IP Routers

CS 168, Fall 2014

Kay Ousterhout (standing in for Sylvia Ratnasamy)

http://inst.eecs.berkeley.edu/~cs168/

Material thanks to Ion Stoica, Scott Shenker, Jennifer Rexford, Nick McKeown, and many other colleagues

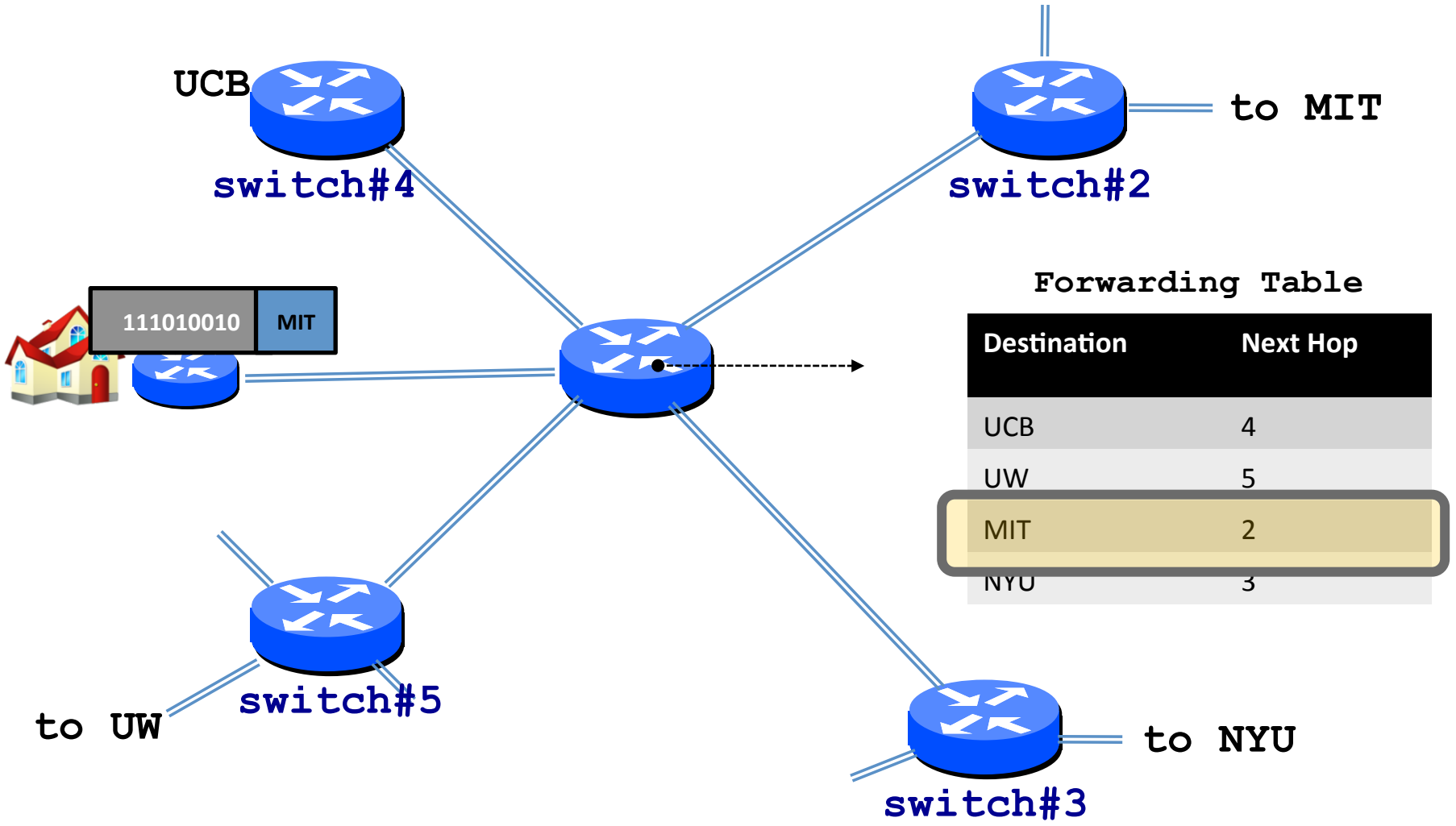
Context

- Control plane
  - How to route traffic to each possible destination
  - Jointly computed using BGP
- Data plane
  - Necessary fields in IP header of each packet
- Today: How IP routers forward packets
  - Focus on data plane
- Today (maybe): Transport layer

IP Routers

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

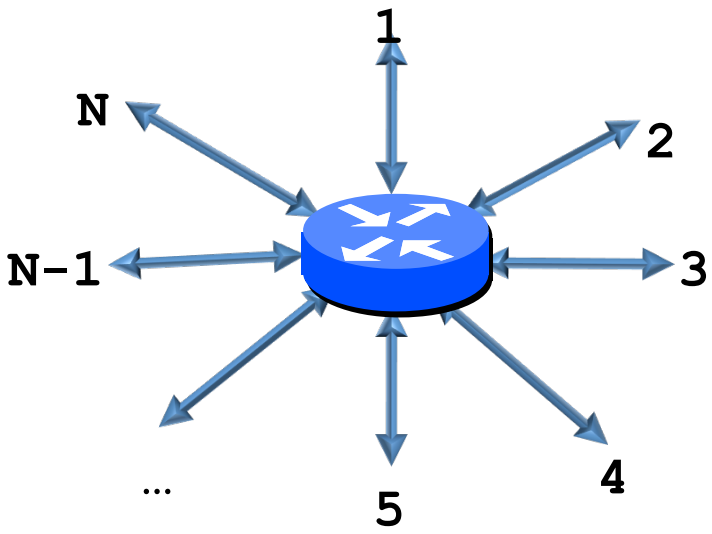
Lecture #4: Routers Forward Packets



Forwarding Table

Destination	Next Hop
UCB	4
UW	5
MIT	2
NYU	3

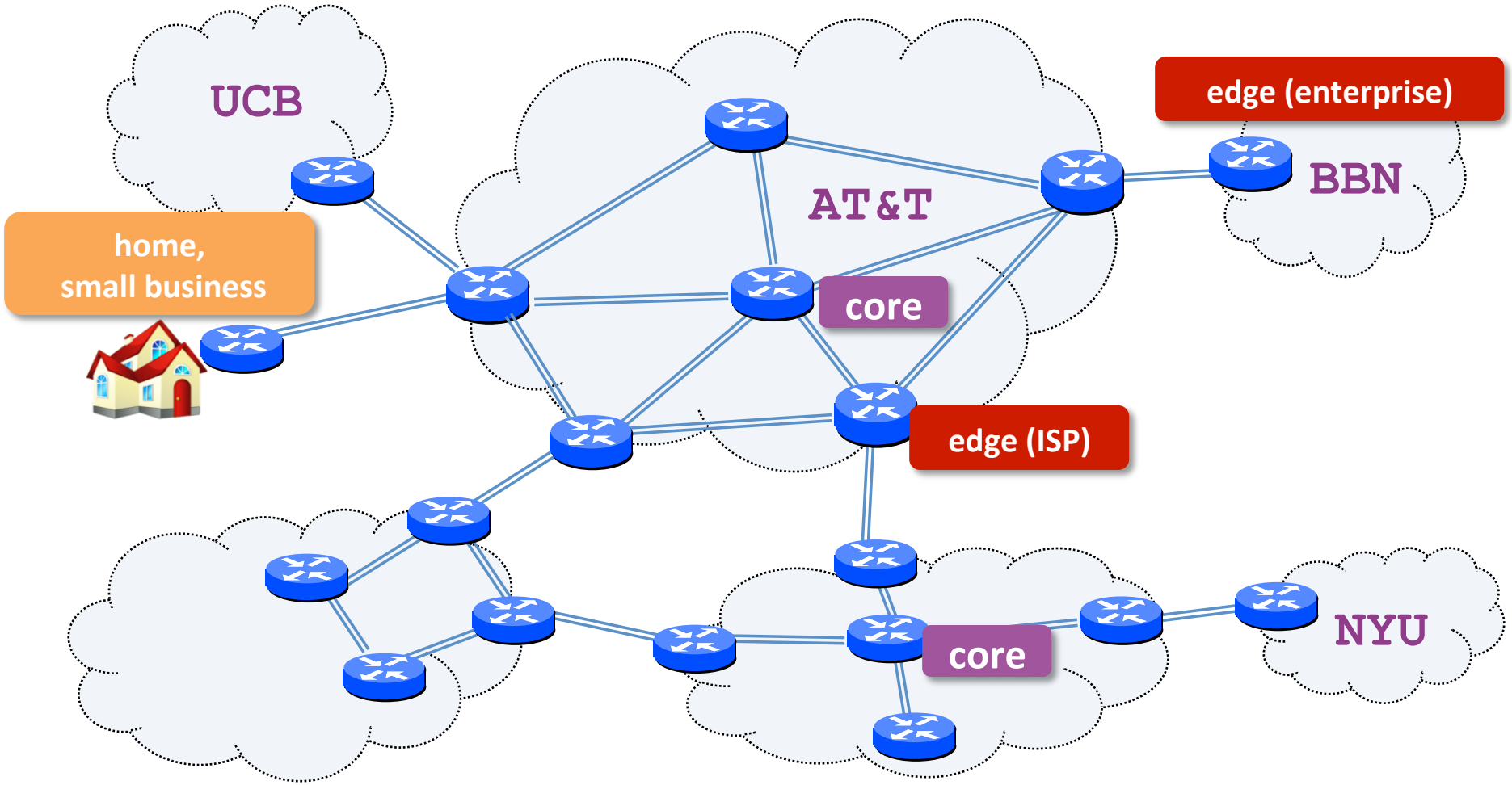
Router definitions



R *bits/sec*

- N = number of external router "ports"
- R = speed ("line rate") of a port
- Router capacity = N x R

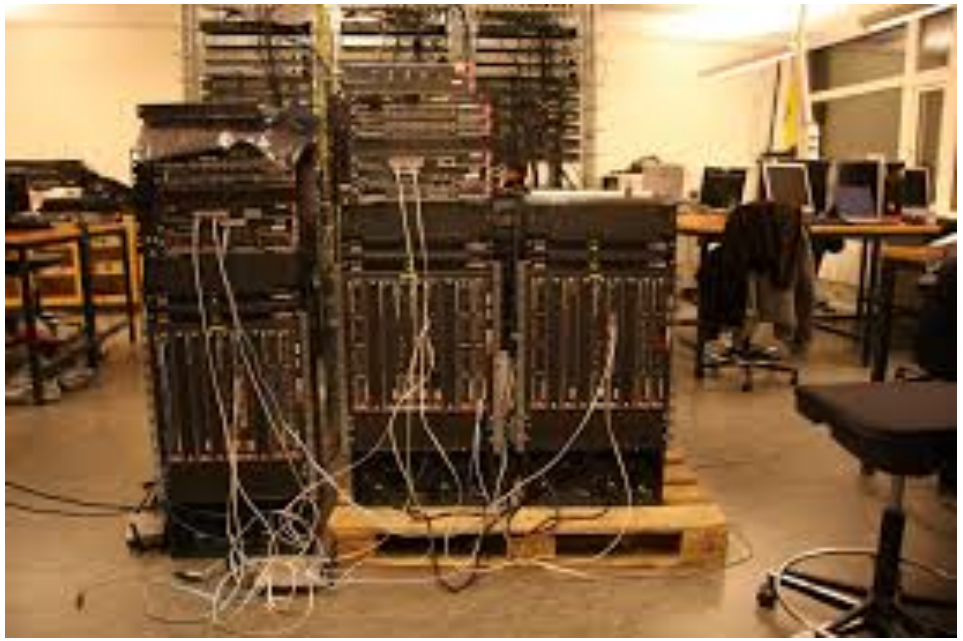
Networks and routers



Examples of routers (core)

Cisco CRS

- R=10/40/100 Gbps
- NR = 922 Tbps
- Netflix: 0.7GB per hour (1.5Mb/s)
- ~600 million concurrent Netflix users



72 racks, >1MW

Examples of routers (edge)

Cisco ASR

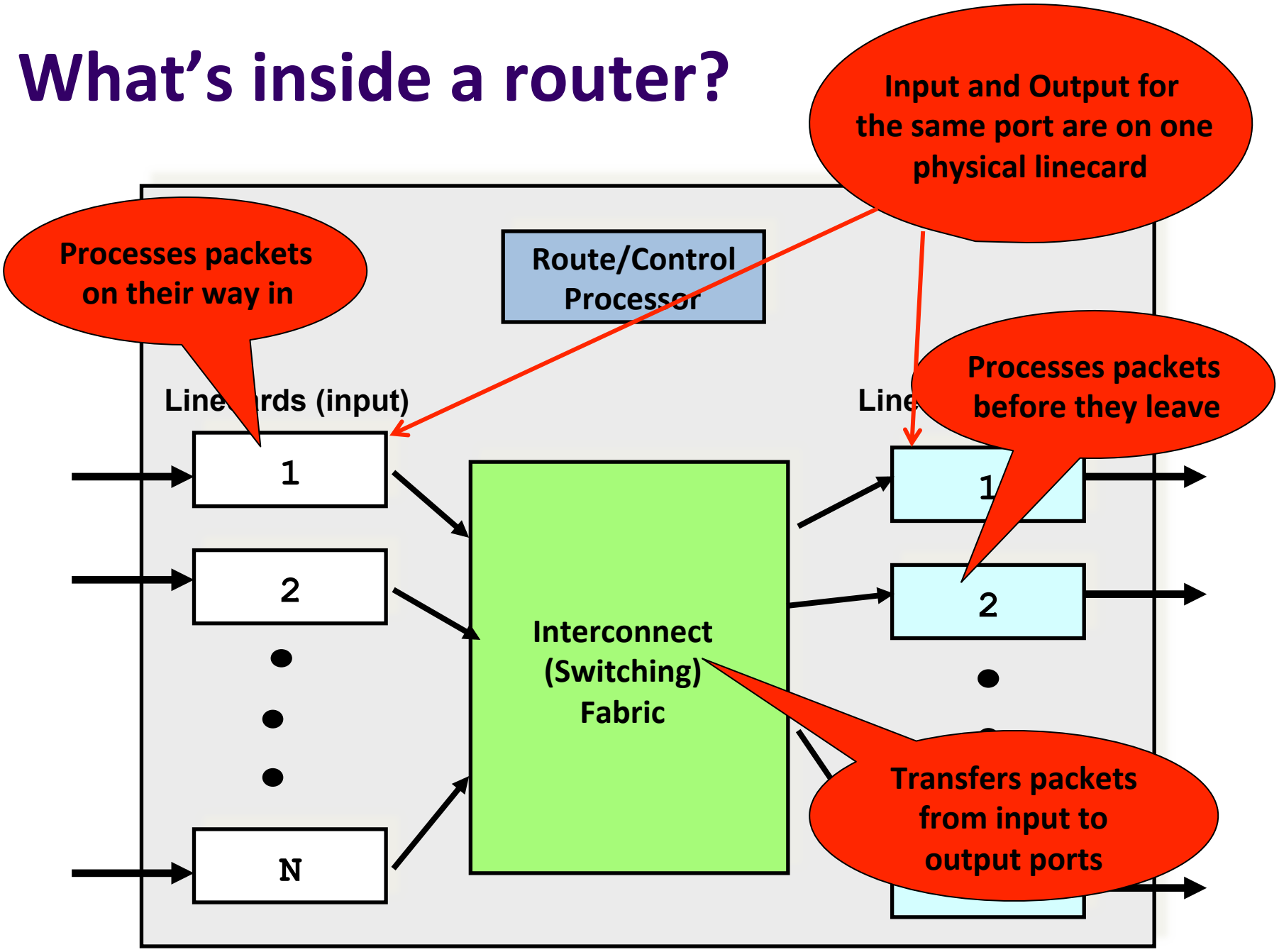
- R=1/10/40 Gbps
- NR = 120 Gbps

Examples of routers (small business)

