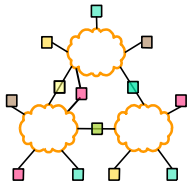


Computer Networks

(Graduate level)

Lecture 6: **Router & Switch Design**

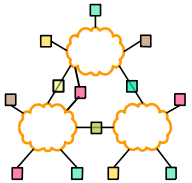
University of Tehran

Dept. of EE and Computer Engineering

By:

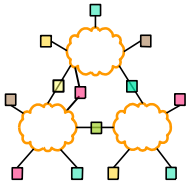
Dr. Nasser Yazdani

Routers



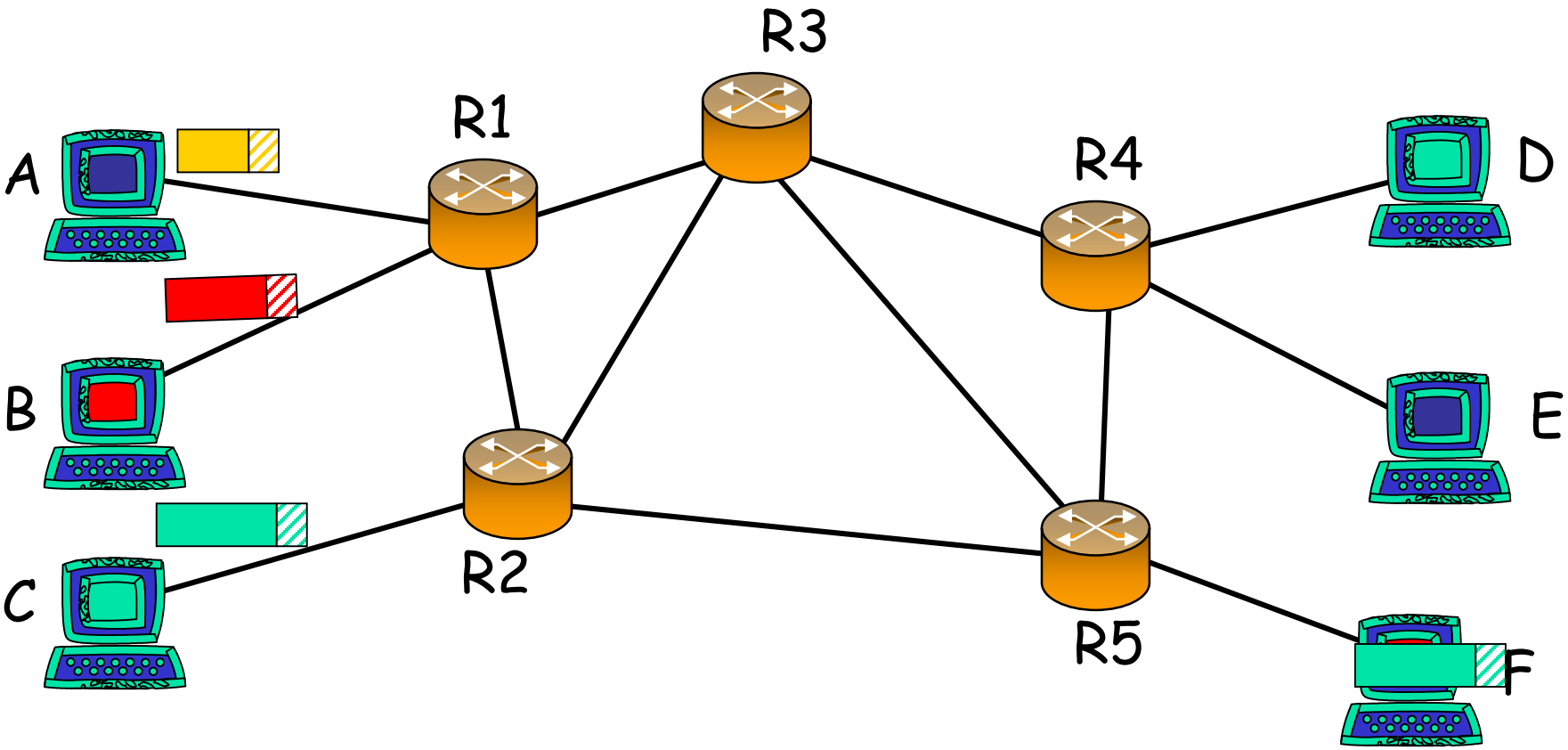
- How do you build a router
- How to process and forward packets
- Switch fabric design.
- Assigned reading
  - [P+98] A 50 Gb/s IP Router
  - Chapter 3 of the book

Outline

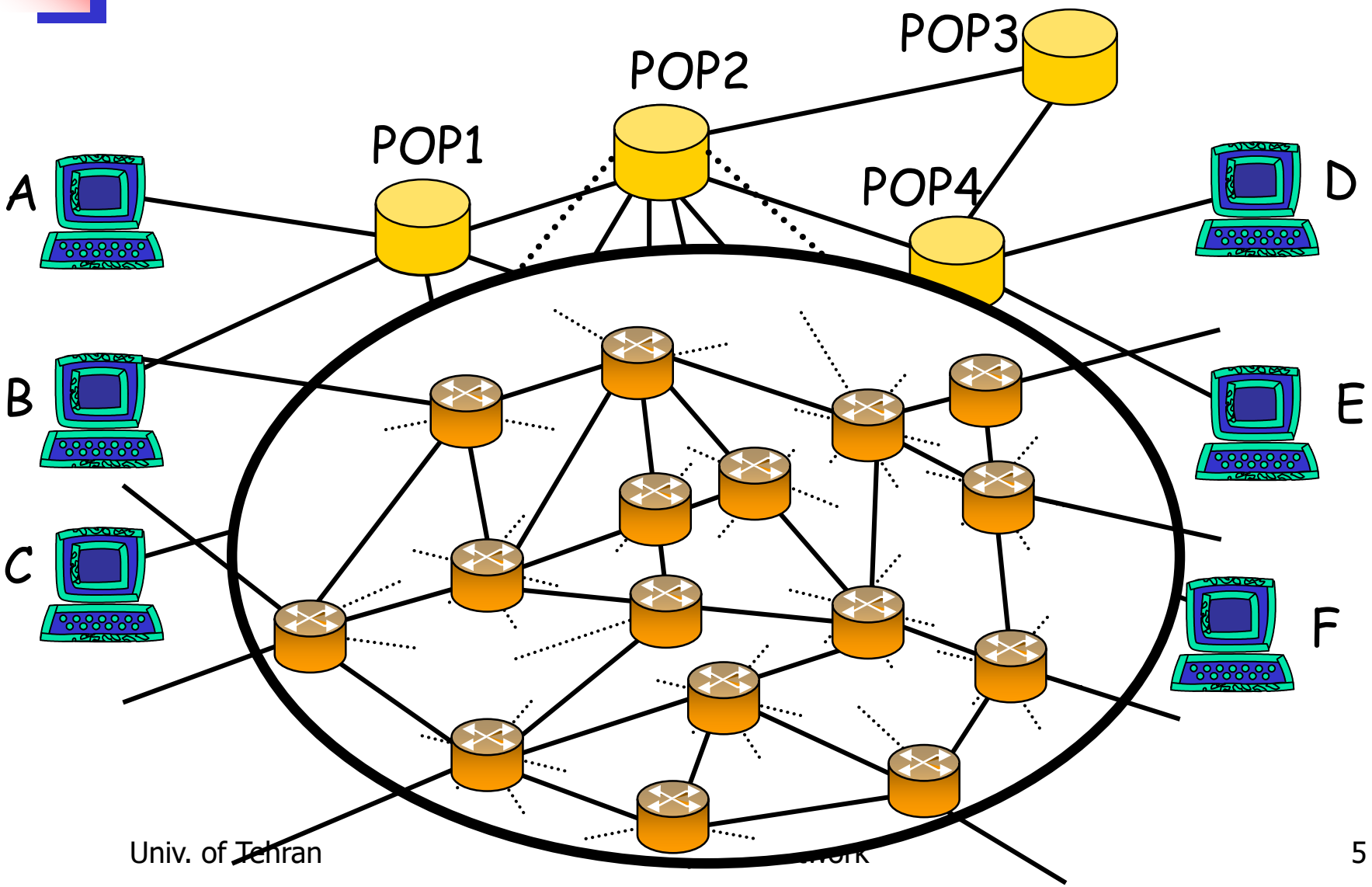


- What is a router?
- Router Architecture
  - Different router Architectures
  - The evolution of router architecture.
- A case Study

