CS4450, CS5456
Introduction to Computer Networks

Lecture 11

Rachee Singh

Recap: Routing

1. **Routing tables**:
    1. On each switch, map destinations to next-hops
2. **Routing state**:
    1. Collection of routing tables across all switches
3. Routing state is **valid** if and only if:
    1. No dead ends
    2. No loops
4. How do we **verify** if given routing state is valid?
5. Today: how can we **produce** valid routing state?

Goals of routing

1. Goal 1: *valid* routing in the network
    1. How to know if the state of routers' routing tables is *valid*?
2. Goal 2: *efficient* routing in the network
    1. Finding a *least cost path* to a given destination

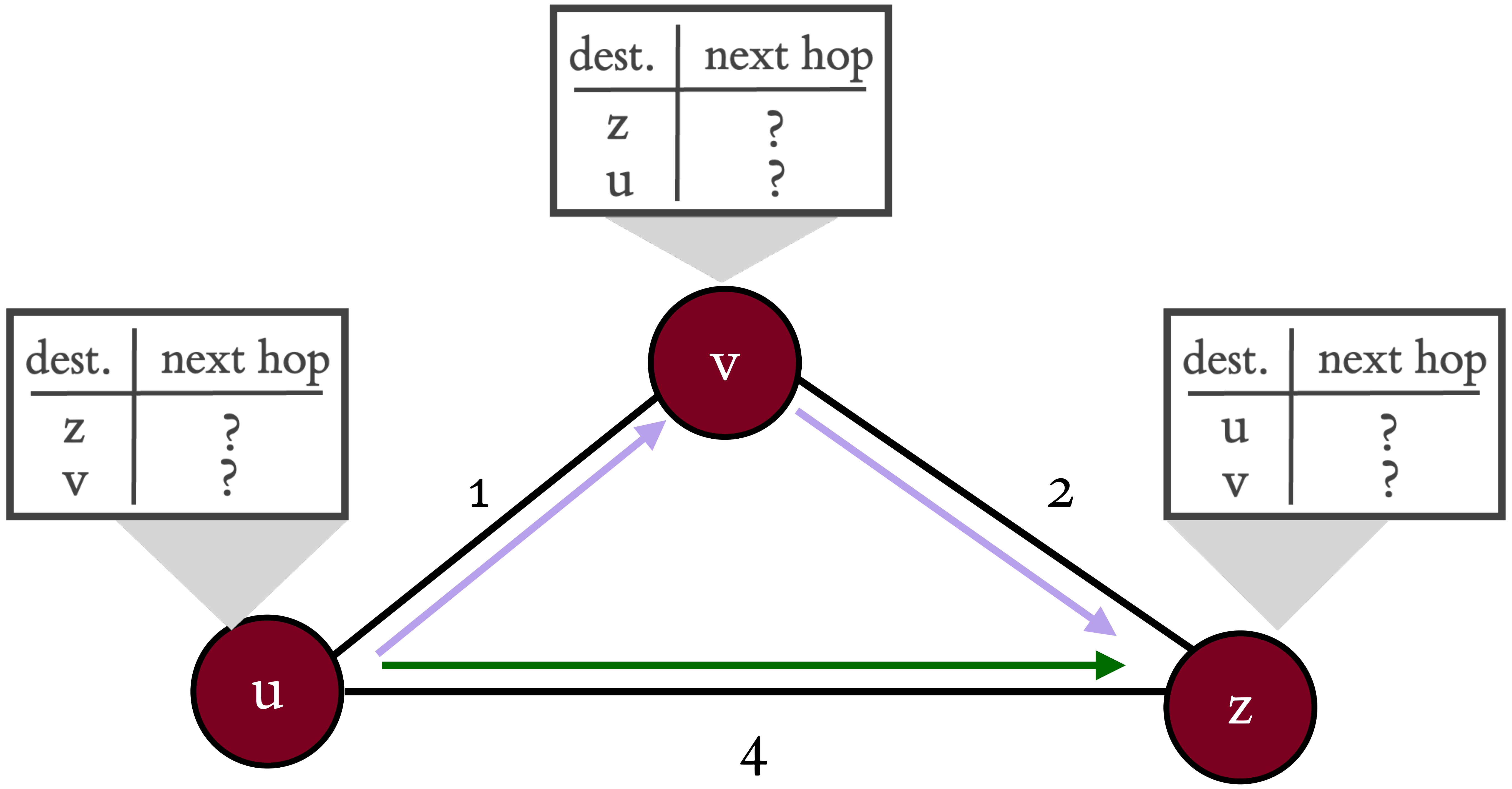
“Cost” in routing

1. Least cost routing tries to find paths with minimum X
2. What can X be?
   1. Latency
   2. Number of hops in the path
   3. Weight
   4. Failure probability
   5. …
3. Assume each link has some cost
4. We want to minimize the cost of paths
   1. Cost of a path = sum of the costs of links on the path

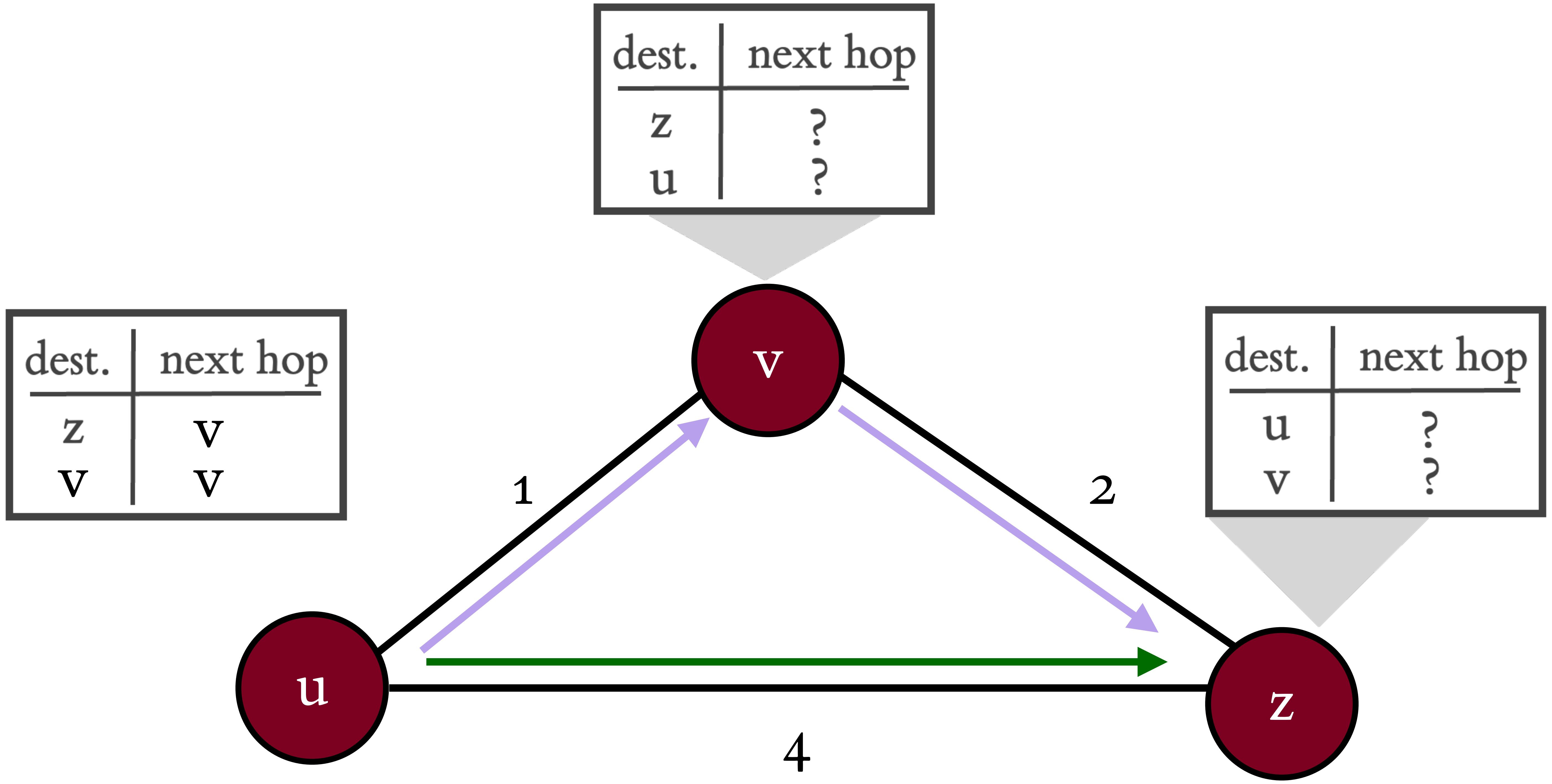
Least cost path routing

1. Approach 1: Link state routing
2. Approach 2: Distance vector routing

Least cost path routing



Least cost path routing



Least cost u—> z path: u—>v —>z

Least cost u—>v path: u->v

Least cost path routing

1. Given: router graph and link cost
2. Goal: find least cost paths
    1. From each source router
    2. To each destination router
3. How do you find least cost paths from a source to ALL destinations?
    1. Dijkstra's algorithm

Least cost routes

1. Least cost routes automatically avoid loops
    1. No sensible cost metric is minimized by traversing loops
    2. Least cost routes end up forming **spanning trees** to the destination

