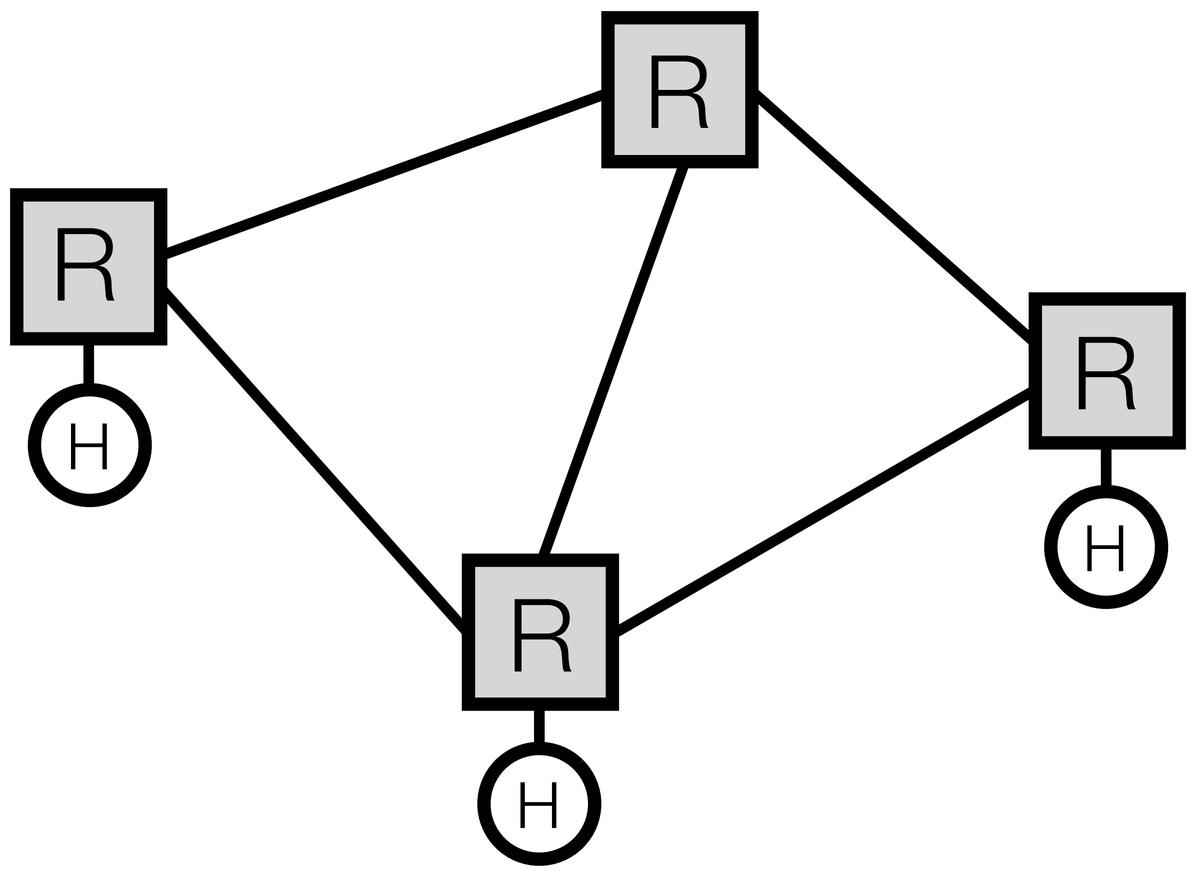
CS 725/825 & IT 725
Lecture 7
Link Layer Switching

September 18, 2024

Recall...

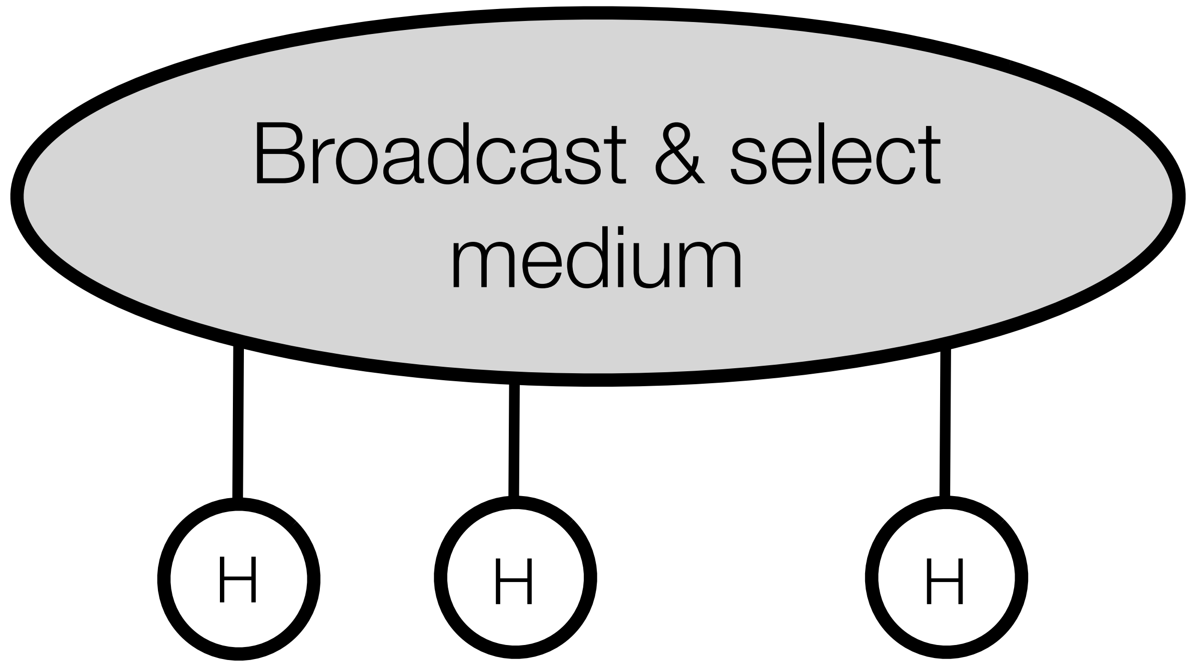
- Routed networks



- topology driven by geography
- long distances (high latency)
- need for scalability
- location-related addresses
- routing

➡ **Network Layer (L3)**

- Broadcast & select



- everyone connected to everyone
- short distances (low latency)
- lesser need for scalability
- arbitrary addresses
- address discovery

➡ **Link Layer (L2)**

Routing Alternative: Bridging

- Motivation
  - L2 networks do not scale due to the broadcast nature of the underlying medium
  - Routers are expensive and require configuration
- Approach - extend the reach of L2
- Solution - limit the scope of packet delivery (bridging)

Motivation (personal)

John Green Hall, the former home of the *Department of Mathematics and Computer Science*, University of Denver

