The Network Layer: Routing

CPSC 433/533, Spring 2021

Anurag Khandelwal

Administrivia

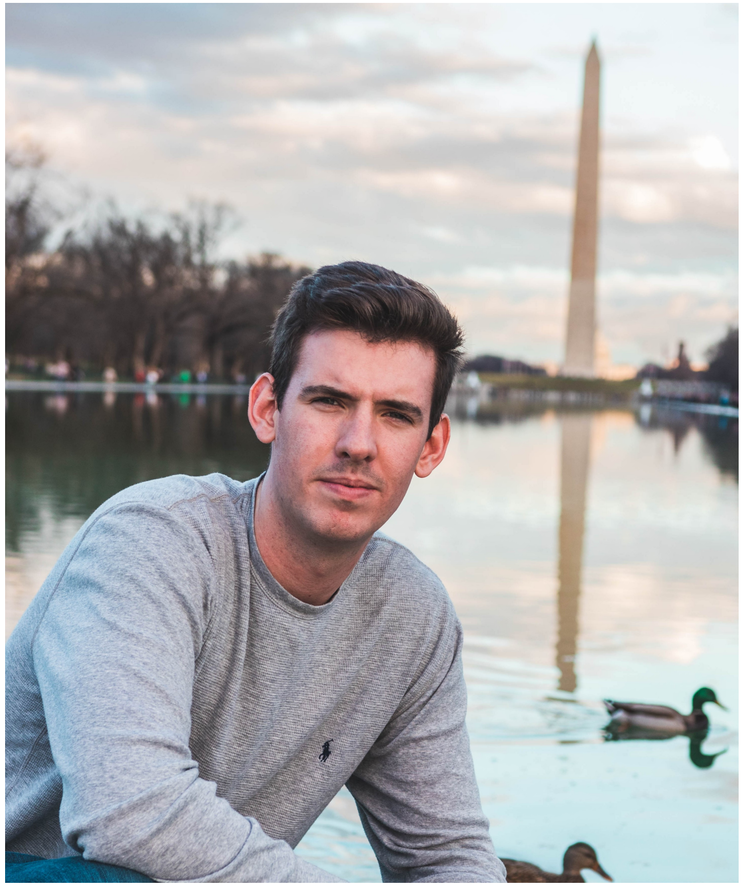
- Have to shift my OH from **4-5pm ET** to **5-6pm ET**
- ULA Introductions

Jonathan Kraft

- **Senior** in Davenport College
- Joint major in **CS** and **Mathematics**
- Office hours: **Wed 11:00am - 12:00pm ET**
  - Zoom (see Canvas for link)
- **jonathan.kraft@yale.edu**

Josh Beasley

- **Senior** in Timothy Dwight College
- Major in **CS**
- Office hours: **Thu 3:00pm - 4:00pm ET**
  - Zoom (see Canvas for link)
- **josh.beasley@yale.edu**



Recap

Recap

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

Recap

- 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 protocol*

Recap

- **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 protocol*
- Routing will be our focus for the next couple of lectures

“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

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

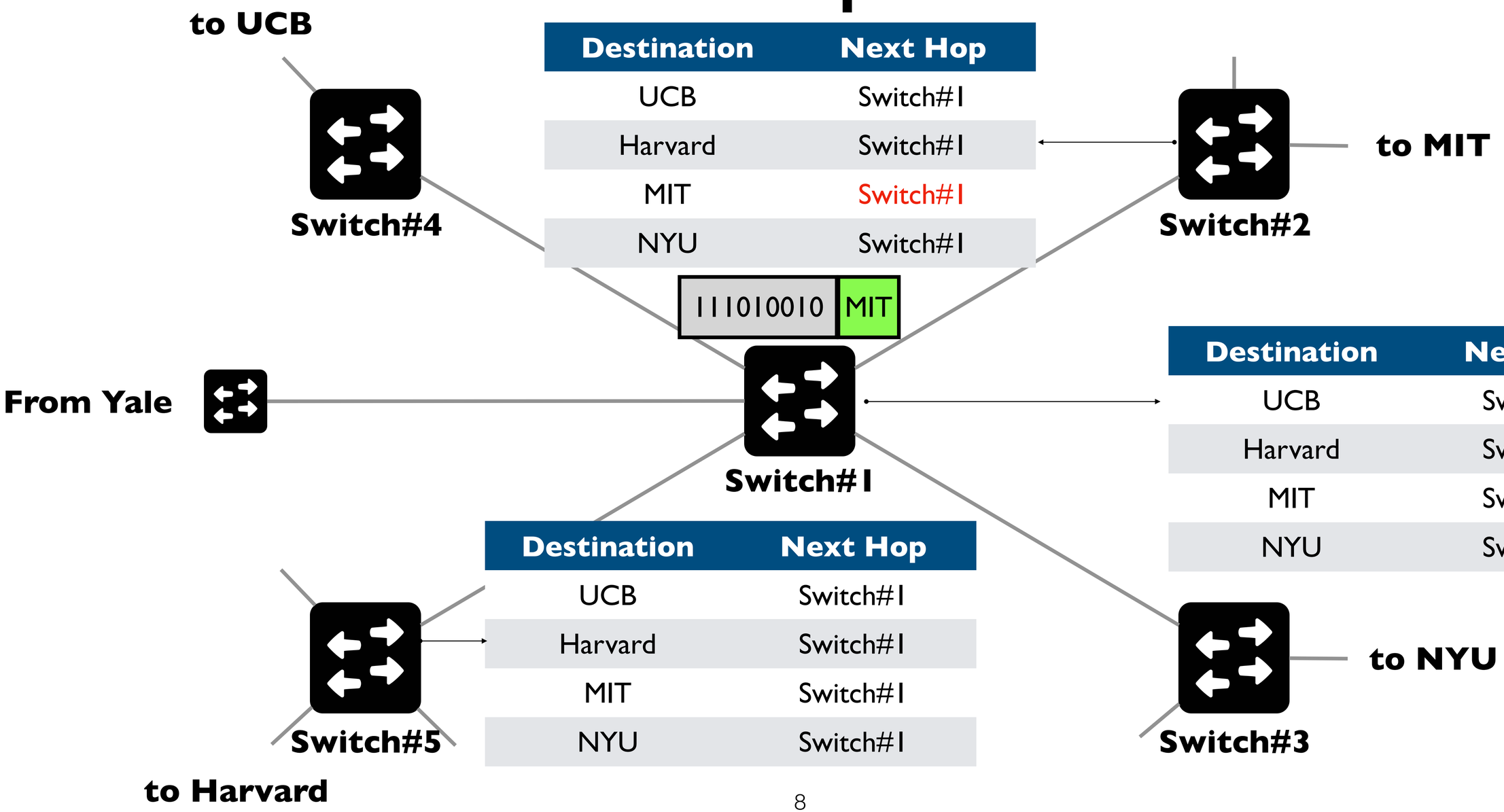
Necessary and Sufficient Condition

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

Loops

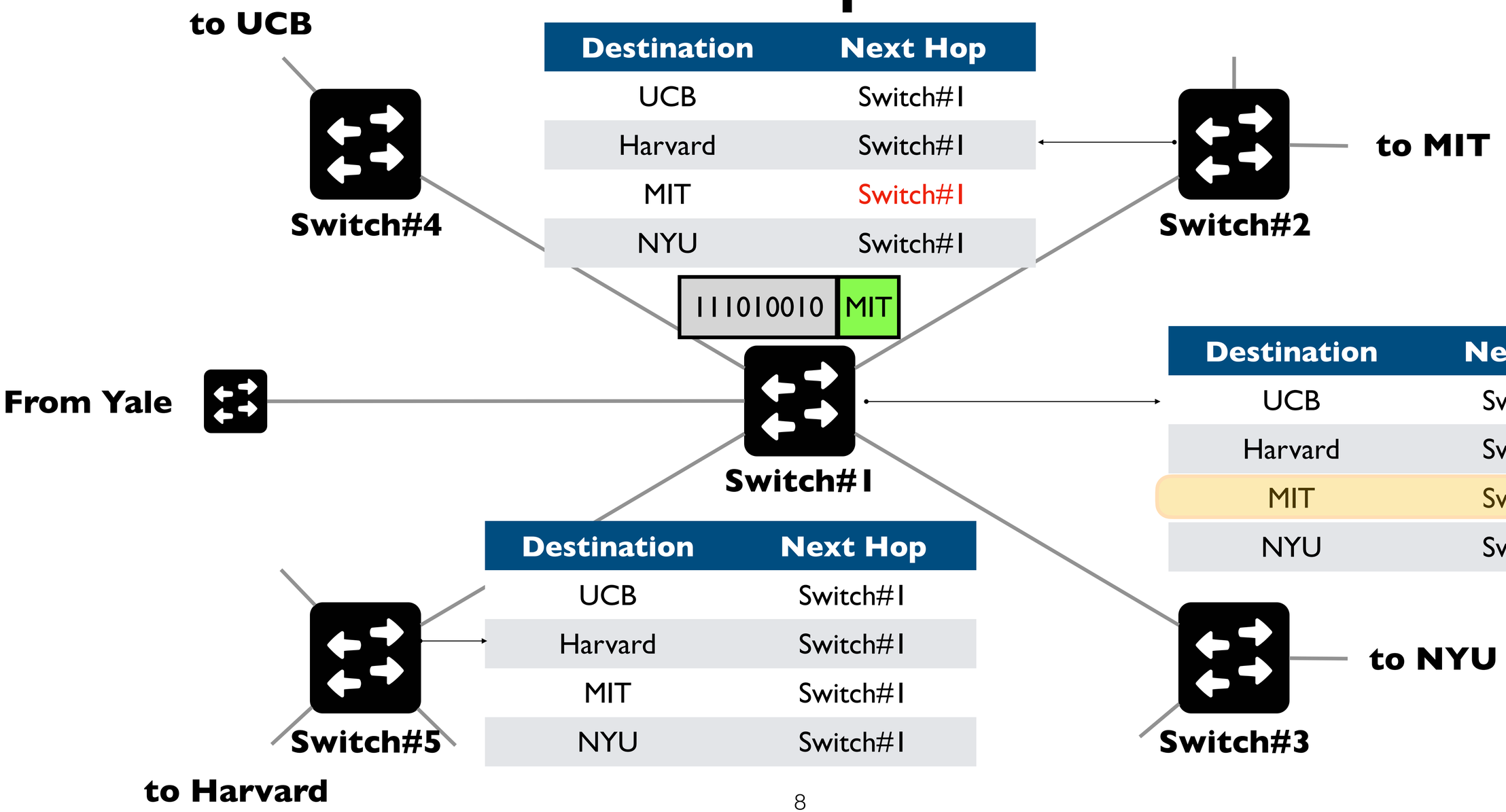


Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

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

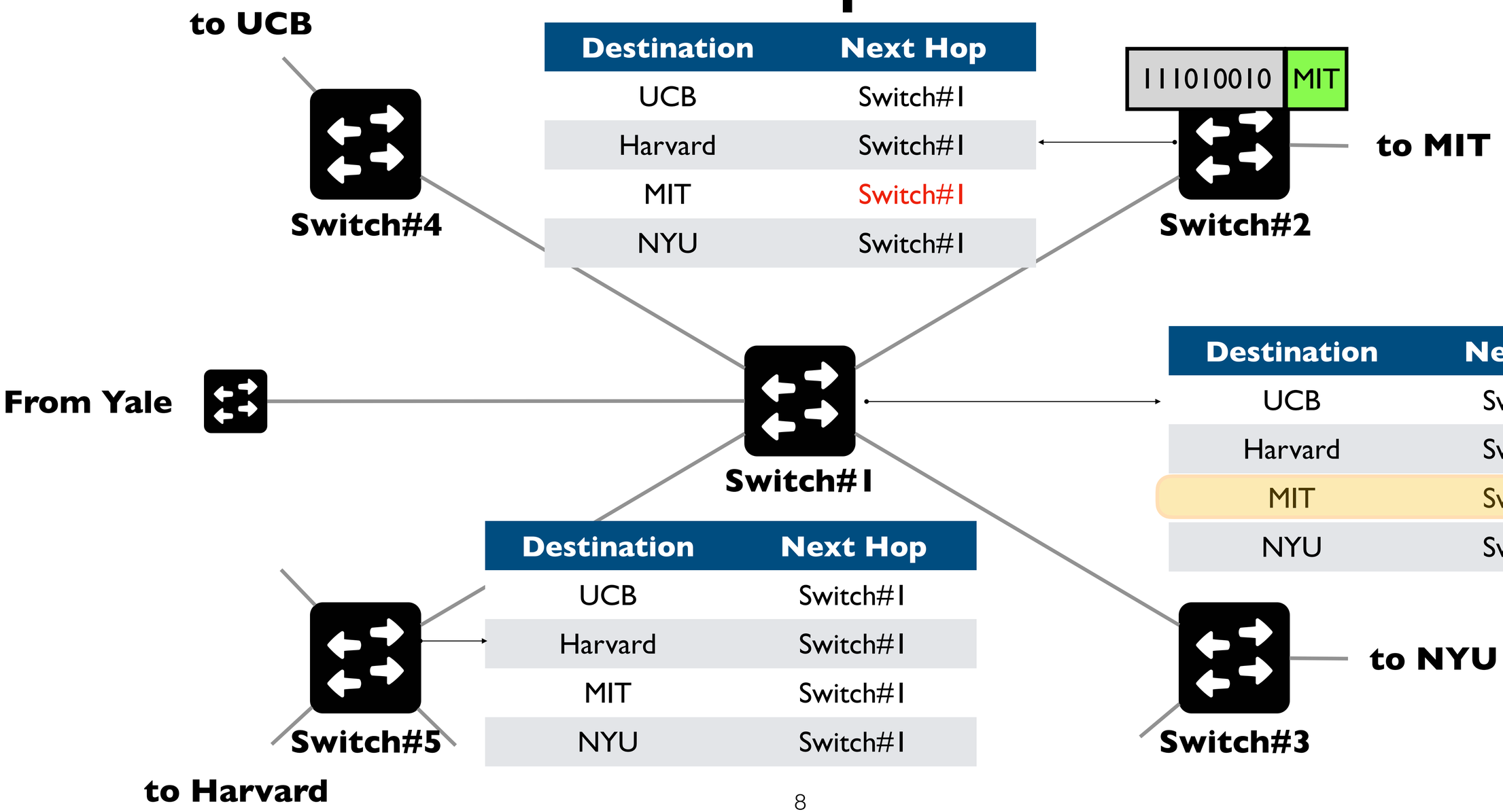


Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

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

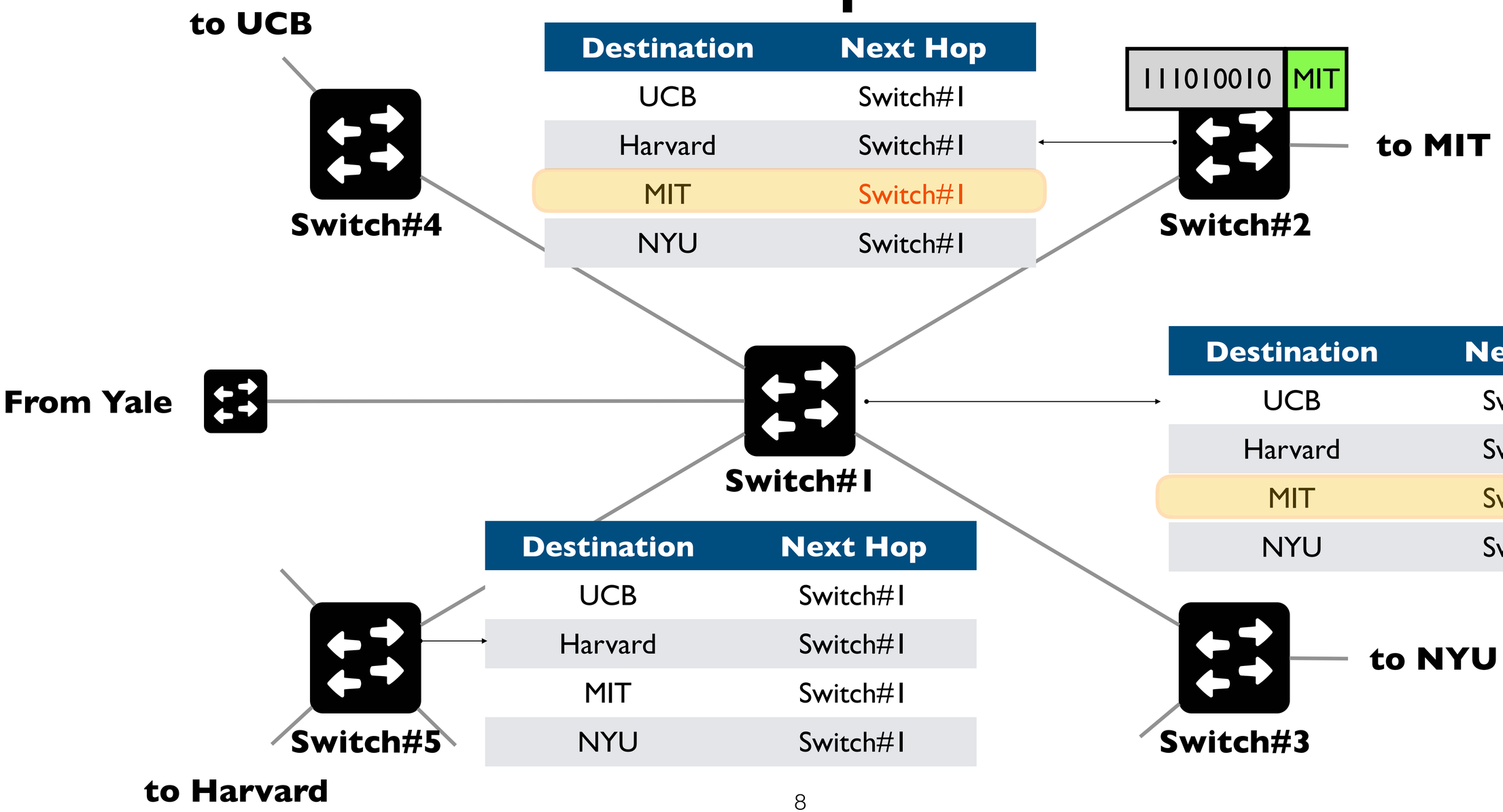


Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

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

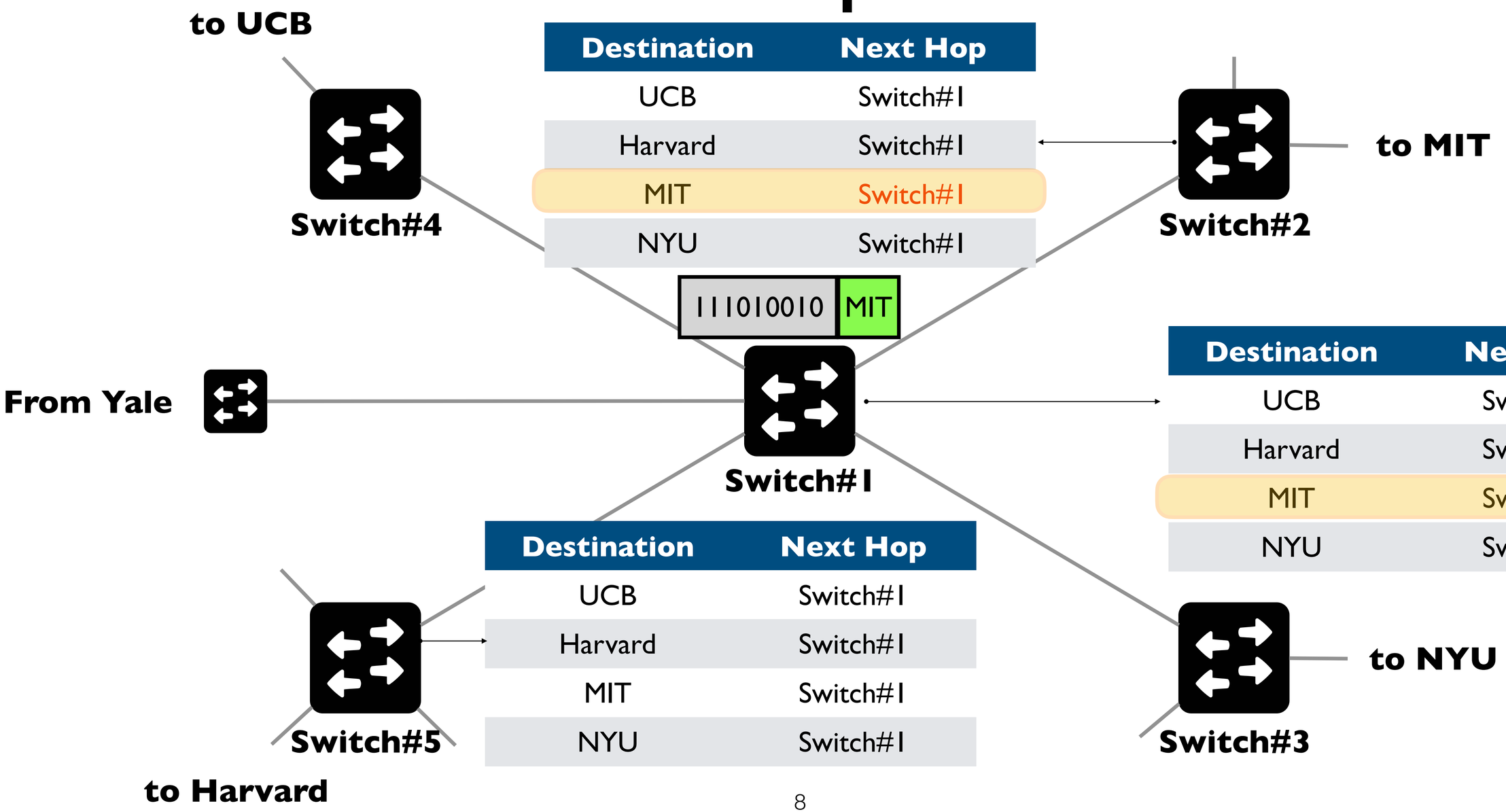


Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

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

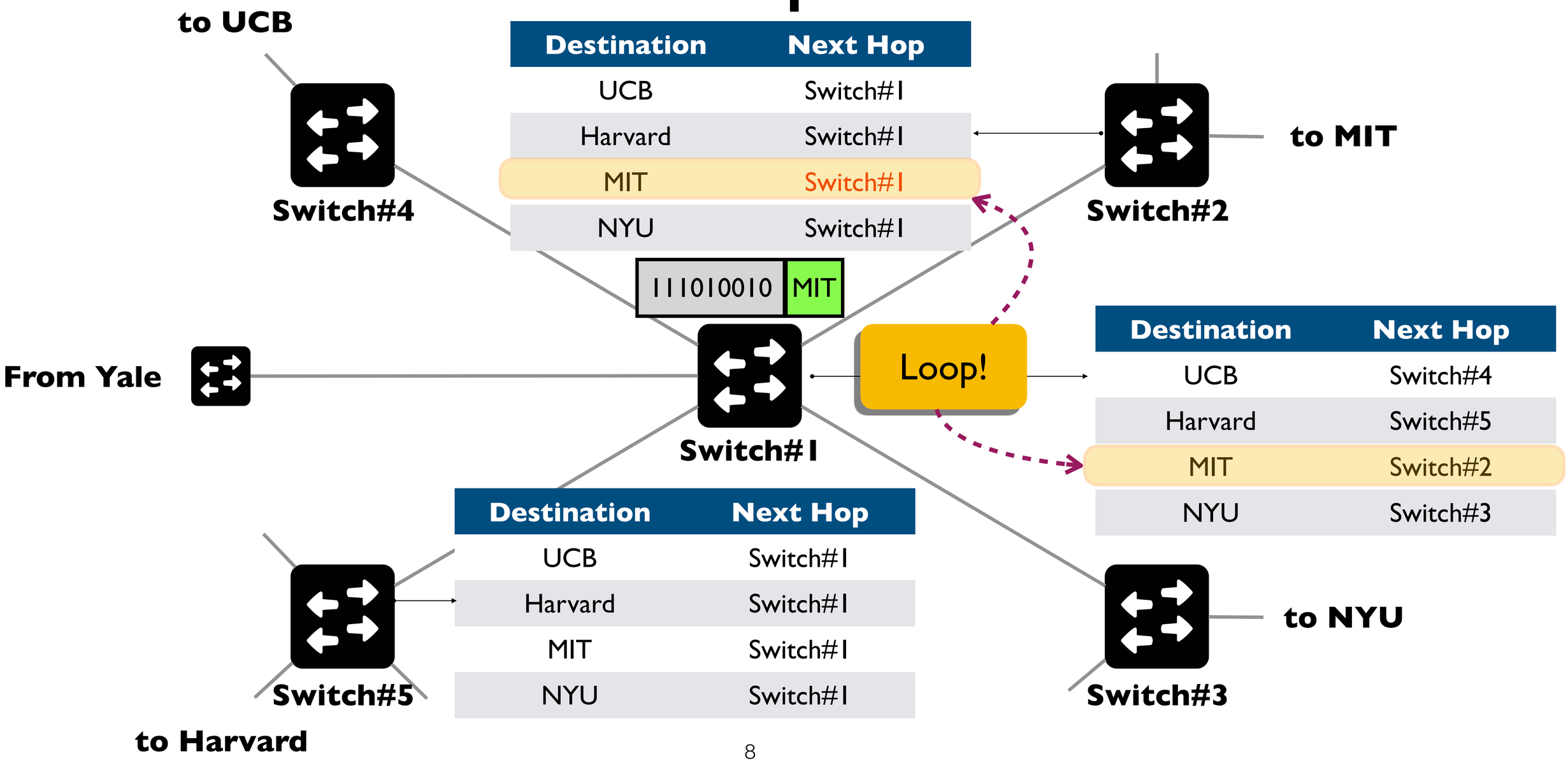


Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

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

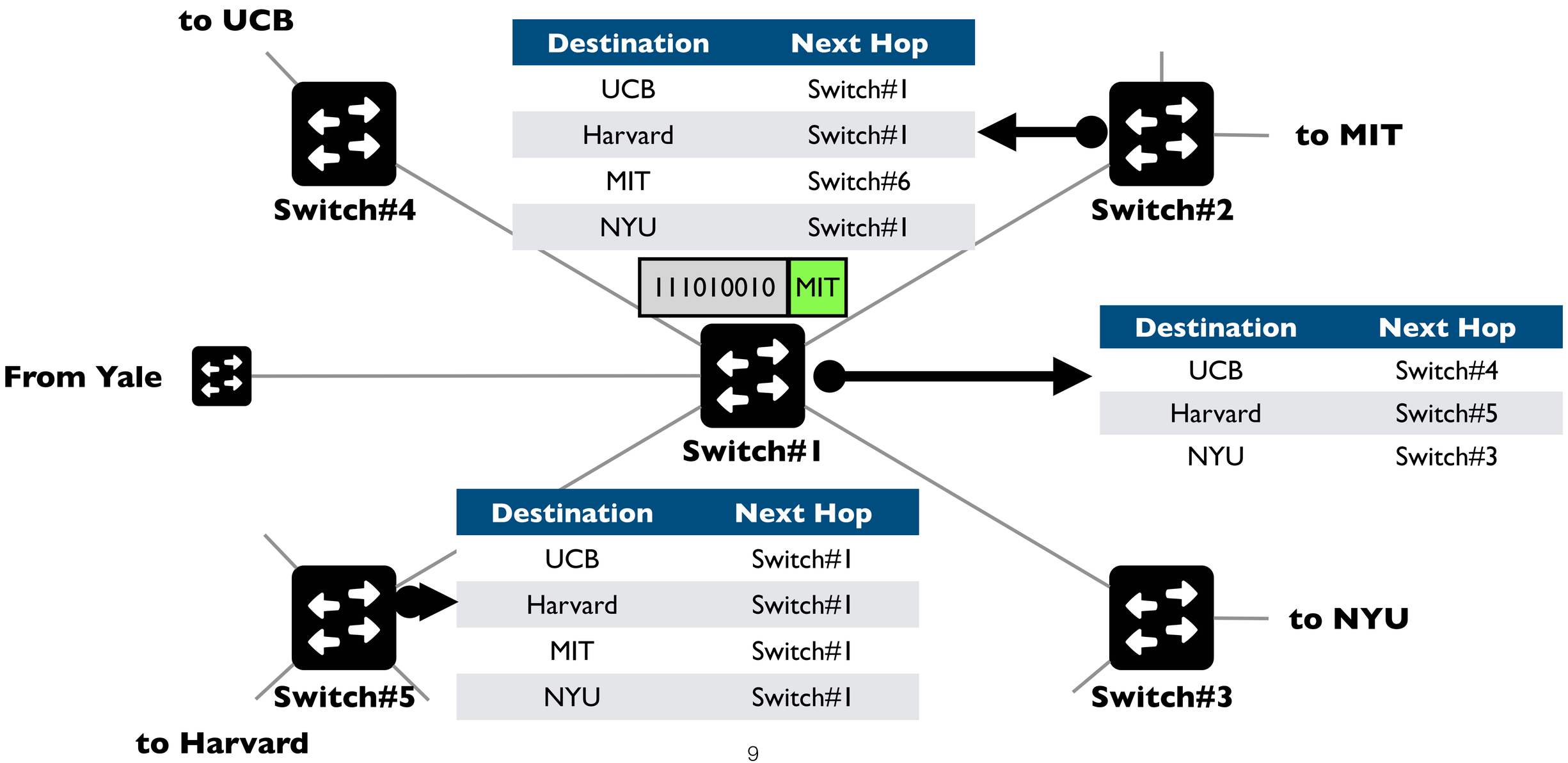


Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#1
NYU	Switch#1

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

Deadends

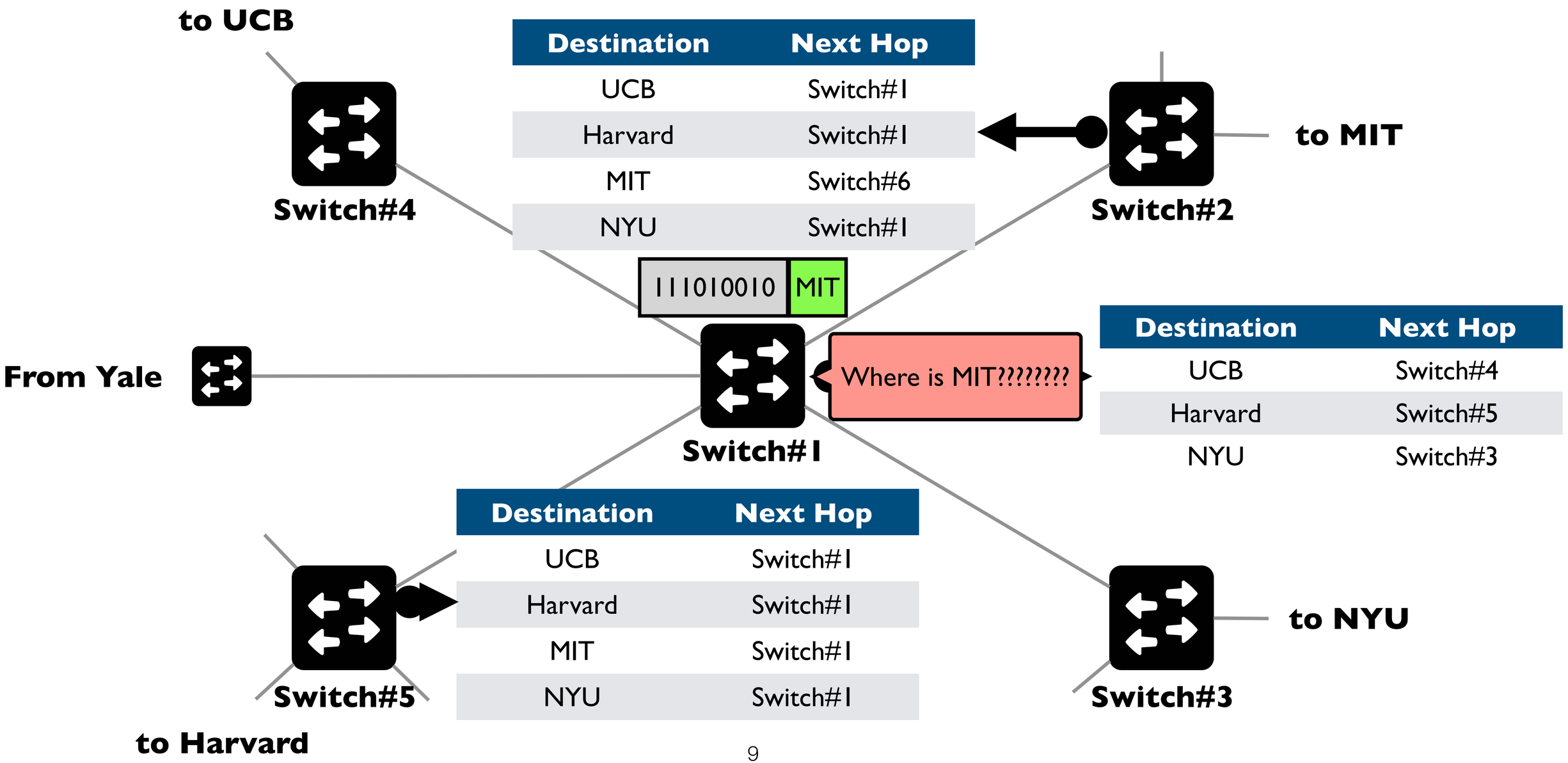


Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#6
NYU	Switch#1

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

Deadends

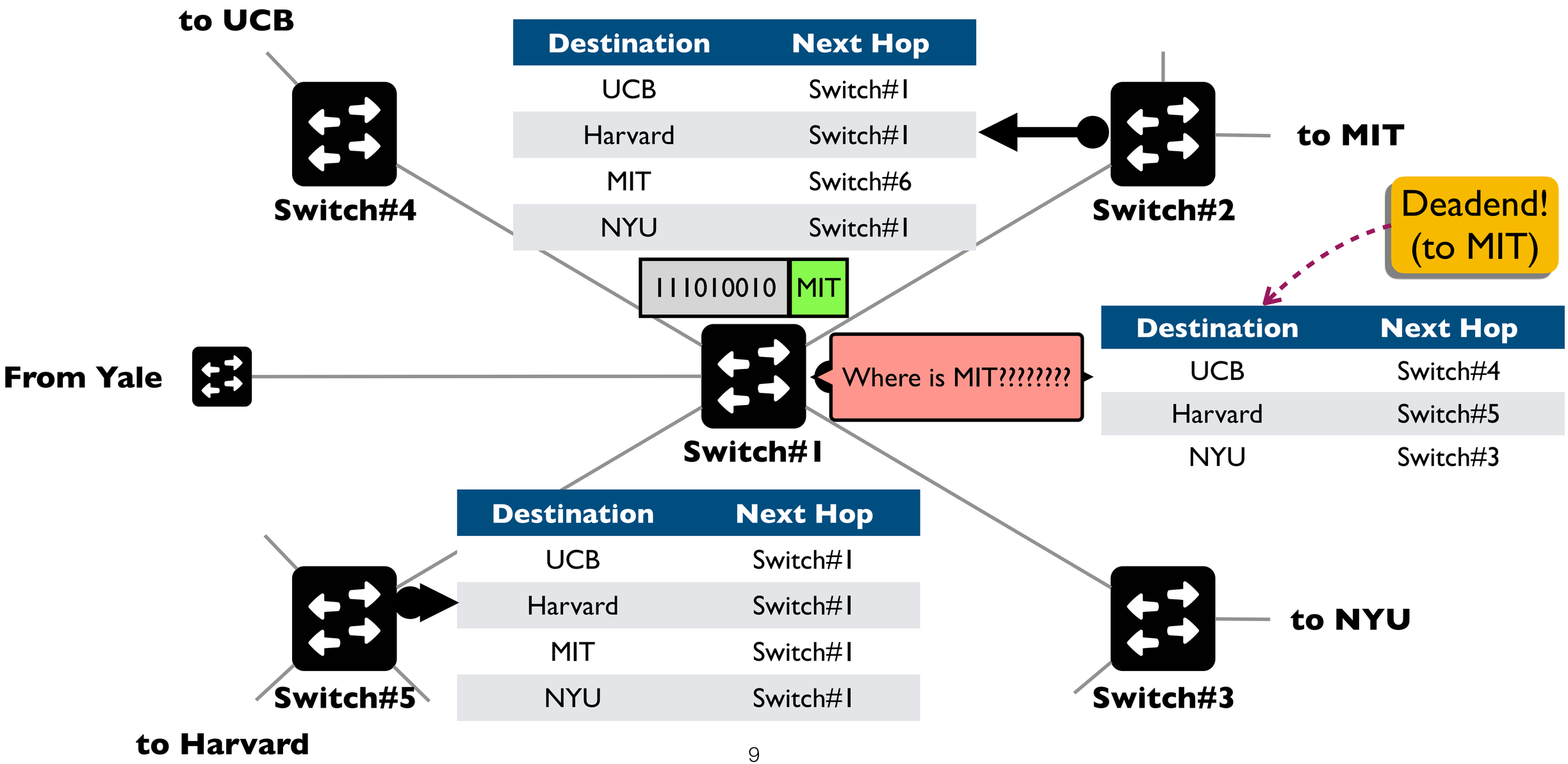


Destination	Next Hop
UCB	Switch#1
Harvard	Switch#1
MIT	Switch#6
NYU	Switch#1

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

Deadends



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

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

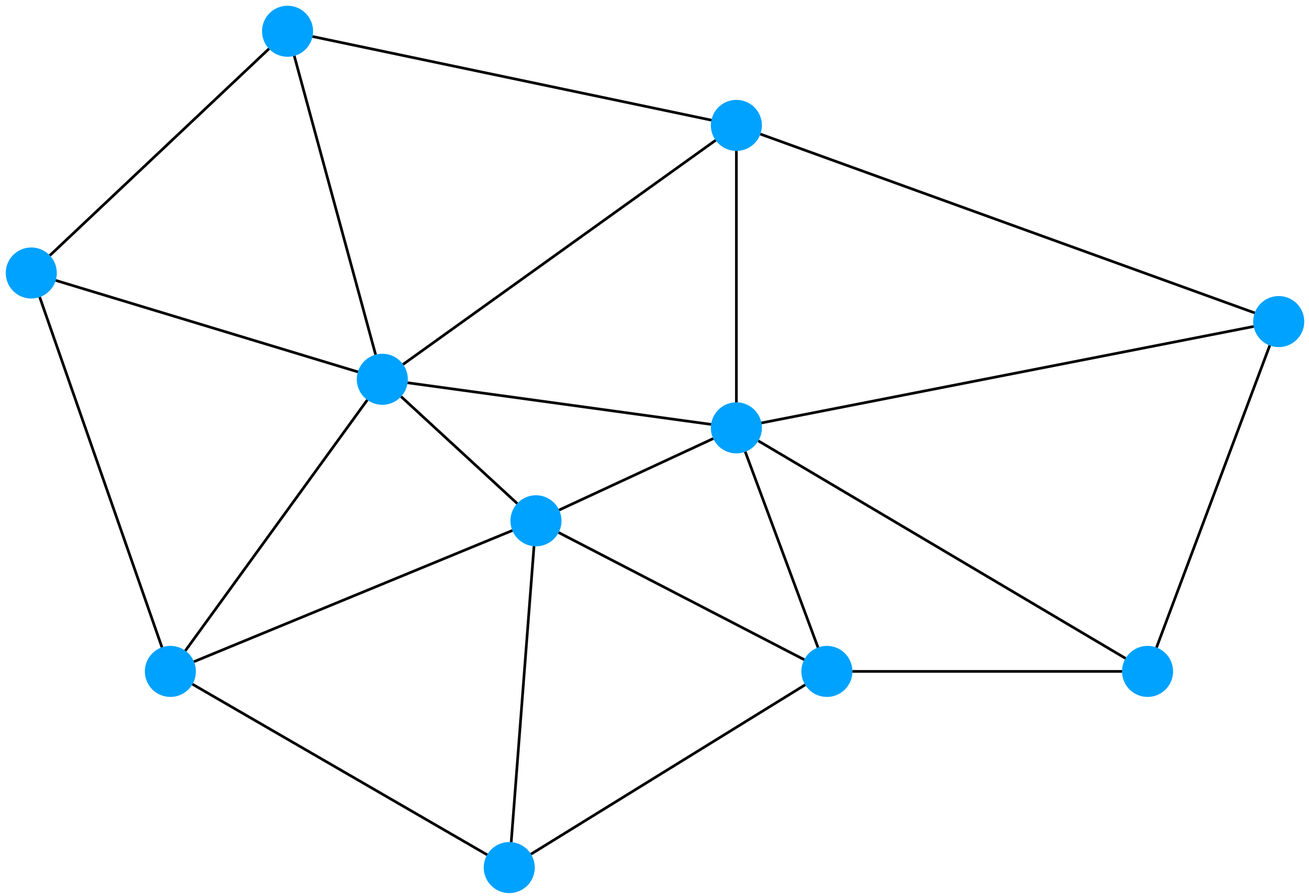
Checking Validity of Routing State

- **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

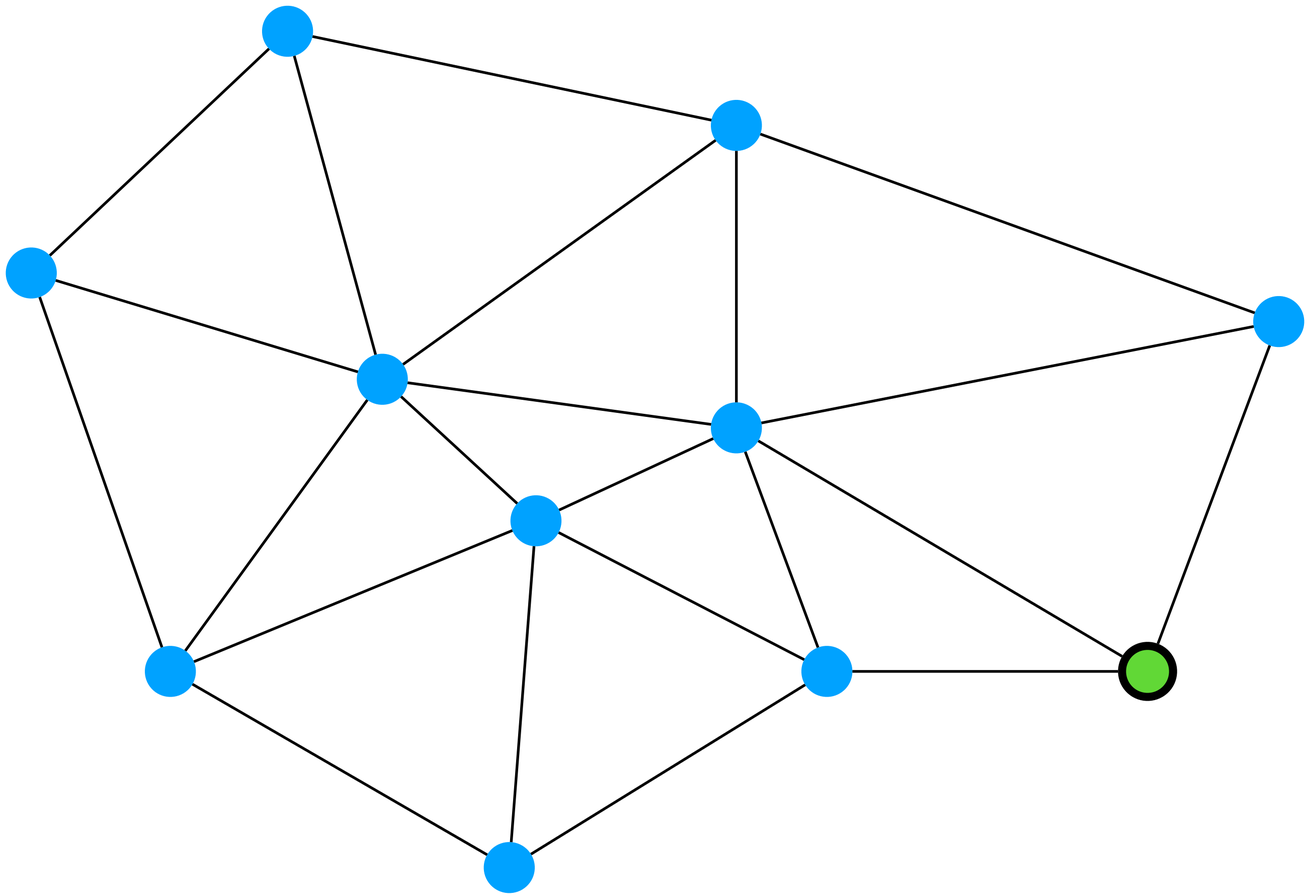
Checking Validity of Routing State

- **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…

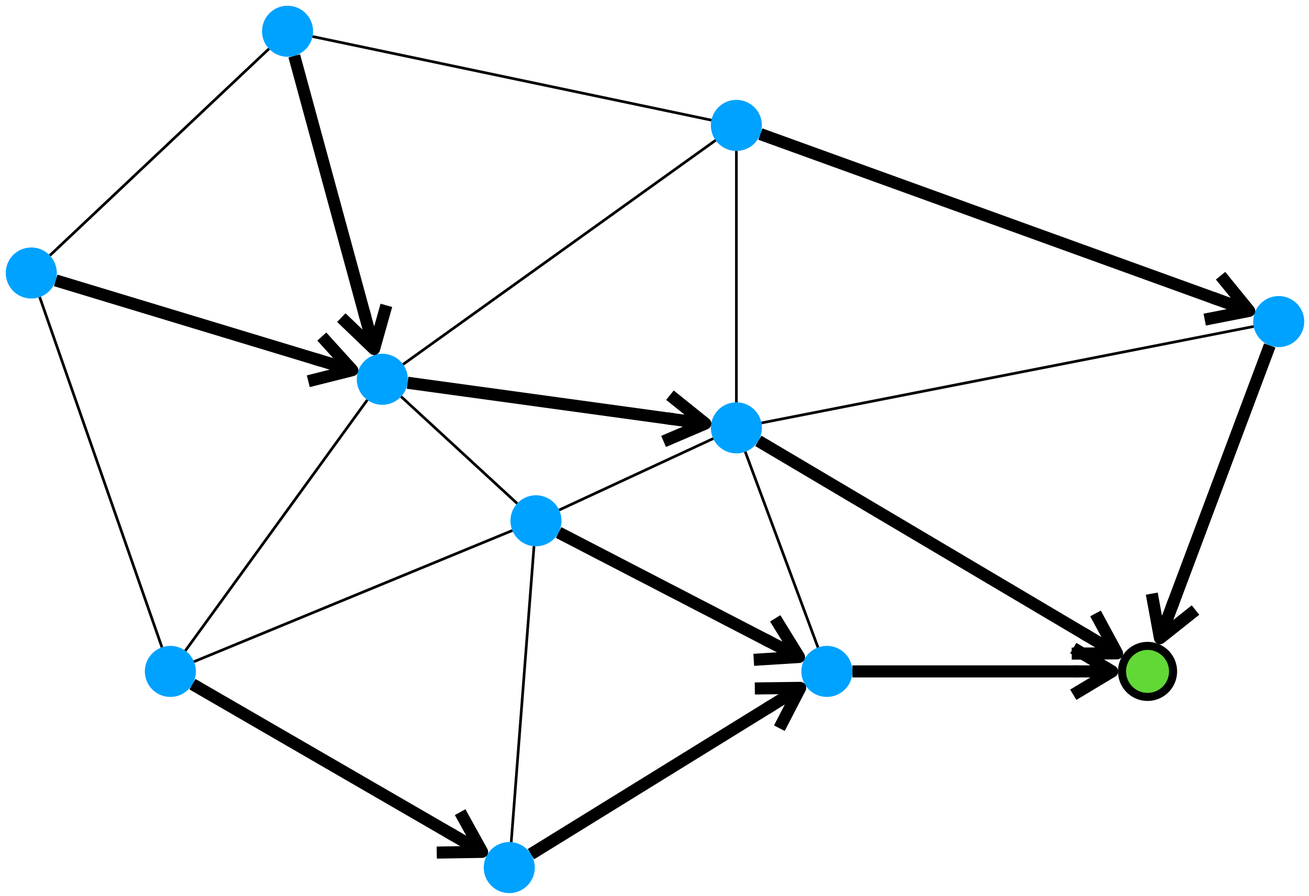
Example 1



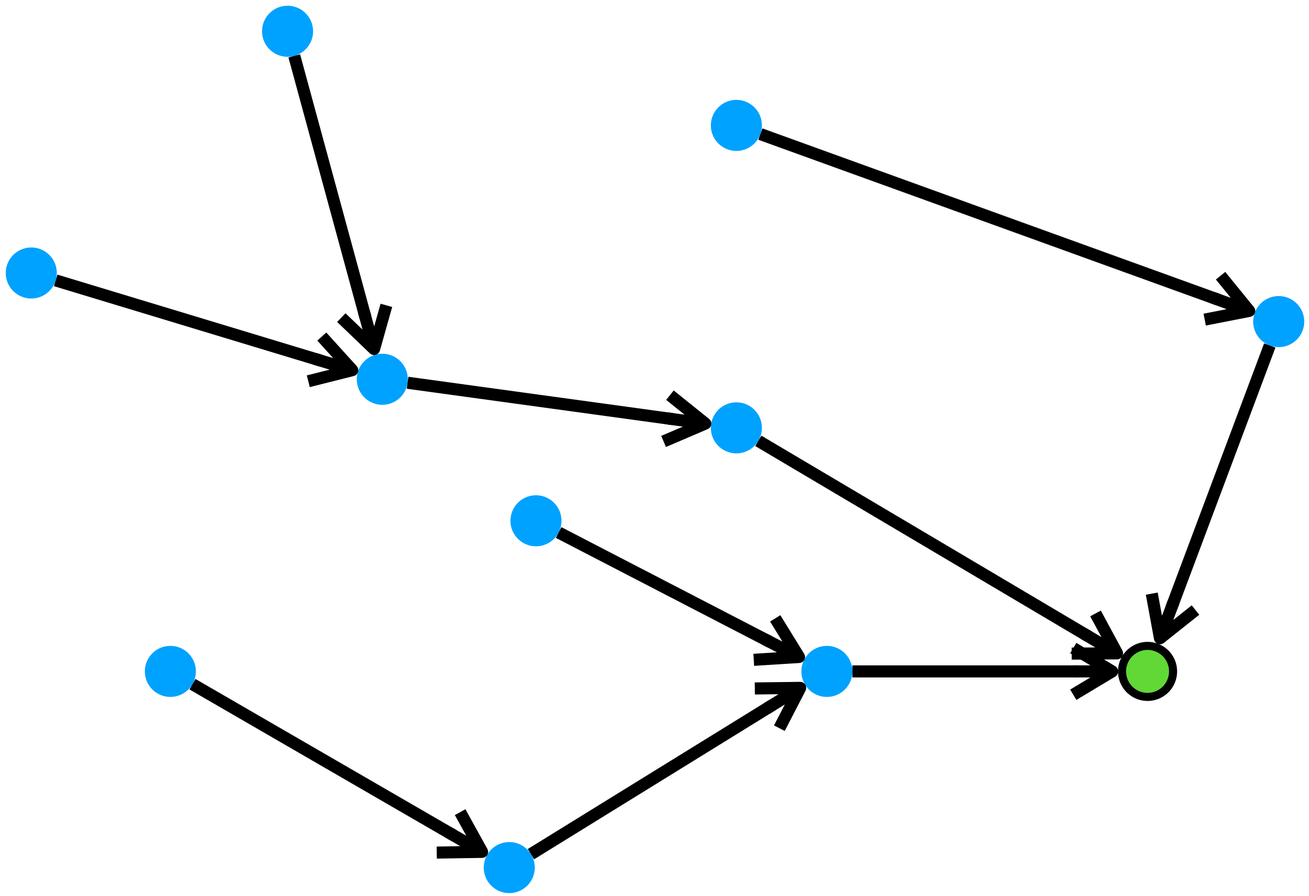
Pick Destination



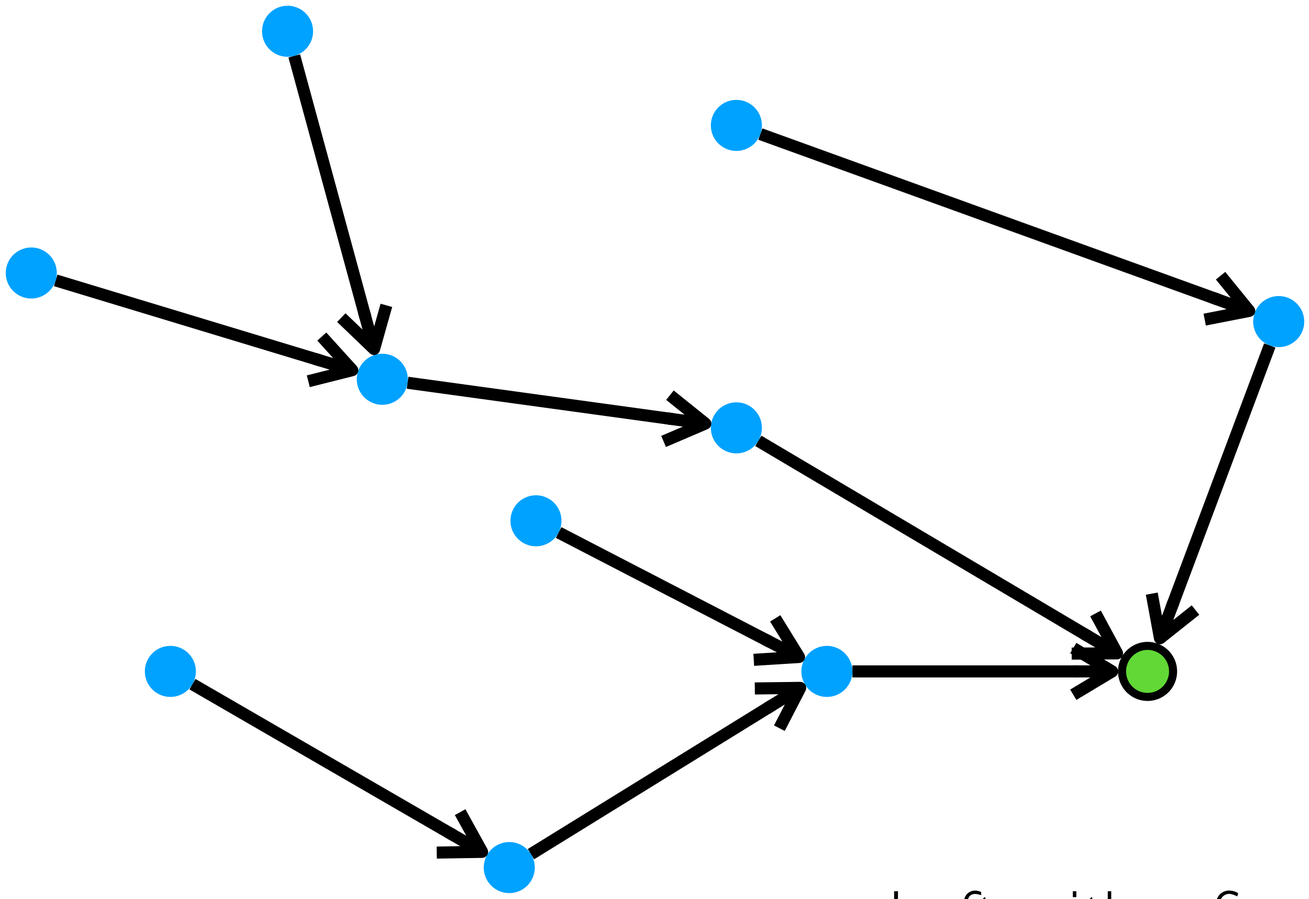
Put arrows on outgoing links (to green dot)



Remove Unused Links

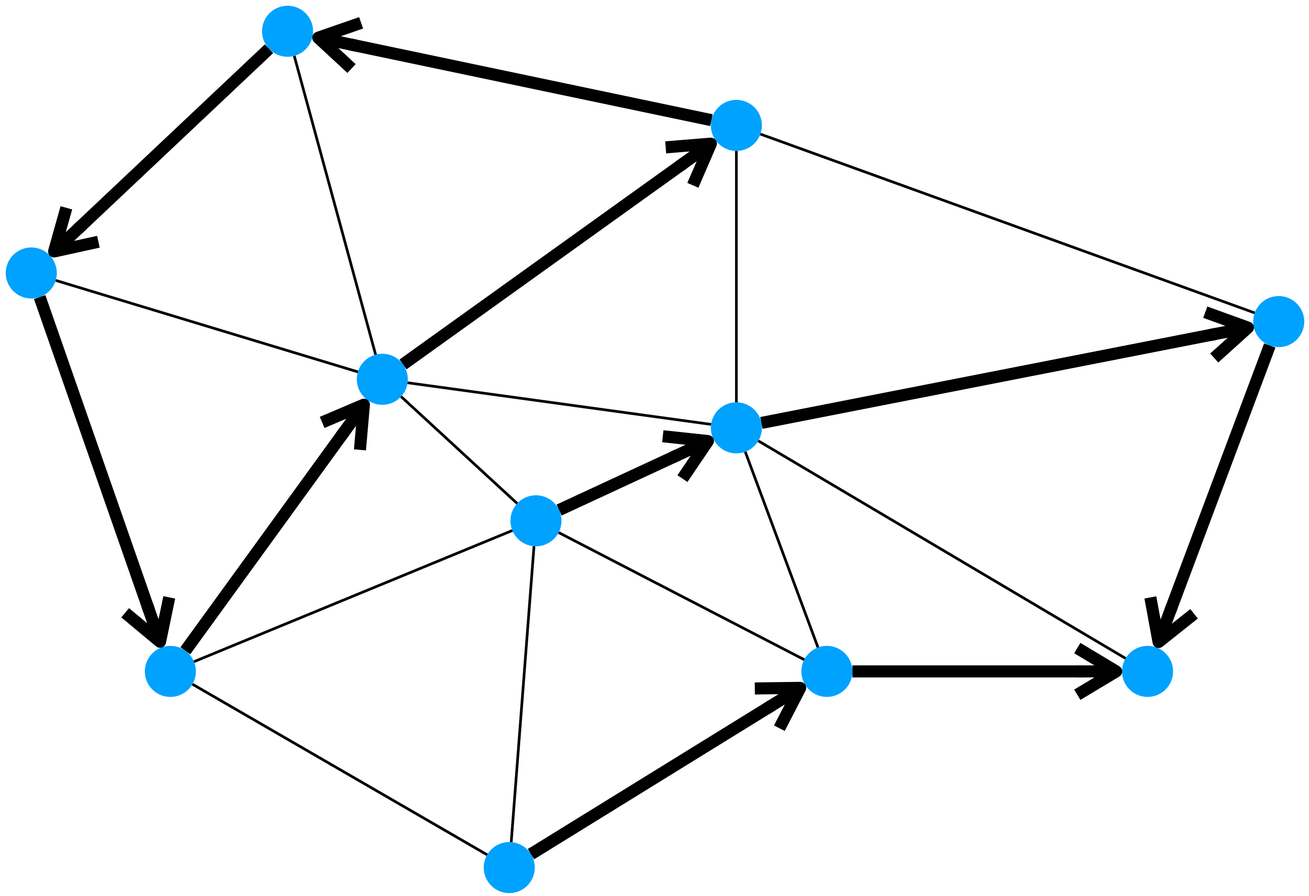


Remove Unused Links

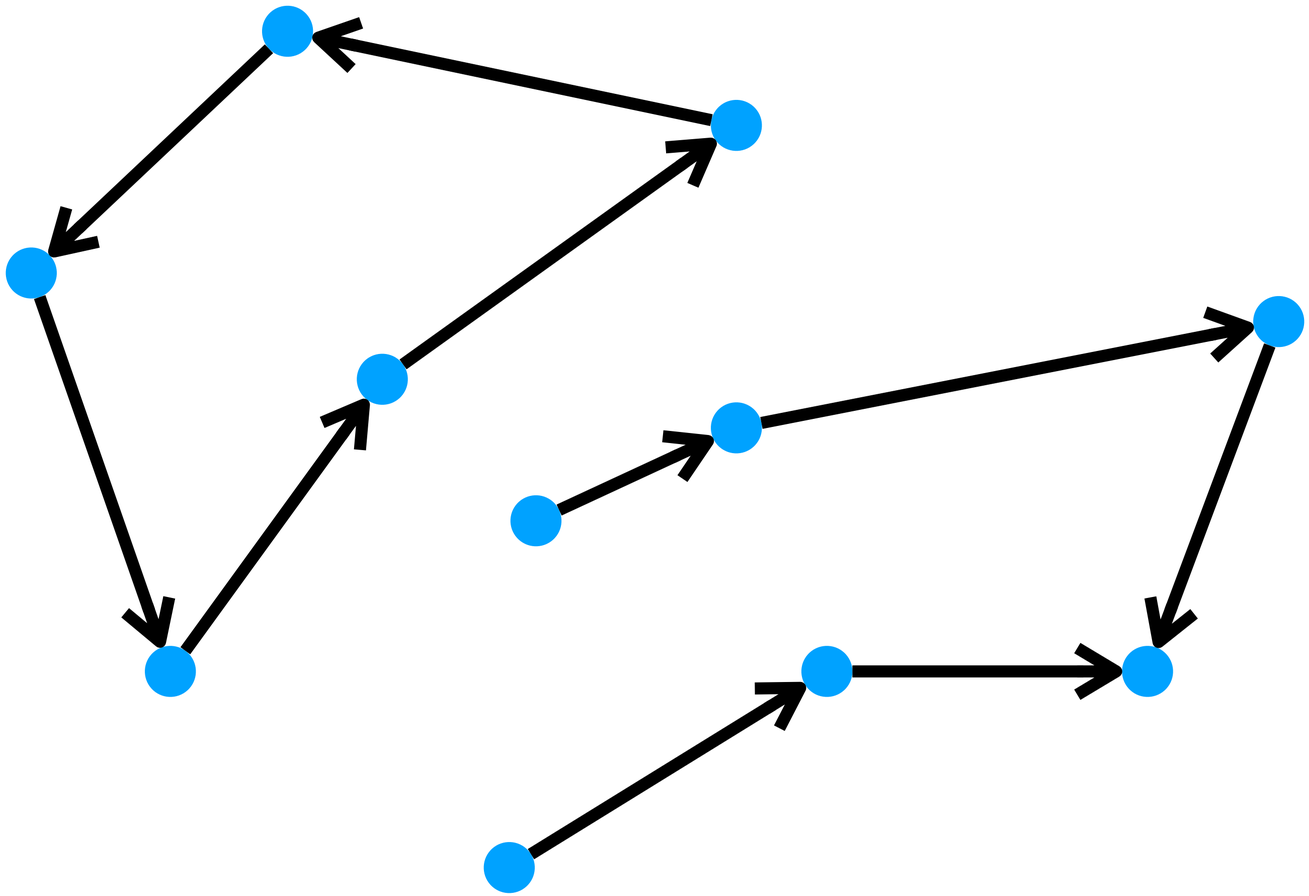


Left with a *Spanning Tree*: **Valid**

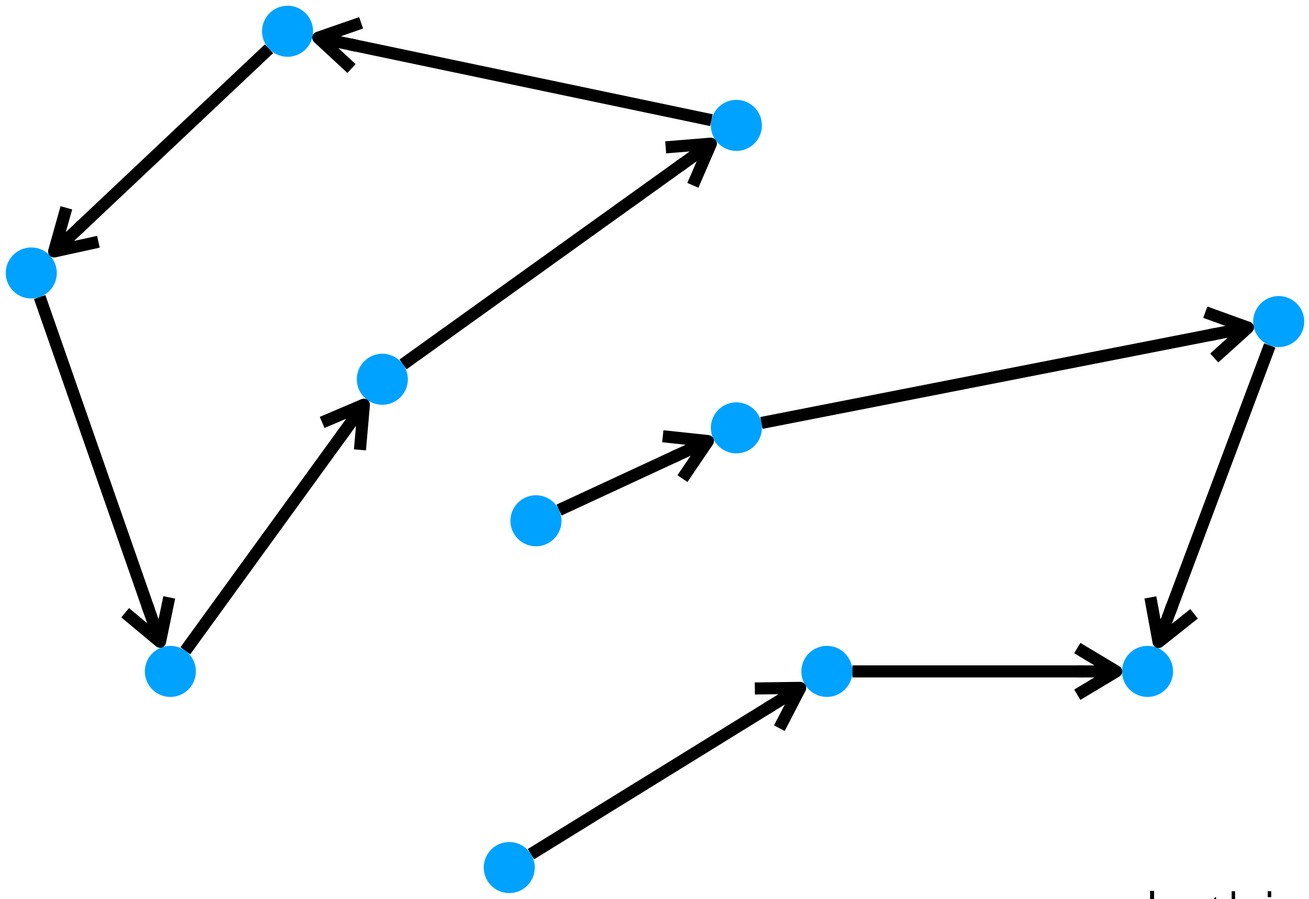
Example 2



Example 2



Example 2



Is this **Valid?**

Lesson?