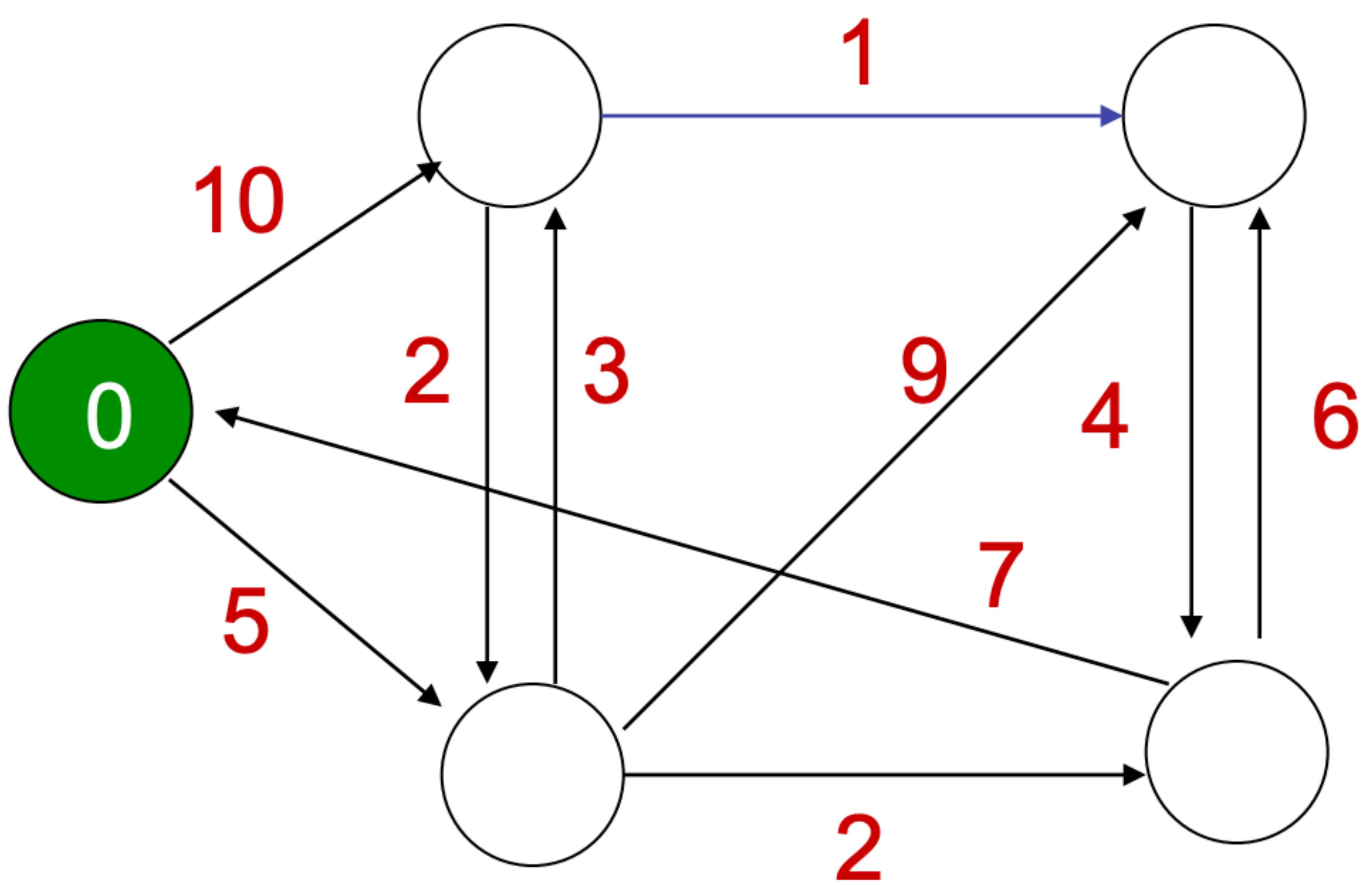
Link state routing: protocol vs. algorithm

1. Link state routing protocol **creates a global view** of the network
    1. Where to create the global view?
    2. How to create the global view?
    3. When to run route computation?
2. Algorithm **finds shortest paths** on the global network view
    1. Create shortest paths using standard algorithms

