Computer Networks
COL 334/672

Transport Layer

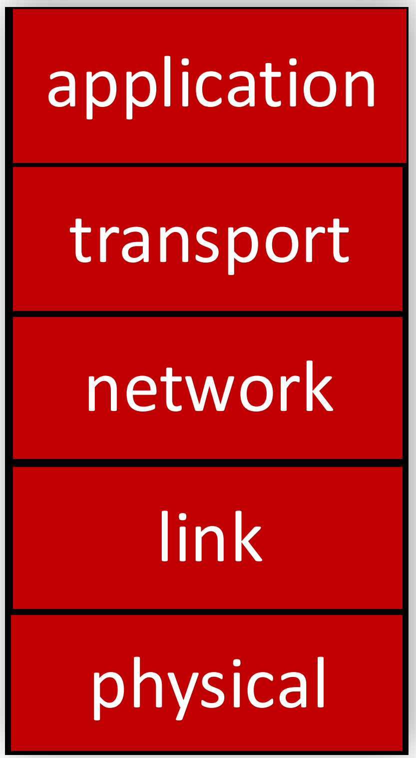
Tarun Mangla

Slides adapted from KR

Sem 1, 2024-25

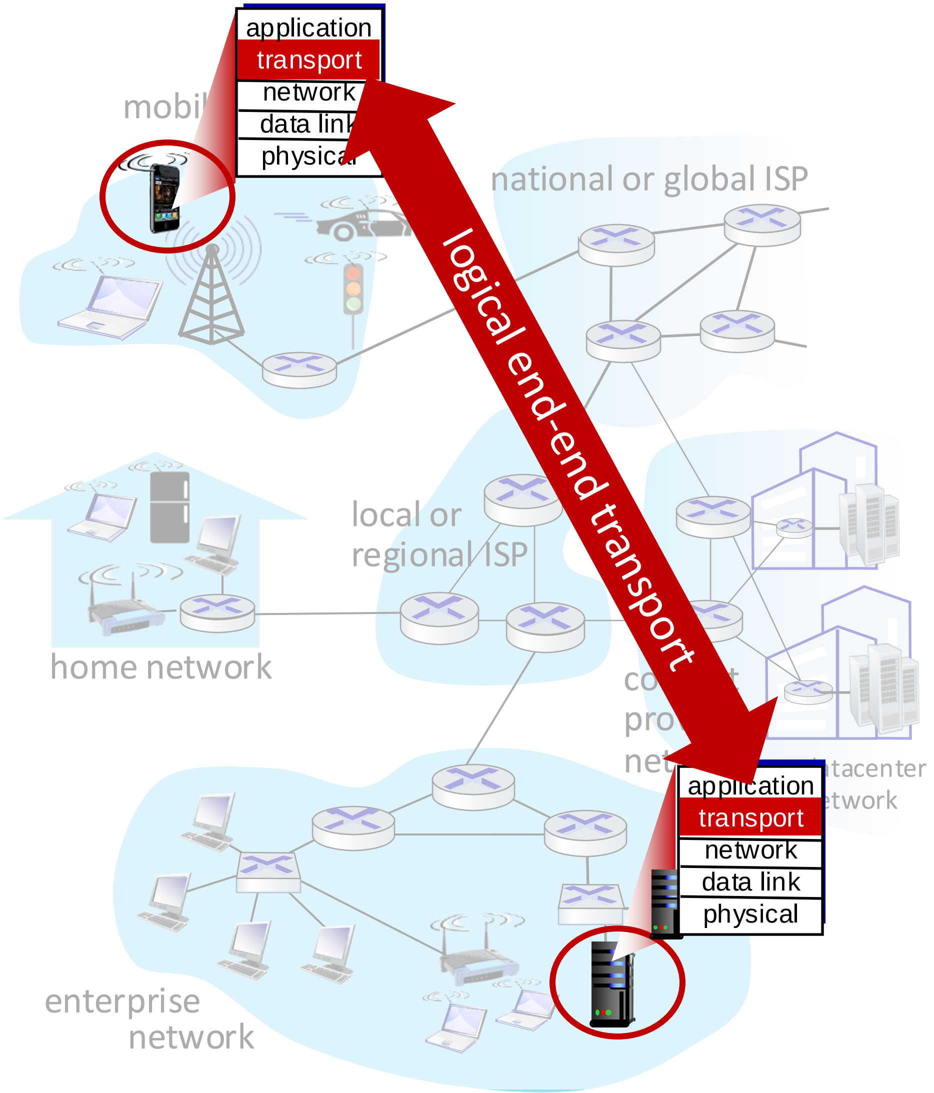
Course Recap

- *application:* supporting network applications
  - HTTP, IMAP, SMTP, DNS
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.11 (WiFi), PPP
- *physical:* bits "on the wire"

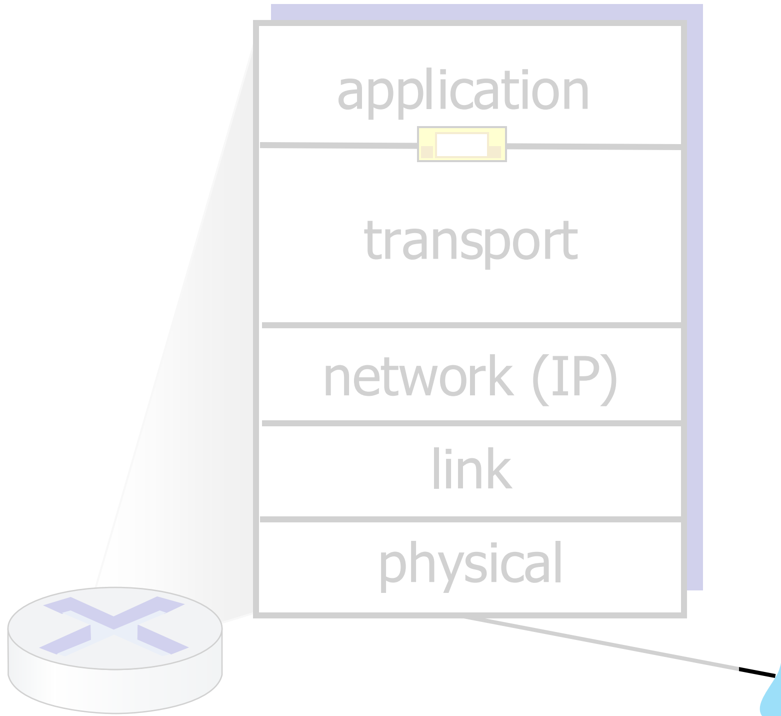


Transport services and protocols

- provide *logical communication* between application processes running on different hosts
- transport protocols actions in end systems:
  - sender: breaks application messages into *segments*, passes to network layer
  - receiver: reassembles segments into messages, passes to application layer
- two transport protocols available to Internet applications
  - TCP, UDP

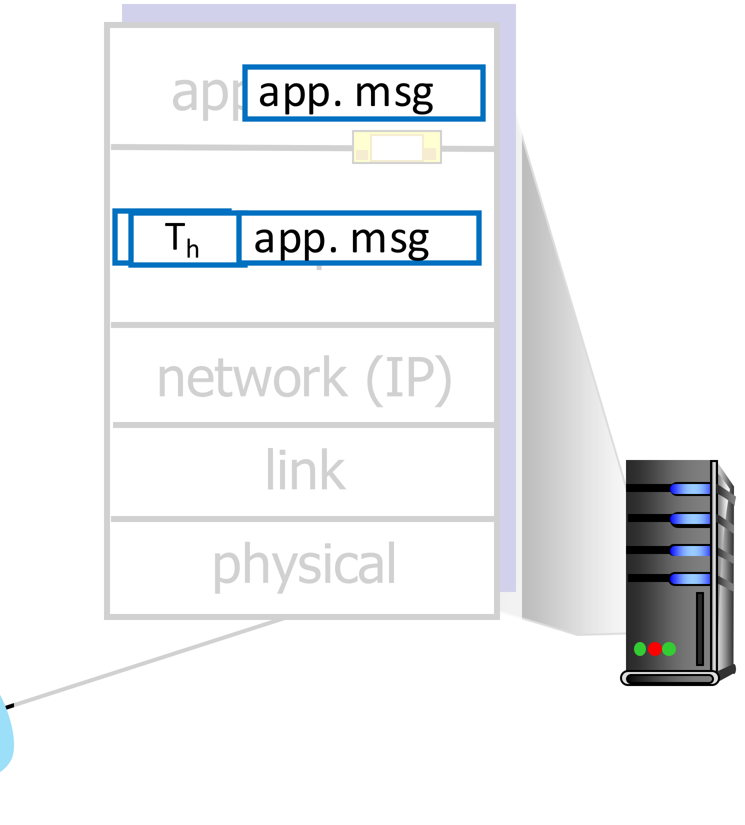


Transport Layer Actions

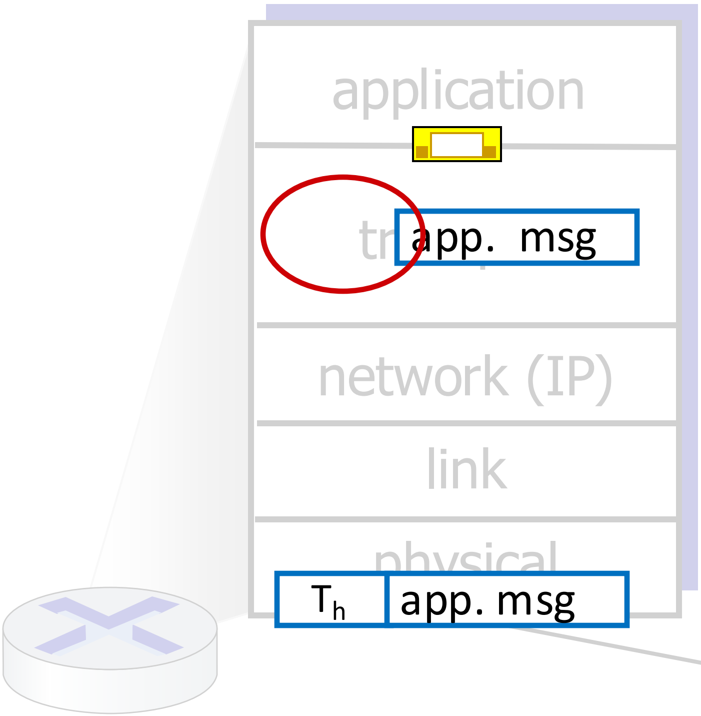


Sender:

- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to IP

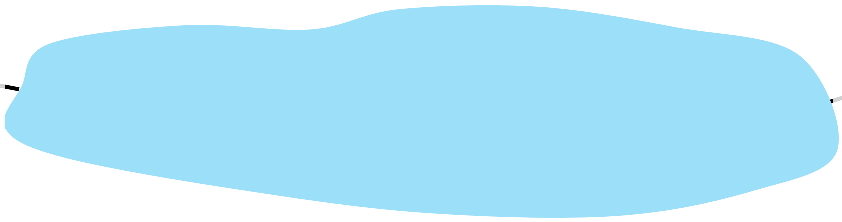


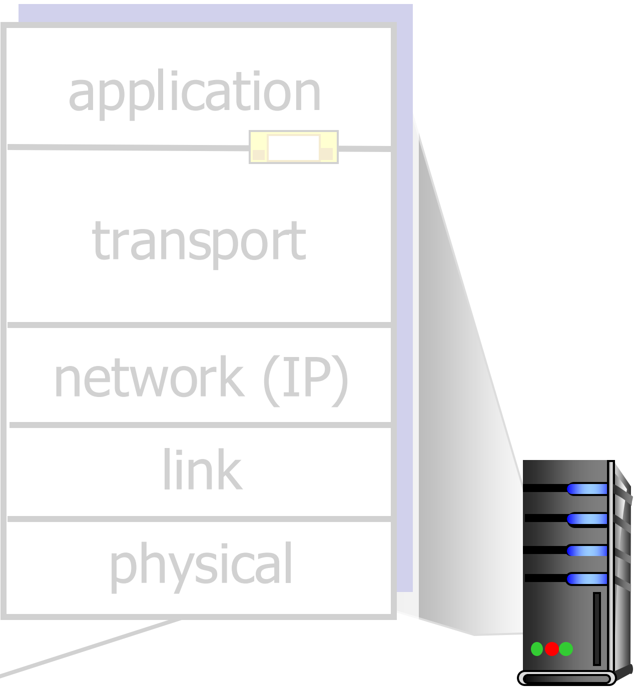
Transport Layer Actions



Receiver:

