

# IP Data Plane

CS/ECE 438: Spring 2014

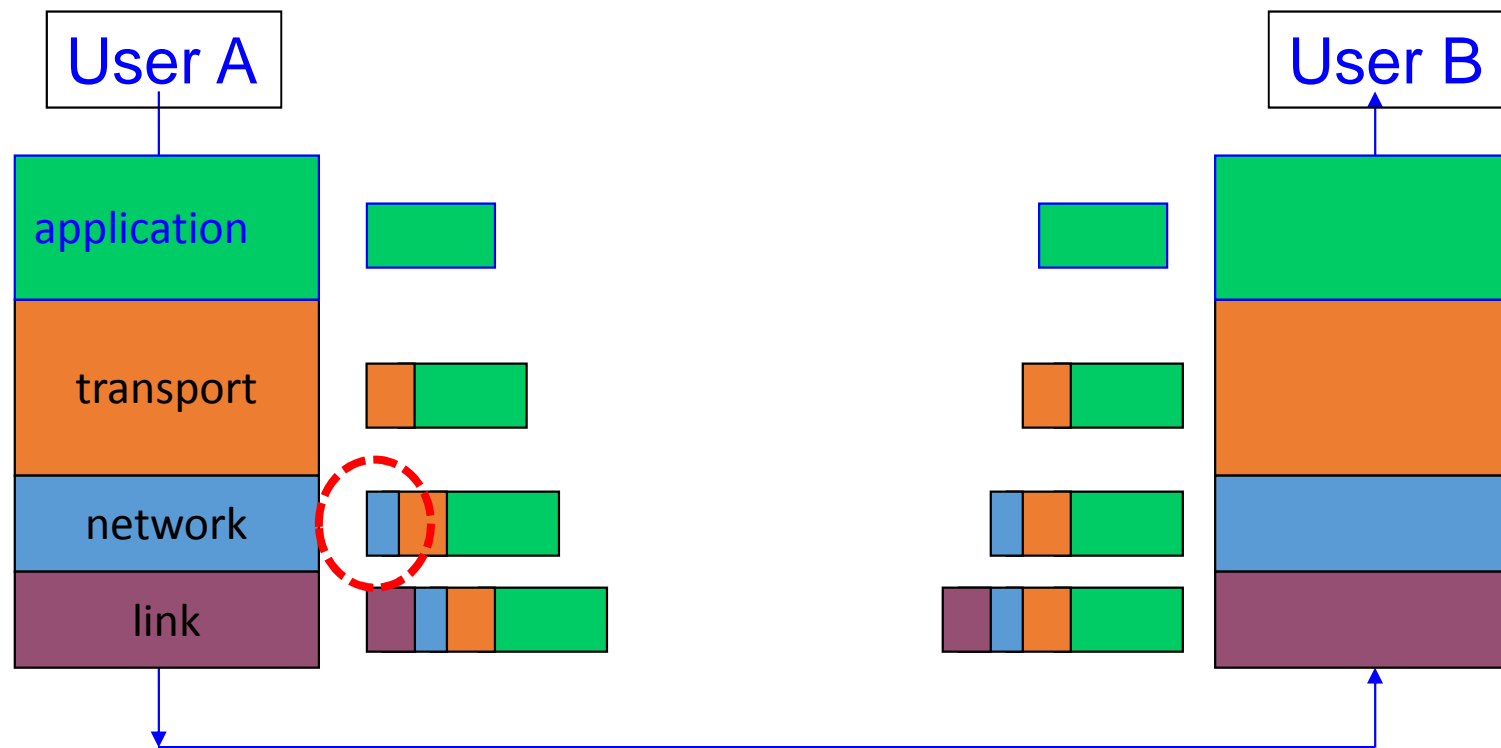
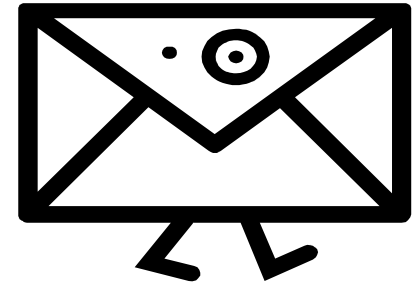
Instructor: Matthew Caesar

<http://courses.engr.illinois.edu/cs438/>

# The IP layer

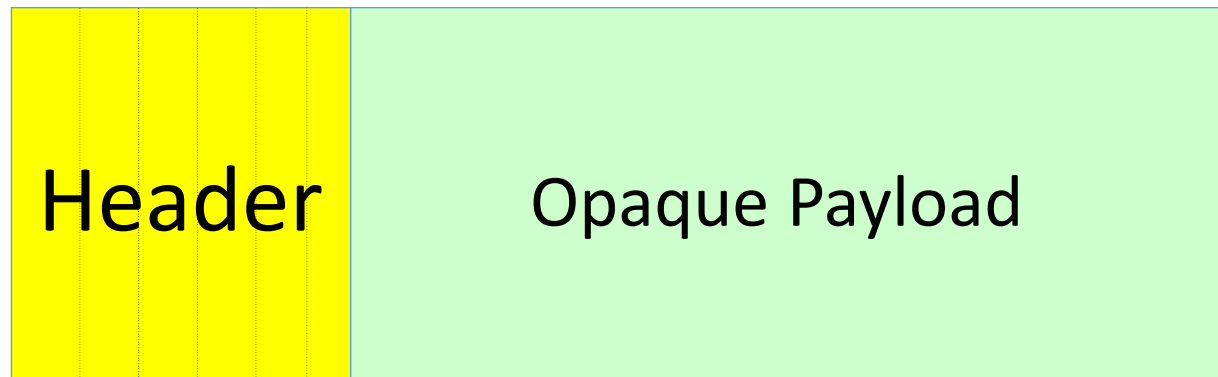
- So far, we've focused mostly on routing protocols
  - how routers discover and select end-to-end paths
  - part of a network's **control** plane
- Today: the **data** plane
  - what data packets look like at the IP layer (the IP header)
  - how routers forward these IP packets

# Recall from Lecture#3: Layer Encapsulation



# What is Designing IP?

- Syntax: format of packet
  - Nontrivial part: packet “header”
  - Rest is opaque payload (*why opaque?*)



- Semantics: meaning of header fields
  - Required processing

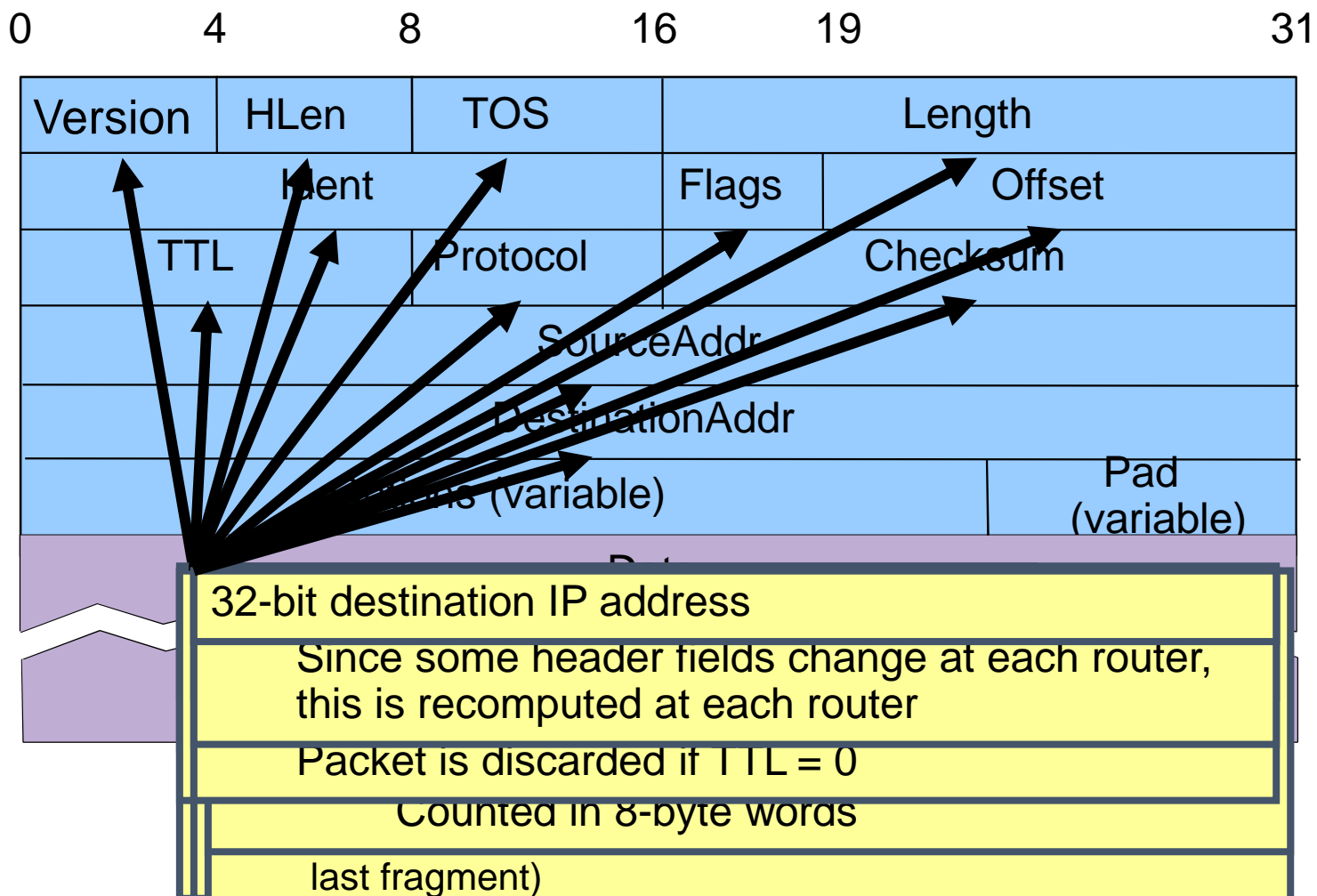
# Packet Headers

- Think of packet header as interface
  - Only way of passing information from packet to switch
- Designing interfaces
  - What task are you trying to perform?
  - What information do you need to accomplish it?
- Header reflects information needed for basic tasks

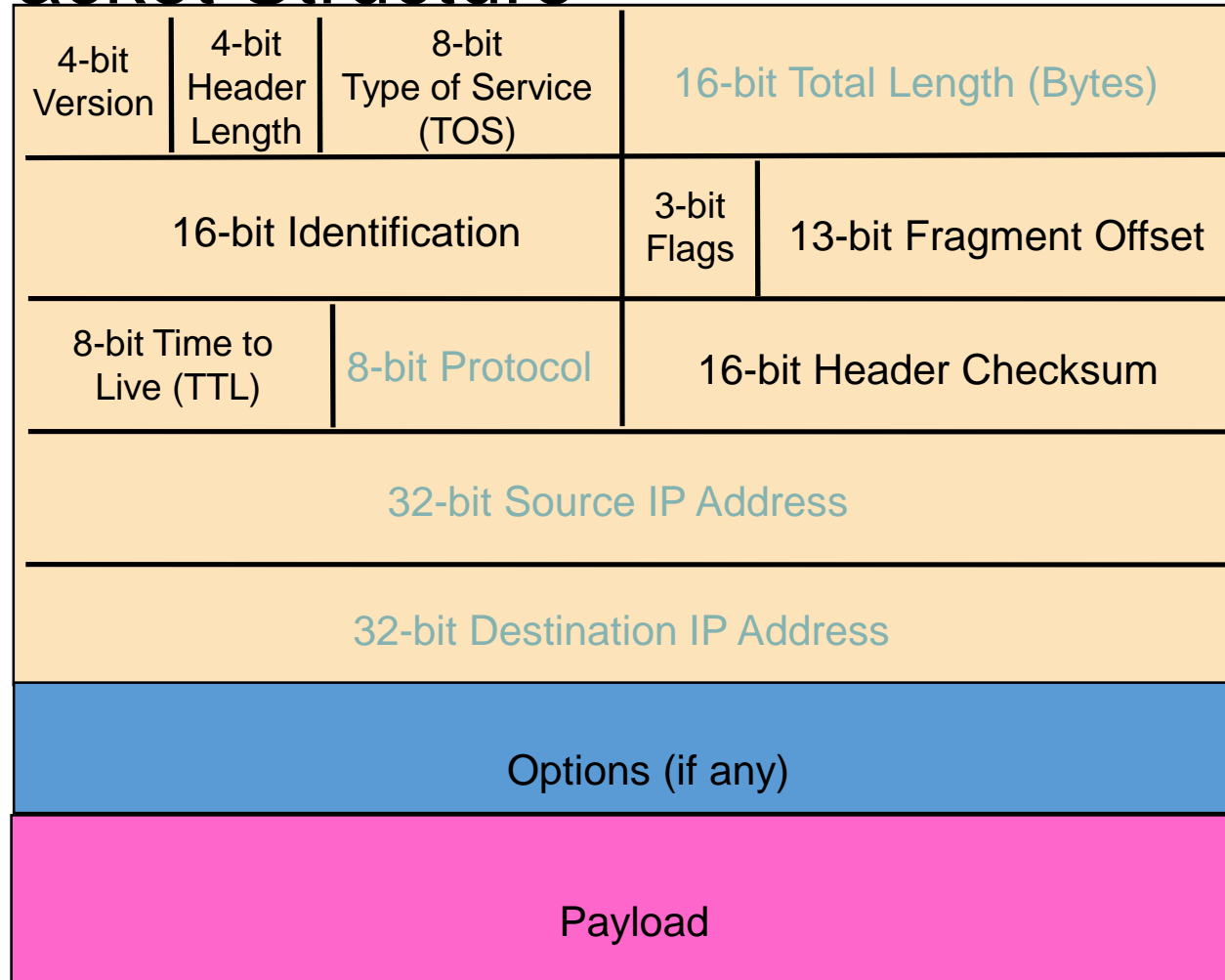
# What Tasks Do We Need to Do?

- Read packet correctly
- Get packet to the destination; responses back to the source
- Carry data
- Tell host what to do with packet once arrived
- Specify any special network handling of the packet
- Deal with problems that arise along the path

# IP Packet Format



# IP Packet Structure





# 20 Bytes of Standard Header, then Options

4-bit Version	4-bit Header Length	8-bit Type of Service (TOS)	16-bit Total Length (Bytes)	
16-bit Identification			3-bit Flags	13-bit Fragment Offset
8-bit Time to Live (TTL)	8-bit Protocol		16-bit Header Checksum	
32-bit Source IP Address				
32-bit Destination IP Address				
Options (if any)				
Payload				

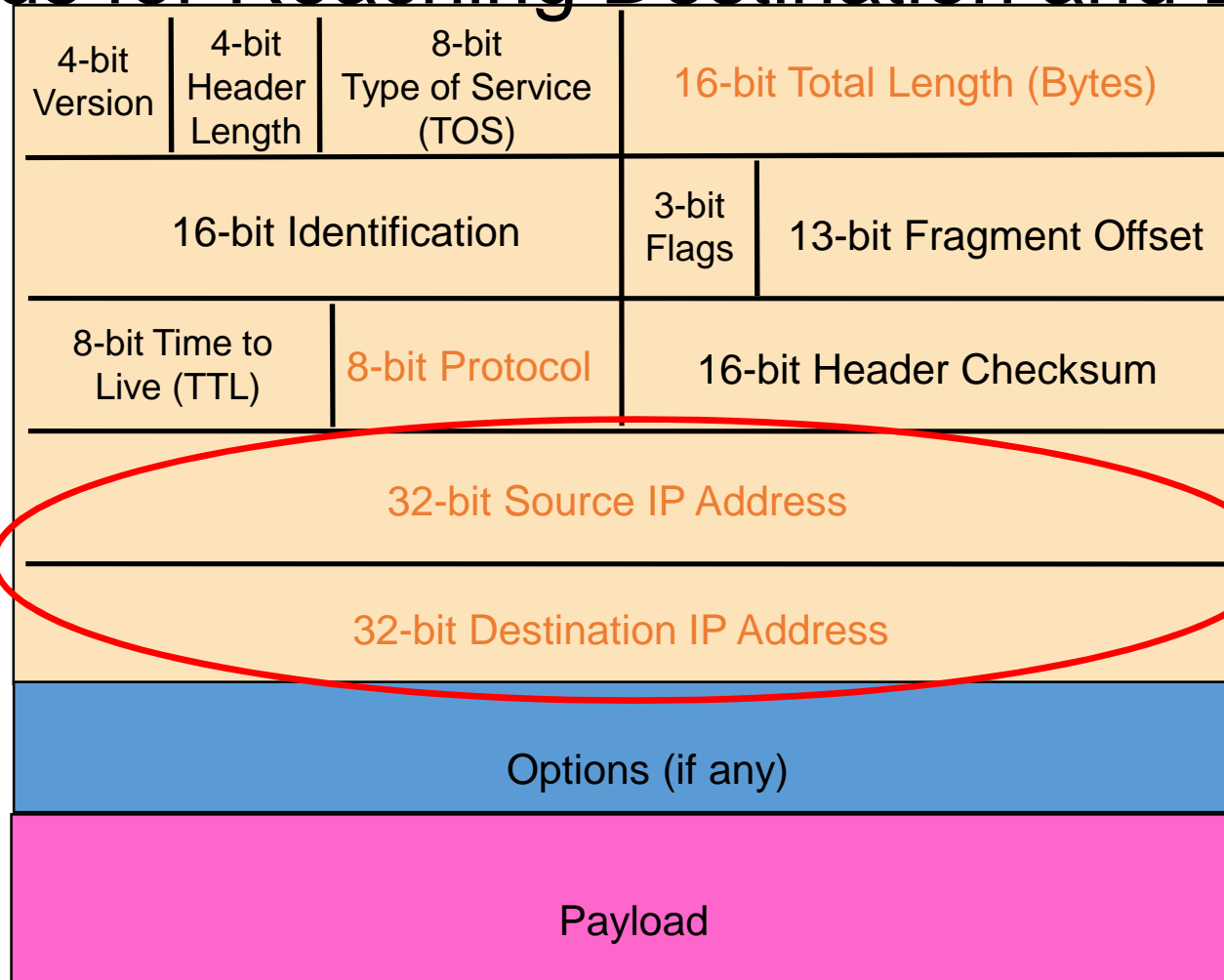
# Fields for Reading Packet Correctly

4-bit Version	4-bit Header Length	8-bit Type of Service (TOS)	16-bit Total Length (Bytes)	
16-bit Identification		3-bit Flags	13-bit Fragment Offset	
8-bit Time to Live (TTL)	8-bit Protocol		16-bit Header Checksum	
32-bit Source IP Address				
32-bit Destination IP Address				
Options (if any)				
Payload				

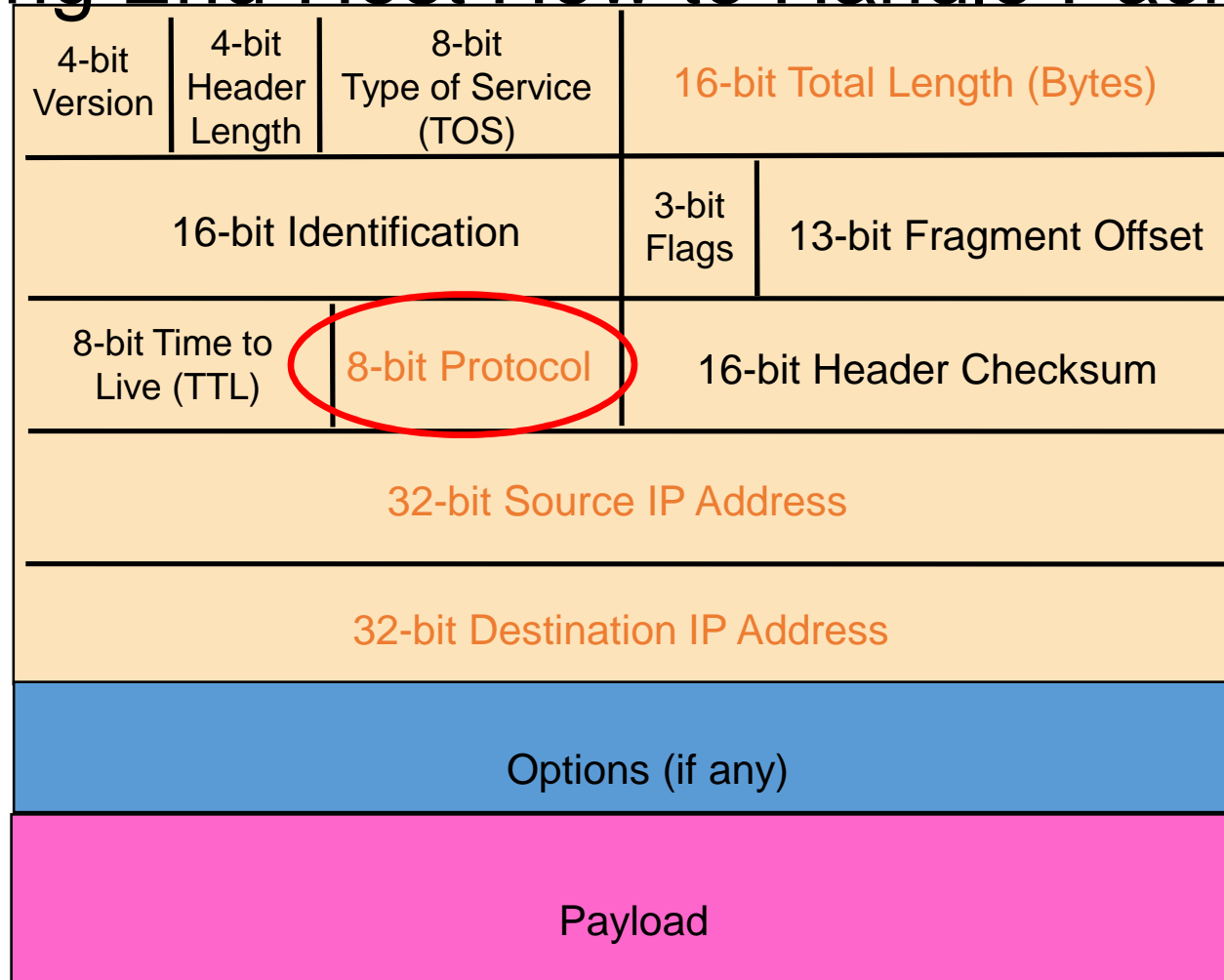
# Reading Packet Correctly

- Version number (4 bits)
  - Indicates the version of the IP protocol
  - Necessary to know what other fields to expect
  - Typically “4” (for IPv4), and sometimes “6” (for IPv6)
- Header length (4 bits)
  - Number of 32-bit words in the header
  - Typically “5” (for a 20-byte IPv4 header)
  - Can be more when IP **options** are used
- Total length (16 bits)
  - Number of bytes in the packet
  - Maximum size is 65,535 bytes ( $2^{16} - 1$ )
  - ... though underlying links may impose smaller limits

# Fields for Reaching Destination and Back

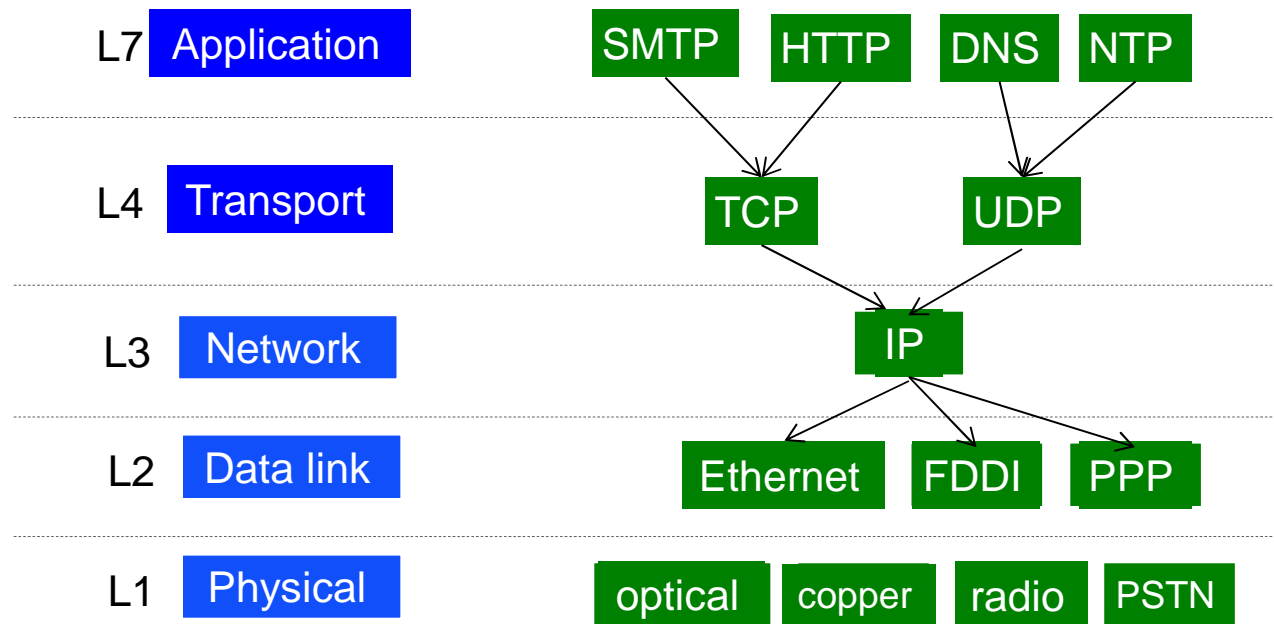


# Telling End-Host How to Handle Packet



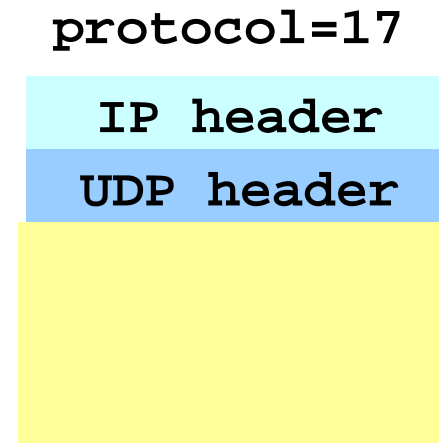
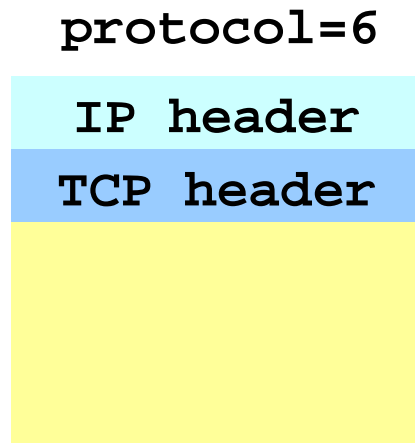
# Telling End-Host How to Handle Packet

- Protocol (8 bits)
  - Identifies the higher-level protocol
  - Important for demultiplexing at receiving host



# Telling End-Host How to Handle Packet

- Protocol (8 bits)
  - Identifies the higher-level protocol
  - Important for demultiplexing at receiving host
- Most common examples
  - E.g., “6” for the Transmission Control Protocol (TCP)
  - E.g., “17” for the User Datagram Protocol (UDP)

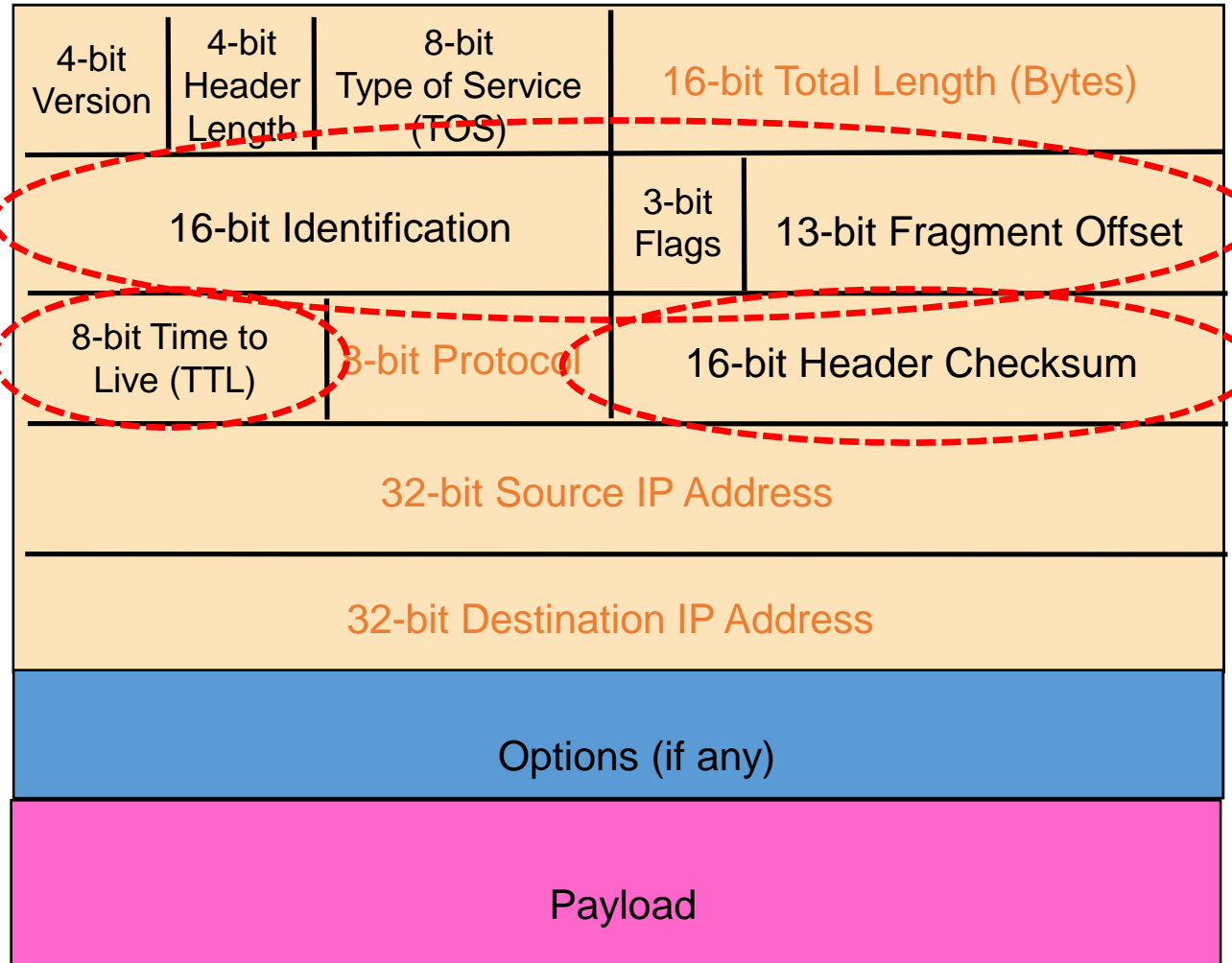


# Potential Problems

- Header Corrupted: **Checksum**
- Loop: **TTL**
- Packet too large: **Fragmentation**



# Checksum, TTL and Fragmentation Fields

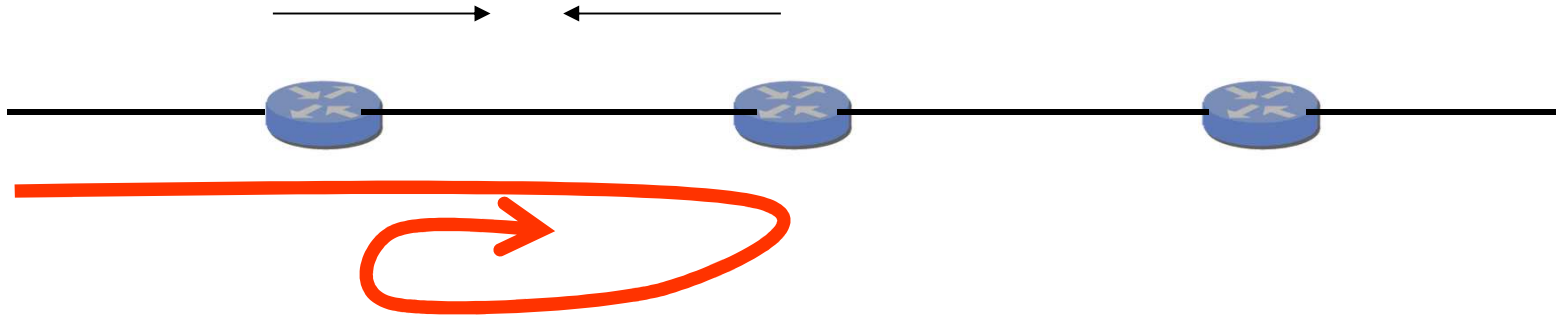


# Header Corruption (Checksum)

- Checksum (16 bits)
  - Particular form of checksum over packet header
- If not correct, router discards packets
  - So it doesn't act on bogus information
- Checksum recalculated at every router
  - **Why?**
  - **Why include TTL?**
  - **Why only header?**

# Preventing Loops (TTL)

- Forwarding loops cause packets to cycle for a looong time
  - As these accumulate, eventually consume **all** capacity



- Time-to-Live (TTL) Field (8 bits)
  - Decrementated at each hop, packet discarded if reaches 0
  - ...and “time exceeded” message is sent to the source

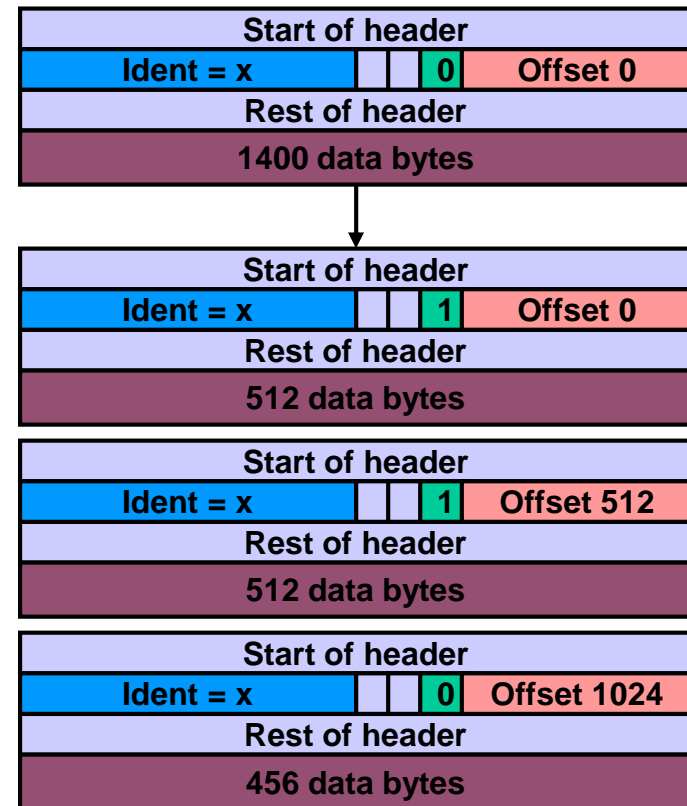
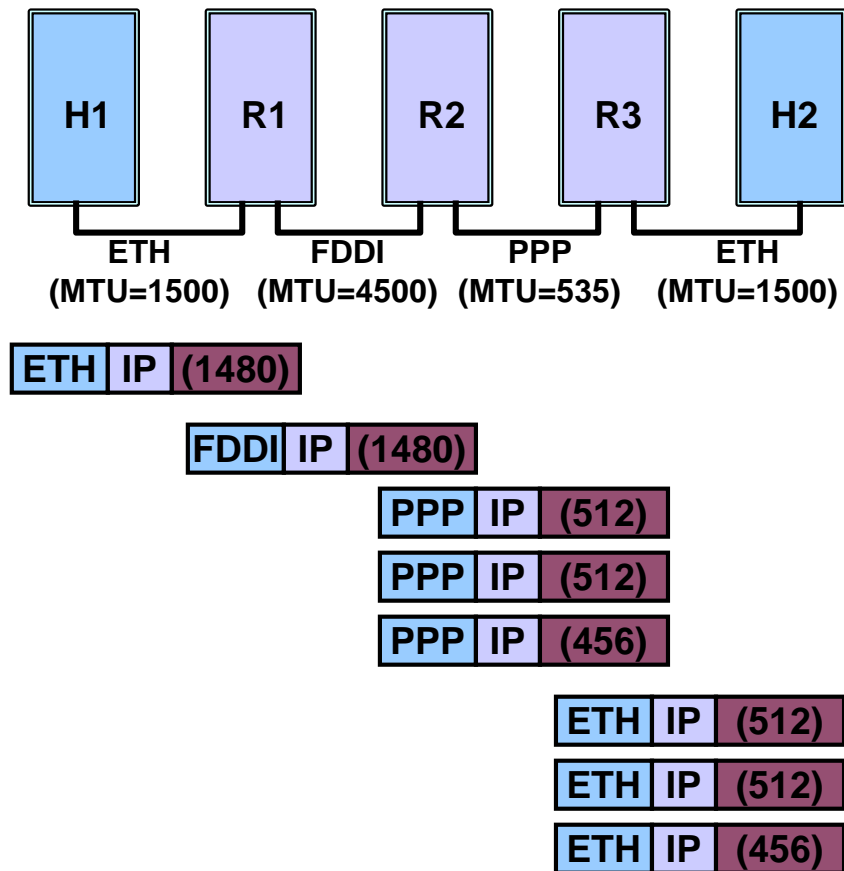
# Fragmentation

- Fragmentation: when forwarding a packet, an Internet router can **split** it into multiple pieces (“fragments”) if the packet is too big for next hop link
  - too big → exceeds the link’s “Max Transmission Unit” (MTU)
- Must **reassemble** to recover original packet
  - Need fragmentation information (32 bits)
  - Packet **identifier**, **flags**, and fragment **offset**

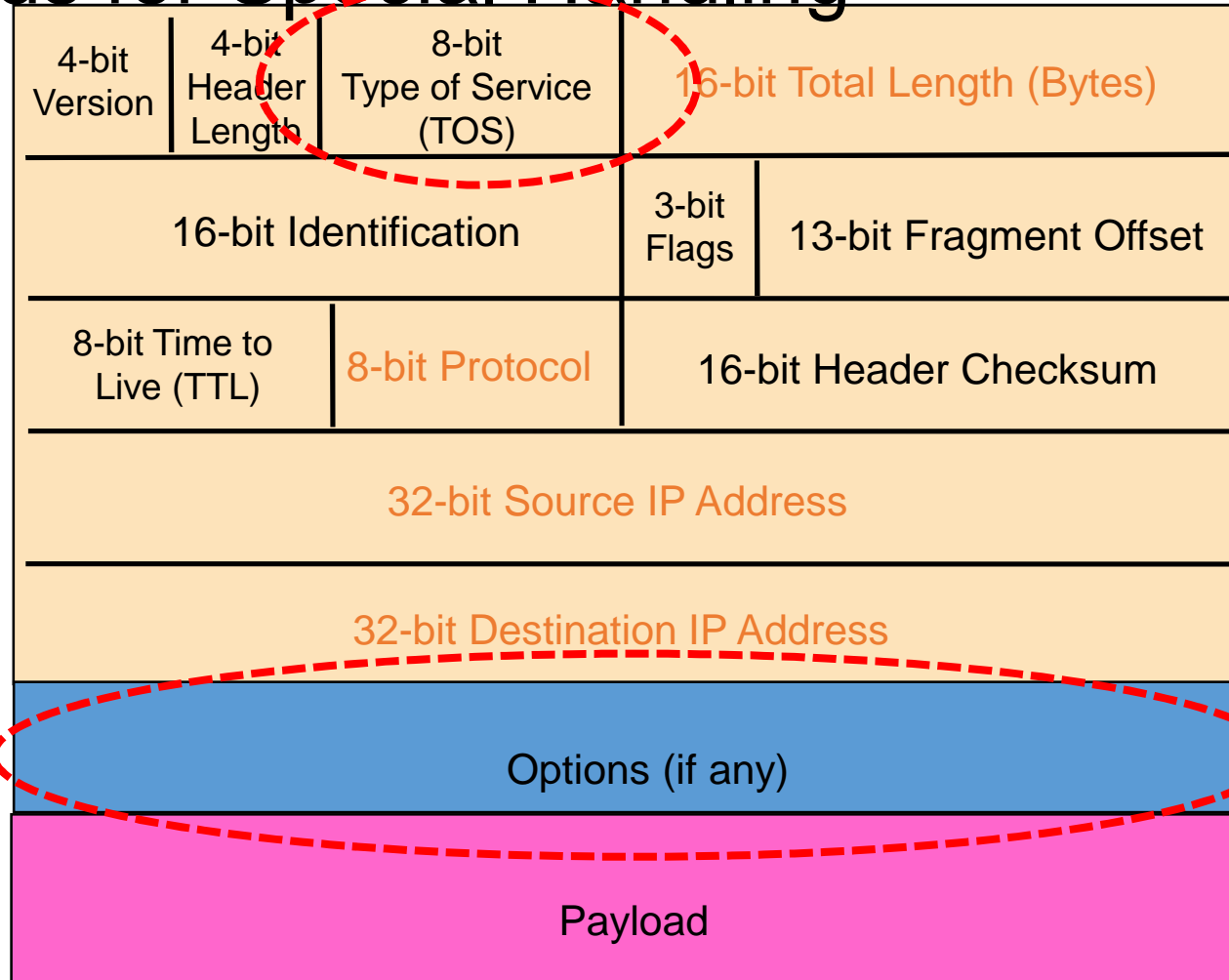
# Path MTU discovery

- Set “don’t fragment” bit in IP header, size is MTU of first hop
- Interface with too-small MTU responds back with “ICMP” message
  - Unfortunately, many networks drop ICMP traffic
- Reduce packet size, repeat until discover smallest MTU on path
- Binary search
  - Better yet: note there are small number of MTUs in the Internet

# IP Fragmentation and Reassembly



# Fields for Special Handling



# Special Handling

- “Type of Service”, or “Differentiated Services Code Point (DSCP)” (8 bits)
  - Allow packets to be treated differently based on needs
  - E.g., low delay for audio, high bandwidth for bulk transfer
  - Has been redefined several times, will cover later in class
- Options (not often used)
  - details in Section



# Examples of Options

- Record Route
- Strict Source Route
- Loose Source Route
- Timestamp
- Traceroute
- Router Alert
- .....

Let's take a quick look at the  
IPv6 header...