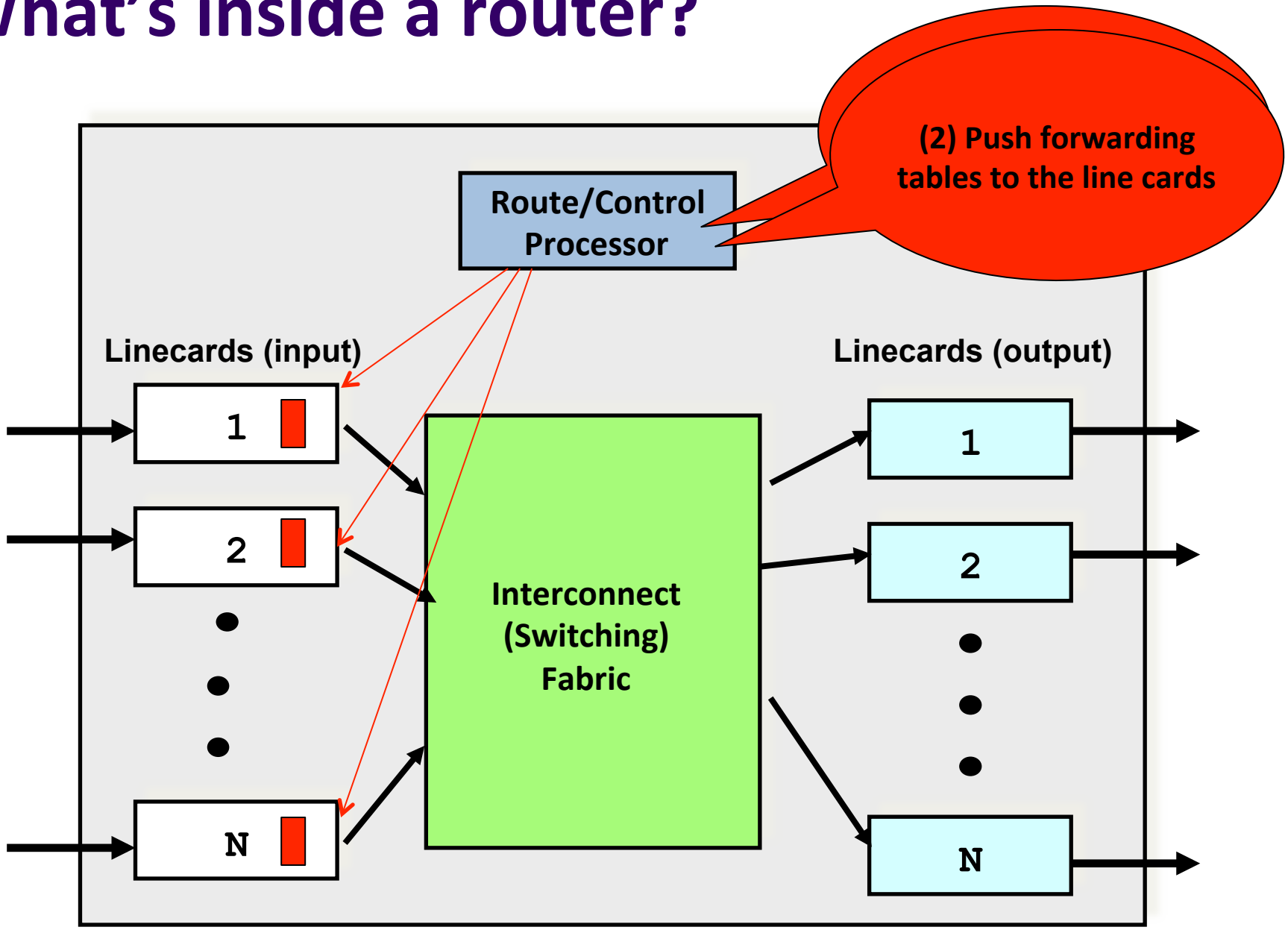
Cisco 3945E

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

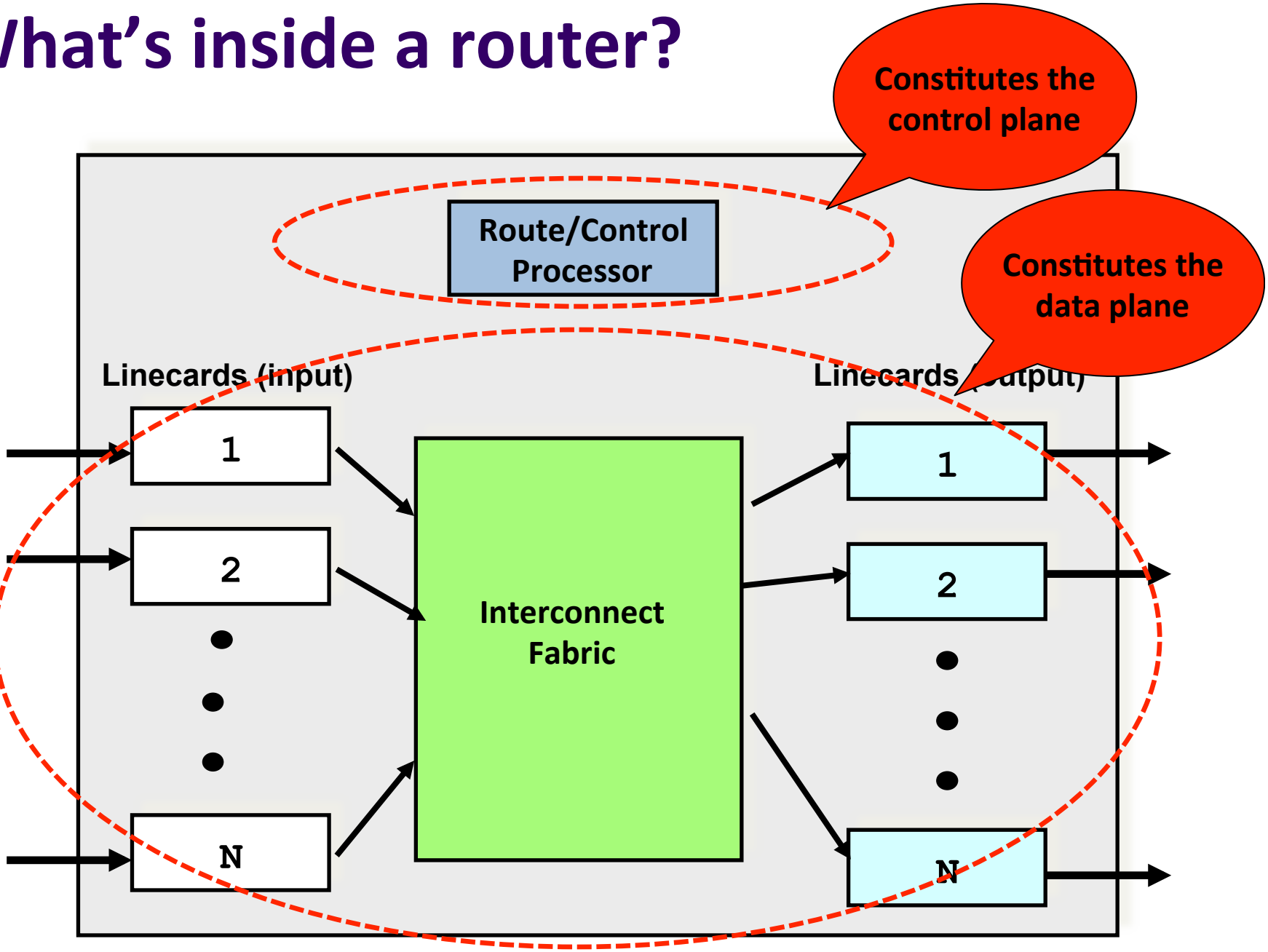
What's inside a router?



What's inside a router?

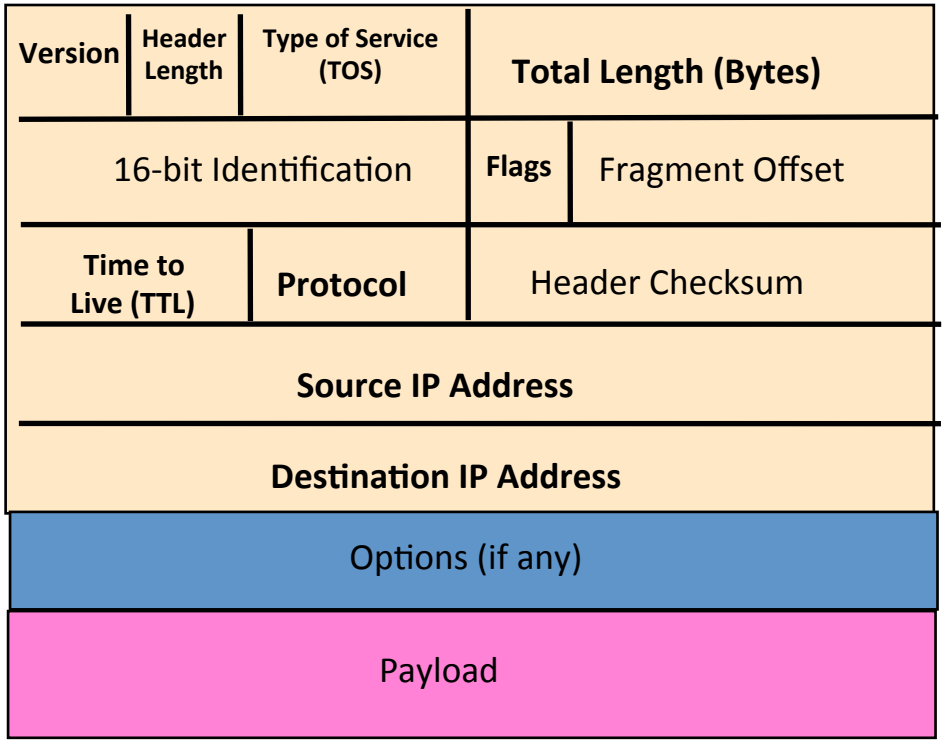


What's inside a router?



Input Linecards

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

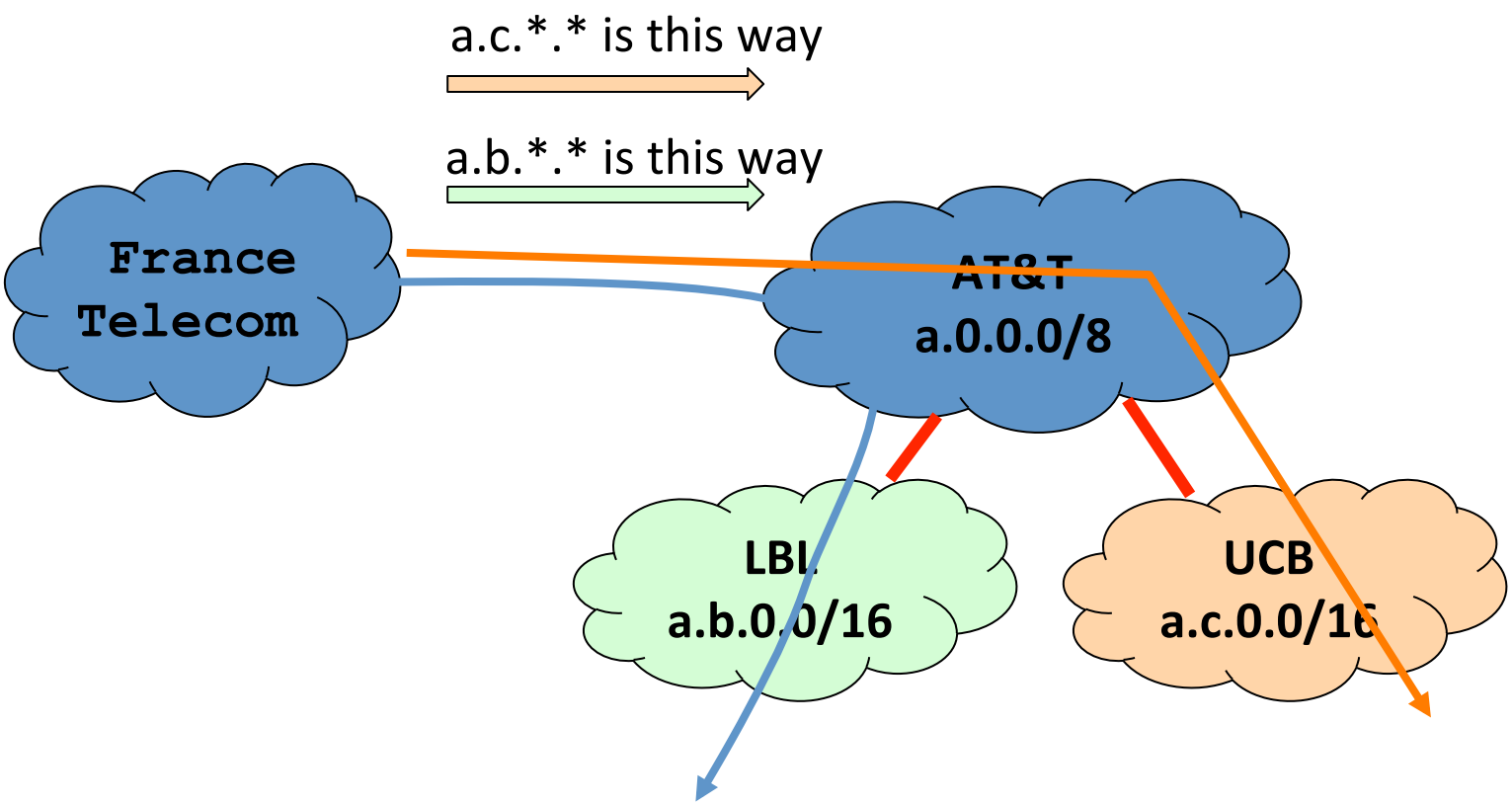


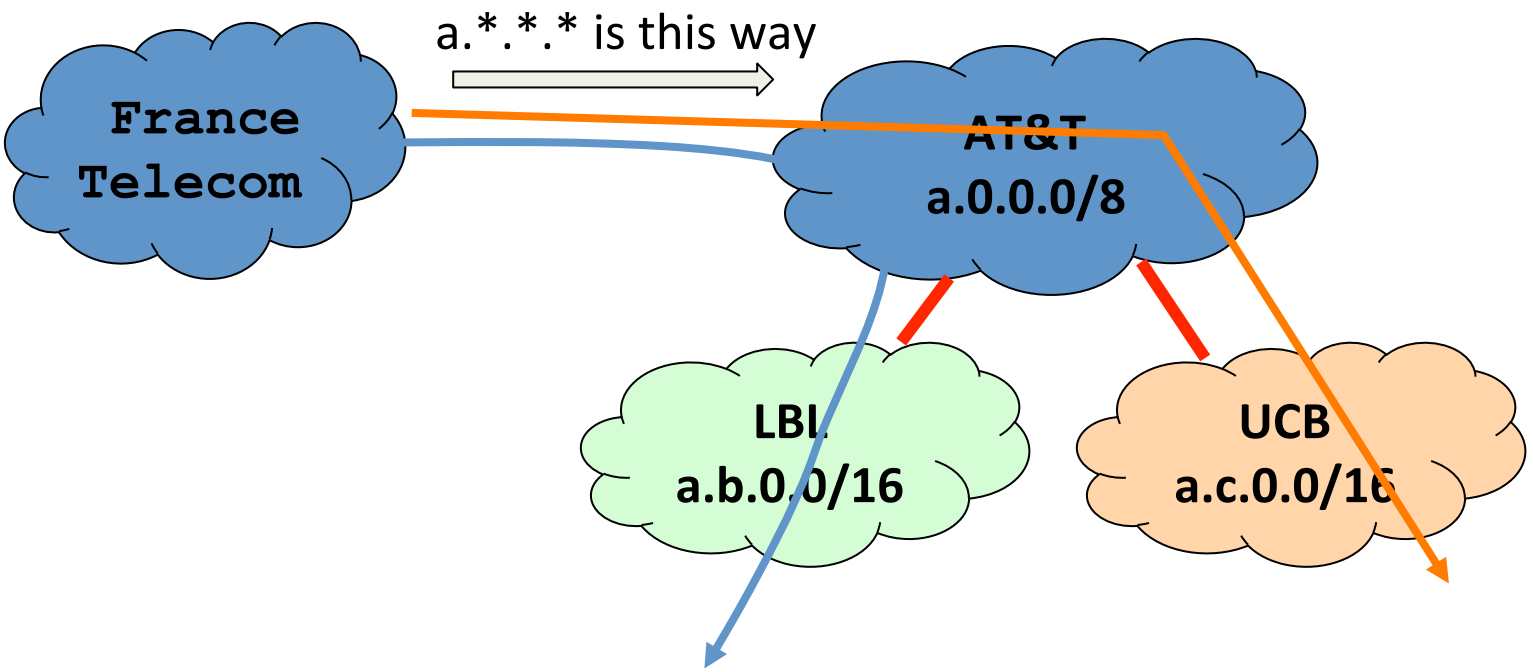
Input Linecards

- 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

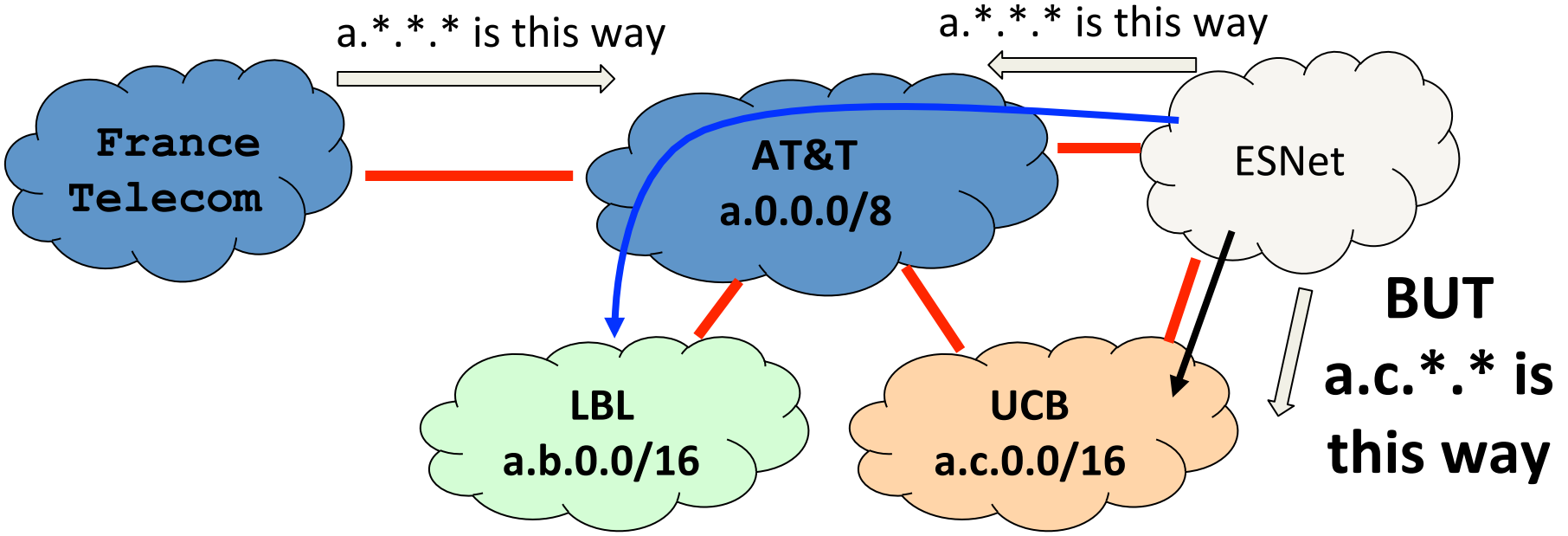
Looking up the output port

- One entry for each address → 4 billion entries!
- For scalability, addresses are **aggregated**


a.c.*.* is this way
a.b.*.* is this way
France Telecom
AT&T
a.0.0.0/8
LBL
a.b.0.0/16
UCB
a.c.0.0/16

