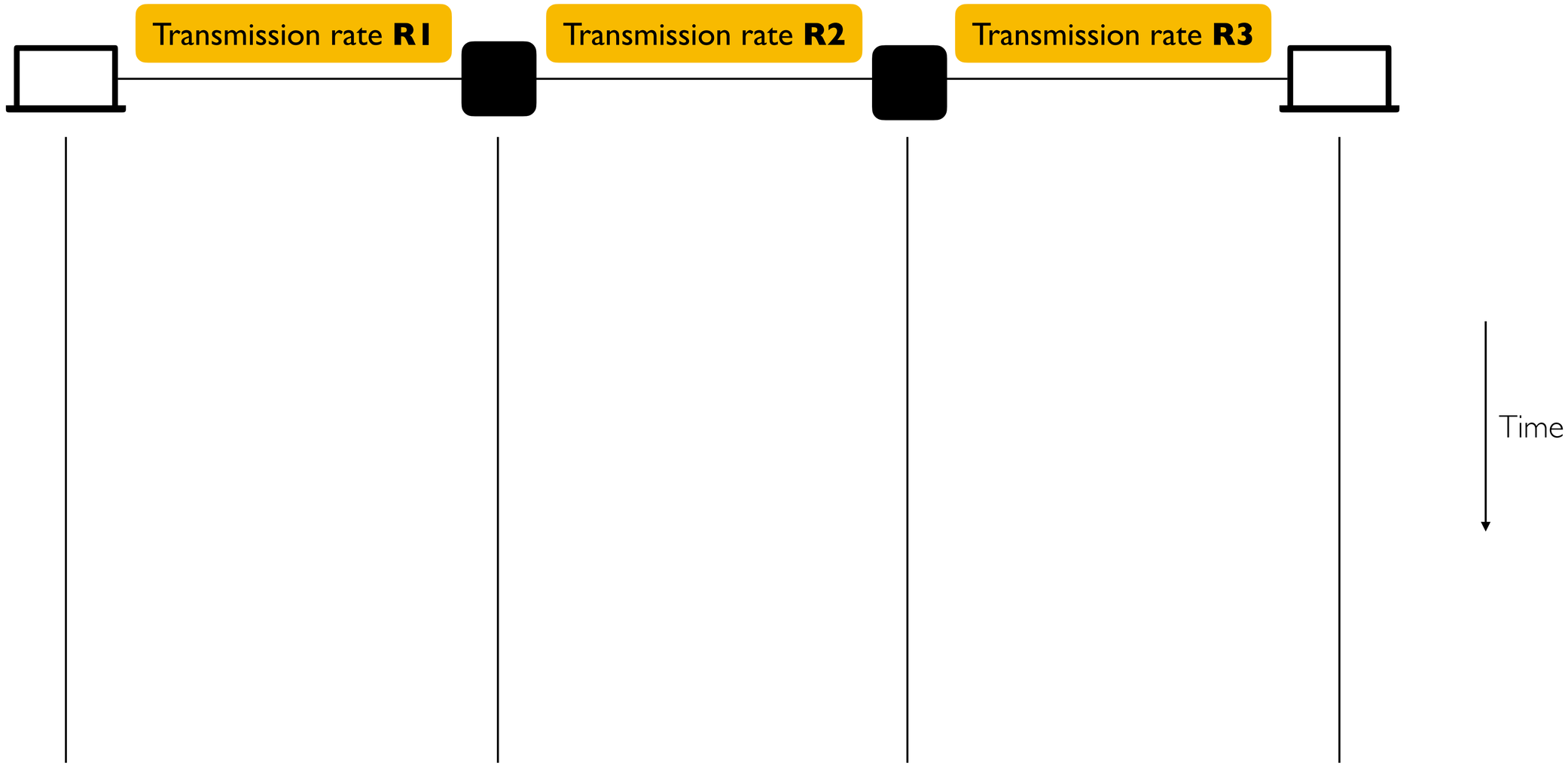
How is communication organized? (Contd.)

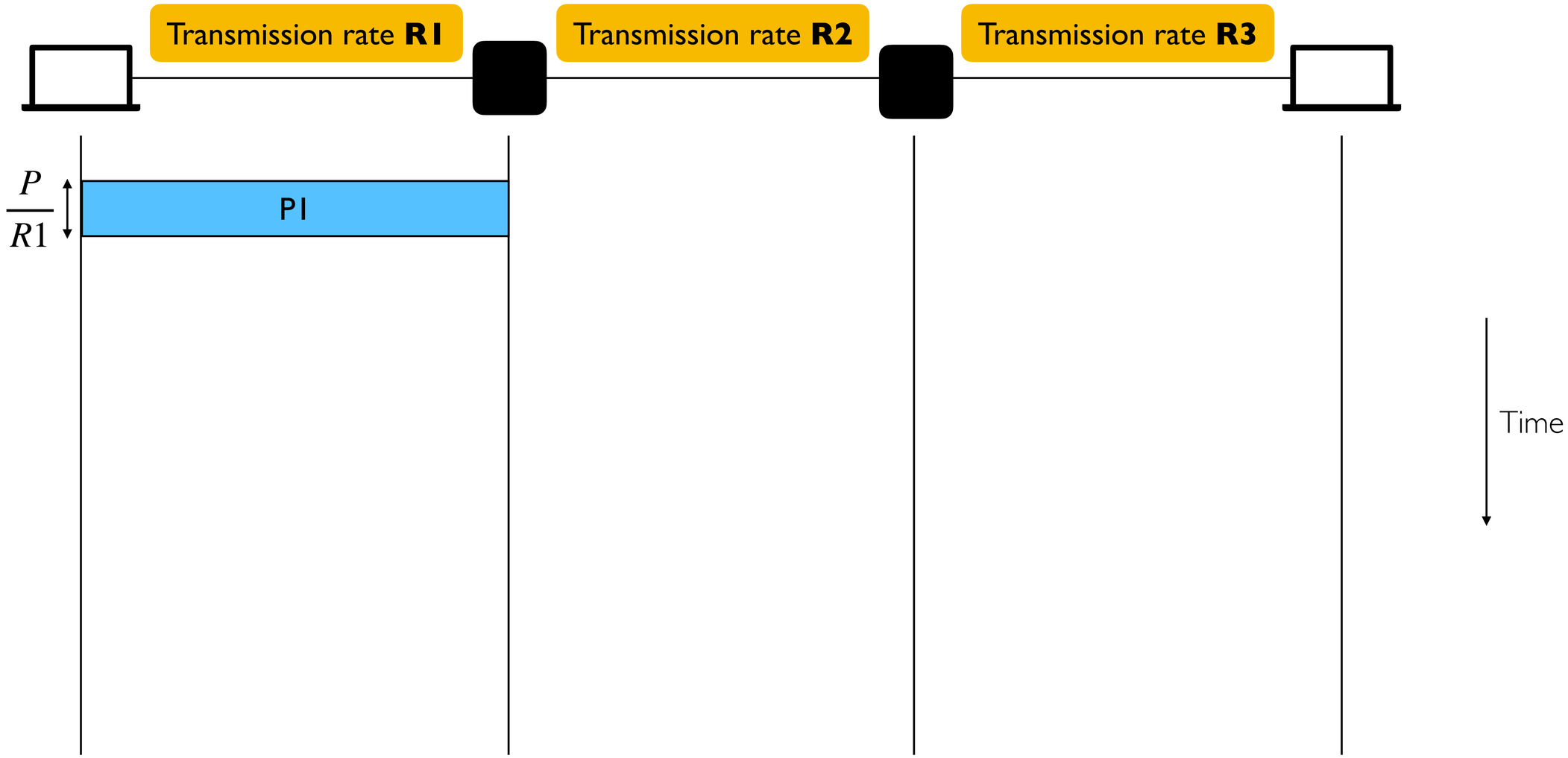
CPSC 433/533, Spring 2021

Anurag Khandelwal

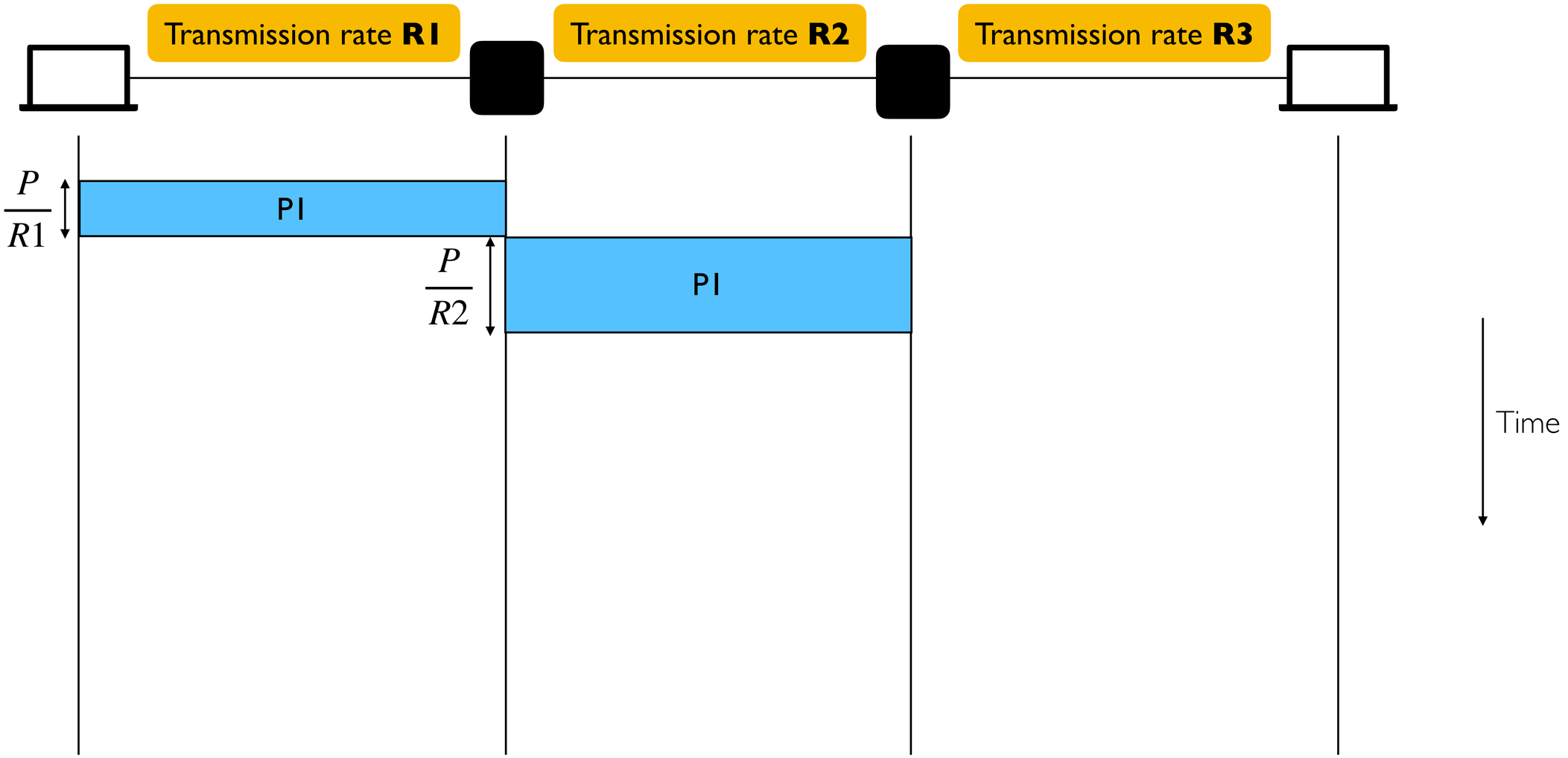
But before that… Throughput Calculation



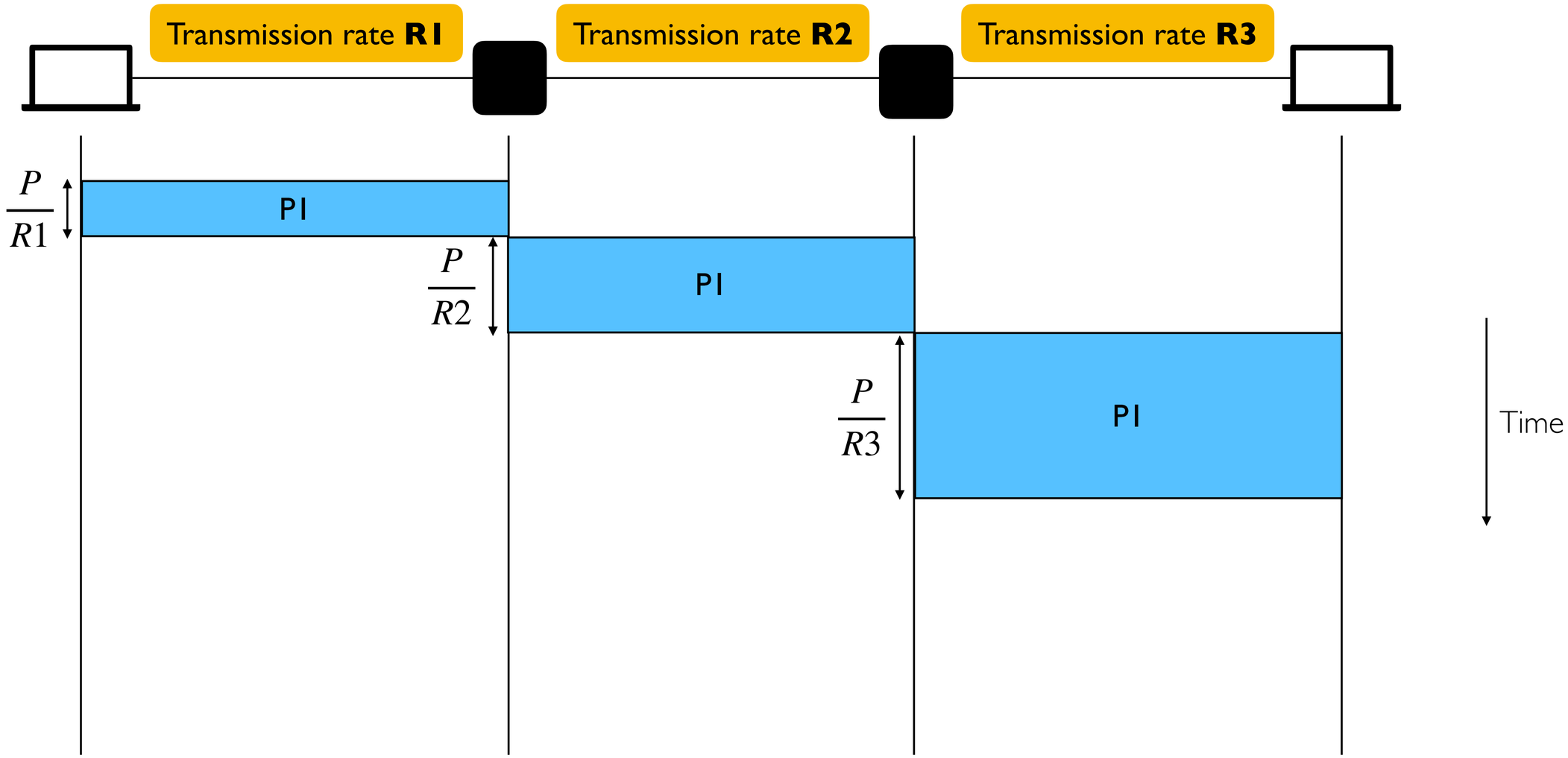
But before that… Throughput Calculation



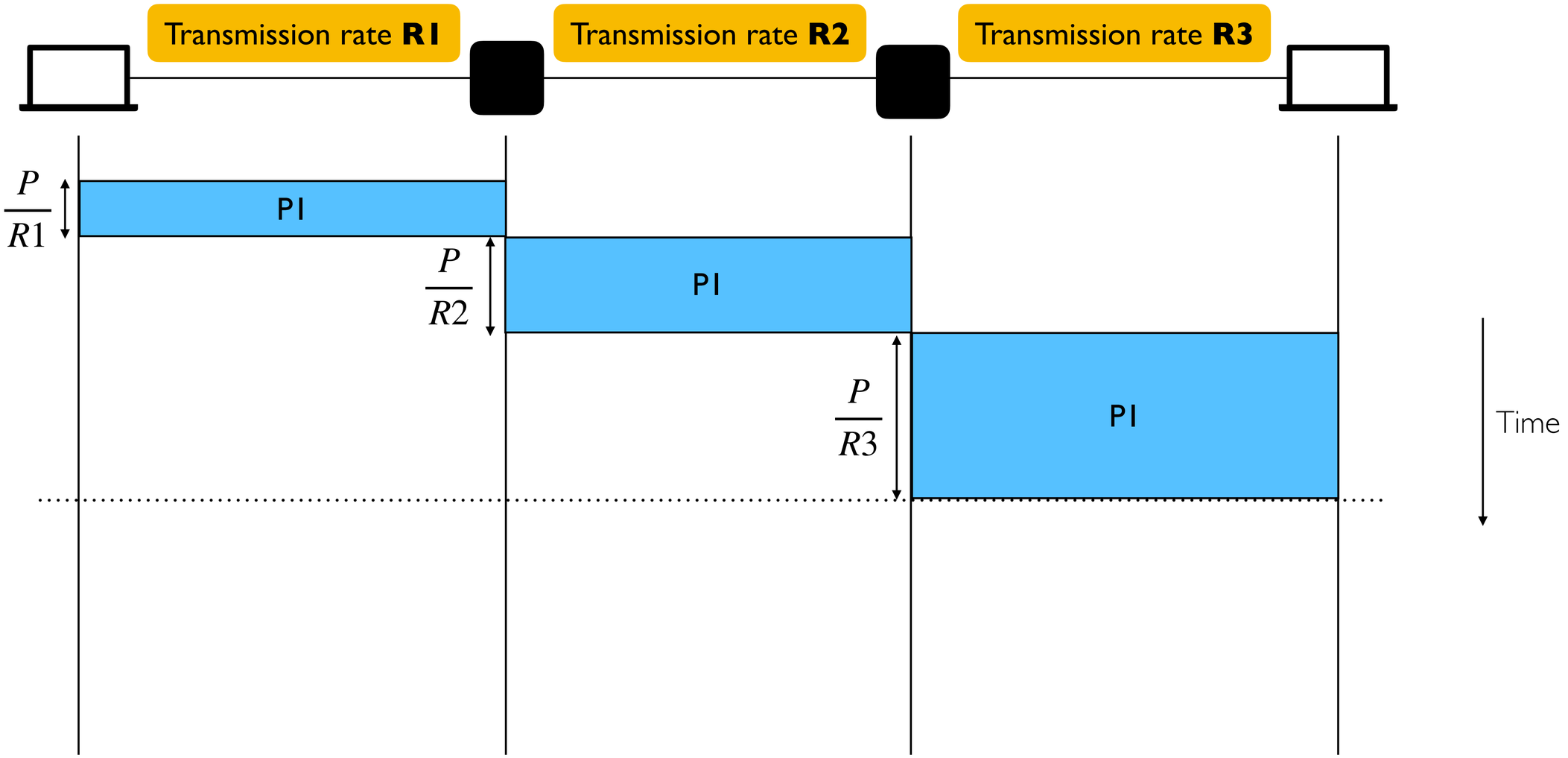
But before that… Throughput Calculation



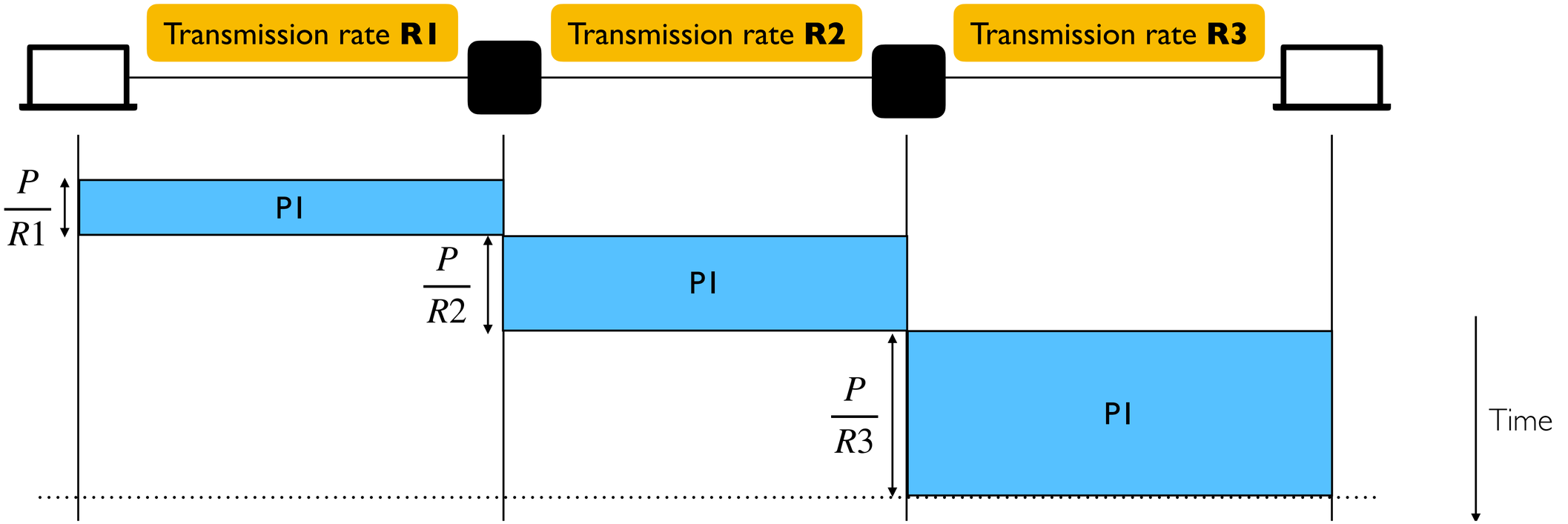
But before that… Throughput Calculation



But before that… Throughput Calculation

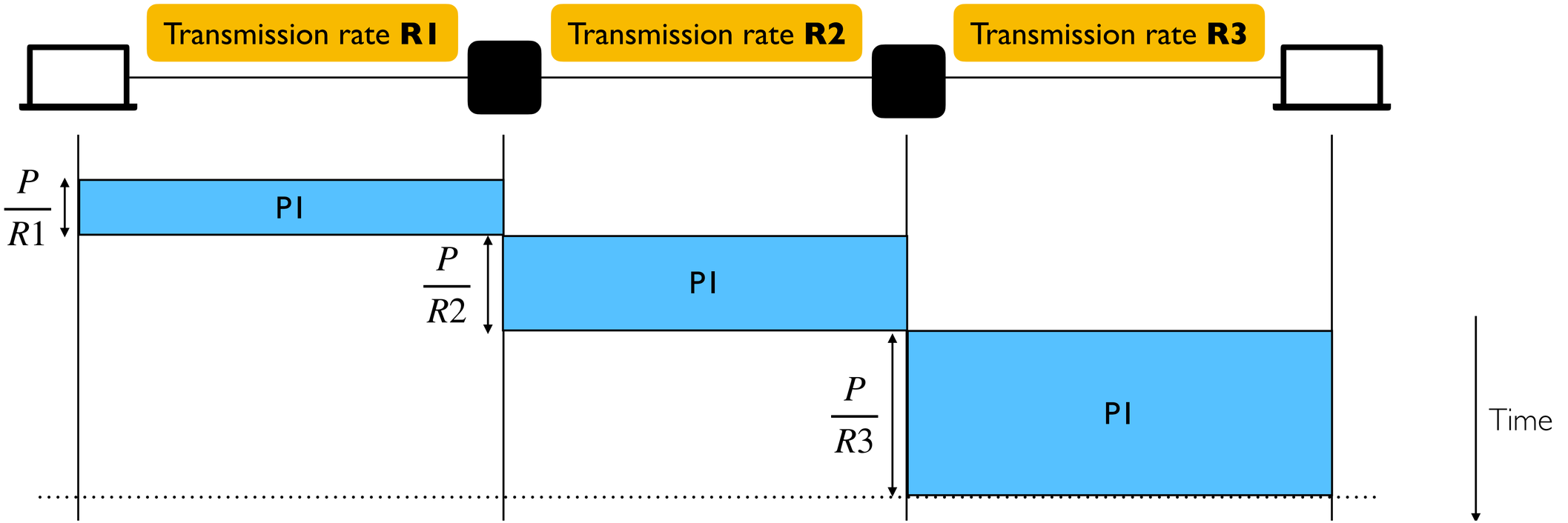


But before that… Throughput Calculation



Ignoring propagation, queueing and processing delays, packet delay for a packet of P bits $= \frac{P}{R1} + \frac{P}{R2} + \frac{P}{R3}$

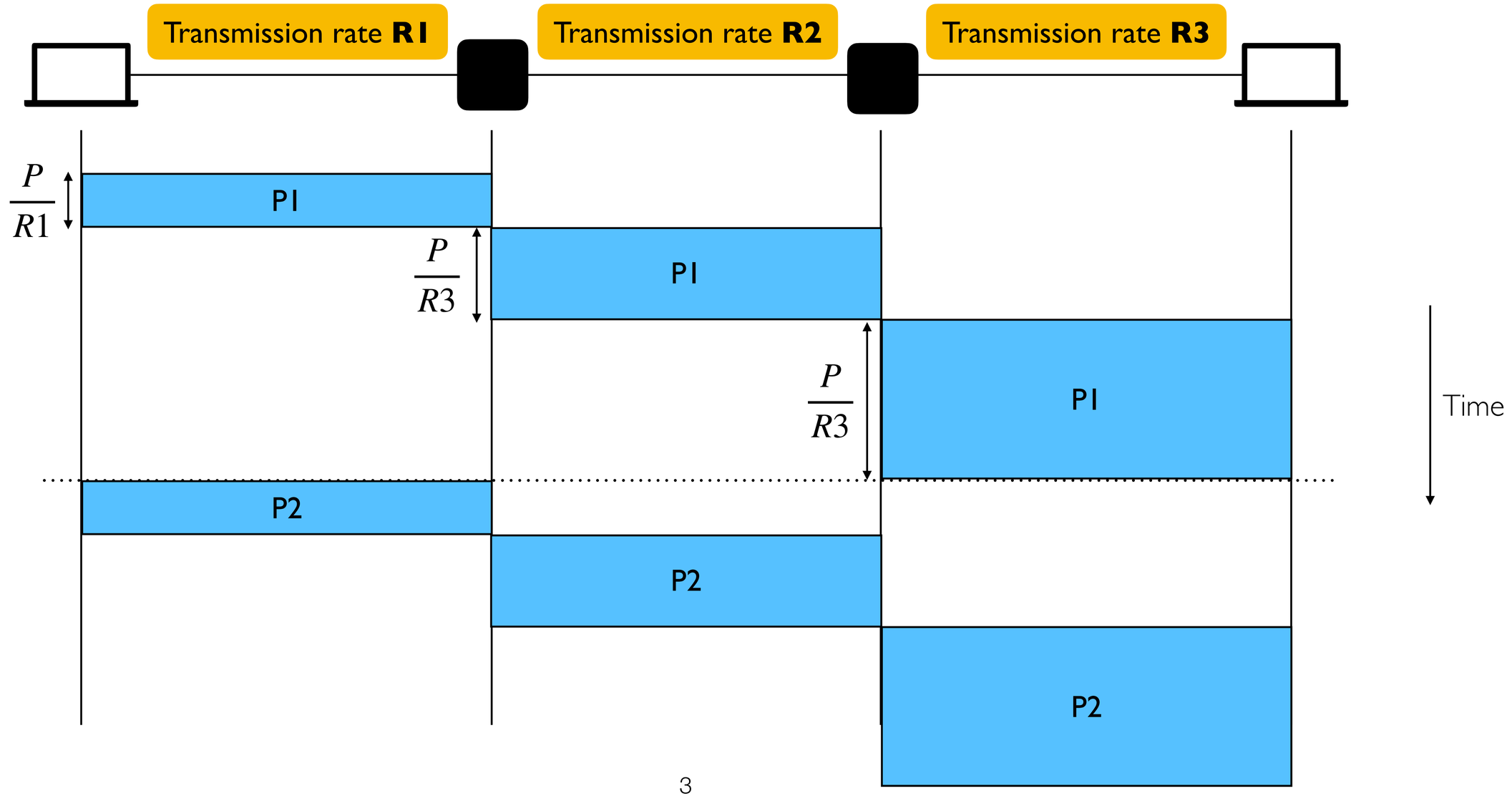
But before that… Throughput Calculation



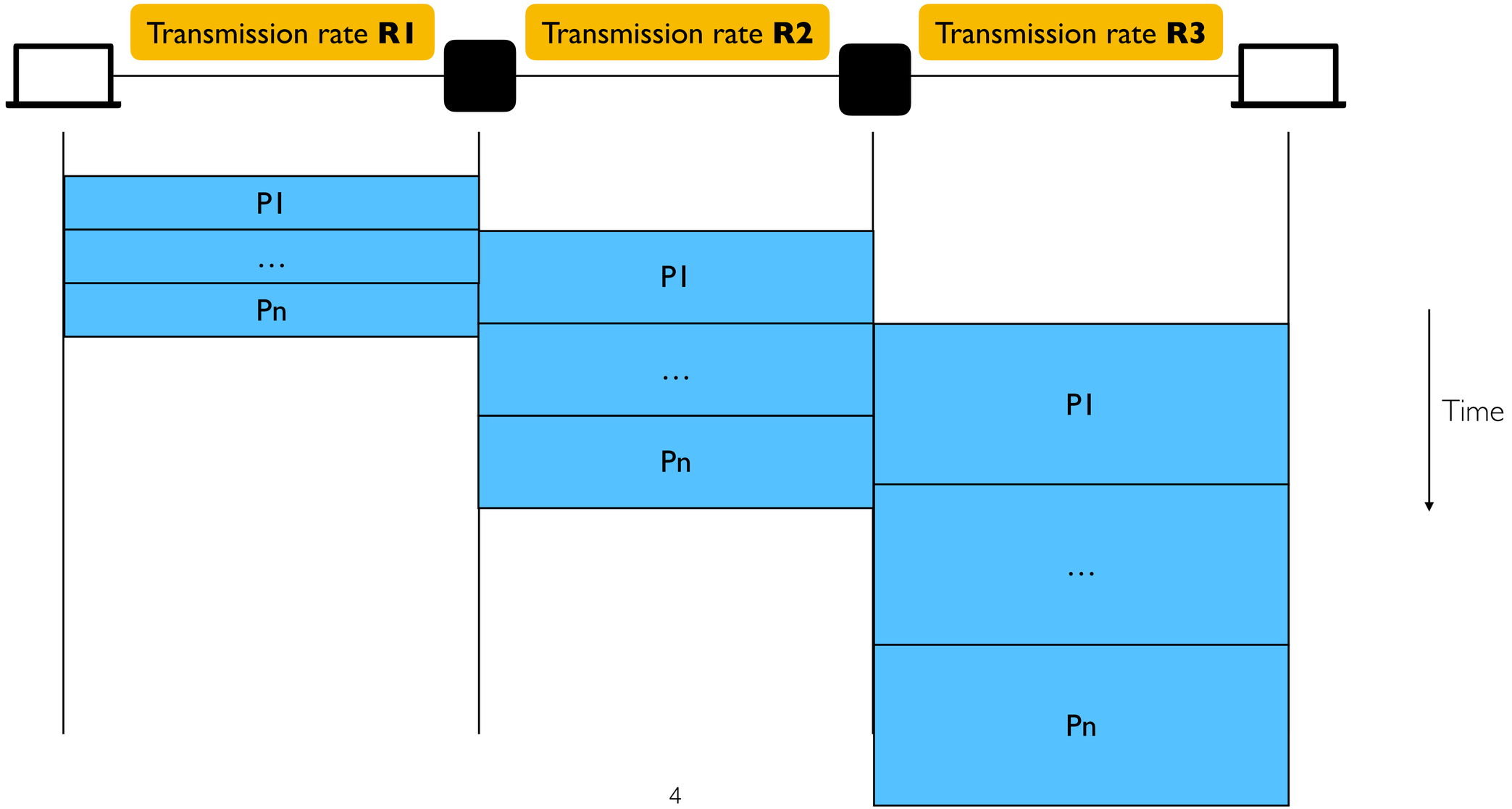
Ignoring propagation, queueing and processing delays, packet delay for a packet of P bits $= \frac{P}{R1} + \frac{P}{R2} + \frac{P}{R3}$

Shouldn't the throughput of a file of F bits sent in P bit packets be $\frac{F}{P} \cdot \frac{P}{\frac{P}{R1} + \frac{P}{R2} + \frac{P}{R3}} = \frac{F}{\frac{P}{R1} + \frac{P}{R2} + \frac{P}{R3}}$?

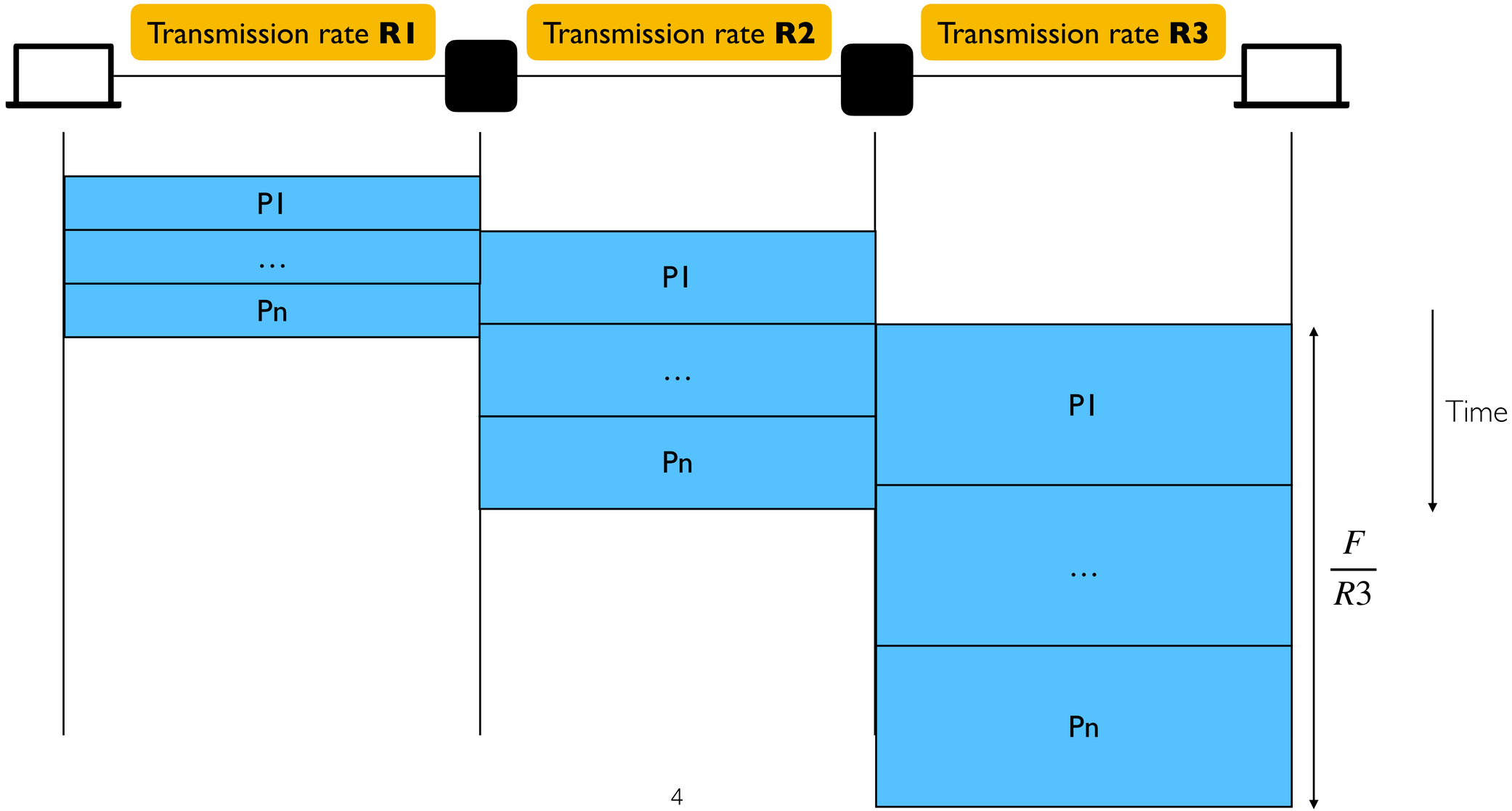
Throughput Calculation: Packet by packet



