IP Data Plane (Contd.)

CPSC 433/533, Spring 2021

Anurag Khandelwal

Administrivia

- HW2 out today
  - **Lead:** Ramla
  - Will test knowledge on IP, and some initial concepts in TCP
- Have to move my OH again…
  - Earlier by 1h: **3pm to 4pm**
  - Same link…

Let’s take a quick look at the IPv6 header…

IPv6

IPv6

- **Motivated by address exhaustion**

IPv6

- **Motivated by address exhaustion**
  - Addresses ***four*** times as big (128 bits)

IPv6

- **Motivated by address exhaustion**
  - Addresses ***four*** times as big (128 bits)
  - Number of possible addresses:

IPv6

- **Motivated by address exhaustion**
  - Addresses ***four*** times as big (128 bits)
  - Number of possible addresses:
    - 340 undecillion, or 340 billion billion billion billion

IPv6

- **Motivated by address exhaustion**
  - Addresses ***four*** times as big (128 bits)
  - Number of possible addresses:
    - 340 undecillion, or 340 billion billion billion billion
- **Steve Deering focused on simplifying IP**

IPv6

- **Motivated by address exhaustion**
  - Addresses ***four*** times as big (128 bits)
  - Number of possible addresses:
    - 340 undecillion, or 340 billion billion billion billion
- **Steve Deering focused on simplifying IP**
  - Got rid of all fields that were not absolutely necessary

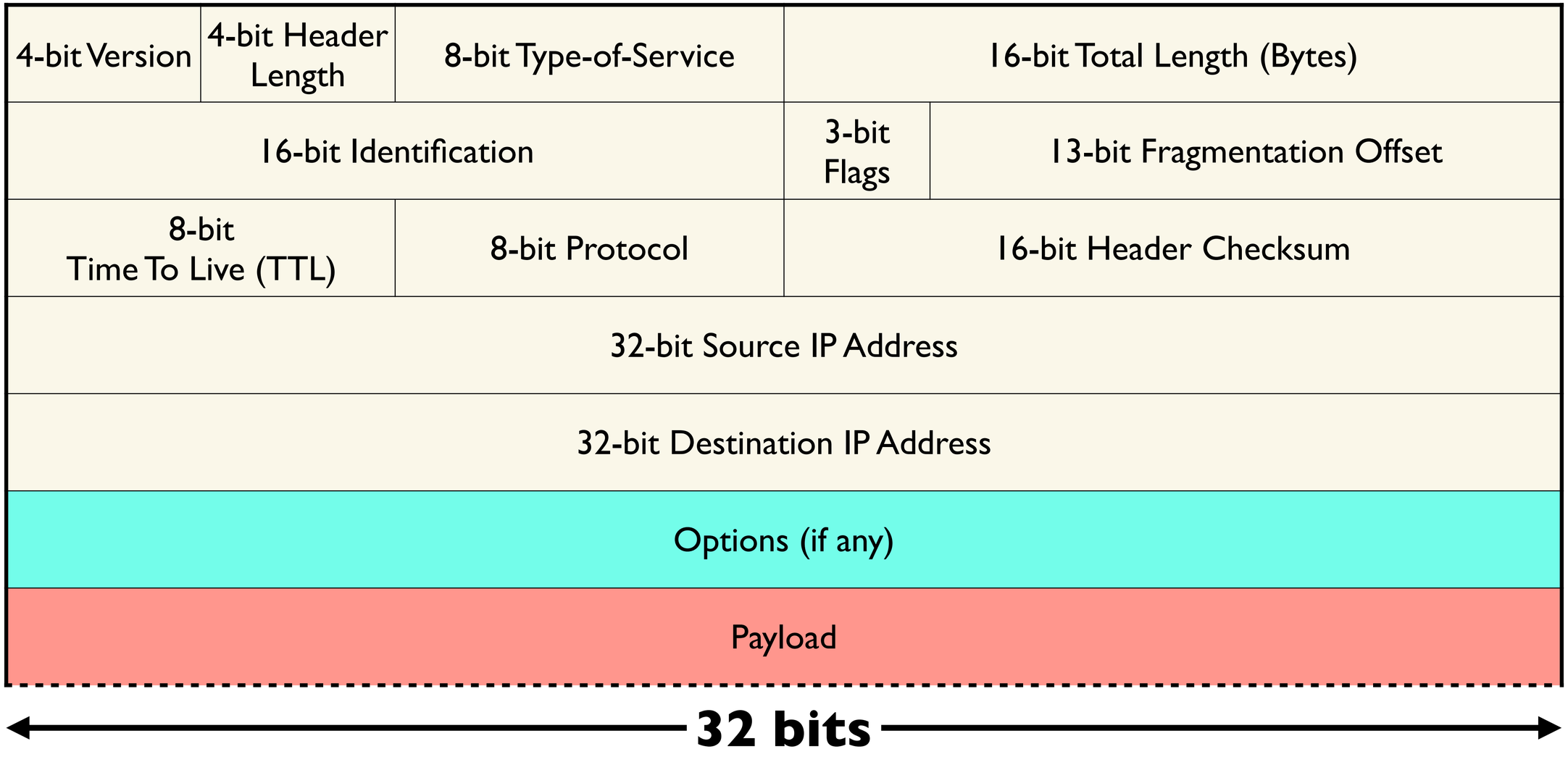
IPv6

- **Motivated by address exhaustion**
  - Addresses ***four*** times as big (128 bits)
  - Number of possible addresses:
    - 340 undecillion, or 340 billion billion billion billion
- **Steve Deering focused on simplifying IP**
  - Got rid of all fields that were not absolutely necessary
  - "Spring cleaning" for IP

IPv6

- **Motivated by address exhaustion**
  - Addresses ***four*** times as big (128 bits)
  - Number of possible addresses:
    - 340 undecillion, or 340 billion billion billion billion
- **Steve Deering focused on simplifying IP**
  - Got rid of all fields that were not absolutely necessary
  - "Spring cleaning" for IP
- **Result is an elegant, if unambitious, protocol**

What "clean up" would you do?



IPv6: Summary of Changes

IPv6: Summary of Changes

- **Eliminated fragmentation (*Why?*)**

IPv6: Summary of Changes

- **Eliminated fragmentation (*Why?*)**
- **Eliminated checksum (*Why?*)**

IPv6: Summary of Changes

- **Eliminated fragmentation (*Why?*)**
- **Eliminated checksum (*Why?*)**
- **New options mechanism (next header) (*Why?*)**

IPv6: Summary of Changes

- **Eliminated fragmentation (*Why?*)**
- **Eliminated checksum (*Why?*)**
- **New options mechanism (next header) (*Why?*)**
- **Eliminated header length (*Why?*)**

IPv6: Summary of Changes

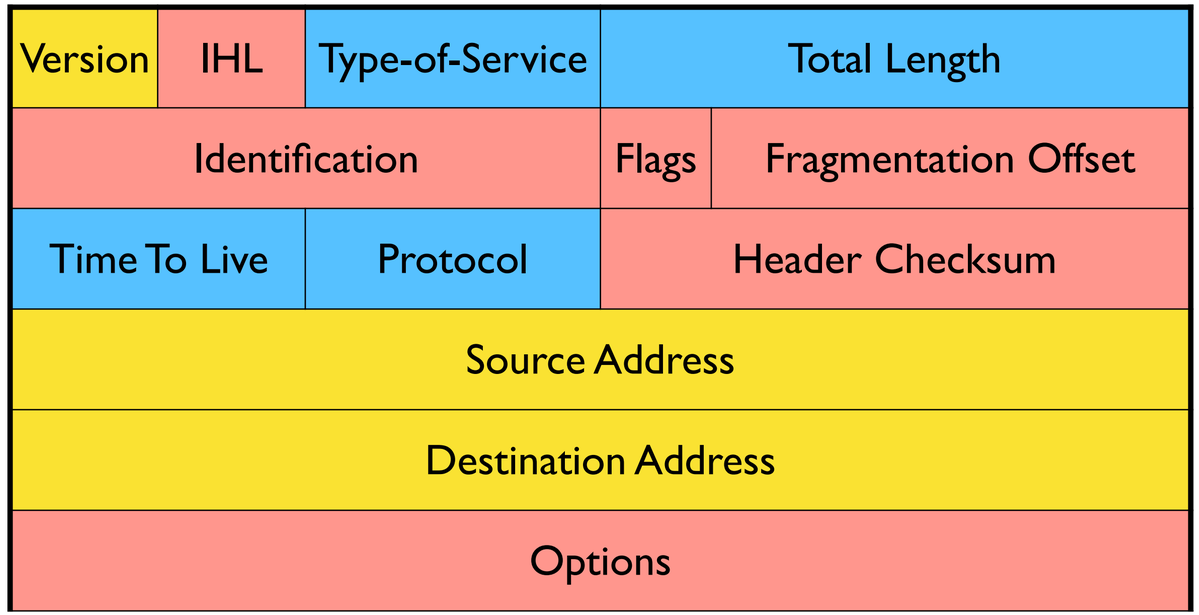
- **Eliminated fragmentation (*Why?*)**
- **Eliminated checksum (*Why?*)**
- **New options mechanism (next header) (*Why?*)**
- **Eliminated header length (*Why?*)**
- **Expanded addresses**

IPv6: Summary of Changes

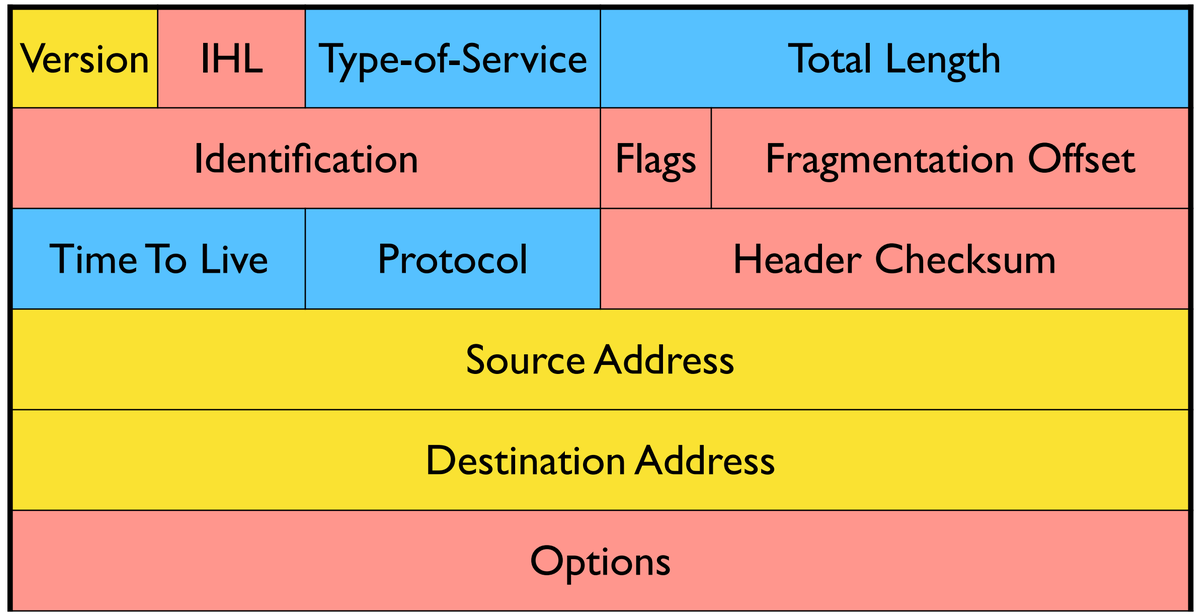
- **Eliminated fragmentation (*Why?*)**
- **Eliminated checksum (*Why?*)**
- **New options mechanism (next header) (*Why?*)**
- **Eliminated header length (*Why?*)**
- **Expanded addresses**
- **Added Flow Label**

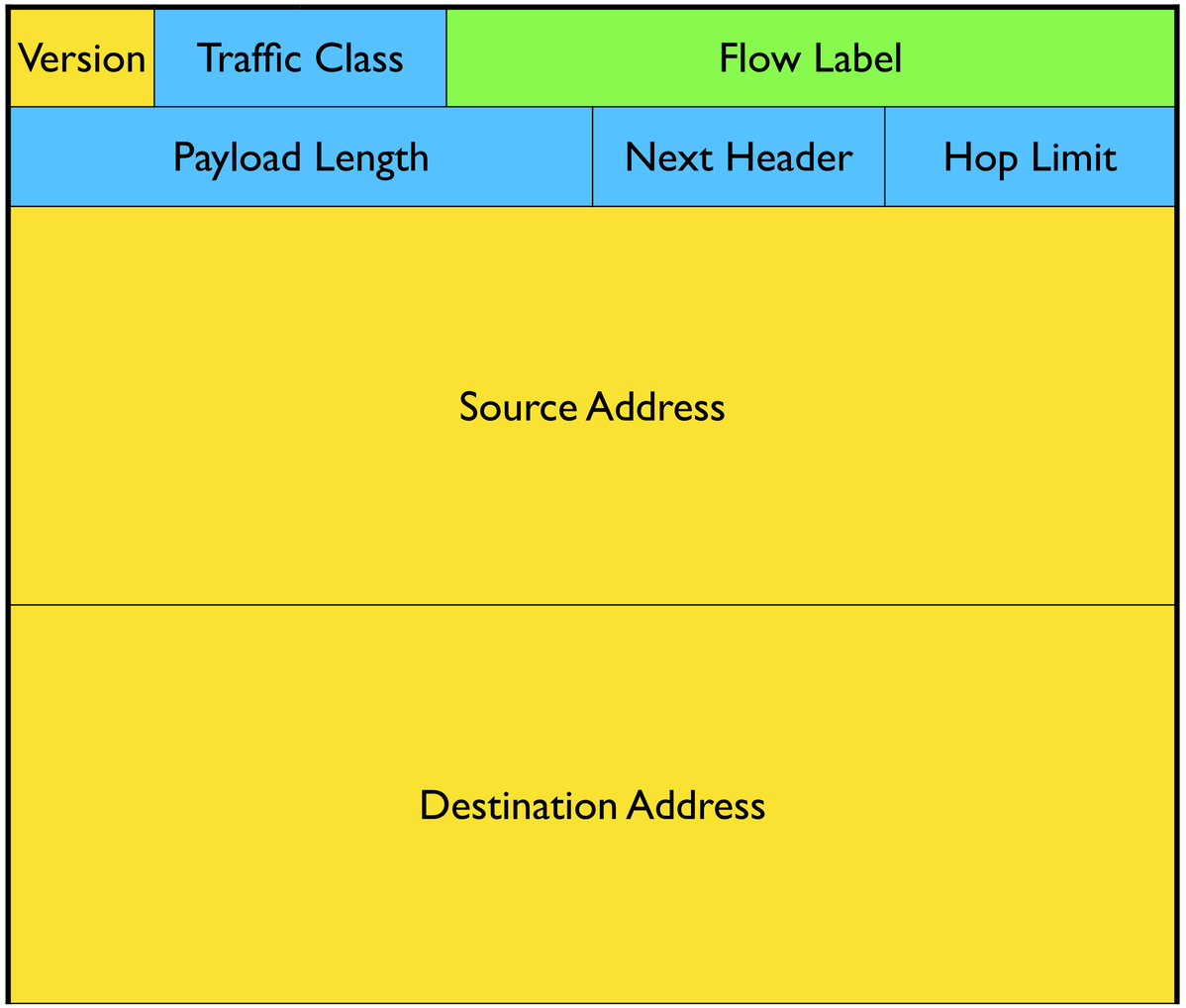
IPv4 and IPv6 Header Comparison

IPv4 and IPv6 Header Comparison

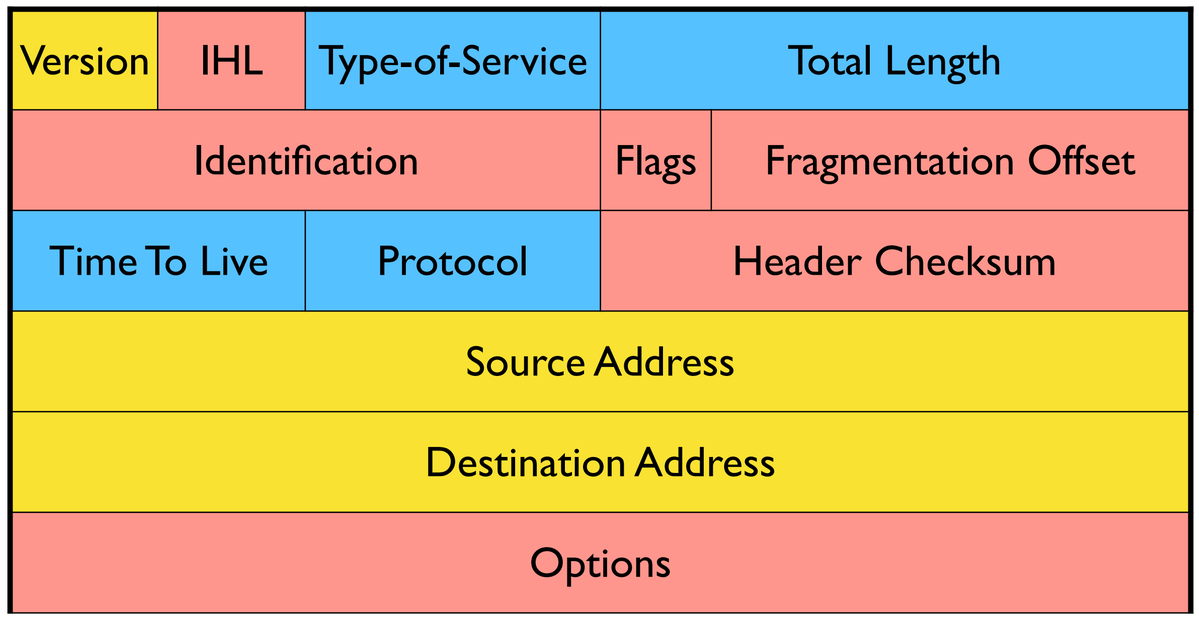


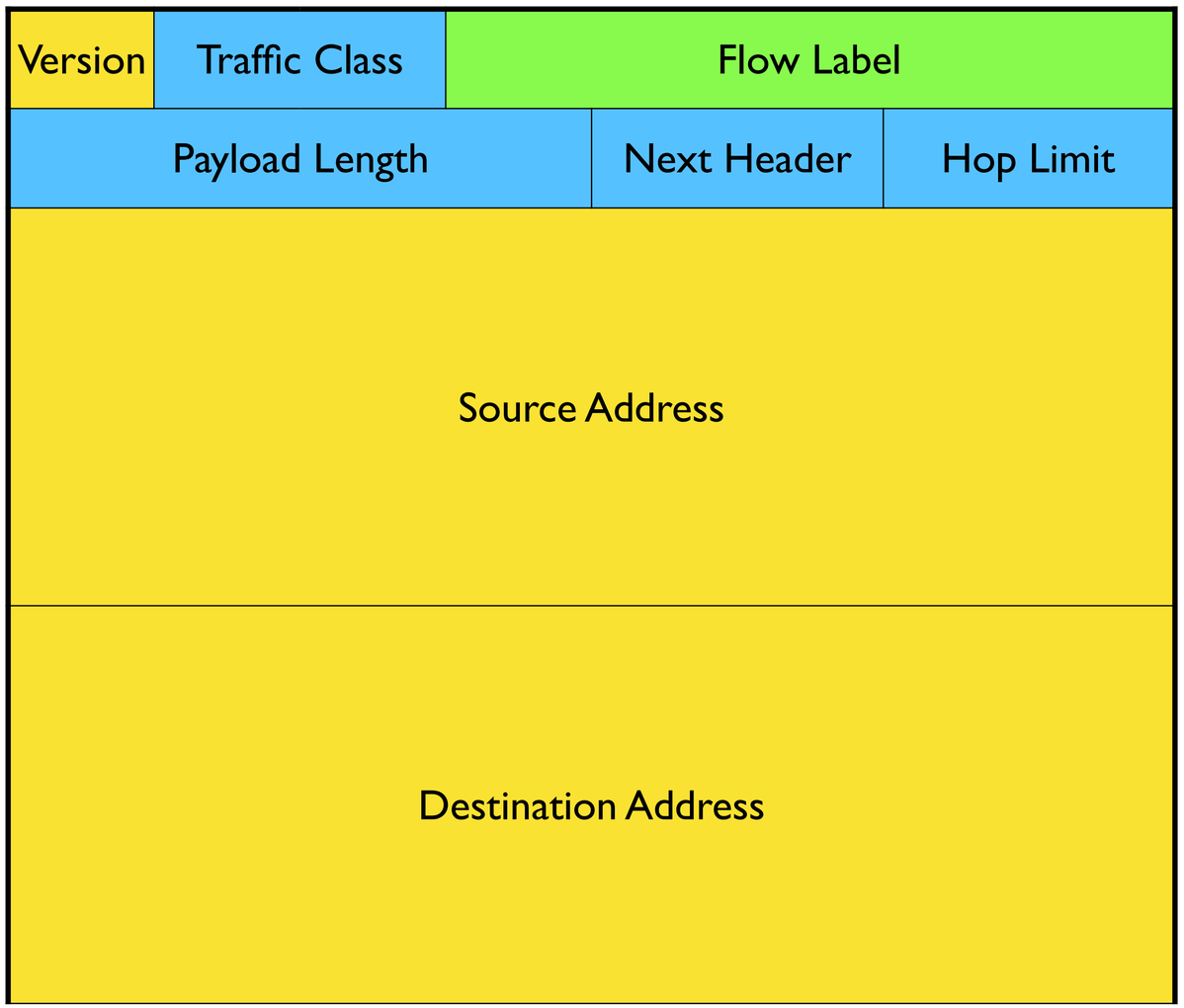
IPv4 and IPv6 Header Comparison

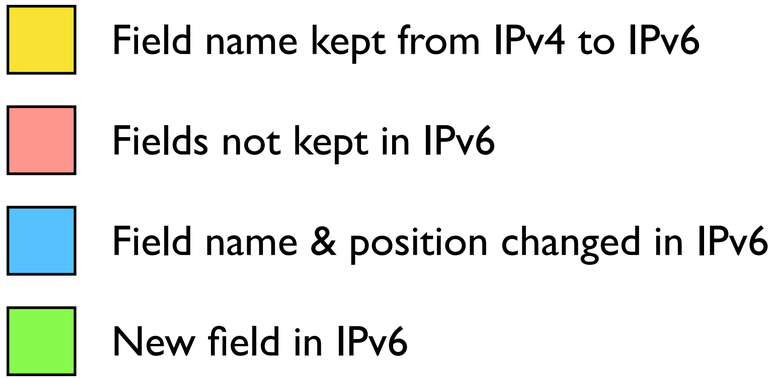




IPv4 and IPv6 Header Comparison







IPv6: Philosophy of changes?

IPv6: Philosophy of changes?

- **Don’t deal with problems: leave to ends**
  - Eliminated fragmentation
  - Eliminated checksum
  - *Why retain TTL?*

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?*

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

Summary: IP header design

Summary: IP header design

- **More nuanced than it first seems!**

Summary: IP header design

- **More nuanced than it first seems!**
- **Must juggle multiple goals**
  - Robustness
  - Efficiency
  - Security
  - Completeness

Summary: IP header design

- **More nuanced than it first seems!**
- **Must juggle multiple goals**
  - Robustness
  - Efficiency
  - Security
  - Completeness
- **Plus feature interactions**
  - E.g., what happens to IP options when we fragment?

Summary: IP header design

- **More nuanced than it first seems!**
- **Must juggle multiple goals**
  - Robustness
  - Efficiency
  - Security
  - Completeness
- **Plus feature interactions**
  - E.g., what happens to IP options when we fragment?
- **And future evolution**

IP Routers

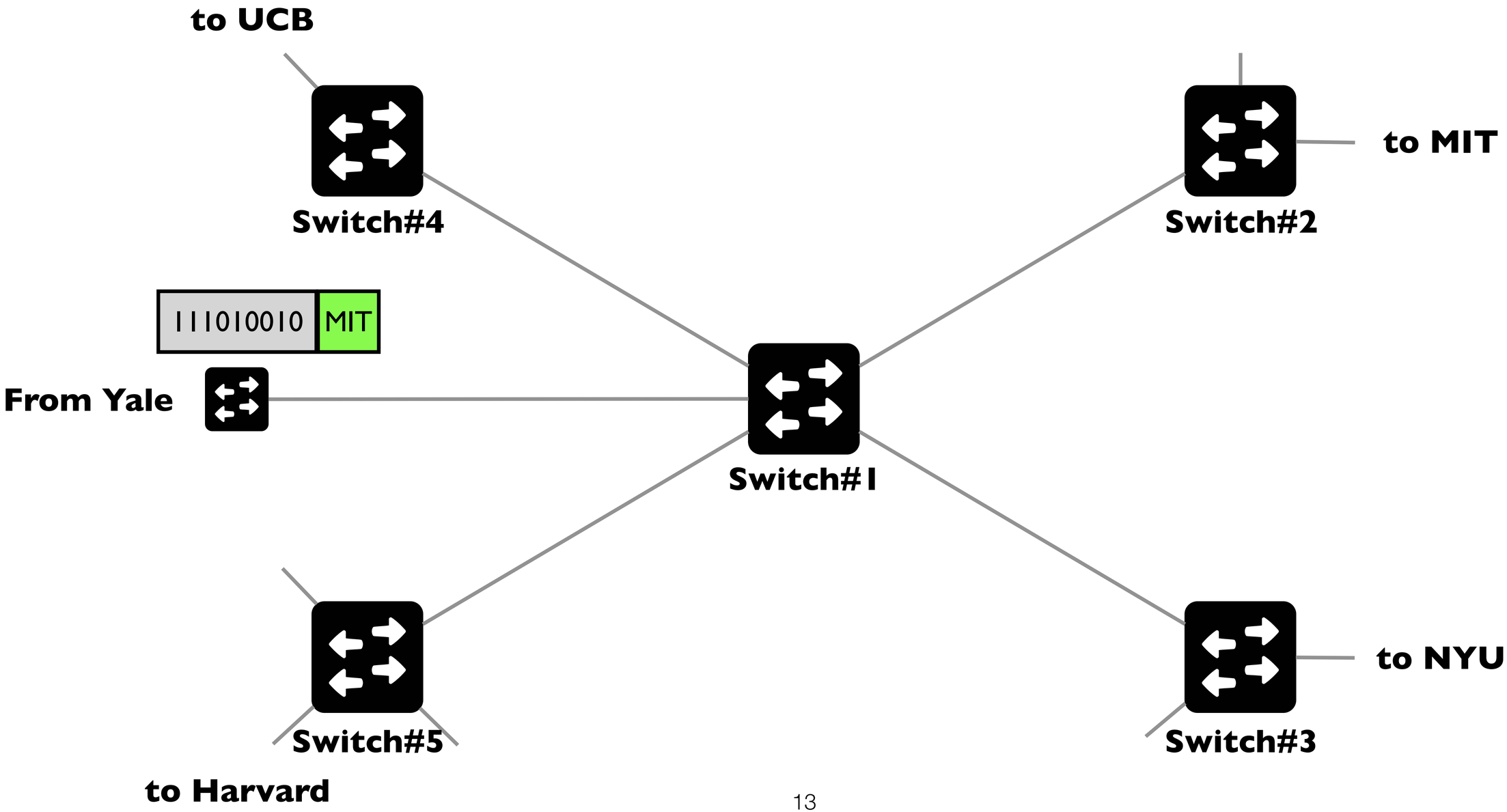
Context

- **Control Plane**
  - How to route traffic to each possible destination
  - Jointly computed using IGP (OSPF, RIP, etc.) and BGP
- **Data Plane**
  - **So Far:** Necessary fields in IP header of each packet for *end-system & routers*
  - **Rest of today's class:** How IP routers forward packets

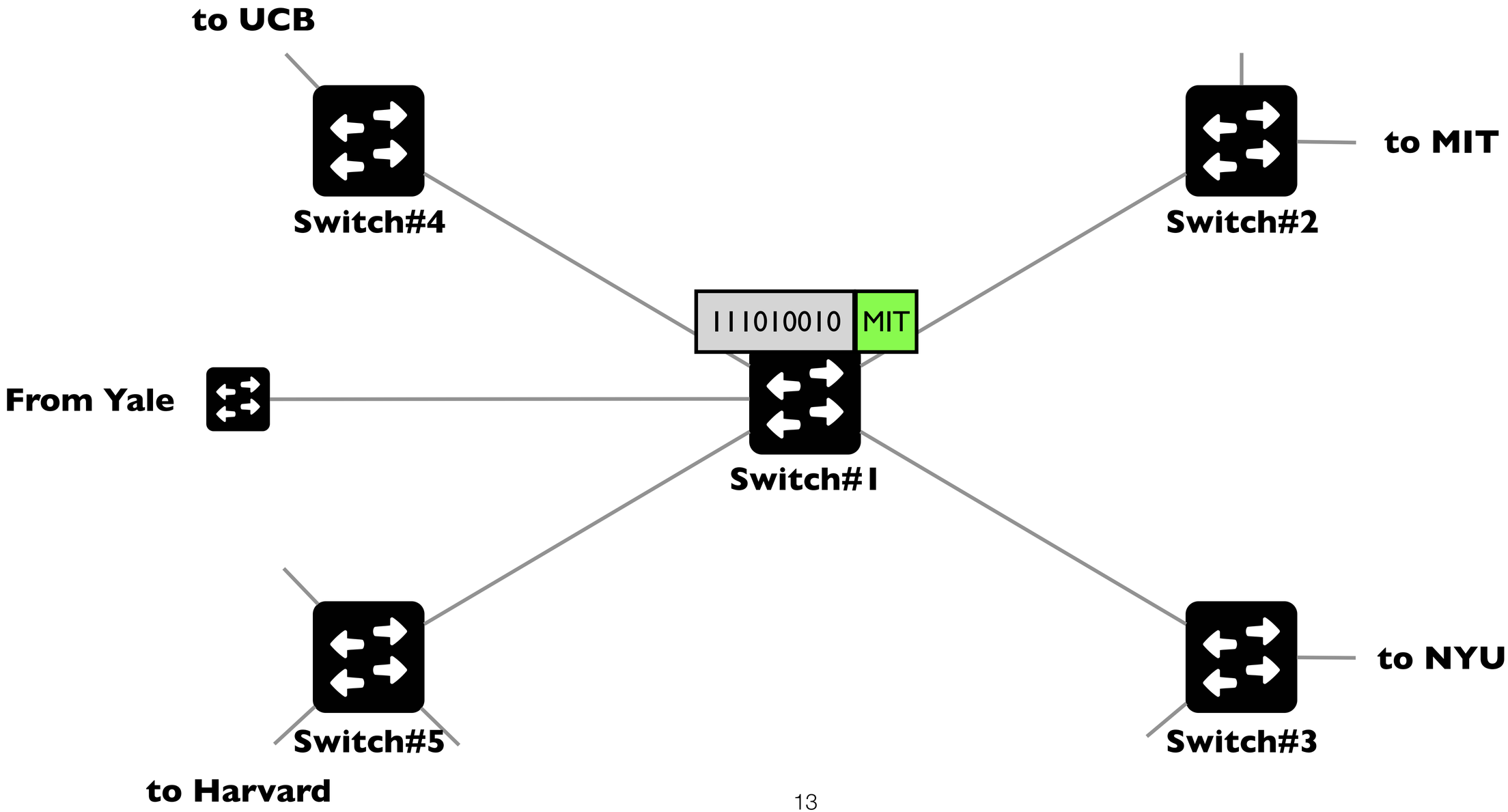
IP Routers

- **Core building block of the Internet infrastructure**
- **$120B+ industry!**
- **Vendors:** Cisco, Juniper, Huawei, HPE, Arista, Dell, EMC, Nokia (merged with Alcatel-Lucent)
  - *account for > 90%*

