

# CS4450

## Computer Networks: Architecture and Protocols

### Lecture 12

### Intra-domain Routing: Deep Dive

**Rachit Agarwal**



# Announcements

- Problem Set 3 released
- **Prelim: 26th March, In-class**

# Goals for Today's Lecture

- **Recap: Link State (Global view, Local computation)**
- **Distance Vector (Local view, Local computation)**
- Maintain sanity: its one of the “harder” lectures
  - I'll try to make it -less- hard, but ...
    - Pay attention
    - Review again tomorrow
    - Work out a few examples

**Recap from last few lectures**

# Recap: Spanning Tree Protocol ...

- Used in switched Ethernet to avoid broadcast storm
- Can be used for routing on the Internet (via “flooding” on spanning tree)
- **Three fundamental issues:**
  - Unnecessary processing at end hosts (that are not the destination)
  - Higher latency
  - Lower available bandwidth

# Recap: Routing Tables

- **Routing table:**
  - Each switch: the next hop for each destination in the network
- **Routing state:** collection of routing tables across all nodes
- Two questions:
  - How can we **verify** given routing state is valid?
  - How can we **produce** valid routing state?
- Global routing state valid **if and only if:**
  - There are no **dead ends** (other than destination)
  - There are no “**persistent**” **loops**

# Recap: The right way to think about Routing Tables

- Routing tables are nothing but ....
  - A collection of (directed) spanning tree
  - One for each destination
- **Routing Protocols**
  - Mechanisms to producing valid routing tables
  - What we will see:
    - “n” spanning tree protocols running in parallel

# Recap: Three flavors of protocols for producing valid routing state

- **Create Tree, route on tree**
  - E.g., Spanning tree protocol (switched Ethernet)
  - **Good:** easy, no (persistent) loops, no dead ends
  - **Not-so-good:** unnecessary processing, high latency, low bandwidth
- **Obtain a global view:**
  - E.g., Link state (last lecture)
- **Distributed route computation:**
  - E.g., Distance vector (last lecture, today)
  - E.g., Border Gateway Protocol (next couple of lectures)



# Recap: Producing Valid Routing State

- Easy to avoid dead ends
- Avoiding loops is hard
- **The key difference between routing protocols is how they avoid loops!**

# Recap: Global View

- **Questions:**
  - Where to create global view
  - How to create global view
  - When to run route computation

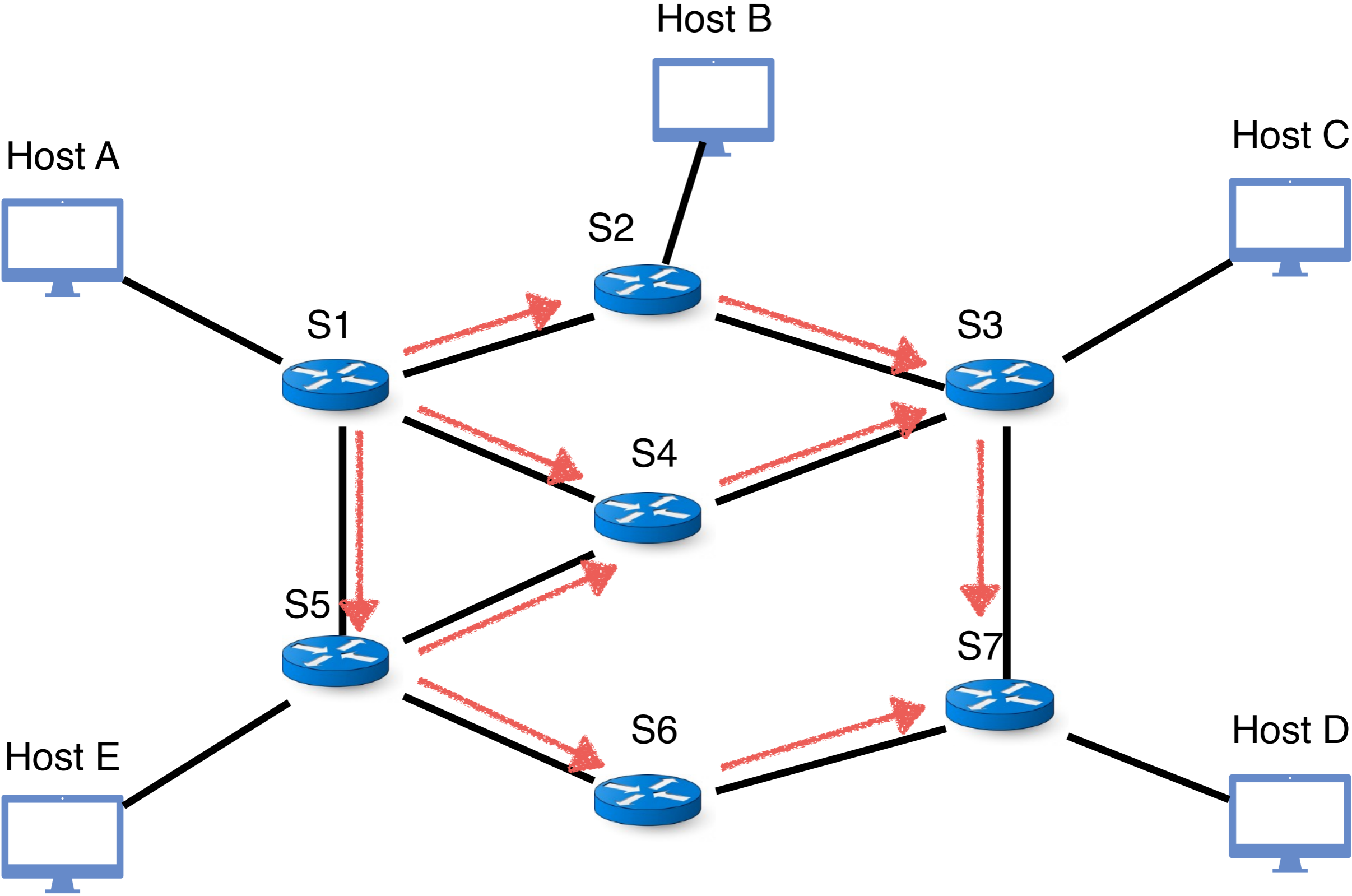
# Recap: Where to create global view?

- One option: Central server
  - Collects a global view
  - Computes the routing table for each node
  - “Installs” routing tables at each node
  - **Software-defined Networks: later in course**
- Second option: At each router
  - Each router collects a global view
  - Computes its own routing table using Link-state protocol
- **Link-state routing protocol**
  - OSPF is a specific implementation of link-state protocol
    - IETF RFC 2328 (IPv4) or 5340 (IPv6)

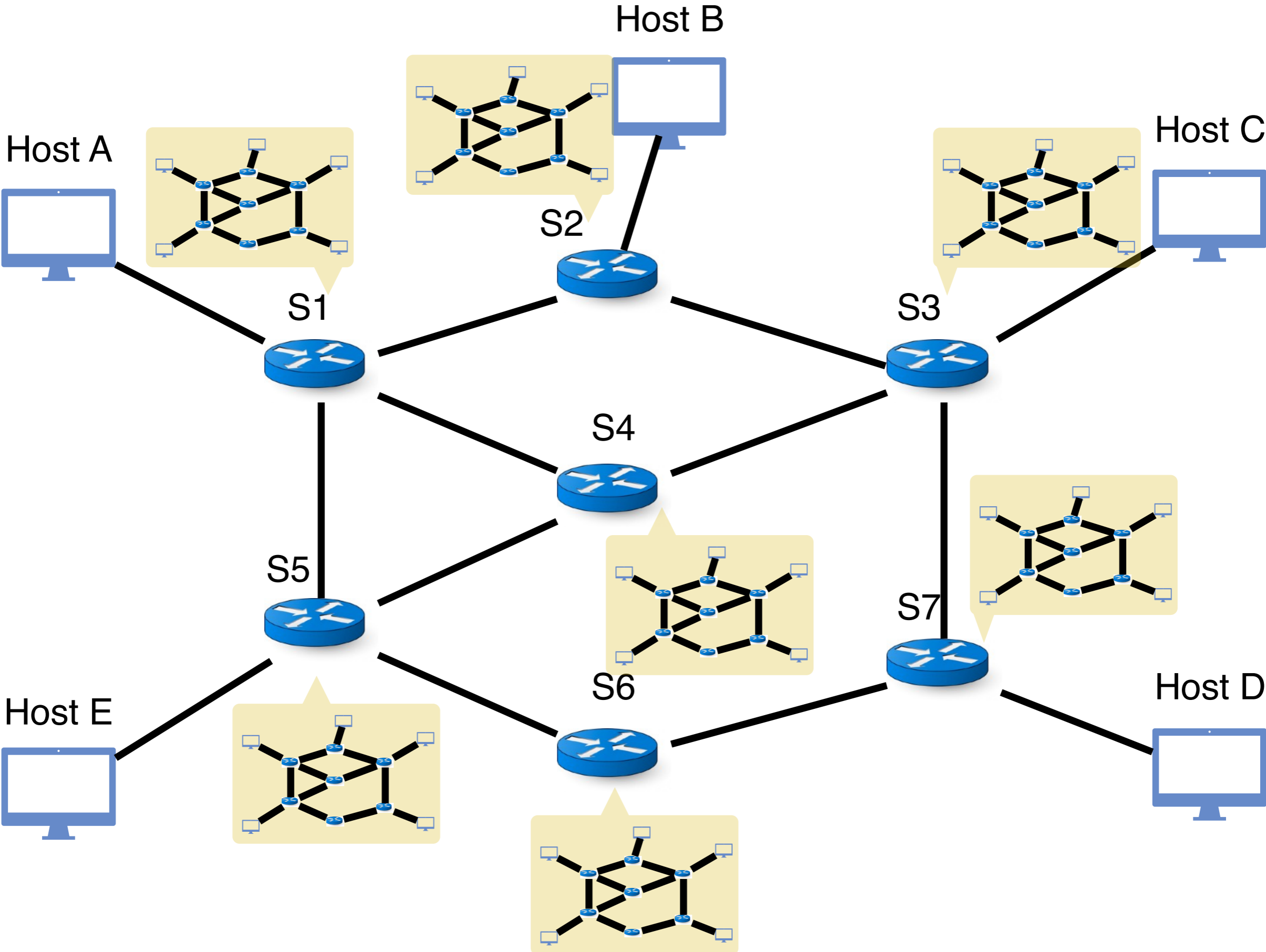
# Recap: Overview of Link-State Routing

- **Every router knows its local “link state”**
  - Knows state of links to neighbors
  - Up/down, and associated cost
- **A router floods its link state to all other routers**
  - Uses a special packet — Link State Announcements (LSA)
  - Announcement is delivered to all nodes (next slide)
  - Hence, every router learns the entire network graph
- **Runs route computation locally**
  - Computing least cost paths from them to all other nodes
  - E.g., using Dijkstra’s algorithm

