# ECE 550
# Fundamentals of Computer Systems and Engineering

Networking

# Networking

- How do computers communicate?
  - Two computers connected by a direct wire?
    - Relatively straight forward: move bits across wire
  - Internet?
    - Many computers
    - All around the world
    - With other communications going on…
    - And un-reliable links
    - And tons of different systems, media, protocols…
- Pretty complicated, so how could we possibly manage it?

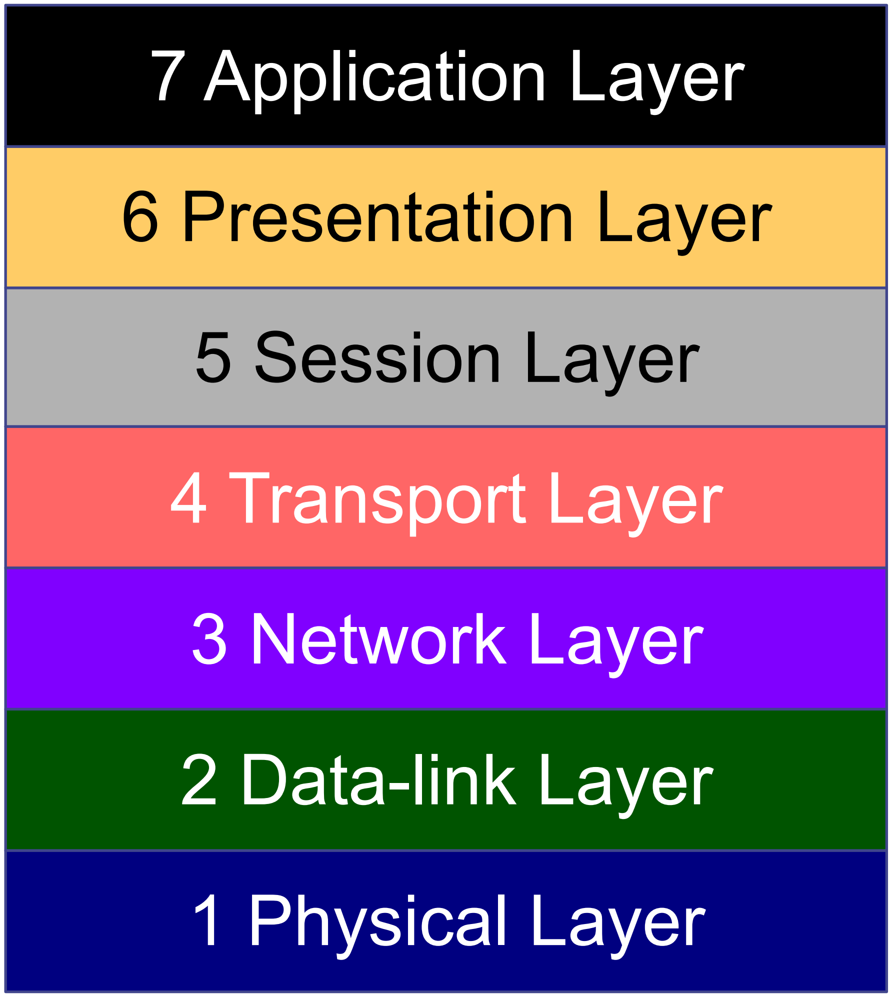
# Networking

- How do computers communicate?
  - Two computers connected by a direct wire?
    - Relatively straight forward: move bits across wire
  - Internet?
    - Many computers
    - All around the world
    - With other communications going on…
    - And un-reliable links
    - And tons of different systems, media, protocols…
- Pretty complicated, so how could we possibly manage it?

**Abstraction**

(oh right, the answer to like…everything)

# 7-layer OSI model



7 Application Layer

6 Presentation Layer

5 Session Layer

4 Transport Layer

3 Network Layer

2 Data-link Layer

1 Physical Layer

- 7 layer networking stack
- (Theoretically) can change out any layer at a time

# 1 Physical Layer

- Defines physical how physical media work
    - Pin layout
    - Voltages
    - Timing Requirements
    - Examples:
        - Cat 5 Cable (“Ethernet Cable”)
        - Wireless radio signal specifications
- Not much interesting to say here

# 2 Data-link Layer

- How to move bits across the wires in a meaningful way
  - Communication between two computers on same physical network
  - May include some error checking
- Example: Ethernet
  - Data transmitted in frames
  - Frame has:
    - **Pre-amble**: used to detect collisions
    - **Header**: source and destination MAC address
    - **Payload**: actual data
    - **CRC check**: detect corrupt data
  - Carrier Sense, Multiple Access, Collision Detect (**CSMACD**)
    - Carrier Sense:  listen for if anyone else transmitting
    - Multiple Access: can wire up many computers to it
    - Collision Detect: two transmissions at once? Detect and retry

# CSMACD

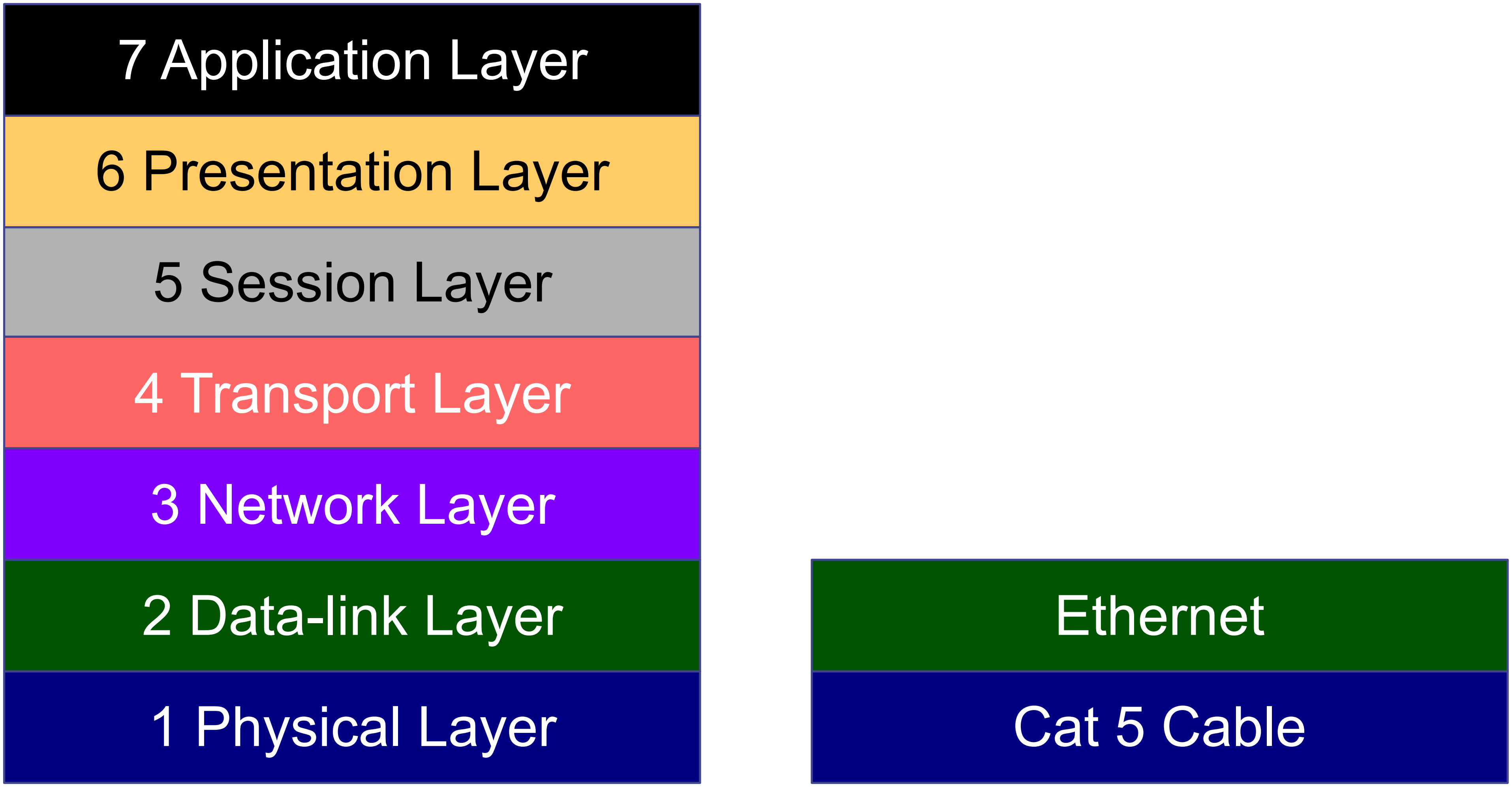
- Ethernet uses CSMACD for multiple systems on a network
    - Other options, but we won't go into them
    - Detection of collisions?
        - Pre-amble is fixed pattern
        - Network card senses medium while transmitting
        - Mismatch with expected?  Collision
    - Collision happens?
        - Exponential backoff
        - Pick random number of time units
        - Retry
        - Fail again?  Pick random number from 2x as big a range
- Analogy: crowded dinner party
    - Try to talk. Someone else talking?  Wait.  Try again. Fail again? Wait longer

# Abstraction: Joys and Limitations

- Joys of abstraction:
  - Can build an Ethernet card without any info about higher layers
  - Will work with all of them

- Limitations:
  - 7-layer model's abstraction not perfect
  - Ethernet protocol imposes max limit on cable length
    - E.g., layer 2 constrains layer 1
    - This arises from the need to detect collisions before finishing sending

# 7-layer OSI model



7 Application Layer

6 Presentation Layer

5 Session Layer

4 Transport Layer

3 Network Layer

2 Data-link Layer

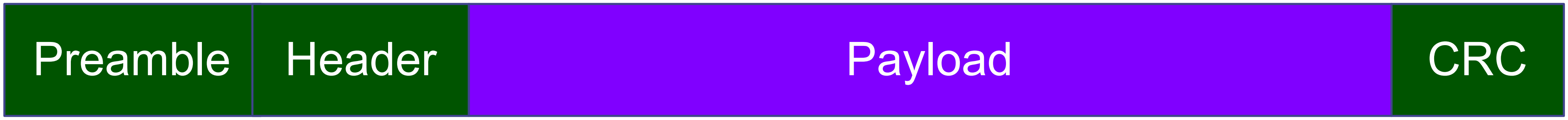
1 Physical Layer

Ethernet

Cat 5 Cable

- Reminder where we are so far

# Our messages so far



Preamble Header Payload CRC

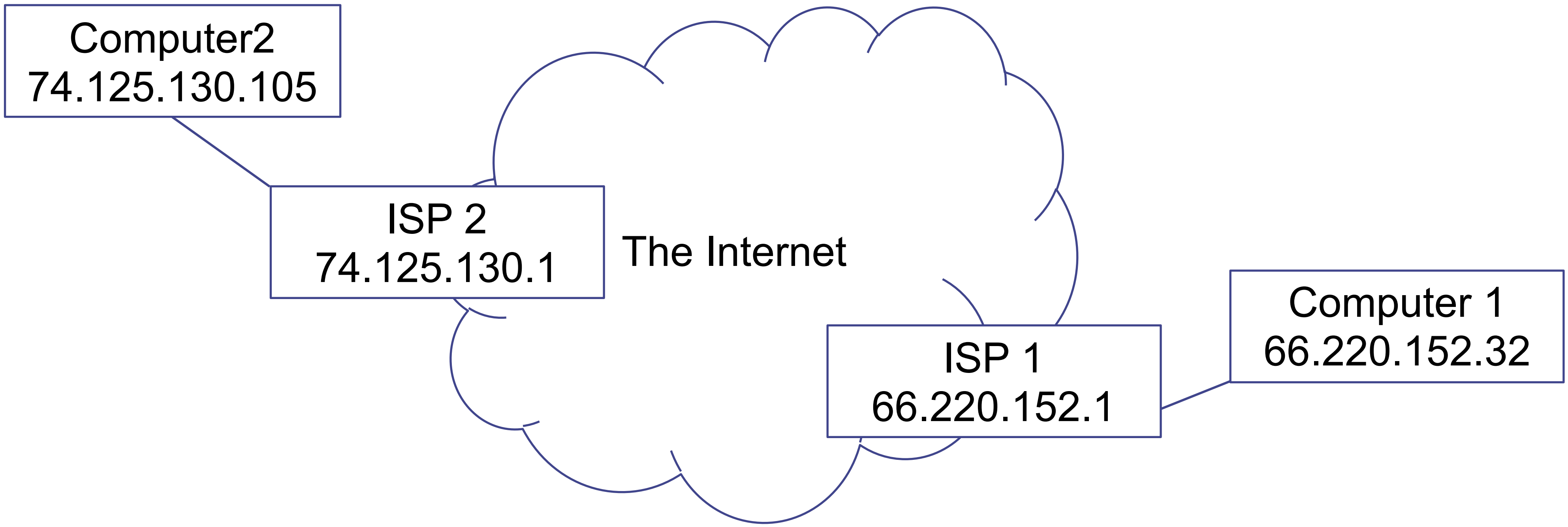
- Header says what network layer protocol the pay load is

# 3 Network Layer

- Layer 2 let’s computers on **same** network talk
- Layer 3 let’s computers talk across networks
  - Addressing
    - How do we specify what computer to talk to?
  - Routing
    - How do we get from here to there?

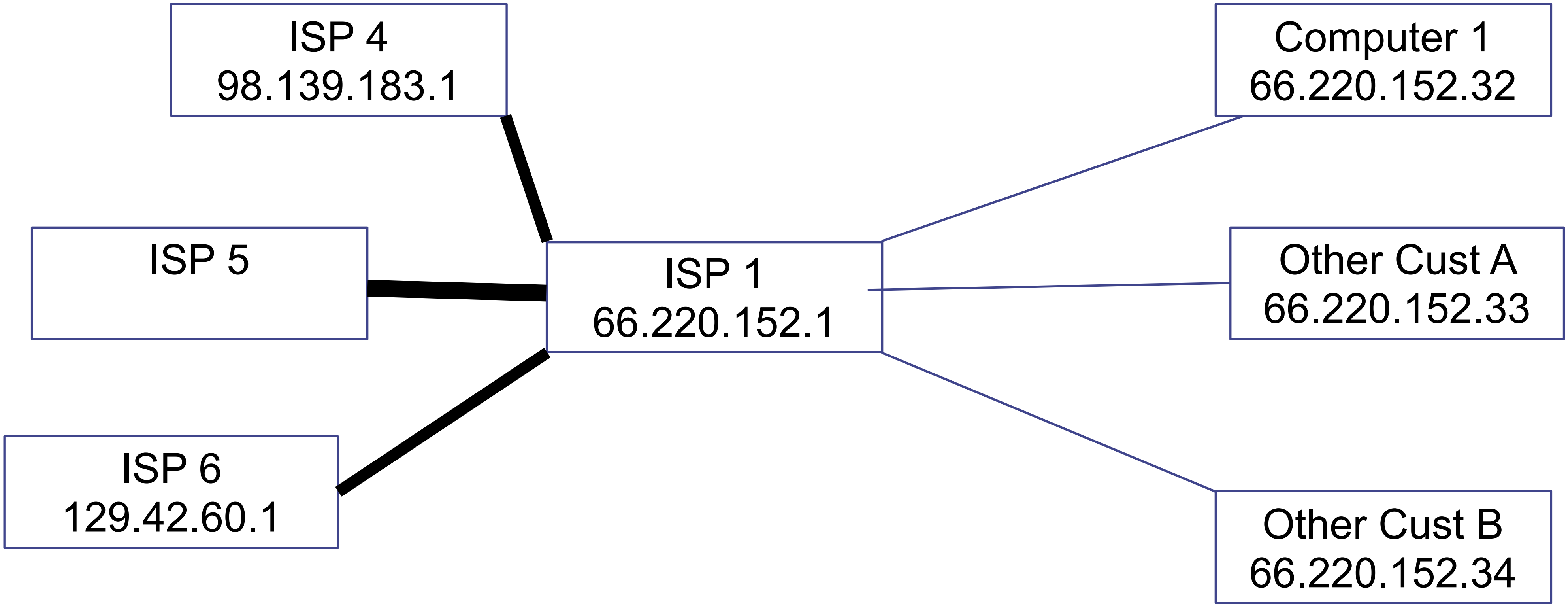
- Example: IP protocol
  - IPv4 and IPv6: pretty similar in most core regards
  - **Best effort delivery**
  - Addressing (IP addresses)
  - Routing
  - Analogy: Mailing a letter

# IP



- Computer 1 wants to send data to Computer 2
  - For now, assume it knows IP address (we'll see DNS later)
  - Has direct connection to its ISP… but then what?
    - The internet is a big place after all..

# Let’s zoom in on ISP 1



- ISP has connections to a handful of other places
  - Generally very high bandwidth connections
  - Will send your data (packet) to one of these, but which one?

# IP Routing

- IP addresses are hierarchical
  - May not know how to find 74.125.130.105…
  - But know which way to go to get to 74.______
  - Move one step closer
  - Within 74 network, know how to find 74.125
  - Then 74.125.130
  - Then find 74.125.130.105
  - Analogy:
    - How do I get to 2200 Mission College Blvd, Santa Clara, CA?

