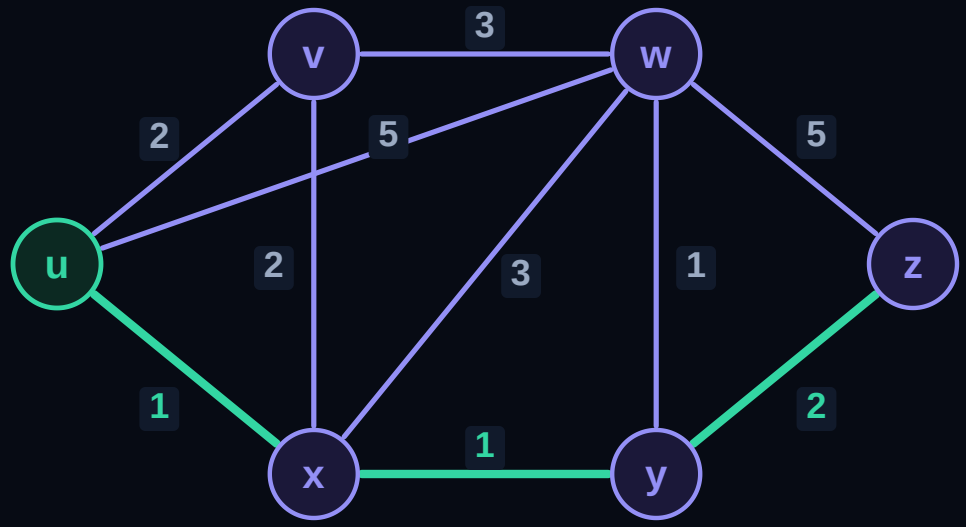
The control plane Link-state routing

Chapter 4 read the forwarding table in hardware, once per packet. This lecture is about who writes it: why the rows cannot be typed in, what the rows at different routers must satisfy together, and the first way routers compute them: every router learns the whole network and runs the same algorithm.

Yaxiong Xie

1,078,012

rows in a core router’s forwarding table

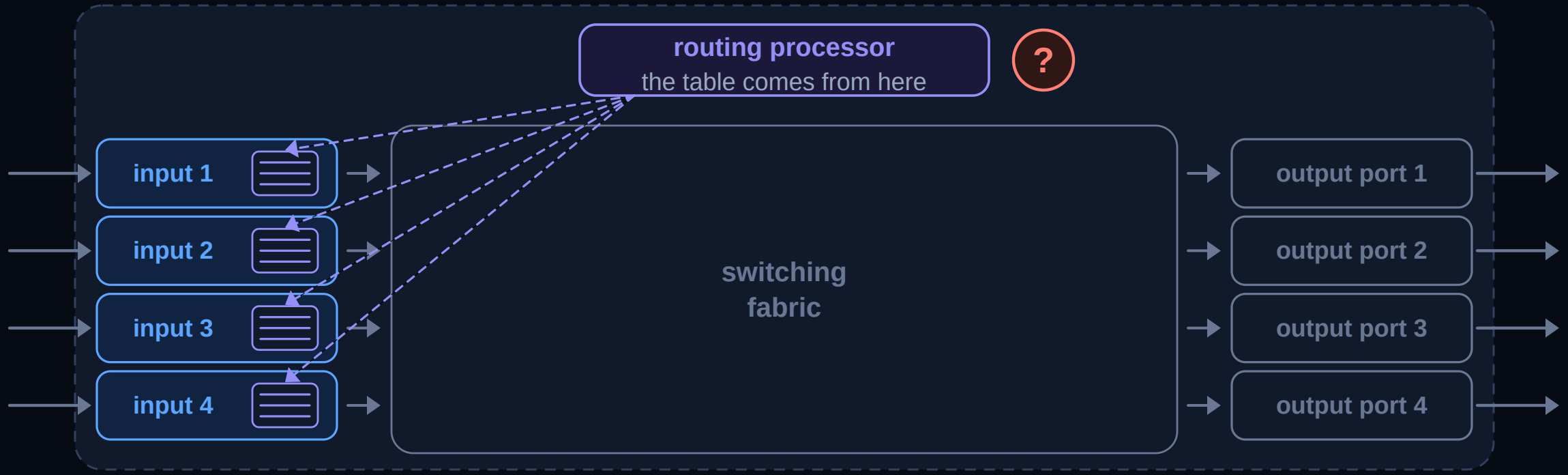


every router learns the whole network

and computes its own least-cost paths

the table the data plane reads

Who writes the table the router reads?



The last page of the Router lecture: the routing processor, with a question mark on its table.

Chapter 4 read the table, in hardware. Chapter 5: who writes it.

How many rows does that table hold?

1,078,012

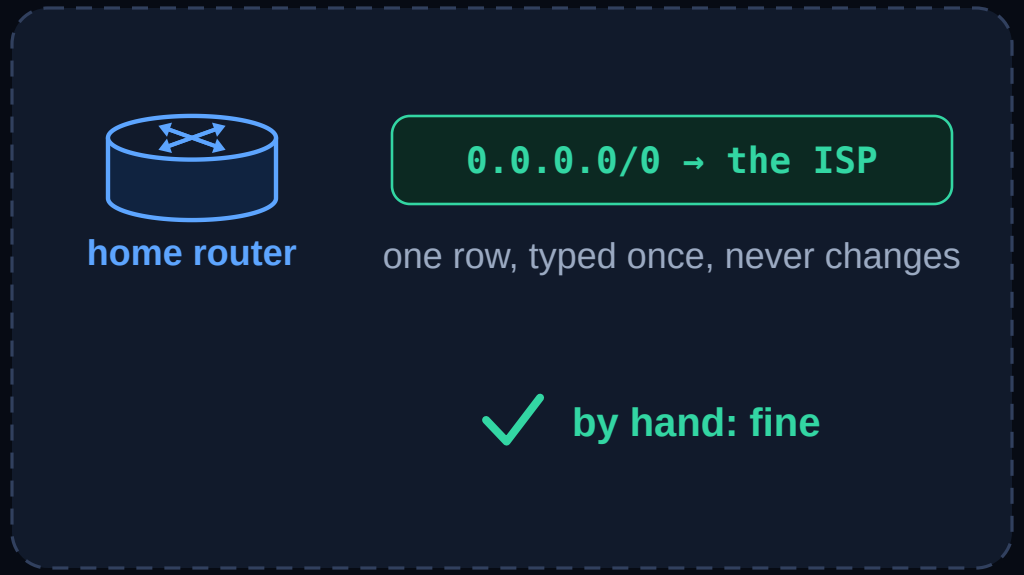
IPv4 prefixes in the global routing table

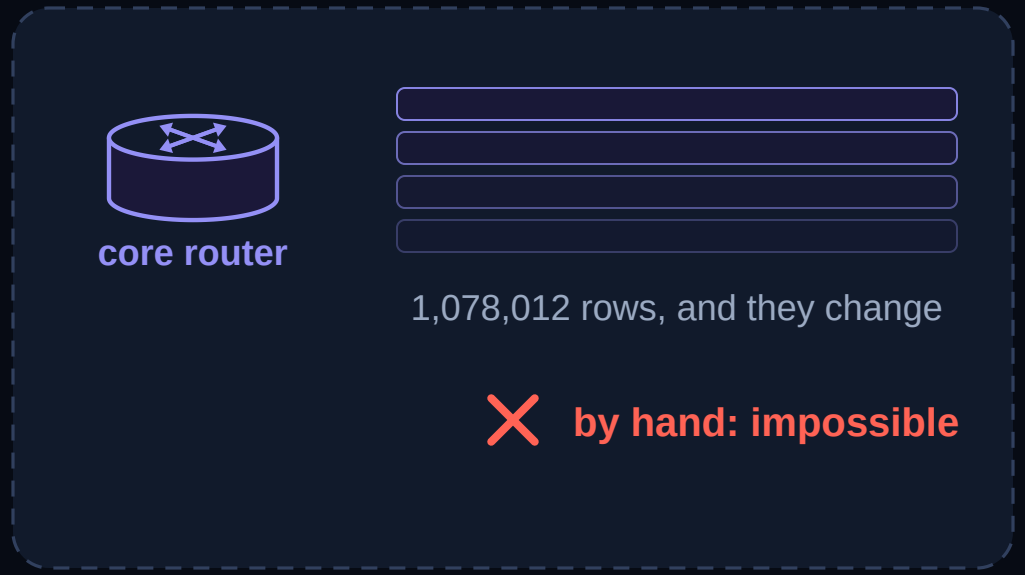
79,444	one row each	no default route
networks (autonomous systems)	in a router at the core	nothing to fall back on

cidr-report.org, 23 Sep 2026. A core router has no default route, so every prefix needs its own row.

A core router holds a row for each of about 1.08 million prefixes.

Could an operator type the rows in?

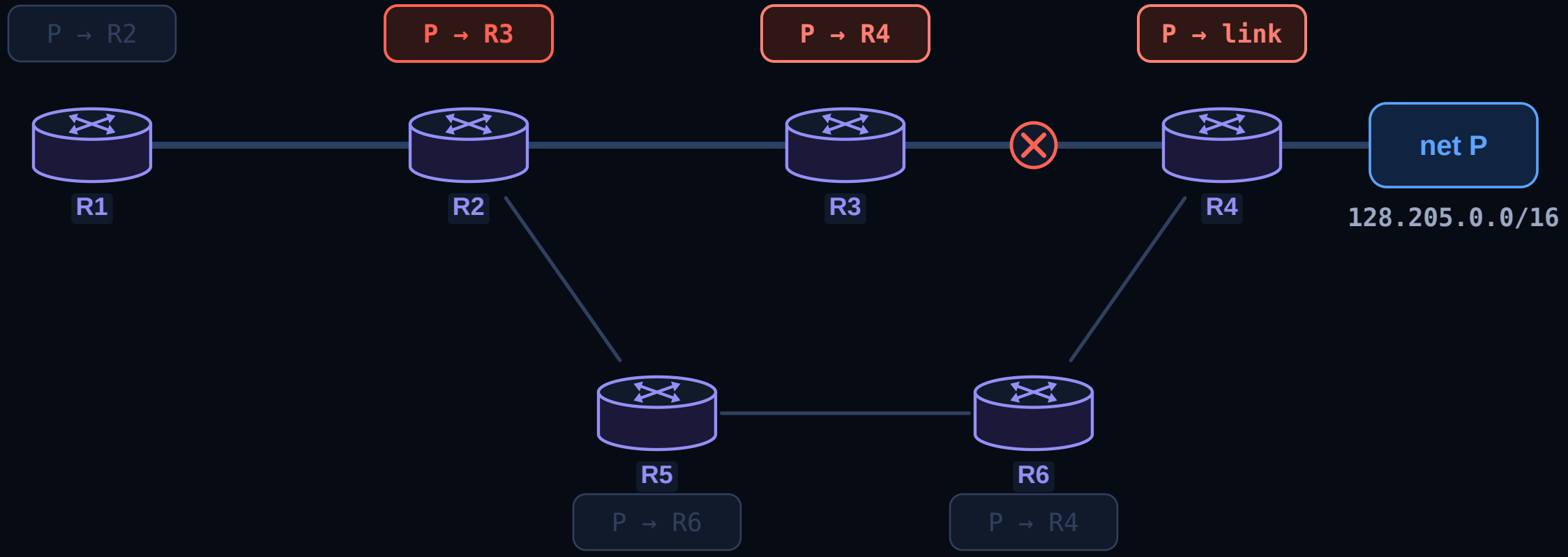




And they change: somewhere on the internet a link fails or comes back every few seconds.

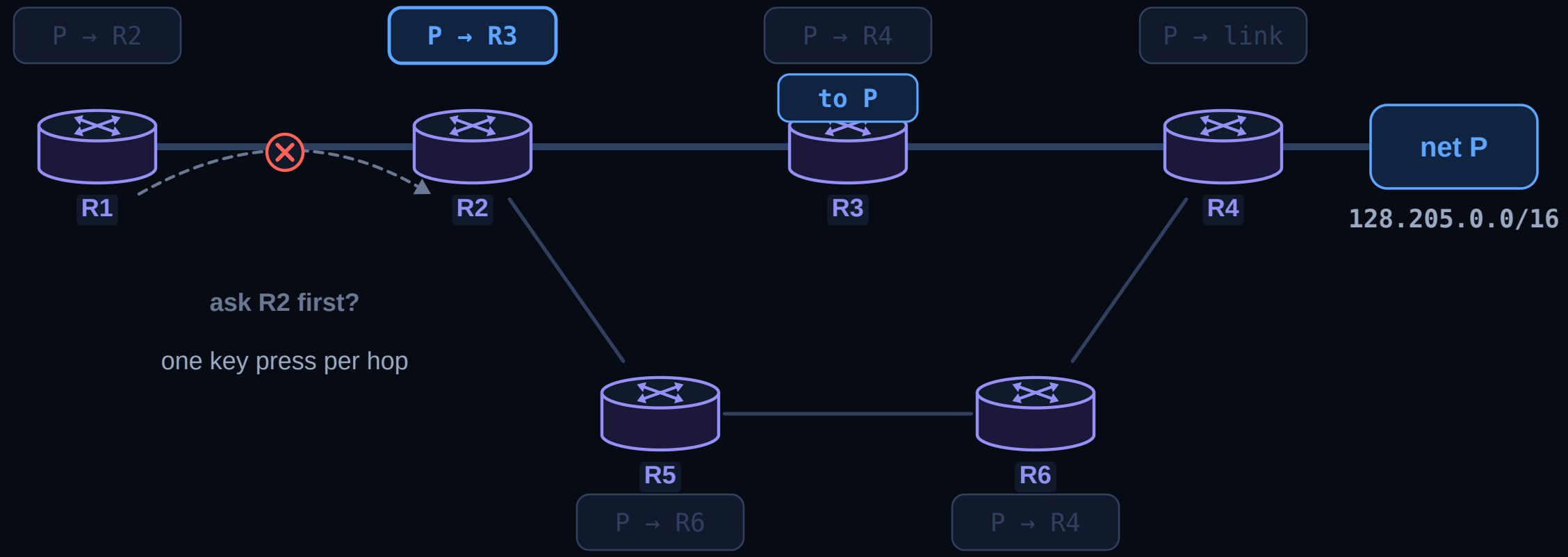
By hand works for a few rows that never change. The core has a million.

What happens to the rows when a link fails?



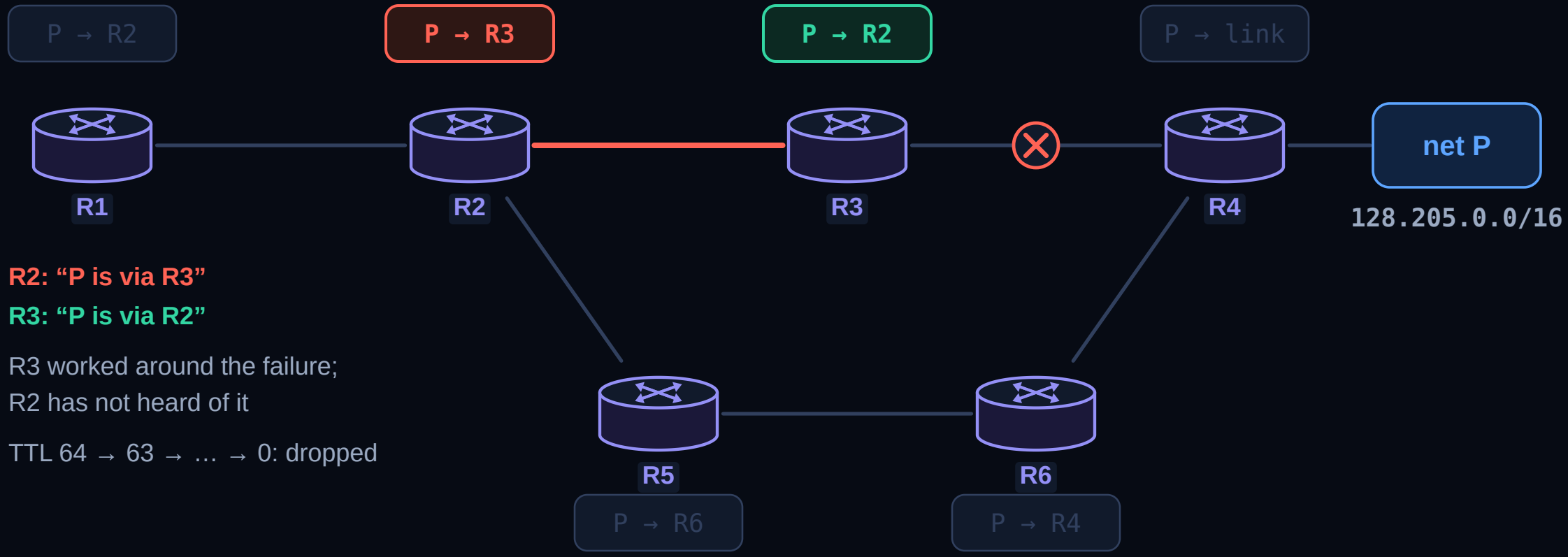
Only the two ends see it. R2’s row is now wrong, and nothing told R2.

Does a router check with the next one before it forwards?



No. Each router reads its own row; the path is what the rows add up to.

Can every row look right and the path still fail?



R2: “P is via R3”

R3: “P is via R2”

R3 worked around the failure;
R2 has not heard of it

TTL 64 → 63 → … → 0: dropped

Rows must agree with each other. TTL discards the packet; it fixes no row.

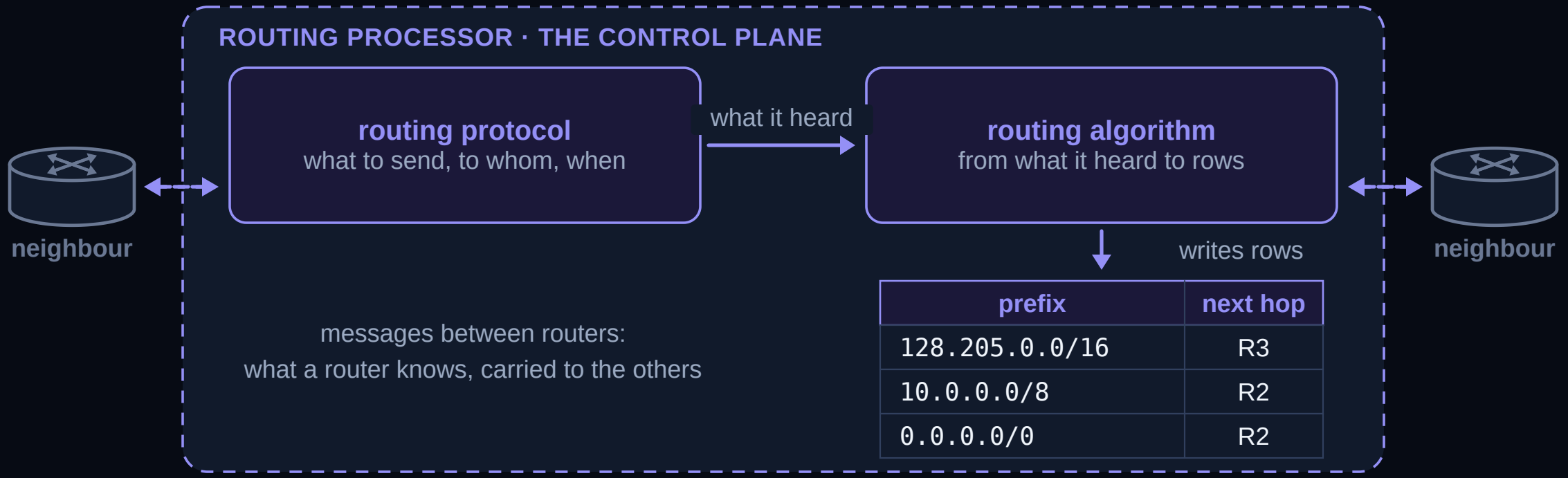
What must the control plane do, then?