# Recap: Link-State Routing



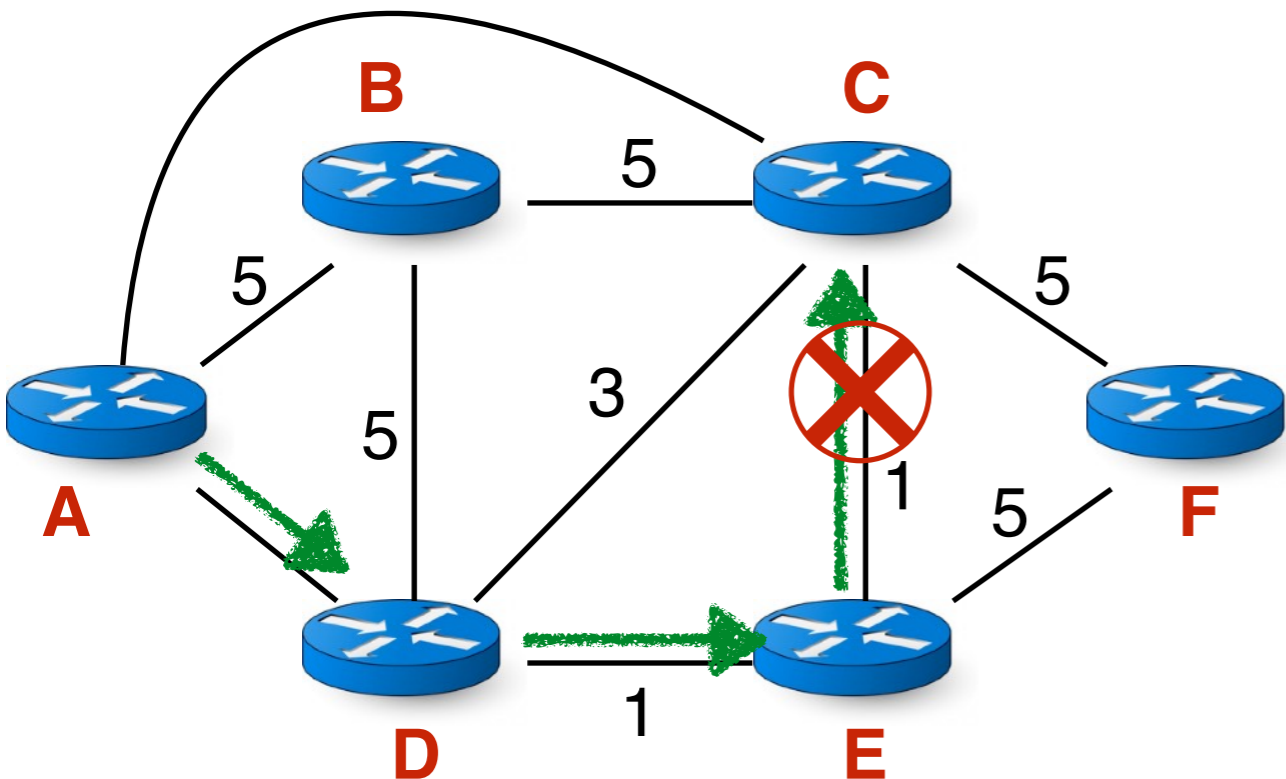
# Recap: Each Node Then has a Global View



# Recap: When to Initiate Flooding of announcements?

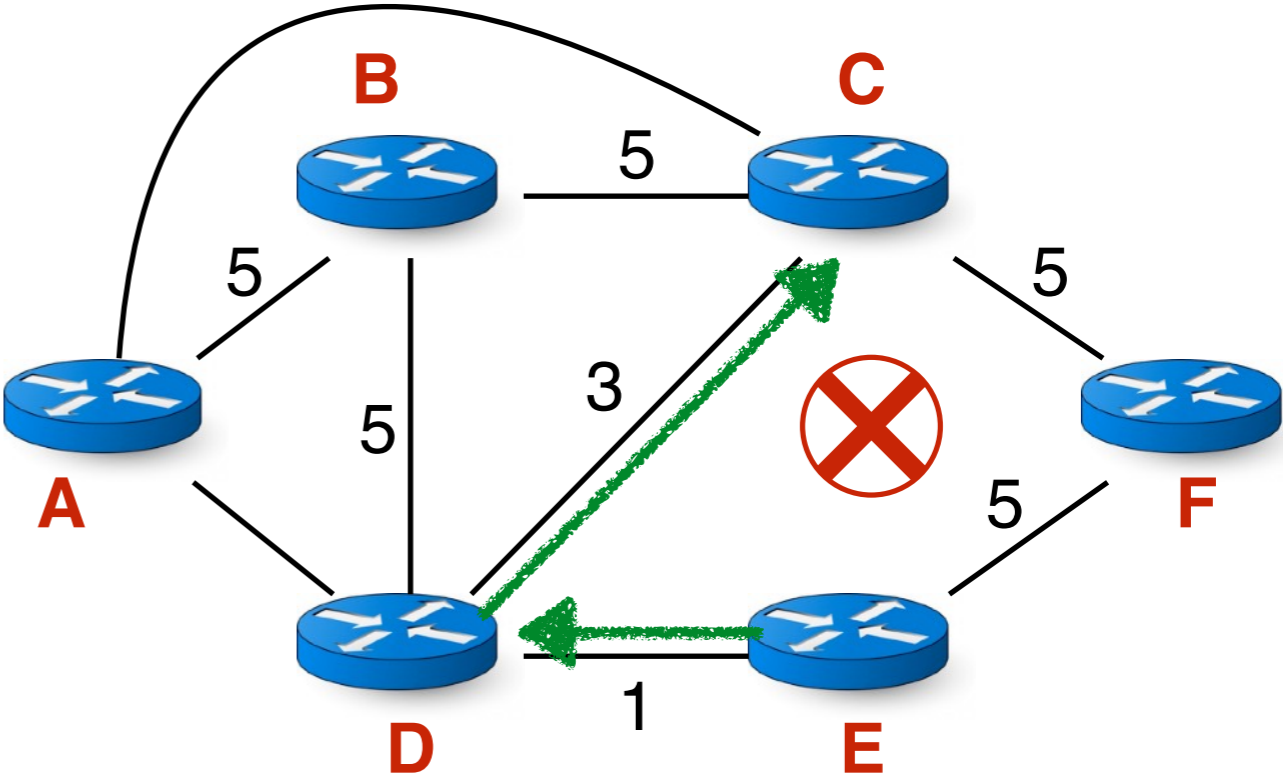
- **Topology change**
  - Link failures
  - Link recovery
- **Configuration change**
  - Link cost change (why would one change link cost?)
- **Periodically**
  - Refresh the link-state information
  - Typically (say) 30 minutes
  - Corrects for possible corruption of data

# Recap: Are Loops Still Possible?



A and D think this is the path to C

E-C link fails, but D doesn't know yet

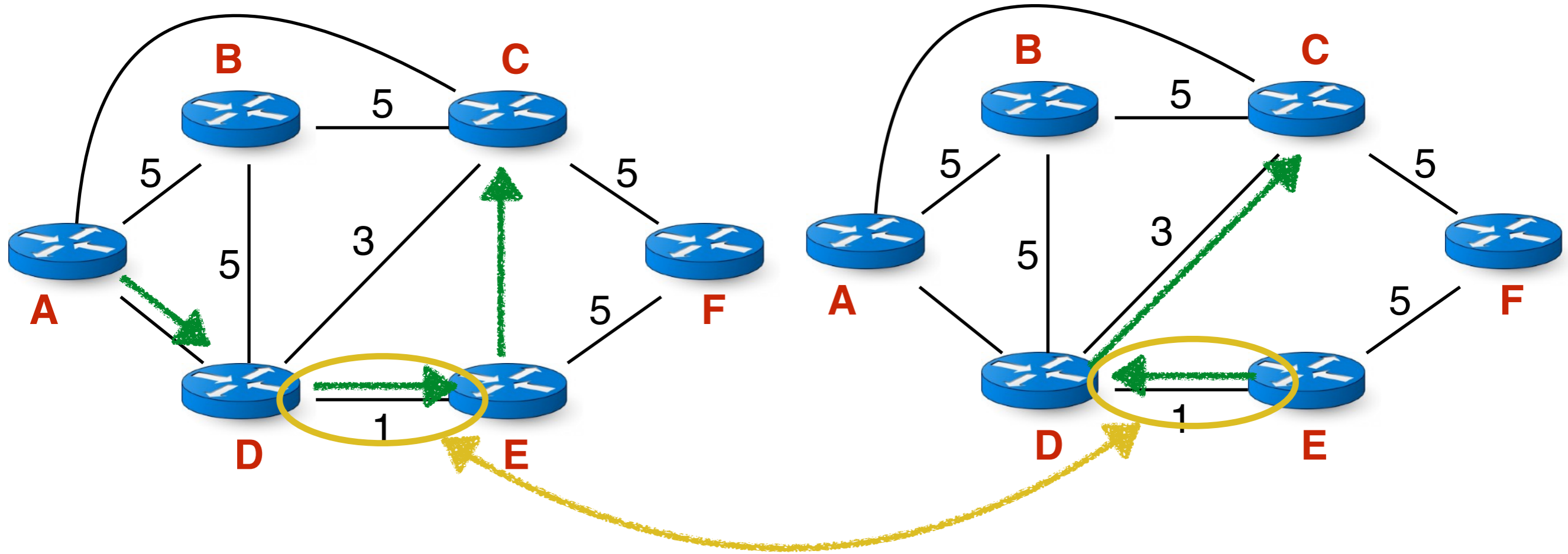


E thinks that this the path to C

E reaches C via D, D reaches C via E  
Loop!



# Recap: Transient Disruptions



- Inconsistent link-state views
  - Some routers know about failure before others
  - The shortest paths are no longer consistent
  - Can cause **transient forwarding loops**
    - **Transient loops** are still a problem!

