Lecture 8:

The Network Layer

Katerina Argyraki, EPFL

Welcome back to Computer Networks. Today, we will continue our discussion on the network layer.

Outline

- Network-layer functions
  - forwarding
  - routing
- Network-layer types
  - virtual-circuit networks
  - datagram networks
- IP forwarding
- IP routing

Last week we discussed the two basic functions of a network layer: forwarding and routing.

Forwarding

- Local process that takes place at a router and determines output link for each packet
- How: read value from packet's network-layer header, search forwarding table for output link

Routing

- Network-wide process that populates forwarding tables
- How: routing algorithm run on a logically centralised network controller or the routers themselves

Outline

- Network-layer functions
  - forwarding
  - routing
- Network-layer types
  - virtual-circuit networks
  - datagram networks
- IP forwarding
- IP routing

Last week, will also discussed two basic types of network layers: virtual-circuit networks and datagram networks.

Outline

- Network-layer functions
  - forwarding
  - routing
- Network-layer types
  - virtual-circuit networks
  - datagram networks
- **IP forwarding**
- IP routing

Then we focused on the network layer of the Internet, IP, which is a datagram network, and we discussed IP forwarding.

IP forwarding

- Router's forwarding table maps
  IP prefixes to output links
- Reads destination IP address
  from packet's network-layer header
- Performs longest prefix matching
  - finds, in its forwarding table, the IP prefix
    that matches the dst IP address the best

We said that an IP router's forwarding table maps IP prefixes to output links.

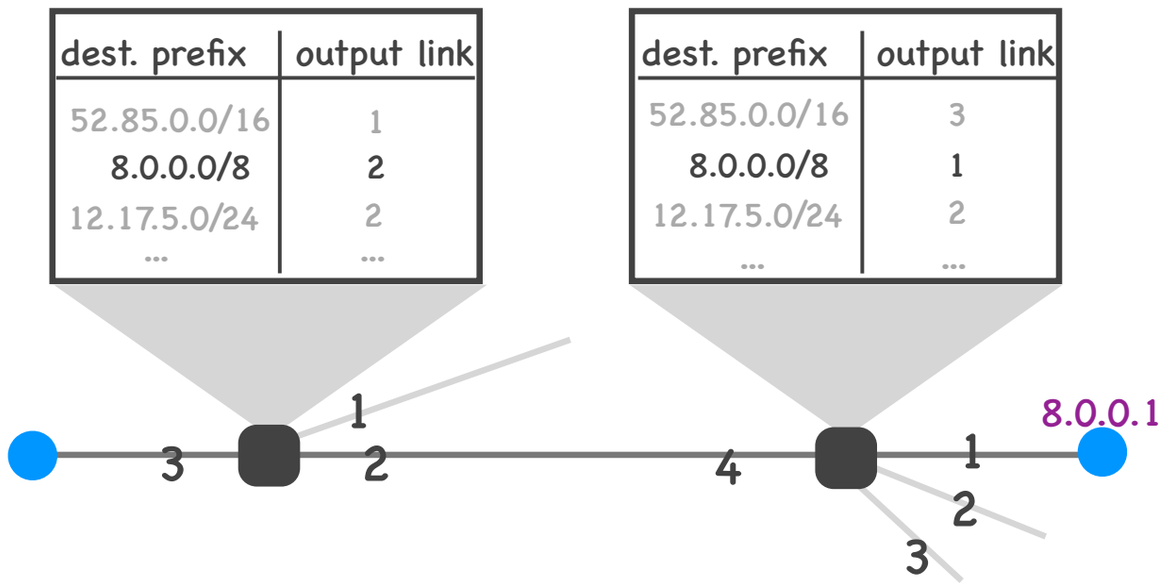
So, when a packet arrives, a router reads the destination IP address from the packet's network-layer header…

…and performs longest prefix matching, which means that it finds, in its forwarding table, the IP prefix that matches the packet's destination IP address the best.

Outline

- Network-layer functions
  - forwarding
  - routing
- Network-layer types
  - virtual-circuit networks
  - datagram networks
- IP forwarding
- **IP routing**

And now it's time to turn to IP routing.

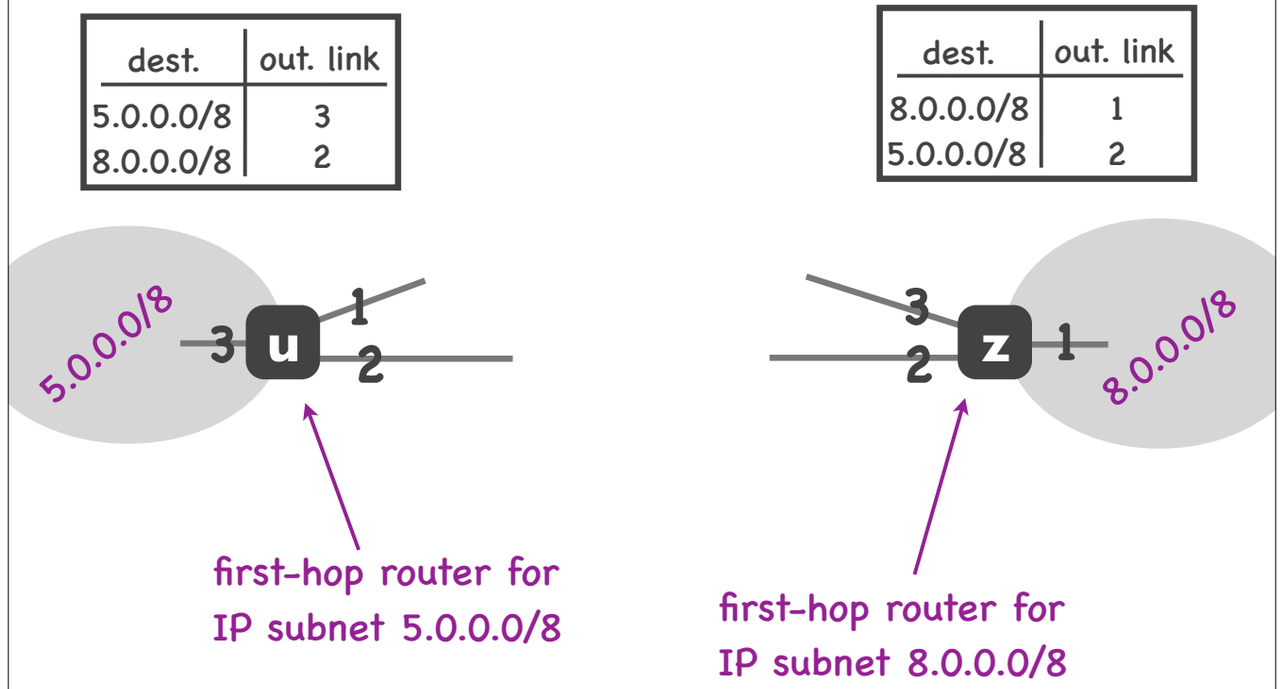


Consider a source and a destination end-system, the latter with IP address 8.0.0.1.

Every IP router on the Internet has, in its forwarding table, an IP prefix that matches 8.0.0.1 and indicates the correct output link for reaching that end-system.

(In fact, every IP router on the Internet has, in its forwarding table, an IP prefix that matches *any* public Internet IP address.)

The purpose of IP routing is to populate the forwarding tables of IP routers.



Suppose we have two IP subnets:
- the one on the left, with IP prefix 5.0.0.0/8,
- and the one on the right, with IP prefix 8.0.0.0/8.

Suppose each of these IP subnets has a single IP router: u and z, respectively.

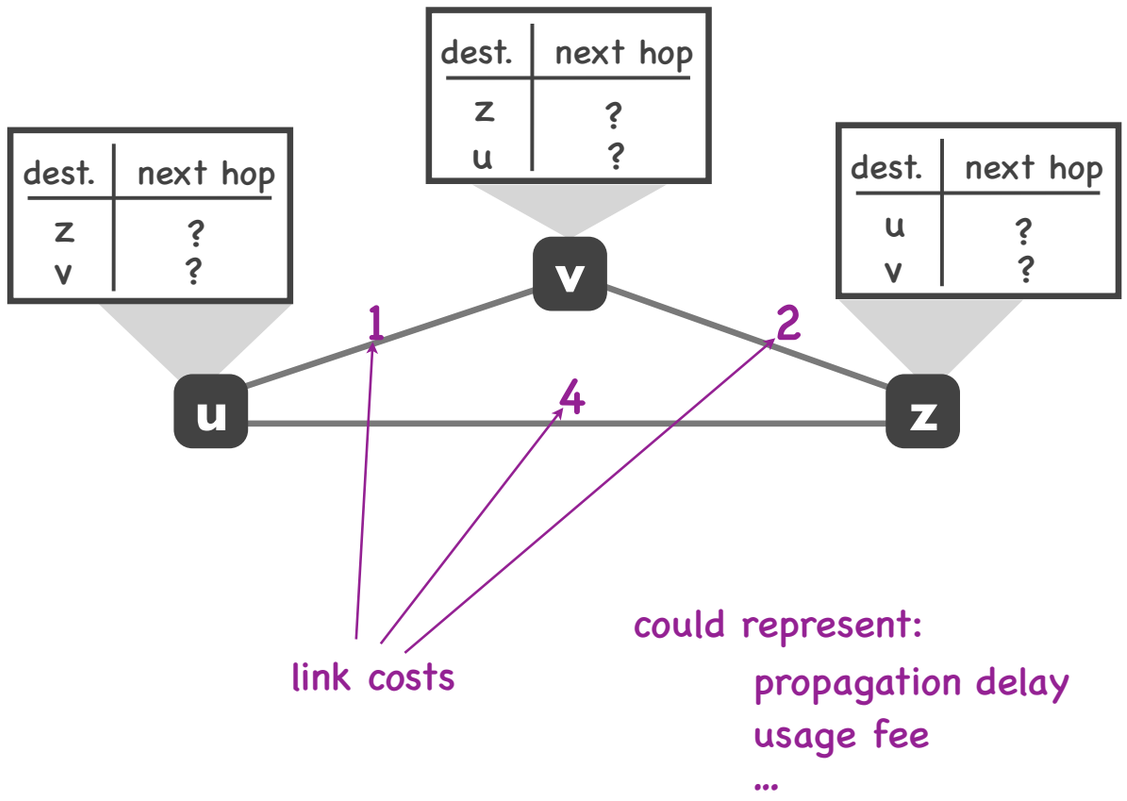
We say that router u is the “first-hop router” for the IP subnet on the left: it is the first router that handles packets coming from this subnet.

Similarly, router z is the “first-hop router” for the IP subnet on the right.

Each router knows how to forward packets addressed to its own subnet.
E.g., router u has, in its forwarding table, an entry, which says that, any packet with destination IP address matching 5.0.0.0/8 must be forwarded through output link 3 (which is connected to the router’s local subnet).
Similarly, router z has, in its forwarding table, an entry, which says that, any packet with destination IP address matching 8.0.0.0/8 must be forwarded through output link 1 (which is connected to the router’s local subnet).
These entries are typically setup when the network administrator configures each router.

But what about packets addressed to foreign subnets?
E.g., how does router u learn how to forward packets addressed to router z’s subnet and vice versa?
Who adds the second entry to each of our example forwarding tables?
An IP routing protocol.

Let’s forget about IP subnets for a moment.



Suppose routers u and z are part of a 3-router topology.

When these 3 routers participate in a routing protocol,
their goal is to learn the best path to each other.

For example, router u on the left, when it wants to send a packet to router z, how should it do that? Send the packet to z through the direct link that connects them? Or through router v? How about when it wants to send a packet to router v?

The other two routers, v and z, need to answer similar questions.

But what does “best path” mean?

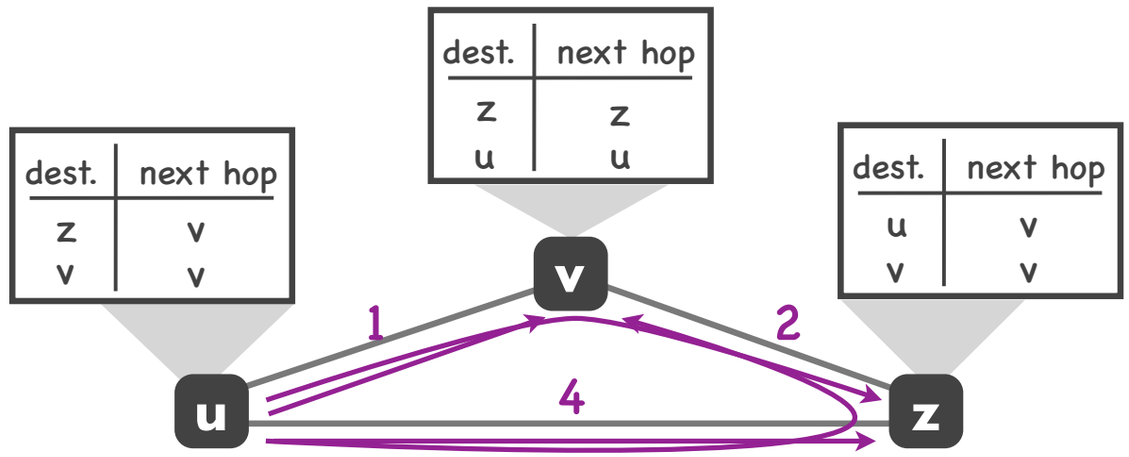
Each link has a cost, which represents the link’s “badness.” E.g., the cost could be computed based on the link’s propagation delay, or how much money it costs to send traffic over that link, or some other cost metric.

(Parenthesis: In all the examples I will show you, a link’s cost is the same in both directions. In reality, however, links can have different characteristics, hence different costs, in each direction.)

The cost of a path is equal to the sum of the costs of its links.

The best path from a source router to a destination router is the path with the least cost.

Btw, the data structures I show here are *not* forwarding tables; they are auxiliary data structures that I will use to illustrate various IP routing protocols.



least-cost path from u to z: u v z

least-cost path from u to v: u v

From router u to router z, there are two paths: a direct one, of cost 4, and an indirect one, through router v, of cost 1+2=3. The least-cost path is the indirect one.
Hence, from the point of view of router u, the best next hop for reaching router z is router v.

From router u to router v, there are also two paths: a direct one, of cost 1, and an indirect one, of cost 6. The least-cost path is the direct one.
Hence, from the point of view of router u, the best next hop for reaching router v is v itself.

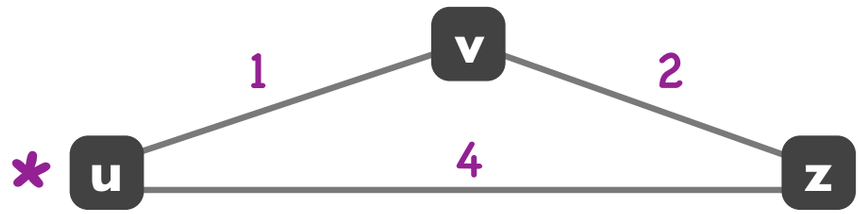
With similar reasoning, we can fill the tables of routers z and v.

Least-cost path routing

- Goal: find least-cost path
  from each source router to each
  destination router

So, the goal of least-cost path routing is to find the least-cost path (or least-cost route) from each source router to each destination router.

dest.	next hop	cost
z		
v		



It’s time to dive in and study a representative routing algorithm.

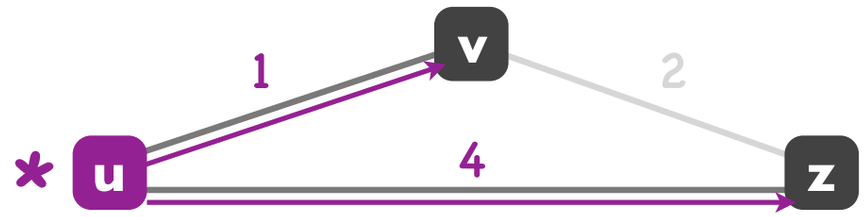
In the first algorithm we will see:
- first, *all* routers exchange information about their adjacent links;
- once each router has computed the network graph (with link costs), it runs this algorithm to compute the best path to each of the other routers.

So, each router runs its own separate instance of the algorithm. But given that they all provide the same input to the algorithm (the same network graph and link costs), they all reach compatible conclusions.

Let’s now focus on router u and see what happens when router u runs the algorithm.

The algorithm works in steps.

dest.	next hop	cost
z	z	4
v	v	1



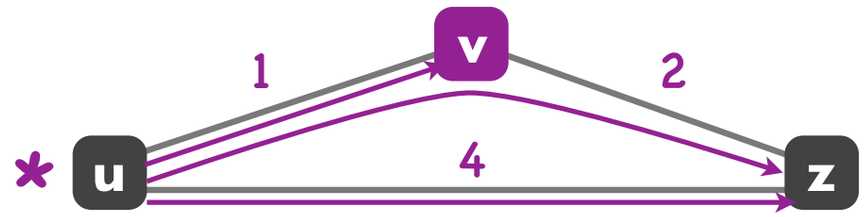
In the first step, the algorithm considers only the links that are directly connected to router u. It forgets about the rest of the network (it's as if the other links do not exist).

Considering *only* these two direct links,
the algorithm fills its table as follows:

- It asks: is there a path from router u to router z?
- The answer is: yes, there exists a direct path, the next-hop is router z itself, and the cost is 4.

- Then it asks: is there a path from router u to router v?
- The answer is: yes, there exists a direct path, the next-hop is router v itself, and the cost is 1.

dest.	next hop	cost
z	~~z~~ v	~~4~~ 3
v	v	1



In the second step, the algorithm chooses a neighbor of router u — in our example, router v, and considers the links that are directly connected to that neighbor as well.

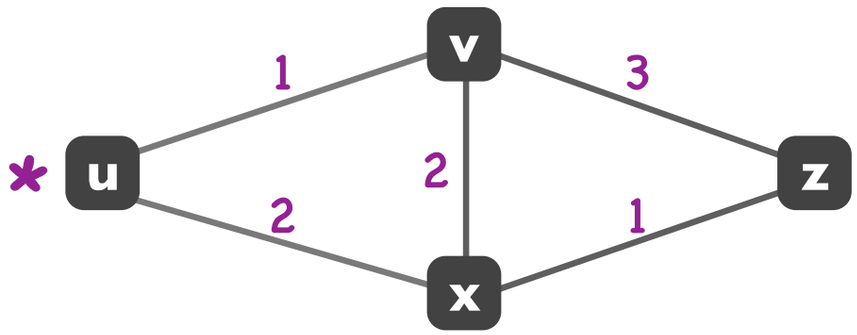
The algorithm asks: can router u improve its existing paths by routing through router v?

Let’s consider the path to destination z:
- Can router u improve its path to destination z by routing through v?
- Yes, indeed, it can.
- So, the algorithm removes the old, direct path from u to z and adds a new path through v.
- Now let’s update the table: for destination router z, the new next hop is router v, and the new cost is 3.

Now let’s consider the path to destination v:
- Can router u improve its path to destination v by routing through router v?
- No, because the path to destination v is already through router v itself, so there is nothing to improve.

At this point, the algorithm is done:
it has computed the least-cost path from router u to every other router in the network.

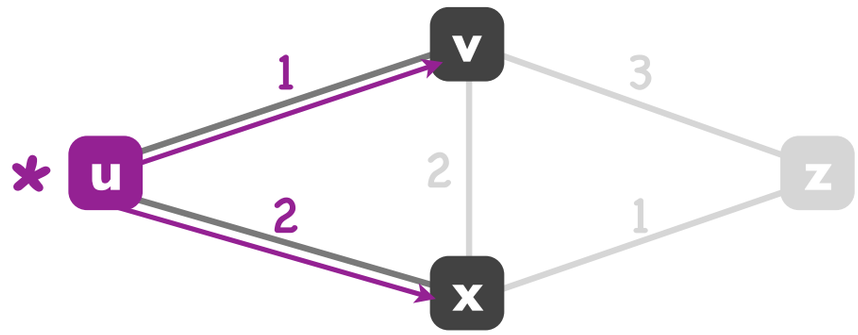
dest.	next hop	cost
z		
v		
x		



Let’s look at a second example.

Again, we will concentrate on router u.

dest.	next hop	cost
z	-	-
v	v	1
x	x	2



In the first step, the algorithm considers only the the links that are directly connected to router u.

