

**CS 168**  
Lecture 4

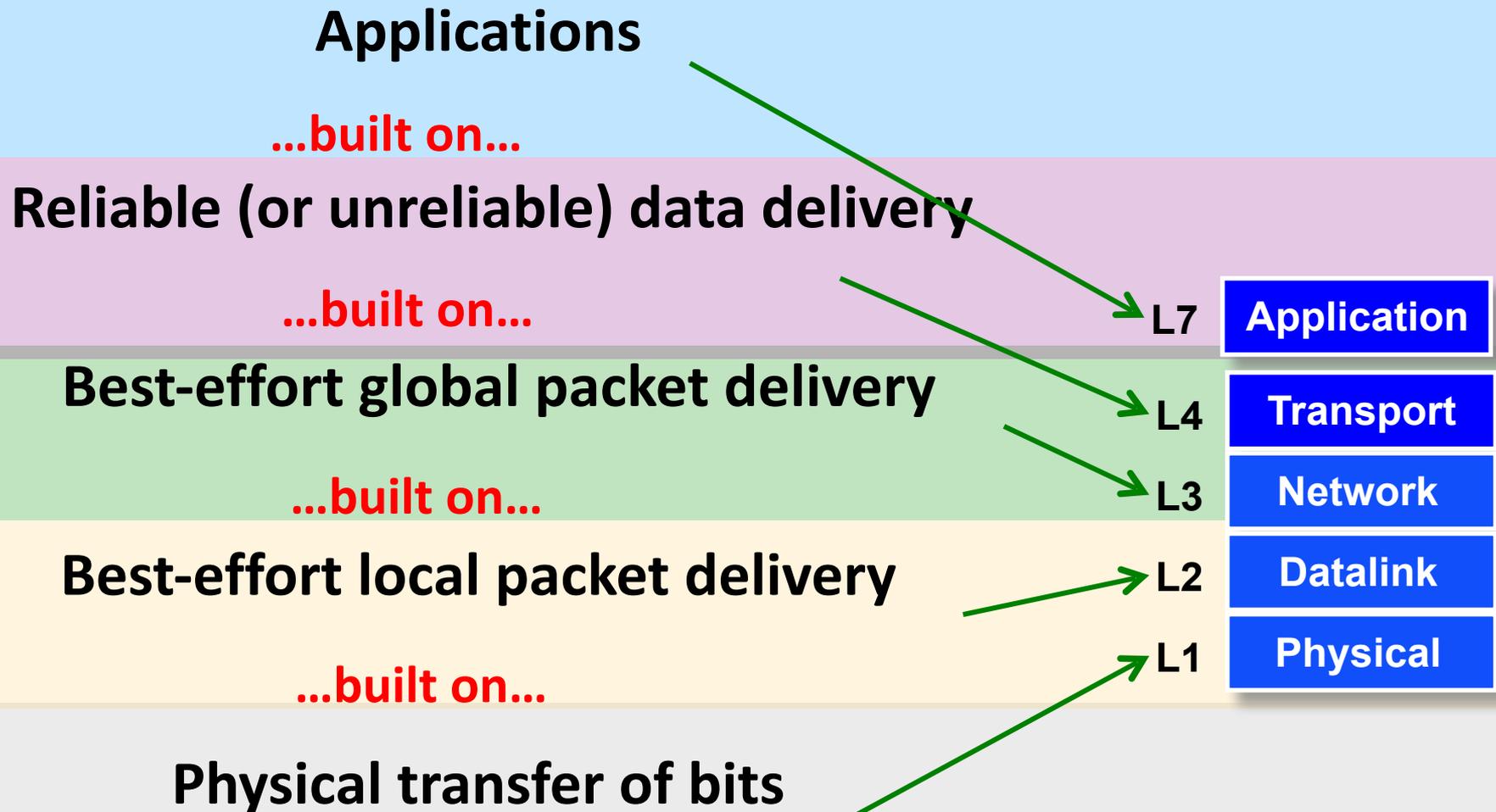
**Designing the Internet**

Sylvia Ratnasamy  
Spring 2024

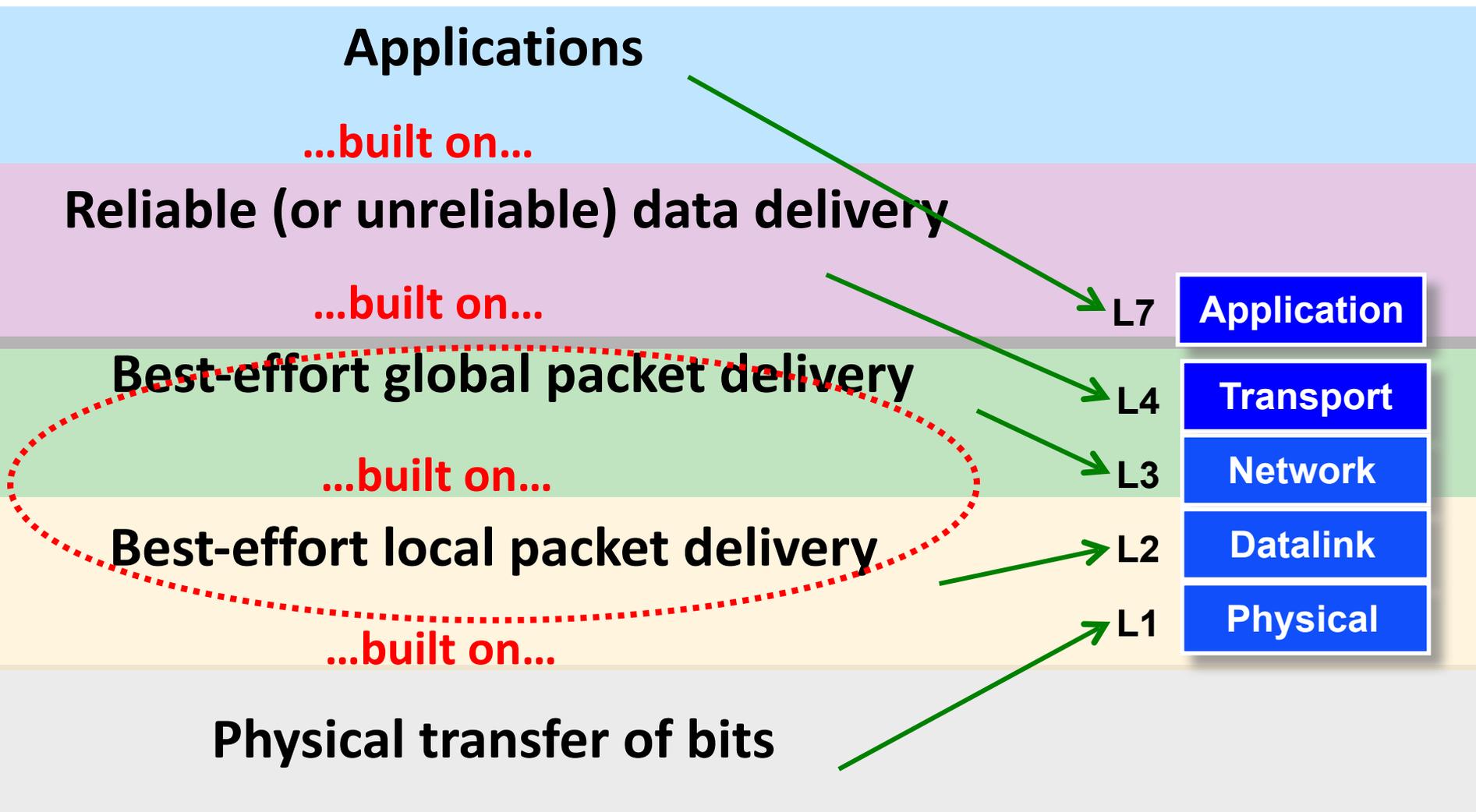
# Recall: How do you solve a problem?

1. **Decompose** it (into tasks and abstractions)
2. **Assign** tasks to entities (who does what)