# IPv6

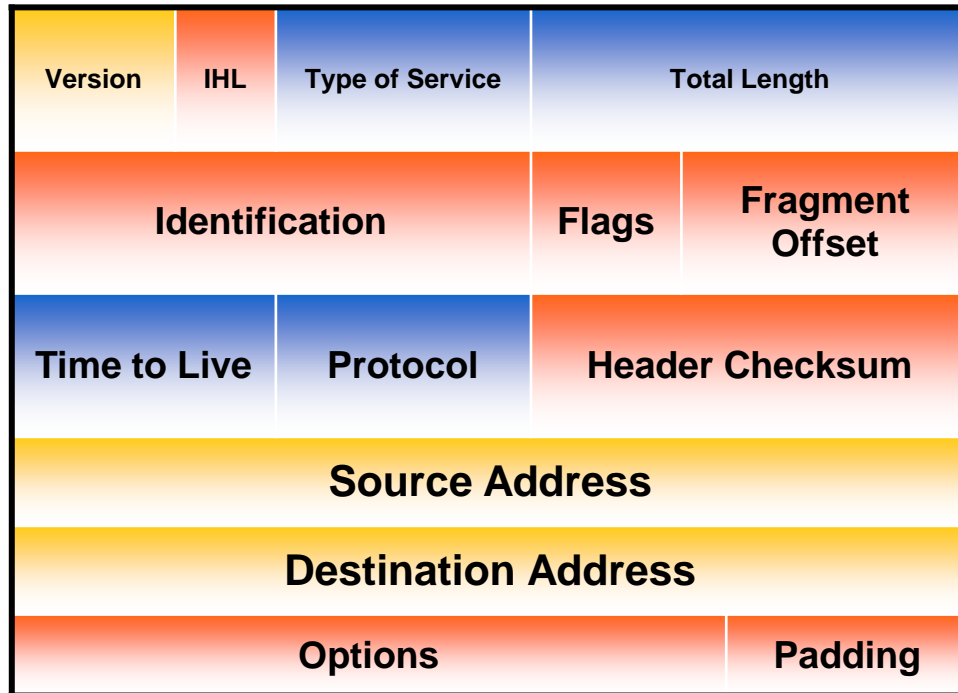
- Motivated (prematurely) by address exhaustion
  - Addresses *four* times as big
- Steve Deering focused on simplifying IP
  - Got rid of all fields that were not absolutely necessary
  - “Spring Cleaning” for IP

# Summary of Changes

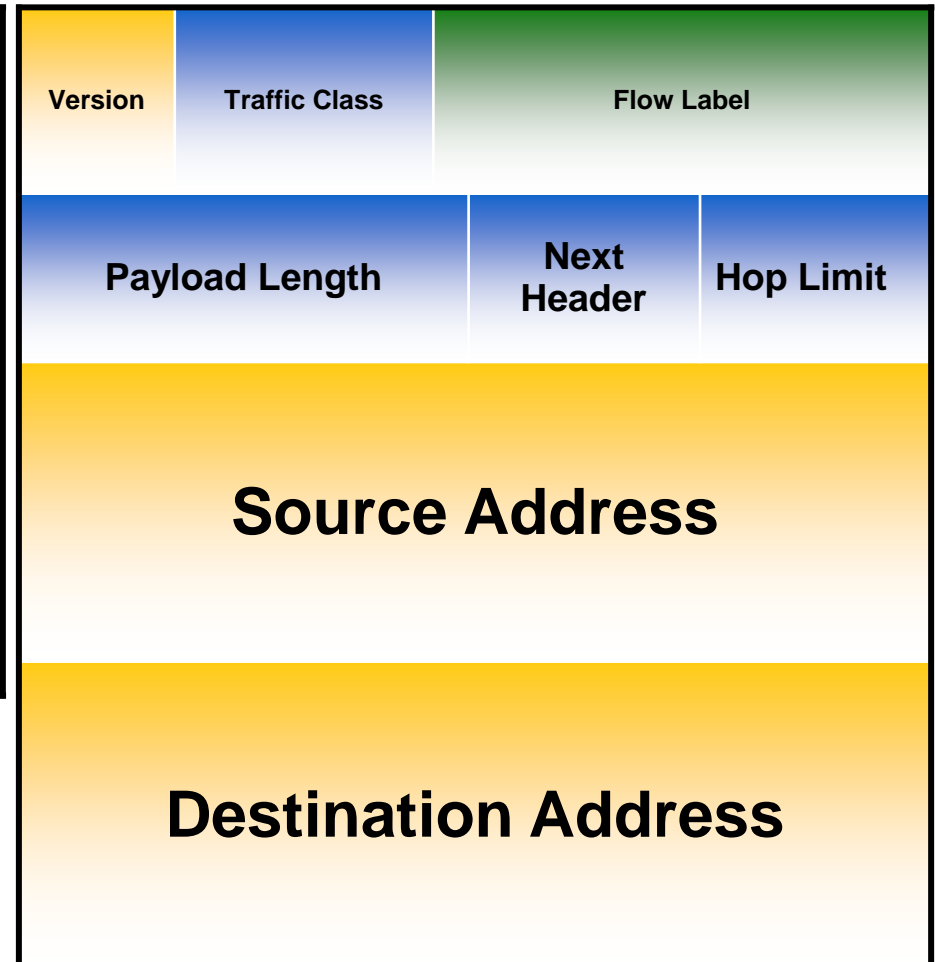
- Eliminated fragmentation (*why?*)
- Eliminated checksum (*why?*)
- New options mechanism (next header) (*why?*)
- Eliminated header length (*why?*)
- Expanded addresses (*why?*)
- Added Flow Label (*why?*)





# IPv4 and IPv6 Header Comparison

IPv4



IPv6



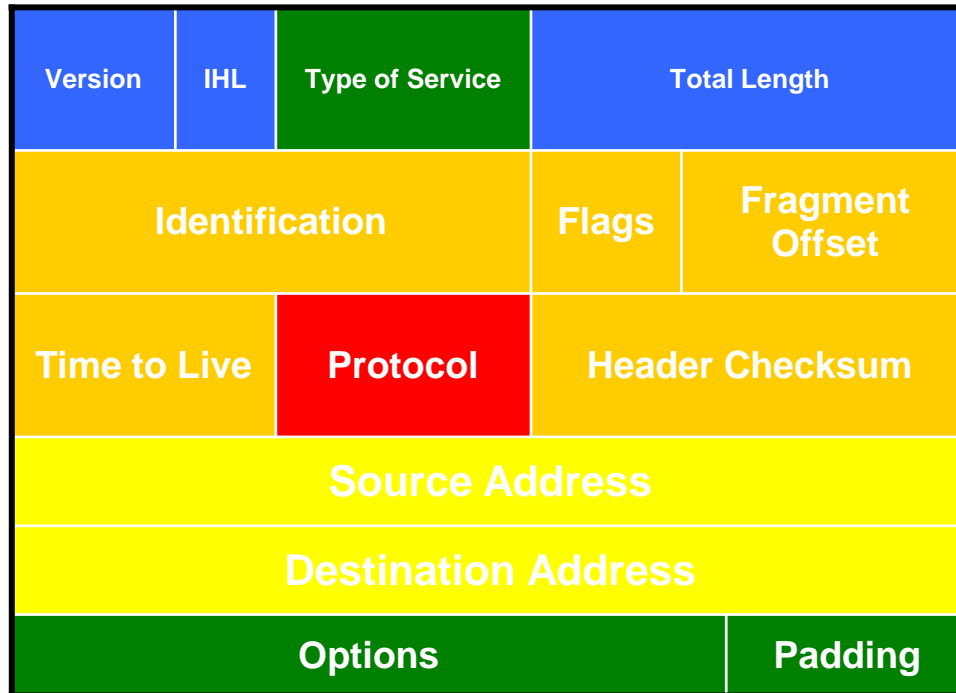
-  Field name kept from IPv4 to IPv6
-  Fields not kept in IPv6
-  Name & position changed in IPv6
-  New field in IPv6

# Philosophy of Changes

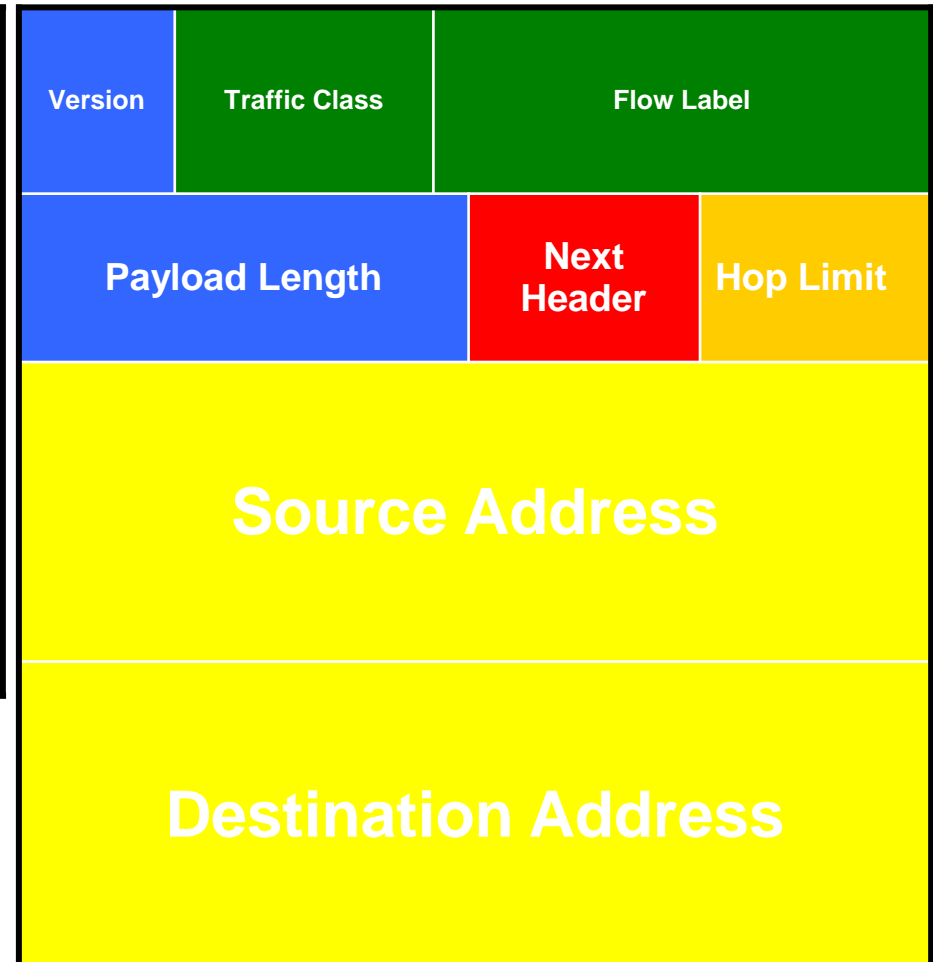
- Don't deal with problems: leave to ends
  - Eliminated fragmentation
  - Eliminated checksum
  - *Why retain TTL?*
- Simplify handling:
  - New options mechanism (uses next header approach)
  - Eliminated header length
    - *Why couldn't IPv4 do this?*
- Provide general flow label for packet
  - Not tied to semantics
  - Provides great flexibility





# Comparison of Design Philosophy

IPv4



IPv6



-  To Destination and Back (expanded)
-  Deal with Problems (greatly reduced)
-  Read Correctly (reduced)
-  Special Handling (similar)

# IP Routers

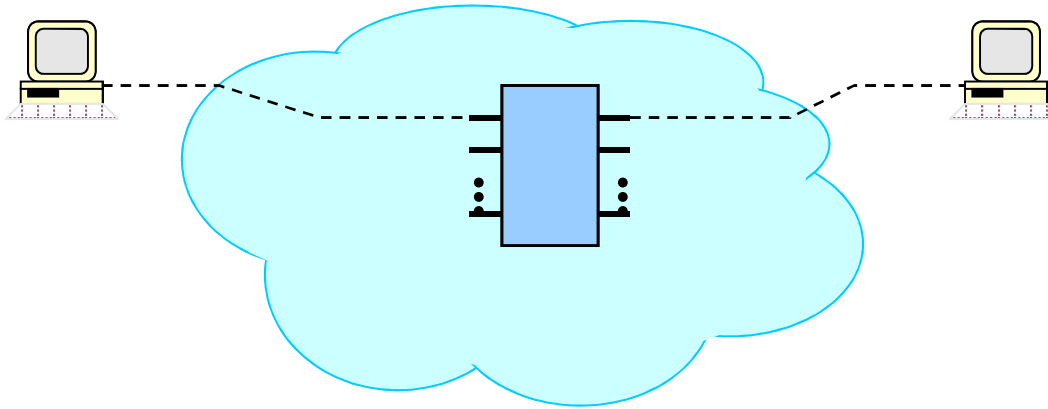
- Core building block of the Internet infrastructure
- \$120B+ industry
- Vendors: Cisco, Huawei, Juniper, Alcatel-Lucent (account for >90%)



# This lecture

- Network devices
  - Their internals and how they work
- Network connections
  - How to plug devices together

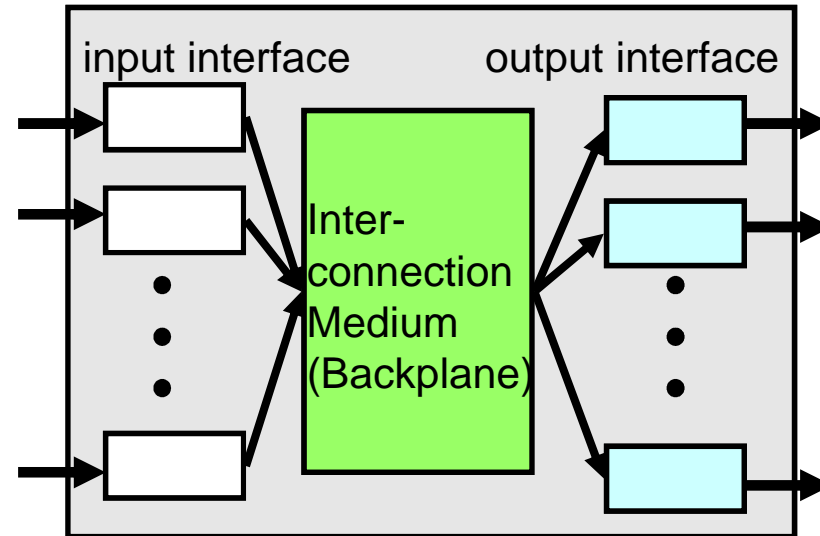
# IP Router



- A router consists
  - A set of input interfaces at which packets arrive
  - A set of output interfaces from which packets depart
- Router implements two main functions
  - Forward packet to corresponding output interface
  - Manage congestion

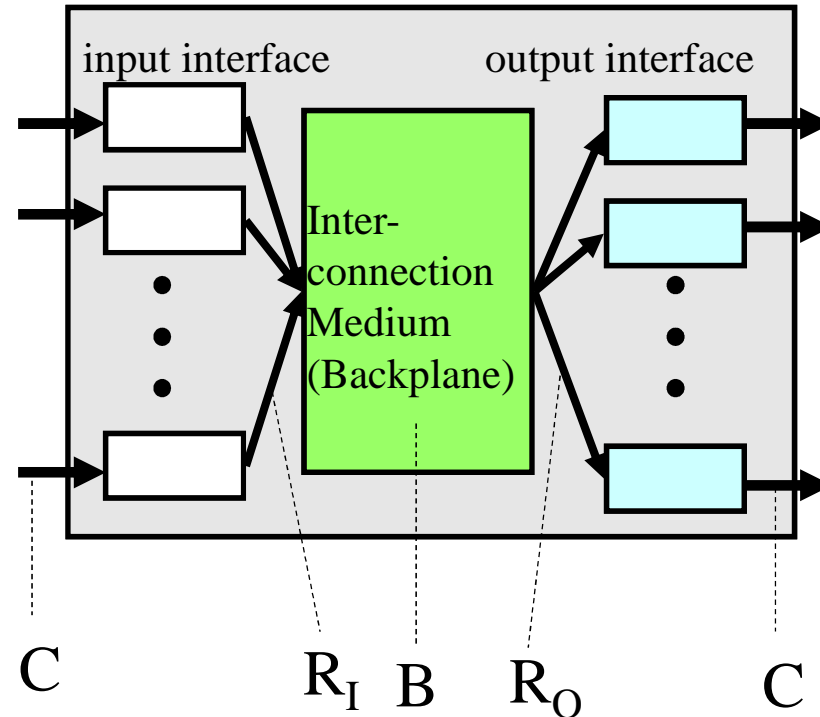
# Generic Router Architecture

- Input and output interfaces are connected through a backplane
- A backplane can be implemented by
  - Shared memory
    - Low capacity routers (e.g., PC-based routers)
  - Shared bus
    - Medium capacity routers
  - Point-to-point (switched) bus
    - High capacity routers



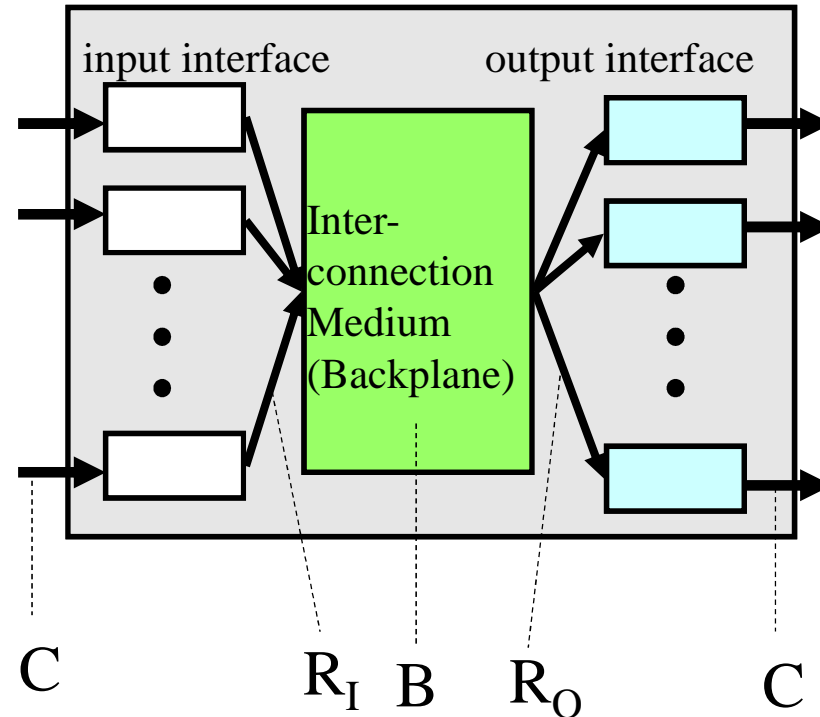
# Speedup

- $C$  – input/output link capacity
- $R_I$  – maximum rate at which an input interface can send data into backplane
- $R_O$  – maximum rate at which an output can read data from backplane
- $B$  – maximum aggregate backplane transfer rate
- Back-plane speedup:  $B/C$
- Input speedup:  $R_I/C$
- Output speedup:  $R_O/C$



# Function division

- Input interfaces:
  - **Must** perform packet forwarding – need to know to which output interface to send packets
  - May enqueue packets and perform scheduling
- Output interfaces:
  - May enqueue packets and perform scheduling

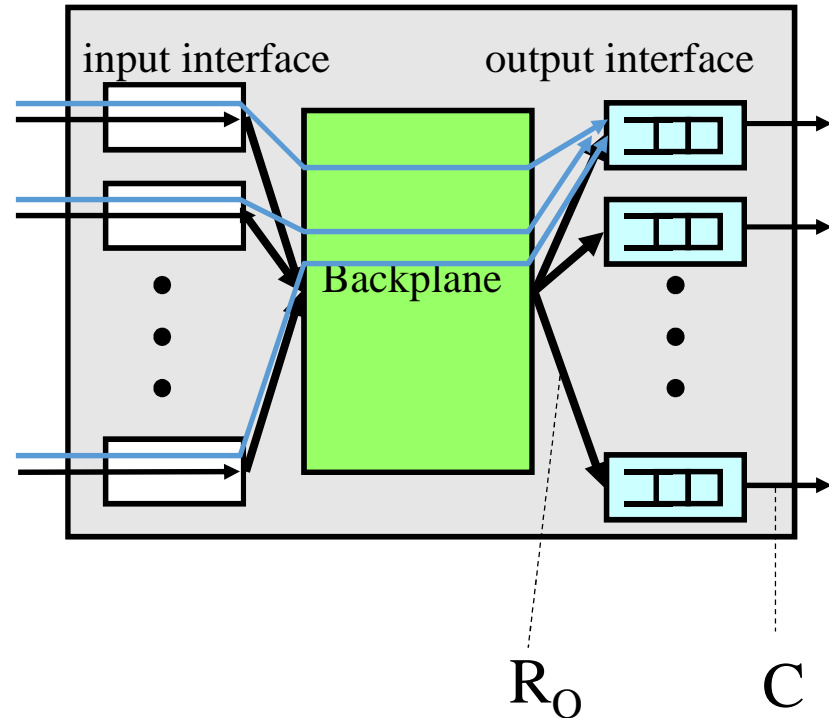


# Three Router Architectures

- Output queued
- Input queued
- Combined Input-Output queued

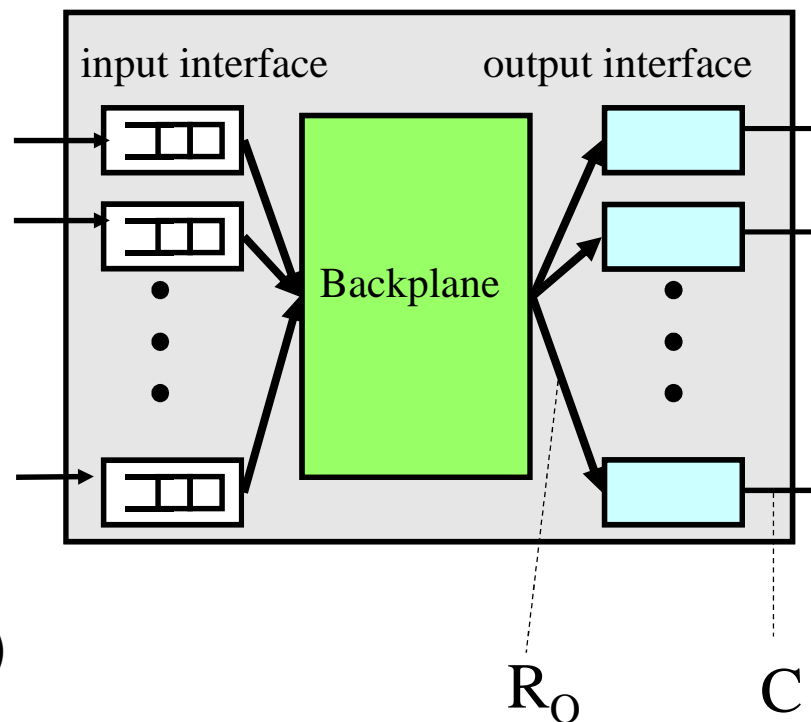
# Output Queued (OQ) Routers

- Only output interfaces store packets
- Advantages
  - Easy to design algorithms: only one congestion point
- Disadvantages
  - Requires an output speedup of  $N$ , where  $N$  is the number of interfaces  
→ not feasible



# Input Queueing (IQ) Routers

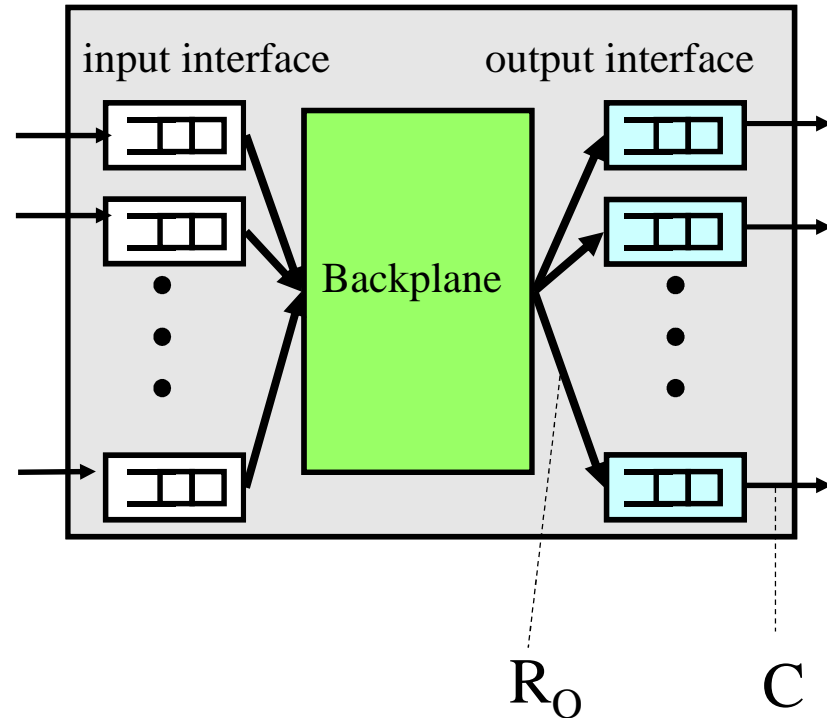
- Only input interfaces store packets
- Advantages
  - Easy to build
    - Store packets at inputs if contention at outputs
  - Relatively easy to design algorithms
    - Only one congestion point, but not output...
    - need to implement backpressure
- Disadvantages
  - Hard to achieve utilization  $\rightarrow 1$  (due to output contention, head-of-line blocking)
    - However, theoretical and simulation results show that for **realistic** traffic an input/output speedup of 2 is enough to achieve utilizations close to 1





# Combined Input-Output Queueing (CIOQ) Routers

- Both input and output interfaces store packets
- Advantages
  - Easy to built
    - Utilization 1 can be achieved with limited input/output speedup ( $\leq 2$ )
- Disadvantages
  - Harder to design algorithms
    - Two congestion points
    - Need to design flow control
  - Note: results show that with a input/output speedup of 2, a CIOQ can emulate any work-conserving OQ [G+98,SZ98]

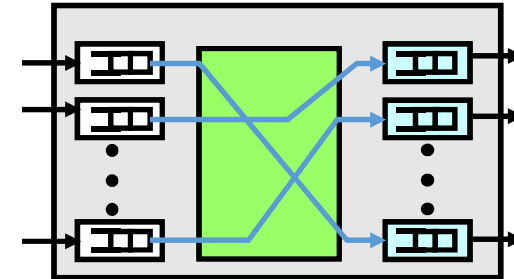


# Generic Architecture of a High Speed Router Today

- Combined Input-Output Queued Architecture
  - Input/output speedup  $\leq 2$
- Input interface
  - Perform packet forwarding (and classification)
- Output interface
  - Perform packet (classification and) scheduling
- Backplane
  - Point-to-point (switched) bus; speedup  $N$
  - Schedule packet transfer from input to output

# Backplane

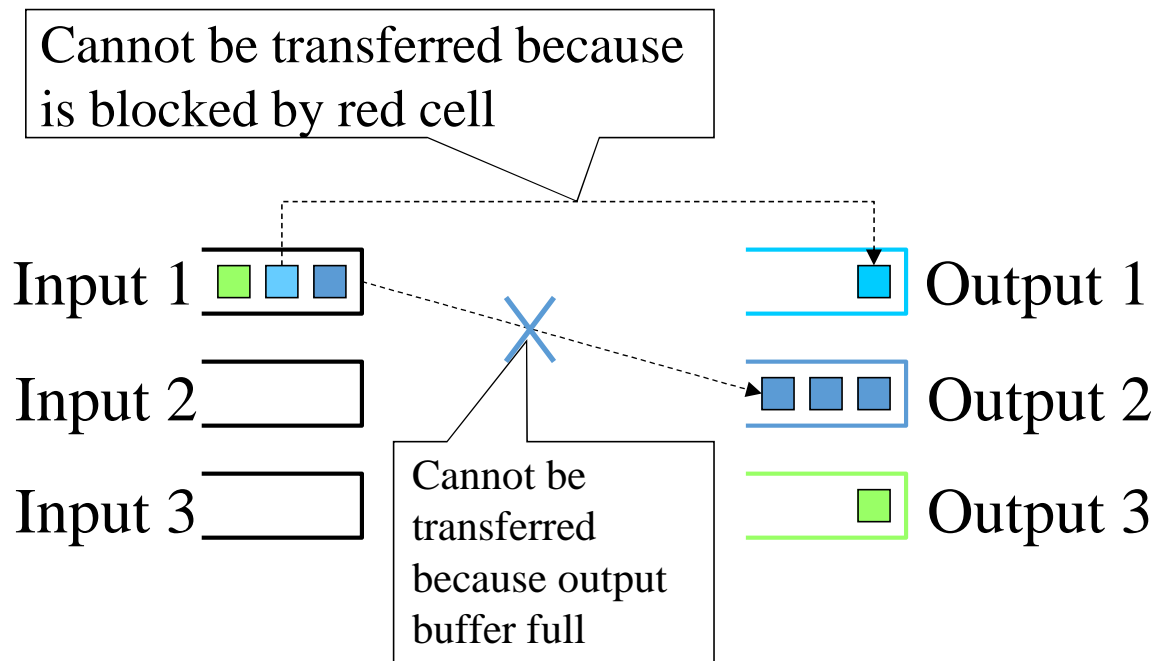
- Point-to-point switch allows to **simultaneously** transfer a packet between any two disjoint pairs of input-output interfaces
- Goal: come-up with a schedule that
  - Meet flow QoS requirements
  - Maximize router throughput
- Challenges:
  - Address head-of-line blocking at inputs
  - Resolve input/output speedups contention
  - Avoid packet dropping at output if possible
- Note: packets are fragmented in fix sized **cells** (why?) at inputs and reassembled at outputs
  - In Partridge et al, a cell is 64 B (what are the trade-offs?)





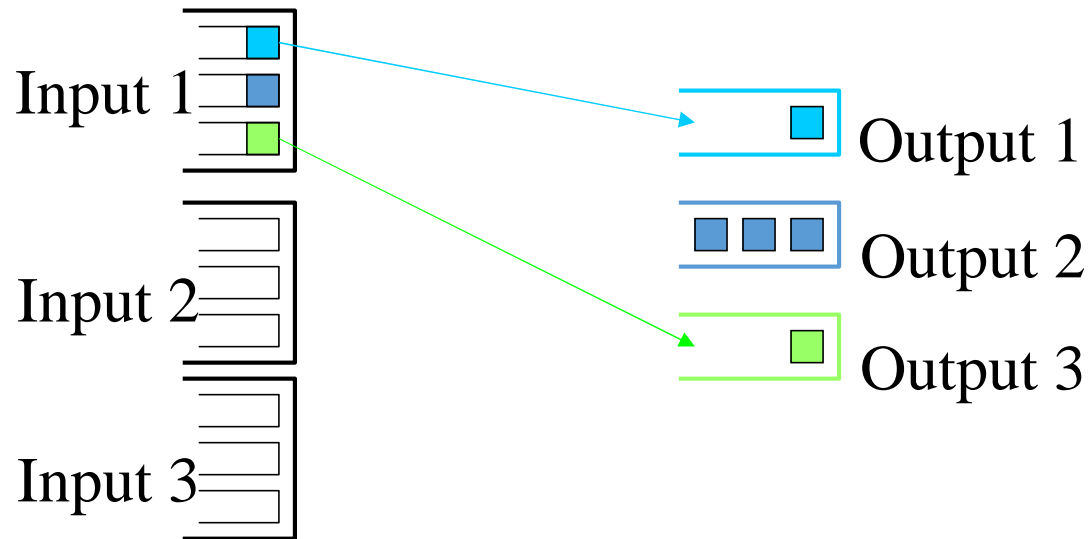
# Head-of-line Blocking

- The cell at the head of an input queue cannot be transferred, thus blocking the following cells



# Solution to Avoid Head-of-line Blocking

- Maintain at each input N virtual queues, i.e., one per output



# Cell transfer

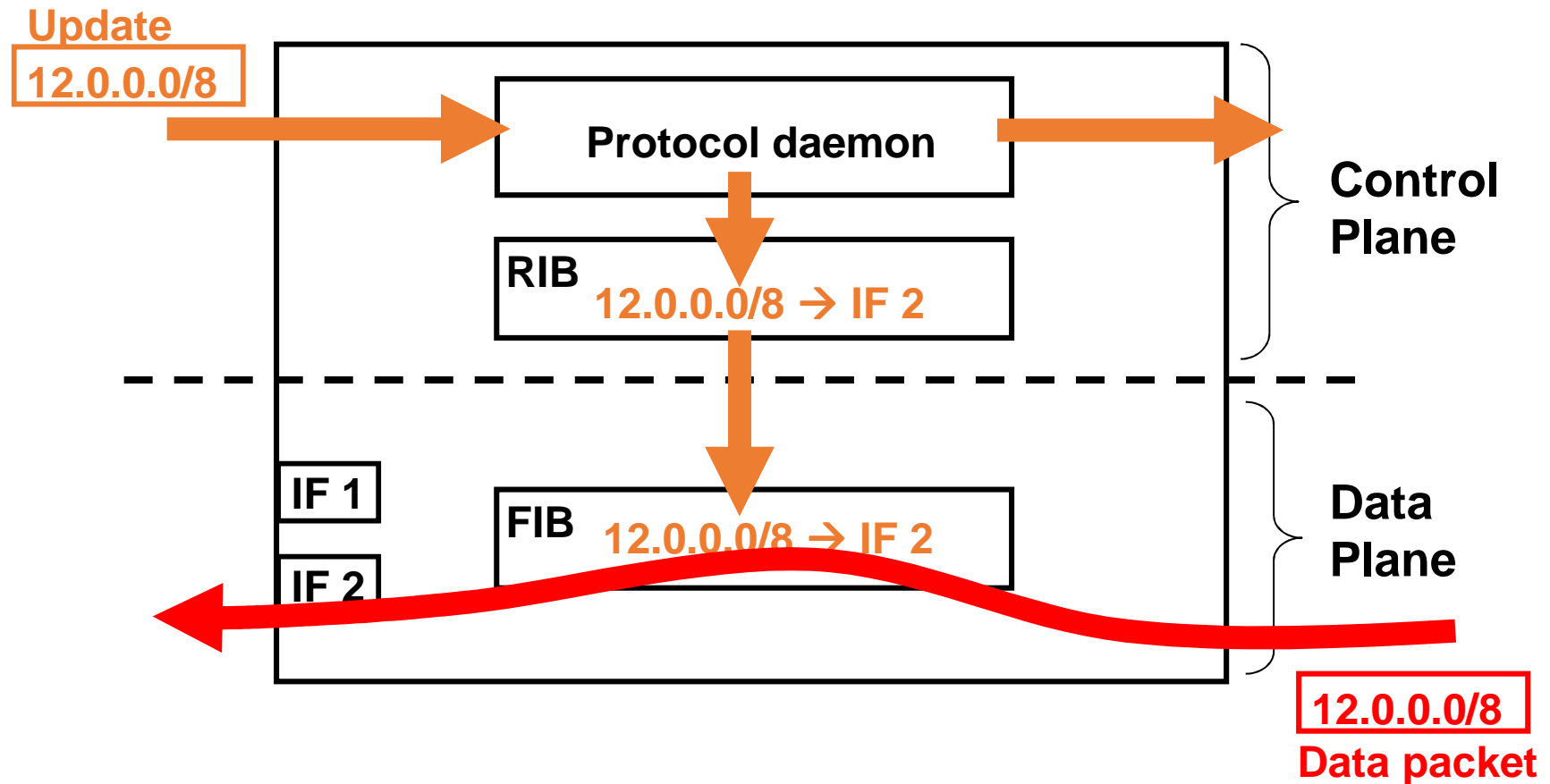
- Schedule:
  - Ideally: find the maximum number of input-output pairs such that:
    - Resolve input/output contentions
    - Avoid packet drops at outputs
    - Packets meet their time constraints (e.g., deadlines), if any
- Example
  - Assign cell preferences at inputs, e.g., their position in the input queue
  - Assign cell preferences at outputs, e.g., based on packet deadlines, or the order in which cells would depart in a OQ router
  - Match inputs and outputs based on their preferences
- Problem:
  - Achieving a high quality matching complex, i.e., hard to do in constant time

# Routing vs. Forwarding

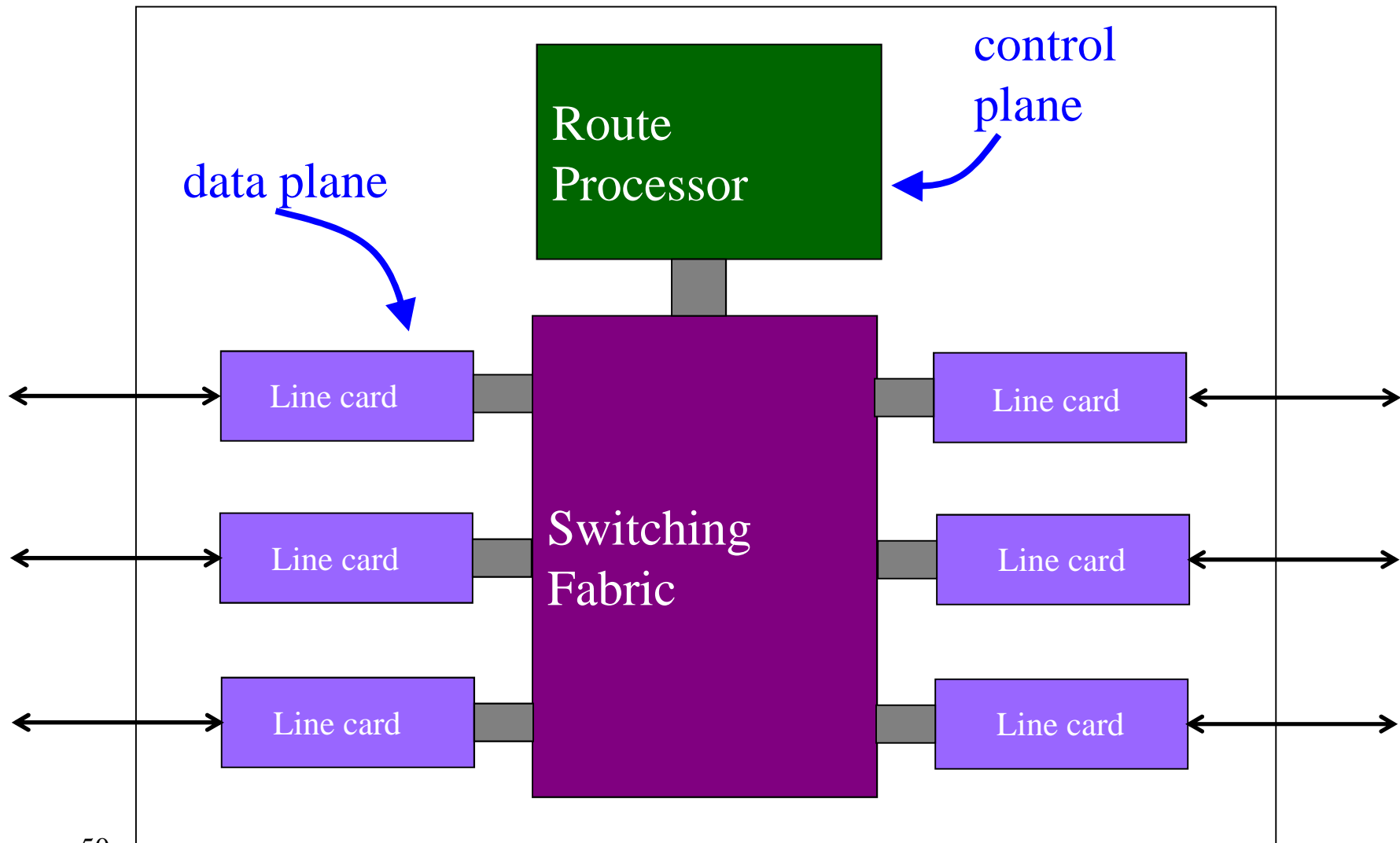
- Routing: control plane
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Individual router *creating* a forwarding table
- Forwarding: data plane
  - Directing a data packet to an outgoing link
  - Individual router *using* a forwarding table



# How the control and data planes work together (logical view)

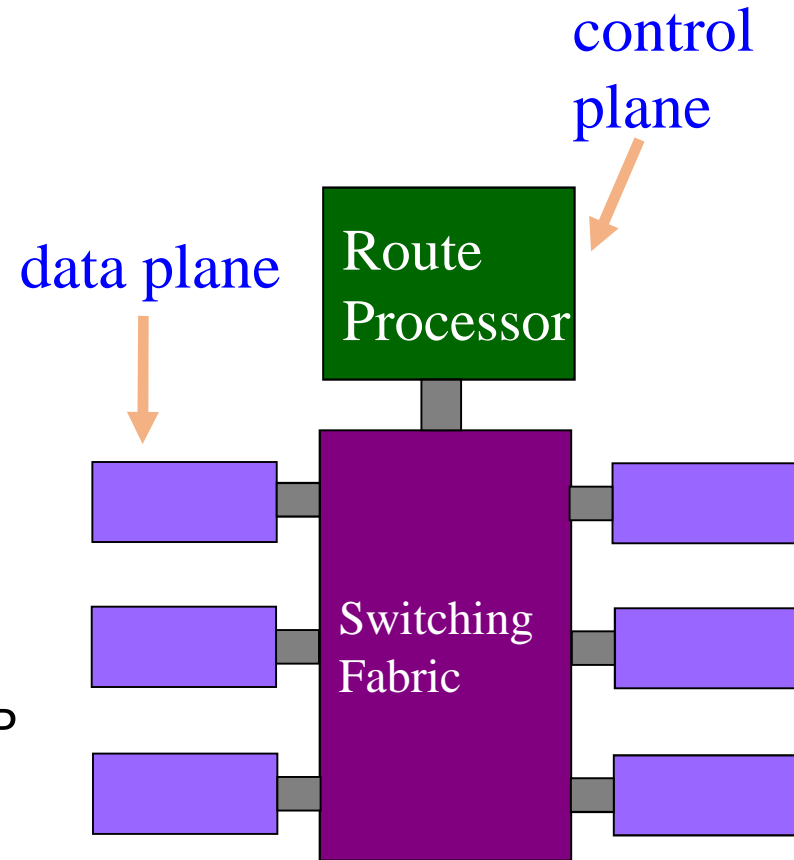


# Physical layout of a high-end router



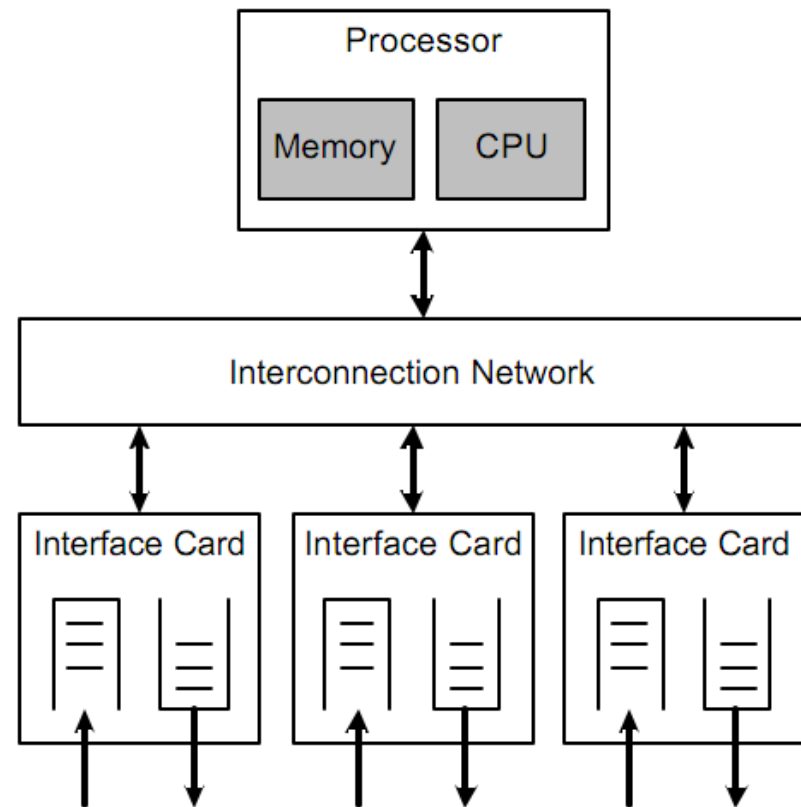
# Routing vs. Forwarding

- Control plane's jobs include
  - Route calculation
  - Maintenance of routing table
  - Execution of routing protocols
- On commercial routers, handled by special-purpose processor called "route processor"
- IP forwarding is per-packet processing
  - On high-end commercial routers, IP forwarding is distributed
  - Most work is done by interface cards

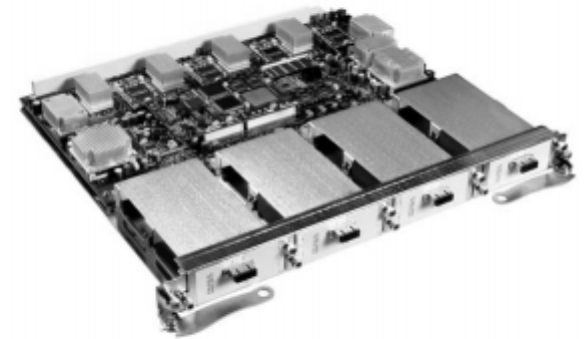
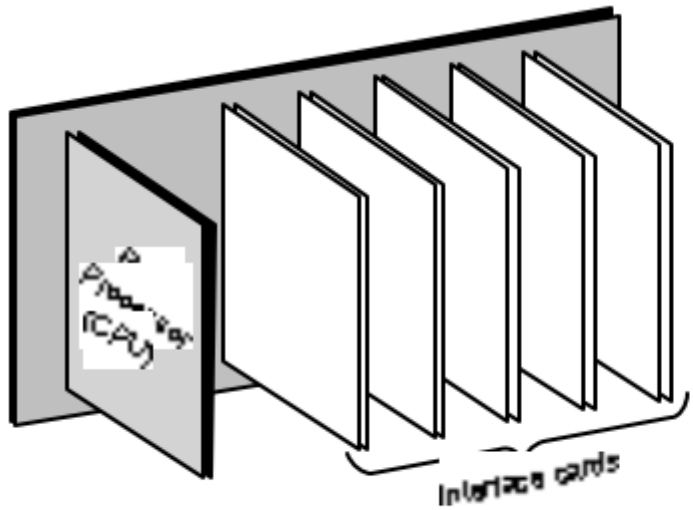


# Router Components

- On a PC router:
  - Interconnection network is the PCI bus
  - Interface cards are the NICs (e.g., Ethernet cards)
  - All forwarding and routing is done on a commodity CPU
- On commercial routers:
  - Interconnection network and interface cards are sophisticated, special-purpose hardware
  - Packet forwarding oftend implemented in a custom ASIC
  - Only routing (control plane) is done on the commodity CPU (route processor)



# Slotted Chassis



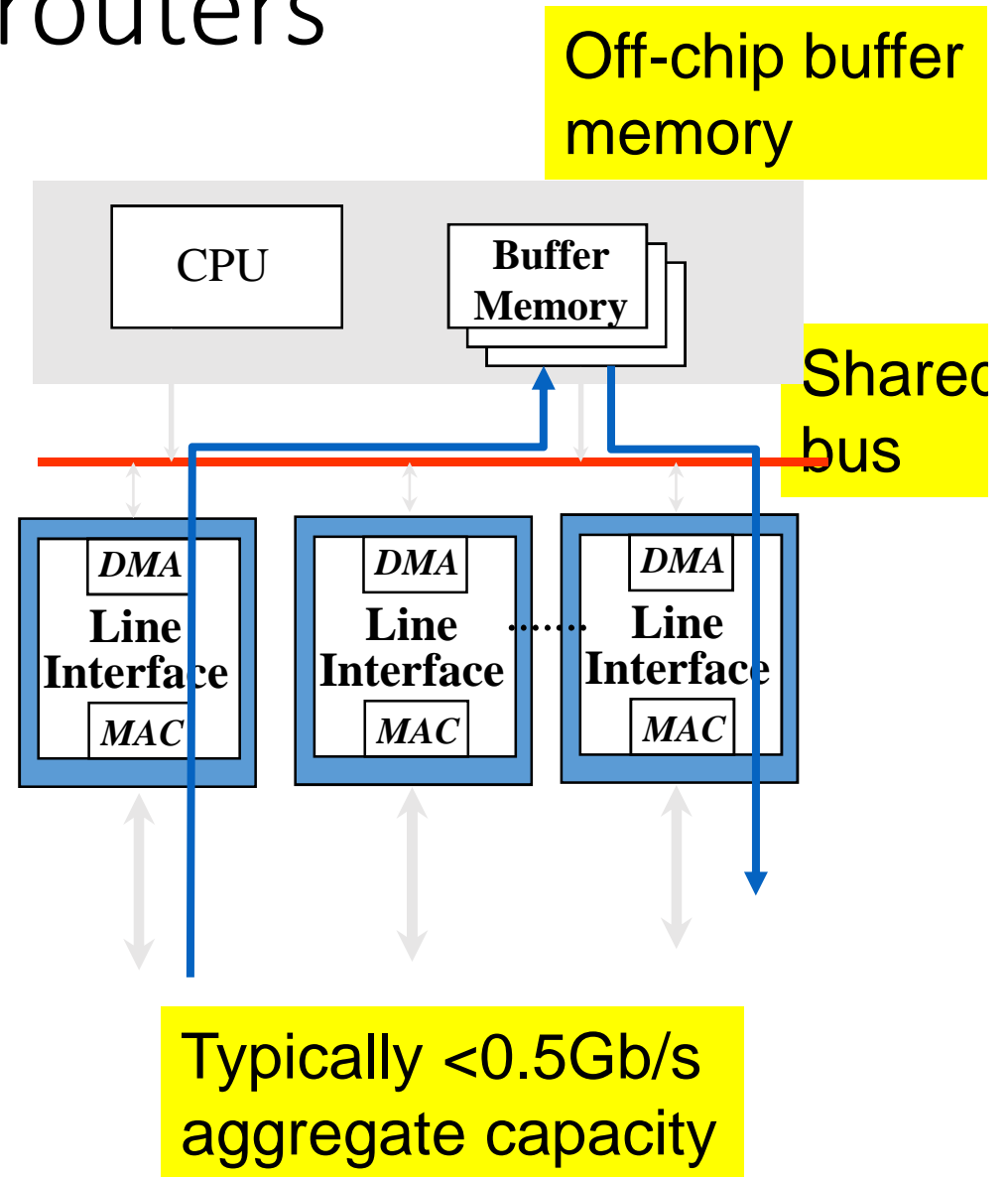
- Large routers are built as a slotted chassis
  - Interface cards are inserted in the slots
  - Route processor is also inserted as a slot
- This simplifies repairs and upgrades of components
  - E.g., “hot-swapping” of components

# Evolution of router architectures

- Early routers were just general-purpose computers
- Today, high-performance routers resemble mini data centers
  - Exploit parallelism
  - Specialized hardware
- Until 1980s (1<sup>st</sup> generation): standard computer
- Early 1990s (2<sup>nd</sup> generation): delegate packet processing to interfaces
- Late 1990s (3<sup>rd</sup> generation): distributed architecture
- Today: distributed across multiple racks

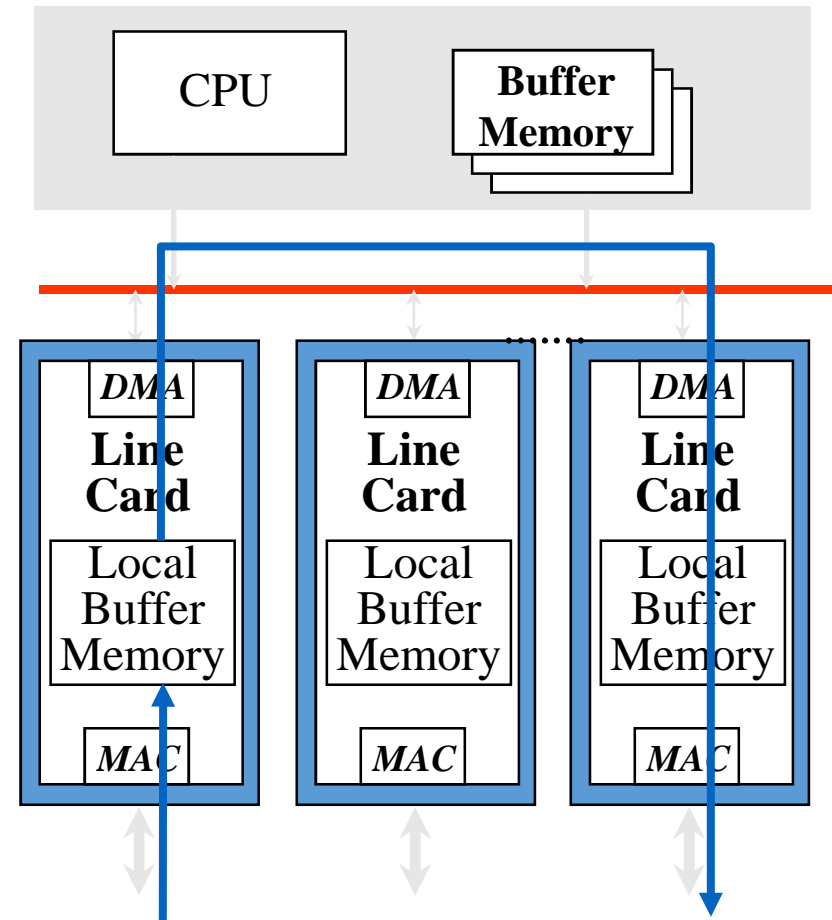
# First generation routers

- This architecture is still used in low-end routers
- Arriving packets are copied to main memory via direct memory access (DMA)
- Interconnection network is a backplane (shared bus)
- All IP forwarding functions are performed by a commodity CPU
- Routing cache at processor can accelerate the routing table lookup
- Drawbacks:
  - Forwarding performance is limited by the CPU
  - Capacity of shared bus limits the number of interface cards that can be connected



# Second generation routers

- Bypasses memory bus with direct transfer over bus between line cards
- Moves forwarding decisions local to card to reduce CPU utilization
- Trap to CPU for “slow” operations



Typically <5Gb/s aggregate capacity



# Speeding up the common case with a “Fast path”

- IP packet forwarding is complex
  - But, vast majority of packets can be forwarded with simple algorithm
  - Main idea: put common-case forwarding in hardware, trap to software on exceptions
  - Example: BBN router had 85 instructions for fast-path code, which fits entirely in L1 cache
- Non-common cases handled by slow path:
  - Route cache misses
  - Errors (e.g., ICMP time exceeded)
  - IP options
  - Fragmented packets
  - Multicast packets

# Improving upon second-generation routers

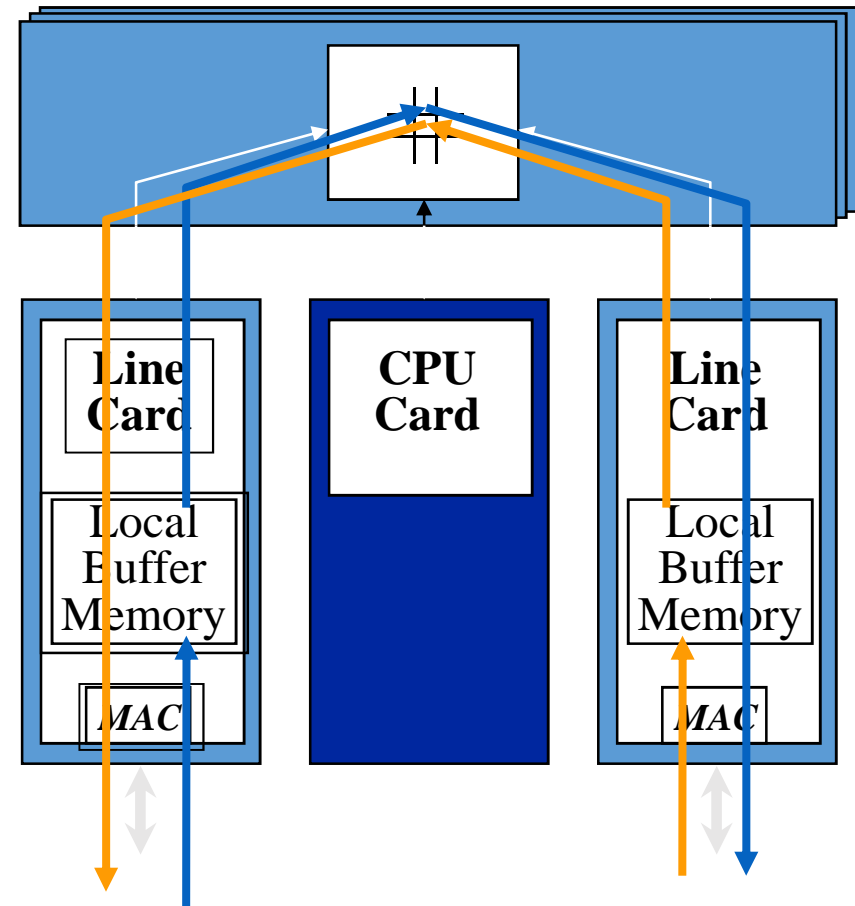
- Control plane must remember lots of information (BGP attributes, etc.)
  - But data plane only needs to know FIB
  - Smaller, fixed-length attributes
  - Idea: store FIB in hardware
- Going over the bus adds delay
  - Idea: Cache FIB in line cards
  - Send directly over bus to outbound line card

# Improving upon second-generation routers

- Shared bus is a big bottleneck
  - E.g., *modern* PCI bus (PCIx16) is only 32Gbit/sec (in theory)
  - Almost-modern Cisco (XR 12416) is 320 Gbit/sec
  - Ow! How do we get there?
  - Idea: put a “network” inside the router
    - Switched backplane for larger cross-section bandwidths

# Third generation routers

- Replace bus with interconnection network (e.g., a crossbar switch)
- Distributed architecture:
  - Line cards operate independently of one another
  - No centralized processing for IP forwarding
- These routers can be scaled to many hundreds of interface cards and capacity of > 1 Tbit/sec



## Technology: Cisco Introduces a 322 Tbit/sec. Router

Posted by [kdawson](#) on Tuesday March 09 2010, @02:45PM  
from the one-loc-per-second dept.