1. **a row for every destination** — at every router: a million rows, each
2. **rows that agree across routers** — so the hops add up to a path, not a loop
3. **all of it again after every change** — a link fails, a link is added, a cost changes

And nobody types.

Every router, every destination, consistent, redone after every change.

Two parts: talk, then compute



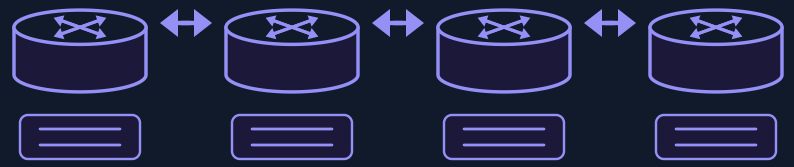
Together they are routing, and the software that runs them is the control plane.

A protocol carries information between routers; an algorithm makes rows.

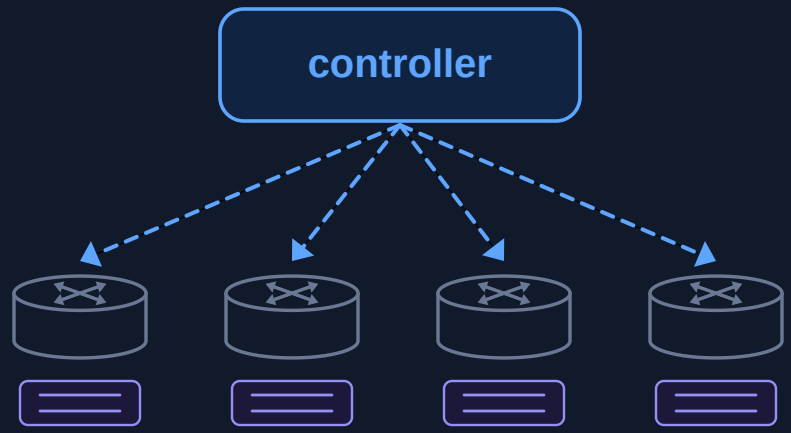
Where does the computation run?

every router computes its own rows

routers talk to each other
and each runs the algorithm



one controller computes every table



The right side is logically centralized control, the idea behind software-defined networking.

The data plane is the same either way. This chapter: the left side.

Five lectures, one question each

L1	**every router learns the whole network**	link state: today
L2	**routers share only totals, with neighbours**	distance vector
L3	**routing inside one organisation’s network**	OSPF
L4	**routing between organisations**	BGP
L5	**who tells whom, and what goes wrong**	policy, hijacks

The same job, at growing scale and with less trust.

Today: five points

01



The problem

what exactly to compute

02



Flooding

the whole map to every router

03



Dijkstra

least-cost paths from the map

04



Tree to table

one link per row and why it works

05



Load costs

when the cost follows traffic

Each from its own problem: why it is needed, how it works, what it costs.

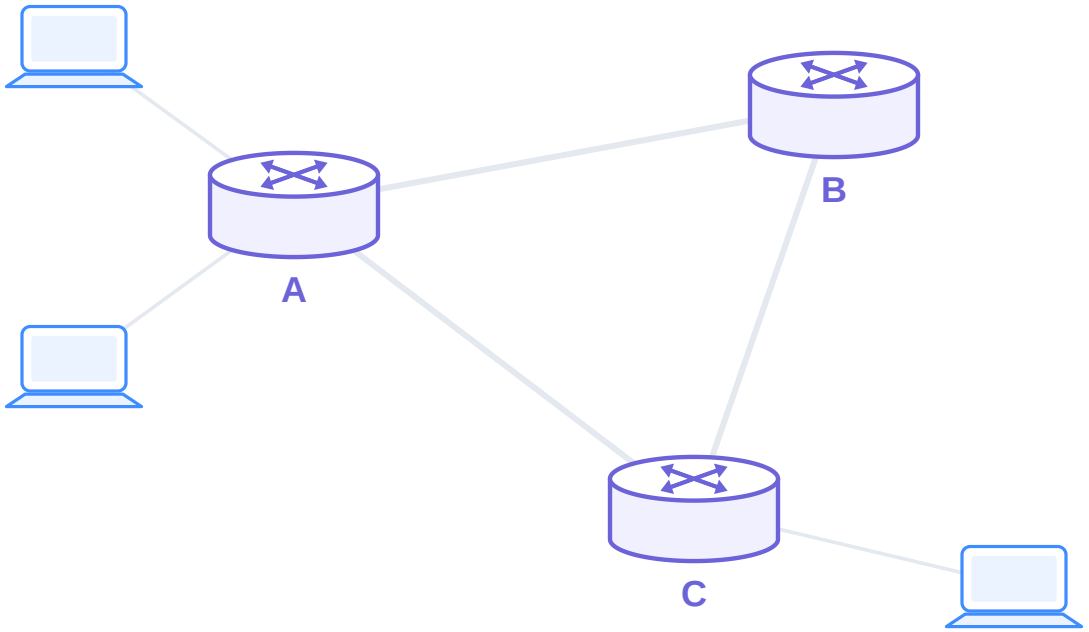
What exactly should the rows say?

01

The problem

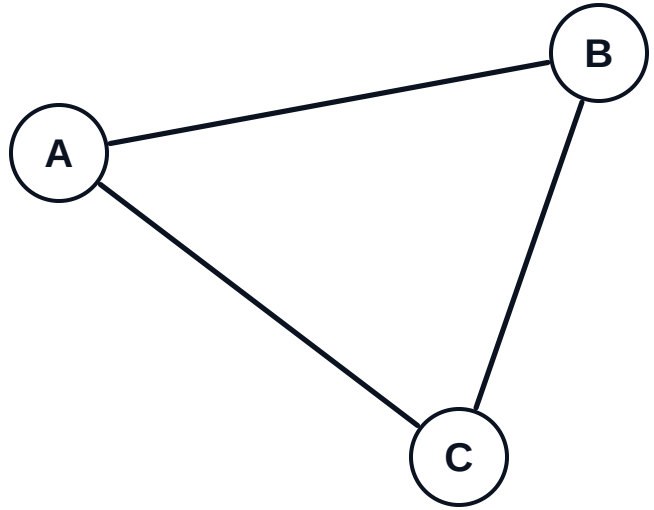
Many sets of rows deliver packets. To pick one, a program needs a picture of the network and a meaning for “best”.

What does the algorithm see of the network?



routers, fibre, hosts on a LAN

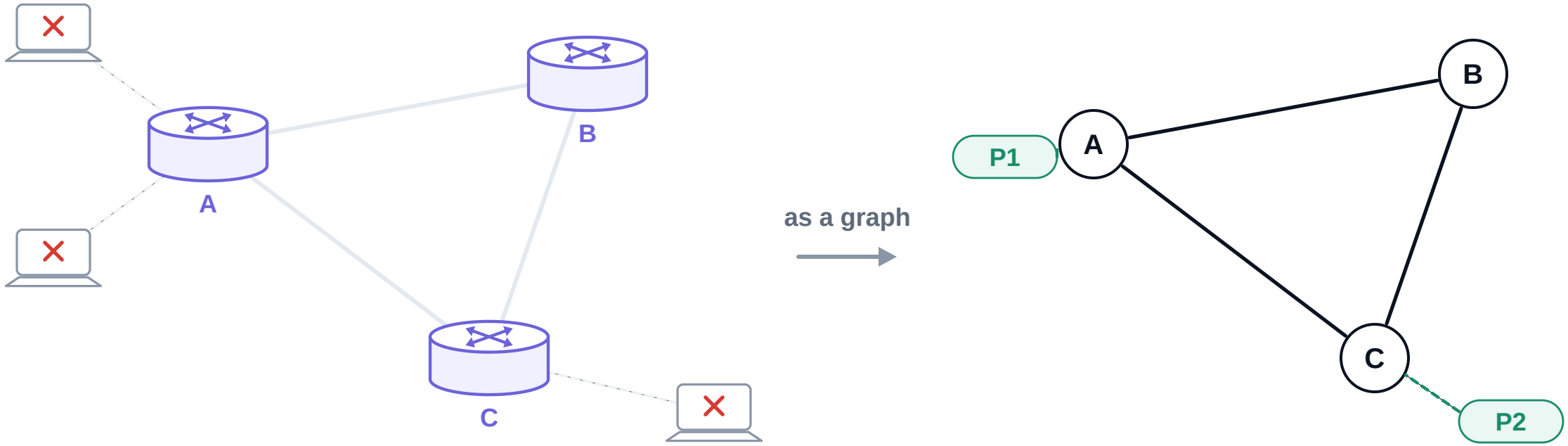
as a graph



nodes and edges: G = (N, E)

A graph G = (N, E): routers are the nodes, links are the edges.

Where are the destinations in this graph?



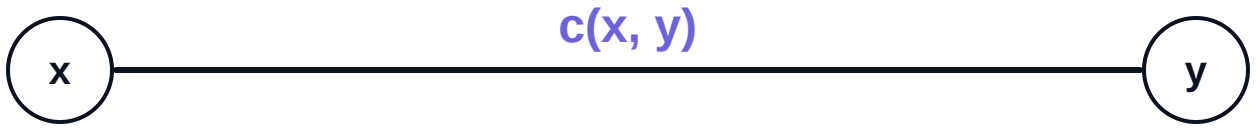
routers, fibre, hosts on a LAN

nodes, edges, and a prefix on its router

The Forwarding lecture: every row is a prefix. Hosts are not nodes; the router they hang off is.

A destination is a prefix on a router. Reach the router, reach the prefix.

What does a link cost?



a number the operator assigns to the link

every link 1



hop count

inversely proportional to capacity



1 Gbps: 10 · 10 Gbps: 1

measured delay

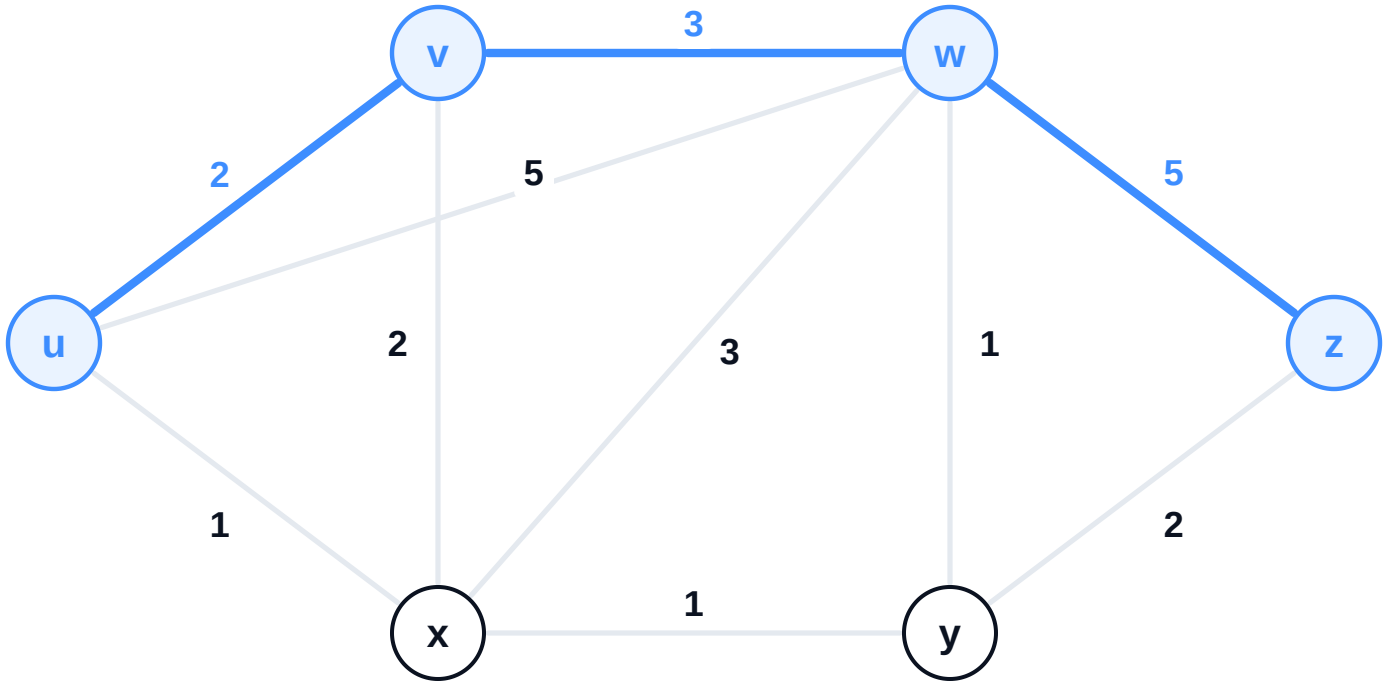


in milliseconds, right now

Each direction has its own cost, so c(x, y) and c(y, x) may differ.

c(x, y) is a number the operator chooses; the algorithm only adds costs up.

What does a path cost?



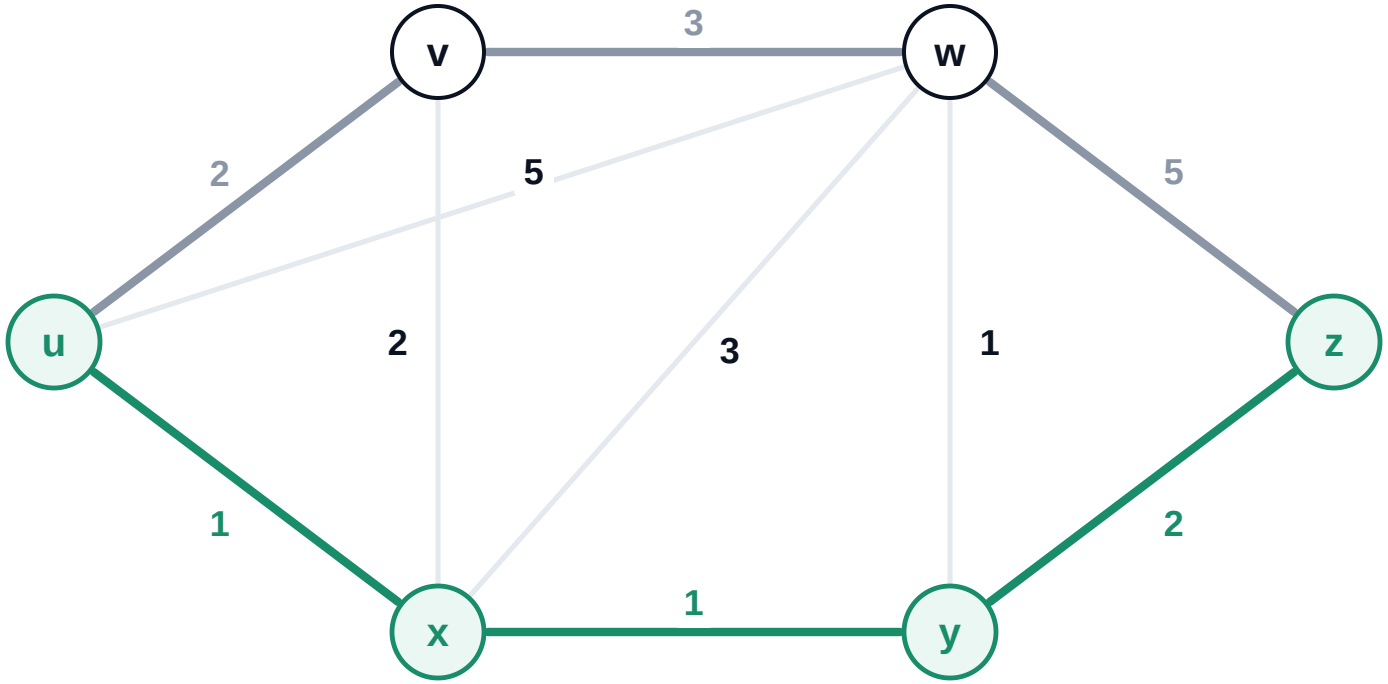
u → v → w → z

2 + 3 + 5 = 10

one number per link, added up

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

Which path from u to z should packets take?



u → v → w → z

2 + 3 + 5 = 10

u → x → y → z

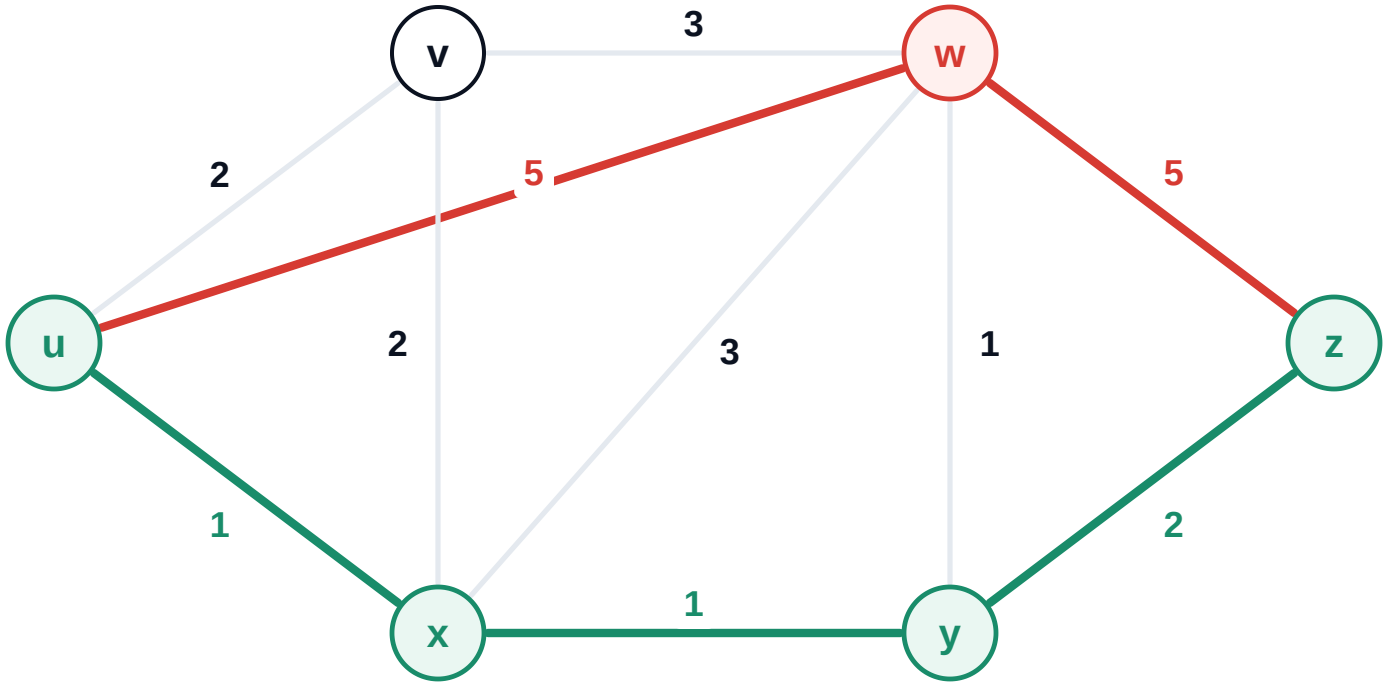
1 + 1 + 2 = 4

the least-cost path
from u to z

Finding the least-cost path for every pair of routers is the routing problem.

The one with the smallest sum: the least-cost path.