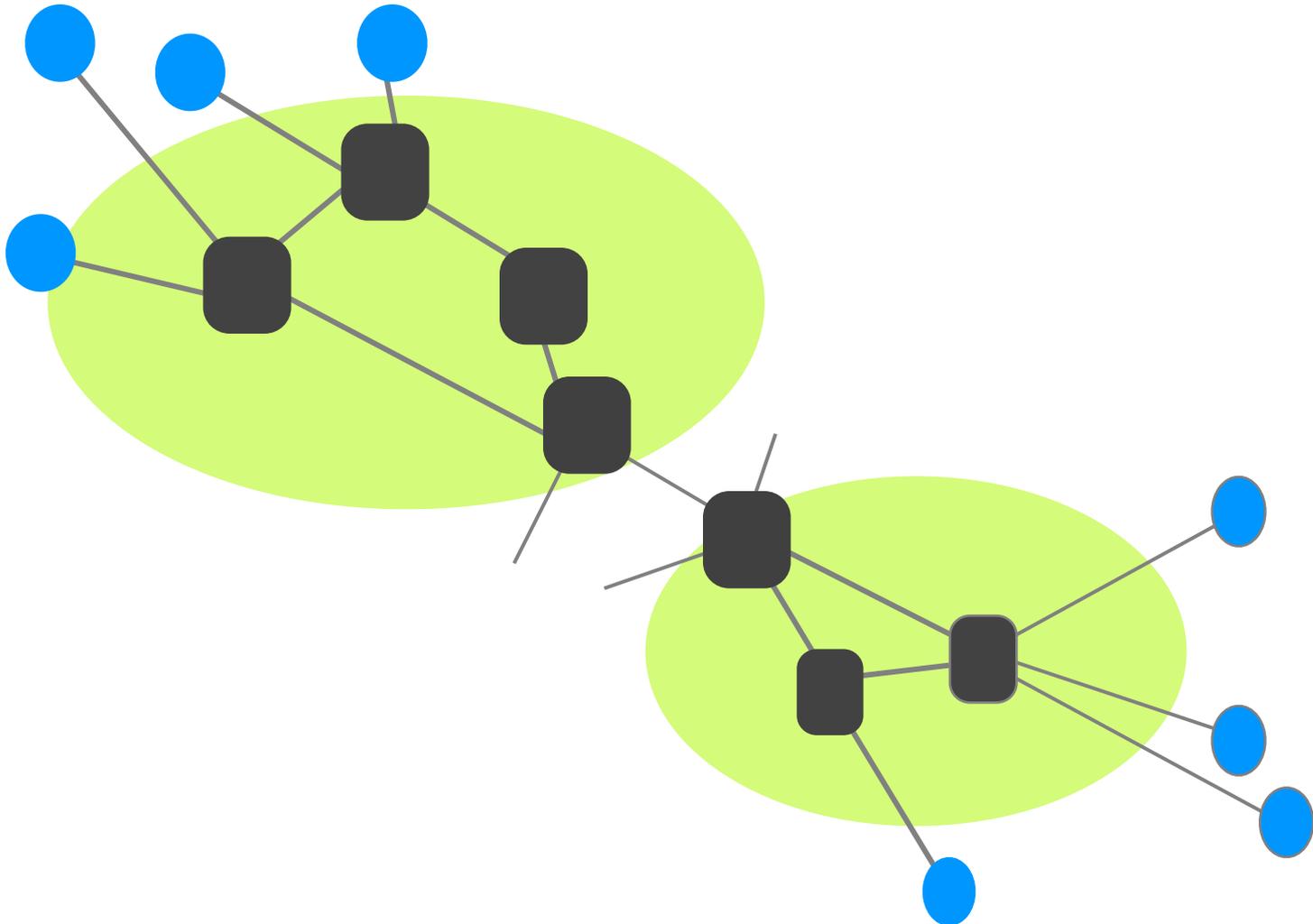
# Recall: a layered architecture



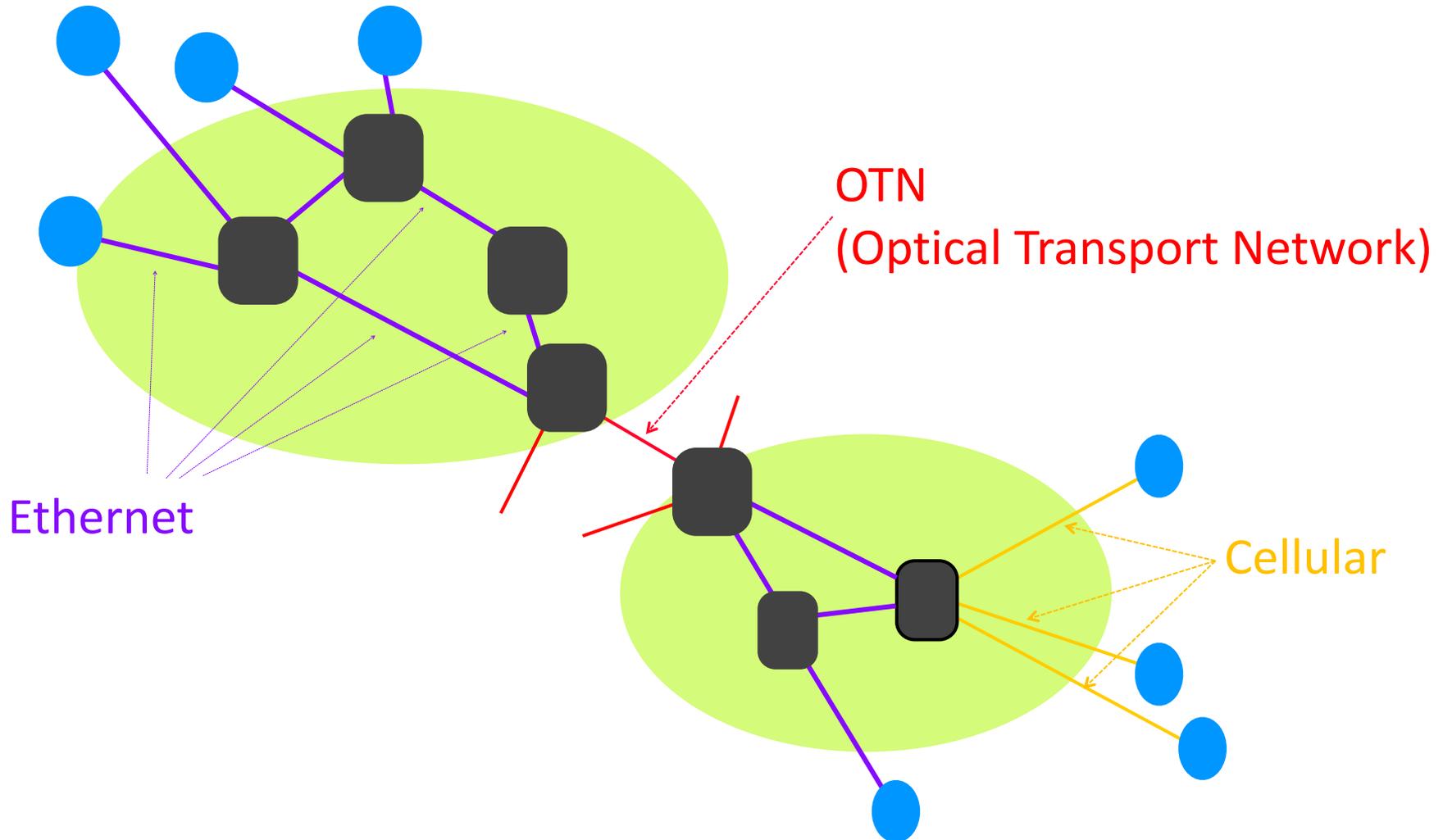
# Recall: a layered architecture



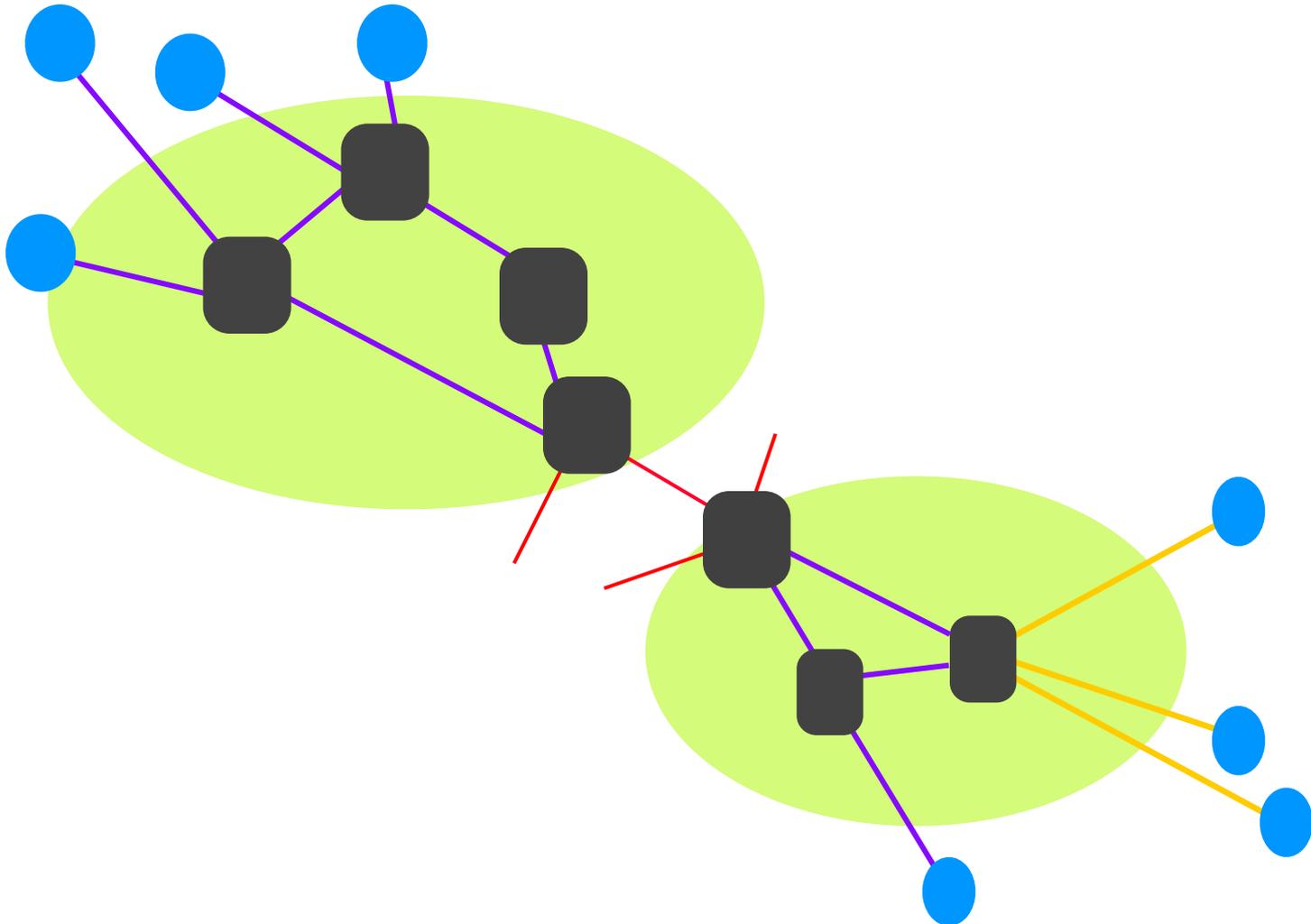
# Local vs. Global Delivery



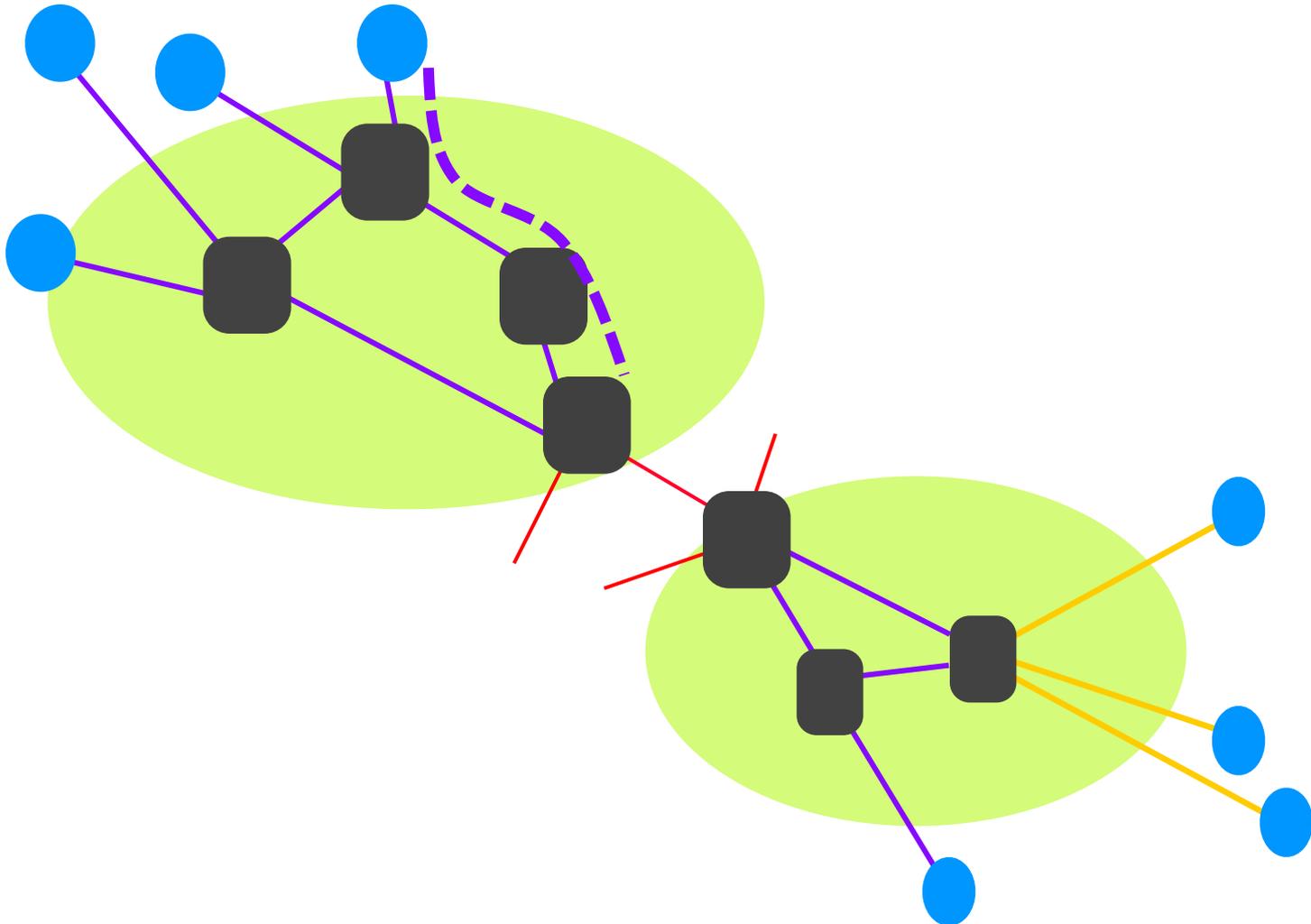
# Local vs. Global Delivery



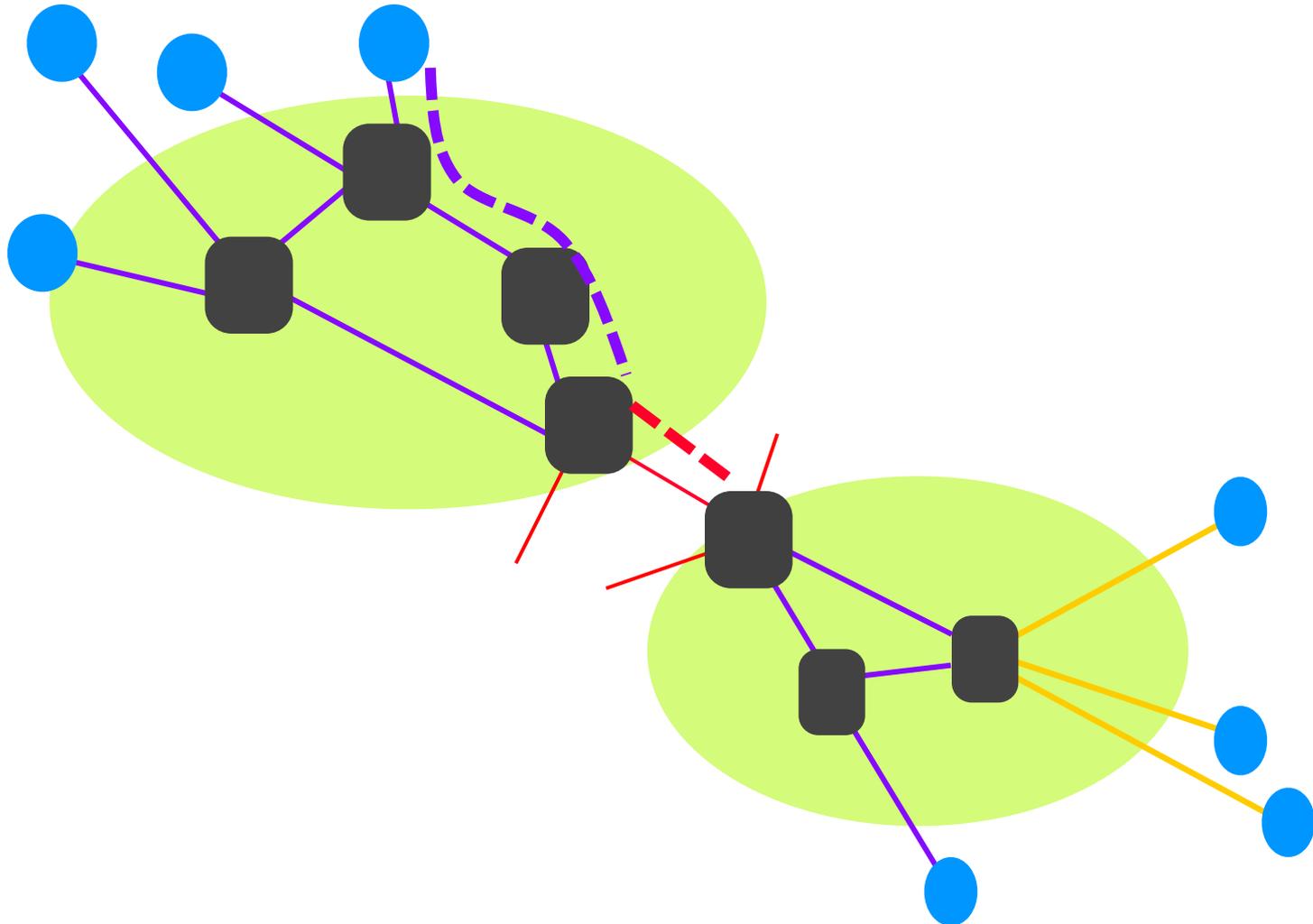
# Local vs. Global Delivery



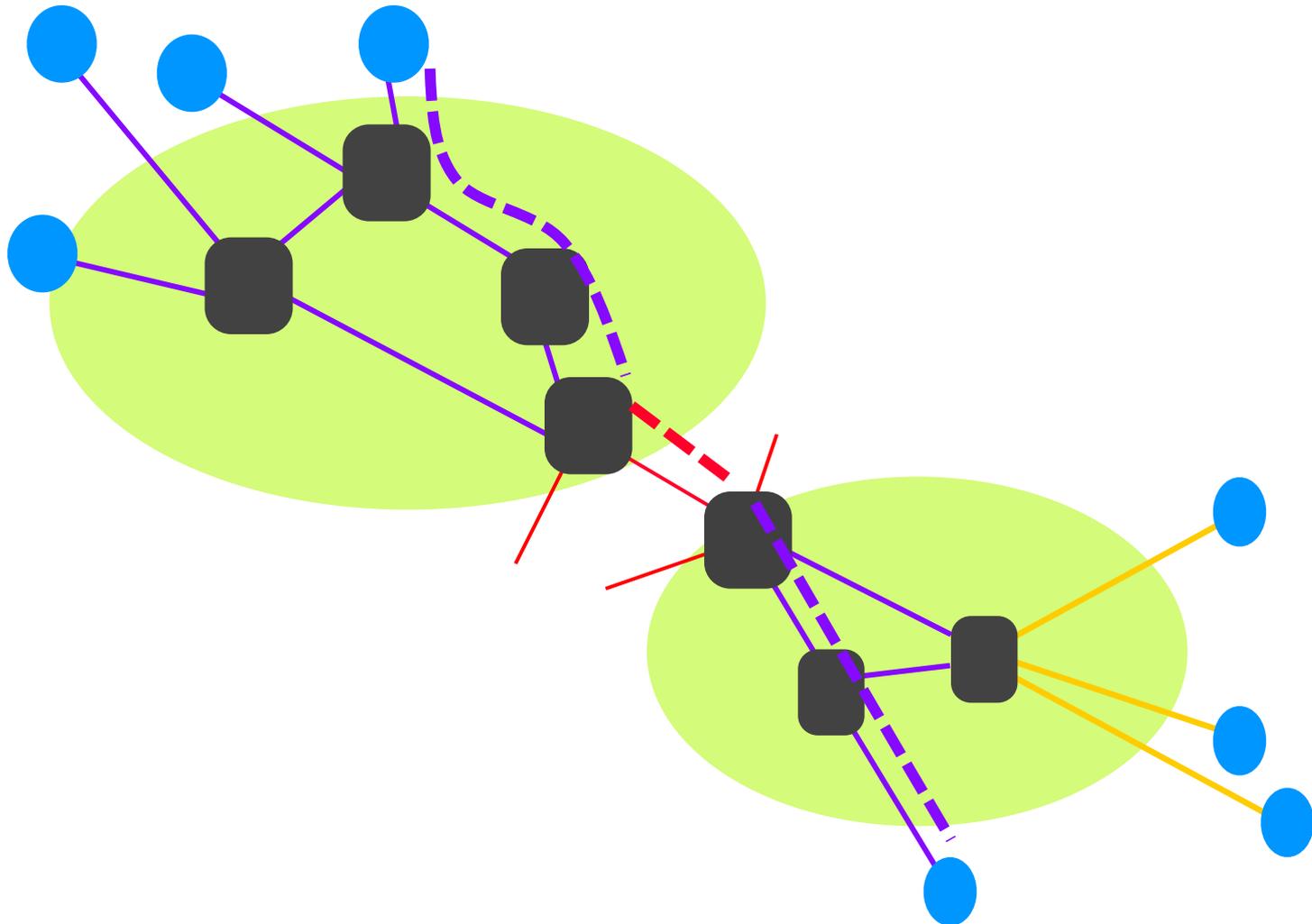
# Local vs. Global Delivery



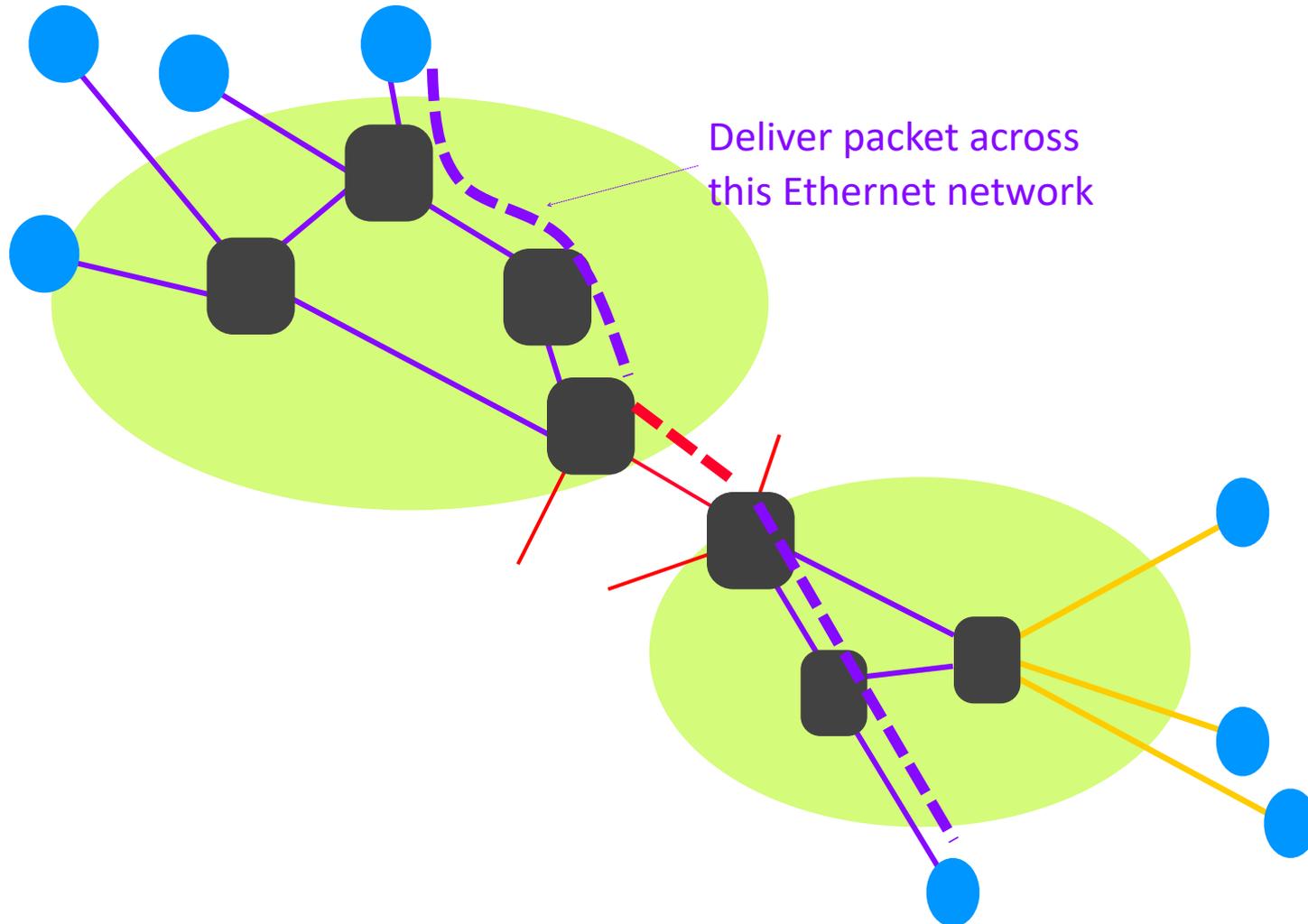
# Local vs. Global Delivery



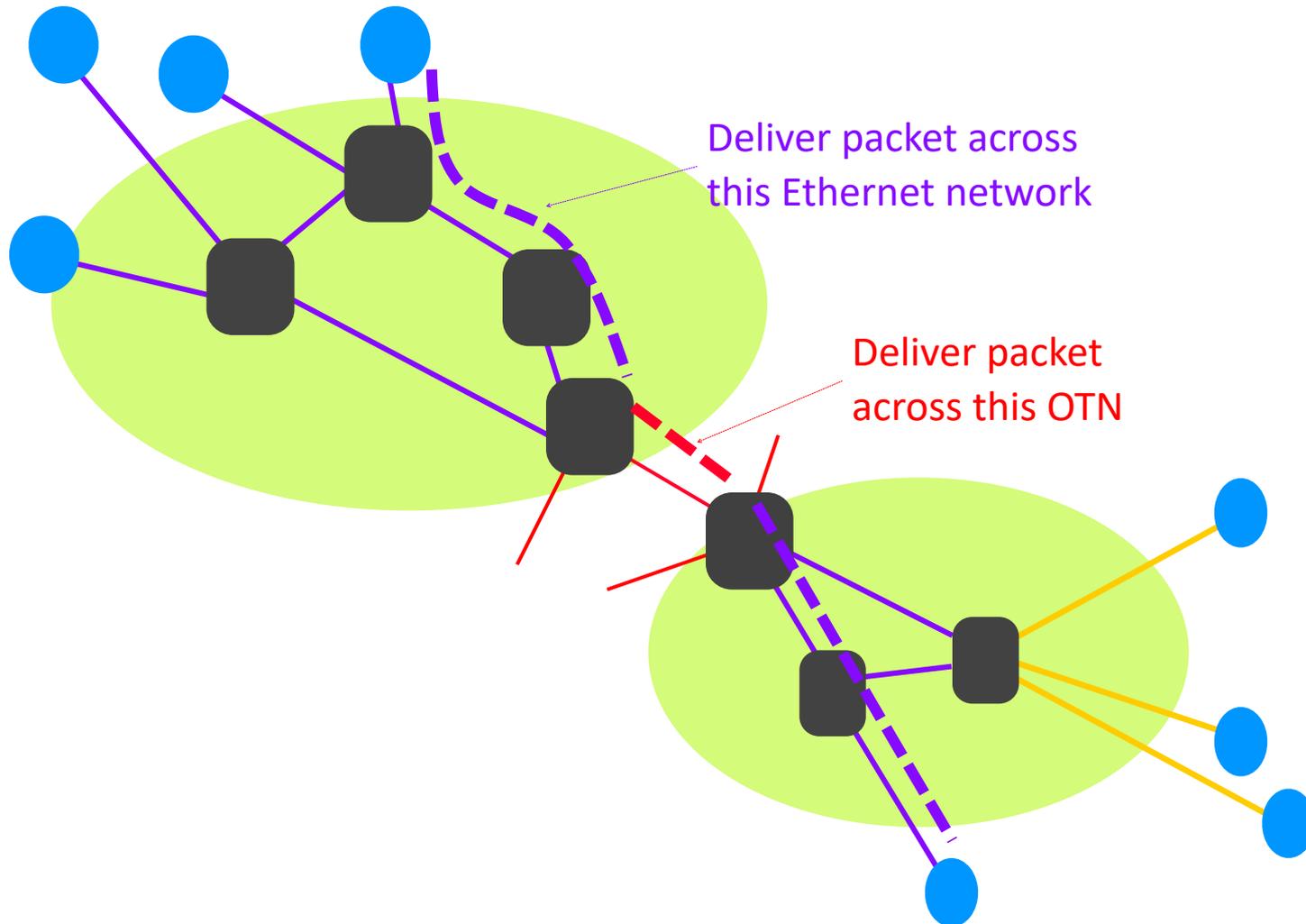
# Local vs. Global Delivery



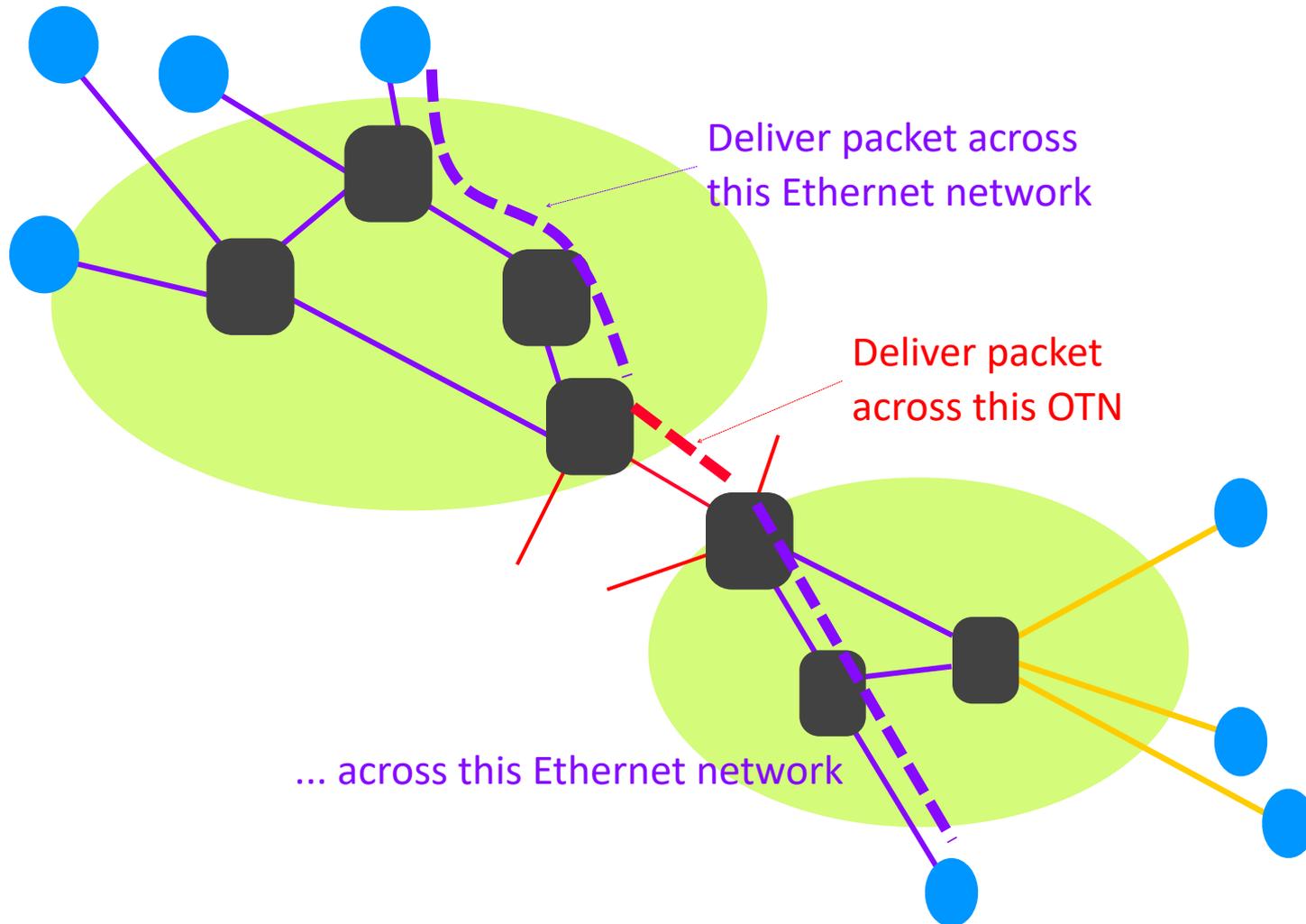
# Local vs. Global Delivery



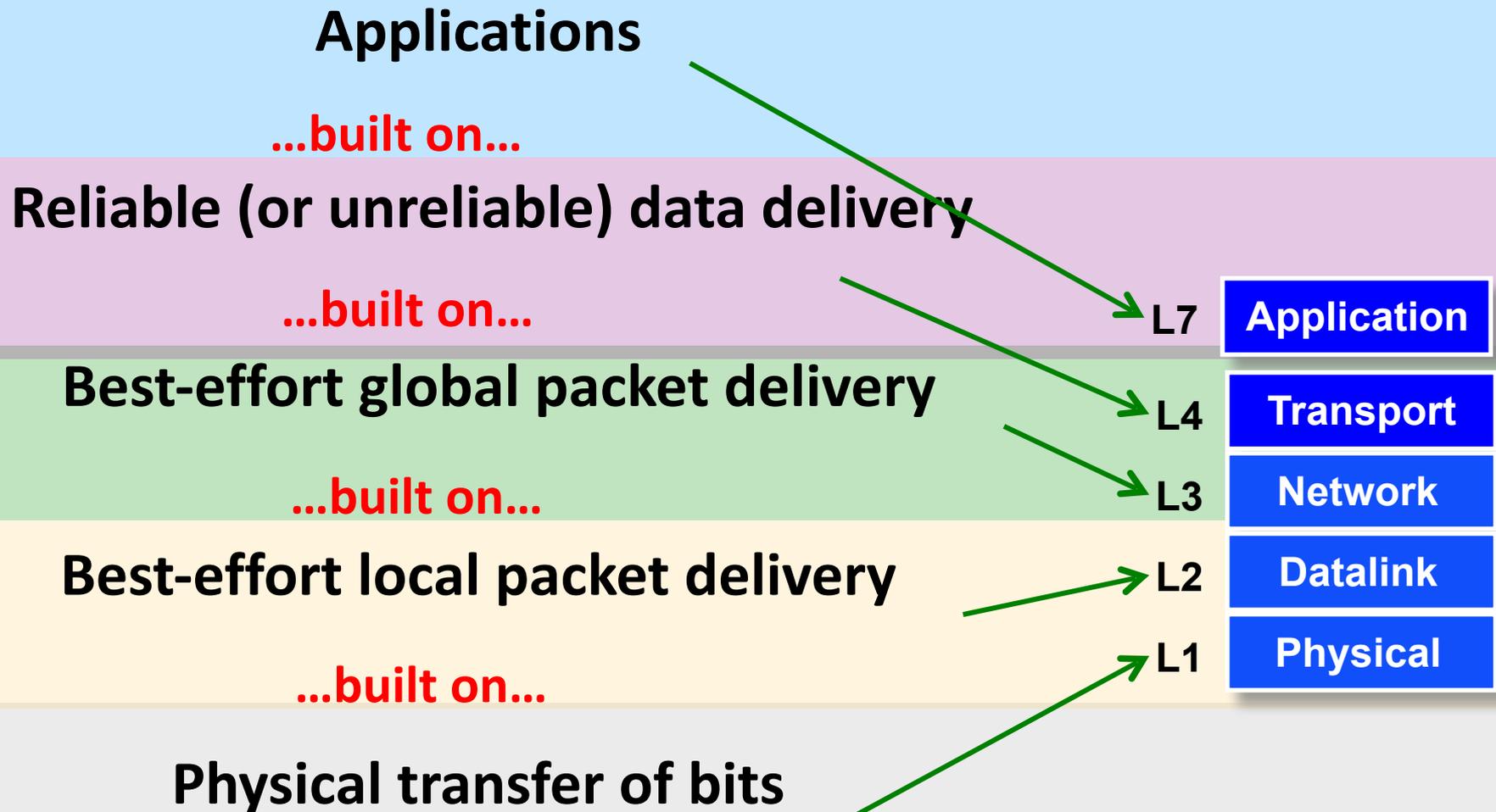
# Local vs. Global Delivery



# Local vs. Global Delivery



# Recall: a layered architecture



**Questions?**

# Recall: peers understand the same things

**CEO**

**Letter**

**CEO**

**Aide**

**Envelope**

**Aide**

**FedEx**

**Fedex Envelope (FE)**

**FedEx**

# Protocols and Layers



Communication between peer layers on different systems is defined by **protocols**

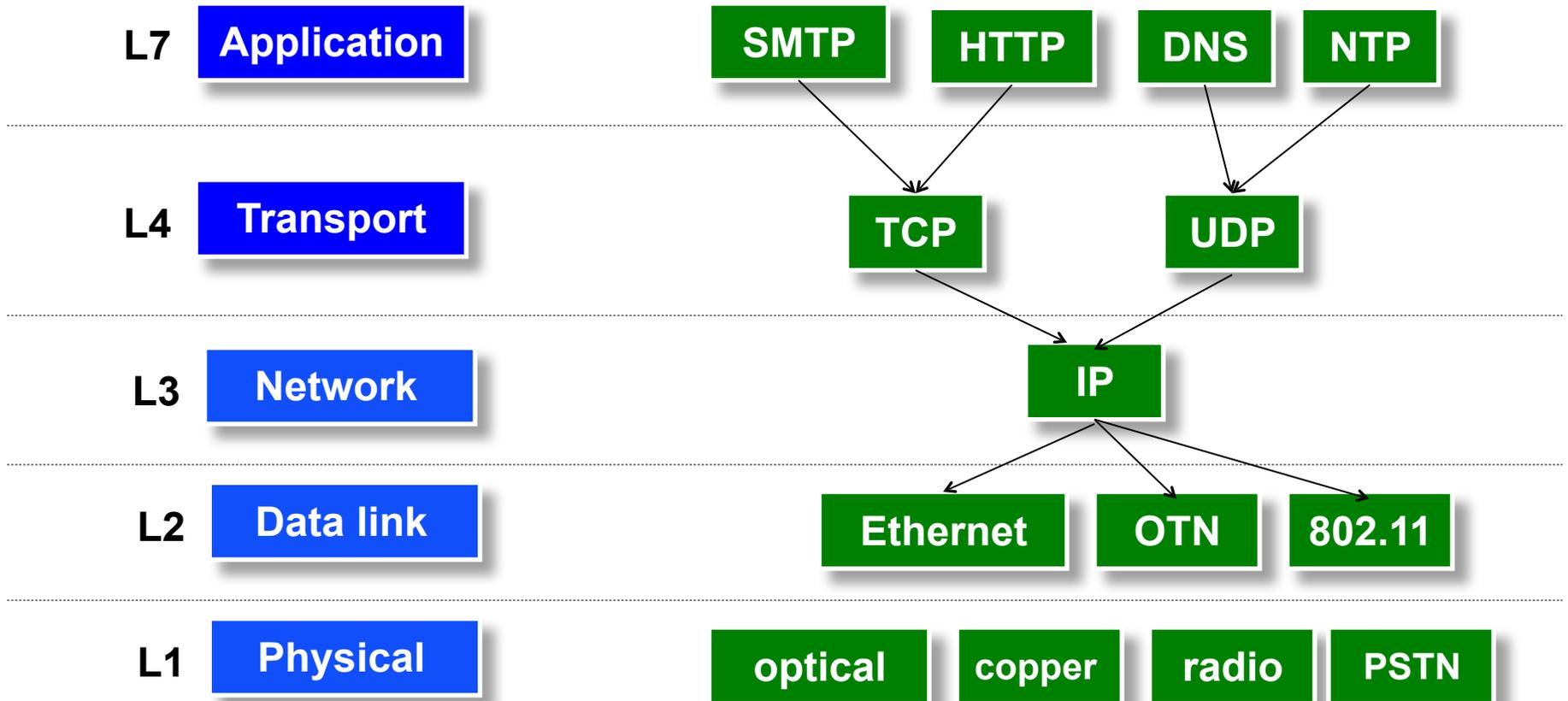
# What is a Protocol?

- An agreement between parties on how to communicate
- Defines the syntax of communication
- And semantics
  - “first a hullo, then a request...”
  - essentially, a state machine
  - we’ll study many protocols later in the semester

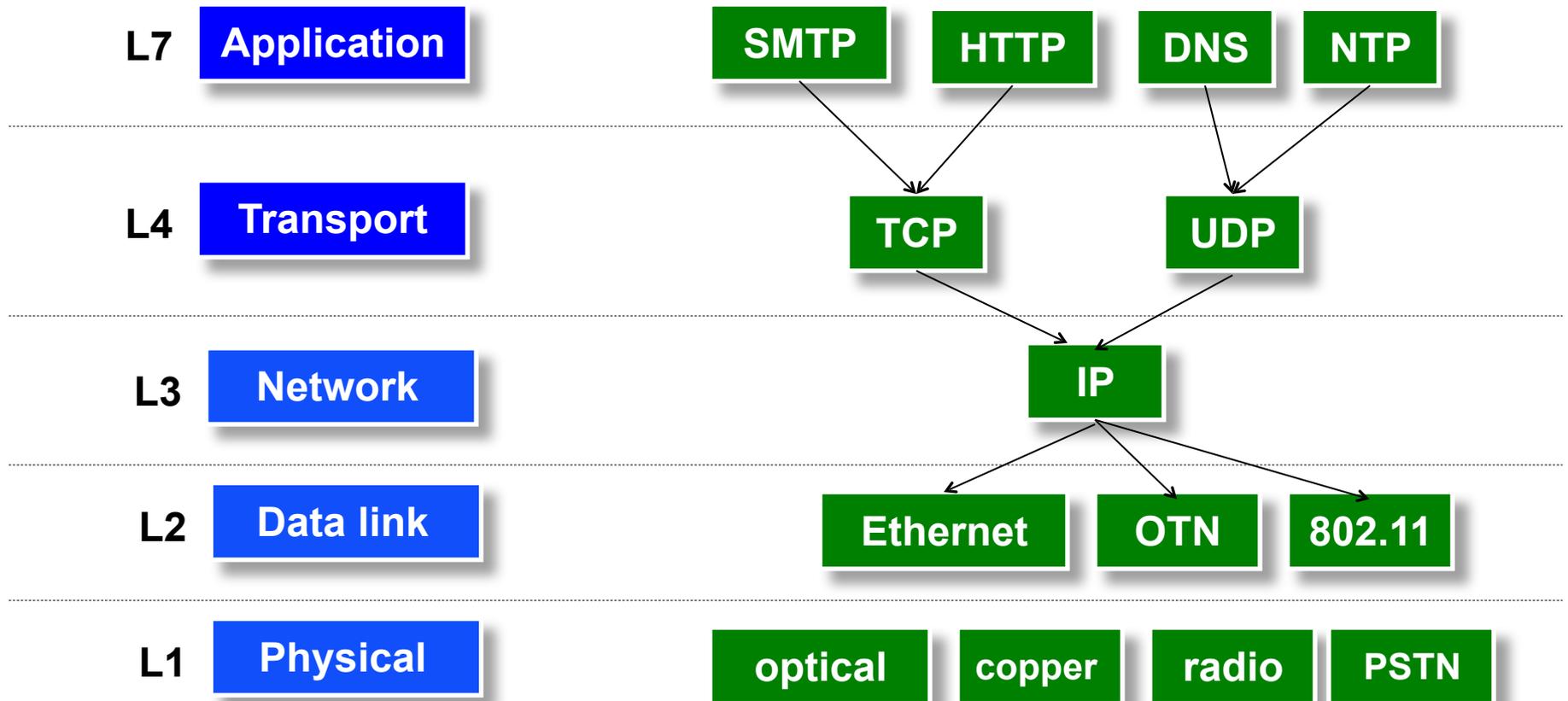
# What is a Protocol?

- An agreement between parties on how to communicate
- Defines the syntax of communication
- And semantics
  - “first a hullo, then a request...”
  - essentially, a state machine
  - we’ll study many protocols later in the semester
- Protocols exist at many levels
  - defined by a variety of standards bodies (IETF, IEEE, ITU)

# Protocols at different layers

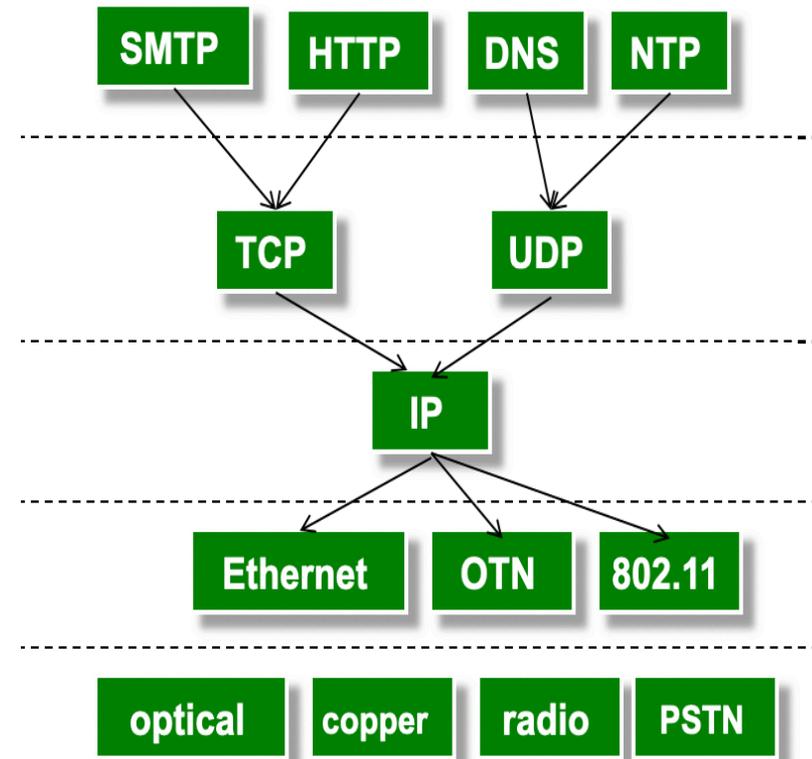


# Protocols at different layers



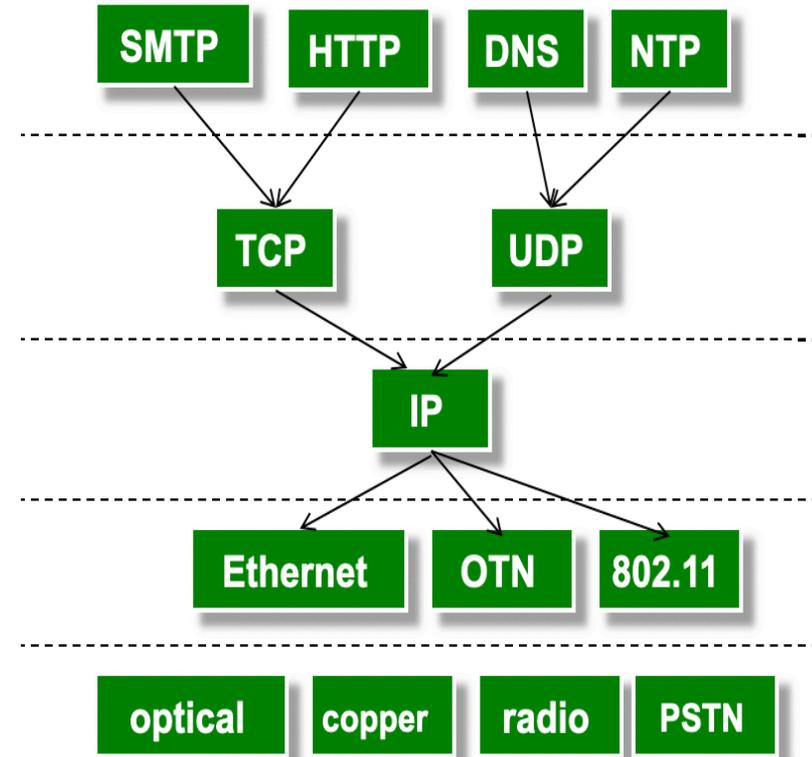
**There is just one network-layer protocol!**

# Recap: Three important properties



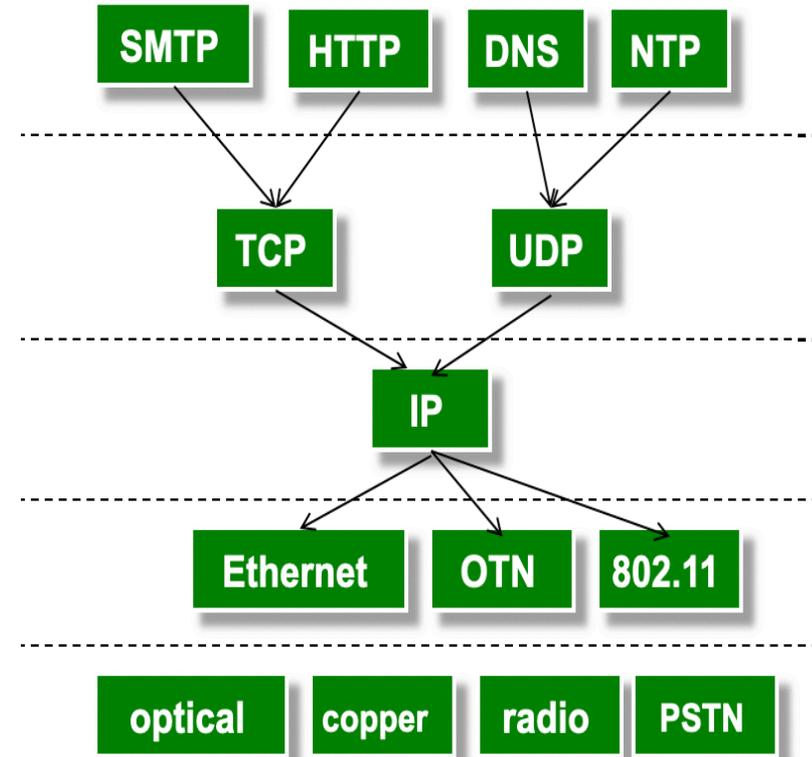
# Recap: Three important properties

- Each layer:
  - Depends on layer below
  - Supports layer above
  - Independent of others



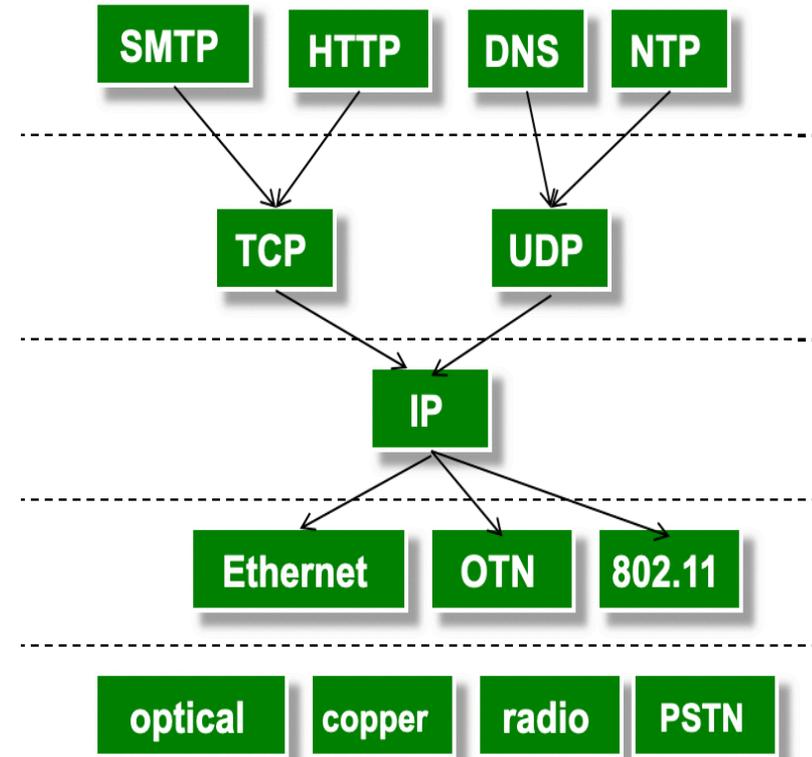
# Recap: Three important properties

- Each layer:
  - Depends on layer below
  - Supports layer above
  - Independent of others
- Multiple versions in a layer
  - Interfaces differ somewhat
  - Components at one layer pick which lower-level protocol to use

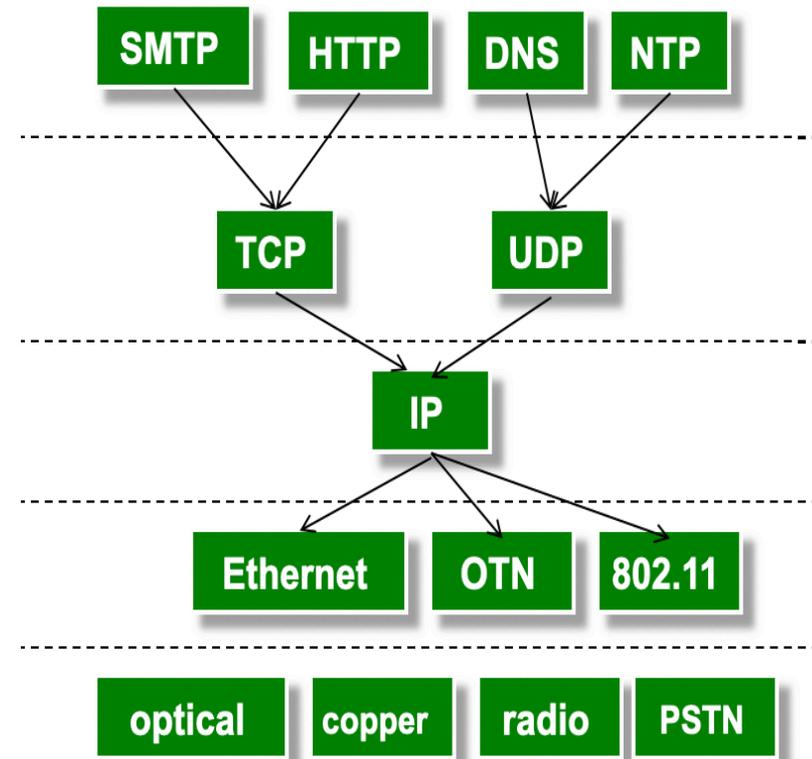


# Recap: Three important properties

- Each layer:
  - Depends on layer below
  - Supports layer above
  - Independent of others
- Multiple versions in a layer
  - Interfaces differ somewhat
  - Components at one layer pick which lower-level protocol to use
- But only one IP layer
  - Unifying protocol

