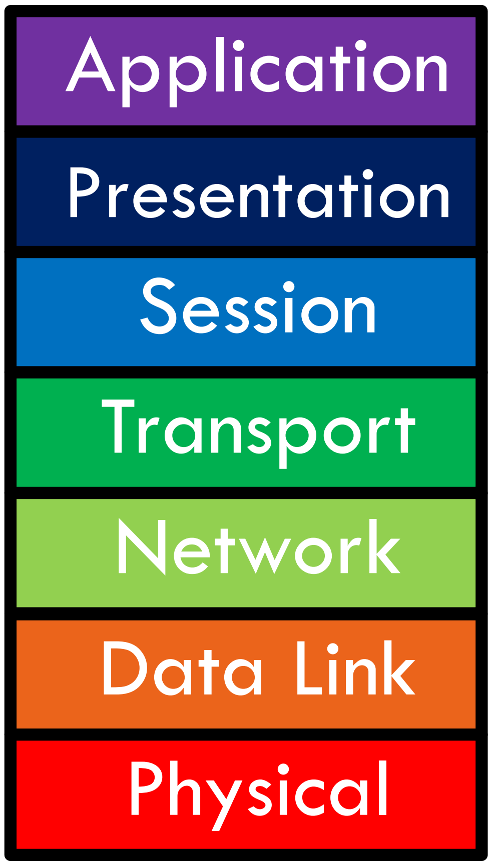
Computer Networks

Lecture 10: Network Layer – Part II

Based on slides from D. Choffnes Northeastern U. and P. Gill from StonyBrook University
Revised Autumn 2015 by S. Laki

Network Layer



- Function:
  - Route packets end-to-end on a network, through multiple hops
- Key challenge:
  - How to represent addresses
  - How to route packets
    - Scalability
    - Convergence

Intra-domain Routing Protocols

- Distance vector
  - Routing Information Protocol (RIP), based on Bellman-Ford
  - Routers periodically exchange reachability information with neighbors
- Link state
  - Open Shortest Path First (OSPF), based on Dijkstra
  - Each network periodically floods immediate reachability information to all other routers
  - Per router local computation to determine full routes

Outline

- Distance Vector Routing
  - RIP
- Link State Routing
  - OSPF
  - IS-IS

Distance Vector Routing

- What is a distance vector?
  - Current best known cost to reach a destination
- Idea: exchange vectors among neighbors to learn about lowest cost paths

DV Table at Node C

Destination	Cost
A	7
B	1
D	2
E	5
F	1

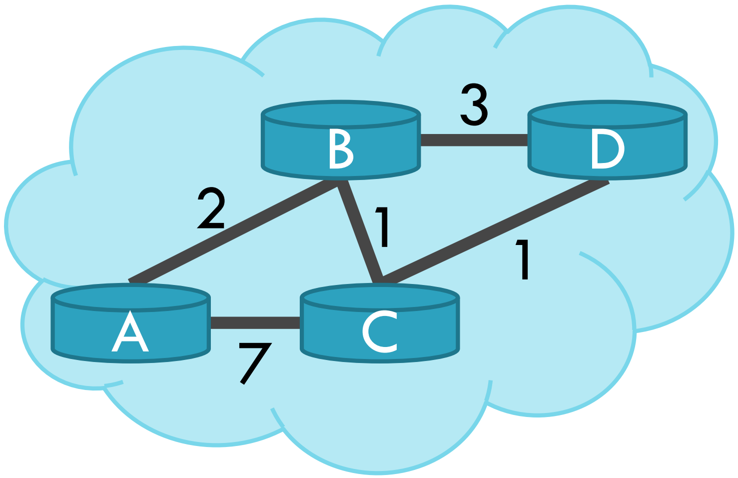
- No entry for C
- Initially, only has info for immediate neighbors
  - Other destinations cost = ∞
- Eventually, vector is filled

- Routing Information Protocol (RIP)

Distance Vector Routing Algorithm

1. Wait for change in local link cost or message from neighbor
2. Recompute distance table
3. If least cost path to any destination has changed, notify neighbors

Distance Vector Initialization



Node A

Dest.	Cost	Next
B	2	B
C	7	C
D	∞	

Node B

Dest.	Cost	Next
A	2	A
C	1	C
D	3	D

1. **Initialization:**
2. **for all** neighbors *V* **do**
3. **if** *V* adjacent to *A*
4. D(*A*, *V*) = c(*A*,*V*);
5. **else**
6. D(*A*, *V*) = ∞;

…

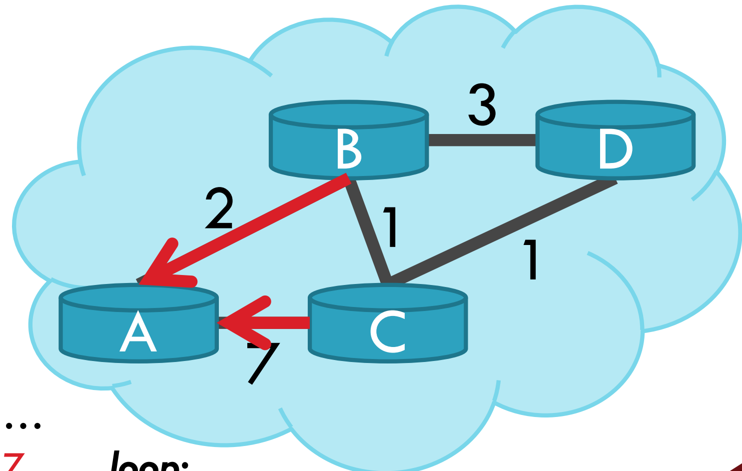
Node C

Dest.	Cost	Next
A	7	A
B	1	B
D	1	D

Node D

Dest.	Cost	Next
A	∞	
B	3	B
C	1	C

Distance Vector: 1st Iteration




```
…
loop:
…
else if (update D(V, Y) recei…
  for all desti…
    if (destina…
      D(A,Y) = …
    else
      D(A, Y) = …
            mi…
            D(A, V) + D(V, Y));
if (there is a new min. for dest. Y)
  send D(A, Y) to all neighbors
forever
```

Node A

Dest.	Cost	Next
B	2	B
C	3	B
D	5	B

Node B

Dest.	Cost	Next
A	2	A
C	1	C
D	2	C

D(A,C) … (A,C), D(A,B)+D(B,C))

D(A,D) = min(D(A,D), D(A,B)+D(B,D))

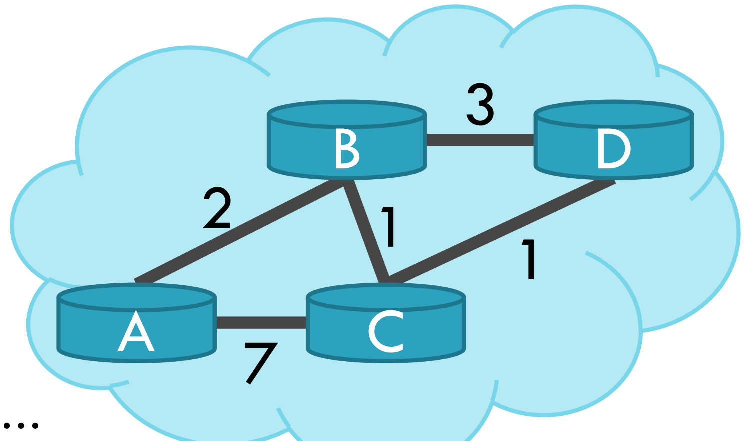
= min(8, 3 + 3) = 5

Dest.	Cost	Next
[illegible]	3	B
B	1	B
D	1	D

…e D

Dest.	Cost	Next
[illegible]	4	B
B	3	B
C	1	C

Distance Vector: End of 3rd Iteration



Node A

Dest.	Cost	Next
B	2	B
C	3	B
D	4	B

Node B

Dest.	Cost	Next
A	2	A
C	1	C
D	2	C

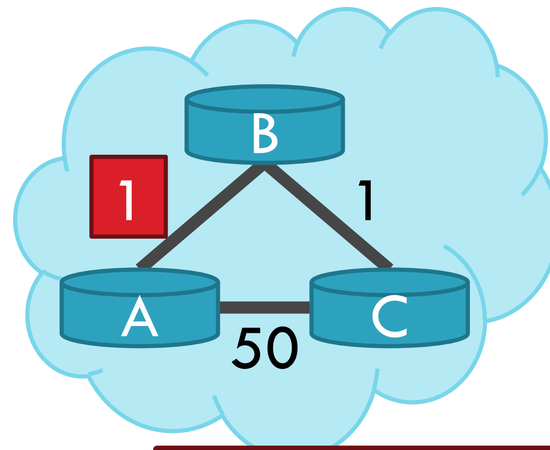
```
…
loop
…
else
 fo
  i

  else
   D(A, Y) =
         min(D(A, Y),
         D(A, V) + D(V, Y));
if (there is a new min. for dest. Y)
 send D(A, Y) to all neighbors
forever
```

- Nothing changes, algorithm terminates
- Until something changes…

Dest.	Cost	Next
A	3	B
B	1	B
D	1	D

Dest.	Cost	Next
A	4	C
B	2	C
C	1	C

```
loop:
  wait (link cost update or update message)
  if (c(A,V) changes by d)
    for all destinations Y through V do
      D(A,Y) = D(A,Y) + d
  else if (update D(V, Y) received from V)
    for all destinations Y do
      if (destination Y through V)
        D(A,Y) = D(A,V) + D(V, Y);
      else
        D(A, Y) = min(D(A, Y), D(A, V) + D(V, Y));
```



Link Co
Algor

Good news travels fast

Algorithm
erminates

Node B

D	C	N
A	4	A
C	1	B

D	C	N
A	1	A
C	1	B

D	C	N
A	1	A
C	1	B

D	C	N
A	1	A
C	1	B

Node C

D	C	N
A	5	B
B	1	B

D	C	N
A	5	B
B	1	B

D	C	N
A	2	B
B	1	B

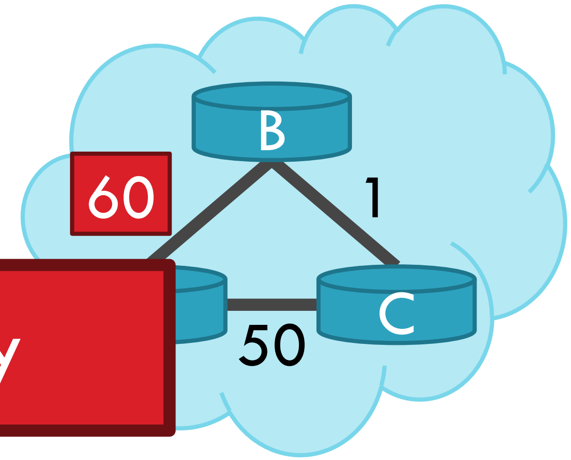
D	C	N
A	2	B
B	1	B

Time

Count to Infinity Problem

- Node B knows D(C, A) = 5
- However, B does not know the path is C
- Thus, D(B,A

Bad news travels slowly



Node B

D	C	N
A	4	A
C	1	B

D		N
A	**6**	**C**
C	1	B

D	C	N
A	6	C
C	1	B

D	C	N
A	**8**	C
C	1	B

Node C

D	C	N
A	5	B
B	1	B

D	C	N
A	5	B
B	1	B

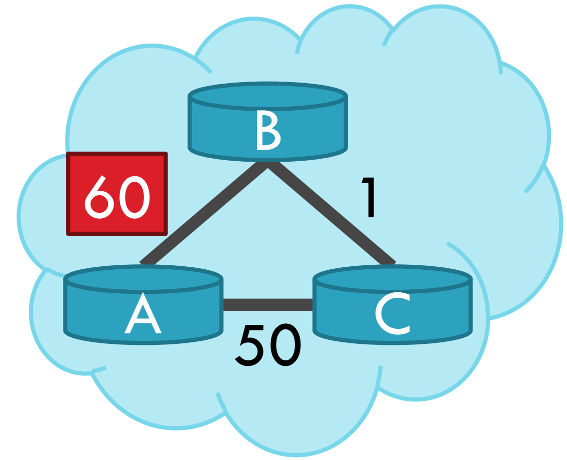
D	C	N
A	**7**	B
B	1	B

D	C	N
A	7	B
B	1	B

Time

Poisoned Reverse

- If C routes through B to get to A
  - C tells B that $D(C, A) = \infty$
  - Thus, B won't route to A via C



