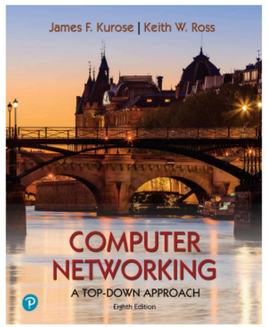
CS 241: Systems Programming
Lecture 27. Sockets I

Fall 2025

Prof. Stephen Checkoway



Slides adapted from the slides that accompany this book

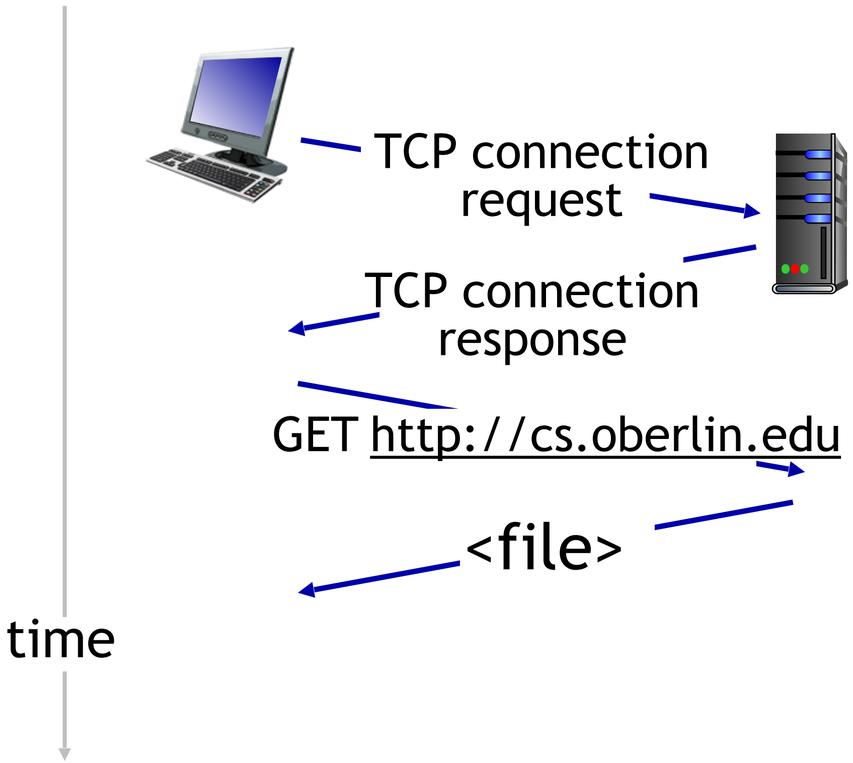
Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Network Protocols

Network protocols are between computers (devices) instead of humans

A protocol defines:

- the ***format*** and ***order*** of messages send/received between network entities
- the ***actions*** taken upon message receipt



Networks are complex systems

Networks are complex systems

They have many components:

- Hosts, routers/switches, links, protocols

Networks are complex systems

They have many components:

- Hosts, routers/switches, links, protocols

They can get quite large:

- Millions of hosts and devices!

Networks are complex systems

They have many components:

- Hosts, routers/switches, links, protocols

They can get quite large:

- Millions of hosts and devices!

They are shared among many traffic flows

Networks are complex systems

They have many components:

- Hosts, routers/switches, links, protocols

They can get quite large:

- Millions of hosts and devices!

They are shared among many traffic flows

They have to provide services to many concurrent applications

Networks are complex systems

They have many components:

- Hosts, routers/switches, links, protocols

They can get quite large:

- Millions of hosts and devices!

They are shared among many traffic flows

They have to provide services to many concurrent applications

Is there any hope of organizing all the functionality a network should provide?

Let’s look at another complex system for inspiration…

Example: organization of air travel

end-to-end transfer of person plus baggage

Example: organization of air travel

end-to-end transfer of person plus baggage

ticket (purchase)

Example: organization of air travel

end-to-end transfer of person plus baggage

ticket (purchase)

baggage (check)

Example: organization of air travel

end-to-end transfer of person plus baggage

ticket (purchase)

baggage (check)

gates (load)

Example: organization of air travel

end-to-end transfer of person plus baggage

ticket (purchase)

baggage (check)

gates (load)

runway takeoff

Example: organization of air travel

end-to-end transfer of person plus baggage

ticket (purchase)

baggage (check)

gates (load)

runway takeoff

airplane routing

Example: organization of air travel

end-to-end transfer of person plus baggage

ticket (purchase)

baggage (check)

gates (load)

runway takeoff

airplane routing

airplane routing

Example: organization of air travel

end-to-end transfer of person plus baggage

ticket (purchase)

baggage (check)

gates (load)

runway takeoff

airplane routing

airplane routing

airplane routing

Example: organization of air travel

end-to-end transfer of person plus baggage

ticket (purchase)

baggage (check)

gates (load)

runway takeoff

airplane routing

runway landing

airplane routing

airplane routing

Example: organization of air travel

end-to-end transfer of person plus baggage

ticket (purchase)

baggage (check)

gates (load)

runway takeoff

airplane routing

gates (unload)

runway landing

airplane routing

airplane routing

Example: organization of air travel

end-to-end transfer of person plus baggage

ticket (purchase)
baggage (check)
gates (load)
runway takeoff
airplane routing

baggage (claim)
gates (unload)
runway landing
airplane routing

airplane routing

Example: organization of air travel

end-to-end transfer of person plus baggage

ticket (purchase)
baggage (check)
gates (load)
runway takeoff
airplane routing

ticket (complain)
baggage (claim)
gates (unload)
runway landing
airplane routing

airplane routing

Example: organization of air travel

end-to-end transfer of person plus baggage

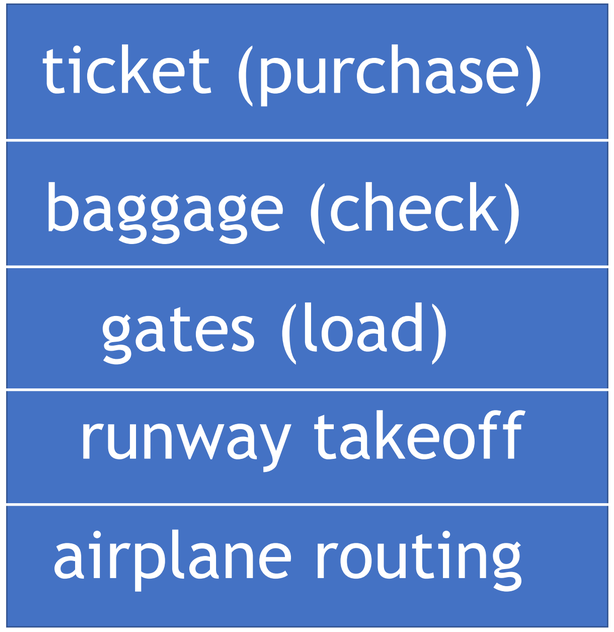
ticket (purchase)
baggage (check)
gates (load)
runway takeoff
airplane routing

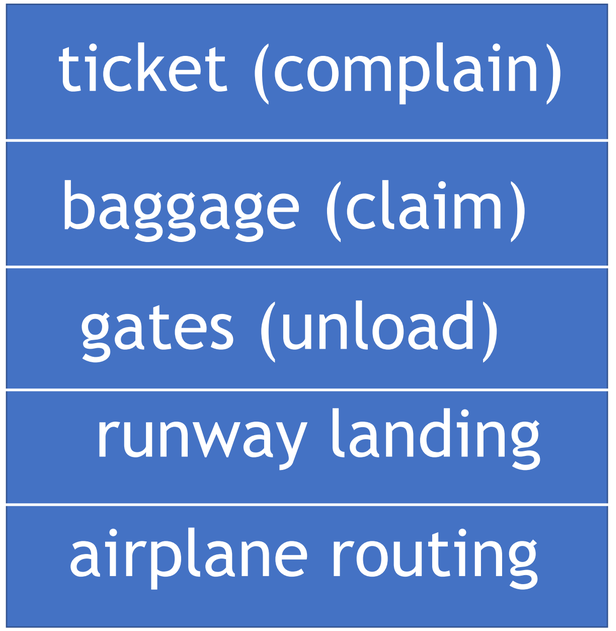
ticket (complain)
baggage (claim)
gates (unload)
runway landing
airplane routing

airplane routing

We can *define* the *system* of airline travel as a series of steps, involving many services

Example: organization of air travel





Example: organization of air travel

Example: organization of air travel

layers: each layer implements a service

- Via its own internal-layer actions
- Relying on services provided by layer below

Why layering?

An approach to designing/discussing complex systems

The explicit structure allows identification, relationship of system's pieces

- It gives us a layered reference model for discussion

Modularization eases maintenance, updating of system

- If you change one layer's service implementation, it's transparent to the rest of the system
- e.g., change in gate procedure doesn't affect rest of system

