CS4450

Computer Networks:
Architecture and Protocols

Lecture 10
Spanning Tree Protocol
Fundamentals of Routing

Spring 2018
Rachit Agarwal

My Tuesday evening and Wednesday ….

- Wallowing in the shame of failure
  - I left you confused at the end of last lecture …
  - **I felt like I have failed, yet again, as a teacher …**
  - **I felt like my class must hate me, yet again …**

- Today's goals:
  - Redeem my esteem, or at least, try it …
  - **See if my students can love me (again?)**

My Tuesday evening and Wednesday ….

- First attempt
  - I (almost) emptied my queues :-)
    - **Answered all the emails**
    - **Updated the website (socket slides/code, PS2, …)**
    - **Things should (hopefully) be good until the spring break!**
- Problem Set 2 solutions released on Piazza
- Quiz solutions will be released by this weekend

Goals for Today's Lecture

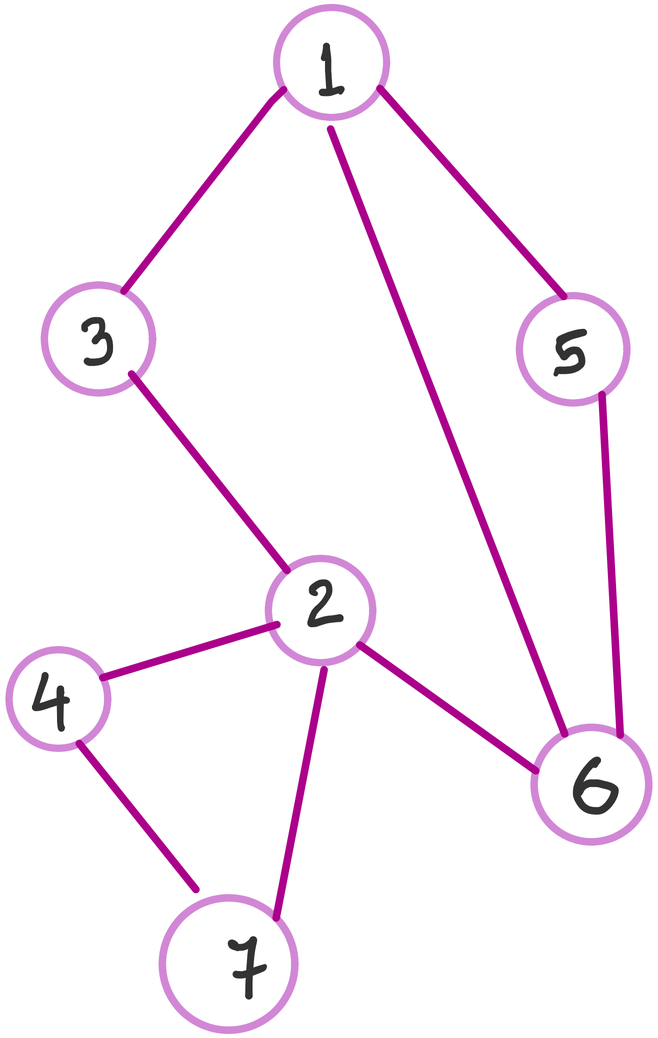
- **Bring us back into our love for computer networks (and me?) …**
- **Quick Review: Spanning Tree Protocol (+Failures)**
- Why do we need routing layer?
    - Why not just use spanning tree protocol?
- Start on Fundamentals of Routing

Recap: Spanning Tree Protocol (failures on later slides)

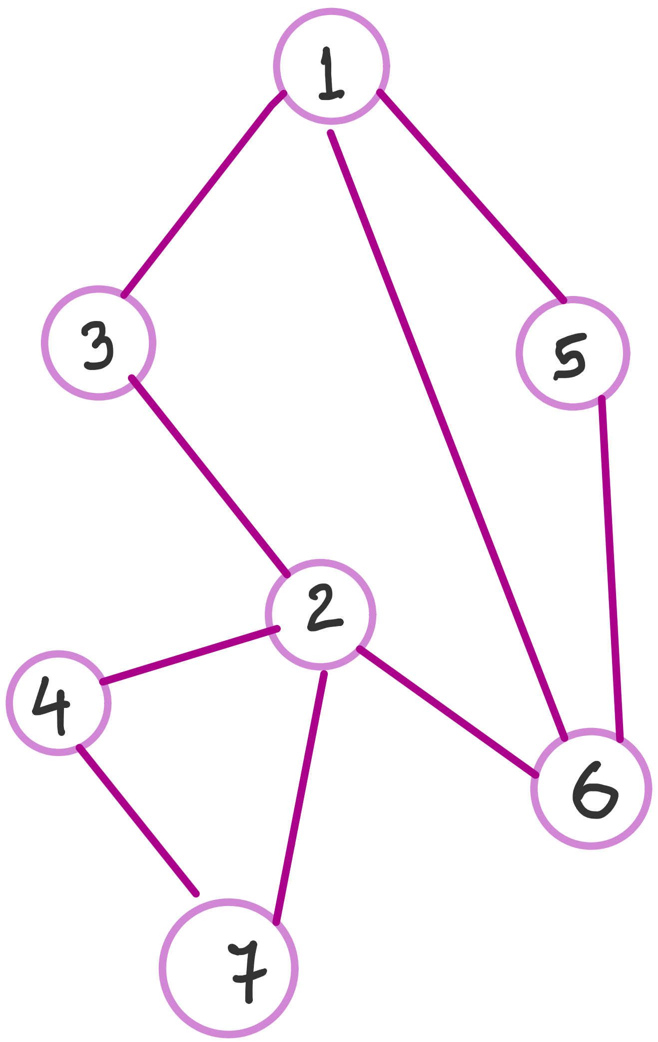
- Messages (Y,d,X): For root Y; From node X; advertising a distance d to Y
- Initially each switch X announces (X,0,X) to its neighbors
- Switches update their view
    - Upon receiving message (Y,d,Y) from Z, check Y's id
    - If Y's id < current root: set root = Y
- Switches compute their distance from the root
    - Add 1 to the shortest distance received from a neighbor
- If root changed OR shortest distance to the root changed, send all neighbors updated message (Y,d+1,X)

Group Exercise:

Lets run the Spanning Tree Protocol on this example

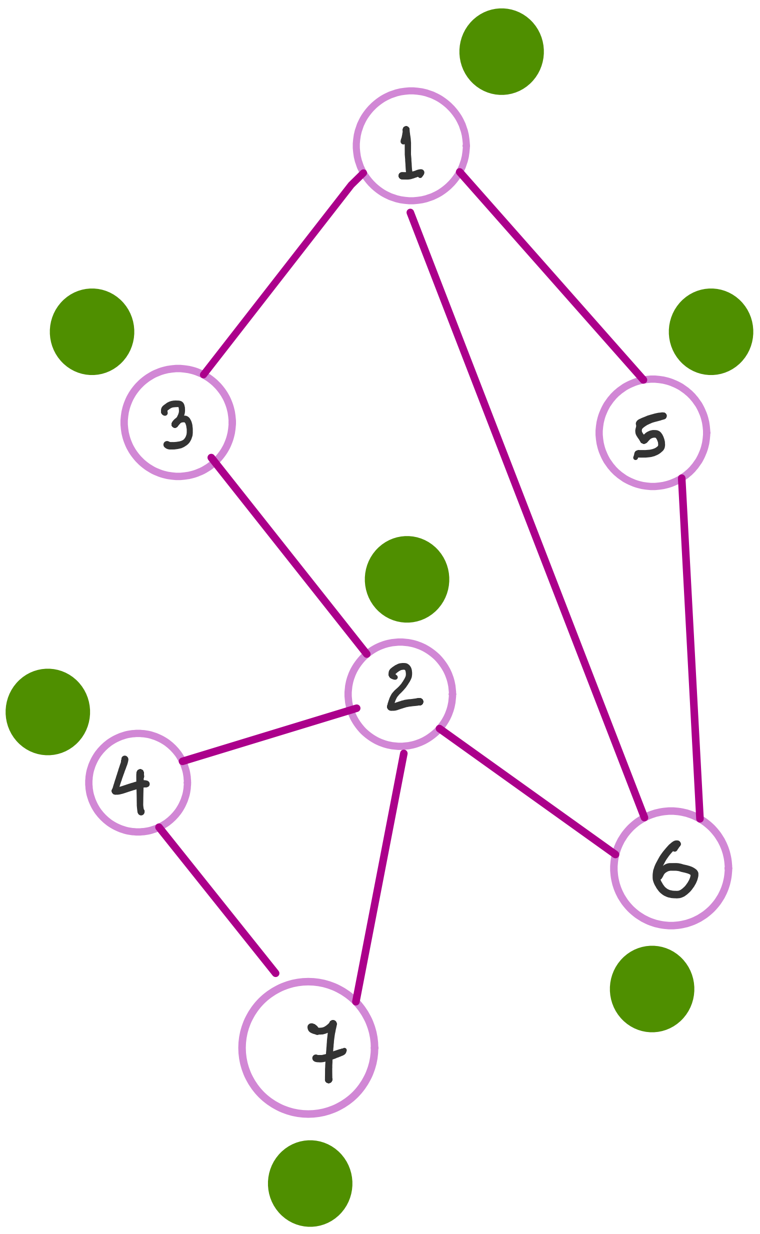


Round 1



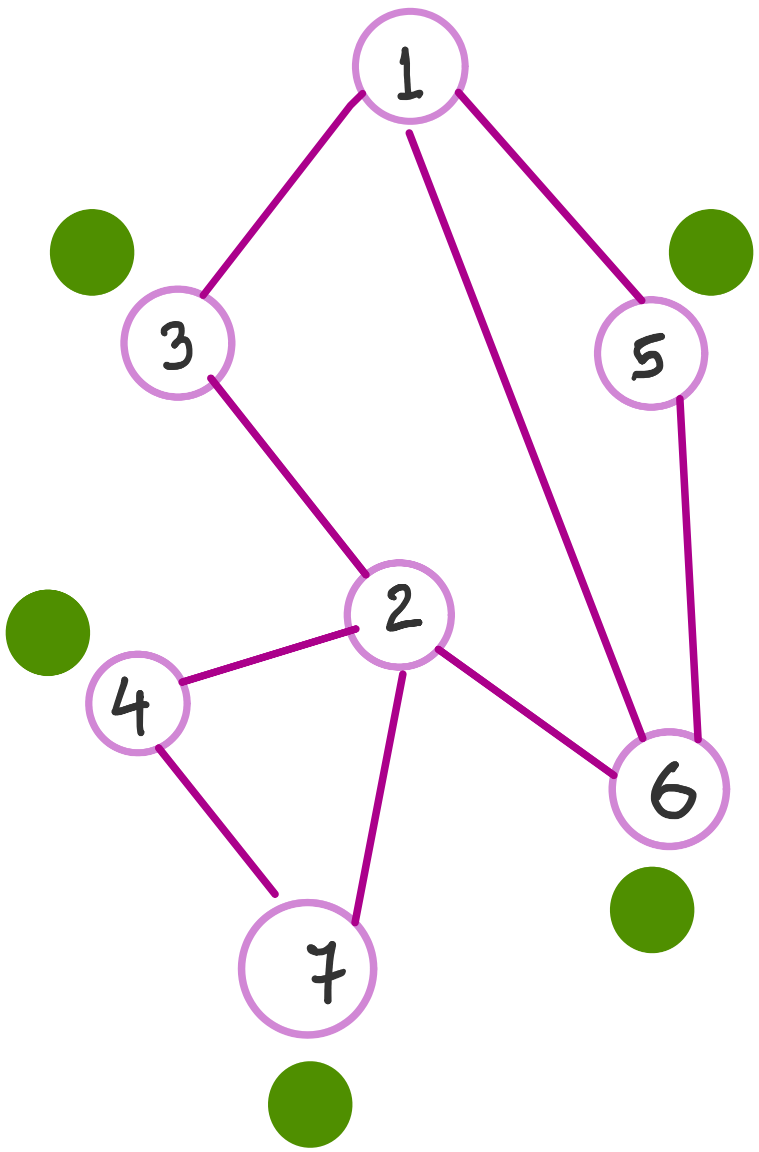
	Receive	Send
1		(1, 0, 1)
2		(2, 0, 2)
3		(3, 0, 3)
4		(4, 0, 4)
5		(5, 0, 5)
6		(6, 0, 6)
7		(7, 0, 7)