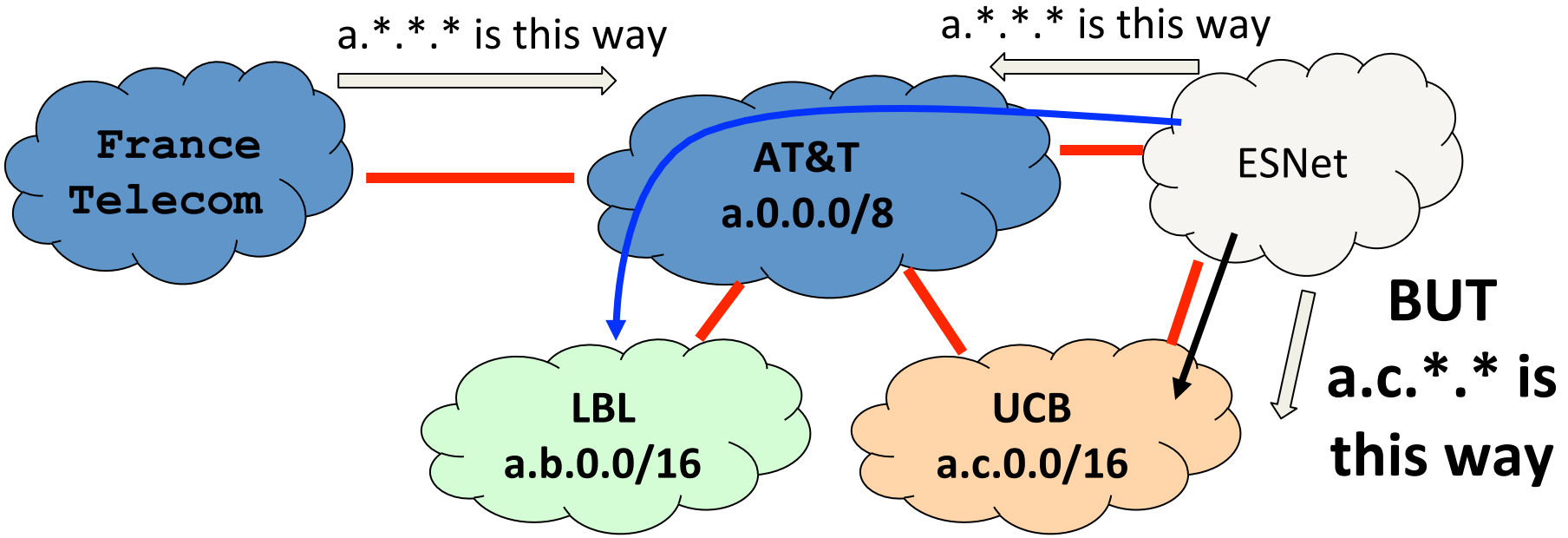

a.*.*.* is this way
France Telecom
AT&T
a.0.0.0/8
LBL
a.b.0.0/16
UCB
a.c.0.0/16

But aggregation is imperfect…



ESNet must maintain routing entries for both a.*.*.* and a.c.*.*

Find the *longest* prefix that matches

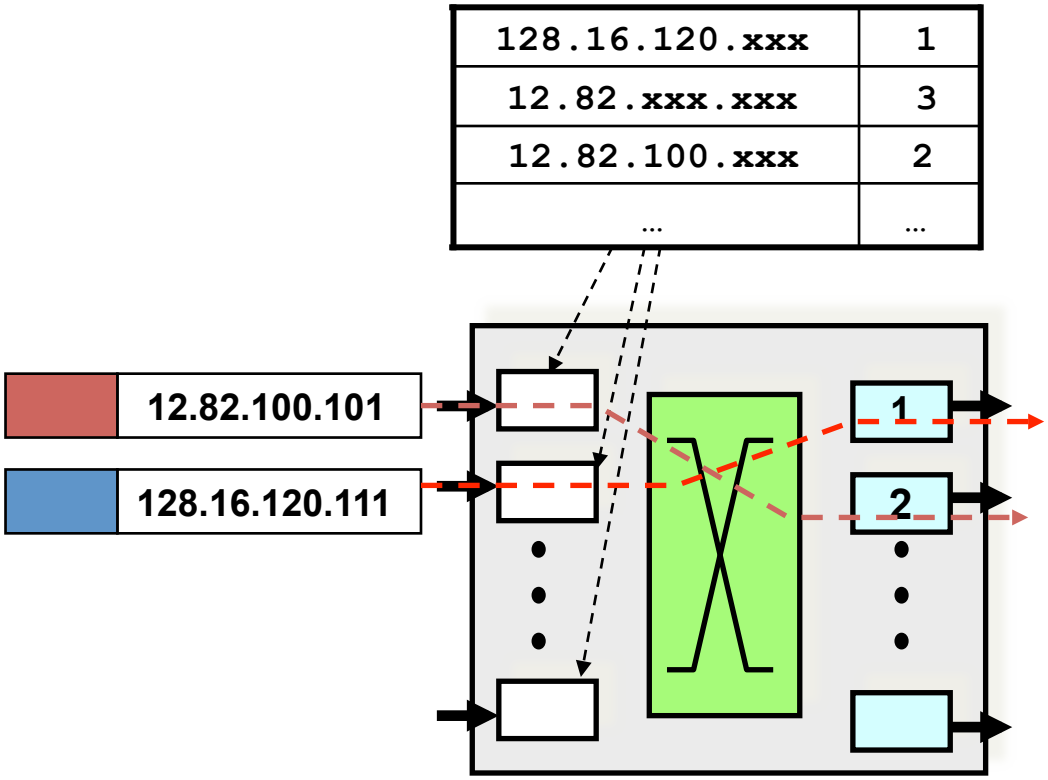


ESNet's Forwarding Table

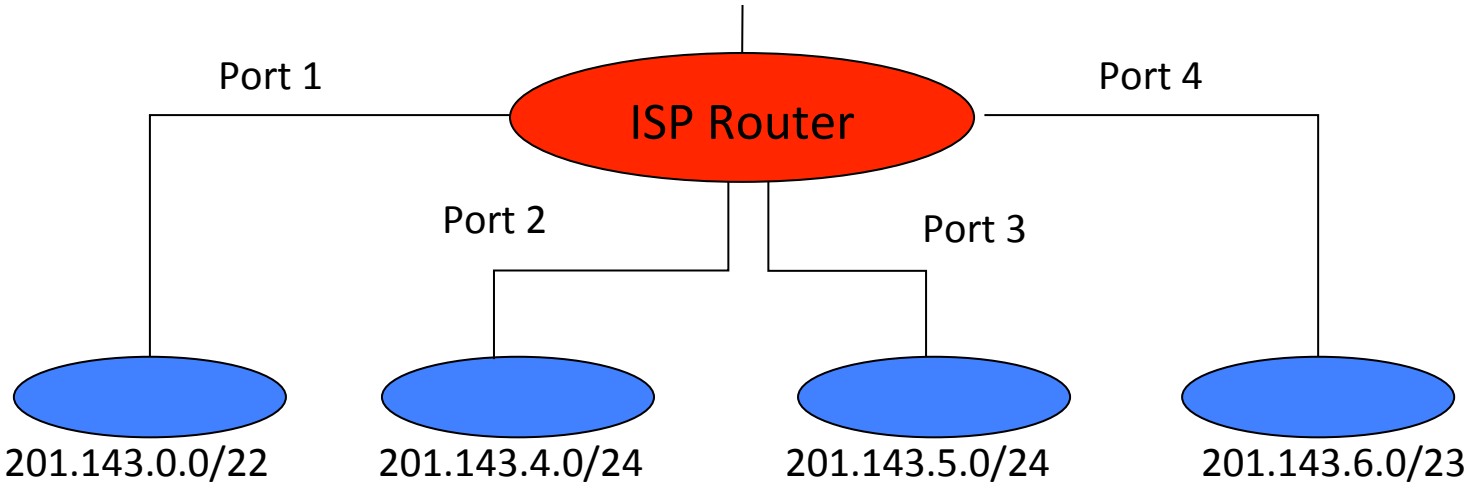
Destination	Next Hop
a.*.*.*	at&t
a.c.*.*	ucb
…	…

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



Example #1: 4 Prefixes, 4 Ports



Prefix	Port
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 a match

- Incoming packet destination: 201.143.7.0

Prefix	Port
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 a match: convert to binary

- Incoming packet destination: 201.143.7.210

11001001	10001111	00000111	11010010

Routing table

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–	––––––––

Finding a match: convert to binary

- Incoming packet destination: 201.143.7.210

11001001	10001111	00000111	11010010

Routing table

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	00000110	––––––––

Finding a match: convert to binary

- Incoming packet destination: 201.143.7.210

11001001	10001111	00000111	11010010

Routing table

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–	––––––––

Finding a match: convert to binary

- Incoming packet destination: 201.143.7.210

11001001	10001111	00000111	11010010

Routing table

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–	––––––––

Longest prefix matching

- Incoming packet destination: 201.143.7.210

11001001	10001111	00000111	11010010

Routing table

201.143.0.0/22

11001001	10001111	000000––	––––––––

201.143.4.0/24

11001001	10001111	00000100	––––––––

201.143.7.0/25

11001001	10001111	00000111	0–––––––

201.143.6.0/23

11001001	10001111	00000110	––––––––

NOT Check an address against all destination prefixes and select the prefix it matches with on the most bits

Finding Match Efficiently

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

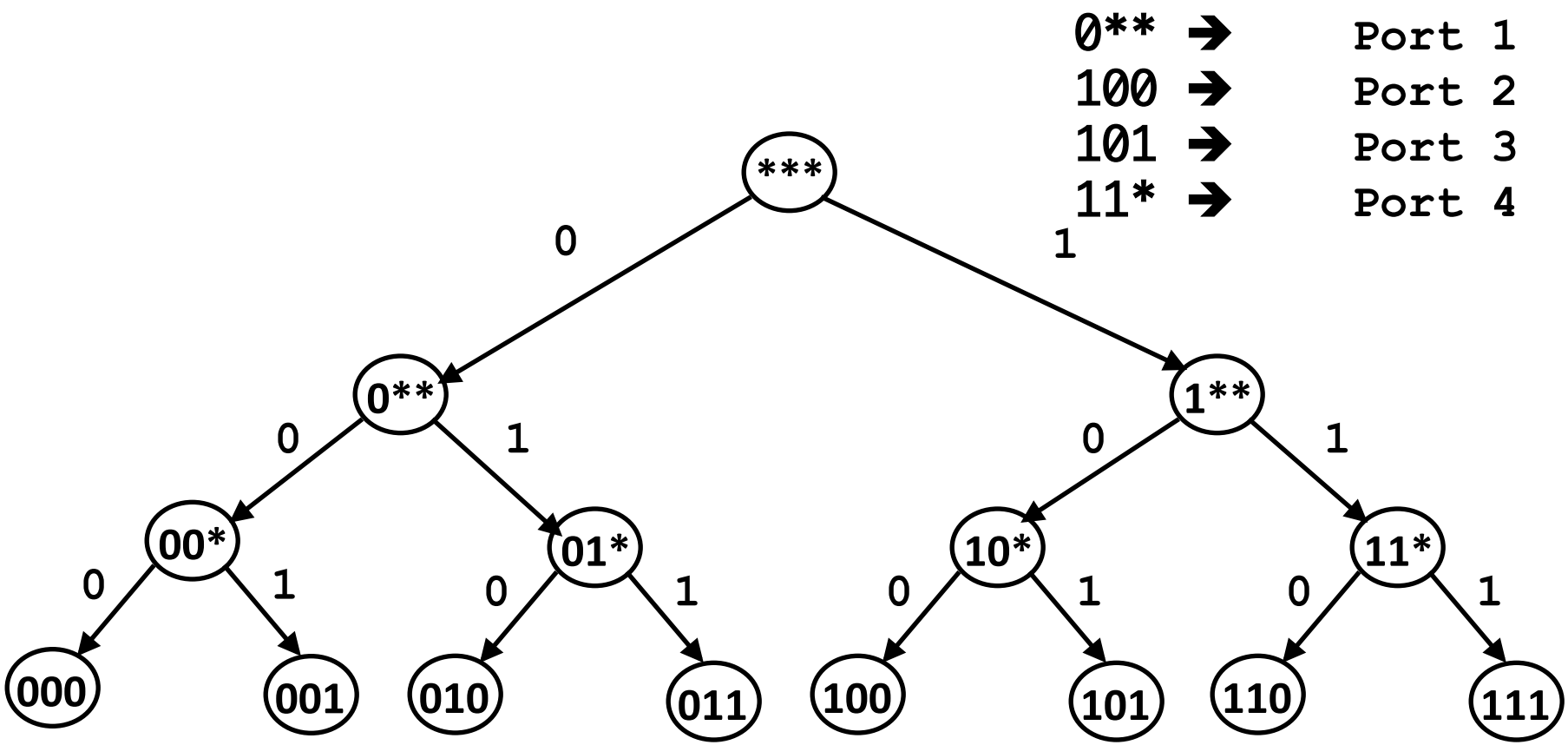
Prefix	Port
11001001100011110000 0*********	1
11001001100011110000 100********	2
11001001100011110000 101********	3
11001001100011110000 11********	4

Consider four three-bit prefixes

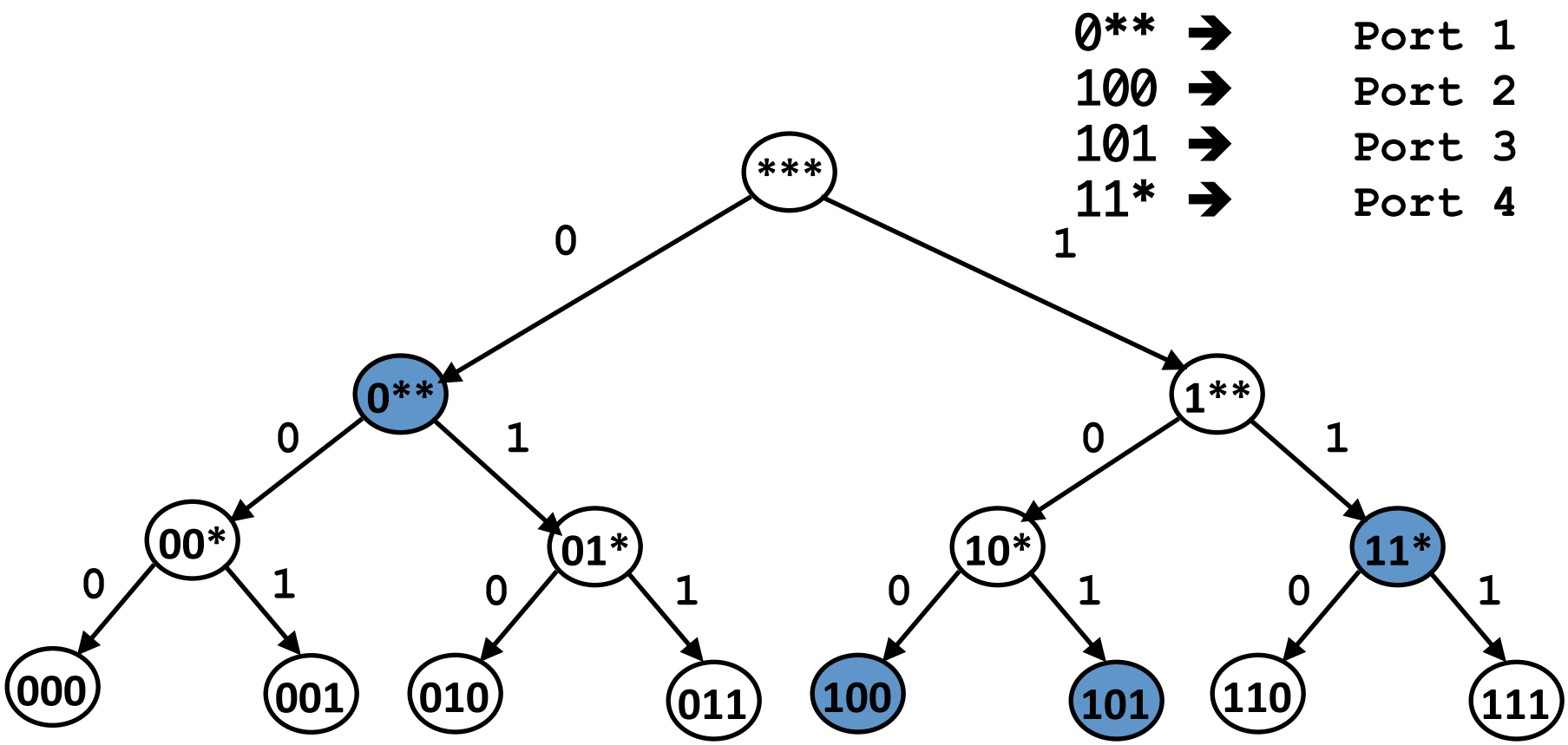
- Just focusing on the bits where all the action is….