Is the path with the fewest links the cheapest?



two links: u → w → z

5 + 5 = 10

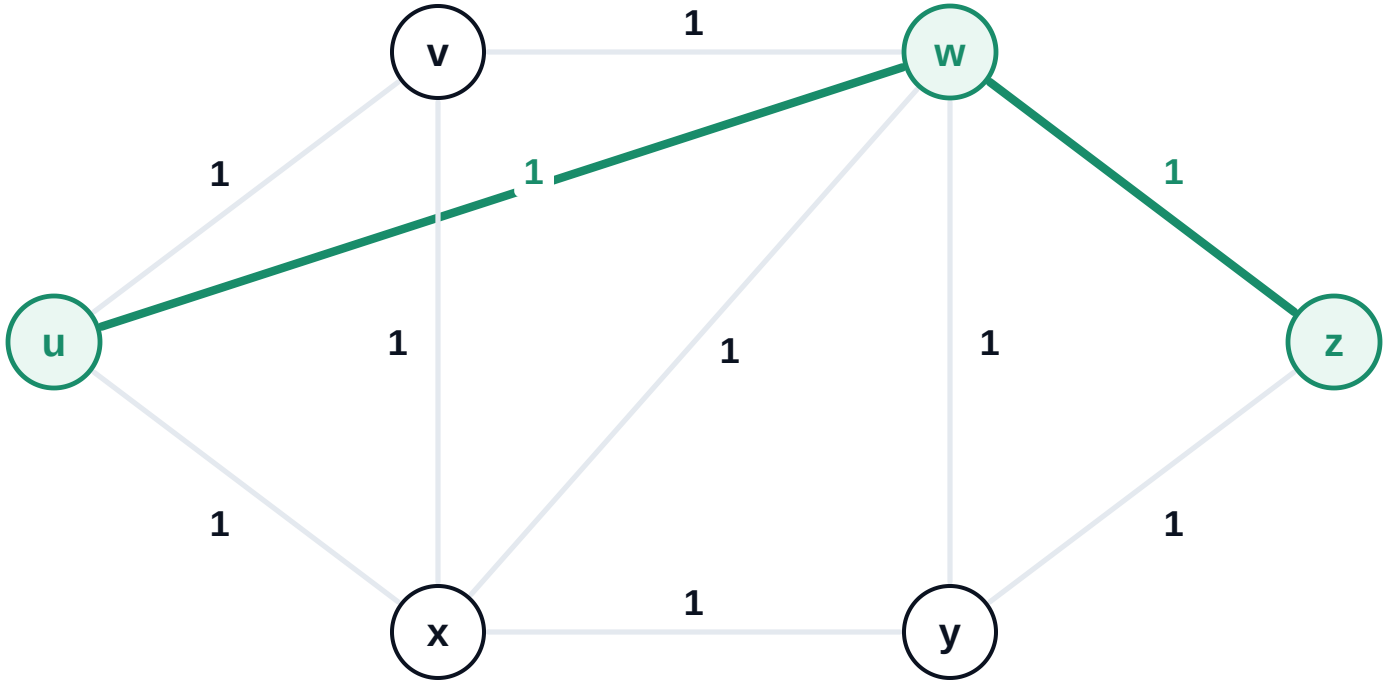
three links: u → x → y → z

1 + 1 + 2 = 4

more links, lower cost

No. Three cheap links can cost less than two expensive ones.

When is the fewest links also the cheapest?



every link costs 1

u → w → z

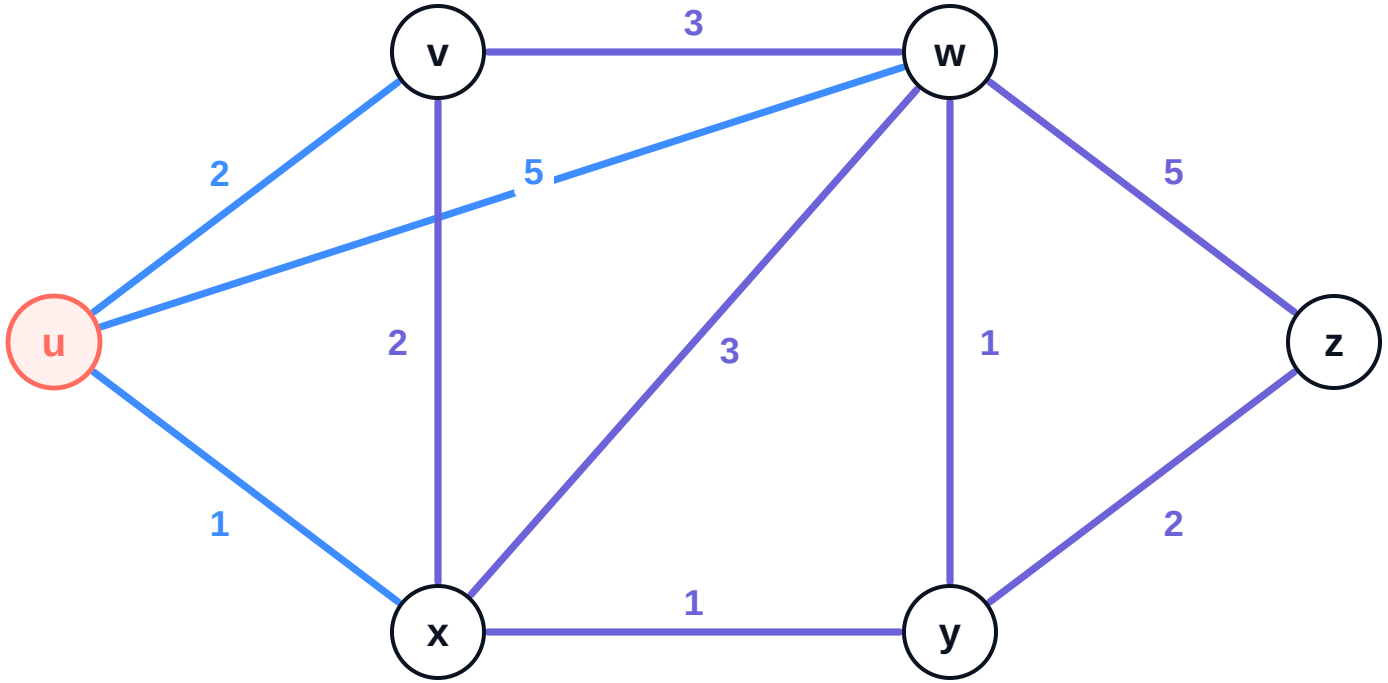
1 + 1 = 2

u → x → y → z

1 + 1 + 1 = 3

When every link costs the same. A special case, not a different rule.

What must u know to find the least-cost path?



u's own links: 3

`(u,v) (u,x) (u,w)`

links u is not on: 7

`(v,x) (v,w) (x,w) (x,y) (w,y) (w,z) (y,z)`

any of them may lie on the best path;
u cannot tell which until it has computed

This lecture: every router learns every link (link state). Next lecture: every router learns only its neighbours' totals (distance vector).

Every link in the graph and its cost, not only the links u is on.

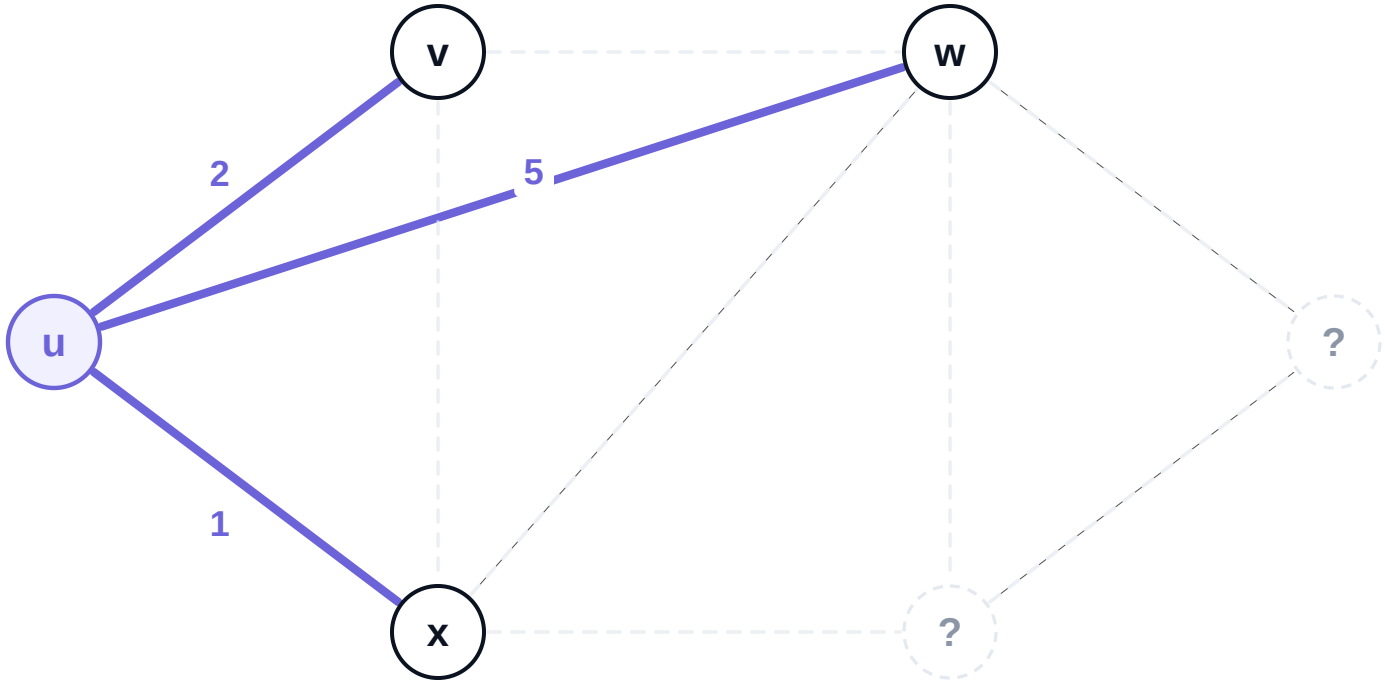
How does u get the whole graph?

02

Flooding

u knows its own three links. Every other link in the graph, it has to learn from someone.

What does u know at time zero?



u knows, first-hand:

```
(u,v) = 2
(u,x) = 1
(u,w) = 5
```

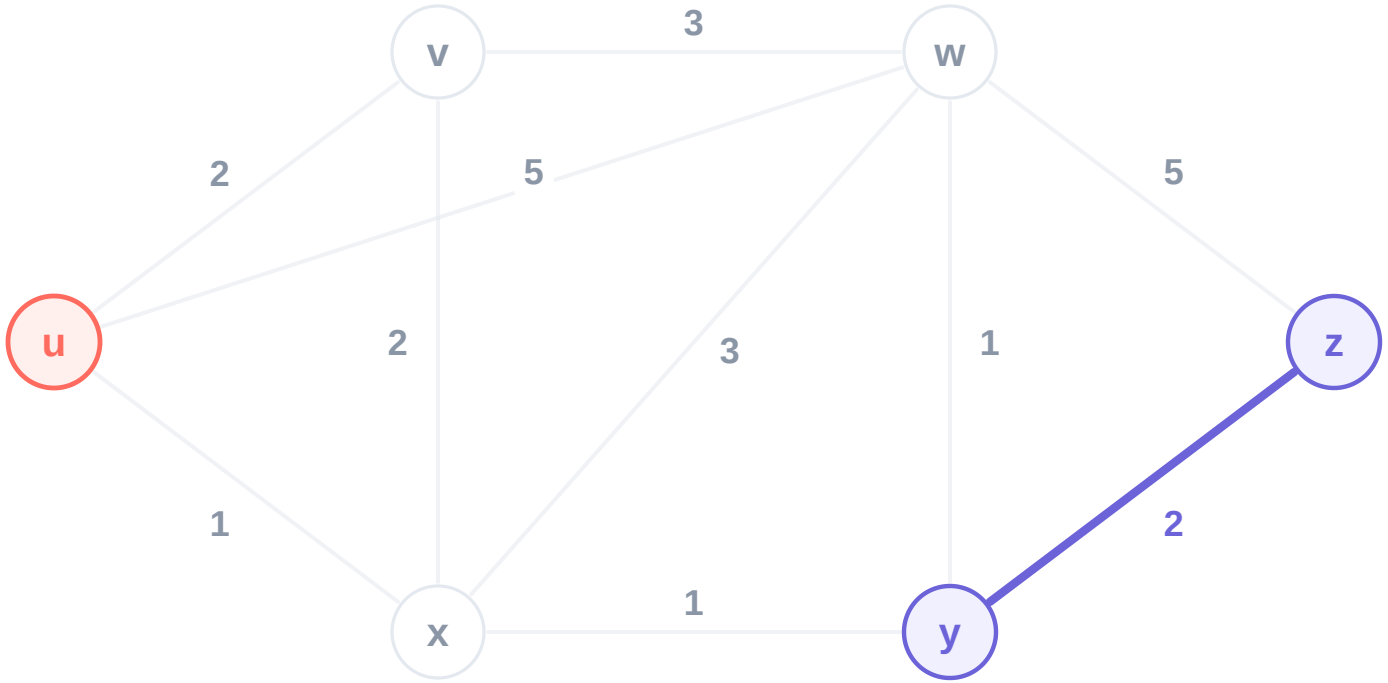
and the prefixes attached to u

the other seven links: nothing
not even that y and z exist

Costs are configured per interface. Hello messages say which neighbours are alive: OSPF sends one every 10 s and declares a silent neighbour down after 40 s.

Its own links, their costs, its prefixes. Nothing beyond one hop.

Who knows the link (y, z)?



first-hand: y and z

the two ends of the link

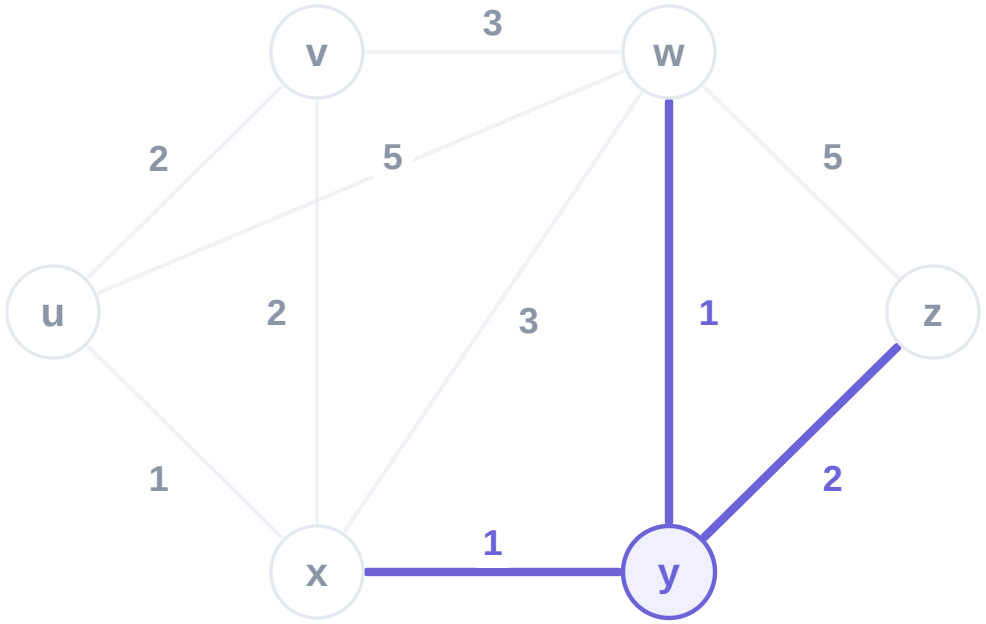
nobody else has seen it

not x, not w, and not u

so u can learn (y,z) only from y or from z

Only its two ends. Every link is known first-hand by exactly two routers.

What should y send?



no paths, no totals: nothing y has computed

from y

```
(x, 1)
(w, 1)
(z, 2)
```

prefixes on y

the origin
whose links these are

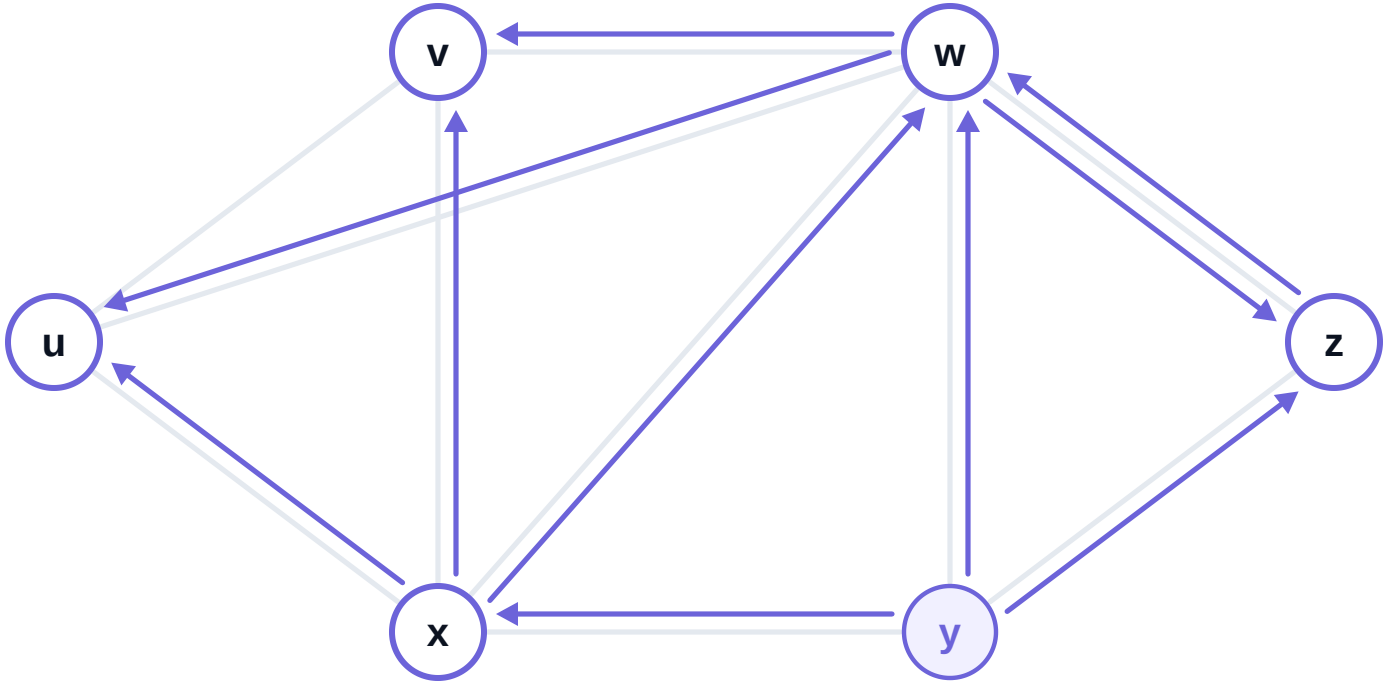
its links, with costs
one line per neighbour

the prefixes attached to it
the destinations behind y

This message is a link-state advertisement (OSPF: an LSA). Every router makes one for its own links.

Only what it knows first-hand: its own links and costs, and its prefixes.