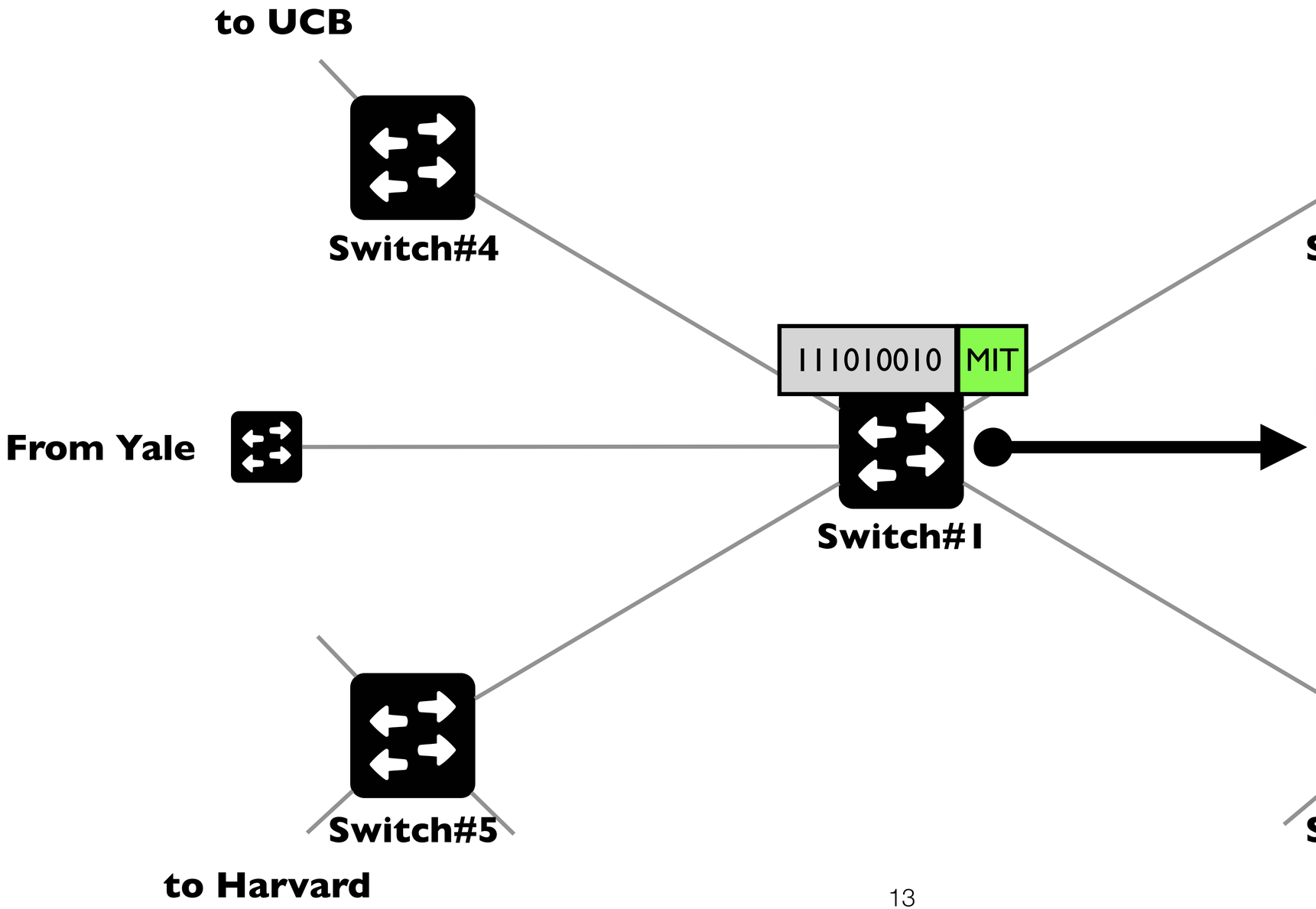
Lecture#4: Routers Forward Packets



Lecture#4: Routers Forward Packets



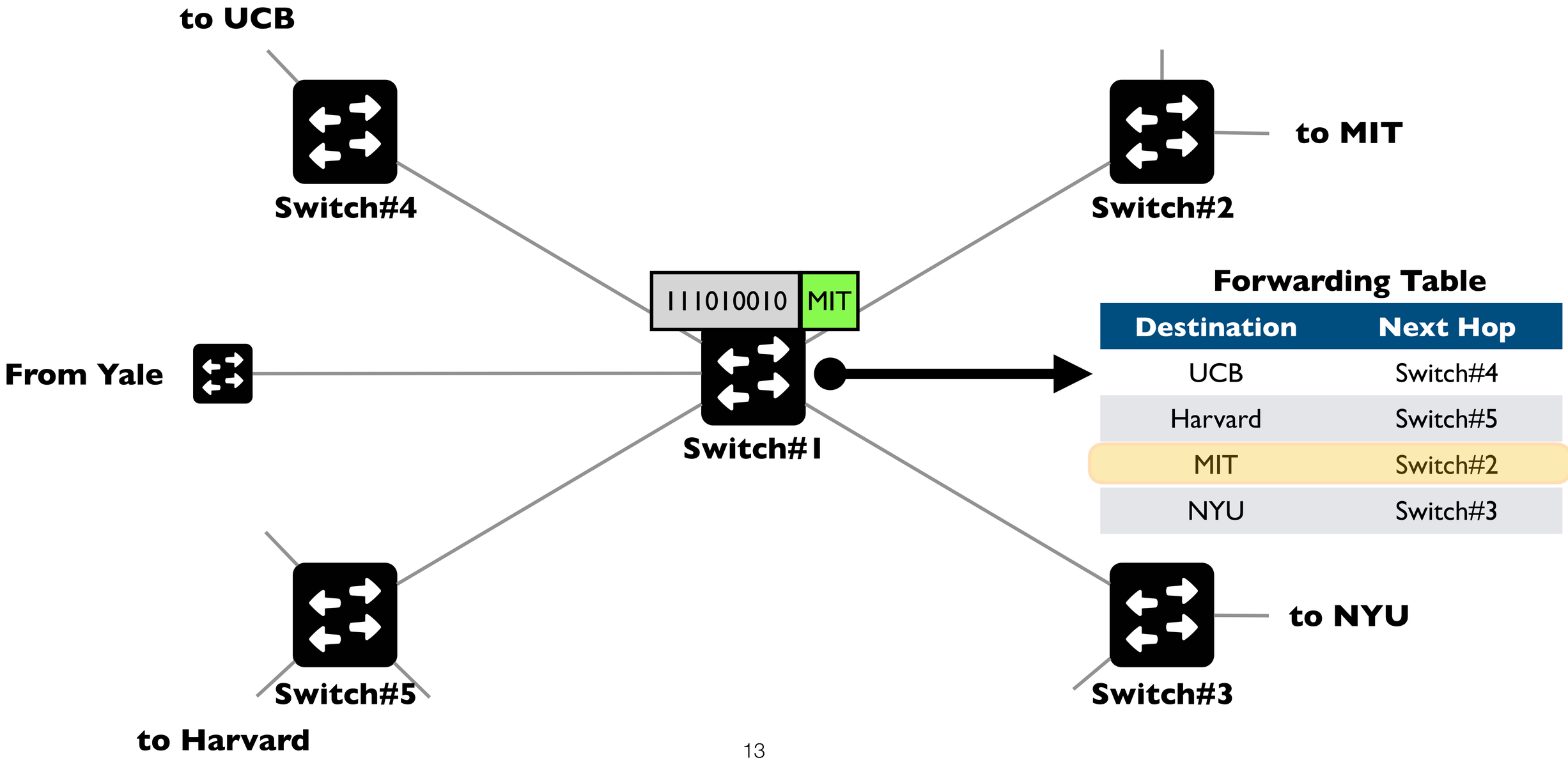
Lecture#4: Routers Forward Packets



Forwarding Table

Destination	Next Hop
UCB	Switch#4
Harvard	Switch#5
MIT	Switch#2
NYU	Switch#3

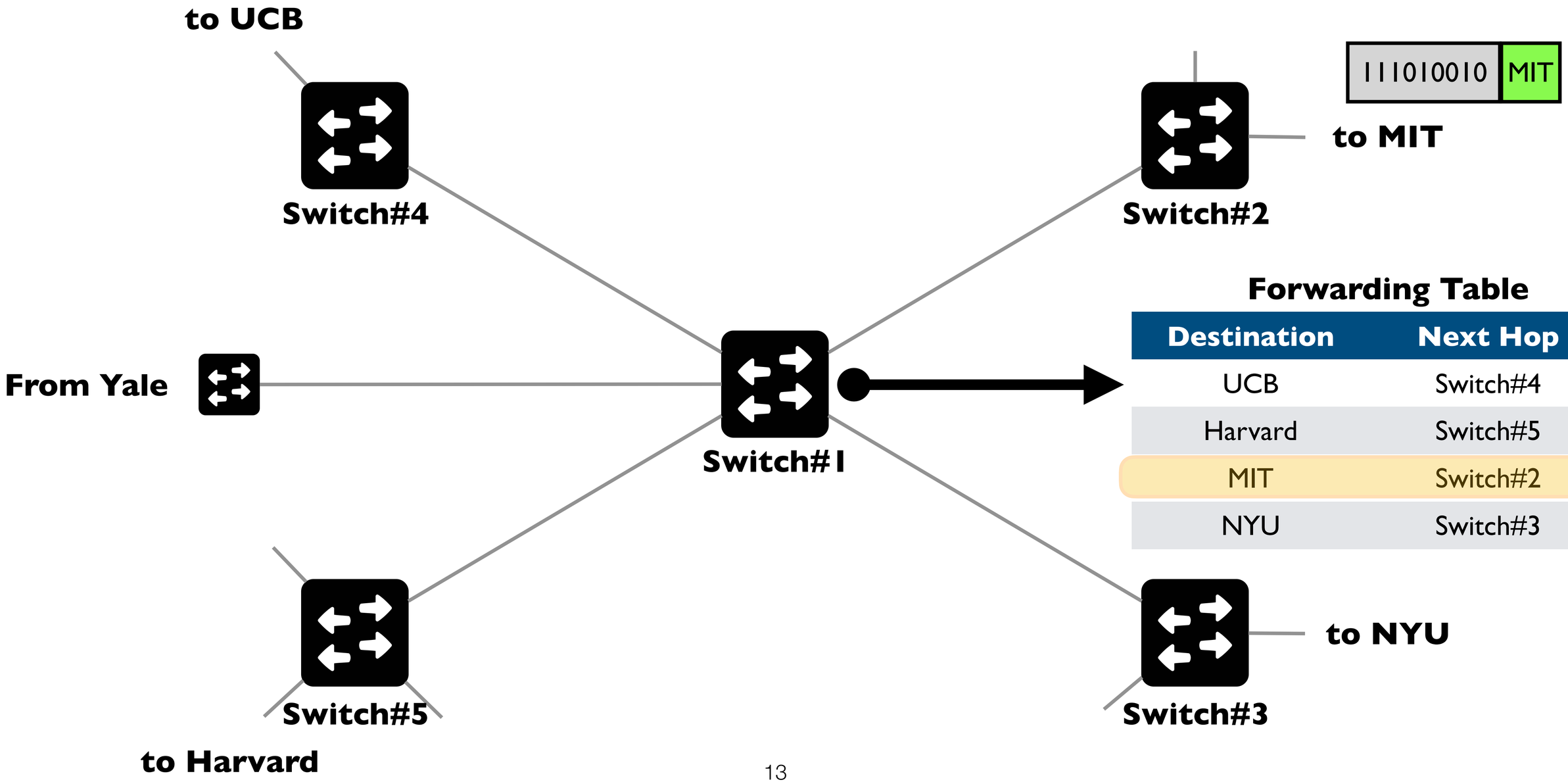
Lecture#4: Routers Forward Packets



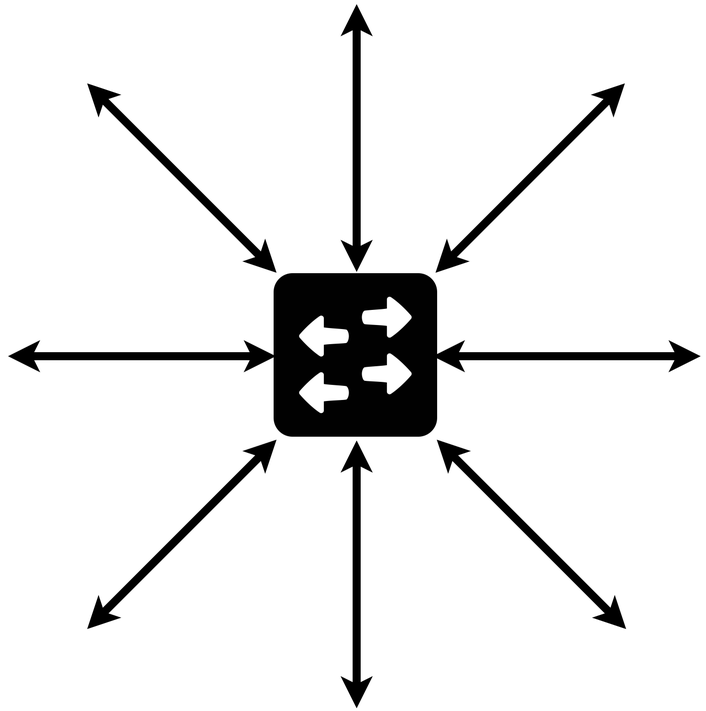
Forwarding Table

Destination	Next Hop
UCB	Switch#4
Harvard	Switch#5
MIT	Switch#2
NYU	Switch#3

Lecture#4: Routers Forward Packets

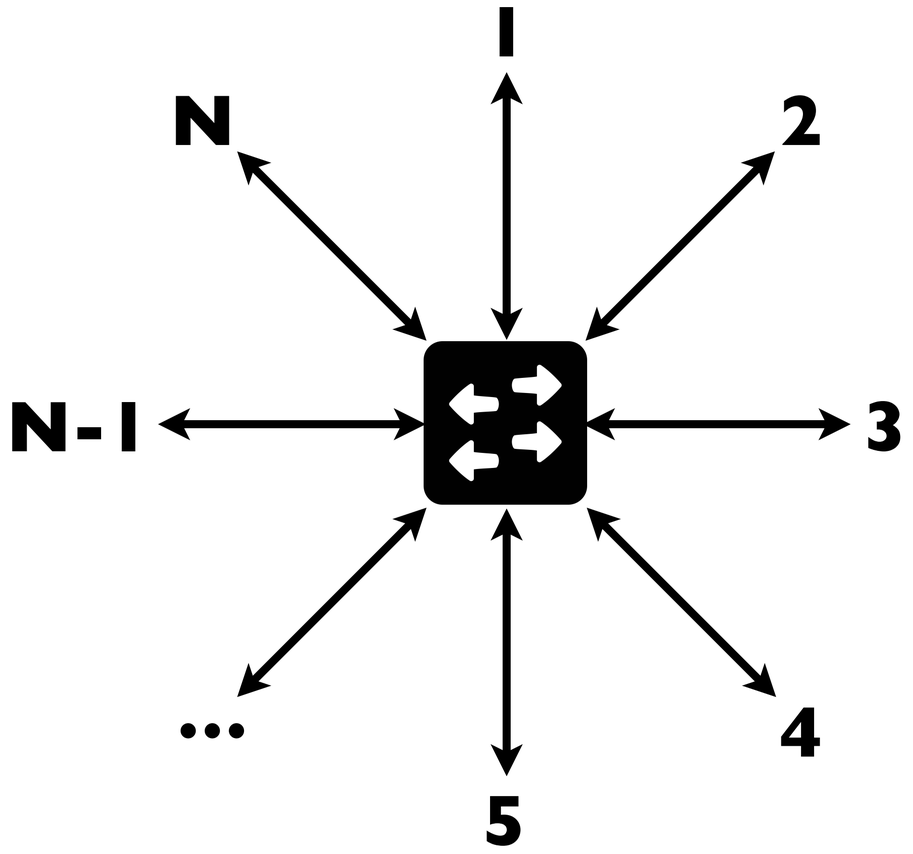


Router Definitions



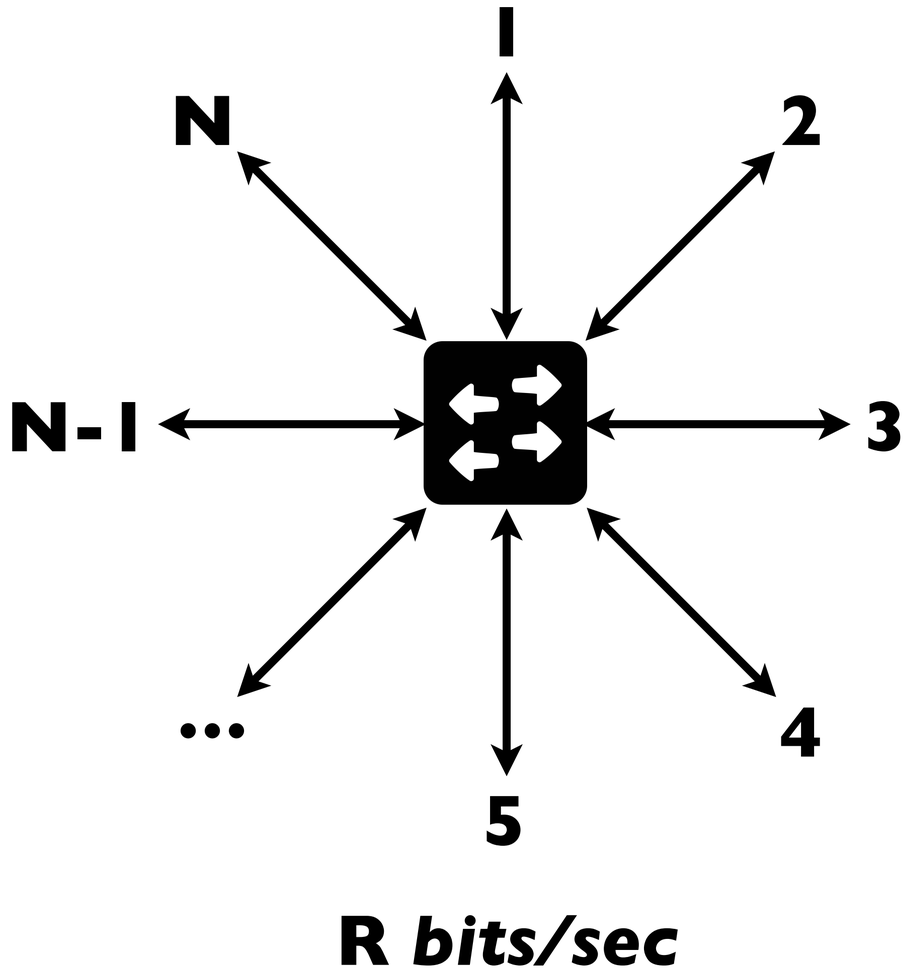
Router Definitions

- **N** = number of external router "ports"



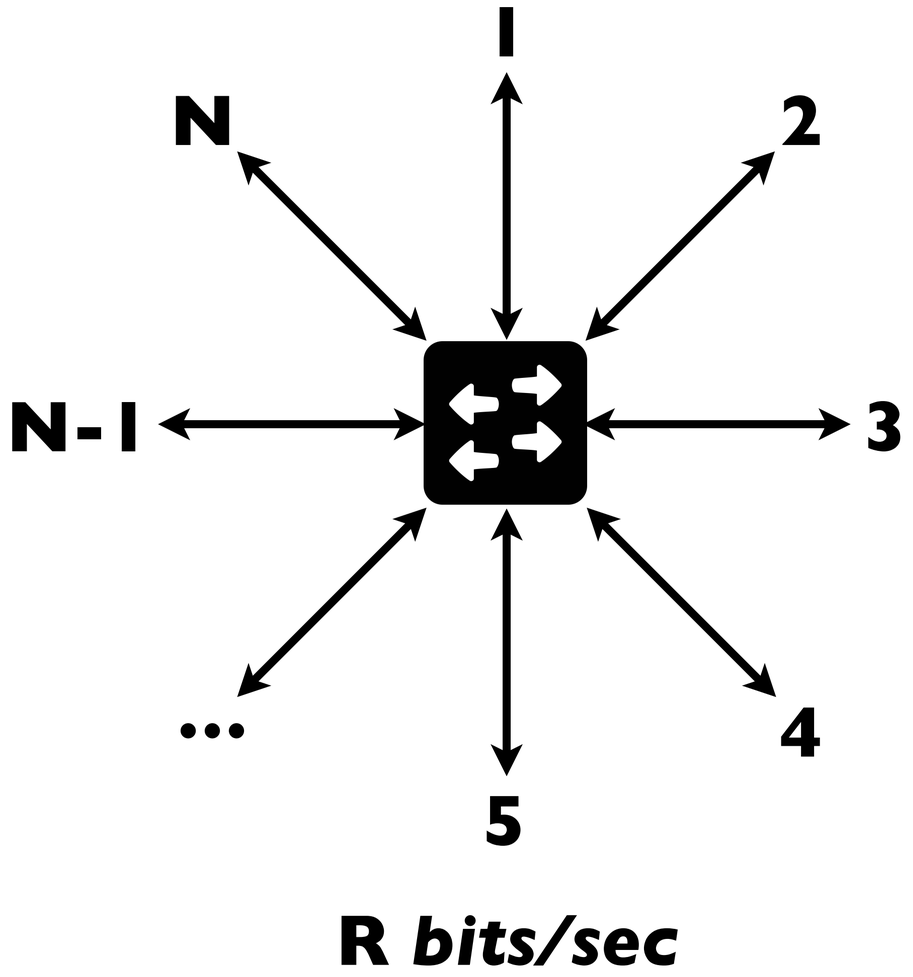
Router Definitions

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

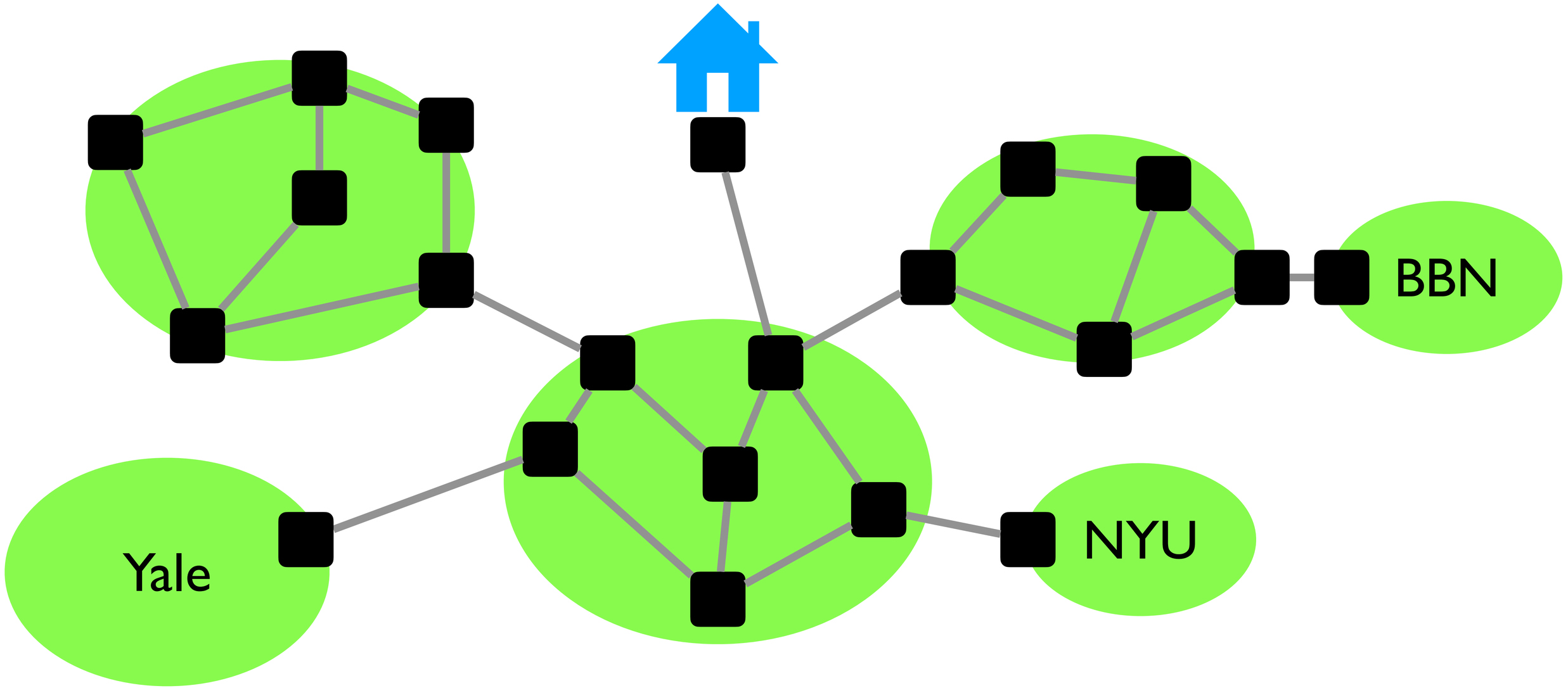


Router Definitions

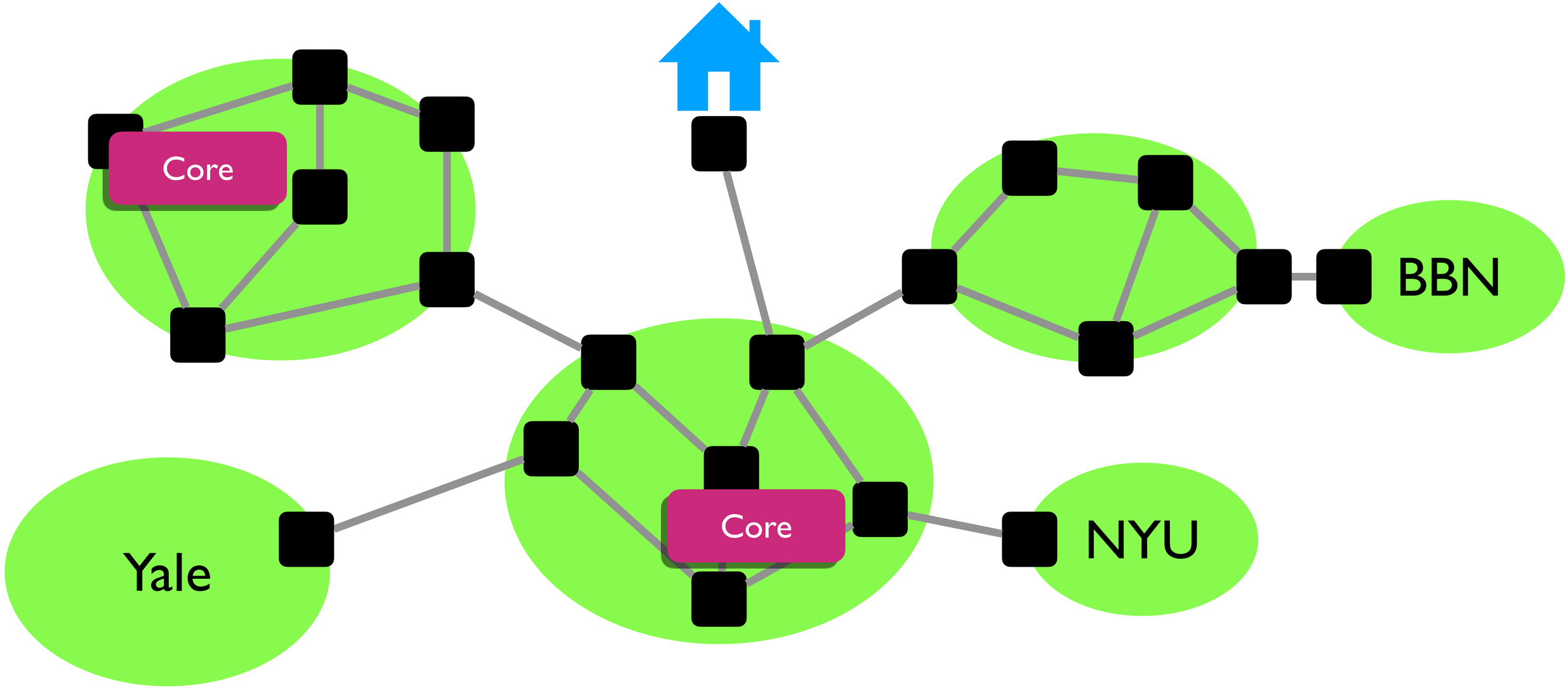
- **N** = number of external router "ports"
- **R** = speed ("line rate") of a port
- Router capacity = **N x R**



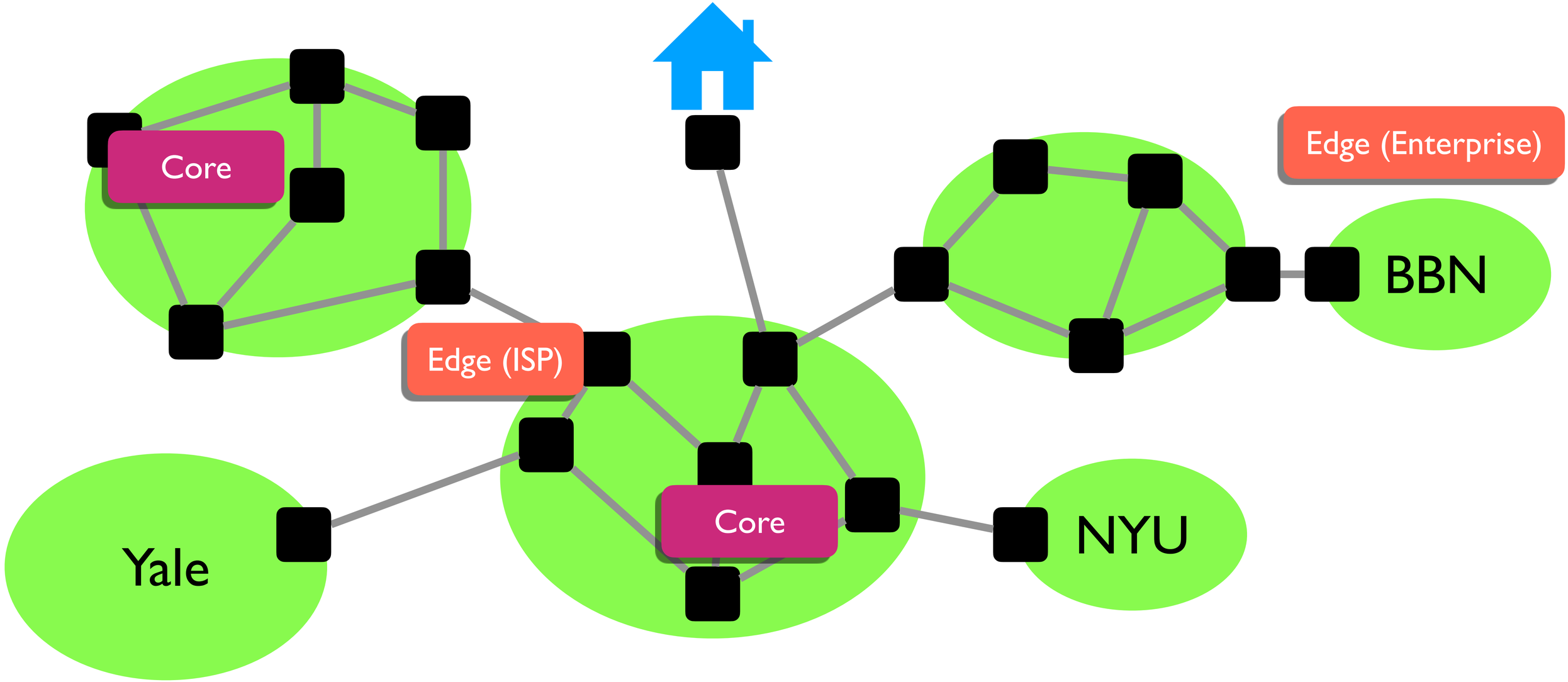
Networks & Routers



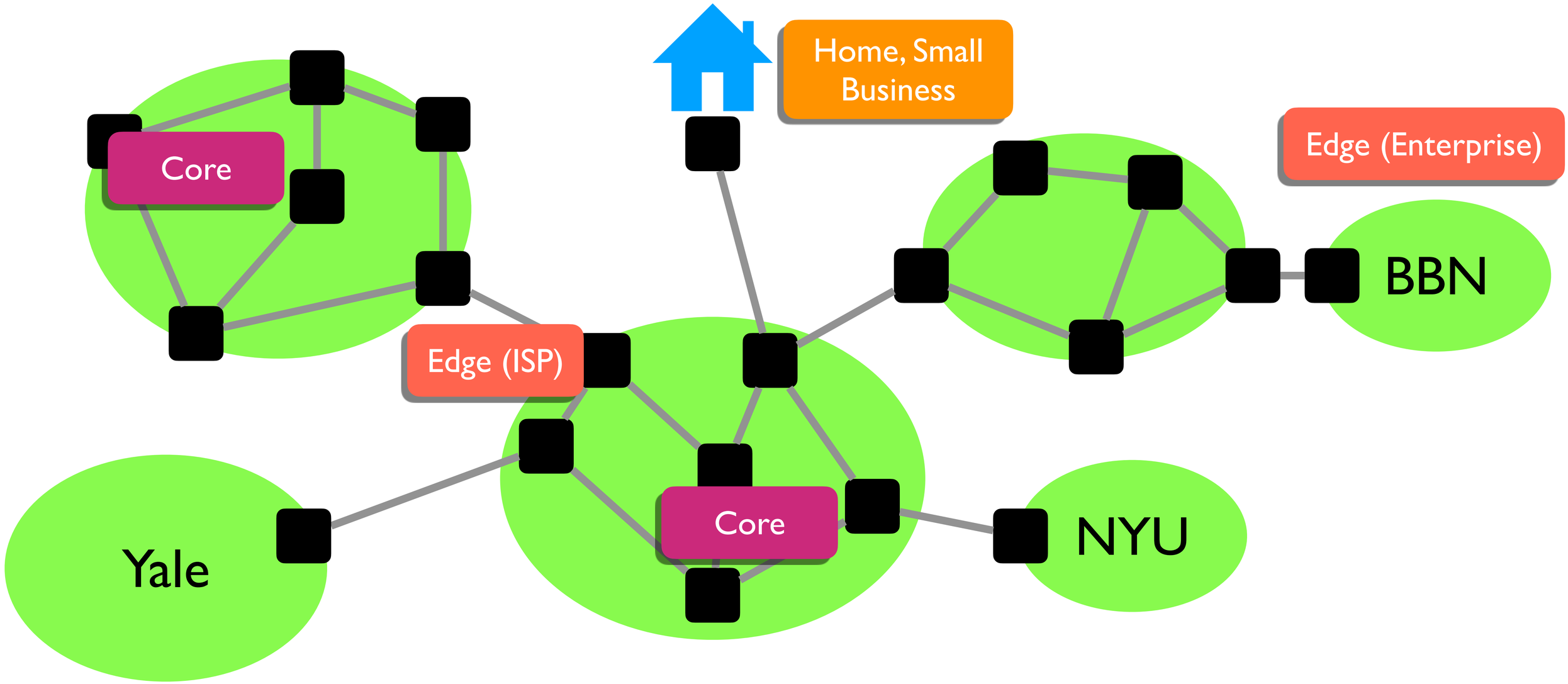
Networks & Routers



Networks & Routers

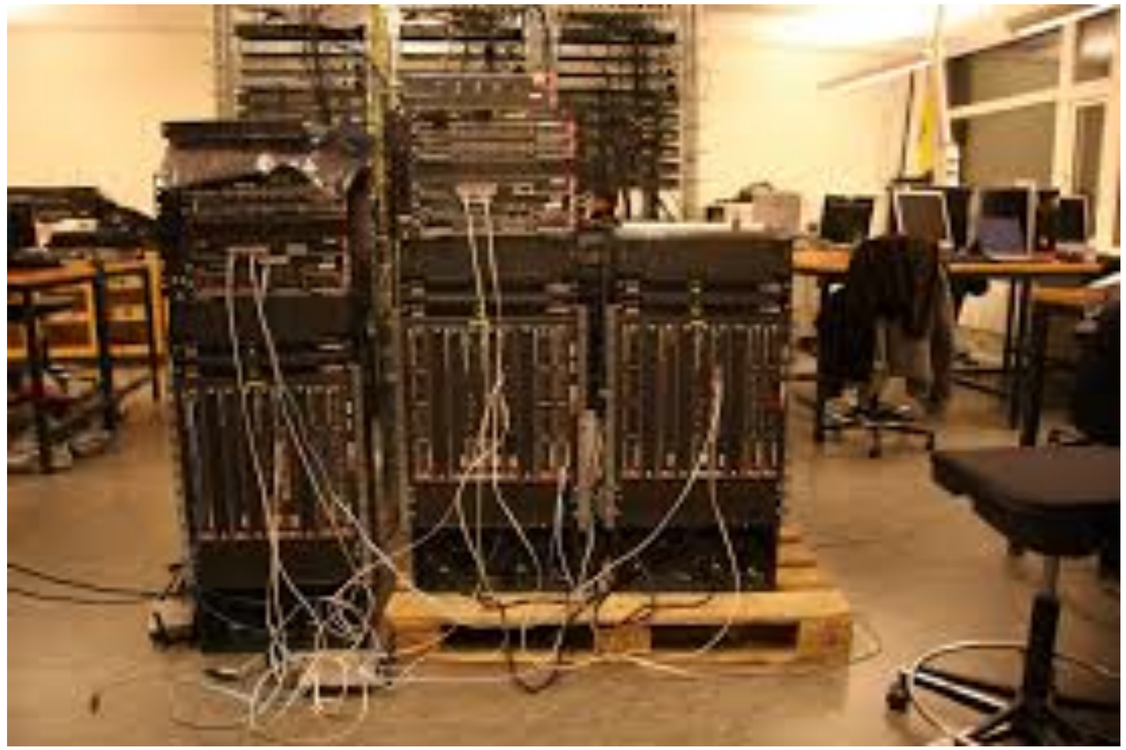


Networks & Routers



Examples of Routers (Core)

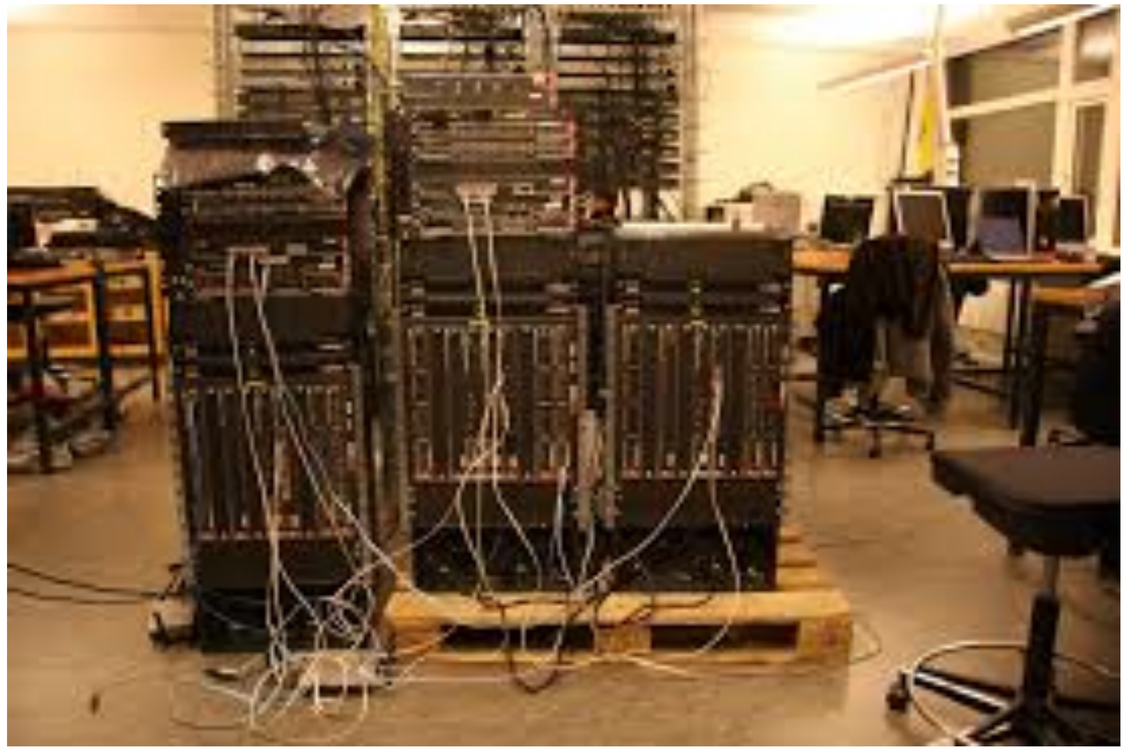
- **Cisco CRS**
  - **R** = 10/40/100 Gbps
  - **N x R** = 922 Tbps



