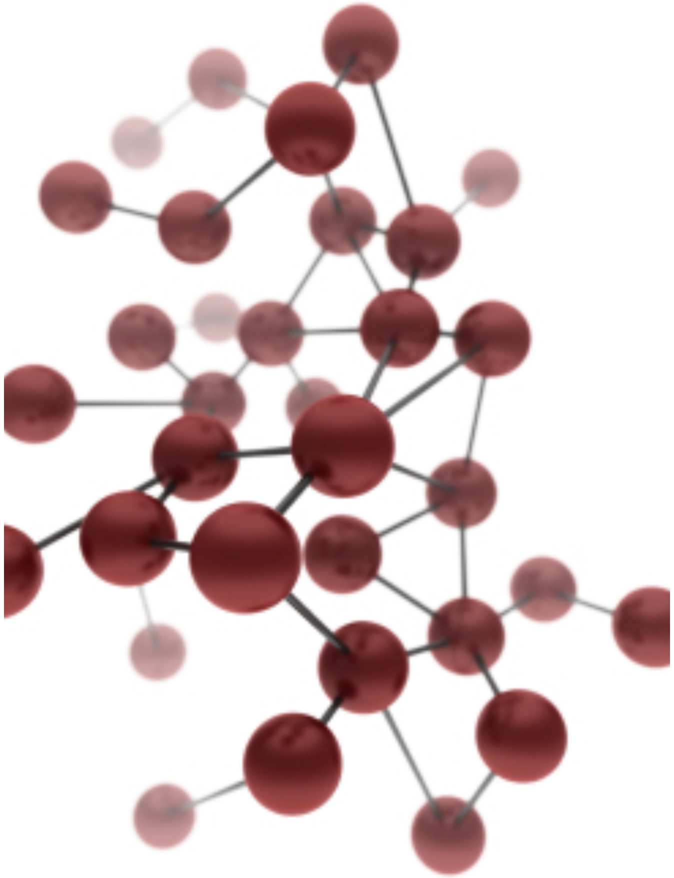
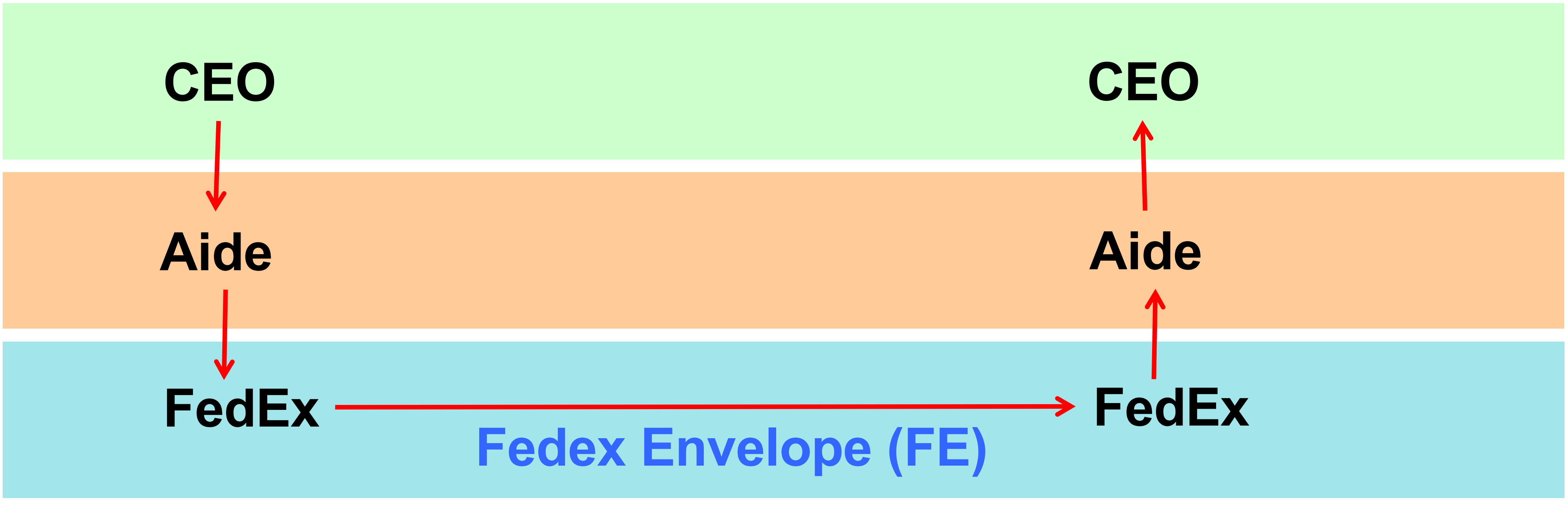


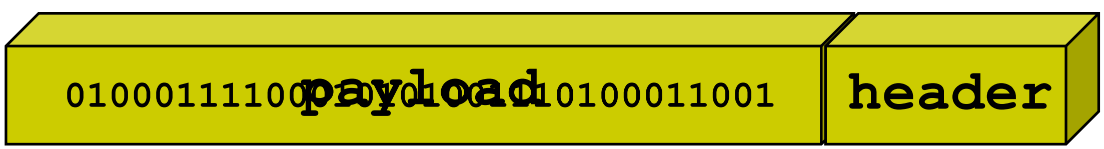
IP Routing Algorithms
Lecture 4


CEO
CEO
Aide
Aide
FedEx
FedEx
Fedex Envelope (FE)

What is a Protocol?

- An agreement between parties on how to communicate

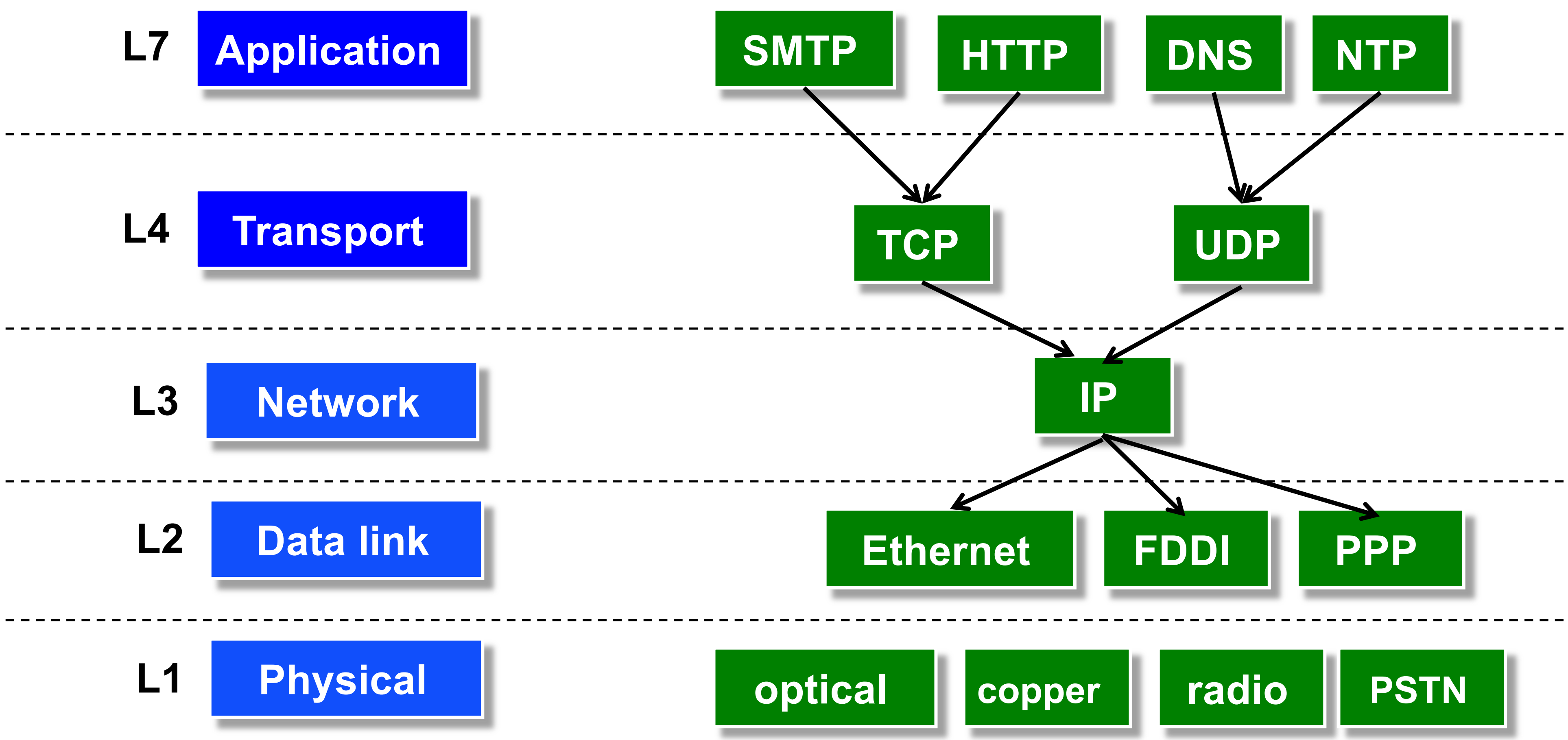
- Defines the syntax of communication
  - header → instructions for how to process the payload
  - Each protocol defines the format of its packet **headers**
    - *e.g. “the first 32 bits carry the destination address”*



What is a Protocol?

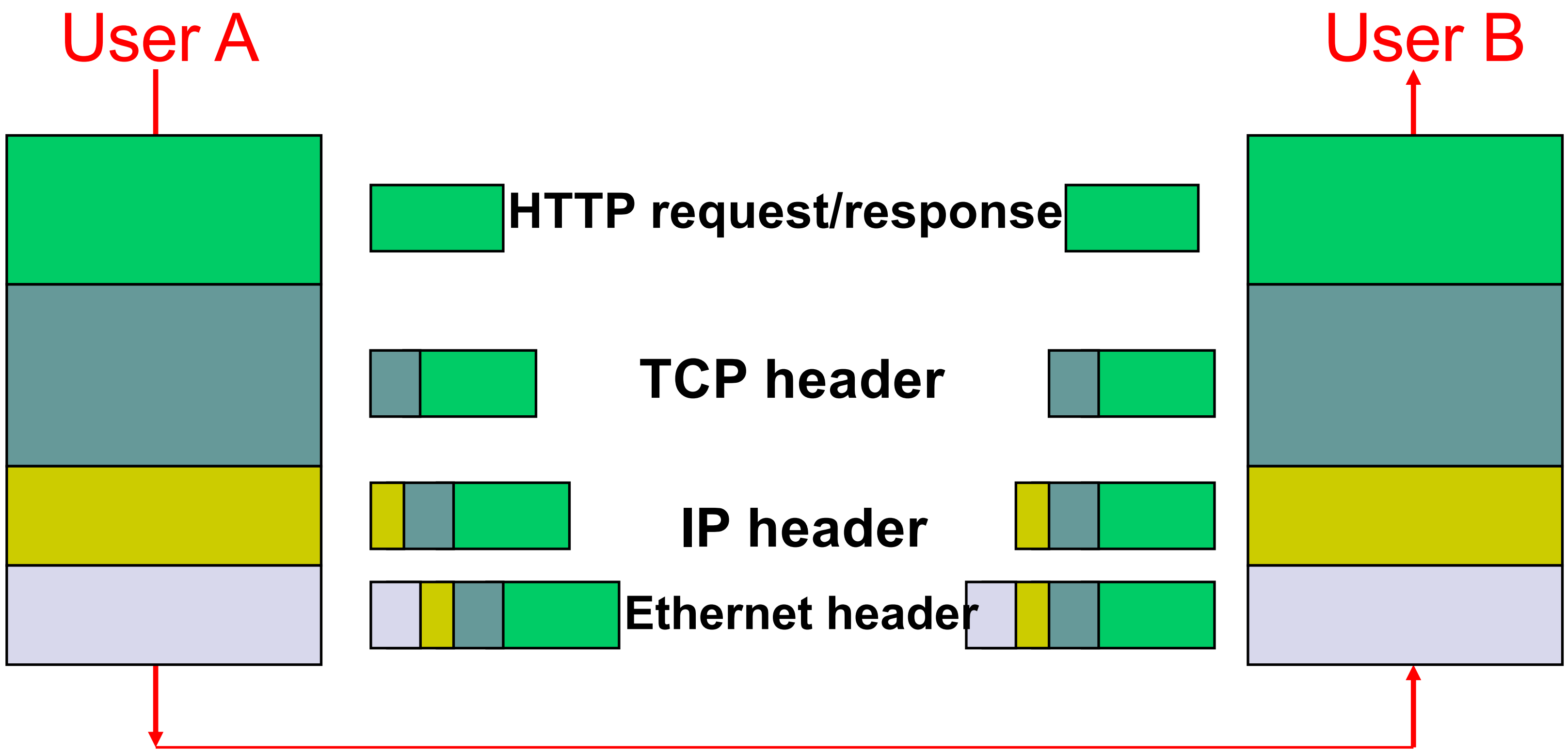
- An agreement between parties on how to communicate
- Defines the syntax of communication
- And semantics
  - “first a hello, then a request…”
  - we’ll study many protocols later in the quarter
- Protocols exist at many levels, hardware and software
  - defined by a variety of standards bodies (IETF, IEEE, ITU)

Protocols at different layers



NOTE: just one network-layer protocol!