How does y's advertisement reach u?



the rule at every router

1. store the copy
2. send it on every link

except the one it came in on

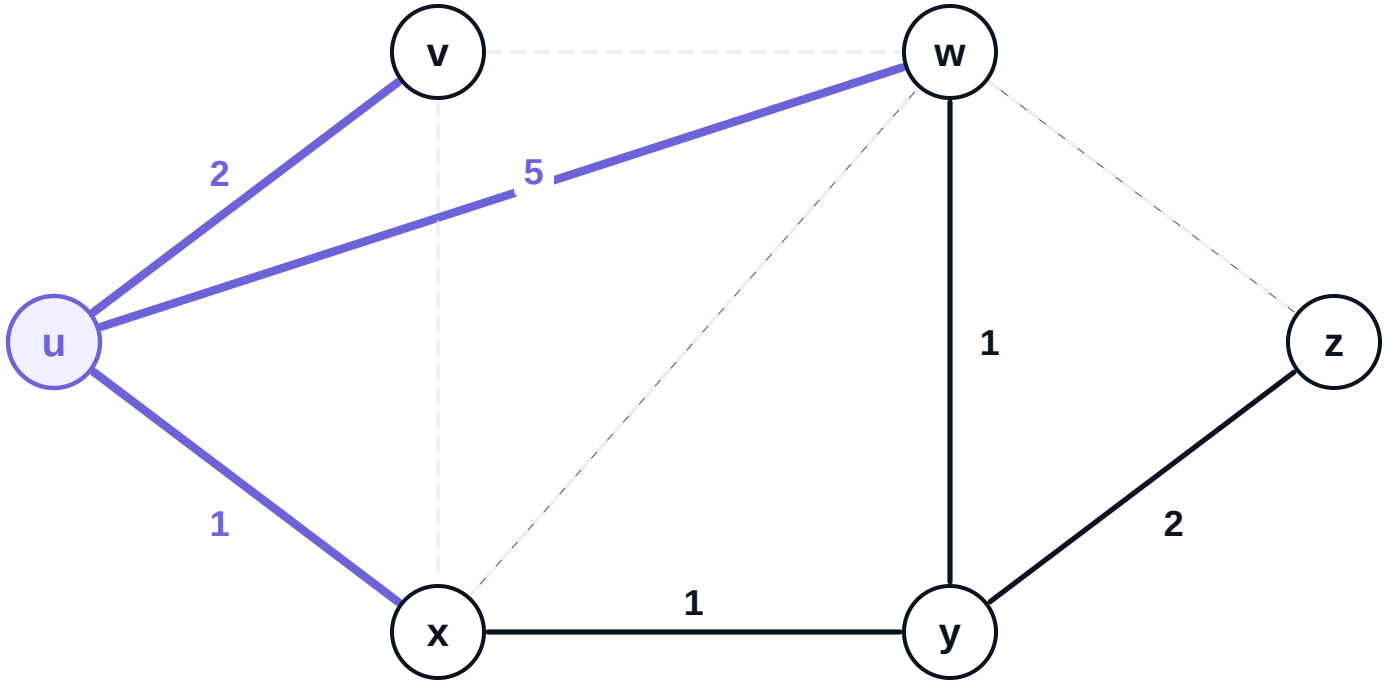
press 1: y's three neighbours

press 2: their neighbours, so u

u is two hops from y, and gets the copy twice: once from x, once from w.

Every router that receives it stores a copy and passes it on. This is flooding.

What does u hold now?



u's picture of the graph

- from u
- from y

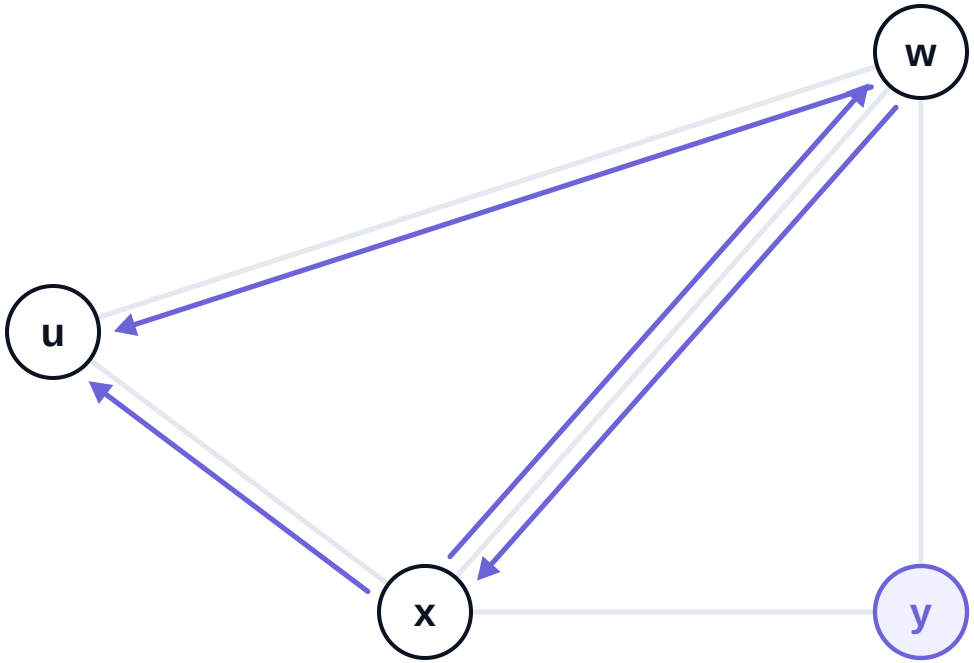
3 links from u itself
+ 3 links from y
and now u knows y and z exist

6 of 10 links known

u learns the cost of (x,y) from y, not from x: each advertisement carries its origin's links only.

Its own links, plus y's. One advertisement fills in one router's links.

Does the flood stop by itself?



one key press = the rule, applied once

1. y sends on both its links

2. x and w forward on every other link: to each other, to u

3. the copies from step 2 arrive: x and w forward them again

4. y gets its own advertisement back, twice, and forwards both

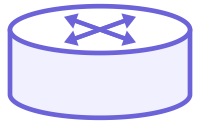
5. this is step 2 again

then 3, then 4, then 2 again: forever

The rule never says “drop”. Every copy that comes back is forwarded again, so u receives the same advertisement forever.

No. The graph has cycles: copies go round and round, and nothing drops them.

How can w tell a copy it has already seen?



w

what w has stored

origin	newest seq
y	12
u	7
x	31

from y ✕

```
(x, 1)
(w, 1)
(z, 2)
seq  12
```

seq 12 is not newer than 12

dropped, not forwarded

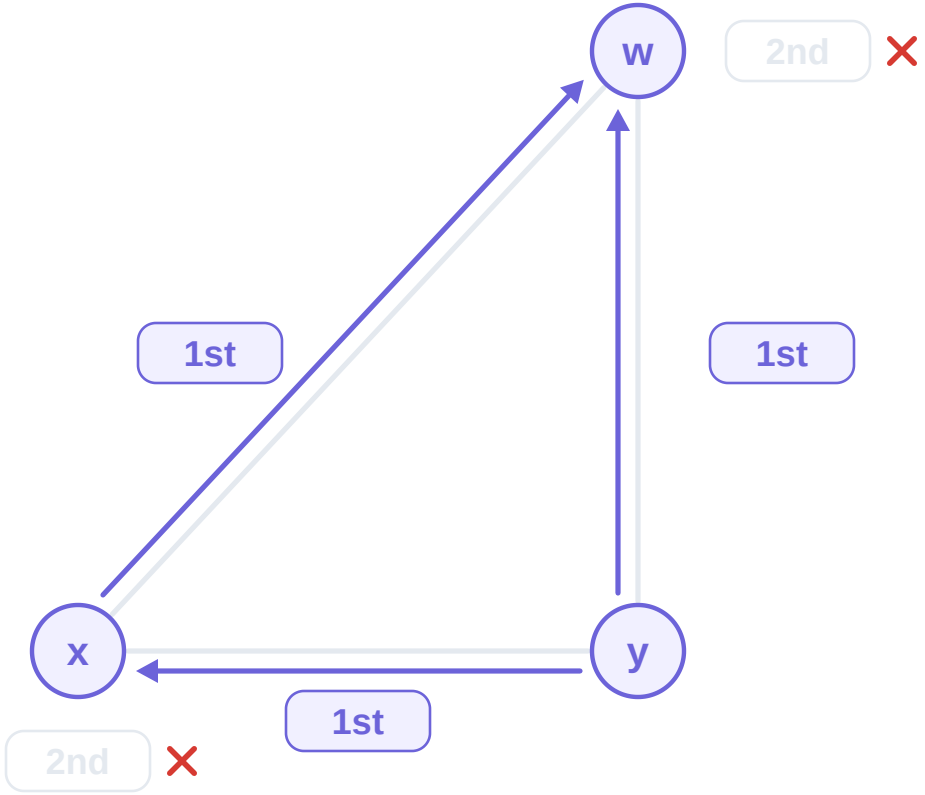
 # the rule, per origin

newer than the number stored:
store it, forward it, remember the number
otherwise:
drop it

y adds one to its number each time it sends a new one

The origin numbers them; a router keeps the newest per origin, drops the rest.

The same flood, with the rule



copies of y's advertisement

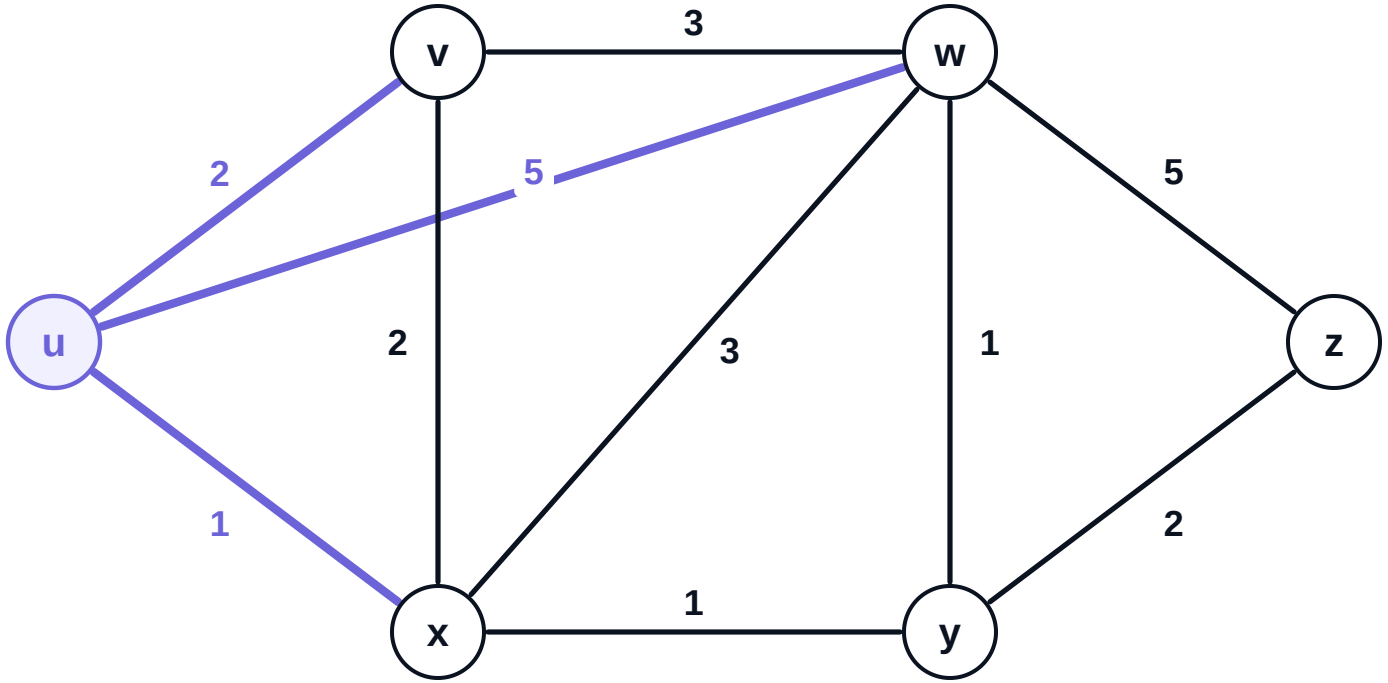
`1 → 2 → 0`

x and w each forward it once;
the copy that comes back
is not newer, so it is dropped

at most 2|E| transmissions

Each router forwards it once: at most 2|E| sends, then the flood is over.

If every router floods its own links, what does u hold?



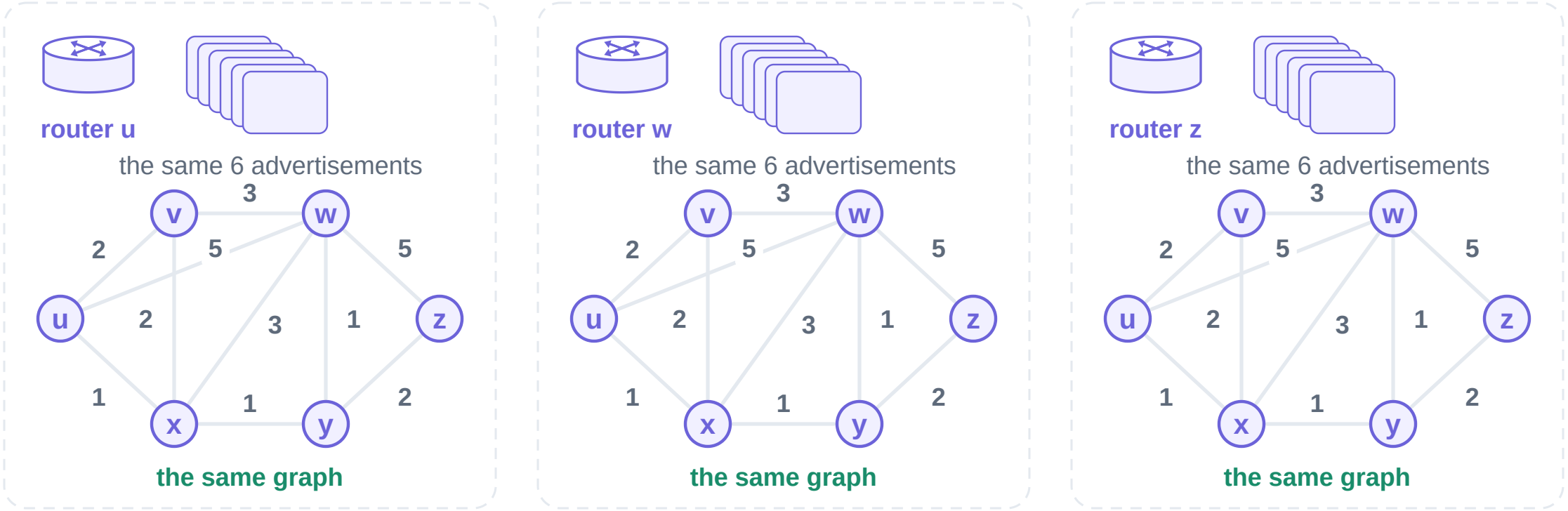
u's picture of the graph

- from u
- from v
- from x
- from w
- from y
- from z

One key press per origin, in the order the copies reach u. z's advertisement adds nothing new: its two links had arrived from their other ends.

Six advertisements, one per router: every link, from both ends. The whole graph.

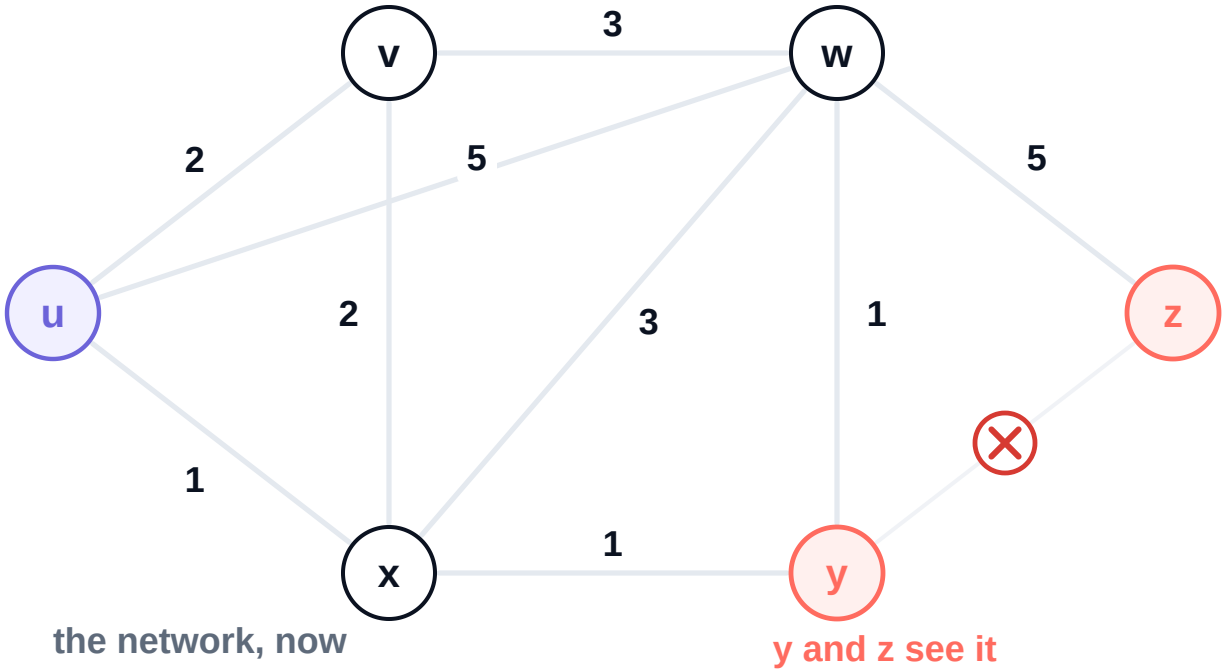
Does every other router hold the same graph?



Once the flood is over and nothing has changed since. This collection is the link-state database.

Yes. The flood carried each of the six to every router, so all hold the same.

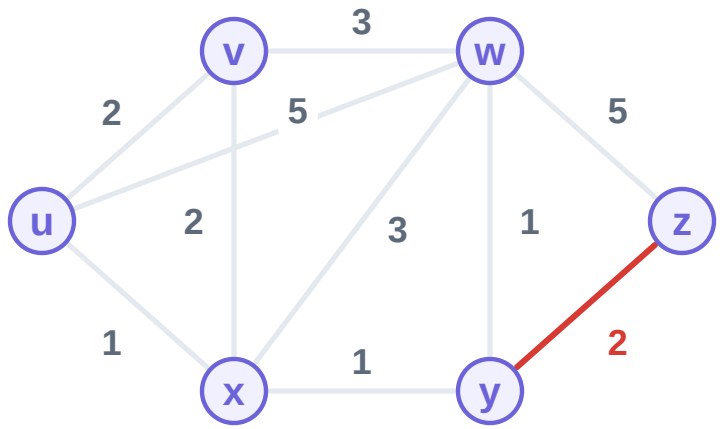
What if (y, z) fails after the flood?



the network, now

y and z see it

u's picture of the network



from y · seq 12

from z · seq 8

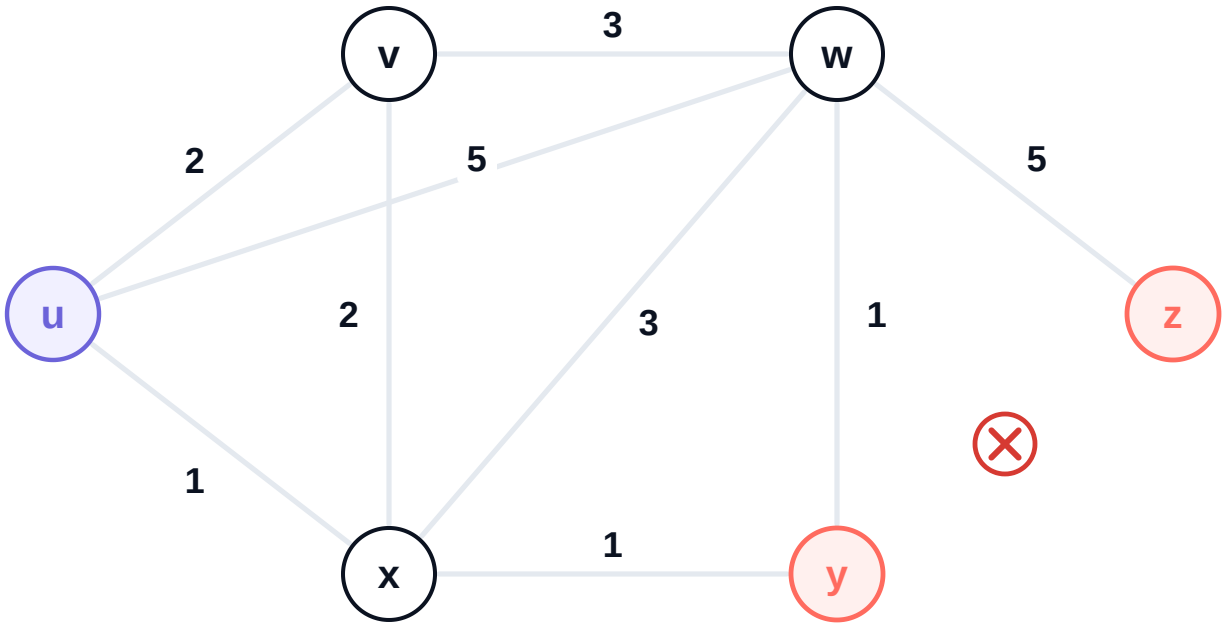
still has (y,z)

nothing told u

The opening's problem again (slide 5): only the two ends see it. Now we have the tool to fix it.