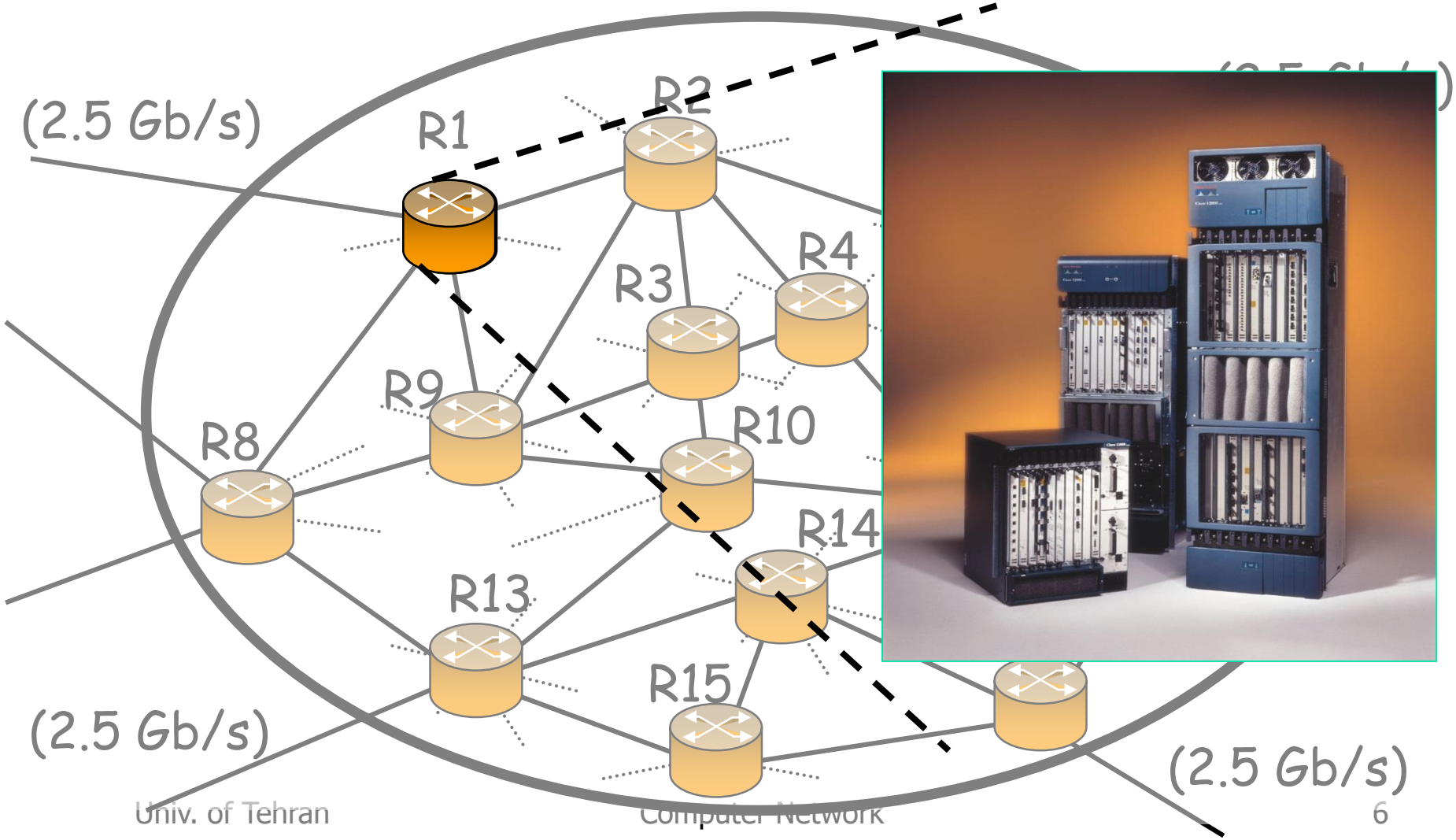
What is Routing?



Points of Presence (POPs)

