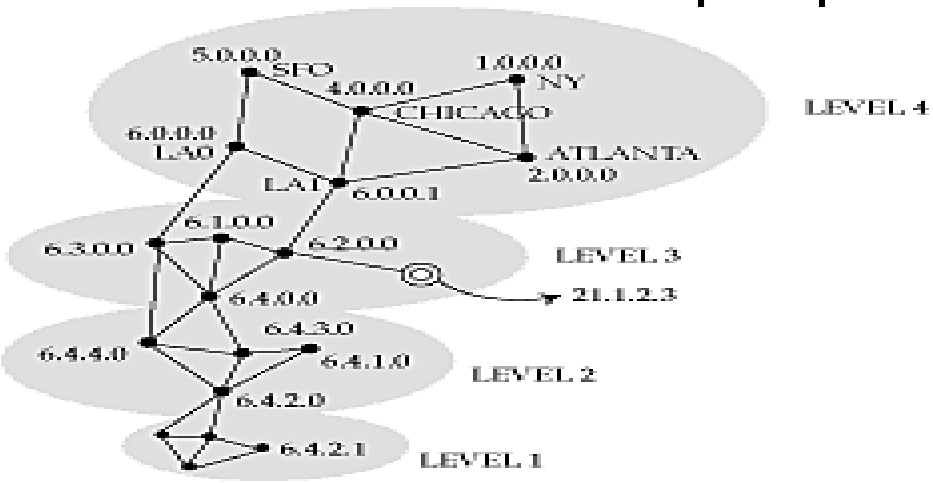
Switch and Router Architectures

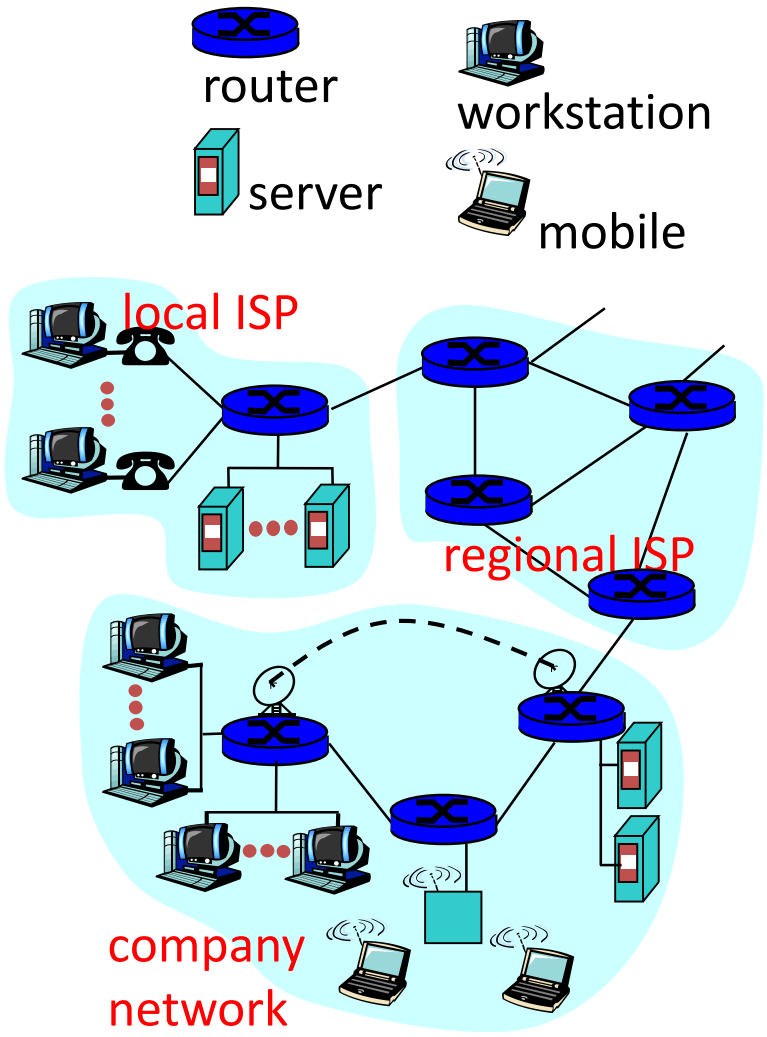
What is it all about?

- How do we move traffic from one part of the network to another?
- Connect end-systems to switches, and switches to each other by links
- Data arriving to an input port of a switch have to be moved to one or more of the output ports

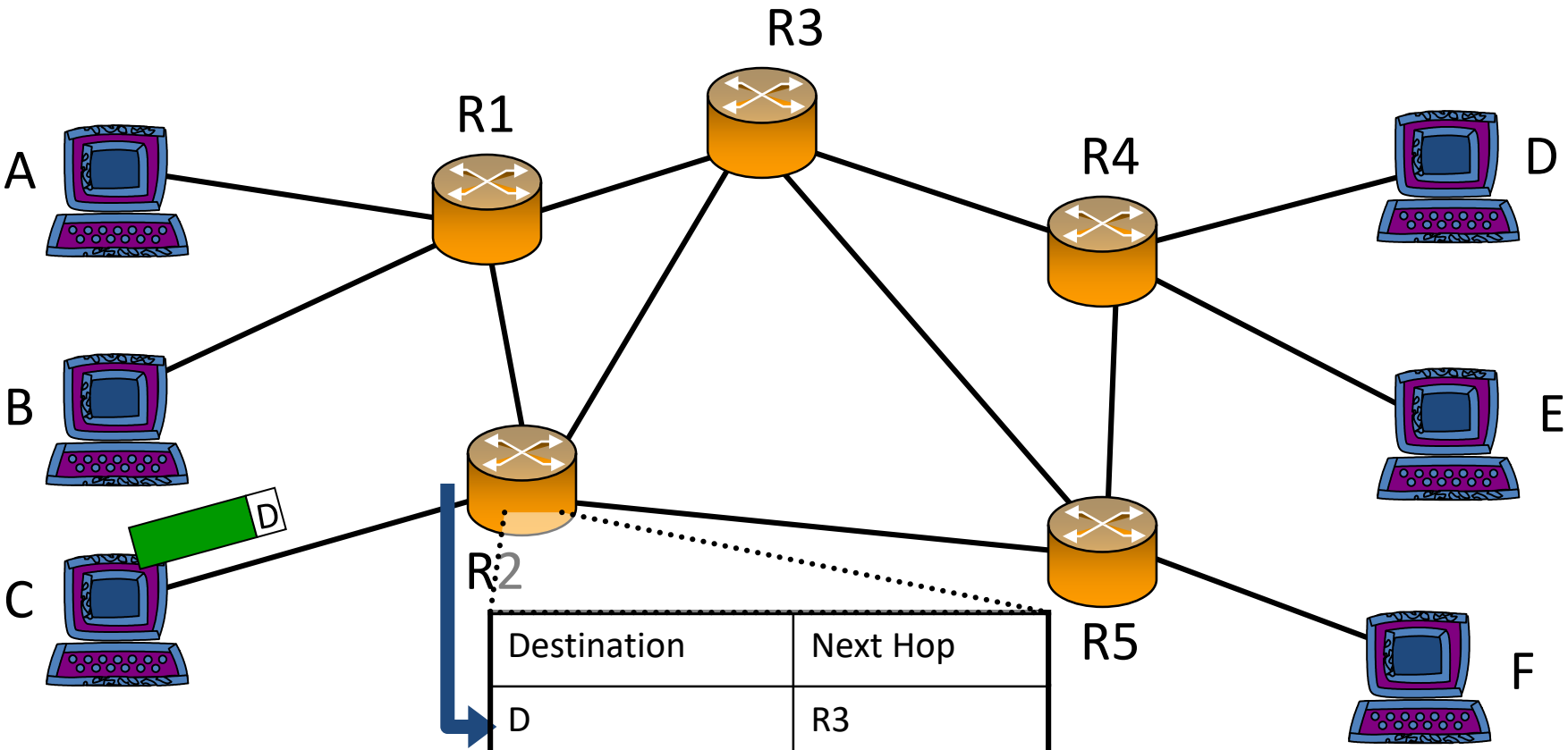


What's the Internet: "nuts and bolts" view

- *Internet:* "network of networks"
  - Any to any reachability
  - but loosely hierarchical
  - Routing protocols populate routing tables in the routers
- Traffic Aggregation
  - Through multiplexing and switching
  - Access Networks
  - Edge
  - Core

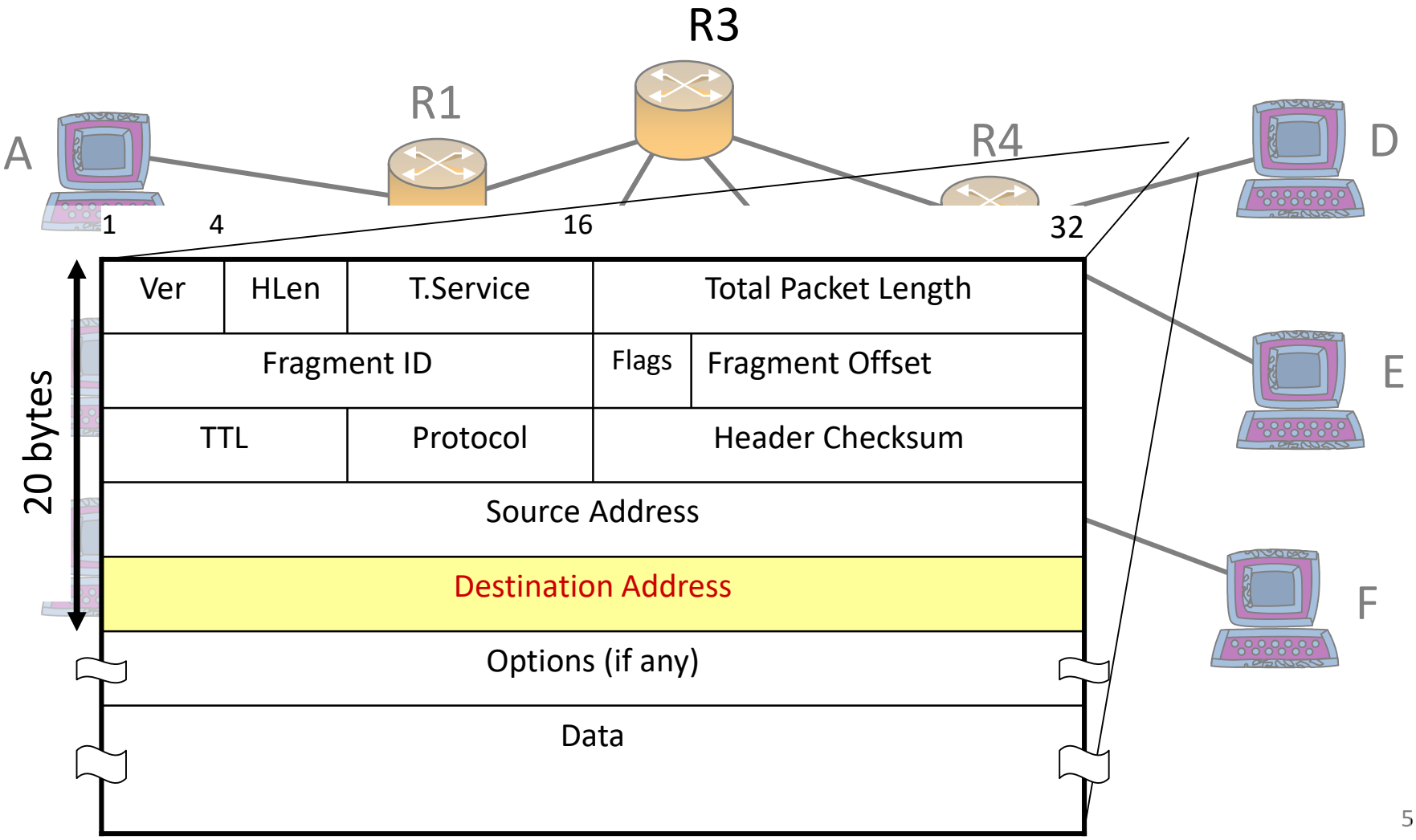


What is Routing?

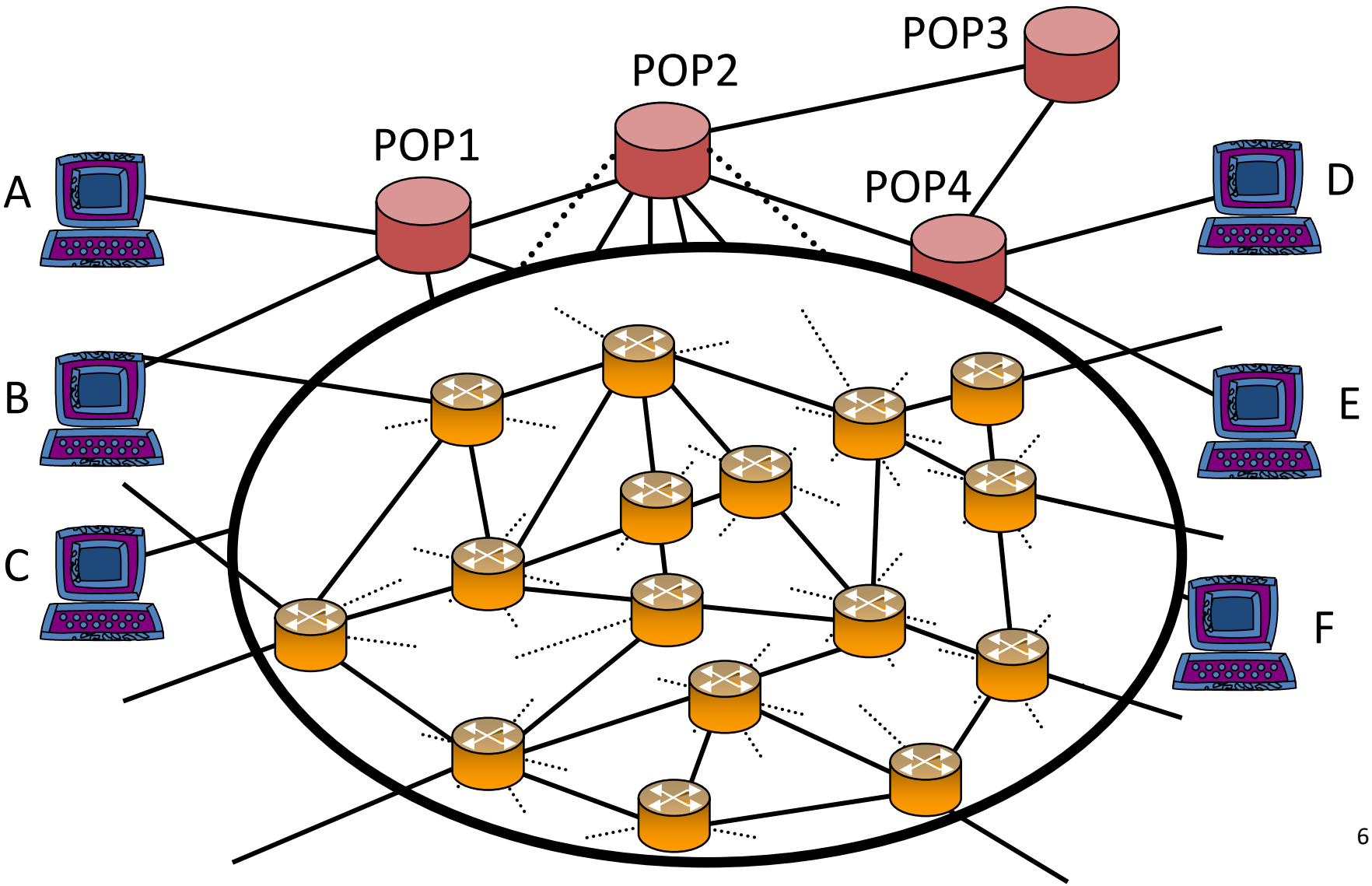


Destination	Next Hop
D	R3
E	R3
F	R5

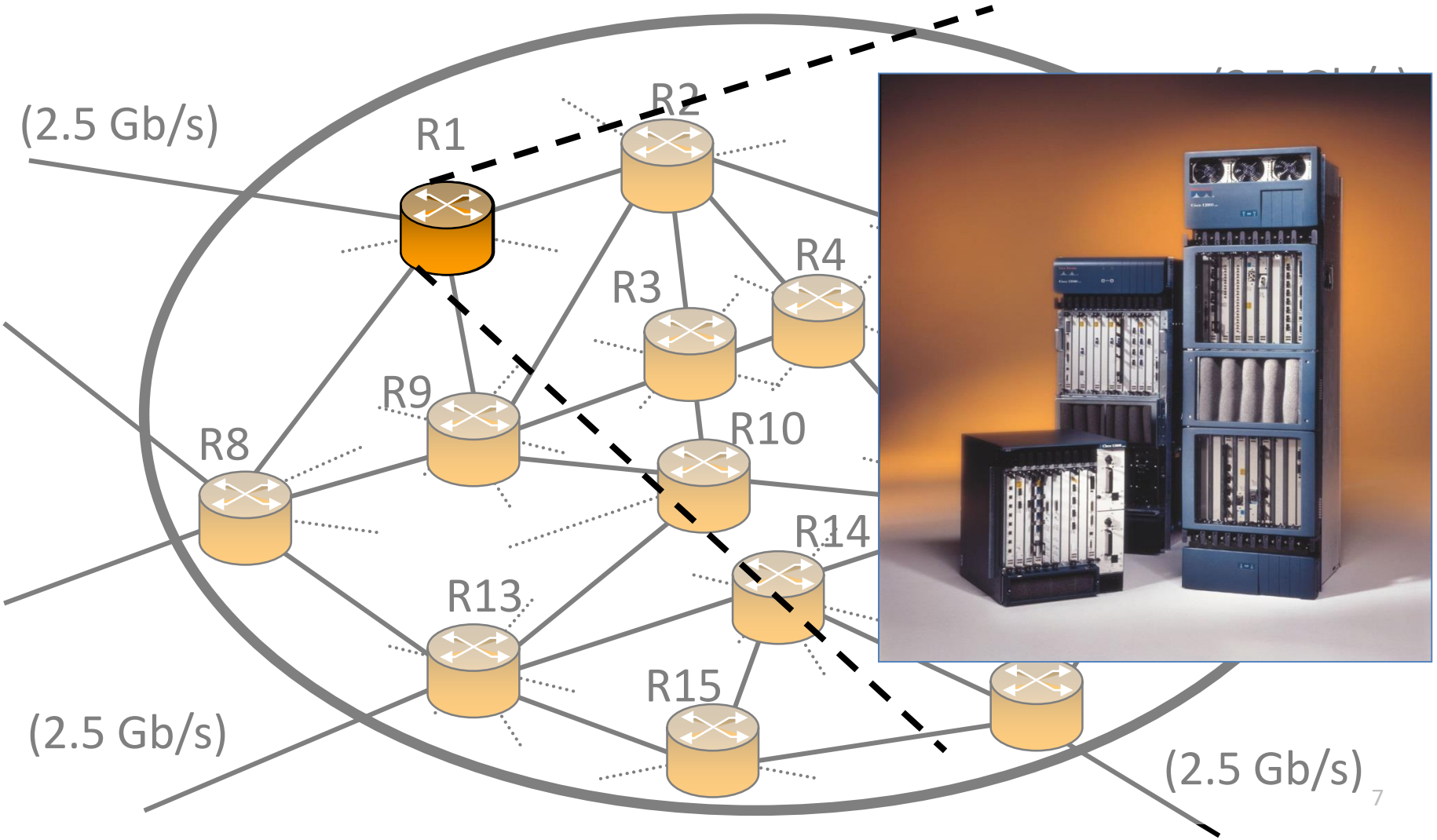
What is Routing?