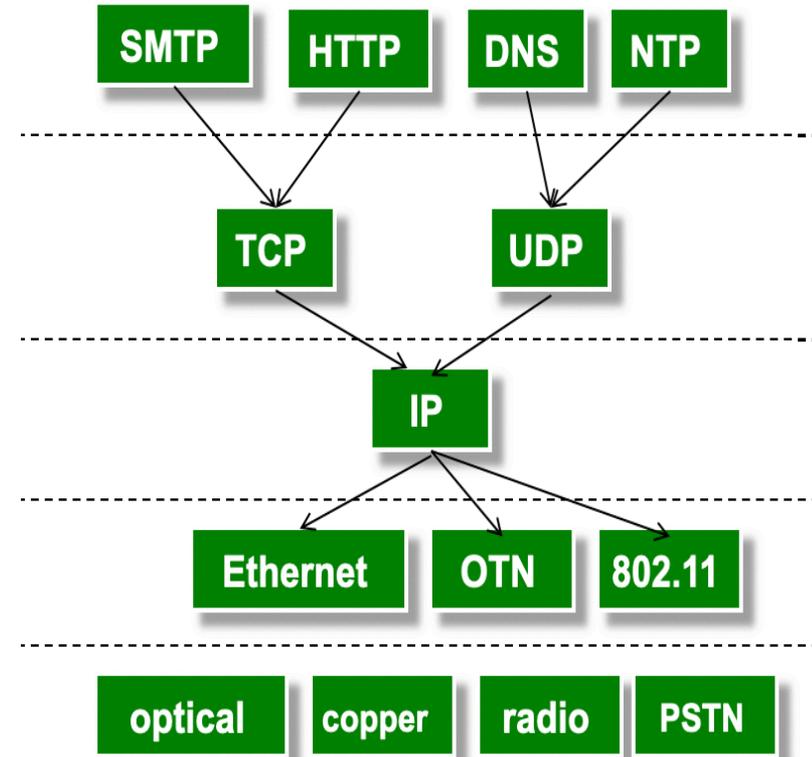


# Why is layering important?



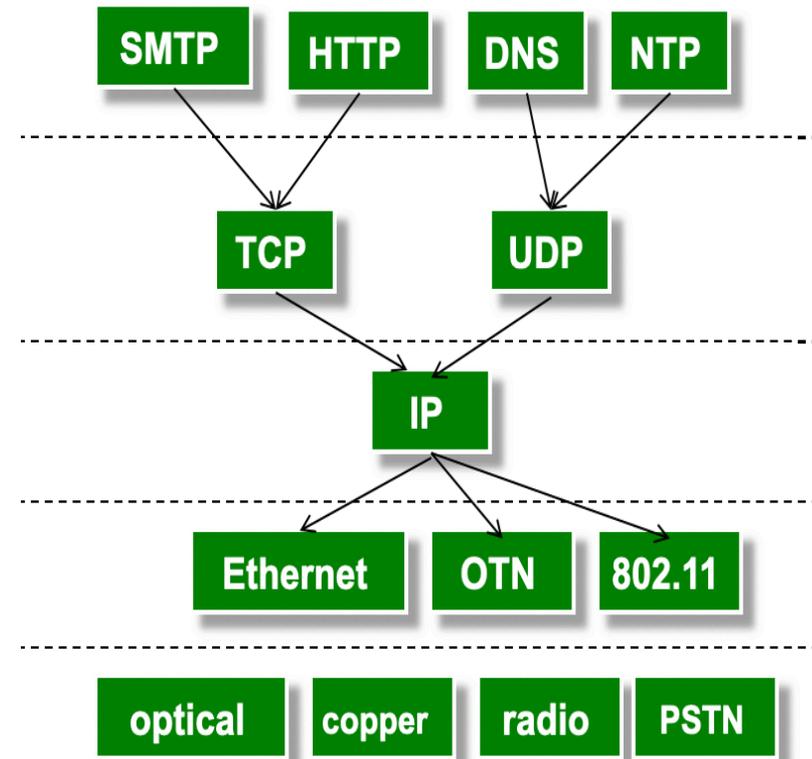
# Why is layering important?

- Innovation can proceed largely in parallel!



# Why is layering important?

- Innovation can proceed largely in parallel!
- Pursued by very different communities
  - App devs (L7), chip designers (L1/L2)



**Questions?**

# How do you solve a problem?

1. Decompose it (into tasks and abstractions)
2. **Assign** tasks to entities (who does what)

# Distributing Layers Across Network

# Distributing Layers Across Network

- Layers are simple if only on a single machine
  - Just stack of modules interacting with those above/below

# Distributing Layers Across Network

- Layers are simple if only on a single machine
  - Just stack of modules interacting with those above/below
- But we need to implement layers across:
  - Hosts
  - Routers (switches)

# Distributing Layers Across Network

- Layers are simple if only on a single machine
  - Just stack of modules interacting with those above/below
- But we need to implement layers across:
  - Hosts
  - Routers (switches)
- What gets implemented where?

**What gets implemented  
at the end host?**

# What gets implemented at the end host?

- Bits arrive on wire ... → must implement L1

# What gets implemented at the end host?

- Bits arrive on wire ... → must implement L1
- ... must make it up to app → must implement L7

# What gets implemented at the end host?

- Bits arrive on wire ... → must implement L1
- ... must make it up to app → must implement L7
- Therefore, all layers must exist at host!

# What gets implemented in the network?

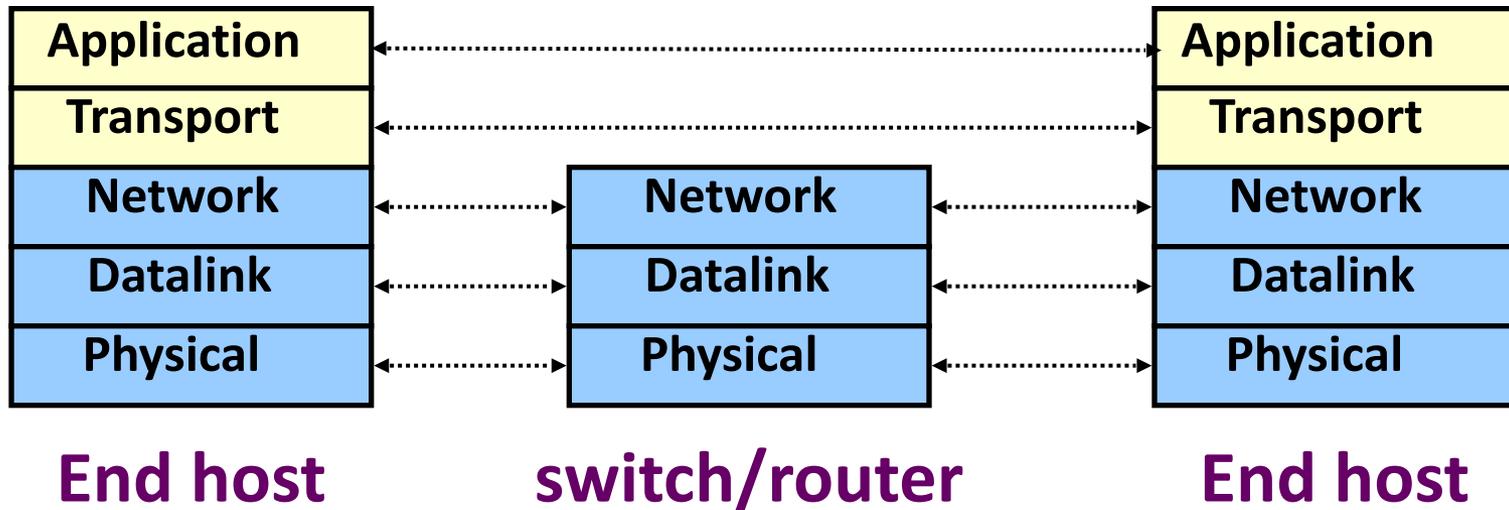
- Bits arrive on wire → physical layer (L1)
- Packets must be delivered across links and local networks → datalink layer (L2)
- Packets must be delivered between networks for global delivery → network layer (L3)

# What gets implemented in the network?

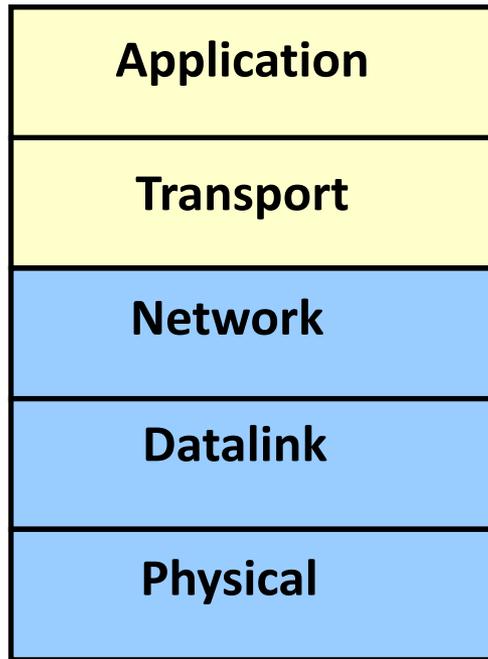
- Bits arrive on wire → physical layer (L1)
- Packets must be delivered across links and local networks → datalink layer (L2)
- Packets must be delivered between networks for global delivery → network layer (L3)
- The network does not support reliable delivery
  - Transport layer (and above) **not** supported

# Simple Diagram

- Lower three layers implemented everywhere
- Top two layers implemented only at hosts



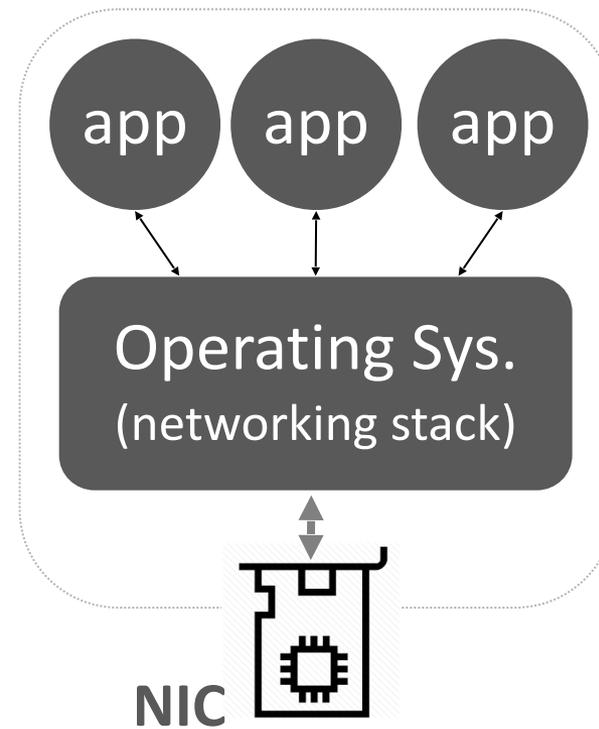
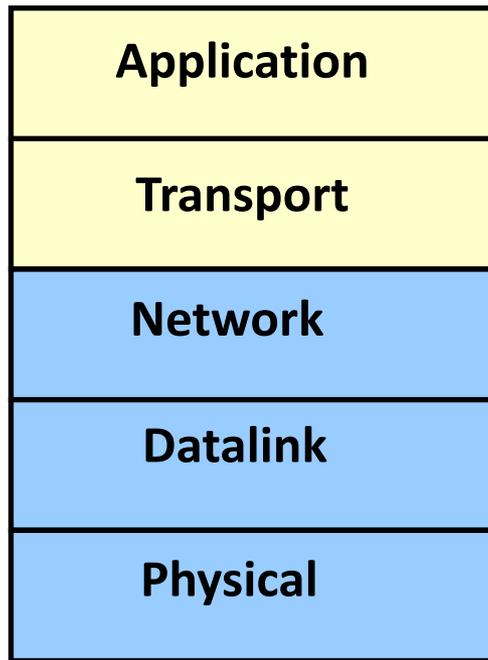
# A closer look: end host



end host

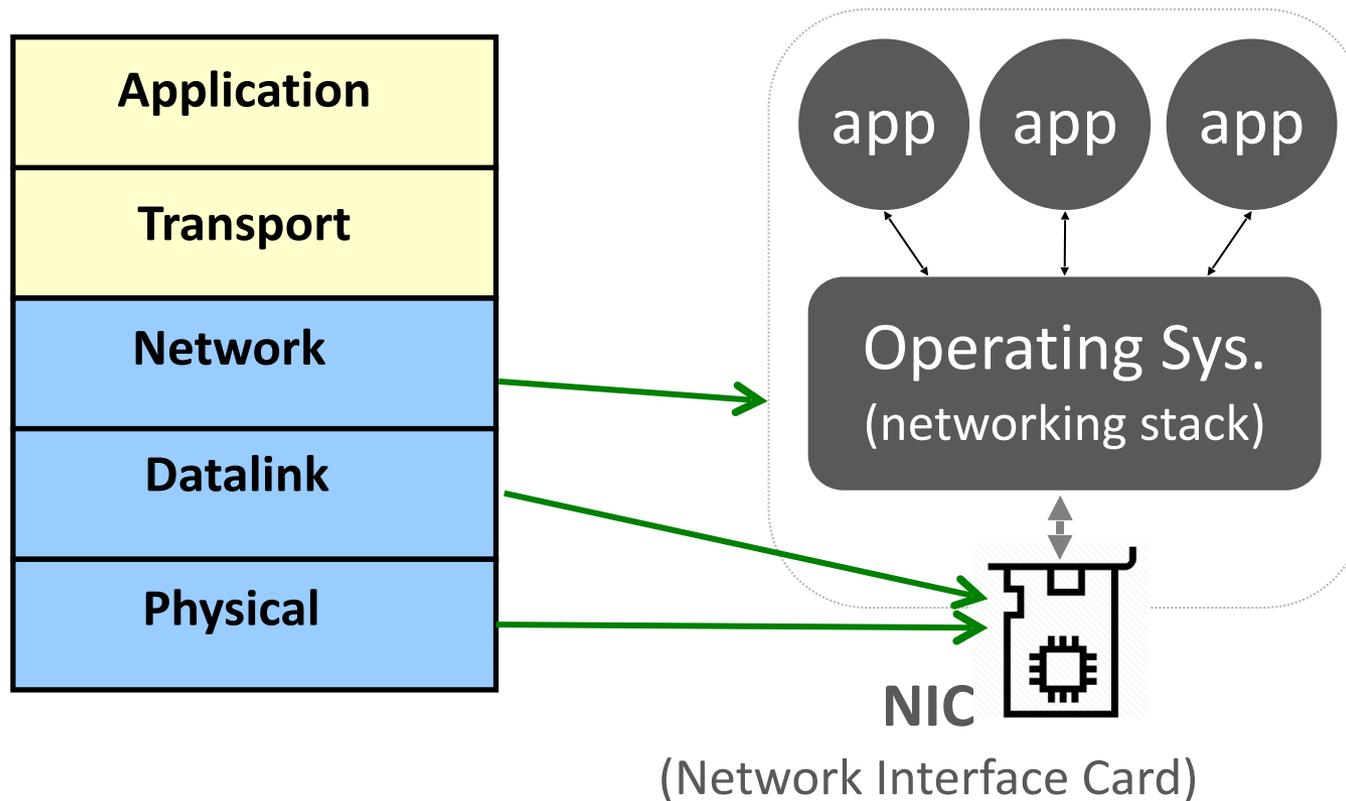


# A closer look: end host

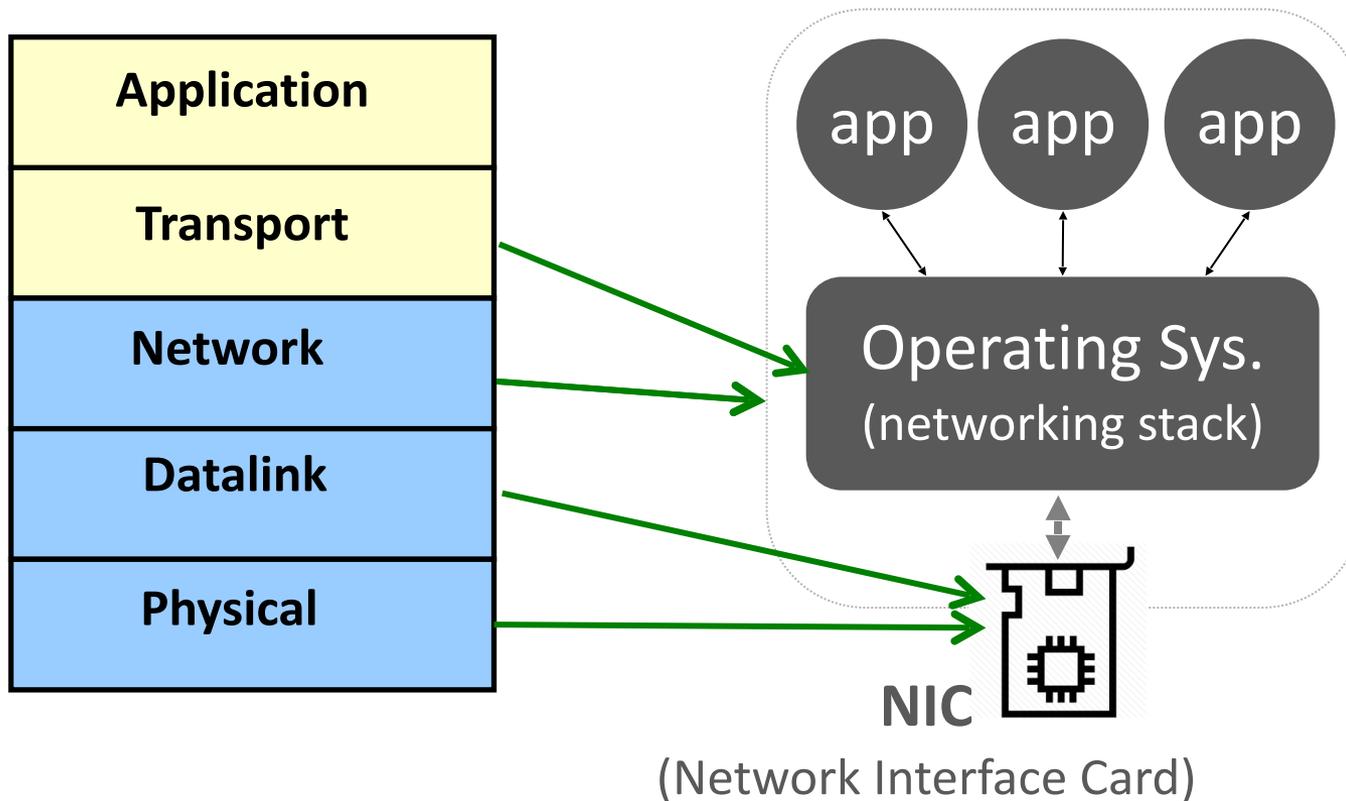


(Network Interface Card)

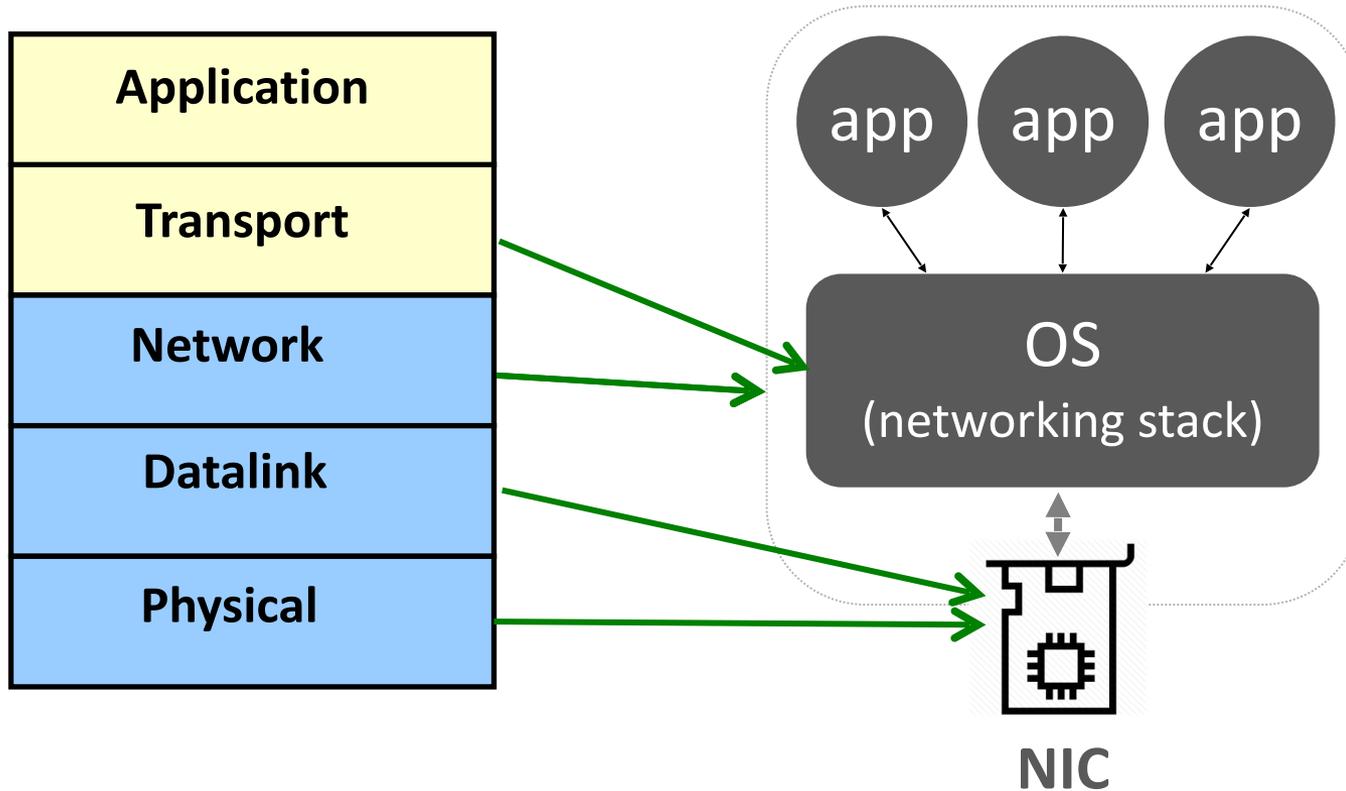
# A closer look: end host



# A closer look: end host

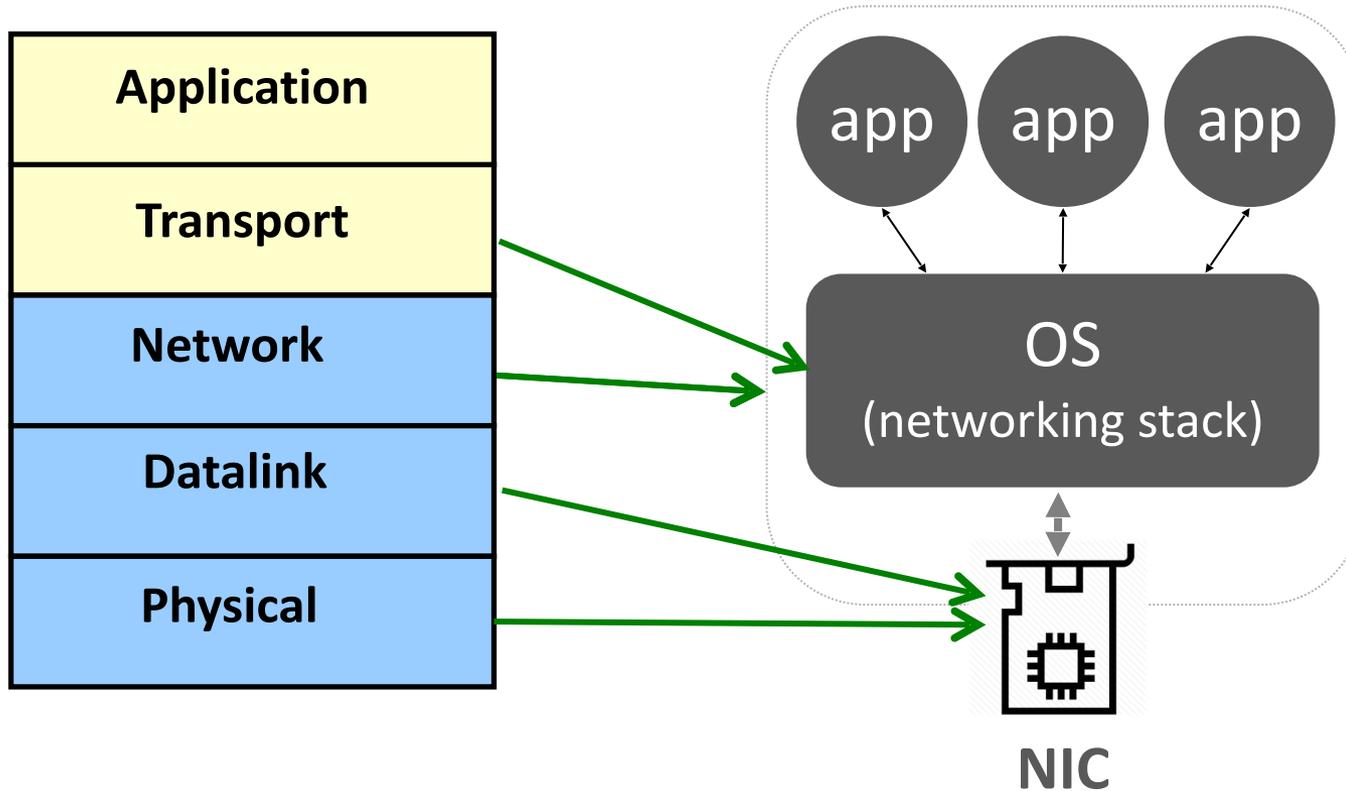


# Note: addressing *within* the end host



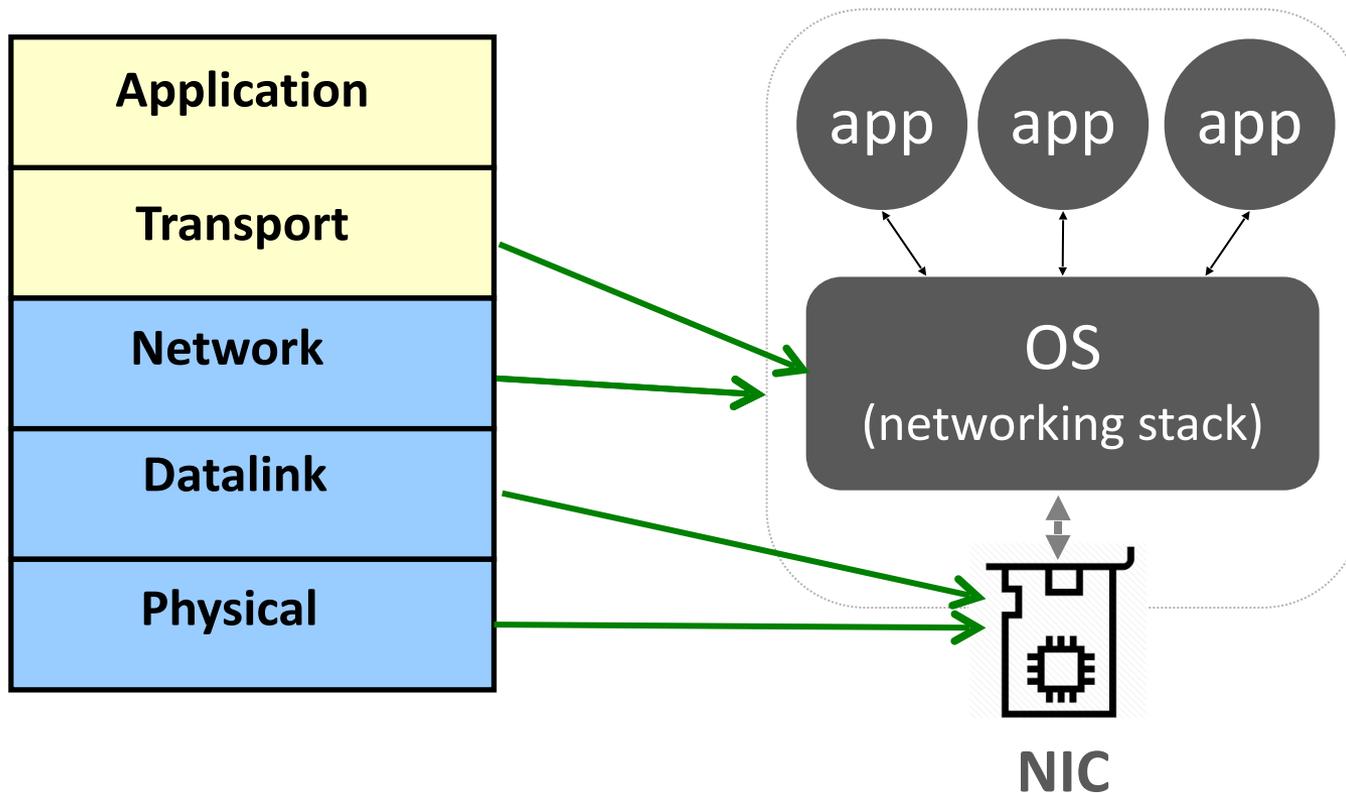
# Note: addressing *within* the end host

Recall: packet contains the destination host's address



# Note: addressing *within* the end host

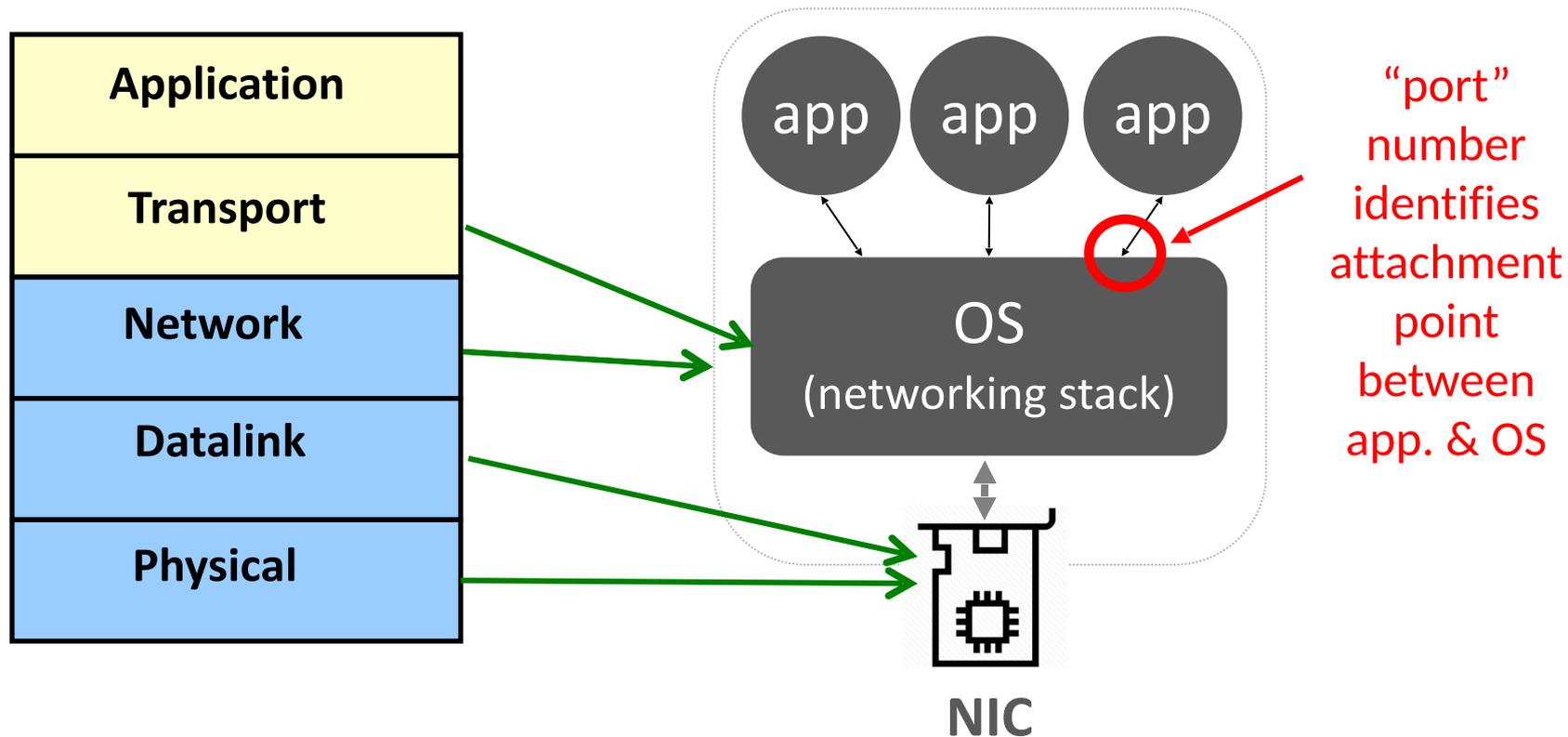
Recall: packet contains the destination host's address



When a packet arrives at the host, how does the OS know which app to send the packet to?