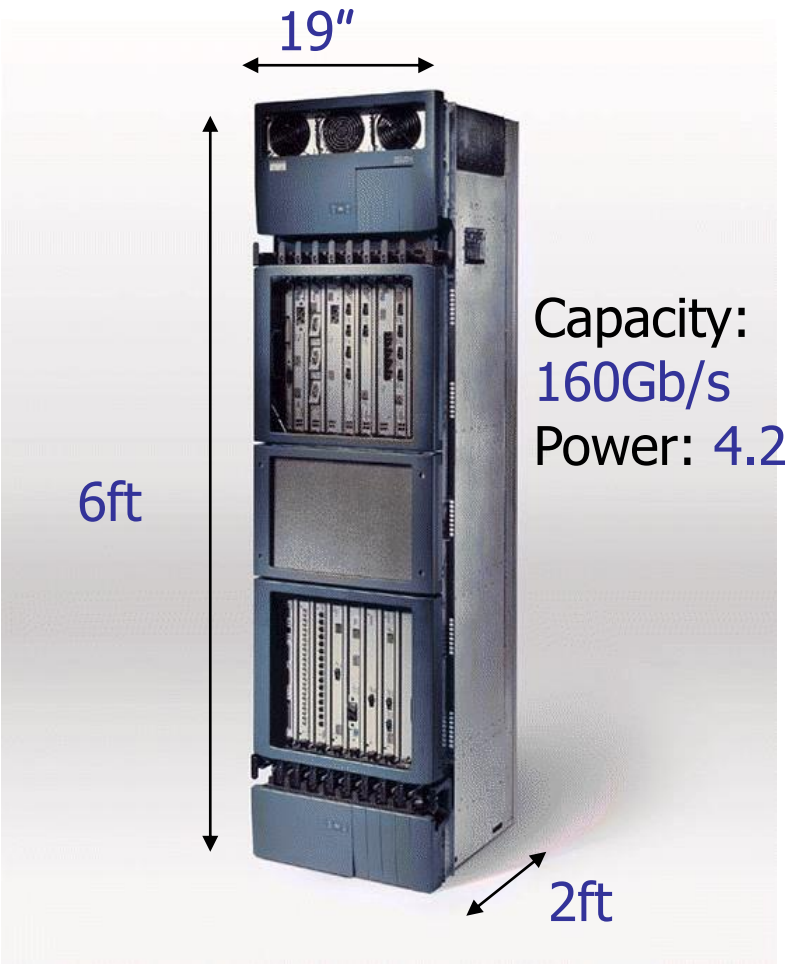


High Performance Routers Usage



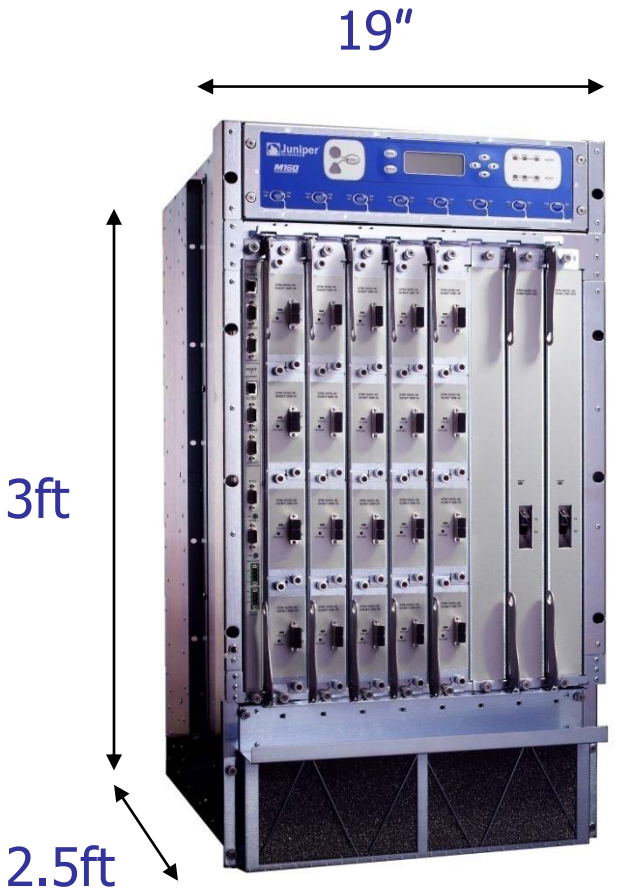
What a Router Looks Like

Cisco GSR 12416



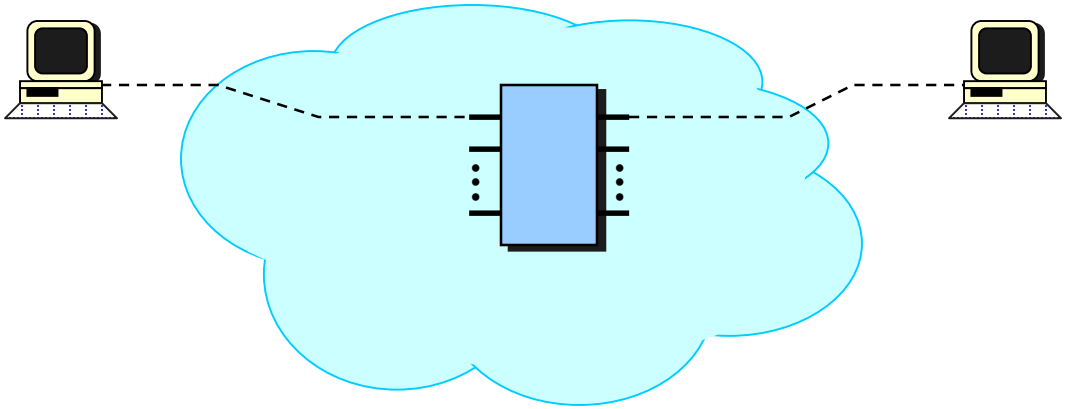
Capacity: 160Gb/s
Power: 4.2kW

Juniper M160



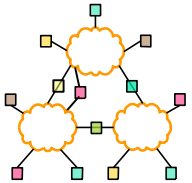
Capacity: 80Gb/s
Power: 2.6kW

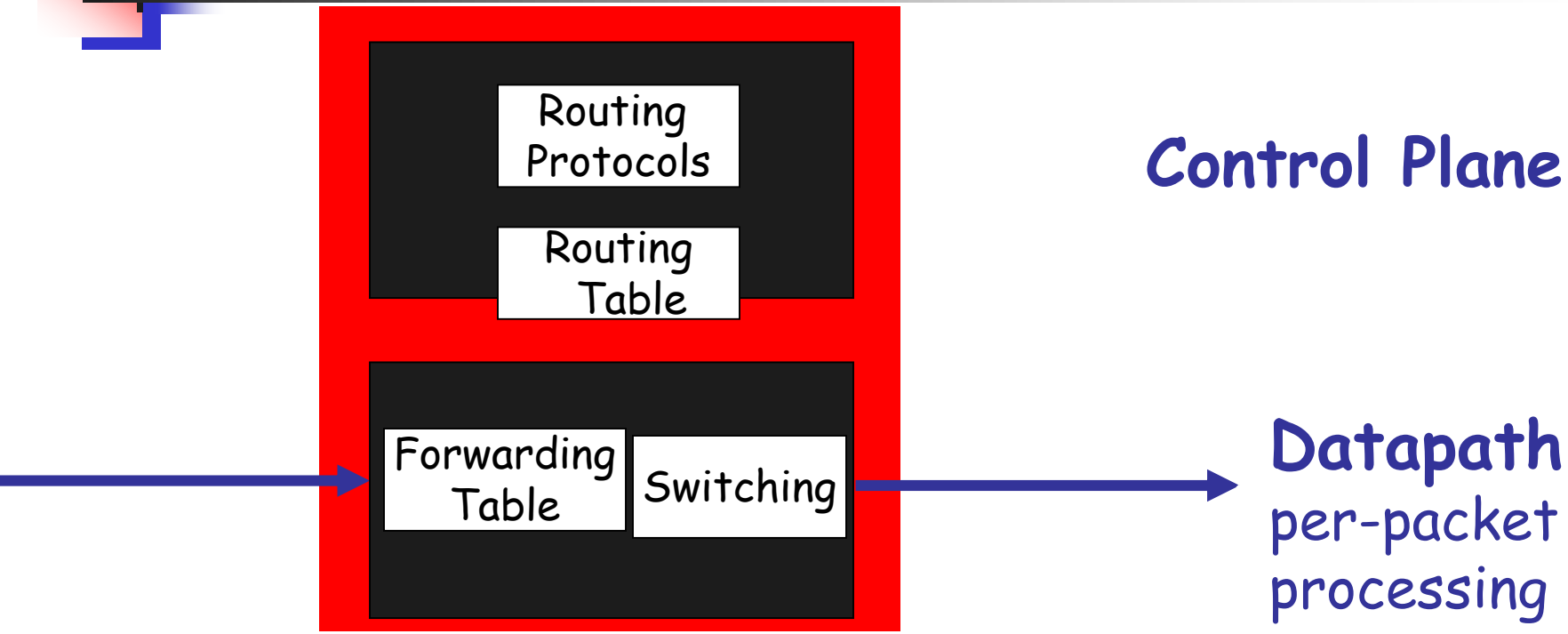
IP Router



- Router implements the following functionalities
  - Forward packet to corresponding output interface, Forwarding engine.
  - Manage routing/congestion

Components of an IP Router

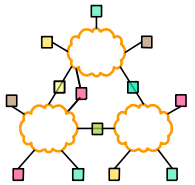




Two distinguish functional planes

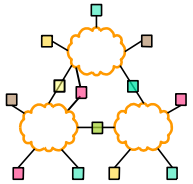
- Slow path or Control Plane
- Fast Path or Data Plane

Per-packet processing

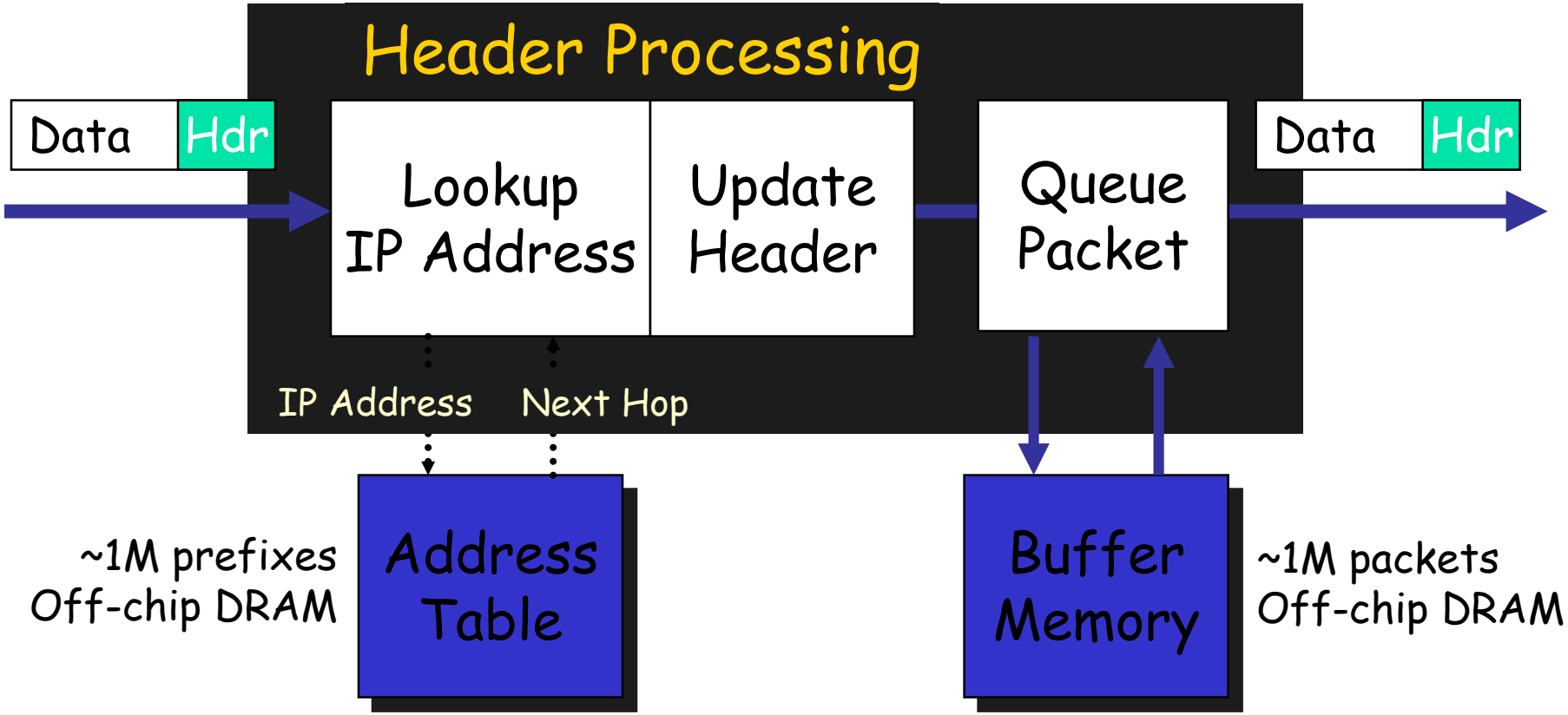


1. Accept packet arriving on an incoming link.
2. Lookup packet destination address in the forwarding table, to identify outgoing port(s).
3. Manipulate packet header: e.g., decrement TTL, update header checksum.
4. Send packet to the outgoing port(s).
5. Buffer packet in the queue.
6. Transmit packet onto outgoing link.