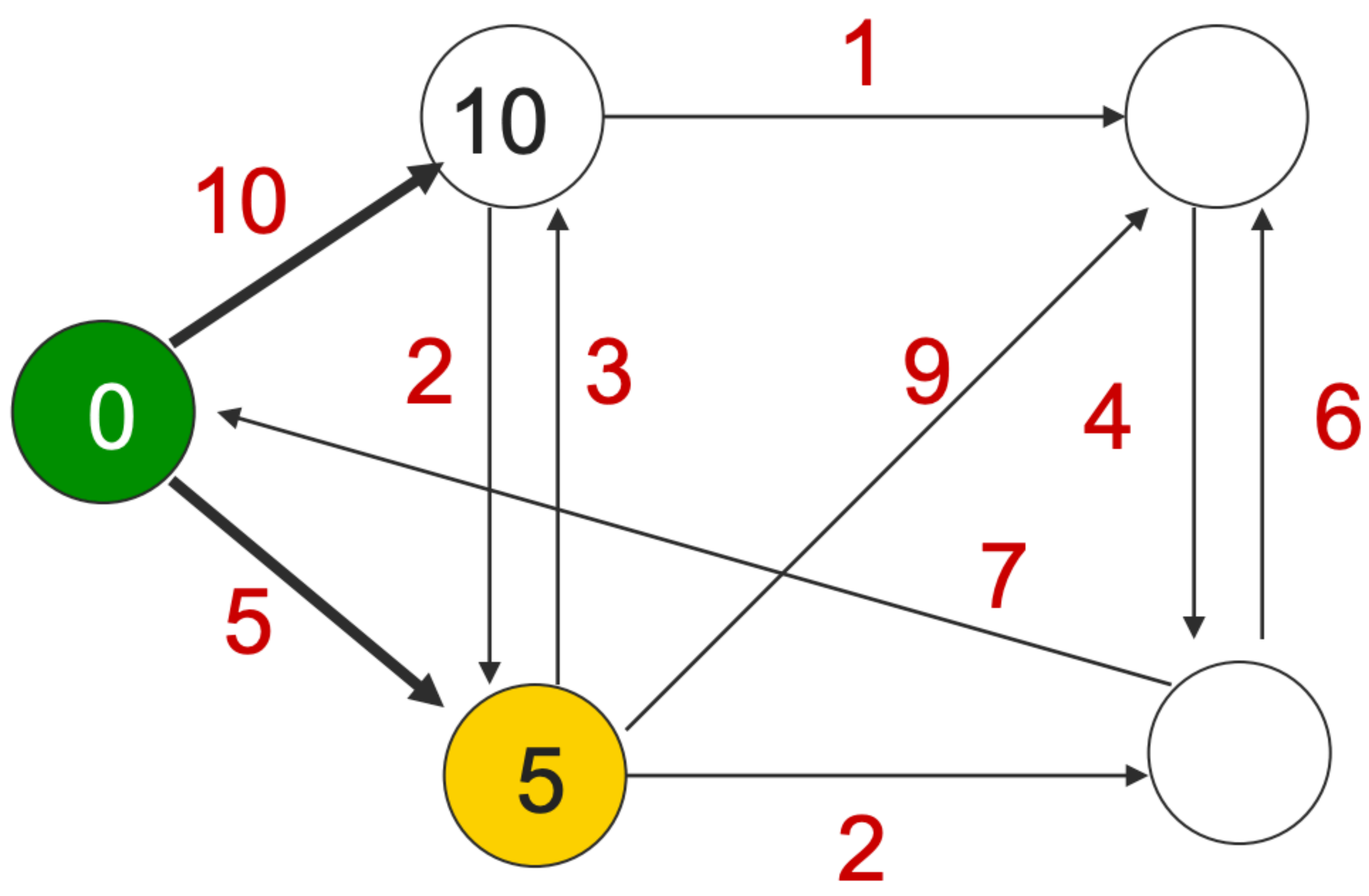
Link State Routing: Dijkstra’s shortest path algorithm

```
function Dijkstra(Graph, source):

    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q

        for each neighbor v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

Link State Routing: Dijkstra’s shortest path algorithm

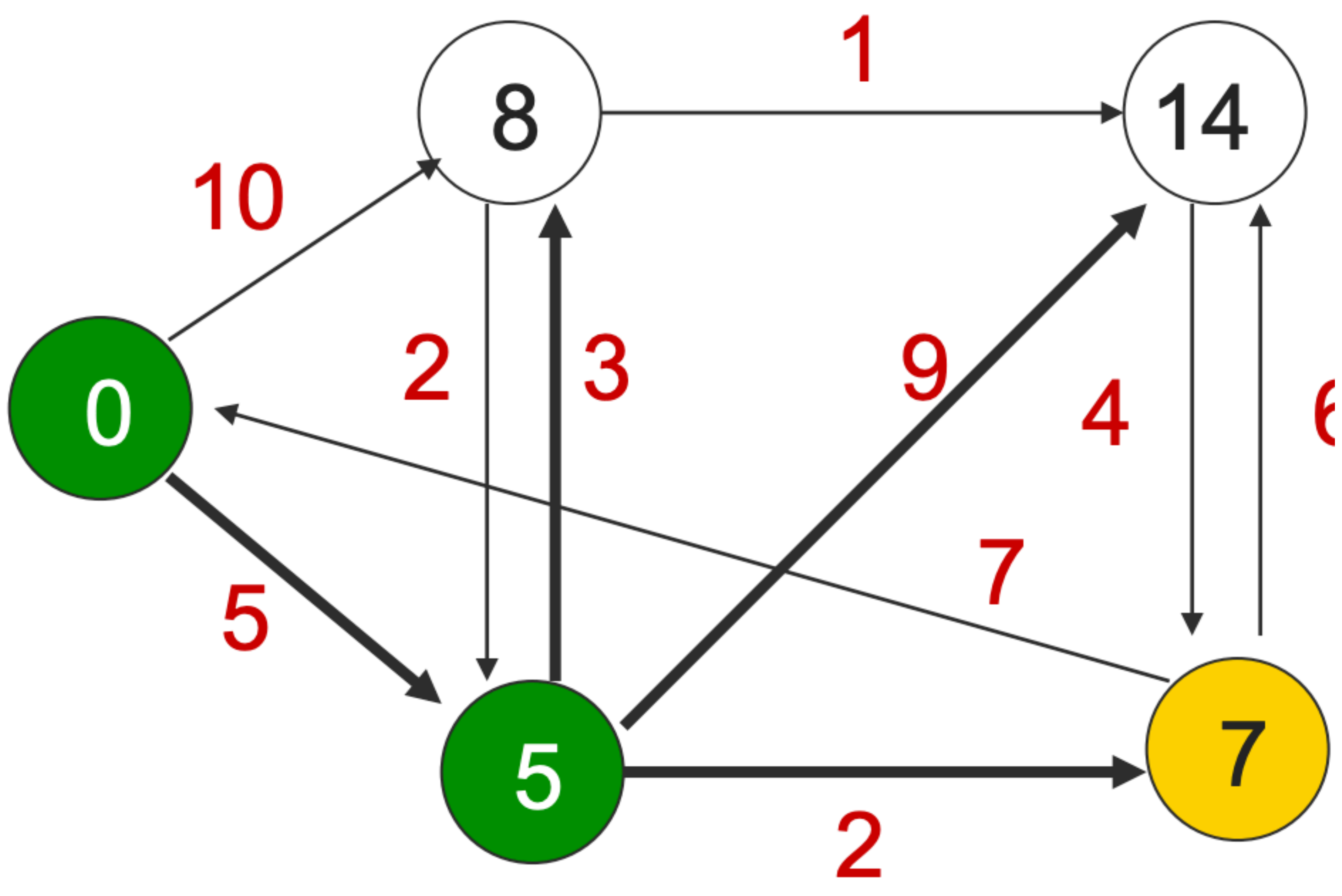


Step 1

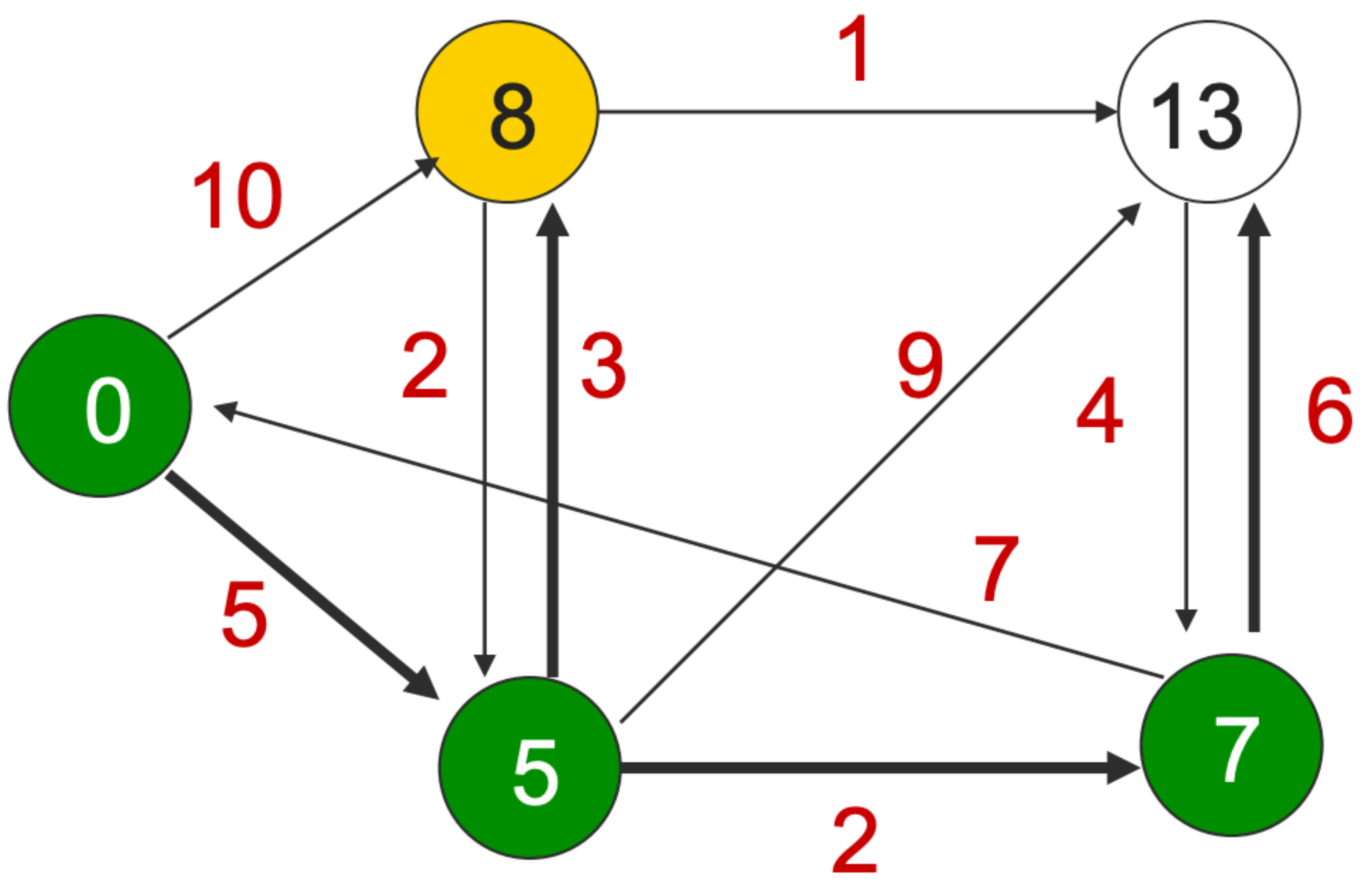


Step 2

Link State Routing: Dijkstra’s shortest path algorithm

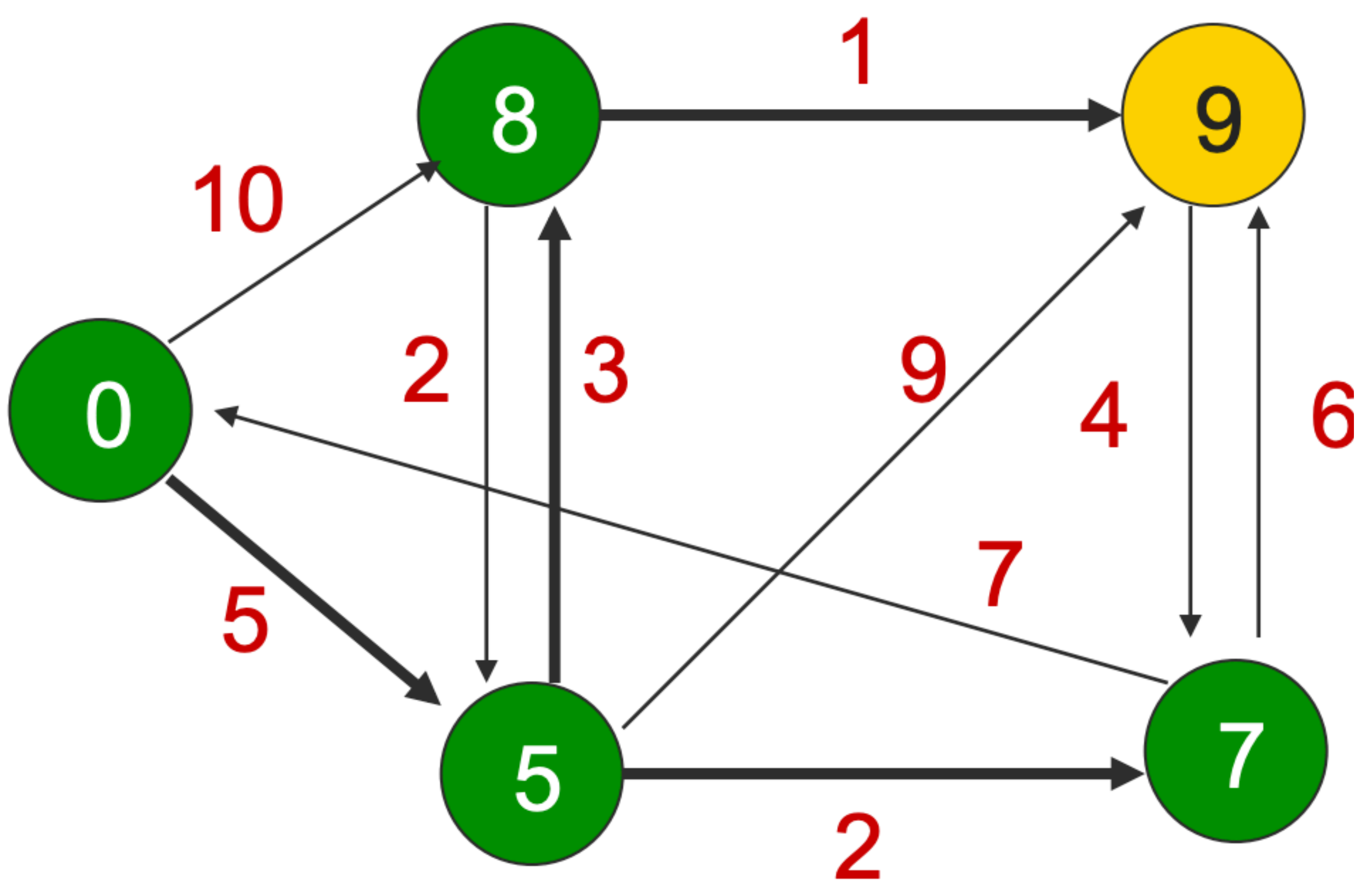


Step 3

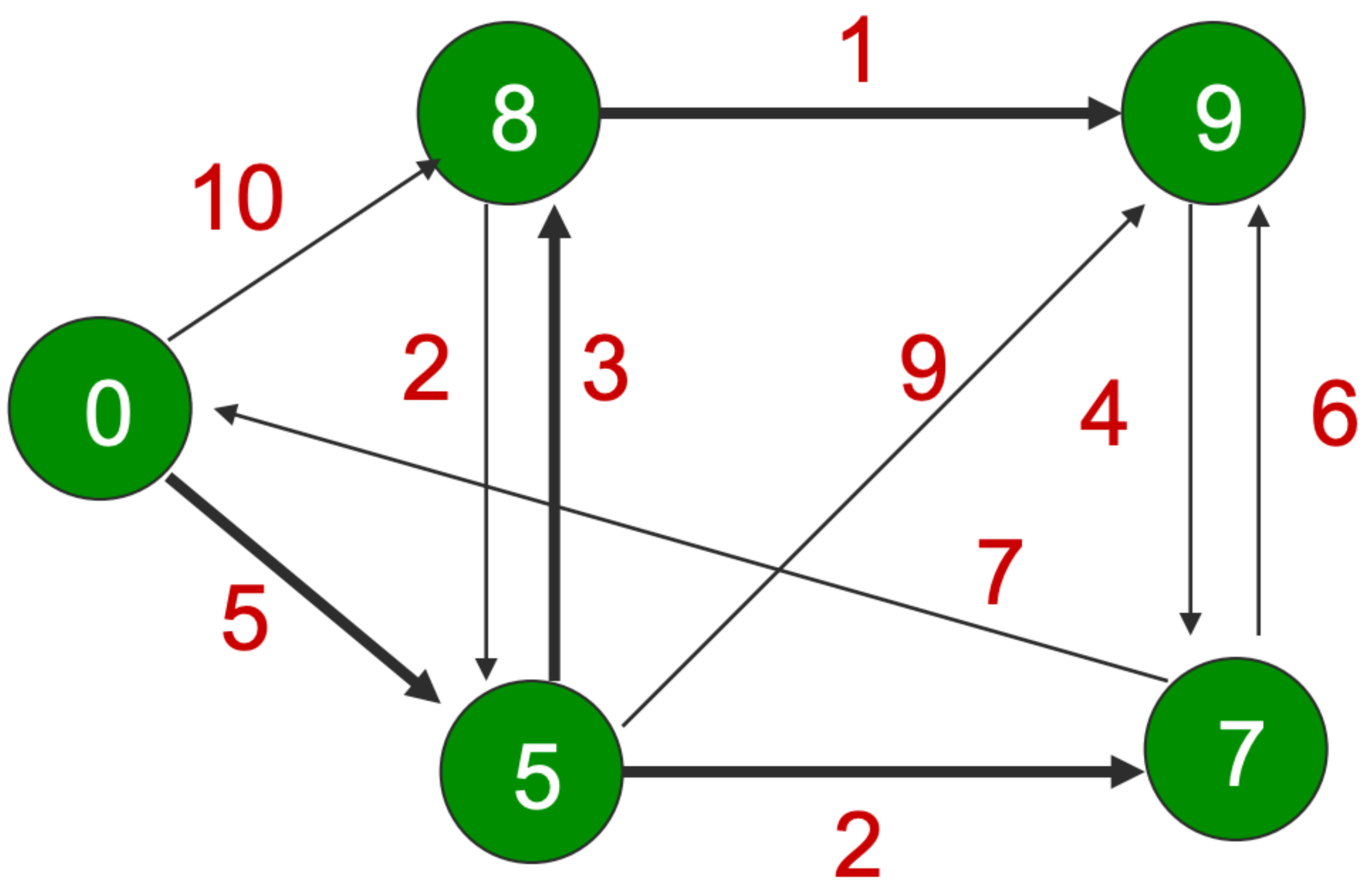


Step 4

Link State Routing: Dijkstra's shortest path algorithm



Step 5



Step 6

Where to create the global view?

1. Option 1: Centralized server
    1. One machine (server?) collects information from routers
    2. Makes a global graph
    3. Software-defined networking uses this mechanism (later in the course)
2. Option 2: At every router
    1. Each router makes a global view
    2. The Internet uses this mechanism
3. Link state routing protocol
    1. Example: OSPF (Open shortest path first)
    2. IETF RFC 2328 (IPv4) or 5340 (IPv6)