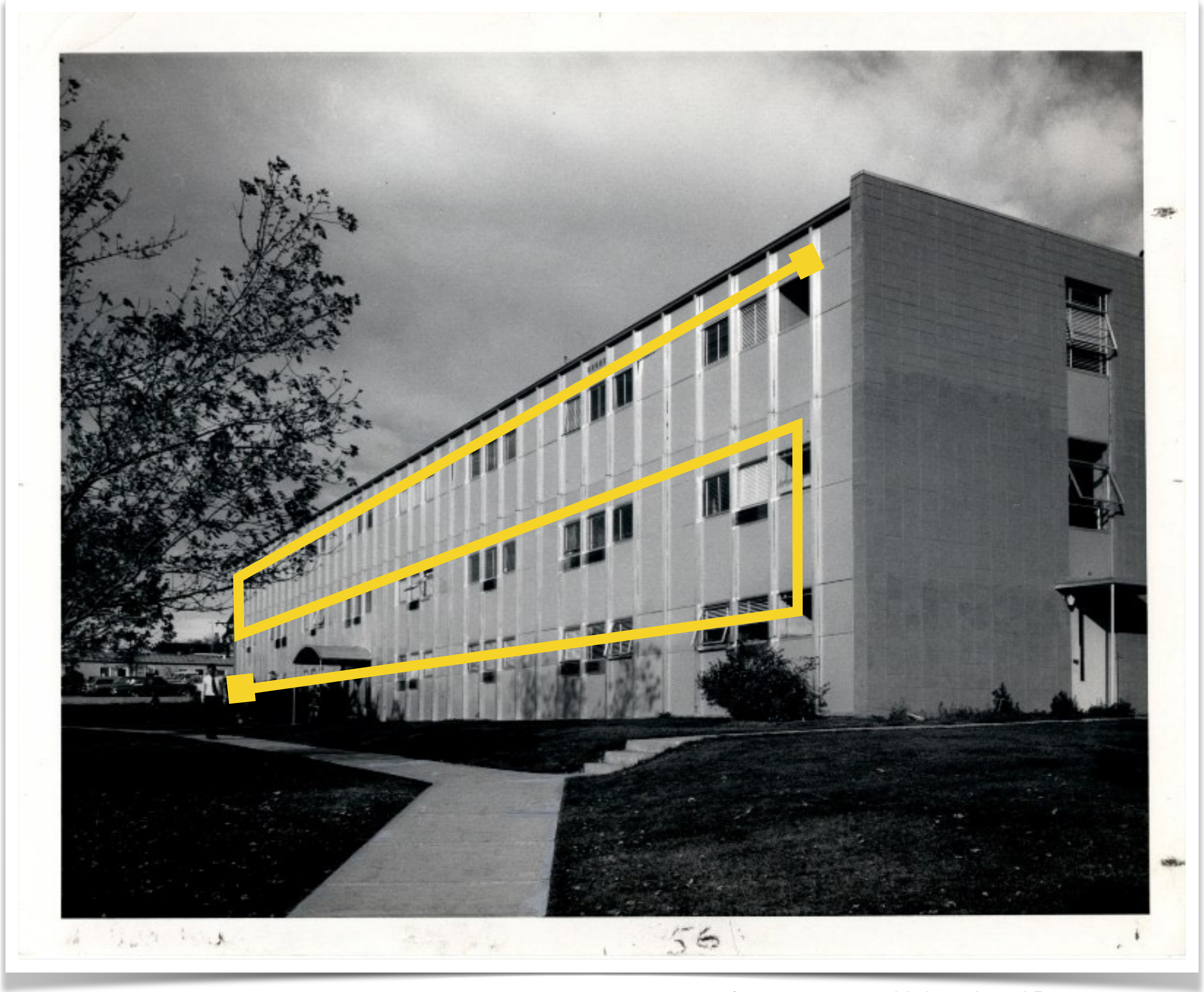
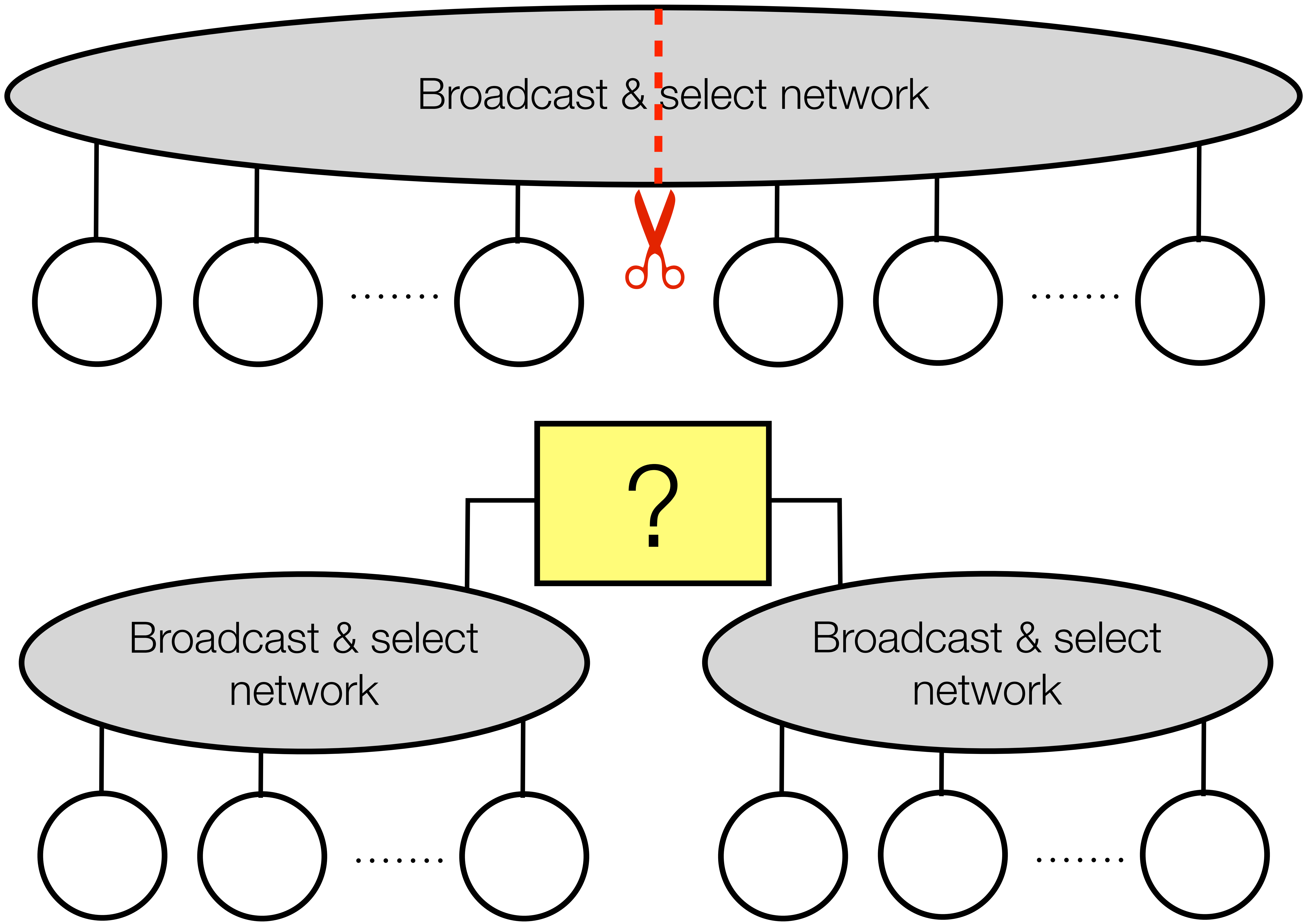