# IP Routing

- Analogy:
  - How do I get to 2200 Mission College Blvd, Santa Clara, CA?
  - I have no idea, but I can get you to I-40 West
    - Then you can ask someone else when you get to CA
  - Once in CA, you ask someone else:
    - "I only know its north of here, so take the 5 North and ask someone else"
  - Etc..
- This works because our physical addresses are hierarchical: Country, State, City, Street, Number

# Routing Basics

- Routing is done with tables
  - CIDR notation: 40.1.0.0/16
    - Match first 16 bits of 40.1.0.0, ignore remaining 16 bits
    - Each number in IP addr is 8 bits, written in decimal
  - Find match, entry tells what link to send out on
  - Example
    - 40.0.0.0/8 => Link 0
    - 50.1.0.0/16 => Link 1
    - 50.2.0.0/16 => Link 2
    - 50.3.27.0/24 => Link 3
    - 50.3.42.0/24 => Link 1

# Routing: More Complex

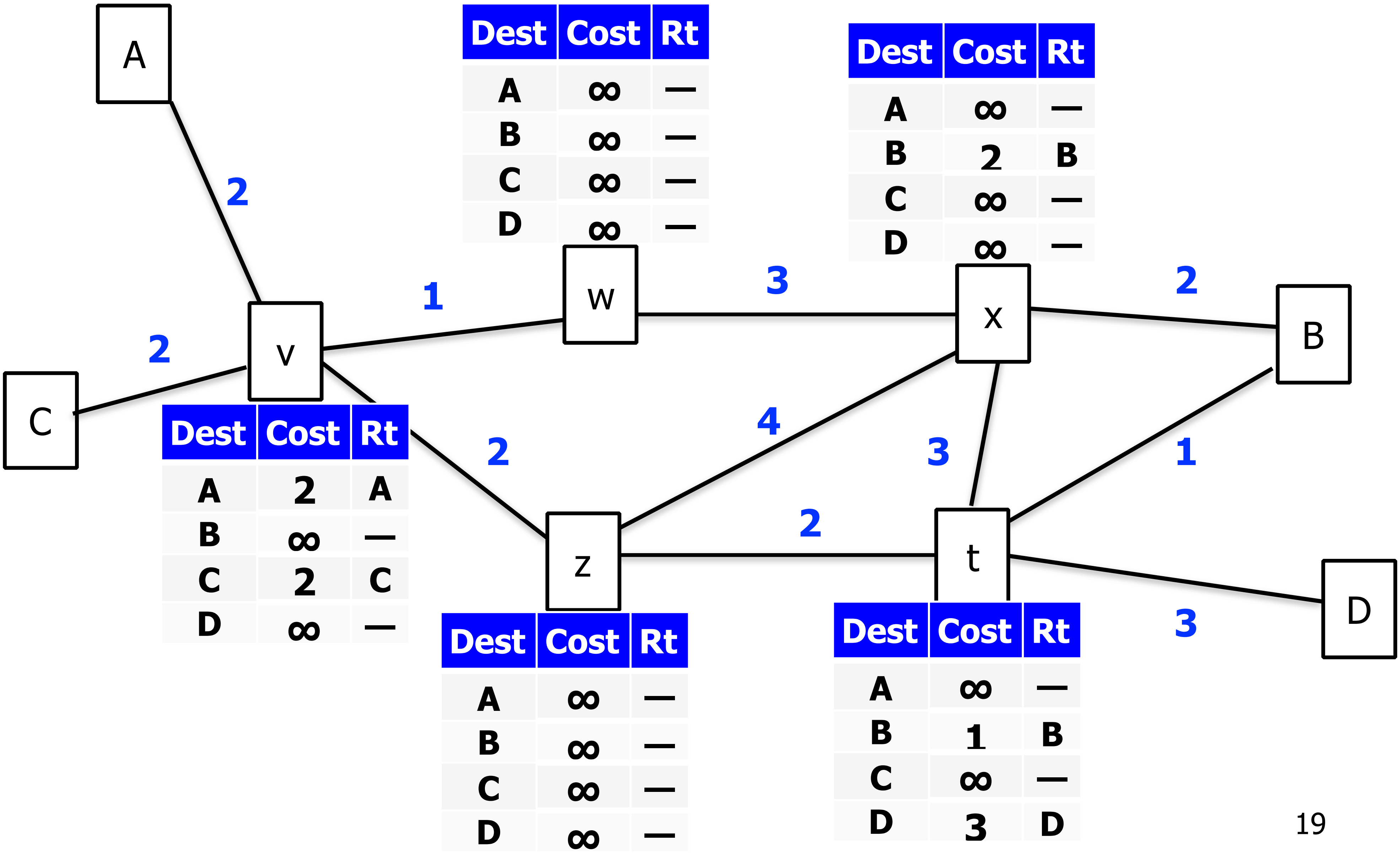
- Approach one: Static Routing
  - Enter all routes
  - Let system run
  - Hope nothing goes down
  - Works fine for small networks

- Reality:
  - Network links/systems go down
  - Often multiple paths to same place
    - Changing traffic patterns = changing fastest route

# Distance Vector Protocols

- Routers
  - Know distances to immediate neighbors
  - Compute distance vector
    - How far to any destination from all known info
  - Transmit distance vectors to neighbors
    - Discover better (shorter) route?  Update table
  - Now know more info, so repeat process

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | ∞ | — |
| C | ∞ | — |
| D | ∞ | — |

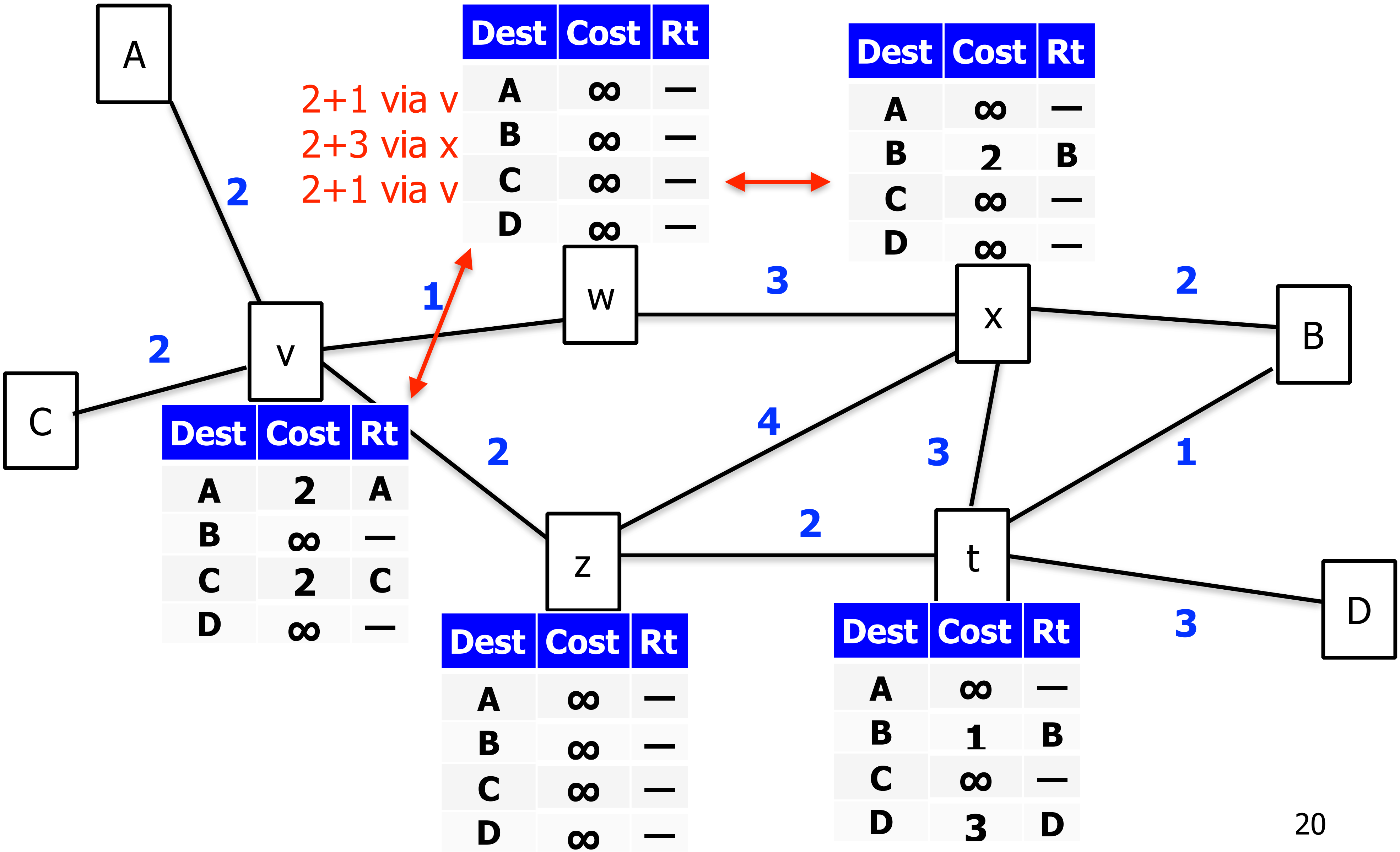
| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | 2 | B |
| C | ∞ | — |
| D | ∞ | — |

| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | ∞ | — |
| C | 2 | C |
| D | ∞ | — |

| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | ∞ | — |
| C | ∞ | — |
| D | ∞ | — |

| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | 1 | B |
| C | ∞ | — |
| D | 3 | D |

# Distance Vector Routing



2+1 via v
2+3 via x
2+1 via v

| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | ∞ | — |
| C | ∞ | — |
| D | ∞ | — |

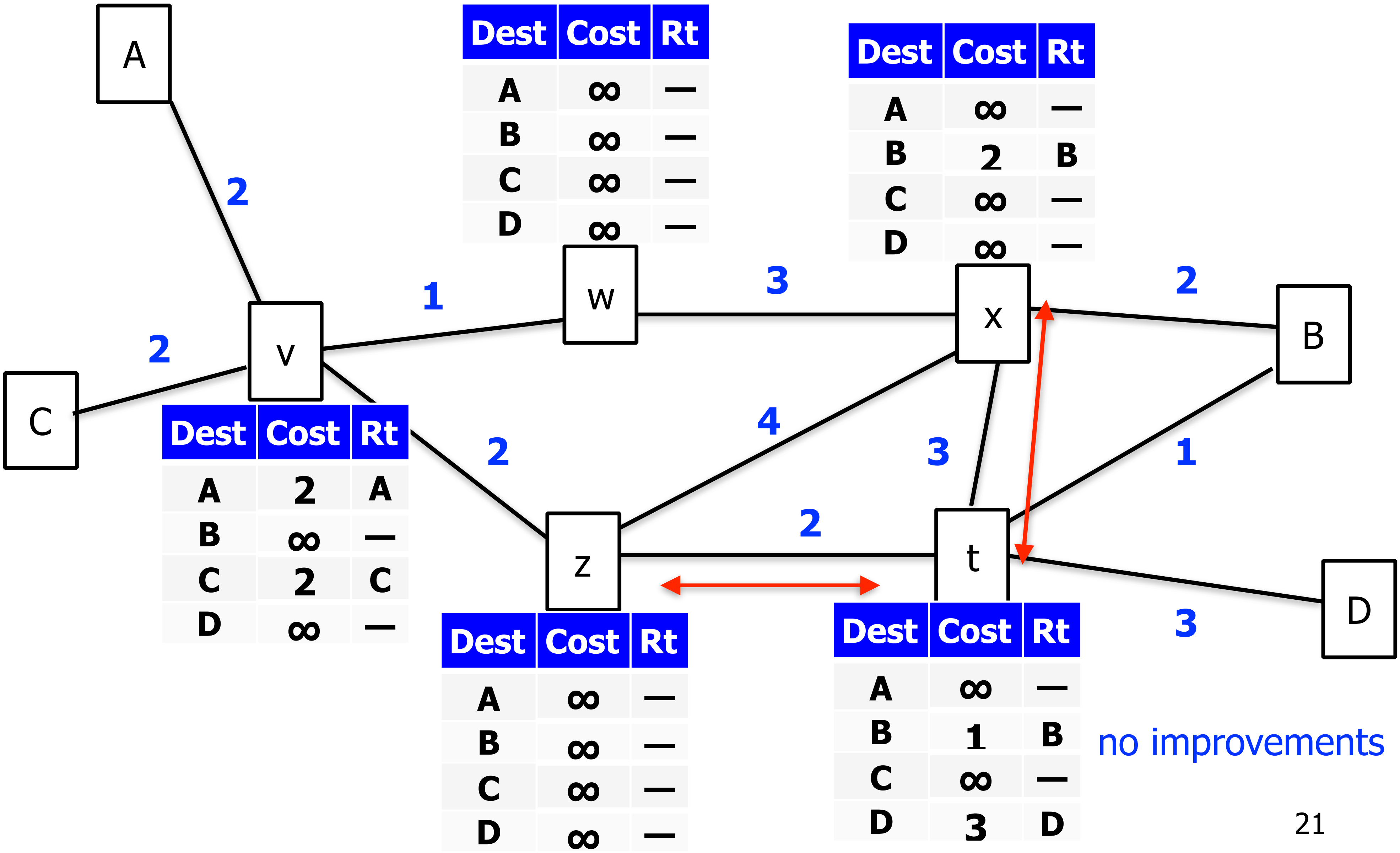
| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | 2 | B |
| C | ∞ | — |
| D | ∞ | — |

| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | ∞ | — |
| C | 2 | C |
| D | ∞ | — |

| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | ∞ | — |
| C | ∞ | — |
| D | ∞ | — |

| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | 1 | B |
| C | ∞ | — |
| D | 3 | D |

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | ∞ | — |
| C | ∞ | — |
| D | ∞ | — |

| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | 2 | B |
| C | ∞ | — |
| D | ∞ | — |

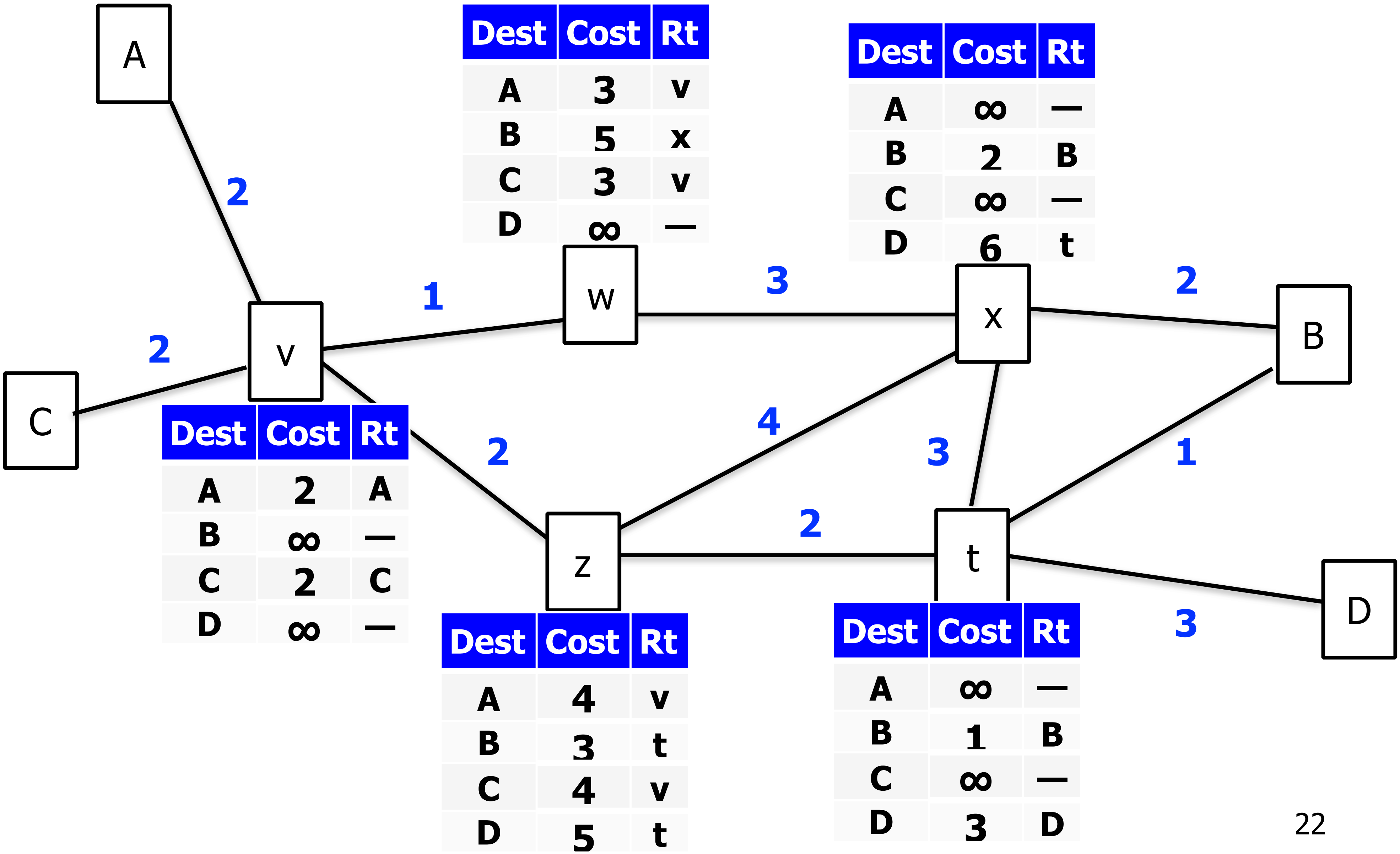
| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | ∞ | — |
| C | 2 | C |
| D | ∞ | — |

| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | ∞ | — |
| C | ∞ | — |
| D | ∞ | — |

| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | 1 | B |
| C | ∞ | — |
| D | 3 | D |

no improvements

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 3 | v |
| B | 5 | x |
| C | 3 | v |
| D | ∞ | — |