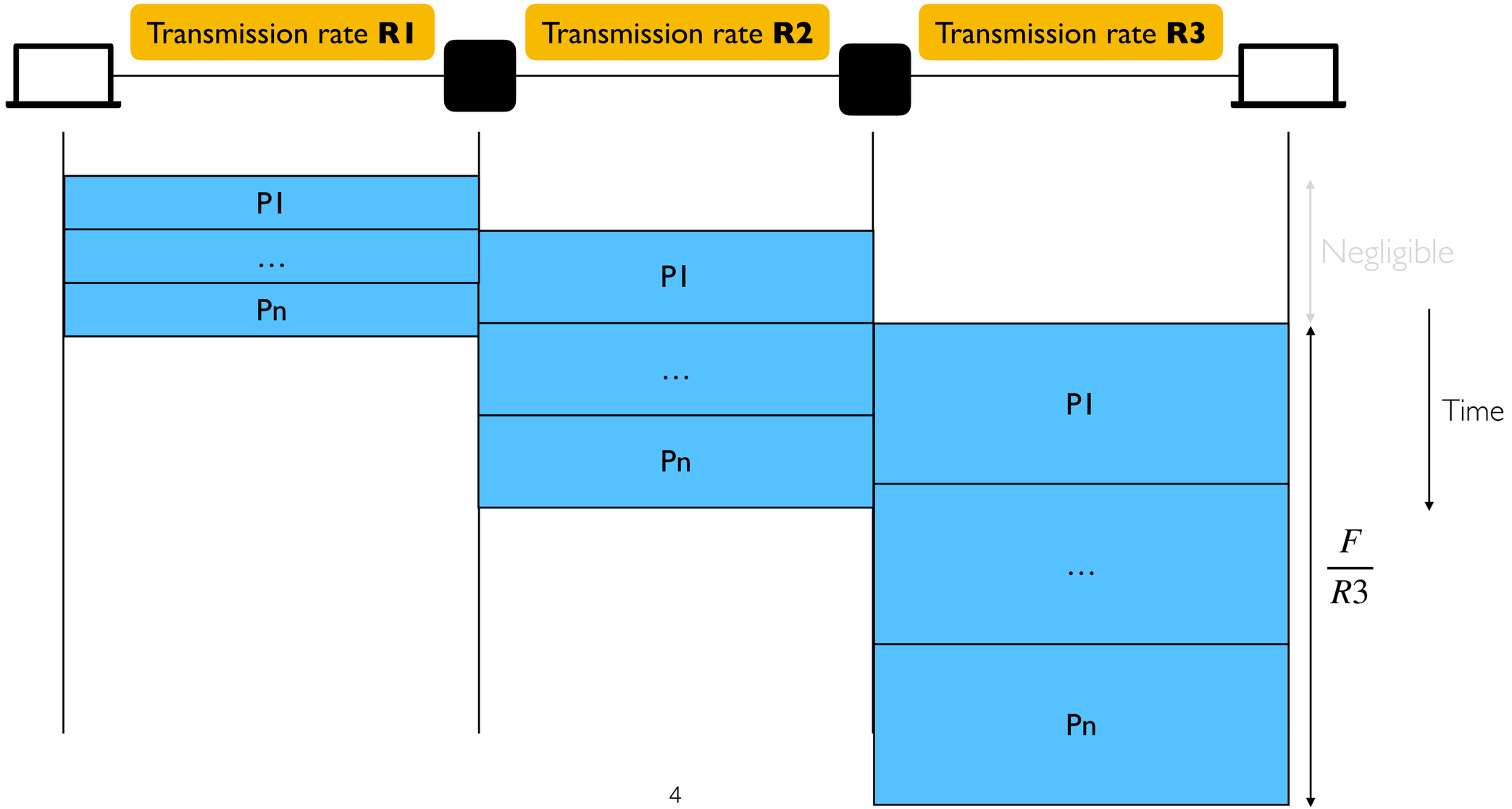
Throughput Calculation: In Reality



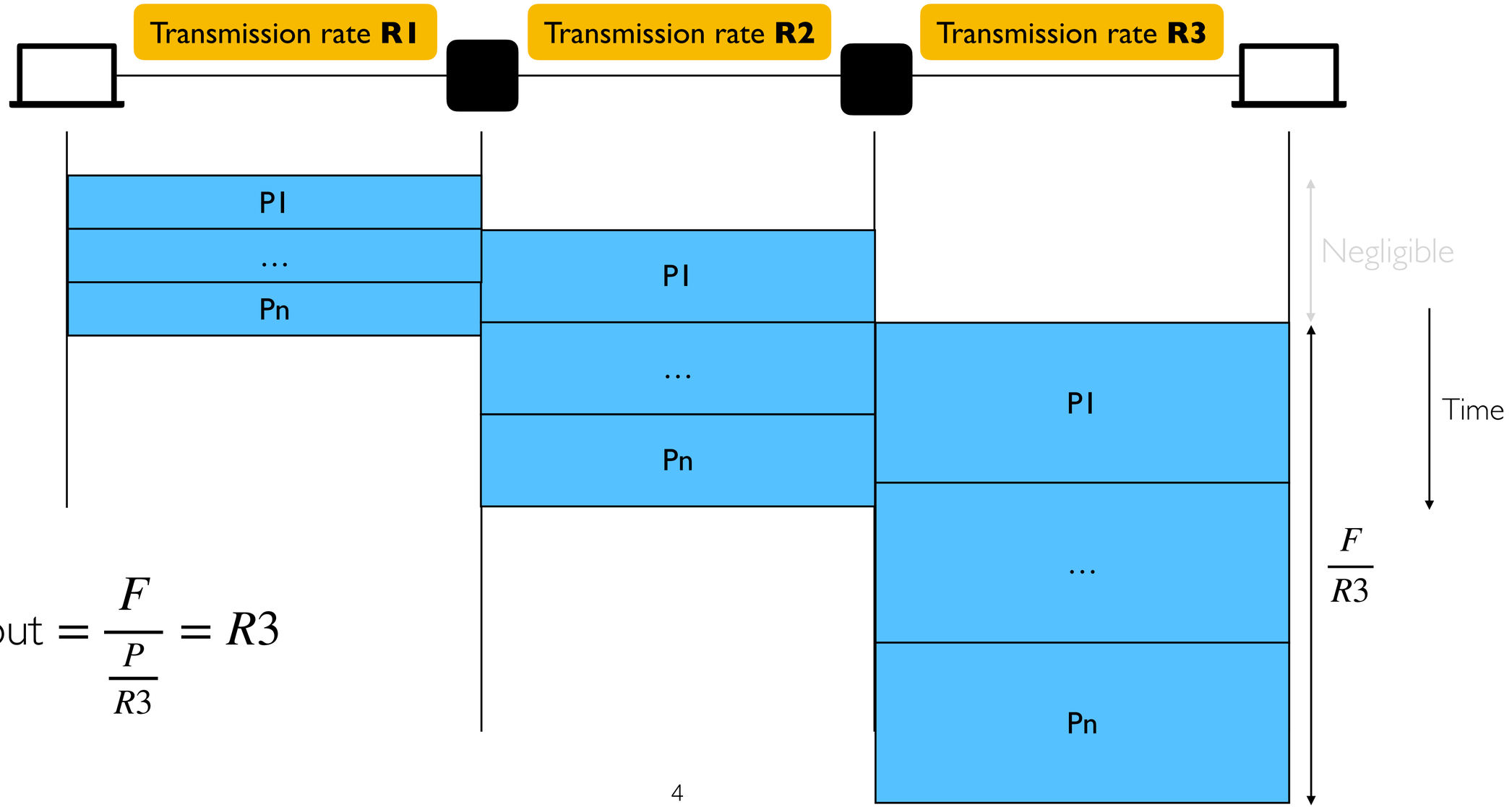
Throughput Calculation: In Reality



Throughput Calculation: In Reality



Throughput Calculation: In Reality



$$\text{Throughput} = \frac{F}{\frac{P}{R3}} = R3$$

Three steps

- **Decompose** the problem into tasks
- **Organize** these tasks
- **Assign** tasks to entities (who does what)

Three steps

- **Decomposition**
- **Organization**
- **Assignment**

Three steps

- **Decomposition**
- **Organization**
- **Assignment**

Layering principle

Three steps

- **Decomposition**
- **Organization**
- **Assignment**

The Path through FedEx

FE = FedEx Envelope

Truck

Truck

Sorting Office

Sorting Office

Sorting Office

Airport

Airport

Airport

The Path through FedEx

FE = FedEx Envelope

Truck

Sorting Office

Airport

Sorting Office

Airport

Truck

Sorting Office

Airport

The Path through FedEx

FE = FedEx Envelope

Truck

Truck

Sorting Office

Sorting Office

Sorting Office

Crate

Airport

Airport

Airport

The Path through FedEx

FE = FedEx Envelope

Truck

Truck

Sorting Office

Sorting Office

Sorting Office

Airport

Airport

Airport

The Path through FedEx

FE = FedEx Envelope

Truck

Truck

Sorting Office

Sorting Office

Sorting Office

Crate

Airport

Airport

Airport

The Path through FedEx

FE = FedEx Envelope

Truck

Truck

FE

Sorting Office

Sorting Office

Sorting Office

Airport

Airport

Airport

The Path through FedEx

FE = FedEx Envelope

Truck

Truck

Sorting Office

Sorting Office

Sorting Office

New Crate

Airport

Airport

Airport

The Path through FedEx

FE = FedEx Envelope

Truck

Truck

Sorting Office

Sorting Office

Sorting Office

Airport

Airport

Airport

The Path through FedEx

FE = FedEx Envelope

Truck

Truck

Sorting Office

Sorting Office

Sorting Office

Crate

Airport

Airport

Airport

The Path through FedEx

FE = FedEx Envelope

Truck

Sorting Office

Airport

Sorting Office

Airport

Truck

FE

Sorting Office

Airport

The Path through FedEx

FE = FedEx Envelope

Truck

Truck

Sorting Office

Sorting Office

Sorting Office

Airport

Airport

Airport

The Path through FedEx

FE = FedEx Envelope

Truck

Truck

Sorting Office

Sorting Office

Sorting Office

Airport

Airport

Airport

Deepest packaging (Envelope+FE+Crate) at Lowest level of transport

The Path through FedEx

FE = FedEx Envelope

Higher "Stack" at Ends

Truck

Truck

Sorting Office

Sorting Office

Sorting Office

Airport

Airport

Airport

Deepest packaging (Envelope+FE+Crate) at Lowest level of transport

The Path through FedEx

FE = FedEx Envelope

Higher "Stack" at Ends

Partial "Stack" during Transit

Truck

Truck

Sorting Office

Sorting Office

Sorting Office

Airport

Airport

Airport

Deepest packaging (Envelope+FE+Crate) at Lowest level of transport

Path through Internet

Path through Internet

What gets implemented where?

What layers get implemented at the end-systems?

What layers get implemented at the end-systems?

- Bits arrive on wire, must make it up to application

What layers get implemented at the end-systems?

- Bits arrive on wire, must make it up to application
- Therefore, all layers must exist at host!

What layers get implemented in the network?

What layers get implemented in the network?

- Bits arrive on wire → physical layer (L1)

What layers get implemented in the network?

- Bits arrive on wire → physical layer (L1)
- Packets must be delivered across link and local networks → datalink layer (L2)

What layers get implemented in the network?

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

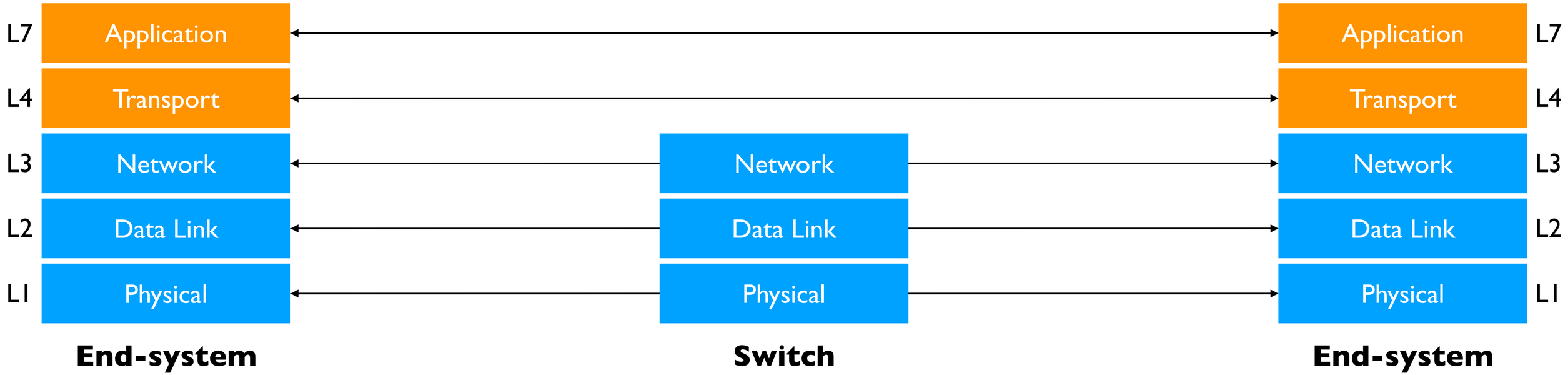
What layers get implemented in the network?

- Bits arrive on wire → physical layer (L1)
- Packets must be delivered across link and local networks → datalink layer (L2)
- Packets must be delivered b/w networks for global delivery → network layer (L3)
- The network does not support reliable delivery

What layers get implemented in the network?

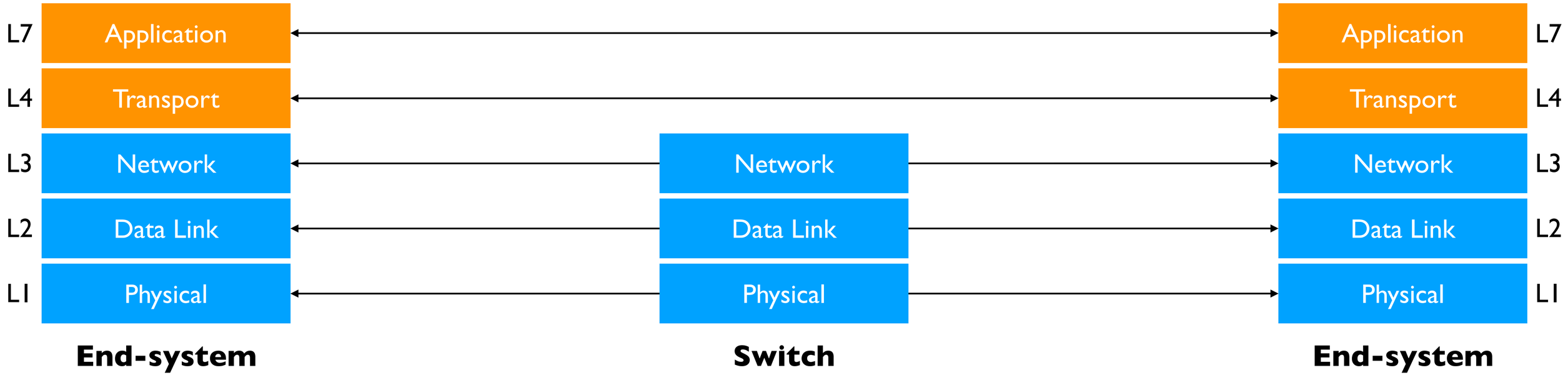
- Bits arrive on wire → physical layer (L1)
- Packets must be delivered across link and local networks → datalink layer (L2)
- Packets must be delivered b/w networks for global delivery → network layer (L3)
- The network does not support reliable delivery
  - *Transport layer (and above) <u>not</u> supported* ***(Why?)***

Path through Internet



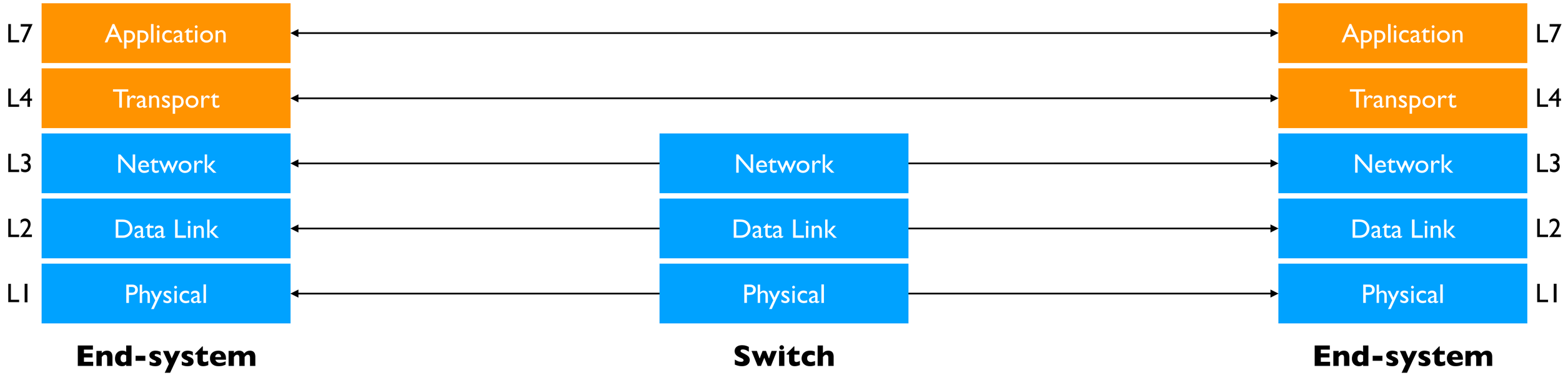
Path through Internet

- Lower three layers implemented everywhere



Path through Internet

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



A closer look: end-system

A closer look: end-system

- Application
  - *Web server, browser, mail, game*

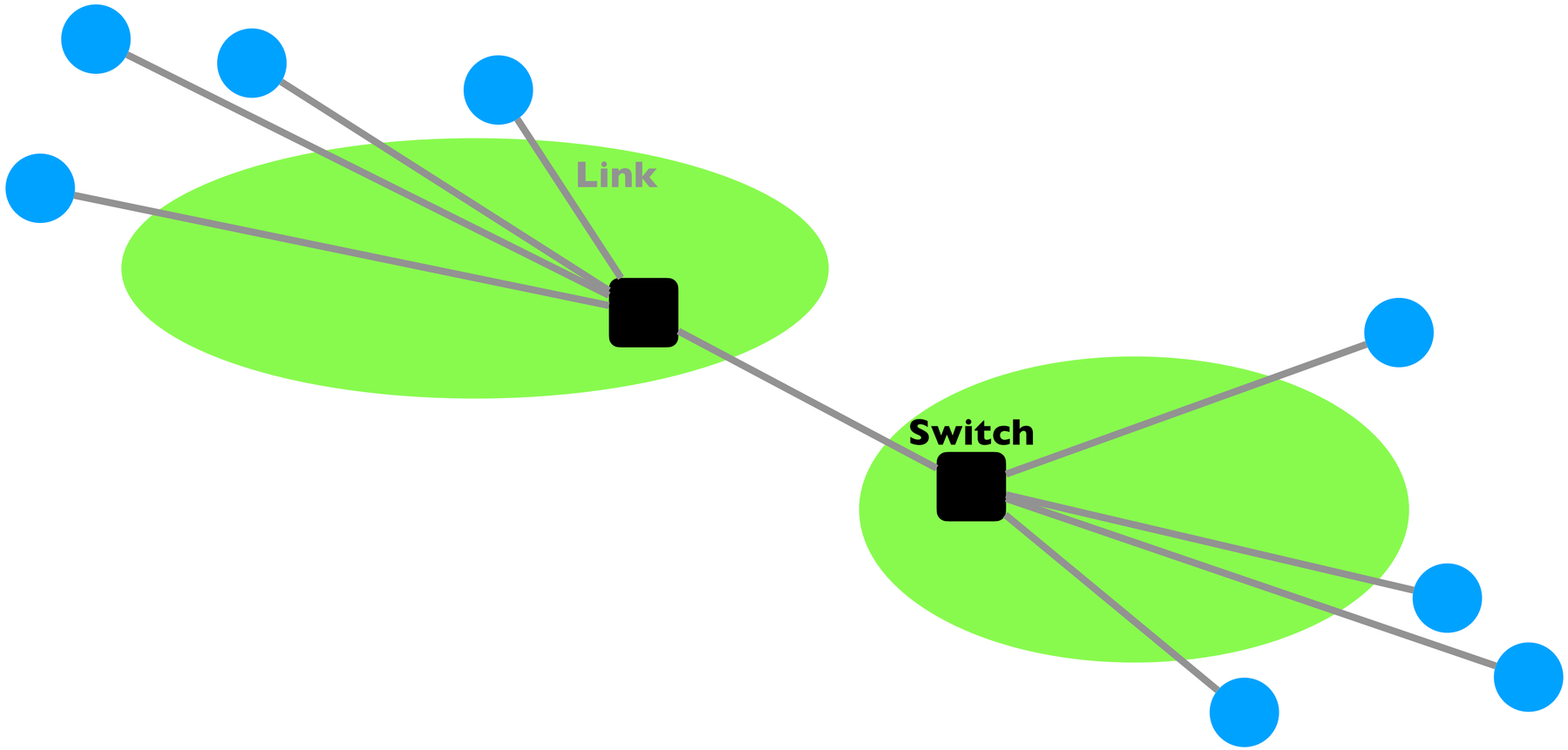
A closer look: end-system

- Application
  - *Web server, browser, mail, game*
- Transport and network layer
  - *Typically part of the operating system*

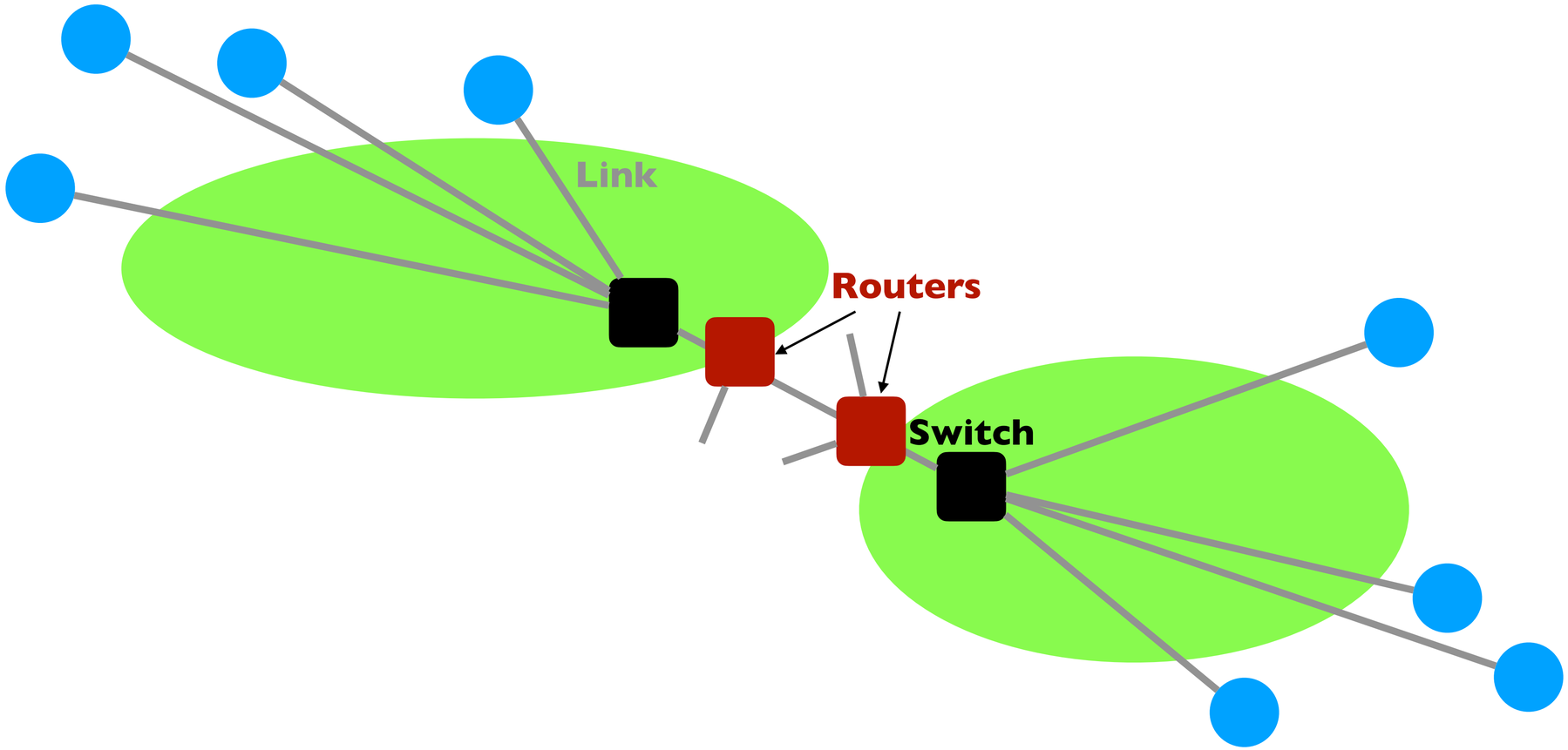
A closer look: end-system

- Application
  - *Web server, browser, mail, game*
- Transport and network layer
  - *Typically part of the operating system*
- Datalink and physical layer
  - *Hardware/firmware/drivers*

A closer look: Network



A closer look: Network



Switches vs. Routers

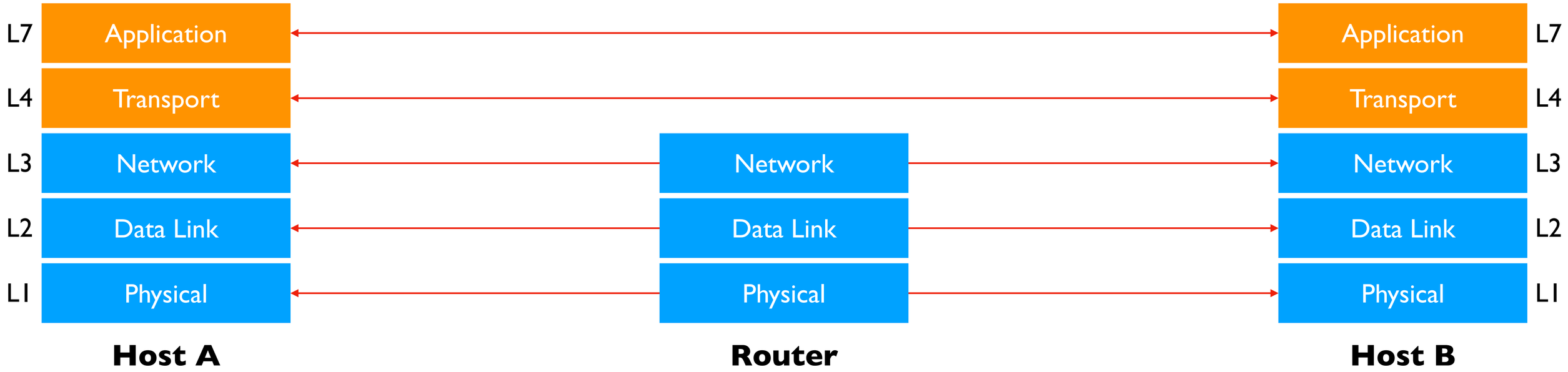
Switches vs. Routers

- Switches do what routers do, but don't participate in global delivery, just local delivery
  - *Switches only need to support L1, L2*
  - *Routers support L1-L3*

Switches vs. Routers

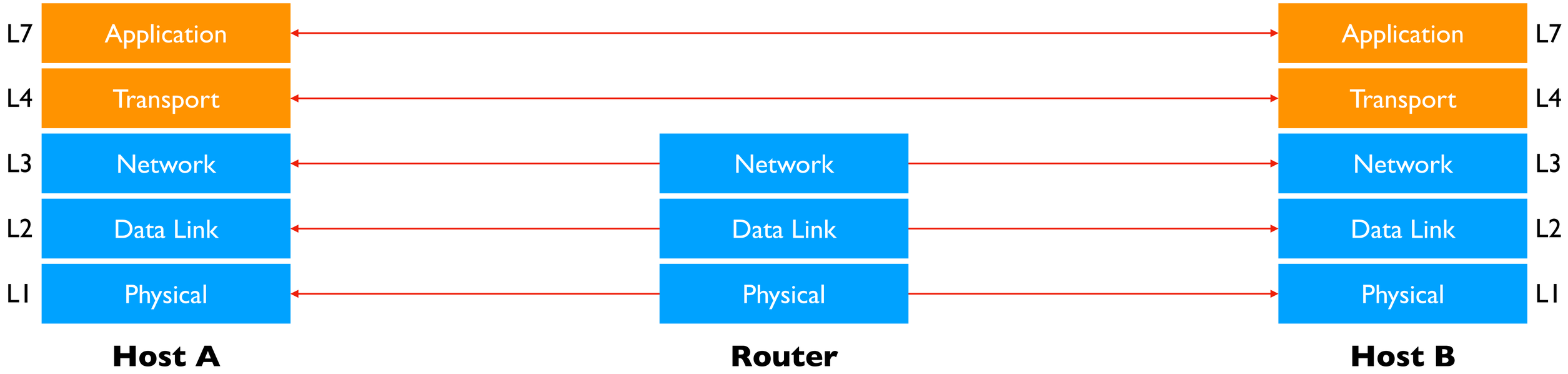
- Switches do what routers do, but don't participate in global delivery, just local delivery
  - *Switches only need to support L1, L2*
  - *Routers support L1-L3*
- Won't focus on the router/switch distinction
  - *When I say switch, I almost always mean router*
  - *Almost all boxes support network layer these days*

Logical Communication

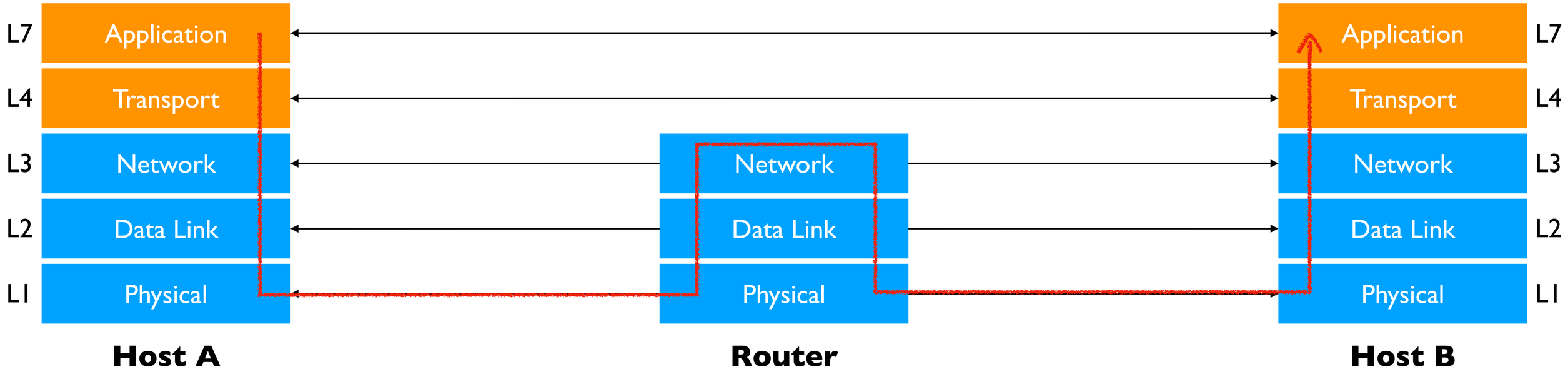


Logical Communication

- Layers interact with peer's corresponding layer

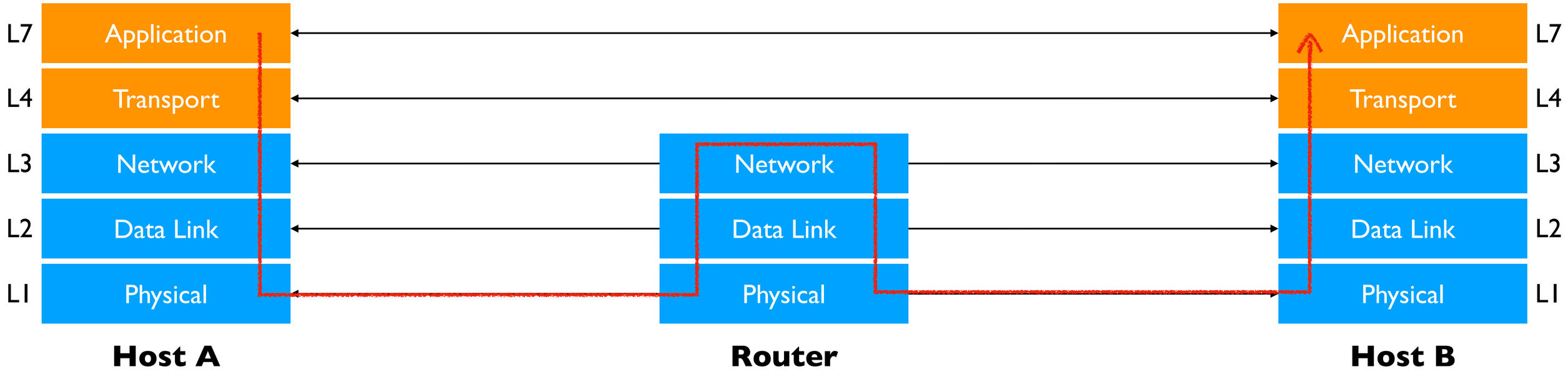


Physical Communication



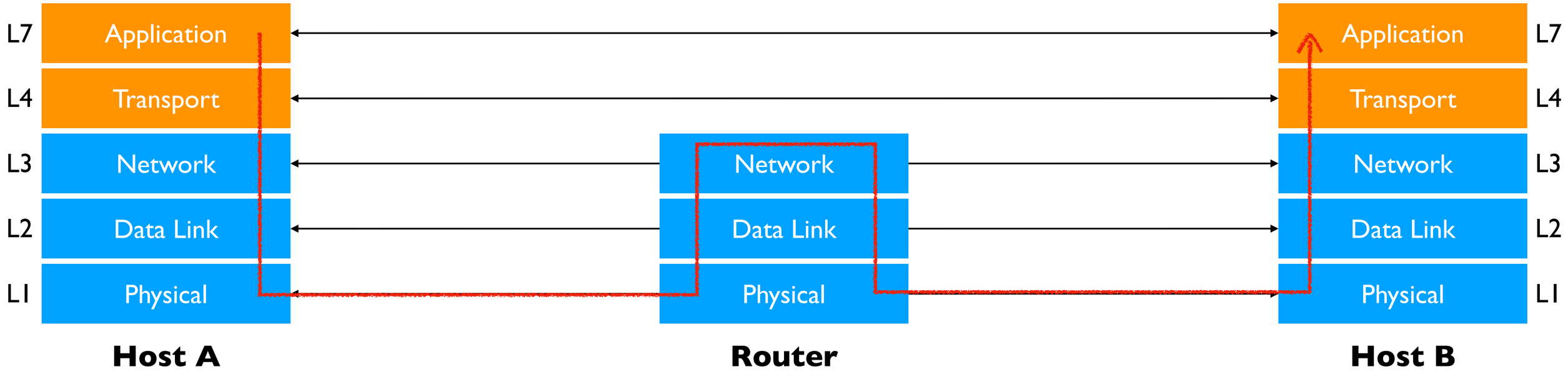
Physical Communication

- Communication goes down to the physical network

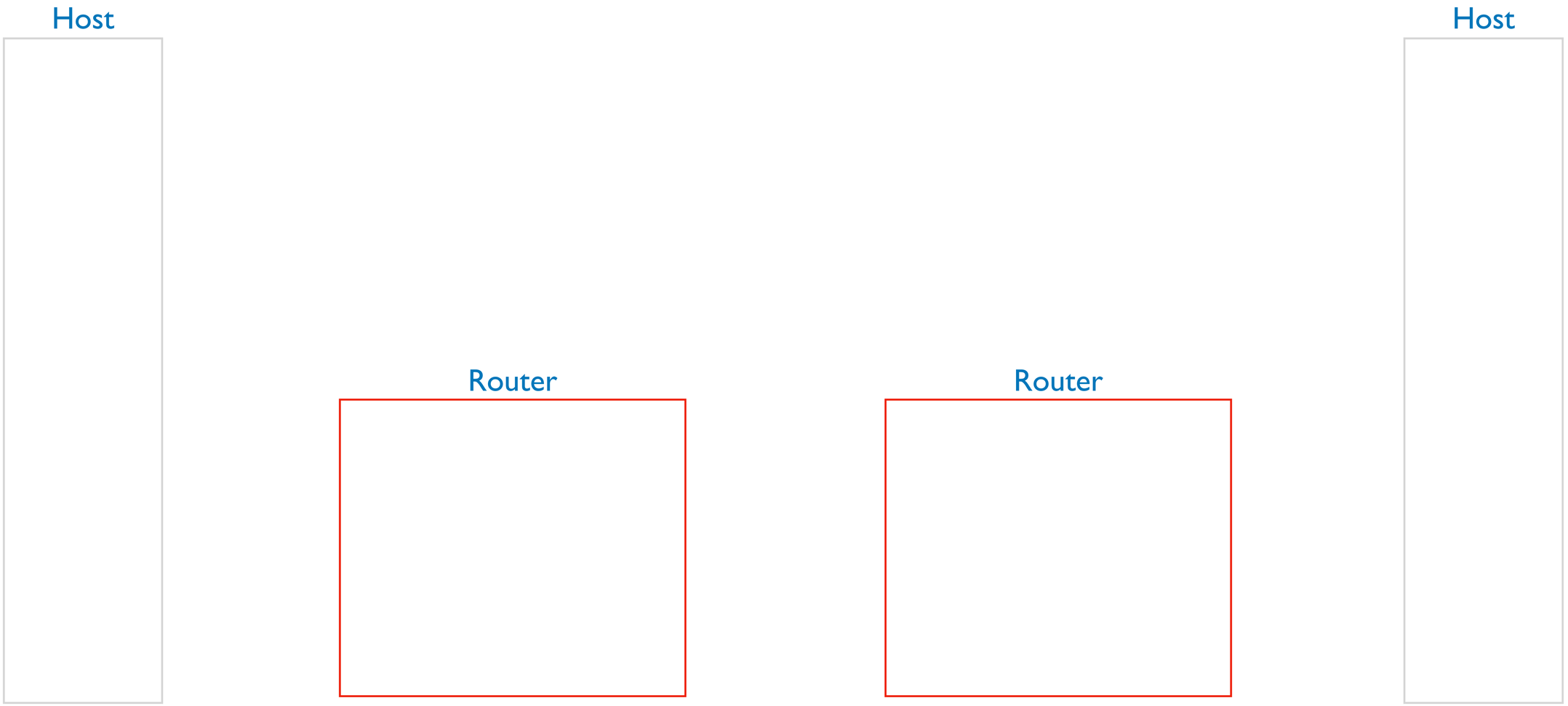


Physical Communication

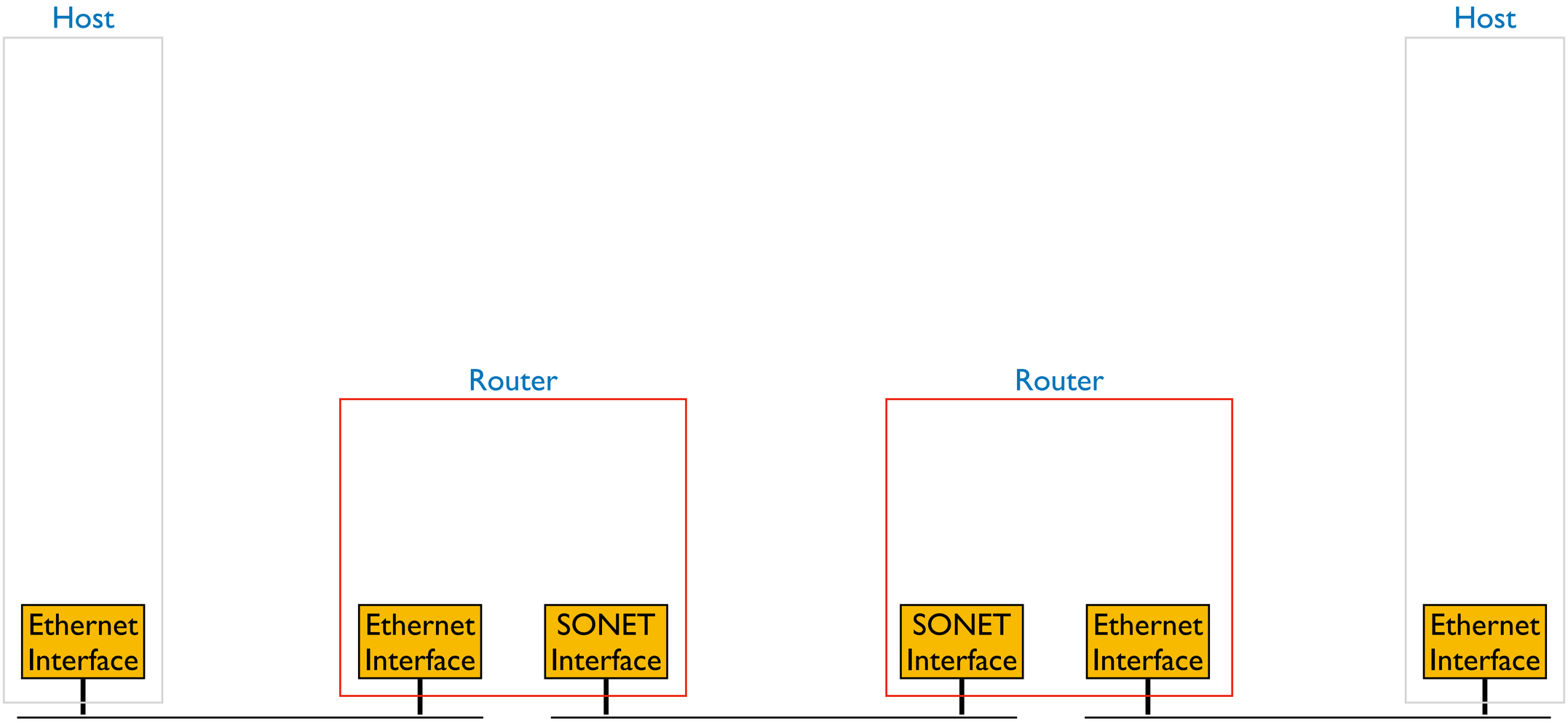
- Communication goes down to the physical network
- Then up to relevant layer