Lesson?

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

Lesson?

- Very easy to check validity of routing state for a particular destination
- Dead-ends are obvious
  - Node without outgoing arrow

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

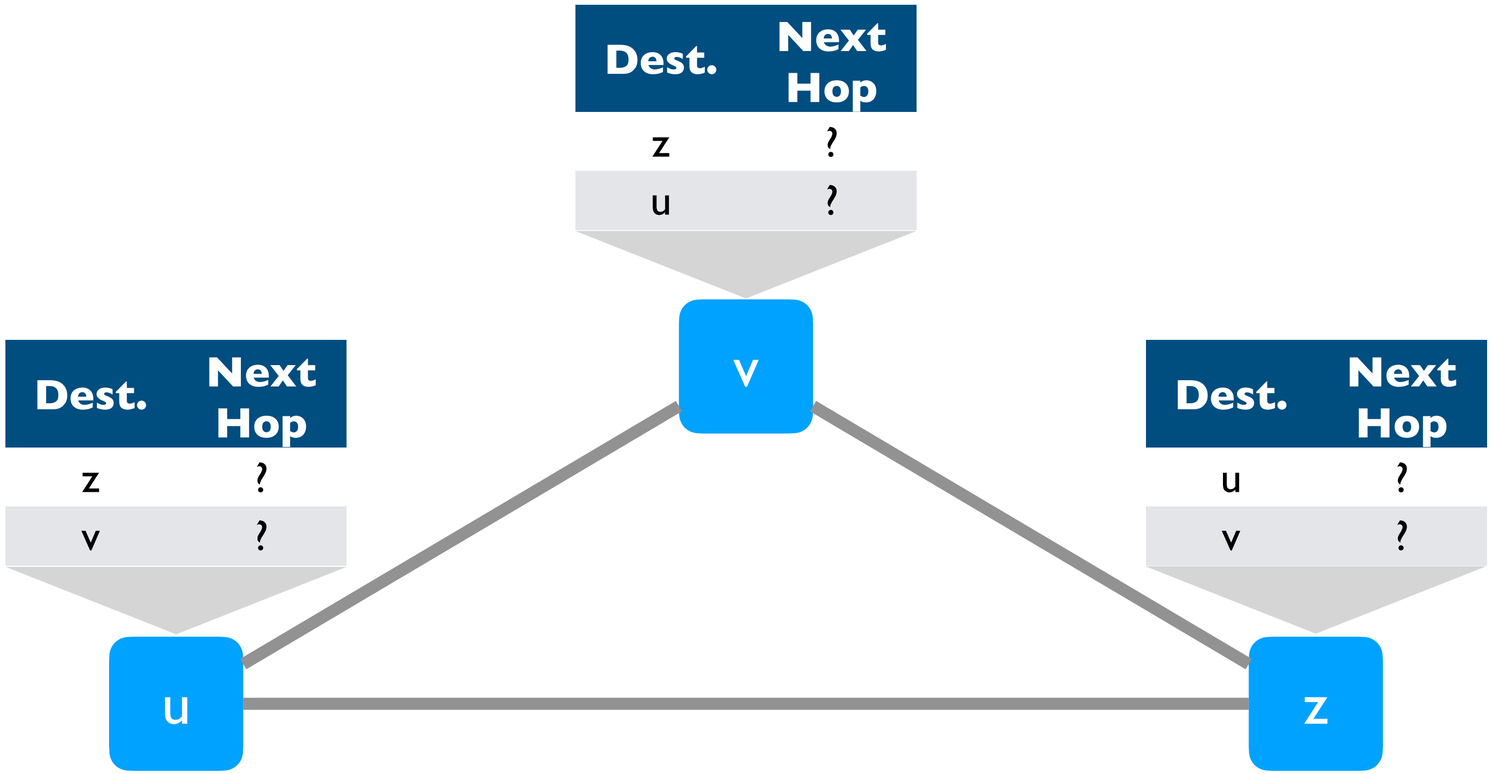
Goal (for routing)

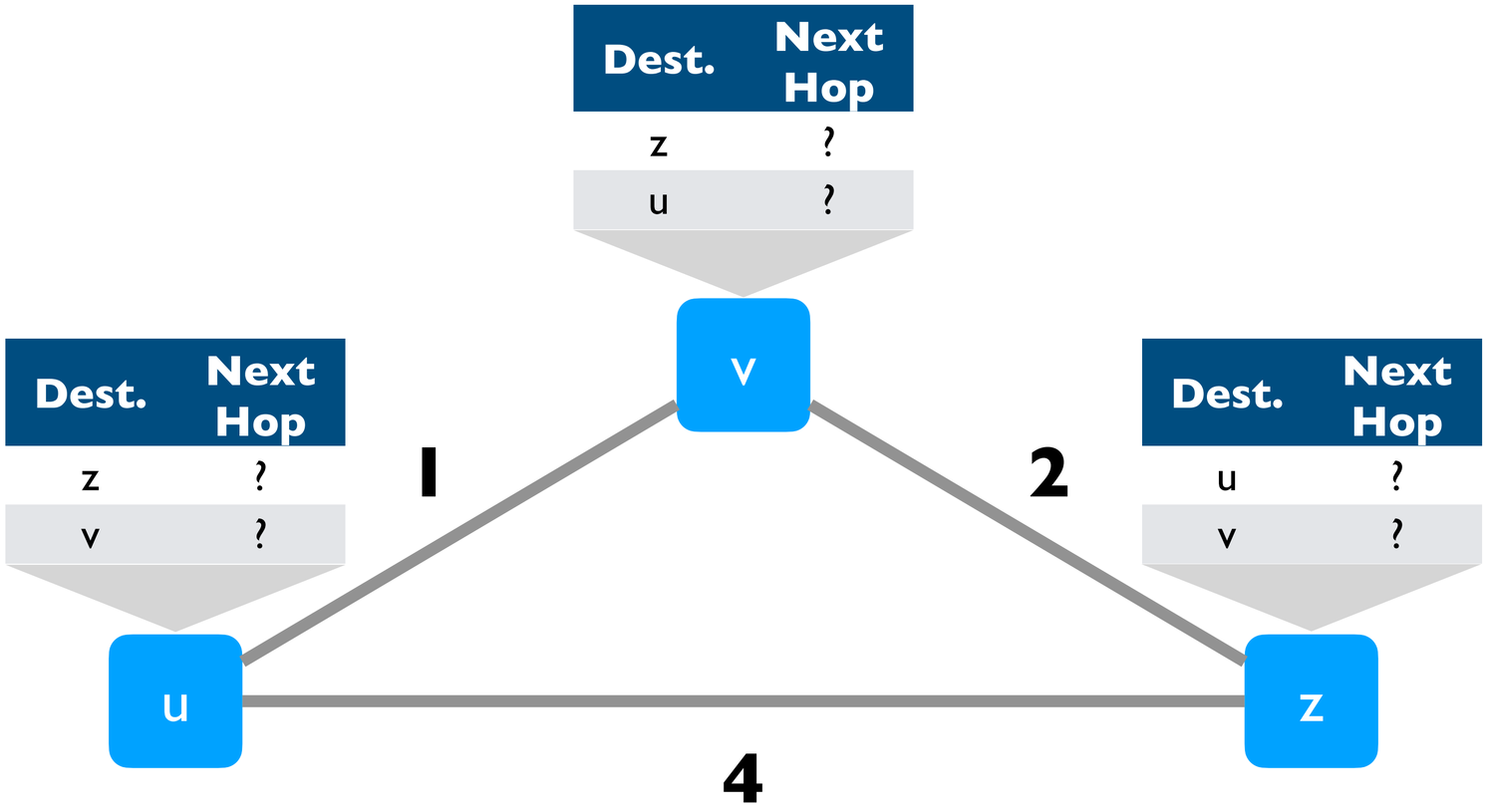
Goal (for routing)

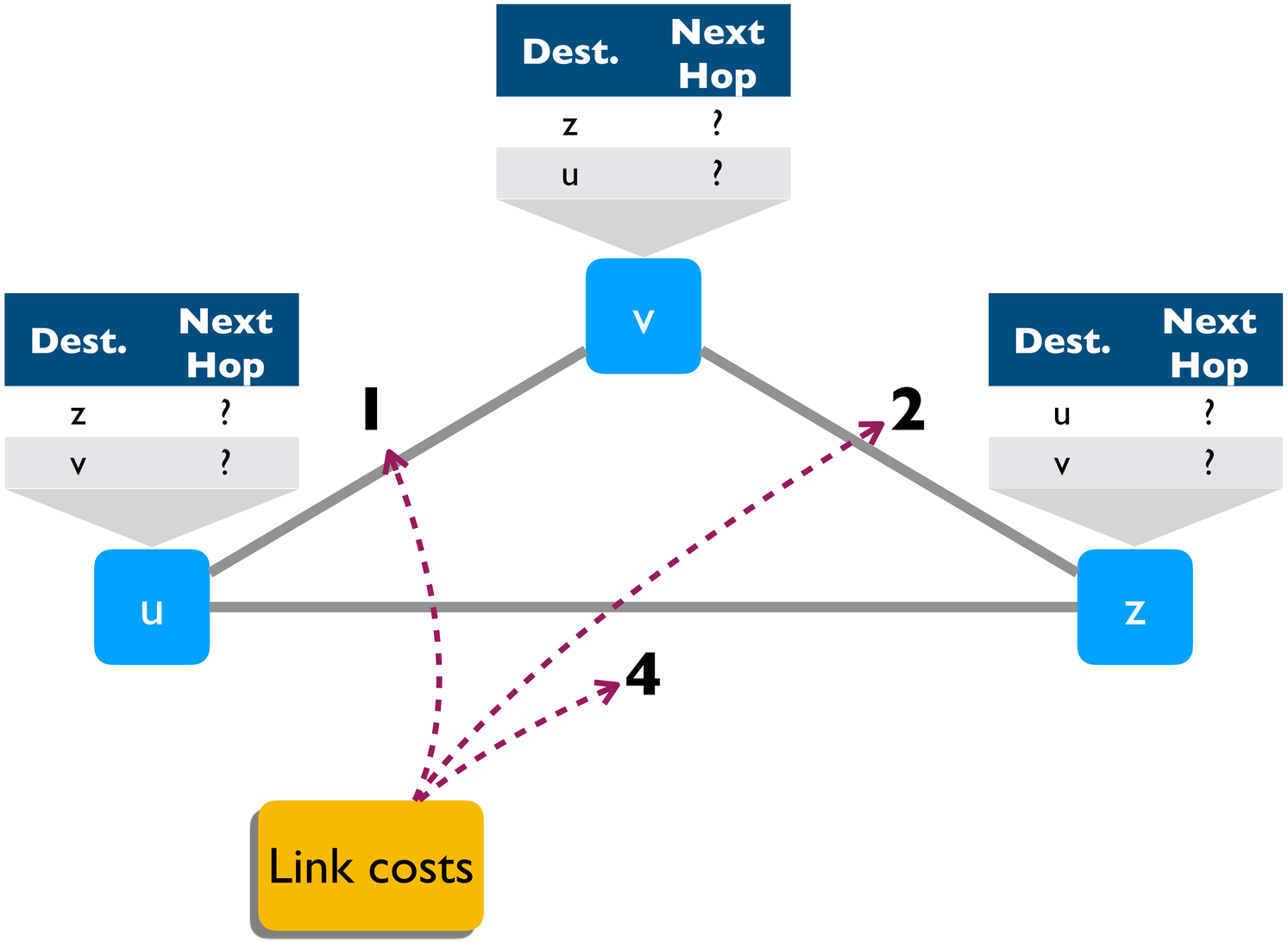
- **V1: Find a path to a given destination**

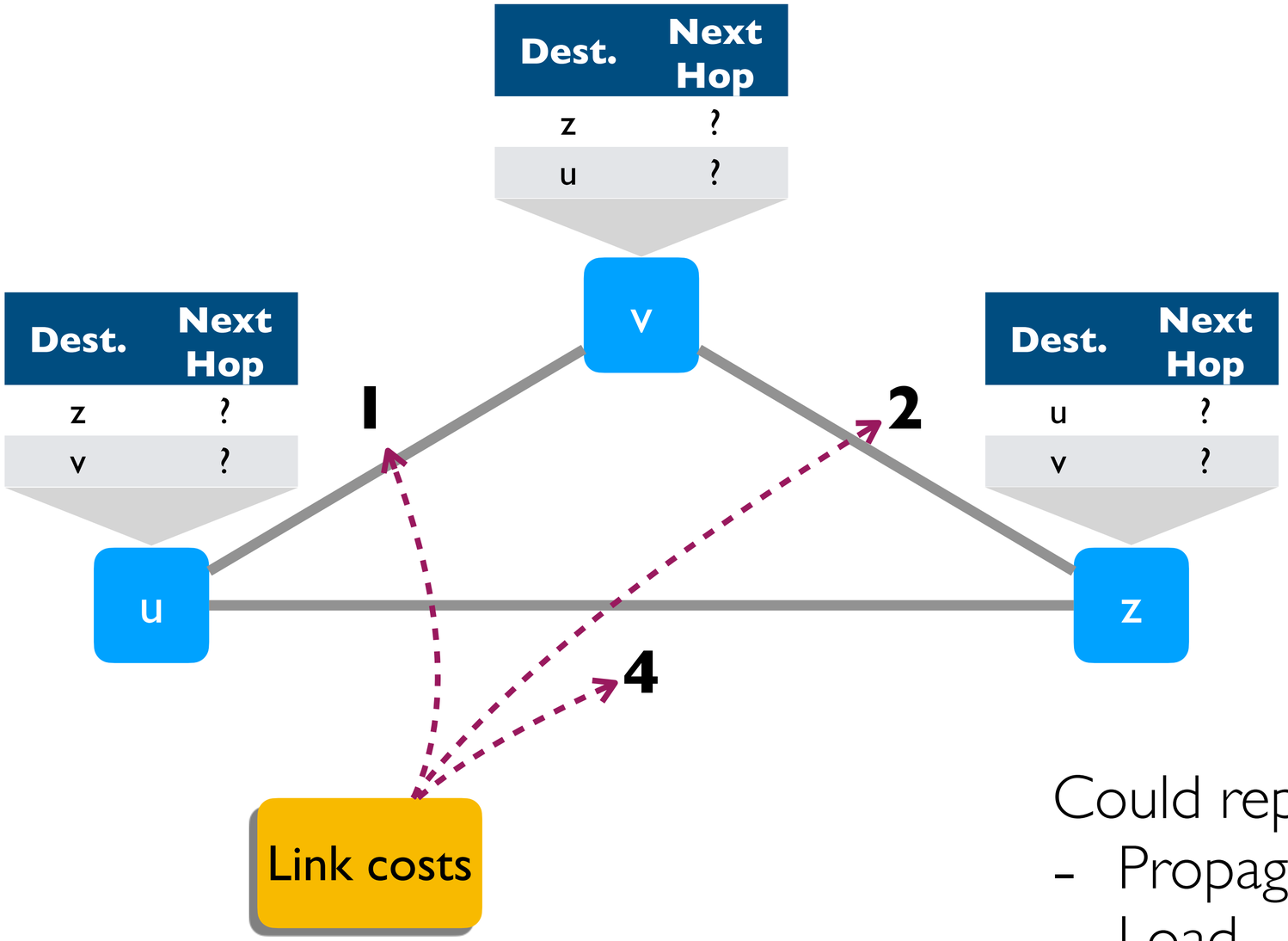
Goal (for routing)

- **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
?
z

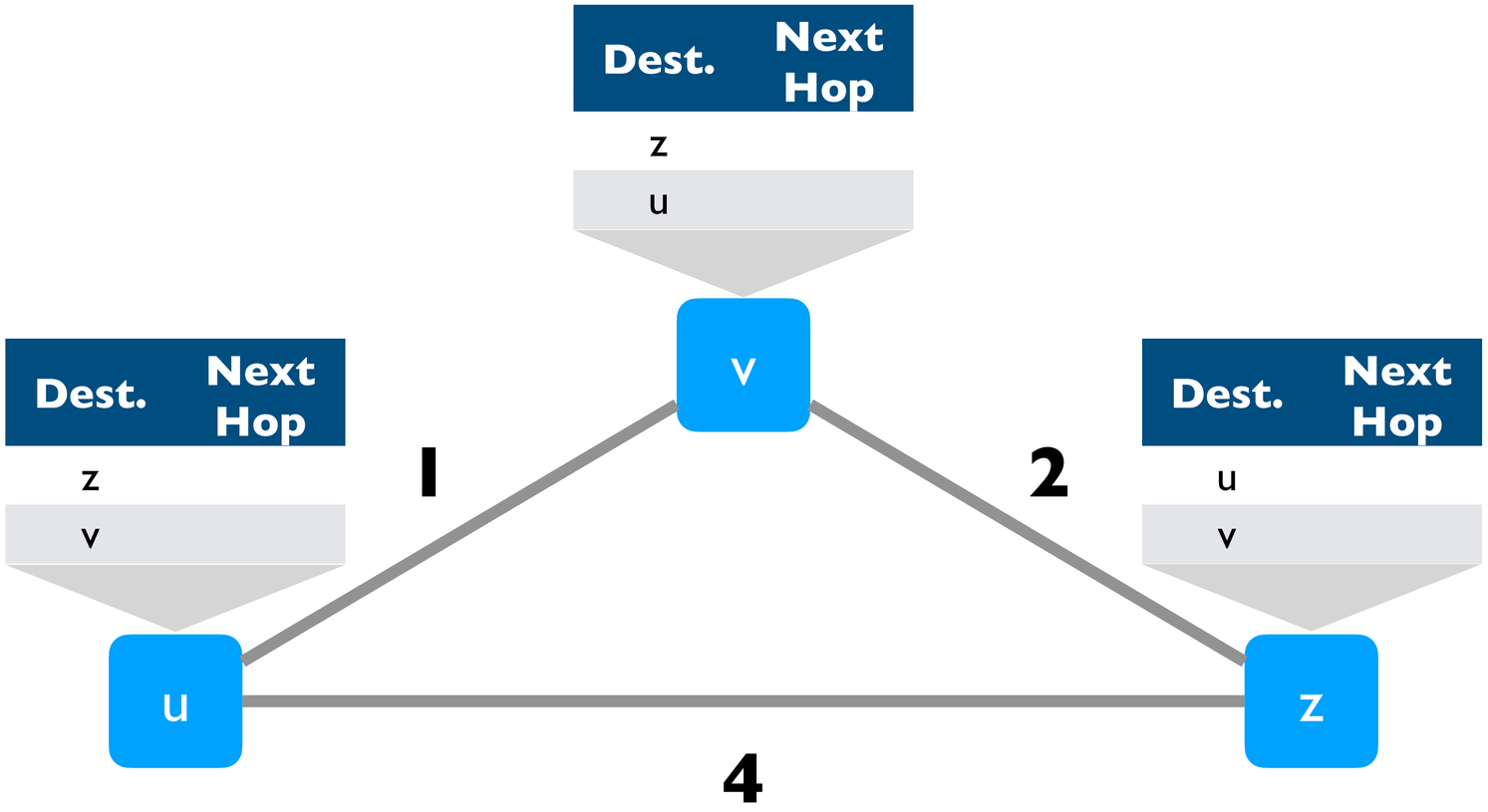

Dest.
Next Hop
z
?
u
?
v
Dest.
Next Hop
z
?
v
?
u
1
2
Dest.
Next Hop
u
?
v
?
z
4

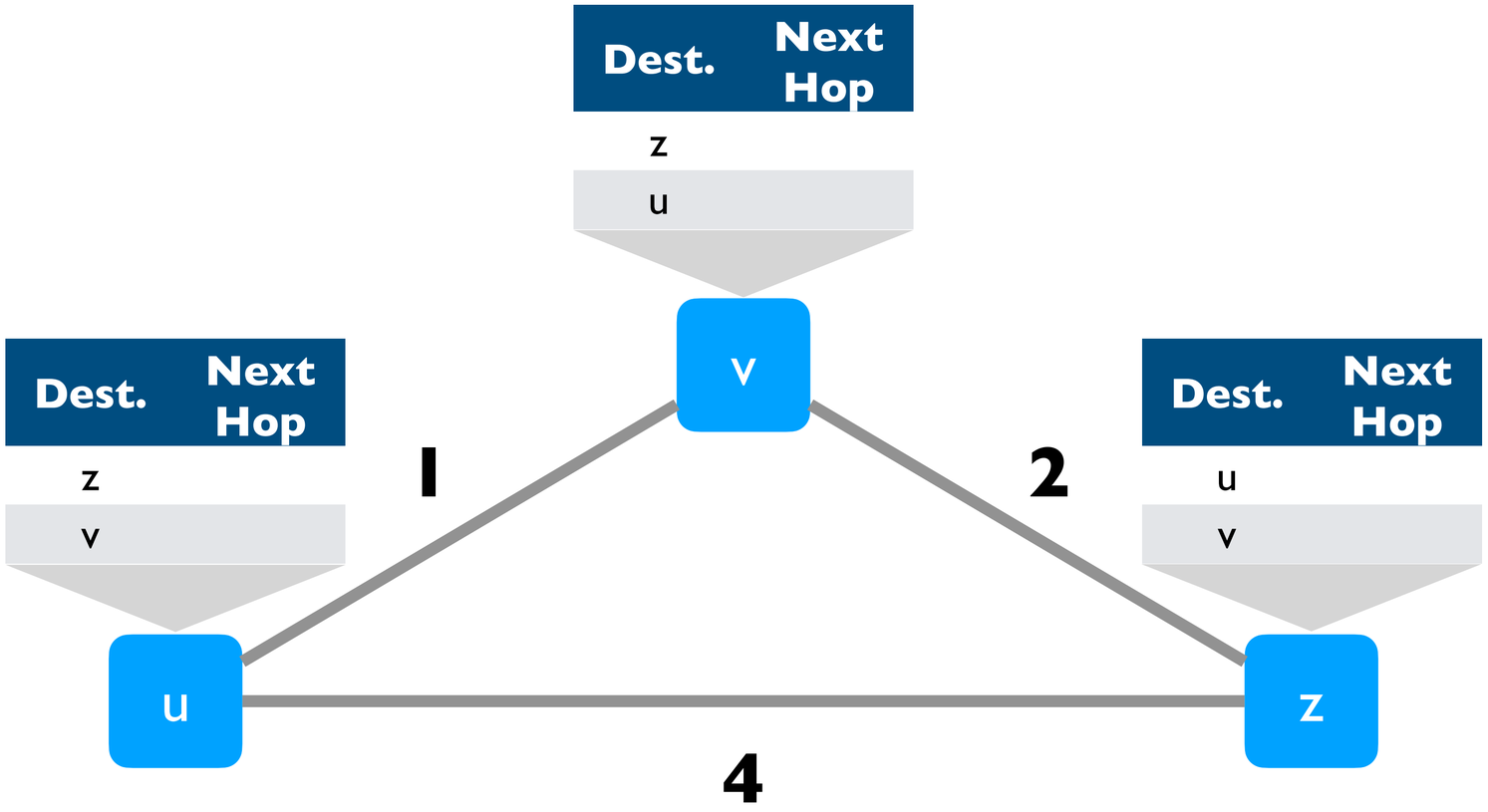

Dest.
Next Hop
z
?
u
?
v
Dest.
Next Hop
z
?
v
?
u
Dest.
Next Hop
u
?
v
?
z
1
2
4
Link costs



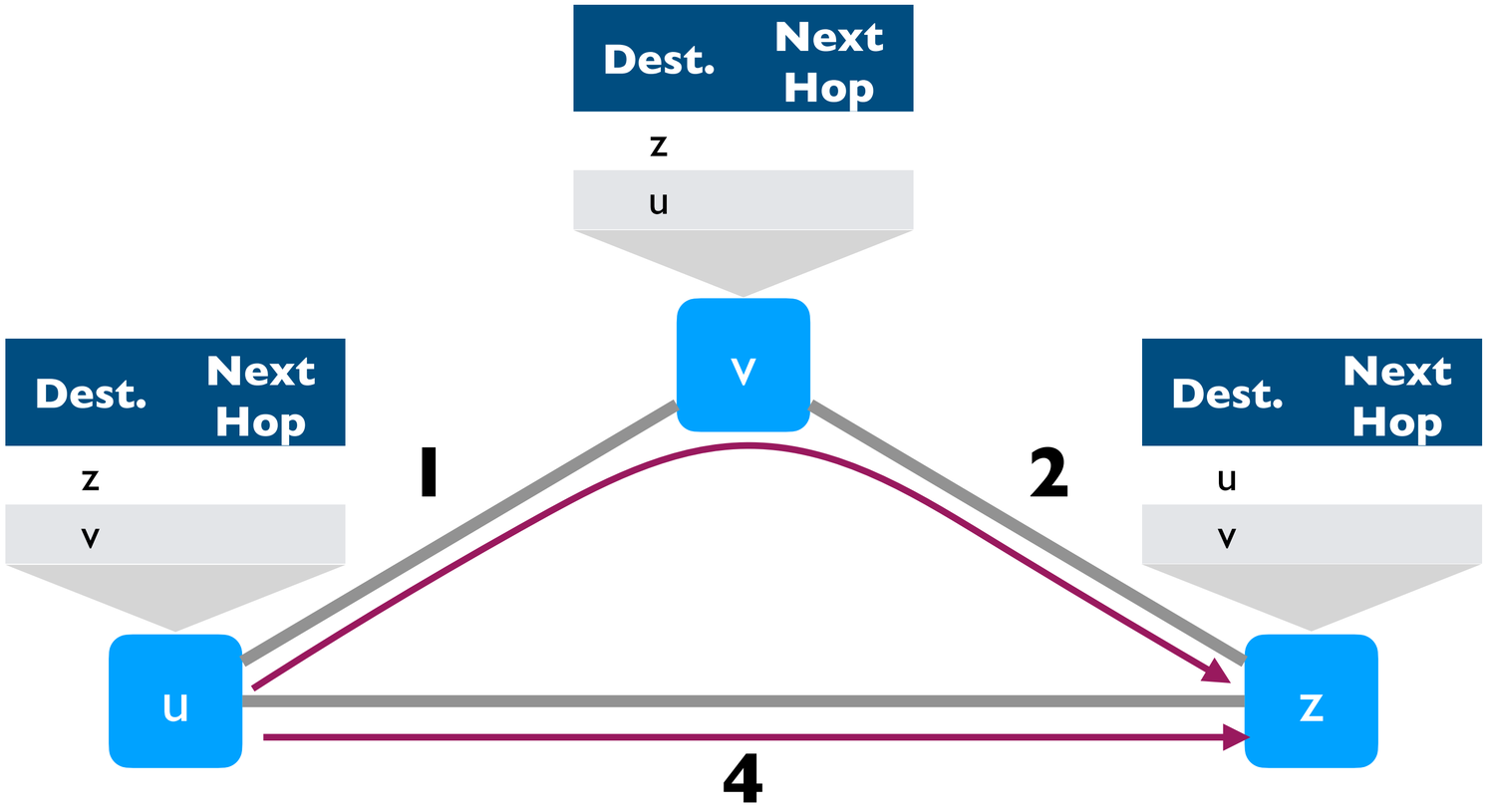
Could represent:

- Propagation delay
- Load
- Cost

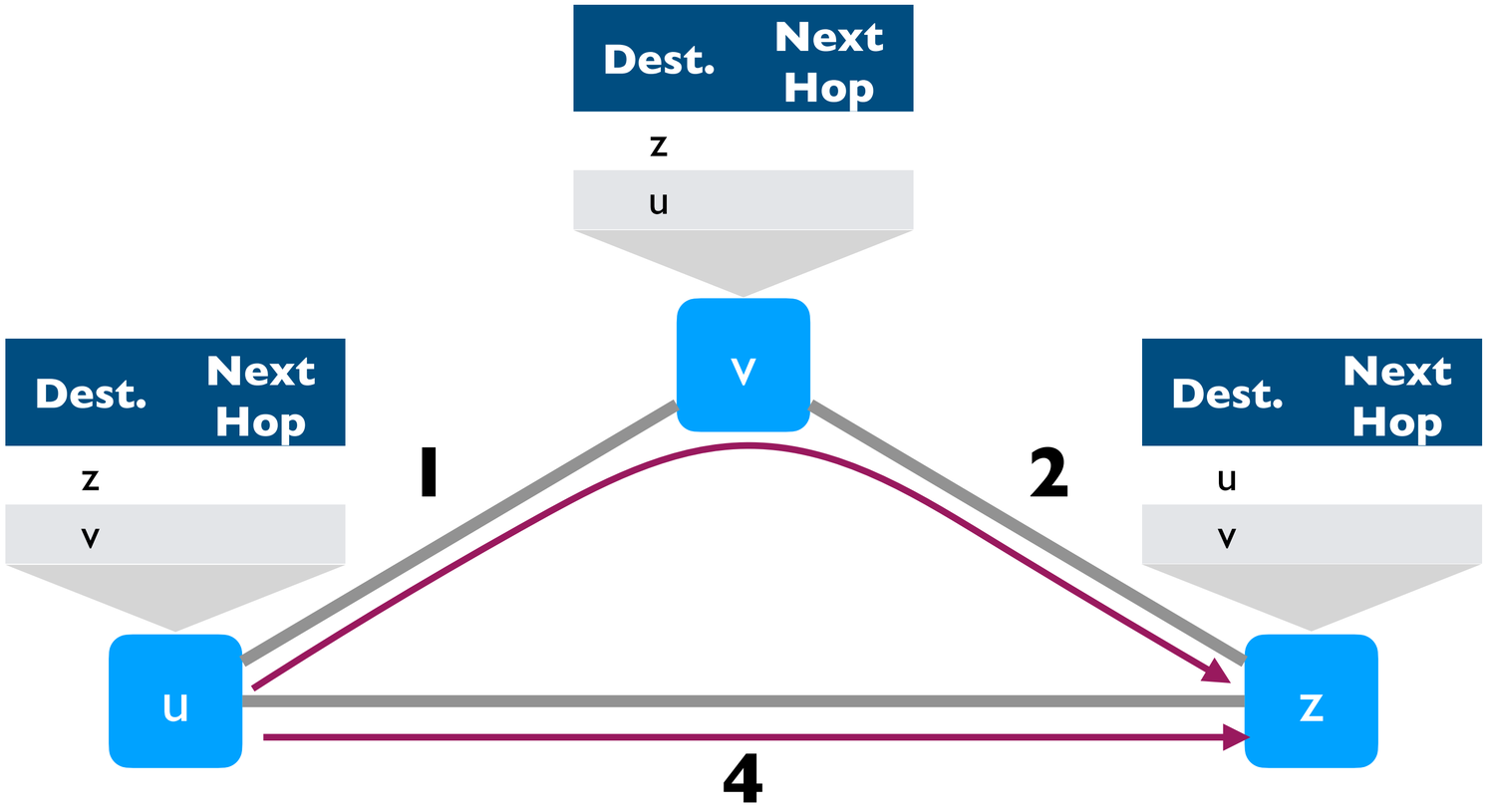

Dest.
Next Hop
z
u
v
Dest.
Next Hop
z
v
u
Dest.
Next Hop
u
v
z
1
2
4



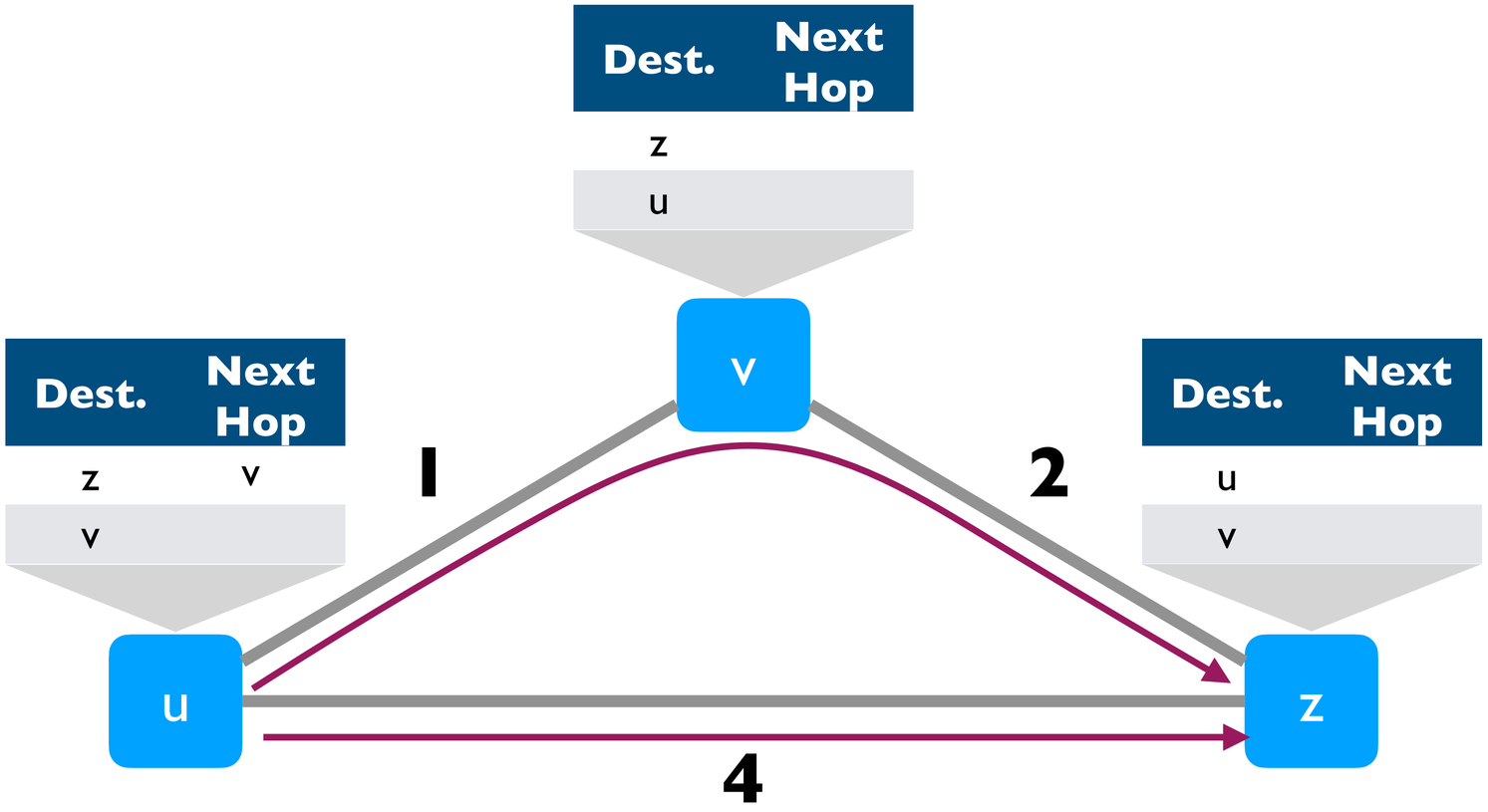
Least-cost path from **u** to **z**:



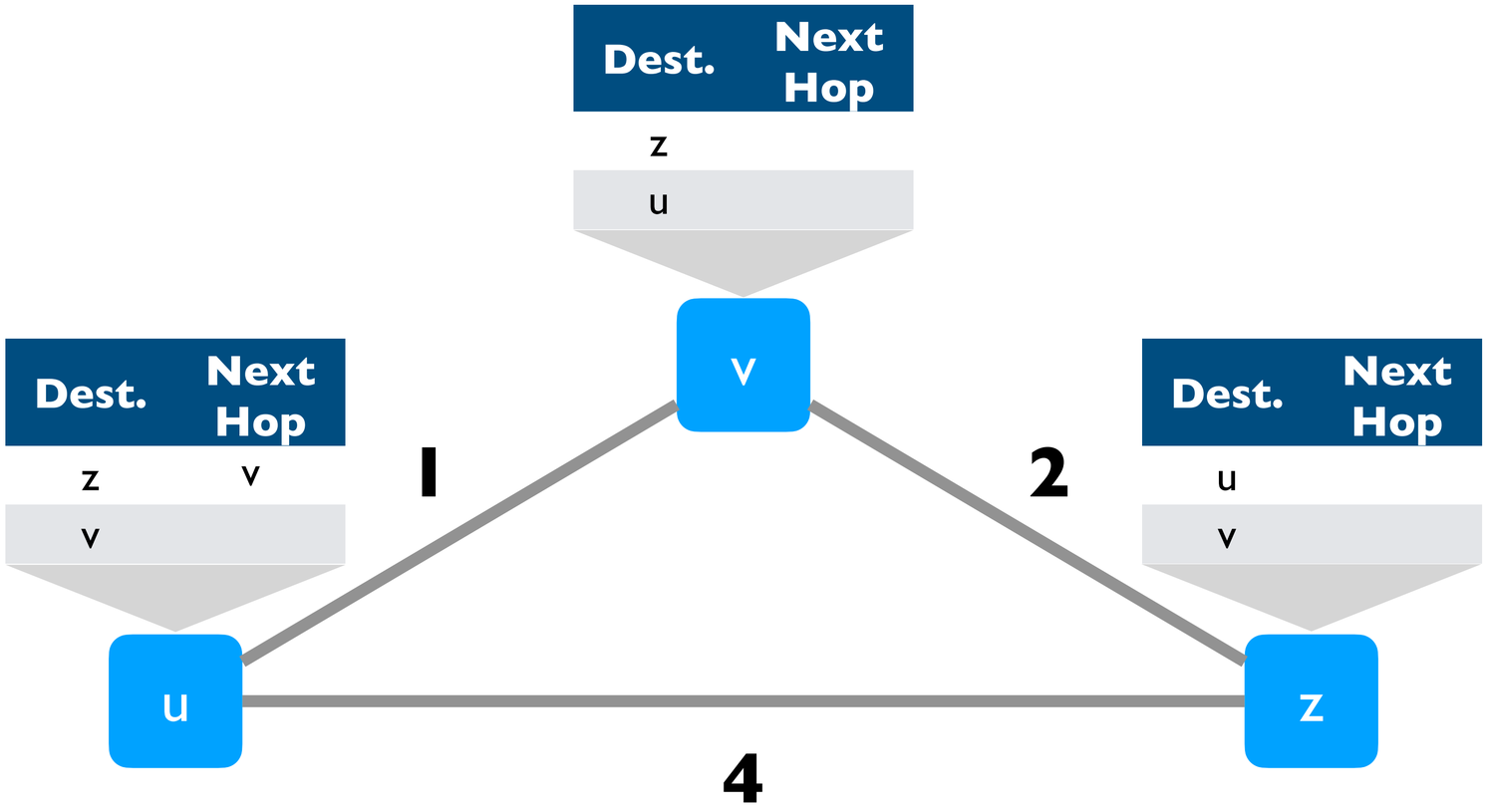
Least-cost path from **u** to **z**:



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

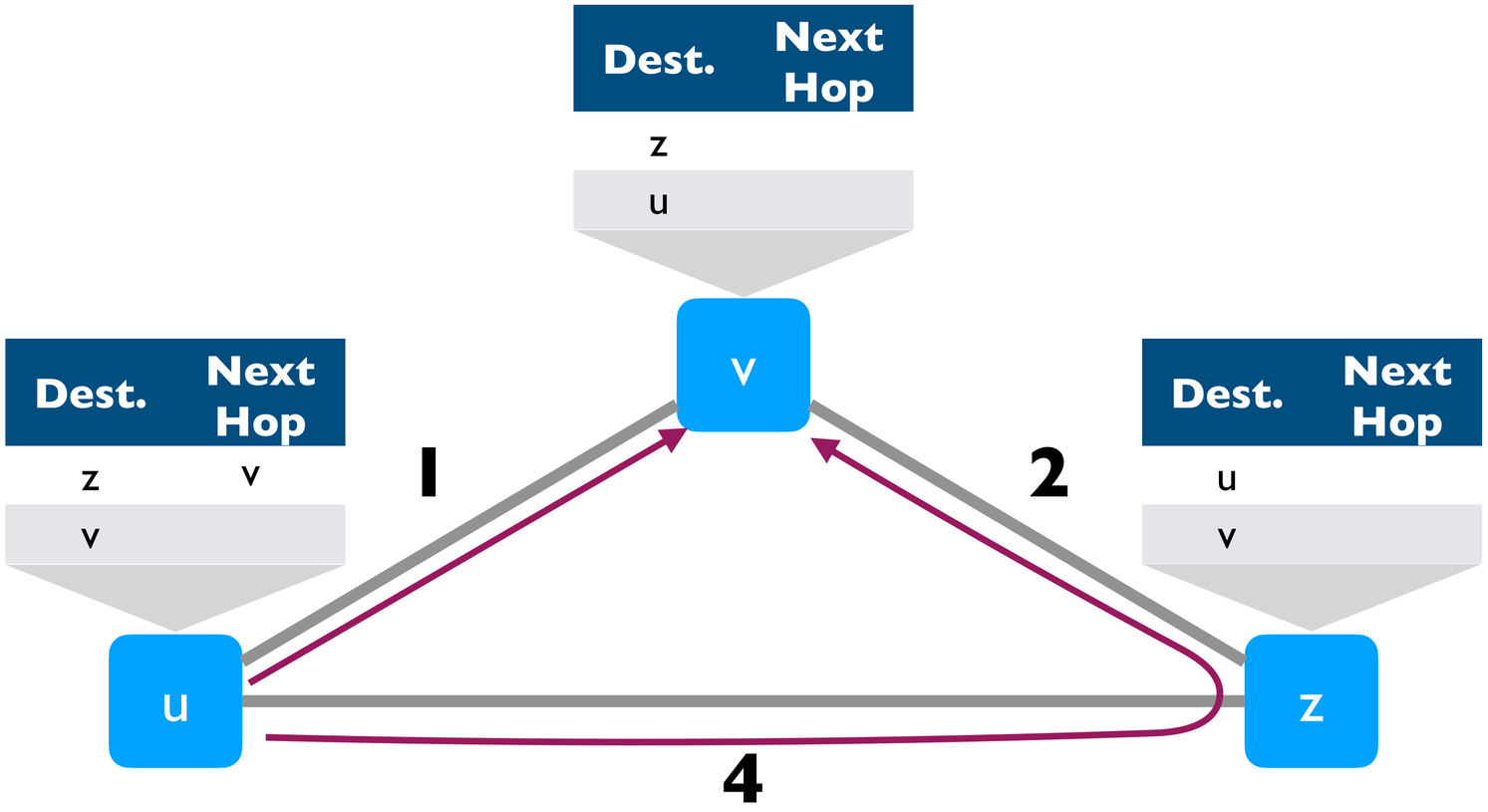


Least-cost path from **u** to **z**: **u v z**



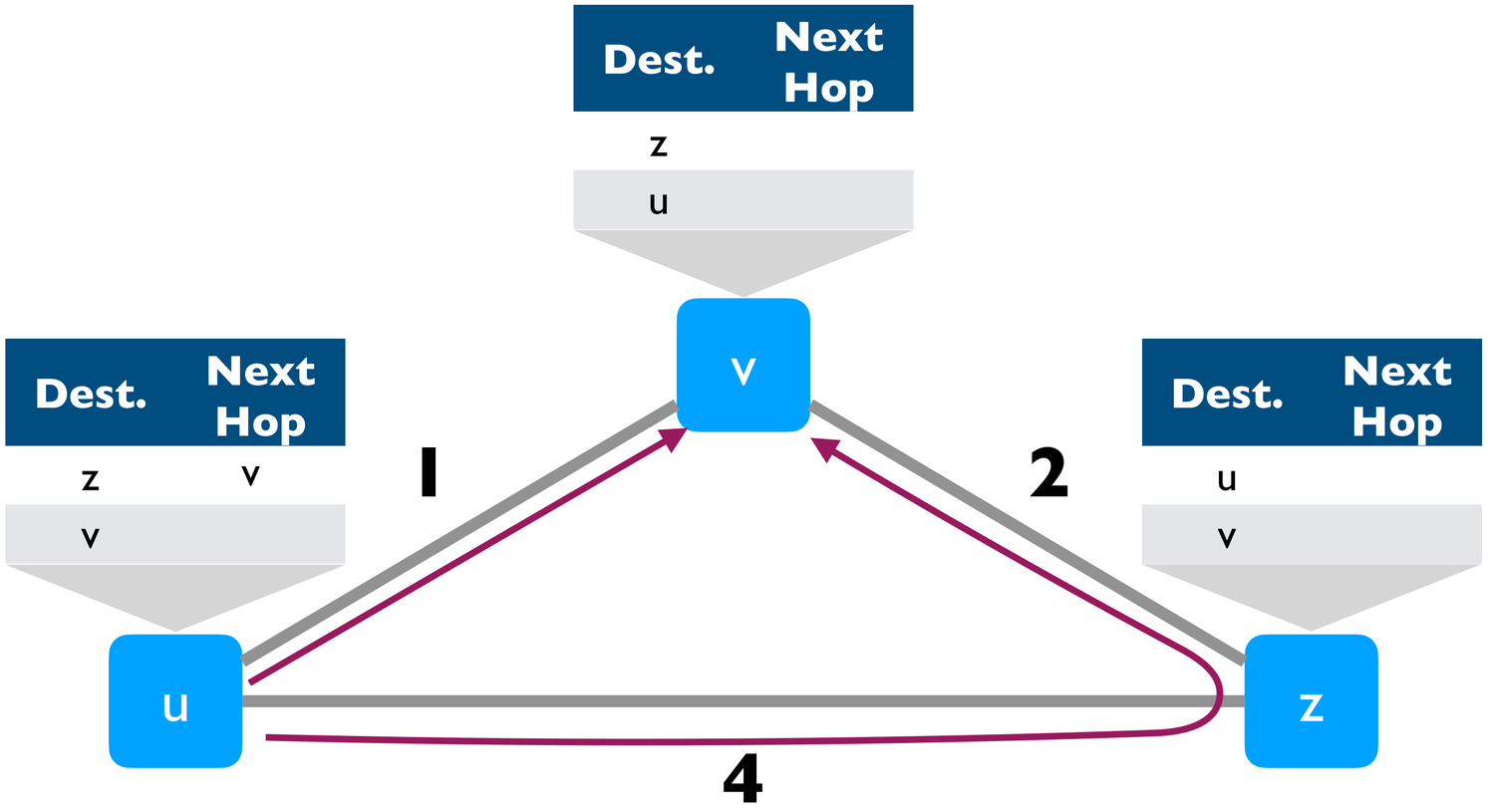
Least-cost path from **u** to **z**: **u v z**

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



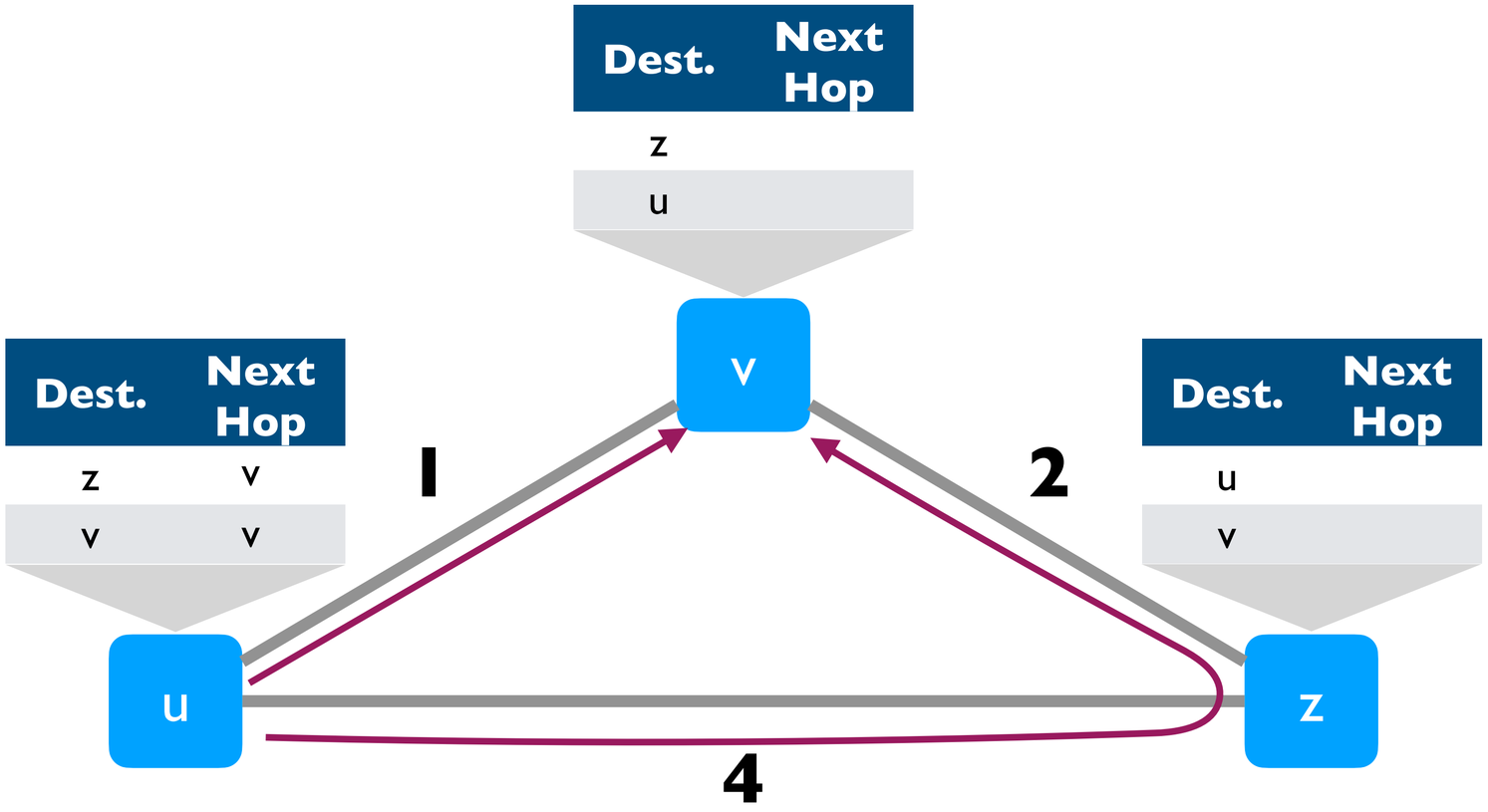
Least-cost path from **u** to **z**: **u v z**

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



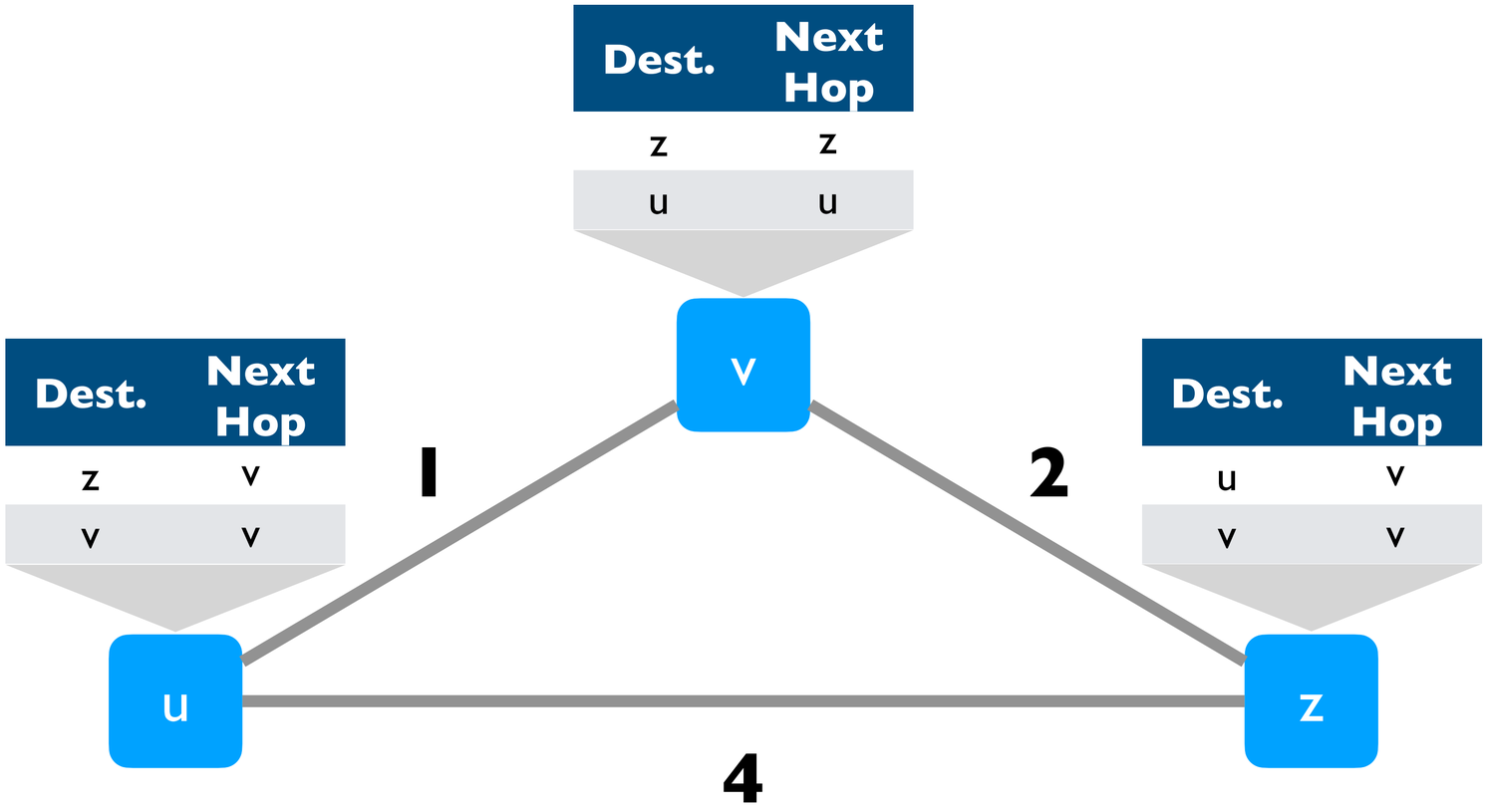
Least-cost path from **u** to **z**: **u v z**

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” Routes

“Least Cost” Routes

- “Least cost” routes are an easy way to avoid loops
  - *No **sensible** cost metric is minimized by traversing a loop*

“Least Cost” Routes

- “Least cost” routes are an easy way to avoid loops
  - *No **sensible** cost metric is minimized by traversing a loop*
- Least cost routes are also “destination-based”
  - *i.e., they do not depend on the source*

“Least Cost” Routes

- “Least cost” routes are an easy way to avoid loops
  - *No **sensible** cost metric is minimized by traversing a loop*
- Least cost routes are also “destination-based”
  - *i.e., they do not depend on the source*
- Least cost paths form a spanning tree

Outline

- Approach 1: link-state routing
- Approach 2: distance-vector routing
- Routing in the internet

Outline

- Approach 1: link-state routing
- Approach 2: distance-vector routing
- Routing in the internet

Assumptions

Assumptions

- Each router knows what the entire network looks like

Assumptions

- Each router knows what the entire network looks like
- The network can be abstracted as a graph **G = (N, E)**
  Nodes **N** corresponding to routers
  Edges **E** correspond to links
  Each edge has an associated cost

Dijkstra’s Algorithm

Dijkstra's Algorithm

- Source router considers every other router
  - *Starting from next "closest" neighbor (minimum cost)*