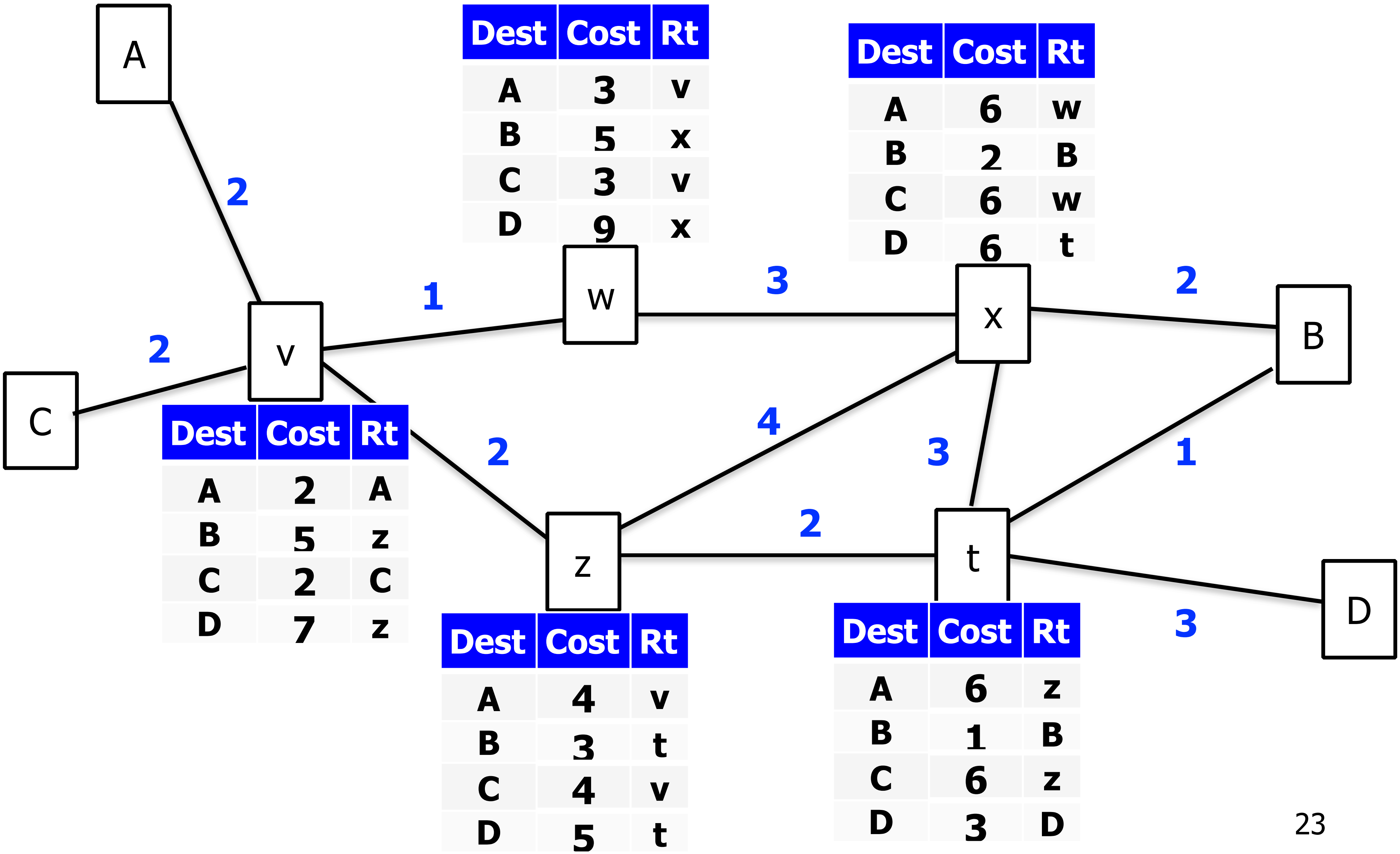
| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | 2 | B |
| C | ∞ | — |
| D | 6 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | ∞ | — |
| C | 2 | C |
| D | ∞ | — |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | ∞ | — |
| B | 1 | B |
| C | ∞ | — |
| D | 3 | D |

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 3 | v |
| B | 5 | x |
| C | 3 | v |
| D | 9 | x |

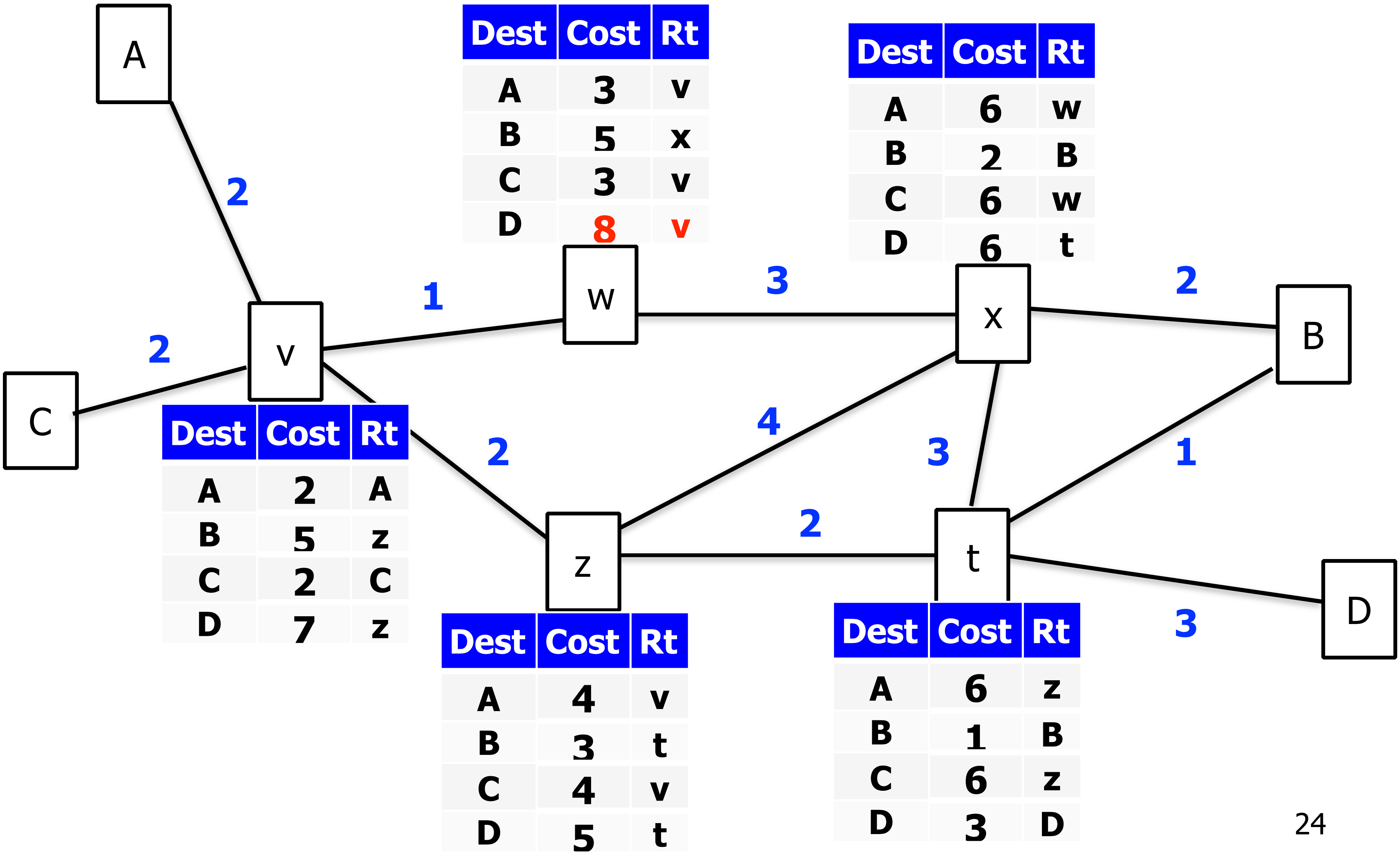
| Dest | Cost | Rt |
|---|---|---|
| A | 6 | w |
| B | 2 | B |
| C | 6 | w |
| D | 6 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | 5 | z |
| C | 2 | C |
| D | 7 | z |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 6 | z |
| B | 1 | B |
| C | 6 | z |
| D | 3 | D |

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 3 | v |
| B | 5 | x |
| C | 3 | v |
| D | 8 | v |

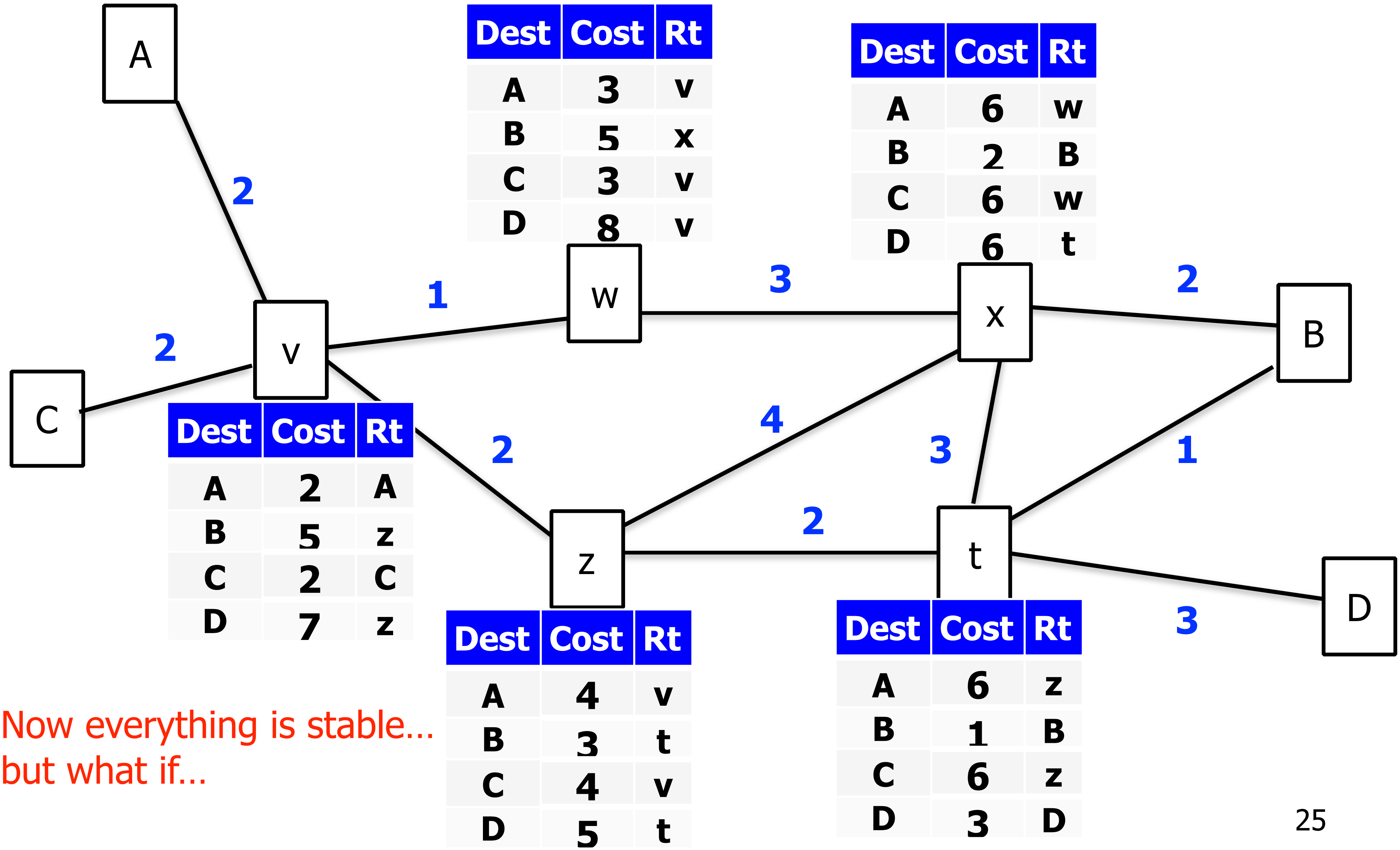
| Dest | Cost | Rt |
|---|---|---|
| A | 6 | w |
| B | 2 | B |
| C | 6 | w |
| D | 6 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | 5 | z |
| C | 2 | C |
| D | 7 | z |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 6 | z |
| B | 1 | B |
| C | 6 | z |
| D | 3 | D |

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 3 | v |
| B | 5 | x |
| C | 3 | v |
| D | 8 | v |

| Dest | Cost | Rt |
|---|---|---|
| A | 6 | w |
| B | 2 | B |
| C | 6 | w |
| D | 6 | t |

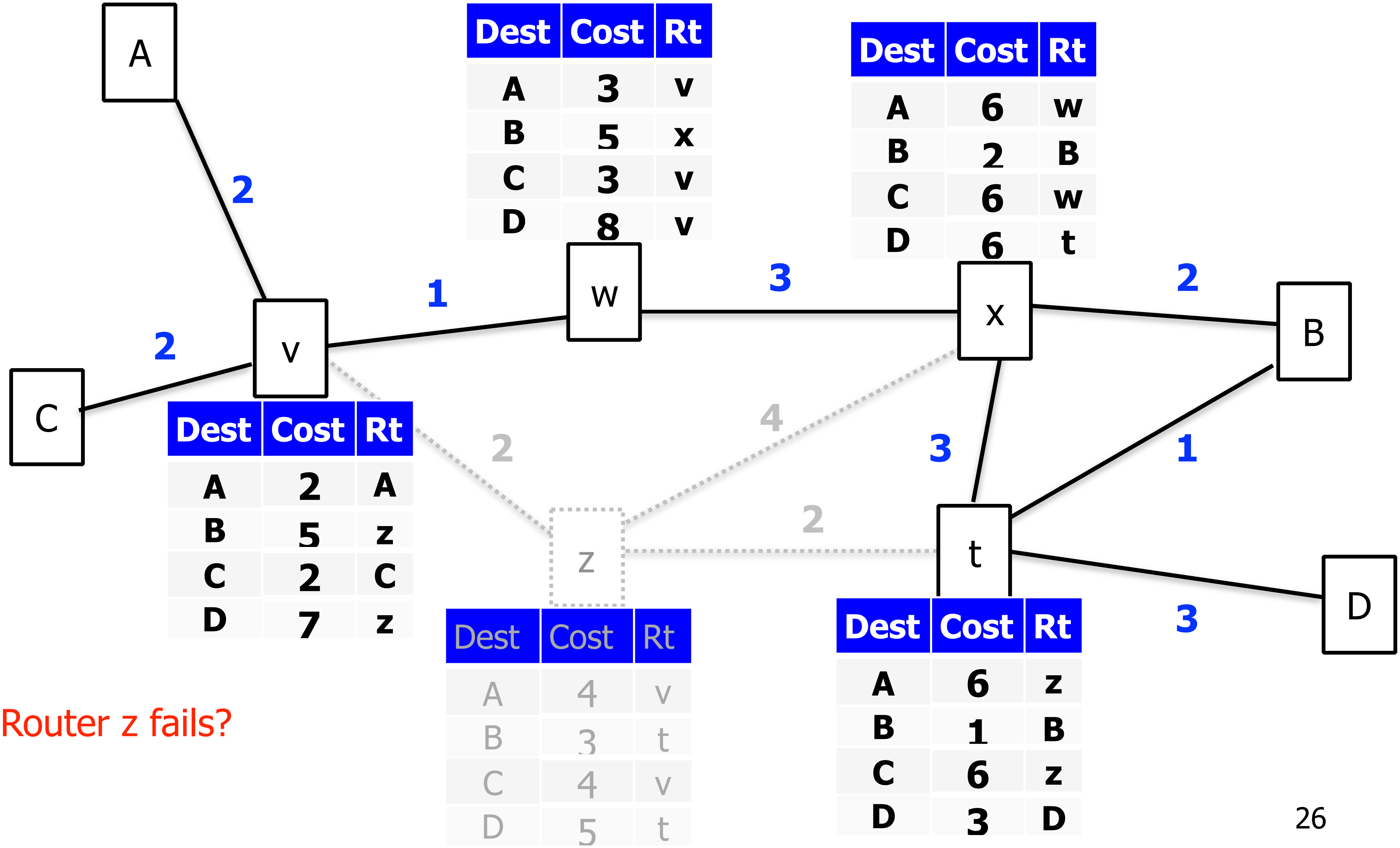
| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | 5 | z |
| C | 2 | C |
| D | 7 | z |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 6 | z |
| B | 1 | B |
| C | 6 | z |
| D | 3 | D |