Node B

D	C	N
A	4	A
C	1	B

D	C	N
A	**60**	A
C	1	B

D	C	N
A	60	A
C	1	B

D	C	N
A	**51**	**C**
C	1	B

Node C

D	C	N
A	5	B
B	1	B

D	C	N
A	5	B
B	1	B

D	C	N
A	**50**	**A**
B	1	B

D	C	N
A	50	A
B	1	B

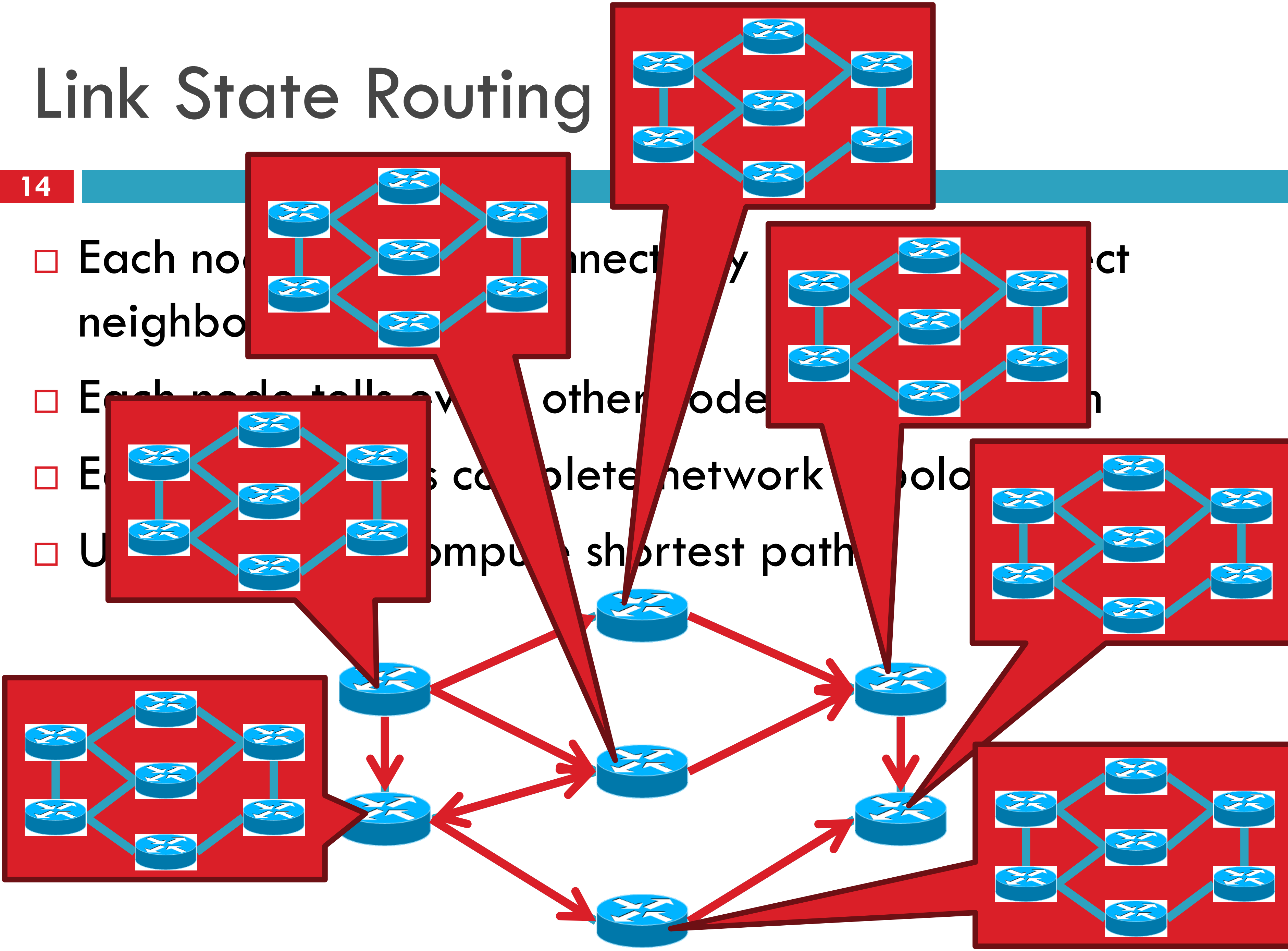
Time

Outline

- Distance Vector Routing
  - RIP
- Link State Routing
  - OSPF
  - IS-IS

Link State Routing

- Each no… nnect… y … ct neighbo…
- Each node tells ev… other node… n
- E… s co… plete network … olo…
- U… ompu… e shortest path



Flooding Details

- Each node periodically generates Link State Packet
  - ID of node generating the LSP
  - List of direct neighbors and costs
  - Sequence number (64-bit, assumed to never wrap)
  - Time to live
- Flood is reliable (ack + retransmission)
- Sequence number "versions" each LSP
- Receivers flood LSPs to their own neighbors
  - Except whoever originated the LSP
- LSPs also generated when link states change

OSPF vs. IS-IS

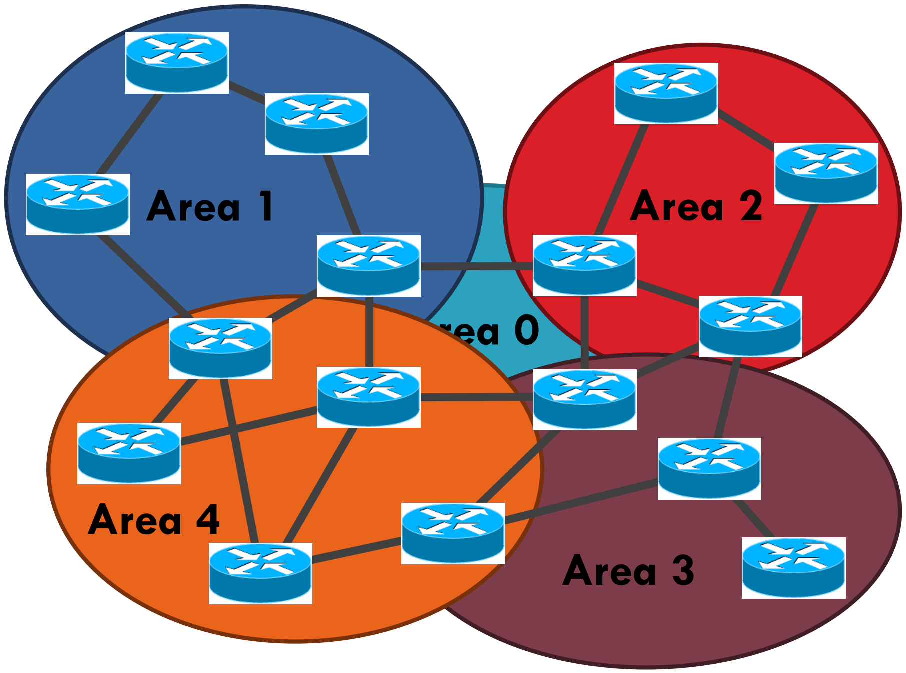
- Two different implementations of link-state routing

OSPF	IS-IS
Favored by companies, datacenters	Favored by ISPs
More optional features	Less "chatty" • Less network overhead • Supports more devices
Built on top of IPv4 • LSAs are sent via IPv4 • OSPFv3 needed for IPv6	Not tied to IP • Works with IPv4 or IPv6

Different Organizational Structure

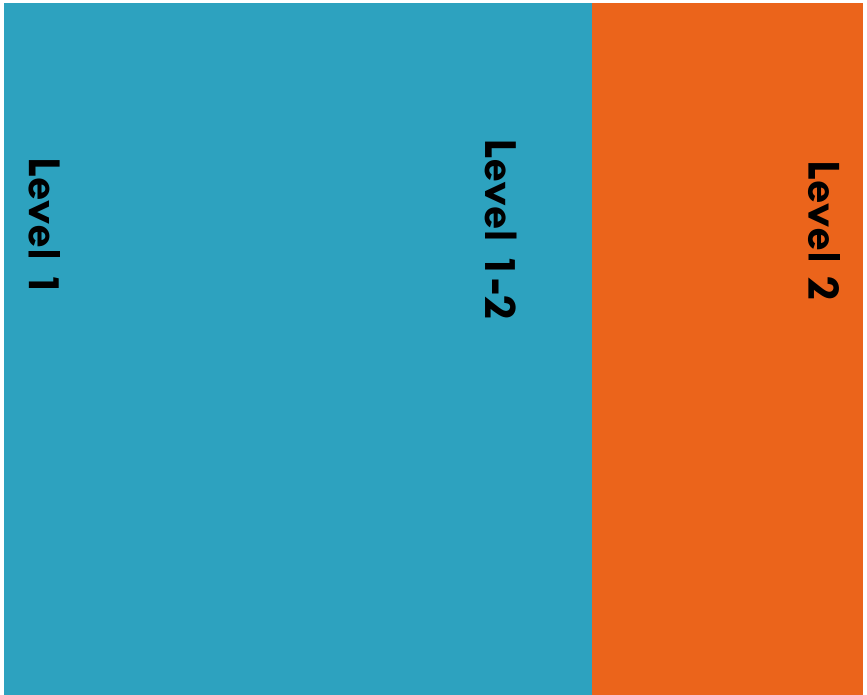
OSPF

- Organized around overlapping areas
- Area 0 is the core network



IS-IS

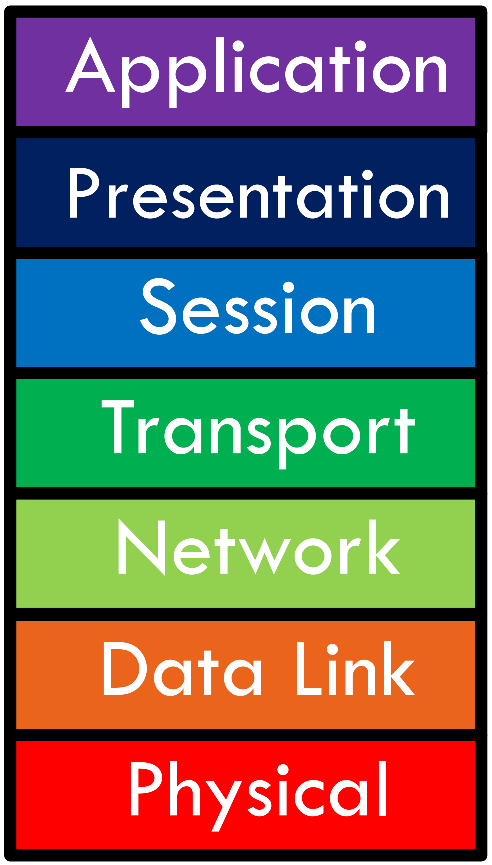
- Organized as a 2-level hierarchy
- Level 2 is the backbone



Network Layer, Control Plane

- Function:
  - Set up routes between networks
- Key challenges:
  - Implementing provider policies
  - Creating stable paths

Data Plane

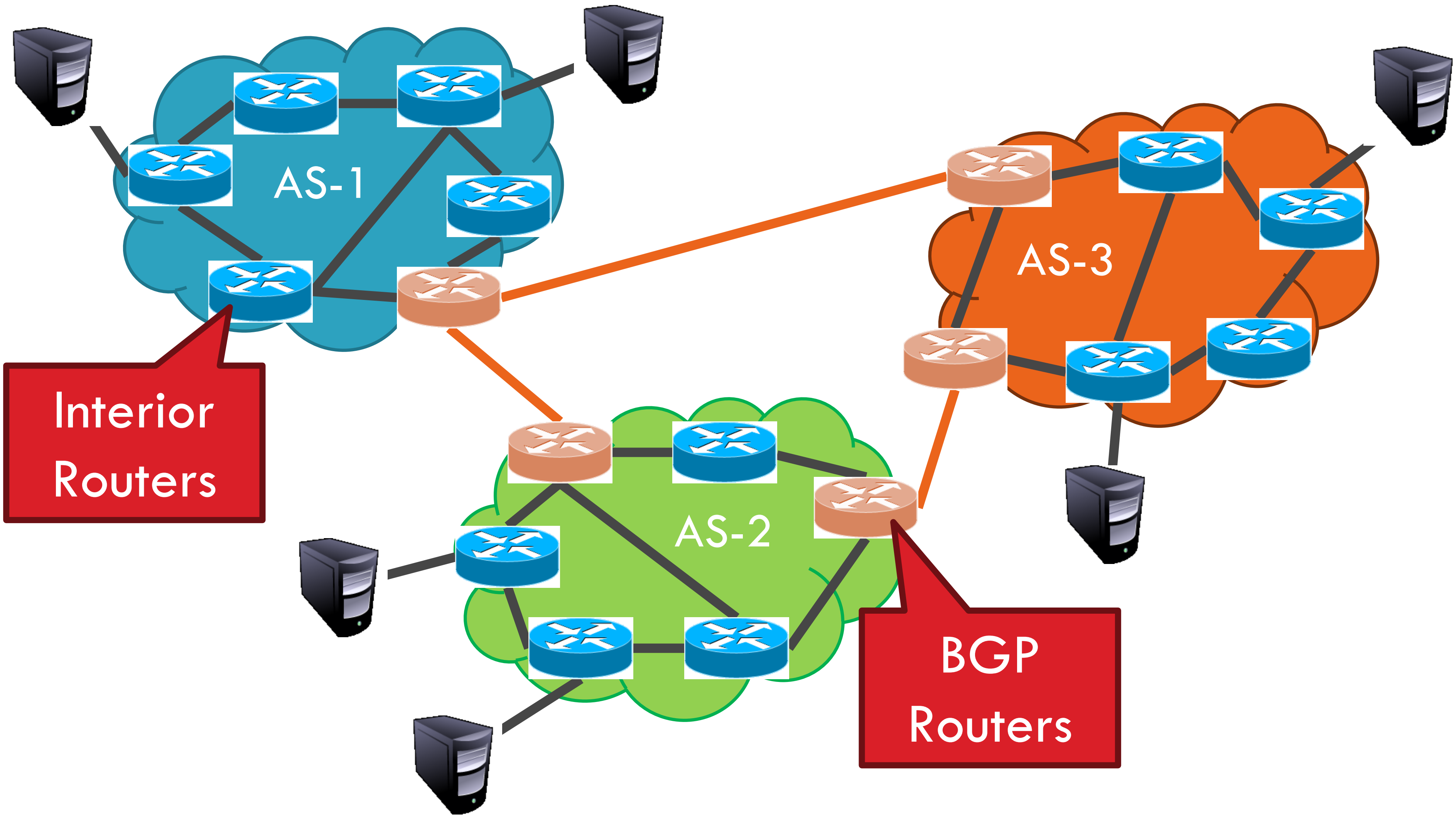


Control Plane

Outline

- BGP Basics
- Stable Paths Problem
- BGP in the Real World
- Debugging BGP Path Problems

ASs, Revisited



AS Numbers

- Each AS identified by an ASN number
  - 16-bit values (latest protocol supports 32-bit ones)
  - 64512 – 65535 are reserved
- Currently, there are ~ 40000 ASNs
  - AT&T: 5074, 6341, 7018, …
  - Sprint: 1239, 1240, 6211, 6242, …
  - ELTE: 2012
  - Google 15169, 36561 (formerly YT), + others
  - Facebook 32934
  - North America ASs → ftp://ftp.arin.net/info/asn.txt