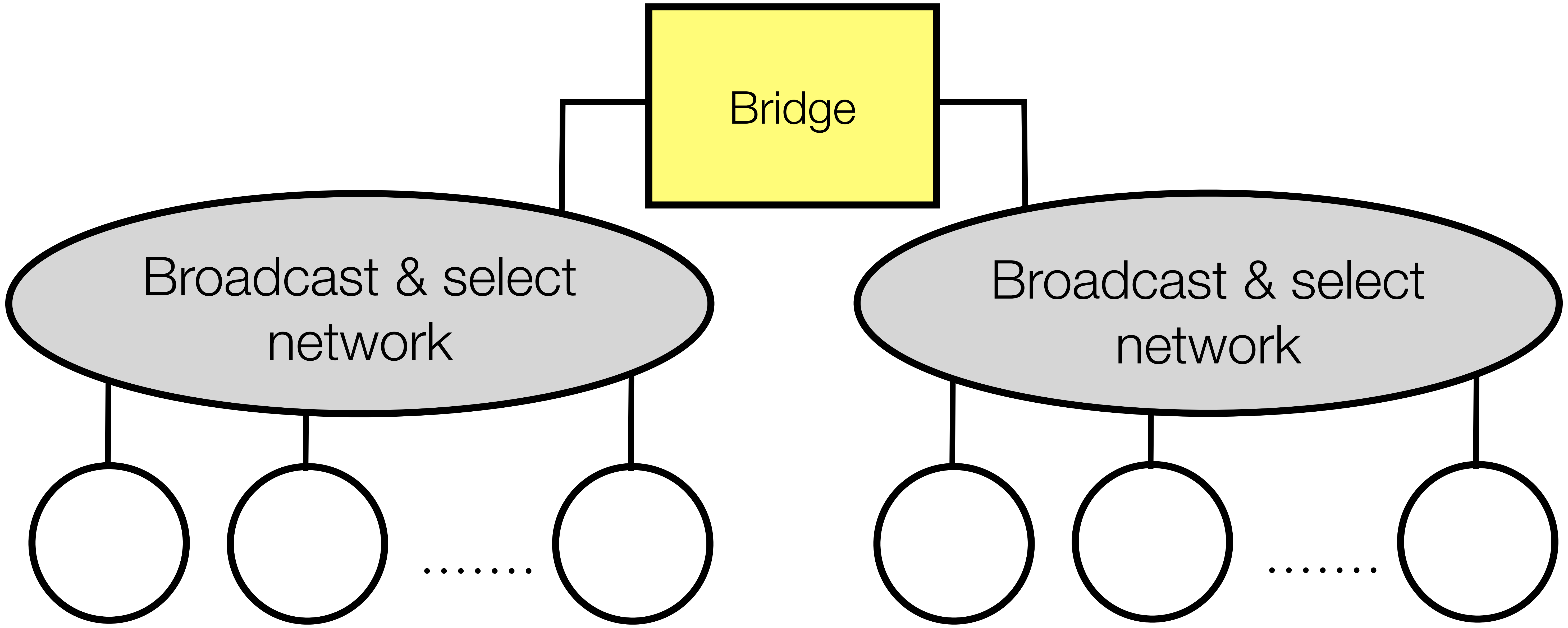
Image source: University of Denver

Historical Evolution



Link Layer Bridging

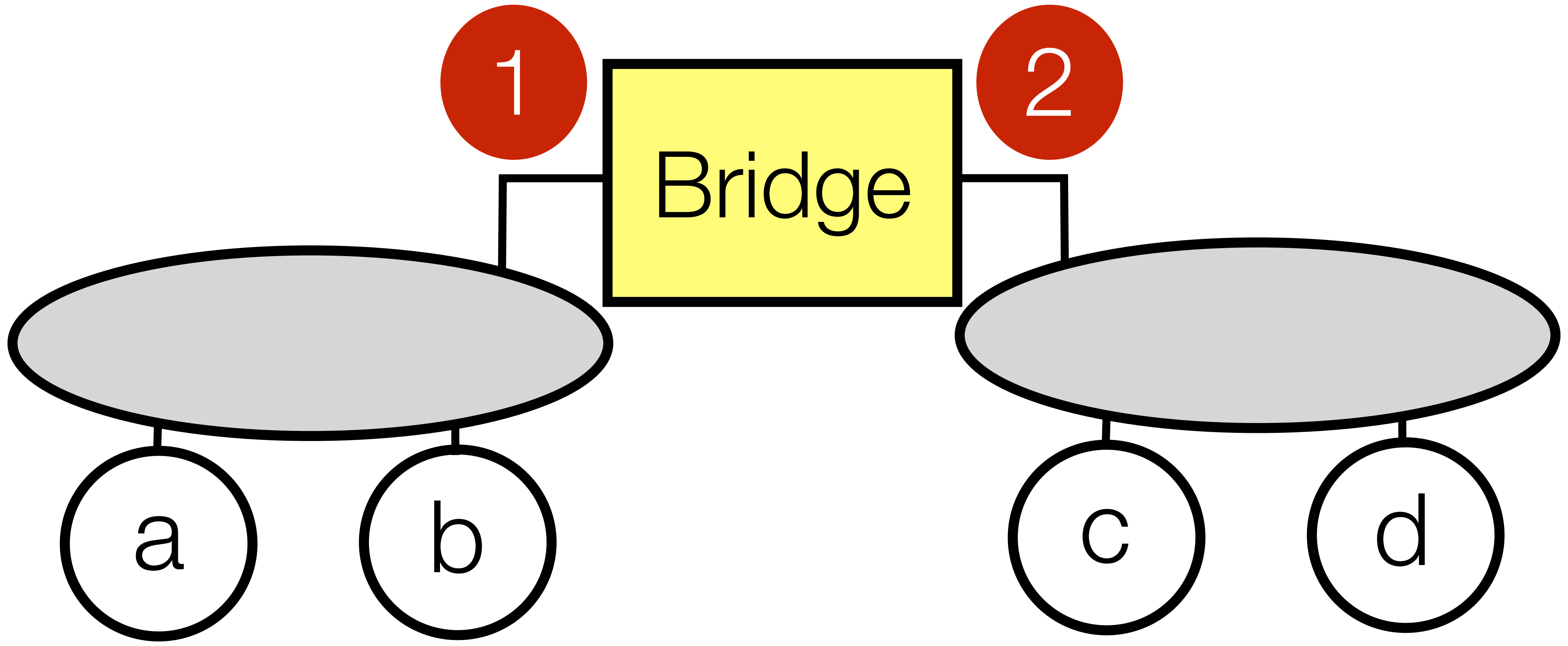
- Bridge "opens" for non-local traffic and broadcasts
- Bridge learns node locations from passing traffic and stores them in its Forwarding Database (FDB) (a.k.a. bridge, bridging, or switching table)