# Recap: Convergence in Link-State Routing

- **Eventually**, all routers have consistent routing information
  - E.g., all nodes having the same link-state database
- Here, eventually means “if nothing changes after a while”
- Forwarding is consistent after convergence
  - All nodes have the same link-state database
  - All nodes forward packets on same paths
- **But while still converging, bad things can happen**

**Questions?**

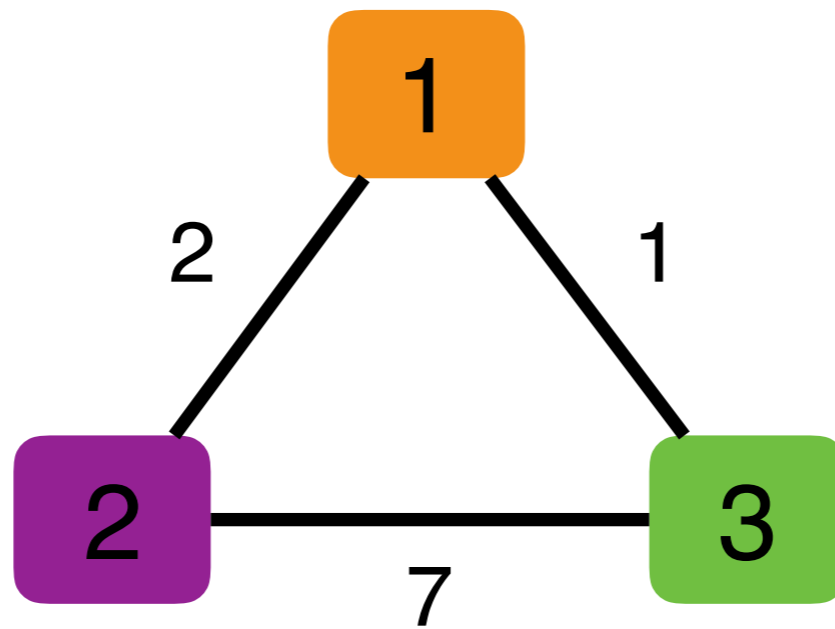
# **Distributed Route Computation**

# Towards Distance-vector protocol with next-hops (no failures)

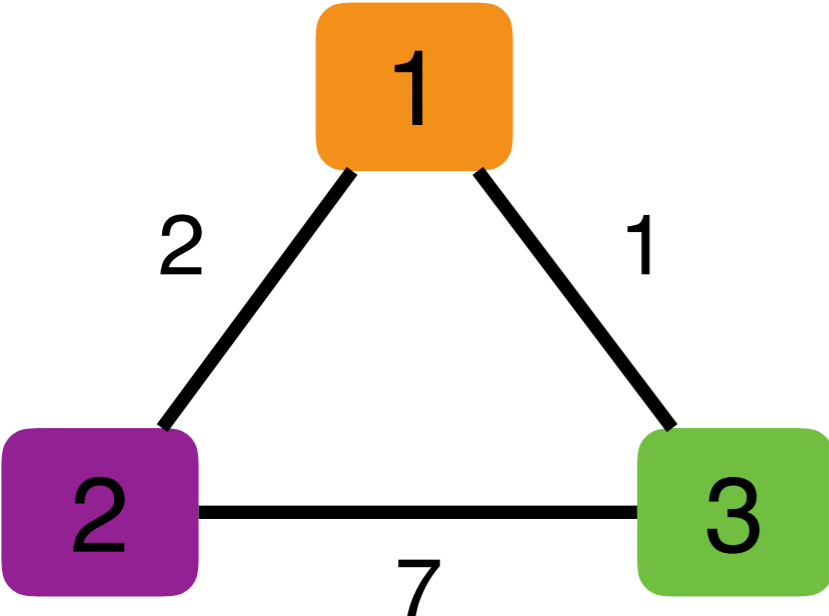
- **Messages (Y,d,X):** For root Y; From node X; advertising a distance d to Y
- Initially each switch X announces (X,0,X) to its neighbors
- Switch Z updates its view
  - Upon receiving message (Y,d,X) from X, ~~check Y's id~~
  - ~~If Y's id < current root: set root destination = Y~~
- Switch Z computes its shortest distance from the ~~root~~ destination
  - If  $\text{current\_distance\_to\_Y} > d + \text{cost of link to X}$ :
    - update  $\text{current\_distance\_to\_Y} = d + \text{cost of link to X}$
    - **update next\_hop\_to\_destination = X**
- If ~~root~~ **changed** OR shortest distance to the ~~root~~ destination **changed**, send all neighbors updated message (Y, d+cost-of-link-to-X, Z)

## Group Exercise:

Lets run the Protocol on this example

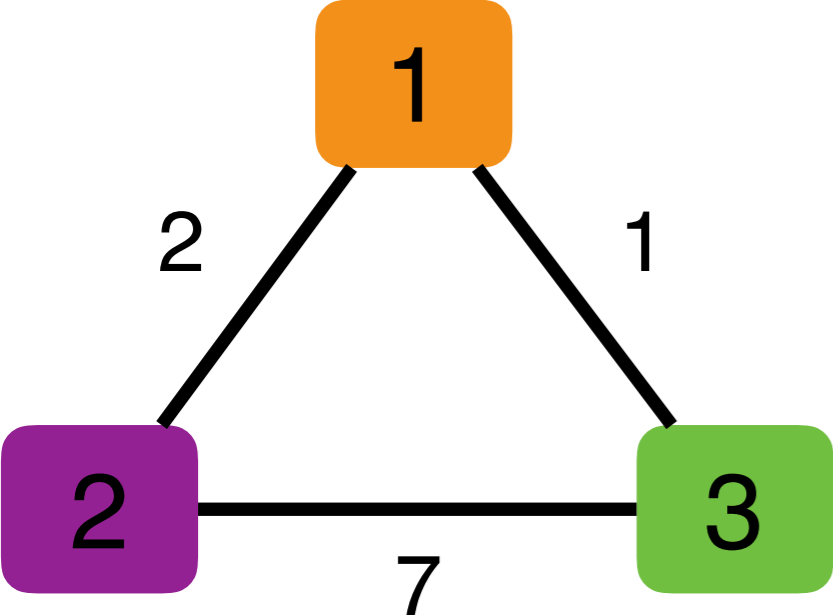


# Round 1



	Receive	Send
1		(1, 0, 1)
2		(2, 0, 2)
3		(3, 0, 3)

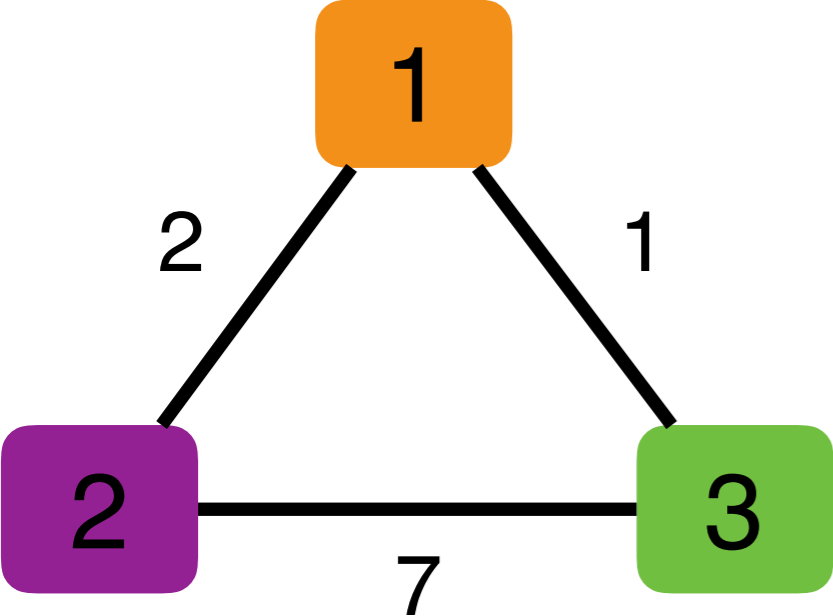
# Round 2



	Receive	Send
1 (1, 0, 1)	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)
2 (2, 0, 2)	(1, 0, 1), (3, 0, 3)	(1, 2, 2), (3, 7, 2)
3 (3, 0, 3)	(1, 0, 1), (2, 0, 2)	(1, 1, 3), (2, 7, 3)

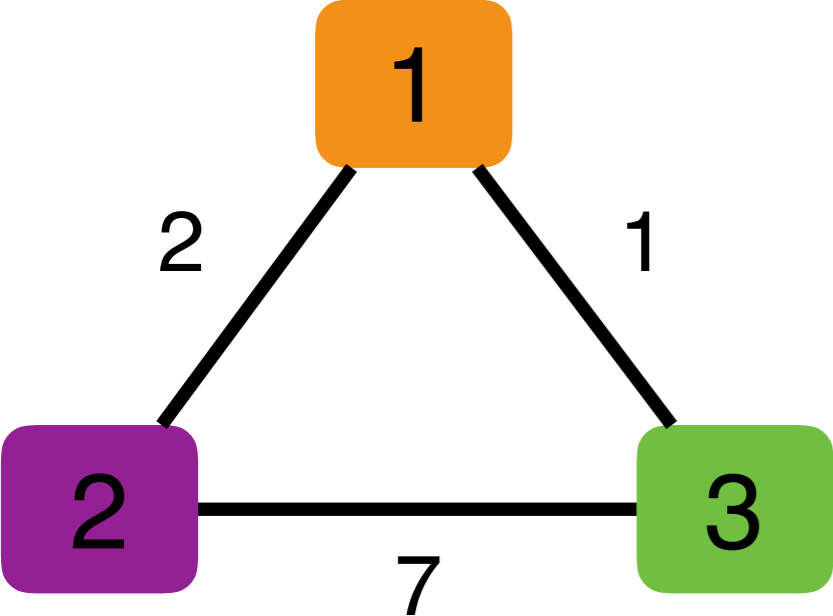


# Round 3



	Receive	Send
1 (1, 0, 1) (2, 2, 1), (3, 1, 1)	(1, 2, 2), (3, 7, 2), (1, 1, 3), (2, 7, 3)	
2 (1, 2, 2), (2, 0, 2), (3, 7, 2)	(2, 2, 1), (3, 1, 1), (1, 1, 3), (2, 7, 3)	(3, 3, 2)
3 (1, 1, 3), (2, 7, 3), (3, 0, 3)	(2, 2, 1), (3, 1, 1), (1, 2, 2), (3, 7, 2)	(2, 3, 3)

# Round 4



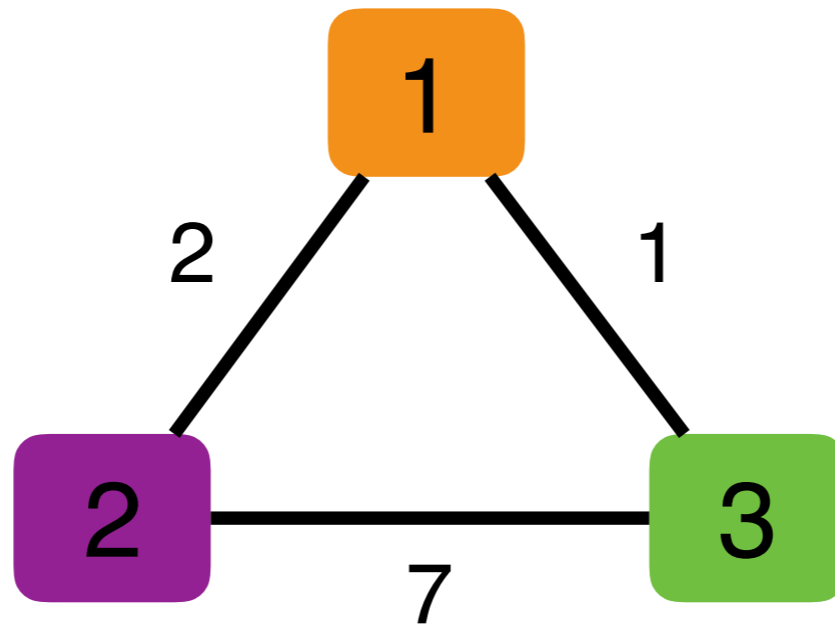
	Receive	Send
1 (1, 0, 1) (2, 2, 1), (3, 1, 1)	(3, 3, 2), (2, 3, 3)	
2 (1, 2, 2), (2, 0, 2), (3, 3, 2)	(2, 3, 3)	
3 (1, 1, 3), (2, 3, 3), (3, 0, 3)	(3, 3, 2)	