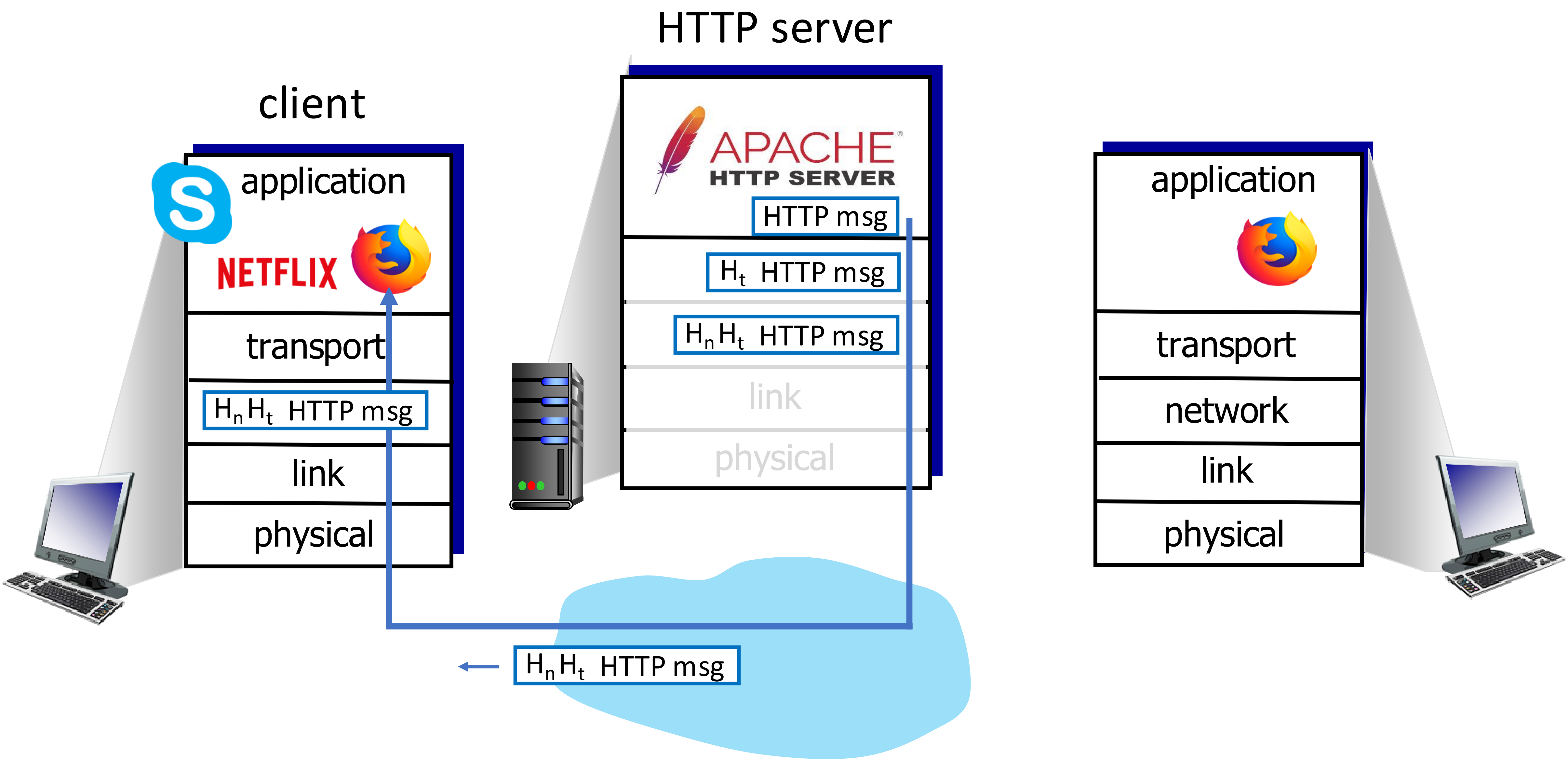
- receives segment from IP
- checks header values
- extracts application-layer message
- demultiplexes message up to application via socket



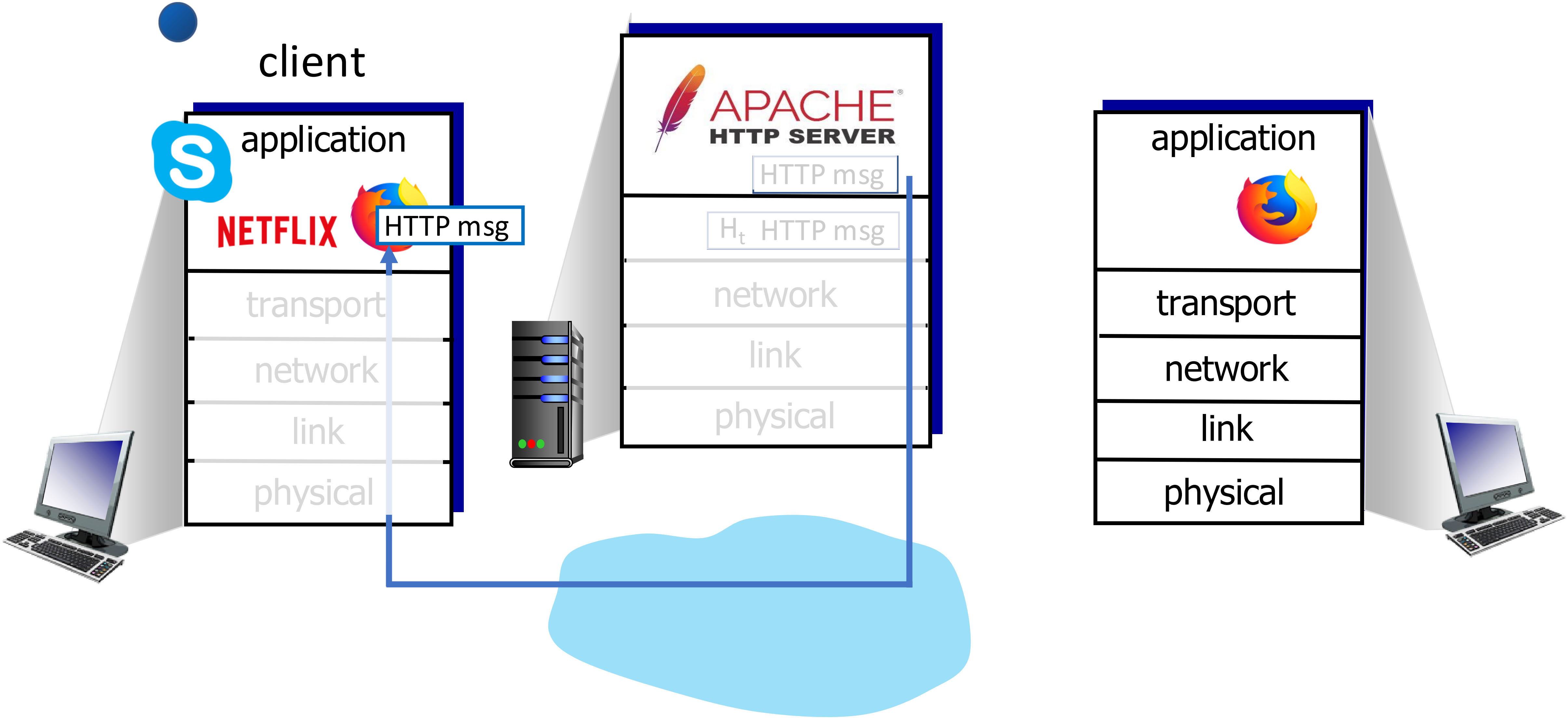


Transport-layer services

- Multiplexing/demultiplexing
- Reliable delivery
- Flow control
- Congestion control

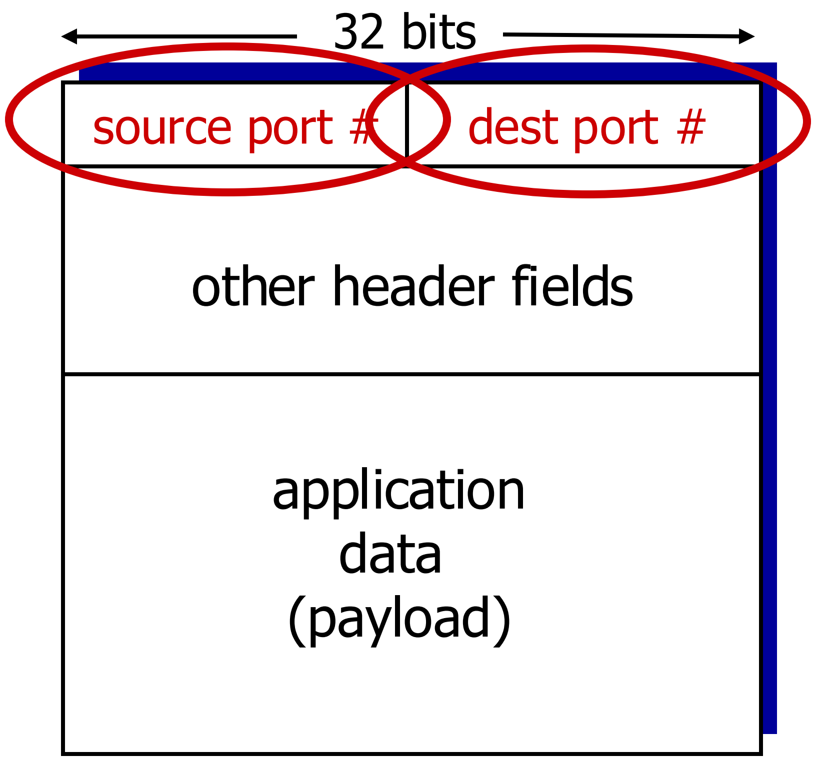

HTTP server
client
application
NETFLIX
transport
$H_n H_t$ HTTP msg
link
physical
APACHE
HTTP SERVER
HTTP msg
H_t HTTP msg
$H_n H_t$ HTTP msg
link
physical
$H_n H_t$ HTTP msg
application
transport
network
link
physical

Q: how did transport layer know to deliver message to Firefox browser process rather then Netflix process or Skype process?



How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

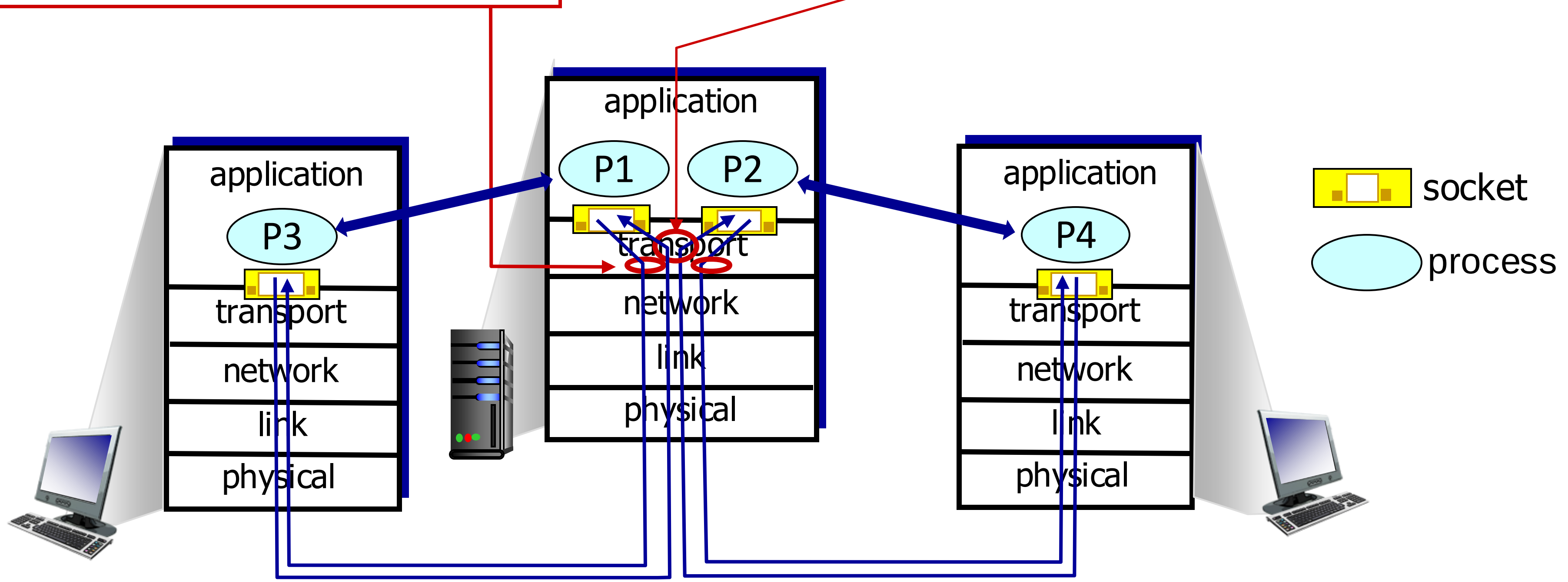
Multiplexing/demultiplexing

multiplexing as sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing as receiver:

use header info to deliver received segments to correct socket



Multiplexing/Demultiplexing

- Implemented using logical ports
- Difference in UDP/TCP multiplexing
- UDP or connection-less: Uses 2-tuple
  - (Dst IP, Dst Port) of the incoming packet aka bind (IP, port)
- TCP or connection-oriented: Uses 4-tuple
  - (Src IP, Src Port, Dst IP, Dst Port)

UDP: User Datagram Protocol

- “no frills,” “bare bones” Internet transport protocol
- “best effort” service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- No connection establishment delay. Makes it particularly suitable for applications involving short transactions:
  - DNS
  - SNMP
  - DHCP
- Provides applications with the flexibility to implement only a subset of features
  - E.g., reliability is not needed for video conferencing applications

UDP: User Datagram Protocol [RFC 768]