Layered Internet Protocol Stack

Layered Internet Protocol Stack

Application: supporting network applications

- e.g., HTTP

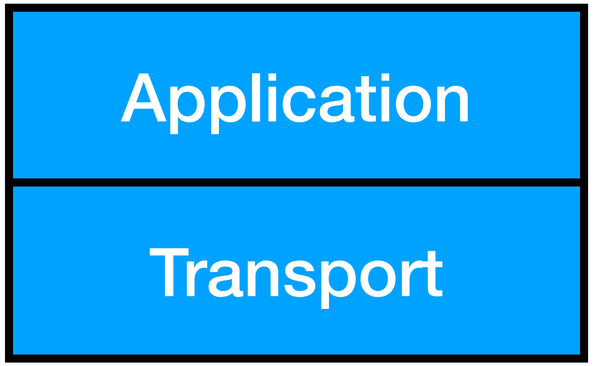
Layered Internet Protocol Stack

Application: supporting network applications

- e.g., HTTP

Transport: data transfer between processes on hosts

- e.g., TCP, UDP

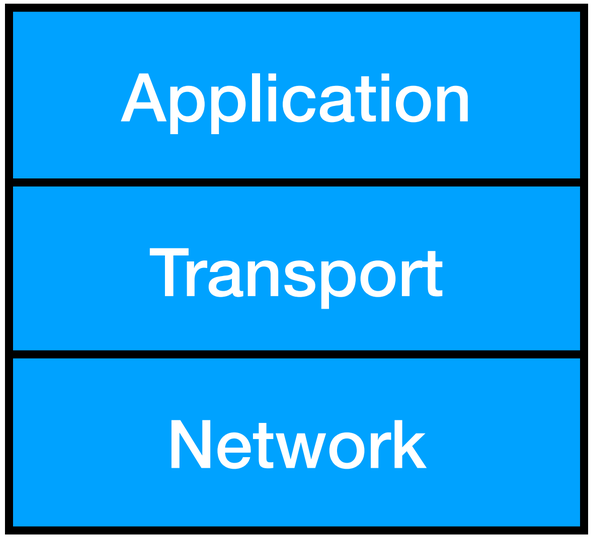


Layered Internet Protocol Stack

Application: supporting network applications
- e.g., HTTP

Transport: data transfer between processes on hosts
- e.g., TCP, UDP

Network: routing packets from source to destination
- e.g., IP



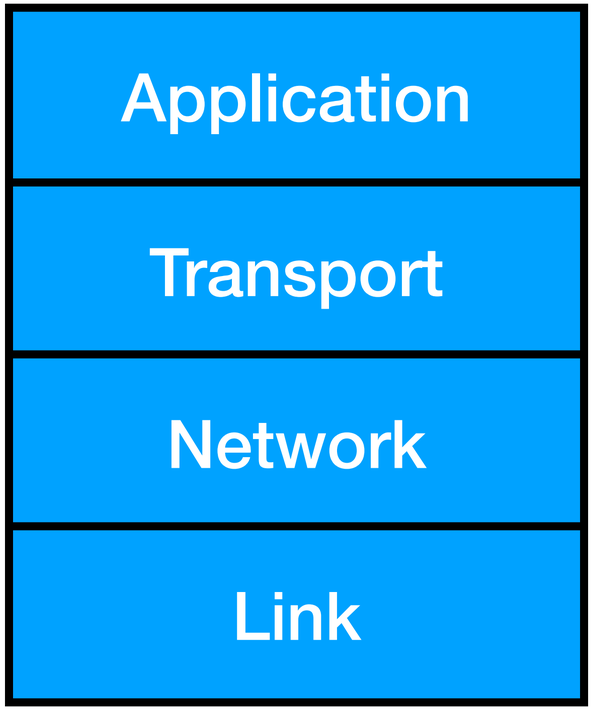
Layered Internet Protocol Stack

Application: supporting network applications
- e.g., HTTP

Transport: data transfer between processes on hosts
- e.g., TCP, UDP

Network: routing packets from source to destination
- e.g., IP

Link: data transfer between neighboring elements
- e.g., Ethernet, WiFi



Layered Internet Protocol Stack

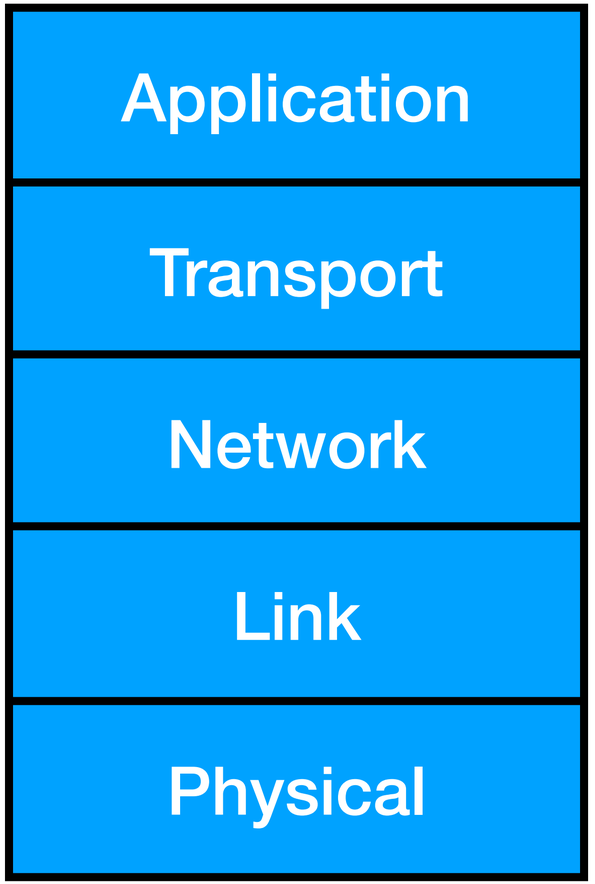
Application: supporting network applications
- e.g., HTTP

Transport: data transfer between processes on hosts
- e.g., TCP, UDP

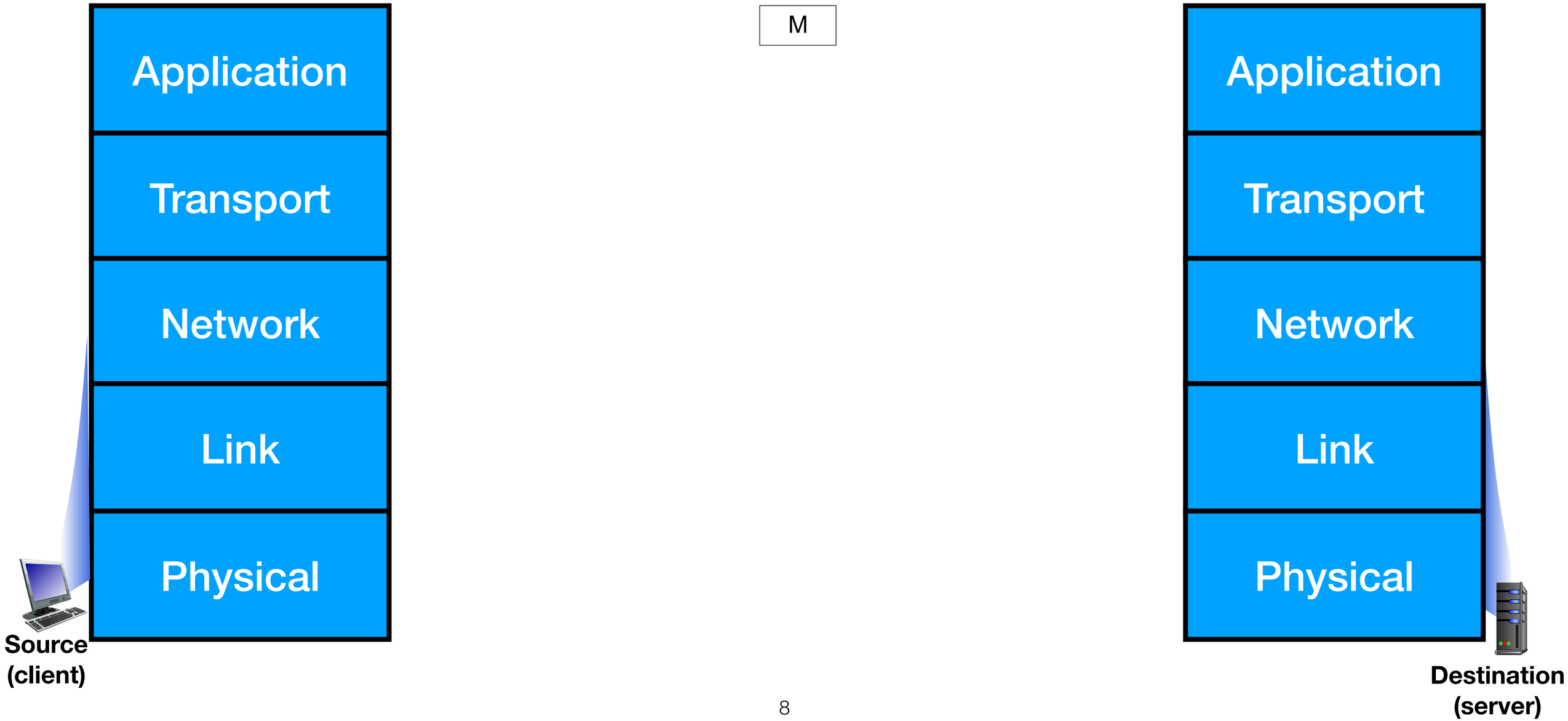
Network: routing packets from source to destination
- e.g., IP

Link: data transfer between neighboring elements
- e.g., Ethernet, WiFi

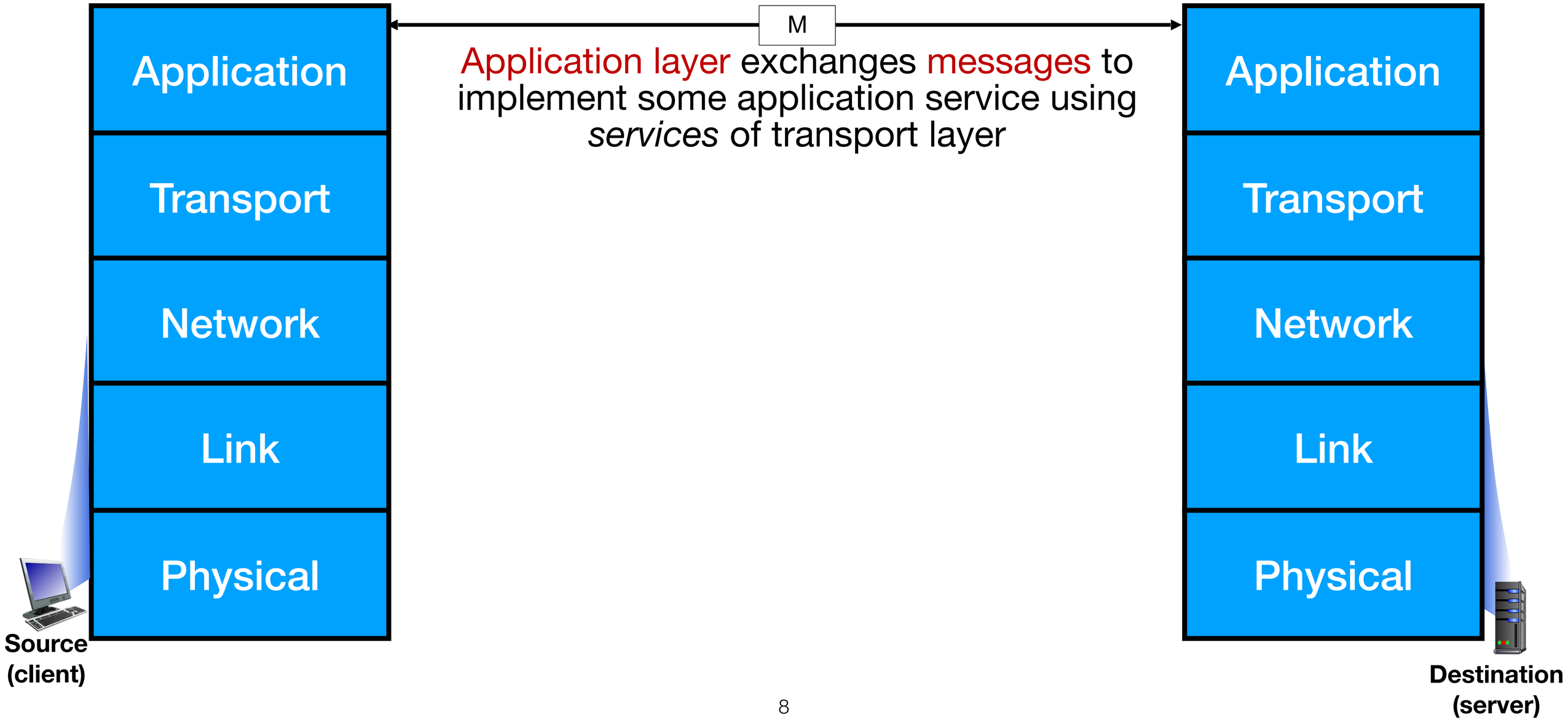
Physical: transmit data over wires (or wireless signals)



