

Routing #2

Today in Computing

- First patent for integrated circuits was filed 61 years ago today by *Jack Kilby* of Texas Instruments
- Sort of the beginning of the computer chip!
- Bob Noyce invented the second IC shortly after...
 - His was made of silicon instead of germanium...
 - He later went on to co-found Intel...
- But still, it all goes back to Jack Kilby on February 6th of 1959
- And in today in Internet history...

Never gonna give you up

- Never gonna give you up
- Never gonna let you down
- Never gonna run around and desert you



Today

- Kinds of routing, kinds of routers
 - Intradomain and Interdomain routing (IGPs and EGPs)
 - Internal and External routers
- “Least-cost” routing
- Trivial and static routes
- Distance-Vector in depth

Kinds of routing, kinds of routers

Interdomain and Intradomain Routing

- The Internet does not work by having a single giant routing protocol
- The Internet is a network of networks
 - The best way to route on one may not be the best way on another (why not?)
 - Networks differ!
 - Physical size, number of hosts, number of routers, bandwidth, latency, failure rate, topology (densely vs. sparsely connected, etc.), support staff size, when built, ...
- So...
 - Let individual networks choose how to route *inside* their network (intradomain)
 - .. all networks agree on how to route *between* themselves and other networks (interdomain)

Interdomain and Intradomain Routing

- Intradomain routing
 - More or less means *routing within a single network* (technically an “Autonomous System”)
 - Protocols used are often called IGPs or *Interior Gateway Protocols*
 - A number are actively used today

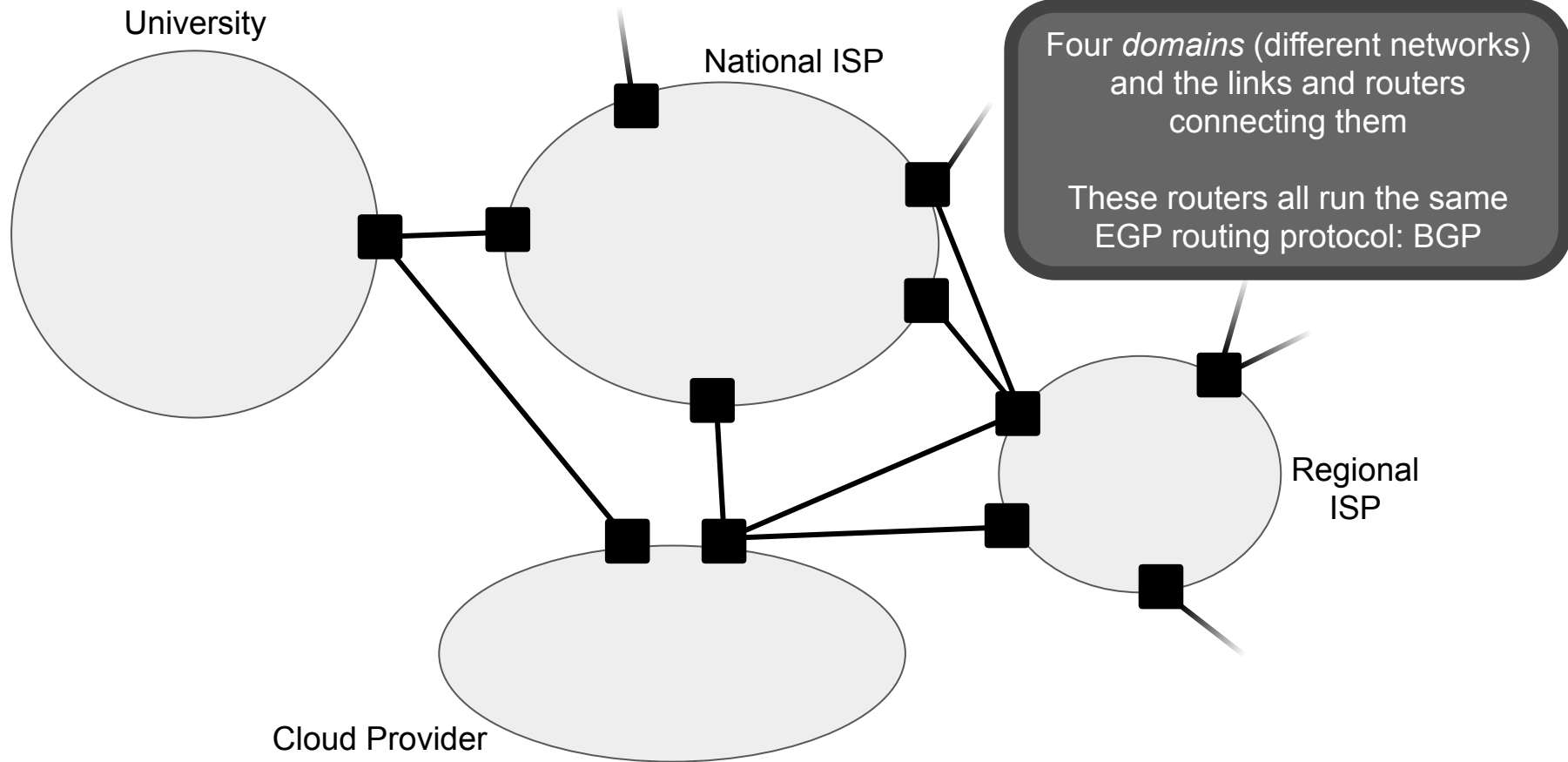
Interdomain and Intradomain Routing

- Intradomain routing
 - More or less means *routing within a single network* (technically an “Autonomous System”)
 - Protocols used are often called IGPs or *Interior Gateway Protocols*
 - A number are actively used today
- Interdomain routing
 - Routing *between* networks (ASes)
 - The routing glue which binds many networks into the Internet!
 - Protocols used are called EGPs or *Exterior Gateway Protocols*
 - Only one is ever used at a time! All ASes agree!
 - Internet has used BGP since the mid-1990s

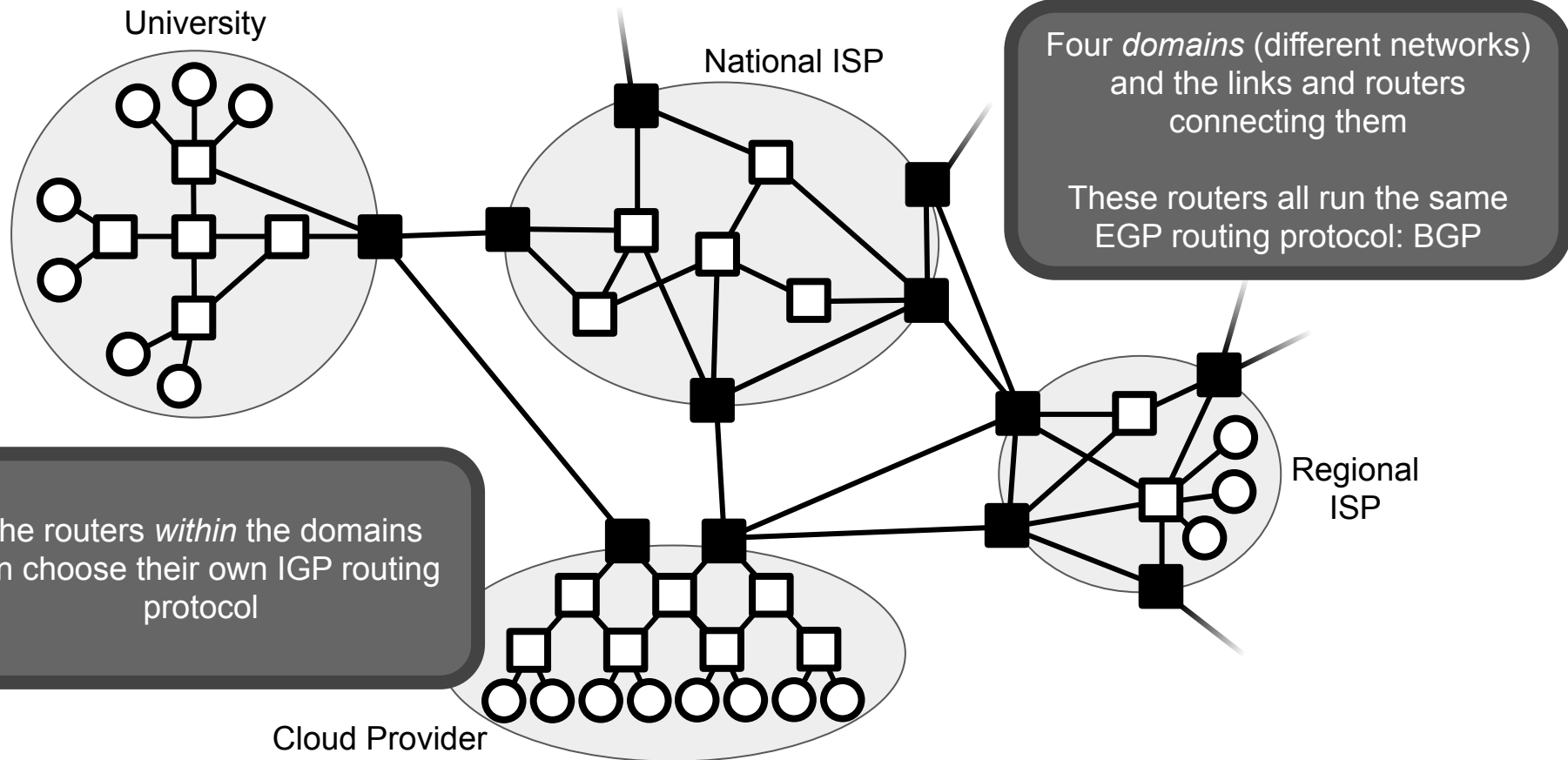
Interdomain and Intradomain Routing

- Intradomain routing
 - More or less means *routing within a single network* (technically an “Autonomous System”)
 - Protocols used are often called IGPs or *Interior Gateway Protocols*
 - A number are actively used today
- Interdomain routing
 - Routing *between* networks (ASes)
 - The routing glue which binds many networks into the Internet!
 - Protocols used are called EGPs or *Exterior Gateway Protocols*
 - Only one is ever used at a time! All ASes agree!
 - Internet has used BGP since the mid-1990s
- Much of discussion this & next week is very general (true of any routing)
 - .. but focus is on intradomain routing
 - .. we’ll talk about BGP specifically later (week 7?)

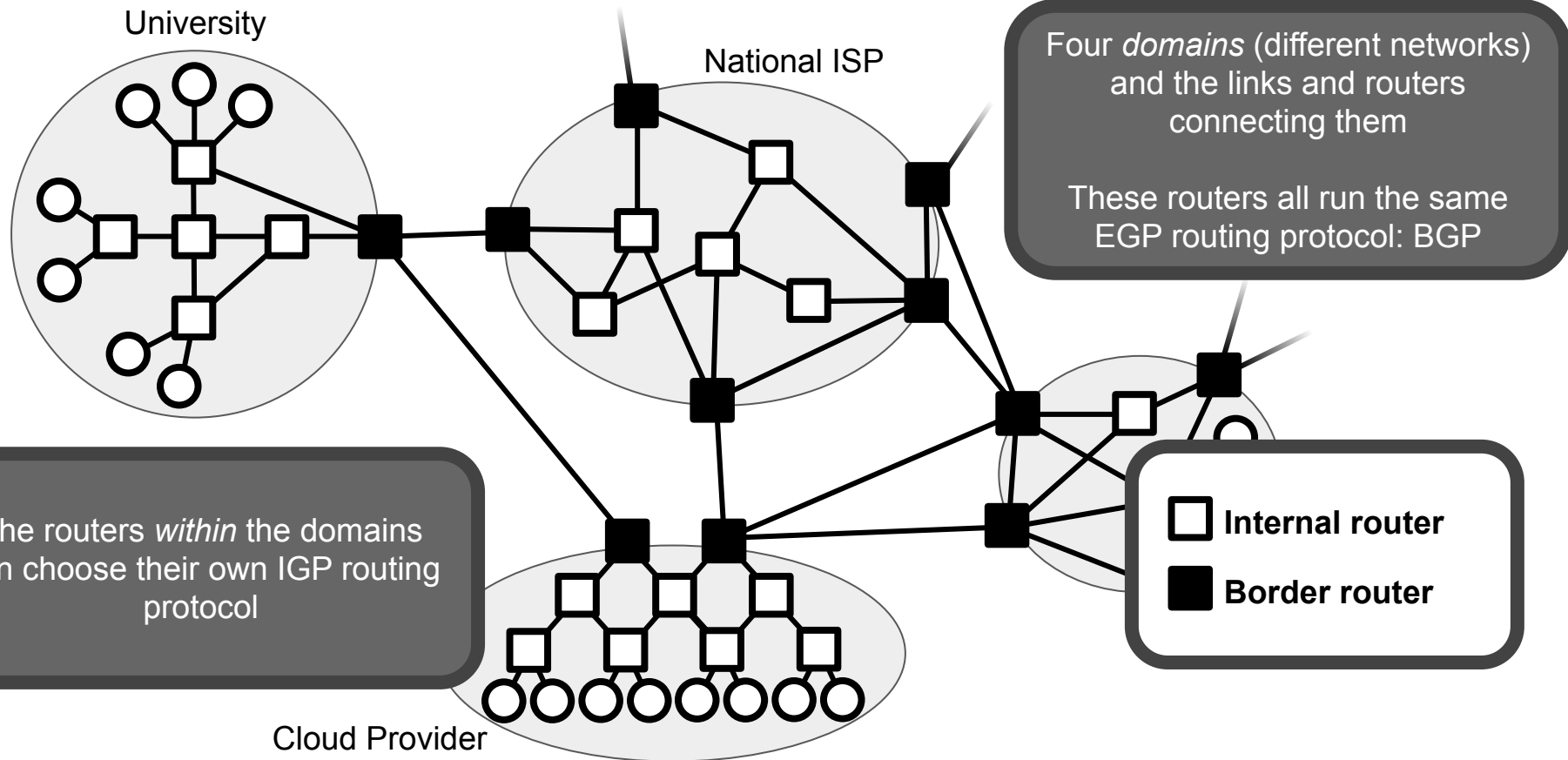
Interdomain and Intradomain Routing



Interdomain and Intradomain Routing



Interdomain and Intradomain Routing



Questions?

Least-Cost Routing

Least-Cost Routing

- Last time, we said we wanted “good” routes

Least-Cost Routing

- Last time, we said we wanted “good” routes
- Goal #1: Routes that work!
 - State must not have any loops. Must not have any dead ends. Both of these.

Least-Cost Routing

- Last time, we said we wanted “good” routes
- Goal #1: Routes that work!
 - State must not have any loops. Must not have any dead ends. Both of these.
- Goal #2: Routes that are in some way “good”
 - Commonly this is done by *minimizing* some “bad” quantity which we might call a *cost*
 - Hence *least-cost routing!*

Least-Cost Routing

- Last time, we said we wanted “good” routes
- Goal #1: Routes that work!
 - State must not have any loops. Must not have any dead ends. Both of these.
- Goal #2: Routes that are in some way “good”
 - Commonly this is done by *minimizing* some “bad” quantity which we might call a *cost*
 - Hence *least-cost routing*!
- What did we (attempt to?) minimize in the routing activity we did last time?

Least-Cost Routing

- Last time, we said we wanted “good” routes
- Goal #1: Routes that work!
 - State must not have any loops. Must not have any dead ends. Both of these.
- Goal #2: Routes that are in some way “good”
 - Commonly this is done by *minimizing* some “bad” quantity which we might call a *cost*
 - Hence *least-cost routing!*
- What did we (attempt to?) minimize in the routing activity we did last time?
 - Number of people who handled the envelope -- the *hop count*

Least-Cost Routing

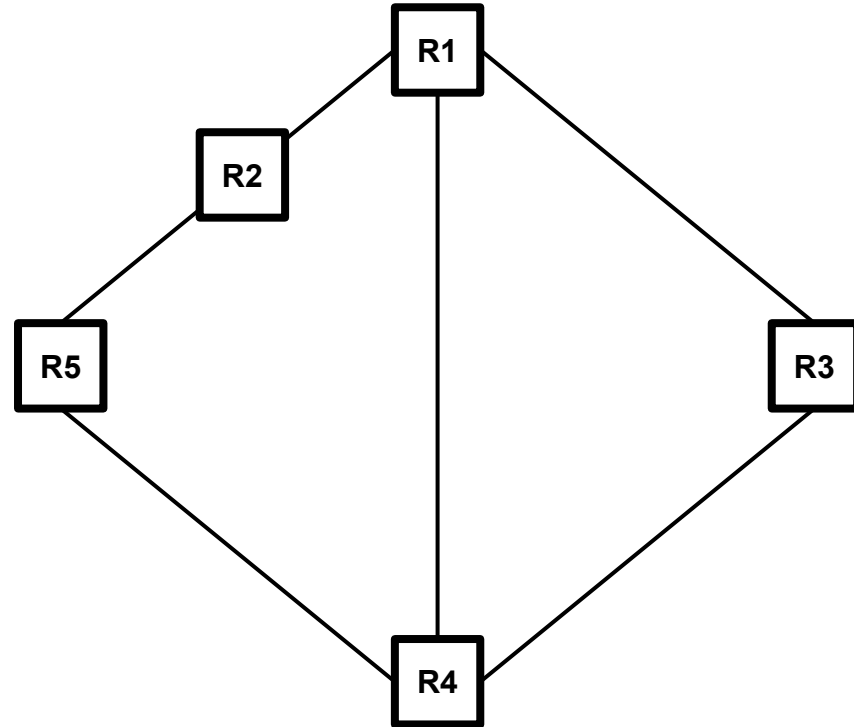
- Last time, we said we wanted “good” routes
- Goal #1: Routes that work!
 - State must not have any loops. Must not have any dead ends. Both of these.
- Goal #2: Routes that are in some way “good”
 - Commonly this is done by *minimizing* some “bad” quantity which we might call a *cost*
 - Hence *least-cost routing*!
- What did we (attempt to?) minimize in the routing activity we did last time?
 - Number of people who handled the envelope -- the *hop count*
- What else might we minimize? (I mentioned some last time...)

Least-Cost Routing

- Last time, we said we wanted “good” routes
- Goal #1: Routes that work!
 - State must not have any loops. Must not have any dead ends. Both of these.
- Goal #2: Routes that are in some way “good”
 - Commonly this is done by *minimizing* some “bad” quantity which we might call a *cost*
 - Hence *least-cost routing!*
- What did we (attempt to?) minimize in the routing activity we did last time?
 - Number of people who handled the envelope -- the *hop count*
- What else might we minimize? (I mentioned some last time...)
 - Price, propagation delay, distance, unreliability, others, ...
 - .. we can sort of just abstract this away

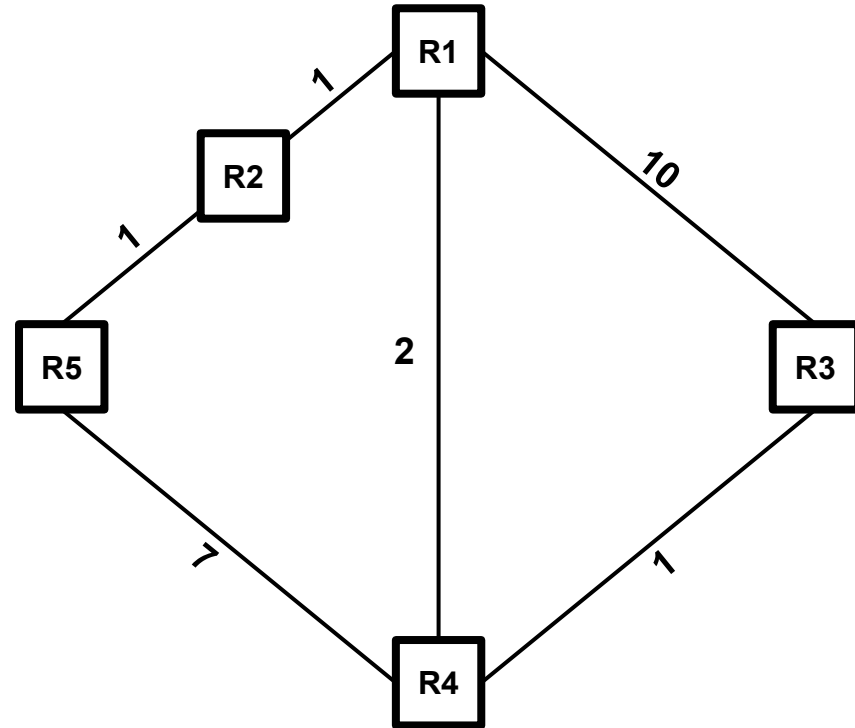
Least-Cost Routing

- So if we have a topology like this...



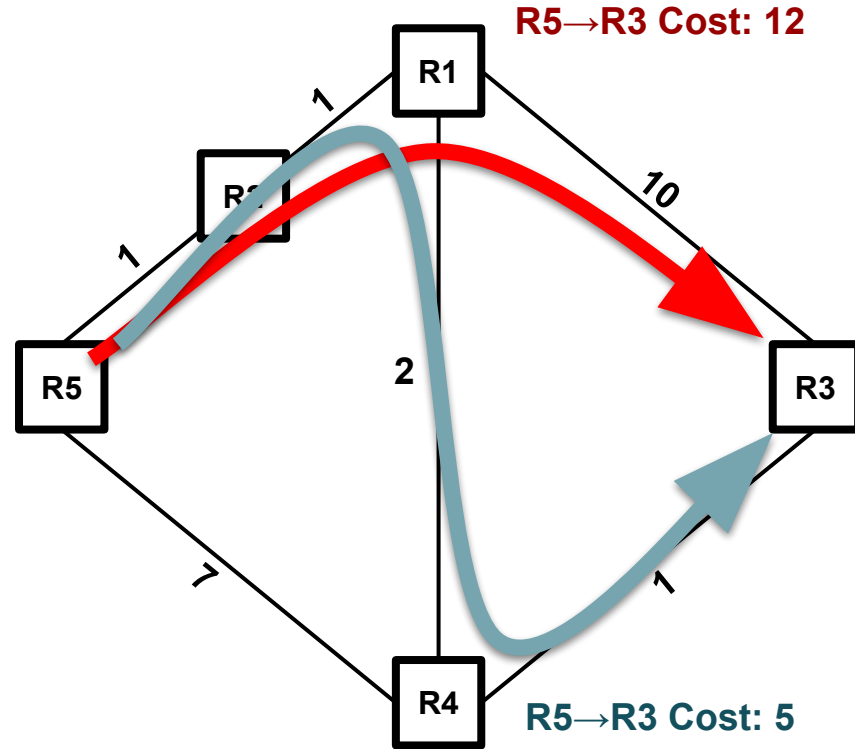
Least-Cost Routing

- So if we have a topology like this...
- .. we associate a cost with each edge



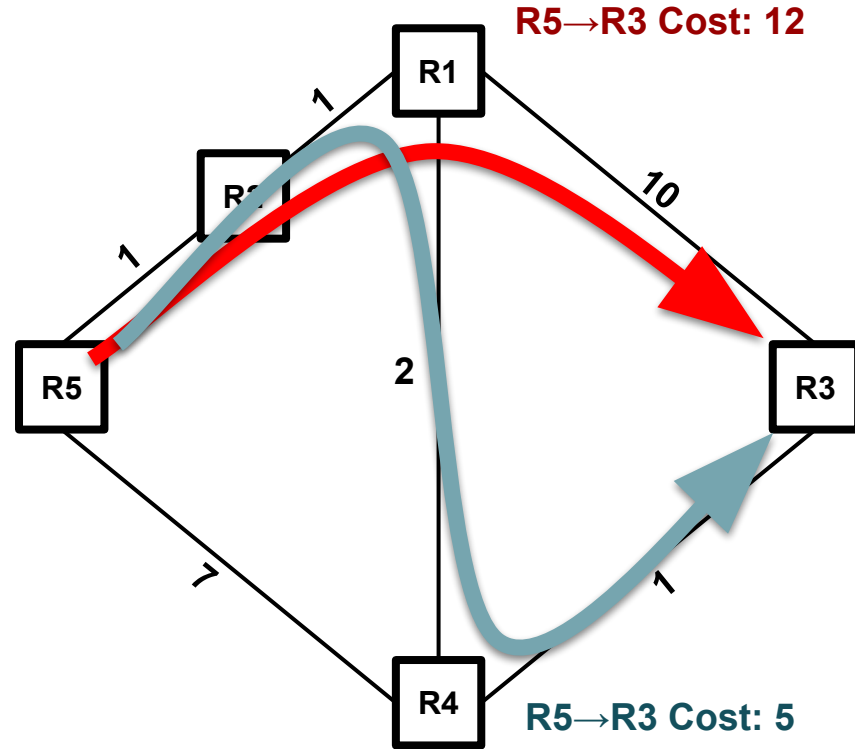
Least-Cost Routing

- So if we have a topology like this...
- .. we associate a cost with each edge
- .. and find path with the smallest sum



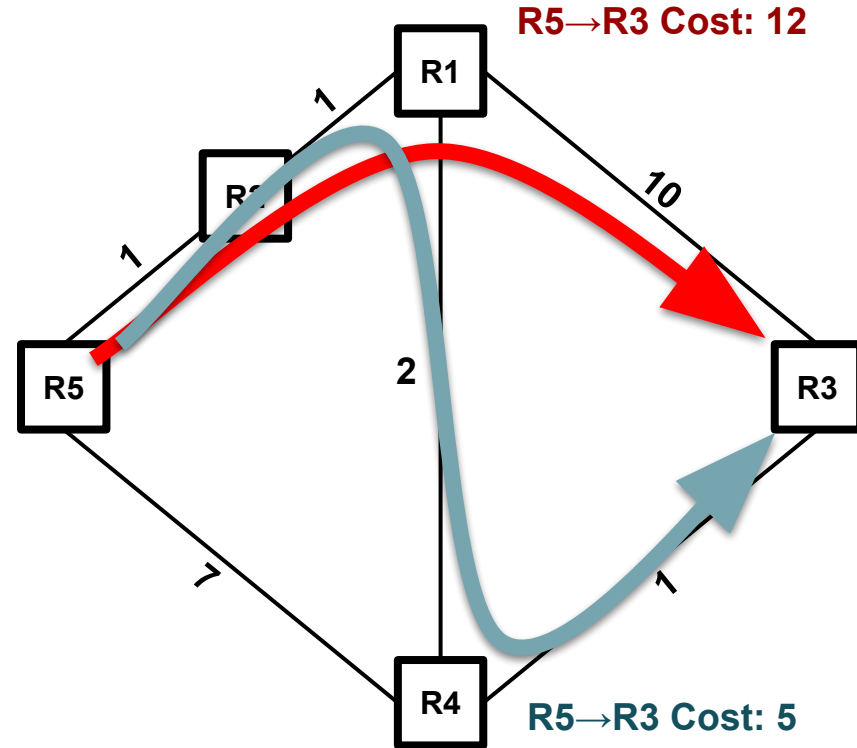
Least-Cost Routing

- So if we have a topology like this...
- .. we associate a cost with each edge
- .. and find path with the smallest sum
- .. you've probably seen this before!