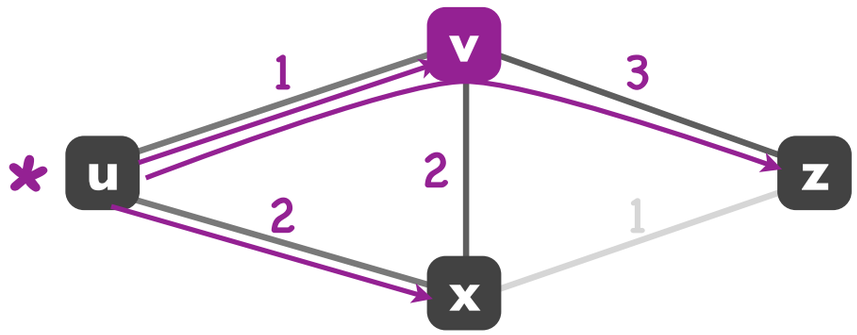
Considering only these 2 links:

- Router u cannot reach router z yet.

- Router u can reach router v through a direct link.
- The next hop is router v itself, and the cost is 1.

- Router u can reach router x through a direct link.
- The next hop is router x itself, and the cost is 2.

dest.	next hop	cost
z	~~-~~ v	~~-~~ 4
v	v	1
x	x	2



In the second step, the algorithm considers router v, and the links that are directly connected to router v.

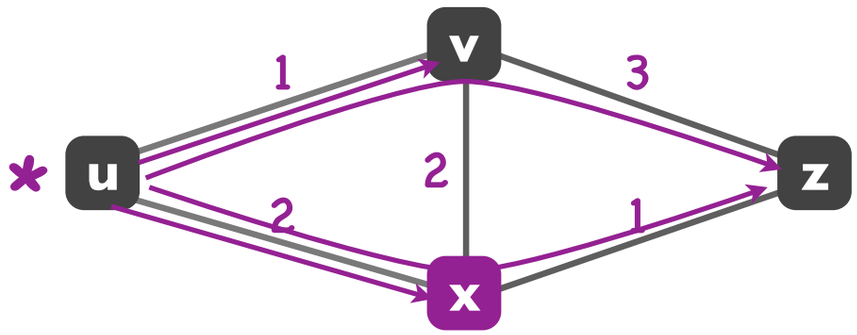
The algorithm asks: can router u improve its existing paths by routing through router v?

Let’s consider the path to destination router z:
- Can router u improve its path to destination z by routing through router v?
- Yes, it can: it previously could not reach router z at all, now it can reach router z through router v.
- So, the algorithm adds a new path through v.
- The next hop is router v, and the cost is 4.

Now let’s consider the path to destination router v:
- Can router u improve its path to destination v by routing through router v?
- No, because the path to destination router v is already through router v itself, so there is nothing to improve.

The same is true about the path to destination router x.

dest.	next hop	cost
z	~~-~~ ~~v~~ x	~~-~~ ~~4~~ 3
v	v	1
x	x	2



In the third step, the algorithm considers the other neighbor, router x,
and the links that are directly connected to router x.

The algorithm asks: can router u improve its existing paths by routing through router x?

Let’s consider the path to destination router z:
- Can router u improve its path to destination router z by routing through router x?
- Yes, it can, so the algorithm removes the old path through router v, and adds a new path through router x.
- The new next hop is router x, and the new cost is 3.

None of the other paths can be improved, so, the algorithm is done:
it has computed the least-cost path from router u to every other router in the network.

Link-state routing algorithm for source u

- Input: router graph & link costs
- Output: least-cost path from source router u to every other router

What we saw is an example of a link-state routing algorithm:

- It takes as input is the graph of all routers and the costs of all the links between them.
- It produces as output the least-cost path from a given source router to every other router in the network.

Link-state routing algorithm for source u

- "Centralized" algorithm: runs on a single entity
- Option #1: Router u runs the algorithm
- Option #2: Separate computer ("network controller") runs the algorithm for all the routers

A link-state routing algorithm can be characterised as a "centralized" algorithm, in the sense that, once an entity has the input (network graph and link costs), it runs the algorithm without any further communication.

One option is that that entity is the router itself, i.e., each router computes the least-cost path from itself to every other router in the network. This is what is typically done in most networks today.

Another option is that a separate computer, called a "network controller" runs the algorithm on behalf of all the routers, i.e., computes the least-coast path from every router to every other router in the network, and communicates the results to all the routers.

Dijkstra's algorithm

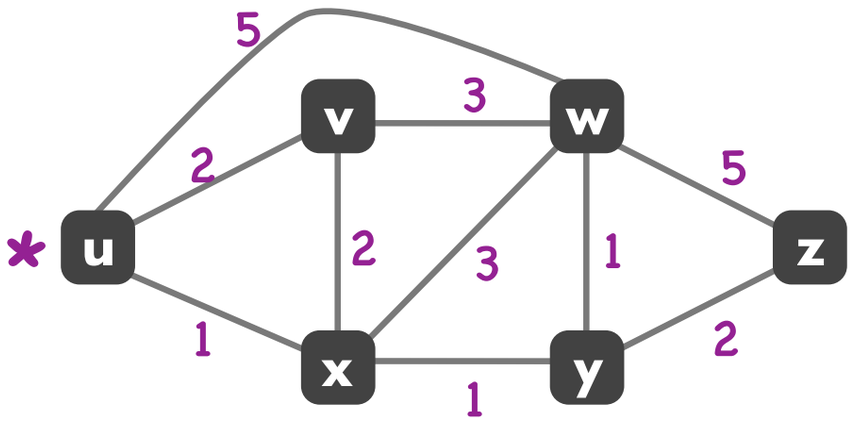
- At each step, consider a new router
  - starting from "closest" neighbor
- Check whether current paths can be improved
  - by using that router as an intermediate point
- End when no improvement is possible

The particular link-state routing algorithm that we discussed is called Dijstra's algorithm.

In summary:

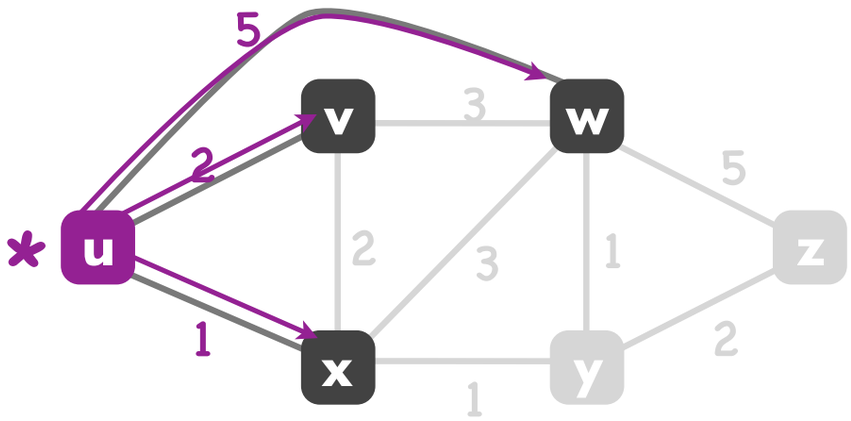
- At each step, the algorithm considers a new router.
- It checks whether it can improve the previously computed paths by routing through this new router.
- It ends when no improvement is possible.

dest.	next hop	cost
z		
w		
y		
v		
x		

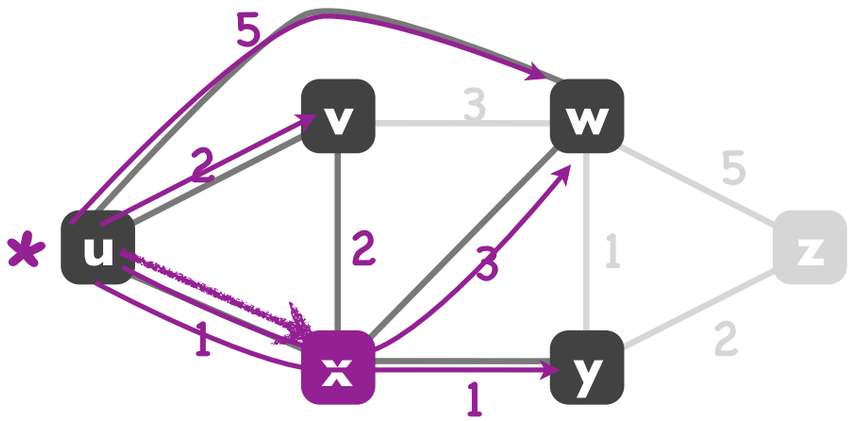


And one more example.
We always concentrate on the instance of the algorithm run by router u.

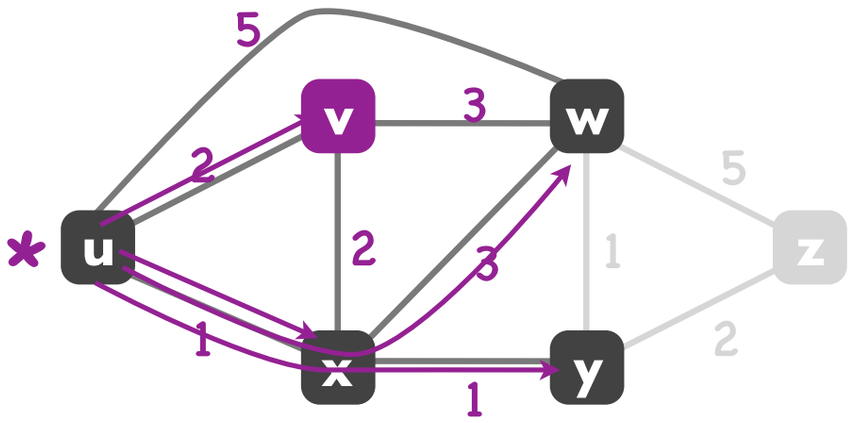
dest.	next hop	cost
z	-	-
w	w	5
y	-	-
v	v	2
x	x	1



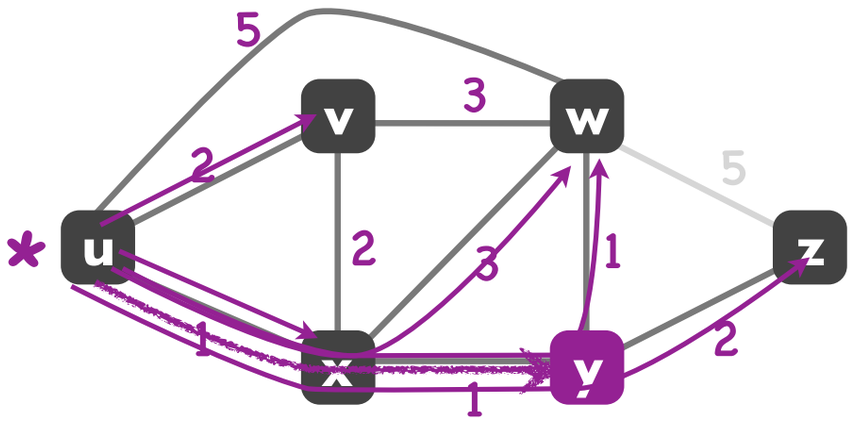
dest.	next hop	cost
z	-	-
w	~~w~~ x	~~5~~ 4
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



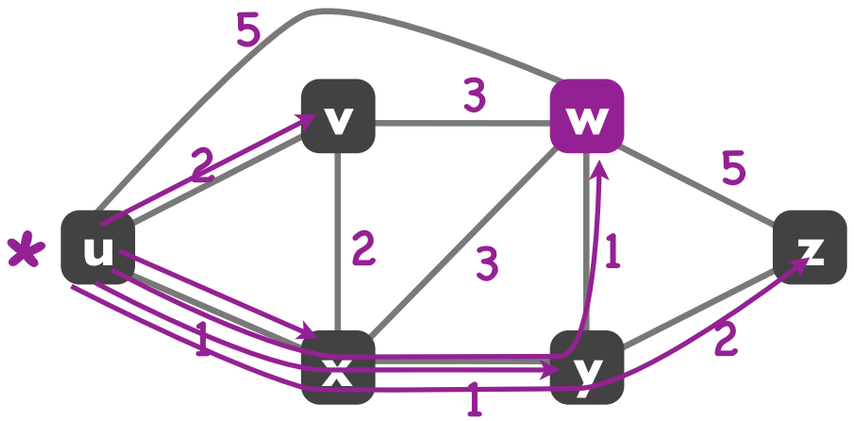
dest.	next hop	cost
z	-	-
w	~~w~~ x	~~5~~ 4
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



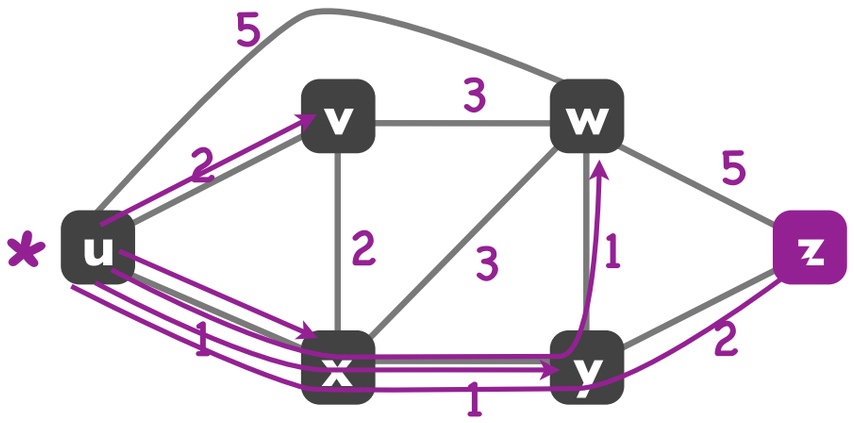
dest.	next hop	cost
z	~~-~~ x	~~-~~ 4
w	~~w~~ x	~~5~~ ~~4~~ 3
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1

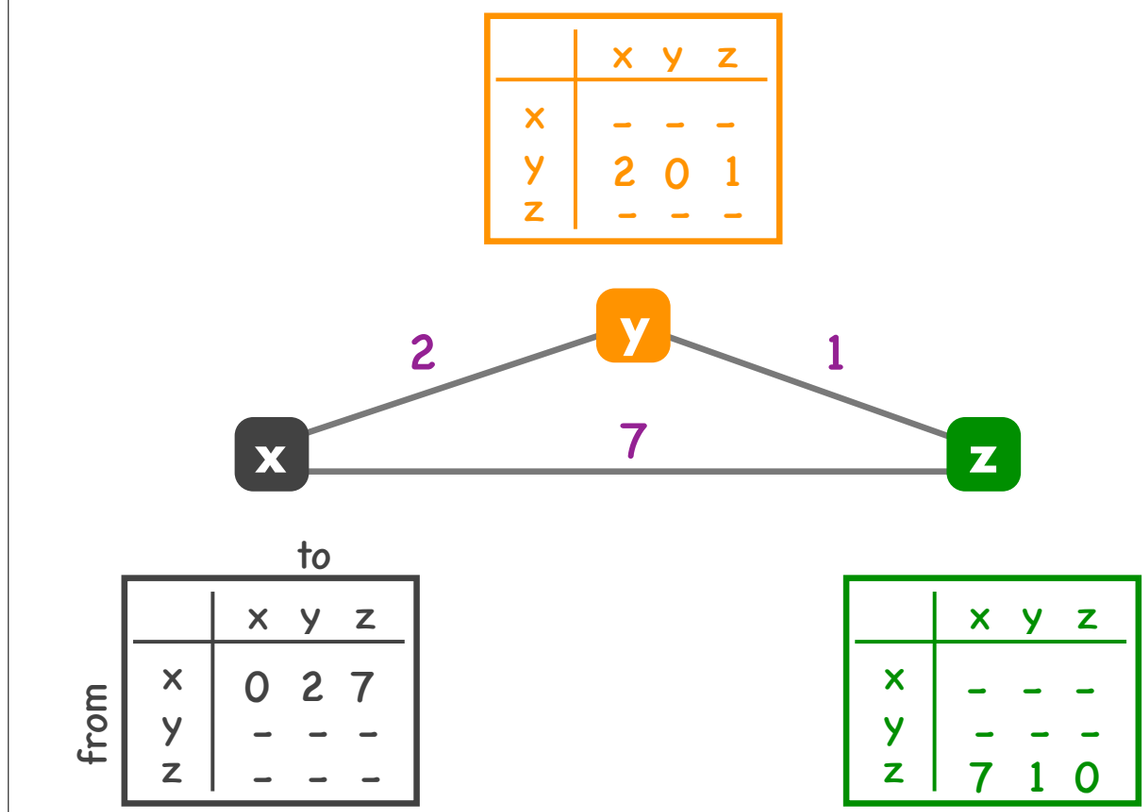


dest.	next hop	cost
z	~~-~~ x	~~-~~ 4
w	~~w~~ x	~~5~~ ~~4~~ 3
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1



dest.	next hop	cost
z	~~-~~ x	~~-~~ 4
w	~~w~~ x	~~5~~ ~~4~~ 3
y	~~-~~ x	~~-~~ 2
v	v	2
x	x	1





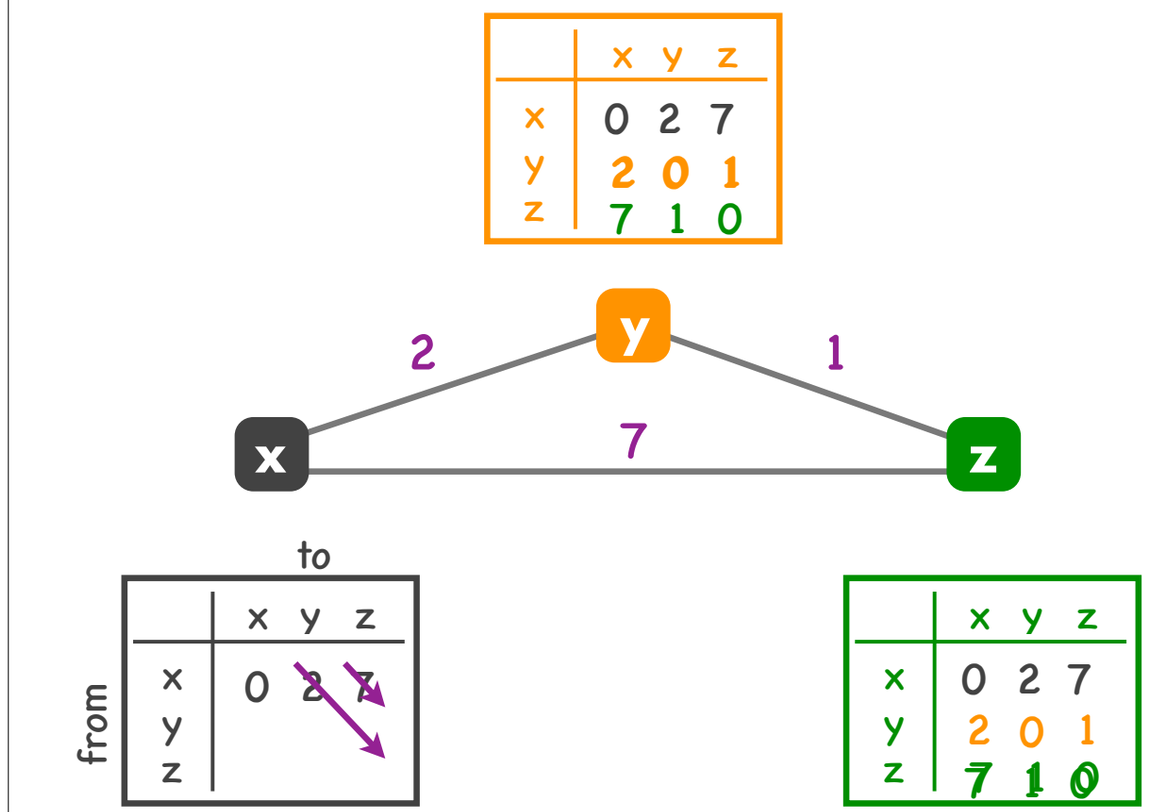
Now let’s look at another routing algorithm, which is quite different from Dijkstra.

This algorithm works in rounds and, in every round, each router exchanges information with its neighbours and recomputes its paths based on the exchanged information. So, it is a distributed algorithm, in the sense that, in each round, each router expects input from and provides input to other routers.