# Note: addressing *within* the end host

Recall: packet contains the destination host's address

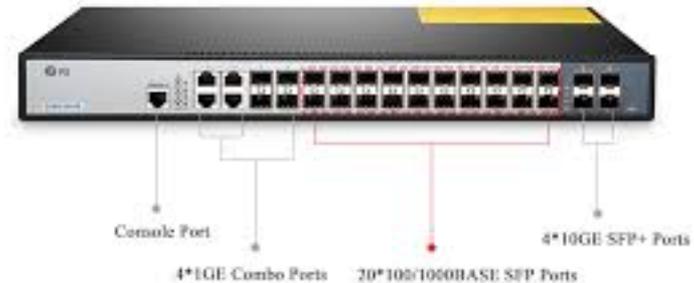


When a packet arrives at the host, how does the OS know which app to send the packet to?

# Network “ports”: two types

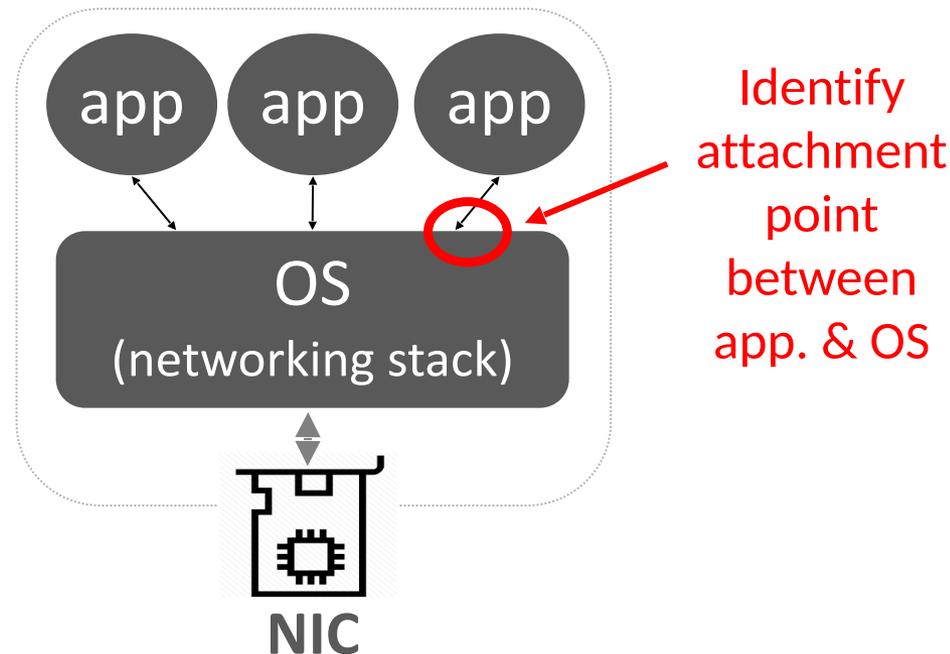
# Network “ports”: two types

- Switches/routers have **physical ports**:
  - Places where links connect to switches



# Network “ports”: two types

- Switches/routers have **physical ports**:
  - Places where links connect to switches
- The OS supports **logical ports**:
  - Place where app connects to OS network stack



# Of Sockets and Ports

*Details about sockets in section!*

# Of Sockets and Ports

- **Socket:** an OS mechanism that connects app processes to the networking stack

*Details about sockets in section!*

# Of Sockets and Ports

- **Socket:** an OS mechanism that connects app processes to the networking stack
- When an app wants access to the network, it opens a **socket**, which is associated with a **port**
  - *This is not a physical port, just a logical one*

*Details about sockets in section!*

# Of Sockets and Ports

- **Socket:** an OS mechanism that connects app processes to the networking stack
- When an app wants access to the network, it opens a **socket**, which is associated with a **port**
  - *This is not a physical port, just a logical one*
- The **port number** is used by the OS to direct incoming packets to its associated socket

*Details about sockets in section!*

# Implications for Packet Header

- Packet header must include:
  - Destination host address (used by network to reach host)

# Implications for Packet Header

- Packet header must include:
  - Destination host address (used by network to reach host)
  - Destination port (used by host OS to reach app) **[new!]**

# Implications for Packet Header

- Packet header must include:
  - Destination host address (used by network to reach host)
  - Destination port (used by host OS to reach app) **[new!]**
- When a packet arrives at the destination end-host, it is delivered to the socket (process) associated with the packet's destination port

# OS Network Stack Is An Intermediary

# OS Network Stack Is An Intermediary

- Application has very clear task (w.r.t. network)
  - Thinks about data

# OS Network Stack Is An Intermediary

- Application has very clear task (w.r.t. network)
  - Thinks about data
- NIC/driver has very clear task
  - Thinks about packets

# OS Network Stack Is An Intermediary

- Application has very clear task (w.r.t. network)
  - Thinks about data
- NIC/driver has very clear task
  - Thinks about packets

# OS Network Stack Is An Intermediary

- Application has very clear task (w.r.t. network)
  - Thinks about data
- NIC/driver has very clear task
  - Thinks about packets
- Network stack in the intermediary between them
  - Translates between their abstractions

# Recap: layers at the end host

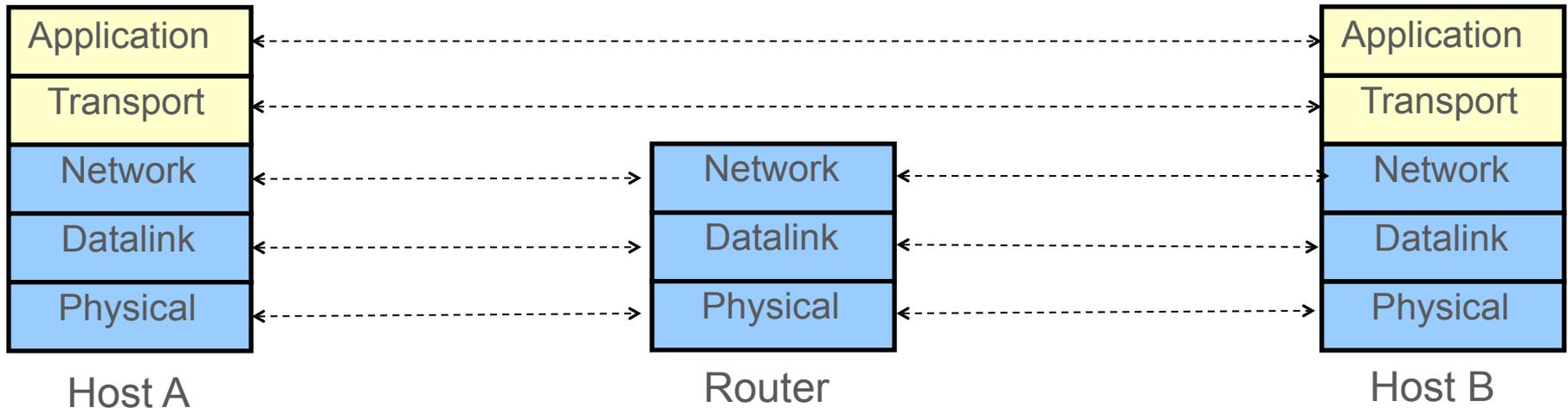
- **Application layer (L7)**
  - part of the app: browser, mail client,...
- **Transport and network layer (L3, L4)**
  - typically part of the OS
- **Datalink and physical layer (L1, L2)**
  - hardware/firmware/drivers

# A closer look: network

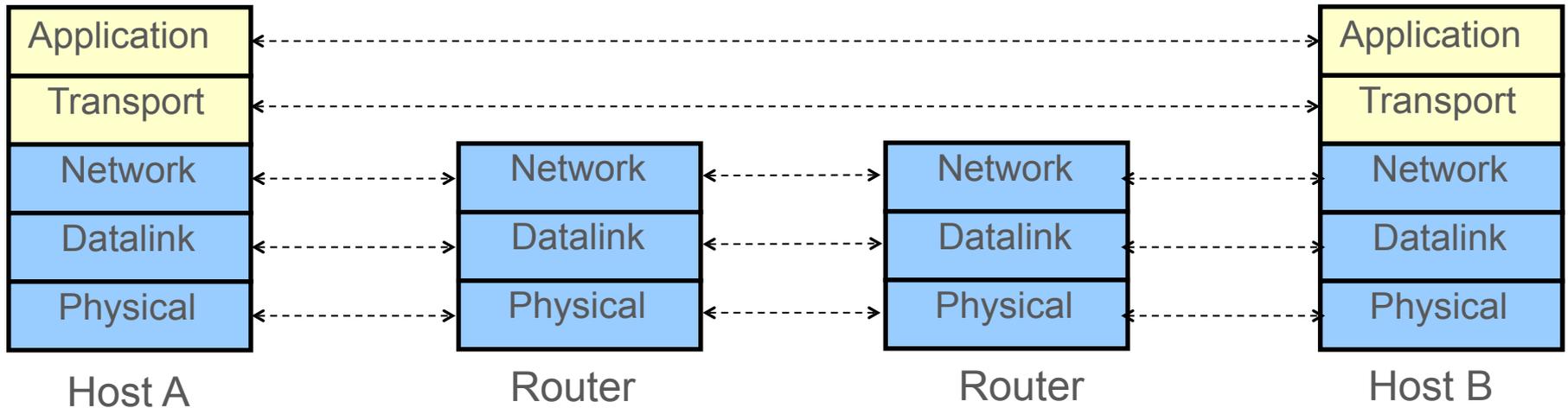
- Bits on wire → physical layer (L1)
- Local delivery of packets → datalink layer (L2)
- Global delivery of packets → network layer (L3)

# Recall: Logical Communication

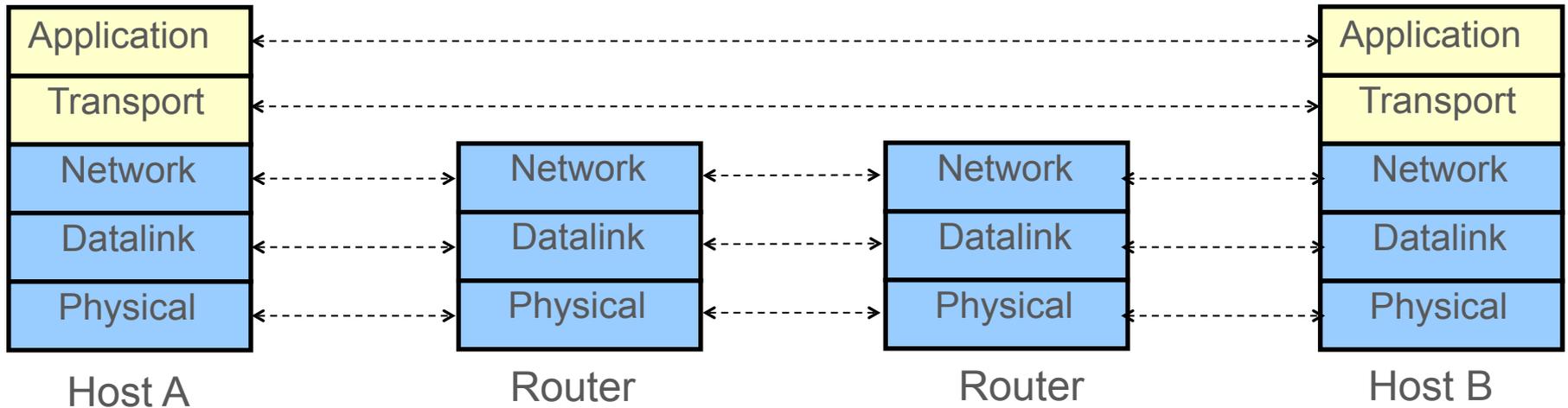
- Layers interact with peer's corresponding layer
- Lower three layers implemented everywhere
- Top two layers implemented only at hosts



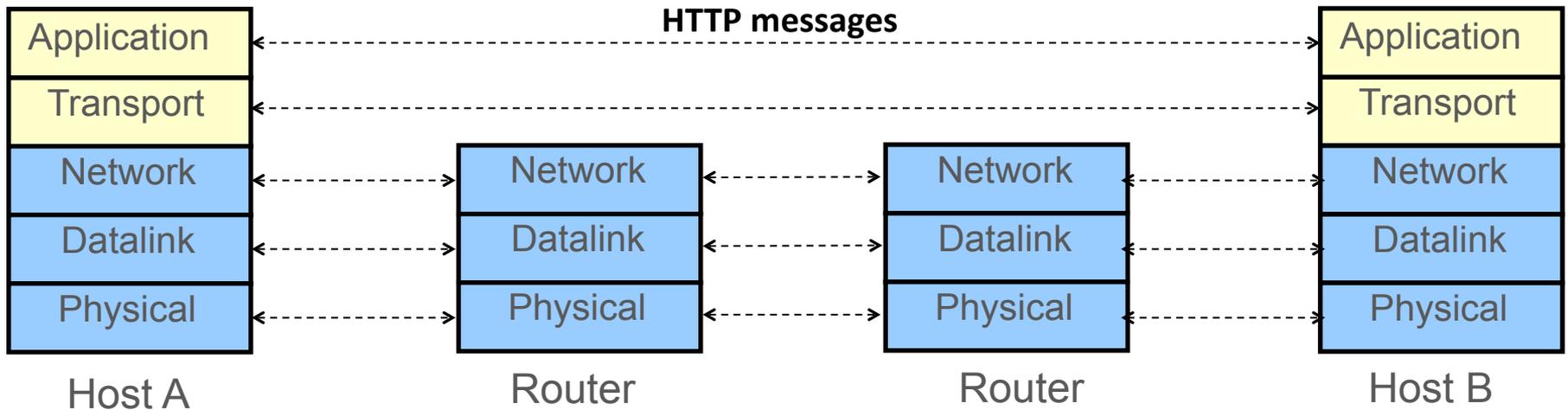
# A closer look: network



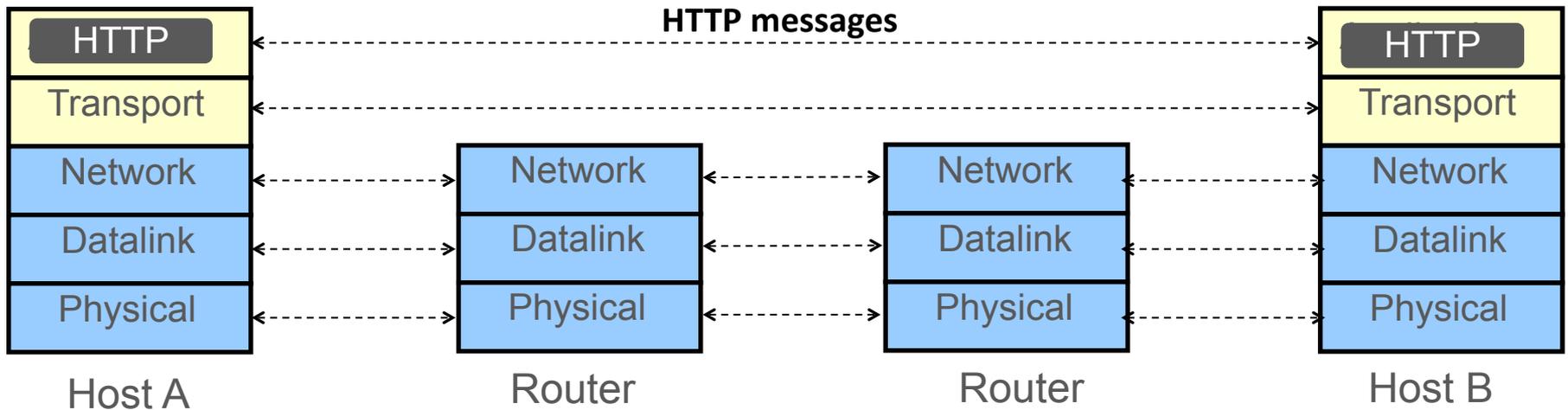
# Example: simple protocol diagram



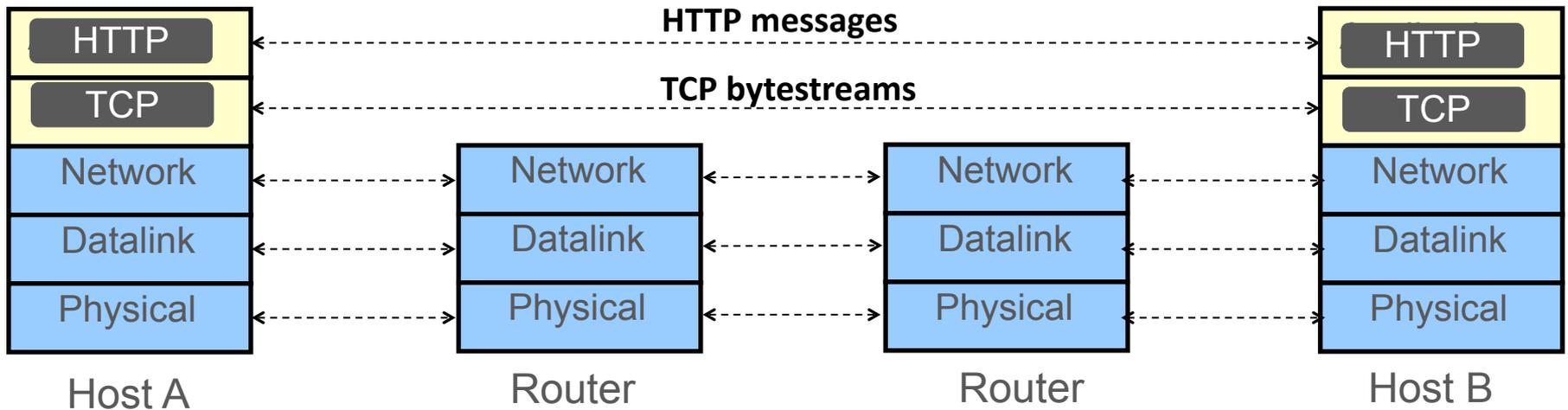
# Example: simple protocol diagram



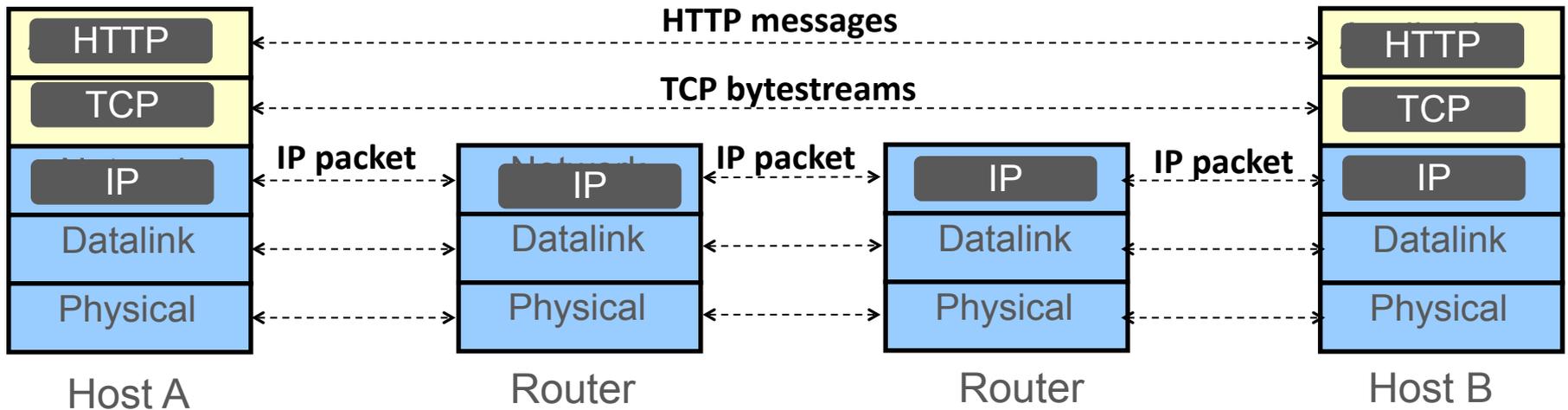
# Example: simple protocol diagram



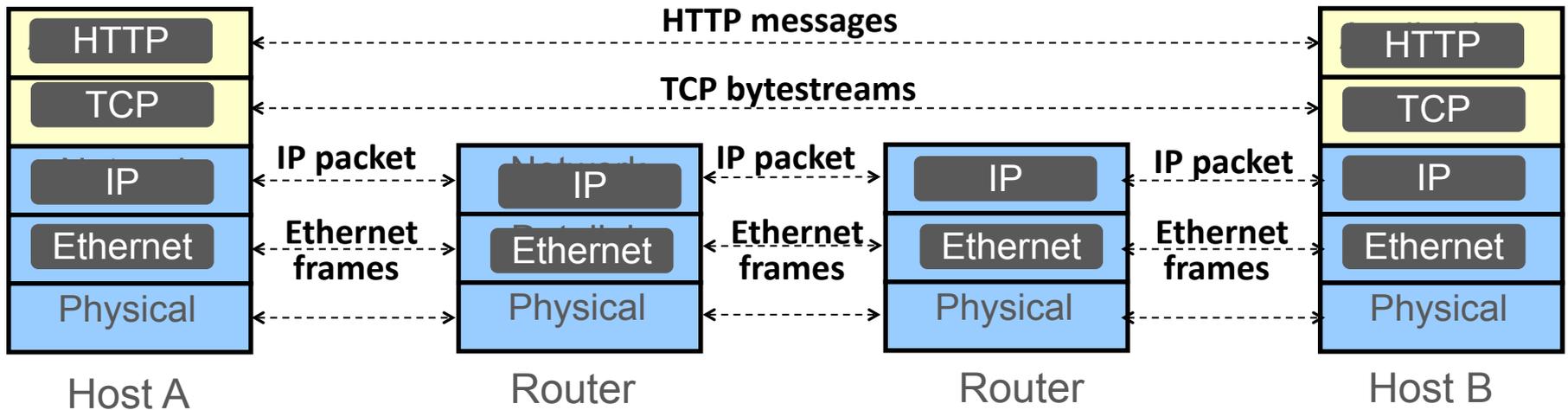
# Example: simple protocol diagram



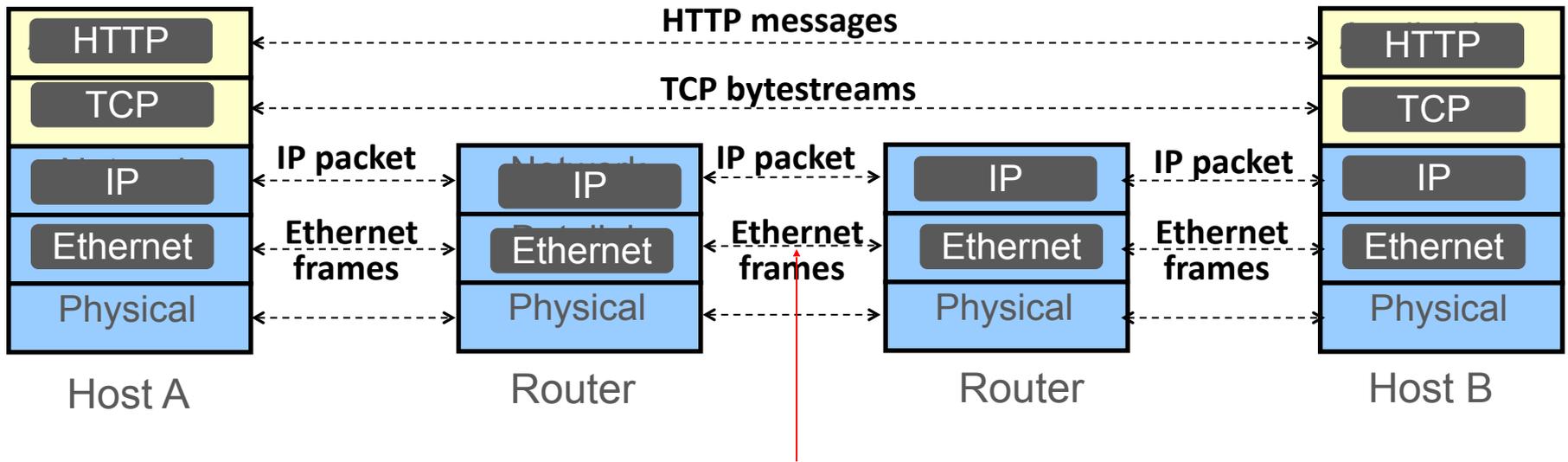
# Example: simple protocol diagram



# Example: simple protocol diagram

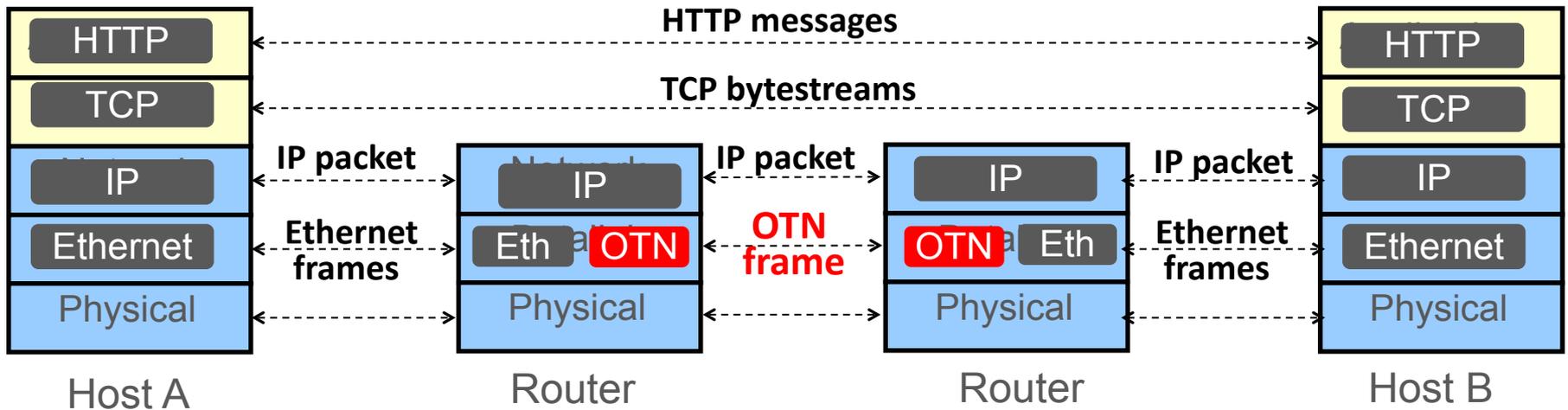


# Example: simple protocol diagram



What if this was an OTN network?