Least-Cost Routing

- So if we have a topology like this...
- .. we associate a cost with each edge
- .. and find path with the smallest sum
- .. you've probably seen this before!
- In routing activity, every "edge" had cost of 1 -- gives you *hop count*
 - If costs not given, assume 1



Least-Cost Routing

- Where do these costs come from?
- Generally, local to router
 - That is, routers know the costs of their attached links
- May be configured by an operator
- May be determined automatically
 - OSPF (an intradomain routing protocol) uses the link bandwidth
 - Higher bandwidth = smaller cost (gets highest-average-bandwidth paths)

Least-Cost Routing

- Least cost routes are an easy way to avoid loops
 - No sensible cost metric is minimized by traversing a loop
- Least cost routes are destination-based
 - Only depend on the destination
- They form a spanning tree
 - (Hence no loops)

Least-Cost Routing

- Least cost routes are an easy way to avoid loops
 - No sensible cost metric is minimized by traversing a loop
- Least cost routes are destination-based
 - Only depend on the destination
- They form a spanning tree
 - (Hence no loops)
- Terminology note:
 - When I say “shortest path”, I mean “least cost path”
 - When weights are 1, these are the same thing -- the hop count
 - The text uses “shortest path” to always mean hop count

Questions?

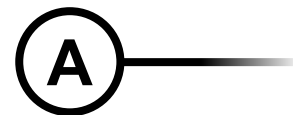
Trivial and Static Routes

Trivial Routes

- There are a couple kinds of routes which...
 - .. are uninteresting
 - .. you probably get “for free”
 - .. I’ll often ignore
- I’ll call these “*trivial routes*” (not a standard term)

Trivial Routes

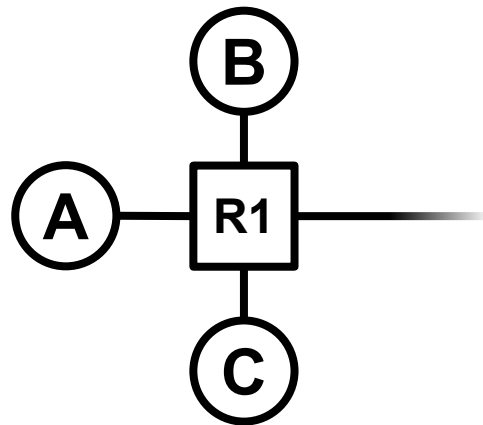
- There are a couple kinds of routes which...
 - .. are uninteresting
 - .. you probably get “for free”
 - .. I’ll often ignore
- I’ll call these “*trivial routes*” (not a standard term)
- #1: A route to yourself
 - With cost of zero!



<i>A's Table</i>	
<i>Dst</i>	<i>NextHop, Cost</i>
A	None, 0
...	

Trivial Routes

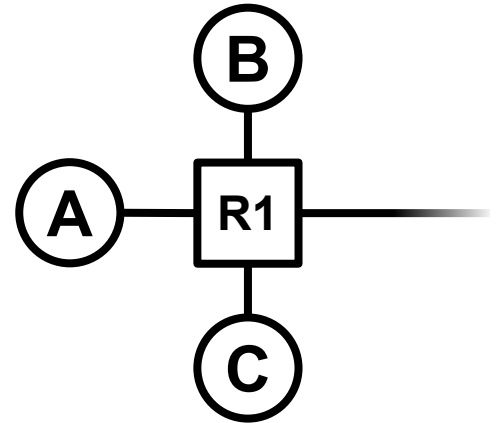
- There are a couple kinds of routes which...
 - .. are uninteresting
 - .. you probably get “for free”
 - .. I’ll often ignore
- I’ll call these “*trivial routes*” (not a standard term)
- #1: A route to yourself
 - With cost of zero!
- #2: If you have only one neighbor, a *default route*
 - Cost doesn’t matter; it’s the only way!



<i>A's Table</i>	
<i>Dst</i>	<i>NextHop, Cost</i>
Any	R1, 1

Trivial Routes

- There are a couple kinds of routes which...
 - .. are uninteresting
 - .. you probably get “for free”
 - .. I’ll often ignore
- I’ll call these “*trivial routes*” (not a standard term)
- #1: A route to yourself
 - With cost of zero!
- #2: If you have only one neighbor, a *default route*
 - Cost doesn’t matter; it’s the only way!
 - Hosts with multiple neighbors usually have one anyway
 - e.g., use WiFi by default if available; not cellular!



A's Table	
<i>Dst</i>	<i>NextHop, Cost</i>
Any	R1, 1

Static Routes

- *Static routes* are entered manually by an operator
- Why would you do this?

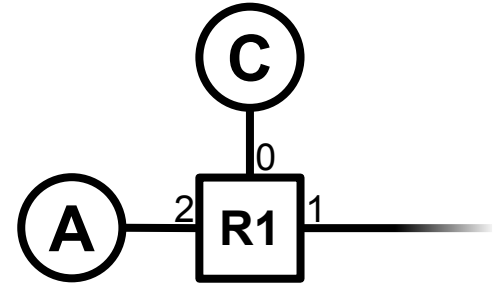
Static Routes

- *Static routes* are entered manually by an operator
- Why would you do this?

- Sometimes operator just knows what they want!

Static Routes

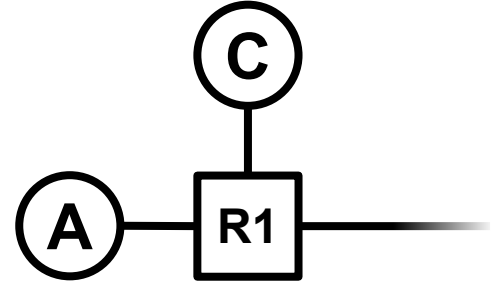
- *Static routes* are entered manually by an operator
- Why would you do this?
- Sometimes operator just knows what they want!
- More importantly:
 - Hosts don't generally participate in routing protocols
 - So how do the routers know where the hosts are?!
 - Operator adds static route on router next to host



<i>R1's Table</i>	
<i>Dst</i>	<i>Port, Cost</i>
A	Port2, 1 (static)
B	Port1, 6
C	Port0, 1 (static)
...	

Static Routes

- *Static routes* are entered manually by an operator
- Why would you do this?
- Sometimes operator just knows what they want!
- More importantly:
 - Hosts don't generally participate in routing protocols
 - So how do the routers know where the hosts are?!
 - Operator adds static route on router next to host



<i>R1's Table</i>	
<i>Dst</i>	<i>NextHop, Cost</i>
A	A (or Direct), 1
B	R5, 6
C	C (or Direct), 1
...	

Blue arrows point from the left to the 'A' and 'C' rows of the table.

Questions?

Distance-Vector Protocols

Distance-Vector Routing Protocols

- Long history on the Internet (and its predecessor ARPANET in 1969)
- “Prototypical” D-V protocol is RIP (Routing Information Protocol)
 - Our discussion of D-V pretty similar
- Strong relationship to Bellman-Ford shortest path algorithm
 - Our in-class exercise last time was basically a version of Bellman-Ford
 - .. sort of half way between normal Bellman-Ford and a useful routing protocol
 - .. we’ll much closer today!
- The text has a much more formal/algorithmic write-up of distance-vector
 - You are welcome to read it!
- I am going to try to give you a sense of how it actually works.

Thinking back to the activity...

Thinking back to the activity (and making changes)...

Thinking back to the activity (and making changes)...

- You remembered:
 - Your *distance* to destination (lan) along the best path *as far as you know* (your magic number)

Thinking back to the activity (and making changes)...

- You remembered:
 - Your *distance* to destination (lan) along the best path *as far as you know* (your magic number)
 - Which neighbor is along that best path (the *nexthop*, AKA your best friend)

Thinking back to the activity (and making changes)...

- You remembered:
 - Your *distance* to destination (lan) along the best path *as far as you know* (your magic number)
 - Which neighbor is along that best path (the *nexthop*, AKA your best friend)
 - Maybe you didn't *actually* remember and figured it out later
 - But you *could* have remembered, right?

Thinking back to the activity (and making changes)...

- You remembered:
 - Your *distance* to destination (lan) along the best path *as far as you know* (your magic number)
 - Which neighbor is along that best path (the *nexthop*, AKA your best friend)
 - Maybe you didn't *actually* remember and figured it out later
 - But you *could* have remembered, right?
- When you changed your mind about distance:
 - You told your neighbors (this might change *their* mind!)

Thinking back to the activity (and making changes)...

- You remembered:
 - Your *distance* to destination (lan) along the best path *as far as you know* (your magic number)
 - Which neighbor is along that best path (the *nexthop*, AKA your best friend)
 - Maybe you didn't *actually* remember and figured it out later
 - But you *could* have remembered, right?
- When you changed your mind about distance:
 - You told your neighbors (this might change *their* mind!)
 - You added the additional distance to your neighbor (offering them your number plus one)
 - Could have done it the other way around, right?
 - Could have told you *their* number, and *you* added 1 and compared
 - And you *could* have added some other number, right?

Thinking back to the activity (and making changes)...

- You remembered:
 - Your *distance* to destination (lan) along the best path *as far as you know* (your magic number)
 - Which neighbor is along that best path (the *nexthop*, AKA your best friend)
 - Maybe you didn't *actually* remember and figured it out later
 - But you *could* have remembered, right?
- When you changed your mind about distance:
 - You told your neighbors (this might change *their* mind!)
 - You added the additional distance to your neighbor (offering them your number plus one)
 - Could have done it the other way around, right?
 - Could have told you *their* number, and *you* added 1 and compared
 - And you *could* have added some other number, right?
- You *could* have done this for multiple destinations at once, right?
 - Offer neighbors your best lan number, Shriya number, Rafael number, ...

An aside: Routing vs. Forwarding

Routing

- Communicate with other routers to determine how to populate tables for forwarding

Forwarding

- Looks up packet's destination in table and sends packet to given neighbor

An aside: Routing vs. Forwarding

Routing

- Communicate with other routers to determine how to populate tables for forwarding
- Figuring out best friends by sharing magic numbers with neighbors

Forwarding

- Looks up packet's destination in table and sends packet to given neighbor

An aside: Routing vs. Forwarding

Routing

- Communicate with other routers to determine how to populate tables for forwarding
- Figuring out best friends by sharing magic numbers with neighbors

Forwarding

- Looks up packet's destination in table and sends packet to given neighbor



An aside: Routing vs. Forwarding

Routing

- Communicate with other routers to determine how to populate tables for forwarding
- Figuring out best friends by sharing magic numbers with neighbors

Forwarding

- Looks up packet's destination in table and sends packet to given neighbor
- Passing envelopes to our best friends

In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...

```
def bellman_ford (dst, routers, links):
    distance = {}; nexthop = {}

    for each r in routers:
        distance[r] = INFINITY
        nexthop[r] = None
    distance[dst] = 0

    for _ in range(len(routers)-1):
        for (r1,r2,dist) in links:
            if distance[r1] + dist < distance[r2]:
                distance[r2] = distance[r1] + dist
                nexthop[r2] = r1

    return distance, nexthop
```

In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...

```
def bellman_ford (dst, routers, links):  
    distance = {}; nexthop = {}
```

- We can see things we recognize:

- Magic number & best friend

```
    for each r in routers:  
        distance[r] = INFINITY  
        nexthop[r] = None  
    distance[dst] = 0  
  
    for _ in range(len(routers)-1):  
        for (r1,r2,dist) in links:  
            if distance[r1] + dist < distance[r2]:  
                distance[r2] = distance[r1] + dist  
                nexthop[r2] = r1  
  
    return distance, nexthop
```


In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...

```
def bellman_ford (dst, routers, links):  
    distance = {}; nexthop = {}
```

- We can see things we recognize:

- Magic number & best friend
- Start with infinity (except destination)



```
    for each r in routers:  
        distance[r] = INFINITY  
        nexthop[r] = None  
    distance[dst] = 0  
  
    for _ in range(len(routers)-1):  
        for (r1,r2,dist) in links:  
            if distance[r1] + dist < distance[r2]:  
                distance[r2] = distance[r1] + dist  
                nexthop[r2] = r1  
  
    return distance, nexthop
```

In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...
- We can see things we recognize:
 - Magic number & best friend
 - Start with infinity (except destination)
 - Compare offer to current


```
def bellman_ford (dst, routers, links):  
    distance = {}; nexthop = {}  
  
    for each r in routers:  
        distance[r] = INFINITY  
        nexthop[r] = None  
    distance[dst] = 0  
  
    for _ in range(len(routers)-1):  
        for (r1,r2,dist) in links:  
            if distance[r1] + dist < distance[r2]:  
                distance[r2] = distance[r1] + dist  
                nexthop[r2] = r1  
  
    return distance, nexthop
```



In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...
- We can see things we recognize:
 - Magic number & best friend
 - Start with infinity (except destination)
 - Compare offer to current
 - Accept offer

```
def bellman_ford (dst, routers, links):  
    distance = {}; nexthop = {}  
  
    for each r in routers:  
        distance[r] = INFINITY  
        nexthop[r] = None  
    distance[dst] = 0  
  
    for _ in range(len(routers)-1):  
        for (r1,r2,dist) in links:  
            if distance[r1] + dist < distance[r2]:  
                distance[r2] = distance[r1] + dist  
                nexthop[r2] = r1  
  
    return distance, nexthop
```



In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...
- We can see things we recognize:
 - Magic number & best friend
 - Start with infinity (except destination)
 - Compare offer to current
 - Accept offer
 - Remember best friend

```
def bellman_ford (dst, routers, links):  
    distance = {}; nexthop = {}  
  
    for each r in routers:  
        distance[r] = INFINITY  
        nexthop[r] = None  
    distance[dst] = 0  
  
    for _ in range(len(routers)-1):  
        for (r1,r2,dist) in links:  
            if distance[r1] + dist < distance[r2]:  
                distance[r2] = distance[r1] + dist  
                nexthop[r2] = r1  
  
    return distance, nexthop
```



In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...

```
def bellman_ford (dst, routers, links):  
    distance = {}; nexthop = {}
```

- We can see things we recognize:

- Magic number & best friend
- Start with infinity (except destination)
- Compare offer to current
- Accept offer
- Remember best friend



```
    for each r in routers:  
        distance[r] = INFINITY  
        nexthop[r] = None  
    distance[dst] = 0  
  
    for _ in range(len(routers)-1):  
        for (r1,r2,dist) in links:  
            if distance[r1] + dist < distance[r2]:  
                distance[r2] = distance[r1] + dist  
                nexthop[r2] = r1
```

- .. but we did it in parallel!

```
    return distance, nexthop
```

In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...
- We can see things we recognize:
 - Magic number & best friend
 - Start with infinity (except destination)
 - Compare offer to current
 - Accept offer
 - Remember best friend
- .. but we did it in parallel!
 - Nobody iterated over *all* the links
 - .. only between you and neighbors

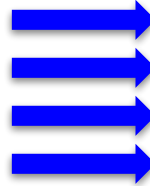
```
def bellman_ford (dst, routers, links):  
    distance = {}; nexthop = {}  
  
    for each r in routers:  
        distance[r] = INFINITY  
        nexthop[r] = None  
    distance[dst] = 0  
  
    for _ in range(len(routers)-1):  
        for (r1,r2,dist) in links:  
            if distance[r1] + dist < distance[r2]:  
                distance[r2] = distance[r1] + dist  
                nexthop[r2] = r1  
  
    return distance, nexthop
```



In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...
- We can see things we recognize:
 - Magic number & best friend
 - Start with infinity (except destination)
 - Compare offer to current
 - Accept offer
 - Remember best friend
- .. but we did it in parallel!
- .. and asynchronously
 - You and others doing all this work in no strict order order...

```
def bellman_ford (dst, routers, links):  
    distance = {}; nexthop = {}  
  
    for each r in routers:  
        distance[r] = INFINITY  
        nexthop[r] = None  
    distance[dst] = 0  
  
    for _ in range(len(routers)-1):  
        for (r1,r2,dist) in links:  
            if distance[r1] + dist < distance[r2]:  
                distance[r2] = distance[r1] + dist  
                nexthop[r2] = r1  
  
    return distance, nexthop
```



In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...
- We can see things we recognize:
 - Magic number & best friend
 - Start with infinity (except destination)
 - Compare offer to current
 - Accept offer
 - Remember best friend
- .. but we did it in parallel!
- .. and asynchronously
- .. and what is this?!

```
def bellman_ford (dst, routers, links):  
    distance = {}; nexthop = {}  
  
    for each r in routers:  
        distance[r] = INFINITY  
        nexthop[r] = None  
    distance[dst] = 0  
  
    for _ in range(len(routers)-1):  
        for (r1,r2,dist) in links:  
            if distance[r1] + dist < distance[r2]:  
                distance[r2] = distance[r1] + dist  
                nexthop[r2] = r1  
  
    return distance, nexthop
```


In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...
- We can see things we recognize:
 - Magic number & best friend
 - Start with infinity (except destination)
 - Compare offer to current
 - Accept offer
 - Remember best friend
- .. but we did it in parallel!
- .. and asynchronously
- .. our version self-terminated
 - Eventually could only offer same thing to neighbor -- we *converged*

```
def bellman_ford (dst, routers, links):
    distance = {}; nexthop = {}

    for each r in routers:
        distance[r] = INFINITY
        nexthop[r] = None
    distance[dst] = 0

    for _ in range(len(routers)-1):
        for (r1,r2,dist) in links:
            if distance[r1] + dist < distance[r2]:
                distance[r2] = distance[r1] + dist
                nexthop[r2] = r1

    return distance, nexthop
```



In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...
- We can see things we recognize:
 - Magic number & best friend
 - Start with infinity (except destination)
 - Compare offer to current
 - Accept offer
 - Remember best friend
- .. but we did it in parallel!
- .. and asynchronously
- .. our version self-terminated
- .. nobody knew the whole topology!

```
def bellman_ford (dst, routers, links):  
    distance = {}; nexthop = {}  
  
    for each r in routers:  
        distance[r] = INFINITY  
        nexthop[r] = None  
    distance[dst] = 0  
  
    for _ in range(len(routers)-1):  
        for (r1,r2,dist) in links:  
            if distance[r1] + dist < distance[r2]:  
                distance[r2] = distance[r1] + dist  
                nexthop[r2] = r1  
  
    return distance, nexthop
```

In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...
- We can see things we recognize:
 - Magic number & best friend
 - Start with infinity (except destination)
 - Compare offer to current
 - Accept offer
 - Remember best friend
- .. but we did it in parallel!
- .. and asynchronously
- .. our version self-terminated
- .. nobody knew the whole topology!

```
def bellman_ford (dst, routers, links):
    distance = {}; nexthop = {}

    for each r in routers:
        distance[r] = INFINITY
        nexthop[r] = None
    distance[dst] = 0

    for _ in range(len(routers)-1):
        for (r1,r2,dist) in links:
            if distance[r1] + dist < distance[r2]:
                distance[r2] = distance[r1] + dist
                nexthop[r2] = r1

    return distance, nexthop
```

In-Class Activity and Bellman-Ford

- Serial Bellman-Ford: `def bellman_ford(routers, links):`
 - o `dist = {}`
 - o `for r1, r2, cost in links:`
 - o `dist[r2] = dist[r1] + cost`
 - o `for r1, r2, cost in links:`
 - o `if dist[r1] + cost < dist[r2]:`
 - o `dist[r2] = dist[r1] + cost`
 - o `return dist`
- We can see that Bellman-Ford is **slow** and **not asynchronous**
 - o Magic number 16
 - o Start with infinite distance
 - o Compare old and new distances
 - o Accept offer if better
 - o Remember to update neighbors
- .. but we did it **serially**
- .. and asynchronously
- .. our version serially terminated
- .. nobody knew the whole topology!


Why not Dijkstra's algorithm instead?
Isn't it faster?

Dijkstra's:
 $O(|E| + |V| \log |V|)$

Bellman-Ford:
 $O(|E| \cdot |V|)$

In-Class Activity and Bellman-Ford

- Serial Bellman-Ford algorithm...
- We can see things we recognize:
 - Magic number & best friend
 - Start with infinity (except destination)
 - Compare offer to current
 - Accept offer
 - Remember best friend
- .. but we did it in parallel!
- .. and asynchronously
- .. our version self-terminated
- .. nobody knew the whole topology!

```
def bellman_ford (dst, routers, links):  
    distance = {}; nexthop = {}  
  
    for each r in routers:  
        distance[r] = INFINITY  
        nexthop[r] = None  
    distance[dst] = 0  
  
    for _ in range(len(routers)-1):  
        for (r1,r2,dist) in links:  
             if distance[r1] + dist < distance[r2]:  
                distance[r2] = distance[r1] + dist  
                nexthop[r2] = r1  
  
    return distance, nexthop
```

Putting together Distance-Vector

<i>Your Table</i>	
<i>Dst</i>	<i>NextHop, Distance</i>
Ian	Person in front of me, 14

- Note: this can be used for forwarding too
 - (It has everything our tables from last time did *and* the distance)
- If a neighbor handed you an envelope (packet!), you'd hand to person in front

Putting together Distance-Vector

<i>Your Table</i>	
<i>Dst</i>	<i>NextHop, Distance</i>
lan	Person in front of me, 14

- Person to your left tells you “I can reach lan in 7”
 - They *advertised* a route with distance/cost of 7
- You think... well then I can reach lan in $7 + 1$...
 - Your neighbor’s distance to lan, plus your distance to the neighbor

Putting together Distance-Vector

<i>Your Table</i>	
<i>Dst</i>	<i>NextHop, Distance</i>
lan	Person in front of me, 14 Person to my left, 8

- Person to your left tells you “I can reach lan in 7”
 - They *advertised* a route with distance/cost of 7
- You think... well then I can reach lan in $7 + 1$...
 - Your neighbor’s distance to lan, plus your distance to the neighbor
 - Update your table with that!