Points of Presence (POPs)



Where High Performance Routers are Used

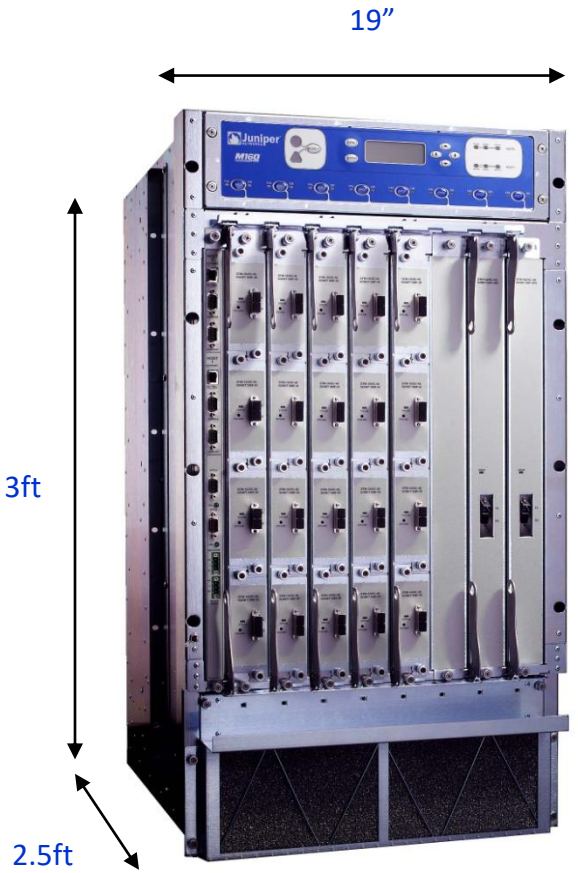


What a Router Looks Like

Cisco GSR 12416

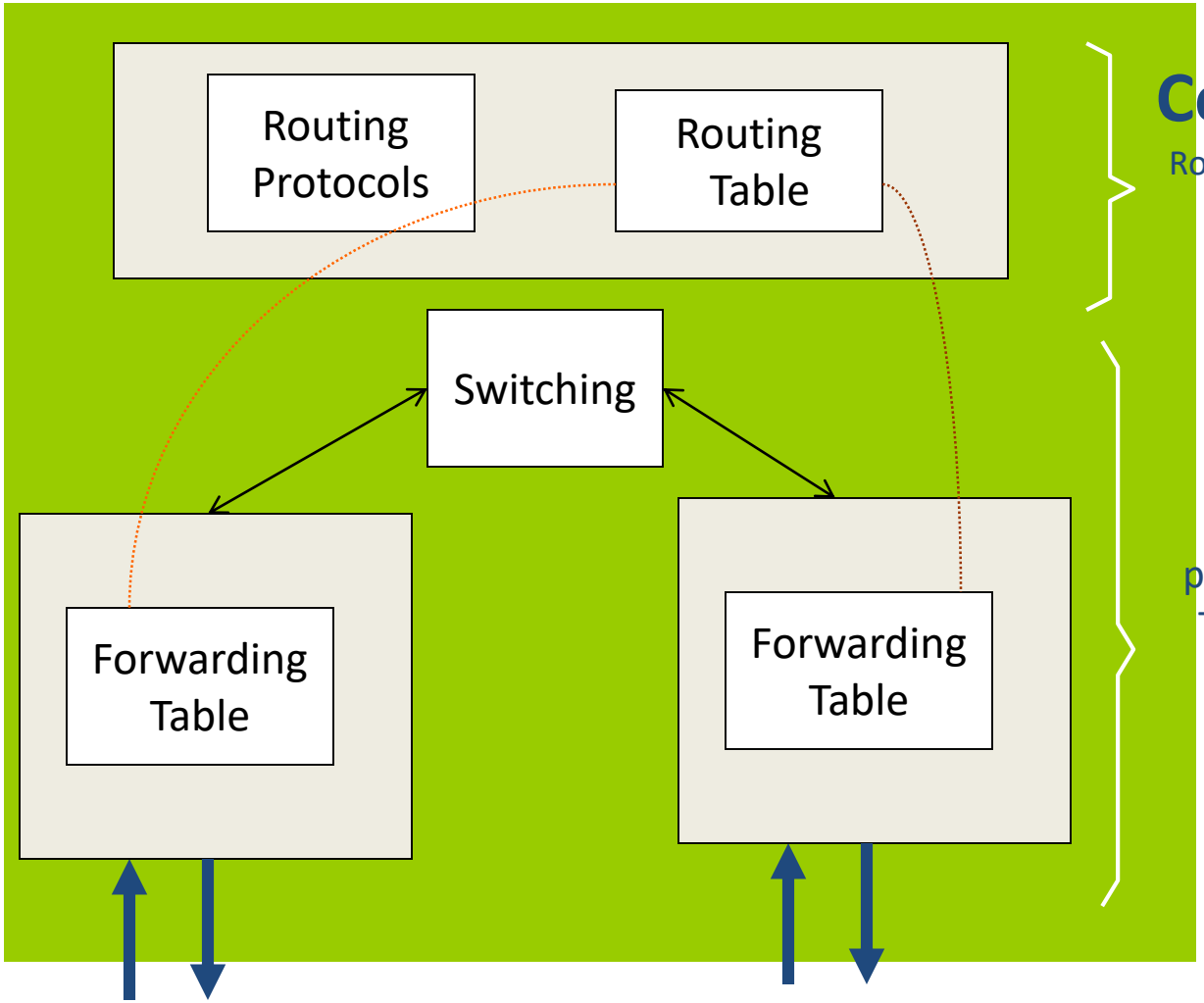
Capacity: 160Gb/s
Power: 4.2kW

Juniper M160



Capacity: 80Gb/s
Power: 2.6kW

Basic Architectural Components of an IP Router



Control Plane

Routing and other control Protocols Management

Datapath

per-packet processing
Traffic Management
Congestion Control
Switching

Types of switching elements

- Telephone switches
  - switch samples (8 bits)
- Datagram routers
  - route datagrams (variable length 64 bytes minimum)
- ATM (Asynchronous Transfer Mode) switches
  - switch ATM cells (constant length packets = 53 bytes = 5 bytes header + 48 bytes payload)
- MPLS switches
  - switch MPLS packets (variable length)
  - modified routers or ATM switches

What's the difference between routing and switching??

Routing and Switching

- Routing
  - Packet forwarding based on routing tables (established through routing protocols)
  - Longest Prefix Match lookup
  - datagram switching (no circuit setup)

- Switching
  - Pre-establish a circuit (physical or virtual) for communication
  - Packet forwarding is based on cross-connect tables (established through call setup procedures)
  - Uses physical or logical (virtual) circuit identifier (VCI)

Equipment Characteristics

- Switching Fabric Capacity
  - e.g., 1Gb, 10Gb, 320Gb, 5Tb
- Number of Interfaces (or ports)
  - 2, 4, 8, 16, 32, 64, 128
- Types of Interfaces (or ports)
  - Ethernet, T1, DS3, OC3, OC48, OC192
- Redundancy
  - Fabric, Port and Power Supply redundancy
- Control Plane (in-band or out-of-band)
  - Protocols supported
  - Management (Command Line Interface CLI, Web based, SNMP)

Classification

- Packet vs. Circuit switches
  - packets have headers (self-routing info) and samples don't
- Connectionless vs. connection oriented
  - connection oriented switches need a call setup
  - setup is handled in *control plane* by *switch controller* using *signaling protocols*
  - connectionless switches deal with *self-contained* datagrams

	Connectionless (router)	*Connection-oriented (switching system)*
Packet switch	Internet router	ATM switching system MPLS Switch
Circuit switch		Telephone switching system

Other switching element functions

- Participate in routing algorithms
  - to build routing tables
- Resolve contention for output trunks
  - scheduling
- Admission control
  - to guarantee resources to certain streams
- Here we focus on pure data movement (data path)

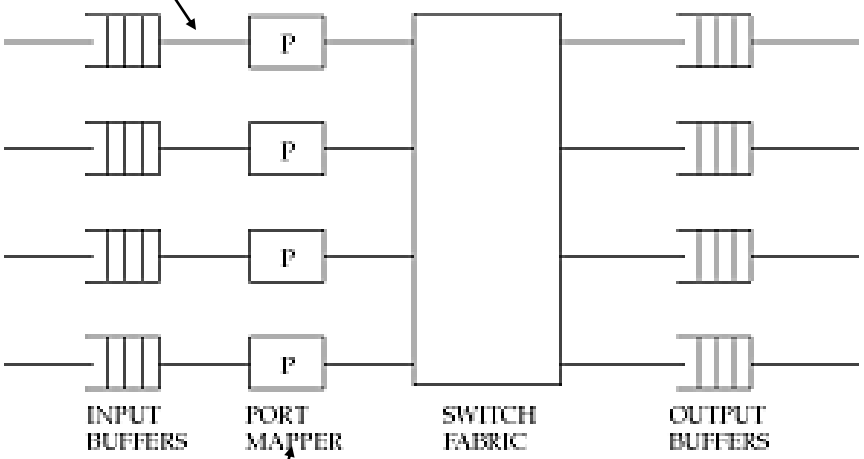
Requirements

- Capacity of switch is the maximum rate at which it can move information, *assuming all data paths are simultaneously active (e.g, 32 ports each at 10G=320G)*
- Primary goal: maximize capacity
  - subject to cost and reliability constraints
- Circuit switch must reject calls if it can't find a path for samples from input to output
  - goal: minimize call blocking
- Packet switch must reject a packet if it can't find a buffer to store it awaiting access to output trunk
  - goal: minimize packet loss
- Don't reorder packets (why??)

A generic switch

internal links can be serial links or buses

routes the packet to proper output



Ingress

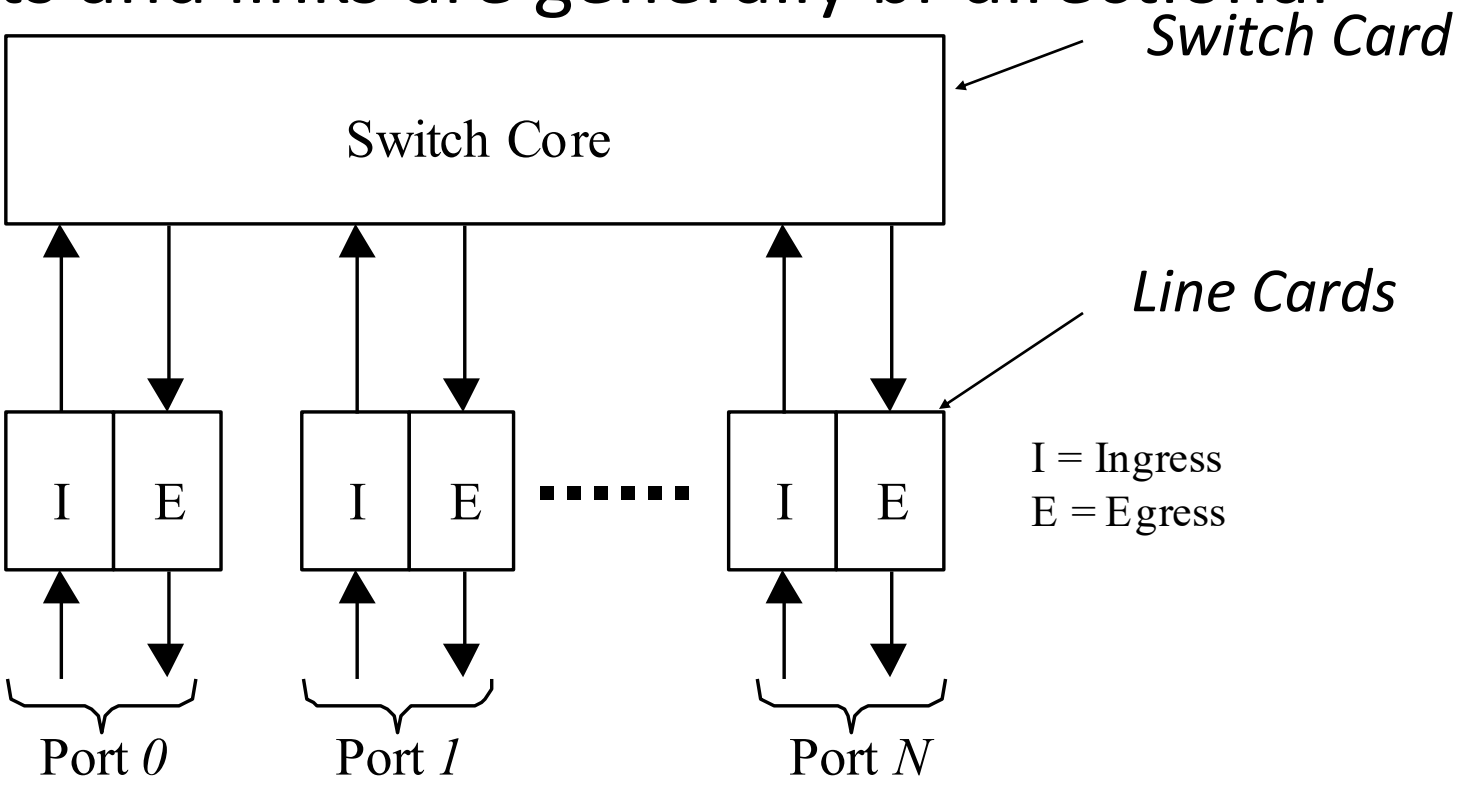
Egress

read header for destination or VCI, index to forwarding table for output port (used only in packet switches)

Ingress, Egress Linecards will host Framing, Traffic Management functions; Not all switches may have all components

Generic Switch – Folded Diagram

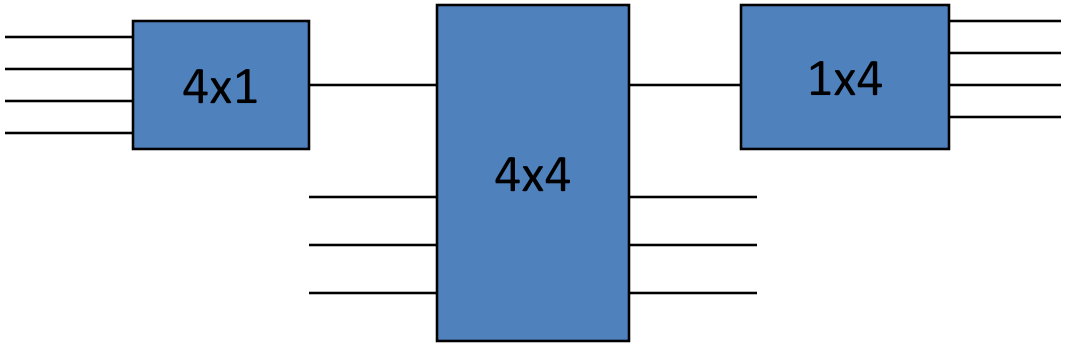
- Ports and links are generally bi-directional



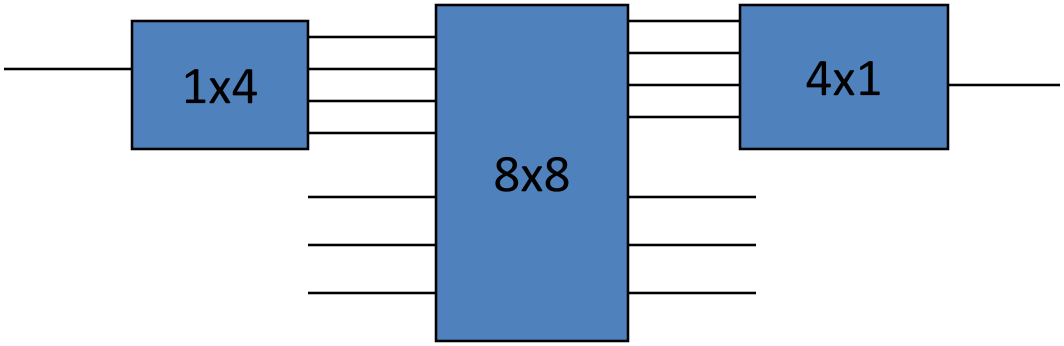
Outline

- Packet switching
  - Switch generations
  - Switch fabrics
  - Buffer placement
  - Multicast switches

Line Heterogeneity



Low speed to High speed



High Speed to Low Speed

Most switches assume all trunks are same speed. If not muiltiplexors and de-multiplexors can be used to match speeds

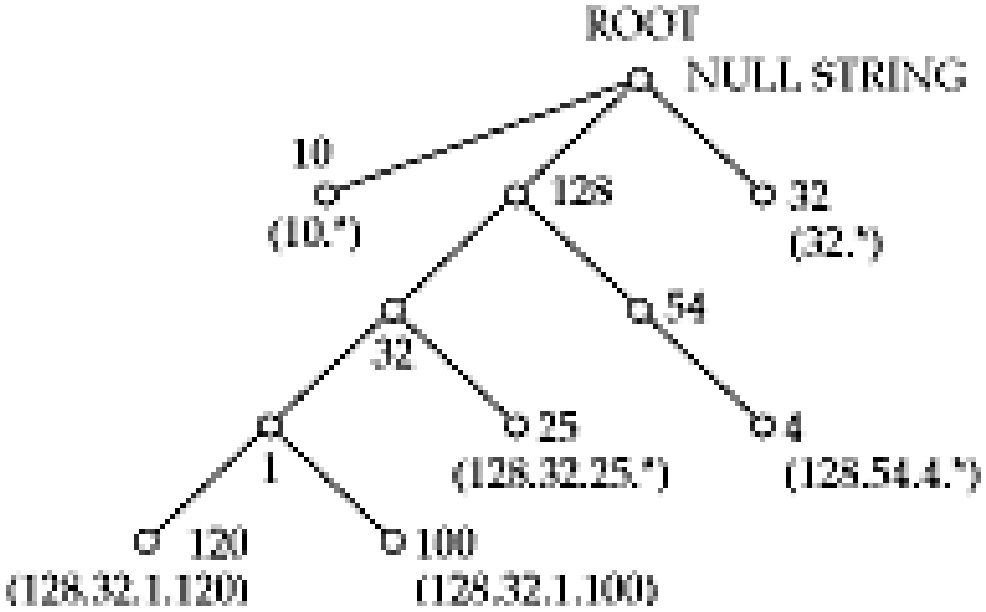
Packet switching

- In a circuit switch, path of a sample is determined at time of connection establishment
- No need for a sample header--position in frame is enough
- In a packet switch, packets carry a destination field
- Need to look up destination port on-the-fly
- Datagram
  - lookup based on entire destination address
- ATM Cell
  - lookup based on VCI
- MPLS Packet
  - Lookup based on label in the packet
- Other than that, very similar

Port mappers

- Look up output port based on destination address
- Easy for VCI: just use a table (Cross Connect)
- Harder for datagrams:
  - need to find *longest prefix match*
    - e.g. packet with address 128.32.1.20
    - entries: (128.32.*, 3), (128.32.1.*, 4), (128.32.1.20, 2)
- A standard solution: trie
  - A tree in which each node corresponds to a string that is defined by the path to that node from the root
  - *Alphabet* is a finite set of elements used to form address strings and contains natural numbers [0,1,2,…,255]
  - Children of each node correspond to every element of the alphabet

Tries

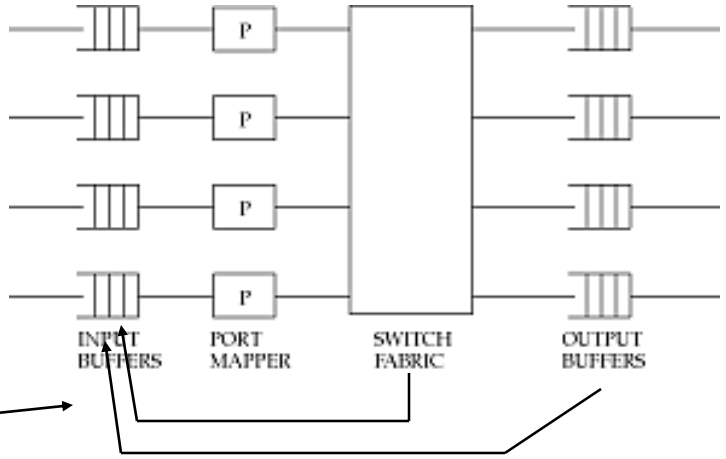


- Two ways to improve performance
  - cache recently used addresses (principle of locality) in a CAM
  - move common entries up to a higher level (match longer strings)

Blocking in packet switches

- Can have both internal and output blocking
- Internal
  - no path to output
- Output
  - trunk unavailable
- Unlike a circuit switch, cannot predict if packets will block (why?)
- If packet is blocked, must either buffer or drop it

Dealing with blocking



- Over-provisioning
  - internal links much faster than inputs
  - expensive, waste of resources
- Buffers
  - at input or output
- Backpressure
  - if switch fabric doesn't have buffers, prevent packet from entering until path is available, by sending signals from output to input quickly.
- Sorting and Randomization
  - For certain fabrics, sorting or randomization reduces internal blocking
- Parallel switch fabrics
  - increases effective switching capacity