Transparent Bridging

- Initially, the bridging table (FDB) is empty
- Broadcast traffic is let to pass, source address recorded in the FDB
- Traffic to an unknown destination is let to pass through the bridge, source address address is recorded in the FDB
- Non-local traffic (to a known destination that is associated with different interface, e.g., *a* to *d*) is let to pass, “local” traffic (e.g., *a* to *b*) is blocked

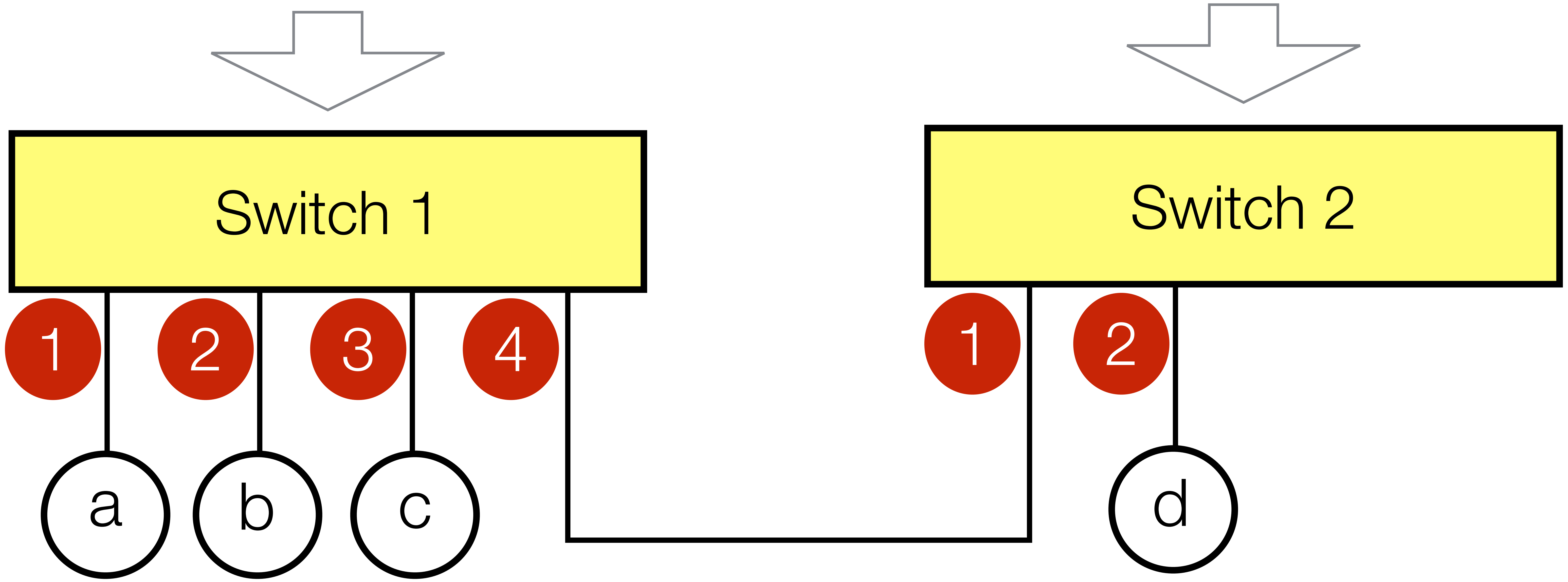
MAC	Interface
a	1
b	1
c	2
d	2



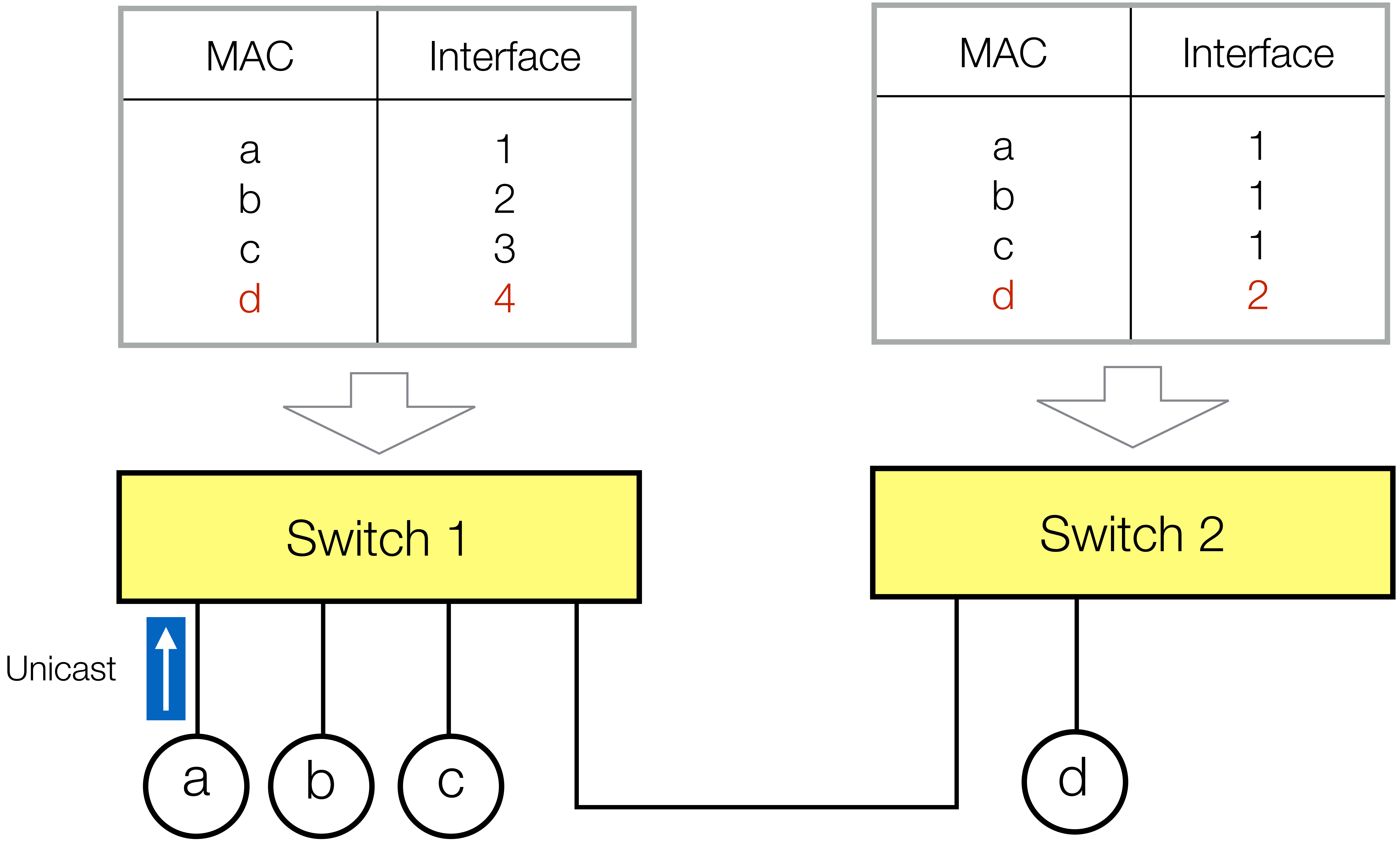
L2 (Ethernet) Switching

MAC	Interface
a	1
b	2
c	3
d	4

MAC	Interface
a	1
b	1
c	1