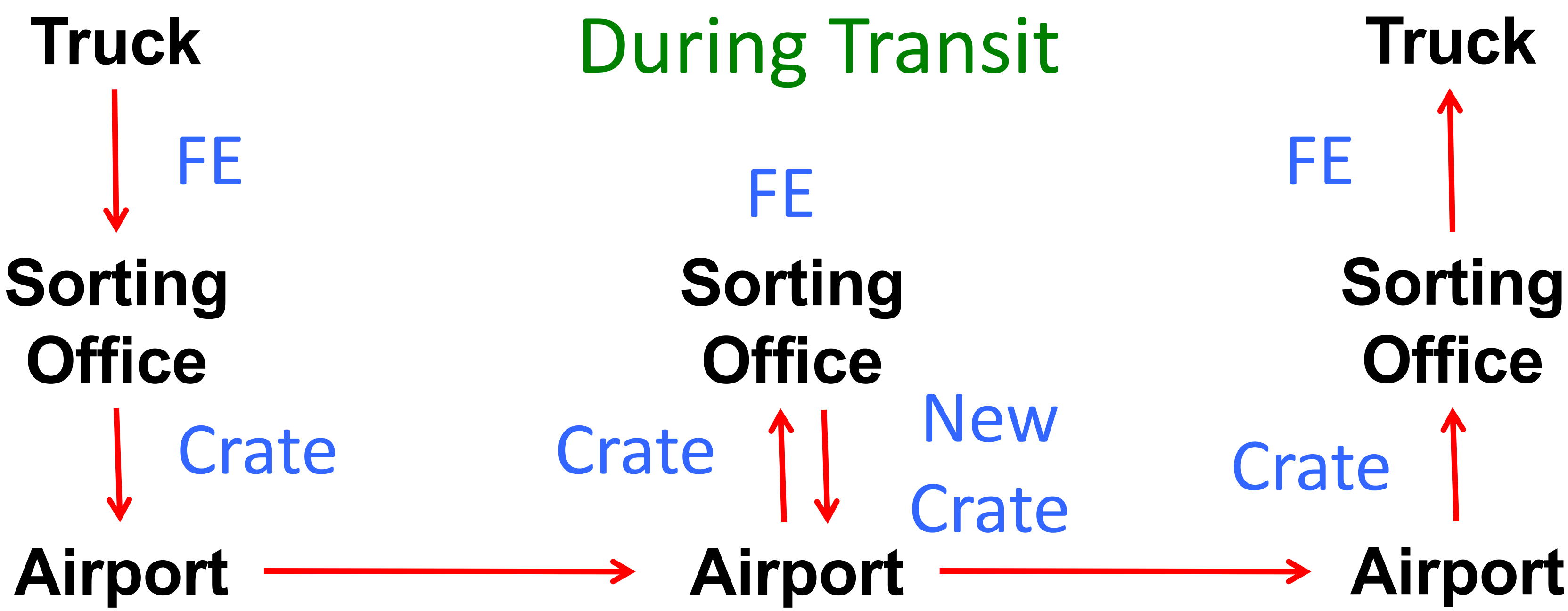
Layer Encapsulation: Protocol Headers



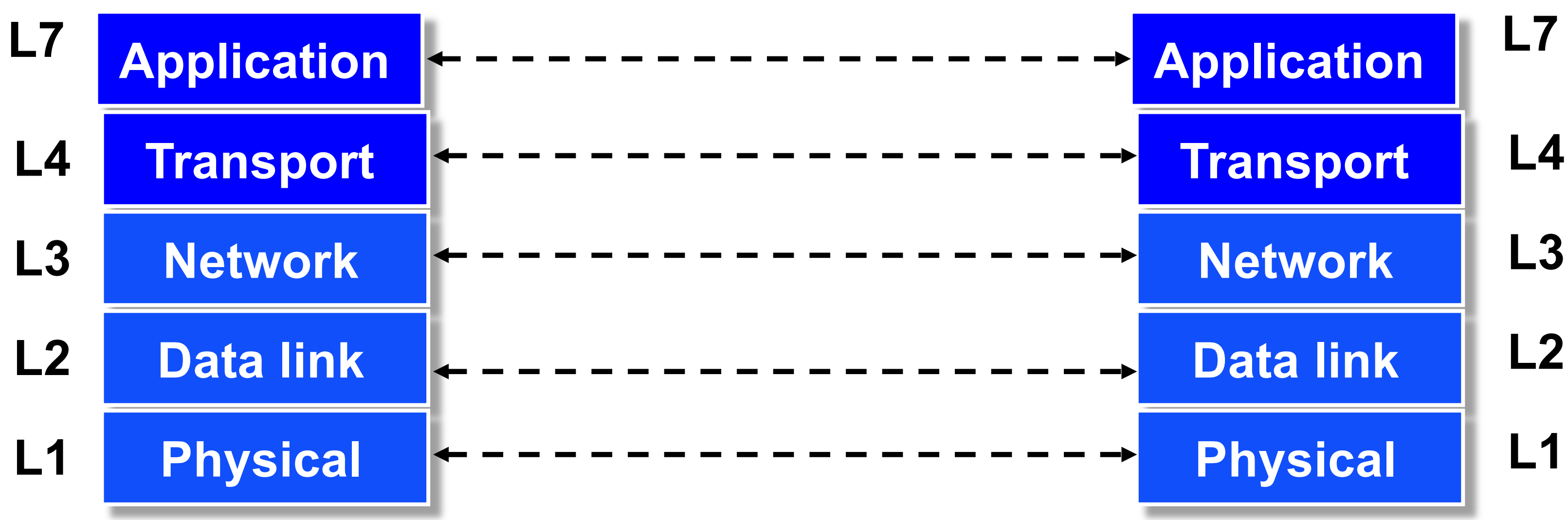
The Path Through FedEx

Higher "Stack" at Ends

Partial "Stack" During Transit



Deepest Packaging (Envelope+FE+Crate) at the Lowest Level of Transport



What gets implemented where?

What gets implemented at the end systems?

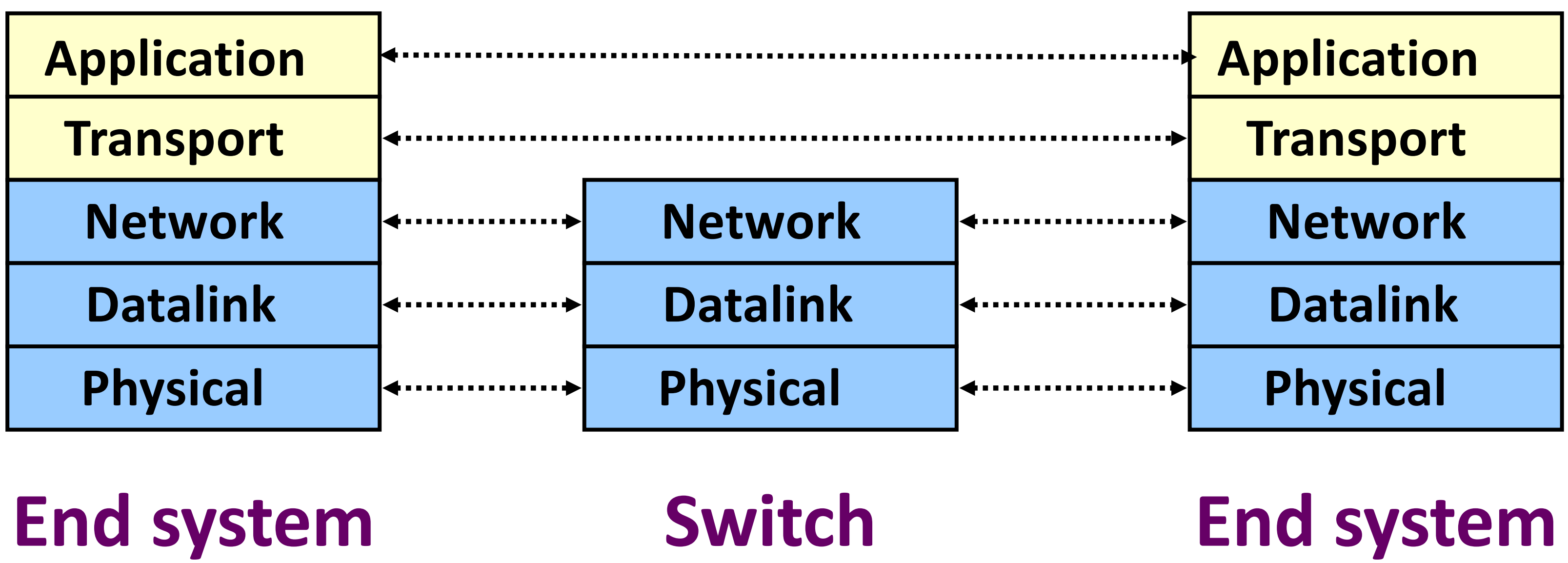
- Bits arrive on wire, must make it up to application
- Therefore, all layers must exist at host!

What gets implemented in the network?

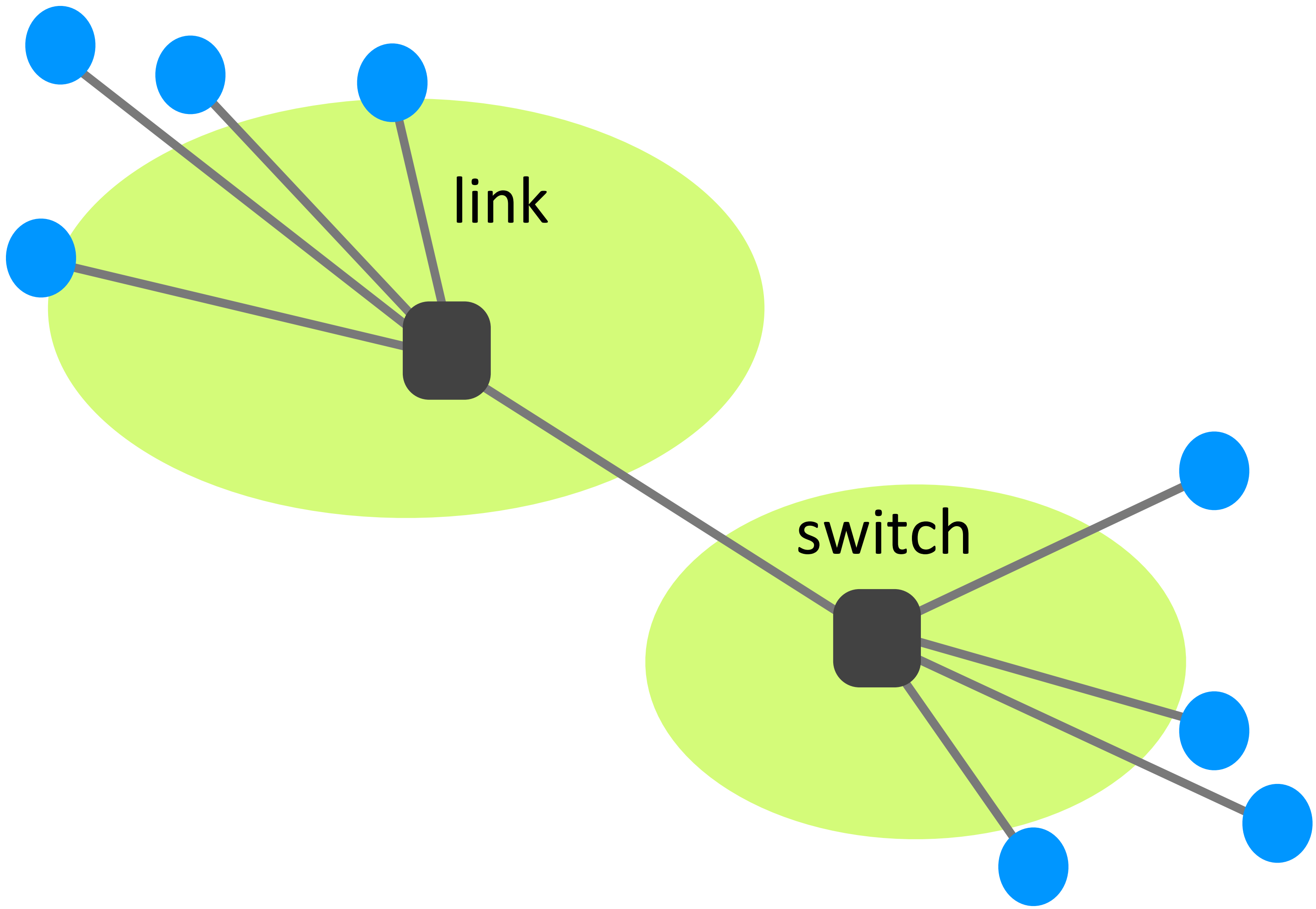
- Bits arrive on wire → physical layer (L1)
- Packets must be delivered across links and local networks → datalink layer (L2)
- Packets must be delivered between networks for global delivery → network layer (L3)
- The network does not support reliable delivery
  - Transport layer (and above) ***not***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