CWmike writes

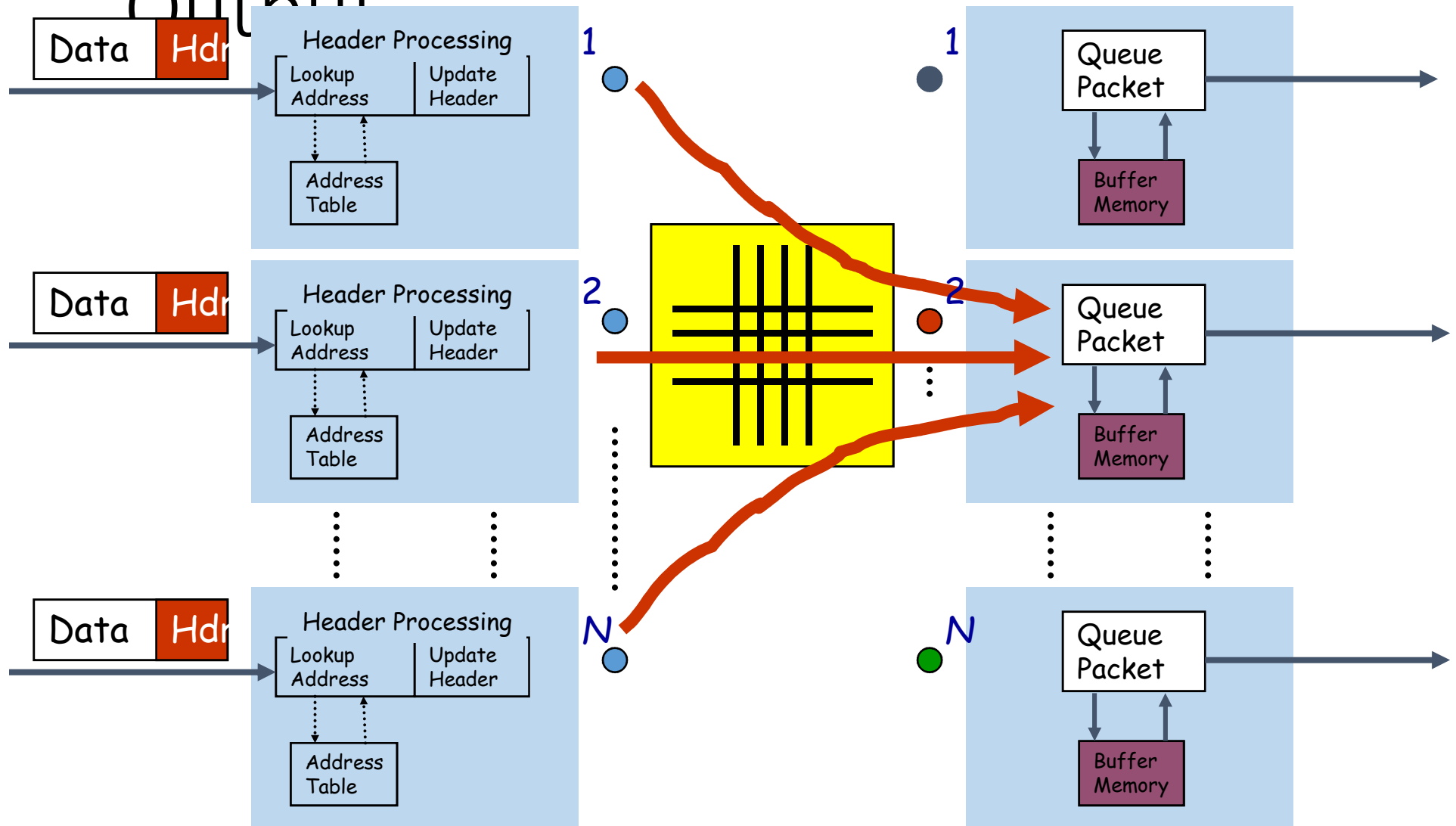
"Today Cisco Systems introduced its next-generation Internet core router, the CRS-3, with about three times the capacity of its current platform. 'The Internet will scale faster than any of us anticipate,' Cisco's John Chambers said while announcing the product. At full scale, the [CRS-3 has a capacity of 322Tbit/sec.](#), roughly three times that of the CRS-1, introduced in 2004. It also has more than 12 times the capacity of its nearest competitor, Chambers said. The CRS-3 will help the Internet evolve from a messaging to an entertainment and media platform, with video emerging as the 'killer app,' Chambers said. Using a CRS-3, every person in China, which has a population of 1.3 billion, could participate in a video phone call at the same time. (Or you could give nearly one Library of Congress per second through the device, or give everyone in San Francisco a 1Gbps internet connection.) AT&T said it has been using the CRS-3 to test 100Gbit/sec. data links in tests on a commercial fiber route in Florida and Louisiana."

- Multi-chassis router
  - A single router that is a distributed collection of racks
  - Scales to 322 Tbps, can replace an entire PoP



# Switch Fabric: From Input to

## Output



# Crossbars

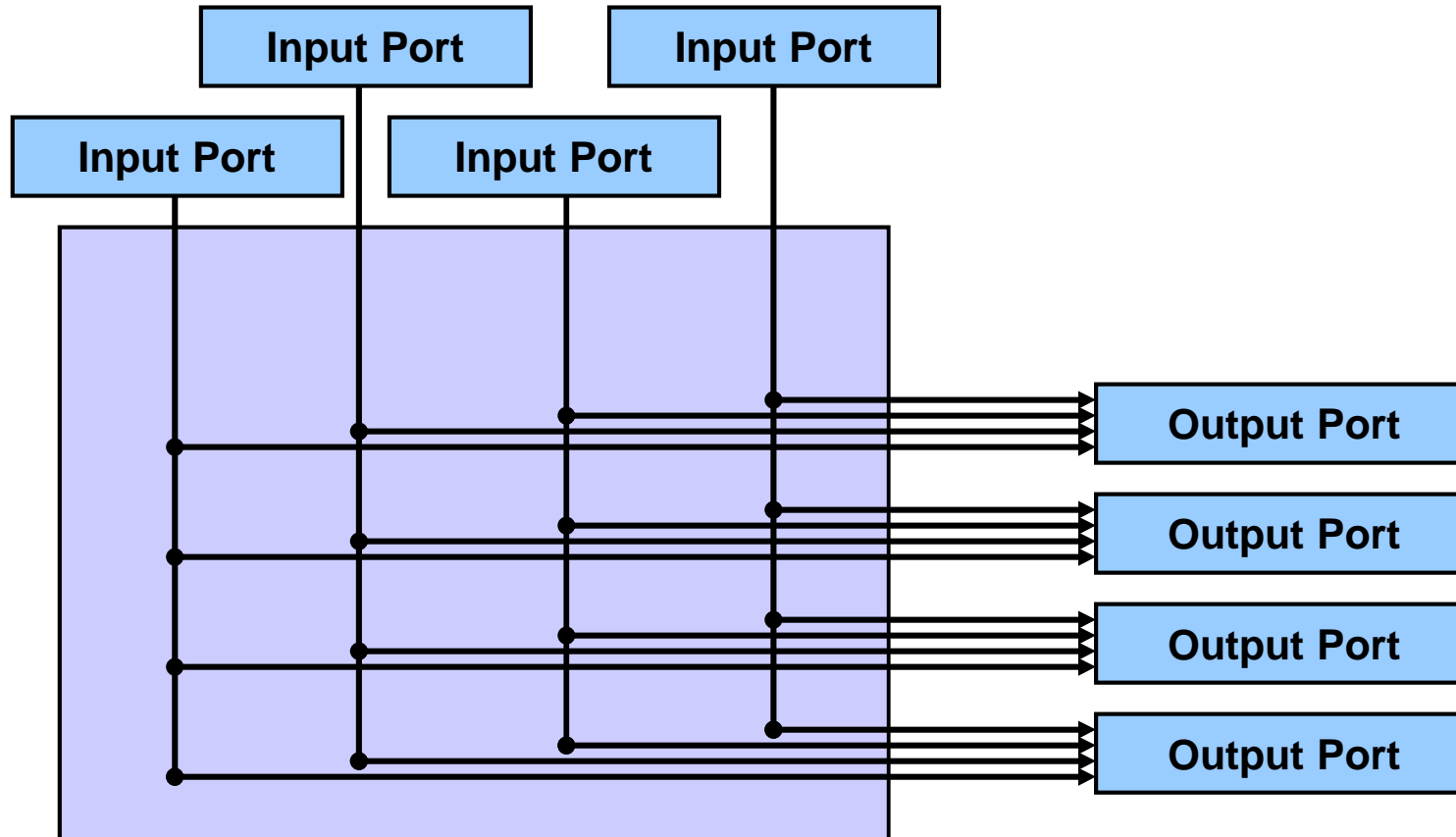
- N input ports, N output ports
  - One per line card, usually
- Every line card has its own forwarding table/classifier/etc --- removes CPU bottleneck
- Scheduler
  - Decides which input/output port pairs to connect in a given *time slot*
  - Often forward fixed-sized “cells” to avoid variable-length time slots
  - Crossbar constraint
    - If input  $i$  is connected to output  $j$ , no other input connected to  $j$ , no other output connected to  $i$
    - Scheduling is a bipartite matching

# Crossbar Switch

- Every input port is connected to every output port
  - $N \times N$
- Output ports
  - Complexity scales as  $O(N^2)$



# Crossbar Switch



# Knockout Switch

- Full crossbar requires each output port to handle up to  $N$  input packets
- $N$  simultaneous inputs for the same output is unlikely, especially in a large switch
- Instead, let's implement each port to handle  $L < N$  packets at the same time
- Hard issue: what value of  $L$  to use?

# Knockout switch

- Components:
  - Packet filters (recognize packets destined for this output port)
  - Concentrator (selects subset of  $L$  packets, “knocks out” others)
  - A queue with capacity  $L$  packets

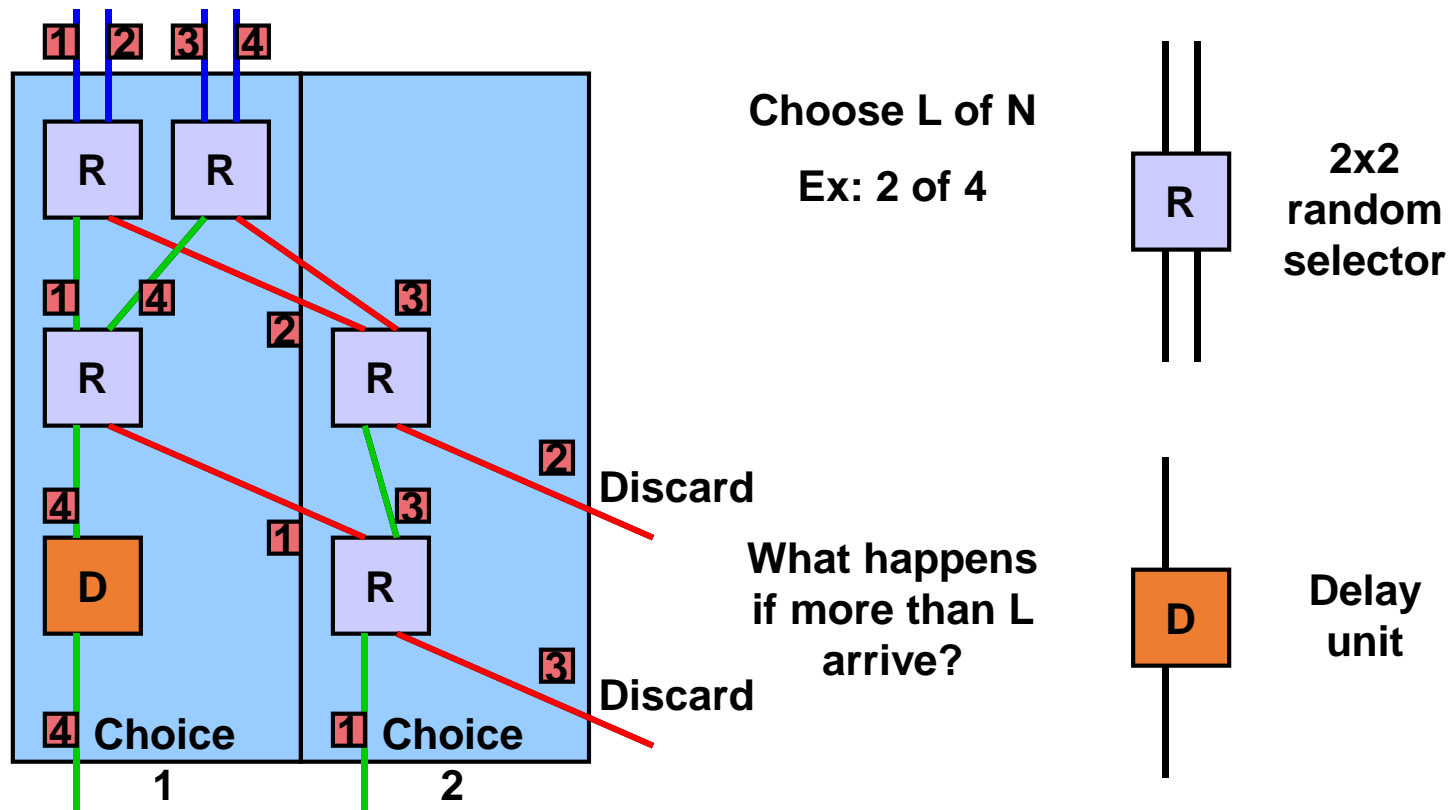
# Knockout switch

- Want some fairness: no single input should have its packets always “knocked out”
- Essentially a “knock out” tennis tournament with each game of 2 players (packets) chosen randomly
- Overall winner is selected by playing  $\log N$  rounds, and keeping the winner

# Knockout switch

- Pick  $L$  from  $N$  packets at a port
  - Output port maintains  $L$  cyclic buffers
  - Shifter places up to  $L$  packets in one cycle
  - Each buffer gets only one packet
  - Output port uses round-robin between buffers
  - Arrival order is maintained
- Output ports scale as  $O(N)$

# Knockout Switch

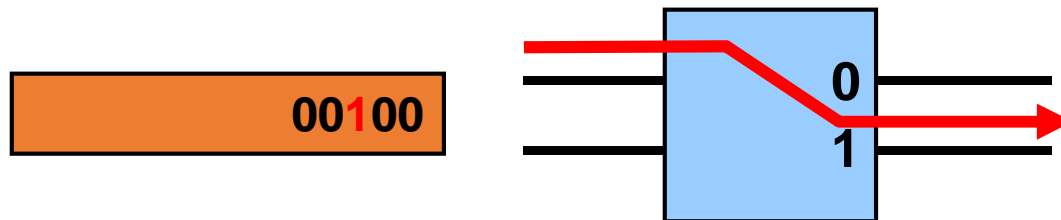


# Self-Routing Fabrics

- Idea
  - Use source routing on “network” in switch
  - Input port attaches output port number as header
  - Fabric routes packet based on output port
- Types
  - Banyan Network
  - Batchter-Banyan Network
  - Sunshine Switch

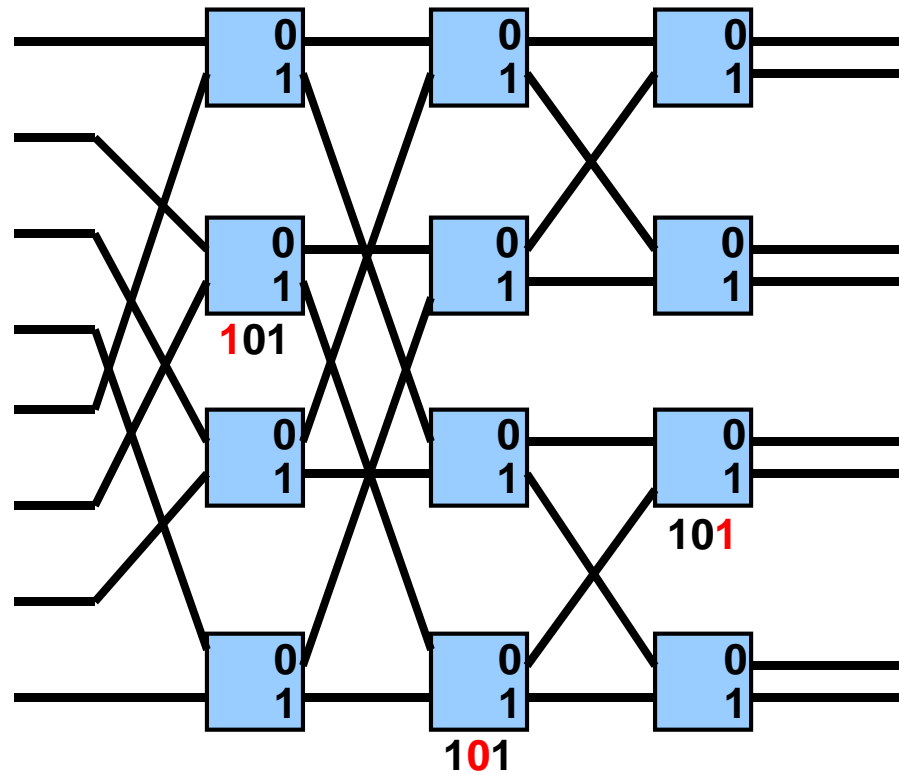
# Banyan Network

- A network of 2x2 switches
  - Each element routes to output 0 or 1 based on packet header
  - A switching element at stage  $i$  looks at bit  $i$  in the header

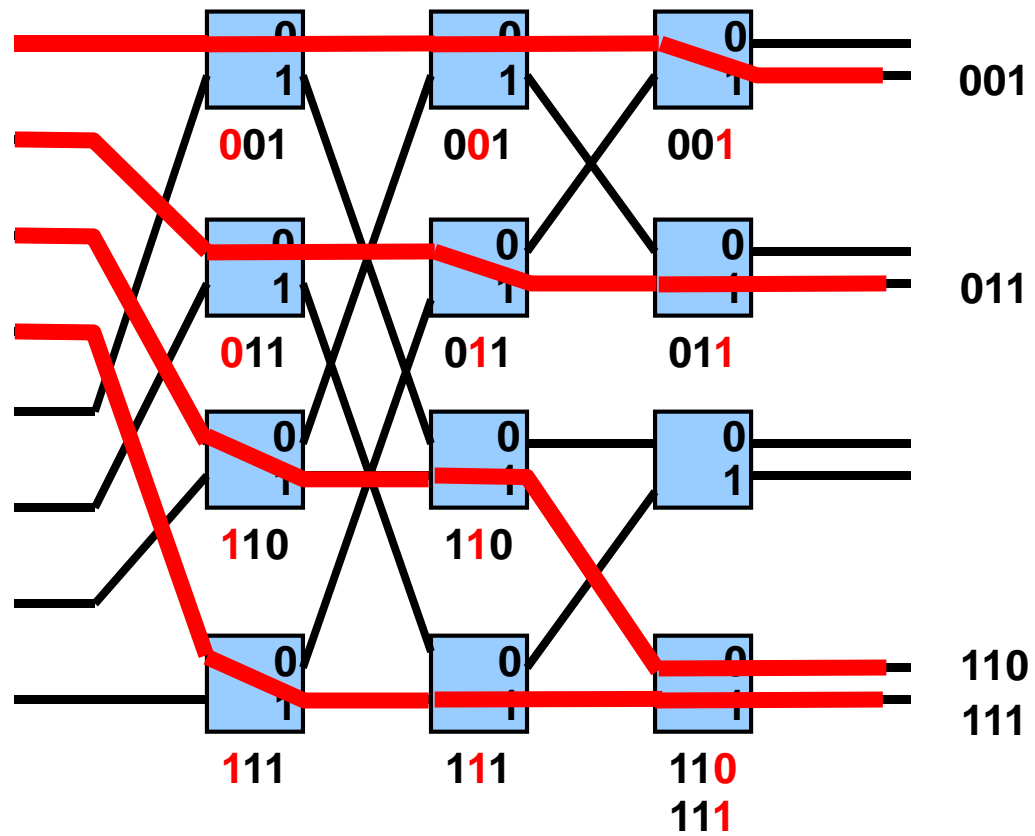




# Banyan Network



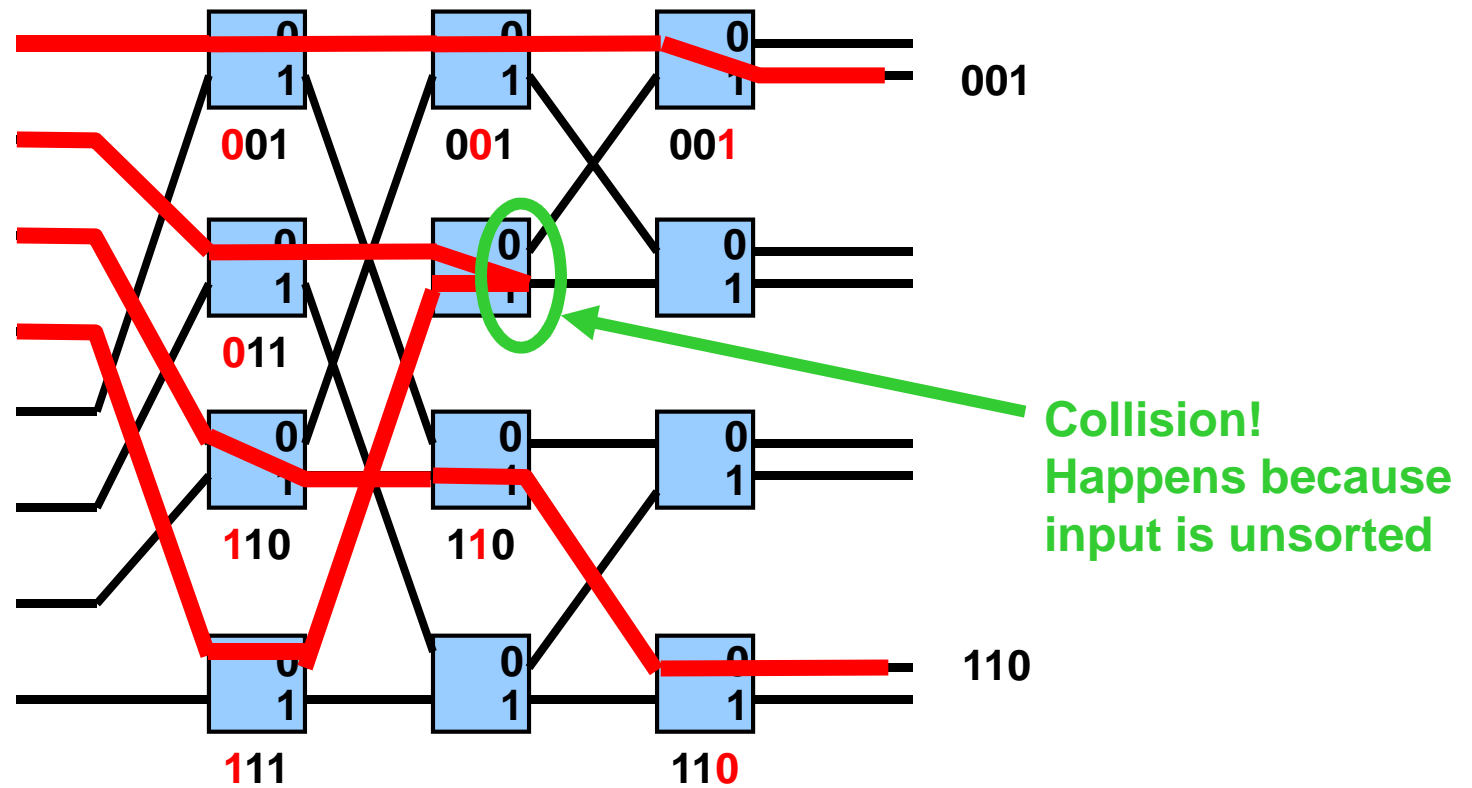
# Banyan Network



# Banyan Network

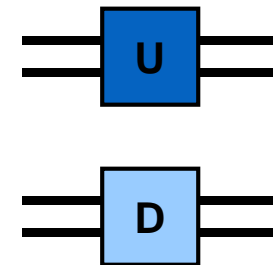
- Perfect Shuffle
  - N inputs requires  $\log_2 N$  stages of  $N/2$  switching elements
  - Complexity on order of  $N \log_2 N$
- Collisions
  - If two packets arrive at the same switch destined for the same output port, a collision will occur
  - If all packets are sorted in ascending order upon arrival to a banyan network, no collisions will occur!

# Collision in a Banyan Network

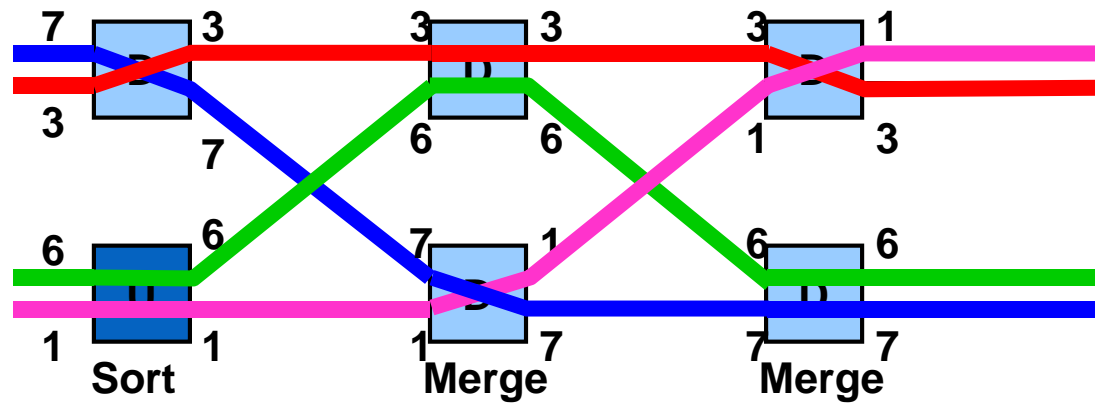


# Batcher Network

- Performs merge sort
- A network of 2x2 switches
  - Each element routes to output 0 or 1 based on packet header
  - A switch at stage  $i$  looks at the whole header
  - Two types of switches
    - Up switch
      - Sends higher number to top output (0)
    - Down switch
      - Sends higher number to bottom output (1)



# Batcher Network

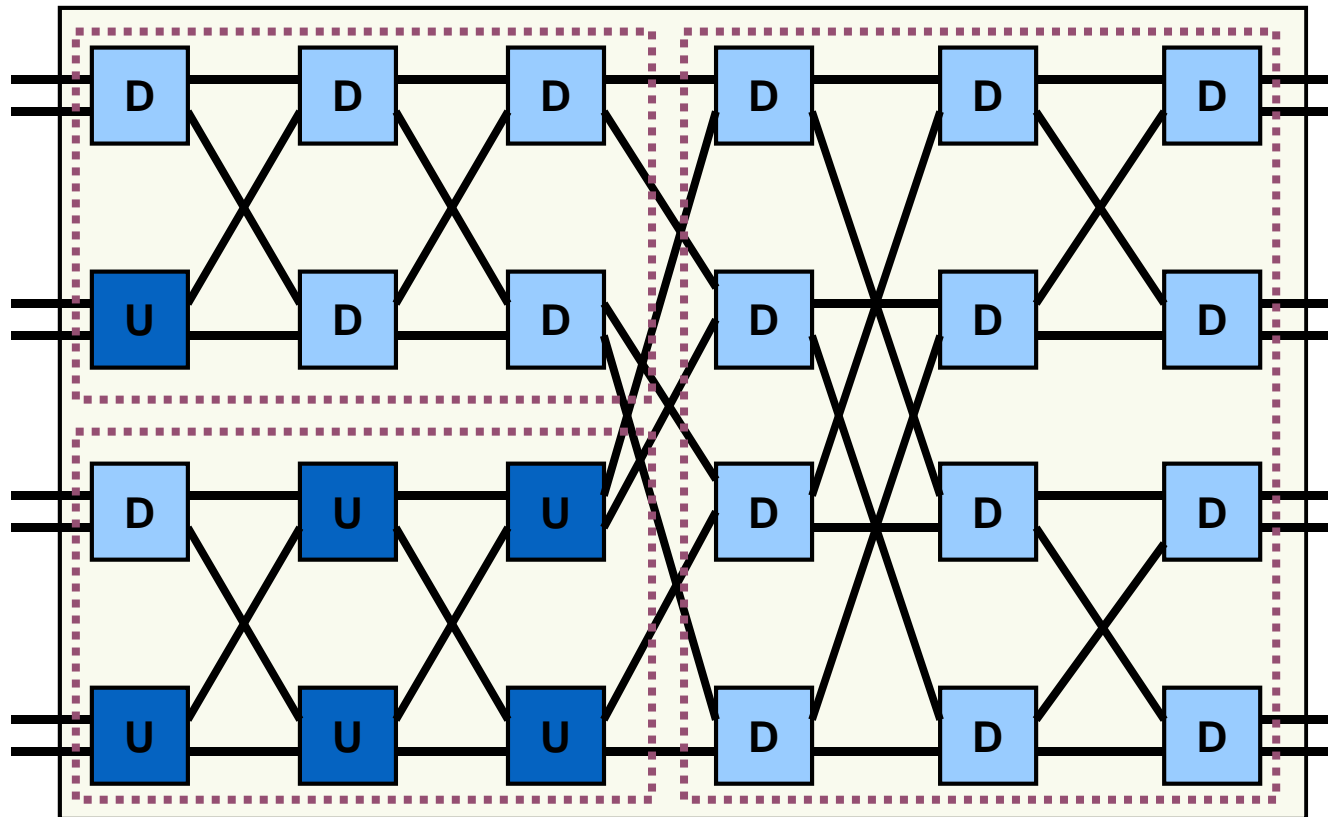


# Batcher Network

Sort inputs 0 – 3 in ascending order

Merge 0 – 3 with 4 – 7

8x8  
Switch



Sort inputs 4 – 7 in descending order

# Batcher Network

- How it really works
  - Merger is presented with a pair of sorted lists, one in ascending order, one in descending order
  - First stage of merger sends packets to the correct half of the network
  - Second stage sends them to the correct quarter
- Size
  - $N/2$  switches per stage
  - $\log_2 N \times (1 + \log_2 N)/2$  stages
  - Complexity =  $N \log_2^2 N$



# Batcher-Banyan Network

- Idea

- Attach a batcher network back-to-back with a banyan network
- Arbitrary unique permutations can be routed without contention

# Data Plane Details: Checksum

- Takes too much time to verify checksum
  - Increases forwarding time by 21%
- Take an optimistic approach: just incrementally update it
  - Safe operation: if checksum was correct it remains correct
  - If checksum bad, it will be anyway caught by end-host
- Note: IPv6 does not include a header checksum anyway!

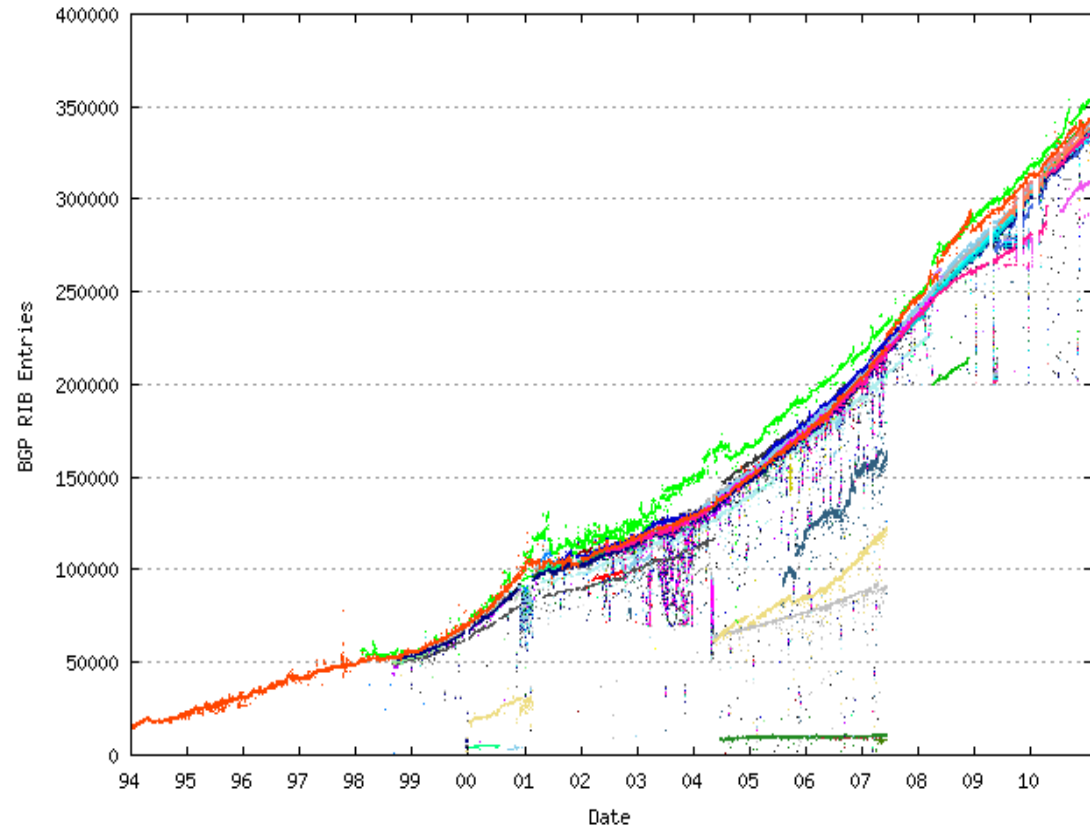
# Matching Algorithms

# What's so hard about IP packet forwarding?

- Back-of-the-envelope numbers
  - Line cards can be 40 Gbps today (OC-768)
    - Getting faster every year!
  - To handle minimum-sized packets (~40b)
    - 125 Mpps, or 8ns per packet
    - Can use parallelism, but need to be careful about reordering
- For each packet, you must
  - Do a routing lookup (where to send it)
  - Schedule the crossbar
  - Maybe buffer, maybe QoS, maybe ACLs,...

# Routing lookups

- Routing tables: 200,000 to 1M entries
  - Router must be able to handle routing table loads 5-10 years hence
- How can we store routing state?
  - What kind of memory to use?
- How can we quickly lookup with increasingly large routing tables?



# Memory technologies

Technology	Single chip density	\$/MByte	Access speed	Watts/ chip
Dynamic RAM (DRAM) <i>cheap, slow</i>	64 MB	\$0.50- \$0.75	40-80ns	0.5-2W
Static RAM (SRAM) <i>expensive, fast, a bit higher heat/power</i>	4 MB	\$5-\$8	4-8ns	1-3W
Ternary Content Addressable Memory (TCAM) <i>very expensive, very high heat/power, very fast (does parallel lookups in hardware)</i>	1 MB	\$200-\$250	4-8ns	15-30W

- Vendors moved from DRAM (1980s) to SRAM (1990s) to TCAM (2000s)
- Vendors are now moving back to SRAM and parallel banks of DRAM due to power/heat

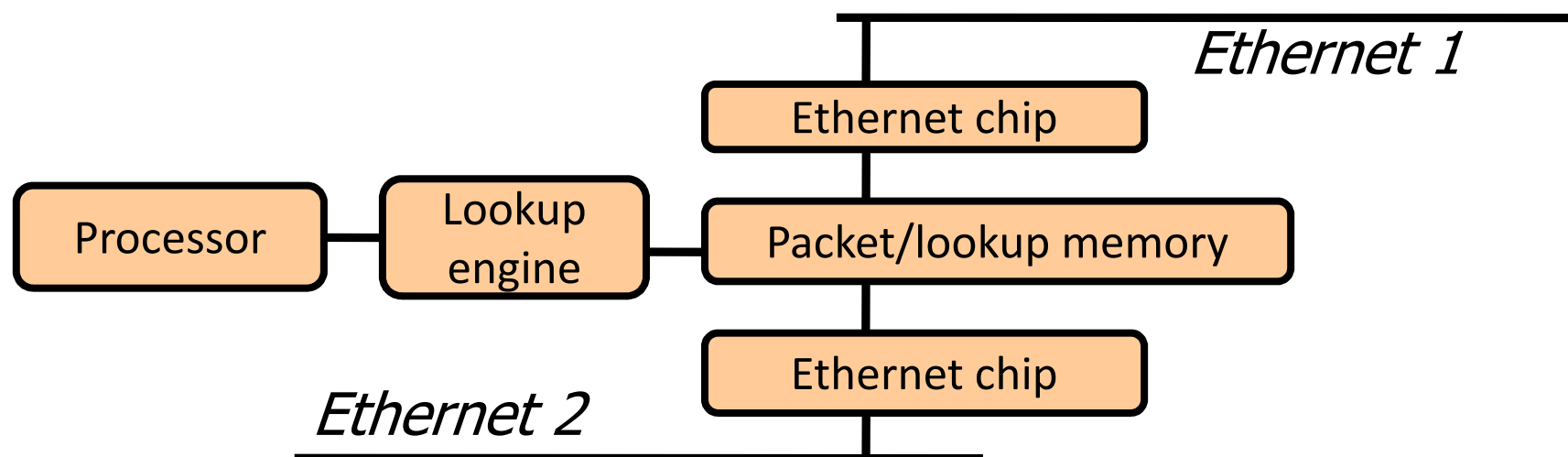
# Fixed-Length Matching Algorithms

# Ethernet Switch

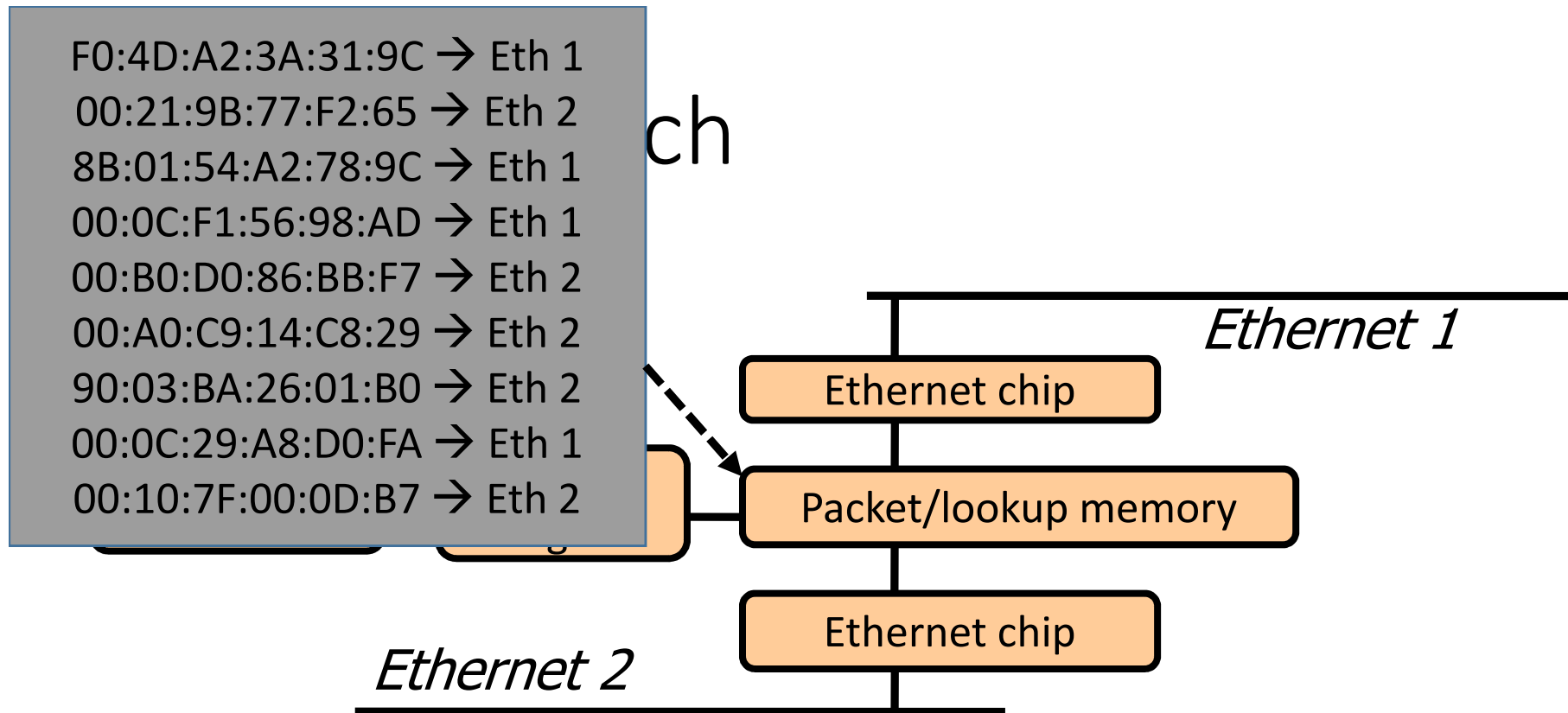
- Lookup frame DA in forwarding table.
  - If known, forward to correct port.
  - If unknown, broadcast to all ports.
- Learn SA of incoming frame.
- Forward frame to outgoing interface.
- Transmit frame onto link.
- How to do this quickly?
  - Need to determine next hop quickly
  - Would like to do so without reducing line rates



# Inside a switch

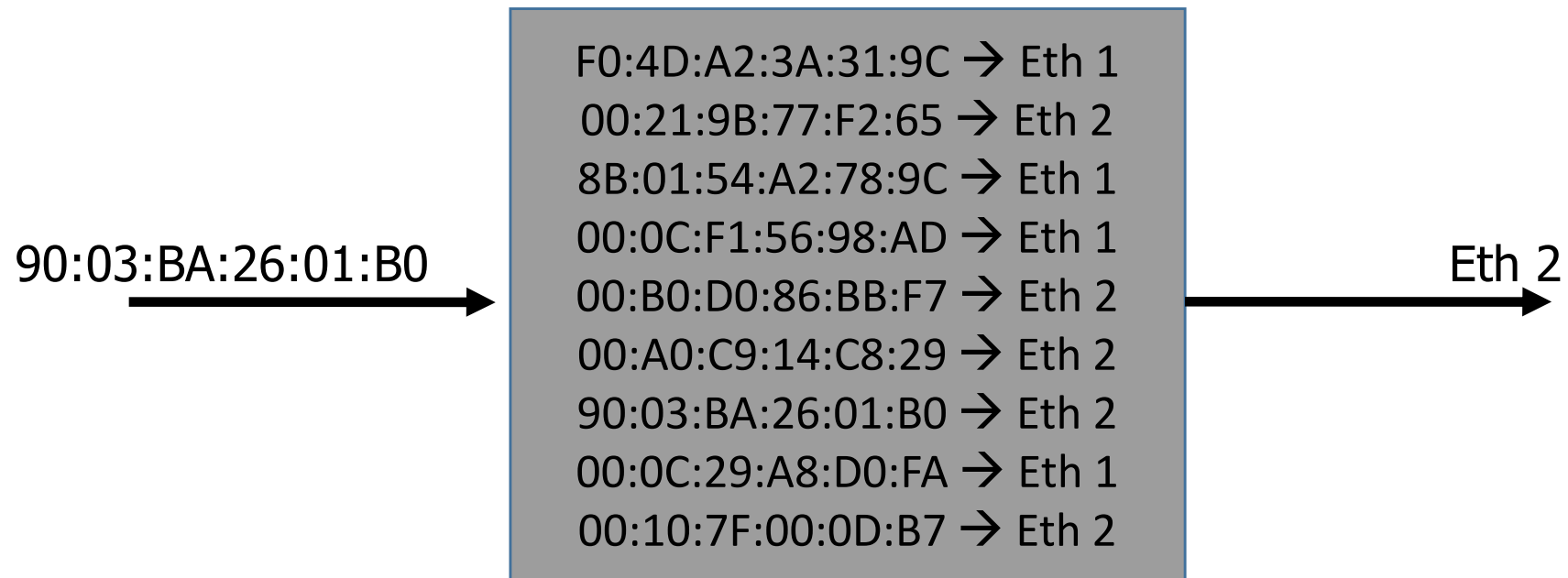


- Packet received from upper Ethernet
- Ethernet chip extracts source address  $S$ , stored in shared memory, in receive queue
  - Ethernet chips set in “promiscuous mode”
- Extracts destination address  $D$ , given to lookup engine



- Lookup engine looks up D in database stored in memory
  - If destination is on upper Ethernet: set packet buffer pointer to free queue
  - If destination is on lower Ethernet: set packet buffer pointer to transmit queue of the lower Ethernet
- How to do the lookup quickly?