For the purposes of this algorithm, each router maintains a special table, which specifies the least cost between each pair of routers, and it is at first populated based on direct links.

For example, in the beginning, router x on the left knows that:
- it can reach itself with cost 0
- it can reach router y with cost 2 (through a direct link)
- and it can reach router z with cost 7 (through another direct link).

Similarly, in the beginning, router y in the middle knows that:
- it can reach router x with cost 2 (through a direct link)
- it can reach itself with cost 0
- and it can reach router z with cost 1 (through a direct link).



In the first round, all neighbors exchange tables.

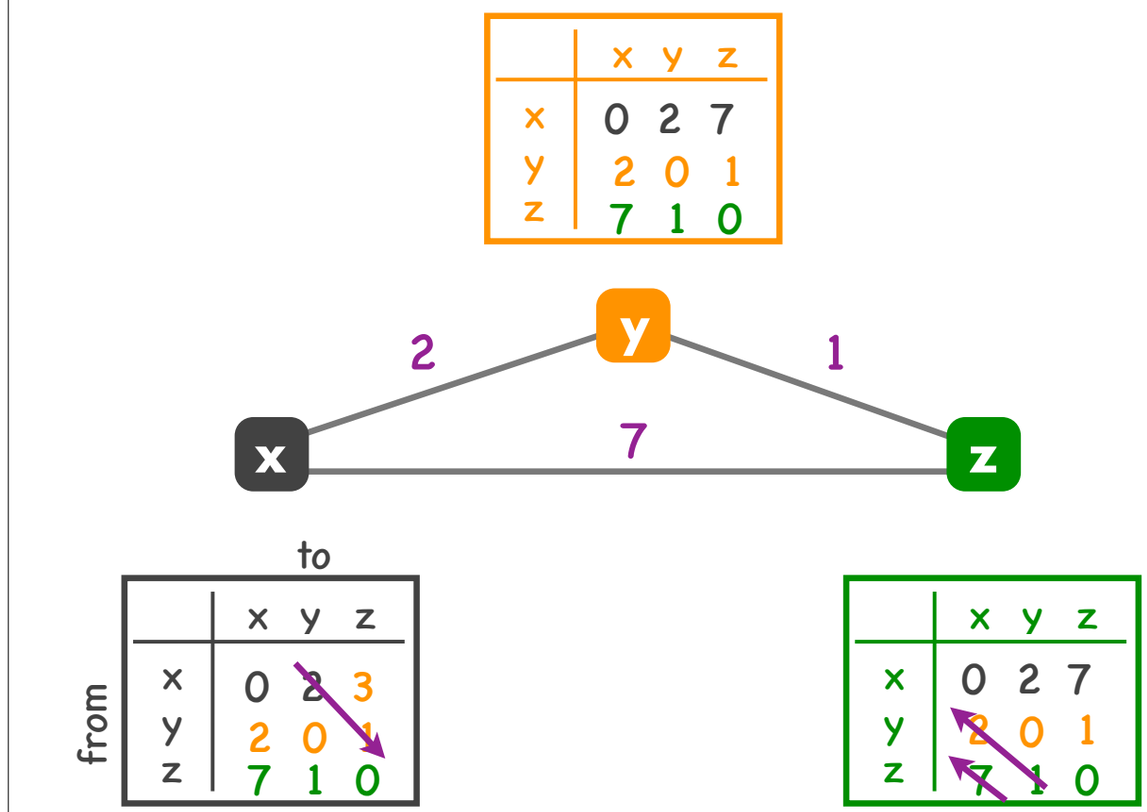
So, router x is going to learn some information from router y
and some other information from router z.

Similarly, routers y and z are going to learn new information.

Now, each router checks whether it can improve any of its current paths,
based on the new information it just learnt.

For example, concentrate on router x:

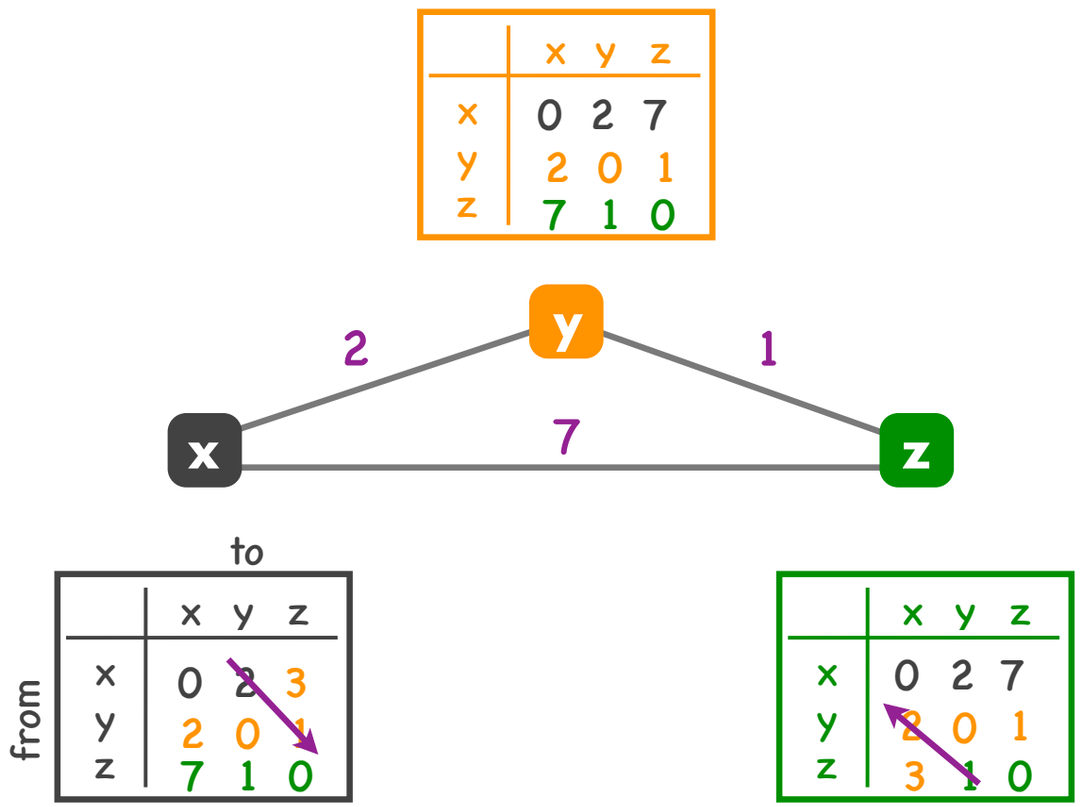
- Previously, router x could reach router z with cost 7.
- But now router x has learned that router y can reach router z with cost 1.
- And router x can reach router y with cost 2.
- This means that router x can reach router z with cost 3, through router y.



Hence, router x updates its structure with the fact that it can reach router z with cost 3.
I have made the number 3 orange to indicate that router x can reach router z with this cost through router y (the orange router).

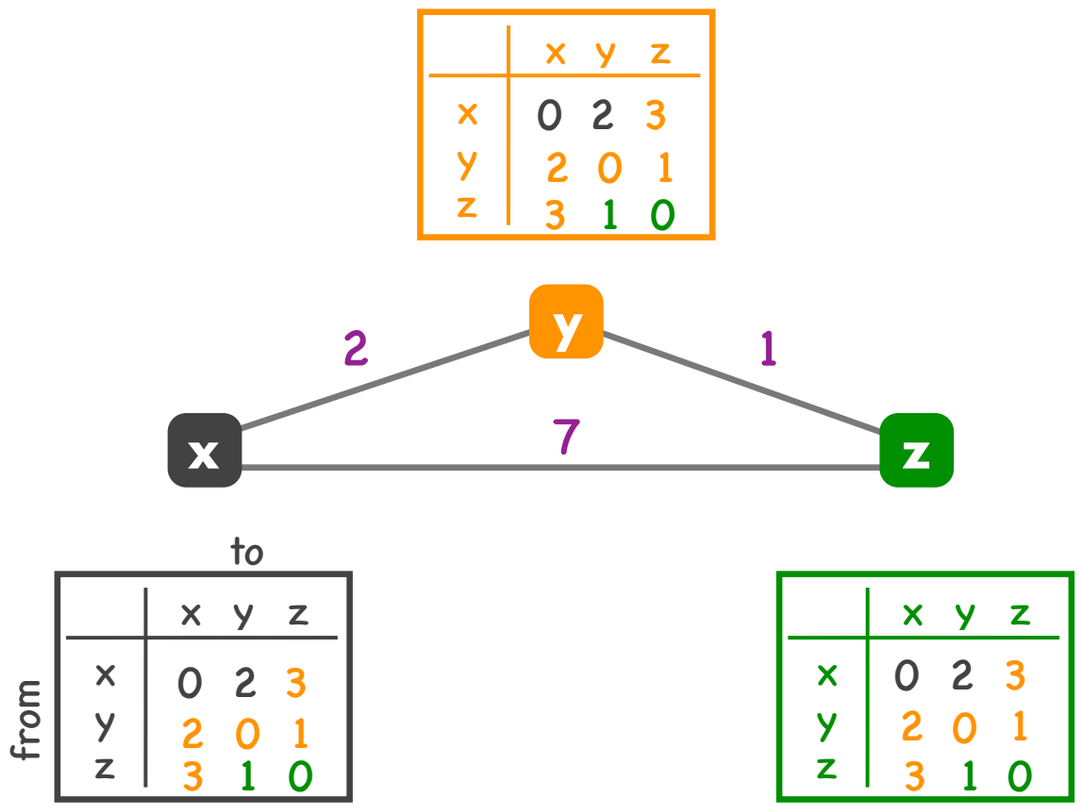
Now concentrate on router z:

- Previously, router z could reach router x with cost 7.
- But now router z has learned that router y can reach router x with cost 2.
- And router z can reach router y with cost 1.
- This means that router z can reach router x with cost 3, through router y.



Hence, router z updates its structure with the fact that it can reach router x with cost 3.
I have made the number 3 orange to indicate that router z can reach router x with this cost through router y.

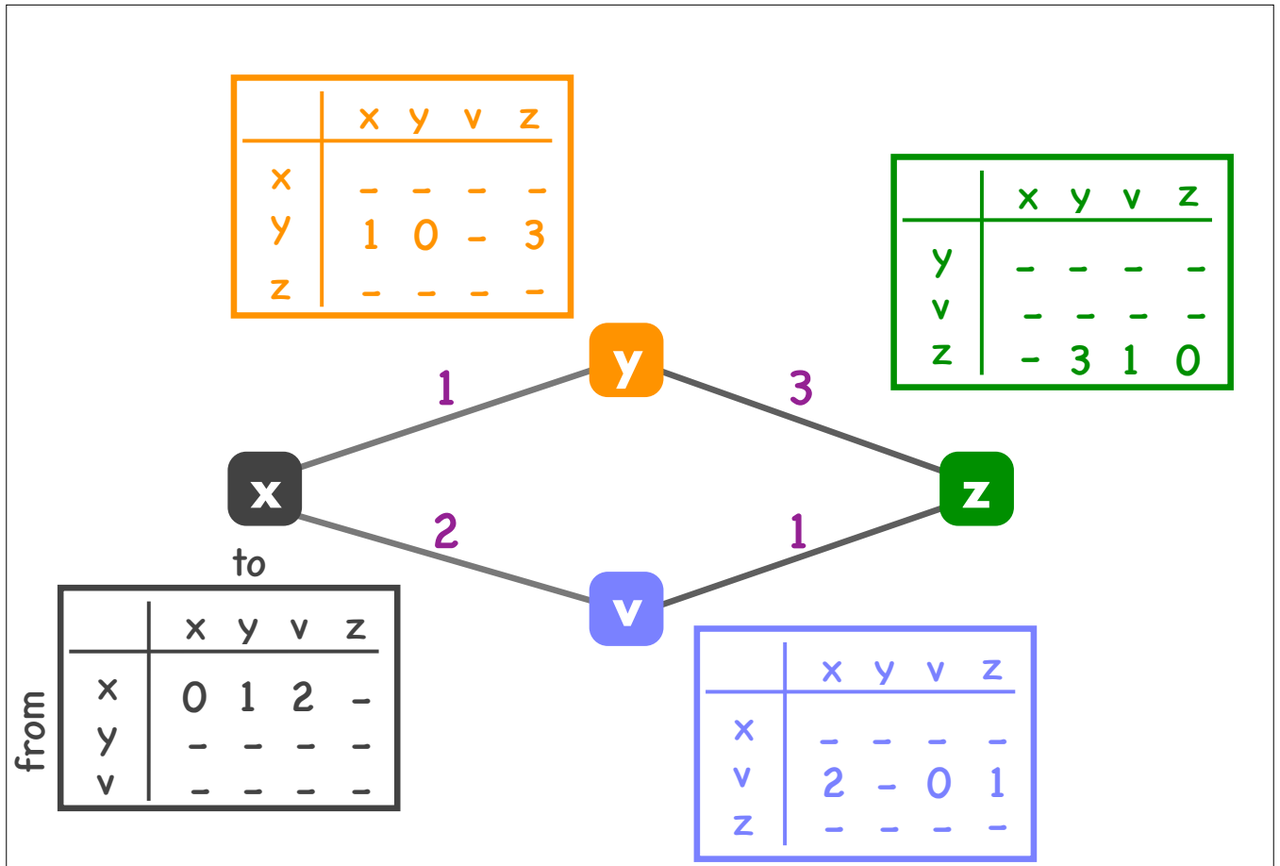
Finally, unlike routers x and z, router y cannot improve its paths with the new information it received. This is because router y's best paths to both x and z are direct paths that router y already knew about before exchanging any information with neighbors.



In the second round, all the neighbors exchange tables again.

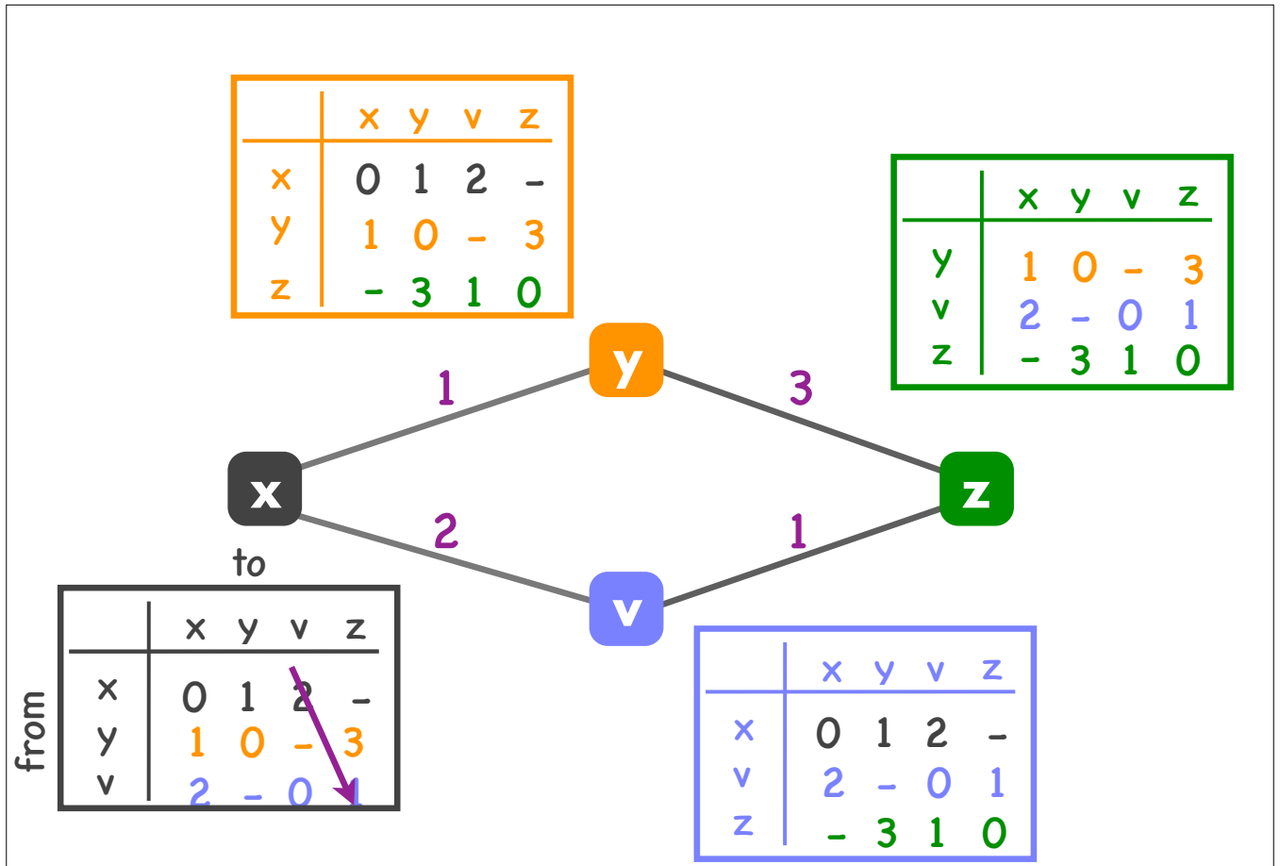
Each router checks whether it can improve its existing routes based on the new information it has just received.

In this example, they cannot, so the algorithm is done: each router has computed the least-cost path to every other router in the network.



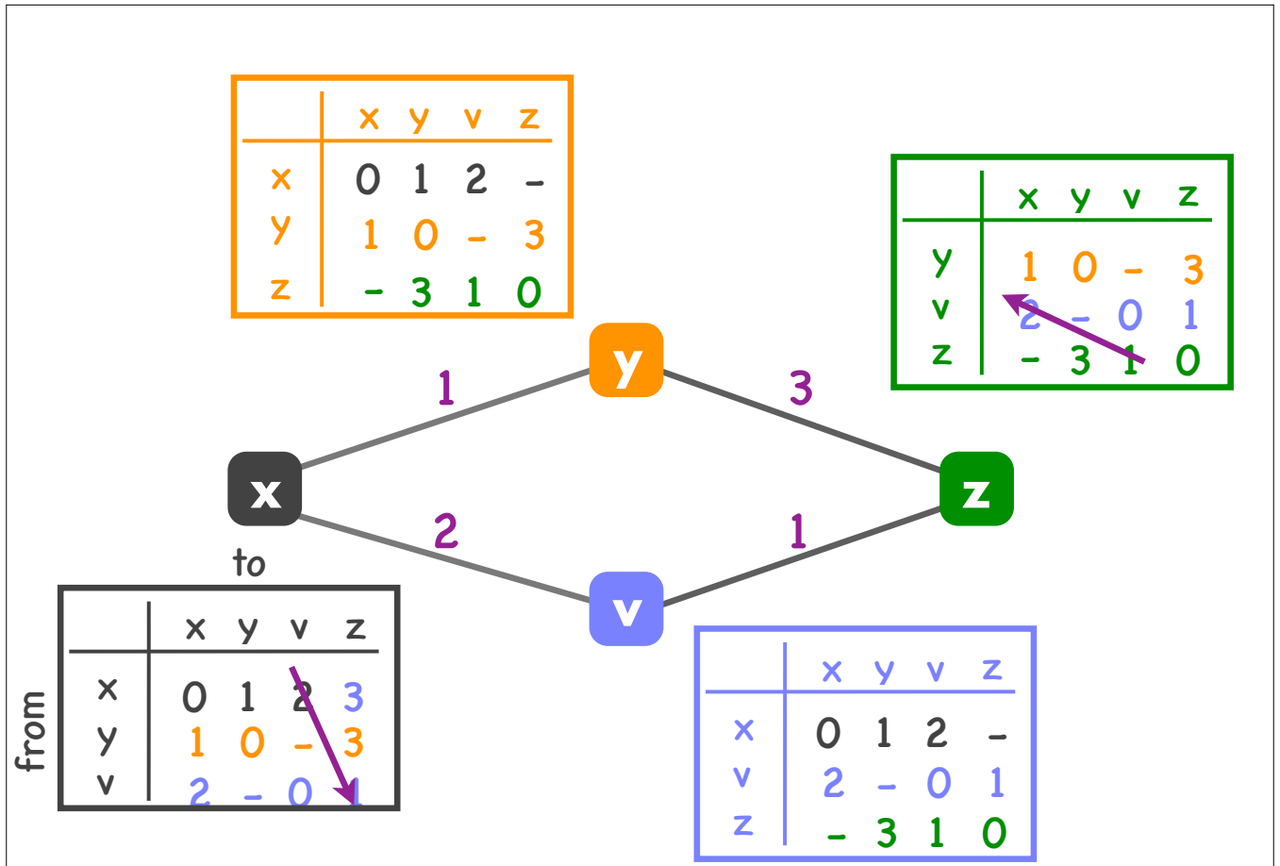
Here is a second, a bit more complicated example:

In the beginning, each router knows how to reach only its direct neighbors.