INTERNET STANDARD

RFC 768 J. Postel
ISI
28 August 1980

User Datagram Protocol

Introduction

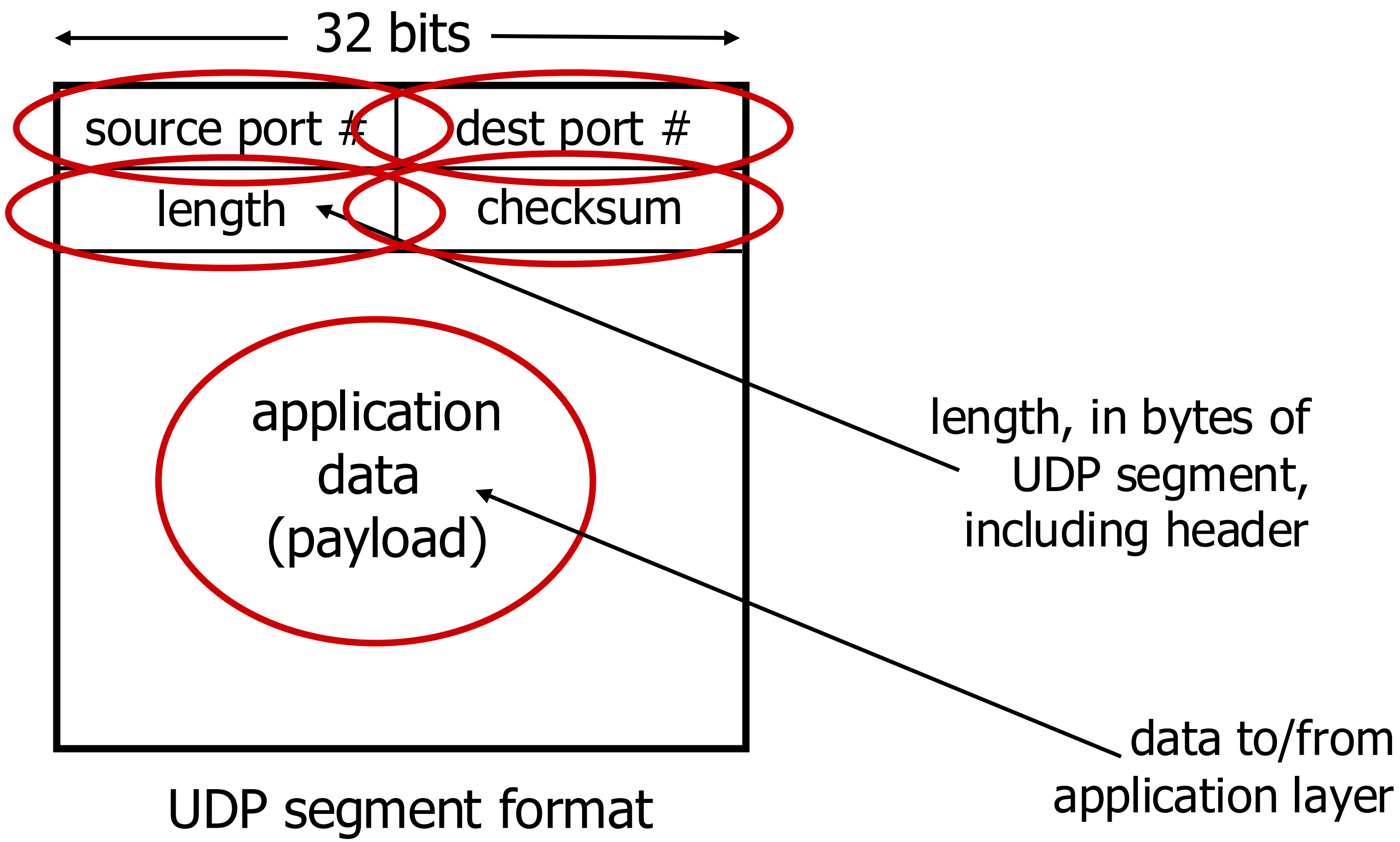
This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...
```

UDP segment header

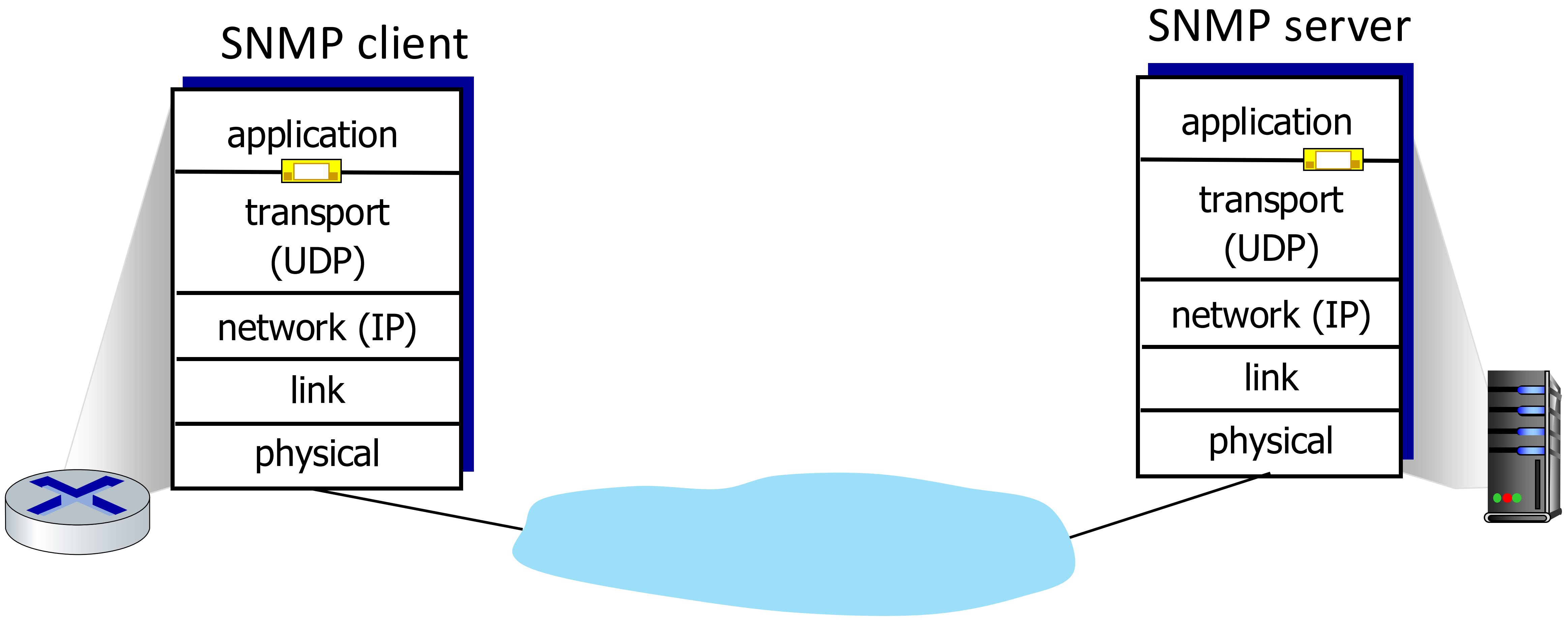


UDP segment format

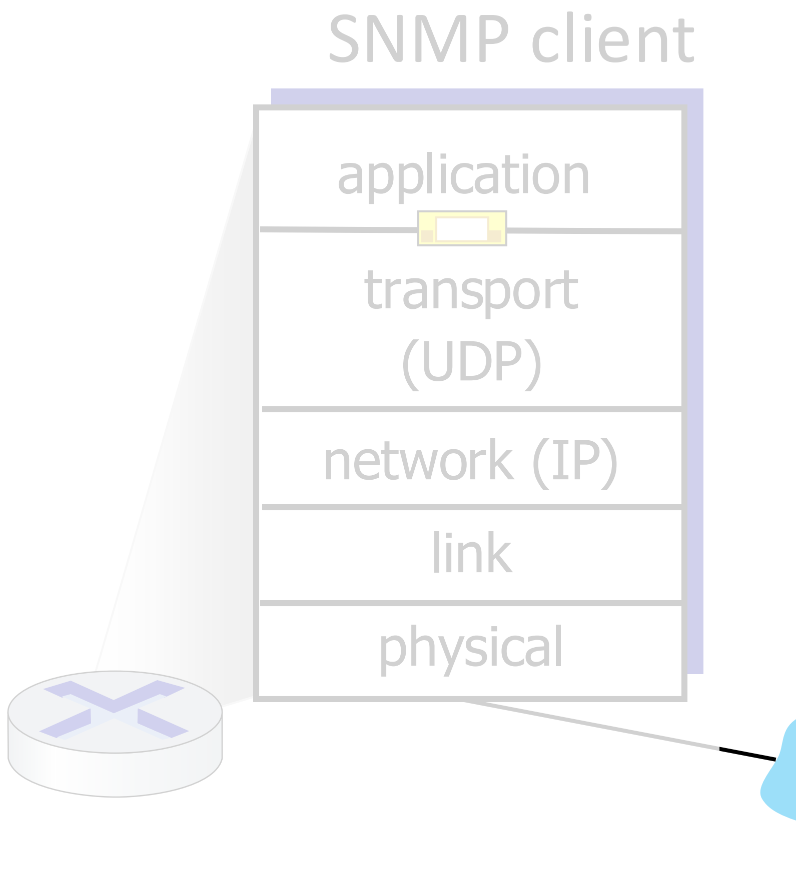
```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|          source address           |
+--------+--------+--------+--------+
|        destination address        |
+--------+--------+--------+--------+
|  zero  |protocol|   UDP length    |
+--------+--------+--------+--------+
```

Pseudo-header

UDP: Transport Layer Actions

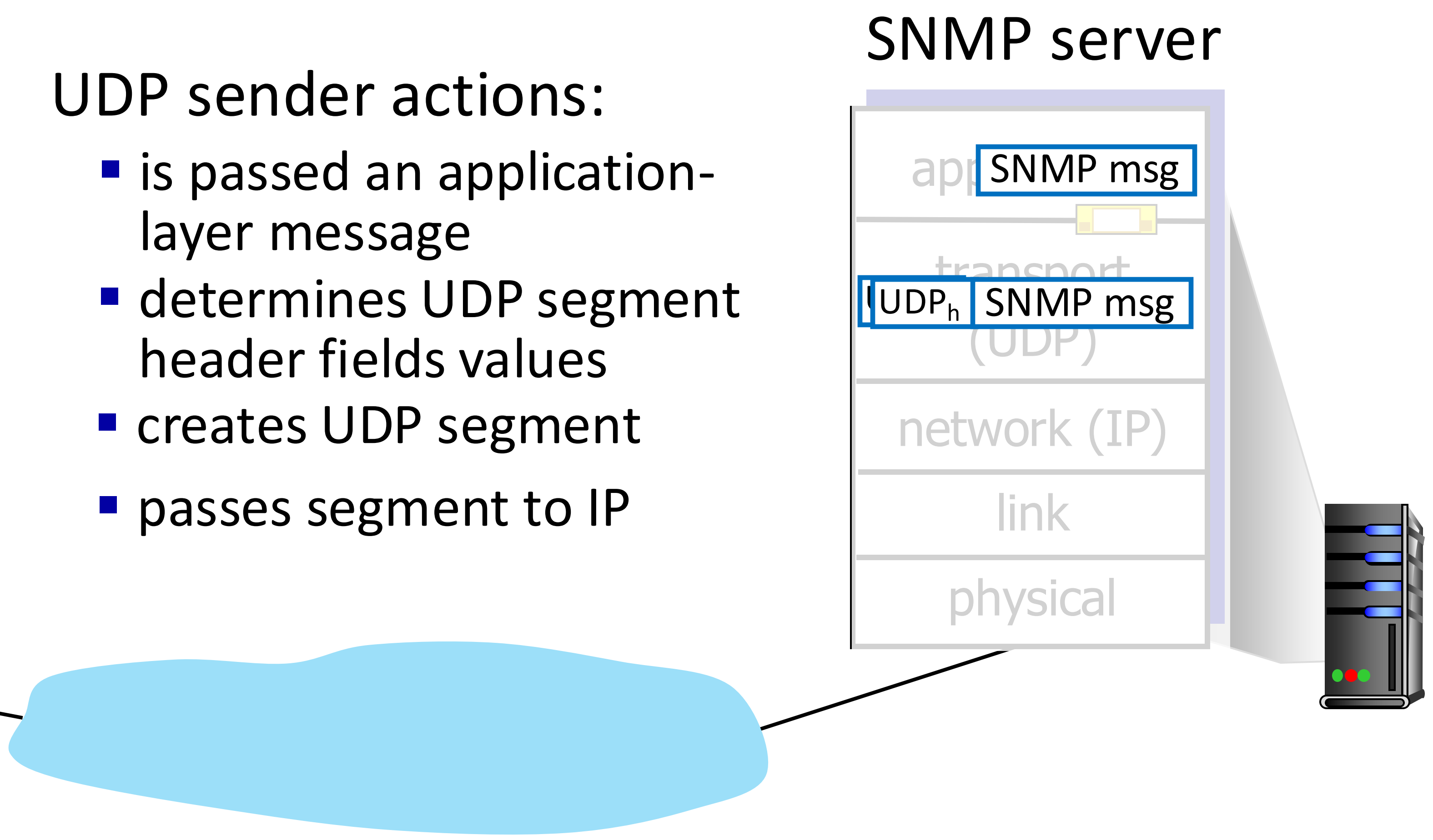


UDP: Transport Layer Actions

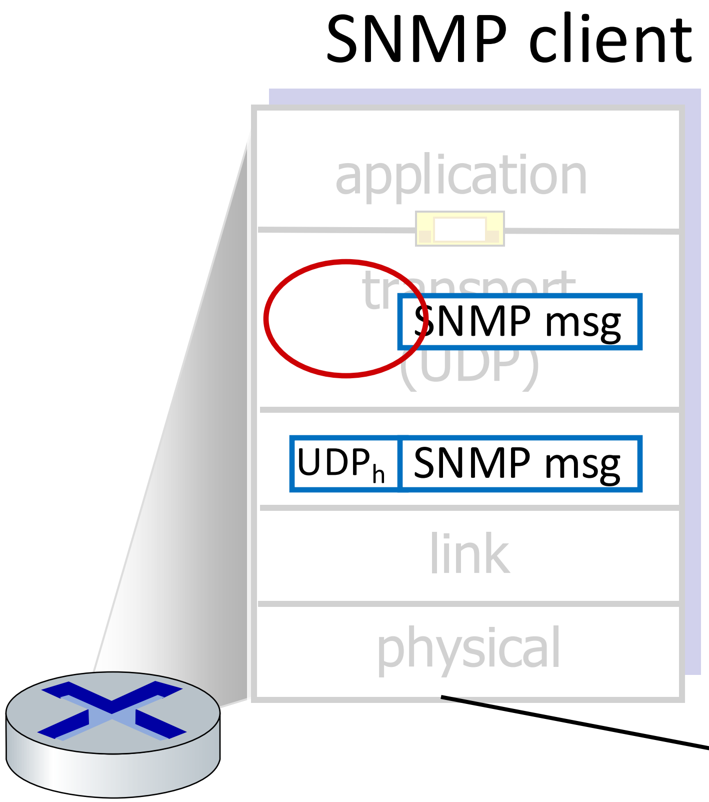


UDP sender actions:

- is passed an application-layer message
- determines UDP segment header fields values
- creates UDP segment
- passes segment to IP

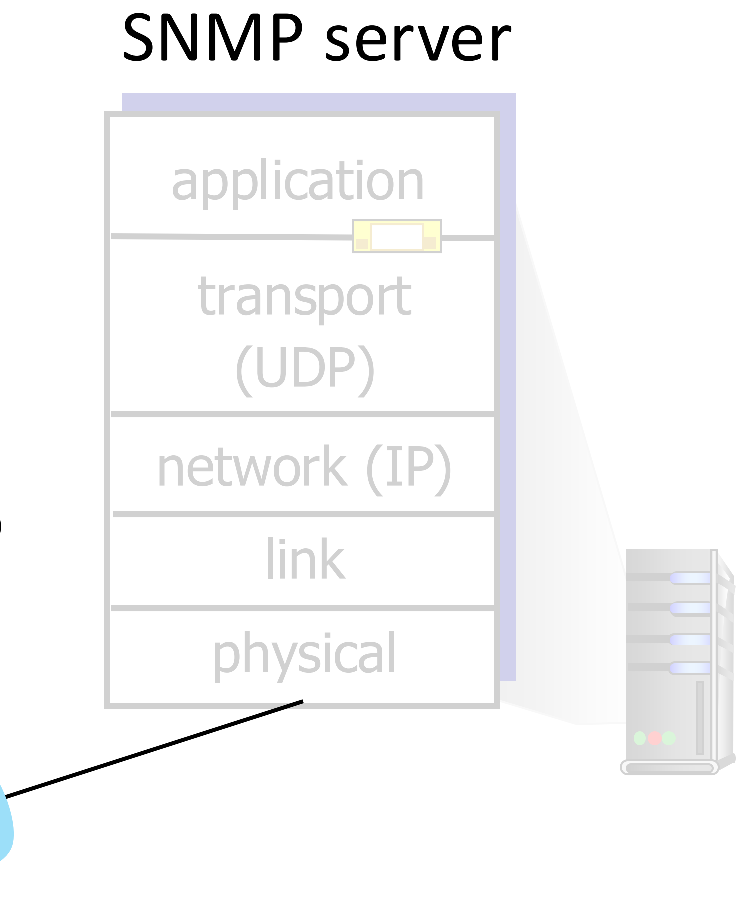


UDP: Transport Layer Actions



UDP receiver actions:

- receives segment from IP
- checks UDP checksum header value
- extracts application-layer message
- demultiplexes message up to application via socket



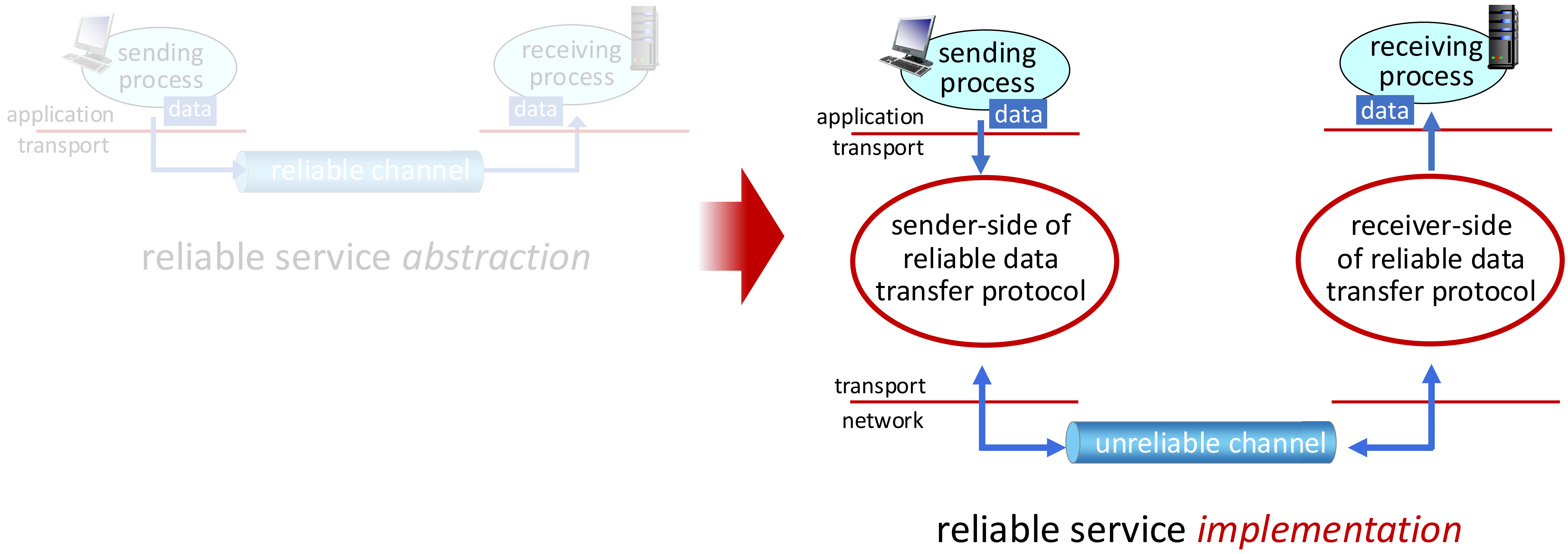
Transport-layer services

- Multiplexing/demultiplexing
- **Reliable delivery**
- Flow control
- Congestion control

Principles of reliable data transfer

reliable service *abstraction*

Principles of reliable data transfer



Reliability

- Error correction codes
- Automatic Repeat reQuest (ARQ)

Error correction code

- Also known as Forward Error Correction
- Using 2D parity

  Can detect *and* correct errors (without retransmission!)