# Problem overview



- Goal: given address, look up outbound interface
  - Do this quickly (few instructions/low circuit complexity)
- Linear search too low

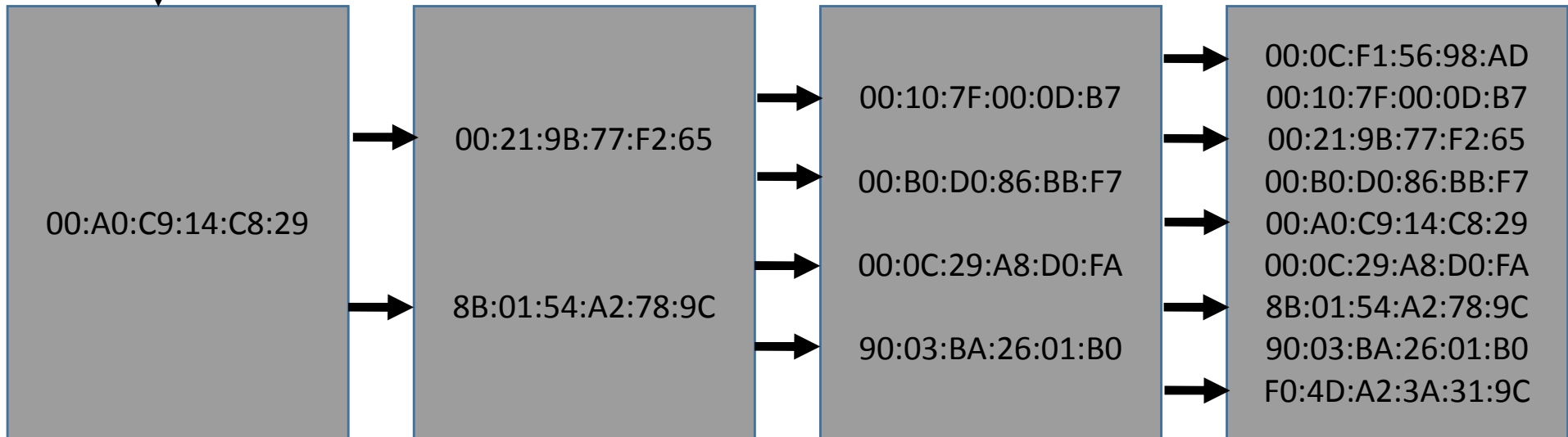
# Idea #1: binary search



- Put all destinations in a list, sort them, binary search
- Problem: logarithmic time

8B:01:54:A2:78:9C  
 F0:4D:A2:3A:31:9C  
 00:10:7F:00:0D:B7

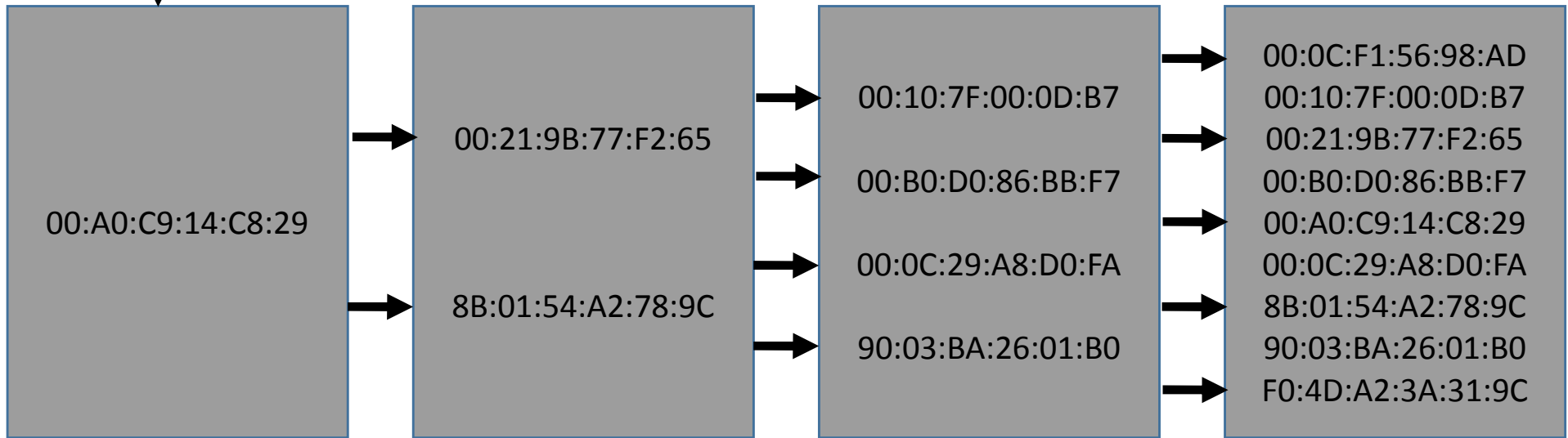
# Element: Binary search



- Packets still have  $O(\log n)$  delay, but can process  $O(\log n)$  packets in parallel  $\rightarrow O(1)$

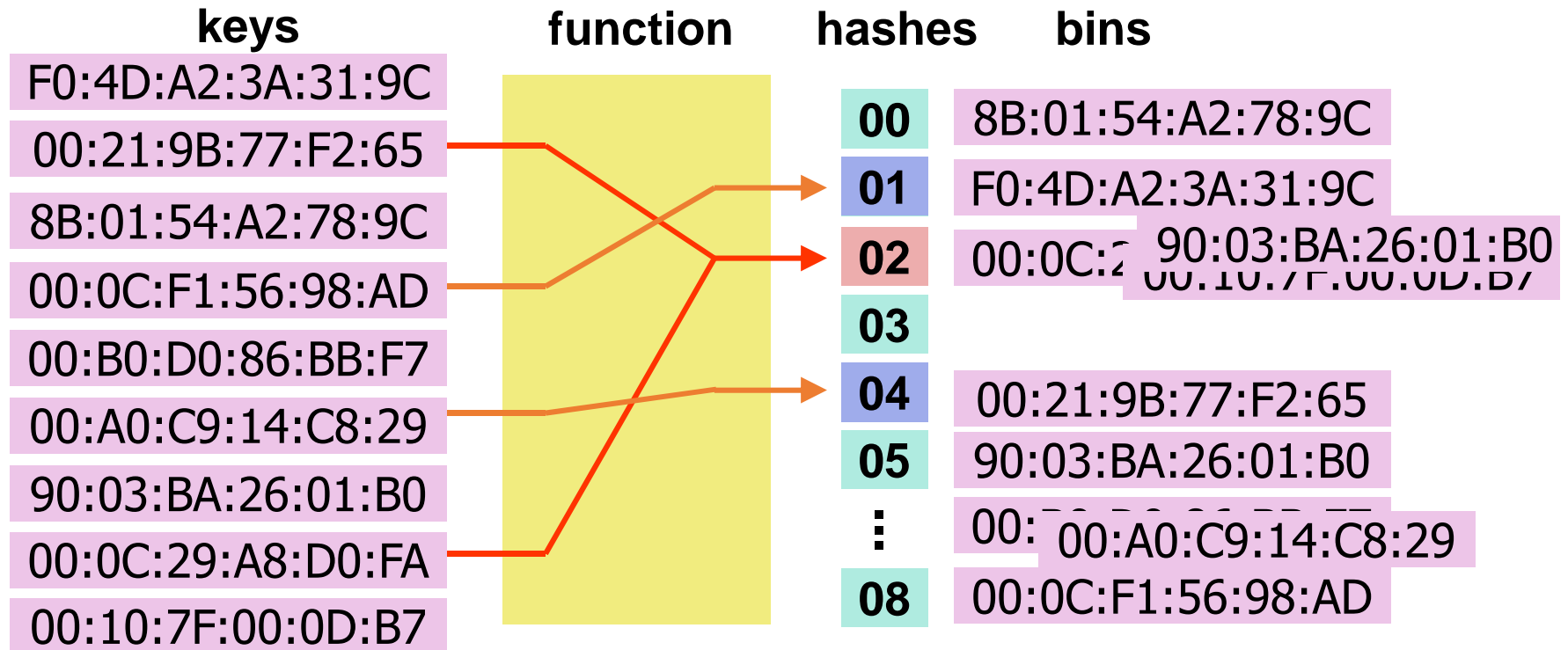
8B:01:54:A2:78:9C  
 F0:4D:A2:3A:31:9C  
 00:10:7F:00:0D:B7

# Element: Binary search



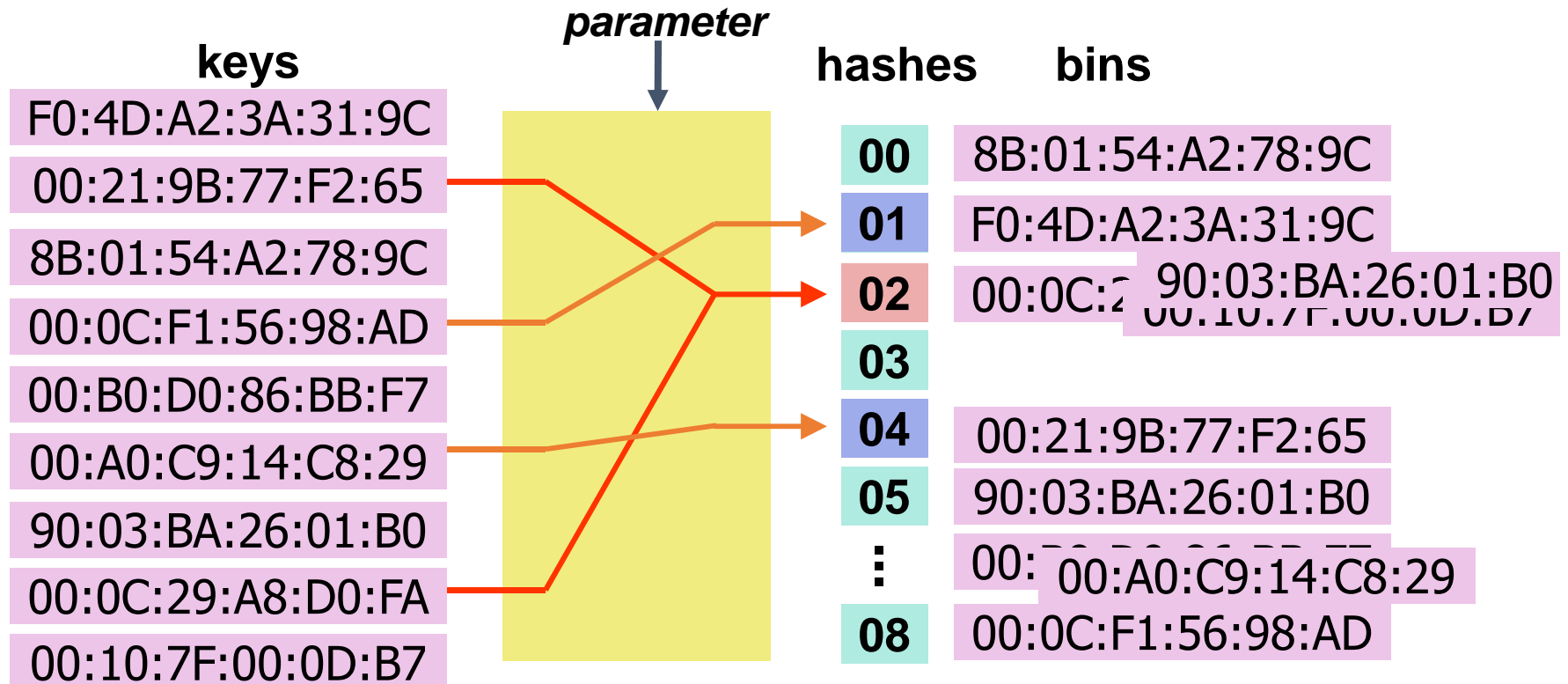
- Packets still have  $O(\log n)$  delay, but can process  $O(\log n)$  packets in parallel  $\rightarrow O(1)$

# Idea #2: hashing



- Hash key=destination, value=interface pairs
- Lookup in  $O(1)$  with hash
- Problem: chaining (not really  $O(1)$ )

# Improvement: Perfect hashing



- **Perfect hashing**: find a hash function that maps perfectly with no collisions
- Gigaswitch approach
  - Use a parameterized hash function
  - Precompute hash function to bound worst case number of collisions



# Variable-Length Matching Algorithms

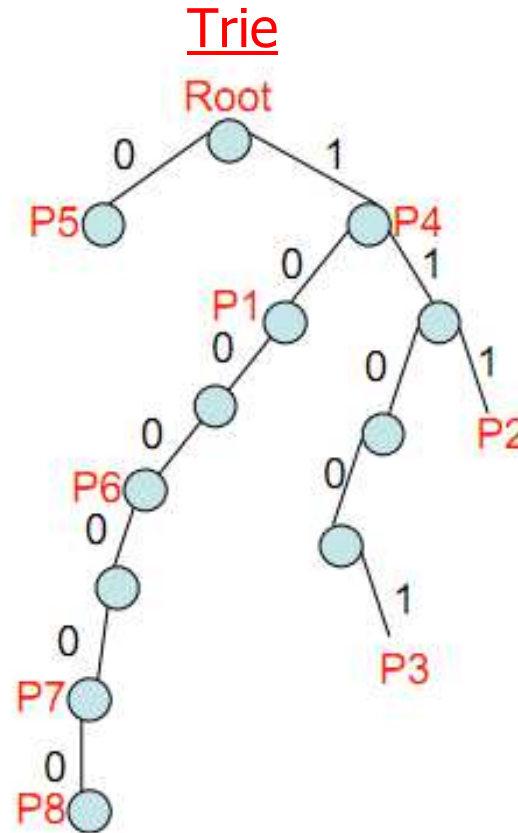
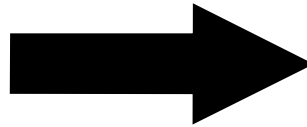
# Longest Prefix Match

- Not just one entry that matches a destination
  - 128.174.252.0/24 and 128.174.0.0/16
  - Which one to use for 128.174.252.14?
  - By convention, Internet routers choose the longest (most-specific) match
- Need variable prefix match algorithms
  - Several methods

# Method 1: Trie

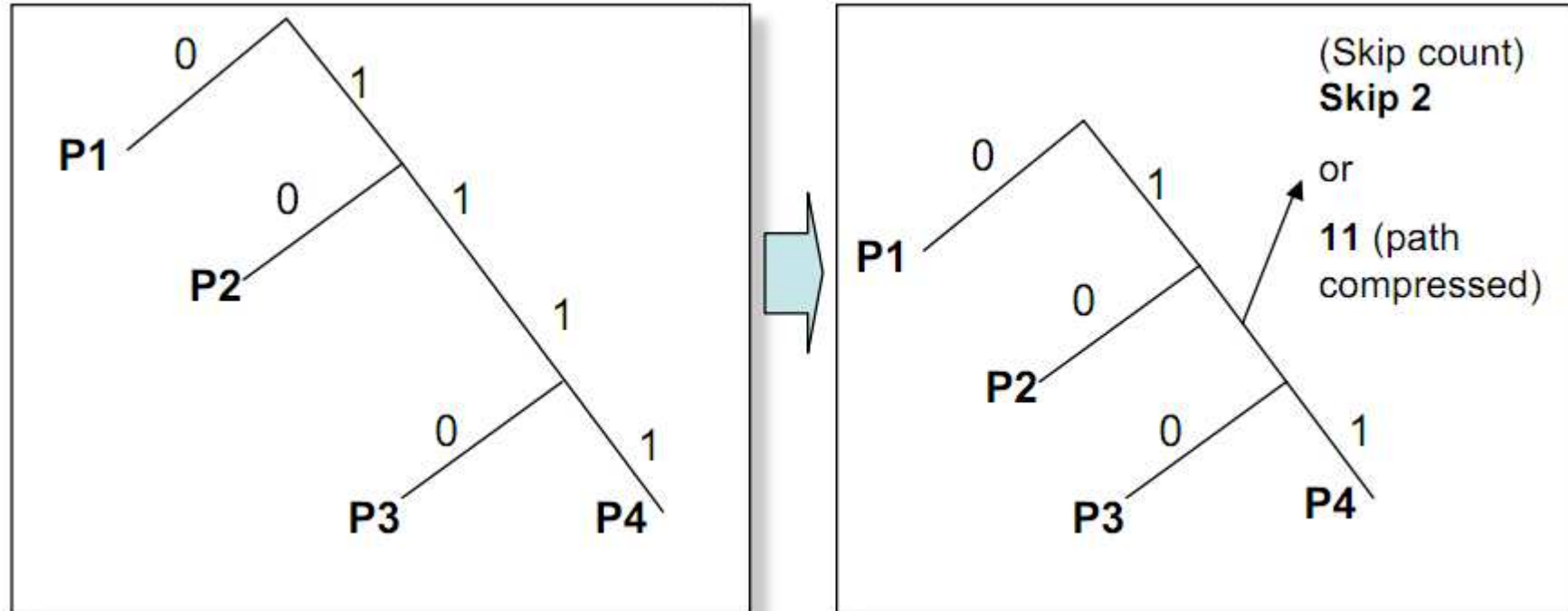
## Sample Database

- P1=10\*
- P2=111\*
- P3=11001\*
- P4=1\*
- P5=0\*
- P6=1000\*
- P7=100000\*
- P8=1000000\*



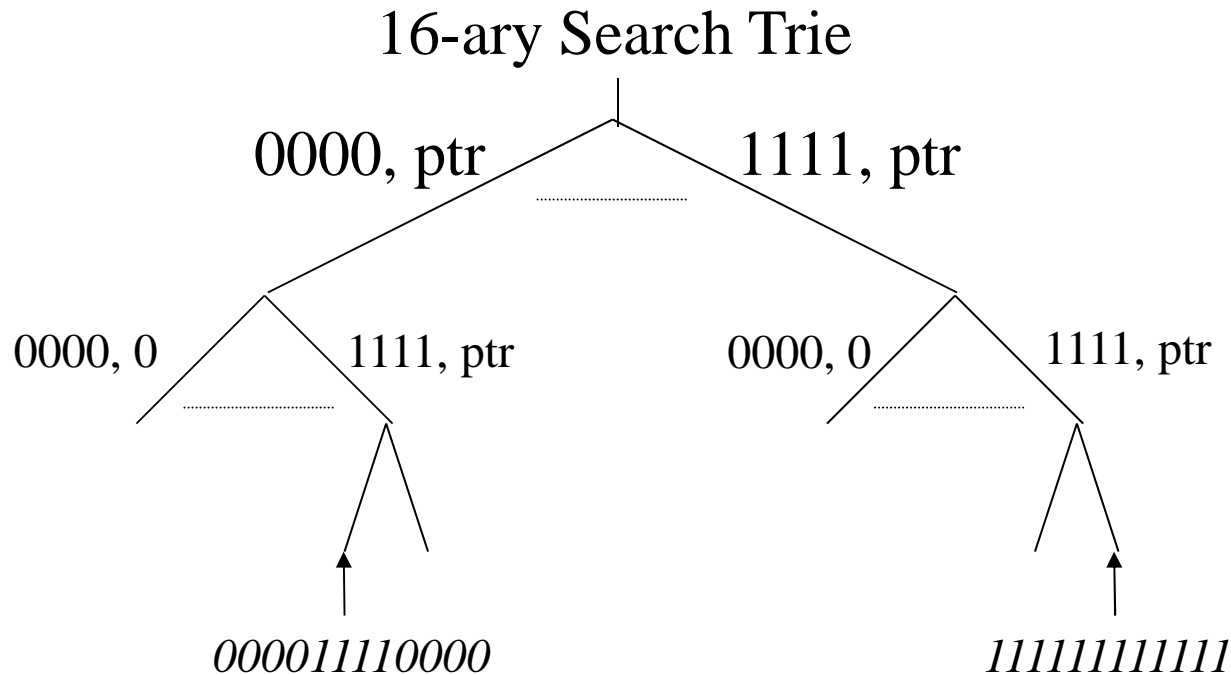
- Tree of (left ptr, right ptr) data structures
- May be stored in SRAM/DRAM
- Lookup performed by traversing sequence of pointers
- Lookup time  $O(\log N)$  where  $N$  is # prefixes

# Improvement 1: Skip Counts and Path Compression



- Removing one-way branches ensures # of trie nodes is at most twice # of prefixes
- Using a skip count requires exact match at end and backtracking on failure → path compression is simpler
- Main idea behind Patricia Tries

# Improvement 2: Multi-way tree



- Doing multiple comparisons per cycle accelerates lookup
  - Can do this for free to the width of CPU word (modern CPUs process multiple bits per cycle)
- But increases wasted space (more unused pointers)

# Improvement 2: Multi-way tree

$$E_w = D^{L-1} \left( 1 - \left( 1 - \frac{N}{D^L} \right)^D \right) + \sum_{i=1}^{L-1} D^i \left( (1 - D^{i-1})^N - (1 - D^{1-i})^N \right)$$

$$E_n = 1 + D^L \left( 1 - \frac{N}{D^L} \right)^D + \sum_{i=1}^{L-1} D^i - D^{i-1} (1 - D^{i-1})^N$$

Where:

D = Degree of tree

L = Number of layers/references

N = Number of entries in table

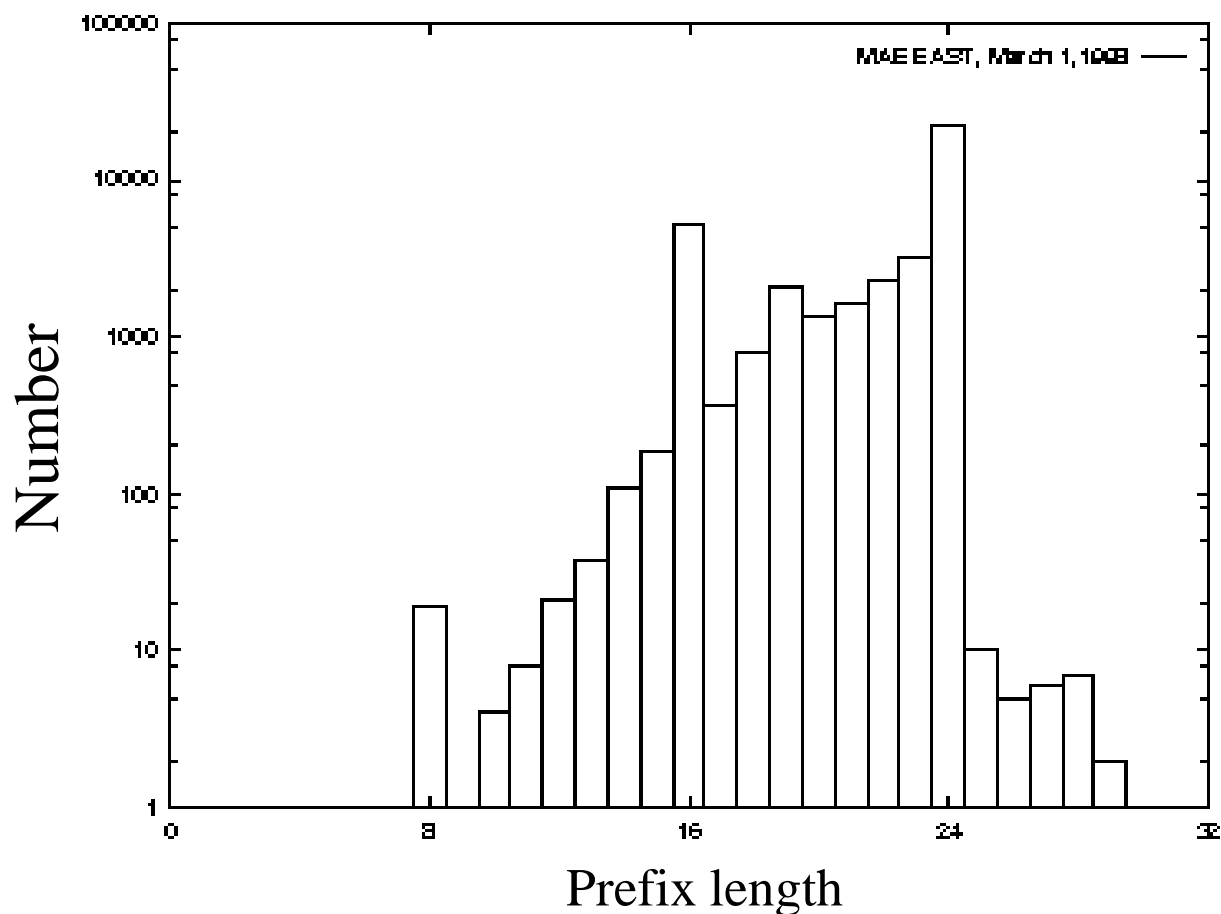
$E_n$  = Expected number of nodes

$E_w$  = Expected amount of wasted memory

<i>Degree of Tree</i>	<i># Mem References</i>	<i># Nodes (x10<sup>6</sup>)</i>	<i>Total Memory (Mbytes)</i>	<i>Fraction Wasted (%)</i>
2	48	1.09	4.3	49
4	24	0.53	4.3	73
8	16	0.35	5.6	86
16	12	0.25	8.3	93
64	8	0.17	21	98
256	6	0.12	64	99.5

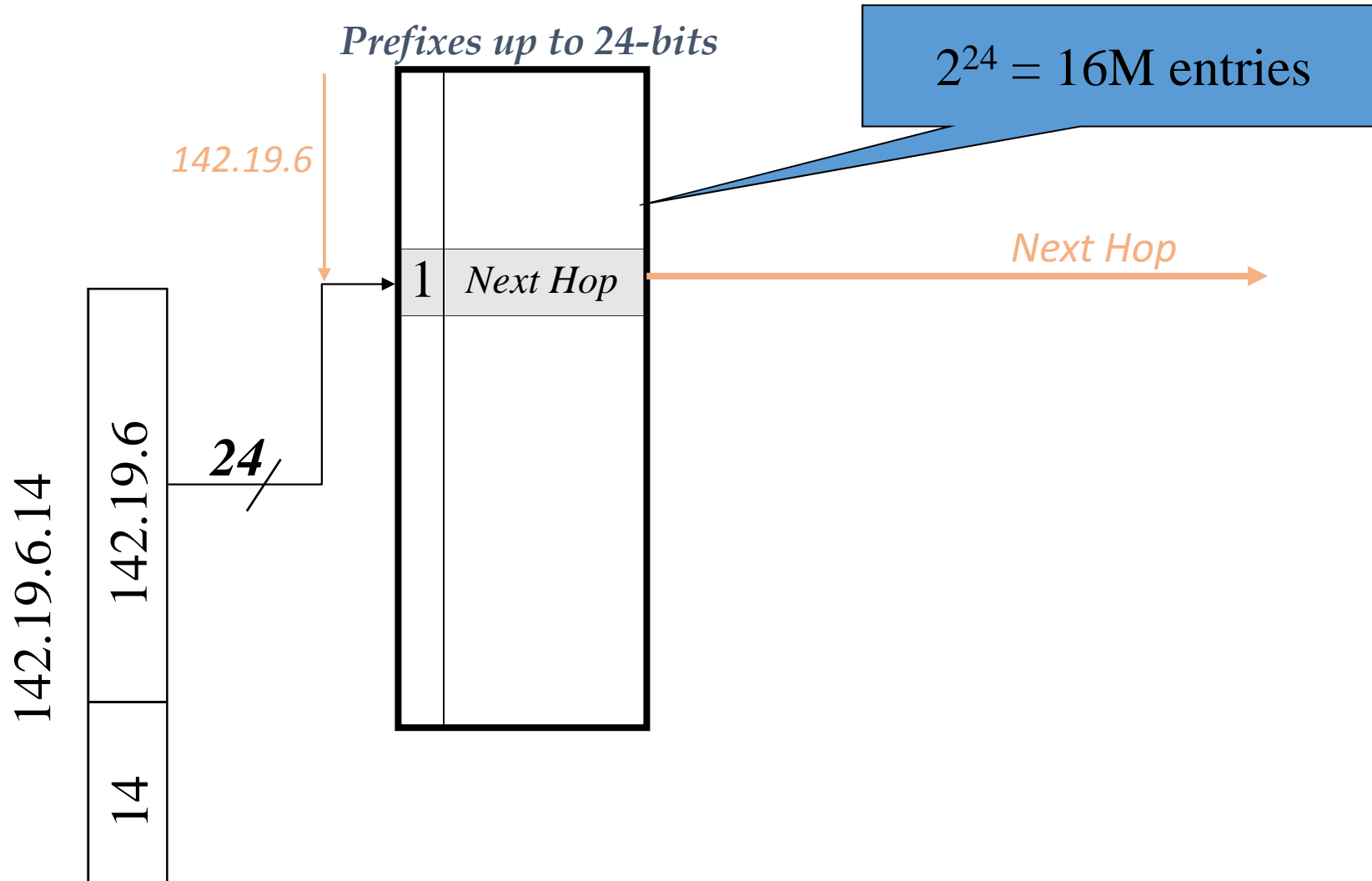
*Table produced from 2<sup>15</sup> randomly generated 48-bit addresses*

# Method 2: Use a Really Big Array



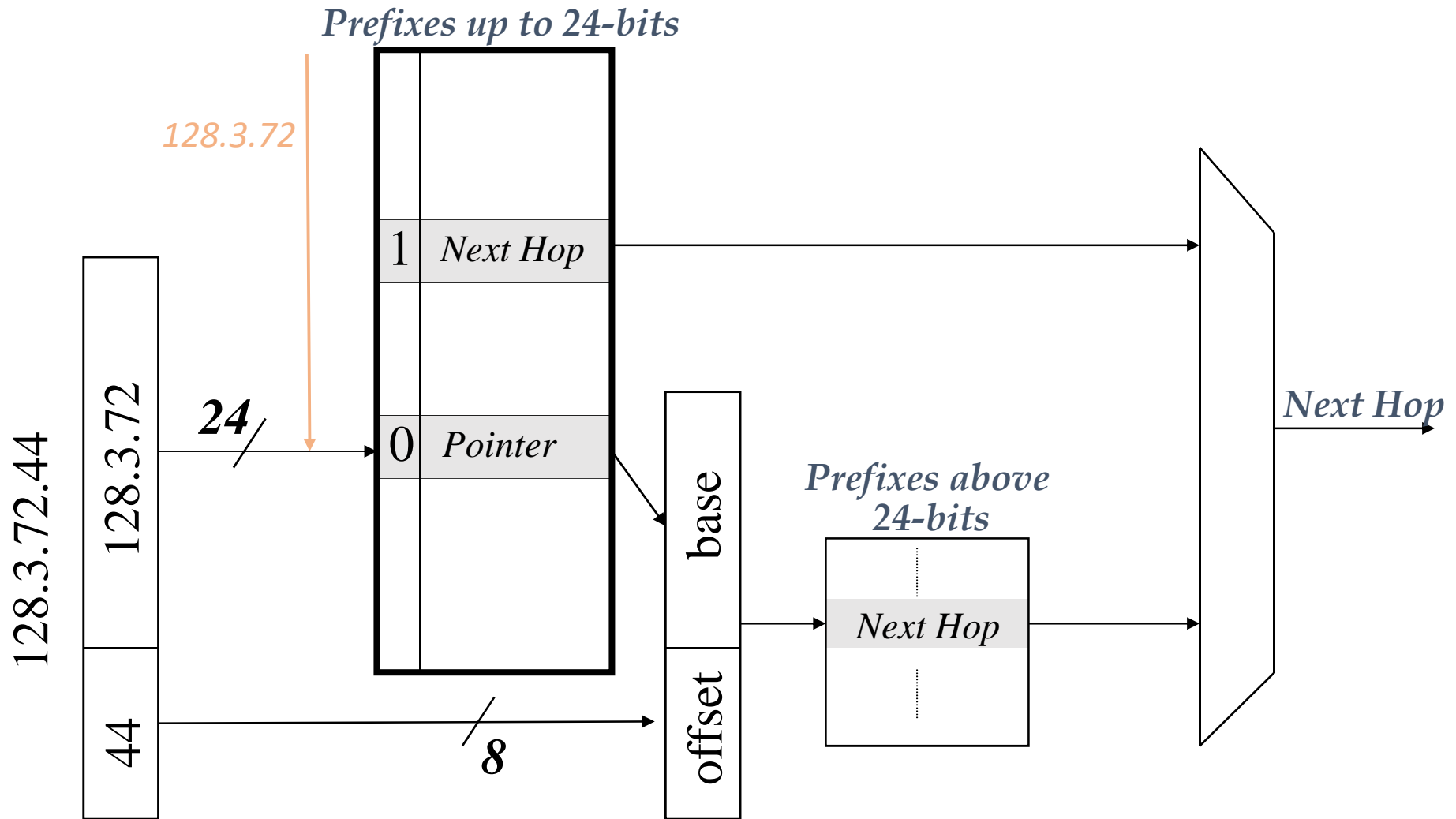
- Observation: most prefixes are /24 or shorter
- So, just store a big  $2^{24}$  table with next hop for each prefix
- Nonexistent prefixes → just leave that entry empty

# Method 2: Use a Really Big Array





# Method 2: Use a Really Big Array



# Method 2: Use a Really Big Array

- Advantages

- Very fast lookups
  - 20 Mpps with 50ns DRAM
- Easy to implement in hardware

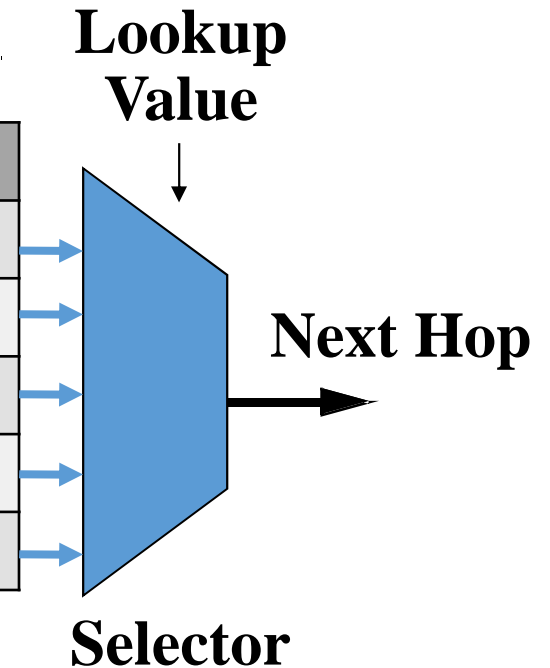
- Disadvantages

- Large memory required
- Performance depends on prefix length distribution

# Method 3: Ternary CAMs

## *Associative Memory*

Value	Mask	Next hop
10.0.0.0	255.0.0.0	IF 1
10.1.0.0	255.255.0.0	IF 3
10.1.1.0	255.255.255.0	IF 4
10.1.3.0	255.255.255.0	IF 2
10.1.3.1	255.255.255.255	IF 2



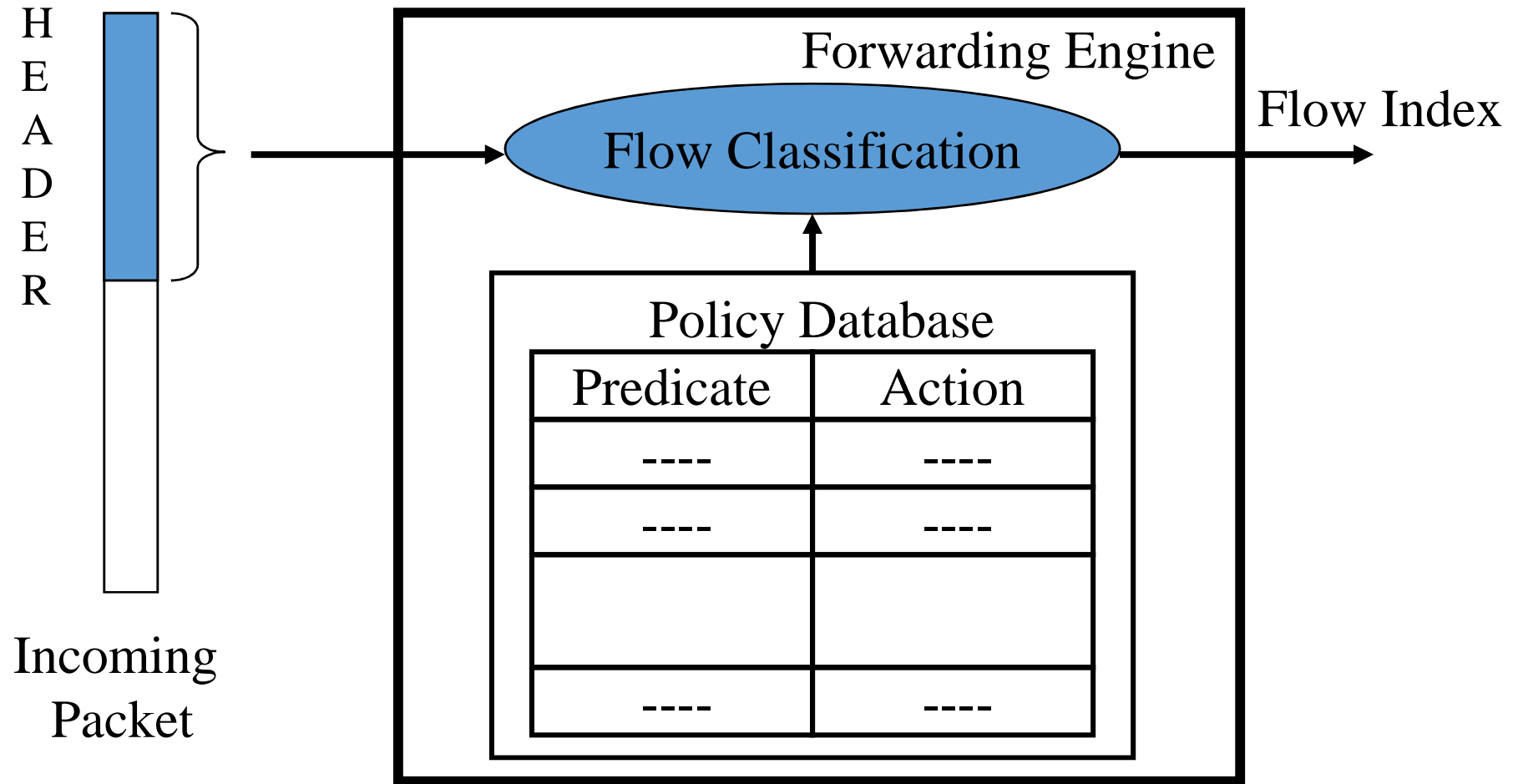
- “Content Addressable”
  - Hardware searches entire memory to find supplied value
  - Similar interface to hash table
- “Ternary”: memory can be in three states
  - True, false, don’t care
  - Hardware to treat don’t care as wildcard match

# Classification Algorithms

# Providing Value-Added Services

- Differentiated services
  - Regard traffic from AS#33 as `platinumgrade`
- Access Control Lists
  - Deny udp host 194.72.72.33 194.72.6.64 0.0.0.15 eq snmp
- Committed Access Rate
  - Rate limit WWW traffic from subinterface#739 to 10Mbps
- Policybased Routing
  - Route all voice traffic through the ATM network
- Peering Arrangements
  - Restrict the total amount of traffic of precedence 7 from
  - MAC address N to 20 Mbps between 10 am and 5pm
- Accounting and Billing
  - Generate hourly reports of traffic from MAC address M
- → Need to address the **Flow Classification** problem

# Flow Classification

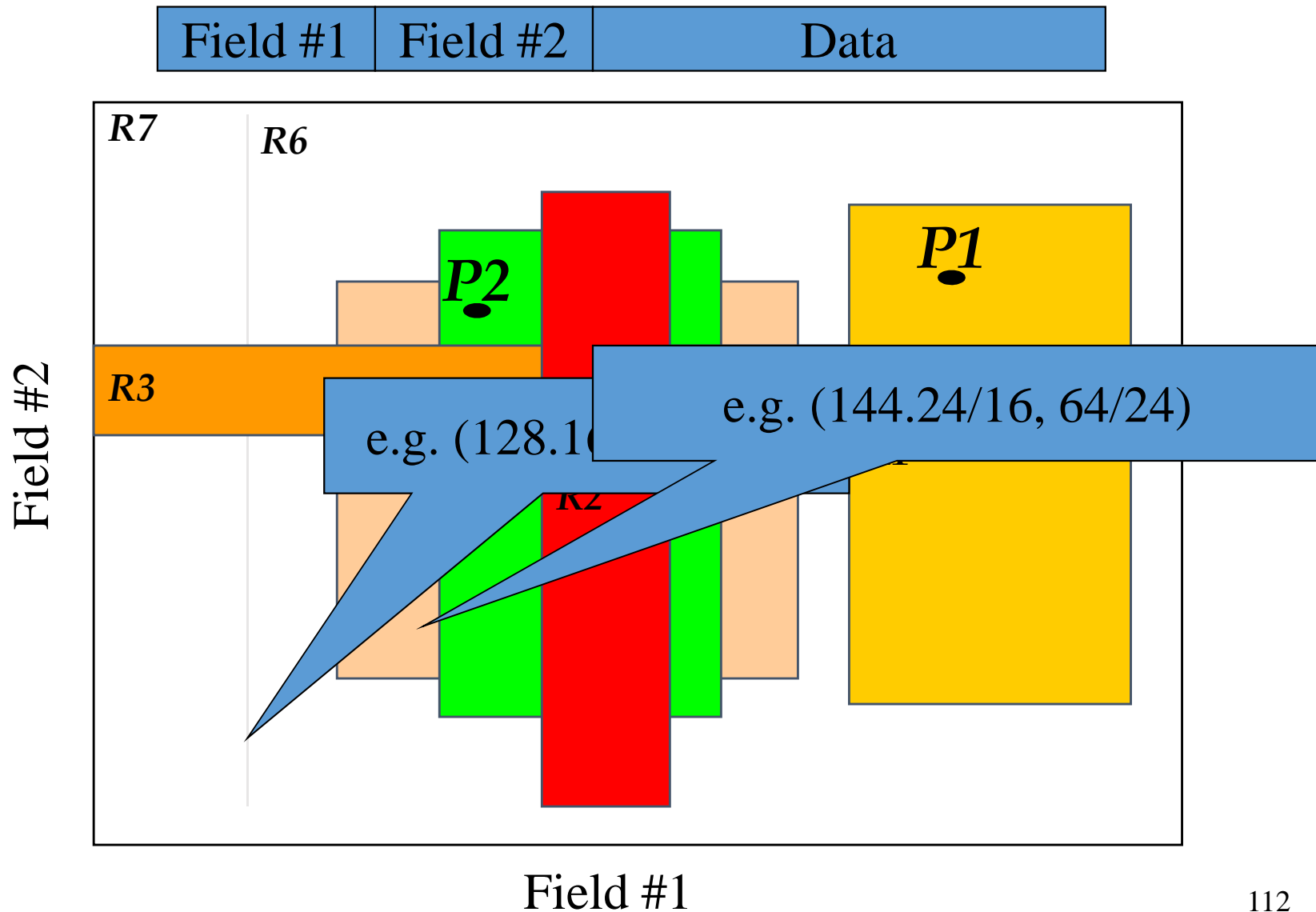


# A Packet Classifier

	<b>Field 1</b>	<b>Field 2</b>	<b>...</b>	<b>Field k</b>	<b>Action</b>
<b>Rule 1</b>	152.163.190.69/21	152.163.80.11/32	<b>...</b>	Udp	A1
<b>Rule 2</b>	152.168.3.0/24	152.163.200.157/16	<b>...</b>	Tcp	A2
<b>...</b>	<b>...</b>	<b>...</b>	<b>...</b>	<b>...</b>	<b>...</b>
<b>Rule N</b>	152.168.3.0/16	152.163.80.11/32	<b>...</b>	<b>Any</b>	<b>An</b>