# Why not Spanning Tree Protocol? Why Distance “Vector”?

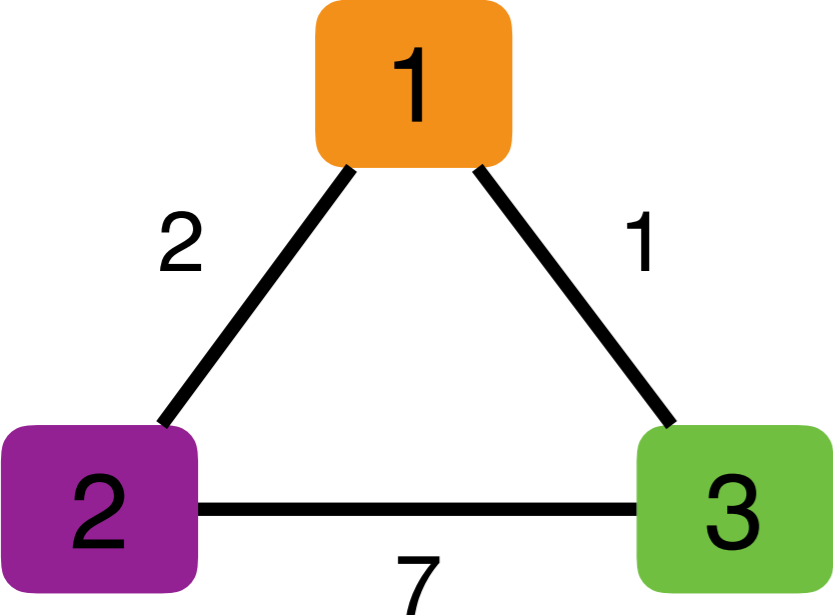
- The same protocol/algorithm applies to all destinations
- Each node announces distance to **each** dest
  - I am 4 hops away from node A
  - I am 6 hops away from node B
  - I am 3 hops away from node C
  - ...
- Nodes are exchanging a **vector** of distances

## Group Exercise:

Lets run the Protocol on this example  
(this time with next-hops)

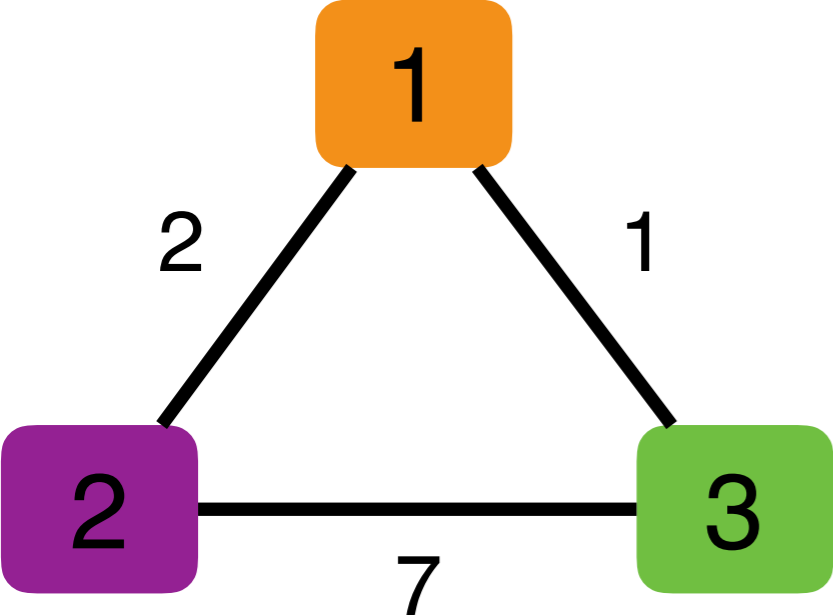


# Round 1



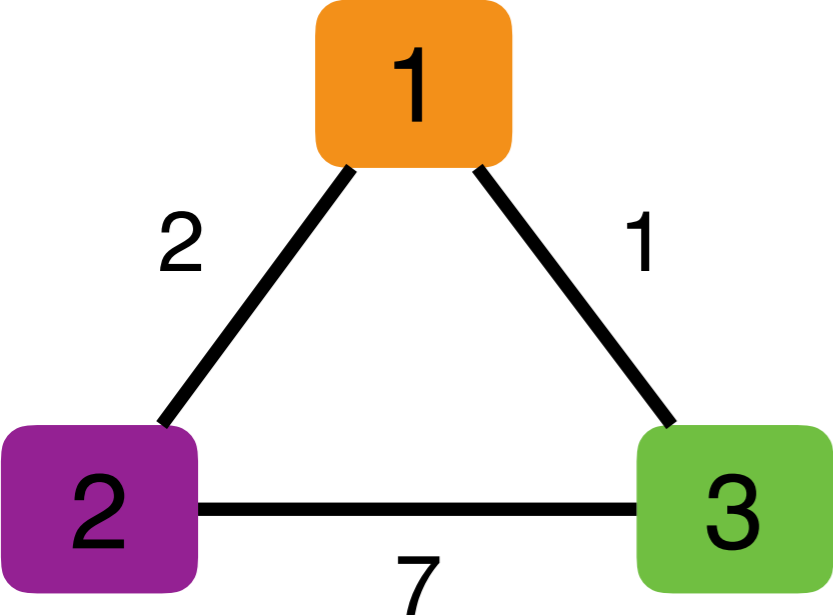
	Receive	Send	Next-hops
1		(1, 0, 1)	[-]
2		(2, 0, 2)	[-]
3		(3, 0, 3)	[-]

# Round 2



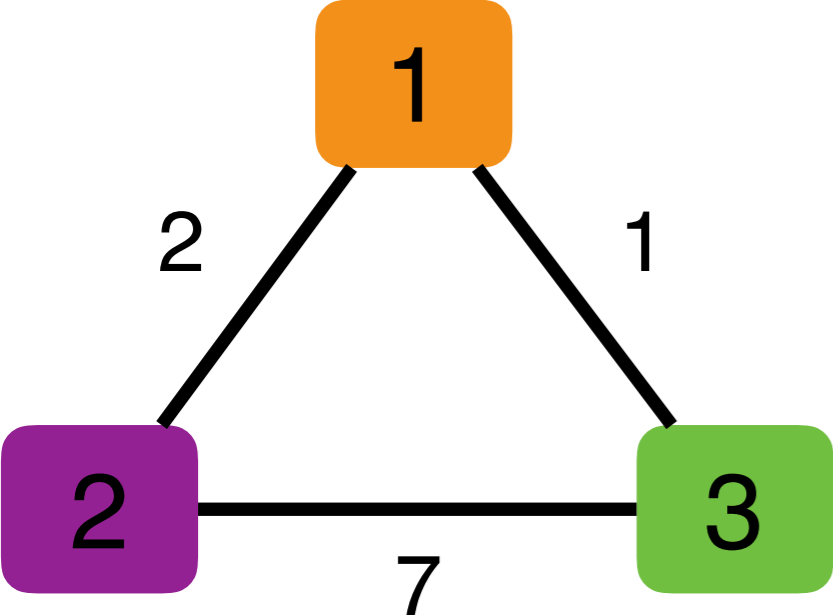
	Receive	Send	Next-hops
1 (1, 0, 1)	(2, 0, 2), (3, 0, 3)	(2, 2, 1), (3, 1, 1)	[-, 2, 3]
2 (2, 0, 2)	(1, 0, 1), (3, 0, 3)	(1, 2, 2), (3, 7, 2)	[1, -, 3]
3 (3, 0, 3)	(1, 0, 1), (2, 0, 2)	(1, 1, 3), (2, 7, 3)	[1, 2, -]

# Round 3



	Receive	Send	Next-hops
1 (1, 0, 1) (2, 2, 1), (3, 1, 1)	(1, 2, 2), (3, 7, 2), (1, 1, 3), (2, 7, 3)		[-, 2, 3]
2 (1, 2, 2), (2, 0, 2), (3, 7, 2)	(2, 2, 1), (3, 1, 1), (1, 1, 3), (2, 7, 3)	(3, 3, 2)	[1, -, 1]
3 (1, 1, 3), (2, 7, 3), (3, 0, 3)	(2, 2, 1), (3, 1, 1), (1, 2, 2), (3, 7, 2)	(2, 3, 3)	[1, 1, -]

# Round 4



	Receive	Send	Next-hops
1 (1, 0, 1) (2, 2, 1), (3, 1, 1)	(3, 3, 2), (2, 3, 3)		[-, 2, 3]
2 (1, 2, 2), (2, 0, 2), (3, 3, 2)	(2, 3, 3)		[1, -, 1]
3 (1, 1, 3), (2, 3, 3), (3, 0, 3)	(3, 3, 2)		[1, 1, -]



# Distance Vector Protocol

- **Messages (Y,d,X):** For root Y; From node X; advertising a distance d to Y
- Initially each switch X initializes its routing table to (X,0,-) and distance infinity to all other destinations
- Switches announce their entire distance vectors (routing table w/0 next hops)
- Upon receiving a distance vector from a node (say X), each node does:
  - For each destination Y in the announcement ( $\text{distance}(X, Y) = d$ ):
    - If  $\text{current\_distance\_to\_Y} > d + \text{cost of link to X}$ :
      - update  $\text{current\_distance\_to\_Y} = d$
      - update  $\text{next\_hop\_to\_destination} = X$
- If shortest distance to any destination changed, send all neighbors your distance vectors

# Two Aspects to This Approach

- **Protocol:**

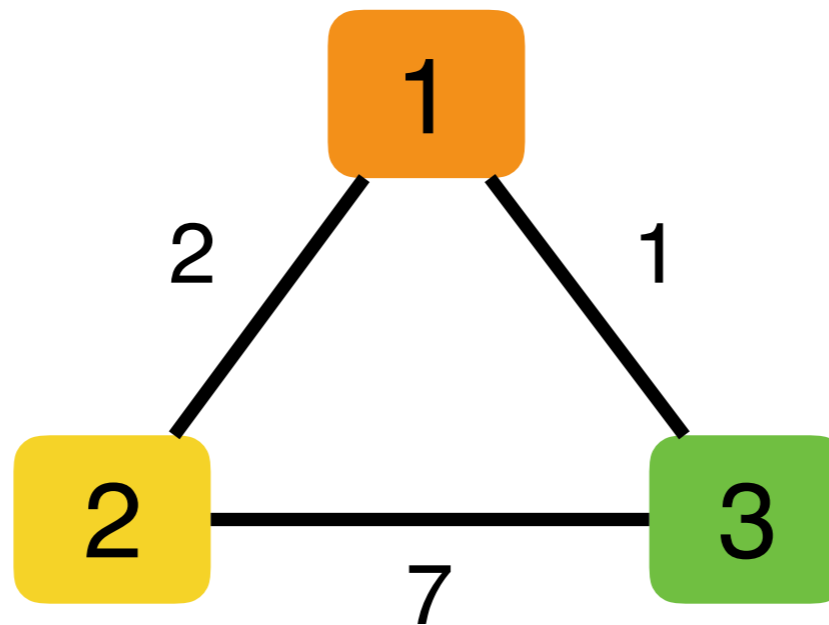
- Exchanging routing information with neighbors
- When to exchange distance vectors
- RIP is a protocol that implements DV (IETF RFC 2080)

- **Algorithm:**

- How to use the information from your neighbors to update your own routing tables?