In the first round, all the neighbors exchange tables.

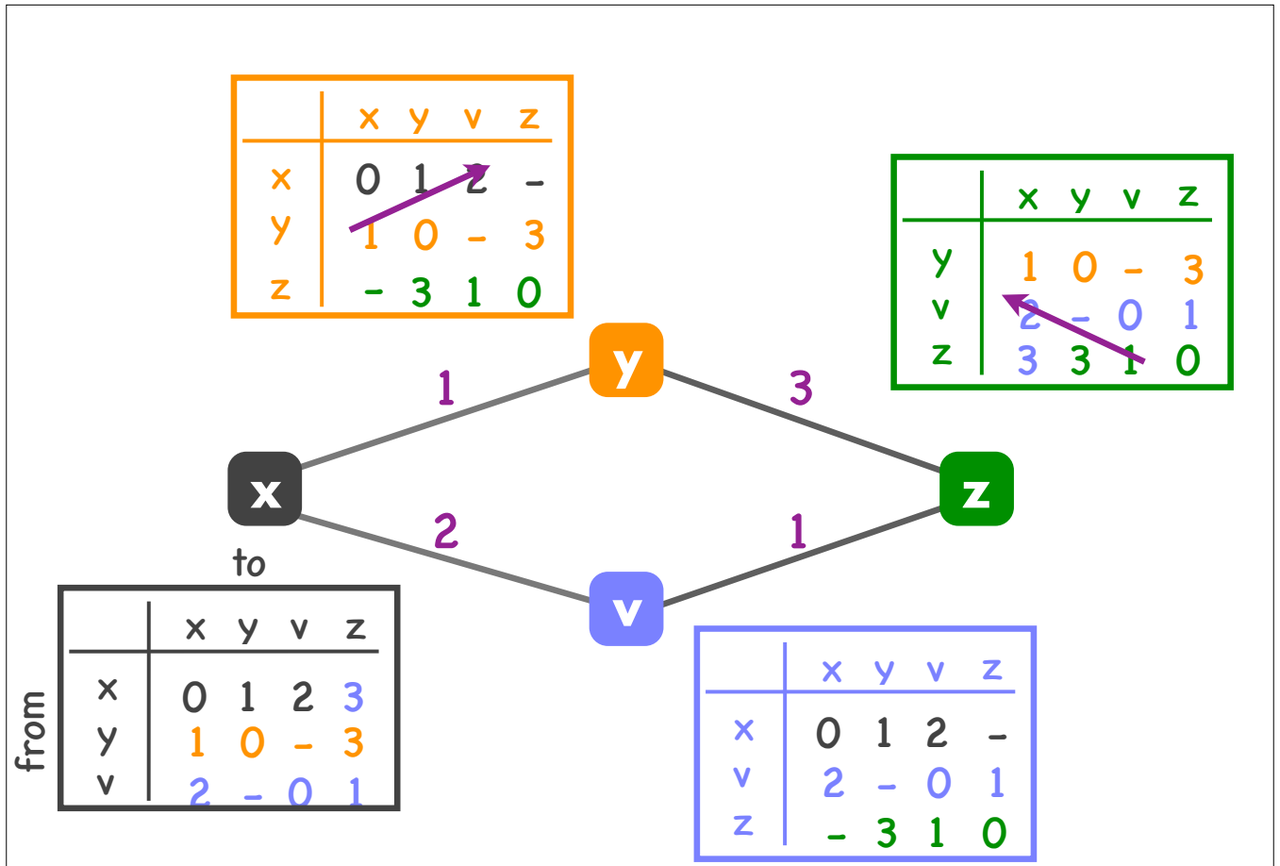
As a result, router x discovers a new path to router z, through router v, with cost 2+1=3…



and updates its table accordingly (note that the new number 3 that appears in x's table is purple, because it was computed based on information propagated through router v).

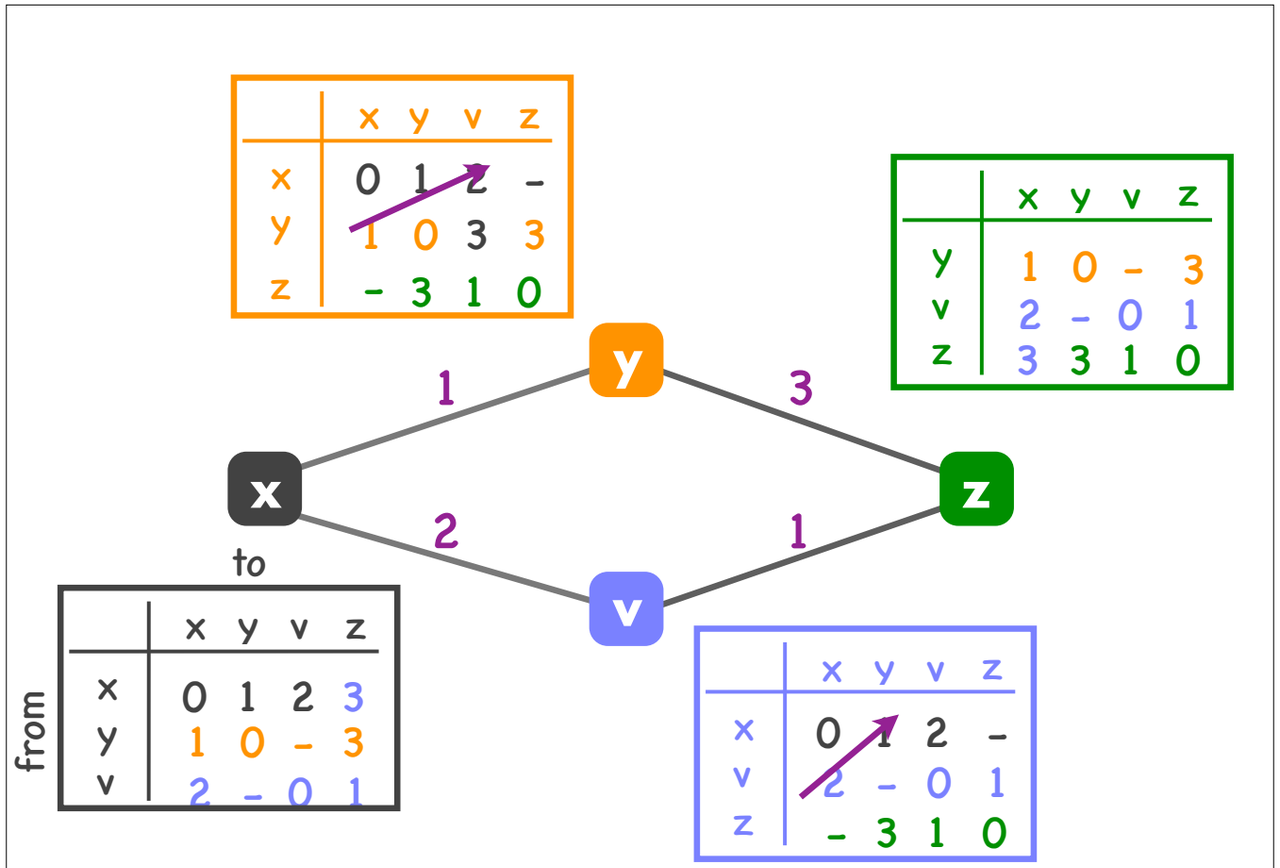
(Parenthesis: Router x actually discovers two new paths to router z: one through v, of cost 3, and one through y, of cost 4. It compares the two and determines that the one through v is better.)

Similarly, router z discovers a new path to router x, through router v, with cost 3…



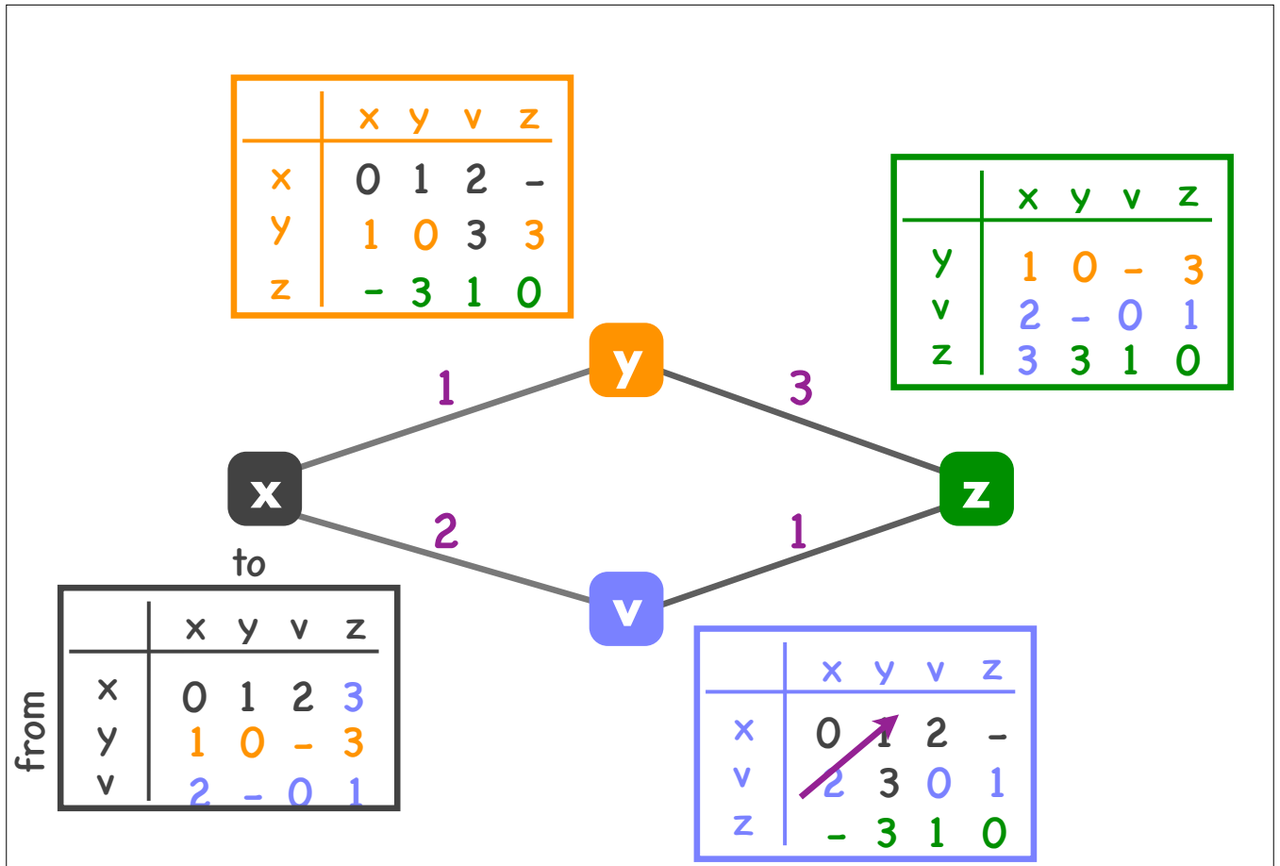
and updates its table accordingly.

At the same time, router y learns a new path to router v, through router x, with cost 1+2=3…

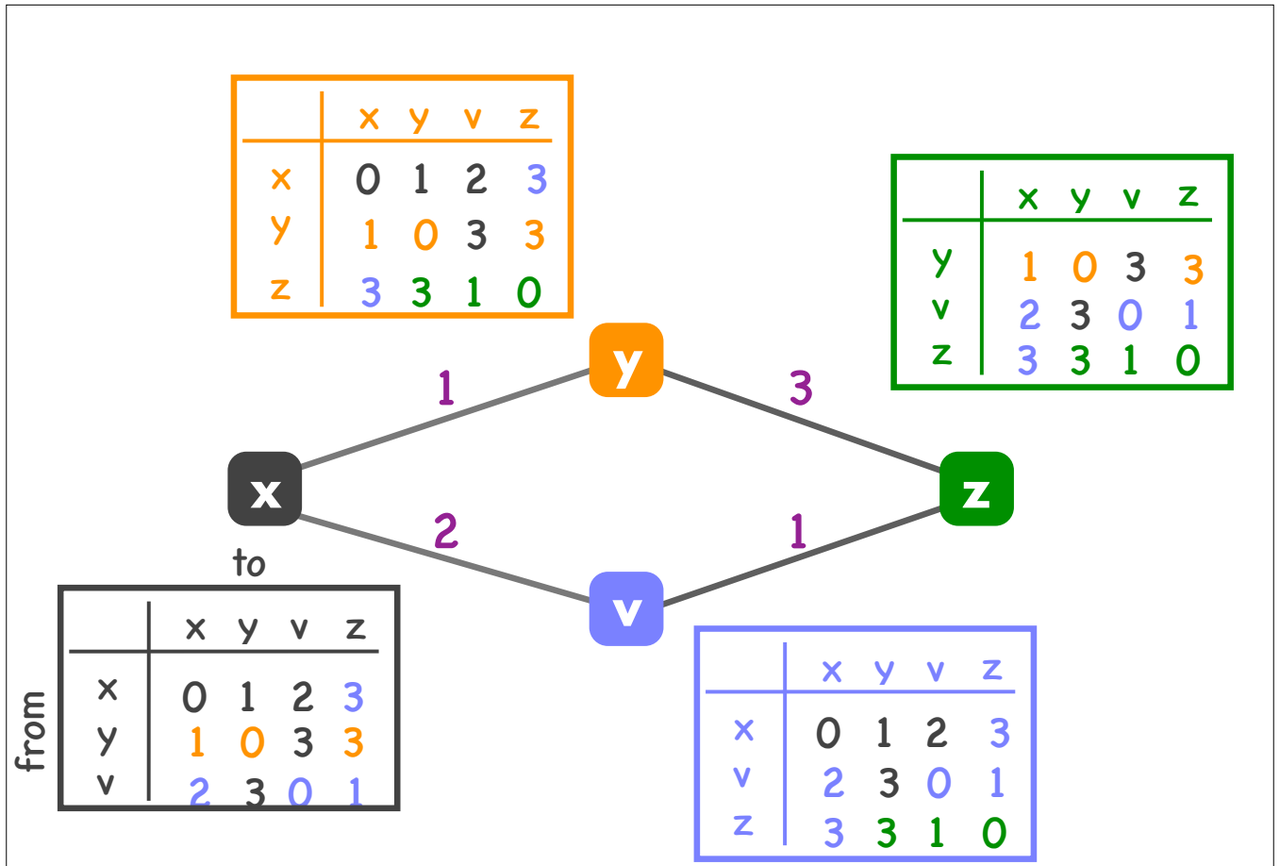


and updates its table accordingly (note that the new number 3 that appears in y's table is black, because it was computed based on information propagated by router x).

Similarly, router v learns a new path to router y, through router x, with cost 2+1=3…



and updates its table accordingly.



In the second round, all the neighbors exchange tables again.

This time, no router can improve any path with the new information that it has just received, so the algorithm is done.

Distance-vector routing algorithm

- Input to each router: local link costs
  & neighbor messages
- Output of each router: least-cost path to every other router

What we saw is an example of a distance-vector routing algorithm:

- In each round, the instance of the algorithm running at each router takes as input the router's current table plus the tables of its neighbors.
- At the end, the instance of the algorithm running at each router produces as output the least-cost path to every other router in the network.

Distance-vector routing algorithm

- "Distributed" algorithm
- All routers run it "together": neighbors exchange and react to each other's messages

A distance-vector routing algorithm is a "distributed" algorithm, in the sense that all routers run it "together": they exchange messages and they keep reacting to each other's messages.

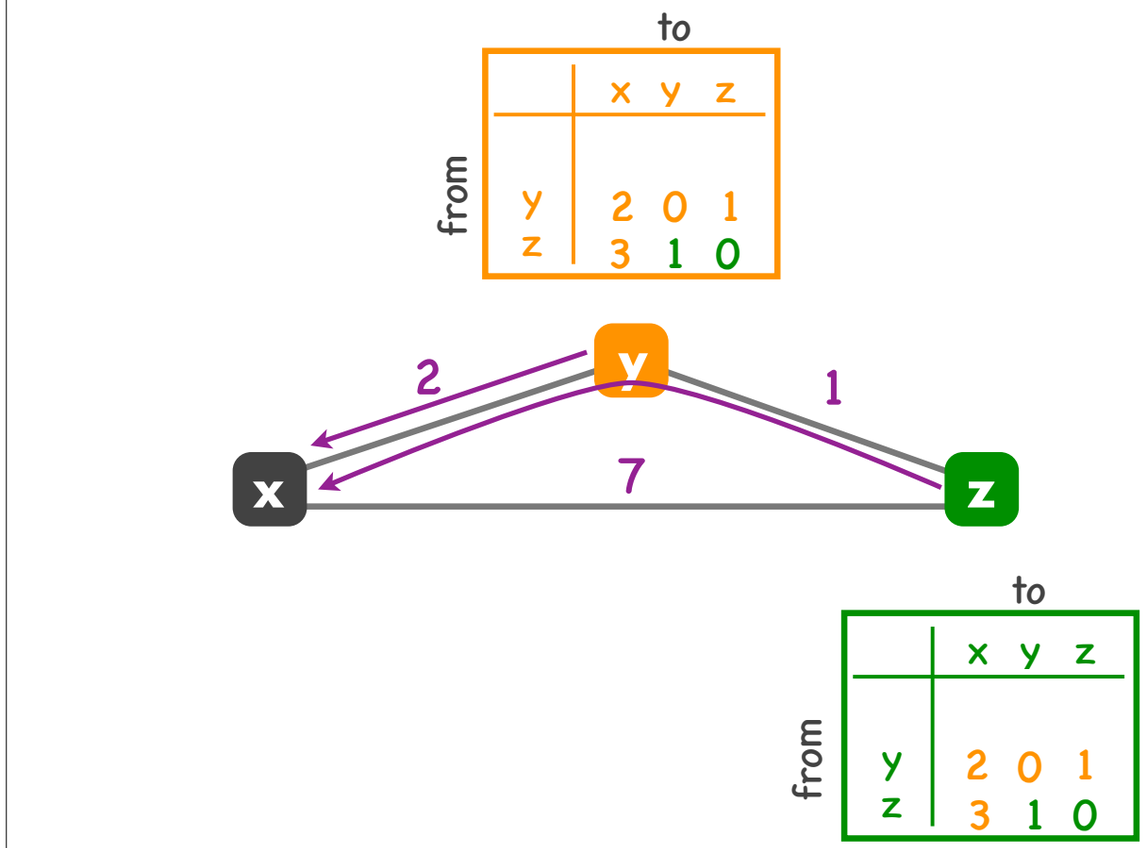
Bellman-Ford algorithm

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

The particular distance-vector routing algorithm that we discussed is called the Bellman-Ford algorithm.

In summary:

- At each step, all neighbors exchange tables.
- Each router checks whether it can use the new information it just received to improve its current paths.
- The algorithm ends when no improvement is possible.