# Example: simple protocol diagram



# Recap: Physical Communication

# Recap: Physical Communication

CEO



Aide



FedEx



CEO



Aide



FedEx

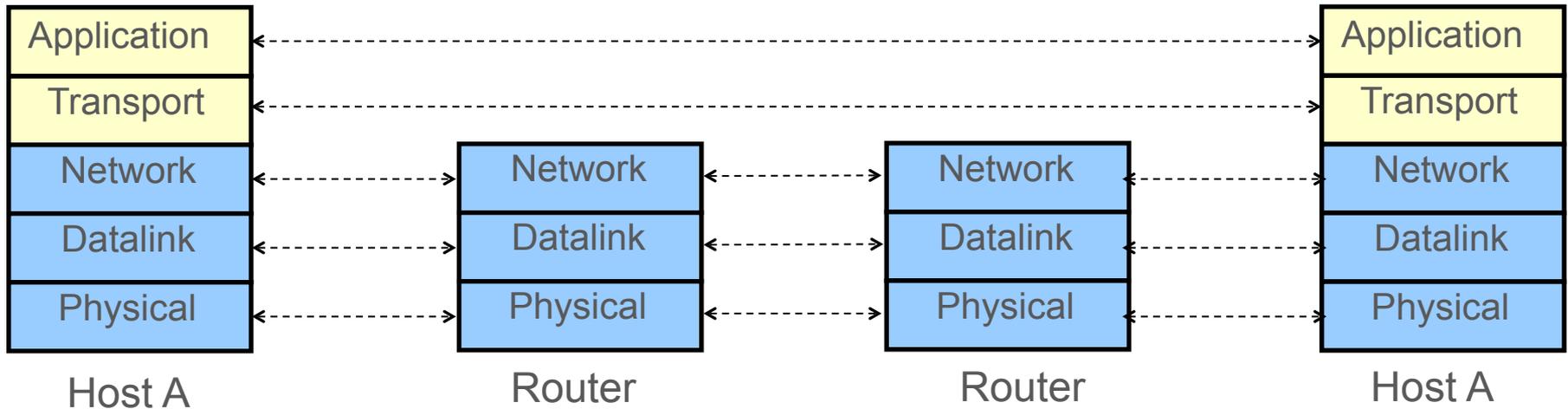
Recall: the path of the letter

# Recap: Physical Communication

- Communication goes down to physical network
- Then up to relevant layer

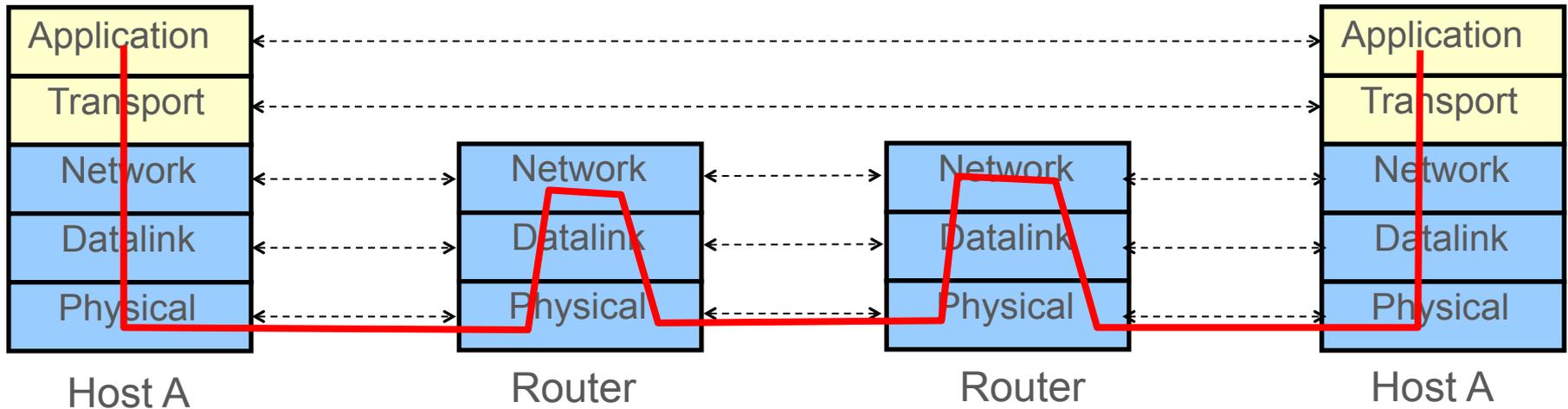
# Recap: Physical Communication

- Communication goes down to physical network
- Then up to relevant layer

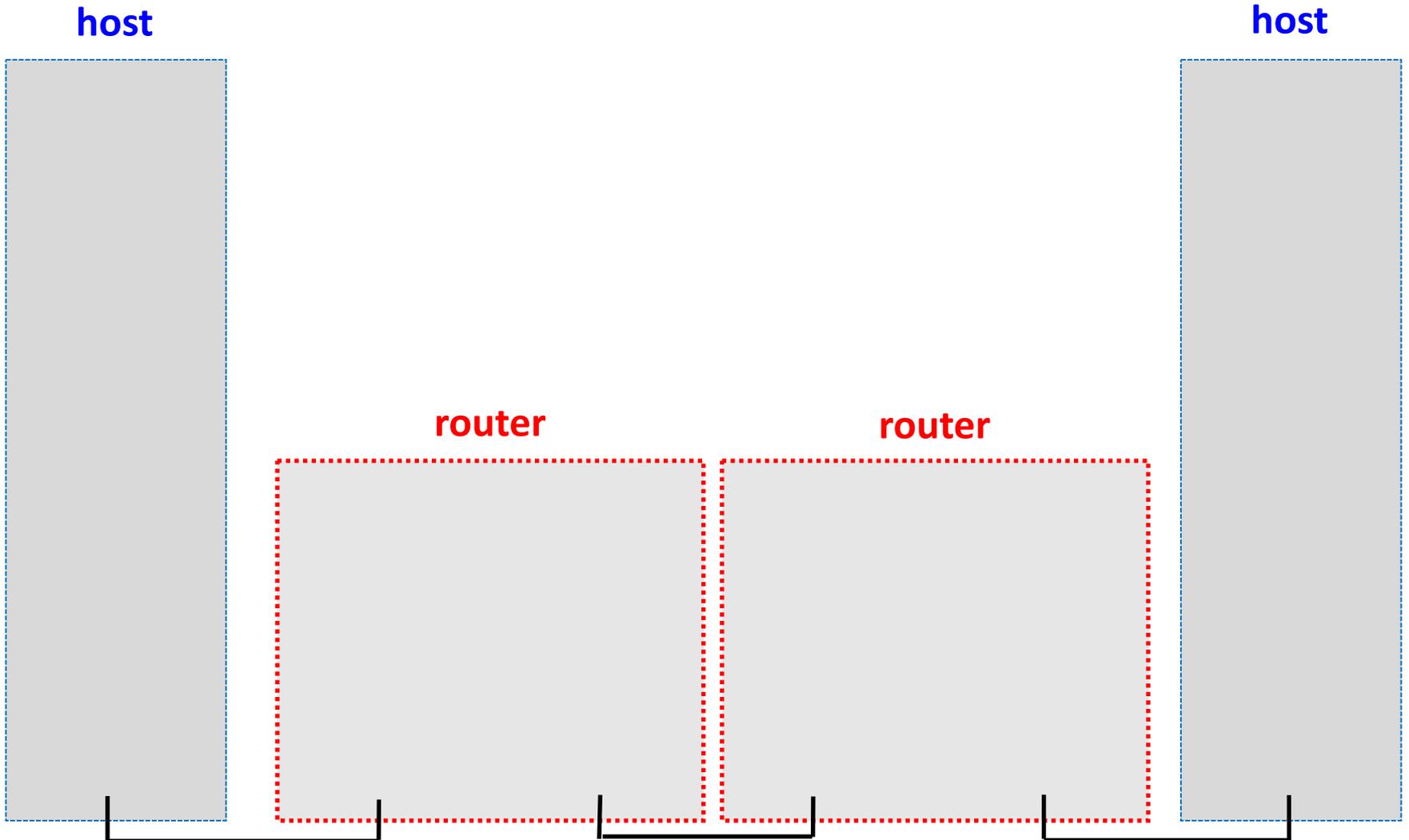


# Recap: Physical Communication

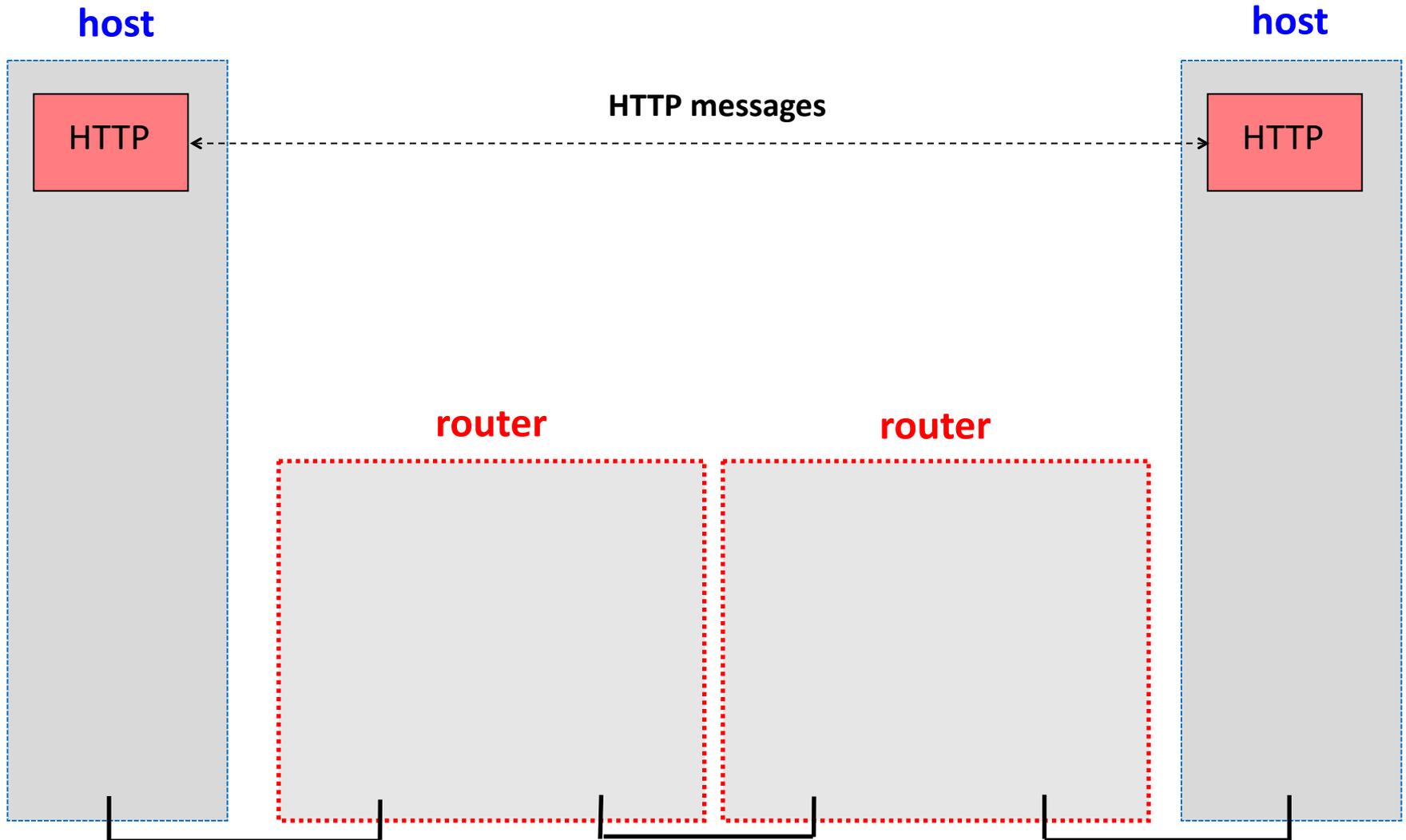
- Communication goes down to physical network
- Then up to relevant layer



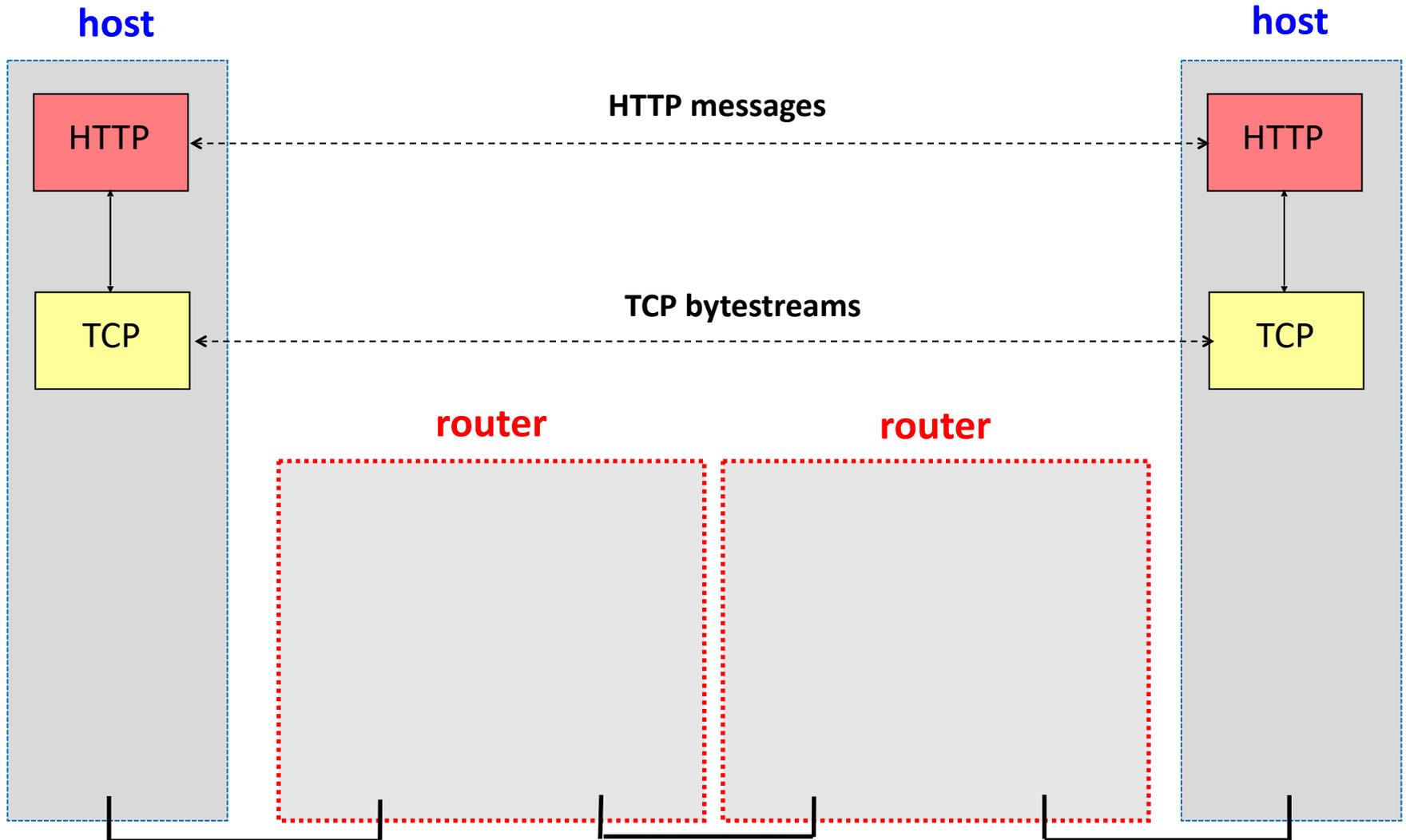
# Complicated protocol diagram



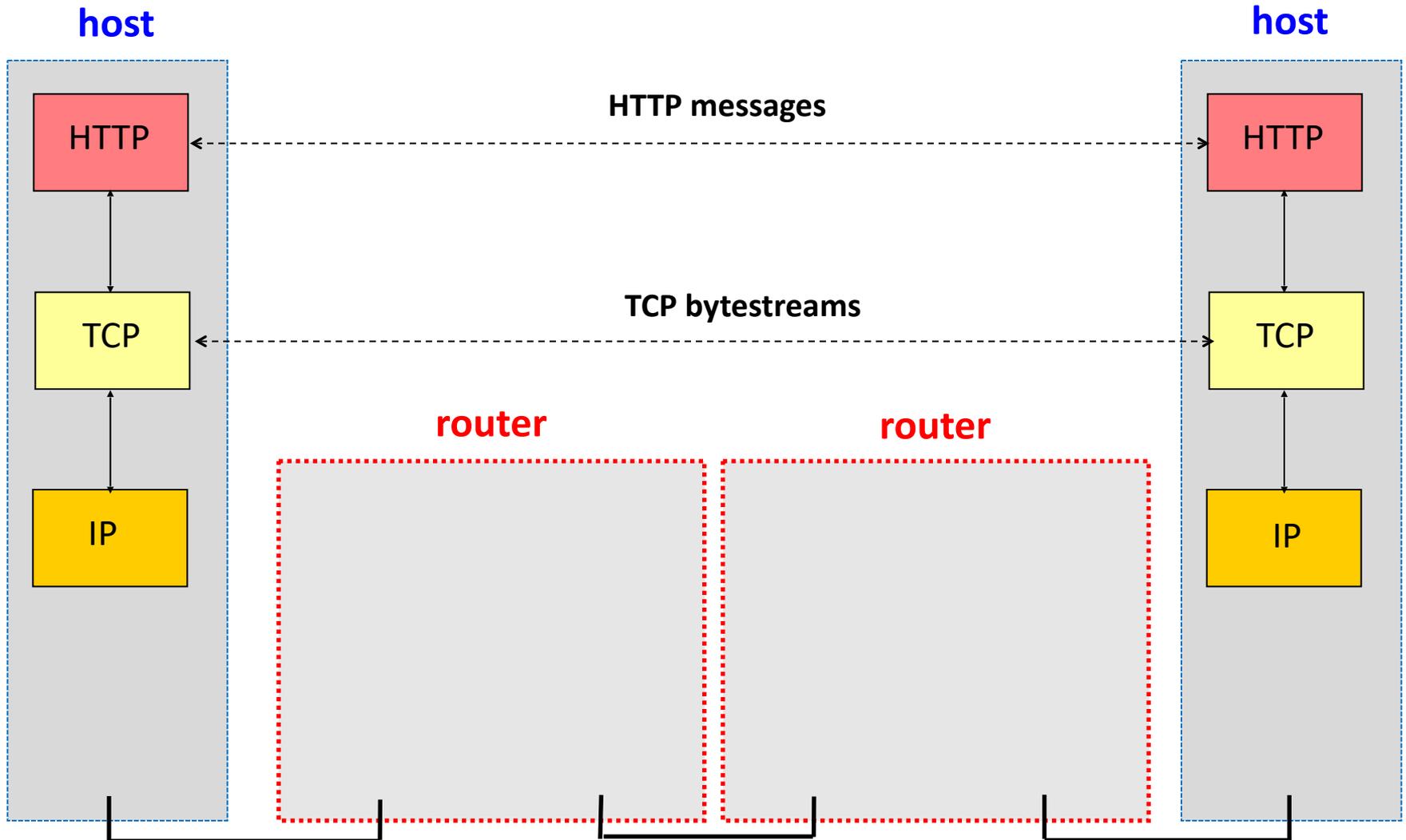
# Complicated protocol diagram



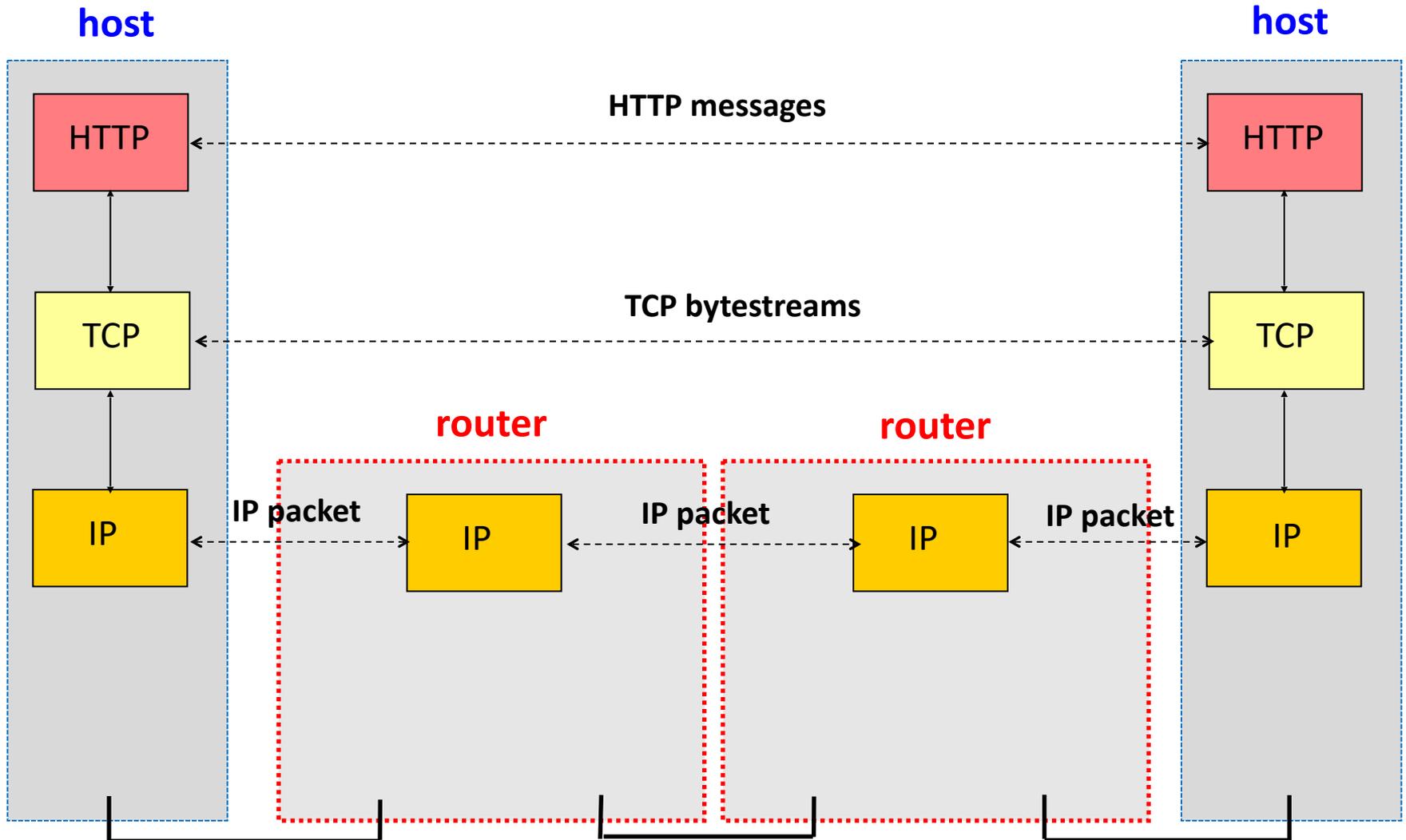
# Complicated protocol diagram



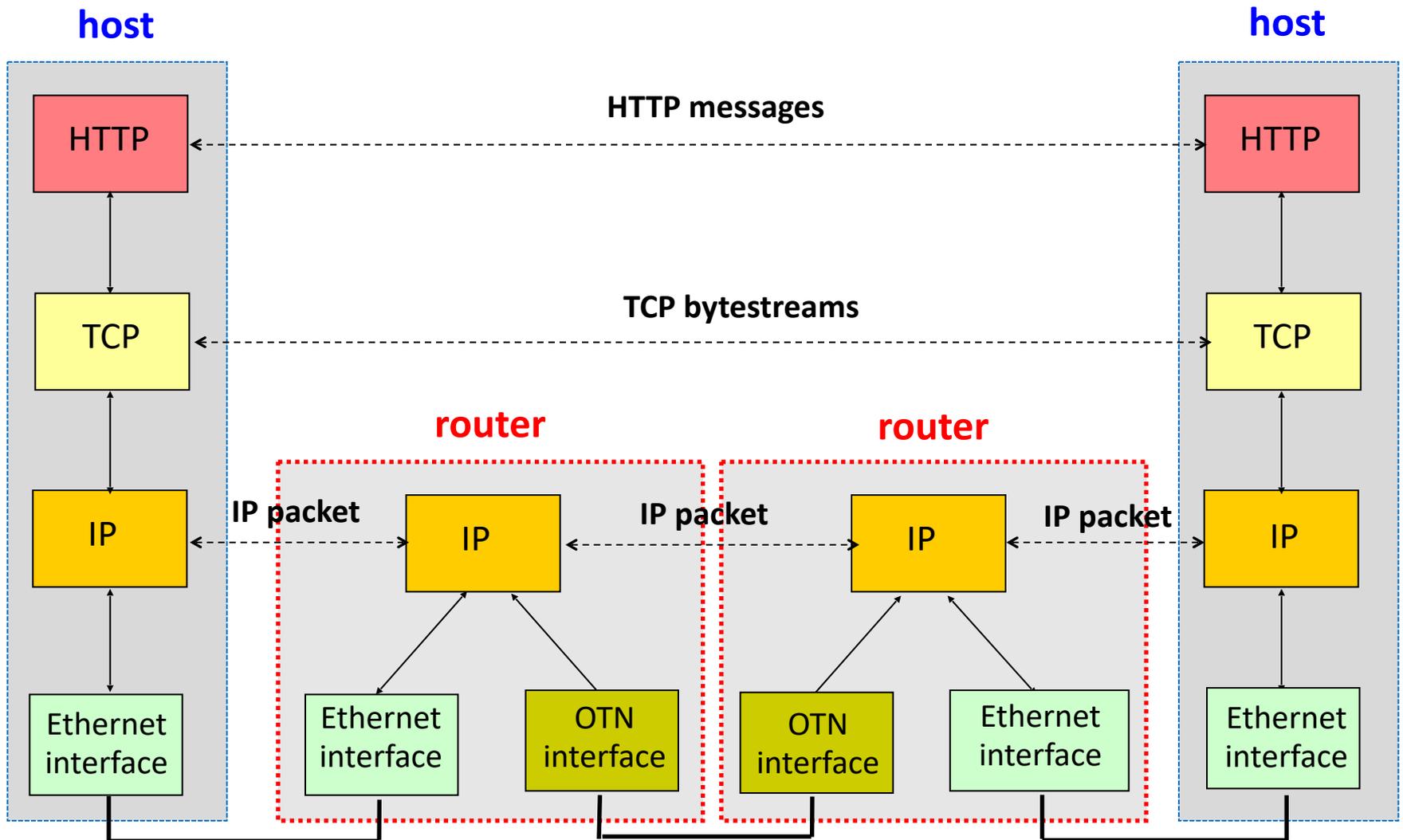
# Complicated protocol diagram



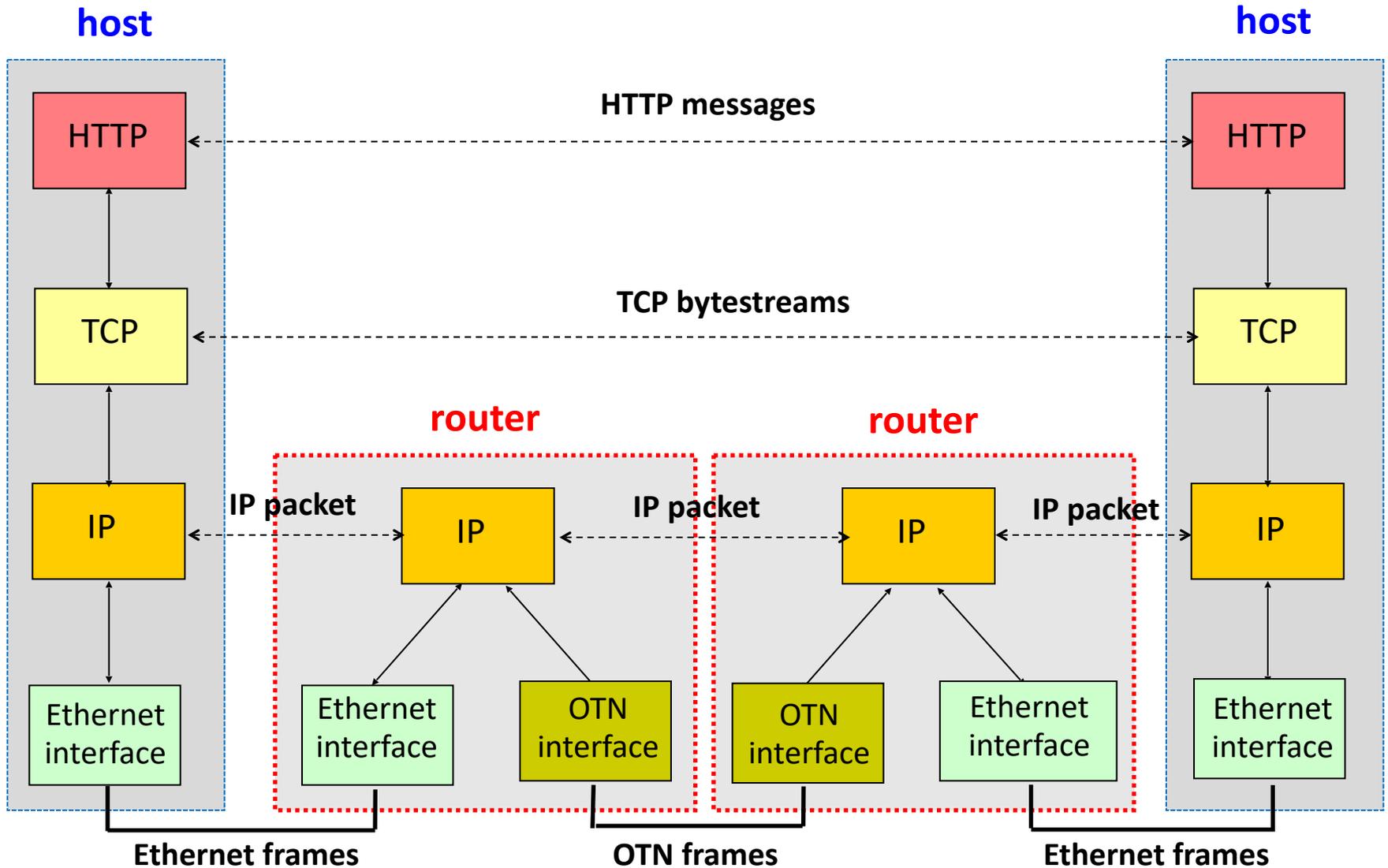
# Complicated protocol diagram



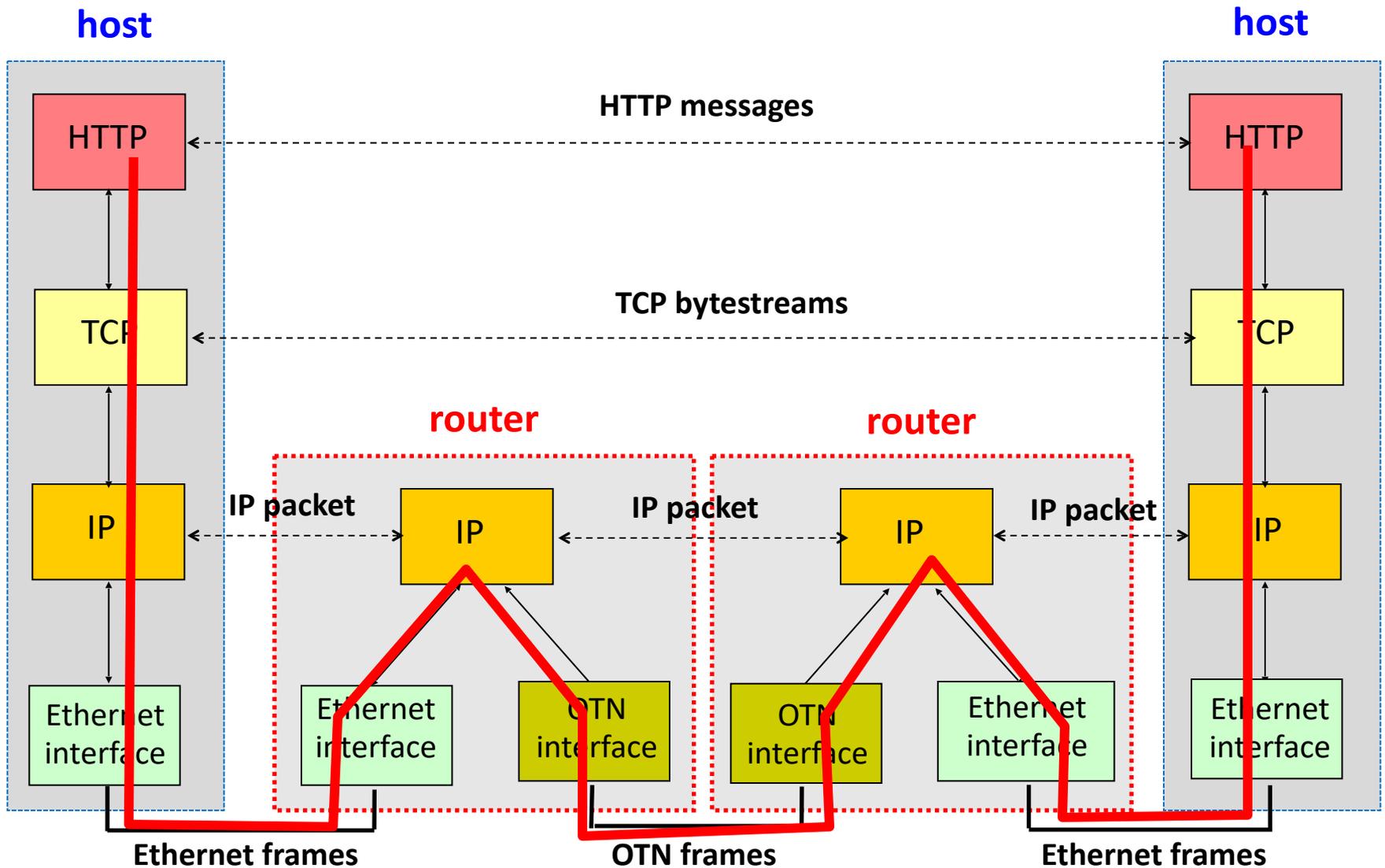
# Complicated protocol diagram



# Complicated protocol diagram



# Complicated protocol diagram



# Recap: Physical Communication

CEO



Aide



FedEx



CEO



Aide



FedEx

# Recap: Physical Communication

**CEO**

**Letter**

**CEO**



**Aide**

**Envelope [Letter]**



**Aide**



**FedEx**



**Fedex Env [ Envelope [Letter] ]**

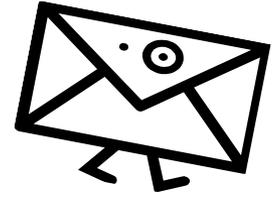


**FedEx**

# Recap: Physical Communication

- Communication goes down to physical network
- Then up to relevant layer
- Lowest layer has the most “packaging”