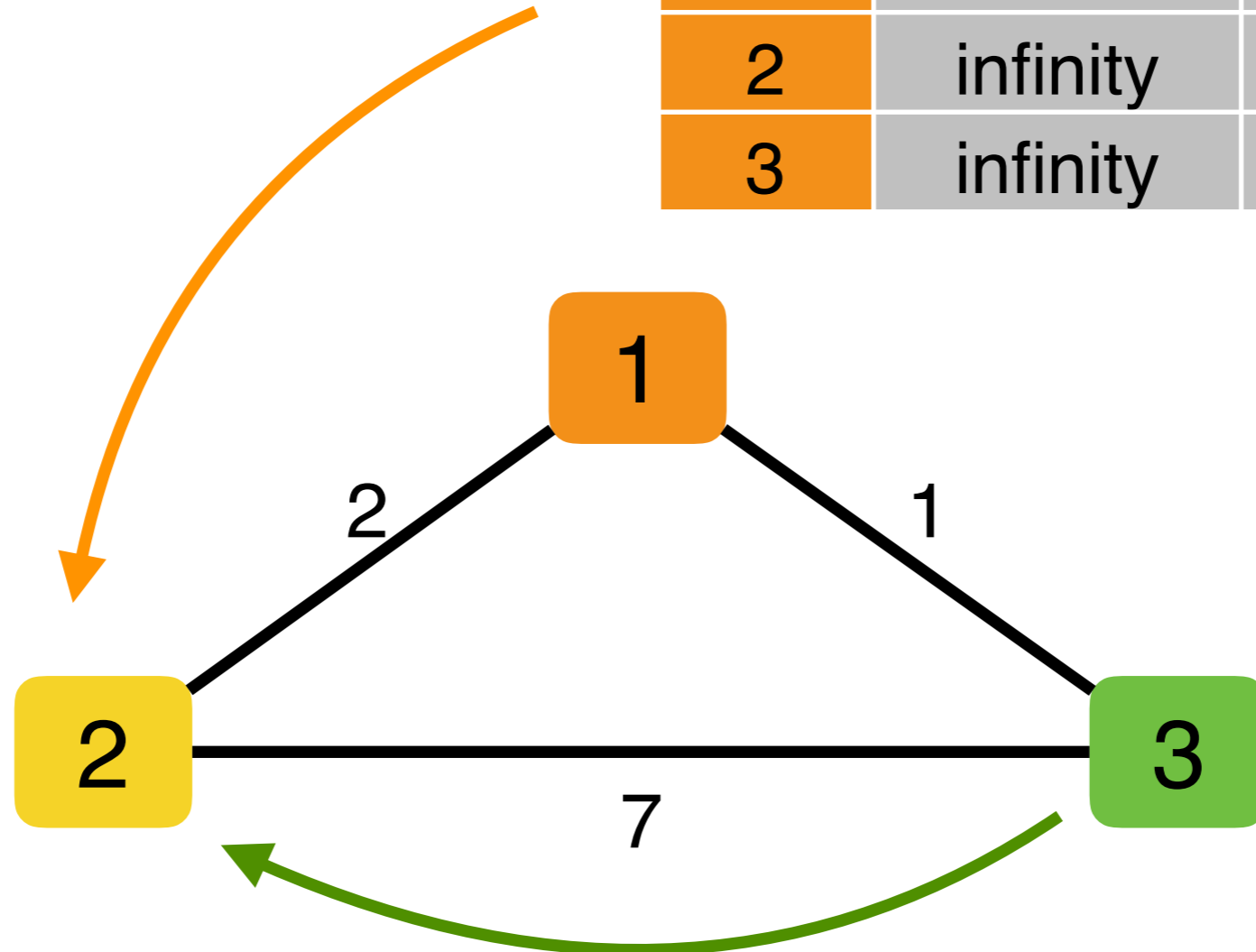
## Group Exercise:

Lets run the Protocol again on this example  
(this time with distance vectors)