72 racks, >1MW

Examples of Routers (Core)

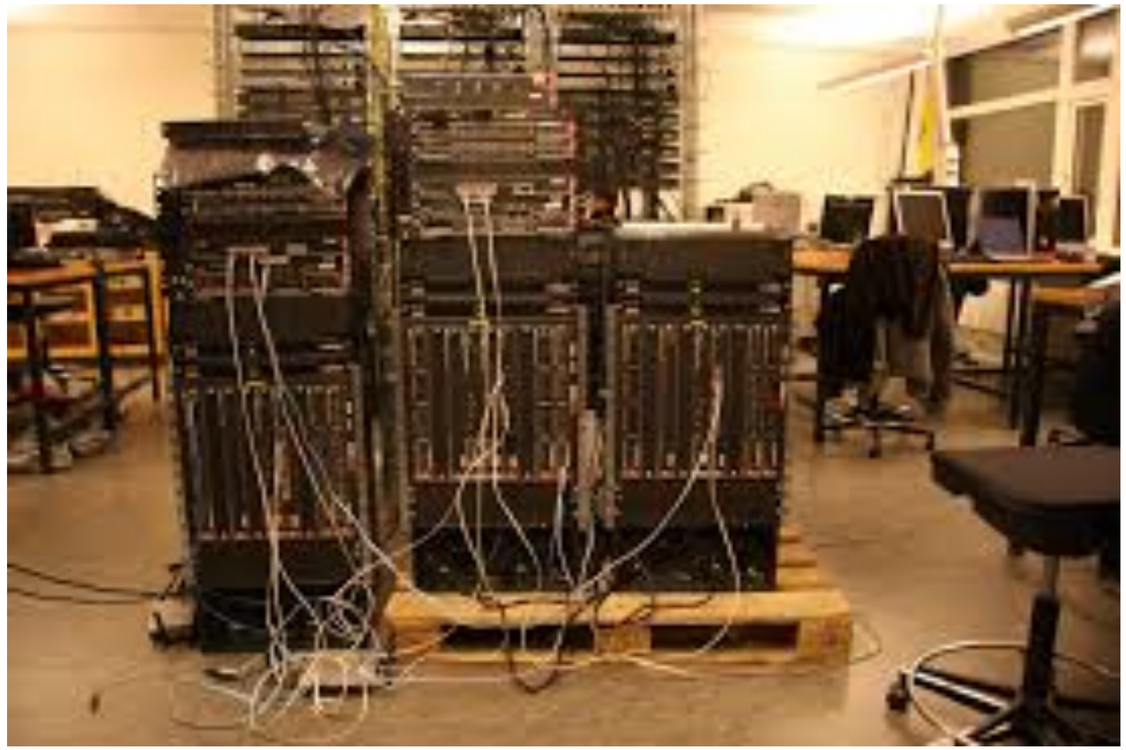
- **Cisco CRS**
  - **R** = 10/40/100 Gbps
  - **N x R** = 922 Tbps
  - **Netflix:** 0.7GB per hour (1.5 Mb/s)



72 racks, >1MW

Examples of Routers (Core)

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



72 racks, >1MW

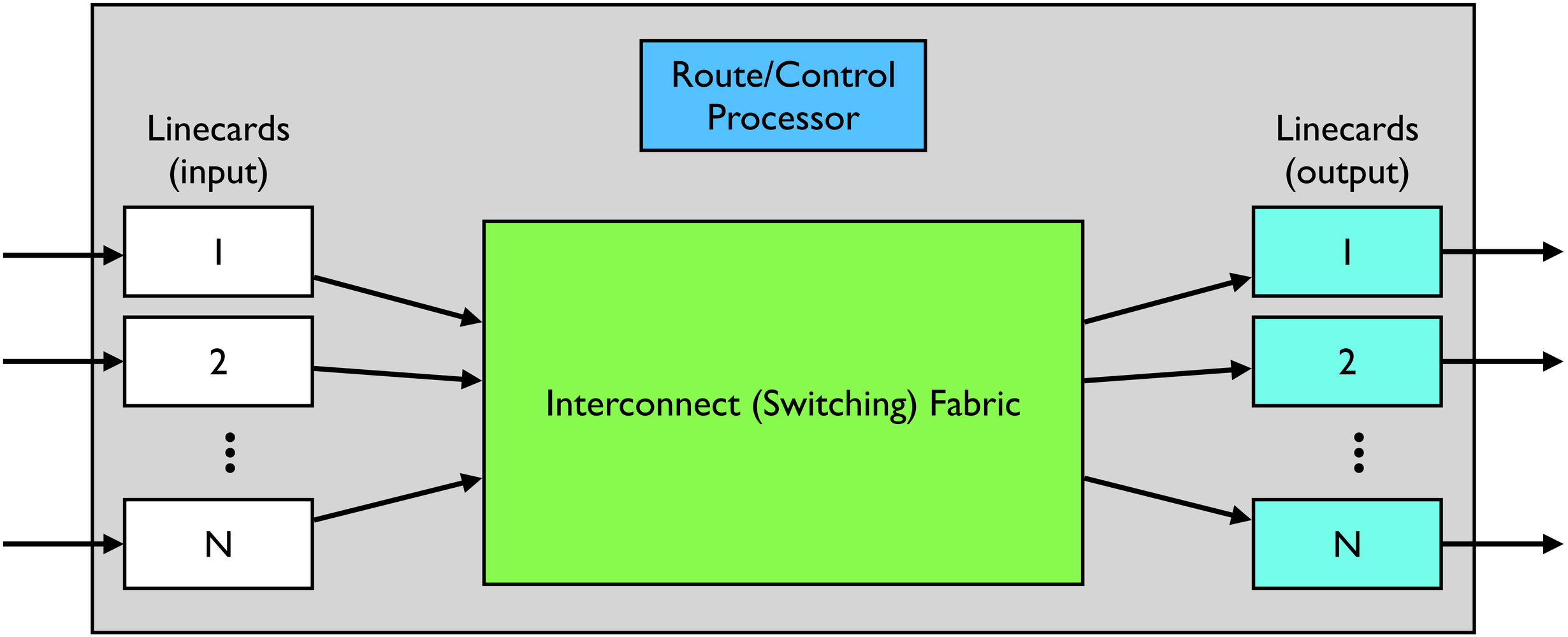
Examples of Routers (Edge)

- **Cisco ASR**
  - **R** = 1/10/40 Gbps
  - **N x R** = 120 Gbps

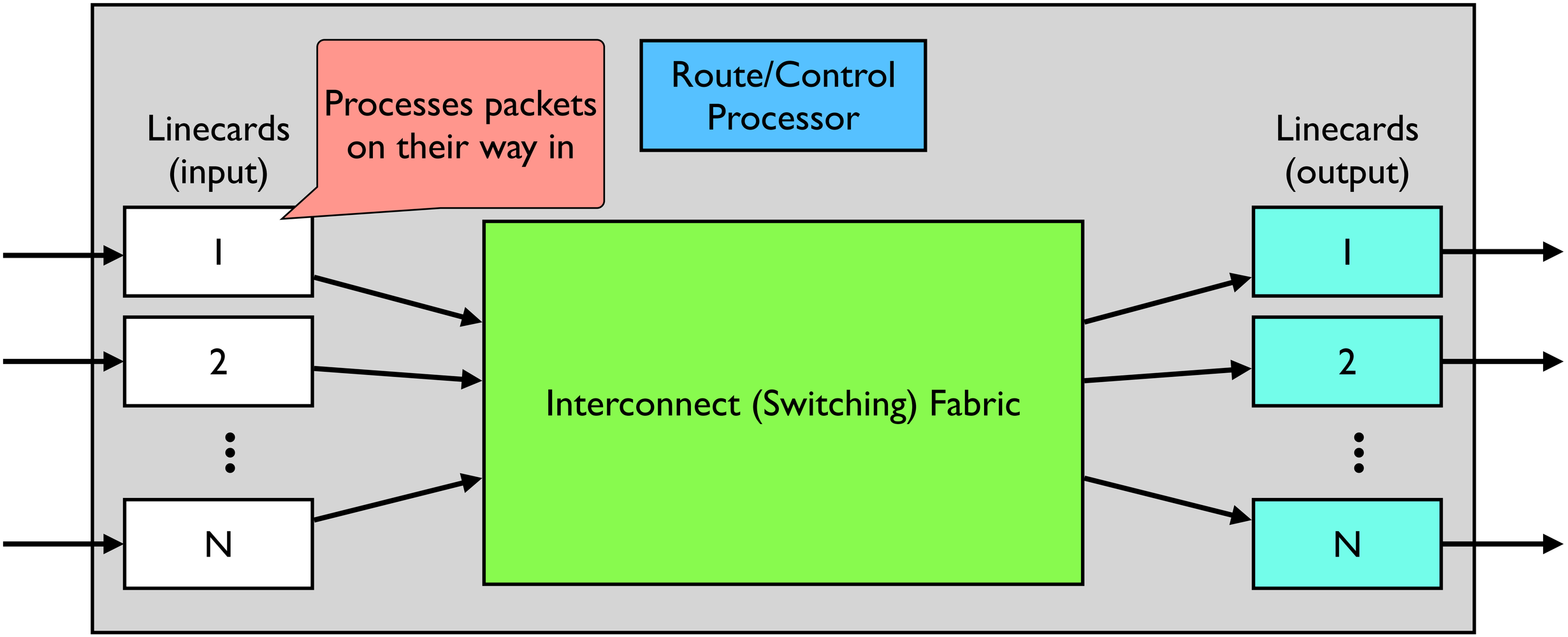
Examples of Routers (Small Business)

- **Cisco 3945E**
  - **R** = 10/100/1000 Mbps
  - **N x R** <10 Gbps

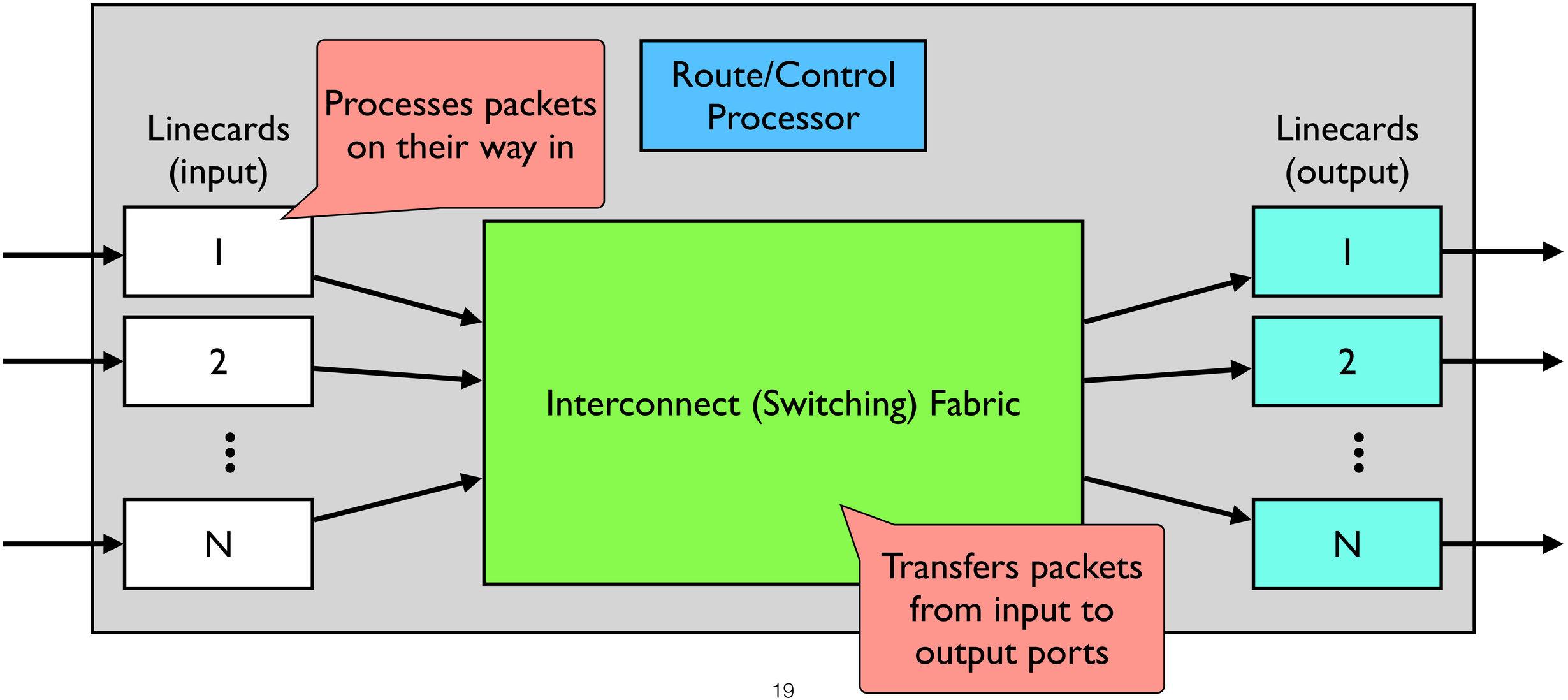
What's inside a router?



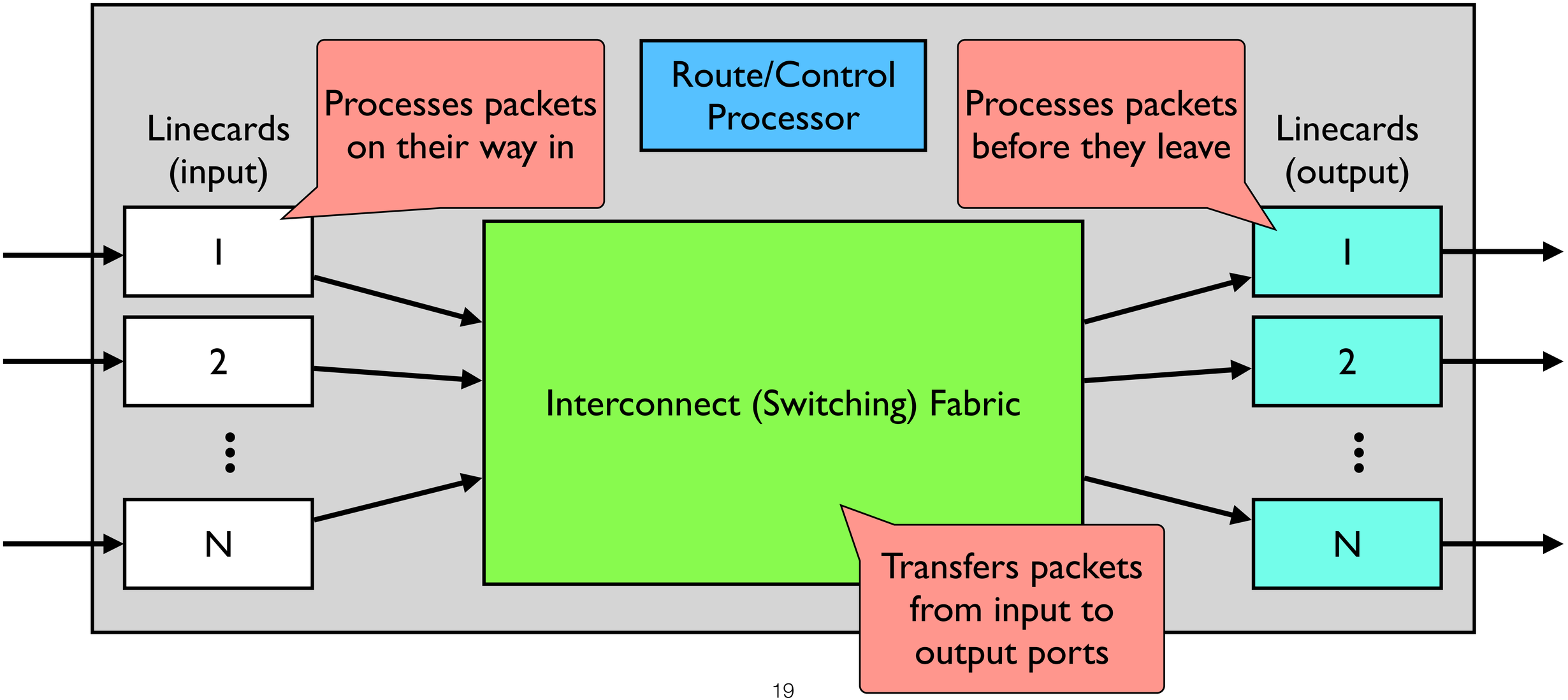
What's inside a router?



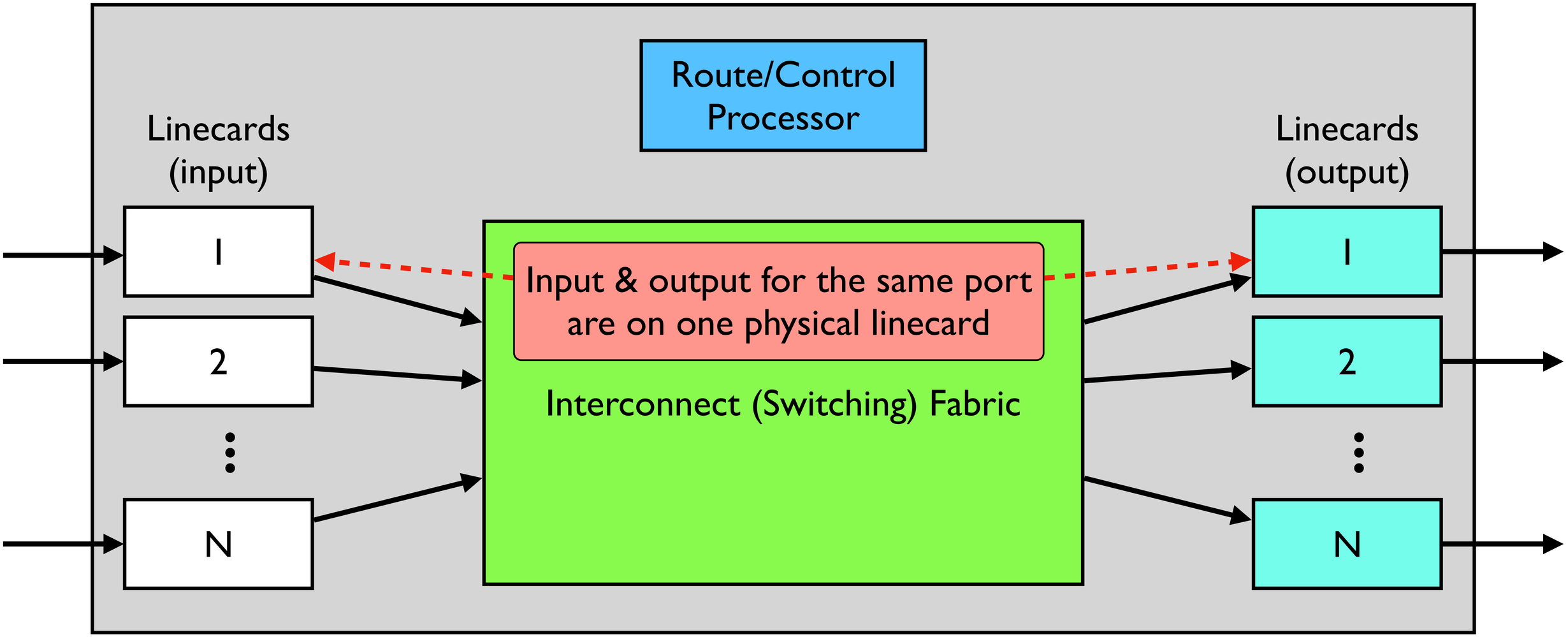
What's inside a router?



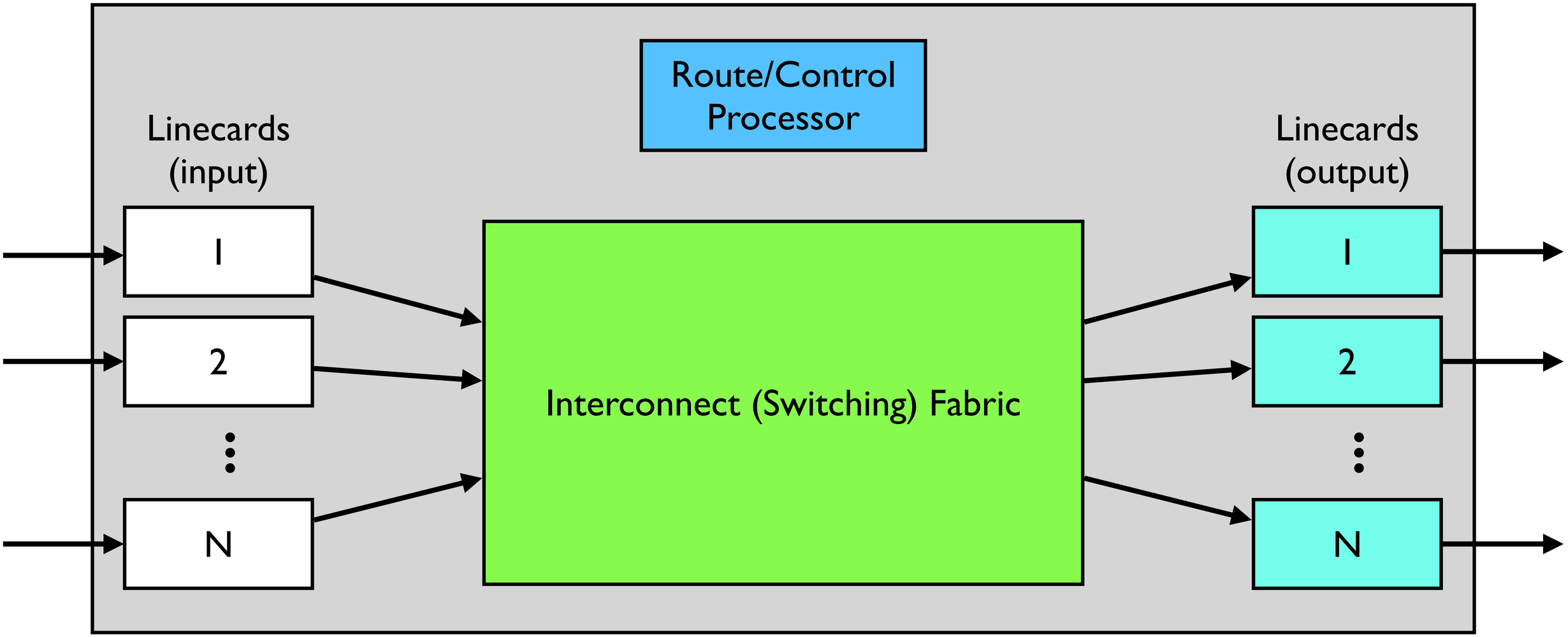
What's inside a router?



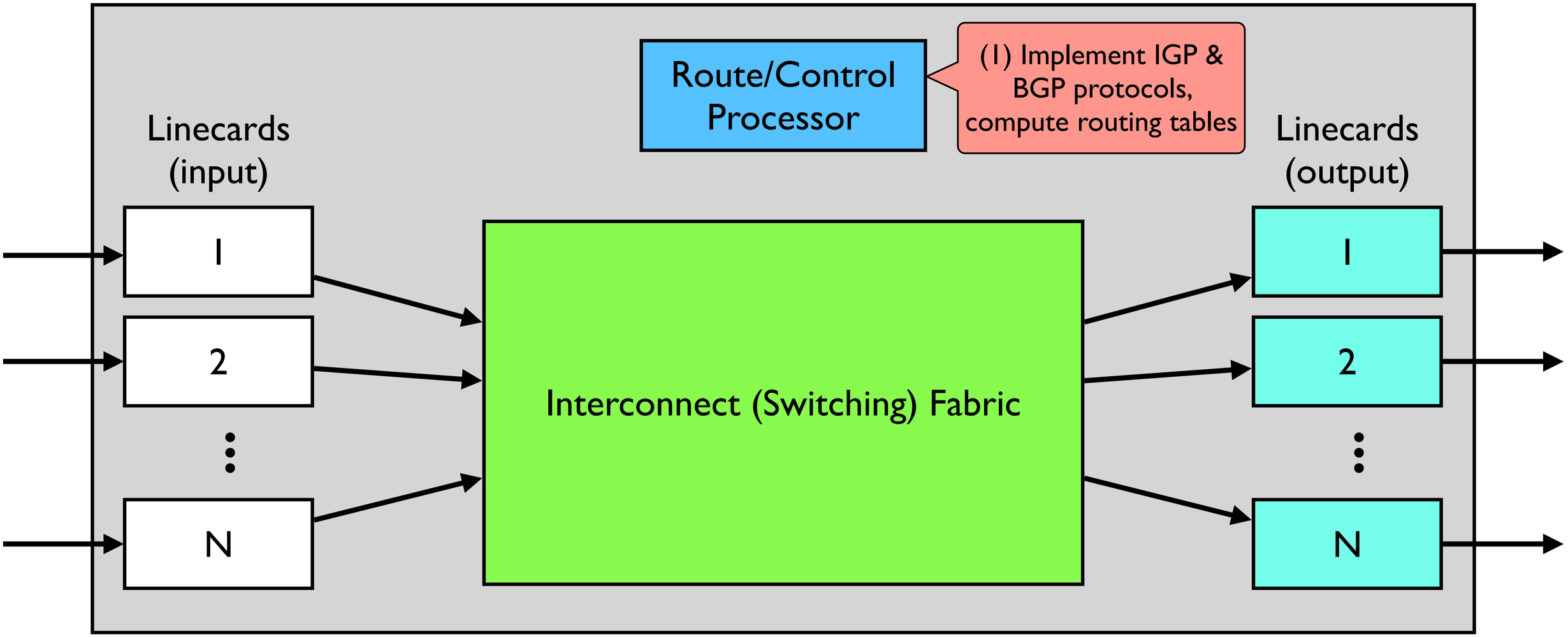
What's inside a router?



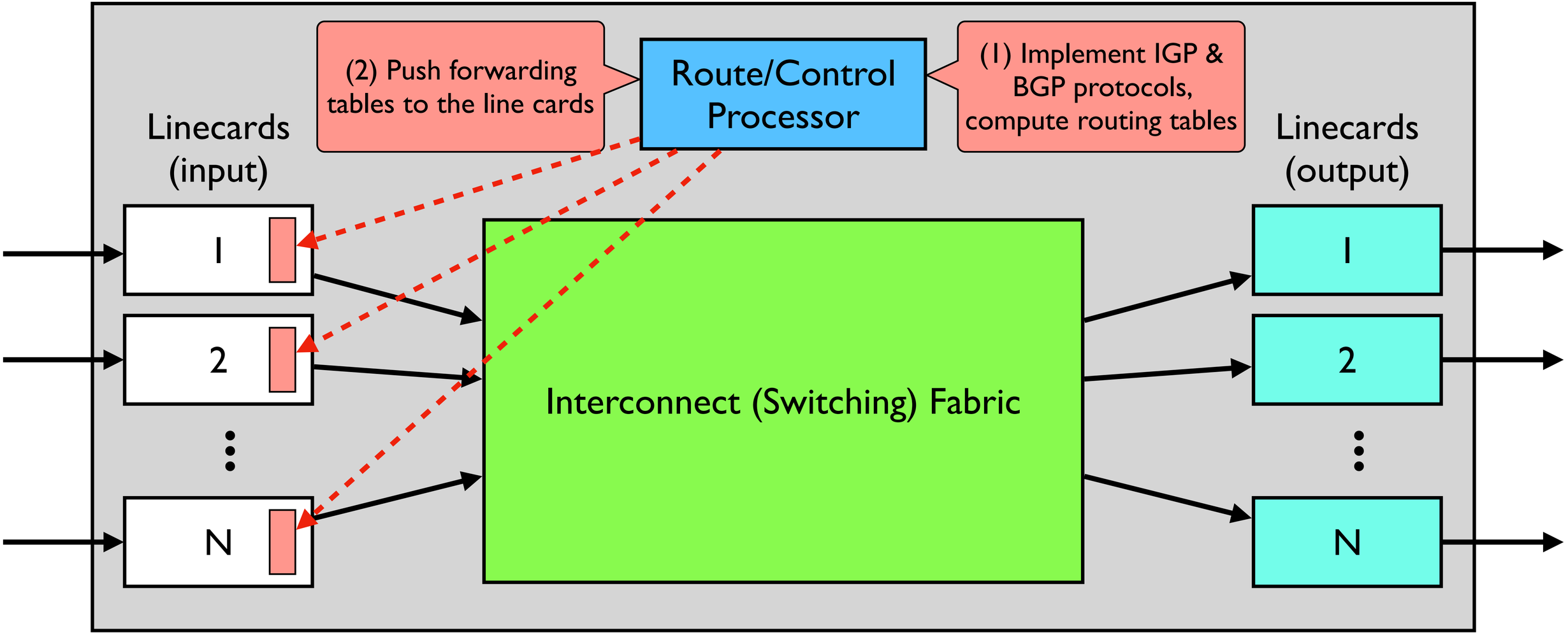
What's inside a router?



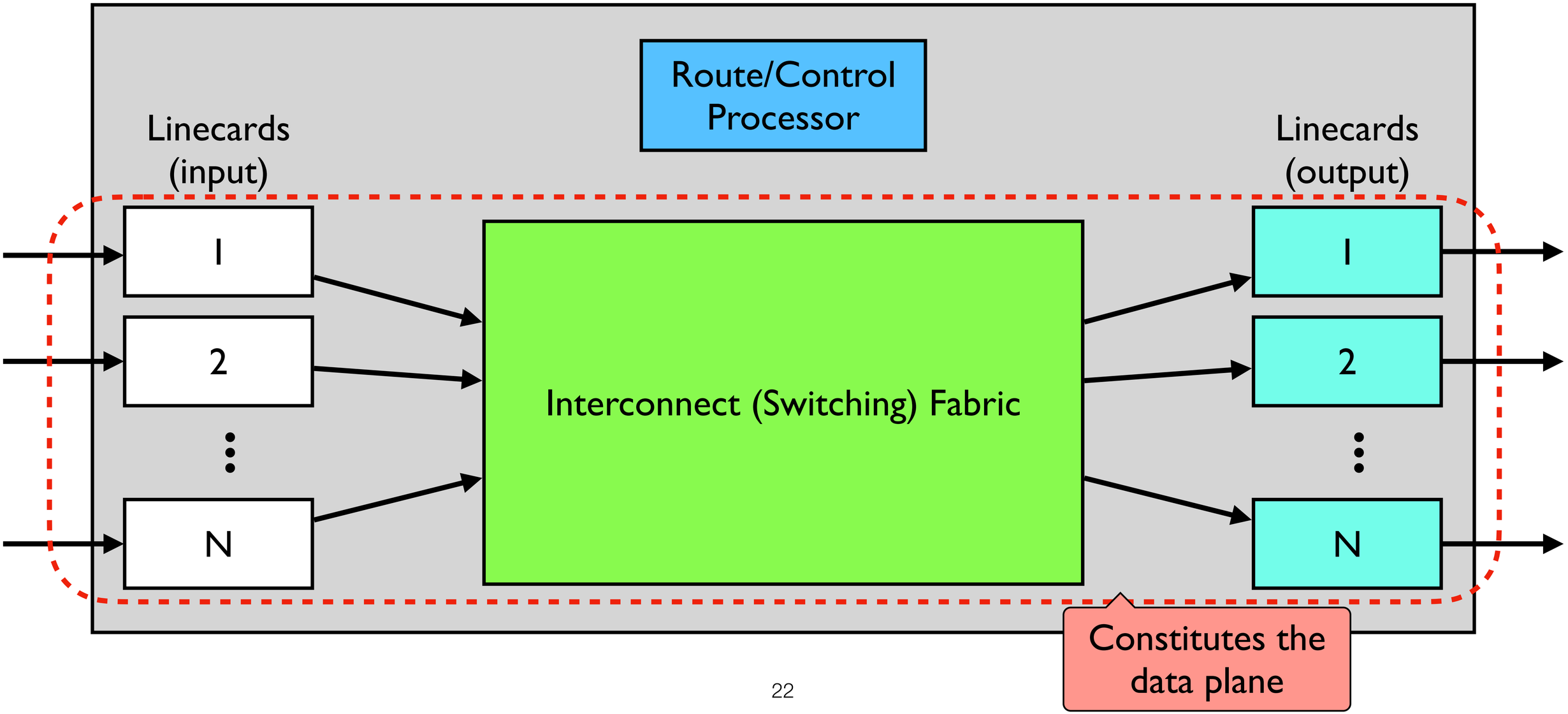
What's inside a router?



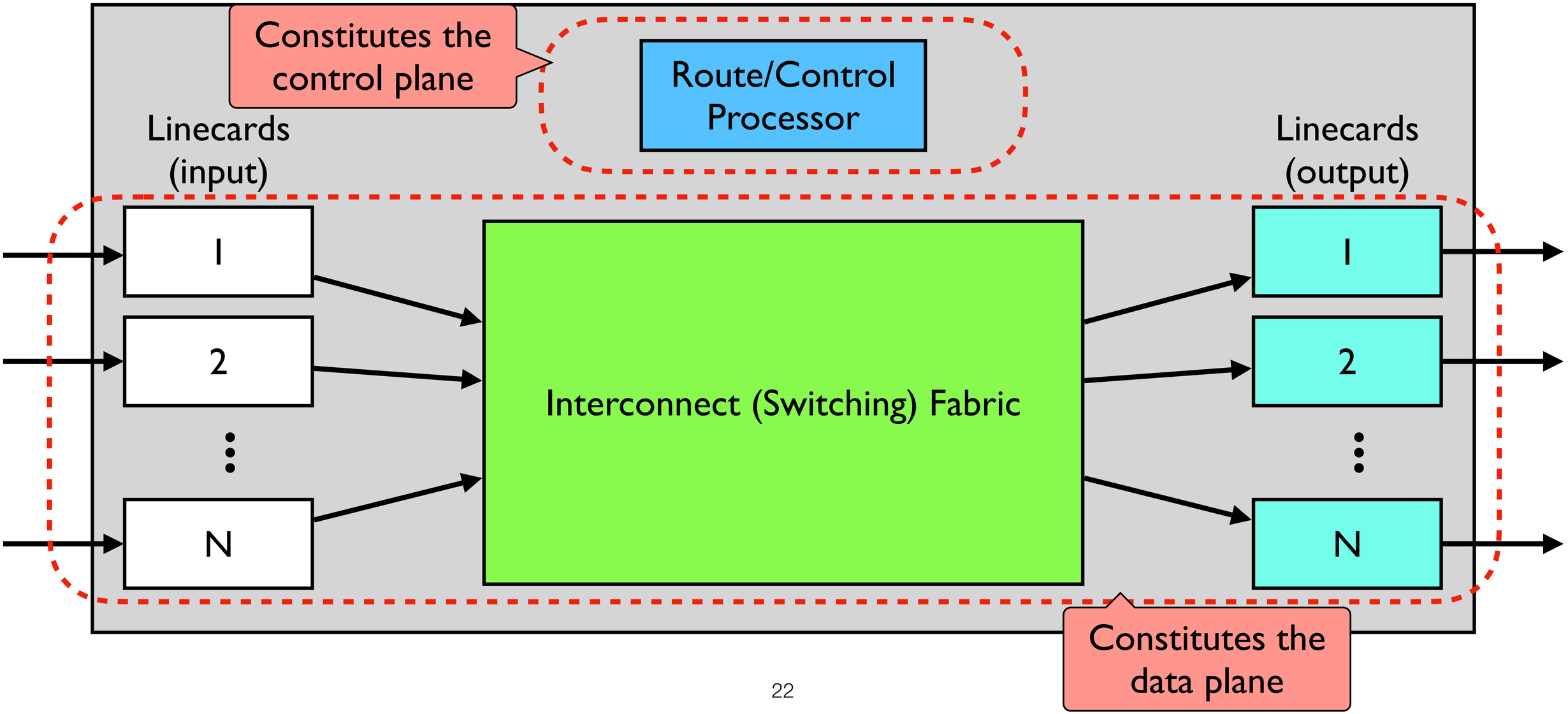
What's inside a router?