Round 2



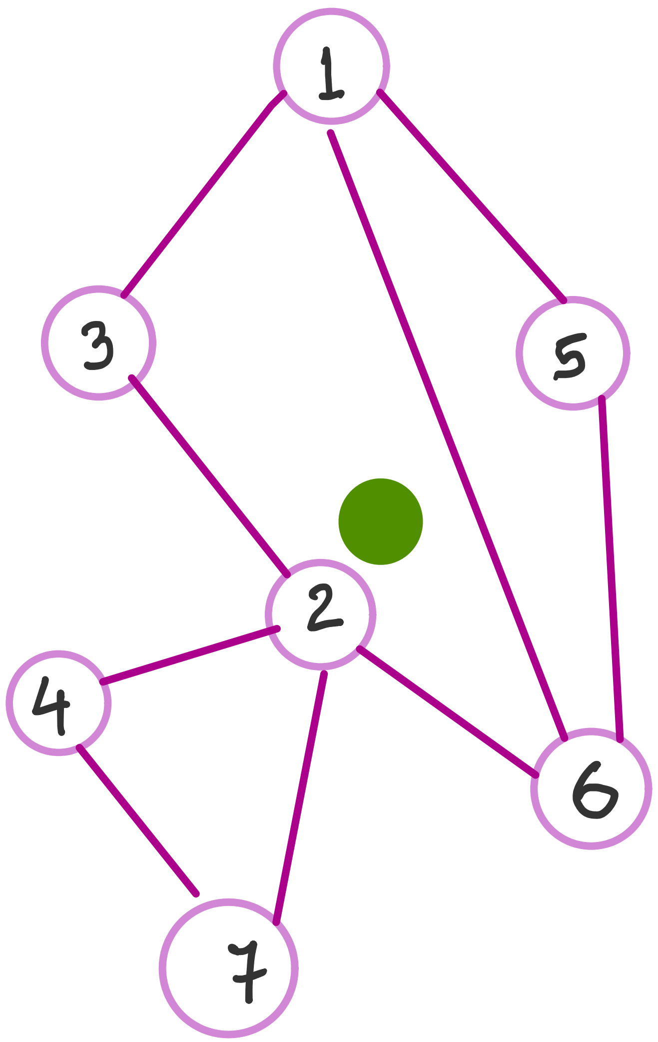
	Receive	Send
1 (1, 0, 1)	(3, 0, 3), (5, 0, 5), (6, 0, 6)	
2 (2, 0, 2)	(3, 0, 3), (4, 0, 4), (6, 0, 6), (7, 0, 7)	
3 (3, 0, 3)	(1, 0, 1), (2, 0, 2)	**(1, 1, 3)**
4 (4, 0, 4)	(2, 0, 2), (7, 0, 7)	**(2, 1, 4)**
5 (5, 0, 5)	(1, 0, 1), (6, 0, 6)	**(1, 1, 5)**
6 (6, 0, 6)	(1, 0, 1), (2, 0, 2), (5, 0, 5)	**(1, 1, 6)**
7 (7, 0, 7)	(2, 0, 2), (4, 0, 4)	**(2, 1, 7)**

Round 3



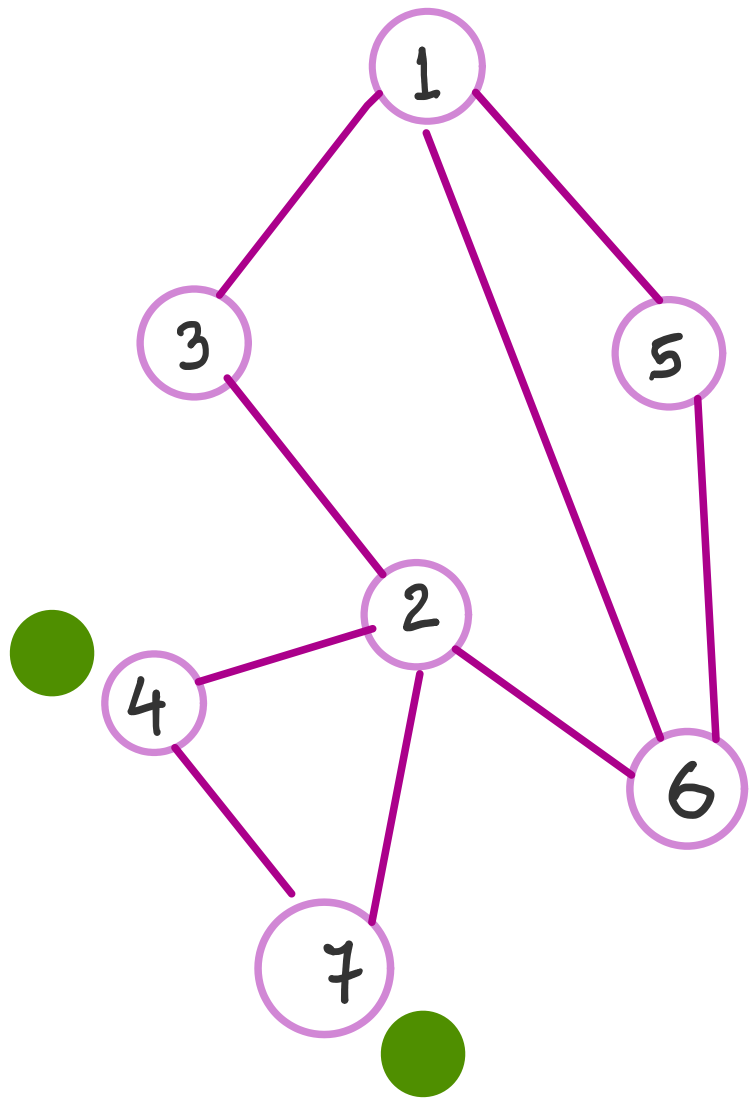
	Receive	Send
1 (1, 0, 1)	(1, 1, 3), (1, 1, 5), (1, 1, 6)	
2 (2, 0, 2)	(1, 1, 3), (2, 1, 4), (1, 1, 6), (2, 1, 7)	**(1, 2, 2)**
3 (1, 1, 3)		
4 (2, 1, 4)	(2, 1, 7)	
5 (1, 1, 5)	(1, 1, 6)	
6 (1, 1, 6)	(1, 1, 5)	
7 (2, 1, 7)	(2,1, 4)	

Round 4



	Receive	Send
1 (1, 0, 1)		
2 (1, 2, 2)		
3 (1, 1, 3)	(1, 2, 2)	
4 (2, 1, 4)	(1, 2, 2)	**(1, 3, 4)**
5 (1, 1, 5)		
6 (1, 1, 6)	(1, 2, 2)	
7 (2, 1, 7)	(1, 2, 2)	**(1, 3, 7)**

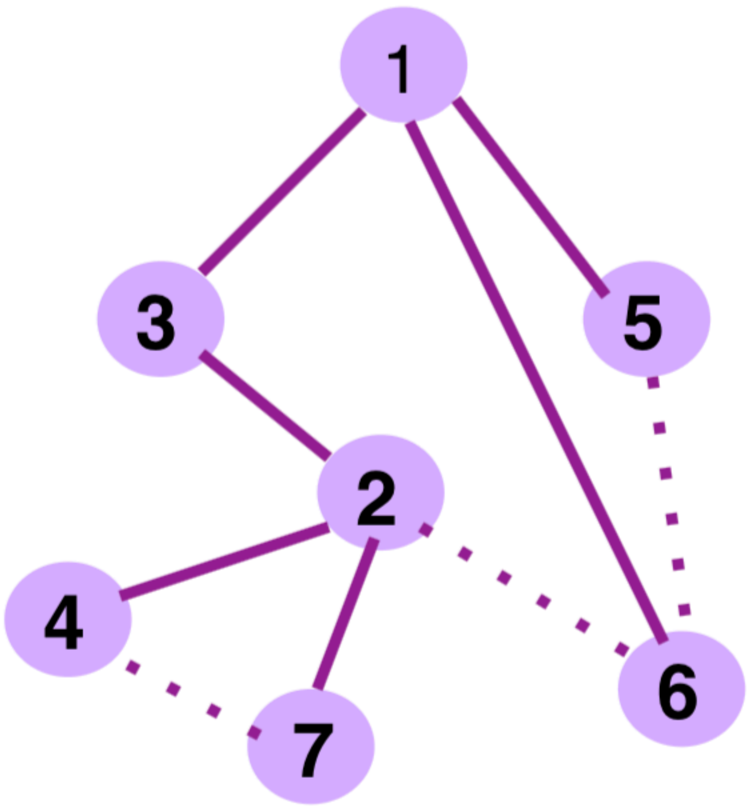
Round 5



	Receive	Send
1 (1, 0, 1)		
2 (1, 2, 2)	(1, 3, 4), (1, 3, 7)	
3 (1, 1, 3)		
4 (1, 3, 4)	(1, 3, 7)	
5 (1, 1, 5)		
6 (1, 1, 6)		
7 (1, 3, 7)	(1, 3, 4)	

After Round 5: We have our Spanning Tree

- 3-1
- 5-1
- 6-1
- 2-3
- 4-2
- 7-2



Questions?

Spanning Tree Protocol ++ (incorporating failures)

- Protocol must react to failures
  - Failure of the root node
  - Failure of switches and links

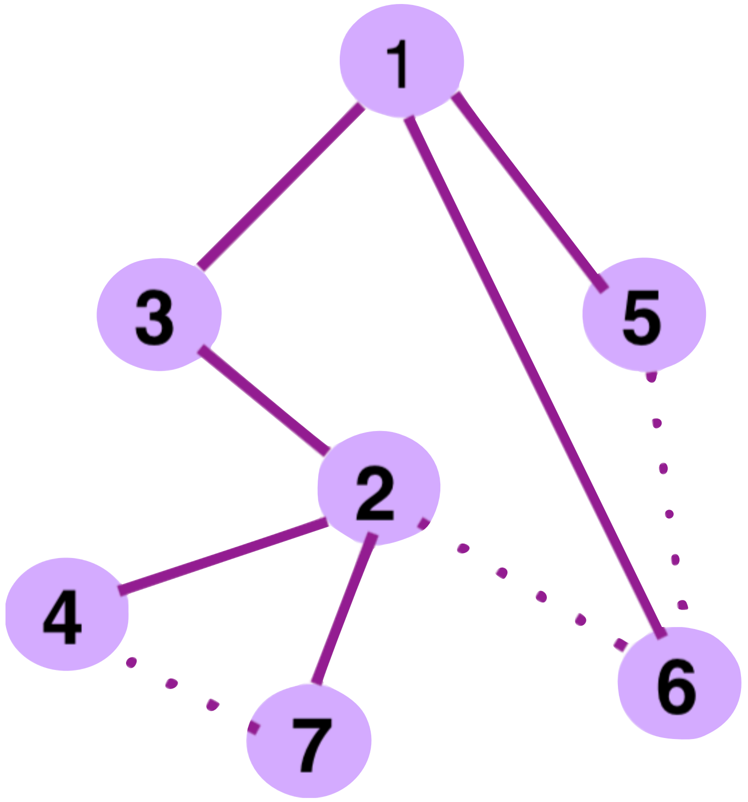
- **Root node sends periodic announcement messages**
  - Few possible implementations, but this is simple to understand
  - Other switches continue forwarding messages

- Detecting failures through timeout (soft state)
  - If no word from root, time out and send a (Y, 0, Y) message to all neighbors (in the graph)!

- **If multiple messages with a new root received, send message (Y, d, X) to the neighbor sending the message**

Suppose link 2-4 fails

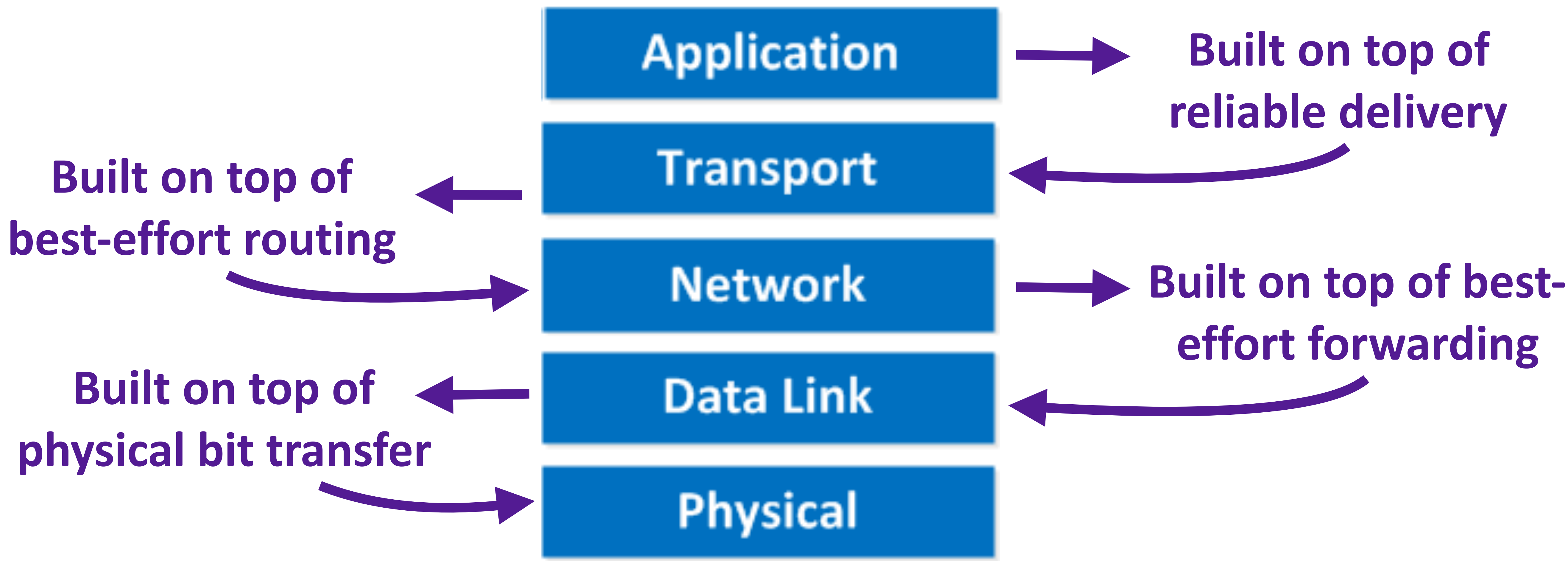
- 4 will send (4, 0, 4) to all its neighbors
  - 4 will stop receiving announcement messages from the root
  - Why?
- At some point, 7 will respond with (1, 3, 7)
- 4 will now update to (1, 4, 4) and send update message
- New spanning tree!