Dijkstra's Algorithm

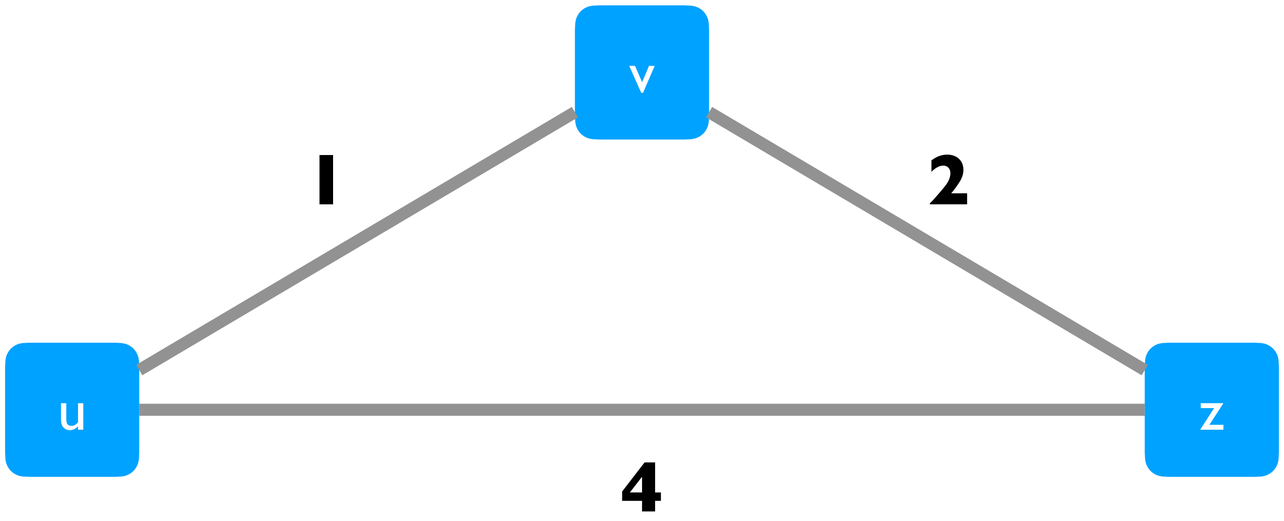
- Source router considers every other router
  - *Starting from next "closest" neighbor (minimum cost)*
- Checks whether it can improve current paths
  - *By using that router as an intermediate point*

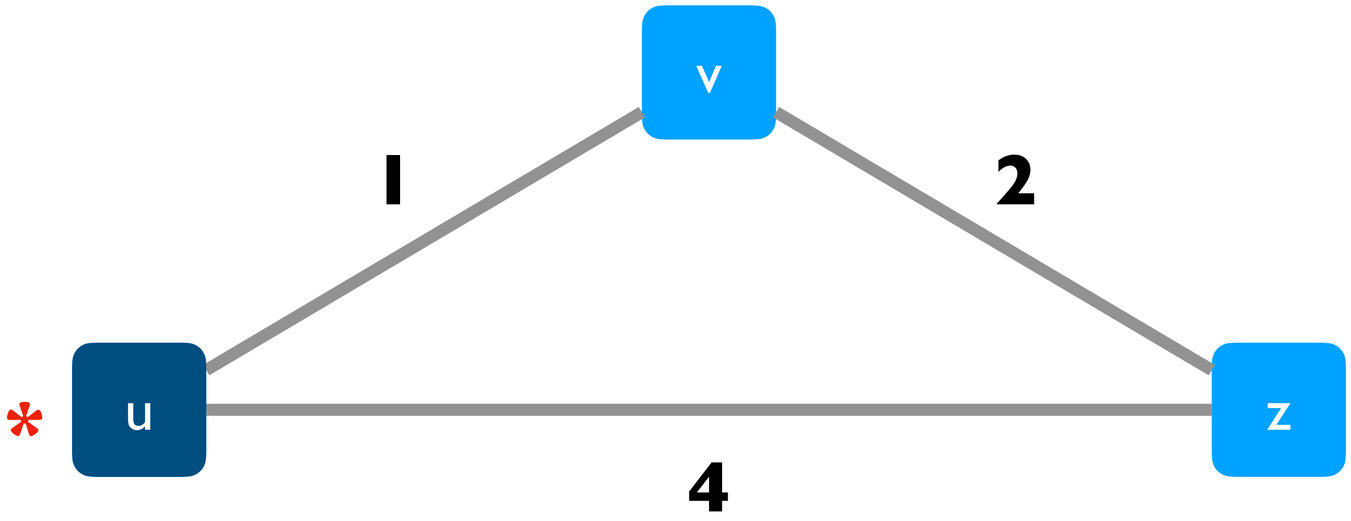
Dijkstra's Algorithm

- Source router considers every other router
  - *Starting from next "closest" neighbor (minimum cost)*
- Checks whether it can improve current paths
  - *By using that router as an intermediate point*
- Ends when all intermediaries have been considered

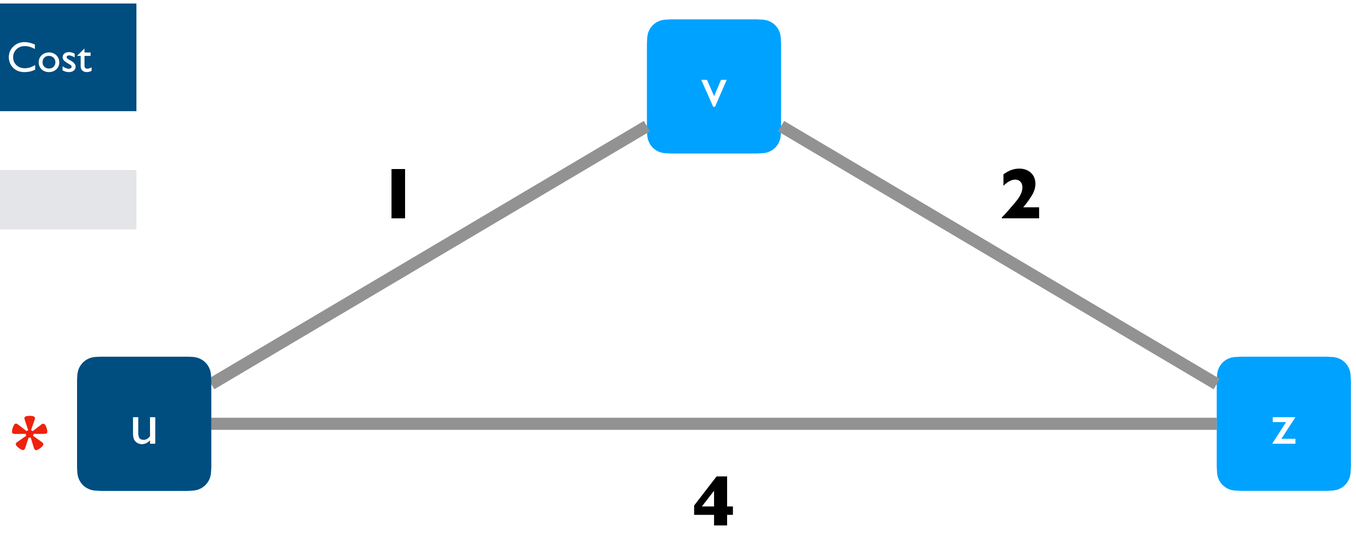
Dijkstra's Algorithm

- Source router considers every other router
  - *Starting from next "closest" neighbor (minimum cost)*
- Checks whether it can improve current paths
  - *By using that router as an intermediate point*
- Ends when all intermediaries have been considered
- See text for exact algorithm

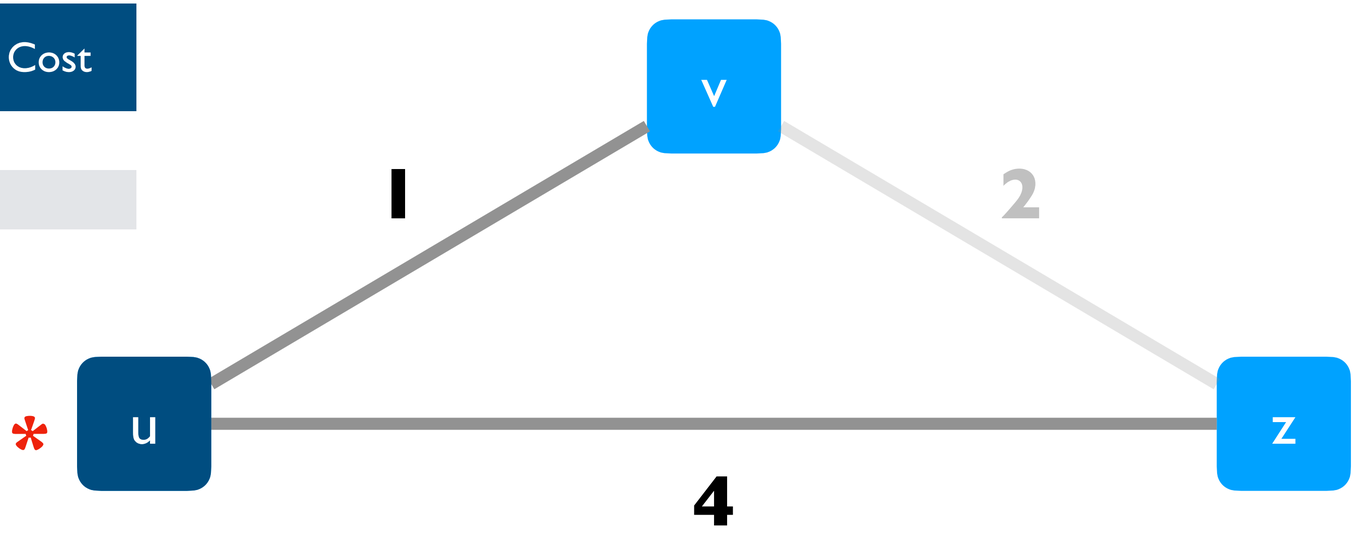

v
1
2
u
z
4


v
1
2
*
u
z
4

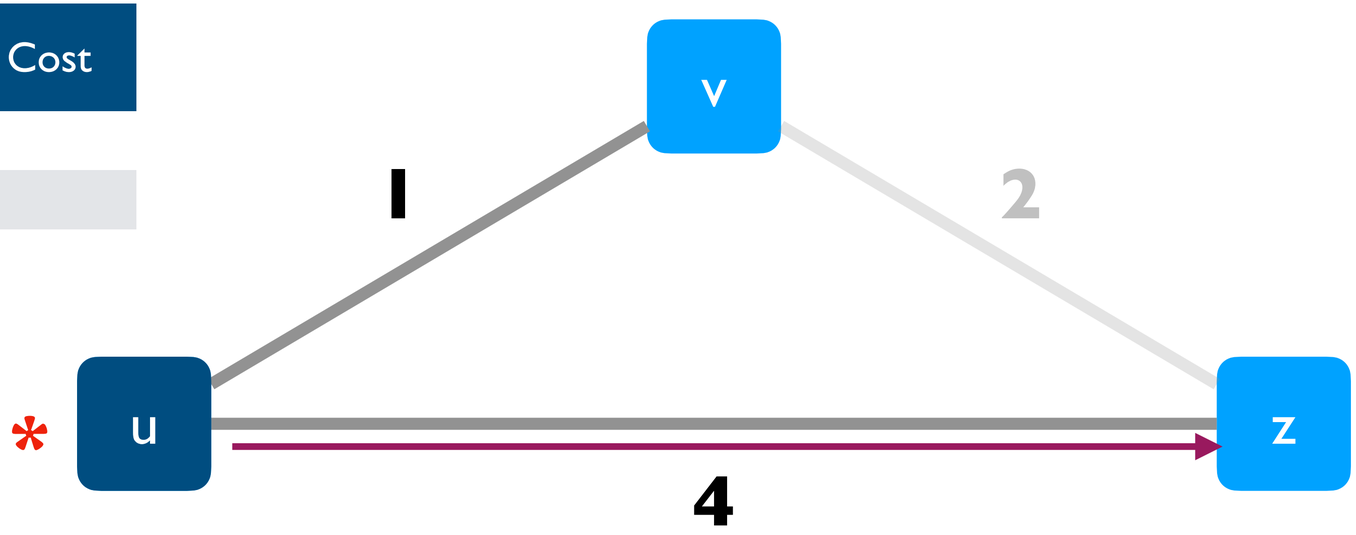
Dest.	Next Hop	Cost
z		
v		



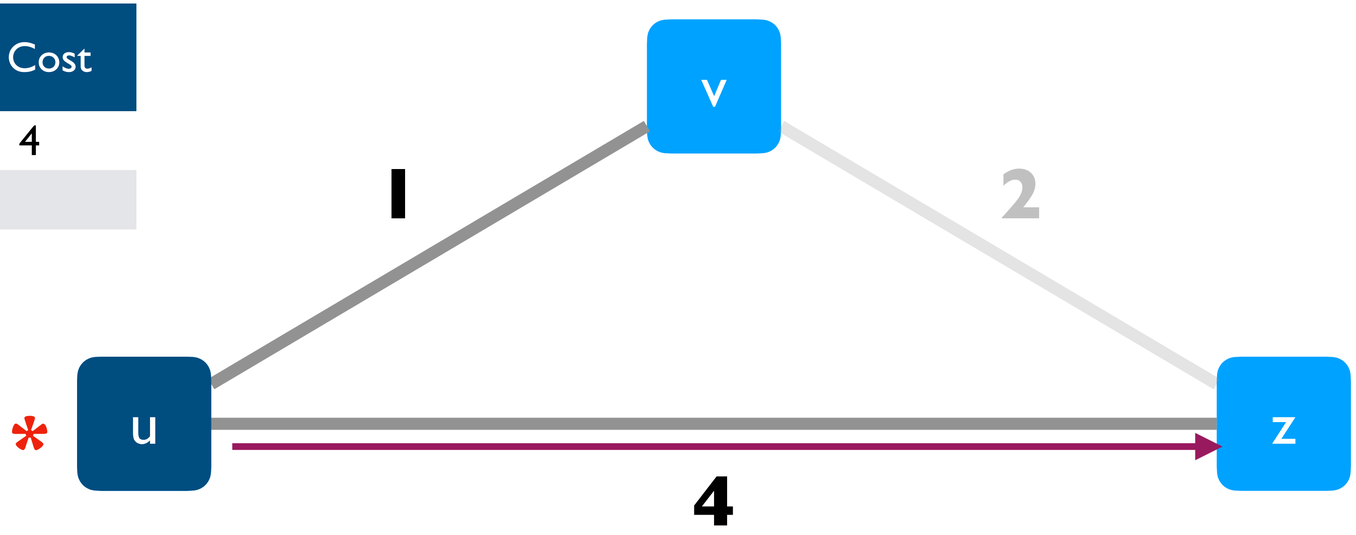
Dest.	Next Hop	Cost
z		
v		



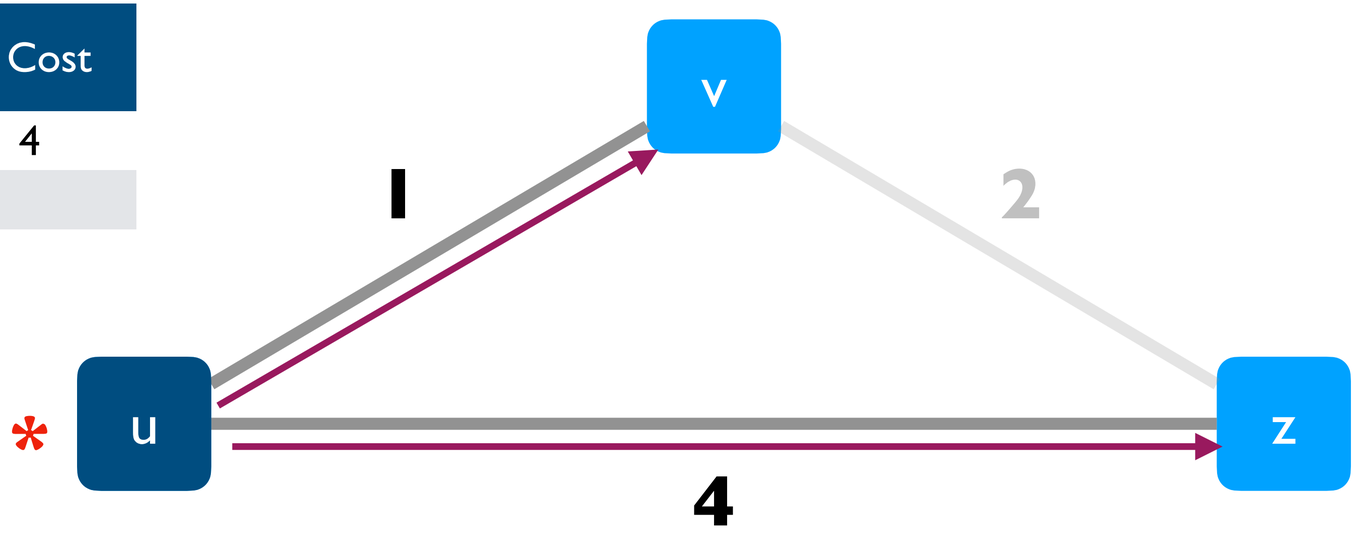
Dest.	Next Hop	Cost
z		
v		



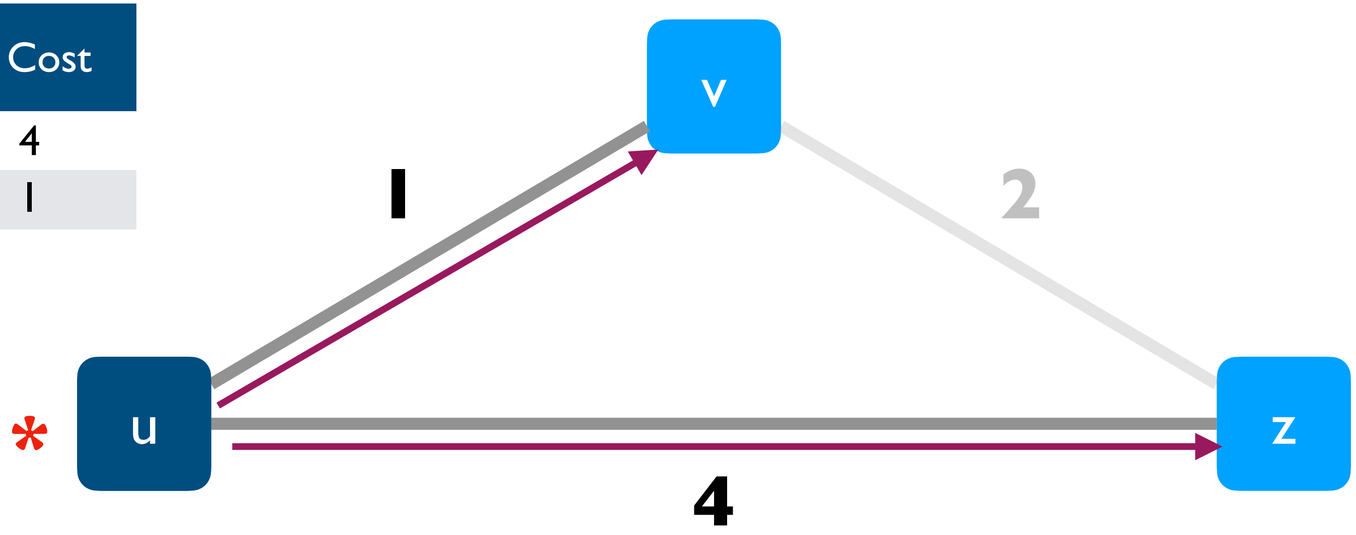
Dest.	Next Hop	Cost
z	z	4
v		



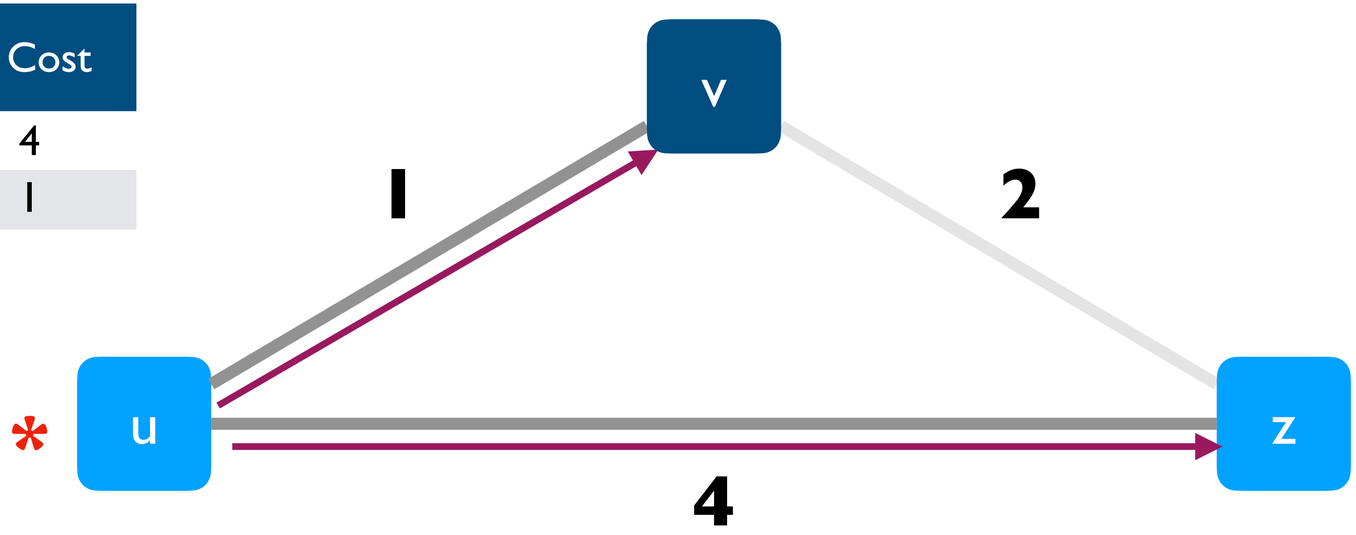
Dest.	Next Hop	Cost
z	z	4
v		



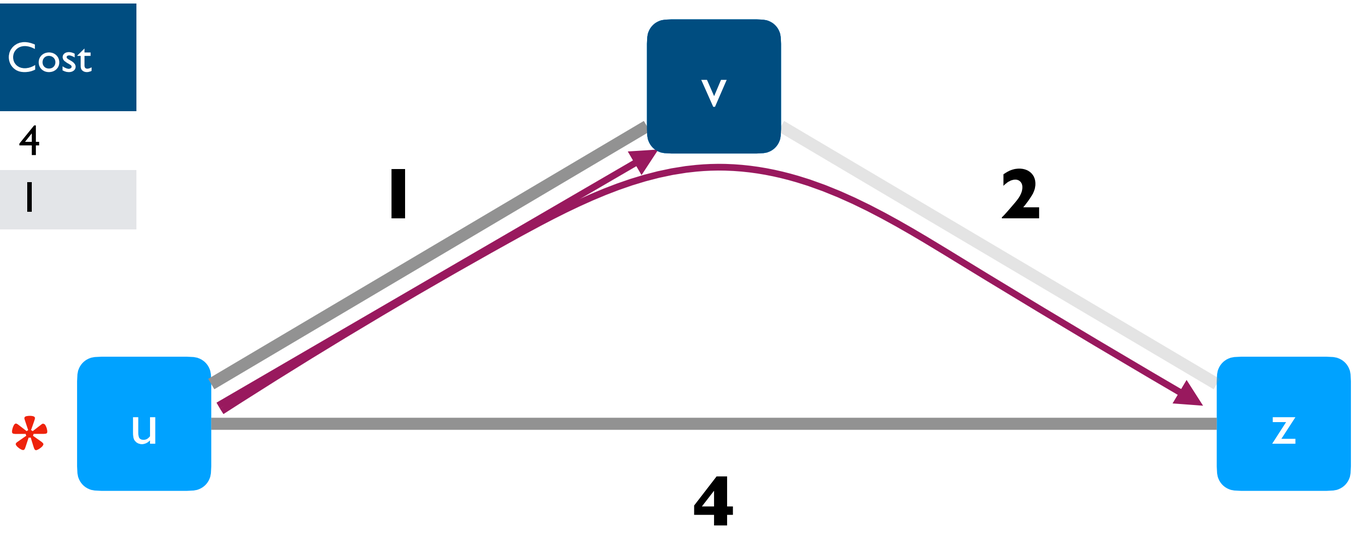
Dest.	Next Hop	Cost
z	z	4
v	v	1



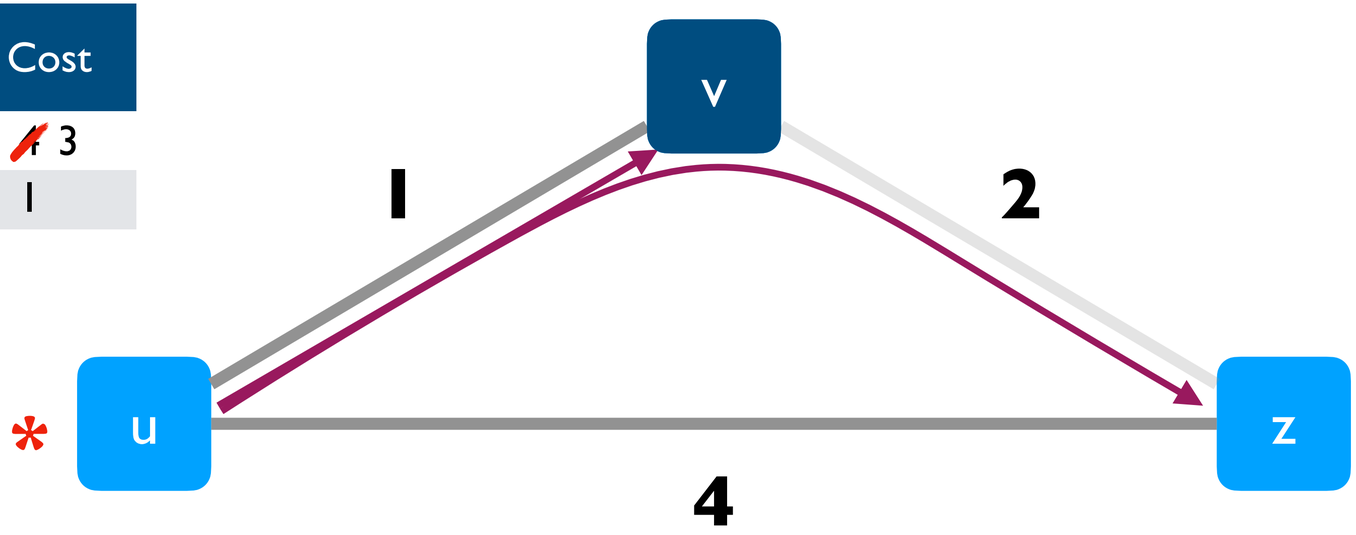
Dest.	Next Hop	Cost
z	z	4
v	v	1



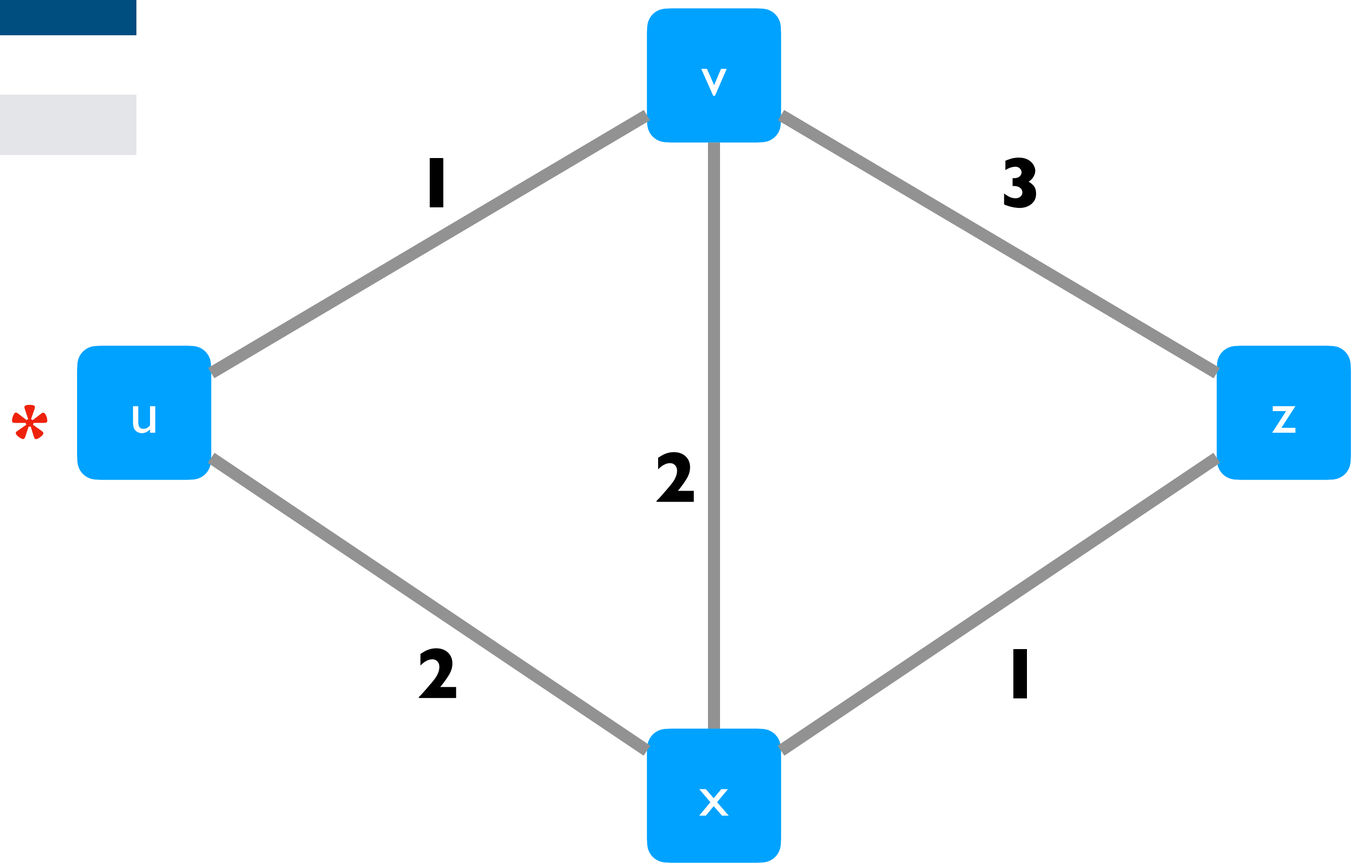
Dest.	Next Hop	Cost
z	z	4
v	v	1



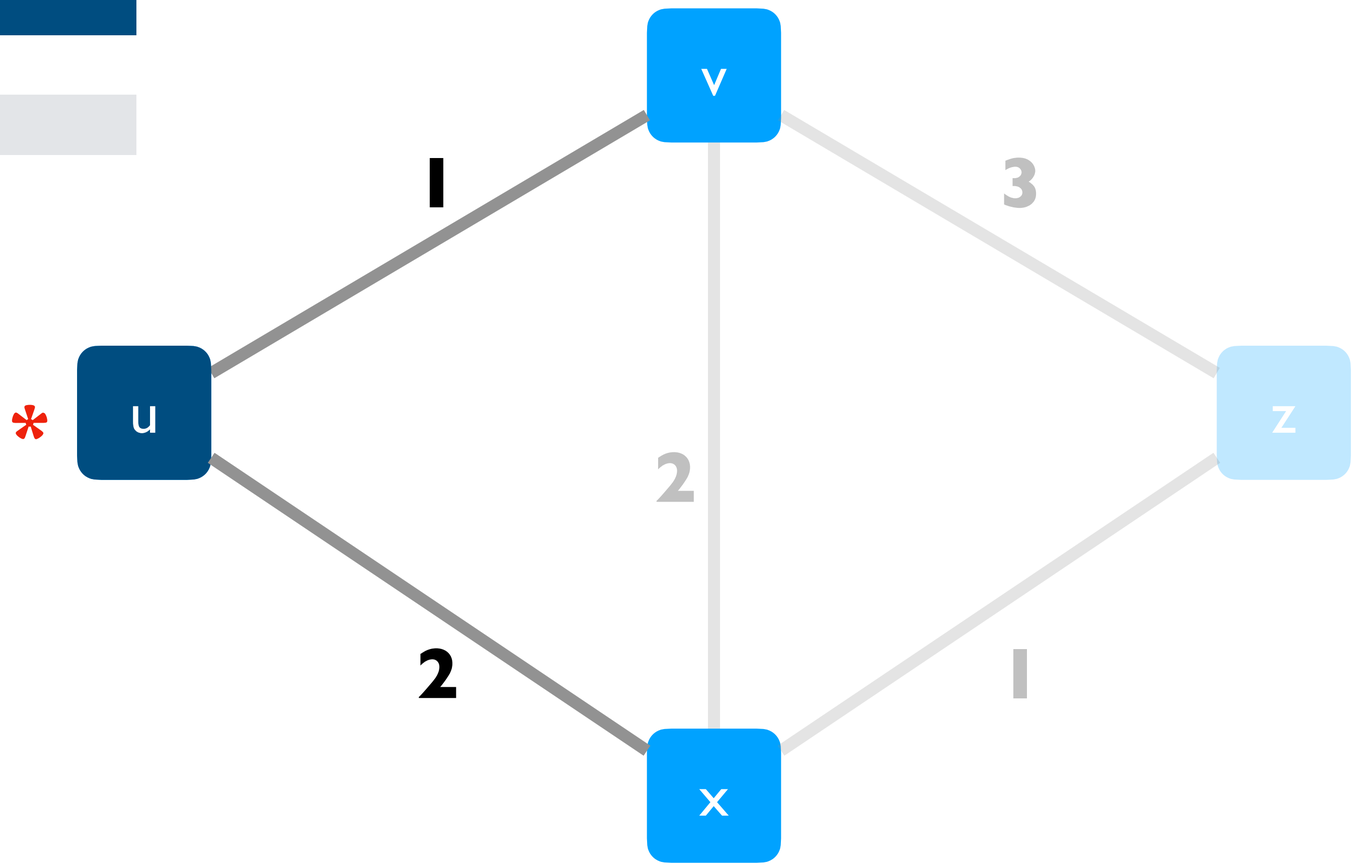
Dest.	Next Hop	Cost
z	~~z~~ v	~~4~~ 3
v	v	1