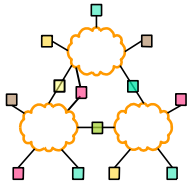
Generic Network Processor

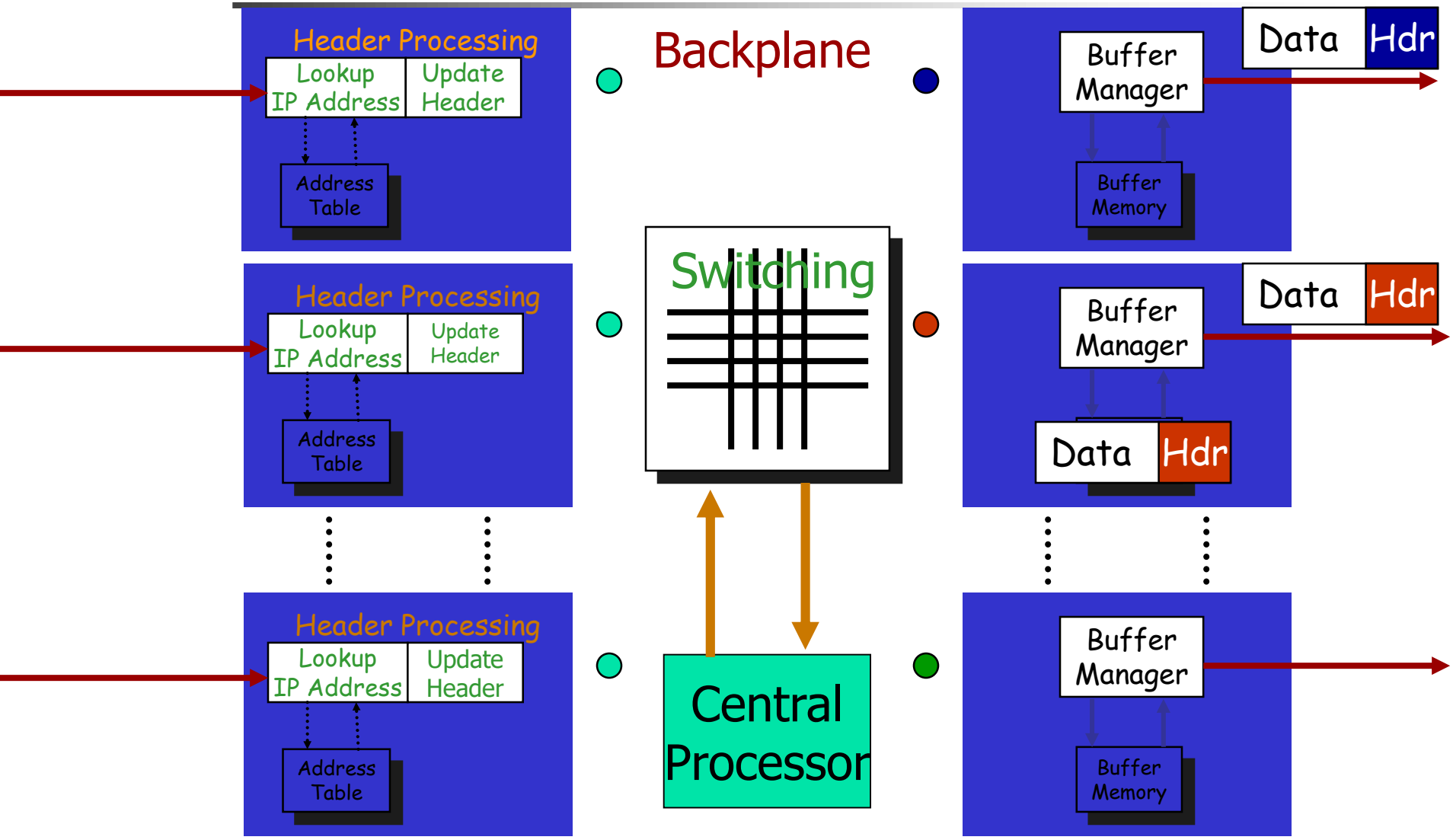


Processing View

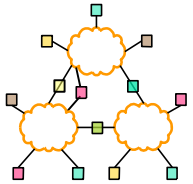


Generic Router Architecture



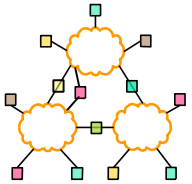


Function division



- Line cards
  - Network interface cards
- Input interfaces:
  - May enqueue packets and perform scheduling
- Output interfaces:
  - May enqueue packets and perform scheduling
- Forwarding engine
  - Fast path routing (hardware vs. software)
- Backplane
  - Switch or bus interconnect
- Network controller or central unit
  - Handles routing protocols, error conditions

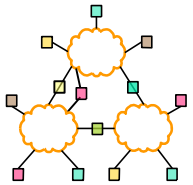
Router Architectures

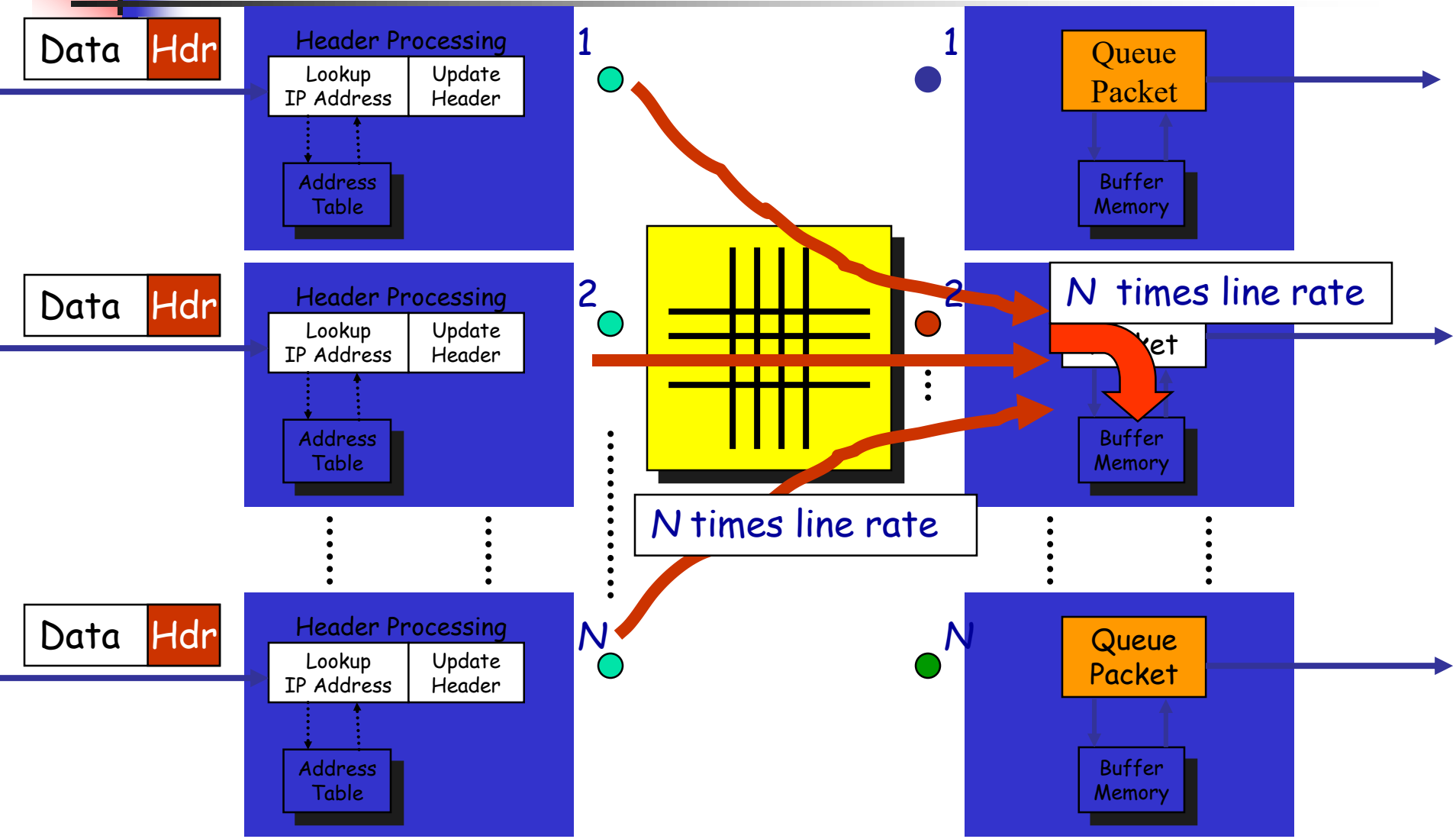


Where to queue incoming packets?