  - detect *and correct* single bit errors
- Always useful?
  - When cost of retransmissions are high
  - When there are frequent bit errors

row parity

column parity

$$\begin{array}{ccc|c}
d_{1,1} & \cdots & d_{1,j} & d_{1,j+1} \\
d_{2,1} & \cdots & d_{2,j} & d_{2,j+1} \\
\cdots & \cdots & \cdots & \cdots \\
d_{i,1} & \cdots & d_{i,j} & d_{i,j+1} \\
\hline
d_{i+1,1} & \cdots & d_{i+1,j} & d_{i+1,j+1}
\end{array}$$

no errors:

$$\begin{array}{ccccc|c}
1 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 \\
\hline
1 & 0 & 1 & 0 & 1 & 0
\end{array}$$

detected and correctable single-bit error:

$$\begin{array}{ccccc|c}
1 & 0 & 1 & 0 & 1 & 1 \\
1 & \color{red}{0} & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 \\
\hline
1 & 0 & 1 & 0 & 1 & 0
\end{array}$$

parity error

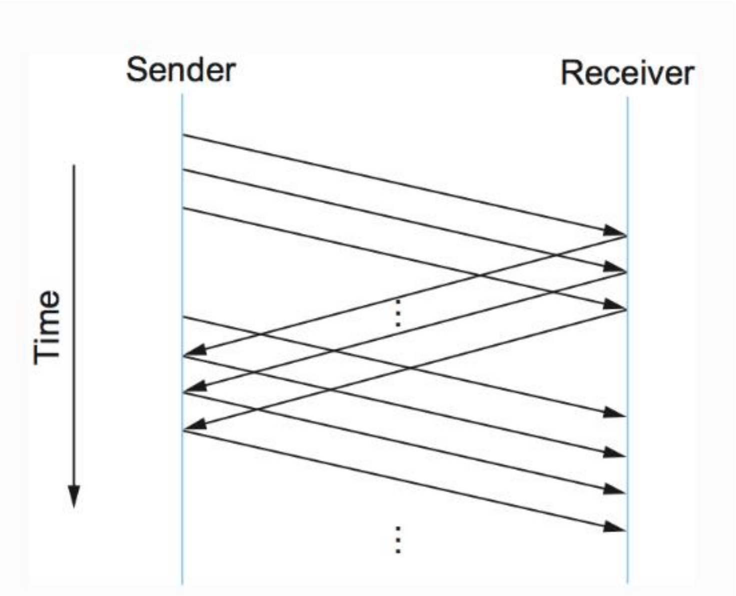
ARQ Protocol: Stop and Wait

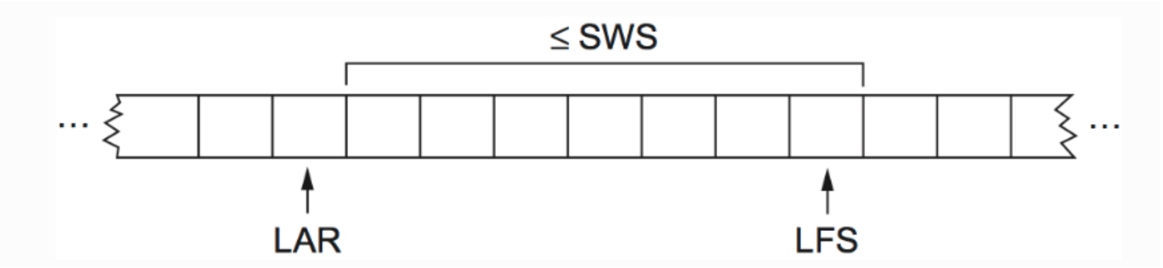
- Transmit one frame, wait for an acknowledgement
  - If no ack and timer expires, resend

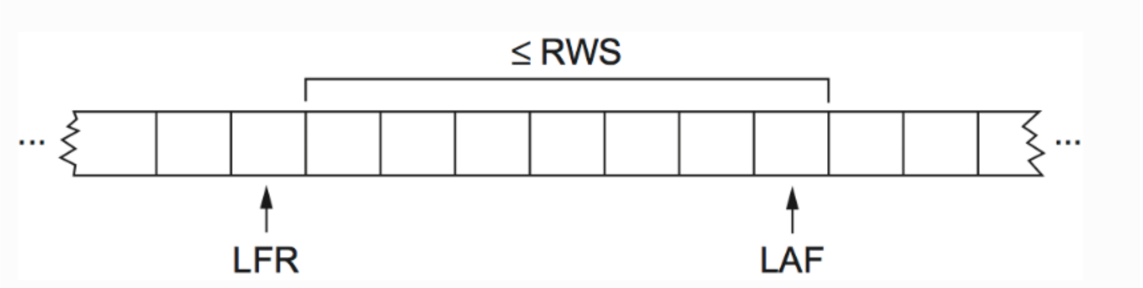
Stop and Wait

- Transmit one frame, wait for an acknowledgement
  - If no ack and timer expires, resend
- How to handle duplicate frames?
  - Sequence numbers for duplicate frames
- Any limitation?
  - Under-utilization of link
  - Example, 4 Mbps link, RTT – 10ms, Frame size – 1 KB
  - How to achieve full-link utilization?
    - Bandwidth delay product

Sliding Window Protocol







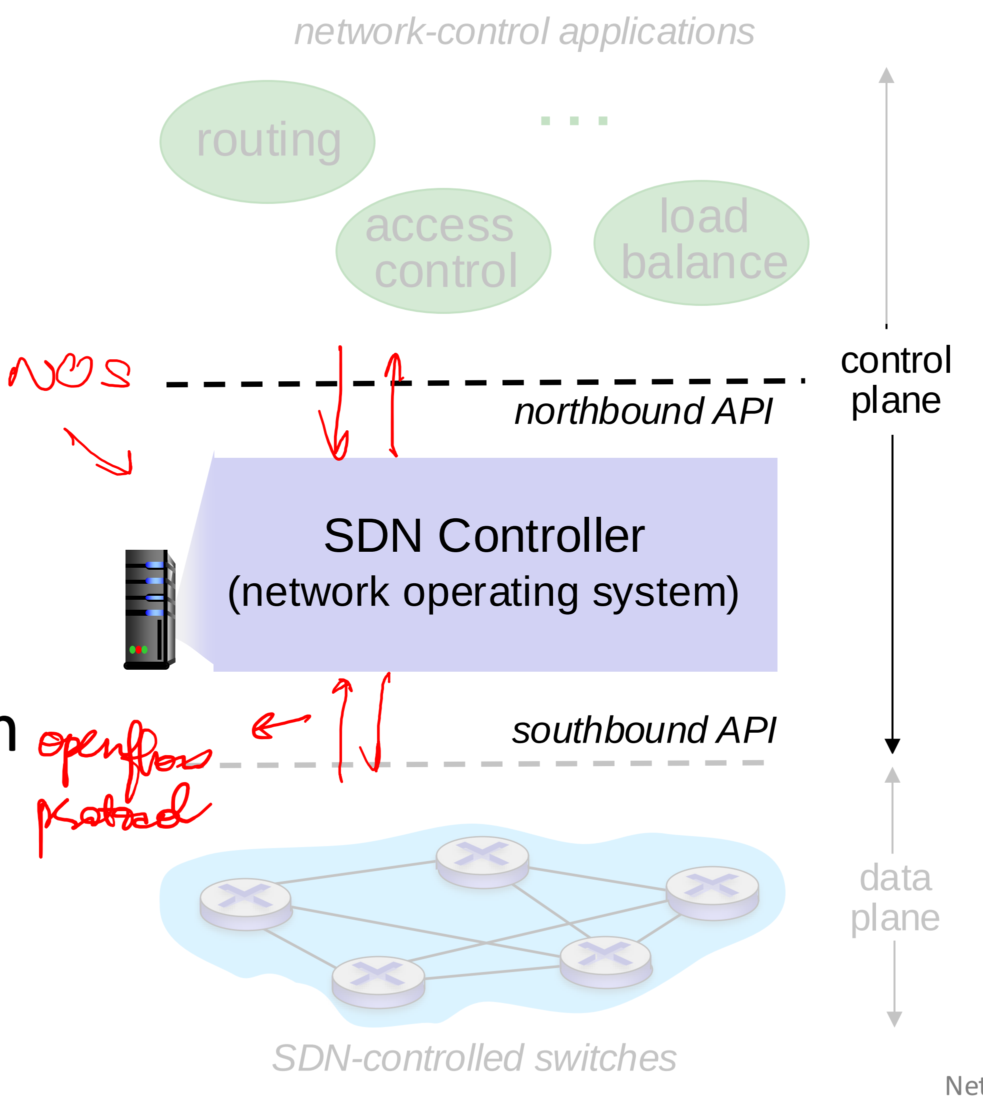
Summary

- Transport-layer services
- Multiplexing/demultiplexing
- UDP
- Reliability

Attendance

Recap: SDN Controller

- maintains network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness

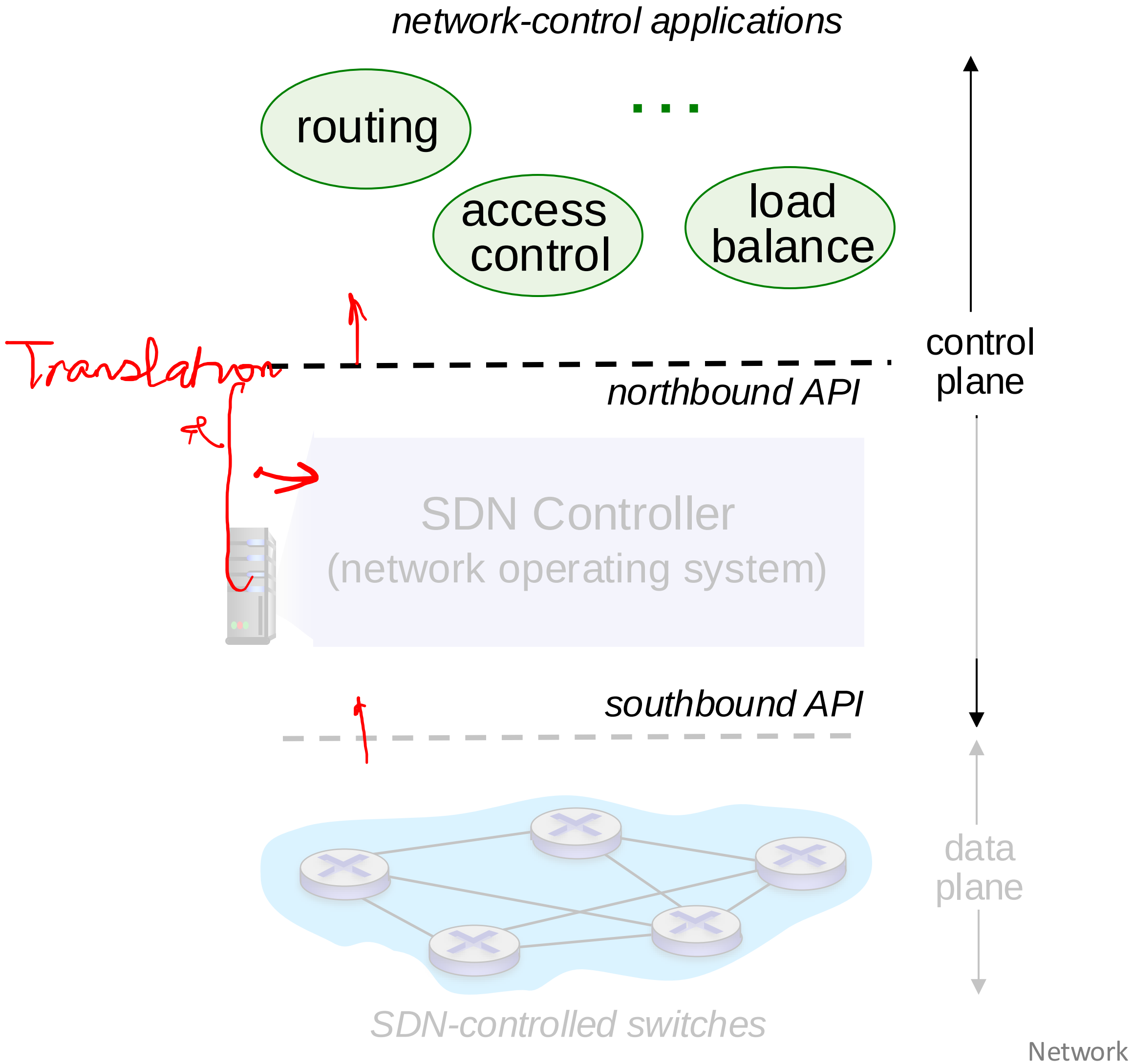


protocol → application

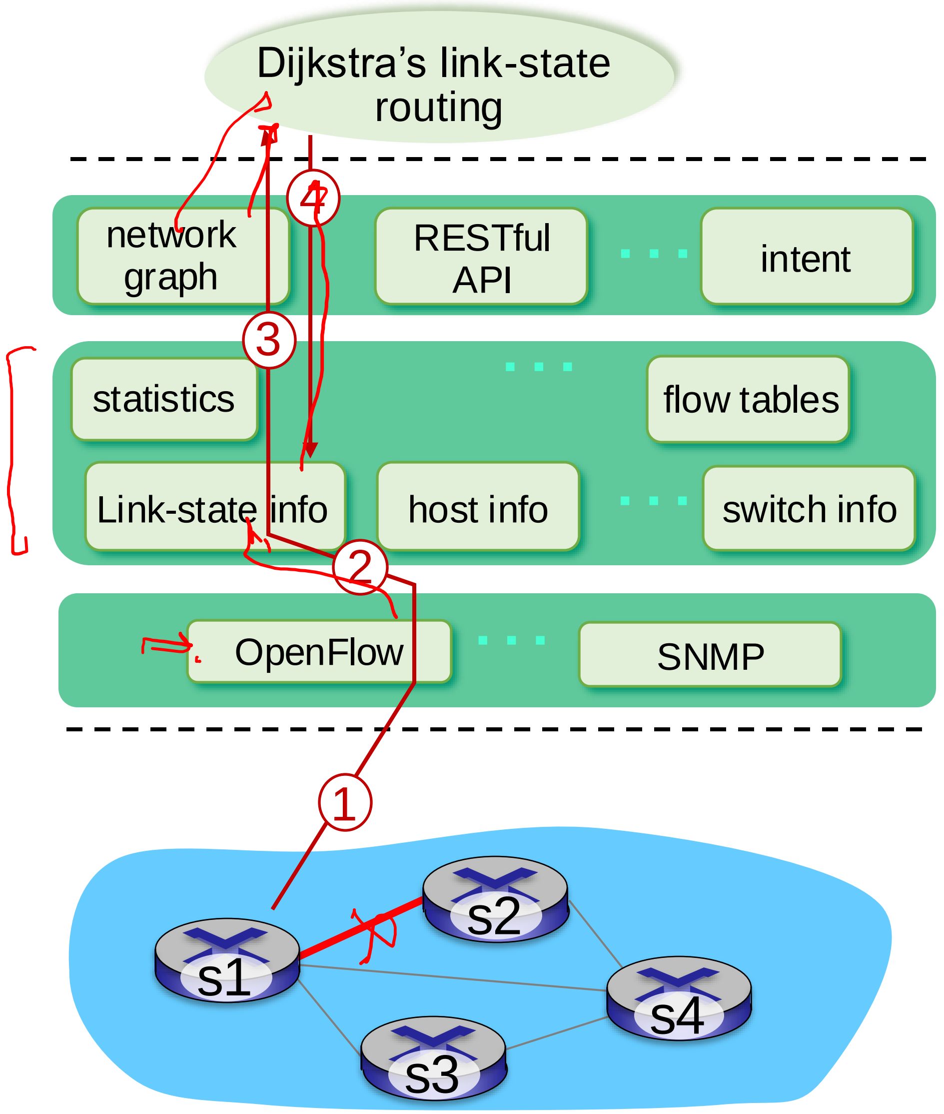
Software defined networking (SDN)

- operators don't "program" switches by creating/sending OpenFlow messages directly.
- Instead use higher-level abstraction at controller
- "brains" of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller

Intent-driven network

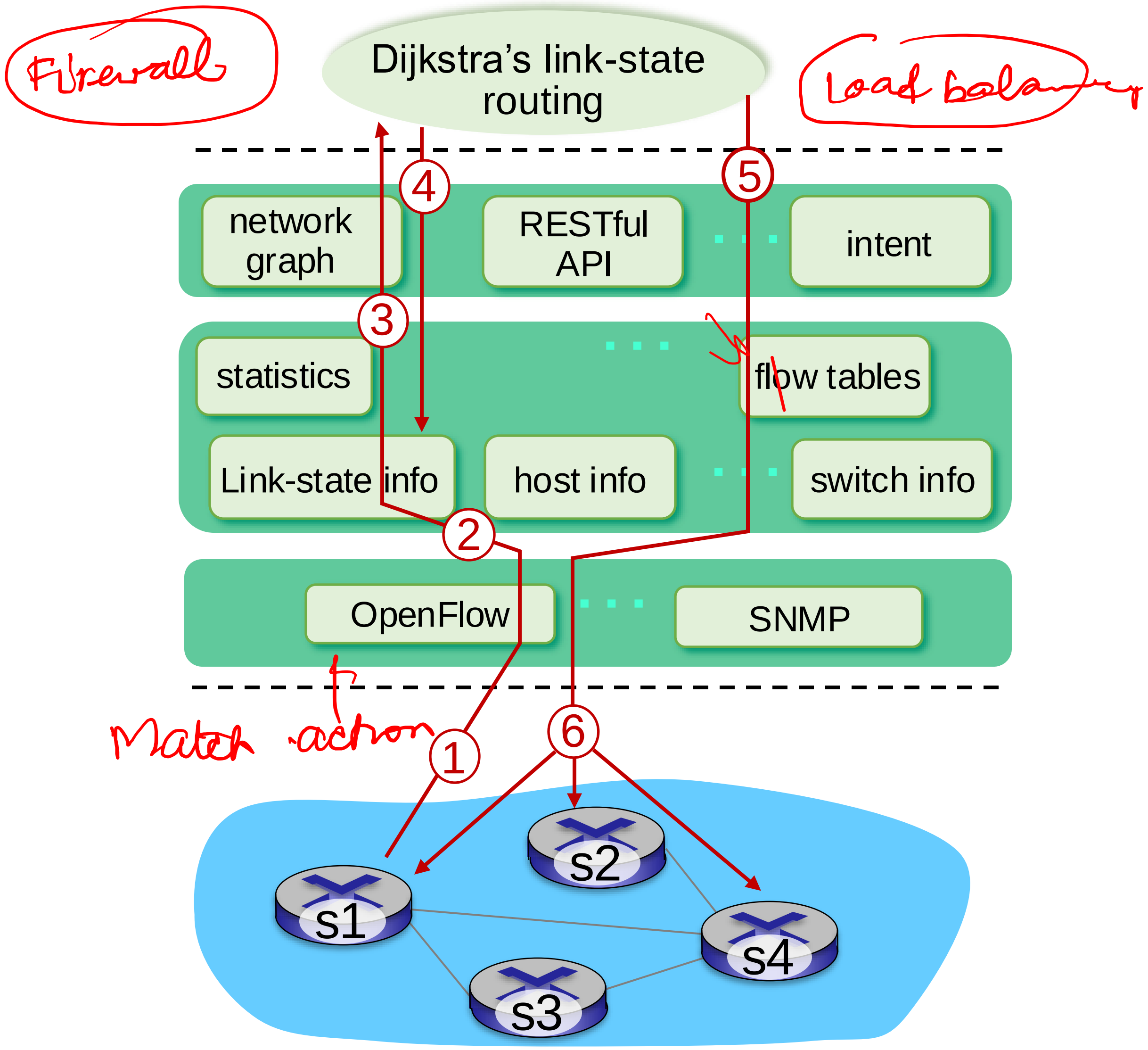


SDN: control/data plane interaction example



1. S1, experiencing link failure uses OpenFlow port status message to notify controller
2. SDN controller receives OpenFlow message, updates link status info
3. Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.
4. Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

SDN: control/data plane interaction example



⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed

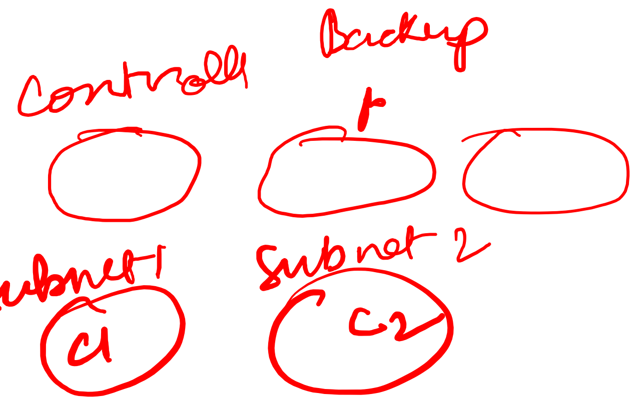
⑥ controller uses OpenFlow to install new tables in switches that need updating

SDN: Key Challenges

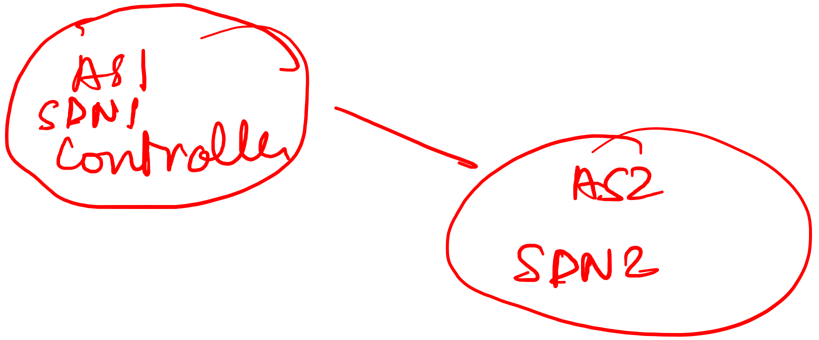
- Hardening the control plane
  - Scalability
  - Reliability
  - Consistency
  - Security

→ Real-time → logically centralized

→ Distributed systems



- Internet-scaling: beyond a single AS (?)

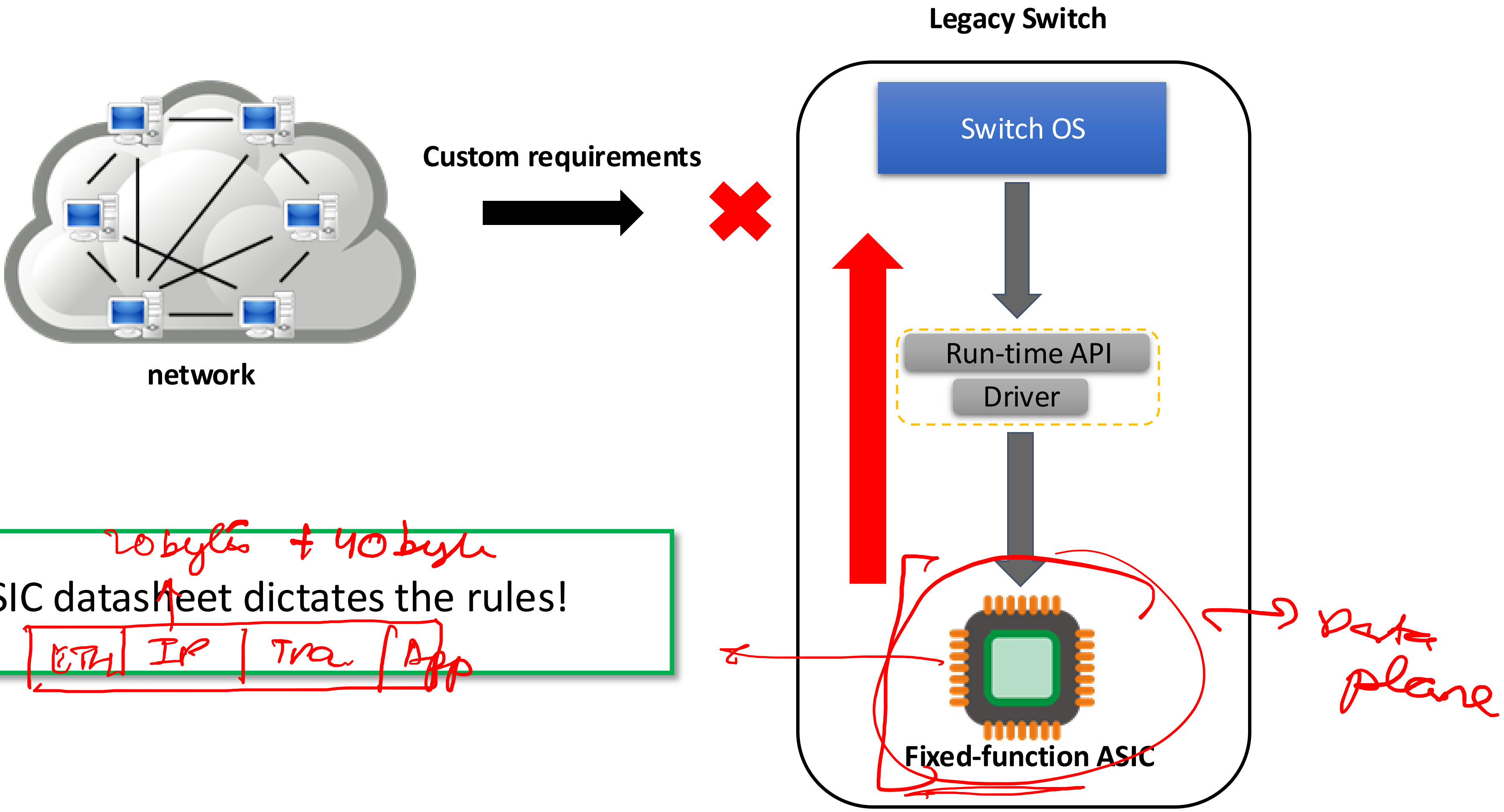


What else is programmable in the network?

- Programmable data plane
- Network function virtualization (NFV)

making middleboxes programmable

Status Quo: Bottom-up design

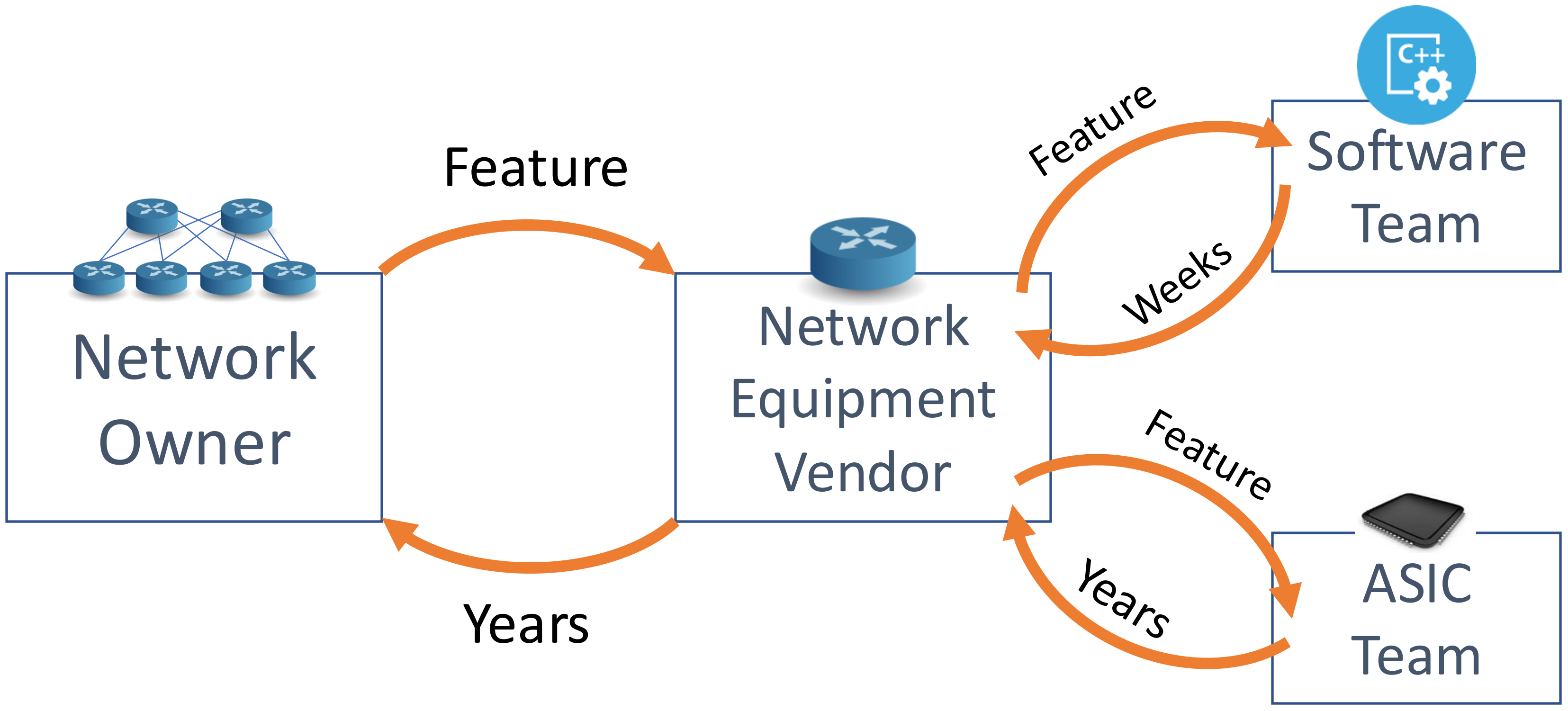


congestion aware
ETH | IP

20 bytes + 40 bytes

The ASIC datasheet dictates the rules!

ETH | IP | Tra | App



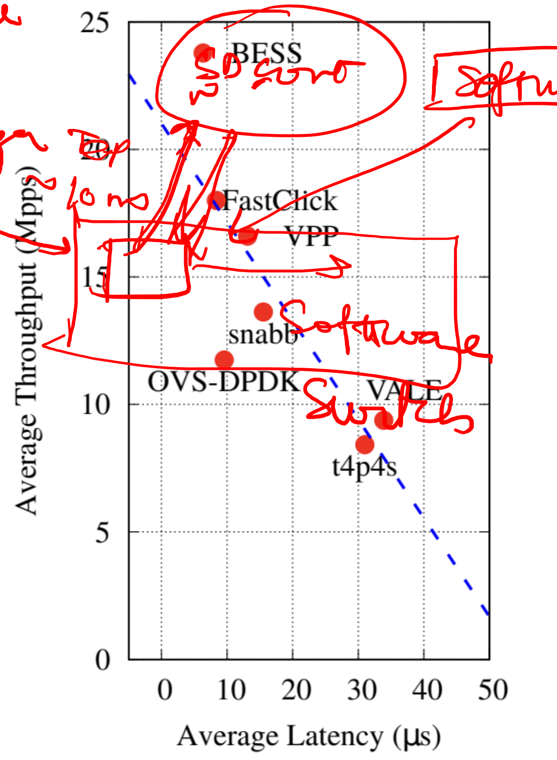