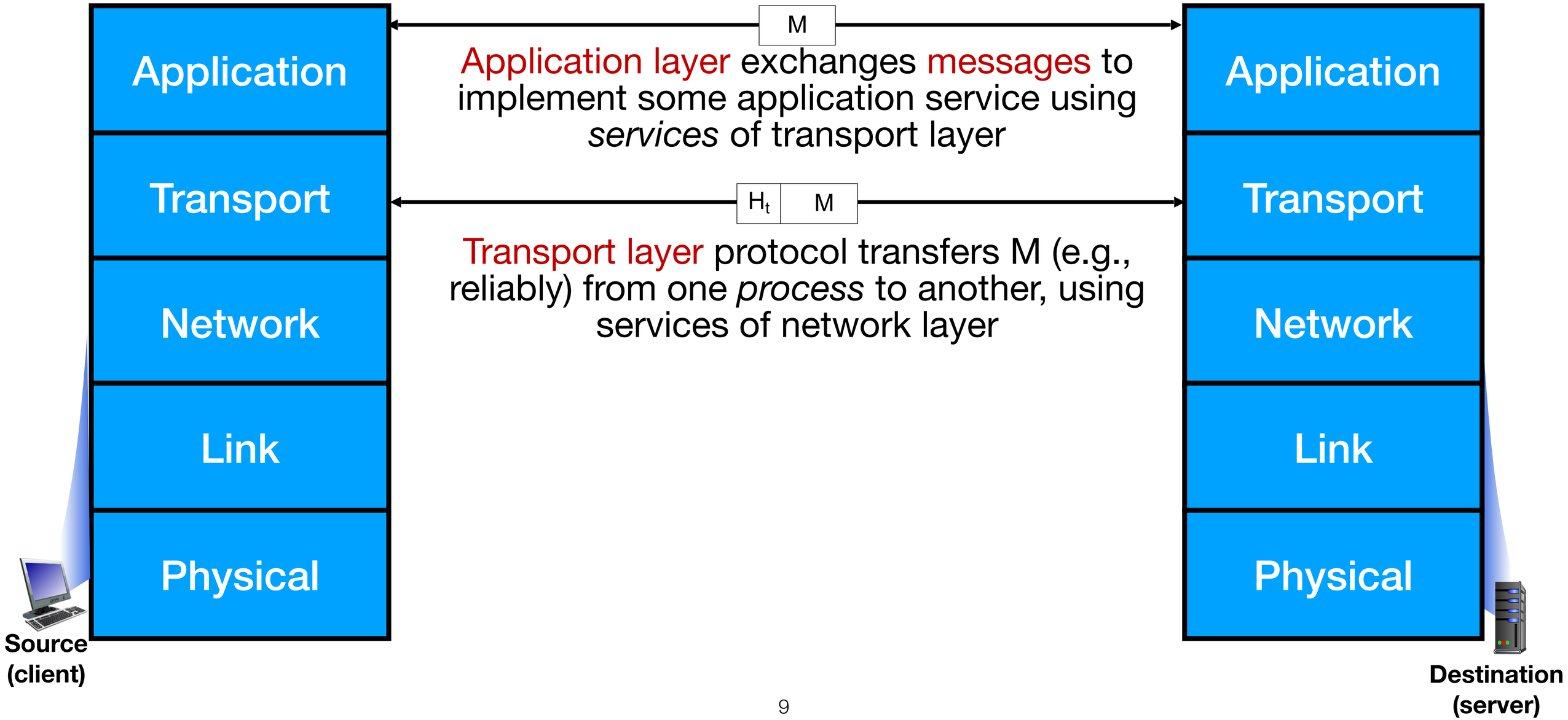
Services, Layering, and Encapsulation



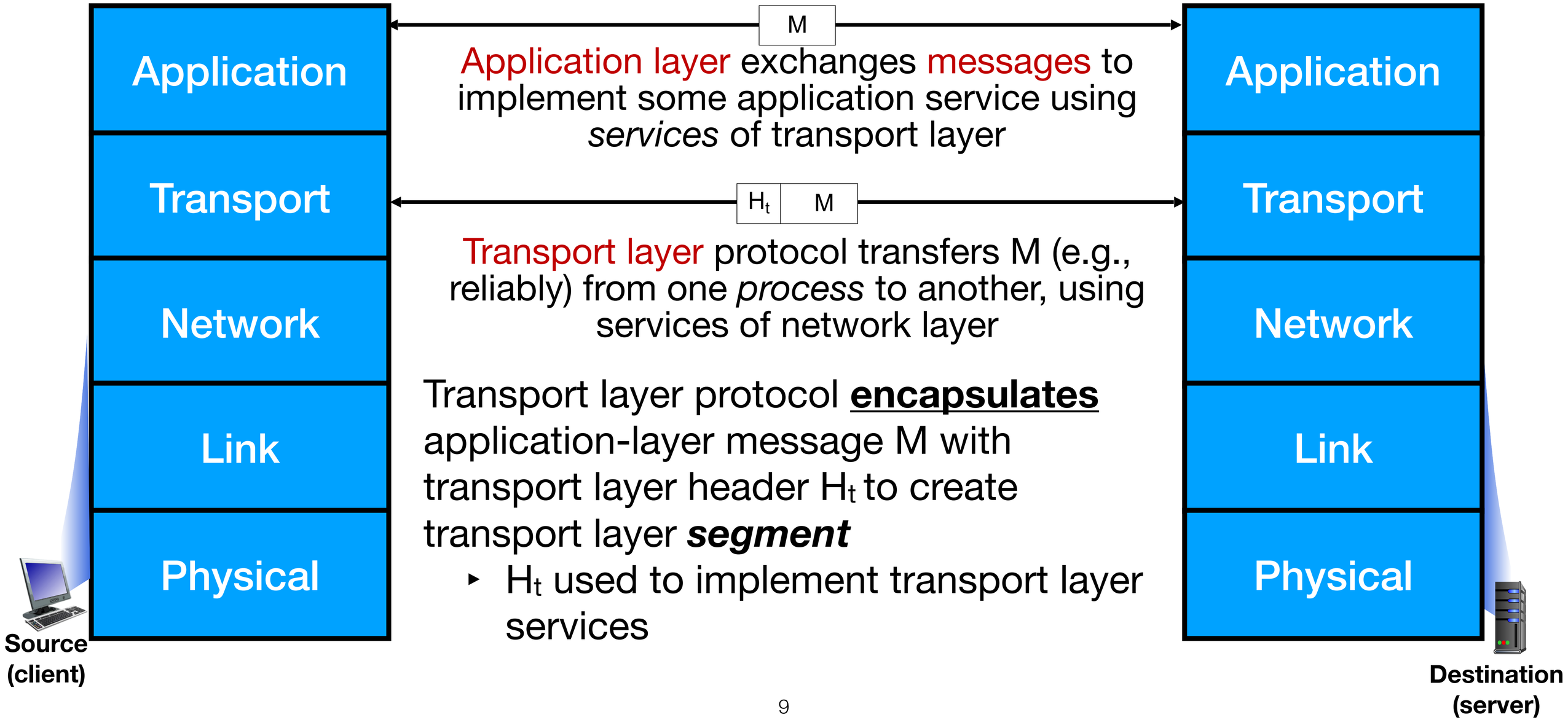
Services, Layering, and Encapsulation



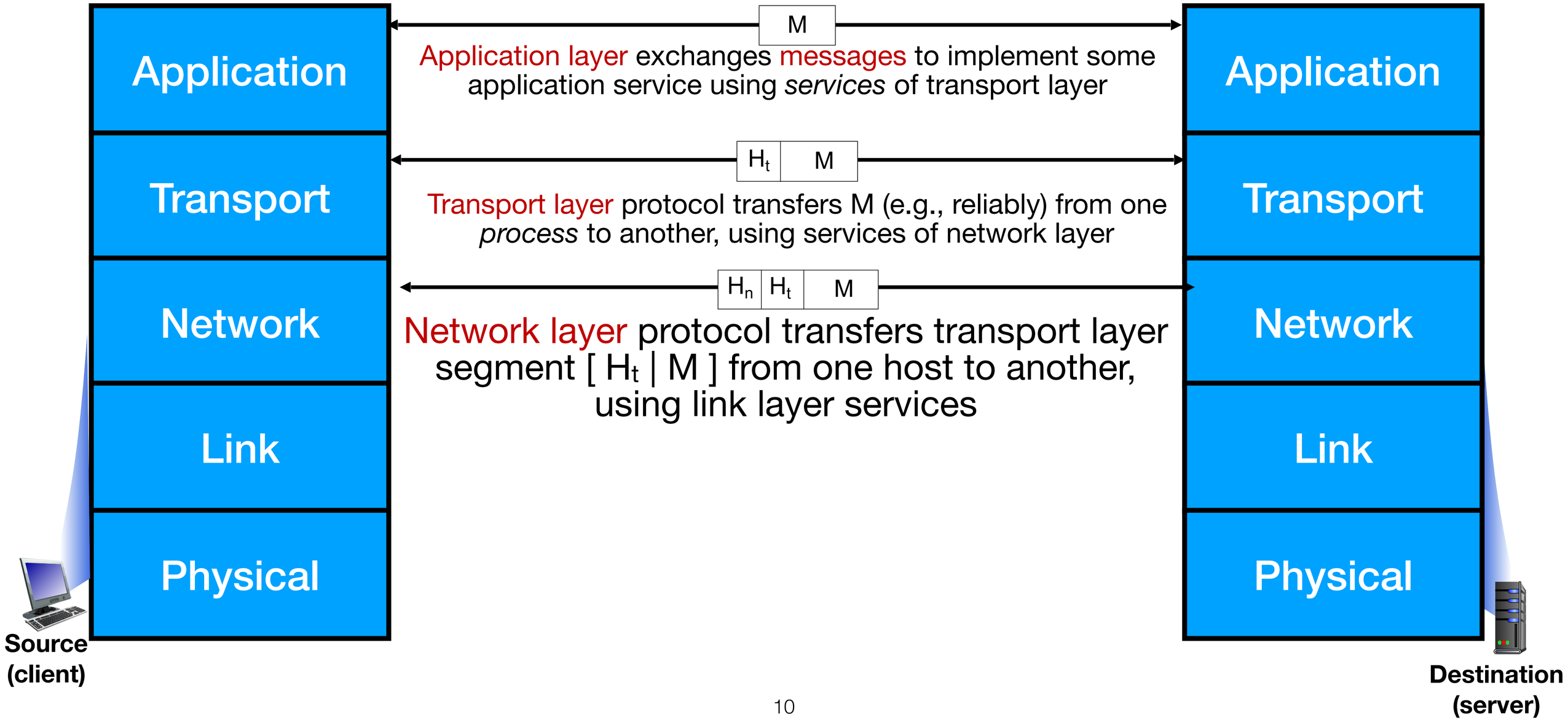
Services, Layering, and Encapsulation



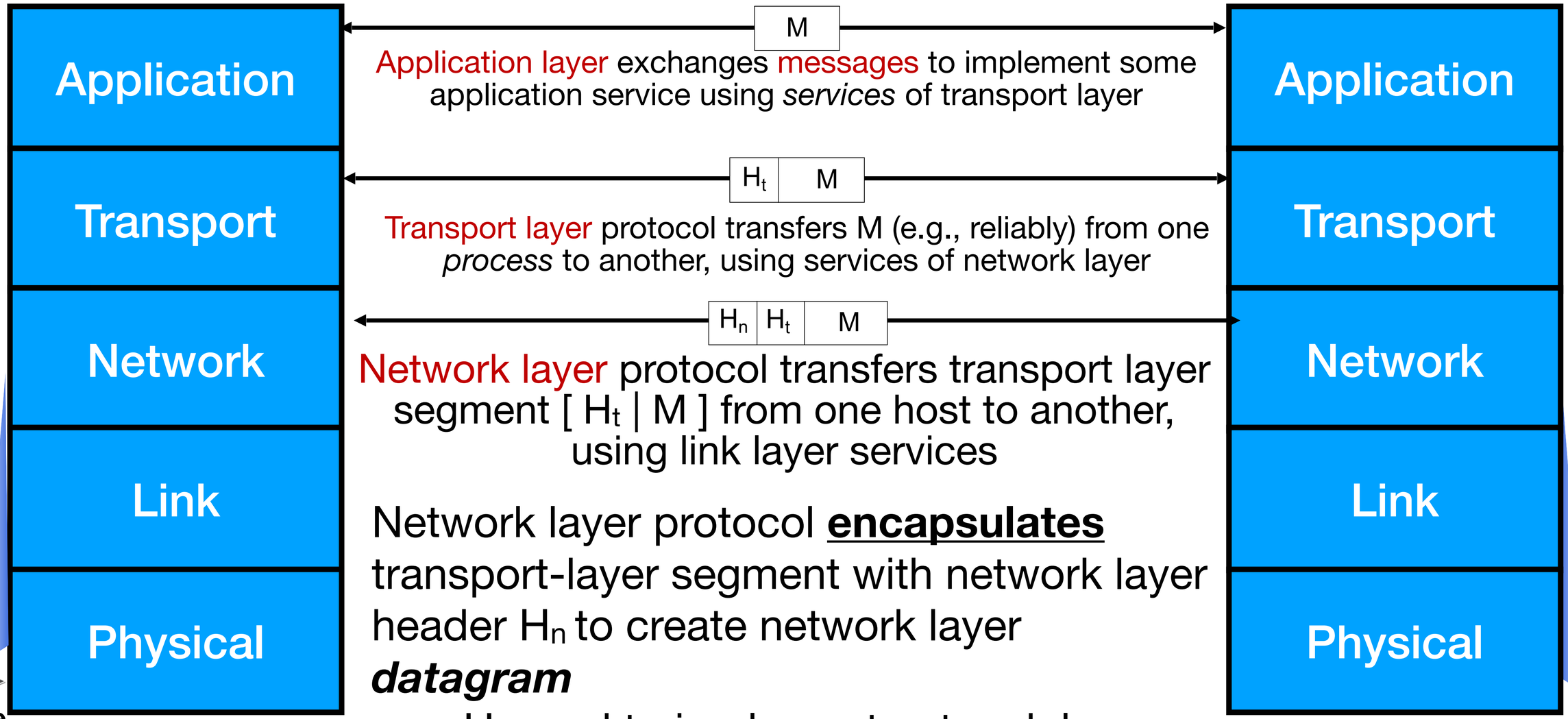
Services, Layering, and Encapsulation



Services, Layering, and Encapsulation



Services, Layering, and Encapsulation



Network layer protocol **<u>encapsulates</u>** transport-layer segment with network layer header H_n to create network layer ***datagram***