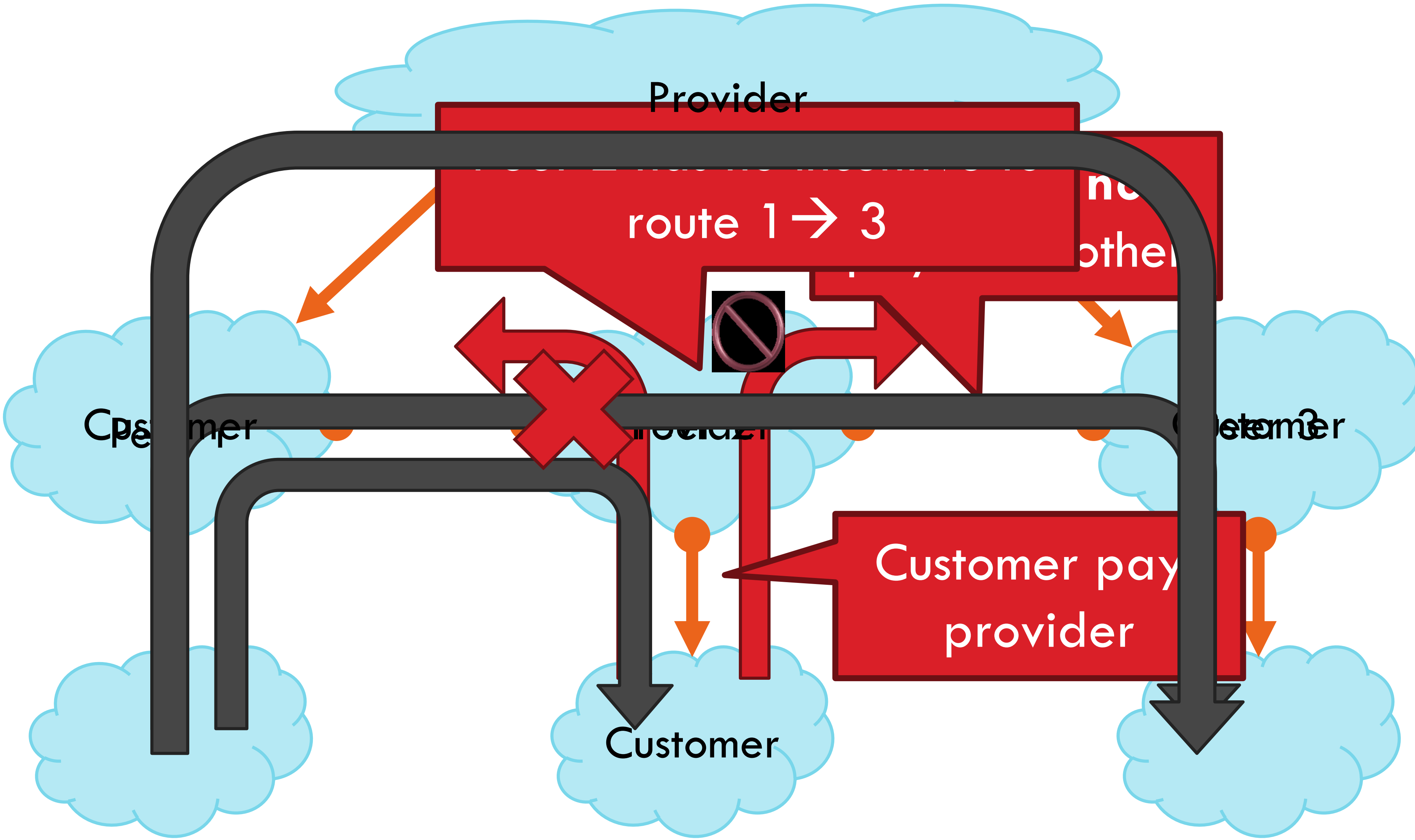
Inter-Domain Routing

- Global connectivity is at stake!
  - Thus, all ASs must use the same protocol
  - Contrast with intra-domain routing
- What are the requirements?
  - Scalability
  - Flexibility in choosing routes
    - Cost
    - Routing around failures
- Question: link state or distance vector?
  - Trick question: BGP is a path vector protocol

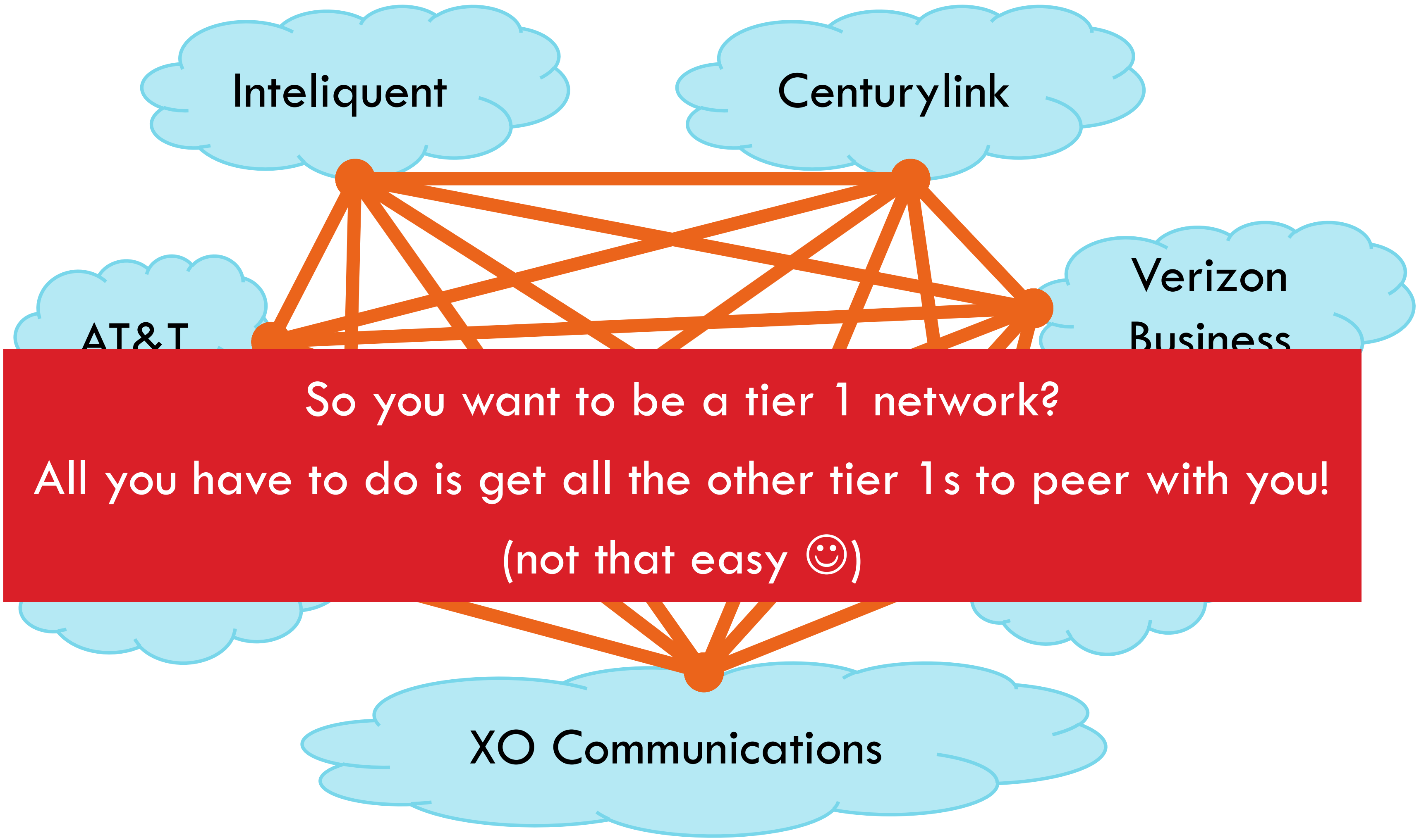
BGP

- Border Gateway Protocol
  - De facto inter-domain protocol of the Internet
  - Policy based routing protocol
  - Uses a Bellman-Ford path vector protocol
- Relatively simple protocol, but…
  - Complex, manual configuration
  - Entire world sees advertisements
    - Errors can screw up traffic globally
  - Policies driven by economics
    - How much $$$ does it cost to route along a given path?
    - Not by performance (e.g. shortest paths)

BGP Relationships



Tier-1 ISP Peering

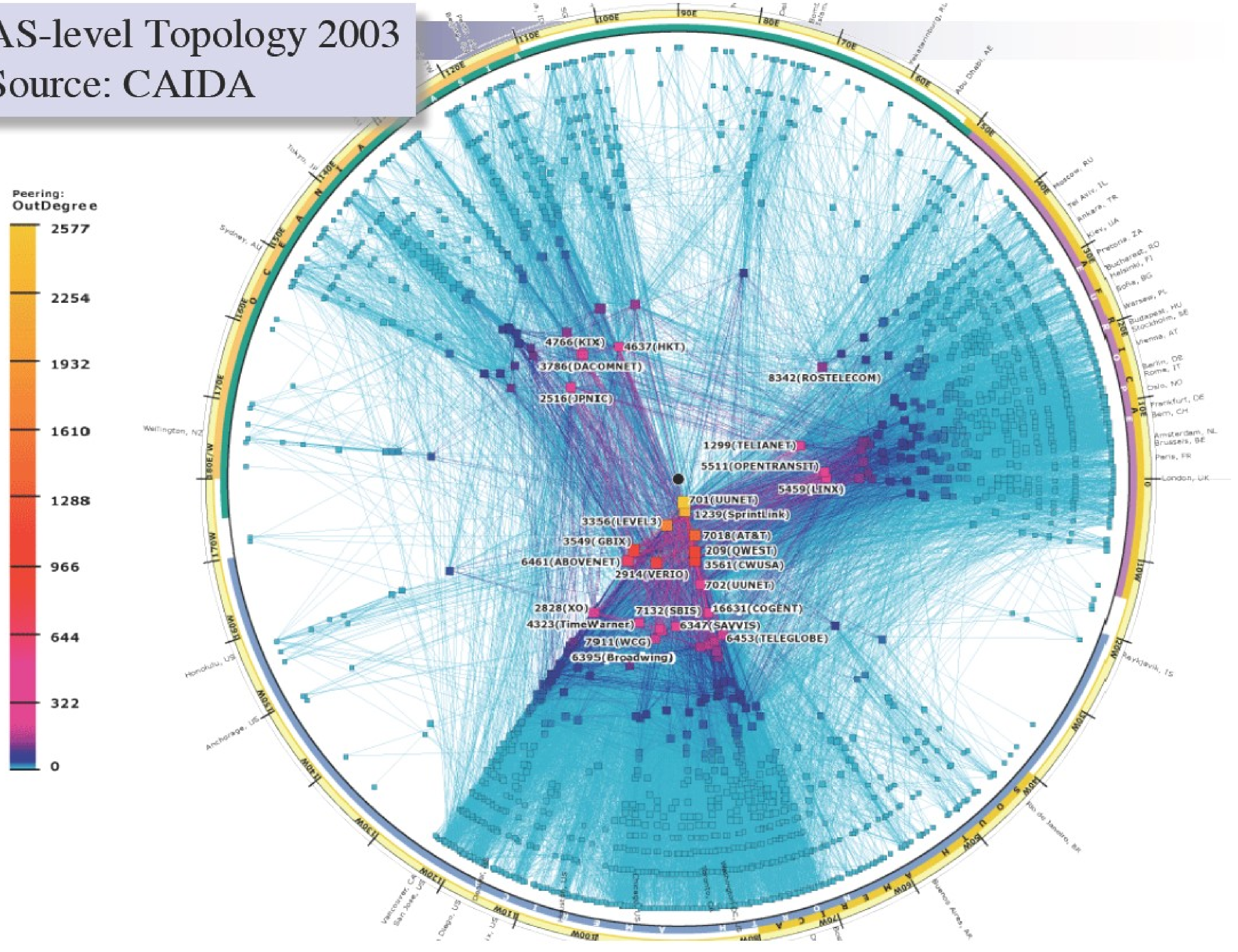


So you want to be a tier 1 network?

All you have to do is get all the other tier 1s to peer with you!

(not that easy ☺)

AS-level Topology 2003
Source: CAIDA



Peering Wars

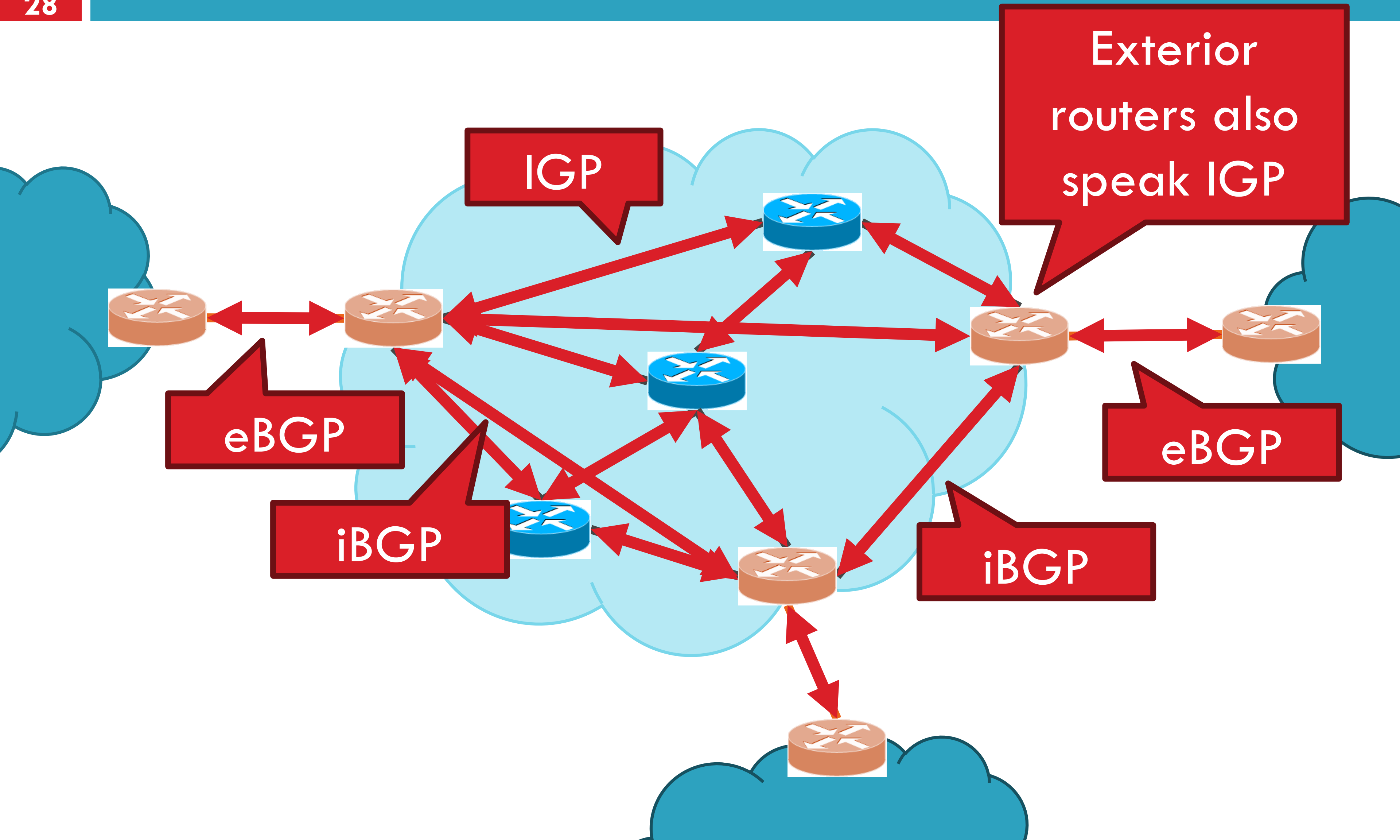
Peer	Don't Peer
Reduce upstream costs	You would rather have