What's inside a router?



What's inside a router?



Input Linecards

Input Linecards

- **Tasks**

Input Linecards

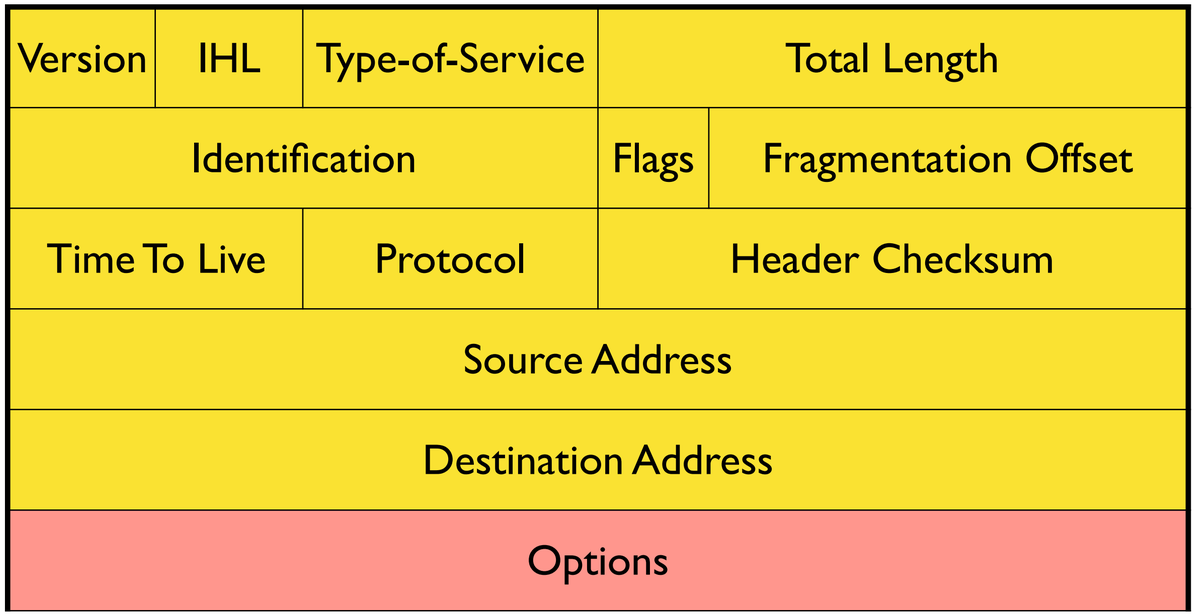
- **Tasks**
  - Receive Incoming packets (physical layer stuff)

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
  - **Q:** What are the header fields that need to be updated?



Input Linecards

- **Tasks**
  - Receive Incoming packets (physical layer stuff)
  - Update the IP header
    - *TTL, Checksum, Options (maybe), Fragmentation fields (maybe)*

Input Linecards

- **Tasks**
  - Receive Incoming packets (physical layer stuff)
  - Update the IP header
    - *TTL, Checksum, Options (maybe), Fragmentation fields (maybe)*
  - Lookup the output port for the destination IP address

Input Linecards

- **Tasks**
  - Receive Incoming packets (physical layer stuff)
  - Update the IP header
    - *TTL, Checksum, Options (maybe), Fragmentation fields (maybe)*
  - Lookup the output port for the destination IP address
  - Queue the packet at the switch fabric

Input Linecards

- **Tasks**
  - Receive Incoming packets (physical layer stuff)
  - Update the IP header
    - *TTL, Checksum, Options (maybe), Fragmentation fields (maybe)*
  - Lookup the output port for the destination IP address
  - Queue the packet at the switch fabric
- **Challenge: need for speed!**

Input Linecards

- **Tasks**
  - Receive Incoming packets (physical layer stuff)
  - Update the IP header
    - *TTL, Checksum, Options (maybe), Fragmentation fields (maybe)*
  - Lookup the output port for the destination IP address
  - Queue the packet at the switch fabric
- **Challenge: need for speed!**
  - 100B packets @ 100Gbps → new packet every 10 nanoseconds!

Input Linecards

- **Tasks**
  - Receive Incoming packets (physical layer stuff)
  - Update the IP header
    - *TTL, Checksum, Options (maybe), Fragmentation fields (maybe)*
  - Lookup the output port for the destination IP address
  - Queue the packet at the switch fabric
- **Challenge: need for speed!**
  - 100B packets @ 100Gbps → new packet every 10 nanoseconds!
- **Typically implemented with specialized hardware**

Input Linecards

- **Tasks**
  - Receive Incoming packets (physical layer stuff)
  - Update the IP header
    - *TTL, Checksum, Options (maybe), Fragmentation fields (maybe)*
  - Lookup the output port for the destination IP address
  - Queue the packet at the switch fabric
- **Challenge: need for speed!**
  - 100B packets @ 100Gbps → new packet every 10 nanoseconds!
- **Typically implemented with specialized hardware**
  - ASICs, specialized "network processors"

Input Linecards

- **Tasks**
  - Receive Incoming packets (physical layer stuff)
  - Update the IP header
    - *TTL, Checksum, Options (maybe), Fragmentation fields (maybe)*
  - Lookup the output port for the destination IP address
  - Queue the packet at the switch fabric
- **Challenge: need for speed!**
  - 100B packets @ 100Gbps → new packet every 10 nanoseconds!
- **Typically implemented with specialized hardware**
  - ASICs, specialized "network processors"
  - "Exception processing" often done at control processor

Looking up the output port

Looking up the output port

- **One entry for each address → 4 billion entries!**

Looking up the output port

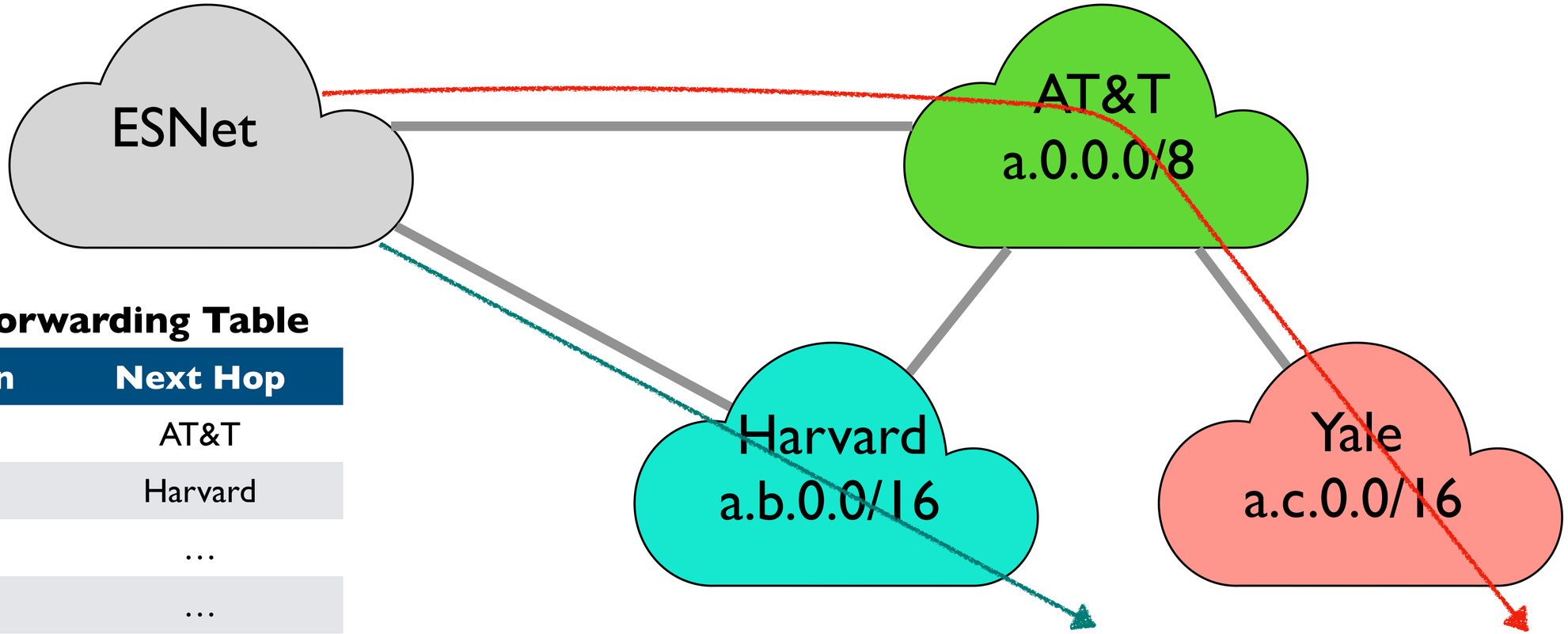
- **One entry for each address → 4 billion entries!**
- **For scalability, address are aggregated**

Looking up the output port

- **One entry for each address → 4 billion entries!**
- **For scalability, address are aggregated**
  - We've already seen this!

But aggregation is imperfect…

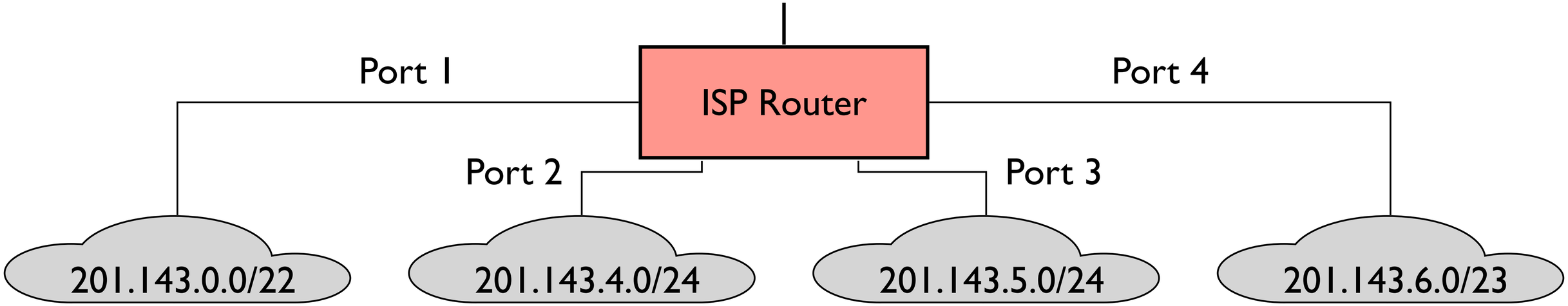
ESNet must maintain routing entries for a.*.*.* and a.b.*.*



ESNet's Forwarding Table

Destination	Next Hop
a.*.*.*	AT&T
a.b.*.*	Harvard
…	…
…	…

Example #1: 4 prefixes, 4 ports



Prefix	Port
201.143.0.0/22	Port 1
201.143.4.0/24	Port 2
201.143.5.0/24	Port 3
201.143.6.0/23	Port 4

Finding a match

Prefix	Port
201.143.0.0/22	Port 1
201.143.4.0/24	Port 2
201.143.5.0/24	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
201.143.4.0/24	Port 2
201.143.5.0/24	Port 3
201.143.6.0/23	Port 4

Finding a match

- **Incoming packet destination: 201.143.7.0**

201.143.0.0/22

201.143.4.0/24

201.143.7.0/25

201.143.6.0/23

Routing Table

Finding a match

- **Incoming packet destination: 201.143.7.0**

11001001	10001111	00000111	10100010

201.143.0.0/22

201.143.4.0/24

201.143.7.0/25

201.143.6.0/23

Routing Table

Finding a match

- **Incoming packet destination: 201.143.7.0**

Routing Table

201.143.0.0/22

201.143.4.0/24

201.143.7.0/25

201.143.6.0/23

Finding a match

- **Incoming packet destination: 201.143.7.0**

Routing Table

201.143.0.0/22

201.143.4.0/24

201.143.7.0/25

201.143.6.0/23

Finding a match

- **Incoming packet destination: 201.143.7.0**

201.143.0.0/22

201.143.4.0/24

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

201.143.7.0/25

201.143.6.0/23

Routing Table

Finding Match Efficiently

Finding Match Efficiently

- **Testing each entry to find a match scales poorly**
  - On average: O(number of entries)

Finding Match Efficiently

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

Finding Match Efficiently

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

Finding Match Efficiently

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

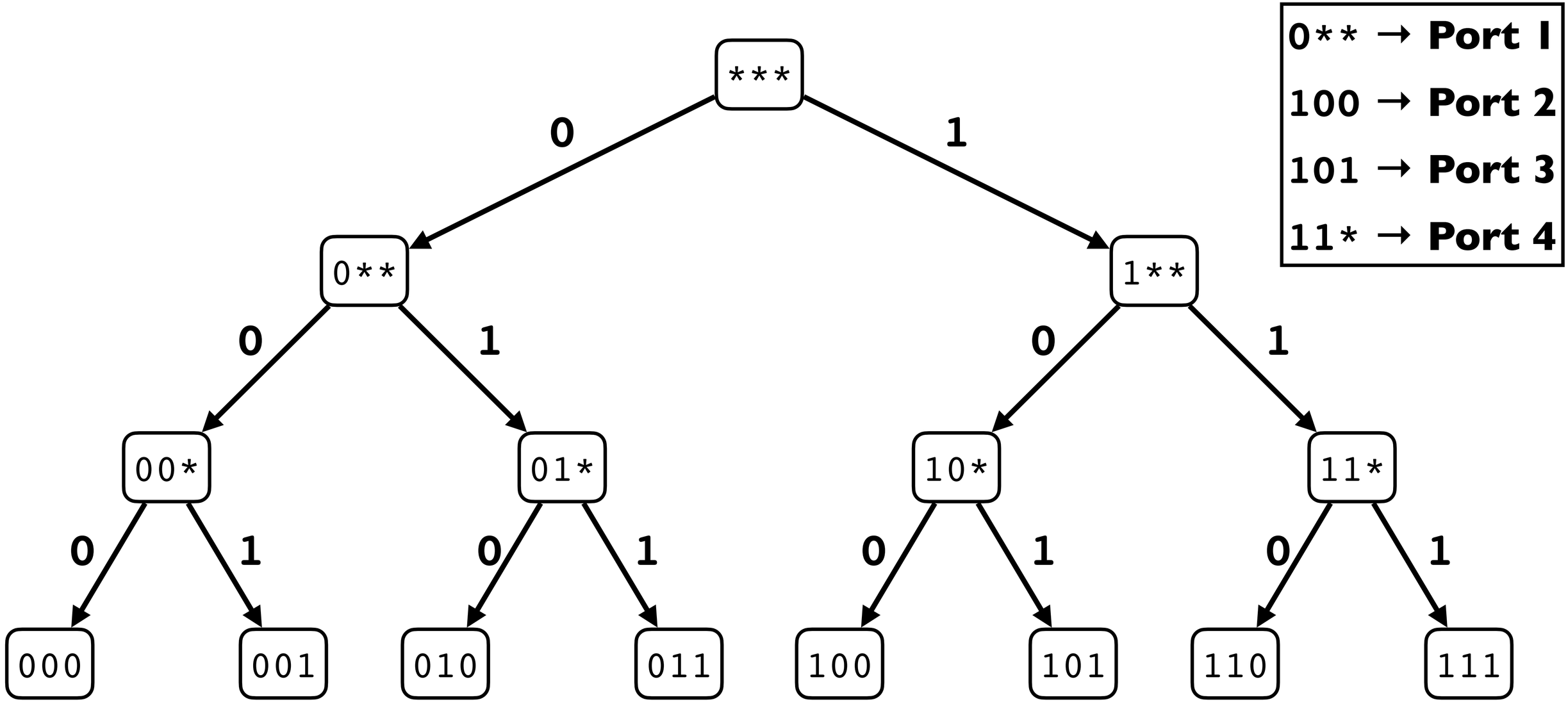
Prefix	Port
11001001100011110000**0**********	**Port 1**
11001001100011110000**100********	**Port 2**
11001001100011110000**101********	**Port 3**
11001001100011110000**11*********	**Port 4**

Consider four three-bit prefixes

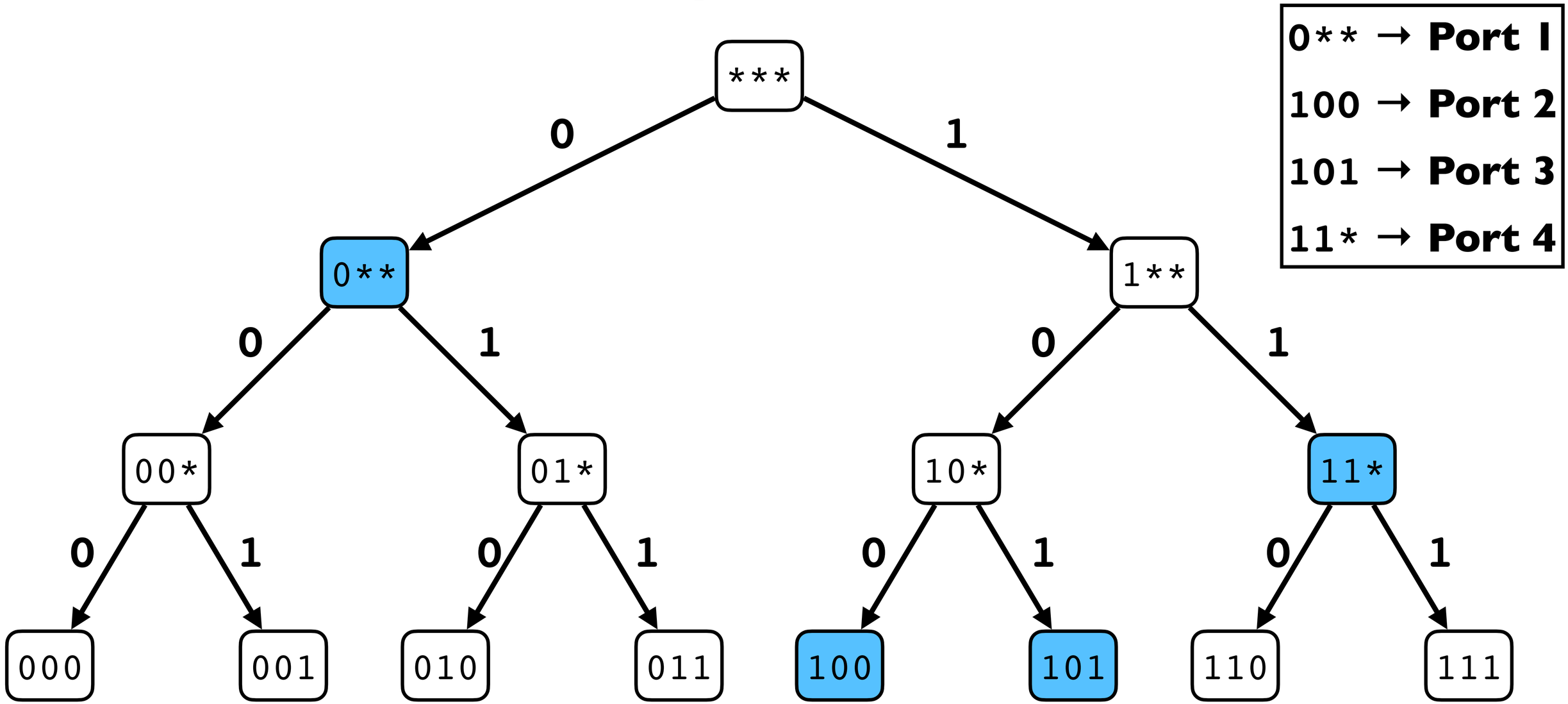
- **Just focusing on the bits where all the action is…**

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

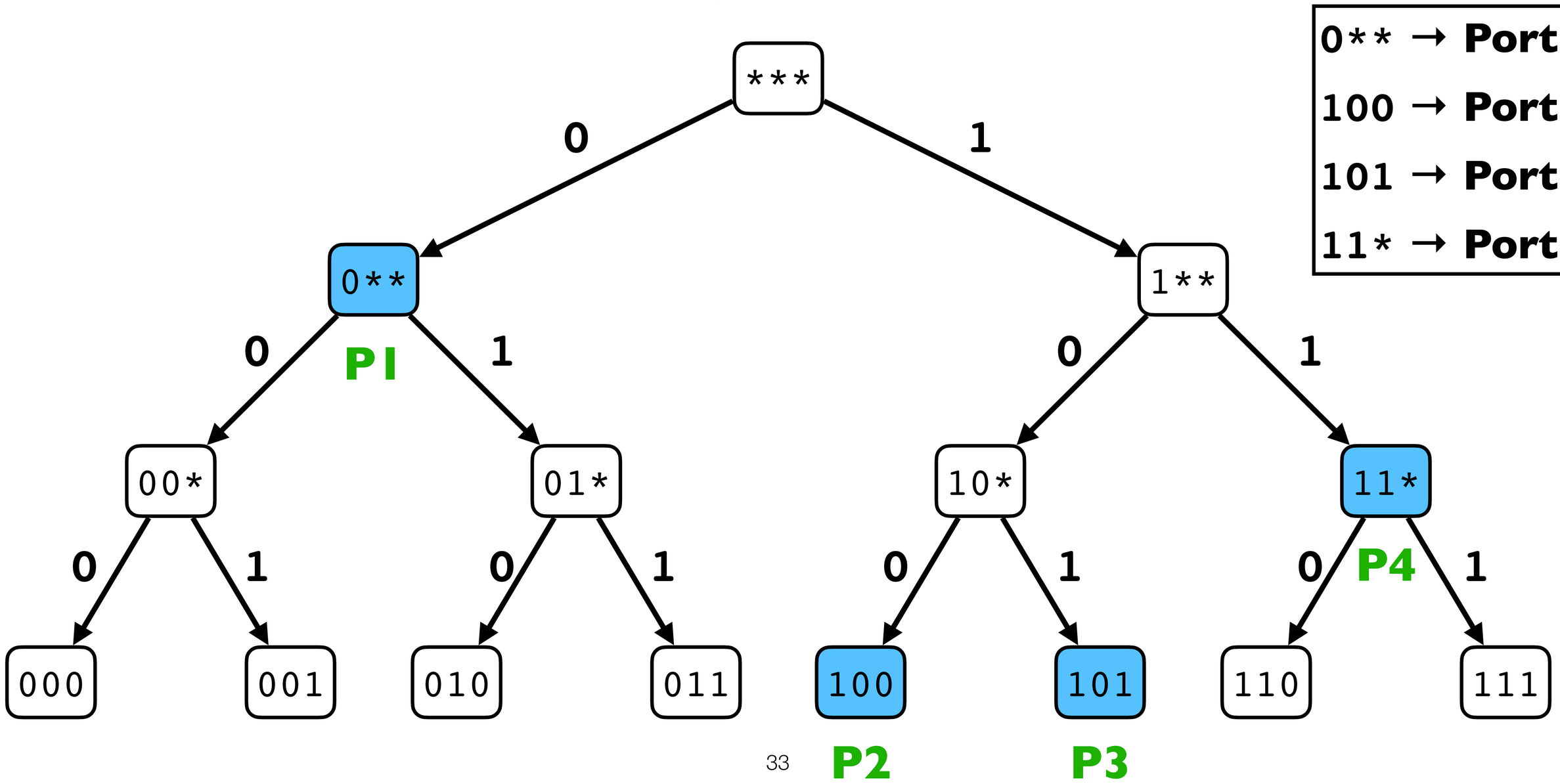
Tree Structure



Walk Tree: Stop at Prefix Entries



Walk Tree: Stop at Prefix Entries

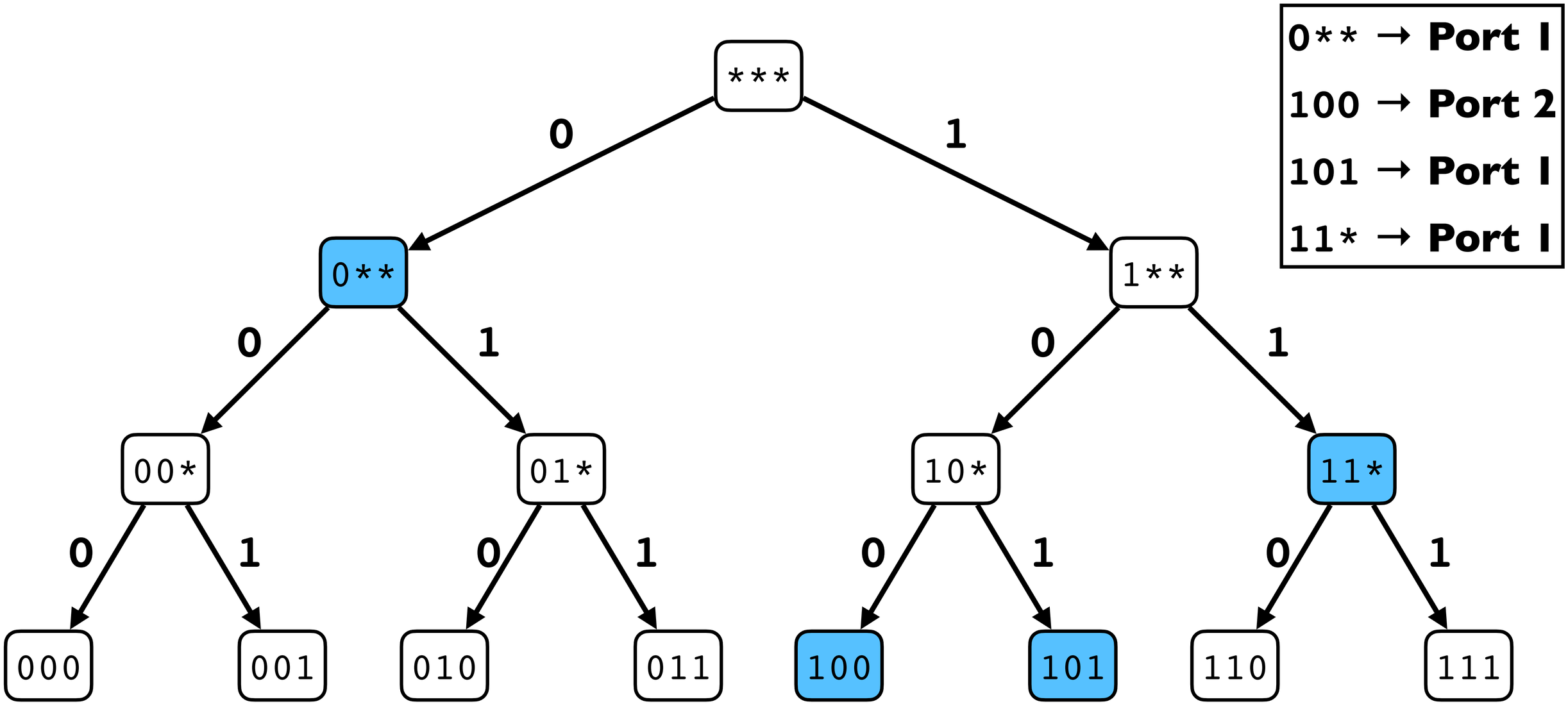


Slightly Different Example

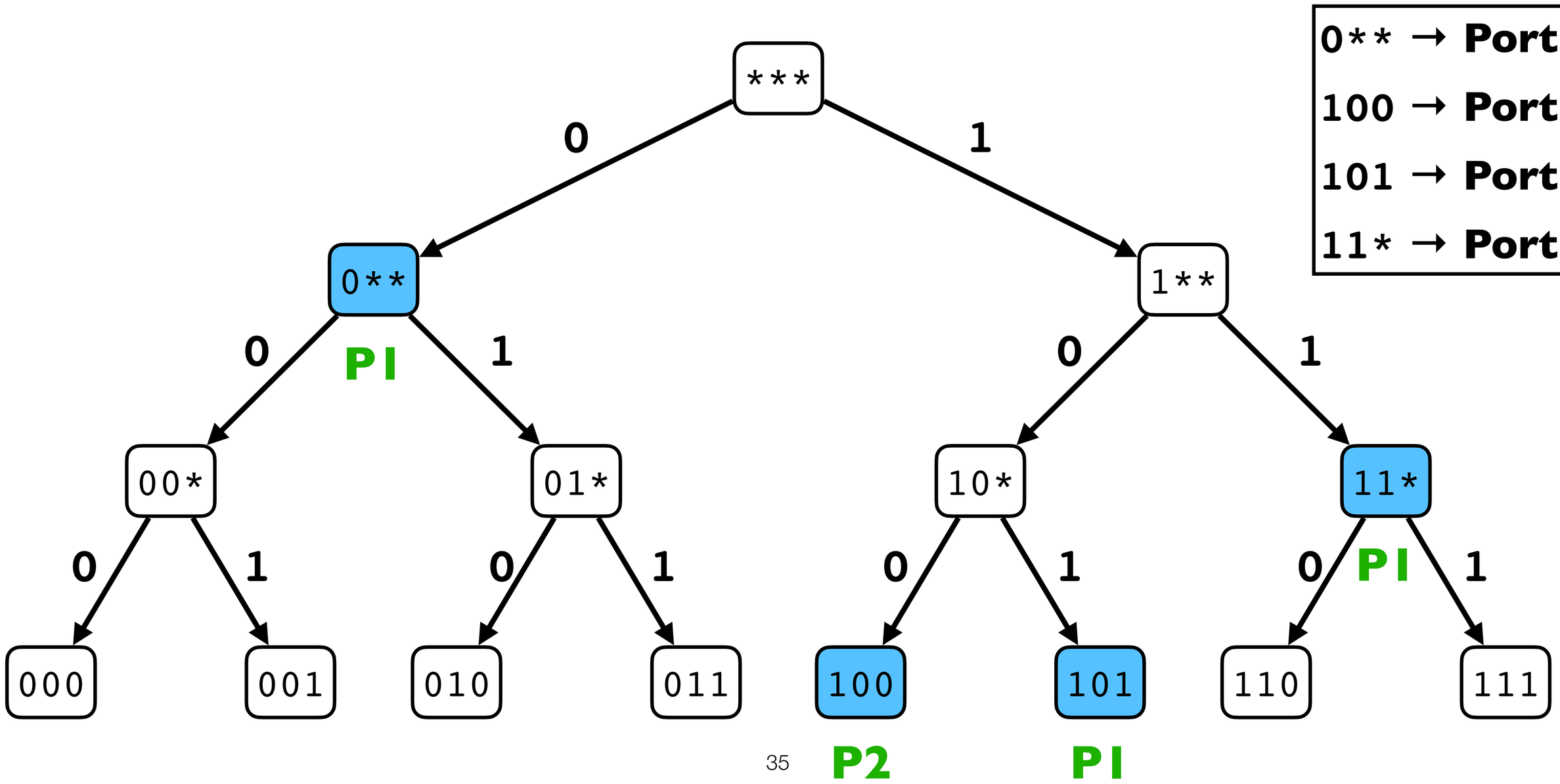
- **Several of the unique prefixes go to the same port**