- H_t used to implement network layer services

Source (client)

Destination (server)

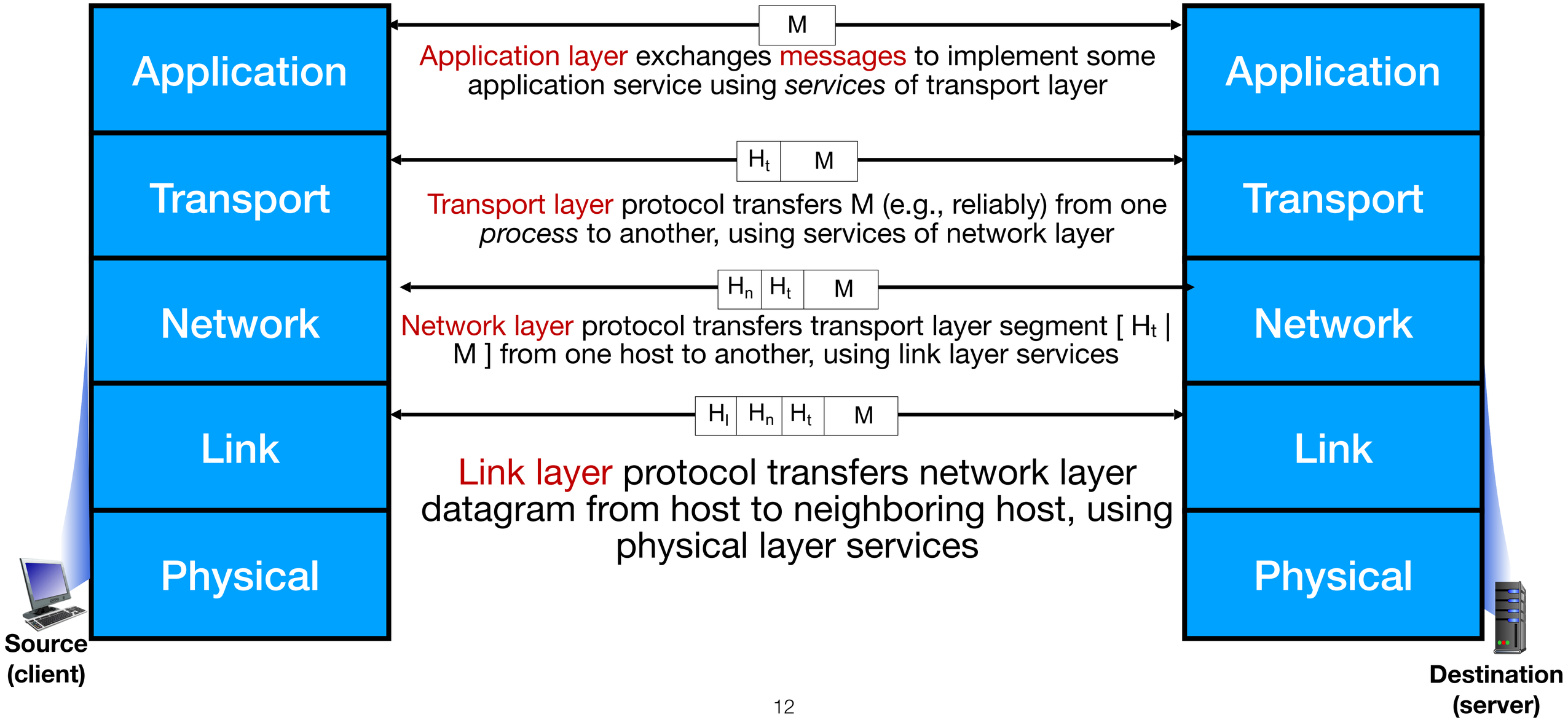
To send multiple messages between two end hosts, we should

A. Pick one path and follow it for all messages

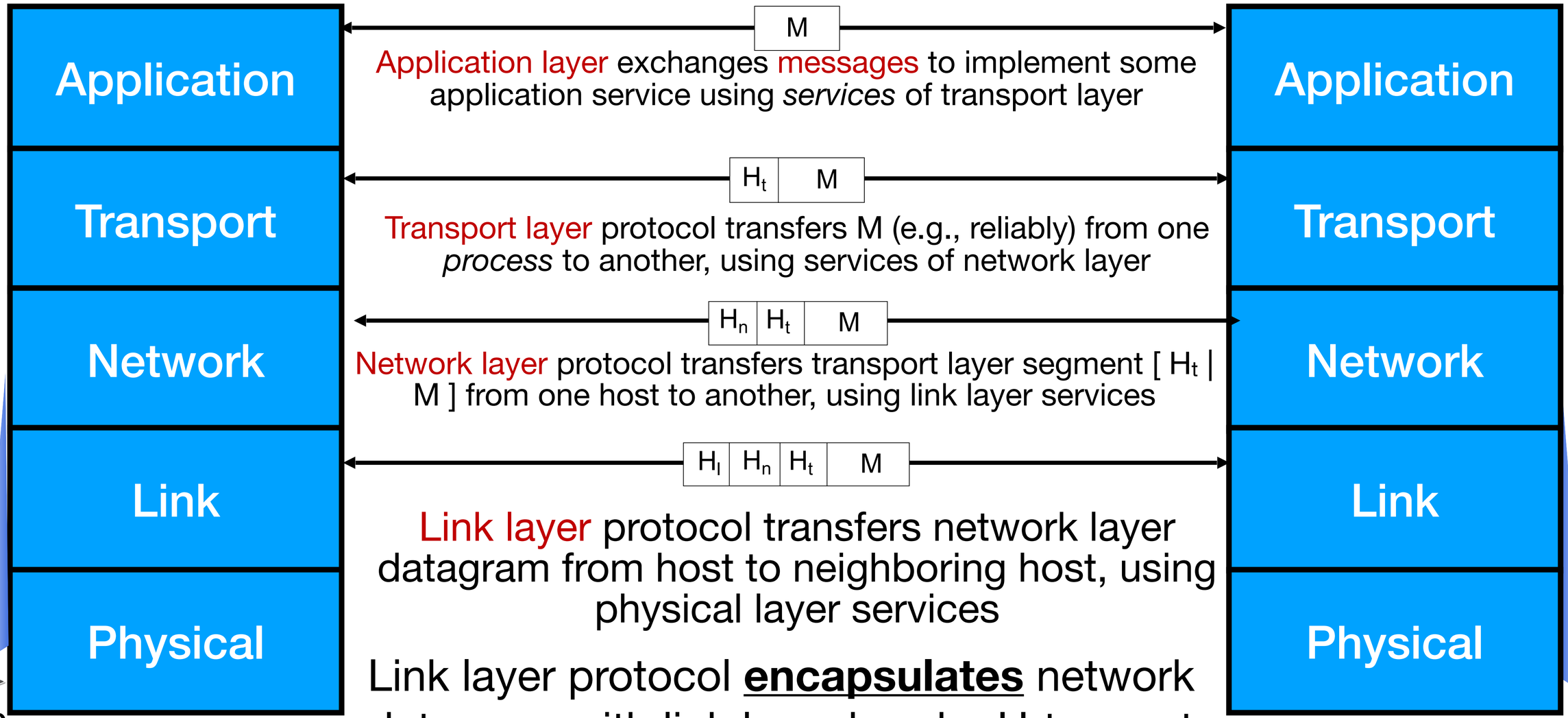
B. Choose a different path each time

C. Either of these will work

Services, Layering, and Encapsulation



Services, Layering, and Encapsulation



Source (client)