Feature
Network Owner
Years
Network Equipment Vendor
Feature
Weeks
Software Team
C++
Feature
Years
ASIC Team

How to make Data Plane Programmable?

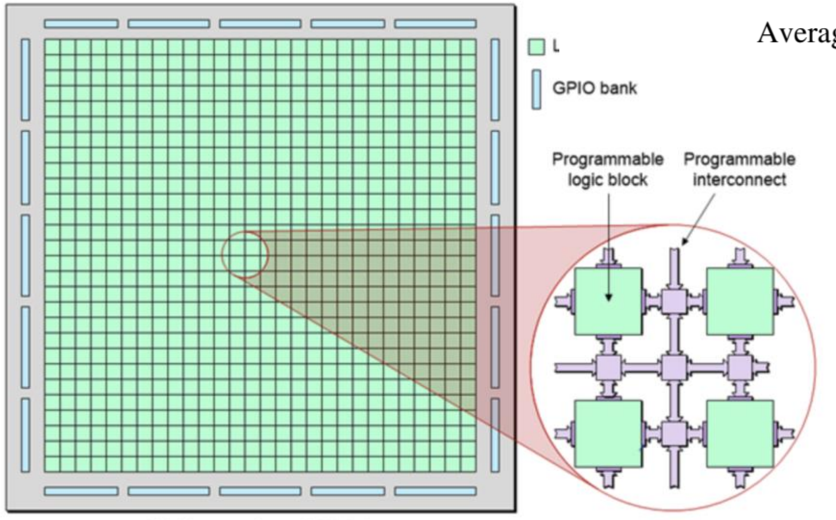
- Move the data plane to software
  - Generally, too slow for data plane functions
  - Any optimization techniques?
    - Parallelism across multiple servers and cores
    - Optimizations in NUMA
    - Fast I/O …

App
CPU
OS
eBPF
NIO

I/O
Multiple CPU
Multiple level of storage



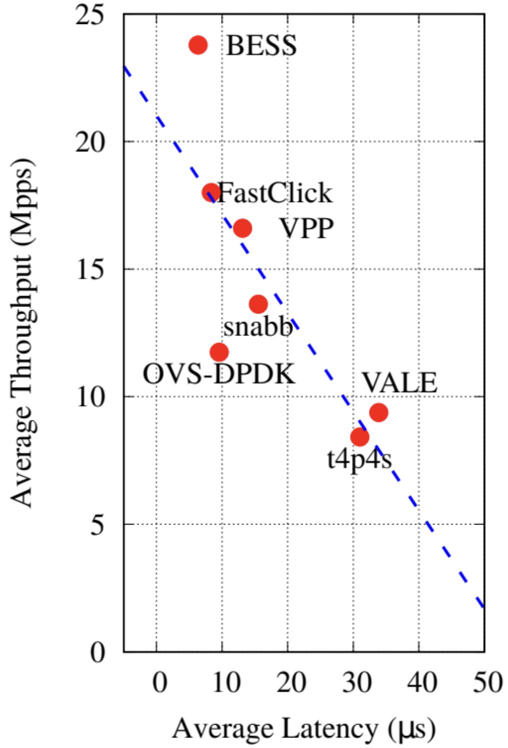
- What about programmable hardware?
  - FPGA: but costly, power-hungry, slower



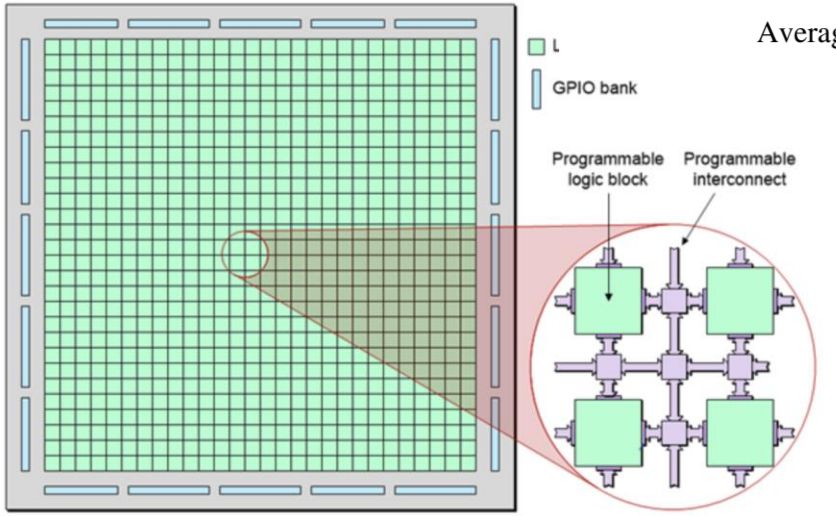
Bird's-eye view of FPGA

How to make Data Plane Programmable?

- Move the data plane to software
  - Generally, too slow for data plane functions
  - Any optimization techniques?
    - Parallelism across multiple servers and cores
    - Optimizations in NUMA
    - Fast I/O …



- What about programmable hardware?
  - FPGA: but costly, power-hungry, slower



Bird's-eye view of FPGA

What about Programmable Hardware?

“Programmable swi[...] 10-100x slower than fixe[...]on switches. They are mor[...]sive and consume [...]e power.”

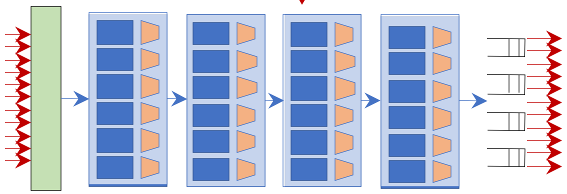
Not true any more!

onventional wisdom in networking

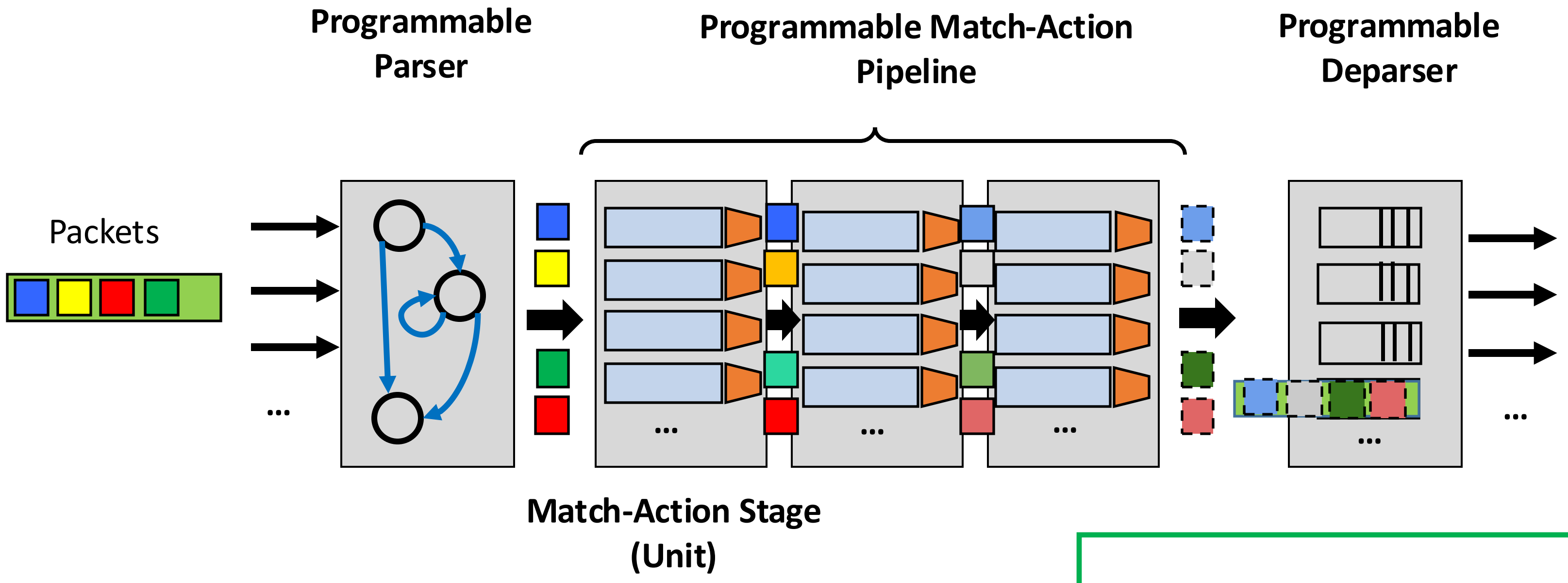
Domain Specific Processors

Computers	Graphics	Signal Processing	Machine Learning		Networking
Java	OpenCL	Matlab	TensorFlow	>>>	Language
Compiler	Compiler	Compiler	Compiler		Compiler
					?
CPU	GPU	DSP	TPU		

Domain Specific Processors

Computers	Graphics	Signal Processing	Machine Learning		Networking
Java	OpenCL	Matlab	TensorFlow	>>>	P4