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

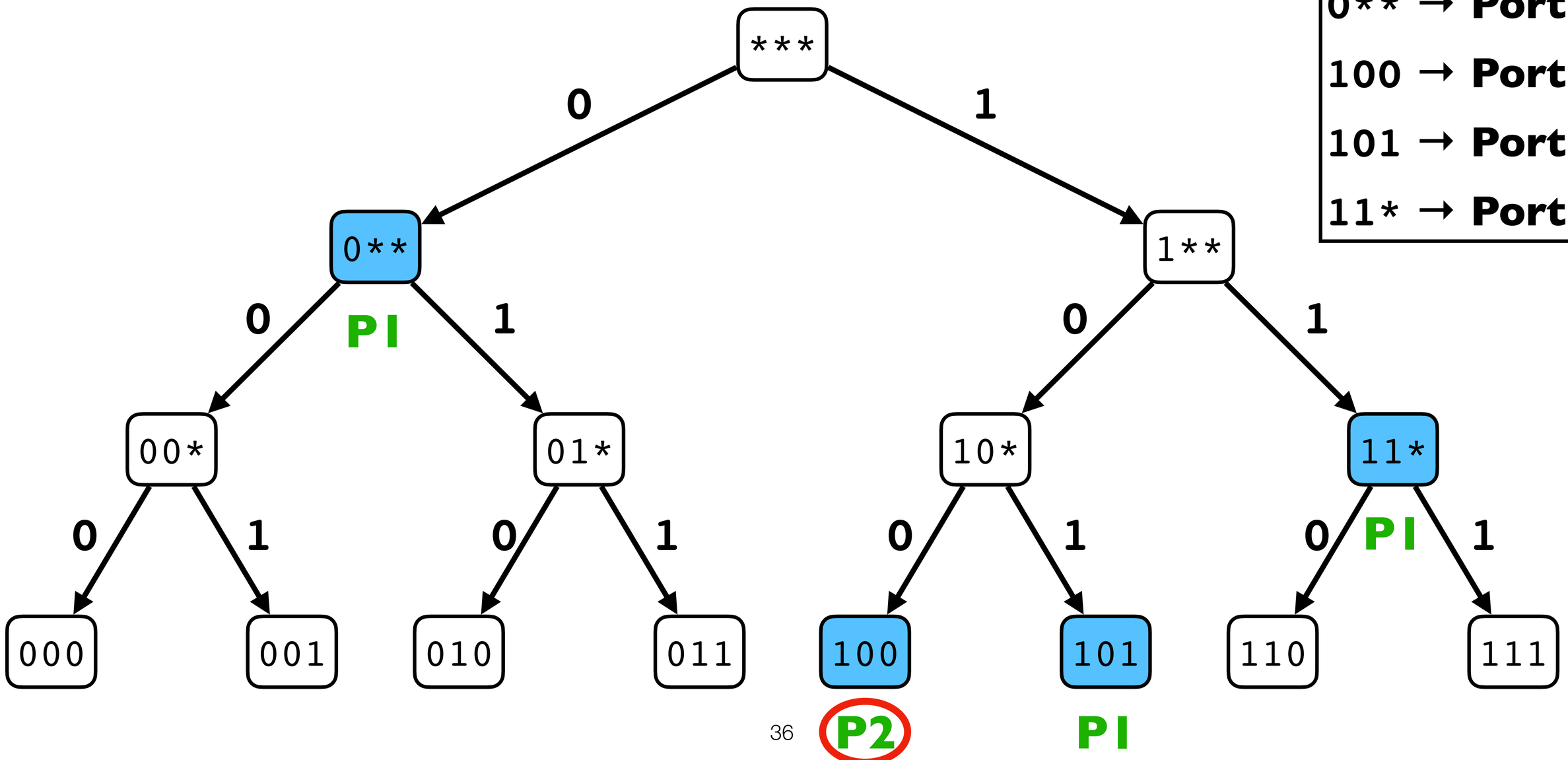
Prefix Tree


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

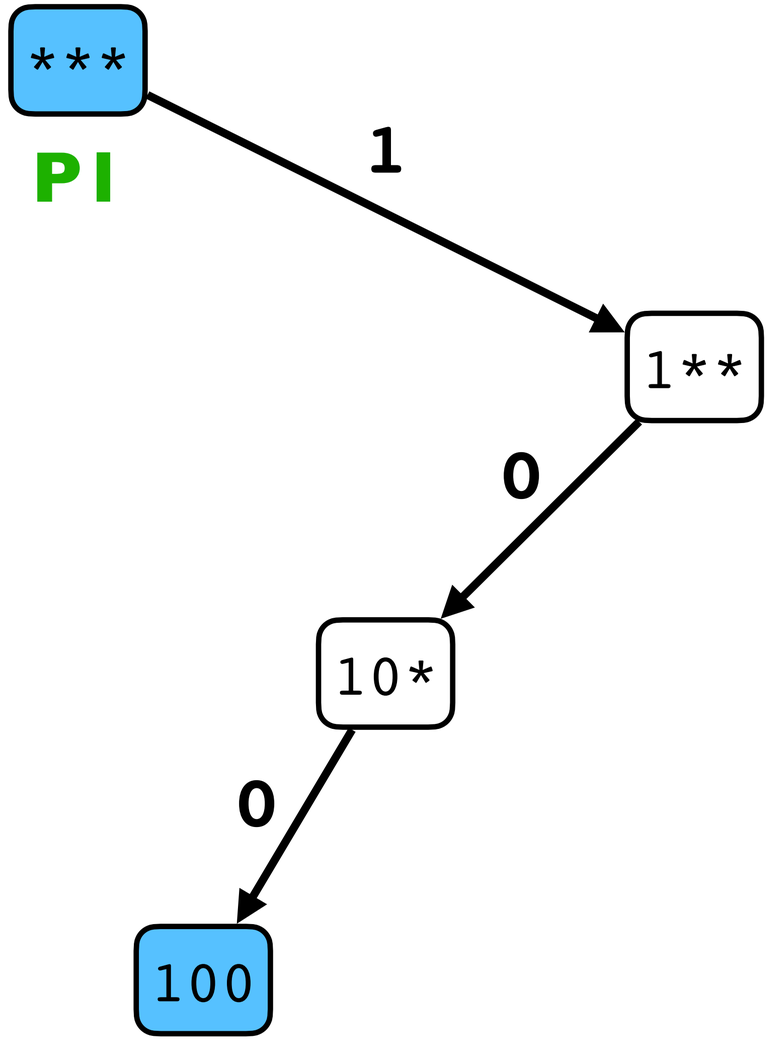
Prefix Tree


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

More Compact Representation

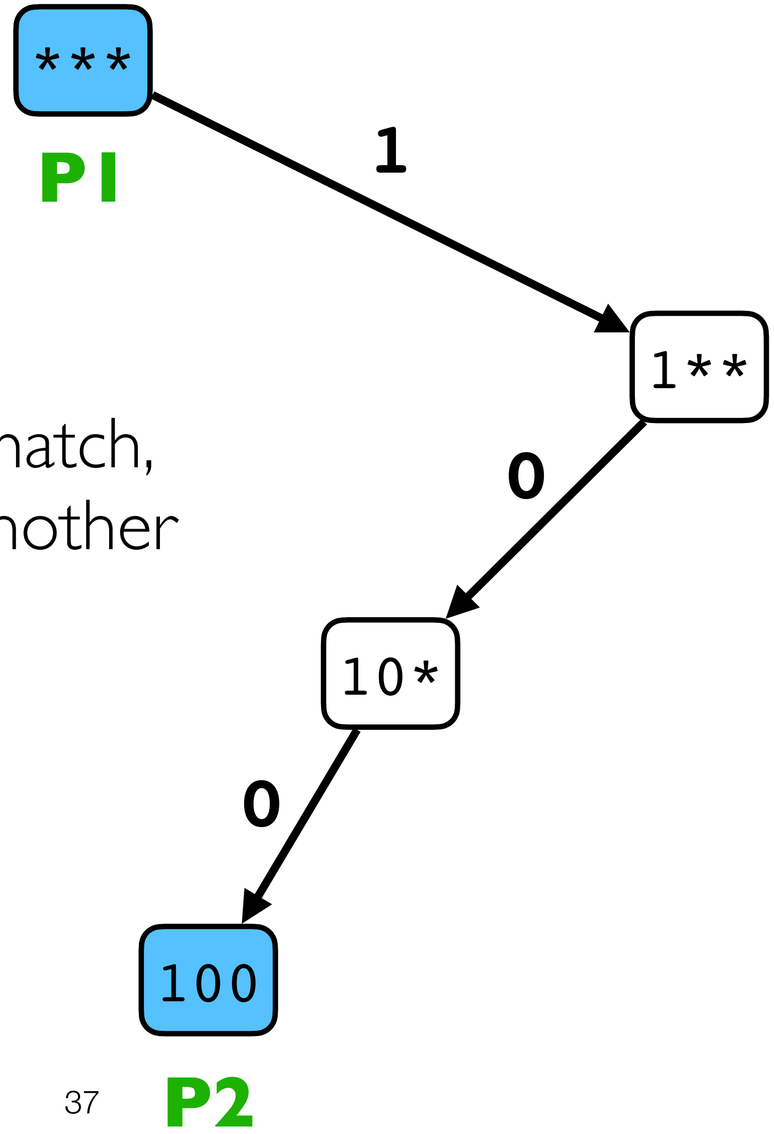


More Compact Representation



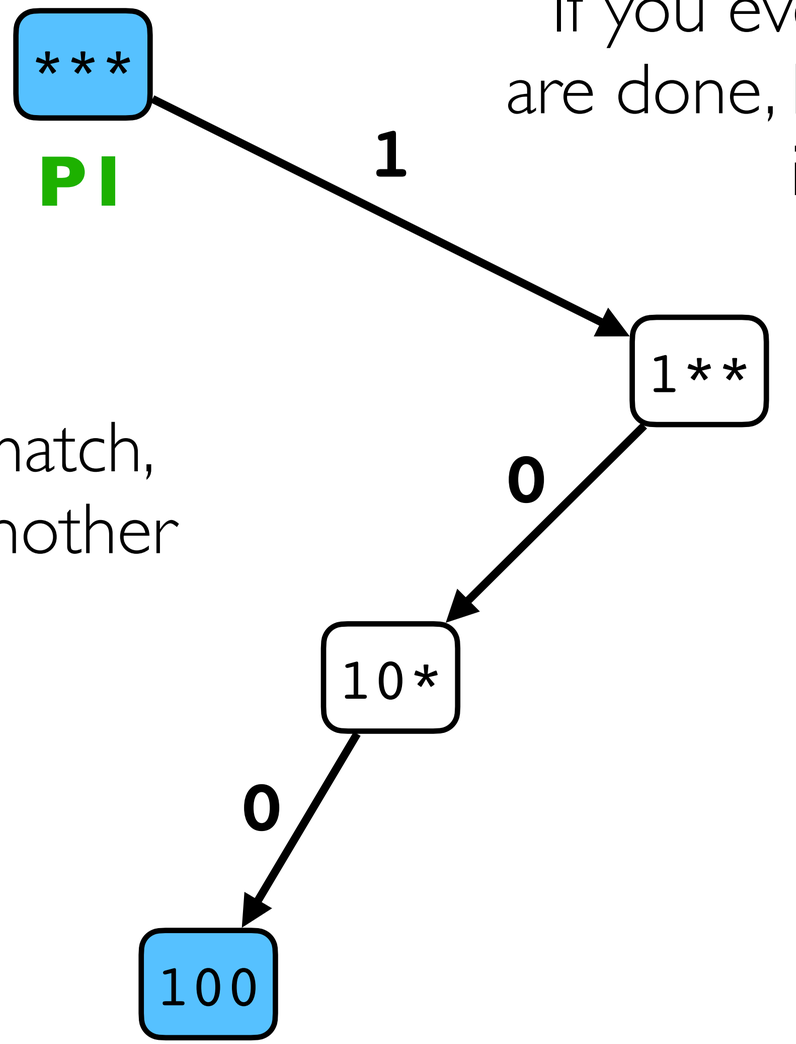
More Compact Representation

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



More Compact Representation

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



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

LPM in Real Routers

LPM in Real Routers

- **Real routers use far more advanced/complex solutions than the approaches just described**
  - But what we discussed is their starting point

LPM in Real Routers

- **Real routers use far more advanced/complex solutions than the approaches 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

Recap: Input Linecards

- **Main challenge is processing speeds**

Recap: Input Linecards

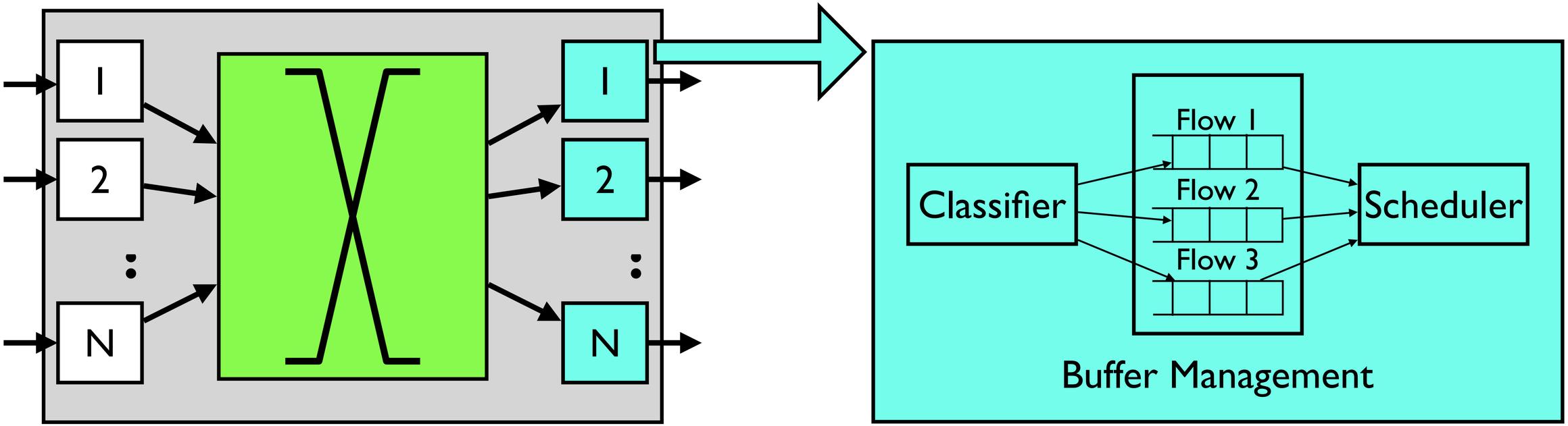
- **Main challenge is processing speeds**
- **Tasks involved:**
  - Update packet header (easy)
  - LPM lookup on destination address (harder)

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



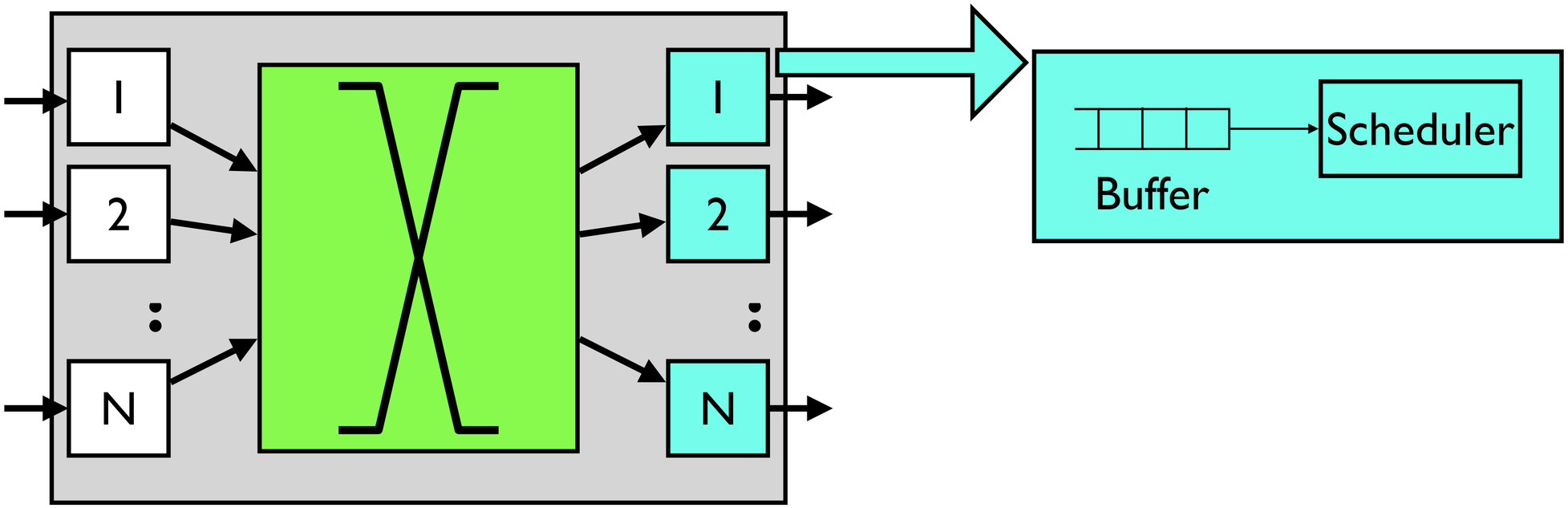
Output Linecard

- **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)

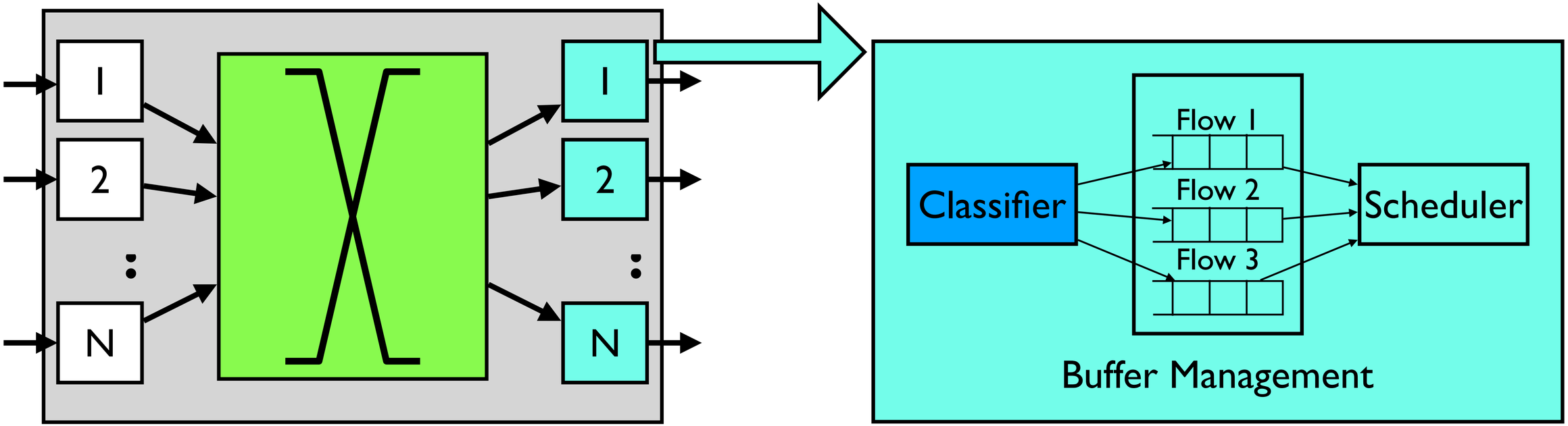
Simplest: FIFO Router

- **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



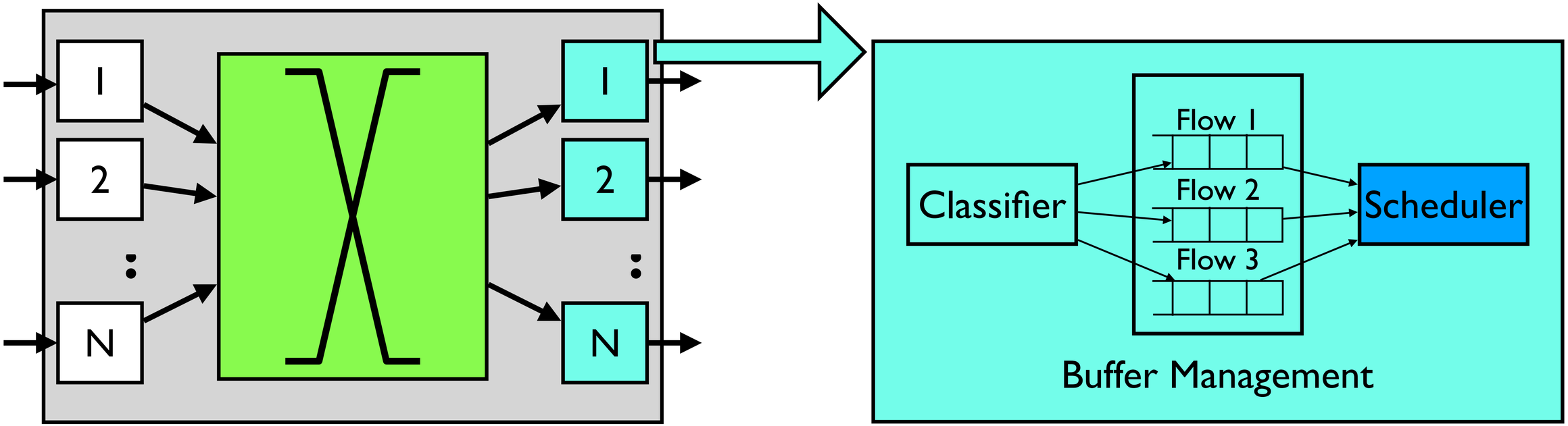
Packet Classification

- **Classify an IP packet based on fields in the packet header, e.g,**
  - Src/dst IP address, src/dst TCP port number, Type-of-Service (ToS), Protocol
- **In general fields are specified by range**
  - Classification requires a multi-dimensional range search!



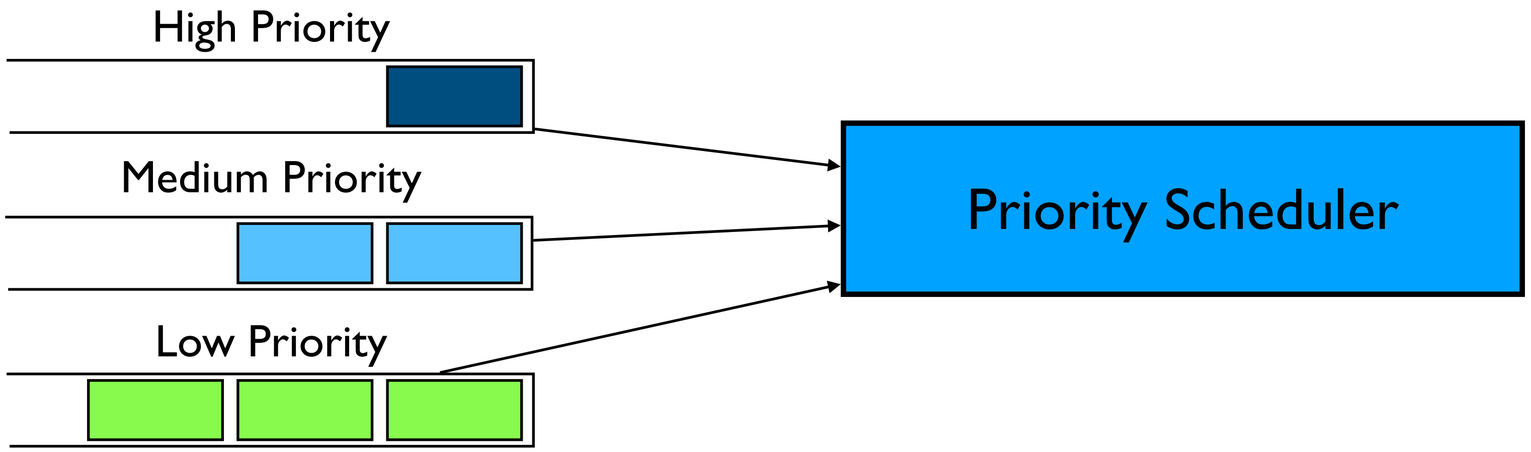
Scheduler

- **One queue per "flow"**
- **Scheduler decides when and from which queue to send a packet**
- **Goals of a scheduling algorithm**
  - (1) Fast! (2) Depends on the policy being implemented (fairness, priority, etc.)



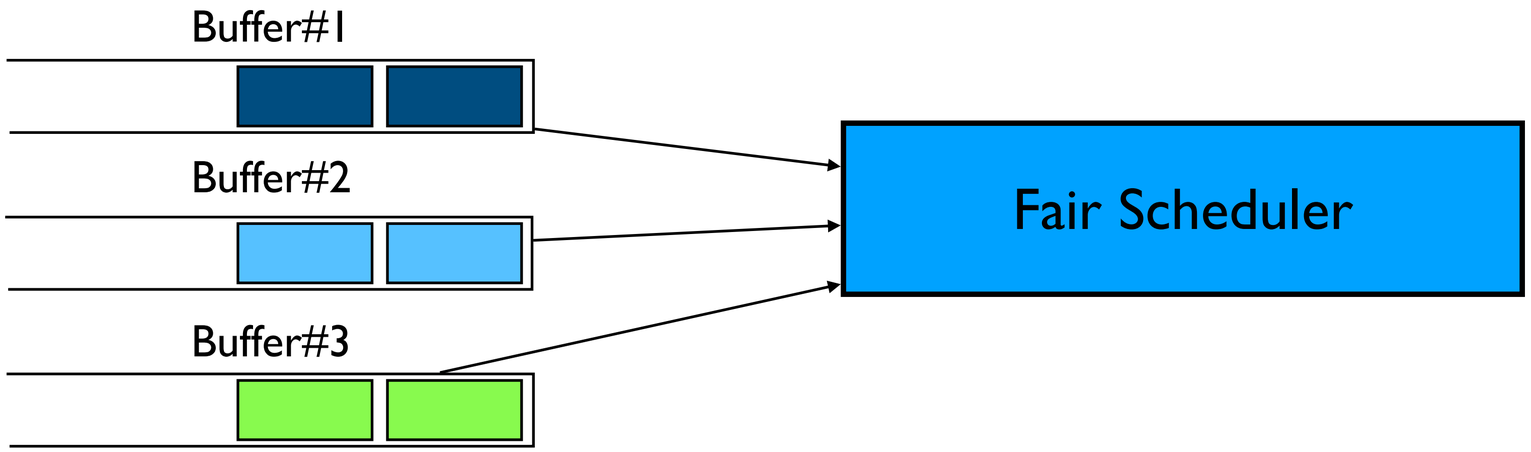
Example: Priority Scheduler

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



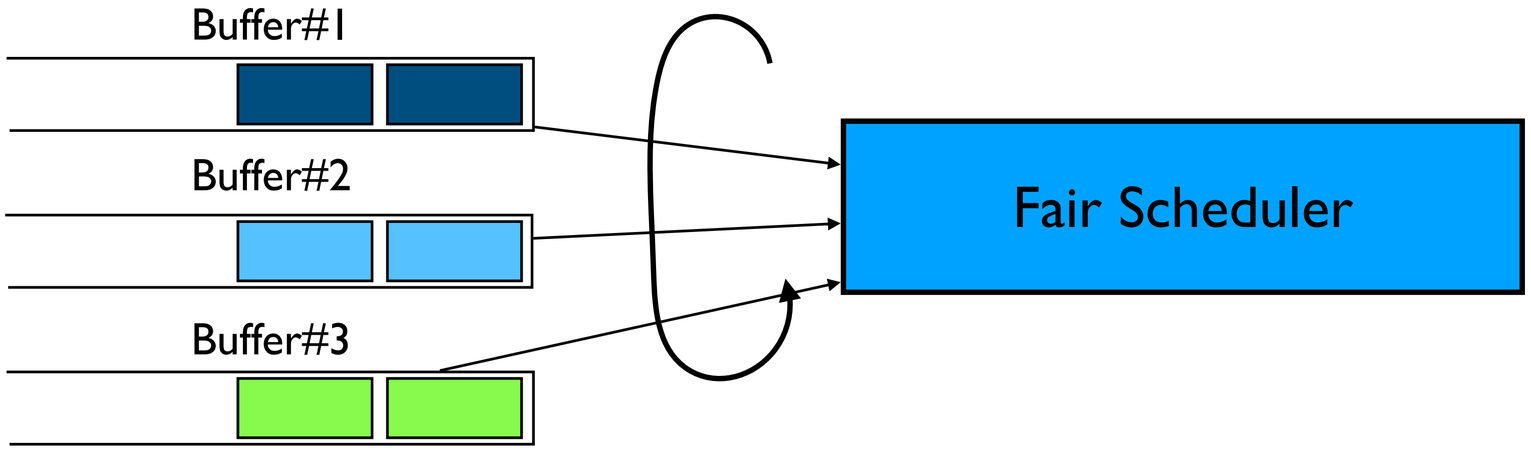
Example: Round-Robin Scheduler

- **Priority Scheduler:** packets are served from each queue in turn

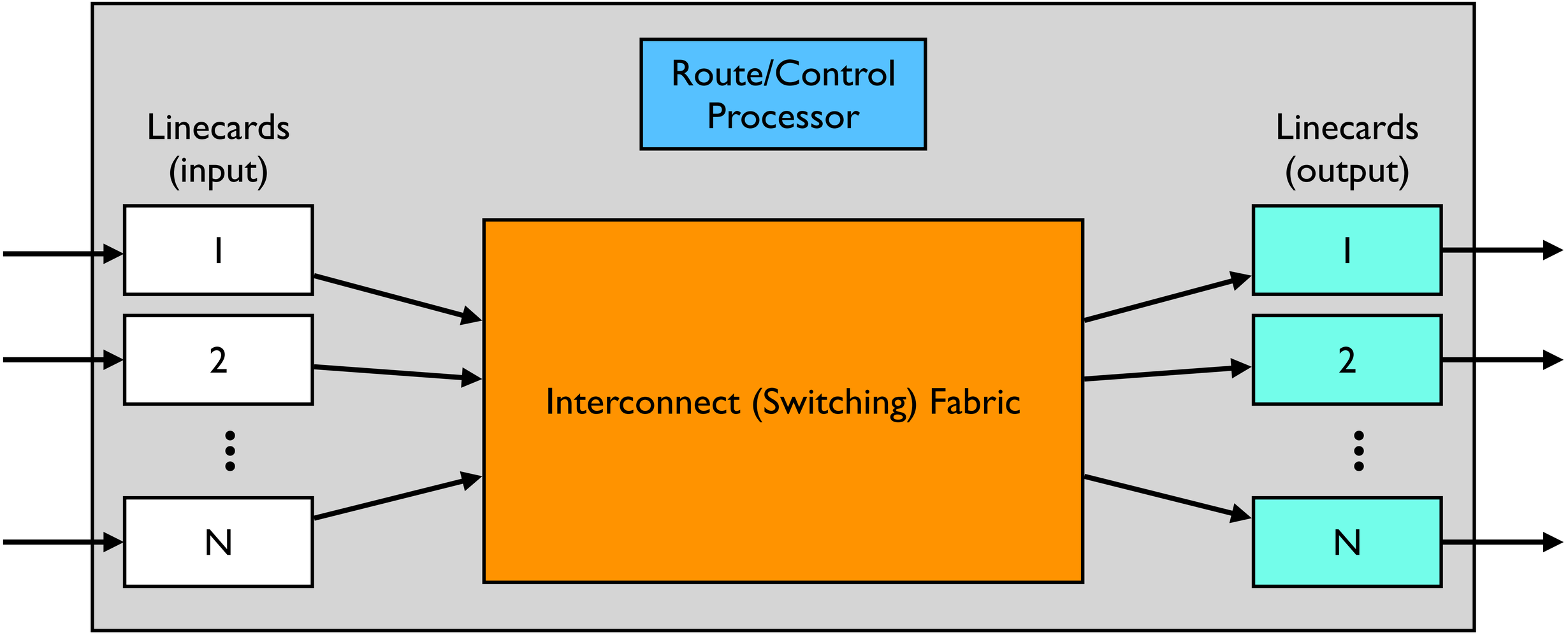


Example: Round-Robin Scheduler

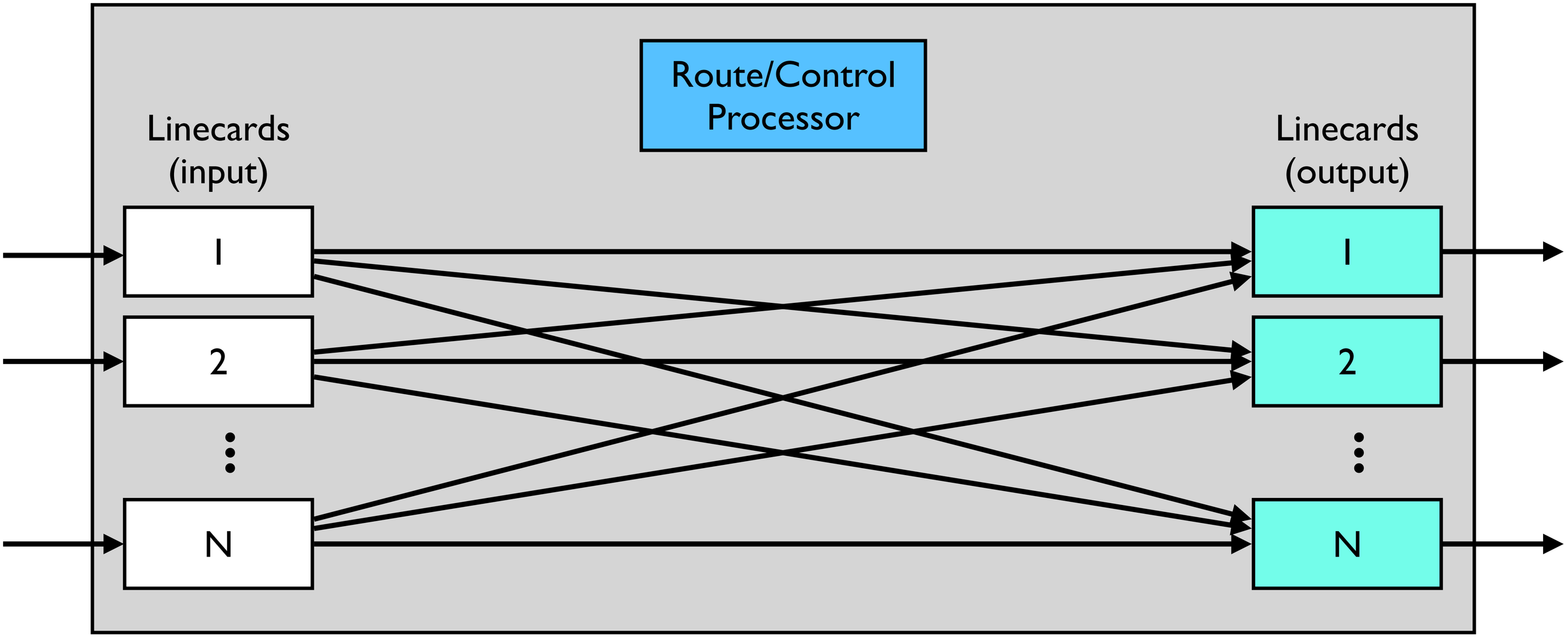
- **Priority Scheduler:** packets are served from each queue in turn



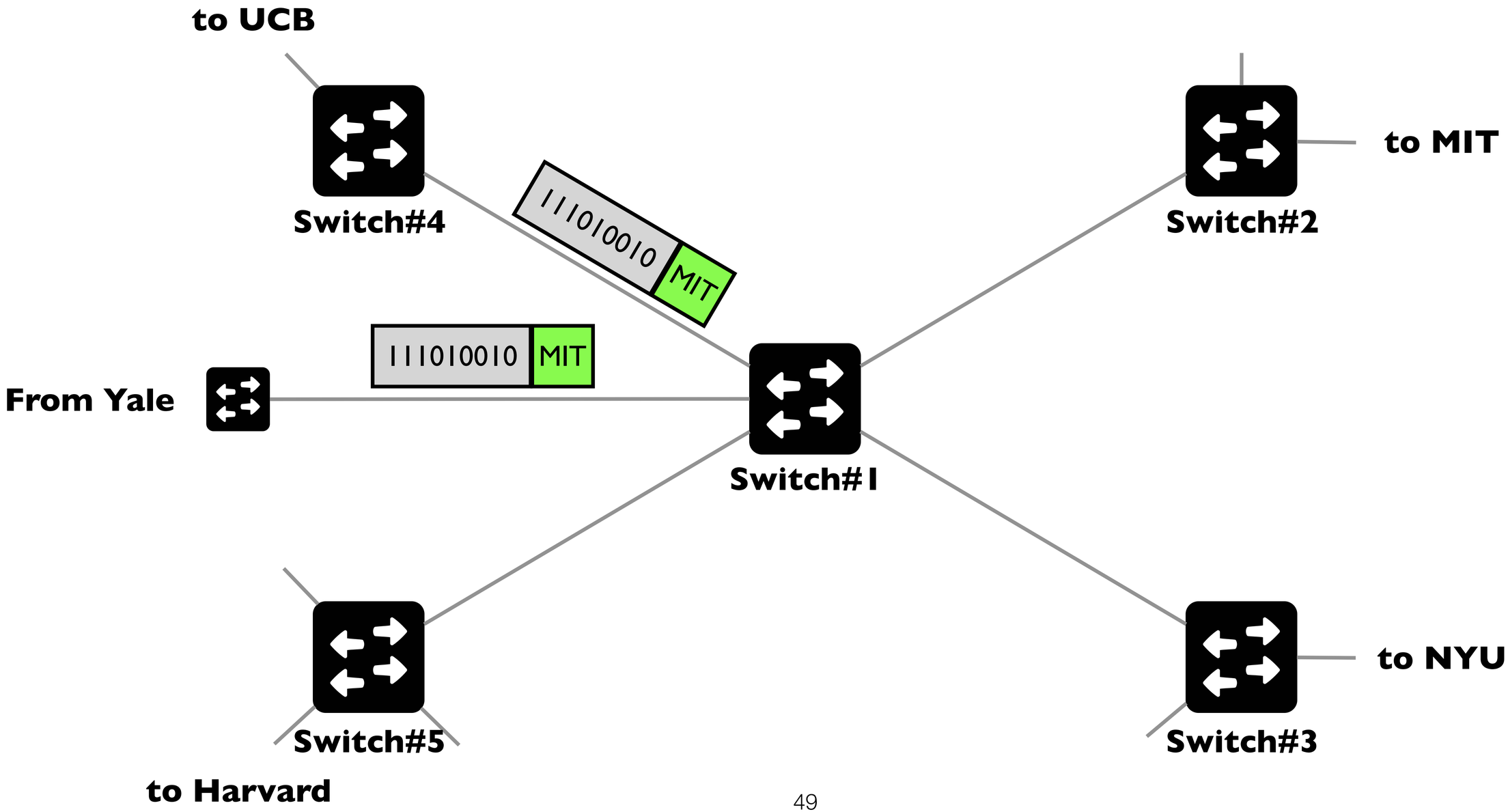
Connecting Input to Output: Switch Fabric



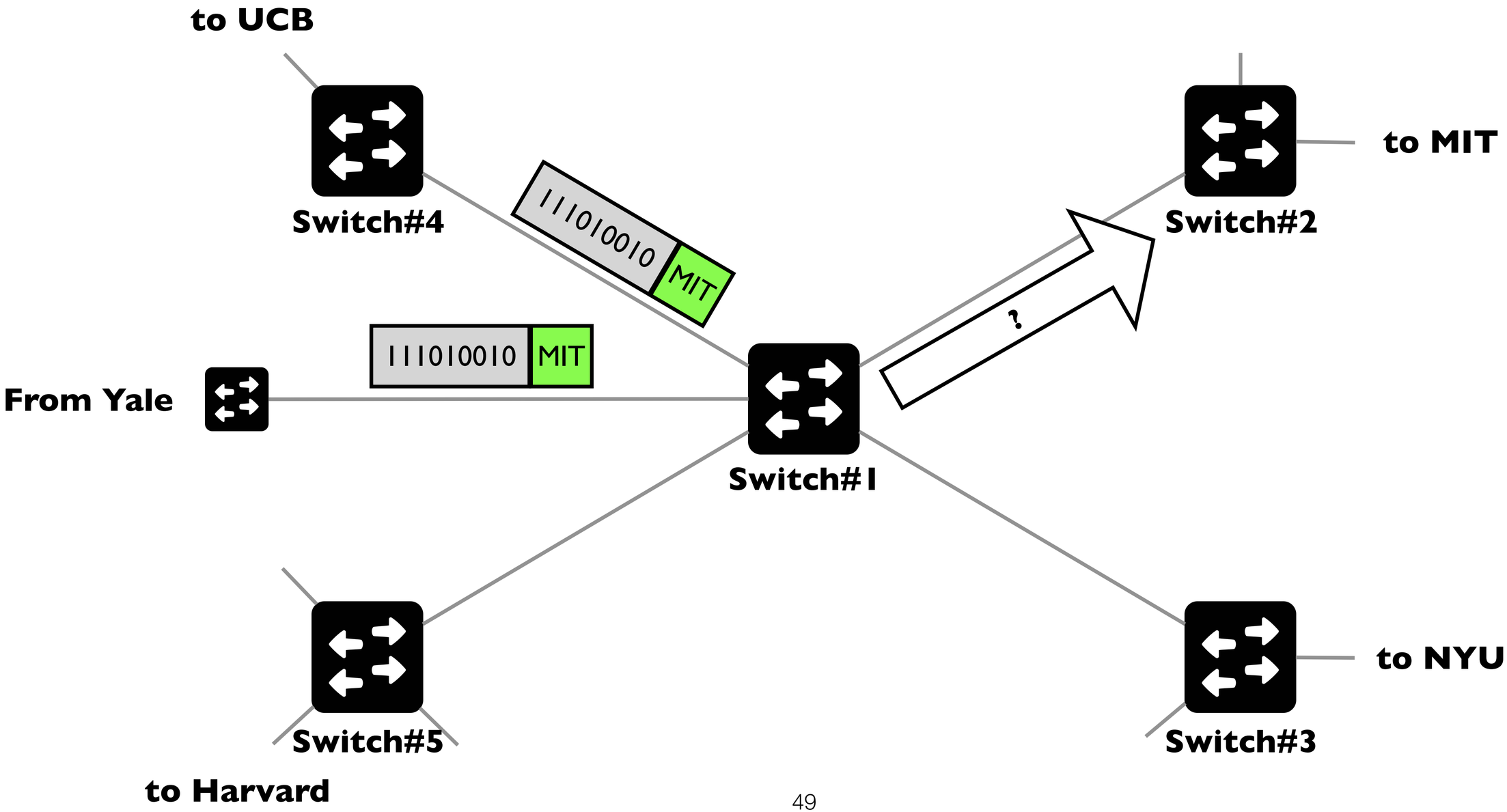
Today's Switch Fabrics: Mini-Network!