- `0**` → Port 1
- `100` → Port 2
- `101` → Port 3
- `11*` → Port 4

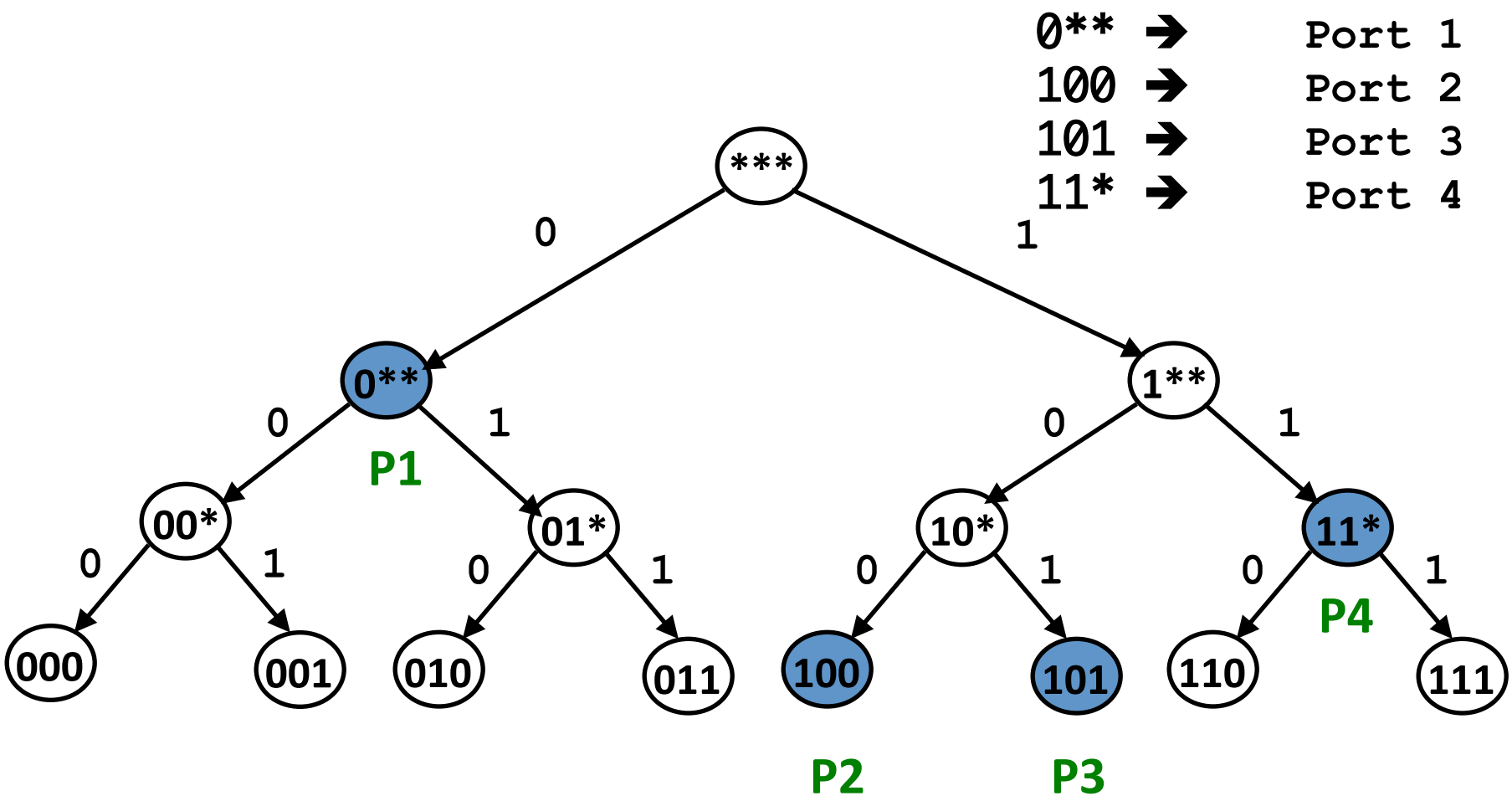
Tree Structure



Walk Tree: Stop at Prefix Entries



Walk Tree: Stop at Prefix Entries

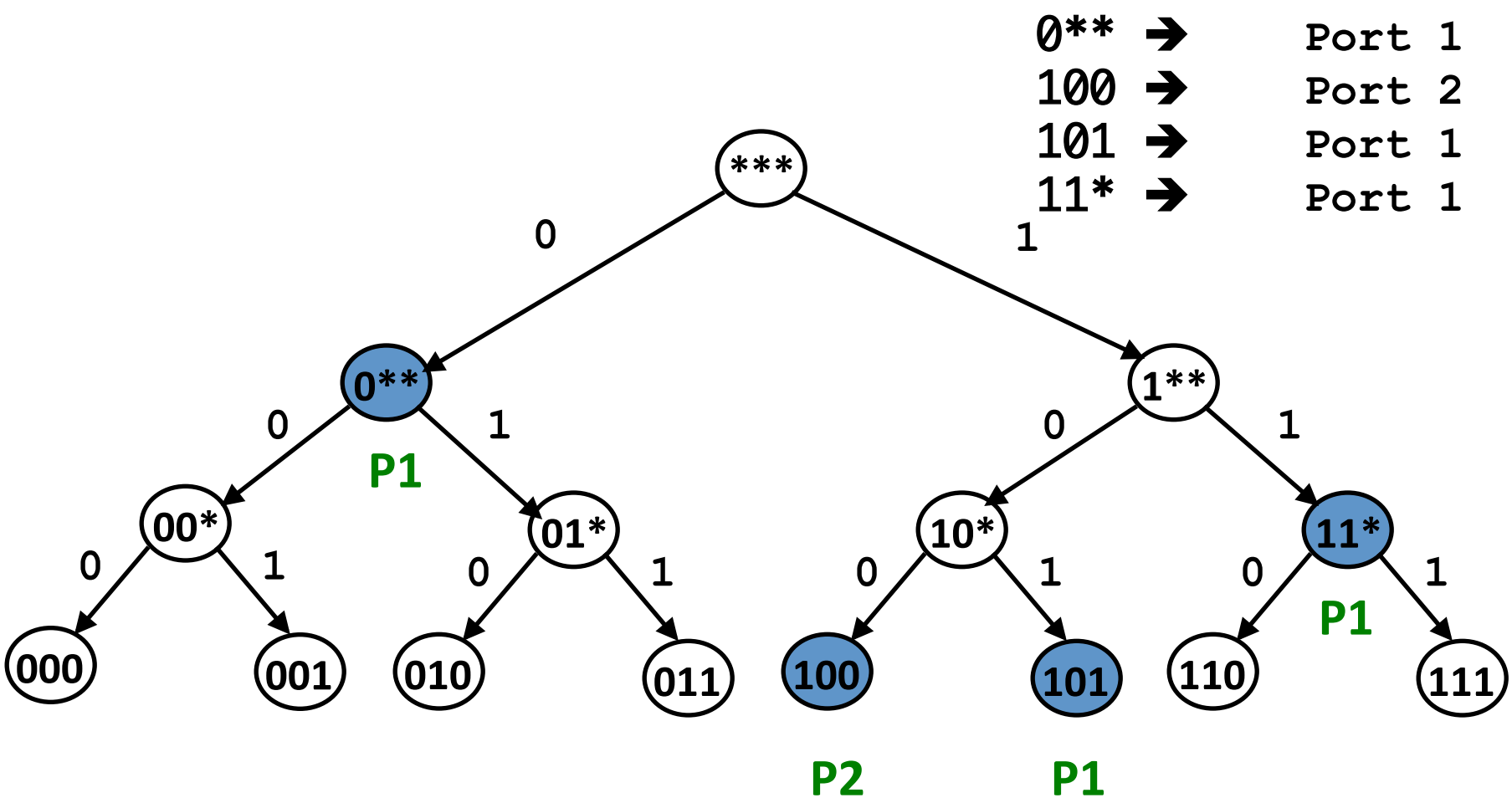


Slightly Different Example

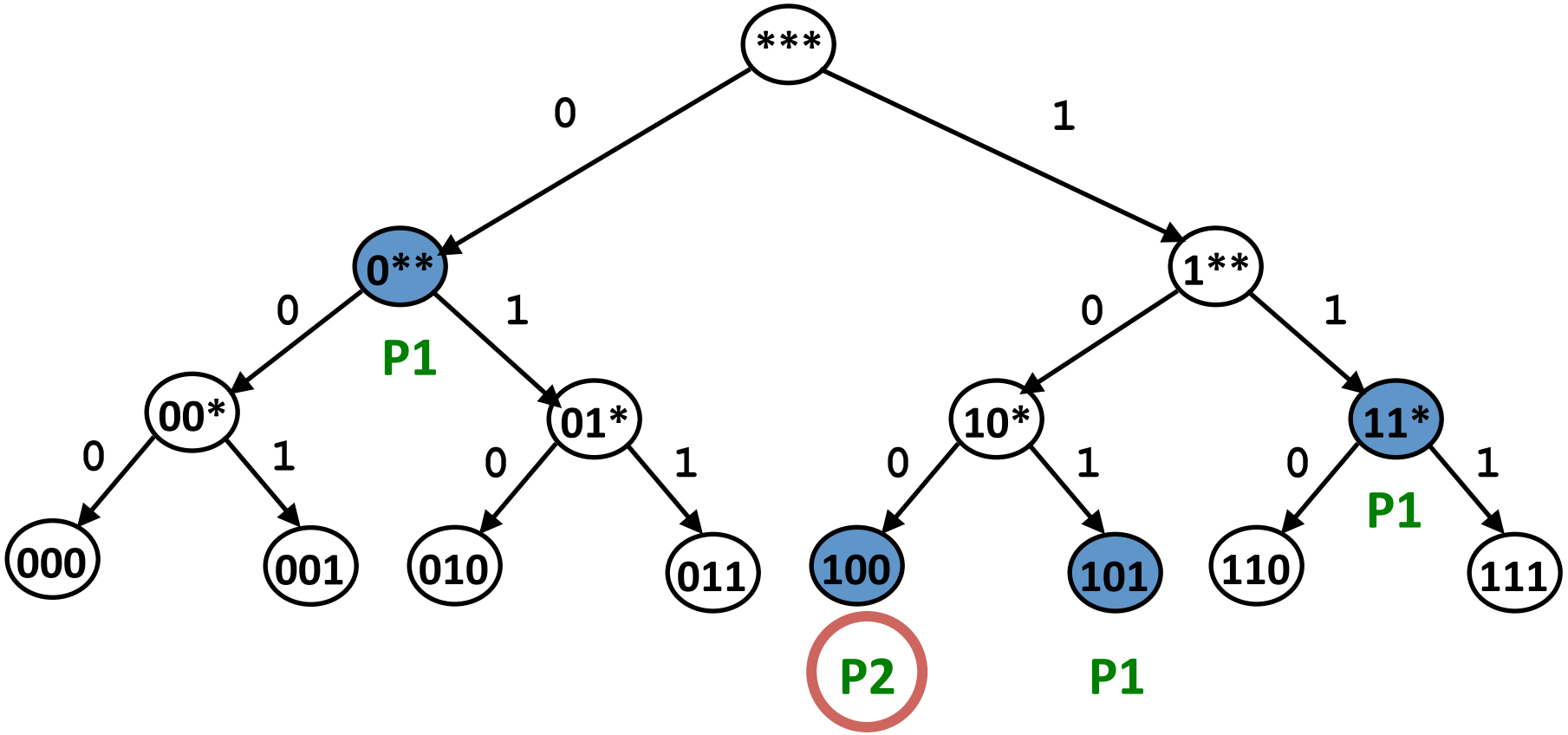
- Several of the unique prefixes go to same port

- `0**` → Port 1
- `100` → Port 2
- `101` → Port 1
- `11*` → Port 1

Prefix Tree


0** → Port 1
100 → Port 2
101 → Port 1
11* → Port 1
***
0
1
0**
1**
0
1
0
1
P1
00*
01*
10*
11*
0
1
0
1
0
1
0
1
P1
000
001
010
011
100
101
110
111
P2
P1

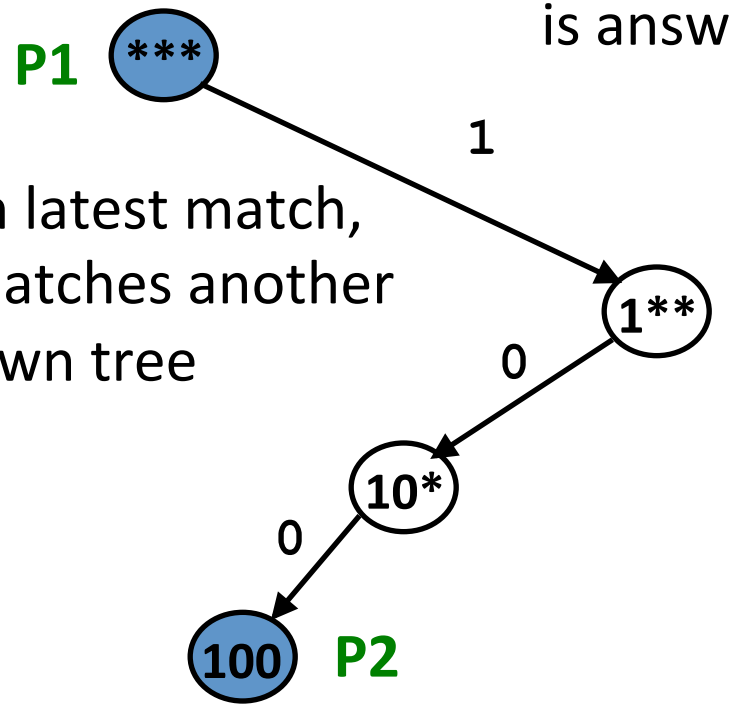
More Compact Representation



More Compact Representation

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

Record port associated with latest match, and only over-ride when it matches another prefix during walk down tree



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

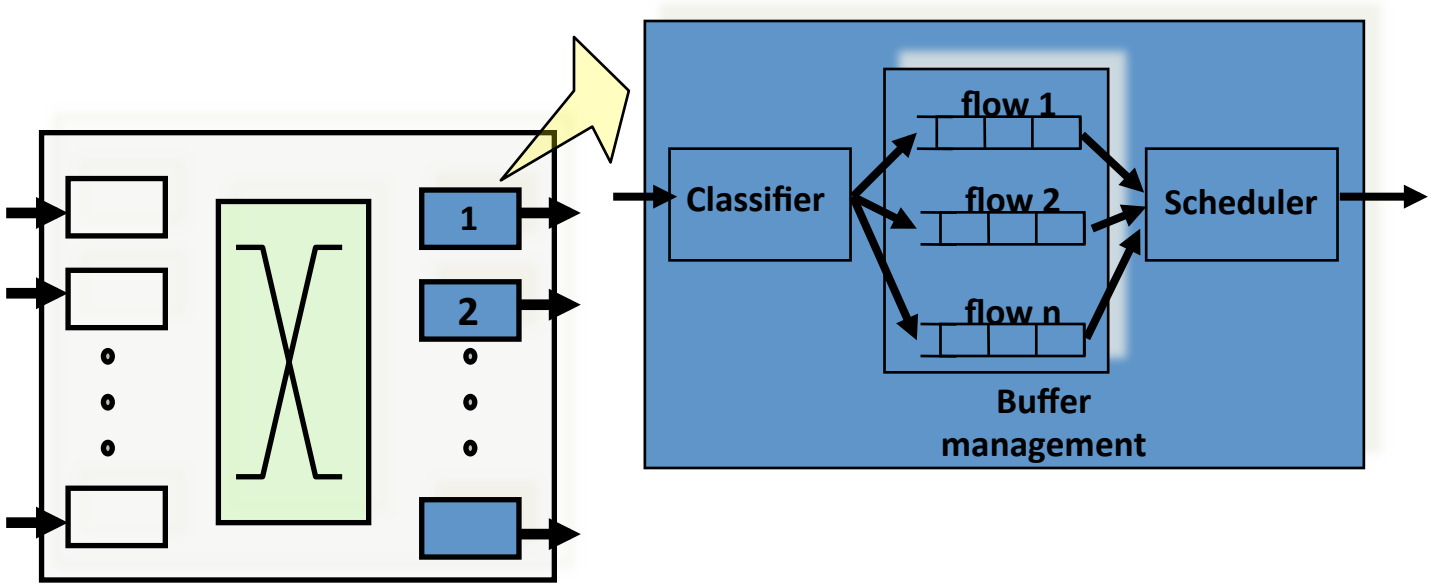
- With many heuristics and optimizations that leverage real-world patterns
  - Some destinations more popular than others
  - Some ports lead to more destinations
  - Typical prefix granularities

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



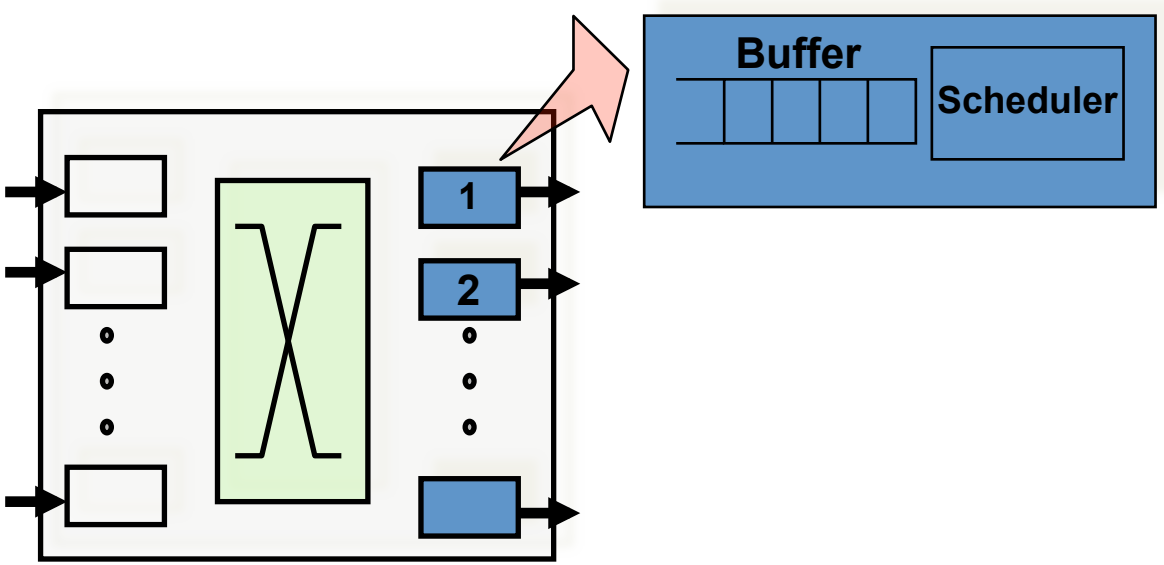
Output Linecard

- 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)

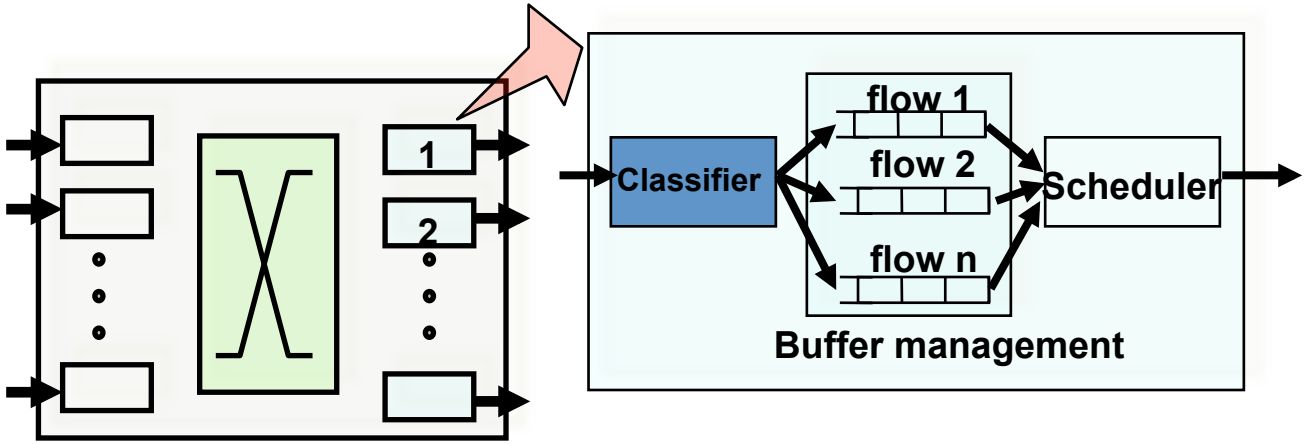
Simplest: FIFO Router

- 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



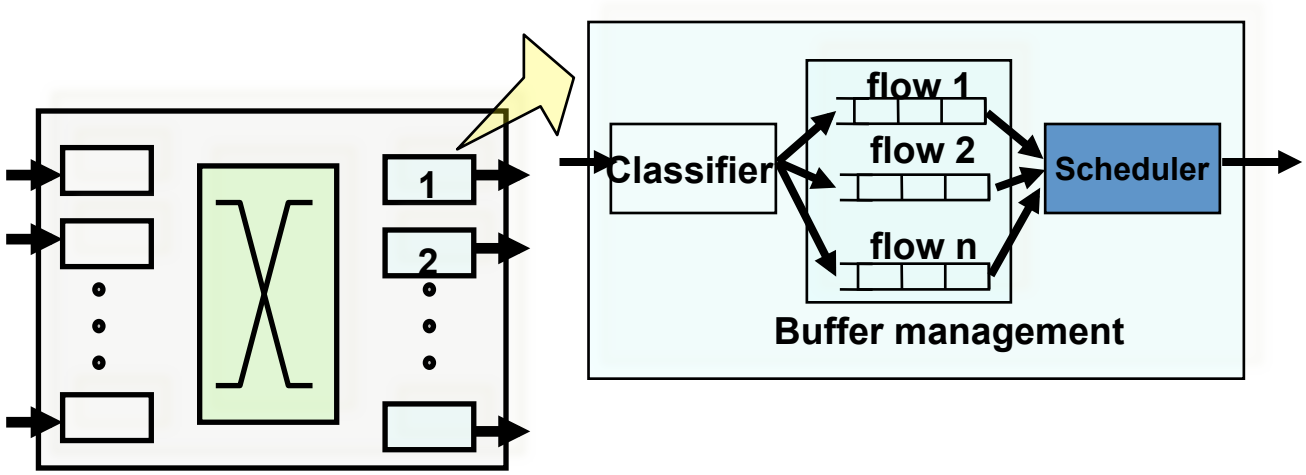
Packet Classification

- 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!