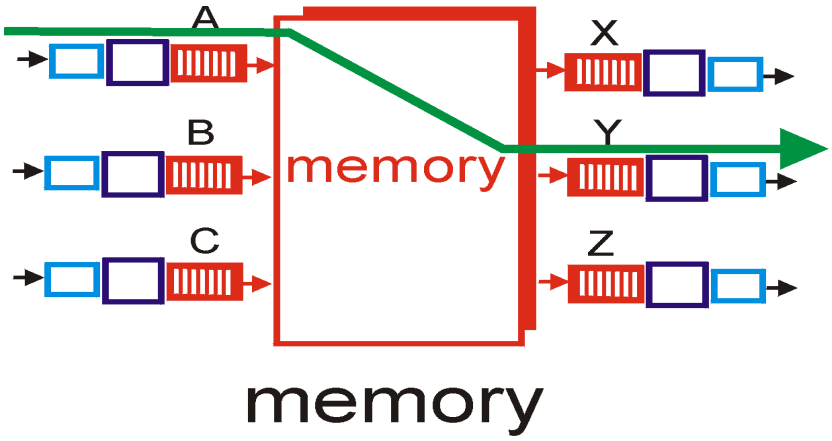
Outline

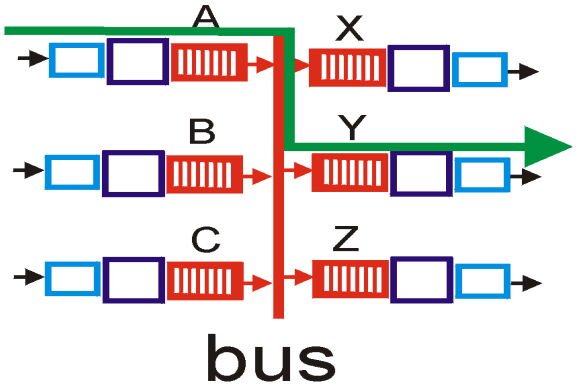
- Circuit switching
- Packet switching
  - Switch generations
  - Switch fabrics
  - Buffer placement
  - Multicast switches

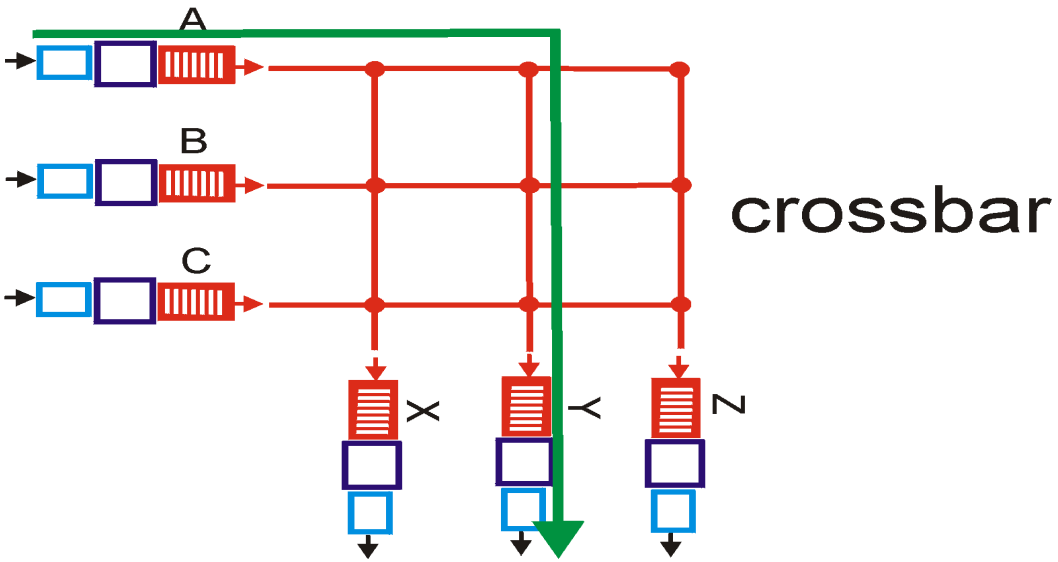
Three generations of packet switches

- Different trade-offs between cost and performance
- Represent evolution in switching capacity
- All three generations are represented in current products
  - Cisco, Juniper, Nortel, Alcatel and many others

Three types of switching fabrics



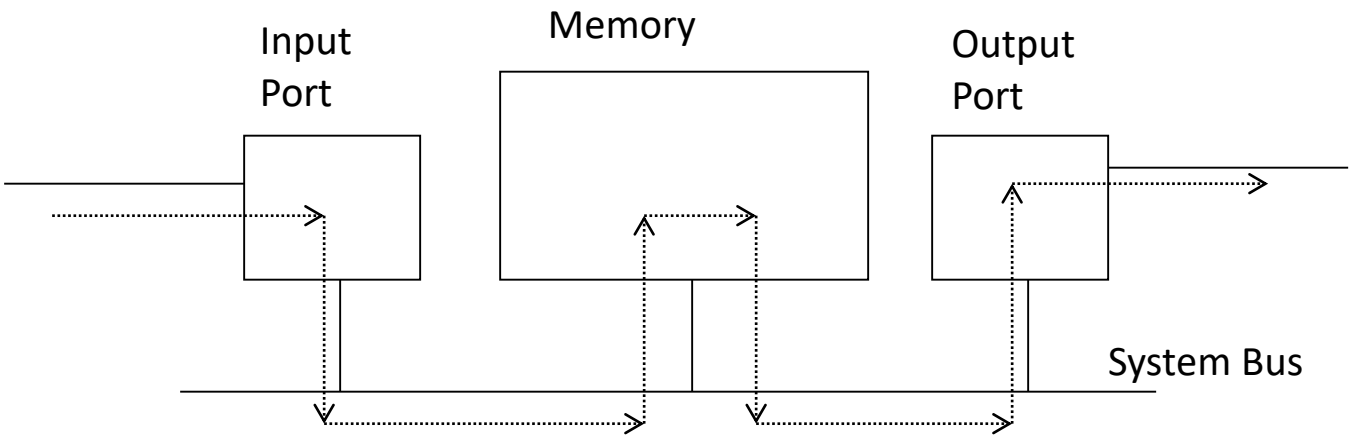




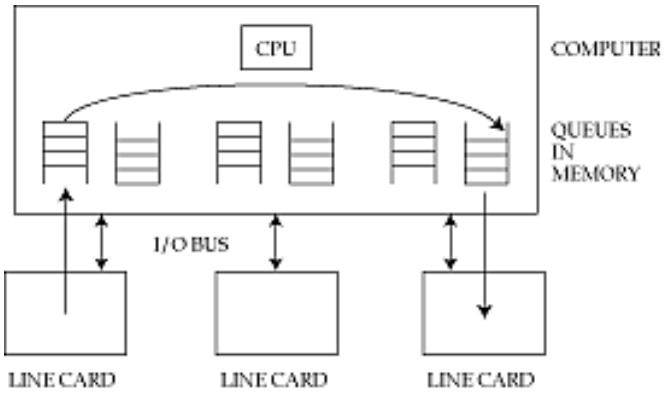
Switching Via Memory

First generation routers:

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

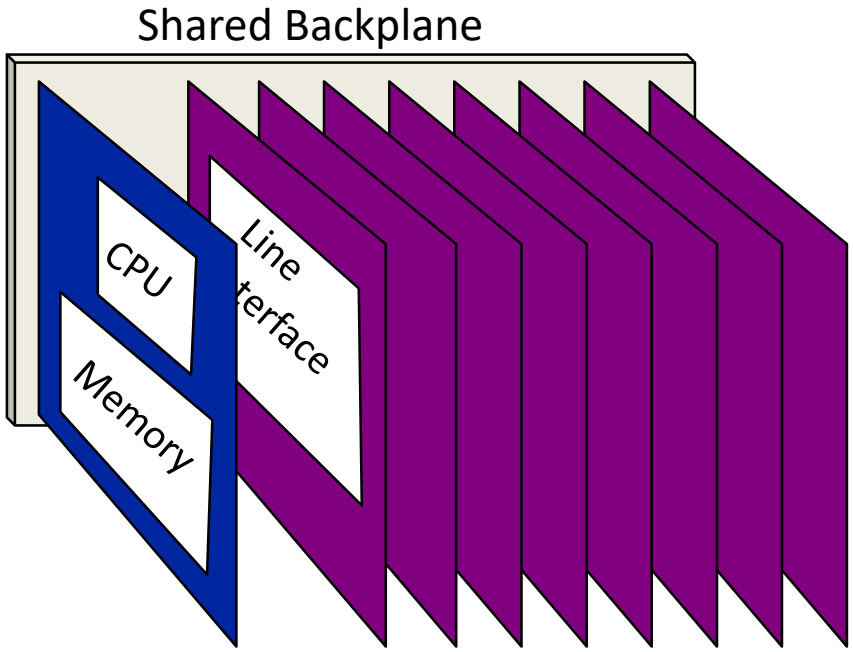


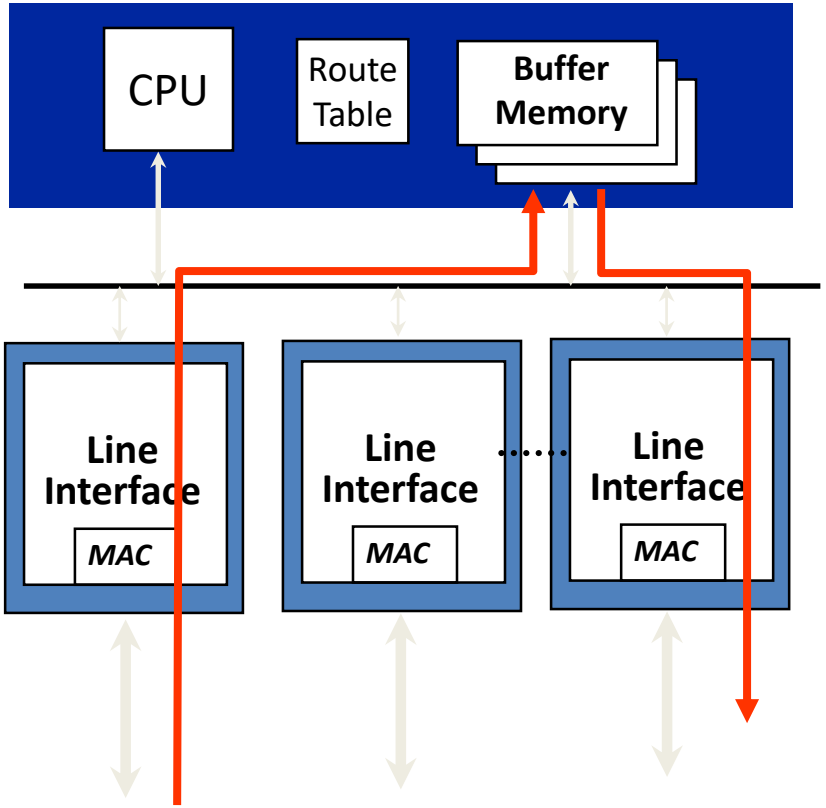
First generation switch



- A computer with multiple line cards
- Processor periodically polls inputs (or is interrupted)
- Most Ethernet switches and cheap packet routers
- Bottleneck can be CPU, host-adaptor or I/O bus, depending on the traffic scenario
- Line card can be cheap (no CPUs on line cards!!)

First Generation Routers





Typically <0.5Gb/s aggregate capacity

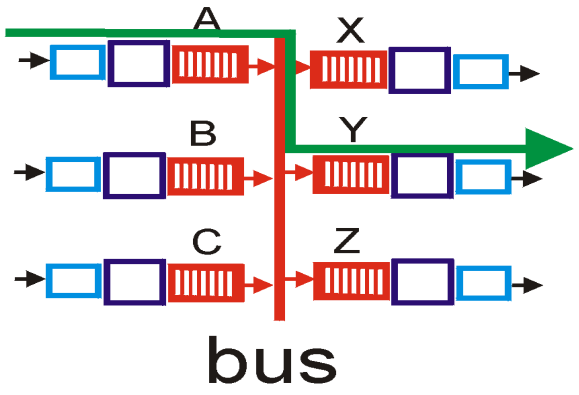
Example

- First generation router built with 133 MHz Pentium
  - Assume instruction takes one clock cycle (=7.52 ns)
  - Mean packet size 500 bytes
  - Interrupt takes 10 microseconds, word (4 bytes) access takes 50 ns
  - Per-packet processing time (routing table lookup and others) takes 200 instructions = 1.504 µs
- Copy loop (one word copy)

```
register <- memory[read_ptr]
memory [write_ptr] <- register
read_ptr <- read_ptr + 4
write_ptr <- write_ptr + 4
counter <- counter -1
if (counter not 0) branch to top of loop
```

- 4 instructions + 2 memory accesses = 130.08 ns
- Copying packet takes 500/4 *130.08 = 16.26 µs; interrupt 10 µs
- Total time = 16.26+10+1.504=27.764 µs => speed is 144.1 Mbps
- How many Ethernet ports (10Mbps, 100Mbps) can be support??
  - Linux, Windows can do this now!!

Switching Via a Bus

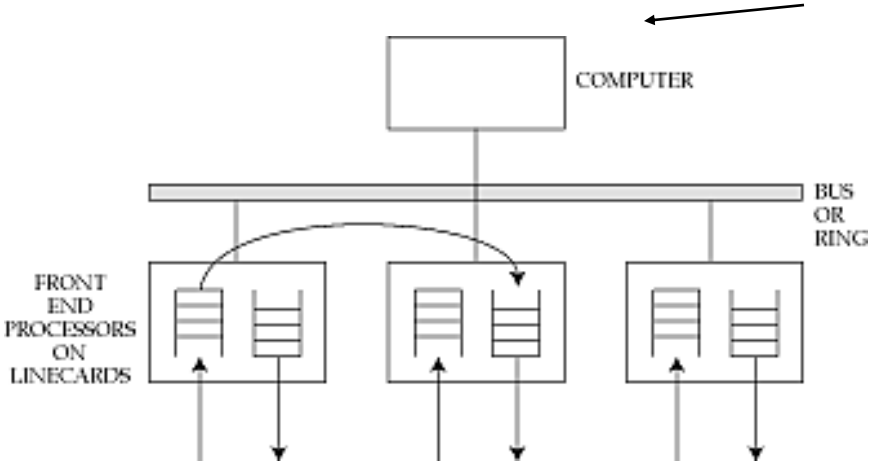


- datagram from input port memory

  to output port memory via a shared bus
- bus contention: switching speed limited by bus bandwidth
- 1 Gbps bus, Cisco 1900: sufficient speed for access and enterprise routers (not regional or backbone)

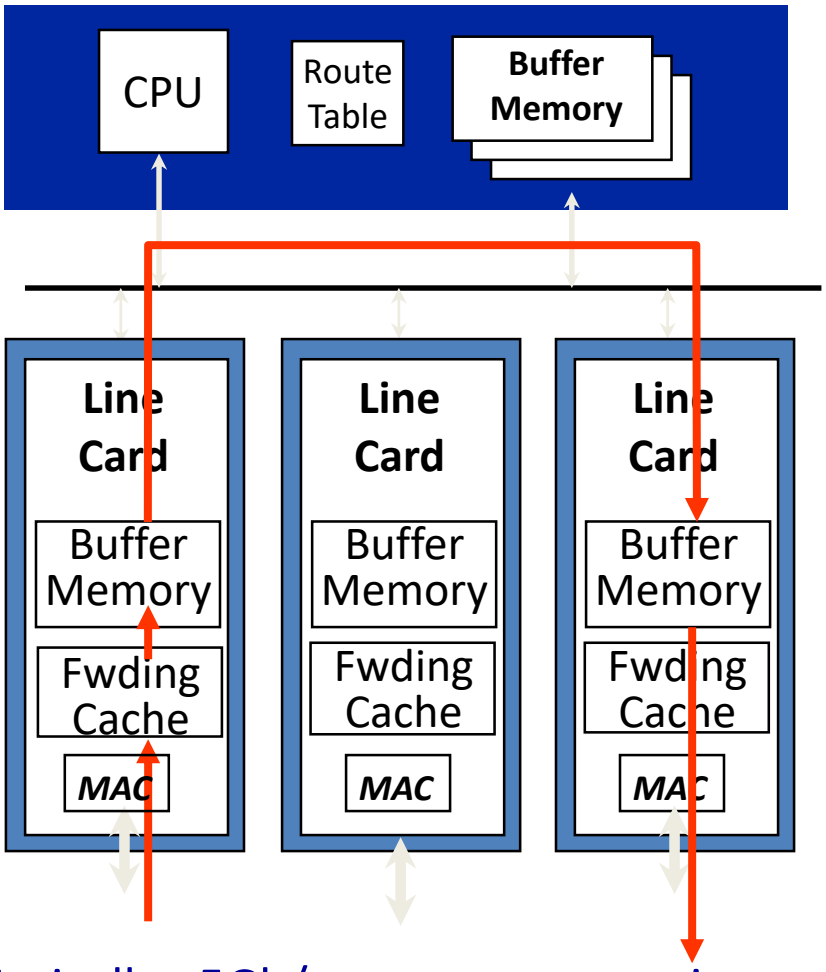
Second generation switch

Control plane
Routing Protocols
Routing Tables



- Port mapping intelligence in line cards (processor based)
- Ring or Bus based backplane to connect line cards
  - Bottleneck -> performance impact (discuss bus and ring)
- Lookup cache on line cards (for better performance)
- For switch, cross connect table (port mapping entries) only changed when calls are setup / torn down
- For datagram router, ask the control processor if the entry is not found in local forwarding table or automatically updated.

Second Generation Routers



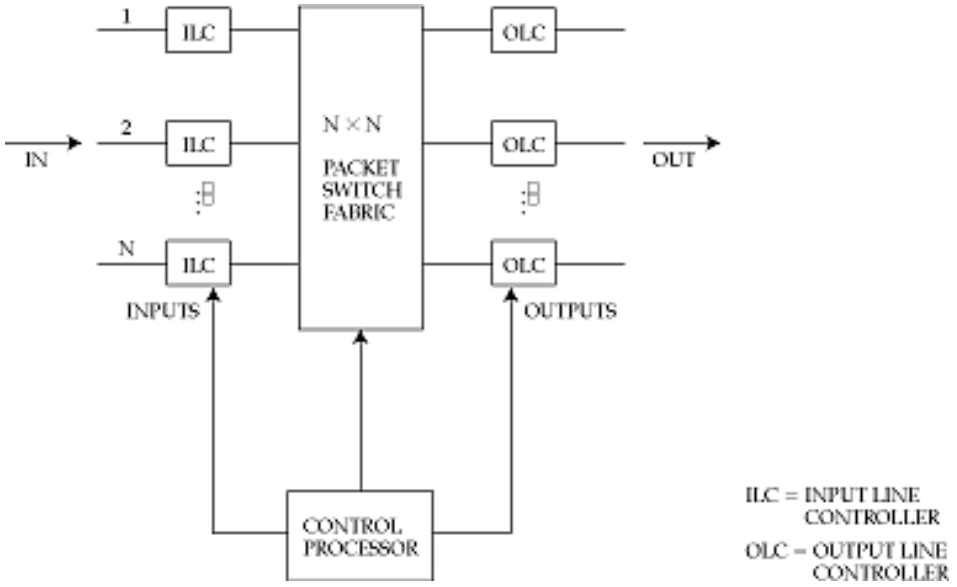
Typically <5Gb/s aggregate capacity

Switching Via An Interconnection Network

- overcome bus bandwidth limitations
- Banyan networks, other interconnection nets initially developed to connect processors in multiprocessor
- Advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
  - Segmentation and Reassembly (SAR)
    - Discuss overheads
- Cisco 12000: switches Gbps through the interconnection network

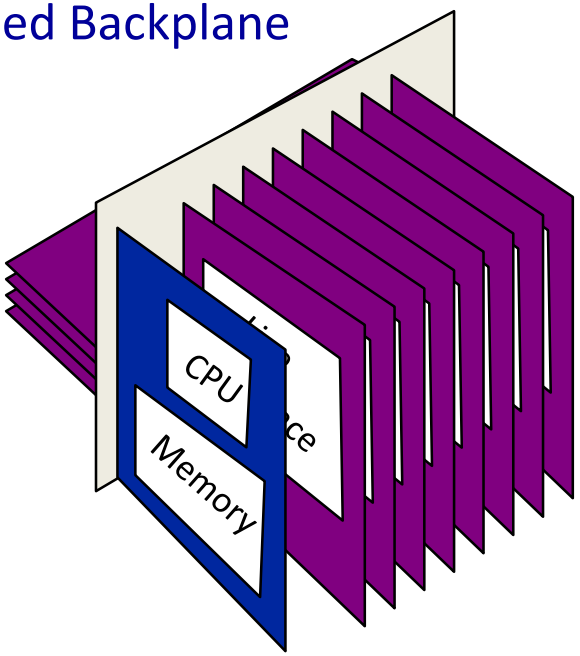
Third generation switches

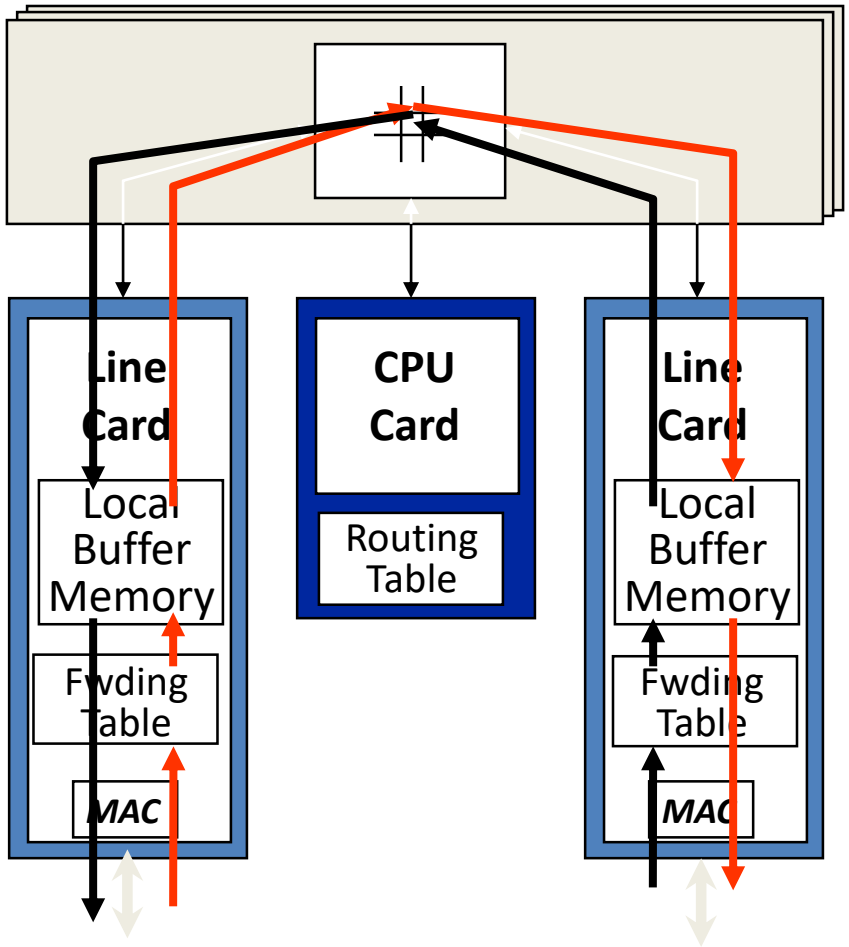
- Bottleneck in second generation switch is the bus (or ring)
- Third generation switch provides parallel paths using a switch fabric