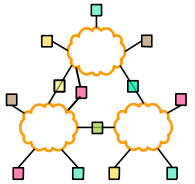
- Output queued
- Input queued
- Combined Input-Output queued
- It is the same as switch architecture

Output Router

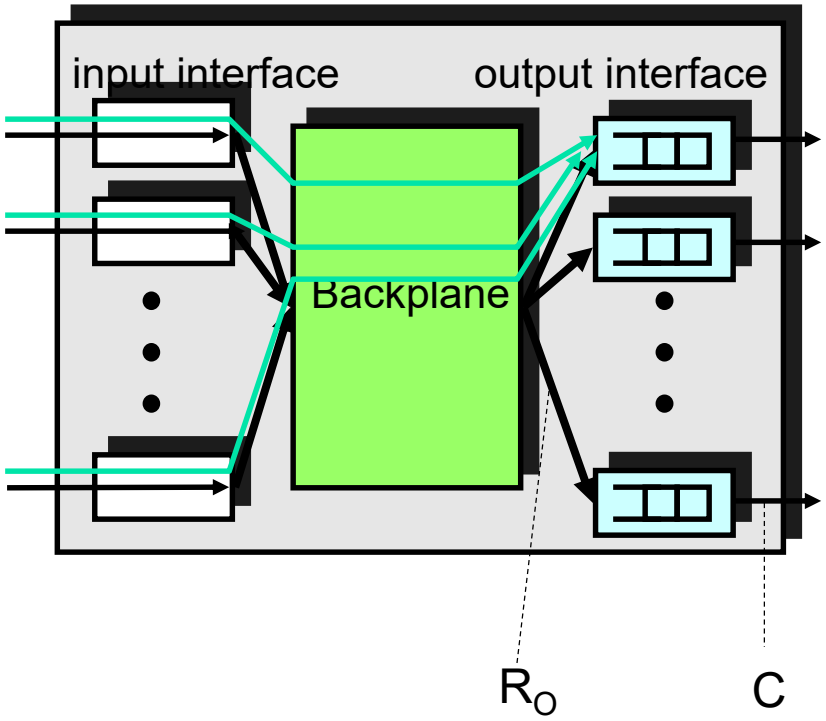




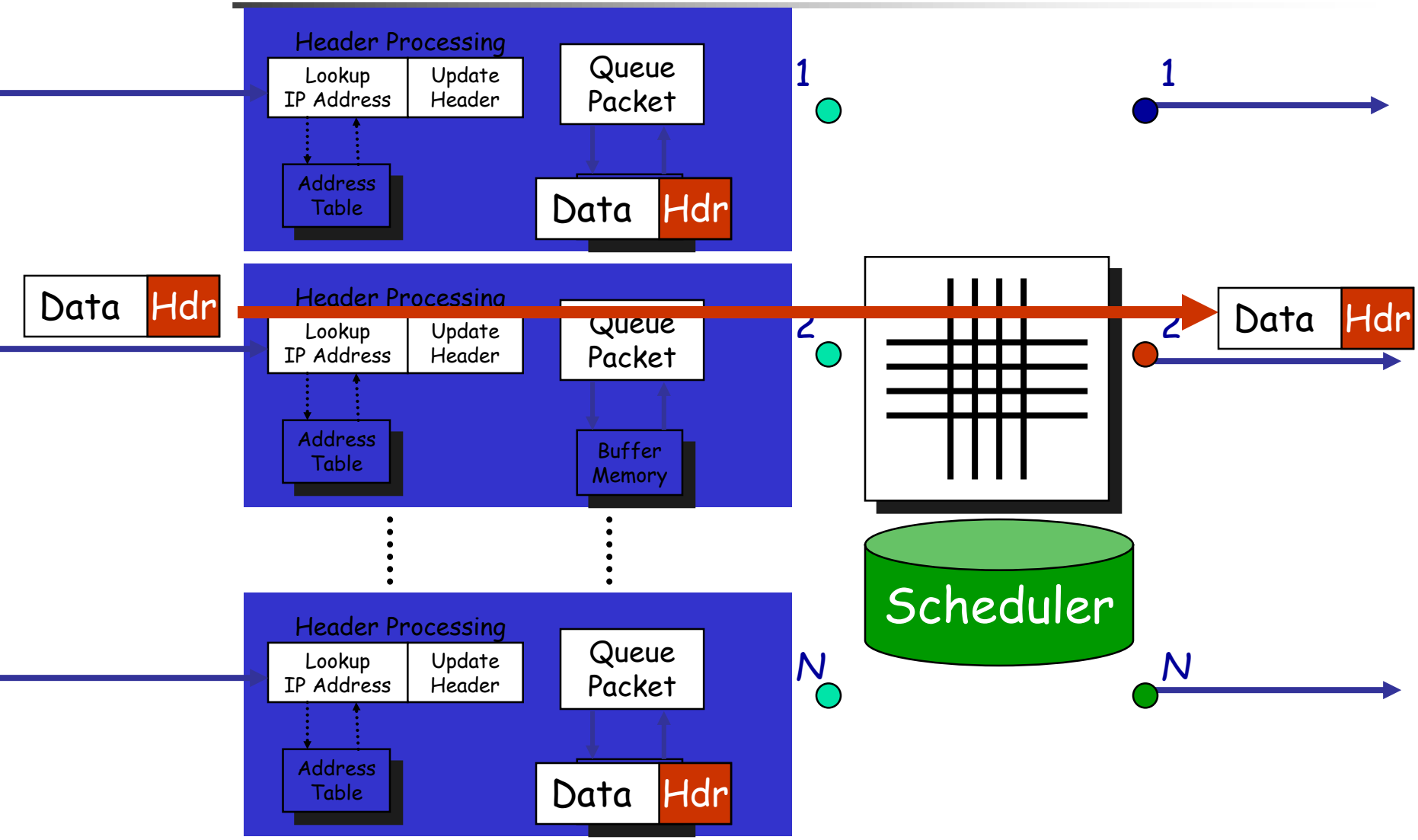
Output Queuing (OQ)



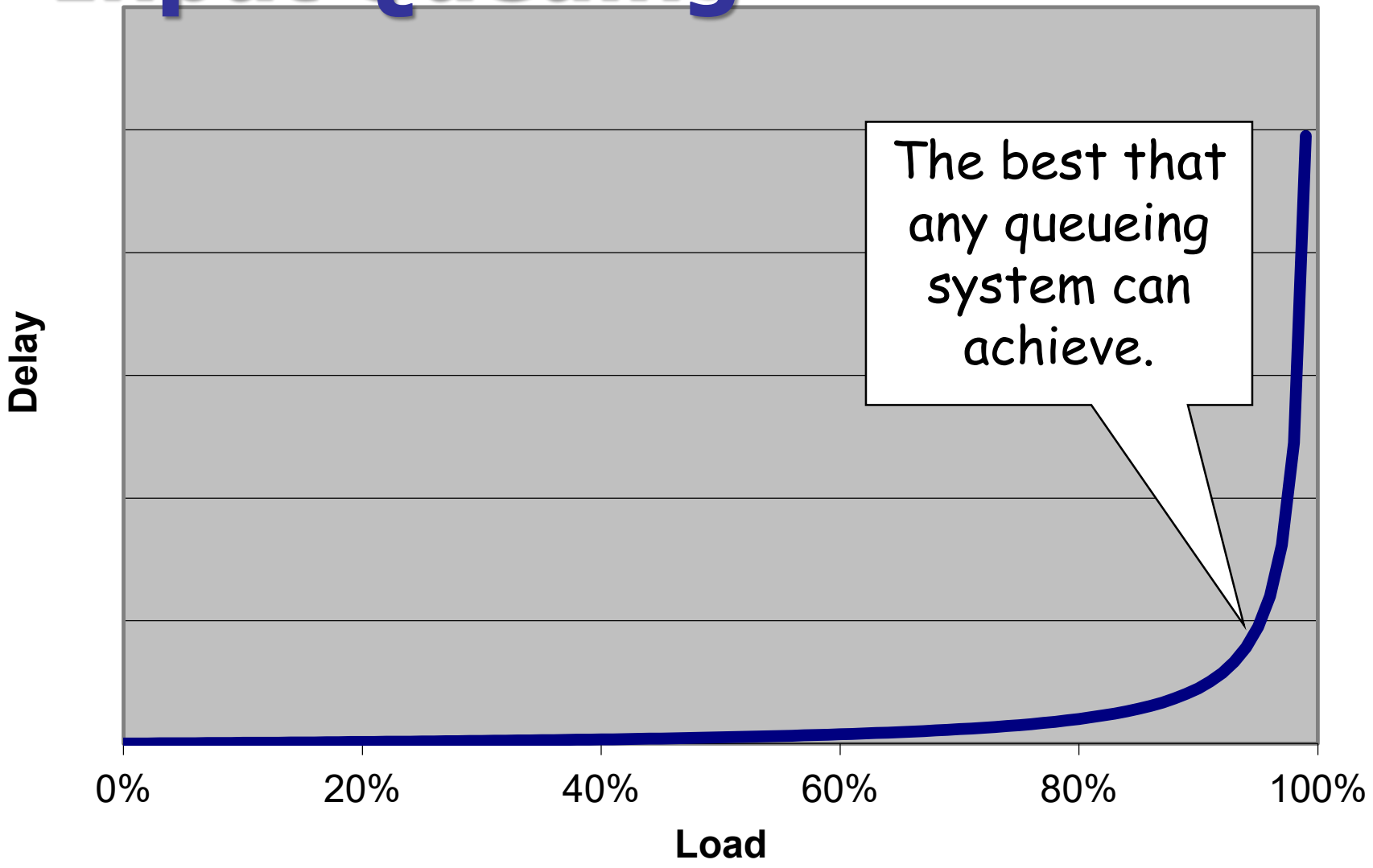
- Only output interfaces store packets
- Advantages
  - Easy to design algorithms: only one congestion point
- Disadvantages
  - Requires an output speedup of N, where N is the number of interfaces → not feasible