Putting together Distance-Vector

<i>Your Table</i>	
<i>Dst</i>	<i>NextHop, Distance</i>
Ian	Person in front of me, 14 Person to my left, 8

- Person in front tells you, “I can reach Rafael in 3”
 - .. Rafael?

Putting together Distance-Vector

<i>Your Table</i>	
<i>Dst</i>	<i>NextHop, Distance</i>
Ian	Person in front of me, 14 Person to my left, 8
Rafael	Person in front of me, 4

- Person in front tells you, “I can reach Rafael in 3”
 - Add a new row for Rafael with *neighbor’s distance to Rafael .. plus your distance to the neighbor* ($3 + 1$)

Putting together Distance-Vector

<i>Your Table</i>	
<i>Dst</i>	<i>NextHop, Distance</i>
Ian	Person in front of me, 14 Person to my left, 8
Rafael	Person in front of me, 4
Shriya	Person to my right, 16 Person behind, 13 Person behind, 12
Jichan	Person behind me, 9

- Keep “processing” updates / advertisements from neighbors...
- Questions?

Distance-Vector

<i>Dst</i>	<i>Nxt, Cost</i>



<i>Dst</i>	<i>Nxt, Cost</i>

<i>Dst</i>	<i>Nxt, Cost</i>

Distance-Vector

<i>Dst</i>	<i>Nxt, Cost</i>

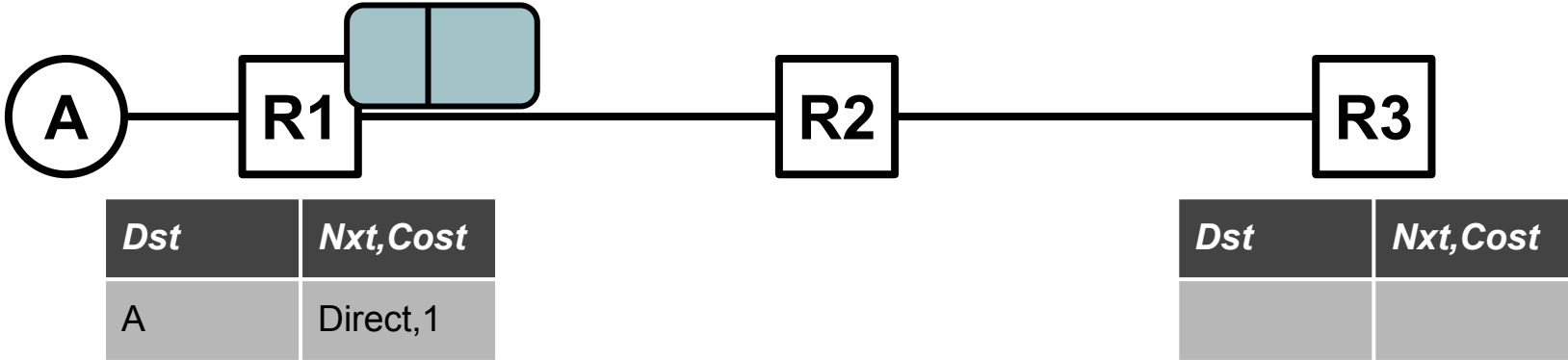


<i>Dst</i>	<i>Nxt, Cost</i>
A	Direct, 1

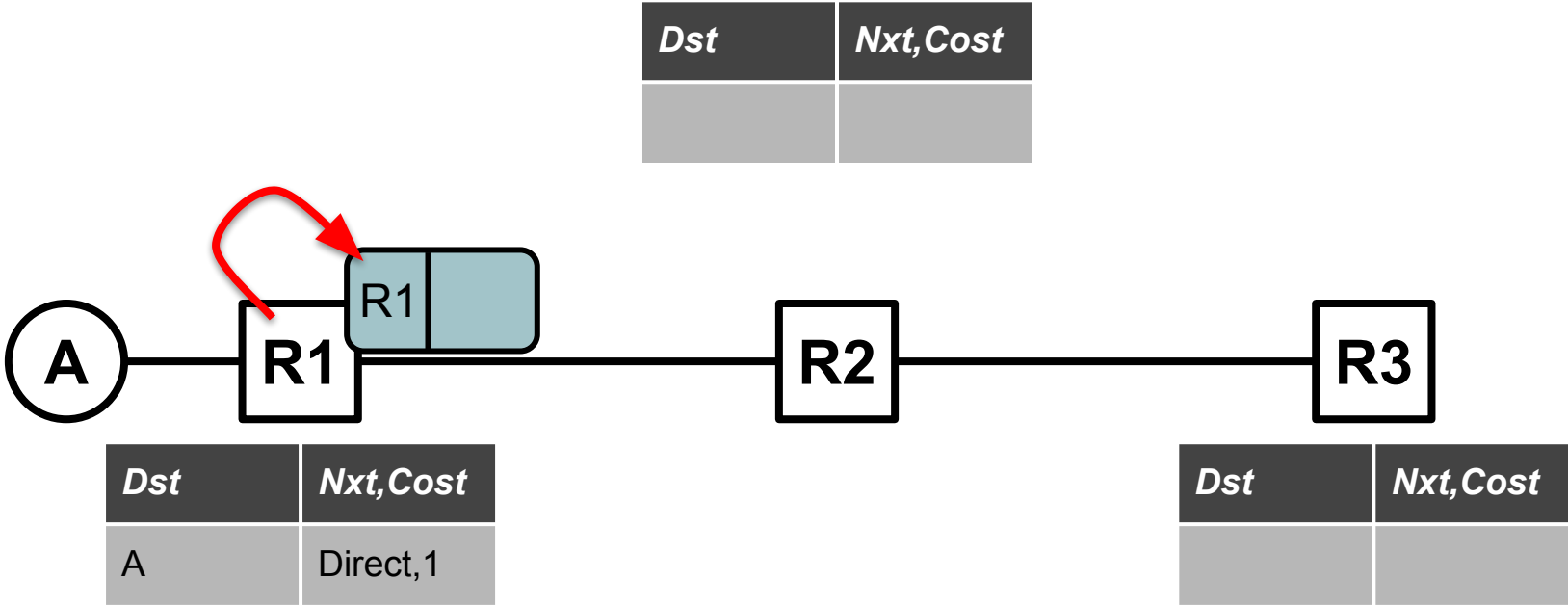
<i>Dst</i>	<i>Nxt, Cost</i>

Distance-Vector

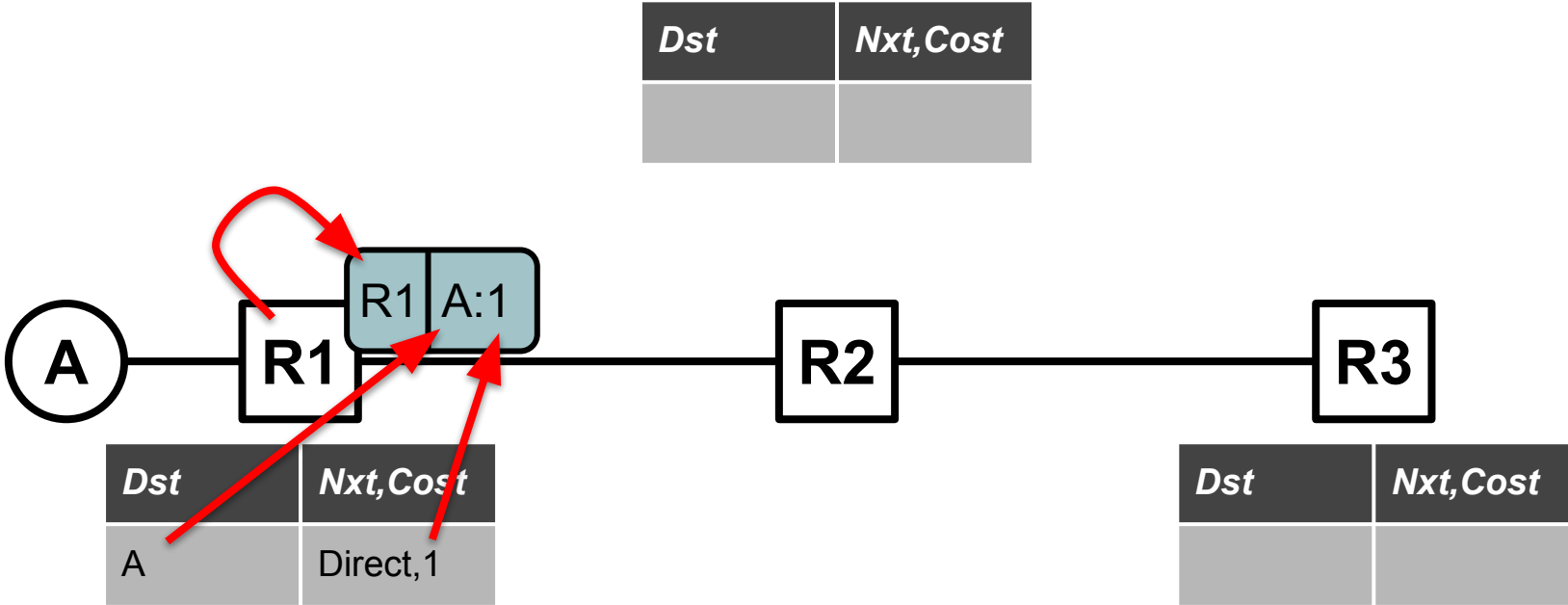
<i>Dst</i>	<i>Nxt, Cost</i>



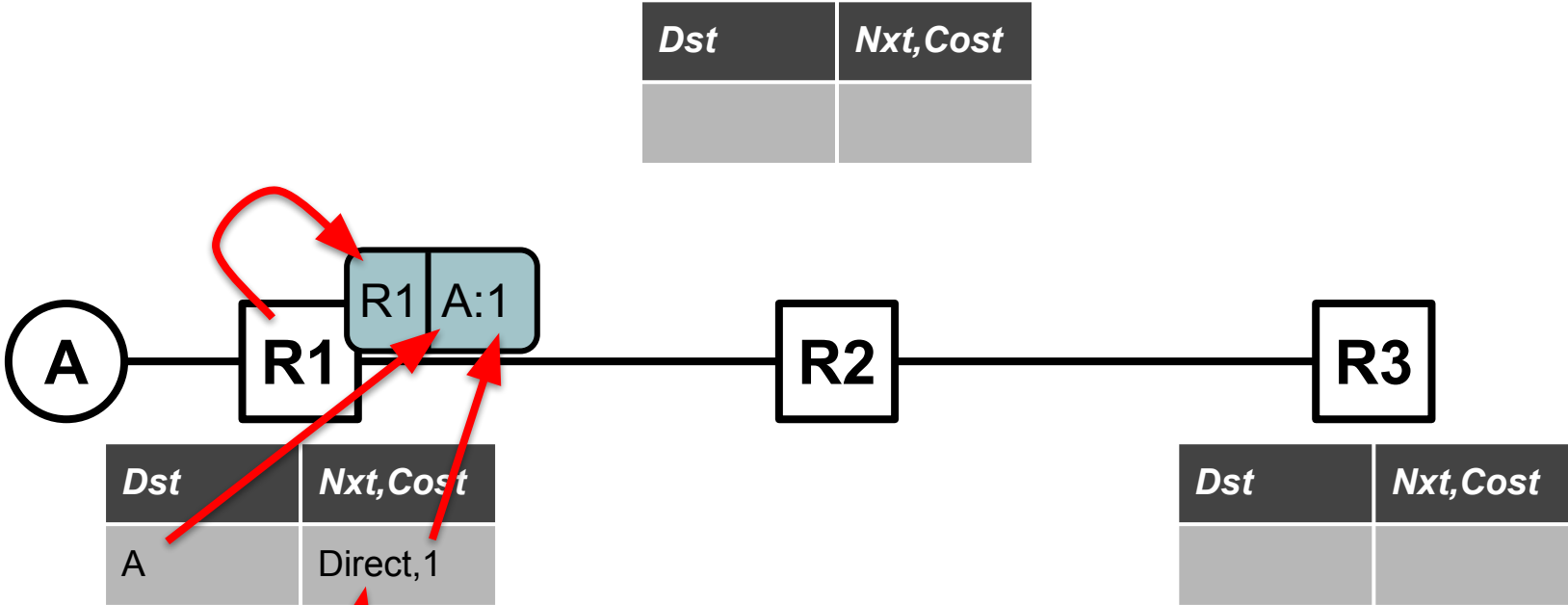
Distance-Vector



Distance-Vector

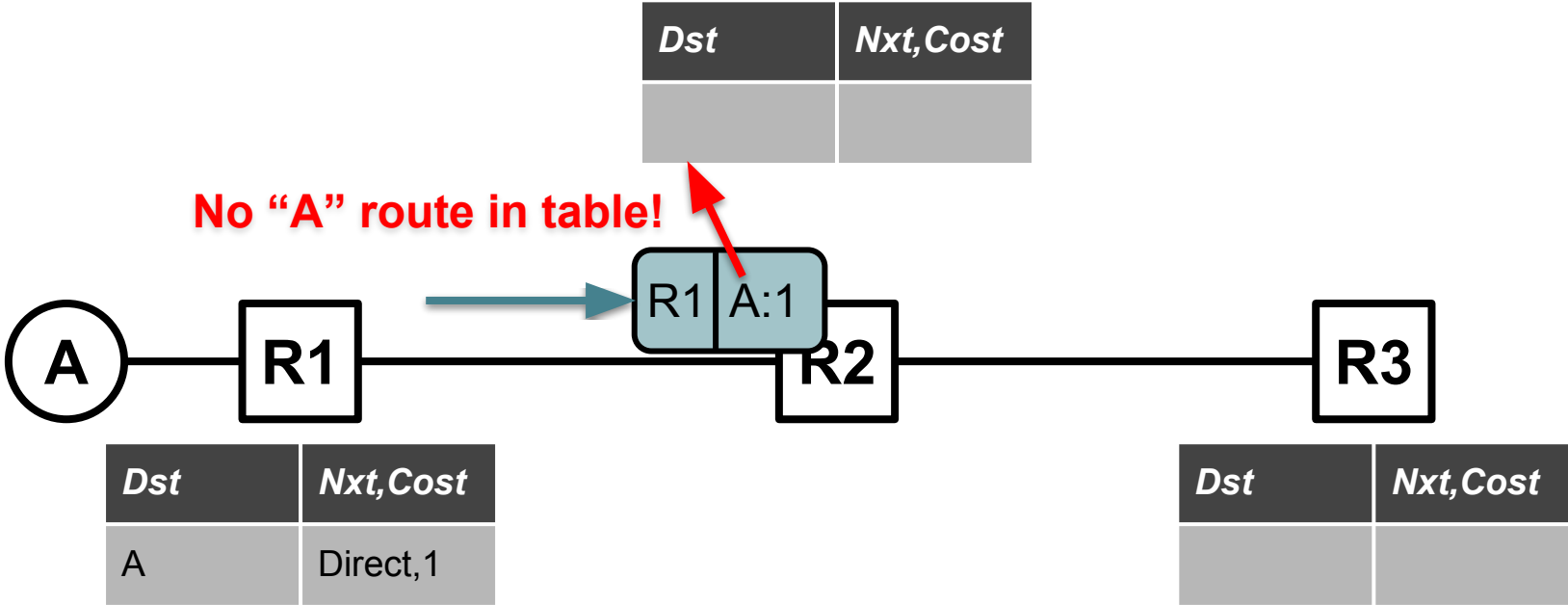


Distance-Vector

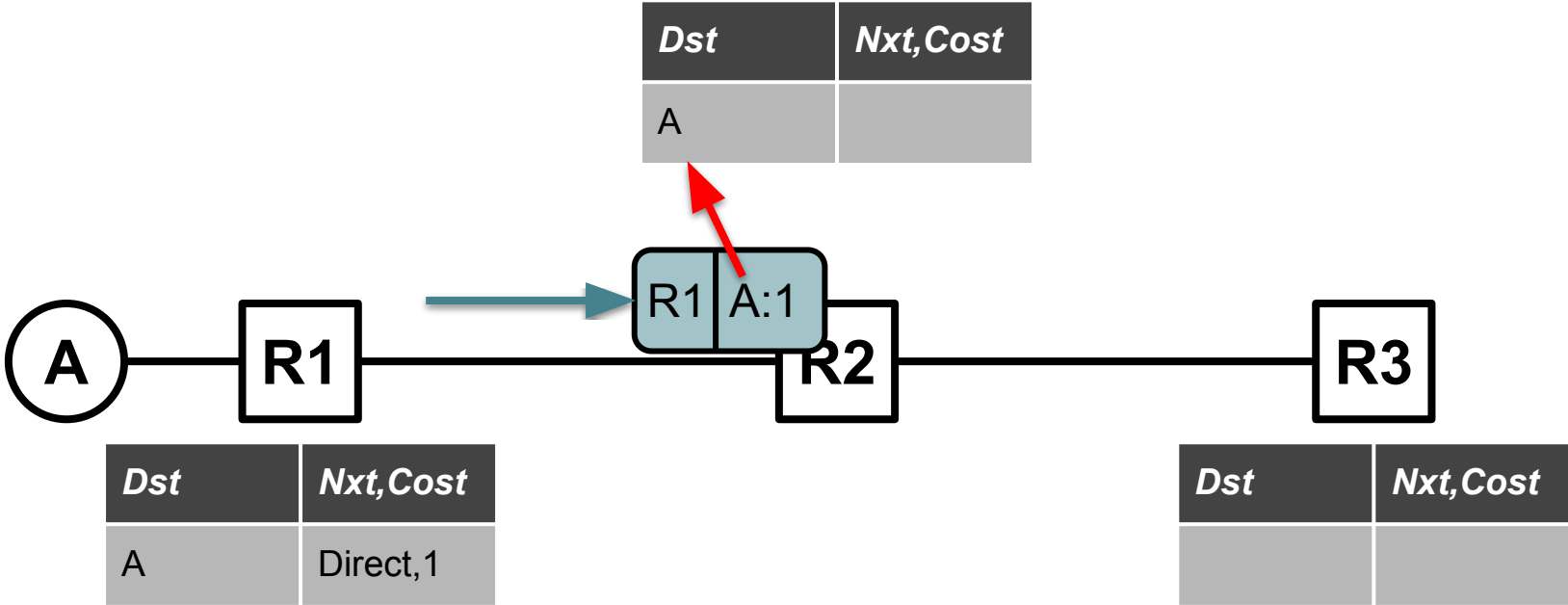


Only R1 needs to know its own next hop!

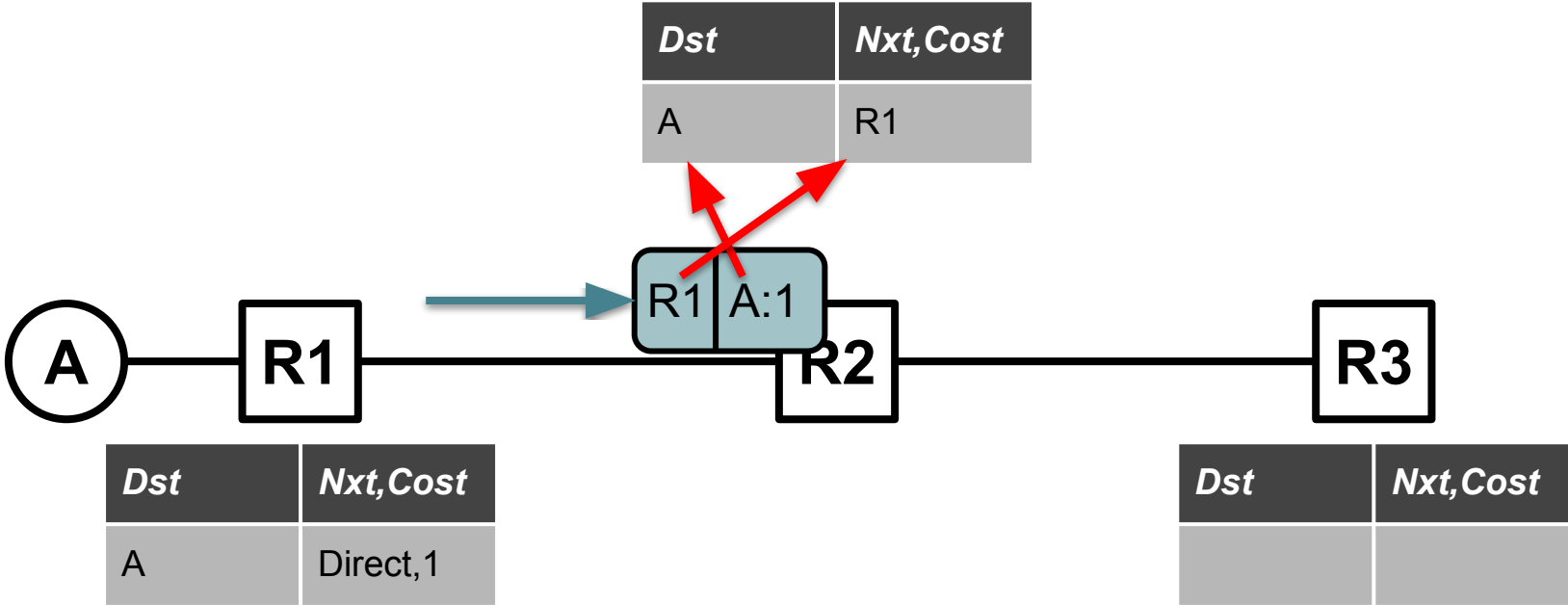
Distance-Vector



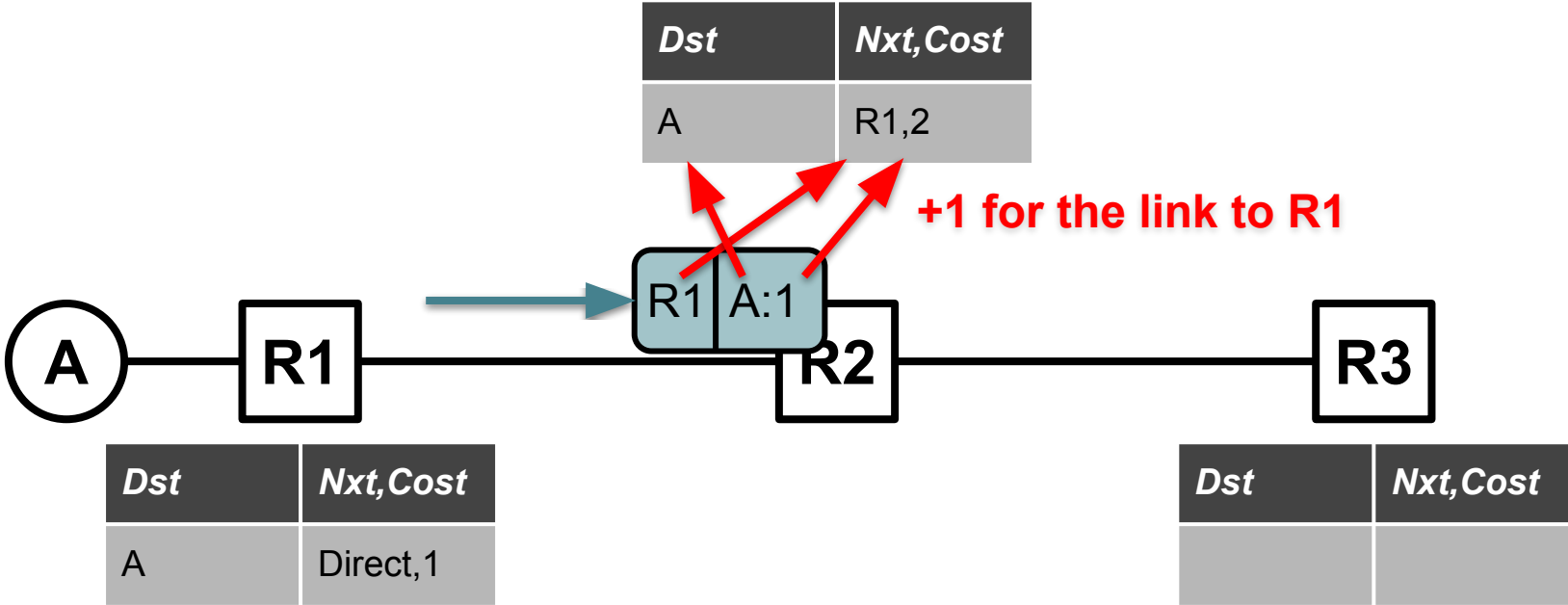
Distance-Vector



Distance-Vector

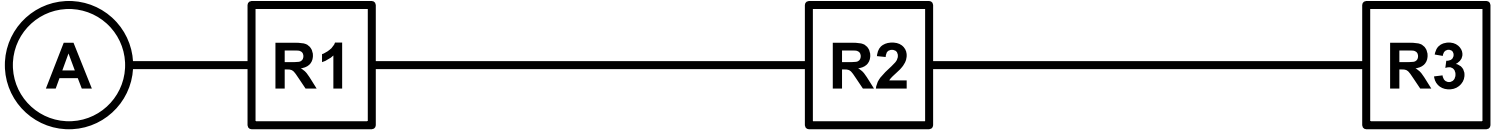


Distance-Vector



Distance-Vector

<i>Dst</i>	<i>Nxt, Cost</i>
A	R1,2



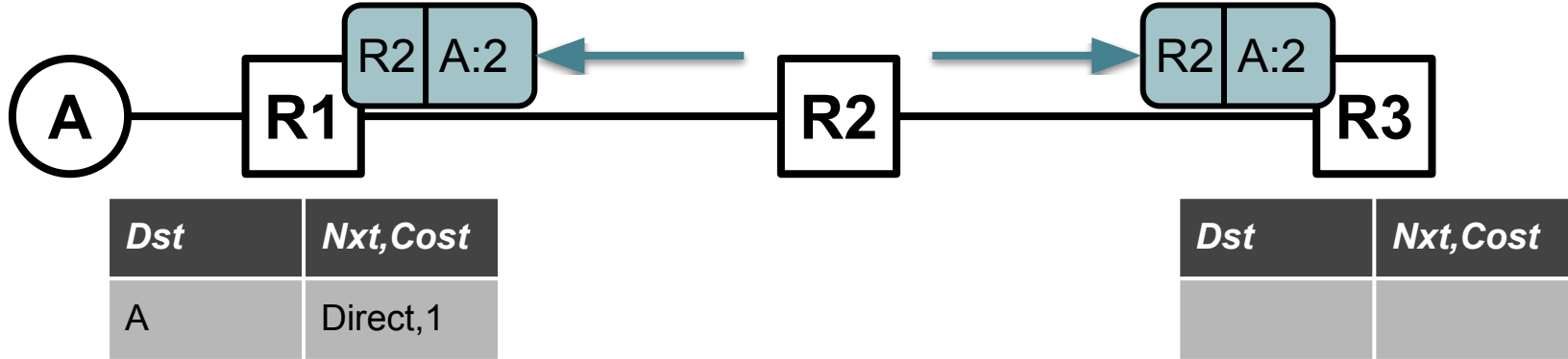
<i>Dst</i>	<i>Nxt, Cost</i>
A	Direct, 1

<i>Dst</i>	<i>Nxt, Cost</i>

Distance-Vector

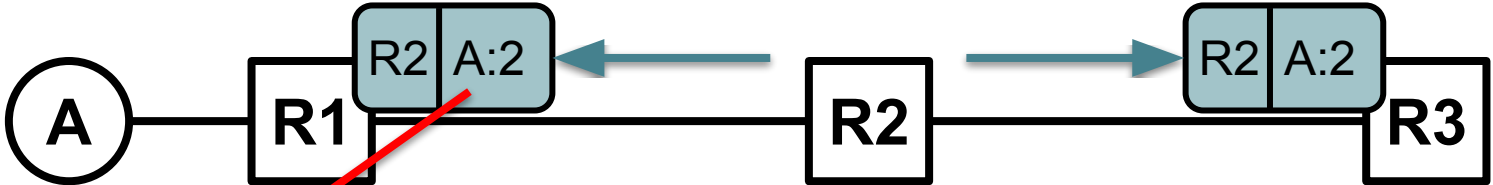
- If your neighbor offers you a lower number...
 - *Take it!* It's now your magic number
 - *Immediately* offer your magic number *plus one* to all your neighbors

I doubt you did this!