Third Generation Routers

Switched Backplane



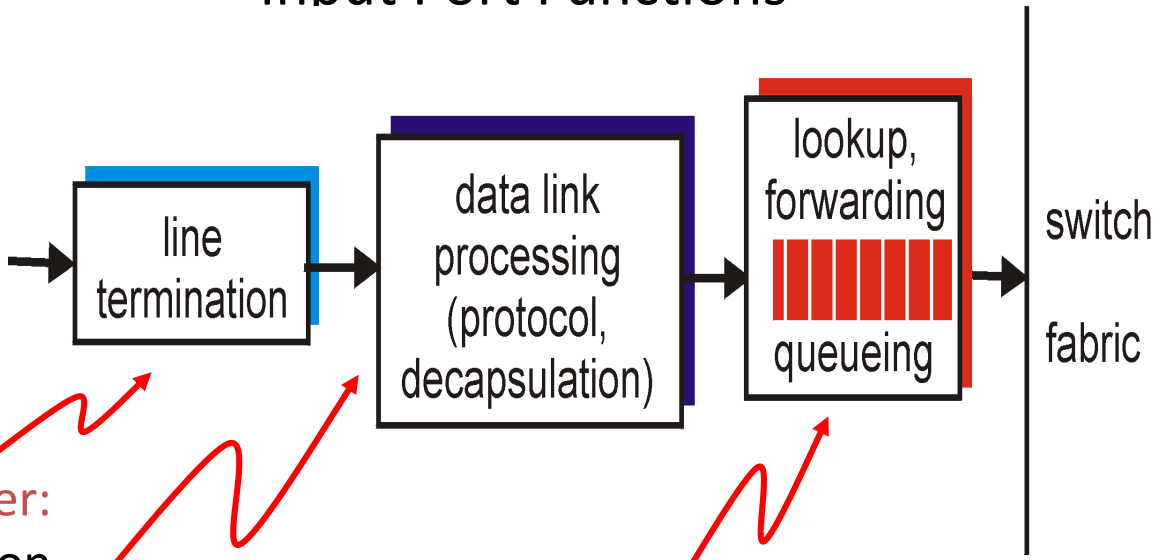


Typically <50Gb/s aggregate capacity

Third generation (contd.)

- Features
  - self-routing fabric
  - output buffer is a point of contention
    - unless we *arbitrate* access to fabric
  - potential for unlimited scaling, as long as we can resolve contention for output buffer

Input Port Functions



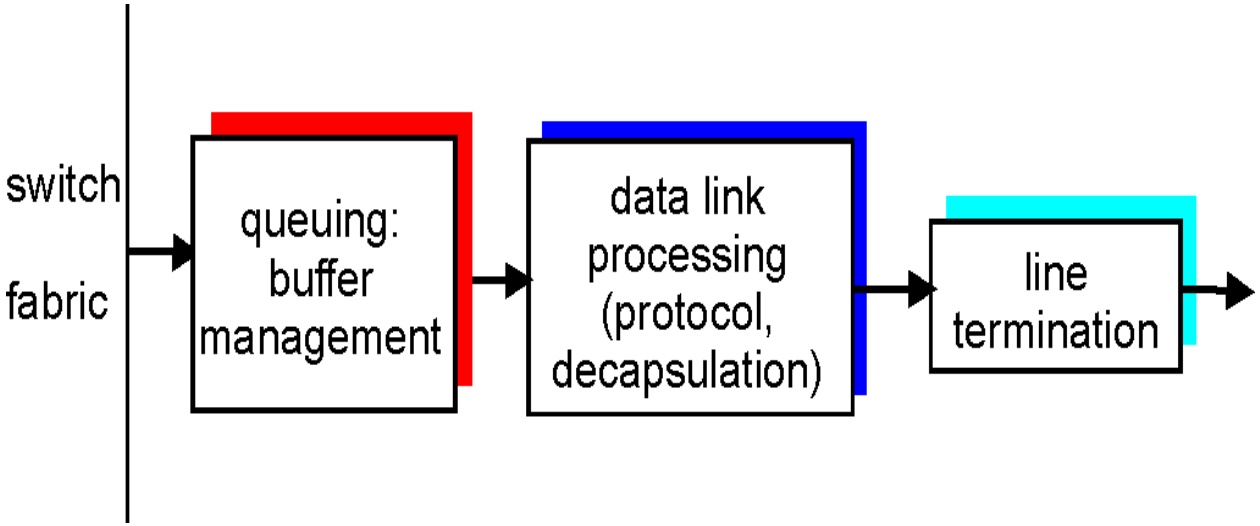
Physical layer:
bit-level reception

Data link layer:
e.g., Ethernet
see chapter 5

Decentralized switching:

- given datagram dest., lookup output port using forwarding table in input port memory
- goal: complete input port processing at 'line speed'
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

Output Ports



- *Buffering* required when datagrams arrive from fabric faster than the transmission rate
- *Scheduling discipline* chooses among queued datagrams for transmission

Outline

- Circuit switching
- Packet switching
  - Switch generations
  - Switch fabrics
  - Buffer placement
  - Multicast switches

Switch fabrics

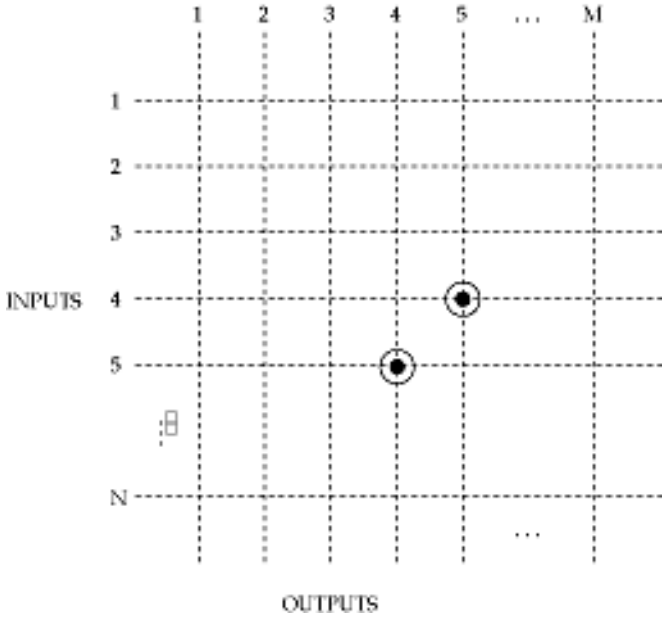
- Transfer data from input to output, ignoring scheduling and buffering
- Usually consist of links and *switching elements*

Crossbar

- Simplest switch fabric
  - think of it as 2N buses in parallel
- Used here for *packet* routing: cross-point is left open long enough to transfer a packet from an input to an output
- For fixed-size packets and known arrival pattern, can compute schedule in advance (e.g., circuit switching)
- Otherwise, need to compute a schedule on-the-fly (what does the schedule depend on?)

Crossbar

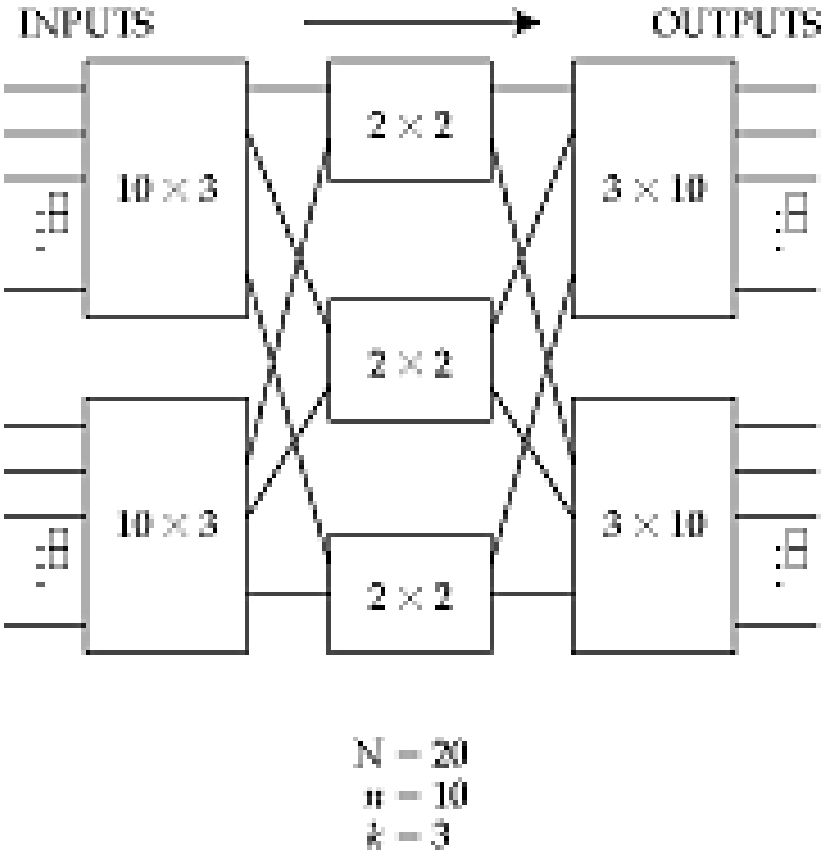
- Simplest possible space-division switch. *NxM* crossbar has *N* inputs and *M* outputs
- *Crosspoints* can be turned on or off (think of a design)
- Need a switching *schedule* (why and what frequency??)
- Internally non-blocking (why?)
  - but needs N^2 cross-points for NxN switch
  - time taken to set each cross-point grows with N
  - vulnerable to single faults (why?)
- What about output blocking?



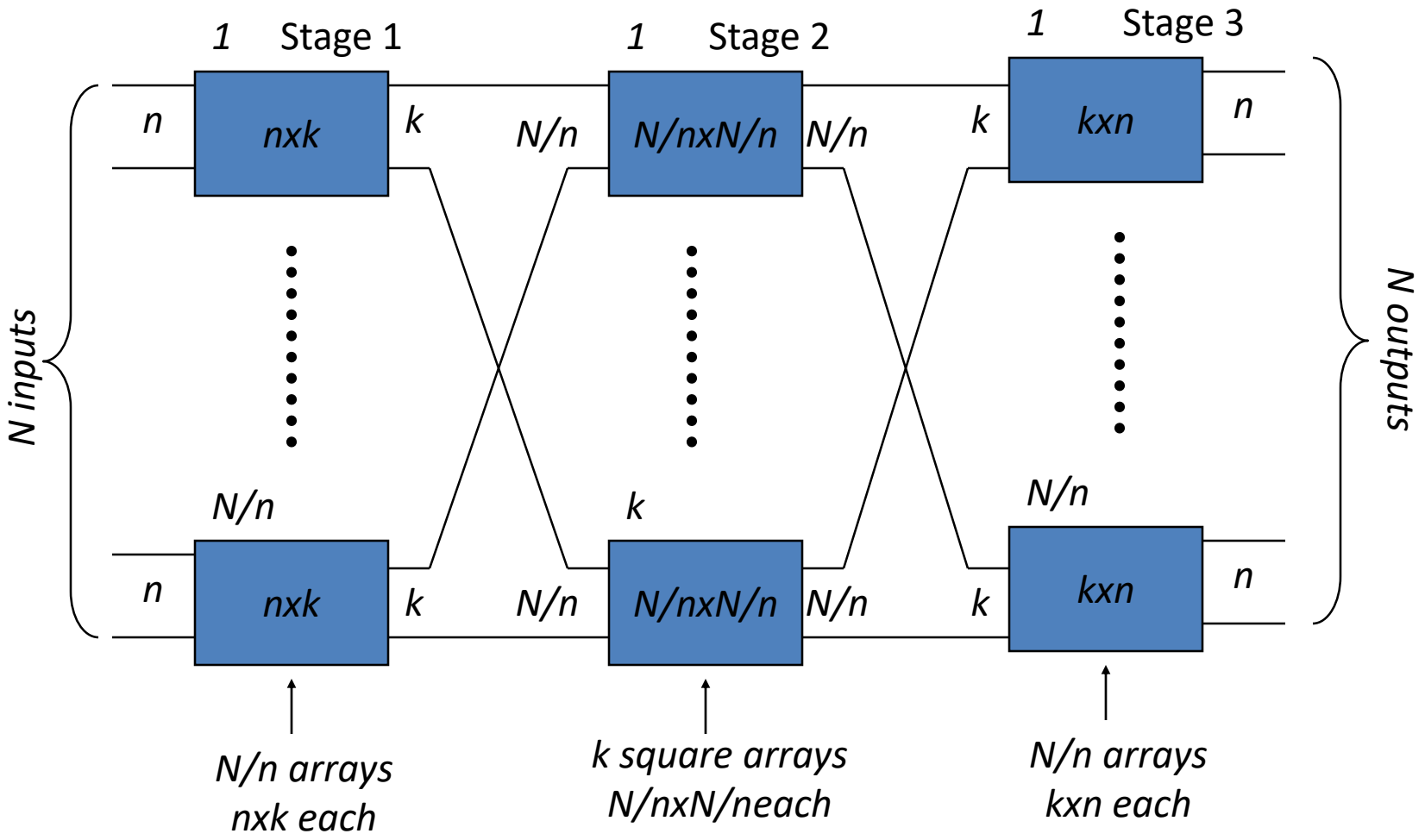
Switch Schedule determines which crosspoint is turned on and when.

Multistage crossbar

- In a crossbar during each switching time only one crosspoint per row or column is active
- Can save crosspoints if a crosspoint switch can attach to more than one input line (why?)
- This is done in a multistage crossbar
- Inputs are broken into groups (e.g, 20 lines, 2 groups of 10 lines each)
- Multiple paths between inputs and output group share a centre stage switch
- Need to rearrange connections every switching time (switching schedule)



Multistage Switching



Total Number of Cross Points = $2Nk+k(N/k)^2$

Multistage crossbar

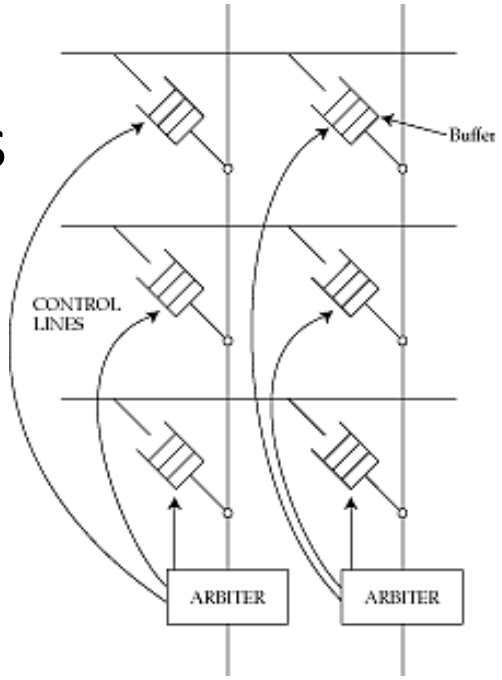
- First stage consists of *N/n* arrays of size *nxk each*
- Second stage consists of *k* arrays of size *N/n x N/n each*
- Third stage consists of *N/n* arrays of size *kxn each*
- *Can suffer internal blocking*
  - *unless sufficient number of second-level stages*
- Number of crosspoints < N^2
- Finding a path from input to output
  - switch controller needs to find a path at the time of call setup
  - uses path search algorithms, such as depth-first-search
  - the path is then stored in the switch schedule
- Scales better than crossbar, but still not too well
  - 120,000 call switch needs ~250 million crosspoints

Clos Network

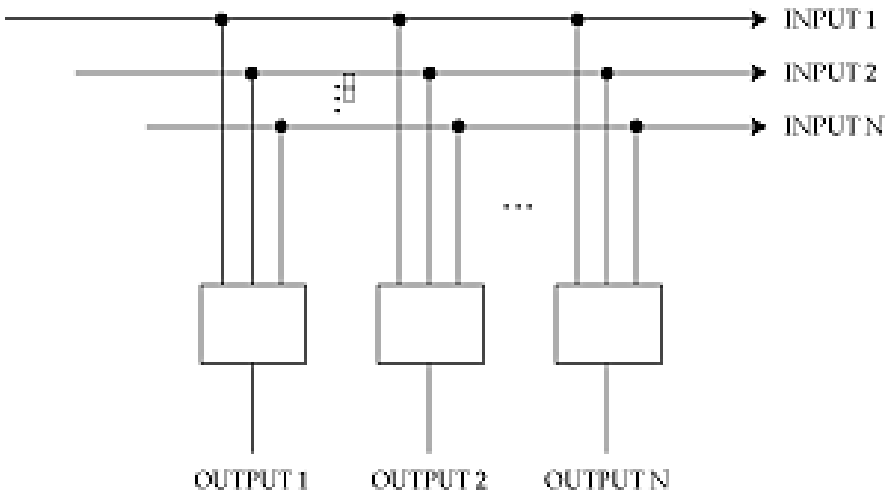
- How large should be *k* (# of center stages) for the switch to be internally non-blocking??
  - Clos [1953 paper] showed that if a switch controller is willing to *rearrange* existing connections when a new call is arrived, the condition is
    - *k≥n* (i.e., the number of center stages must be greater than the number of inputs in a group) (*k=2n-1)*
    - Also called *re-arrangably non-blocking switch*
    - In practice we cannot rearrange live calls (without breaking the circuit) - becomes complex (make before break)
  - Clos network of size *NxN* has $2N(2n-1)+(2n-1)x(N/n)^2$ cross points, way smaller than N^2
  - Optimal $n=O(\sqrt{N})$

Buffered crossbar

- What happens if packets at two inputs both want to go to same output?
  - Output blocking
- Can defer one at an input buffer
- Or, buffer cross-points
- How large is the buffer s
- Overflow in the switch
  - Can we afford?
  - Solutions?
    - Backpressure



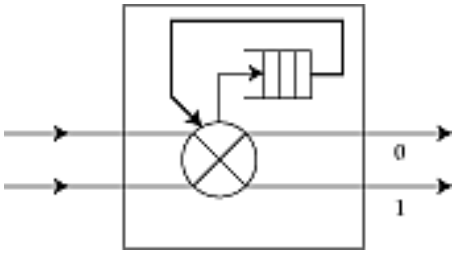
Broadcast



- Packets are tagged with output port #
- Each output matches tags
- Need to match N addresses in parallel at each output
- Useful only for small switches, or as a stage in a large switch

Switch fabric element

- Can build complicated fabrics from a simple element consisting of two inputs, two outputs and an optional buffer
- Packets arrive simultaneously; Look at the header;