d	2



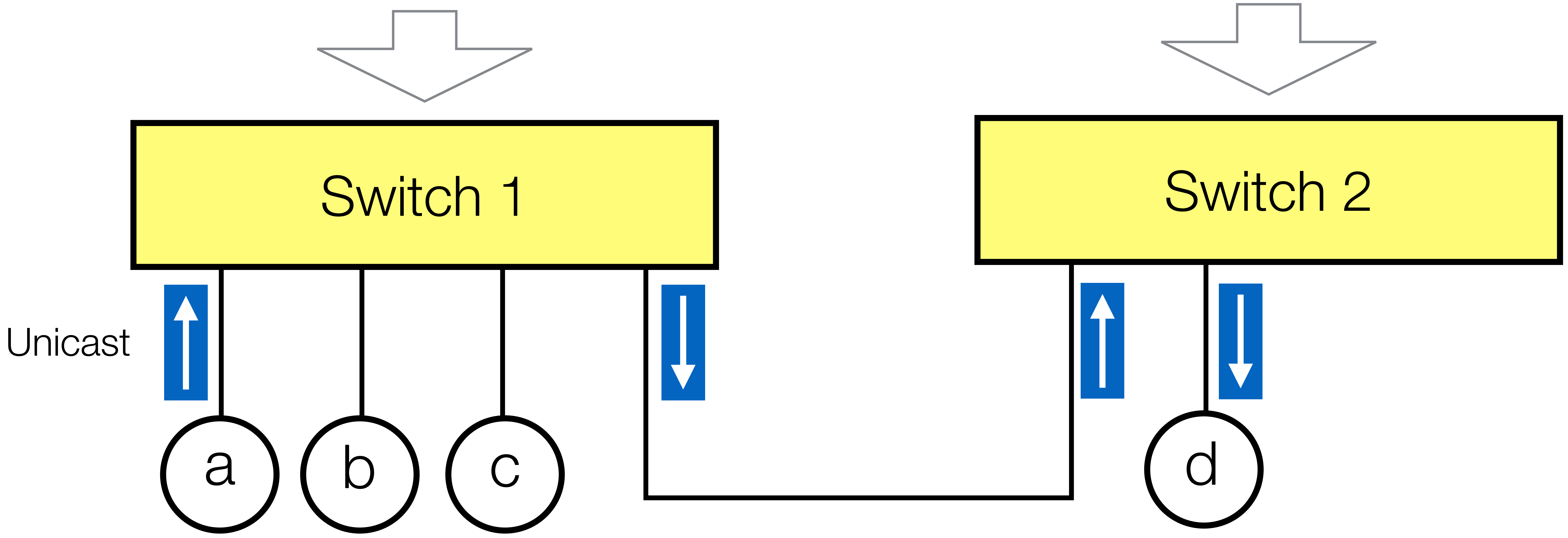
Unicast packet from a to d



Unicast packet from a to d

MAC	Interface
a	1
b	2
c	3
d	4

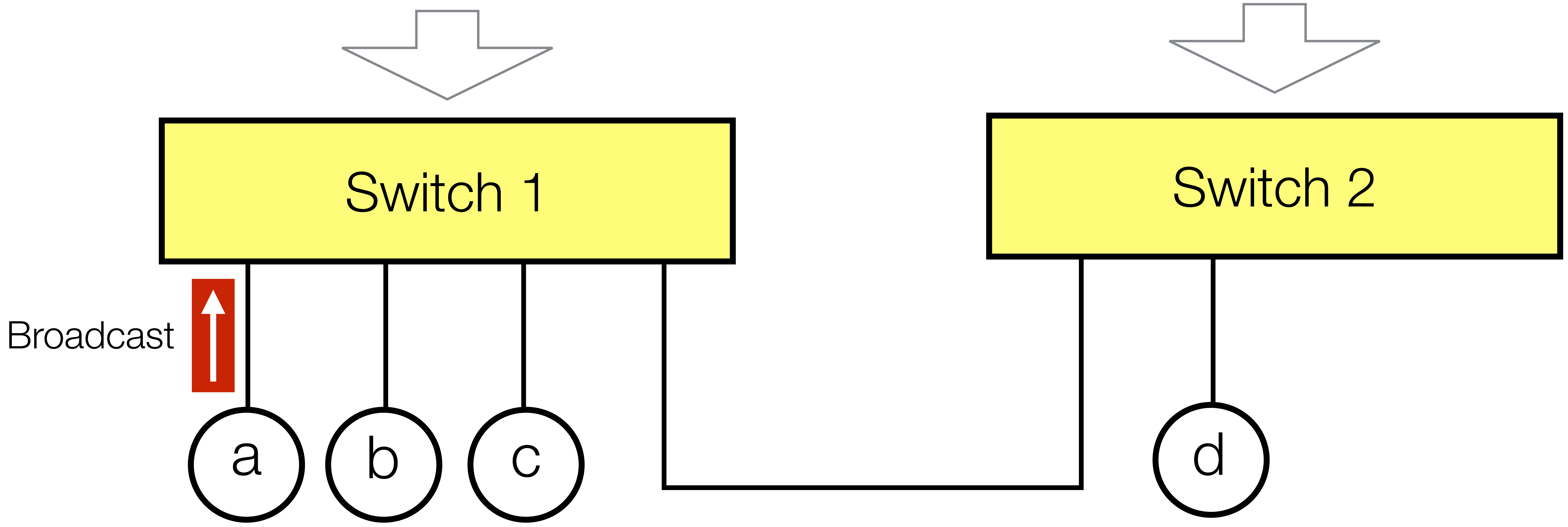
MAC	Interface
a	1
b	1
c	1
d	2



Broadcast packet

MAC	Interface
a	1
b	2
c	3
d	4

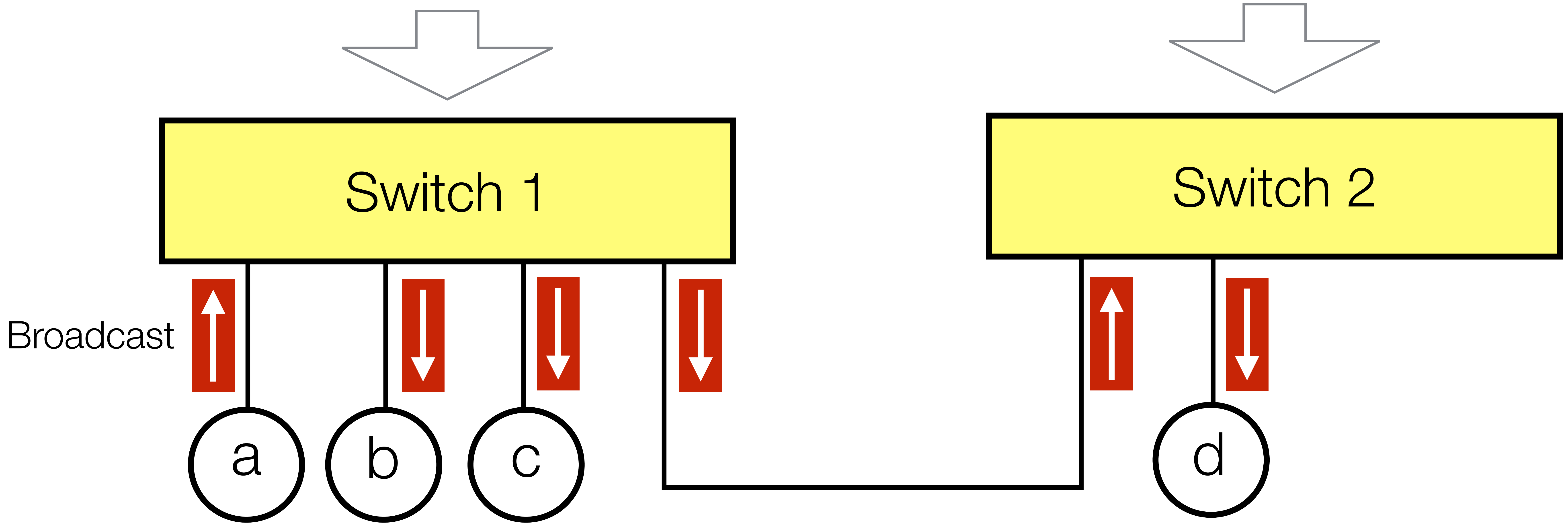
MAC	Interface
a	1
b	1
c	1
d	2



Broadcast packet

MAC	Interface
a	1
b	2
c	3
d	4

MAC	Interface
a	1
b	1
c	1
d	2



Standardization

- ISO - International Organization for Standardization
- ITU-T - International Telecommunication Union - Telecommunication Sector
- IEEE - Institute of Electrical and Electronic Engineers