Destination (server)

Link layer protocol **encapsulates** network datagram with link layer header H_l to create link layer ***frame***

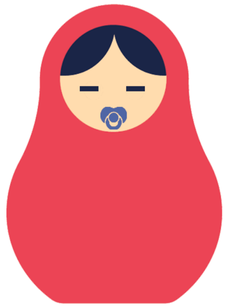
Encapsulation

Matryoshka dolls (stacking dolls)

Credit: https://dribbble.com/shots/7182188-Babushka-Boi

Encapsulation

Matryoshka dolls (stacking dolls)

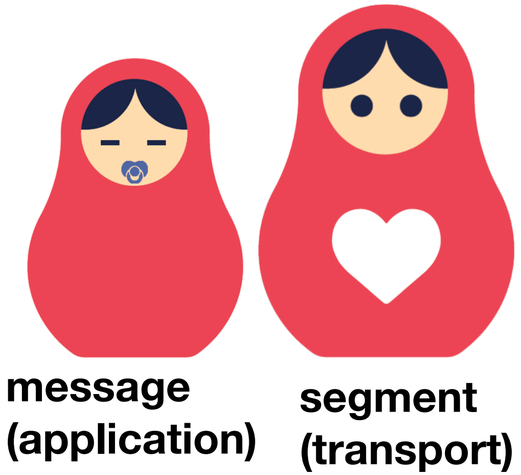


message (application)

Credit: https://dribbble.com/shots/7182188-Babushka-Boi

Encapsulation

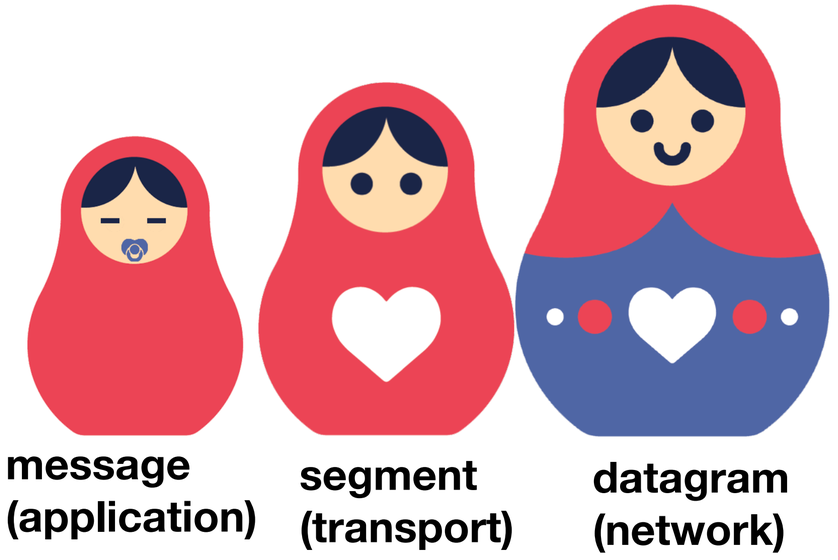
Matryoshka dolls (stacking dolls)



Credit: https://dribbble.com/shots/7182188-Babushka-Boi

Encapsulation

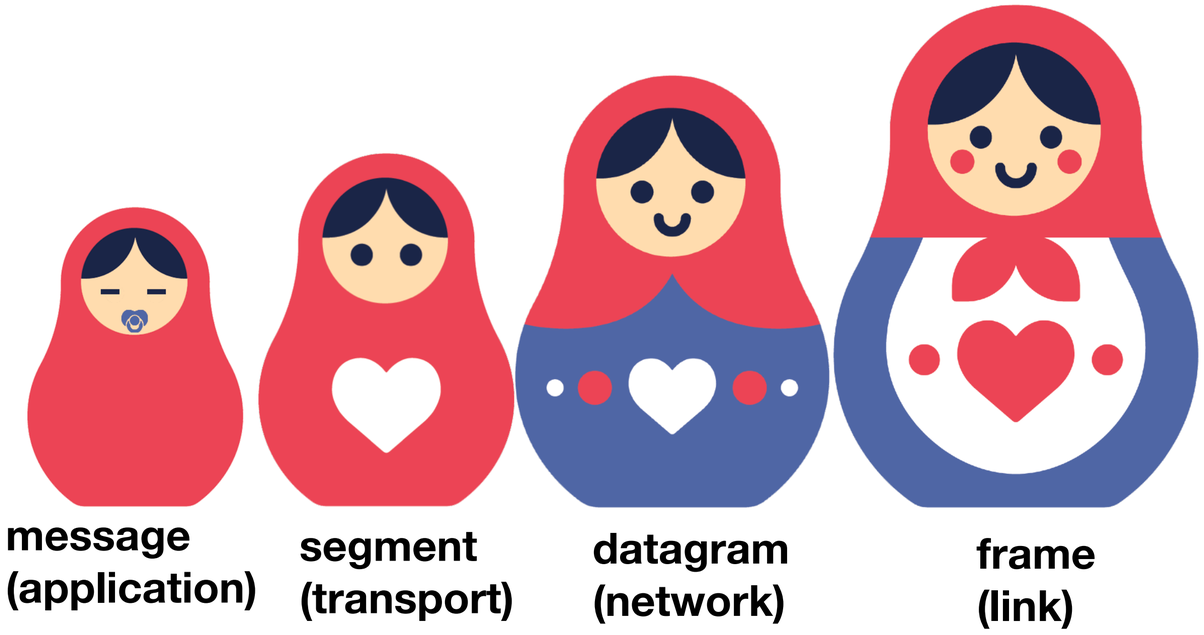
Matryoshka dolls (stacking dolls)



Credit: https://dribbble.com/shots/7182188-Babushka-Boi

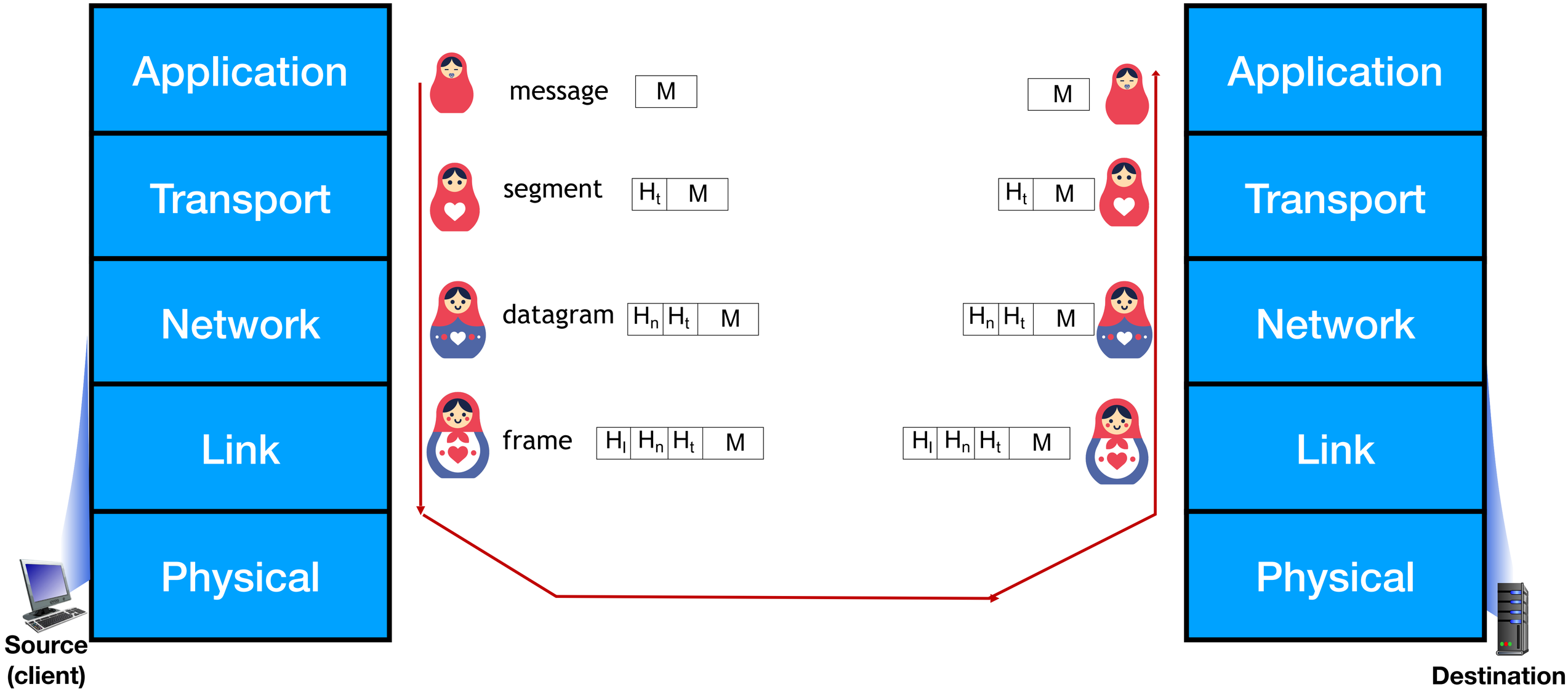
Encapsulation

Matryoshka dolls (stacking dolls)



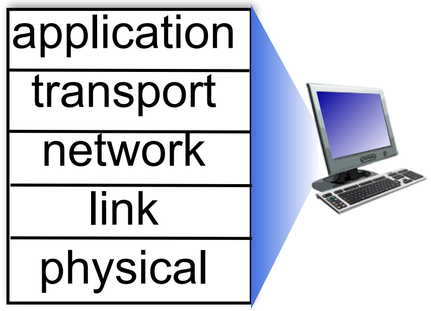
Credit: https://dribbble.com/shots/7182188-Babushka-Boi