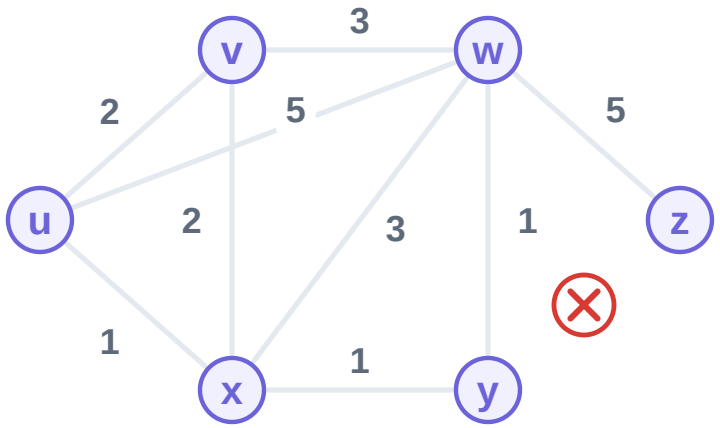
u's picture is only as good as the last advertisement it received.

How does the change reach u?



y and z each flood a new advertisement, without (y,z)

u's picture of the network



from y · seq 13

from z · seq 9

13 is newer than 12:
replace it, forward it

u recomputes its paths on the new picture

If a copy is lost: every router re-floods its advertisement every 30 minutes (OSPF), and acknowledges every copy it receives.

Its two ends flood a new advertisement with the next number; old copies are replaced.

What does the map cost?

storage, at every router

every router's advertisement: n of them
the whole graph, |E| links
plus a sequence number per origin

a million-prefix internet as one map:
no router could hold it

traffic, for every change

one change → up to 2|E| sends
every router advertising once
→ about n · 2|E| sends

a big network changes all the time:
the flood never stops

That is the price of a full map.

Every change reaches every router; every router stores everything.

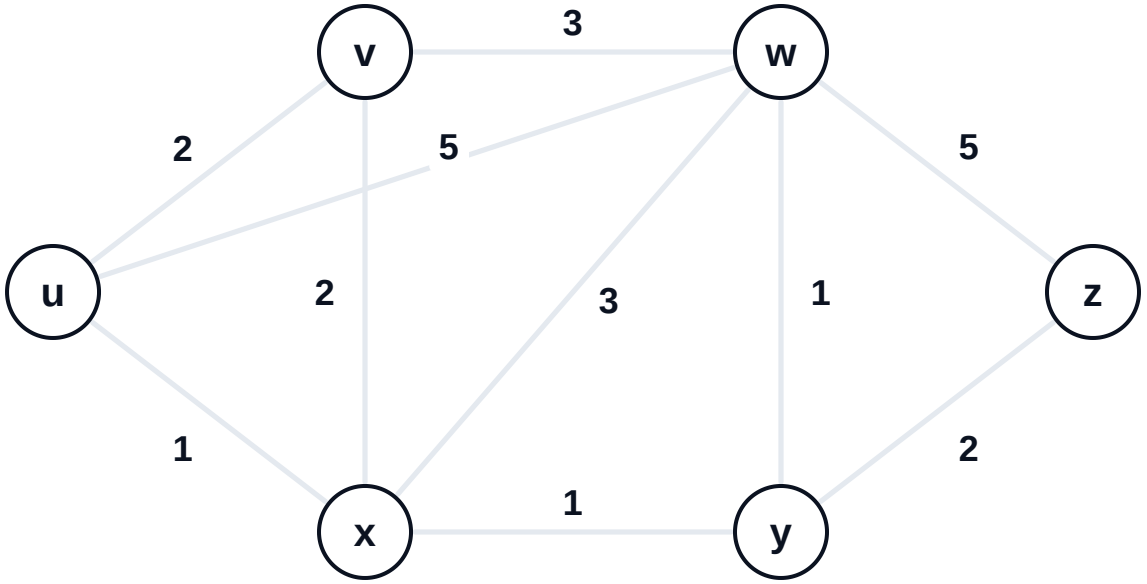
How does u turn the graph into least-cost paths?

03

Dijkstra

u now holds every link and its cost. It still has to find the cheapest path to every other router.

Why not list every path and compare?



paths from u to z

u–x–y–z	4
u–v–x–y–z	7
u–x–w–y–z	7
u–w–y–z	8
u–v–w–y–z	8
u–x–y–w–z	8

… 17 in all, in a graph of six

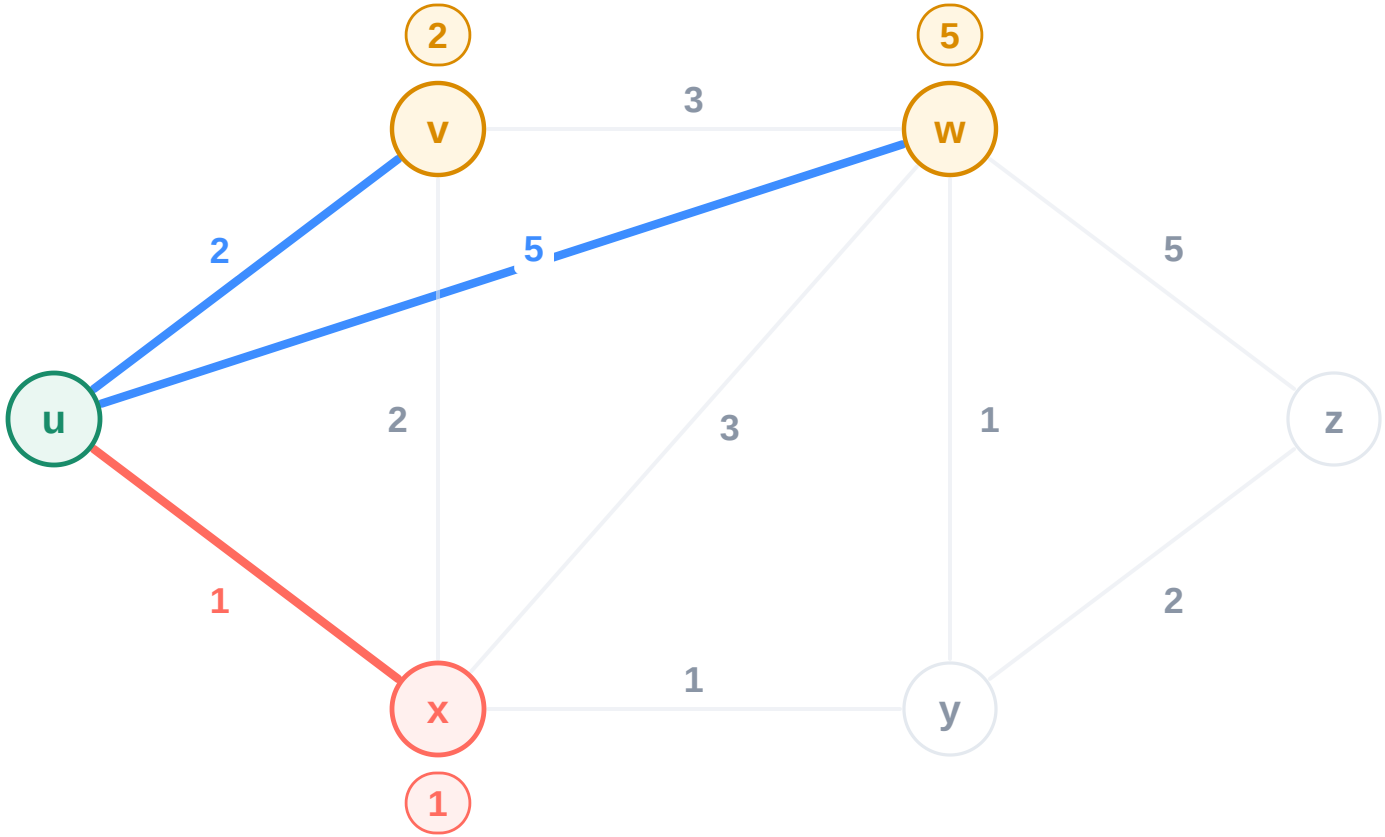
10 routers, each linked to each: 109,601 paths between two of them

20 routers: 1.7×10^{16}

Paths multiply far faster than routers. Listing them is hopeless.

1 The problem | 2 Flooding | 3 Dijkstra | 4 Tree to table | 5 Load costs

Whose least cost does u know for sure, right now?



to x, directly: 1

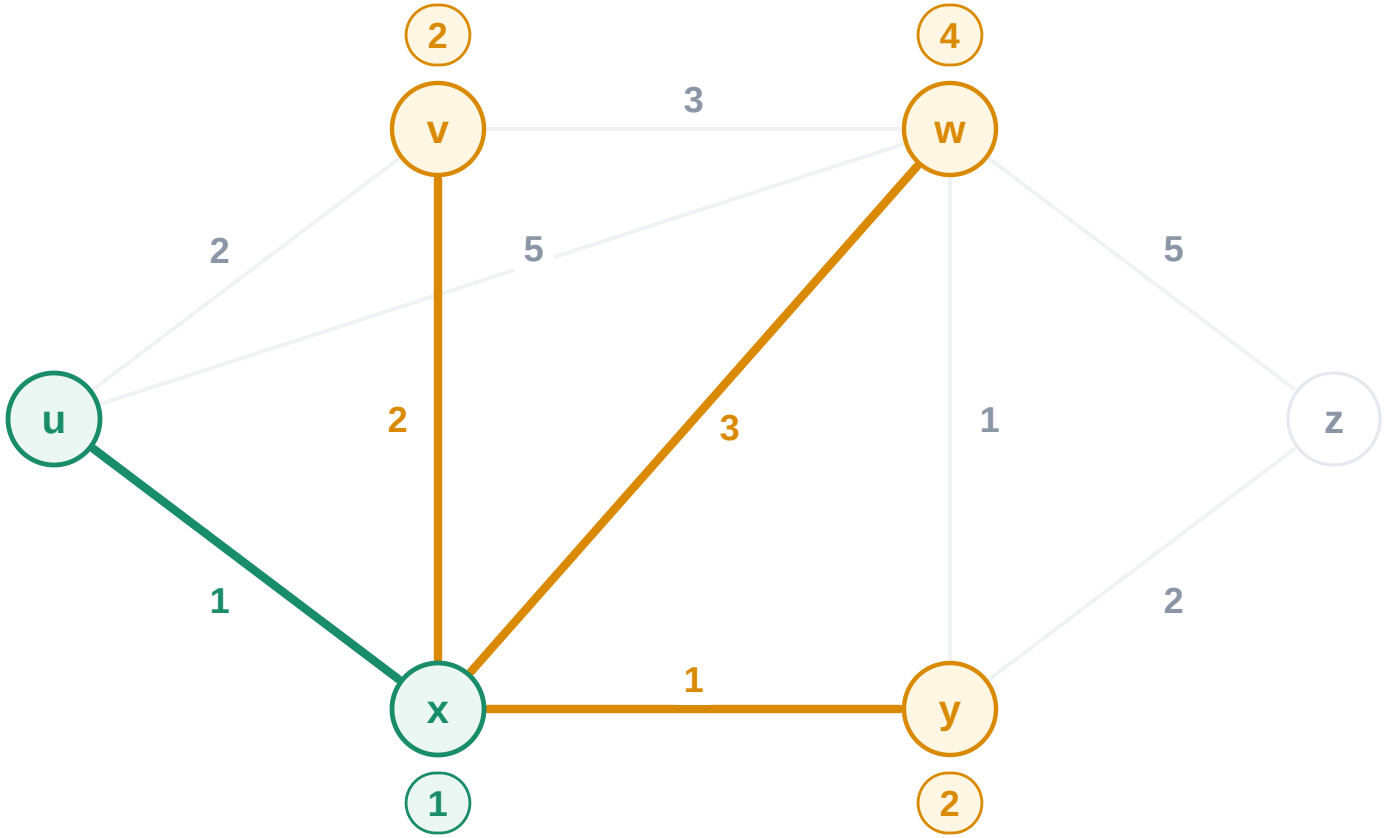
any other route to x starts
with u–v (2) or u–w (5)
and only gets more expensive

no link has a negative cost,
so nothing brings it back under 1

x is settled: 1

x, at 1. Any other route to x costs 2 or more at its first link already.

What does knowing x tell u about the others?



through x:

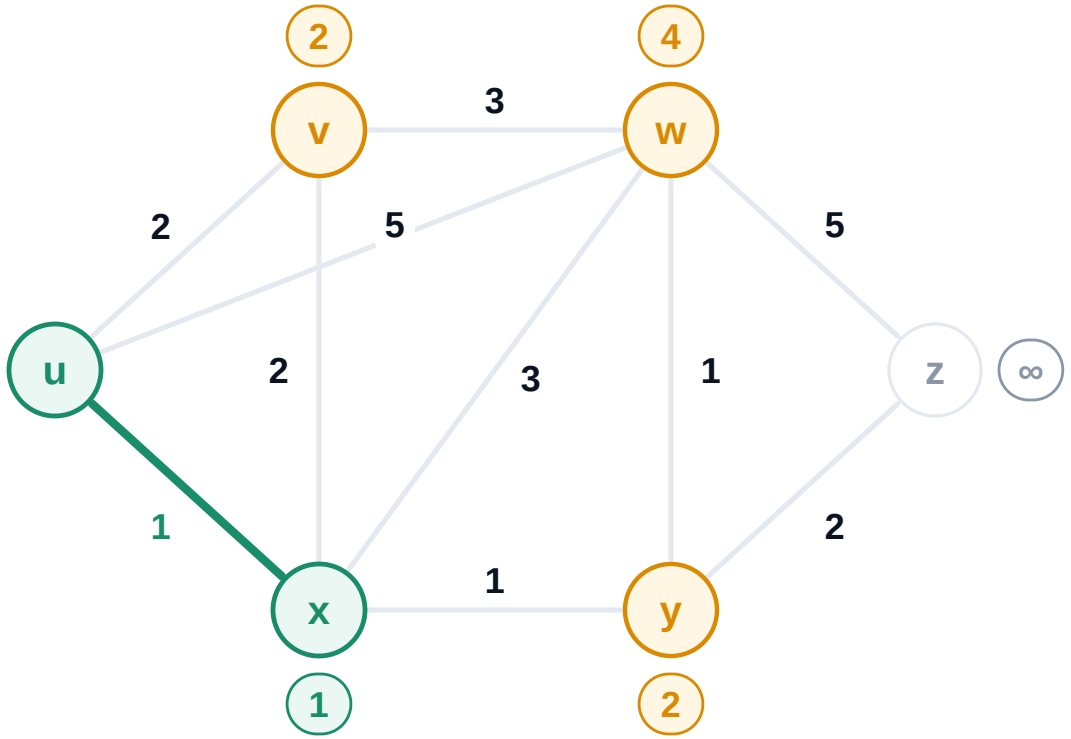
`v: 1 + 2 = 3`
not better than 2

`w: 1 + 3 = 4`
better than 5

`y: 1 + 1 = 2`
the first route to y

A settled node can only make its neighbours cheaper: w to 4, y to 2.

What does the algorithm keep track of?

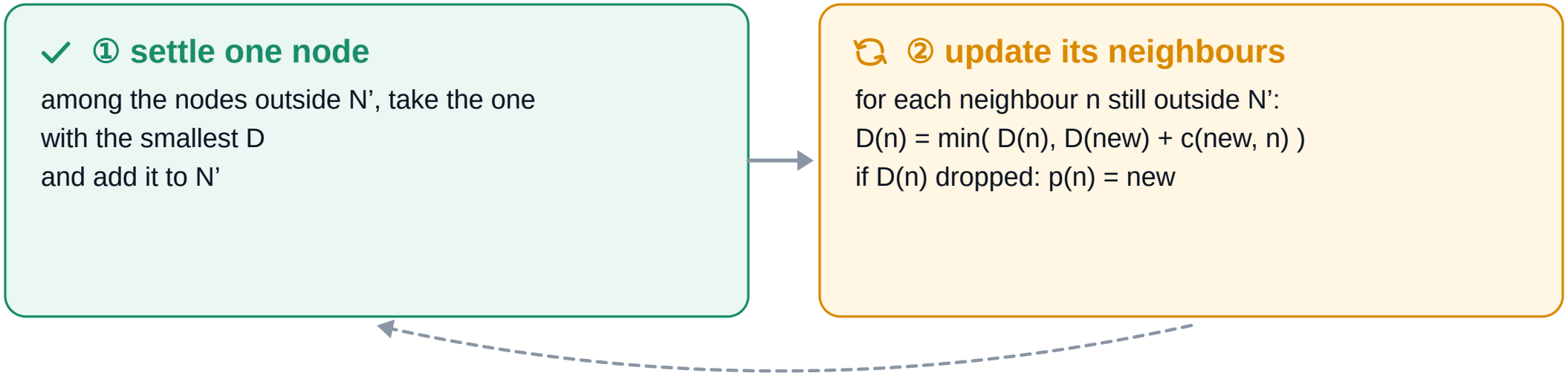


step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞

`N'`	the settled set: costs final	`{u, x}`
`D(n)`	the cheapest cost found so far	`v 2 · w 4 · y 2`
`p(n)`	the node before n on that path	`p(w) = x`

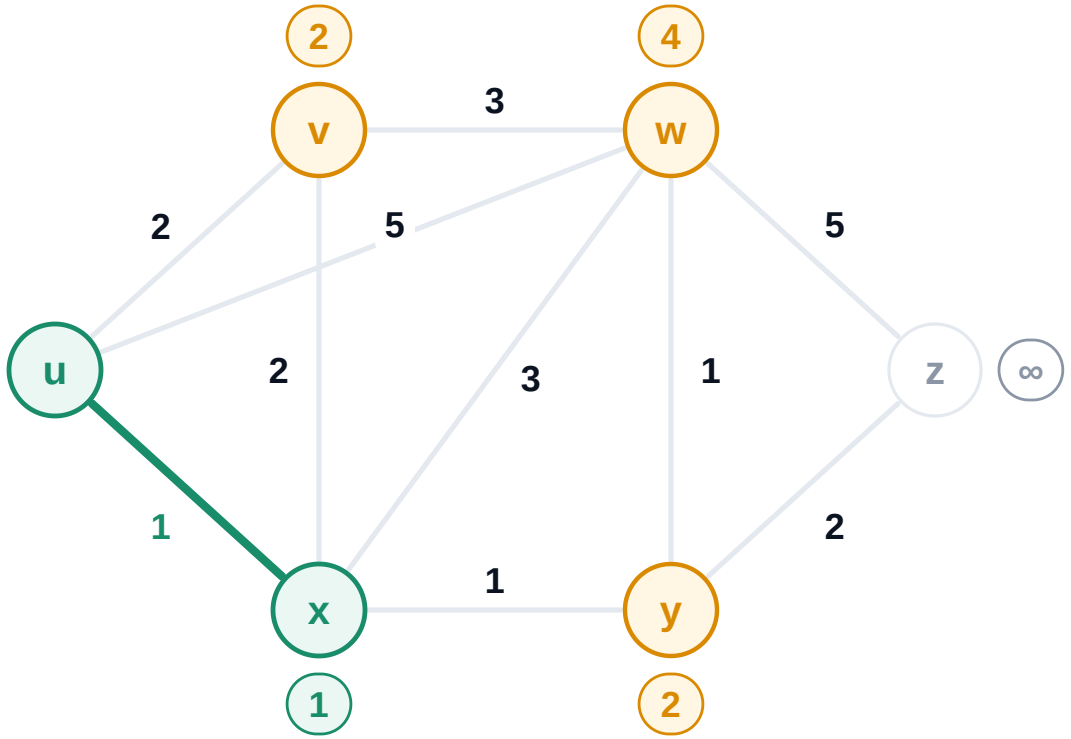
A settled set, a best-so-far cost and a predecessor for every node.