Peering struggles in the ISP world are extremely contentious agreements are usually confidential

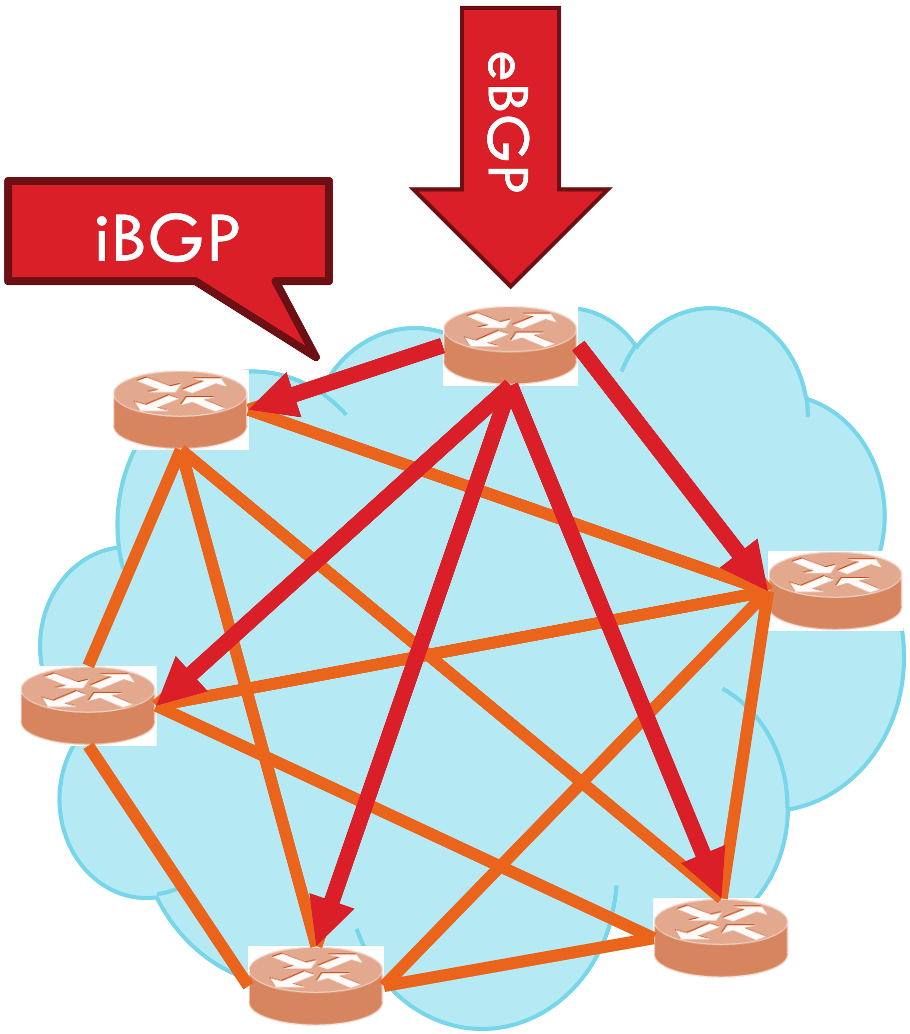
Example: If you are a customer of my peer why should I peer with you? You should pay me too!

Incentive to keep relationships private!

Two Types of BGP Neighbors



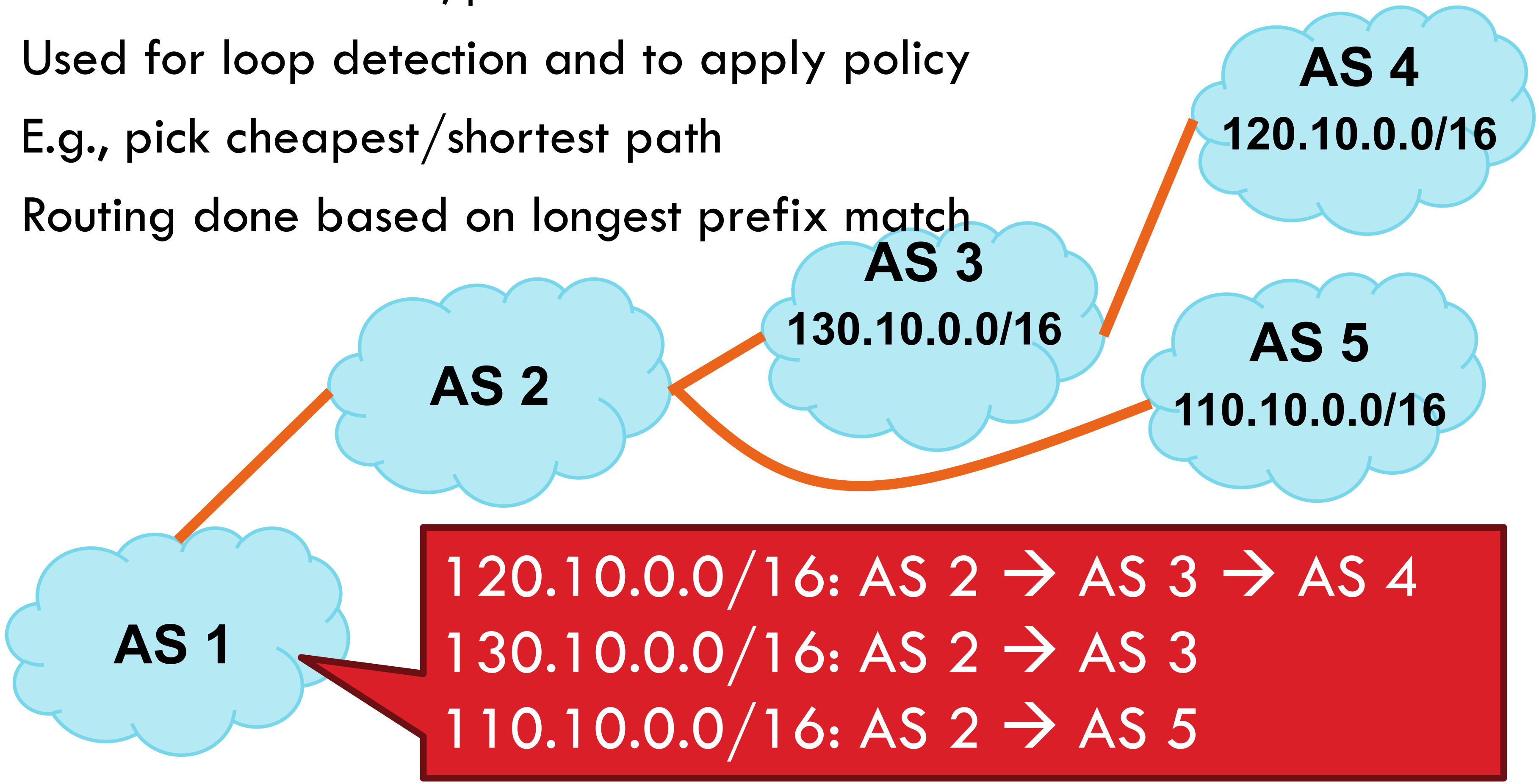
Full iBGP Meshes



- Question: why do we need iBGP?
  - OSPF does not include BGP policy info
  - Prevents routing loops within the AS
- iBGP updates do not trigger announcements

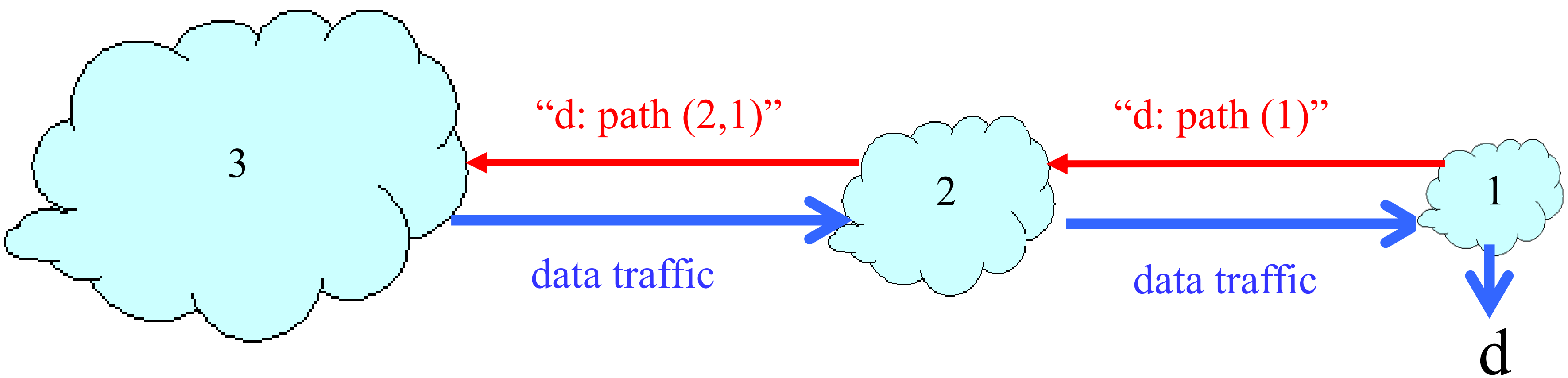
Path Vector Protocol

- AS-path: sequence of ASs a route traverses
  - Like distance vector, plus additional information
- Used for loop detection and to apply policy
- E.g., pick cheapest/shortest path
- Routing done based on longest prefix match



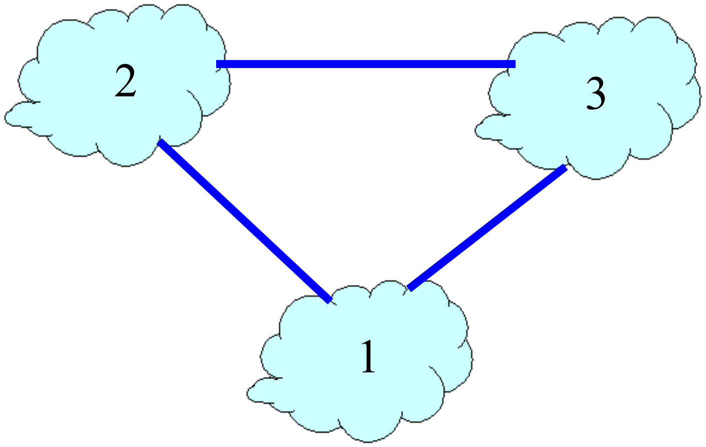
Path-Vector Routing

- Extension of distance-vector routing
  - Support flexible routing policies
  - Avoid count-to-infinity problem
- Key idea: advertise the entire path
  - Distance vector: send *distance metric* per dest d
  - Path vector: send the *entire path* for each dest d



Flexible Policies

- Each node can apply local policies
  - Path selection: Which path to use?
  - Path export: Which paths to advertise?
- Examples
  - Node 2 may prefer the path "2, 3, 1" over "2, 1"
  - Node 1 may not let node 3 hear the path "1, 2"

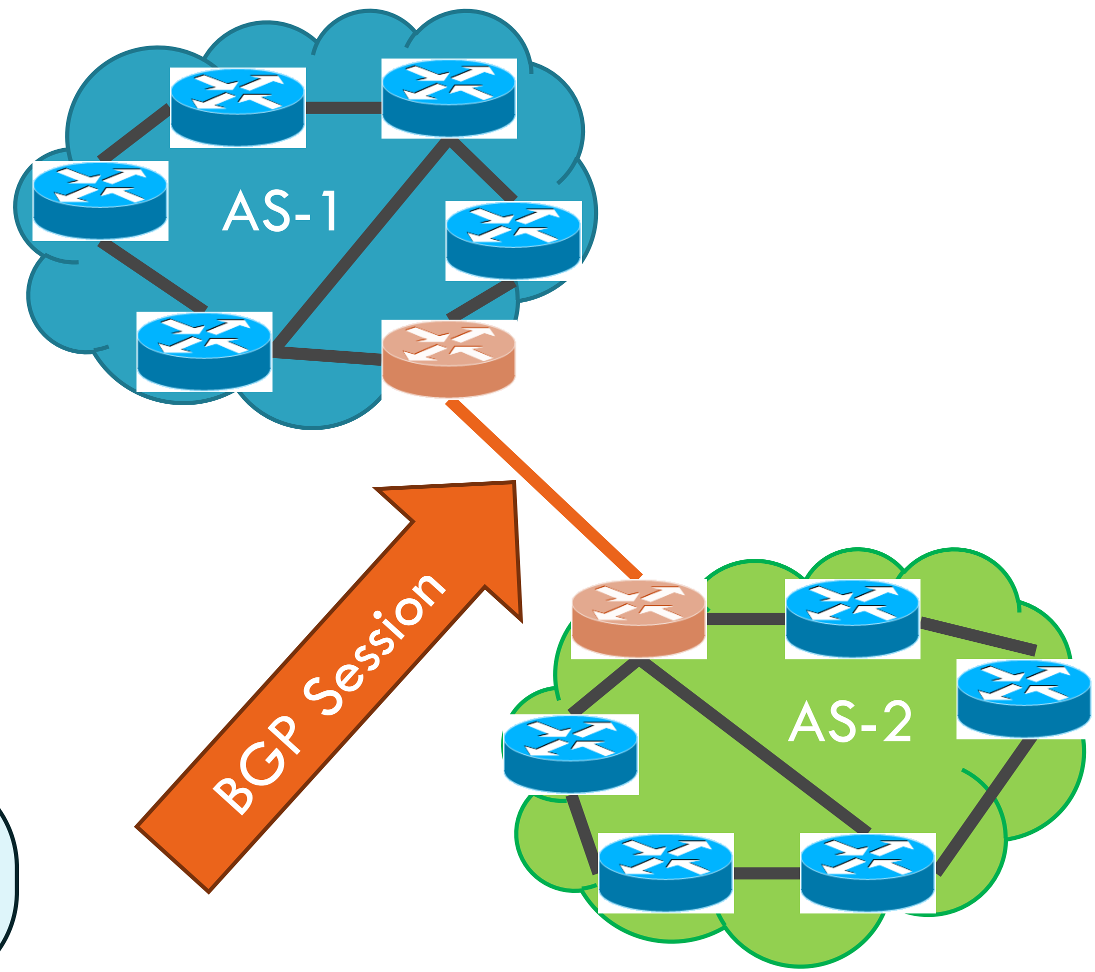


BGP Operations (Simplified)