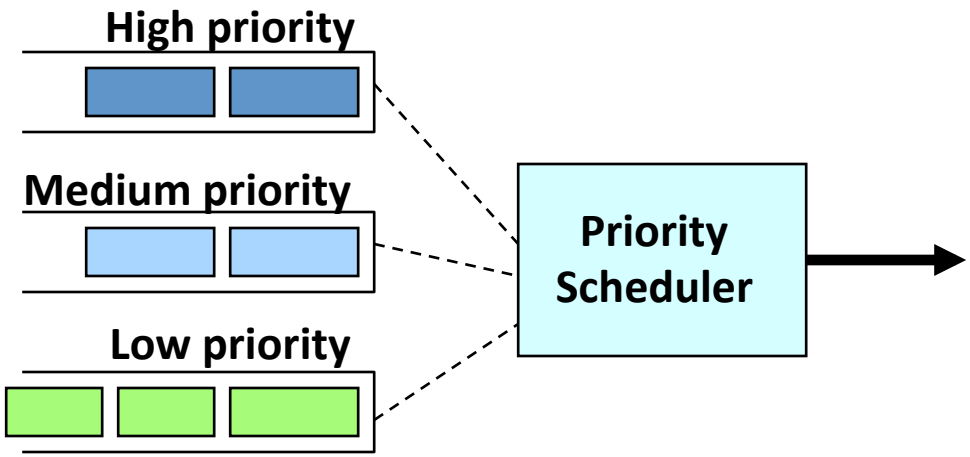
Scheduler

- 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.*)



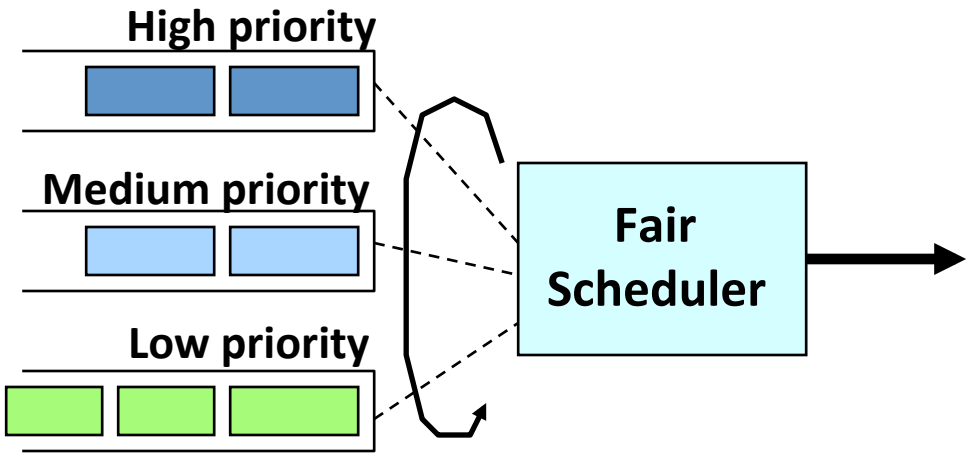
Example: Priority Scheduler

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

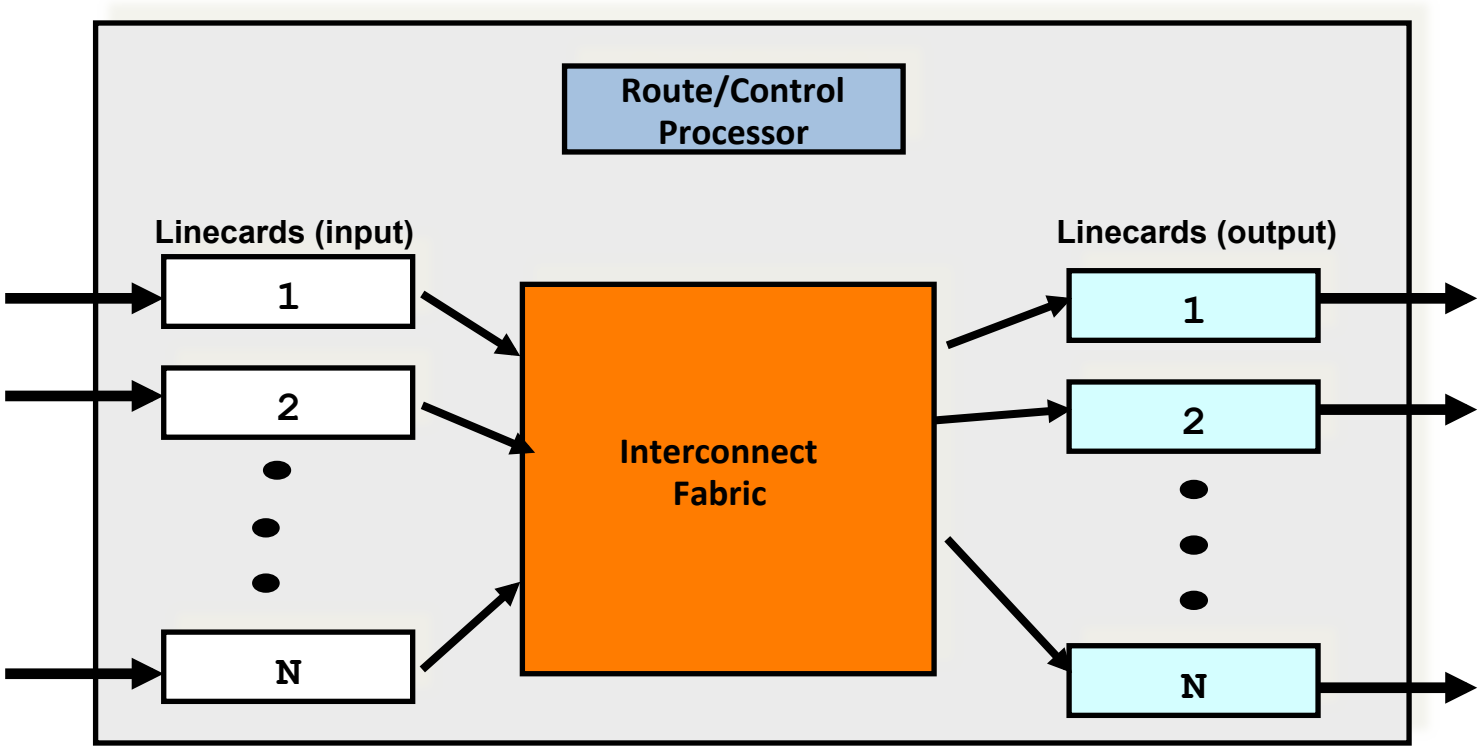


Example: Round Robin Scheduler

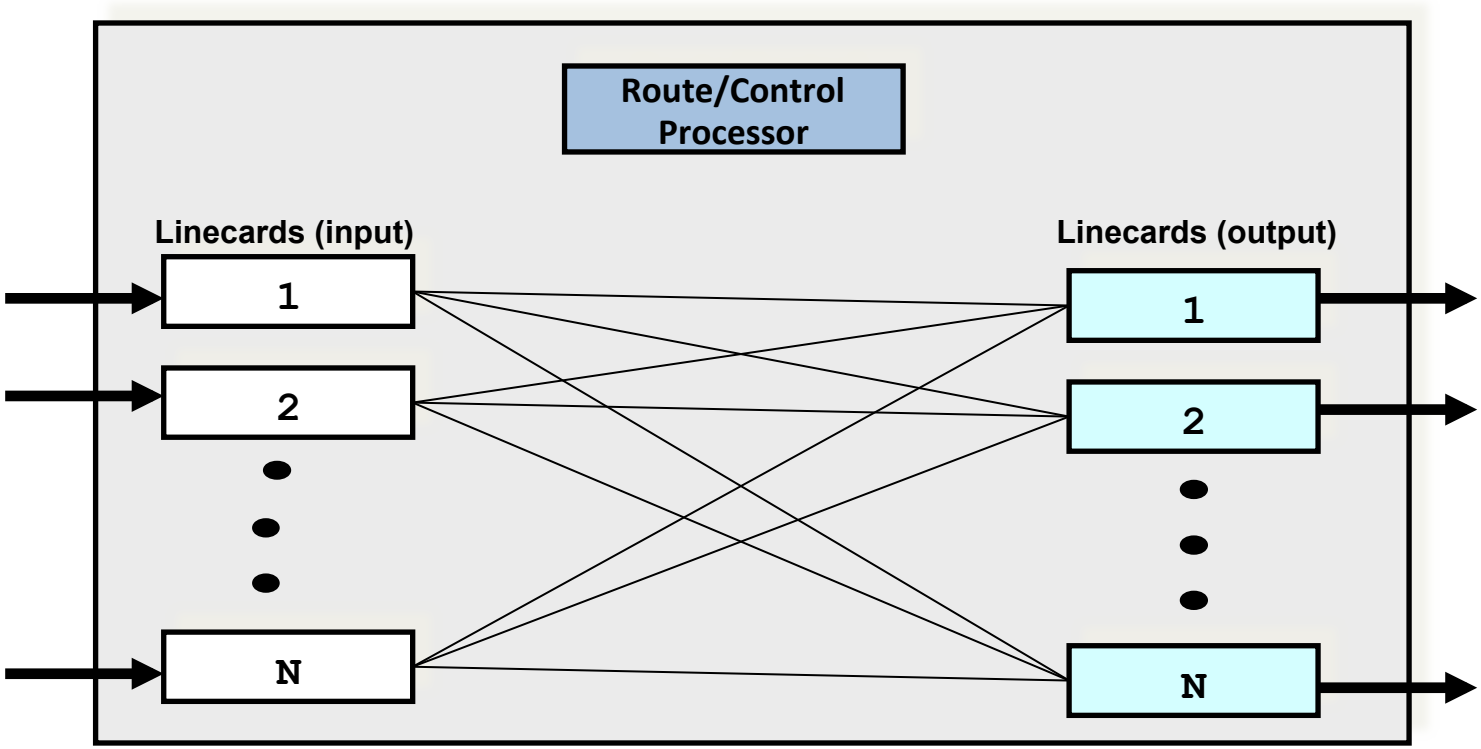
- Round robin: packets are served from each queue in turn



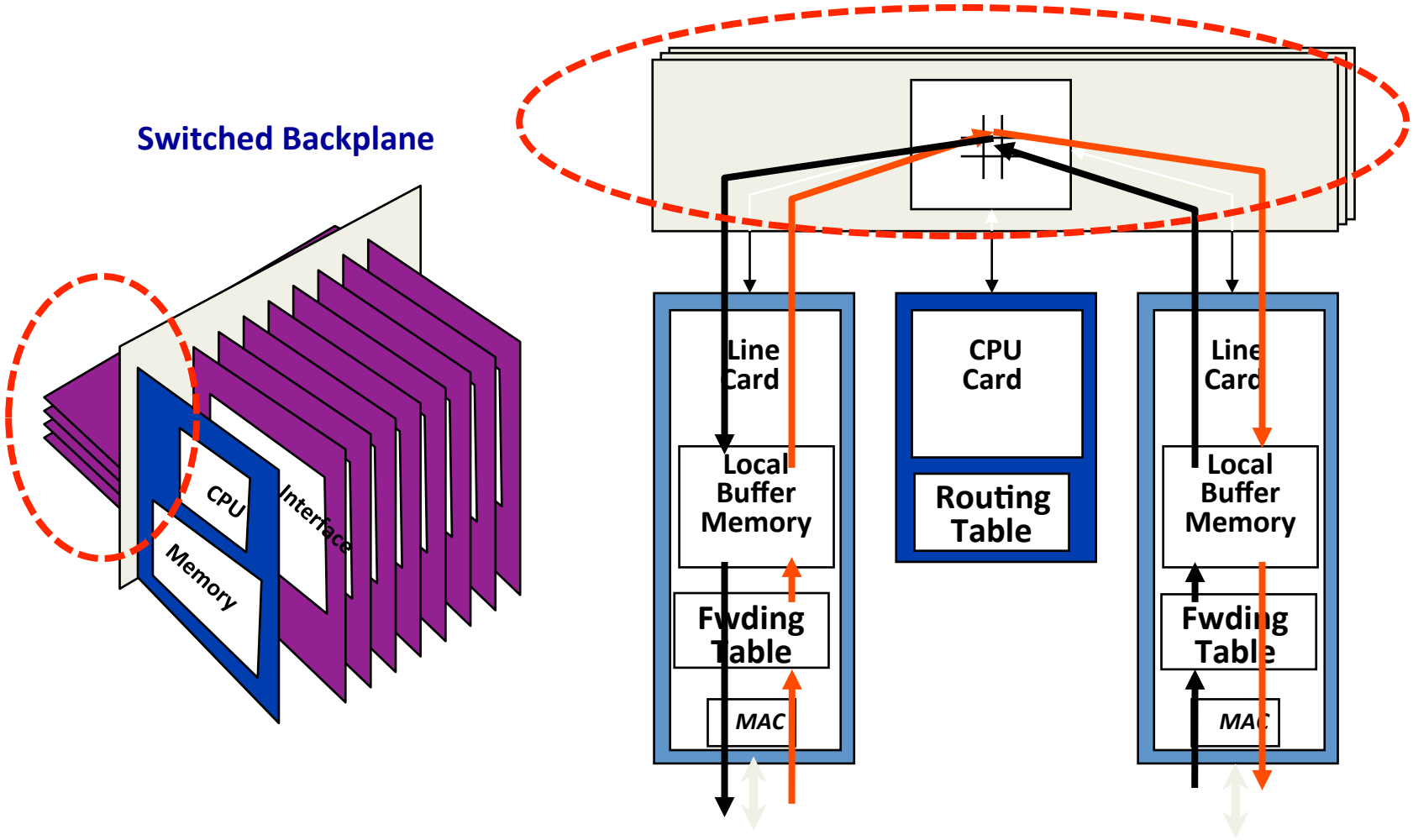
Connecting input to output: Switch fabric



Today's Switch Fabrics: Mini-Network!

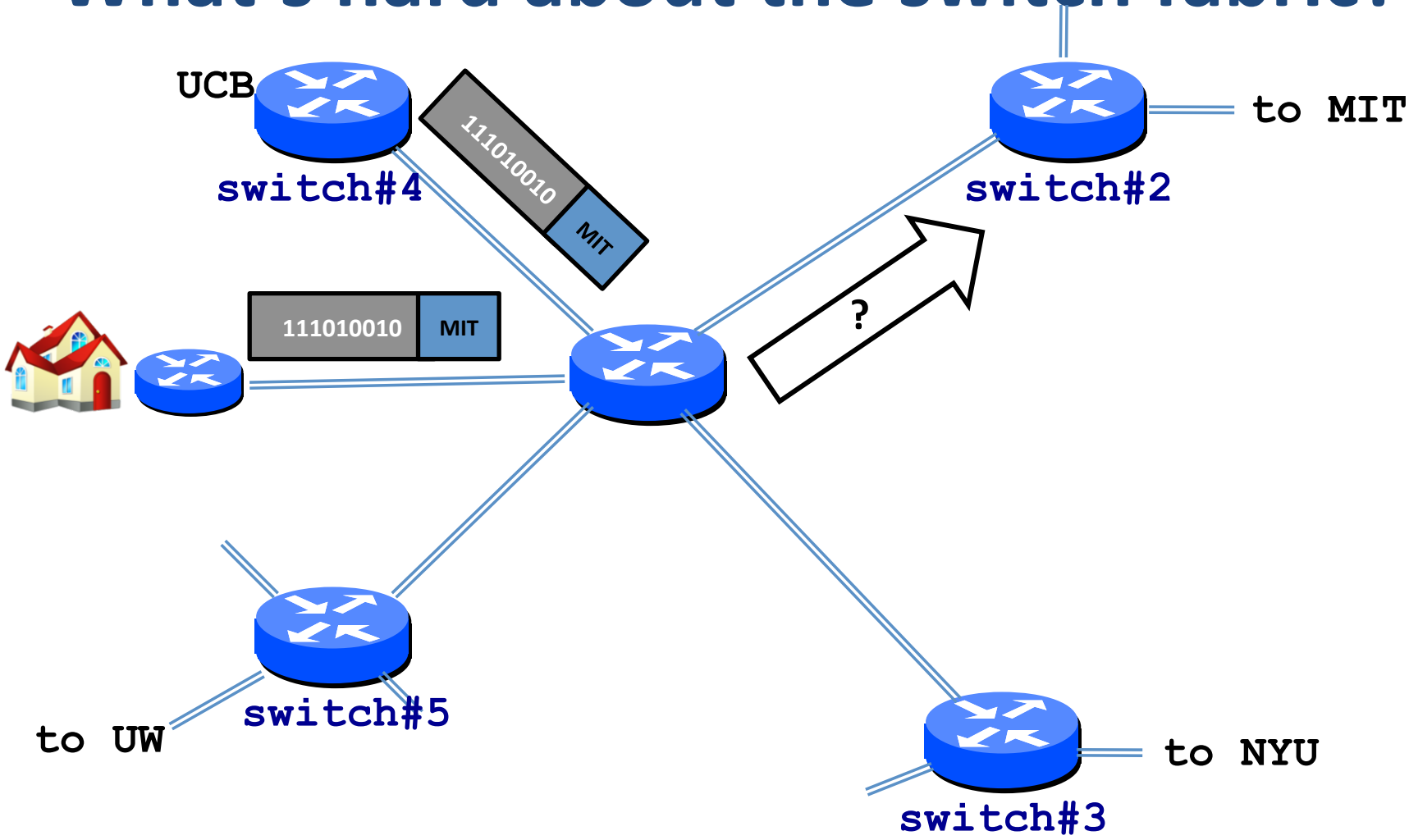


Point-to-Point Switch (3rd Generation)



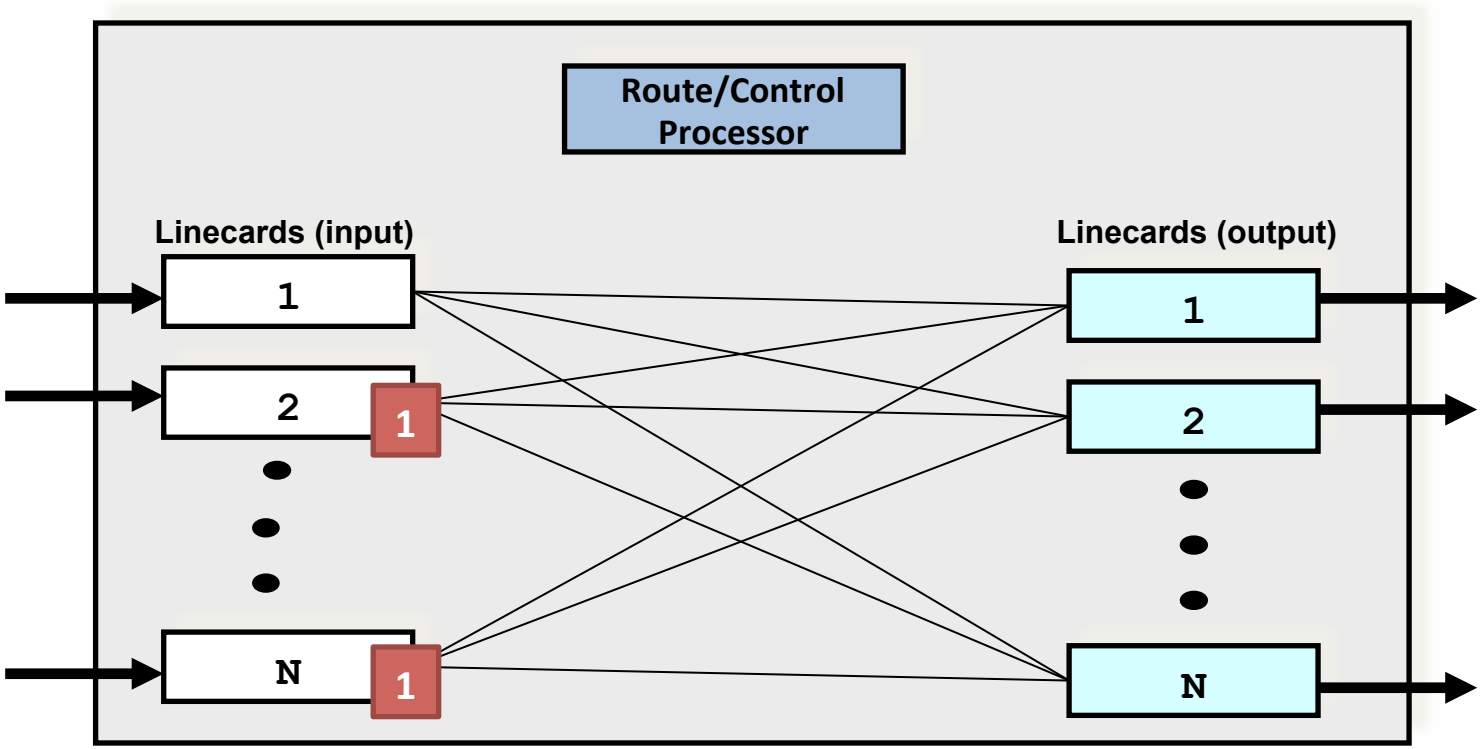
(*Slide by Nick McKeown)

What's hard about the switch fabric?