Compiler	Compiler	Compiler	Compiler		Compiler
					
CPU	GPU	DSP	TPU		PISA

Protocol Independent Switch Architecture



P4 describes PISA's behavior

Reducing complexity

switch.p4

Switch OS

IPv4 and IPv6 routing
- Unicast Routing
  - Routed Ports & SVI
  - VRF
- Unicast RPF
  - Strict and Loose
- ~~Multicast~~
  - ~~PIM-SM/DM & PIM-Bidir~~

Ethernet switching
- ~~VLAN Flooding~~
- MAC Learning & Aging
- STP state
- ~~VLAN Translation~~

Load balancing
- ~~LAG~~
- ECMP & WCMP
- Resilient Hashing
- ~~Flowlet Switching~~

Fast Failover
– LAG & ECMP

Tunneling
- IPv4 and IPv6 Routing & Switching
  - ~~IP in IP (6in4, 4in4)~~
  - VXLAN, NVGRE, GENEVE & GRE
  - ~~Segment Routing, ILA~~

~~MPLS~~
- ~~LER and LSR~~
- ~~IPv4/v6 routing (L3VPN)~~
- ~~L2 switching (EoMPLS, VPLS)~~
- ~~MPLS over UDP/GRE~~

ACL
- MAC ACL, IPv4/v6 ACL, RACL
- ~~QoS ACL, System ACL, PBR~~
- Port Range lookups in ACLs

QOS
- QoS Classification & marking
- ~~Drop profiles/WRED~~
- ~~RoCE v2 & FCoE~~
- CoPP (Control plane policing)

~~NAT and L4 Load Balancing~~

Security Features
- ~~Storm Control, IP Source Guard~~

Monitoring & Telemetry
- ~~Ingress Mirroring and Egress Mirroring~~
- Negative Mirroring
- ~~Sflow~~
- INT

Counters
- Route Table Entry Counters
- ~~VLAN/Bridge Domain Counters~~
- Port/Interface Counters

Protocol Offload
- BFD, OAM

Multi-chip Fabric Support
- ~~Forwarding, QOS~~

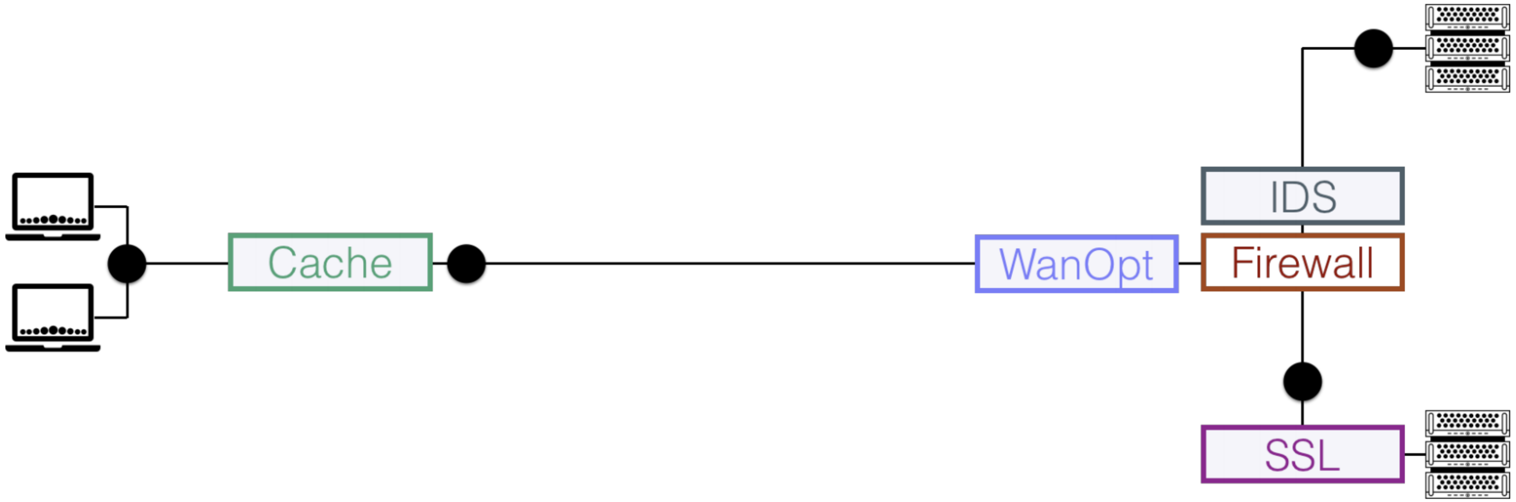
What else is programmable in the network?

- Programmable data plane

- **Network function virtualization**

Middleboxes

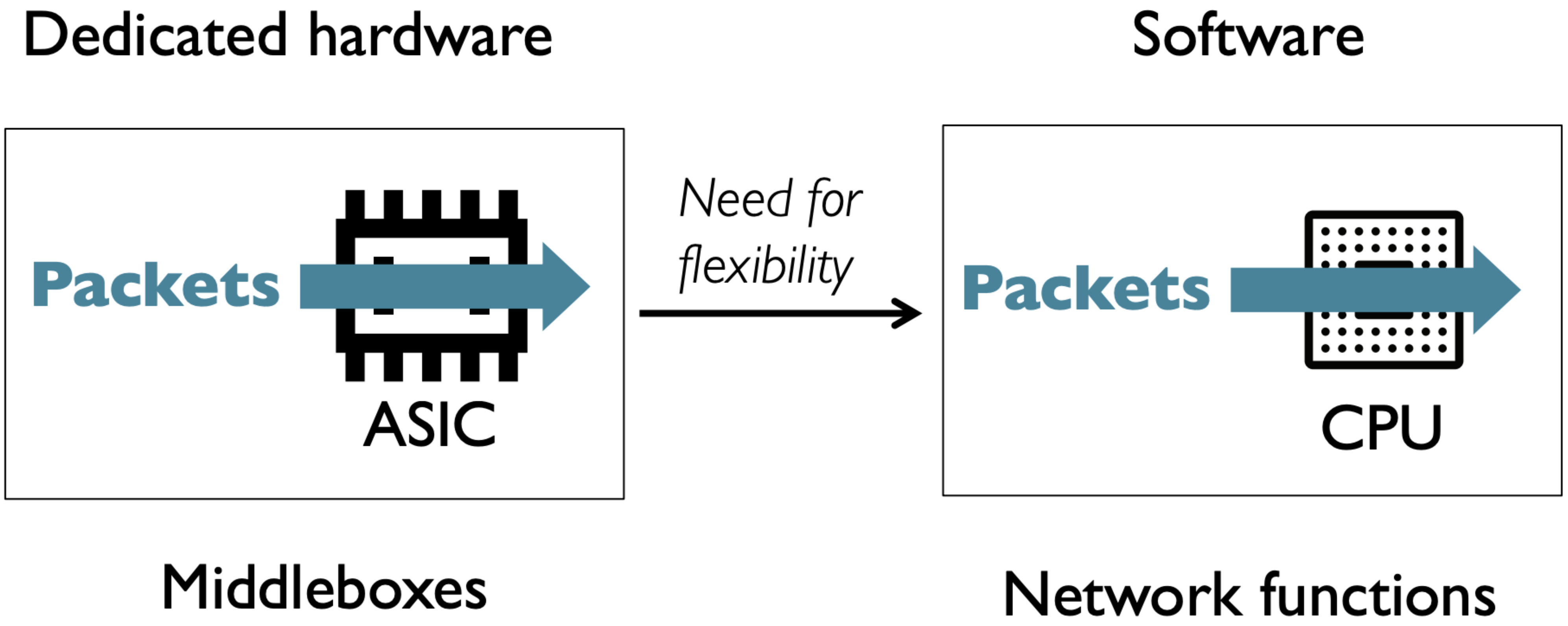
Data delivery is not the only required functionality.