# Round 1

	distance	next-hop
1	0	-
2	infinity	
3	infinity	

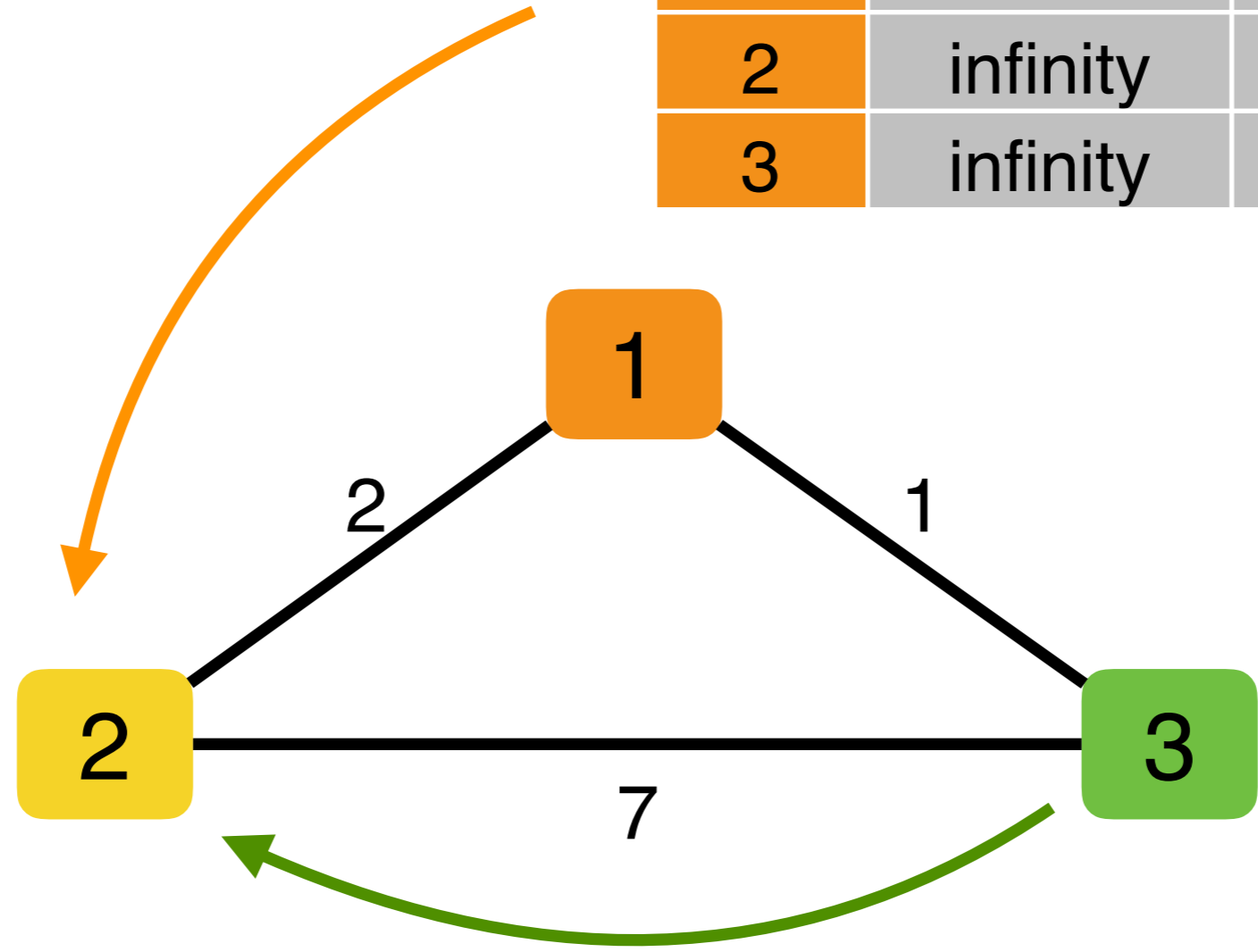


	distance	next-hop
1	infinity	
2	0	-
3	infinity	

	distance	next-hop
1	infinity	
2	infinity	
3	0	-

# Exchange after Round 1

	distance	next-hop
1	0	-
2	infinity	
3	infinity	

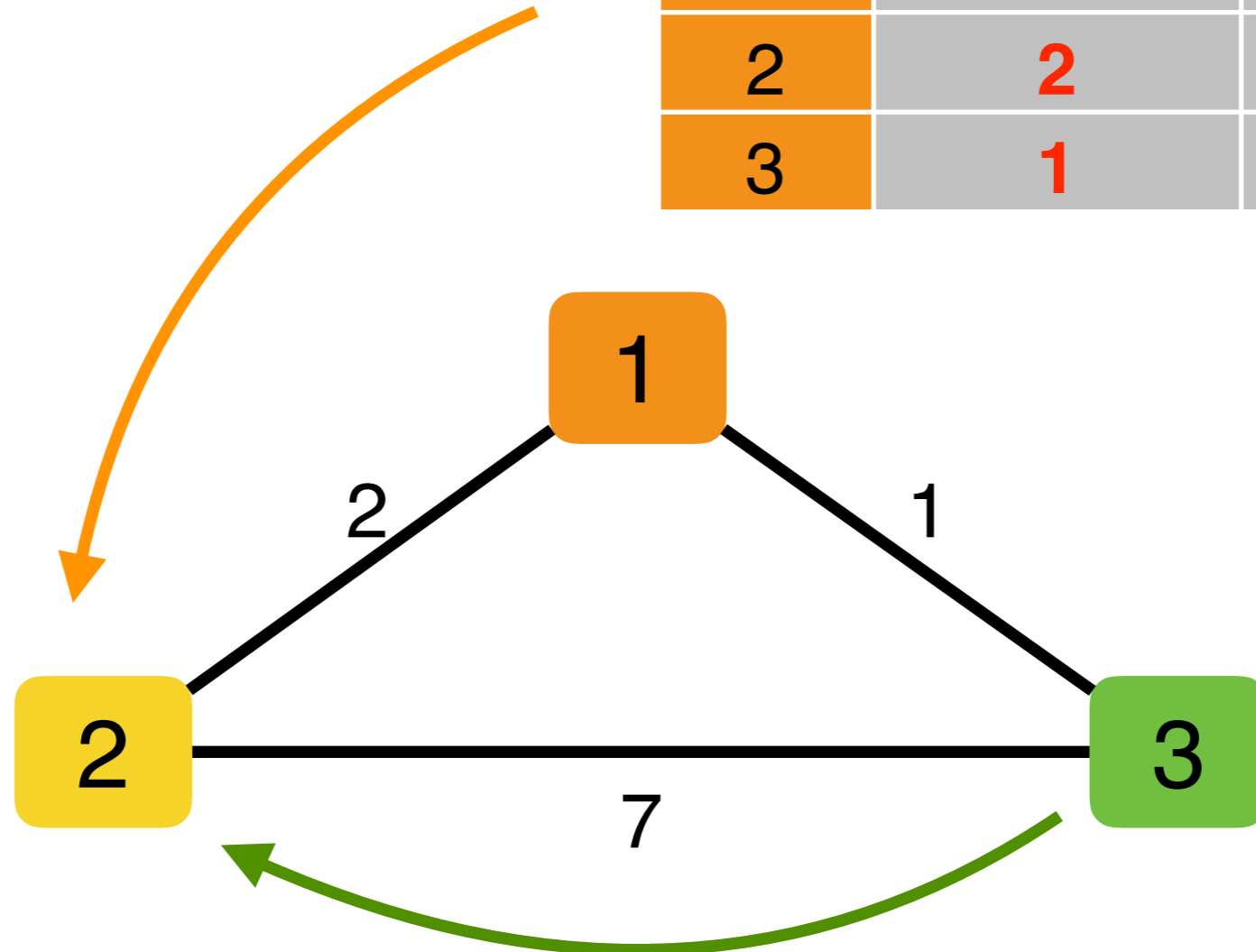


	distance	next-hop
1	2	1
2	0	-
3	7	3

	distance	next-hop
1	infinity	
2	infinity	
3	0	-

# Round 2

	distance	next-hop
1	0	-
2	<b>2</b>	<b>2</b>
3	<b>1</b>	<b>3</b>

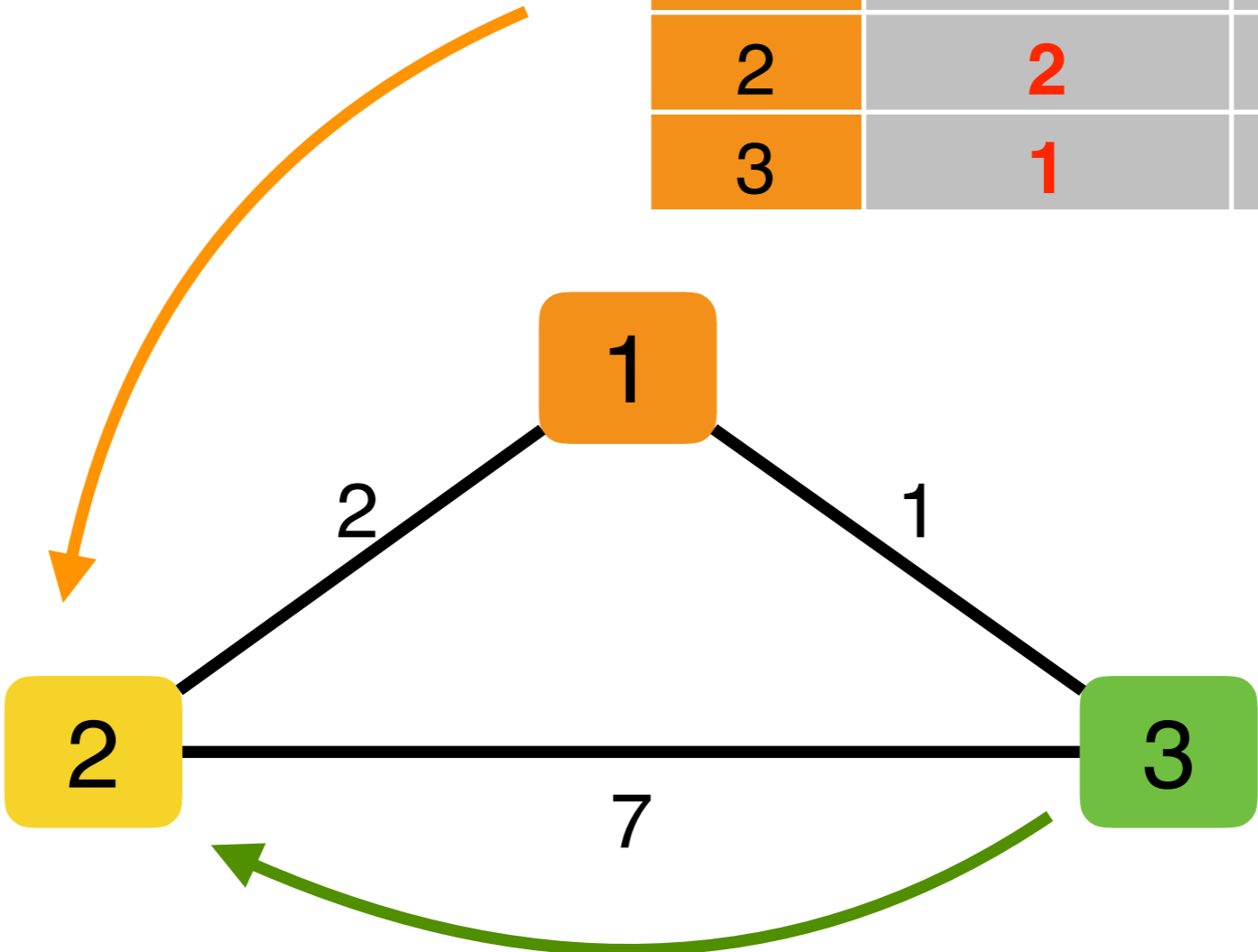


	distance	next-hop
1	<b>2</b>	<b>1</b>
2	0	-
3	<b>7</b>	<b>3</b>

	distance	next-hop
1	<b>1</b>	<b>1</b>
2	<b>7</b>	<b>2</b>
3	0	-

# Exchange after Round 2

	distance	next-hop
1	0	-
2	<b>2</b>	<b>2</b>
3	<b>1</b>	<b>3</b>

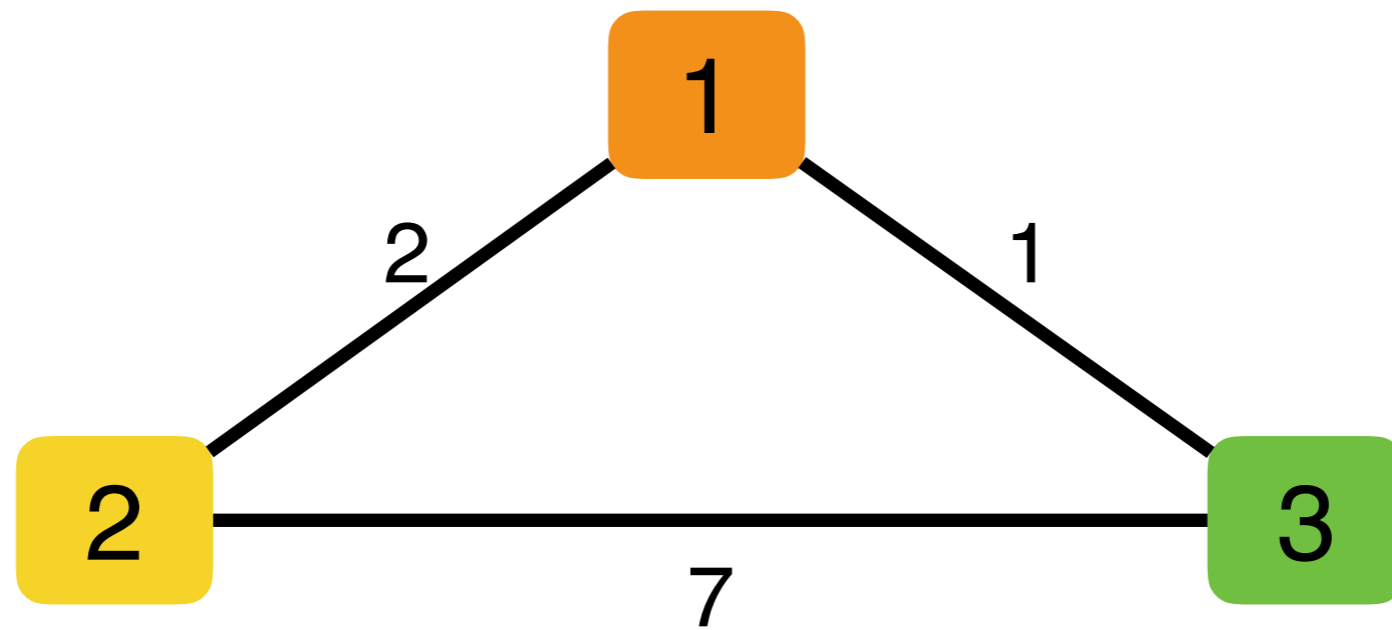


	distance	next-hop
1	2	1
2	0	-
3	<b>3</b>	<b>1</b>

	distance	next-hop
1	<b>1</b>	<b>1</b>
2	<b>7</b>	<b>2</b>
3	0	-

# Round 3

	distance	next-hop
1	0	-
2	2	2
3	1	3



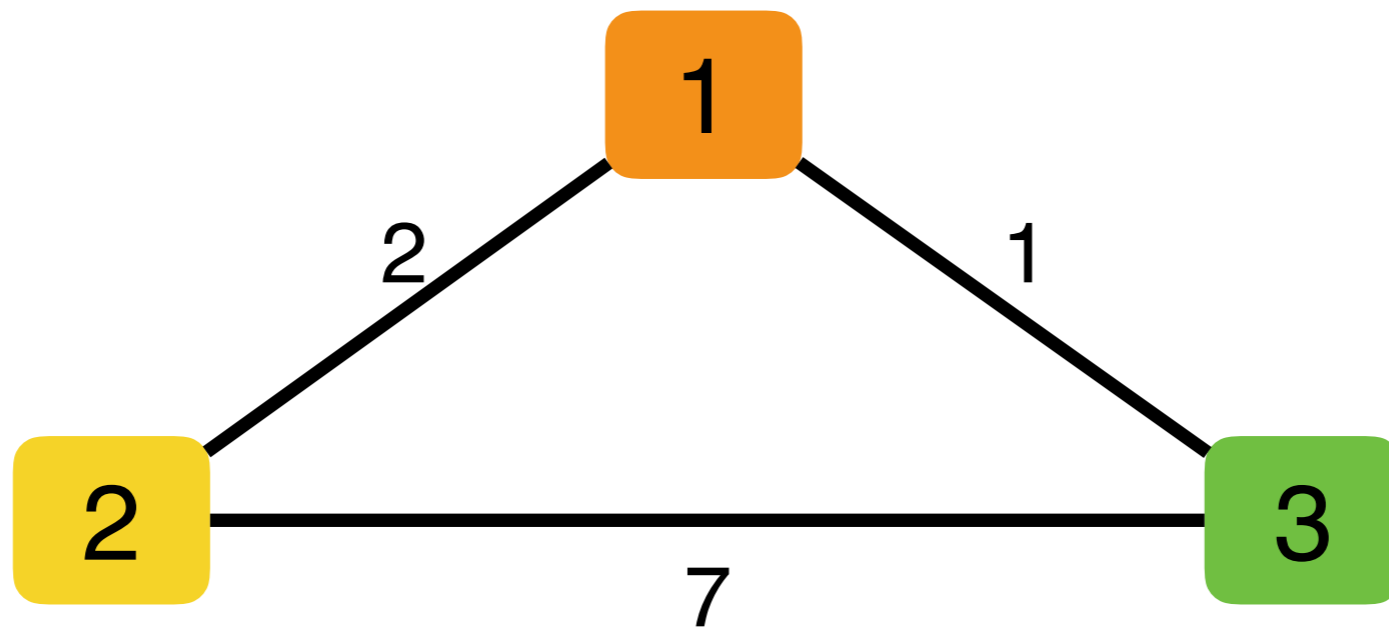
	distance	next-hop
1	2	1
2	0	-
3	<b>3</b>	<b>1</b>