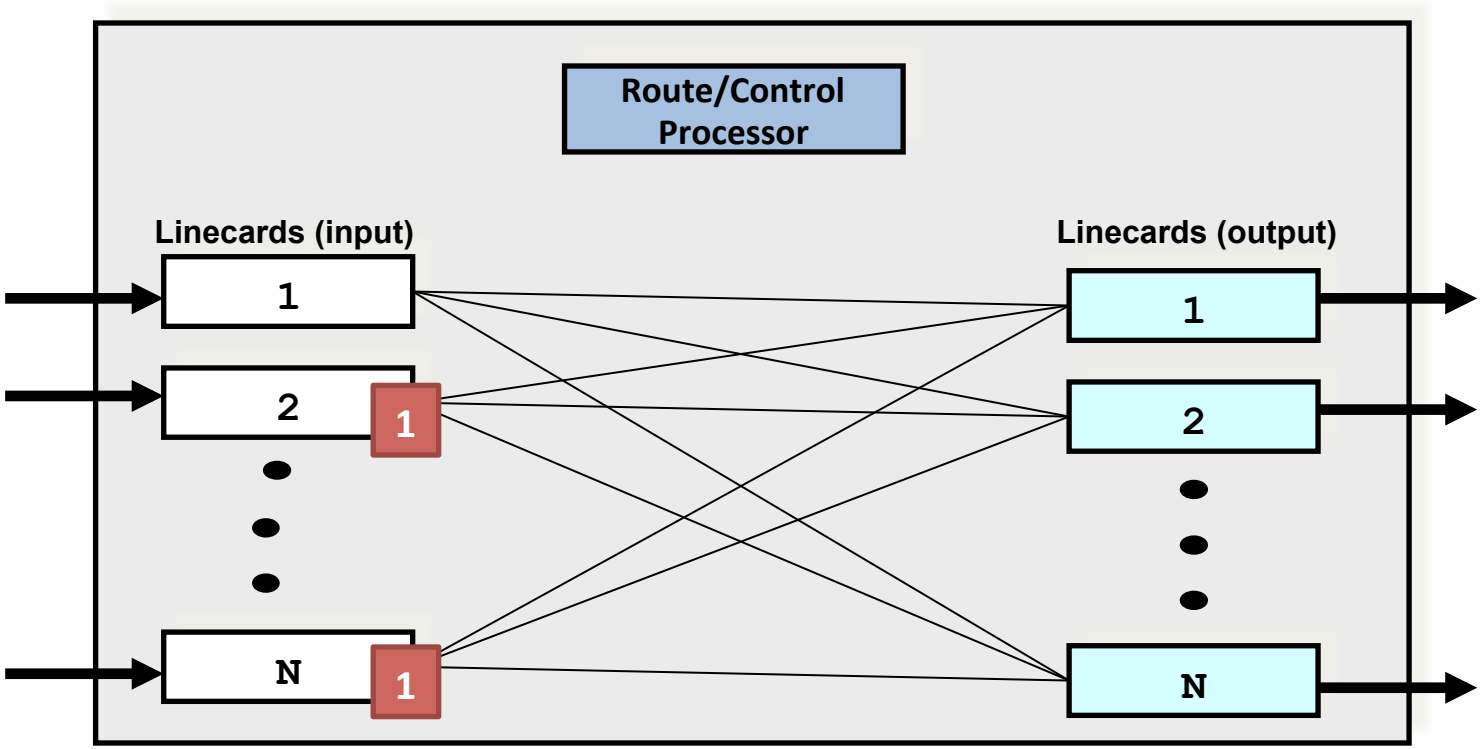


Queuing!

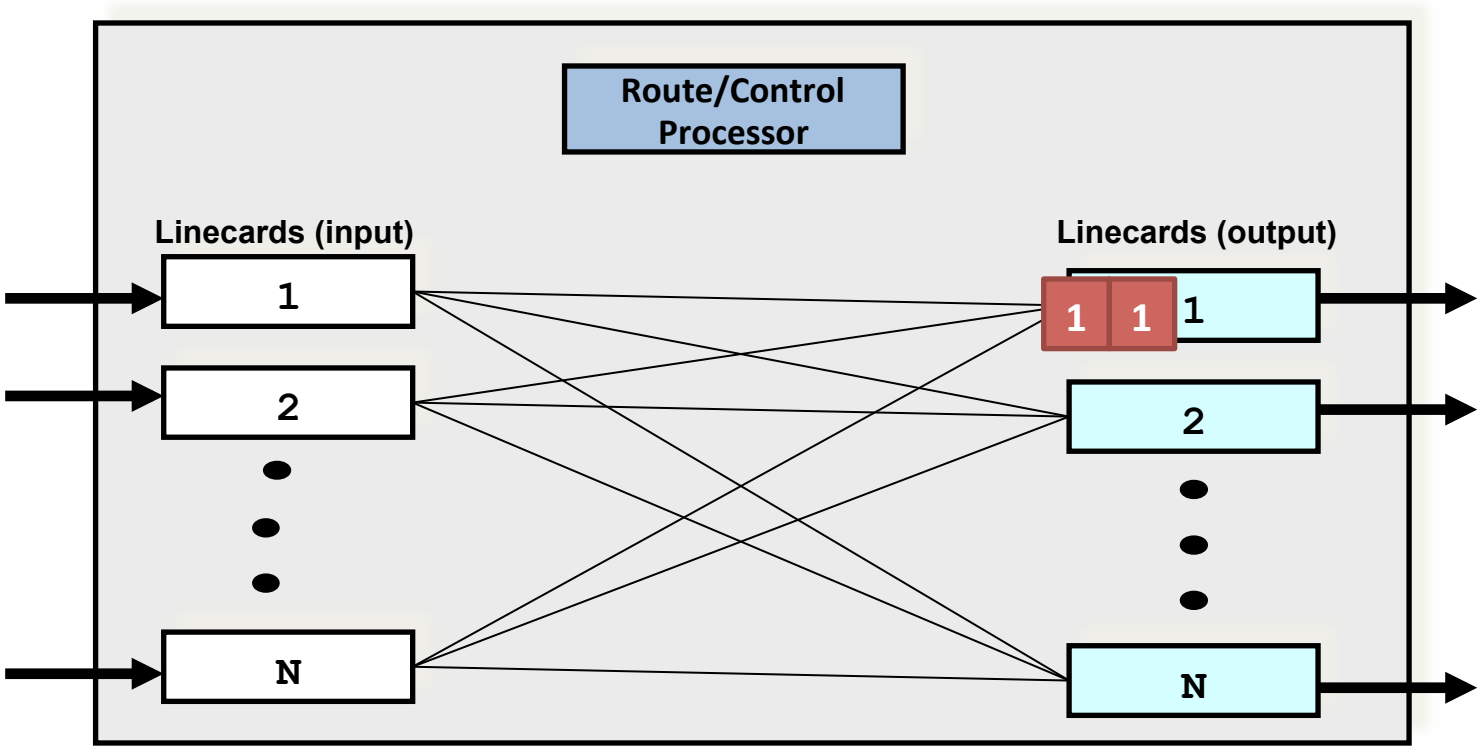
Queuing


Route/Control Processor
Linecards (input)
Linecards (output)
1
2
1
N
1
1
2
N

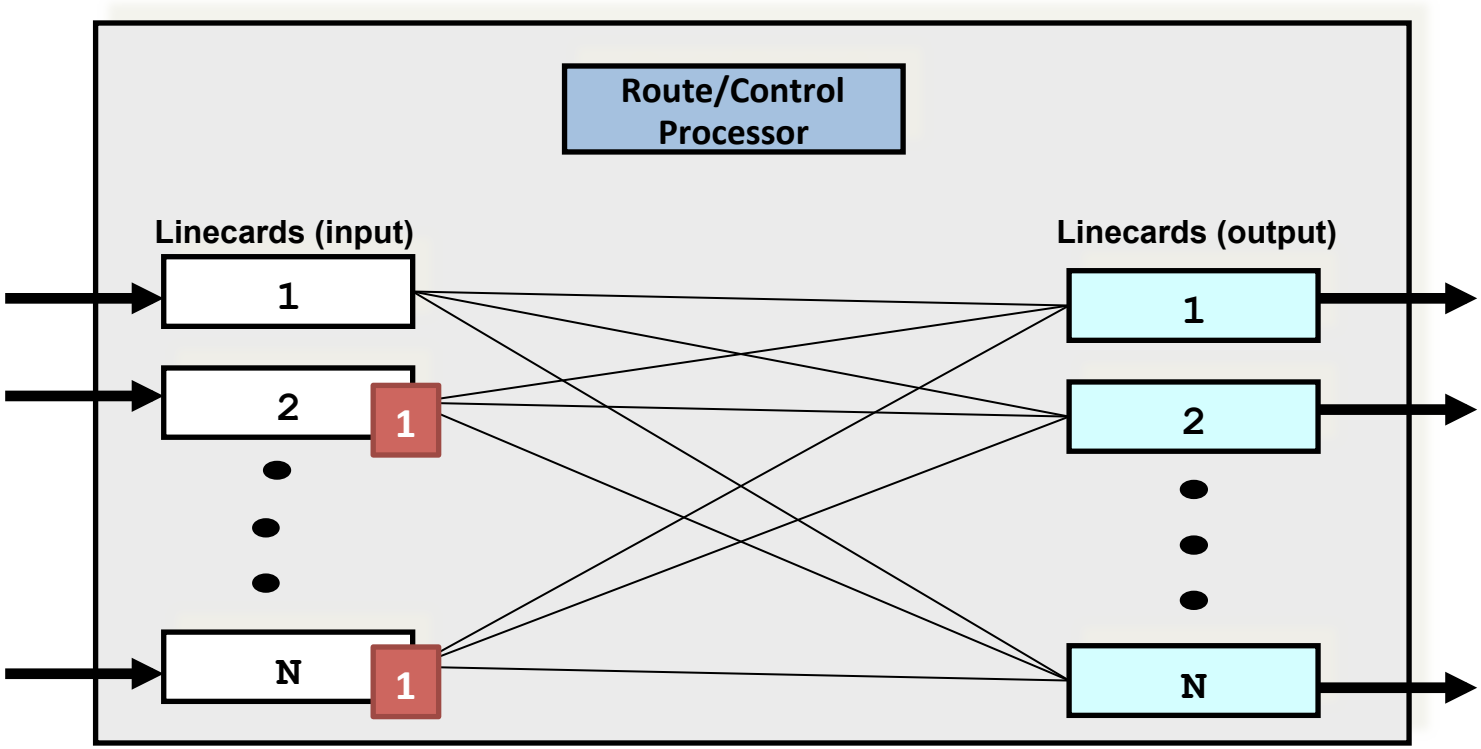
Output queuing



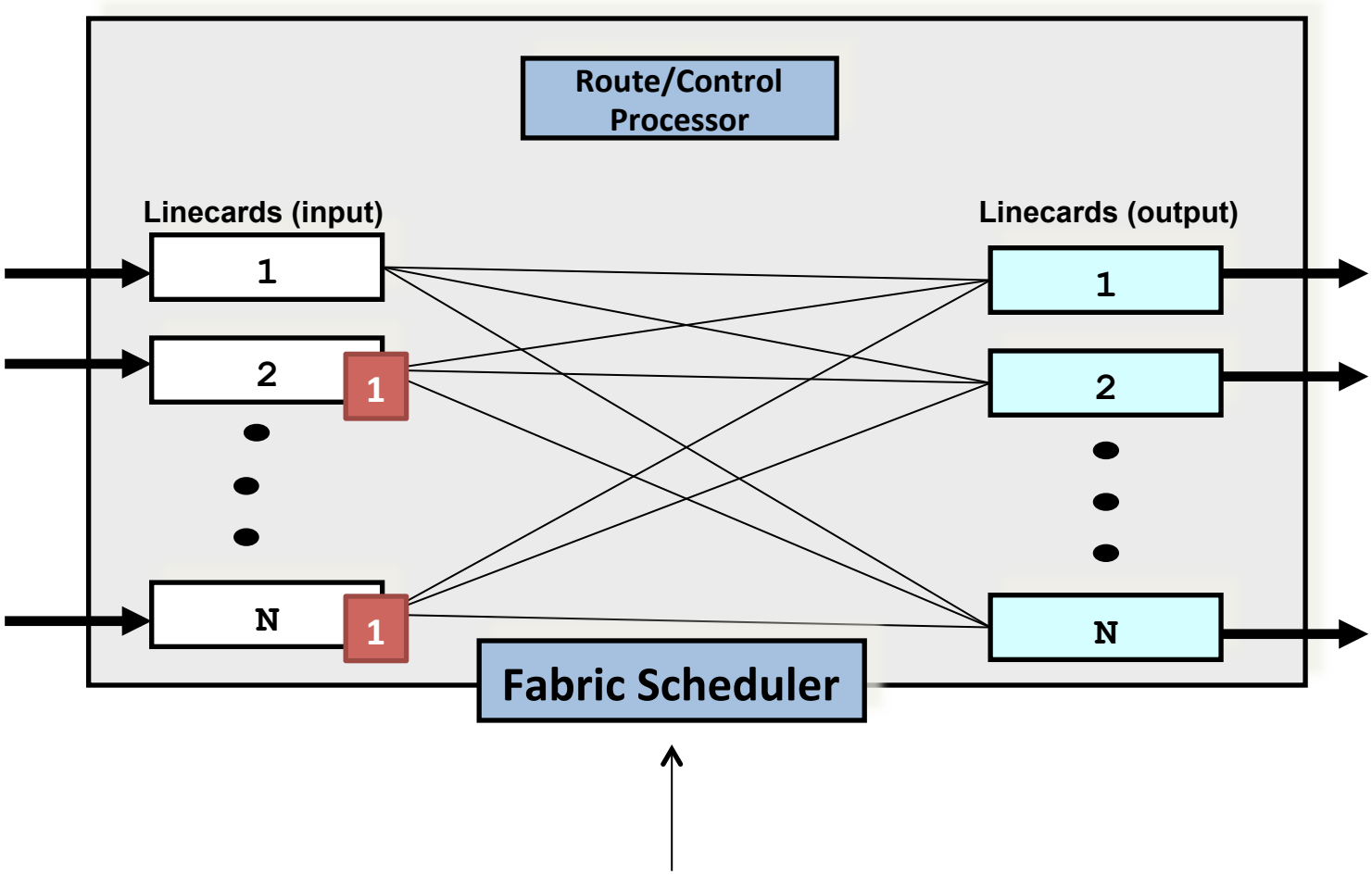
Output queuing



Input queuing

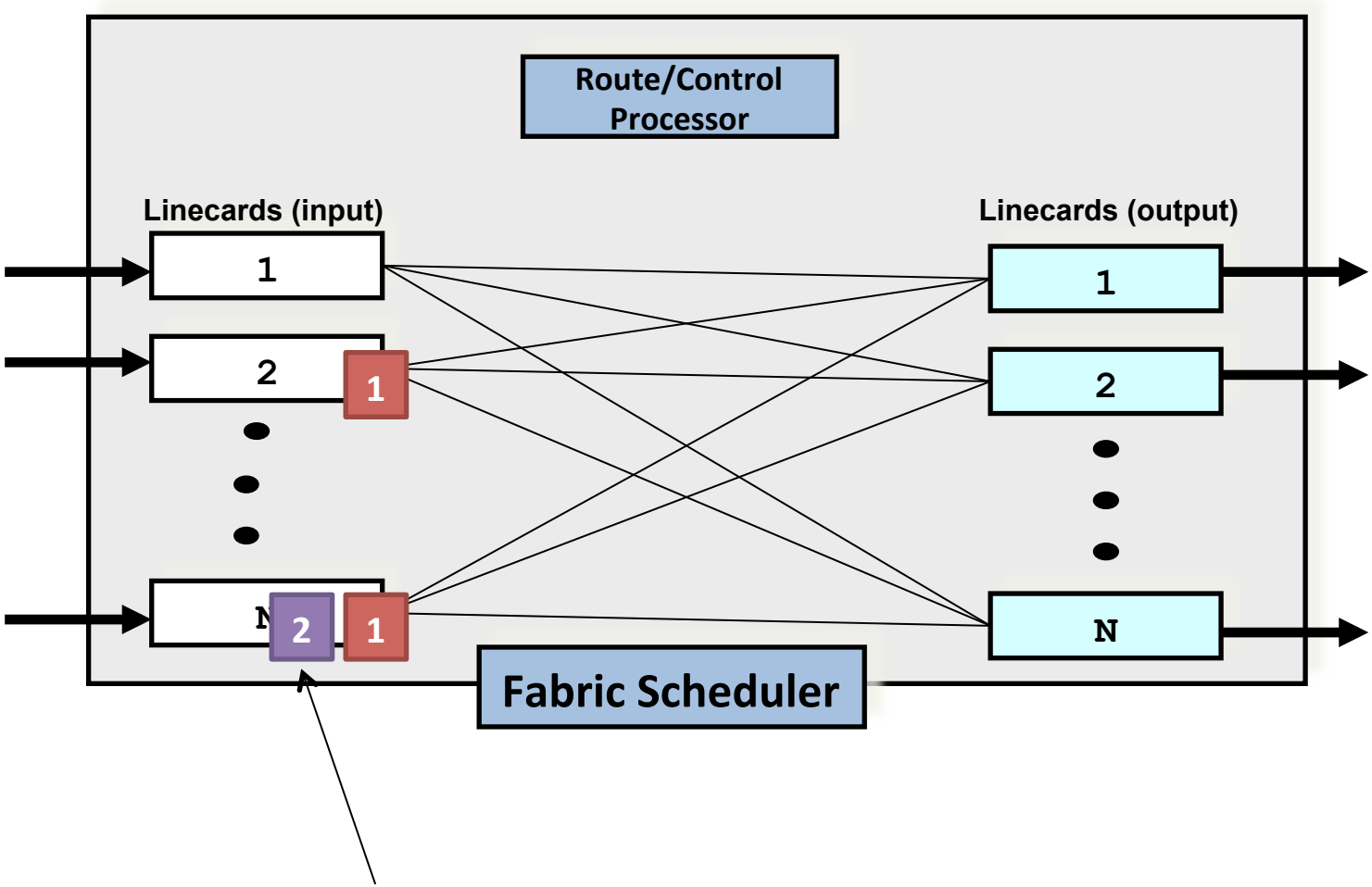


Input Queuing: Challenges



(1) Need a (FAST) internal fabric scheduler!