Simple Diagram

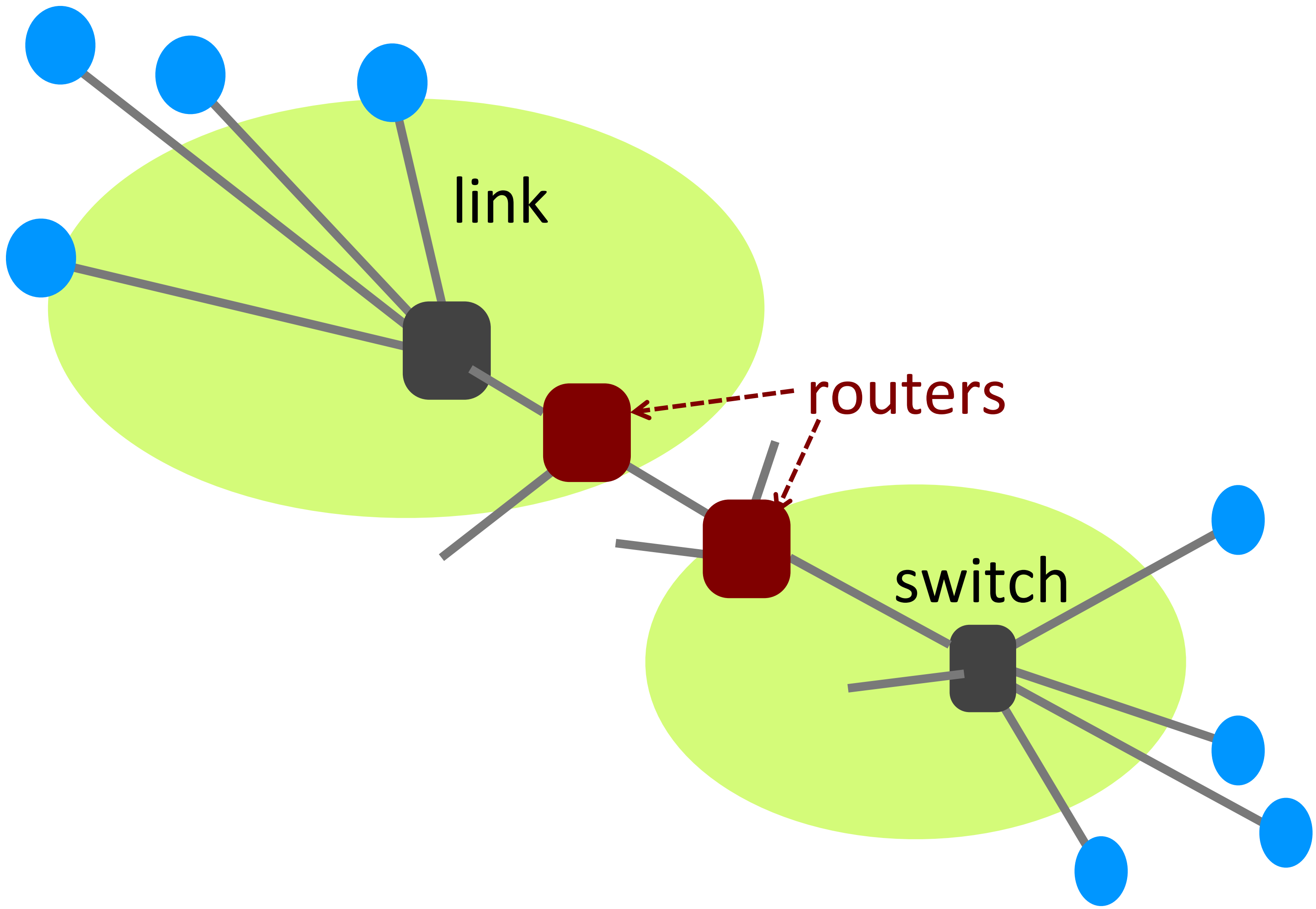
- Lower three layers implemented everywhere
- Top two layers implemented only at hosts



A closer look: network



A closer look: network



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)

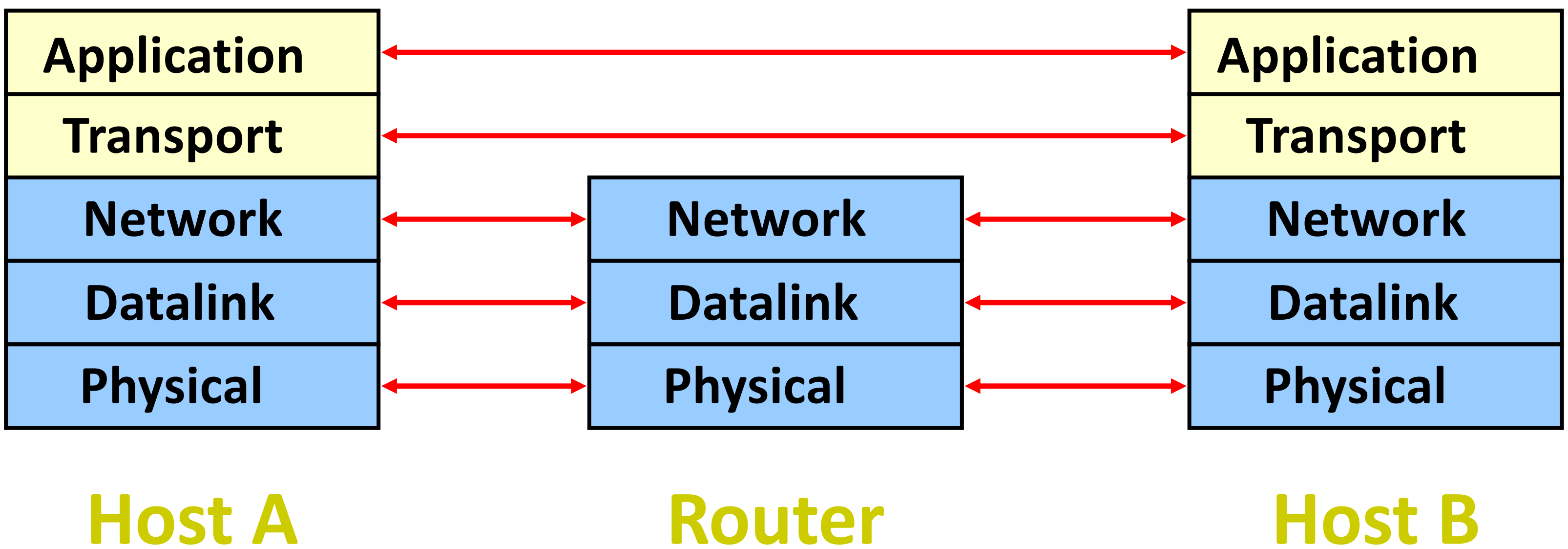
- Hence:
  - switches: implement physical and datalink layers (L1, L2)
  - routers: implement physical, datalink, network layers (L1, L2, L3)