Input Router

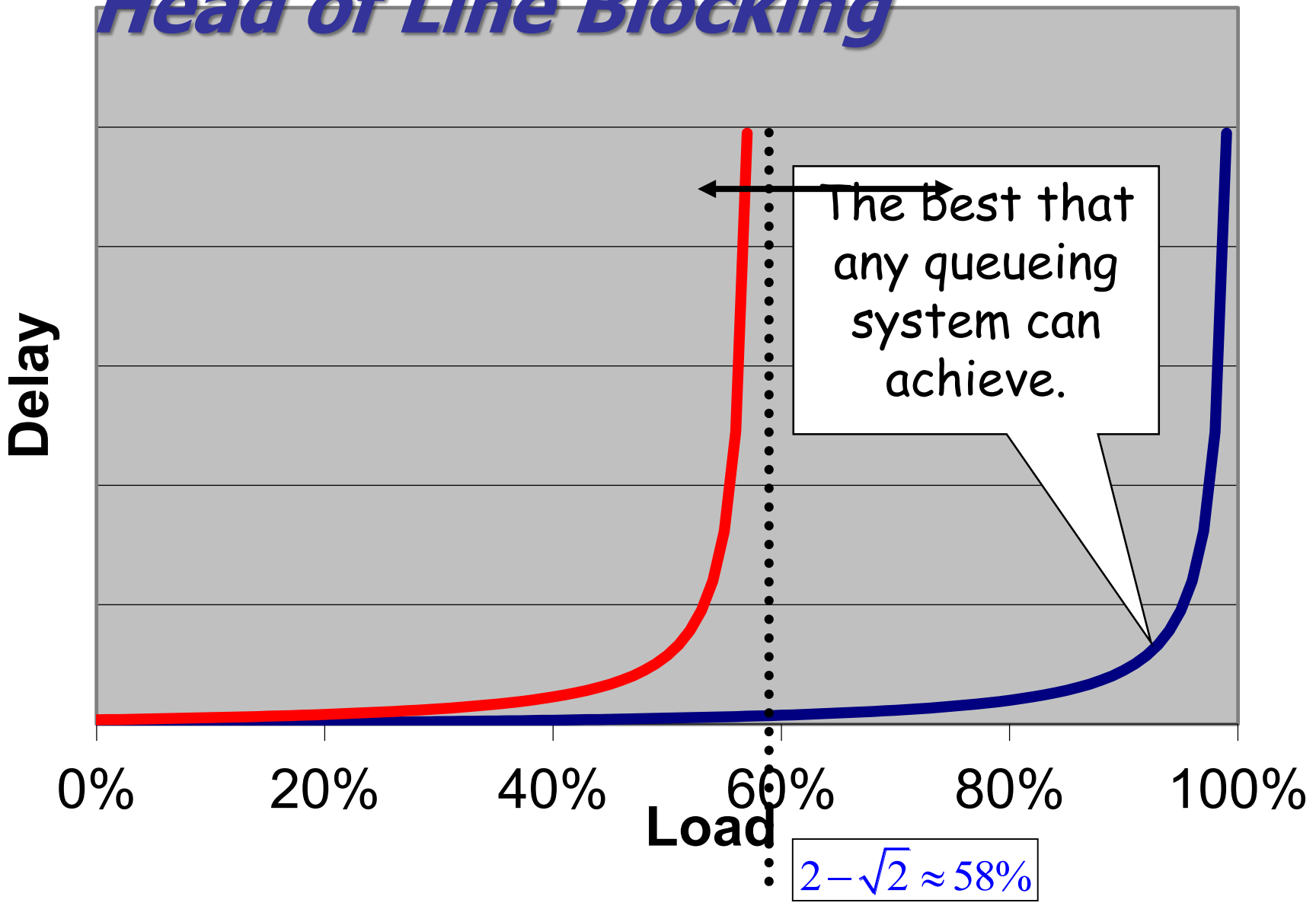


Input Queuing

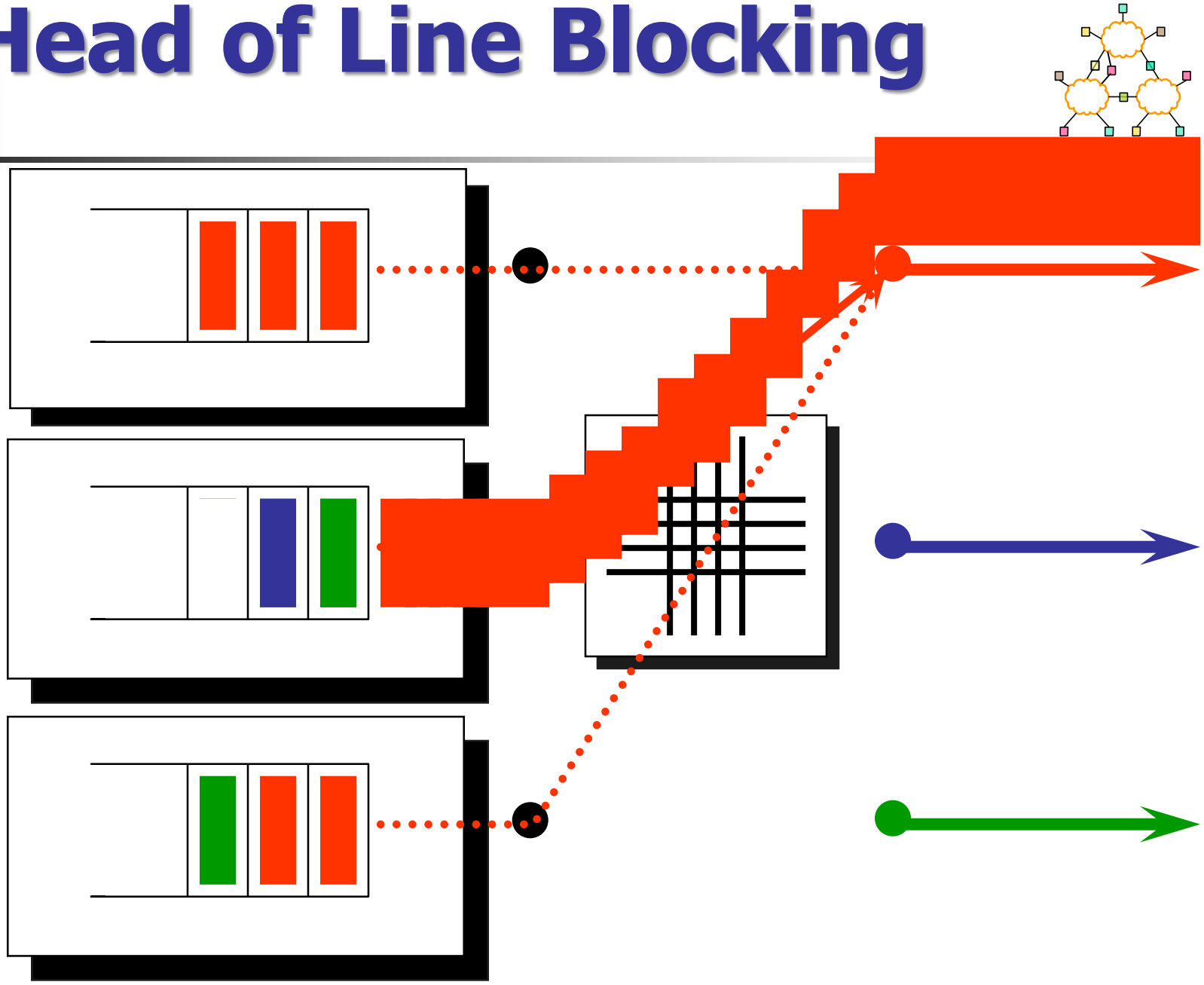


Input Queuing

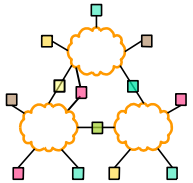
Head of Line Blocking

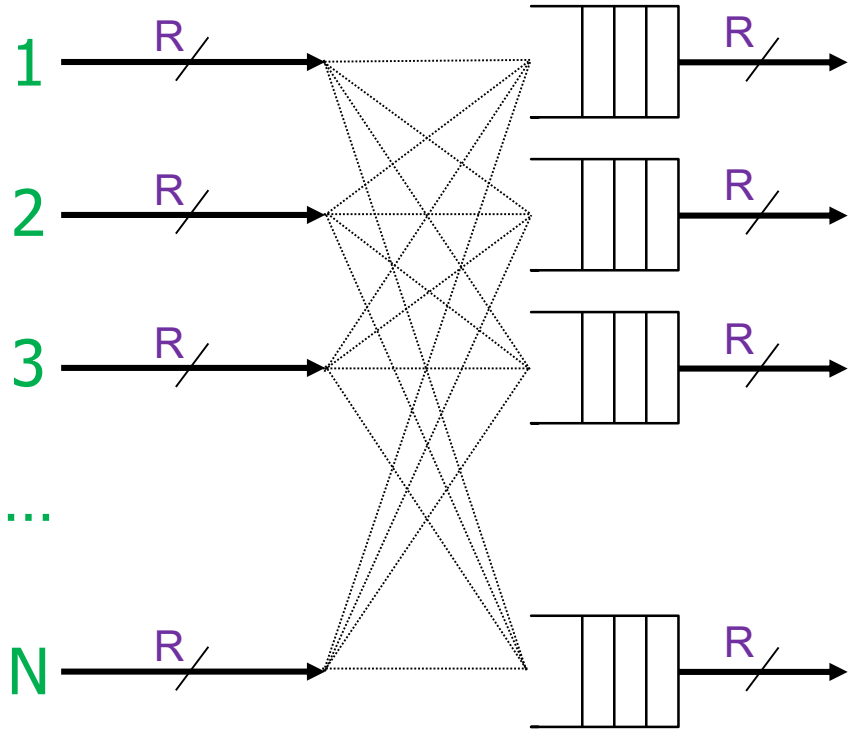


Head of Line Blocking



An input-queued (IQ) switch





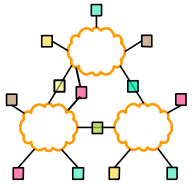
Properties of an IQ switch

- All buffering takes place at the input.
- Input queues only need to be able to write packets at rate R (instead of N x R).

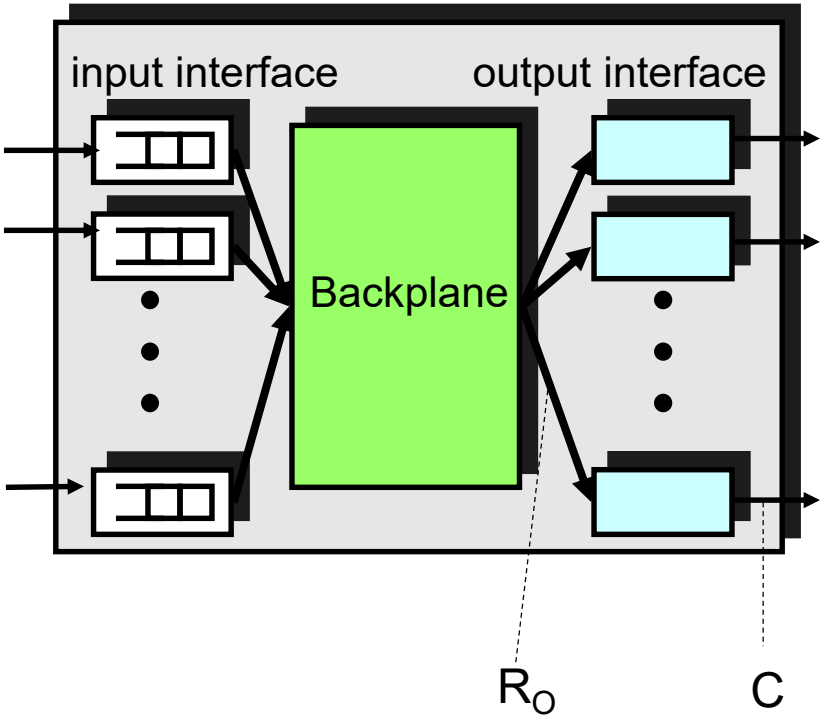
Consequences