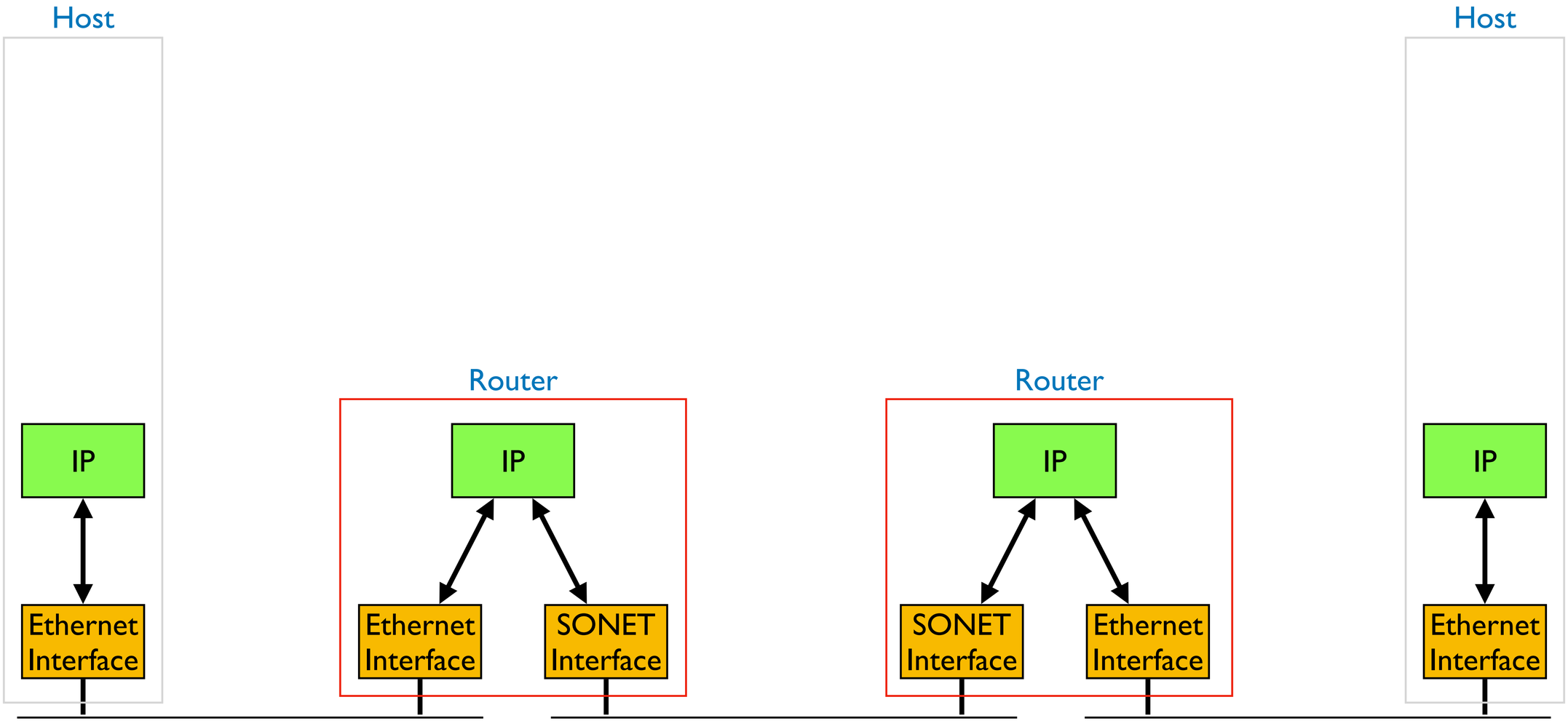
A Protocol-centric Diagram



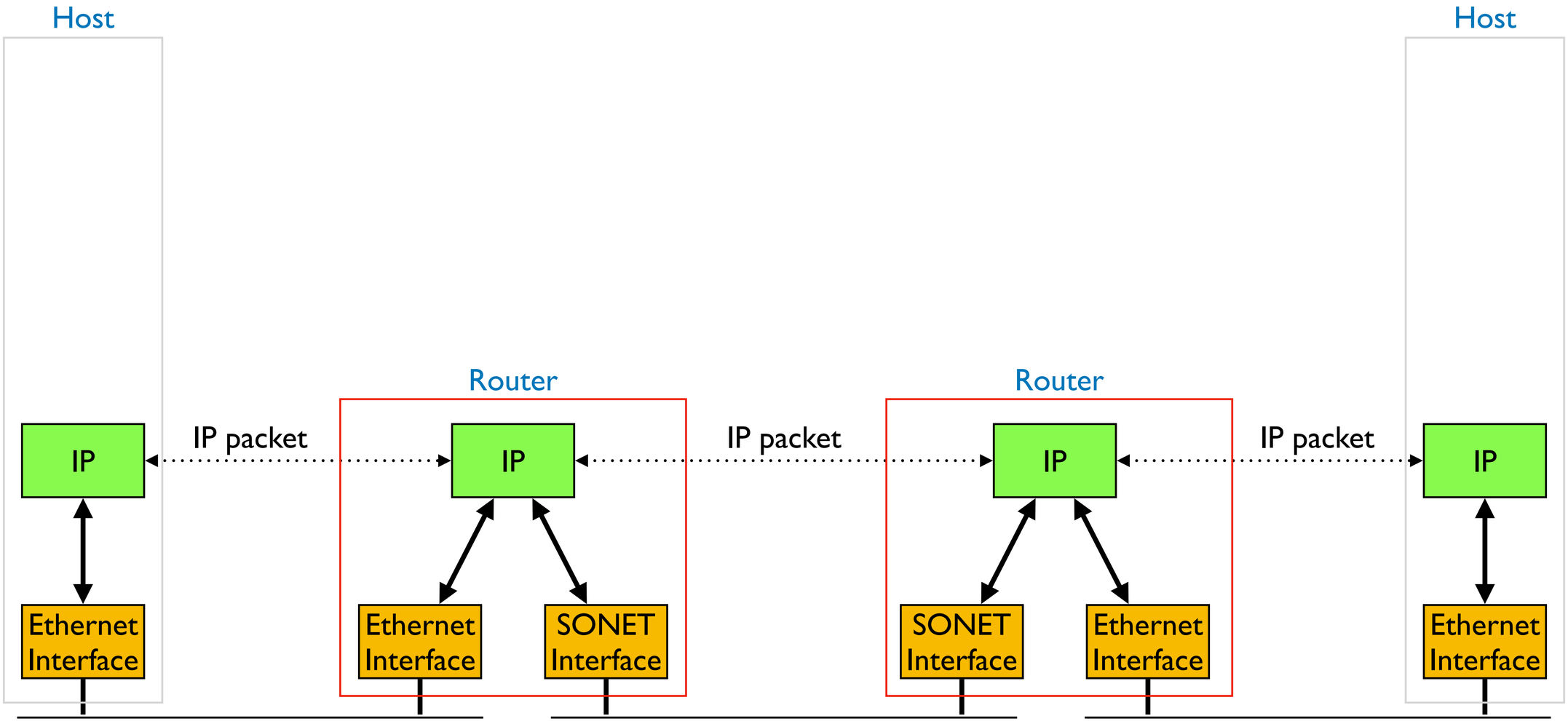
A Protocol-centric Diagram



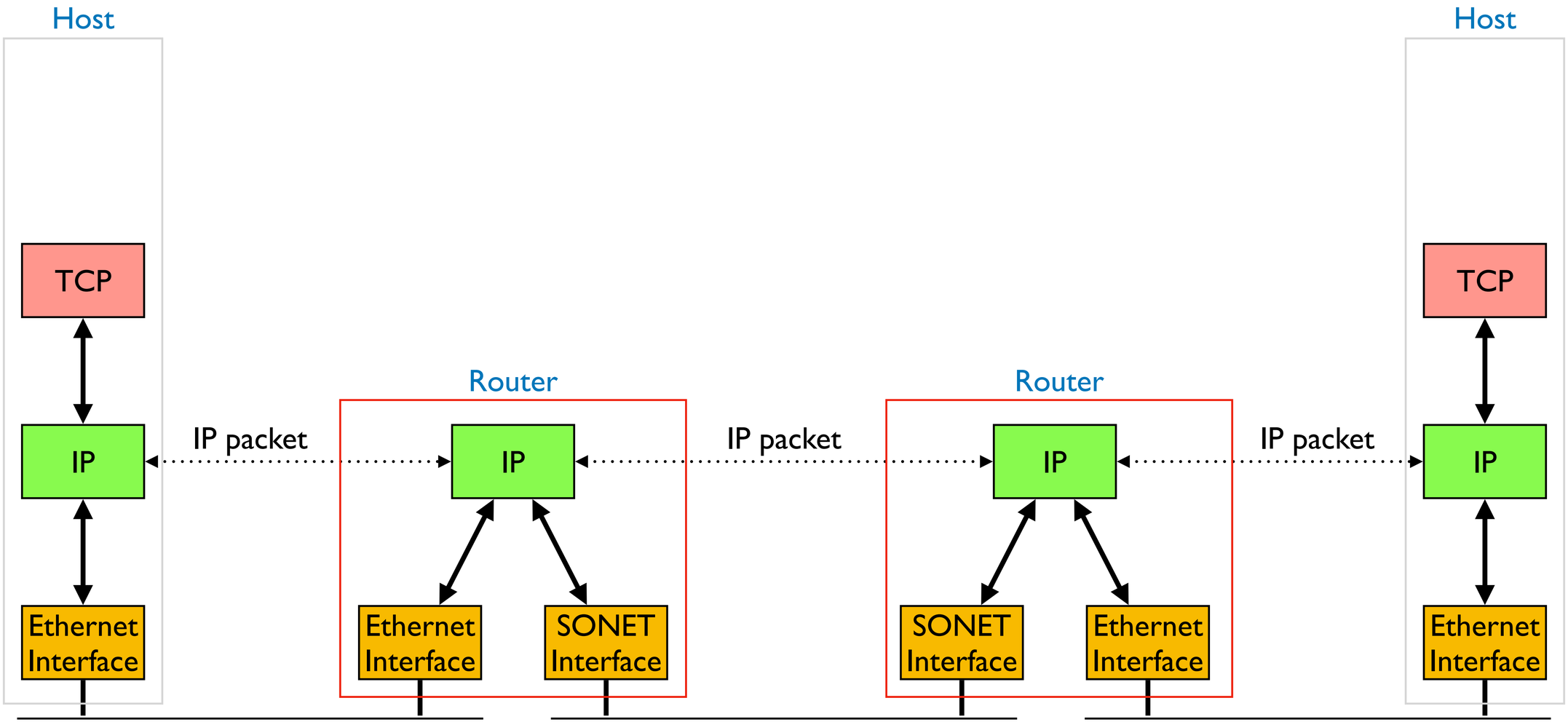
A Protocol-centric Diagram



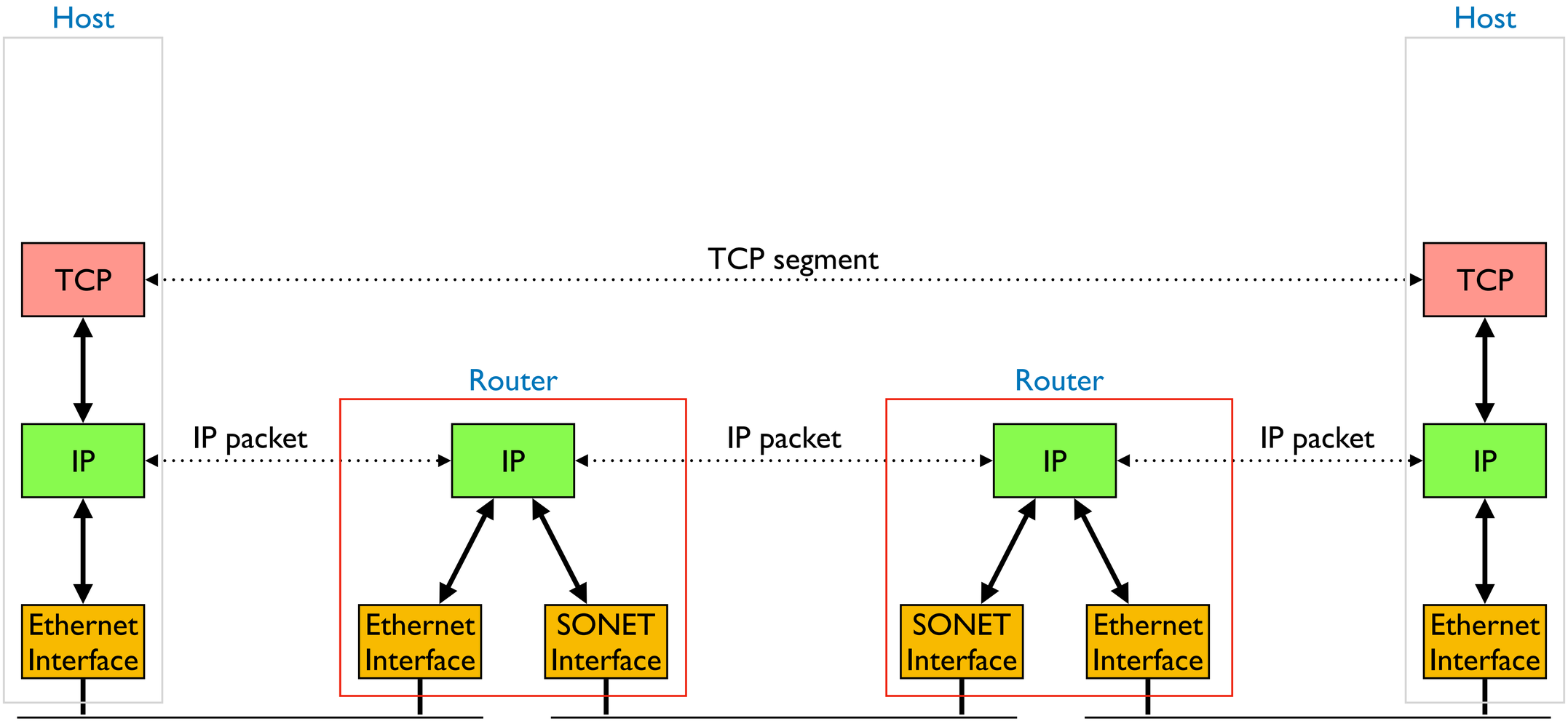
A Protocol-centric Diagram



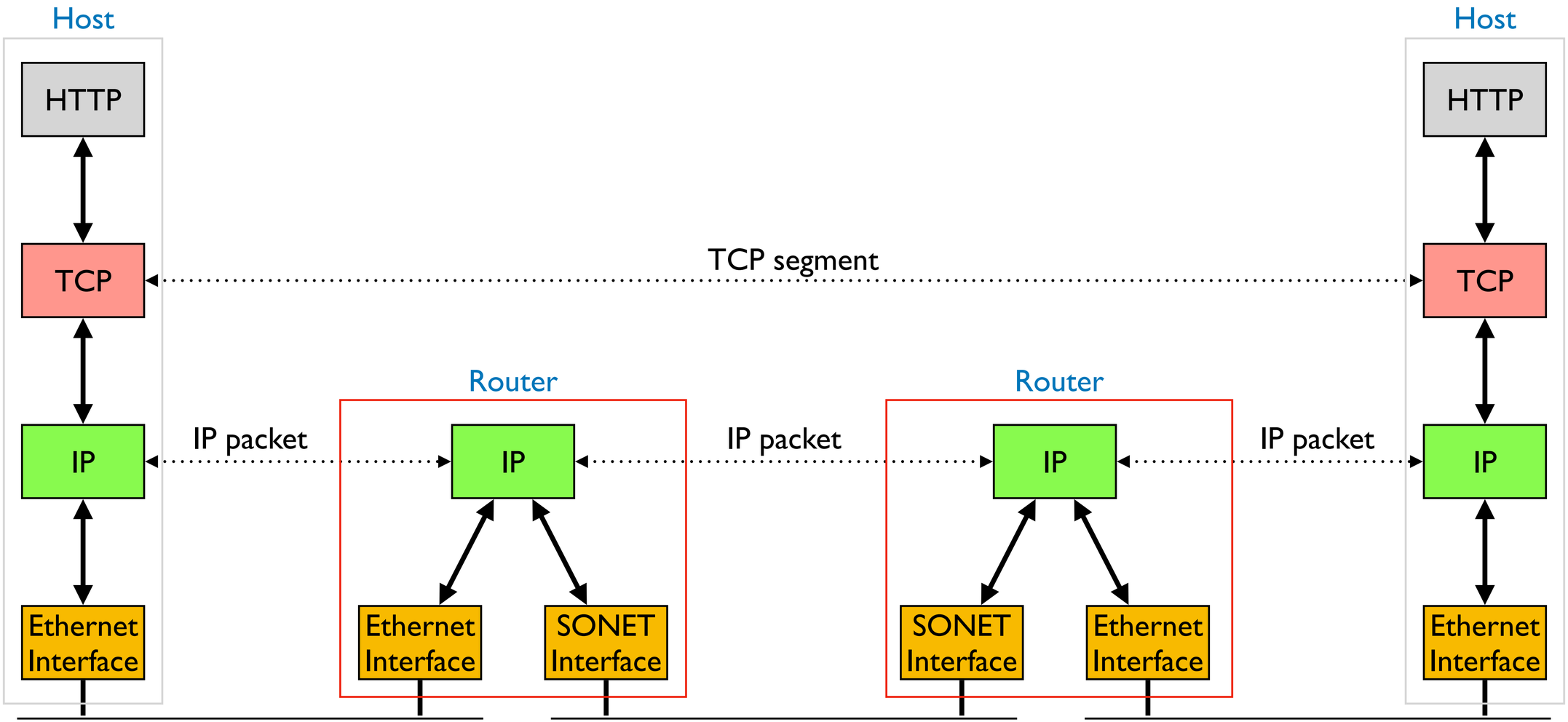
A Protocol-centric Diagram



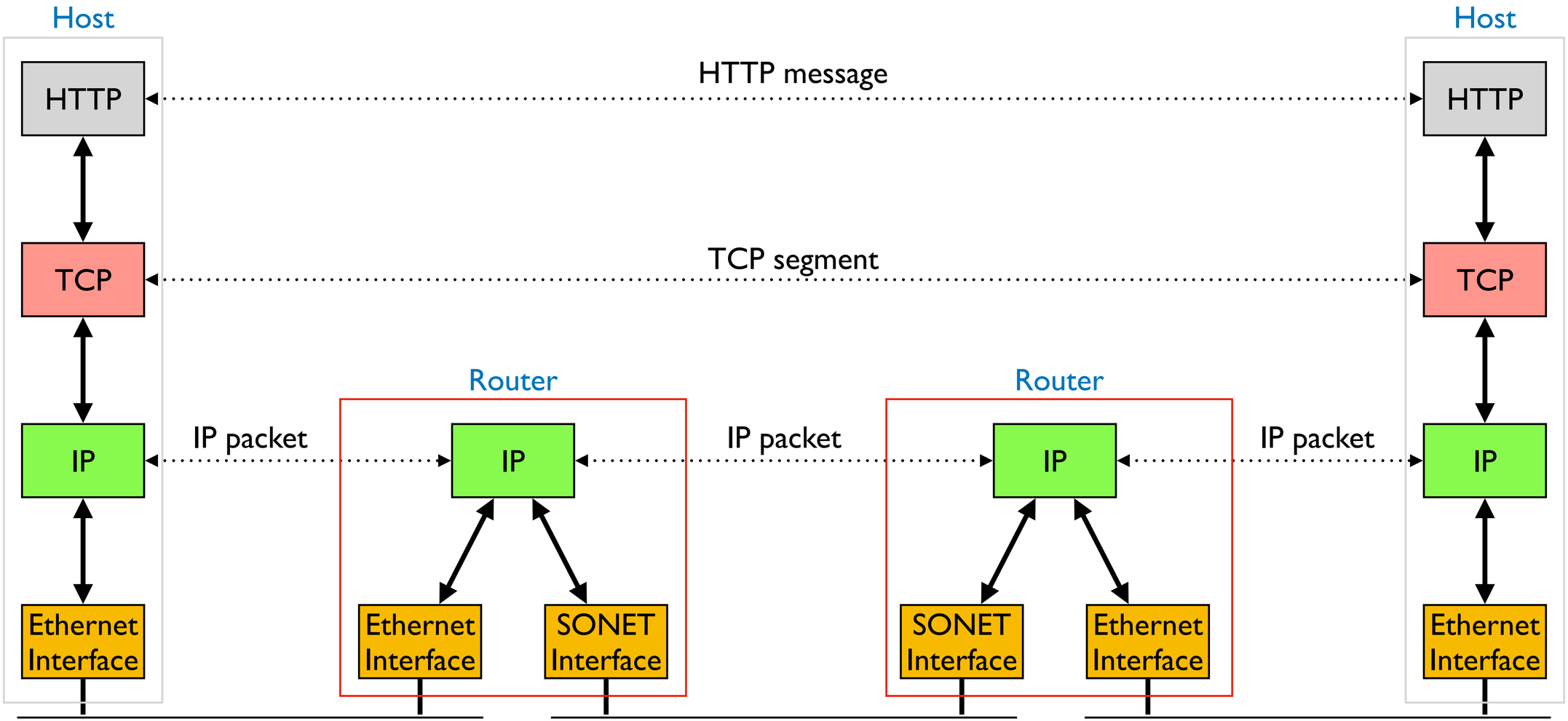
A Protocol-centric Diagram



A Protocol-centric Diagram



A Protocol-centric Diagram



Questions?

Three steps

- **Decomposition**
- **Organization**
- **Assignment**

Three steps

- **Decomposition**
- **Organization**
- **Assignment**

End-to-end principle

Placing Network Functionality

Placing Network Functionality

- Hugely influential paper: **“End-to-end Arguments in System Design” by Saltzer, Reed, and Clark (’84)**
  - *Articulated the “End-to-end Principle” (E2E)*

Placing Network Functionality

- Hugely influential paper: **“End-to-end Arguments in System Design” by Saltzer, Reed, and Clark (’84)**
  - *Articulated the “End-to-end Principle” (E2E)*
- Endless debate over what it means

Placing Network Functionality

- Hugely influential paper: **“End-to-end Arguments in System Design” by Saltzer, Reed, and Clark (’84)**
  - *Articulated the “End-to-end Principle” (E2E)*
- Endless debate over what it means
- Many have used their own interpretation to support their position

Placing Network Functionality

- Hugely influential paper: **“End-to-end Arguments in System Design” by Saltzer, Reed, and Clark (’84)**
  - *Articulated the “End-to-end Principle” (E2E)*
- Endless debate over what it means
- Many have used their own interpretation to support their position
- But core argument holds value, esp. in understanding its impact on Internet design

The Statement

The Statement

- **If** a function **completely and correctly be implemented only with the knowledge and help of** the applications standing at the **end points** of the communication system,

The Statement

- **If** a function **completely and correctly be implemented only with the knowledge and help of** the applications standing at the **end points** of the communication system,

- **Then,** providing that function as a **feature of the communication system** itself is not possible.

The Statement

- **If** a function **completely and correctly be implemented only with the knowledge and help of** the applications standing at the **end points** of the communication system,

- **Then,** 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**

Deconstructing the Statement

Deconstructing the Statement

- Some application requirements can only be correctly implemented **at the end-system**

Deconstructing the Statement

- Some application requirements can only be correctly implemented **at the end-system**
  - *Reliability, security, etc.*

Deconstructing the Statement

- Some application requirements can only be correctly implemented **at the end-system**
  - *Reliability, security, etc.*
- Implementing these completely in the network may not only be hard, but impossible

Deconstructing the Statement

- Some application requirements can only be correctly implemented **at the end-system**
  - *Reliability, security, etc.*
- Implementing these completely in the network may not only be hard, but impossible
- End-systems:

Deconstructing the Statement

- Some application requirements can only be correctly implemented **at the end-system**
  - *Reliability, security, etc.*
- Implementing these completely in the network may not only be hard, but impossible
- End-systems:
  - ***Can*** *satisfy the requirement without network's help*

Deconstructing the Statement

- Some application requirements can only be correctly implemented **at the end-system**
  - *Reliability, security, etc.*
- Implementing these completely in the network may not only be hard, but impossible
- End-systems:
  - ***Can*** *satisfy the requirement without network's help*
  - ***Will/must*** *do so, since they cannot rely on the network*

Example: Reliable File Transfer

Example: Reliable File Transfer

- **Solution 1:** Make each step reliable, and string them together to make the end-to-end process reliable

Example: Reliable File Transfer

- **Solution 1:** Make each step reliable, and string them together to make the end-to-end process reliable

Example: Reliable File Transfer

- **Solution 1:** Make each step reliable, and string them together to make the end-to-end process reliable

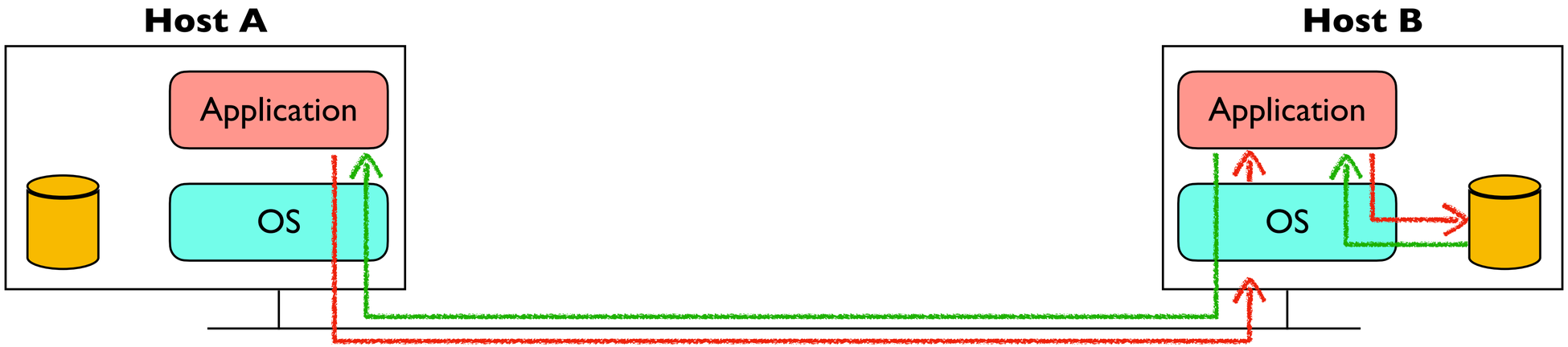
Example: Reliable File Transfer

- **Solution 1:** Make each step reliable, and string them together to make the end-to-end process reliable

Example: Reliable File Transfer

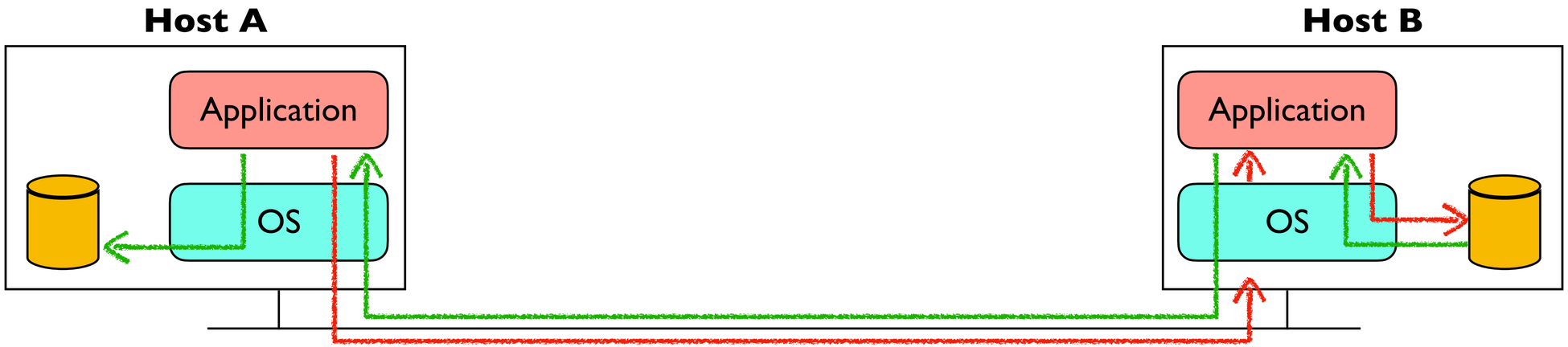
- **Solution 1:** Make each step reliable, and string them together to make the end-to-end process reliable

Example: Reliable File Transfer



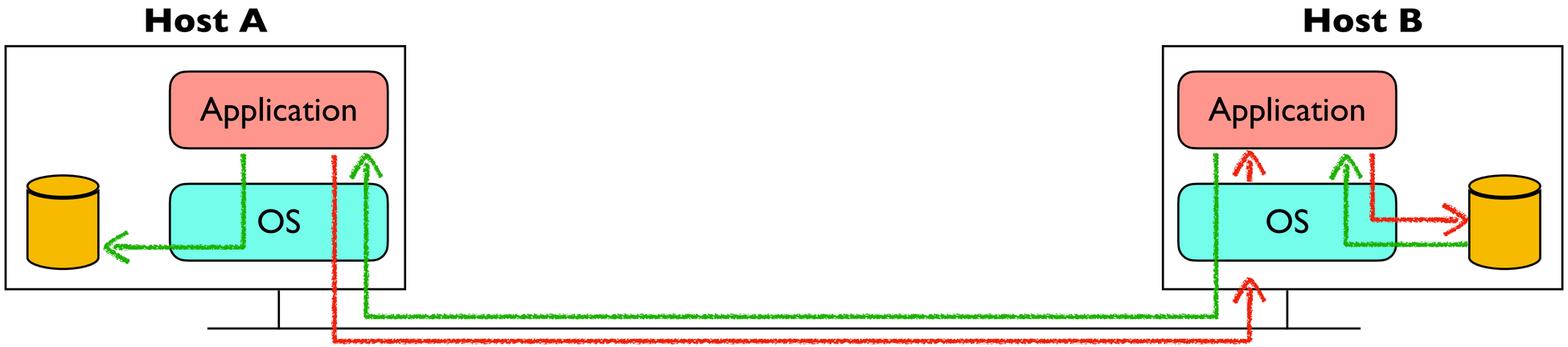
- **Solution 1:** Make each step reliable, and string them together to make the end-to-end process reliable

Example: Reliable File Transfer



- **Solution 1:** Make each step reliable, and string them together to make the end-to-end process reliable

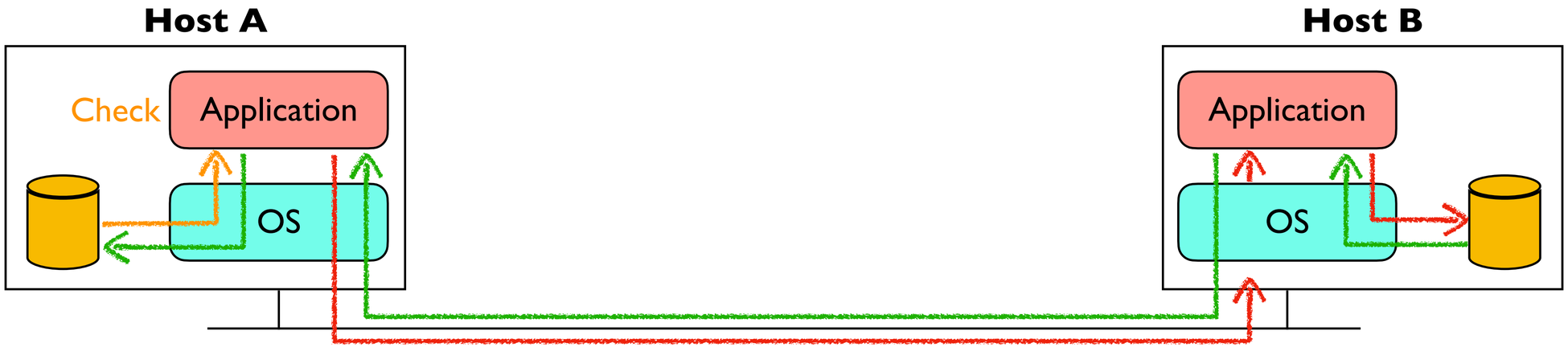
Example: Reliable File Transfer



- **Solution 1:** Make each step reliable, and string them together to make the end-to-end process reliable

- **Solution 2:** End-to-end check and retry

Example: Reliable File Transfer



- **Solution 1:** Make each step reliable, and string them together to make the end-to-end process reliable

- **Solution 2:** End-to-end check and retry

Discussion

Discussion

- **Solution I** is incomplete
  - *Receiver has to do the check anyway!*

Discussion

- **Solution 1** is incomplete
  - *Receiver has to do the check anyway!*
- **Solution 2** is complete
  - *Full functionality can be entirely implemented at application layer with no need for reliability from lower layers*

Discussion

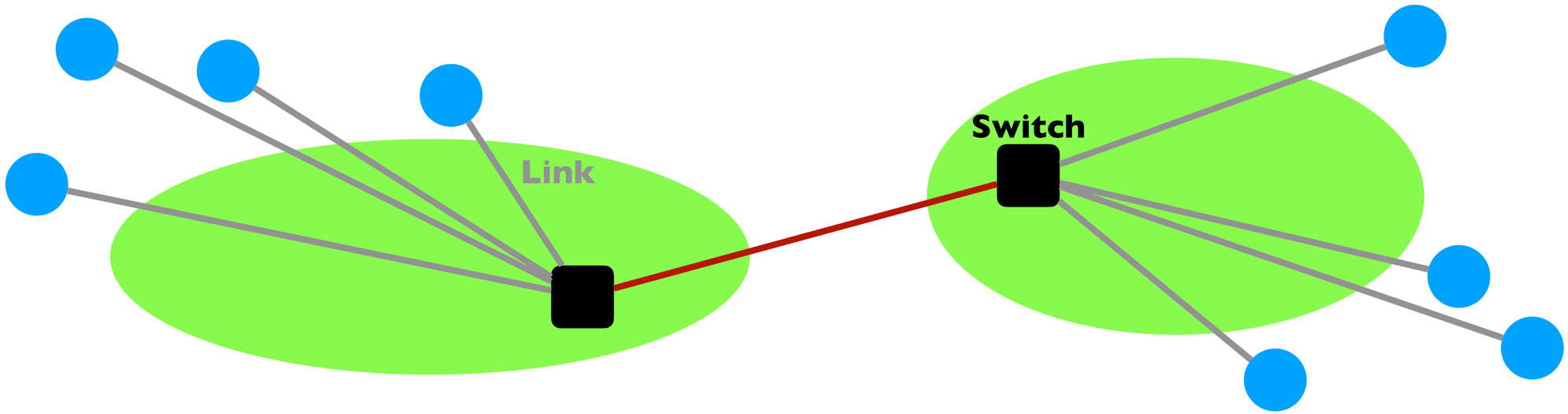
- **Solution 1** is incomplete
  - *Receiver has to do the check anyway!*
- **Solution 2** is complete
  - *Full functionality can be entirely implemented at application layer with no need for reliability from lower layers*
- Is there any **need** to implement reliability at lower layers?