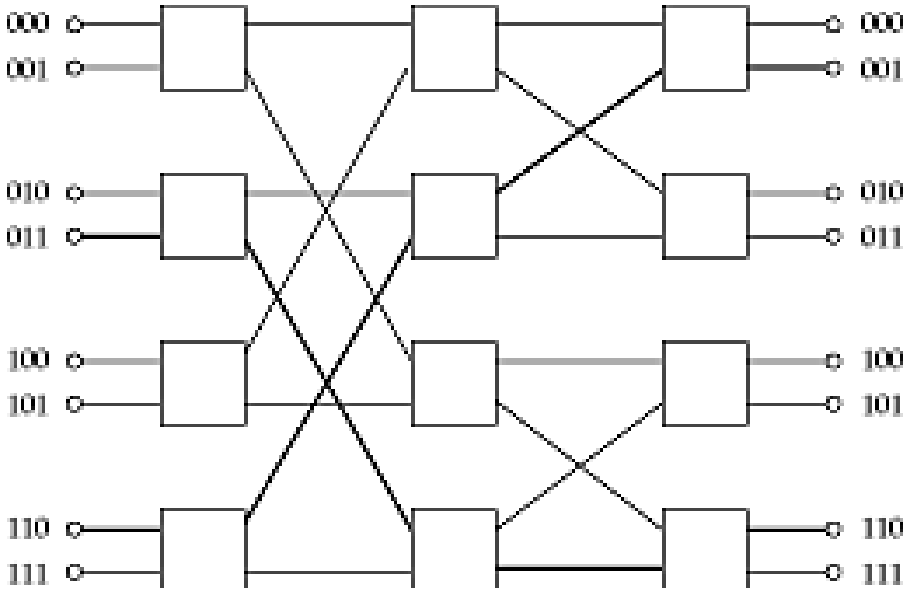
- Routing rule: if 0, send packet to upper output, else to lower output
- If both packets to same output, buffer or drop

Features of fabrics built with switching elements

- NxN switch with bxb elements has $\lceil \log_b N \rceil$ stages with $\lceil N / b \rceil$ elements per stage
  - e.g., 8x8 switch with 2x2 elements has 3 stages of 4 elements per stage
  - e.g., 4096x4096 switch built with 8x8 blocks has four stages with 512 elements in each stage
- Fabric is *self routing*
  - Once a packet is labeled to a correct output, it will automatically makes its way
- Recursive
  - composed of smaller components that resemble larger network
- Can be synchronous or asynchronous (permits variable length packets)
- Regular and suitable for VLSI implementation

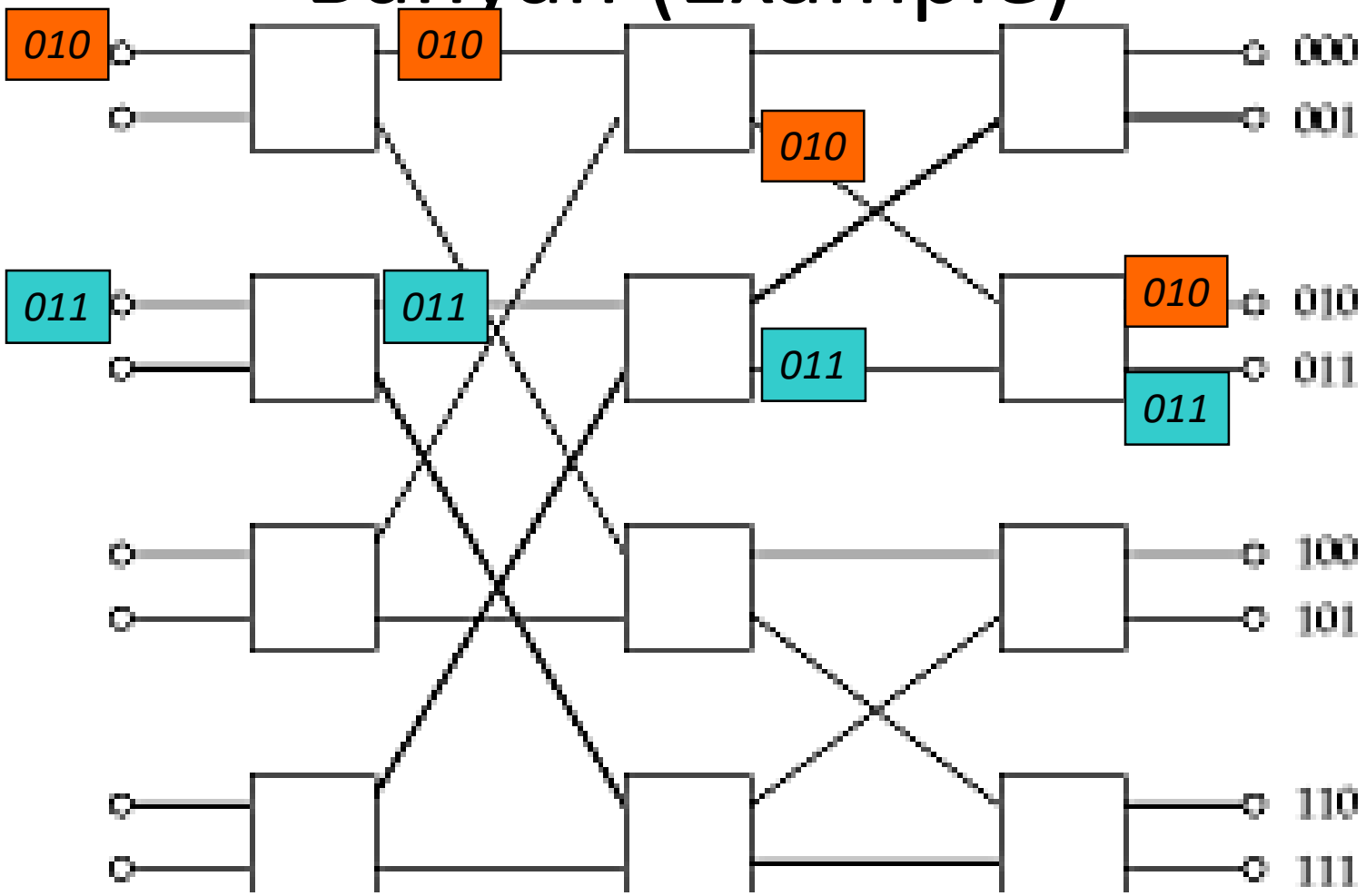
Banyan

- Simplest self-routing recursive fabric. Packets are tagged with output port in binary
- Made of 2x2 switches
- Fabric needs *n* stages for 2^n outputs with 2^{n-1} elements in each stage



- (why does it work?) Each switching element at the i^{th} stage looks at the i^{th} bit to make a forwarding decision
- What if two packets both want to go to the same output?
  - output blocking

Banyan (Example)



Blocking

- Can avoid with a buffered banyan switch
  - but this is too expensive; how much buffer at each element?
  - hard to achieve zero loss even with buffers
- Instead, can check if path is available before sending packet
  - three-phase scheme
  - send requests
  - inform winners
  - send packets
- Or, use several banyan fabrics in parallel
  - intentionally misroute and tag one of a colliding pair
  - divert tagged packets to a second banyan, and so on to k stages
  - expensive (e.g., 32x32 switch with 9 banyans can achive 10^{-9} loss)
  - can reorder packets
  - output buffers have to run k times faster than input

Sorting

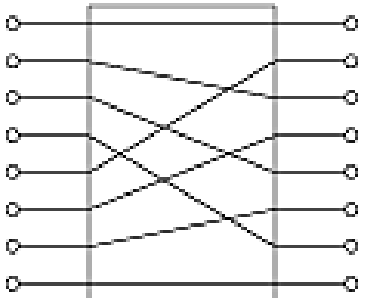
- Or we can avoid blocking by choosing order in which packets appear at input ports
- If we can
  - present packets at inputs sorted by output
  - remove duplicates
  - remove gaps
  - precede banyan with a perfect shuffle stage
  - then no internal blocking
- For example

  [X, 011, 010, X, 011, X, X, X] -(sort)->

  [010, 011, 011, X, X, X, X, X] -(remove dups)-> use a trap network

  [010, 011, X, X, X, X, X, X] -(shuffle)->

  [010, X, 011, X, X, X, X, X] → *This input when presented to Banyan Network is non-blocking*
- Need sort, shuffle, and trap networks



Shuffle Exchange

Sorting

- Build sorters from merge networks
- Assume we can merge two sorted lists to make a larger sorted list
  - Called Batcher Network
  - Needs $\lceil \log N \rceil \lceil \log N+1/2 \rceil$ stages
- Sort pairwise, merge, recurse
- Divide list of *N* elements into pairs and sort each pair (gives *N/2* lists)
- Merge pair wise to form *N/4* and recurse to form *N/8* etc to form one fully sorted list
- All we need is way to sort two elements and a way to merge sorted lists

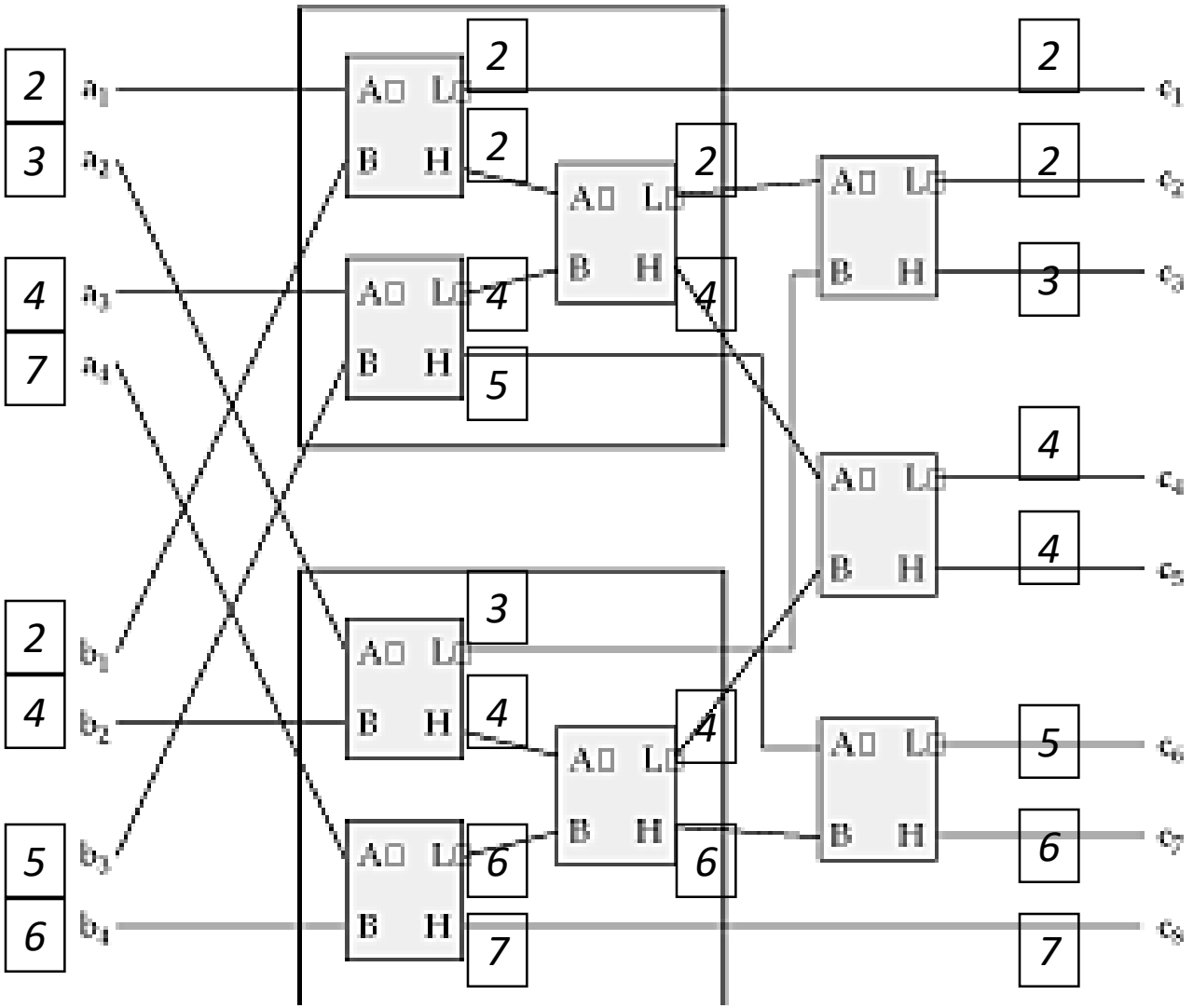
Sorting (Example)

- Sort the list 5,7,2,3,6,2,4,5 by merging
- Solution:
  - Sort elements two-by-two to get four sorted lists {5,7}, {2,3}, {2,6}, {4,5}
  - Second step is to merge adjacent lists to get four element sorted lists {2,3,5,7}, {2,4,5,6}
  - In the third step, we merge two lists to create a fully sorted list {2,2,3,4,5,5,6,7}
- Sorter is easy to build
  - Use a comparator
- Merging needs a separate network

Merging Networks

- A merging network of size 2N takes two sorted lists of size N as inputs and creates a merged list of size 2N
- Consists of two N-sized merging networks
- One of them merges all the even elements of the two inputs and the other merges all the odd elements
- The outputs of the mergers are handed to a set of 2x2 comparators

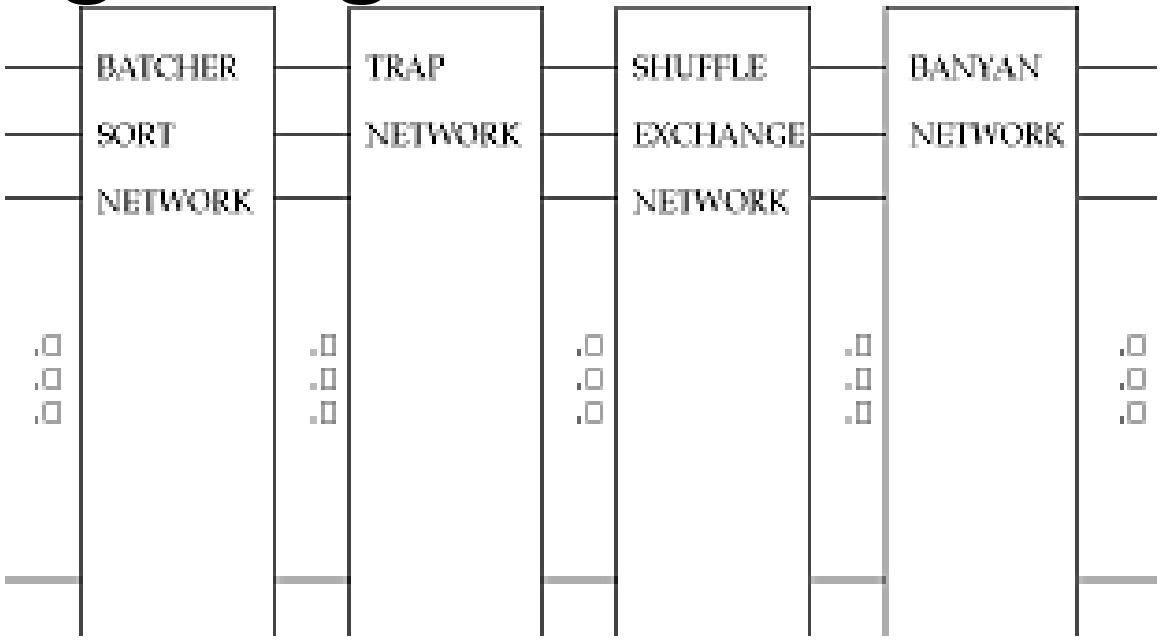
Merging


2
3
4
7
2
4
5
6
a1
a2
a3
a4
b1
b2
b3
b4
A L
B H
2
2
2
4
5
3
4
6
7
2
2
3
4
4
5
6
7
c1
c2
c3
c4
c5
c6
c7
c8

Merging Example

- Merge the two sorted lists {2,3,4,7} and {2,4,5,6}
- Solution:
  - First stage, we merge even elements from the two lists {2,4} with {2,5}
  - Recursing we need to merge {2} with {2} and {4} with {5} then compare them
  - Results of the two merges are {2,2} and {4,5}
  - Comparing higher element of the first list with lower element of the second list, we determine the merged list is {2,2,4,5}
  - Next merge odd elements {3,7} with {4,6} with result {3,4} and {6,7}
  - Comparing the high and low elements we get merged list {3,4,6,7}
  - Carrying out the comparisons we get {2,2,3,4,4,5,6,7}

Putting it together- Batcher Banyan



- What about trapped duplicates?
  - re-circulate to beginning
  - or run output of trap to multiple banyans (*dilation*)

Effect of packet size on switching fabrics

- A major motivation for small fixed packet size in ATM is ease of building large parallel fabrics
- In general, smaller size => more per-packet overhead, but more preemption points/sec
  - At high speeds, overhead dominates!
- Fixed size packets helps build synchronous switch
  - But we could fragment at entry and reassemble at exit
  - Or build an asynchronous fabric
  - Thus, variable size doesn't hurt too much
- Maybe Internet routers can be almost as cost-effective as ATM switches

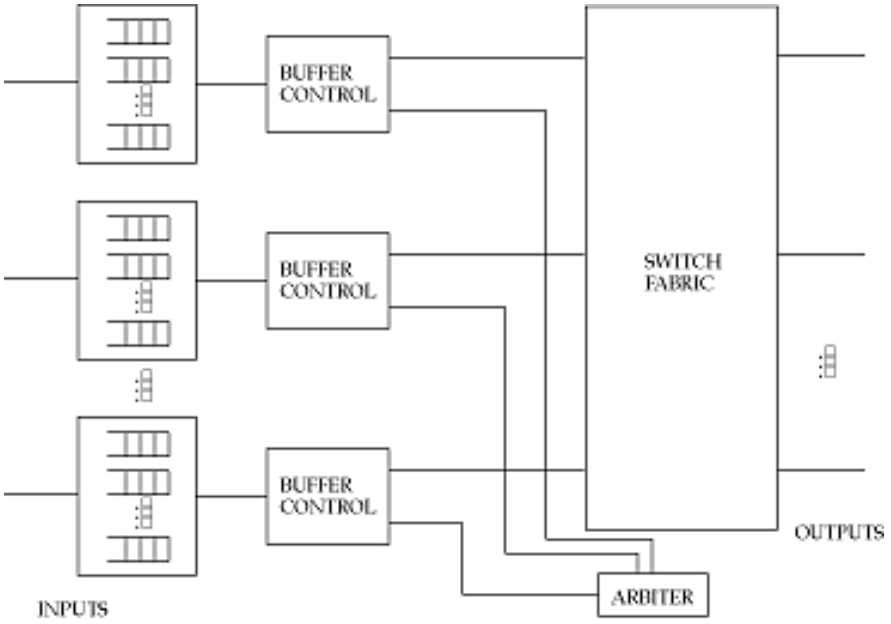
Outline

- Circuit switching
- Packet switching
  - Switch generations
  - Switch fabrics
  - Buffer placement
  - Multicast switches