- IETF - Internet Engineering Task Force
- *x* Forum / *x* Alliance / *x* Group

Standardization

- IEEE - Institute of Electrical and Electronic Engineers
  - 802.3an: 10GBASE-T 10 Gbit/s (1,250 MB/s) Ethernet over unshielded twisted pair (UTP)
  - 802.11ad: (in works) gigabit “WiFi” in 60 GHz band
- IETF - Internet Engineering Task Force
  - RFC791: Internet Protocol - DARPA Internet Program Protocol Specification (1981)

Standardization Process

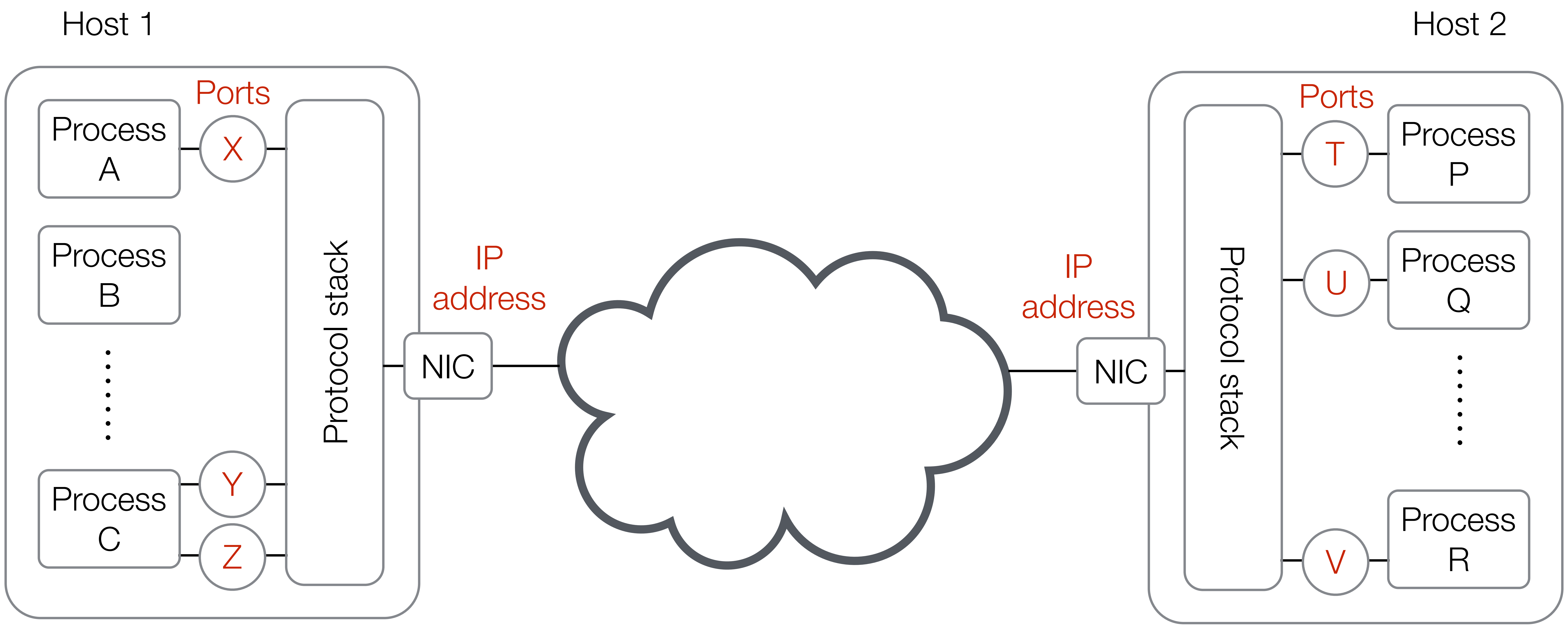
- BOF - Birds of a Feather (... flock together)
- WG - Work Group
  - working documents / draft standards
- Editor
  - standard documents