Except…

Sometimes an **incomplete version of the function** provided by the communication system may be useful as a **performance enhancement**

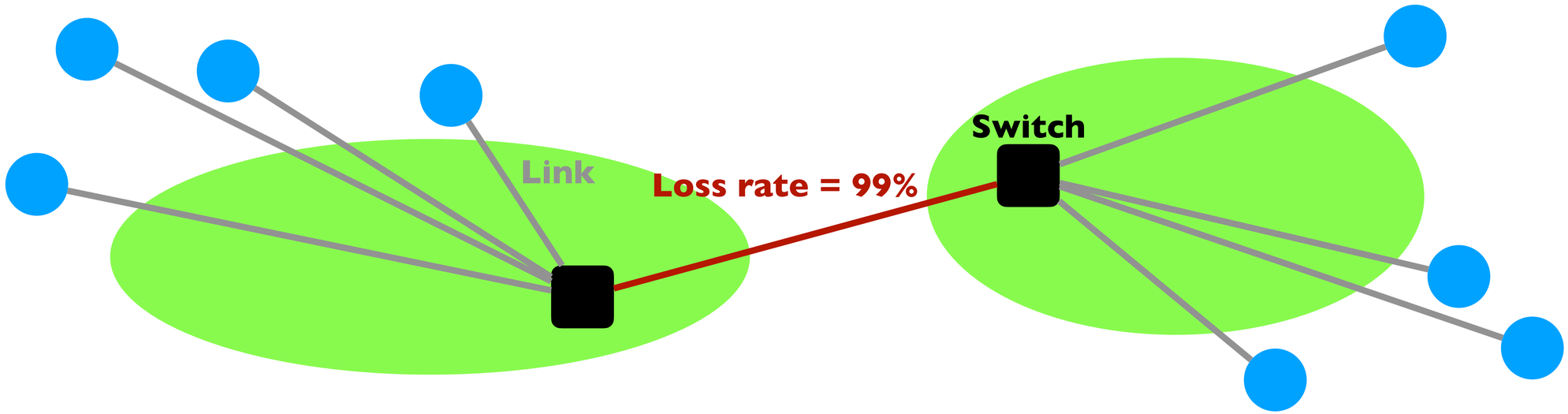
Except…

Sometimes an **incomplete version of the function** provided by the communication system may be useful as a **performance enhancement**



Except…

Sometimes an **incomplete version of the function** provided by the communication system may be useful as a **performance enhancement**



Summary of the E2E principle

Summary of the E2E principle

- Implementing functionality (e.g., reliability) in the network
  - *Doesn't reduce host implementation complexity*
  - *Does increase network complexity*
  - *Probably increases delay and overhead on all applications even if they don't need the functionality (e.g., reliable data transfer vs. VoIP)*

Summary of the E2E principle

- Implementing functionality (e.g., reliability) in the network
  - *Doesn't reduce host implementation complexity*
  - *Does increase network complexity*
  - *Probably increases delay and overhead on all applications even if they don't need the functionality (e.g., reliable data transfer vs. VoIP)*
- However, implementing in the network can improve performance in some cases
  - *E.g., consider a very lossy link*

Some consequences

Some consequences

- In layered design, the E2E principle provides guidance on which layers are implemented where

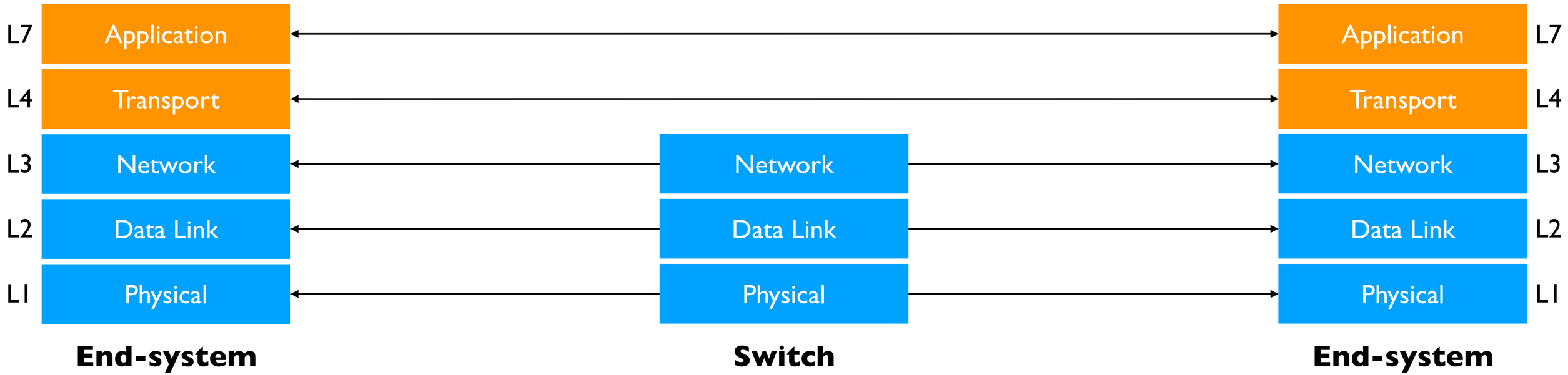
Some consequences

- In layered design, the E2E principle provides guidance on which layers are implemented where
- "Dumb" network and "smart" end-systems
  - *Often credited as a key to the Internet's success!*

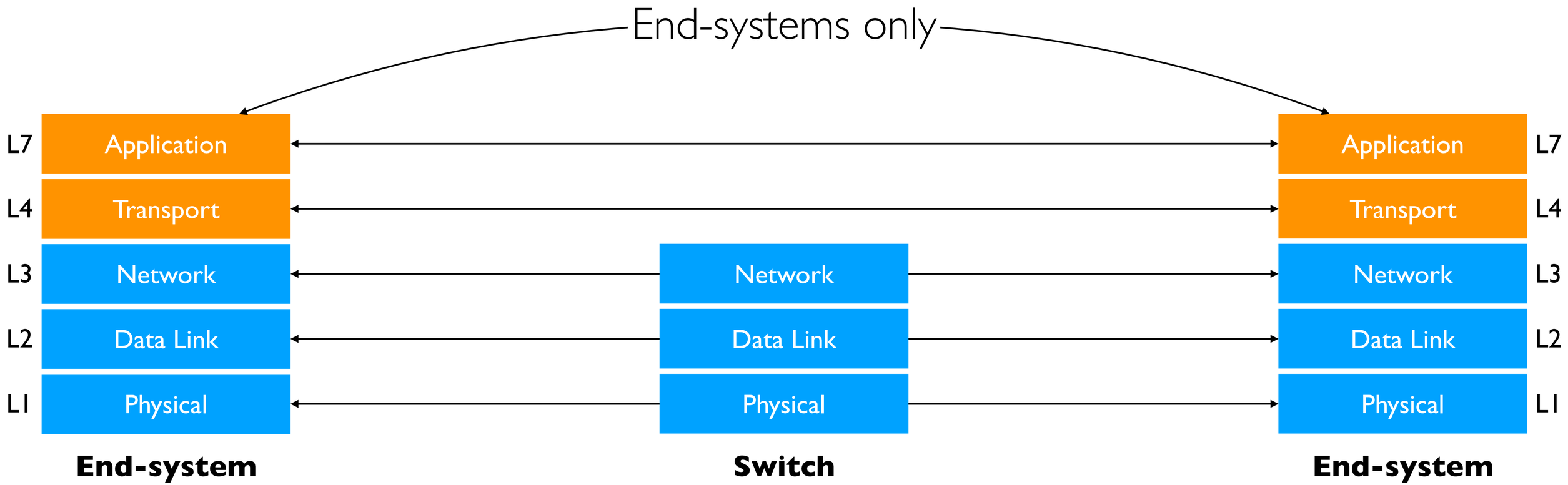
Some consequences

- In layered design, the E2E principle provides guidance on which layers are implemented where
- "Dumb" network and "smart" end-systems
  - *Often credited as a key to the Internet's success!*
- "Fate-sharing"
  - *Store state at the system entities that rely on the state*
  - *Often translated to keep end-host state out of routers*

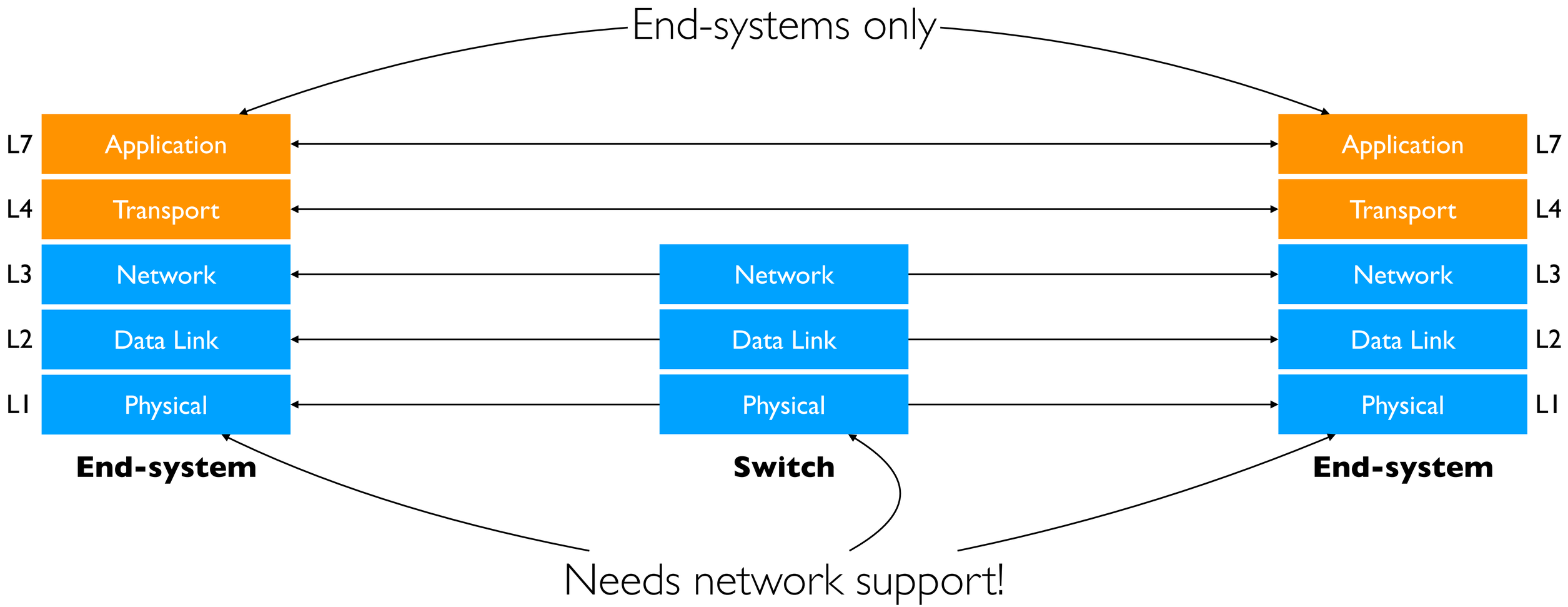
E2E meets Layering



E2E meets Layering



E2E meets Layering



Cracks in the E2E argument?

Cracks in the E2E argument?

- Ignores incentives of different stakeholders

Cracks in the E2E argument?

- Ignores incentives of different stakeholders
  - *e.g., ISP looking for new revenue-generating services*

Cracks in the E2E argument?

- Ignores incentives of different stakeholders
  - *e.g., ISP looking for new revenue-generating services*
  - *e.g., users looking for a performance boost*

Cracks in the E2E argument?

- Ignores incentives of different stakeholders
  - *e.g., ISP looking for new revenue-generating services*
  - *e.g., users looking for a performance boost*
- Sometimes we don't trust end-systems to do the job

Cracks in the E2E argument?

- Ignores incentives of different stakeholders
  - *e.g., ISP looking for new revenue-generating services*
  - *e.g., users looking for a performance boost*
- Sometimes we don't trust end-systems to do the job
  - *e.g., firewalls, intrusion detection systems*

Questions?

Asking the right architectural questions…

Asking the right architectural questions…

- How to break system into modules?

Asking the right architectural questions…

- How to break system into modules?
  - *Classic decomposition into tasks*

Asking the right architectural questions…

- How to break system into modules?
  - *Classic decomposition into tasks*
- Where are modules implemented?

Asking the right architectural questions…

- How to break system into modules?
  - *Classic decomposition into tasks*
- Where are modules implemented?
  - *Hosts? Routers? Both?*

Asking the right architectural questions…

- How to break system into modules?
  - *Classic decomposition into tasks*
- Where are modules implemented?
  - *Hosts? Routers? Both?*
- Where is state stored?

Asking the right architectural questions…

- How to break system into modules?
  - *Classic decomposition into tasks*
- Where are modules implemented?
  - *Hosts? Routers? Both?*
- Where is state stored?
  - *Hosts? Routers? Both?*

… leads to three design principles

- How to break system into modules?
  - *Layering*
- Where are modules implemented?
  - *End-to-end principle*
- Where is state stored?
  - *Fate-sharing*

Taking Stock

Taking Stock

- Layering is a good way to organize networks
  - *Unified IP layer decouples applications from networks*

Taking Stock

- Layering is a good way to organize networks
  - *Unified IP layer decouples applications from networks*
- E2E argument encourages us to keep IP simple

Taking Stock

- Layering is a good way to organize networks
  - *Unified IP layer decouples applications from networks*
- E2E argument encourages us to keep IP simple
- Fate-sharing principle keeps state out of routers

Taking Stock: So Far…

Taking Stock: So Far…

- What is a network made of?

Taking Stock: So Far…

- What is a network made of?
- How is it shared?

Taking Stock: So Far…

- What is a network made of?
- How is it shared?
- How do we evaluate a network?

Taking Stock: So Far…

- What is a network made of?
- How is it shared?
- How do we evaluate a network?
- How is communication architected?

Taking Stock

Taking Stock

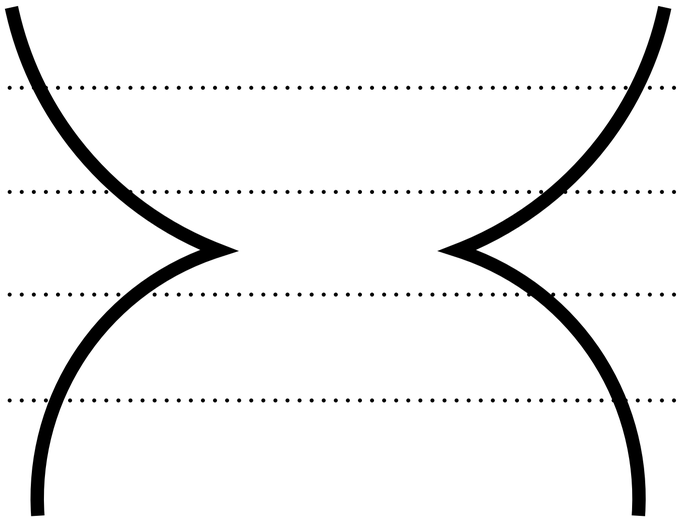
- **Basic concepts**
  - Components: end-systems, links, switches, routers
  - Performance: delay, loss, throughput

Taking Stock

- **Basic concepts**
  - Components: end-systems, links, switches, routers
  - Performance: delay, loss, throughput
- **Basic design choices**
  - Packets vs circuit switching
  - Best-effort vs guaranteed service
  - Layering
  - "Dumb" network, "smart" ends
    - E2E principle, fate-sharing

Coming up next…

- How, in detail, layer by layer (we will obsess a lot over "why" too…)



Coming up next…

- How, in detail, layer by layer (we will obsess a lot over "why" too…)

Coming up next…

- How, in detail, layer by layer (we will obsess a lot over "why" too…)

Coming up next…

- How, in detail, layer by layer (we will obsess a lot over "why" too…)

Coming up next…

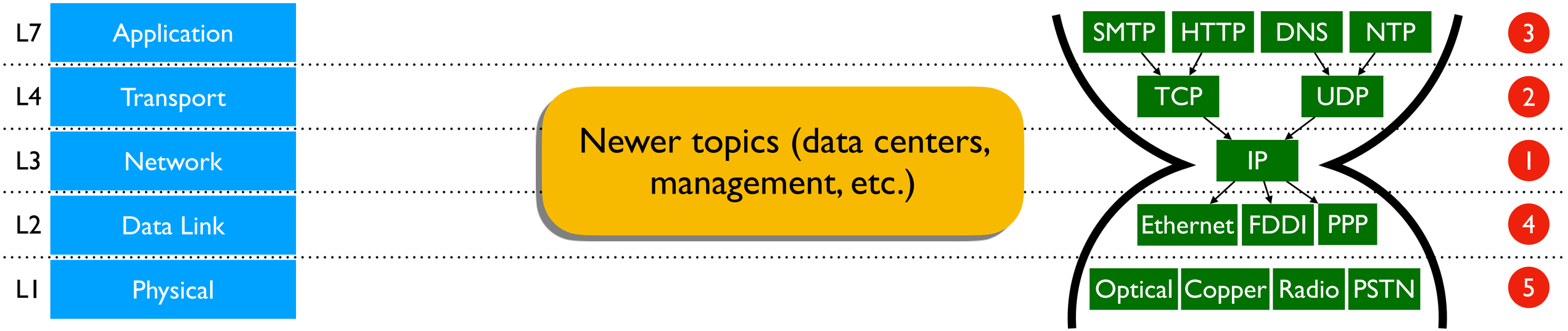
- How, in detail, layer by layer (we will obsess a lot over "why" too…)

Coming up next…

- How, in detail, layer by layer (we will obsess a lot over "why" too…)

Coming up next…

- How, in detail, layer by layer (we will obsess a lot over "why" too…)



The Network Layer

CPSC 433/533, Spring 2021

Anurag Khandelwal

What does the Network Layer do?

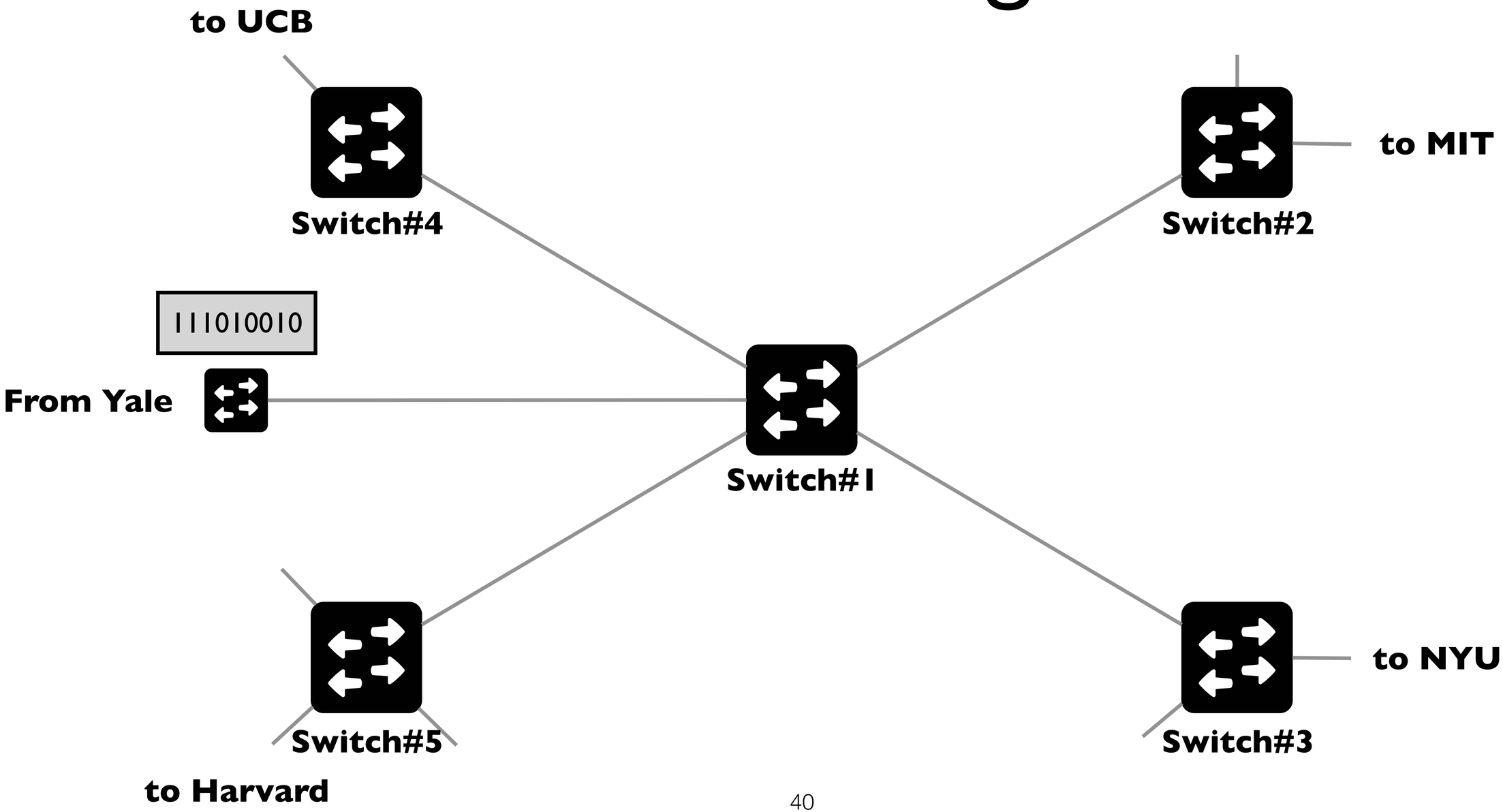
What does the Network Layer do?

- ***Best-effort global delivery of packets***

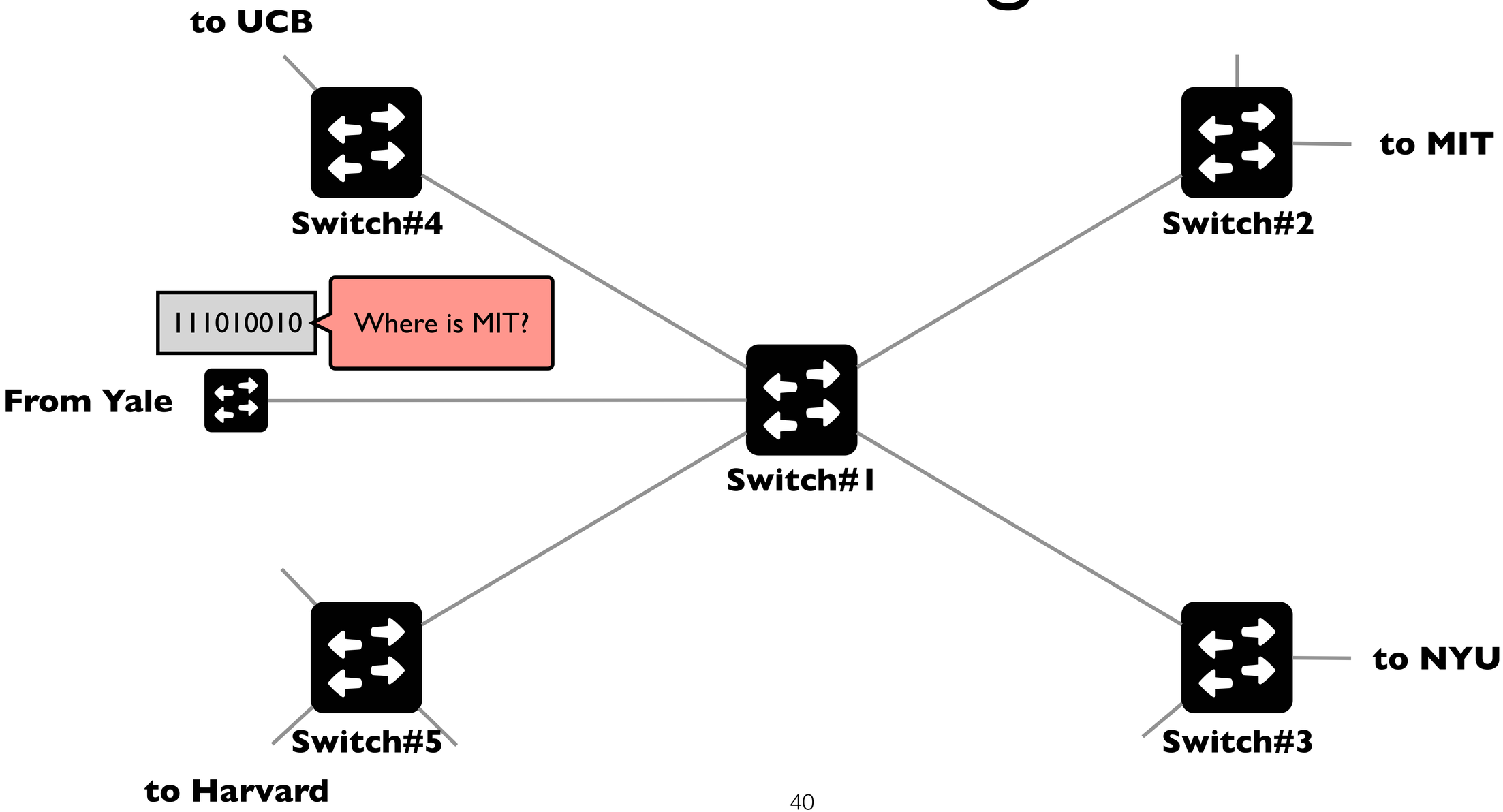
What does the Network Layer do?

- ***Best-effort global delivery of packets***
- Ok, but what does that entail?

Addressing


to UCB
Switch#4
Switch#2
to MIT
111010010
From Yale
Switch#1
Switch#5
Switch#3
to NYU
to Harvard

Addressing


to UCB
Switch#4
Switch#2
to MIT
111010010
Where is MIT?
From Yale
Switch#1
Switch#5
to Harvard
Switch#3
to NYU

Addressing (for now)

Addressing (for now)

- Assume each end-host has a unique address

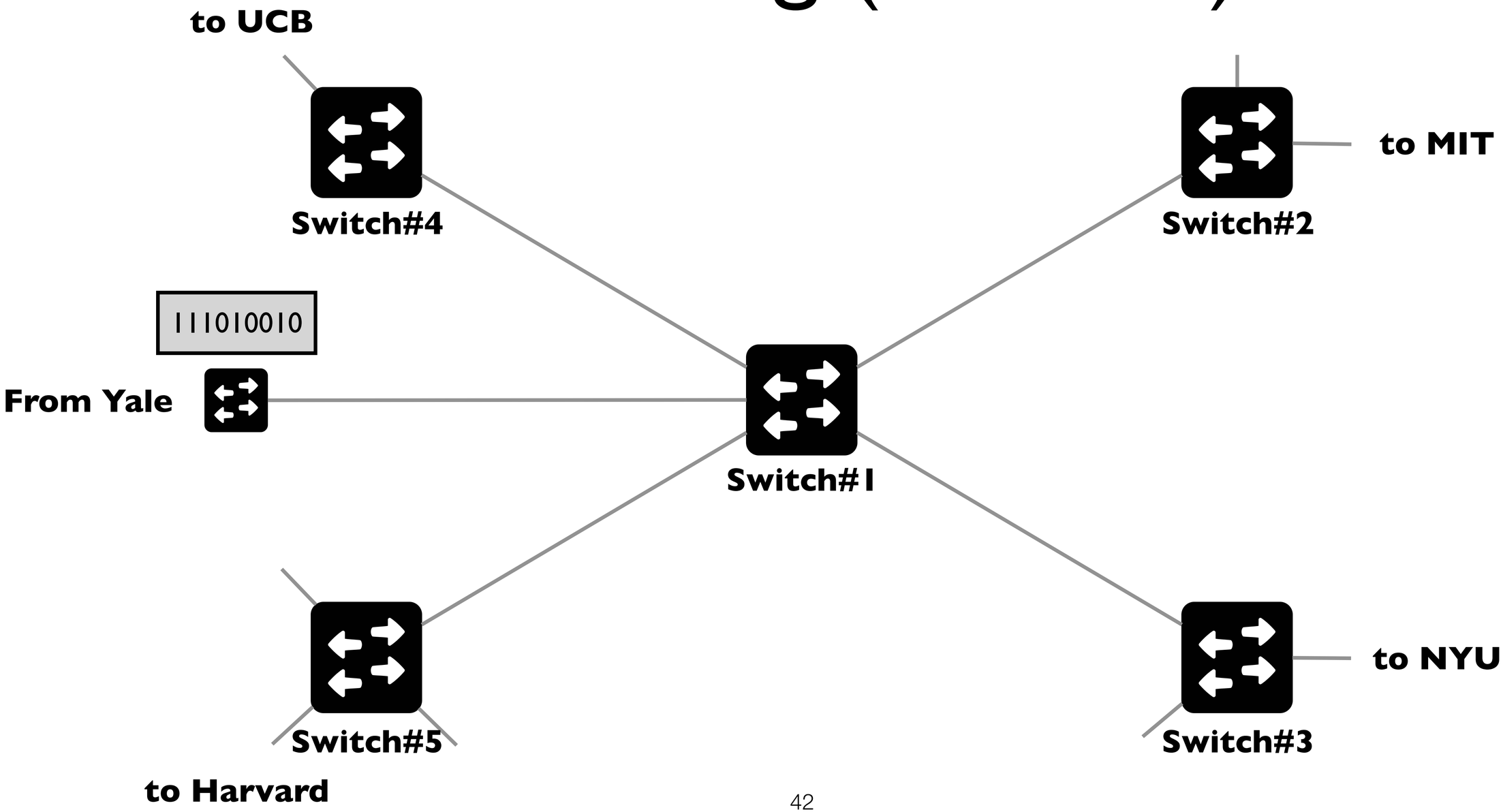
Addressing (for now)

- Assume each end-host has a unique address
- No particular structure to these addresses

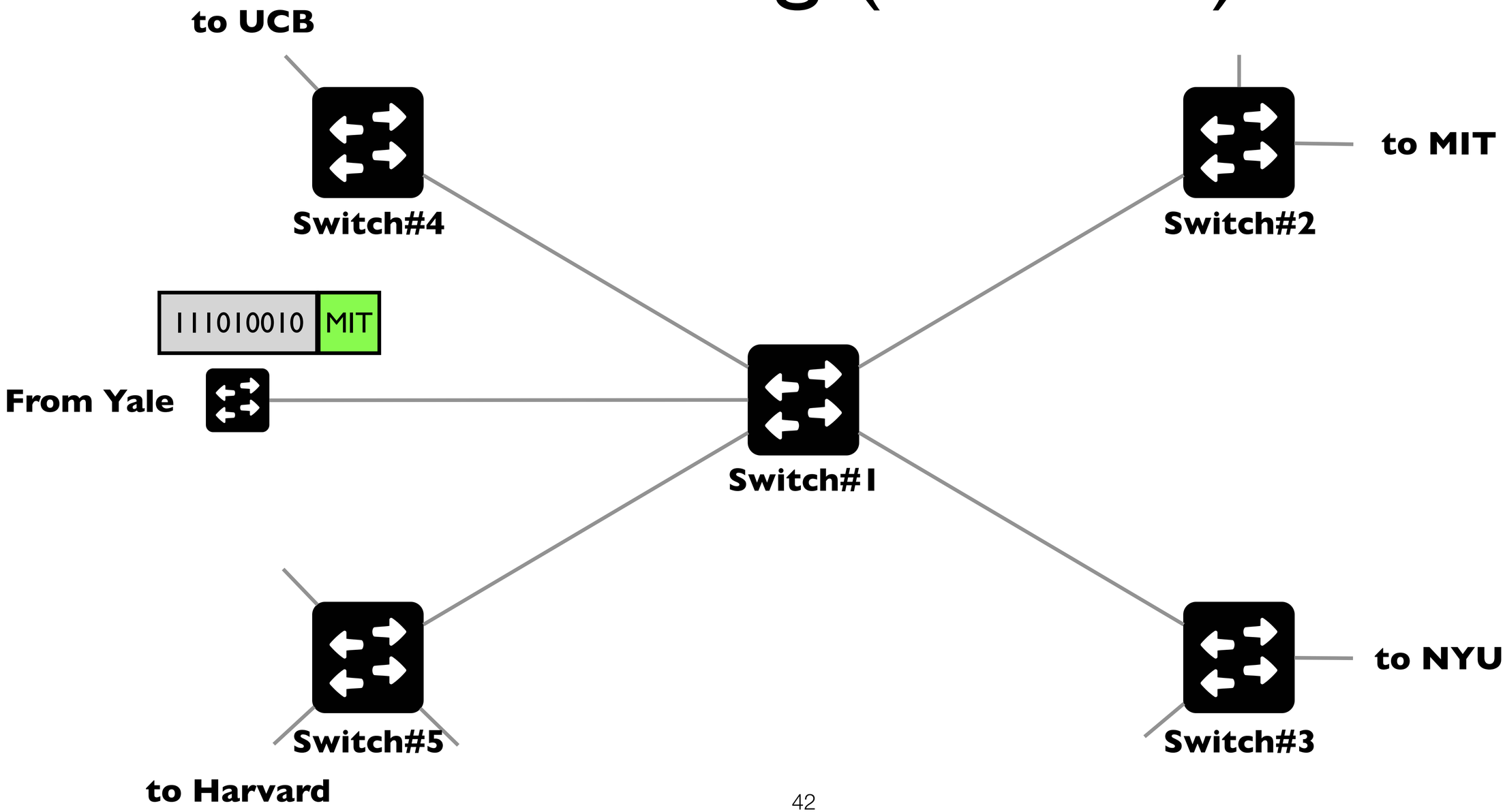
Addressing (for now)

- Assume each end-host has a unique address
- No particular structure to these addresses
- Will cover IP addresses later in the course

Addressing (for now)



Addressing (for now)



Forwarding

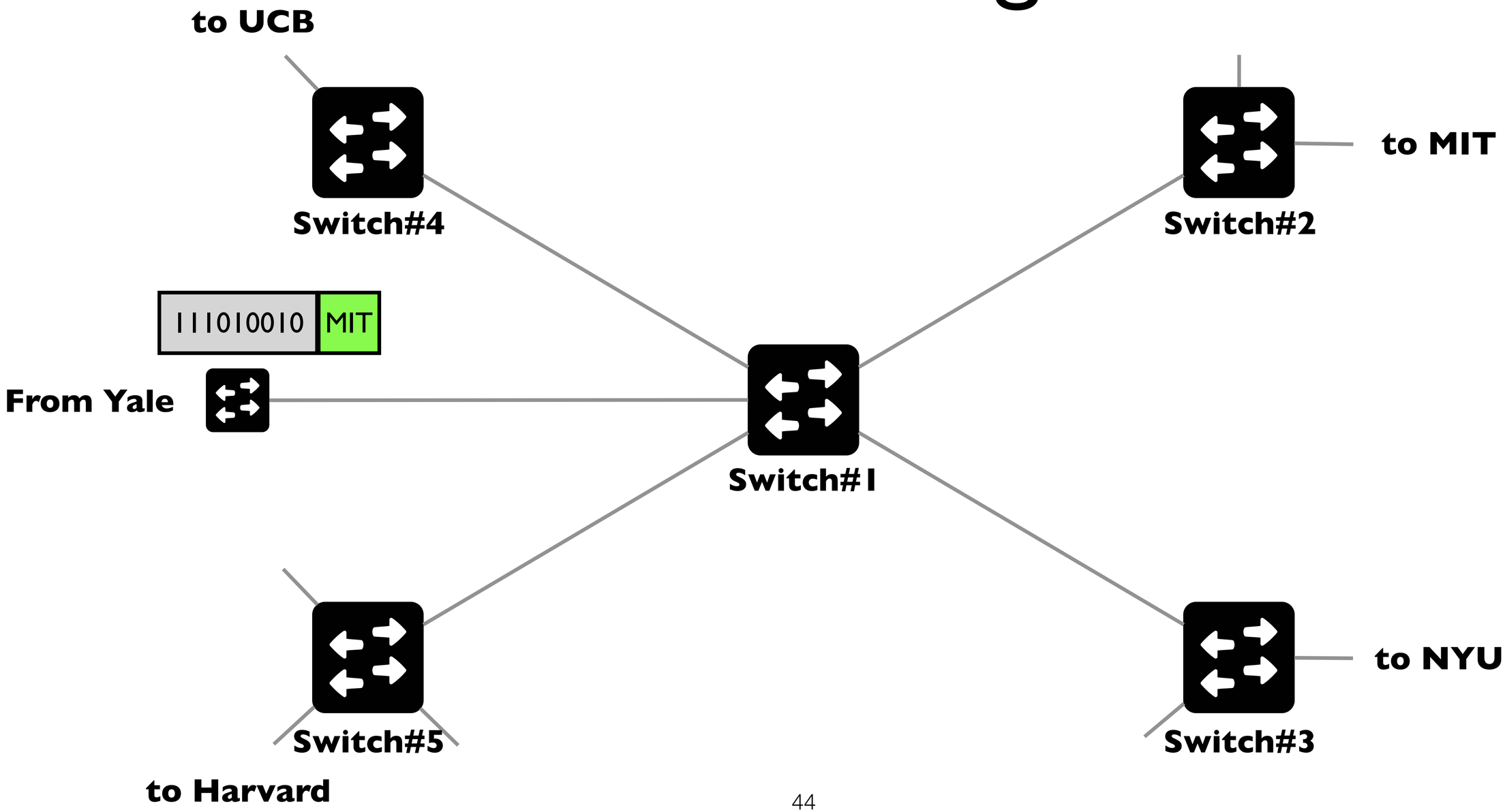
Forwarding

- **Local** router process that determines the output link (aka "next hop") for each packet

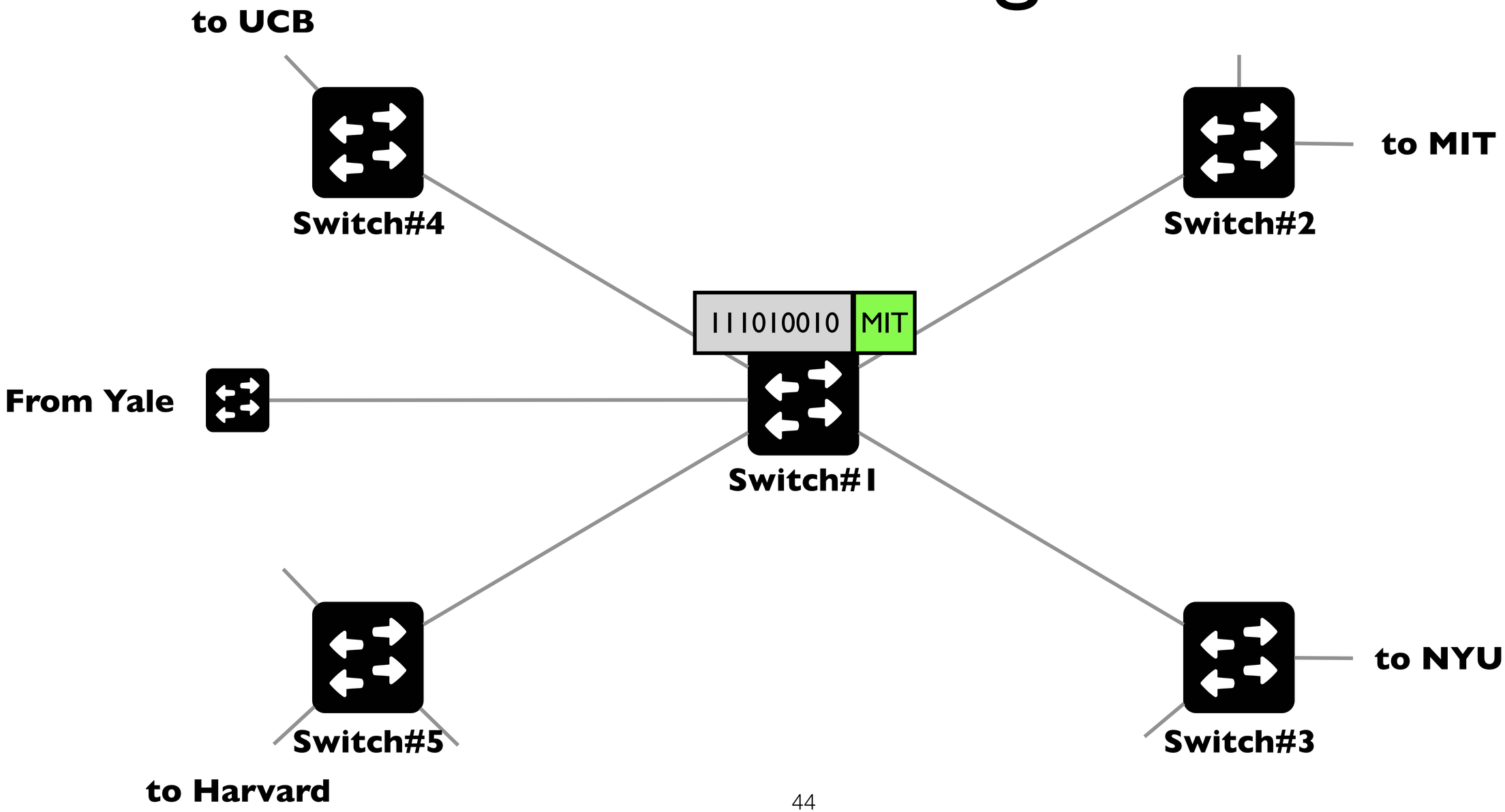
Forwarding

- **Local** router process that determines the output link (aka "next hop") for each packet
- **How?**
  - Read address from packet's network layer header
  - Search forwarding table