Elements in the network path for security, performance enhancements etc.

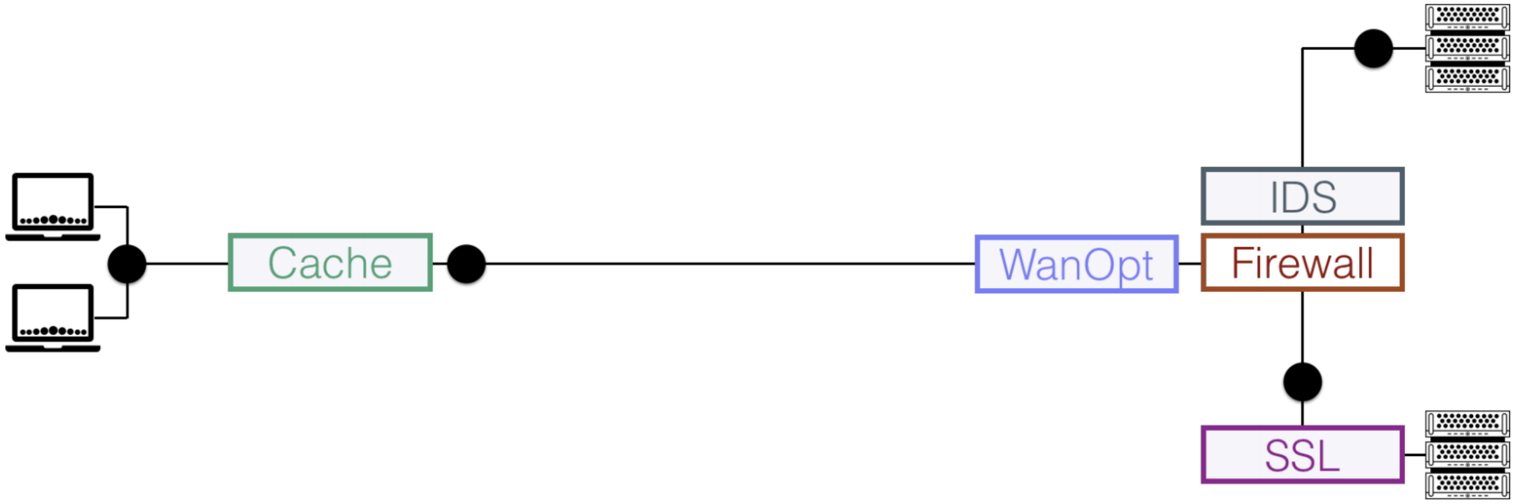
One-third of all network devices in enterprises are middleboxes!
Sherry et al., SIGCOMM'12

Evolution of Middleboxes

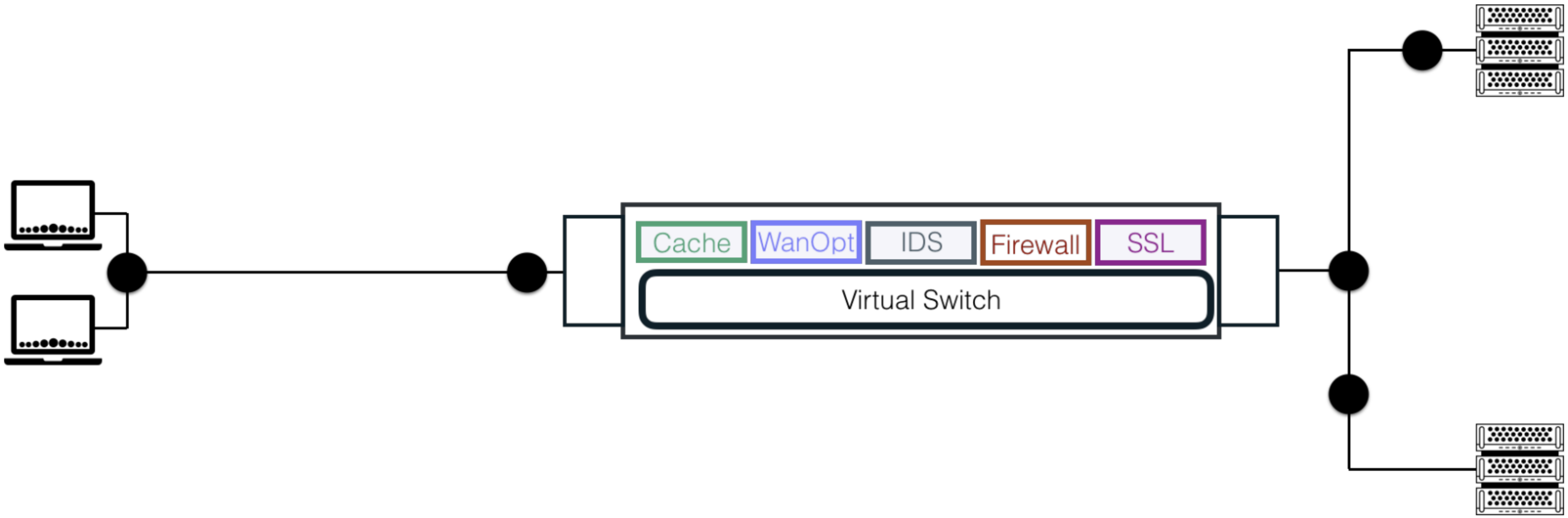


From Hardware Middleboxes..

Data delivery is not the only required functionality.



To Software Network Functions (NF)



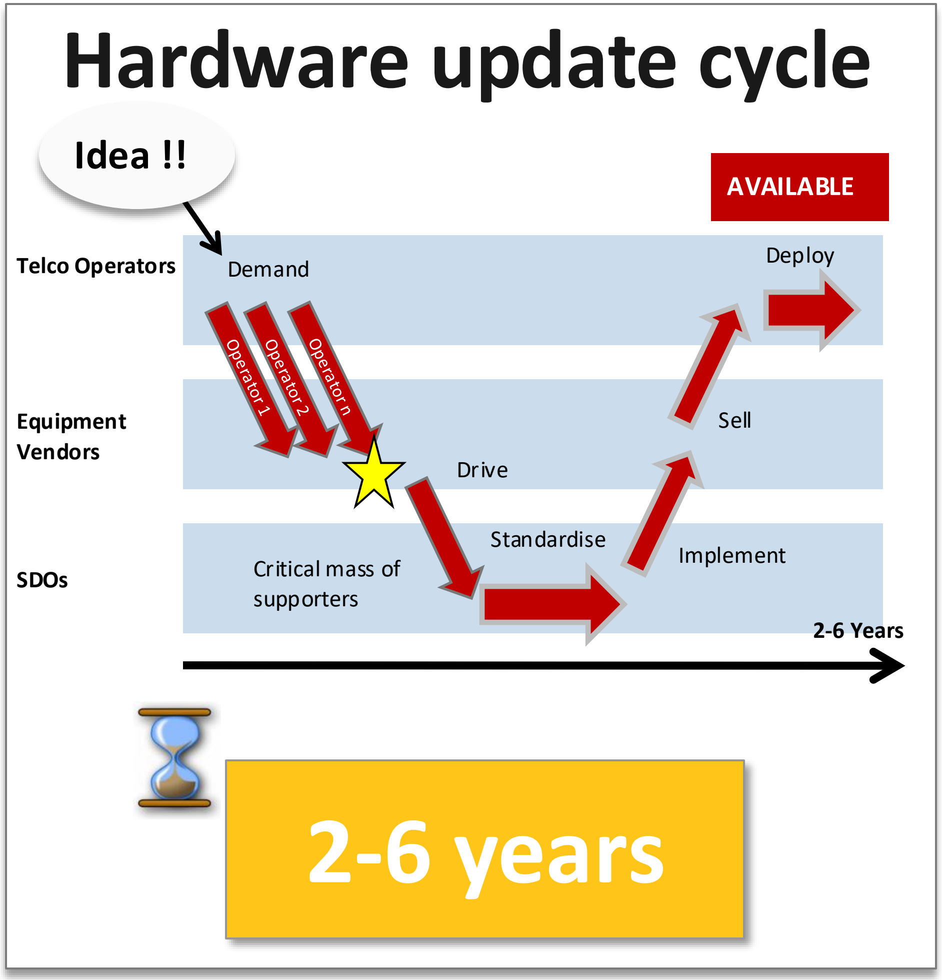
Primarily deployed in a VM
(Network Function Virtualization or NFV)

Functional Elements, not Middleboxes

- WAN Optimizer = Caching + Deduplication + Compression + Encryption + Forward Error Correction + Rate Limiter
- Application Firewall = IP Defragmenter + Application Detection Engine + Logger + Blocker
- IDS = IP Defragmenter + Preprocessing + Misuse Detection Engine + Logger

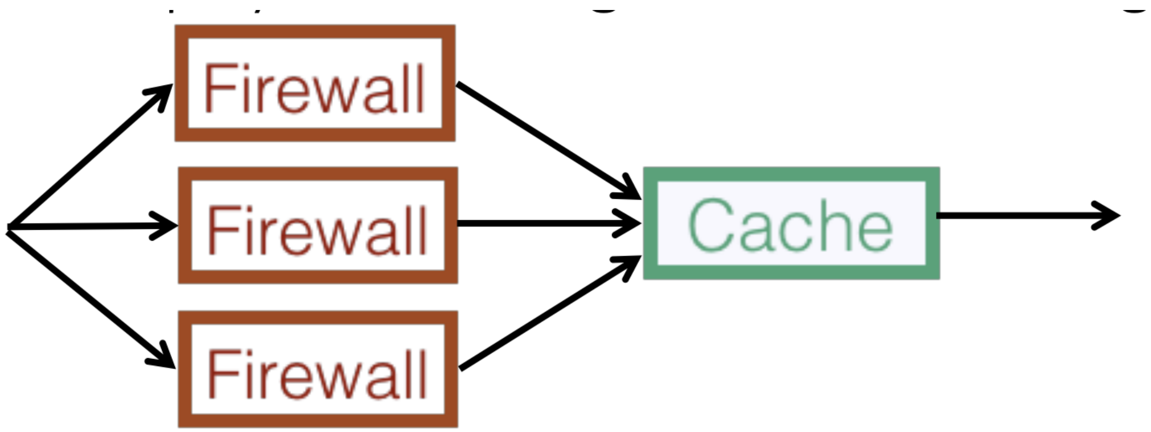
Why NFV?

- Softwarization leads to faster innovation



Why NFV?

- Softwarization leads to faster innovation
- Ease of deployment, configuration, and management
- Consolidation: Reduce number of hardware boxes in the network



Being adopted by both carriers and cloud providers

NFV Challenges

- Virtual network function management
  - Where and how to install network functions?
- Unpredictable (low) Performance
  - How to mitigate the overheads of virtualization?
- Fault Tolerance
  - How to handle recovery in case of faults?

Summary

- Increased programmability in the networks
  - Greater flexibility → Faster innovation
- Programmable control plane
  - SDN
- Programmable data plane
  - SDN-2 / P4
- Software middleboxes implemented in VMs
  - Network function virtualization (NFV)

Attendance