# Layer Encapsulation

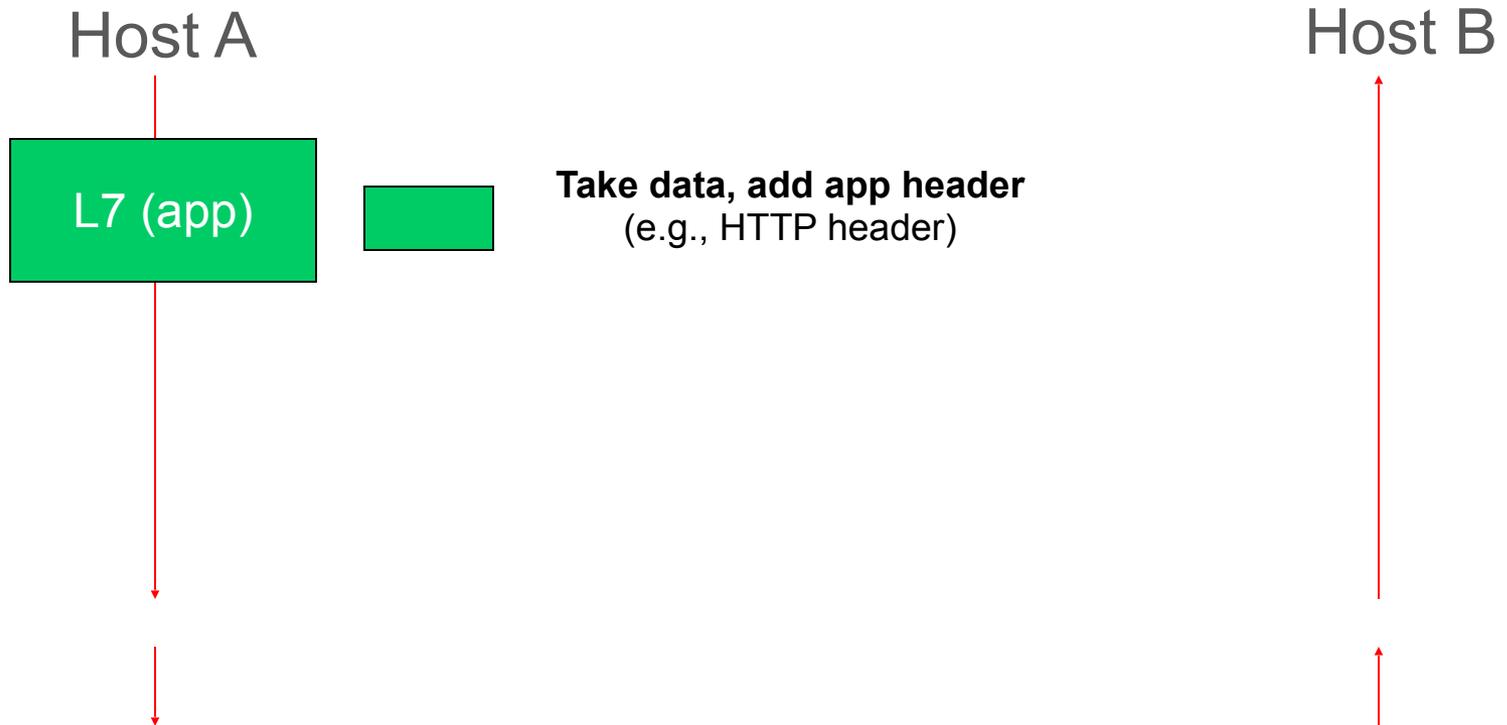
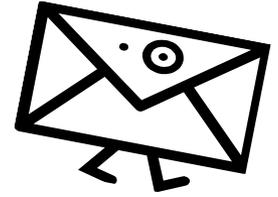


Host A

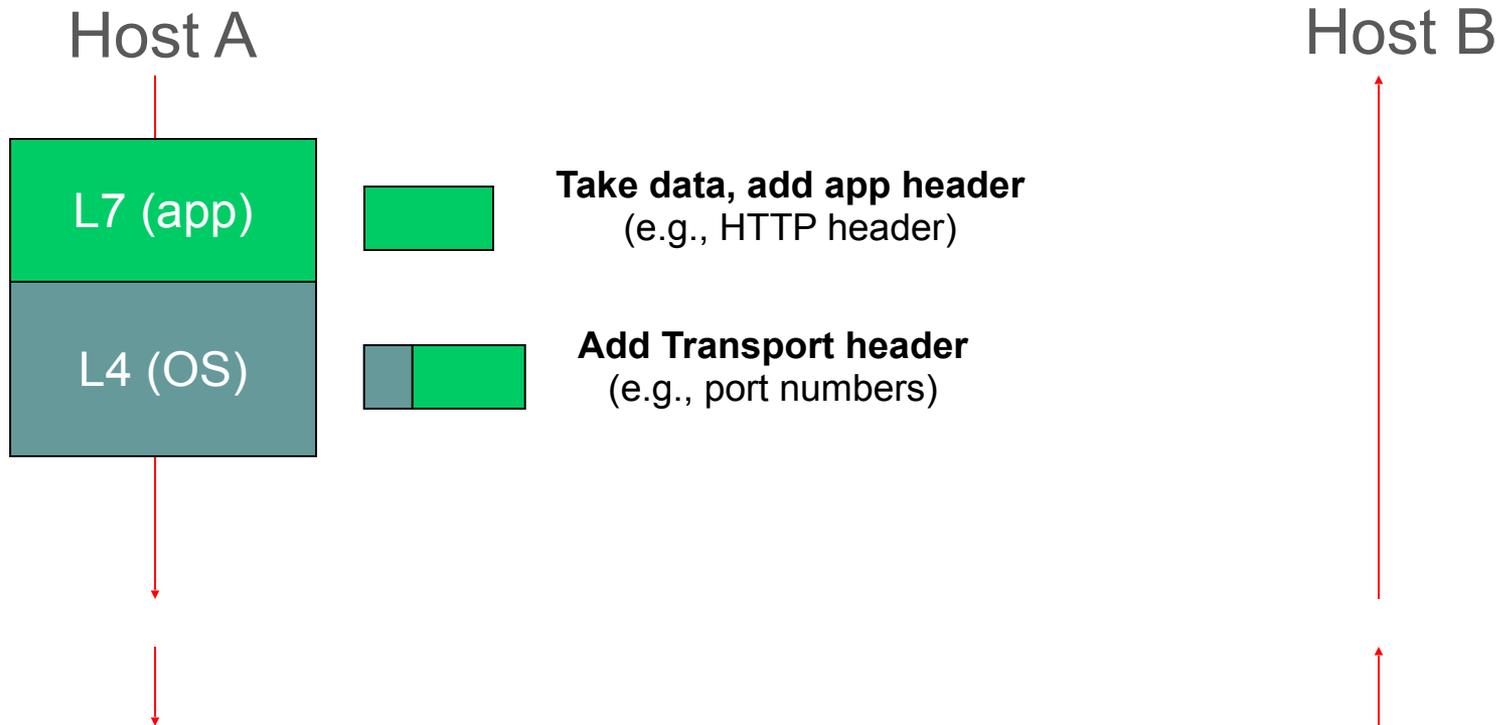
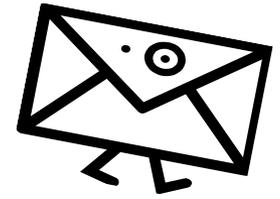
Host B



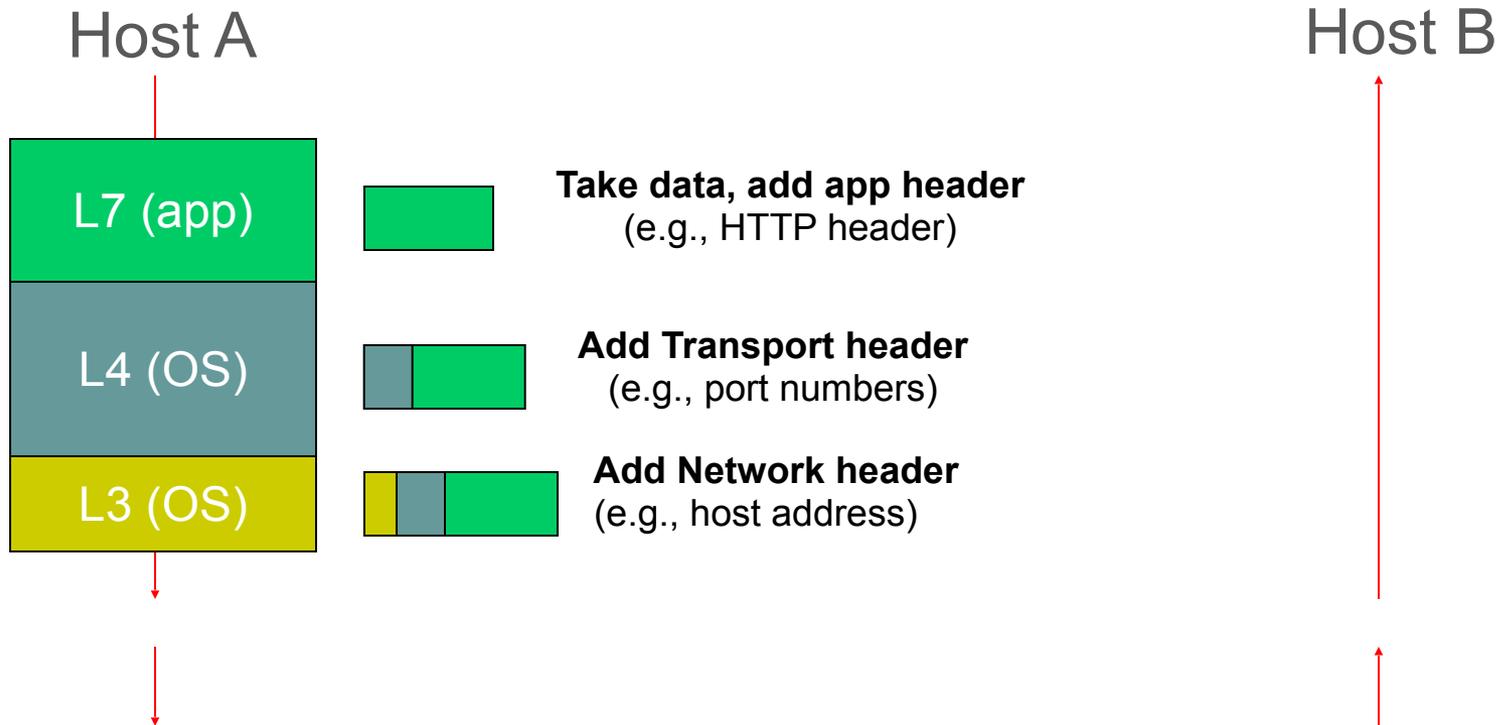
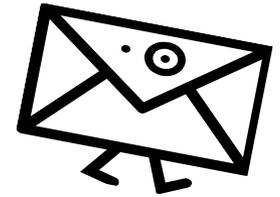
# Layer Encapsulation



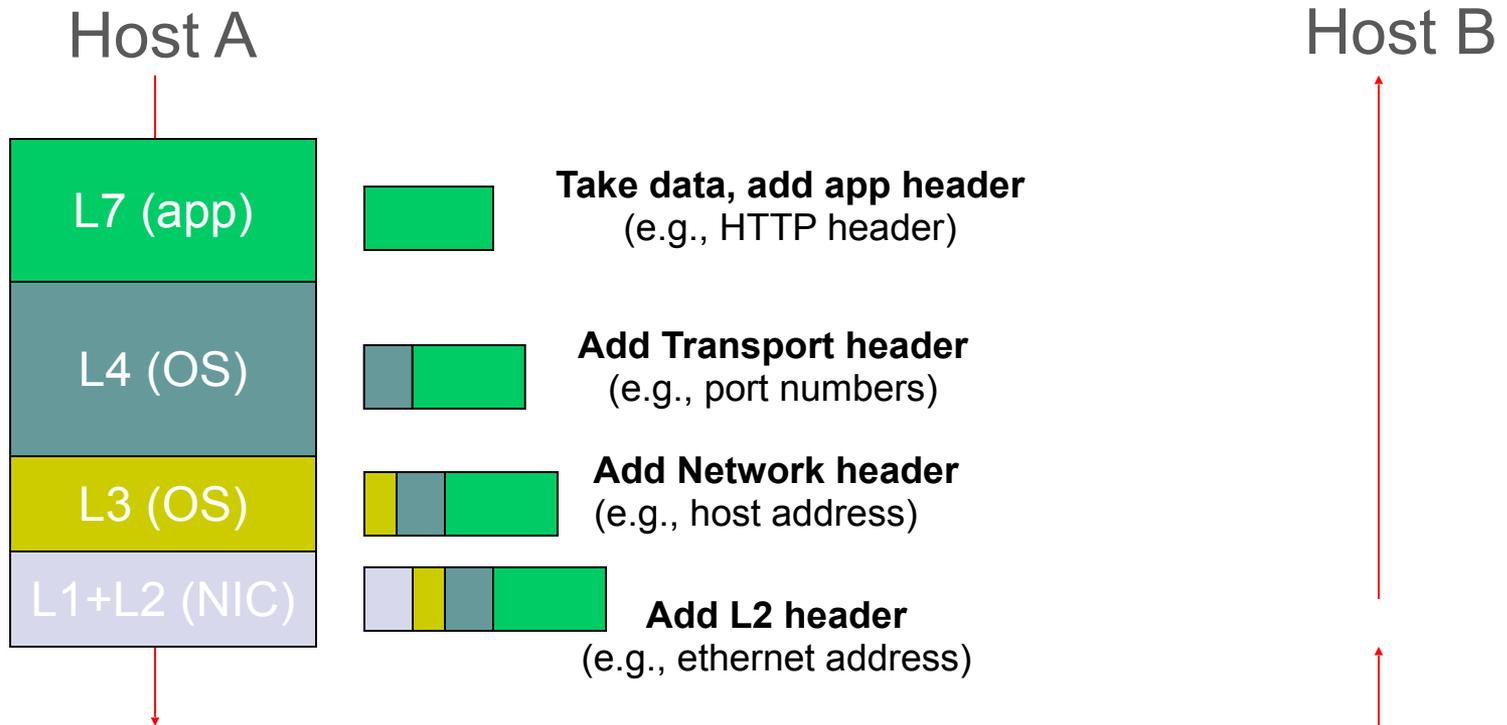
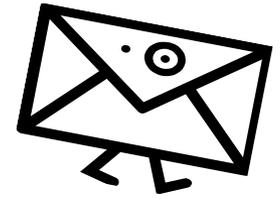
# Layer Encapsulation



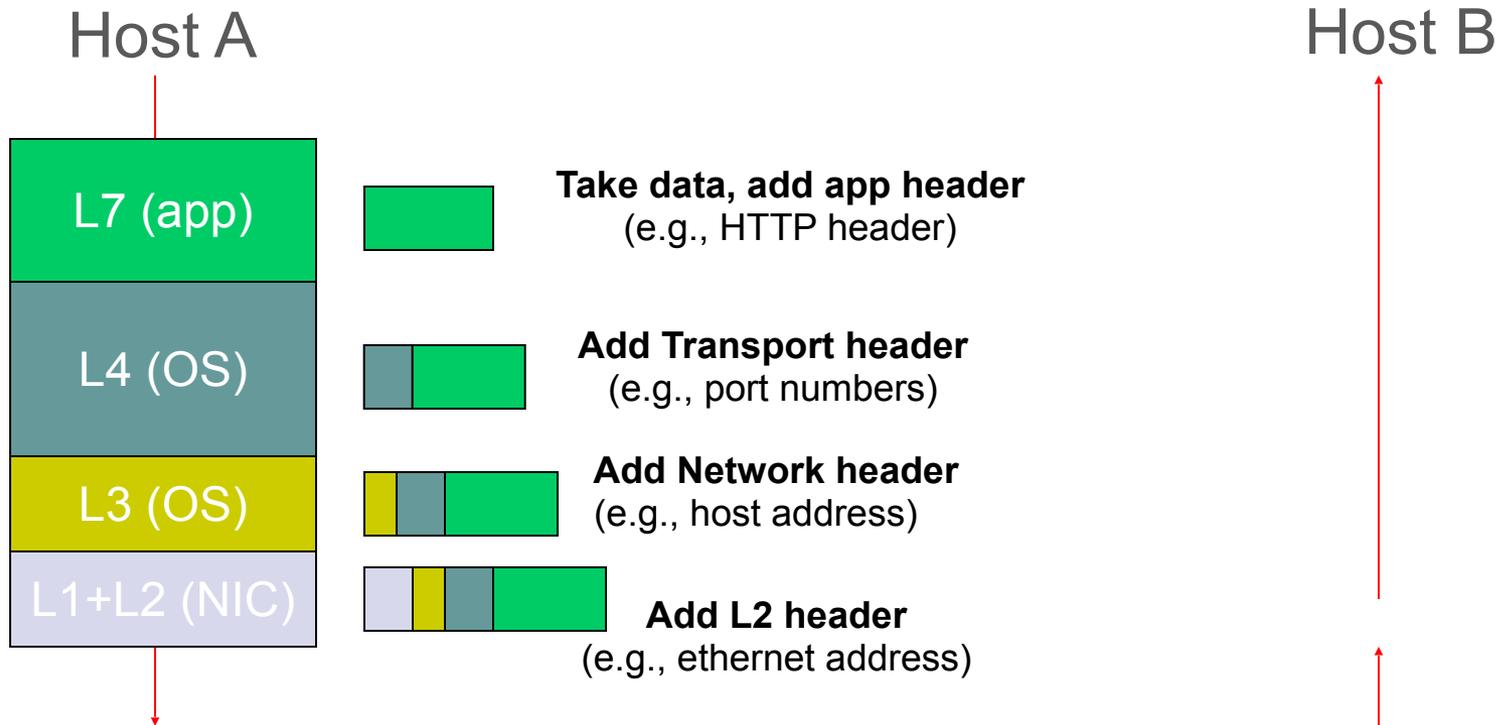
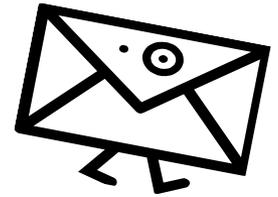
# Layer Encapsulation



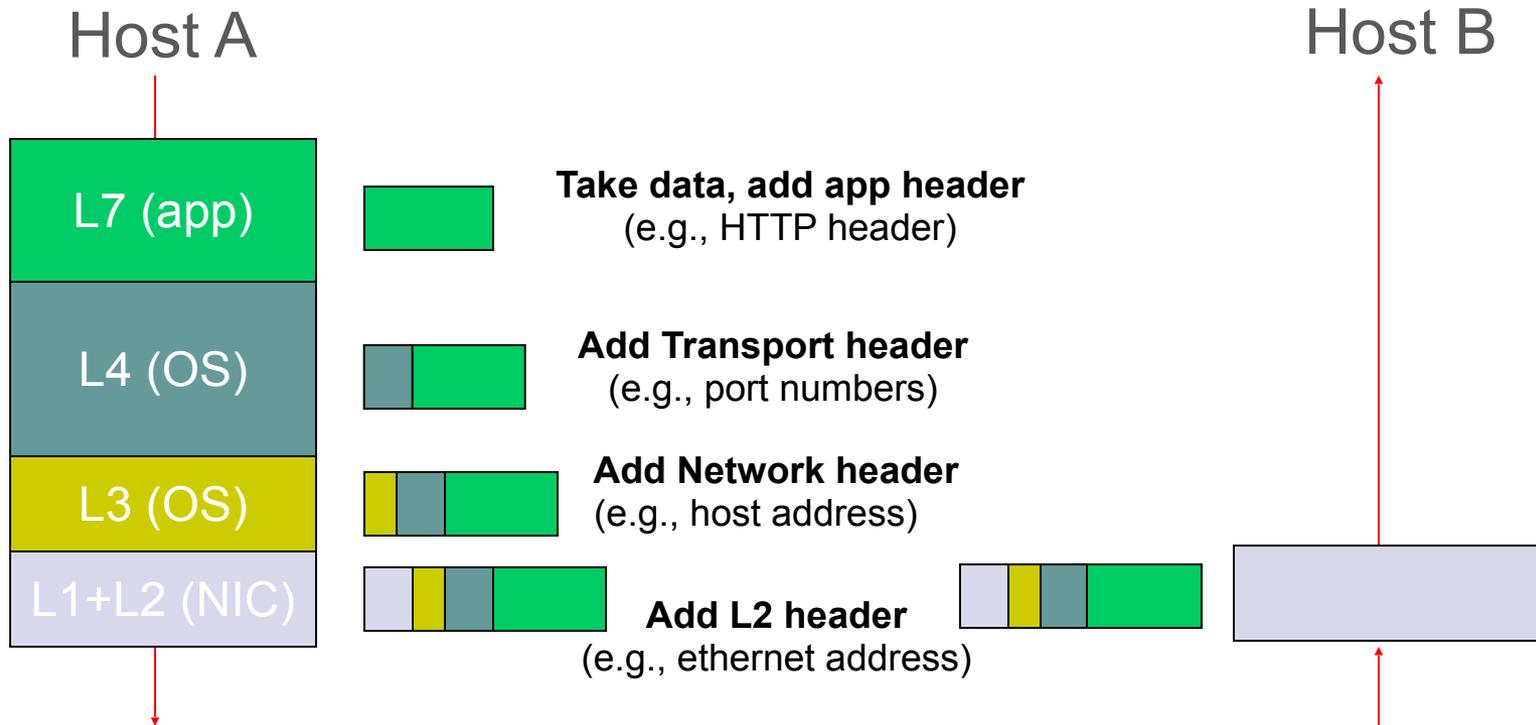
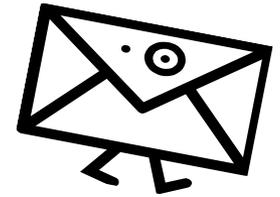
# Layer Encapsulation



# Layer Encapsulation

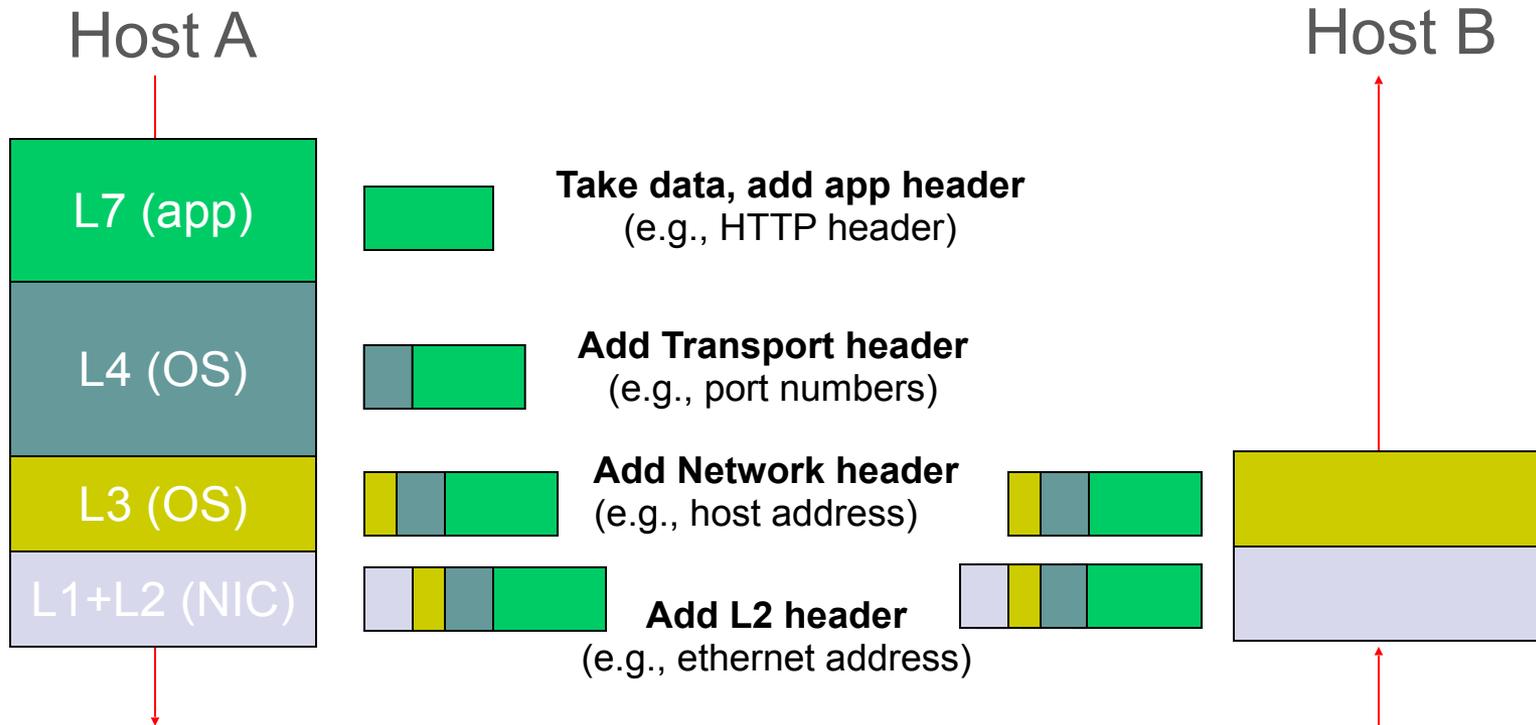
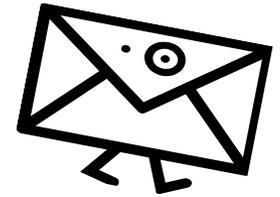


# Layer Encapsulation

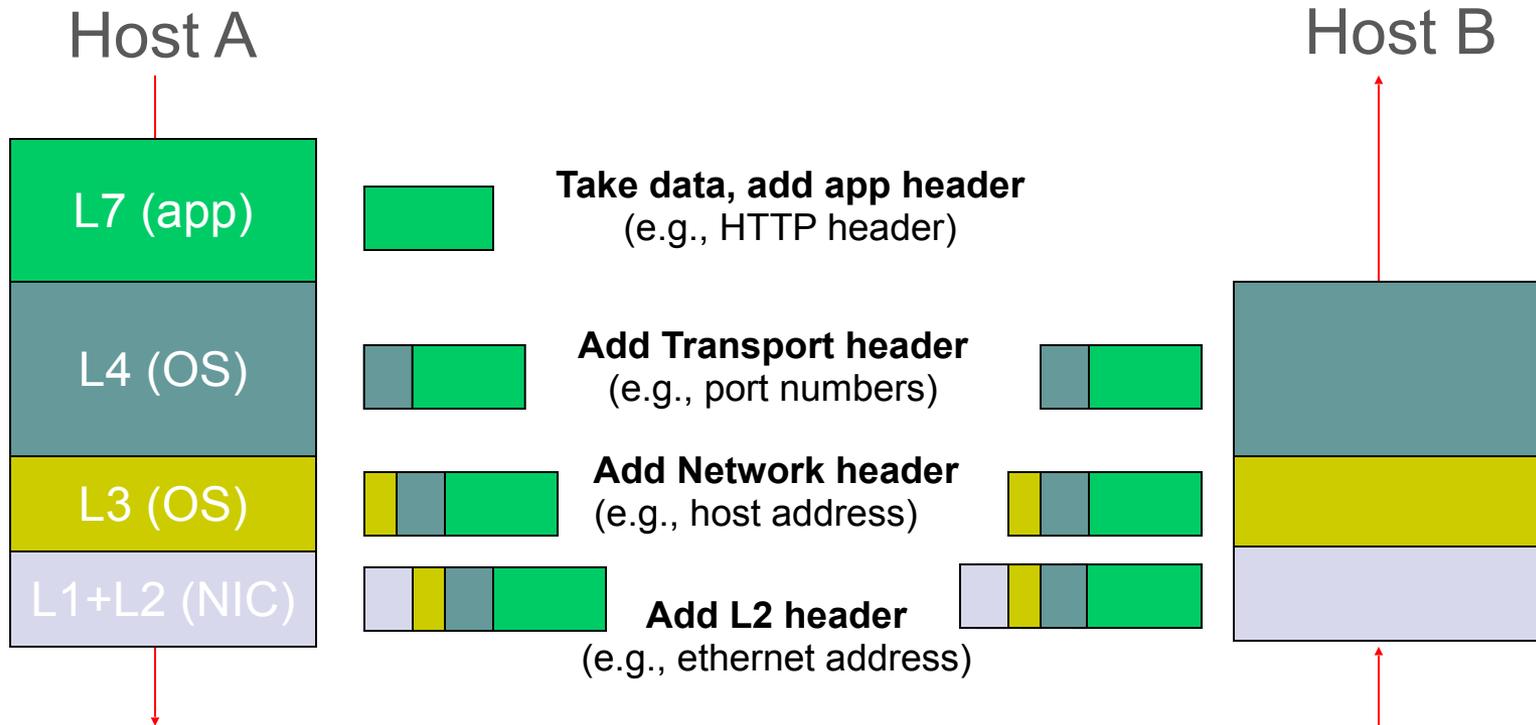
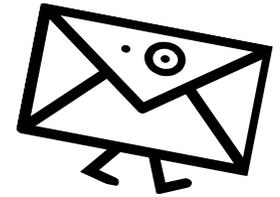


On wire: packet has data + headers from all layers!