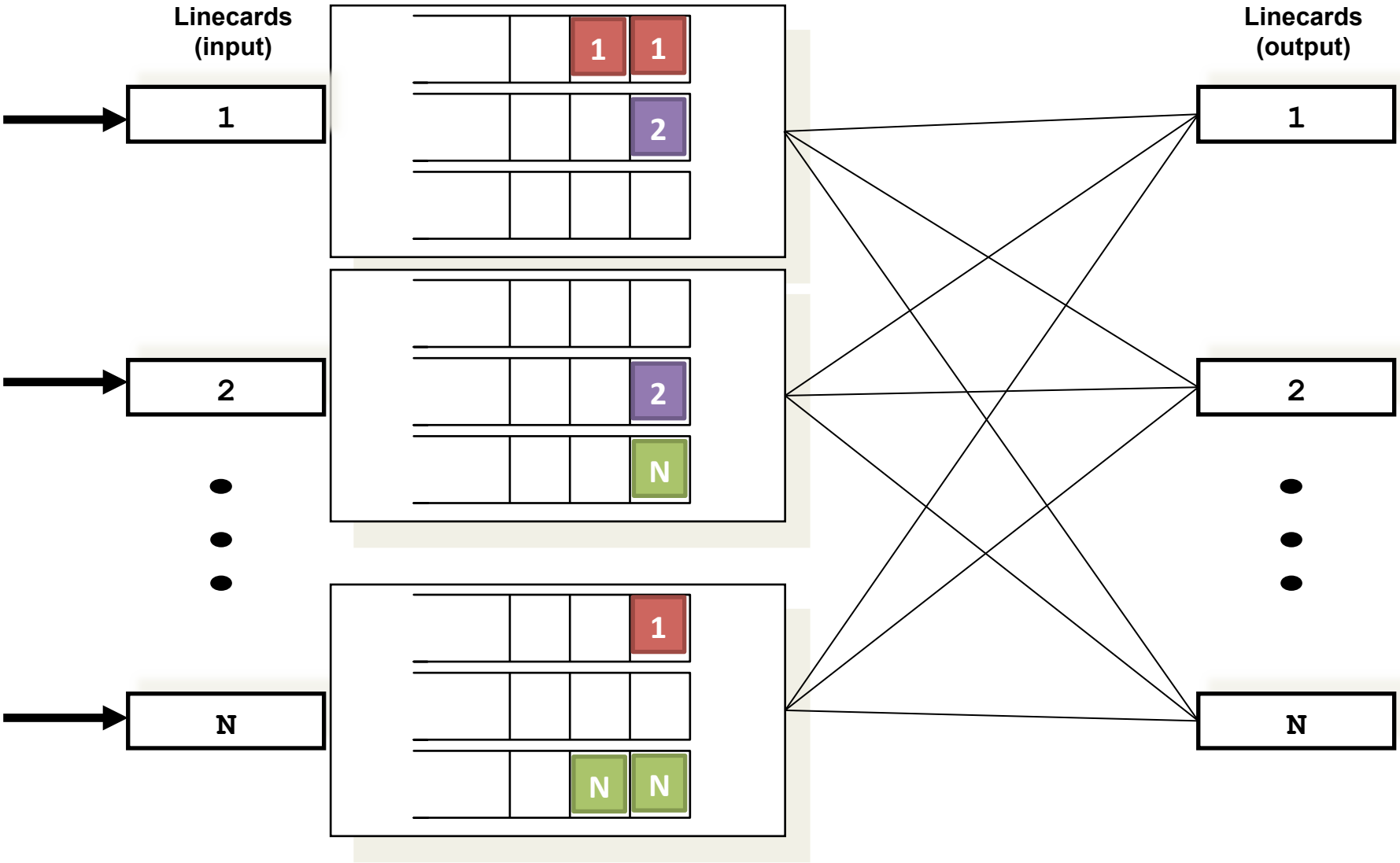
Challenge 2: Head of line blocking



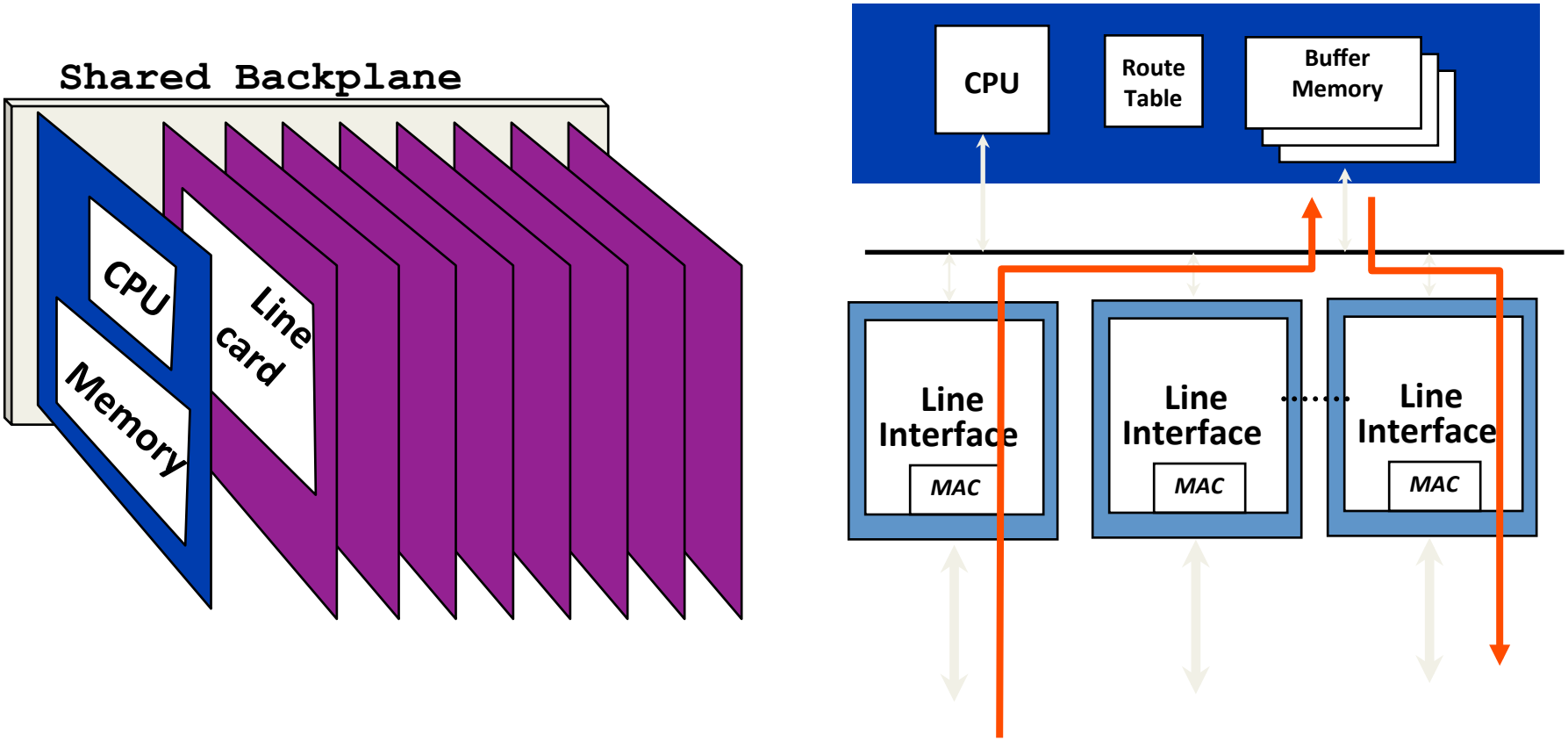
Head of line blocking

Limits throughput to approximately 58% of capacity

Fixing head of line blocking:
Virtual Output Queues



Shared Memory (1st Generation)

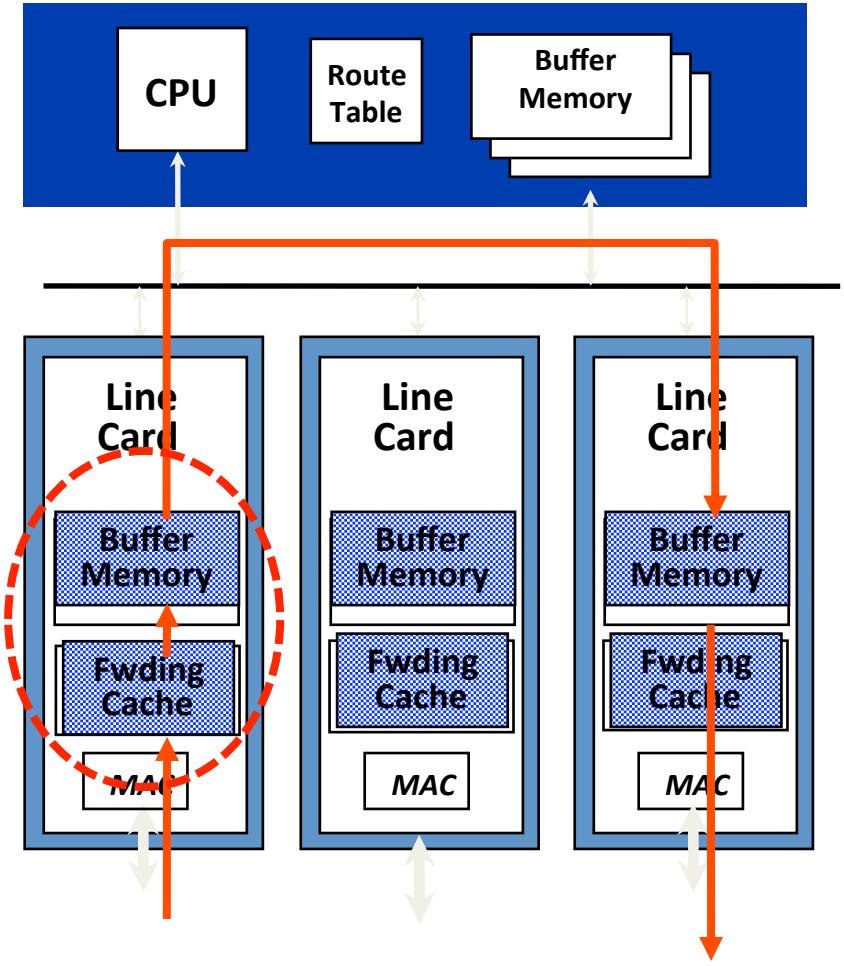


Limited by rate of shared memory

(* Slide by Nick McKeown, Stanford Univ.)

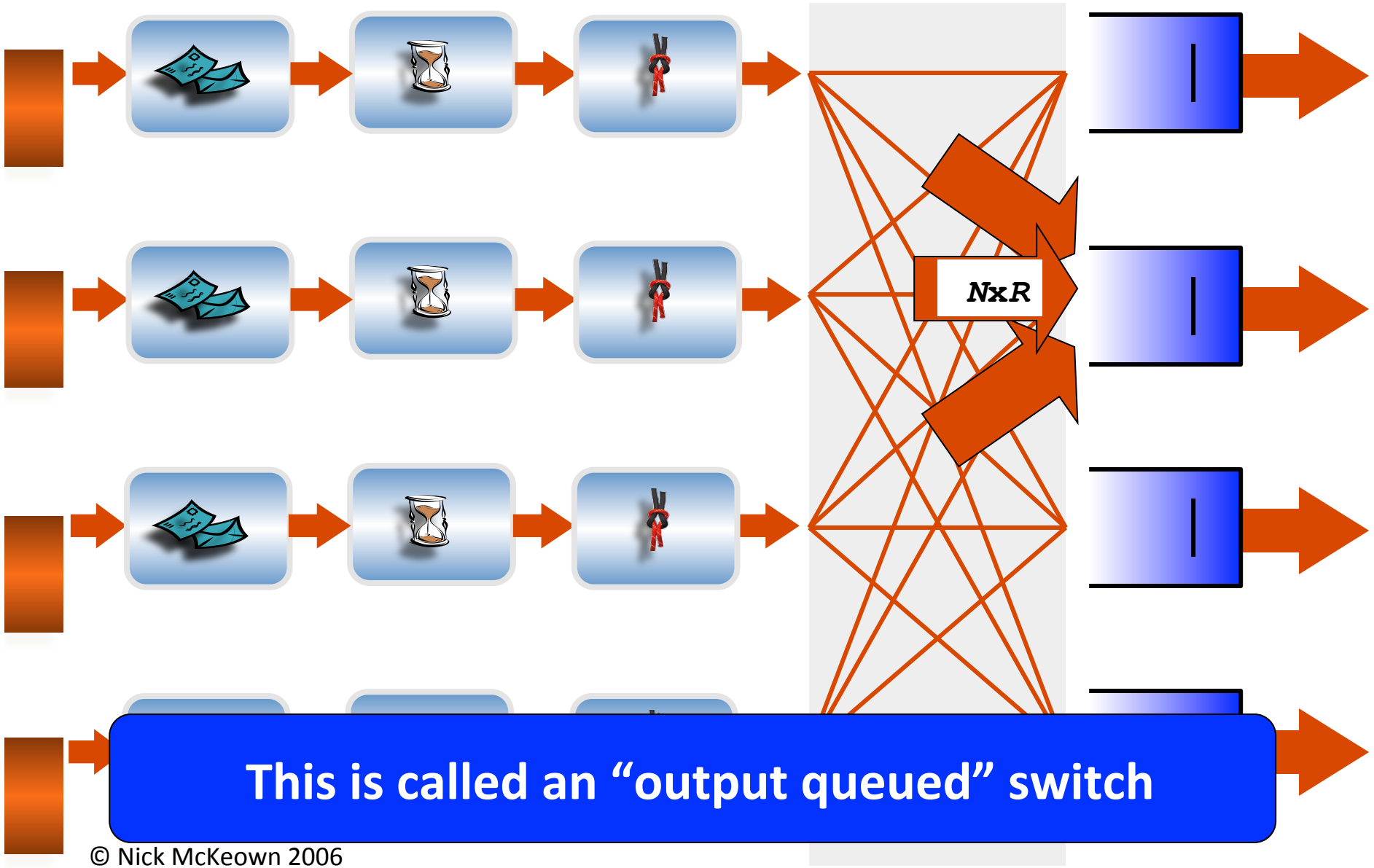
Shared Bus (2nd Generation)

Limited by shared bus

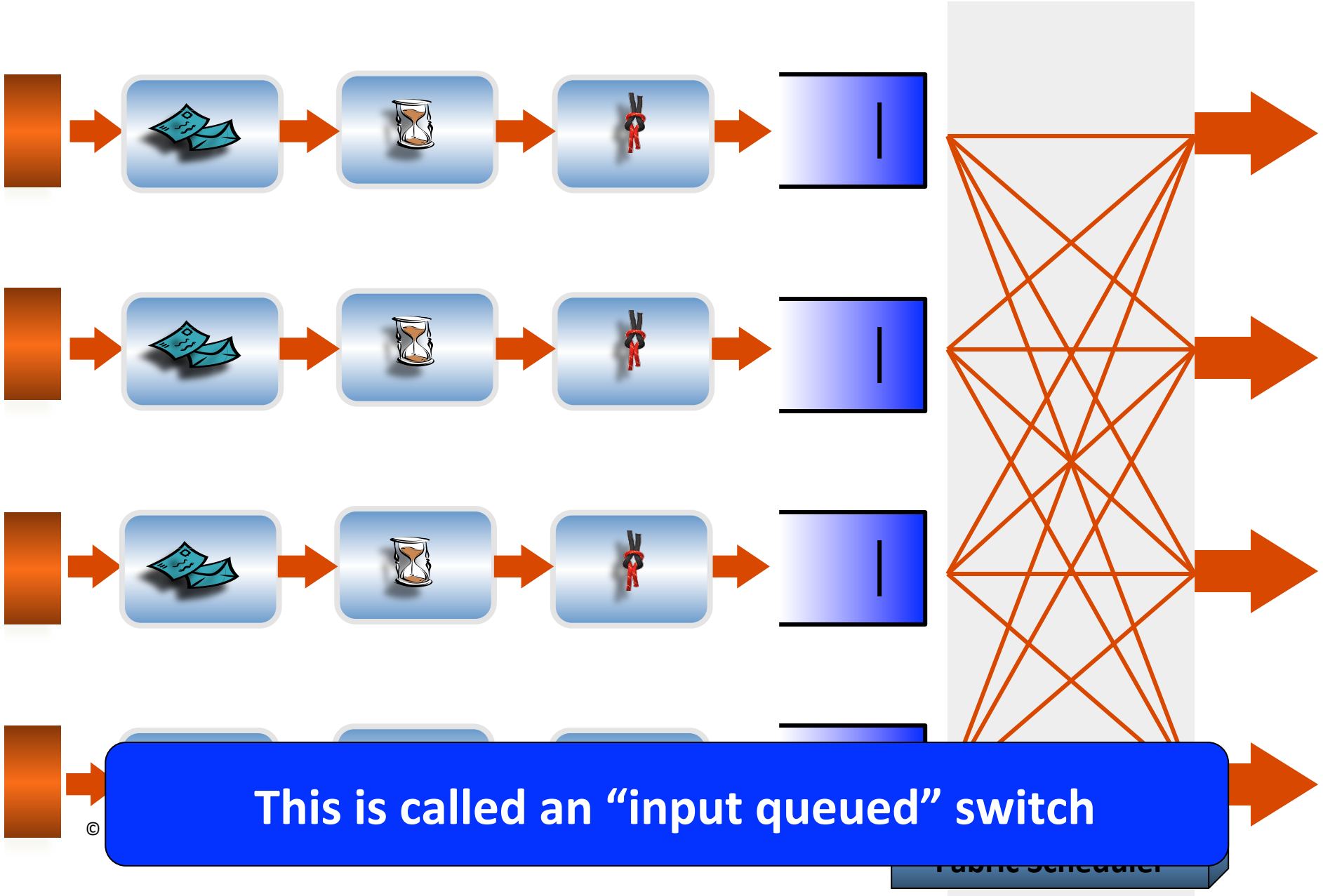


(* Slide by Nick McKeown)