Services, Layering, and Encapsulation

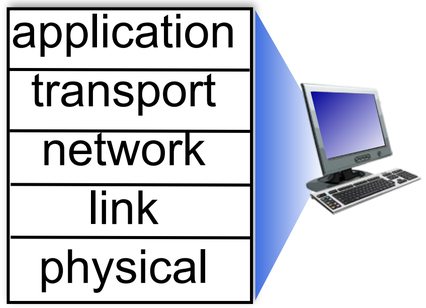


Encapsulation: end-to-end view

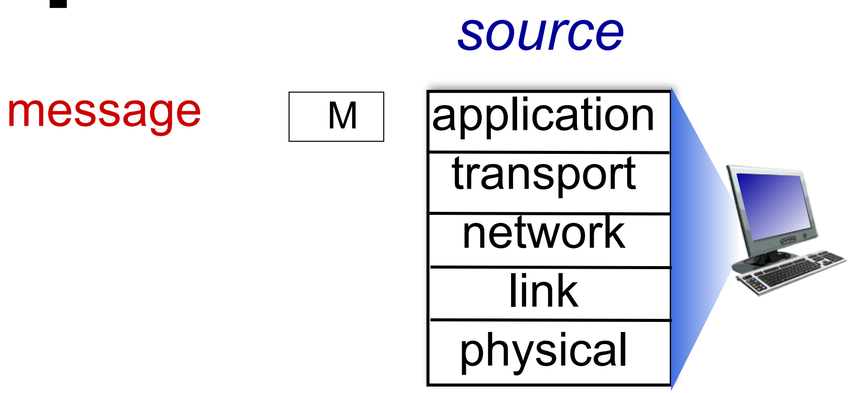
source

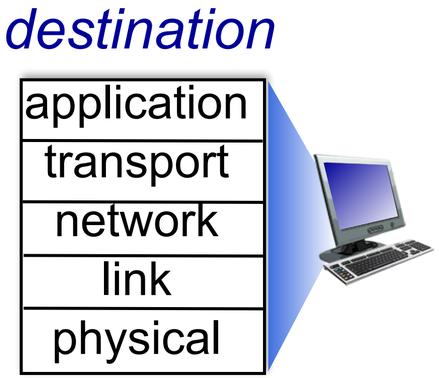


destination

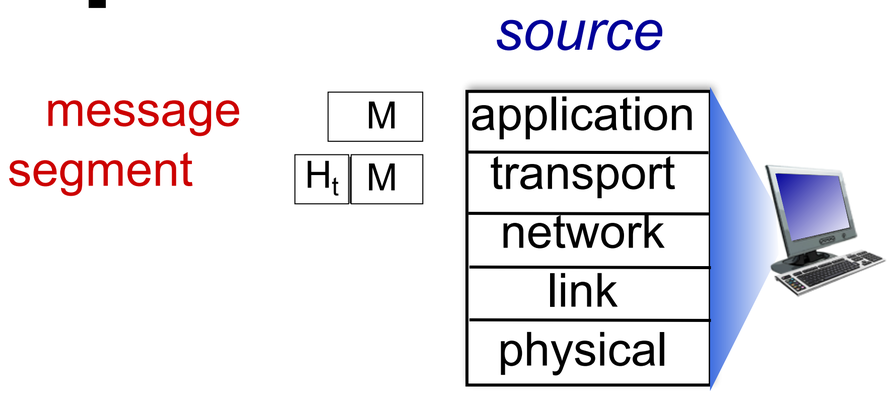


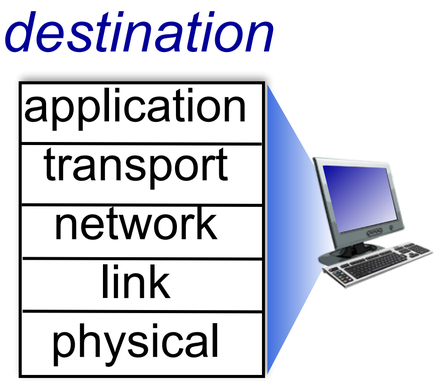
Encapsulation: end-to-end view



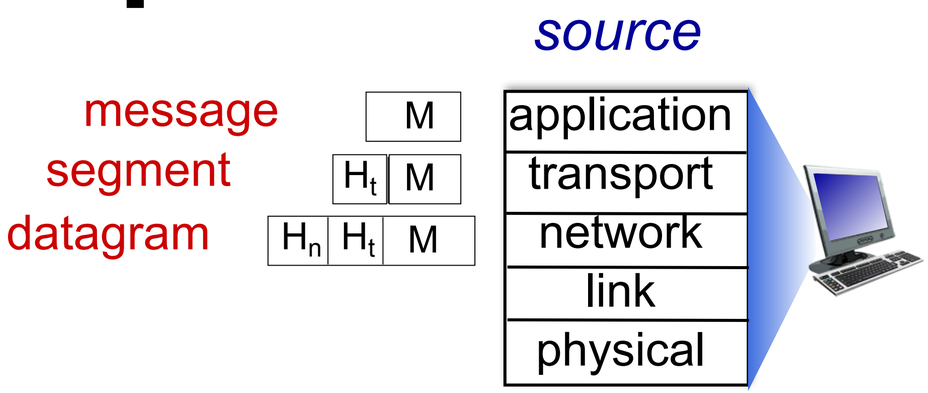


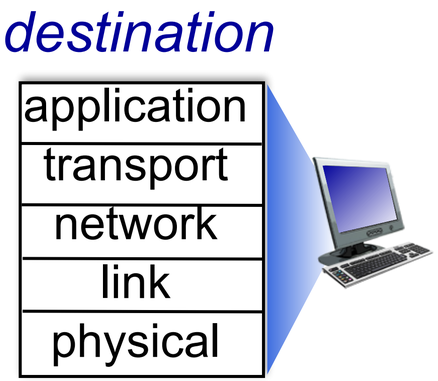
Encapsulation: end-to-end view



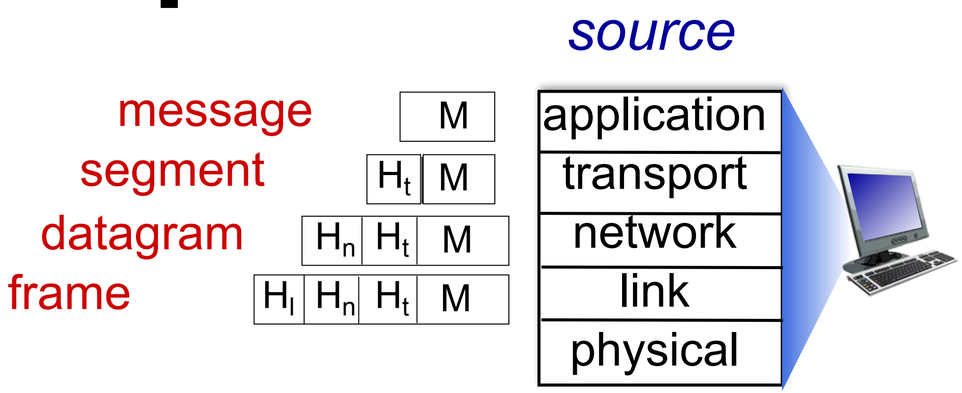


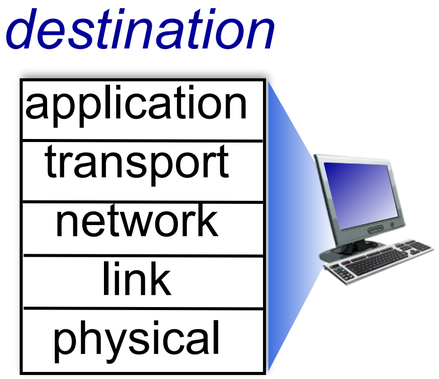
Encapsulation: end-to-end view



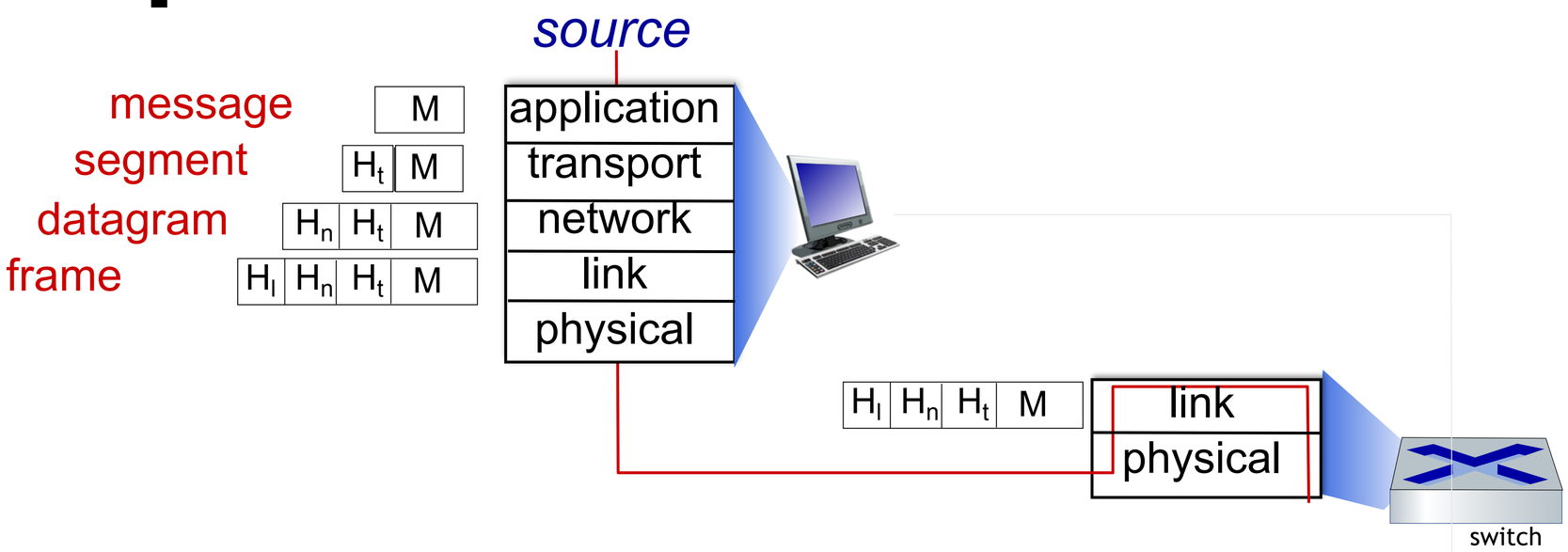


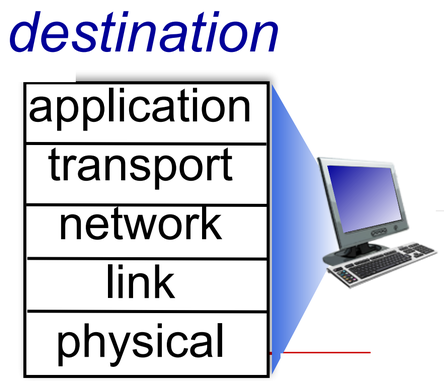
Encapsulation: end-to-end view



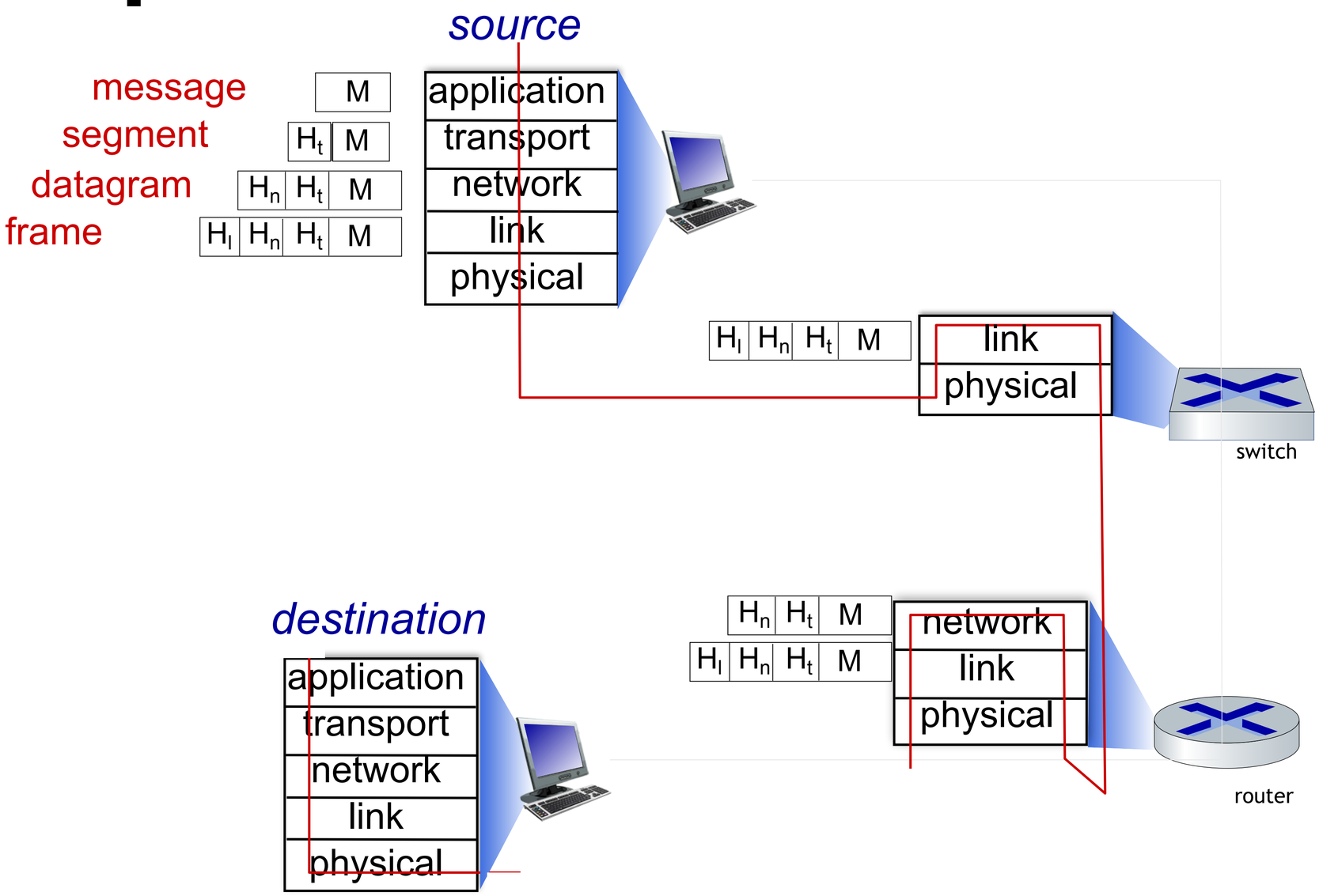


Encapsulation: end-to-end view

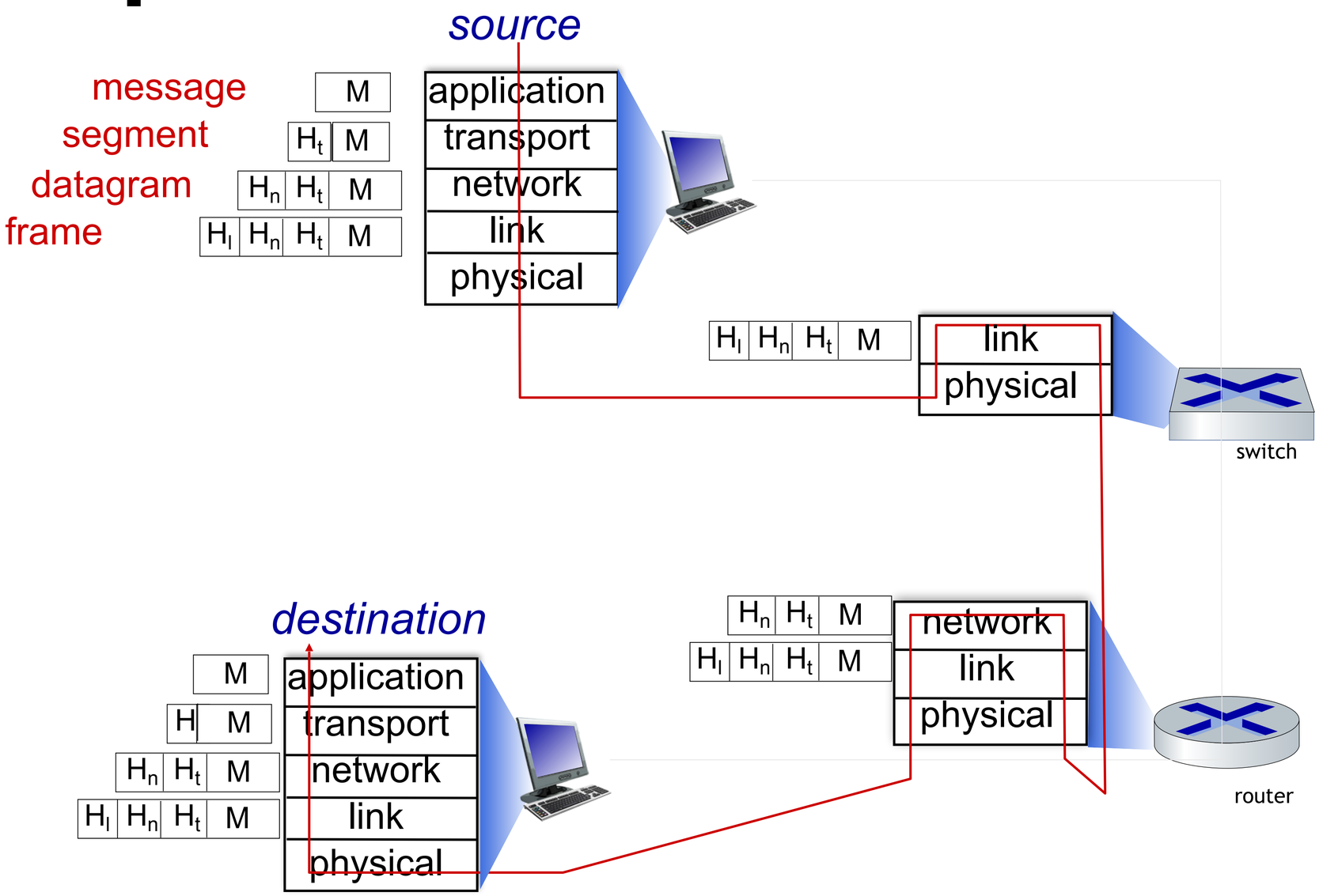




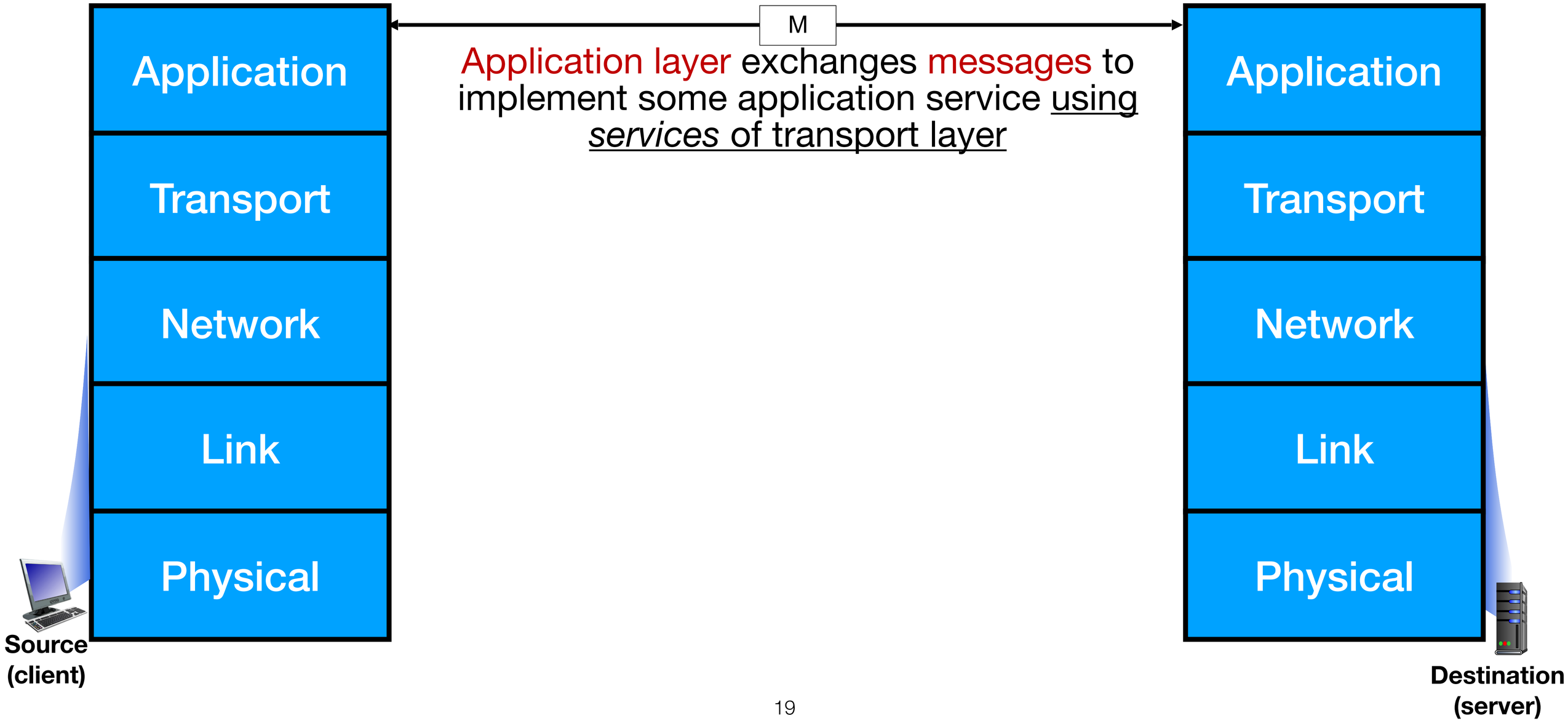
Encapsulation: end-to-end view



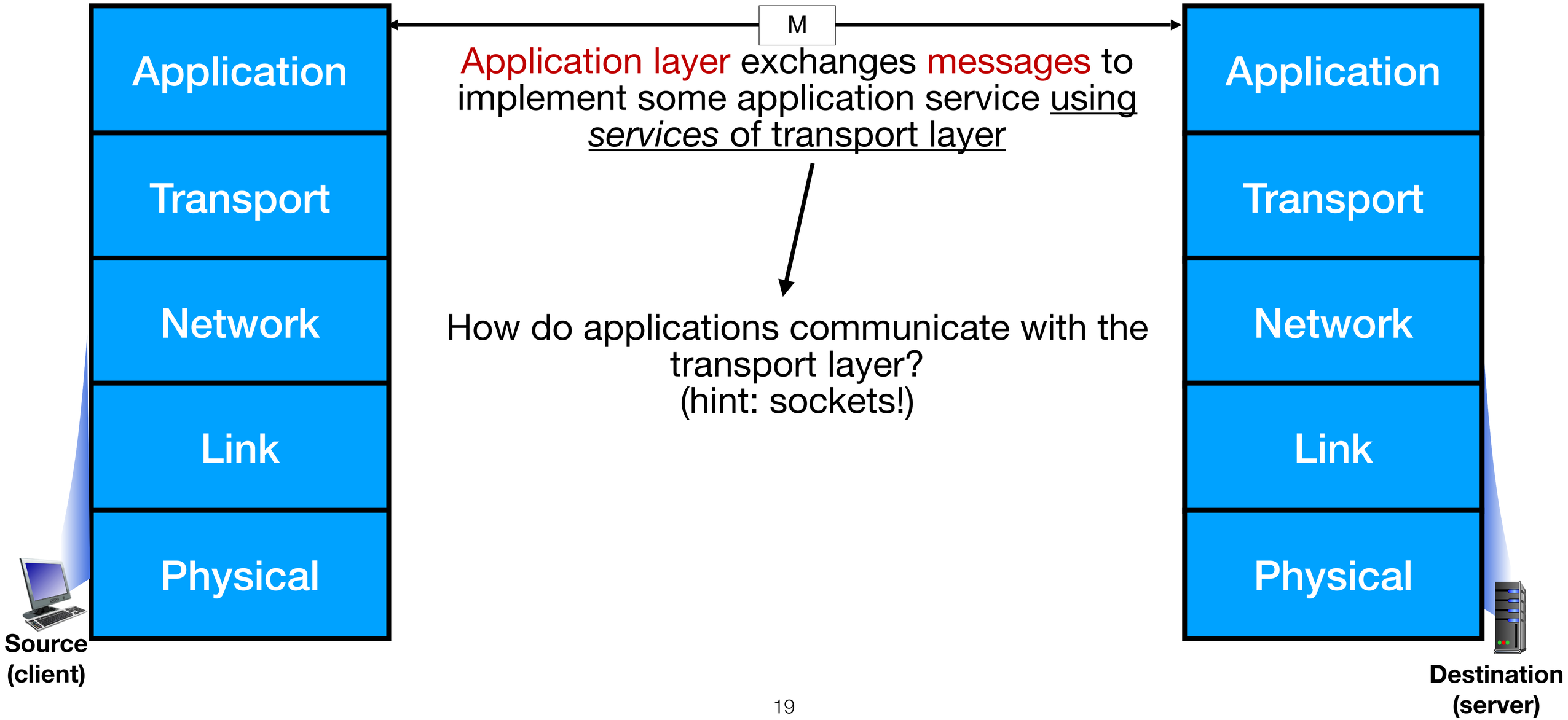
Encapsulation: end-to-end view



Communicating with the transport layer



Communicating with the transport layer



Communicating with the transport layer

The application needs to specify:

- The destination that will receive the data
- What type of transport service it wants
  - Does it need security?
  - Reliability? (e.g., no packets lost)
  - …
- The data that should be sent

Communicating with the transport layer

The application needs to specify:

- The destination that will receive the data
- What type of transport service it wants
  - Does it need security?
  - Reliability? (e.g., no packets lost)
  - …
- The data that should be sent

Communication endpoints are processes

process: program running within a host

- e.g., a web browser
- Processes in different hosts communicate by exchanging **messages** over the network
- The destination of data produced by an application is a process on another host

Communication endpoints are processes

process: program running within a host

- e.g., a web browser
- Processes in different hosts communicate by exchanging **messages** over the network
- The destination of data produced by an application is a process on another host

clients, servers