# Layer Encapsulation

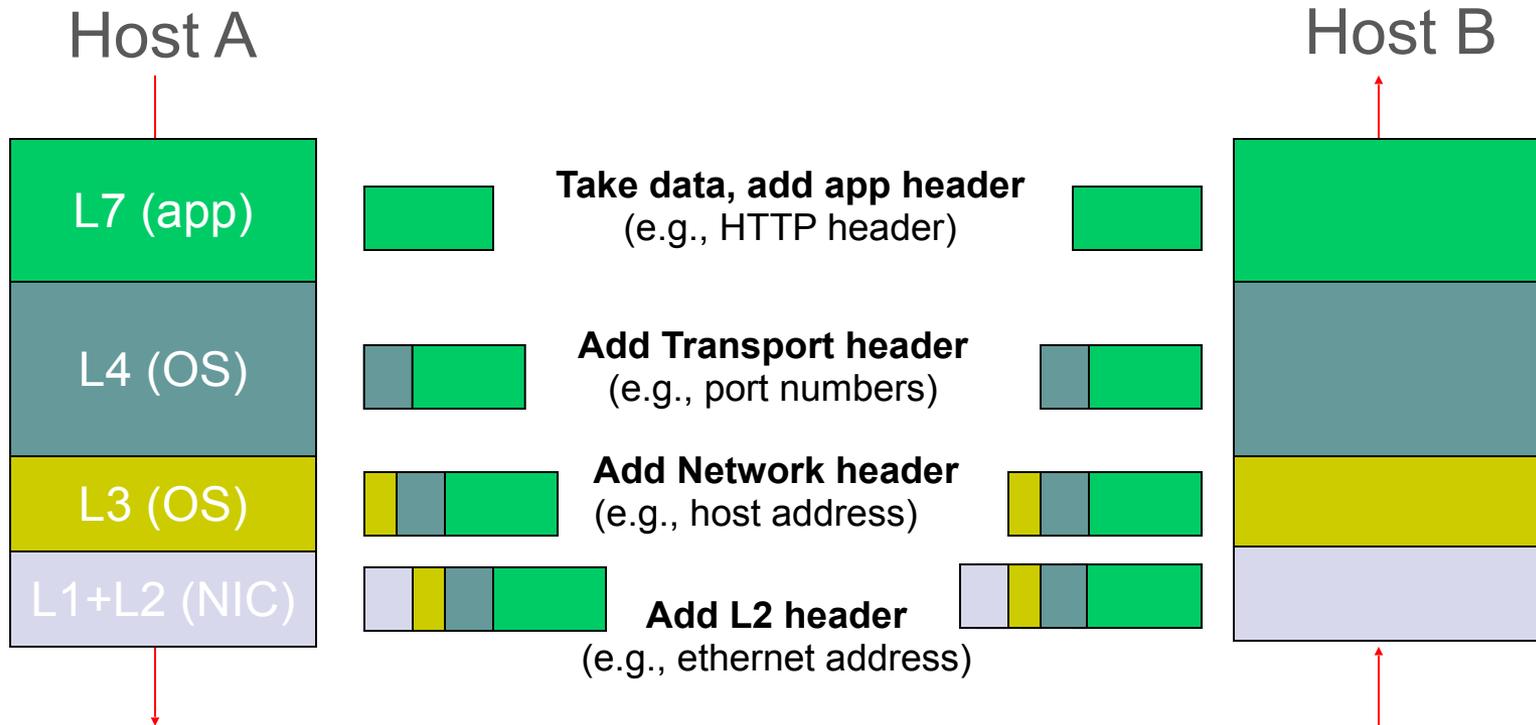
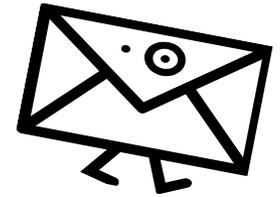


# Layer Encapsulation



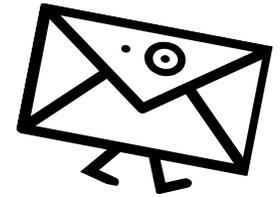
On wire: packet has data + headers from all layers!

# Layer Encapsulation

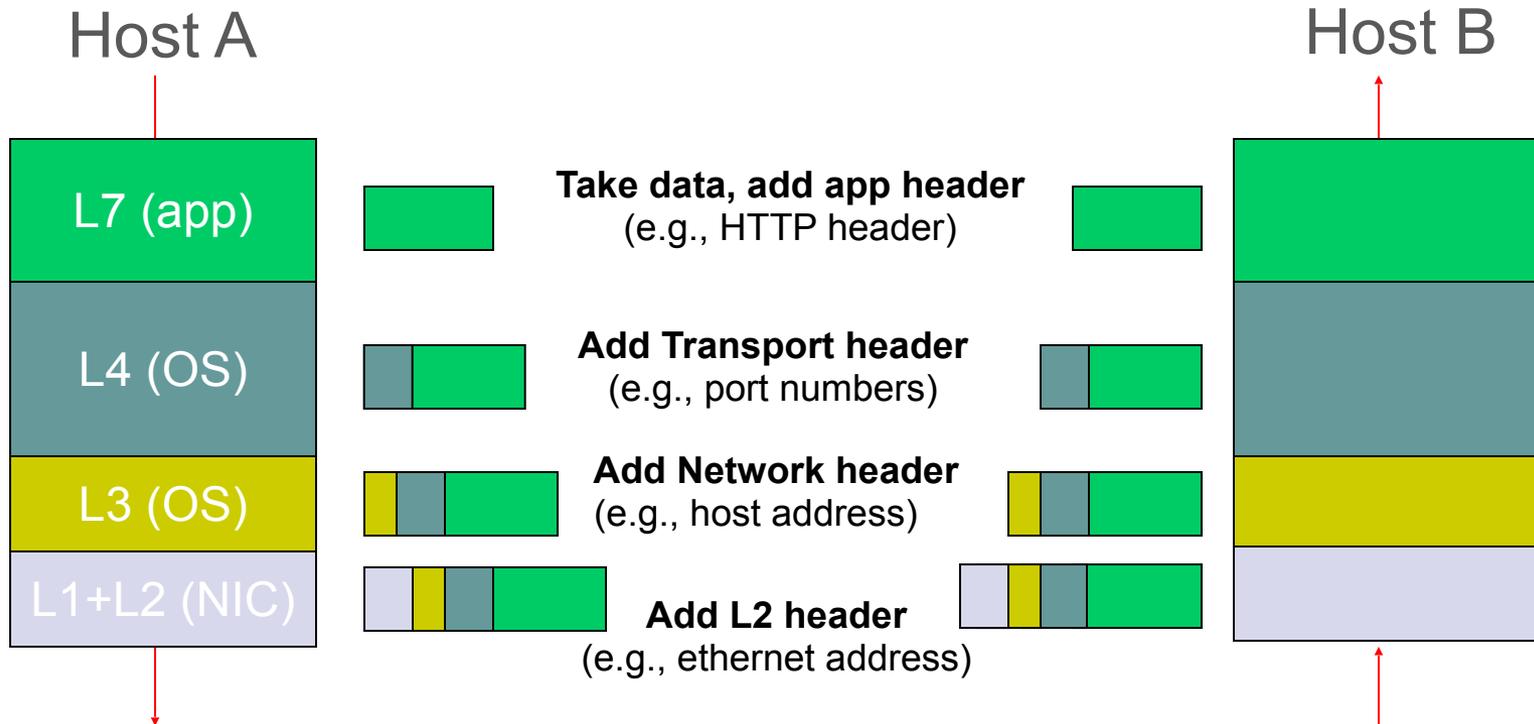


On wire: packet has data + headers from all layers!

# Layer Encapsulation

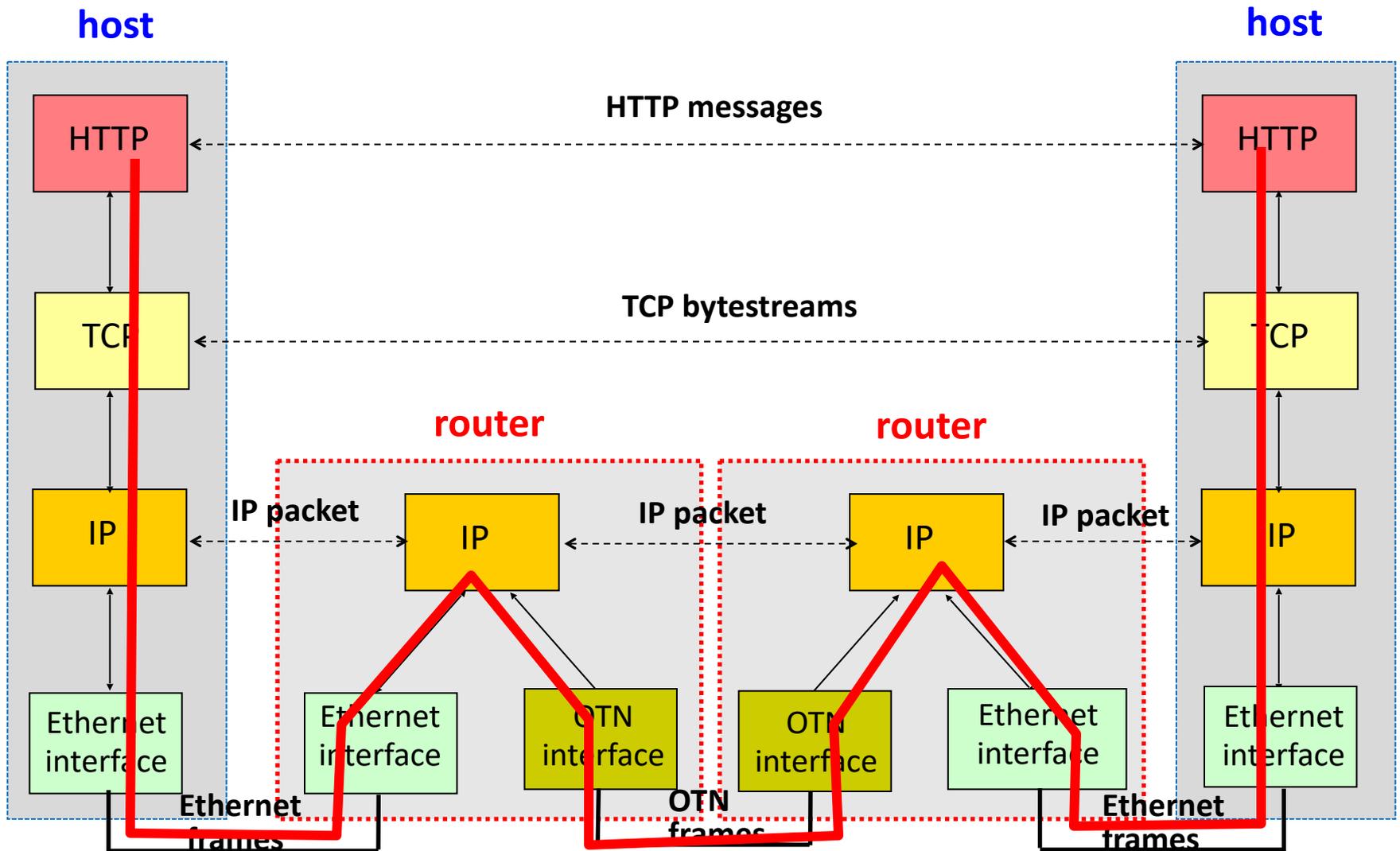


Packets contain multiple headers!

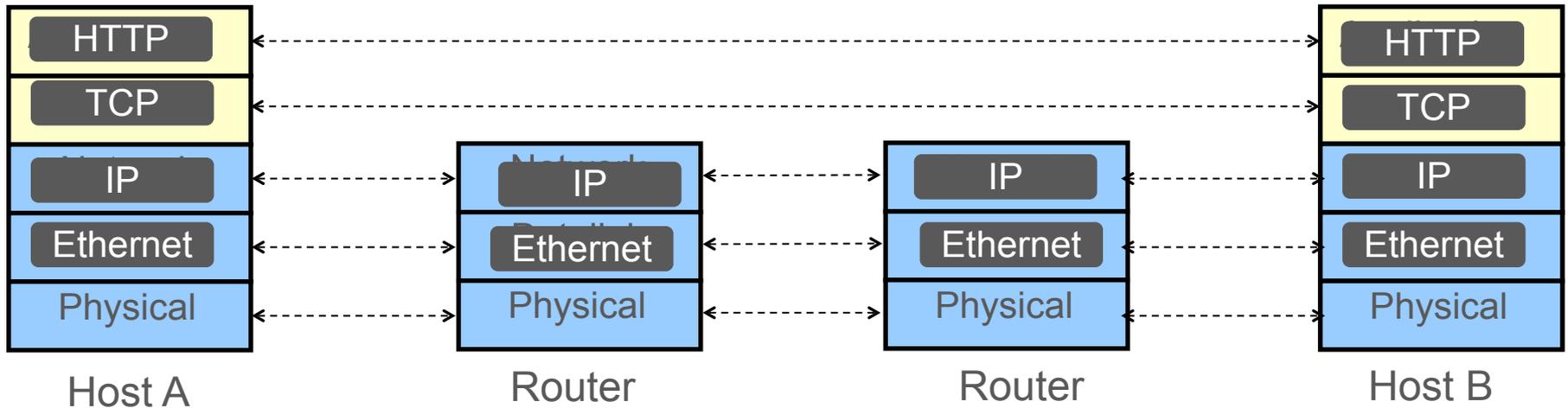


On wire: packet has data + headers from all layers!

# Complicated protocol diagram

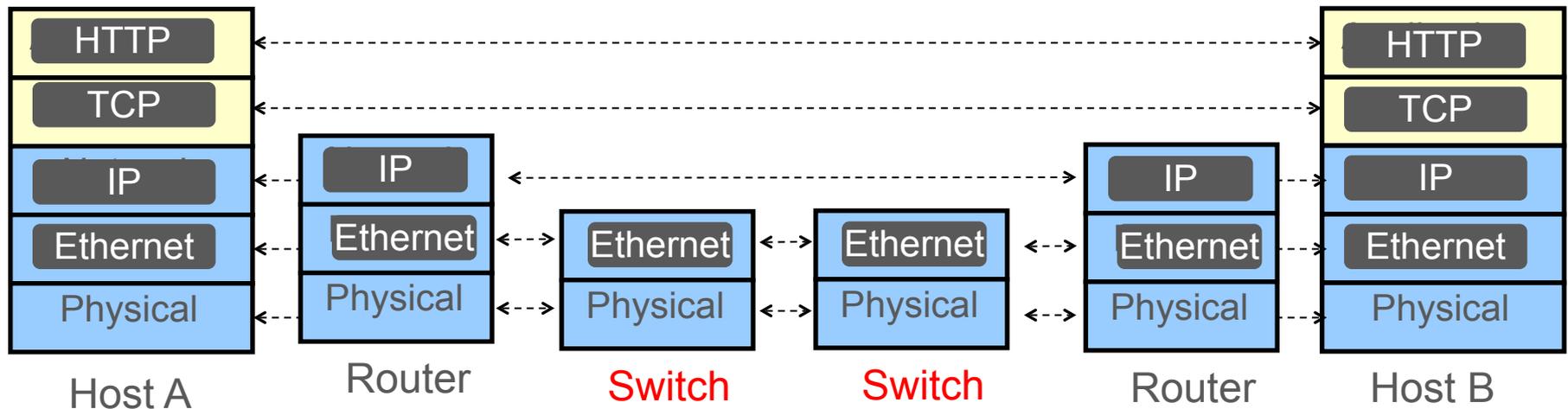


# A side note: switches vs. routers

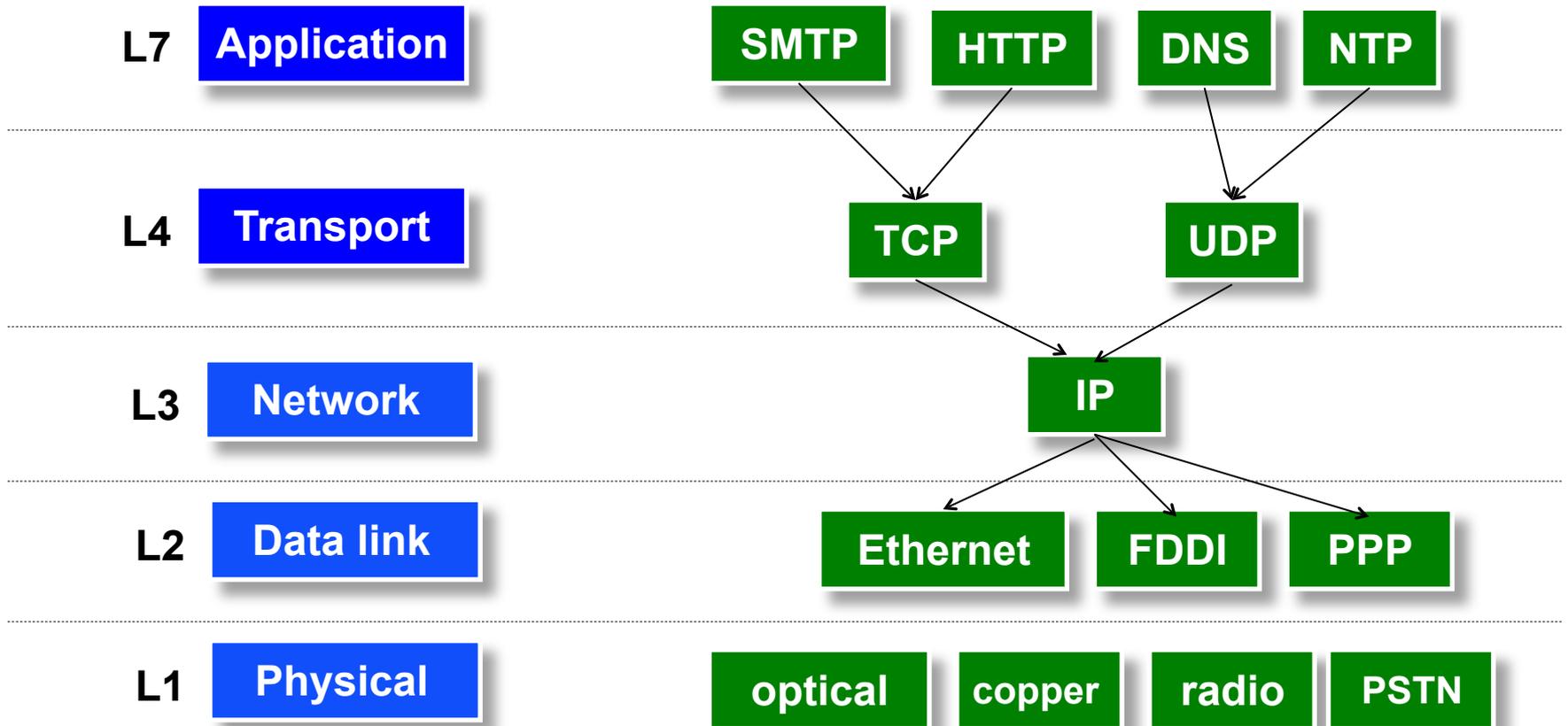


# A side note: switches vs. routers

- Historically: switches implemented L1, L2 and routers L1, L2, L3
- These days, most switches also implement L3 hence we use the term switches and router interchangeably



# Review



# Review

L7 **Application**

---

L4 **Transport**

---

L3 **Network**

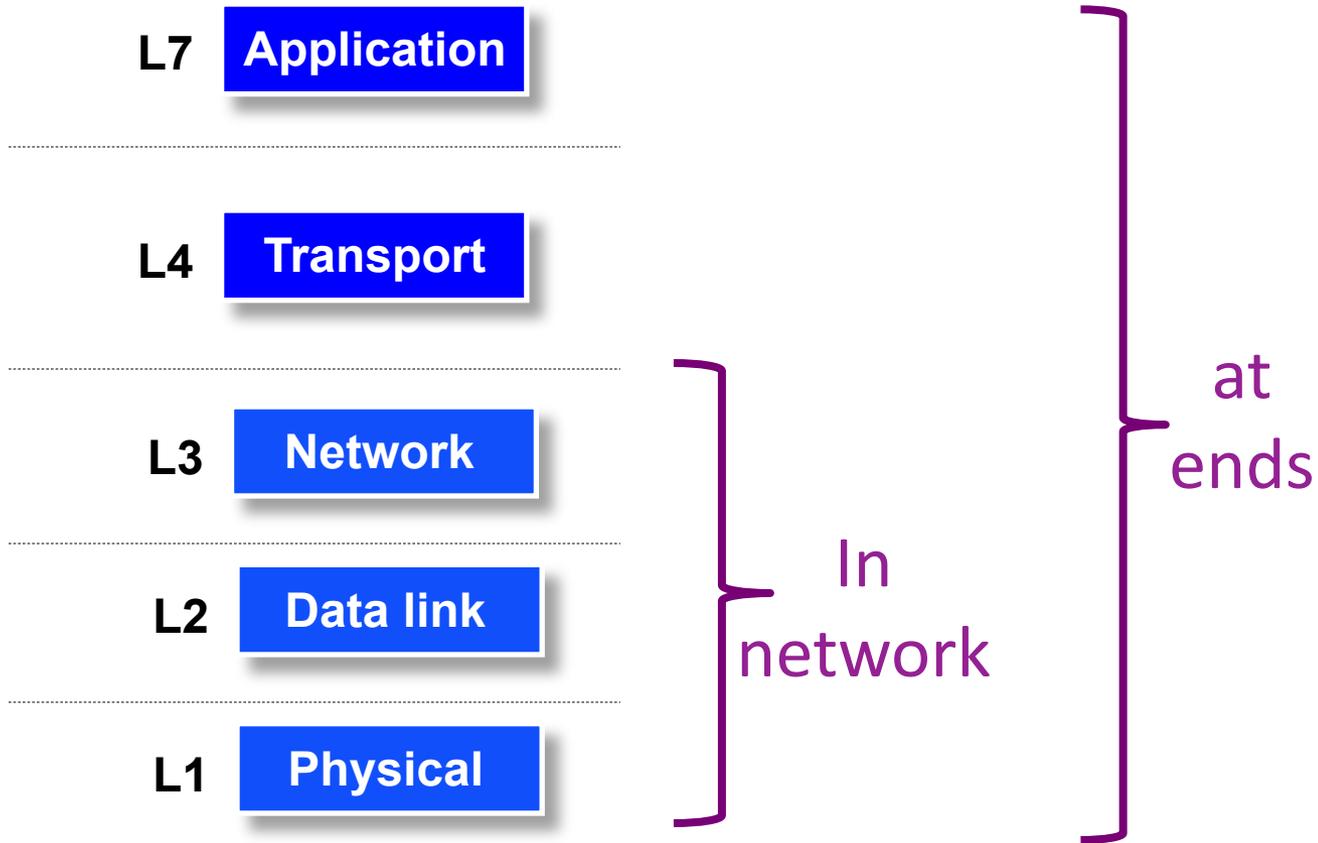
---

L2 **Data link**

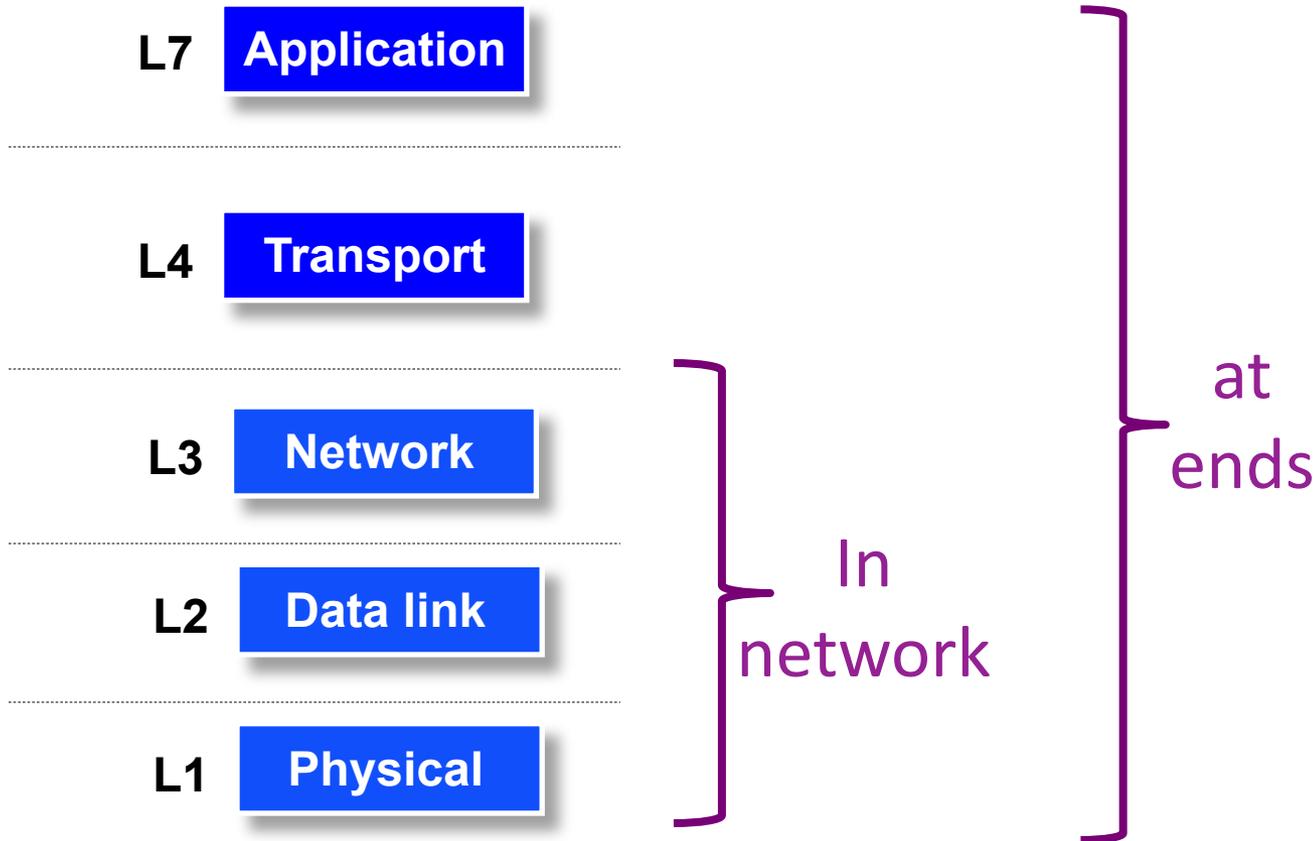
---

L1 **Physical**

# Review



# Review



Why is *this* assignment of tasks good?

# Architectural Wisdom

- David D. Clark
  - Chief protocol architect for the Internet in the 80s



# Architectural Wisdom

- David D. Clark
  - Chief protocol architect for the Internet in the 80s
- Co-authored two classics
  - “End-to-End Arguments in System Design” (1981)
  - “The Design Philosophy of the DARPA Internet Protocols” (1988)



# Architectural Wisdom

- David D. Clark
  - Chief protocol architect for the Internet in the 80s
- Co-authored two classics
  - “End-to-End Arguments in System Design” (1981)
  - “The Design Philosophy of the DARPA Internet Protocols” (1988)
- Articulates the rationale underlying the Internet’s arch.



# The End-to-End Principle

# The End-to-End Principle

- Guides the debate about what functionality the network does or doesn't implement

# The End-to-End Principle

- Guides the debate about what functionality the network does or doesn't implement
- Today: should we implement reliability in the network?

# The End-to-End Principle

- Guides the debate about what functionality the network does or doesn't implement
- Today: should we implement reliability in the network?

What does it mean for the network to implement reliability?

# The End-to-End Principle

- Guides the debate about what functionality the network does or doesn't implement
- Today: should we implement reliability in the network?



What does it mean for the network to implement reliability?

# The End-to-End Principle

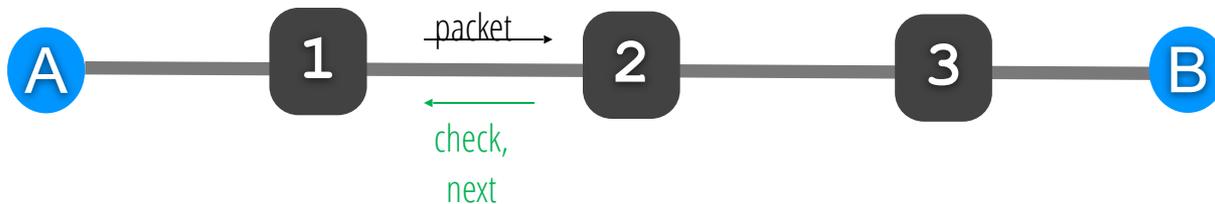
- Guides the debate about what functionality the network does or doesn't implement
- Today: should we implement reliability in the network?



What does it mean for the network to implement reliability?

# The End-to-End Principle

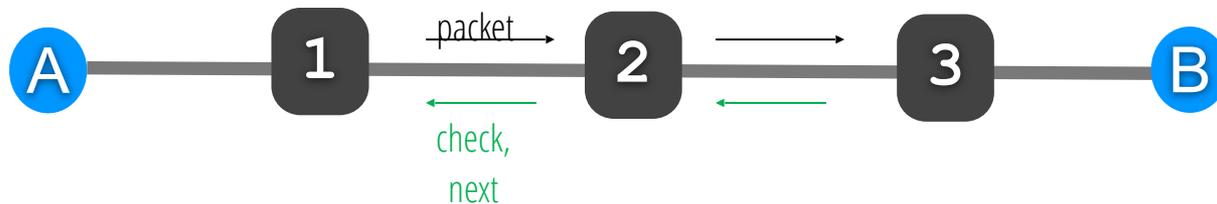
- Guides the debate about what functionality the network does or doesn't implement
- Today: should we implement reliability in the network?



What does it mean for the network to implement reliability?

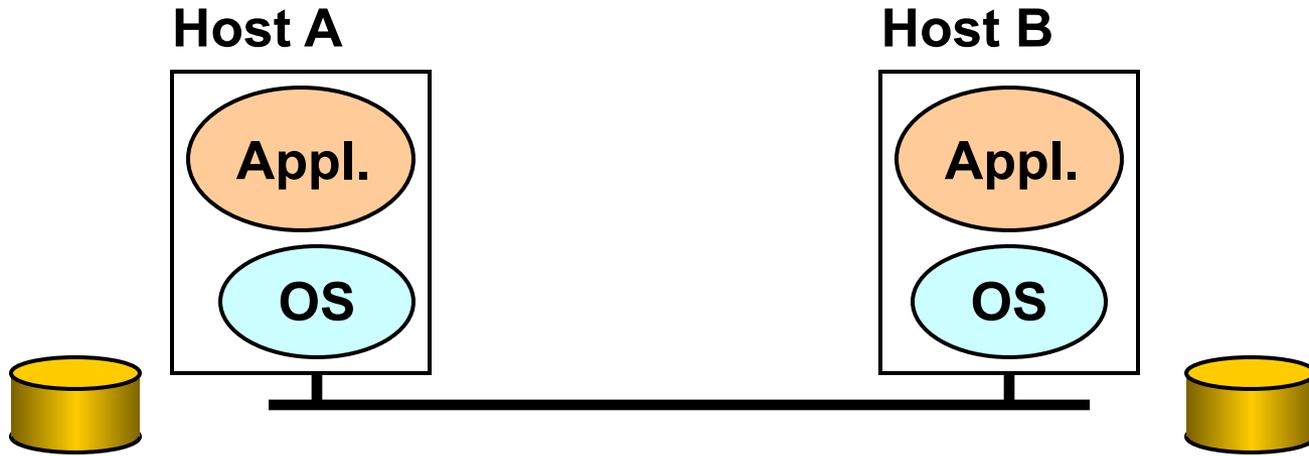
# The End-to-End Principle

- Guides the debate about what functionality the network does or doesn't implement
- Today: should we implement reliability in the network?

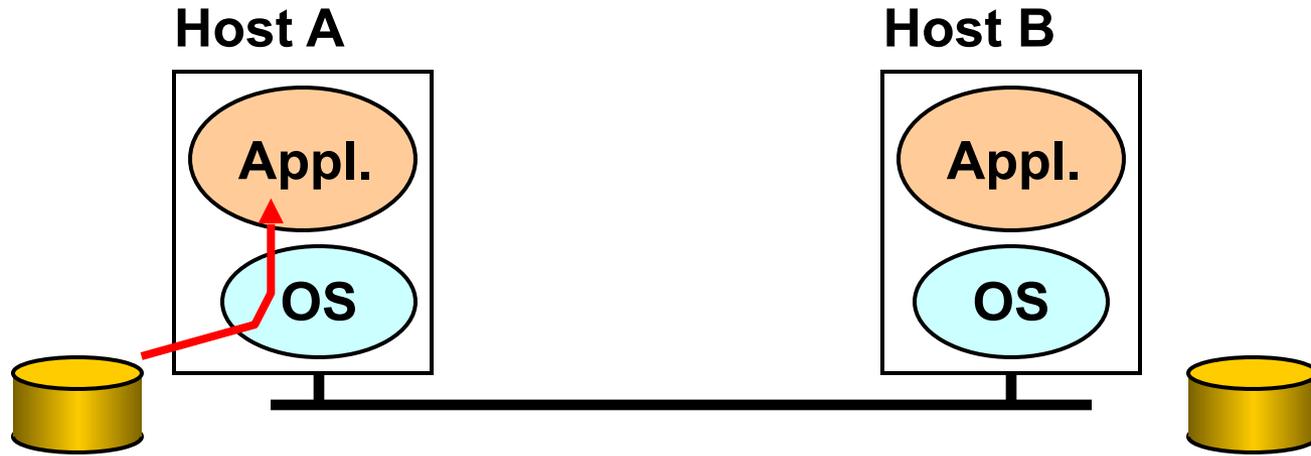


What does it mean for the network to implement reliability?

