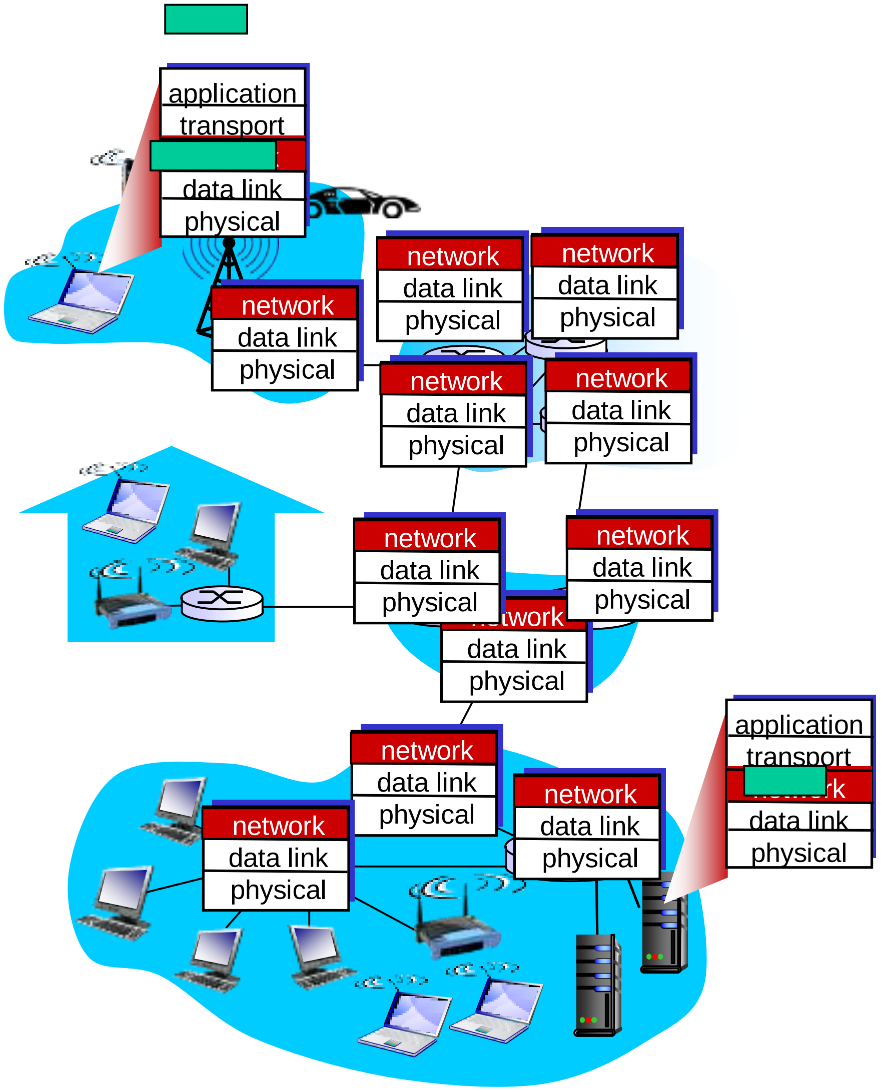
Outline

* Overview of Network layer
  - data plane
  - control plane
* What's inside a router
* Generalized Forward and SDN
  - Match
  - Action
  - OpenFlow examples of match-plus-action in action

Network layer

- transport segment from sending to receiving host
- on sending side encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in *every* host, router
- router examines header fields in all IP datagrams passing through it



key network-layer functions

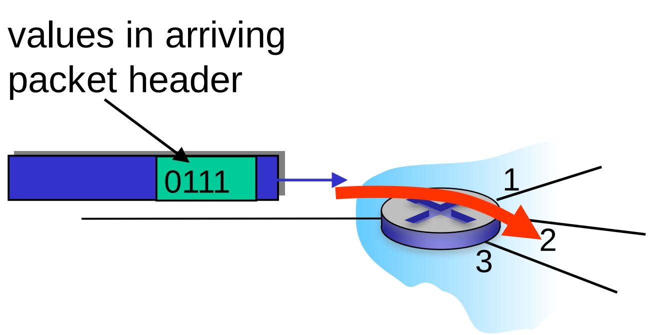
network-layer functions:

- *forwarding:* move packets from router's input to appropriate router output
- *routing:* determine route taken by packets from source to destination
  - *routing algorithms*

Network layer: data plane, control plane

Data plane

- local, per-router function
- determines how datagram arriving on router input port is forwarded to router output port
- forwarding function