Establish session on TCP port 179

Exchange active routes

Exchange incremental updates



Four Types of BGP Messages

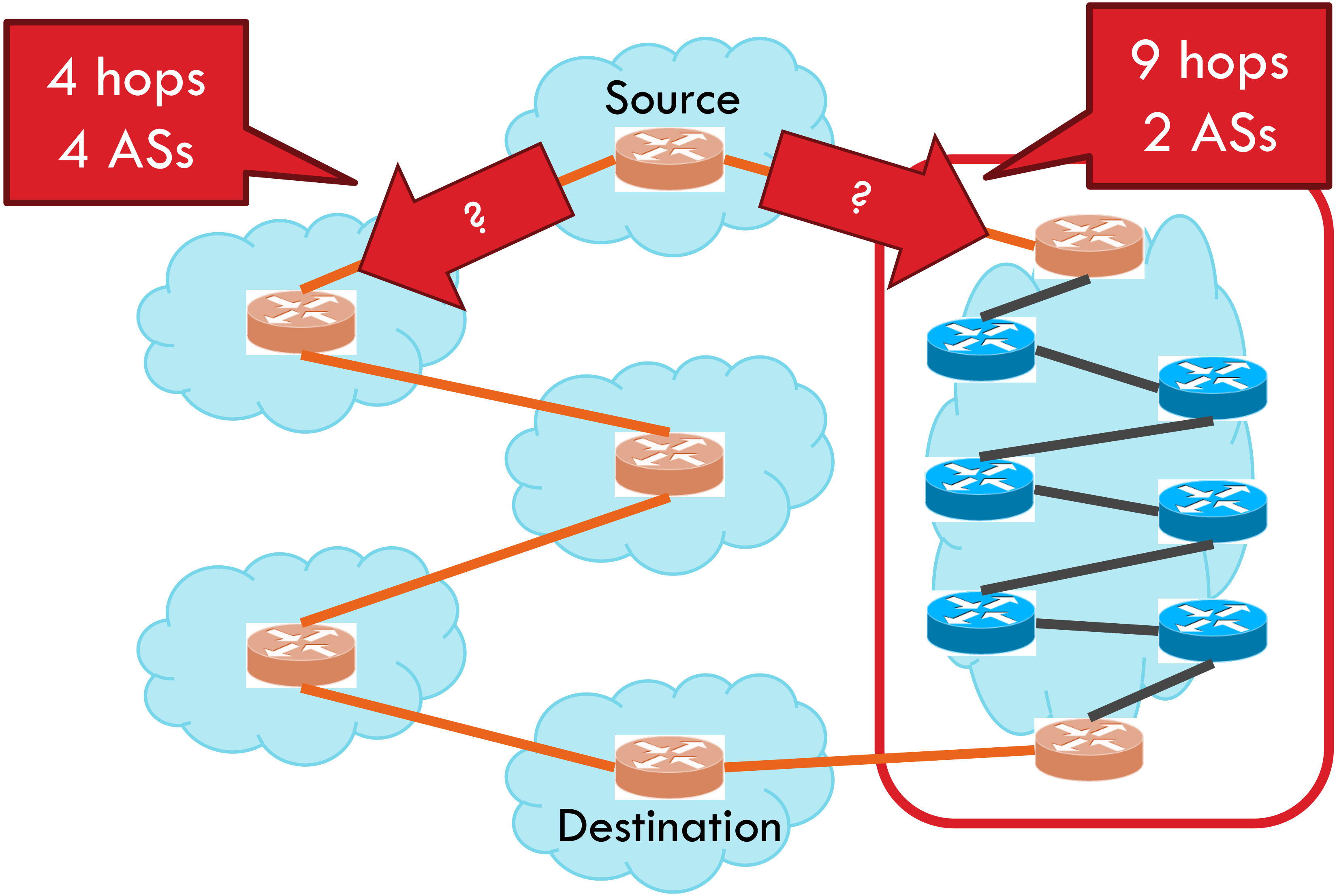
- Open: Establish a peering session.
- Keep Alive: Handshake at regular intervals.
- Notification: Shuts down a peering session.
- Update: Announce new routes or withdraw previously announced routes.

announcement = IP prefix + attributes values

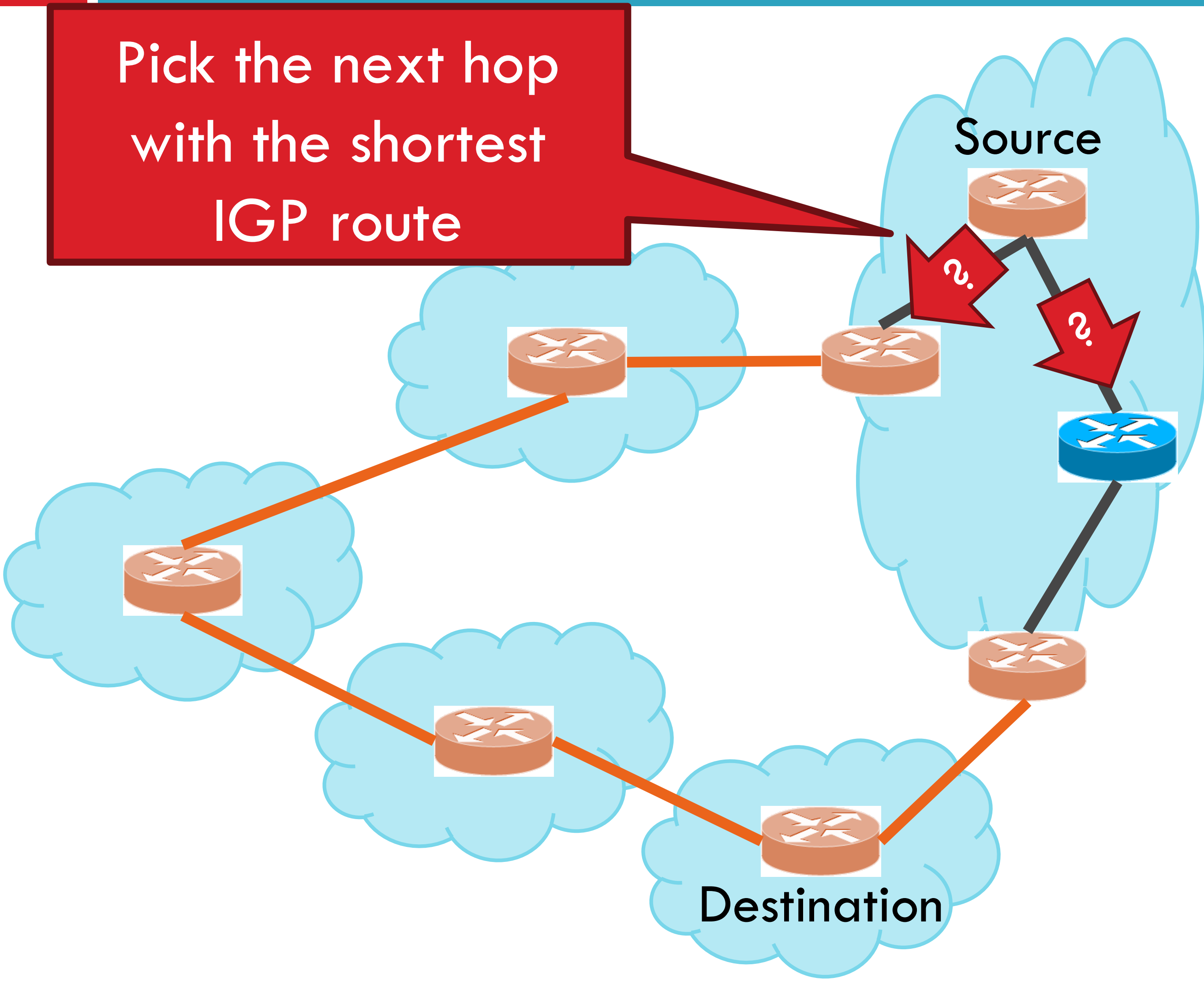
BGP Attributes

- Attributes used to select "best" path
  - LocalPref
    - Local preference policy to choose most preferred route
    - Overrides default fewest AS behavior
  - Multi-exit Discriminator (MED)
    - Specifies path for external traffic destined for an internal network
    - Chooses peering point for your network
  - Import Rules
    - What route advertisements do I accept?
  - Export Rules
    - Which routes do I forward to whom?