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

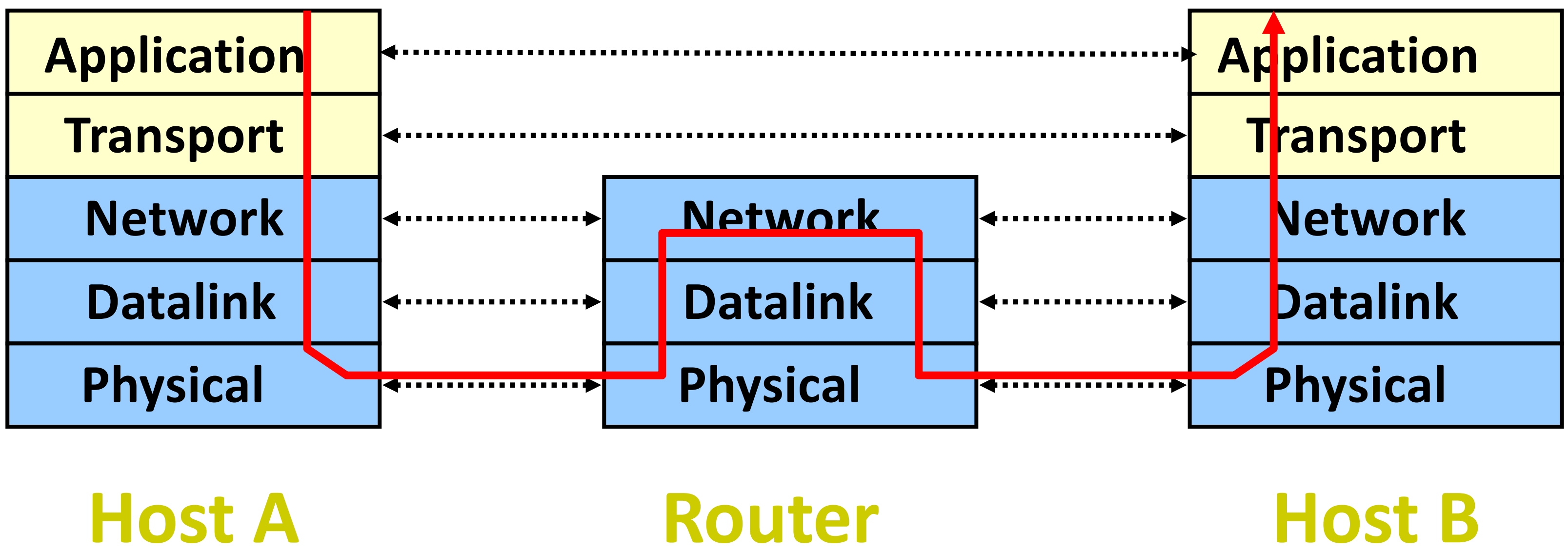
Logical Communication

- Layers interacts with peer's corresponding layer

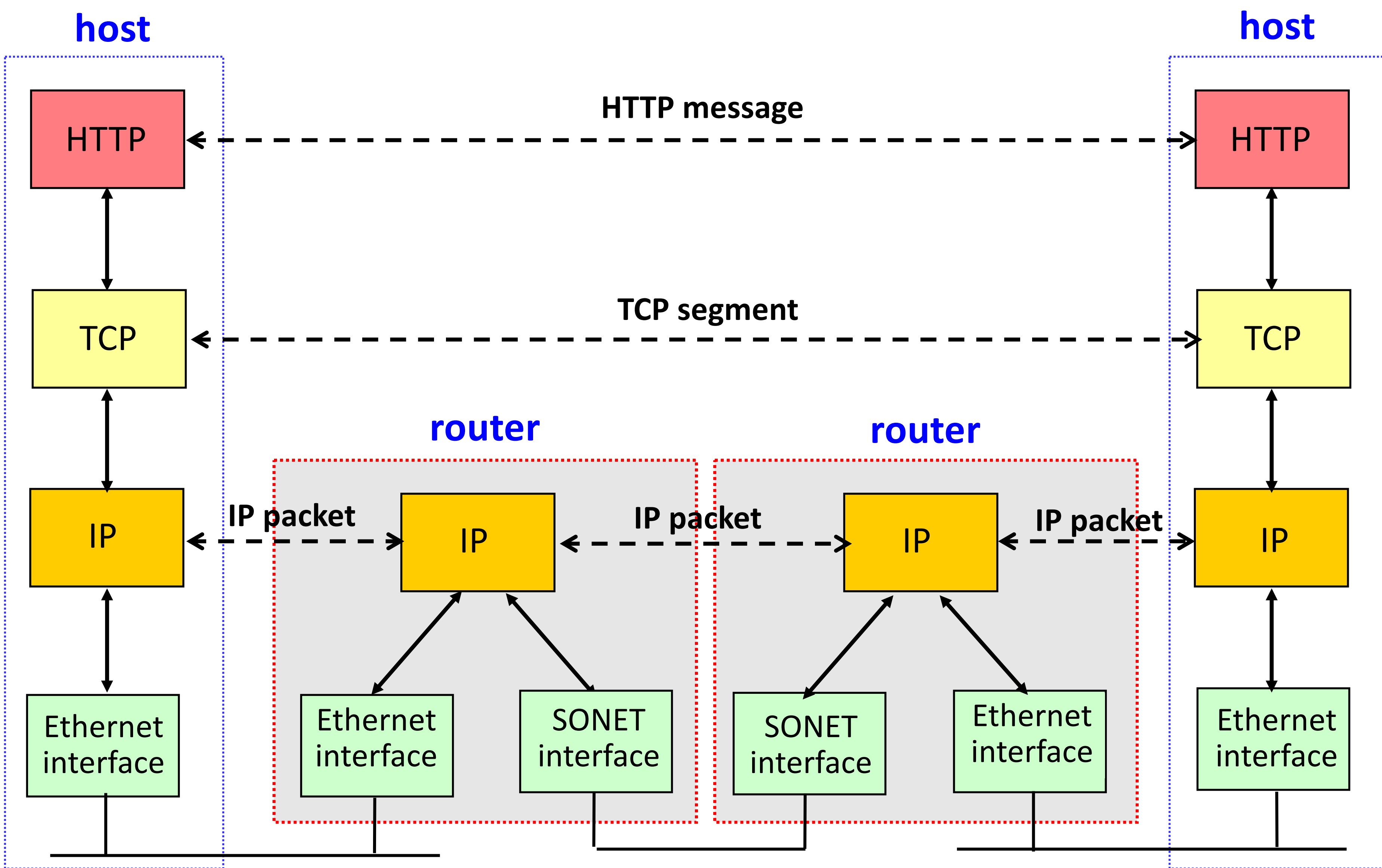


Physical Communication

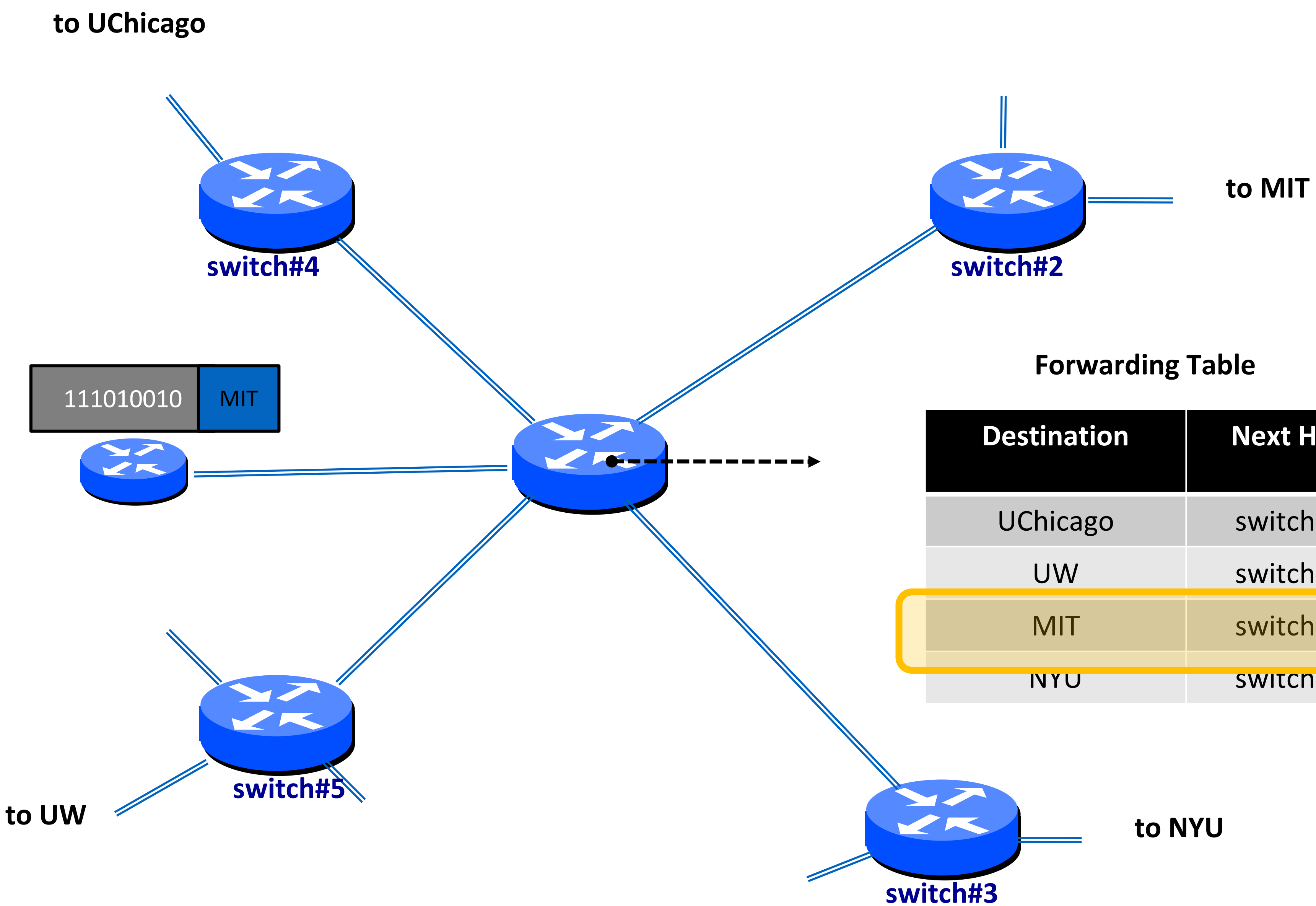
- Communication goes down to physical network
- Then up to relevant layer



A Protocol-Centric Diagram



The Network Layer



Forwarding Table

Destination	Next Hop
UChicago	switch#4
UW	switch#5
MIT	switch#2
NYU	switch#3

Three topics

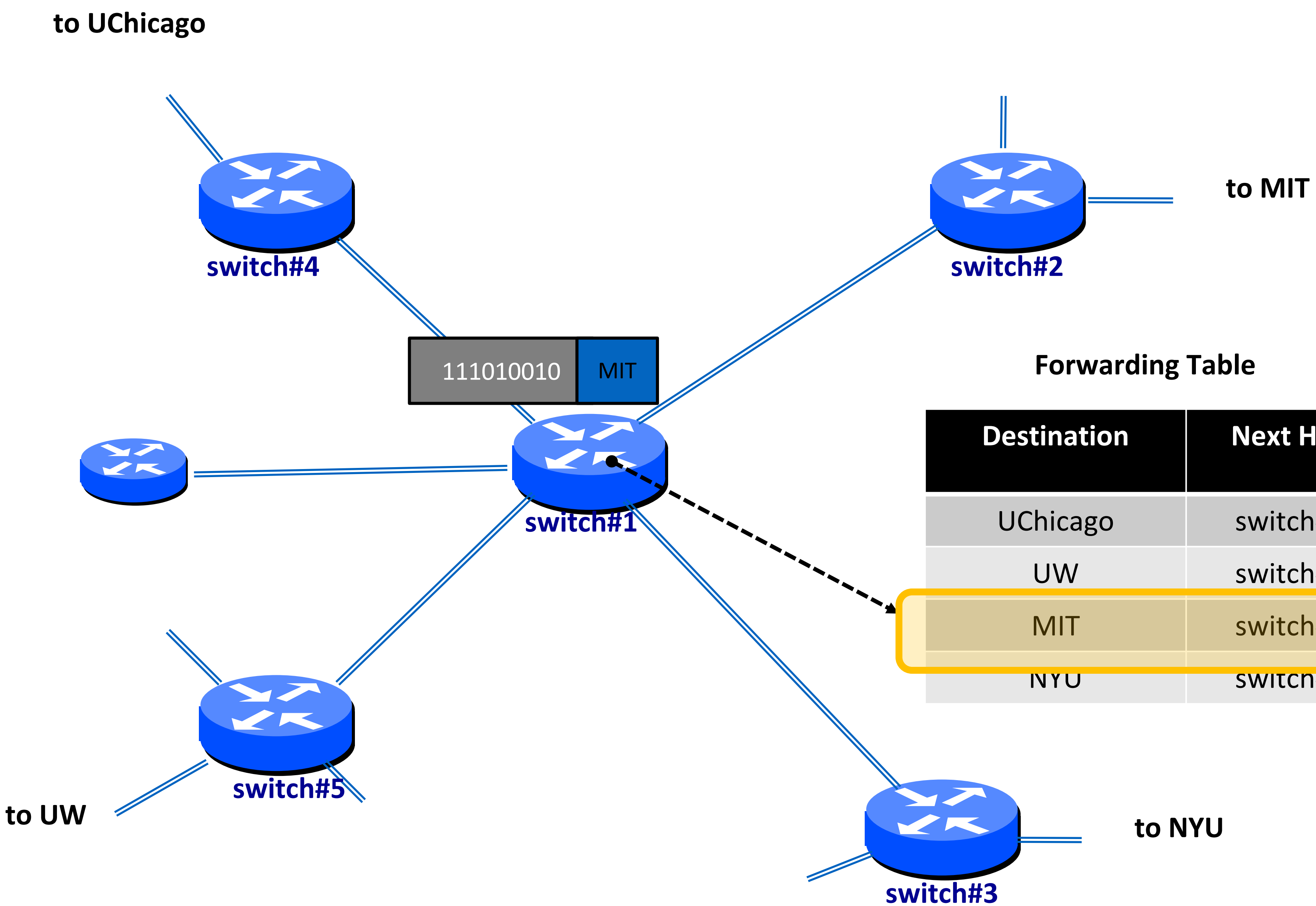
- Addressing
- Forwarding
- Routing

Addressing (for now)

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

Forwarding

- **Local** router process that determines the output link (a.k.a “next hop”) for each packet
- How
  - *read address from packet’s network layer header*
  - *search forwarding table*



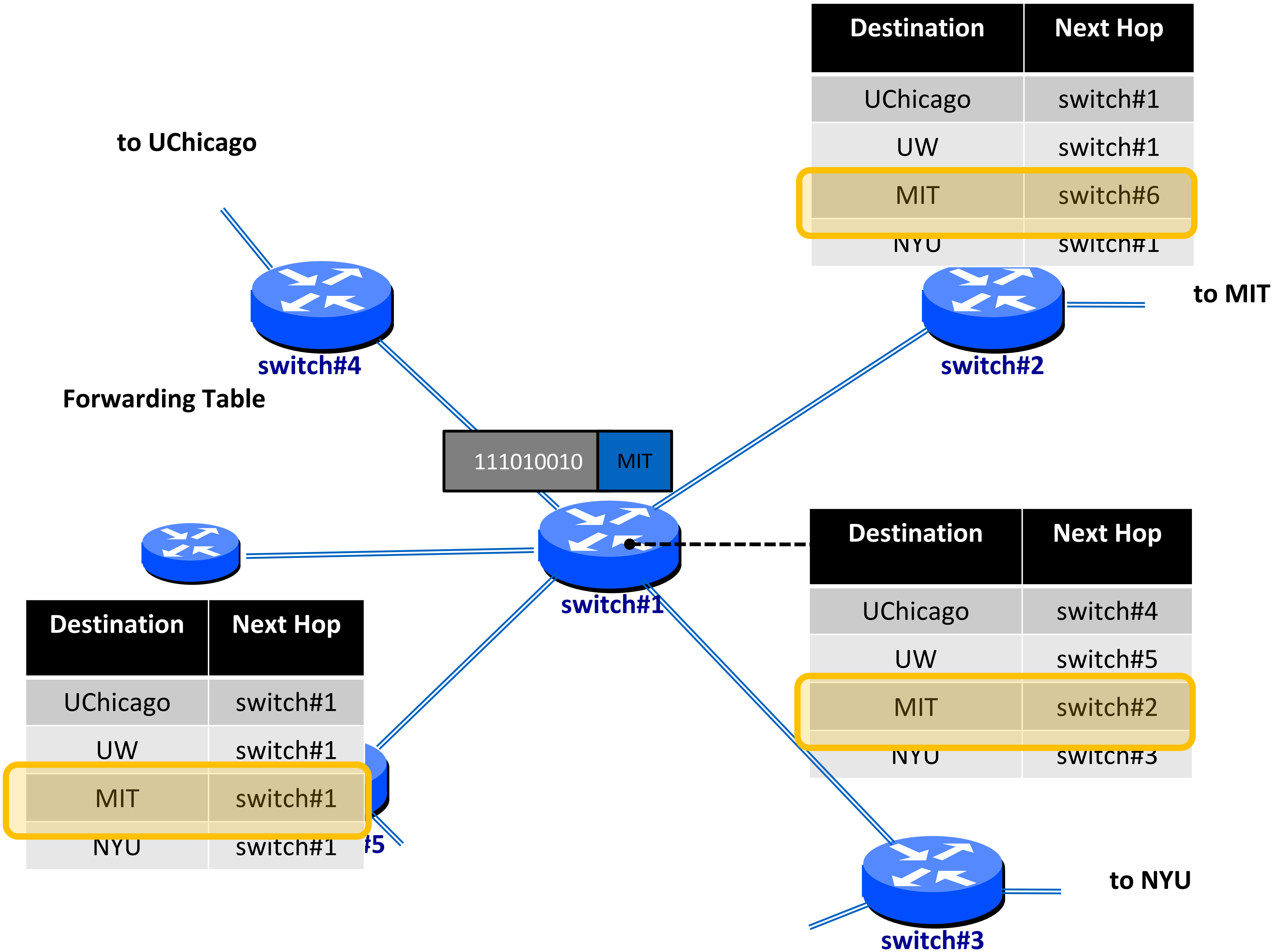
Forwarding Table

Destination	Next Hop
UChicago	switch#4
UW	switch#5
MIT	switch#2
NYU	switch#3

Routing

- **Network-wide** process that determines the content of forwarding tables

  → *determines the end-to-end path for each destination*



Destination	Next Hop
UChicago	switch#1
UW	switch#1
MIT	switch#6
NYU	switch#1

Destination	Next Hop
UChicago	switch#4
UW	switch#5
MIT	switch#2
NYU	switch#3

Destination	Next Hop
UChicago	switch#1
UW	switch#1
MIT	switch#1
NYU	switch#1

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: “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!