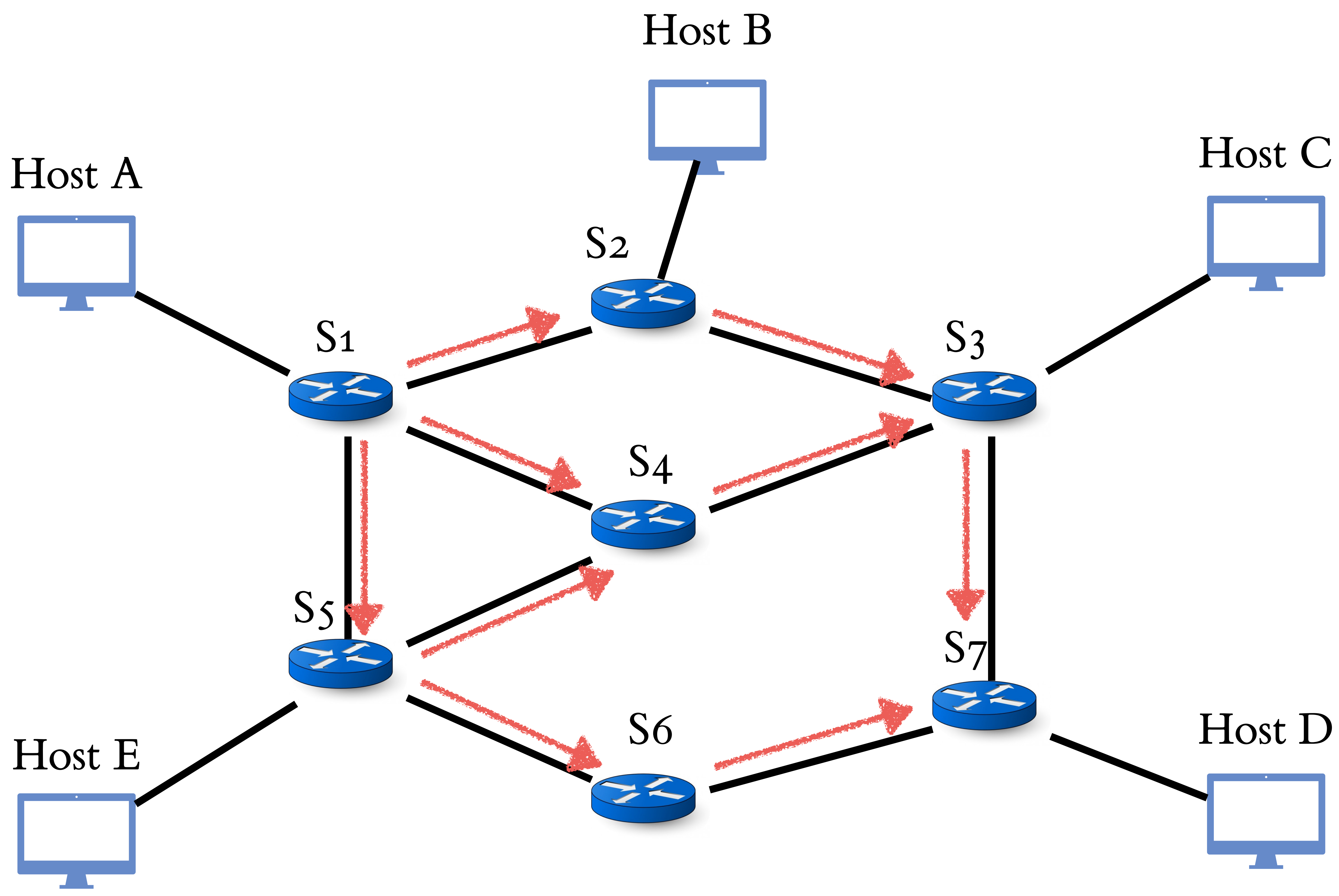
Link State Routing

1. Part 1: **Link state flooding**:
    1. Every router knows its local "link state" (links to neighbors, costs)
    2. Flood local link state to all other routers
    3. Each router builds a global view of the graph
2. Part 2: Local **Path calculation**:
    1. Each router runs Dijkstra's algorithm to compute shortest path trees
    2. Each router uses the shortest path tree to populate routing tables
3. Global view of the network allows optimal route computation

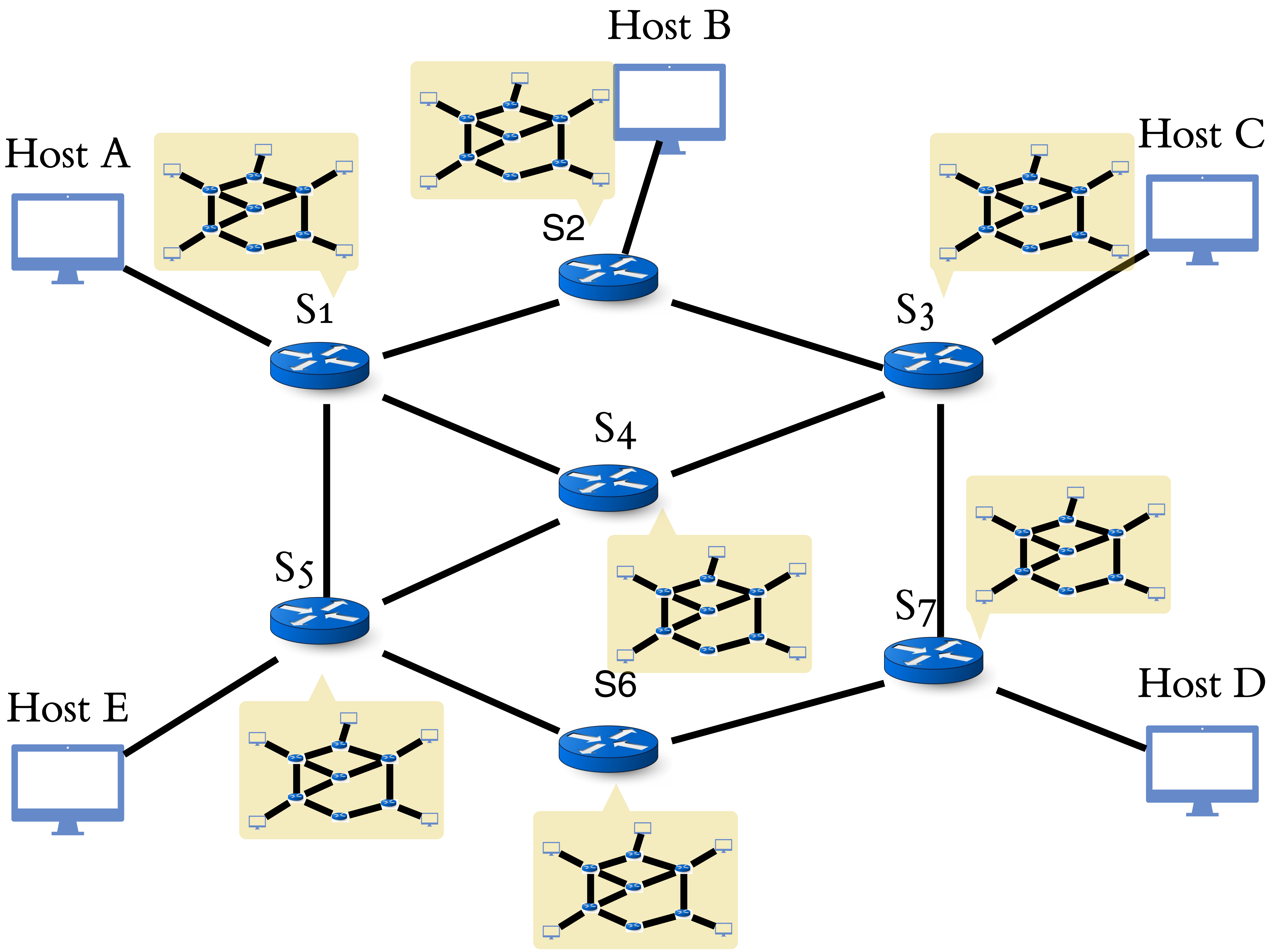
Link State Routing: flooding link state

1. The packets that flood local link state are called Link State Advertisements (LSAs)
2. When an LSA arrives at a router
    1. It remembers the packet
    2. Forwards it to all other routers
    3. Does not send it out on the incoming link
        1. Why?
3. If the same LSA arrives again
    1. Drop it, do not forward again

Link State Routing: flooding link state



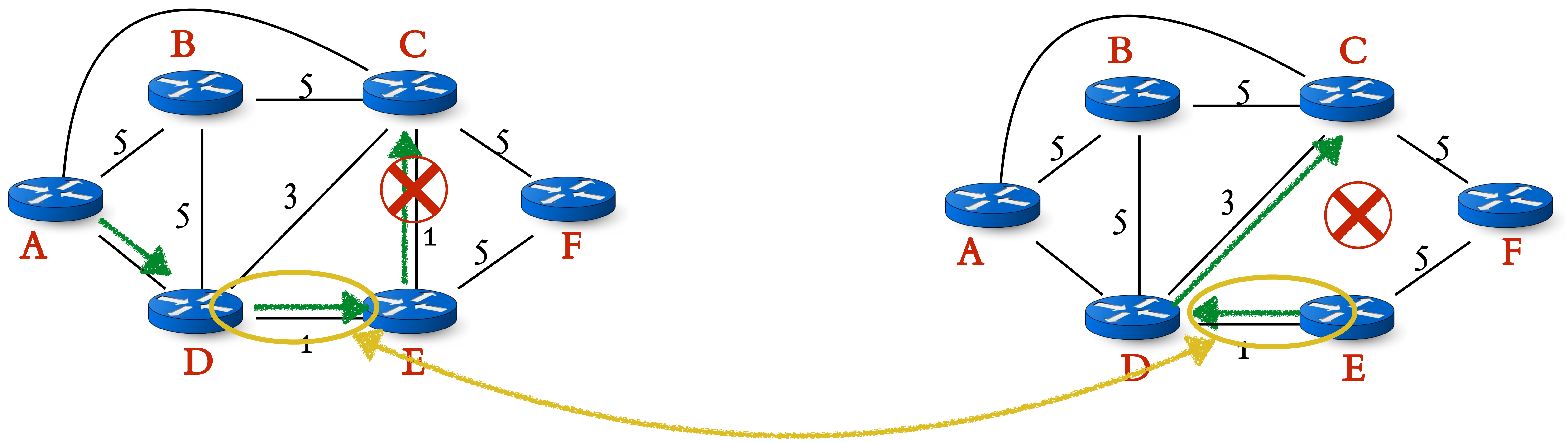
Link State Routing: flooding link state



When should LSAs be sent out?