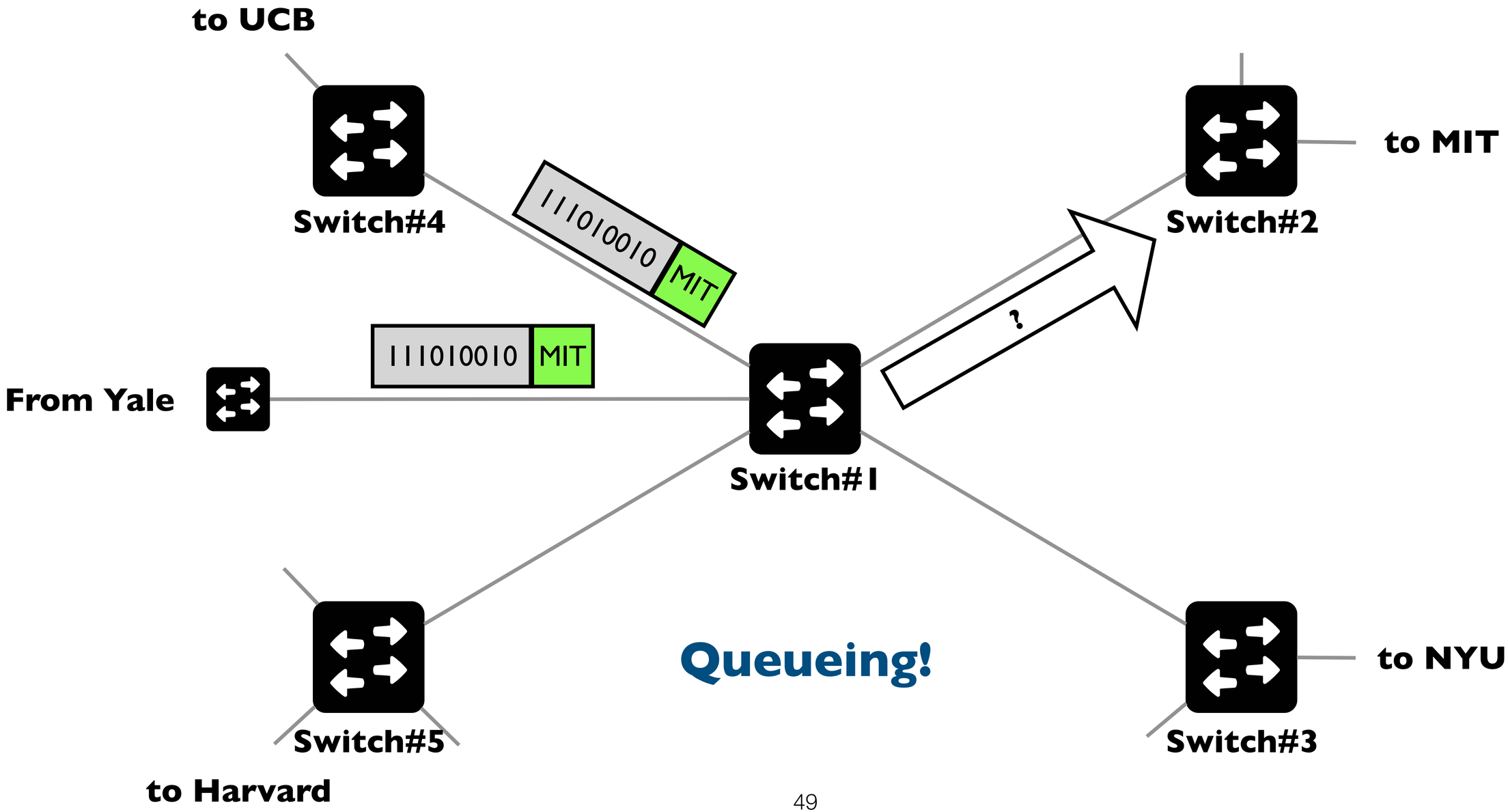
What's hard about the switch fabric?



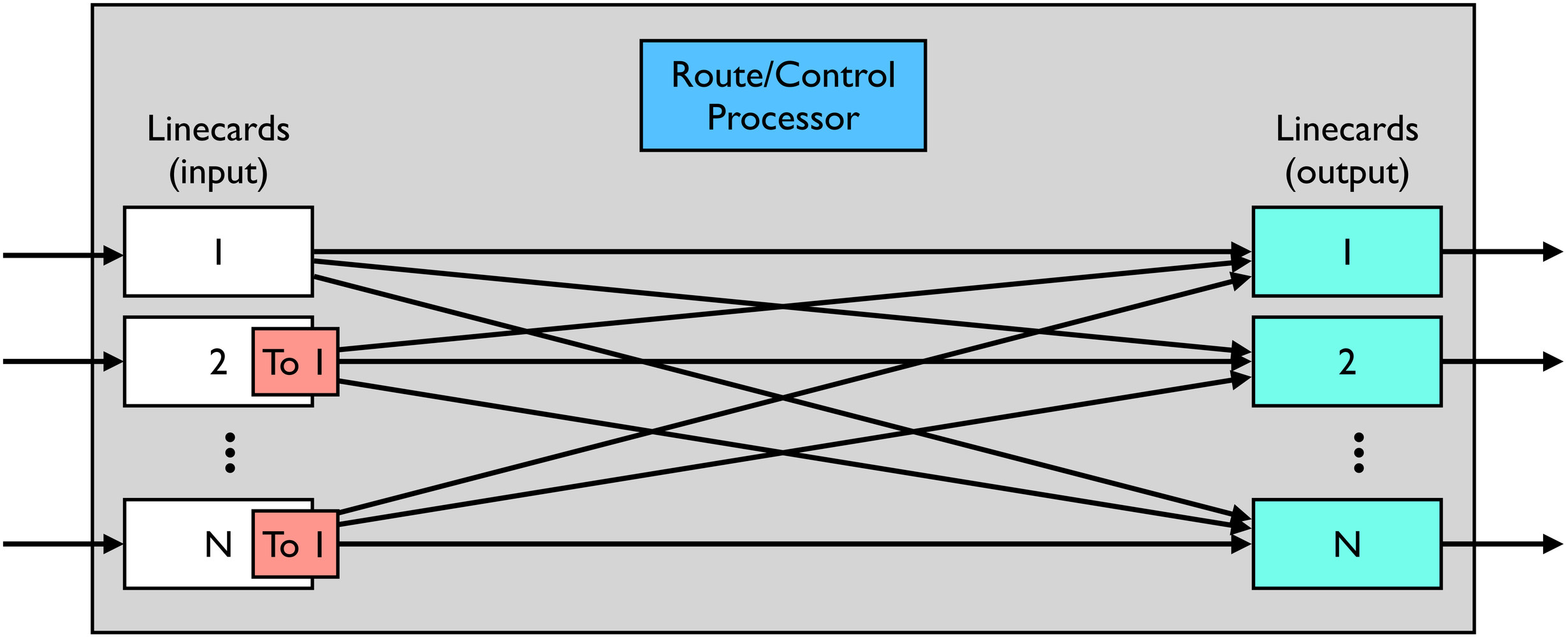
What's hard about the switch fabric?



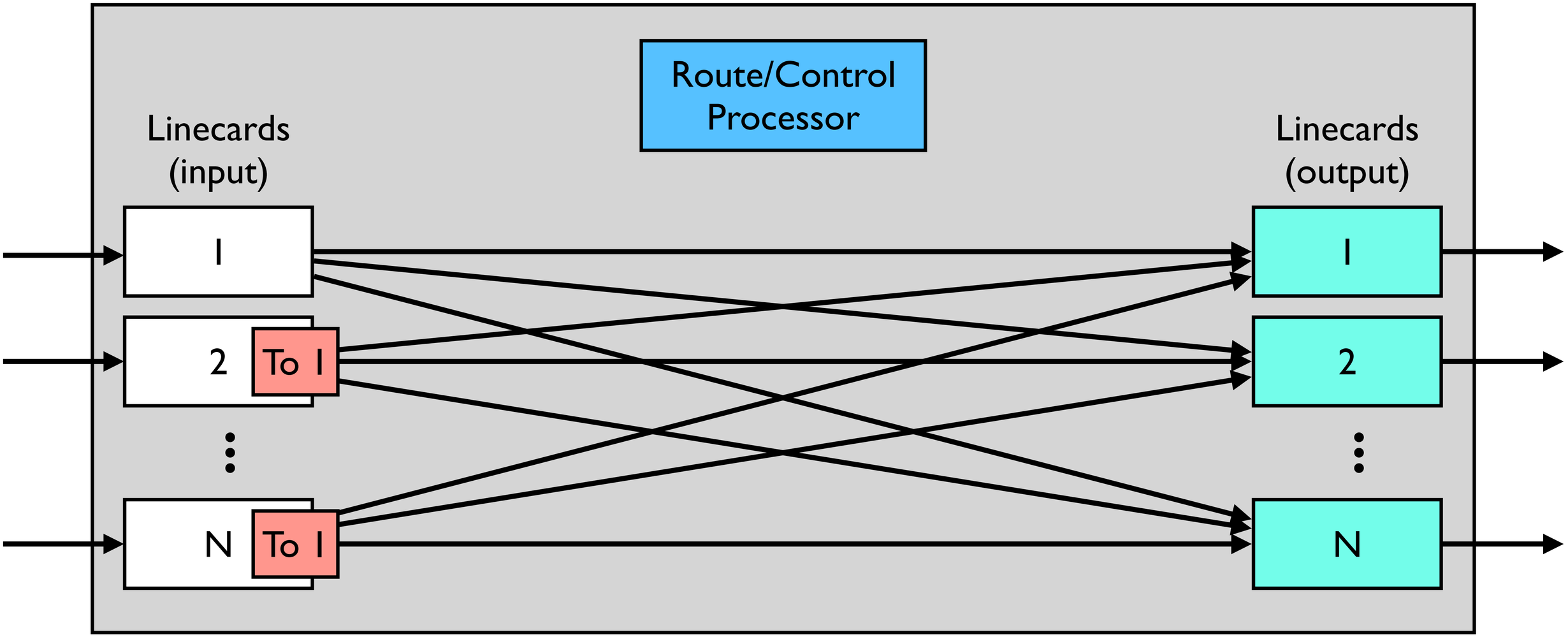
What's hard about the switch fabric?