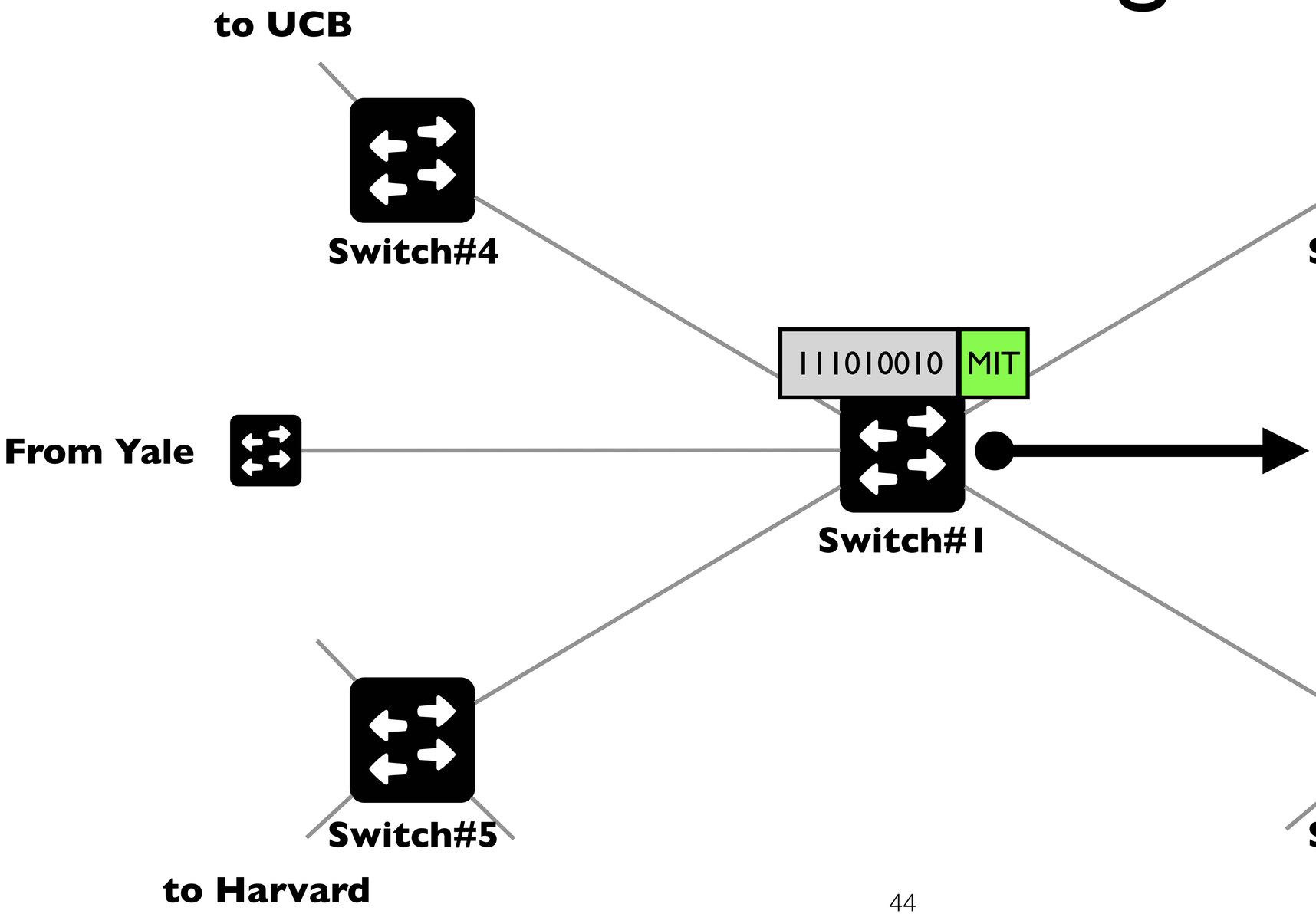
Forwarding


to UCB
Switch#4
Switch#2
to MIT
111010010
MIT
From Yale
Switch#1
Switch#5
Switch#3
to NYU
to Harvard

Forwarding


to UCB
Switch#4
Switch#2
to MIT
111010010
MIT
From Yale
Switch#1
Switch#5
Switch#3
to NYU
to Harvard

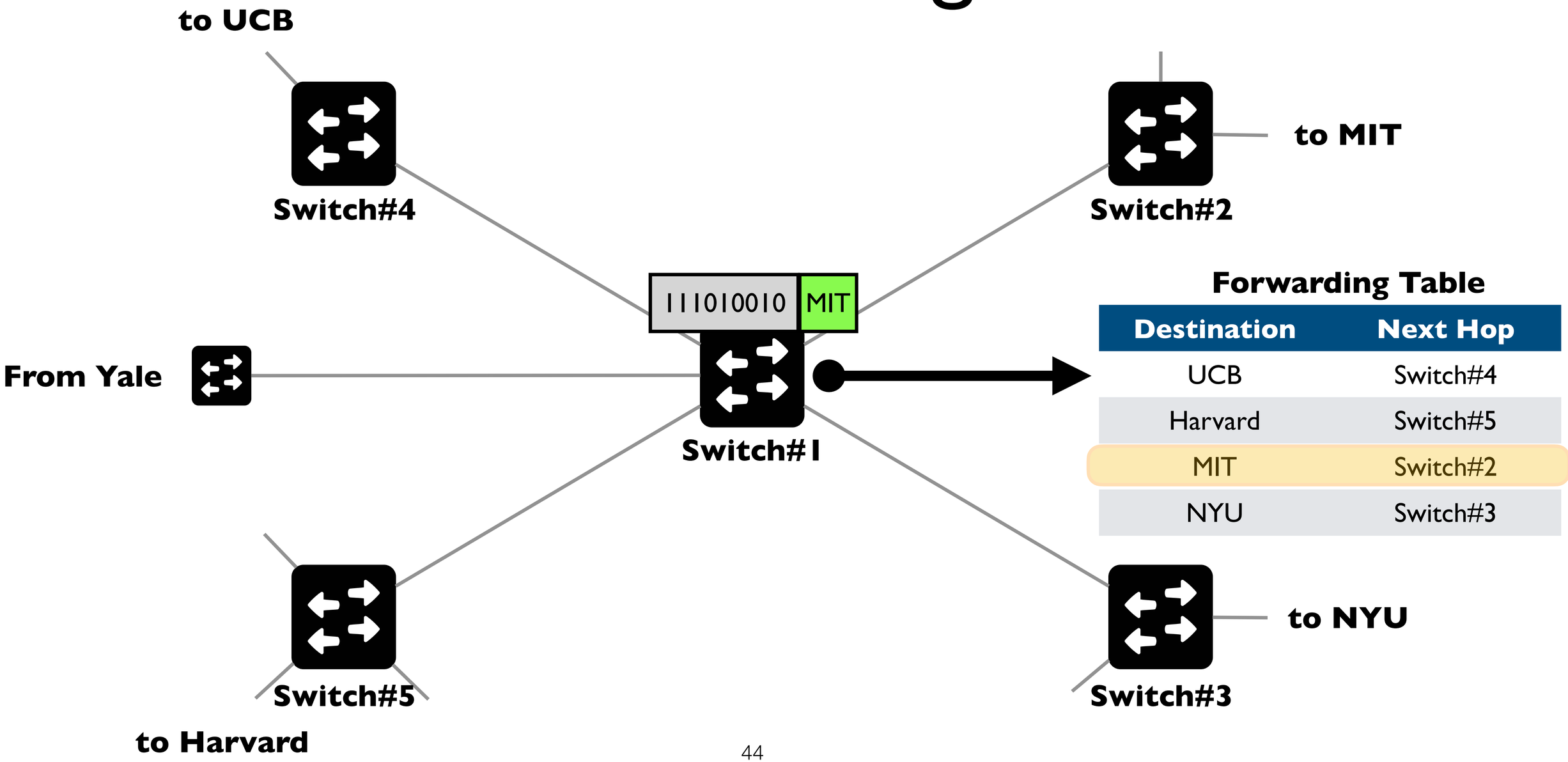
Forwarding



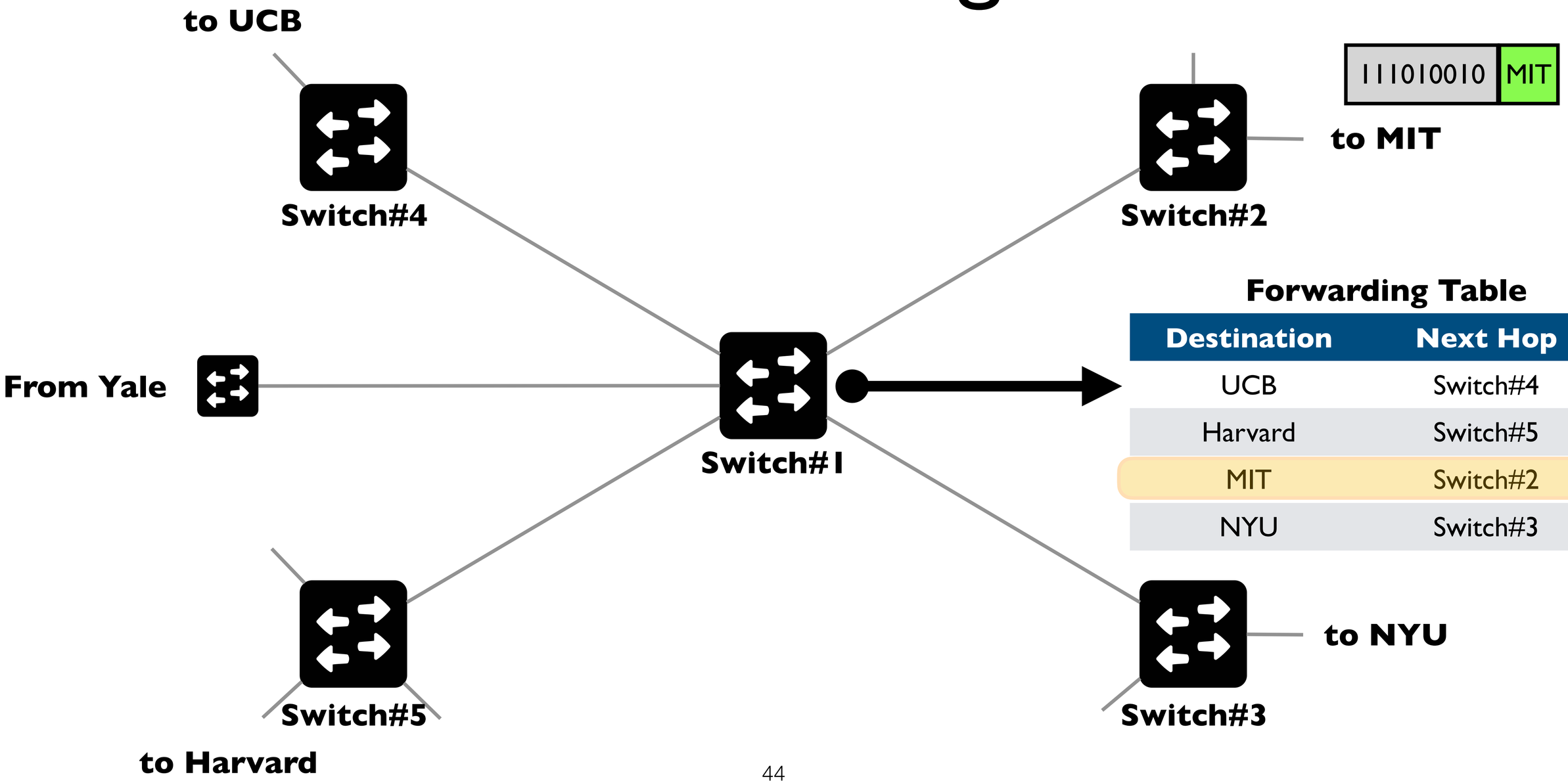
Forwarding Table

Destination	Next Hop
UCB	Switch#4
Harvard	Switch#5
MIT	Switch#2
NYU	Switch#3

Forwarding


to UCB
Switch#4
Switch#2
to MIT
111010010
MIT
From Yale
Switch#1
Forwarding Table
Destination
Next Hop
UCB
Switch#4
Harvard
Switch#5
MIT
Switch#2
NYU
Switch#3
Switch#5
to Harvard
to NYU
Switch#3

Forwarding

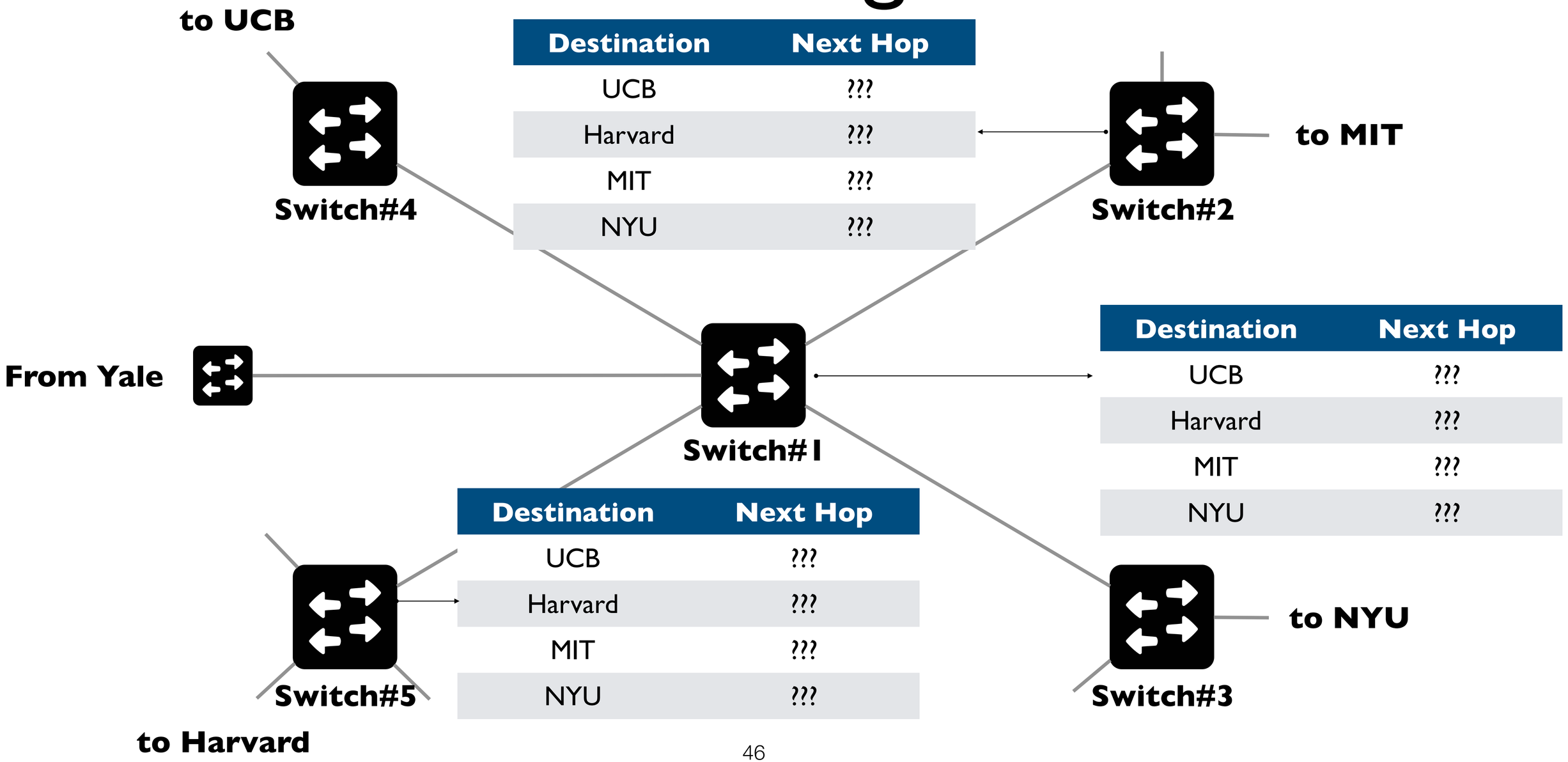

to UCB
Switch#4
Switch#2
111010010
MIT
to MIT
From Yale
Switch#1
Forwarding Table
Destination
Next Hop
UCB
Switch#4
Harvard
Switch#5
MIT
Switch#2
NYU
Switch#3
Switch#5
to Harvard
to NYU
Switch#3

Routing

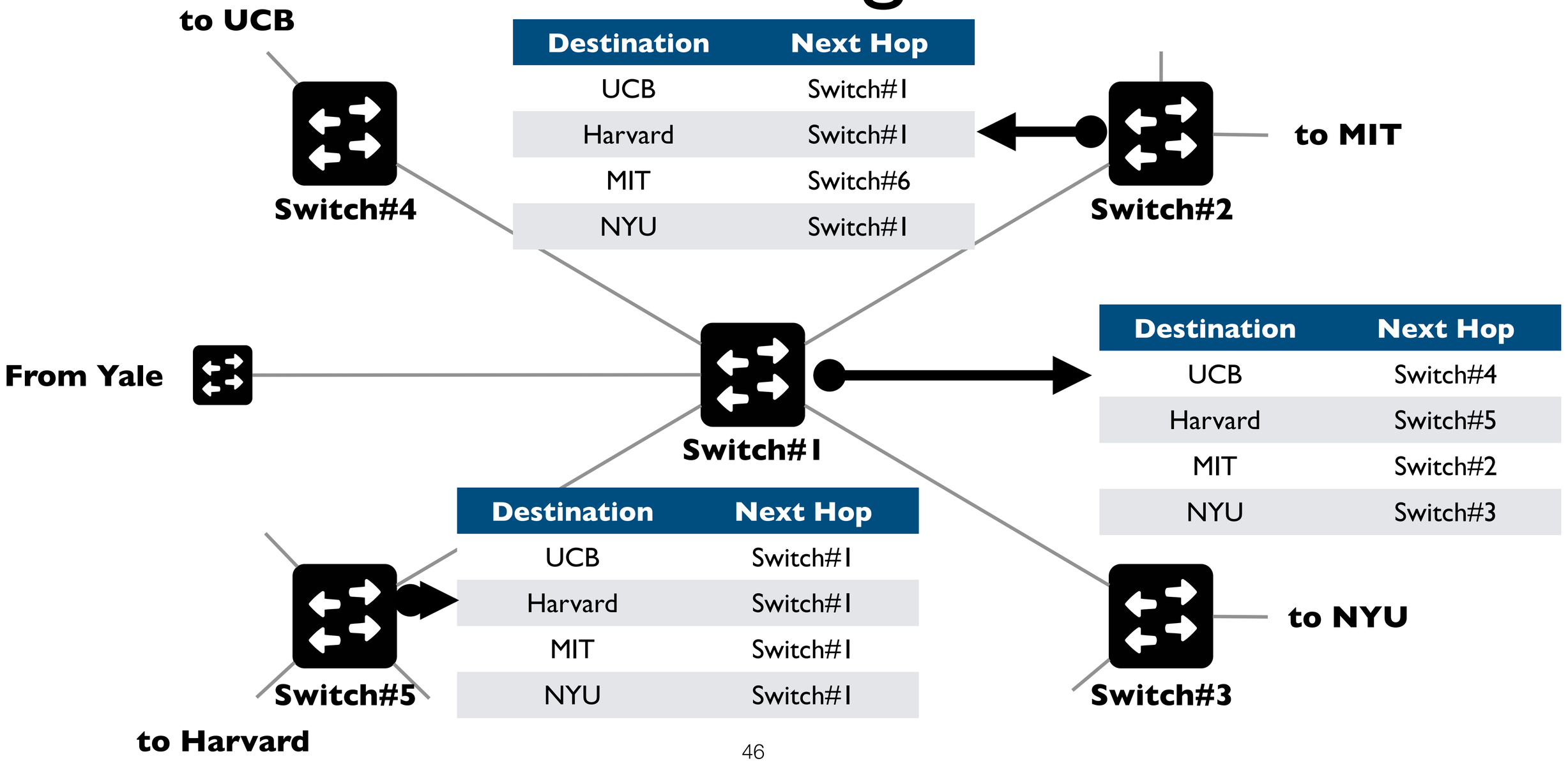
Routing

- **Network-wide** process that determines the content of forwarding tables
  - Determines the end-to-end path for each destination

Routing


to UCB
Switch#4
Destination
Next Hop
UCB
???
Harvard
???
MIT
???
NYU
???
Switch#2
to MIT
From Yale
Switch#1
Destination
Next Hop
UCB
???
Harvard
???
MIT
???
NYU
???
Switch#5
to Harvard
Destination
Next Hop
UCB
???
Harvard
???
MIT
???
NYU
???
Switch#3
to NYU

Routing



Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#6
NYU	Switch#1

Destination	Next Hop
UCB	Switch#4
Harvard	Switch#5
MIT	Switch#2
NYU	Switch#3

Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

Routing

Routing

- **Network-wide** process that determines the content of forwarding tables
  - Determines the end-to-end path for each destination

Routing

- **Network-wide** process that determines the content of forwarding tables
  - Determines the end-to-end path for each destination
- **How?**
  - Coming up soon…

Forwarding vs. Routing

Forwarding vs. Routing

- **Forwarding:** "data plane"
  - Directing one data packet
  - Each router using local routing state

Forwarding vs. Routing

- **Forwarding:** "data plane"
  - Directing one data packet
  - Each router using local routing state
- **Routing:** "control plane"
  - Computing the forwarding tables that guide packets
  - Jointly computed by routers using a distributed algorithm

Forwarding vs. Routing

- **Forwarding:** "data plane"
  - Directing one data packet
  - Each router using local routing state
- **Routing:** "control plane"
  - Computing the forwarding tables that guide packets
  - Jointly computed by routers using a distributed algorithm
- Very different timescales!

Goals (for Routing)

Goals (for Routing)

- **V1: Find a path to a given destination**

Goals (for Routing)

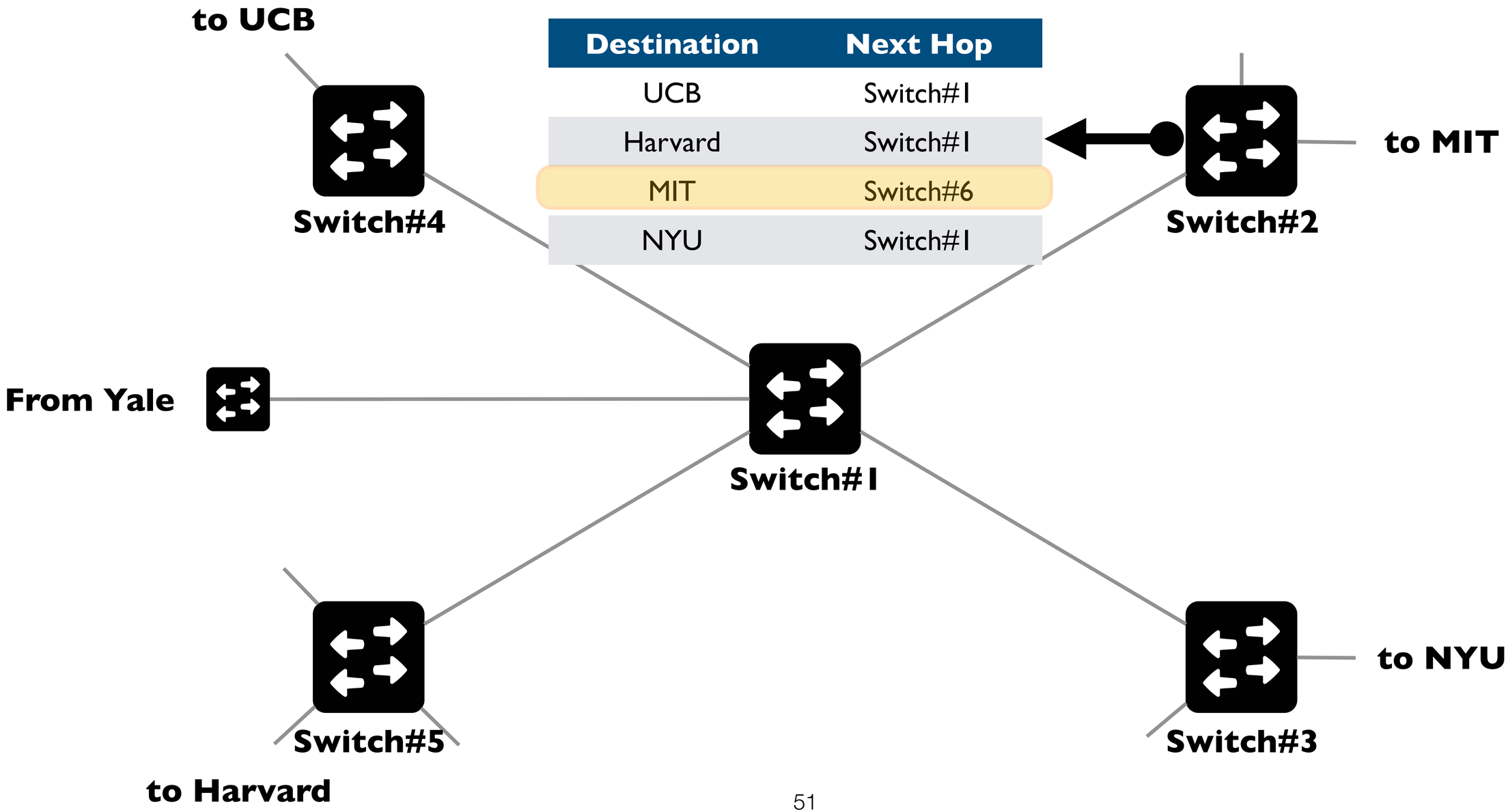
- **V1: Find a path to a given destination**
- How do we know that the state contained in forwarding tables meets our goal?
  - This is what "validity" of routing state tells us
  - [this is non-standard terminology]

Local vs. Global view of state

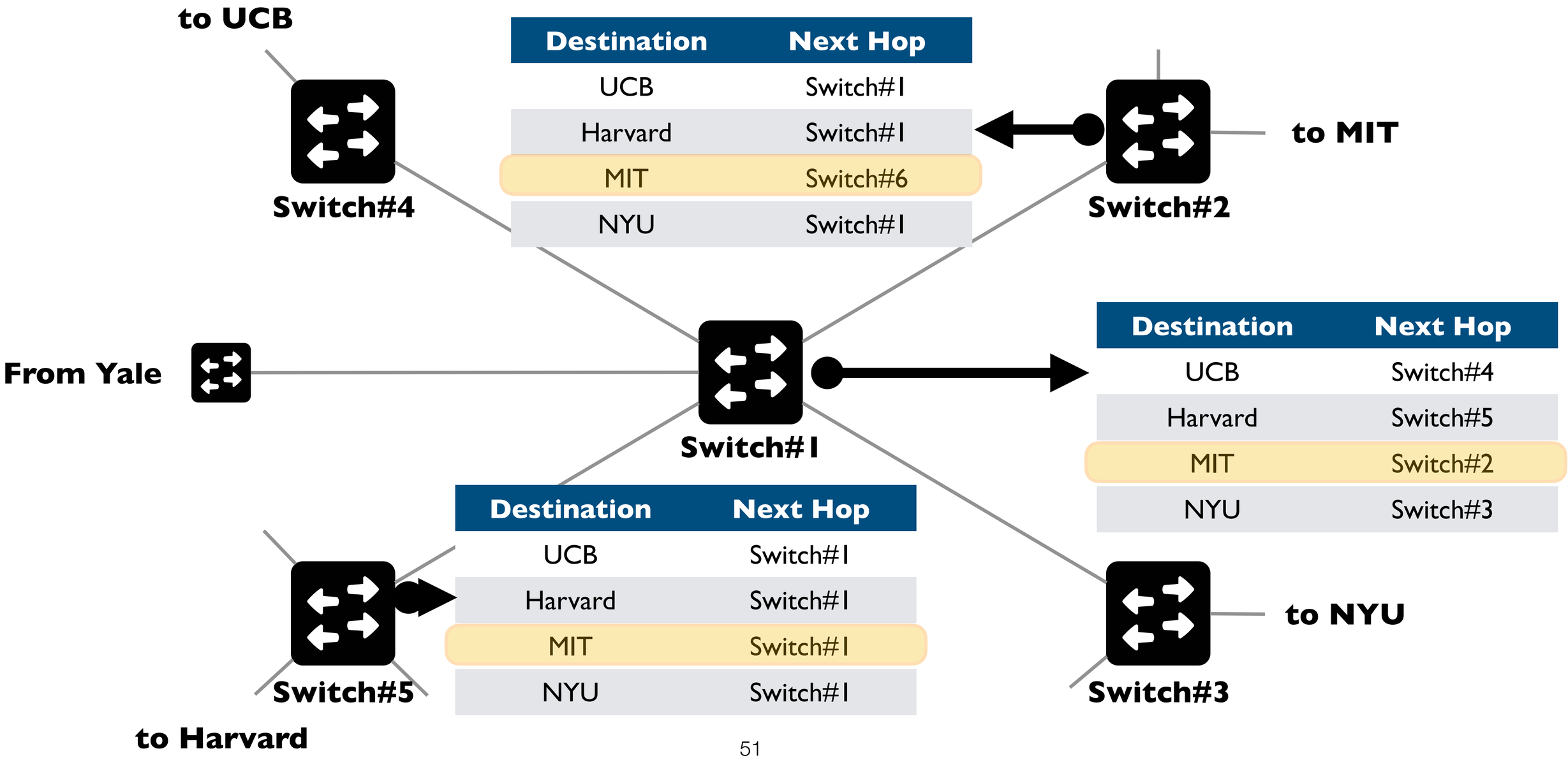
Local vs. Global view of state

- **Local** routing state is the forwarding table in a single router
  - By itself, the state in a single router cannot be evaluated for validity
  - It must be evaluated in terms of the global context

Local vs. Global view of state



Local vs. Global view of state



Local vs. Global view of state

Local vs. Global view of state

- **Local** routing state is the forwarding table in a single router
  - By itself, the state in a single router cannot be evaluated for validity
  - It must be evaluated in terms of the global context

Local vs. Global view of state

- **Local** routing state is the forwarding table in a single router
  - By itself, the state in a single router cannot be evaluated for validity
  - It must be evaluated in terms of the global context
- **Global** state refers to the collection of forwarding tables in each of the routers
  - Global state determines which paths packet take

    (Will discuss later *where* the state comes from)

“Valid” Routing State

“Valid” Routing State

- Global state is **“valid”** if it produces forwarding decisions that always deliver packets to their destinations

“Valid” Routing State

- Global state is **“valid”** if it produces forwarding decisions that always deliver packets to their destinations
- **Goal of routing protocols:** compute valid state
  - But how can you tell if routing state is valid

“Valid” Routing State

- Global state is **“valid”** if it produces forwarding decisions that always deliver packets to their destinations
- **Goal of routing protocols:** compute valid state
  - But how can you tell if routing state is valid
- Need a **succinct** correctness condition for routing
  - *Suggestions?*

Necessary and Sufficient Condition

Necessary and Sufficient Condition

- Global routing state is "valid" ***if and only if:***
  - There are no dead-ends (other than the destination)
  - There are no loops

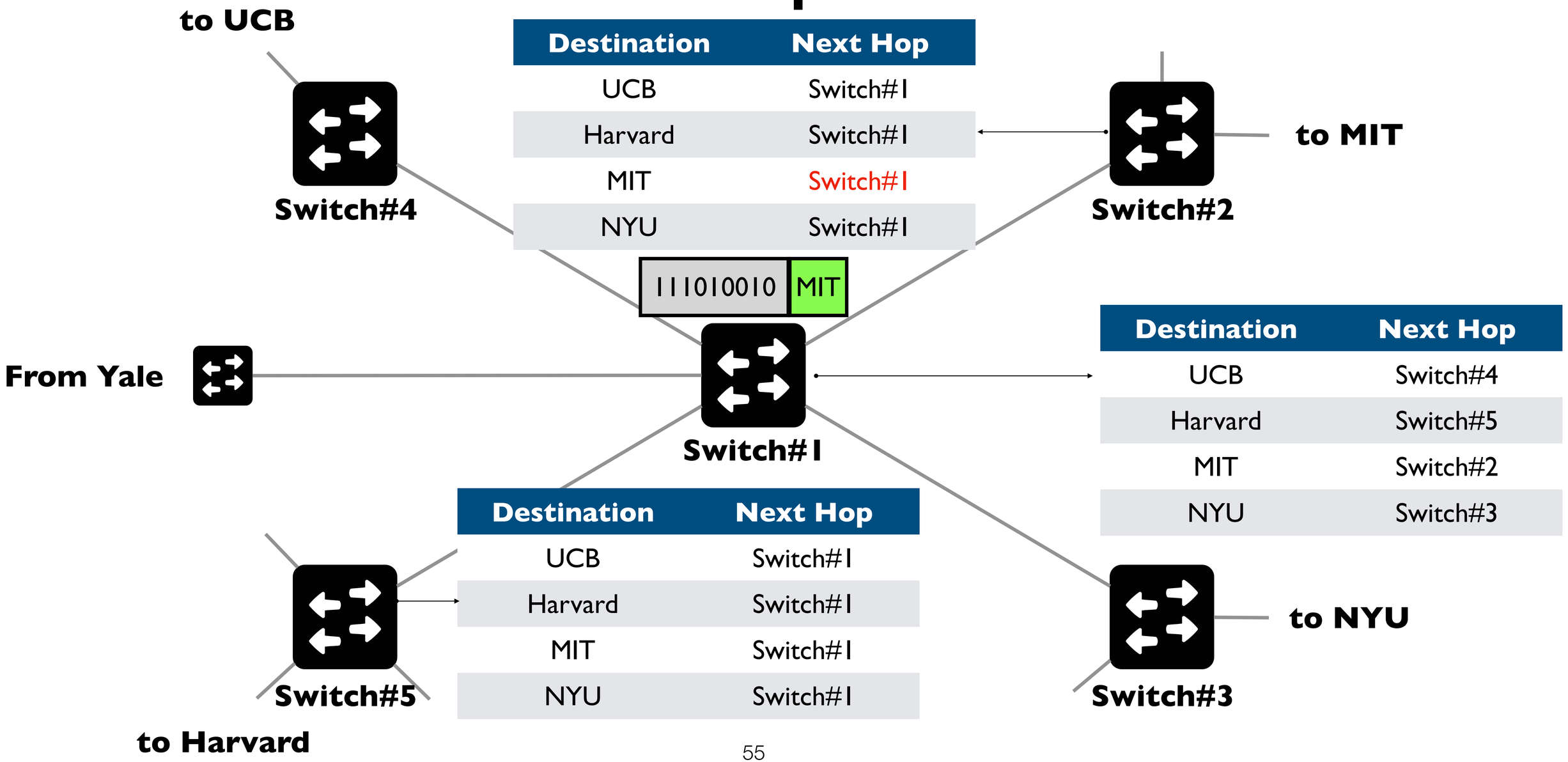
Necessary and Sufficient Condition

- Global routing state is "valid" ***if and only if:***
  - There are no dead-ends (other than the destination)
  - There are no loops
- Dead-end: when there is no outgoing link (next-hop)
  - A packet arrives, but the forwarding decision does not yield any outgoing link

Necessary and Sufficient Condition

- Global routing state is "valid" ***if and only if:***
  - There are no dead-ends (other than the destination)
  - There are no loops
- Dead-end: when there is no outgoing link (next-hop)
  - A packet arrives, but the forwarding decision does not yield any outgoing link
- Loop: when a packet cycles around the same set of nodes forever