Routing Fundamentals

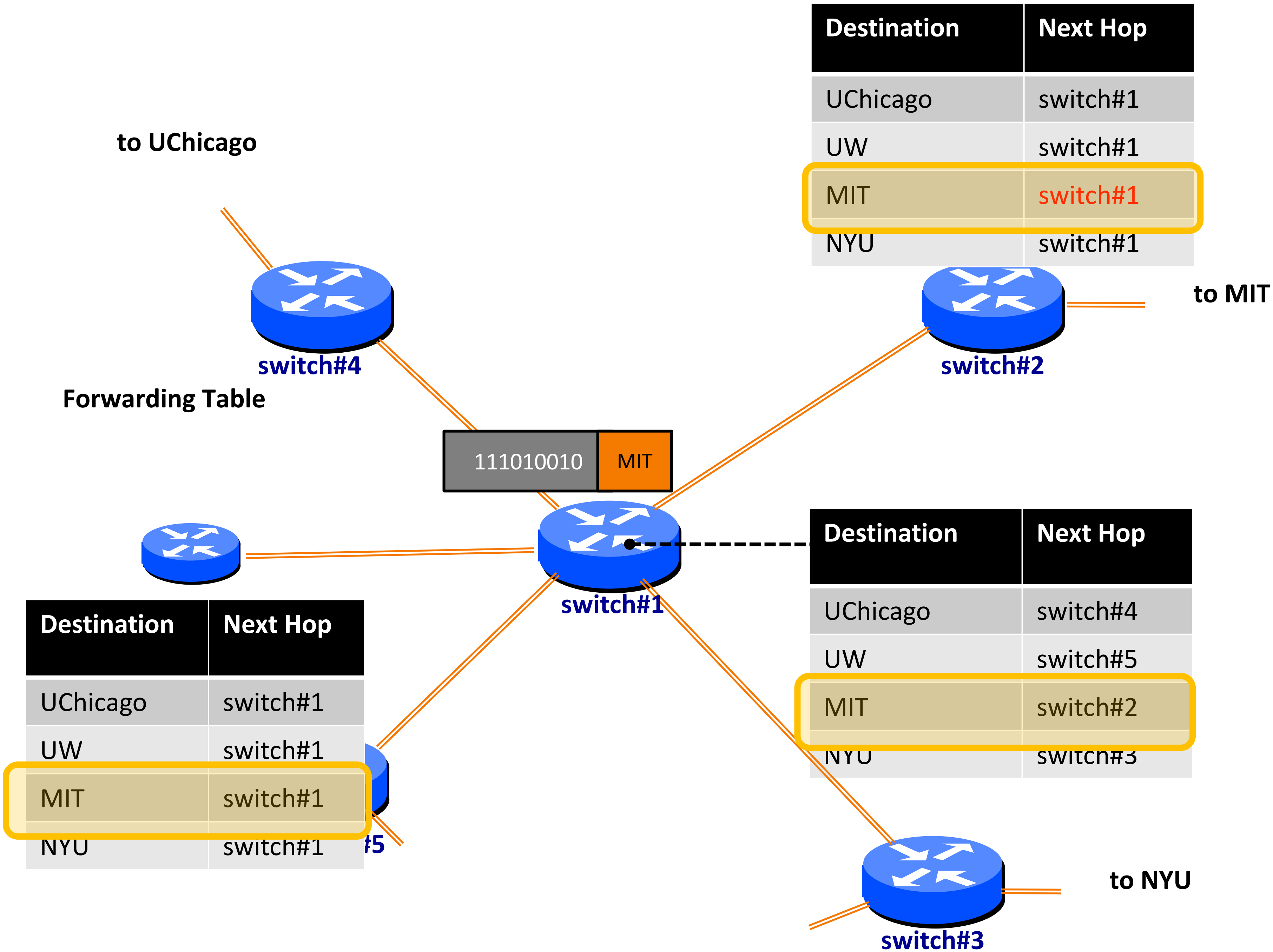
- Validity of routing state

Goal (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* routing state is the forwarding table in a single router
  - By itself, the state in a single router can't be evaluated
  - It must be evaluated in terms of the global context



Destination	Next Hop
UChicago	switch#1
UW	switch#1
MIT	switch#1
NYU	switch#1

Destination	Next Hop
UChicago	switch#4
UW	switch#5
MIT	switch#2
NYU	switch#3

Destination	Next Hop
UChicago	switch#1
UW	switch#1
MIT	switch#1
NYU	switch#1

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 can’t be evaluated
  - 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 packets take

(Will discuss later where this routing state comes from)

“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 if valid?
- Need a succinct correctness condition for routing
  - Suggestions?

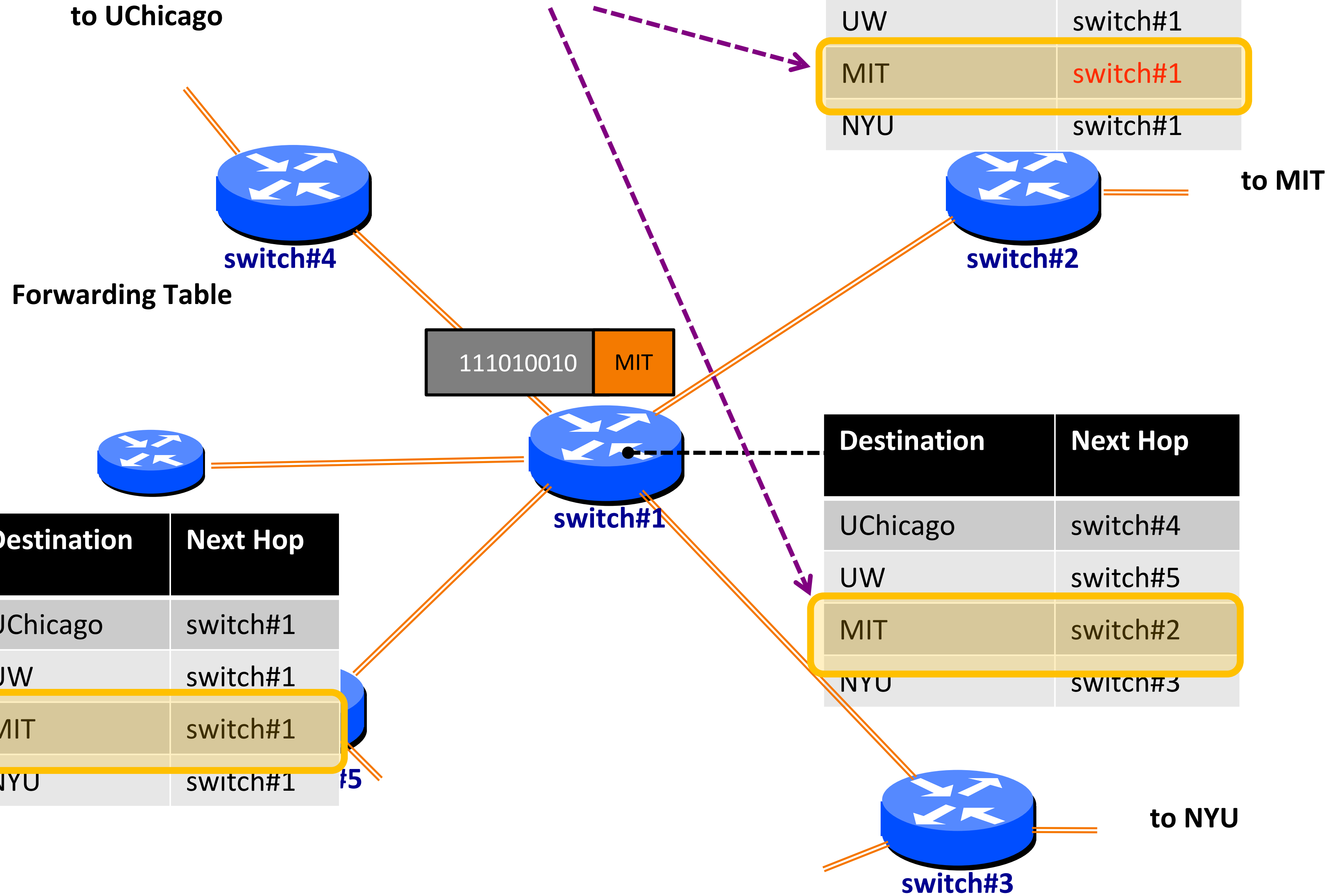
Necessary and Sufficient Condition

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

- A dead end is when there is no outgoing link (next-hop)
  - A packet arrives, but the forwarding decision does not yield any outgoing link

- A loop is when a packet cycles around the same set of nodes forever

Loop!

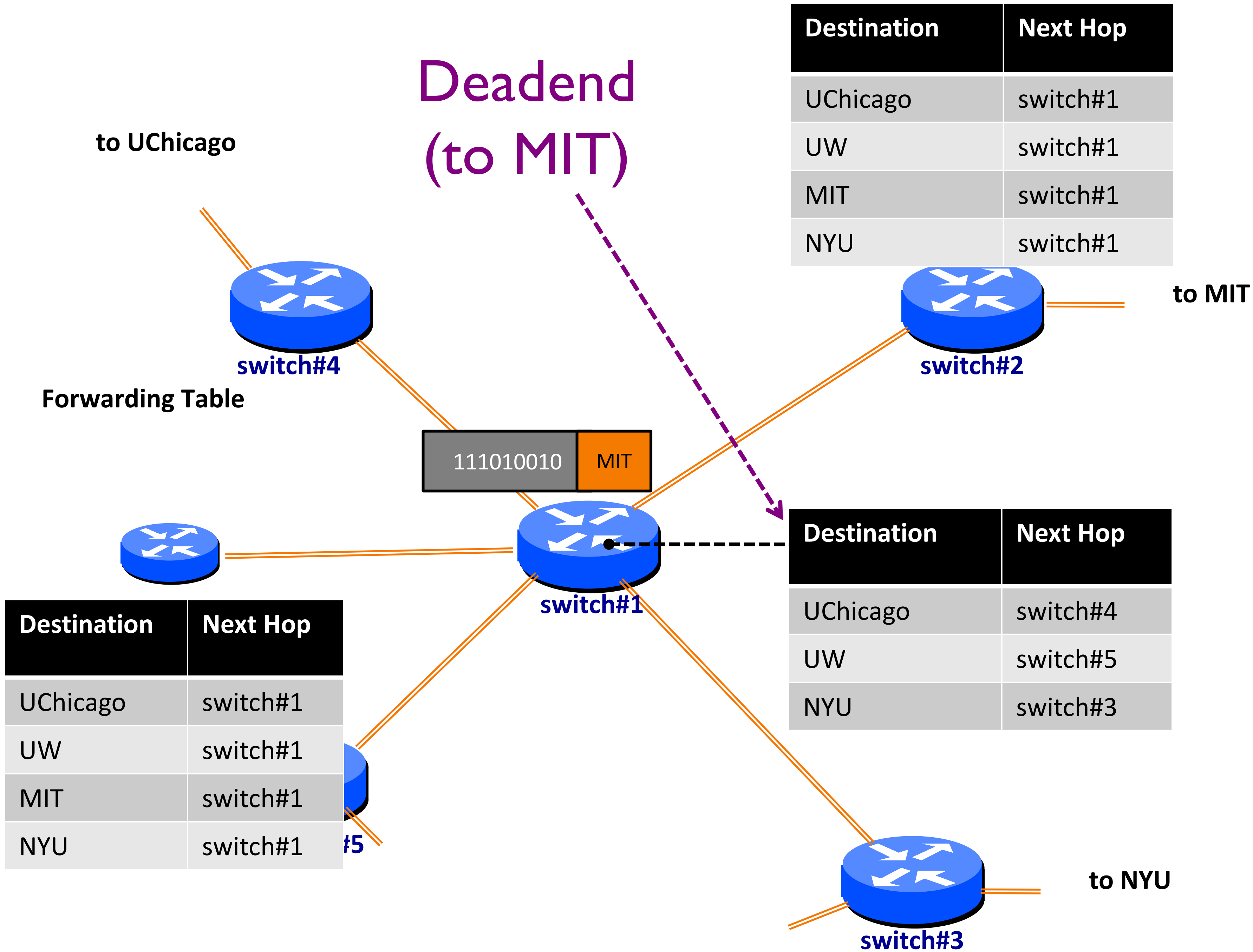


Destination	Next Hop
UChicago	switch#1
UW	switch#1
MIT	switch#1
NYU	switch#1

Destination	Next Hop
UChicago	switch#4
UW	switch#5
MIT	switch#2
NYU	switch#3

Destination	Next Hop
UChicago	switch#1
UW	switch#1
MIT	switch#1
NYU	switch#1

Deadend (to MIT)



Destination	Next Hop
UChicago	switch#1
UW	switch#1
MIT	switch#1
NYU	switch#1

Destination	Next Hop
UChicago	switch#4
UW	switch#5
NYU	switch#3

Destination	Next Hop
UChicago	switch#1
UW	switch#1
MIT	switch#1
NYU	switch#1

Necessary and Sufficient Condition

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

Necessary (“only if”): Easy

- If you run into a deadend before hitting destination, you’ll never reach the destination

- If you run into a loop, you’ll never reach destination

Sufficient ("if"):