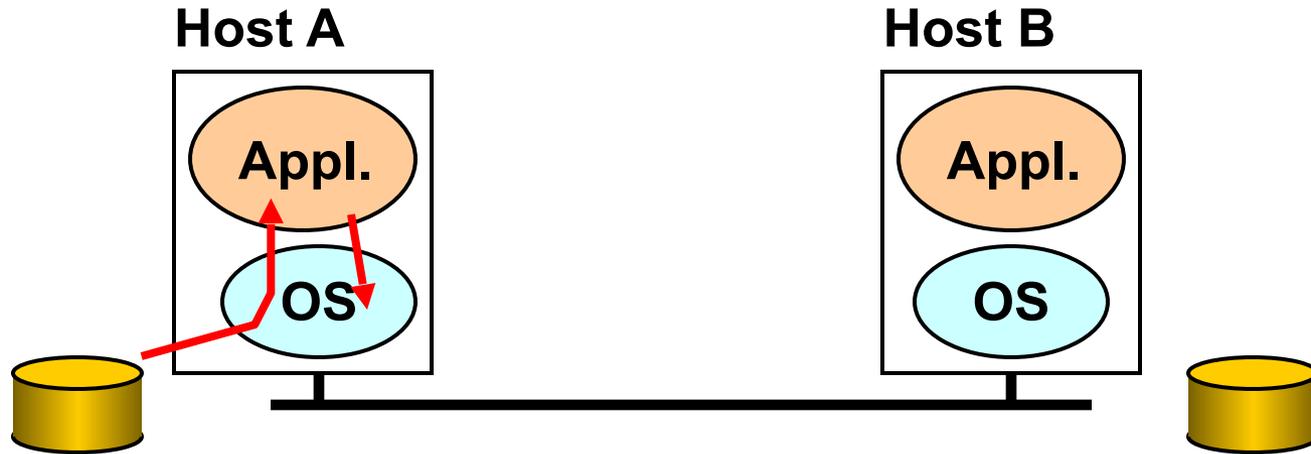
# Example: Reliable File Transfer



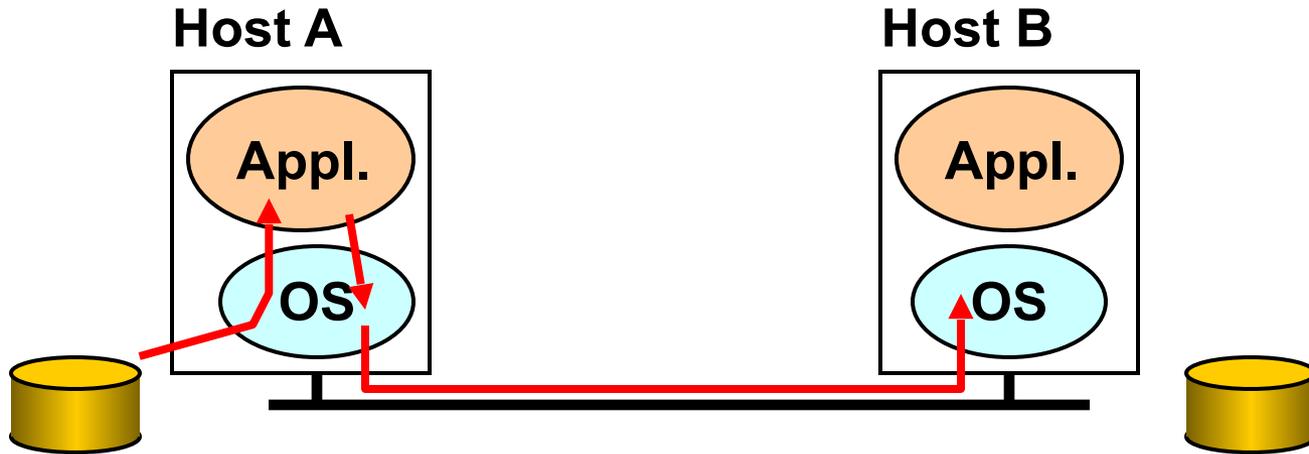
# Example: Reliable File Transfer



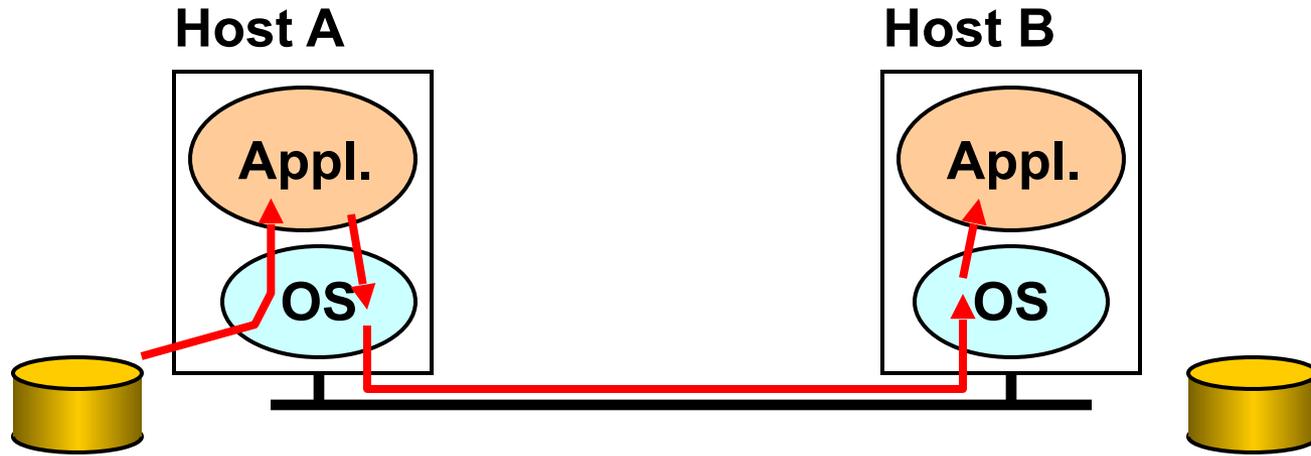
# Example: Reliable File Transfer



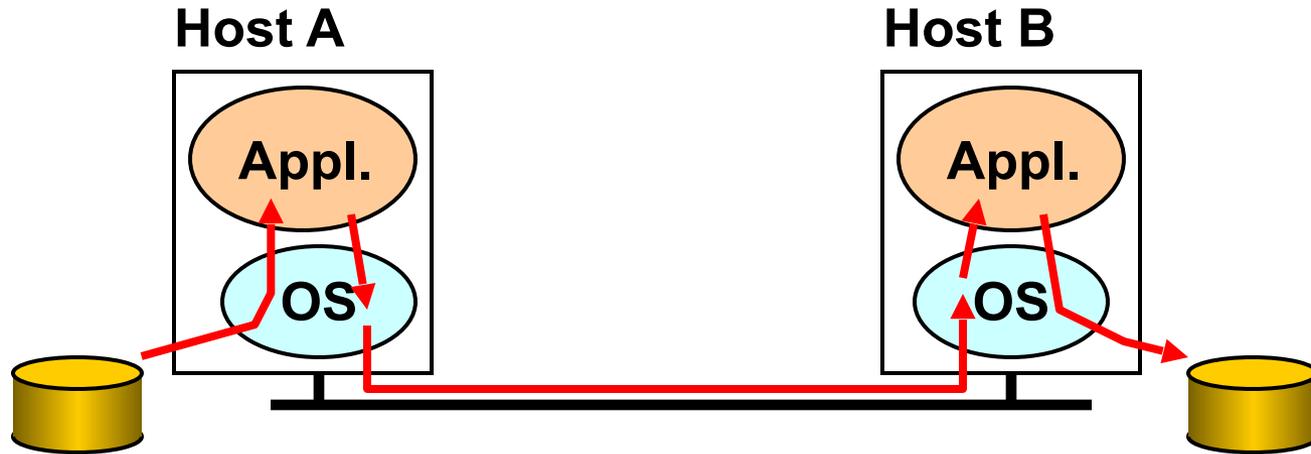
# Example: Reliable File Transfer



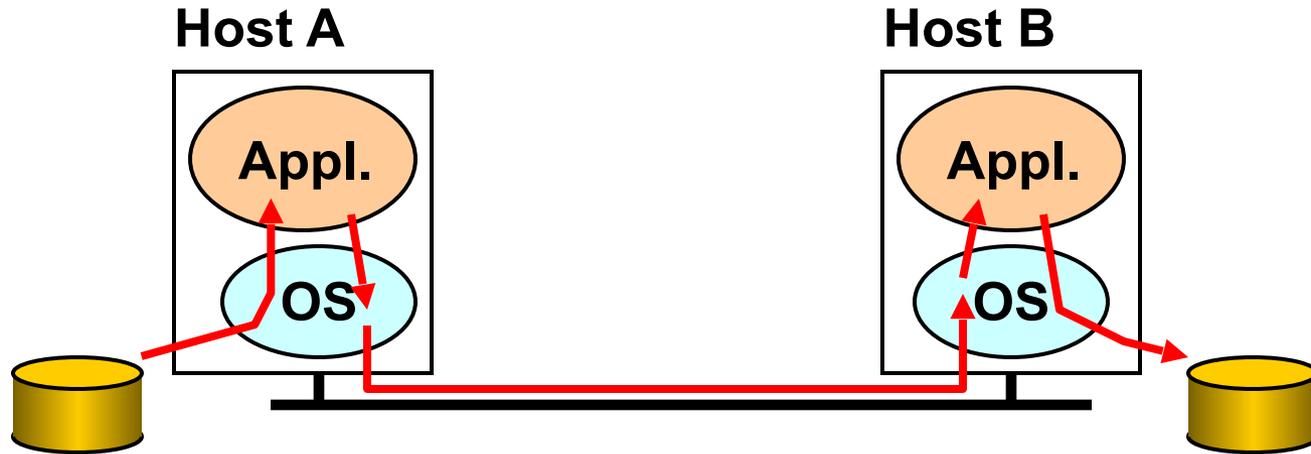
# Example: Reliable File Transfer



# Example: Reliable File Transfer

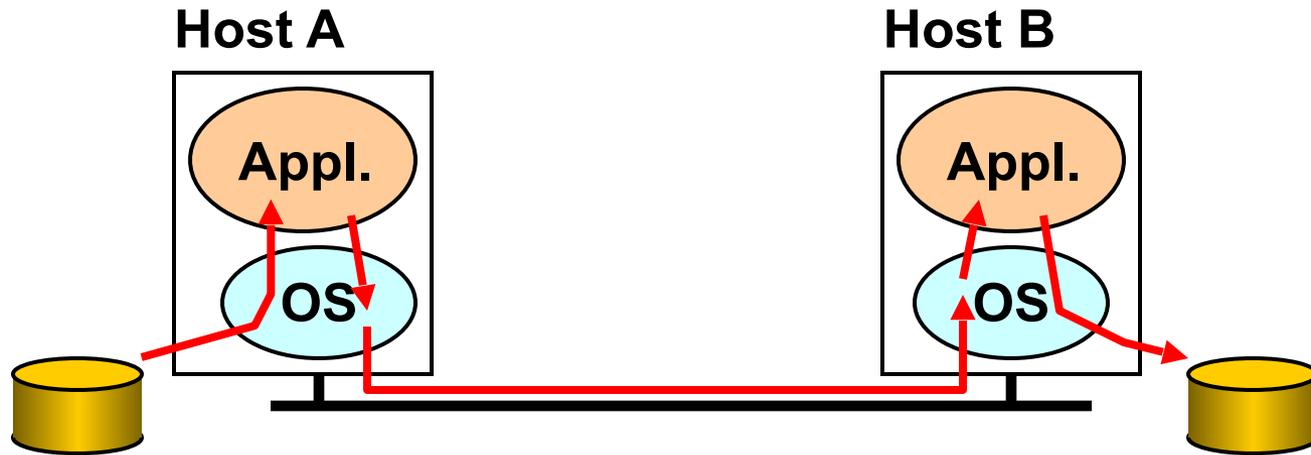


# Example: Reliable File Transfer



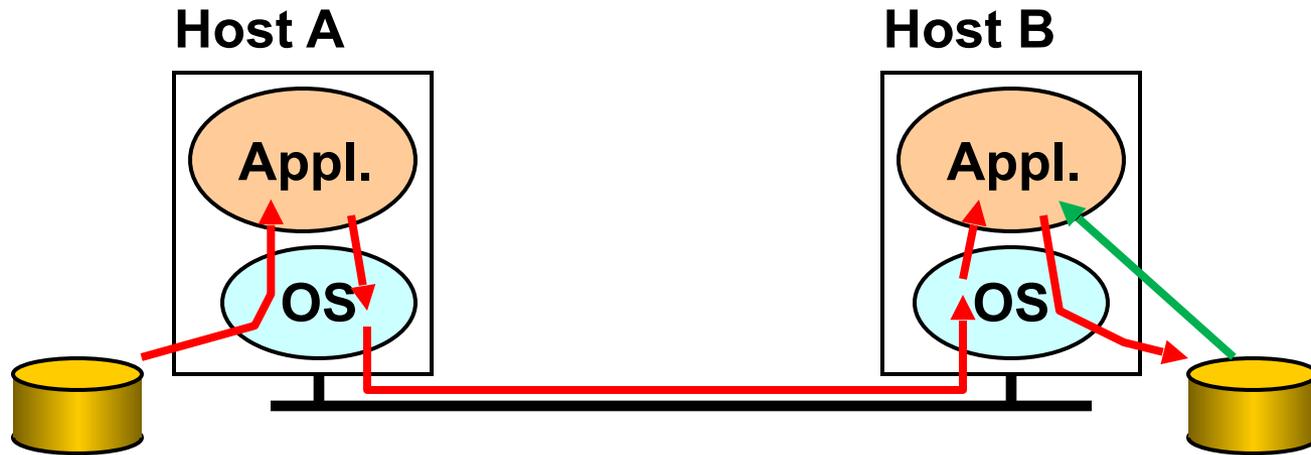
- Solution 1: implement reliability at each step  
(→ *network implements reliability*)

# Example: Reliable File Transfer



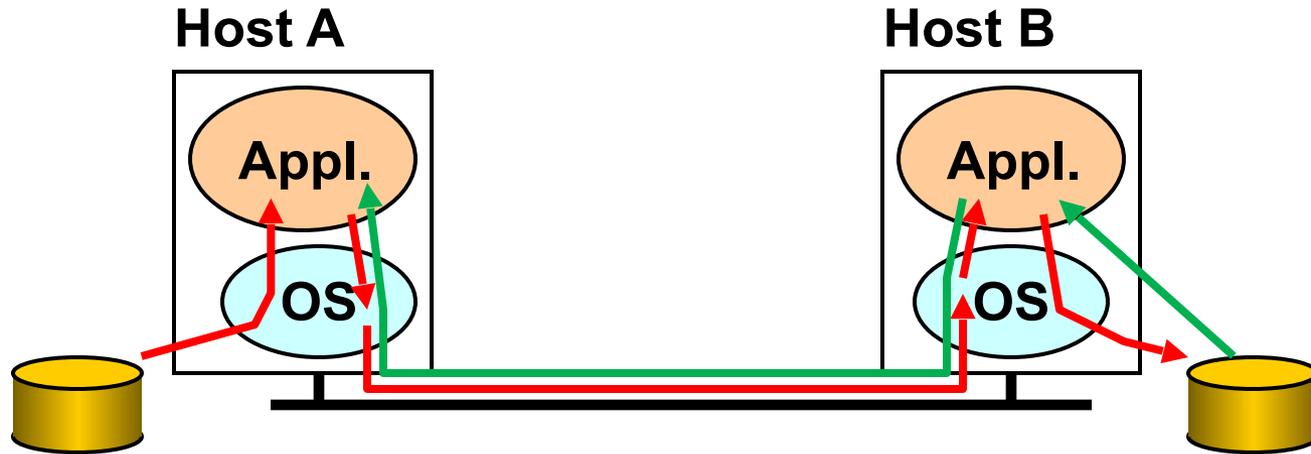
- Solution 1: implement reliability at each step  
(→ *network implements reliability*)
- Solution 2: **end-to-end check** and retry  
(→ *does not assume network is reliable*)

# Example: Reliable File Transfer



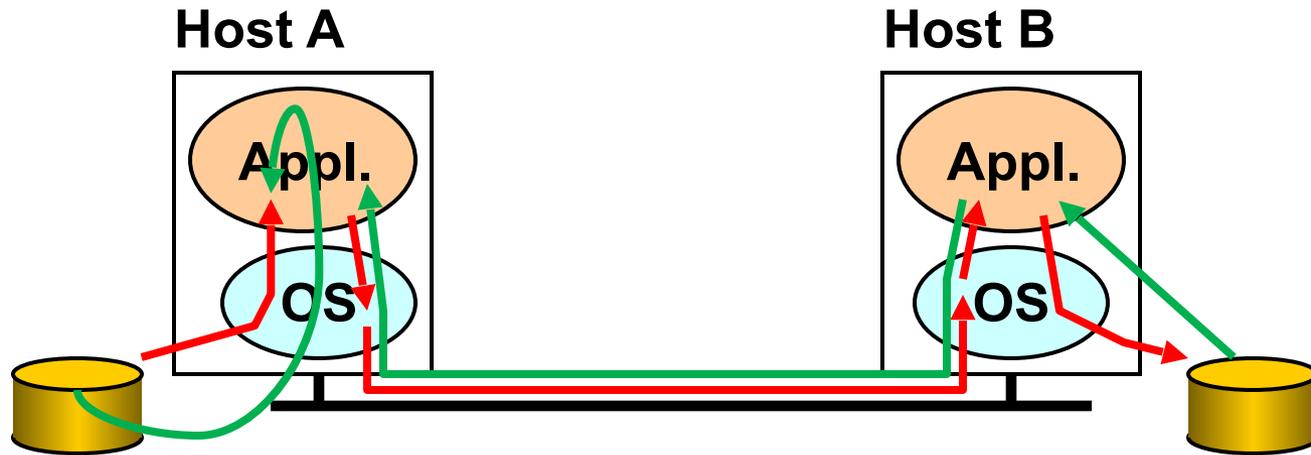
- Solution 1: implement reliability at each step  
(→ *network implements reliability*)
- Solution 2: **end-to-end check** and retry  
(→ *does not assume network is reliable*)

# Example: Reliable File Transfer



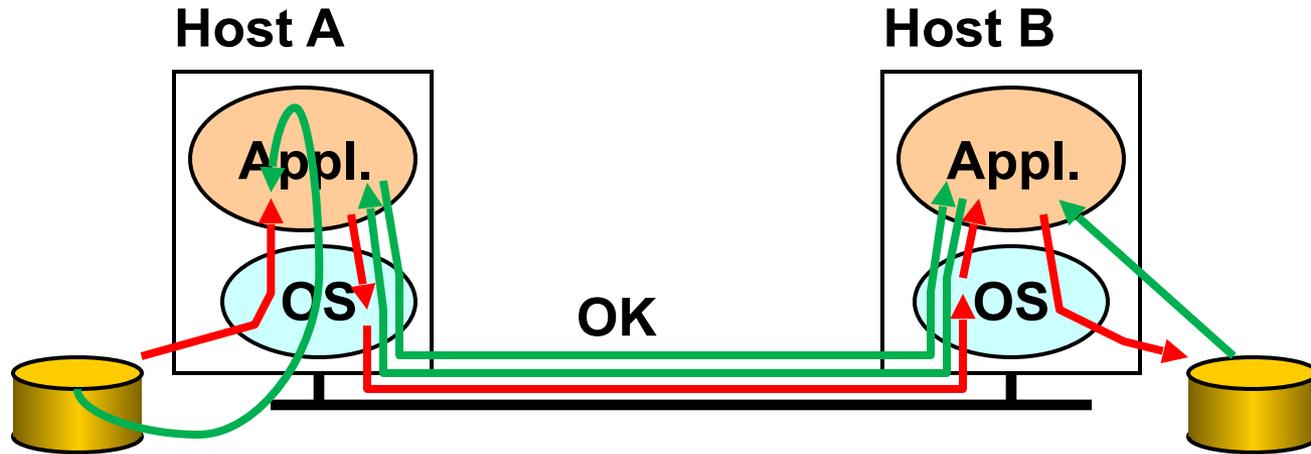
- Solution 1: implement reliability at each step  
(→ *network implements reliability*)
- Solution 2: **end-to-end check** and retry  
(→ *does not assume network is reliable*)

# Example: Reliable File Transfer



- Solution 1: implement reliability at each step  
(→ *network implements reliability*)
- Solution 2: **end-to-end check** and retry  
(→ *does not assume network is reliable*)

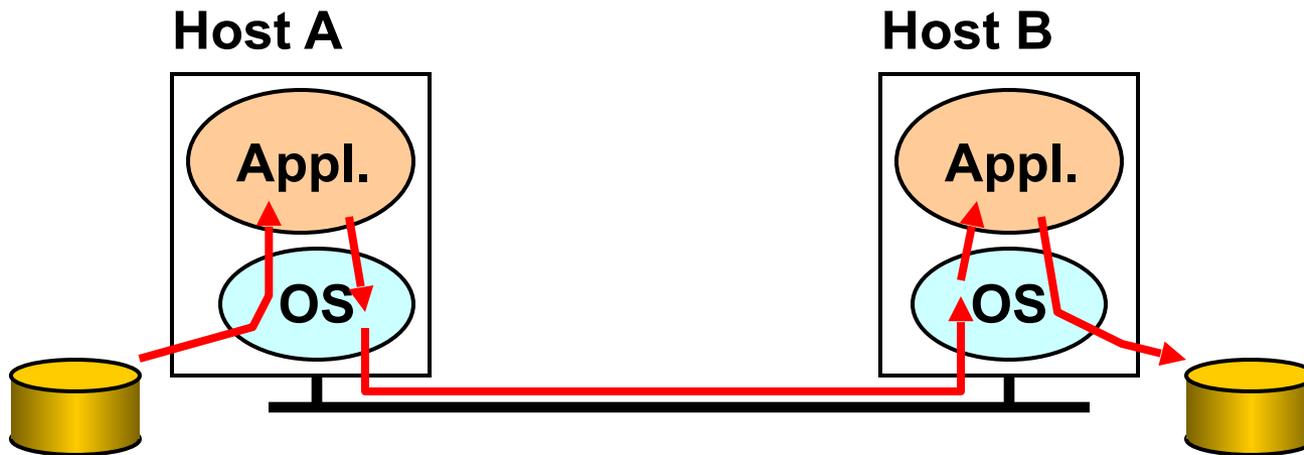
# Example: Reliable File Transfer



- Solution 1: implement reliability at each step  
(→ *network implements reliability*)
- Solution 2: **end-to-end check** and retry  
(→ *does not assume network is reliable*)

# Correctness

- Solution 1 cannot be made perfectly reliable
  - What happens if a component fails between two steps?
  - What happens if a component has a bug?



# Correctness

- Solution 1 cannot be made perfectly reliable
  - Receiver has to do the end-to-end check anyway

# Correctness

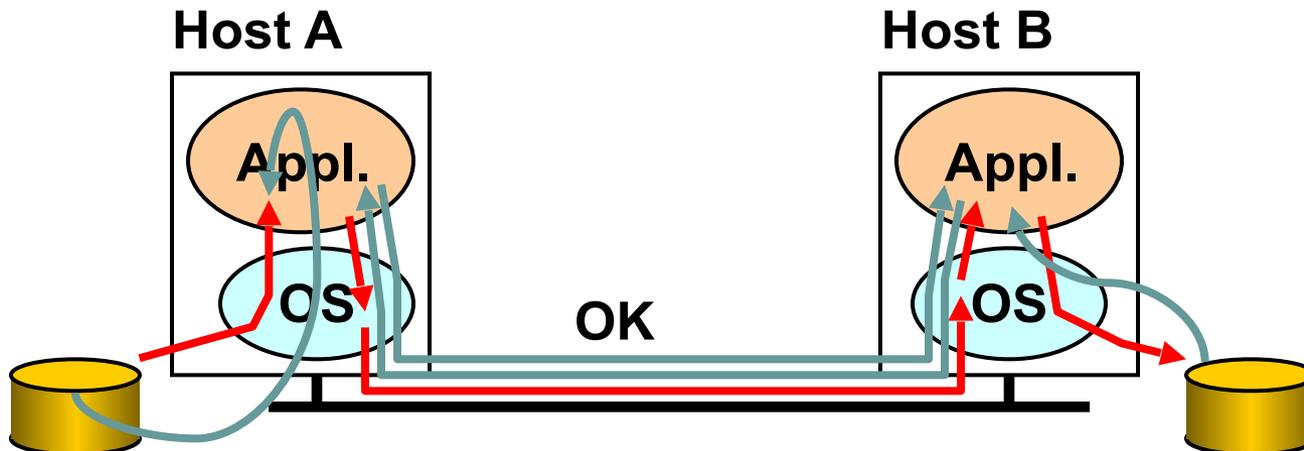
- Solution 1 cannot be made perfectly reliable
  - Receiver has to do the end-to-end check anyway
  - If not, the endpoints might be left in an incorrect state

# Correctness

- Solution 1 cannot be made perfectly reliable
  - Receiver has to do the end-to-end check anyway
  - If not, the endpoints might be left in an incorrect state
- Solution 2 can also fail but will never leave the endpoints in an incorrect state
  - Host B will never accept a corrupted file

# Correctness

- Solution 1 cannot be made perfectly reliable
  - Receiver has to do the end-to-end check anyway
  - If not, the endpoints might be left in an incorrect state
- Solution 2 can also fail but will never leave the endpoints in an incorrect state
  - Host B will never accept a corrupted file



# Impact of Network Failures

# Impact of Network Failures

- Solution 1
  - Network failures/bugs impact endpoint *semantics*
  - Requires endpoints trust other elements!

# Impact of Network Failures

- Solution 1
  - Network failures/bugs impact endpoint *semantics*
  - Requires endpoints trust other elements!
- Solution 2
  - Endpoint semantics decoupled from network failure
  - Endpoint only relies on what it can control!

# End-to-end argument: Intuition

# End-to-end argument: Intuition

- Some application requirements can only be correctly implemented **end-to-end**
  - reliability, security, *etc.*

# End-to-end argument: Intuition

- Some application requirements can only be correctly implemented **end-to-end**
  - reliability, security, *etc.*
- End-systems
  - **Must** do so, for correctness
  - **Can** satisfy the requirement without network's help

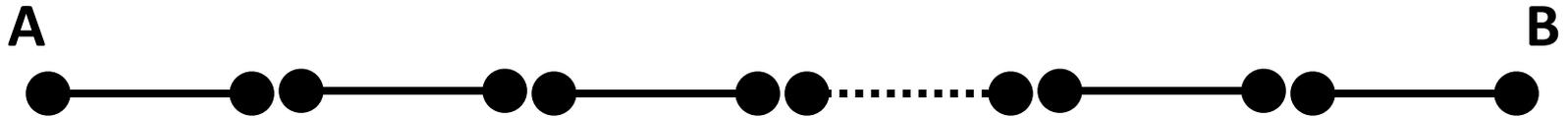
# End-to-end argument: Intuition

- Some application requirements can only be correctly implemented **end-to-end**
  - reliability, security, *etc.*
- End-systems
  - **Must** do so, for correctness
  - **Can** satisfy the requirement without network's help
- Implementing these functions in the network is unnecessary and adds complexity to the network

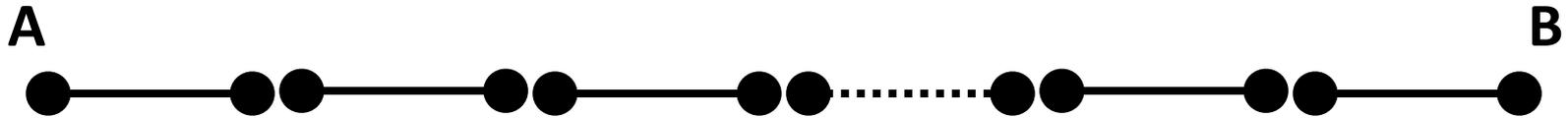
# So...

- Should you ever implement reliability in network?
  - I.e., in addition to doing so in the hosts

# Performance

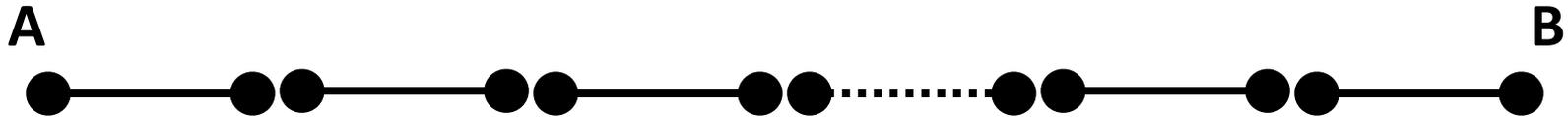


# Performance



- If each link drops packets 10% of the time, and we have 10 links, then E2E failure rate is ~65%

# Performance



- If each link drops packets 10% of the time, and we have 10 links, then E2E failure rate is ~65%
- What if the link implemented two retransmissions?
  - Per-link drop rate reduced to 0.1%, E2E error rate is ~1%

# Performance

- Should you ever implement reliability in network?
  - I.e., in addition to doing so in the hosts

# Performance

- Should you ever implement reliability in network?
  - I.e., in addition to doing so in the hosts
- Perhaps, as a performance optimization

# Performance

- Should you ever implement reliability in network?
  - I.e., in addition to doing so in the hosts
- Perhaps, as a performance optimization
  - Need for it must be evaluated on a case-by-case basis

# Recap: End-to-End Principle

# Recap: End-to-End Principle

Implementing this function (reliability) in the network:

# Recap: End-to-End Principle

Implementing this function (reliability) in the network:

- Doesn't reduce host implementation complexity

# Recap: End-to-End Principle

Implementing this function (reliability) in the network:

- Doesn't reduce host implementation complexity
- Does increase network complexity

# Recap: End-to-End Principle

Implementing this function (reliability) in the network:

- Doesn't reduce host implementation complexity
- Does increase network complexity
- Imposes overhead on all applications

# Recap: End-to-End Principle

Implementing this function (reliability) in the network:

- Doesn't reduce host implementation complexity
- Does increase network complexity
- Imposes overhead on all applications
- However, implementing in network *can* enhance performance in some cases
  - E.g., very lossy link

# The end-to-end argument in Clark's words

*“The function in question can completely and correctly be implemented only with the knowledge and help of the application at the end points. Therefore, providing that function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)”*

**Recap: architectural wisdom  
(the "how")**

## Recap: architectural wisdom (the "how")

- How to decompose system into modules?
  - Layering

## Recap: architectural wisdom (the "how")

- How to decompose system into modules?
  - Layering
- Where are layers implemented?
  - End hosts implement all layers (L1-L7)
  - Network implements only layers (L1-L3)

## Recap: architectural wisdom (the "how")

- How to decompose system into modules?
  - Layering
- Where are layers implemented?
  - End hosts implement all layers (L1-L7)
  - Network implements only layers (L1-L3)
- One unifying protocol at the network layer
  - Internet Protocol (IP)

# Recap: architectural wisdom (the "why")

# Recap: architectural wisdom (the "why")

- **Layering** provided a clean separation of concerns
  - And hence enabled innovation!

# Recap: architectural wisdom (the "why")

- **Layering** provided a clean separation of concerns
  - And hence enabled innovation!
- **End-to-end principle** kept unnecessary state and functionality out of the network
  - And hence allowed the Internet to scale!

**Questions?**