Loops

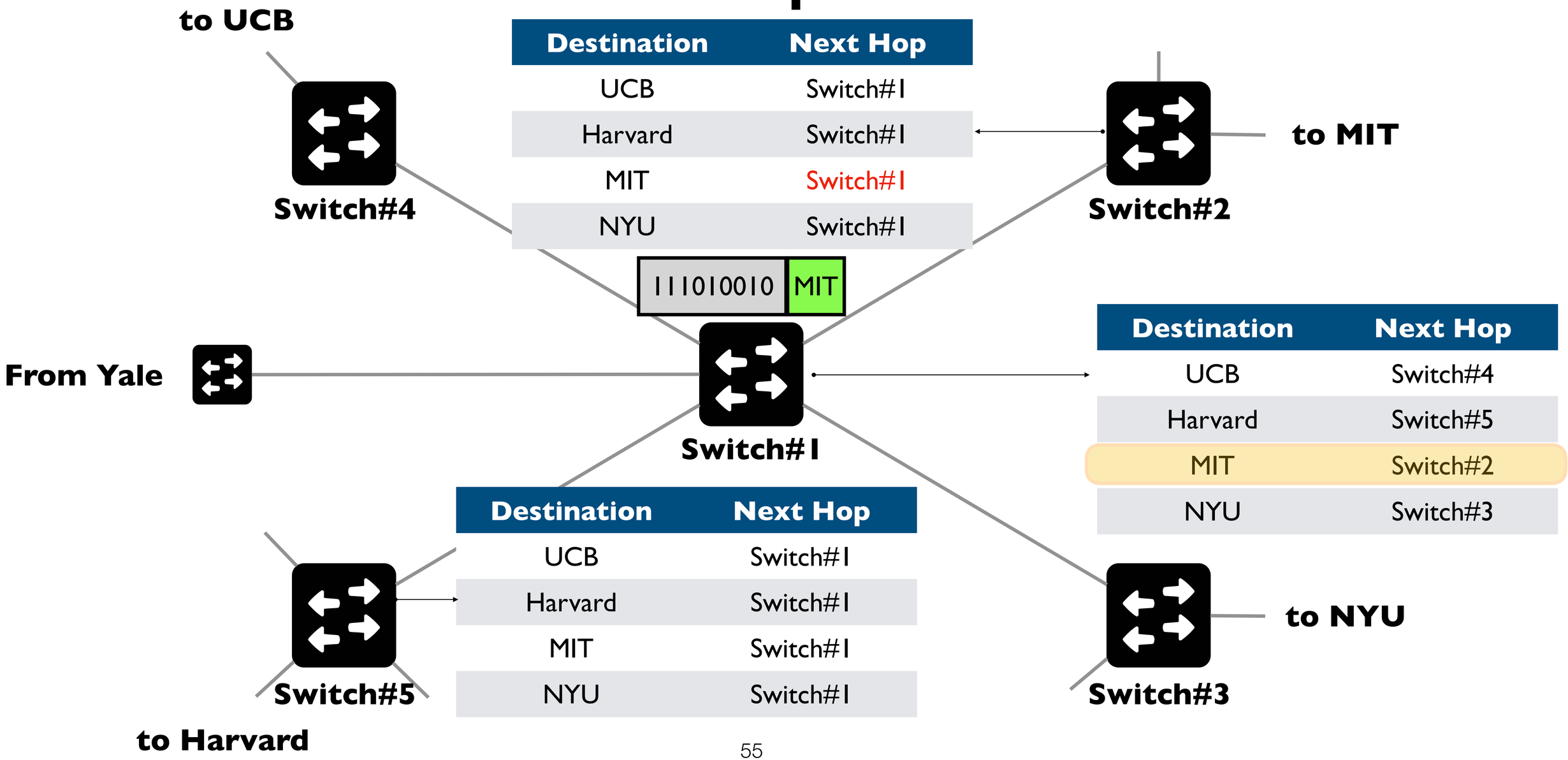


Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

Destination	Next Hop
UCB	Switch#4
Harvard	Switch#5
MIT	Switch#2
NYU	Switch#3

Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

Loops

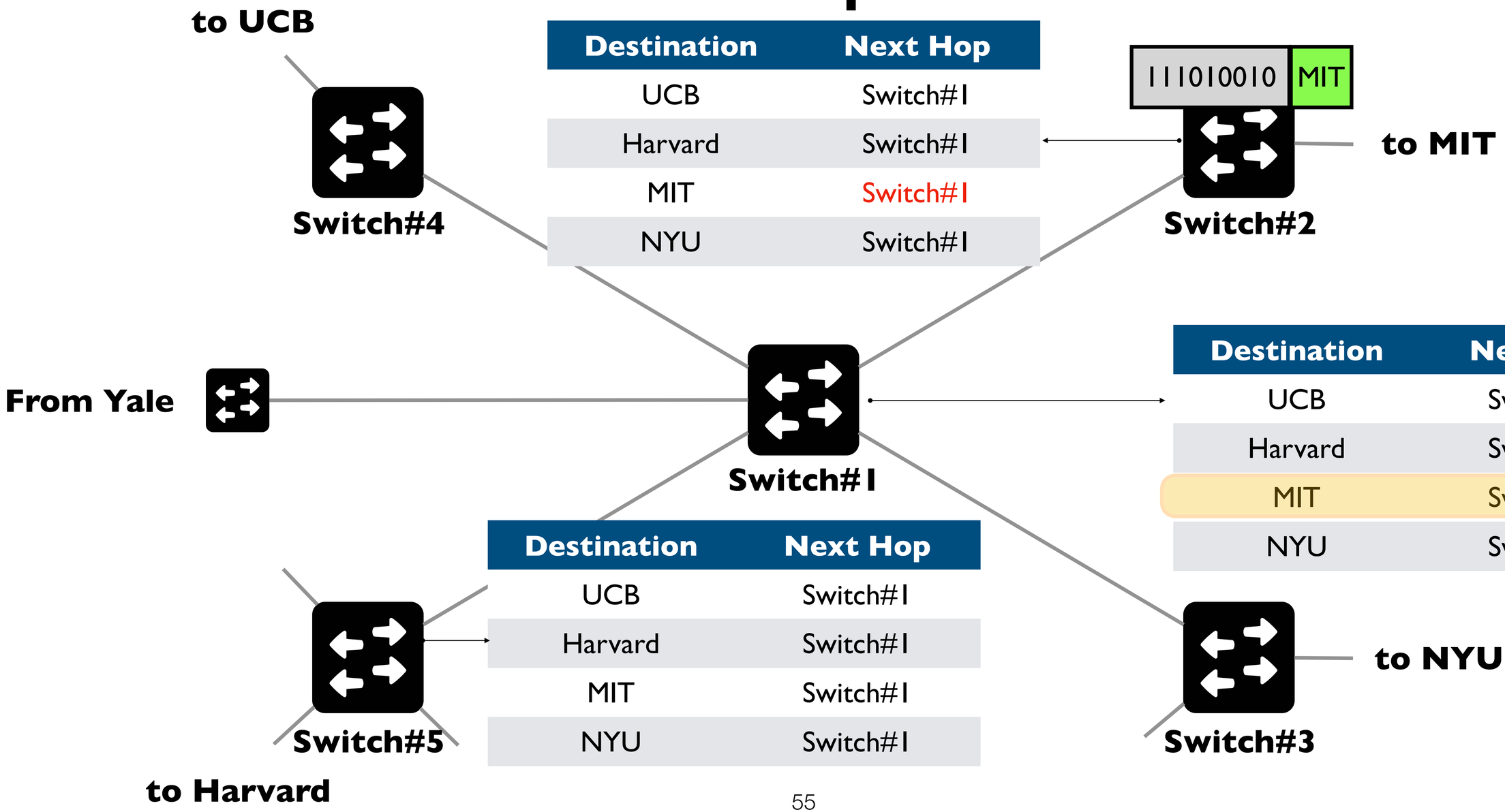


Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

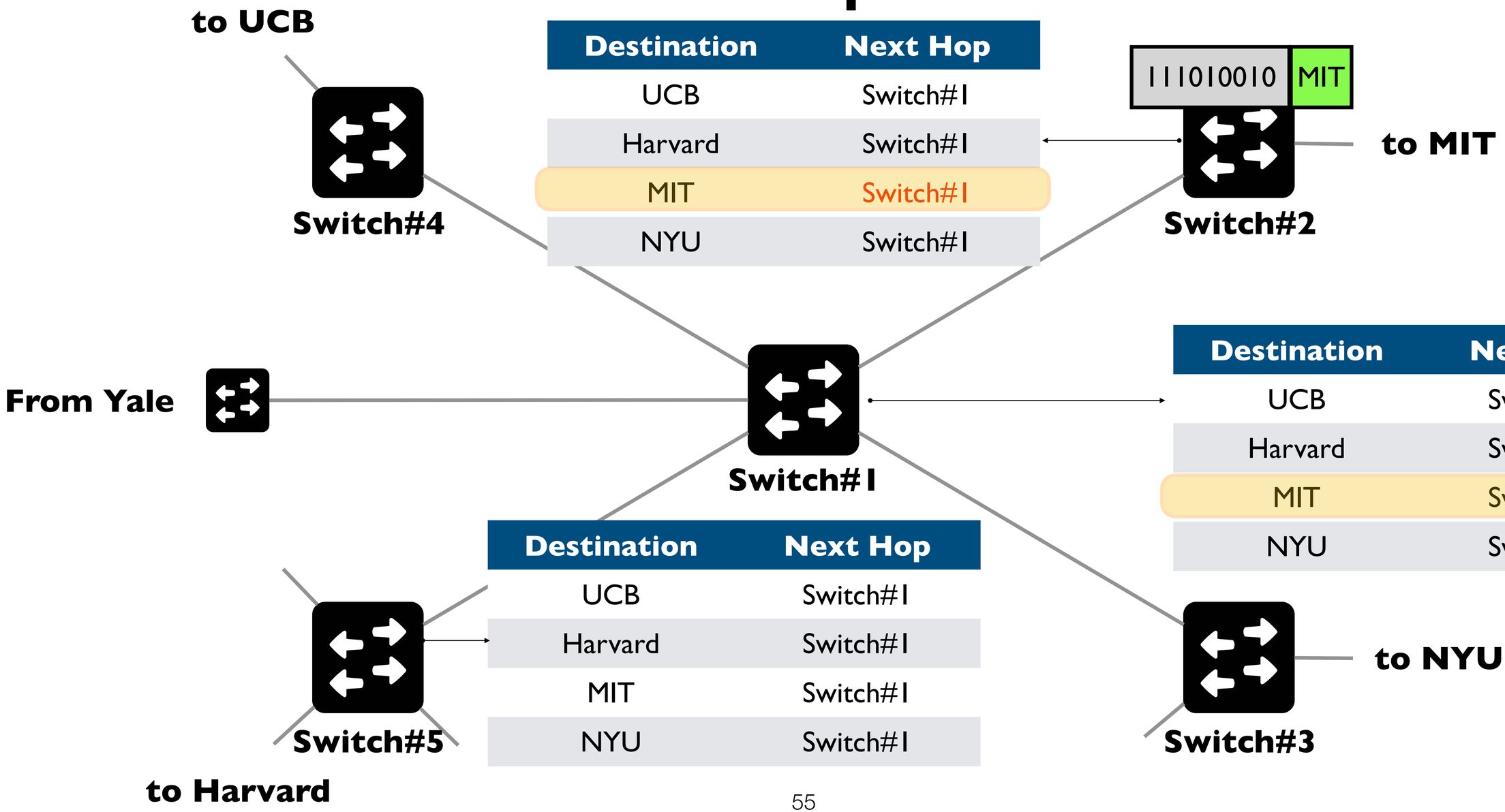
Destination	Next Hop
UCB	Switch#4
Harvard	Switch#5
MIT	Switch#2
NYU	Switch#3

Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

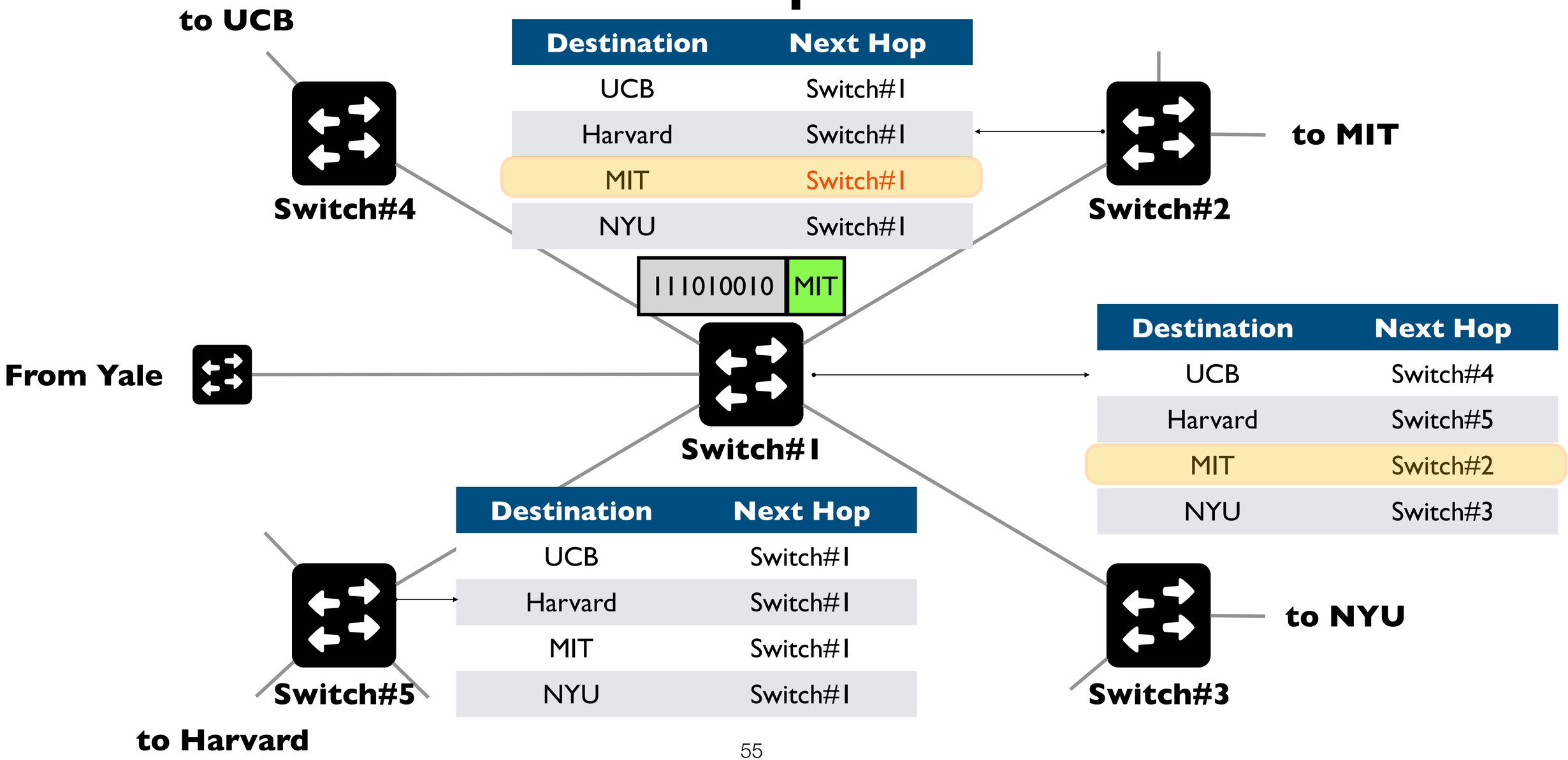
Loops


to UCB
Switch#4
Destination
Next Hop
UCB
Switch#1
Harvard
Switch#1
MIT
Switch#1
NYU
Switch#1
111010010
MIT
Switch#2
to MIT
From Yale
Switch#1
Destination
Next Hop
UCB
Switch#4
Harvard
Switch#5
MIT
Switch#2
NYU
Switch#3
Destination
Next Hop
UCB
Switch#1
Harvard
Switch#1
MIT
Switch#1
NYU
Switch#1
Switch#5
to Harvard
Switch#3
to NYU

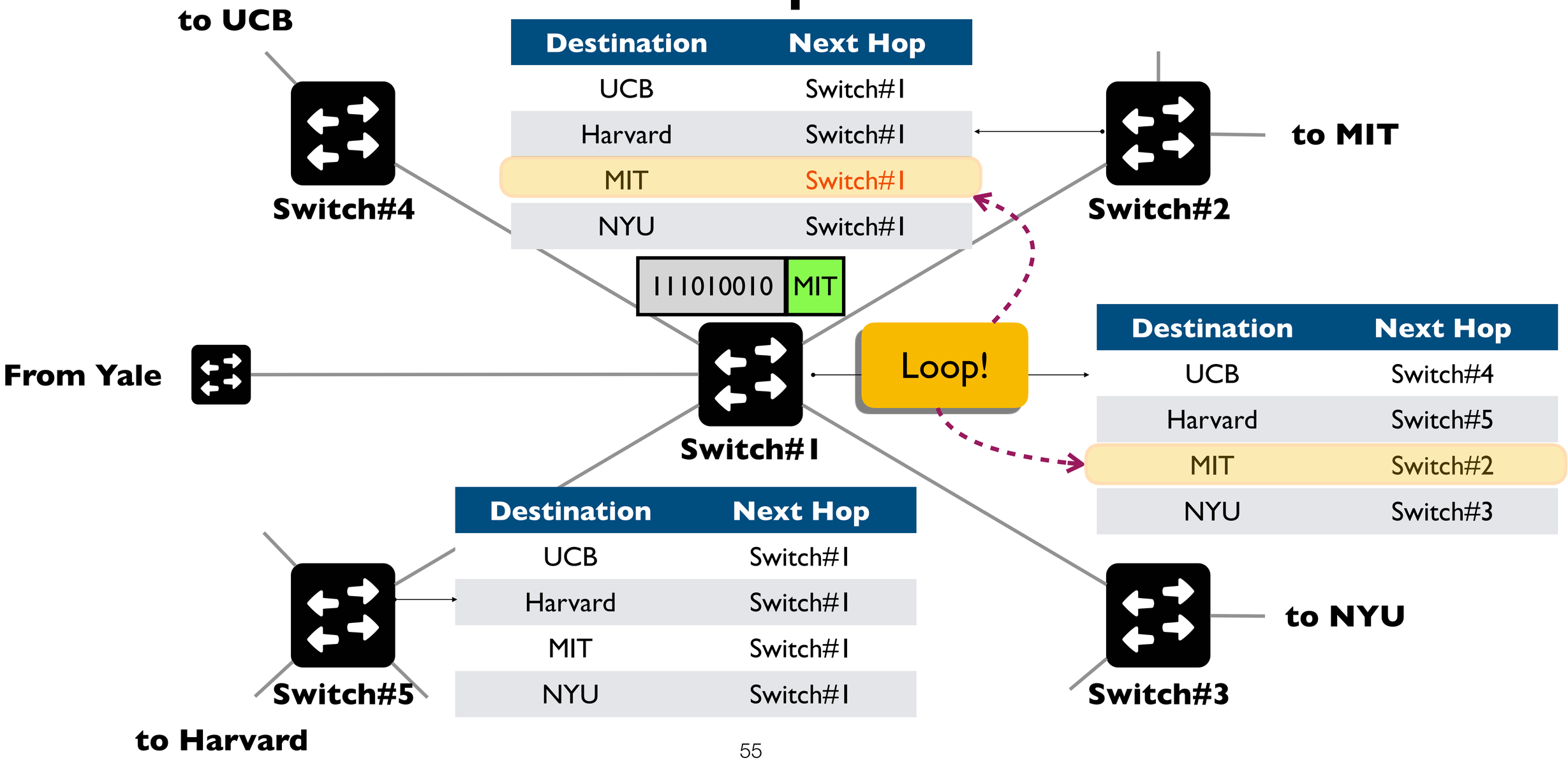
Loops


to UCB
Switch#4
Destination
Next Hop
UCB
Switch#1
Harvard
Switch#1
MIT
Switch#1
NYU
Switch#1
111010010
MIT
Switch#2
to MIT
From Yale
Switch#1
Destination
Next Hop
UCB
Switch#4
Harvard
Switch#5
MIT
Switch#2
NYU
Switch#3
Destination
Next Hop
UCB
Switch#1
Harvard
Switch#1
MIT
Switch#1
NYU
Switch#1
Switch#5
to Harvard
Switch#3
to NYU

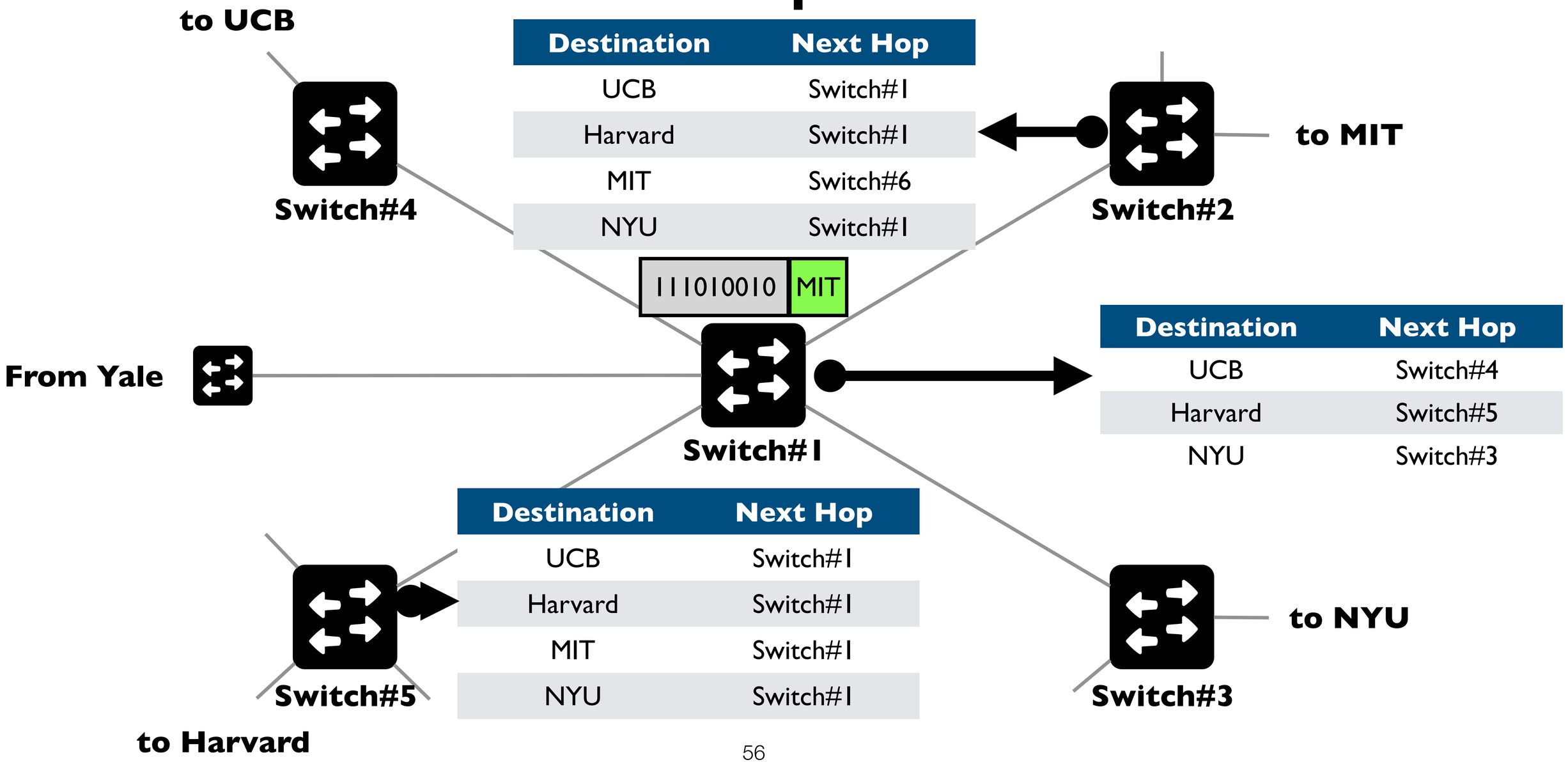
Loops


to UCB
Switch#4
Destination
Next Hop
UCB
Switch#1
Harvard
Switch#1
MIT
Switch#1
NYU
Switch#1
111010010
MIT
Switch#2
to MIT
From Yale
Switch#1
Destination
Next Hop
UCB
Switch#4
Harvard
Switch#5
MIT
Switch#2
NYU
Switch#3
Destination
Next Hop
UCB
Switch#1
Harvard
Switch#1
MIT
Switch#1
NYU
Switch#1
Switch#5
to Harvard
Switch#3
to NYU

Loops


to UCB
Switch#4
Destination
Next Hop
UCB
Switch#1
Harvard
Switch#1
MIT
Switch#1
NYU
Switch#1
111010010
MIT
to MIT
Switch#2
From Yale
Switch#1
Loop!
Destination
Next Hop
UCB
Switch#4
Harvard
Switch#5
MIT
Switch#2
NYU
Switch#3
Destination
Next Hop
UCB
Switch#1
Harvard
Switch#1
MIT
Switch#1
NYU
Switch#1
Switch#5
to Harvard
to NYU
Switch#3

Loops

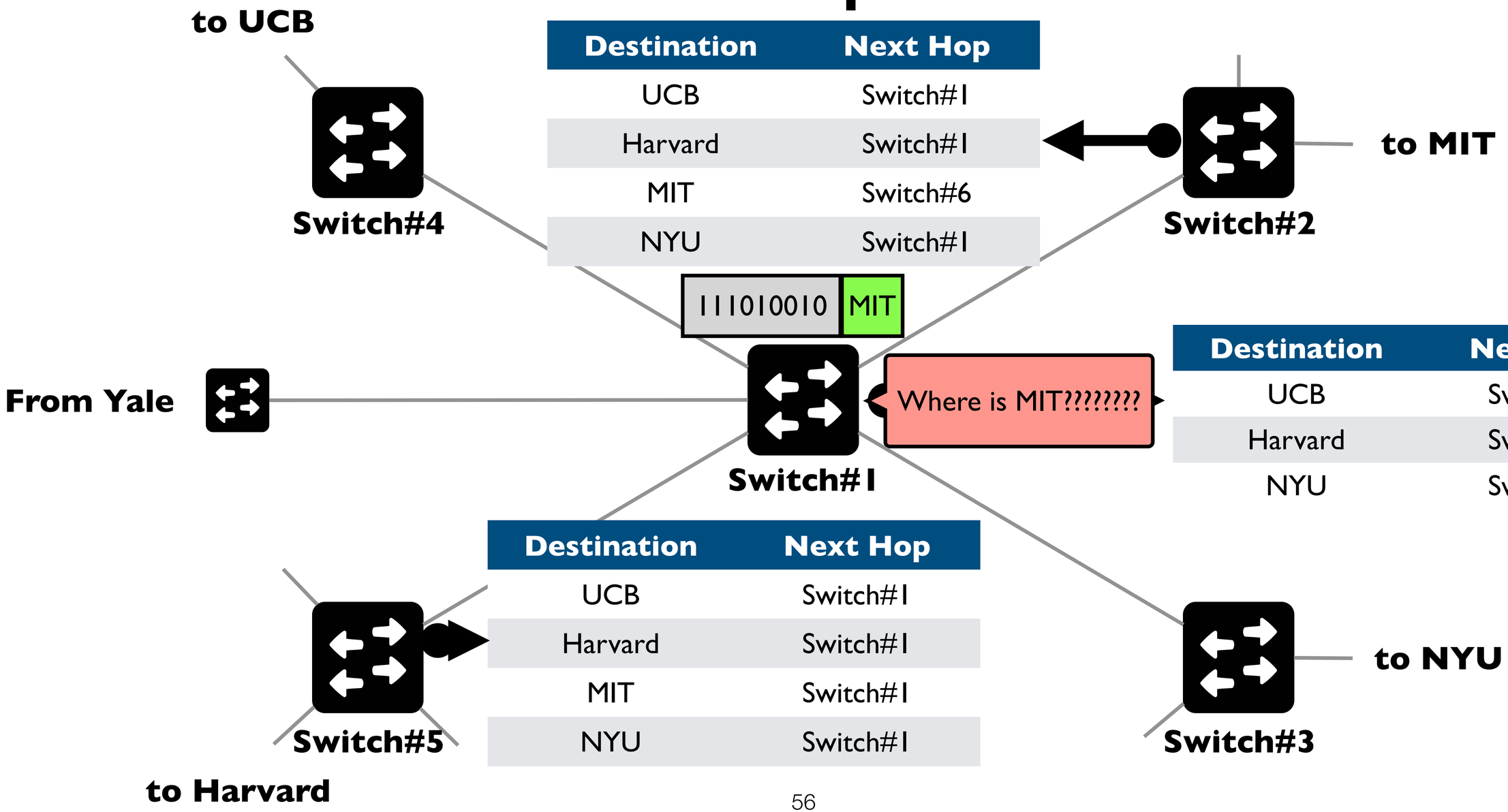


Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#6
NYU	Switch#1

Destination	Next Hop
UCB	Switch#4
Harvard	Switch#5
NYU	Switch#3

Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

Loops

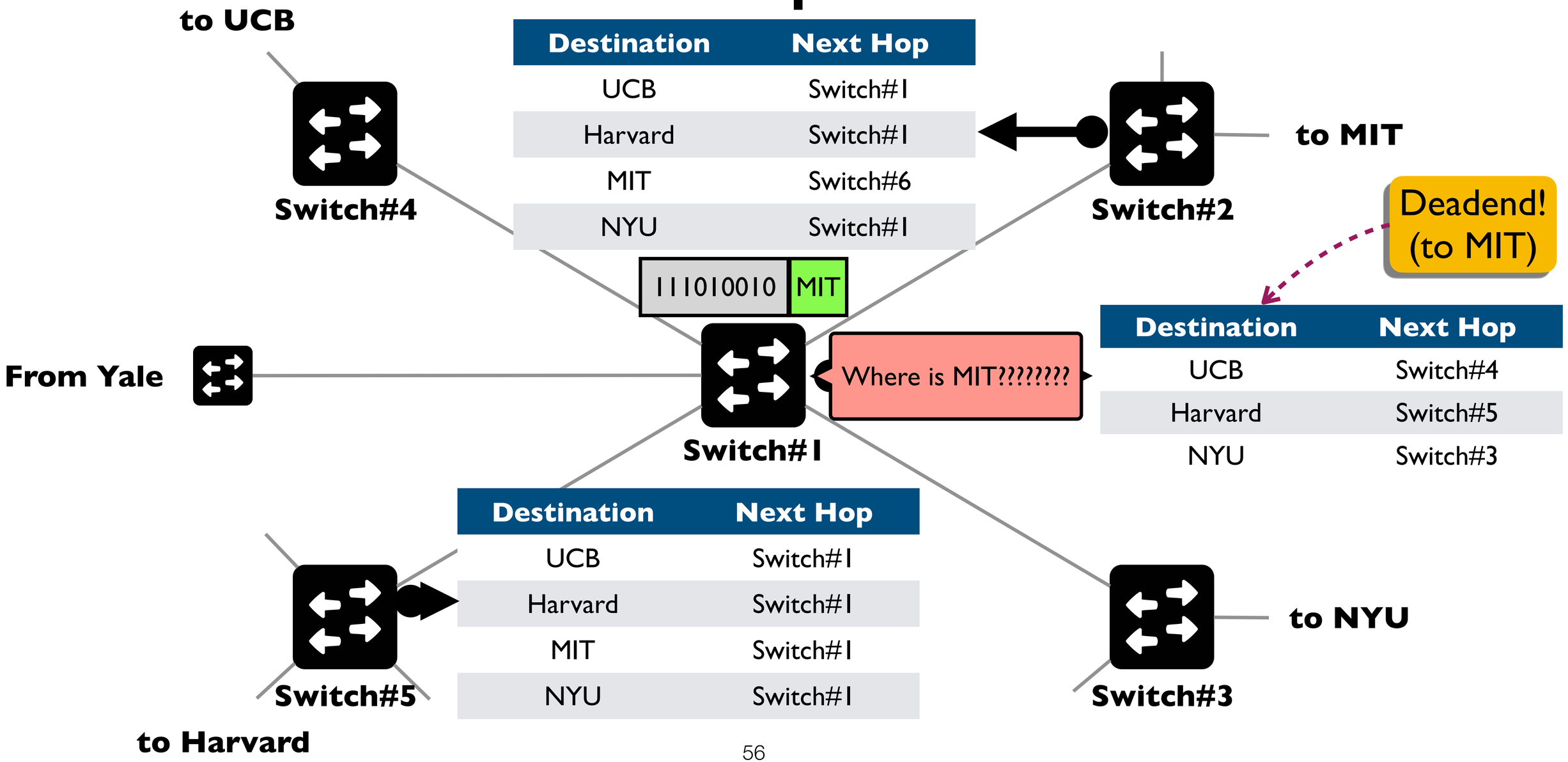


Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#6
NYU	Switch#1

Destination	Next Hop
UCB	Switch#4
Harvard	Switch#5
NYU	Switch#3

Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

Loops


to UCB
Switch#4
Destination
Next Hop
UCB
Switch#1
Harvard
Switch#1
MIT
Switch#6
NYU
Switch#1
111010010
MIT
Switch#2
to MIT
Deadend!
(to MIT)
From Yale
Switch#1
Where is MIT????????
Destination
Next Hop
UCB
Switch#4
Harvard
Switch#5
NYU
Switch#3
Destination
Next Hop
UCB
Switch#1
Harvard
Switch#1
MIT
Switch#1
NYU
Switch#1
Switch#5
to Harvard
Switch#3
to NYU

Necessary and Sufficient Condition

Necessary and Sufficient Condition

- Global routing state is "valid" ***if and only if:***
  - There are no dead-ends (other than the destination)
  - There are no loops

Necessary (“only if”): Easy

Necessary ("only if"): Easy

- If you run into a dead-end before reaching destination, you'll **never** reach the destination

Necessary (“only if”): Easy

- If you run into a dead-end before reaching destination, you’ll **never** reach the destination
- If you run into a loop before reaching destination, you’ll **never** reach the destination

Sufficient (“if”)

Sufficient (“if”)

- Assume no dead-ends, no loops

Sufficient (“if”)

- Assume no dead-ends, no loops
- Packets may keep wandering, but without repeating link
  - If ever a packet enters the same switch from the same link, there must be a loop

Sufficient ("if")

- **Assume no dead-ends, no loops**
- **Packets may keep wandering, but without repeating link**
  - If ever a packet enters the same switch from the same link, there must be a loop
- **Only a finite number of possible links for it to visit**
  - It cannot keep wandering forever without looping
  - Must eventually hit destination!

Checking Validity of Routing State

Checking Validity of Routing State

- **Focus only on a single destination**
  - Ignore all other routing state

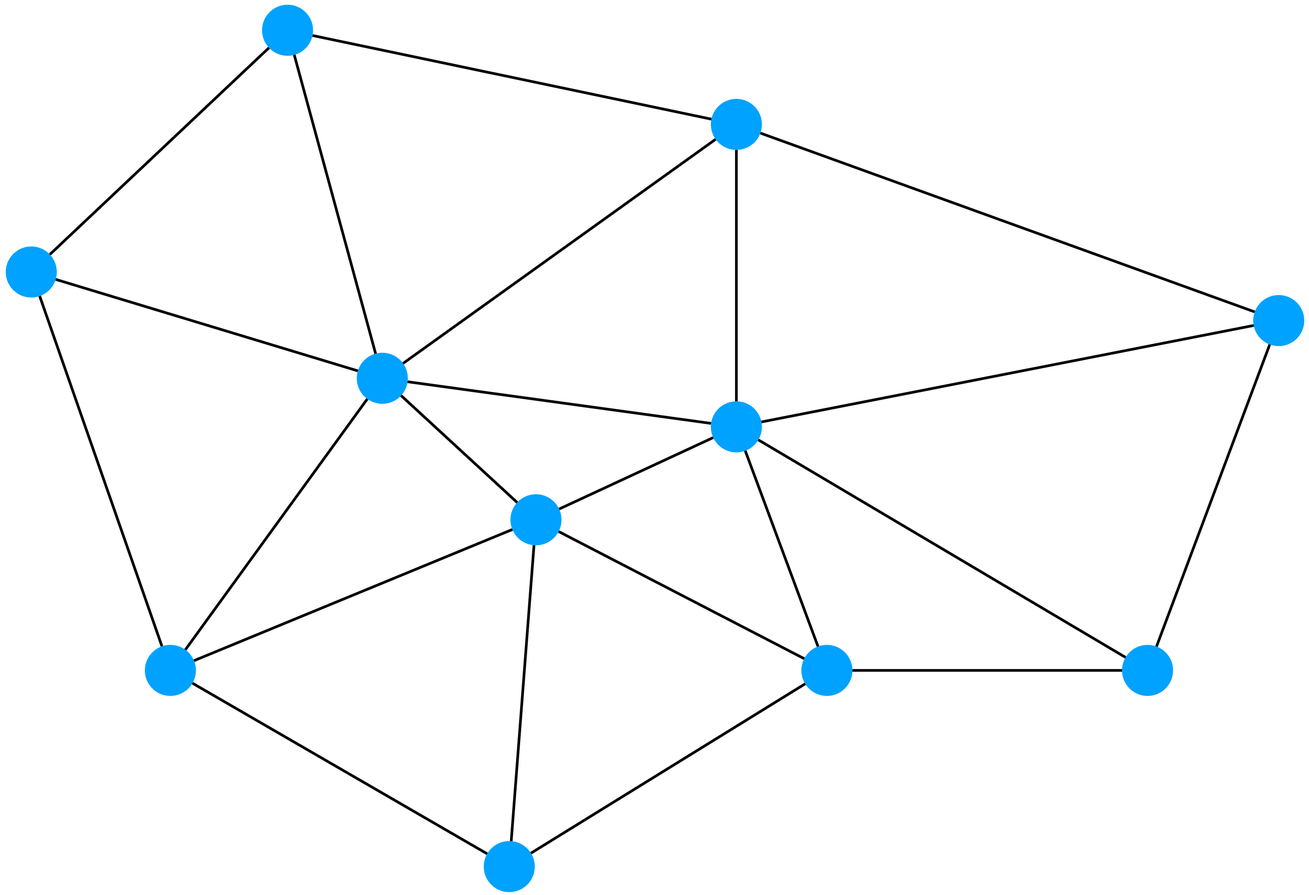
Checking Validity of Routing State

- **Focus only on a single destination**
  - Ignore all other routing state
- **Mark outgoing link to that destination ("next hop") with arrow**
  - There is only one at each node

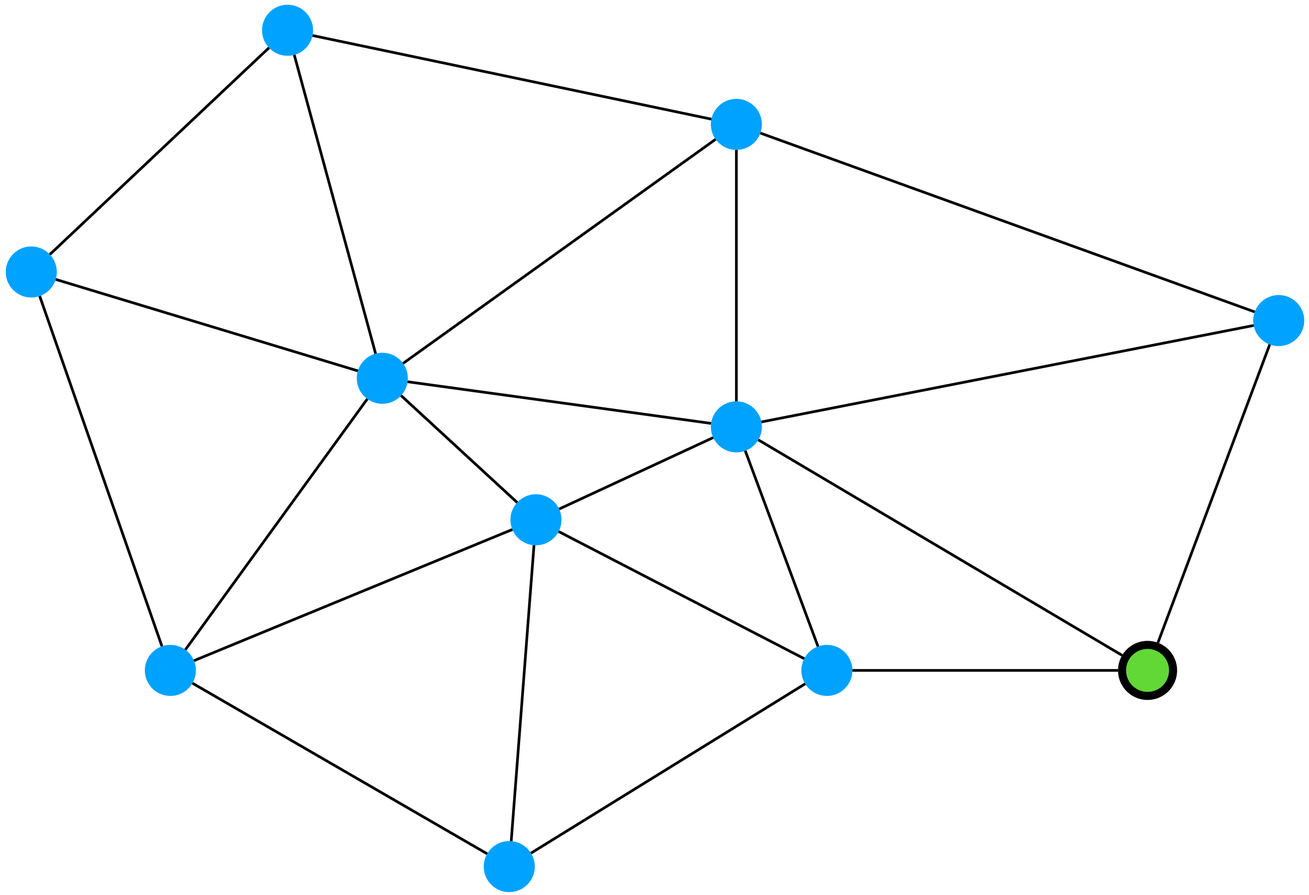
Checking Validity of Routing State

- **Focus only on a single destination**
  - Ignore all other routing state
- **Mark outgoing link to that destination ("next hop") with arrow**
  - There is only one at each node
- **Eliminate all links with no arrows**
  - Look at what's left…