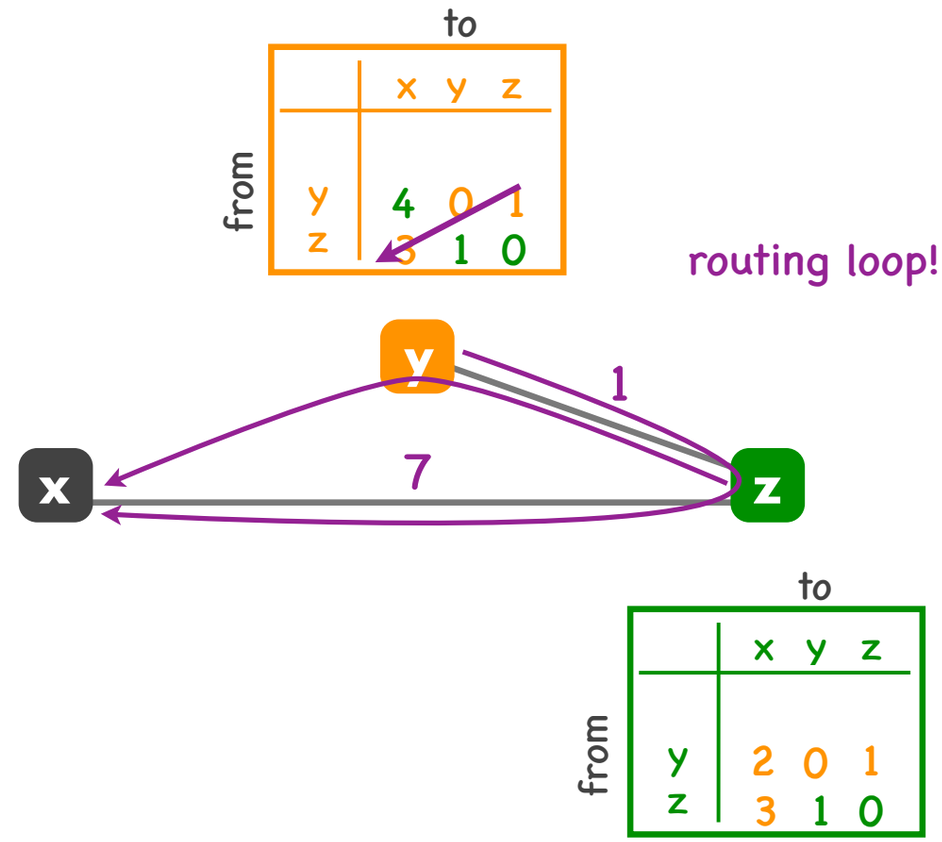


One of the main challenges in designing routing protocols is avoiding routing loops.

I will now illustrate this challenge: I will show you how a naive implementation of the Bellman-Ford routing protocol can lead to routing loops.

Consider this 3-router network, where router y's path to router x is direct,
and router z's path to router x is indirect, through router y.

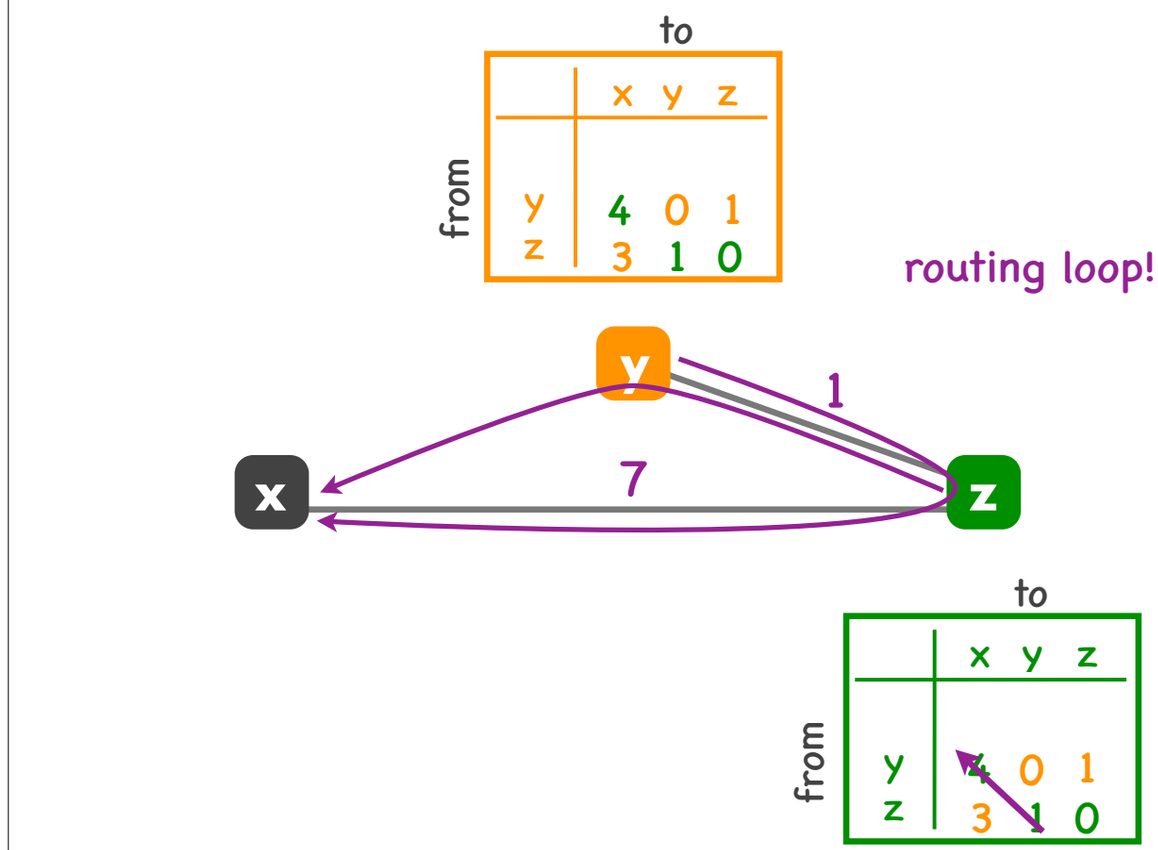
Now suppose that link (x,y) breaks down.



Router y immediately detects the problem, because it is directly connected to router x;
so, it realizes that it does not have a link to router x any more.

But, router y makes a mistake:
- It says: oh, router z can route to router x with cost 3.
- So, let me route to router x through router z, since I cannot route directly any more.
- Since I can route to router z with cost 1, the cost of this new indirect path to router x is 1 + 3 = 4.

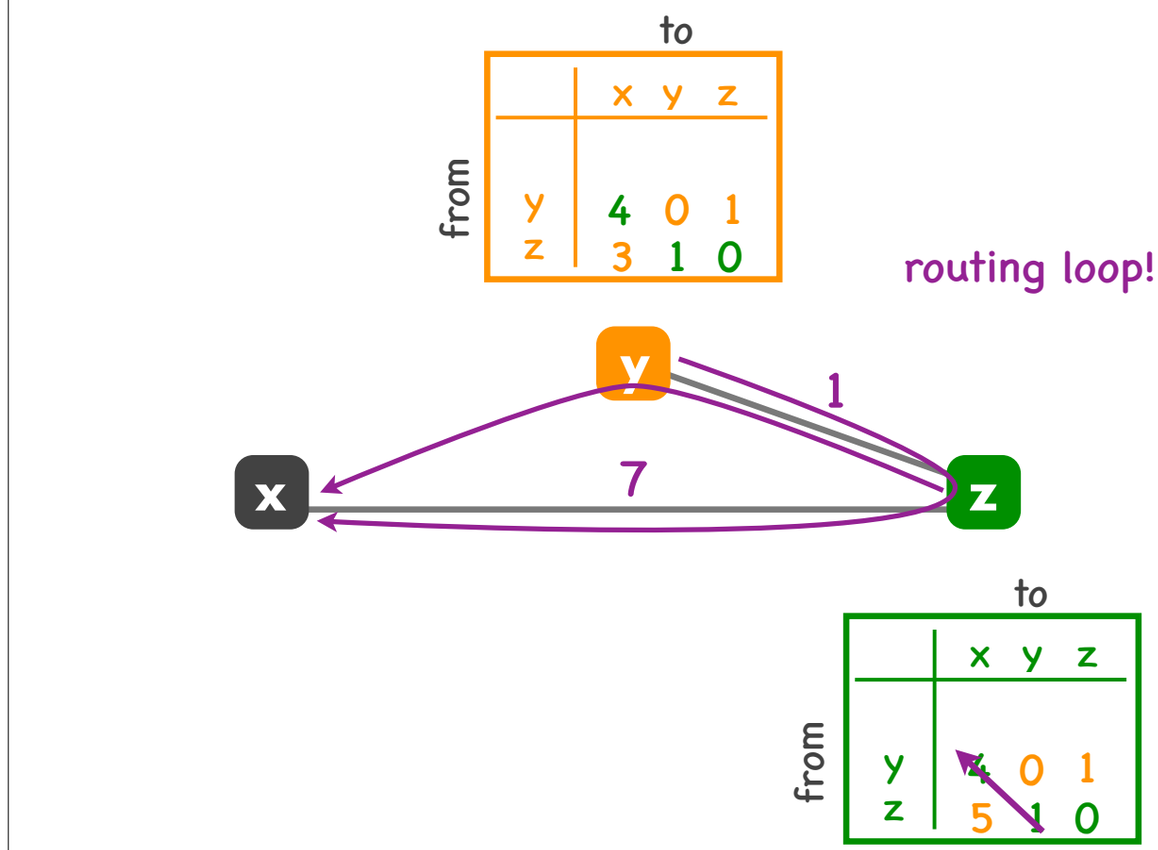
So, at this point, we have a nice routing loop:
- router z is routing to router x through router y
- and router y is routing to router x through router z.
- so, packets going to router x will ping-pong between routers y and z.



Here is what happens next:

Routers y and z exchange tables.

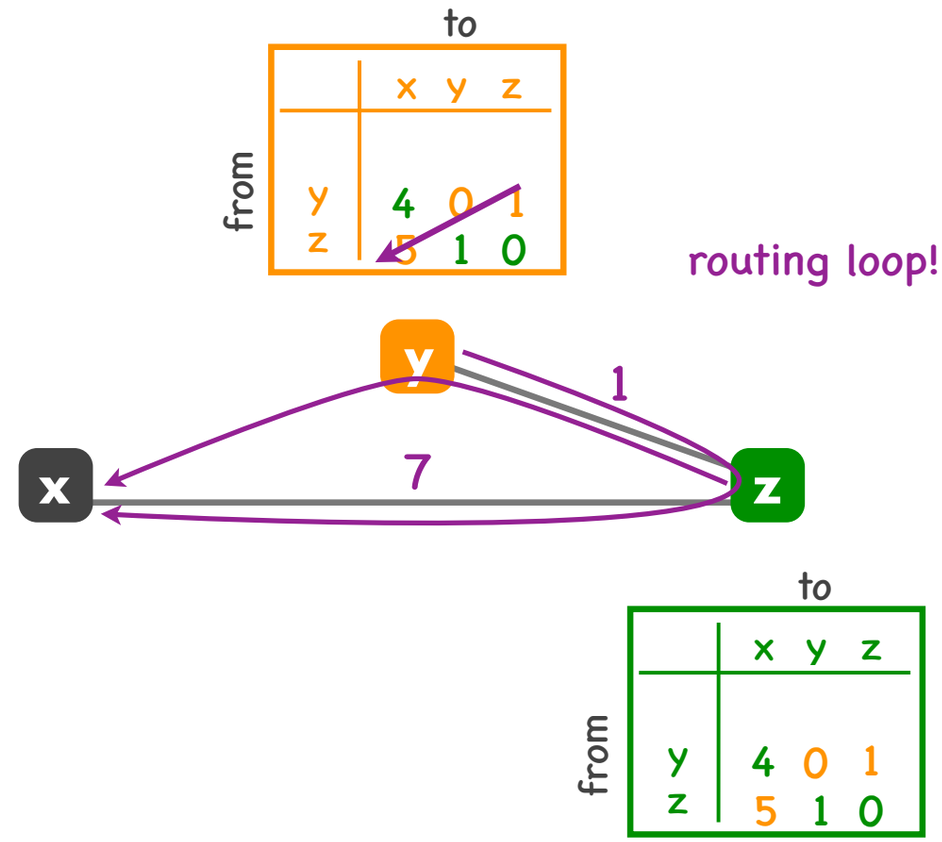
Router z says:
- my path to router x is through router y;
- router y's cost to router x has increased to 4;



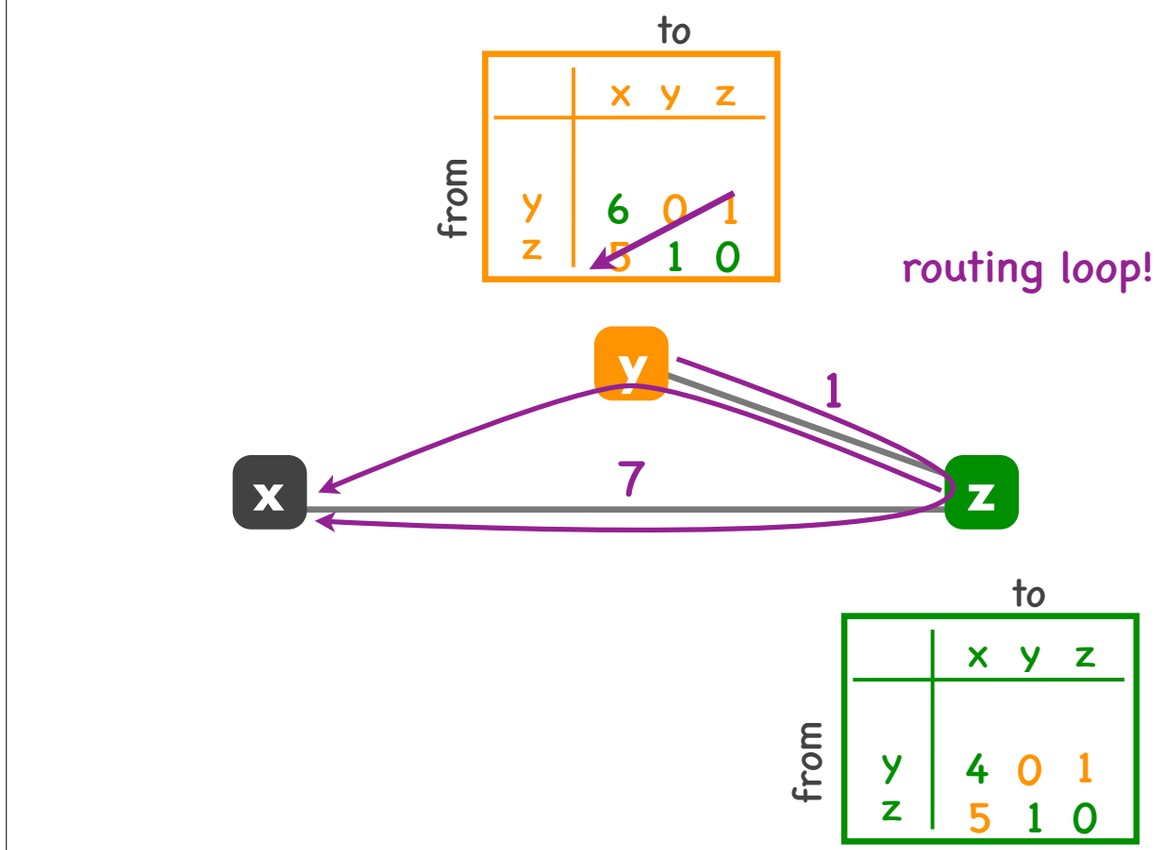
- so let me update my cost to router x to 1+4=5.

And the loop persists.

Then the neighbors exchange tables again.



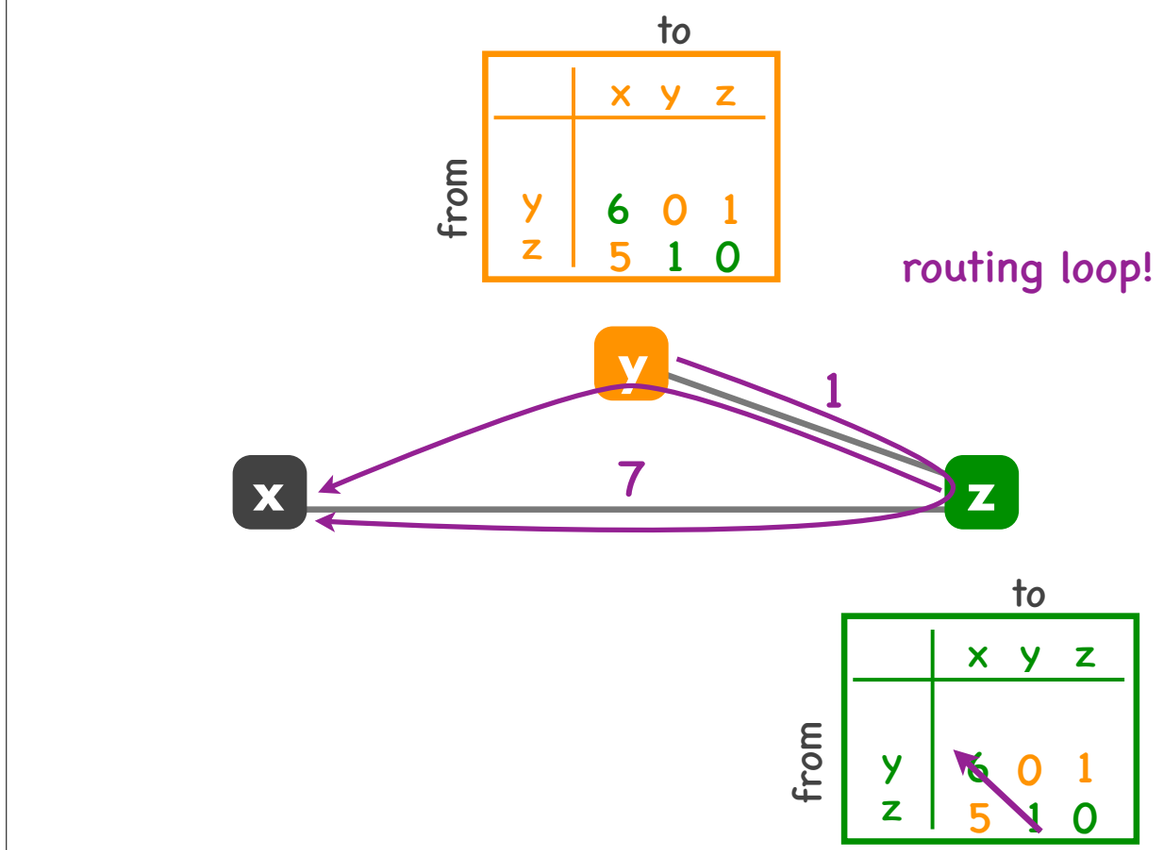
Router y says:
- my path to router x is through router z;
- router z's cost to router x has increased to 5;



- so, let me also update my cost to router x to 1+5=6.

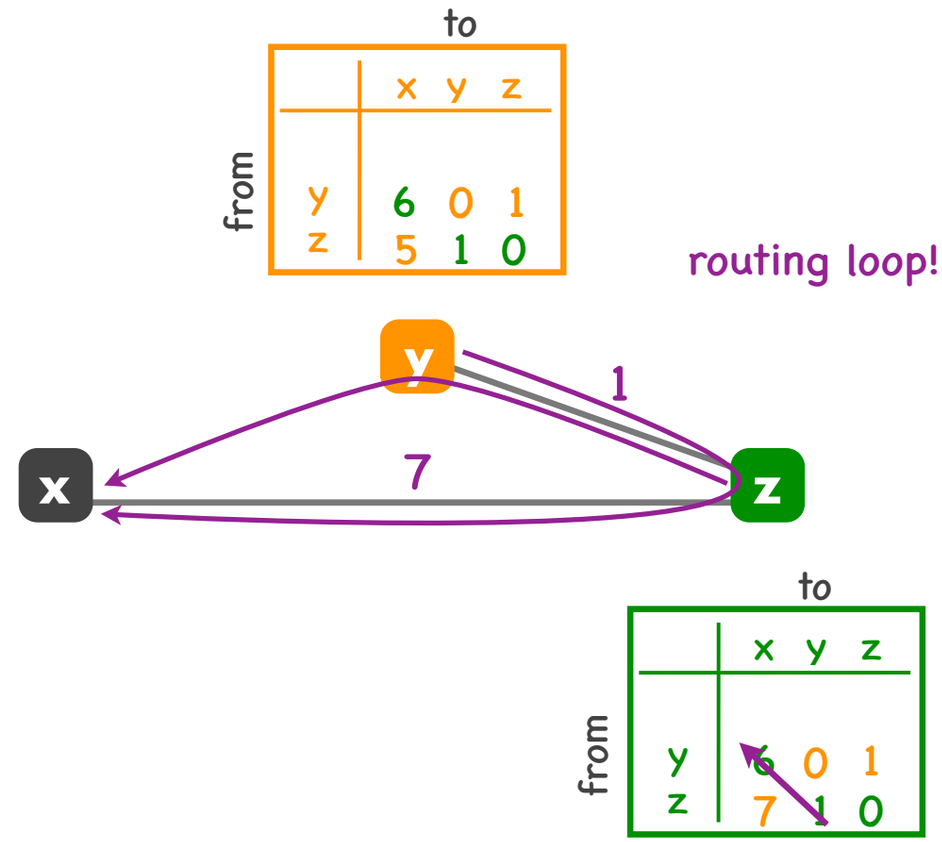
And the loop persists.