Application Layer

Application Layer

- Assumptions:
  - each host (each network interface, actually) has a globally unique id (IP address)
  - each communication endpoint of an application has an id that is unique within the host (port number)
  - underlying network provides reliable connection-oriented or unreliable connection-less service (TCP or UDP)
- For a particular transport protocol, each “communication” is uniquely identified by a quadruple: *src/dst IP addresses* & *src/dst port numbers*

Client and Server



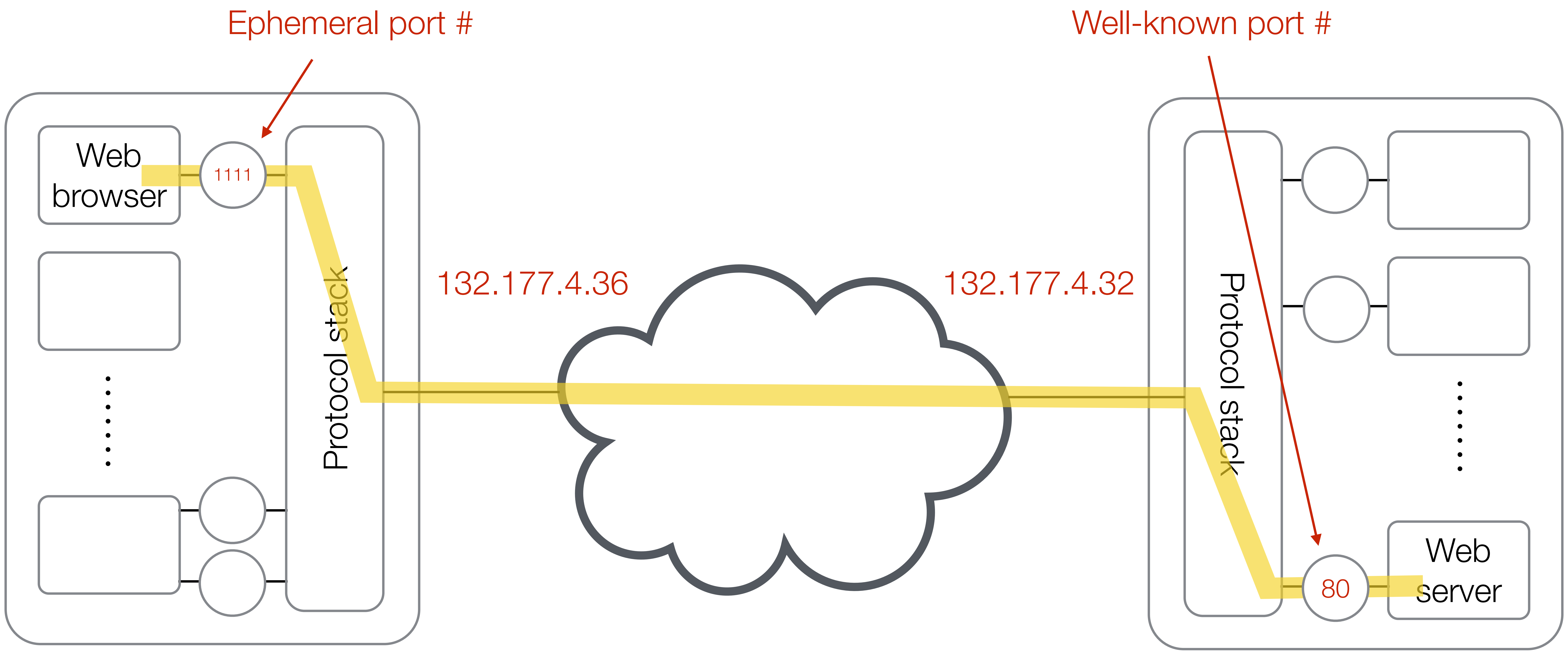
NIC - Network Interface Card

Client and Server

- Client (caller)
  - actively opens connection to a server
  - must know server's IP address and port #
  - typically uses *ephemeral* source (local) port number

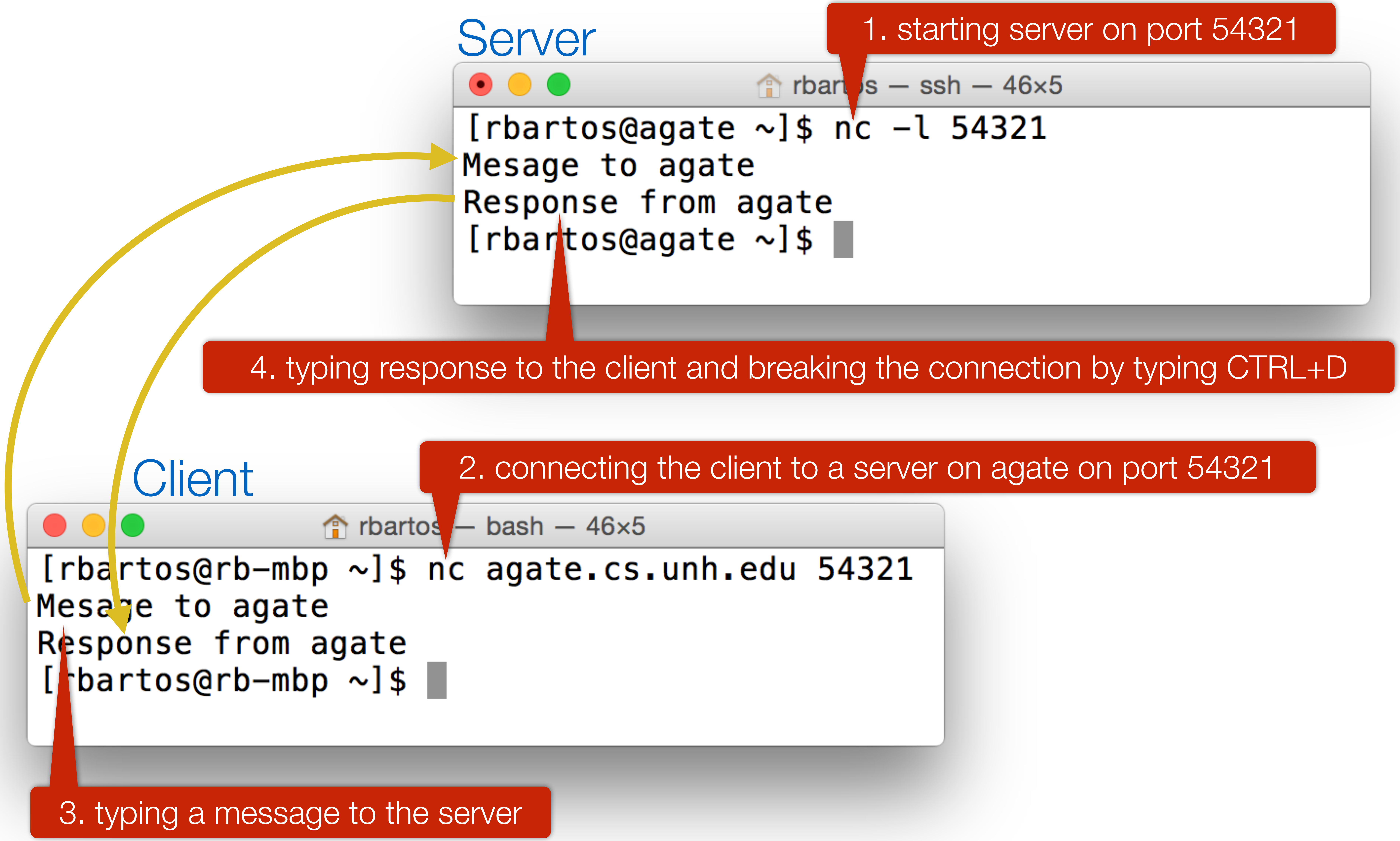
- Server (callee)
  - connects to a local port (typically a well-known one)
  - waits for clients to connect
  - may handle multiple simultaneous client connections