Questions?

The end of Link Layer ….

And the beginning of network layer :-D

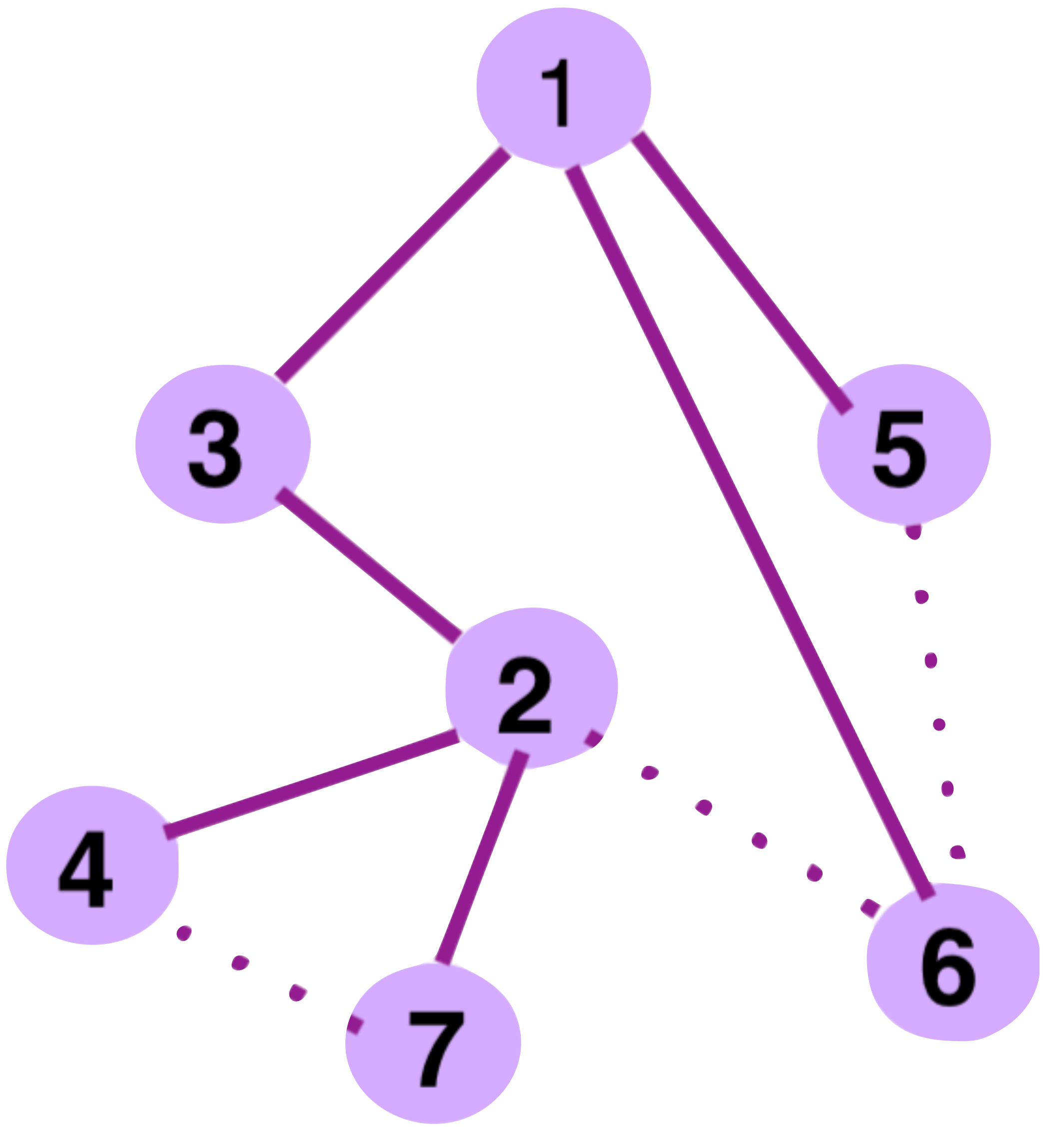


Why do we need a network layer?

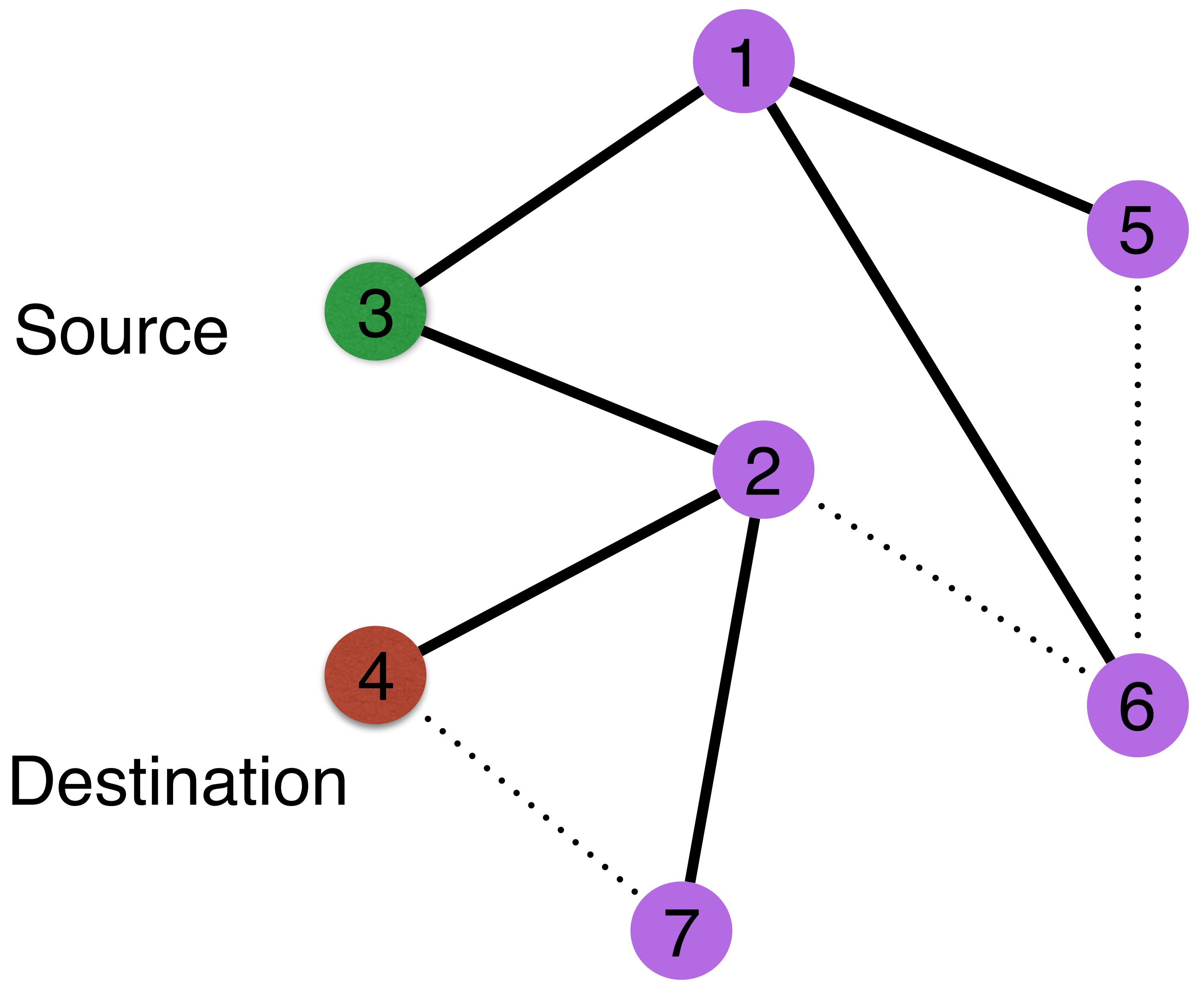
- There's only one path from source to destination
- How do you find that path? Ideas?
- Easy to design routing algorithms for trees
    - **Nodes can "flood" packet to all other nodes**

Flooding on a Spanning Tree

- Sends packet to ***every*** node in the network
- **Step 1**: Ignore the links not belonging to the Spanning Tree
- **Step 2**: Originating node sends "flood" packet out every link (on spanning tree)
- **Step 3**: Send incoming packet out to all links **other than the one that sent the packet**