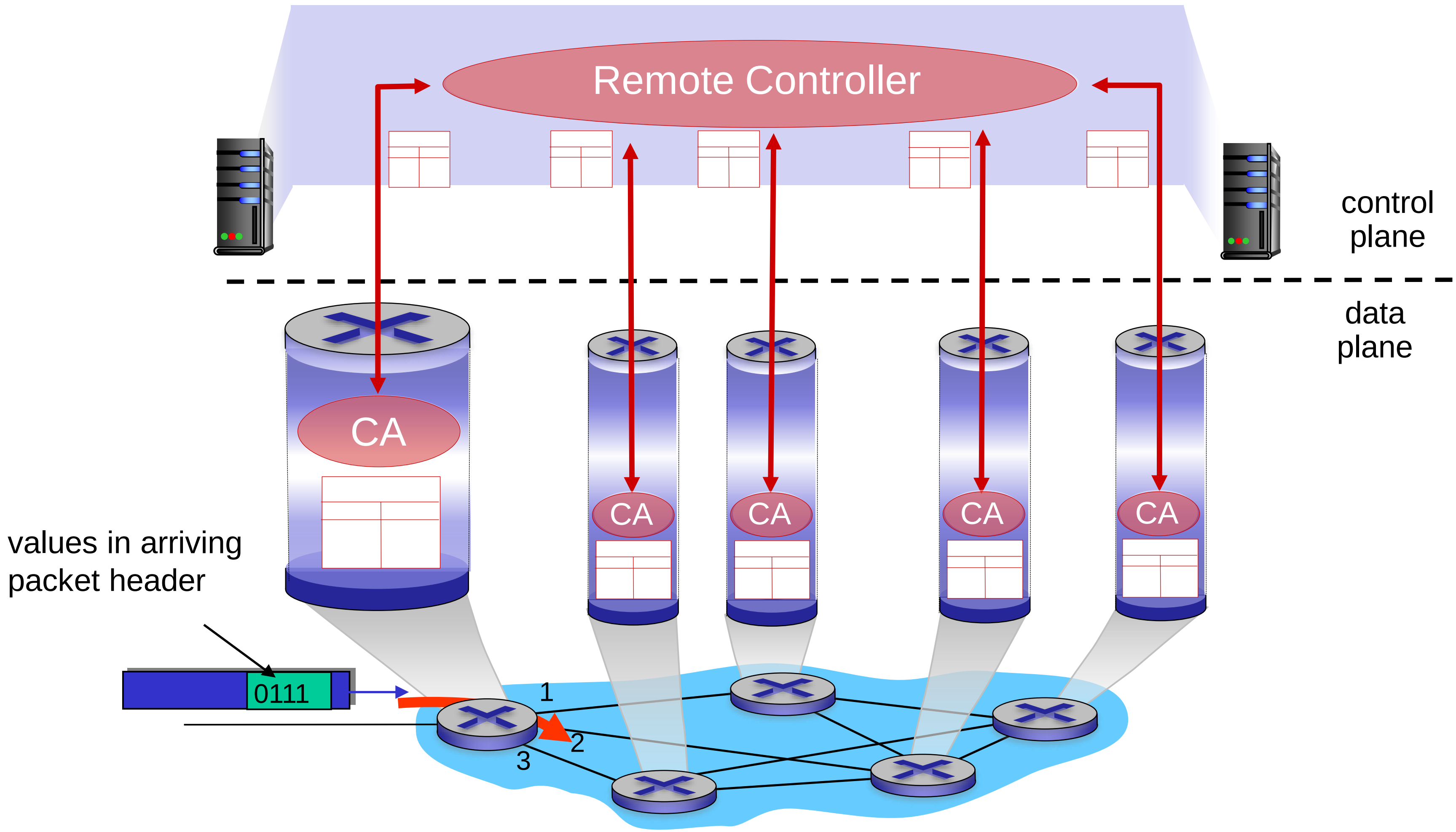


Control plane

- network-wide logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms:* implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers

Logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs)



Network service model

Q: What *service model* for "channel" transporting datagrams from sender to receiver?

example services for individual datagrams:

- guaranteed delivery
- guaranteed delivery with less than 40 msec delay

example services for a flow of datagrams:

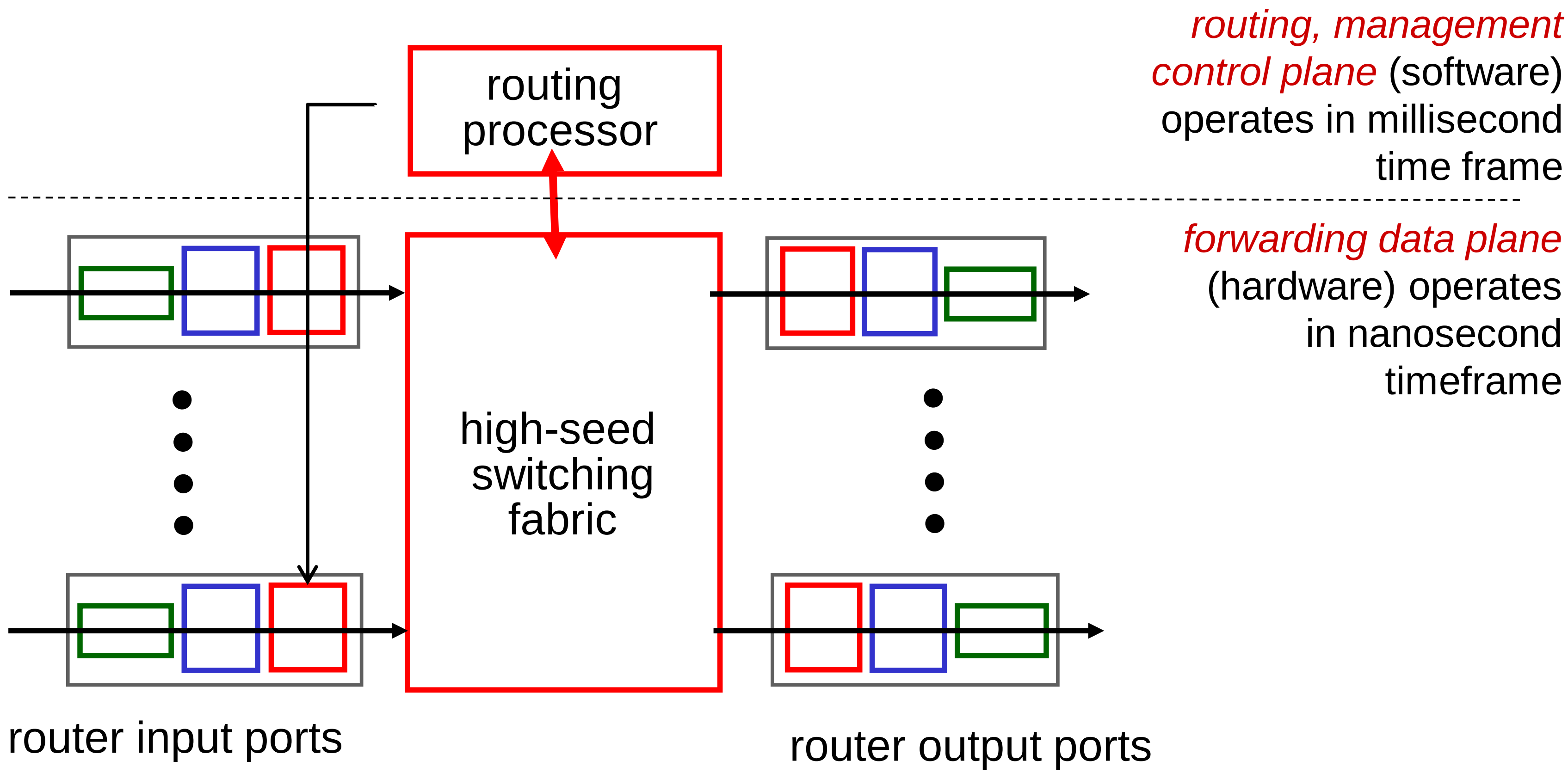
- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing