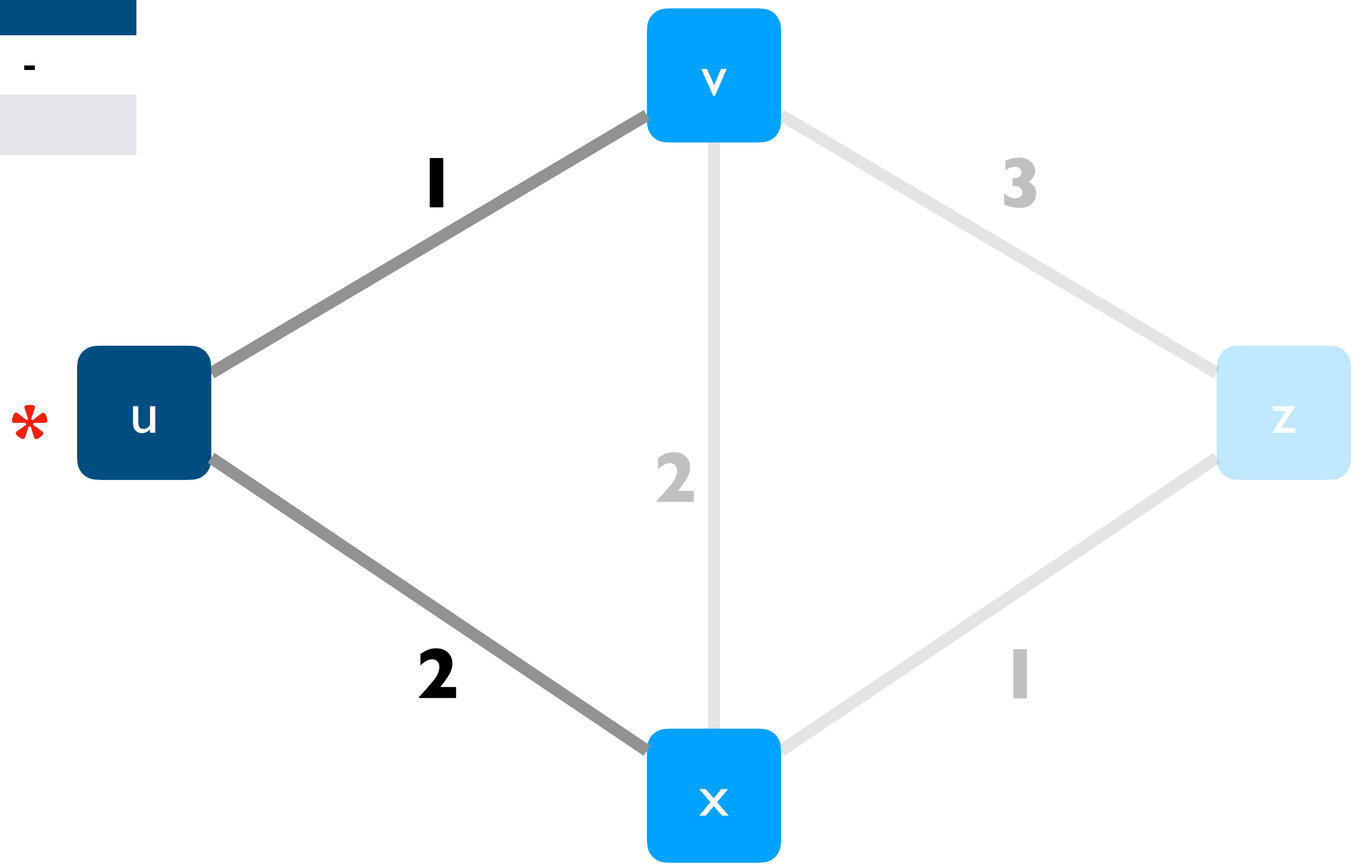
Dest.	Next Hop	Cost
z		
v		
x		



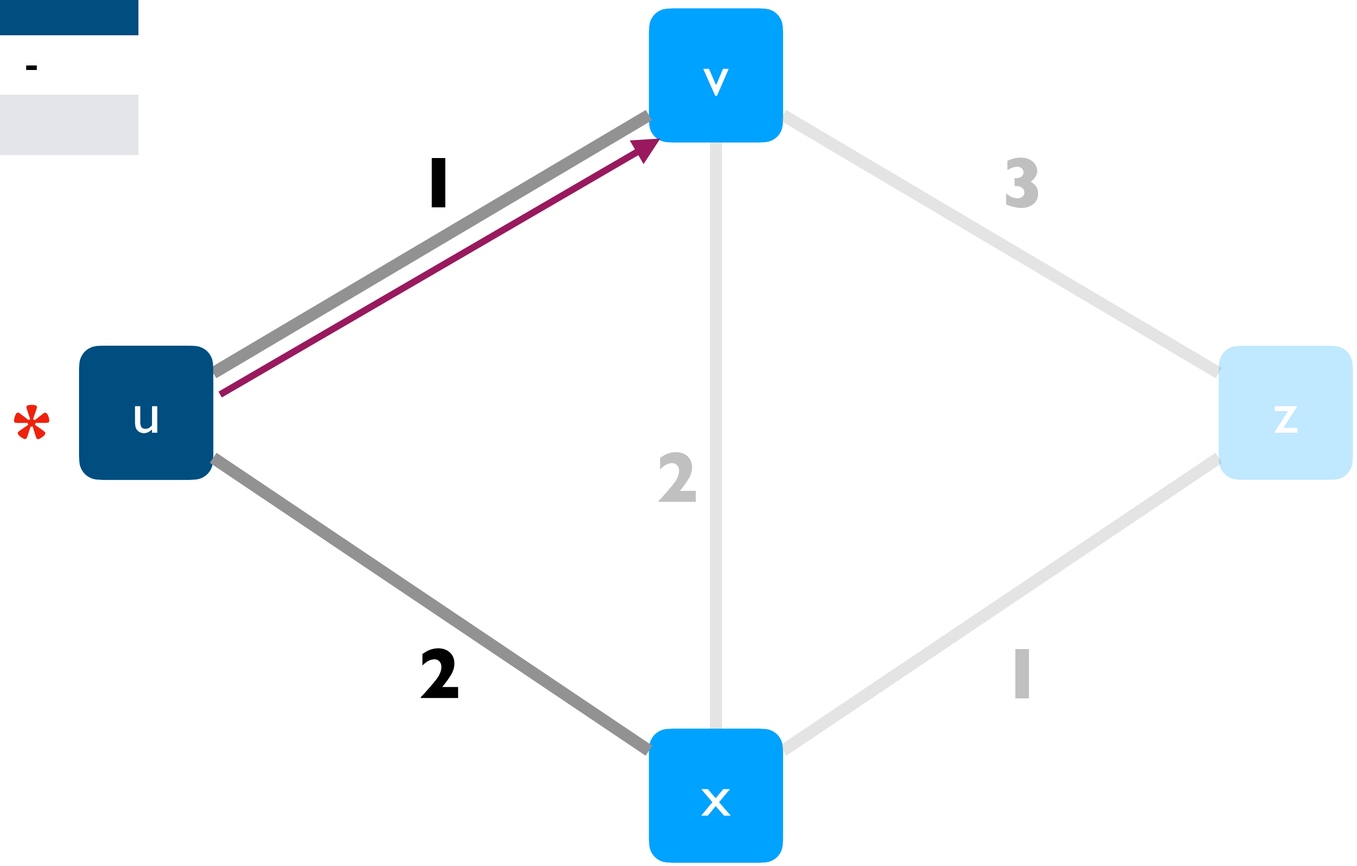
Dest.	Next Hop	Cost
z		
v		
x		



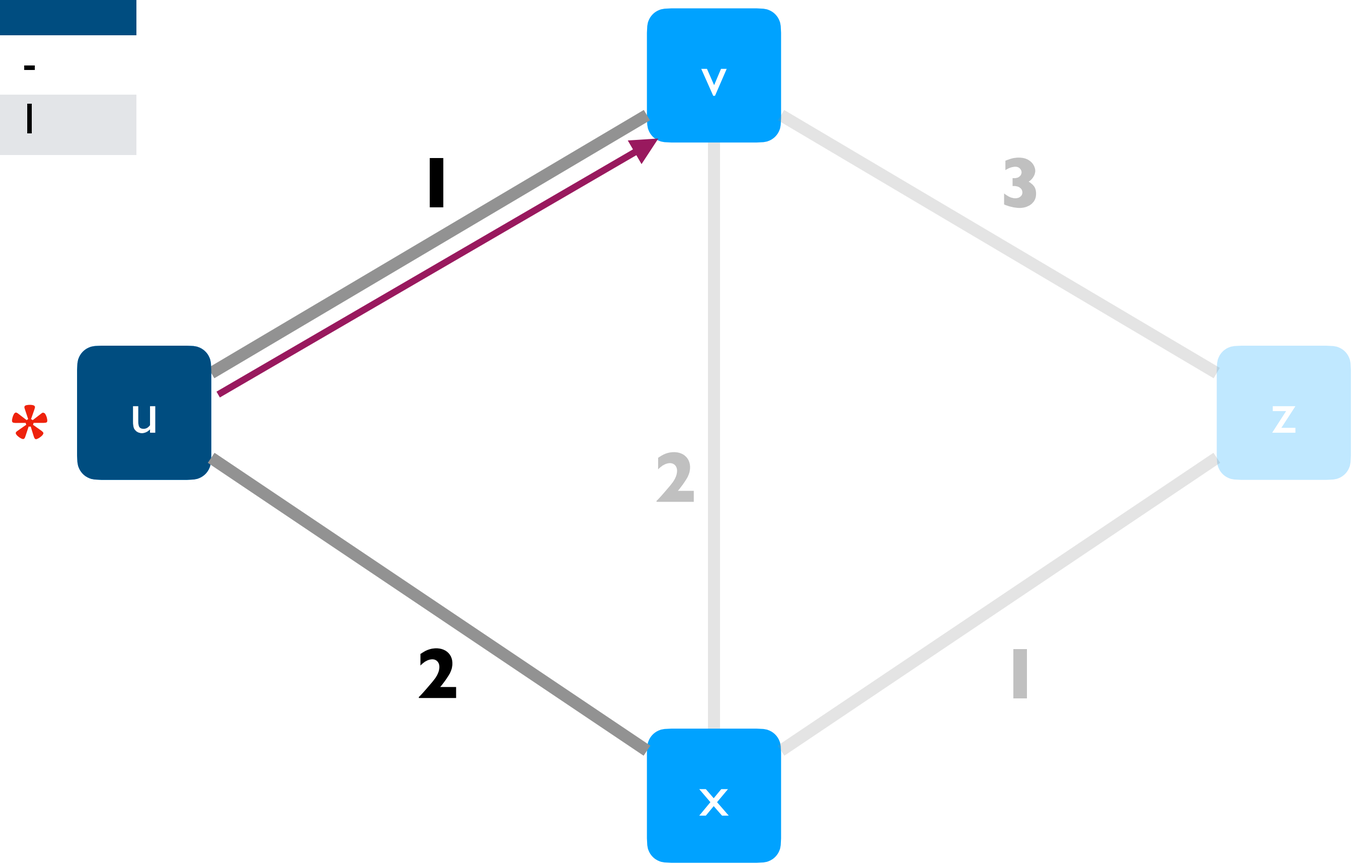
Dest.	Next Hop	Cost
z	-	-
v		
x		



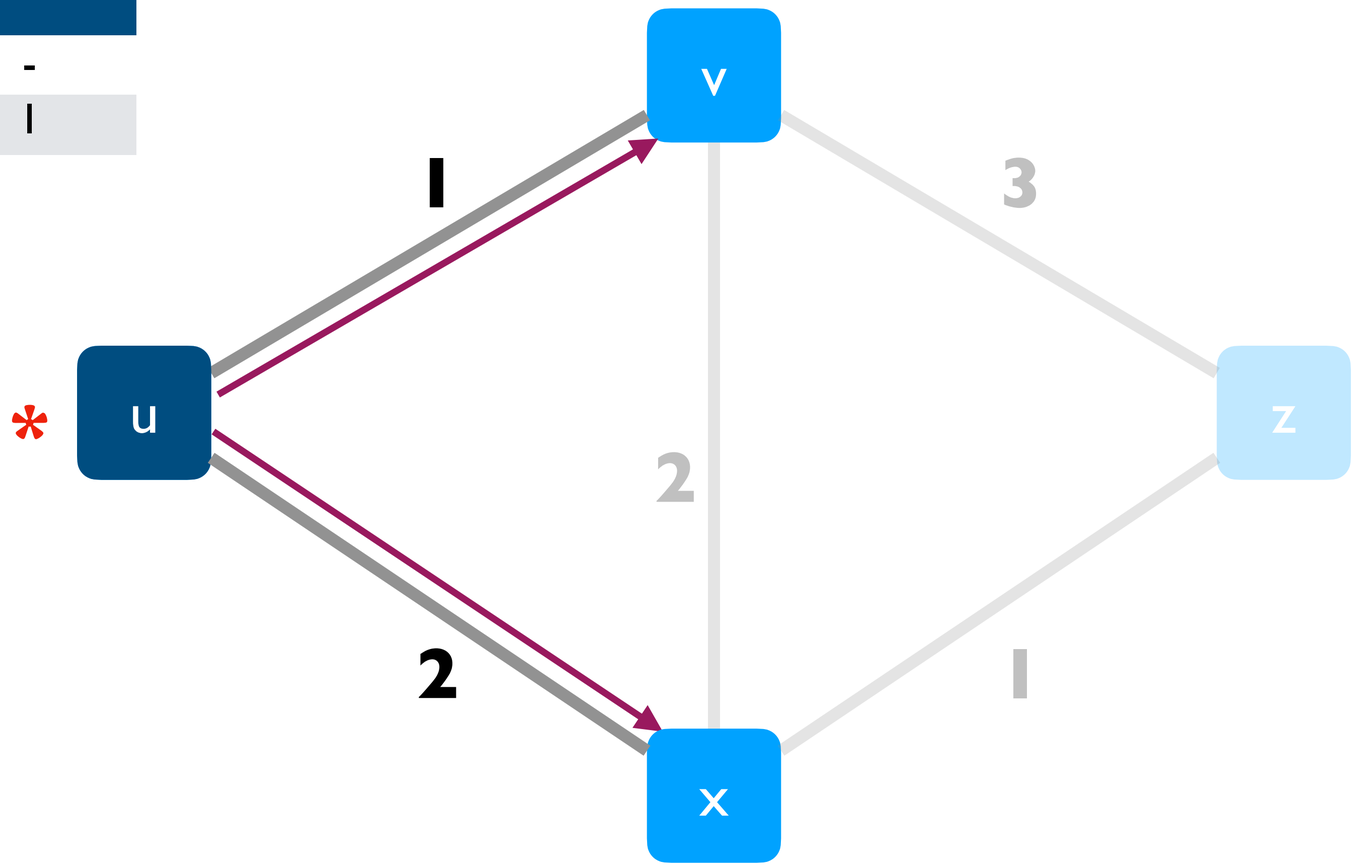
Dest.	Next Hop	Cost
z	-	-
v		
x		



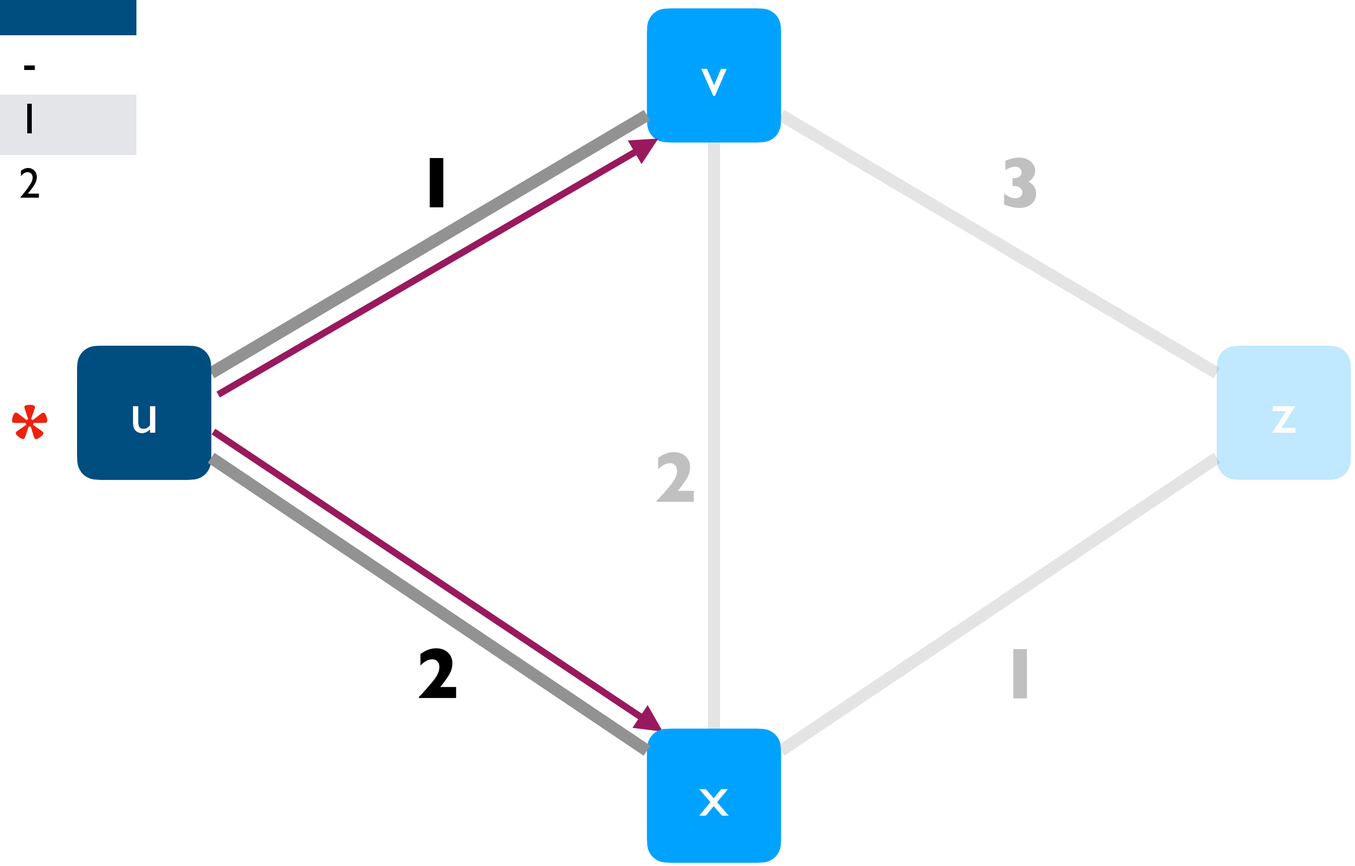
Dest.	Next Hop	Cost
z	-	-
v	v	1
x		



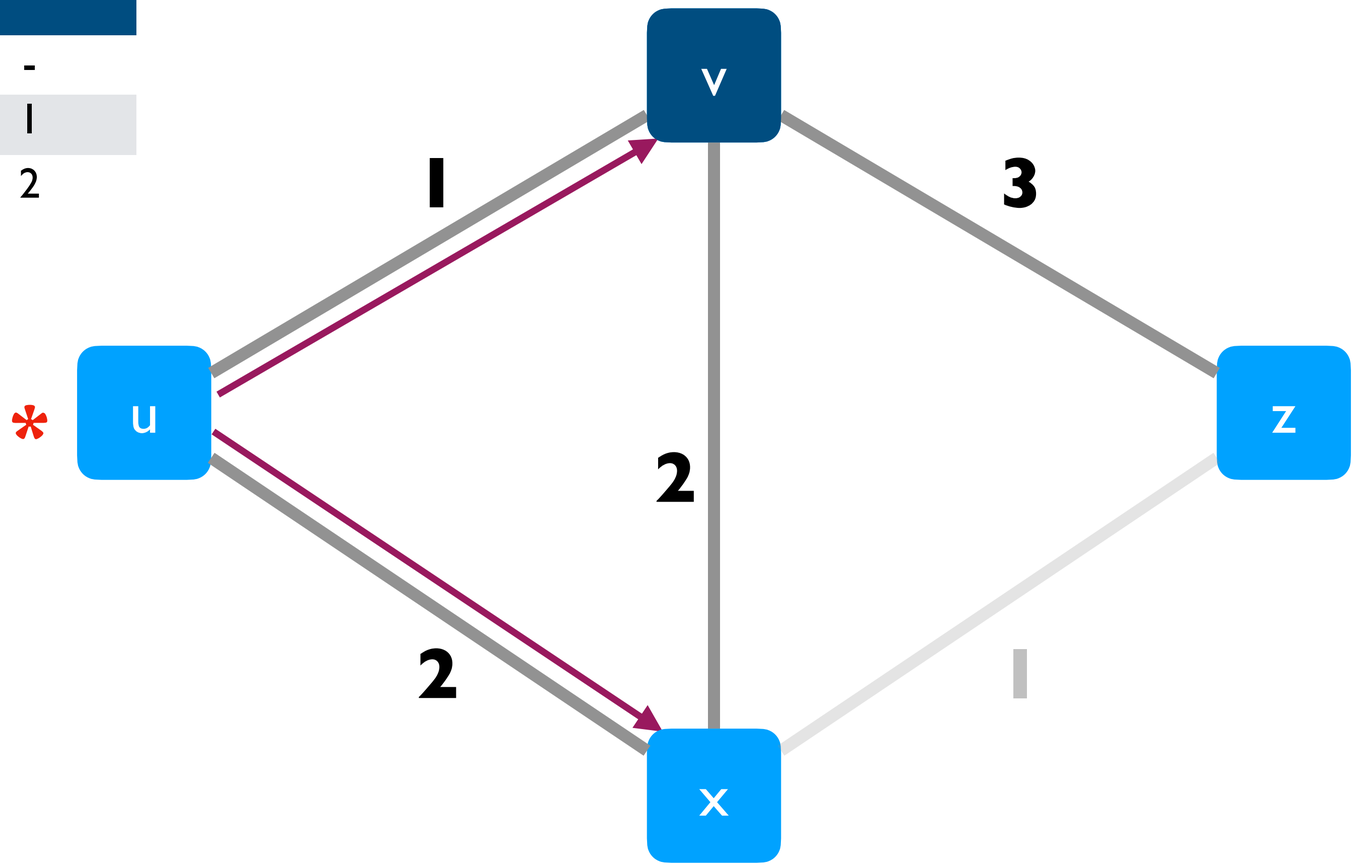
Dest.	Next Hop	Cost
z	-	-
v	v	1
x		



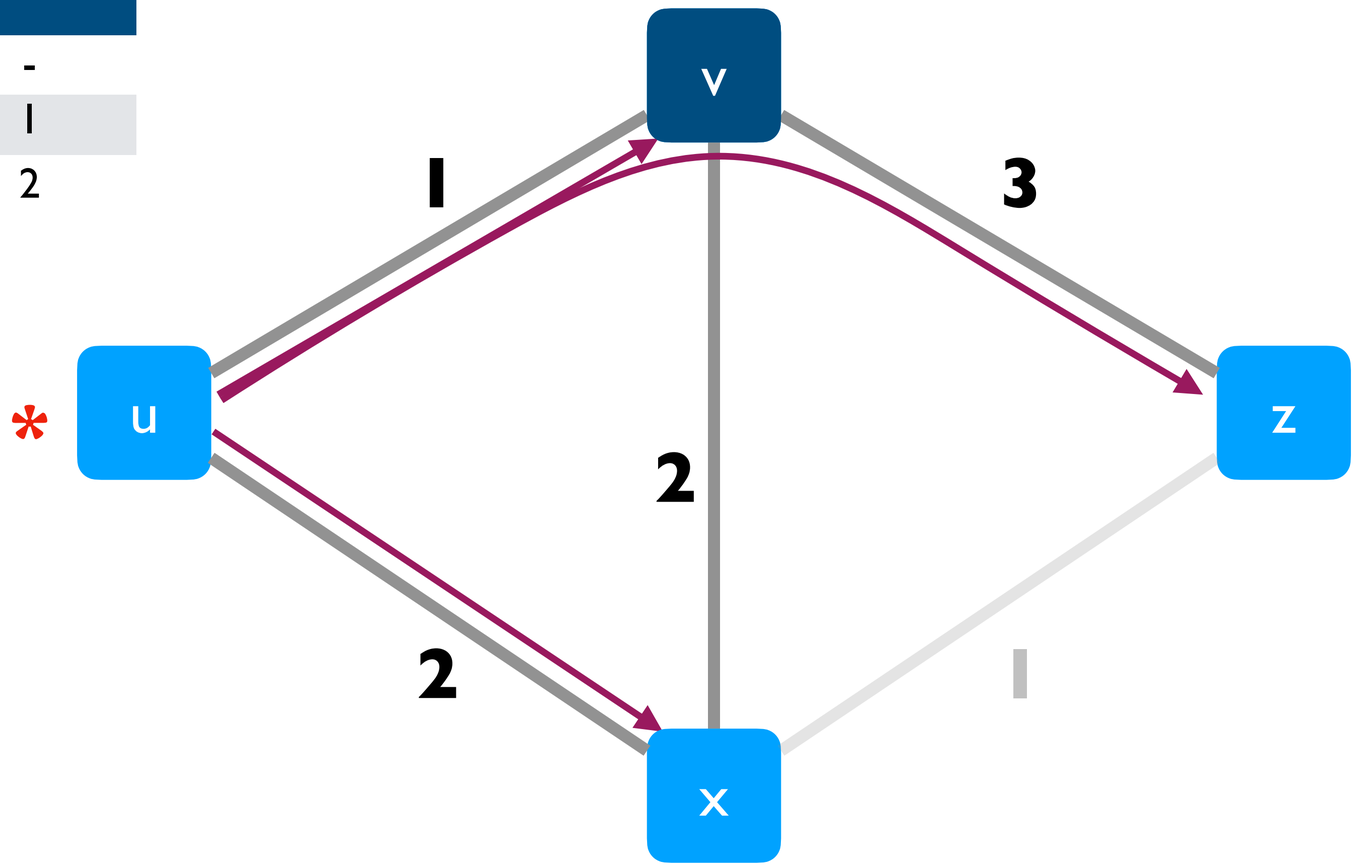
Dest.	Next Hop	Cost
z	-	-
v	v	1
x	x	2



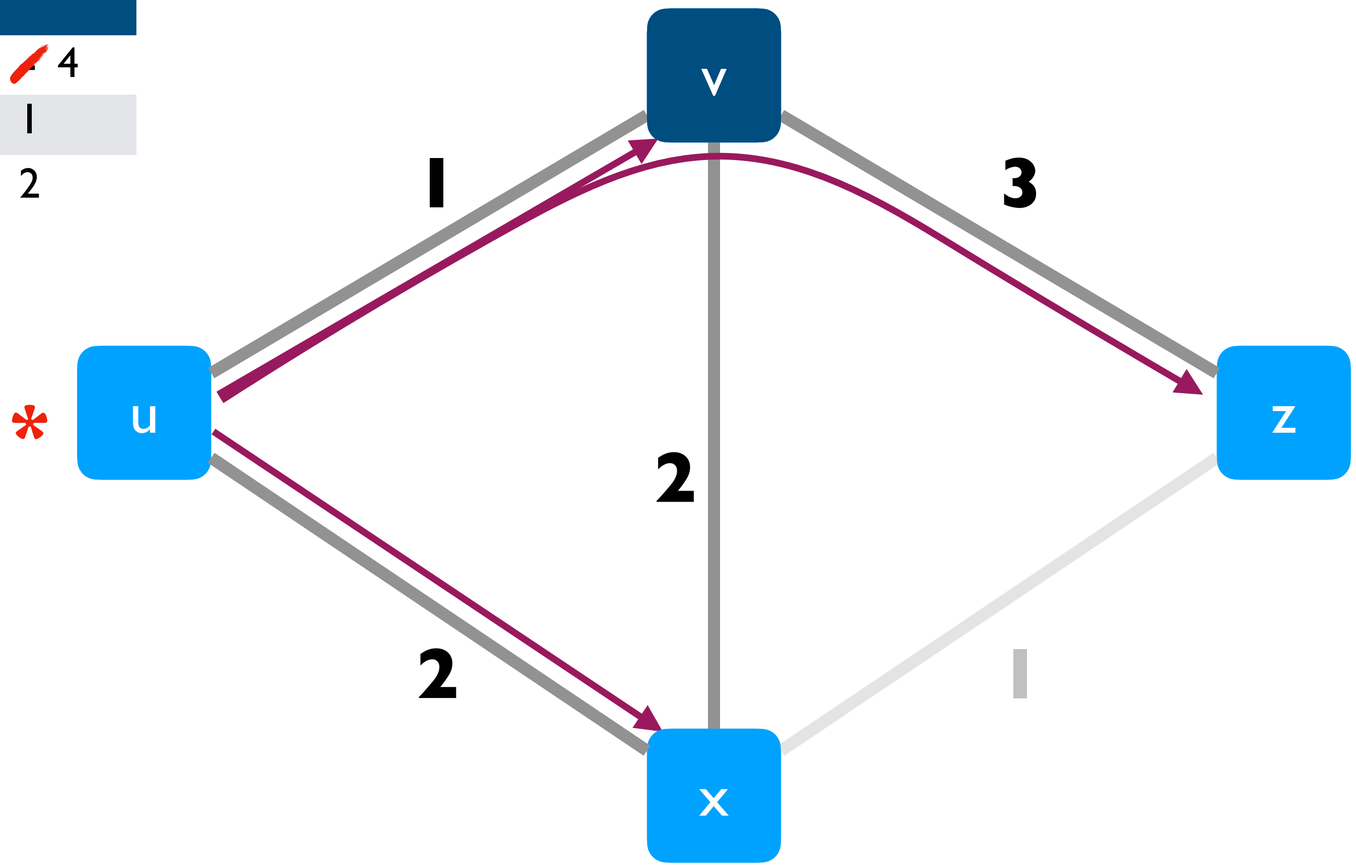
Dest.	Next Hop	Cost
z	-	-
v	v	1
x	x	2



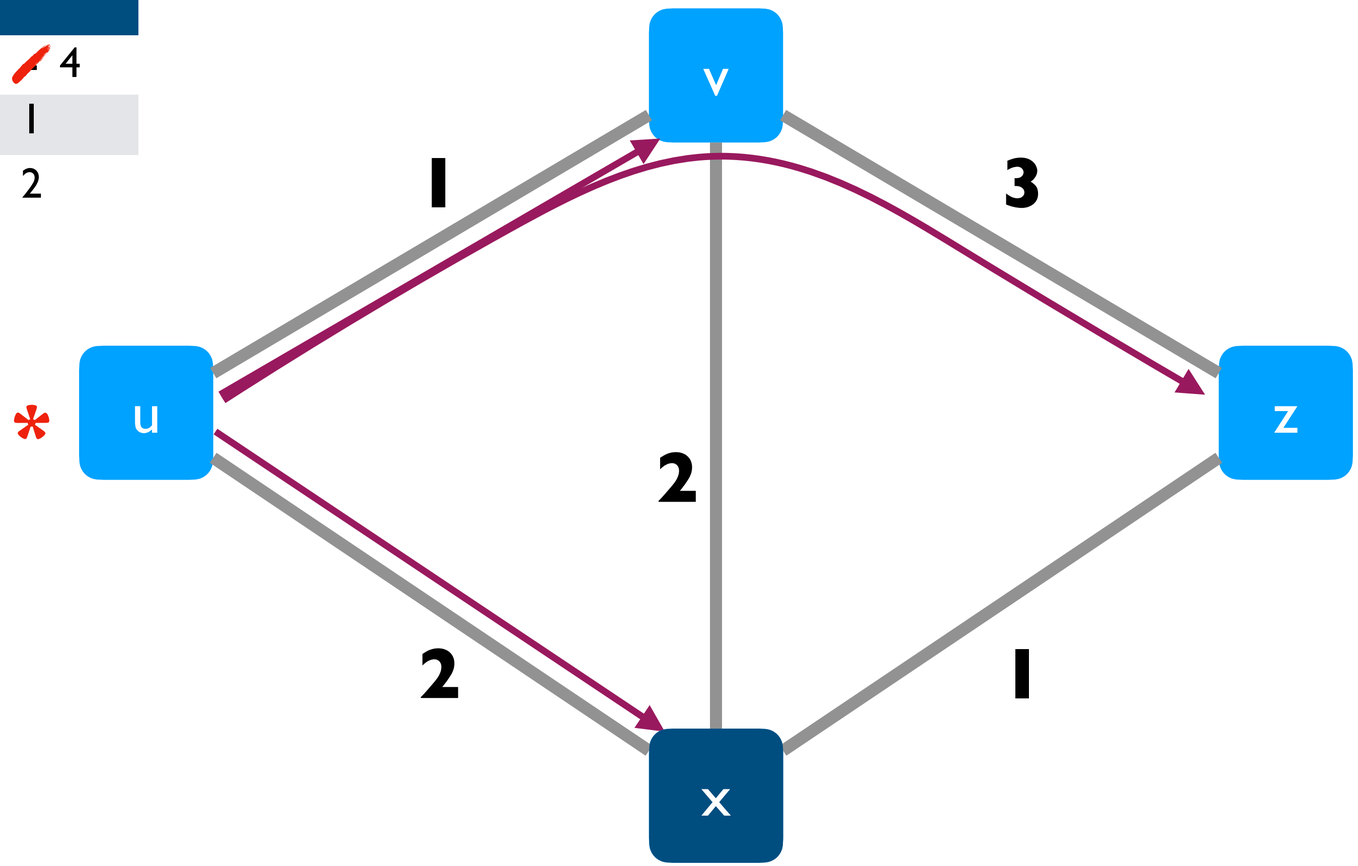
Dest.	Next Hop	Cost
z	-	-
v	v	1
x	x	2



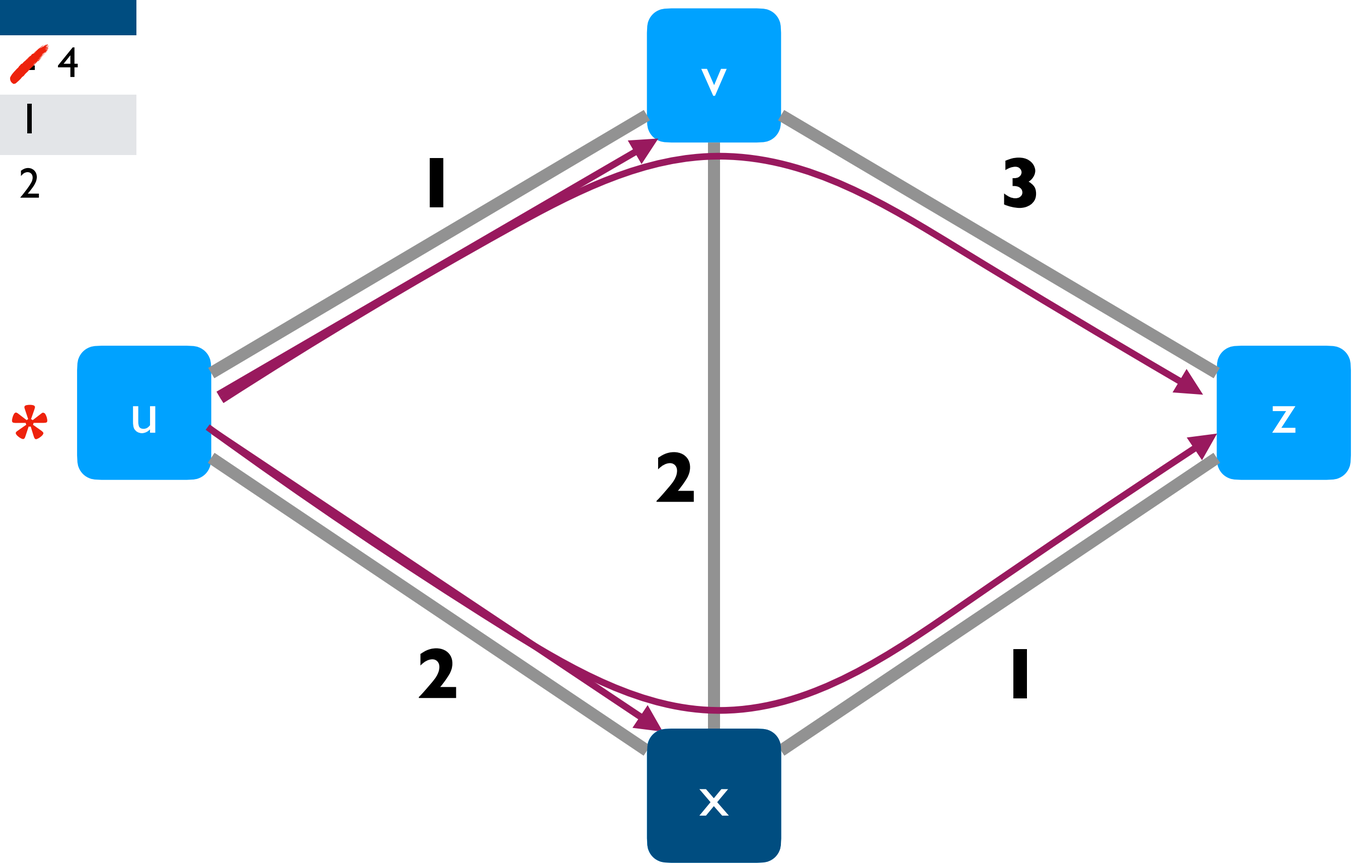
Dest.	Next Hop	Cost
z	~~-~~ v	~~-~~ 4
v	v	1
x	x	2



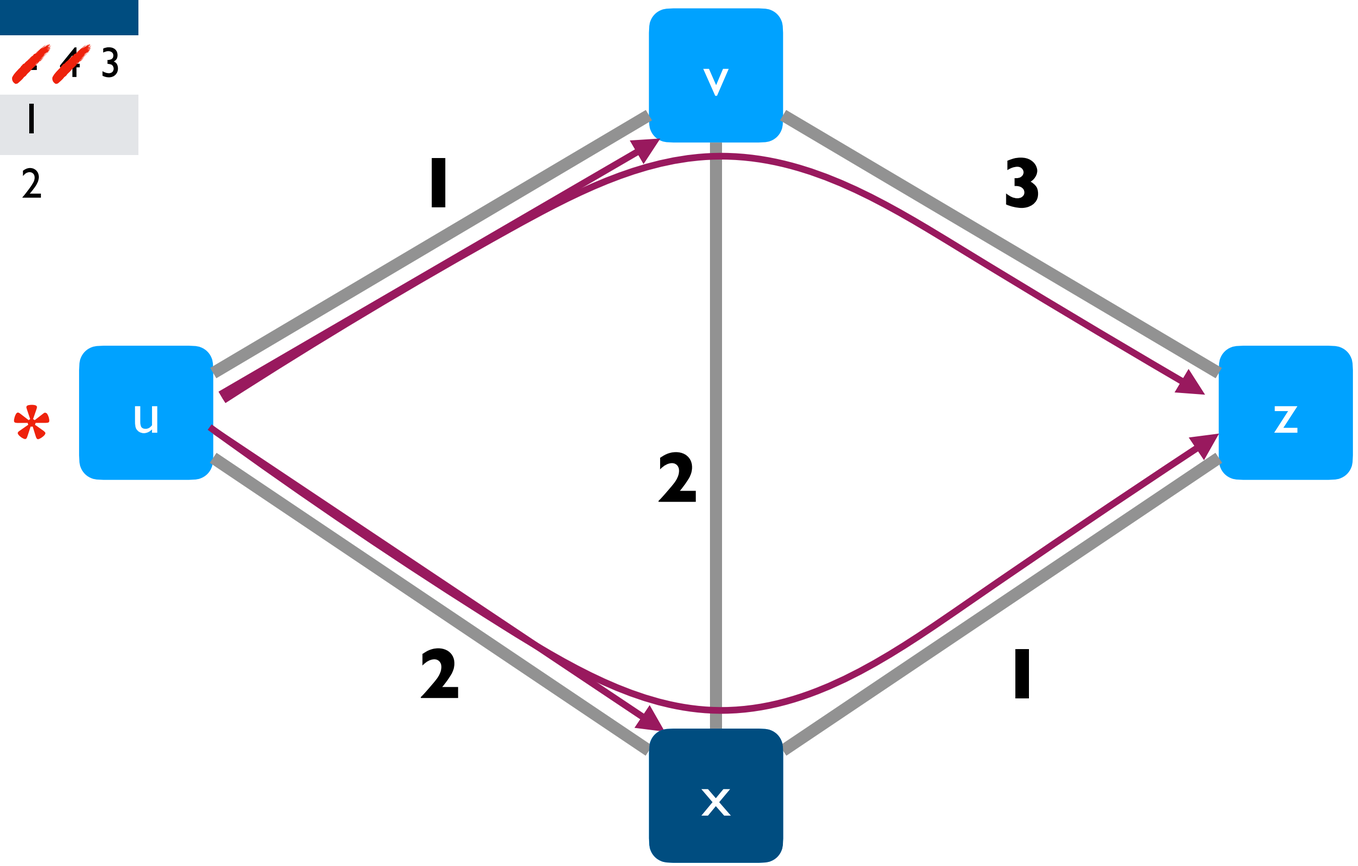
Dest.	Next Hop	Cost
z	~~-~~ v	~~-~~ 4
v	v	1
x	x	2



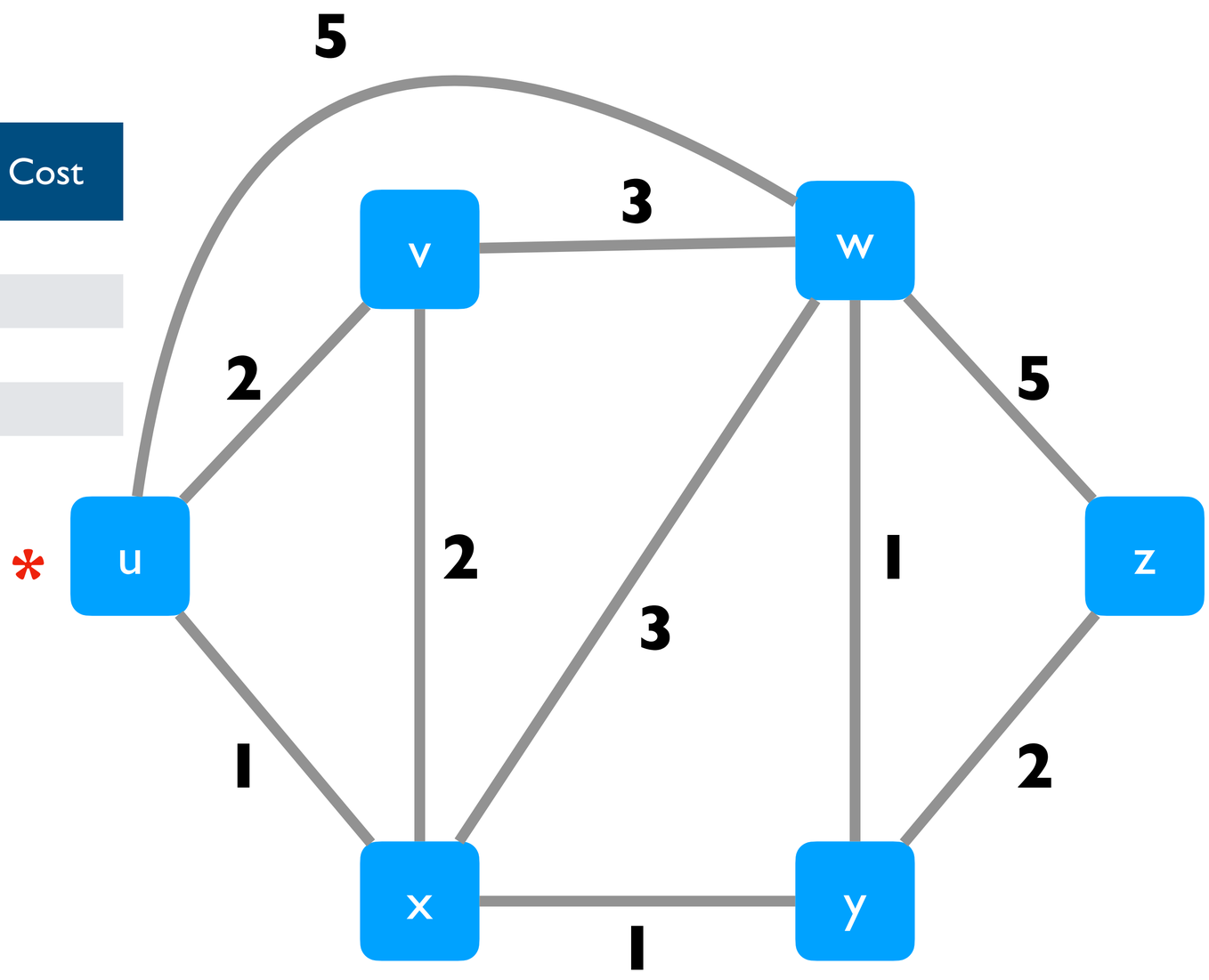
Dest.	Next Hop	Cost
z	~~-~~ v	~~-~~ 4
v	v	1
x	x	2



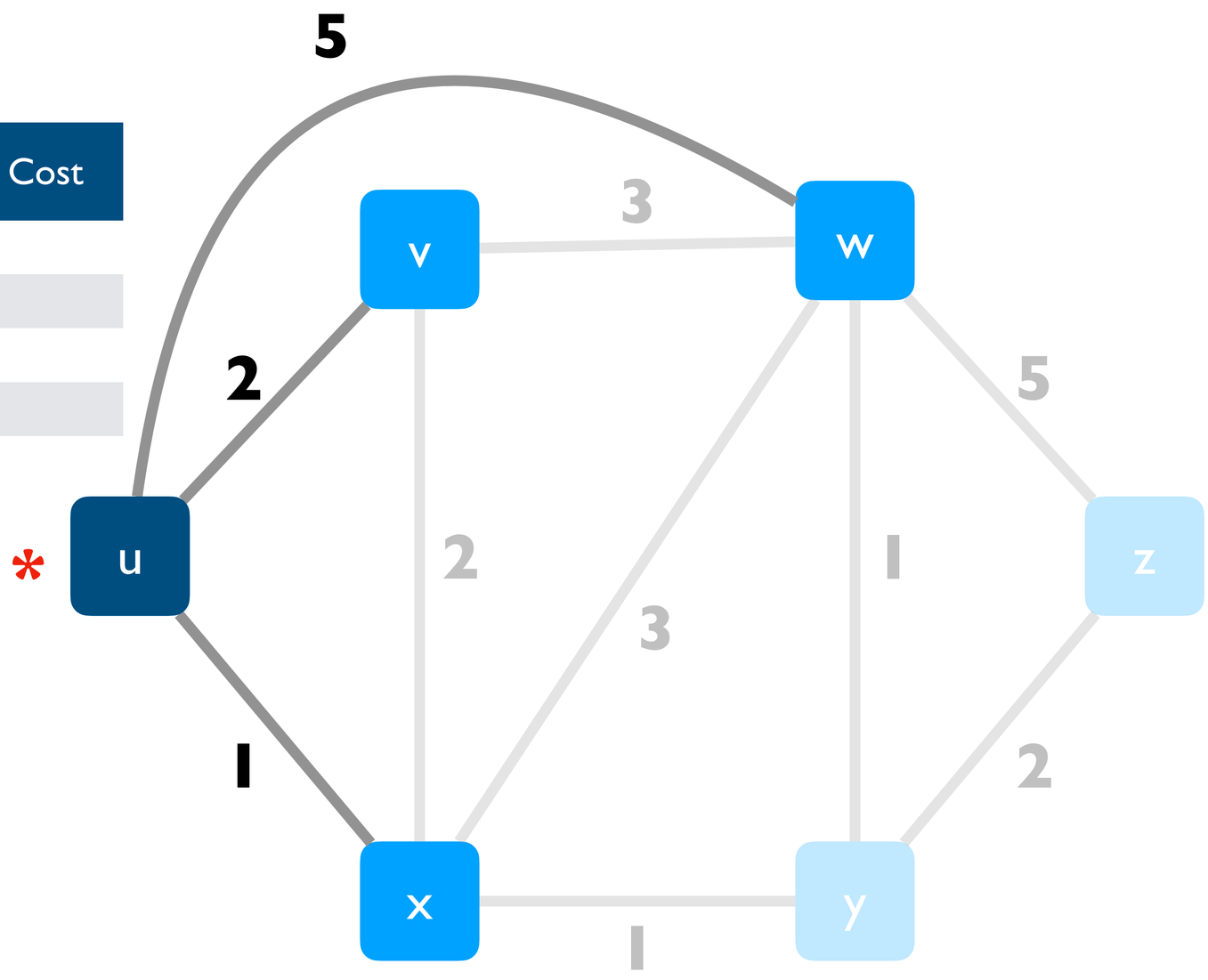
Dest.	Next Hop	Cost
z	~~-~~ ~~v~~ x	~~-~~ ~~4~~ 3
v	v	1
x	x	2



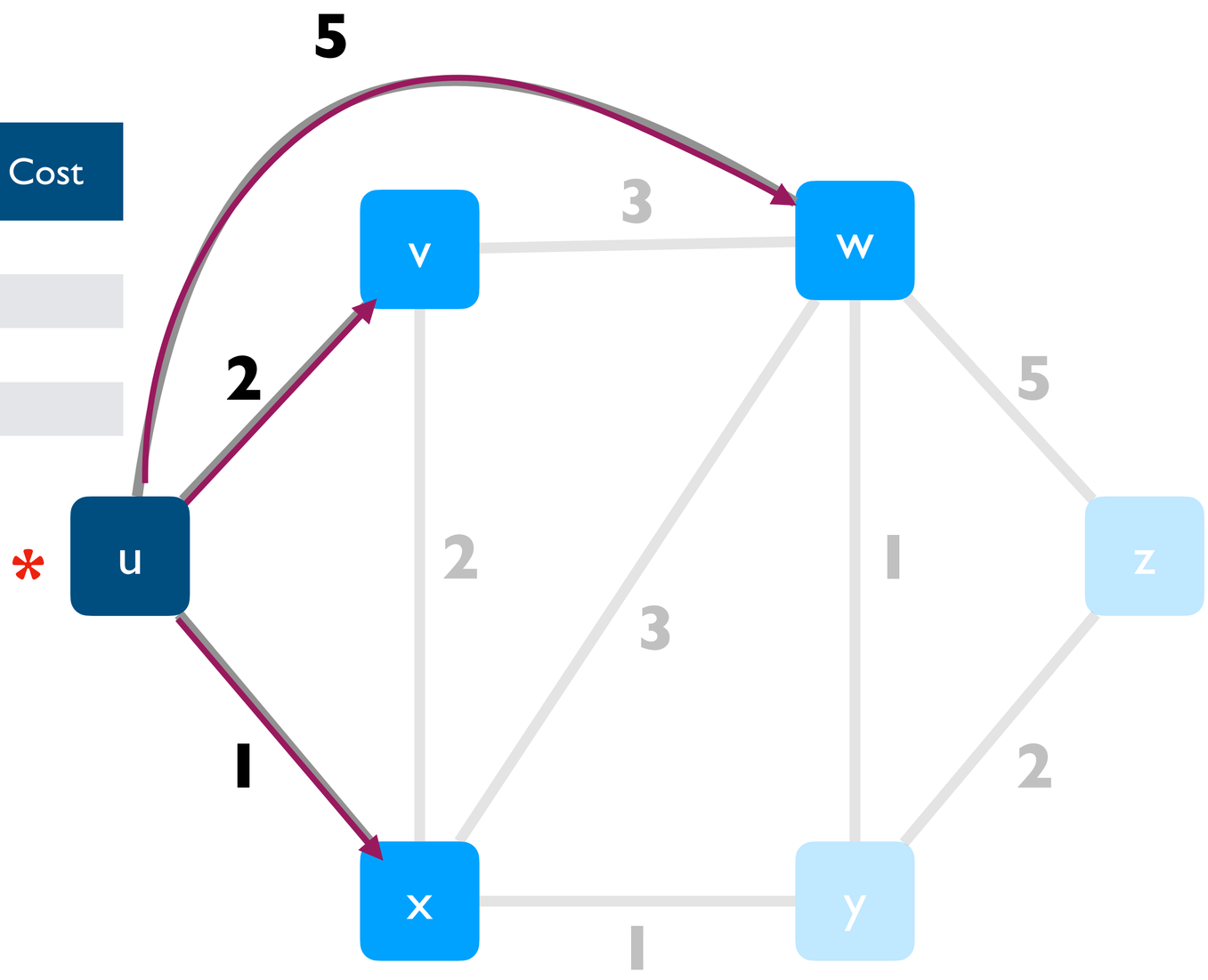
Dest.	Next Hop	Cost
z		
w		
y		
v		
x		



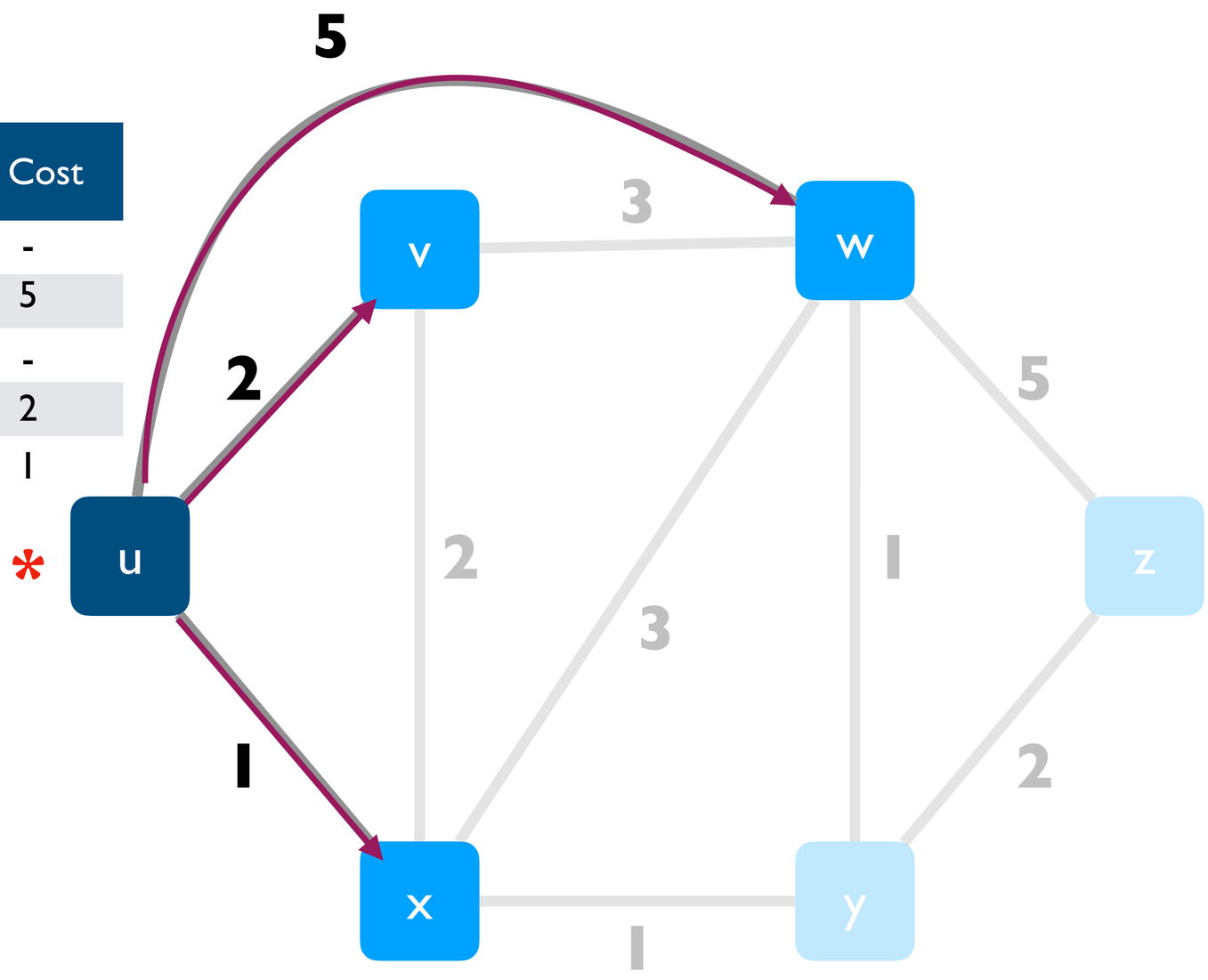
Dest.	Next Hop	Cost
z		
w		
y		
v		
x		



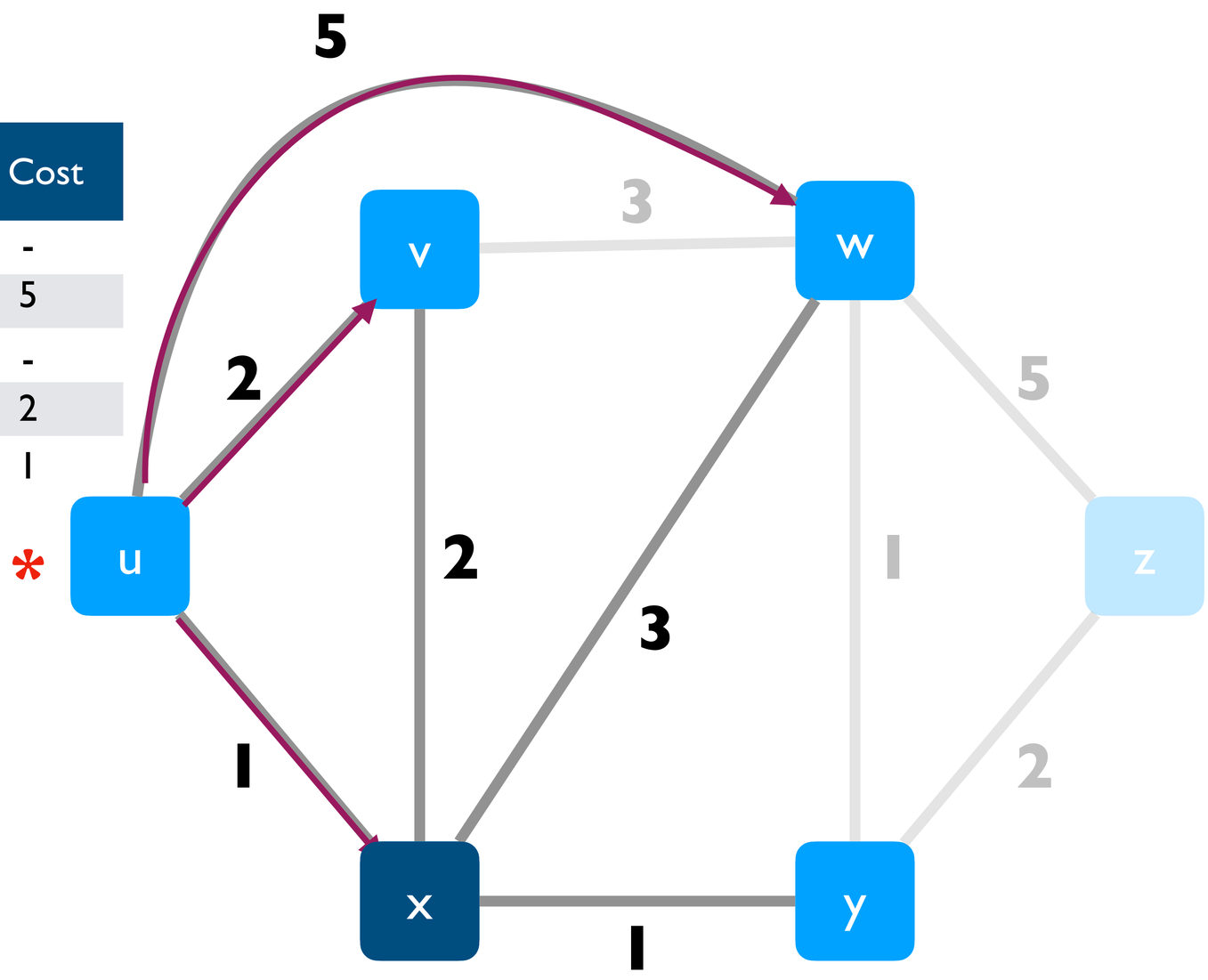
Dest.	Next Hop	Cost
z		
w		
y		
v		
x		



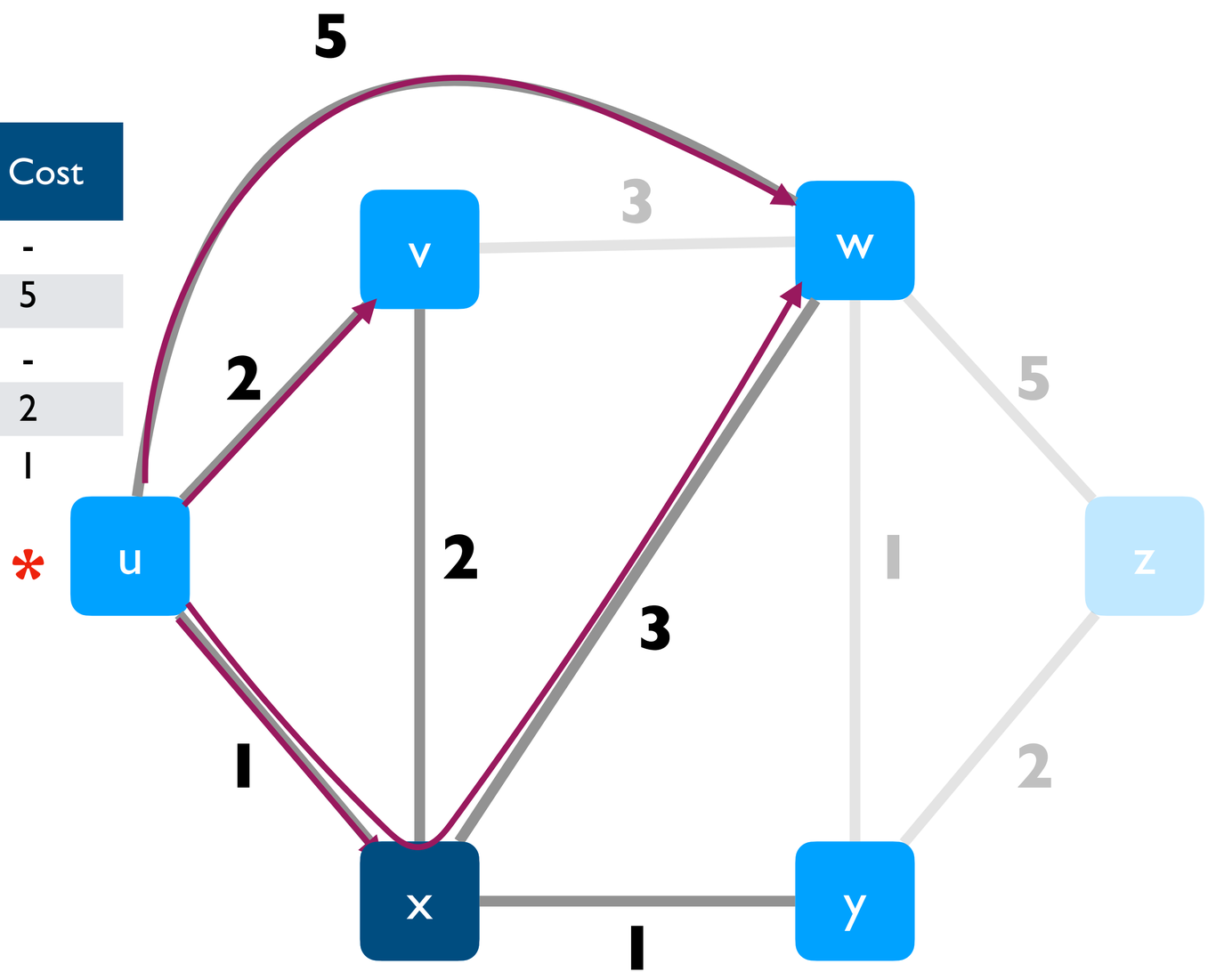
Dest.	Next Hop	Cost
z	-	-
w	w	5
y	-	-
v	v	2
x	x	1



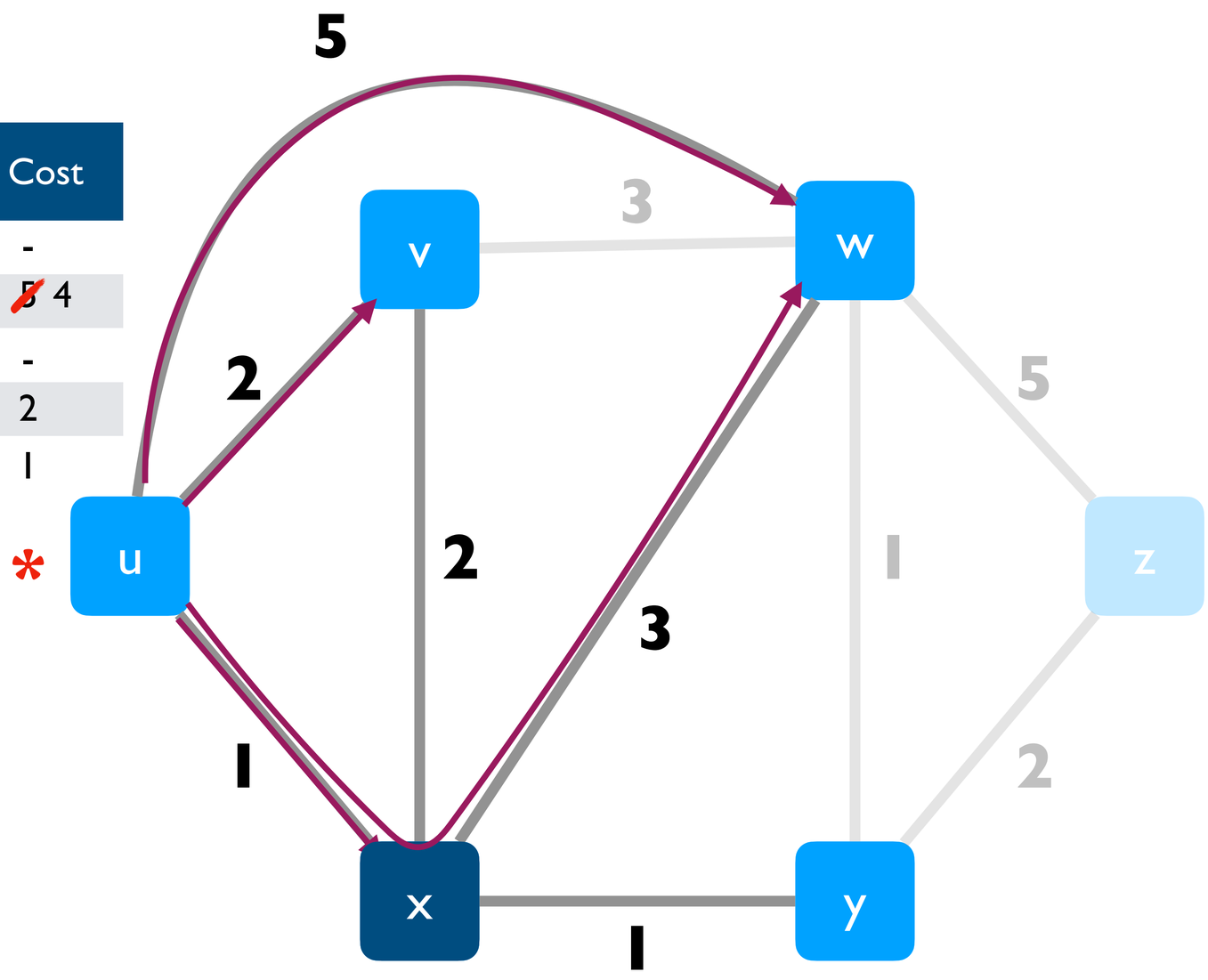
Dest.	Next Hop	Cost
z	-	-
w	w	5
y	-	-
v	v	2
x	x	1



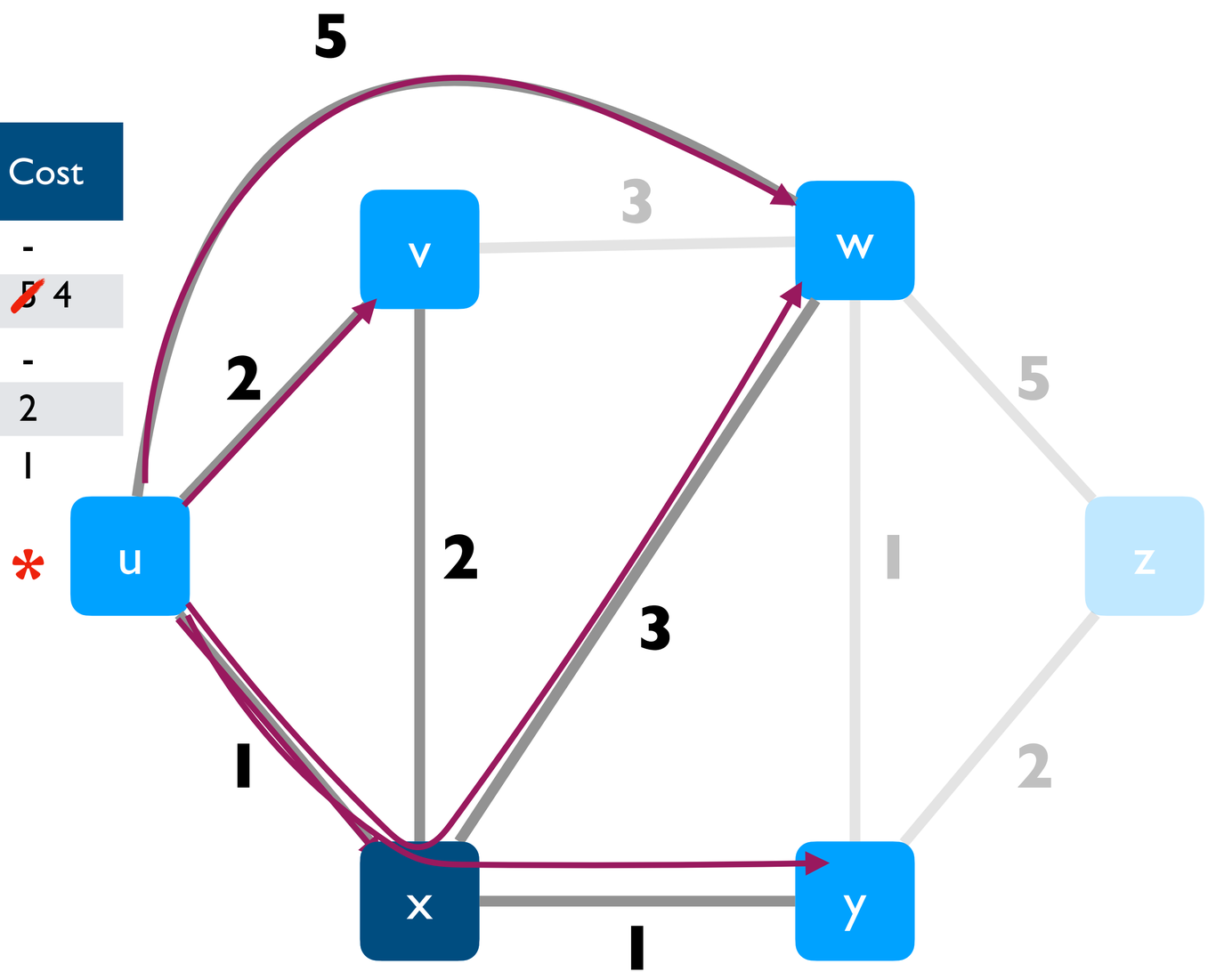
Dest.	Next Hop	Cost
z	-	-
w	w	5
y	-	-
v	v	2
x	x	1



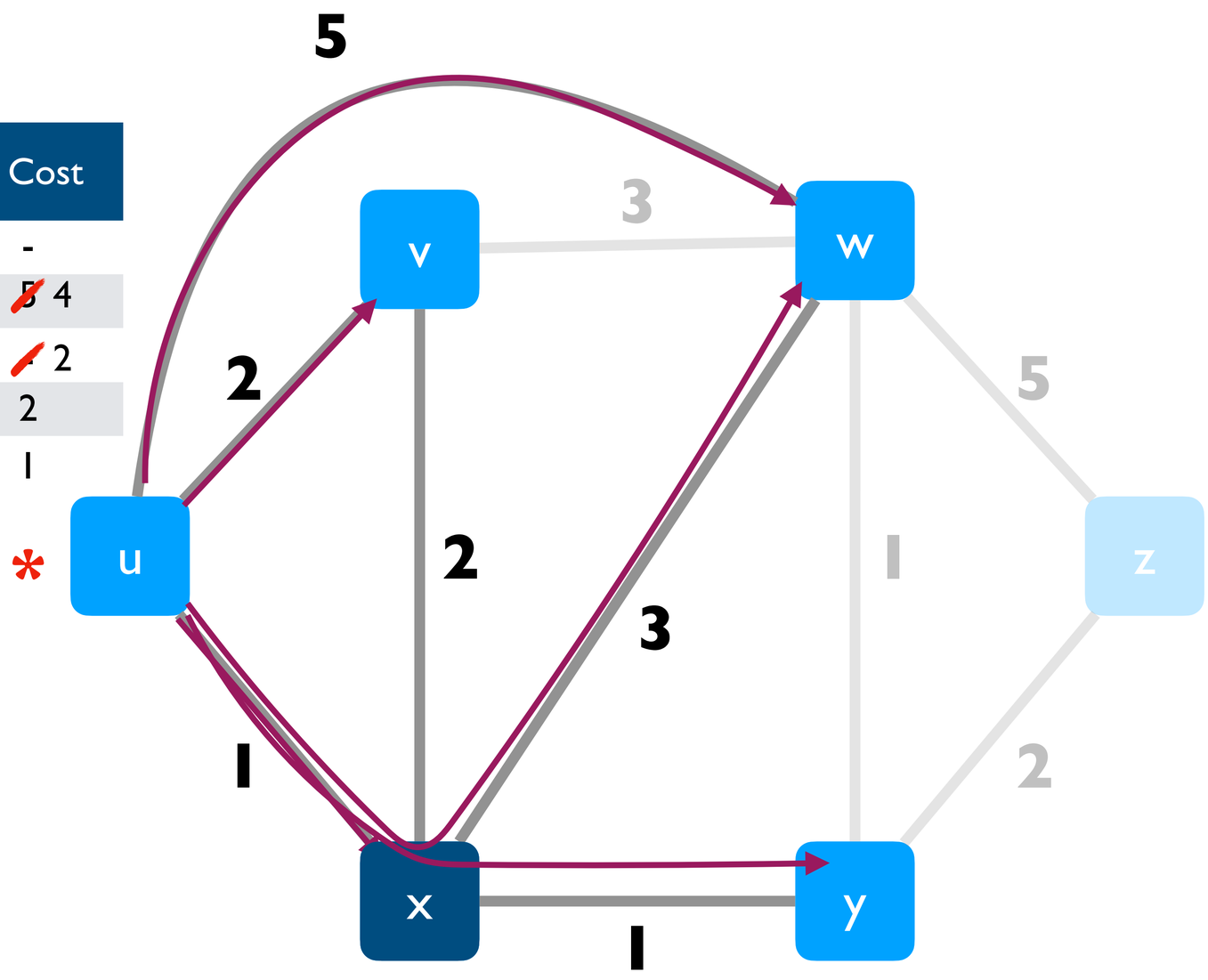
Dest.	Next Hop	Cost
z	-	-
w	~~w~~ x	~~5~~ 4
y	-	-
v	v	2
x	x	1



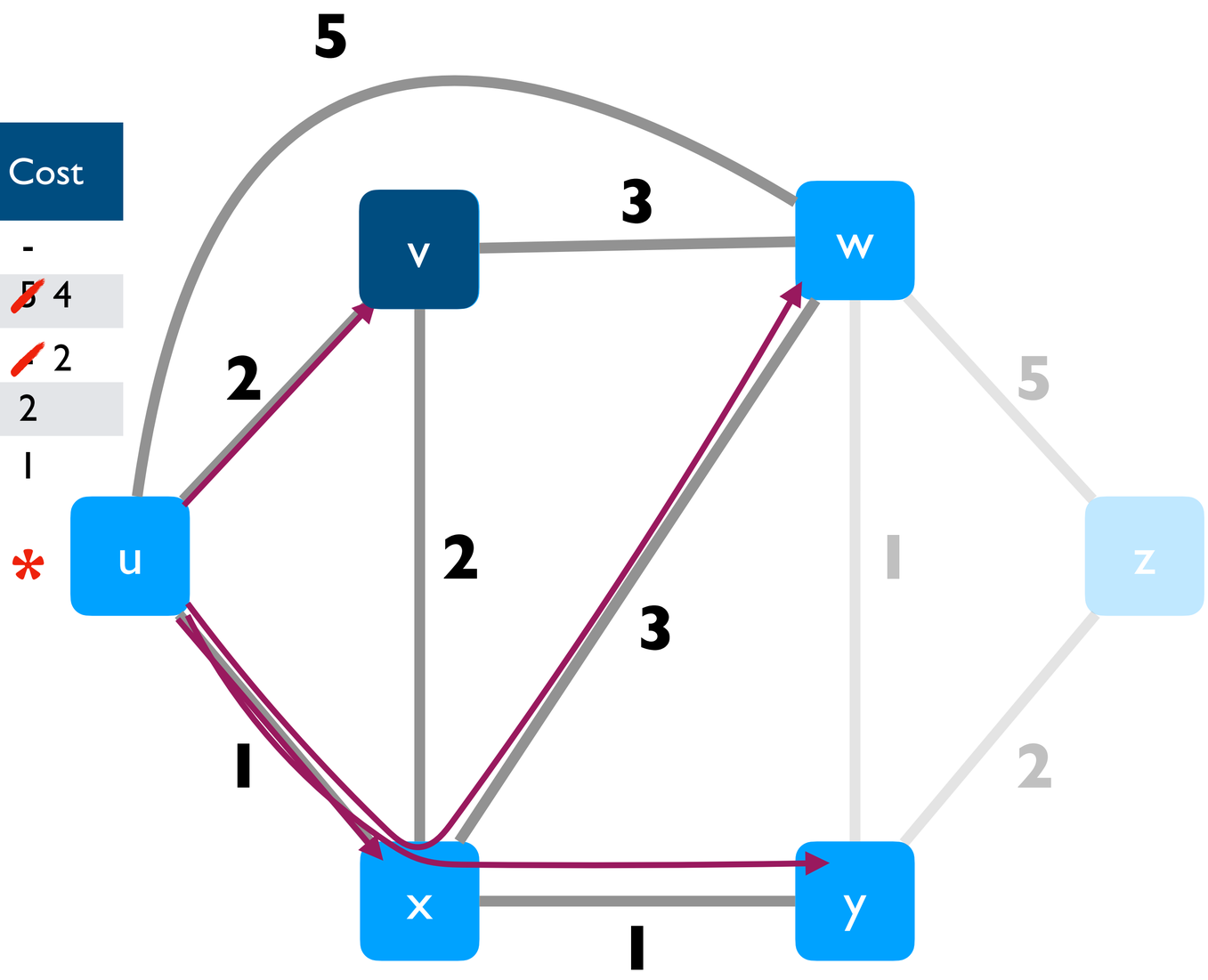
Dest.	Next Hop	Cost
z	-	-
w	~~w~~ x	~~5~~ 4
y	-	-
v	v	2
x	x	1



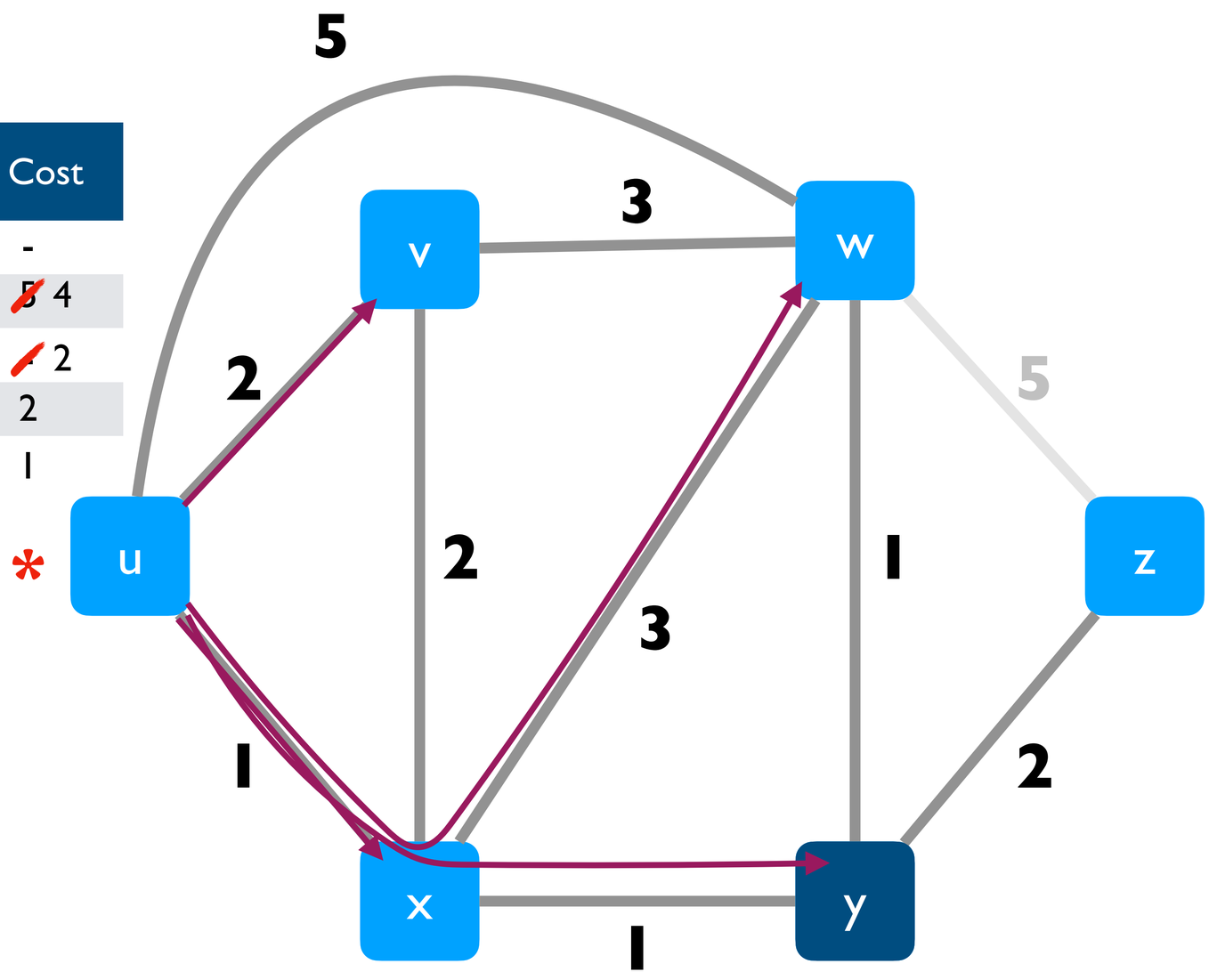
Dest.	Next Hop	Cost
z	-	-
w	~~w~~ x	~~5~~ 4
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



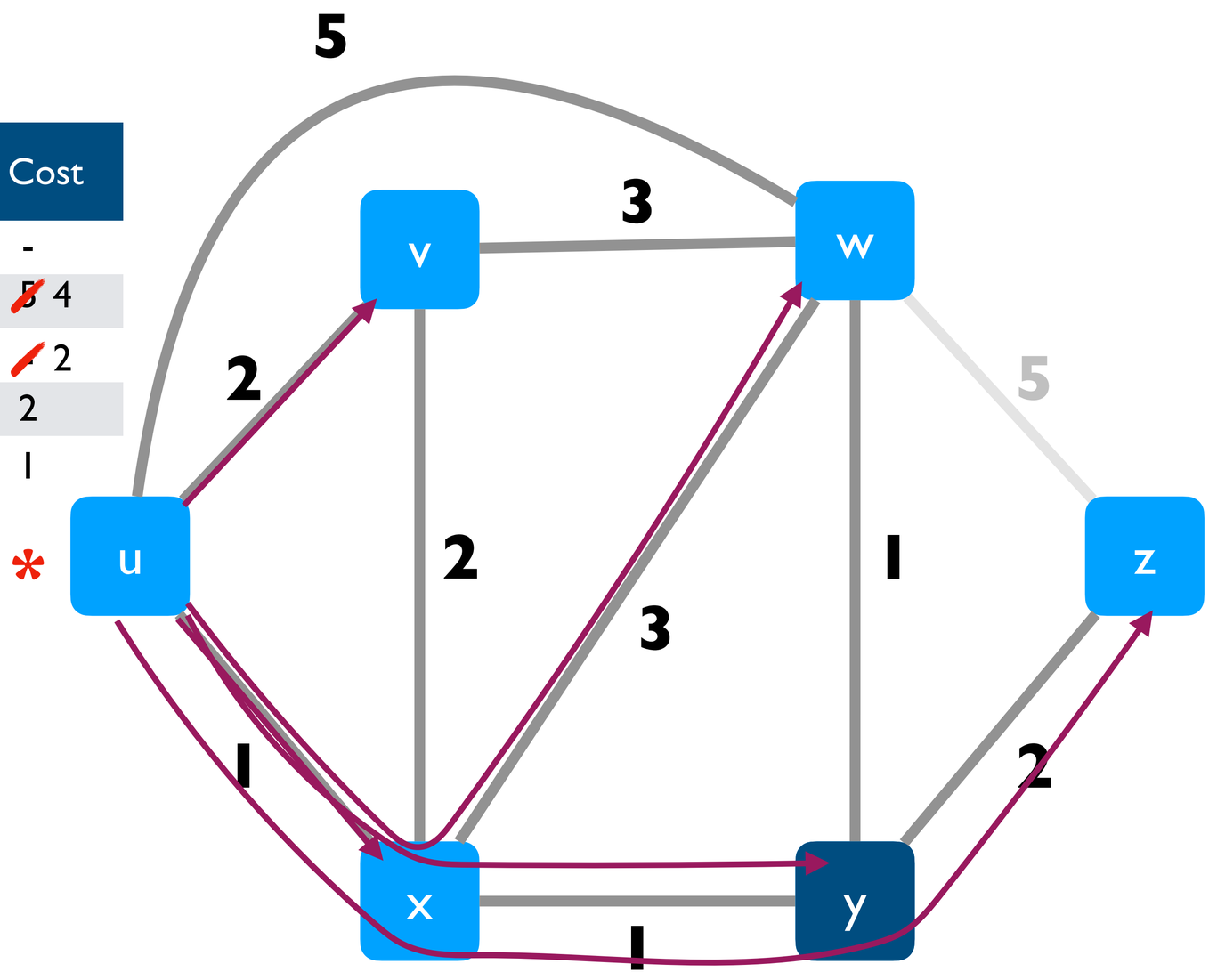
Dest.	Next Hop	Cost
z	-	-
w	~~w~~ x	~~5~~ 4
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