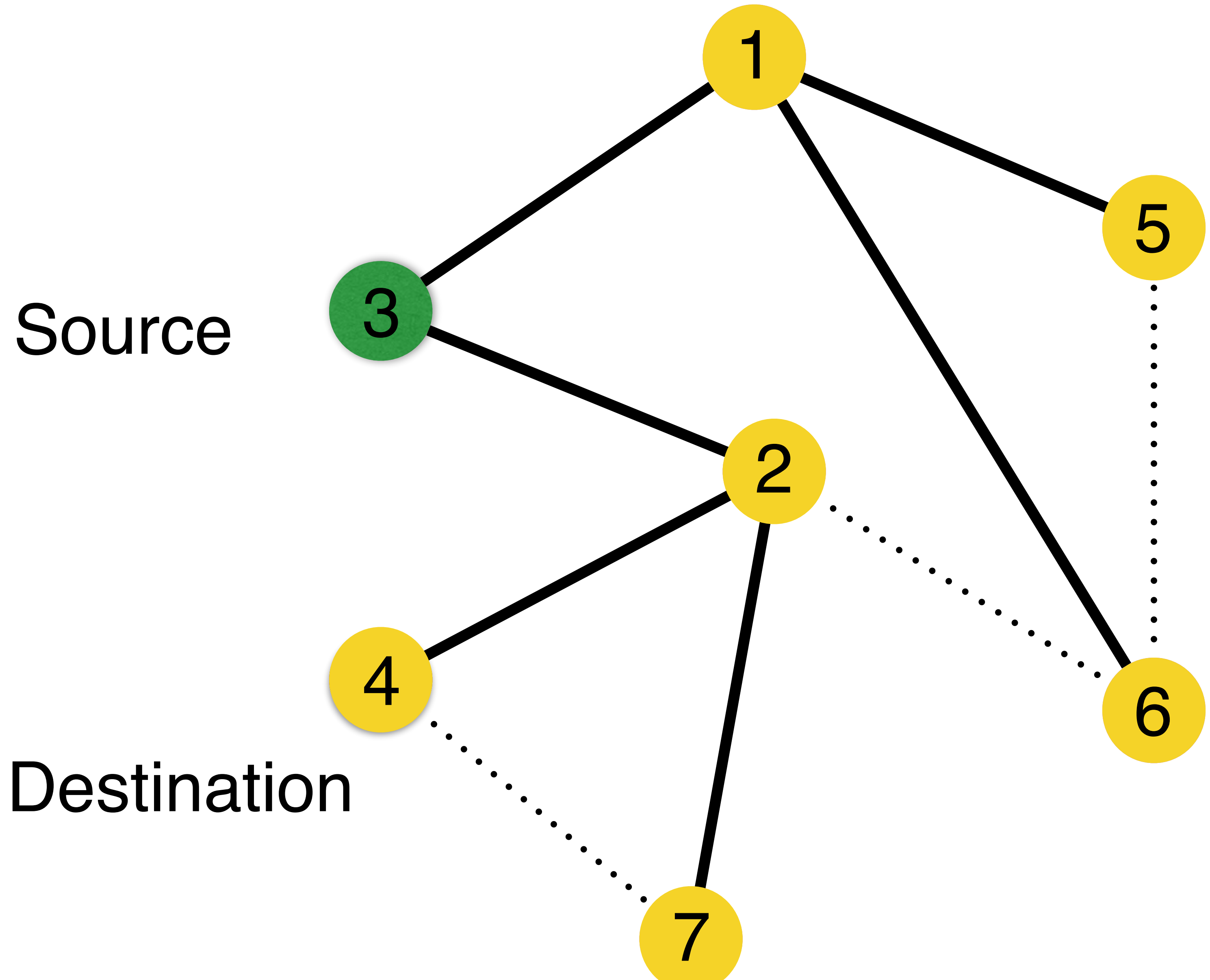


Flooding Example



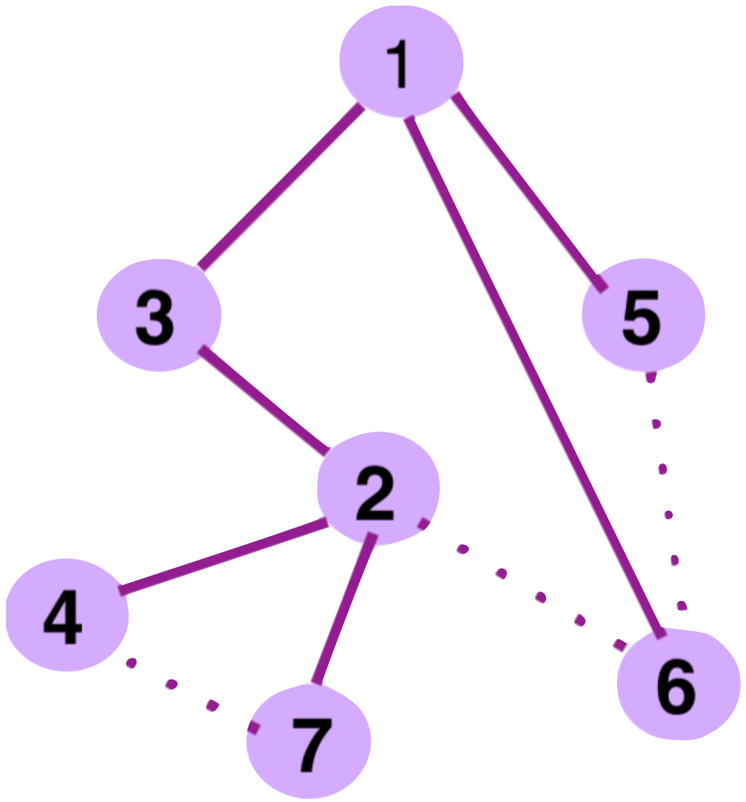
Flooding Example

Eventually all nodes are covered



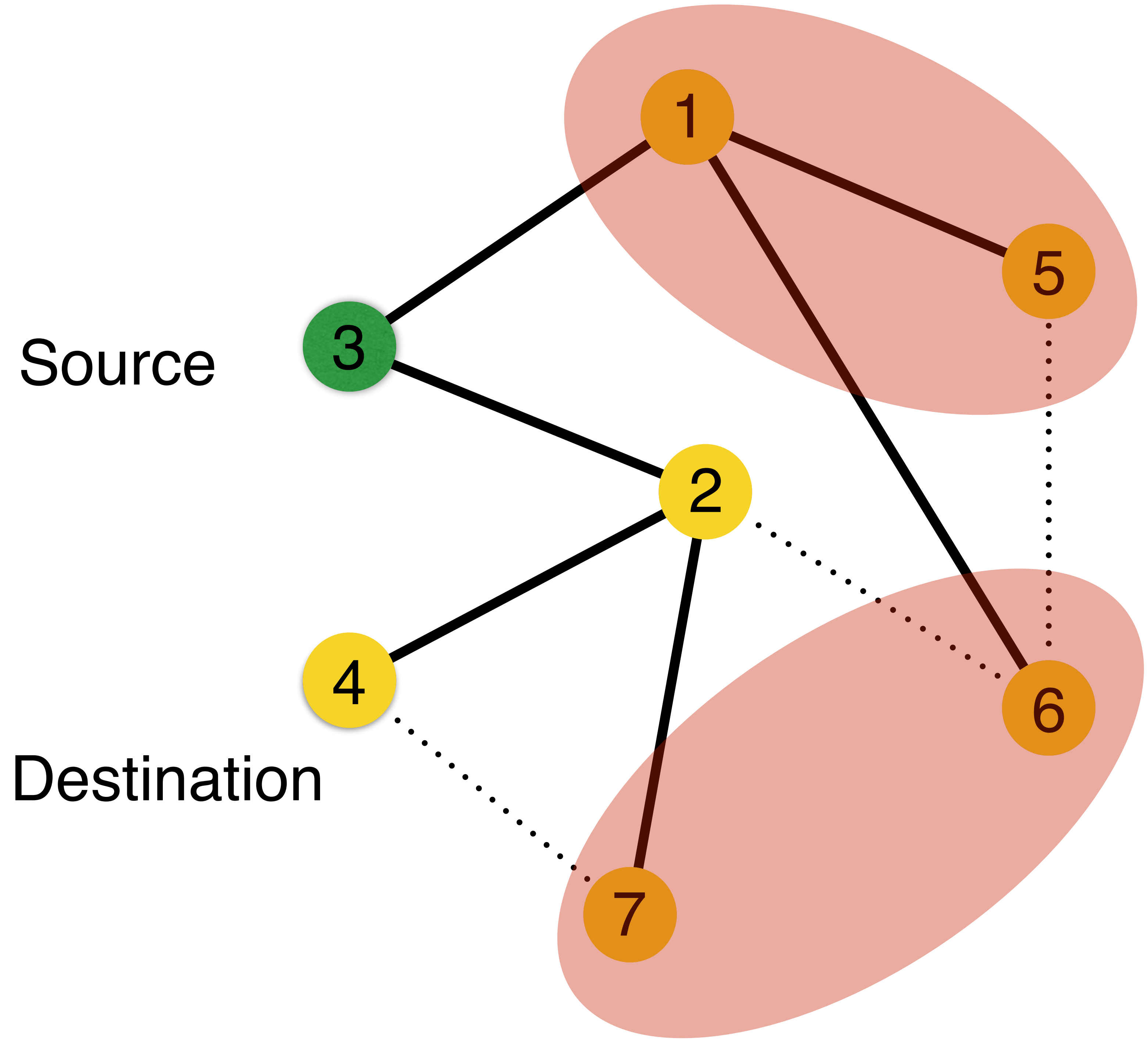
One copy of packet delivered to destination

Routing via Flooding on Spanning Tree …



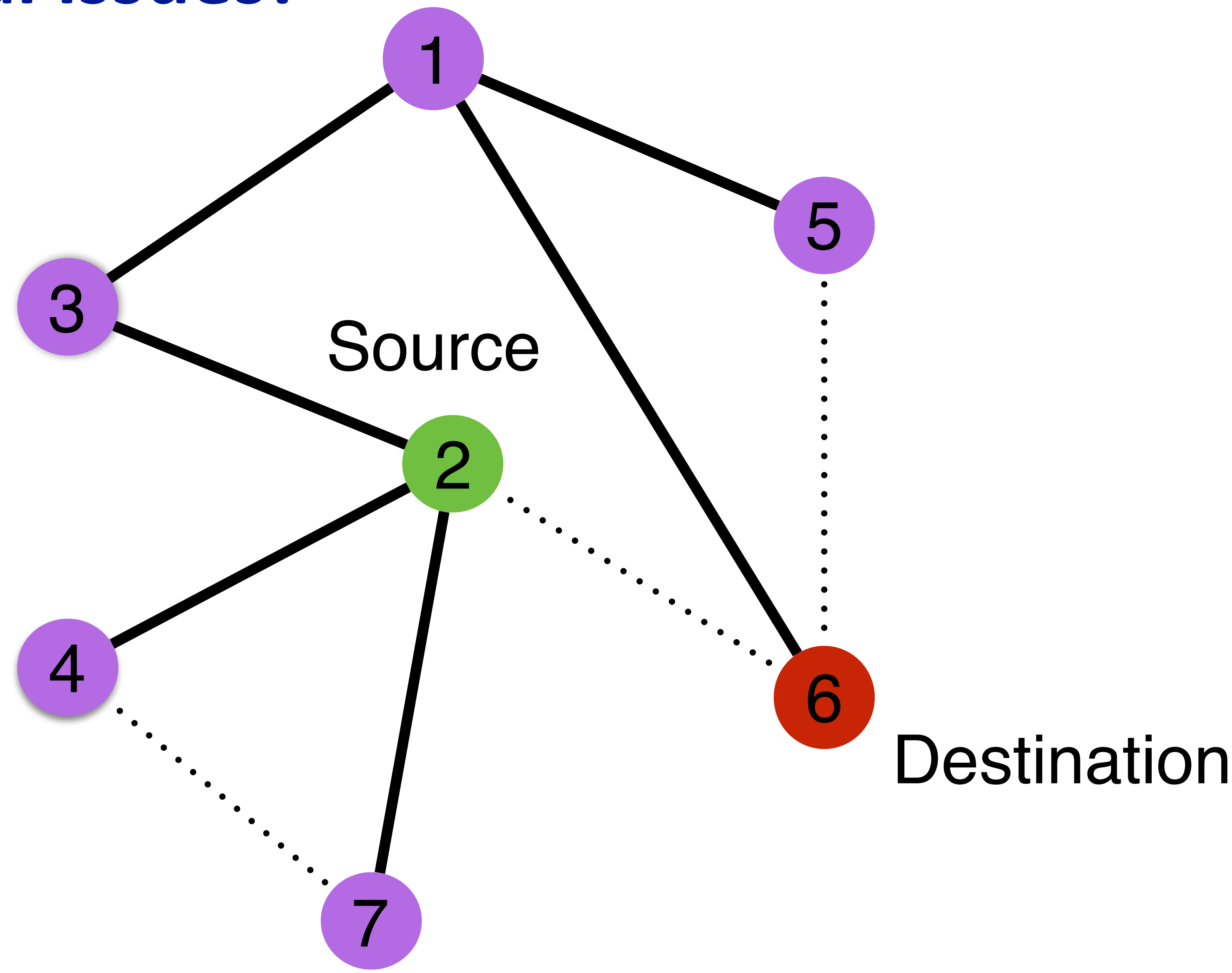
- There's only one path from source to destination
- How do you find that path? Ideas?
- Easy to design routing algorithms for trees
    - **Nodes can "flood" packet to all other nodes**
- Amazing properties:
    - No routing tables needed!
    - No packets will ever loop.
    - At least (and exactly) one packet must reach the destination
        - Assuming no failures

Three fundamental issues!



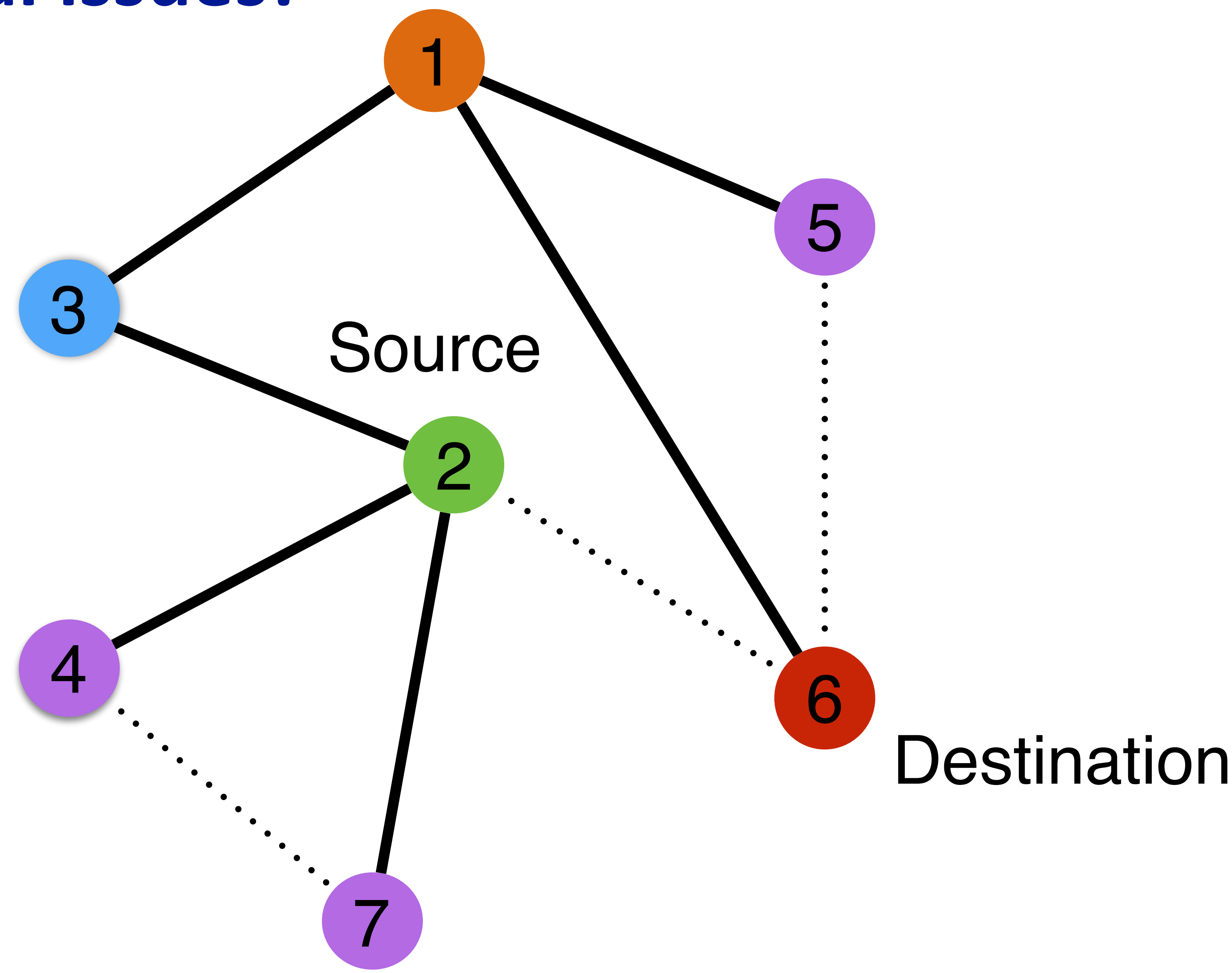
Issue 1: Each host has to do unnecessary packet processing!
(to decide whether the packet is destined to the host)

Three fundamental issues!



Issue 2: Higher latency!
(The packets unnecessarily traverse much longer paths)

Three fundamental issues!