Now everything is stable…
but what if…

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 3 | v |
| B | 5 | x |
| C | 3 | v |
| D | 8 | v |

| Dest | Cost | Rt |
|---|---|---|
| A | 6 | w |
| B | 2 | B |
| C | 6 | w |
| D | 6 | t |

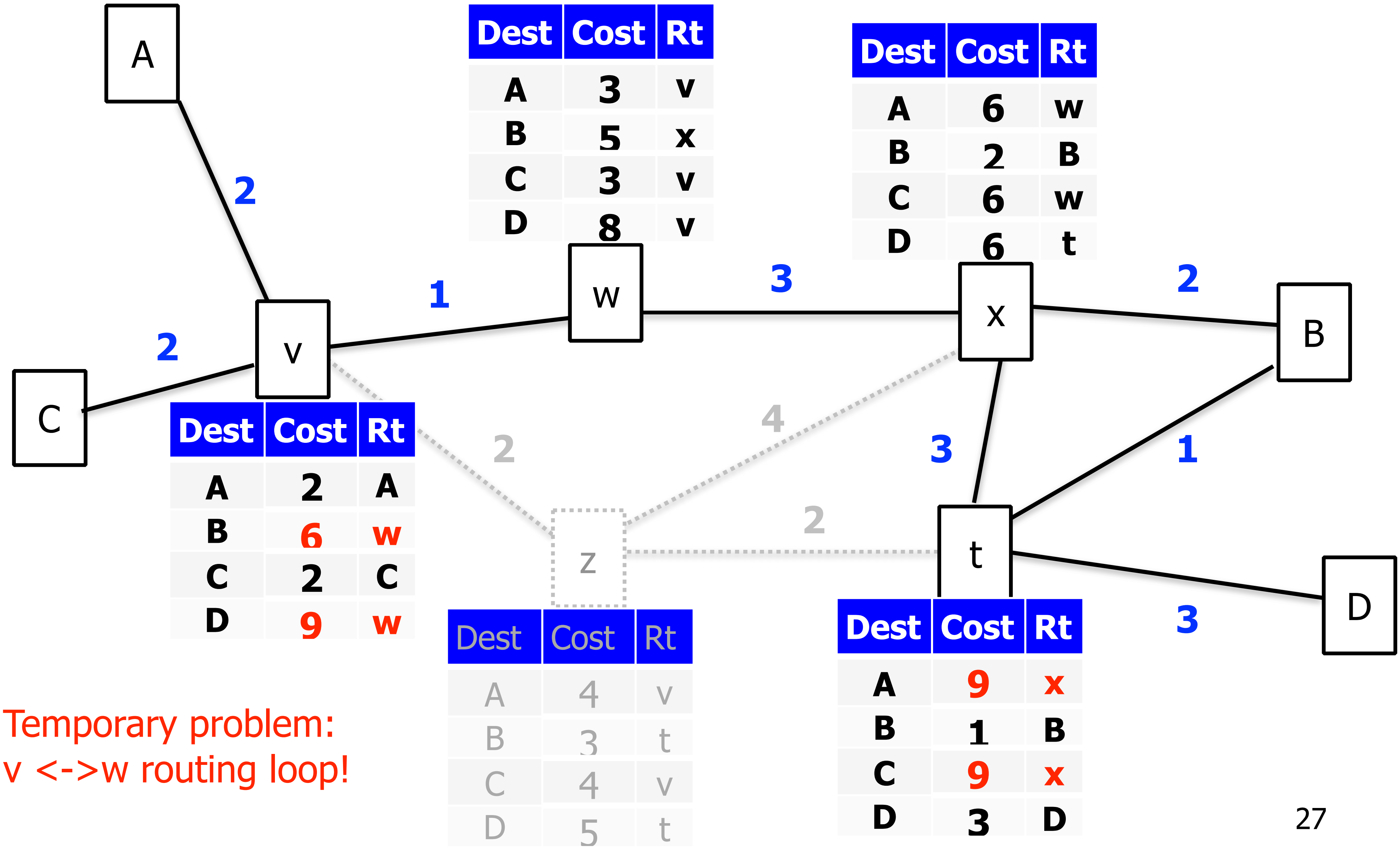
| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | 5 | z |
| C | 2 | C |
| D | 7 | z |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 6 | z |
| B | 1 | B |
| C | 6 | z |
| D | 3 | D |

Router z fails?

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 3 | v |
| B | 5 | x |
| C | 3 | v |
| D | 8 | v |

| Dest | Cost | Rt |
|---|---|---|
| A | 6 | w |
| B | 2 | B |
| C | 6 | w |
| D | 6 | t |

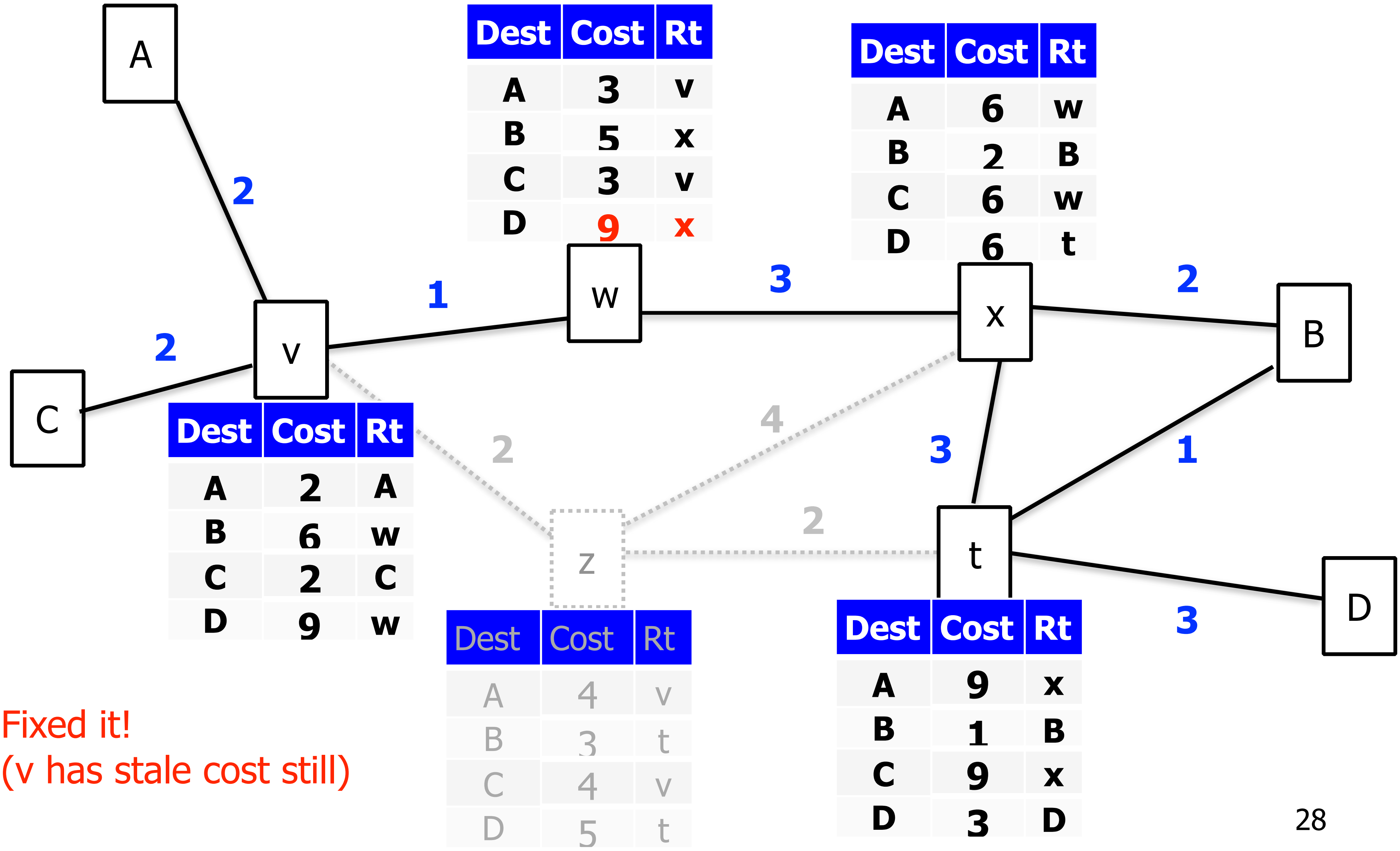
| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | 6 | w |
| C | 2 | C |
| D | 9 | w |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 9 | x |
| B | 1 | B |
| C | 9 | x |
| D | 3 | D |

Temporary problem:
v <->w routing loop!

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 3 | v |
| B | 5 | x |
| C | 3 | v |
| D | 9 | x |

| Dest | Cost | Rt |
|---|---|---|
| A | 6 | w |
| B | 2 | B |
| C | 6 | w |
| D | 6 | t |

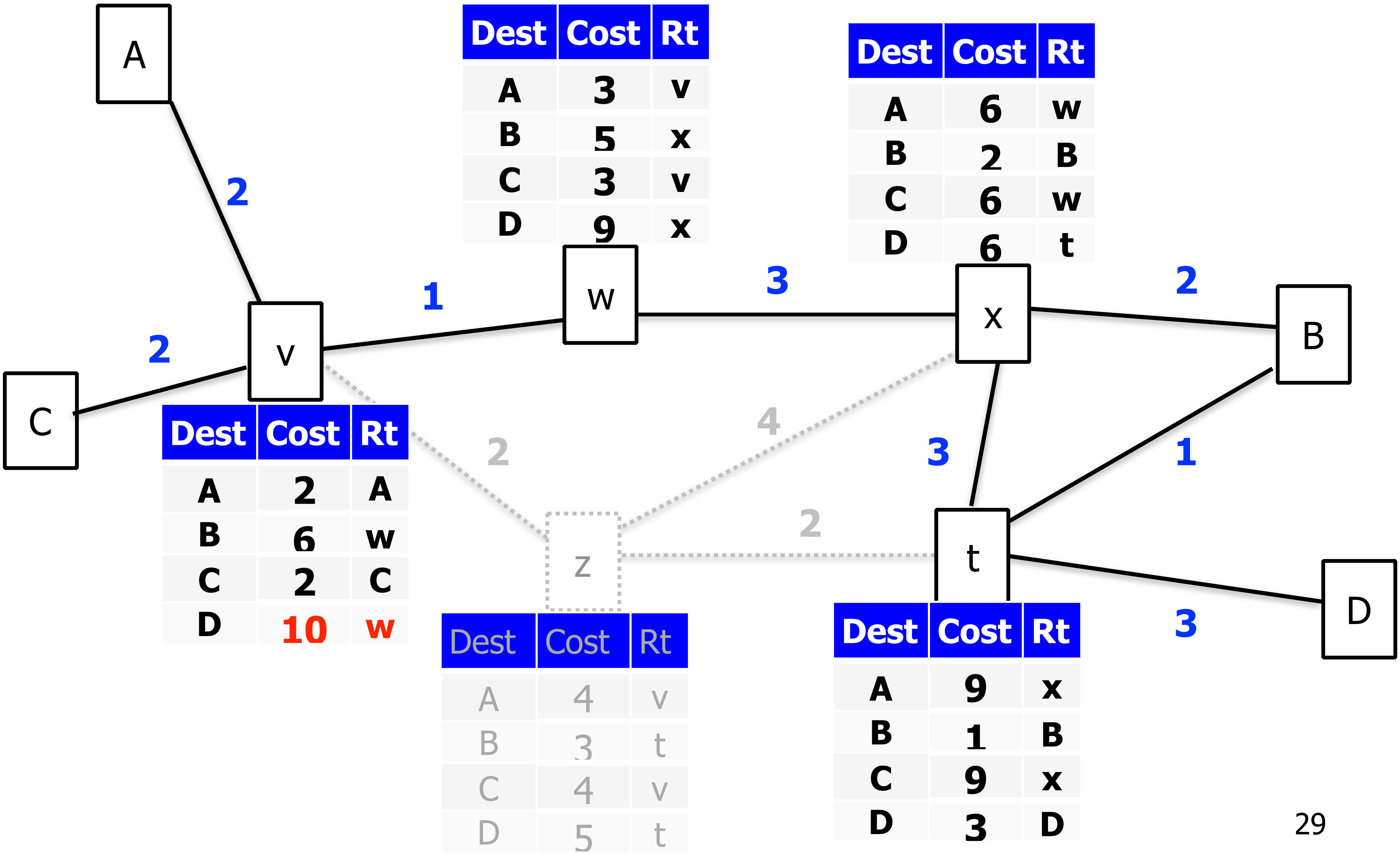
| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | 6 | w |
| C | 2 | C |
| D | 9 | w |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 9 | x |
| B | 1 | B |
| C | 9 | x |
| D | 3 | D |

Fixed it!
(v has stale cost still)

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 3 | v |
| B | 5 | x |
| C | 3 | v |
| D | 9 | x |

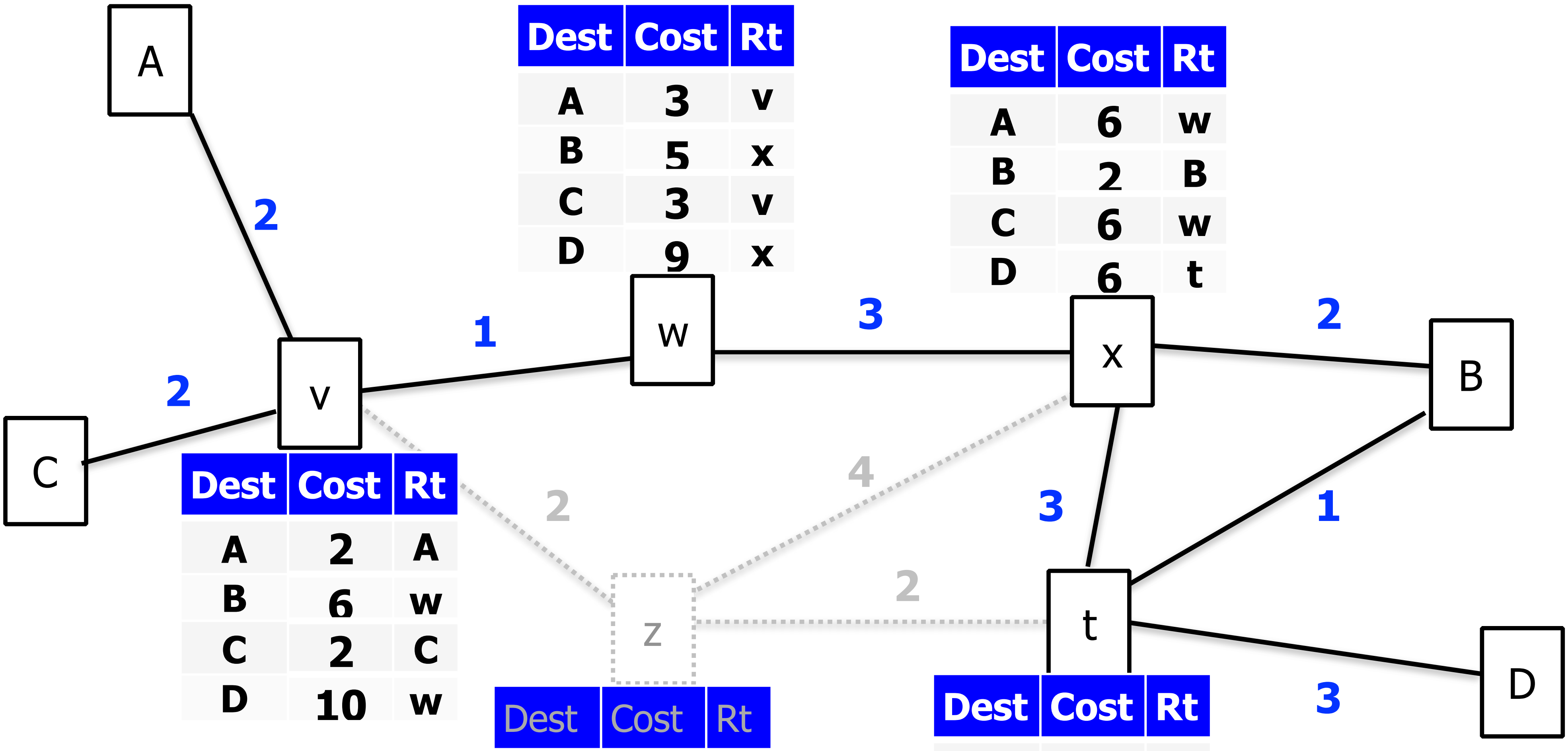
| Dest | Cost | Rt |
|---|---|---|
| A | 6 | w |
| B | 2 | B |
| C | 6 | w |
| D | 6 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | 6 | w |
| C | 2 | C |
| D | 10 | w |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 9 | x |
| B | 1 | B |
| C | 9 | x |
| D | 3 | D |

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 3 | v |
| B | 5 | x |
| C | 3 | v |
| D | 9 | x |

| Dest | Cost | Rt |
|---|---|---|
| A | 6 | w |
| B | 2 | B |
| C | 6 | w |
| D | 6 | t |