	distance	next-hop
1	1	1
2	<b>3</b>	<b>1</b>
3	0	-



# Round 4

	distance	next-hop
1	0	-
2	2	2
3	1	3



	distance	next-hop
1	2	1
2	0	-
3	3	1

	distance	next-hop
1	1	1
2	3	1
3	0	-

# From Algorithm to Protocol

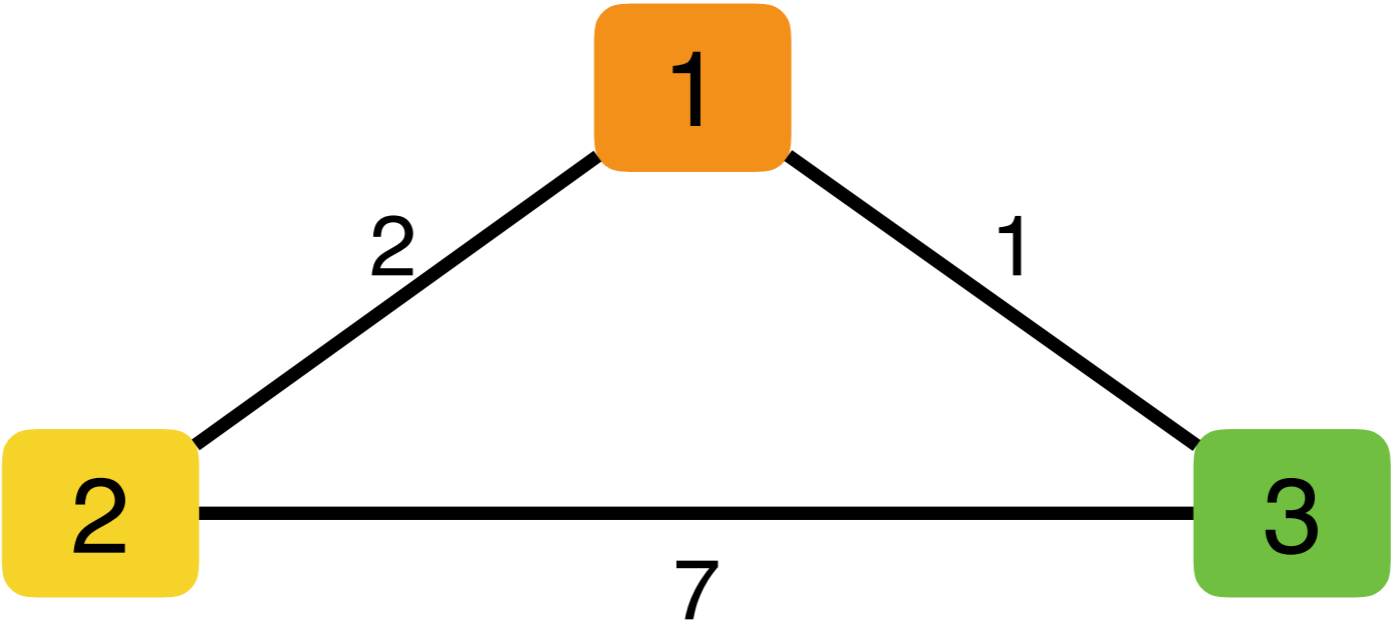
- Algorithm:
  - Nodes use Bellman-Ford to compute distances
- Protocol
  - Nodes exchange distance vectors
  - Update their own routing tables
  - And exchange again...
  - Details: when to exchange, what to exchange, etc....

# Other Aspects of Protocol

- When do you send messages?
  - When any of your distances  $d(u,v)$  change
    - What about when  $c(u,v)$  changes?
  - Periodically, to ensure consistency between neighbors
- What information do you send?
  - Could send entire vector
  - Or just updated entries
- Do you send everyone the same information
  - Consider the following slides

# Three node network

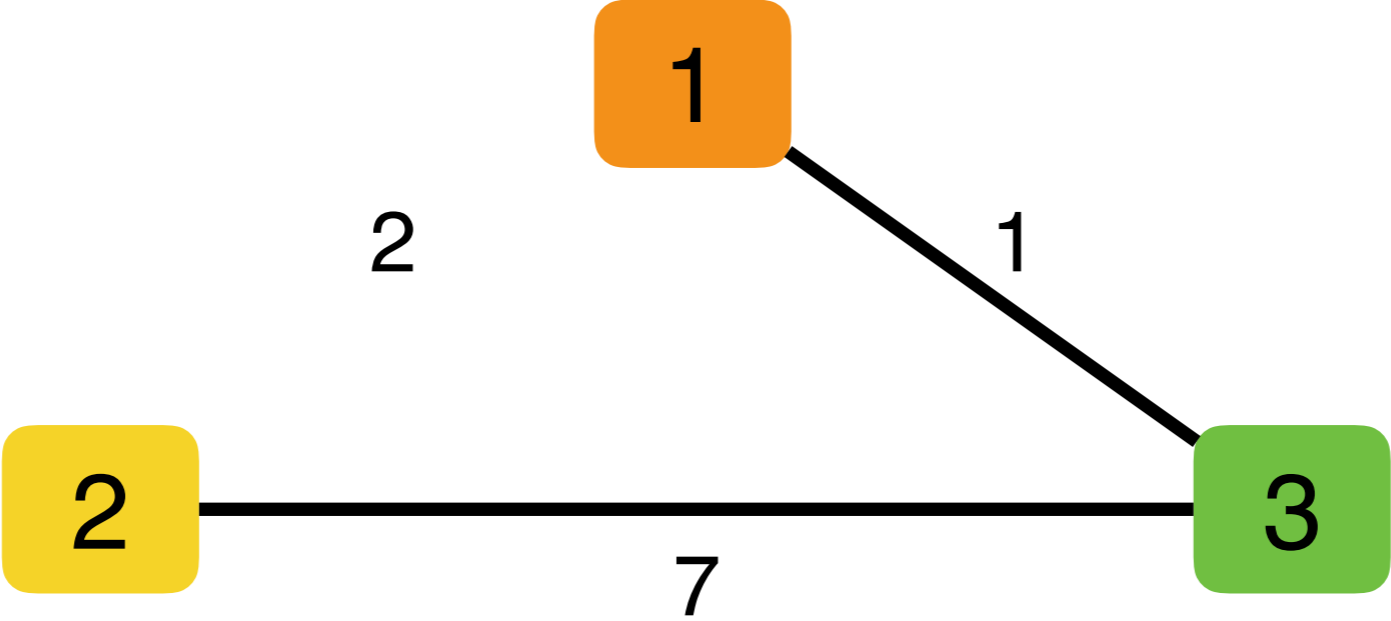
	distance	next-hop
1	0	-
2	2	2
3	1	3



	distance	next-hop
1	1	1
2	3	1
3	0	-

# Three node network

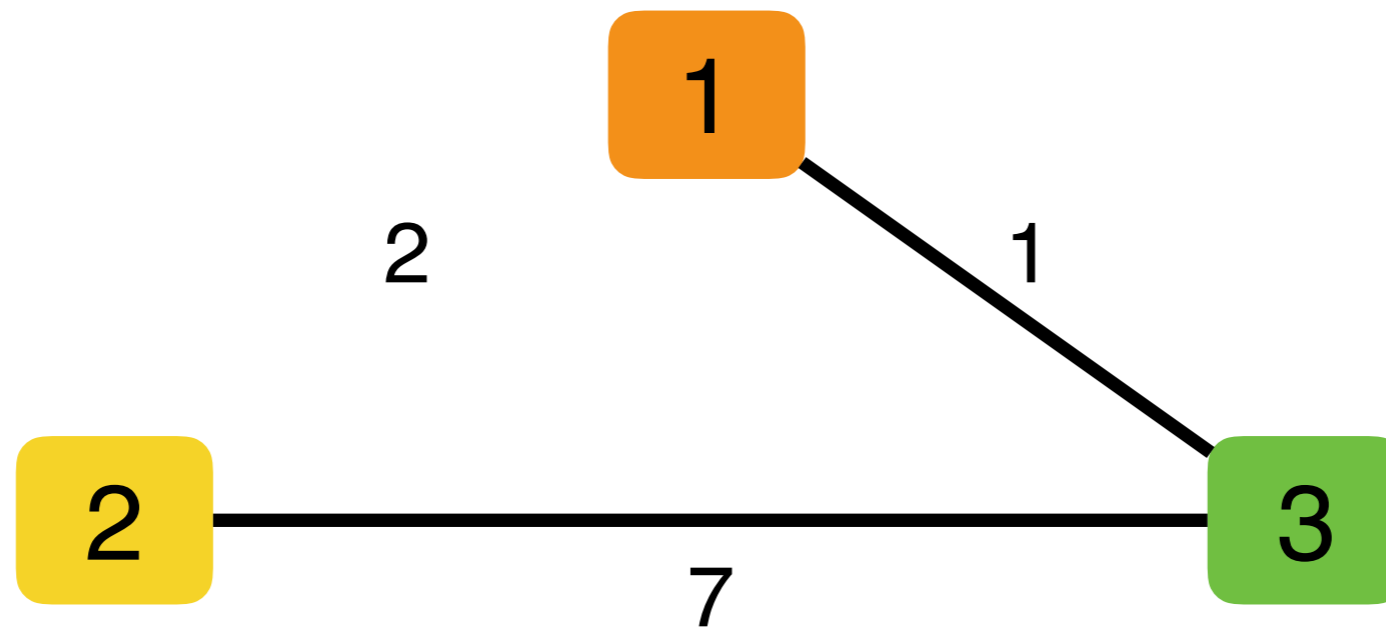
	distance	next-hop
1	0	-
2	infinity	
3	1	3