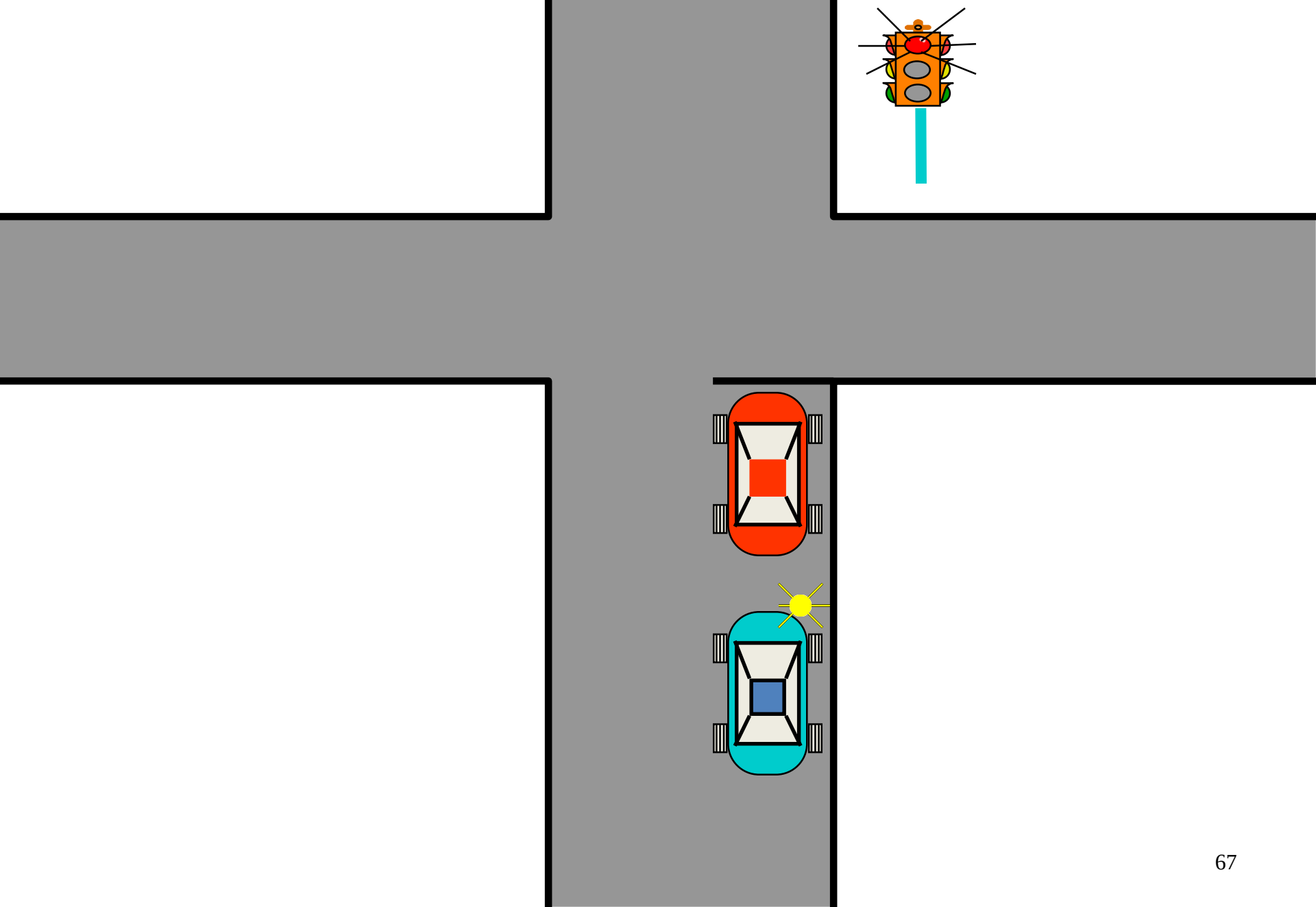
Buffering

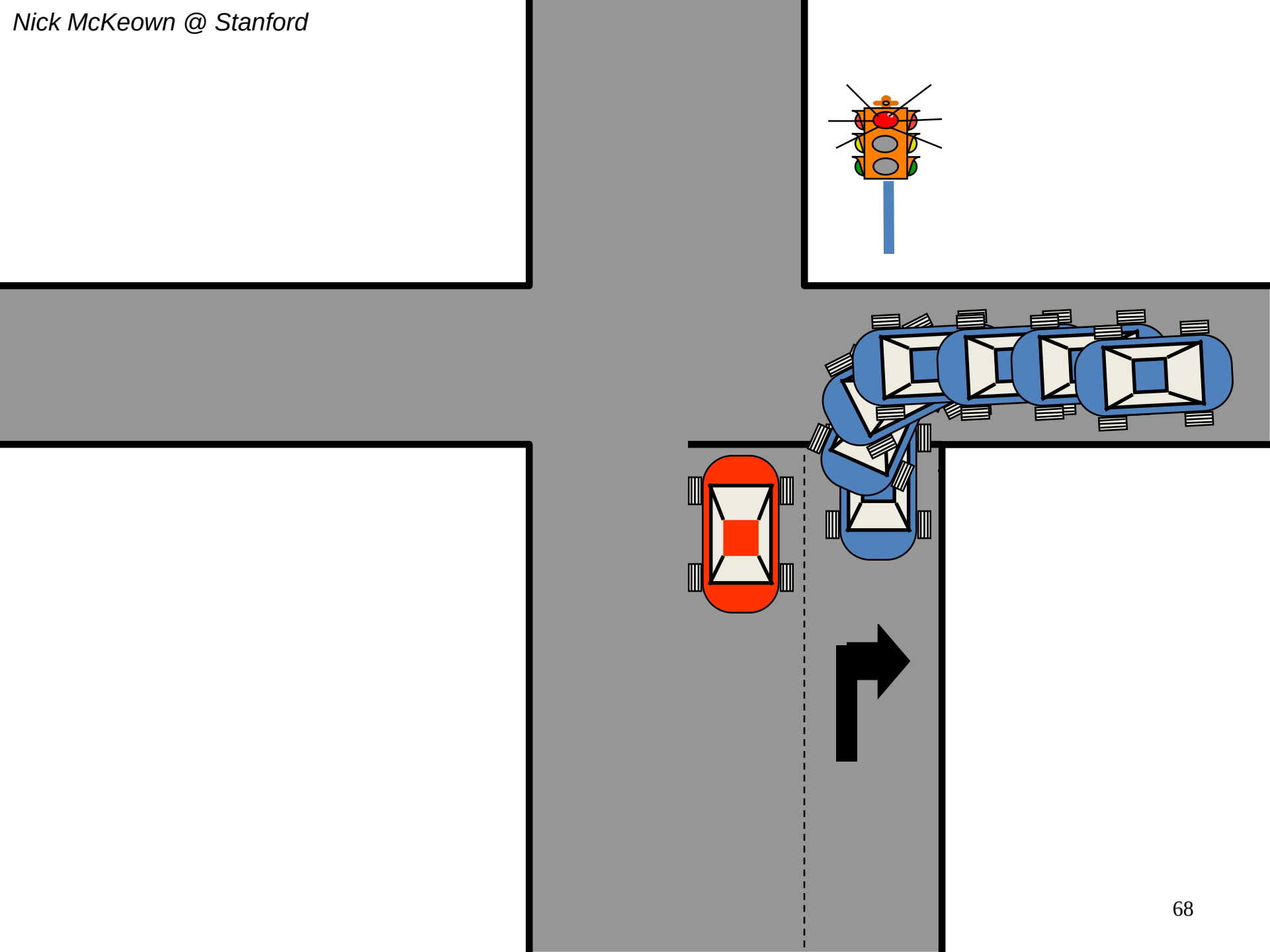
- All packet switches need buffers to match input rate to service rate
  - or cause heavy packet loses
- Where should we place buffers?
  - input
  - in the fabric
  - output
  - shared

Input buffering (input queueing)

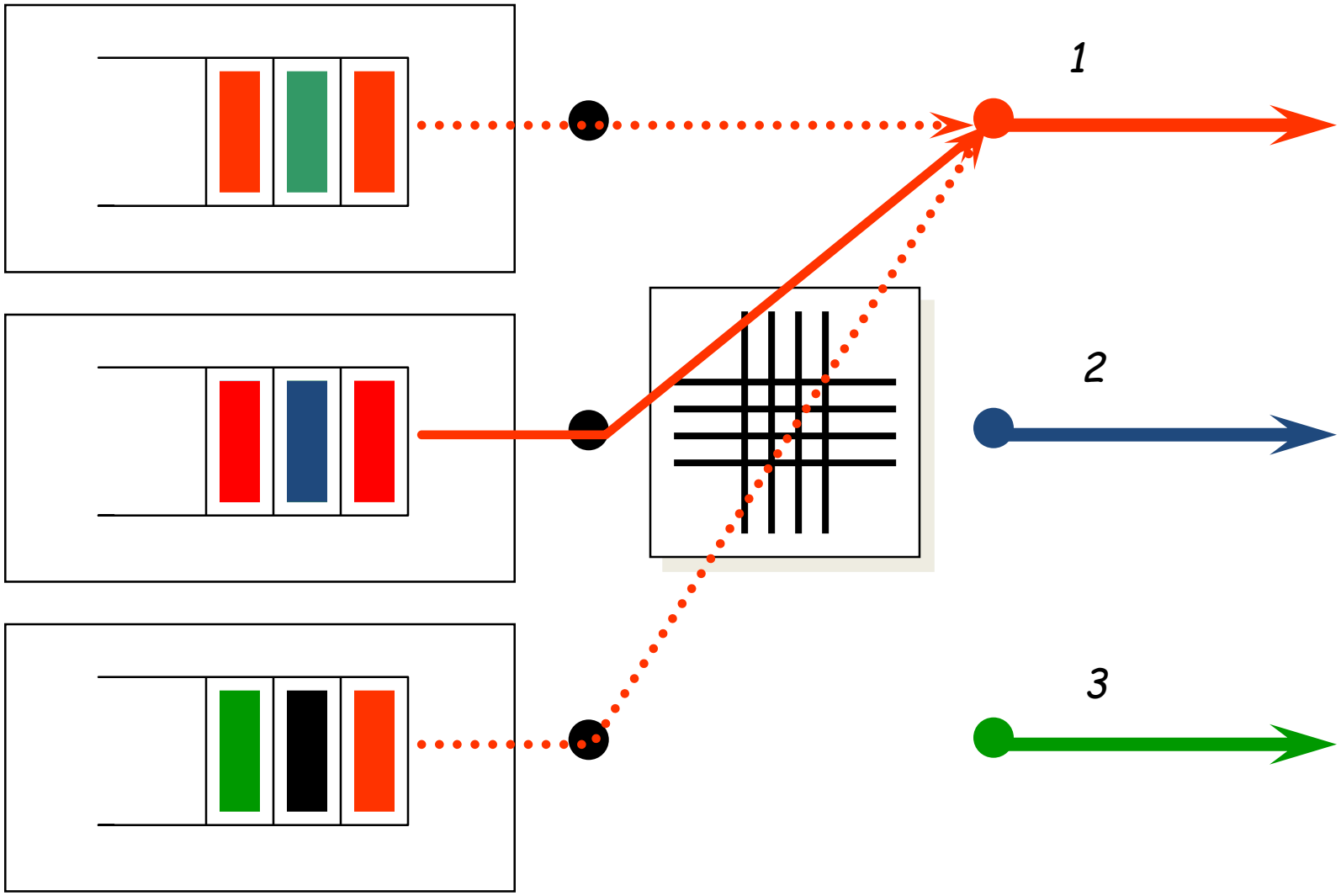


- No speedup in buffers or trunks (unlike output queued switch)
- Needs arbiter
- Problem: HOL (*head of line blocking)*
  - with randomly distributed packets, utilization at most 58.6%
  - worse with *hot spots*





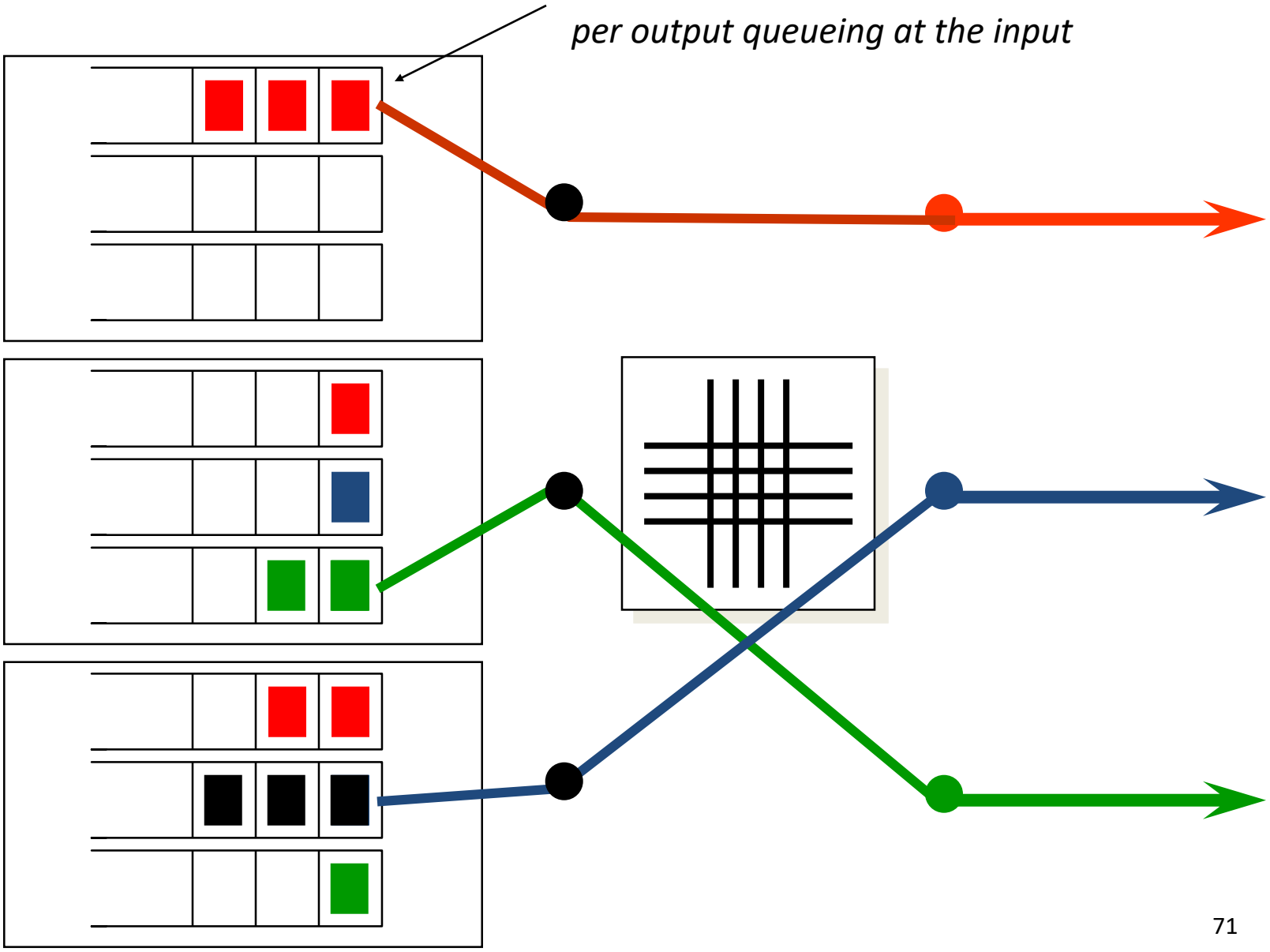
Head of Line blocking



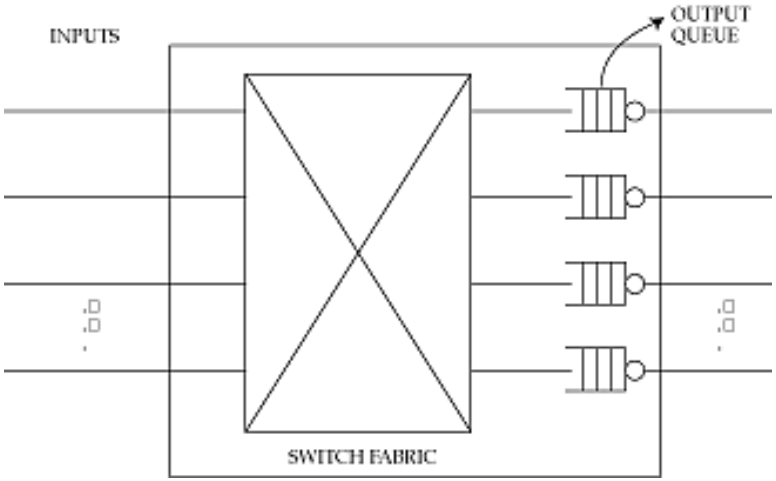
Dealing with HOL blocking

- Per-output queues at inputs (Virtual Input Queueing)
- Arbiter must choose one of the input ports for each output port
- How to select?
- Parallel Iterated Matching (PIM)
  - inputs tell arbiter which outputs they are interested in
  - output selects one of the inputs
  - some inputs may get more than one *grant*, others may get none
  - if >1 grant, input picks one at random, and tells output
  - losing inputs and outputs try again
- Used in many large switches

Virtual Input (Output) Queueing

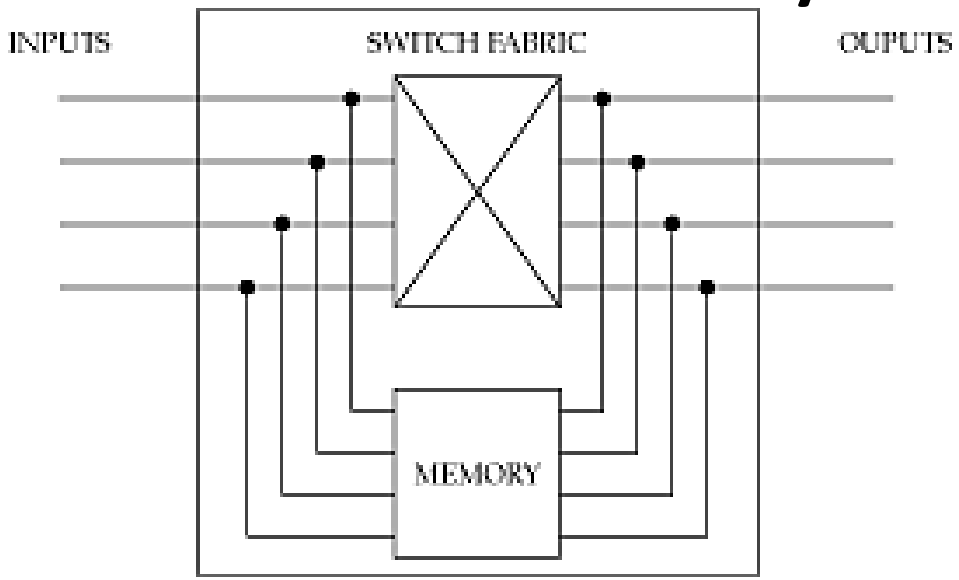


Output queueing



- Doesn't suffer from head-of-line blocking
- But output buffers need to run much faster than trunk speed (why?)
- Can reduce some of the cost by using the *knockout* principle
  - unlikely that all N inputs will have packets for the same output
  - drop extra packets, fairly distributing losses among inputs

Shared memory

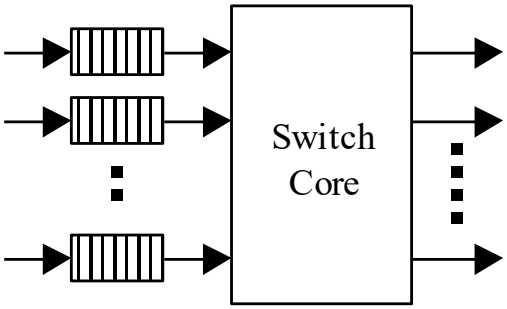


- Route only the header to output port
- Bottleneck is time taken to read and write multiported memory
- Doesn’t scale to large switches
- But can form an element in a multistage switch

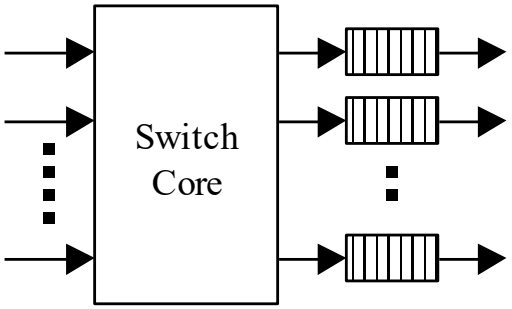
Buffered fabric

- Buffers in each switch element
- Pros
  - Speed up is only as much as fan-in
  - Hardware backpressure reduces buffer requirements
- Cons
  - costly (unless using single-chip switches)
  - scheduling is hard

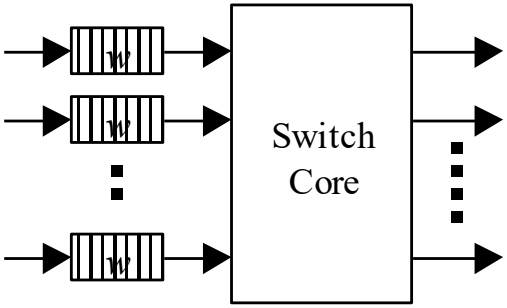
Summary of Buffer Placement



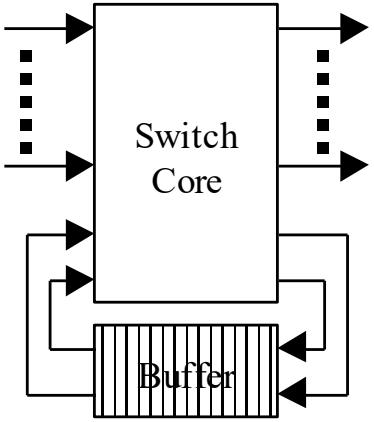
a) Input Queueing



c) Output Queueing



b) Window Selection



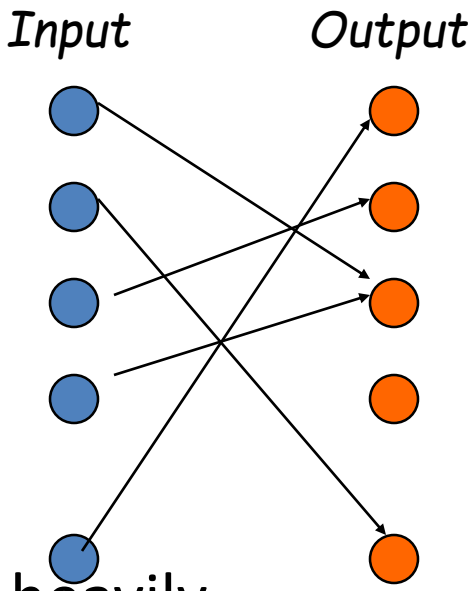
d) Shared Buffering

Switch Schedules

- To be determined in every packet slot
- No buffering
  - What would be the best schedule?
- Output Queued Switch
  - What would be the best schedule?
- Input Queued Switch
  - Simple FIFO
    - Window Selection
  - Virtual Input Queueing

Schedule for no Input Buffers

- Random Selection
  - For every output port
    - Look at all the contending inputs
    - Select one randomly
    - Drop all other packets (no buffer)
    - Good for uniform traffic
- Weighted Selection
  - Some contending inputs are more heavily weighted upon than others
  - Good for non-uniform traffic

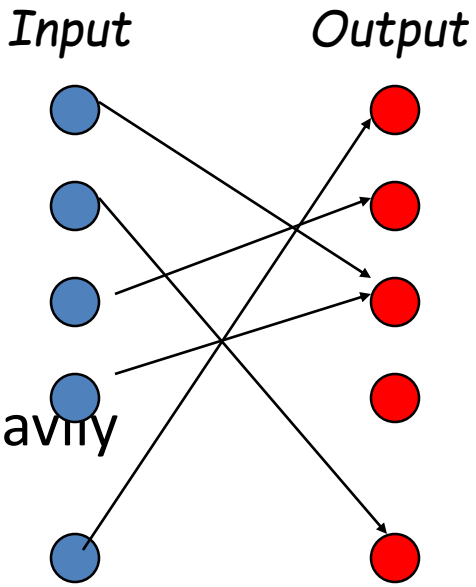


Schedule for Output buffered Switch

- What's the best switch schedule??
  - Don't need one !!!

