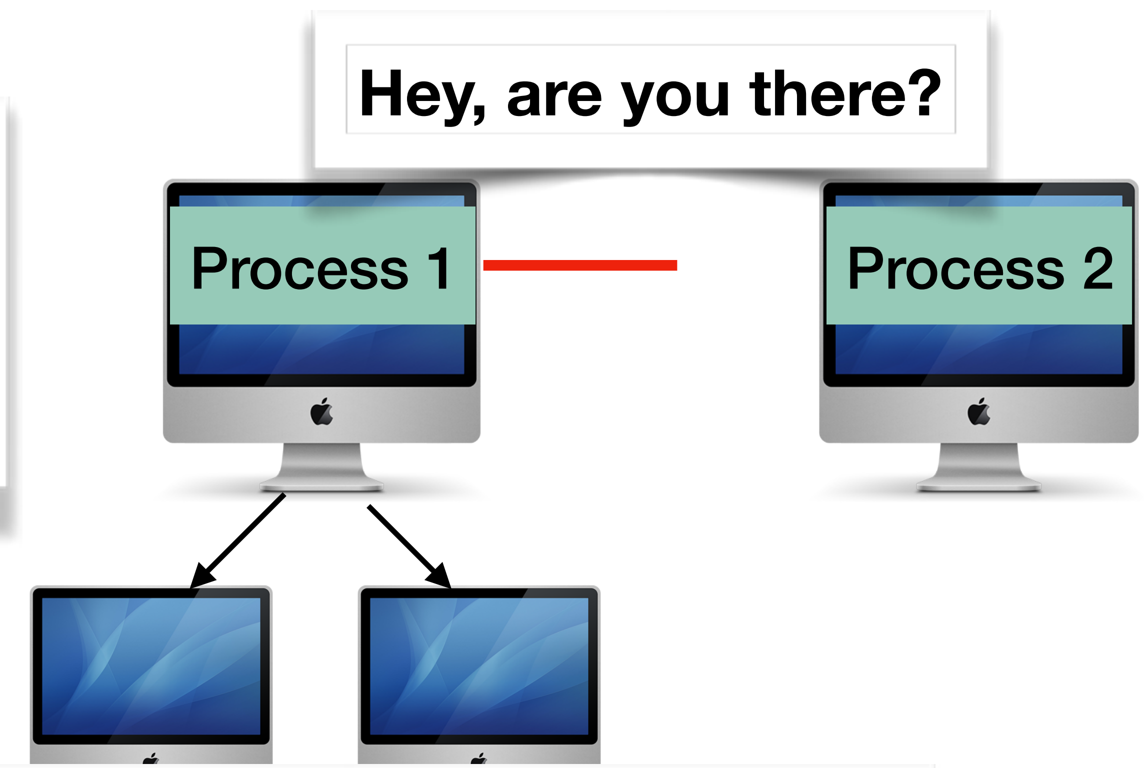


Spoiler alert: You can not tell the difference in a distributed system between a computer failing and network being arbitrarily slow!

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

Socrative

- Reminder - class name is CS475