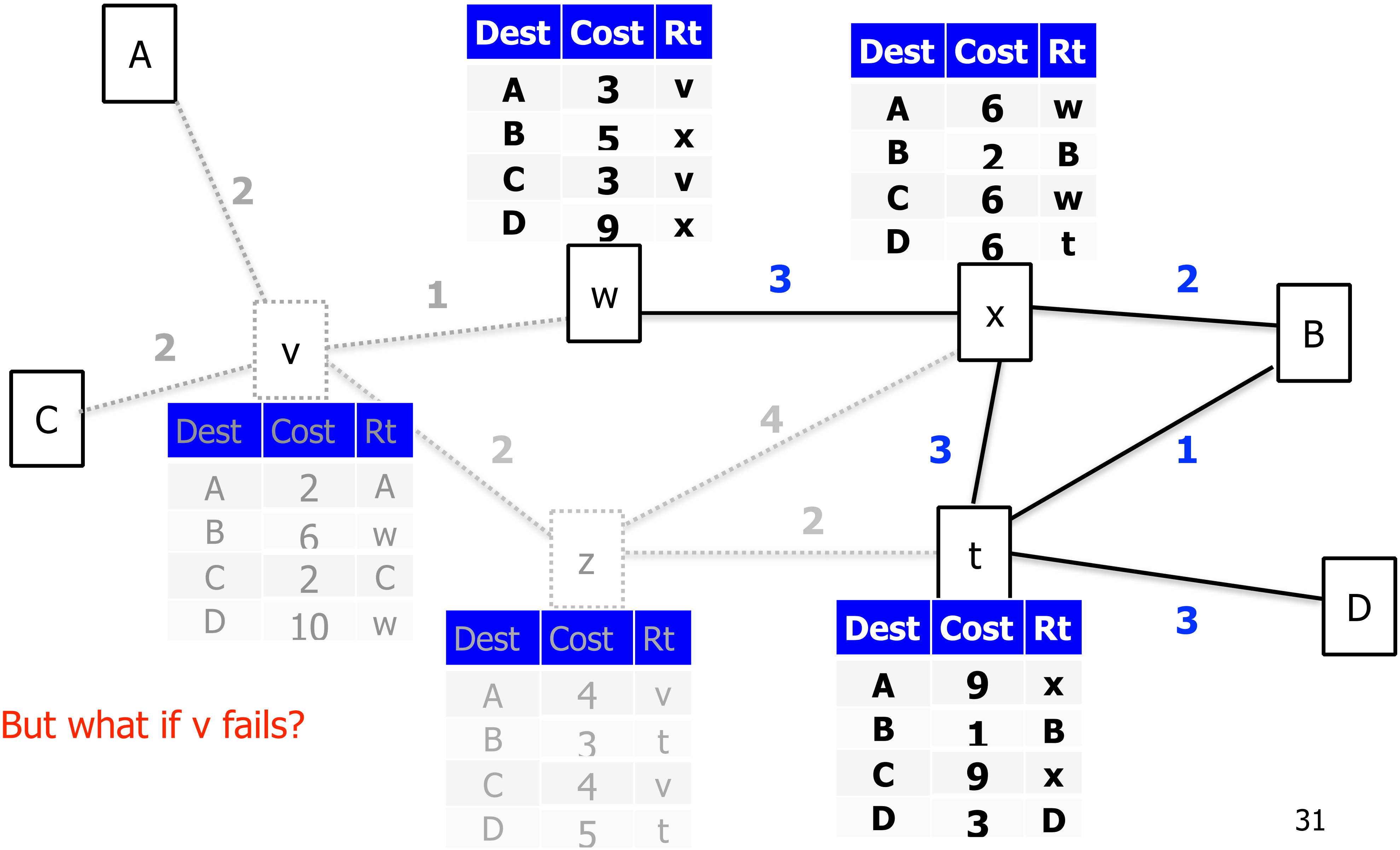
| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | 6 | w |
| C | 2 | C |
| D | 10 | w |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 9 | x |
| B | 1 | B |
| C | 9 | x |
| D | 3 | D |

Stable again.
If z comes back, we'll rediscover better routes

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 3 | v |
| B | 5 | x |
| C | 3 | v |
| D | 9 | x |

| Dest | Cost | Rt |
|---|---|---|
| A | 6 | w |
| B | 2 | B |
| C | 6 | w |
| D | 6 | t |

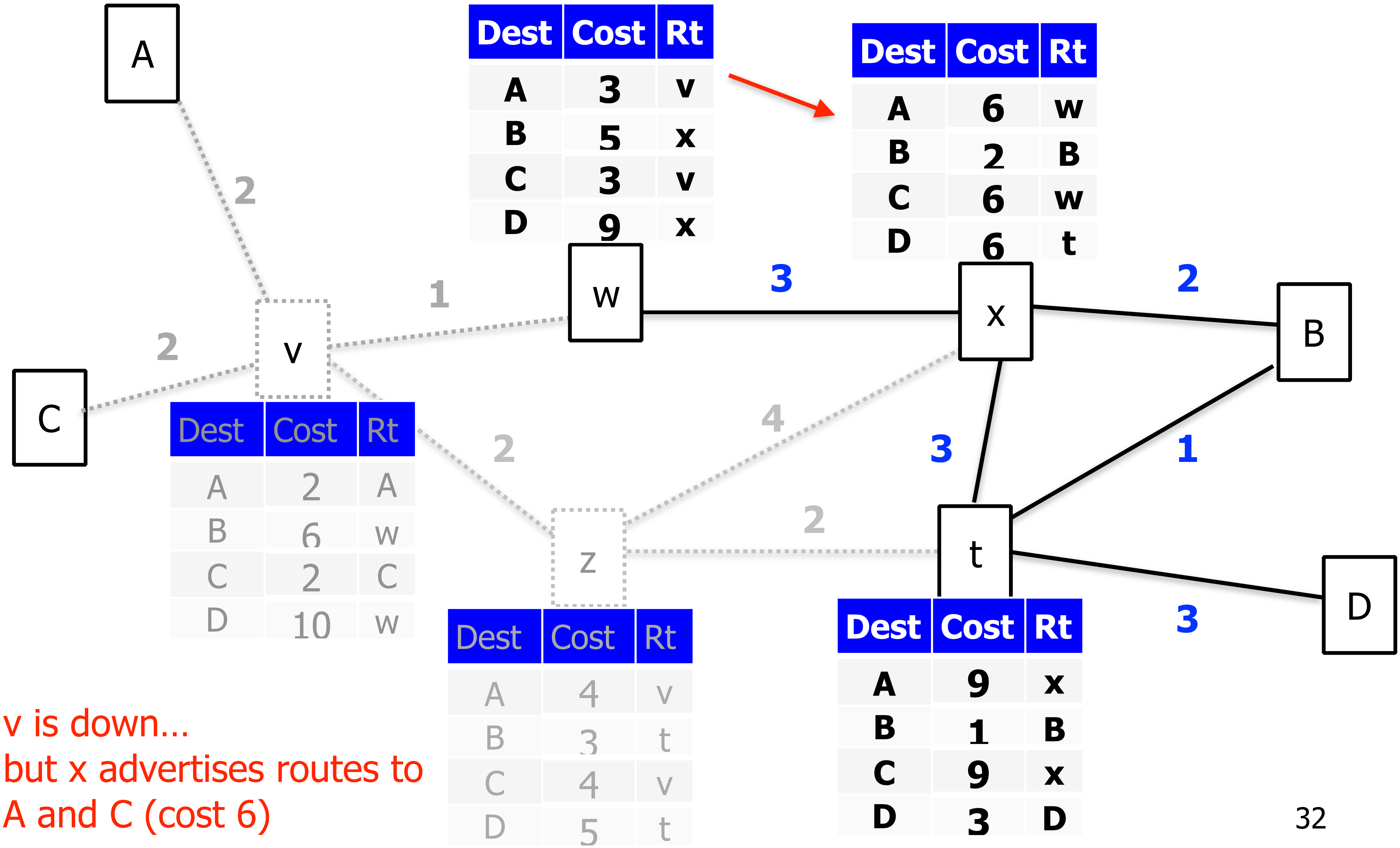
| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | 6 | w |
| C | 2 | C |
| D | 10 | w |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 9 | x |
| B | 1 | B |
| C | 9 | x |
| D | 3 | D |

But what if v fails?

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 3 | v |
| B | 5 | x |
| C | 3 | v |
| D | 9 | x |

| Dest | Cost | Rt |
|---|---|---|
| A | 6 | w |
| B | 2 | B |
| C | 6 | w |
| D | 6 | t |

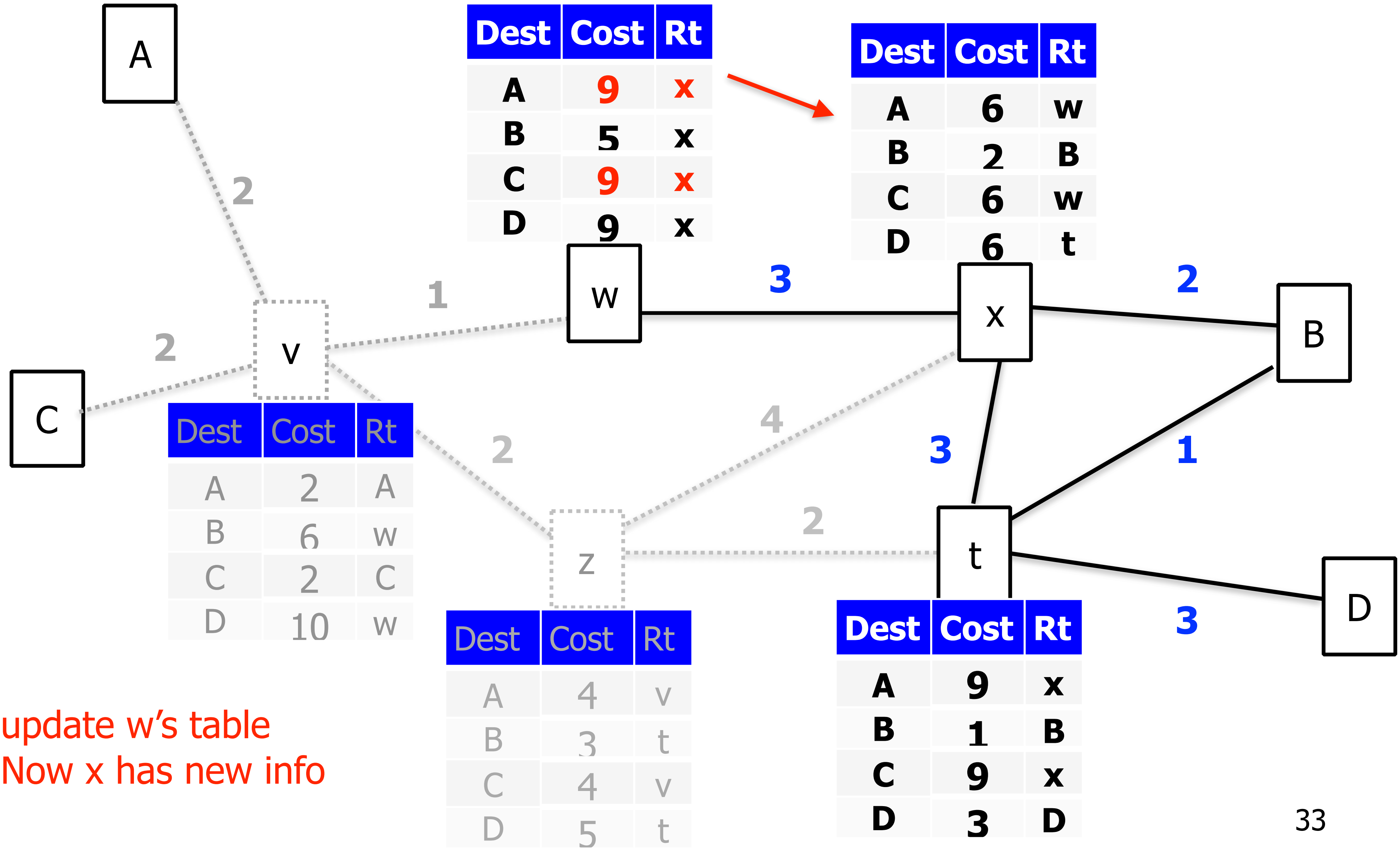
| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | 6 | w |
| C | 2 | C |
| D | 10 | w |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 9 | x |
| B | 1 | B |
| C | 9 | x |
| D | 3 | D |

v is down…
but x advertises routes to
A and C (cost 6)

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 9 | x |
| B | 5 | x |
| C | 9 | x |
| D | 9 | x |

| Dest | Cost | Rt |
|---|---|---|
| A | 6 | w |
| B | 2 | B |
| C | 6 | w |
| D | 6 | t |

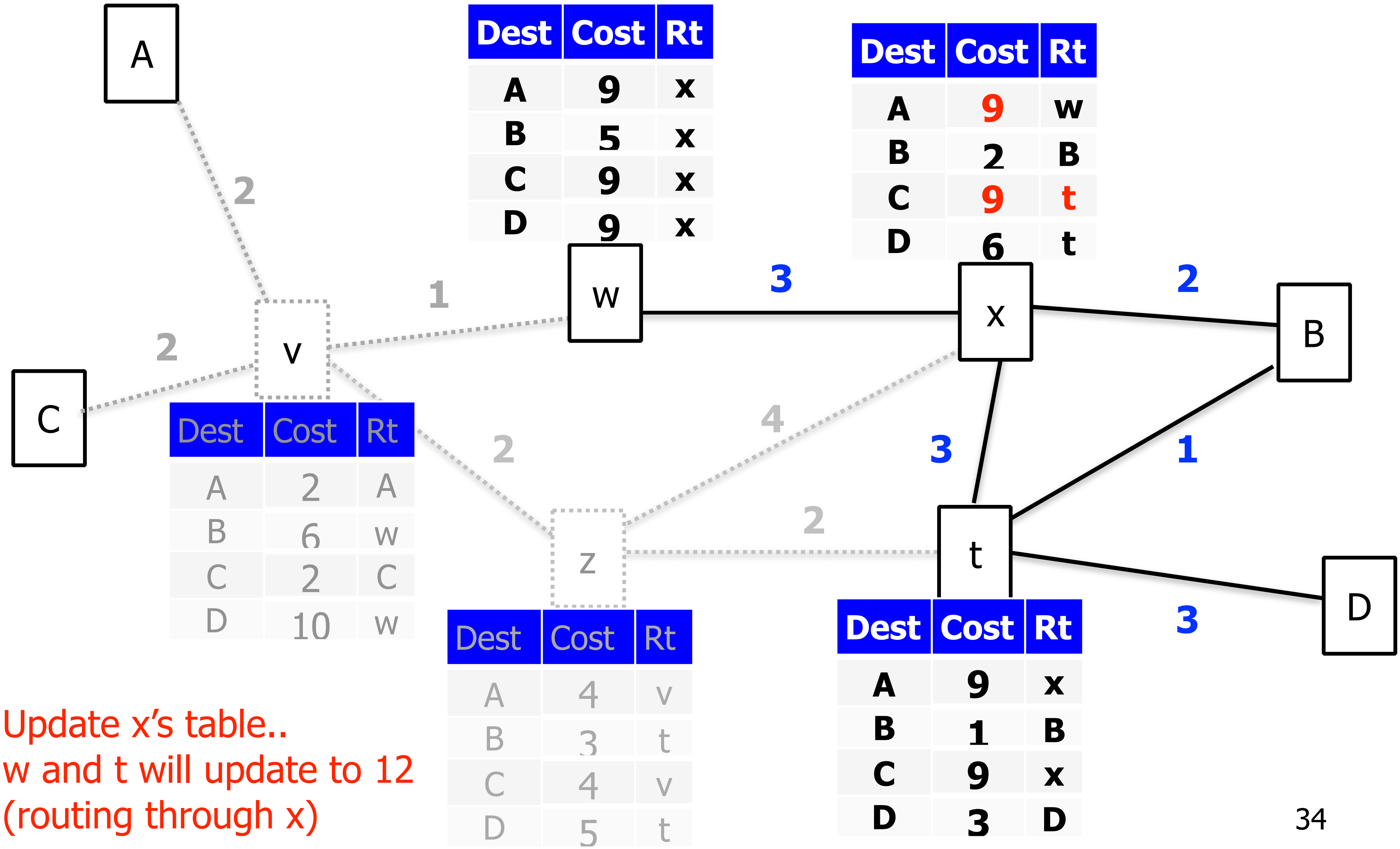
| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | 6 | w |
| C | 2 | C |
| D | 10 | w |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 9 | x |
| B | 1 | B |
| C | 9 | x |
| D | 3 | D |

update w's table
Now x has new info

# Distance Vector Routing



| Dest | Cost | Rt |
|---|---|---|
| A | 9 | x |
| B | 5 | x |
| C | 9 | x |
| D | 9 | x |

| Dest | Cost | Rt |
|---|---|---|
| A | 9 | w |
| B | 2 | B |
| C | 9 | t |
| D | 6 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 2 | A |
| B | 6 | w |
| C | 2 | C |
| D | 10 | w |

| Dest | Cost | Rt |
|---|---|---|
| A | 4 | v |
| B | 3 | t |
| C | 4 | v |
| D | 5 | t |

| Dest | Cost | Rt |
|---|---|---|
| A | 9 | x |
| B | 1 | B |
| C | 9 | x |
| D | 3 | D |