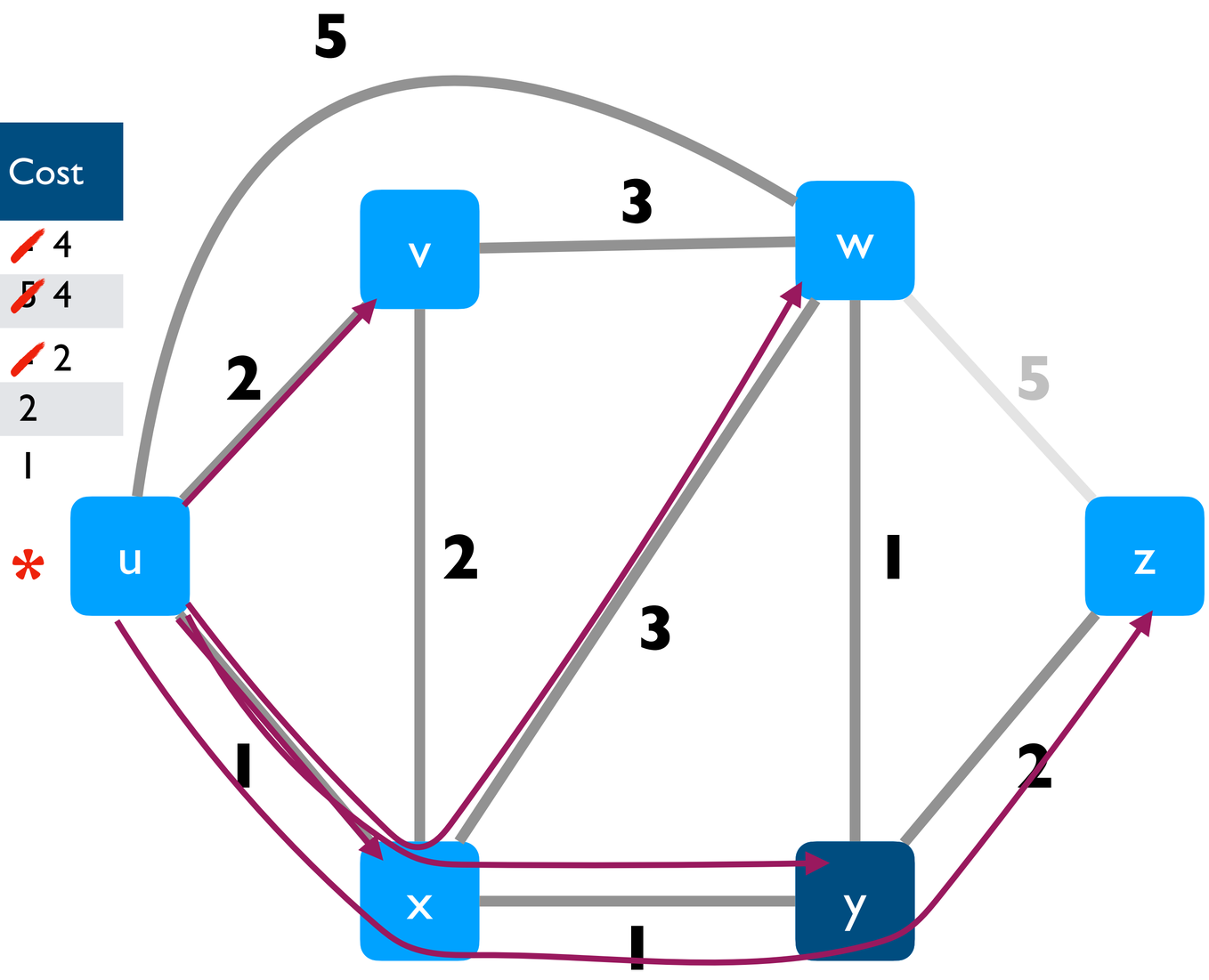
Dest.	Next Hop	Cost
z	-	-
w	~~w~~ x	~~5~~ 4
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



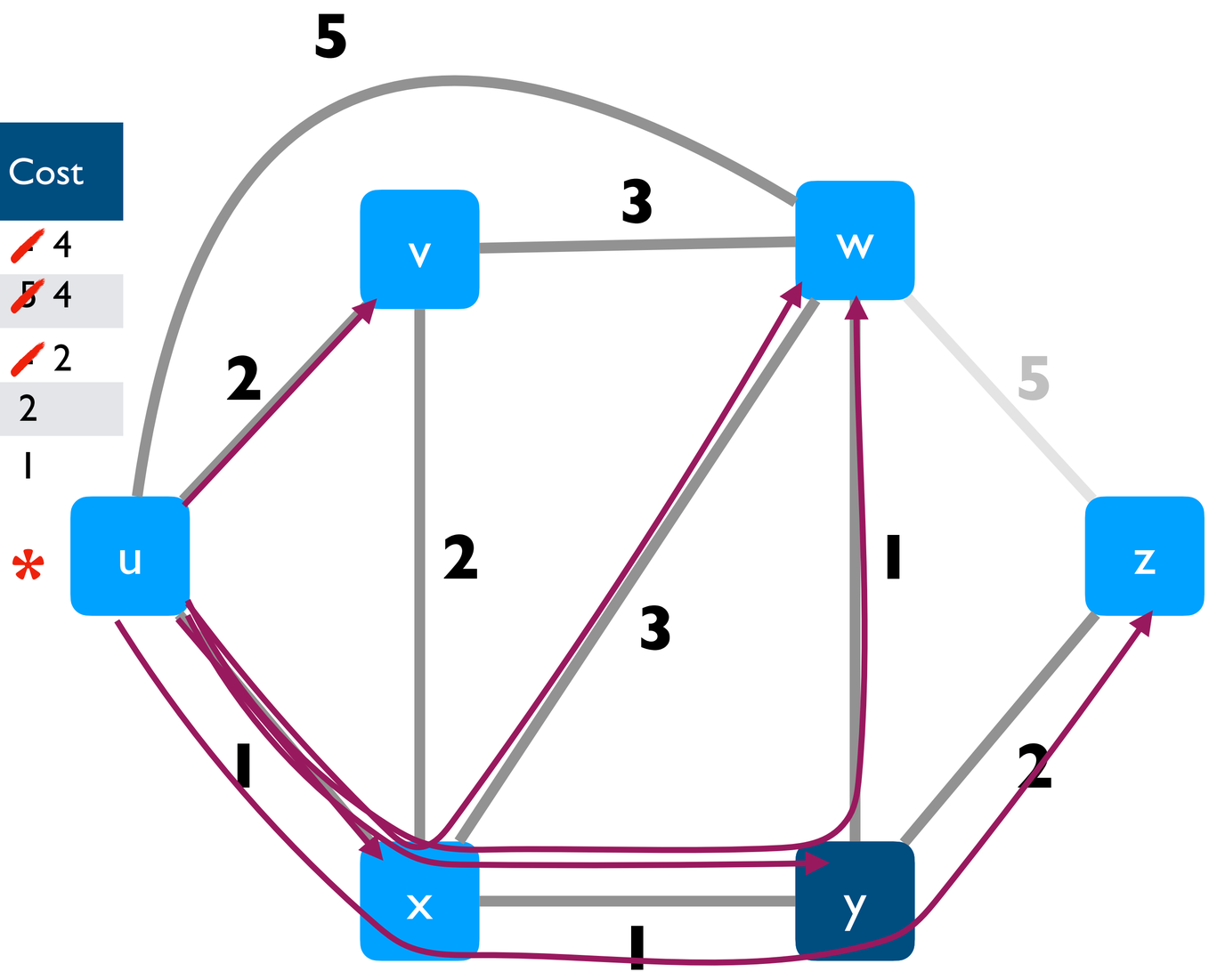
Dest.	Next Hop	Cost
z	-	-
w	~~w~~ x	~~5~~ 4
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



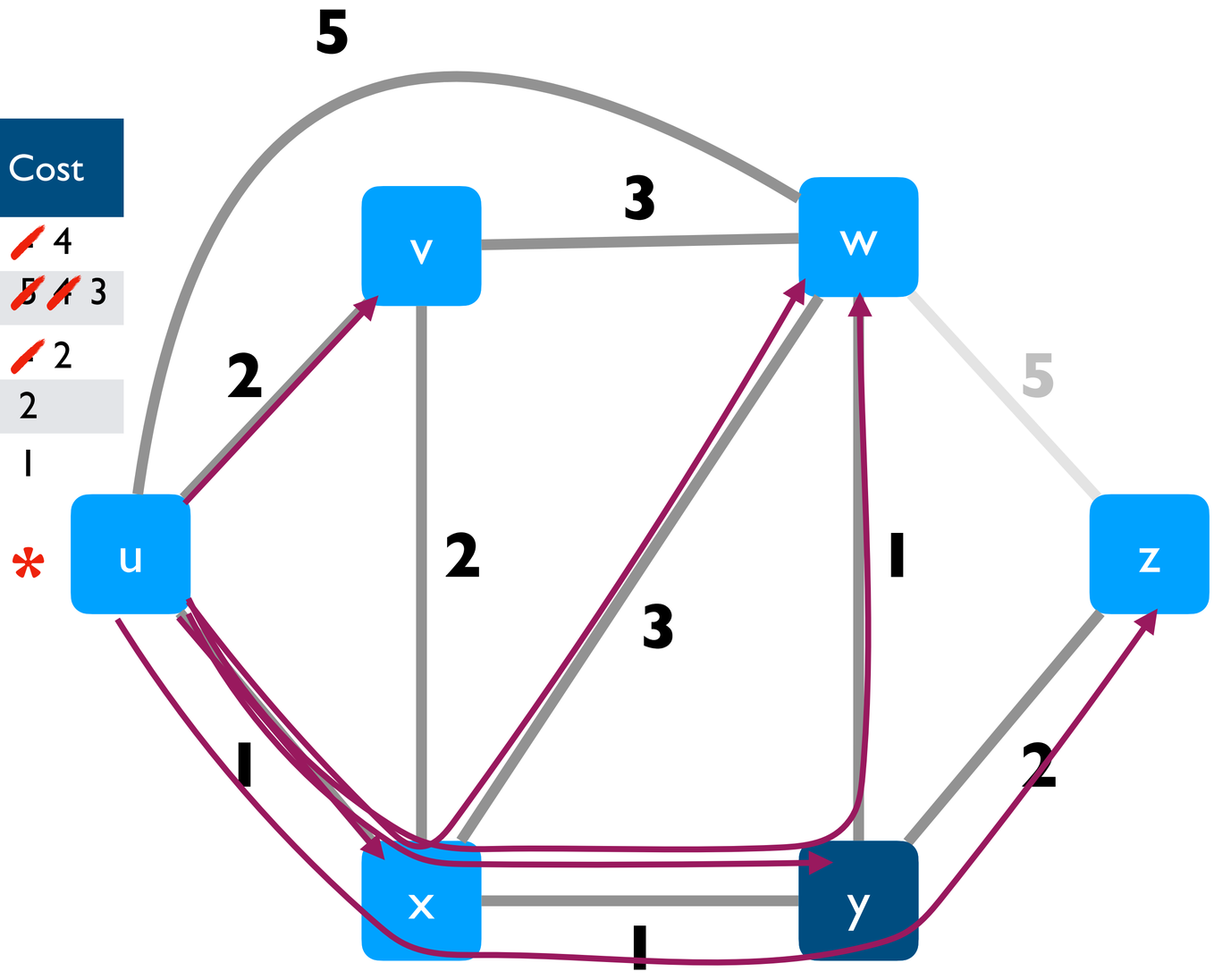
Dest.	Next Hop	Cost
z	~~-~~ x	~~-~~ 4
w	~~w~~ x	~~5~~ 4
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



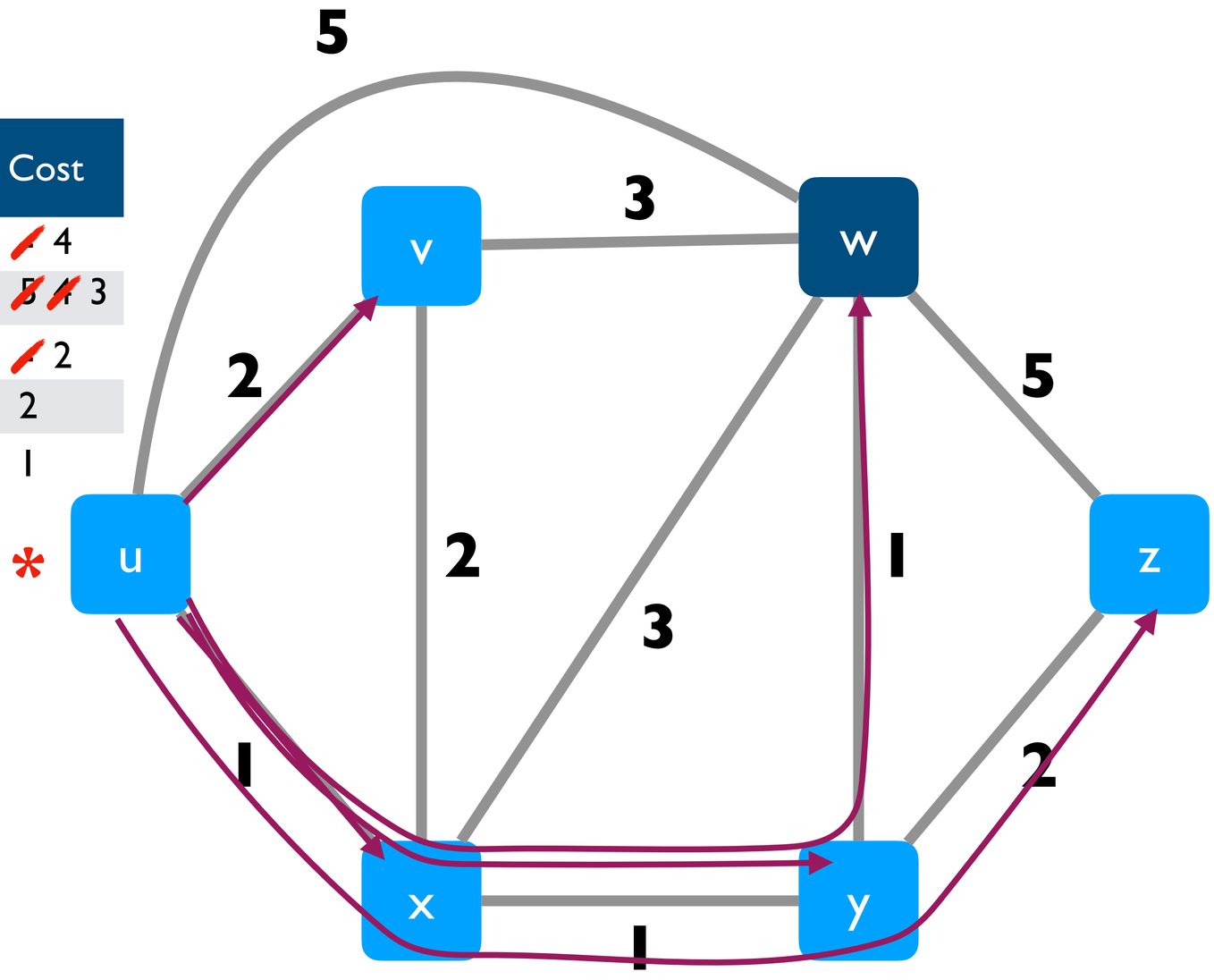
Dest.	Next Hop	Cost
z	~~-~~ x	~~-~~ 4
w	~~w~~ x	~~5~~ 4
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



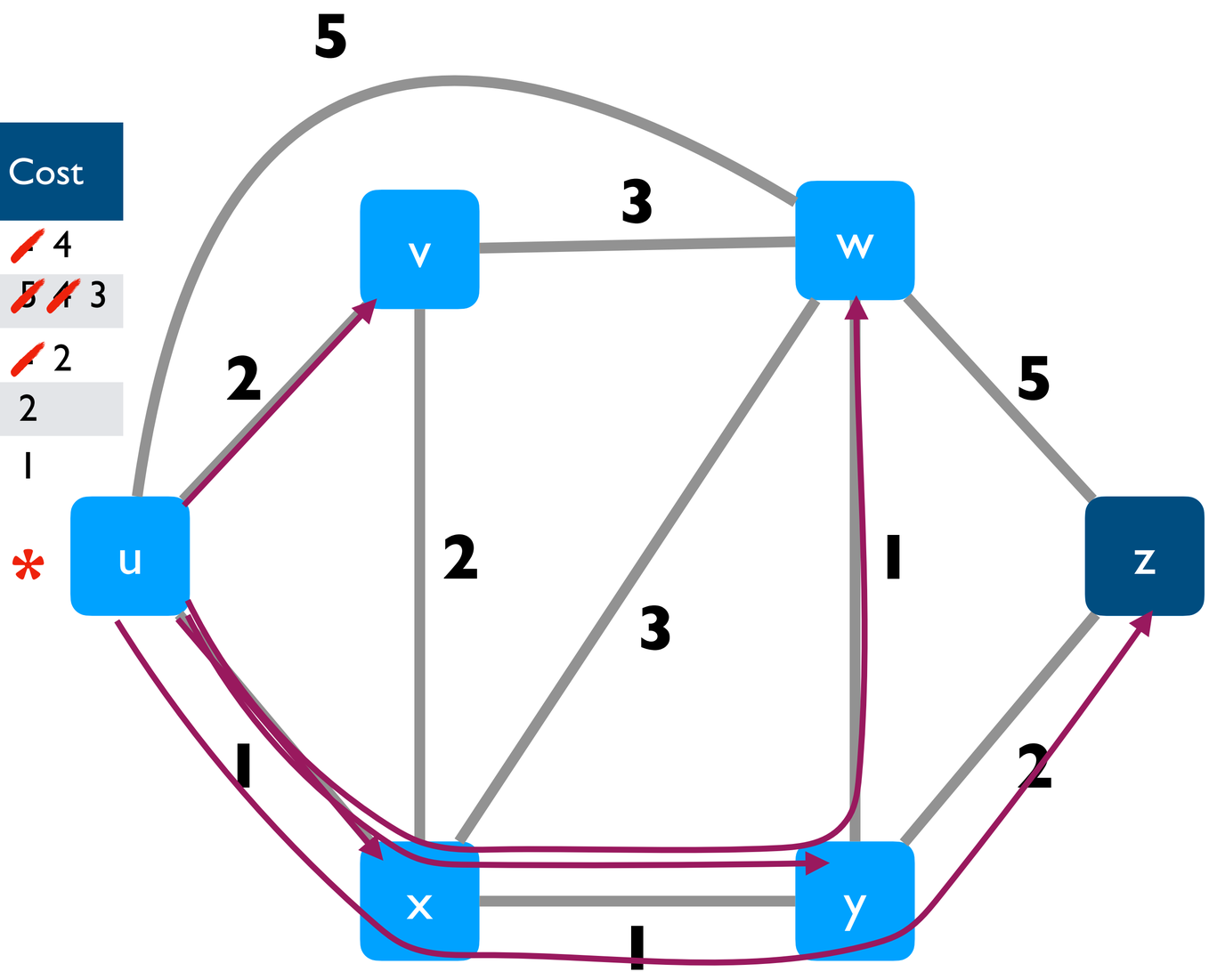
Dest.	Next Hop	Cost
z	~~-~~ x	~~-~~ 4
w	~~w~~ x	~~5~~ ~~4~~ 3
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



Dest.	Next Hop	Cost
z	~~-~~ x	~~-~~ 4
w	~~w~~ x	~~5~~ ~~4~~ 3
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



Dest.	Next Hop	Cost
z	~~-~~ x	~~-~~ 4
w	~~w~~ x	~~5~~ ~~4~~ 3
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



From Routing Algorithm to Protocol

From Routing Algorithm to Protocol

- Note: Dijkstra's is a <u>local</u> computation!
  - *Computed by one node given complete network graph*

From Routing Algorithm to Protocol

- Note: Dijkstra's is a <u>local</u> computation!
  - *Computed by one node given complete network graph*
- Possibilities
  - *Option#1: a separate machine runs the algorithm*
  - *Option#2: every router runs the algorithm*

From Routing Algorithm to Protocol

- Note: Dijkstra's is a <u>local</u> computation!
  - *Computed by one node given complete network graph*
- Possibilities
  - *Option#1: a separate machine runs the algorithm*
  - *Option#2: every router runs the algorithm*
- The protocol variant used in the IP layer uses Option#2

From Routing Algorithm to Protocol

- Note: Dijkstra's is a <u>local</u> computation!
  - *Computed by one node given complete network graph*
- Possibilities
  - *Option#1: a separate machine runs the algorithm*
  - *Option#2: every router runs the algorithm*
- The protocol variant used in the IP layer uses Option#2
- We will talk about Option#1 when we talk about Software Defined Networks (SDNs)

Link State Routing

Link State Routing

- Every router knows its local "link state"
  - *Router u: "(u, v) with cost = 2; (u, x) with cost = 1"*

Link State Routing

- Every router knows its local "link state"
  - *Router u: "(u, v) with cost = 2; (u, x) with cost = 1"*
- A router floods its link state to all other routers
  - *Does so periodically or when its link state changes*

Link State Routing

- Every router knows its local "link state"
  - *Router u: "(u, v) with cost = 2; (u, x) with cost = 1"*
- A router floods its link state to all other routers
  - *Does so periodically or when its link state changes*
- Hence, every router learns the entire network graph
  - *Runs Disjkstra's locally to compute its forwarding table*

Link State Routing

- Every router knows its local "link state"
  - *Router u: "(u, v) with cost = 2; (u, x) with cost = 1"*
- A router floods its link state to all other routers
  - *Does so periodically or when its link state changes*
- Hence, every router learns the entire network graph
  - *Runs Disjkstra's locally to compute its forwarding table*
- OSPF is a link-state protocol (IETF RFC 2328 or 5340)

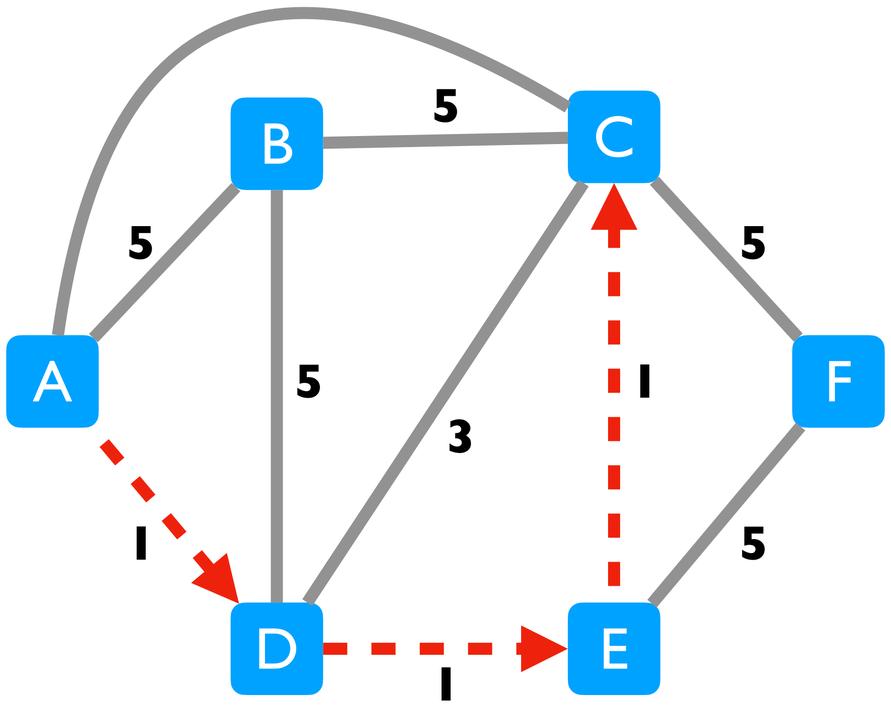
Link State Routing

Link State Routing

Are loops possible?

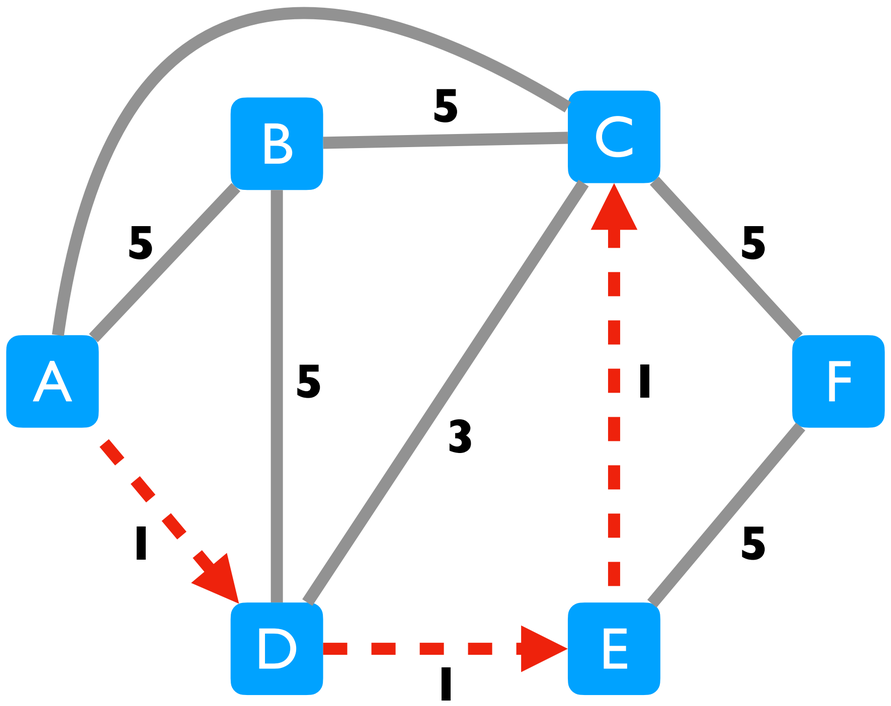
Link State Routing

Are loops possible?



Link State Routing

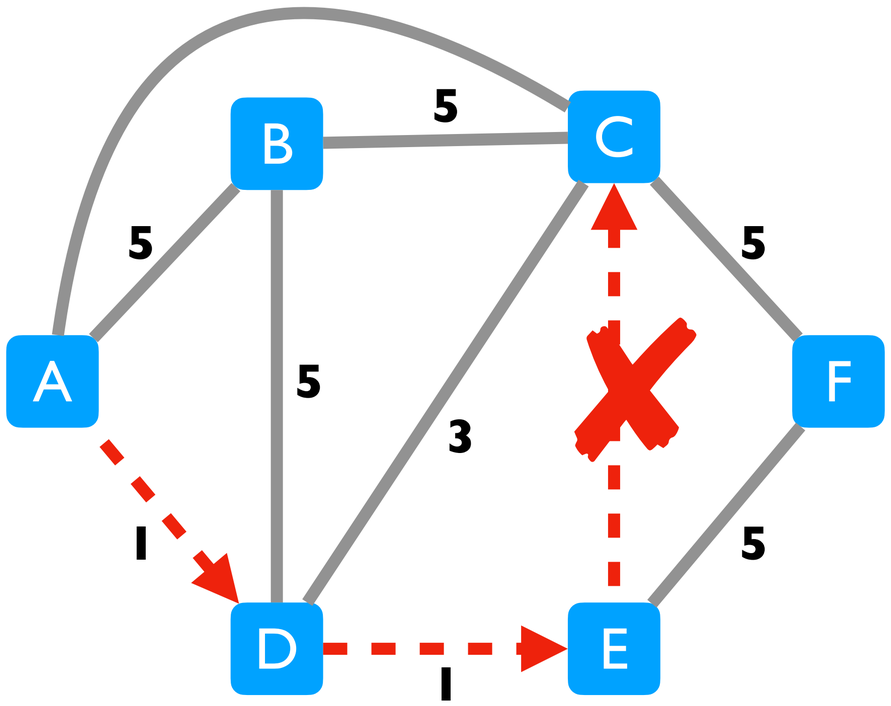
Are loops possible?



A and D think this is the path to C

Link State Routing

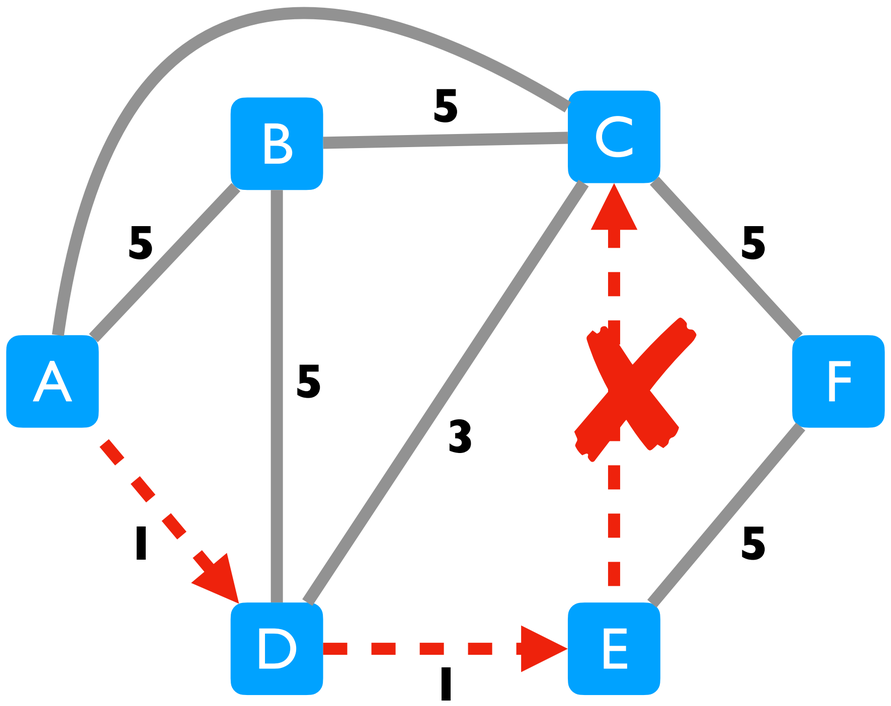
Are loops possible?

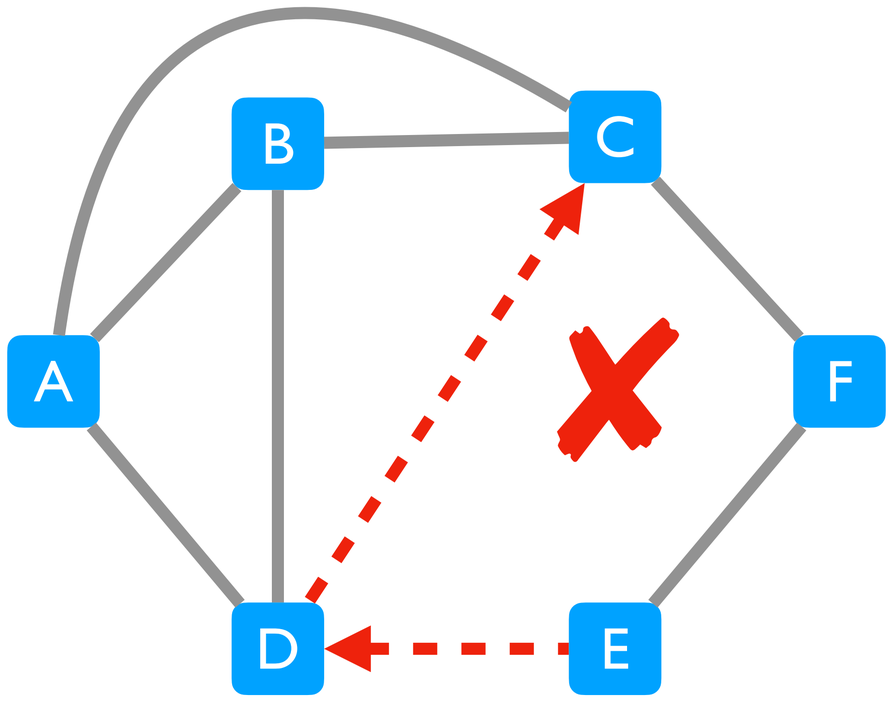


A and D think this is the path to C

Link State Routing

Are loops possible?

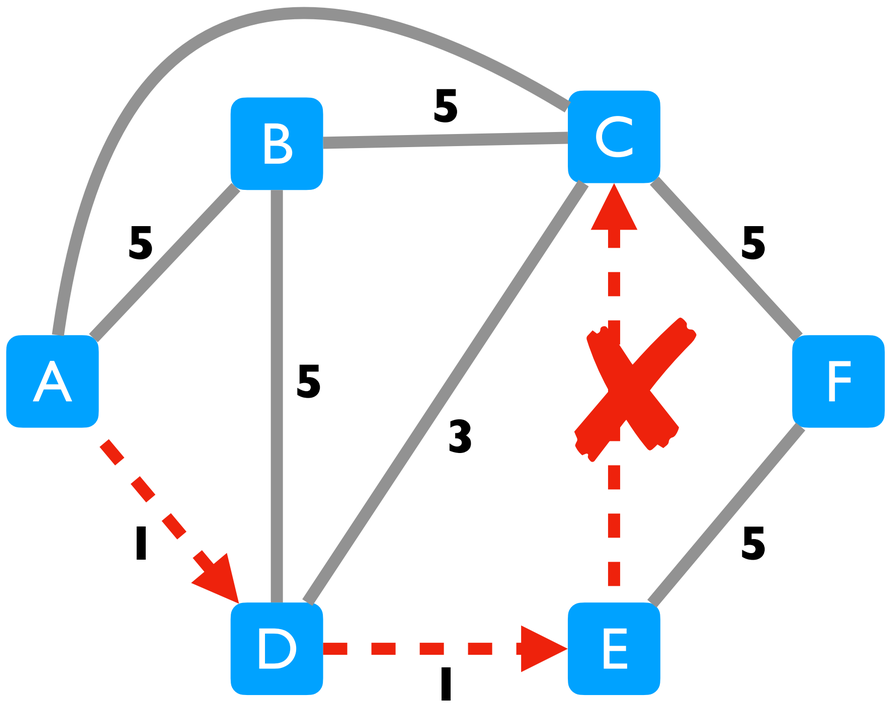




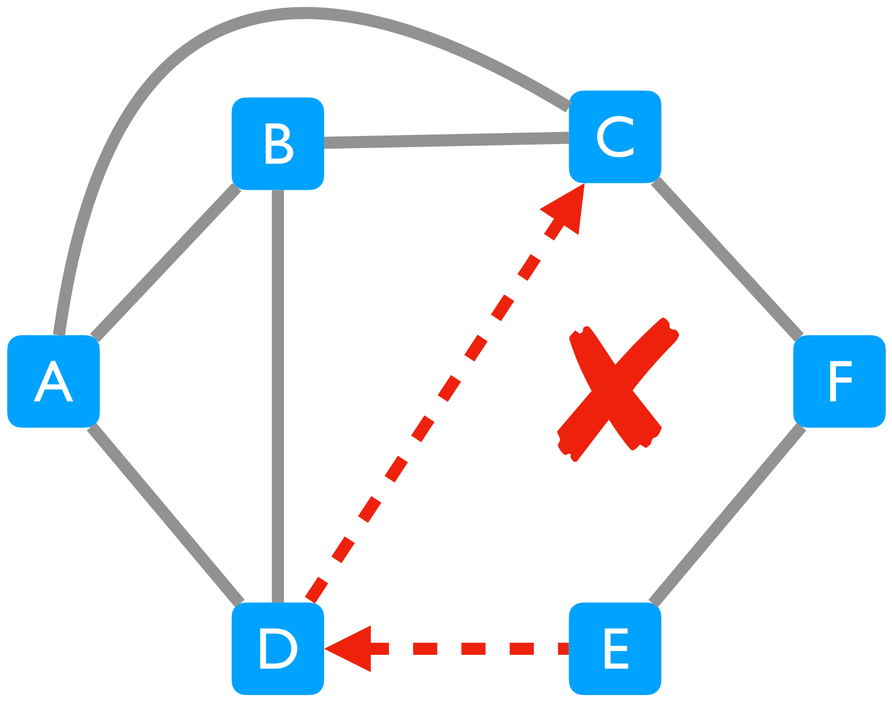
A and D think this is the path to C

Link State Routing

Are loops possible?



A and D think this is the path to C



E thinks that this is the path to C

Convergence

Convergence

- All routers have consistent routing information
  - *E.g., all nodes having the same link-state database*

Convergence

- All routers have consistent routing information
  - *E.g., all nodes having the same link-state database*
- Forwarding is consistent after convergence
  - *All nodes have the same link-state database*
  - *All nodes forward packets on the same paths*

Convergence Delay

Convergence Delay

- Time to achieve convergence

Convergence Delay

- Time to achieve convergence
- Sources of convergence delay?

Convergence Delay

- Time to achieve convergence
- Sources of convergence delay?
  - *Time to detect failure*

Convergence Delay

- Time to achieve convergence
- Sources of convergence delay?
  - *Time to detect failure*
  - *Time to flood link-state information*

Convergence Delay

- Time to achieve convergence
- Sources of convergence delay?
  - *Time to detect failure*
  - *Time to flood link-state information*
  - *Time to recompute forwarding tables*

Convergence Delay

- Time to achieve convergence
- Sources of convergence delay?
  - *Time to detect failure*
  - *Time to flood link-state information*
  - *Time to recompute forwarding tables*
- Performance during convergence period?

Convergence Delay

- Time to achieve convergence
- Sources of convergence delay?
  - *Time to detect failure*
  - *Time to flood link-state information*
  - *Time to recompute forwarding tables*
- Performance during convergence period?
  - *Lost packets due to "black-holes"*

Convergence Delay

- Time to achieve convergence
- Sources of convergence delay?
  - *Time to detect failure*
  - *Time to flood link-state information*
  - *Time to recompute forwarding tables*
- Performance during convergence period?
  - *Lost packets due to "black-holes"*
  - *Looping packets*

Convergence Delay

- Time to achieve convergence
- Sources of convergence delay?
  - *Time to detect failure*
  - *Time to flood link-state information*
  - *Time to recompute forwarding tables*
- Performance during convergence period?
  - *Lost packets due to "black-holes"*
  - *Looping packets*
  - *Out-of-order packets reaching the destination*

Link State Routing

Link State Routing

- Are loops possible?
  - *Yes, until convergence*

Link State Routing

- Are loops possible?
  - *Yes, until convergence*
- Scalability?
  - *O(NE) messages*
  - *O(N^2) computation time*
  - *O(network diameter) convergence time*
  - *O(N) entries in forwarding table*

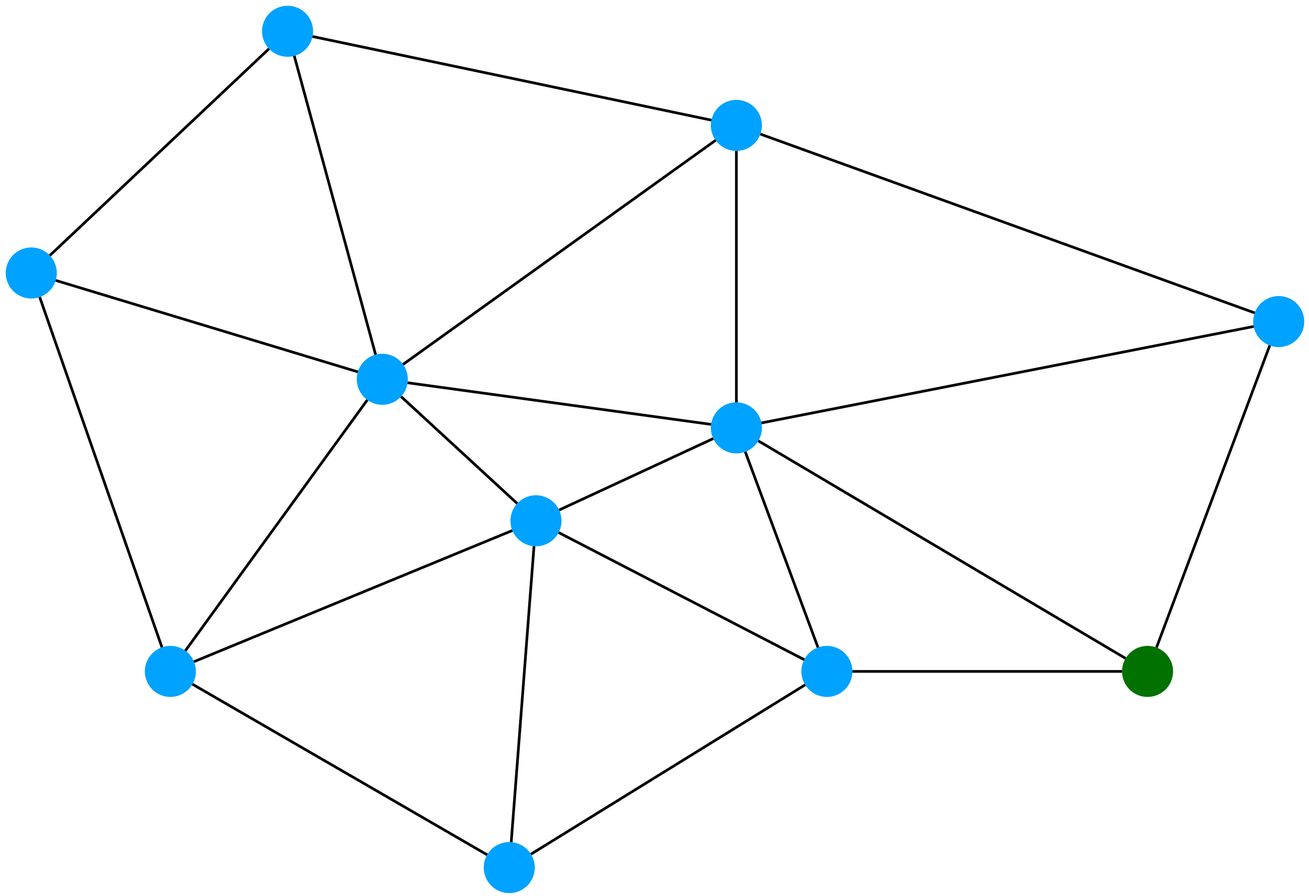
Link State Routing

- Are loops possible?
  - *Yes, until convergence*
- Scalability?
  - *O(NE) messages*
  - *O(N^2) computation time*
  - *O(network diameter) convergence time*
  - *O(N) entries in forwarding table*
- Do we have to use Dijkstra's?

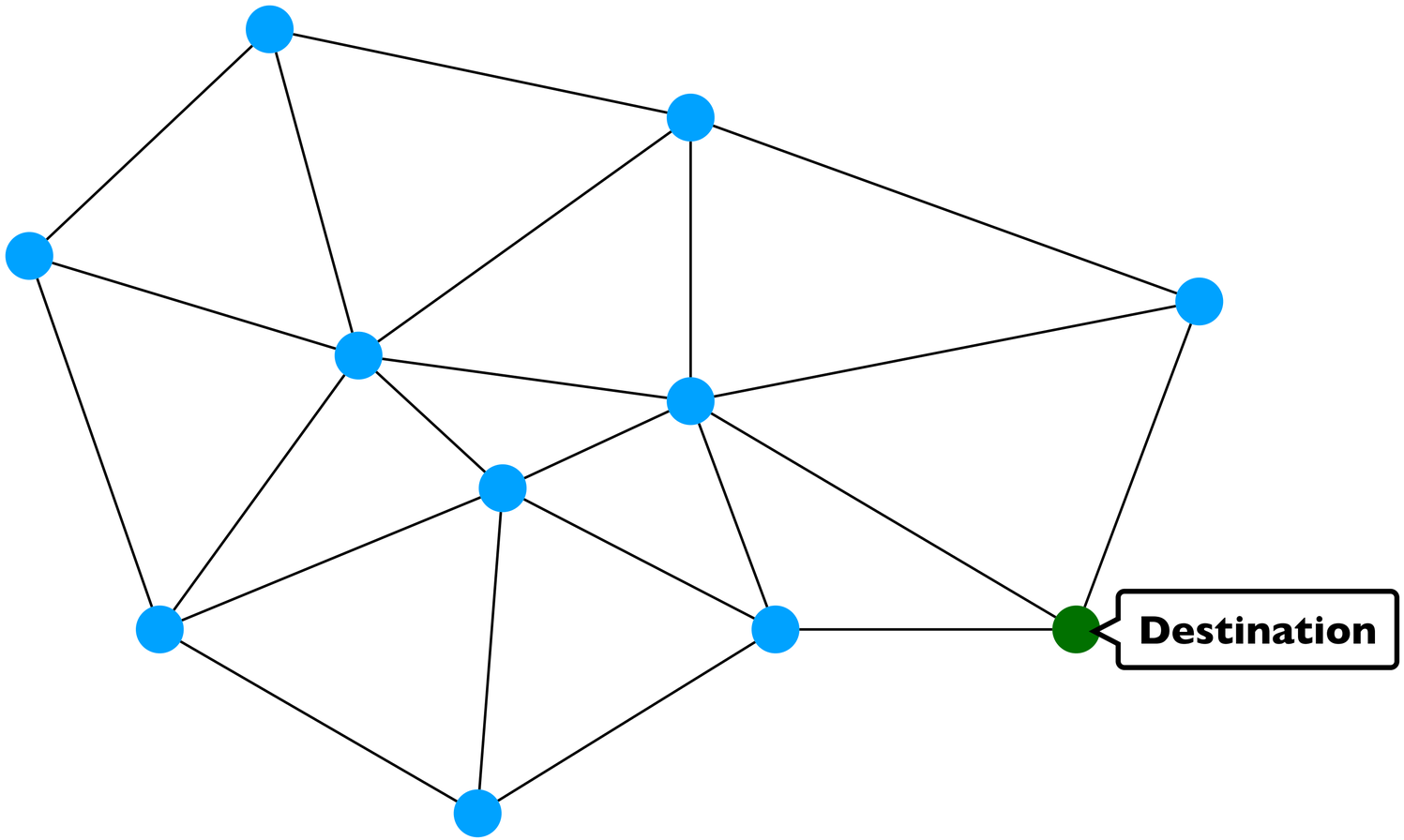
Outline

- Approach 1: link-state routing
- Approach 2: distance-vector routing
- Routing in the internet

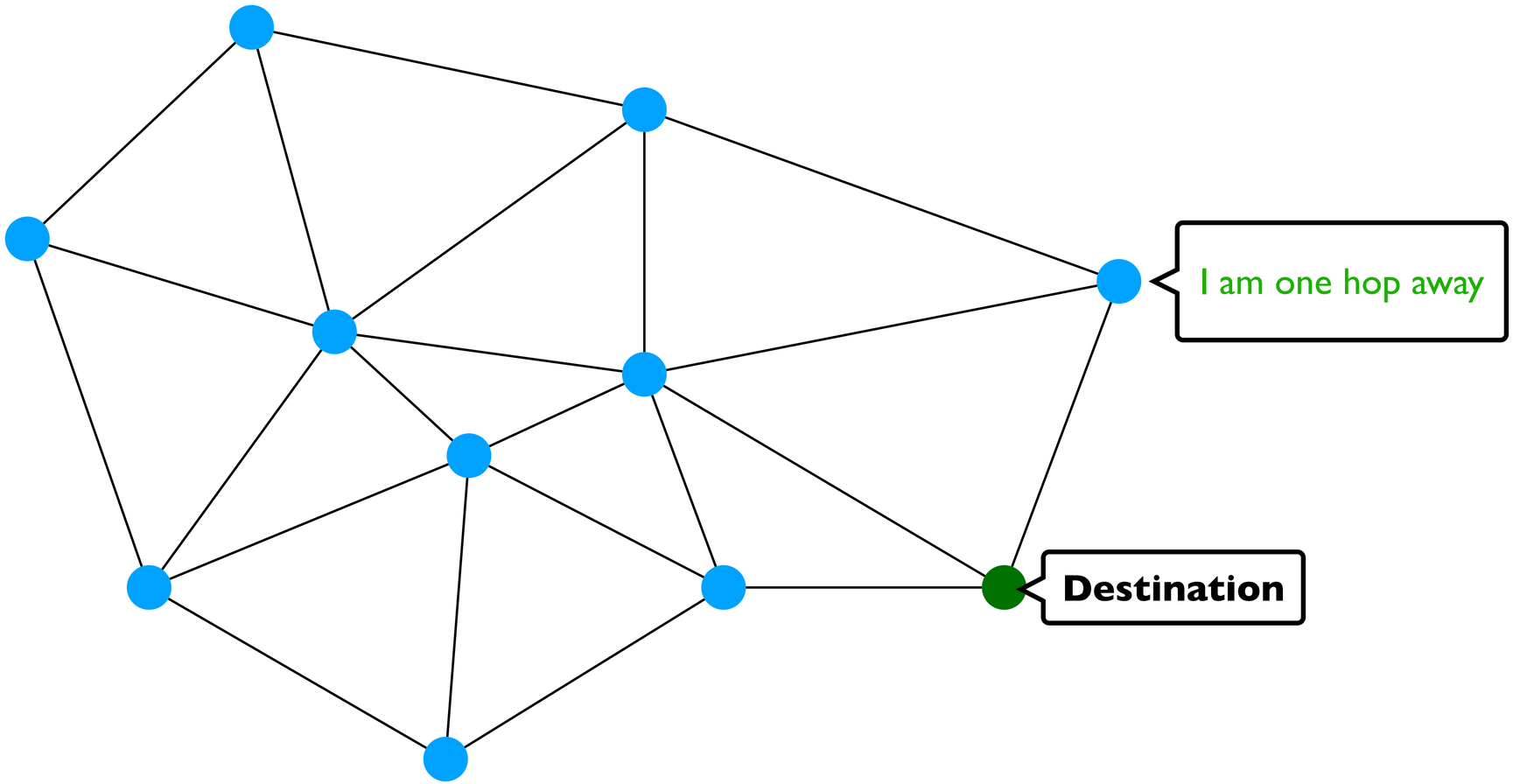
Distance Vector Routing: An example of ***Distributed*** Computation



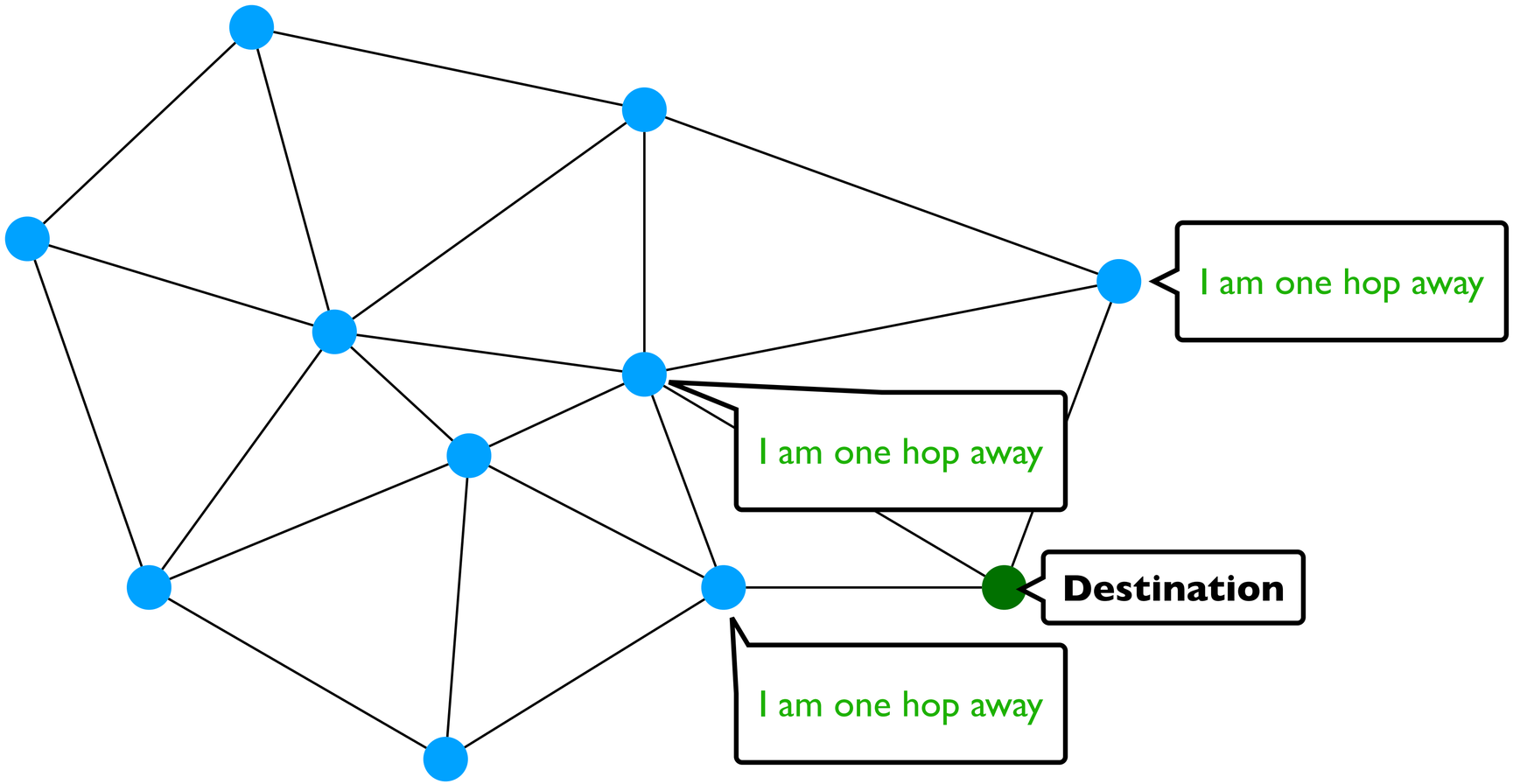
Distance Vector Routing: An example of ***Distributed*** Computation



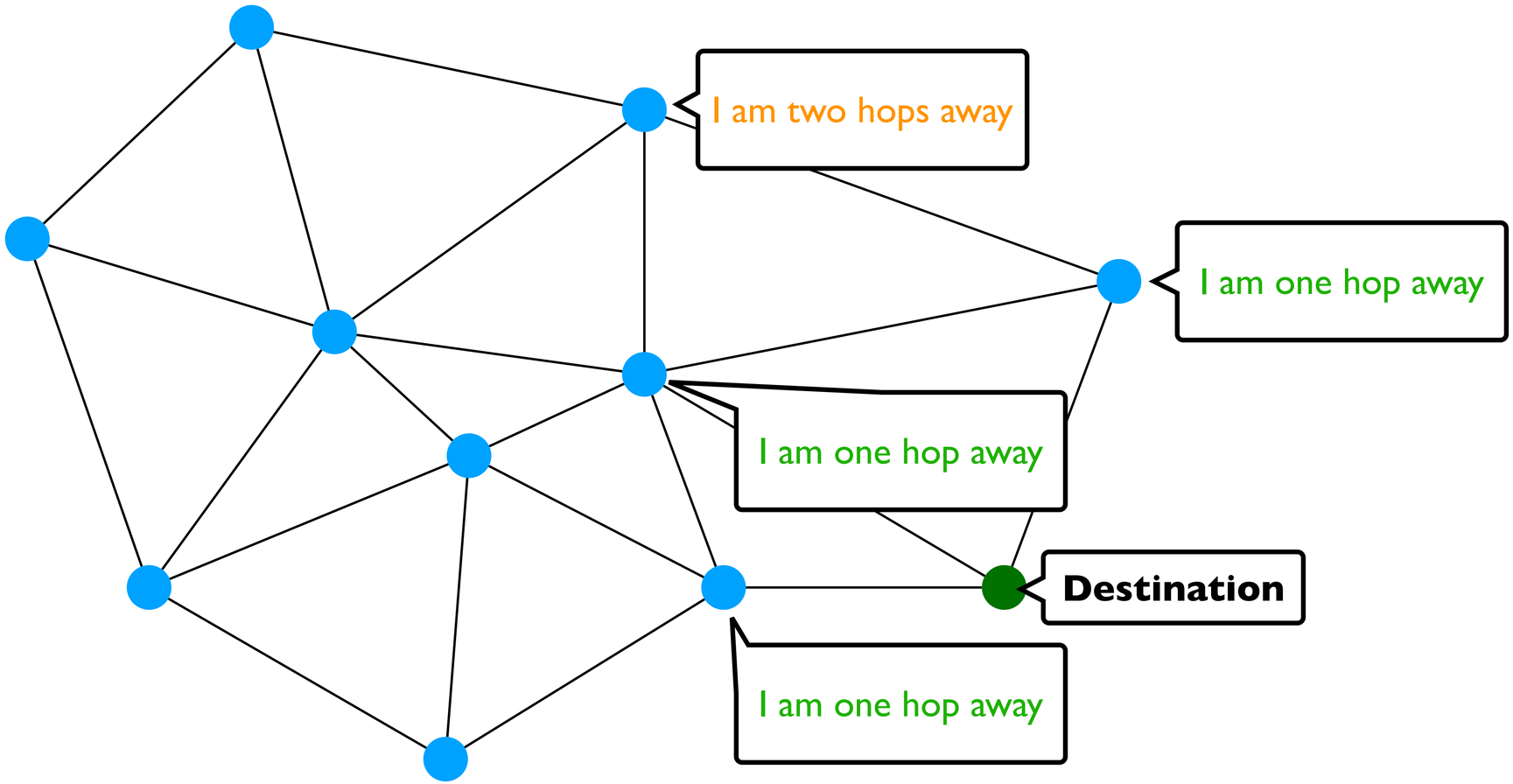
Distance Vector Routing: An example of ***Distributed*** Computation



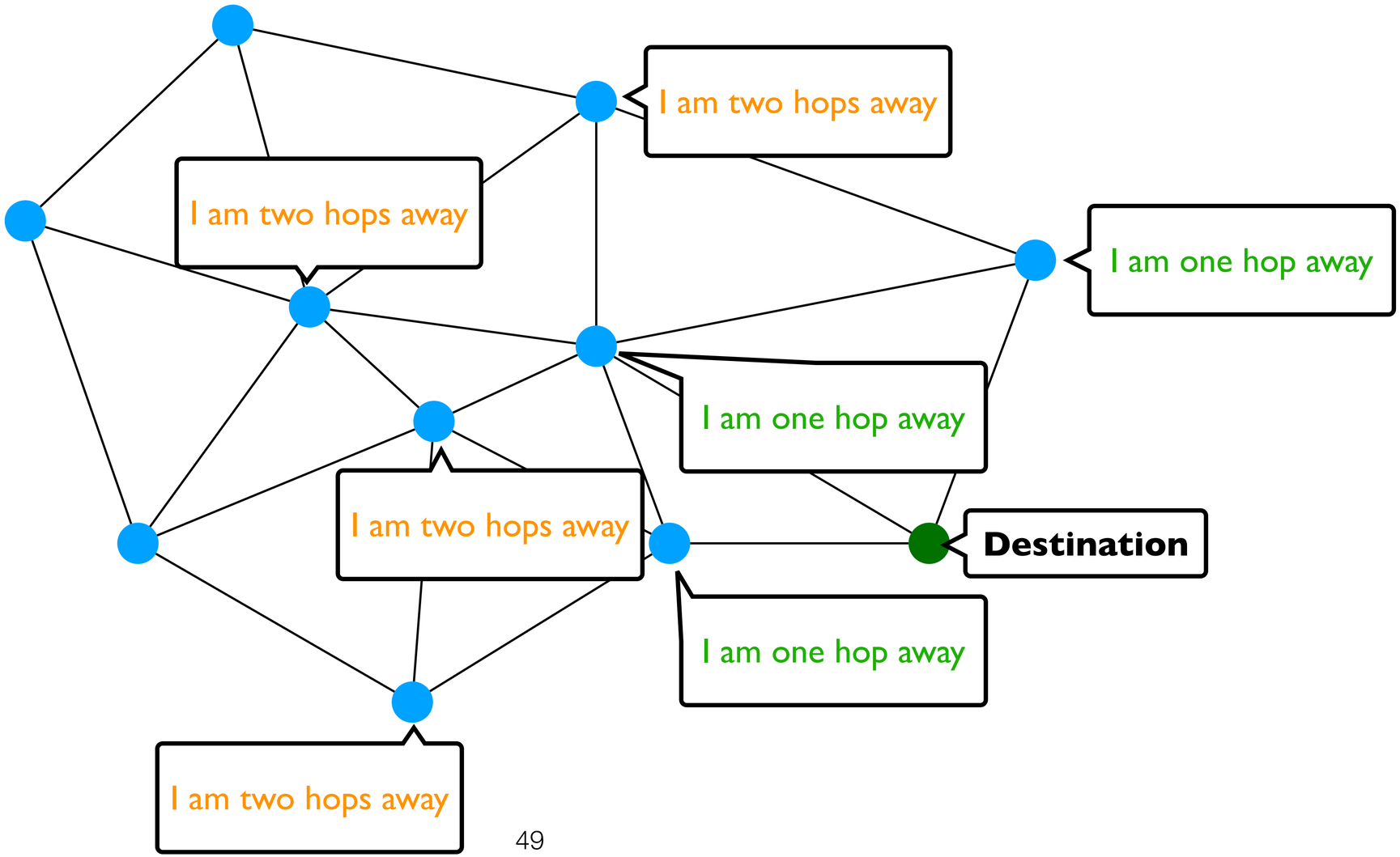
Distance Vector Routing: An example of ***Distributed*** Computation