Distance-Vector

<i>Dst</i>	<i>Nxt, Cost</i>
A	R1,2



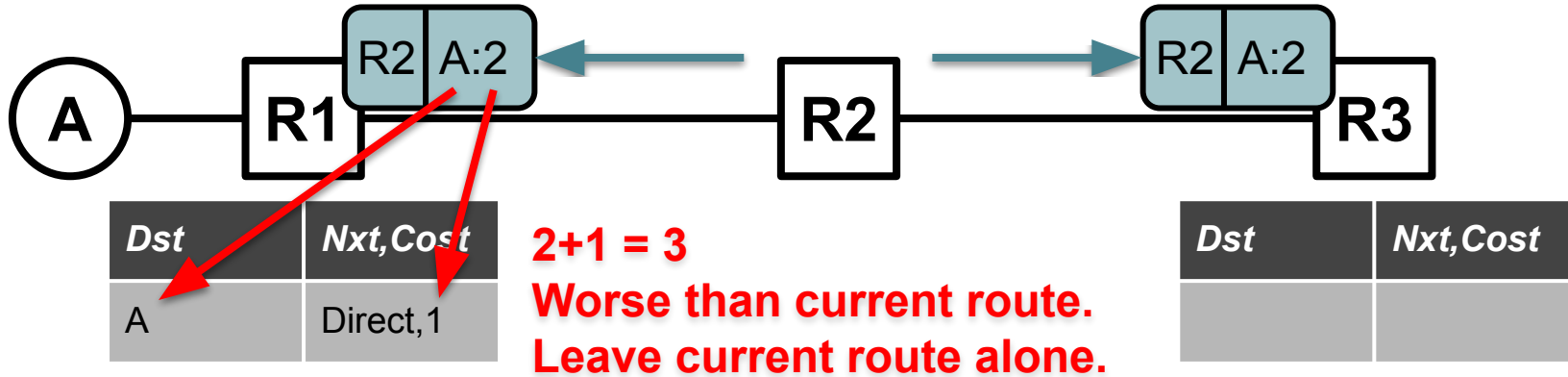
<i>Dst</i>	<i>Nxt, Cost</i>
A	Direct, 1

2+1 = 3

<i>Dst</i>	<i>Nxt, Cost</i>

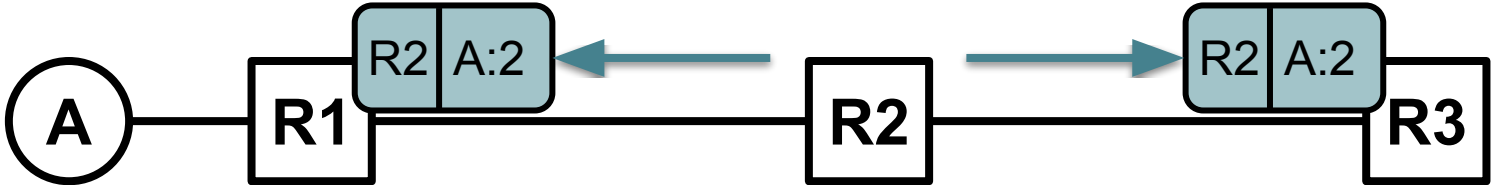
Distance-Vector

<i>Dst</i>	<i>Nxt, Cost</i>
A	R1,2



Distance-Vector

<i>Dst</i>	<i>Nxt, Cost</i>
A	R1,2

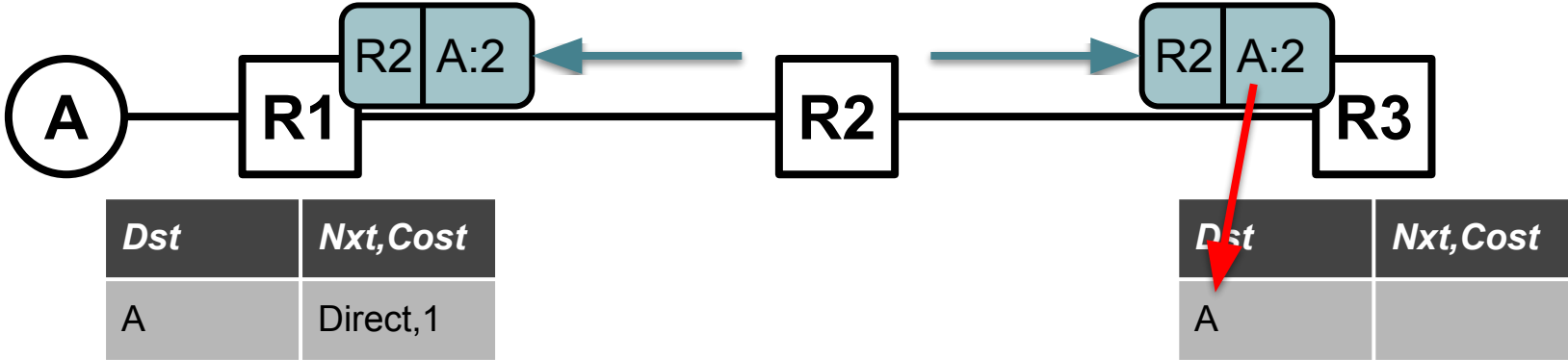


<i>Dst</i>	<i>Nxt, Cost</i>
A	Direct, 1

<i>Dst</i>	<i>Nxt, Cost</i>

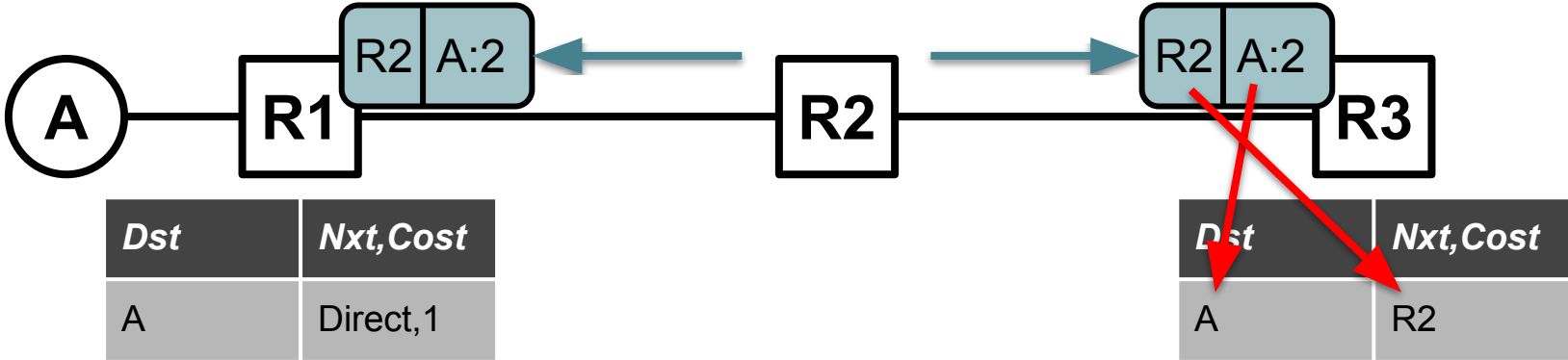
Distance-Vector

<i>Dst</i>	<i>Nxt, Cost</i>
A	R1,2



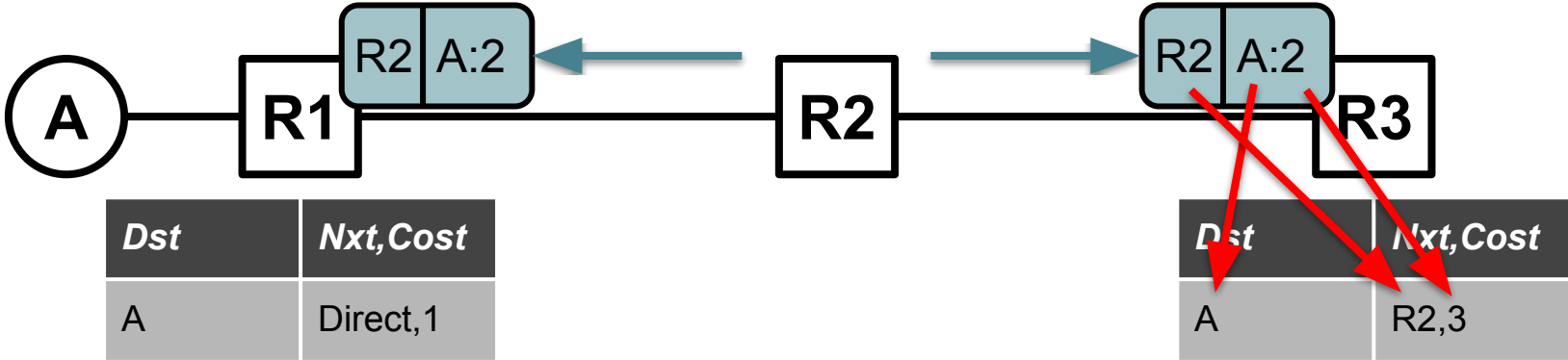
Distance-Vector

<i>Dst</i>	<i>Nxt, Cost</i>
A	R1,2

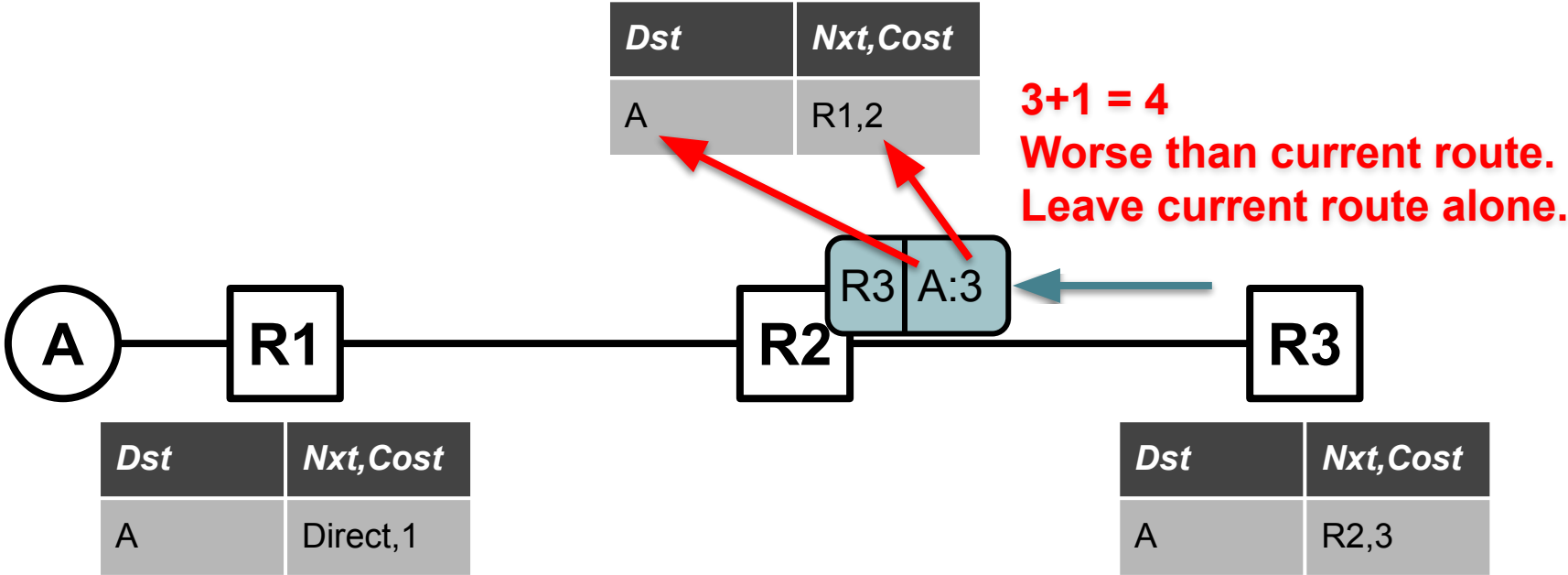


Distance-Vector

<i>Dst</i>	<i>Nxt, Cost</i>
A	R1,2

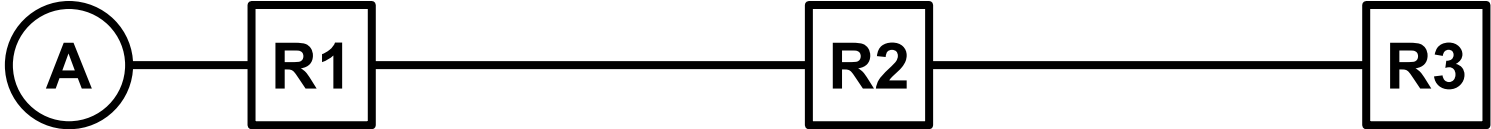


Distance-Vector



Distance-Vector

<i>Dst</i>	<i>Nxt, Cost</i>
A	R1,2



<i>Dst</i>	<i>Nxt, Cost</i>
A	Direct, 1

<i>Dst</i>	<i>Nxt, Cost</i>
A	R2,3

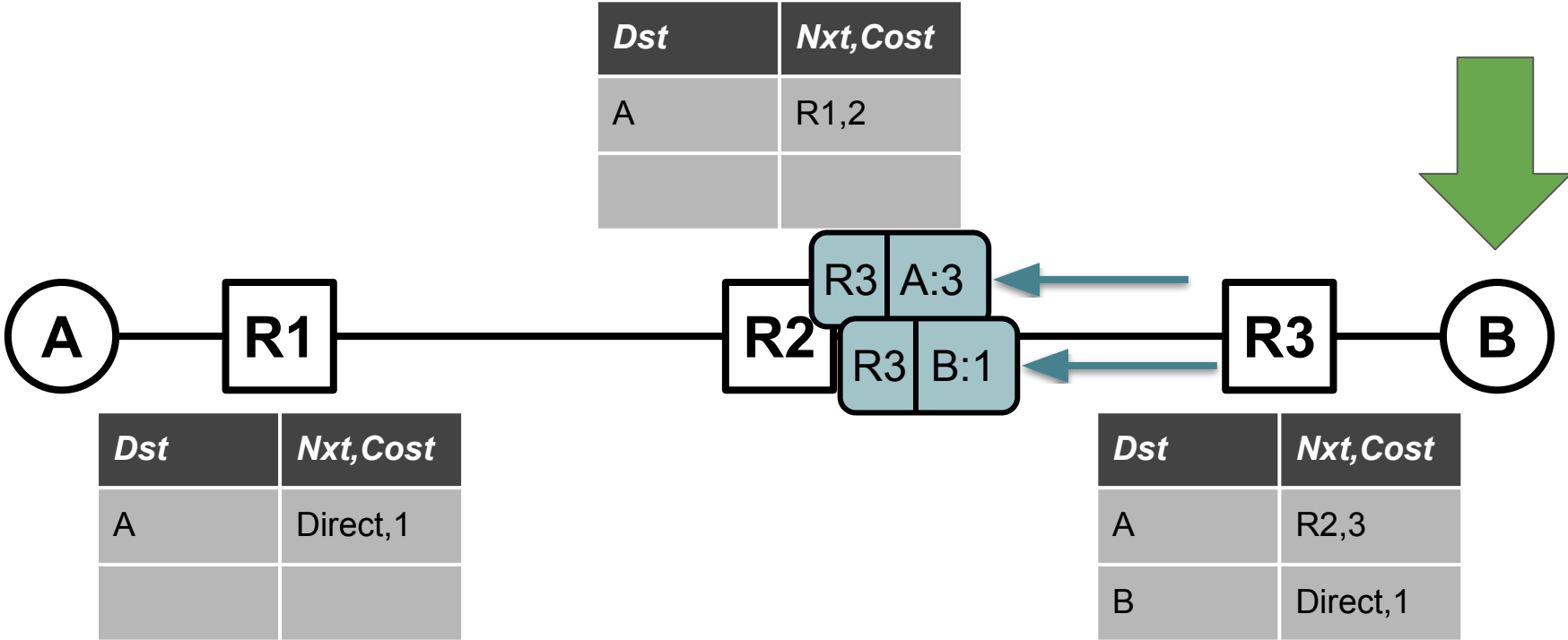
We've converged! 🎉

Questions?

Distance-Vector

What about multiple hosts?

Distance-Vector

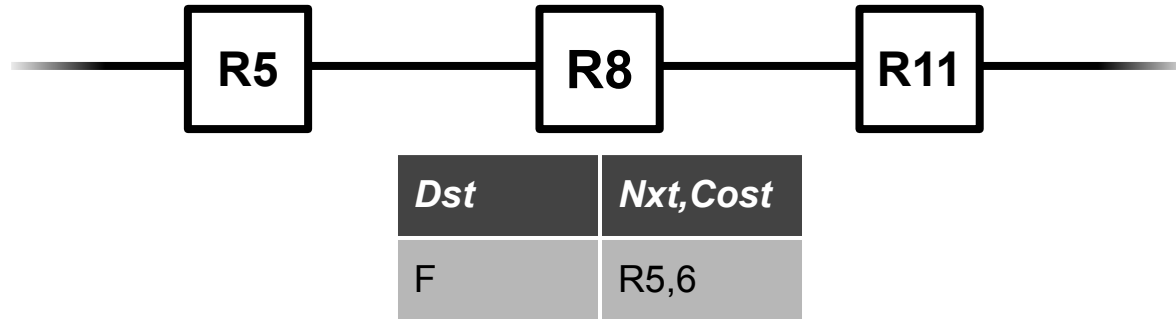


Distance-Vector

An Exception to the Rule

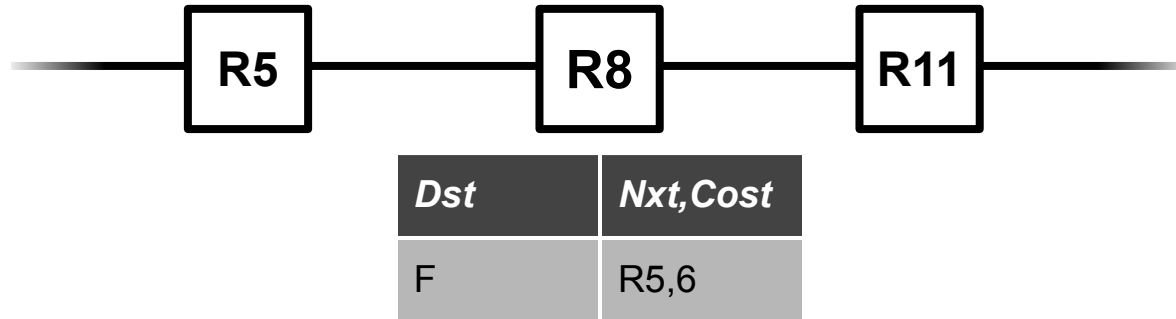
Distance-Vector: An Exception to the Rule

- Our logic for when to update a route:
 - If destination not in table -- add to table
 - If destination...