Outline

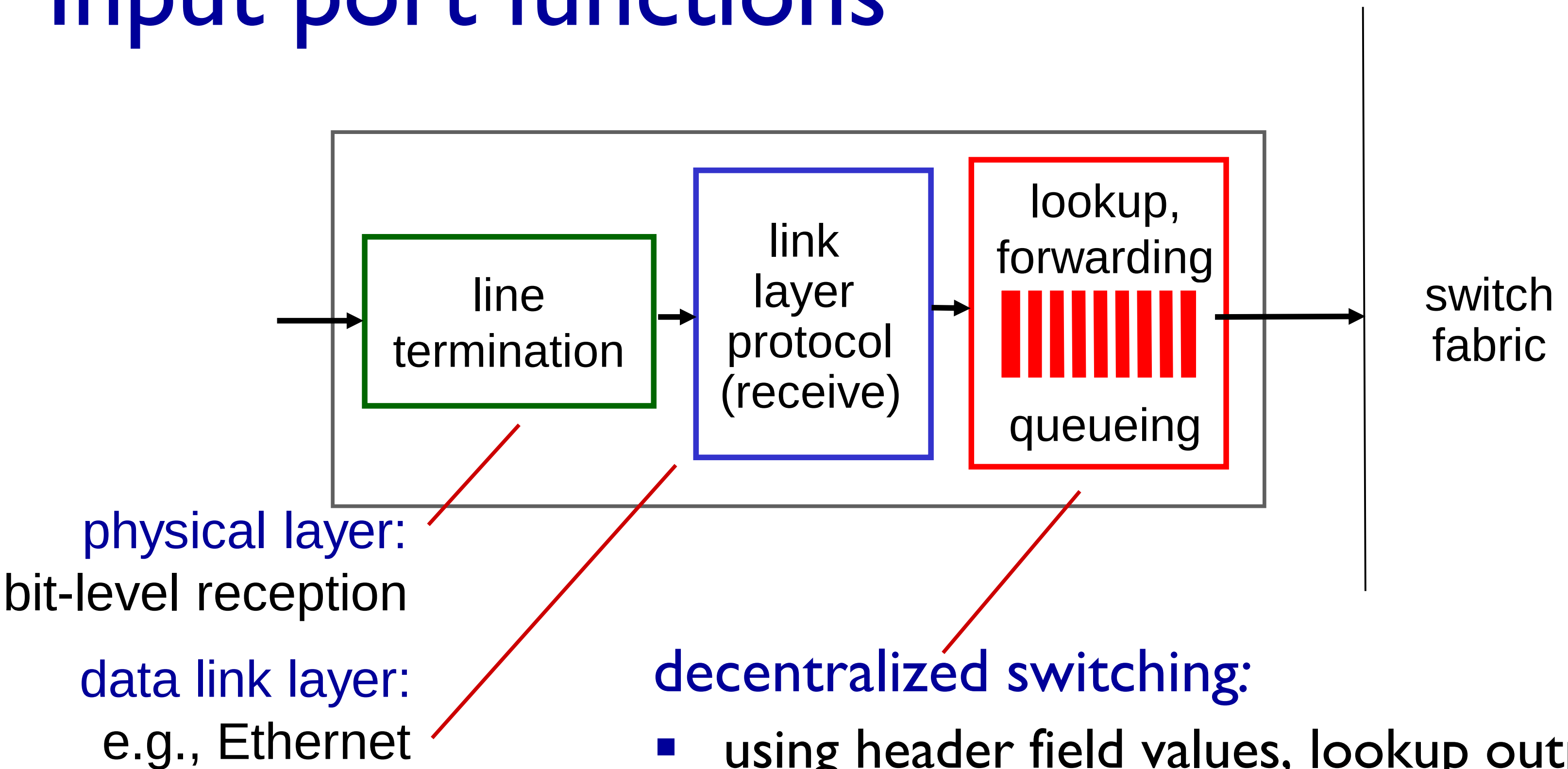
* Overview of Network layer
    • data plane
    • control plane
* What's inside a router
* Generalized Forward and SDN
    • Match
    • Action
    • OpenFlow examples of match-plus-action in action

Router architecture overview

- high-level view of generic router architecture:



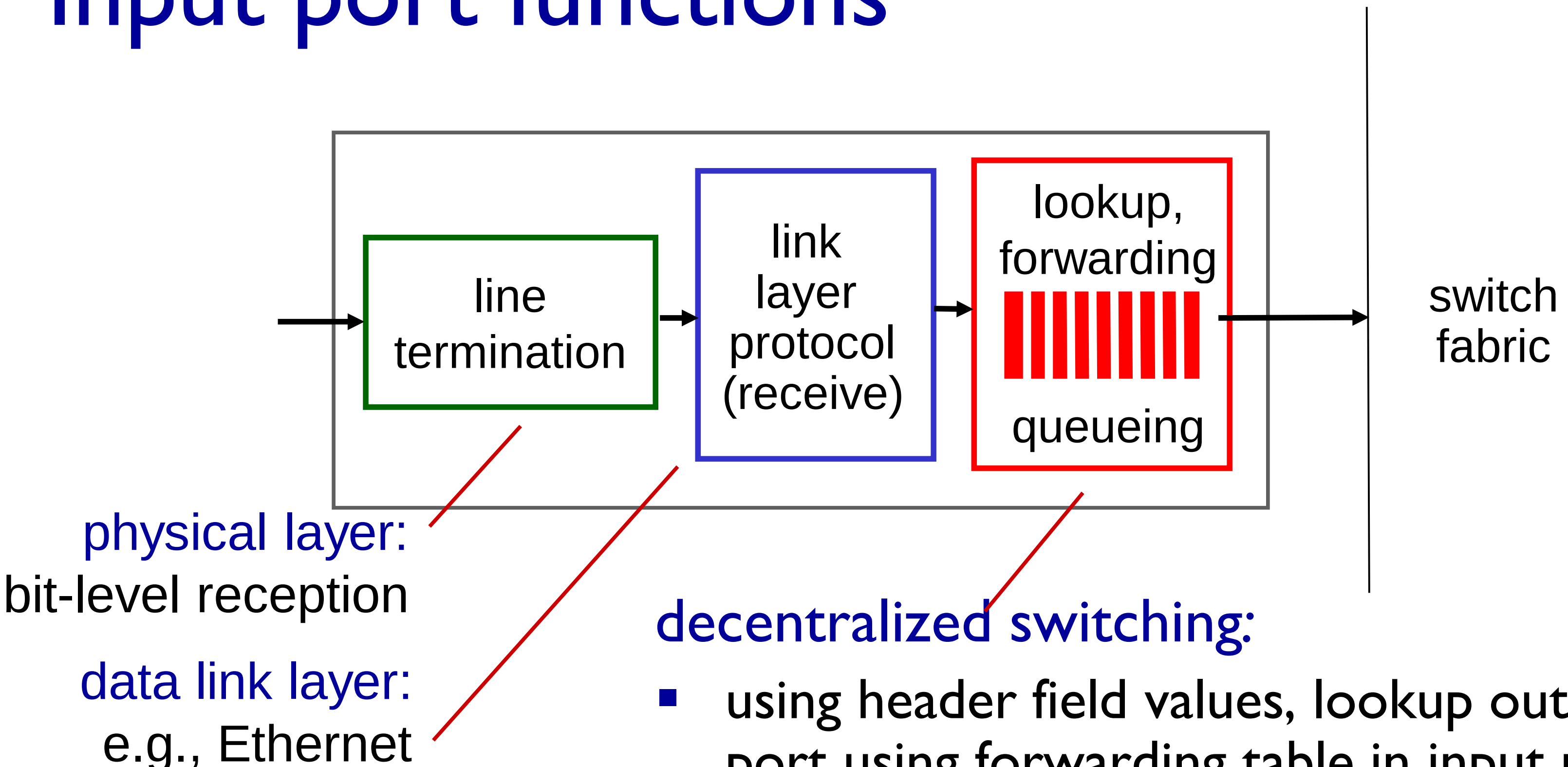
Input port functions



decentralized switching:

- using header field values, lookup output port using forwarding table in input port memory *("match plus action")*
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

Input port functions



decentralized switching:

- using header field values, lookup output port using forwarding table in input port memory *("match plus action")*
- *destination-based forwarding:* forward based only on destination IP address (traditional)
- *generalized forwarding:* forward based on any set of header field values

Destination-based forwarding

forwarding table

Destination Address Range	Link Interface

`11001000 00010111 00010000 00000000`
through
`11001000 00010111 00010111 11111111`
	0

`11001000 00010111 00011000 00000000`
through
`11001000 00010111 00011000 11111111`
	1

`11001000 00010111 00011001 00000000`
through
`11001000 00010111 00011111 11111111`
	2

otherwise	3

Q: but what happens if ranges don't divide up so nicely?

Longest prefix matching

longest prefix matching

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** *********	0
11001000 00010111 00011000 *********	1
11001000 00010111 00011*** *********	2
otherwise	3

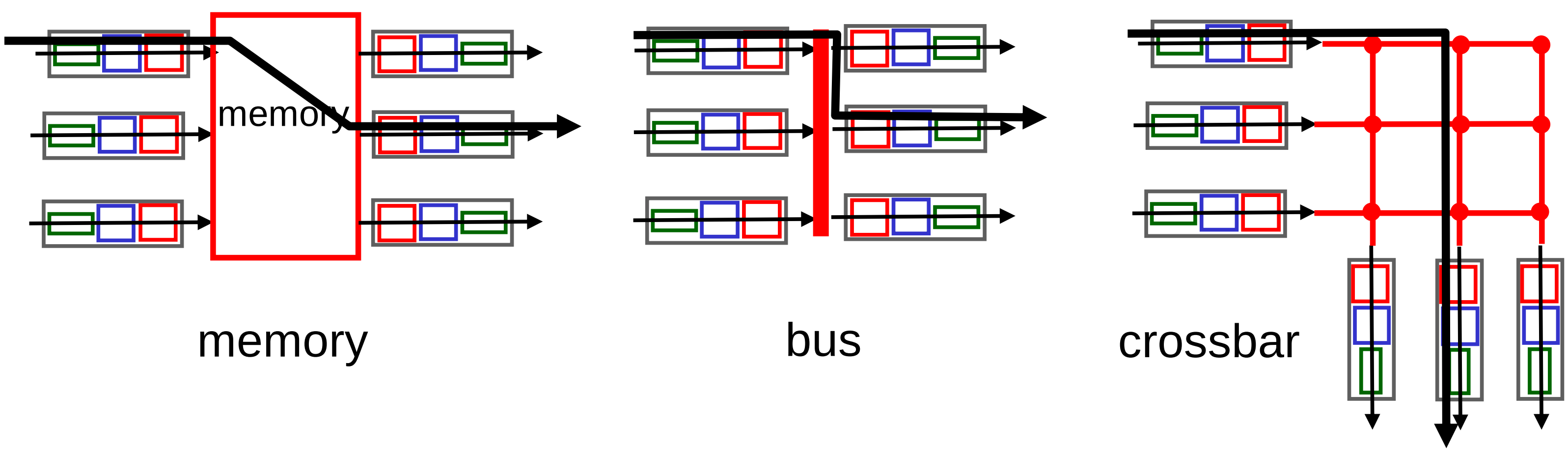
examples:

DA: 11001000 00010111 00010110 10100001 which interface?

DA: 11001000 00010111 00011000 10101010 which interface?

Switching fabrics

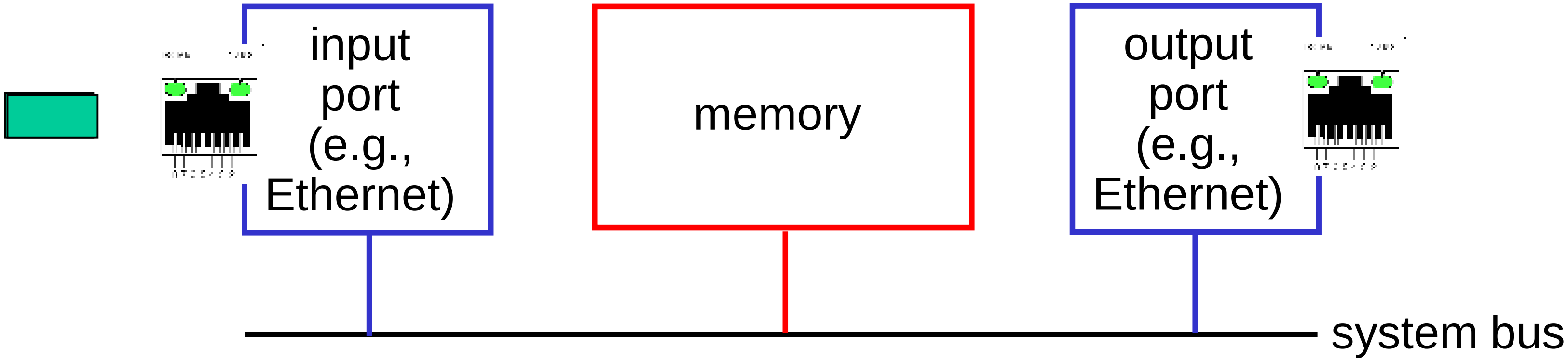
- transfer packet from input buffer to appropriate output buffer
- switching rate: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three types of switching fabrics



Switching via memory

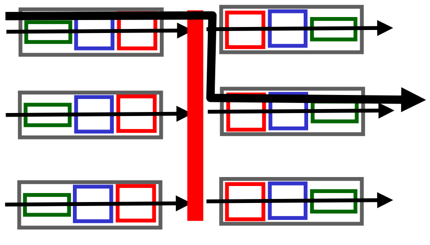
first generation routers:

- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



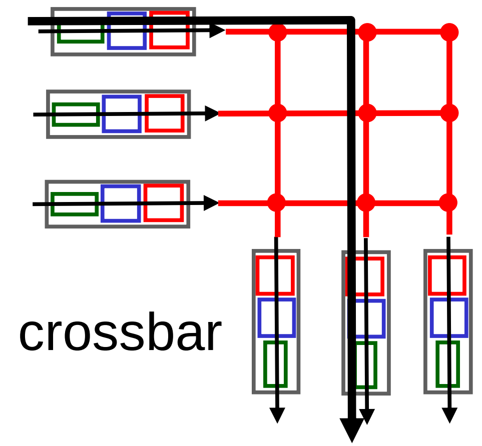
Switching via a bus

- datagram from input port memory to output port memory via a shared bus
- *bus contention:* switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers



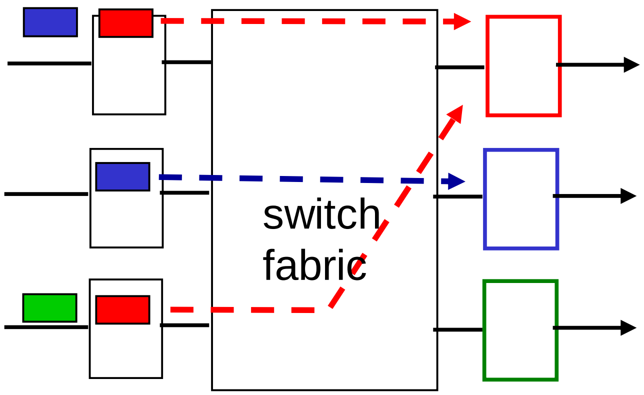
Switching via interconnection network

- overcome bus bandwidth limitations
- banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection network

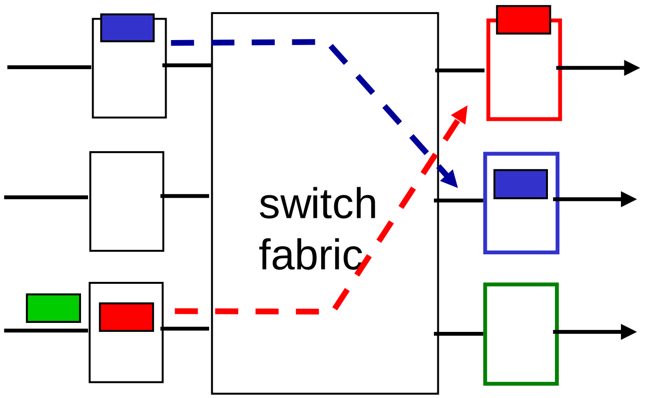


Input port queuing

- fabric slower than input ports combined -> queueing may occur at input queues
  - *queueing delay and loss due to input buffer overflow!*
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward

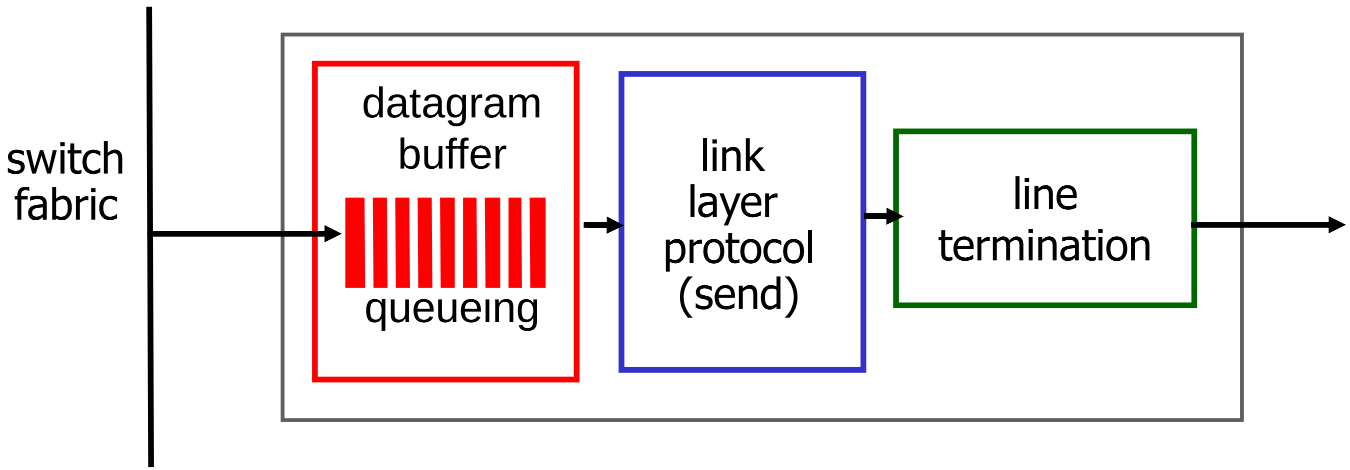


output port contention:
only one red datagram can be transferred.
lower red packet is blocked



one packet time later:
green packet
experiences HOL
blocking

Output ports



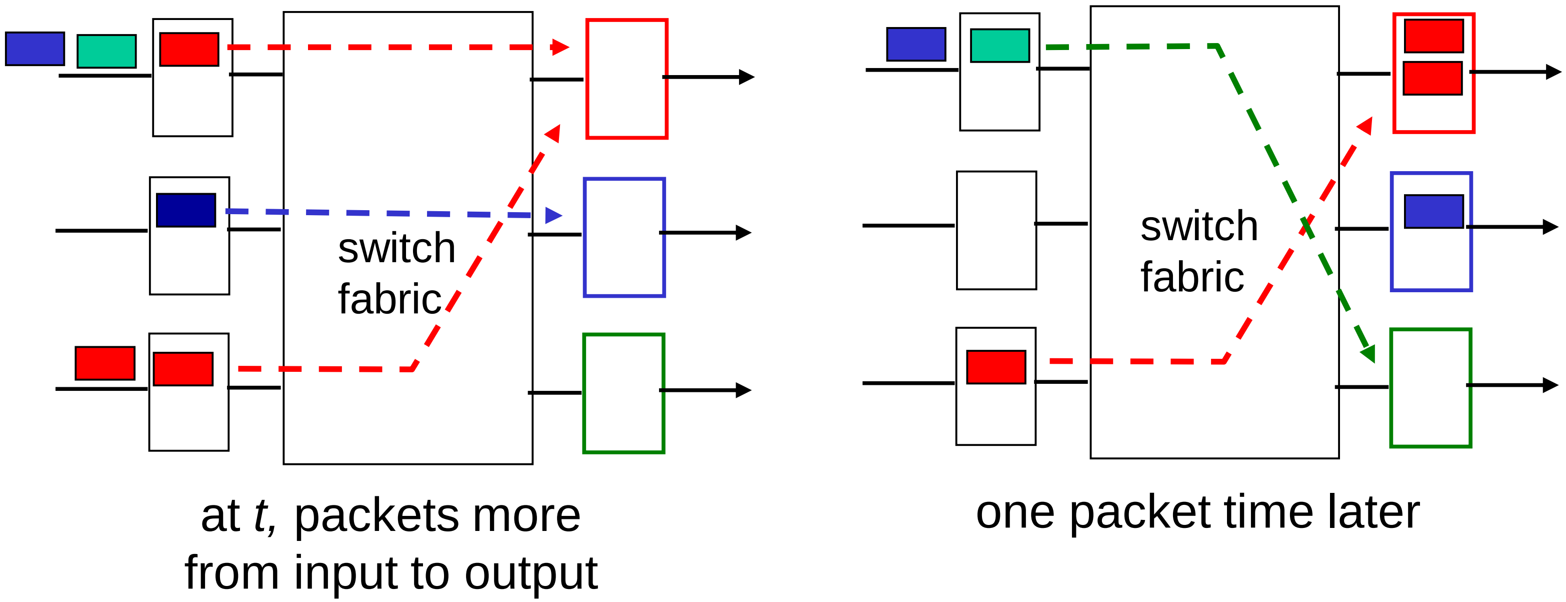
- *buffering* required from fabric faster rate