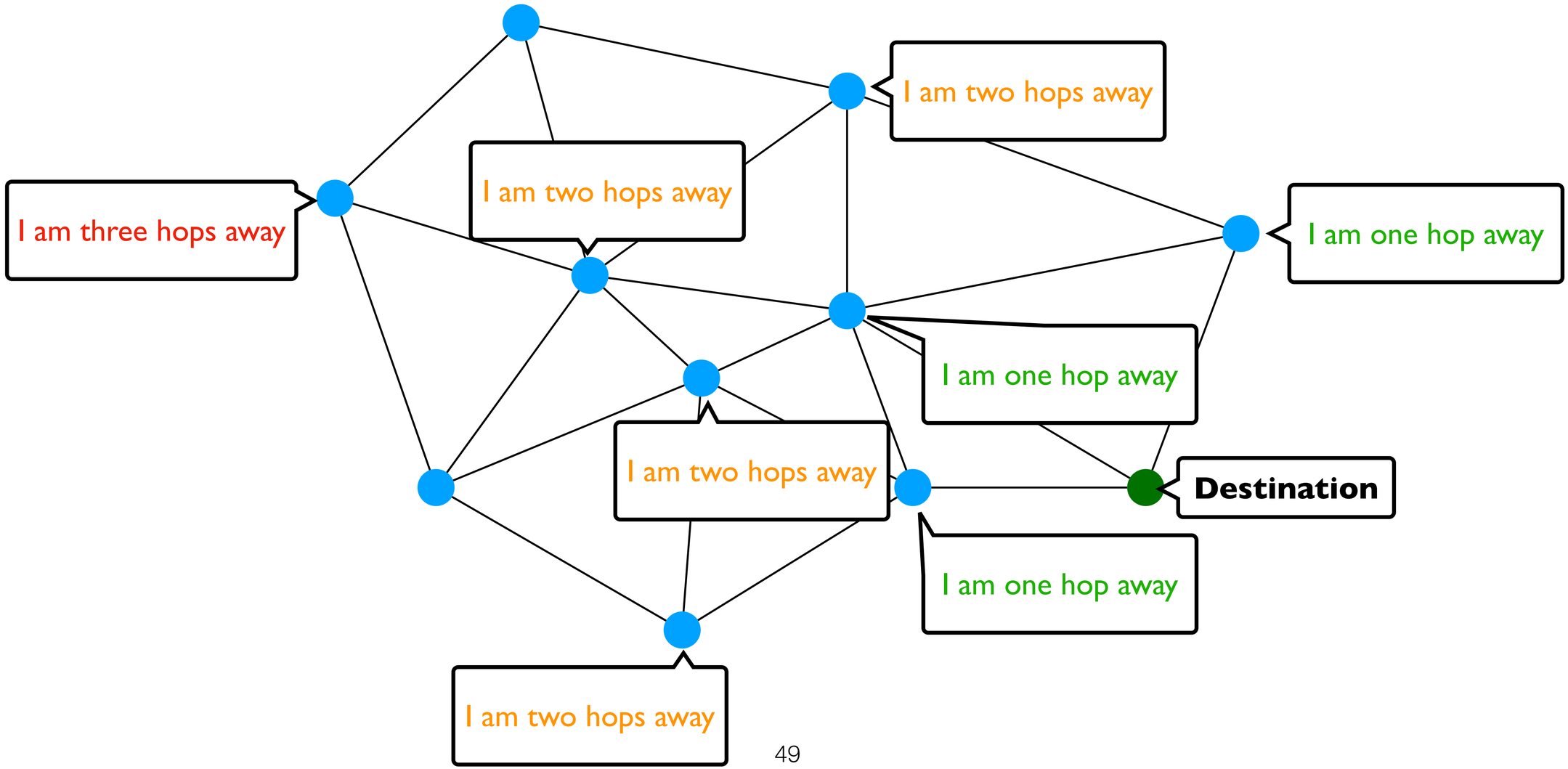
Distance Vector Routing: An example of ***Distributed*** Computation



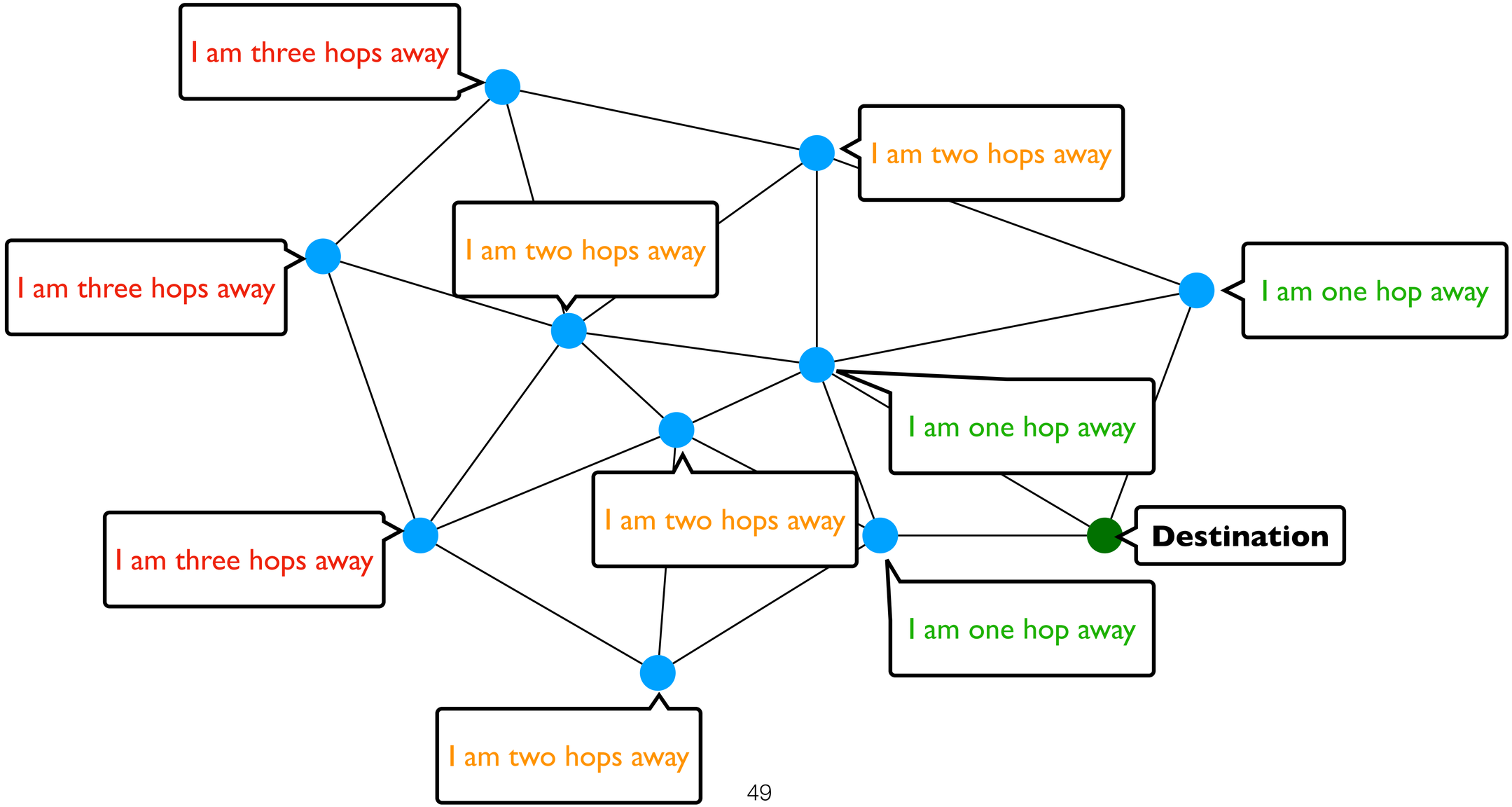
Distance Vector Routing: An example of ***Distributed*** Computation



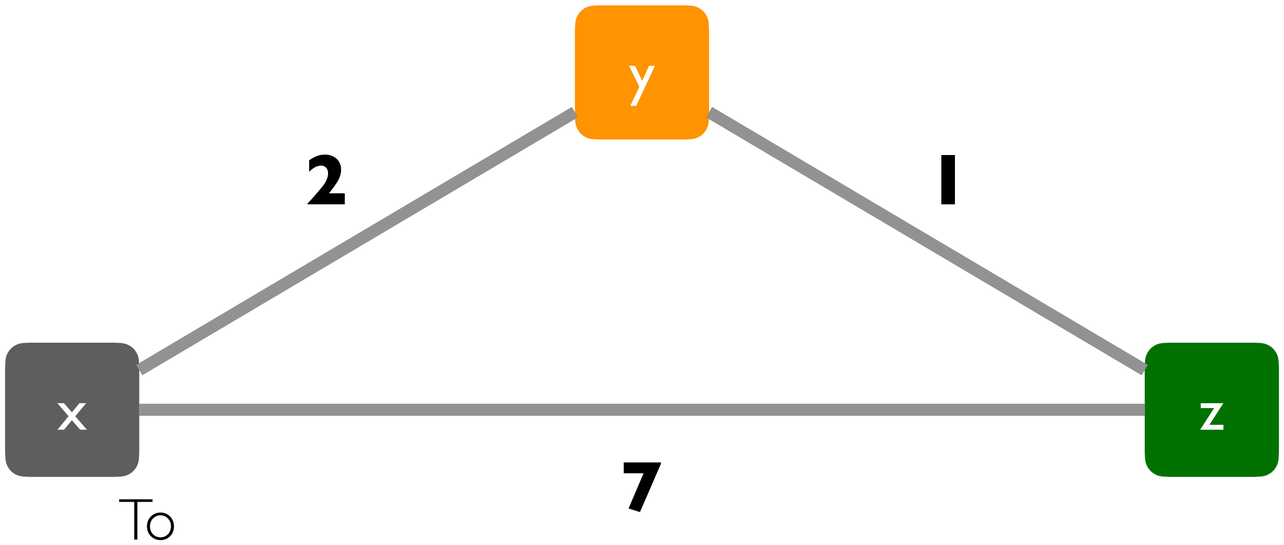
Distance Vector Routing: An example of ***Distributed*** Computation



Distance Vector Routing: An example of ***Distributed*** Computation



	x	y	z
x	-	-	-
y	2	0	1
z	-	-	-



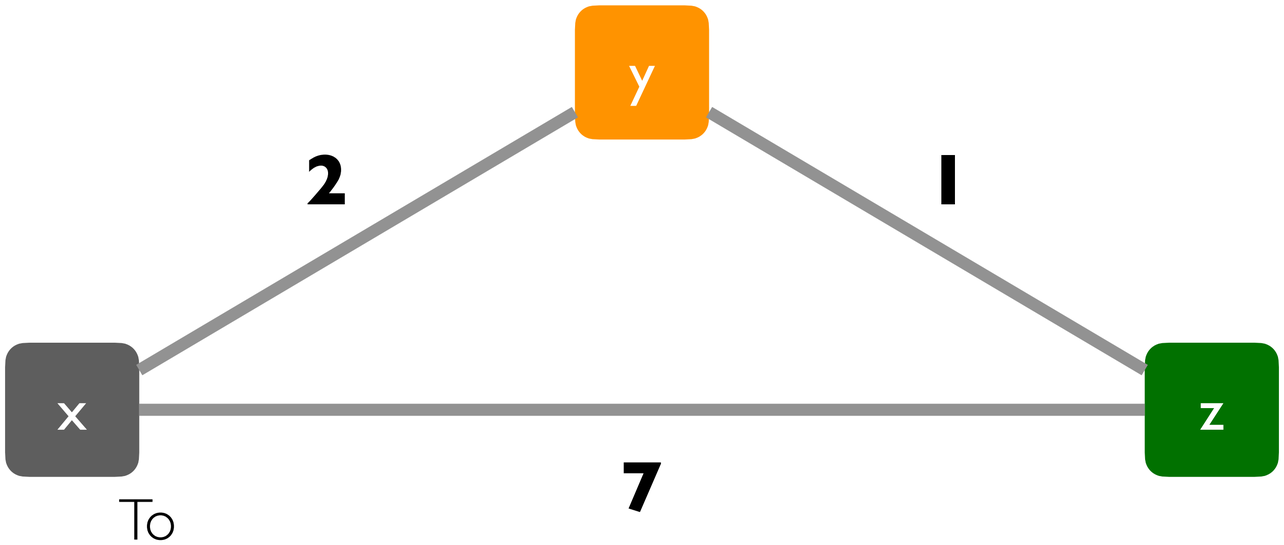
To

From

	x	y	z
x	0	2	7
y	-	-	-
z	-	-	-

	x	y	z
x	-	-	-
y	-	-	-
z	7	1	0

	x	y	z
x	-	-	-
y	2	0	1
z	-	-	-



To

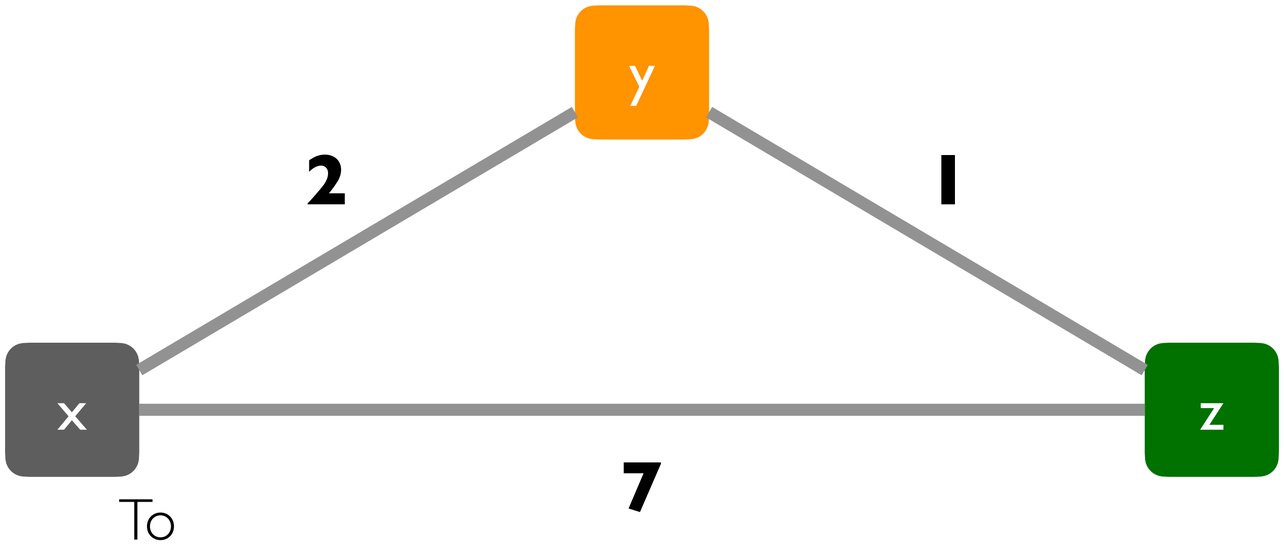
From

	x	y	z
x	0	2	7
y	-	-	-
z	-	-	-

	x	y	z
x	-	-	-
y	-	-	-
z	7	1	0

This is y's "distance vector"

	x	y	z
x	-	-	-
y	2	0	1
z	-	-	-



To

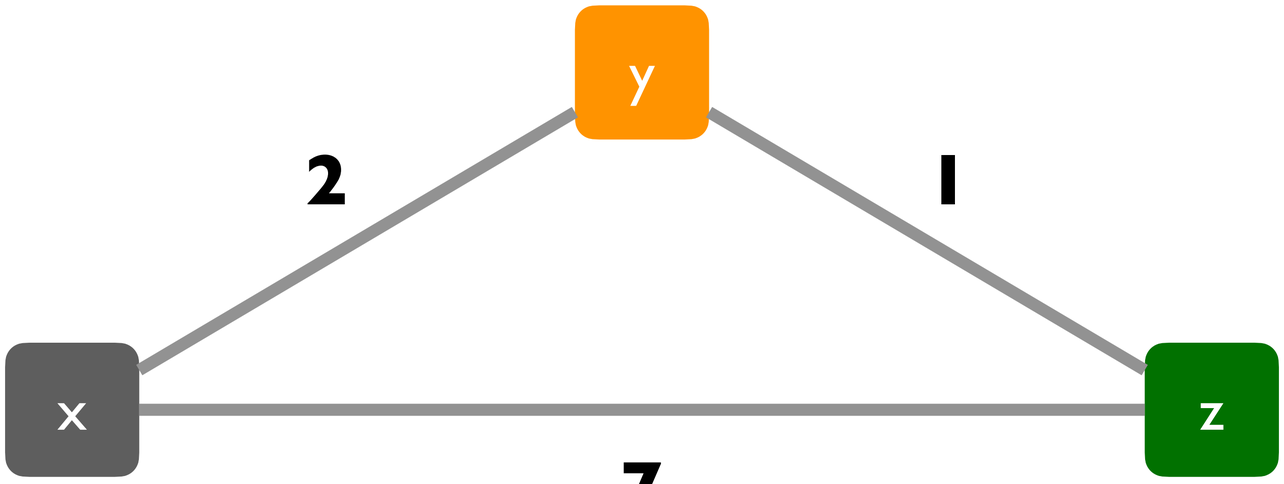
From

	x	y	z
x	0	2	7
y	-	-	-
z	-	-	-

	x	y	z
x	-	-	-
y	-	-	-
z	7	1	0

This is y's "distance vector"

	x	y	z
x	-	-	-
y	2	0	1
z	-	-	-



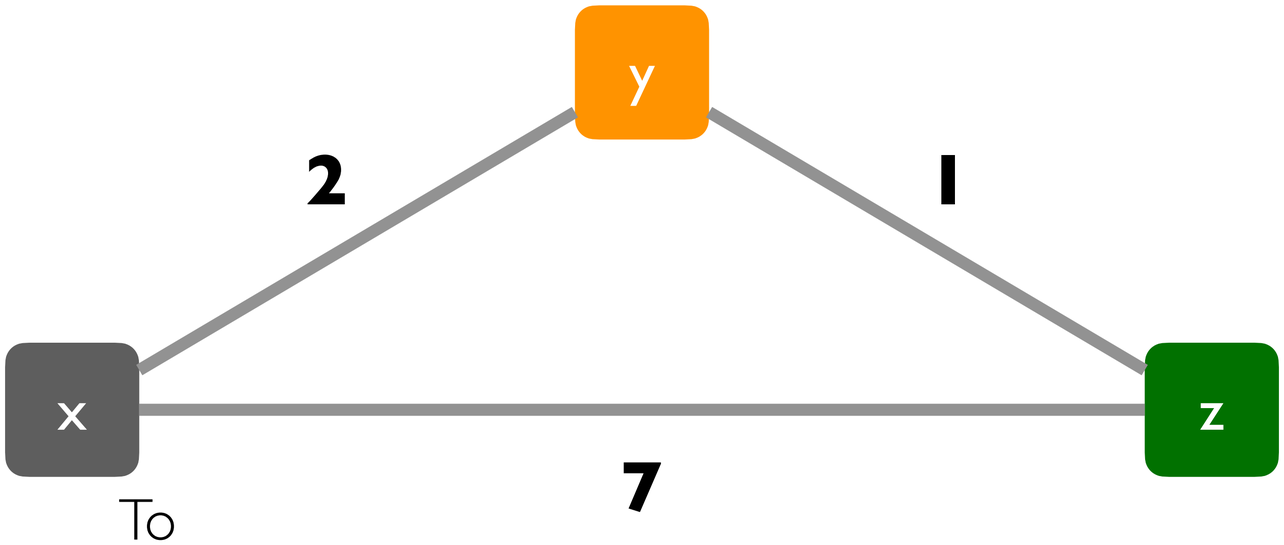
To

From

	x	y	z
x	0	2	7
y	-	-	-
z	-	-	-

	x	y	z
x	-	-	-
y	-	-	-
z	7	1	0

	x	y	z
x			
y	2	0	1
z			



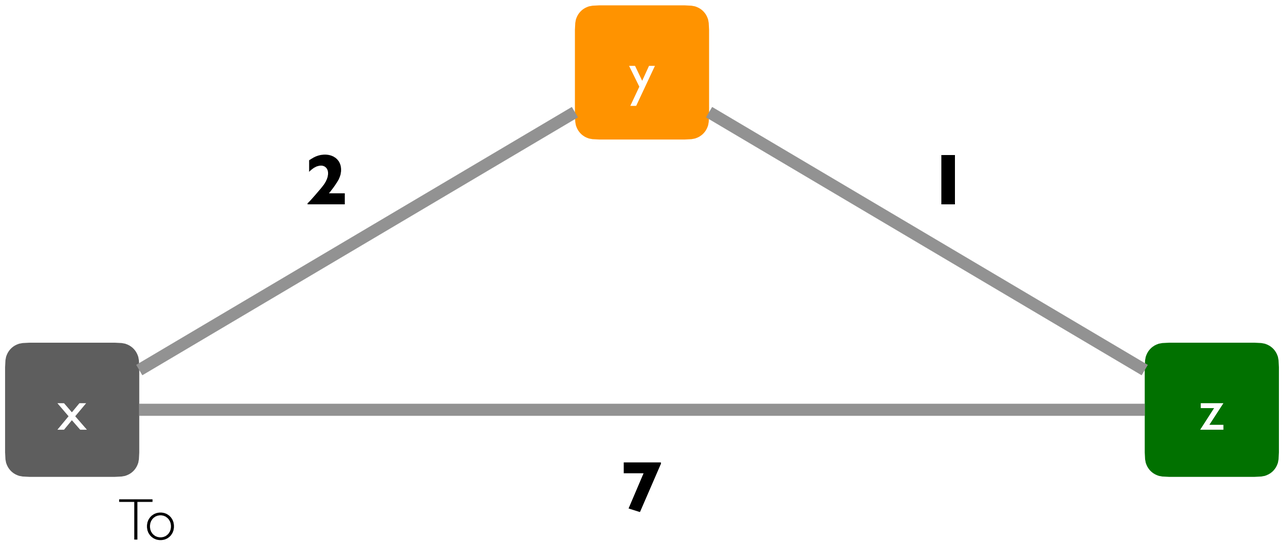
To

From

	x	y	z
x	0	2	7
y			
z			

	x	y	z
x			
y			
z	7	1	0

	x	y	z
x			
y	2	0	1
z			



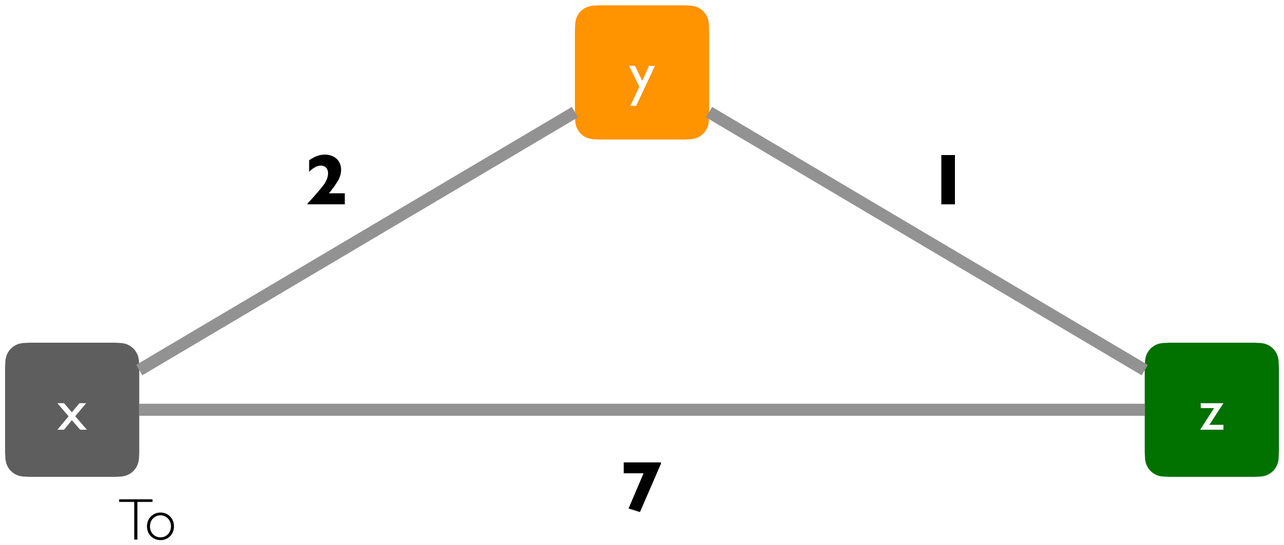
To

From

	x	y	z
x	0	2	7
y	2	0	1
z			

	x	y	z
x			
y			
z	7	1	0

	x	y	z
x			
y	2	0	1
z			



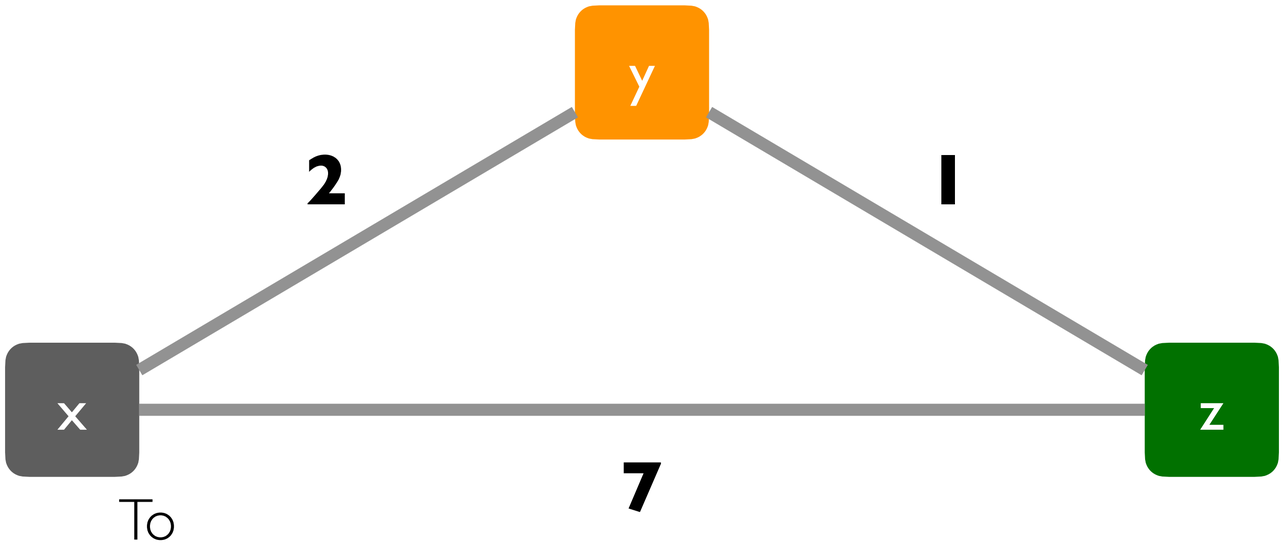
To

From

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x			
y			
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0



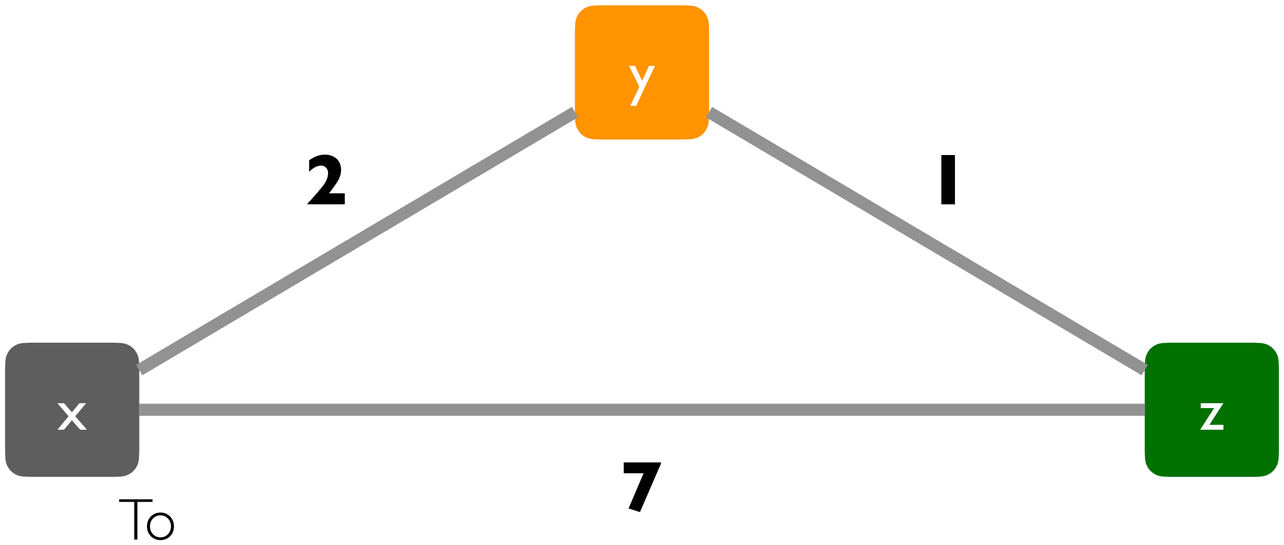
To

From

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0



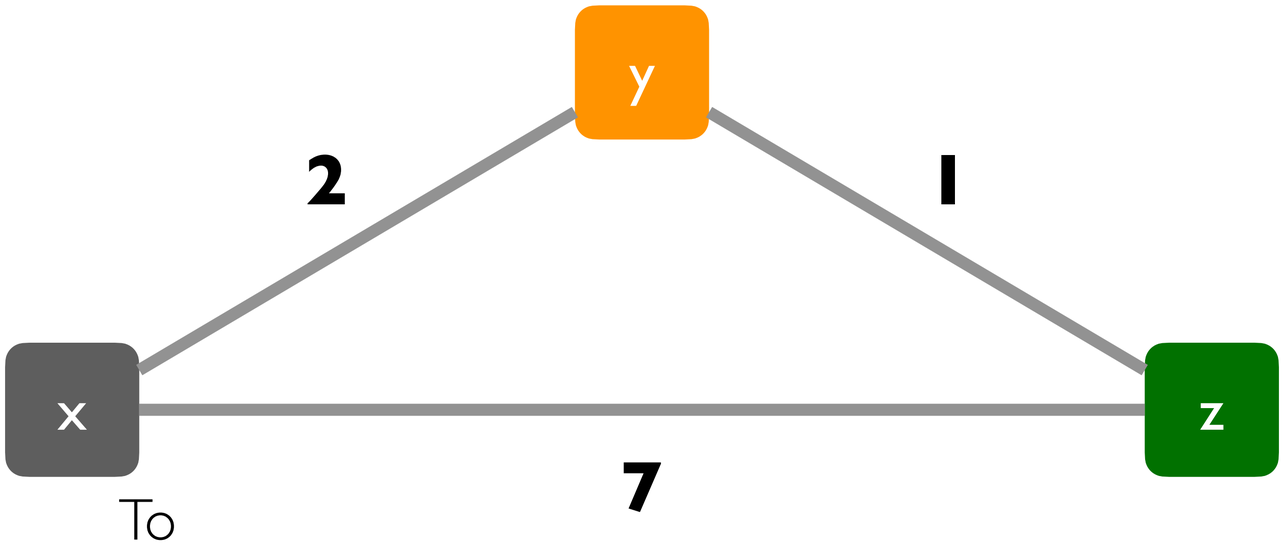
To

From

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0



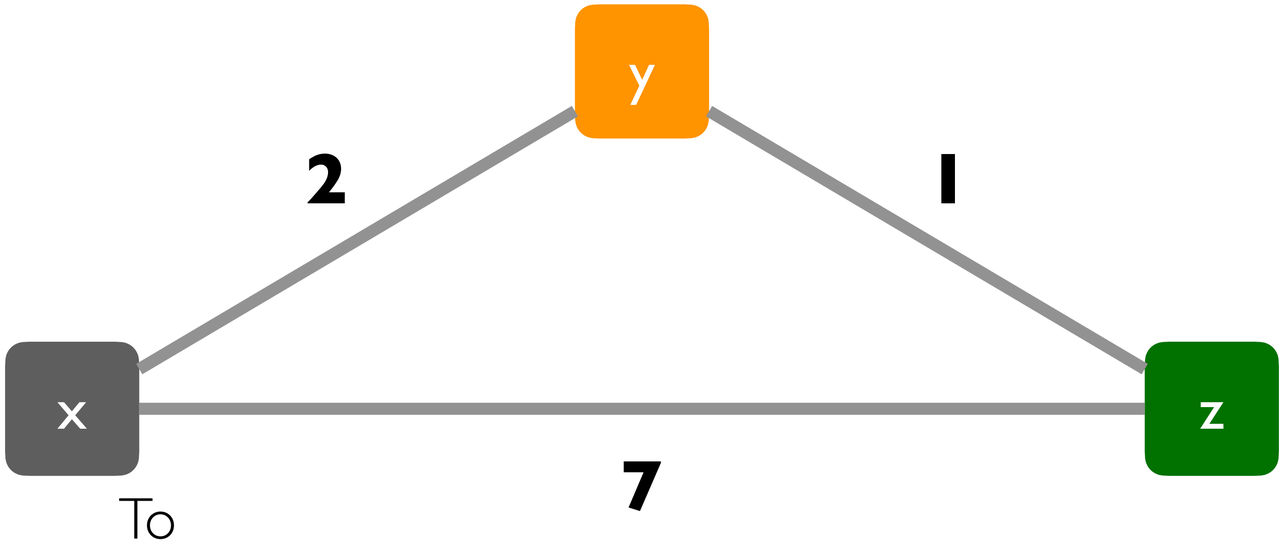
To

From

	x	y	z
x	0	2	~~7~~
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0



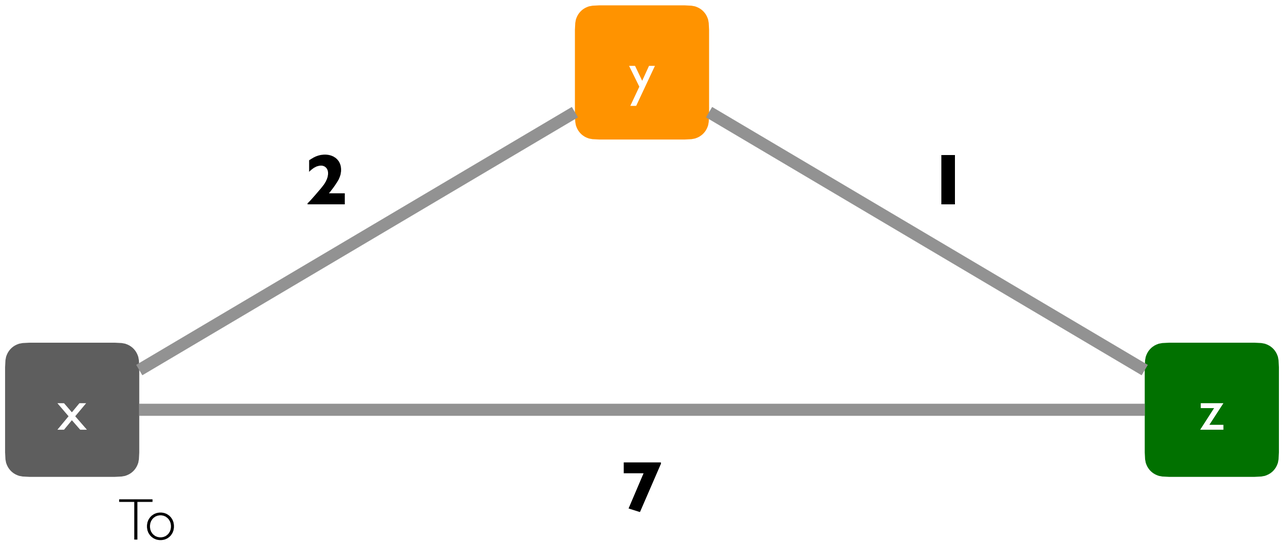
To

From

	x	y	z
x	0	~~2~~	~~7~~
y	2	0	~~1~~
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0



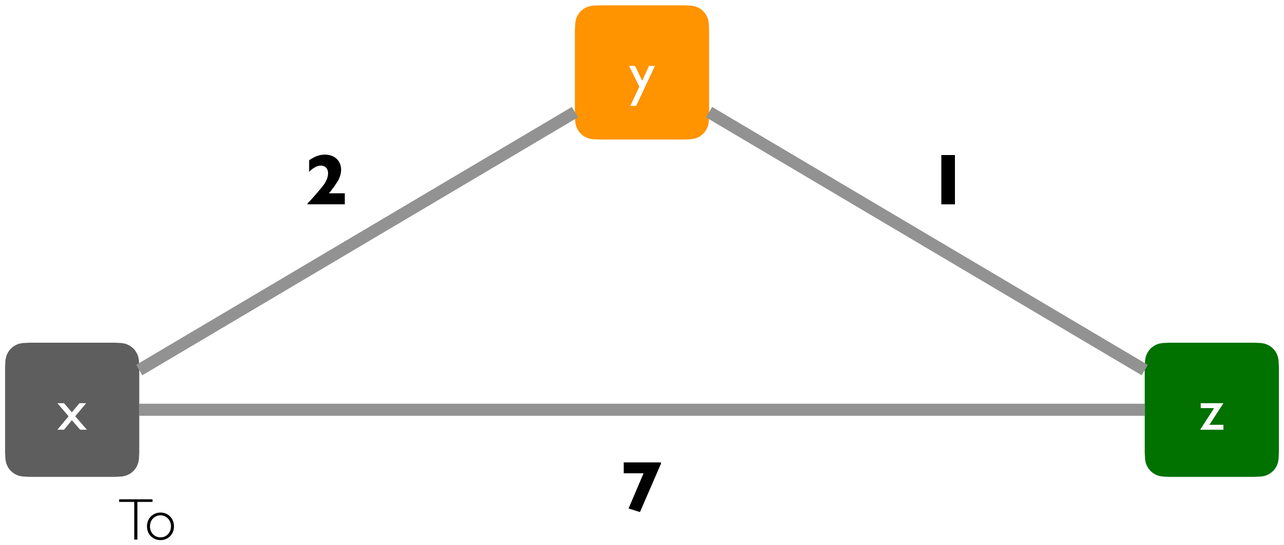
To

From

	x	y	z
x	0	~~2~~	3
y	2	0	~~1~~
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0



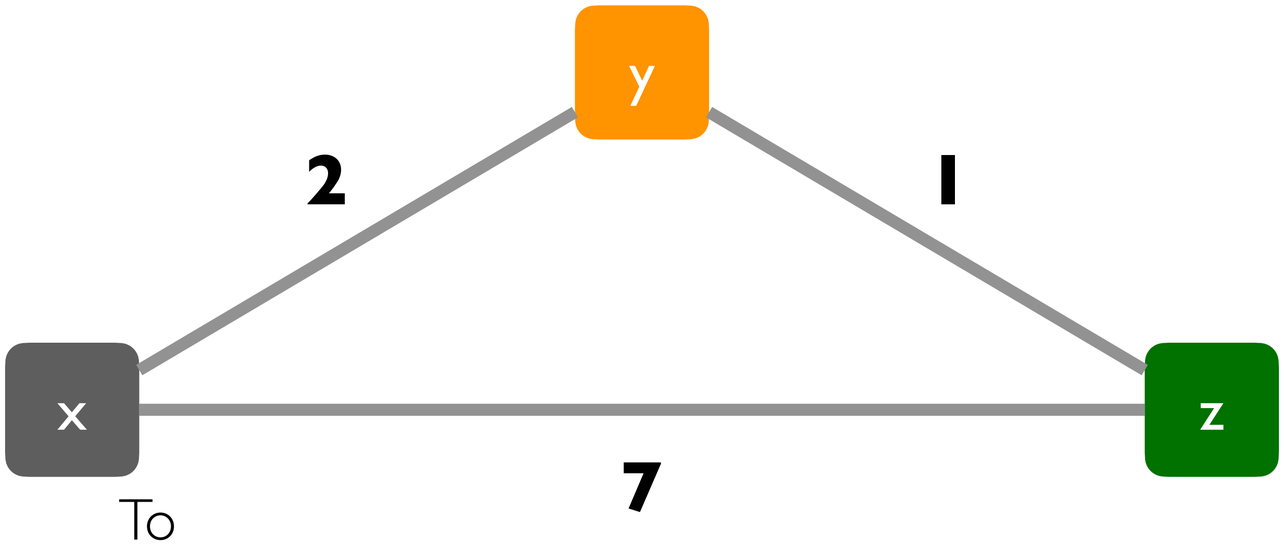
To

From

	x	y	z
x	0	~~2~~	3
y	2	0	~~1~~
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	~~7~~	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0



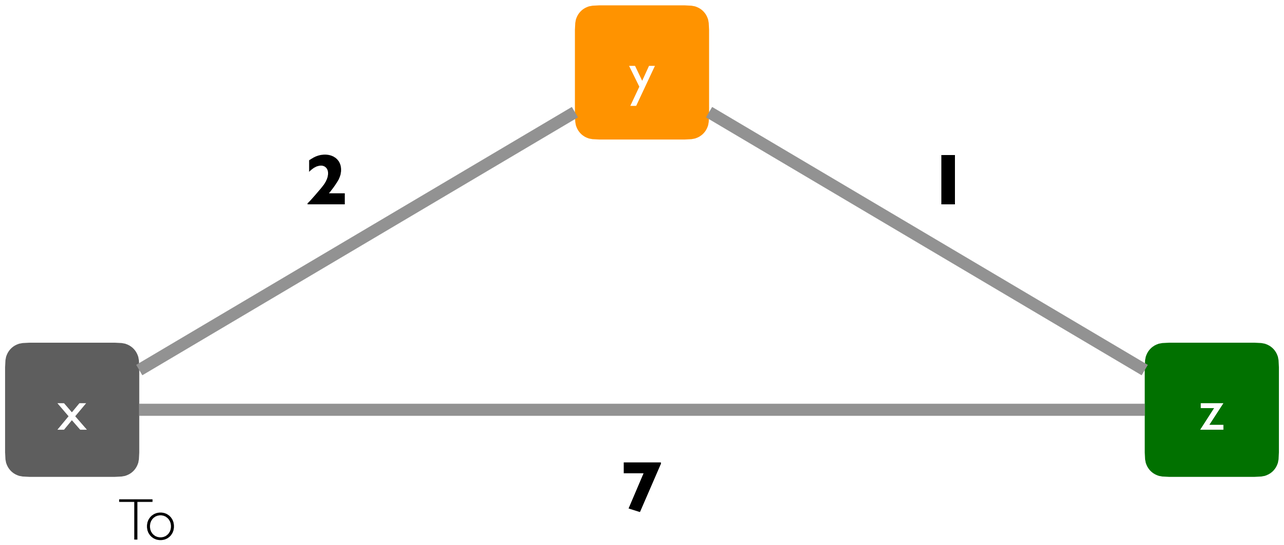
To

From

	x	y	z
x	0	~~2~~	3
y	2	0	~~1~~
z	7	1	0

	x	y	z
x	0	2	7
y	~~2~~	0	1
z	~~7~~	~~1~~	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0



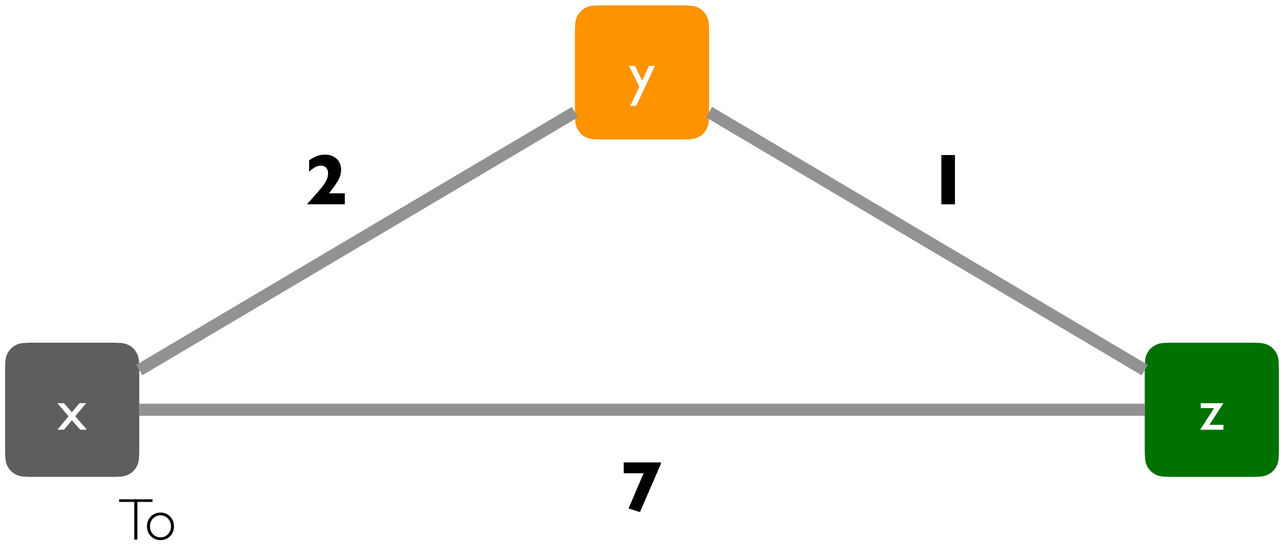
To

From

	x	y	z
x	0	~~2~~	3
y	2	0	~~1~~
z	7	1	0

	x	y	z
x	0	2	7
y	~~2~~	0	1
z	3	~~1~~	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0