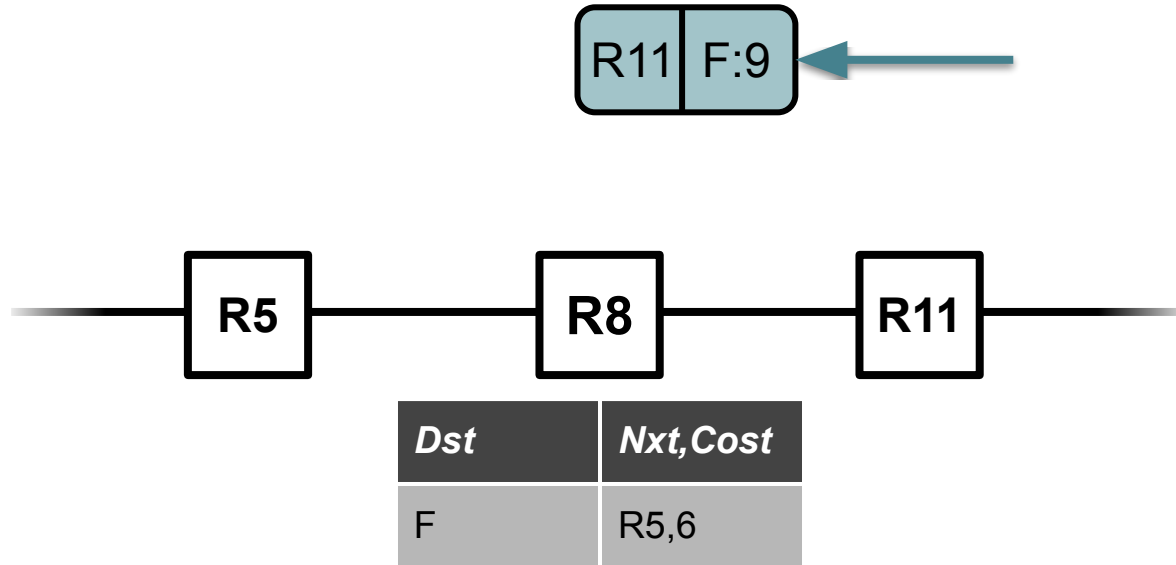
Distance-Vector: An Exception to the Rule

- Our logic for when to update a route:
 - If destination not in table -- add to table
 - If $\text{current_route_distance} > \text{advertised_distance} + \text{distance_to_neighbor}$ -- replace current



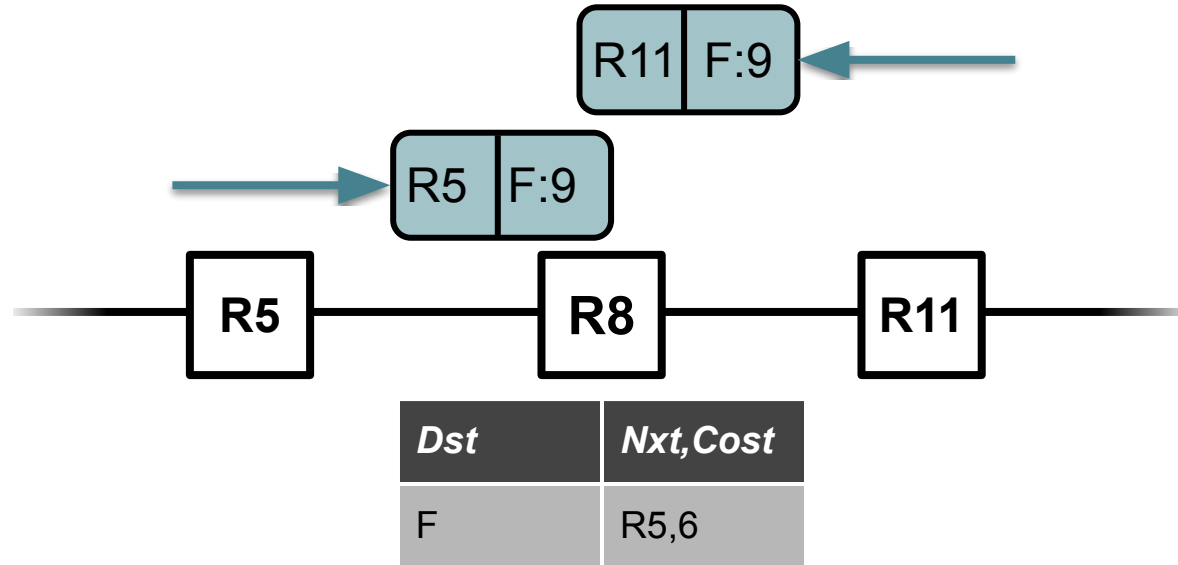
Distance-Vector: An Exception to the Rule

- Our logic for when to update a route:
 - If destination not in table -- add to table
 - If $\text{current_route_distance} > \text{advertised_distance} + \text{distance_to_neighbor}$ -- replace current



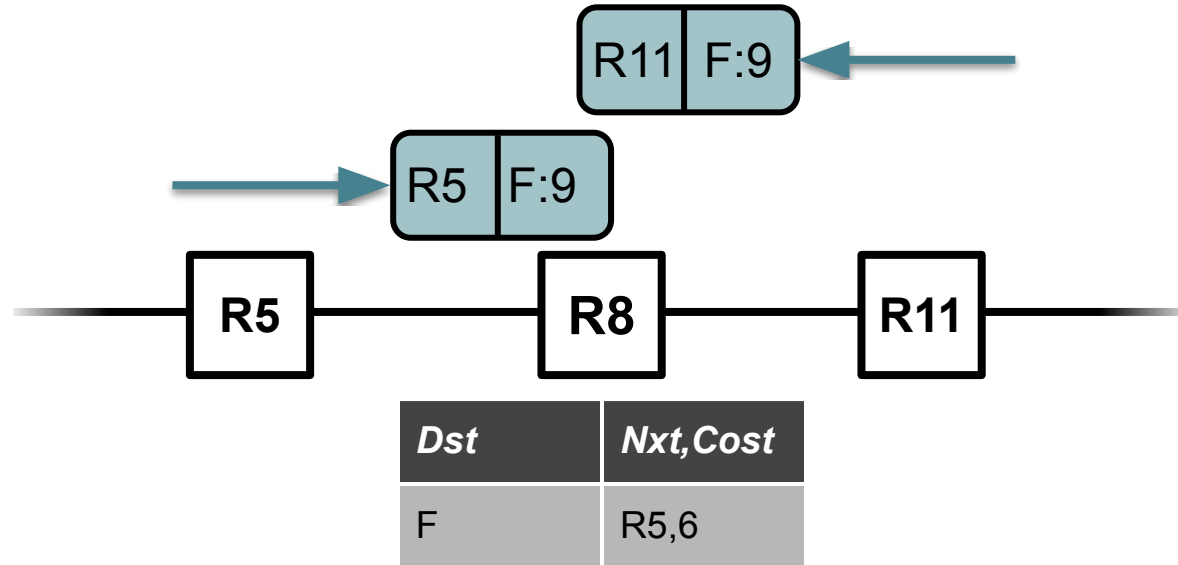
Distance-Vector: An Exception to the Rule

- Our logic for when to update a route:
 - If destination not in table -- add to table
 - If $\text{current_route_distance} > \text{advertised_distance} + \text{distance_to_neighbor}$ -- replace current



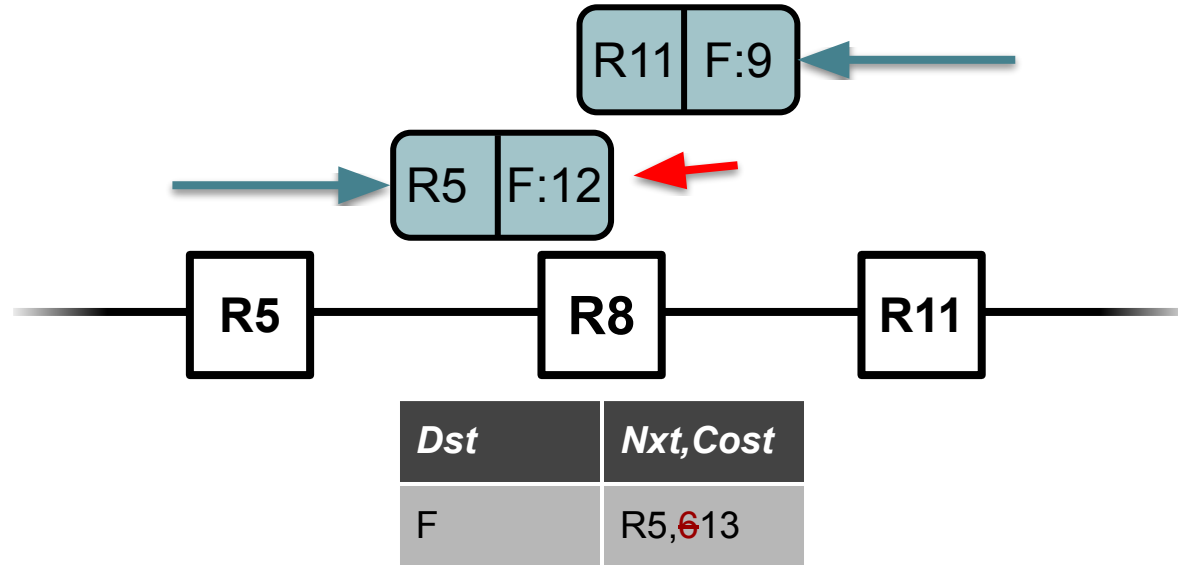
Distance-Vector: An Exception to the Rule

- Our logic for when to update a route:
 - If destination not in table -- add to table
 - If $\text{current_route_distance} > \text{advertised_distance} + \text{distance_to_neighbor}$ -- replace current
 - If advertiser is current_next_hop -- replace current



Distance-Vector: An Exception to the Rule

- Our logic for when to update a route:
 - If destination not in table -- add to table
 - If $\text{current_route_distance} > \text{advertised_distance} + \text{distance_to_neighbor}$ -- replace current
 - If advertiser is current_next_hop -- replace current

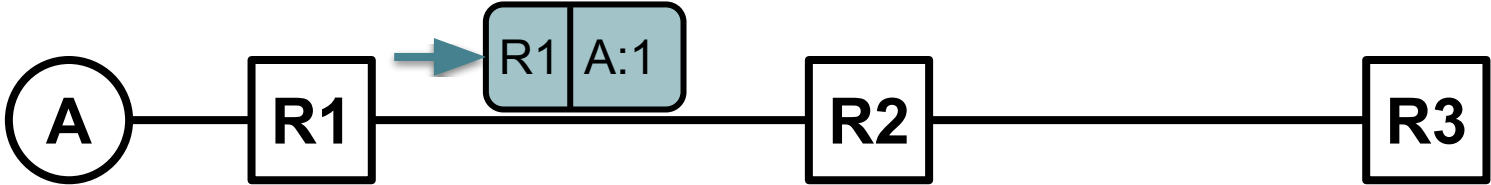


Distance-Vector

Is it reliable?

Distance-Vector: Reliability

<i>Dst</i>	<i>Nxt, Cost</i>



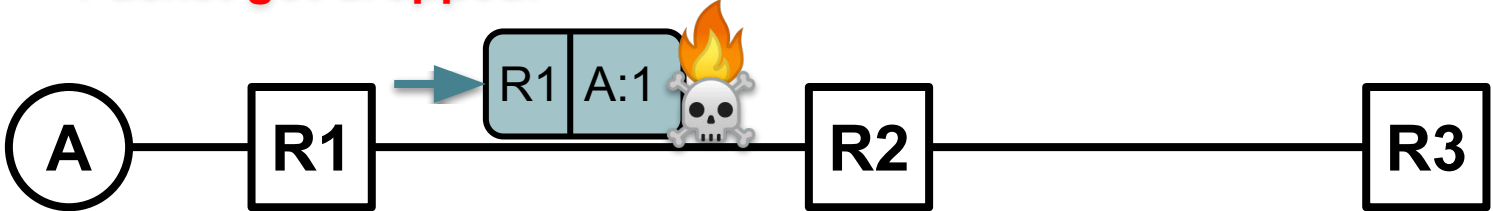
<i>Dst</i>	<i>Nxt, Cost</i>
A	Direct, 1

<i>Dst</i>	<i>Nxt, Cost</i>

Distance-Vector: Reliability

Something bad happened!
Packet got dropped!

<i>Dst</i>	<i>Nxt, Cost</i>



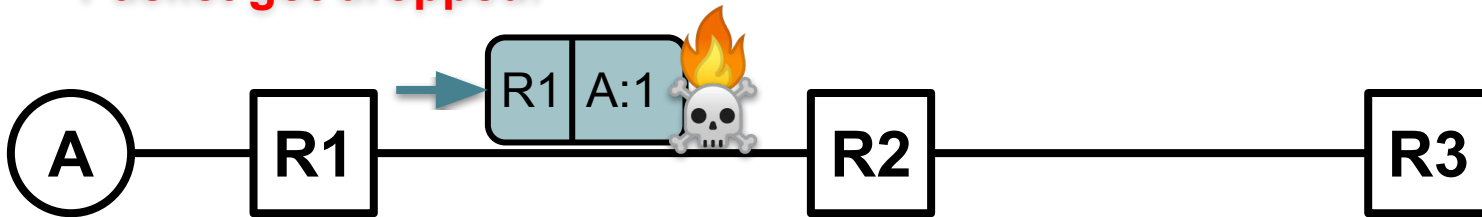
<i>Dst</i>	<i>Nxt, Cost</i>
A	Direct, 1

<i>Dst</i>	<i>Nxt, Cost</i>

Distance-Vector: Reliability

Something bad happened!
Packet got dropped!

<i>Dst</i>	<i>Nxt, Cost</i>



<i>Dst</i>	<i>Nxt, Cost</i>
A	Direct, 1

Super simple reliability

Resend advertisements every X seconds. (*X=advertisement interval*)

This should always work *eventually* (assuming link works at all).
Sending on change (*triggered updates*) acts as an *optimization*.

Distance-Vector: Reliability

Something bad happened!
Packet drops

A

Dst

A

Dst

Nxt, Cost

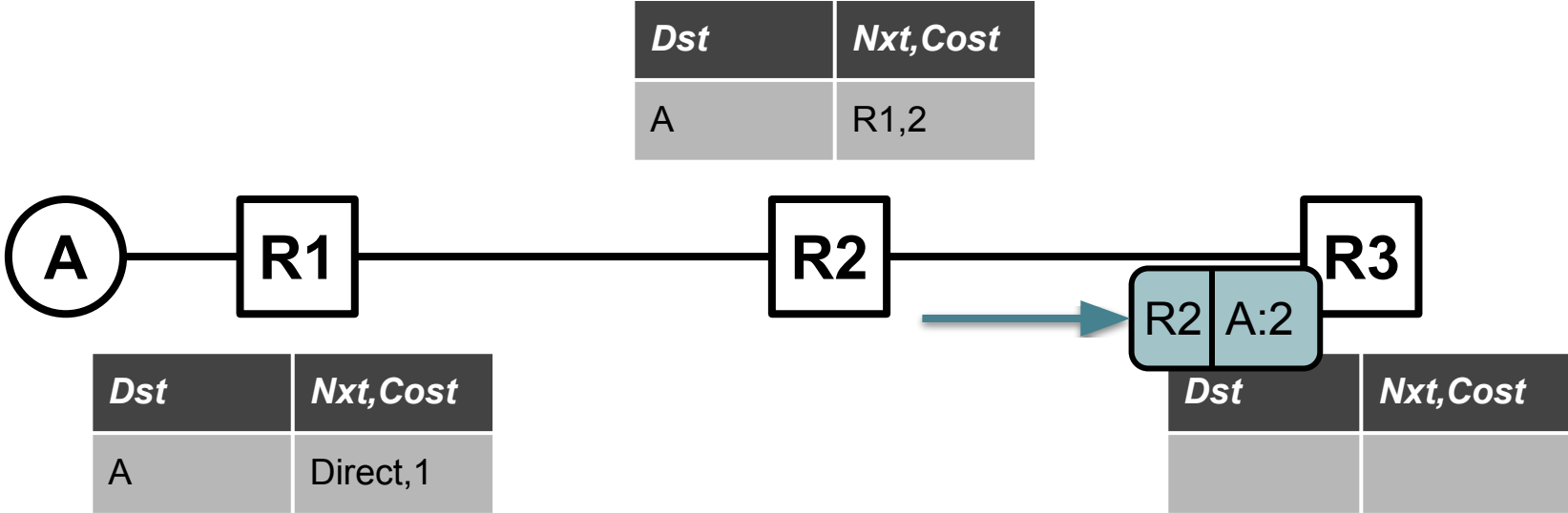
- Router timers aren't synchronized with each other!
- Between offset timers, packet drops, triggered updates, advertisements can come in *many* orders
- In following examples, I'll show things which *can* happen...
...doesn't mean they will always happen!
- I'll often ignore triggered updates because they complicate reasoning about behavior