Then the neighbors exchange tables again.



The same story continues:

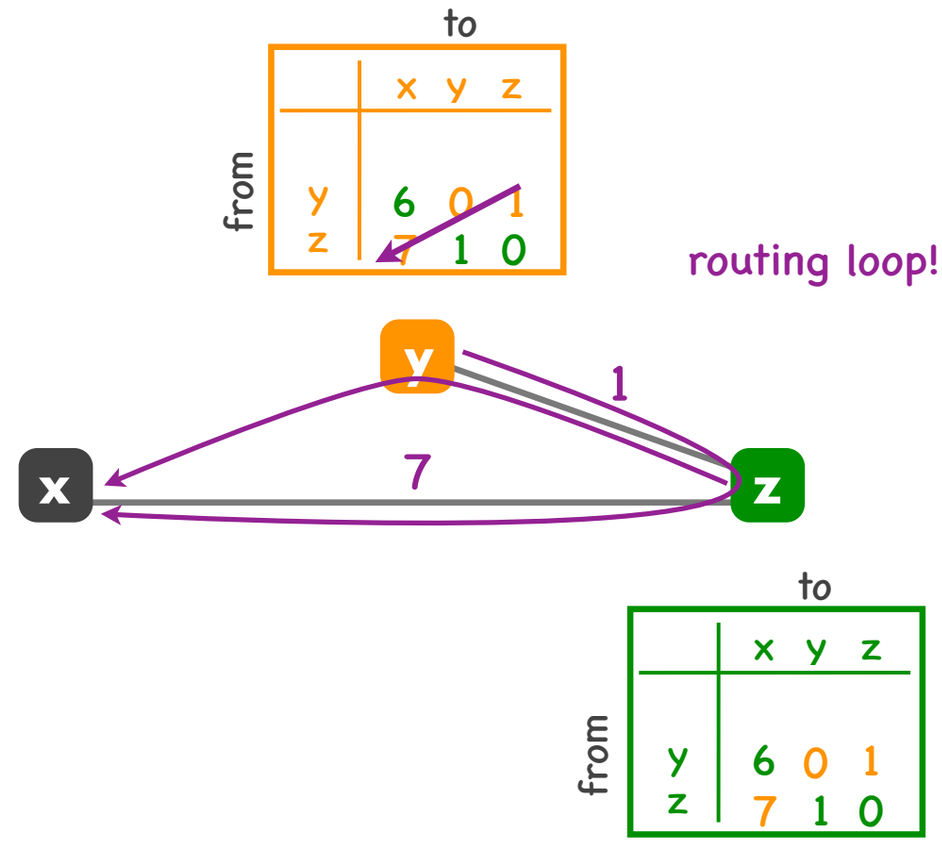
Router z says:
- my path to router x is through router y;
- router y's cost to router x has increased to 6;



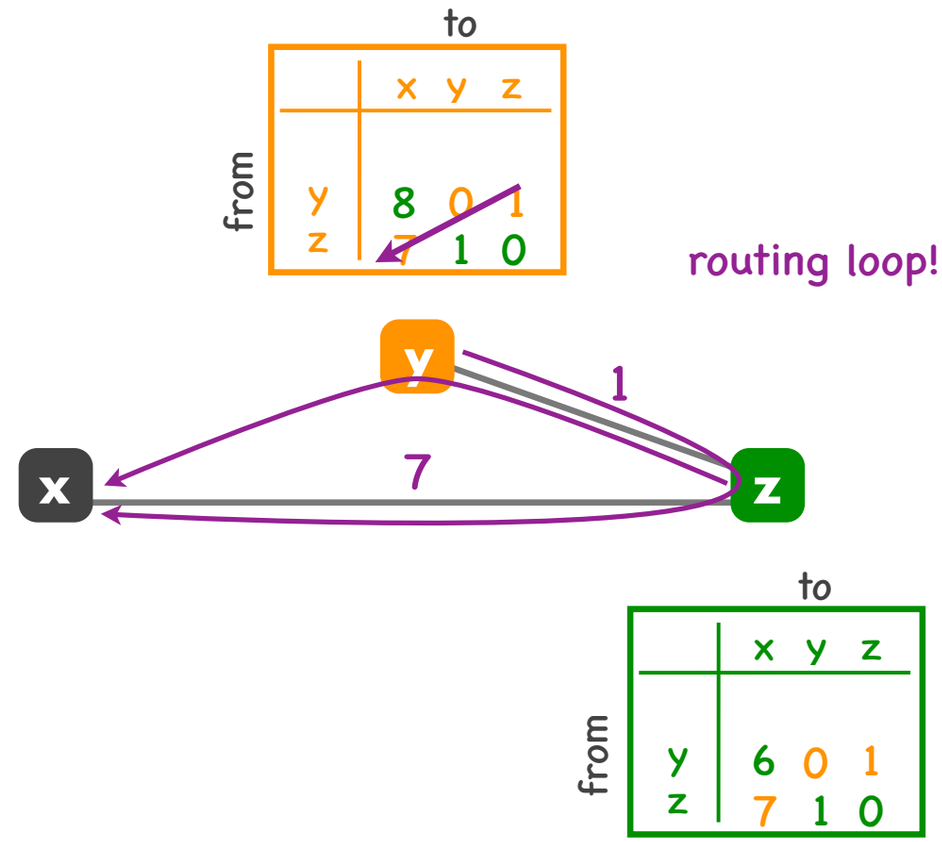
- so let me update my cost to router x to 7.

And the loop persists.

Then the neighbors exchange tables again.



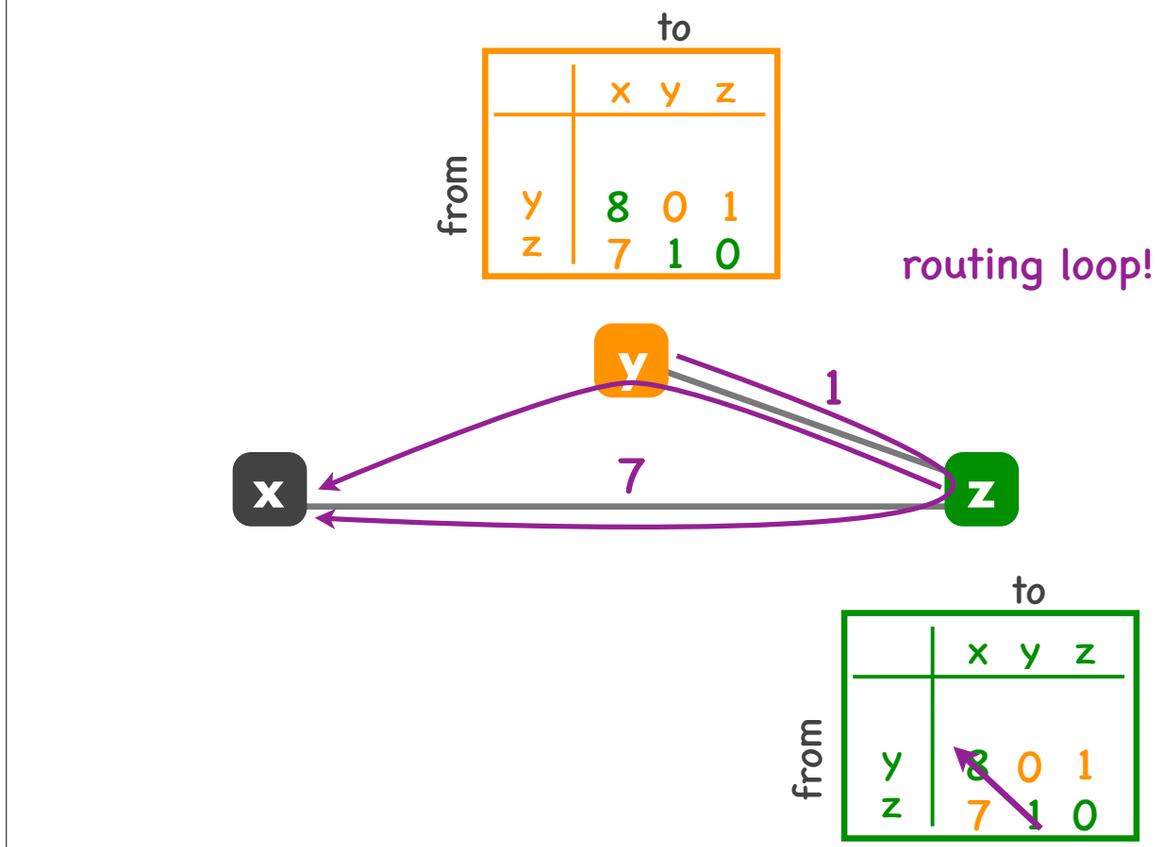
Router y says:
- my path to router x is through router z;
- router z's cost to router x has increased to 7;



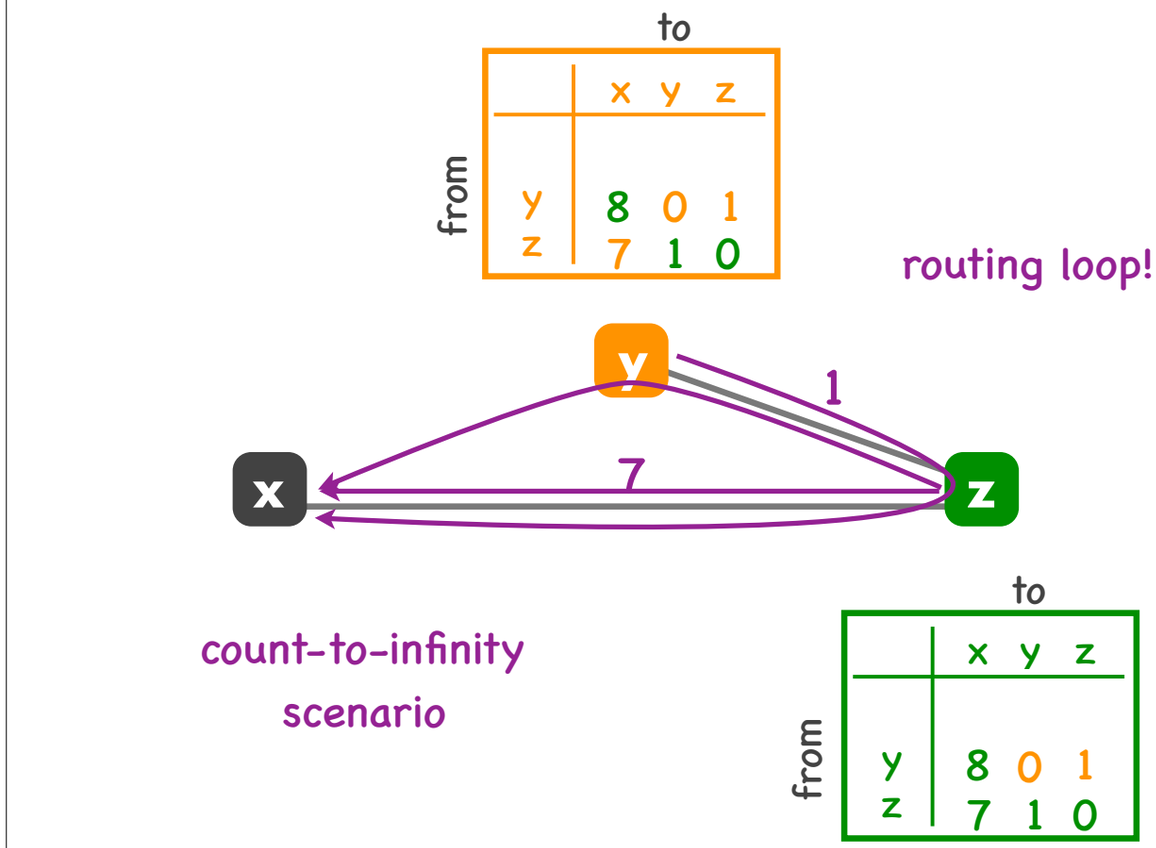
- so, let me update my cost to router x to 8.

And the loop persists.

Then the neighbors exchange tables again.



Router z says:
- my path to router x is through router y;
- router y's cost to router x has increased to 8;
- so, if I continue to route through router y, my cost to router x will become 9;
- I have a better path to router x: I have a direct path, of cost 7.



At this point, router z changes its path to router x: it routes to x through the direct link of cost 7 (note that the number 7 in z's table has now become green, it indicate that it corresponds to a direct link).

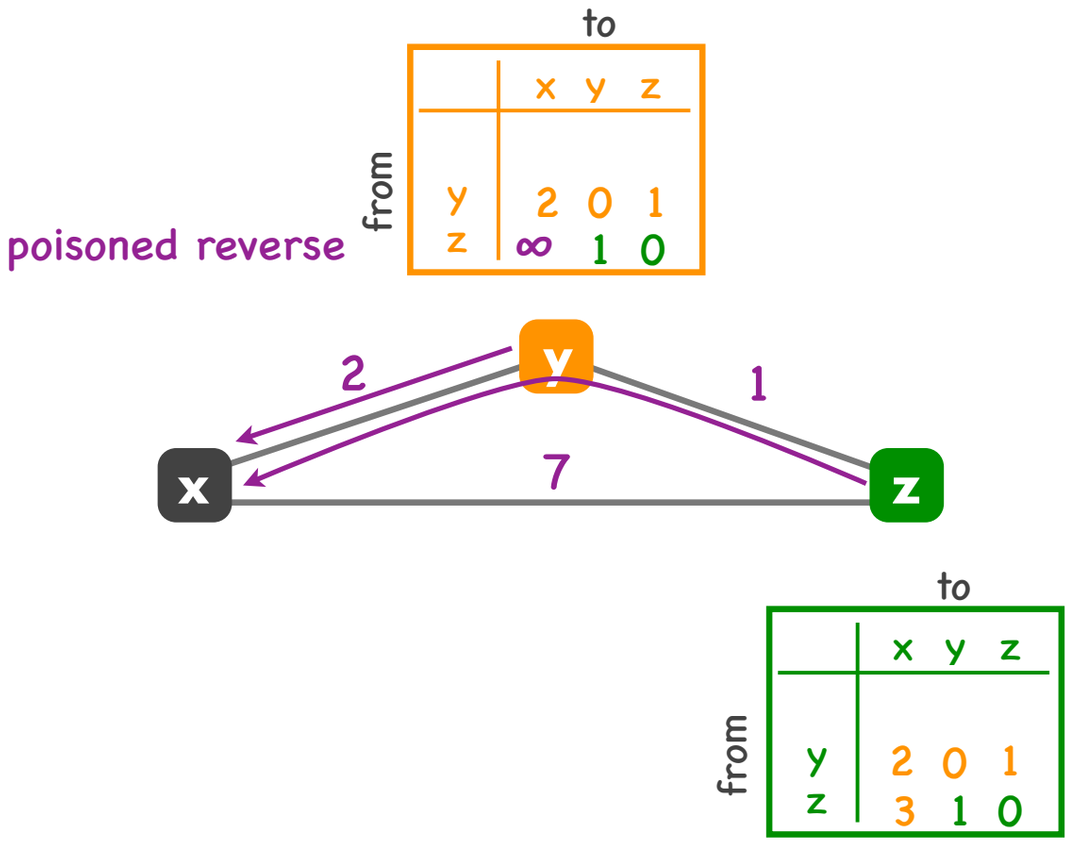
This is where the routing loop ends.

The scenario I just described is called "count-to-infinity" (because the routers keep "counting," i.e., increasing their costs to router x by 1).

Problem with Bellman-Ford

- Routing loop
  - z routes to x through y
  - y loses connectivity to x
  - y decides to route to x through z
- Can take very long to resolve
  - count-to-infinity scenario

So: If we are not careful with the Bellman-Ford implementation (or any routing protocol implementation, for that matter), we may end up with routing loops, which may take a long time to resolve.

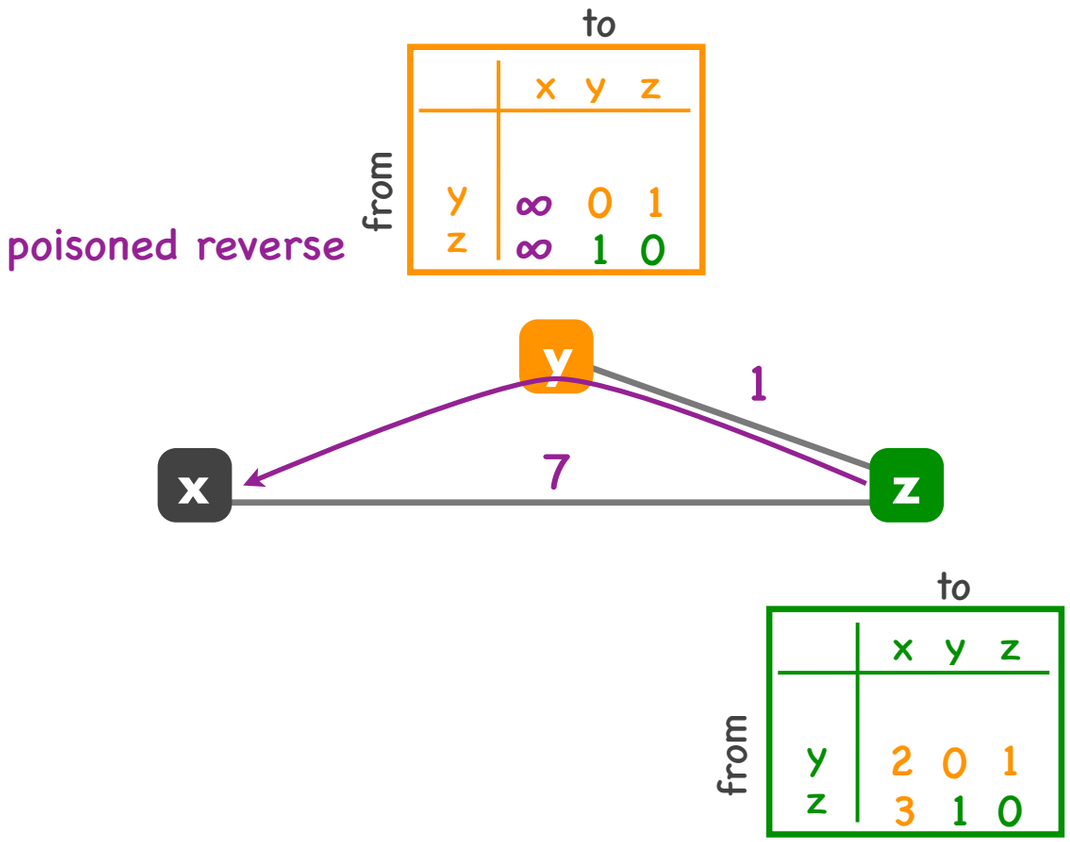


Here is a solution:
- If the path from router z to router x goes through router y,
- then router z tells router y that its cost to router x is infinity.
- In other words, router y should never try to route to router x through router z,
- because z is routing to x through y.

This technique is called *poisoned reverse*. This is because the “reverse path” (the path from y to z to x, in our example) is “poisoned” (i.e., y is not allowed to use it).

Now, if the link between x and y goes down, router y realizes that:
- it cannot route to x directly
- and it cannot route to x through z (because of the poisoned reverse).