One round, two steps



Settle the cheapest unsettled node, then update its neighbours. Repeat.

Round 2: v and y both have D = 2. Which goes first?



step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞

take v first

order: v, y, w, z

v	w	x	y	z
2	3	1	2	4

take y first

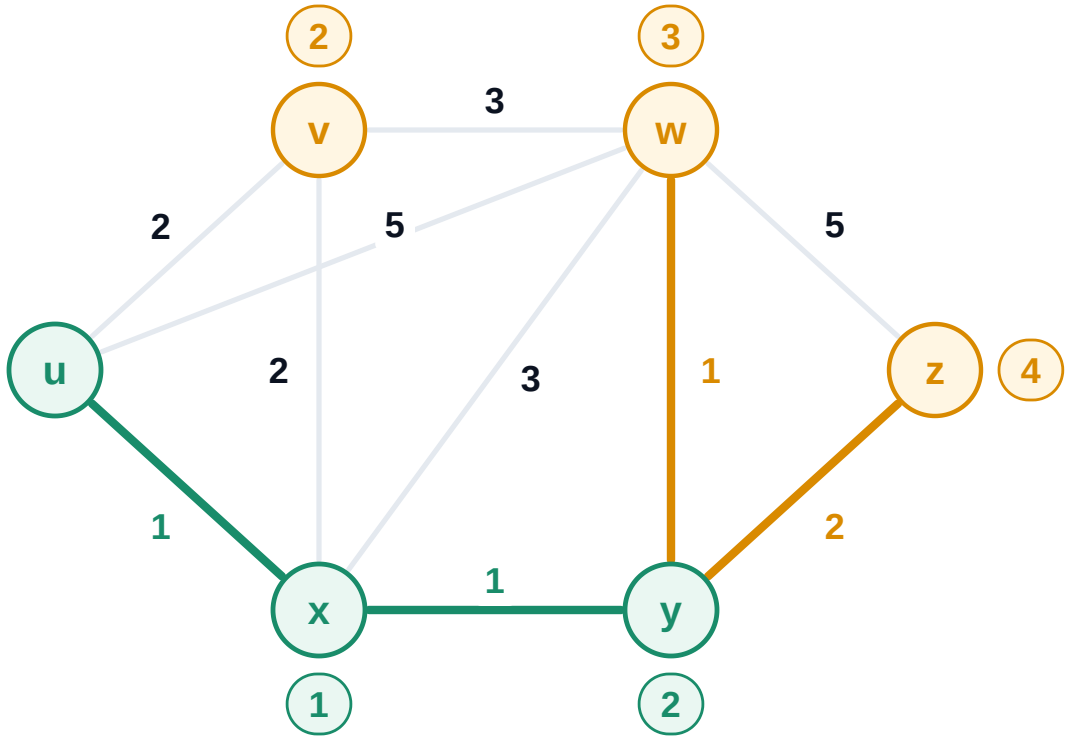
order: y, v, w, z

v	w	x	y	z
2	3	1	2	4

the same final costs, in the same rounds

Either. The final costs come out the same. We take y, as K&R's table does.

Round 2: what does y change?



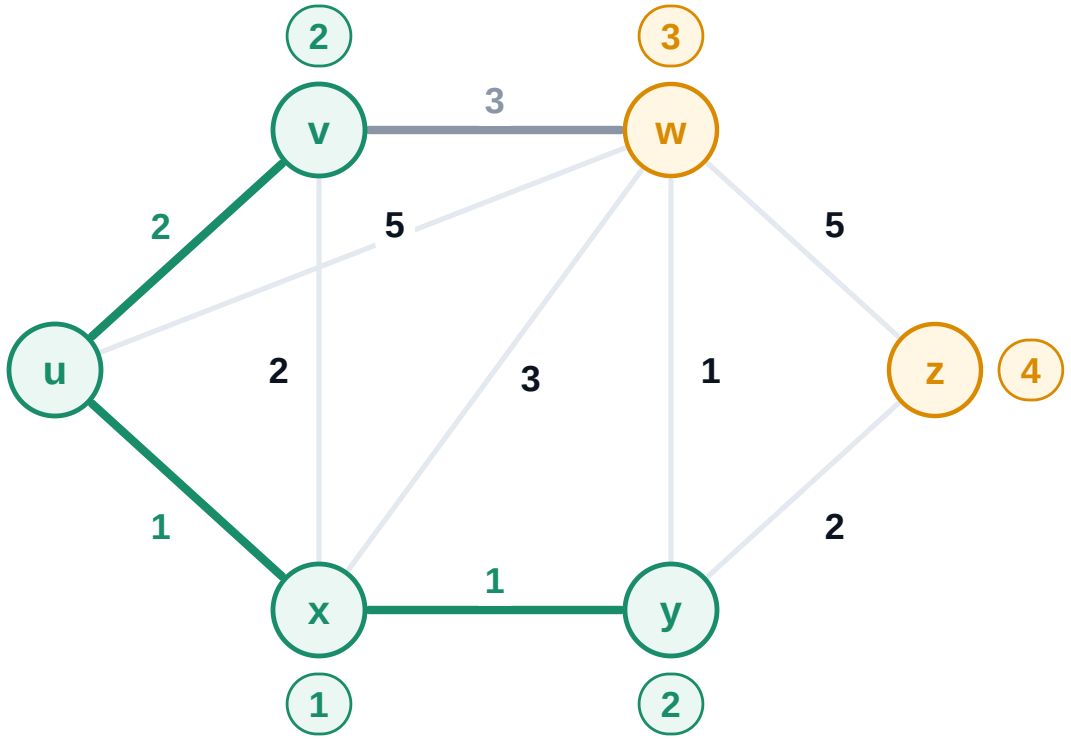
step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	**uxy**	2,u	3,y			4,y

`D(w) = min(4, 2 + 1) = 3` `p(w) = y`

`D(z) = min(∞, 2 + 2) = 4` `p(z) = y`

w drops from 4 to 3; z is reached, at 4. p(w) changes from x to y.

Round 3: v joins. What drops?



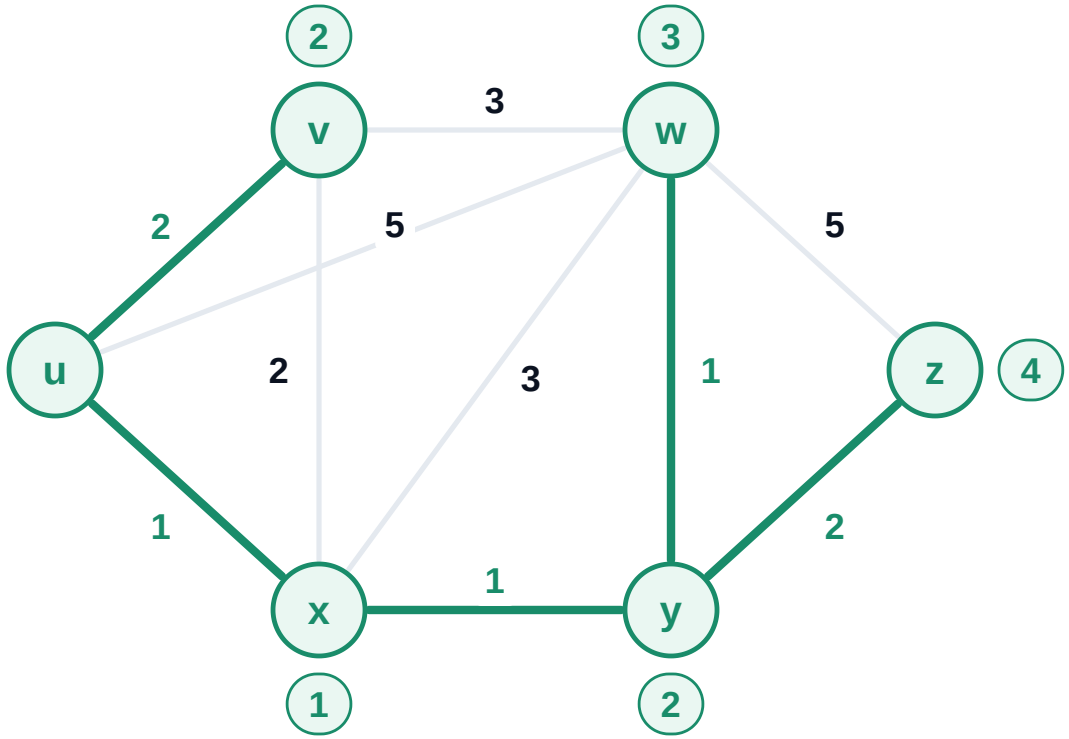
step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	**uxyv**		3,y			4,y

`D(w) = min(3, 2 + 3) = 3` no change

v's only unsettled neighbour is w, and going through v costs 5

Nothing. Every round settles one node; not every round lowers a cost.

Rounds 4 and 5: w, then z



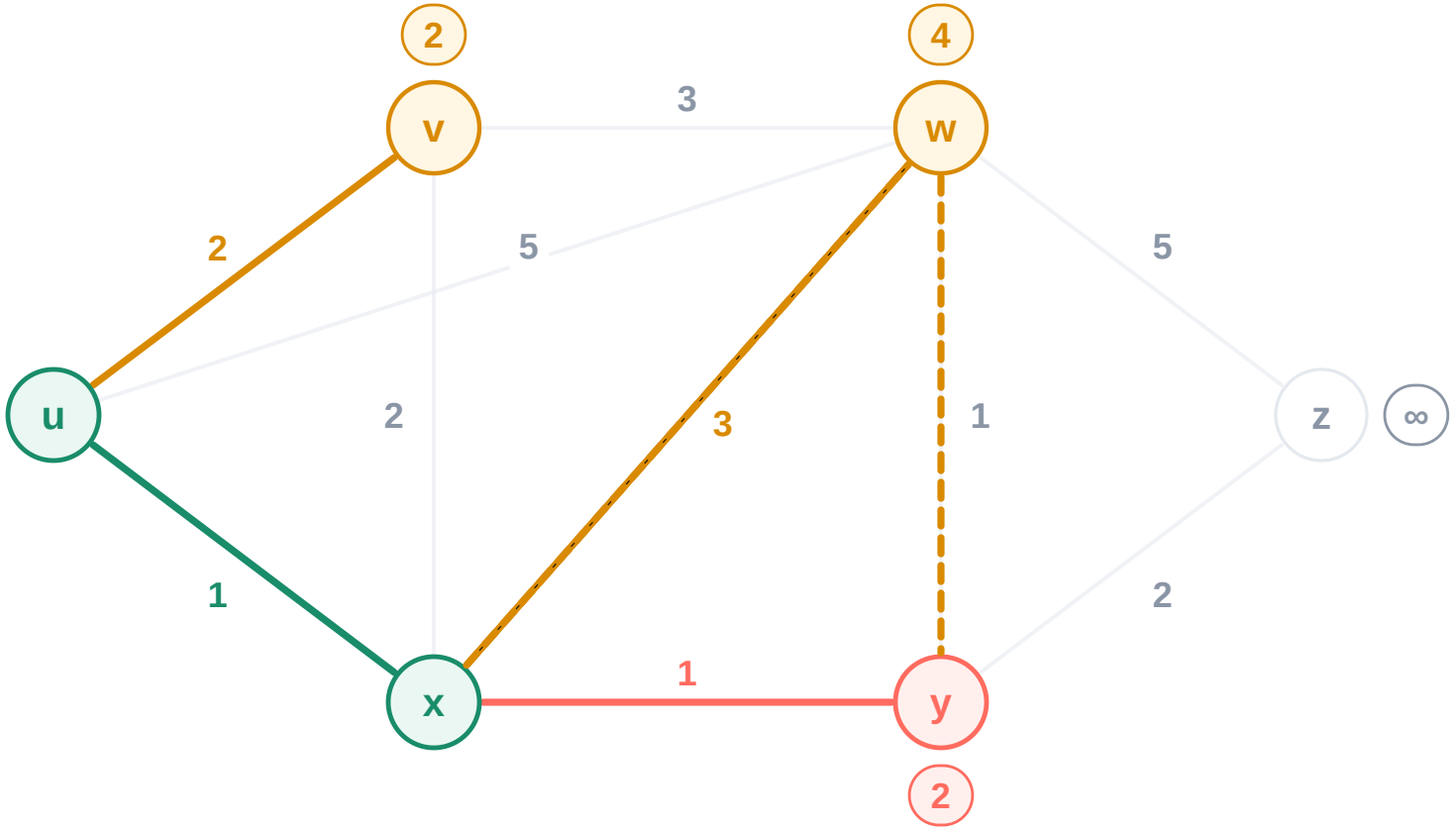
step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

round 4: w at 3. z via w: 3 + 5 = 8, not better than 4

round 5: z at 4. Nothing left outside N’

Five rounds: every node settled, with its least cost and predecessor.

Why is y's 2 already final, before the algorithm checks?



after round 1: N' = {u, x}

the cheapest way out of N' to **v is 2, w is 4, y is 2**

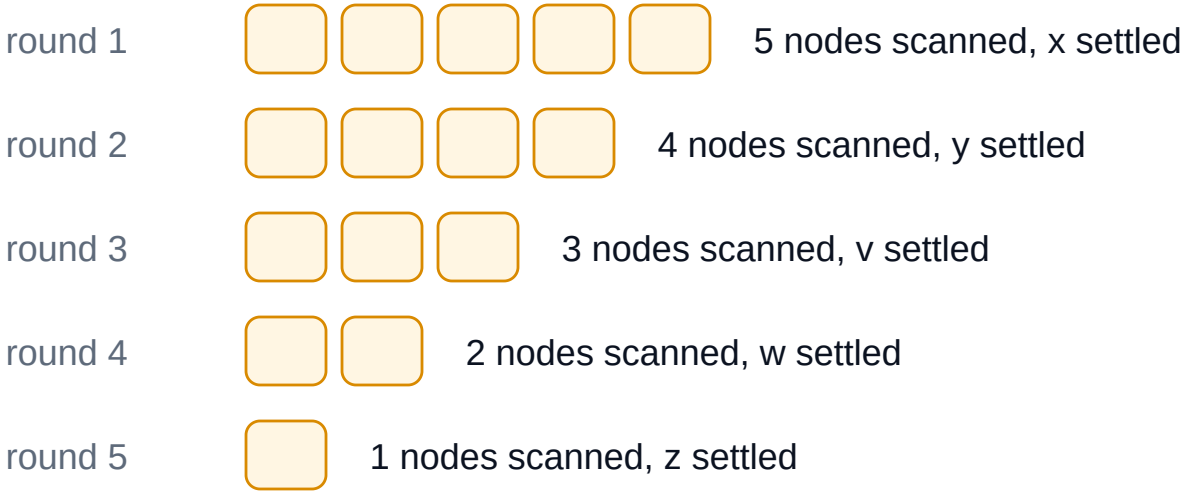
1. any route to y must step out of N' somewhere, and that step already costs ≥ 2

2. try u → x → w → y: 4 out, + 1 = 5, not better

3. after the step out, costs only add (none is negative): nothing beats 2

Every route out of N' already costs at least the smallest D, and costs only add.

How much work is it?



$$5 + 4 + 3 + 2 + 1 = 15$$

$$(n-1) + (n-2) + \ldots + 1 = n(n-1)/2$$

$O(n^2)$

n routers, scanning for the minimum

the updates touch each link twice at most, so they add less

with a priority queue:

$O((n + |E|) \log n)$

About $n^2/2$ comparisons for n routers: $O(n^2)$.

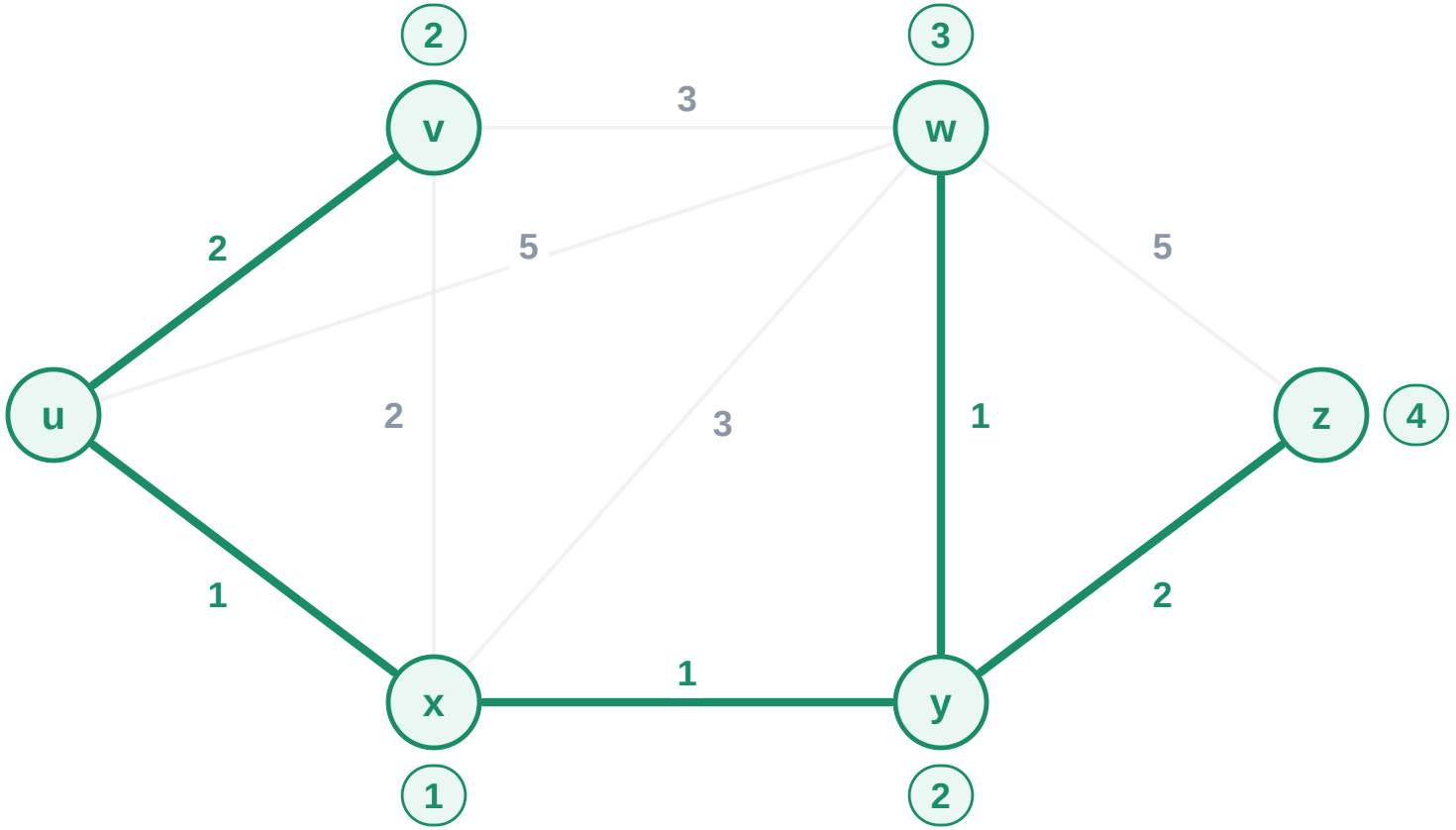
Is the first link enough?