rt, Cost

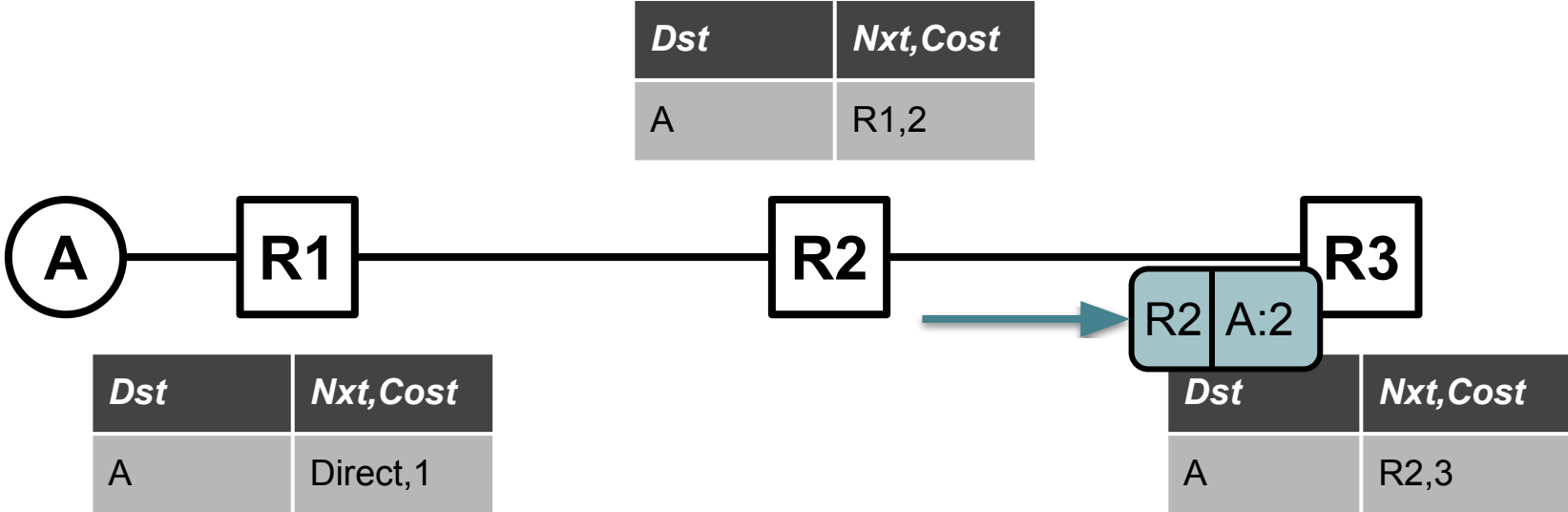
Distance-Vector

Split Horizon
&
Counting to Infinity

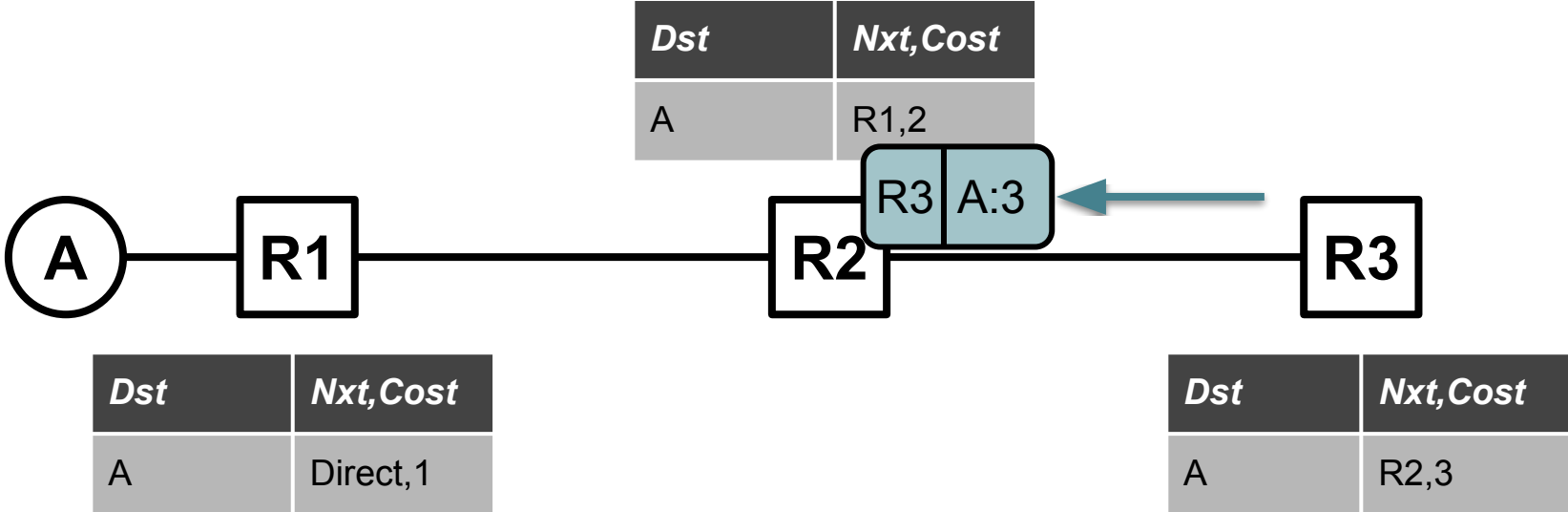
Distance-Vector: Split Horizon



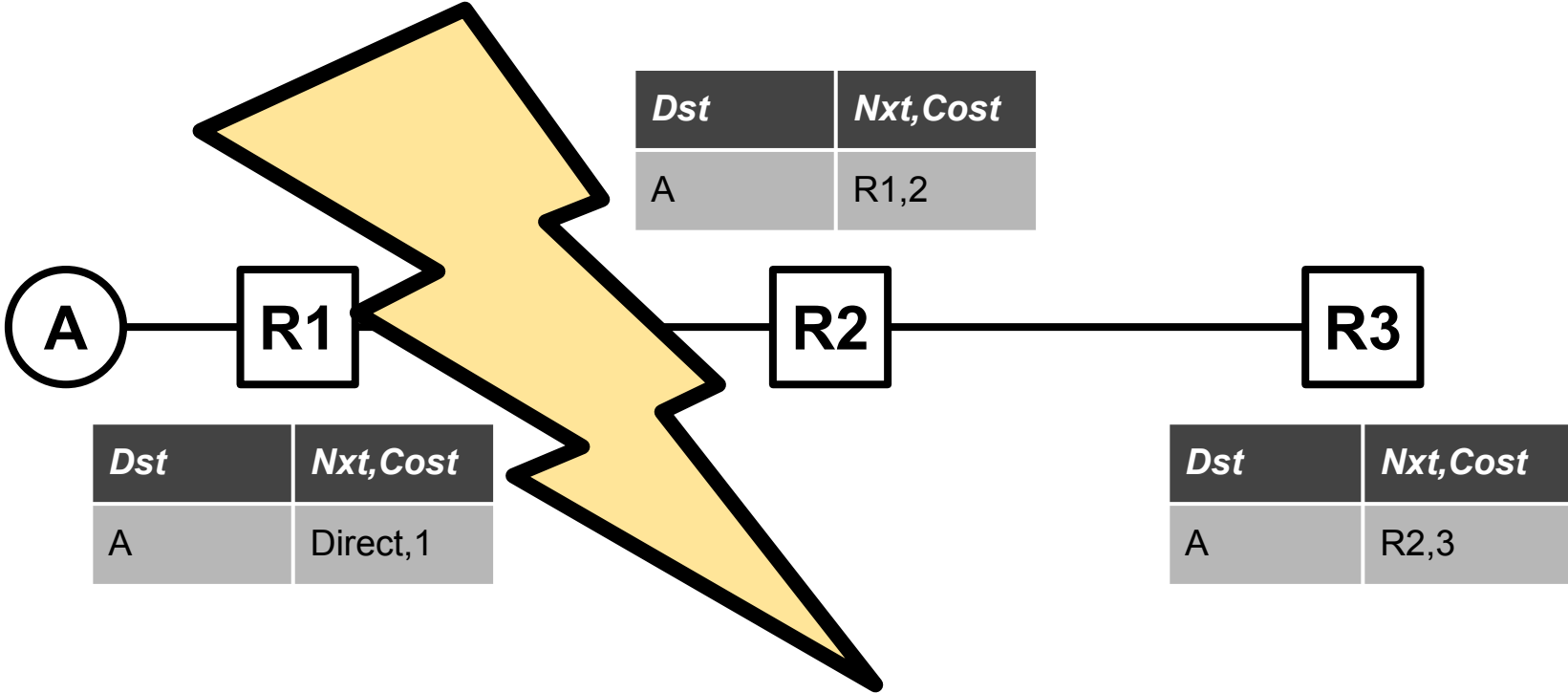
Distance-Vector: Split Horizon



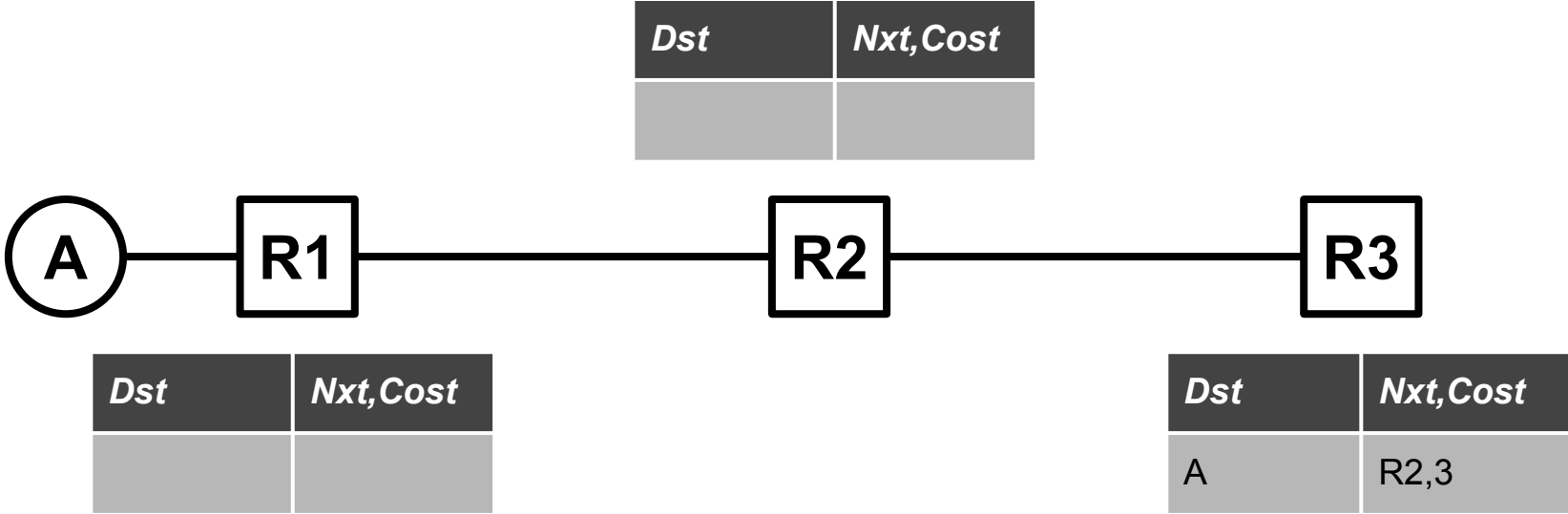
Distance-Vector: Split Horizon



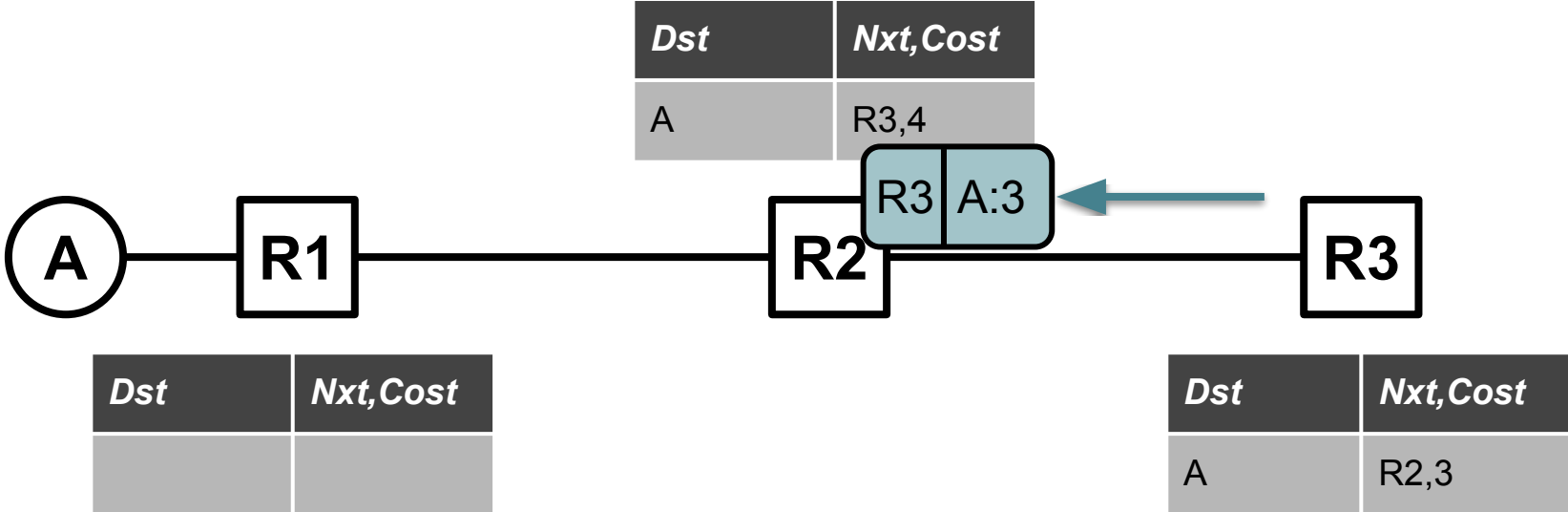
Distance-Vector: Split Horizon



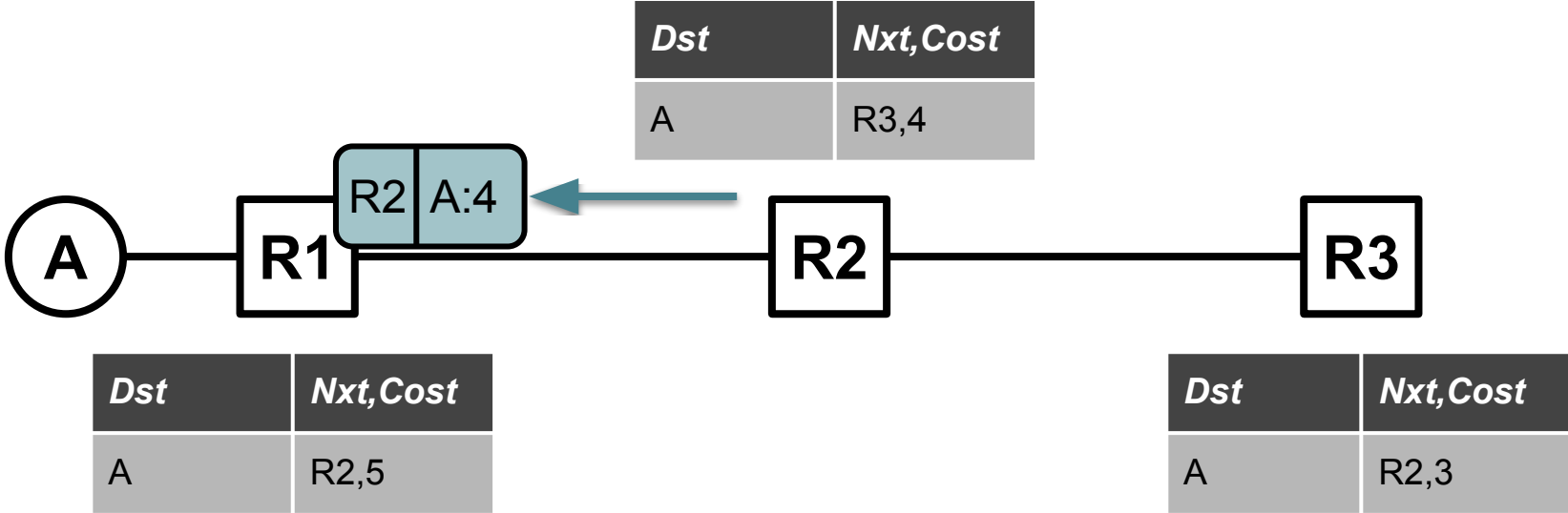
Distance-Vector: Split Horizon



Distance-Vector: Split Horizon



Distance-Vector: Split Horizon



R1 and R2 routes are pointing backwards?!

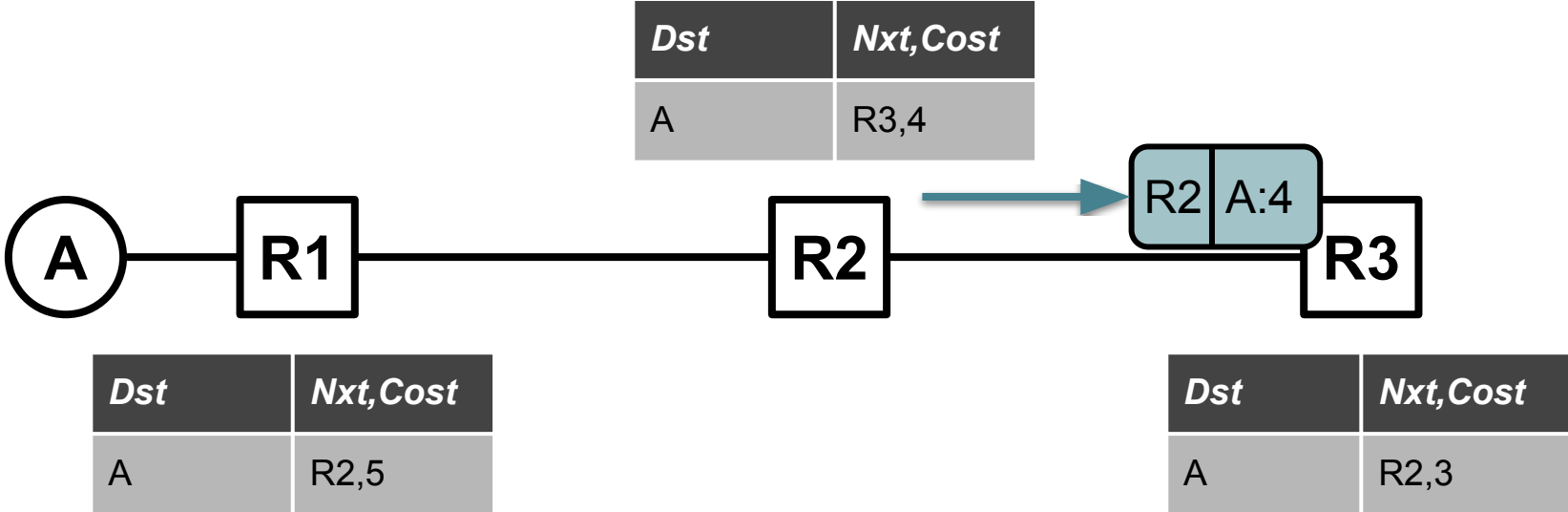
Distance-Vector: Split Horizon

- Why would you advertise a path back to the person who advertised it to you?
- Telling them about your entry going through them:
 - Doesn't tell them anything new
 - Can mislead them into thinking you have independent path

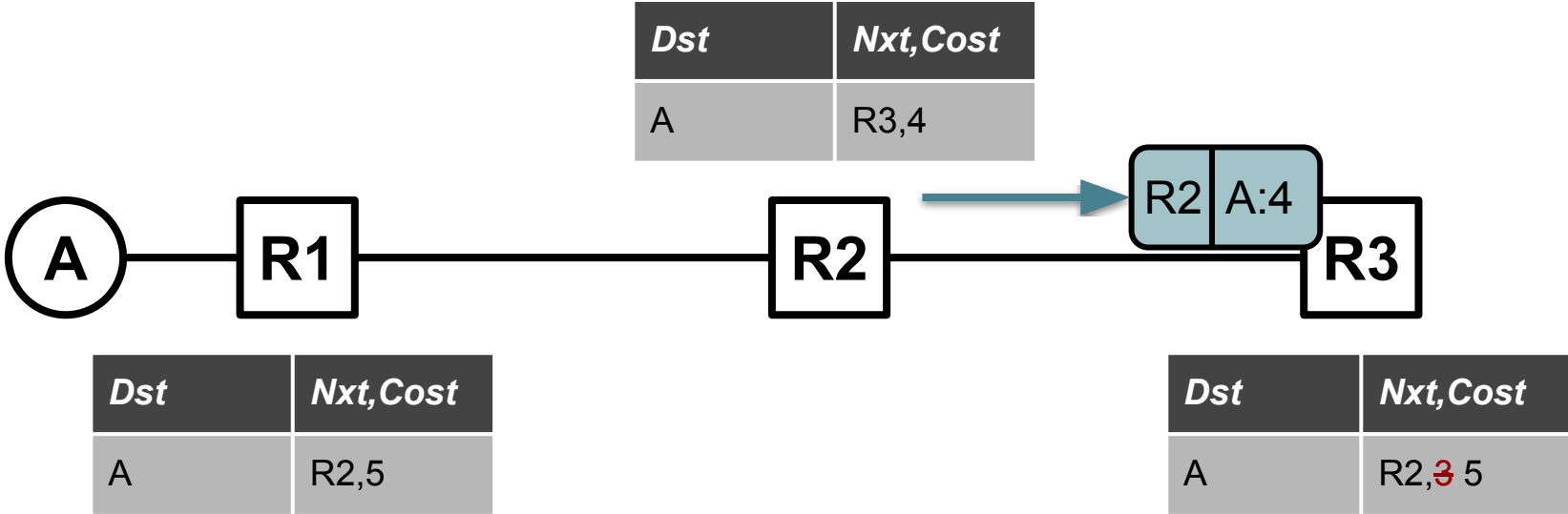
Distance-Vector: Split Horizon

- Why would you advertise a path back to the person who advertised it to you?
- Telling them about your entry going through them:
 - Doesn't tell them anything new
 - Can mislead them into thinking you have independent path
- Solution: if you are using a next-hop's path for some destination...
 - .. don't advertise that destination to them!
 - Called "Split Horizon"

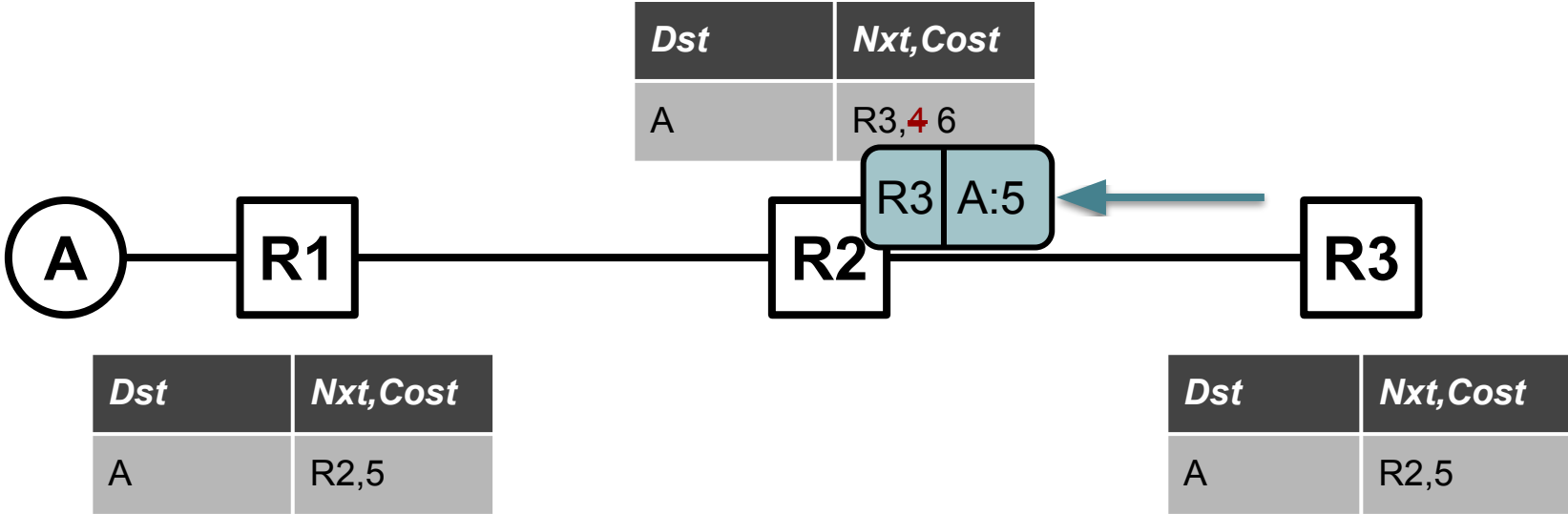
Distance-Vector: Counting to Infinity



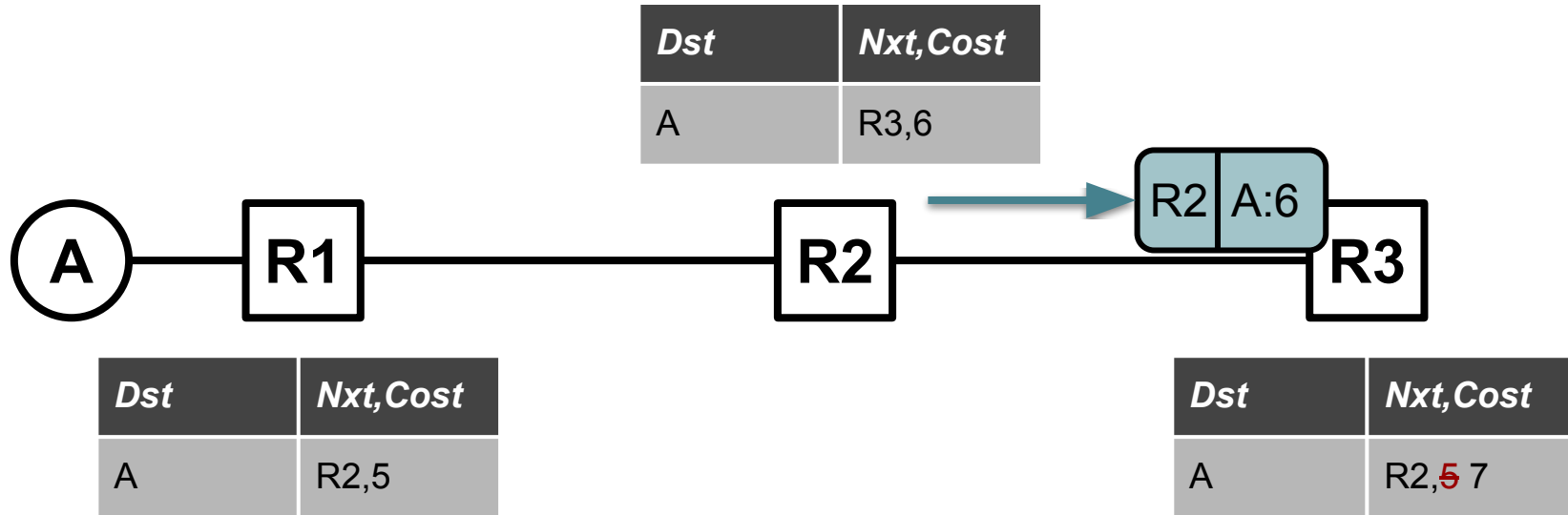
Distance-Vector: Counting to Infinity



Distance-Vector: Counting to Infinity



Distance-Vector: Counting to Infinity



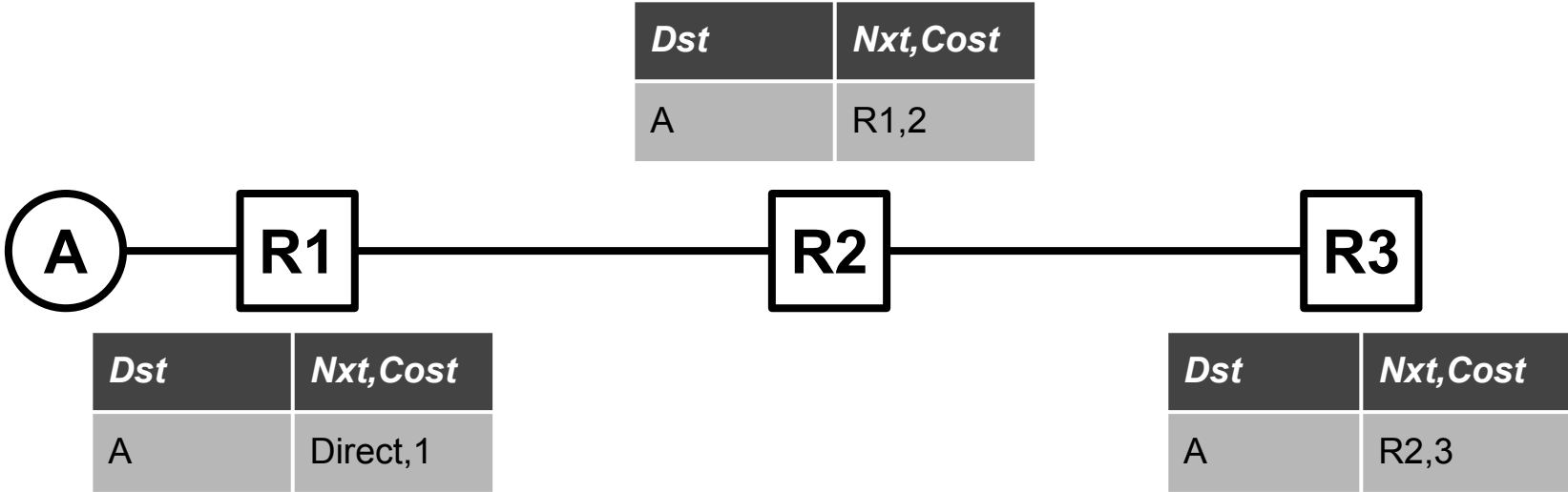
Route costs on R2/R3 count to infinity!

Solution: Pick a maximum value (e.g., 16) and stop there.

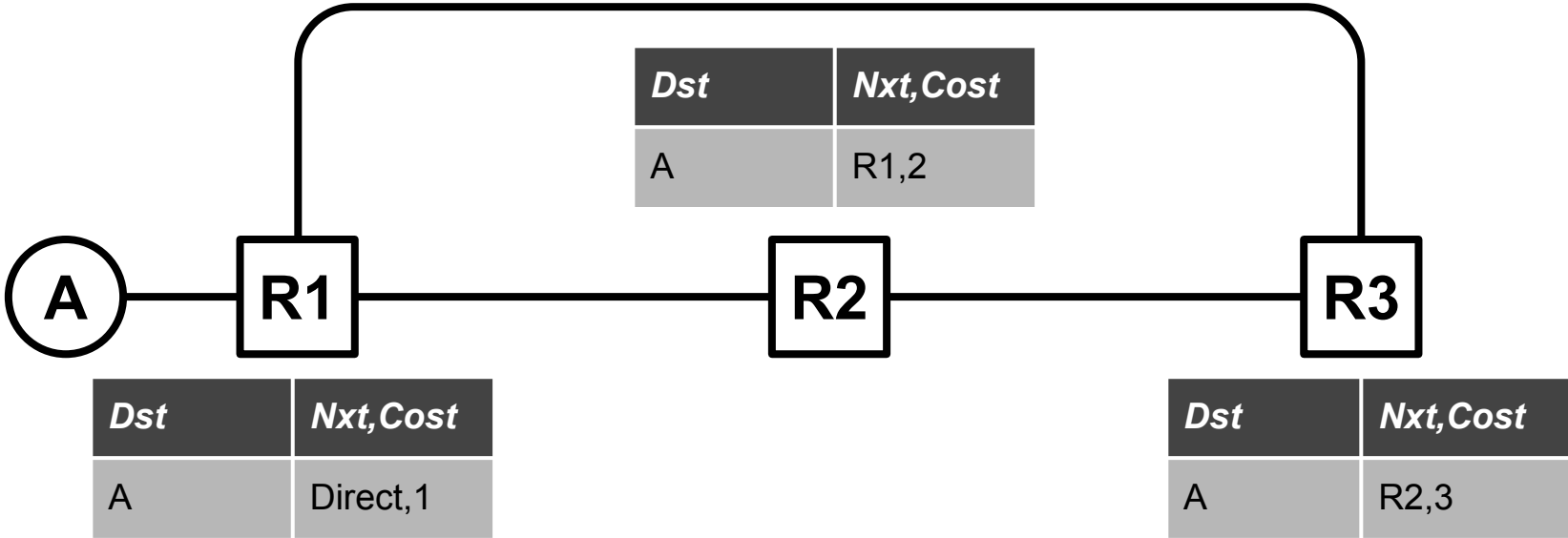
Distance-Vector

Can it handle new links?