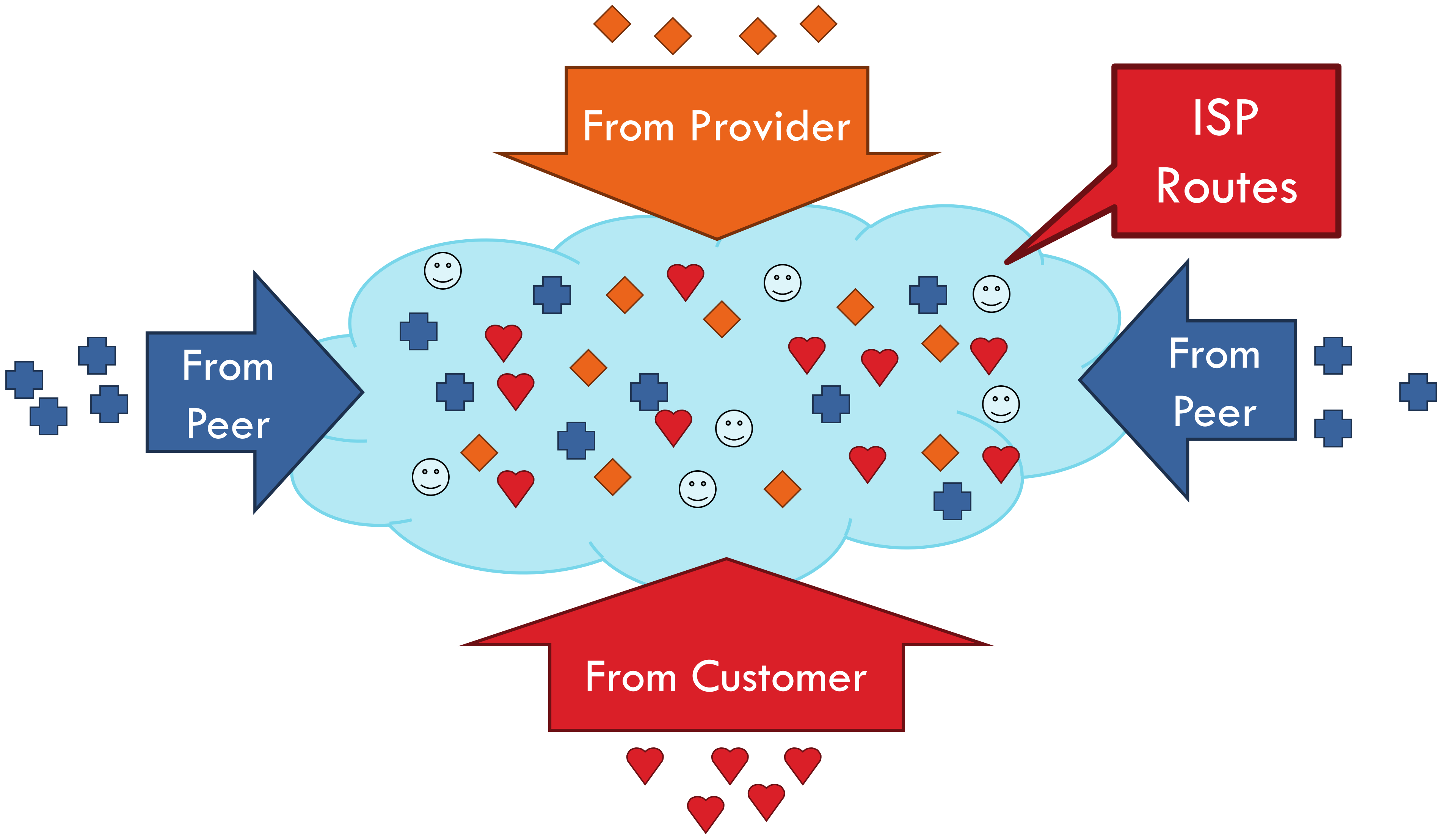
Shortest AS Path != Shortest Path



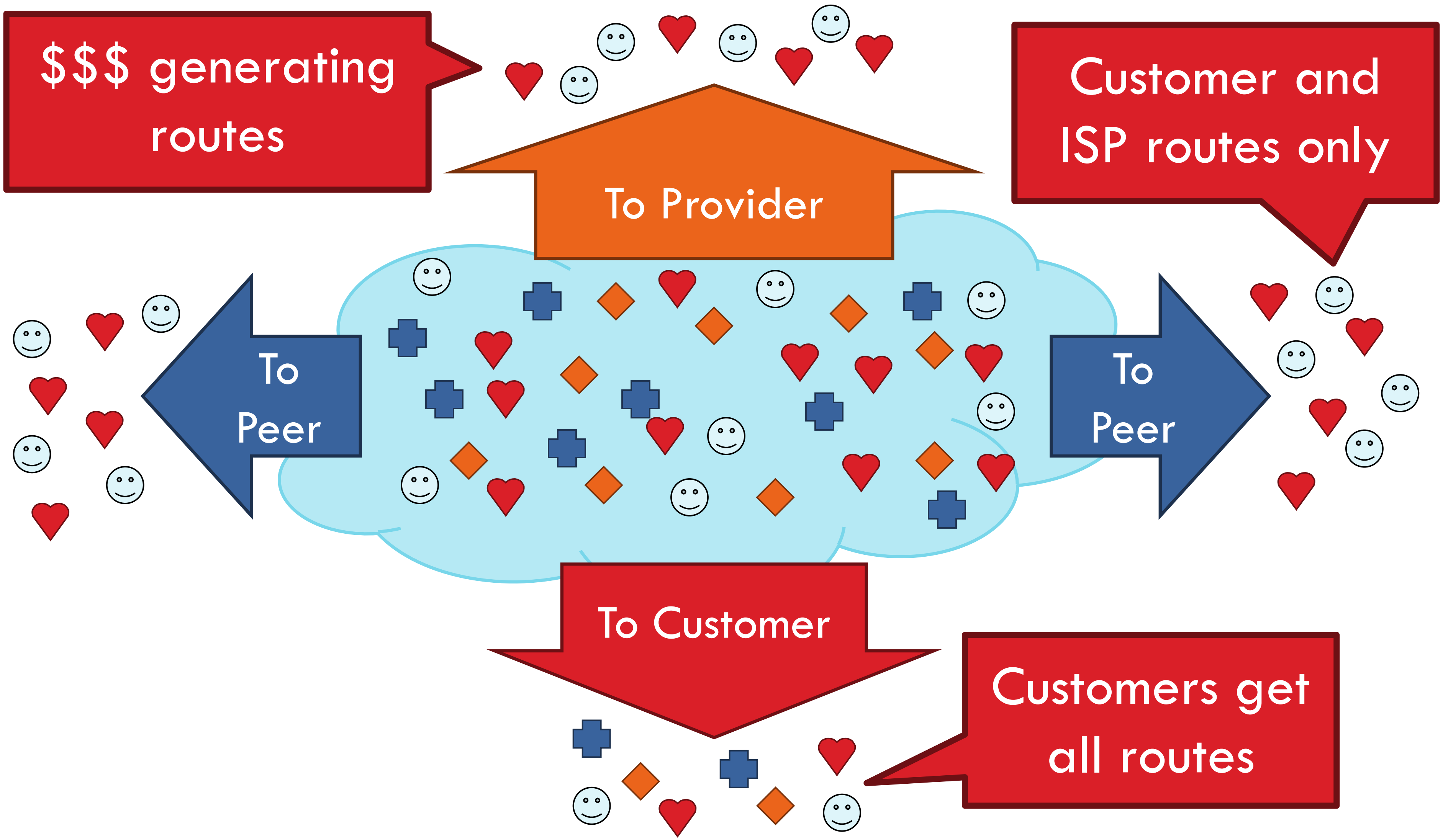
Hot Potato Routing



Importing Routes

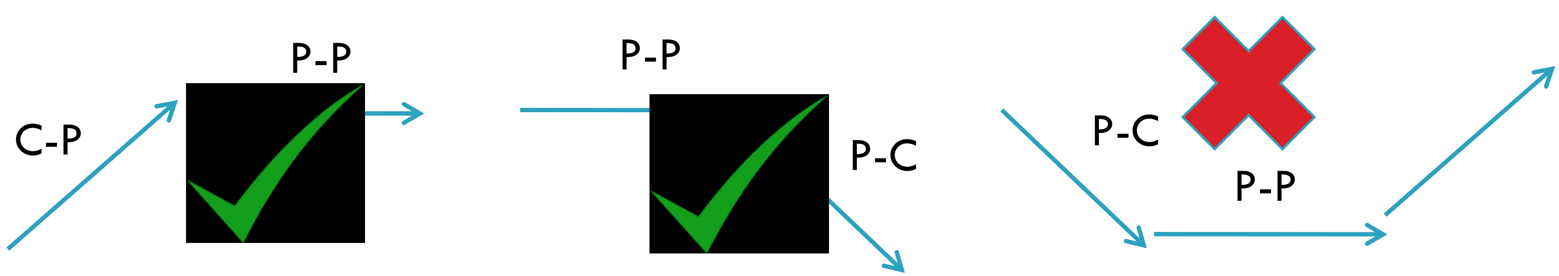


Exporting Routes



Modeling BGP

- AS relationships
  - Customer/provider
  - Peer
  - Sibling, IXP
- Gao-Rexford model
  - AS prefers to use customer path, then peer, then provider
    - Follow the money!
  - Valley-free routing
  - Hierarchical view of routing (incorrect but frequently used)



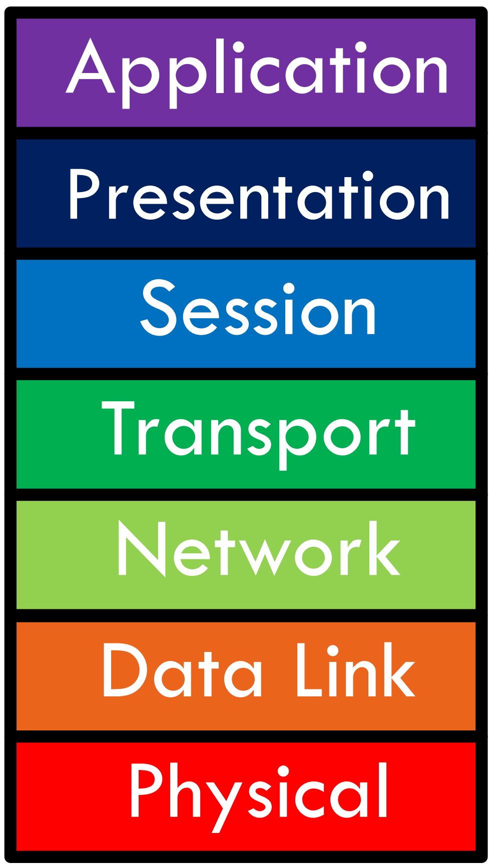
AS Relationships: It's Complicated

- GR Model is strictly hierarchical
  - Each AS pair has exactly one relationship
  - Each relationship is the same for all prefixes
- In practice it's much more complicated
  - Rise of widespread peering
  - Regional, per-prefix peerings
  - Tier-1's being shoved out by "hypergiants"
  - IXPs dominating traffic volume
- Modeling is very hard, very prone to error
  - Huge potential impact for understanding Internet behavior

Other BGP Attributes

- AS_SET
  - Instead of a single AS appearing at a slot, it's a set of Ases
- Communities
  - Arbitrary number that is used by neighbors for routing decisions
    - Export this route only in Europe
    - Do not export to your peers
  - Usually stripped after first interdomain hop
  - Why?
- Prepending
  - Lengthening the route by adding multiple instances of ASN
  - Why?

Transport Layer



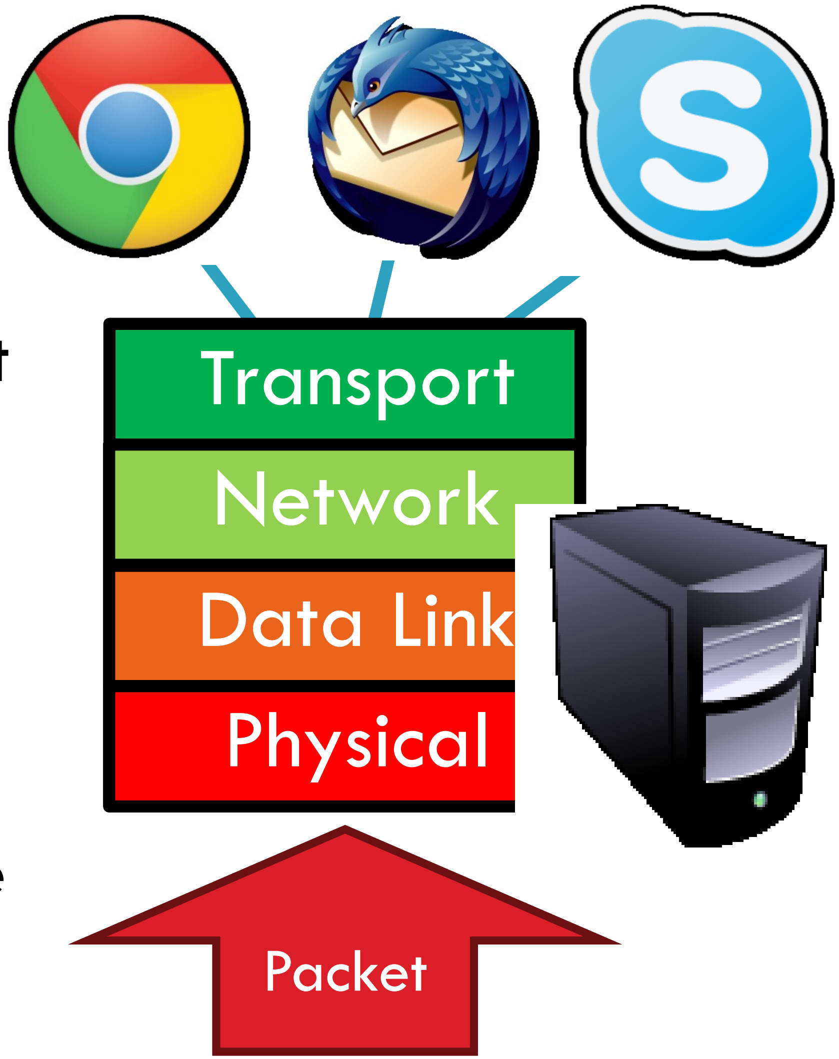
- Function:
  - Demultiplexing of data streams
- Optional functions:
  - Creating long lived connections
  - Reliable, in-order packet delivery
  - Error detection
  - Flow and congestion control
- Key challenges:
  - Detecting and responding to congestion
  - Balancing fairness against high utilization

Outline

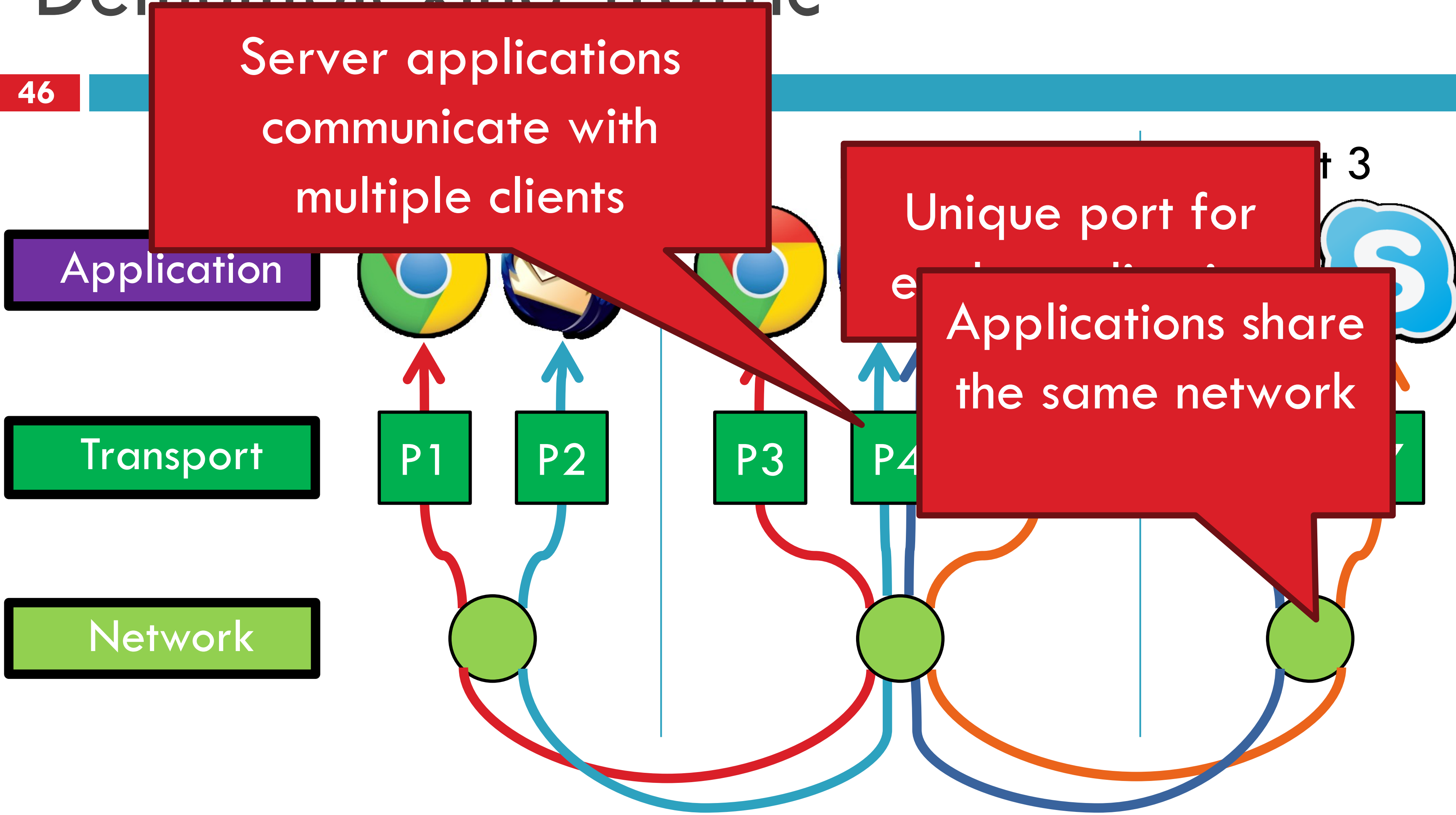
- UDP
- TCP
- Congestion Control
- Evolution of TCP
- Problems with TCP

The Case for Multiplexing

- Datagram network
  - No circuits
  - No connections
- Clients run many applications at the same time
  - Who to deliver packets to?
- IP header "protocol" field
  - 8 bits = 256 concurrent streams
- Insert Transport Layer to handle demultiplexing