- Assume no deadends, no loops

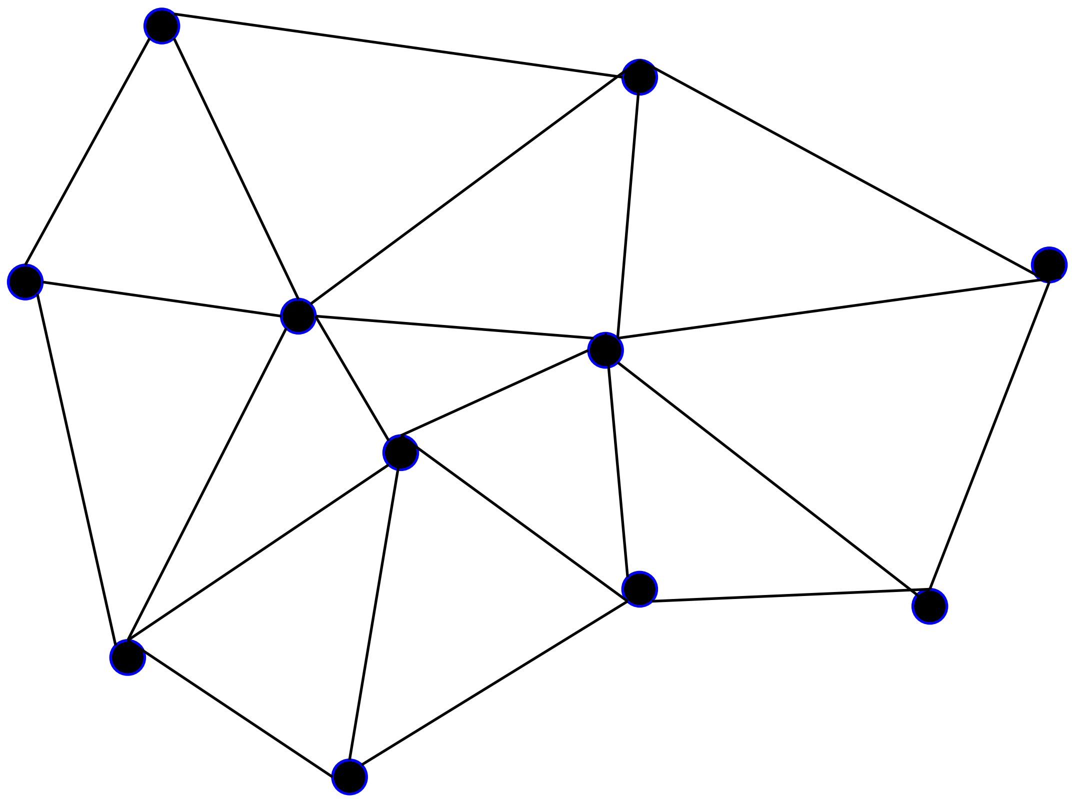
- Packet must keep wandering, but without repeating
  - If ever enter same switch from same link, will loop

- Only a finite number of possible links for it to visit
  - It cannot keep wandering forever without looping
  - Must eventually hit destination

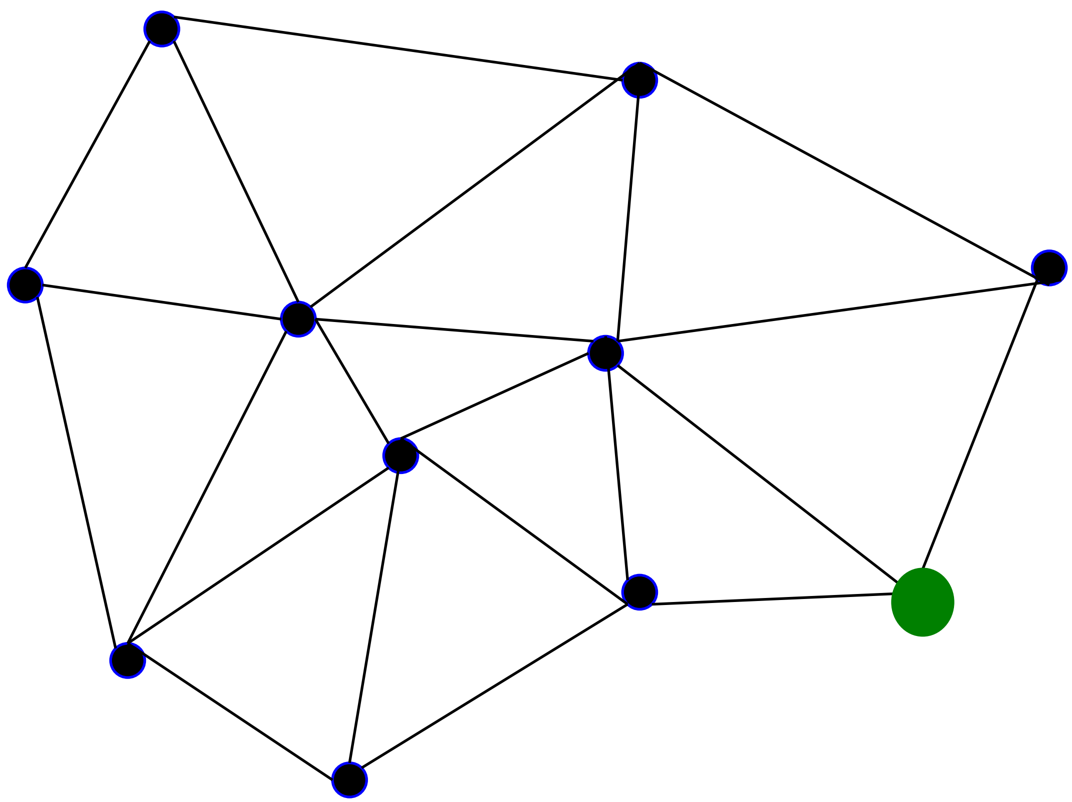
Checking Validity of Routing State

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

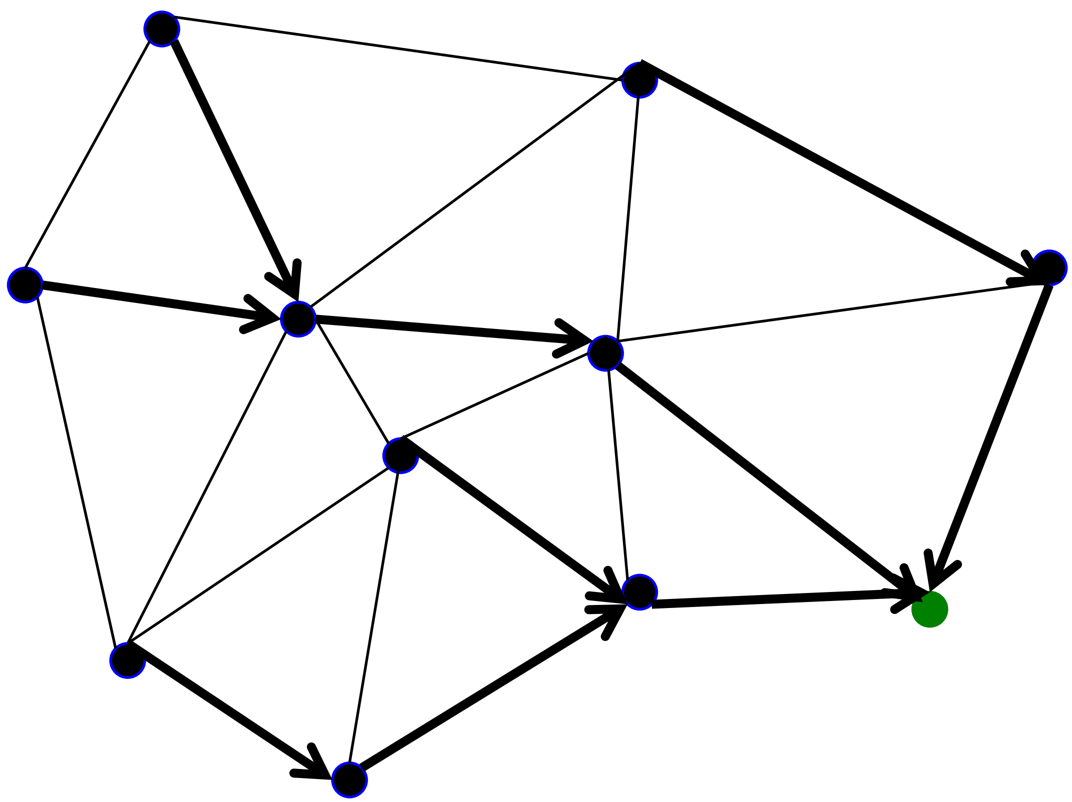
Example 1



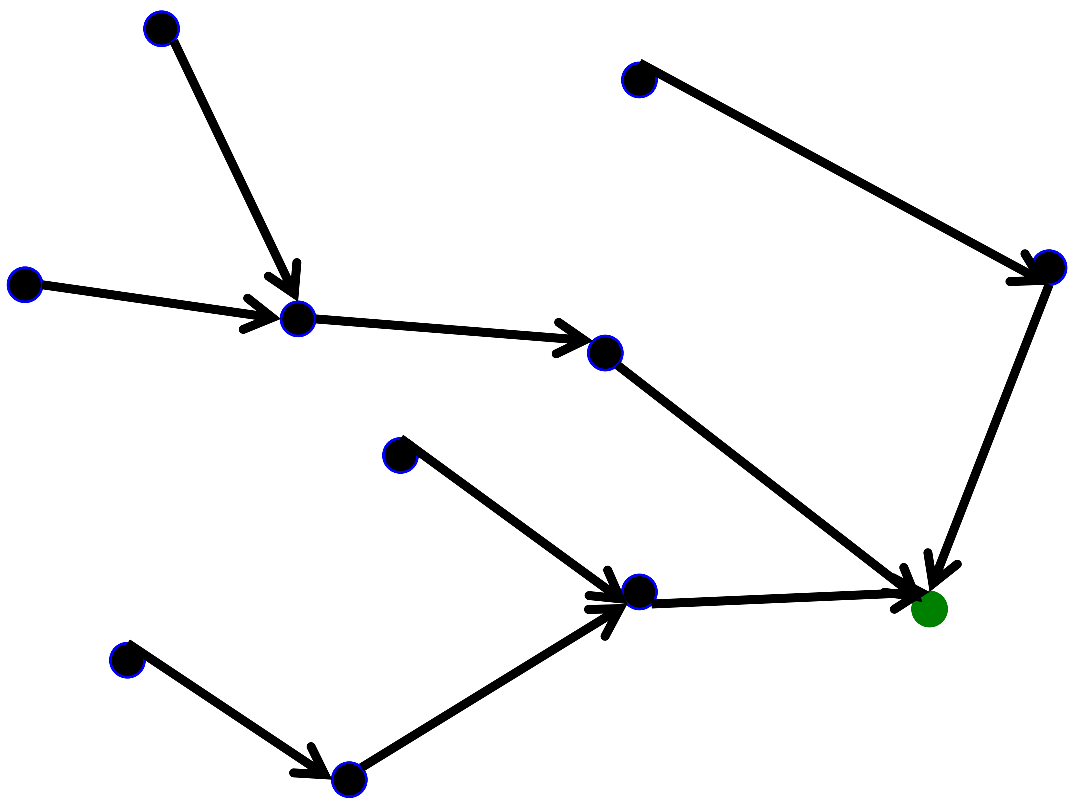
Pick Destination



Put arrows on outgoing links (to green dot)