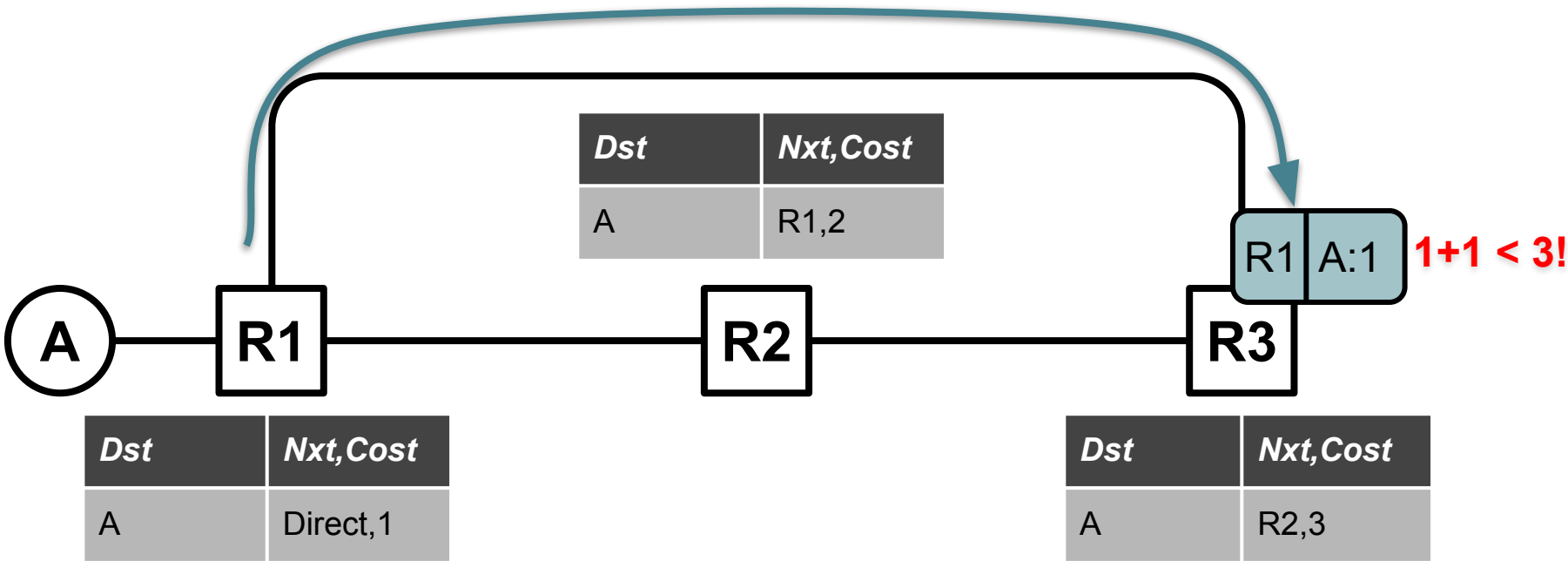
Distance-Vector: Dealing with new links



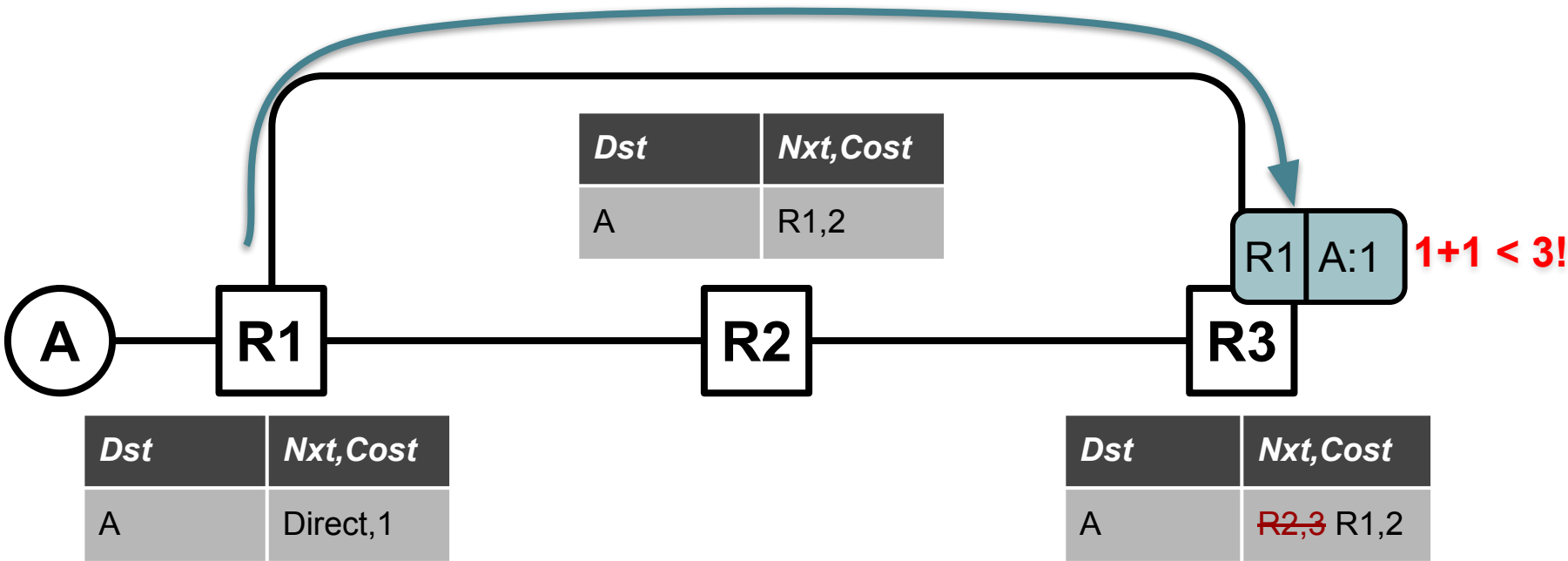
Distance-Vector: Dealing with new links



Distance-Vector: Dealing with new links



Distance-Vector: Dealing with new links



Questions?

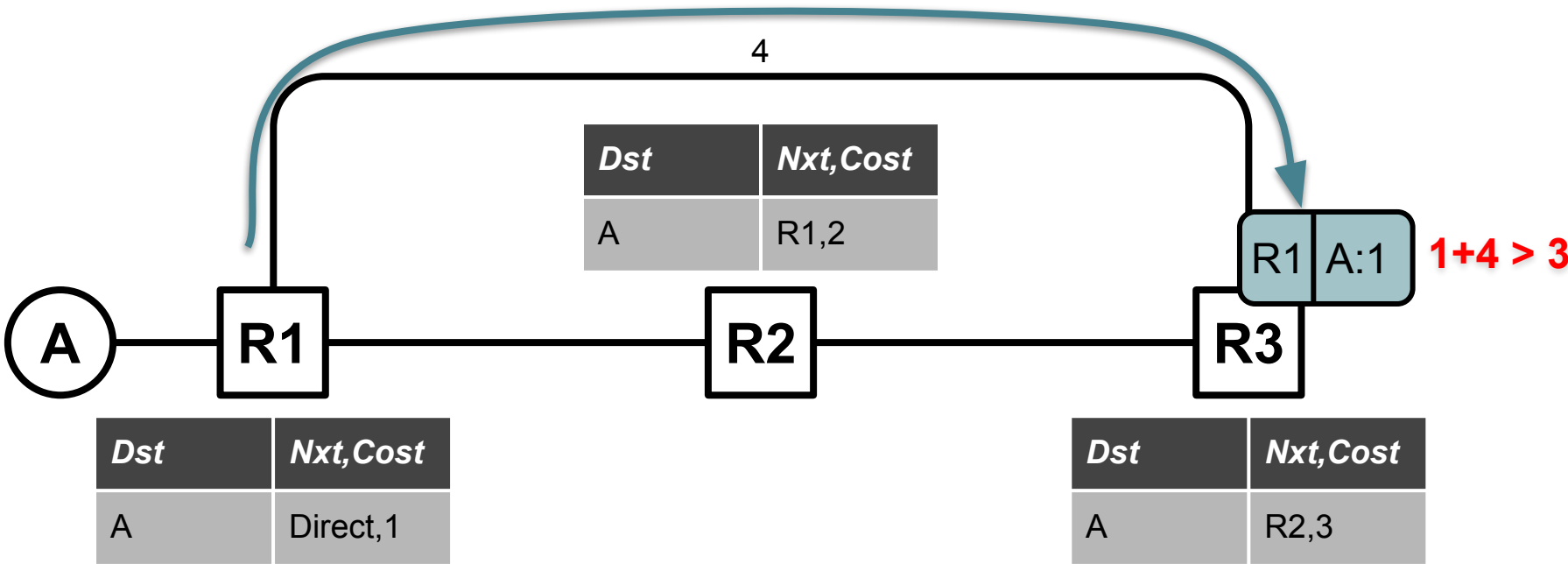
Break?

(Next up: Failed links)

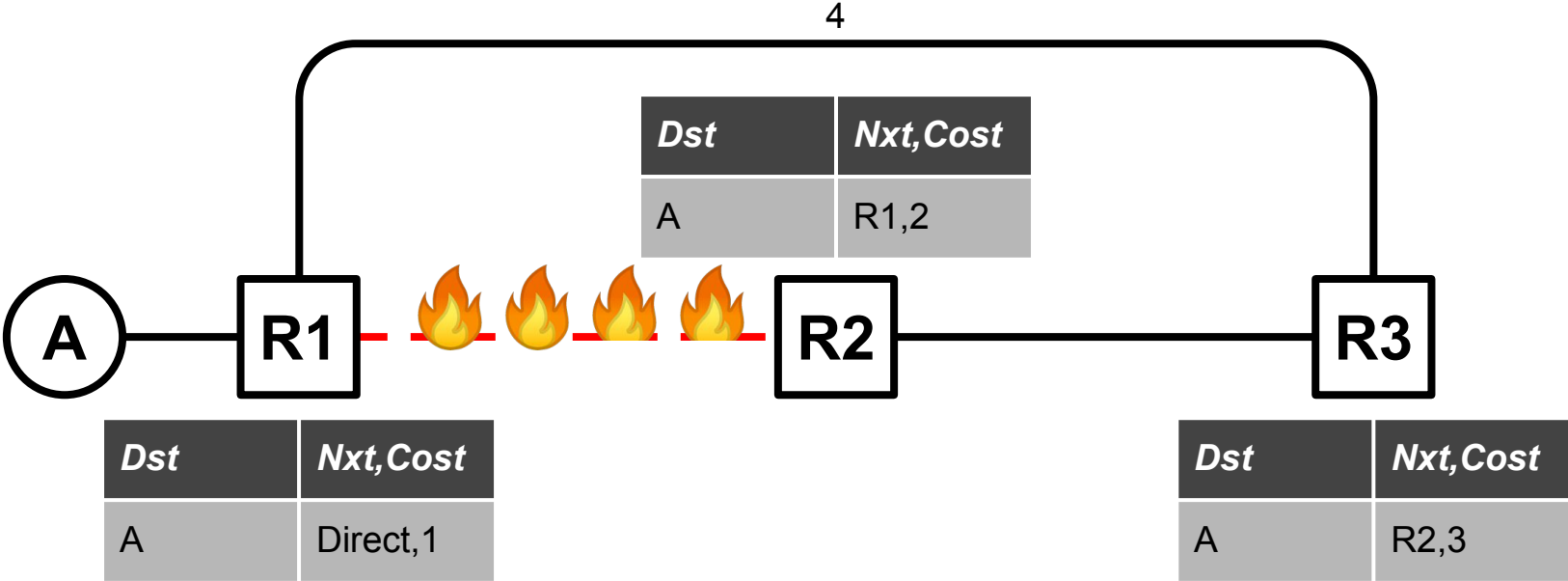
Distance-Vector

Can it handle failed links?

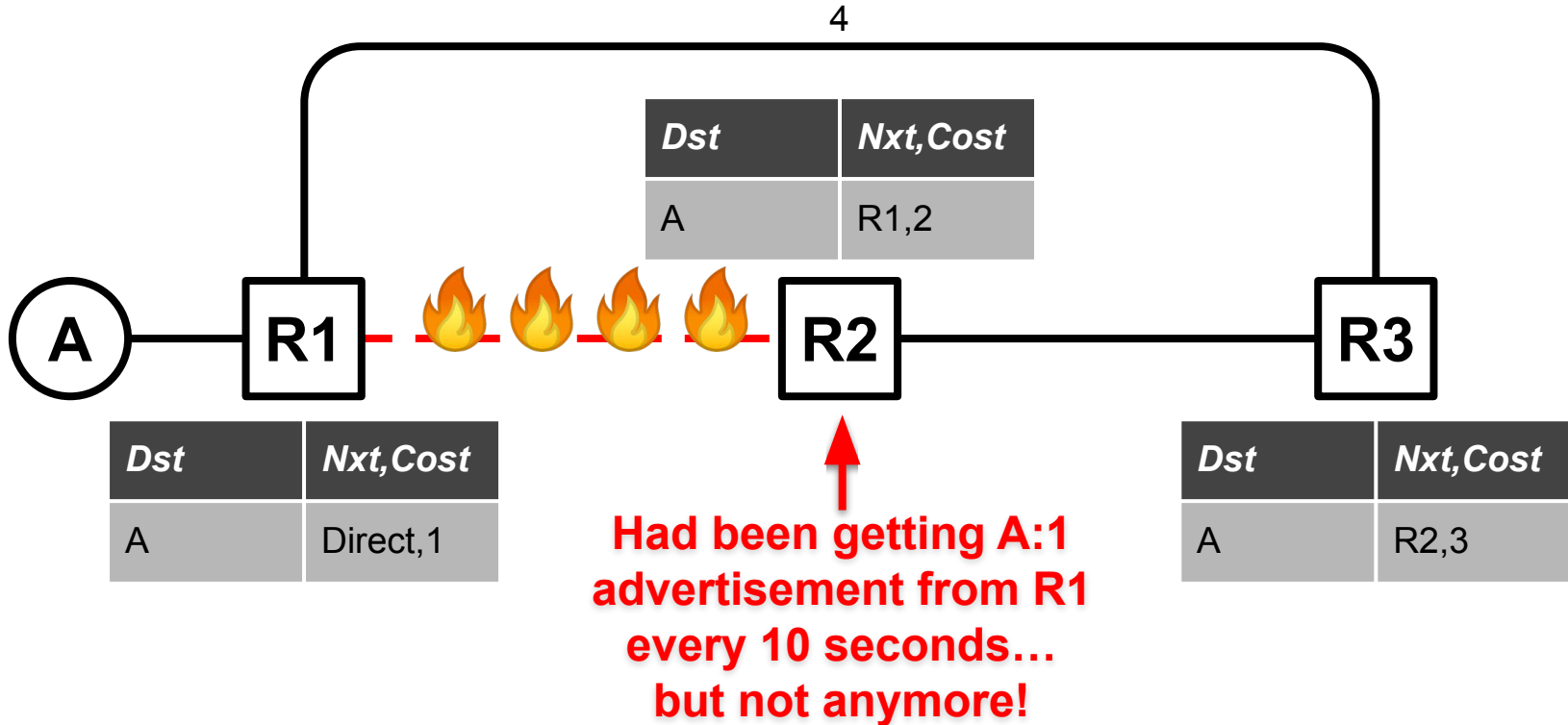
Distance-Vector: Failures



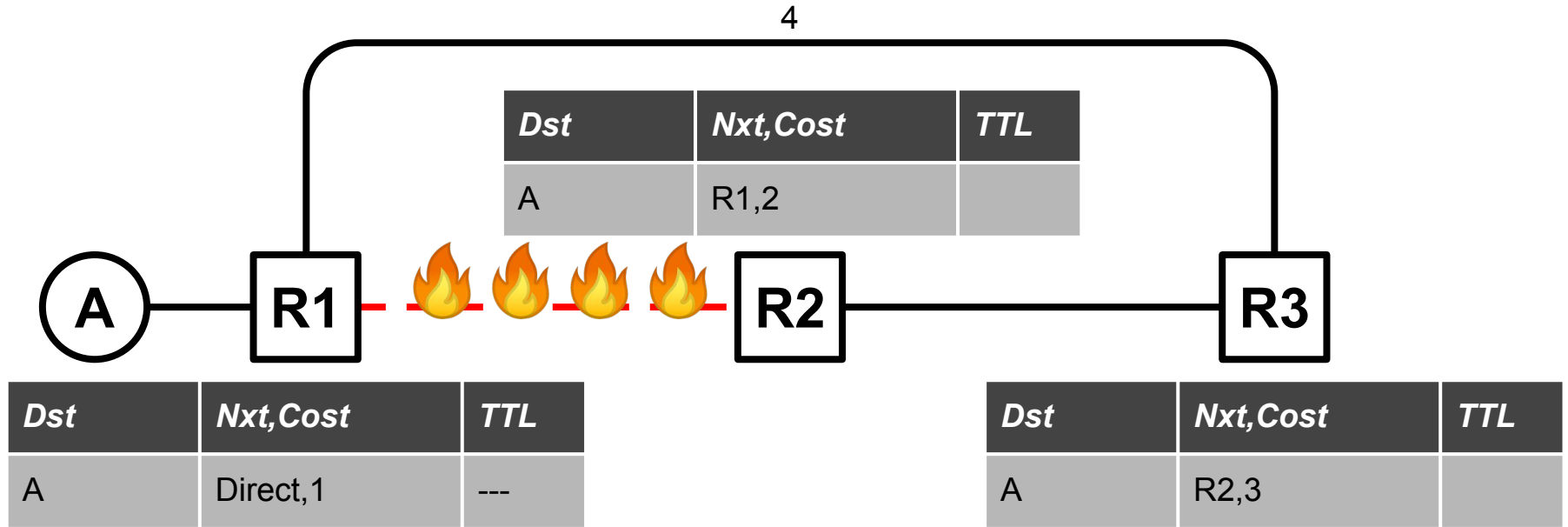
Distance-Vector: Failures



Distance-Vector: Failures



Distance-Vector: Failures

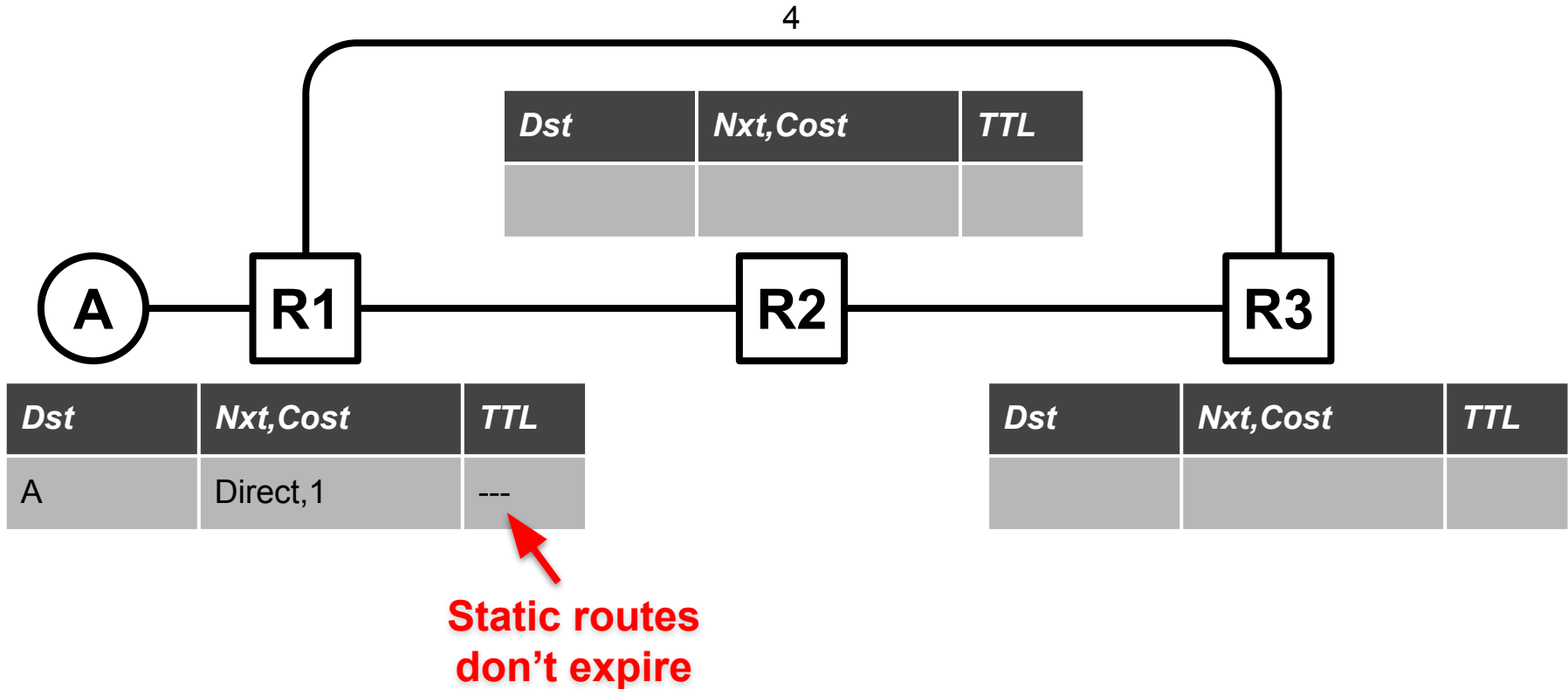


Each route only has a finite *Time To Live* (e.g., 21 seconds).

Gets “recharged” by the periodic advertisements.

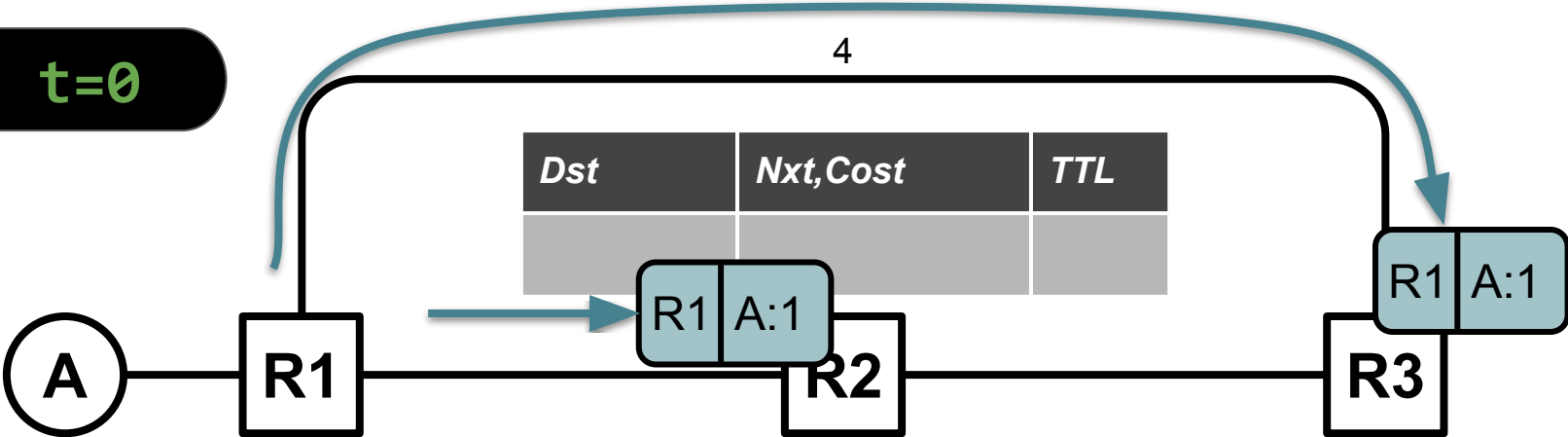
If you don’t get a periodic update (e.g., 10 seconds)... expire & remove route.