Issue 2: Lower bandwidth availability!
(2-6 and 3-1 packets unnecessarily have to share bandwidth)

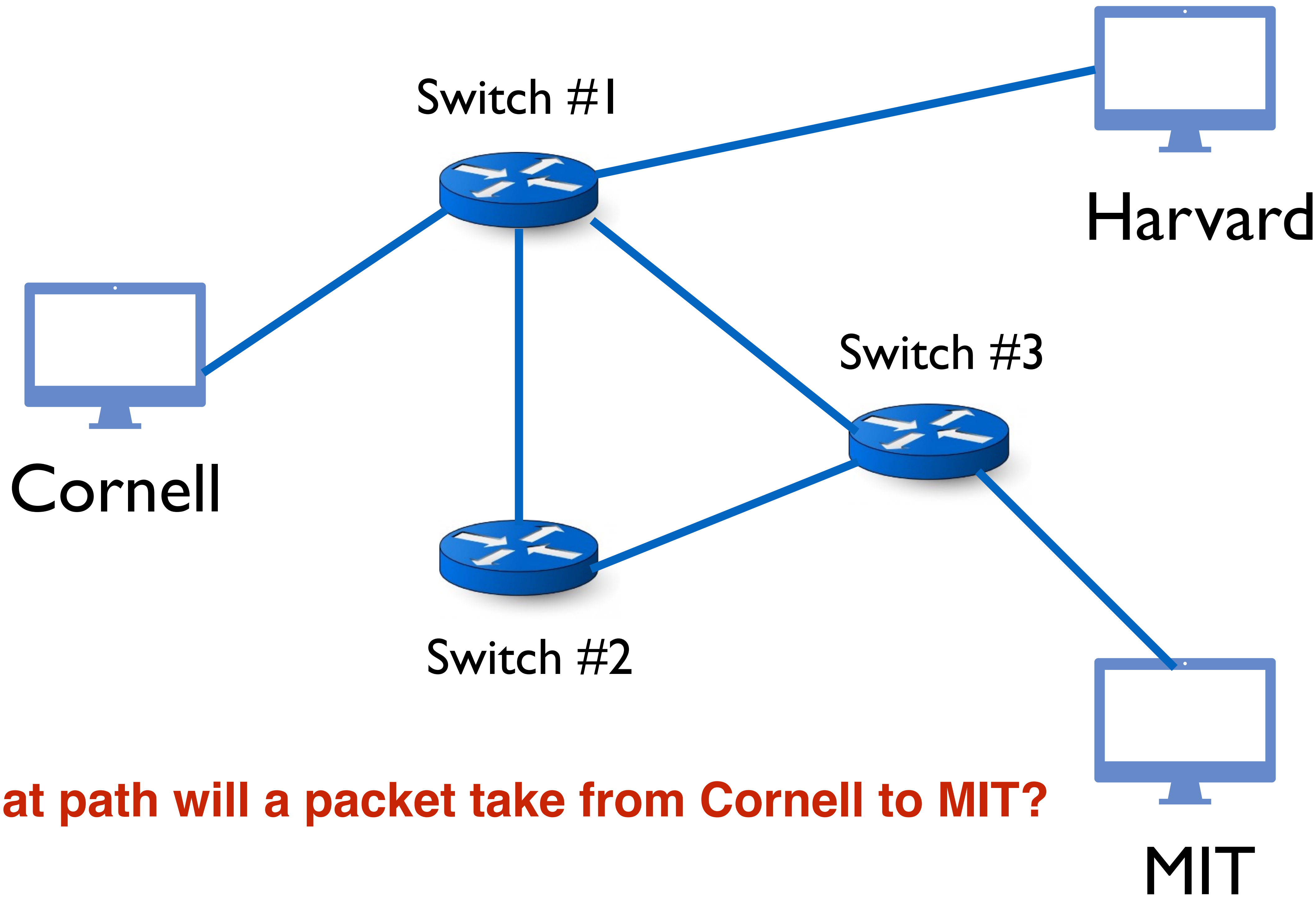
Questions?

Why do we need a network layer?

- Network layer performs “routing” of packets to alleviate these issues
- Uses routing tables
- Lets understand routing tables first
    - **We will see routing tables are nothing but …**
    - **Guess?**
    - **A collection of (carefully constructed) spanning trees**
        - **One per destination**

Routing Packets via Routing Tables

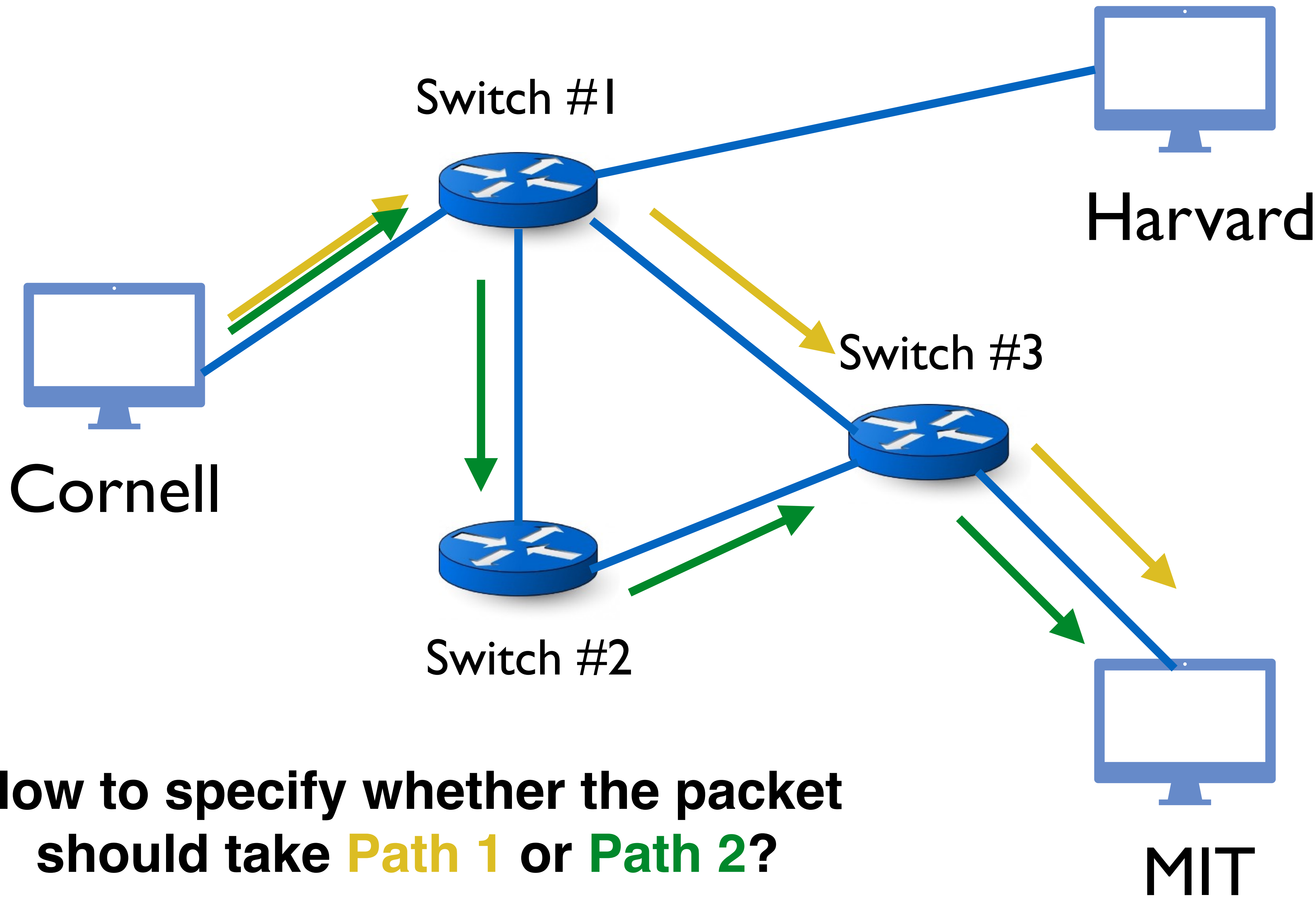
- **Routing tables allow finding path from source to destination**



What path will a packet take from Cornell to MIT?

Routing Packets via Routing Tables

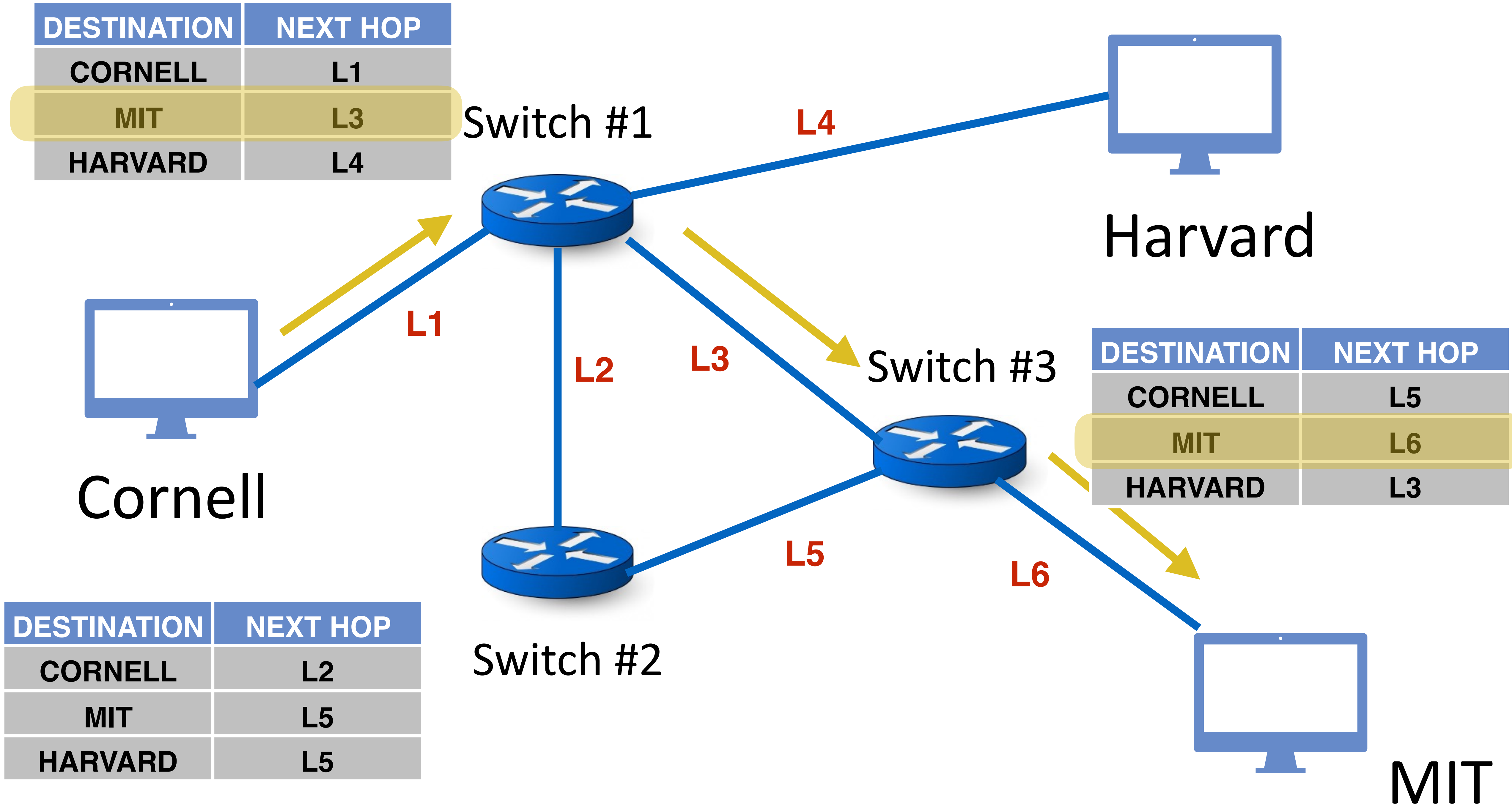
- **Finding path for a packet from source to destination**



How to specify whether the packet should take Path 1 or Path 2?

Routing Table

- Suppose packet follows Path 1: Cornell - S#1 - S#3 - MIT



DESTINATION	NEXT HOP
CORNELL	L1
MIT	L3
HARVARD	L4

DESTINATION	NEXT HOP
CORNELL	L5
MIT	L6
HARVARD	L3

DESTINATION	NEXT HOP
CORNELL	L2
MIT	L5
HARVARD	L5

Each Switch stores a table indicating the next hop for corresponding destination of a packet (called a routing table)

“Valid Routing Tables” (routing state)

- Global routing state is valid if:
  - it **always** results in deliver packets to their destinations

- **Goal of Routing Protocols**
  - Compute a valid state
  - But how to tell if a routing state is valid?…
  - Think about it, what could make routing incorrect?