Update x's table..
w and t will update to 12
(routing through x)

# Count-to-infinity

- Algorithm will slowly "count to infinity"
  - Actually: count to max value it holds
  - Then throw away the route, concluding there is no path there

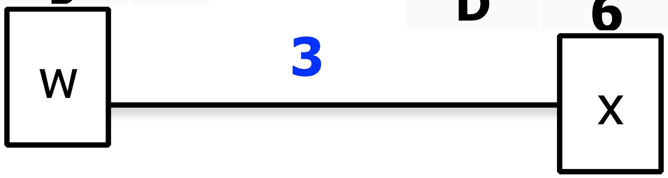
  - Packets sent in meanwhile?
    - IP: Time To Live (TTL)
    - Starts at fixed value (e.g., 255)
    - Decremented every time the packet is forwarded
    - Packet dropped when TTL == 0

    - Traceroute: uses TTL fields to probe to different distances
      - Uses ICMP protocol to get response on TTL
- Note: many fancier routing schemes, we aren't covering

# Distance Vector Routing

| Dest | Cost | Rt |
|---|---|---|
| A | 9 | x |
| B | 5 | x |
| C | 9 | x |
| D | 9 | x |

| Dest | Cost | Rt |
|---|---|---|
| A | 9 | w |
| B | 2 | B |
| C | 9 | t |
| D | 6 | t |



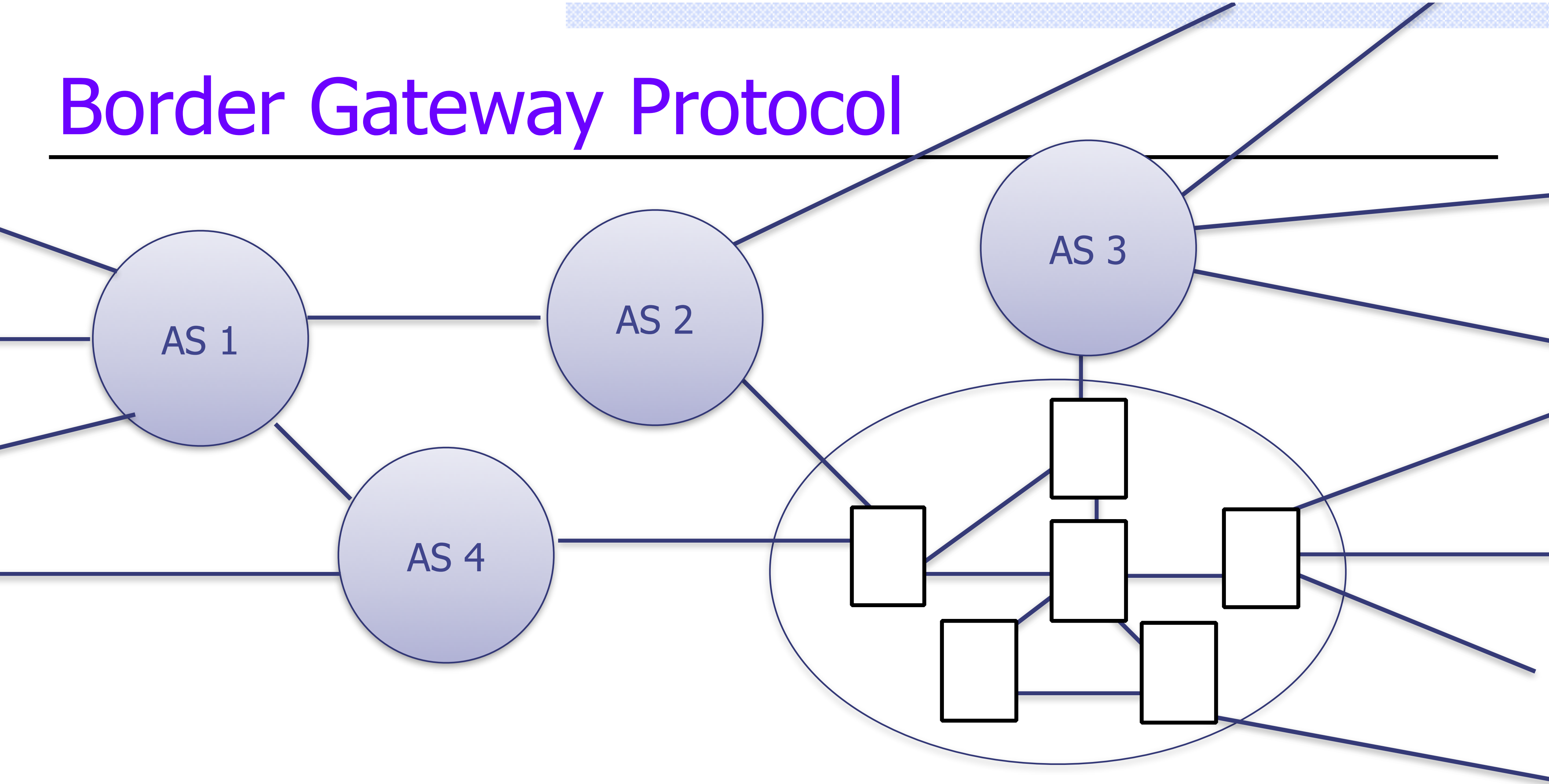
Obvious optimization:

- If x gets a route from w, it should not advertise that route back to w
- Called “split-horizon”
- Helps stabilize faster, but does not solve the problem (may still count to infinity)

# Link State Protocols

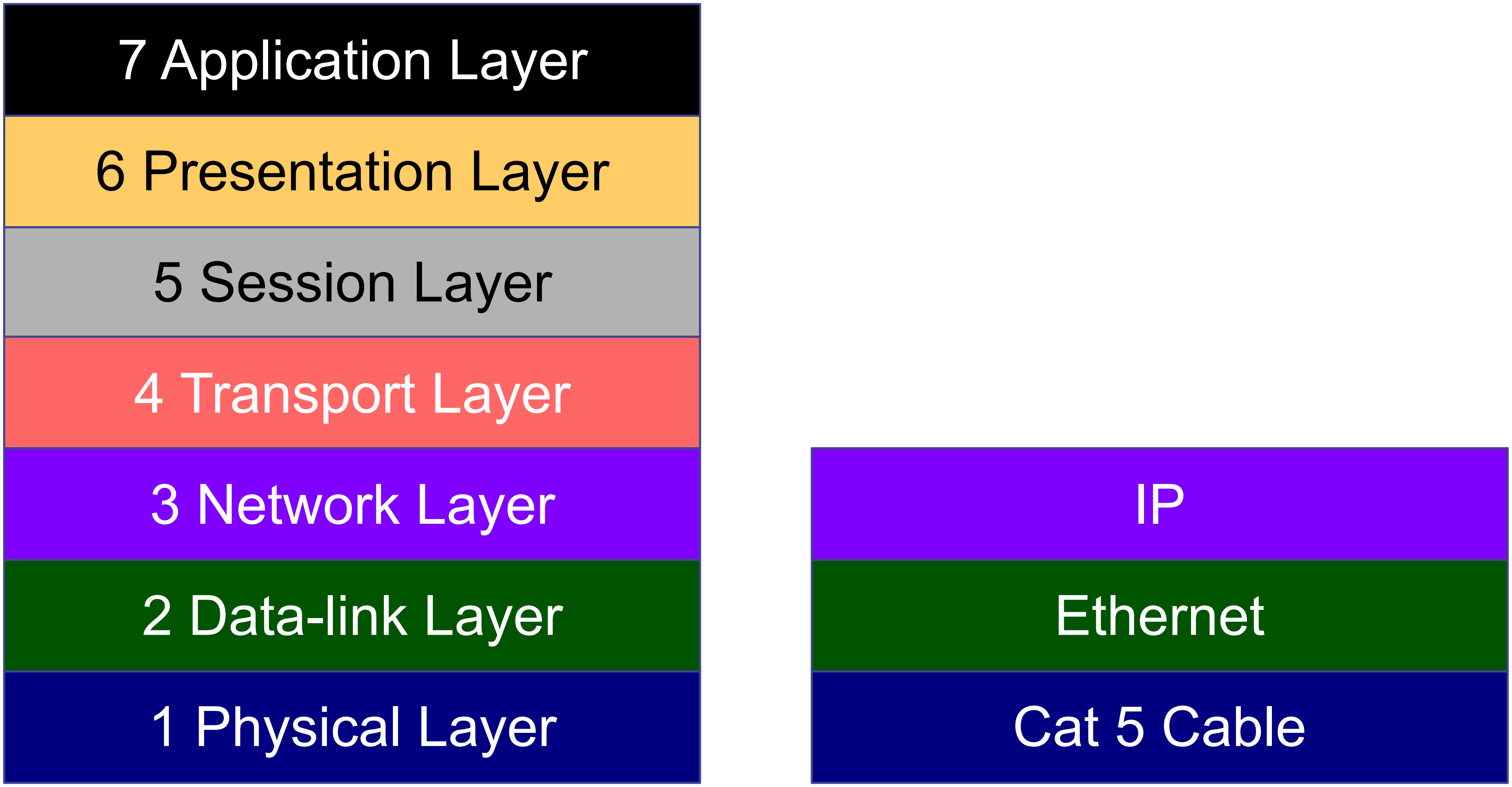
- Another option: link state protocols
  - Send info about direct connections to all routers
  - All routers build global pictures of network
  - Run graph algorithms to find shortest paths
    - E.g., Dijkstra's shortest path algorithm
- Global information is nice, but…
  - Complex for very large systems
  - How many routers on the internet?
  - Do they all exchange all their info and run Dijkstra's?
    - Of course not..
    - So… what do we do? **Use Abstraction… (and hierarchy)**

# Border Gateway Protocol



- Divide internet up into Autonomous Systems (ASes)
  - Each AS can advertise routes to other ASes
  - Routing internal to AS is hidden from outside world
    - Can be Link State, Distance Vector, other…
  - We won't go into too many details

# 7-layer OSI model



7 Application Layer

6 Presentation Layer

5 Session Layer

4 Transport Layer

3 Network Layer

2 Data-link Layer

1 Physical Layer

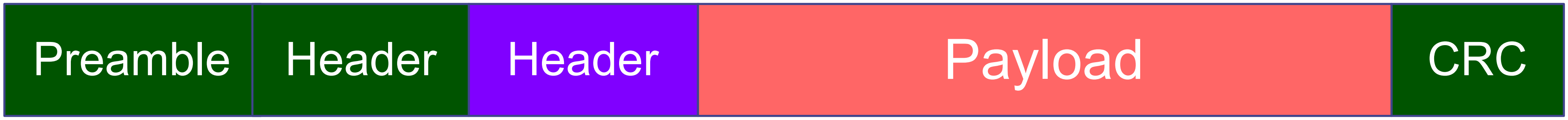
IP

Ethernet

Cat 5 Cable

- IP: hierarchical addresses + best effort delivery

# Our messages so far



Preamble | Header | Header | Payload | CRC

- IP Header has
  - Src IP
  - Dest IP
  - Payload type (what protocol)
  - Other info
- Side note: IP does fragmentation to fit within frame size
  - We aren't covering that

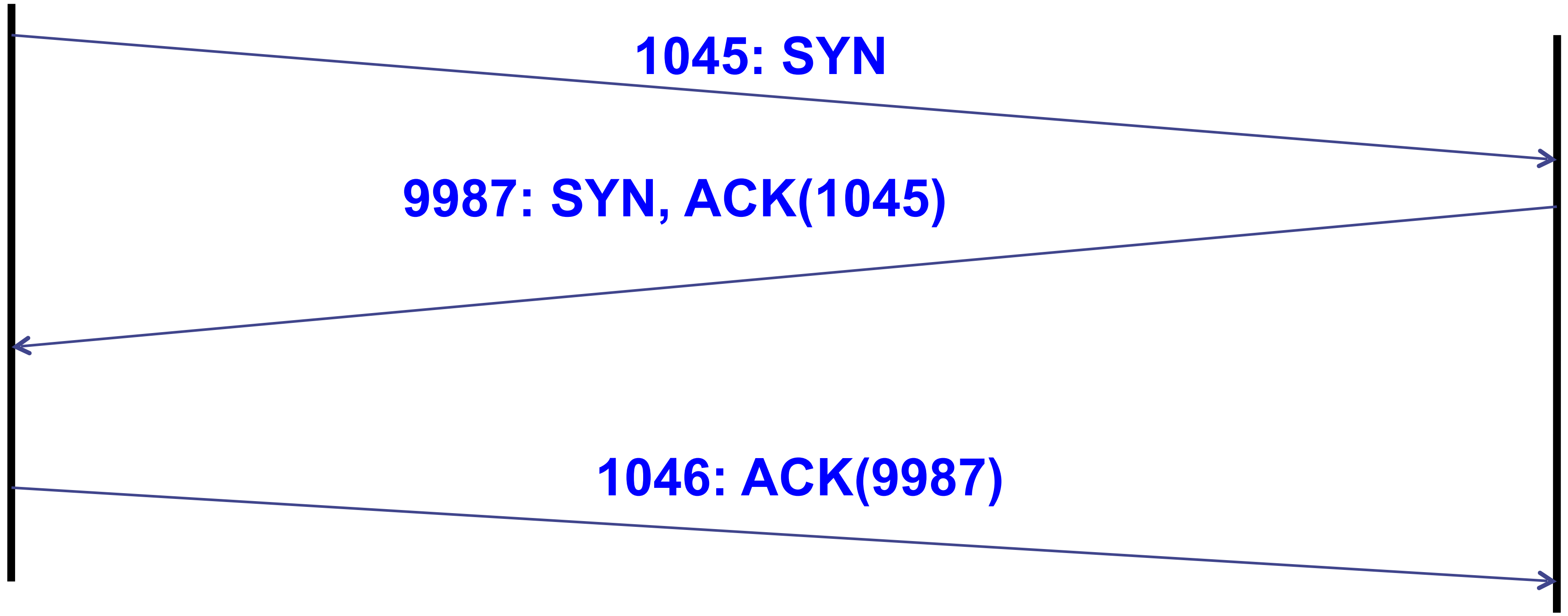
# 4: Transport Layer

- Reliability (if used)
  - Acknowledgements of data receipt
  - Retries of failed data
- Flow Control
  - Restrict rate of data sending
- Multiplexing/De-multiplexing data
  - E.g., Ports: identify which program some data is for
  - Keep data streams separate

# 5: Session Layer

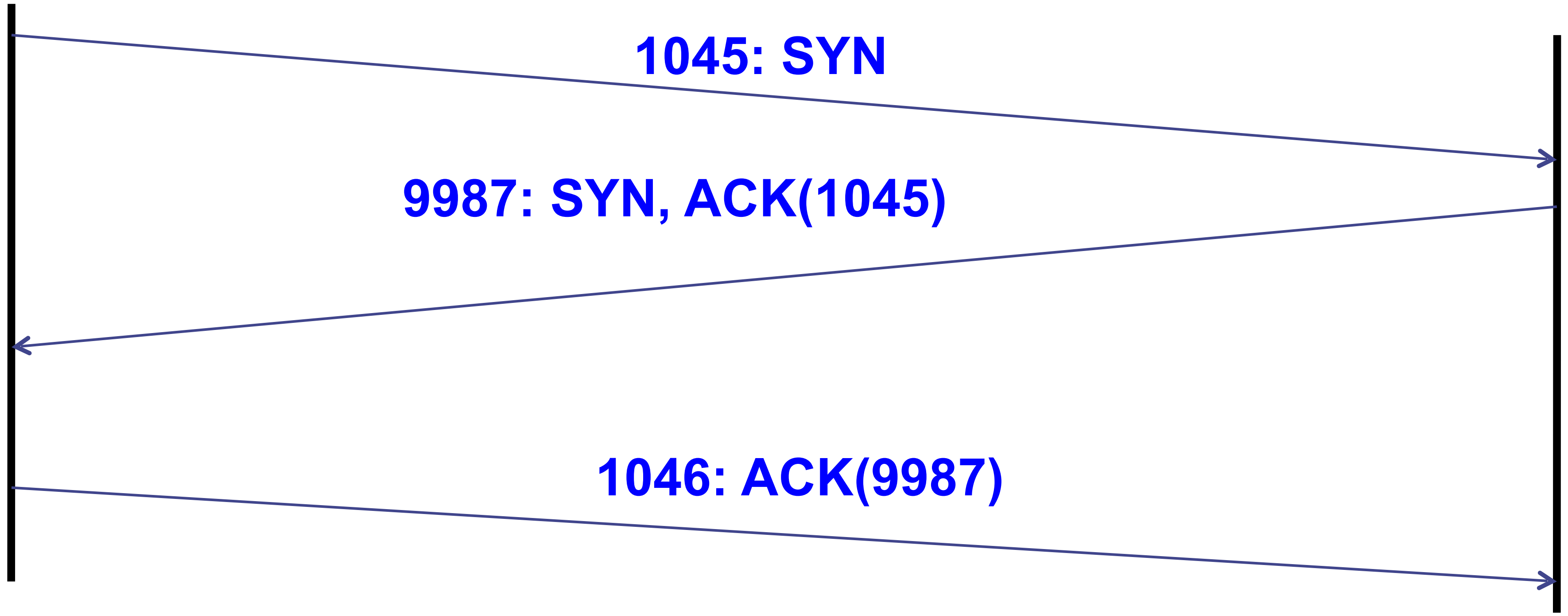
- Concept of “a connection”
  - Establish/terminate
  - (OSI includes a variety of obscure features not often used)
- TCP: combines these two layers together
  - Sets up/terminates sessions
  - Has sequence numbering for packets
  - Acknowledges (ACKs) packets that are received
  - Establishes flow control (responds to congestion by throttling sending)