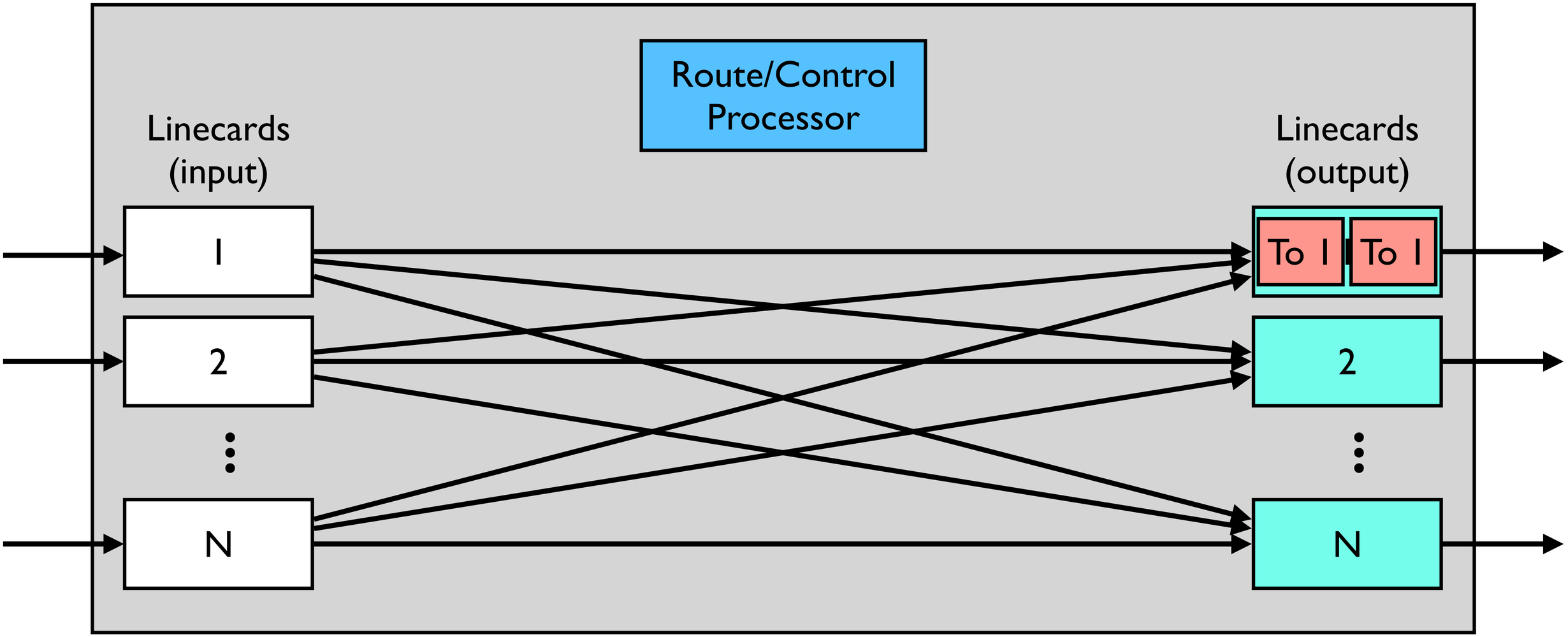
Queueing


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

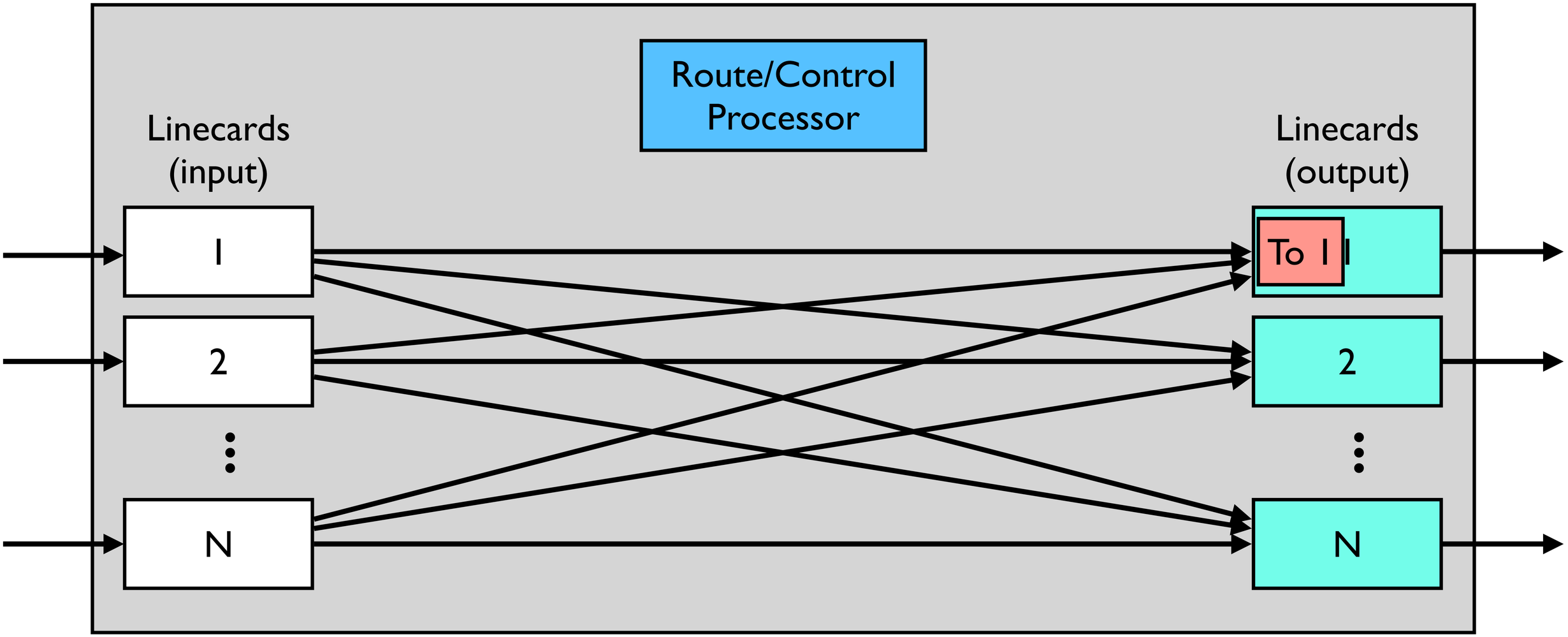
Output Queueing



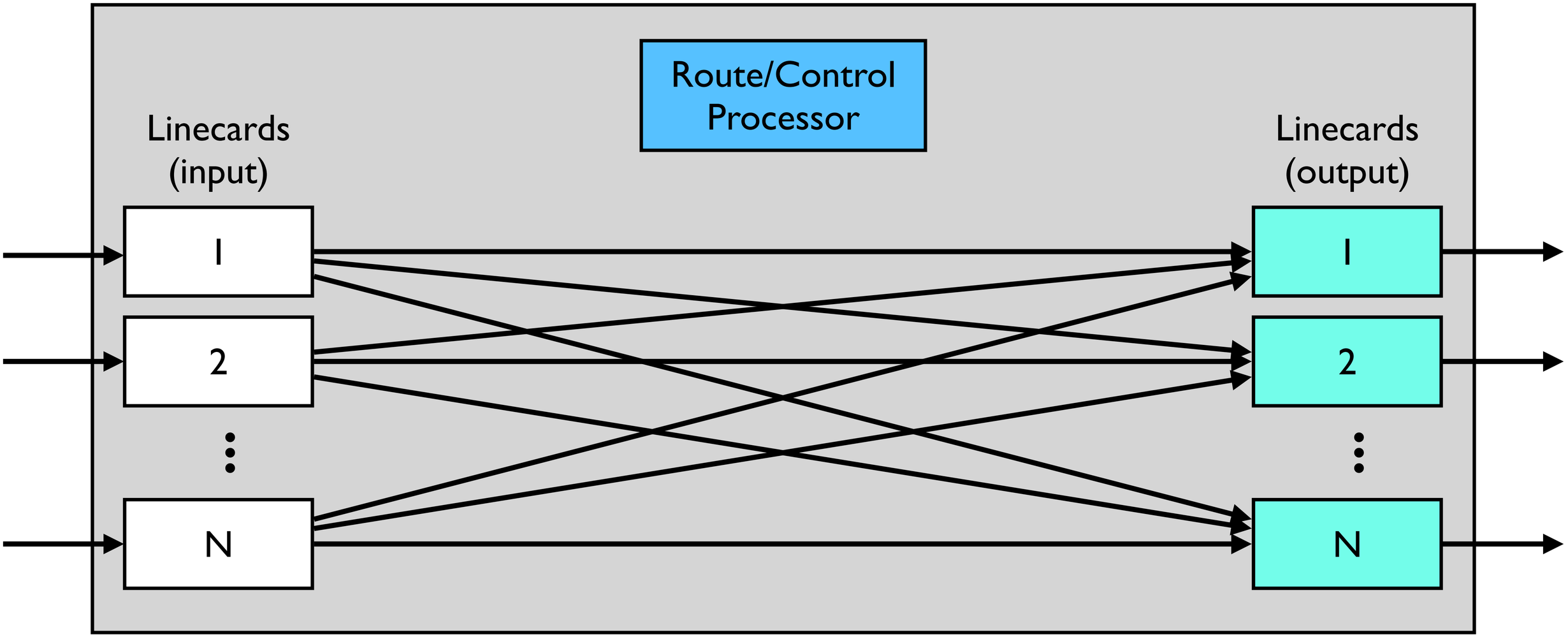
Output Queueing



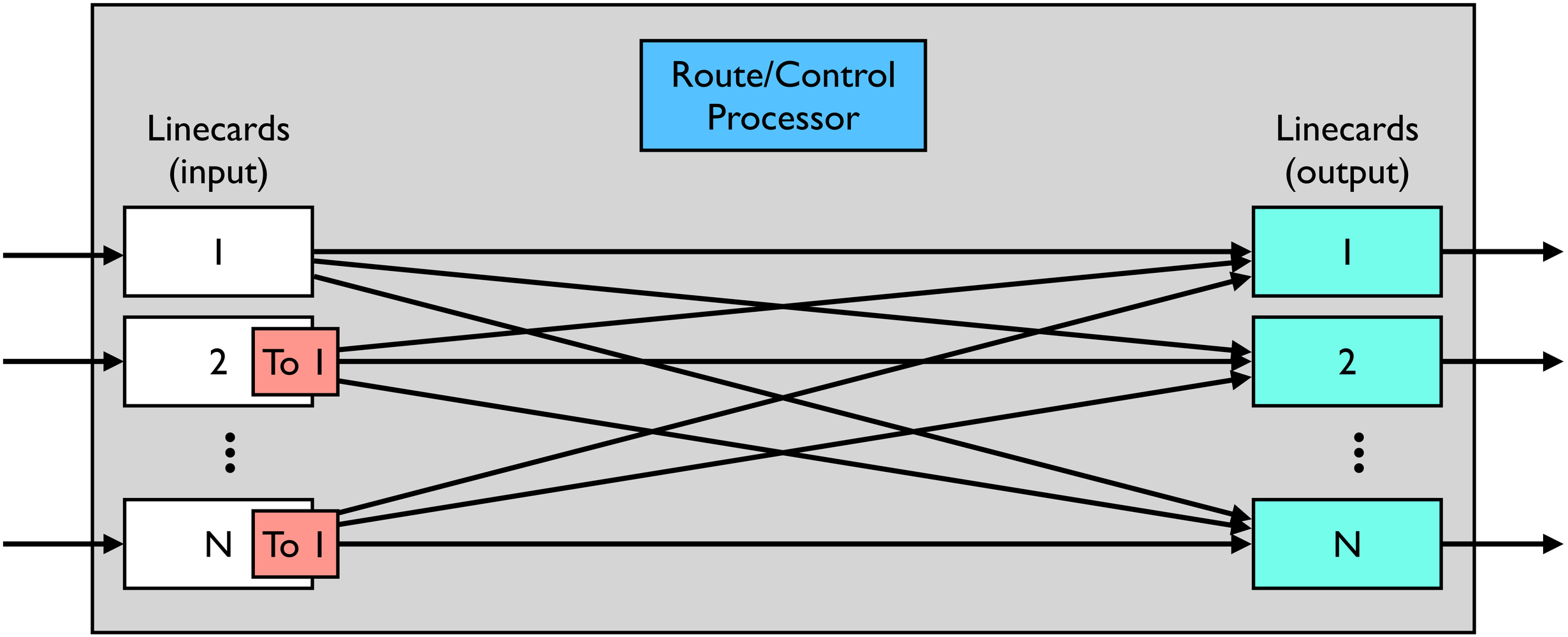
Output Queueing



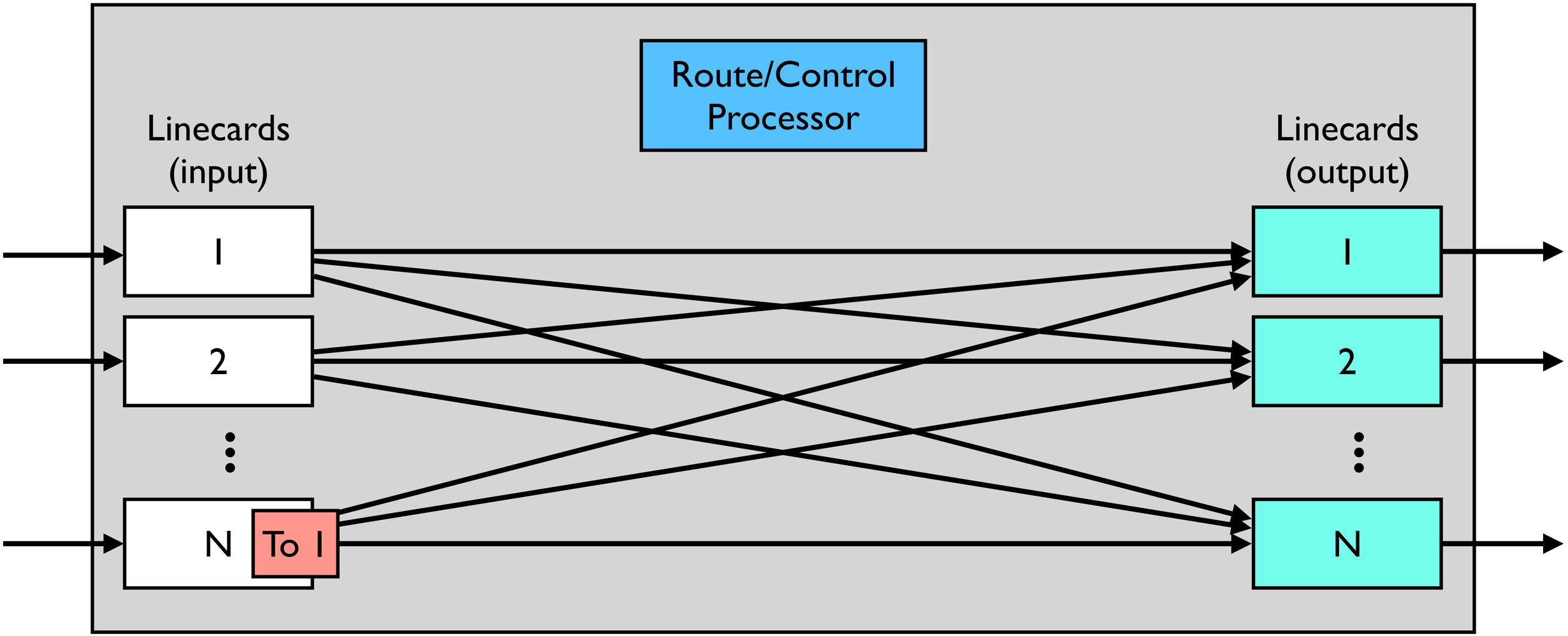
Output Queueing



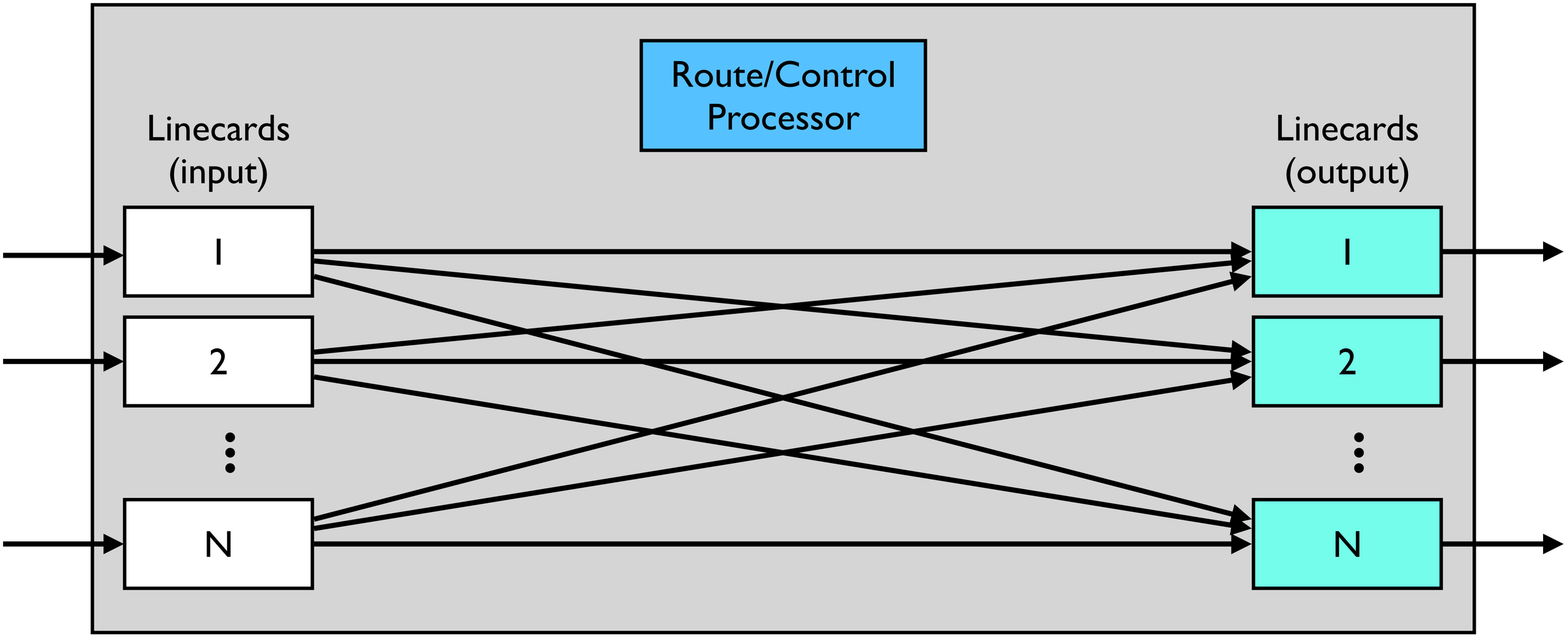
Input Queueing



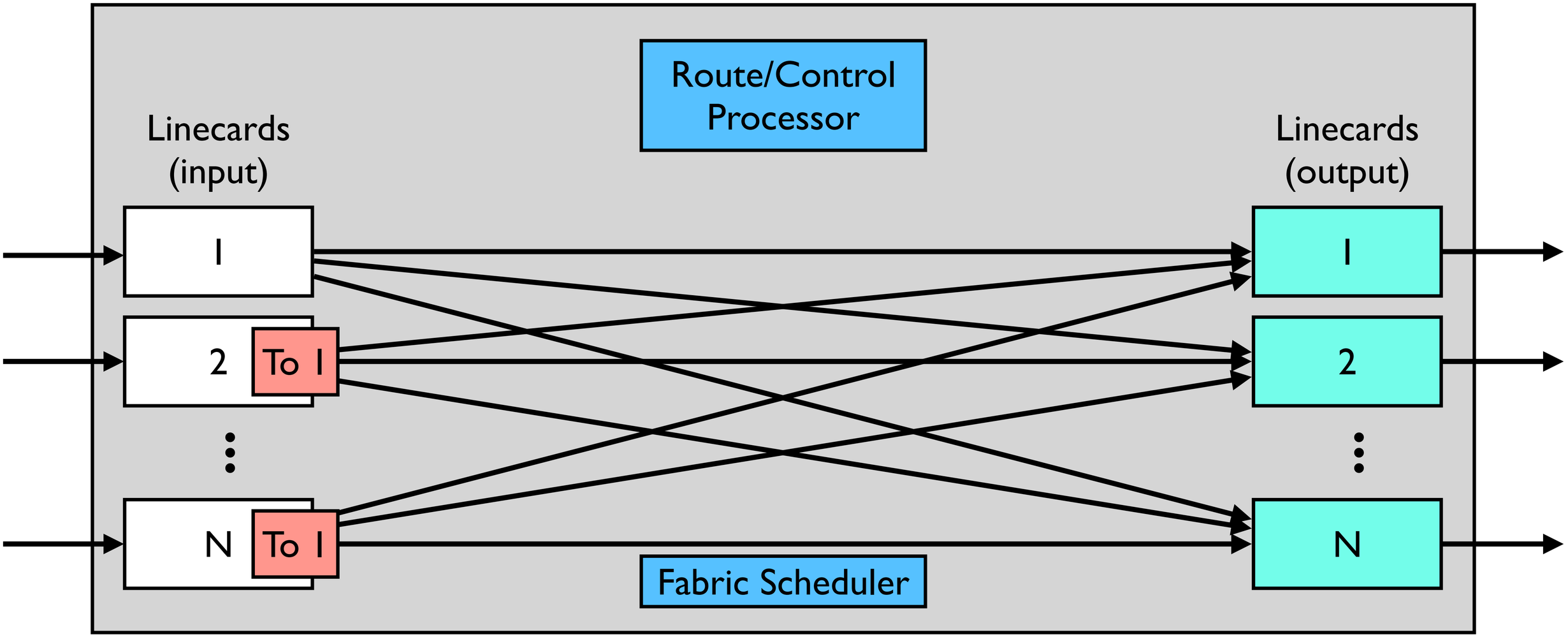
Input Queueing



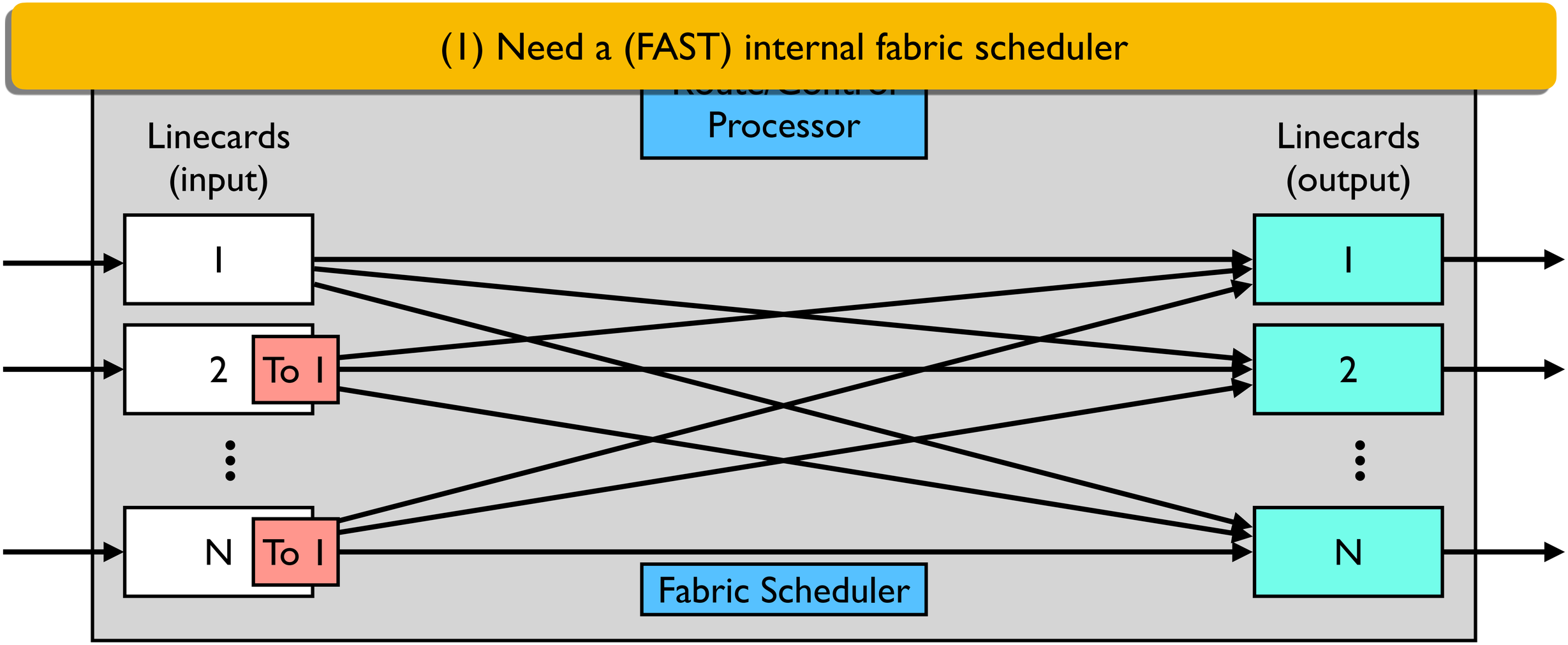
Input Queueing



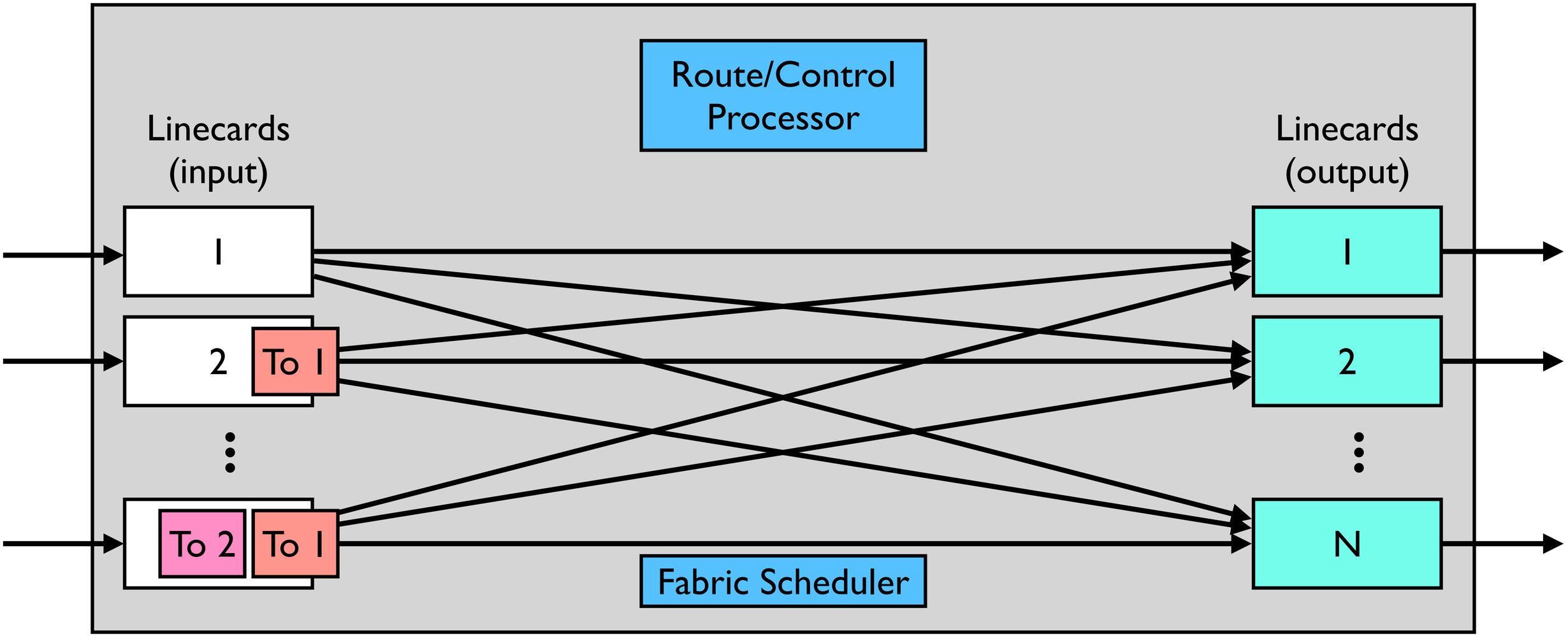
Input Queueing: Challenges



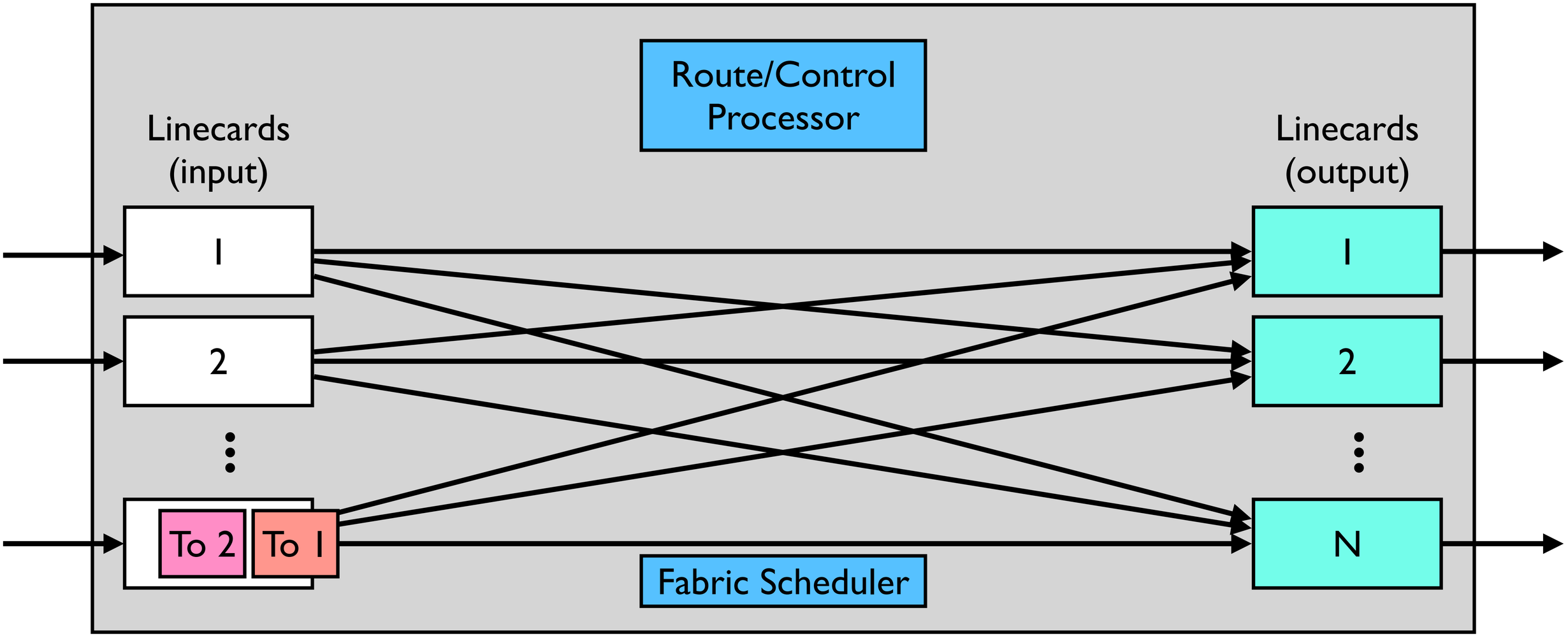
Input Queueing: Challenges



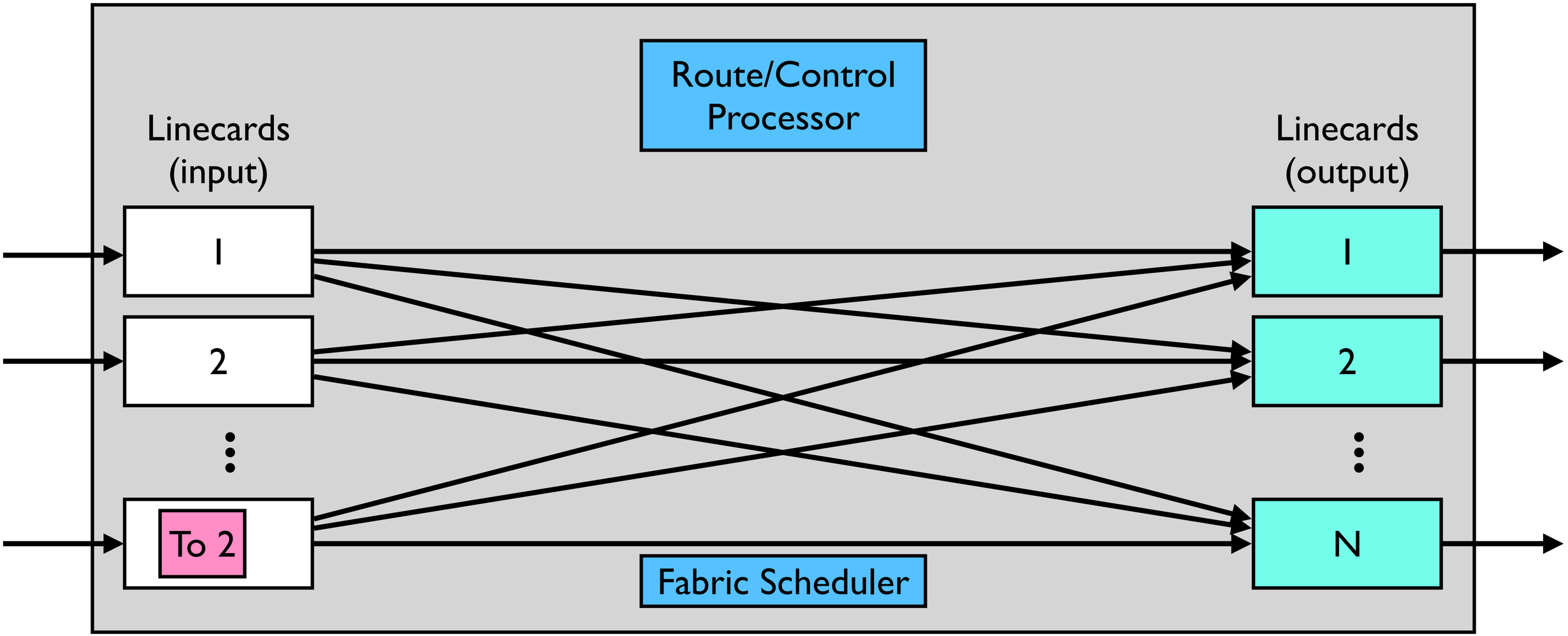
Input Queueing: Challenges



Input Queueing: Challenges

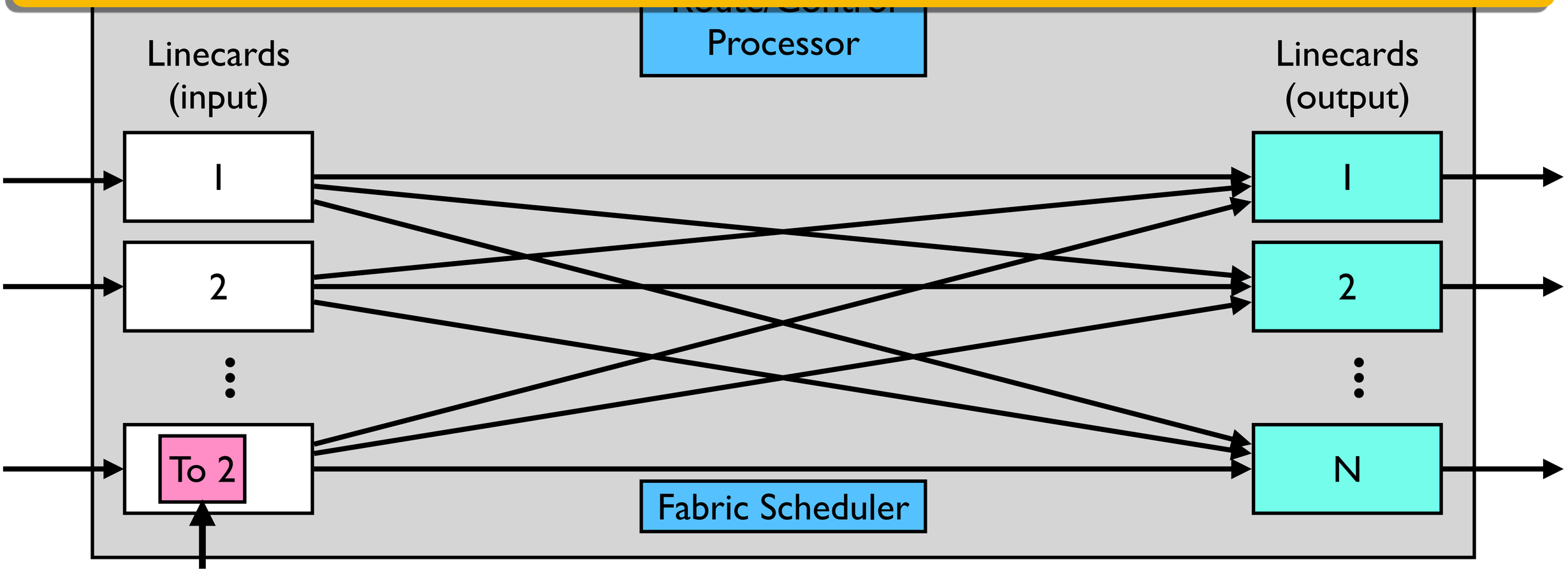


Input Queueing: Challenges

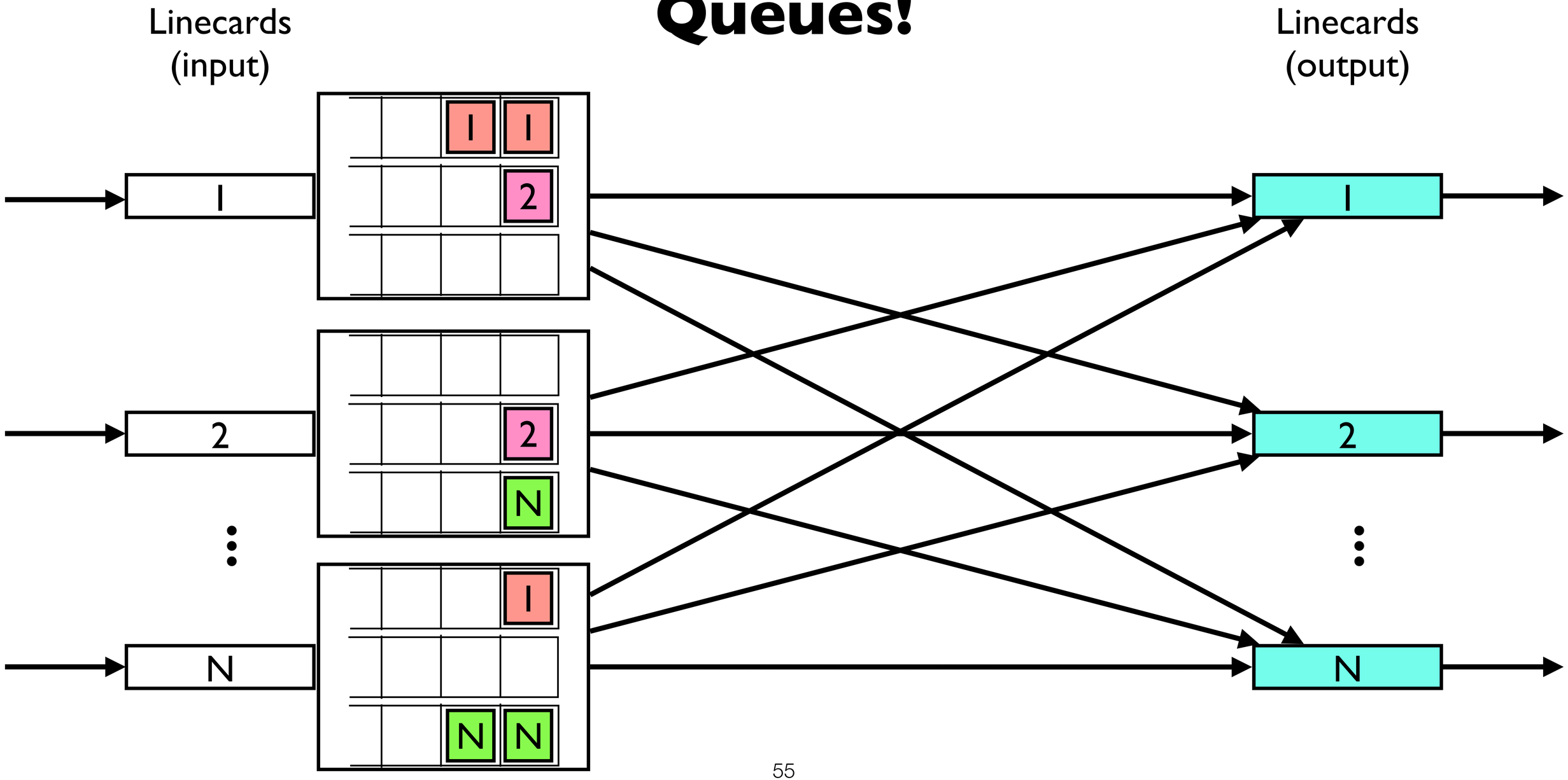


Input Queueing: Challenges

(2) Head of line blocking — **limits throughput to approximately 58% of capacity!**



Fixing head of line blocking: **Virtual Output Queues!**



Reality is more complicated

Reality is more complicated

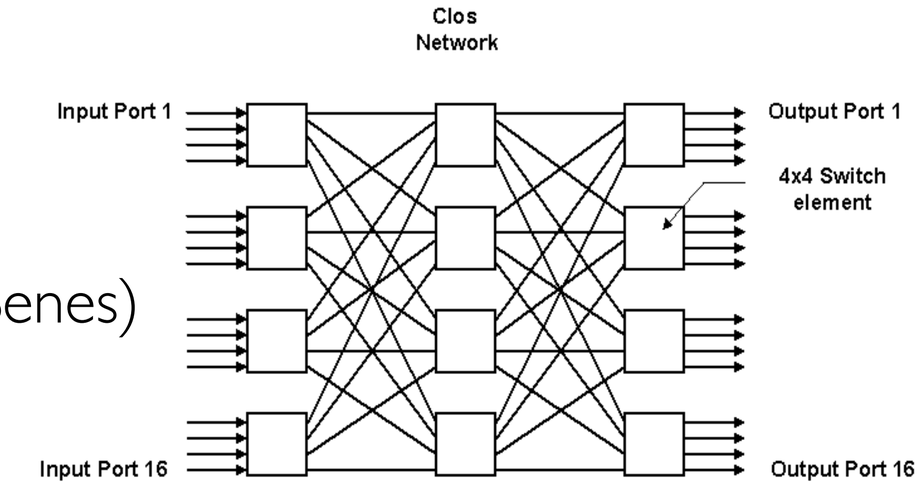
- **Commercial (high-speed) routers use**

Reality is more complicated

- **Commercial (high-speed) routers use**
  - Combination of input and output queueing

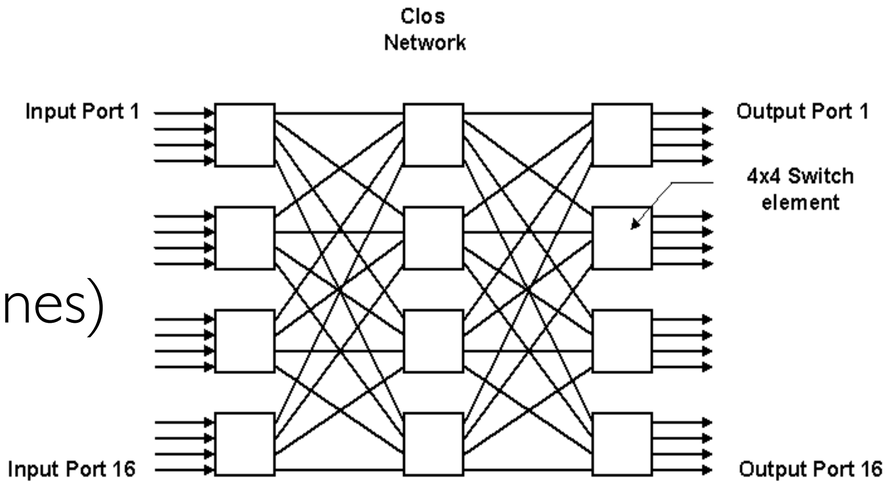
Reality is more complicated

- **Commercial (high-speed) routers use**
  - Combination of input and output queueing
  - Complex multi-stage switching topologies (Clos, Benes)



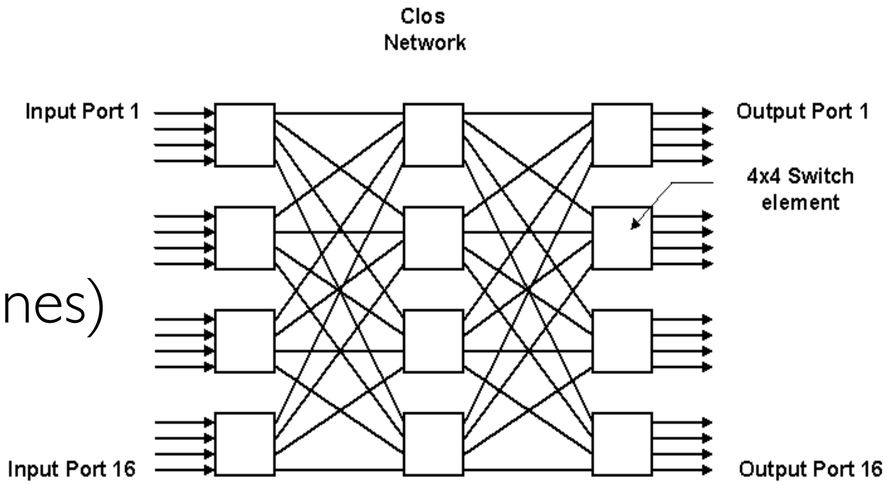
Reality is more complicated

- **Commercial (high-speed) routers use**
  - Combination of input and output queueing
  - Complex multi-stage switching topologies (Clos, Benes)
  - Distributed, multi-stage schedulers (for scalability)



Reality is more complicated

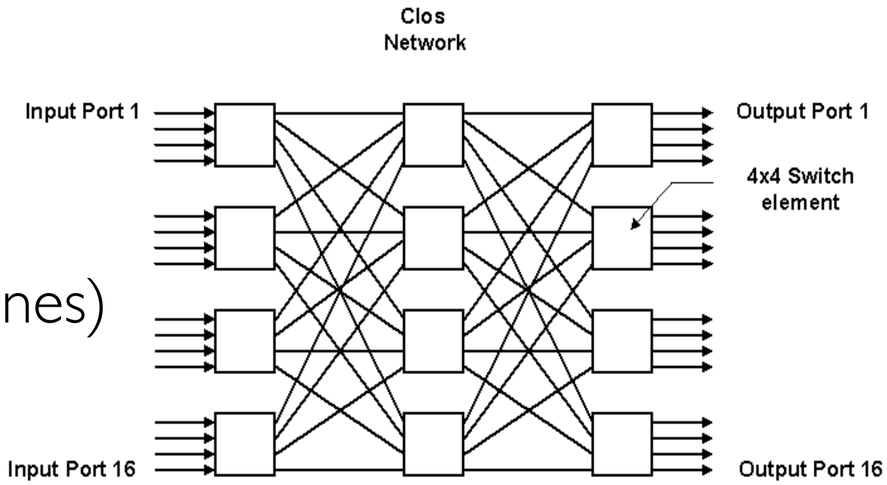
- **Commercial (high-speed) routers use**
  - Combination of input and output queueing
  - Complex multi-stage switching topologies (Clos, Benes)
  - Distributed, multi-stage schedulers (for scalability)



- **We'll consider one simple context**

Reality is more complicated

- **Commercial (high-speed) routers use**
  - Combination of input and output queueing
  - Complex multi-stage switching topologies (Clos, Benes)
  - Distributed, multi-stage schedulers (for scalability)



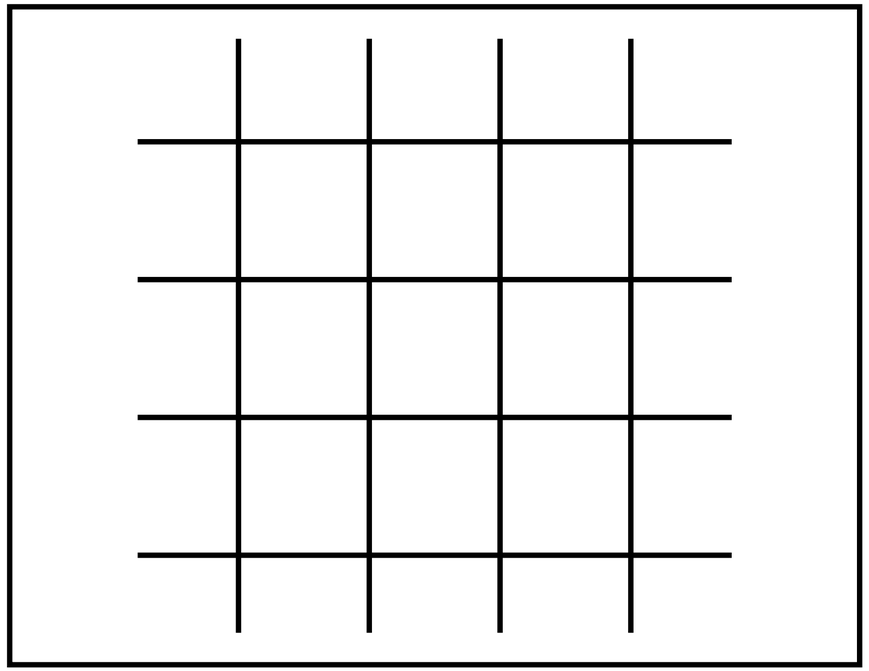
- **We'll consider one simple context**
  - De-facto architecture for a long time and still used in lower-speed routers

Context

- **Crossbar fabric**
- **Centralized scheduler**

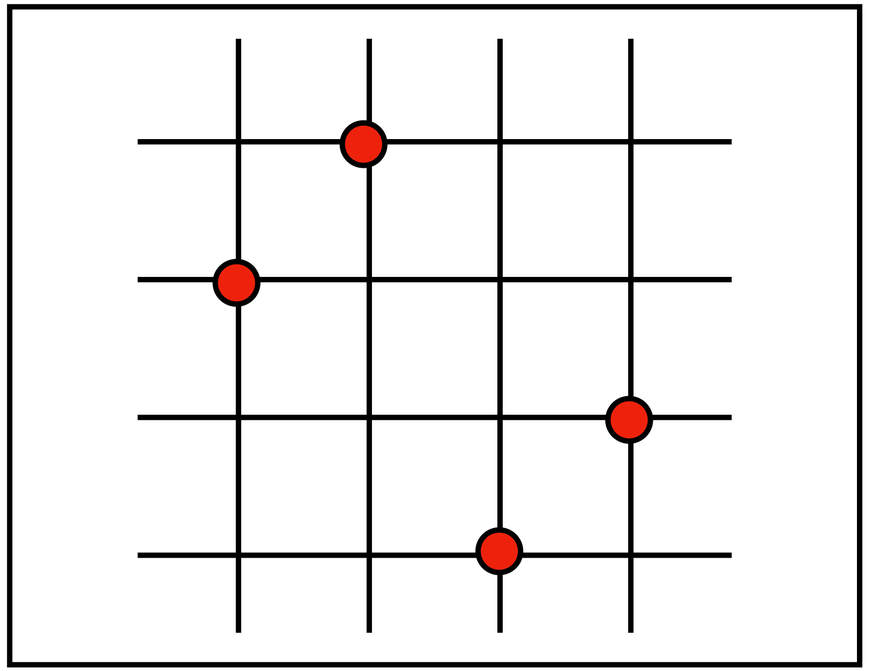
Context

- **Crossbar fabric**
- **Centralized scheduler**

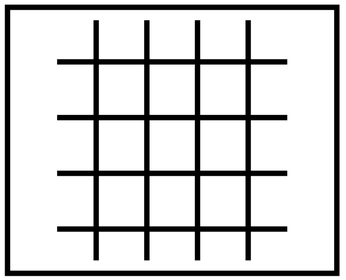


Context

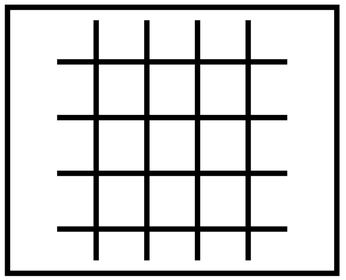
- **Crossbar fabric**
- **Centralized scheduler**



Scheduling

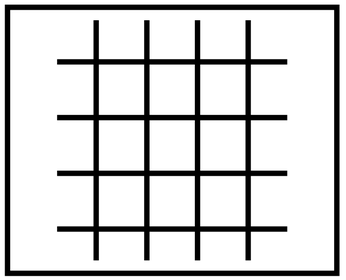


Scheduling



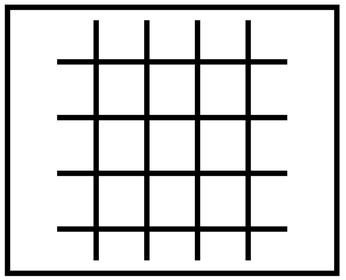
- **Goal: run links at full capacity, fairness across inputs**

Scheduling

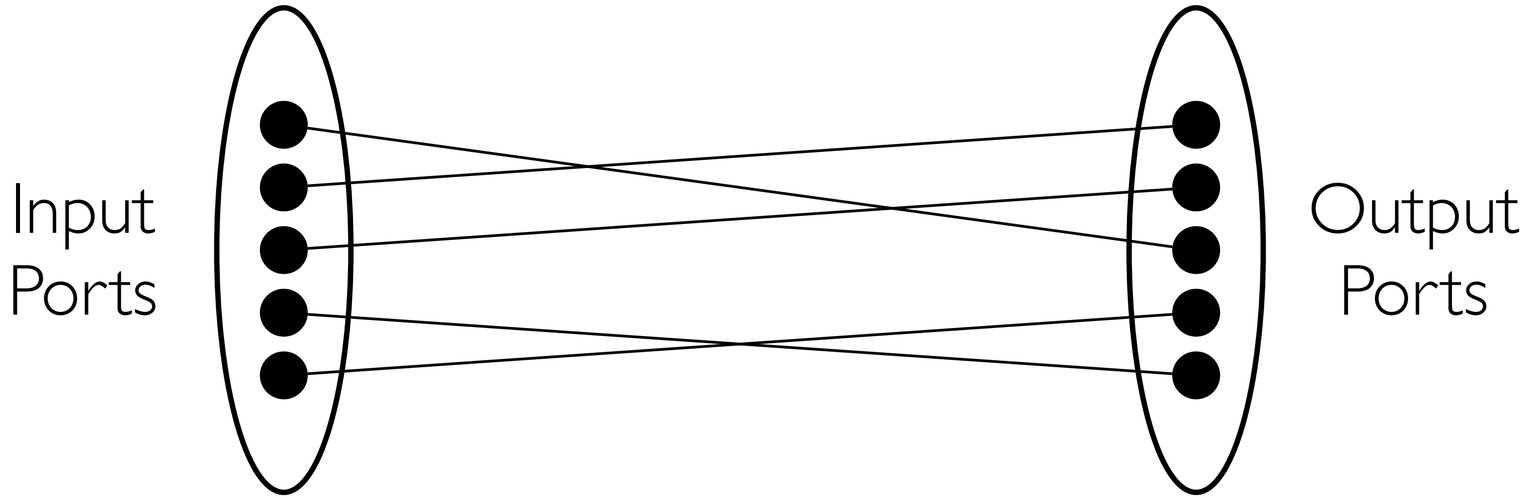


- **Goal: run links at full capacity, fairness across inputs**
- **Scheduling formulated as finding a matching on a bipartite graph**

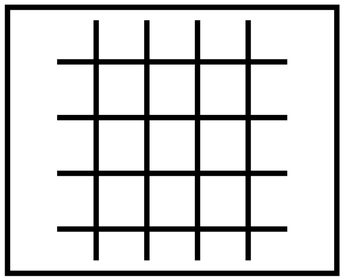
Scheduling



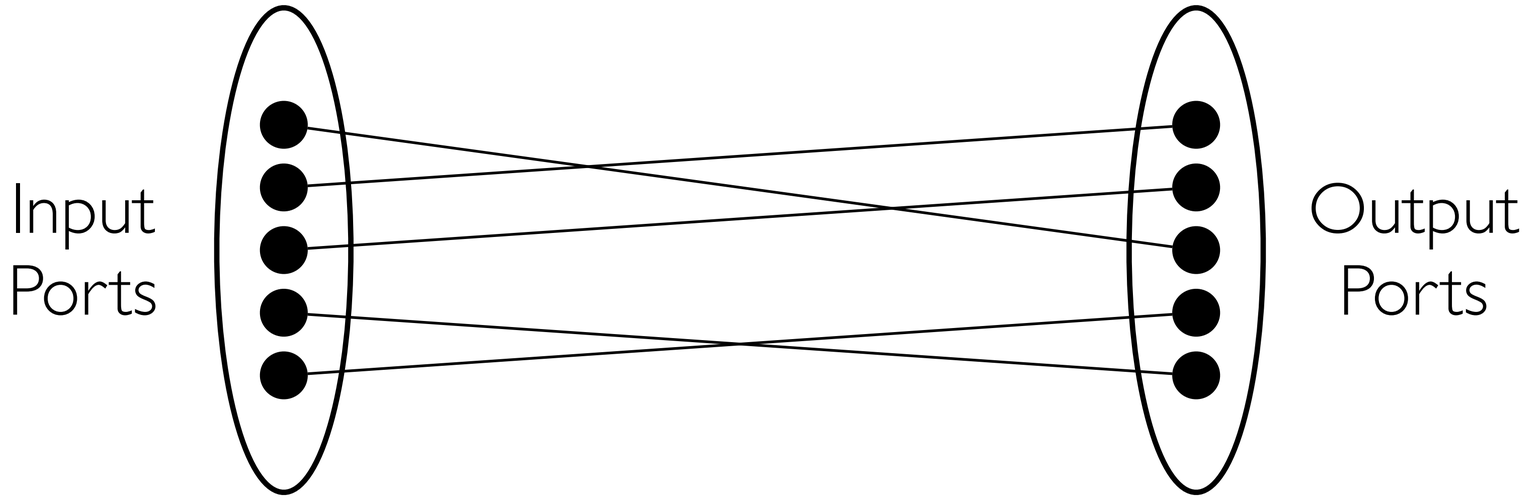
- **Goal: run links at full capacity, fairness across inputs**
- **Scheduling formulated as finding a matching on a bipartite graph**



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)**

Summary: IP Routers

Summary: IP Routers

- **$$$**

Summary: IP Routers

- **$$$**
- **Core building block of Internet**

Summary: IP Routers

- **$$$**
- **Core building block of Internet**
- **Line cards receive packets, change headers**

Summary: IP Routers

- **$$$**
- **Core building block of Internet**
- **Line cards receive packets, change headers**
- **Scalable addressing → Longest Prefix Matching**

Summary: IP Routers

- **$$$**
- **Core building block of Internet**
- **Line cards receive packets, change headers**
- **Scalable addressing → Longest Prefix Matching**
- **Need fast implementations for:**
  - Longest prefix matching
  - Switch fabric scheduling

Taking Stock: Network Layer

Taking Stock: Network Layer

- ***Best-effort global delivery of packets***

Taking Stock: Network Layer

- ***Best-effort global delivery of packets***
- **Control Plane:** Routing

Taking Stock: Network Layer

- ***Best-effort global delivery of packets***
- **Control Plane:** Routing
- **Data Plane:** Forwarding

Taking Stock: Network Layer

- ***Best-effort global delivery of packets***
- **Control Plane:** Routing
- **Data Plane:** Forwarding
- **Key enabler of scalability:** Addressing