To

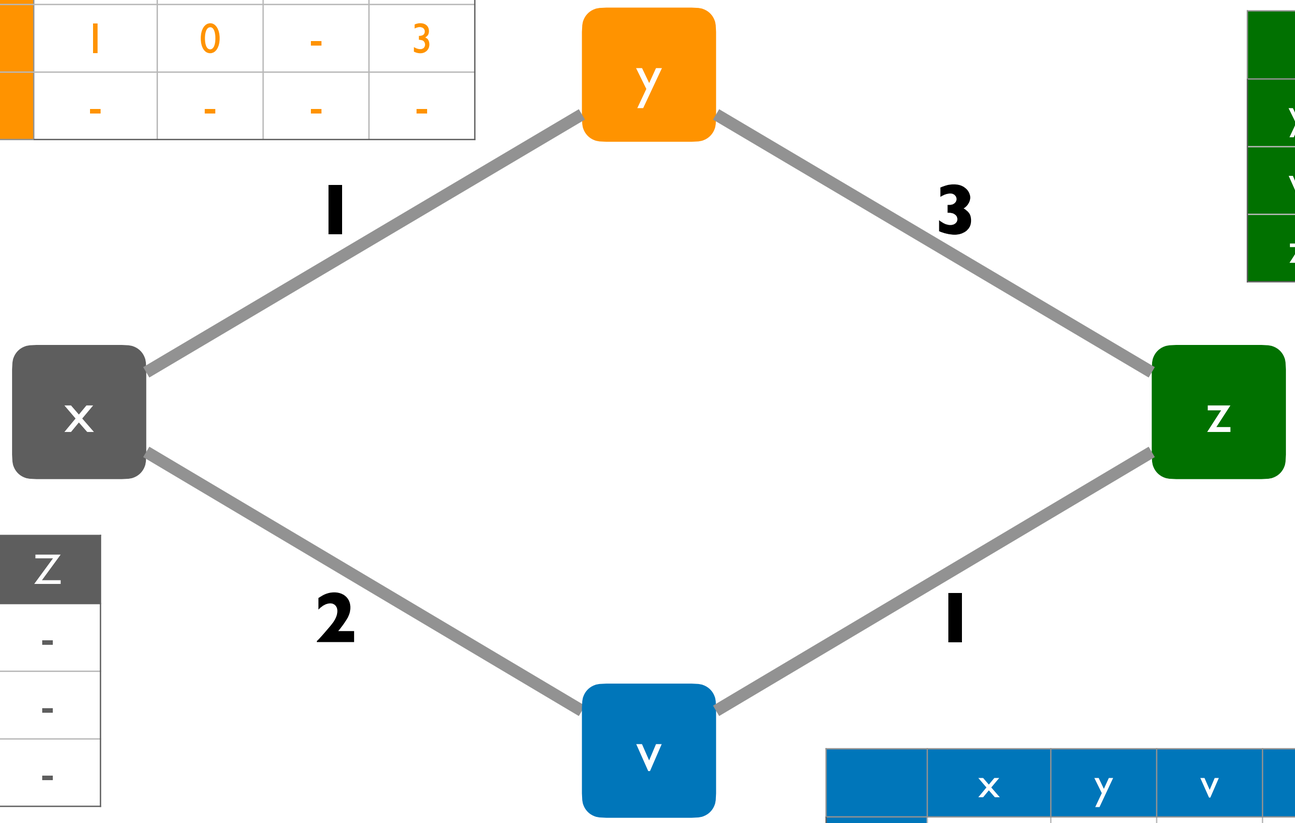
From

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

	x	y	v	z
x	-	-	-	-
y	1	0	-	3
z	-	-	-	-

	x	y	v	z
y	-	-	-	-
v	-	-	-	-
z	-	3	1	0



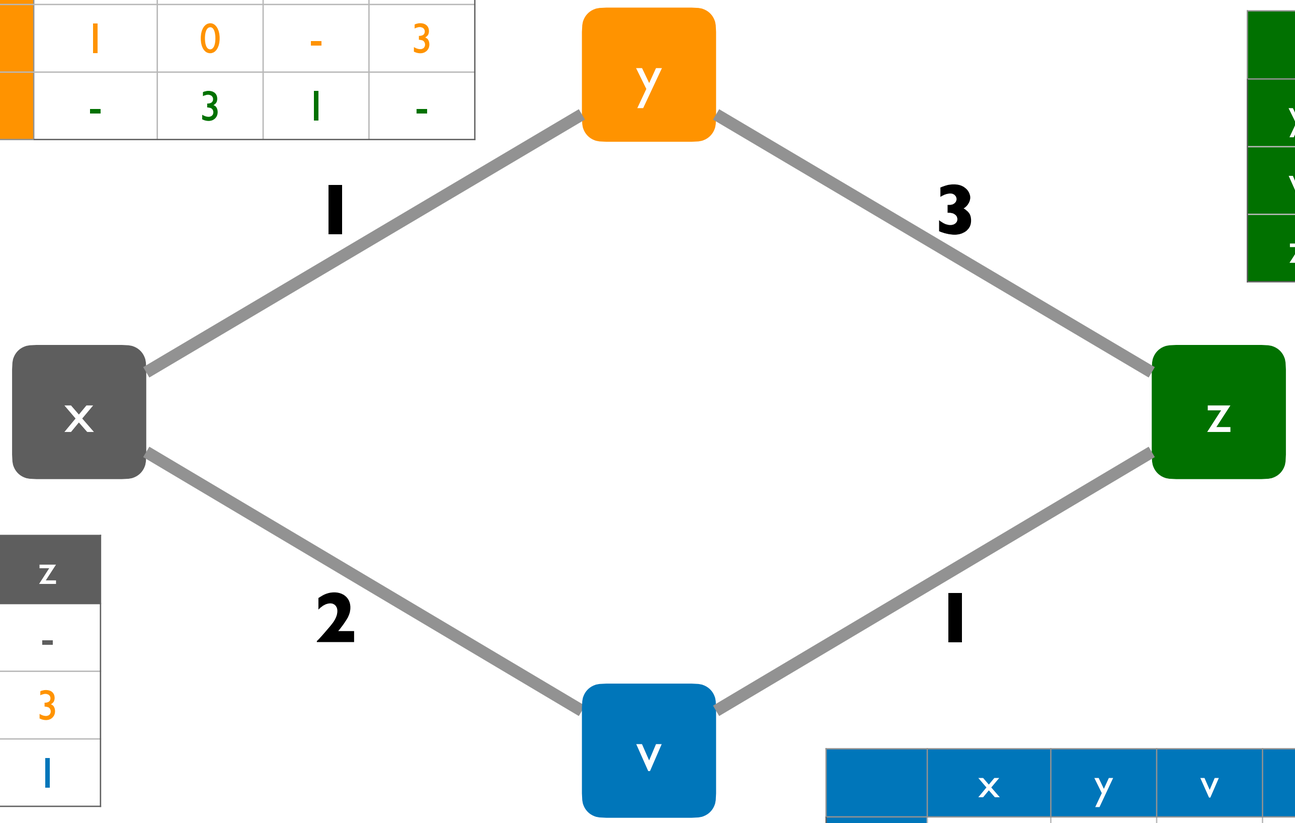
To

From

	x	y	v	Z
x	0	1	2	-
y	-	-	-	-
v	-	-	-	-

	x	y	v	z
x	-	-	-	-
v	2	-	0	1
z	-	-	-	-

	x	y	v	z
x	0	1	2	-
y	1	0	-	3
z	-	3	1	-



	x	y	v	z
y	1	0	-	3
v	2	-	0	1
z	-	3	1	0

To

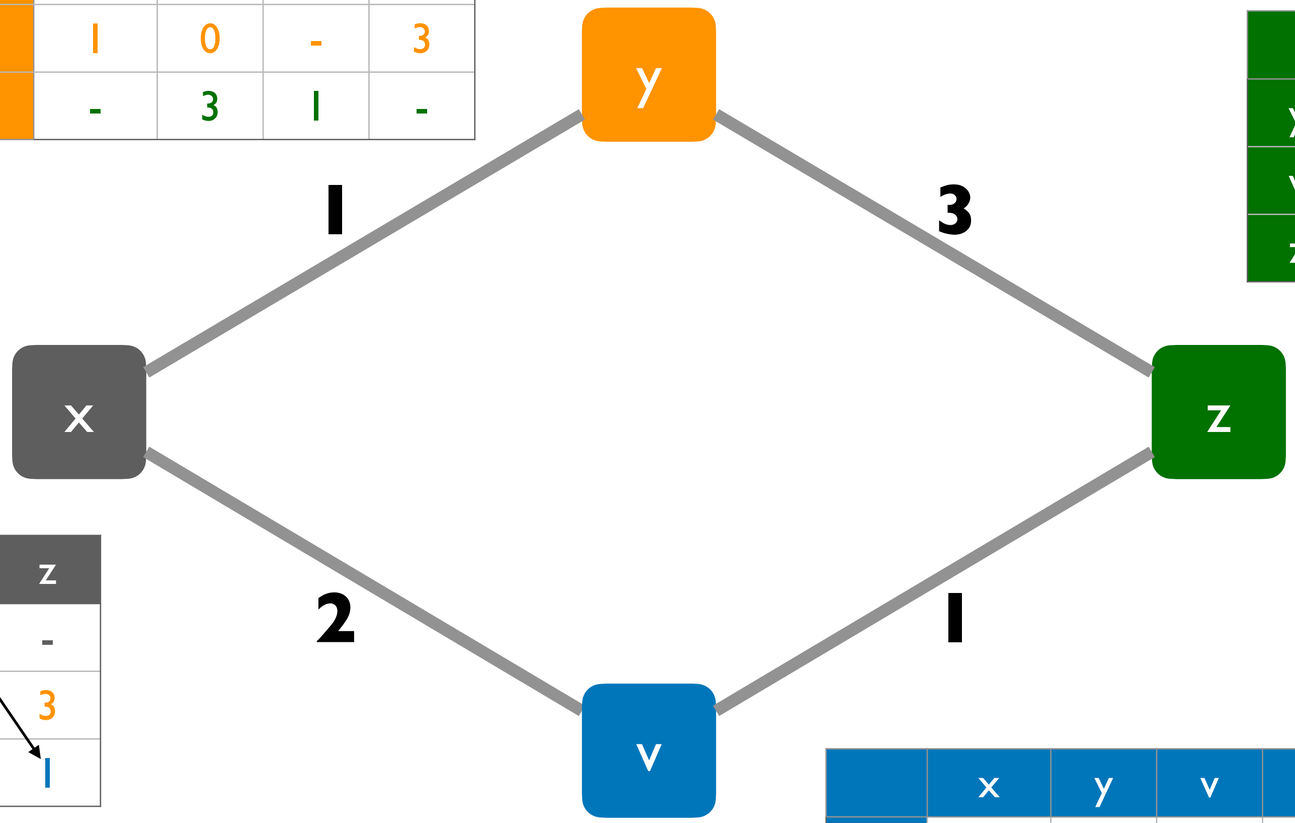
From

	x	y	v	z
x	0	1	2	-
y	1	0	-	3
v	2	-	0	1

	x	y	v	z
x	0	1	2	-
v	2	-	0	1
z	-	3	1	0

	x	y	v	z
x	0	1	2	-
y	1	0	-	3
z	-	3	1	-

	x	y	v	z
y	1	0	-	3
v	2	-	0	1
z	-	3	1	0



To

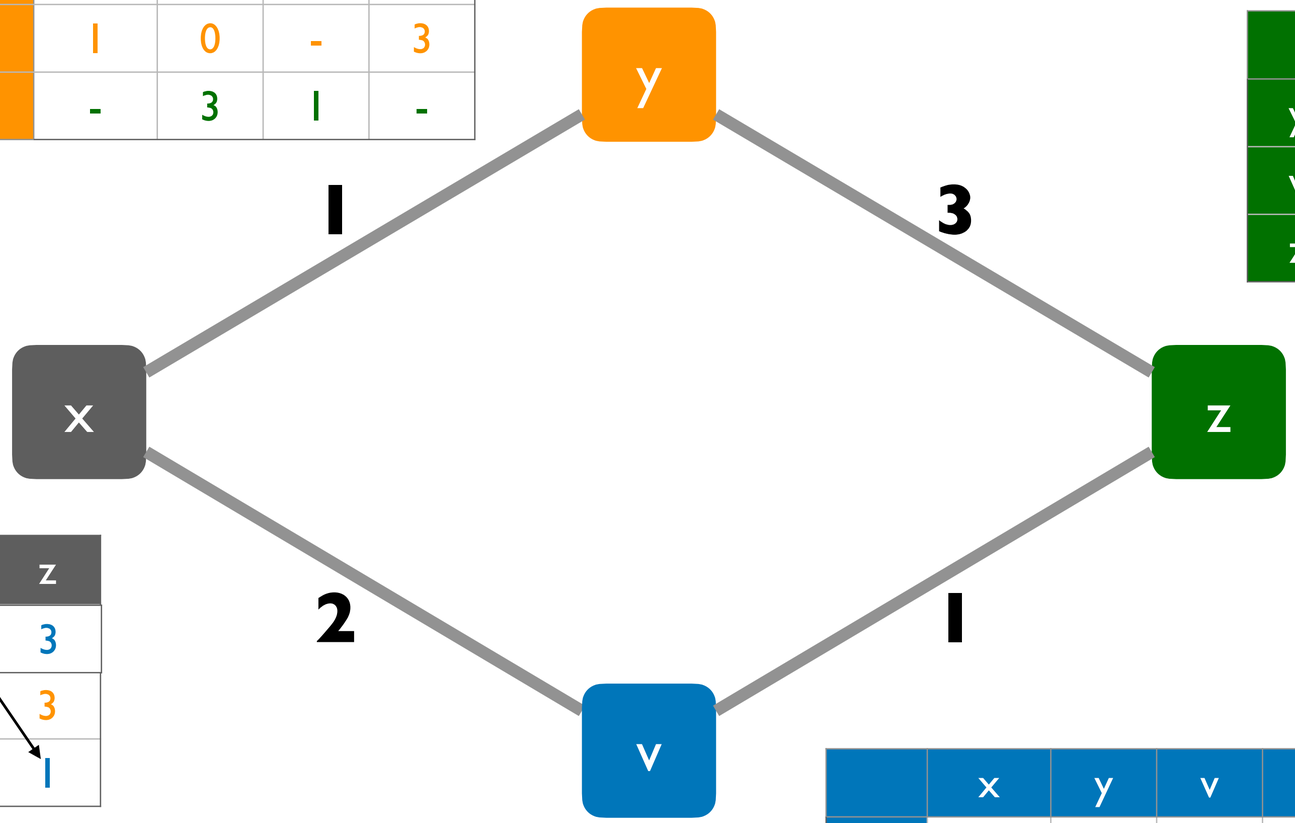
From

	x	y	v	z
x	0	1	2	-
y	1	0	-	3
v	2	-	0	1

	x	y	v	z
x	0	1	2	-
v	2	-	0	1
z	-	3	1	0

	x	y	v	z
x	0	1	2	-
y	1	0	-	3
z	-	3	1	-

	x	y	v	z
y	1	0	-	3
v	2	-	0	1
z	-	3	1	0



To

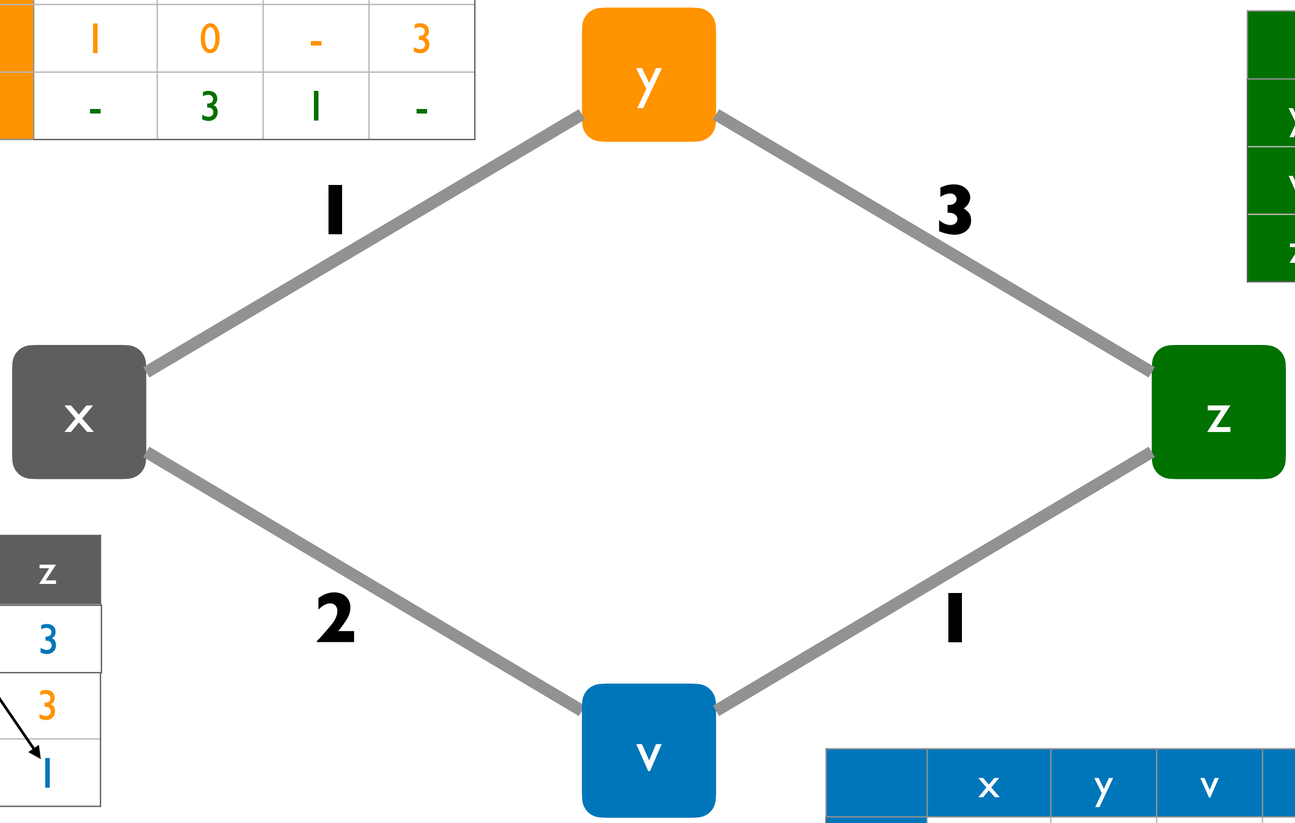
From

	x	y	v	z
x	0	1	2	3
y	1	0	-	3
v	2	-	0	1

	x	y	v	z
x	0	1	2	-
v	2	-	0	1
z	-	3	1	0

	x	y	v	z
x	0	1	2	-
y	1	0	-	3
z	-	3	1	-

	x	y	v	z
y	1	0	-	3
v	2	-	0	1
z	-	3	1	0



To

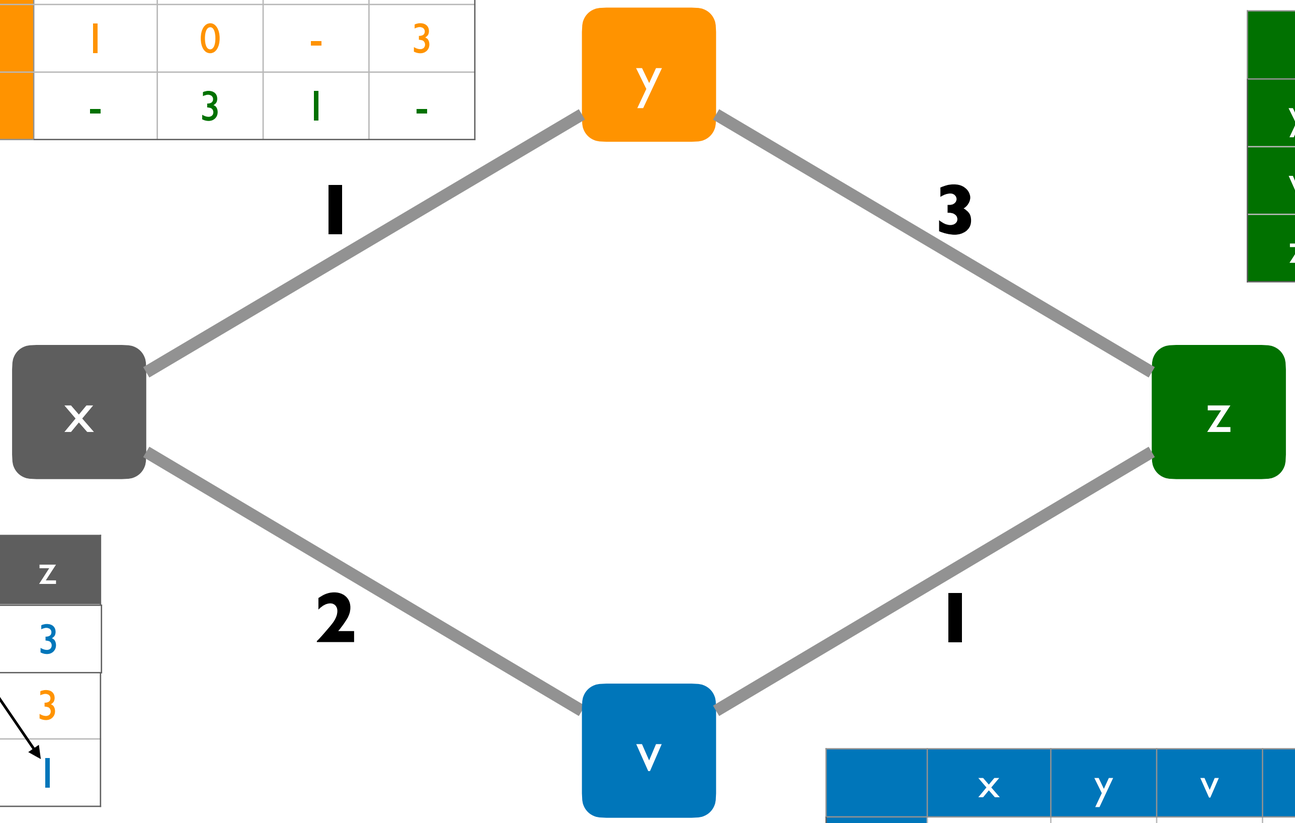
From

	x	y	v	z
x	0	1	2	3
y	1	0	-	3
v	2	-	0	1

	x	y	v	z
x	0	1	2	-
v	2	-	0	1
z	-	3	1	0

	x	y	v	z
x	0	1	2	-
y	1	0	-	3
z	-	3	1	-

	x	y	v	z
y	1	0	-	3
v	2	-	0	1
z	3	3	1	0



To

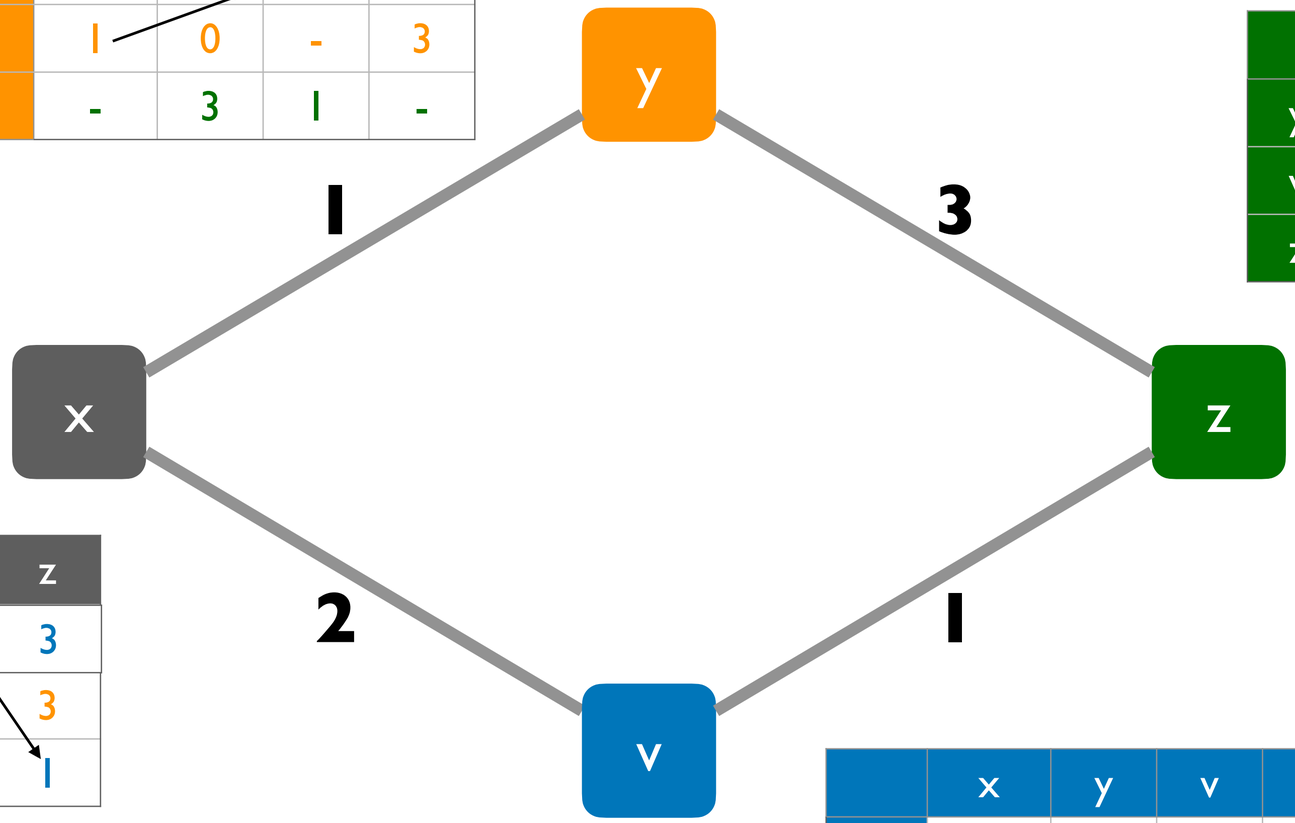
From

	x	y	v	z
x	0	1	2	3
y	1	0	-	3
v	2	-	0	1

	x	y	v	z
x	0	1	2	-
v	2	-	0	1
z	-	3	1	0

	x	y	v	z
x	0	1	2	-
y	1	0	-	3
z	-	3	1	-

	x	y	v	z
y	1	0	-	3
v	2	-	0	1
z	3	3	1	0



To

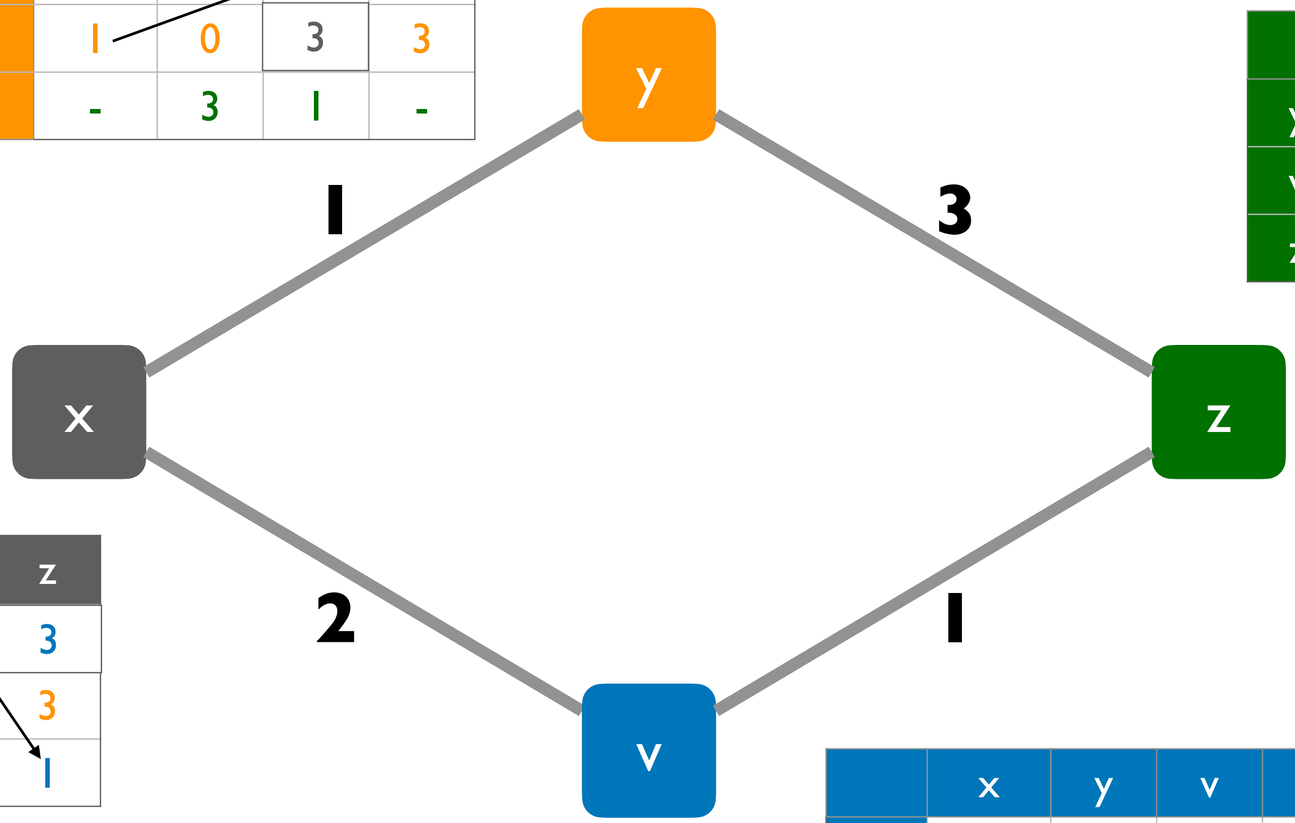
From

	x	y	v	z
x	0	1	2	3
y	1	0	-	3
v	2	-	0	1

	x	y	v	z
x	0	1	2	-
v	2	-	0	1
z	-	3	1	0

	x	y	v	z
x	0	1	2	-
y	1	0	3	3
z	-	3	1	-

	x	y	v	z
y	1	0	-	3
v	2	-	0	1
z	3	3	1	0



To

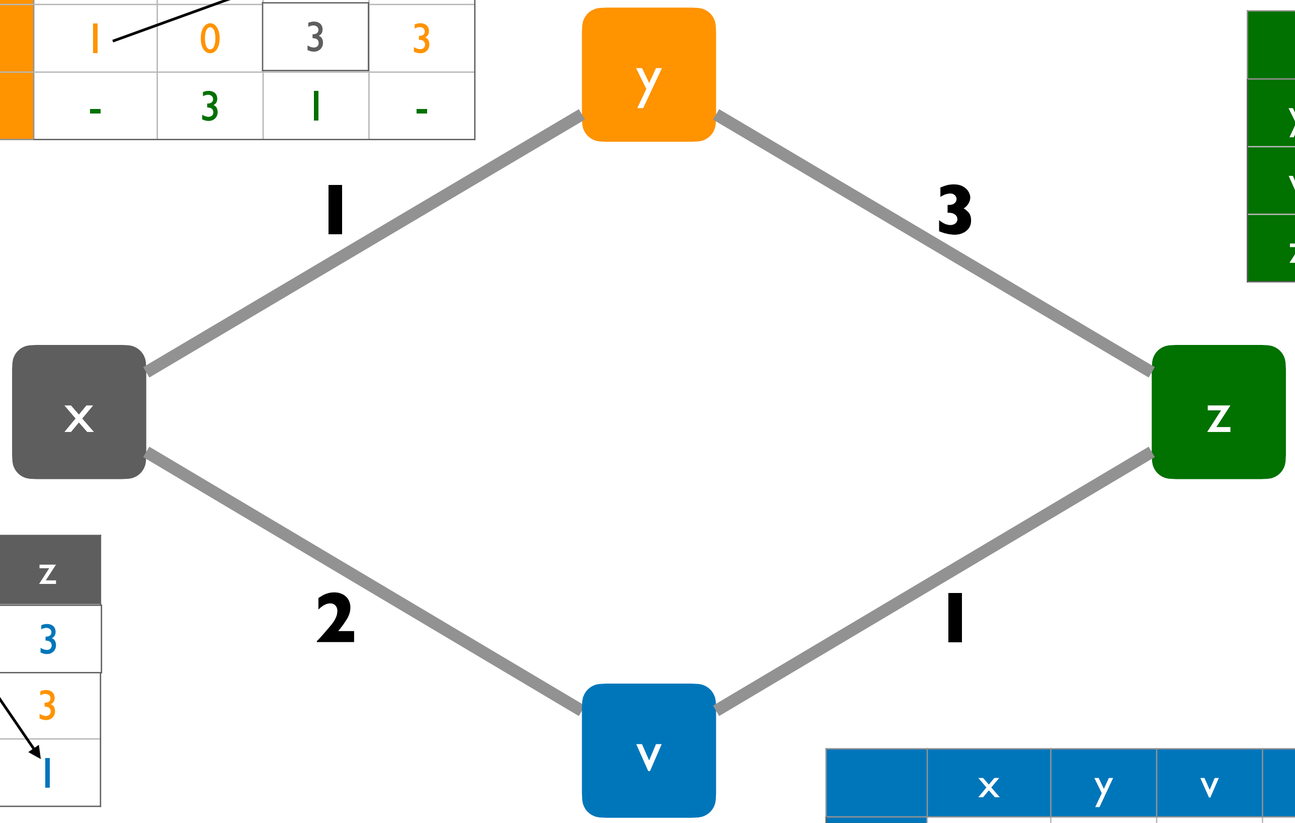
From

	x	y	v	z
x	0	1	2	3
y	1	0	-	3
v	2	-	0	1

	x	y	v	z
x	0	1	2	-
v	2	-	0	1
z	-	3	1	0

	x	y	v	z
x	0	1	2	-
y	1	0	3	3
z	-	3	1	-

	x	y	v	z
y	1	0	-	3
v	2	-	0	1
z	3	3	1	0



To

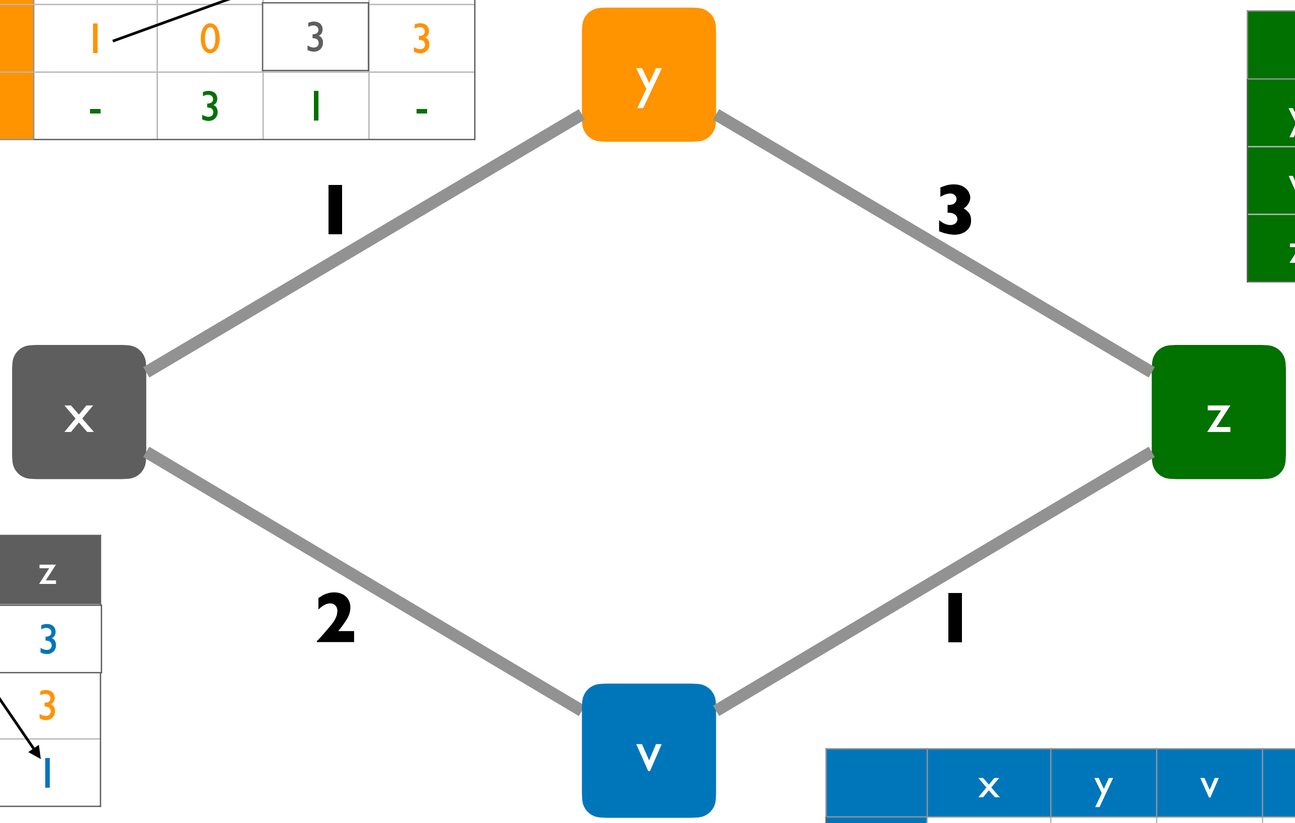
From

	x	y	v	z
x	0	1	2	3
y	1	0	-	3
v	2	-	0	1

	x	y	v	z
x	0	1	2	-
v	2	-	0	1
z	-	3	1	0

	x	y	v	z
x	0	1	2	-
y	1	0	3	3
z	-	3	1	-

	x	y	v	z
y	1	0	-	3
v	2	-	0	1
z	3	3	1	0



To

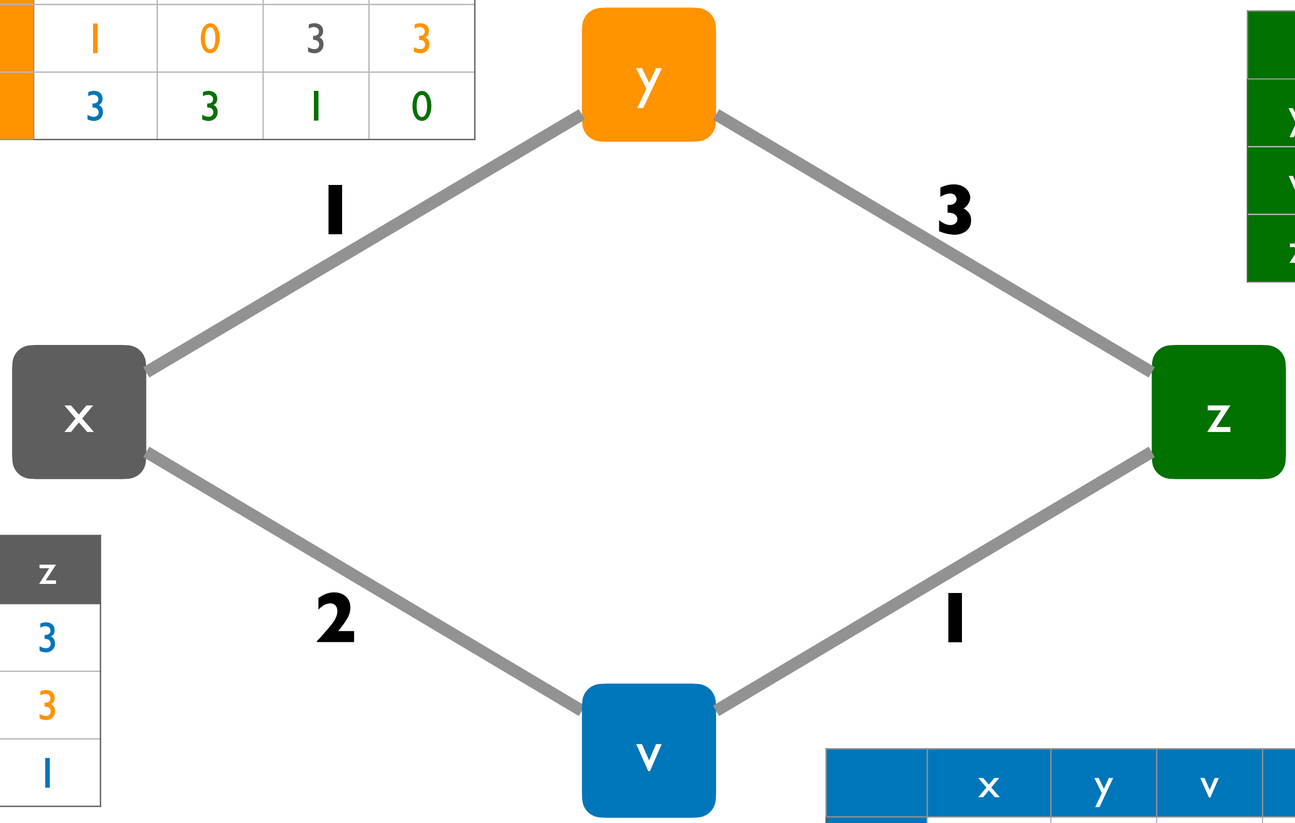
From

	x	y	v	z
x	0	1	2	3
y	1	0	-	3
v	2	-	0	1

	x	y	v	z
x	0	1	2	-
v	2	3	0	1
z	-	3	1	0

	x	y	v	z
x	0	1	2	3
y	1	0	3	3
z	3	3	1	0

	x	y	v	z
y	1	0	3	3
v	2	3	0	1
z	3	3	1	0



To

From

	x	y	v	z
x	0	1	2	3
y	1	0	3	3
v	2	3	0	1

	x	y	v	z
x	0	1	2	3
v	2	3	0	1
z	3	3	1	0

Distance-vector Routing

Distance-vector Routing

- Input to each router:
  - *Local link costs & neighbor messages*

Distance-vector Routing

- Input to each router:
  - *Local link costs & neighbor messages*
- Output of each router:
  - *Least-cost path to every other router*

Distance-vector Routing

Distance-vector Routing

- Distributed algorithm

Distance-vector Routing

- Distributed algorithm
- All routers run it “together"
  - *Each router runs its own instance*
  - *Neighbors exchange and react to each other’s messages*

Bellman-Ford Algorithm

Bellman-Ford Algorithm

- All neighbors exchange information
  - *Each router checks whether it can improve current paths by leveraging the new information*

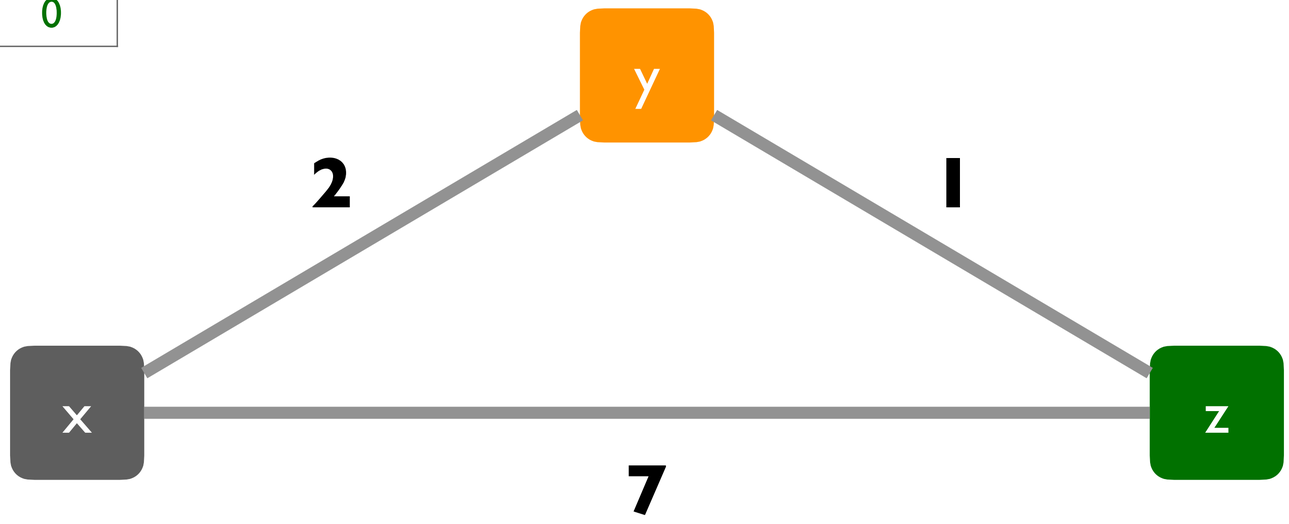
Bellman-Ford Algorithm

- All neighbors exchange information
  - *Each router checks whether it can improve current paths by leveraging the new information*
- Ends when no improvements are possible

To

From

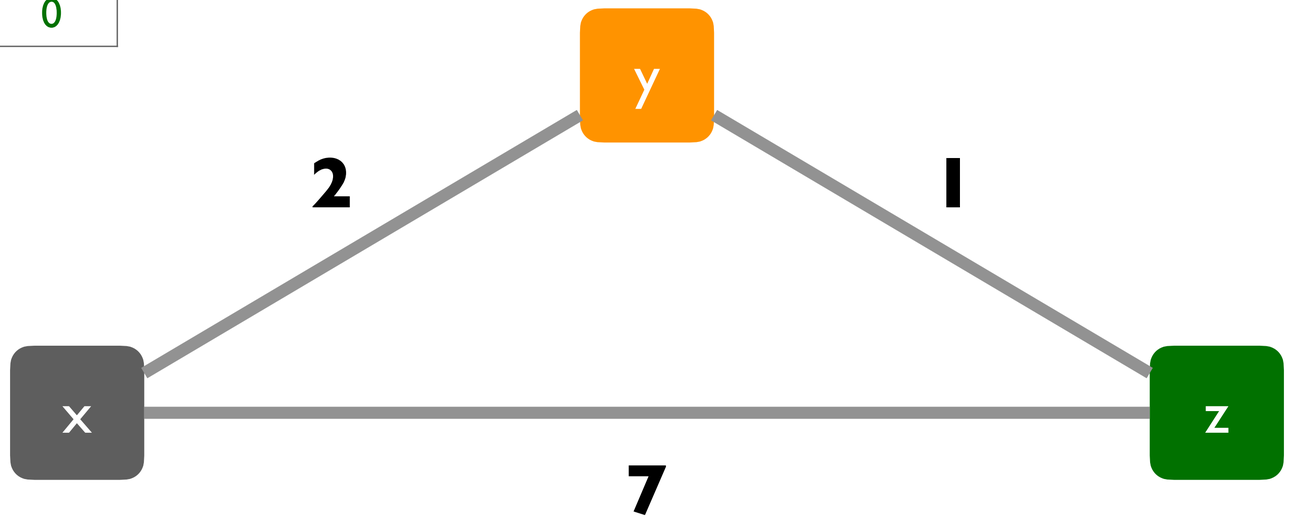
	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0



To

From

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

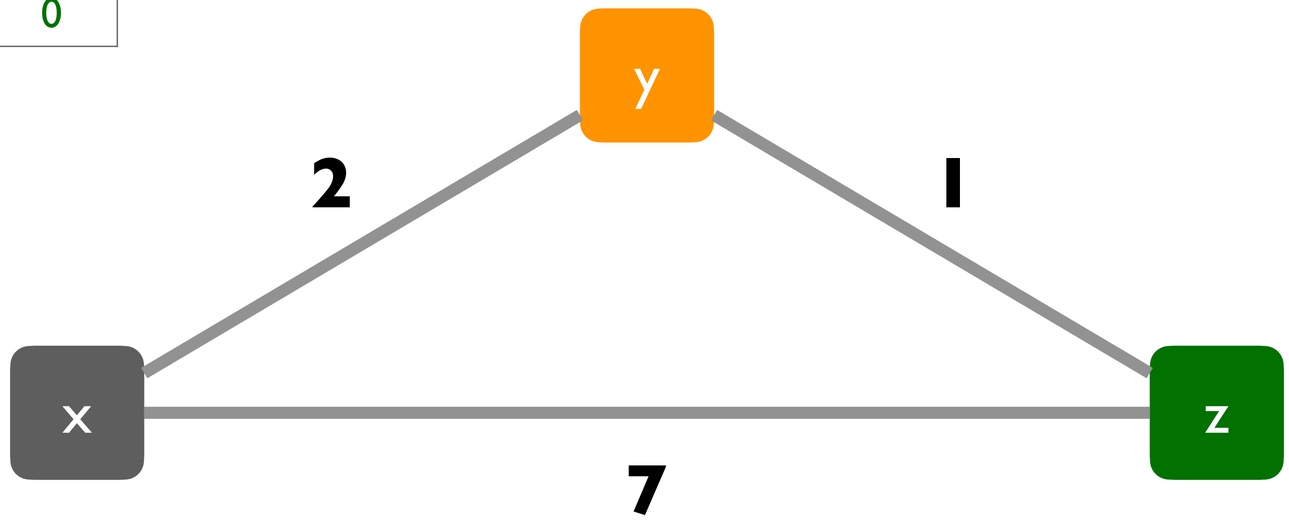


$$d_x(z) =$$

To

From

	x	y	z
x	0	2	~~7~~
y	2	0	1
z	7	1	0

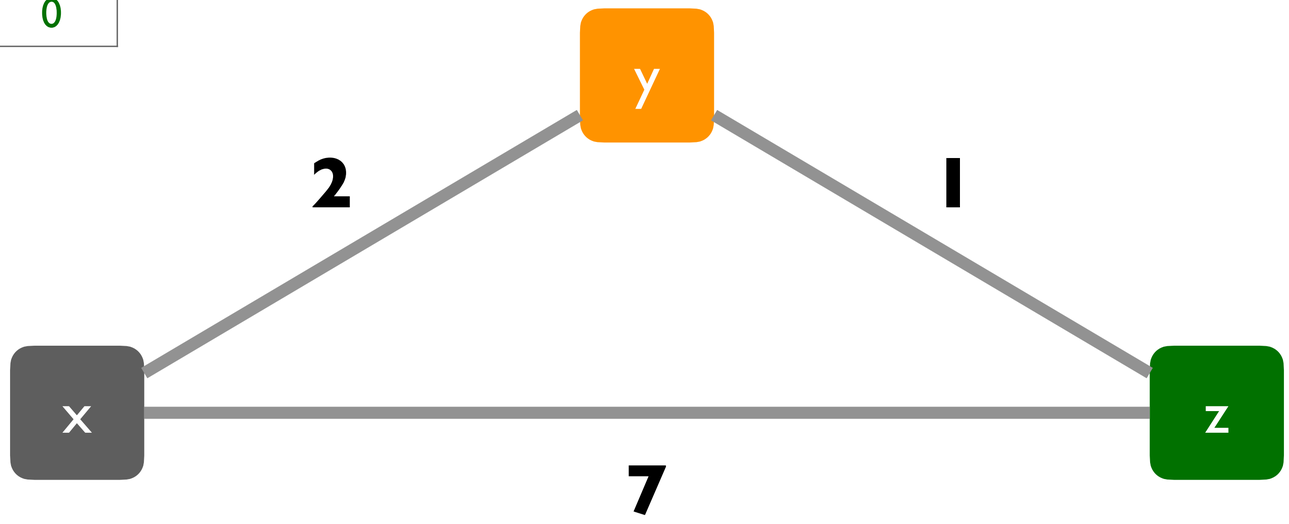


$$d_x(z) = \qquad cost(x, z),$$

To

From

	x	y	z
x	0	~~2~~	~~7~~
y	2	0	~~1~~
z	7	1	0

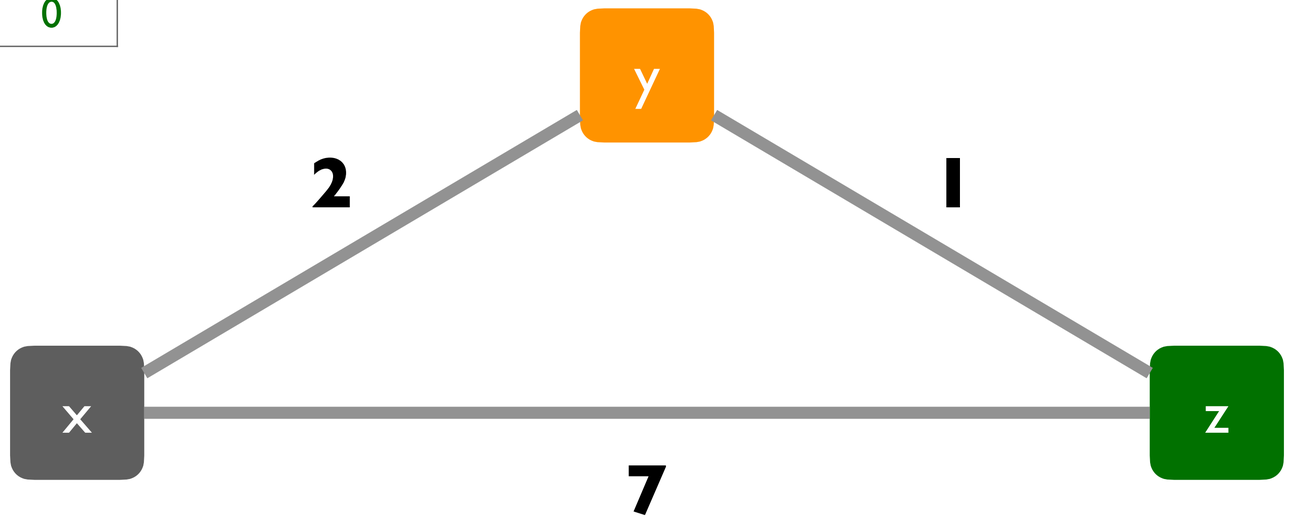


$$d_x(z) = \begin{array}{l} cost(x, z), \\ cost(x, y) + d_y(z) \end{array}$$

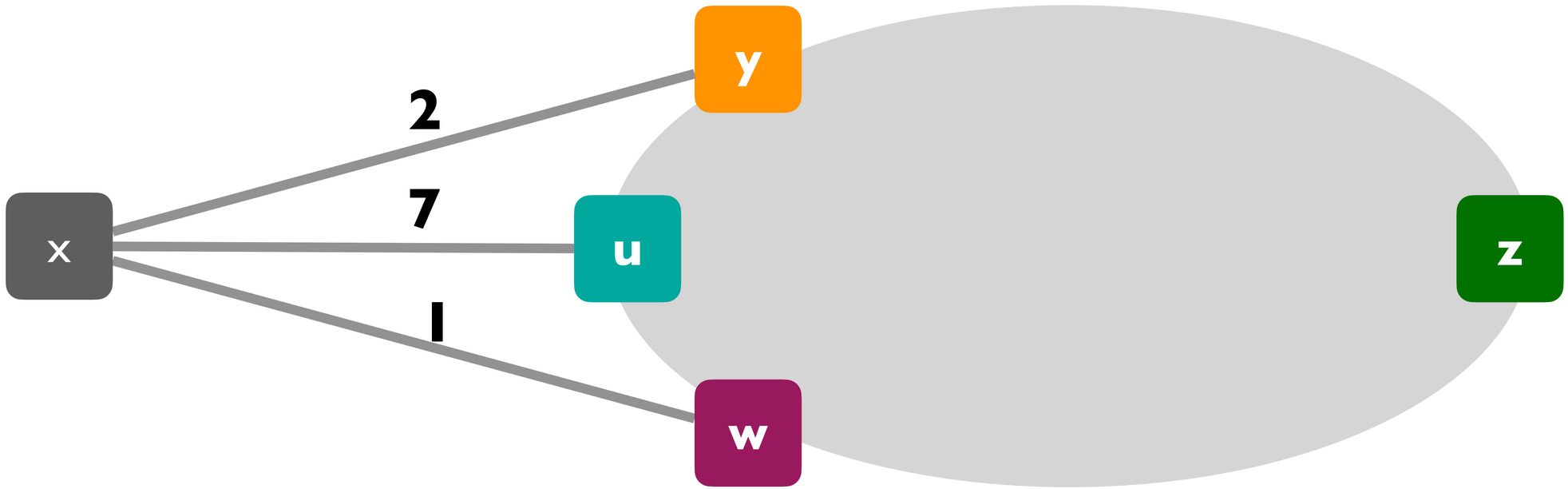
To

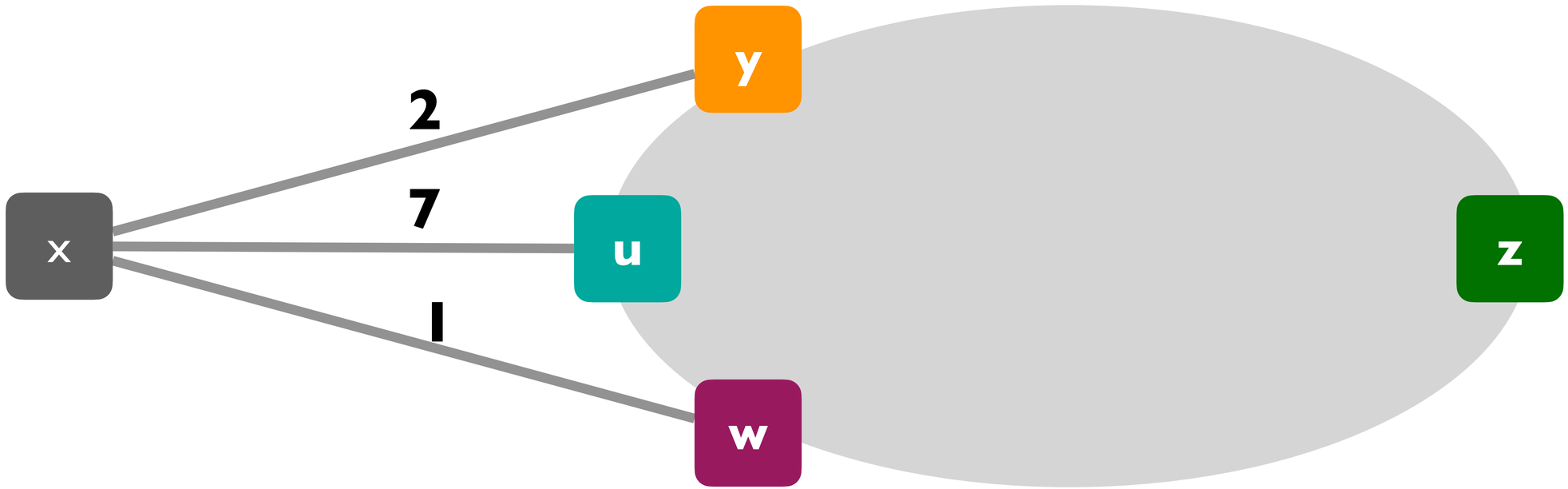
From

	x	y	z
x	0	~~2~~	~~7~~
y	2	0	~~1~~
z	7	1	0

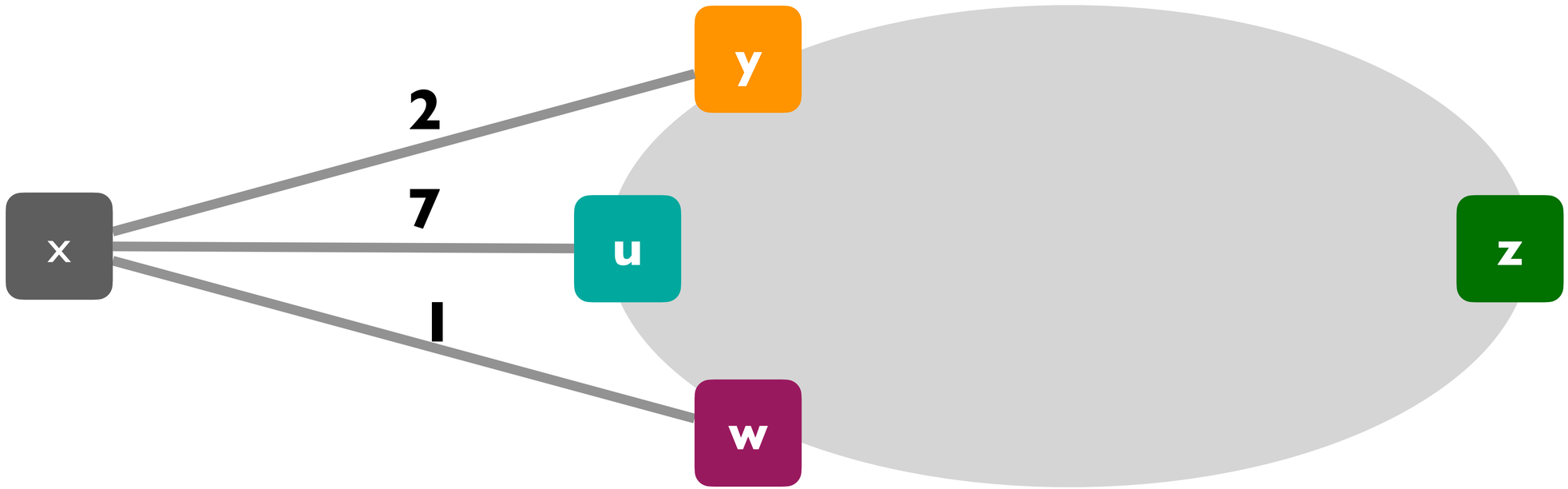


$$d_x(z) = \min\{\ cost(x,z),$$
$$cost(x,y) + d_y(z)\ \}$$

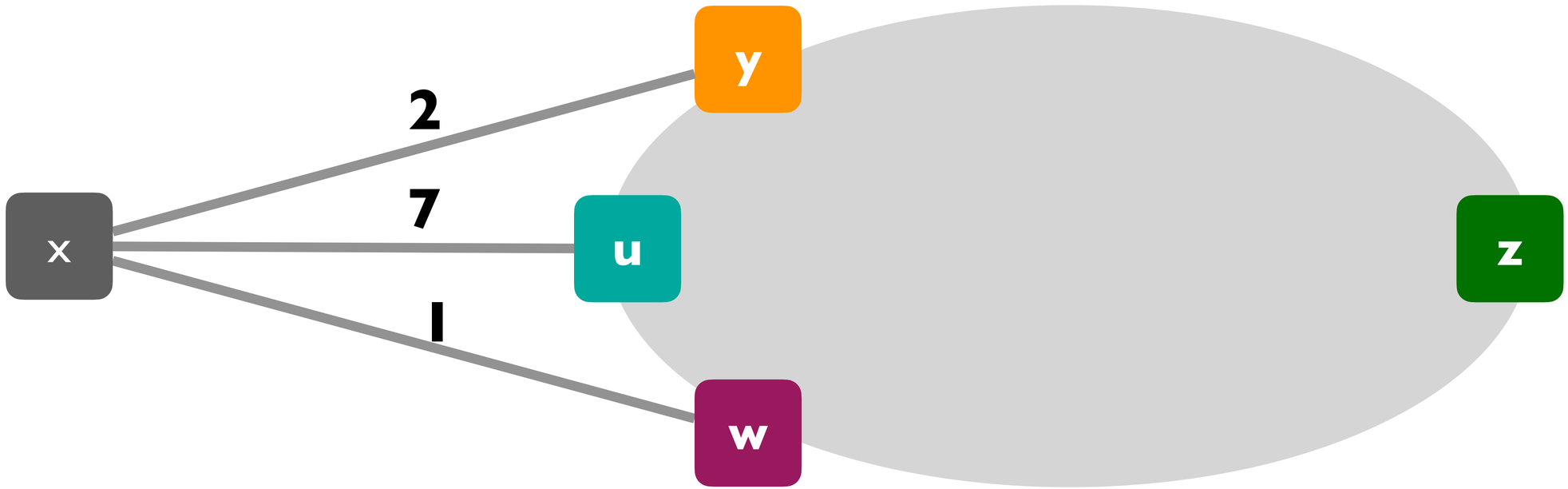

y
2
7
x
u
z
1
w



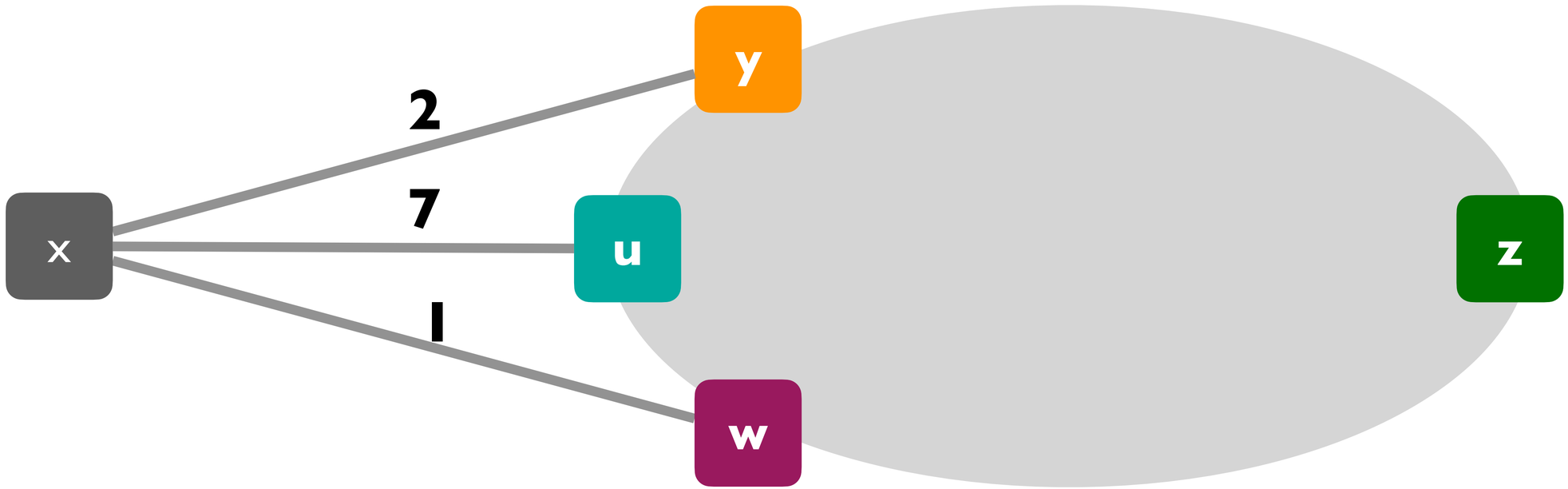
$$d_x(z) =$$



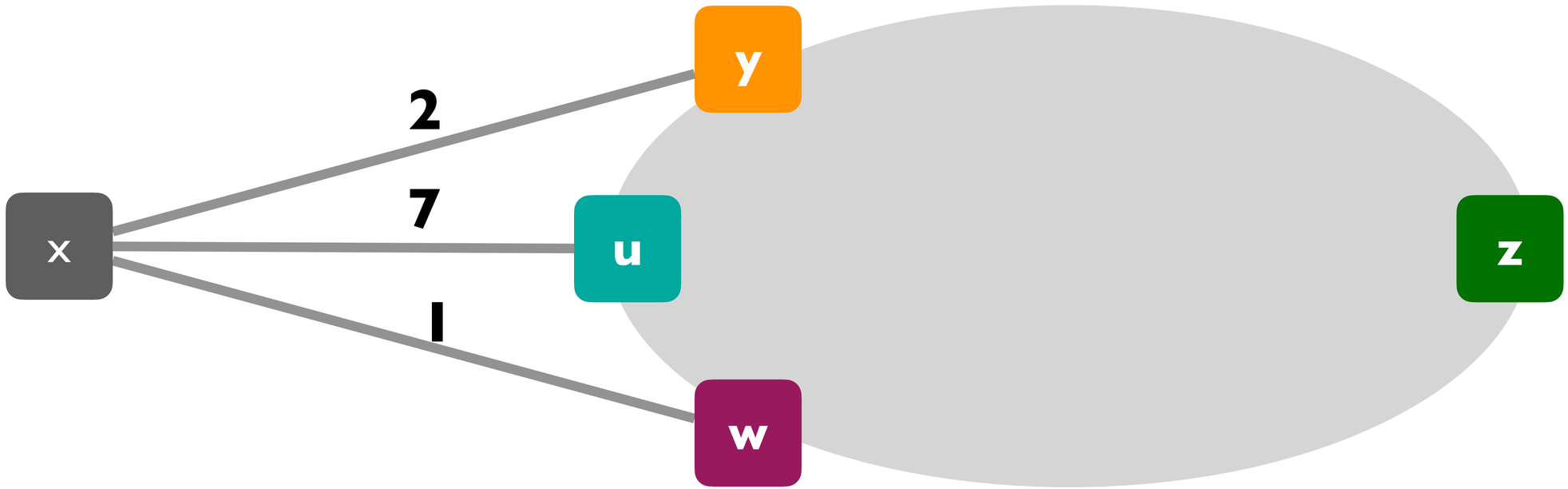
$$d_x(z) = \qquad cost(x, y) + d_y(z),$$



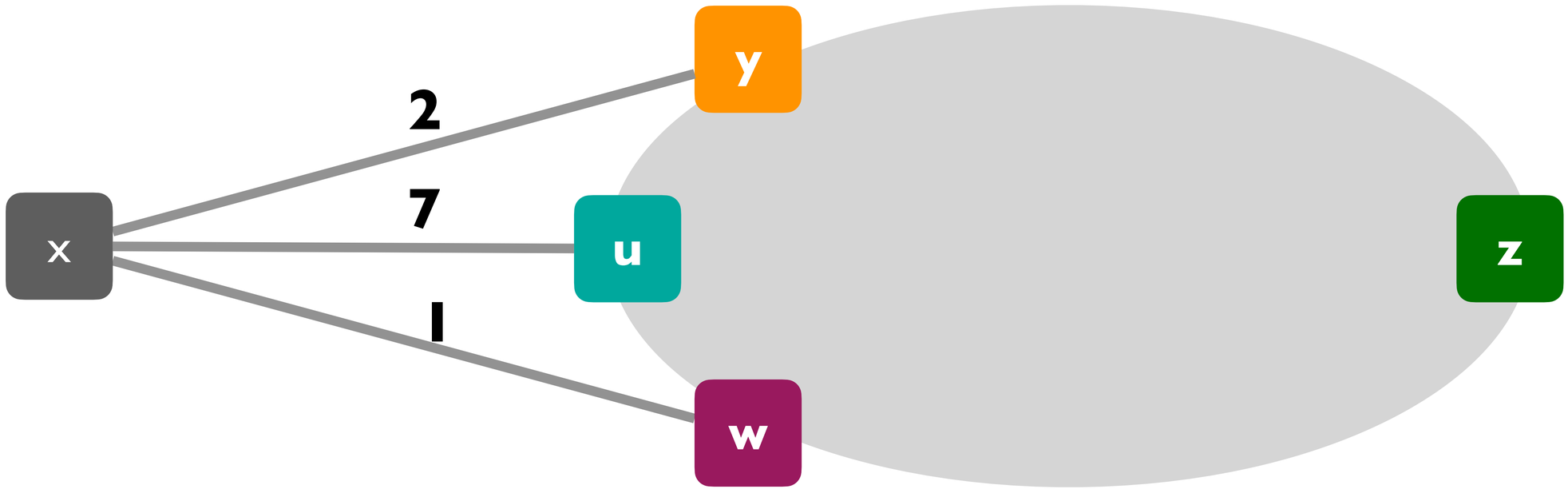
$$d_x(z) = \begin{array}{l} cost(x, y) + d_y(z), \\ cost(x, u) + d_u(z), \end{array}$$



$$d_x(z) = \begin{array}{l} cost(x, y) + d_y(z), \\ cost(x, u) + d_u(z), \\ cost(x, w) + d_w(z) \end{array}$$