	distance	next-hop
1	1	1
2	3	1
3	0	-

# Round 1

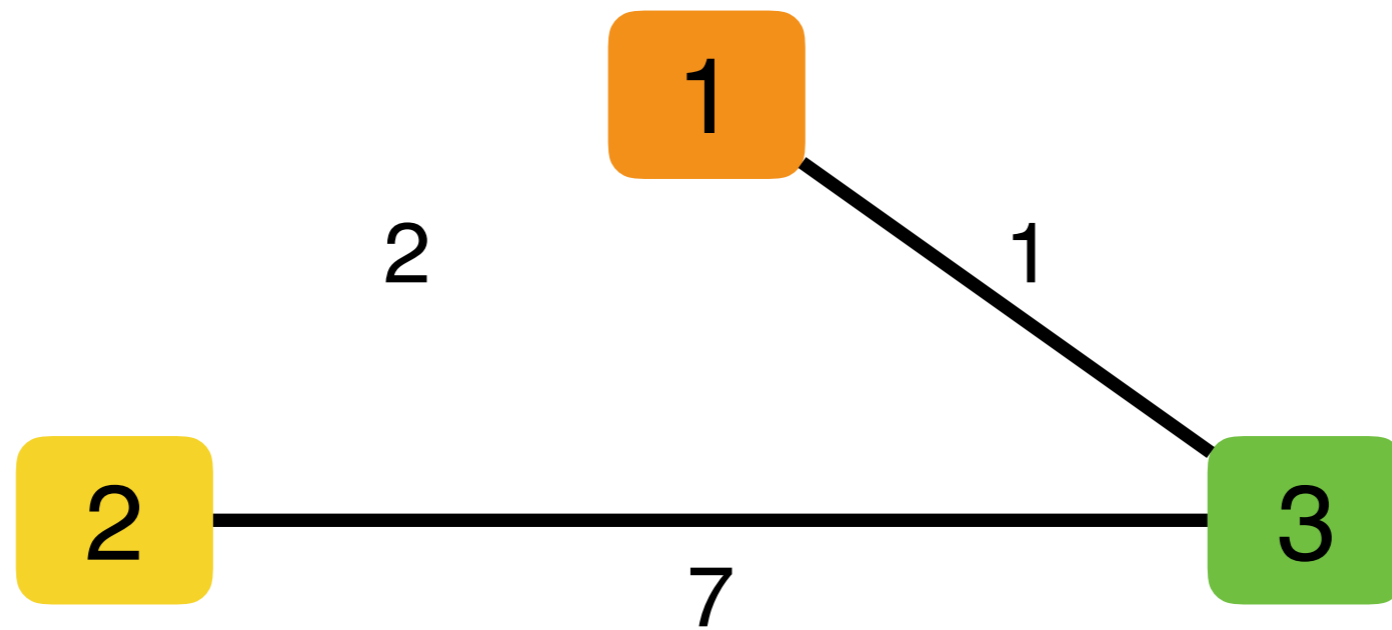
	distance	next-hop
1	0	-
2	4	3
3	1	3



	distance	next-hop
1	1	1
2	3	1
3	0	-

# Round 2

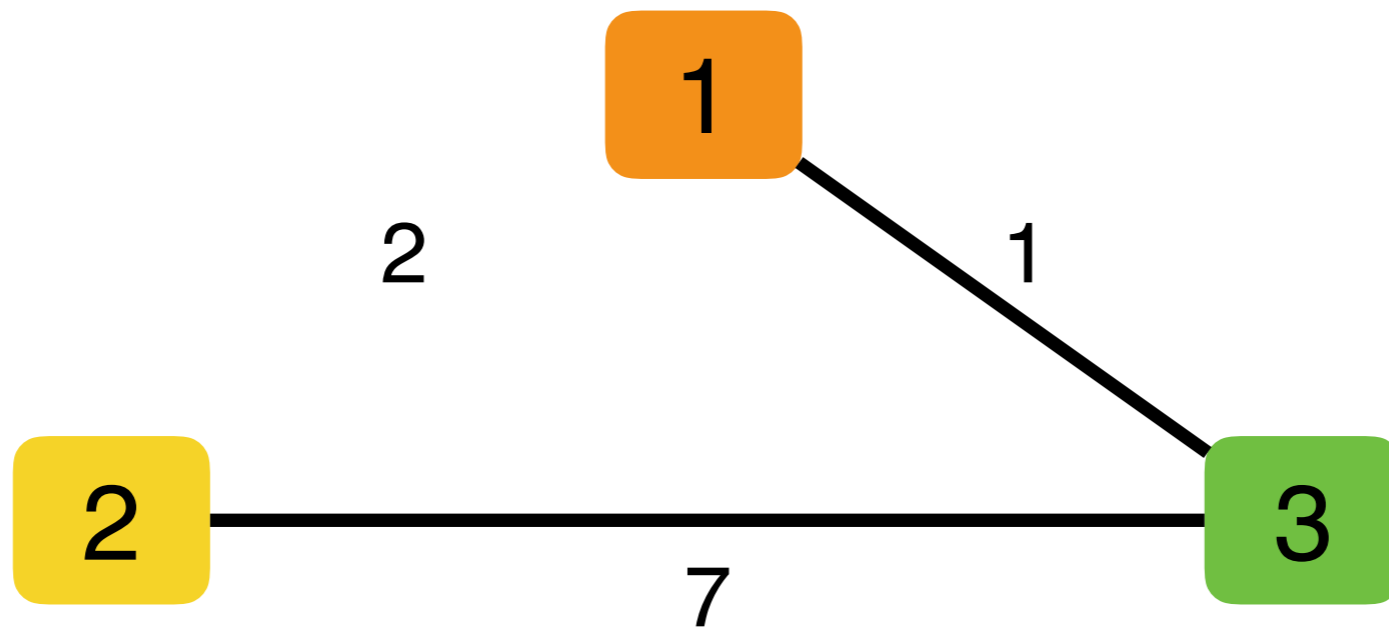
	distance	next-hop
1	0	-
2	4	3
3	1	3



	distance	next-hop
1	1	1
2	5	1
3	0	-

# Round 3

	distance	next-hop
1	0	-
2	6	3
3	1	3

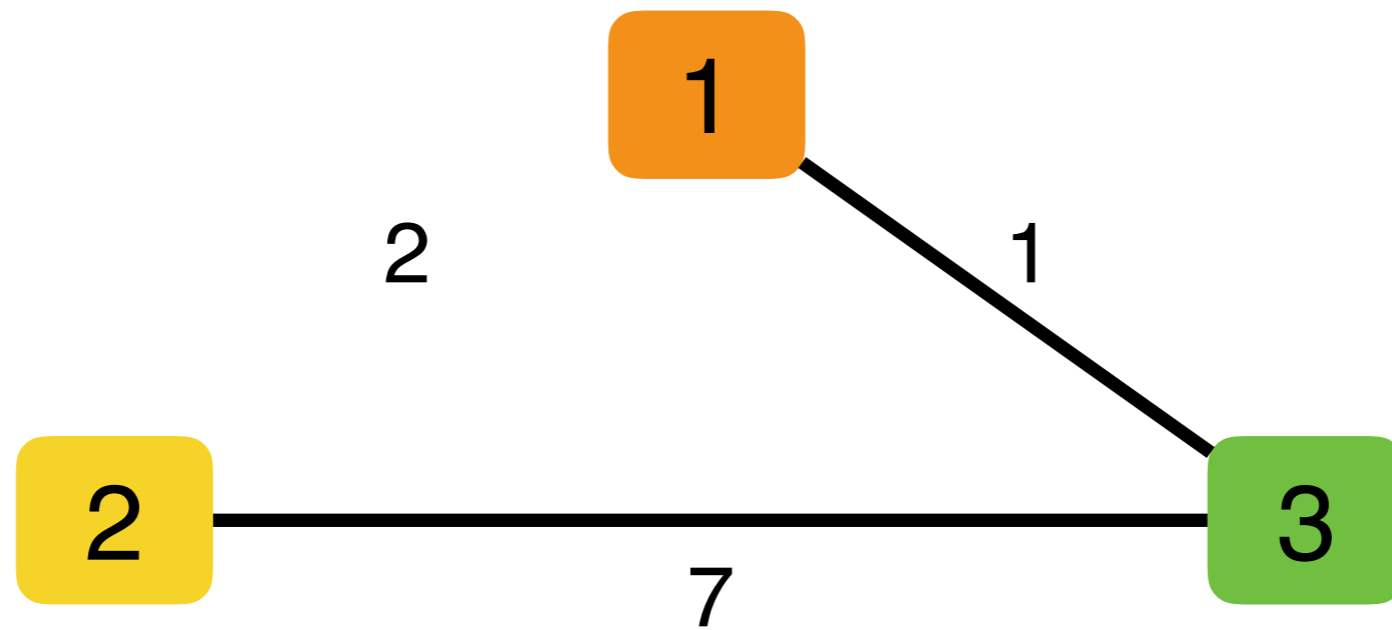


	distance	next-hop
1	1	1
2	5	1
3	0	-



# Round 4

	distance	next-hop
1	0	-
2	6	3
3	1	3

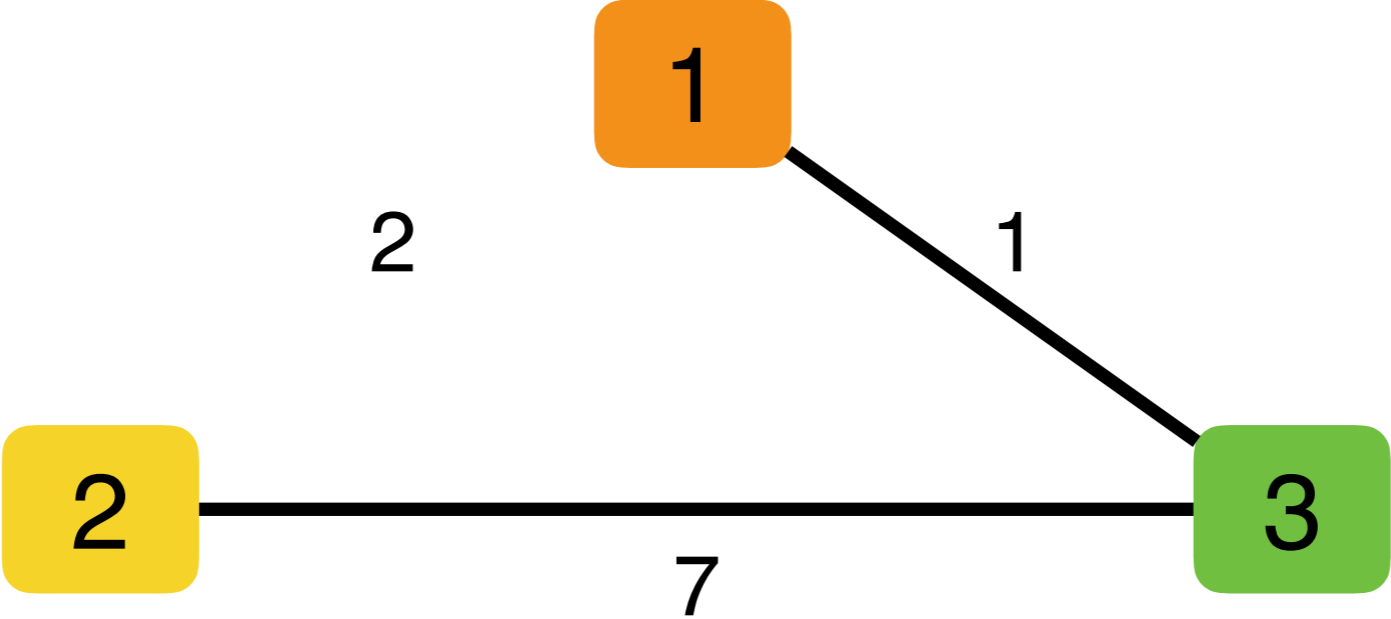


**COUNT-TO-INFINITY  
problem!!!!**

	distance	next-hop
1	1	1
2	7	1
3	0	-

# Count-to-infinity problem

	distance	next-hop
1	0	-
2	6	3
3	1	3



**Not just due to failures:  
Can happen with changes in cost!**

	distance	next-hop
1	1	1
2	7	1
3	0	-

# How Can You Fix This?

- **Do not advertise a path back to the node that is the next hop on the path**
  - Called “**split horizon**”
  - Telling them about your entry going through them
    - Doesn't tell them anything new
    - Perhaps misleads them that you have an independent path
- **Another solution: if you are using a next-hop's path, then:**
  - Tell them not to use your path (by telling them cost of infinity)
  - Called “**poisoned reverse**”

# Convergence

- Distance vector protocols can converge slowly
  - While these corner cases are rare
  - The resulting convergence delays can be significant

# Comparison of Scalability

- Link-State:
  - Global flood: each router's link-state (#ports)
  - Send it once per link event, or periodically
- Distance Vector:
  - Send longer vector (#dest) just to neighbors
    - But might end up triggering their updates
  - Send it every time DV changes (which can be often)
- Tradeoff:
  - LS: Send it everywhere and be done in predictable time
  - DV: Send locally, and perhaps iterate until convergence