Validity of a Routing State

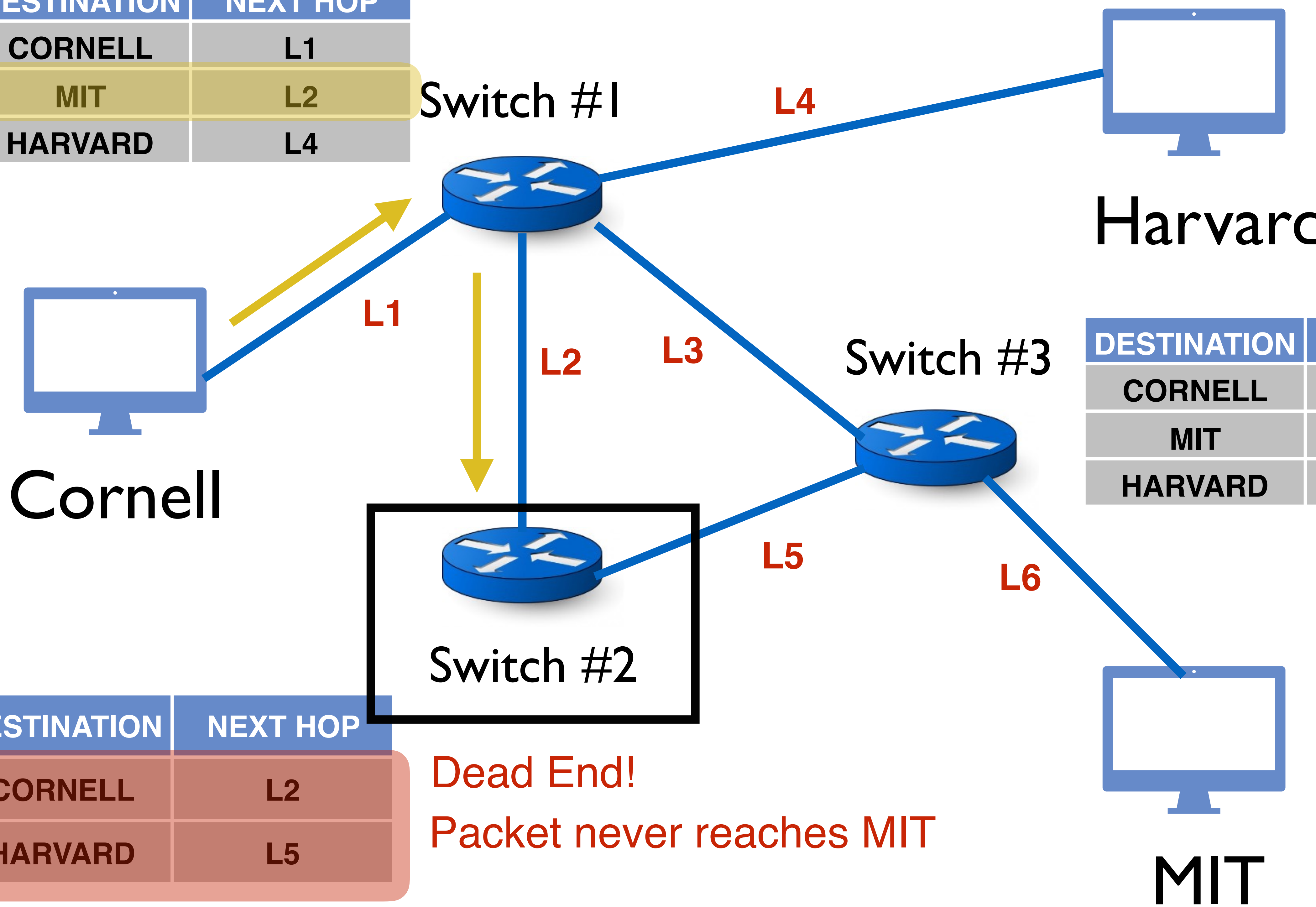
- Global routing state valid **if and only if**:
    - There are no **dead ends** (other than destination)
    - There are no **loops**

- A **dead end** is when there is **no outgoing link**
    - A packet arrives, but ..
        - the routing table does not have an outgoing link
        - And that node is not the destination

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

Example: Routing with Dead Ends

- Suppose packet wants to go from Cornell to MIT using given state:



Switch #1:

DESTINATION	NEXT HOP
CORNELL	L1
MIT	L2
HARVARD	L4

Switch #3:

DESTINATION	NEXT HOP
CORNELL	L5
MIT	L6
HARVARD	L3

Switch #2:

DESTINATION	NEXT HOP
CORNELL	L2
HARVARD	L5

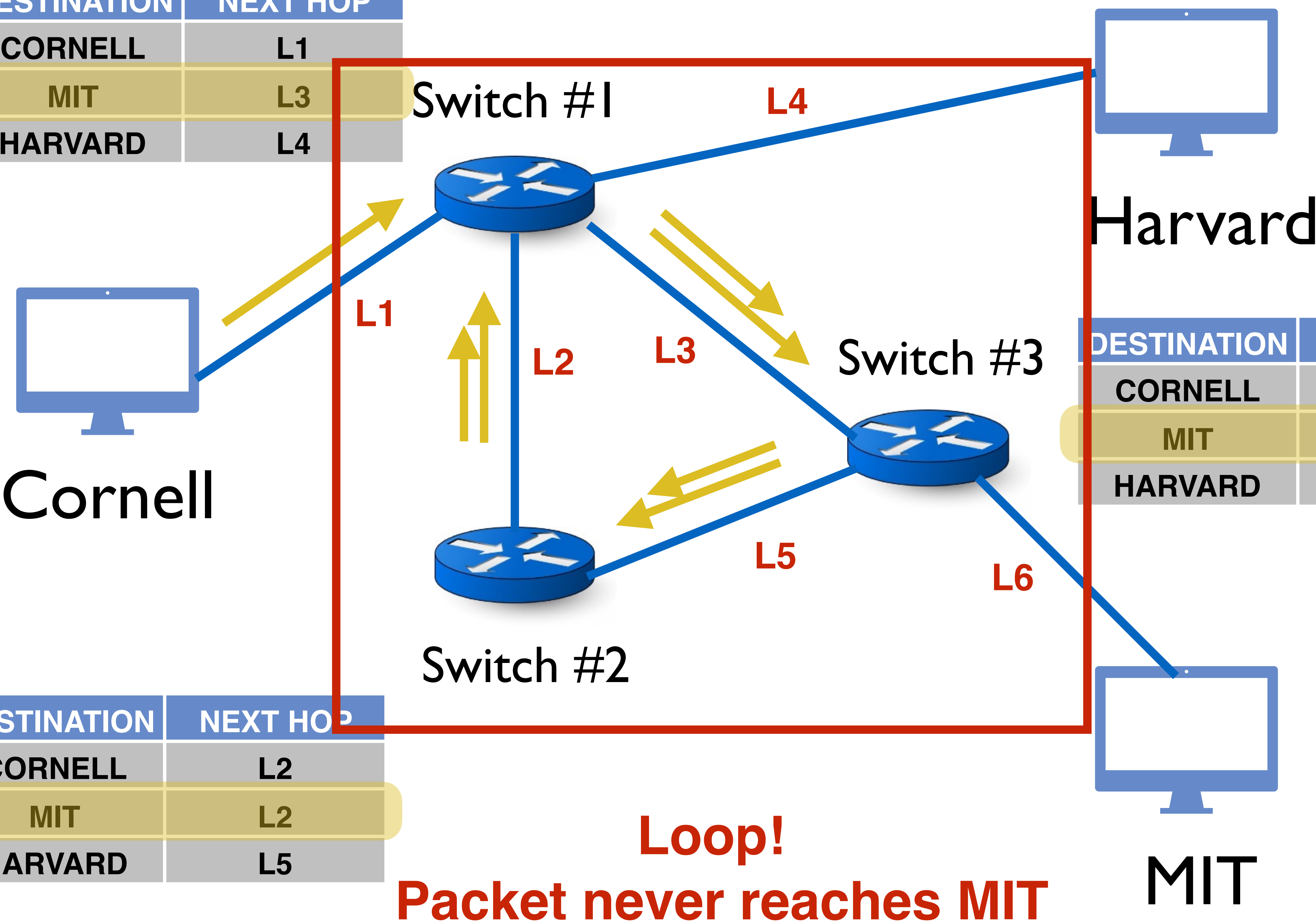
Dead End!
Packet never reaches MIT

No forwarding decision for MIT!

Example: Routing with Loops

- Suppose packet wants to go from Cornell to MIT using given state:

DESTINATION	NEXT HOP
CORNELL	L1
MIT	L3
HARVARD	L4



DESTINATION	NEXT HOP
CORNELL	L5
MIT	L5
HARVARD	L3

DESTINATION	NEXT HOP
CORNELL	L2
MIT	L2
HARVARD	L5

Loop!

Packet never reaches MIT

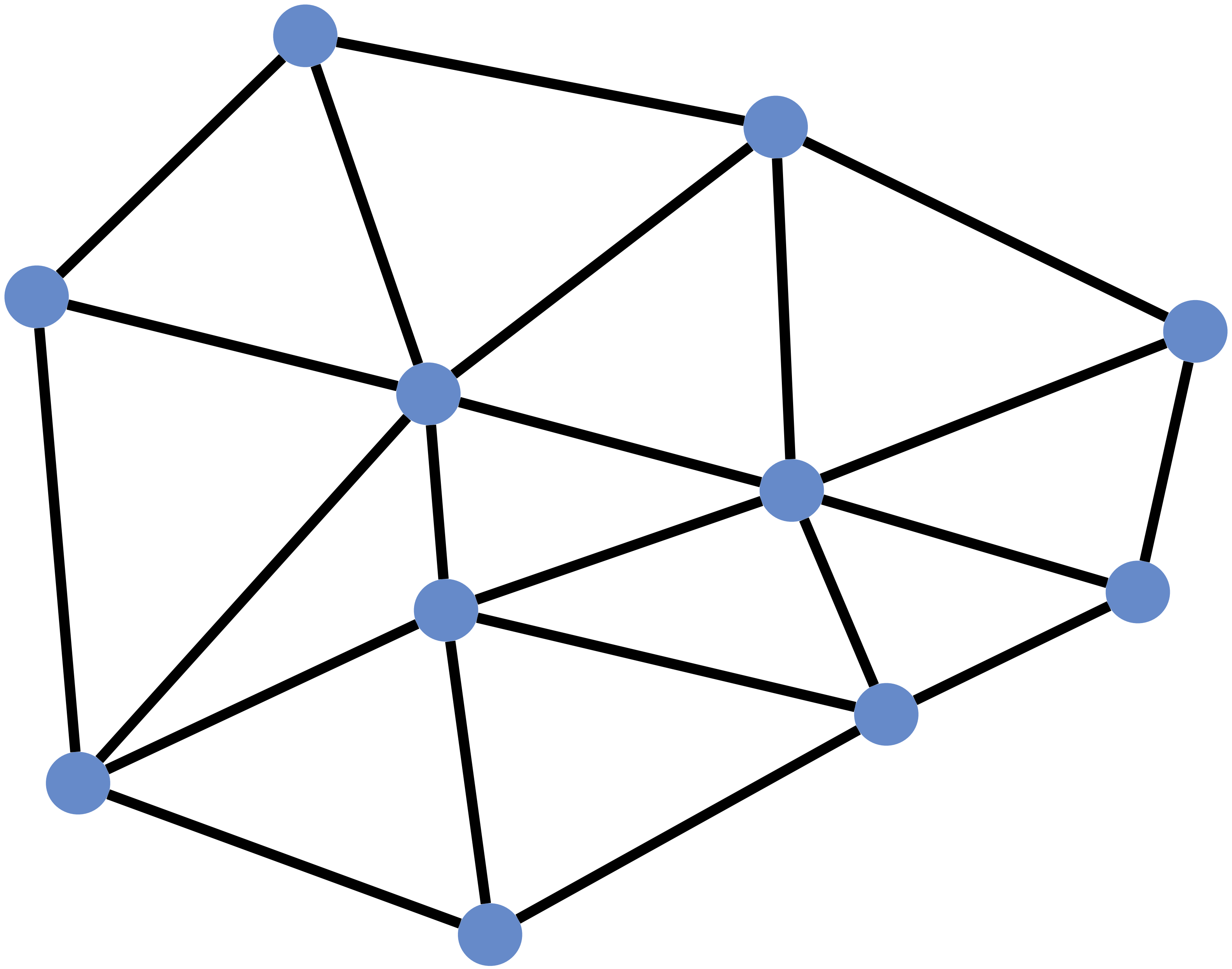
Two Questions

- How can we **verify** given routing state is valid?
- How can we **produce** valid routing state?