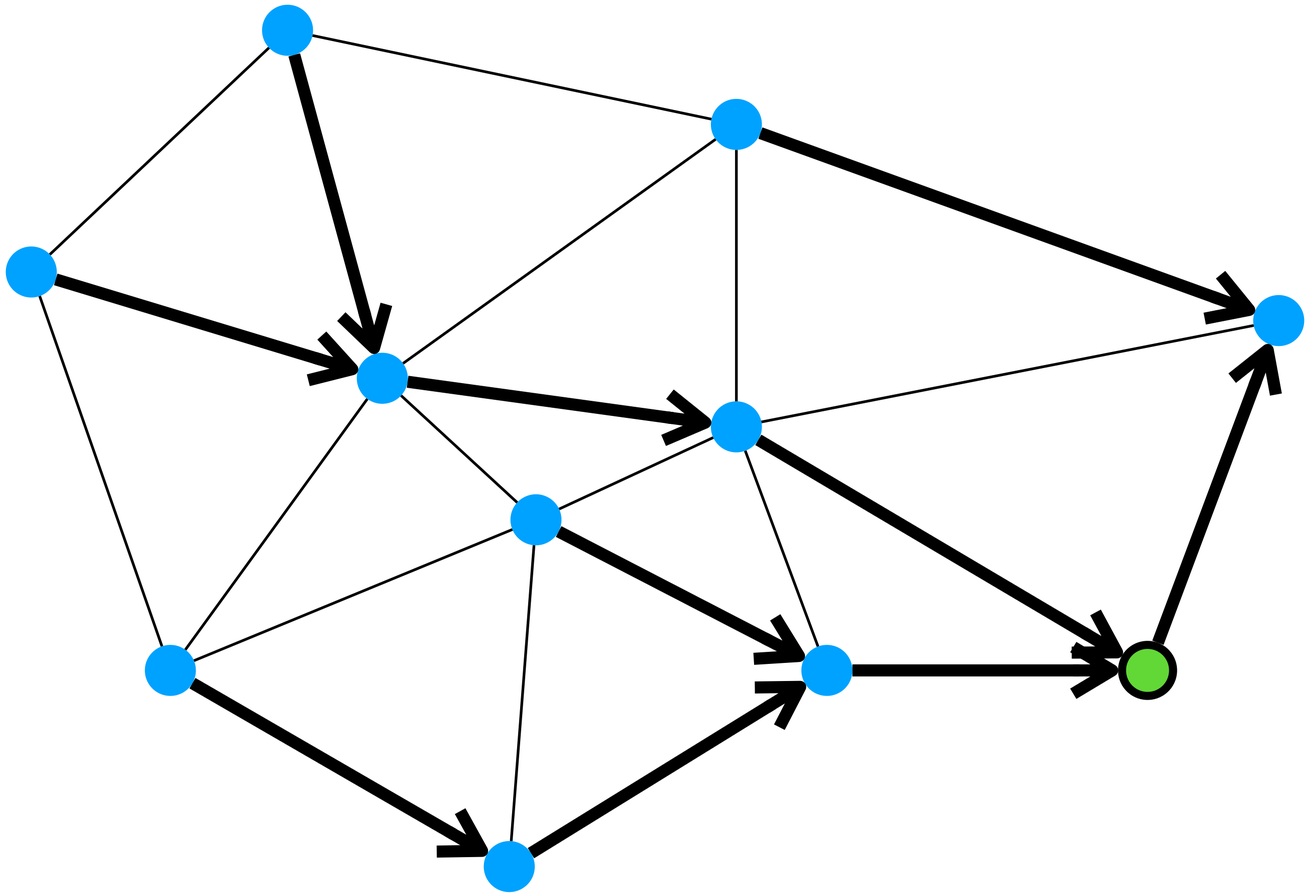
Example 1



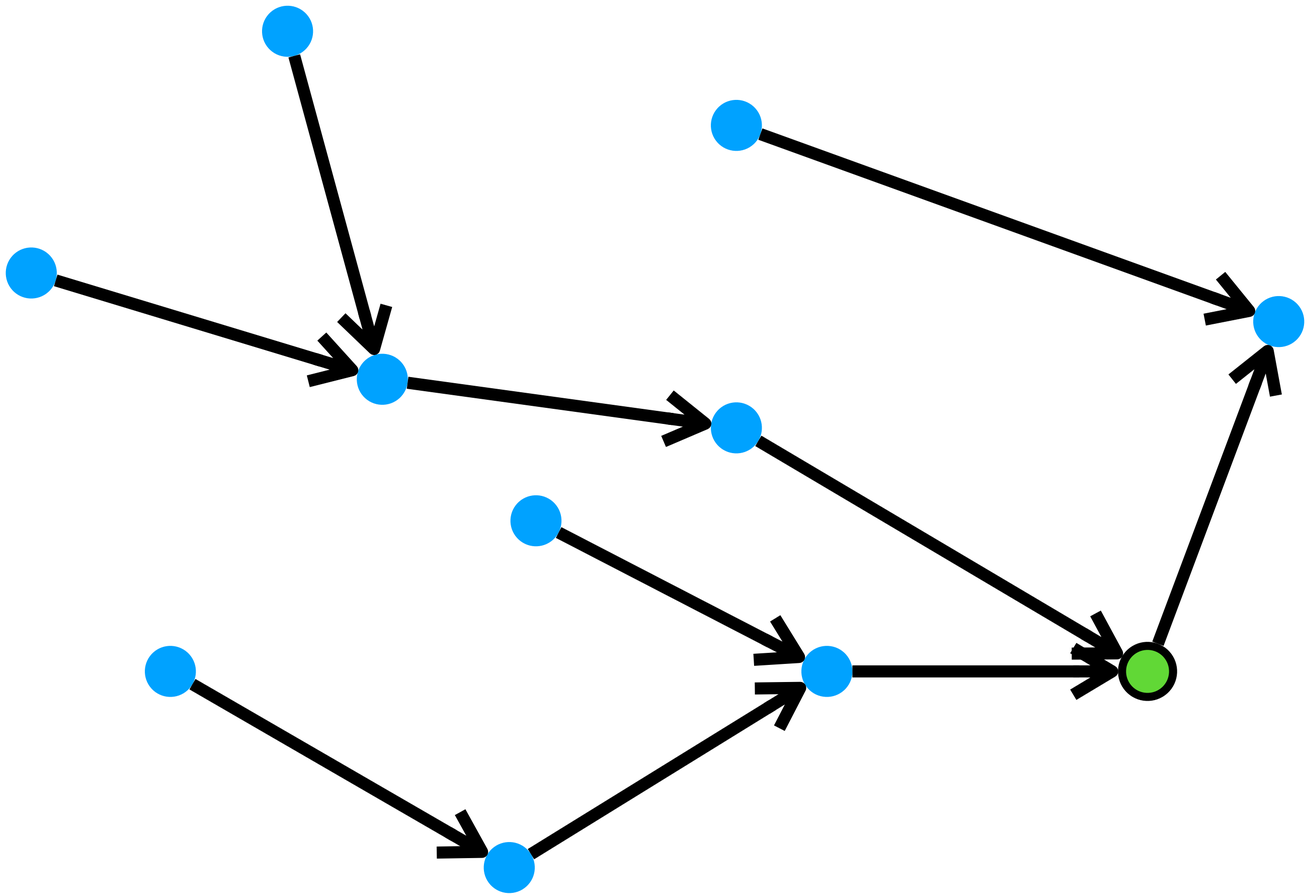
Pick Destination



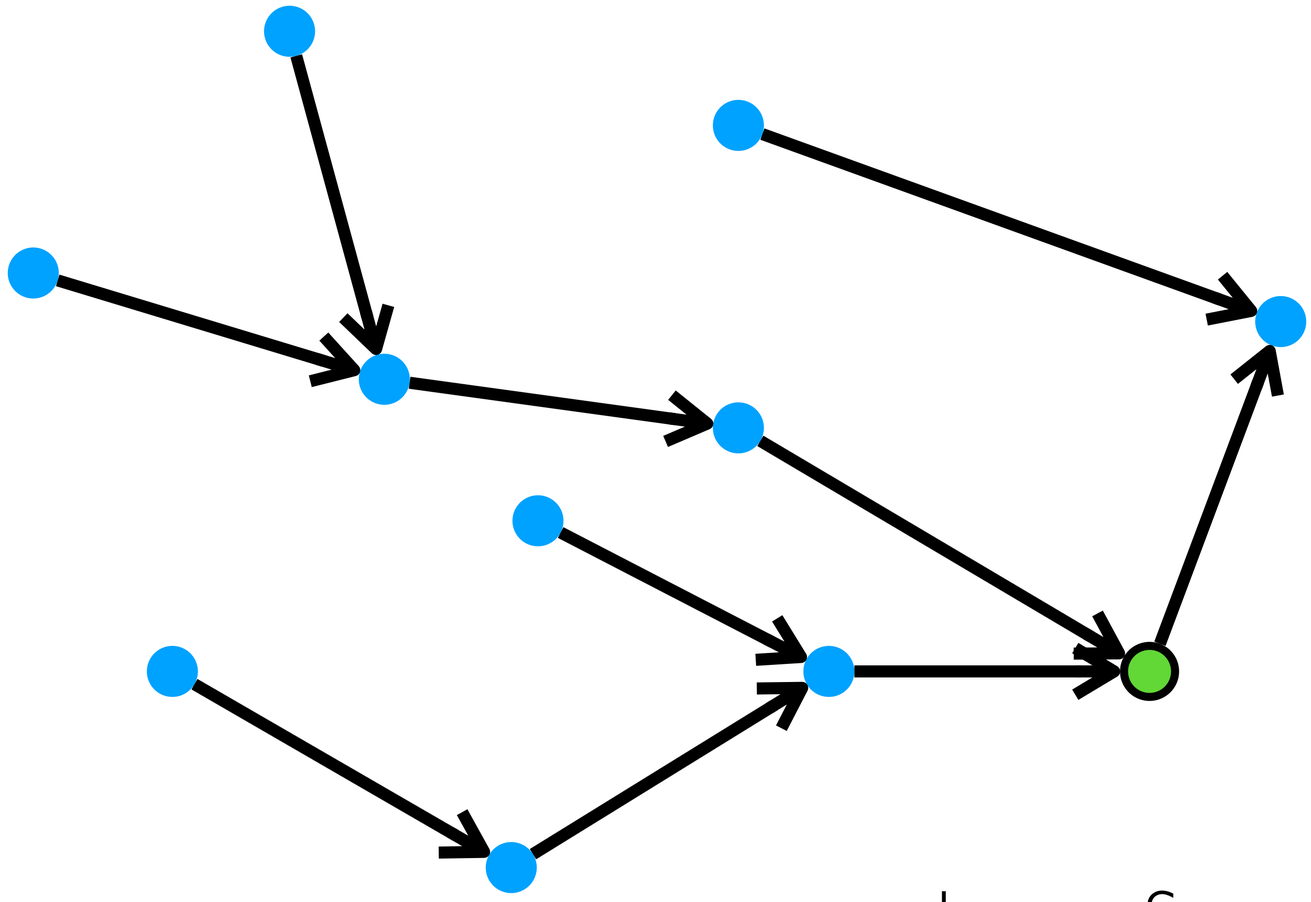
Put arrows on outgoing links (to green dot)



Remove Unused Links

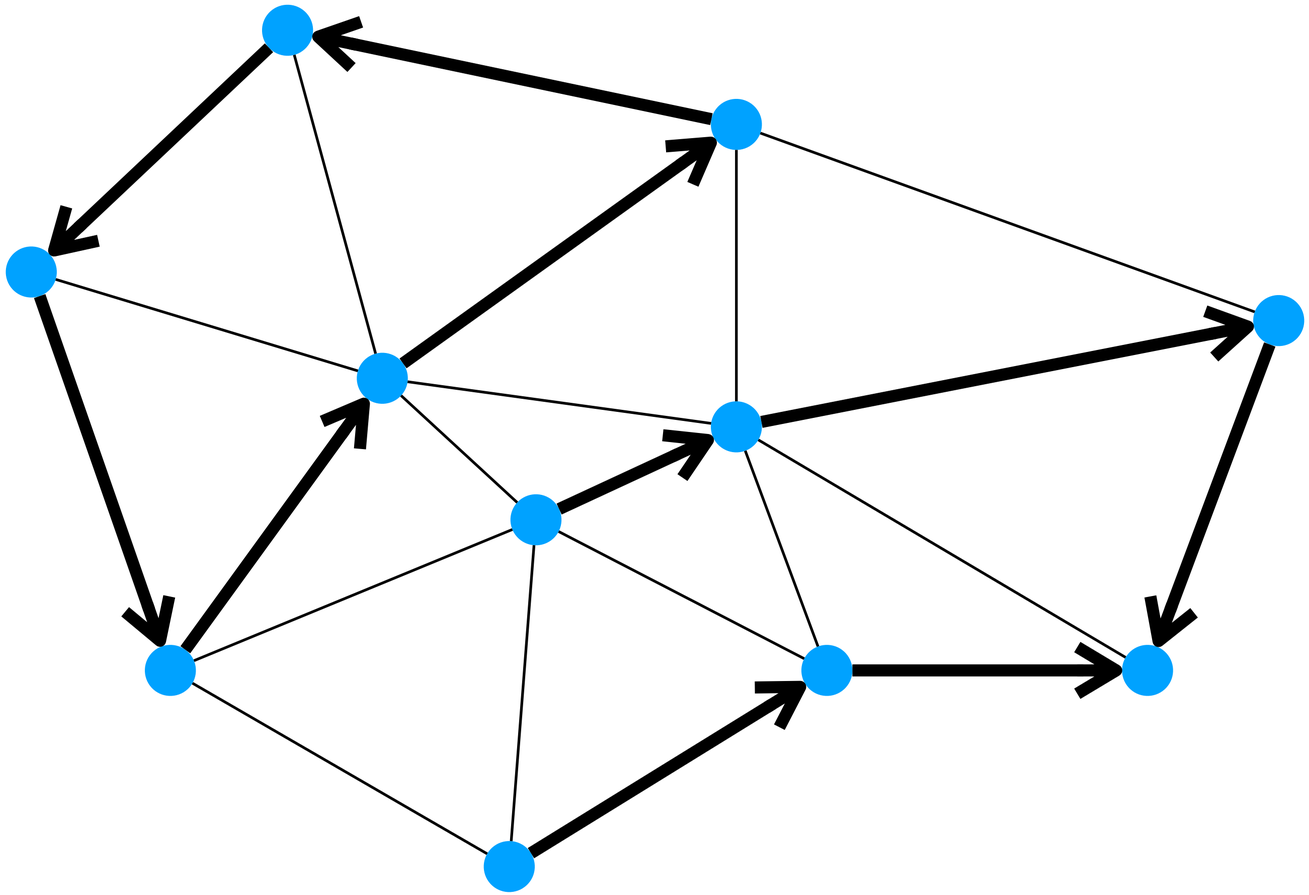


Remove Unused Links

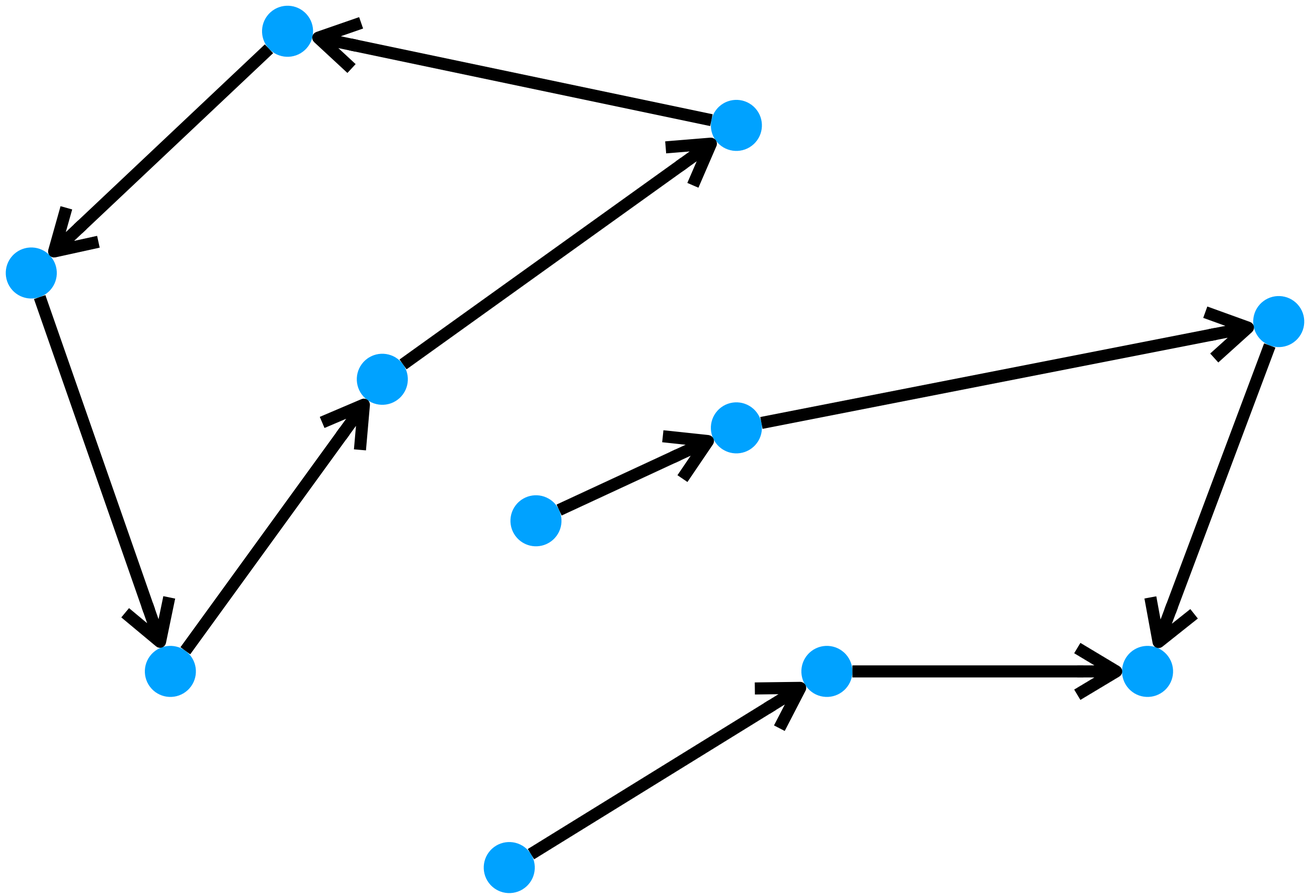


Leaves Spanning Tree: **Valid**

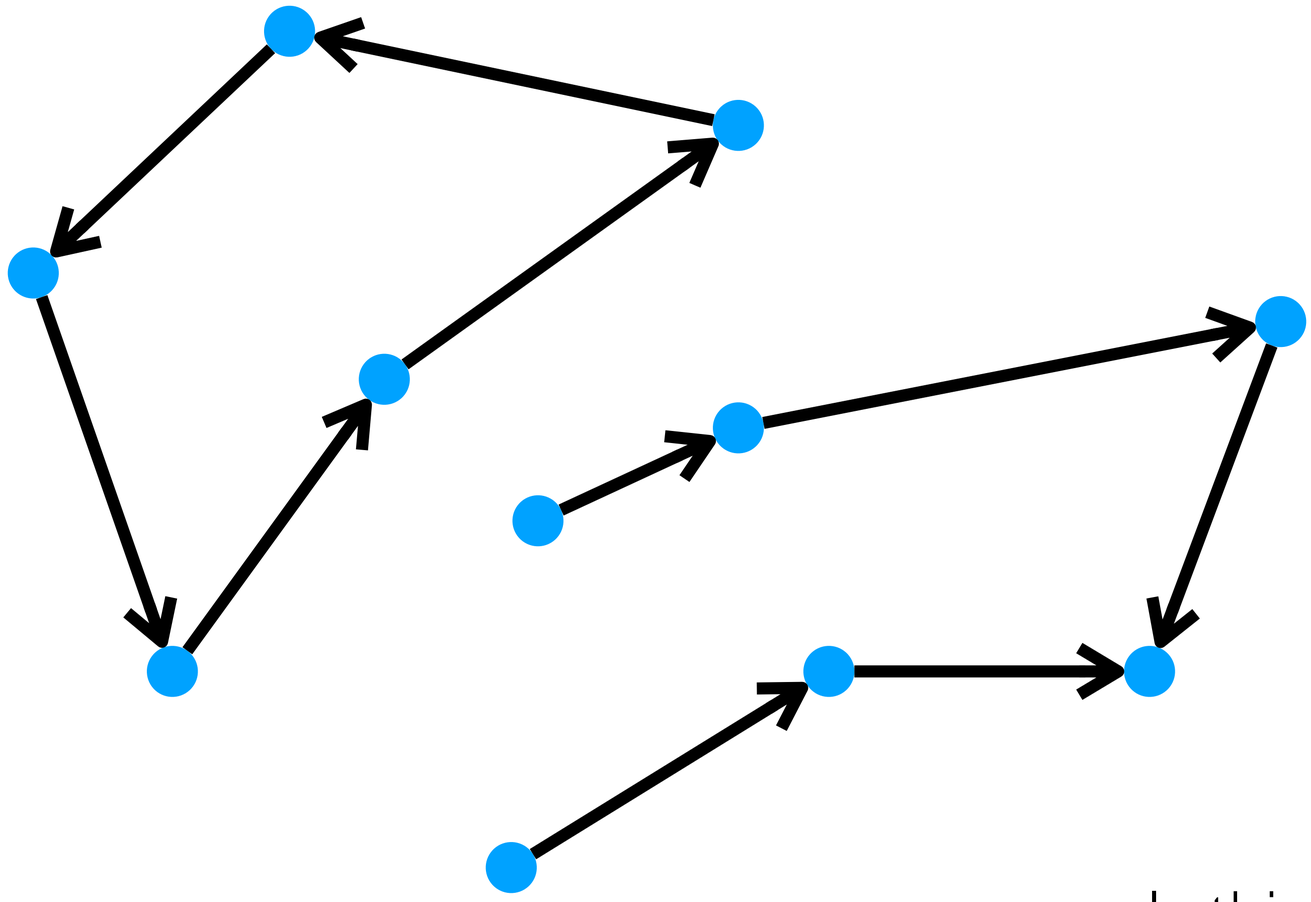
Example 2



Example 2



Example 2



Is this **Valid?**

Lesson?

- Very easy to check validity of routing state for a particular destination
- Dead-ends are obvious
  - Node without outgoing arrow
- Loops are obvious
  - Disconnected from the rest of the graph

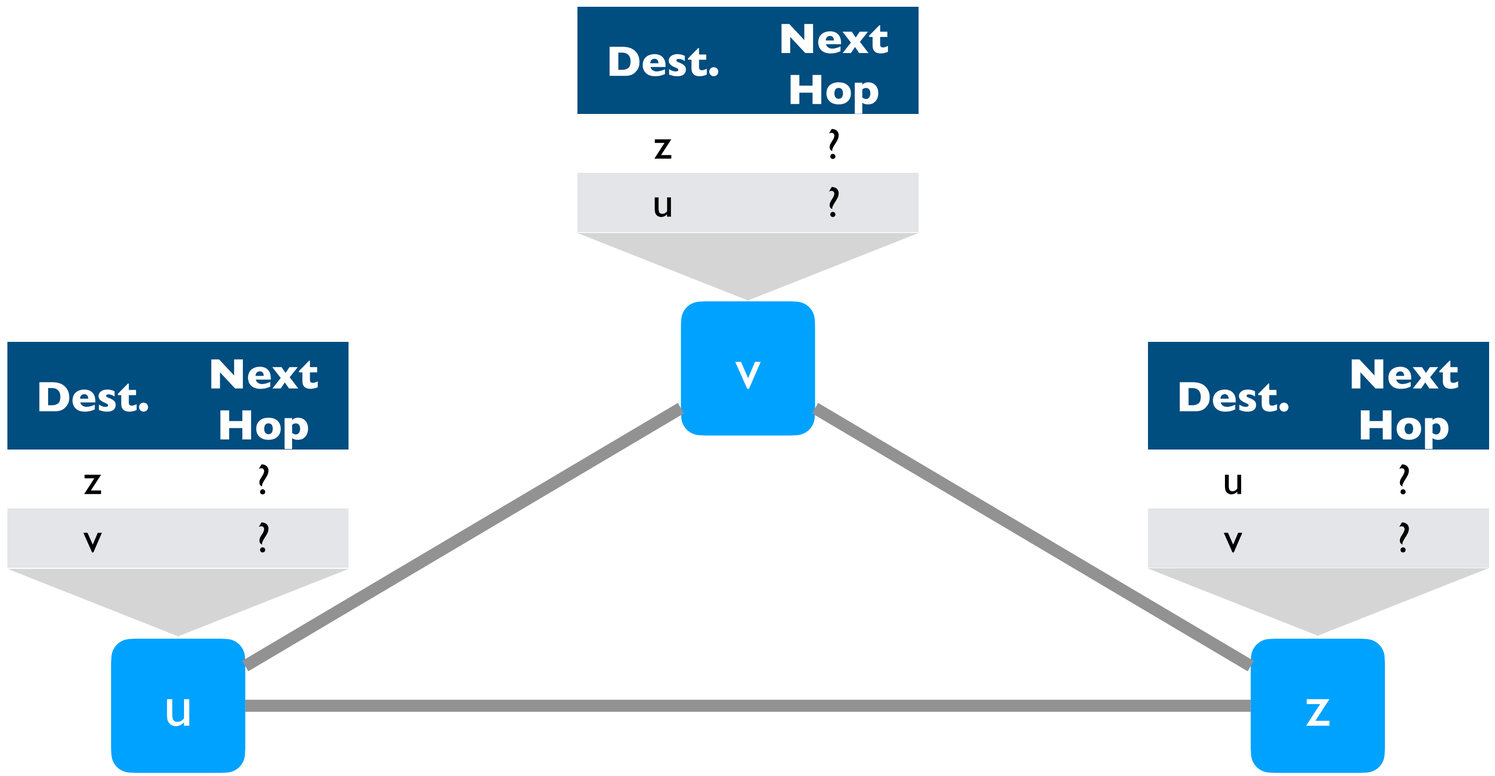
Goal (for routing)

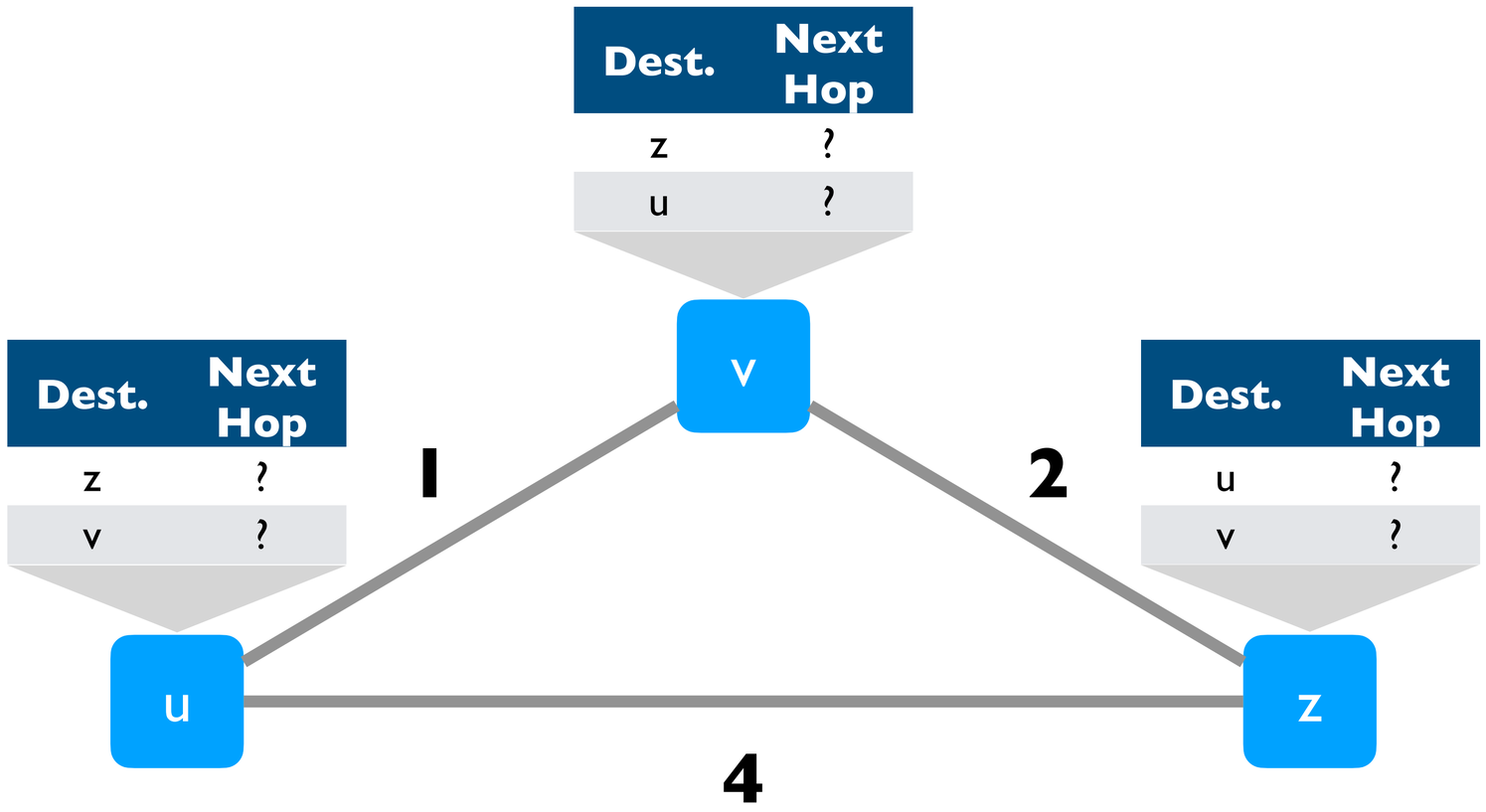
Goal (for routing)

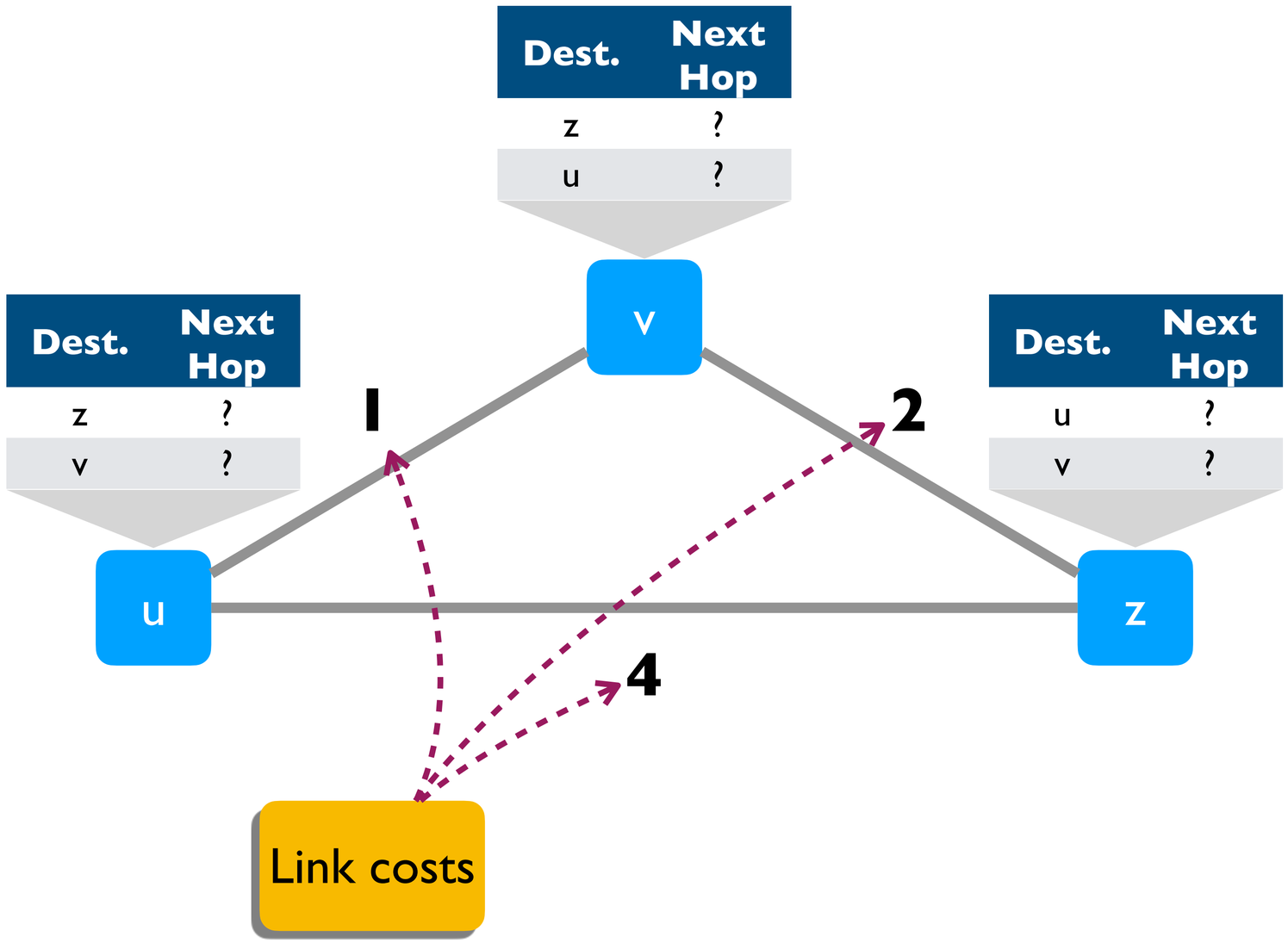
- **V1: Find a path to a given destination**

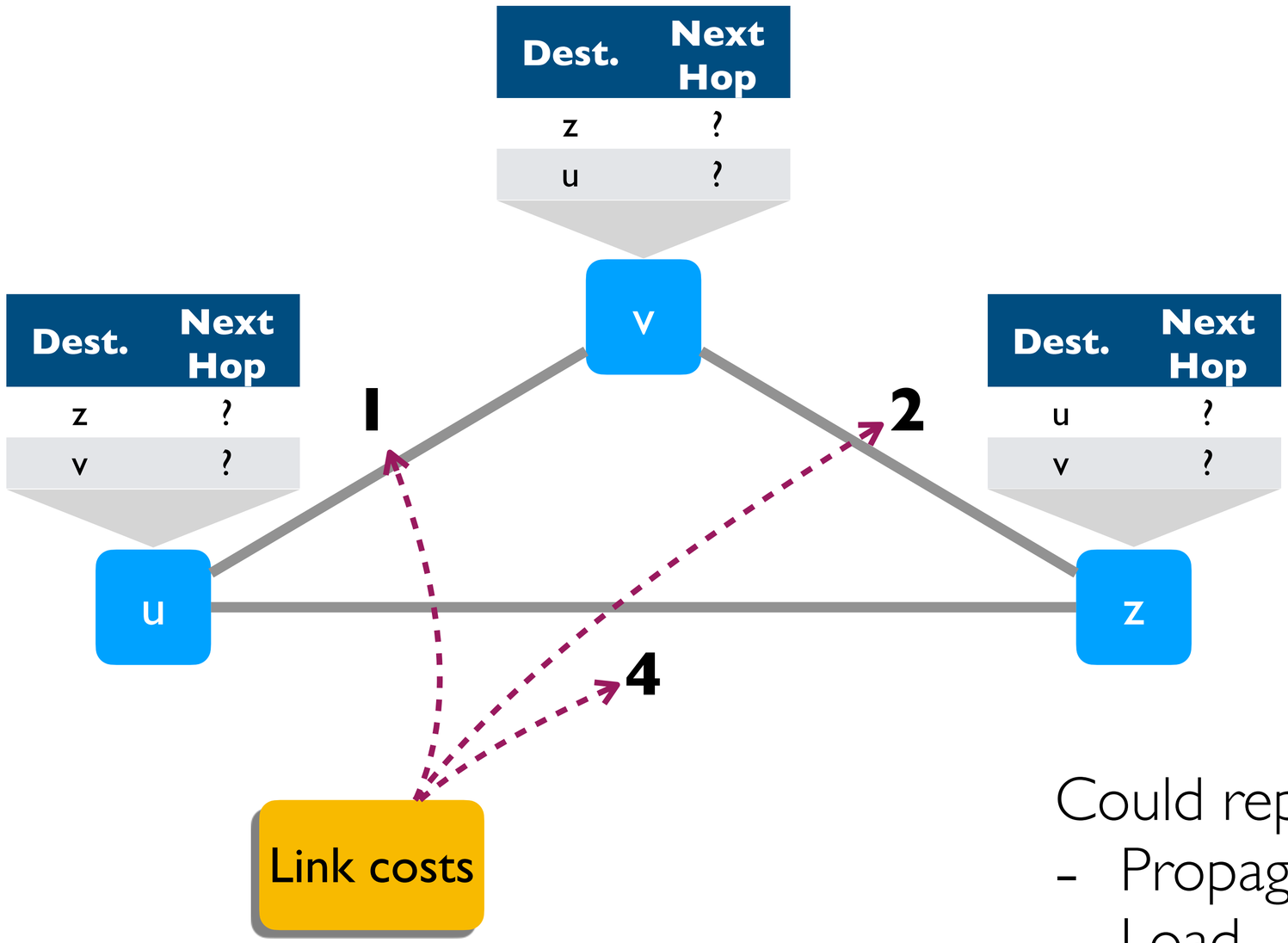
Goal (for routing)

- **V1: Find a path to a given destination**
- **V2: Find a *least cost path* to a given destination**