1. Topology change
    1. Link failure
    2. Link recovery
2. Configuration change
    1. Change in link cost
3. Periodically
    1. Refresh link state information
    2. Every few minutes
    3. Corrects for possible corruption of data

Can loops happen in least cost routing?



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!

Inconsistent link state views between routers
cause loops

Convergence

1. Eventually all routers will have a consistent view of the network graph
    1. All routers have the same **link state database**
    2. Eventually means nothing has changed for a while
2. Forwarding is consistent after convergence
    1. All nodes have the same link state db
    2. All nodes forward packets on the same least cost paths
3. But while convergence has not been achieved
    1. Bad things can happen (like loops)

Convergence Time

1. How long does it to reach convergence?
2. What does it depend on?
    1. Time to detect failures
    2. Time to flood link state information to everyone (~ longest RTT)
    3. Time to recompute shortest paths and forwarding tables
3. Until convergence is reached
    1. Loops can happen
    2. Dead ends can happen

Scalability of Link State Routing

1. Are loops possible?
   1. Yes, until convergence
2. Scalability:
   1. O(NE) messages
   2. $O(N^2)$ computation per router
   3. O(Network diameter) + $O(N^2)$ convergence time
   4. O(N) entries in forwarding table

Least cost path routing

1. Approach 1: Link state routing
2. Approach 2: Distance vector routing

Activity: find the tallest person in the class

1. Rules:
    1. You can talk to your neighbors to exchange information
    2. You can not get up and shout
    3. You can not leave your seat
2. At the end of class:
    1. I will ask different people who they think is the tallest person
    2. Also tell me who told you about this person

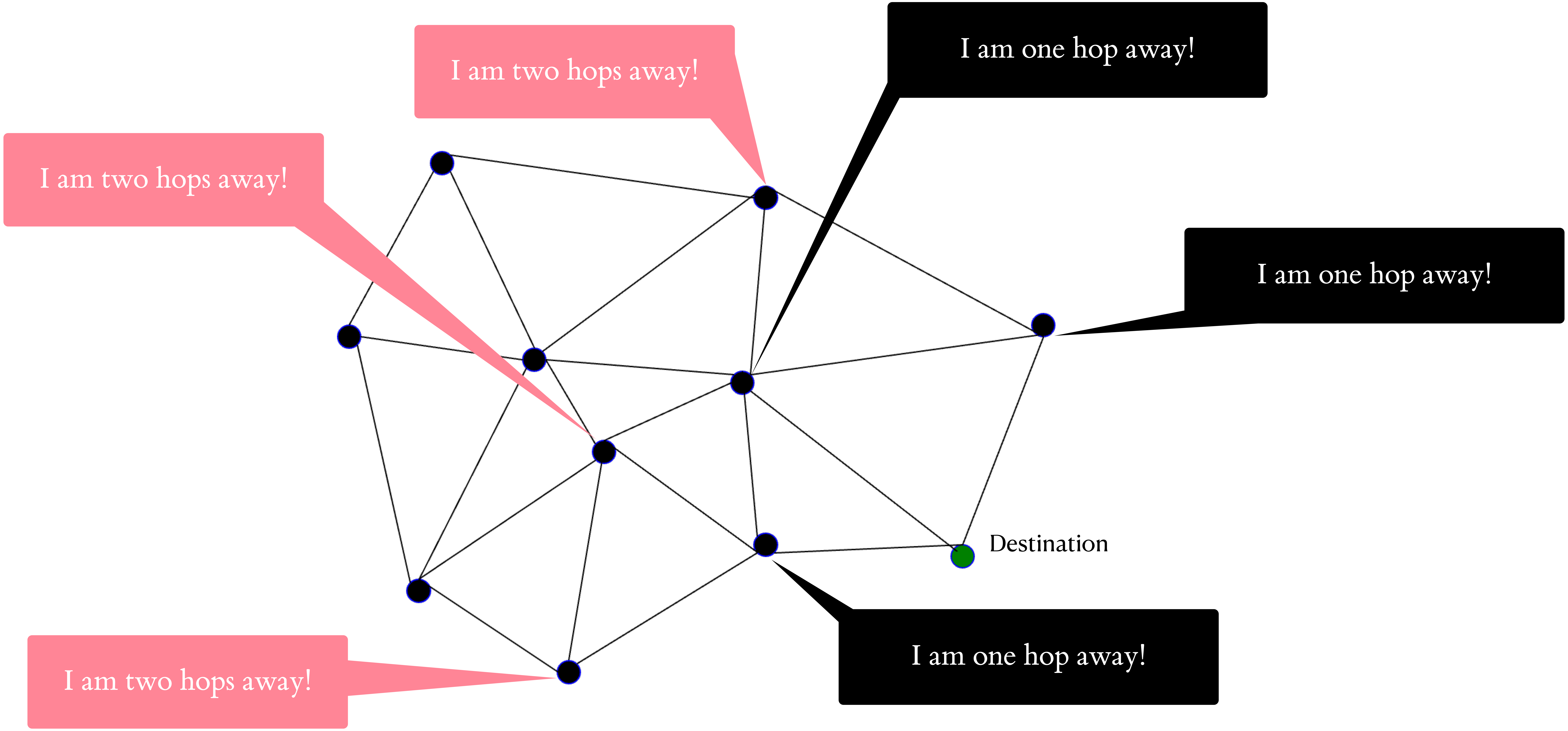
Distance vector routing

1. Distributed algorithm
2. All routers run it “together”
3. Input to each router:
    1. Link costs and neighbor messages
4. Output of each router:
    1. Least cost path to every other router

Distance vector routing: Bellman-Ford algorithm

1. All neighbors exchange information