# TCP



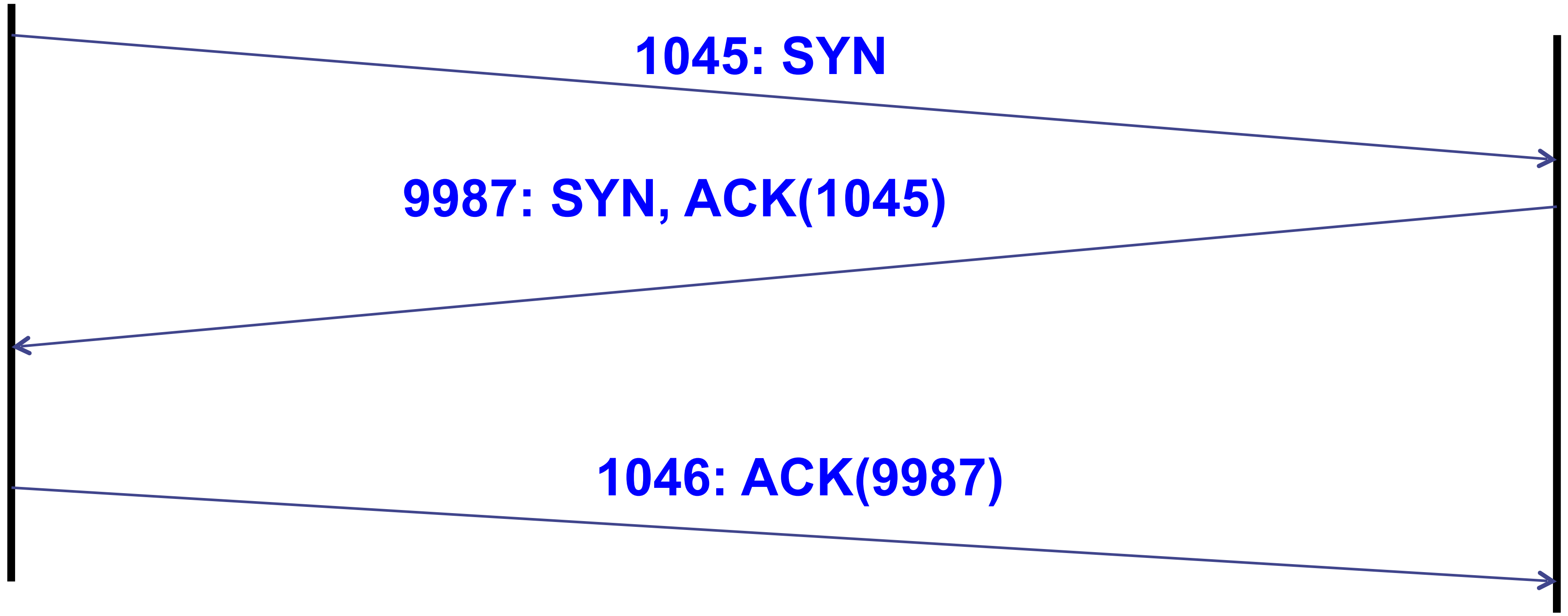
- We’ll draw diagram with computers on each side
- Time goes down
- Three messages above (TCP’s “3 way handshake”)

# TCP



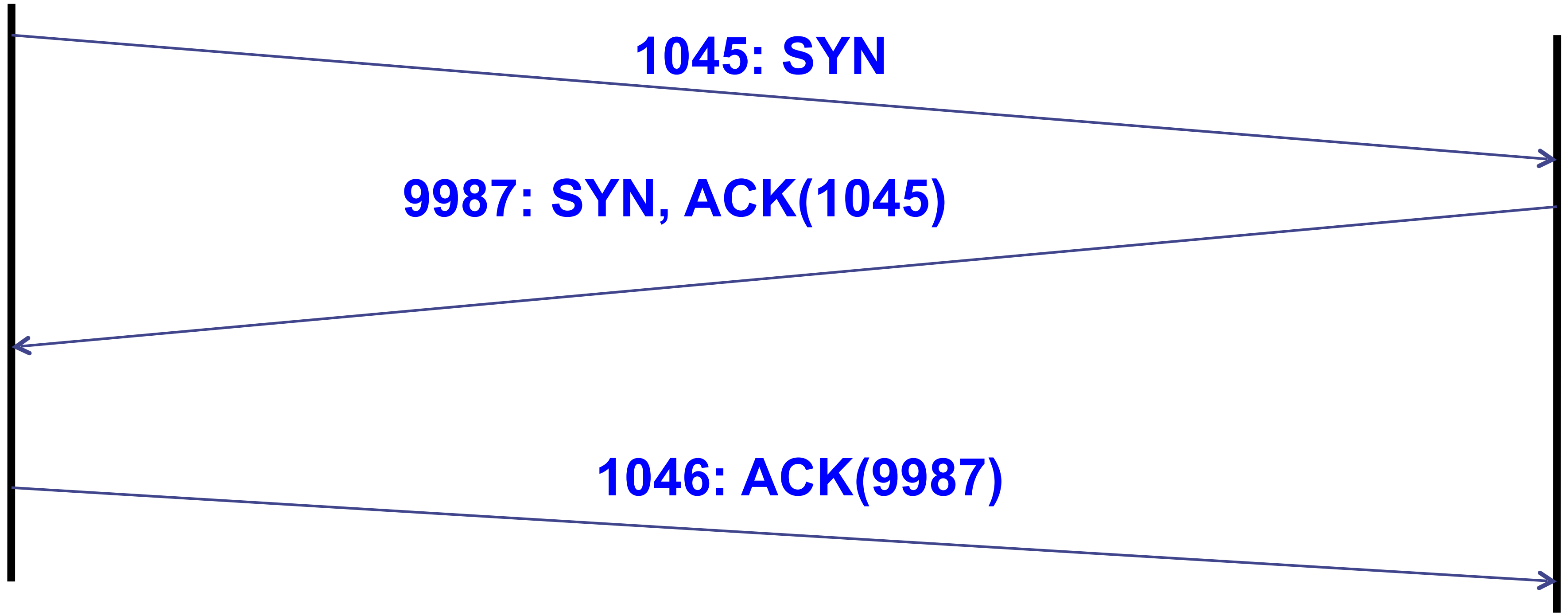
- To open a new connection:
  - 1 computer sends SYN (“Hi, lets talk”)
    - All messages have sequence numbers including SYN
    - First sequence number of a new connection is random
    - TCP sequence numbers by byte

# TCP



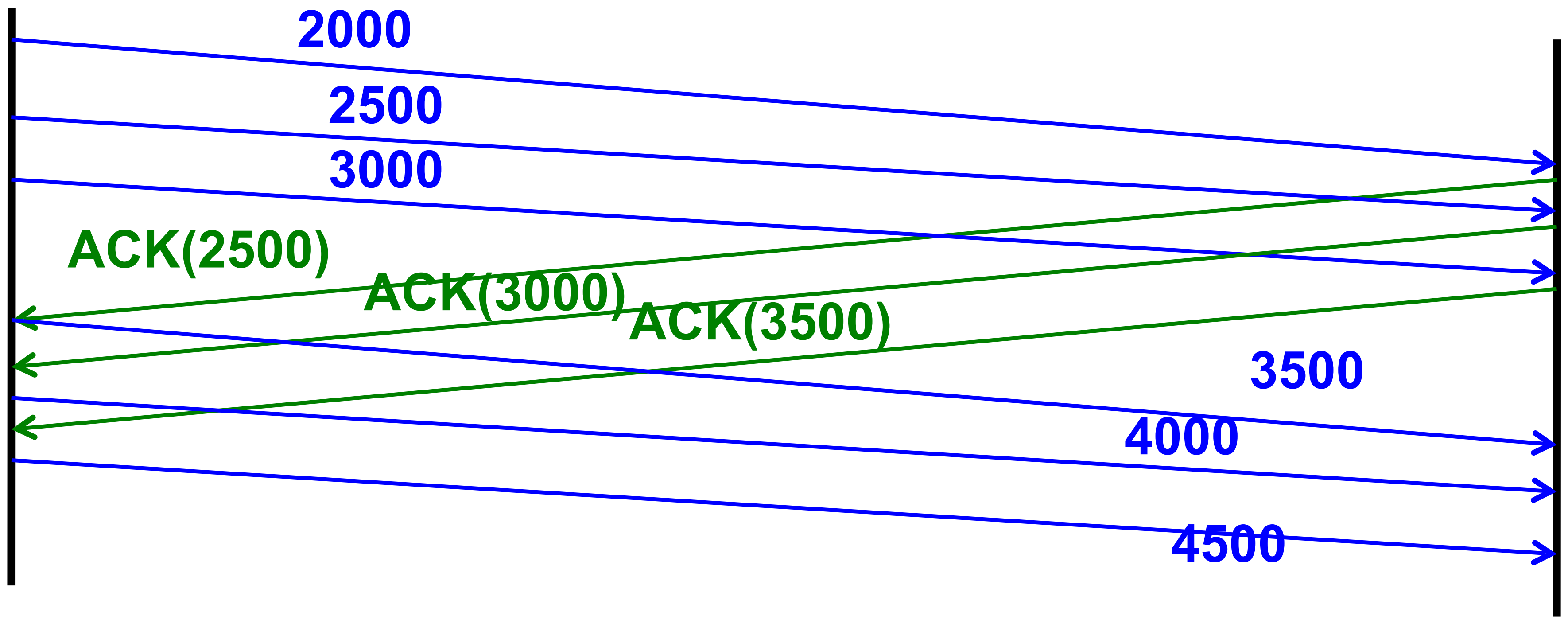
- Other computer
  - ACKs (Acknowledges) the message (says what sequence # it ACKs)
  - Also sends SYN “Hey sure, lets talk”

# TCP



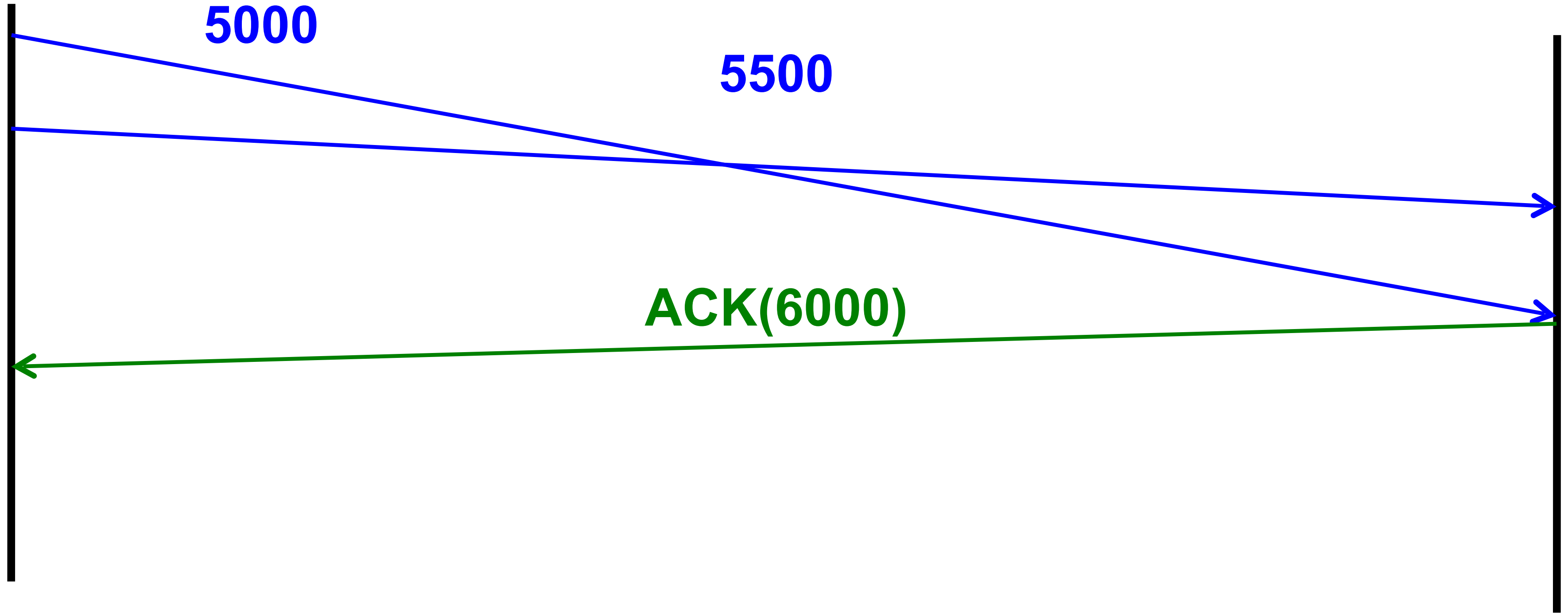
- First computer then ACKs this SYN
  - And probably sends data along with the ACK
- TCP control info (SYN, ACK, FIN): bits in TCP header
  - Packets can have multiple control bits on + carry data

# TCP: Normal operation



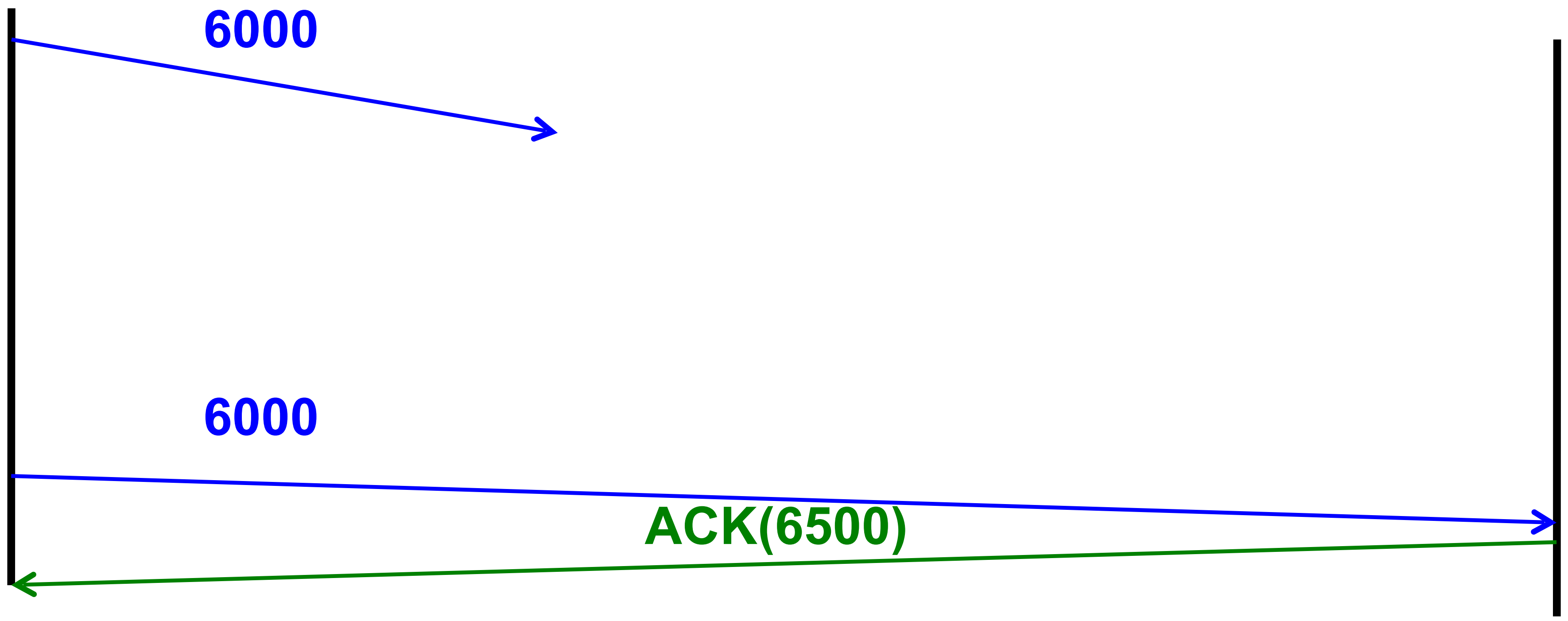
- Data going right in blue
- ACKS coming left in green
  - note: ACK #ed by expected next data
- Sliding window (flow control)
  - Limit amount of un-ACKed data at a time