*Given a classifier, find the action associated with the highest priority rule (here, the lowest numbered rule) matching an incoming packet.*

# Geometric Interpretation in 2D





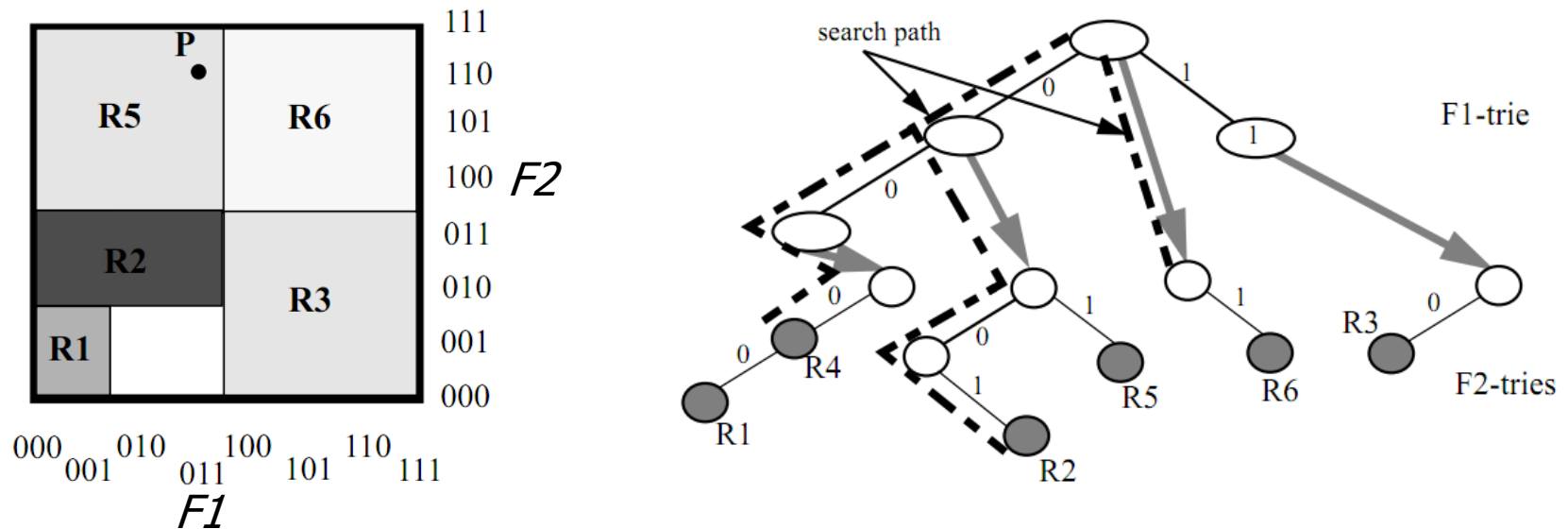
# Approach #1: Linear search

- Build linked list of all classification rules
  - Possibly sorted in order of decreasing priorities
- For each arriving packet, evaluate each rule until match is found
- Pros: simple and storage efficient
- Cons: classification time grows linearly with number of rules
  - Variant: build FSM of rules (pattern matching)

# Approach #2: Ternary CAMs

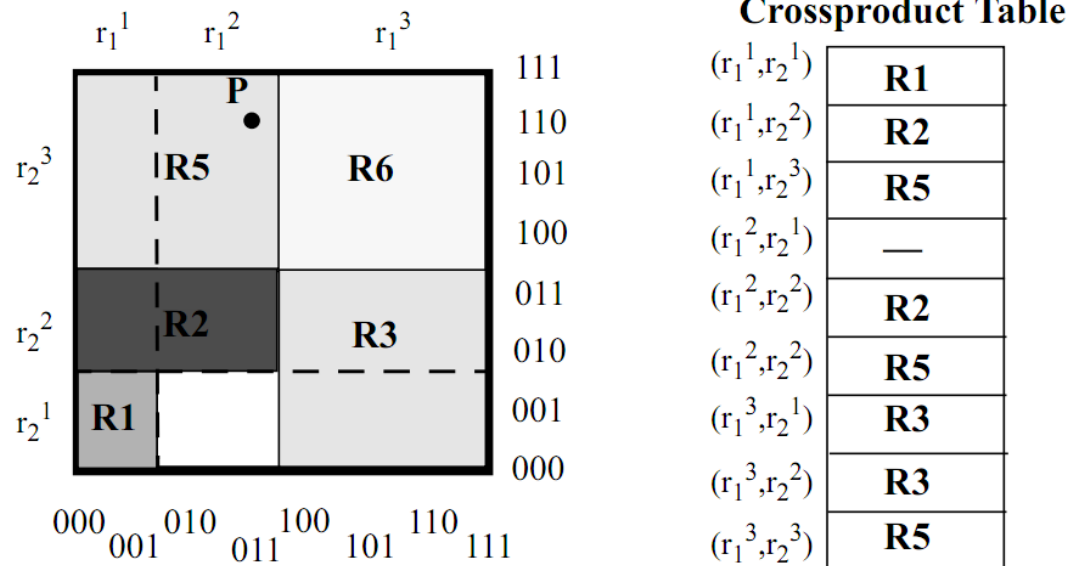
- Similar to TCAM use in prefix matching
  - Need wider than 32-bit array, typically 128-256 bits
- Ranges expressed as don't cares below a particular bit
  - Done for each field
- Pros:  $O(1)$  lookup time, simple
- Cons: heat, power, cost, etc.
  - Power for a TCAM row increases proportionally to its width

# Approach #3: Hierarchical trie



- Recursively build d-dimensional radix trie
  - Trie for first field, attach sub-tries to trie's leaves for sub-field, repeat
- For N-bit rules, d dimensions, W-bit wide dimensions:
  - Storage complexity:  $O(NdW)$
  - Lookup complexity:  $O(W^d)$

# Approach #4: Crossproducting



- Compute separate 1-dimensional range lookups for each dimension
- For N-bit rules, d dimensions, W-bit wide dimensions:
  - Storage complexity:  $O(N^d)$
  - Lookup complexity:  $O(dW)$



# Roadmap

- Midterm high-level questions
- Network Address Translation
- QoS scheduling
- Physical Media

## 1. High level questions (28 points)

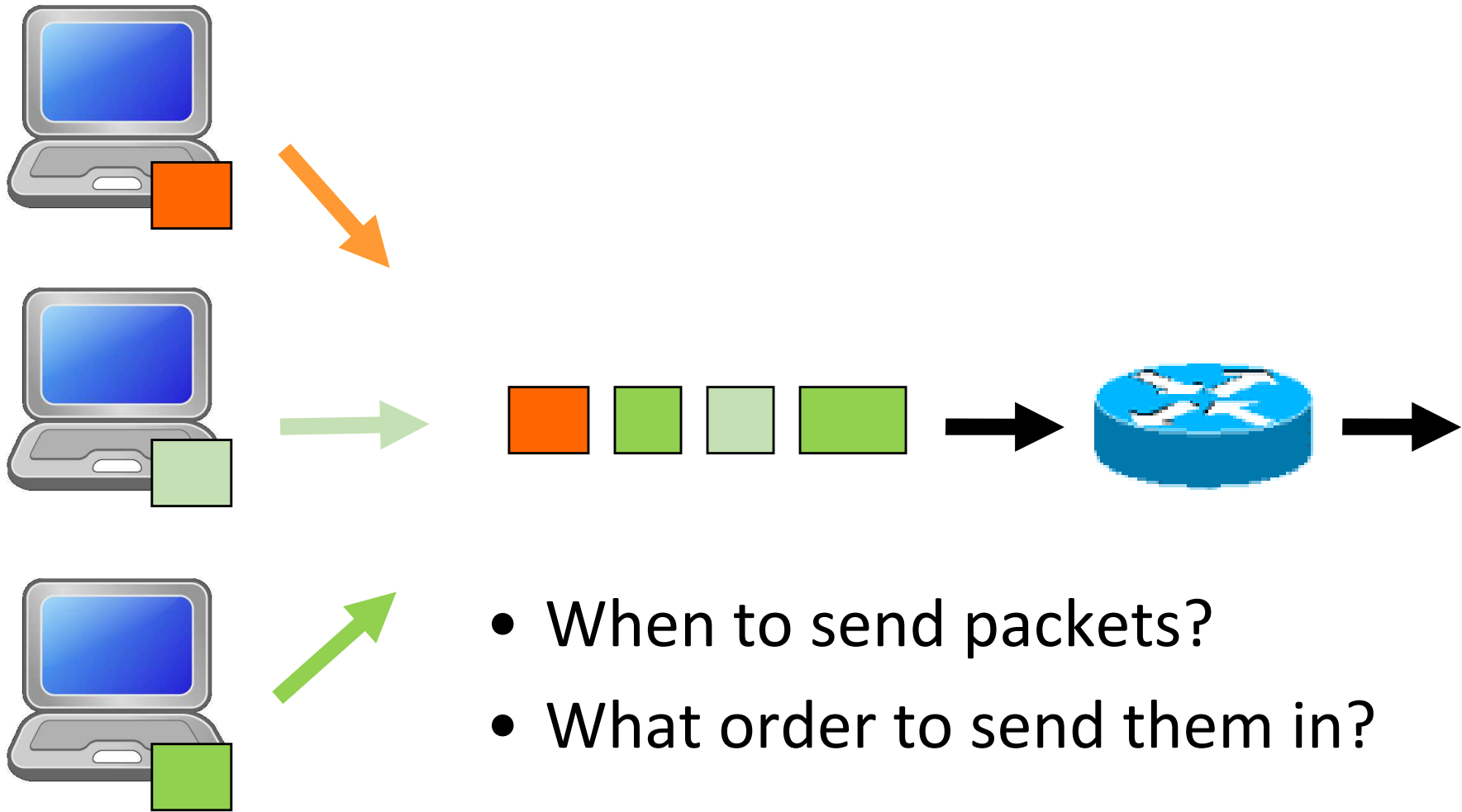
- a. You are the administrator of an Ethernet network in a university's engineering buildings. There was a sale in town on cheap hardware, where some crappy vendor cheated out and instead of assigning MAC addresses properly, used the same 40-bit prefix and just cycled through the lower bits. End result: you have just discovered that a number of newly deployed computers on your network have duplicate addresses.
  - i. Briefly describe what problems this might cause.
  - ii. Give a temporary workaround you could use to avoid those problems.
- b. Suppose you ran an ISP network. What would be some reasons to disable traceroutes on your network?
- c. Currently hosts in the Internet have multiple "names": the IP address, the MAC address, and the DNS name. Imagine an "alternate reality" where the designers of the Internet had decided that each host should have only one name: the DNS name. The MAC address is replaced in the MAC header with the DNS name. IP routers now do lookups on DNS names, etc. Give at least one benefit and at least one downside to this design approach.
- d. Why are IP addresses assigned as a function of the topology? Are MAC addresses also assigned as a function of the topology -- if not, why not?
- e. Define "hub", "switch" and "router" -- what is the difference between these devices? Under what circumstances would you prefer a hub to a switch?
- f. Do ISPs typically configure export from providers to peers? Why or why not?
- g. Why is a consistent export clause important in service-level agreements?
- h. What is the BGP decision process? Why do operators wish to exercise control over this process? How exactly do operators influence the process to achieve their goals?

# Network address translation



# Packet Scheduling and Fair Queuing

# Packet Scheduling: Problem Overview

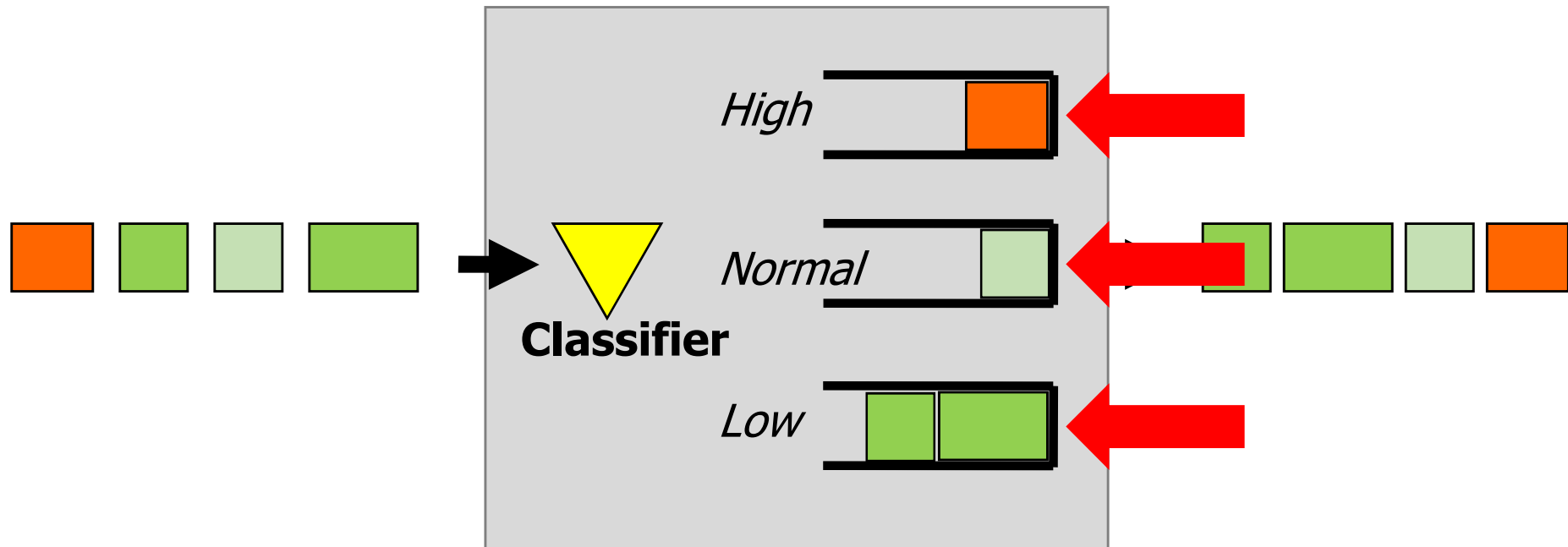


# Approach #1: First In First Out (FIFO)



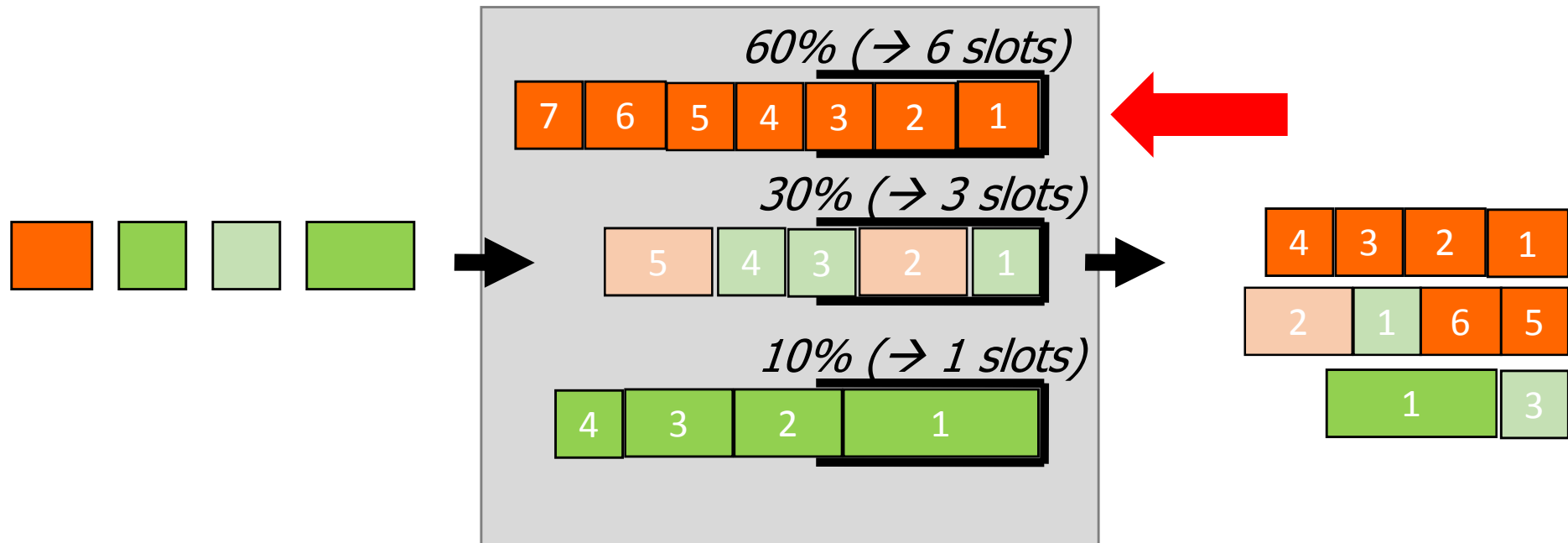
- Packets are sent out in the same order they are received
- Benefits: simple to design, analyze
- Downsides: not compatible with QoS
  - High priority packets can get stuck behind low priority packets

# Approach #2: Priority Queuing



- Operator can configure policies to give certain kinds of packets higher priority
  - Associate packets with priority queues
  - Service higher-priority queue when packets are available to be sent
- Downside:
  - Can lead to starvation of lower-priority queues

# Approach #3: Weighted Round Robin

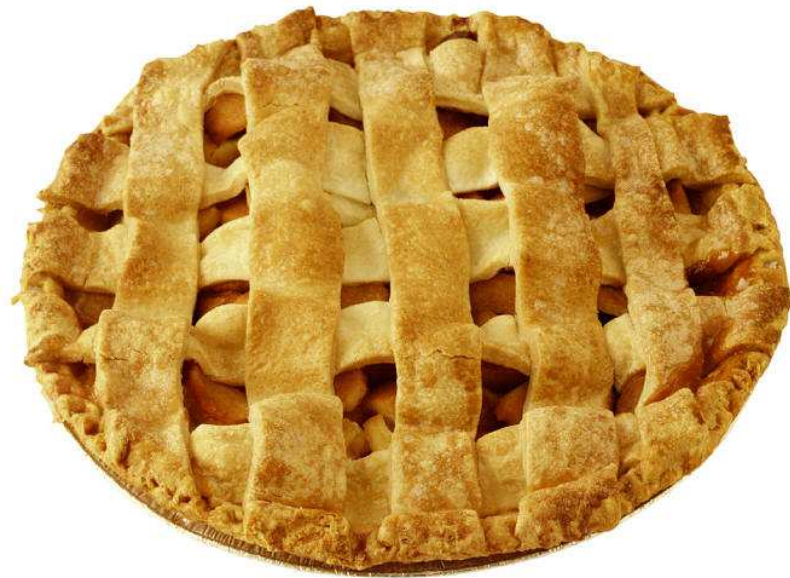


- Round robin through queues, but visit higher-priority queues more often
- Benefit: Prevents starvation
- Downsides: a host sending long packets can steal bandwidth
  - Naïve implementation wastes bandwidth due to unused slots

# Overview

- Fairness
- Fair-queuing
- Core-stateless FQ
- Other FQ variants

What would be a fair allocation here?



# Fairness Goals

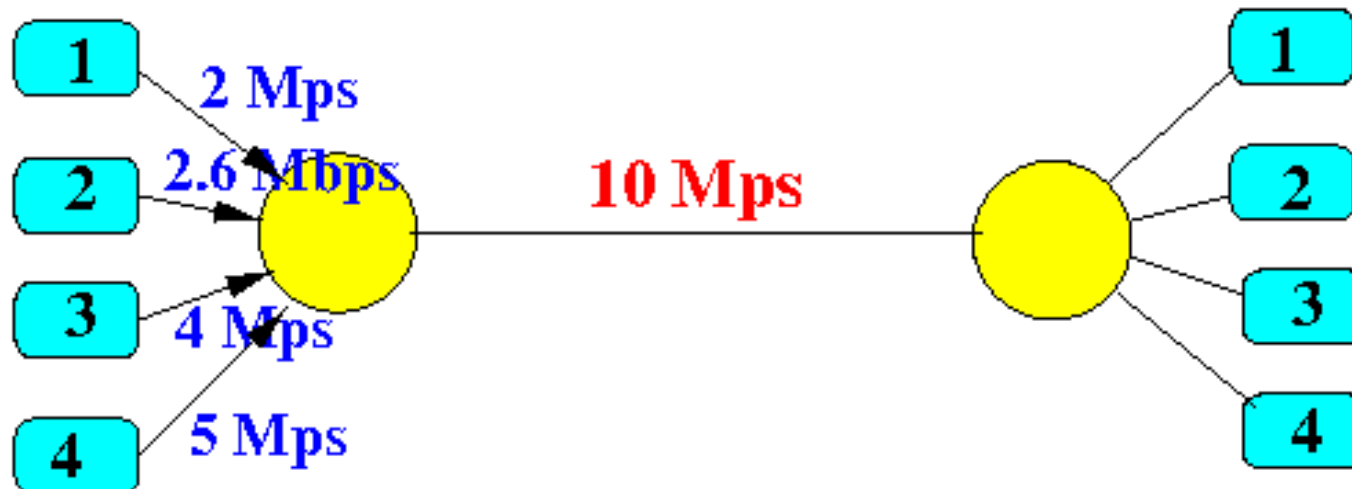
- Allocate resources fairly
- Isolate ill-behaved users
  - Router does not send explicit feedback to source
  - Still needs e2e congestion control
- Still achieve statistical muxing
  - One flow can fill entire pipe if no contenders
  - Work conserving → scheduler never idles link if it has a packet



# What is Fairness?

- At what granularity?
  - Flows, connections, domains?
- What if users have different RTTs/links/etc.
  - Should it share a link fairly or be TCP fair?
- Maximize fairness index?
  - Fairness =  $(\sum x_i)^2 / n(\sum x_i^2)$   $0 < \text{fairness} < 1$
- Basically a tough question to answer – typically design mechanisms instead of policy
  - User = arbitrary granularity

What would be a fair allocation here?



# Max-min Fairness

- Allocate user with “small” demand what it wants, evenly divide unused resources to “big” users
- Formally:
  - Resources allocated in terms of increasing demand
  - No source gets resource share larger than its demand
  - Sources with unsatisfied demands get equal share of resource

# Max-min Fairness Example

- Assume sources 1..n, with resource demands  $X_1..X_n$  in ascending order
- Assume channel capacity  $C$ .
  - Give  $C/n$  to  $X_1$ ; if this is more than  $X_1$  wants, divide excess  $(C/n - X_1)$  to other sources: each gets  $C/n + (C/n - X_1)/(n-1)$
  - If this is larger than what  $X_2$  wants, repeat process

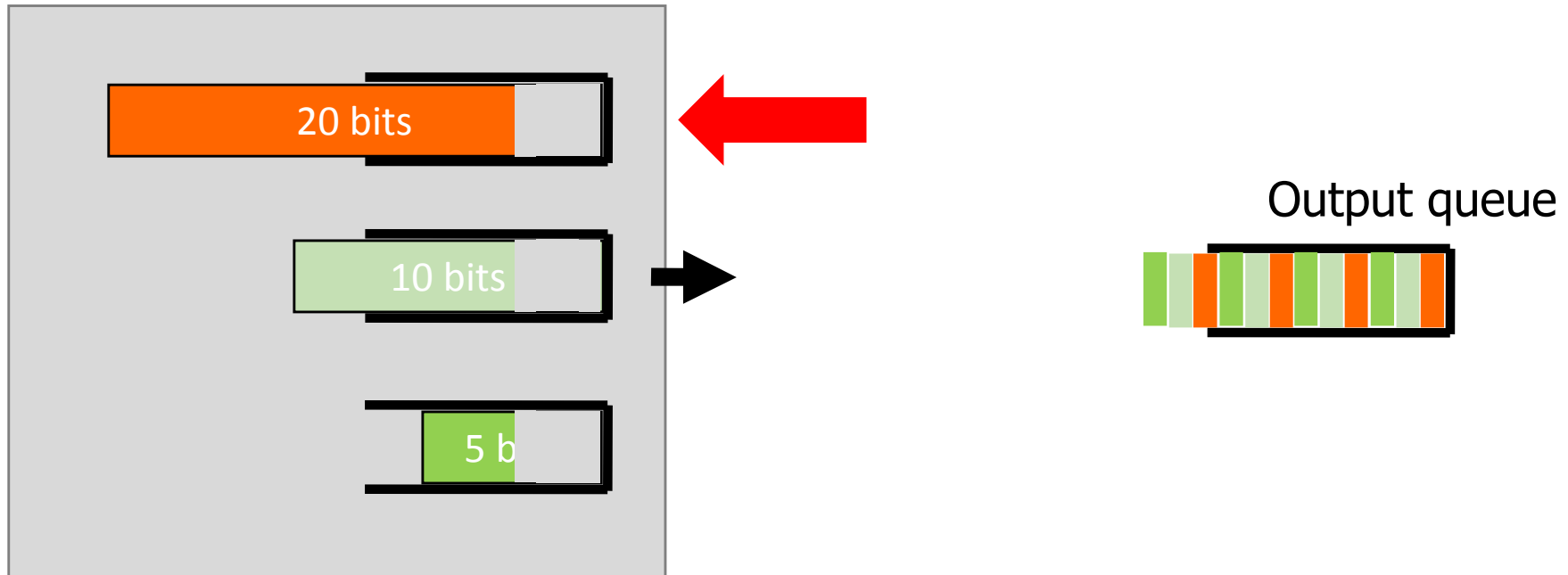
# Implementing max-min Fairness

- Generalized processor sharing
  - Fluid fairness
  - Bitwise round robin among all queues
- Why not simple round robin?
  - Variable packet length → can get more service by sending bigger packets
  - Unfair instantaneous service rate
    - What if arrive just before/after packet departs?

# Bit-by-bit RR

- Single flow: clock ticks when a bit is transmitted.  
For packet  $i$ :
  - $P_i$  = length,  $A_i$  = arrival time,  $S_i$  = begin transmit time,  $F_i$  = finish transmit time
  - $F_i = S_i + P_i = \max(F_{i-1}, A_i) + P_i$
- Multiple flows: clock ticks when a bit from all active flows is transmitted  $\rightarrow$  round number
  - Can calculate  $F_i$  for each packet if number of flows is known at all times
    - This can be complicated

# Approach #4: Bit-by-bit Round Robin



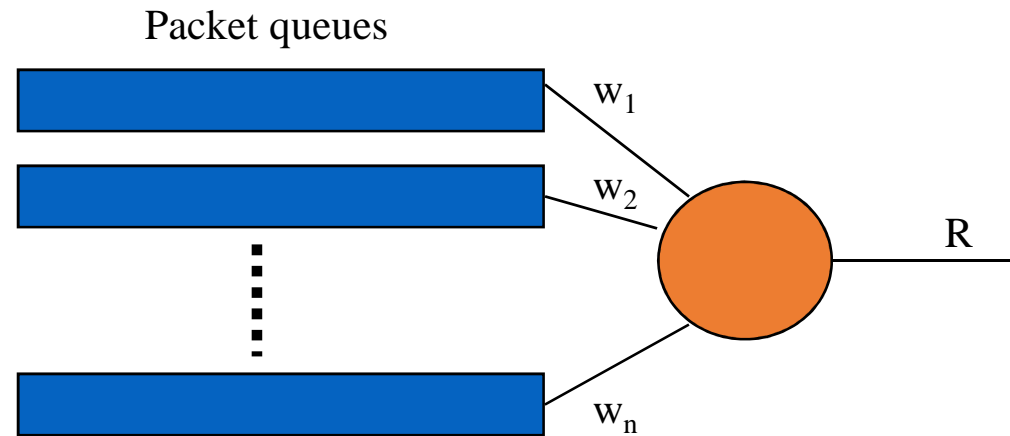
- Round robin through “backlogged” queues (queues with pkts to send)
  - However, only send one bit from each queue at a time
- Benefit:
  - Achieves max-min fairness, even in presence of variable sized pkts
- Downsides:
  - You can’t really mix up bits like this on real networks!

# The next-best thing: Fair Queuing

- Bit-by-bit round robin is fair, but you can't really do that in practice
- Idea: simulate bit-by-bit RR, compute the finish times of each packet
  - Then, send packets in order of finish times
  - This is known as Fair Queuing



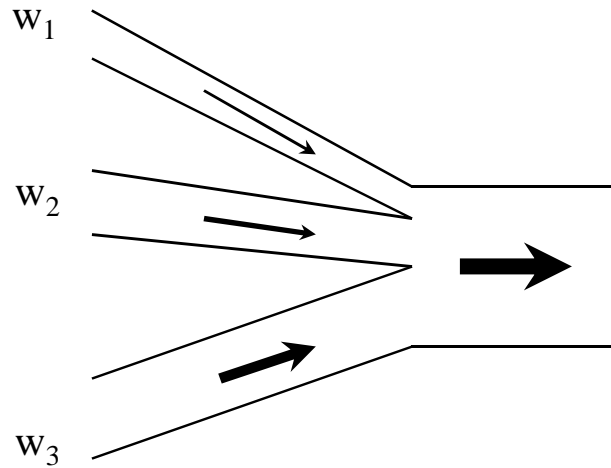
# What is Weighted Fair Queuing?



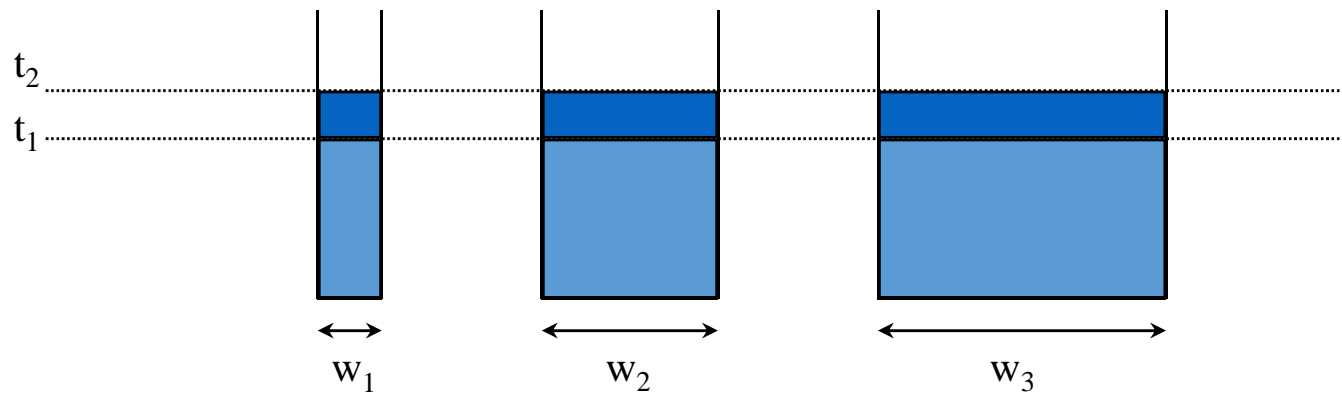
- Each flow  $i$  given a weight (importance)  $w_i$
- WFQ guarantees a minimum service rate to flow  $i$ 
  - $r_i = R * w_i / (w_1 + w_2 + \dots + w_n)$
  - Implies isolation among flows (one cannot mess up another)

# What is the Intuition? Fluid Flow

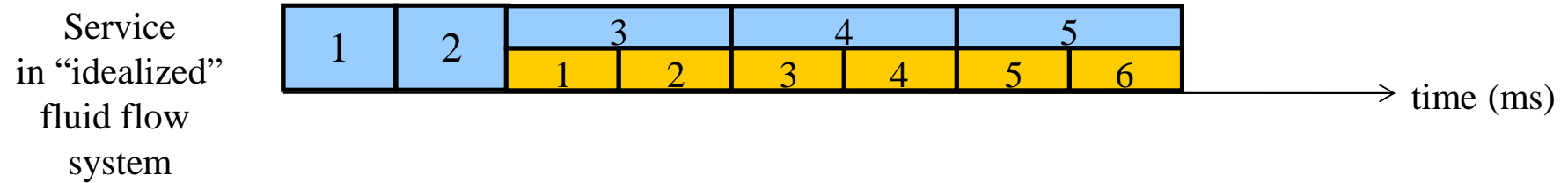
water pipes



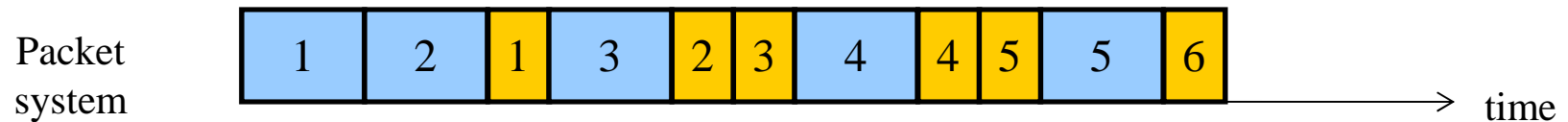
water buckets



# Example



- Practical implementation: send the first packet that finishes in the fluid flow system



# Properties of WFQ

- Guarantee that any packet is transmitted within *packet\_length/link\_capacity* of its transmission time in the fluid flow system
  - Can be used to provide guaranteed services
- Achieve fair allocation
  - Can be used to protect well-behaved flows against malicious flows

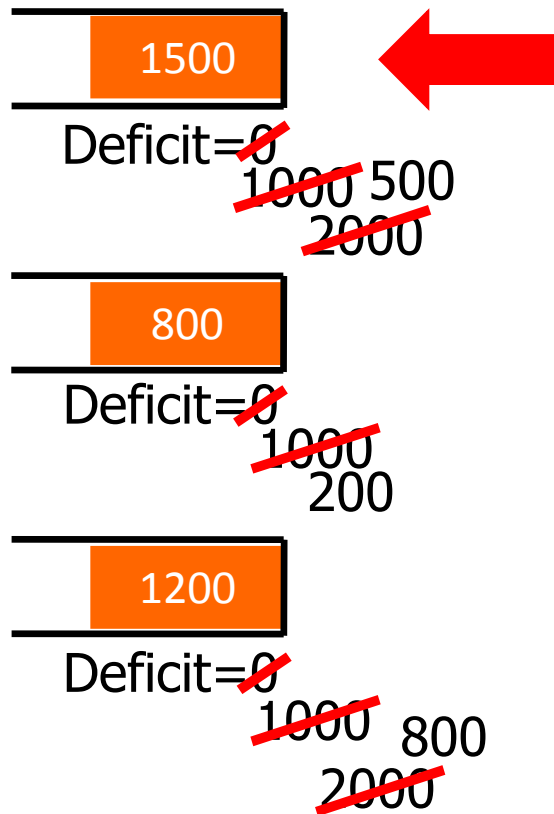
# Deficit Round Robin

- Each queue is allowed to send  $Q$  bytes per round
- If  $Q$  bytes are not sent (because packet is too large) deficit counter of queue keeps track of unused portion
- If queue is empty, deficit counter is reset to 0
- Uses hash bins like Stochastic FQ
- Similar behavior as FQ but computationally simpler
  - Bandwidth guarantees, but no latency guarantees

# Deficit Round Robin

## Example

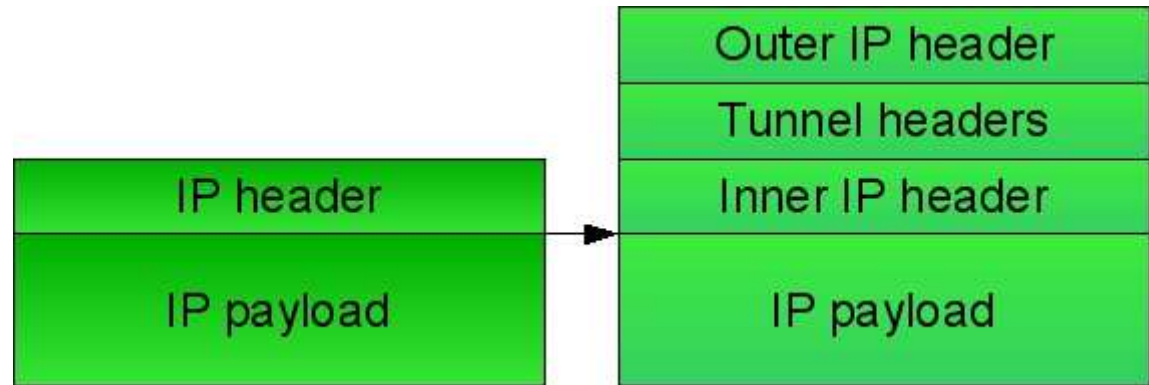
Quantum Size = 1000



1. Increment deficit counter by Quantum Size
2. Send packet if size is greater than deficit
3. When you send a packet, subtract its size from the deficit

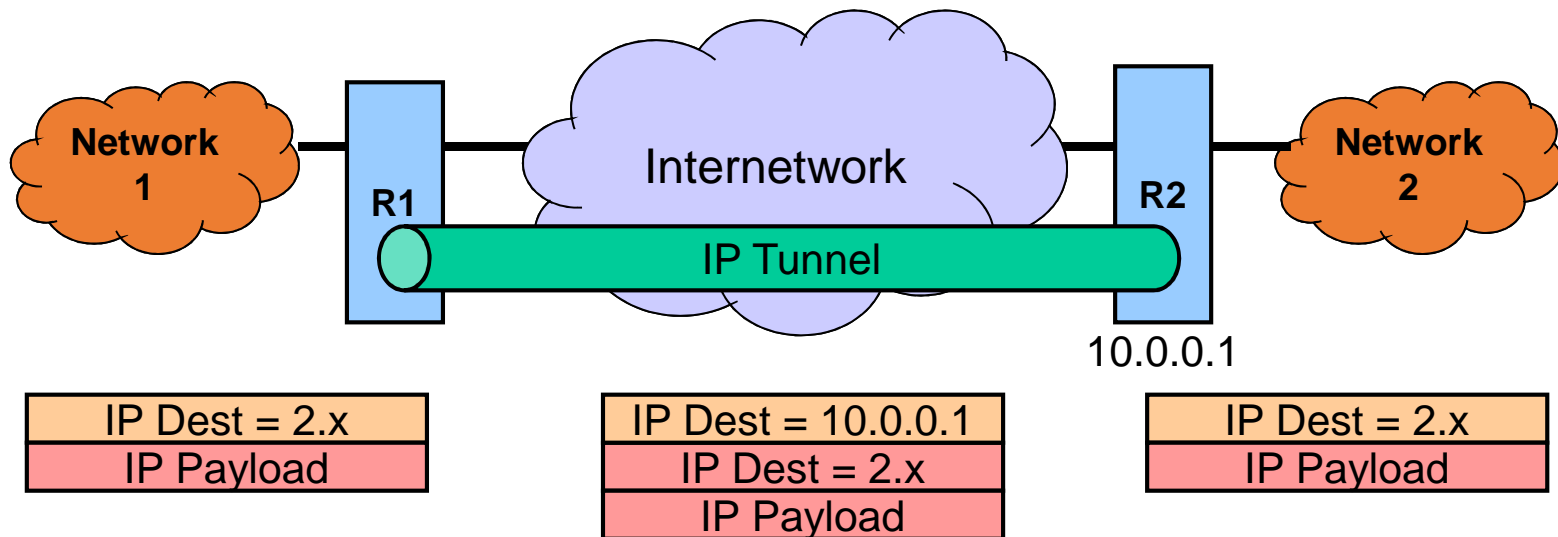


# Tunneling



- IP Tunnel

- Virtual point-to-point link between an arbitrarily connected pair of nodes

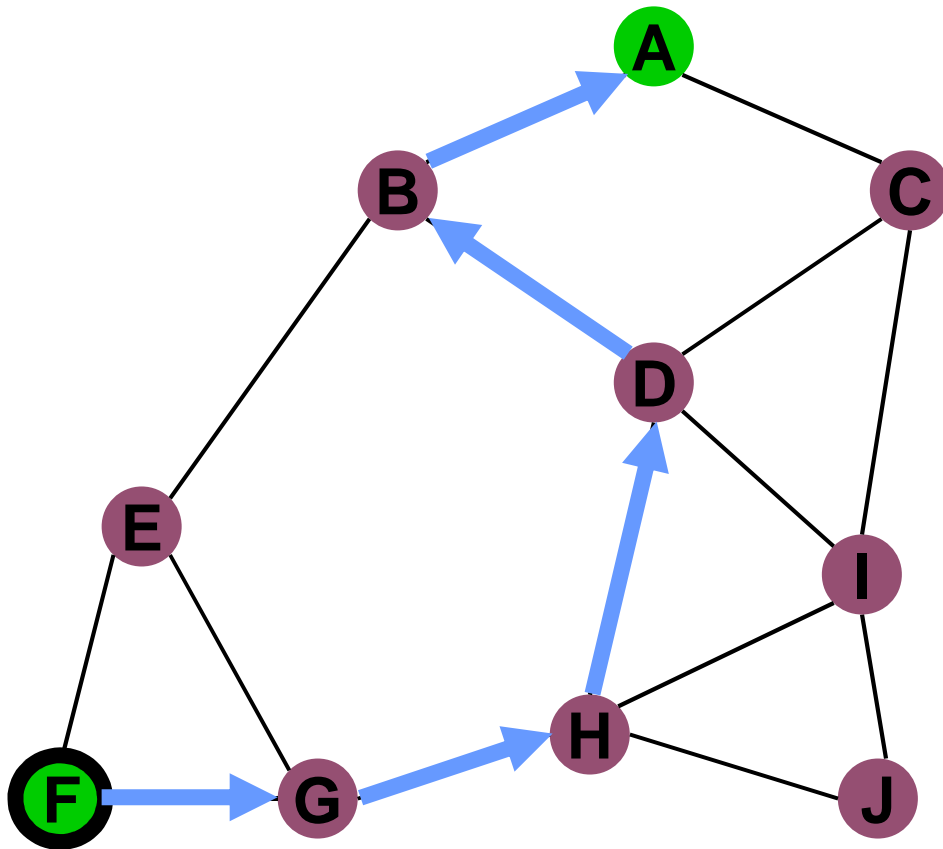


# Tunneling

- Approaches: GRE, MPLS, IPv6-in-IPv4 (6in4), L2TP, VPLS, VPNs, IPSec, SSH, HTTP, Covert channels (eg. ICMP), etc
- Advantages
  - Transparent transmission of packets over a heterogeneous network
  - Only need to change relevant routers
- Disadvantages
  - Increases packet size
  - Processing time needed to encapsulate and unencapsulate packets
  - Management at tunnel-aware routers

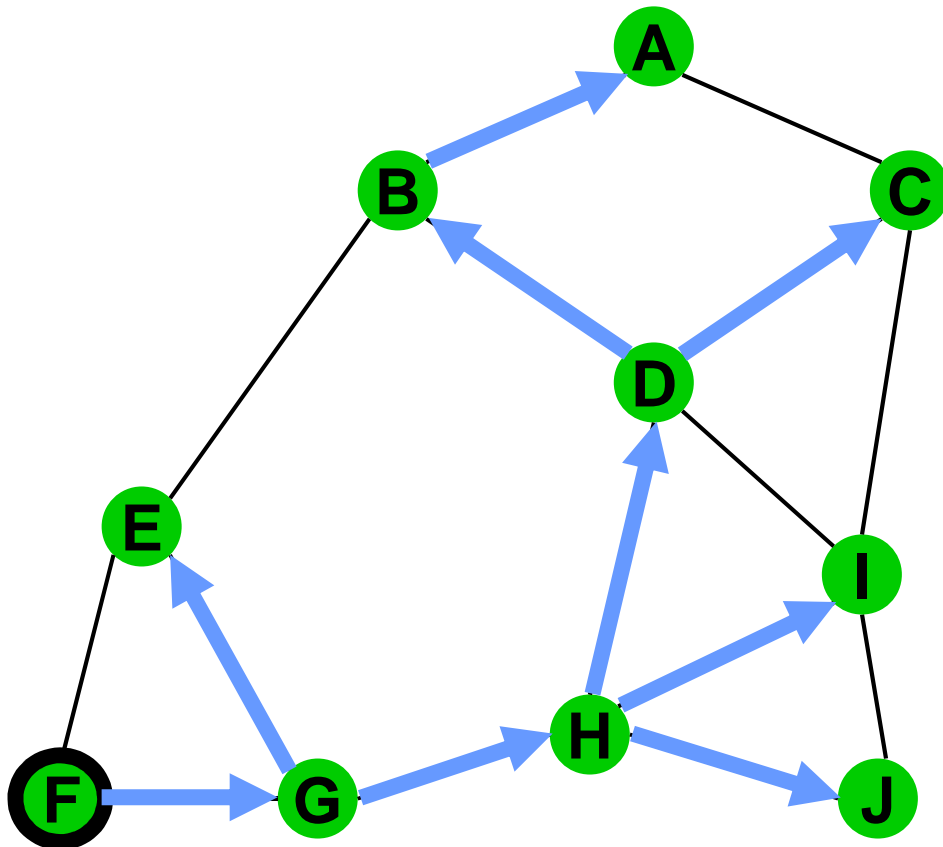


# Delivery models



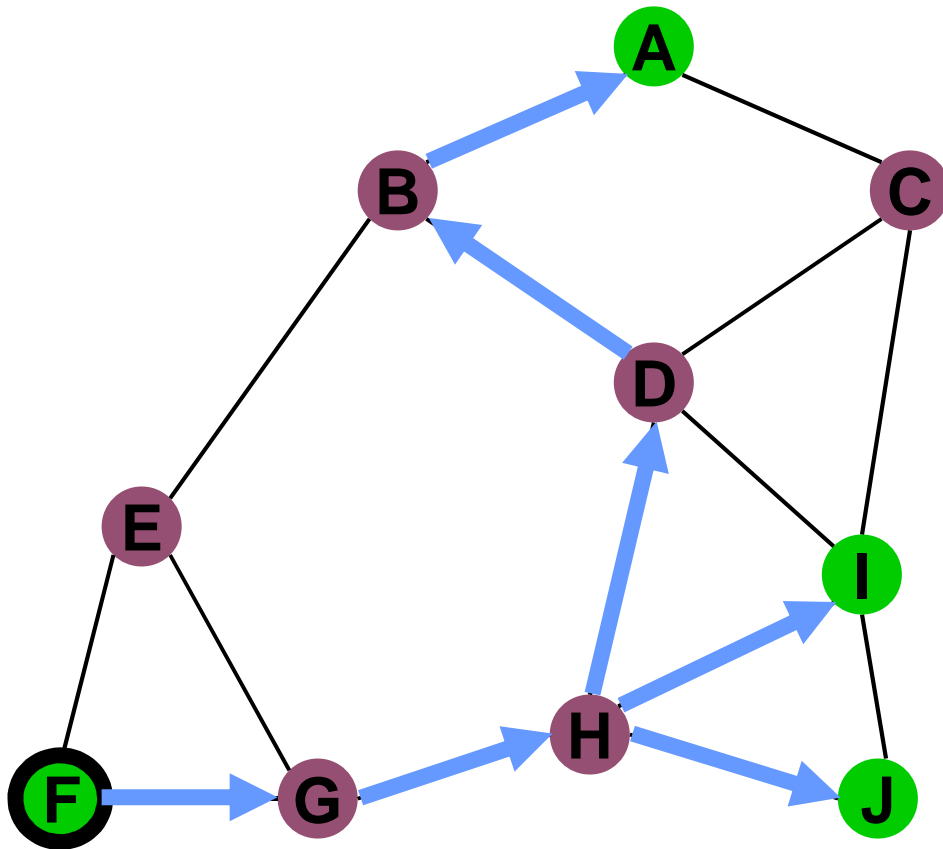
- Unicast
  - One source, one destination
  - Widely used (web, p2p, streaming, many other protocols)
- Broadcast
- Multicast
- Anycast