Addressing: Hierarchical for Scalability

Addressing: Hierarchical for Scalability

- **Hierarchical address structure**

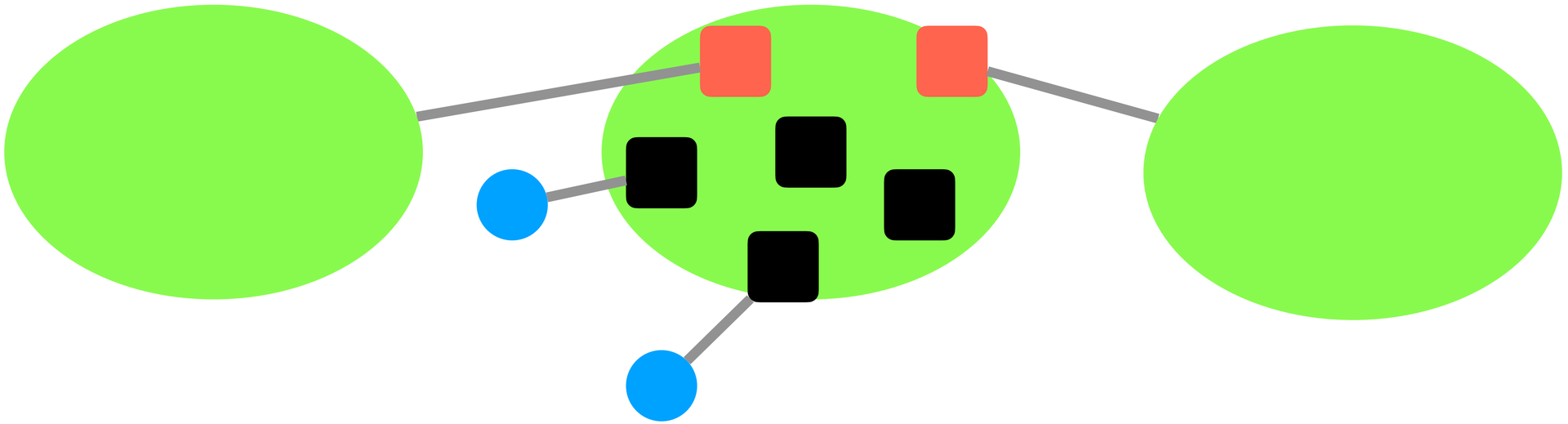
Addressing: Hierarchical for Scalability

- **Hierarchical address structure**
- **Hierarchical address allocation**

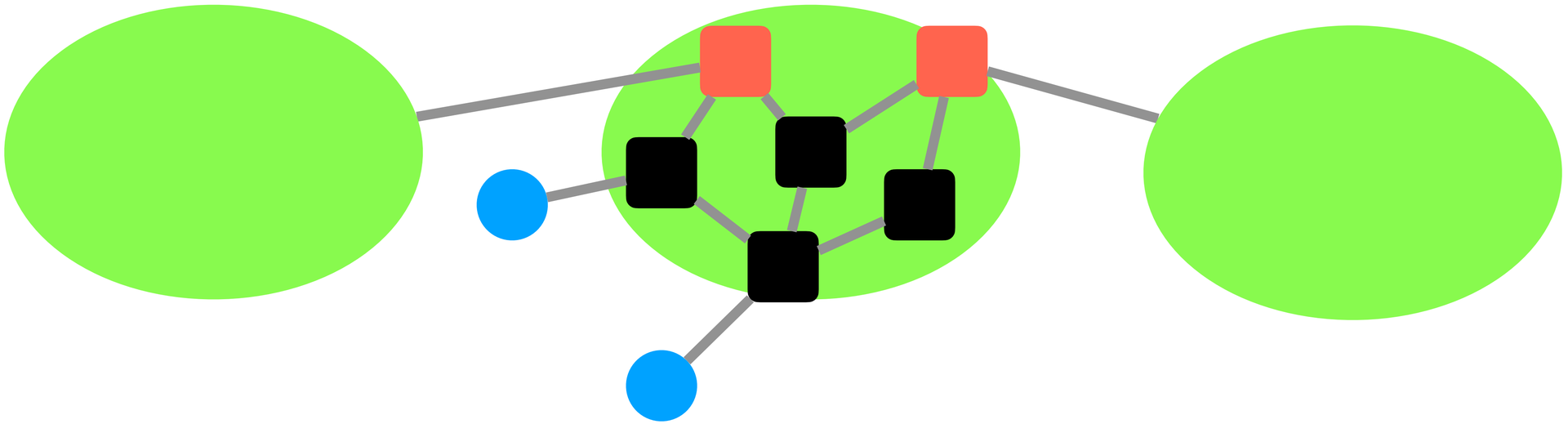
Addressing: Hierarchical for Scalability

- **Hierarchical address structure**
- **Hierarchical address allocation**
- **Hierarchical addresses and routing scalability**

Control Plane: **Routing**

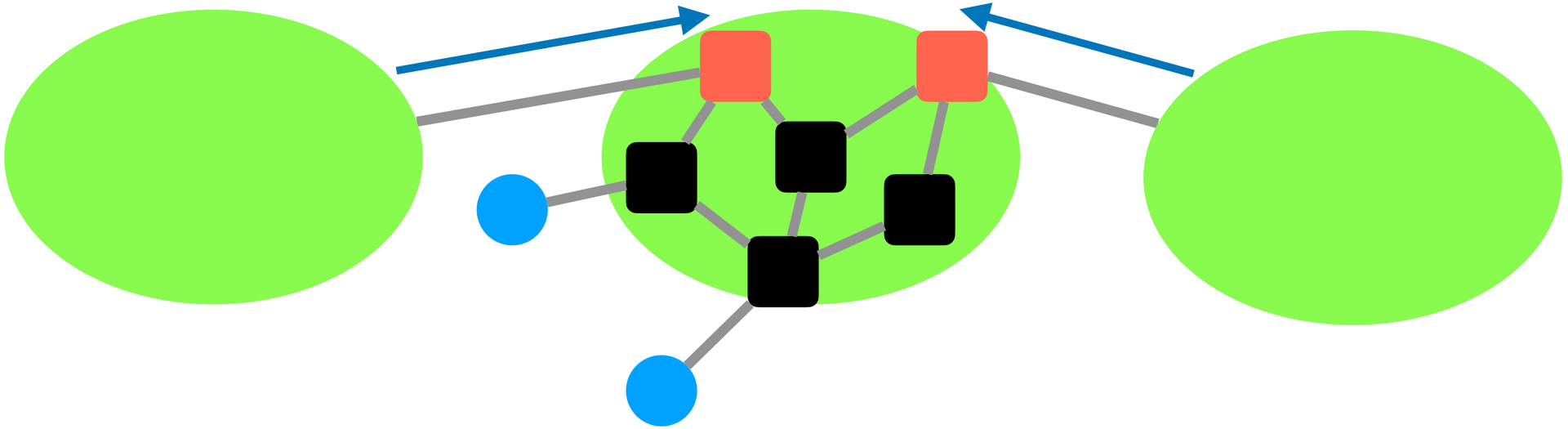


Control Plane: **Routing**



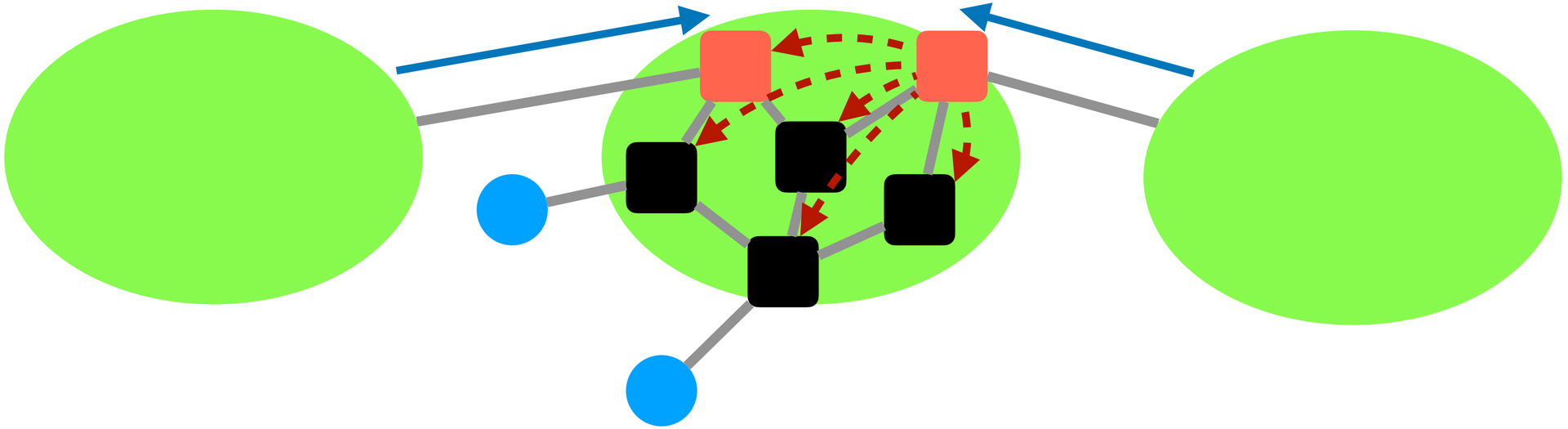
1. Provide internal reachability (**IGP**)

Control Plane: **Routing**



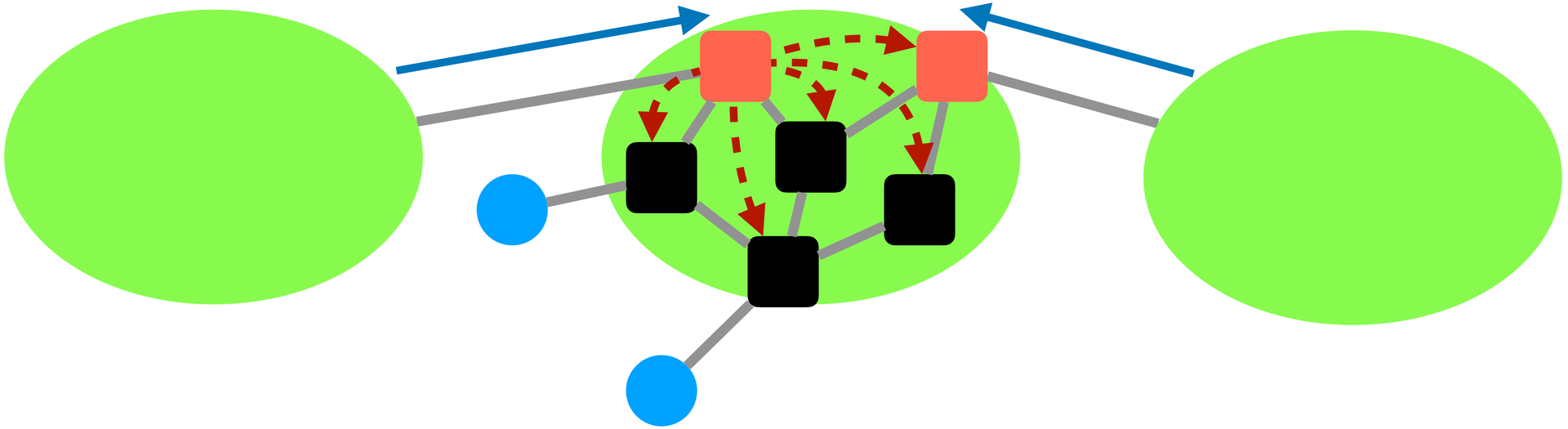
1. Provide internal reachability (**IGP**)
2. Learn routes to external destinations (**eBGP**)

Control Plane: **Routing**



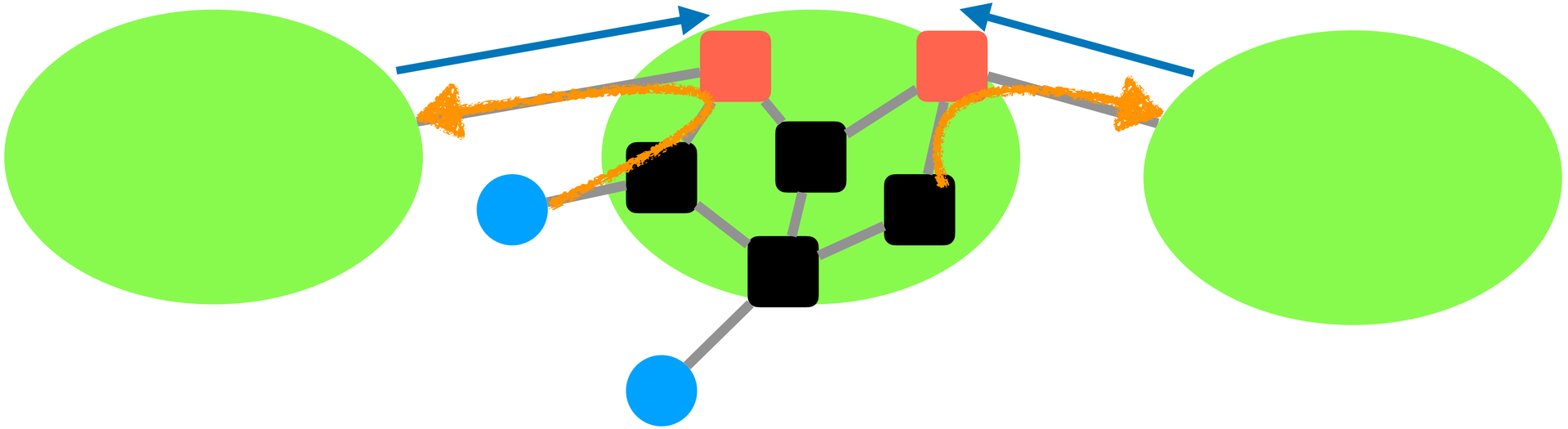
1. Provide internal reachability (**IGP**)
2. Learn routes to external destinations (**eBGP**)
3. Distribute externally learned routes internally (**iBGP**)

Control Plane: **Routing**



1. Provide internal reachability (**IGP**)
2. Learn routes to external destinations (**eBGP**)
3. Distribute externally learned routes internally (**iBGP**)

Control Plane: **Routing**

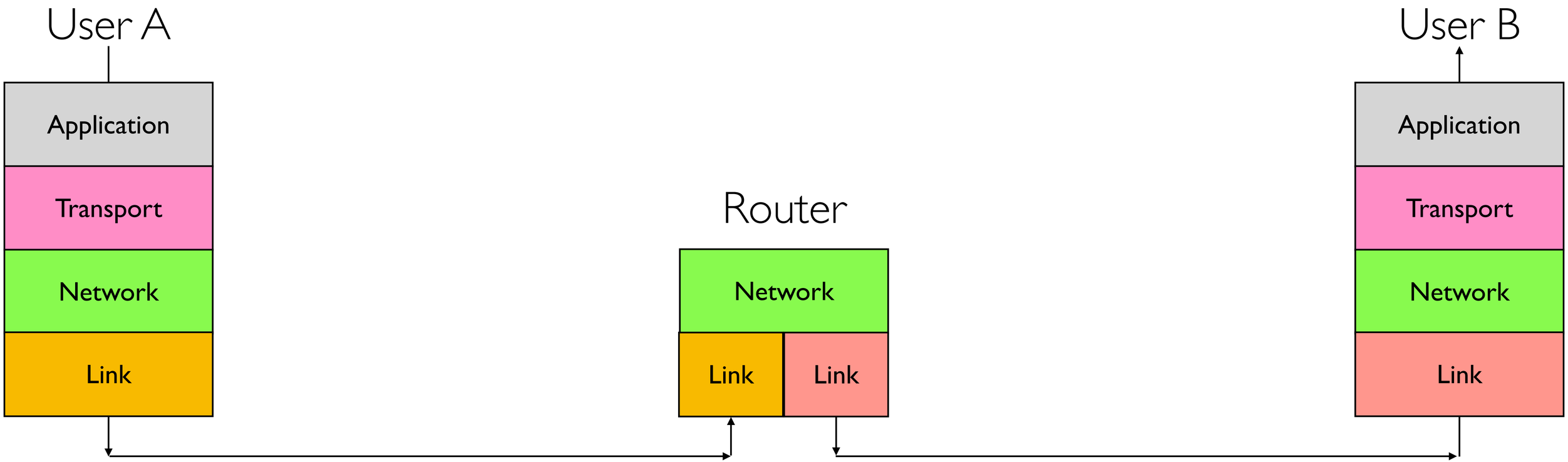


1. Provide internal reachability (**IGP**)
2. Learn routes to external destinations (**eBGP**)
3. Distribute externally learned routes internally (**iBGP**)
4. Travel shortest path to egress (**IGP**)

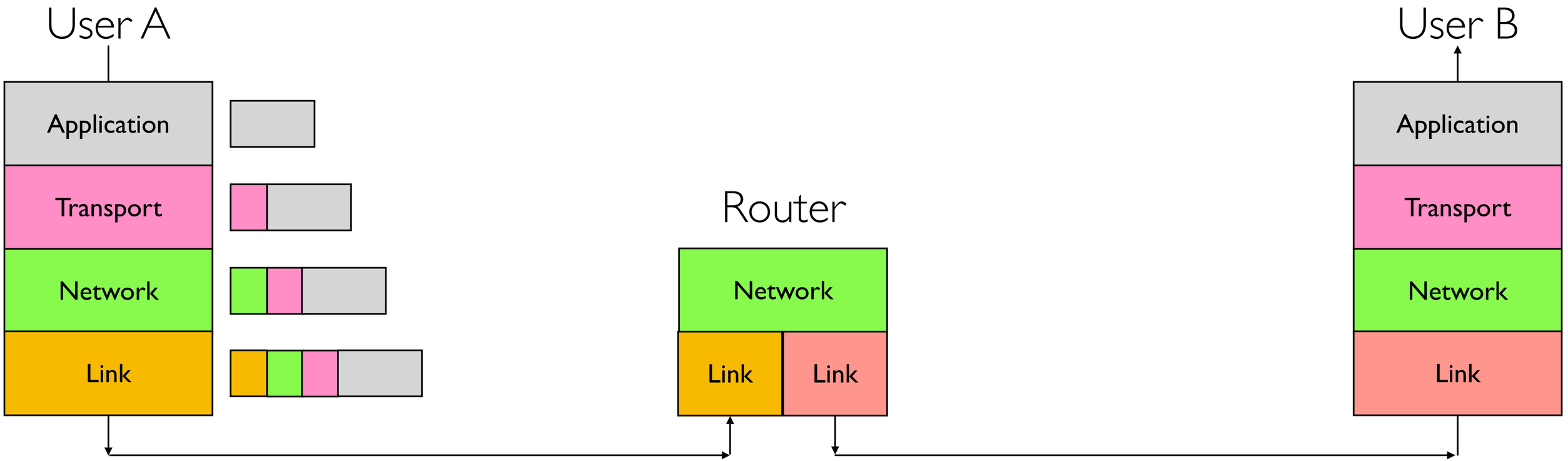
Data Plane: **Forwarding**

Data Plane: **Forwarding**

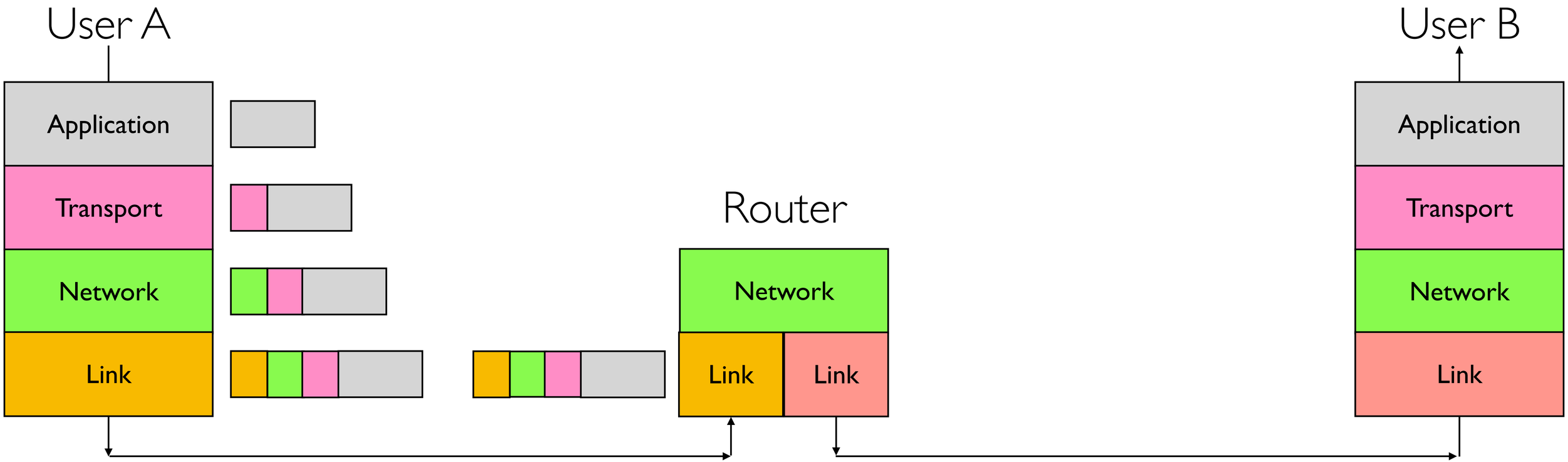
Data Plane: **Forwarding**



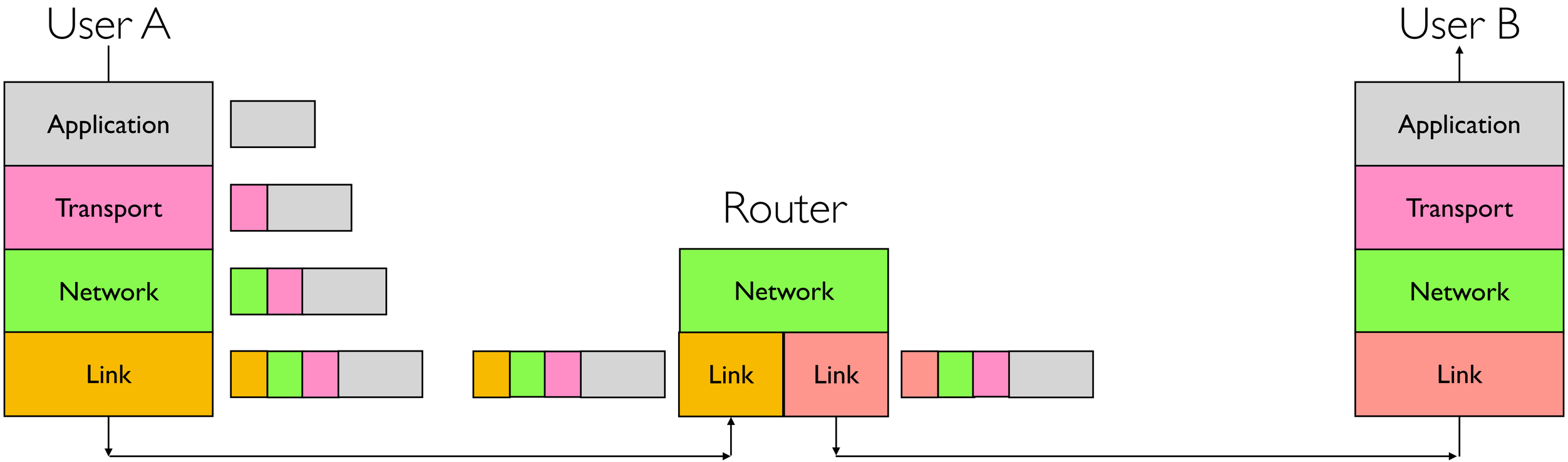
Data Plane: **Forwarding**



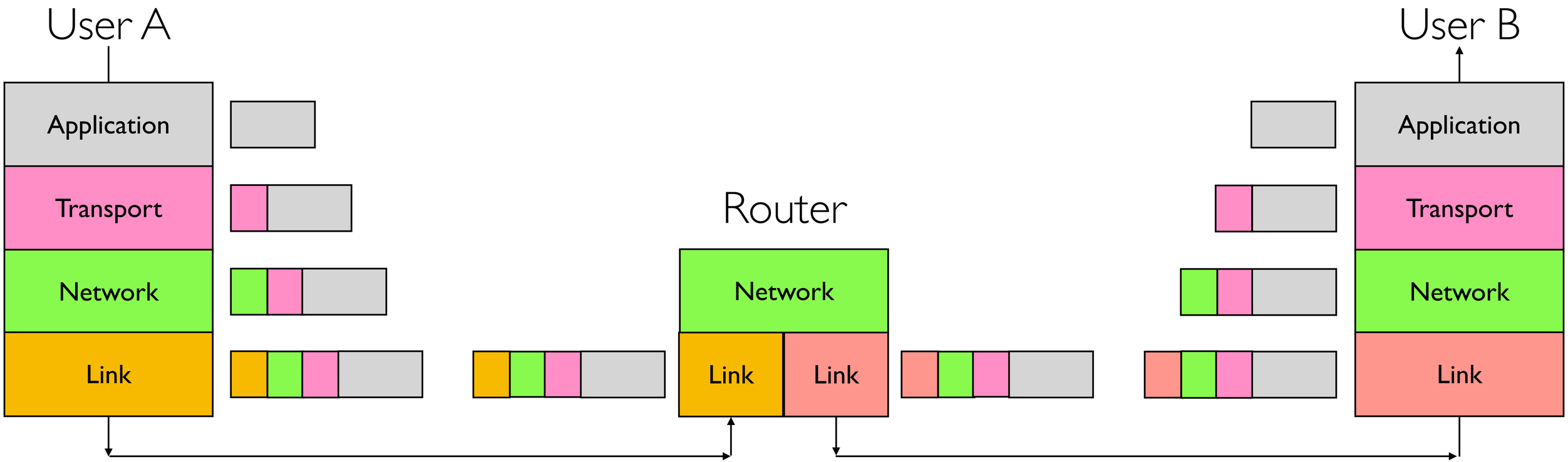
Data Plane: **Forwarding**



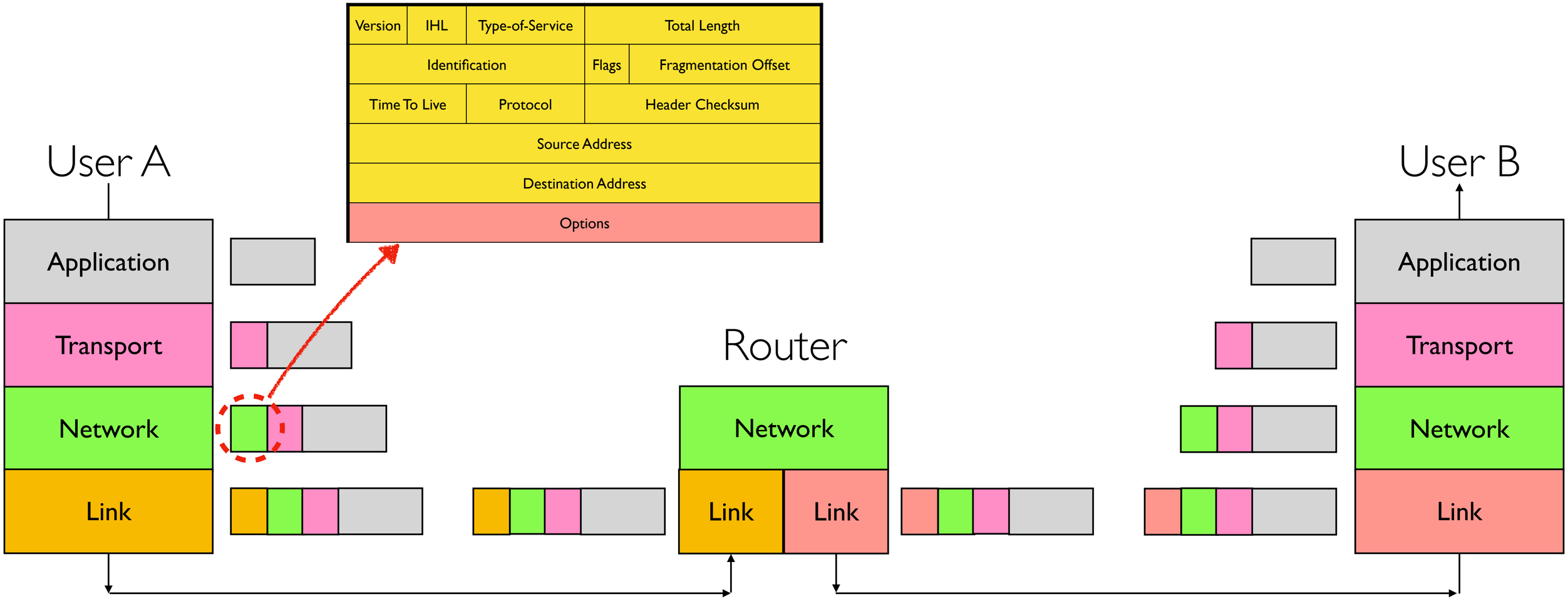
Data Plane: **Forwarding**



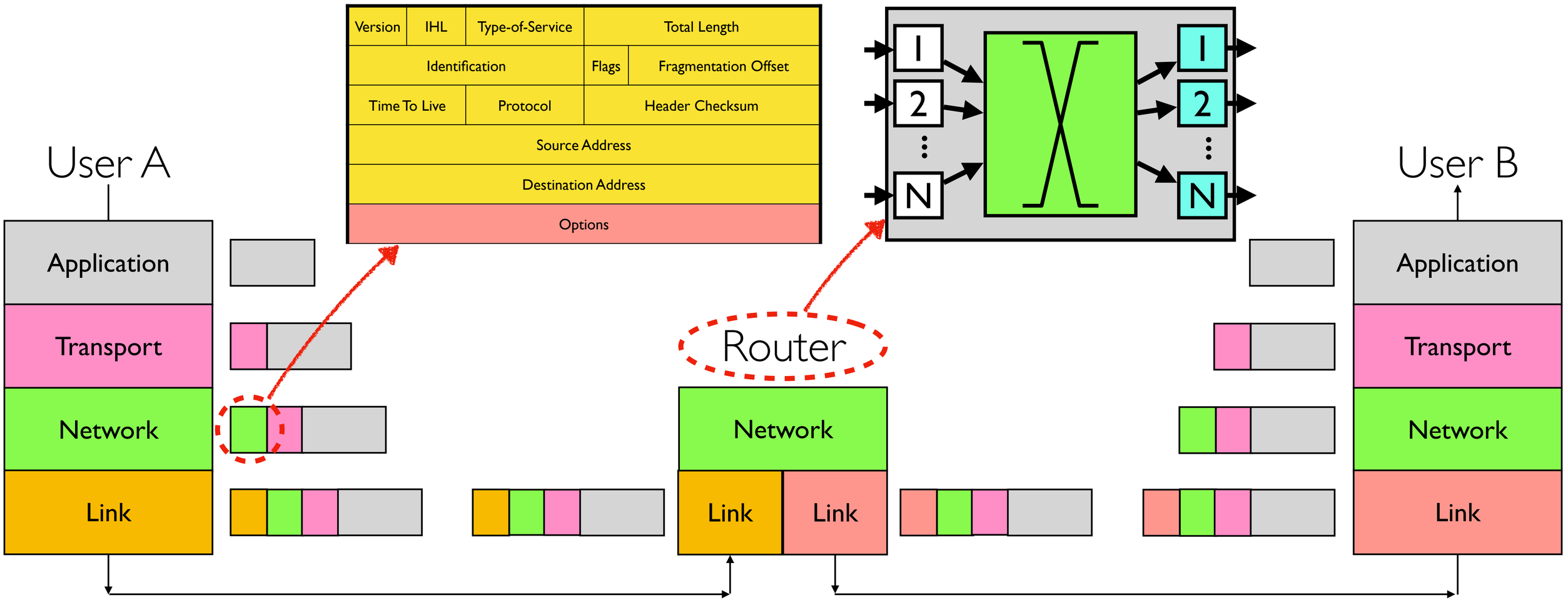
Data Plane: **Forwarding**



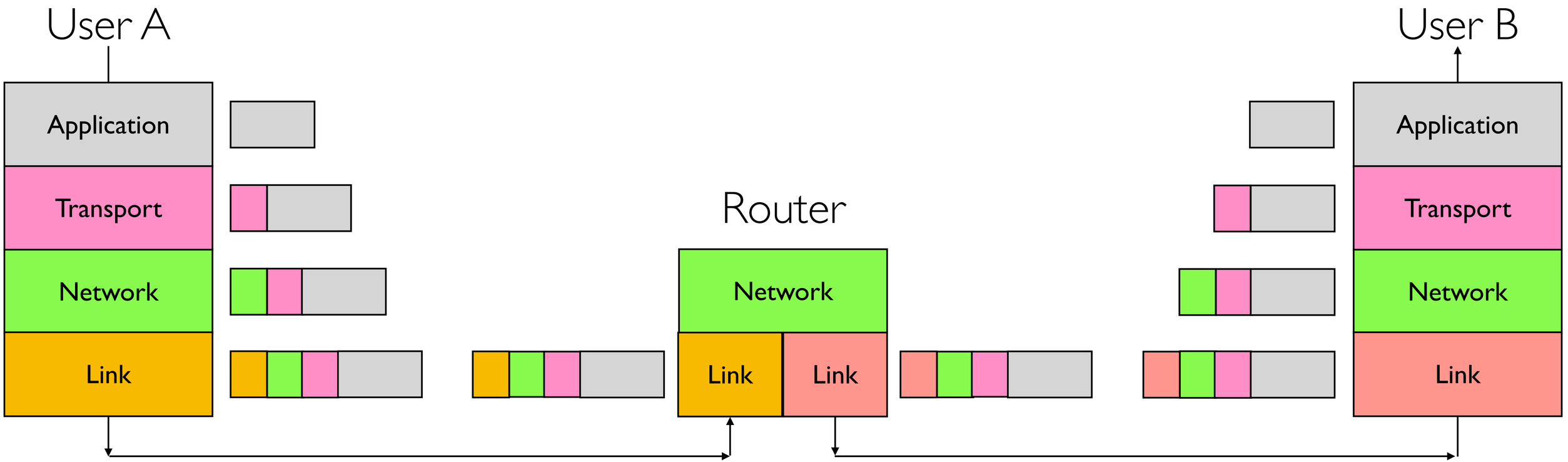
Data Plane: **Forwarding**



Data Plane: **Forwarding**

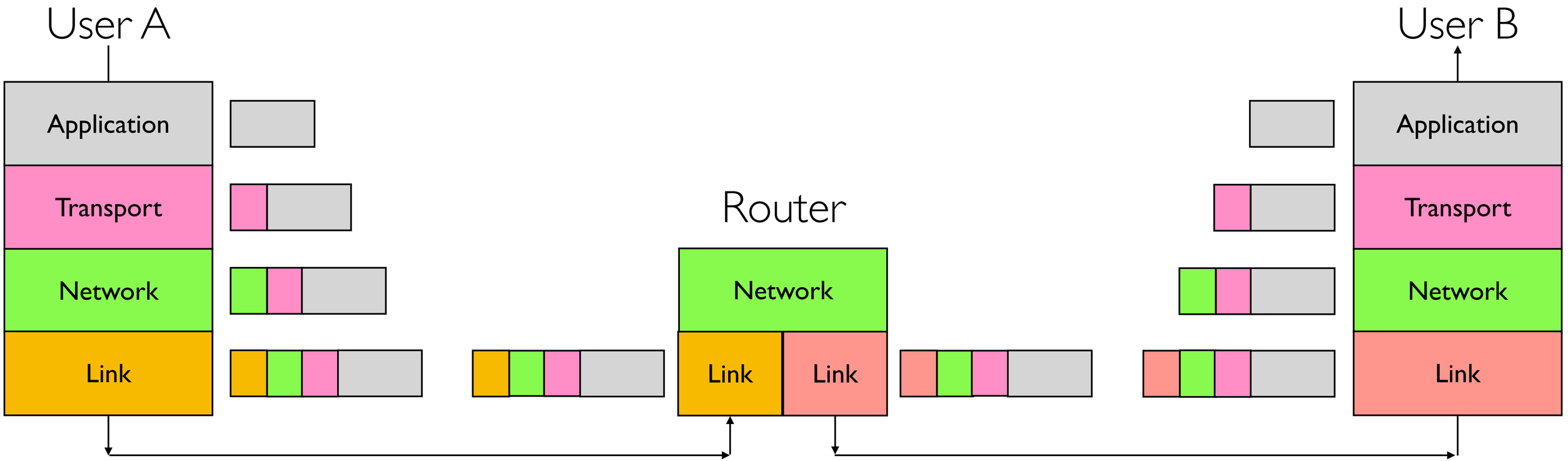


Looking Ahead



Looking Ahead

- **So far: Network Layer,** *Best-effort global delivery of packets*



Looking Ahead

- **So far: Network Layer,** *Best-effort global delivery of packets*
- **Next: Transport Layer,** *Reliable (or unreliable) delivery of data*