Distance-Vector: Failures



Distance-Vector: Failures

t=0

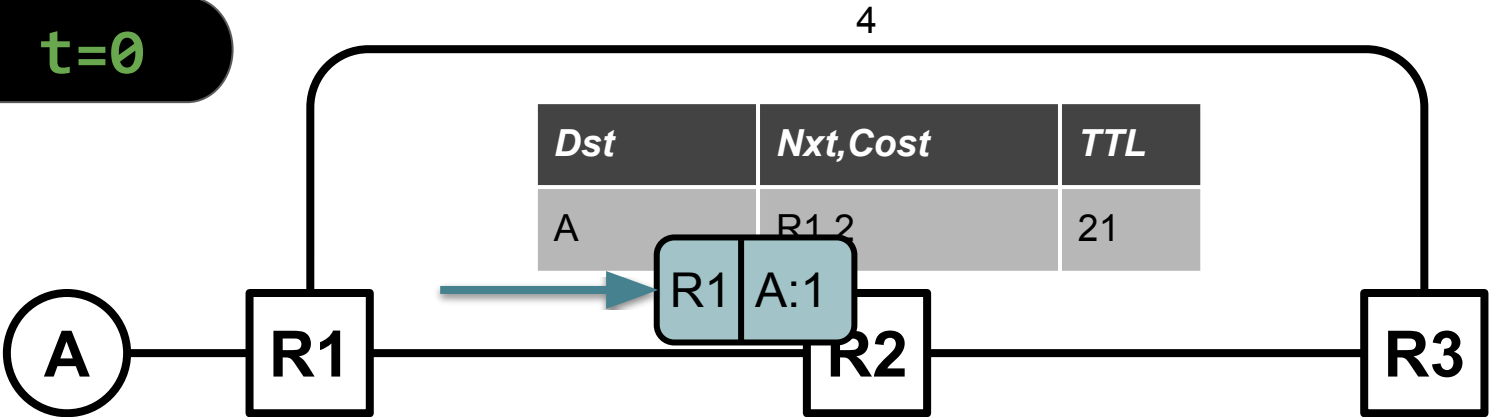


<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>

Distance-Vector: Failures

t=0



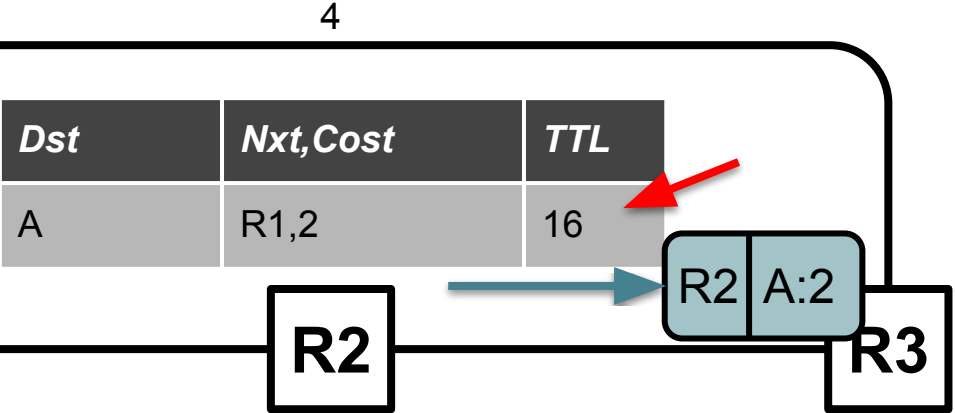
<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R1, 2	21

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R1, 5	21

Distance-Vector: Failures

t=5



<i>Dst</i>	<i>Nxt,Cost</i>	<i>TTL</i>
A	Direct,1	---

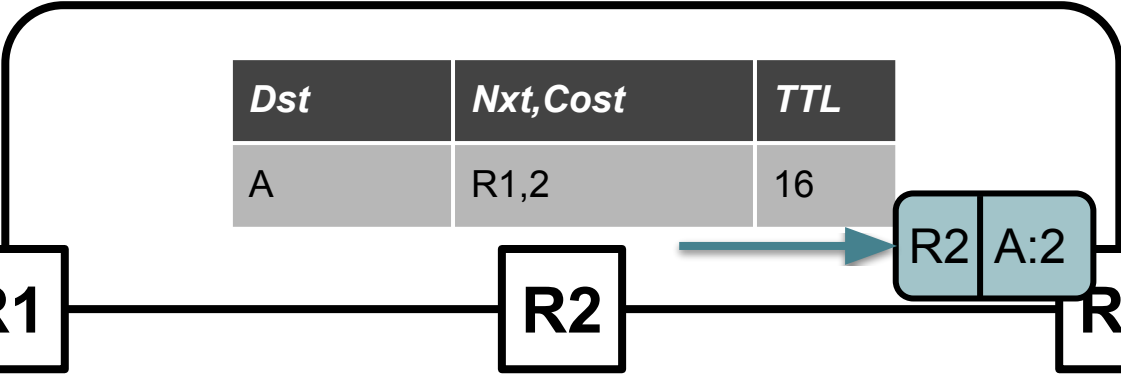
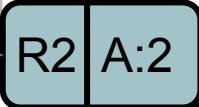
<i>Dst</i>	<i>Nxt,Cost</i>	<i>TTL</i>
A	R1,5	16

Distance-Vector: Failures

t=5

4

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R1,2	16

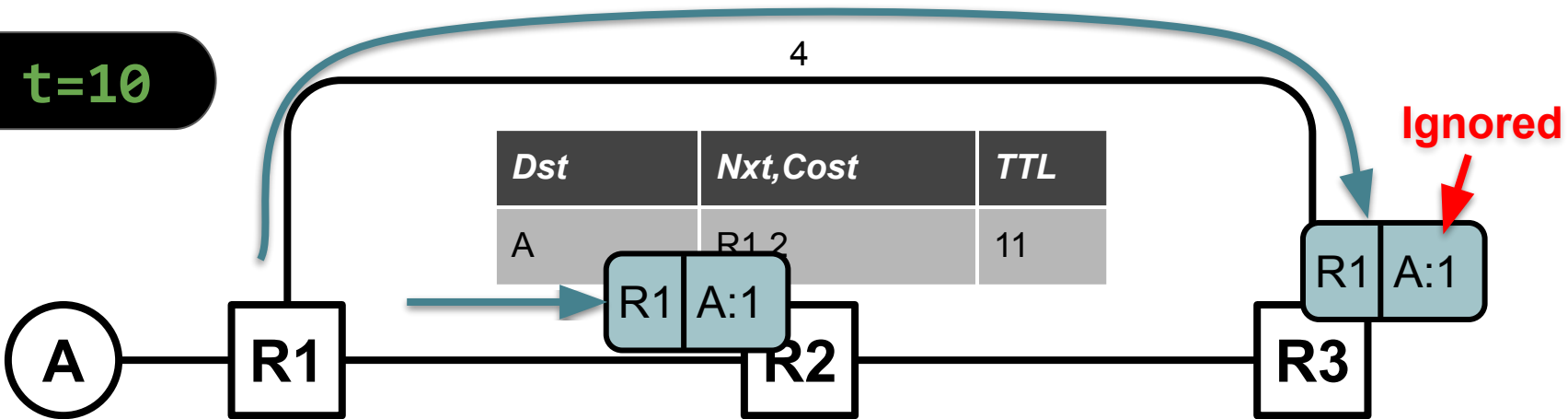


<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R1,5 R2,3	16 21

Distance-Vector: Failures

t=10



<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R1, 2	11

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

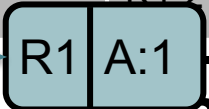
<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R2, 3	16

Distance-Vector: Failures

t=10

4

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R1, 2	11



<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R2, 3	16

Distance-Vector: Failures

t=10

4

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R1, 2	14 21

R1 A:1

A

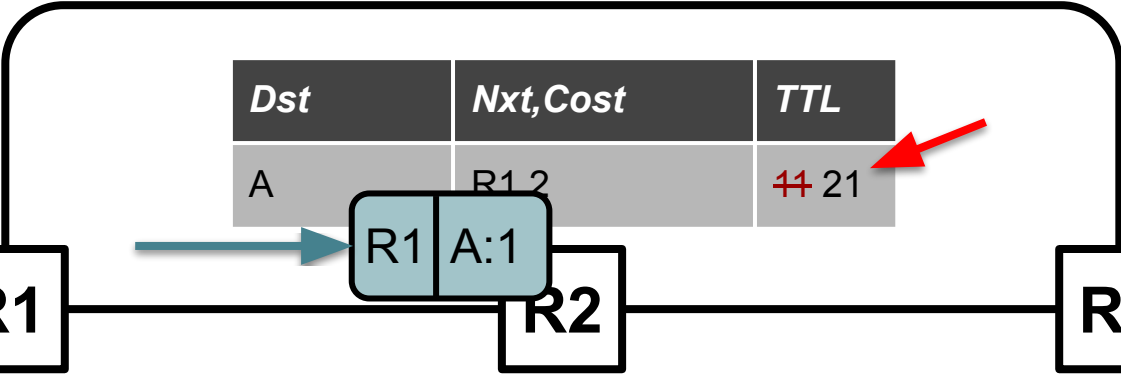
R1

R2

R3

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R2, 3	16



Distance-Vector: Failures

t=10

4

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R1,2	11 21

A

R1

R2

R3

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R2, 3	16

Distance-Vector: Failures

t=10

4

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R1,2	11 21



<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

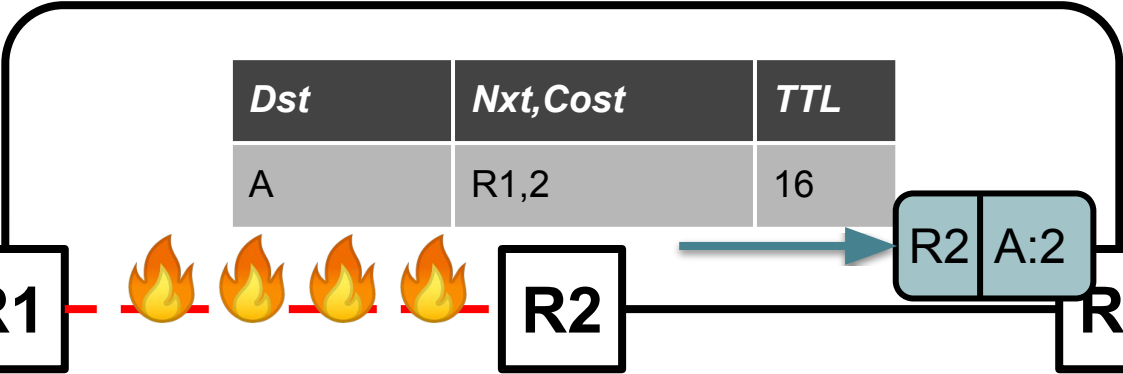
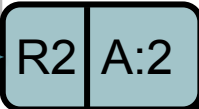
<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R2, 3	16

Distance-Vector: Failures

t=15

4

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R1,2	16

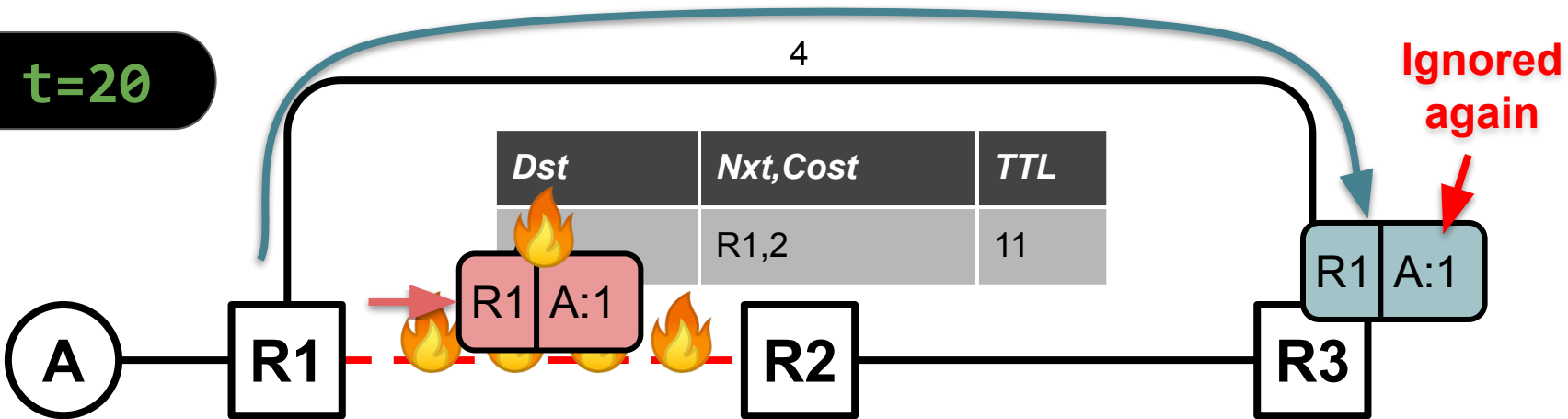


<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R2, 3	14 21

Distance-Vector: Failures

t=20



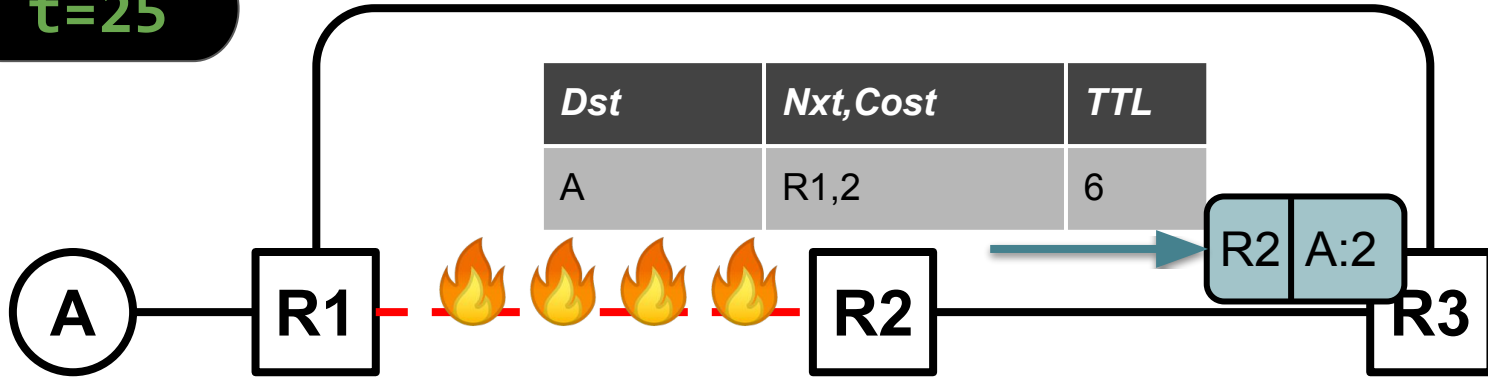
<i>Dst</i>	<i>Nxt,Cost</i>	<i>TTL</i>
A	Direct,1	---

<i>Dst</i>	<i>Nxt,Cost</i>	<i>TTL</i>
A	R2,3	16

Distance-Vector: Failures

t=25

4



<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R1,2	6

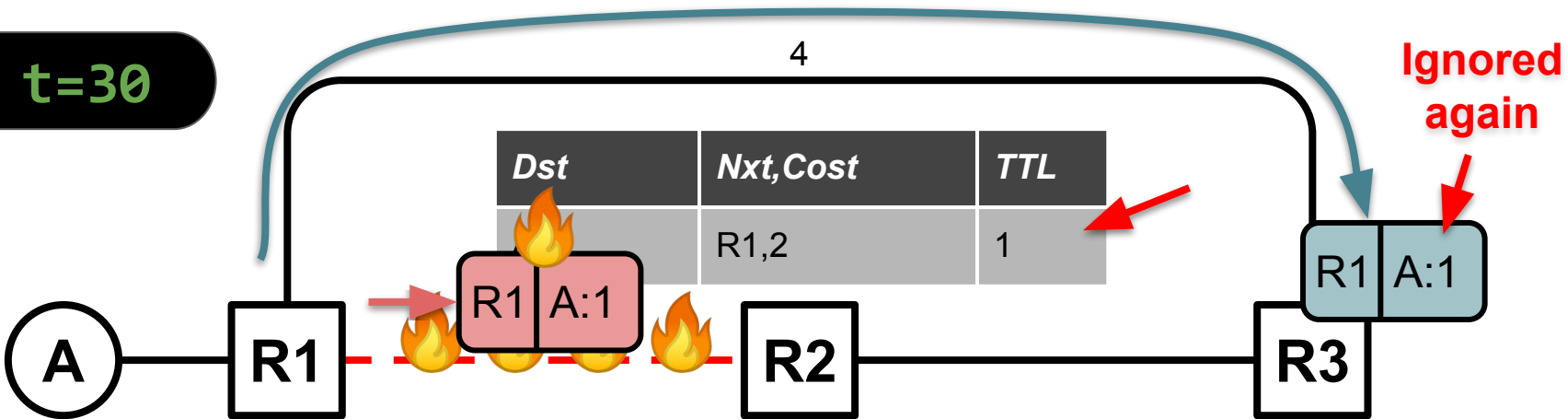
R2	A:2
----	-----

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R2, 3	4 21

Distance-Vector: Failures

t=30



<i>Dst</i>	<i>Nxt,Cost</i>	<i>TTL</i>
A	Direct,1	---

<i>Dst</i>	<i>Nxt,Cost</i>	<i>TTL</i>
A	R2,3	16

Distance-Vector: Failures

t=31

4

<i>Dst</i>	<i>Nxt,Cost</i>	<i>TTL</i>
A	R1,2	0



<i>Dst</i>	<i>Nxt,Cost</i>	<i>TTL</i>
A	Direct,1	---

<i>Dst</i>	<i>Nxt,Cost</i>	<i>TTL</i>
A	R2,3	15

Distance-Vector: Failures

t=31

4

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>

A

R1



R2

R3



<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R2, 3	15



Distance-Vector: Failures

t=46

4

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>

A

R1



R2

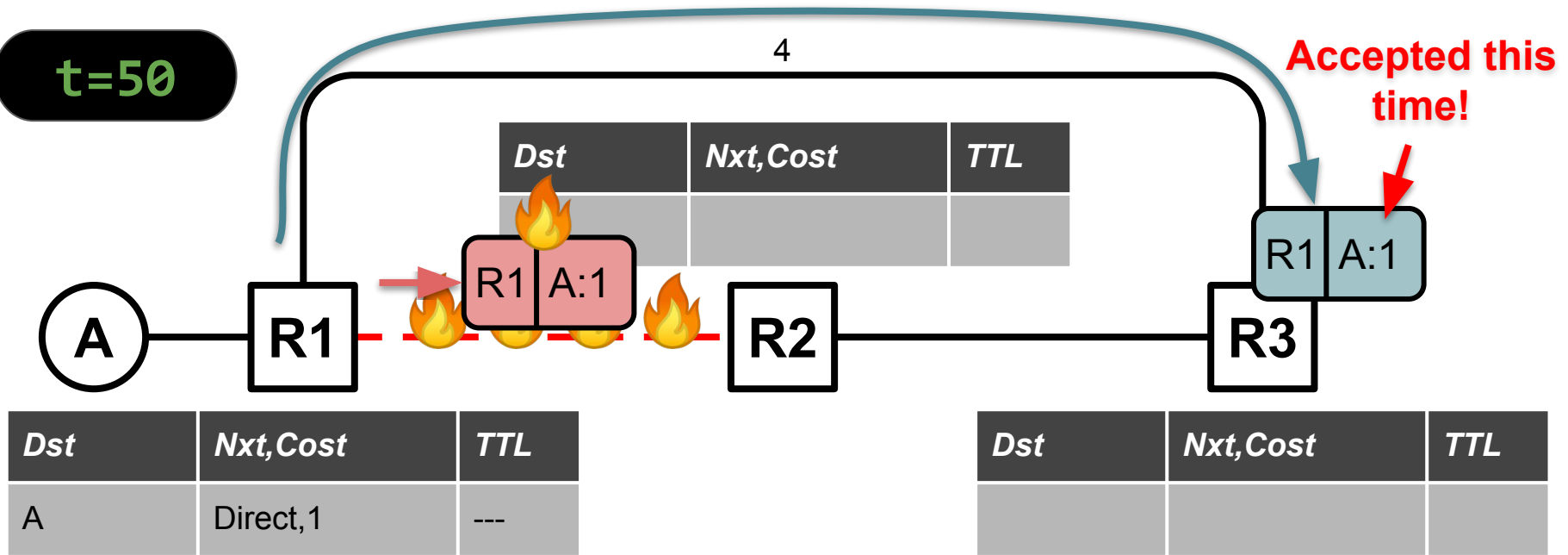
R3

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>

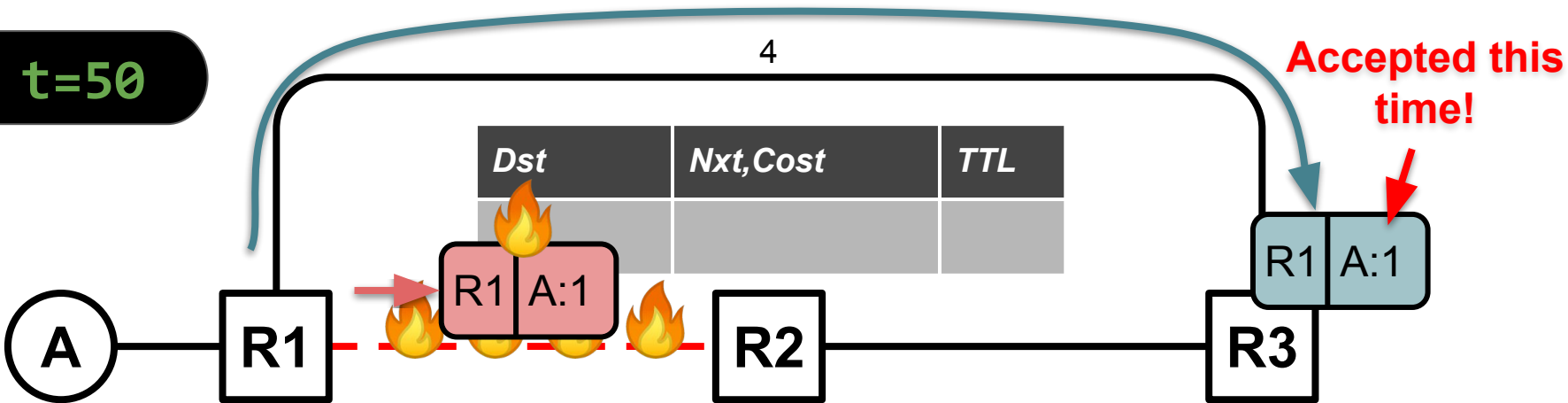
Distance-Vector: Failures

t=50



Distance-Vector: Failures

t=50

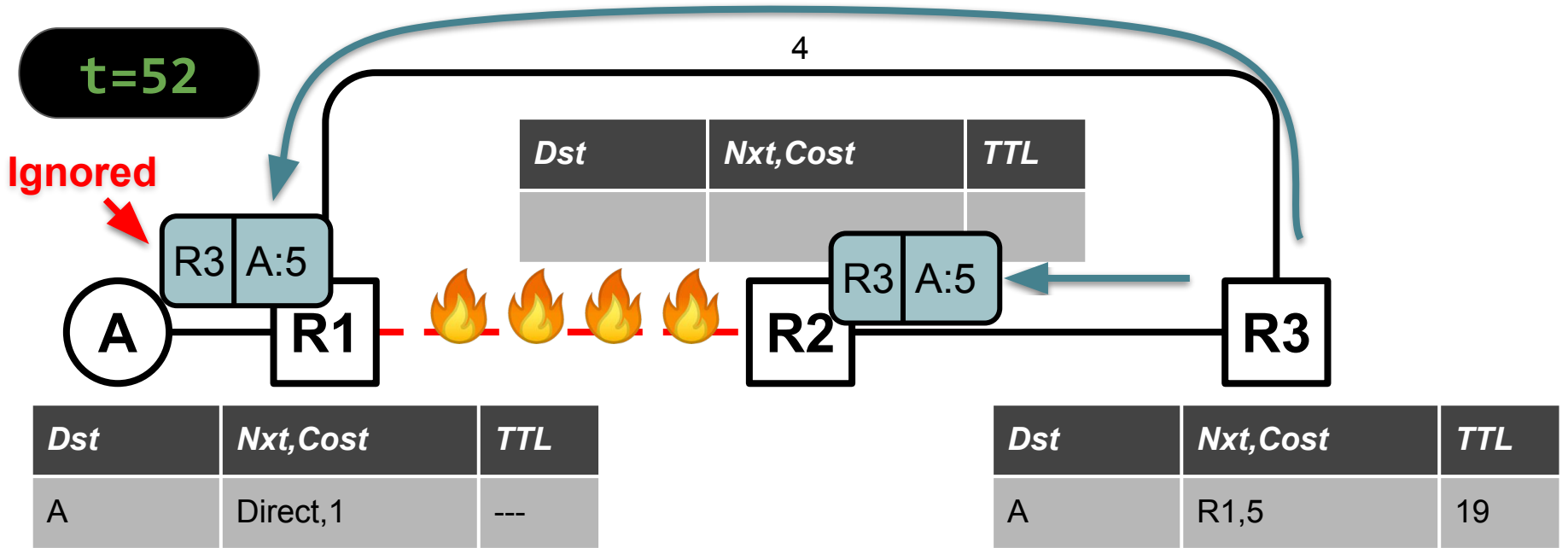


<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	Direct, 1	---

<i>Dst</i>	<i>Nxt, Cost</i>	<i>TTL</i>
A	R1, 5	21

Distance-Vector: Failures



Distance-Vector: Failures

t=52

4

<i>Dst</i>	<i>Nxt,Cost</i>	<i>TTL</i>
A	R3,6	21

A

R1



R2

R3

<i>Dst</i>	<i>Nxt,Cost</i>	<i>TTL</i>
A	Direct,1	---

<i>Dst</i>	<i>Nxt,Cost</i>	<i>TTL</i>
A	R1,5	19

Questions?

Attributions

Rick Astley Dallas.jpg, CC-BY-SA-4.0, https://commons.wikimedia.org/wiki/File:Rick_Astley_Dallas.jpg