# TCP: Re-ordered Data



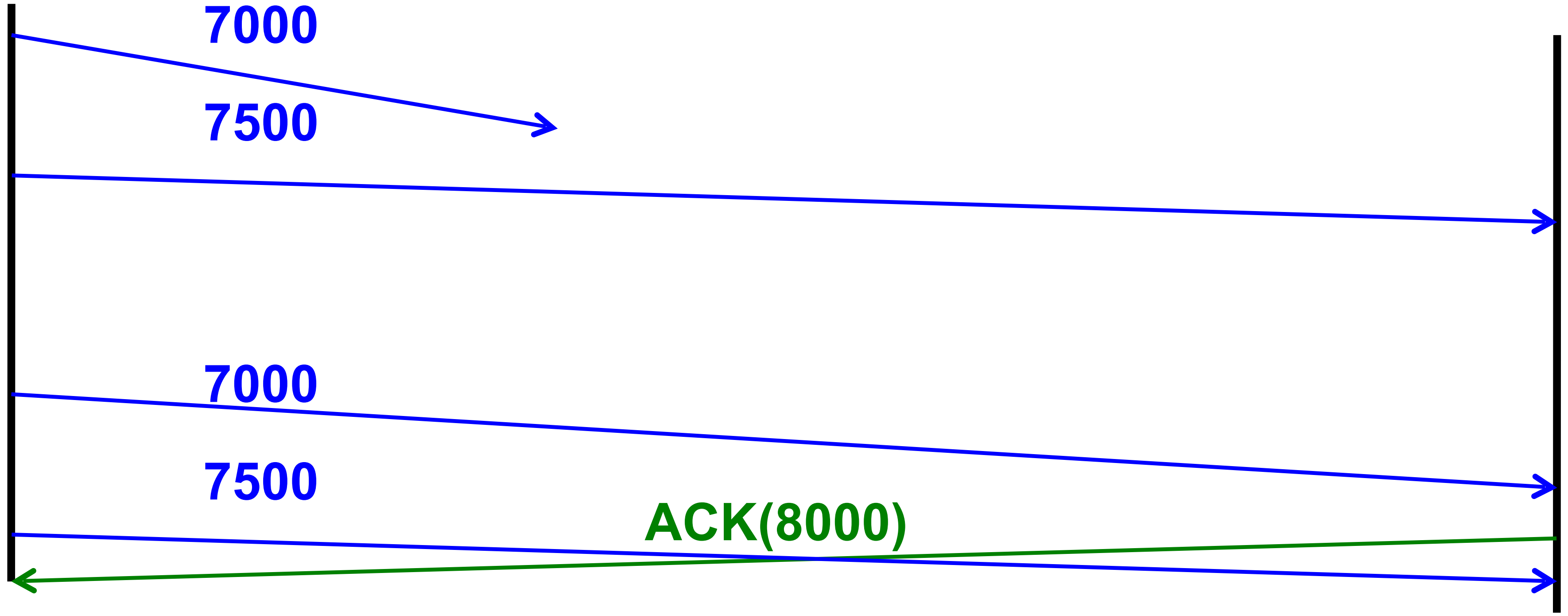
- Data may get re-ordered in network
  - One packet takes one route, another takes another
- TCP: no problem
  - Sender re-orders data properly
  - Sends ACK for as much data as it has

# TCP: Lost data



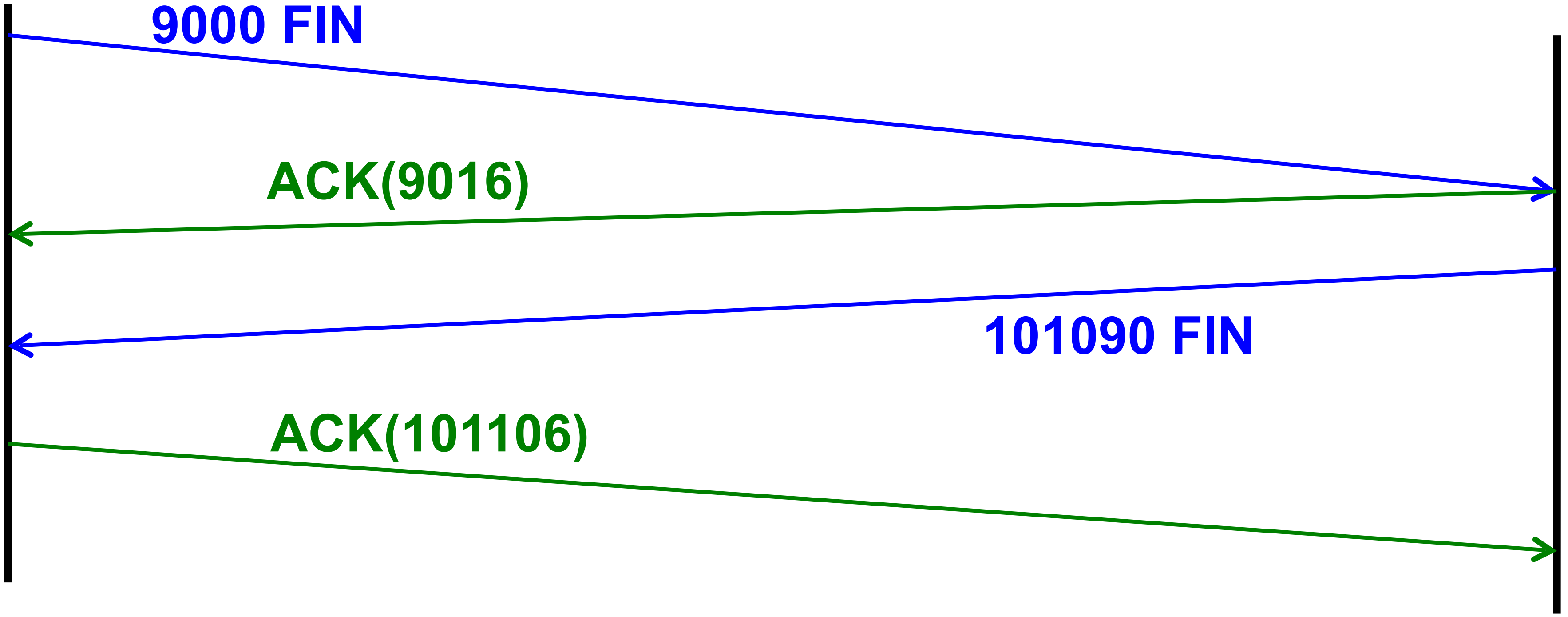
- Data may also get lost in the network
  - E.g., router is backlogged, can’t handle it has to drop from queue
- TCP will re-send un-ACKed data after a timeout

# TCP: Duplicate Data



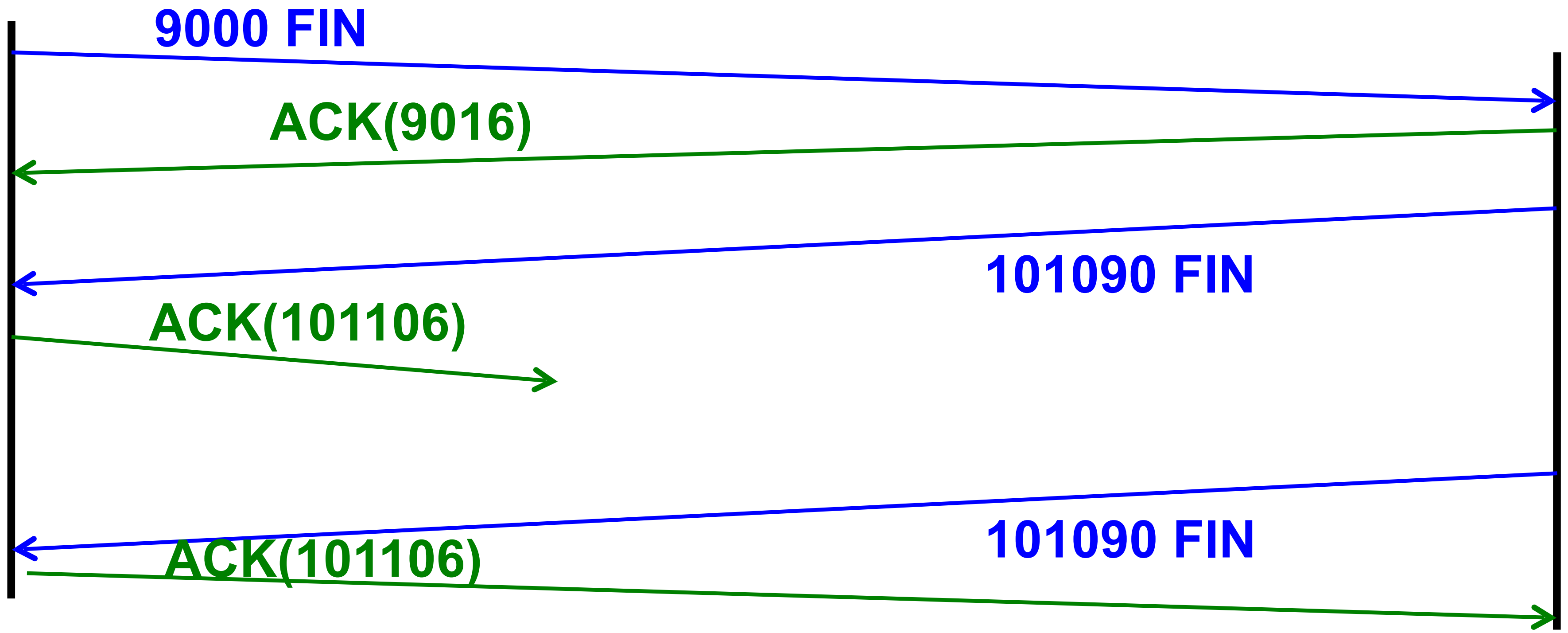
- Receiver may get duplicate data
  - 7000-7499 gets lost
  - But 7500-7599 arrives
  - Then sender re-sends both: no ACK for either (why not?)
  - No problem: receiver drops duplicate (can tell: sequence #s)

# TCP: Closing connection



- Connection closed with FIN message
  - Receiver ACKs
- Other side may close (with FIN) [typical]
  - Or remain open: can still send data
  - Side that closed cannot send, but should receive/ACK
  - FIN/ACK may be one message

# TCP: Closing connection



- What if ACK for FIN gets lost?
  - FIN gets retried… but other side expects connection is closed?
- TCP has a state to handle this
  - Connection expected to be closed, but resources/state still held
  - Times out if no activity (assumes ACK got through if no retry)

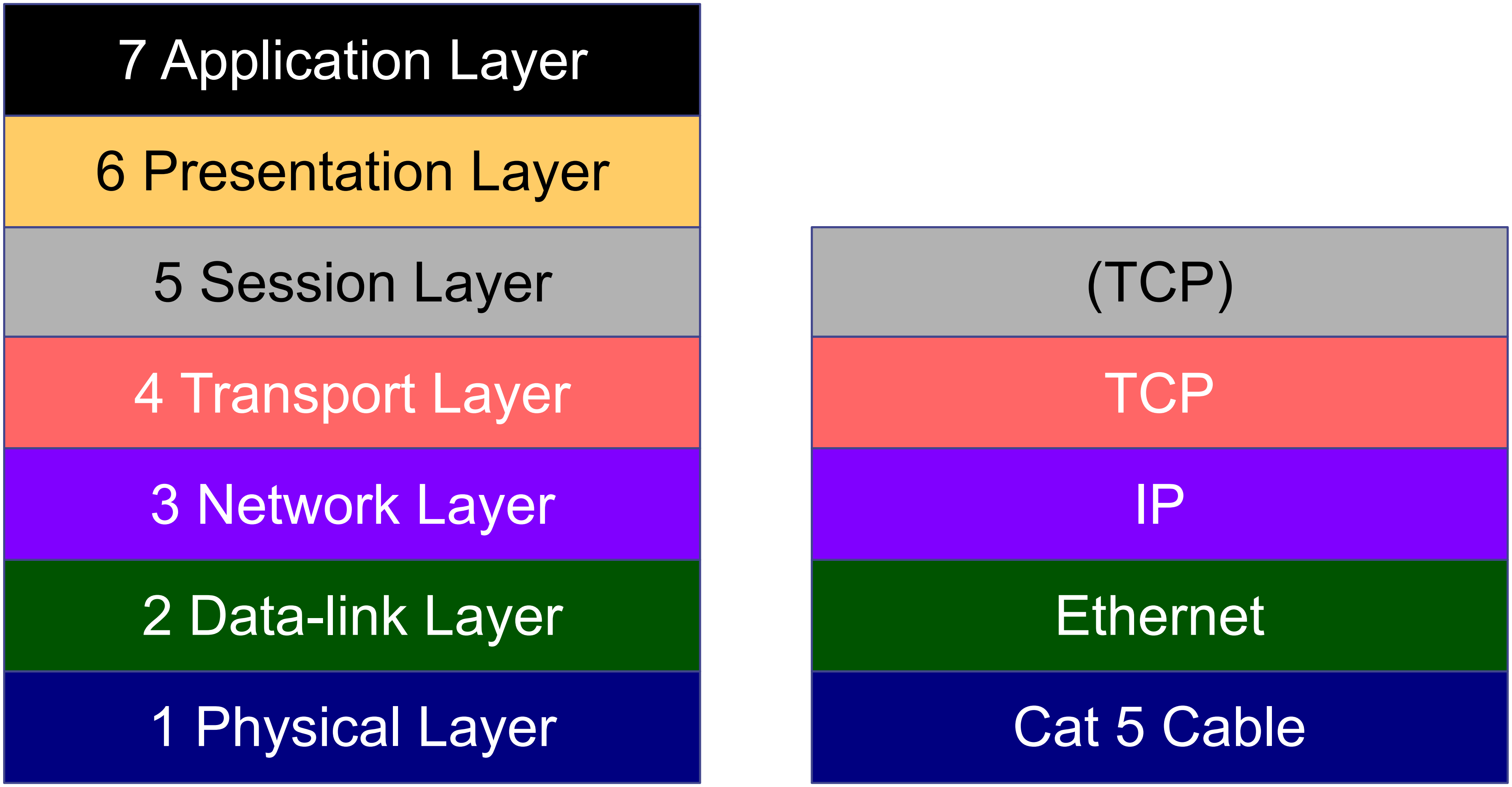
# Flow control: Sliding Window

- Problem:
  - Congestion -> dropped packets
  - Dropped packets -> Retries
  - Retries = duplicates of data -> More congestion

  - Vicious cycle…

- TCP implements **flow control** with a **sliding window**
  - Limitation of amount of un-ACKed data out at a time
  - Retry required?  Shrink window
    - Assumes congestion, tries to avoid it
  - No retries in a while?  Grow window back
    - Maybe it cleared up?

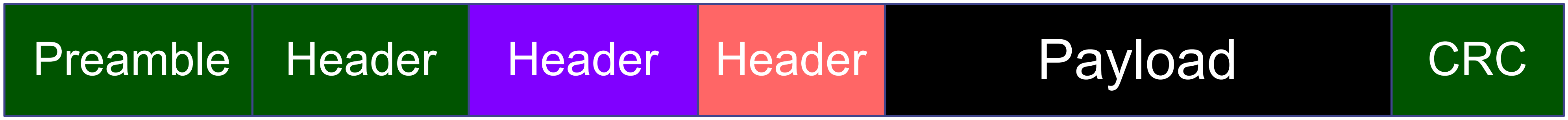
# 7-layer OSI model



| | |
|---|---|
| 7 Application Layer | |
| 6 Presentation Layer | |
| 5 Session Layer | (TCP) |
| 4 Transport Layer | TCP |
| 3 Network Layer | IP |
| 2 Data-link Layer | Ethernet |
| 1 Physical Layer | Cat 5 Cable |

- TCP: It's the coolest thing since memory got sliced into pages!

# Our messages so far



Preamble Header Header Header Payload CRC

- TCP header has
  - Source/Dest port
  - Sequence numbers
  - Control bits (SYN/ACK/FIN)
  - Check sum over data
  - Other stuff

# 6: Presentation Layer

- Responsible for data formats
  - Examples
    - Character encoding schemes
    - Serialization of objects

- We’re not really going to talk about it much

# 7: Application Layer

- Protocol specific to how applications want to communicate
  - Examples:
    - http
    - ftp
    - ssh
    - aim
    - SMTP
    - POP
    - …
- Again, not going into this much…

# 7-layer OSI model

| | |
|---|---|
| HTTP | HTTP |
| - | - |
| (TCP) | (TCP) |
| TCP | TCP |
| IP | IP |
| Ethernet | 802.11 |
| Cat 5 Cable | Wireless Radio |

- Flexibility Example: Wired vs Wireless
  - Change out two layers, rest stay the same

# 7-layer OSI model

| | |
|---|---|
| SMTP | SMTP |
| - | - |
| (TCP) | (TCP) |
| TCP | TCP |
| IP | IP |
| Ethernet | 802.11 |
| Cat 5 Cable | Wireless Radio |

- Flexibility Example: A different application on top of both

# Network programming

- Coding networking code in…
  - Java:  Look in java.net, start with Socket
  - C:
    - socket()
    - connect()
    - accept()
    - bind()
    - listen()

# When I say FIN, you say….?

**FIN**

# When I say FIN, you say….?

**ACK**

# Summary:

- Networking Overview
  - 7-layer model
  - Emphasis on IP (Layer 3) and TCP (Layers 4 and 5)

  - Not comprehensive, but…
  - You are now at least conversant enough to discuss the OSI stack at parties