- Can build a switch N times faster.
- But, a packet can be held up by packet ahead destined to a different output.
- Hence an IQ switch is not "work conserving". It can unnecessarily idle.
- May not achieve "100% throughput".
- Average delay is not minimized.

Input Queueing (IQ)

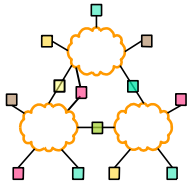


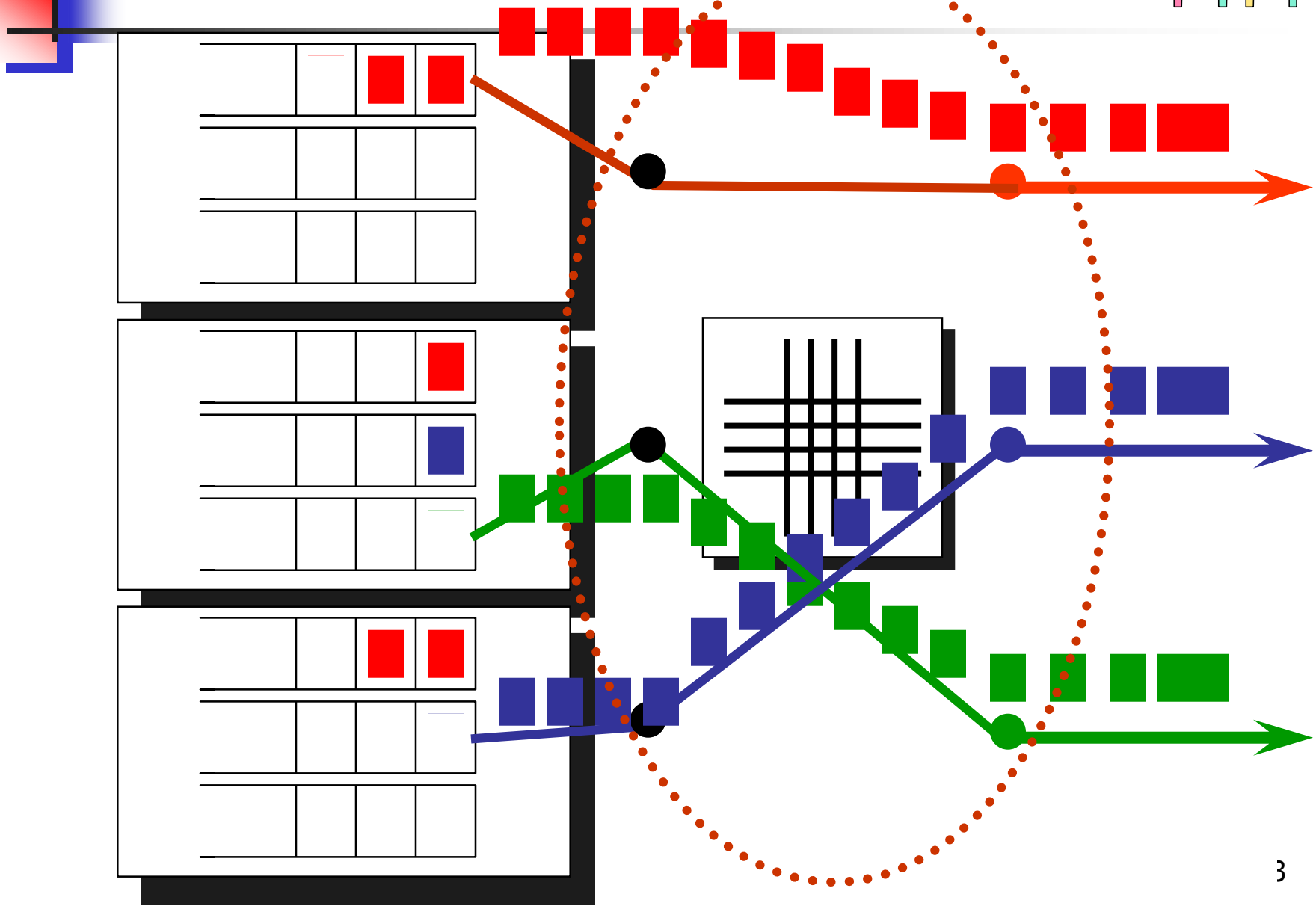
- Advantages
  - Easy to built
    - Store packets at inputs if contention at outputs
  - Relatively easy algorithm
    - Only one congestion point, but not output…
    - need to implement backpressure