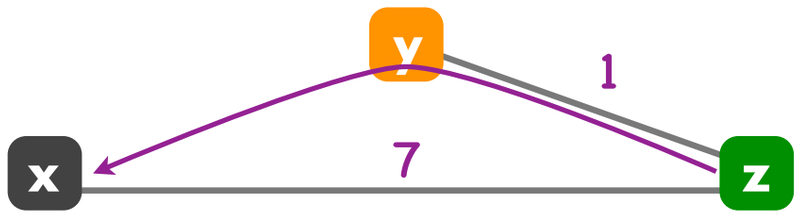
poisoned reverse



Hence, router y sets the cost of the path from itself to router x to infinity. Note that this infinity is NOT a poisoned reverse. It just indicates that the link from y to x is down, and y cannot route to x.

poisoned reverse

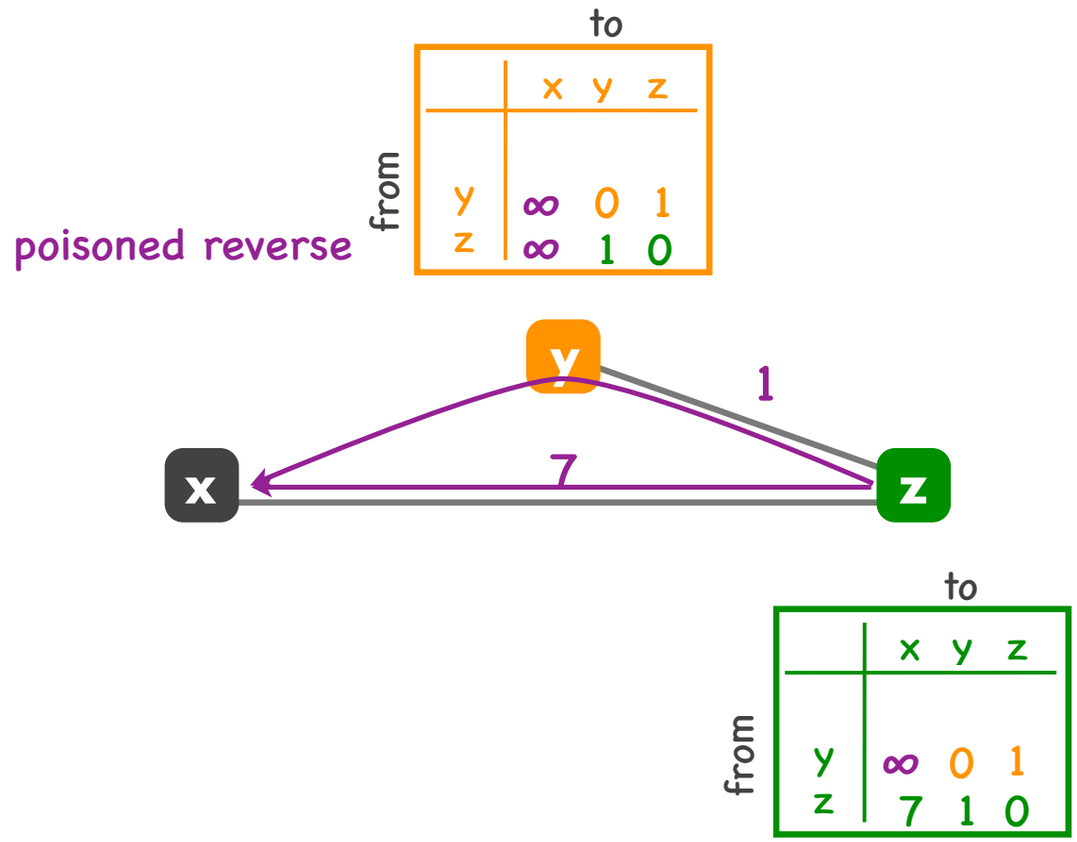
from \ to	x	y	z
y	∞	0	1
z	∞	1	0



from \ to	x	y	z
y	∞	0	1
z	3	1	0

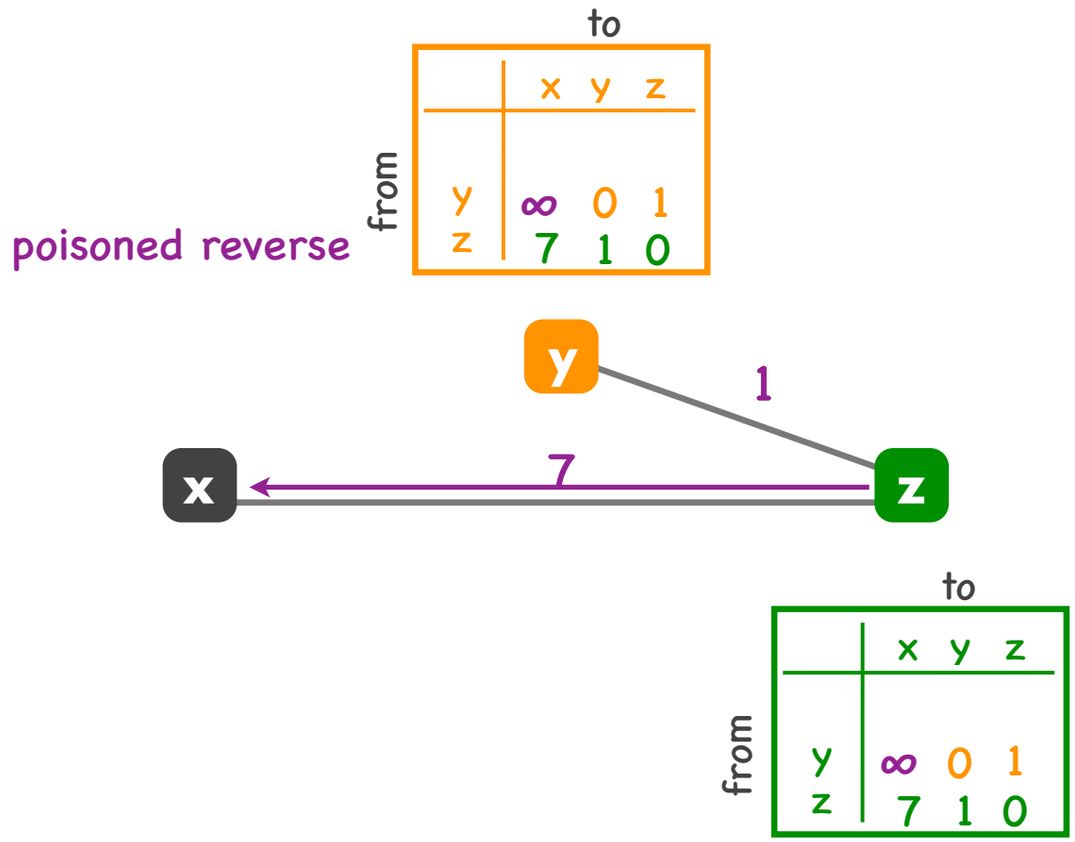
Then the neighbors exchange tables,
router z learns that router y has been disconnected from router x…

poisoned reverse

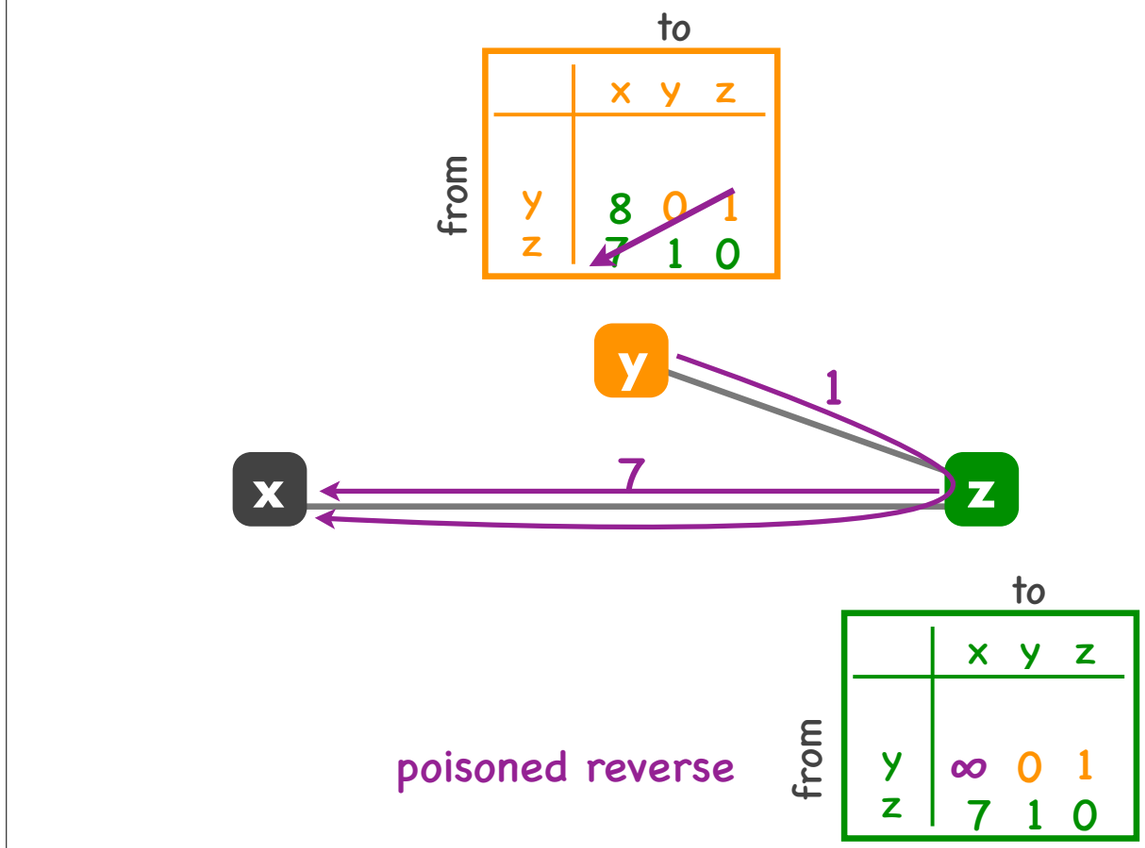


hence chooses the direct path to router x, which has cost 7, and updates its table accordingly.

poisoned reverse



Then the neighbors exchange tables again,
Router y learns that router z has a path to router x of cost 7…



hence realizes that it (router y) can route to router x through router z with cost 1+7=8 and updates its table accordingly.

At this point, both routers have computed their paths to router x correctly.

Important: notice the poisoned reverse in router z's table, which ensures that router z will never try to route to router x through router y.

Solution

- Poisoned reverse
  - if z routes to x through y,
    z advertises to y that its cost to x infinite
  - y never decides to route to x through z
- Algorithm re-converges quickly
  - avoids count-to-infinity scenario

So: One way to avoid the count-to-infinity scenario is poisoned reverse, which enables the Bellman-Ford algorithm to re-converge quickly after a link failure.

Link-state + distance-vector

- They solve the same problem:
  compute the least-cost path
  from each source router
  to each destination router

We discussed two types of routing protocols: link state and distance vector.

They both solve the same problem: compute the least-cost paths between routers.

But, they do it…

Link-state vs. distance-vector

- Link state: each entity first obtains complete view of the network, then computes the least-cost paths
- Distance vector: each entity obtains incrementally new information about the network at every round

differently: …

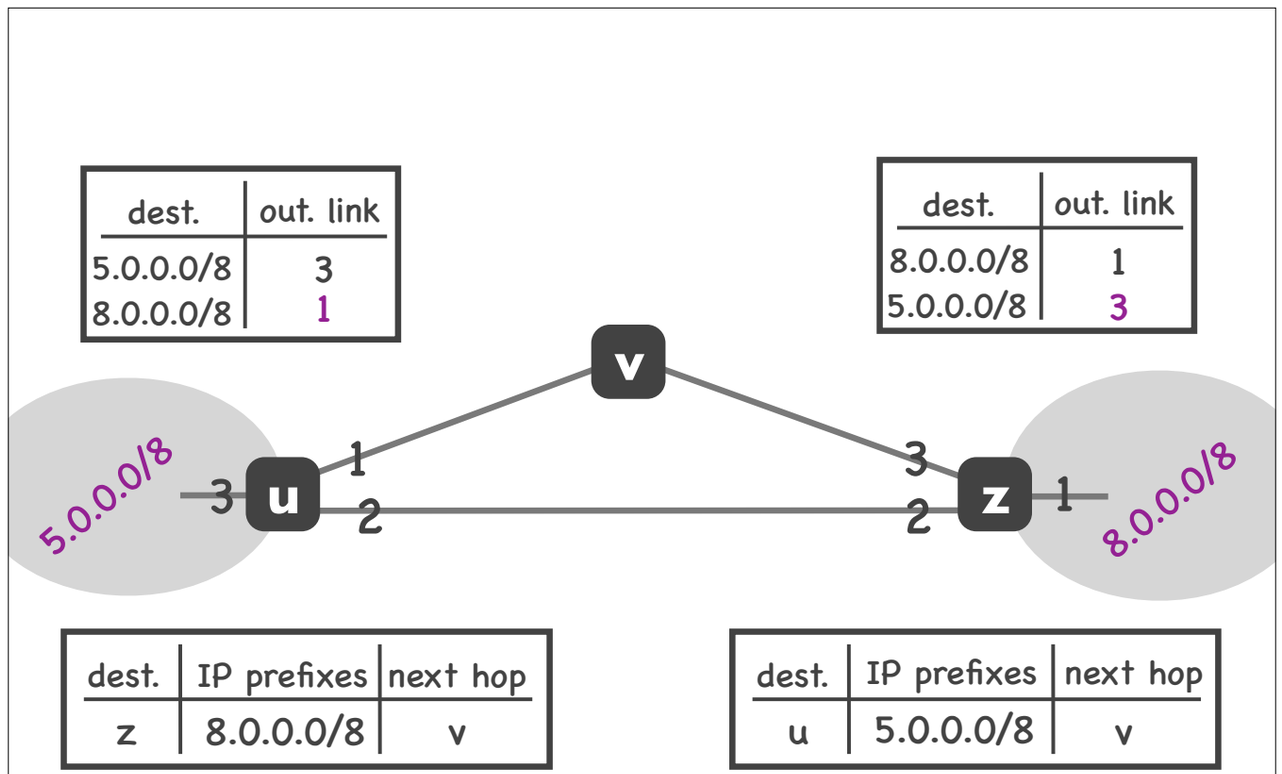
Which one is better and in what scenario? Which one do you think finishes faster? Which one requires more resources?

Link-state vs. distance-vector

- Link-state converges faster
  - each router starts with full picture of the network
- Distance-vector uses less bandwidth
  - each router only talks to its neighbors

Link-state converges faster: We can always reduce its computation time by running it on a faster computer. In contrast, in distance-vector, a new piece of information may take multiple rounds to propagate from router x to router y (as many rounds as there are routers between x and y).

But distance-vector requires fewer messages: In each round, each router exchanges a message with each neighbor (whatever the size of the network). In contrast, in link-state, each router essentially floods the entire network with information about its adjacent links.



Now let us go back to this picture that we introduced in the beginning of the lecture.

In this example, router u knows how to route packets to IP prefix 5.0.0.0/8, because this is the IP prefix of the local IP subnet, but it does not know how to route packets to IP prefix 8.0.0.0/8. How does routing help router u complete its forwarding table?

Router u, together with routers v and z participate in a routing protocol, which could be Dijkstra or Bellman-Ford or any other routing protocol. Through this routing protocol, router u learns that the best next hop to router z is router v. However, through this protocol, router u also learns that router z “owns” IP prefix 8.0.0.0/8, so, any packet whose destination IP address matches 8.0.0.0/8 should be routed to router z. Combining these two pieces of information, router u maps IP prefix 8.0.0.0/8 to output link 1, and it can now complete its forwarding table.

Similarly, router z, learns, through the same routing protocol, that the best next hop to router u is router v. And it also learns that router u owns IP prefix 5.0.0.0/8. Combining these two pieces of information, router z maps IP prefix 5.0.0.0/8 to output link 3, and it can now complete its forwarding table.

Internet routing challenges

- Scale
  - link-state would cause flooding
  - distance-vector would not converge
- Administrative autonomy
  - an ISP may not want to do least-cost routing
  - may want to hide its link costs from the world

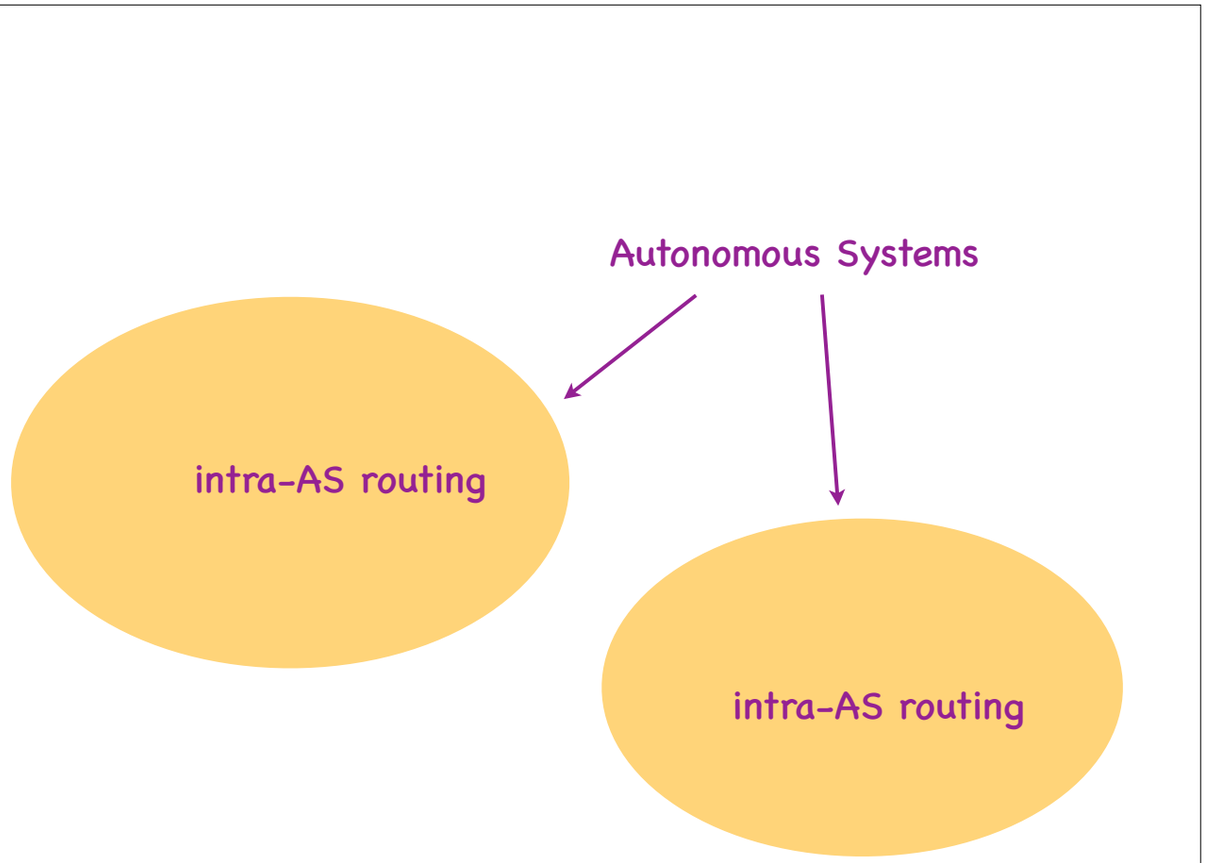
We will close our discussion on routing with how actual Internet routing works.

Designing a routing algorithm for the Internet faces at least two challenges:

One is scale:

- We are talking about the entire Internet -- millions of routers.
- A link-state routing algorithm, like Dijstra's algorithm, would cause flooding, because it requires each router to directly exchange information with every other router in the network.
- A distance-vector algorithm, like Bellman-Ford, would not converge fast enough, because it requires multiple rounds of information exchange between neighbors until every router collects all the information it needs from every other router in the network.