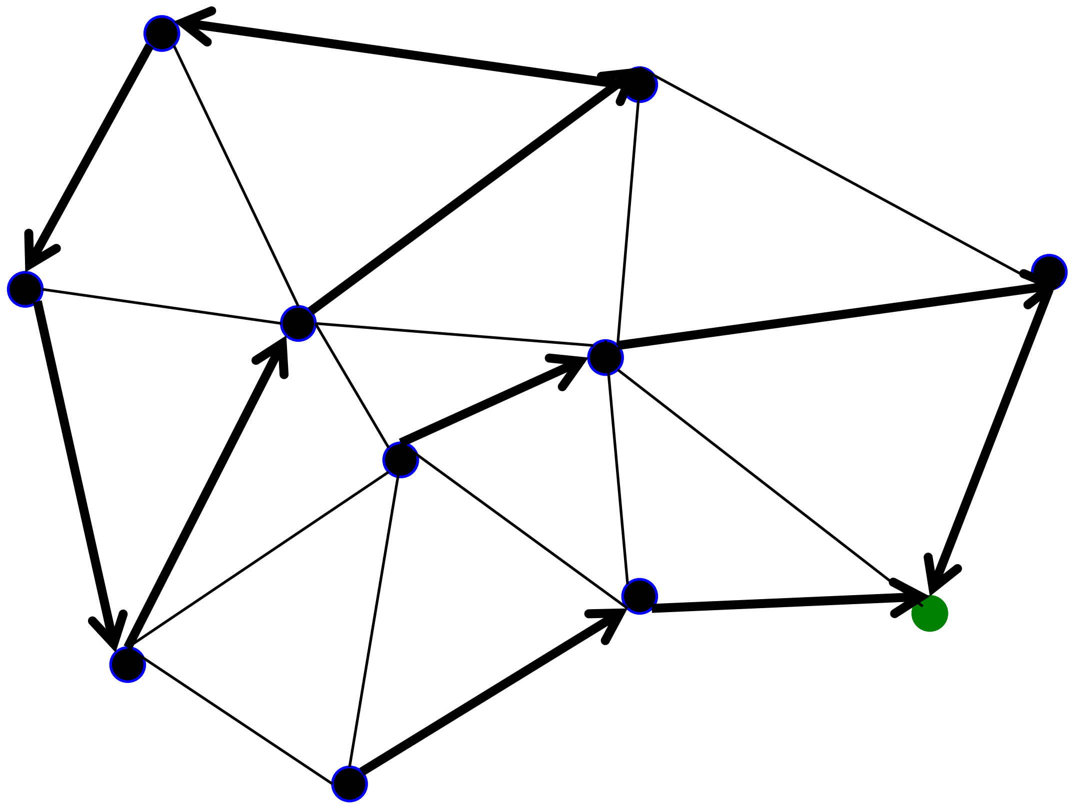


Remove Unused Links

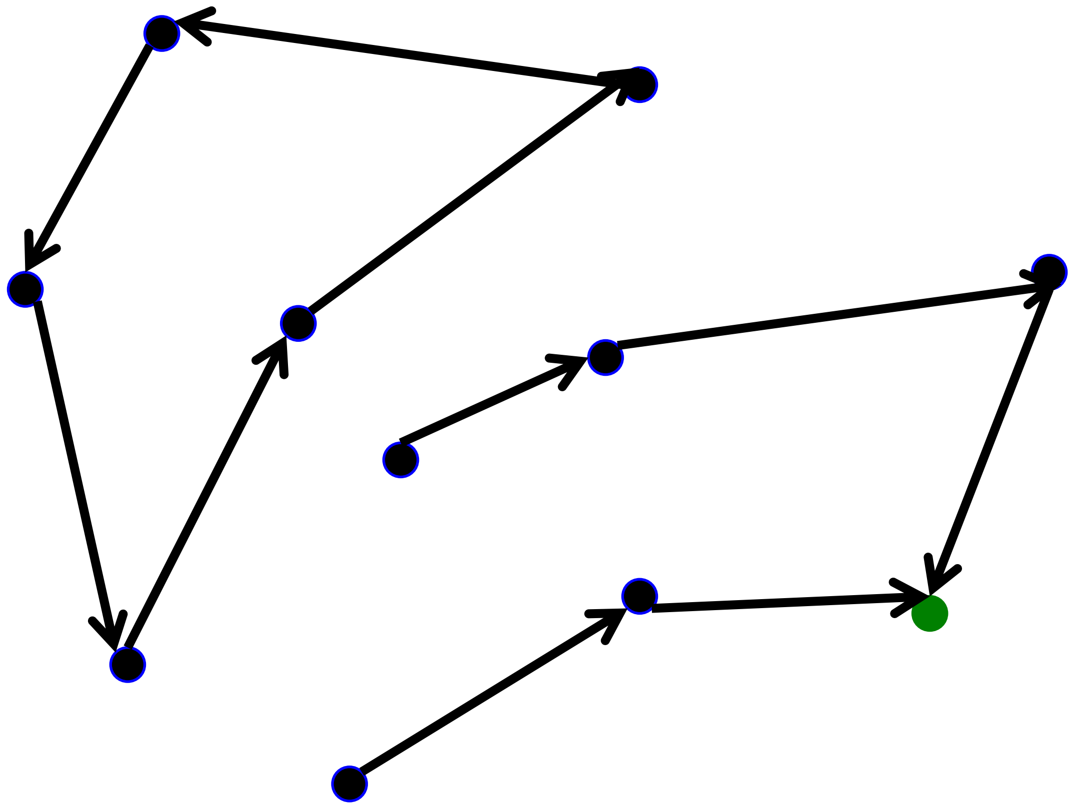


Leaves Spanning Tree: Valid

Second Example



Second Example



Is this valid?

Lesson….

- Very easy to check validity of routing state for a particular destination

- Deadends are obvious
  - Node without outgoing arrow

- Loops are obvious
  - Disconnected from rest of graph

Goal (for routing)

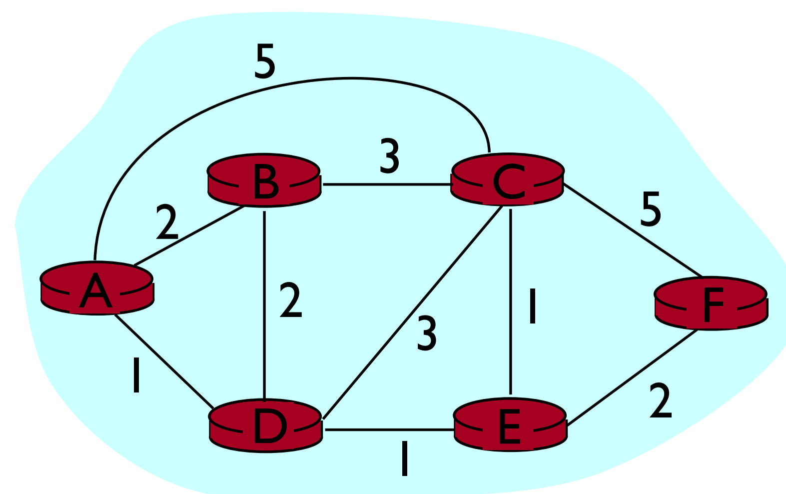
v1: Find a path to a given destination

v2: Find a *least cost path* to a given destination

Routing

- Goal: determine a "good" path through the network from source to destination
  - Good means usually the shortest path

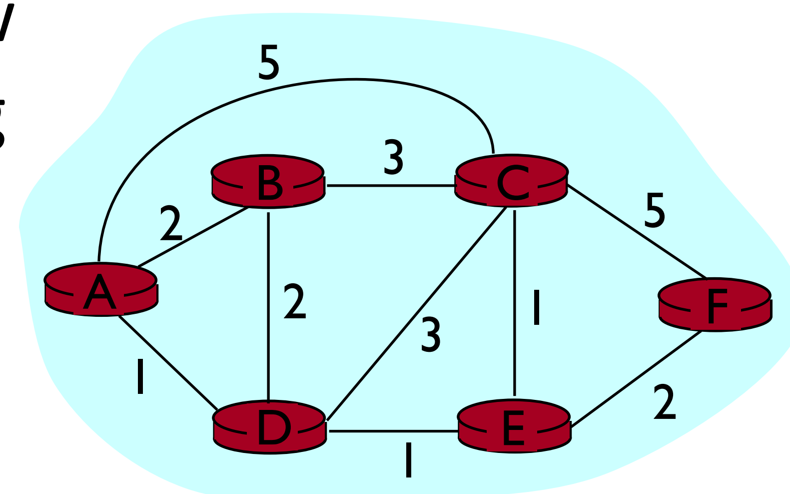
- Network modeled as a graph
  - Routers → nodes
  - Link →edges
    - Edge cost: delay, congestion level,...



Routing Problem

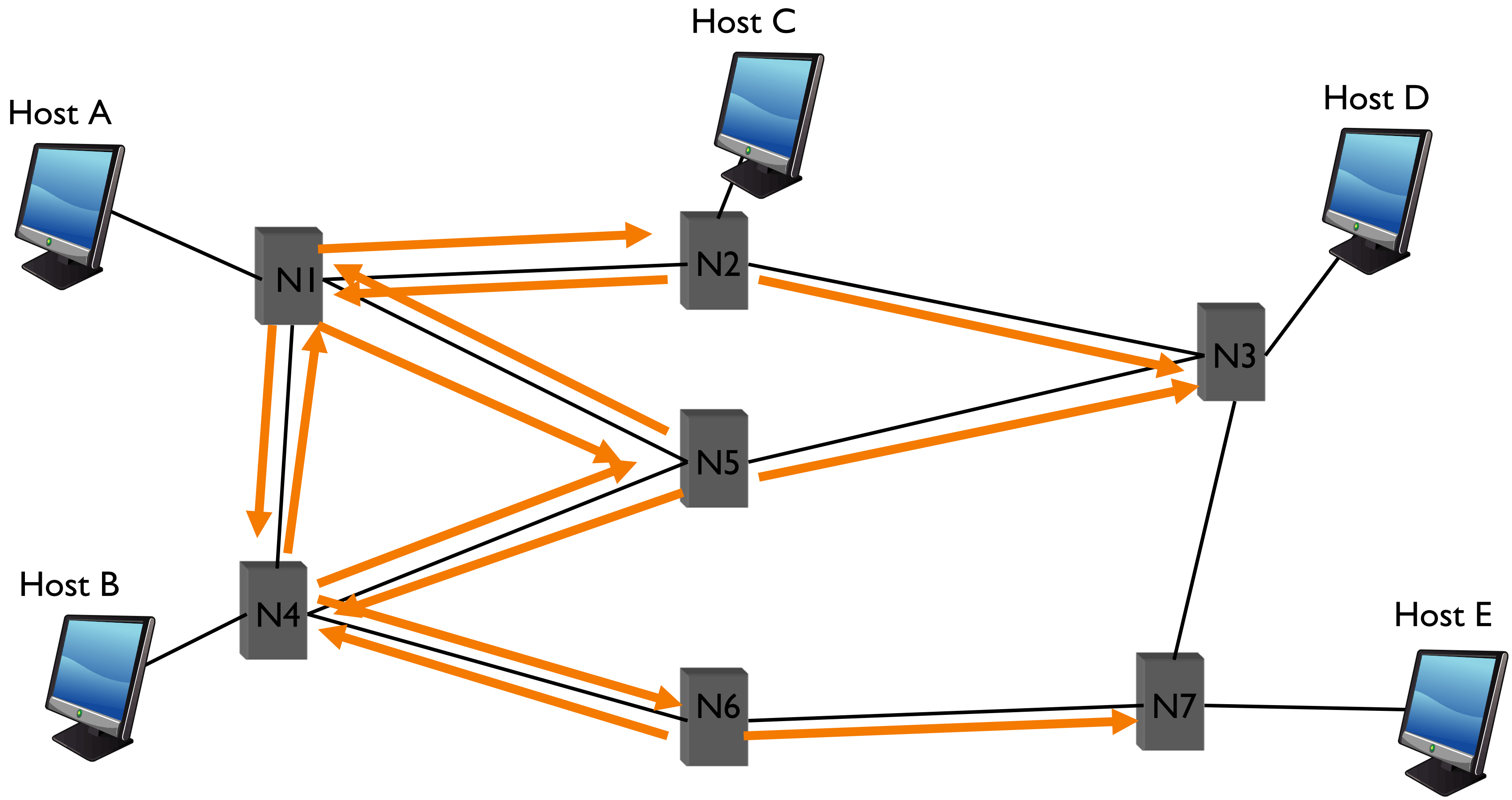
- Assume
  - A network with N nodes, where each edge is associated a cost
  - A node knows only its neighbors and the cost to reach them

- How does each node learns how to reach every other node along the shortest path?

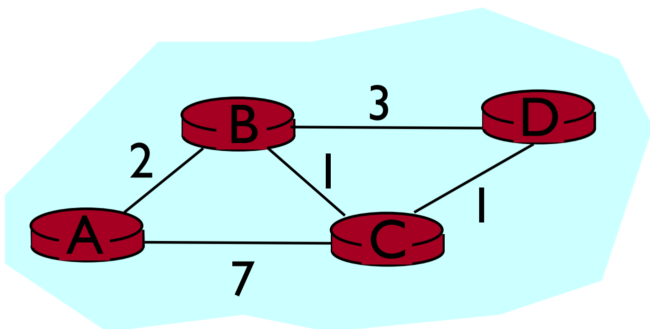


Distance Vector: Control Traffic

- When the routing table of a node changes, it sends table to neighbors
  - A node updates its table with information received from neighbors



Example: Distance Vector Algorithm



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	∞	-

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

```
Initialization:
  for all neighbors V  do
    if V adjacent to A
      D(A, V) = c(A,V);
  else
      D(A, V) = ∞;
…
```

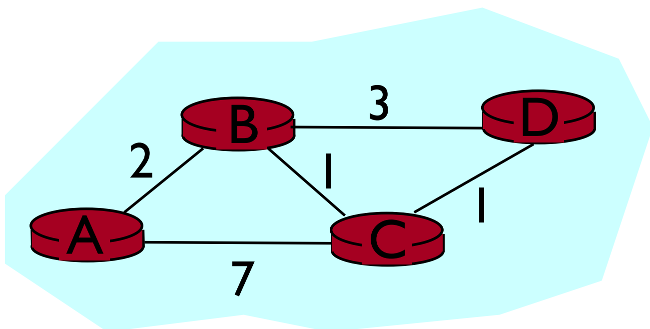
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

Example: 1st Iteration (C → A)



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	∞	-

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

(D(C,A), D(C,B), D(C,D))

Node C

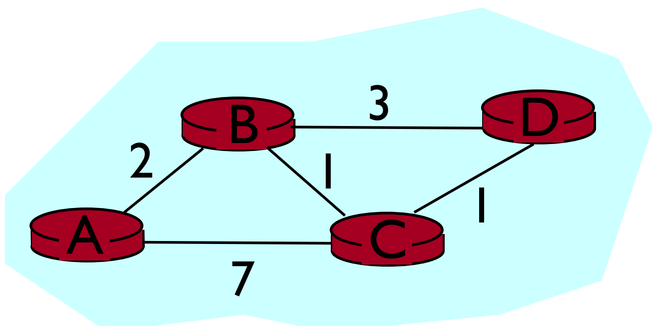
Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

```
…
loop:
…
  else if (update D(V, Y) received from V)
    for all destinations Y do
      if (destination Y through V)
        D(A,Y) = D(A,V) + D(V, Y);
      else
        D(A, Y) = min(D(A, Y),
                      D(A, V) + D(V, Y));
  if (there is a new minimum for dest. Y)
    send D(A, Y) to all neighbors
forever
```

Example: 1st Iteration (C → A)




```
…
loop:
…
  else if (update D(V, Y) received from V)
    for all destinations Y do
      if (destination Y through V)
        D(A,Y) = D(A,V) + D(V, Y);
      else
        D(A, Y) = min(D(A, Y),
                      D(A, V) + D(V, Y));
  if (there is a new minimum for dest. Y)
    send D(A, Y) to all neighbors
forever
```

Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	**8**	**C**

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

$D(A,D) = \min(D(A, D), D(A,C)+D(C,D))$
$= \min(\infty\ , \mathbf{7 + 1}) = \mathbf{8}$

(D(C,A), D(C,B), D(C,D))