Client process: initiates communication

Server process: waits to be contacted

Addressing a process

A process must have an identifier to receive messages

Host device has unique IP address

Is the IP address of the host running the process enough to identify the process receiving data? Or do we need more information?

A. Yes - an IP address is enough (why?)

B. No - we need more than an IP address (why?)

Addressing a process

A process must have an identifier to receive messages

Host device has unique IP address

- But this isn't enough!
- There are *many* processing running on the host at any given time

Addressing a process

A process must have an identifier to receive messages

Host device has unique IP address

- But this isn't enough!
- There are *many* processing running on the host at any given time

An identifier includes both *IP address* and *port numbers* associated with host process

Example port numbers:

- HTTP: 80
- HTTPS: 443
- SSH: 22
- To send HTTP message to cs.oberlin.edu web server:
  - IP address: 132.162.201.24
  - Port number: 80

Communicating with the transport layer

The application needs to specify:

- The destination that will receive the data
- What type of transport service it wants
  - Does it need security?
  - Reliability? (e.g., no packets lost)
  - …
- The data that should be sent

Communicating with the transport layer

The application needs to specify:

- The destination that will receive the data
- What type of transport service it wants
  - Does it need security?
  - Reliability? (e.g., no packets lost)
  - …
- The data that should be sent

Communicating with the transport layer

The application needs to specify:

- The destination that will receive the data
- What type of transport service it wants
  - Does it need security?
  - Reliability? (e.g., no packets lost)
  - …
- The data that should be sent

We have a choice of 2 transport-layer protocols:

- TCP
- UDP

The best choice depends on the services the application needs

TCP vs UDP

TCP: Transmission Control Protocol

TCP guarantees reliability

- All messages will get sent to the application, in order