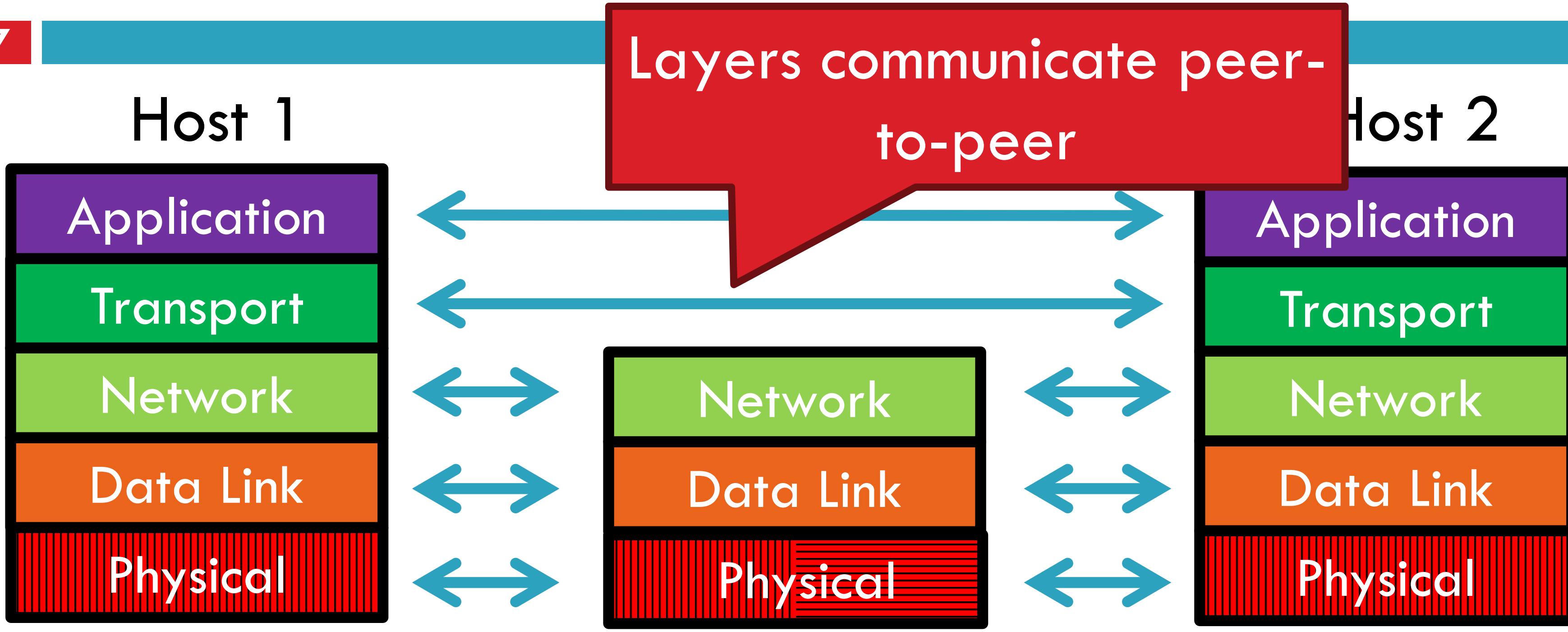


Demultiplexing Traffic



Endpoints identified by <*src_ip, src_port, dest_ip, dest_port*>

Layering, Revisited



- Lowest level end-to-end protocol
  - Transport header only read by source and destination
  - Routers view transport header as payload

User Datagram Protocol (UDP)

0	16	31

Source Port	Destination Port
Payload Length	Checksum

- Simple, connectionless datagram
  - C sockets: SOCK_DGRAM
- Port numbers enable demultiplexing
  - 16 bits = 65535 possible ports
  - Port 0 is invalid
- Checksum for error detection
  - Detects (some) corrupt packets
  - Does not detect dropped, duplicated, or reordered packets

Uses for UDP

- Invented after TCP
  - Why?
- Not all applications can tolerate TCP
- Custom protocols can be built on top of UDP
  - Reliability? Strict ordering?
  - Flow control? Congestion control?
- Examples
  - RTMP, real-time media streaming (e.g. voice, video)
  - Facebook datacenter protocol

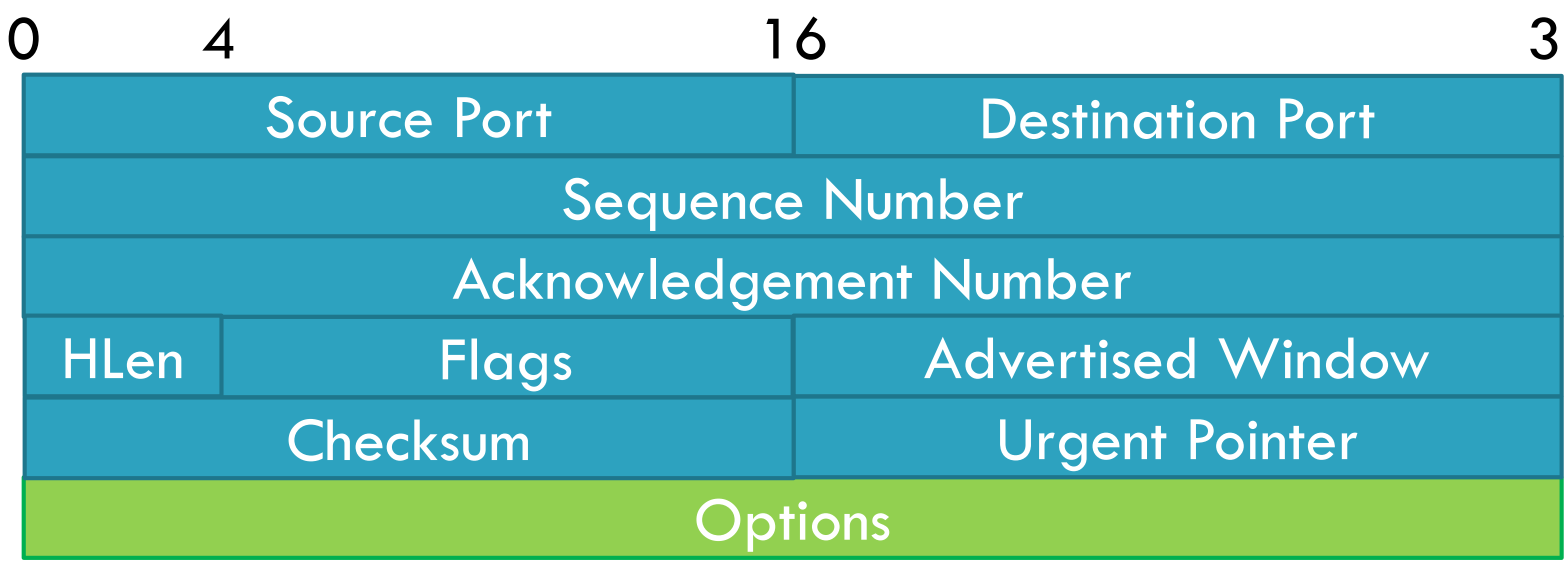
Outline

- UDP – already discussed
- TCP
- Congestion Control
- Evolution of TCP
- Problems with TCP

Transmission Control Protocol

- Reliable, in-order, bi-directional byte streams
  - Port numbers for demultiplexing
  - Virtual circuits (connections)
  - Flow control
  - Congestion control, approximate fairness

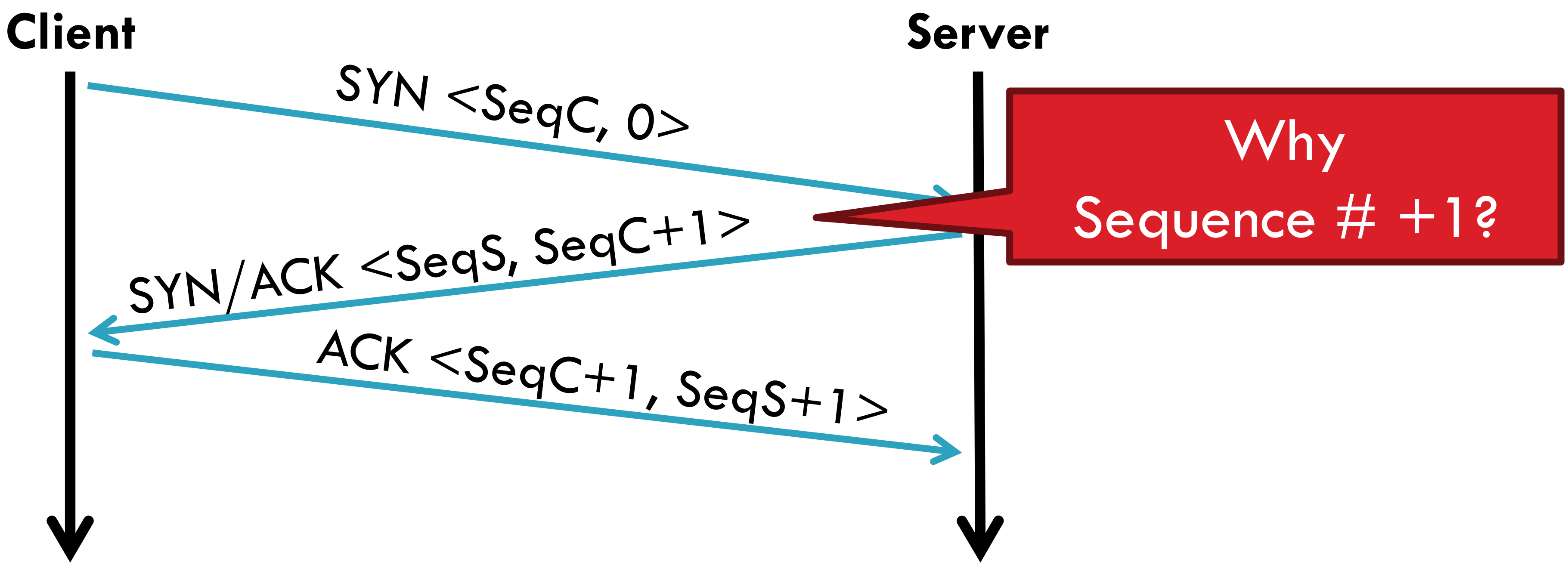
Why these features?



Connection Setup

- Why do we need connection setup?
  - To establish state on both hosts
  - Most important state: sequence numbers
    - Count the number of bytes that have been sent
    - Initial value chosen at random
    - Why?
- Important TCP flags (1 bit each)
  - SYN – synchronization, used for connection setup
  - ACK – acknowledge received data
  - FIN – finish, used to tear down connection

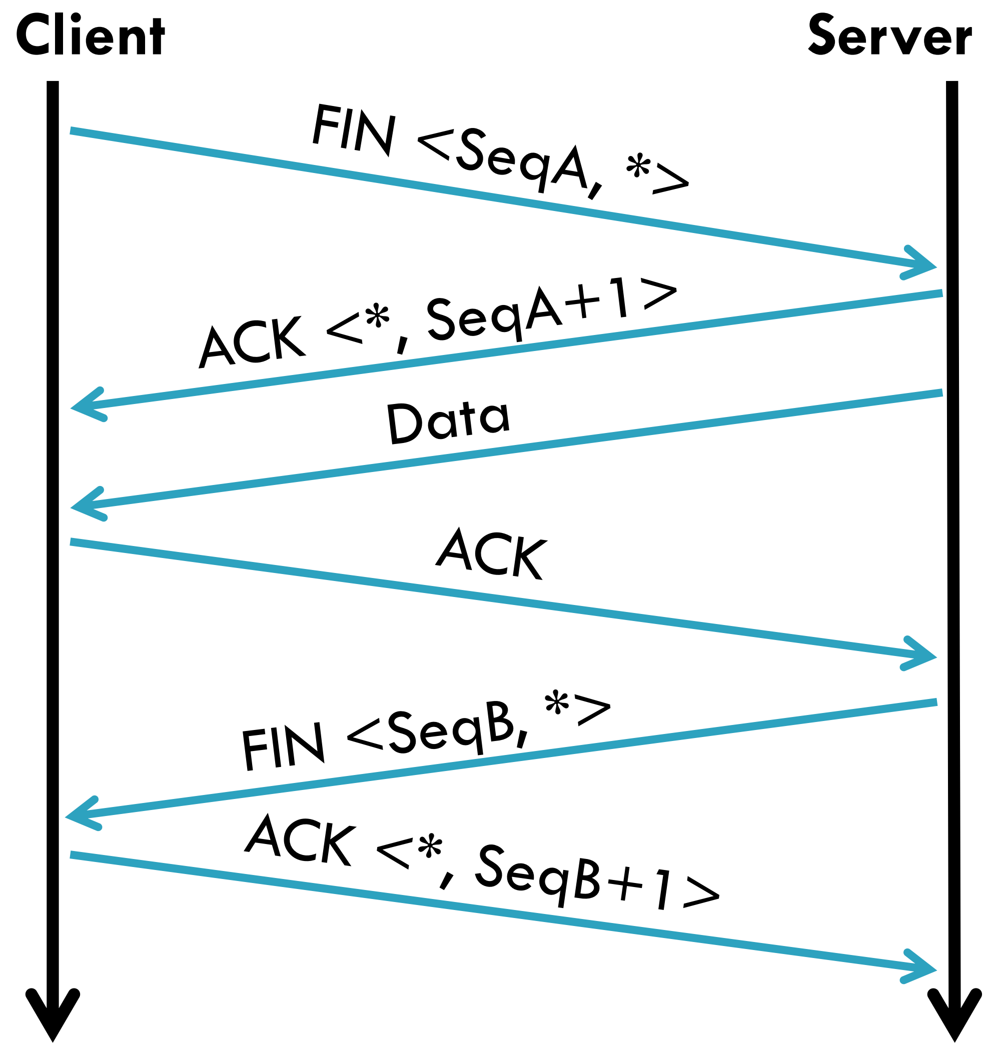
Three Way Handshake



- Each side:
  - Notifies the other of starting sequence number
  - ACKs the other side's starting sequence number

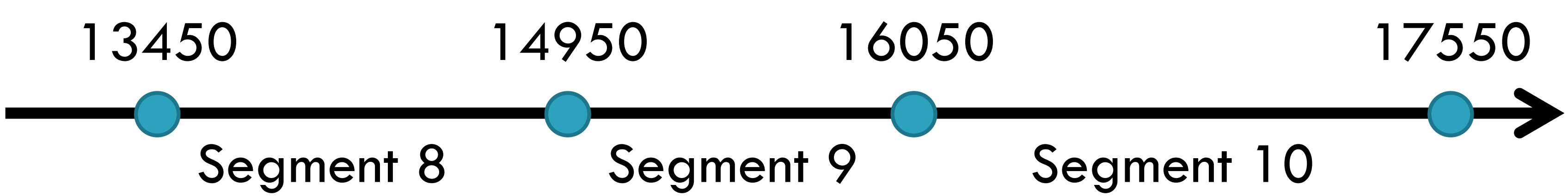
Connection Tear Down

- Either side can initiate tear down
- Other side may continue sending data
  - Half open connection
  - *shutdown()*
- Acknowledge the last FIN
  - Sequence number + 1
- What happens if 2nd FIN is lost?