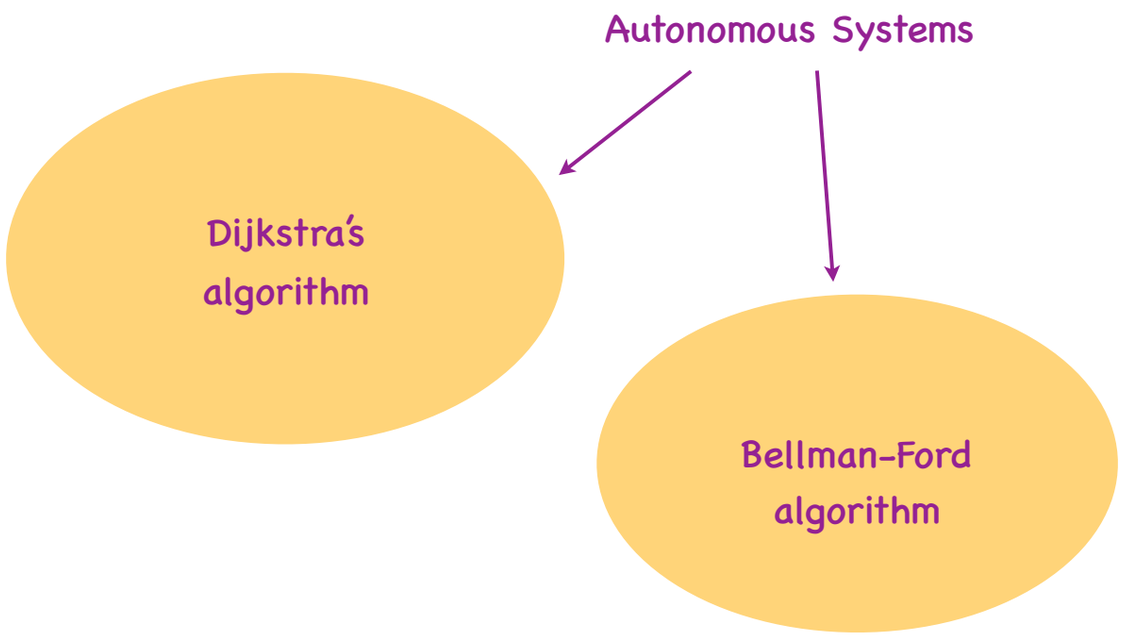
The other challenge is administrative autonomy: In the first lecture, we said that each Internet service provider, or ISP, is typically a separate administrative entity. So, an ISP may not want to apply least-cost path routing inside its own network; or, it may want to hide the characteristics of its internal links from the rest of the world.



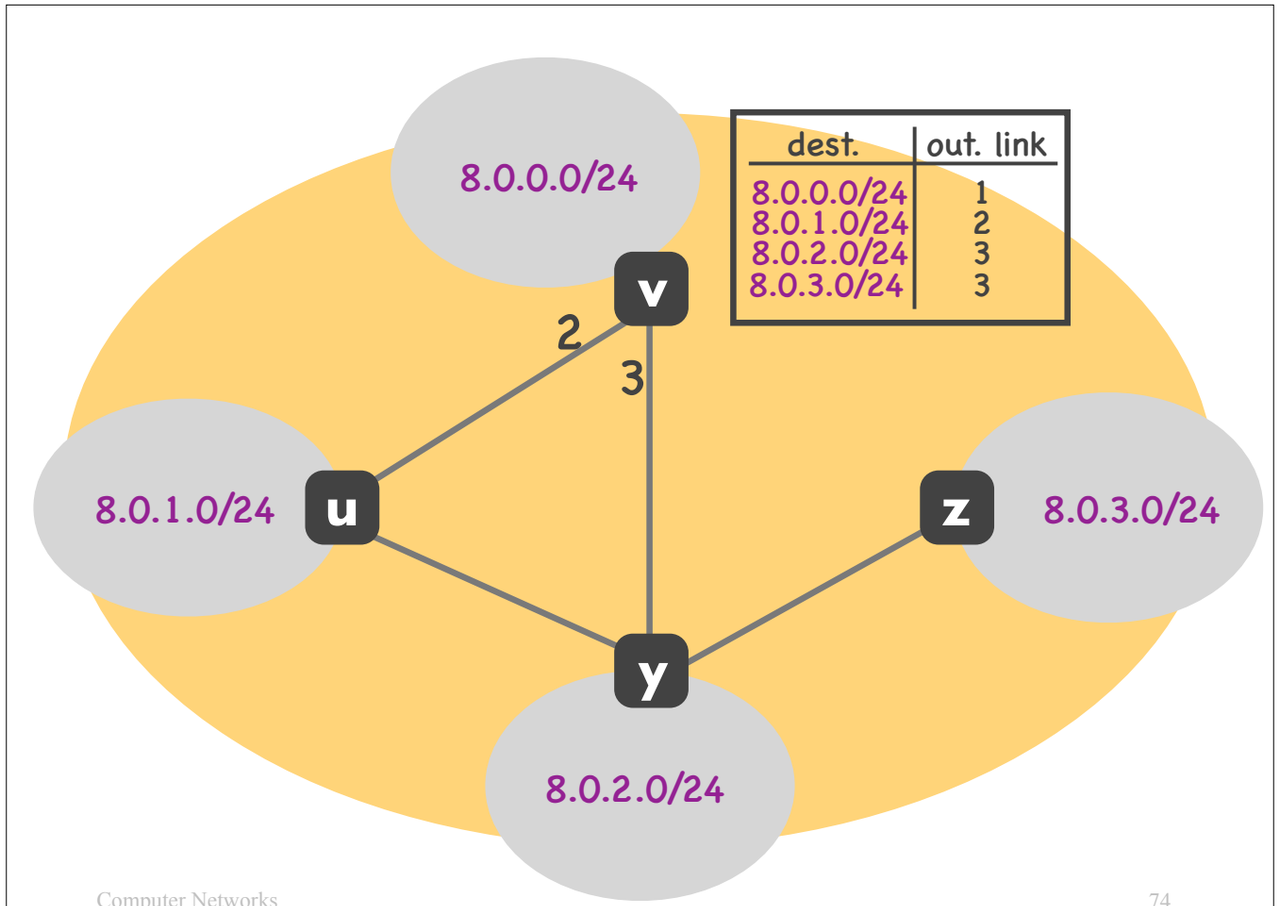
The current Internet architecture addresses these two challenges through *hierarchy*.

In particular, the Internet is divided in separate networks, which are called “Autonomous Systems” or ASes.

Each AS runs its own separate routing algorithm among its local routers. We refer to these algorithms as “intra-AS routing algorithms,” because each algorithm involves only routers from a single AS.



Each AS may run any intra-AS routing algorithm it wants, e.g., Dijkstra or Bellman-Ford.

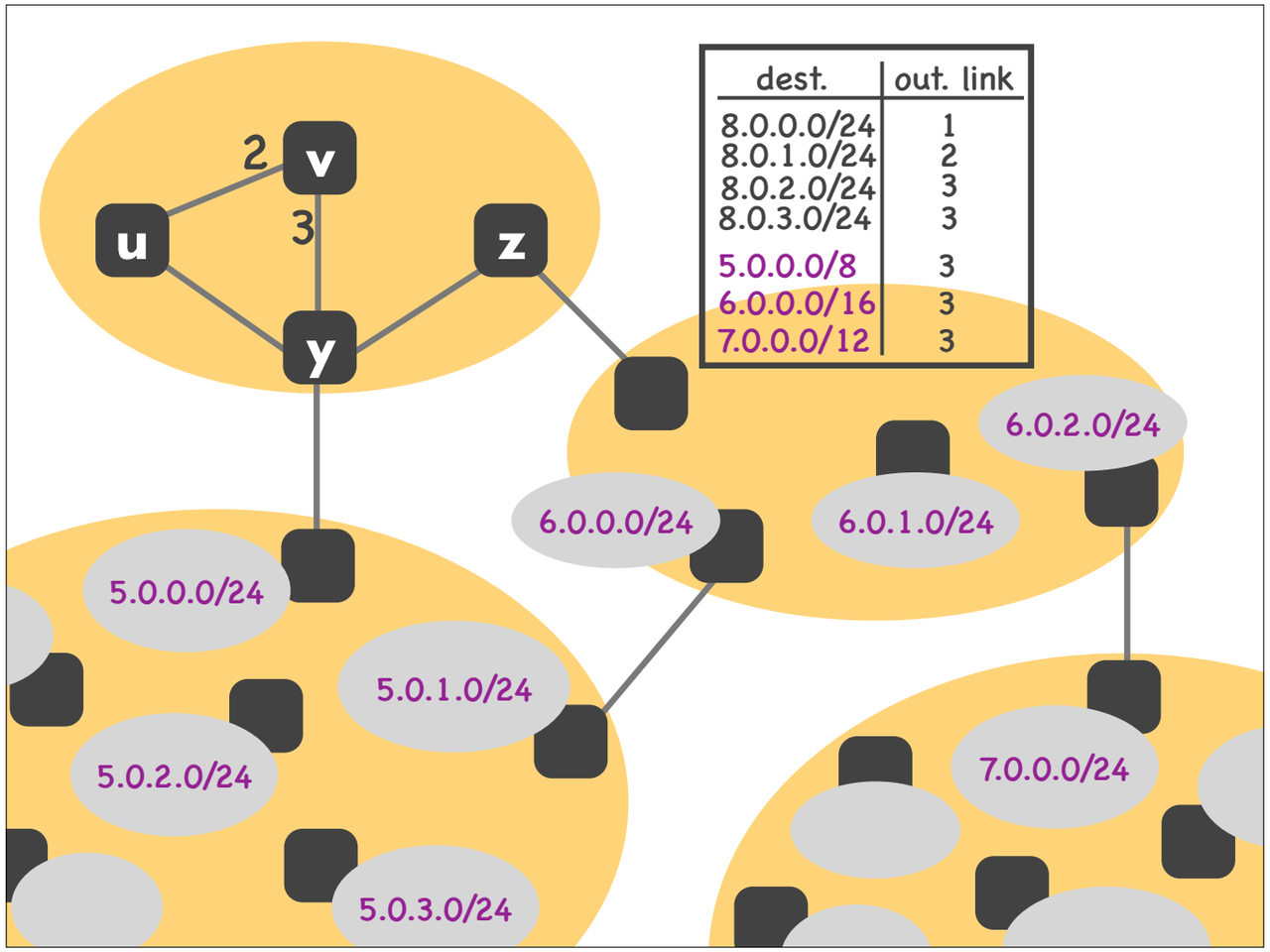


Consider an AS that has 4 routers, each providing connectivity to 1 IP subnet.

Each of the 4 routers must discover the best path to each local router/subnet.

They achieve this by participating in an intra-AS routing protocol.

The slide shows what the forwarding table of router v might look like.



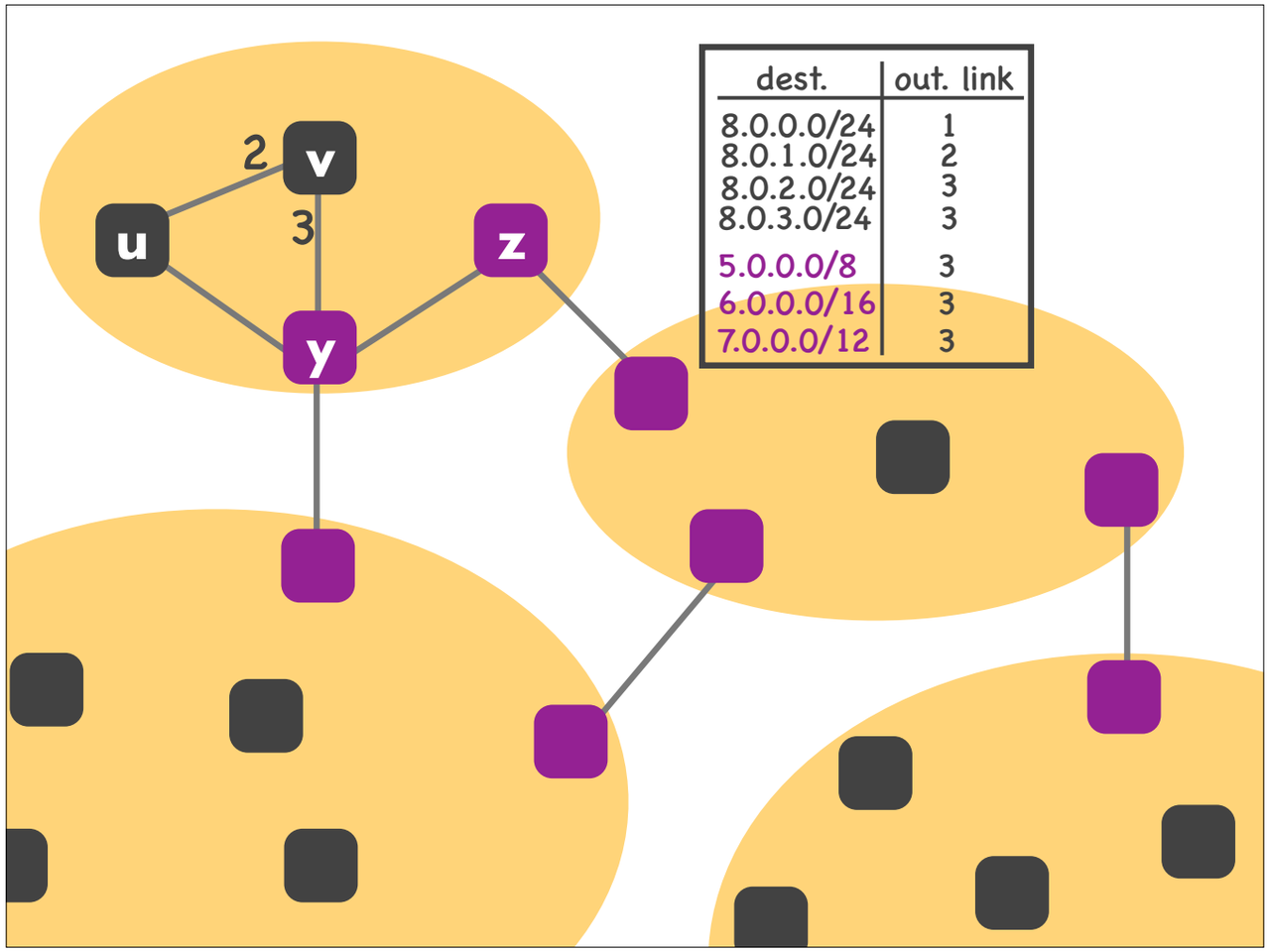
Moreover, each of the 4 routers must discover the best path, not only to each of the local routers but also to each foreign AS.

However, they do not need to discover a path to each individual IP subnet in each foreign AS.

For instance, in this example topology, router v (whose forwarding table is shown on the slide) needs only 1 entry overall for each of the 3 foreign ASes shown in the picture.

This is an important way in which Internet routing scales: A router does not need an entry, in its forwarding table, for every single IP subnet on the entire Internet. It typically merges the many little IP prefixes of each foreign AS into one big IP prefix or a few big IP prefixes.

We said that a router discover routes to the local routers through intra-AS routing.
How does it discover routes to foreign ASes?
That's done through…



…the inter-domain routing algorithm/protocol.

This is a routing algorithm run by all the *border routers* of all ASes (purple routers in the picture).

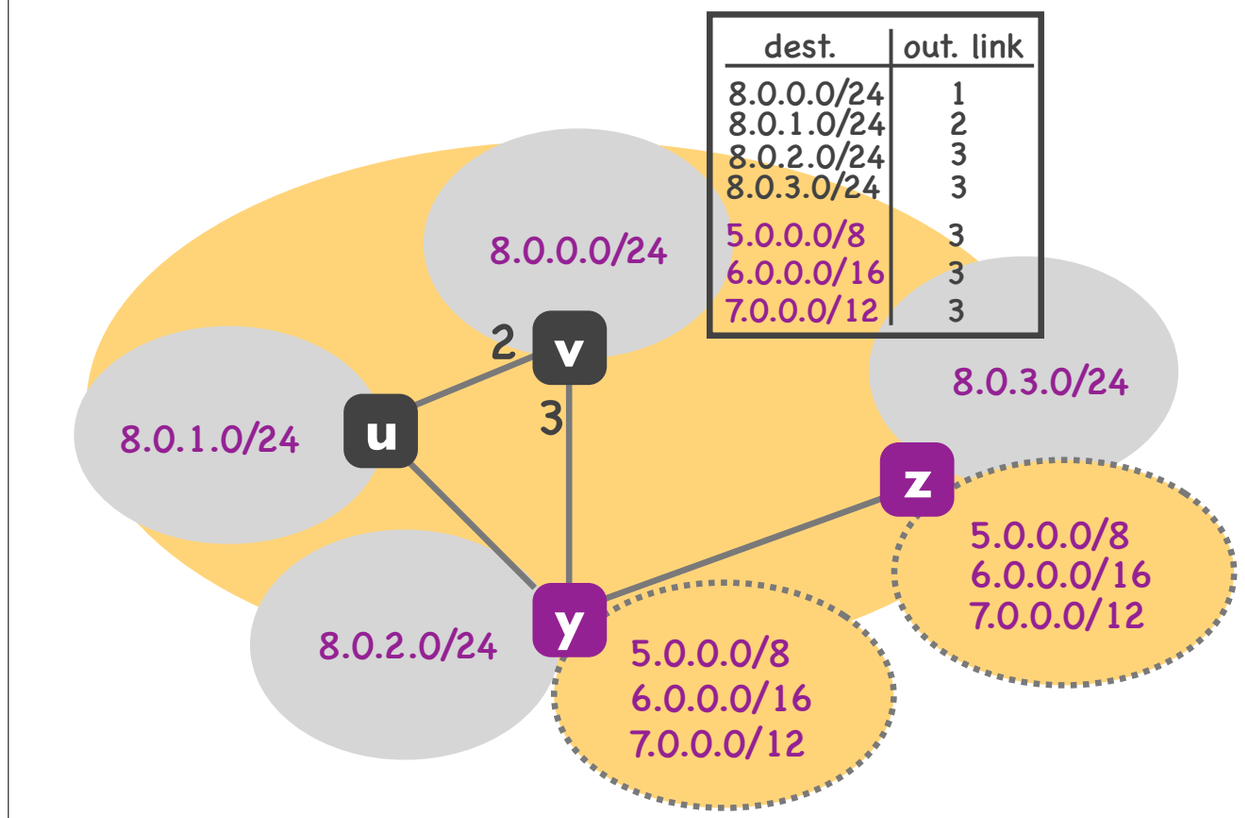
The border routers are the routers that are at the “edge” of each AS and are directly connected to (border) routers of other ASes.

All the border routers participate in an instance of a routing algorithm, called Border Gateway Protocol (BGP), which is a variant of the Bellman-Ford algorithm we saw earlier.

Through BGP, each border router (a) advertizes prefixes from its own AS to the outside world, and (b) it learns the best route to each foreign prefix.

In our example, routers y and z are border routers, so they learn (through BGP) the best route to each of the 3 foreign prefixes.

Then they “inject” these routes into their local AS through intra-domain routing.



From the point of view if router v:

v participates only in intra-domain routing (it's not a border router). So, it learns the best route to each local prefix (from the other local routers) AND it learns the best route to each foreign prefix (from the border routers, which are y and z).

If both y and z have routes to the same prefix, v will choose the least-cost route.

In a way, as local routers act as gateways to local subnets/prefix, border routers act as gateways to foreign ASes/prefixes.

(It is possible for a router to do both. E.g., y is the gateway to 8.0.2.0/24 AND to foreign prefixes.)

Internet routing

- Each router learns one route
  to each IP subnet in local AS
- Each router learns one or a few routes
  to each foreign AS
  - but not one route to each IP subnet
    of each foreign AS

Intra-AS routing

- Run by all routers in the same AS
- Goal: propagate routes within local AS
  - each router advertizes routes to its local IP subnets
  - and potentially routes to other ASes that it has learned through BGP
- OSPF, RIP, …

Inter-AS routing

- Run by all border routers between ASes
- Goal: propagate routes outside local AS
  - each border router talks to external neighbors (eBGP)
  - and to the other border routers of the local AS (iBGP)
- BGP = Border Gateway Protocol

There is 1 inter-AS routing protocol on the Internet: BGP.

There has to be 1, if we want to ensure that all ASes can communicate with each other, i.e., discover paths to each other.

Internet routing challenges

- Scale
  - link-state would cause flooding
  - distance-vector would not converge
- Administrative autonomy
  - an ISP may not want to do least-cost routing
  - may want to hide its link costs from the world

I said earlier that Internet routing faces at least two challenges: scale and administrative autonomy. I would like you to think: how exactly does the approach I just described address these two challenges? How does the separation in ASes and intra-AS and inter-AS routing protocols address the problems of scale and administrative autonomy? Think about it and we will discuss it on Friday.

Solution: hierarchy

- Scale: state not per IP subnet
  - each router needs forwarding entries per local IP subnets and foreign IP prefixes
  - each router may communicate with all other local routers and external neighbors
- Administrative autonomy: each AS chooses its own intra-AS routing protocol