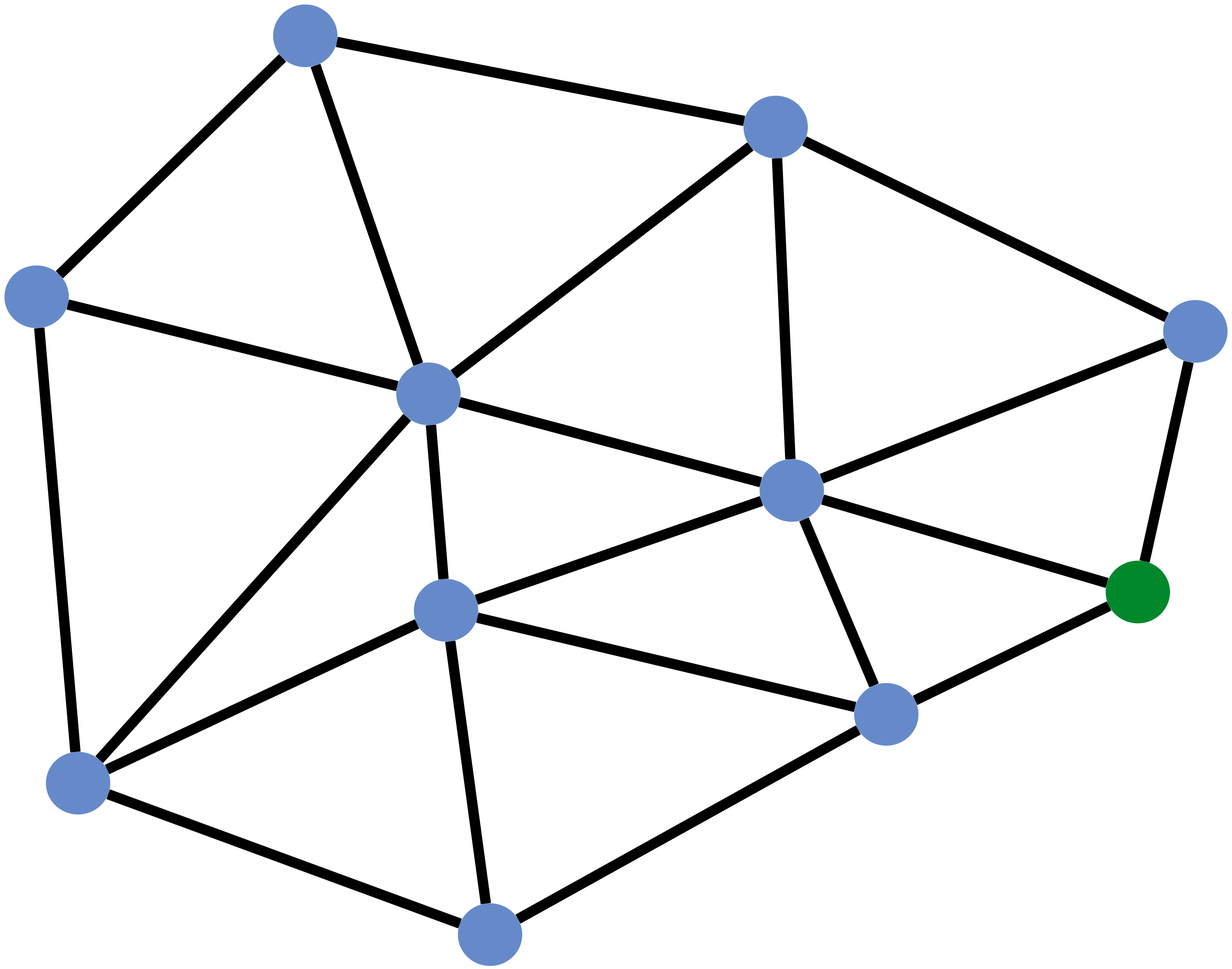
Checking Validity of a Routing State

- Check validity of routing state for one destination at a time…
- For each node:
  - Mark the outgoing link with arrow for the required destination
  - There can only be one at each node
- Eliminate all links with no arrows
- Look what's left. **State is valid if and only if**
  - Remaining graph is a spanning tree with destination as sink
  - Why is this true?
    - Tree -> No loops
    - Spanning (tree) -> No dead ends