04

Tree to table

u found a whole path to every router, but its table keeps one link per destination.

What did the five rounds produce?



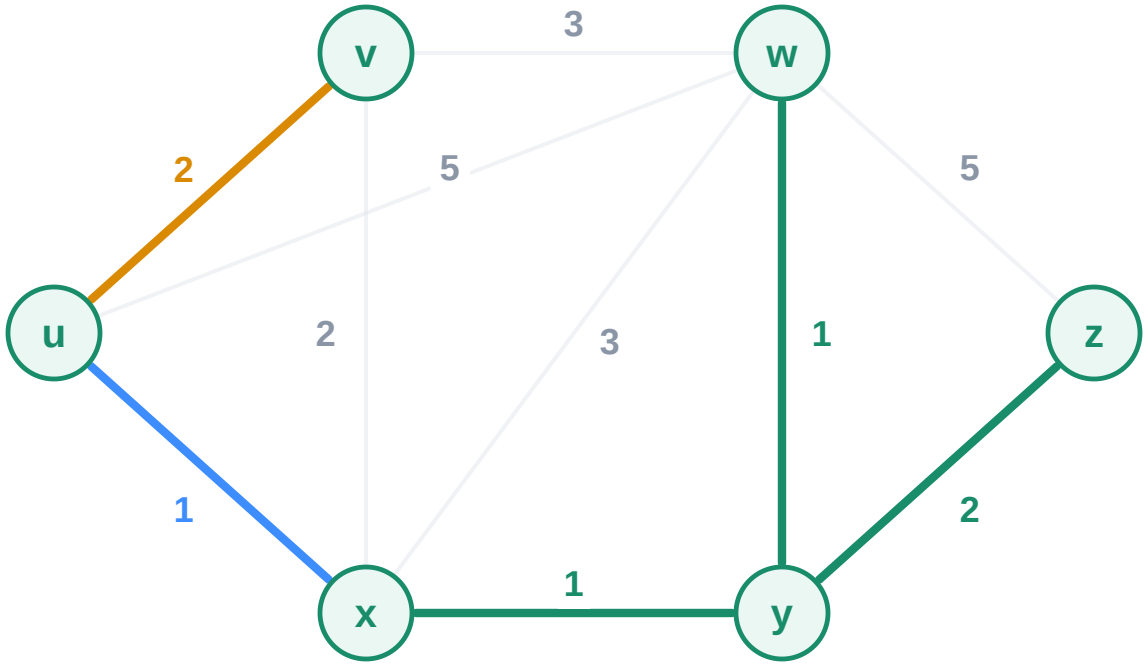

```
p(z) = y
p(y) = x
p(x) = u
```

read backwards: u → x → y → z

five predecessors,
five edges, no cycle: a tree

One predecessor per node; together, a tree of least-cost paths from u.

Which part goes into u's forwarding table?



destination	first link
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

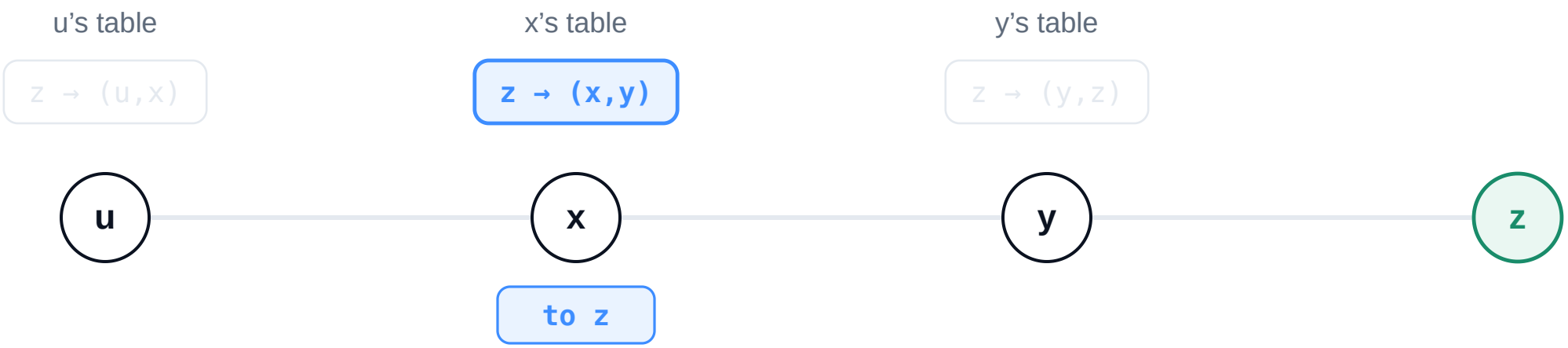
four rows, one link

the rest of each path is not stored

Every prefix attached to z gets z's row: prefix → (u, x).

Each row keeps only the first link. Four of five destinations leave by (u, x).

Why is the first link enough?

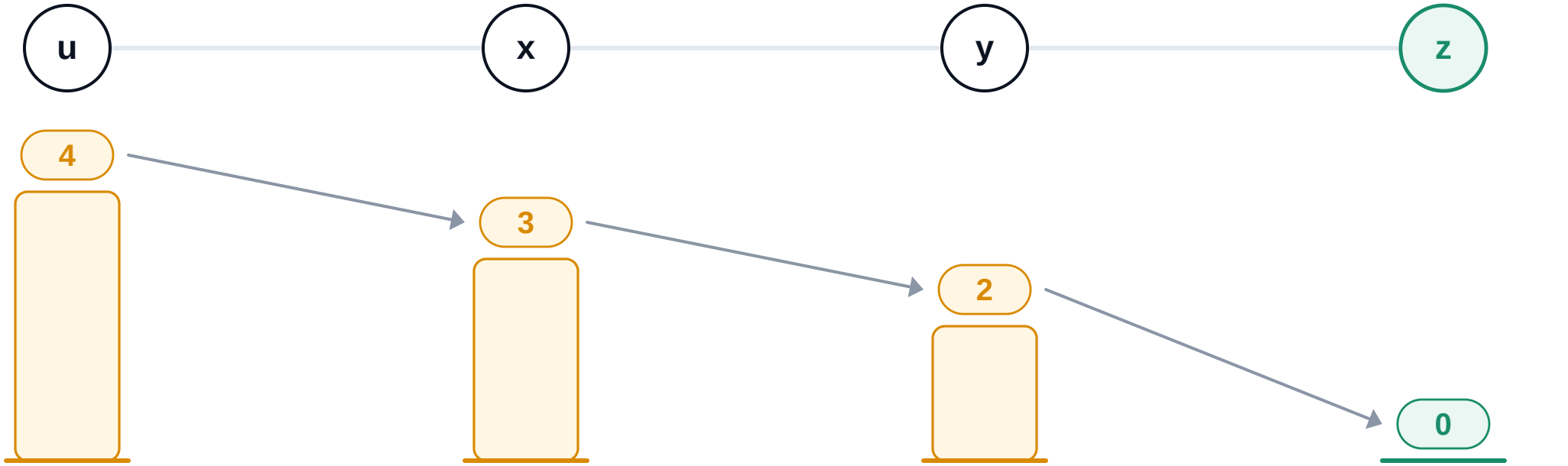


each router computed its own tree, on the same graph

Any piece of a least-cost path is itself a least-cost path. One key press per hop.

Every router computed on the same graph, so each one continues the path.

Could the packet ever come back to a router?



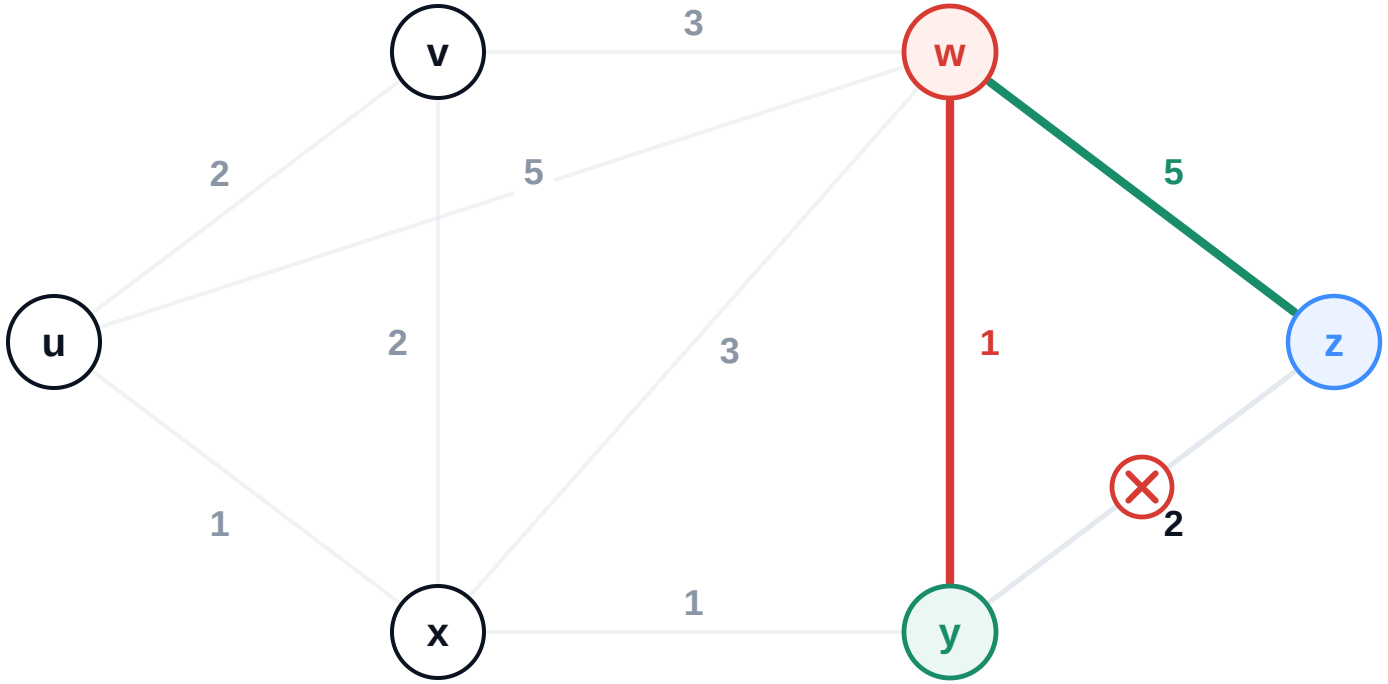
each router's own cost to z, as computed on the same graph

4 > 3 > 2 > 0

A number that keeps falling cannot repeat, so no router is visited twice. This holds even if two routers break a tie differently.

No. Same graph, positive costs: every hop lands strictly closer to z.

What if two routers hold different graphs?



y, recomputed

`z → (y,w): 1 + 5 = 6`

w, not yet told

`z → (w,y): 1 + 2 = 3`

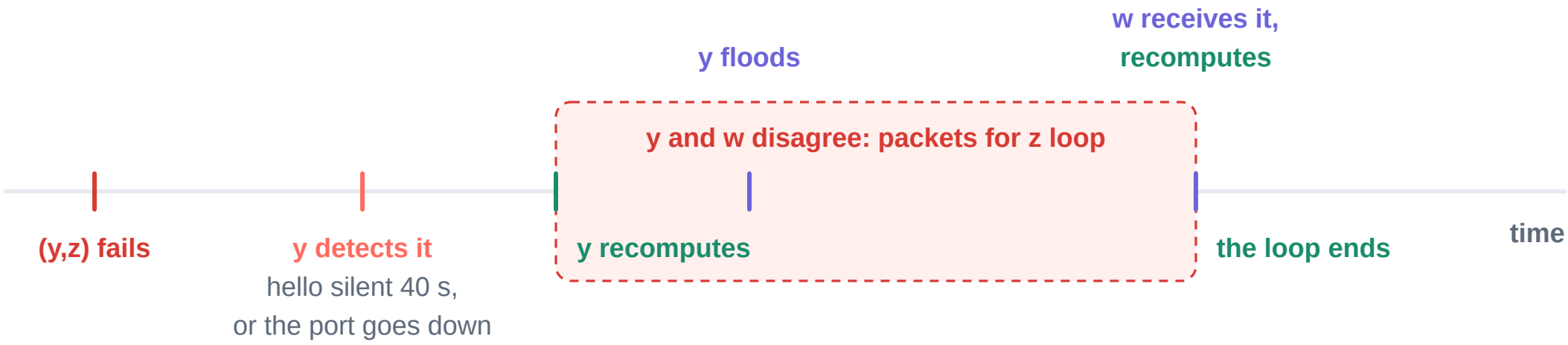
on a graph that still has (y,z)

`y → w → y → w → …`

Slide 49 needed the same graph at every router. Here y and w hold different graphs.

Each router follows its own graph, and the packet goes back and forth.

How long can the graphs differ?



In that window packets can loop; TTL only limits the damage. The length of the window is the convergence time.

Until every router has the new advertisement and has recomputed.

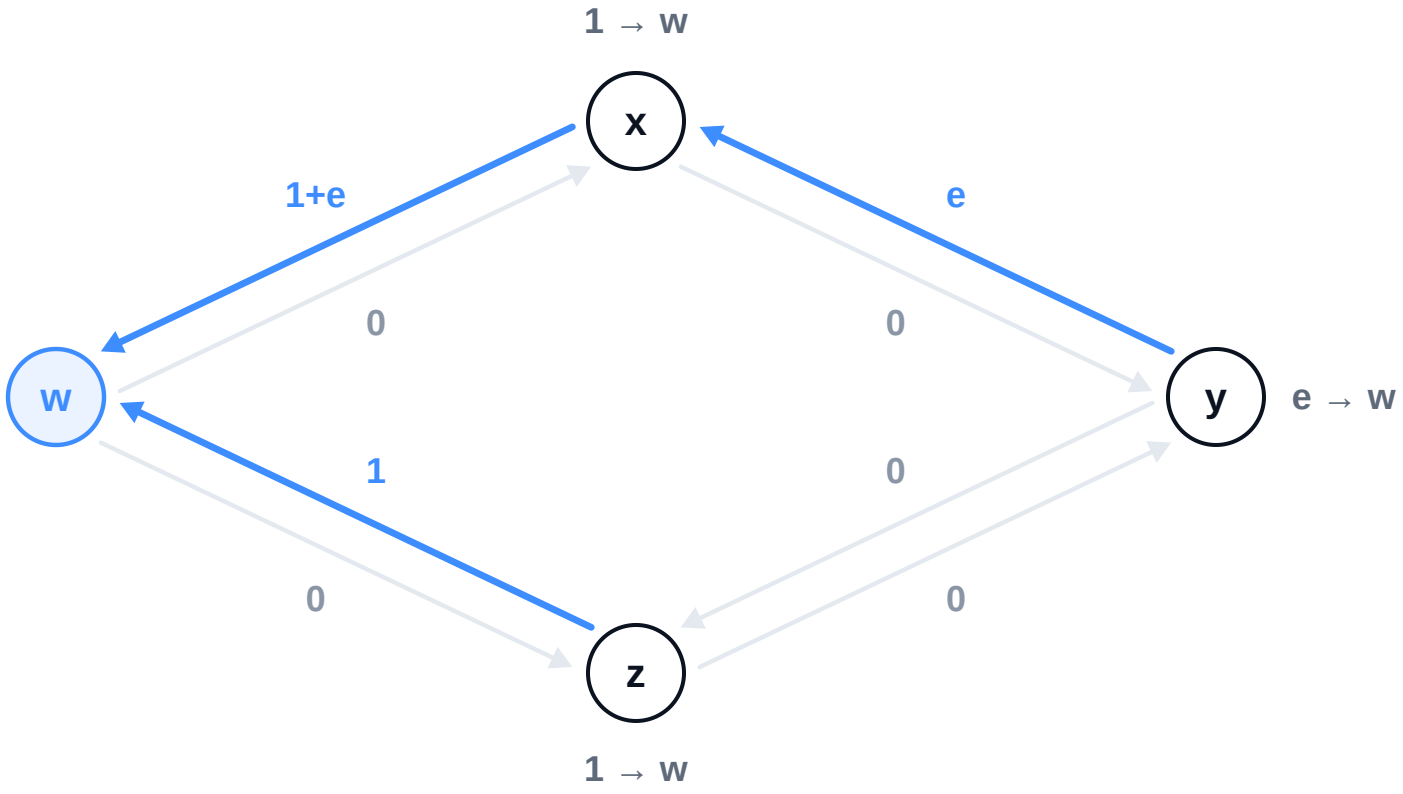
Why not make a link’s cost its current traffic?

05

Load costs

Then paths would steer around busy links by themselves. Would they settle down?