Datagram (packets) can be lost due to congestion, lack of buffers

- *scheduling* datagrams

Priority scheduling – who gets best performance, network neutrality

Output port queueing



- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

How much buffering?

- RFC 3439 rule of thumb: average buffering equal to "typical" RTT (say 250 msec) times link capacity C
  - e.g., C = 10 Gpbs link: 2.5 Gbit buffer
- recent recommendation: with *N* flows, buffering equal to

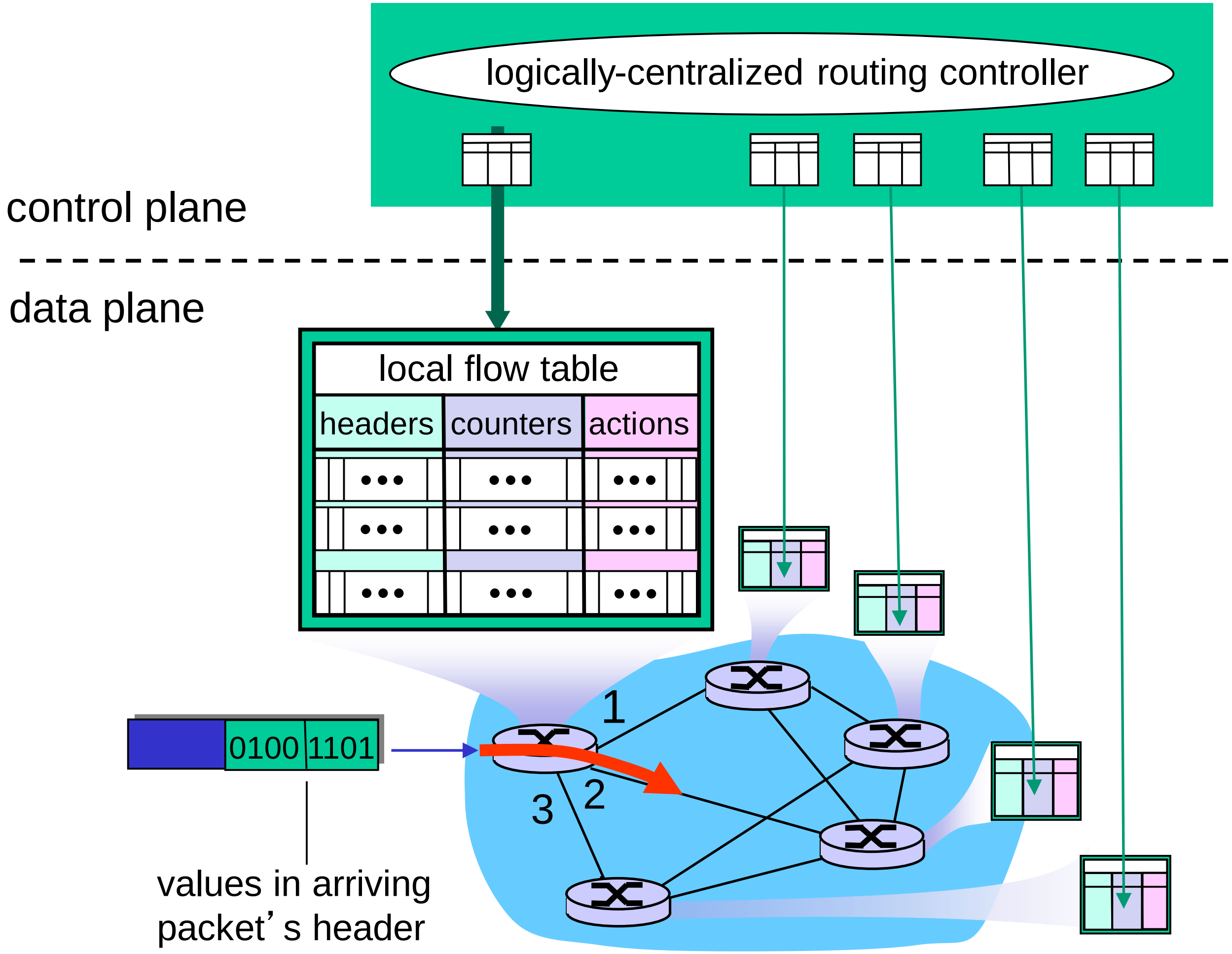
$$\frac{RTT * C}{\sqrt{N}}$$

Outline

* Overview of Network layer
  - data plane
  - control plane
* What's inside a router
* Generalized Forward and SDN
  - Match
  - Action
  - OpenFlow examples of match-plus-action in action

Generalized Forwarding and SDN

Each router contains a *flow table* that is computed and distributed by a *logically centralized* routing controller



OpenFlow

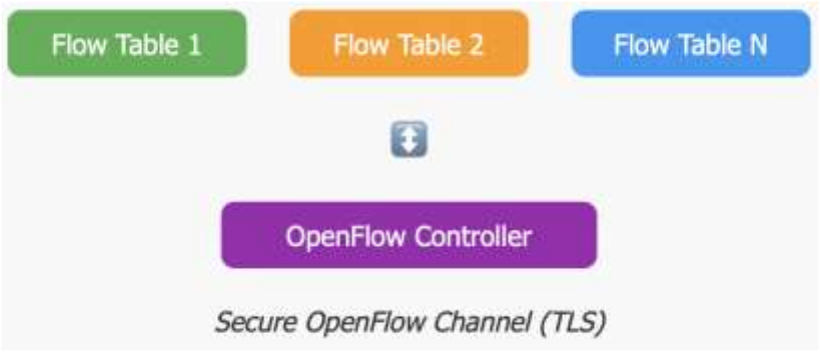
- **What is OpenFlow?**

OpenFlow is the first and most widely adopted southbound API standard for SDN, enabling communication between the controller and network devices for centralized control of packet forwarding.