Client and Server

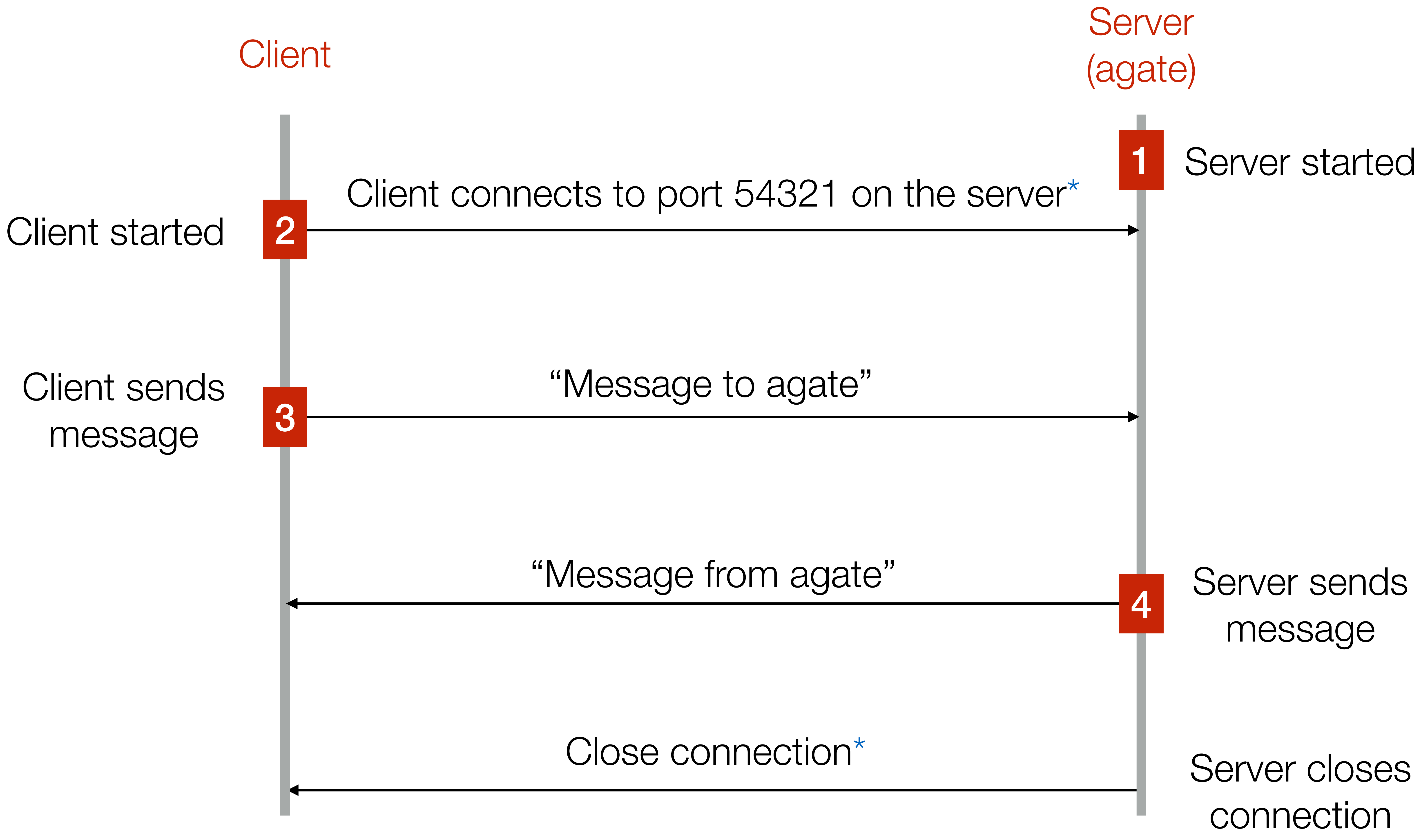


A process (web browser) connected to ephemeral port 1111 on a host with IP address 132.177.4.36 opens connection to a process that listens on well-known port 80 (web server) on a host with IP address 132.177.4.32

Command Line Utility: nc



Sequence Diagram



(*) this is a more complex interaction than show here

Socket API

- Berkeley socket API (4.2 BSD Unix, 1983)
- POSIX socket API (reentrant)
- Designed to support any protocol - not just TCP/UDP/IP
- Defined in C, but adopted by essentially all programming languages

Main Operations

- Address resolution (DNS)
- Binding to a local port number
- Client opening connection to a server
- Server accepting connections from clients
- Sending and receiving data
- Getting and setting connection parameters
- Closing connection
- Server handling of simultaneous connections

Perspectives

- Reliable, stream-oriented service (TCP)
  - Connection-oriented client-side
  - Connection-oriented server-side
- Unreliable, datagram service (UDP)

In Python…

- Address resolution (DNS)
- Binding to a local port number
- Client opening connection to a server
- Server accepting connections from clients
- Sending and receiving data
- Getting and setting connection parameters
- Closing connection
- Server handling of simultaneous connections

Server

```
import socket
BUFFER_SIZE = 100

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 54321))
s.listen(5)
ss, remote_address = s.accept()
print('Received', ss.recv(BUFFER_SIZE).decode())
ss.send('Message from agate\n'.encode())
ss.close()
```

Client

```
import socket
BUFFER_SIZE = 100

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('agate.cs.unh.edu', 54321))
s.send('Message to agate\n'.encode())
print('Received', s.recv(BUFFER_SIZE).decode())
s.close()
```

This is in no way an example of how to write networking code! Among other issues, the code does not even do the most trivial error checking