    1. Each router checks if it can improve current paths
2. End when no improvement is possible

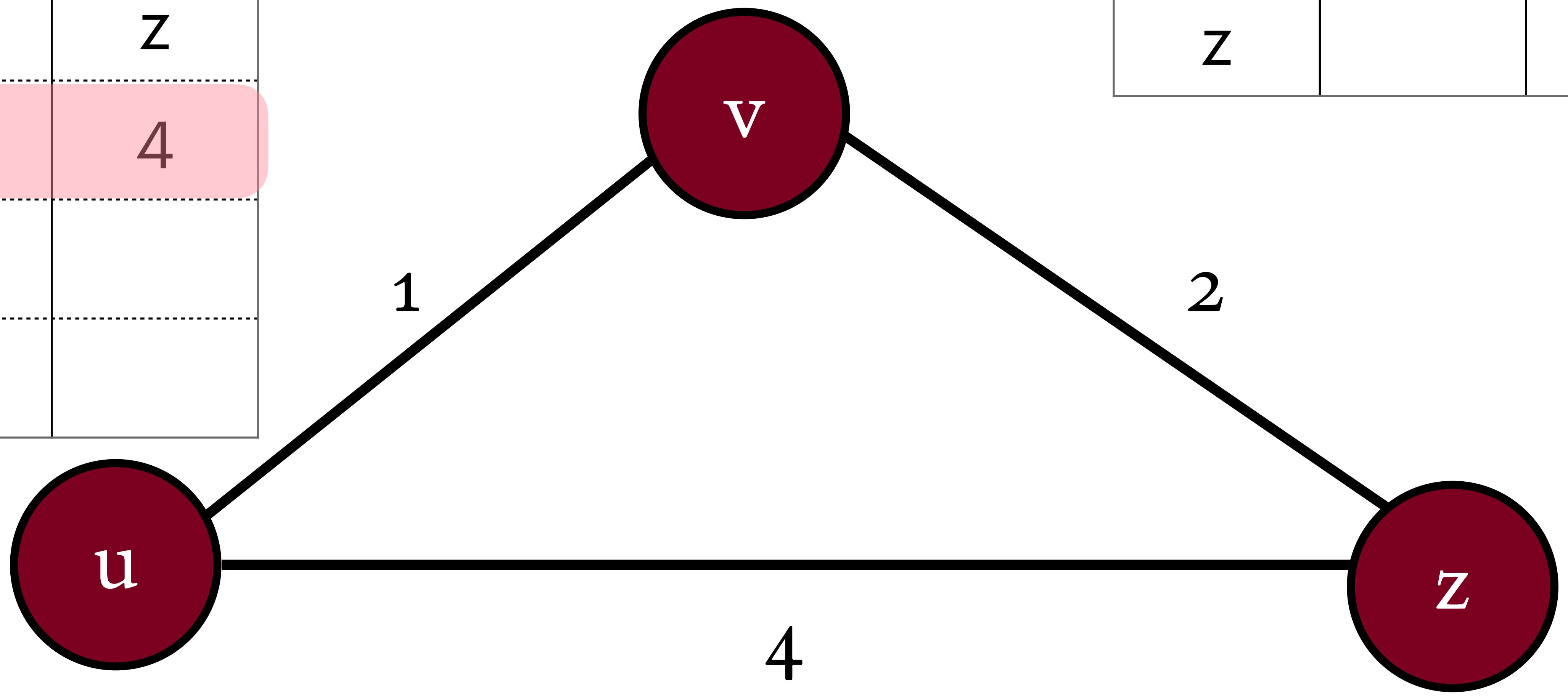
Example of distributed computation



Example of distributed computation

Distance Vector

	u	v	z
u	0	1	4
v			
z			



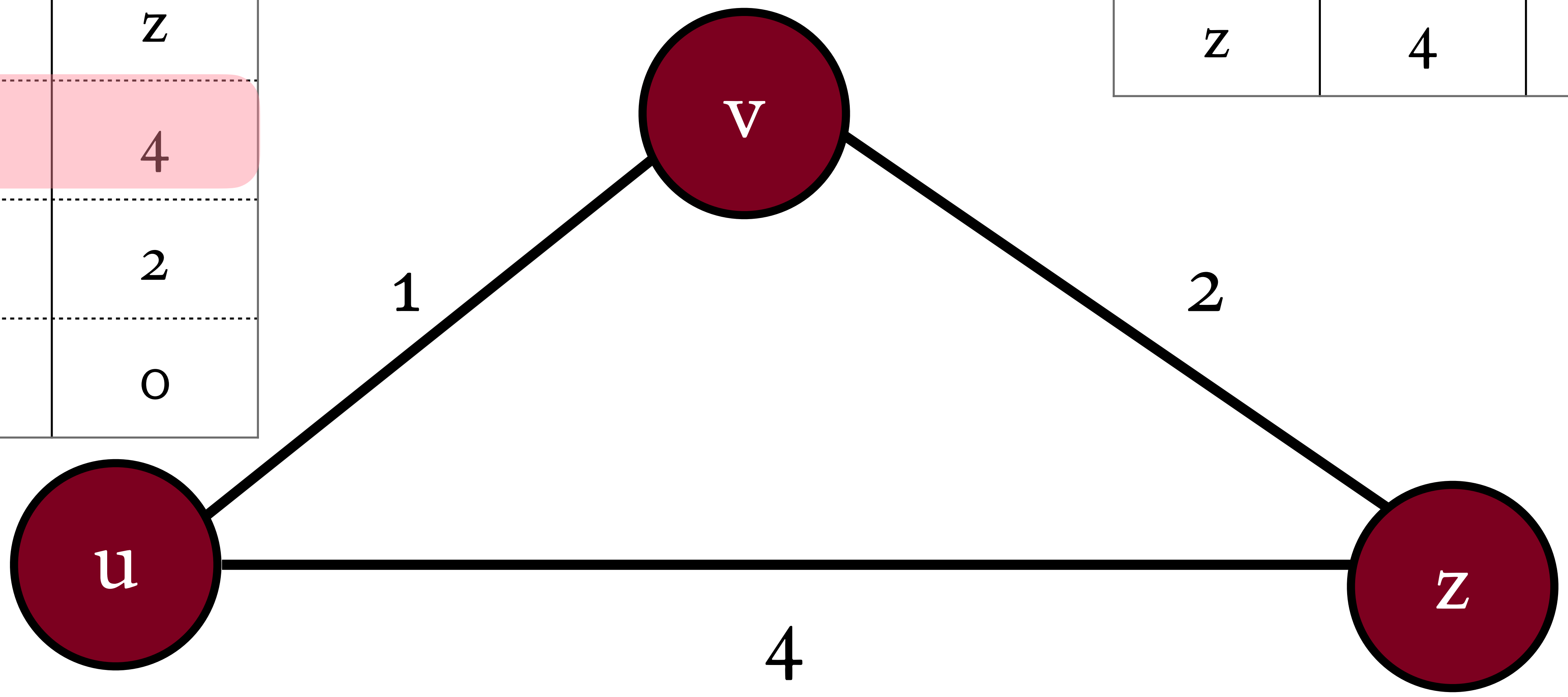
	u	v	z
u			
v	1	0	2
z			

	u	v	z
u			
v			
z	4	2	0

Example of distributed computation

Can I change my estimate of costs?

	u	v	z
u	0	1	4
v	1	0	2
z	4	2	0



	u	v	z
u	0	1	4
v	1	0	2
z	4	2	0

	u	v	z
u	0	1	4
v	1	0	2
z	4	2	0

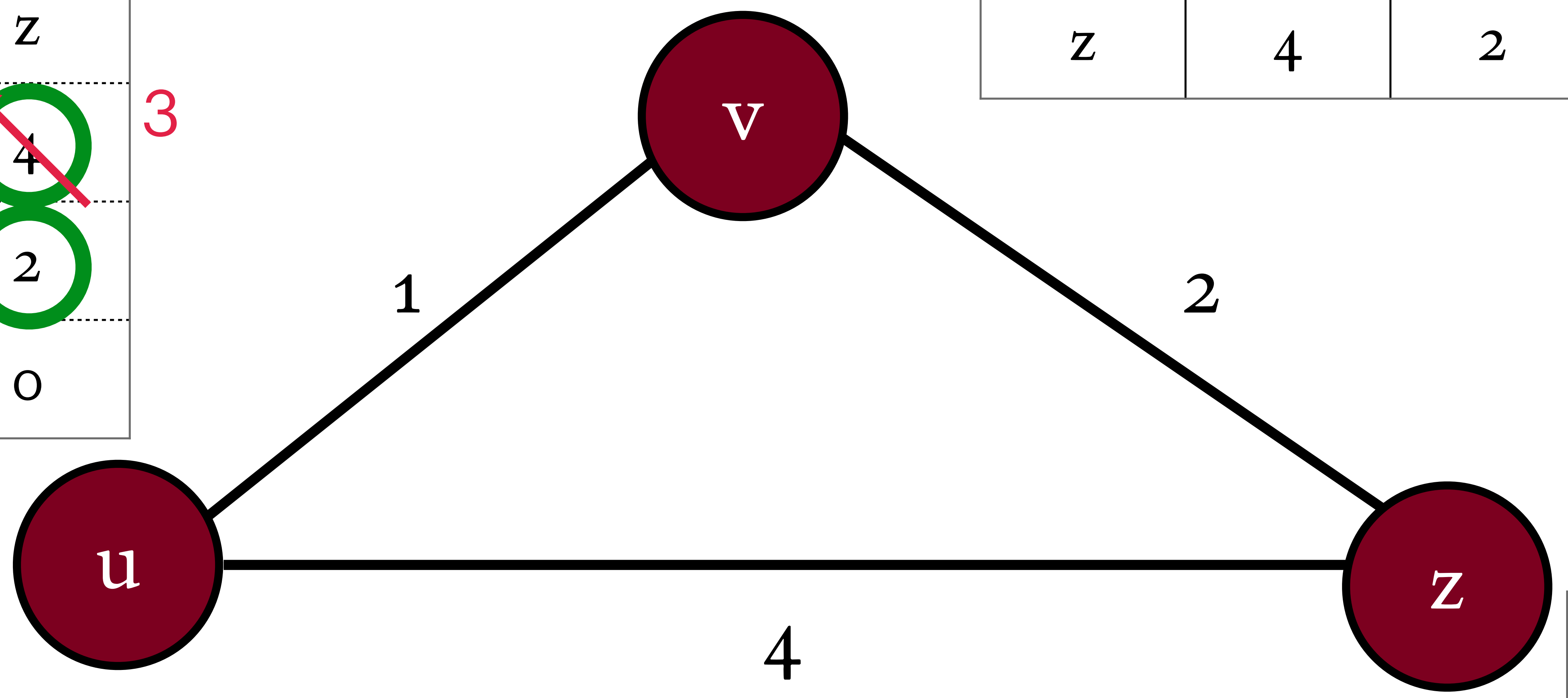
$D_x(y)$: Distance to y from x
$C(x, v)$: Cost of edge (x, v)

Bellman-Ford Equation
$D_x(y) = \min \{ C(x,v) + D_v(y)\}$ for each node $v \in N$

Example of distributed computation

	u	v	z
u	0	1	4
v	1	0	2
z	4	2	0

	u	v	z
u	0	1	~~4~~ 3
v	1	0	2
z	4	2	0



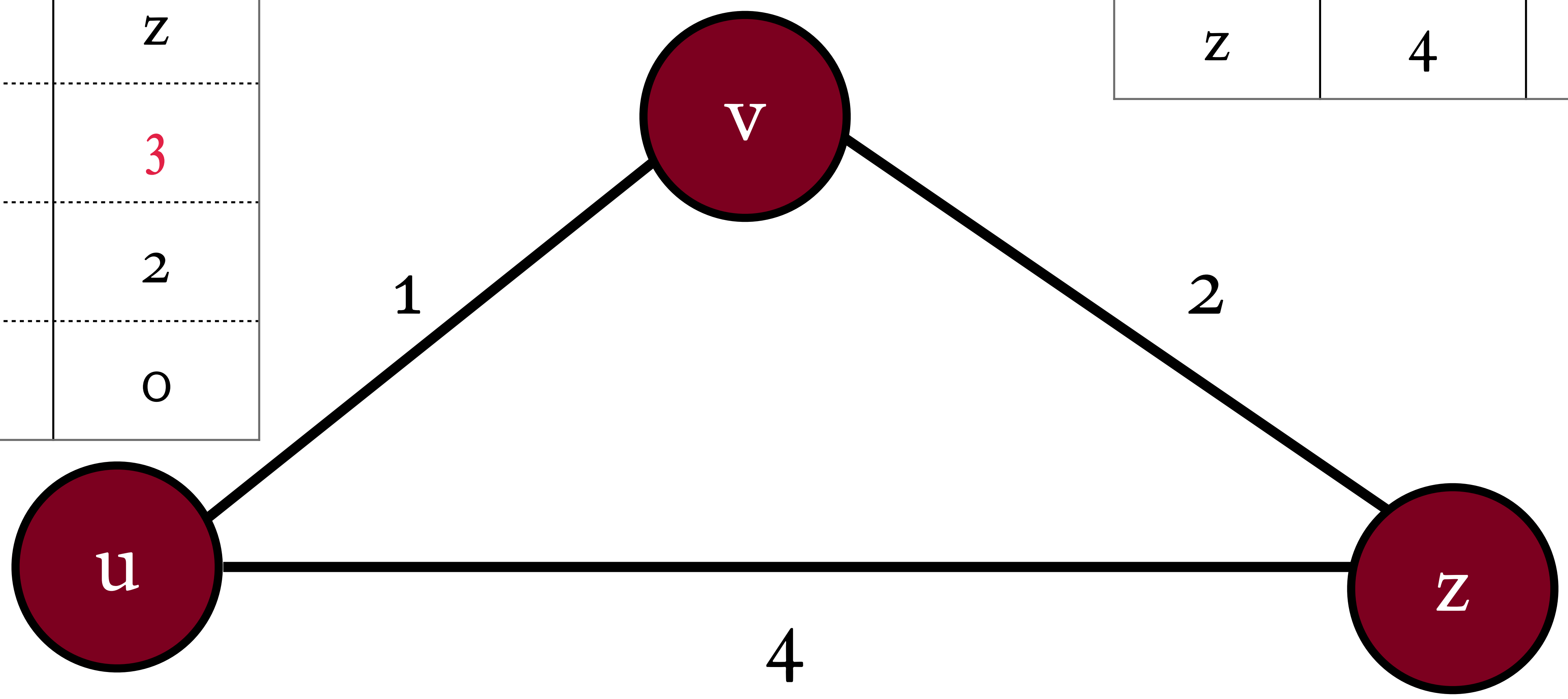
	u	v	z
u	0	1	4
v	1	0	2
z	4	2	0

$D_x(y) = \min\{ C(x,v) + D_v(y)\}$ for each node $v \in N$

Bellman-Ford Equation

Example of distributed computation

	u	v	z
u	0	1	3
v	1	0	2
z	4	2	0



	u	v	z
u	0	1	4
v	1	0	2
z	4	2	0

	u	v	z
u	0	1	4
v	1	0	2
z	4	2	0