- **OpenFlow Key Concepts**
  - **Flow Tables:** Packet matching and action rules
  - **Flow Entries:** Individual forwarding rules
  - **OpenFlow Channel:** Secure connection to controller
  - **OpenFlow Messages:** Control protocol messages

OpenFlow

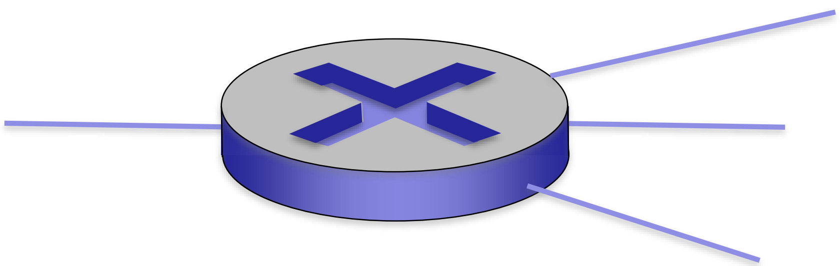
- **OpenFlow Switch Architecture**



- **OpenFlow Versions Evolution**
  - **OpenFlow 1.0:** Basic flow tables and actions
  - **OpenFlow 1.3:** Multiple tables, group tables, meters
  - **OpenFlow 1.4+:** Extensibility, bundles, synchronization

OpenFlow data plane abstraction

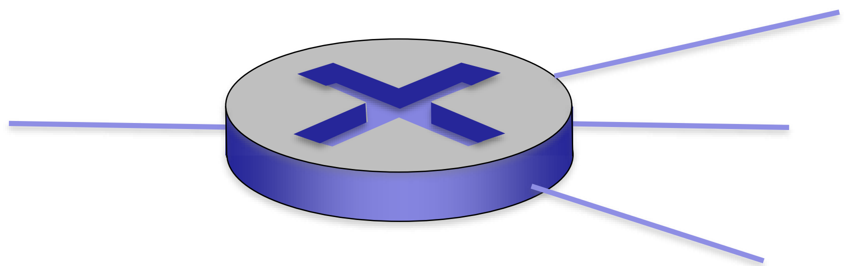
- ***flow***: defined by header fields
- **generalized forwarding**: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters:* #bytes and #packets



Flow table in a router (computed and distributed by controller) define router's match+action rules

OpenFlow data plane abstraction

- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters:* #bytes and #packets



* : wildcard

1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

OpenFlow Actions and Instructions

- **Required Actions**
  - **Output:** Send packet to specified port
  - **Drop:** Discard packet (no action specified)
  - **Controller:** Send packet to controller

- **Optional Actions**

Header Modification Actions:

- Set VLAN ID/Priority
- Strip VLAN header
- Modify MAC addresses
- Modify IP addresses
- Modify TCP/UDP ports
- Modify MPLS labels
- Push/Pop MPLS/VLAN headers

Quality of Service Actions:

- Set queue for rate limiting
- Set DSCP/ToS bits
- Apply traffic shaping

Advanced Actions:

- Group actions (OpenFlow 1.1+)
- Meter actions (OpenFlow 1.3+)
- Copy TTL inward/outward
- Experimenter actions

OpenFlow Messages

- **Message Categories**
  - **Controller-to-Switch:** Control and configuration messages
  - **Asynchronous:** Event notifications from switch
  - **Symmetric:** Bidirectional messages
- **Key OpenFlow Messages**

  Controller-to-Switch Messages:
  - Features Request/Reply: Capabilities exchange
  - Configuration: Switch configuration
  - Flow Mod: Flow table modifications
  - Group Mod: Group table modifications
  - Port Mod: Port configuration
  - Table Mod: Table configuration
  - Statistics Request/Reply: Information gathering

OpenFlow Messages

- **Key OpenFlow Messages**

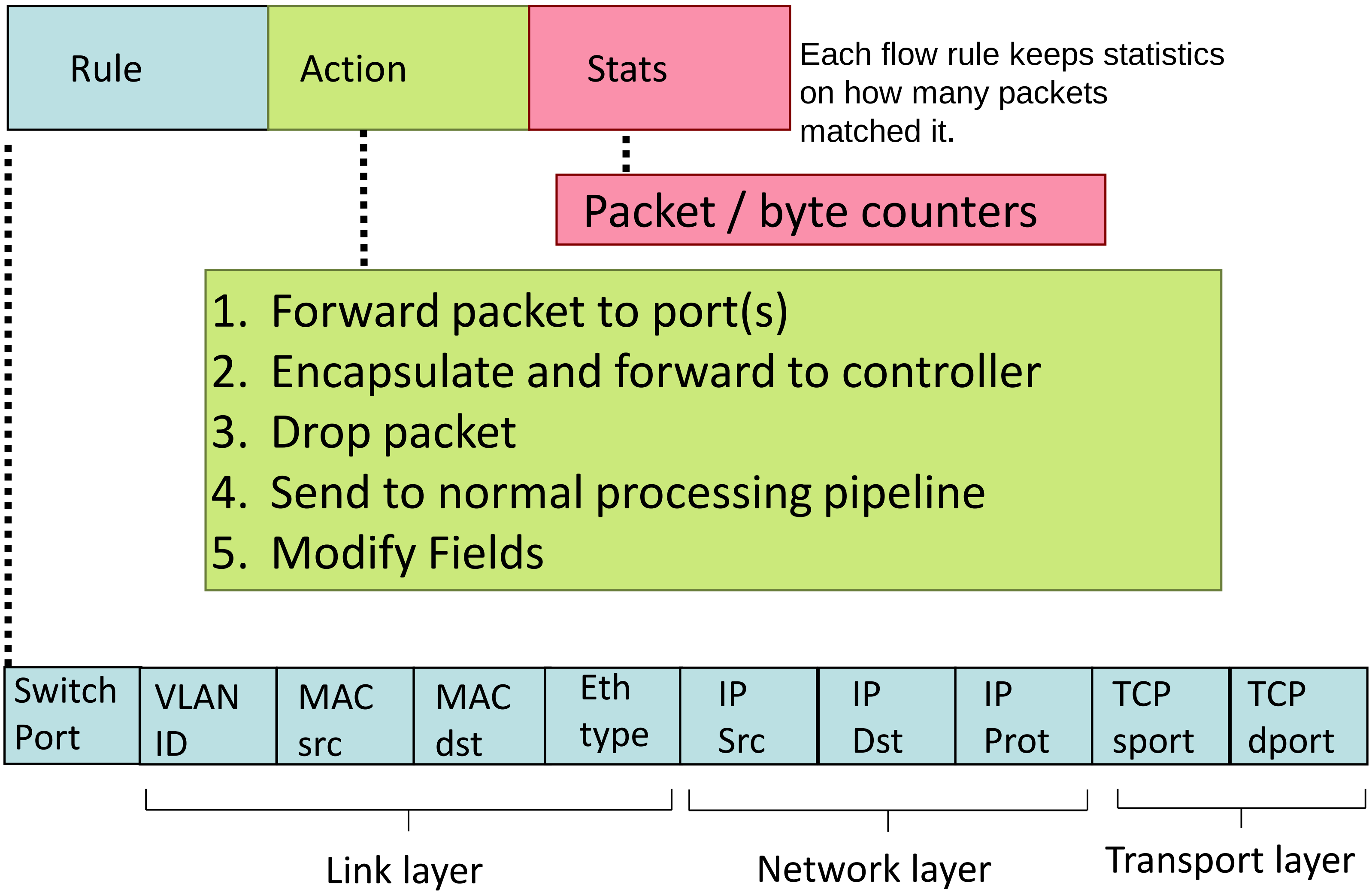
Asynchronous Messages:

- Packet-In: Unmatched packets sent to controller
- Flow Removed: Flow entry expiration notification
- Port Status: Port state change notification
- Error: Error condition reporting

Symmetric Messages:

- Hello: Connection establishment
- Echo Request/Reply: Keepalive mechanism
- Vendor/Experimenter: Vendor-specific extensions
- Barrier Request/Reply: Message ordering

OpenFlow: Flow Table Entries



Flow Tables and Entries

- **Example Flow Table**

Priority	Match	Action	Counters
100	in_port = 1, eth_dst = 00:01:02:03:04:05	output:2	1,024 packets
90	in_port = 1, eth_type = 0x0800, ip_dst = 10.0.0.1	output:3	2,048 packets
50	in_port=1	controller	512 packets
0	*	drop	128 packets

Examples

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	51.6.0.8	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	*	*	*	*	22	drop

do not forward (block) all datagrams destined to TCP port 22

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	128.119.1.1	*	*	*	*	drop

do not forward (block) all datagrams sent by host 128.119.1.1

Examples

Destination-based layer 2 (switch) forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	port3

layer 2 frames from MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

OpenFlow abstraction

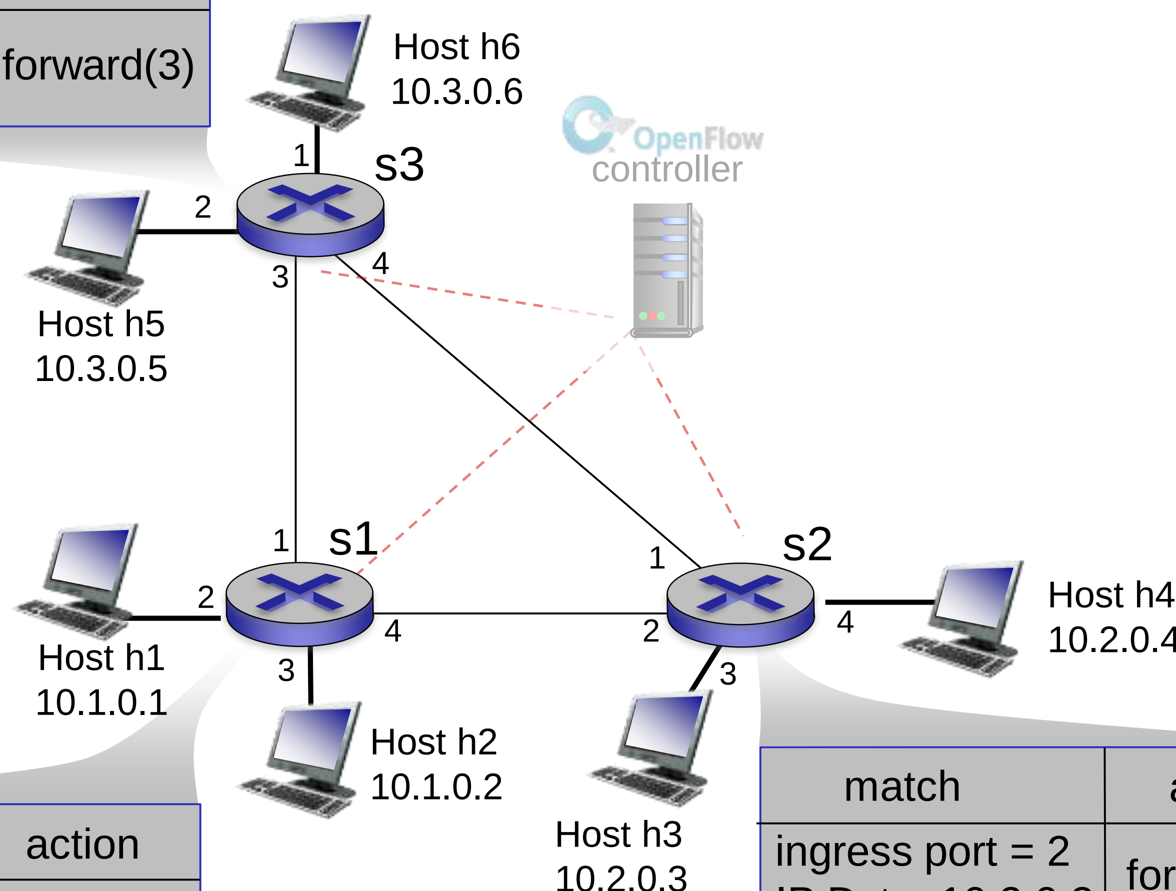
- *match+action:* unifies different kinds of devices

- Router
  - *match:* longest destination IP prefix
  - *action:* forward out a link
- Switch
  - *match:* destination MAC address
  - *action:* forward or flood
- Firewall
  - *match*: IP addresses and TCP/UDP port numbers
  - *action:* permit or deny
- NAT
  - *match:* IP address and port
  - *action:* rewrite address and port

OpenFlow example

Example: datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

match	action
IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(3)



match	action
ingress port = 1 IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(4)

match	action
ingress port = 2 IP Dst = 10.2.0.3	forward(3)
ingress port = 2 IP Dst = 10.2.0.4	forward(4)