# Delivery models



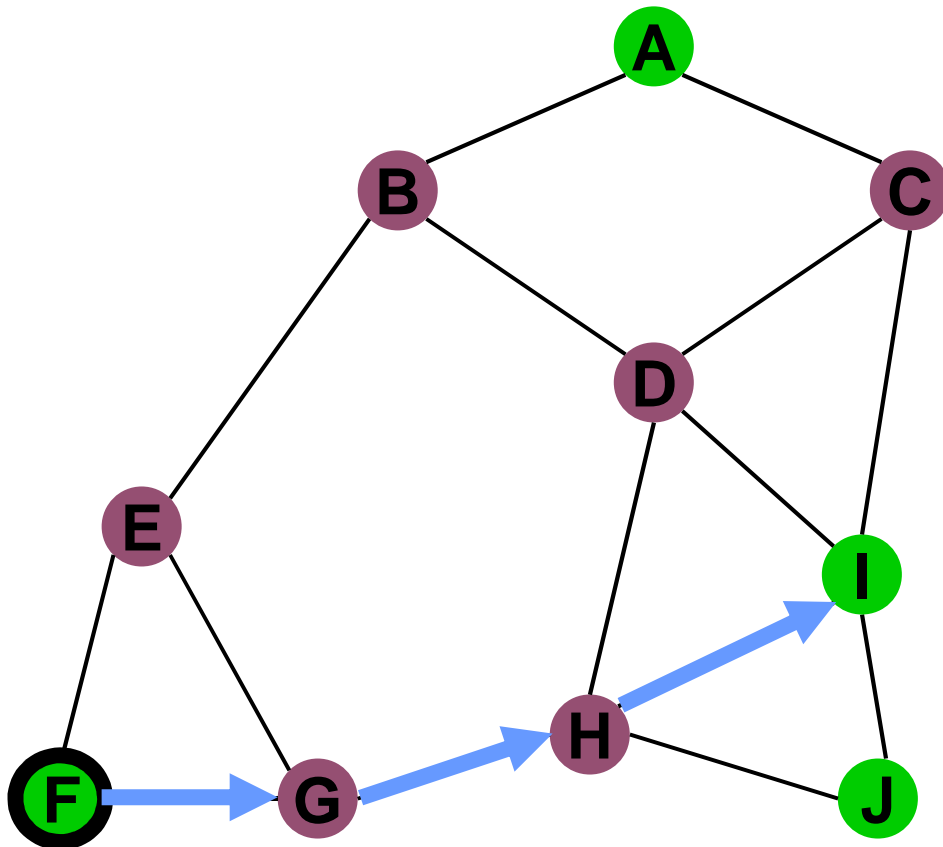
- Unicast
- Broadcast
  - One source, all destinations
  - Used to disseminate control information, perform service discovery
- Multicast
- Anycast

# Delivery models



- Unicast
- Broadcast
- Multicast
  - One source, several (prespecified) destinations
  - Used within some ISP infrastructures for content delivery, overlay networks
- Anycast

# Delivery models



- Unicast
- Broadcast
- Multicast
- Anycast
  - One source, route to “best” destination
  - Used in DNS, content distribution

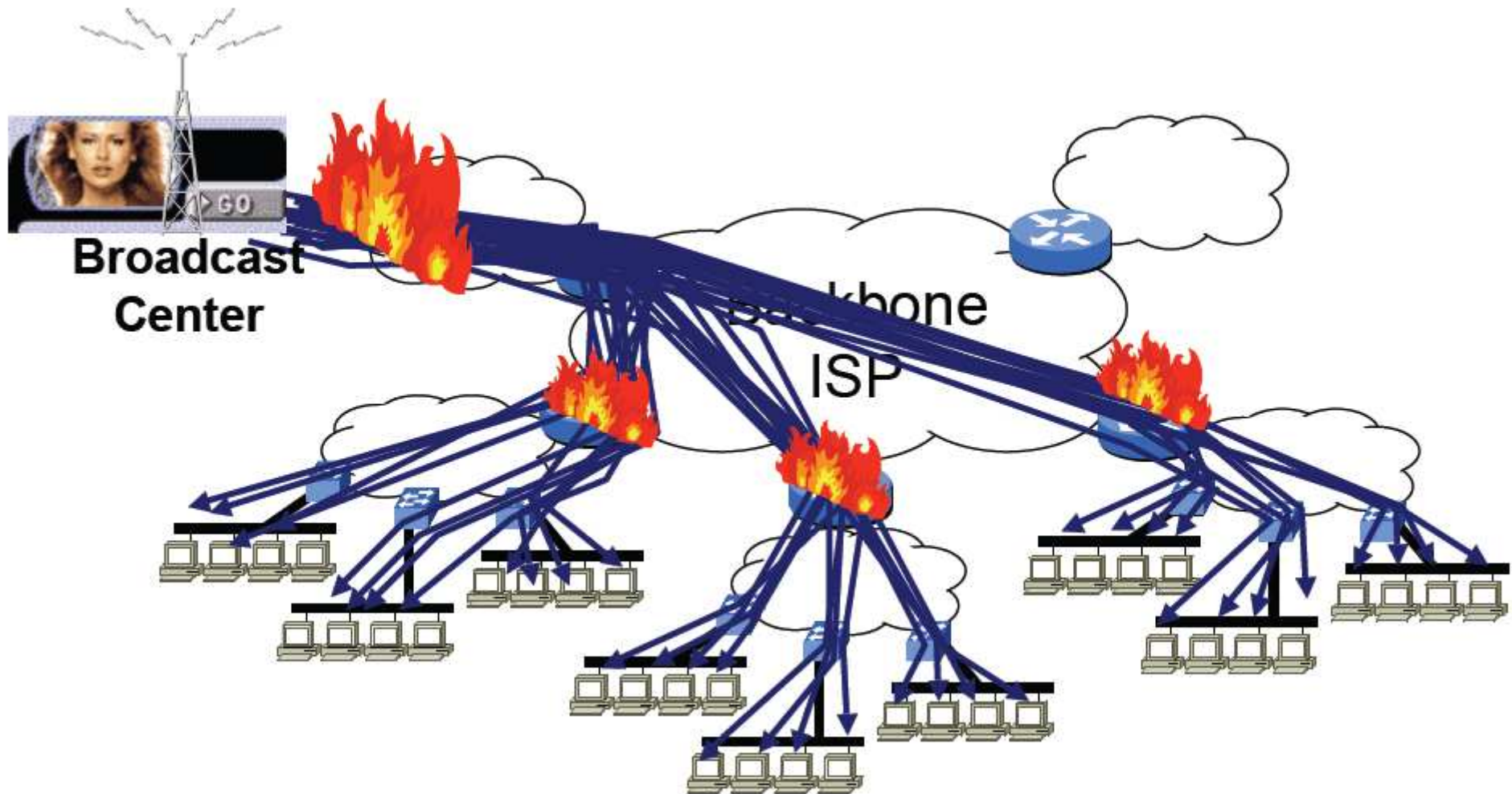
# Internet Multicast

- Motivation and challenges
- Support strategy
- IP multicast service model
- Multicast in the Internet
- Multicast routing protocols
- Limitations

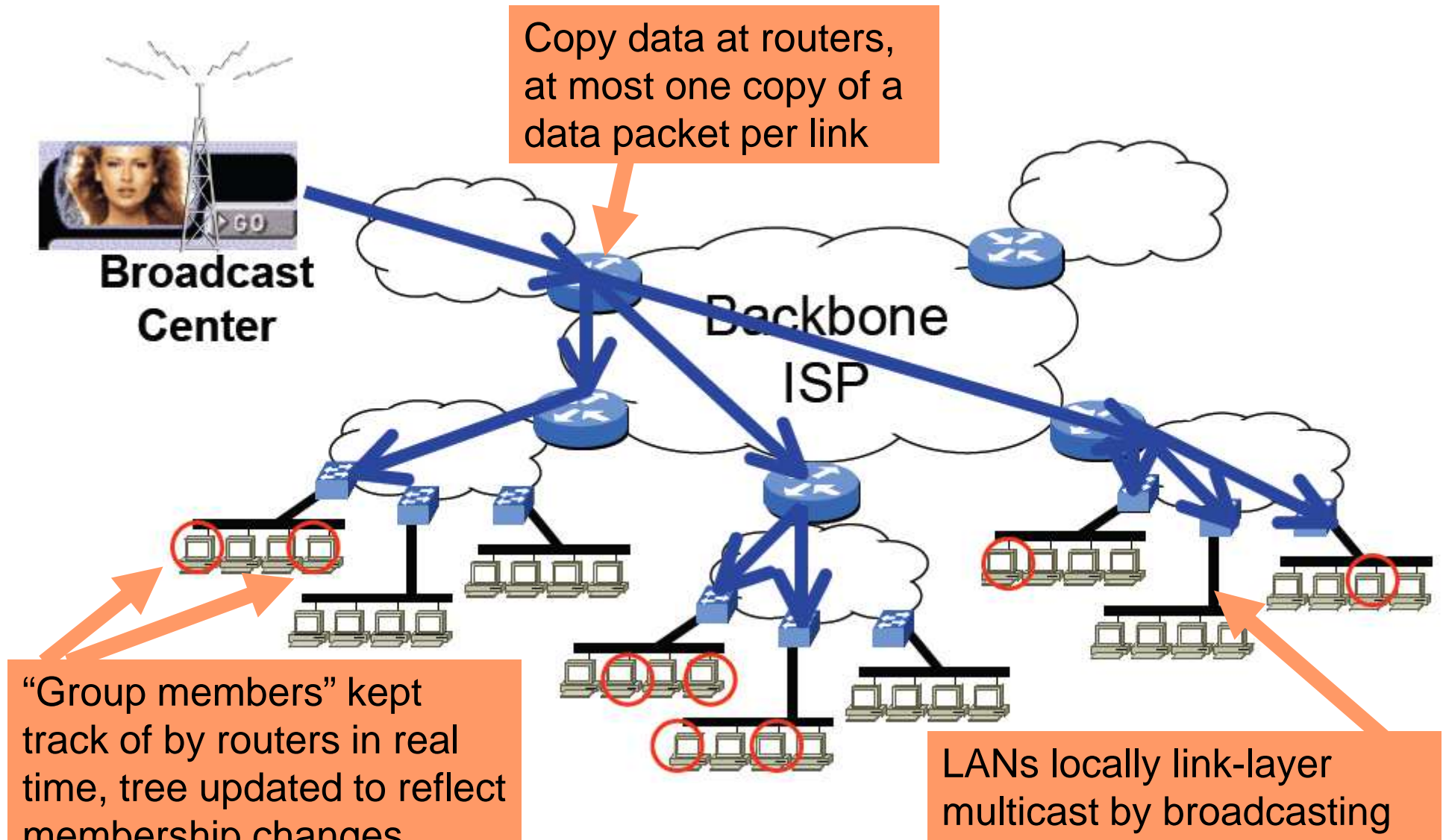
# Multicast: motivating example

- Example: Live 8 concert
  - Send ~300 Kbps video streams
  - Peak usage > 100,000 simultaneous users
  - Consumes > 30 Gbps
- If 1000 people in UIUC, and if the concert is broadcast from a single location, then 1000 unicast streams are sent from that location to UIUC

Problem: this approach does not scale



# Alternative: build trees

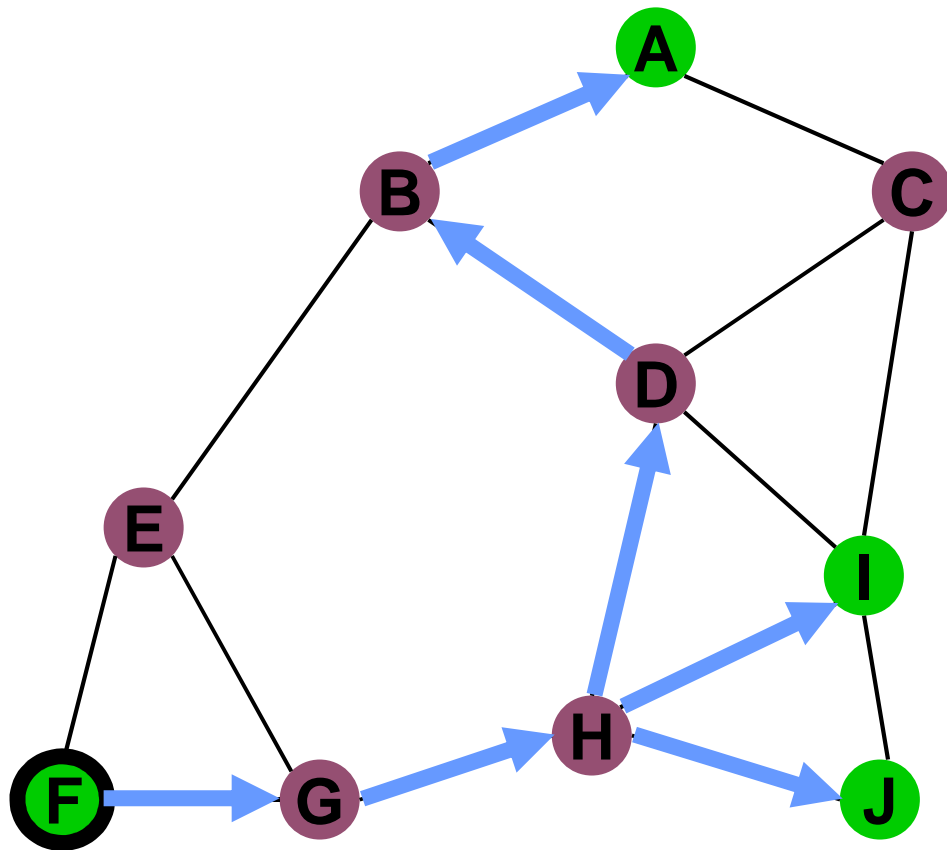




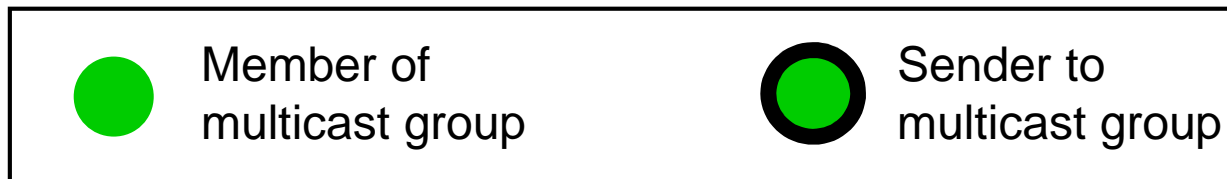
# Multicast routing approaches

- Kinds of trees
  - Source-specific trees vs. Shared tree
- Layer
  - Data-link, network, application
- Tree computation methods
  - Link state vs. Distance vector

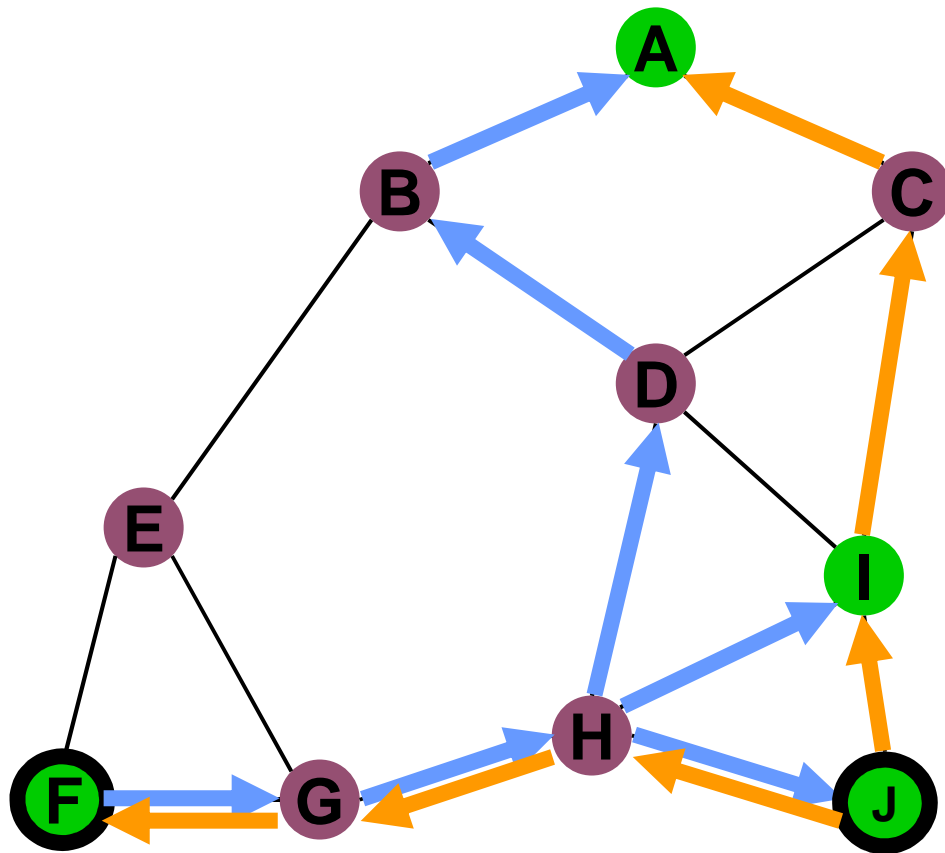
# Source-specific trees



- Each source is the root of its own tree
- One tree per source
- Tree consists of shortest paths to each receiver



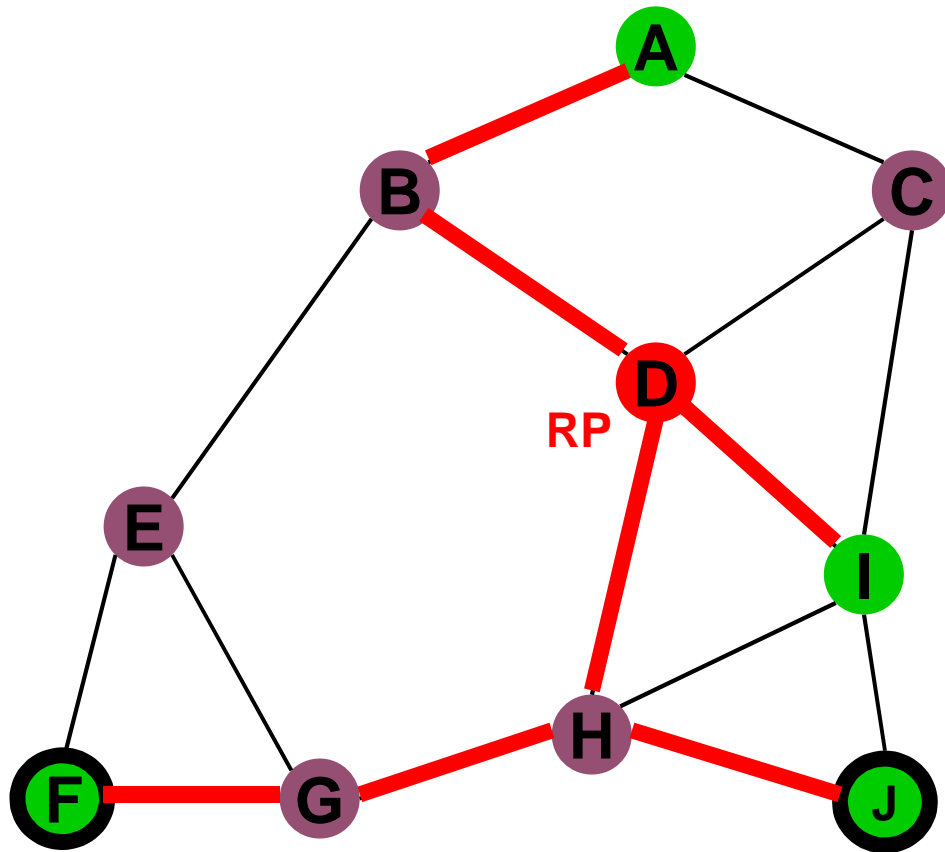
# Source-specific trees



- Each source is the root of its own tree
- One tree per source
- Tree consists of shortest paths to each receiver

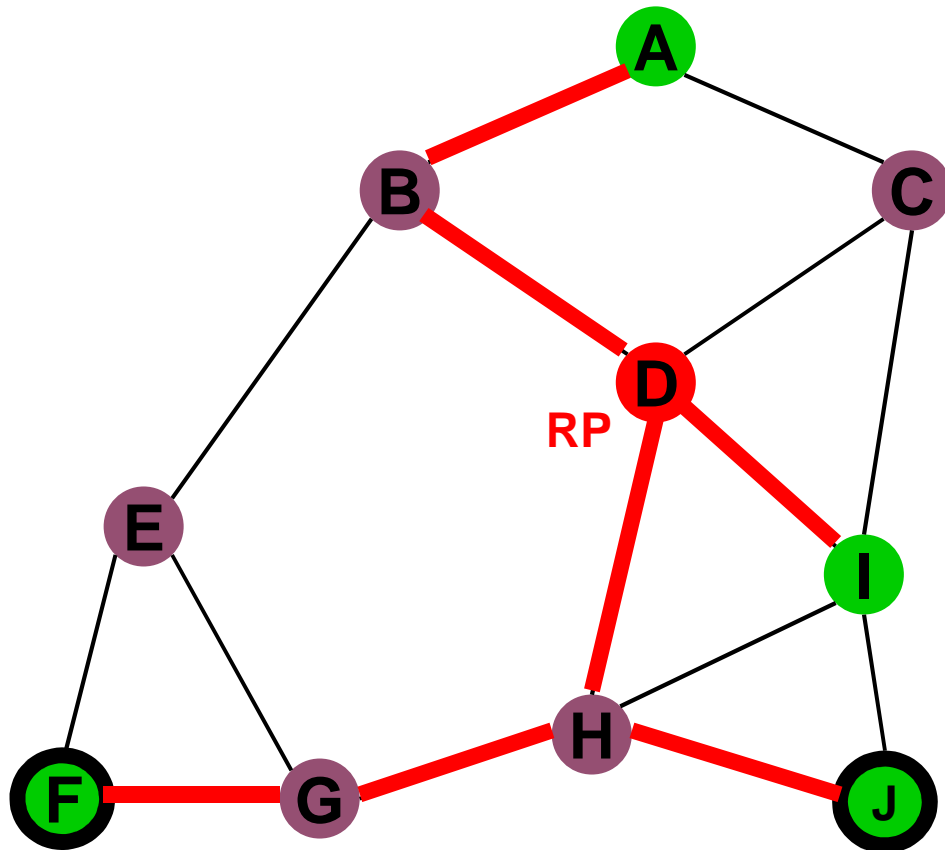


# Shared Tree



- One tree used by all members of a group
- Rooted at “rendezvous point” (RP)
- Less state to maintain, but hard to pick a tree that’s “good” for everybody!

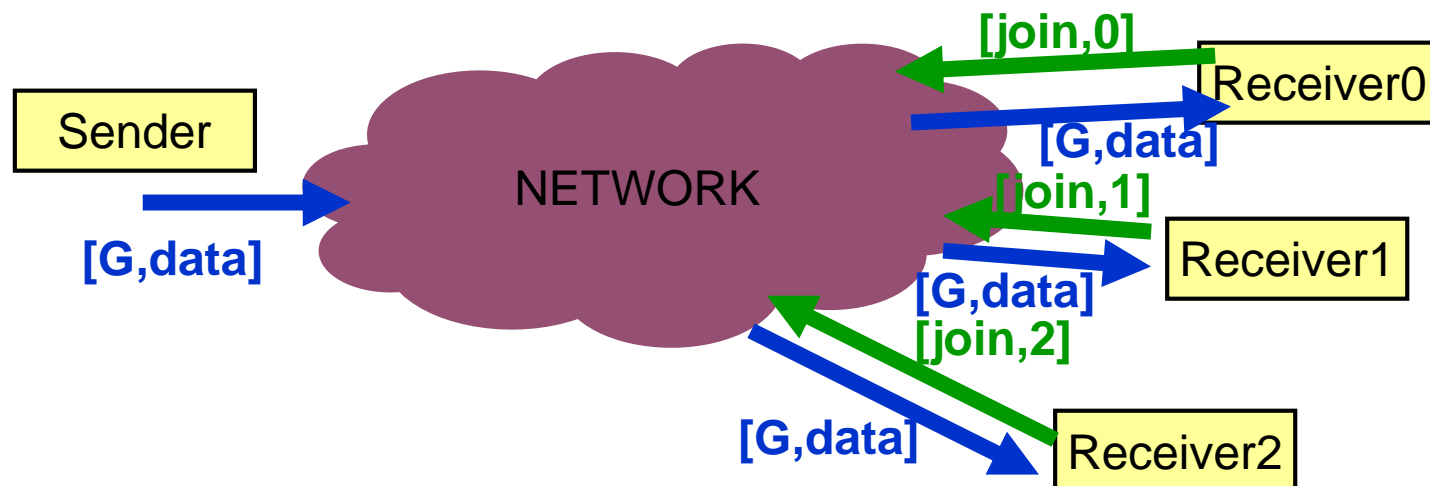
# Shared Tree



- Ideally, find a “Steiner tree” minimum-weighted tree connecting **only** the multicast members
  - Unfortunately, this is NP-hard
- Instead, use heuristics
  - E.g., find a minimum spanning tree (much easier)

# Multicast service model

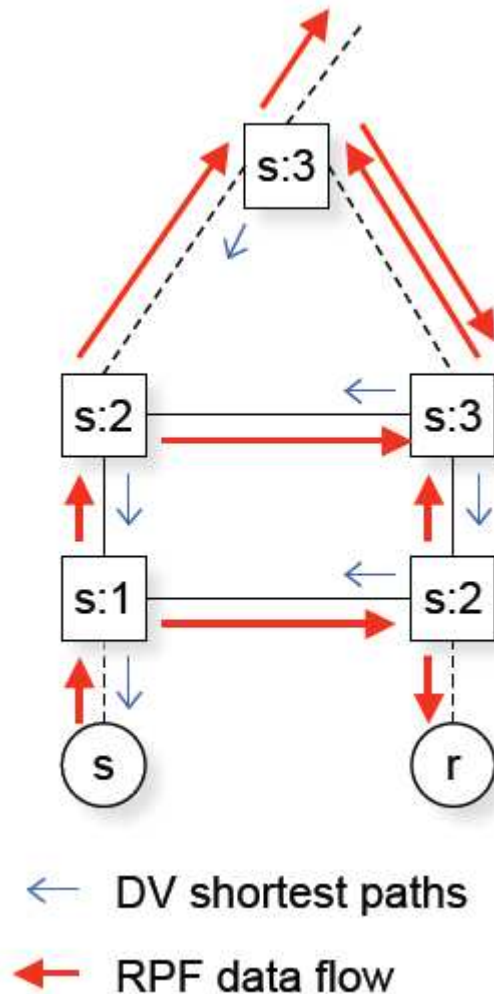
- Unicast: packets are delivered to one host
- Broadcast: packets are delivered to all hosts
- Multicast: packets are delivered to all hosts that have joined the multicast group
  - Multicast group identified by a multicast address



# Concepts

- Reverse-path forwarding
  - Regular routing protocols compute shortest path tree, so forward multicast packets along “reverse” of this tree
- Truncated reverse-path forwarding
  - Routers inform upstreams whether the upstream is on the router’s shortest path, to eliminate unnecessary broadcasting
- Flood-and-prune
  - Hosts must explicitly ask to not be part of multicast tree
  - Alternative: host explicitly sends “join” request to add self to tree

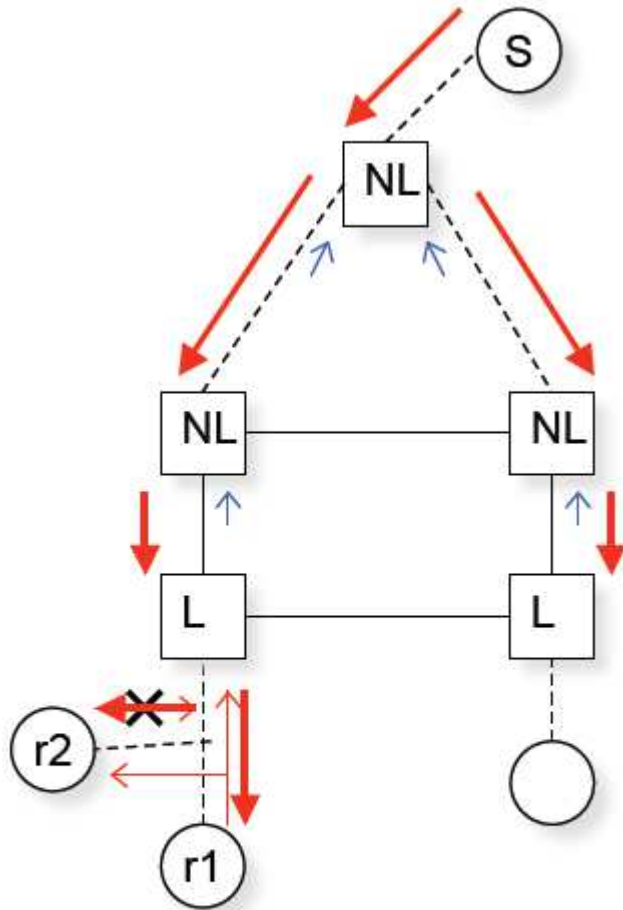
# Reverse-path forwarding



- Extension to DV routing
- Packet forwarding
  - If incoming link is shortest path to source
  - Send on all links except incoming
  - Packets always take shortest path
    - Assuming delay is symmetric
- Issues
  - Routers/LANs may receive multiple copies



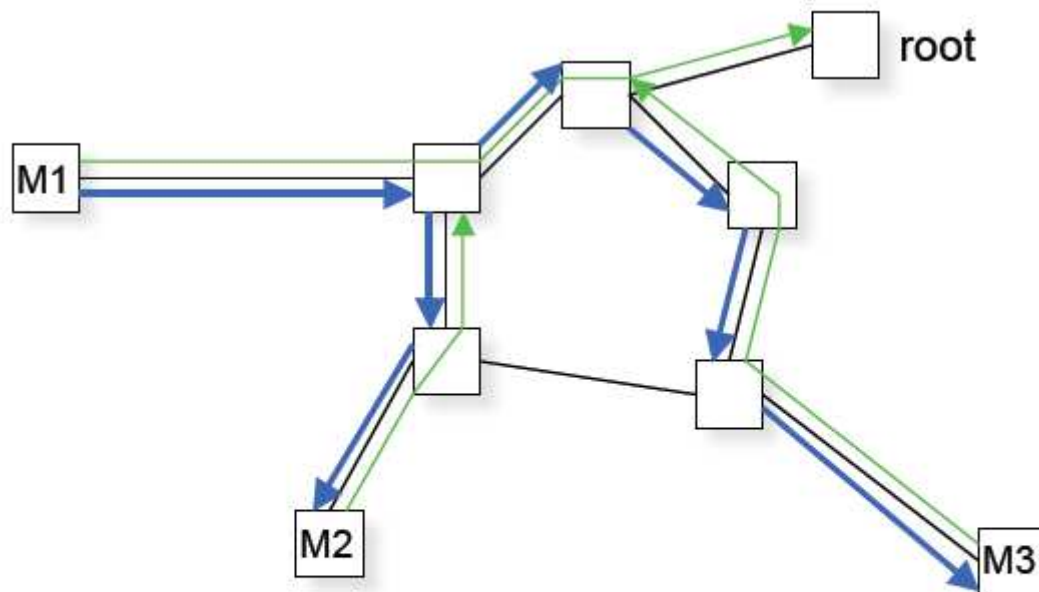
# Truncated reverse-path forwarding



L – leaf node  
NL – Non-leaf node

- Eliminate unnecessary forwarding
  - Routers inform upstreams if used on shortest path
  - Explicit group joining per-LAN
- Packet forwarding
  - If not a leaf router, or have members
  - Then send out all links except incoming

# Core-based trees



- Pick a rendezvous point for each group (called the “core”)
- Unicast packet to core and bounce back to multicast groups
- Reduces routing table state
  - By how much?
  - $O(S \cdot G)$  to  $O(G)$
- Finding optimal core location is hard (use heuristics)

# Other IP Multicast protocols

- Three ways for senders and receivers to “meet”:
  - Broadcast membership advertisement from each receiver to entire network
    - example: MOSPF
  - Broadcast initial packets from each source to entire network; non-members prune
    - examples: DVMRP, PIM-DM
  - Specify “meeting place” to which sources send initial packets, and receivers join; requires mapping between multicast group address and “meeting place”
    - examples: PIM-SM
- What are some problems with IP-layer Multicast?

# Problems with Network Layer Multicast

- Scales poorly with number of groups
  - Routers must maintain state for every group
  - Many groups traverse core routers
- Higher-layer functionality is difficult
  - NLM: **best-effort** delivery
  - Reliability, congestion control, transcoding for NLM complicated
- Deployment is difficult and slow
  - ISPs reluctant to turn on NLM

# Problems with Network Layer Multicast

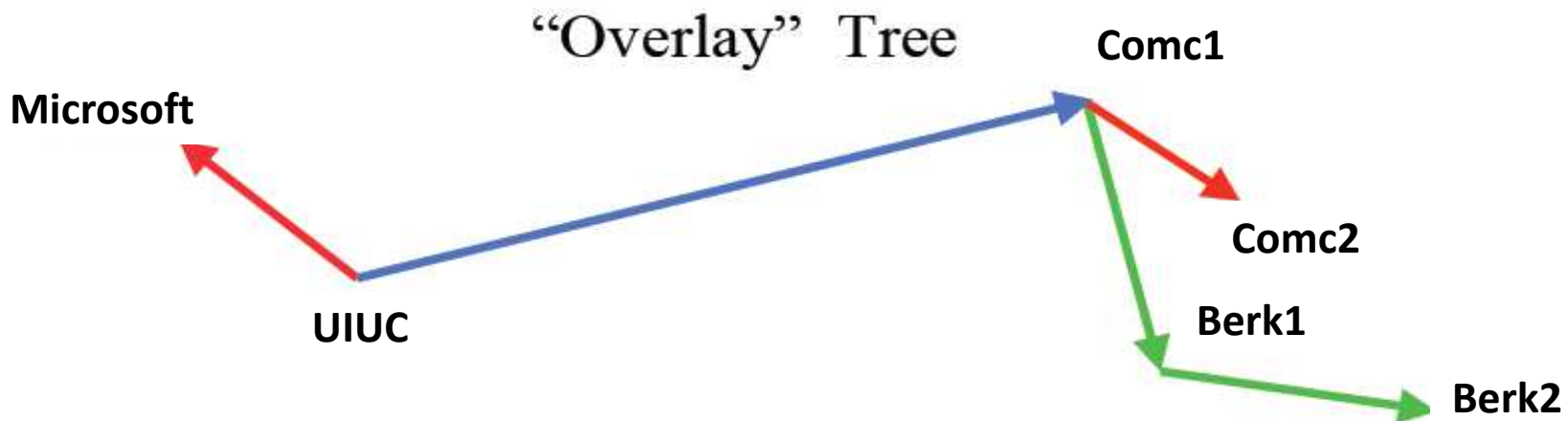
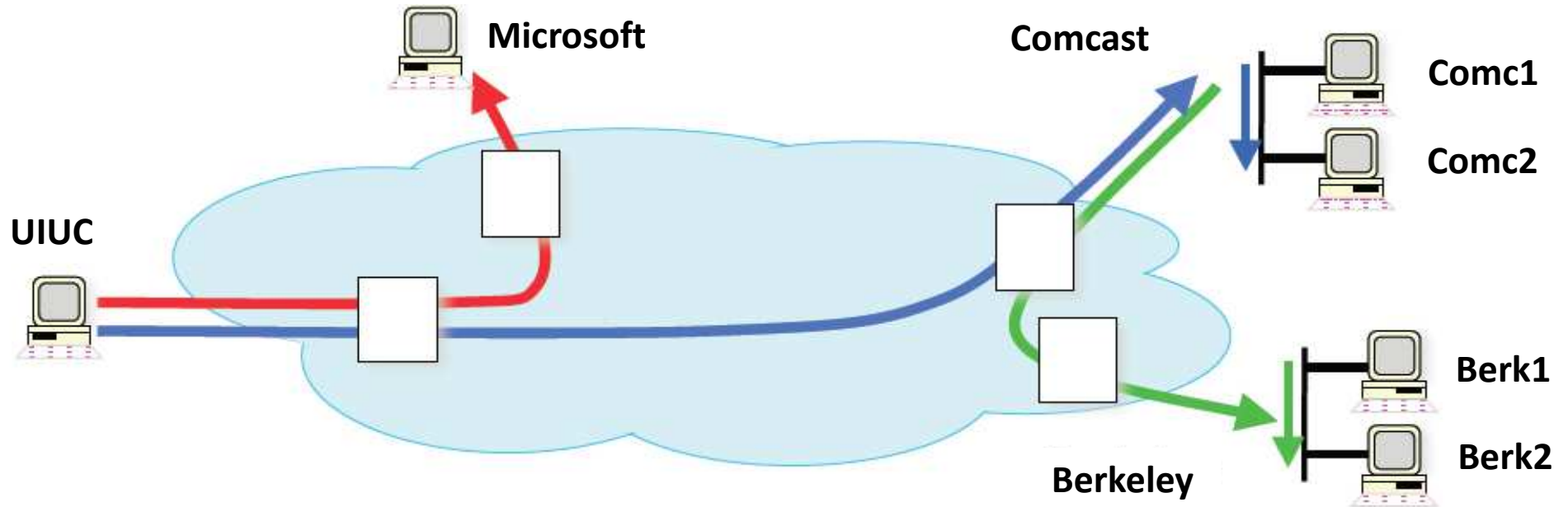
- Inconsistent with ISP charging model
  - Charging today is based on send rate of customer
  - But one multicast packet at ingress may cause millions to be sent on egress
- Troublesome security model
  - Anyone can send to group
  - Denial of service attacks on groups

# Alternative:

## Application-layer Multicast

- Let hosts do all packet copying, tree construction
  - Only require unicast from network
  - Hosts construct unicast channels between themselves to form tree
- Benefits
  - No need to change IP, or for ISP cooperation
  - End hosts can prevent untrusted hosts from sending
  - Easy to implement reliability (per-hop retransmissions)
- Downsides
  - Stretch penalty (latency), Stress penalty (multiple retransmissions over same physical link)

# Example of Application-level Multicast



# IPv6: proposed next generation of IP

- Problems with IPv4
  - Running out of address space
    - Projected depletion 2015-2032
  - Forwarding complicated by fragmentation, checksum computation, many unused fields
- IPv6 adopted by IETF in 1994
- IPv6 deployed incrementally, runs in parallel with IPv4
  - Routers distinguish packets based on version number



# IPv6: Features

→ Larger address space

- $2^{128}$  ( 340,282,366,920,938,000,000,000,000,000,000,000,000) addresses
- $2^{95}$  addresses for every person alive,  $2^{52}$  addresses for every observable star in the universe
- But goal is to simplify addressing, not geographic saturation of devices
  - More hierarchical, systematic approach to allocation
  - Avoids need for splitting up prefixes, renumbering networks

# IPv6: Features

## → Automatic host configuration

- IPv6 hosts probe network to discover gateway, acquire IPv6 address
- “Stateless”: upstream router stores no per-host information
- Steps:
  - Host locally derives IP address from its MAC address
  - Broadcasts to make sure that IP address is not in use on the network
  - Contacts gateway router to get other configuration information
  - Host assigns itself IP address determined above

# IPv6: Other features

- Simplified packet processing
  - Removed rarely used fields
  - Hosts must perform MTU discovery, fragmentation disallowed
  - IPv6 header has no checksum (integrity assumed to assured by transport level)
- Mobile IPv6 simplifies mobility
  - Maintains connectivity while end host moves
- Improved security
  - IPSec support is mandatory in IPv6
- What are the downsides of IPv6?