$D_x(y)$: Distance to y from x

$C(x, v)$: Cost of edge (x, v)

Bellman-Ford Equation

$D_x(y) = \min \{ C(x,v) + D_v(y)\}$ for each node $v \in N$

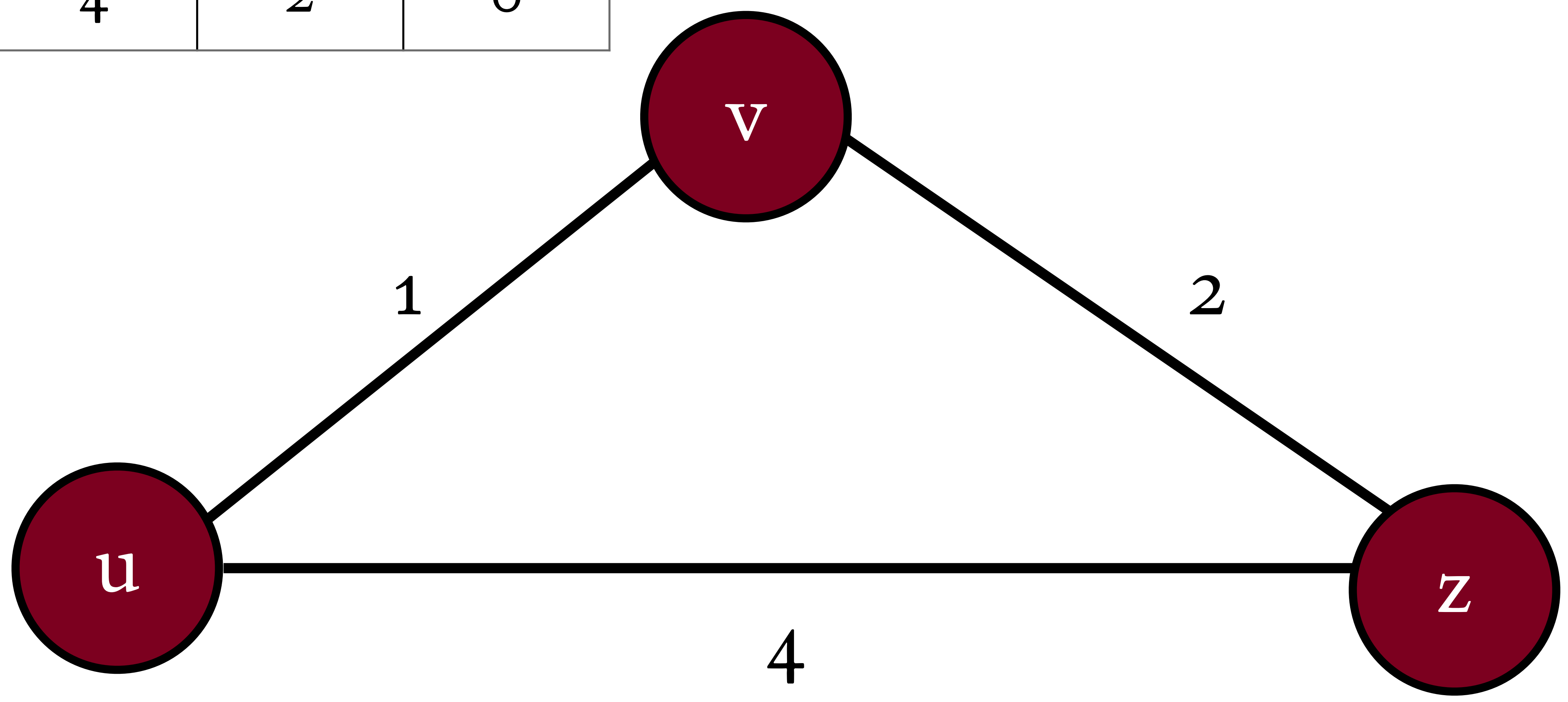
Example of distributed computation

	u	v	z
u	0	1	4
v	1	0	2
z	4	2	0

Bellman-Ford Equation

$D_x(y) = \min \{ C(x,v) + D_v(y) \}$ for each node $v \in N$

	u	v	z
u	0	1	3
v	1	0	2
z	4	2	0



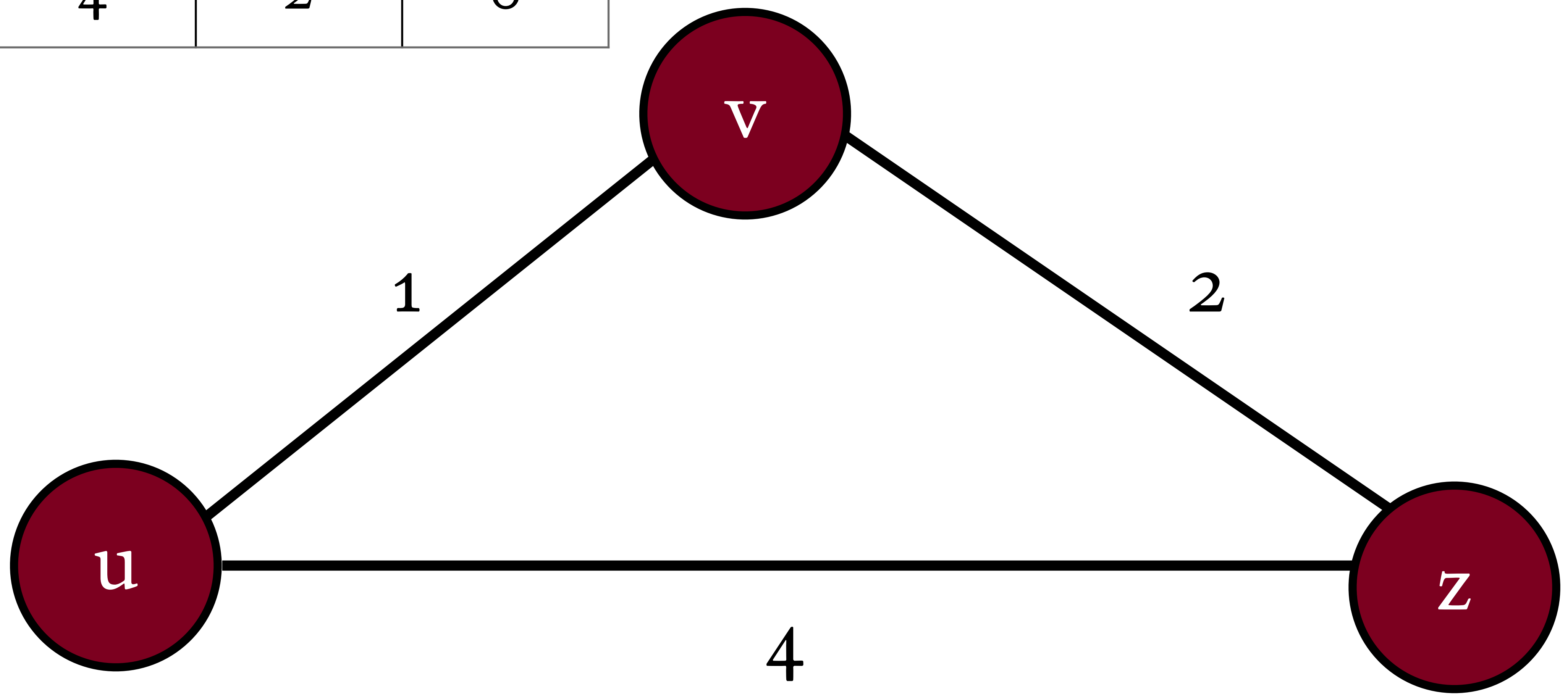
	u	v	z
u	0	1	4
v	1	0	2
z	~~4~~	2	0

3

Example of distributed computation

	u	v	z
u	0	1	4
v	1	0	2
z	4	2	0

	u	v	z
u	0	1	3
v	1	0	2
z	4	2	0

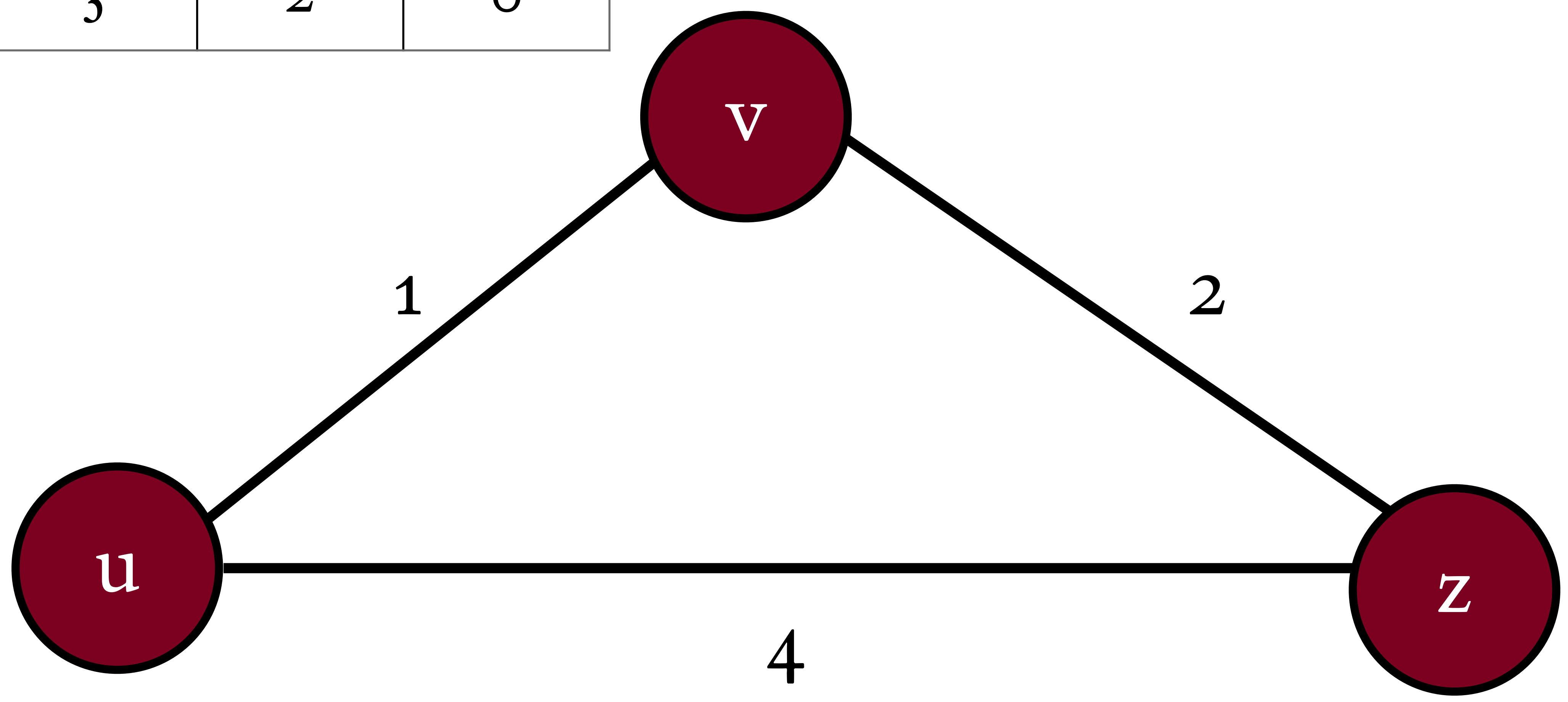


	u	v	z
u	0	1	4
v	1	0	2
z	3	2	0

Convergence

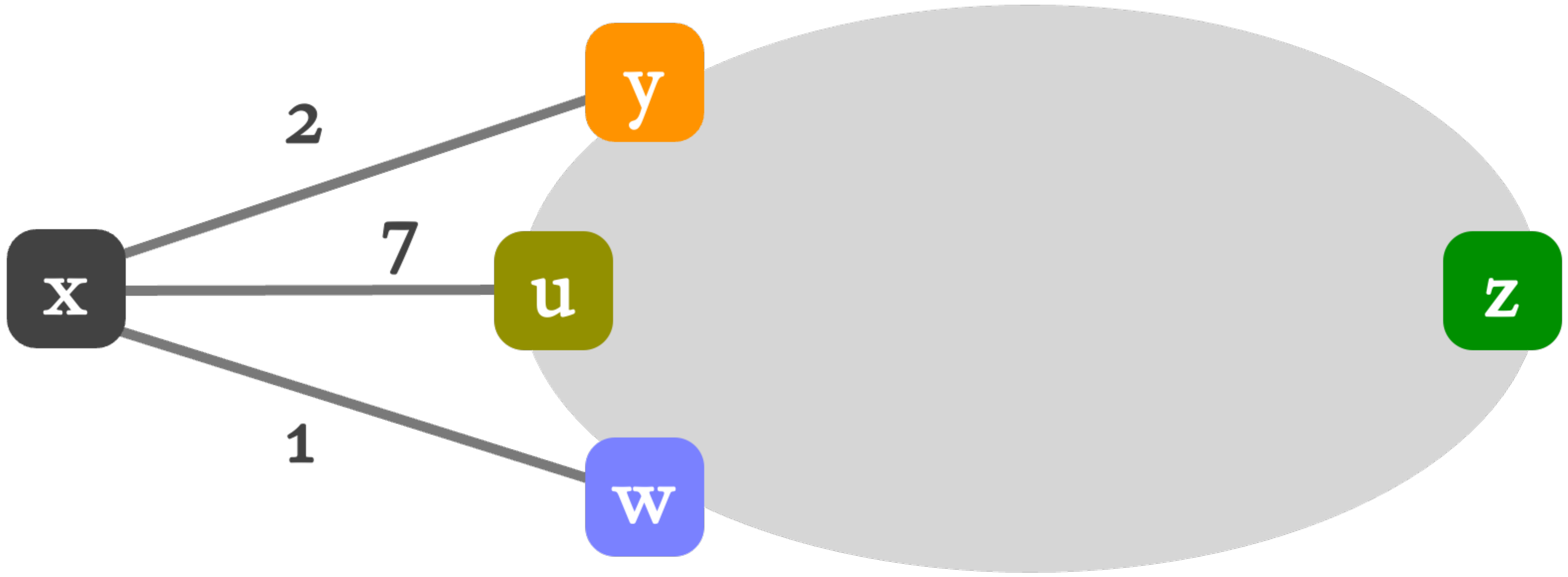
	u	v	z
u	0	1	3
v	1	0	2
z	3	2	0

	u	v	z
u	0	1	3
v	1	0	2
z	3	2	0



	u	v	z
u	0	1	3
v	1	0	2
z	3	2	0

Bellman-Ford Equation



$$d_x(z) = \min\{\ cost(x,y) + d_y(z),\ cost(x,u) + d_u(z),\ cost(x,w) + d_w(z)$$

Bellman-Ford Equation

Bellman-Ford Equation

$D_x(y) = \min \{ C(x,v) + D_v(y) \}$ for each node $v \in N$

1. Formalizes the decision:
    1. Pick as the next-hop for destination z, the neighbor that results in the least cost path to z