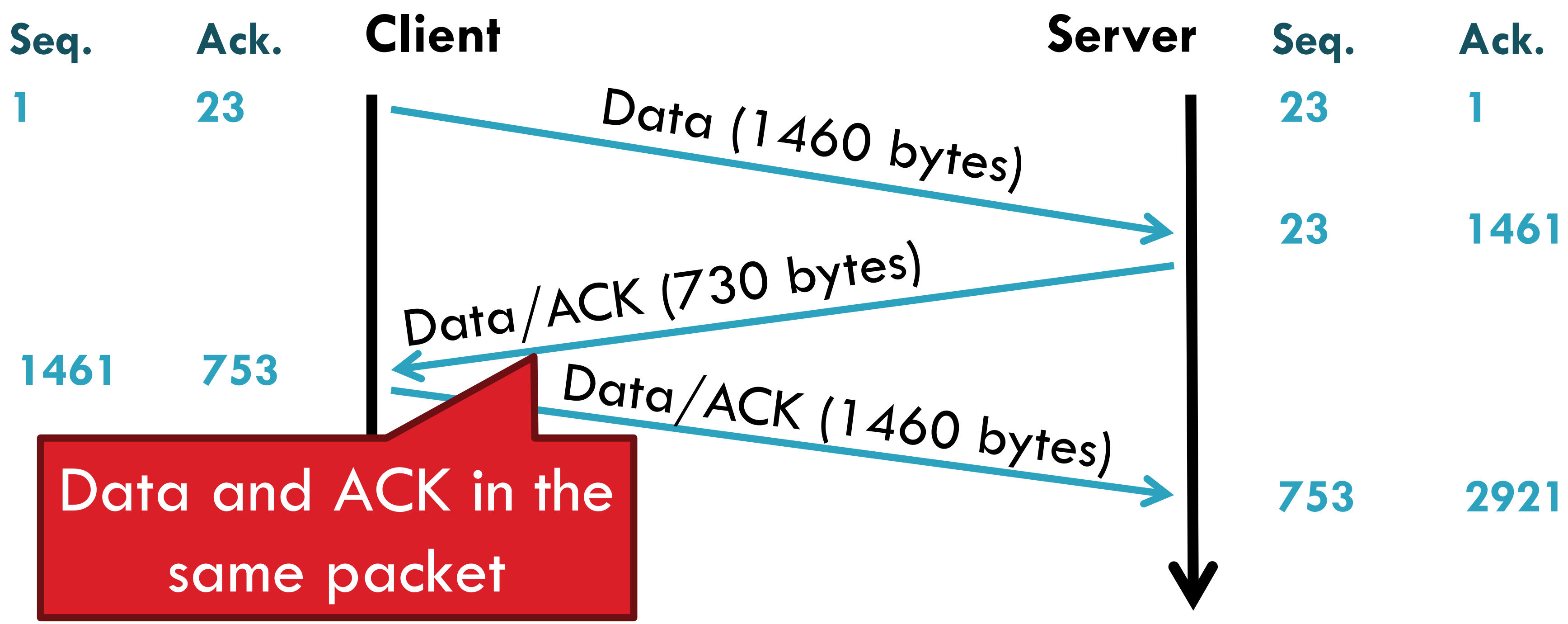


Sequence Number Space

- TCP uses a byte stream abstraction
  - Each byte in each stream is numbered
  - 32-bit value, wraps around
  - Initial, random values selected during setup. Why?
- Byte stream broken down into segments (packets)
  - Size limited by the Maximum Segment Size (MSS)
  - Set to limit fragmentation
- Each segment has a sequence number



Bidirectional Communication

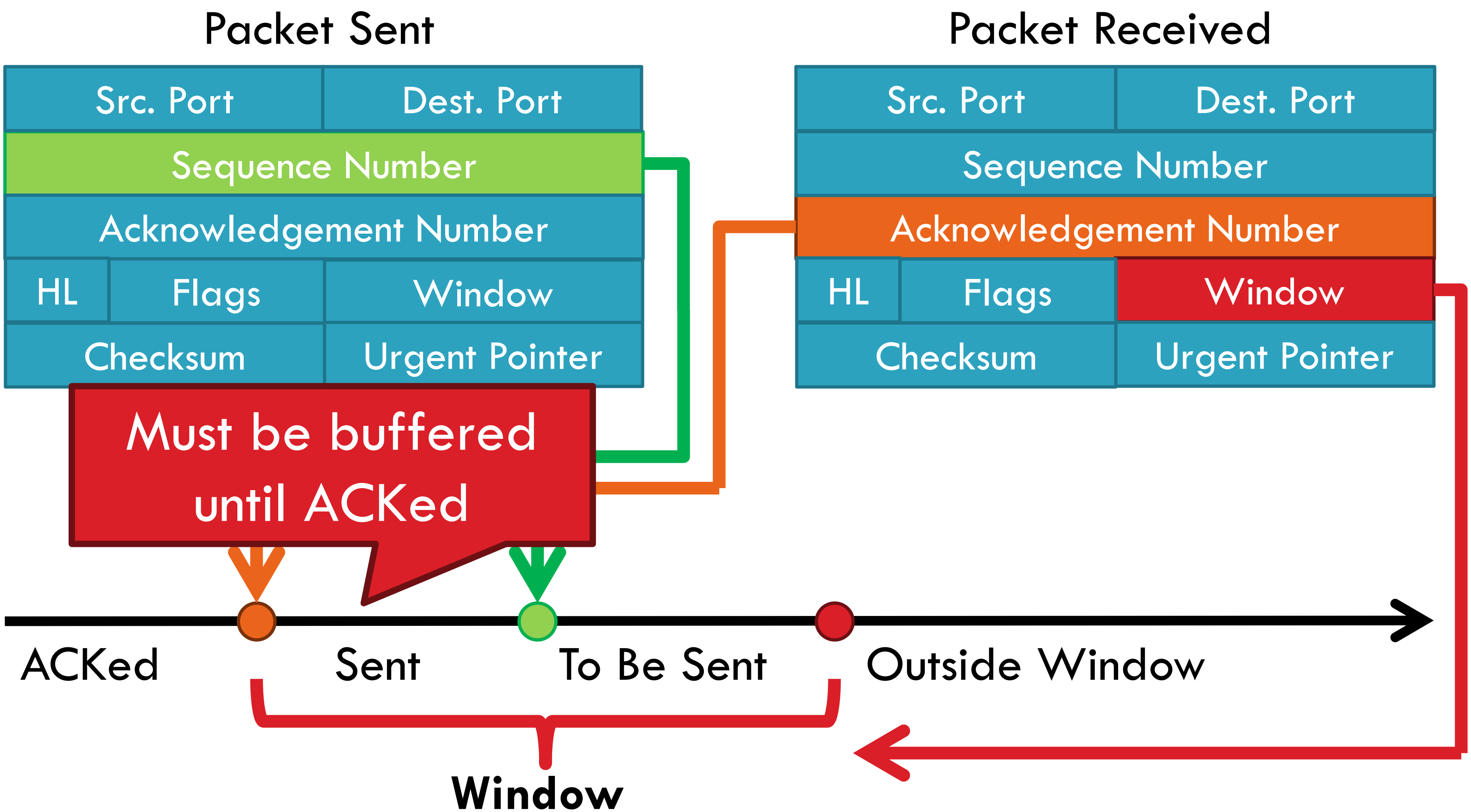


- Each side of the connection can send and receive
  - Different sequence numbers for each direction

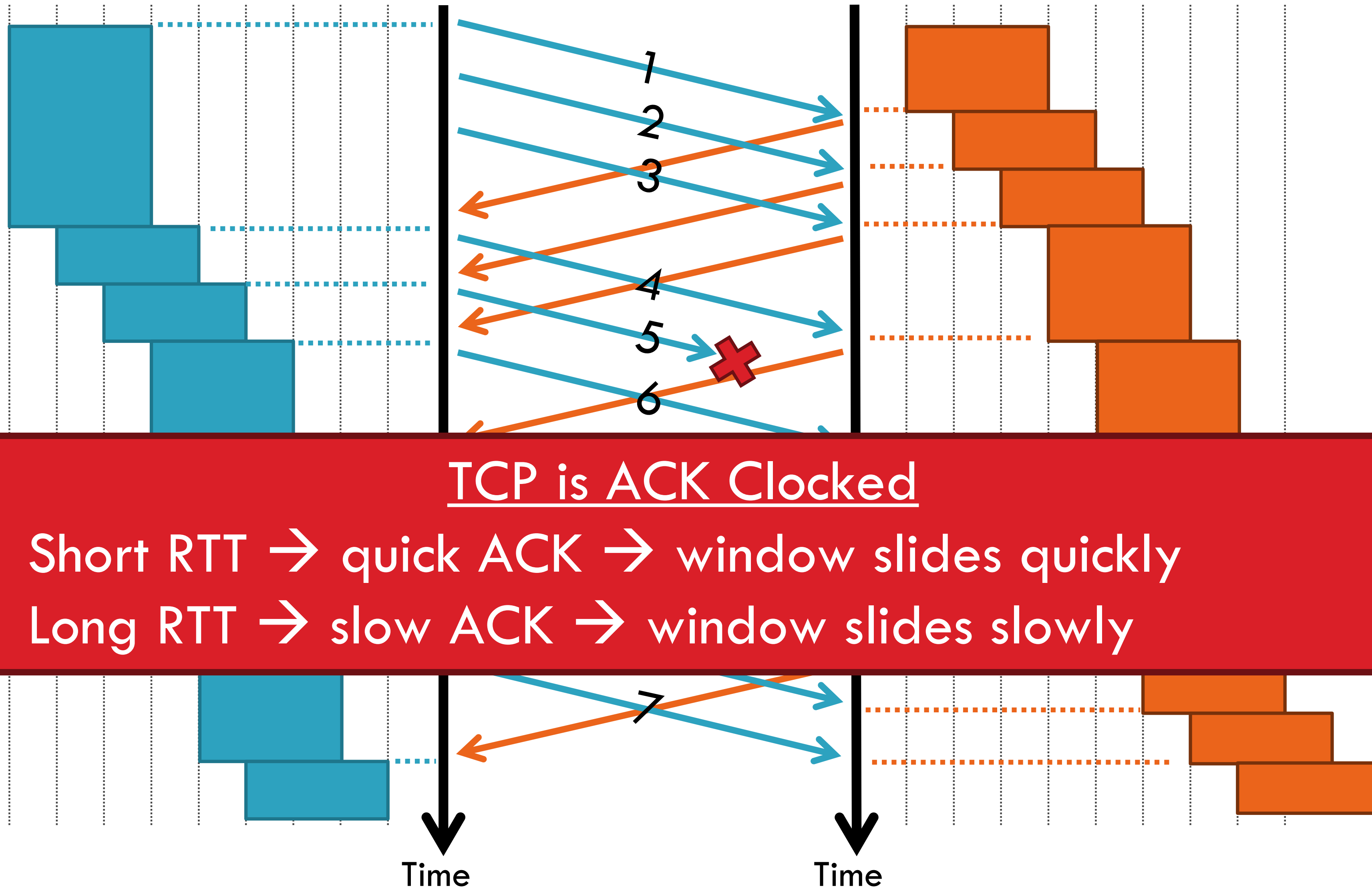
Flow Control

- Problem: how many packets should a sender transmit?
  - Too many packets may overwhelm the receiver
  - Size of the receivers buffers may change over time
- Solution: sliding window
  - Receiver tells the sender how big their buffer is
  - Called the advertised window
  - For window size *n*, sender may transmit *n* bytes without receiving an ACK
  - After each ACK, the window slides forward
- Window may go to zero!

Flow Control: Sender Side



Sliding Window Example



TCP is ACK Clocked

- Short RTT → quick ACK → window slides quickly
- Long RTT → slow ACK → window slides slowly

Observations

- Throughput is ~ w/RTT

- Sender has to buffer all unacknowledges packets, because they may require retransmission

- Receiver may be able to accept out-of-order packets, but only up to buffer limits

What Should the Receiver ACK?

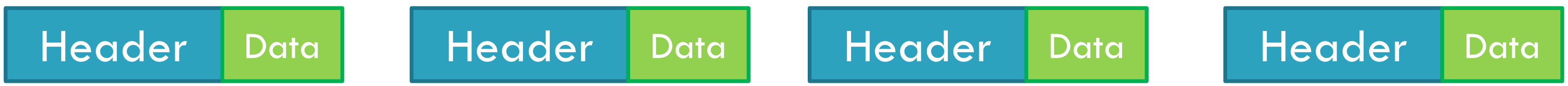
1. ACK every packet
2. Use *cumulative ACK*, where an ACK for sequence n implies ACKS for all $k < n$
3. Use *negative ACKs* (NACKs), indicating which packet did not arrive
4. Use *selective ACKs* (SACKs), indicating those that did arrive, even if not in order
   - SACK is an actual TCP extension

Sequence Numbers, Revisited

- 32 bits, unsigned
  - Why so big?
- For the sliding window you need…
  - |Sequence # Space| > 2 * |Sending Window Size|
  - $2^{32} > 2 * 2^{16}$
- Guard against stray packets
  - IP packets have a maximum segment lifetime (MSL) of 120 seconds
    - i.e. a packet can linger in the network for 2 minutes

Silly Window Syndrome

- Problem: what if the window size is very small?
  - Multiple, small packets, headers dominate data



- Equivalent problem: sender transmits packets one byte at a time

```
for (int x = 0; x < strlen(data); ++x)
  write(socket, data + x, 1);
```

Nagle's Algorithm

1. If the window >= MSS and available data >= MSS:

   Send the data

   Send a full packet

2. Elif there is unACKed data:

   Enqueue data in a buffer until an ACK is received

3. Else: send the data

   Send a non-full packet if nothing else is happening

- Problem: Nagle's Algorithm delays transmissions
  - What if you need to send a packet immediately?

  1. `int flag = 1;`
  2. `setsockopt(sock, IPPROTO_TCP, TCP_NODELAY, (char *) &flag, sizeof(int));`