Dest.
Next Hop
z
?
u
?
v
Dest.
Next Hop
z
?
v
?
u
Dest.
Next Hop
u
?
v
?
u
z

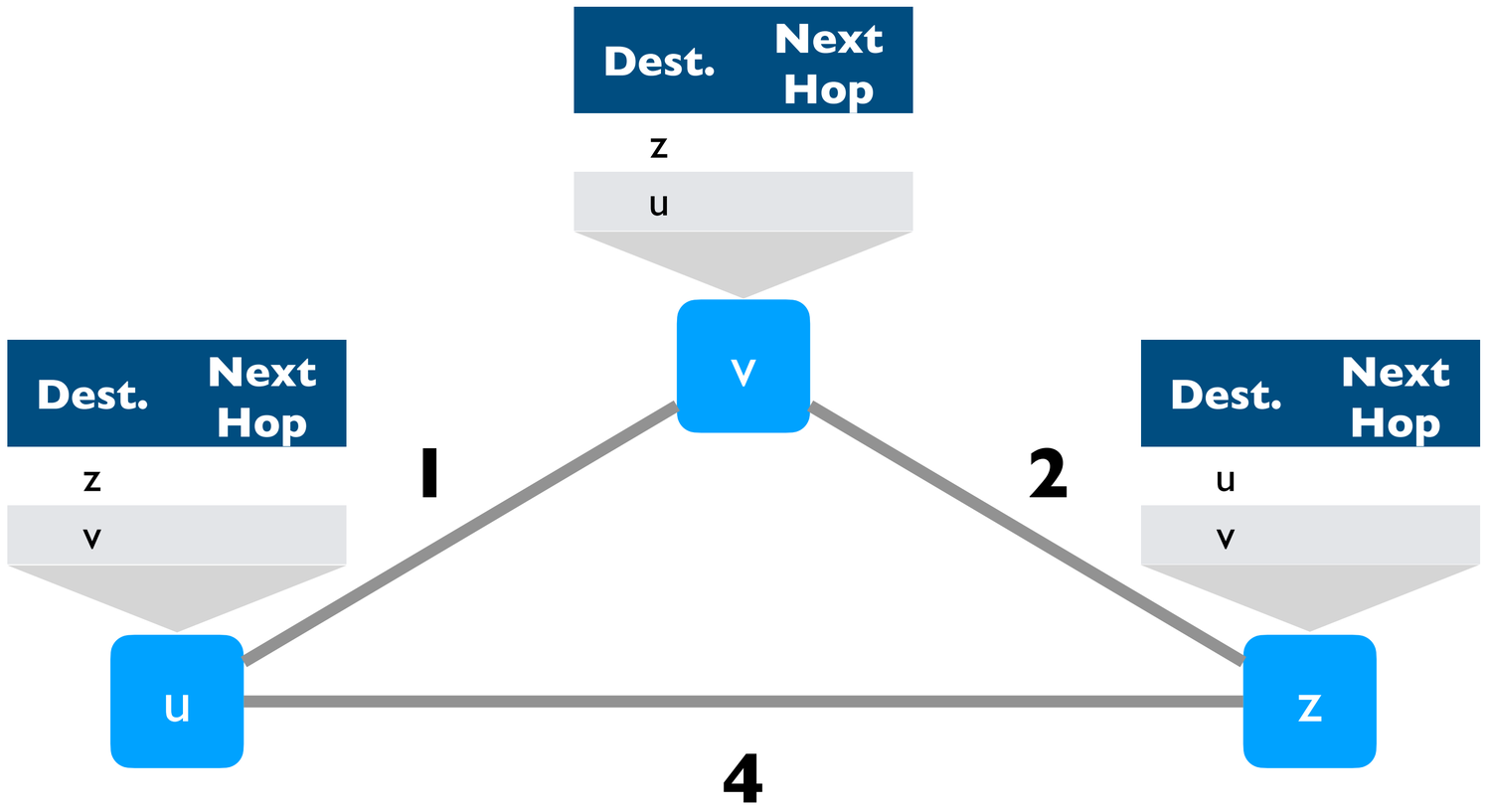

Dest.
Next Hop
z
?
u
?
v
Dest.
Next Hop
z
?
v
?
Dest.
Next Hop
u
?
v
?
1
2
4
u
v
z

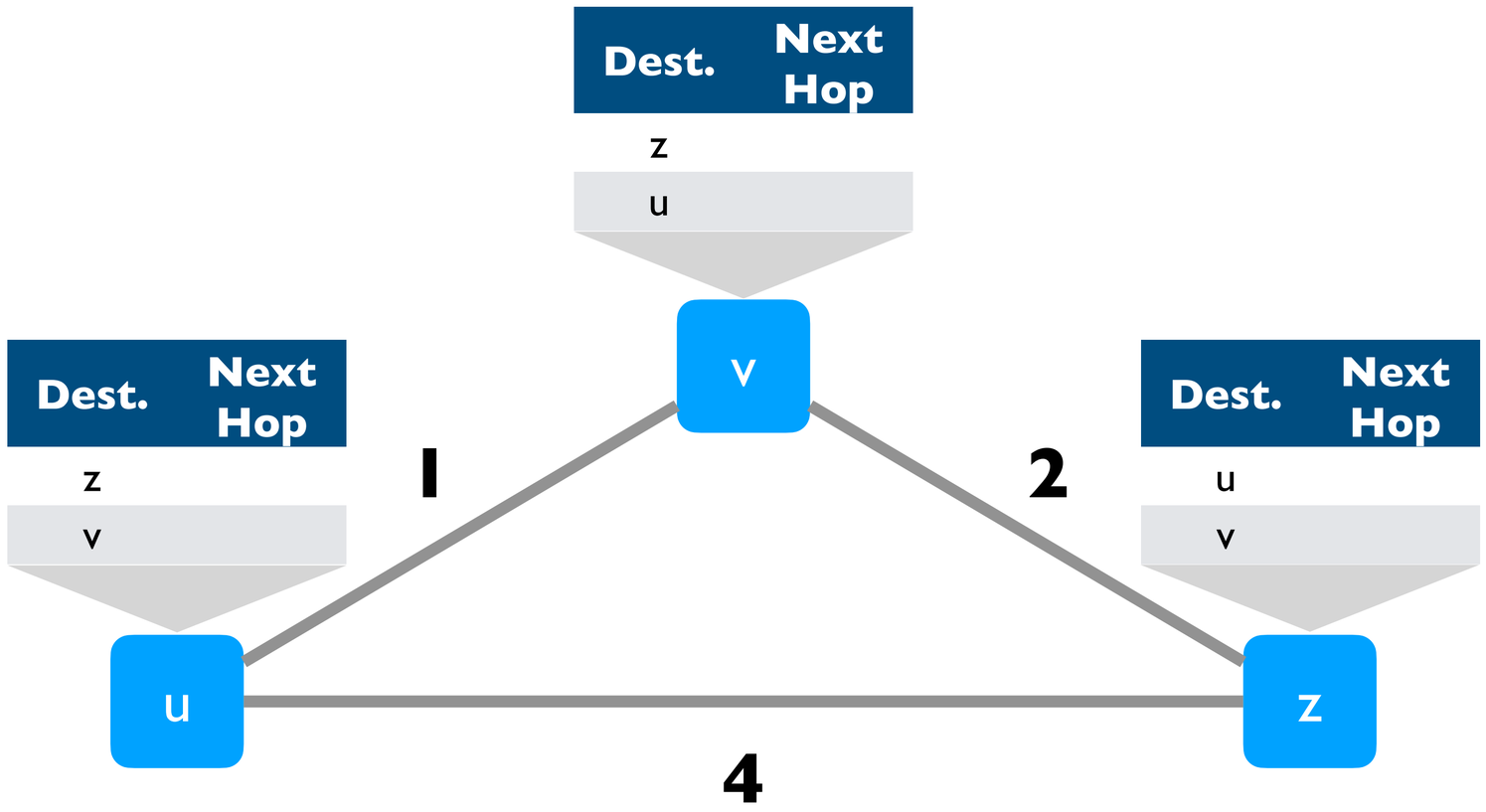

Dest.
Next Hop
z ?
u ?
v
Dest.
Next Hop
z ?
v ?
u
Dest.
Next Hop
u ?
v ?
z
1
2
4
u
z
Link costs



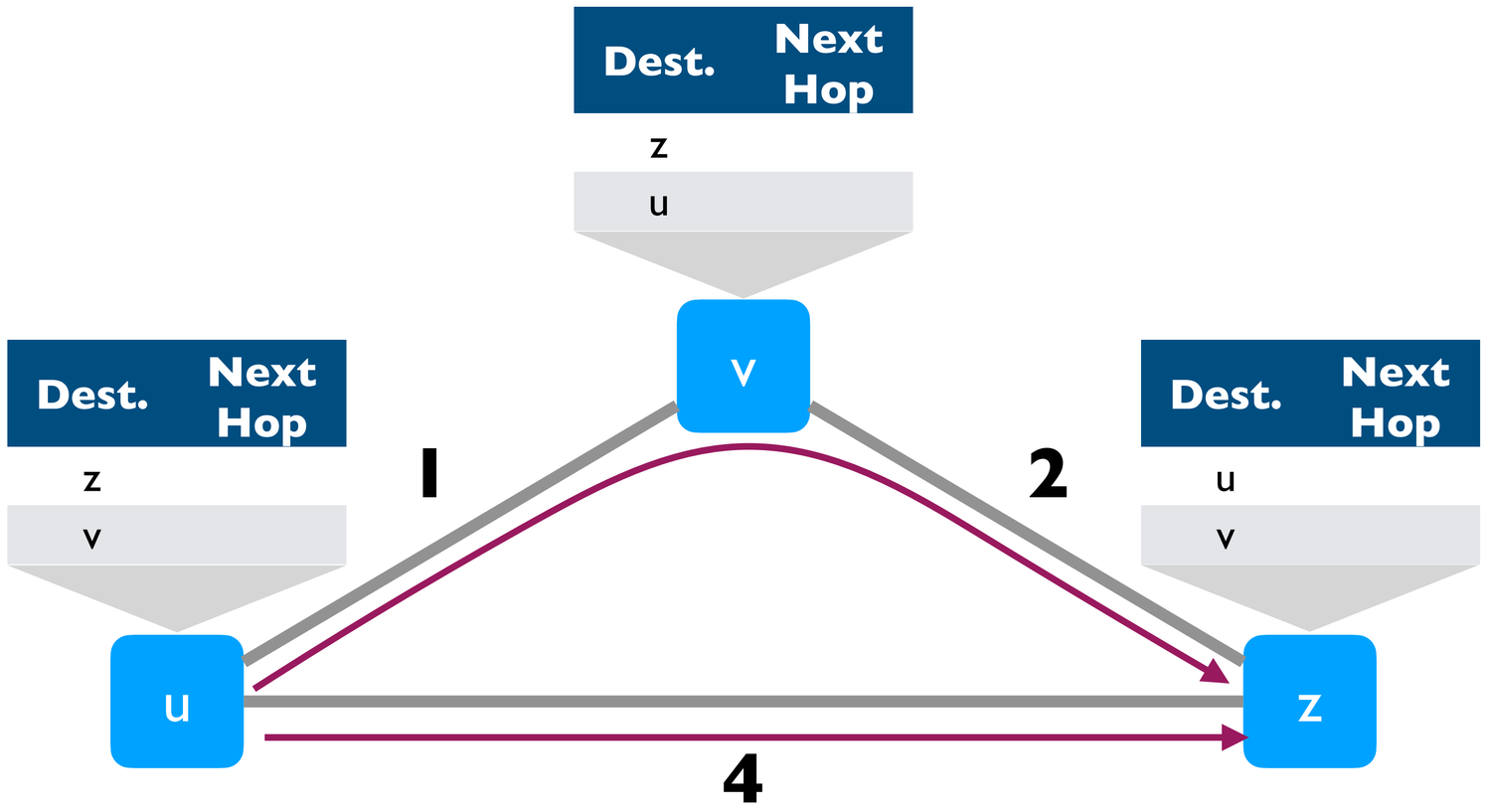
Could represent:

- Propagation delay
- Load
- Cost

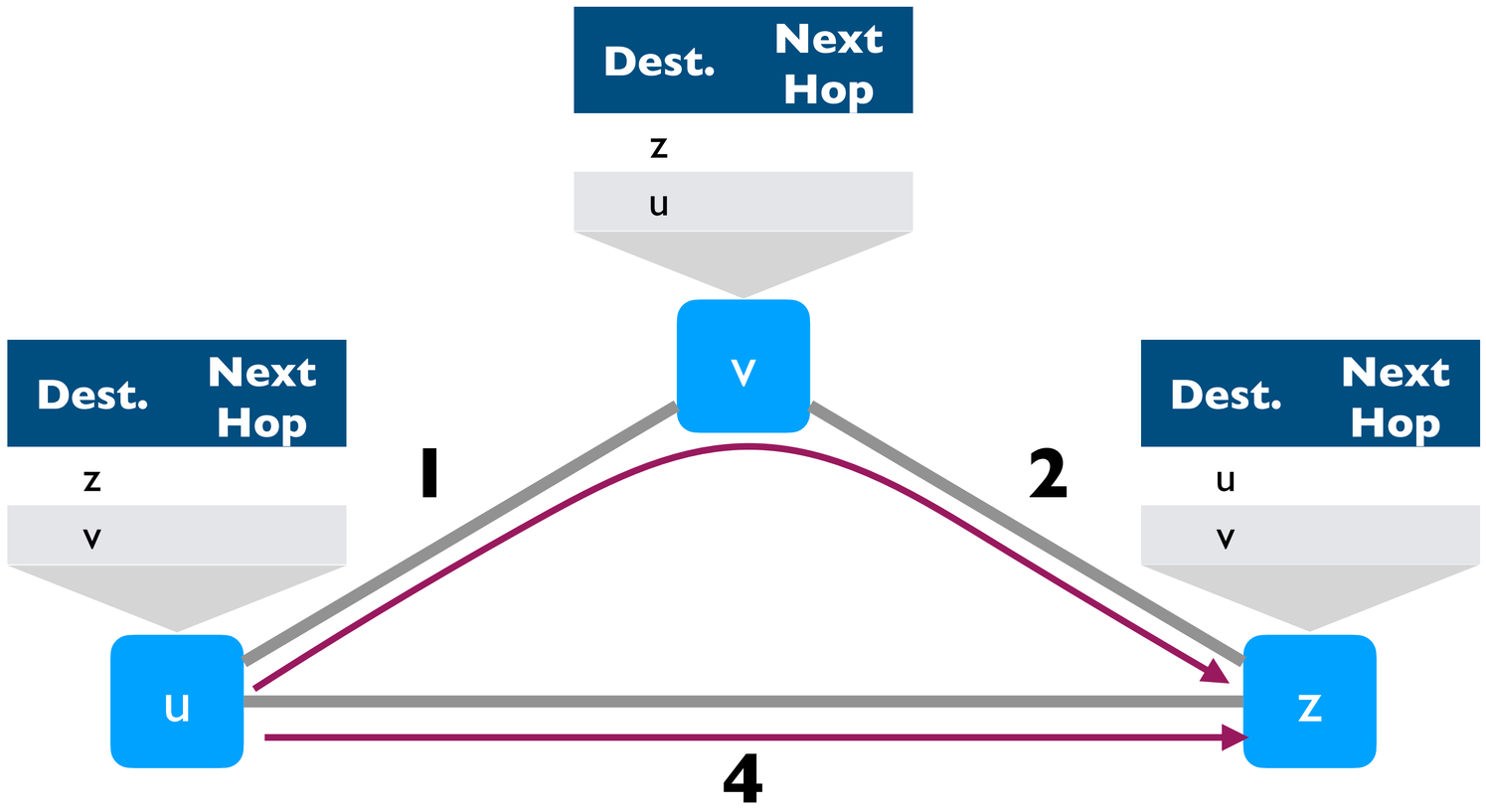

Dest.
Next Hop
z
u
v
Dest.
Next Hop
z
v
Dest.
Next Hop
u
v
1
2
4
u
v
z



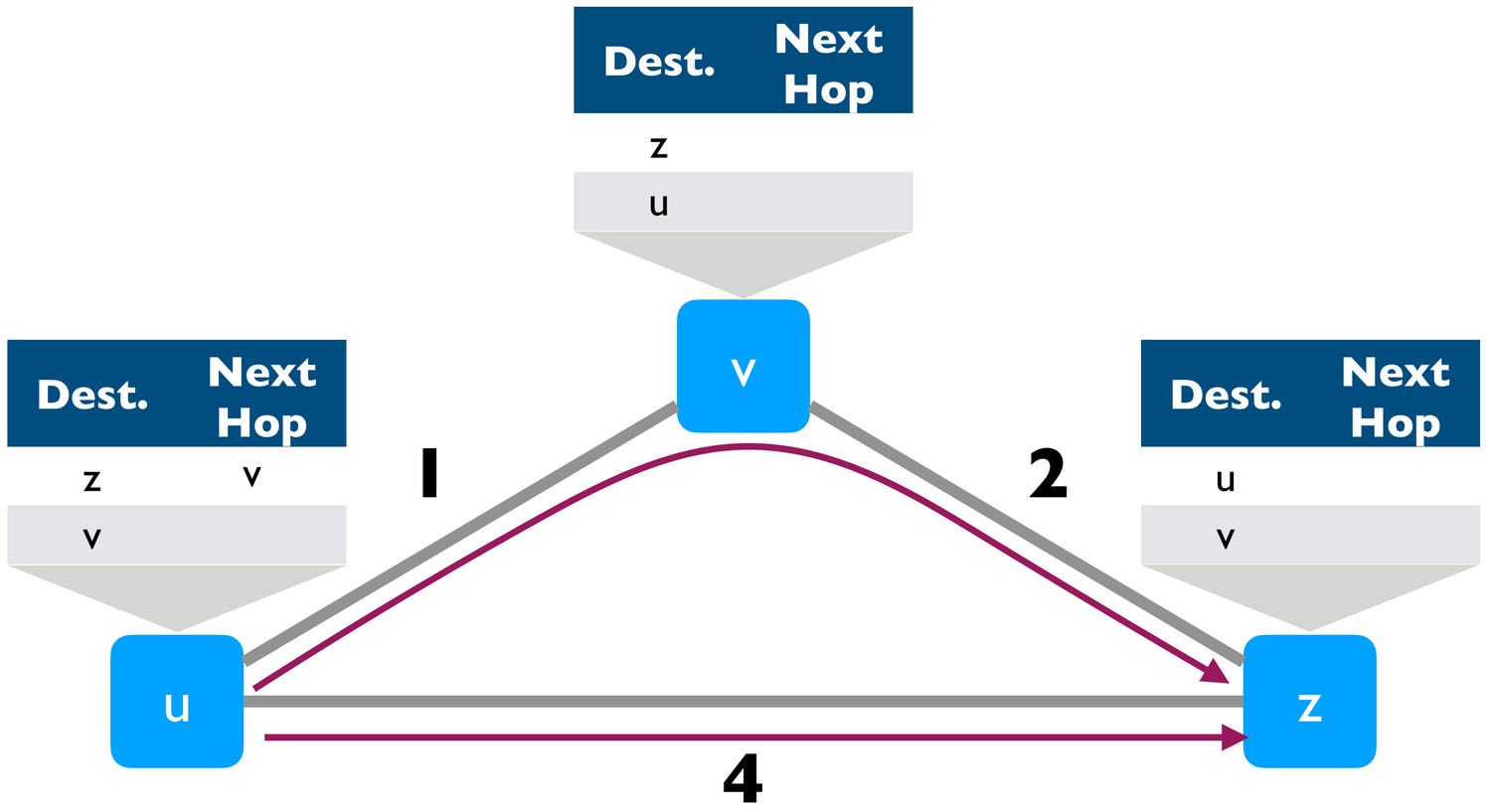
Least-cost path from **u** to **z**:



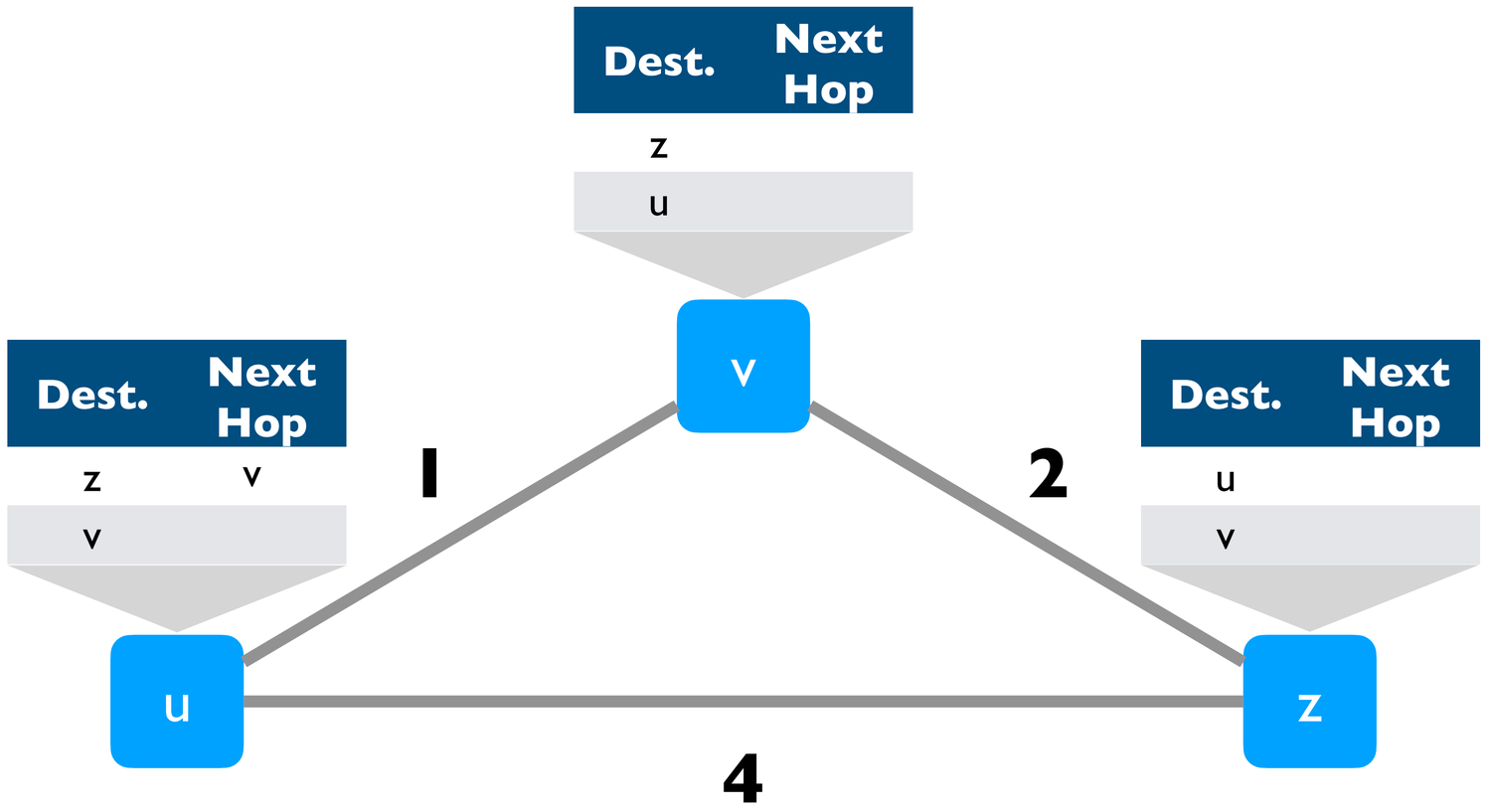
Least-cost path from **u** to **z**:



Least-cost path from **u** to **z**: **u v z**

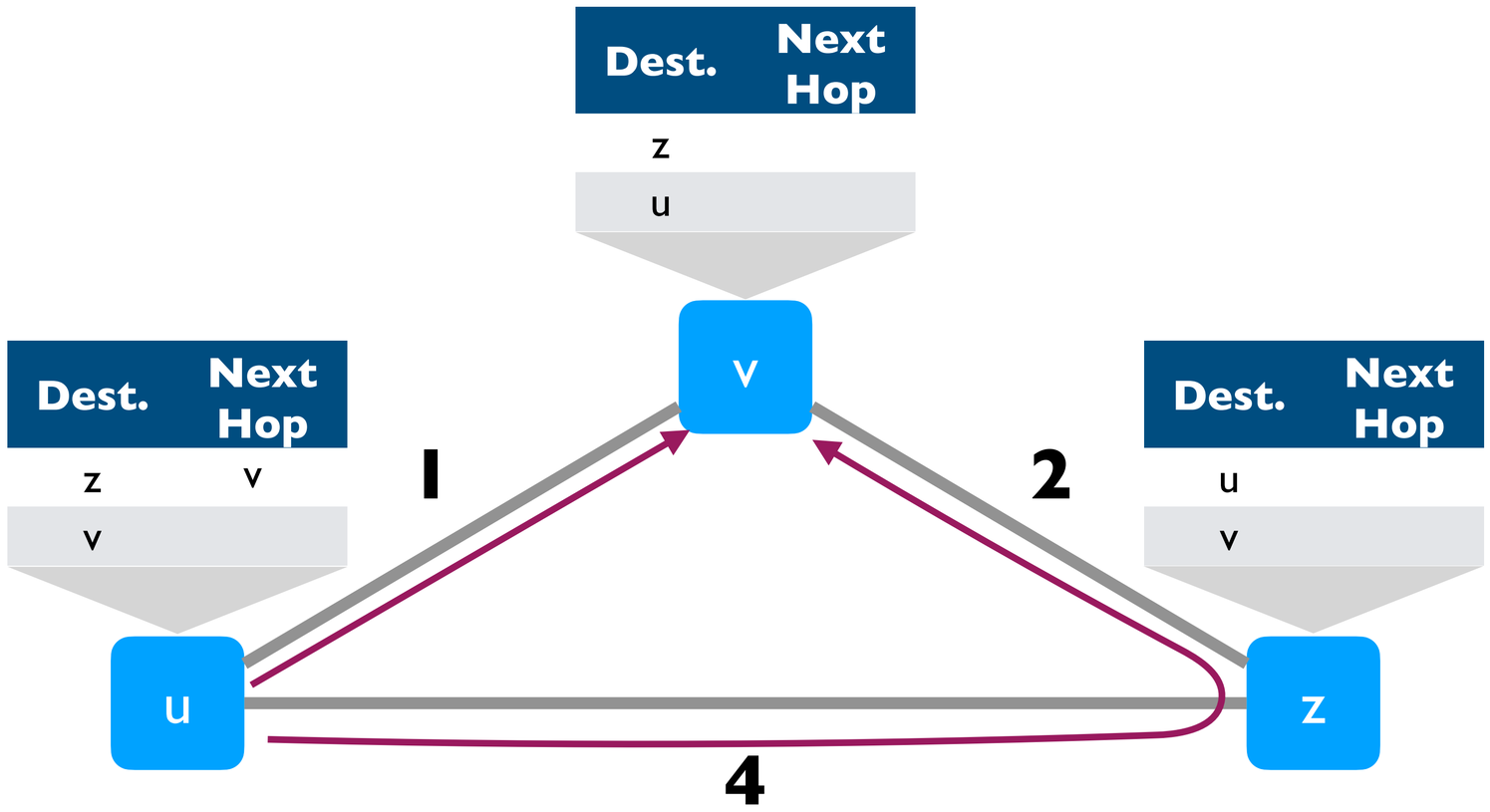


Least-cost path from **u** to **z**: **u v z**



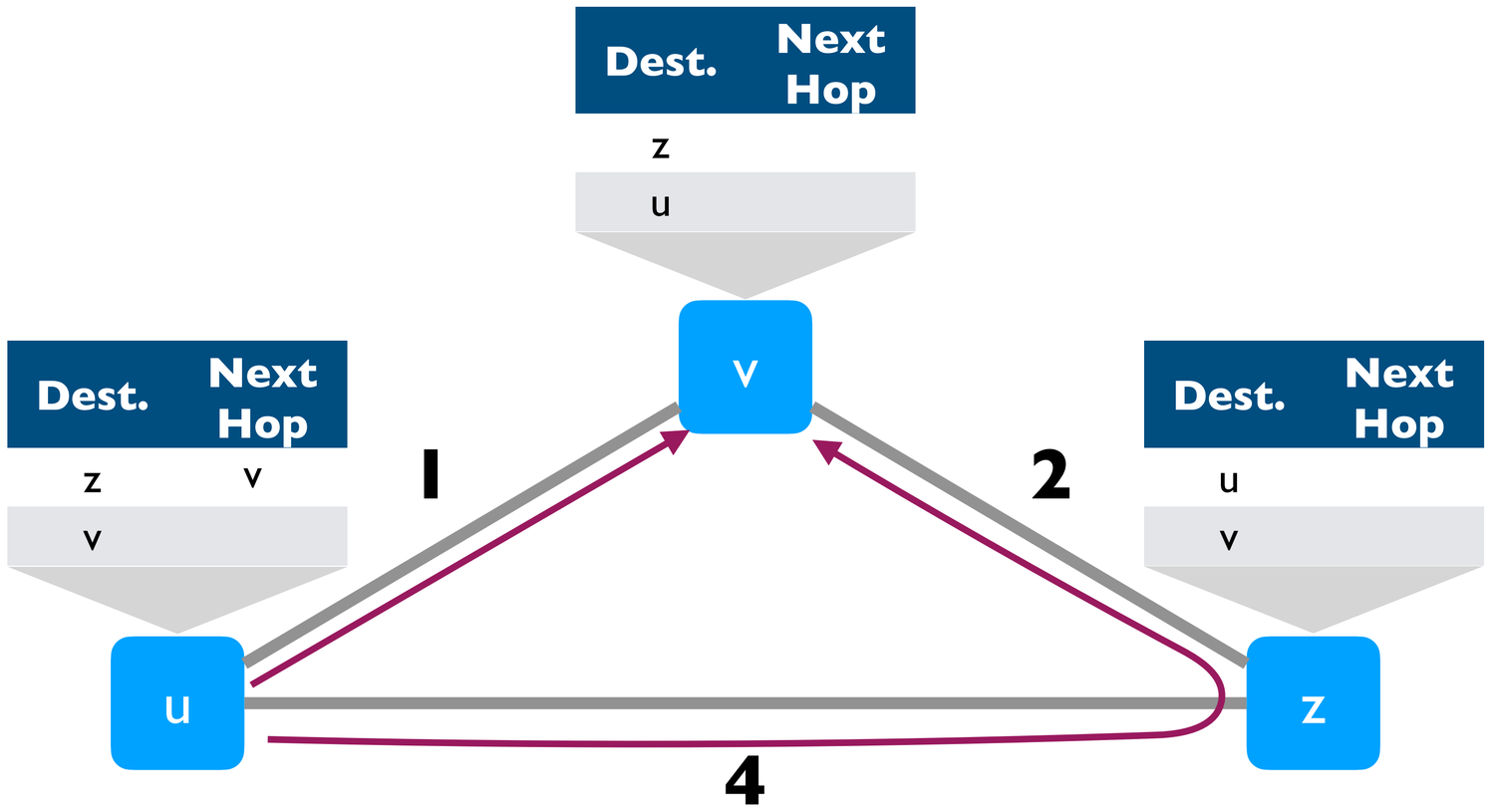
Least-cost path from **u** to **z**: **u v z**

Least-cost path from **u** to **v**:



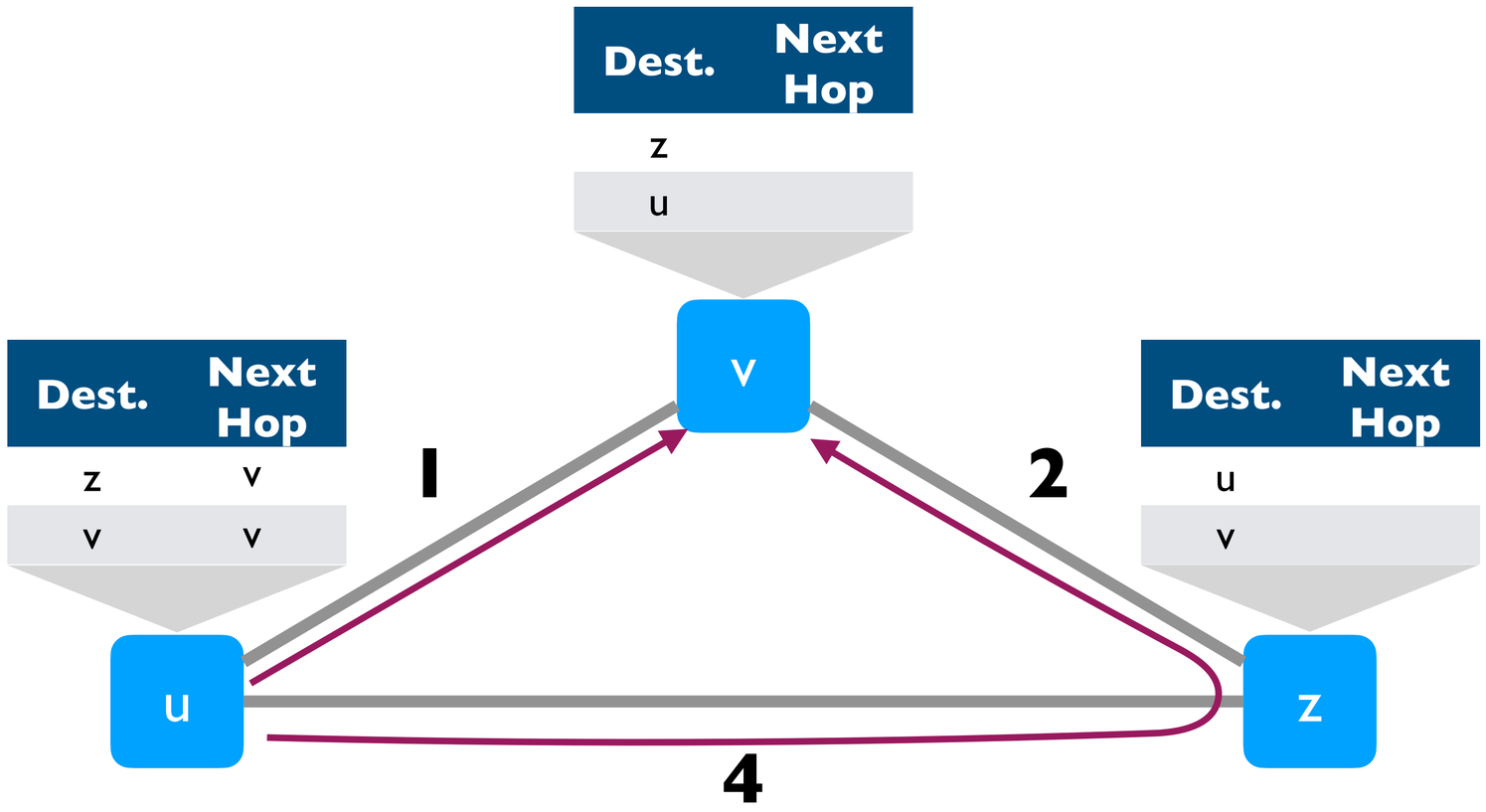
Least-cost path from **u** to **z**: **u v z**

Least-cost path from **u** to **v**:



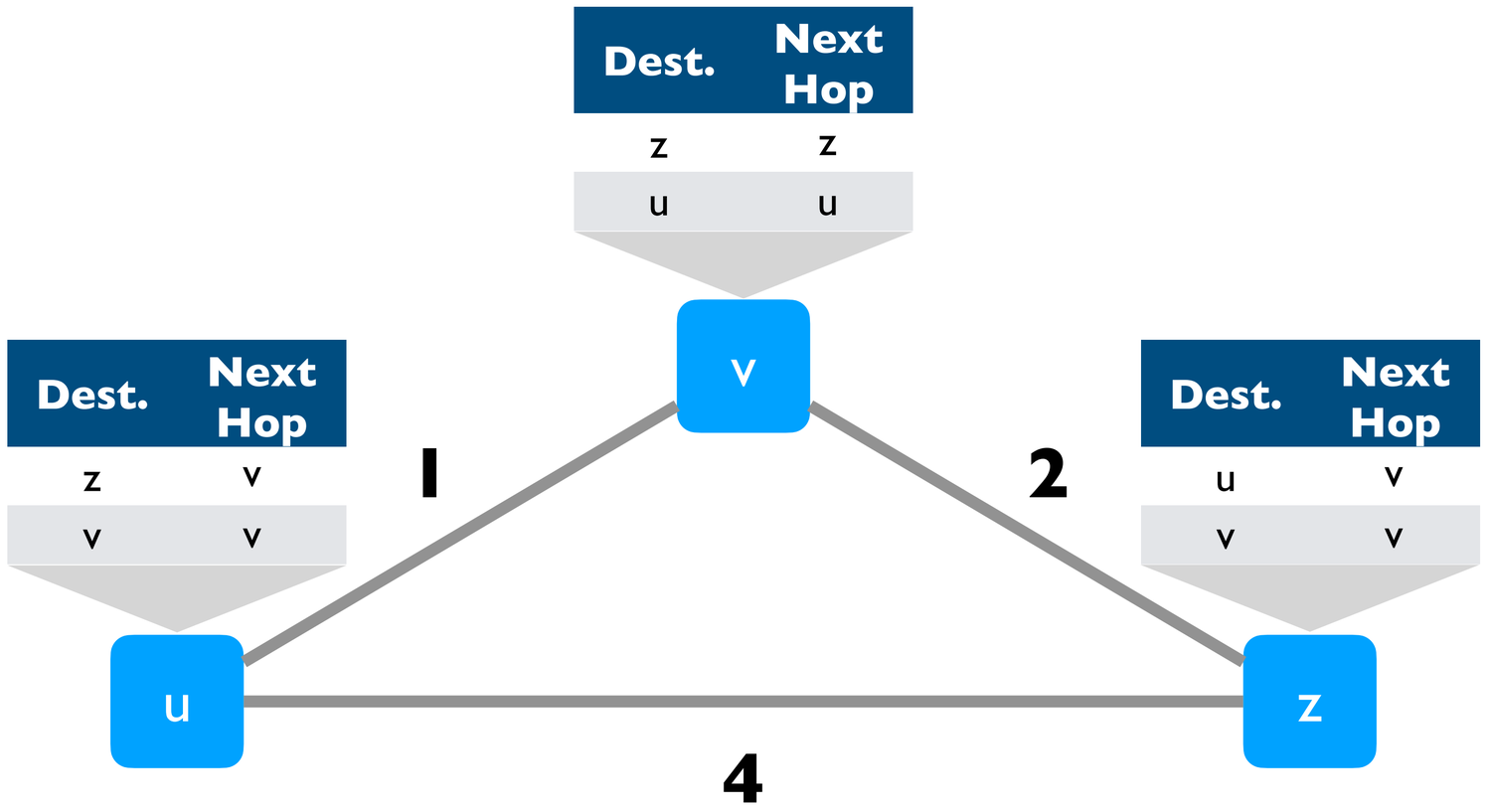
Least-cost path from **u** to **z**: **u v z**

Least-cost path from **u** to **v**: **u v**



Least-cost path from **u** to **z**: **u v z**

Least-cost path from **u** to **v**: **u v**



Least-cost path from **u** to **z**: **u v z**

Least-cost path from **u** to **v**: **u v**

Least-cost path routing

Least-cost path routing

- **Given:** Router graph & link costs

Least-cost path routing

- **Given:** Router graph & link costs

- **Goal:** find least-cost path
  from each source router
  to each destination router

Least-cost path routing

- **Given:** Router graph & link costs
- **Goal:** find least-cost path
  from each source router
  to each destination router
- **Next time on CPSC 433/533:** How we compute these!