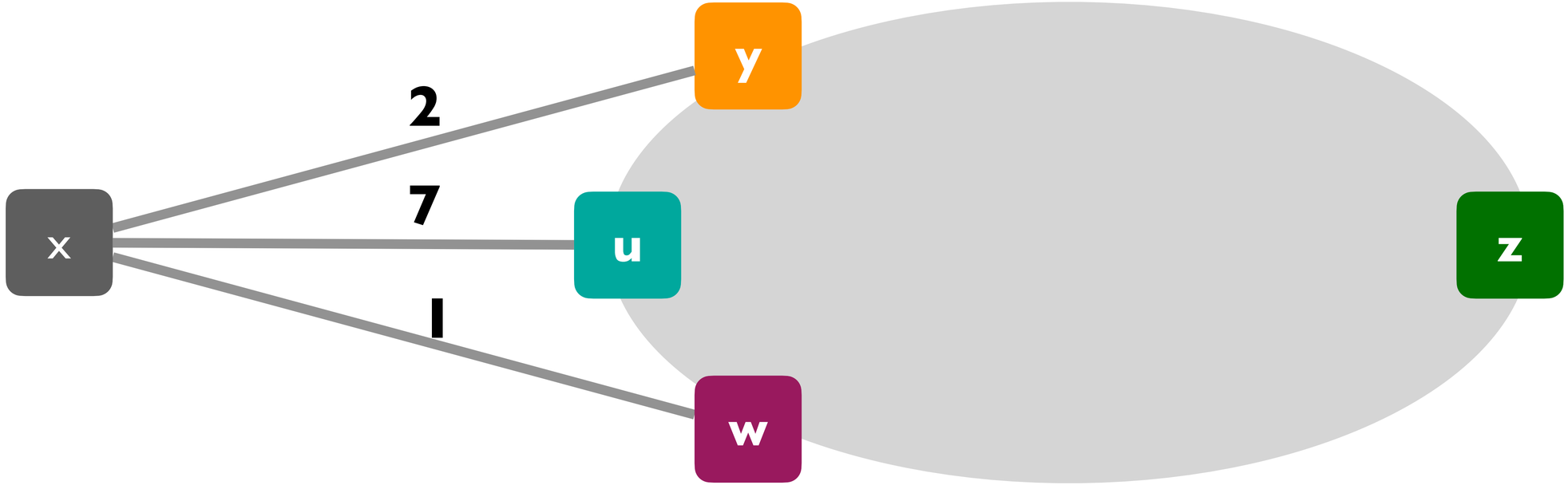


$$
\begin{aligned}
d_x(z) = \ \min\{ \ & cost(x, y) + d_y(z), \\
& cost(x, u) + d_u(z), \\
& cost(x, w) + d_w(z) \ \}
\end{aligned}
$$



$$d_x(z) = \min_n \{ \; cost(x, n) + d_n(z) \; \}$$

For all neighbors n



$$d_x(z) = \min_n \{ \; cost(x, n) + d_n(z) \; \}$$

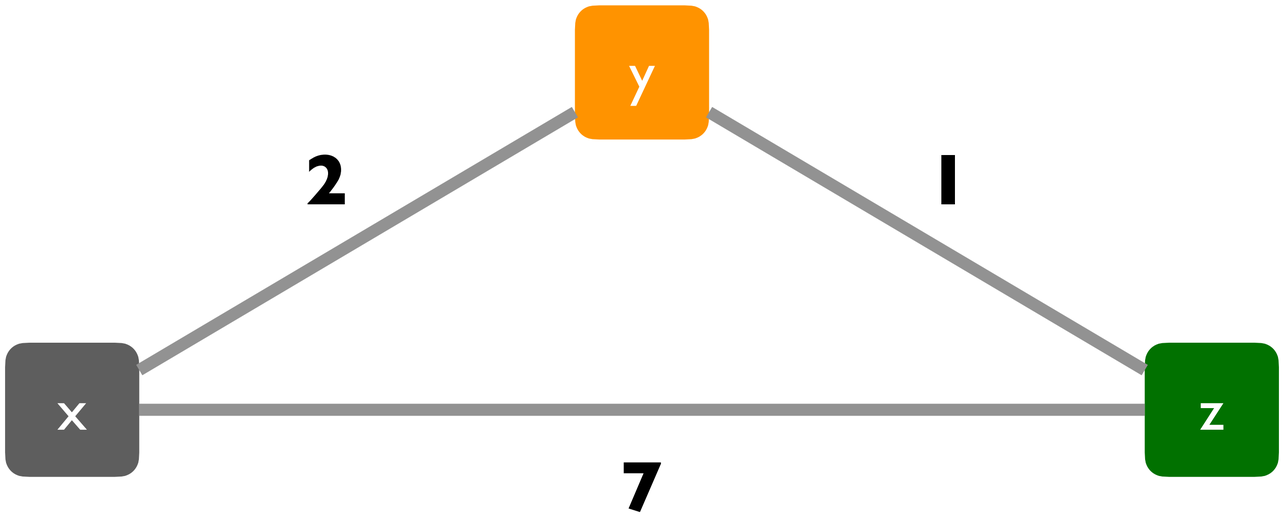
For all neighbors n

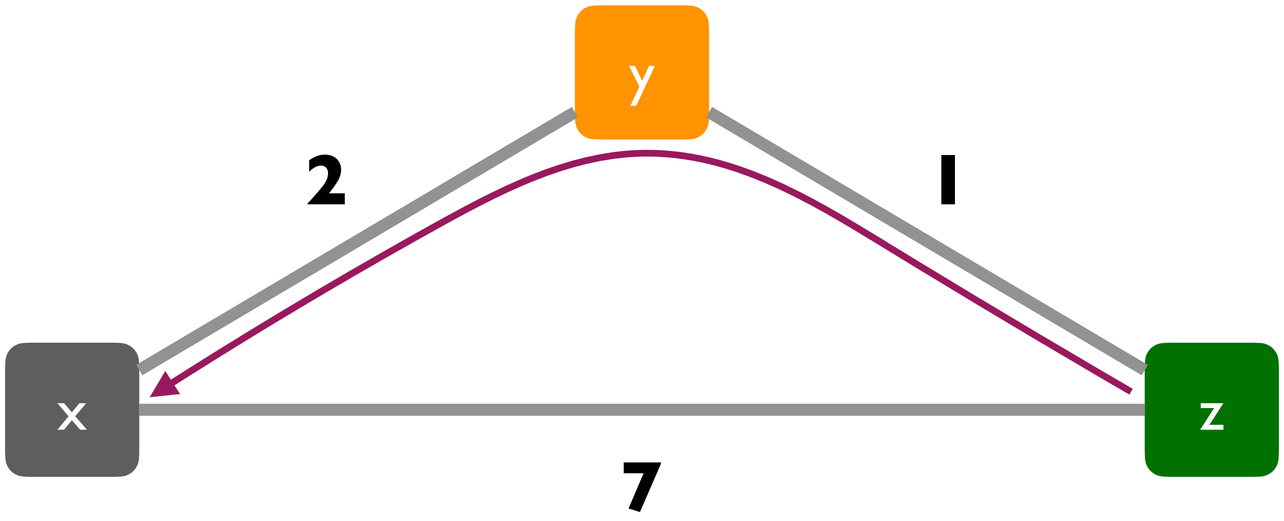
Bellman-Ford Equation

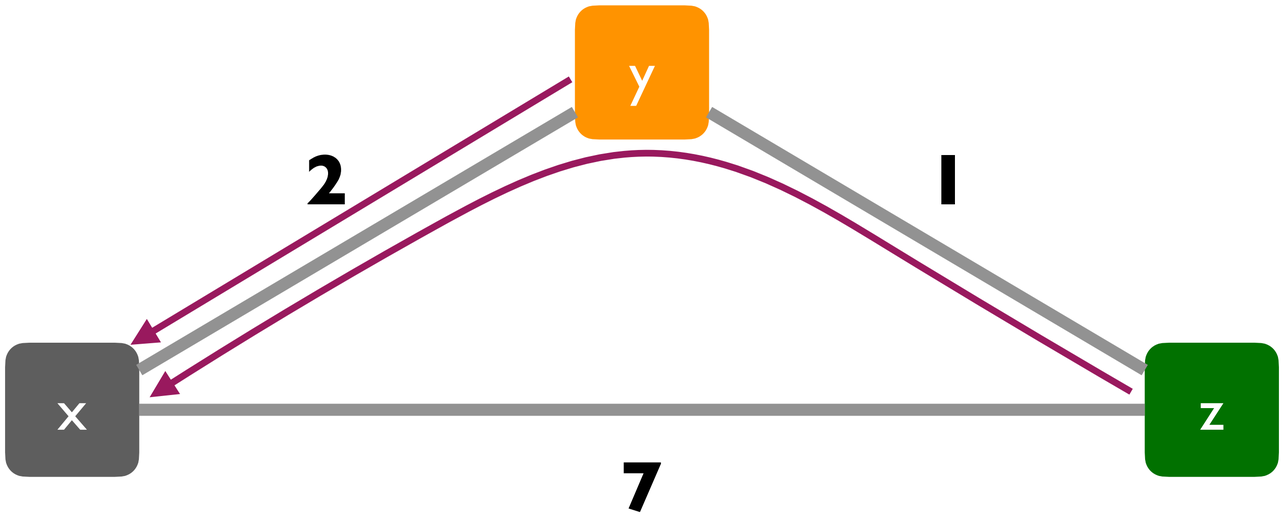
Bellman-Ford Equation

Bellman-Ford Equation

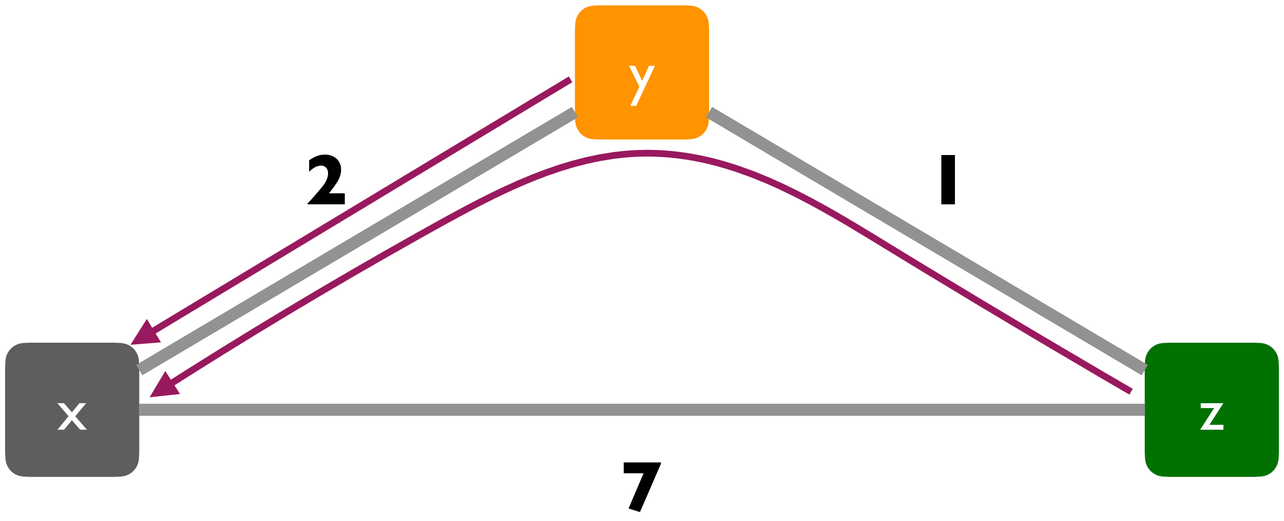
- Formalizes the following decision:
  - *Pick as the next hop for destination z*
    The neighbor that results
    in the least-cost path to z


y
2
1
x
z
7

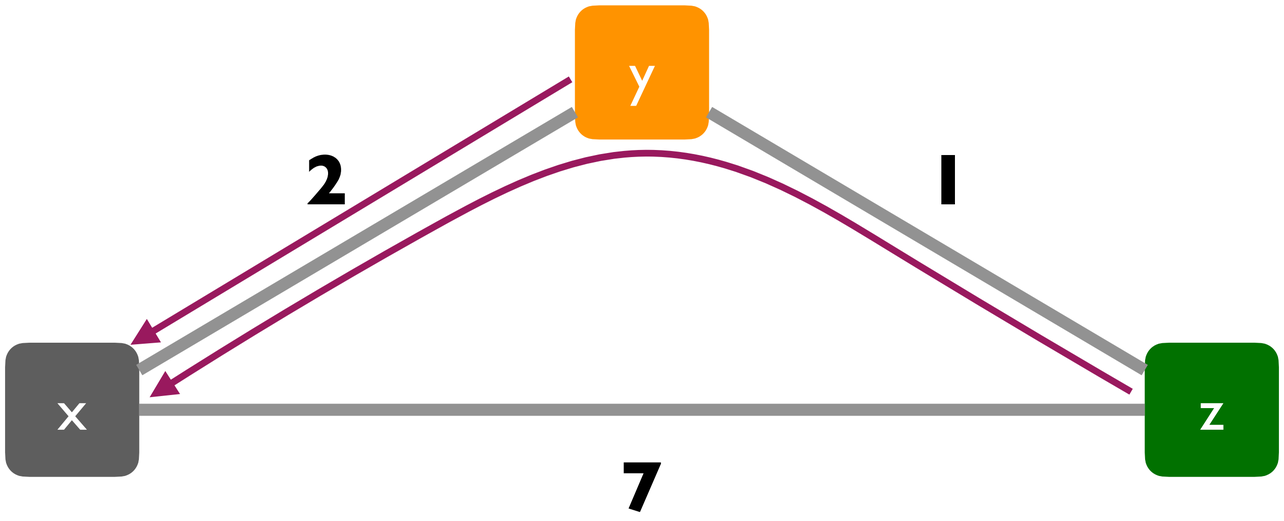

y
2
1
x
z
7


y
2
1
x
z
7

	x	y	z
y	2	0	1
z	3	1	0

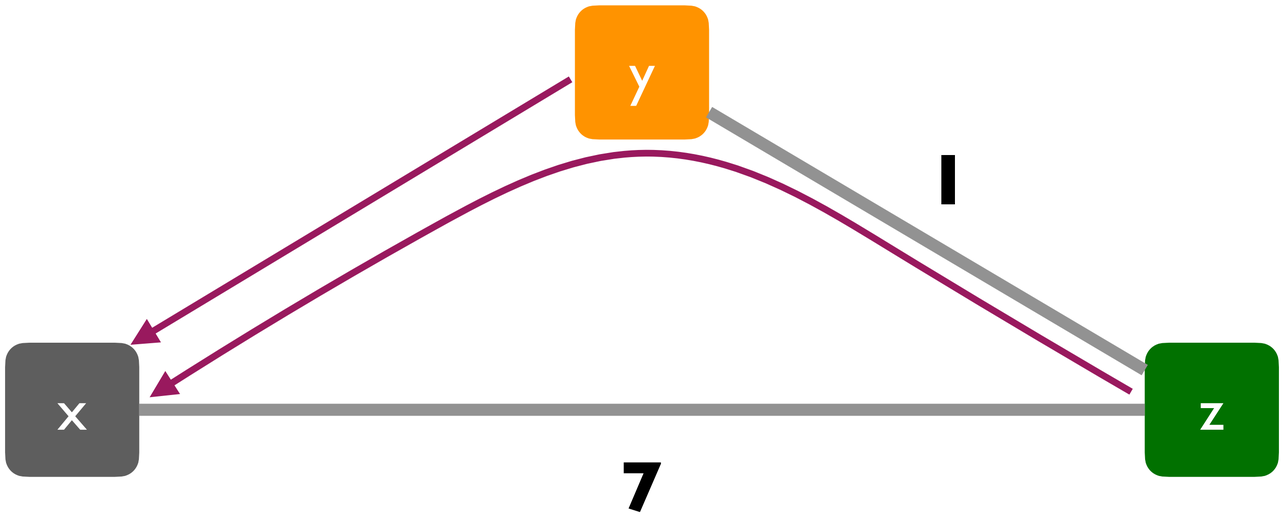


	x	y	z
y	2	0	1
z	3	1	0



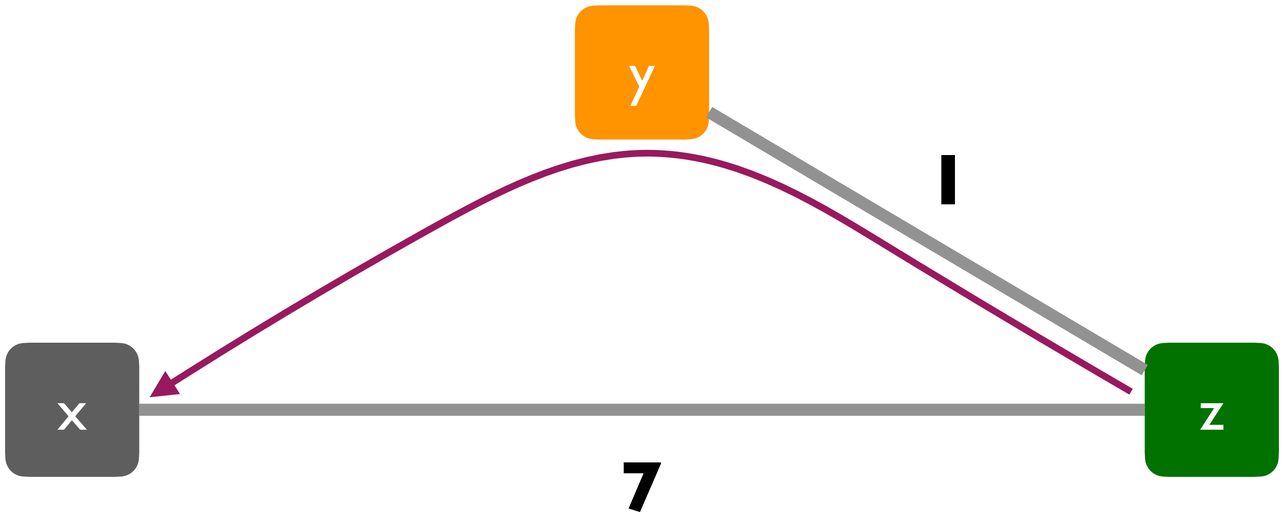
	x	y	z
y	2	0	1
z	3	1	0

	x	y	z
y	2	0	1
z	3	1	0



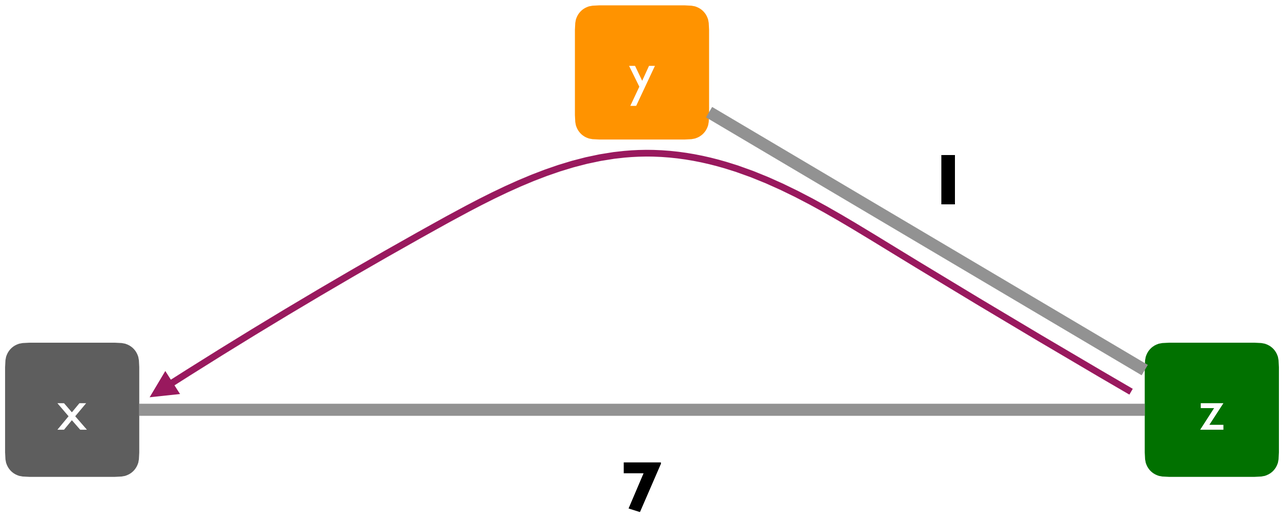
	x	y	z
y	2	0	1
z	3	1	0

	x	y	z
y		0	1
z	3	1	0



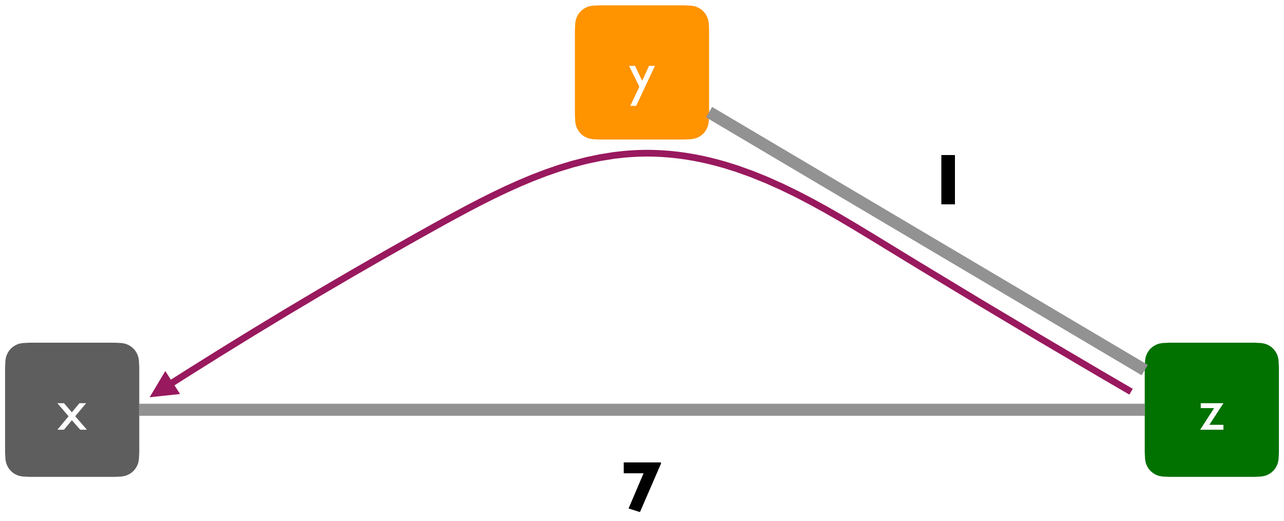
	x	y	z
y	2	0	1
z	3	1	0

	x	y	z
y		0	1
z	3	1	0



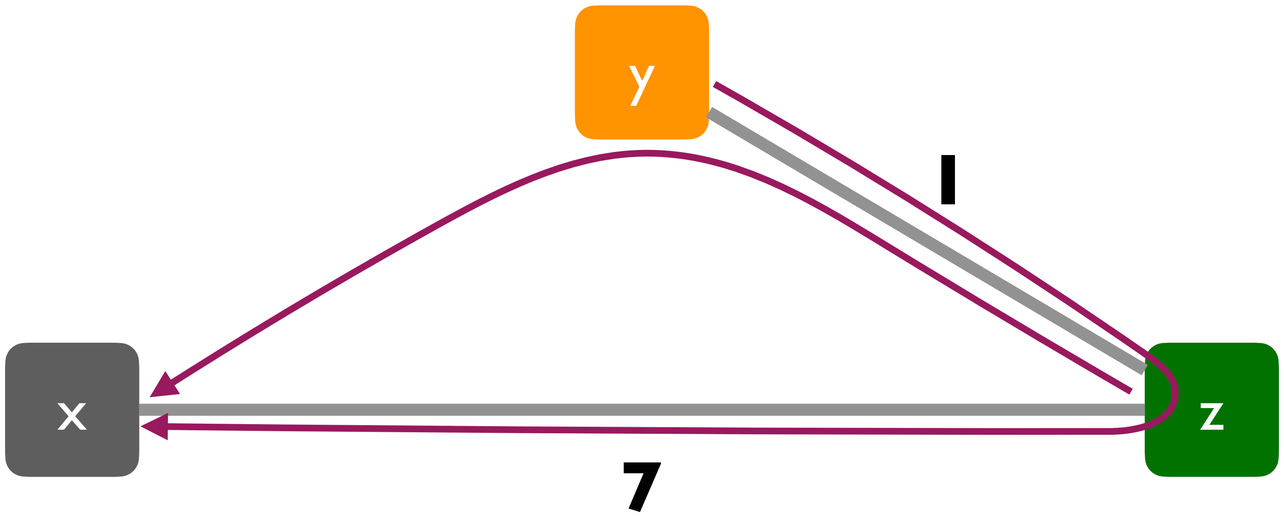
	x	y	z
y	2	0	1
z	3	1	0

	x	y	z
y	4	0	1
z	3	1	0



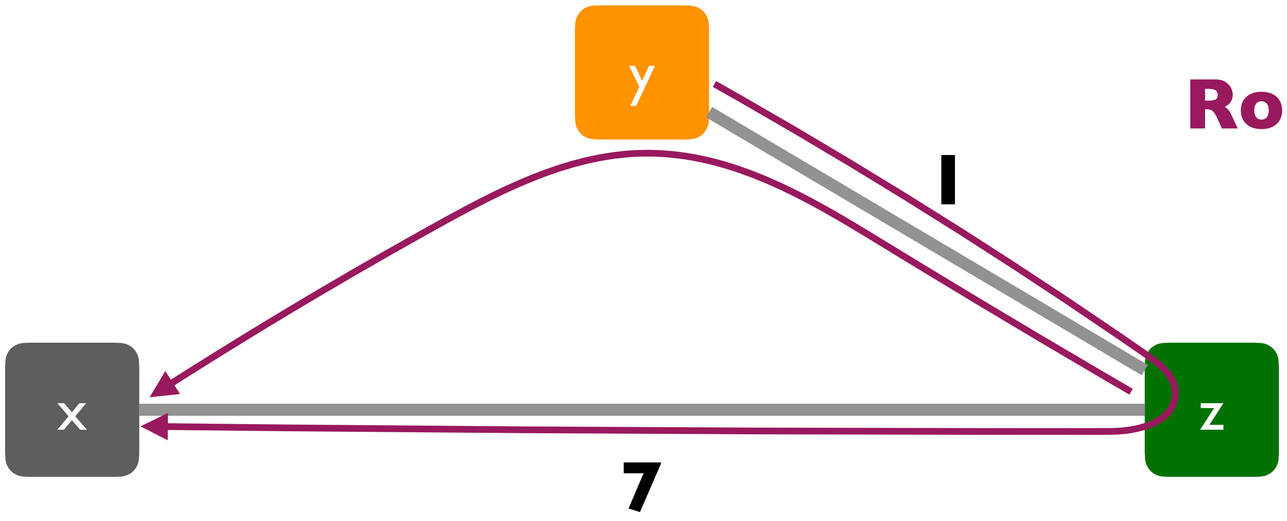
	x	y	z
y	2	0	1
z	3	1	0

	x	y	z
y	4	0	1
z	3	1	0



	x	y	z
y	2	0	1
z	3	1	0

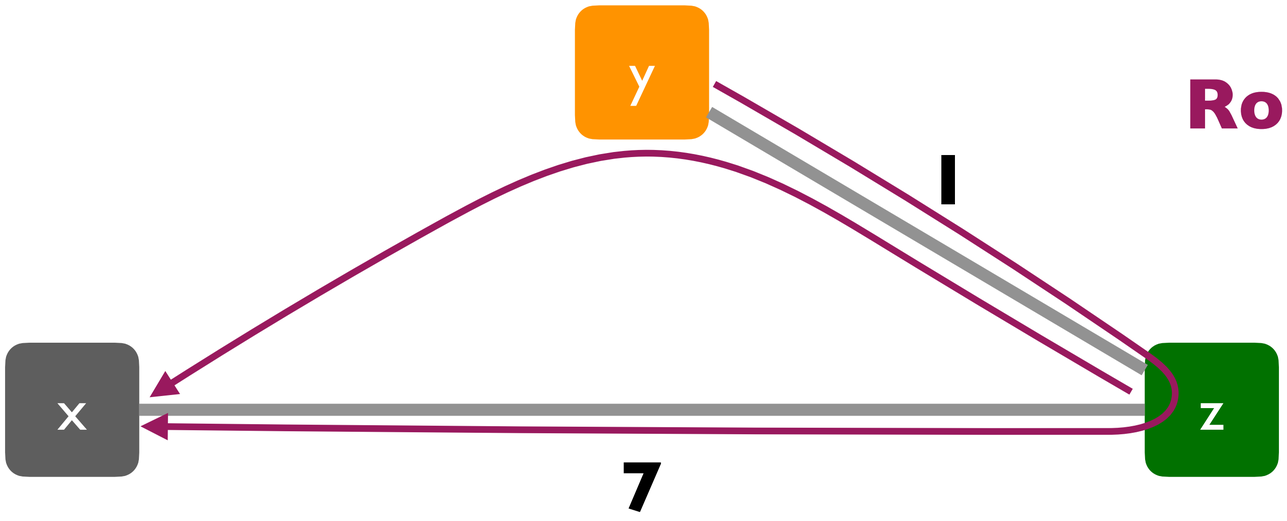
	x	y	z
y	4	0	1
z	3	1	0



Routing loop!

	x	y	z
y	2	0	1
z	3	1	0

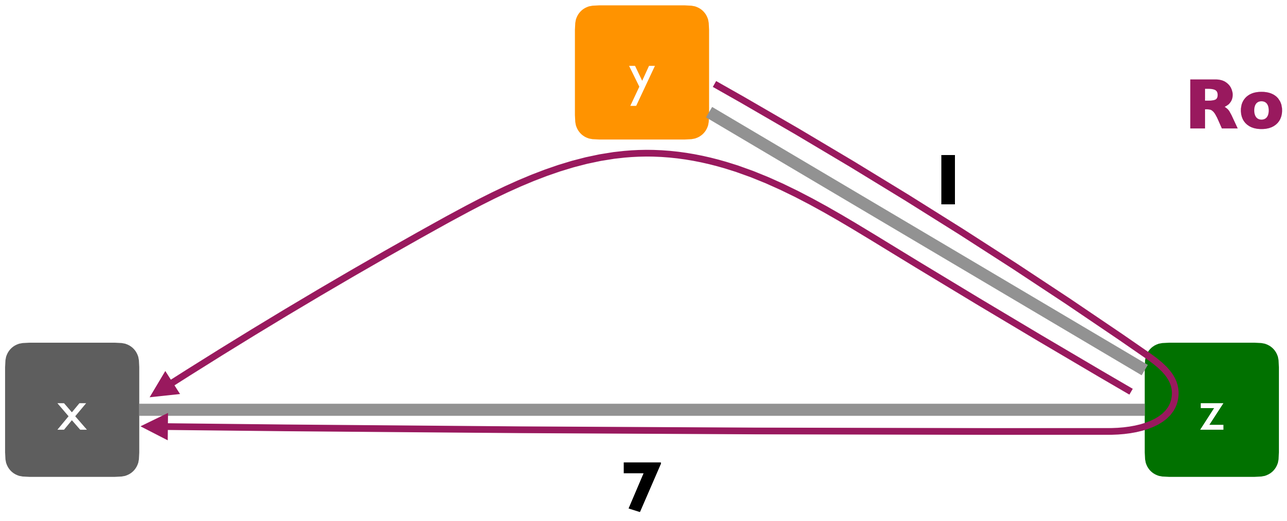
	x	y	z
y	4	0	1
z	3	1	0



Routing loop!

	x	y	z
y	4	0	1
z	3	1	0

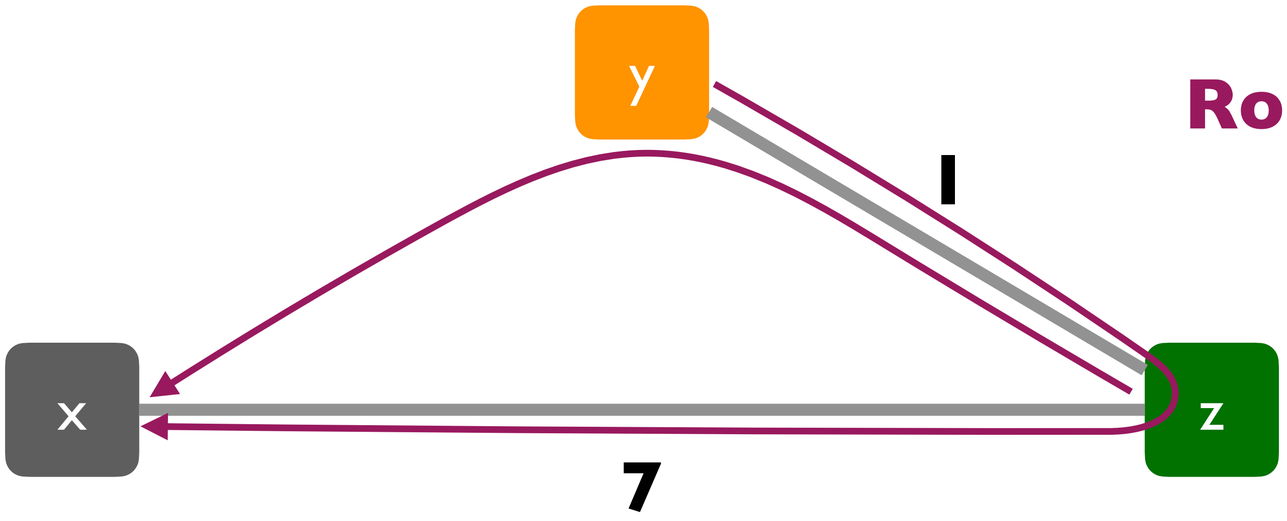
	x	y	z
y	4	0	1
z	3	1	0



Routing loop!

	x	y	z
y	4	0	1
z	3	1	0

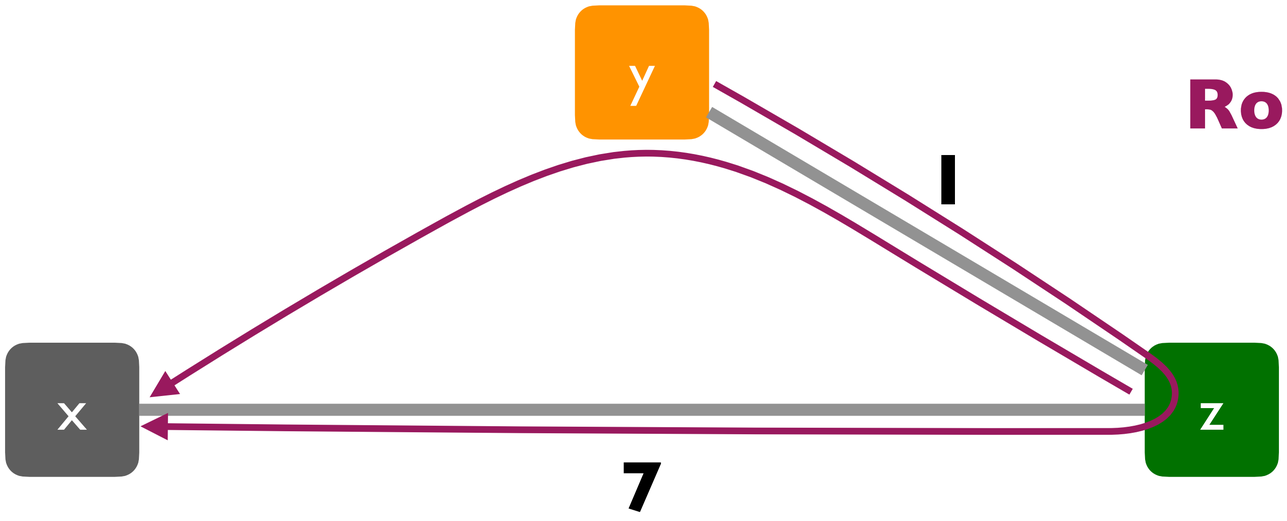
	x	y	z
y	4	0	1
z	3	1	0



Routing loop!

	x	y	z
y	4	0	1
z	5	1	0

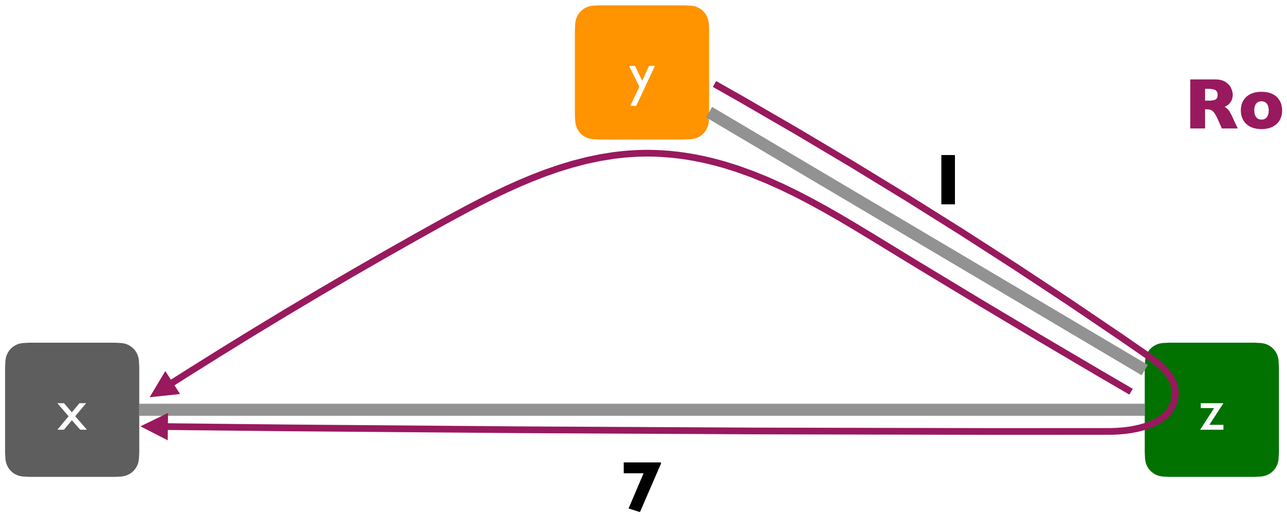
	x	y	z
y	4	0	1
z	5	1	0



Routing loop!

	x	y	z
y	4	0	1
z	5	1	0

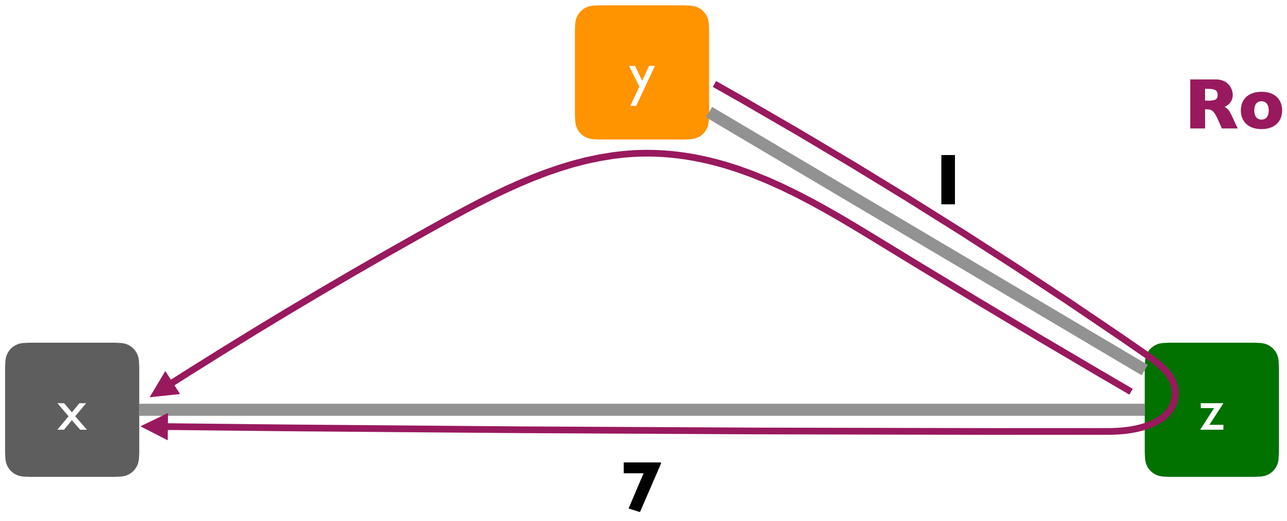
	x	y	z
y	4	0	1
z	5	1	0



Routing loop!

	x	y	z
y	4	0	1
z	5	1	0

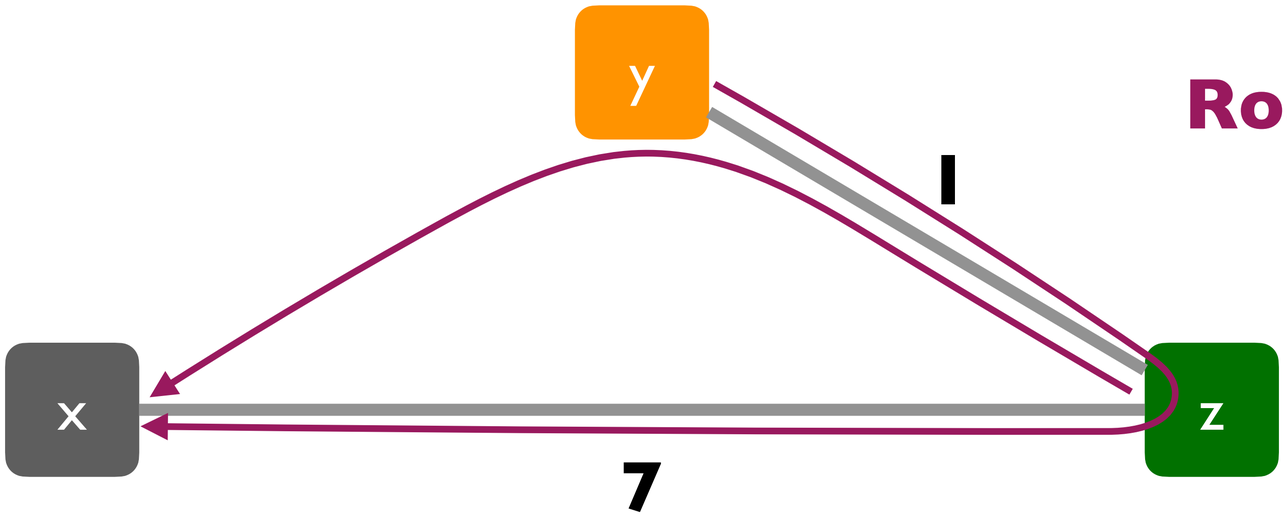
	x	y	z
y	6	0	1
z	5	1	0



Routing loop!

	x	y	z
y	4	0	1
z	5	1	0

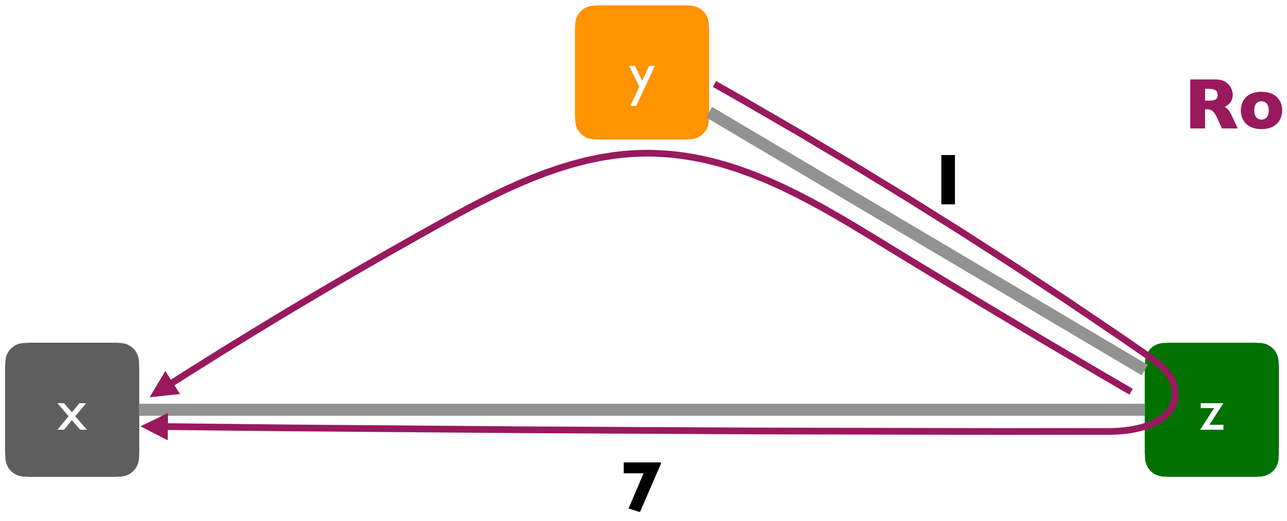
	x	y	z
y	6	0	1
z	5	1	0



Routing loop!

	x	y	z
y	6	0	1
z	5	1	0

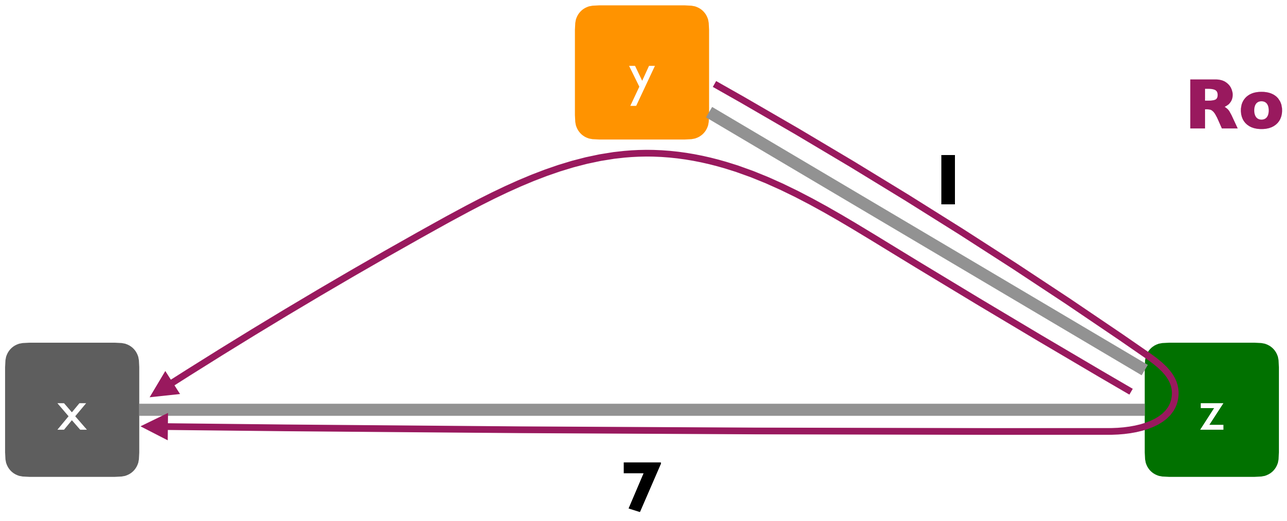
	x	y	z
y	6	0	1
z	5	1	0



Routing loop!

	x	y	z
y	6	0	1
z	5	1	0

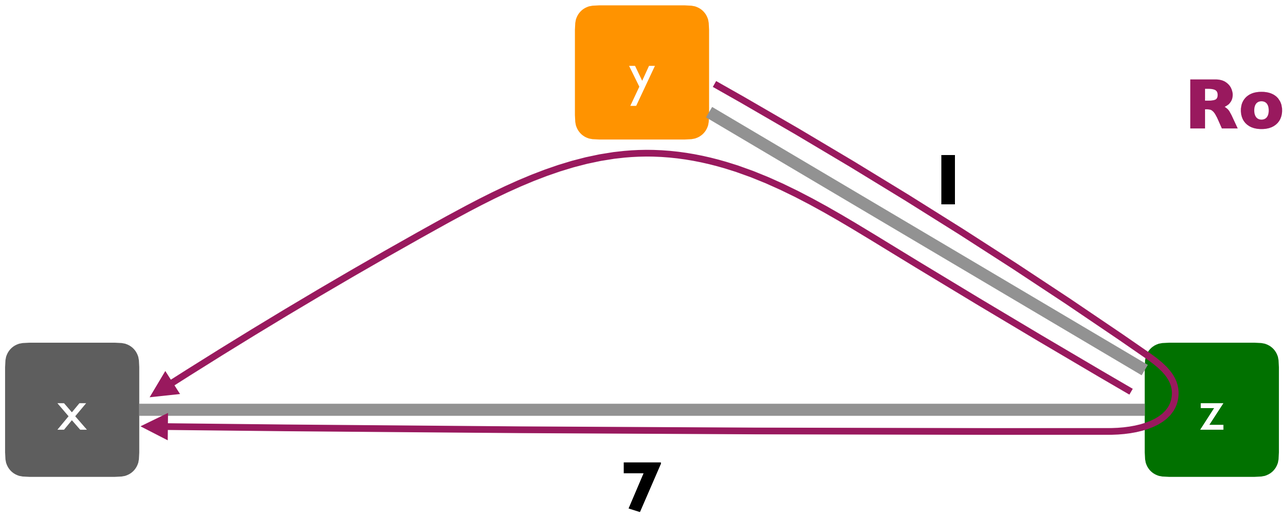
	x	y	z
y	6	0	1
z	5	1	0



Routing loop!

	x	y	z
y	6	0	1
z	7	1	0

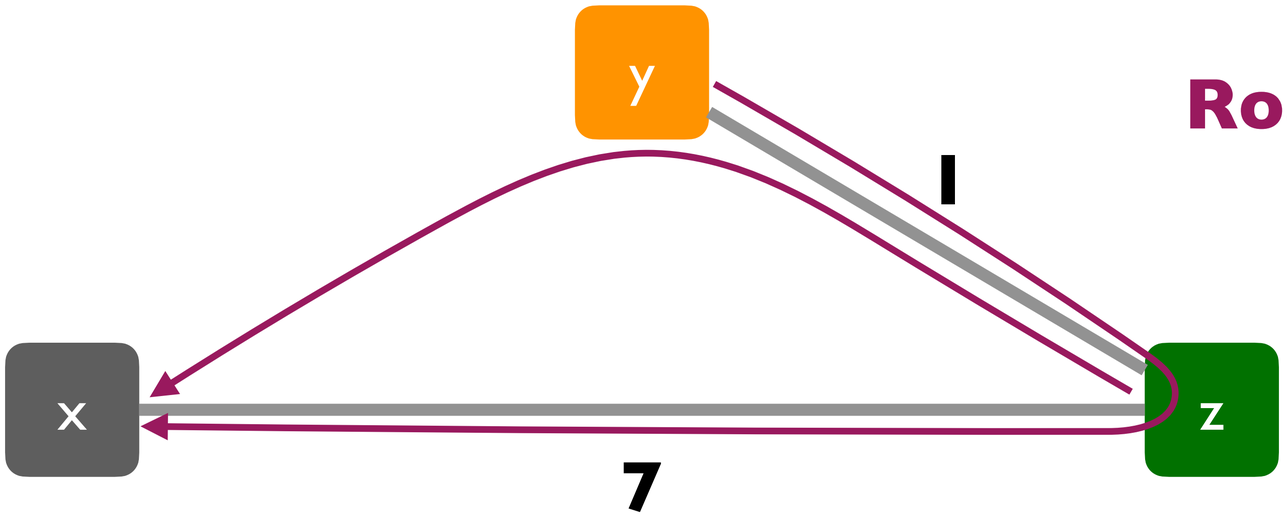
	x	y	z
y	6	0	1
z	7	1	0



Routing loop!

	x	y	z
y	6	0	1
z	7	1	0

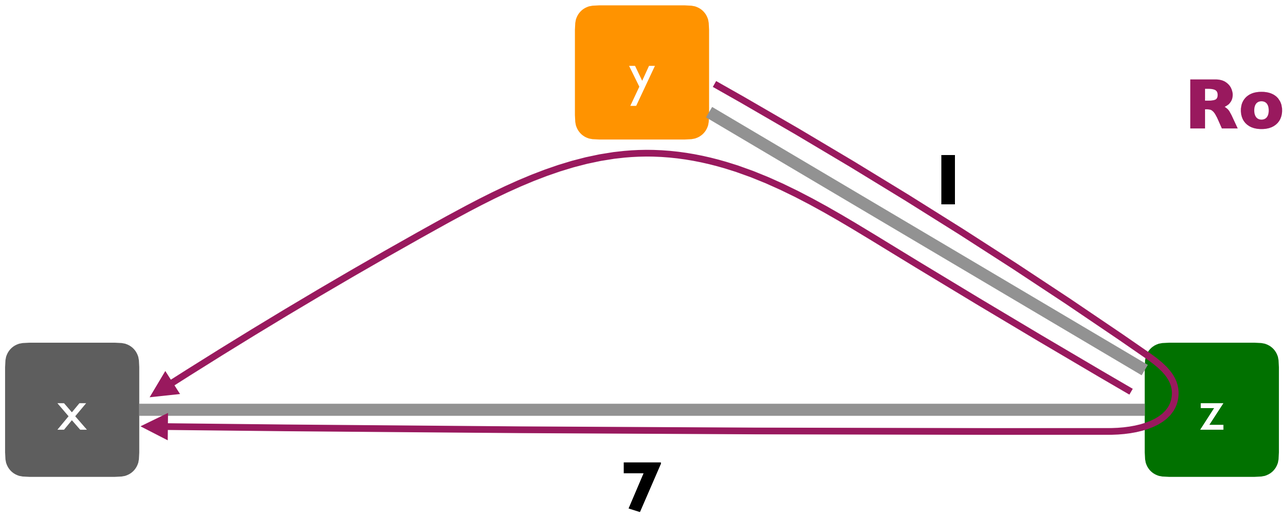
	x	y	z
y	6	0	1
z	7	1	0



Routing loop!

	x	y	z
y	6	0	1
z	7	1	0

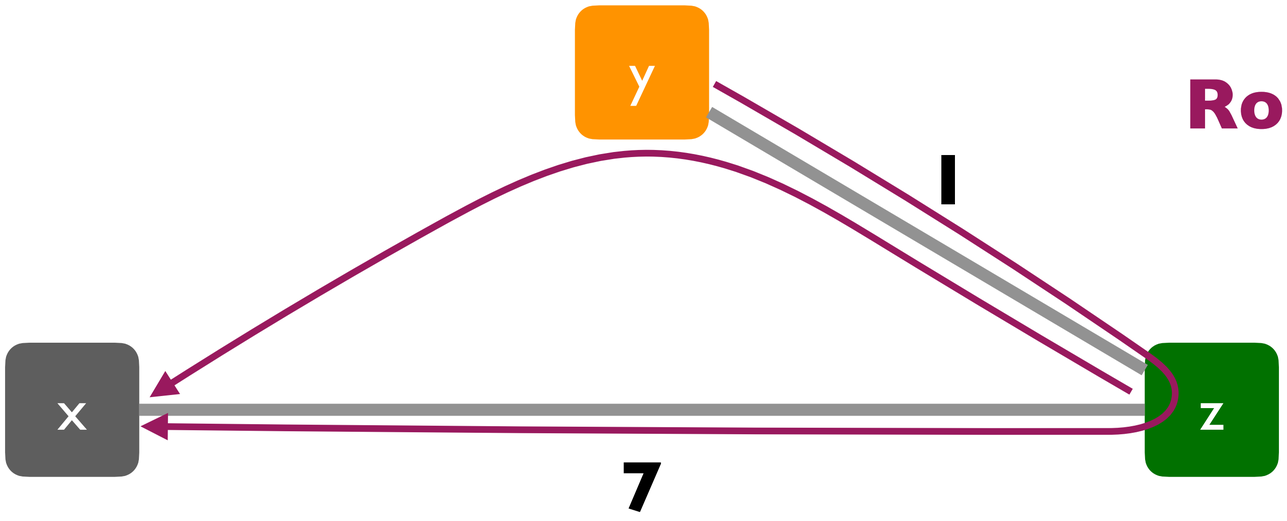
	x	y	z
y	8	0	1
z	7	1	0



Routing loop!

	x	y	z
y	6	0	1
z	7	1	0

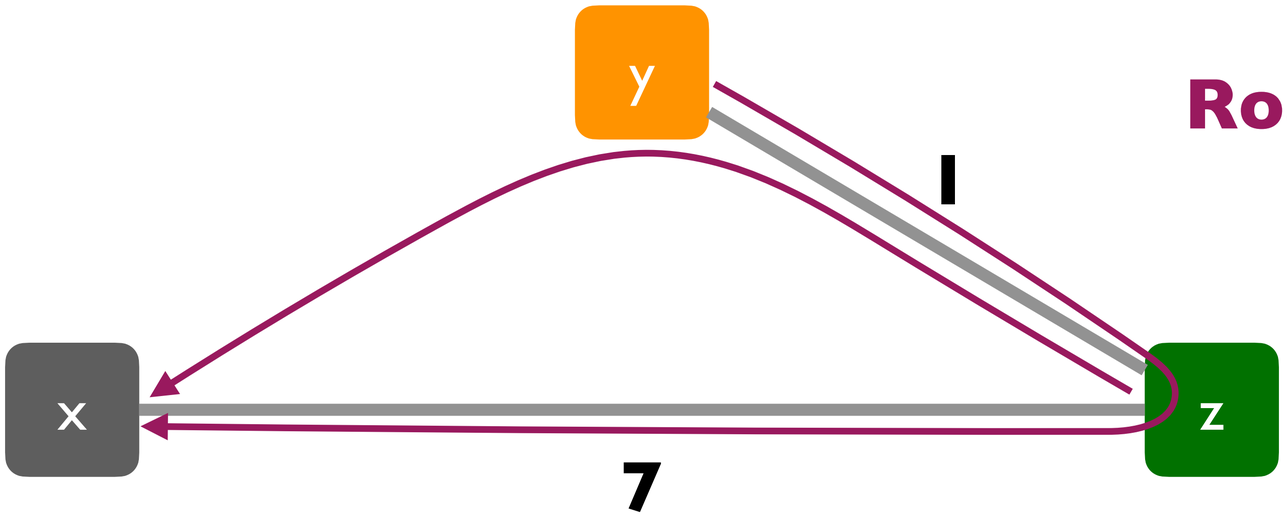
	x	y	z
y	8	0	1
z	7	1	0



Routing loop!

	x	y	z
y	8	0	1
z	7	1	0

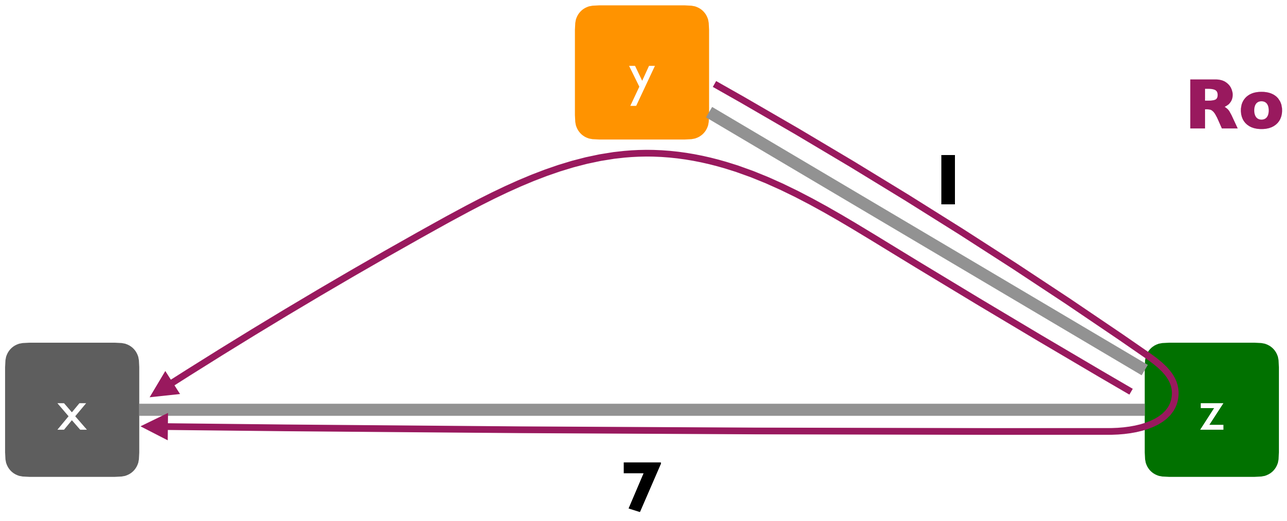
	x	y	z
y	8	0	1
z	7	1	0



Routing loop!

	x	y	z
y	8	0	1
z	7	1	0

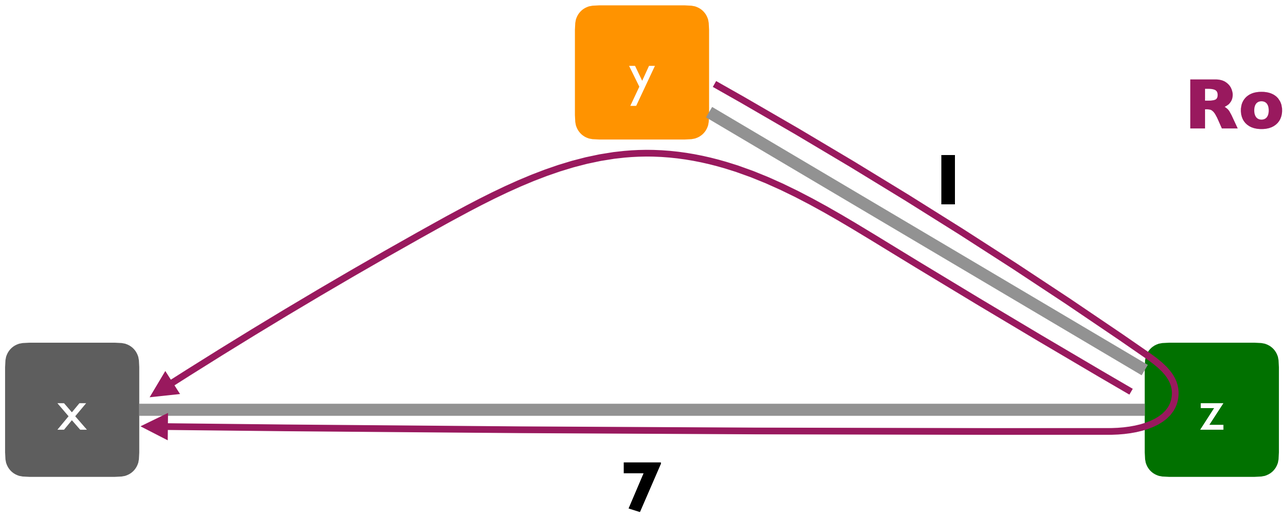
	x	y	z
y	8	0	1
z	7	1	0



Routing loop!

	x	y	z
y	8	0	1
z	7	1	0

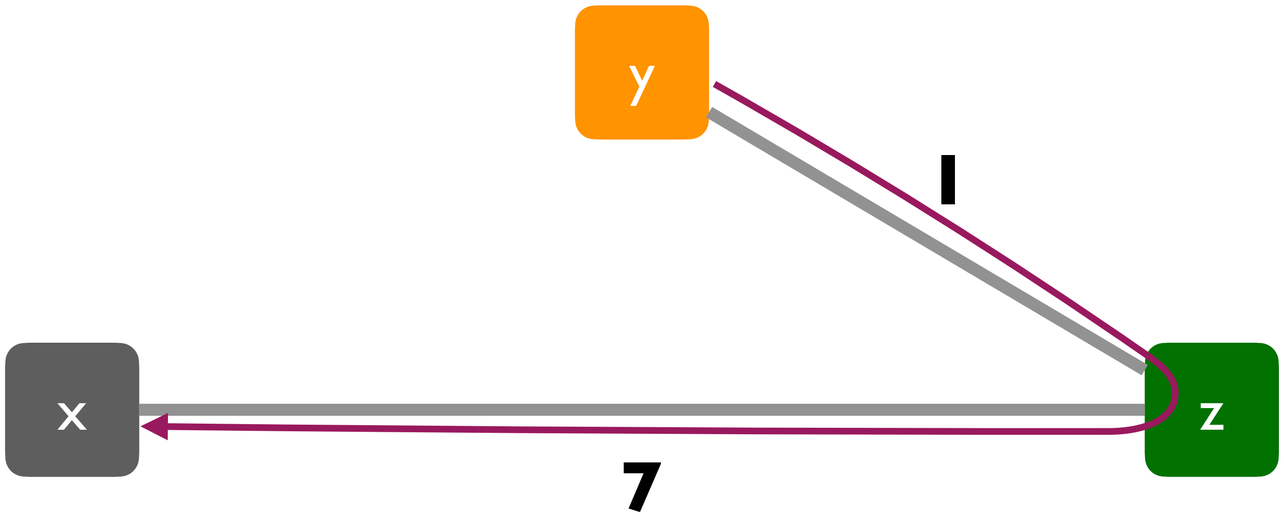
	x	y	z
y	8	0	1
z	7	1	0



Routing loop!

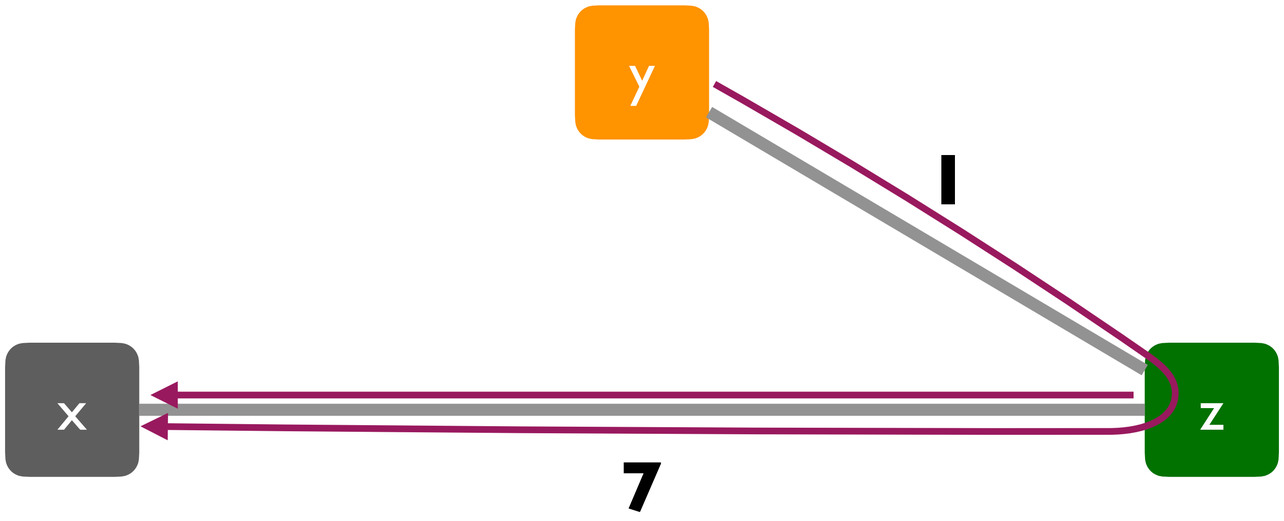
	x	y	z
y	8	0	1
z	7	1	0

	x	y	z
y	8	0	1
z	7	1	0



	x	y	z
y	8	0	1
z	7	1	0

	x	y	z
y	8	0	1
z	7	1	0



	x	y	z
y	8	0	1
z	7	1	0

Problems with Bellman-Ford

Problems with Bellman-Ford

- Routing loops
  - *z routes through y, y routes through x*
  - *y loses connectivity to x*
  - *y decides to route through z*

Problems with Bellman-Ford

- Routing loops
  - *z routes through y, y routes through x*
  - *y loses connectivity to x*
  - *y decides to route through z*
- Can take a very long time to resolve
  - *Count-to-infinity scenario*

Problems with Bellman-Ford

- Routing loops
  - *z routes through y, y routes through x*
  - *y loses connectivity to x*
  - *y decides to route through z*
- Can take a very long time to resolve
  - *Count-to-infinity scenario*
- Solution?
  - *Next time…*