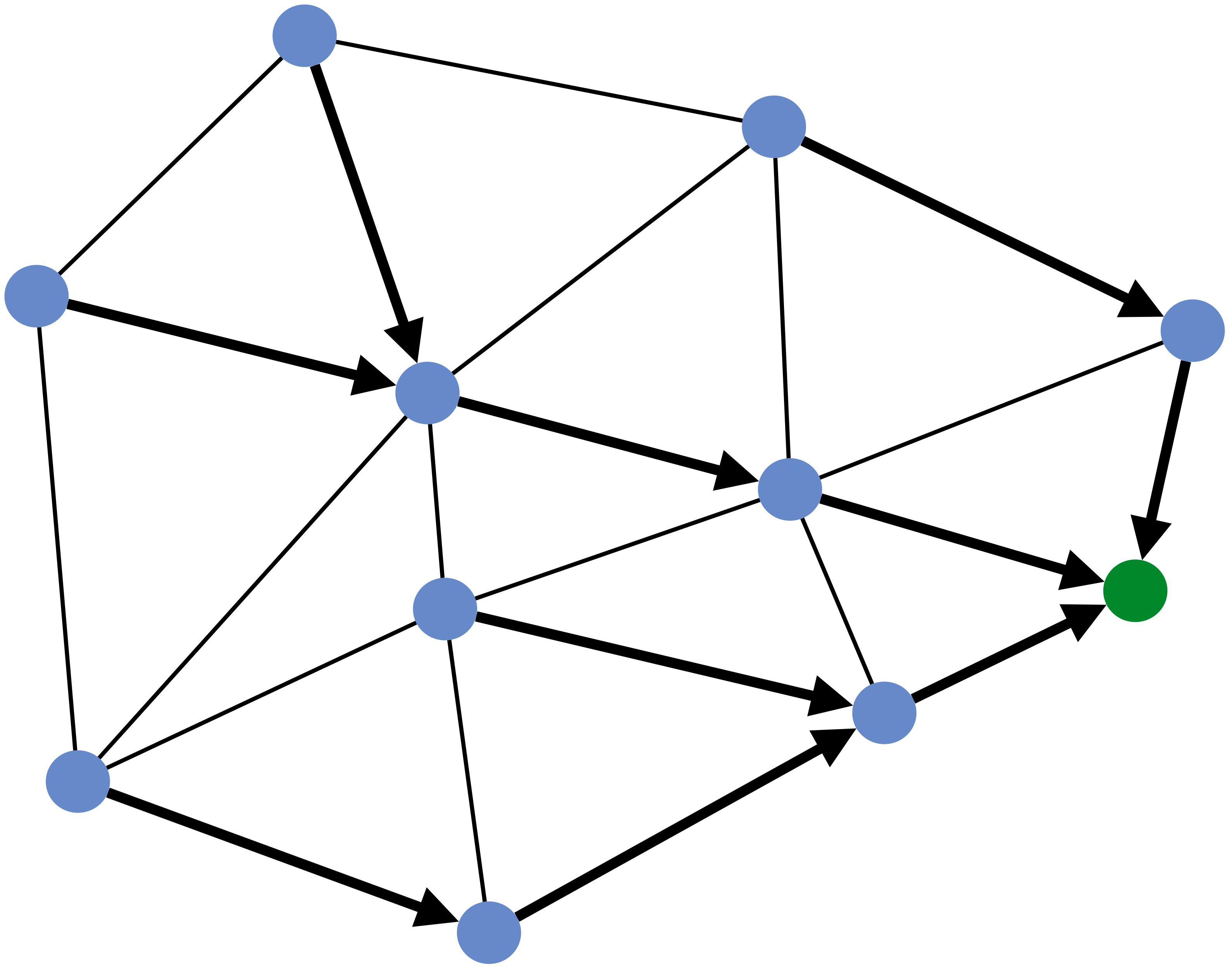
Example 1



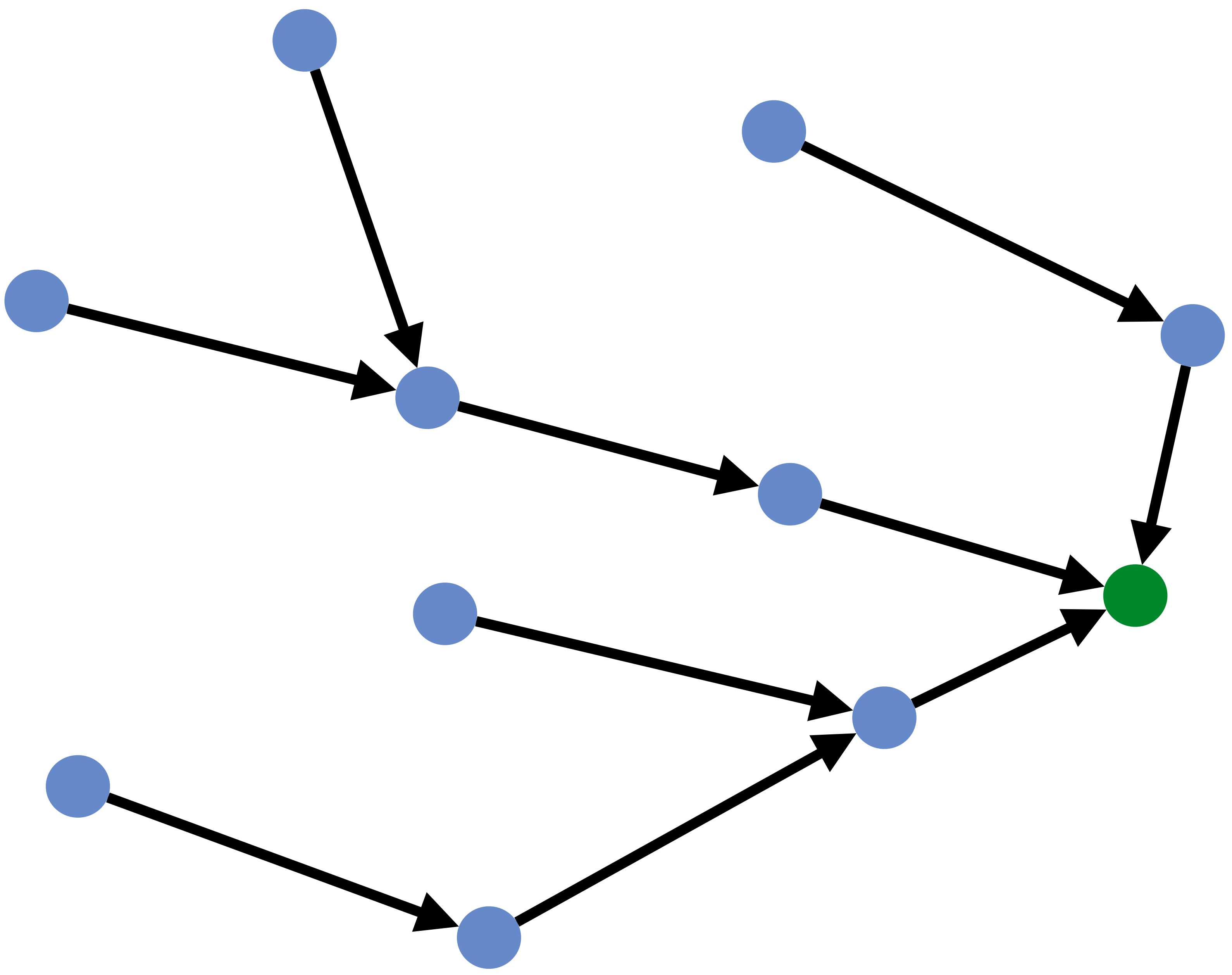
Example 1: Pick Destination



Example 1:Put Arrows on Outgoing Ports

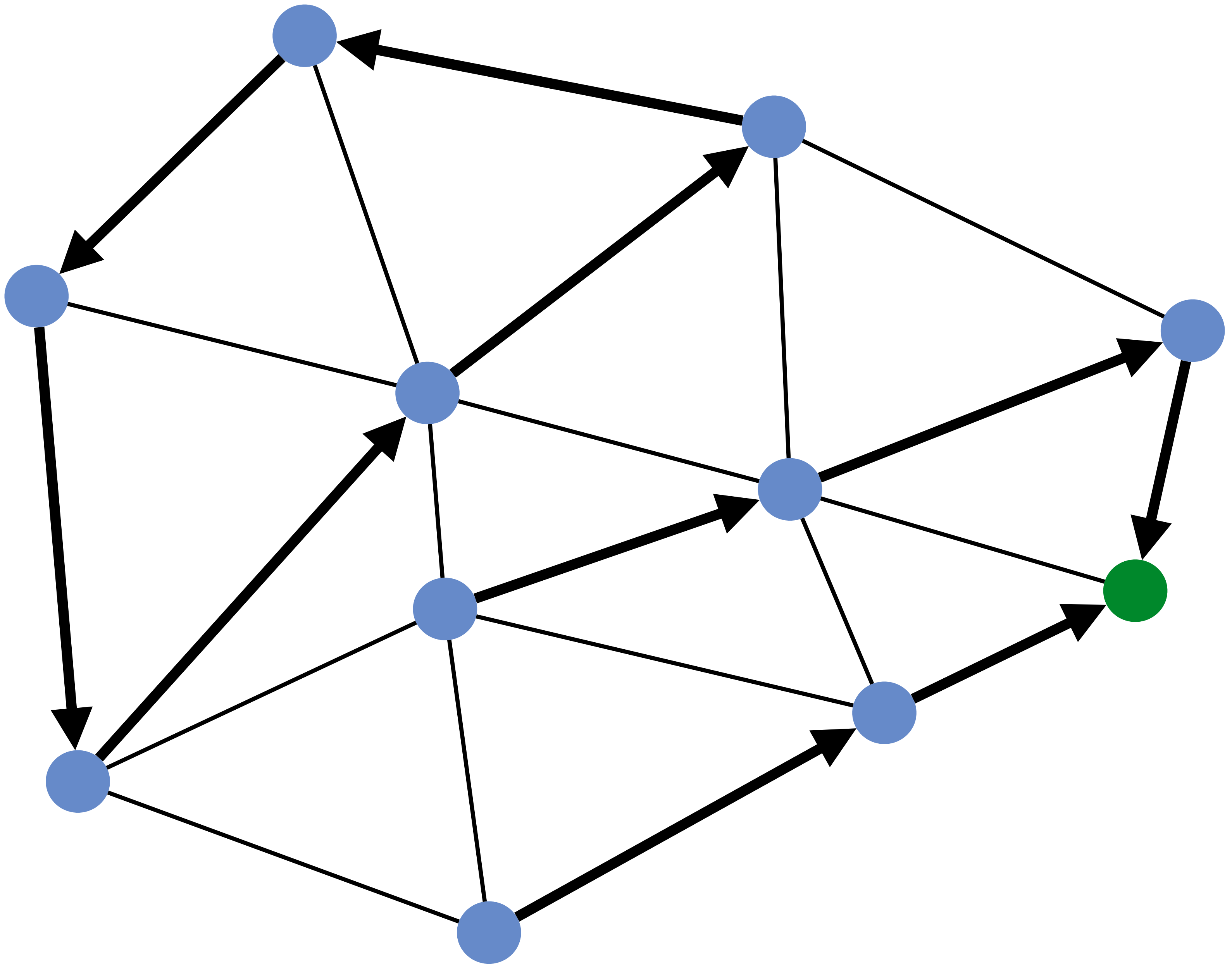


Example 1:Remove unused Links

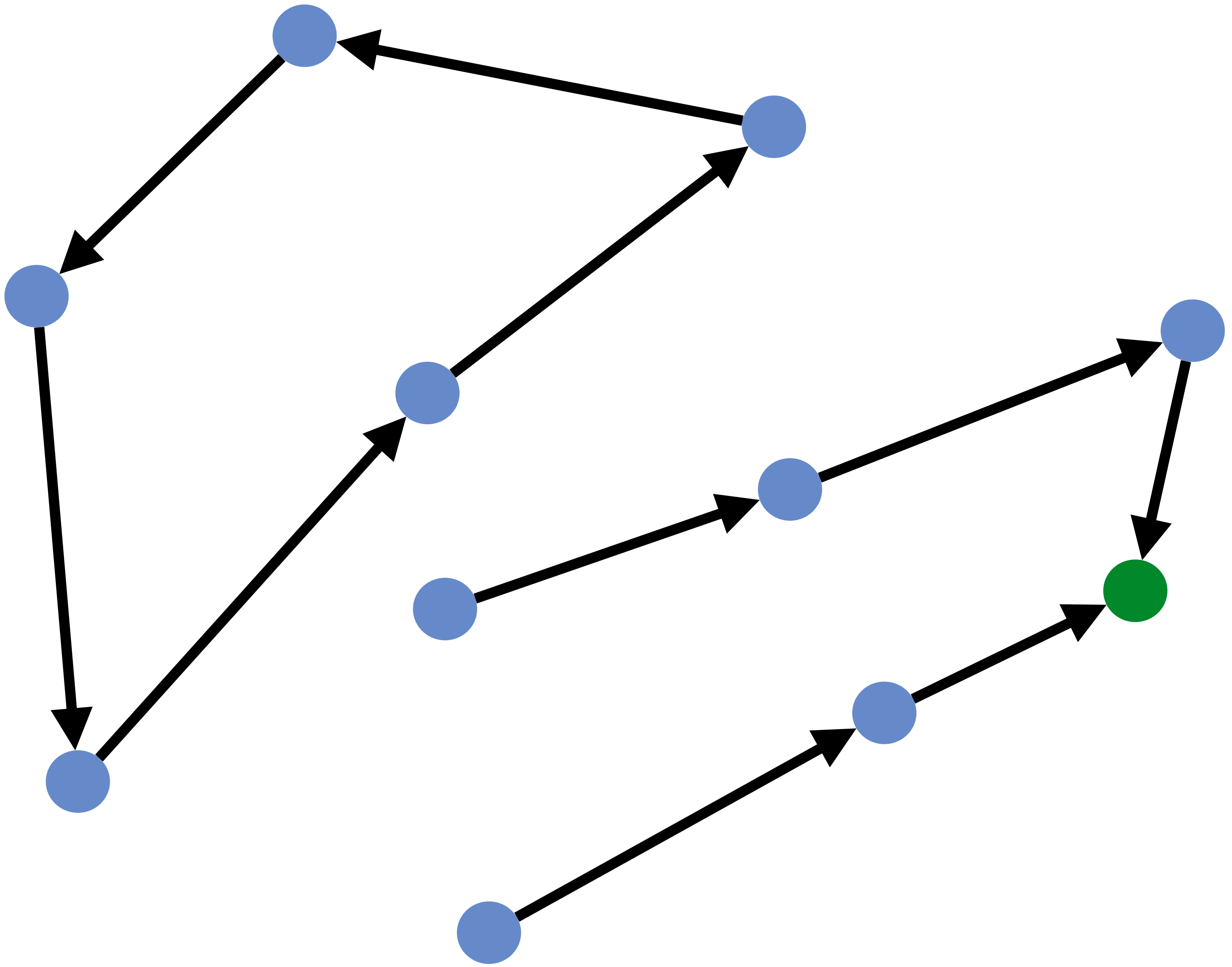


Leaves Spanning Tree: Valid

Example 2:

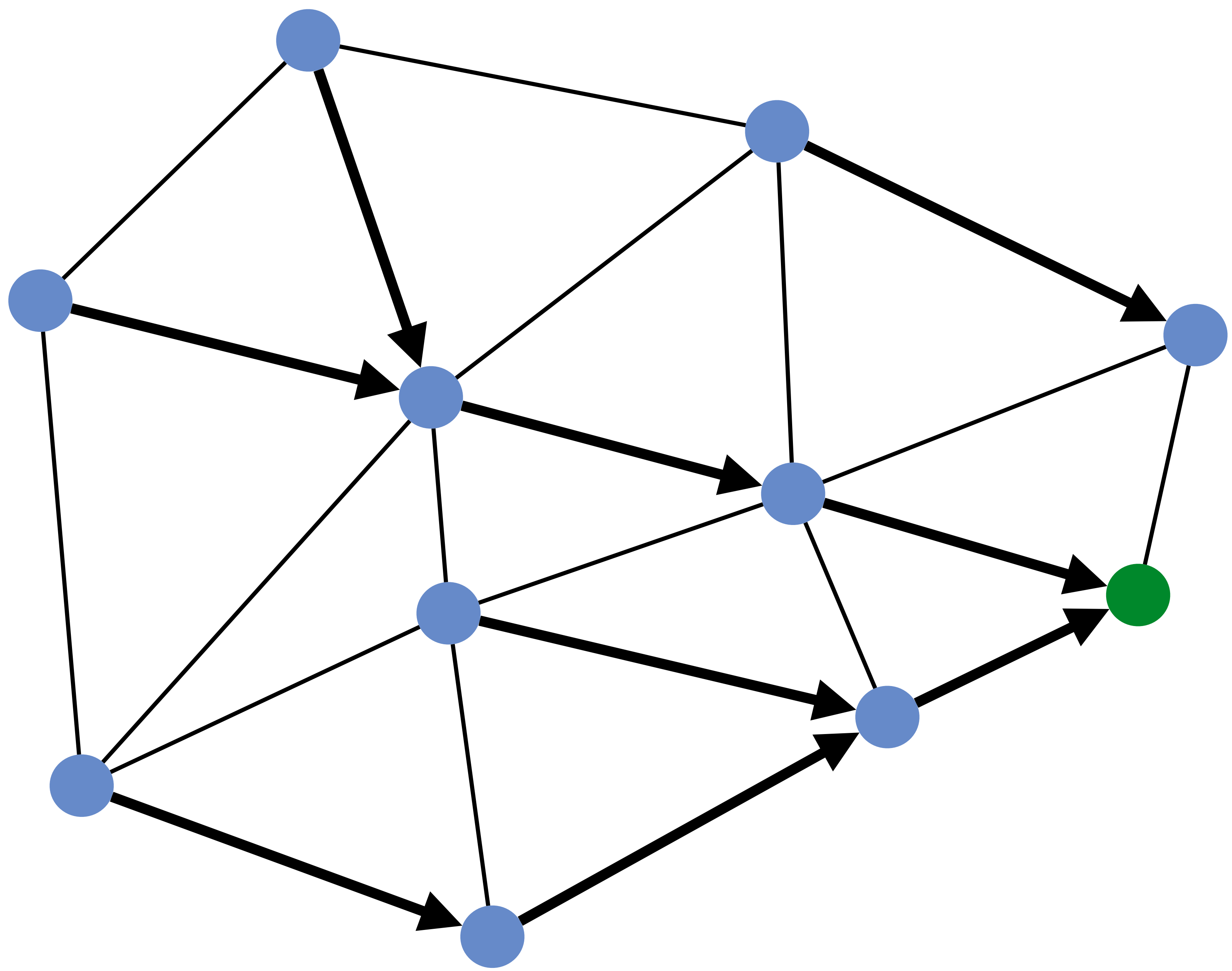


Example 2:

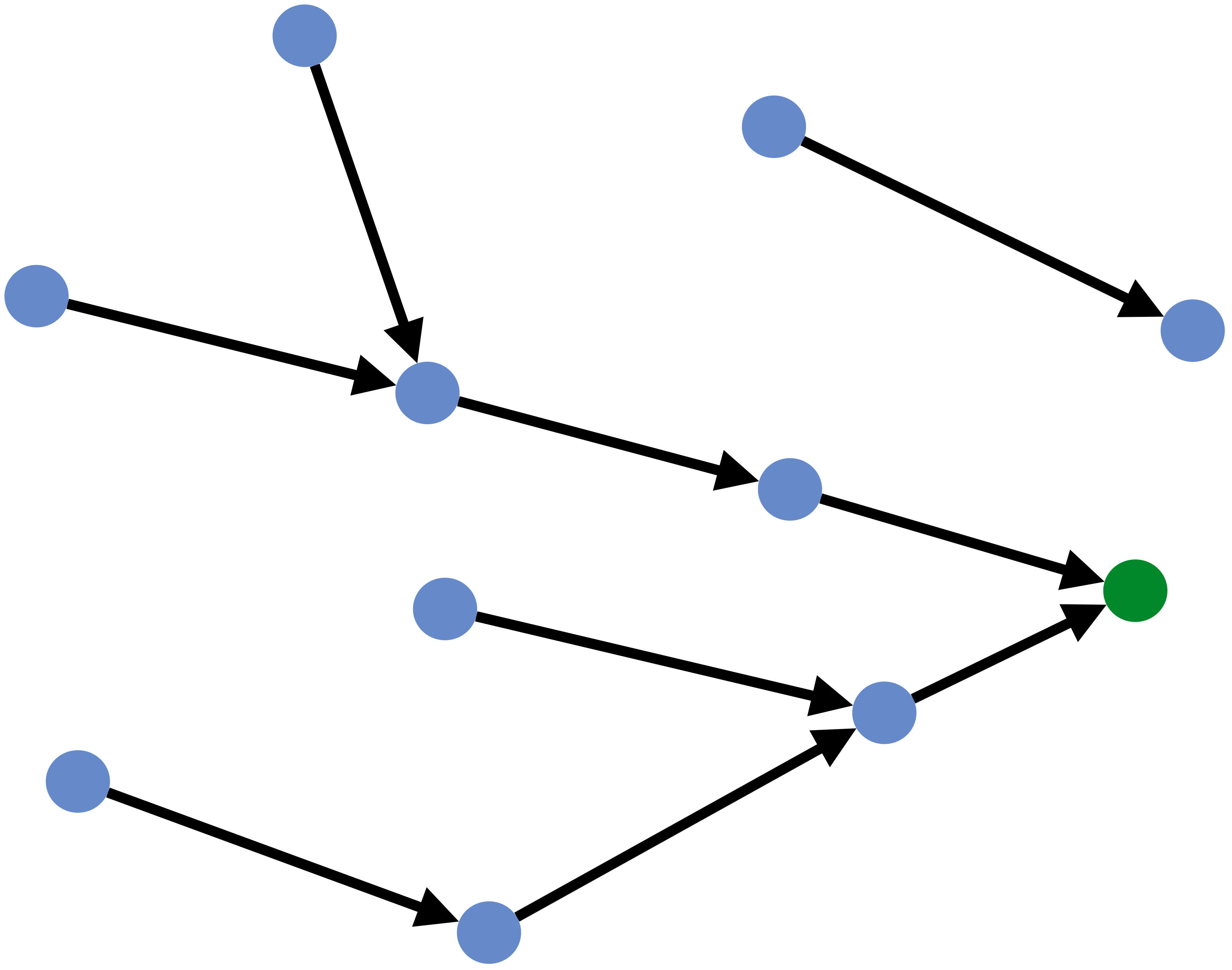


Is this valid?

Example 3:



Example 3:



Is this valid?

Checking Validity of a Routing State

- Simple to check validity of routing state for a particular destination

- Dead ends: nodes without arrows

- Loops: obvious, disconnected from destination and rest of the graph

Two Questions

- How can we **verify** given routing state is valid?

- How can we **produce** valid routing state?

Creating Valid Routing State

- Easy to avoid dead ends
- Avoiding loops is hard
- **The key difference between routing protocols is how they avoid loops!**
- Try to think a loop avoidance design for five minutes

#1: Create Tree Out of Topology

- Remove enough links to create a tree containing all nodes
- Sounds familiar? Spanning trees!
- If the topology has no loops, then just make sure not sending packets back from where they came
  - That causes an immediate loop
- Therefore, if no loops in topology and no formation of immediate loops ensures valid routing
- However… three challenges
  - Unnecessary host resources used to process packets
  - High latency
  - Low bandwidth (utilization)

#2: Obtain a Global View

- A global view of the network makes computing paths without loops easy
    - Many graph algorithms for computing loop-free paths
    - For e.g., Dijkstra's Algorithm

- Getting the global view of network is challenging!

#3: Distributed Route Computation

- Often getting a global view of the network is infeasible
  - Distributed algorithms to compute feasible route

- **Approach A**: Finding optimal route for maximizing/minimizing a metric

- **Approach B**: Finding feasible route via exchanging paths among switches

Welcome to the Network Layer!

- THE functionality: **delivering the data**

- **THE protocol: Internet Protocol (IP)**
  - To achieve its functionality (delivering the data), IP protocol has **three** responsibilities

- **Addressing (next lecture)**
- **Encapsulating data into packets (actually datagrams; next lecture)**
- **Routing (using a variety of protocols; several lectures)**

Next lecture!