- If a message gets lost, TCP will retransmit the message until it's received

TCP makes sure it doesn’t overwhelm receiver by sending too much, too quickly

TCP vs UDP

TCP: Transmission Control Protocol

TCP guarantees reliability

- All messages will get sent to the application, in order
- If a message gets lost, TCP will retransmit the message until it's received

TCP makes sure it doesn’t overwhelm receiver by sending too much, too quickly

UDP: User Datagram Protocol

UDP does NOT guarantee reliability

- Messages may be lost or arrive out-of-order

Because UDP doesn’t have to worry about reliability, it is much faster

For each of the following applications, choose whether you would use TCP or UDP, and justify why you would choose it. [Select any letter on your clicker]

- Online gaming
- SSH remote access
- Email
- Video conferencing
- Whatsapp

Communicating with the transport layer

The application needs to specify:

- The destination that will receive the data → IP address + port number
- What type of transport service it wants → TCP or UDP
  - Does it need security?
  - Reliability? (e.g., no packets lost)
  - …
- The data that should be sent

The most common interface to the transport layer is the **socket** interface

Sockets

Sockets

Process sends/receives messages to/from its socket

- Not unlike communicating between threads!

Sockets

Process sends/receives messages to/from its socket

- Not unlike communicating between threads!

Sockets are like a door

- Sending process shoves message out the door
- Sender relies on transport infrastructure at receiver door to deliver the message to socket at receiving process

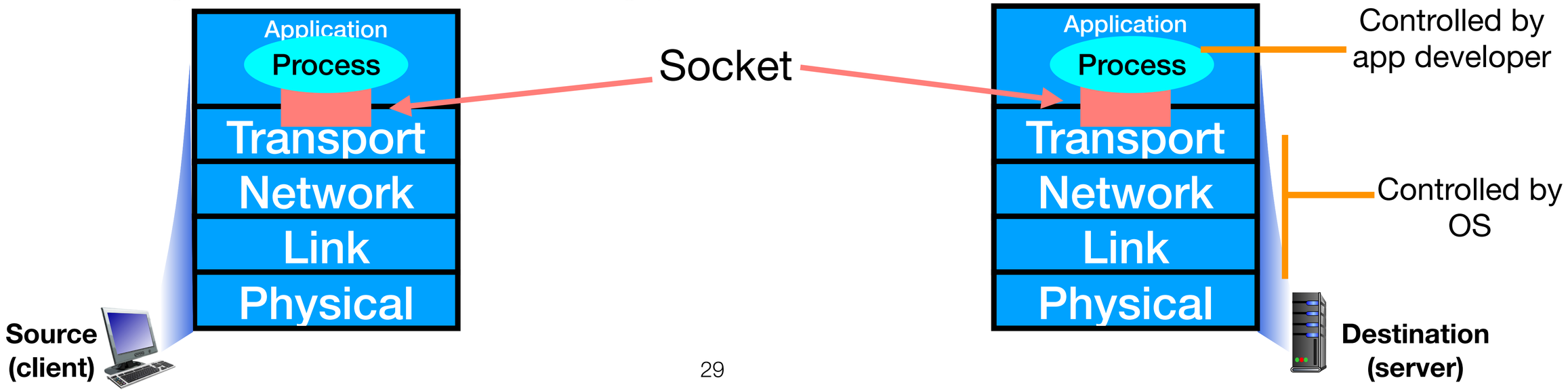
Sockets

Process sends/receives messages to/from its socket

- Not unlike communicating between threads!

Sockets are like a door

- Sending process shoves message out the door
- Sender relies on transport infrastructure at receiver door to deliver the message to socket at receiving process



Socket Programming

Goal: build client/server applications that communicate using sockets

Two types of sockets

- TCP socket (stream)
- UDP socket (datagram)