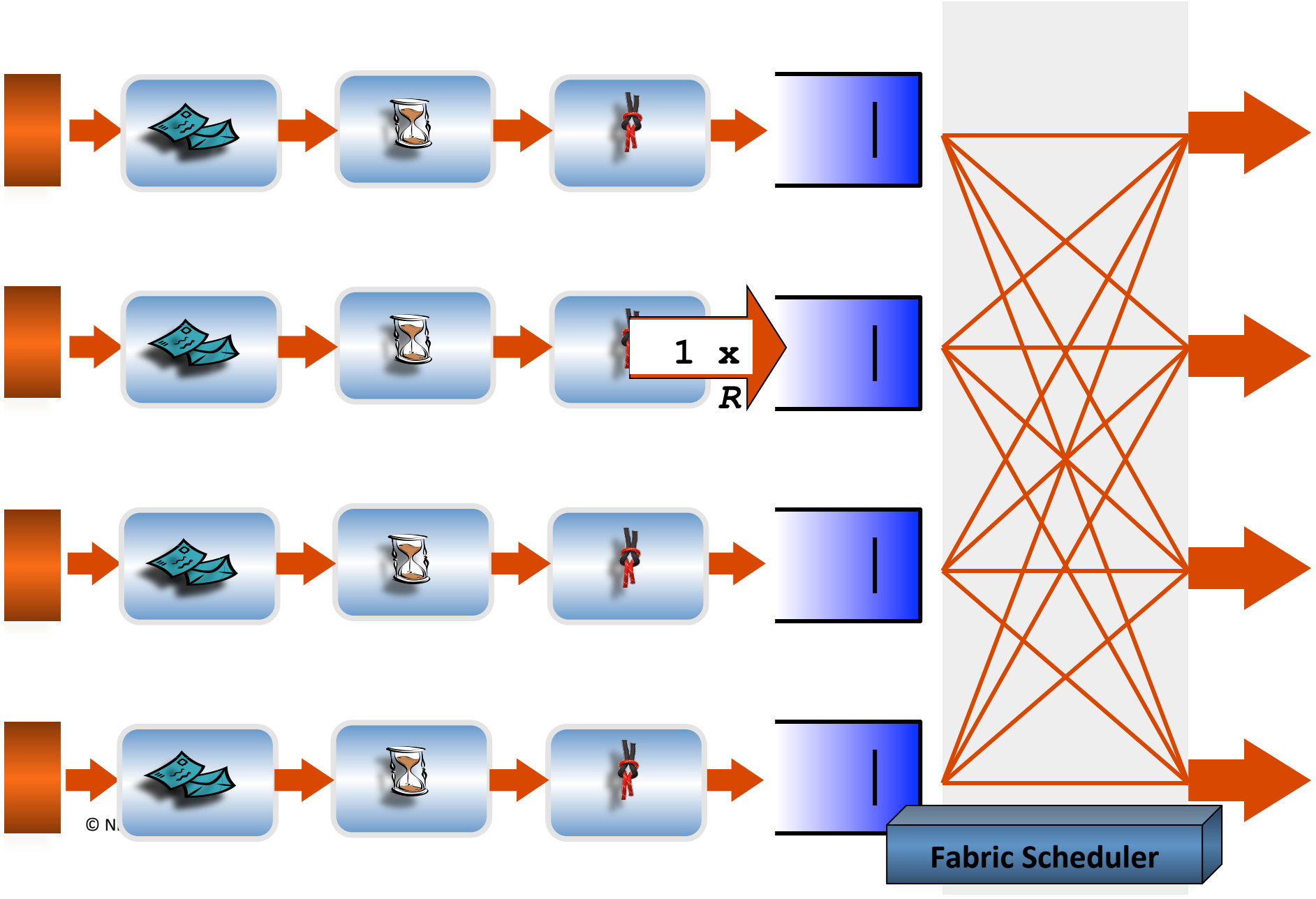
3rd Gen. Router: Switched Interconnects



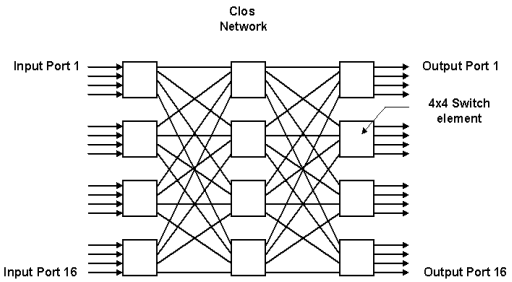
3rd Gen. Router: Switched Interconnects



3rd Gen. Router: Switched Interconnects



Reality is more complicated

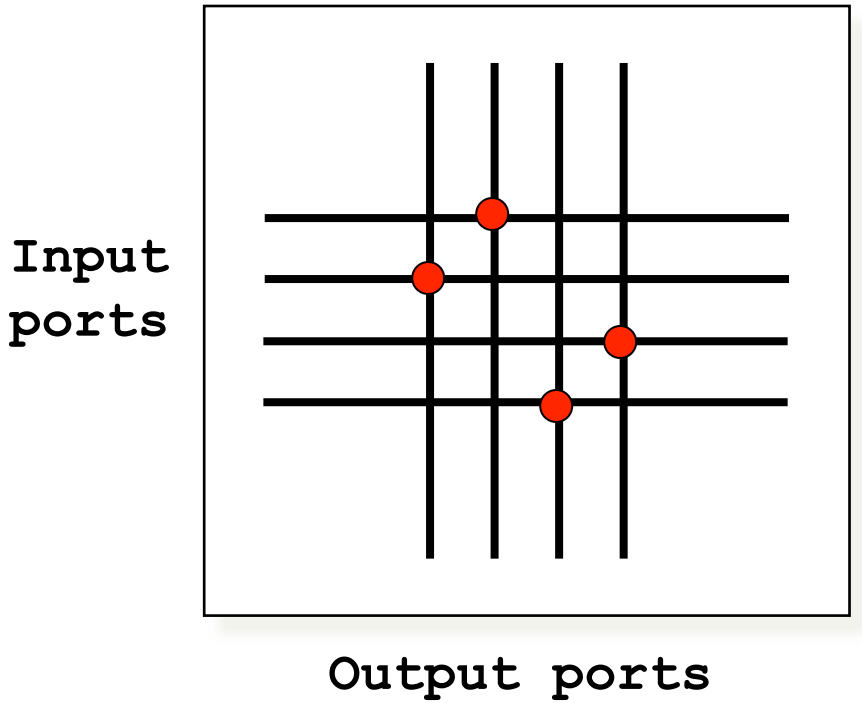


- Commercial (high-speed) routers use
  - combination of input and output queuing
  - complex multi-stage switching topologies (Clos, Benes)
  - distributed, multi-stage schedulers (for scalability)

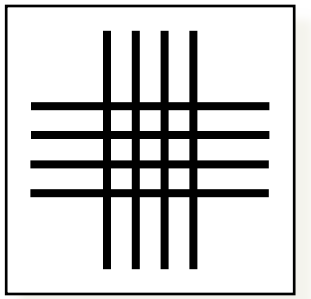
- We'll consider one simpler context
  - de-facto architecture for a long time and still used in lower-speed routers

Context

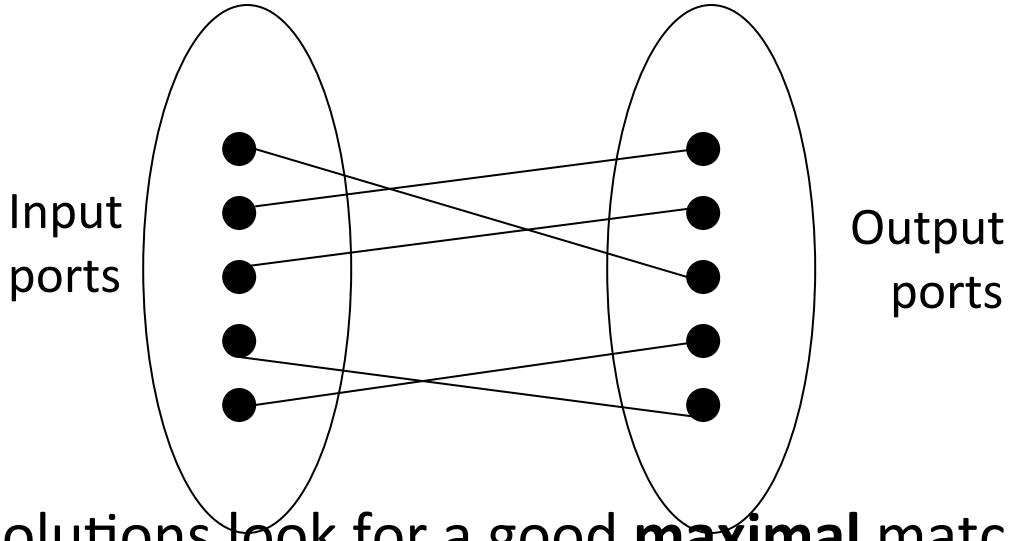
- Crossbar fabric
- Centralized scheduler



Scheduling



- Goal: run links at full capacity, fairness across inputs
- Scheduling formulated as finding a matching on a bipartite graph



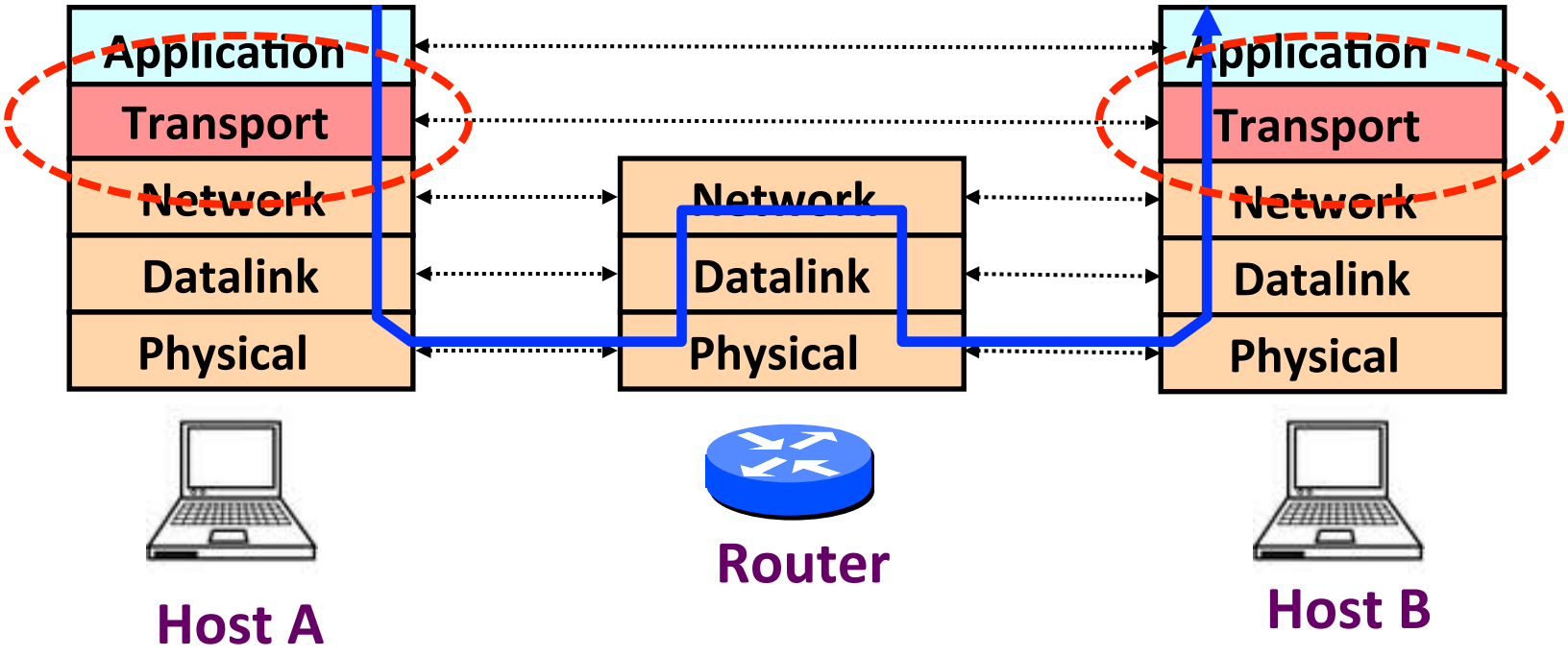
- Practical solutions look for a good **maximal** matching (fast)

IP Routers Recap

- Core building block of Internet
- Scalable addressing → Longest Prefix Matching
- Need fast implementations for:
  - Longest prefix matching
  - Switch fabric scheduling

Transport Layer

- Layer at end-hosts, between the application and network layer



Why do we need a transport layer?

Why a transport layer?

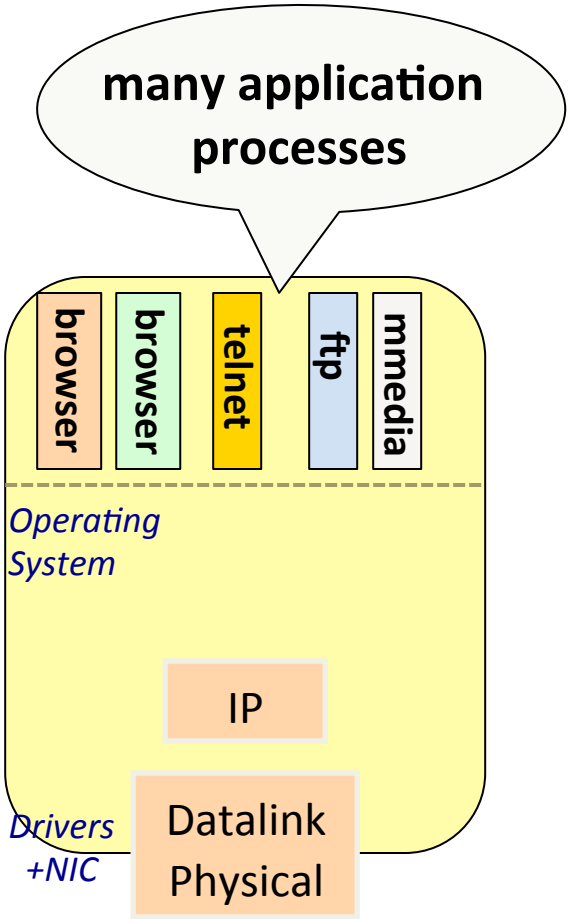
1. Demultiplex packets between many applications

2. Additional services on top of IP

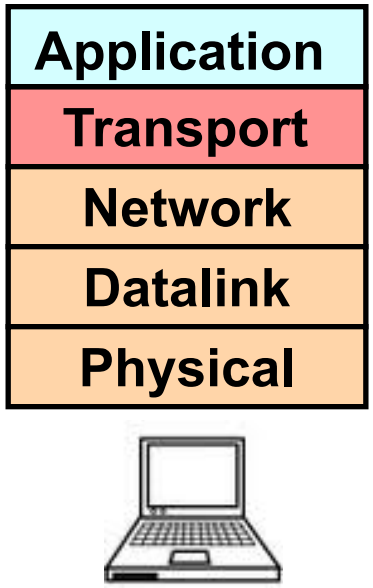
Why a transport layer: Demultiplexing

- IP packets are addressed to a host but end-to-end communication is between application processes at hosts
  - Need a way to decide which packets go to which applications (*multiplexing/demultiplexing*)

Why a transport layer: Demultiplexing

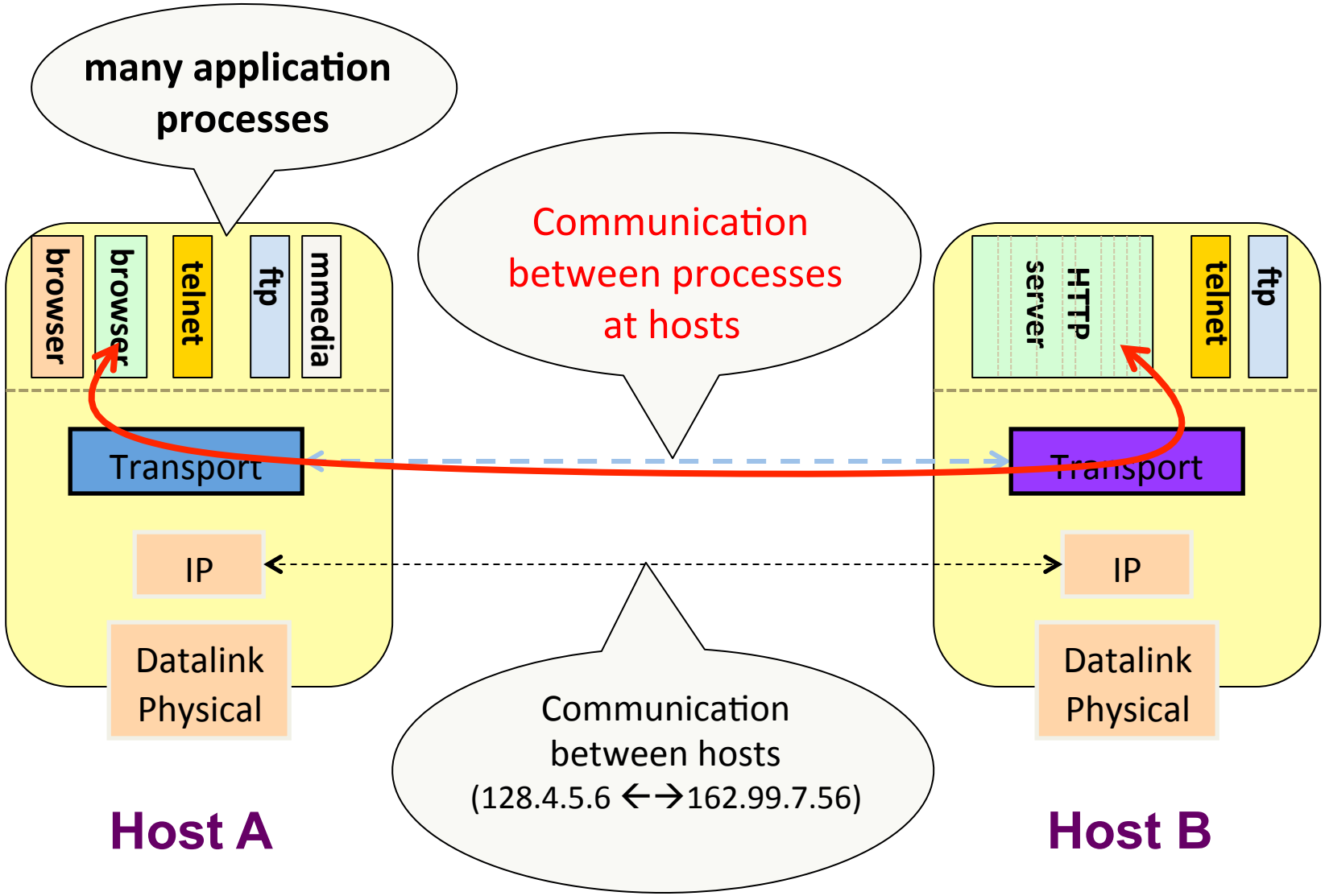


Host A



Host B

Why a transport layer: Demultiplexing



Why a transport layer: Improved service model

- IP provides a weak service model (*best-effort*)
  - Packets can be corrupted, delayed, dropped, reordered, duplicated
  - No guidance on how much traffic to send and when
  - Dealing with this is tedious for application developers

Role of the Transport Layer

- Communication between application processes
  - Mux and demux from/to application processes
  - Implemented using *ports*

Role of the Transport Layer

- Communication between application processes
- Provide common end-to-end services for app layer [optional]
  - Reliable, in-order data delivery
  - Well-paced data delivery
    - too fast may overwhelm the network
    - too slow is not efficient

Role of the Transport Layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
- TCP and UDP are the common transport protocols
  - also SCTP, MTCP, SST, RDP, DCCP, …

Role of the Transport Layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
- TCP and UDP are the common transport protocols
- UDP is a minimalist, no-frills transport protocol
  - only provides mux/demux capabilities

Role of the Transport Layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
- TCP and UDP are the common transport protocols
- UDP is a minimalist, no-frills transport protocol
- TCP is the whole-hog protocol
  - offers apps a reliable, in-order, bytestream abstraction
  - with congestion control
  - but no performance guarantees (delay, bw, etc.)

Summary

- IP Routers
  - $$$
  - Line cards receive packets, change headers
  - LPM for scalable addressing
  - Fast hardware needed for LPM, fabric scheduling
- Transport Layer
  - Demultiplexes between applications on same host
  - 2 protocols:
    - UDP: minimal protocol
    - TCP: reliable, in order byte stream (more next week!)