Schedule for FIFO Input Buffers

- Random Selection
  - For every output port
    - Look at all the contending inputs
    - Select one randomly
    - Drop all other packets (no buffer)
    - Good for uniform traffic
- Weighted Selection
  - Some contending inputs are more heavily
  - weighted upon than others
  - Good for non-uniform traffic
- Window Selection
  - A window of “w” packets are looked for contention successively
  - Needs complex buffer management



Non-blocking Switch Performance

- Non-blocking Switch with no buffers
  - If output contention occurs, only one among *n* contending packets transmitted, all other dropped
  - For $N \rightarrow \infty$ and for Bernoulli packet arrival process (with a probability *p*), the Throughput is given by $(1-e^{-\rho})/\rho$
  - Max Throughput = 63.2%; (ρ=1) But remaining is all packet loss!!
- Non-blocking Switch FIFO input buffers
  - Throughput = 58.6%= $(2-\sqrt{2})$
  - Packet loss is a function of buffer size
  - For a Bernoulli packet arrival process (with a probability *p*)

$$P_{loss} < \frac{p(2-p)}{2(1-p)}\left[\frac{p^2}{2(1-p)^2}\right]^B$$

Switch Performance (contd ..)

- Non-blocking switch with non-FIFO buffers
  - packets are selected from a window (*w*) of buffer to minimize contention

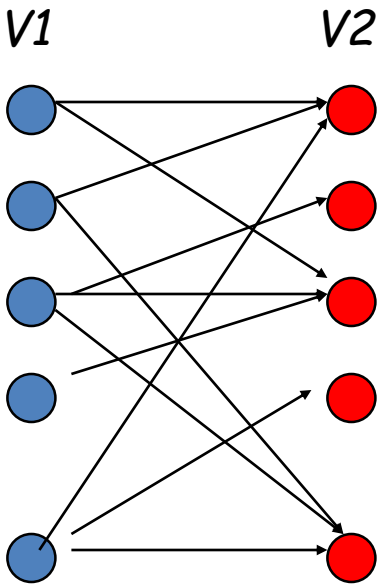
Size	FIFO	Window Size (*w*)							
N		1	2	3	4	5	6	7	8
2	75.0%	75%	84%	89%	92%	93%	94%	95%	96%
4	65.5%	66%	76%	81%	85%	87%	89%	94%	92%
8	61.8%	62%	72%	78%	82%	85%	87%	88%	89%
16		60%	71%	77%	81%	84%	86%	87%	88%
32		59%	70%	76%	80%	83%	85%	87%	88%
64		59%	70%	76%	80%	83%	85%	86%	88%
∞	58.6%								

Switch Performance (contd ..)

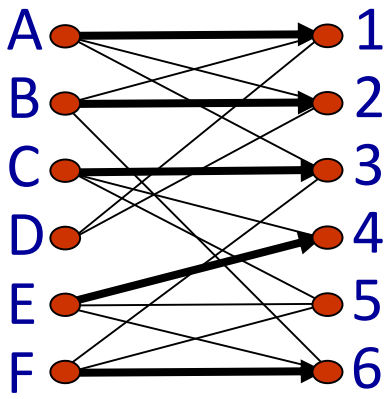
- Non-blocking Switch with Output buffers
  - Best performance (100% Throughput) as there is no HOL blocking
  - Delay performance depends on the output queueing
- Non-blocking Switch with Shared buffers
  - Packets lost in contention are stored in a separate buffer that feeds as direct input (depending upon the number of extra inputs)
  - Performance can be close to 100% with large shared buffer
  - Switch size grows

Side Note: Bipartite Graphs

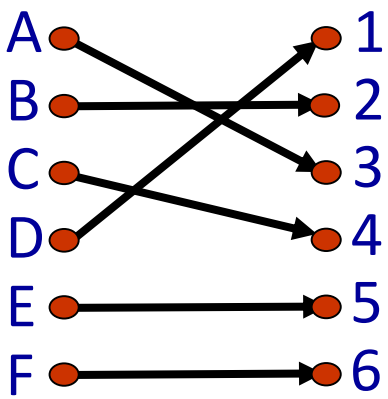
- A *bipartite* graph is a graph *G* whose vertex set *V* can be partitioned into two non empty sets *V1* and *V2* in such a way that every edge of *G* joins a vertex in *V1* to a vertex in *V2*



Side Note: Maximal and Maximum Size Matching



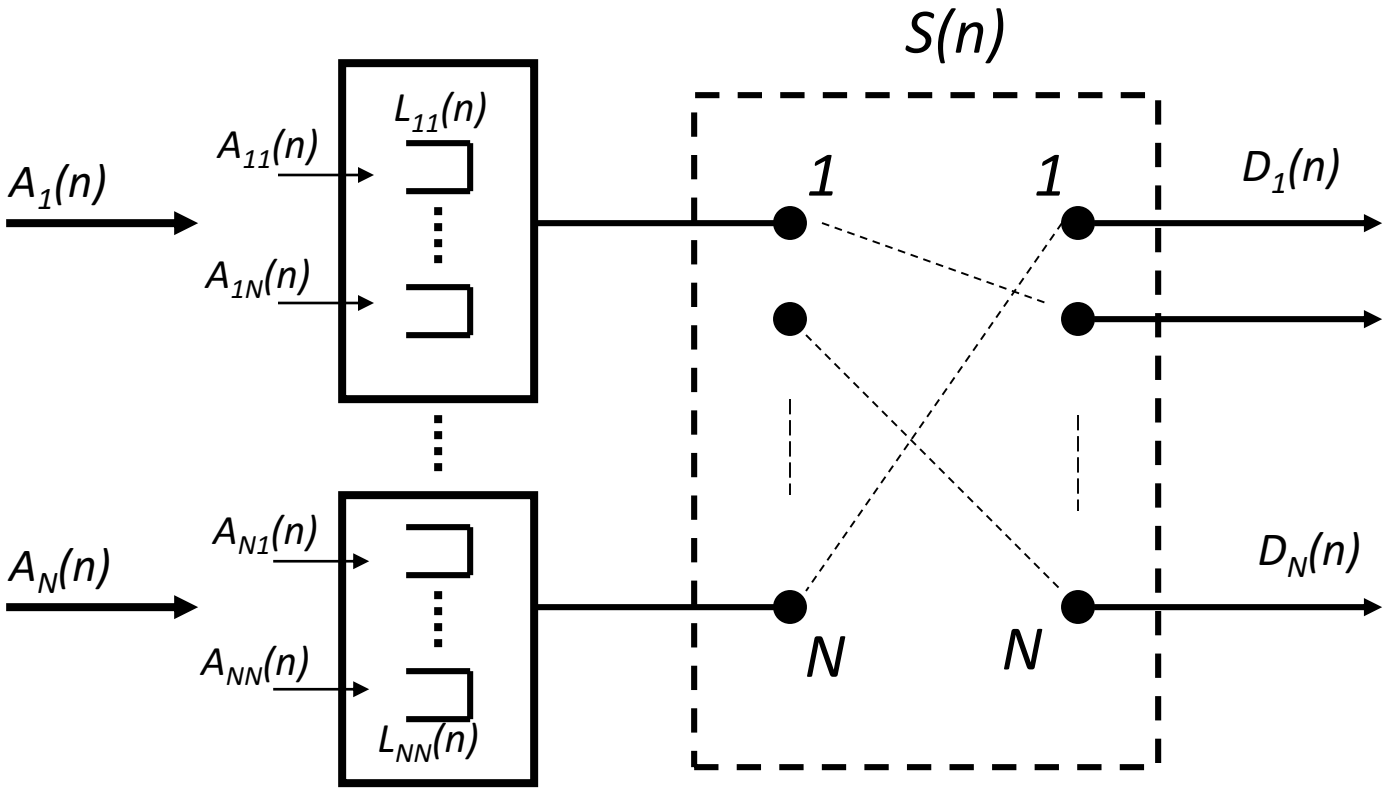
Maximal Matching



Maximum Matching

- A *matching*, M, of G is a subset of the edges E, such that no vertex in V is incident to more than one edge in M. Intuitively we can say that no two edges in M have a common vertex
- A matching M is said to be *maximal* if M is not properly contained (not a subset) in any other matching. Intuitively, this is equivalent to saying that a matching is *maximal* if we cannot add any edge to the existing set.
- A matching M is said to be *Maximum* if for any other matching M', $|M| \geq |M'|$. $|M|$ is the maximum sized matching.

Model for Virtual Queuing Switch

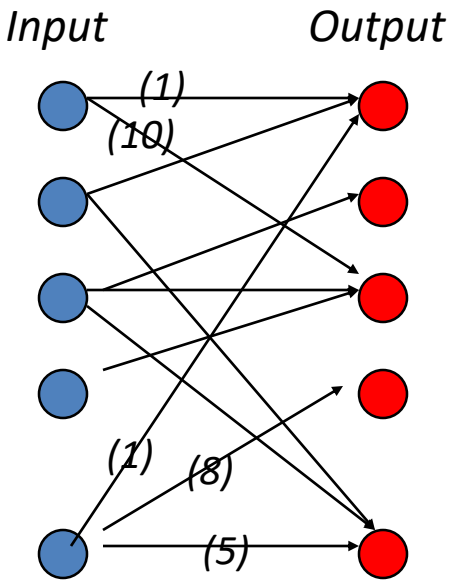


Stability Criterion

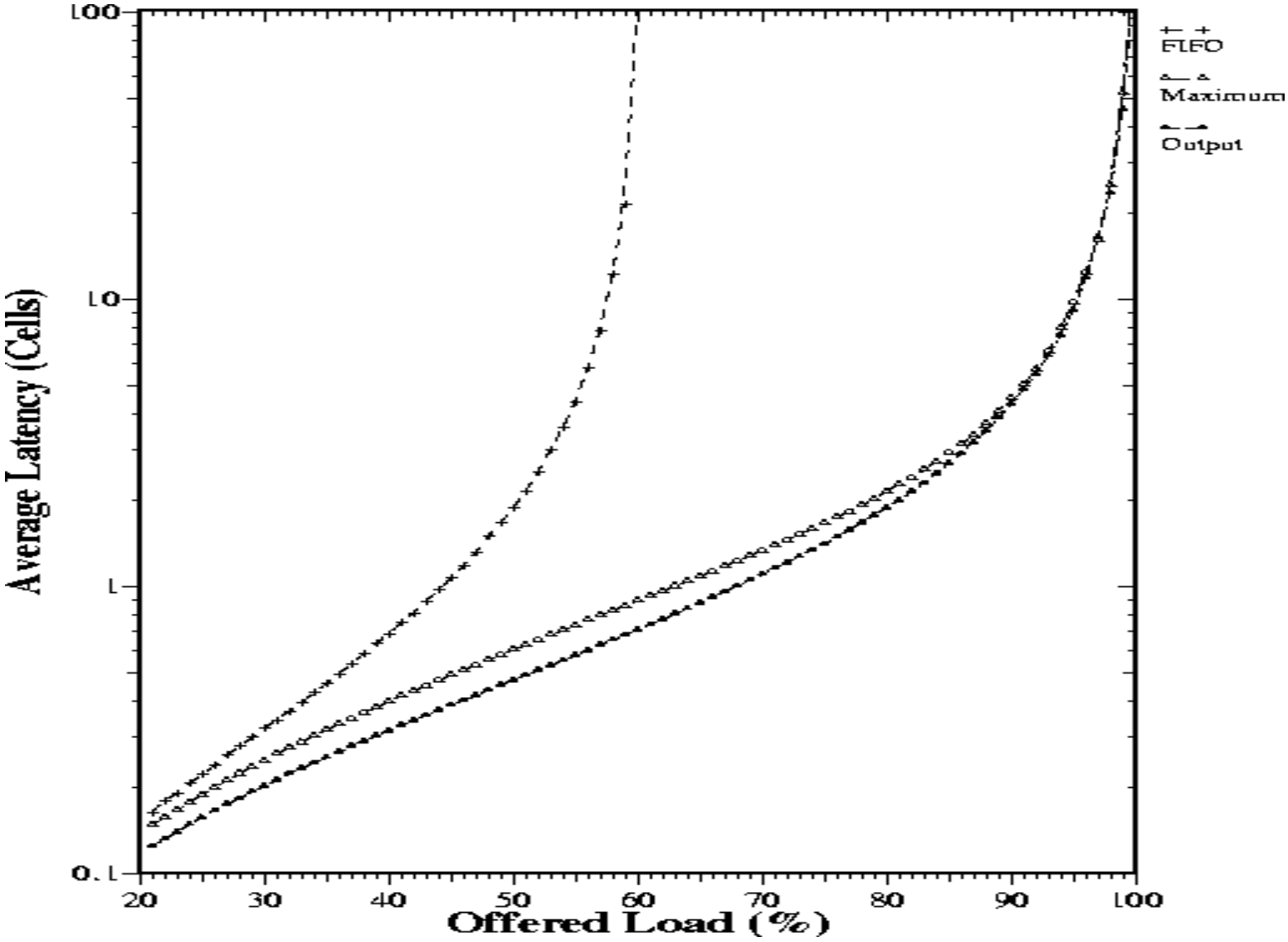
- Traffic Matrix $\Lambda=[\lambda_{ij}]$; $\lambda_{ij}= E\{A_{ij}\}$, i.e, average rate
- Traffic is admissible if
  - $\sum \lambda_{ij} \leq 1 \ \forall$ i (i.e., the total sum of traffic intensity for a given output from all inputs must be less than or equal to 1)
  - and $\sum \lambda_{ij} \leq 1 \ \forall$ j (i.e., the total sum of traffic intensity for all outputs from a given input port must be less than or equal to 1)
- Determine a Switch Schedule
  - S = $[s_{ij}]$ where s_{ij} = 0 or 1

Schedules for Virtual Input Queuing

- The problem boils down to maximum size matching in bipartite graphs; complexity $O(N^{2.5})$
- Maximum Size matching cannot guarantee 100% throughput if ties are broken down randomly
- Maximum Weight matching can achieve 100% throughput; complexity $O(N^3)$
- Maximum Size suitable for uniform traffic
  - Gives equal weightage to all ports
- Maximum Weight suitable for non-uniform traffic
  - Weight is the traffic intensity (or number of packets waiting to be transmitted for a given output port
  - Reduces queue lengths for heavily utilized ports



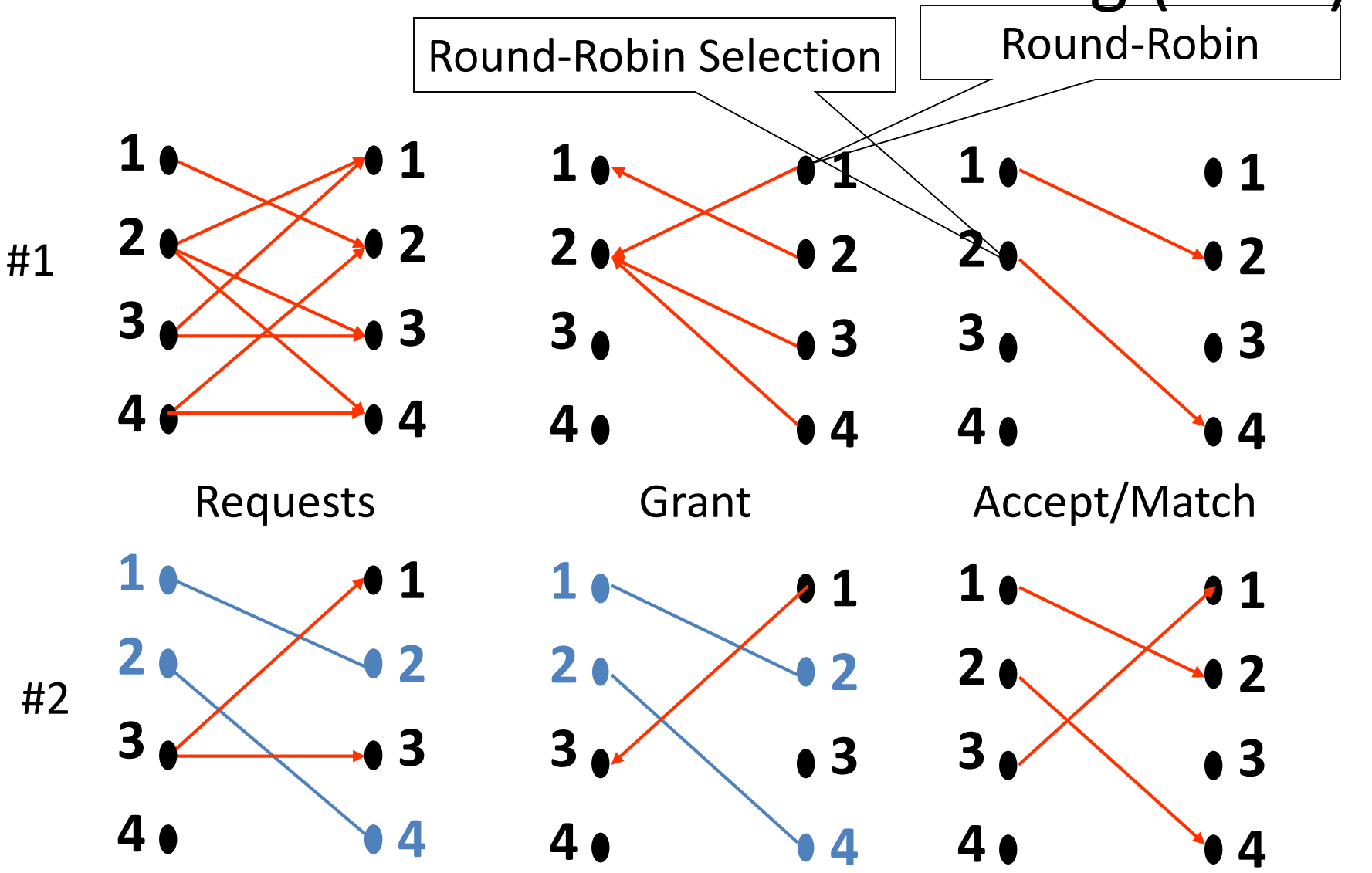
Performance Comparison



Algorithms to achieve 100% Throughput

1. When traffic is uniform (Many algorithms…)
2. When traffic is non-uniform, but traffic matrix is known
   - Technique: Birkhoff-von Neumann decomposition. [Chang '99]
3. When matrix is not known.
   - Technique: Lyapunov function. [McKeown et al. '96]
4. When algorithm is pipelined, or information is incomplete.
   - Technique: Lyapunov function. [Keslassy & McKeown '01]
5. When algorithm does not complete.
   - Technique: Randomized algorithm. [Tassiulas '00]
6. When there is speedup.
   - Technique: Fluid model. [Dai & Prabhakar '00]
7. When there is no algorithm.
   - Technique: 2-stage load-balancing switch. [Chang '01]
   - Technique: Parallel Packet Switch. [Iyer & McKeown '01]