A ring where each direction costs its own traffic



cost of a direction
= the traffic it carries

x sends 1 to w
y sends e to w (e < 1)
z sends 1 to w

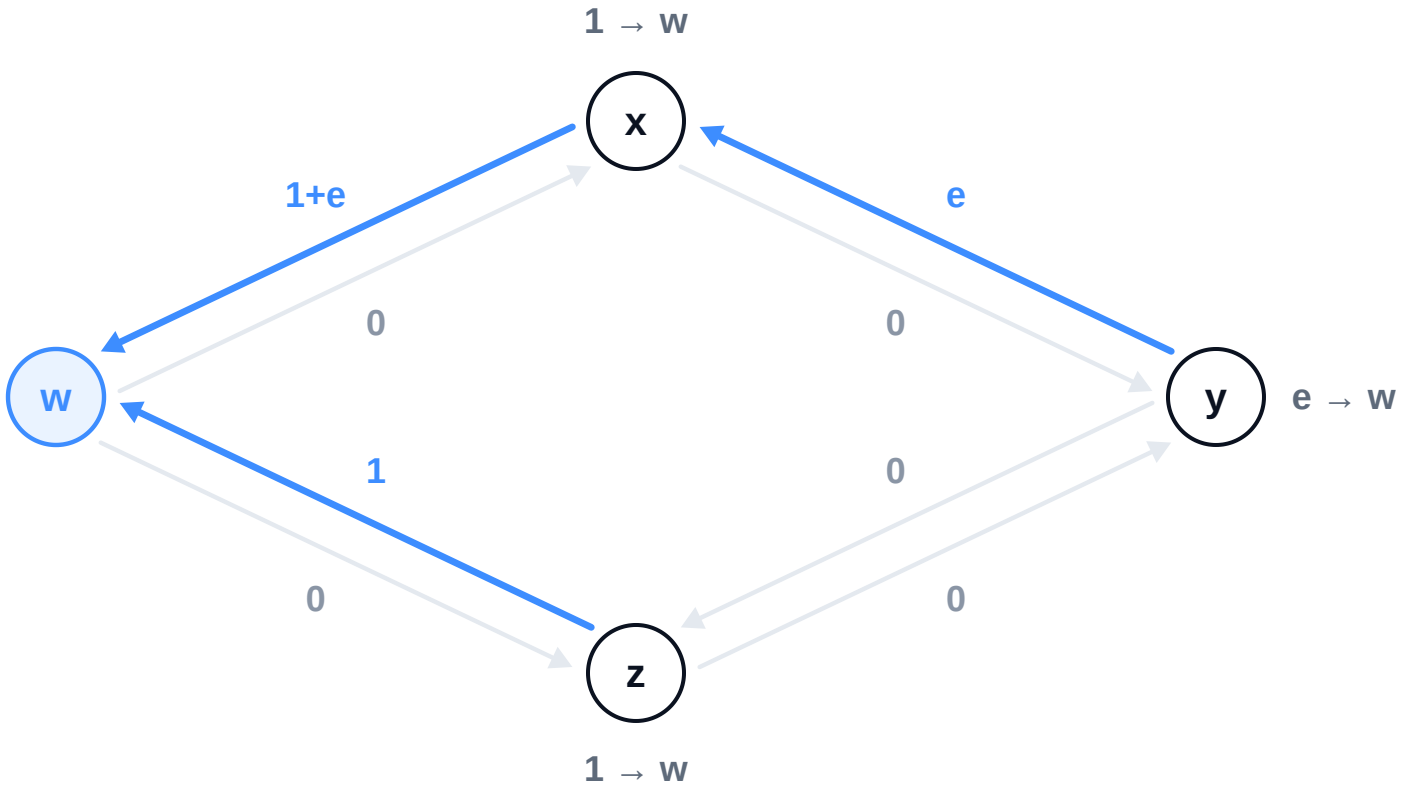
routes right now:

`x → w`
`y → x → w`
`z → w`

K&R, Figure 5.5

Each direction carries its own traffic, so each direction has its own cost.

What does each router compute from these costs?



x to w

direct: 1+e

the other way: 1

moves

y to w

via x: 1+2e

via z: 1

moves

z to w

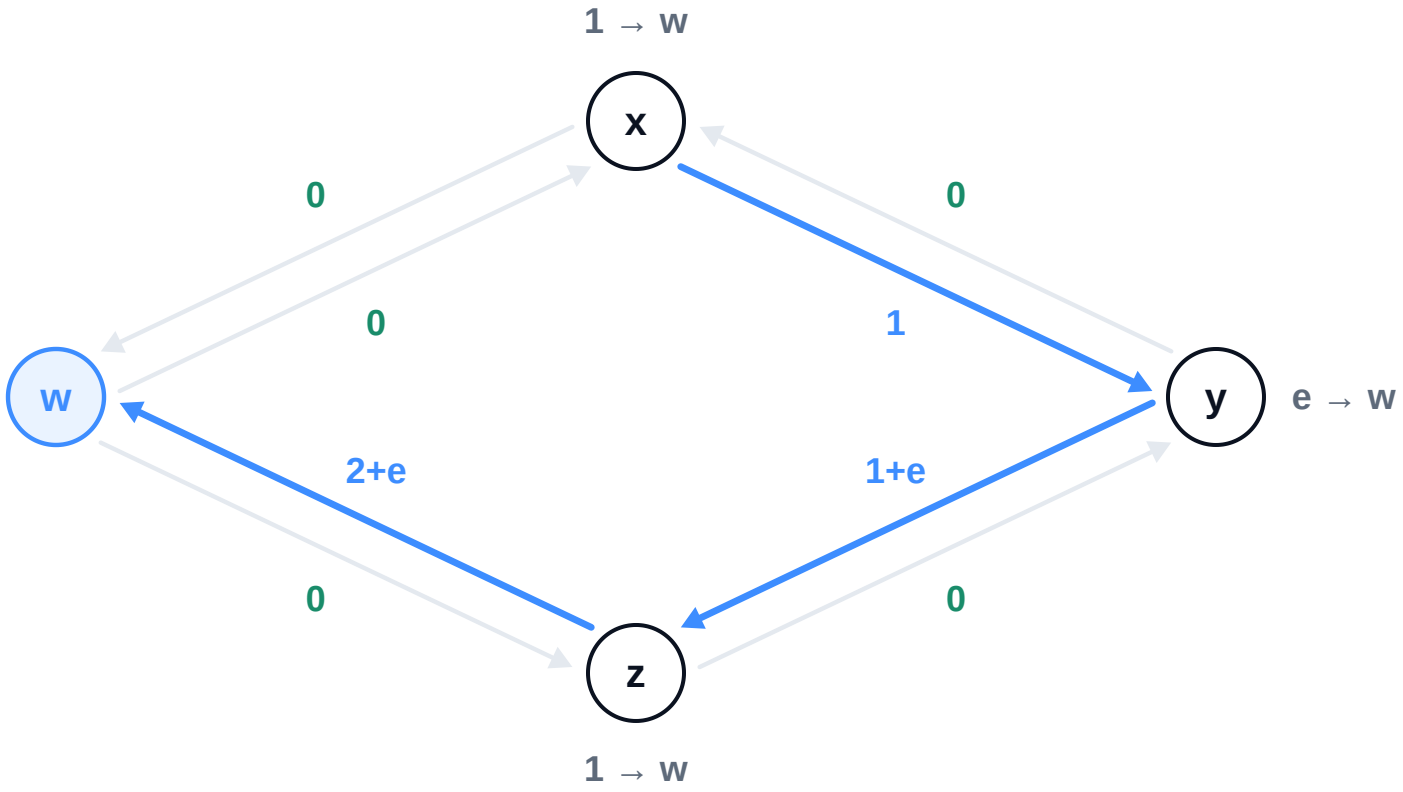
direct: 1

the other way: 1+2e

stays

x and y both move to the direction that looks empty.

After they move, what do the costs look like?



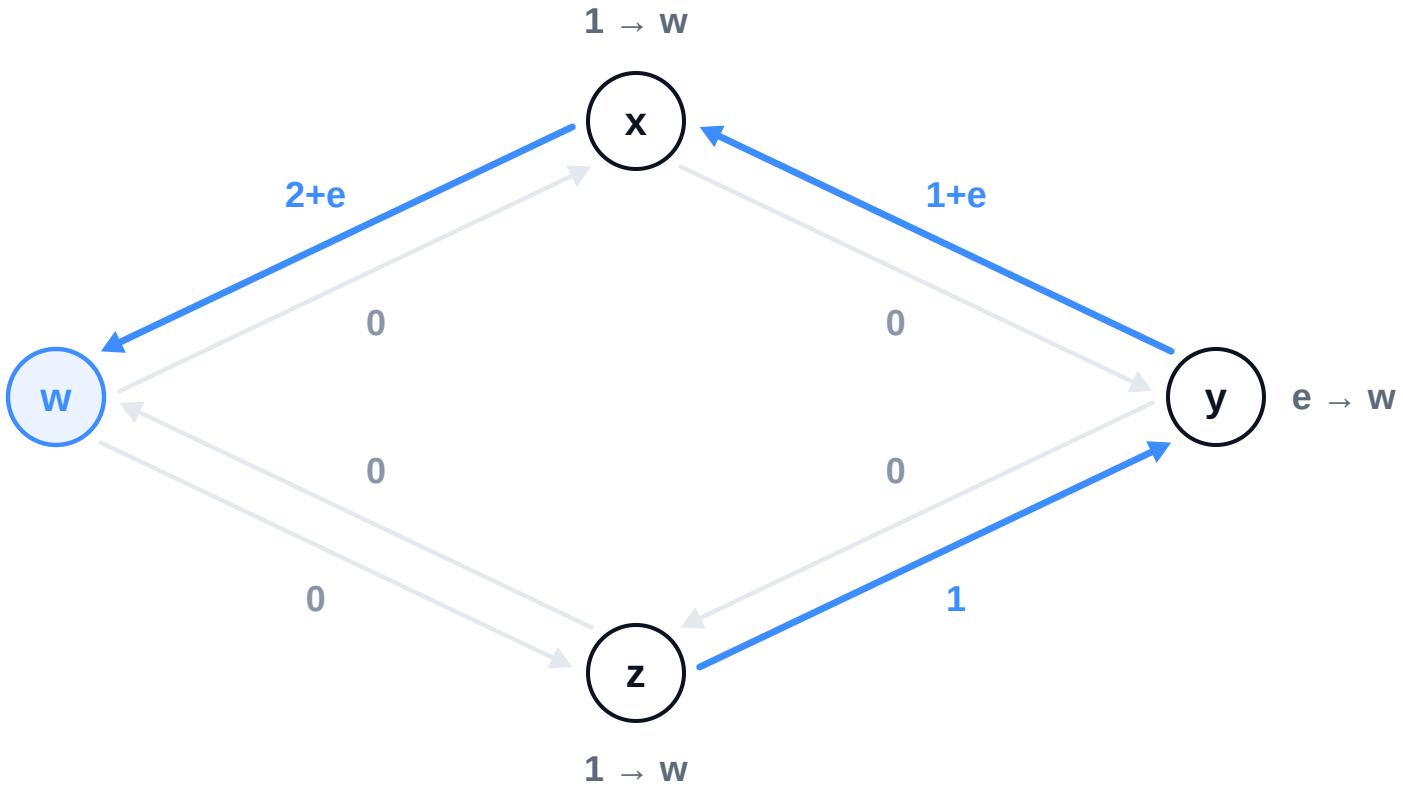
after the move

x → y → z → w
y → z → w
z → w

the other direction: 0, all the way round

Now the direction they left is empty, and costs 0 all the way round.

And the next computation?



the next round

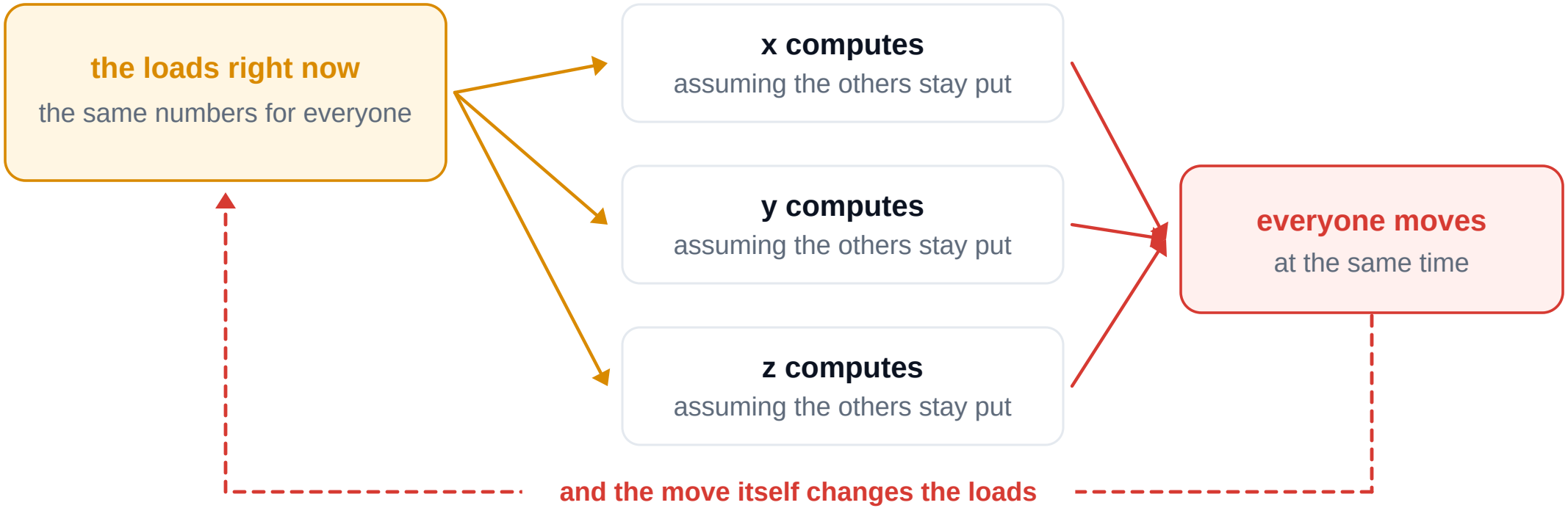
```
x → w
y → x → w
z → y → x → w
```

the direction they left is empty again

press again: back

All three move back, then forward again. The routes never settle.

Why do they all move together?



All react to the same numbers at once, and the move changes the numbers.

How do real networks avoid this?

costs that do not depend on load

set by the operator, per link
traffic moves, the numbers do not
the usual practice with OSPF

costs can still encode capacity:
a 10 Gbps link cheaper than 1 Gbps

if the cost must follow the load

keep the routers out of step:
randomize when each one advertises
so they do not all react at once

routers left to themselves
drift into step

Routers that start out of step drift into step unless the timing is randomized (Floyd and Jacobson, 1994).

Fix the costs, or stop every router from reacting at the same moment.

Five points, one line each

01



The problem

nodes, edges, costs; the least-cost path

02



Flooding

own links only; seq numbers end the flood

03



Dijkstra

settle the cheapest, update neighbours; $O(n^2)$

04



Tree to table

keep the first link; works while graphs agree

05

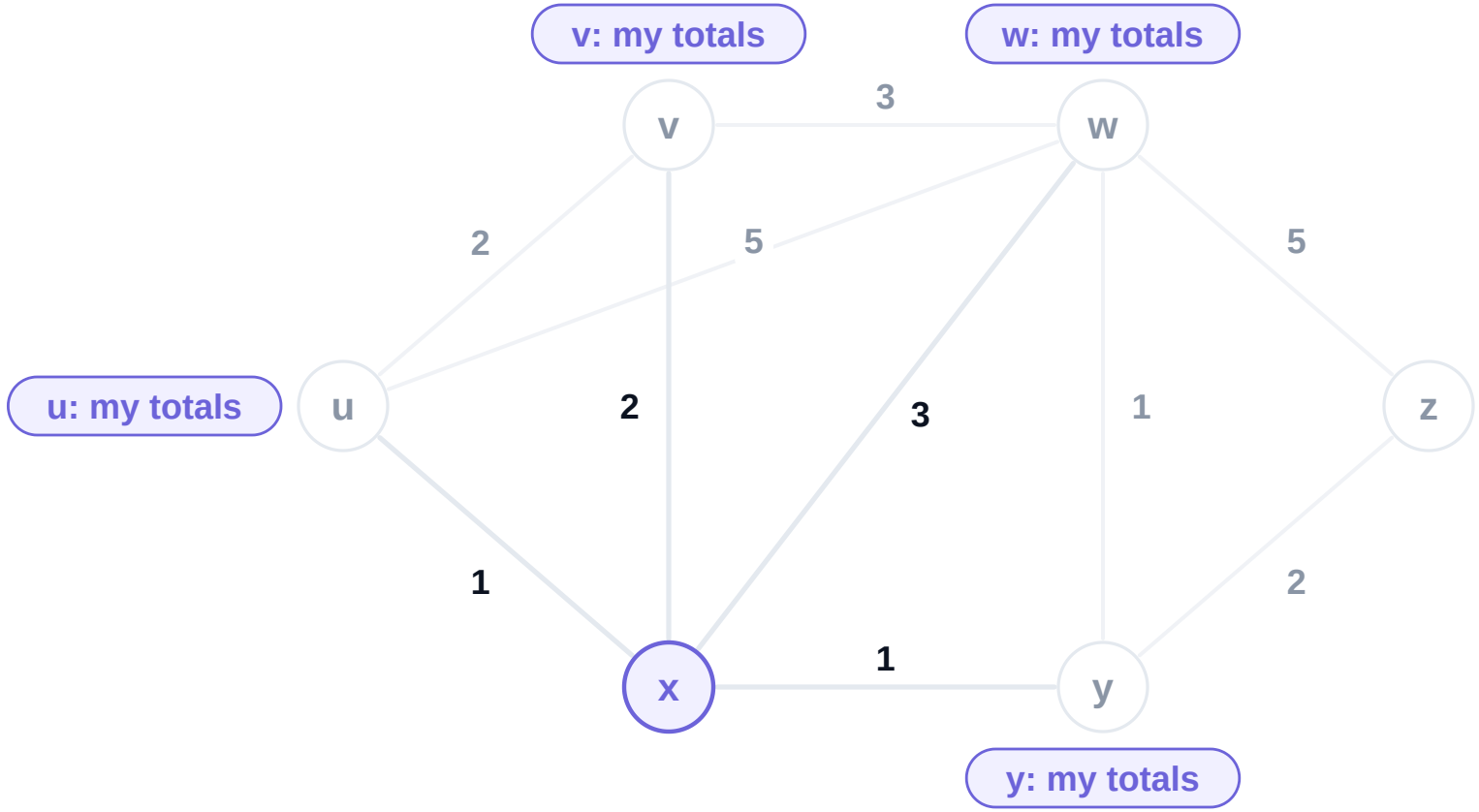


Load costs

react together, oscillate; fix costs or timing

Link state: every router gets the whole map and computes its own table.

What if a router only talks to its neighbours?



link state

every change, to every router

next lecture

x hears only from its neighbours,
and only their totals

far less traffic. What does it give up?

Next: routers that share only their totals, and only with neighbours.

Reading

Kurose & Ross, 8th edition

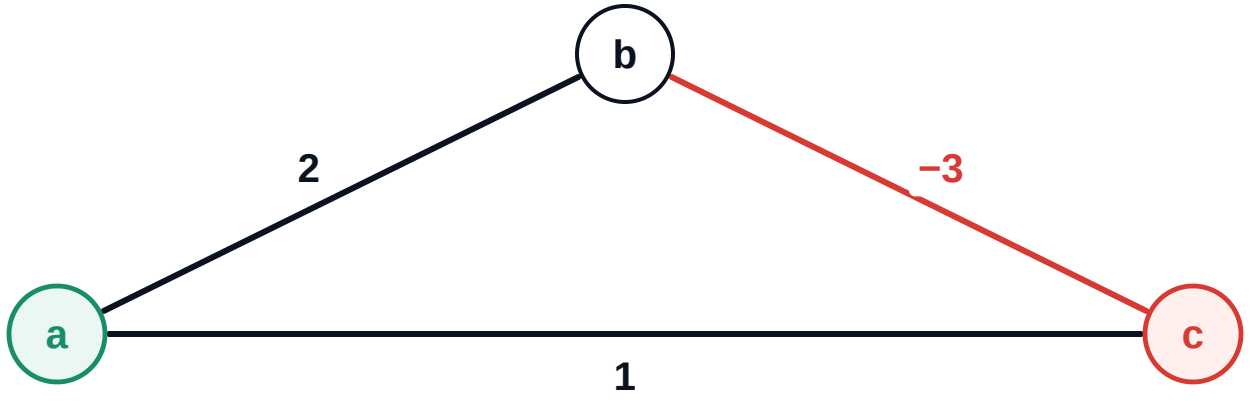
5.1 Introduction

5.2.1 The link-state routing algorithm

Problems P3 and P4 at the end of Chapter 5
(check the numbering against the edition you use)

Run Dijkstra by hand once, on a graph you draw yourself.

Backup: what if a link cost were negative?



Dijkstra settles c at 1 in round 1

but a → b → c costs 2 + (−3) = −1

a node was settled before its cheapest route was found

Link costs are never negative in practice; the algorithm assumes it.

With a negative cost, the smallest D can be settled too early.