# IPv6 Deployment challenges

- Requires infrastructure changes
  - What kind of changes?
  - Hardware: forwarding engines
  - Software: routing protocols
- However, certain strategies simplify deployment
  - Dual stack: routers run IPv4 and IPv6
  - IPv4 addresses can be mapped to IPv6 addresses
    - First 80 bits set to 0, next 16 set to one, final 32 are IPv4 address
  - Tunneling: IPv6 packets encapsulated in IPv4 packets

# IPv6: Do we really need it?

- Larger address space
  - NAT reduces severity of address space depletion
  - Could just extend address sizes (IPv4+4)
- Simplified processing
  - Routers can do checksum processing in hardware at line speeds
- End host configuration, mobility, security
  - This functionality has been back-ported to IPv4
  - DHCP, IPSec, Mobile IP

# Current state of IPv6

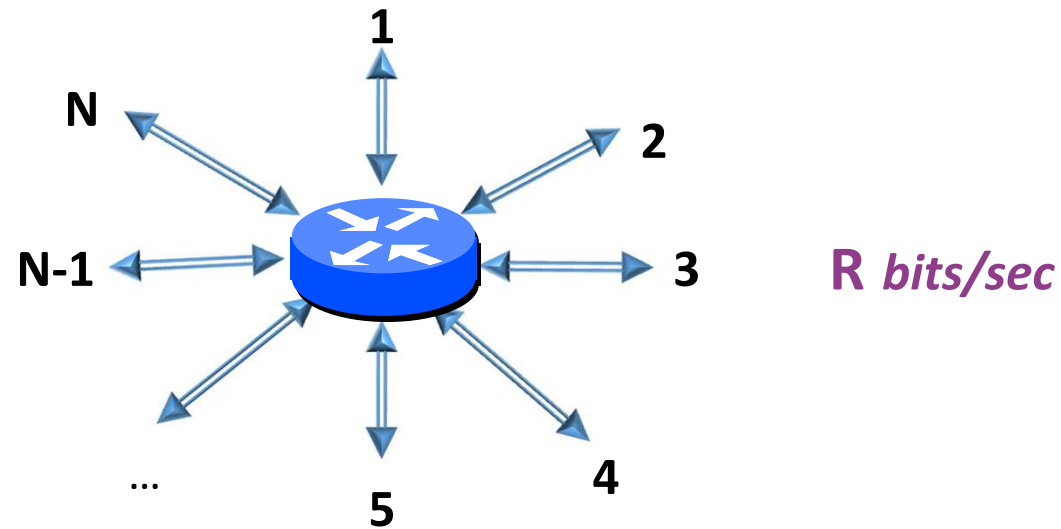
- *Prefix allocations* growing rapidly, but *traffic* showing no substantial growth
  - 500 allocations in 2004, 1200 allocations in 2009
  - Currently less than 1% of internet traffic is IPv6 (0.45% in USA, 0.24% in China)
- Possible reasons for slow growth
  - Lack of incentives (low demand from customers)
  - Not clear benefits are worth deployment efforts





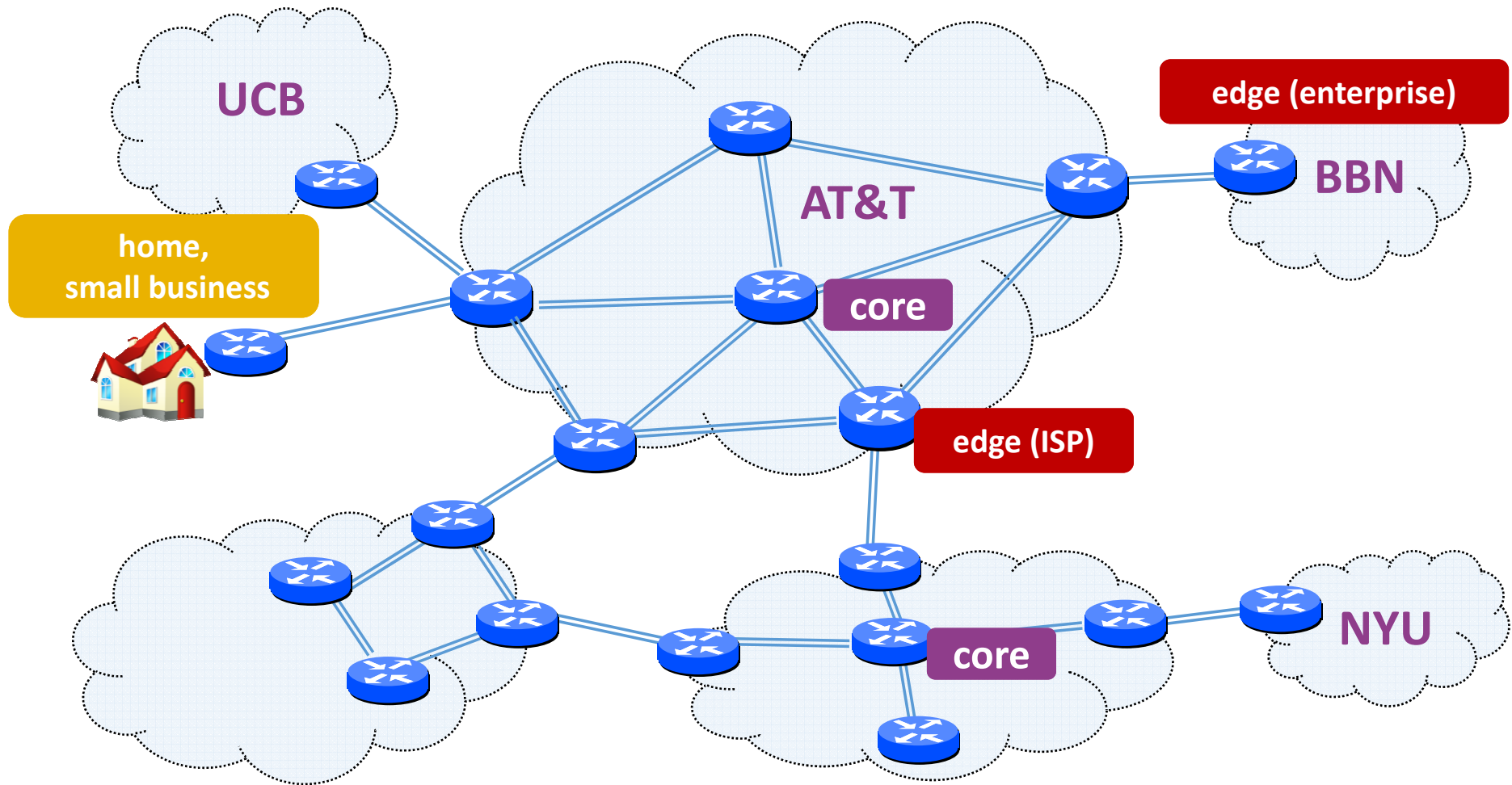


# Router definitions



- **N = number of external router “ports”**
- **R = speed (“line rate”) of a port**
- **Router capacity =  $N \times R$**

# Networks and routers



# Examples of routers (core)

## Juniper T4000

- R= 10/40 Gbps
- NR = 4 Tbps



## Cisco CRS

- R=10/40/100 Gbps
- NR = 322 Tbps



**72 racks, 1MW**

# Examples of routers (edge)

## Cisco ASR 1006

- R=1/10 Gbps
- NR = 40 Gbps



## Juniper M120

- R= 2.5/10 Gbps
- NR = 120 Gbps



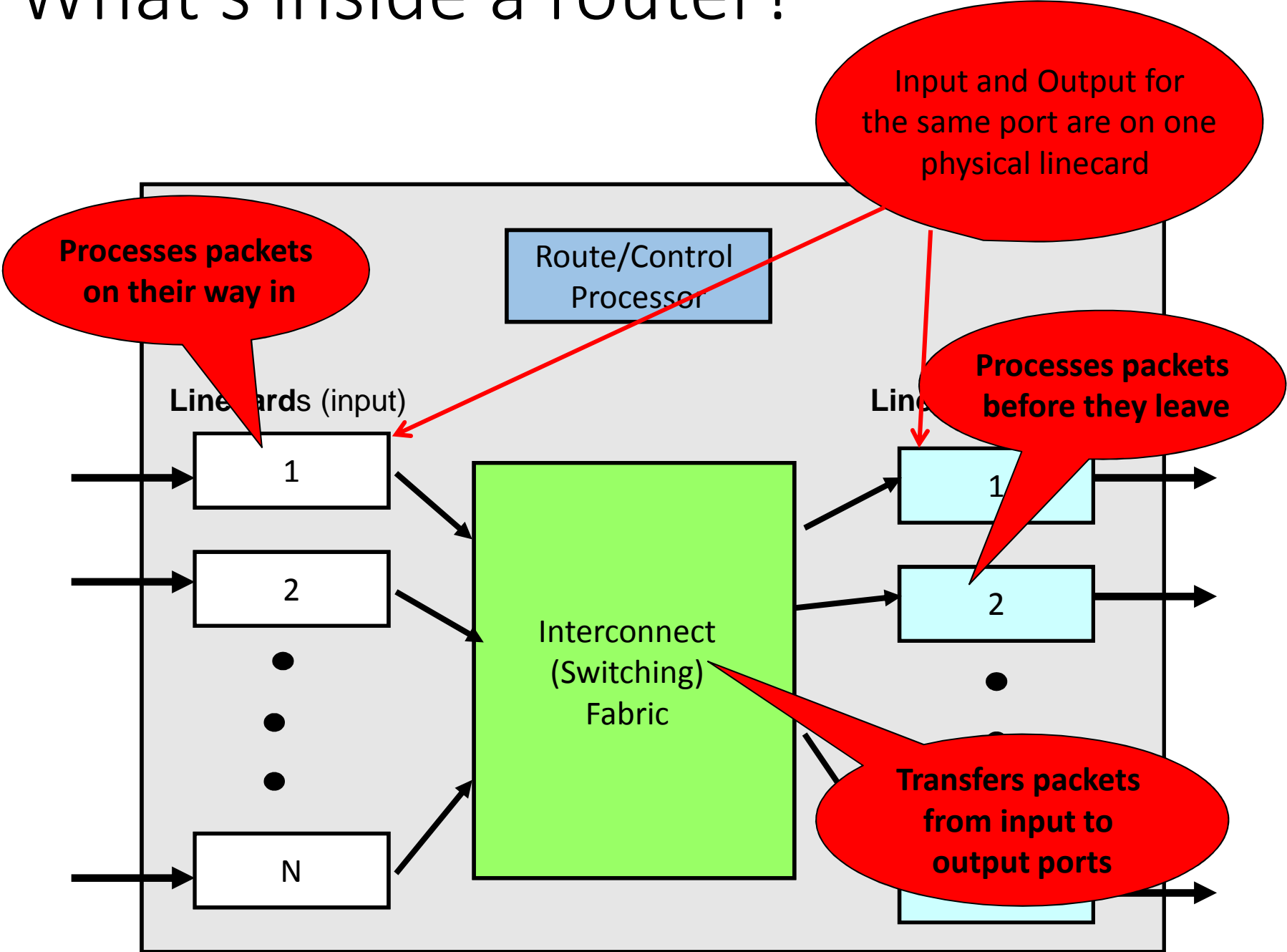
# Examples of routers (small business)

## **Cisco 3945E**

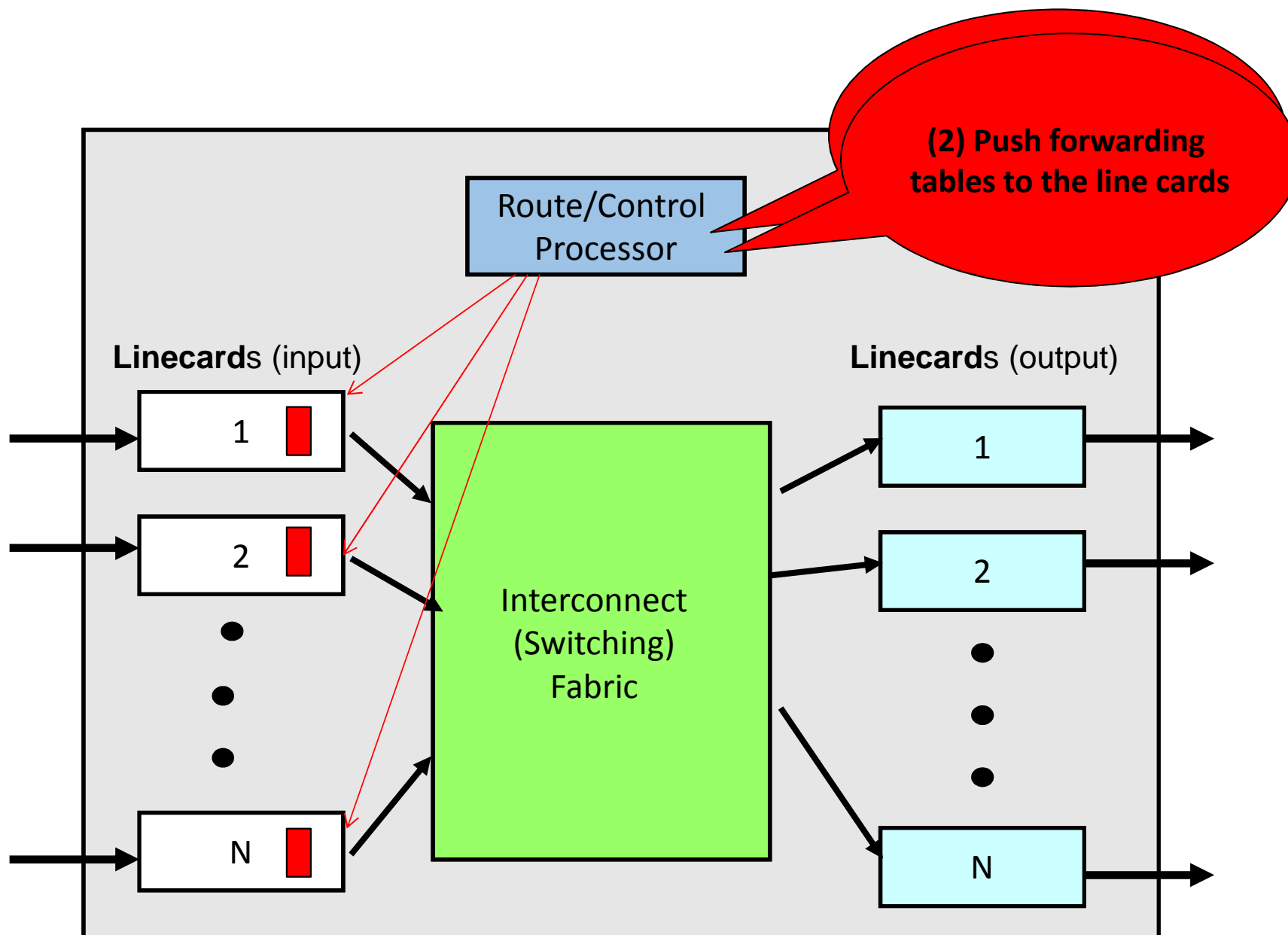
- R = 10/100/1000 Mbps
- NR < 10 Gbps



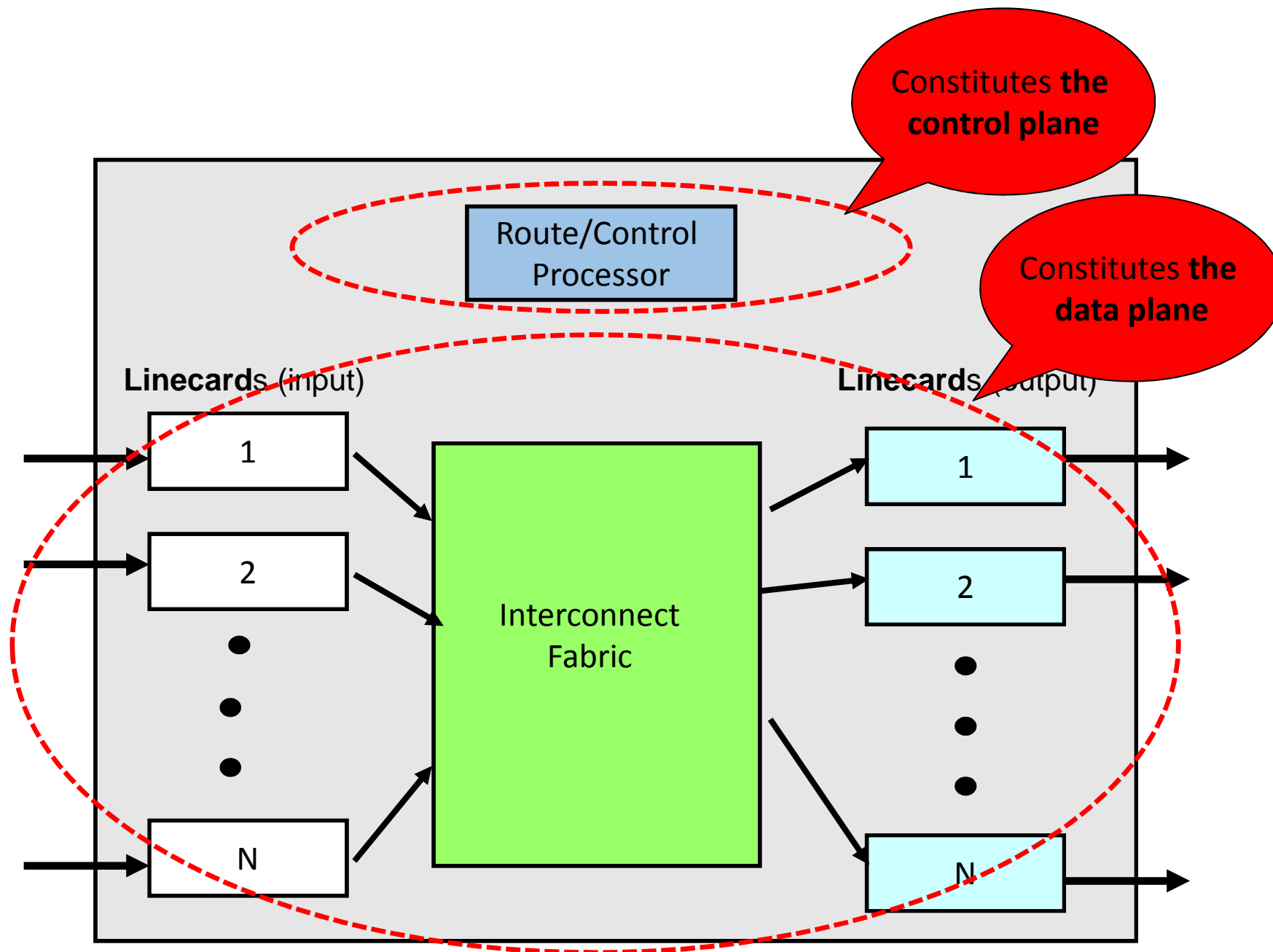
# What's inside a router?



# What's inside a router?



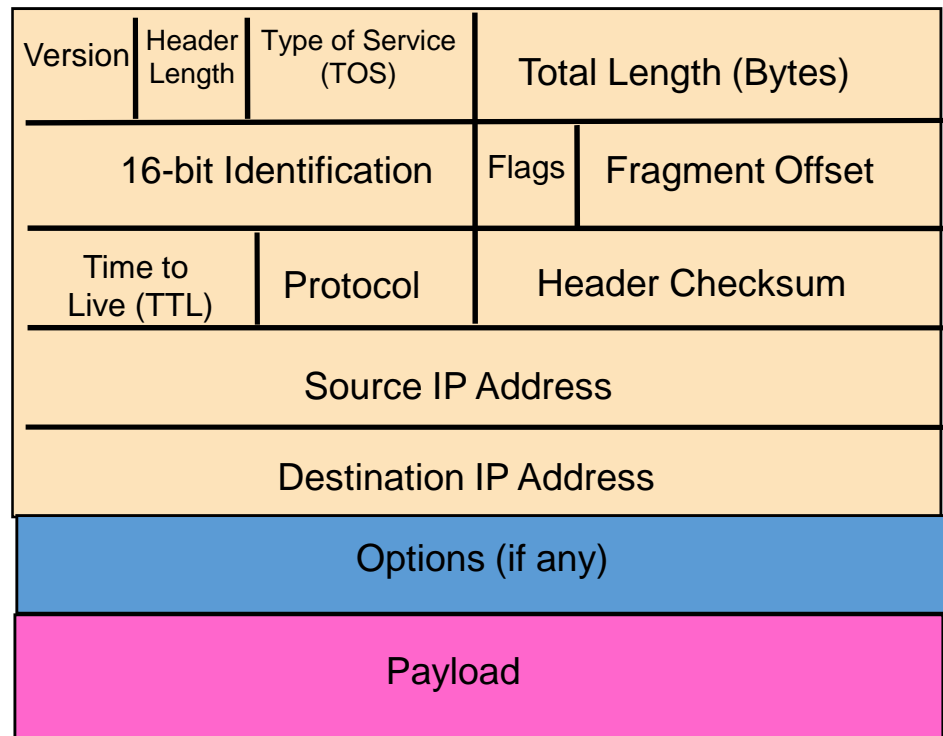
# What's inside a router?





# Input Linecards

- Tasks
  - Receive incoming packets (physical layer stuff)
  - Update the IP header



# Input Linecards

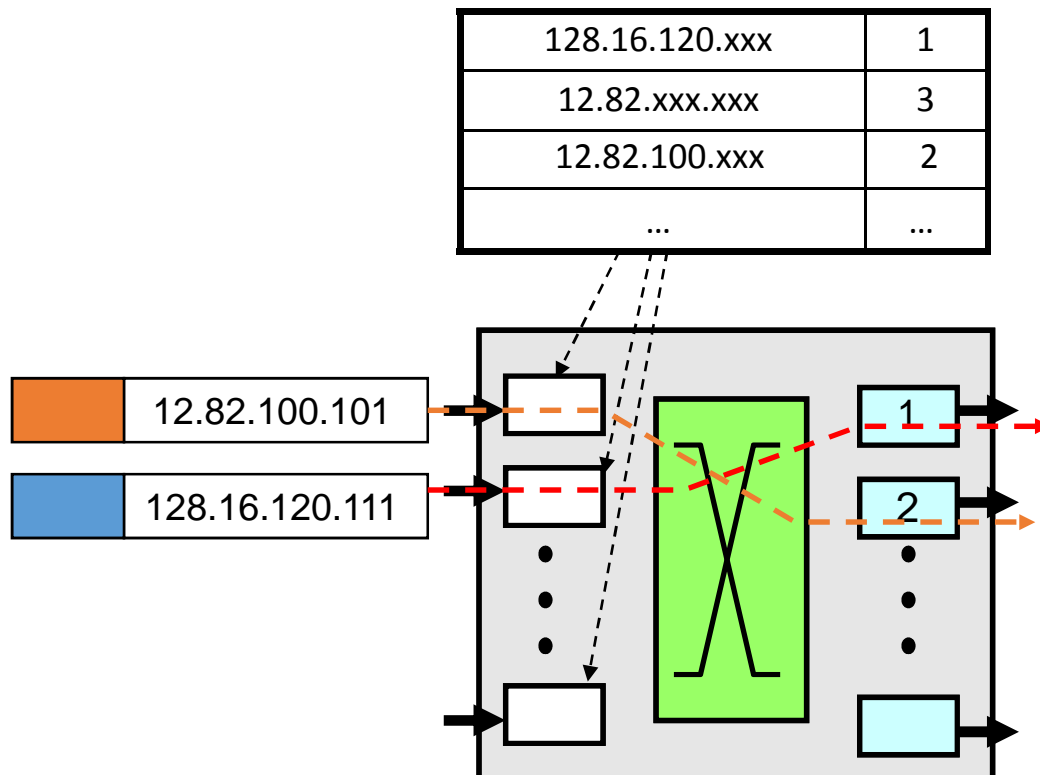
- Tasks
  - Receive incoming packets (physical layer stuff)
  - Update the IP header
    - TTL, Checksum, Options (maybe), Fragment (maybe)
  - Lookup the output port for the destination IP address
  - Queue the packet at the switch fabric
- Challenge: **speed!**
  - 100B packets @ 40Gbps → new packet every 20 nano secs!
- Typically implemented with specialized hardware
  - ASICs, specialized “network processors”
  - “exception” processing often done at control processor

# IP Lookup

- Recall IP addressing and BGP routing
  - For scalability, multiple IP addresses are **aggregated**
  - BGP operates on IP address **prefixes** (recall “/n” notation)
  - IP routing tables maintain a mapping from IP prefixes to output interfaces
- Route lookup → find the **longest** prefix in the table that matches the packet destination address
  - Longest Prefix Match (LPM) lookup

# Longest Prefix Match Lookup

- Packet with destination address 12.82.100.101 is sent to interface 2, as 12.82.100.xxx is the longest prefix matching packet's destination address



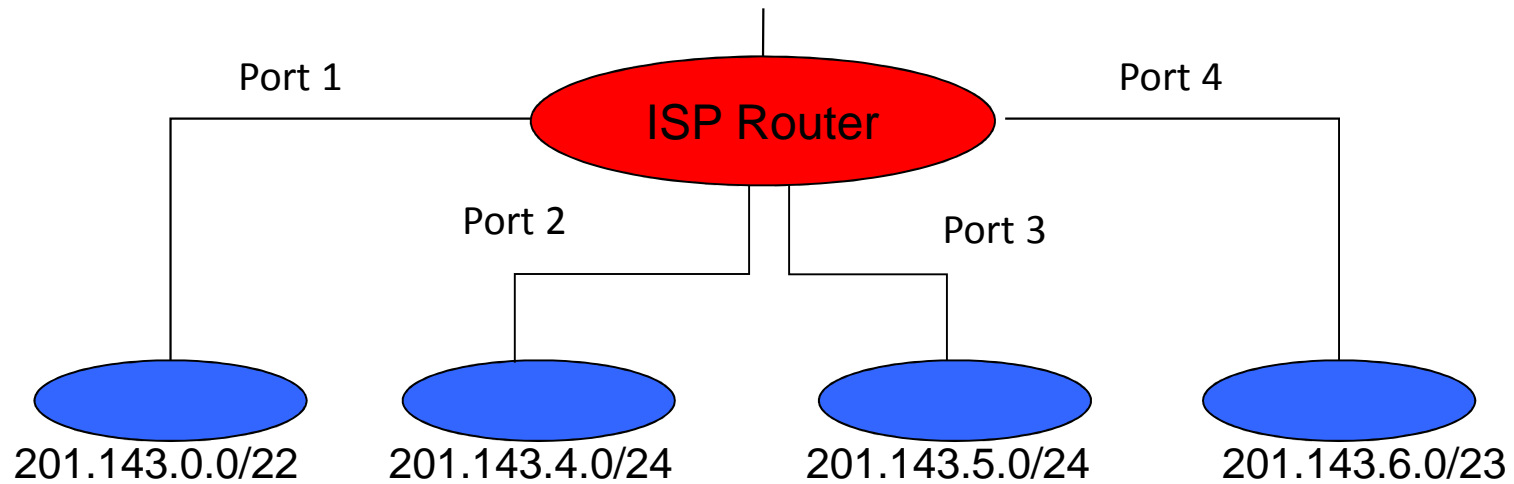
# Longest Prefix Match is NOT...

- Check an address against all destination prefixes and select the prefix it matches with on the most bits

101.xx.xxx.xxx (/3)	1
1**.***.*xxx.xxx(/1)	3
...	..
...	...

- To which port should we send a packet with destination address 100.5.6.7?

# Example #1: 4 Prefixes, 4 Ports



<b>Prefix</b>	<b>Port</b>
201.143.0.0/22	Port 1
201.143.4.0.0/24	Port 2
201.143.5.0.0/24	Port 3
201.143.6.0/23	Port 4

# Finding the Match

201.143.0.0/22	11001001	10001111	000000---	-----	★
201.143.4.0/24	11001001	10001111	00000100	-----	★
201.143.5.0/24	11001001	10001111	00000101	-----	★
201.143.6.0/23	11001001	10001111	0000011-	-----	★

- Consider 11001001100011110000010111010010
  - First 21 bits match 4 partial prefixes
  - First 22 bits match 3 partial prefixes
  - First 23 bits match 2 partial prefixes
  - First 24 bits match exactly one full prefix



# Finding Match Efficiently

- Testing each entry to find a match scales poorly
  - On average:  $O(\text{number of entries})$
- Leverage tree structure of binary strings
  - Set up tree-like data structure
- Return to example:

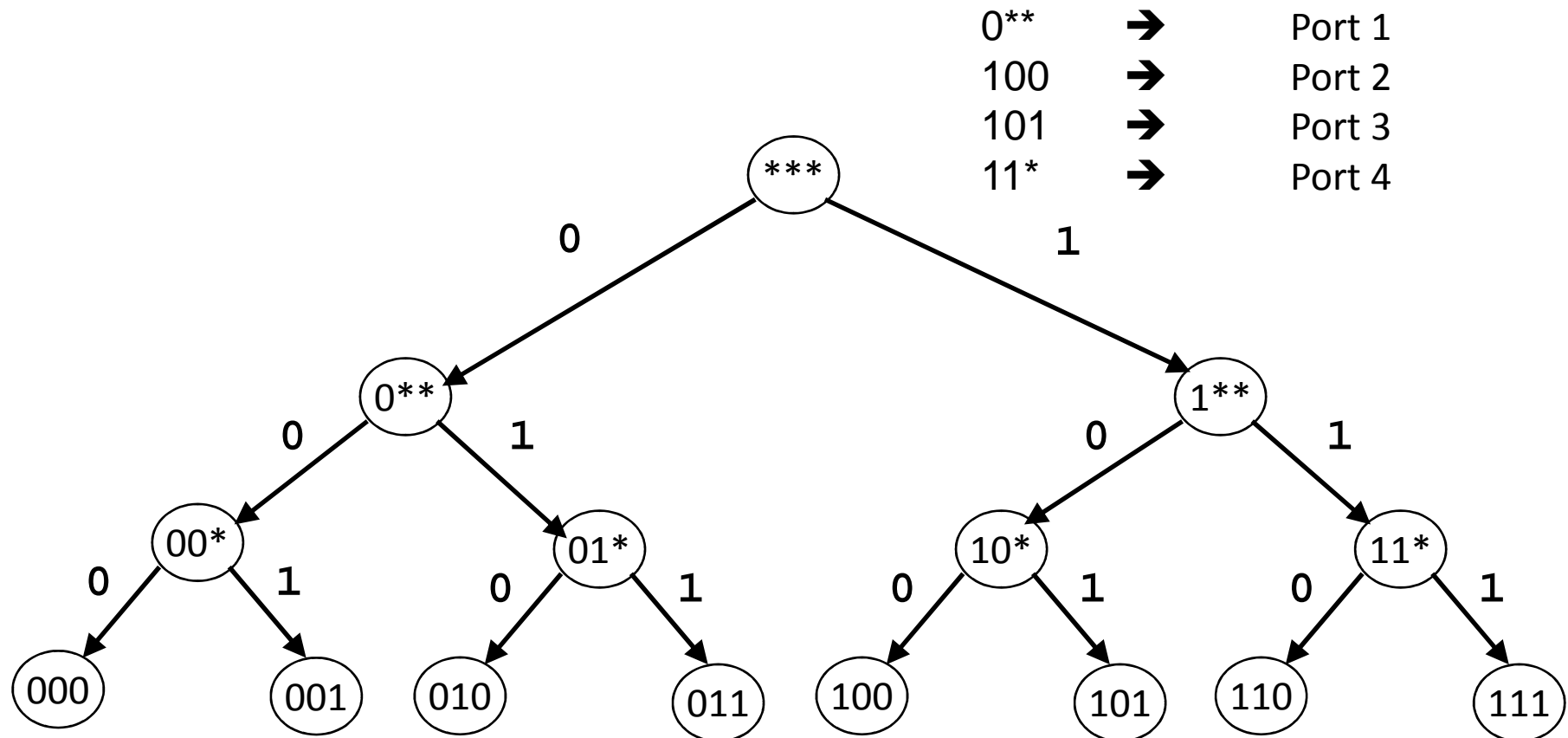
Prefix	Port
11001001100011110000000*****	1
1100100110001111000000100*****	2
1100100110001111000000101*****	3
110010011000111100000011*****	4



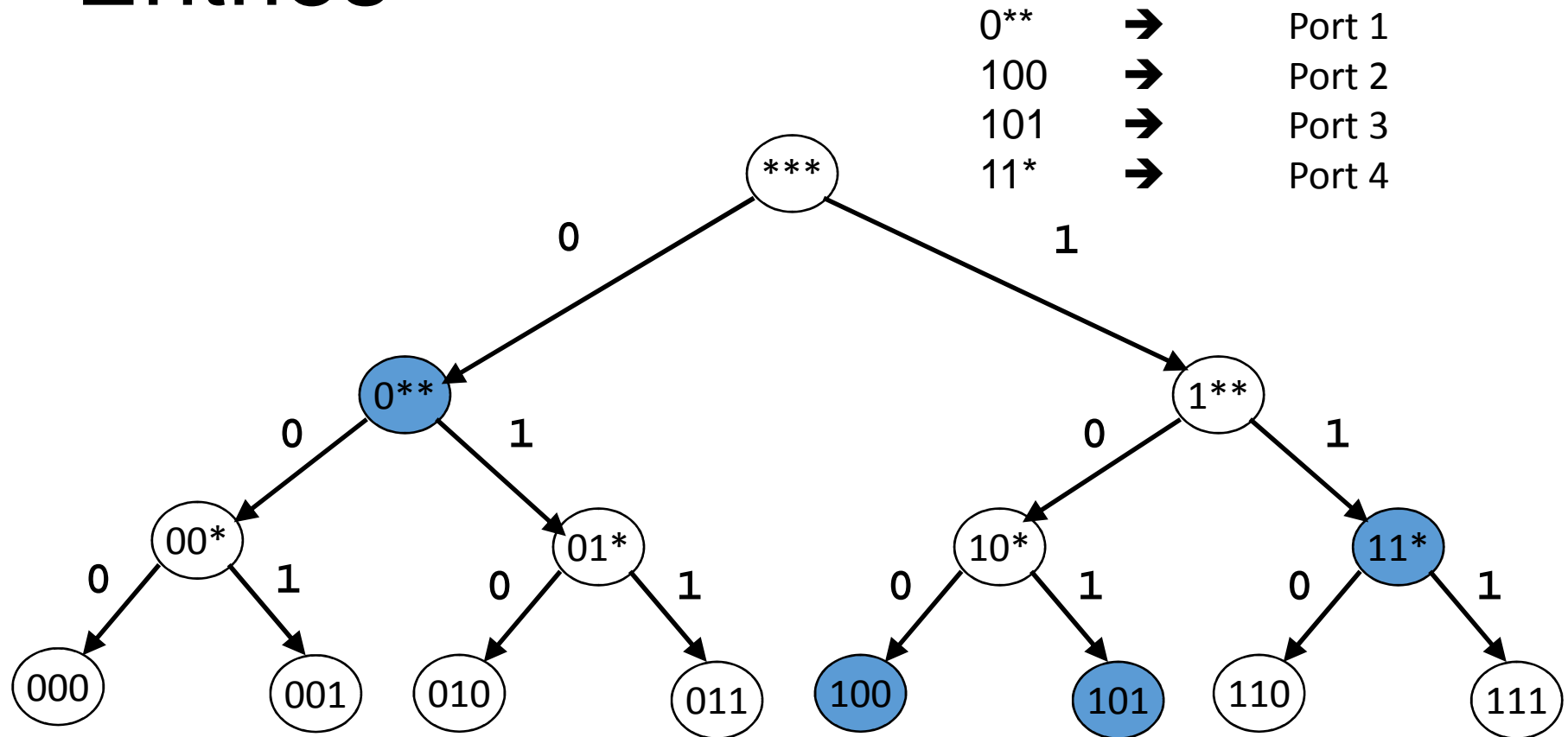
# Consider four three-bit prefixes

- Just focusing on the bits where all the action is....
- $0^{**} \rightarrow$  Port 1
- 100  $\rightarrow$  Port 2
- 101  $\rightarrow$  Port 3
- $11^*$   $\rightarrow$  Port 4

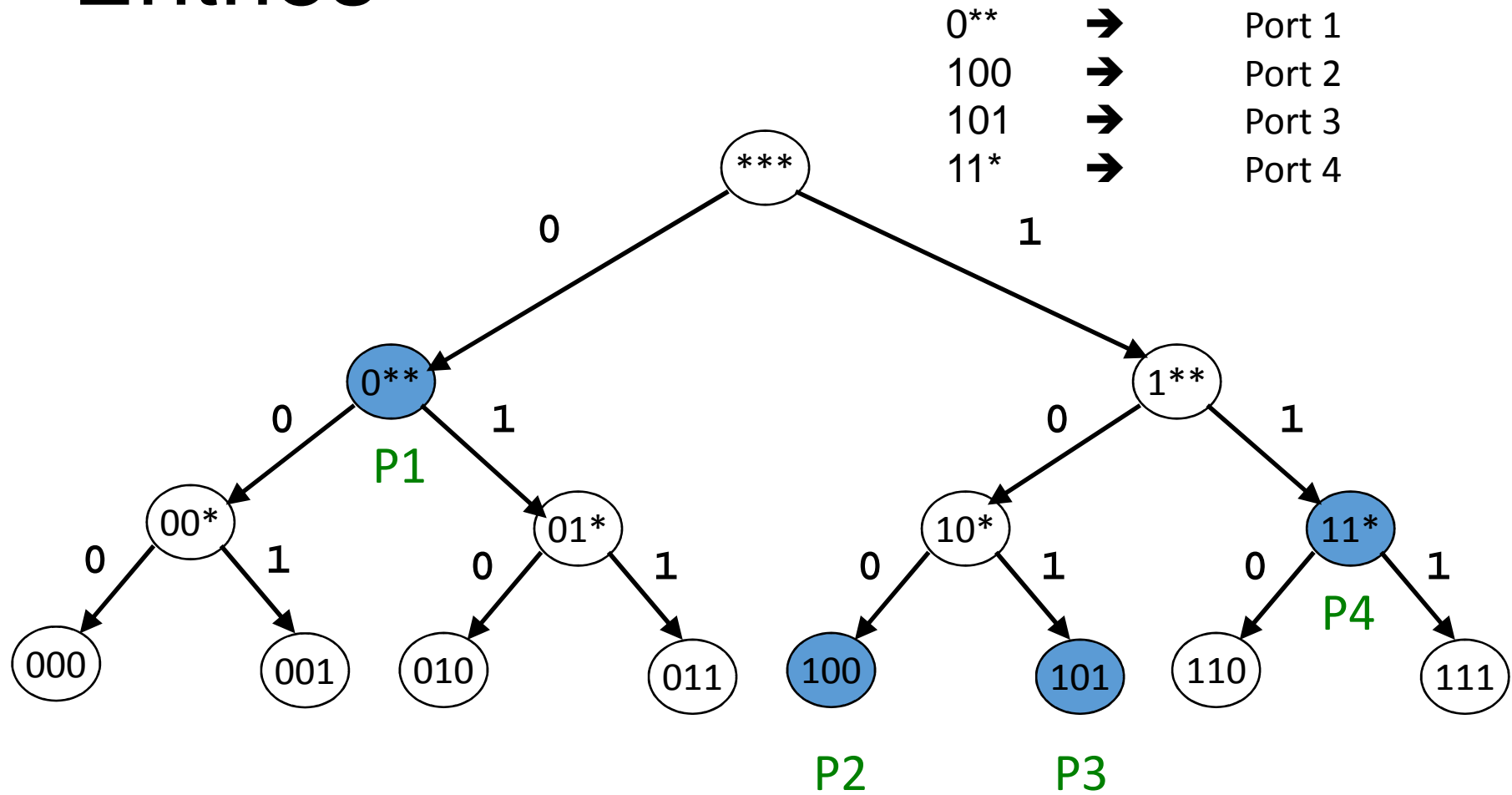
# Tree Structure



# Walk Tree: Stop at Prefix Entries



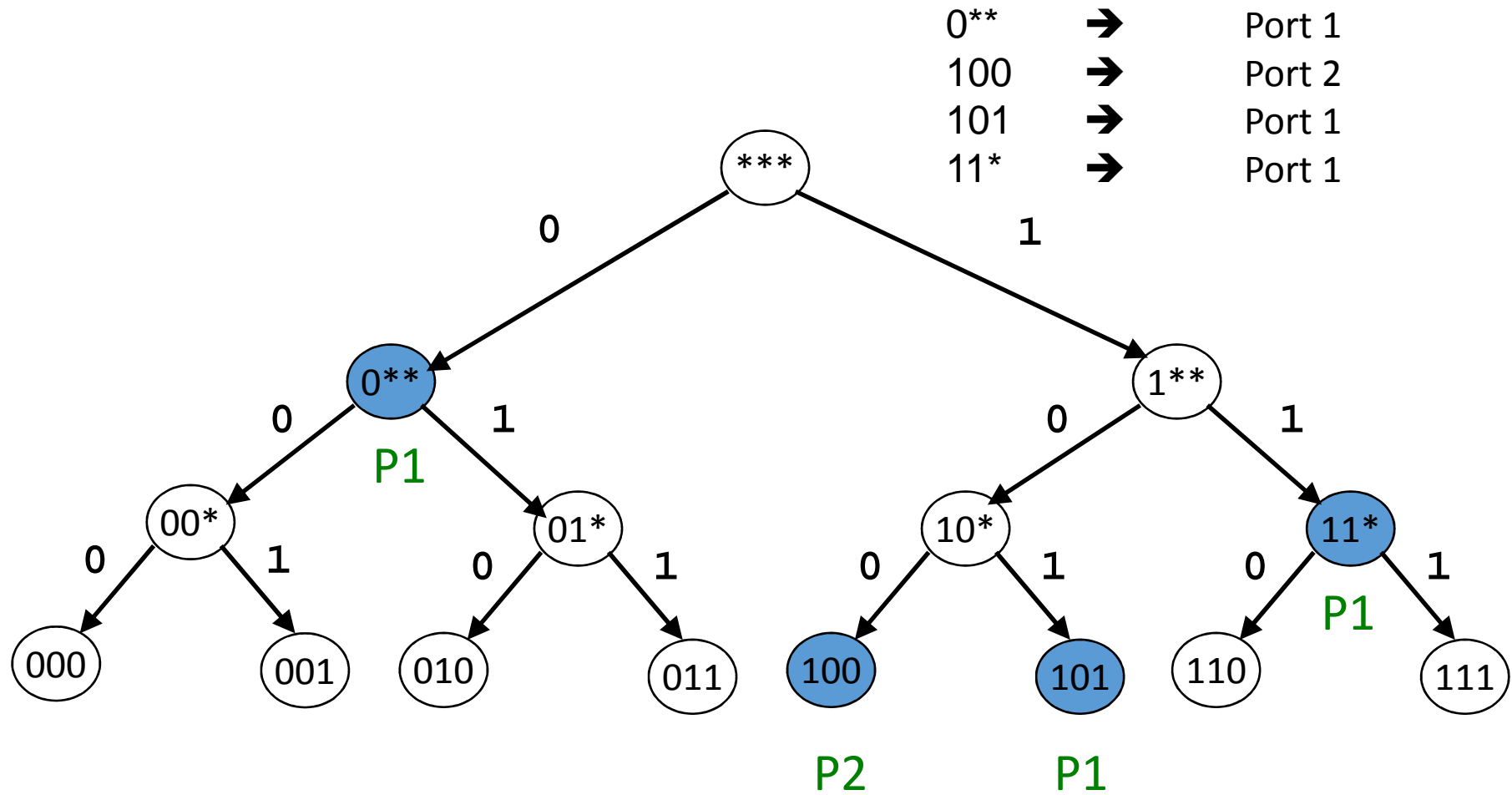
# Walk Tree: Stop at Prefix Entries



# Slightly Different Example

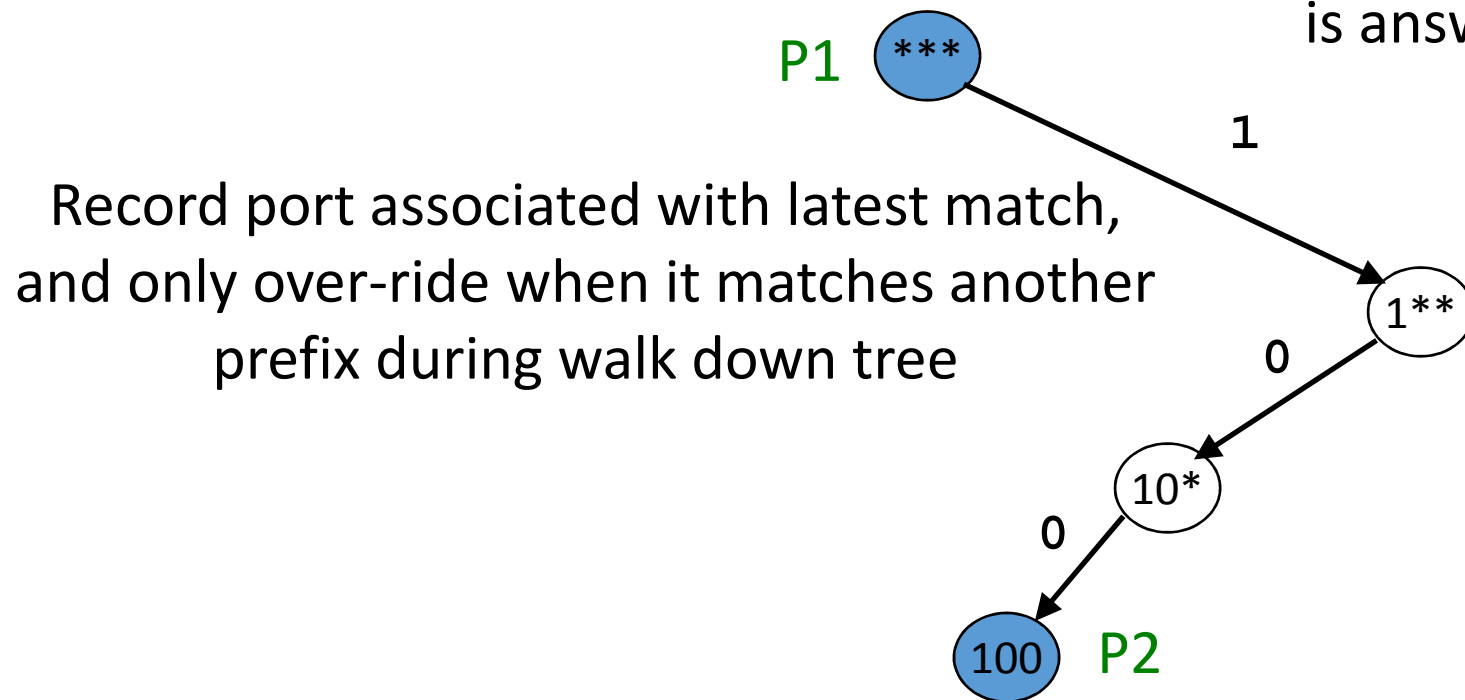
- Several of the unique prefixes go to same port
- $0^{**} \rightarrow$  Port 1
- 100  $\rightarrow$  Port 2
- 101  $\rightarrow$  Port 1
- $11^*$   $\rightarrow$  Port 1

# Prefix Tree



# More Compact Representation

If you ever leave path, you are done, last matched prefix is answer



# LPM in real routers

- Real routers use far more advanced/complex solutions than the approaches I just described
  - but what we discussed is their starting point
- With many heuristics and optimizations that leverage real-world patterns
  - Some destinations more popular than others
  - Some ports lead to more destinations
  - Typical prefix granularities

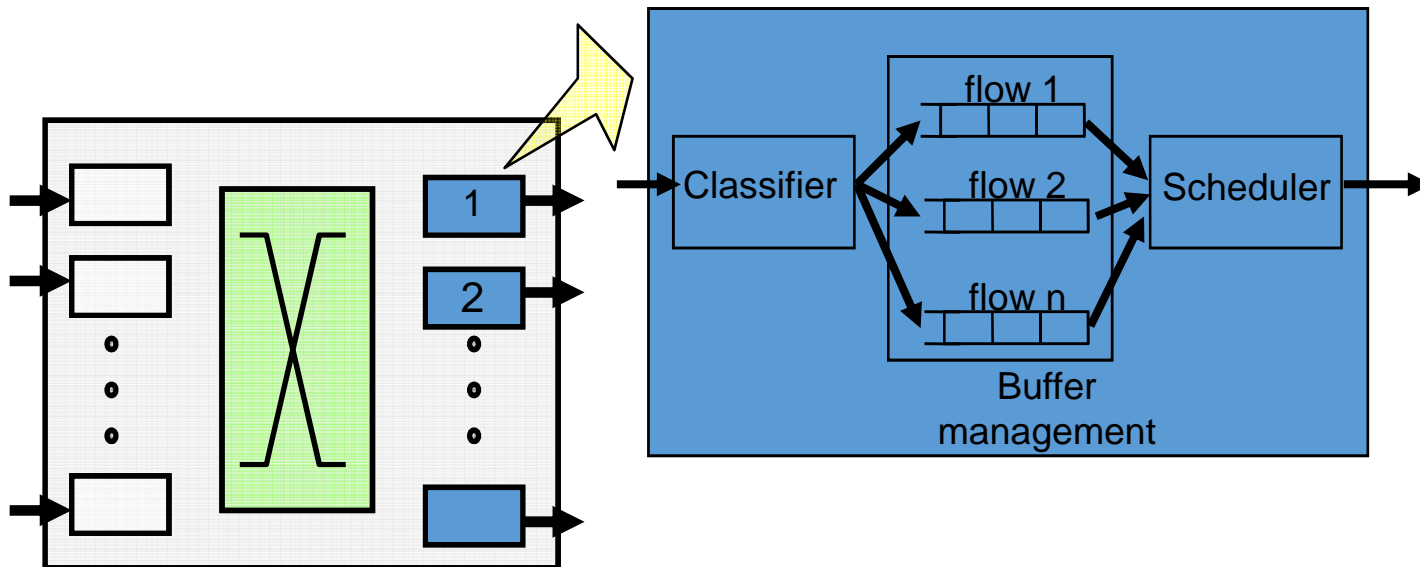


# Recap: Input linecards

- Main challenge is processing speeds
- Tasks involved:
  - Update packet header (easy)
  - LPM lookup on destination address (harder)
- Mostly implemented with specialized hardware

# Output Linecard

- **Packet classification:** map each packet to a “flow”
  - Flow (for now): set of packets between two particular endpoints
- **Buffer management:** decide when and which packet to drop
- **Scheduler:** decide when and which packet to transmit

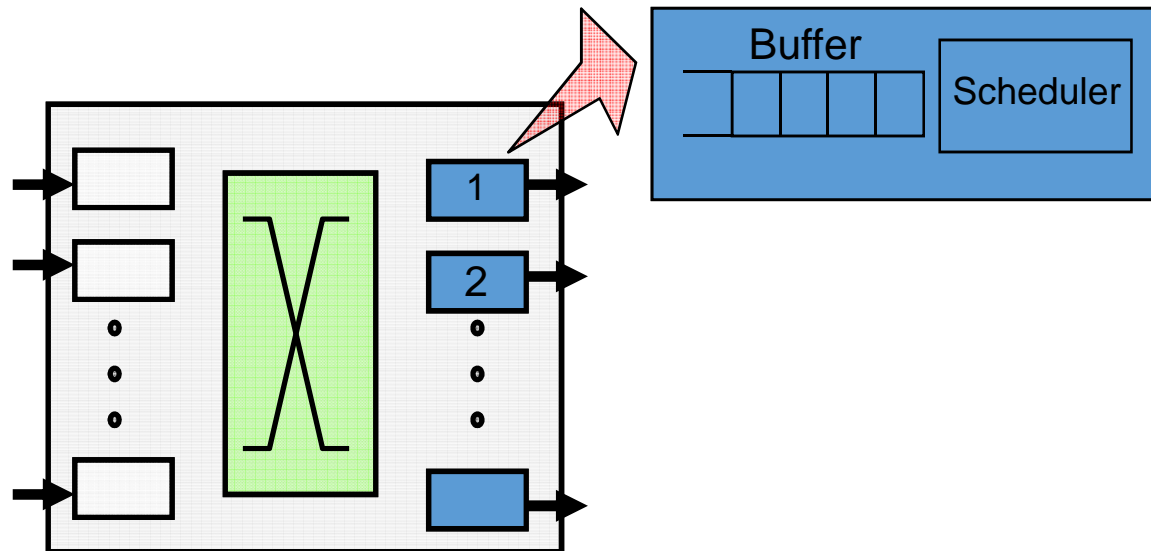


# Output Linecard

- **Packet classification:** map each packet to a “flow”
  - Flow (for now): set of packets between two particular endpoints
- **Buffer management:** decide when and which packet to drop
- **Scheduler:** decide when and which packet to transmit
  
- Used to implement various forms of policy
  - Deny all e-mail traffic from ISP-X to Y (access control)
  - Route IP telephony traffic from X to Y via PHY\_CIRCUIT (policy)
  - Ensure that no more than 50 Mbps are injected from ISP-X (QoS)

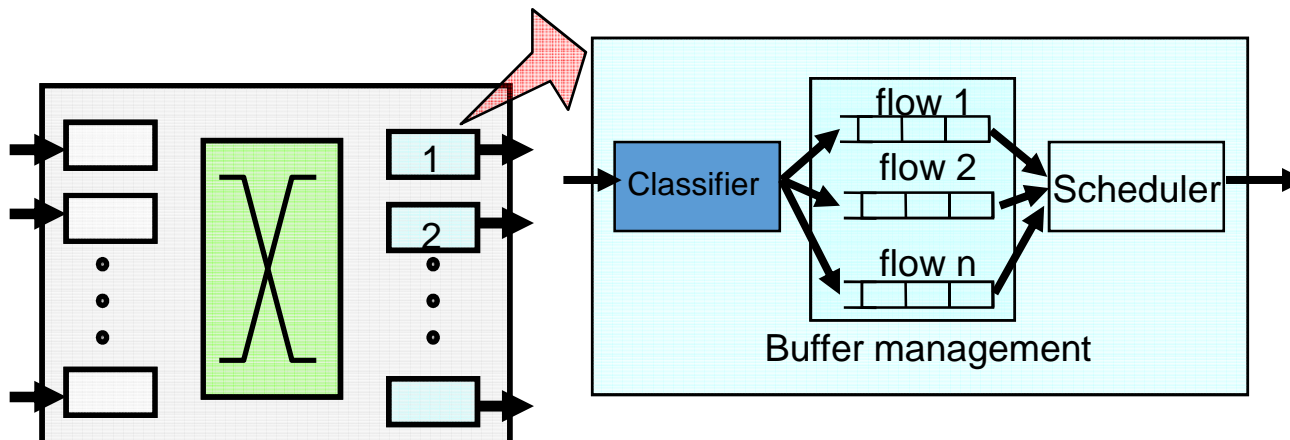
# Simplest: FIFO Router

- No **classification**
- **Drop-tail buffer management**: when buffer is full drop the incoming packet
- **First-In-First-Out (FIFO) Scheduling**: schedule packets in the same order they arrive



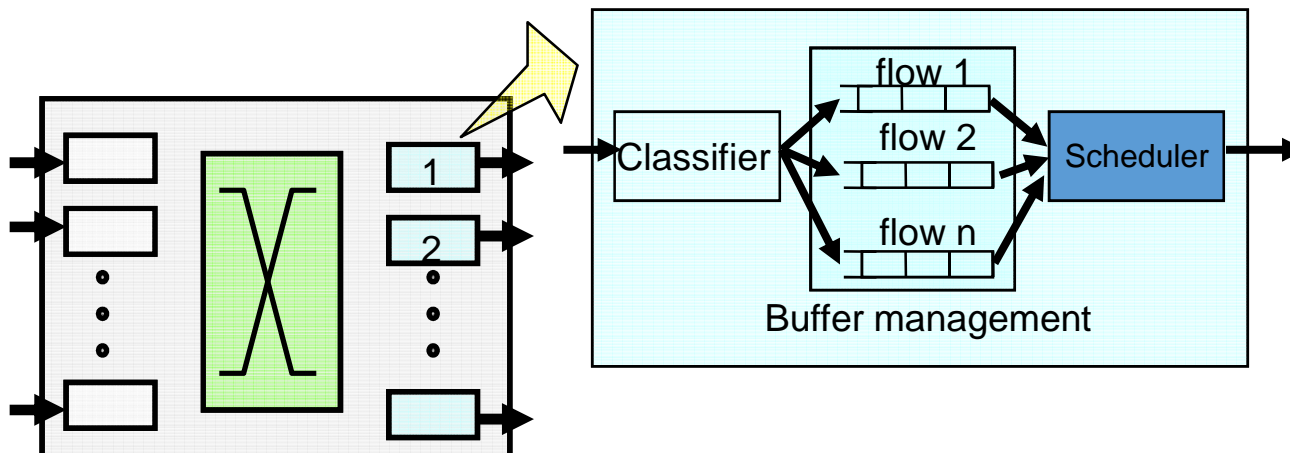
# Packet Classification

- Classify an IP packet based on a number of fields in the packet header, e.g.,
  - source/destination IP address (32 bits)
  - source/destination TCP port number (16 bits)
  - Type of service (TOS) byte (8 bits)
  - Type of protocol (8 bits)
- In general fields are specified by range
  - classification requires a multi-dimensional range search!