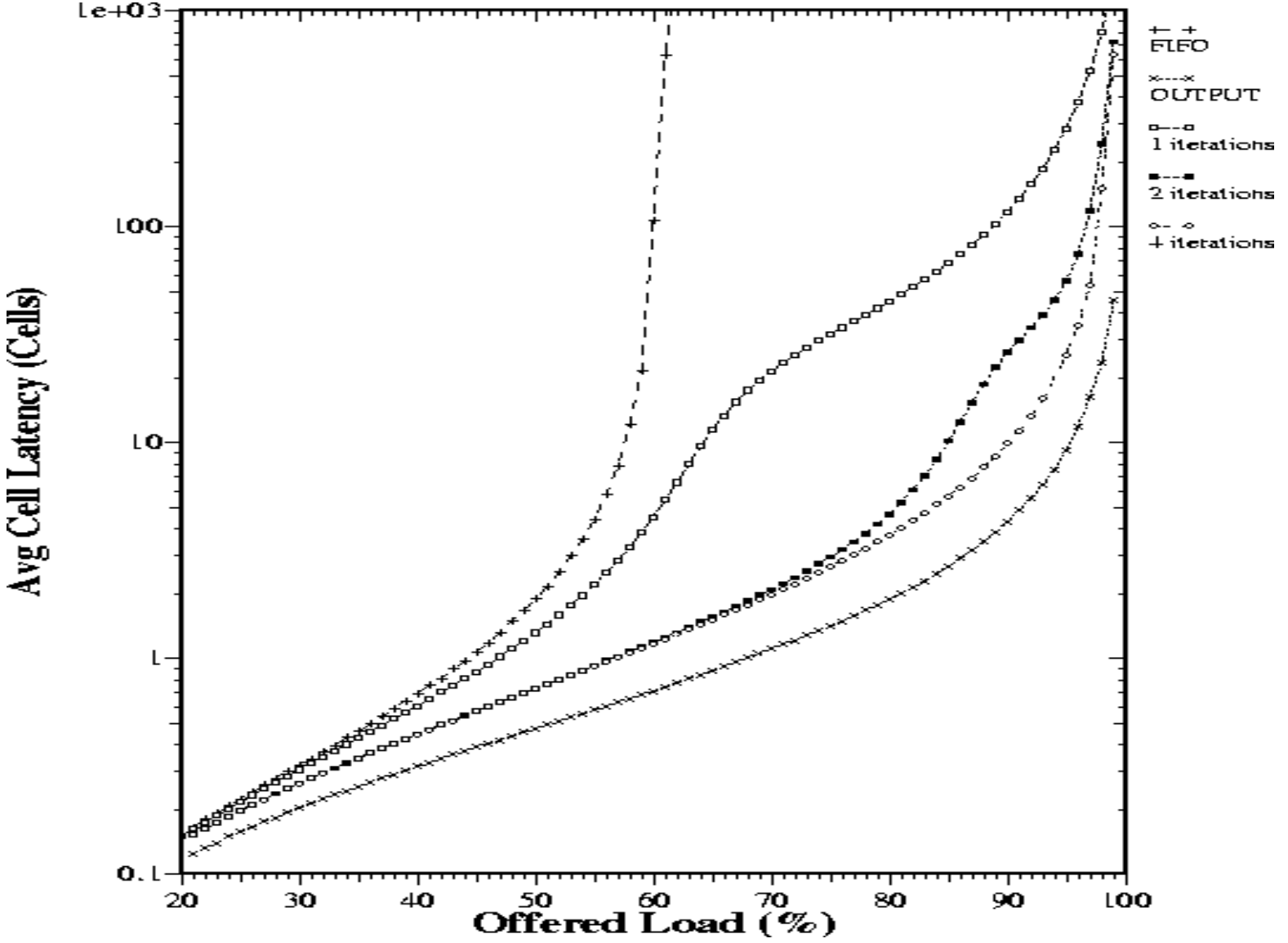
Iterated Round-robin Matching (iSLIP)



iSLIP

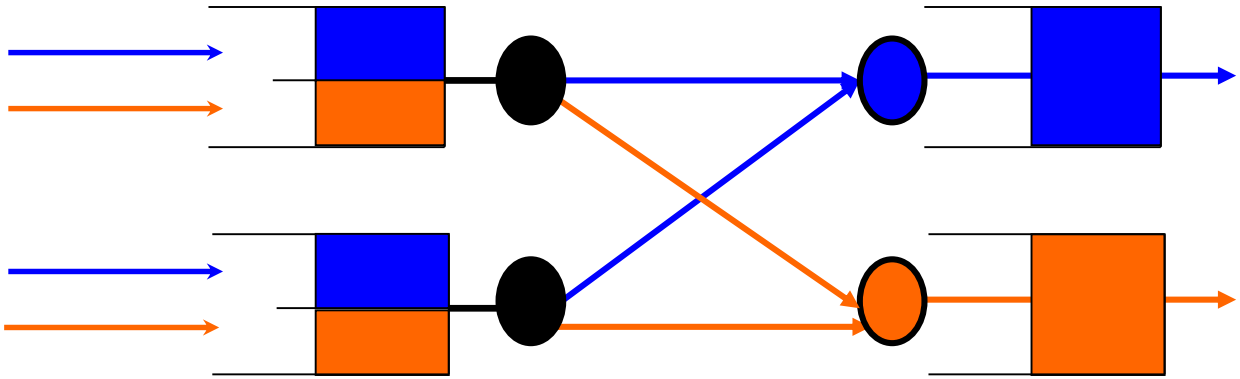
- Step 1 (Request) – Every unmatched input sends a request to every output for which it has a queued cell;
- Step 2 (Grant) – If an unmatched output receives any requests, it chooses the one that appears next in a fixed Round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer to the round-robin schedule is incremented to one location beyond the granted input if the grant is accepted in step 3 of the first iteration. The pointer is not incremented in subsequent iterations.
- Step 3 (Accept) – If an input receives a grant, it accepts the one in a round robin fashion. The pointer to the round-robin schedule is incremented to one location beyond the accepted output.
- Converges in at most N iterations; Average = $\log_2 N$

iSLIP Performance



Can we emulate an Output Queued Switch

- [Chuang, Goel et al. 1997] Precise emulation of an output queued switch is possible with a speedup of two and a "stable marriage" scheduling algorithms
  - Request (propose), Grant (engage), Admit (marriage)



Speedup

- Switch fabric links run at higher speed than external links
  - Fragmentation adds extra overhead
  - Internal links should be at a higher speed to take care of this
- Speed Up refers to how much faster the links/switch should run
  - Generally doesn’t include the fragmentation overheads
  - Needs output buffers as the output links are slower than fabric links
  - More packets from the inputs are switched to output ports
    - reduces HOL blocking
    - reduces requirement for output buffers when compared to pure output queuing

Combined Input Output Queuing

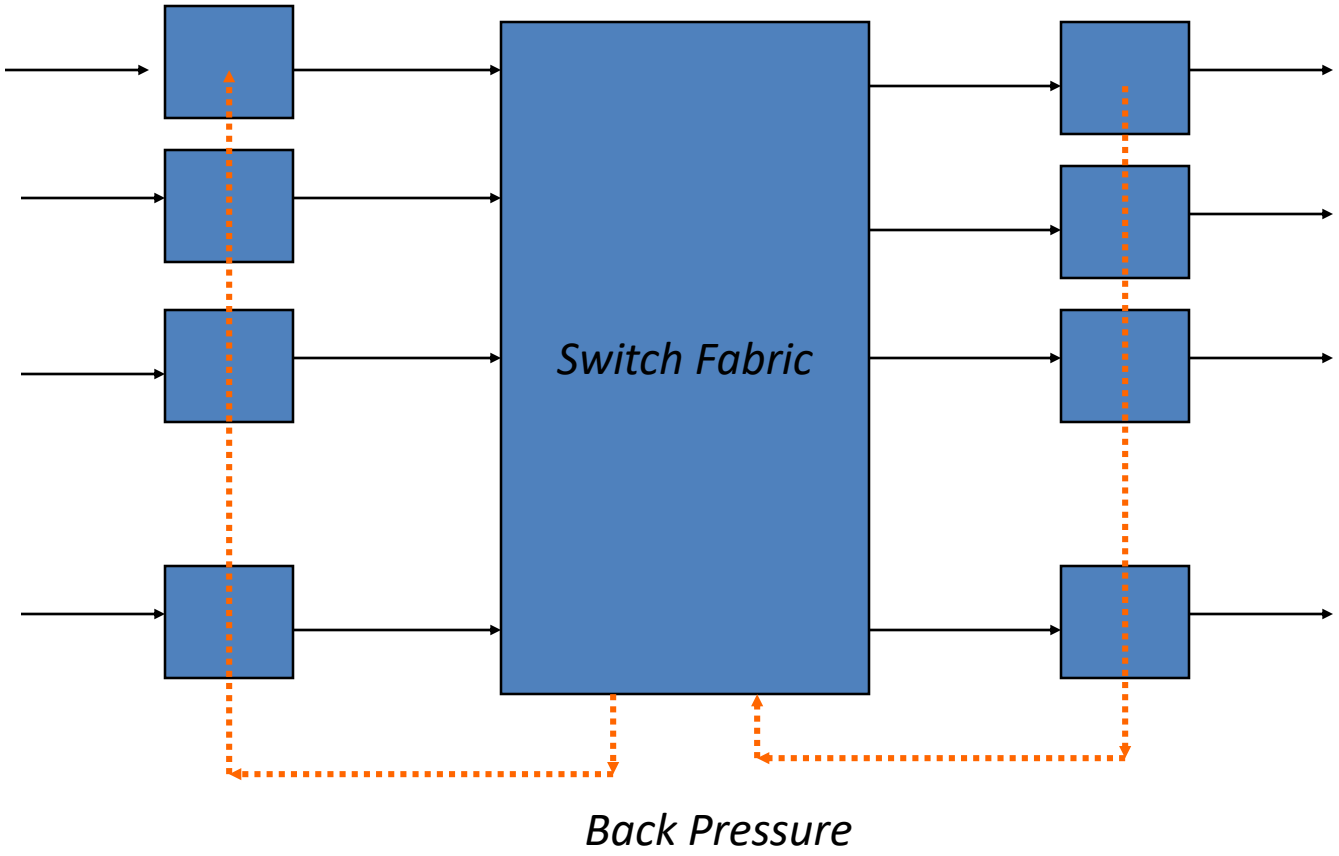
- Output Queuing achieves best performance
  - But requires N times fabric speed-up
  - Huge memory bandwidth requirements (N times)
- Input Queuing (even with Virtual Queuing) cannot achieve similar performance
  - There is no speed-up in the fabric
  - Memory operates at the 2 line rate (one read and one write)
- Combined Input and Output Queuing
  - What's the speed up required to achieve performance comparable to pure Output queuing?
    - Answer: 2-1/N (Nick McKeown et al)

    Speed up is the increase in fabric speed relative to interface speed

Hybrid solutions

- Buffers at more than one point
- Becomes hard to analyze and manage
- But common in practice
  - Backpressure schemes are employed to push queuing to only at the input
  - Backpressure communication mechanism adds overhead to internal headers
  - Three types:
    - Request and Grant (Credit based flow control)
    - On-Off (Buffer Threshold exceeding triggers the control)
    - Rate based (Backpressure is based on fair share bandwidth calculations)

Backpressure Schemes



- Output port buffer congestion signaled to fabric (How can we do this?)
- Fabric congestion signaled to inputs (How can we do this?)

Backpressure Schemes

- Credit based
  - Depending upon the buffer size and availability, certain credits (tokens) are given to each input port. Each input can only transmit up to the available credits
  - Credit allocation algorithms
- On-Off
  - Threshold based; Needs hysteresis to reduce flapping
  - buffer allocation for latency
- Rate-based
  - backpressure is based on rate measurements (or allocation)
  - Fair share calculations

Outline

- Circuit switching
- Packet switching
  - Switch generations
  - Switch fabrics
  - Buffer placement
  - Multicast switches

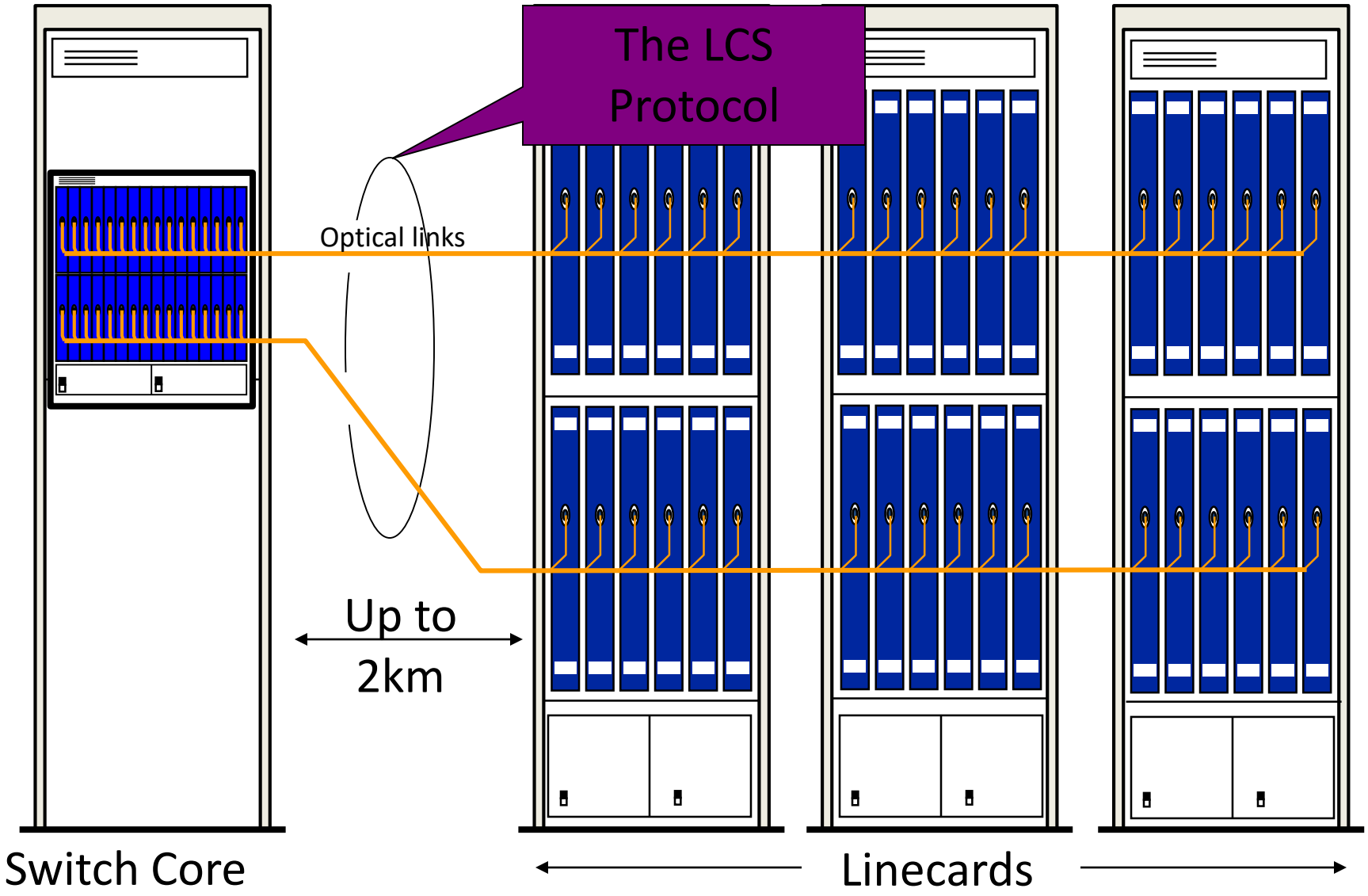
Multicasting

- Useful to do this in hardware
- Assume port-mapper knows list of outputs
- Incoming packet must be copied to these output ports
- Two sub problems
  - generating and distributing copies
  - VCI translation for the copies

Packet Size Impacts

- Fixed Length Packets
- Variable Length Packets
  - Building a synchronous switch with variable length is cumbersome
  - Either pad the packets to make it a constant length (or) do
  - Segmentation and Reassembly (SAR)
  - Packet headers are headed for each segment
    - Increases total bytes to be transported across the fabric
    - Increase the link speeds and switching capacity of the fabric to compensate this

Fourth Generation Routers/Switches



Fourth Generation Routers/Switches

The LCS Protocol

What is LCS?

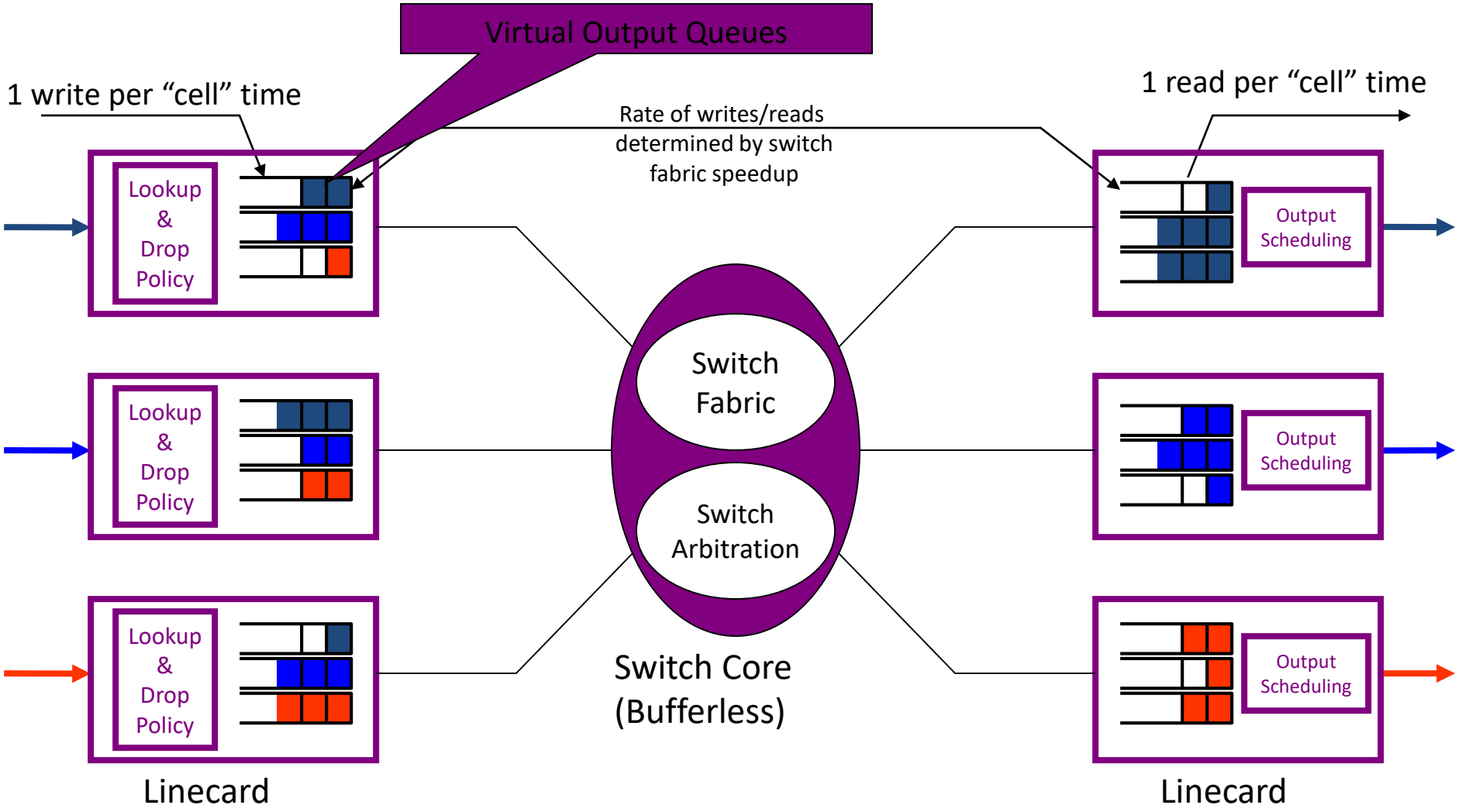
1. It is a Line Card to Switch (LCS) protocol
2. Credit-based flow control: *enables separation*.
3. Label-based multicast: *enables scaling*.

Its Benefits

1. Large Number of Ports.
   Separation enables large number of ports in multiple racks.
2. Minimizes Switch Core Complexity and Power.
   Switch core can be bufferless and lossless. QoS, discard etc. performed on linecard.

Fourth Generation Routers/Switches

Queueing Structure



Typically <5Tb/s aggregate capacity

Myths about CIOQ-based crossbar switches

1. "Input-queued crossbars have low throughput"
   - An input-queued crossbar can have as high throughput as any switch.
2. "Crossbars don't support multicast traffic well"
   - A crossbar *inherently* supports multicast efficiently.
3. "Crossbars don't scale well"
   - Today, it is the number of chip I/Os, not the number of crosspoints, that limits the size of a switch fabric. Expect 5Tb/s crossbar switches.

Myths about CIOQ-based crossbar switches (2)

4. "Crossbar switches can't support delay/QoS guarantees"
   - With an internal speedup of 2, a CIOQ switch can precisely emulate a shared memory switch for all traffic.

What makes sense today?

	Shared Memory	Input Queued	CIOQ	Multistage
Blocking	No	No	No	Yes
Speedup	High	High	Small	High
Emulation of SM	Yes	No	Yes	No
Multicast	Good	Good	Good	Poor
Resequencing	No	No	No	Yes
Power	Low	OK	OK	High
Packaging	-	OK	OK	Complex

What makes sense tomorrow?

Single-stage (if possible):

- Reduces complexity
- Minimizes interconnect b/w
- Minimizes power

Basic Architectural Components of an IP Router