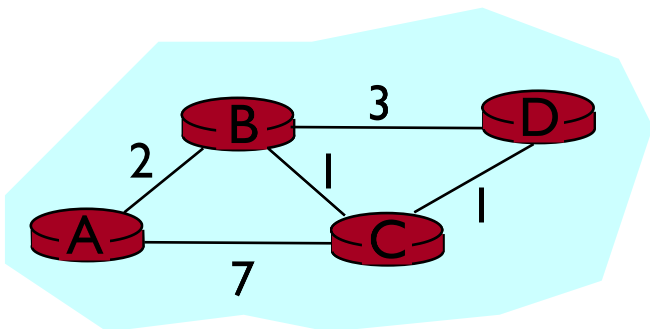
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

Example: 1st Iteration (B → A)



Node A

Dest.	Cost	NextHop
B	2	B
C	7	C
D	8	C

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

Node C

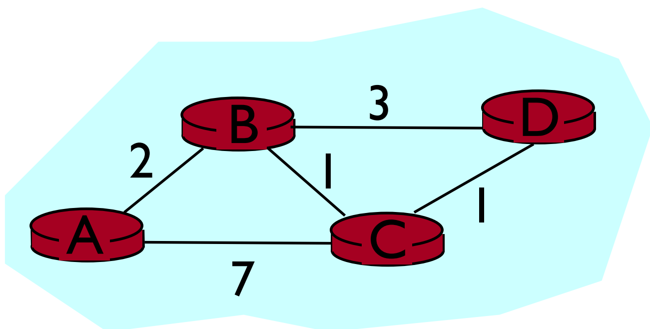
Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

```
…
loop:
…
  else if (update D(V, Y) received from V)
    for all destinations Y do
      if (destination Y through V)
        D(A,Y) = D(A,V) + D(V, Y);
      else
        D(A, Y) = min(D(A, Y),
                      D(A, V) + D(V, Y));
  if (there is a new minimum for dest. Y)
    send D(A, Y) to all neighbors
forever
```

Example: 1st Iteration (B→A, C→A)



Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	5	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	3	D

D(A,D) = min(D(A,D), D(A,B) + D(B,D))
= min(8, 2 + 3) = 5

D(A,C) = min(D(A,C), D(A,B) + D(B,C))
= min(7, 2 + 1) = 3

```
…
loop:
…
else if (update D(V, Y) received from V)
  for all destinations Y do
    if (destination Y through V)
      D(A,Y) = D(A,V) + D(V, Y);
    else
      D(A, Y) = min(D(A, Y),
                    D(A, V) + D(V, Y));
if (there is a new minimum for dest. Y)
  send D(A, Y) to all neighbors
forever
```

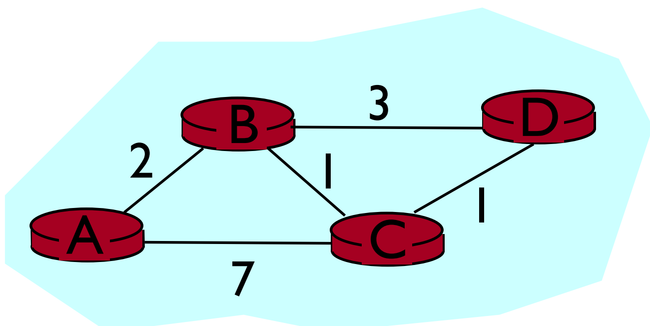
Node C

Dest.	Cost	NextHop
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	∞	-
B	3	B
C	1	C

Example: End of 1st Iteration




```
…
loop:
…
  else if (update D(V, Y) received from V)
    for all destinations Y do
      if (destination Y through V)
        D(A,Y) = D(A,V) + D(V, Y);
      else
        D(A, Y) = min(D(A, Y),
                      D(A, V) + D(V, Y));
  if (there is a new minimum for dest. Y)
    send D(A, Y) to all neighbors
forever
```

Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	5	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	2	C

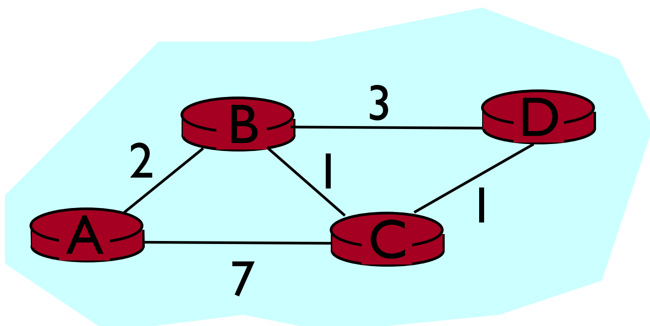
Node C

Dest.	Cost	NextHop
A	3	B
B	1	B
D	1	D

Node D

Dest.	Cost	NextHop
A	4	B
B	3	B
C	1	C

Example: End of 3rd Iteration




```
…
loop:
…
  else if (update D(V, Y) received from V)
    for all destinations Y do
      if (destination Y through V)
        D(A,Y) = D(A,V) + D(V, Y);
      else
        D(A, Y) = min(D(A, Y),
                      D(A, V) + D(V, Y));
  if (there is a new minimum for dest. Y)
    send D(A, Y) to all neighbors
forever
```

Node A

Dest.	Cost	NextHop
B	2	B
C	3	B
D	4	B

Node B

Dest.	Cost	NextHop
A	2	A
C	1	C
D	2	C

Node C

Dest.	Cost	NextHop
A	3	B
B	1	B
D	1	D

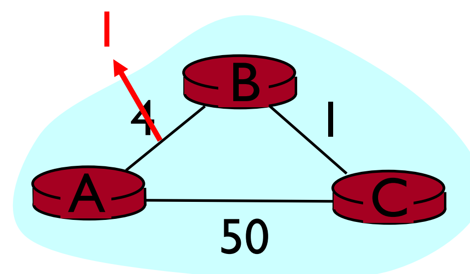
Node D

Dest.	Cost	NextHop
A	4	C
B	2	C
C	1	C

Nothing changes → algorithm terminates

Distance Vector: Link Cost Changes

```
loop:
  wait (link cost update or update message)
  if (c(A,V) changes by d)
    for all destinations Y through V do
      D(A,Y) =  D(A,Y) + d
  else if (update D(V, Y) received from V)
    for all destinations Y do
      if (destination Y through V)
        D(A,Y) = D(A,V) + D(V, Y);
      else
        D(A, Y) = min(D(A, Y), D(A, V) + D(V, Y));
  if (there is a new minimum for destination Y)
    send D(A, Y) to all neighbors
forever
```



Node B

D	C	N
A	4	A
C	1	C

D	C	N
A	1	A
C	1	C

D	C	N
A	1	A
C	1	C

D	C	N
A	1	A
C	1	C

Node C

D	C	N
A	5	B
B	1	B

D	C	N
A	5	B
B	1	B

D	C	N
A	2	B
B	1	B

D	C	N
A	2	B
B	1	B

"good news travels fast"

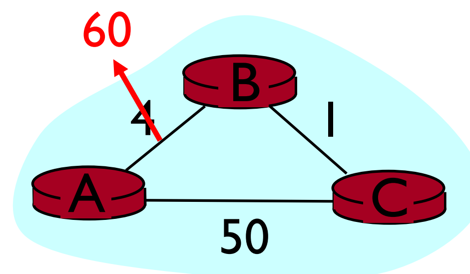
time

Link cost changes here

Algorithm terminates

DV: Count to Infinity Problem

```
loop:
  wait (link cost update or update message)
  if (c(A,V) changes by d)
     for all destinations Y through V do
        D(A,Y) =  D(A,Y) + d
   else if (update D(V, Y) received from V)
     for all destinations Y do
        if (destination Y through V)
          D(A,Y) = D(A,V) + D(V, Y);
        else
           D(A, Y) = min(D(A, Y), D(A, V) + D(V, Y));
   if (there is a new minimum for destination Y)
     send D(A, Y) to all neighbors
forever
```



Node B

D	C	N
A	4	A
C	1	B

D	C	N
A	6	C
C	1	B

D	C	N
A	6	C
C	1	B

D	C	N
A	8	C
C	1	B

Node C

D	C	N
A	5	B
B	1	B

D	C	N
A	5	B
B	1	B

D	C	N
A	7	B
B	1	B

D	C	N
A	2	B
B	1	B

…

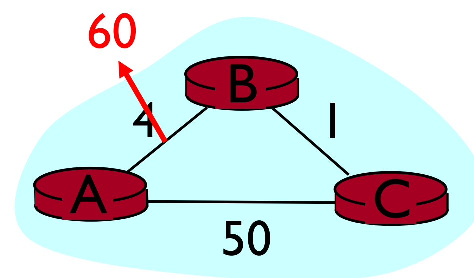
"bad news travels slowly"

Link cost changes here; note that B also maintains the shortest route to A for C, which is 6. Thus D(B, A) becomes 6 !

time

Distance Vector: Poisoned Reverse

- If C routes through B to get to A:
  - *C tells B its (C's) distance to A is infinite (so B won't route to A via C)*
  - *Will this completely solve count to infinity problem?*



NO! multihop loops

Node B

D	C	N
A	4	A
C	1	B

D	C	N
A	60	A
C	1	B

D	C	N
A	60	A
C	1	B

D	C	N
A	51	C
C	1	B

D	C	N
A	51	C
C	1	B

Node C

D	C	N
A	5	B
B	1	B

D	C	N
A	5	B
B	1	B

D	C	N
A	50	A
B	1	B

D	C	N
A	50	A
B	1	B

D	C	N
A	50	A
B	1	B

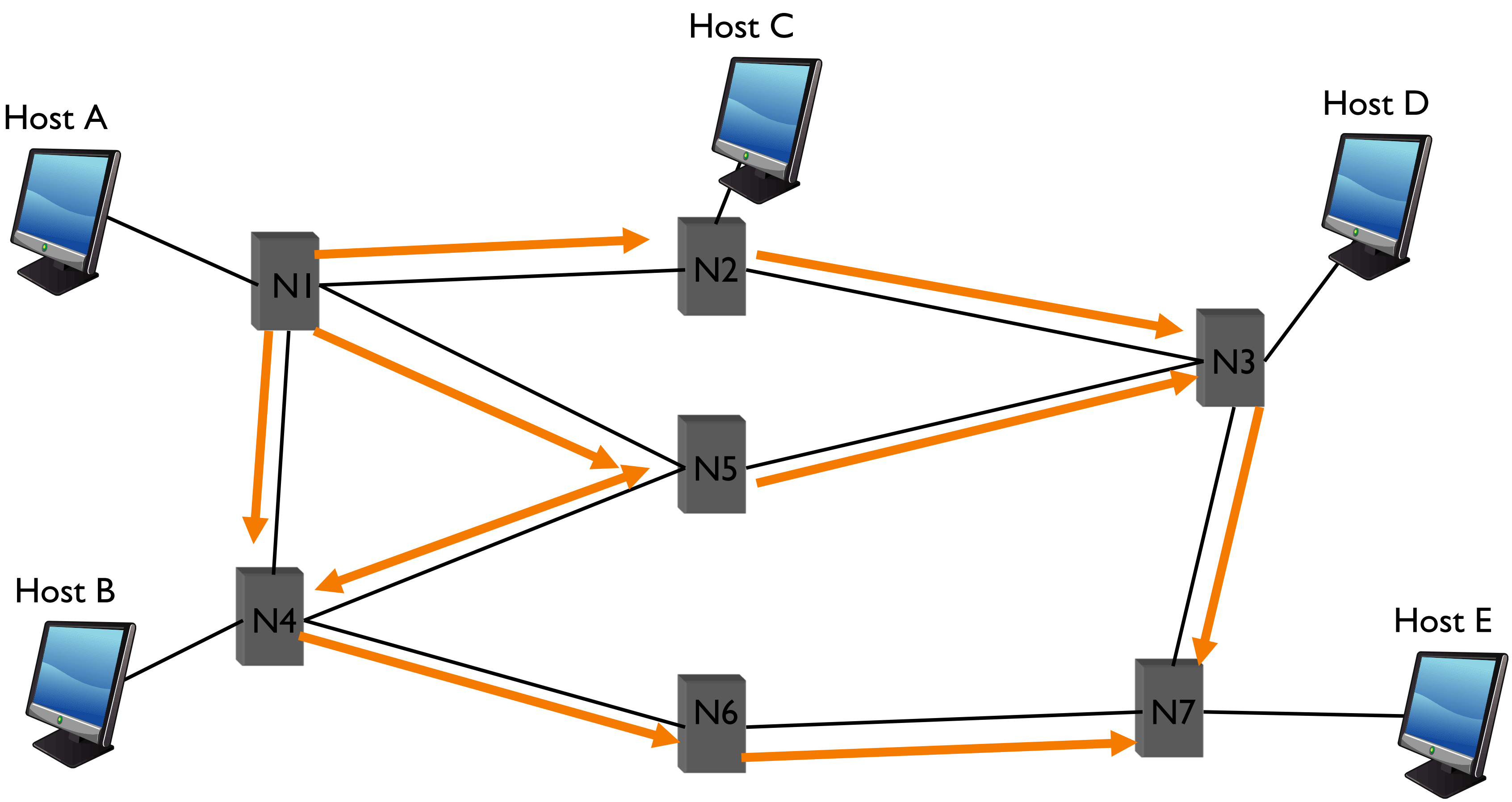
time

Link cost changes here; B updates D(B, A) = 60 as C has advertised D(C, A) = ∞

Algorithm terminates

Link State: Control Traffic

- Each node floods its local information to every other node in the network
- Each node ends up knowing the entire network topology → use Dijkstra to compute the shortest path to every other node



Link State: Node State