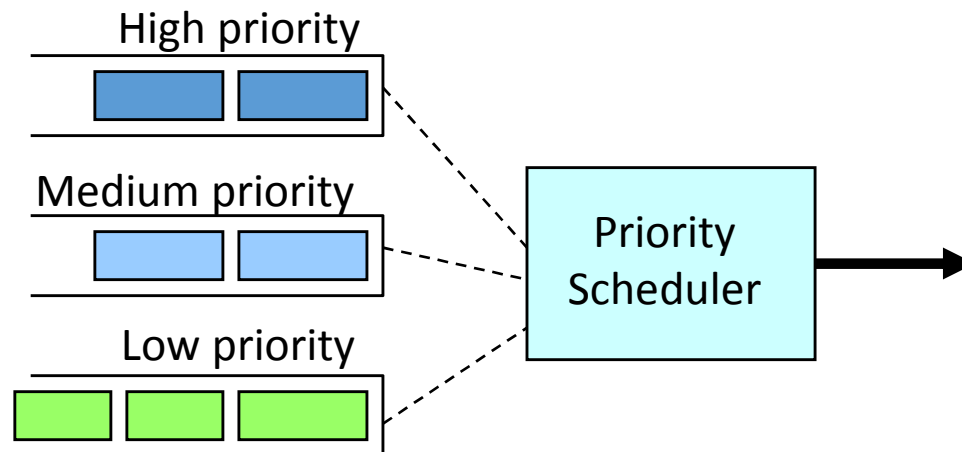
# Scheduler

- One queue per “flow”
- Scheduler decides when and from which queue to send a packet
- Goals of a scheduling algorithm:
  - Fast!
  - Depends on the policy being implemented (fairness, priority, *etc.*)



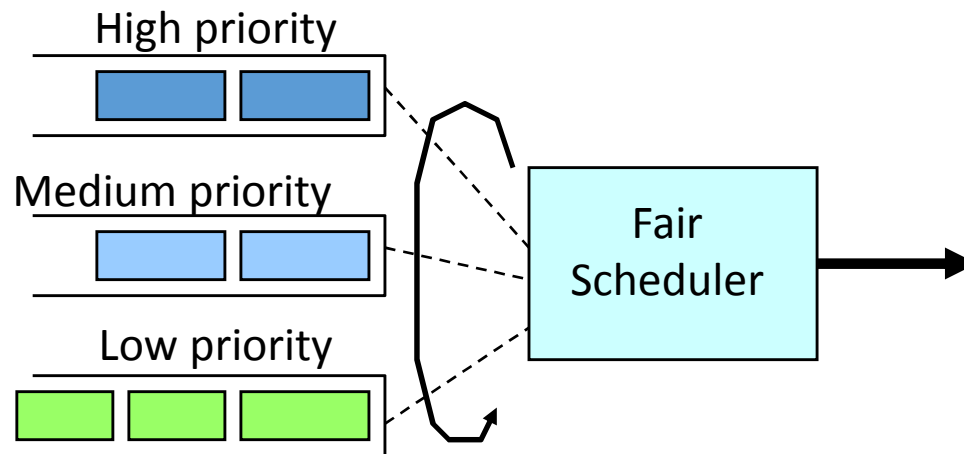
# Example: Priority Scheduler

- Priority scheduler: packets in the highest priority queue are always served **before** the packets in lower priority queues



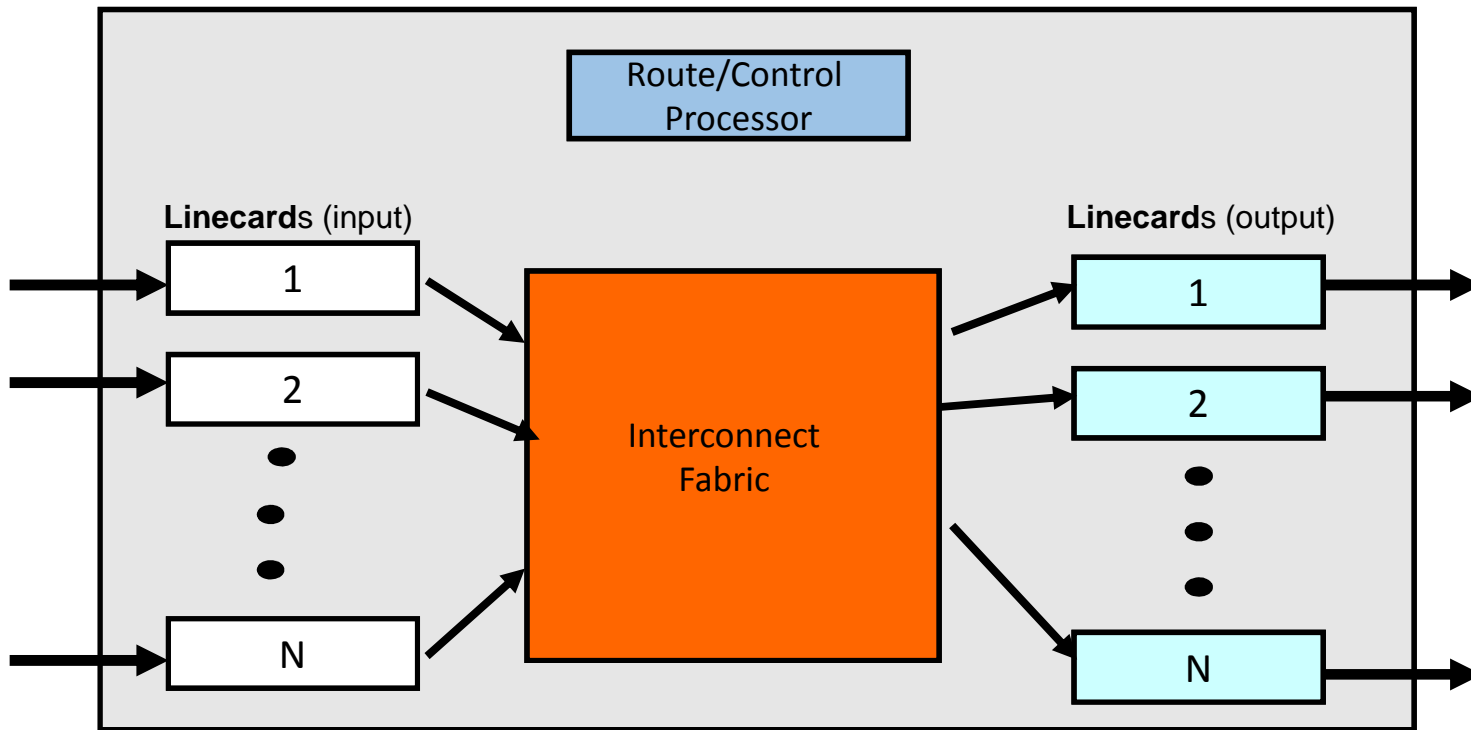
# Example: Round Robin Scheduler

- Round robin: packets are served from each queue in turn

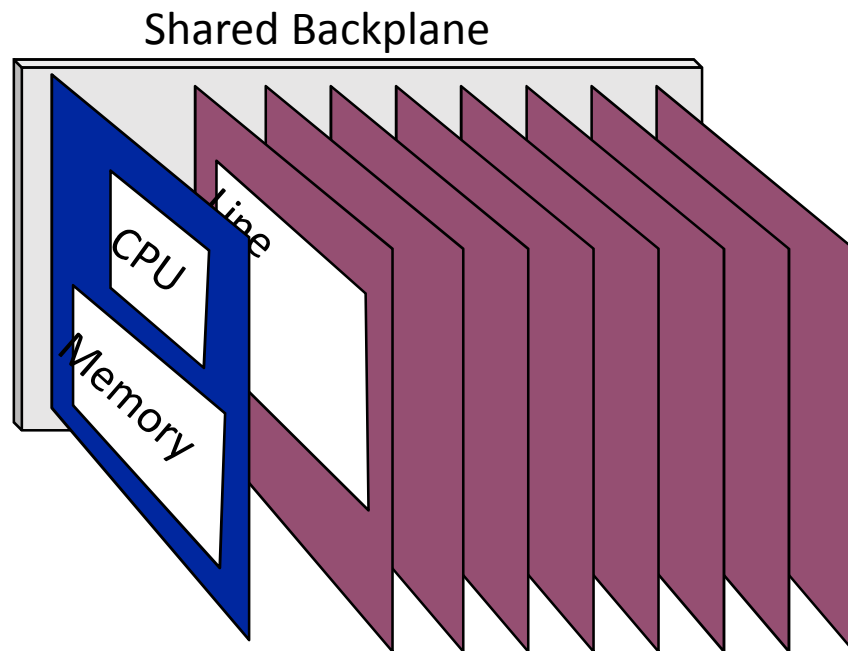




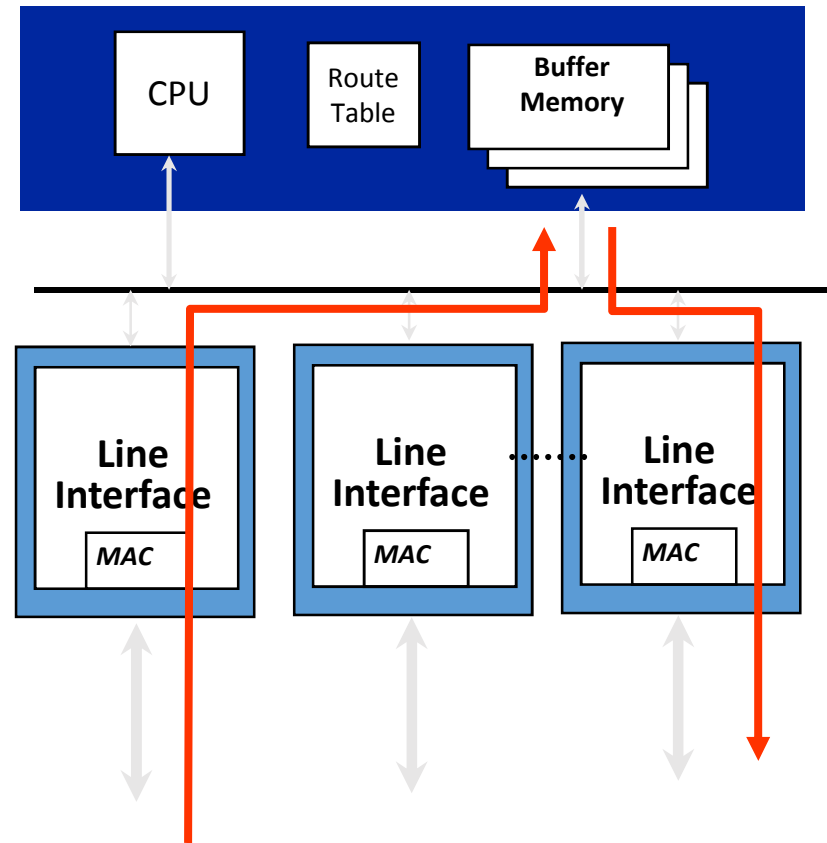
# Switching



# Shared Memory (1<sup>st</sup> Generation)



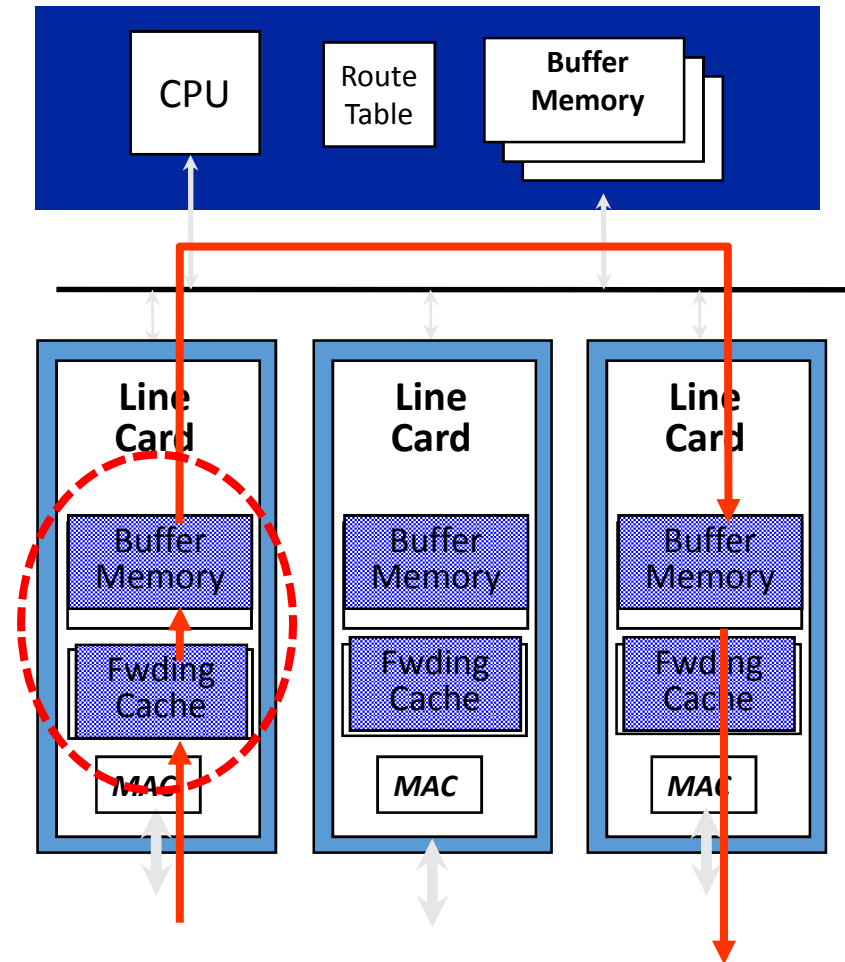
Limited by rate of shared memory



(\* Slide by Nick McKeown, Stanford Univ.)

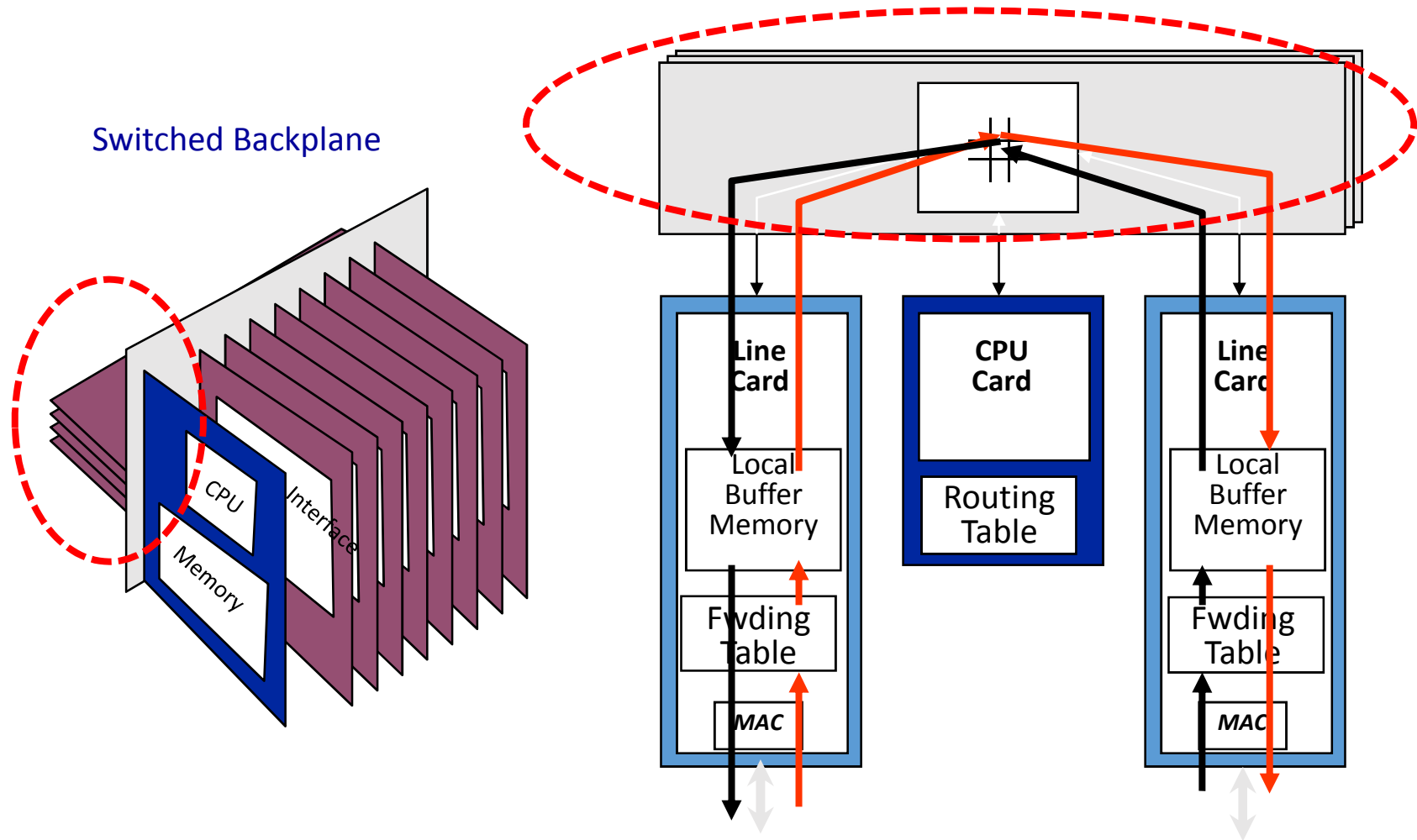
# Shared Bus (2<sup>nd</sup> Generation)

Limited by shared bus



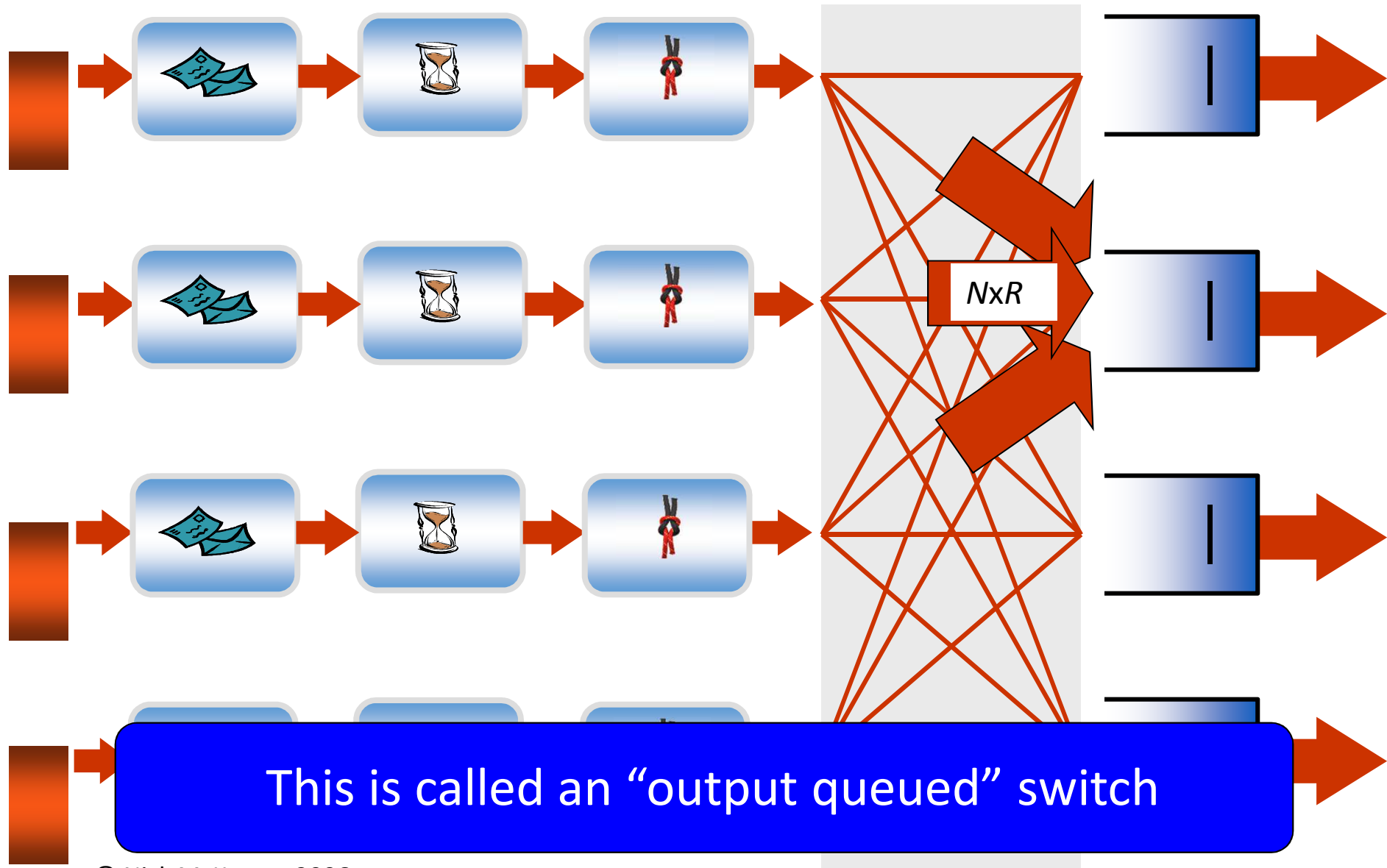
(\* Slide by Nick McKeown)

# Point-to-Point Switch (3<sup>rd</sup> Generation)

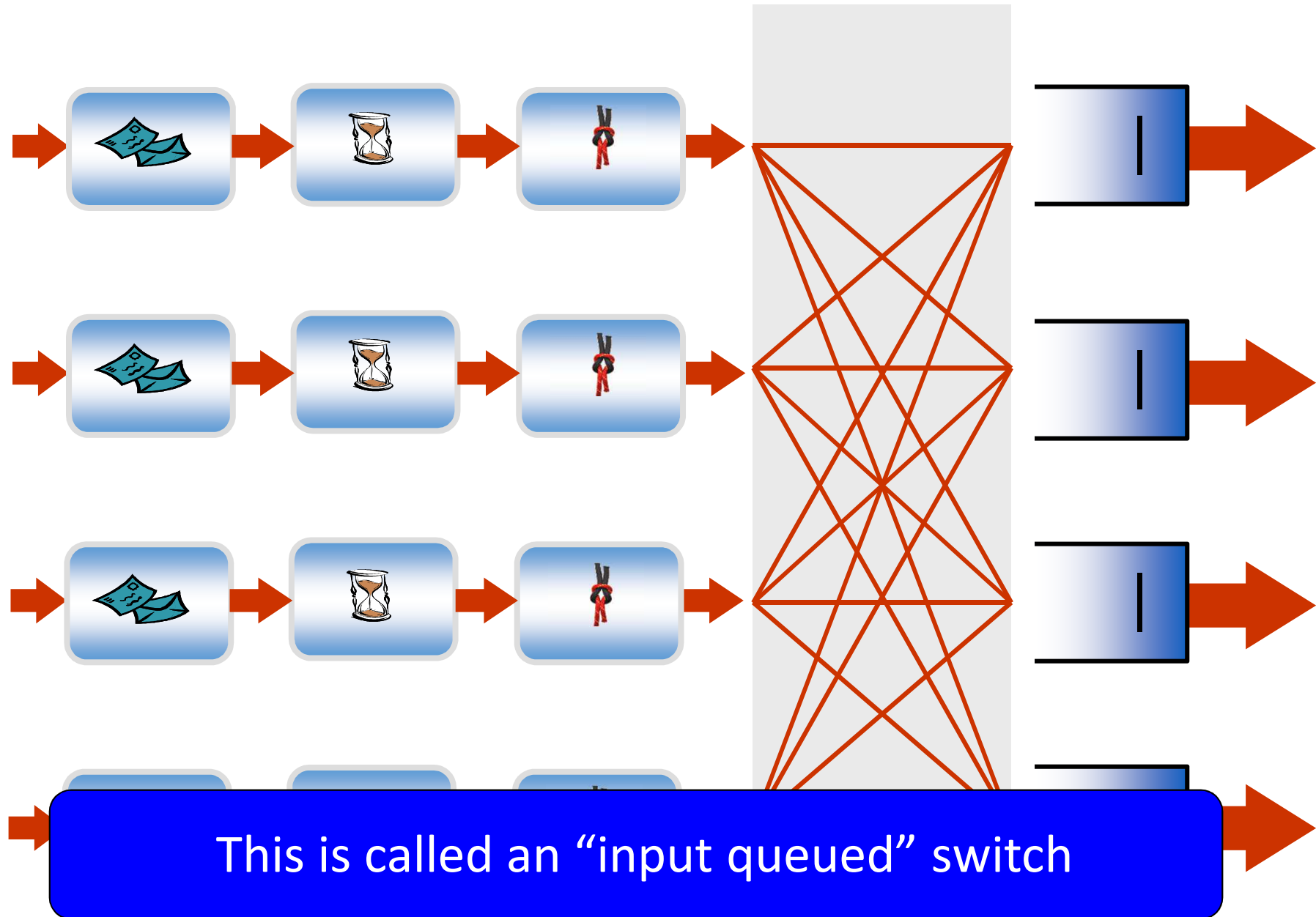


(\*Slide by Nick McKeown)

# 3<sup>rd</sup> Gen. Router: Switched Interconnects



# 3<sup>rd</sup> Gen. Router: Switched Interconnects

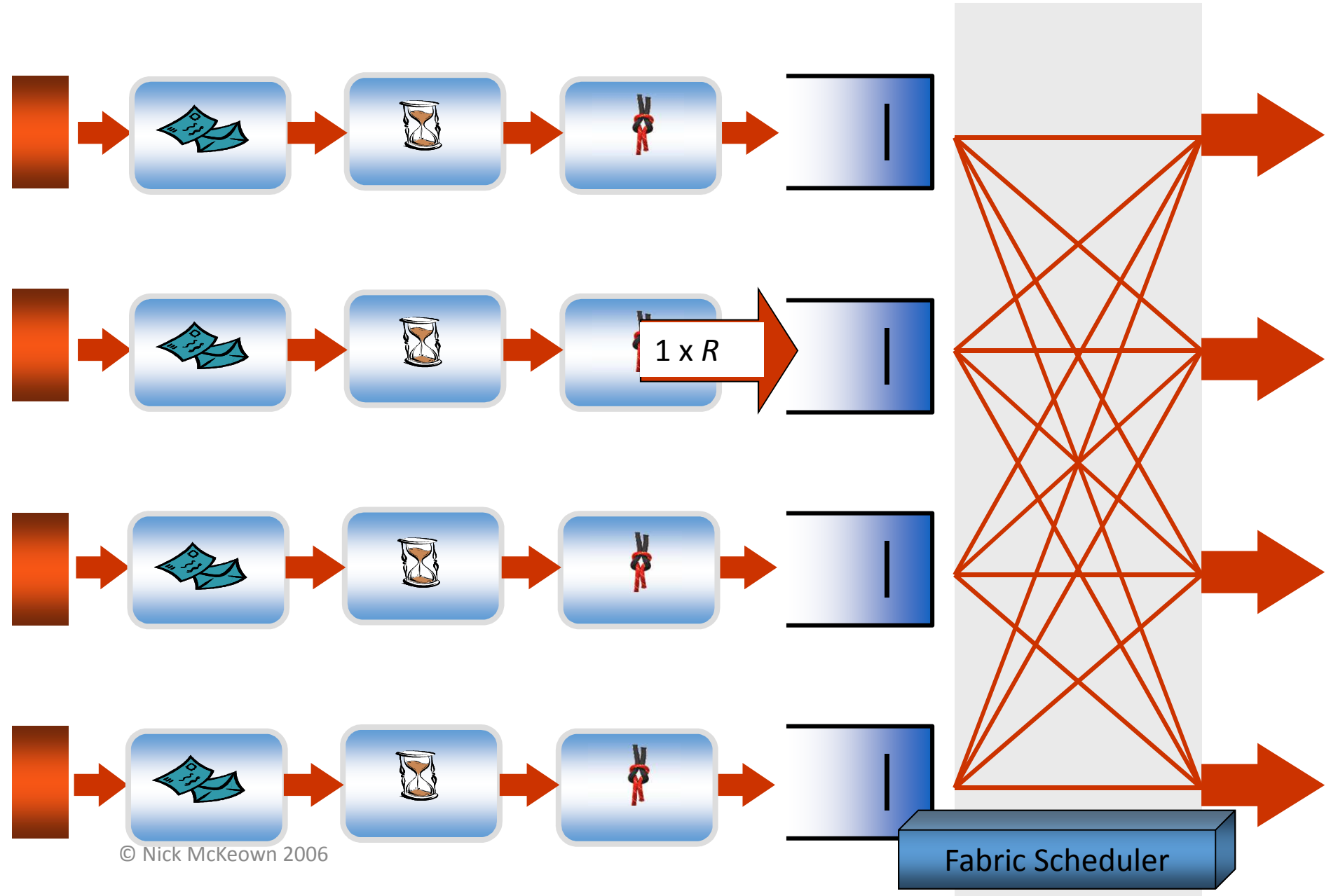


This is called an “input queued” switch

# Two challenges with input queuing

- 1) Need an internal fabric scheduler!

# 3<sup>rd</sup> Gen. Router: Switched Interconnects

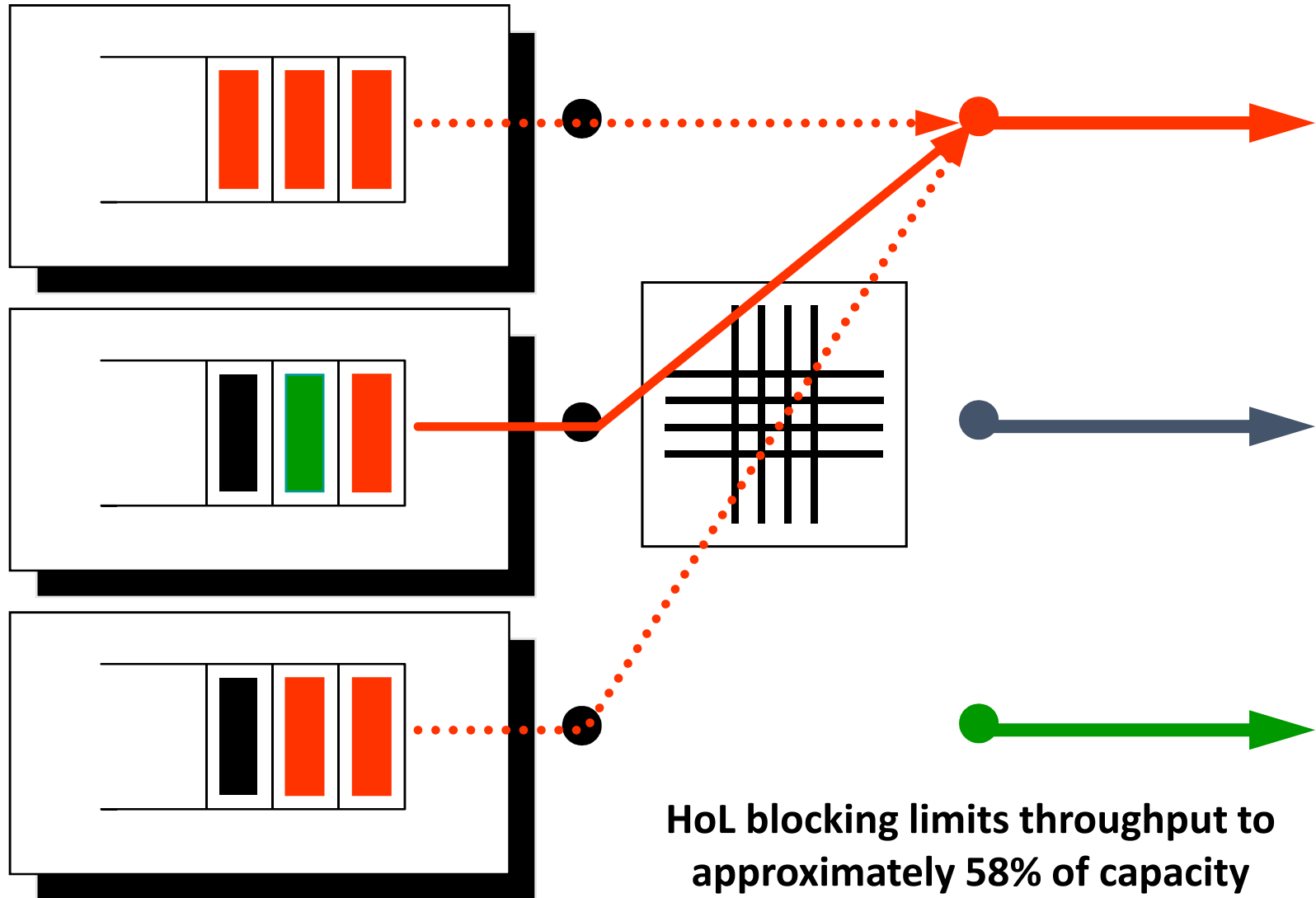




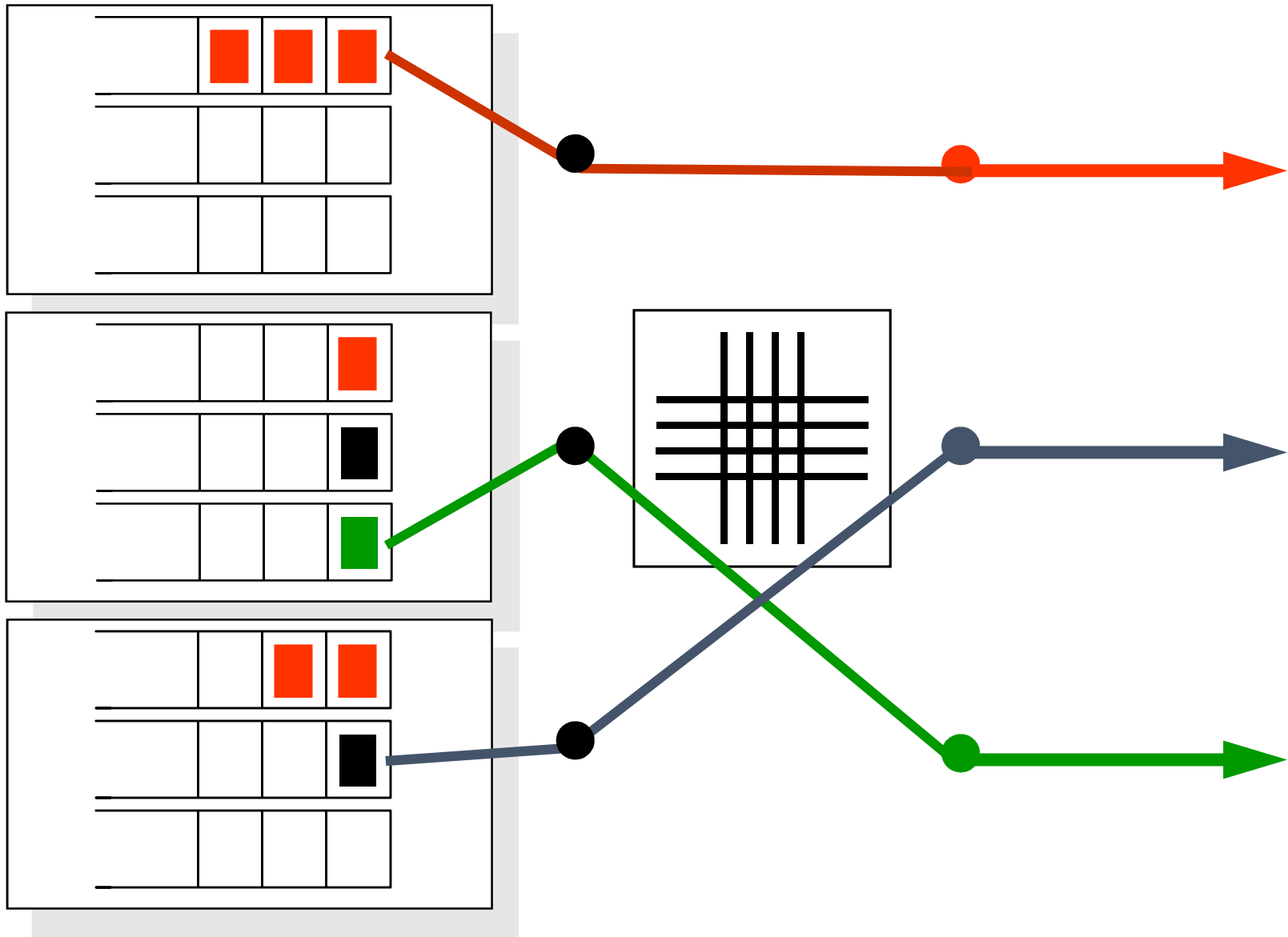
# Two challenges with input queuing

- 1) Need an internal fabric scheduler!
- 2) Must avoid “head-of-line” blocking

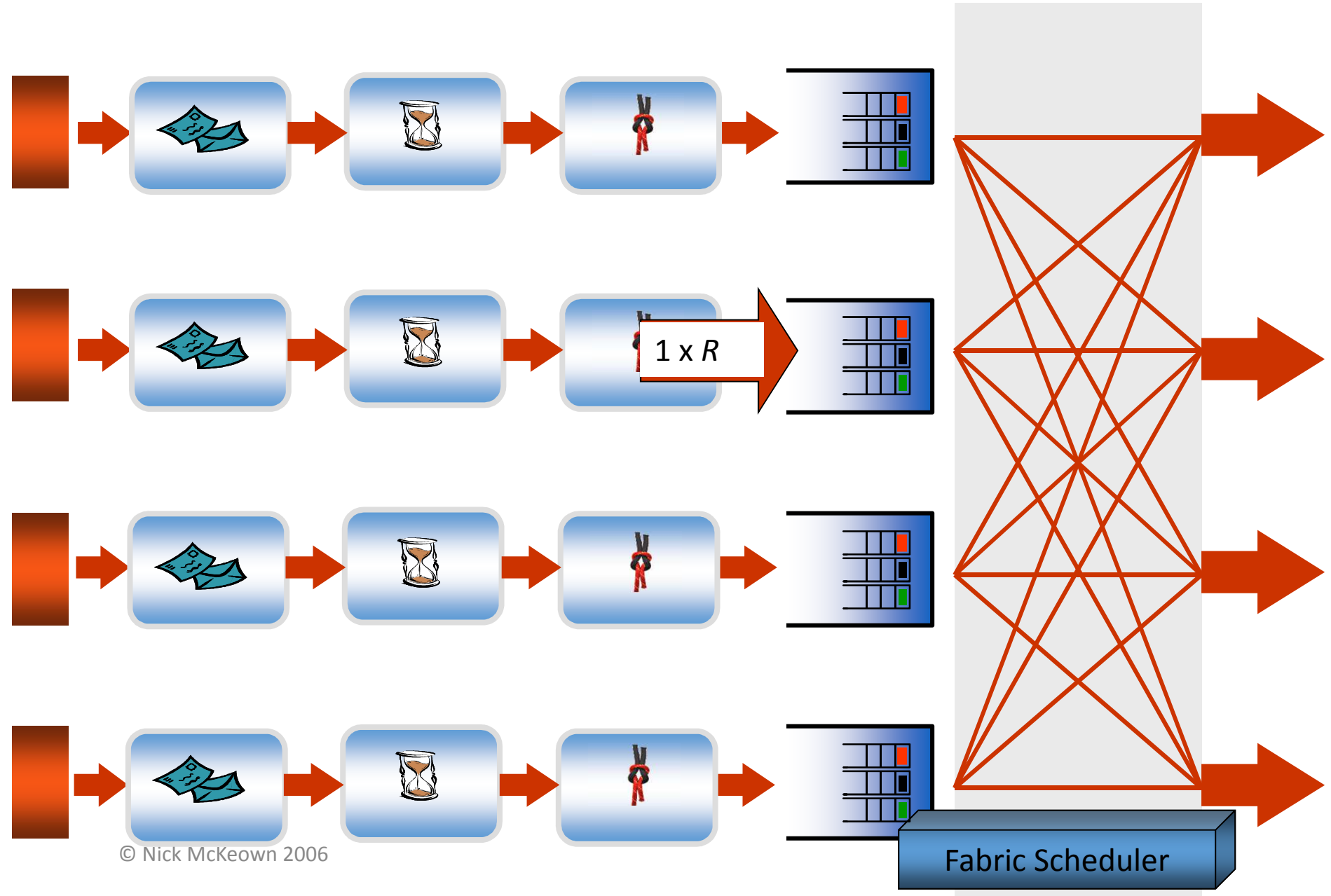
# Head of Line Blocking



# “Virtual Output Queues”

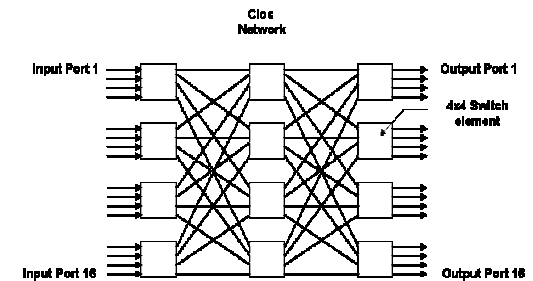


# 3<sup>rd</sup> Gen. Router: Switched Interconnects



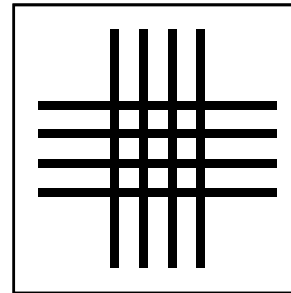
# Reality is more complicated

- Commercial (high-speed) routers use
  - combination of input and output queuing
  - complex multi-stage switching topologies (Clos, Benes)
  - distributed, multi-stage schedulers (for scalability)
- We'll consider one simpler context
  - de-facto architecture for a long time and still used in lower-speed routers



# Scheduling

- Context
  - crossbar fabric
  - centralized scheduler

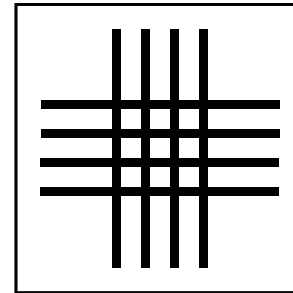


- Goals
  - **Work conserving (100% throughput)**
  - **Fast**

# Scheduling

- Context

- crossbar fabric
- centralized scheduler



- Goal: 100% throughput, fast

- optimal solution: maximum matching on a bipartite graph
- problem: too slow
- **practical solution: a good maximal matching**
- multiple fast algorithms exist for computing a good and fair maximal matching [PIM, iSlip]

# In Summary

- IP header formats
  - not as boring as one might imagine
- IP routers
  - core building block of the infrastructure
  - needs simple, fast implementations for longest-prefix matching, multi-dimensional search, switch and output scheduling