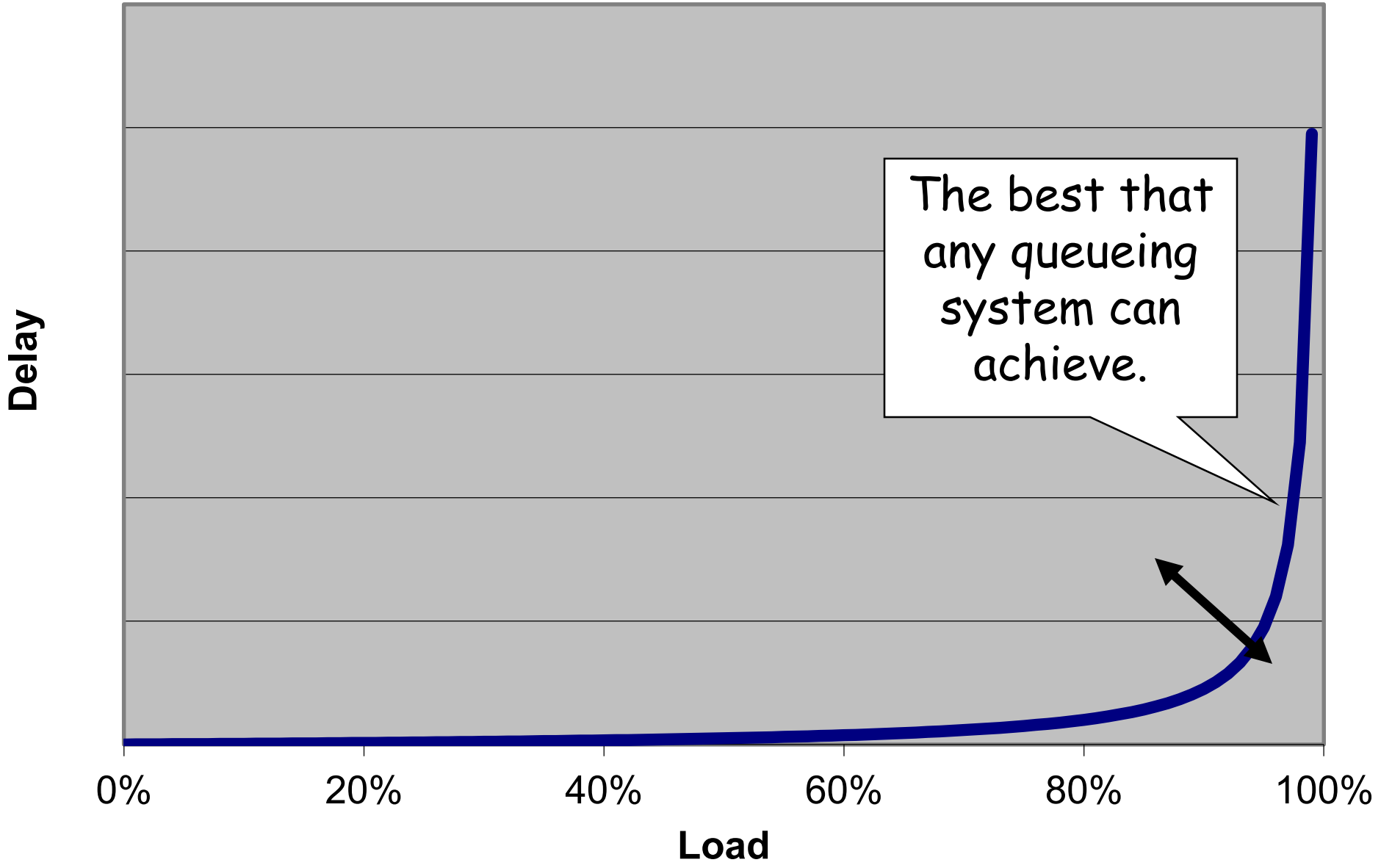
- Disadvantages
  - Hard to achieve %100 utilization (due to output contention, head-of-line blocking)
    - However, theoretical and simulation results show that for realistic traffic an input/output speedup of 2 is enough to achieve utilizations close to 1

Virtual Output Queues

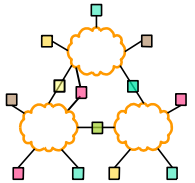


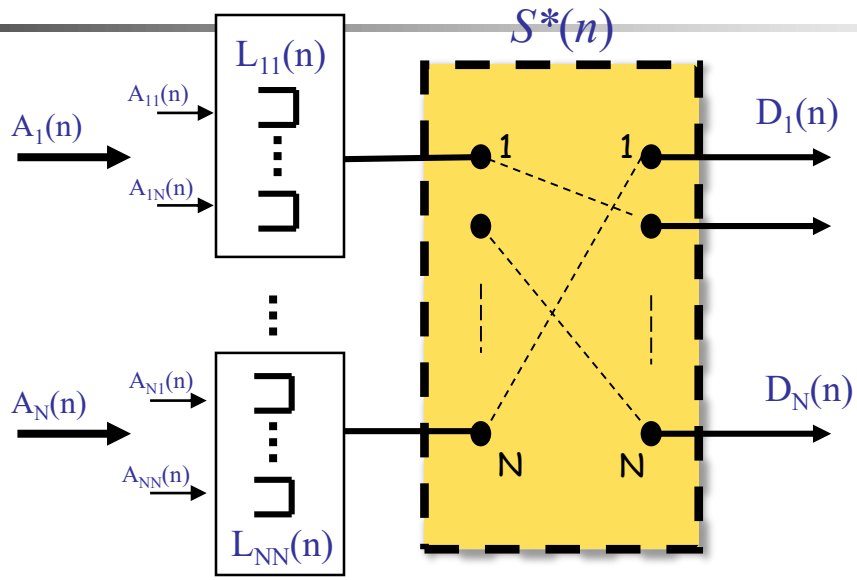


with Virtual Output Queues

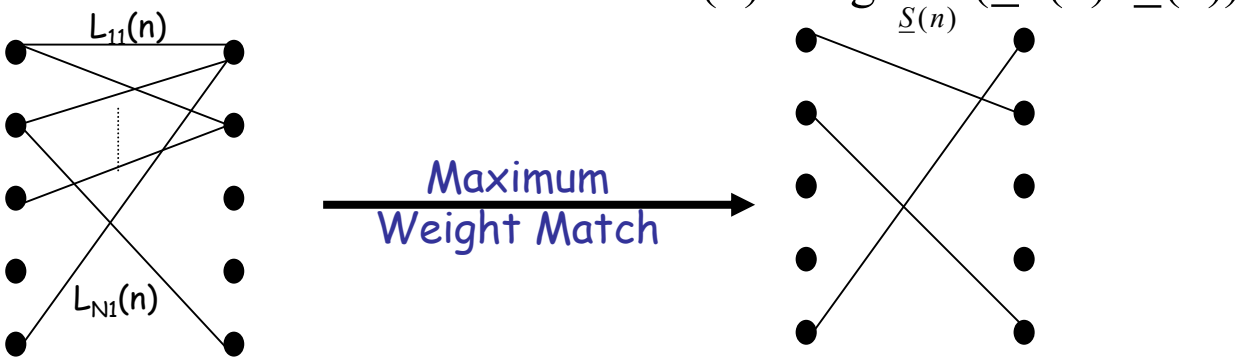


Maximum Weight Matching





$$S^*(n) = \arg\max_{\underline{S}(n)} (\underline{L}^T(n) \cdot \underline{S}(n))$$

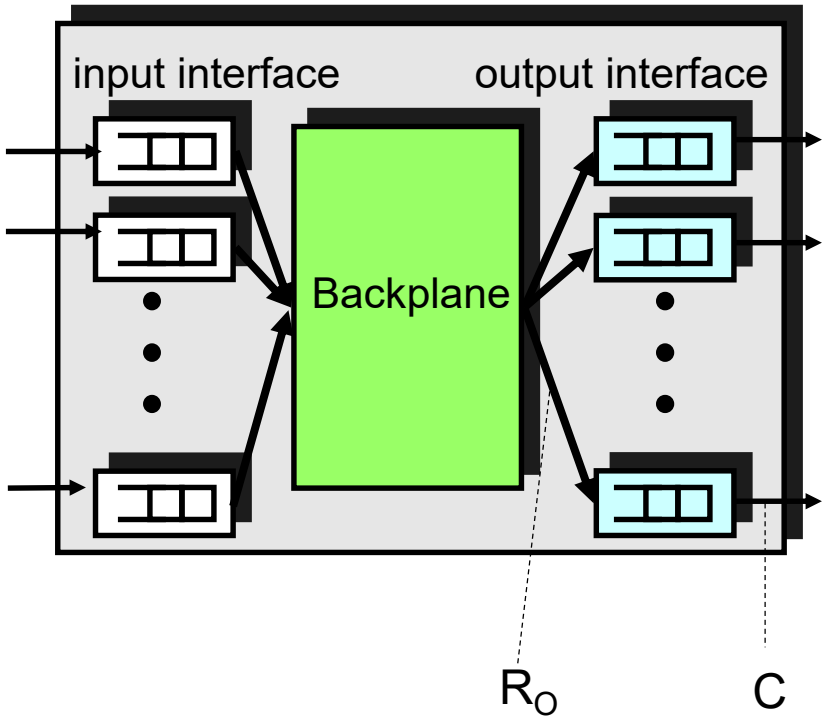


"Request" Graph

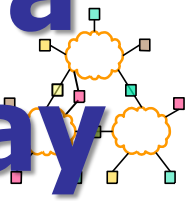
Bipartite Match

Combined Input-Output Queueing (CIOQ) Routers

- Both input and output store packets
- Advantages
  - Easy to achieve higher performance
    - Utilization 1 can be achieved with limited input/output speedup (<= 2)
- Disadvantages
  - Harder design algorithms
    - Two congestion points
    - Need to design flow control
  - Note: recent results show that with a input/output speedup of 2, a CIOQ can emulate any work-

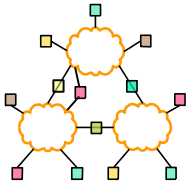


Generic Architecture of a High Speed Router Today

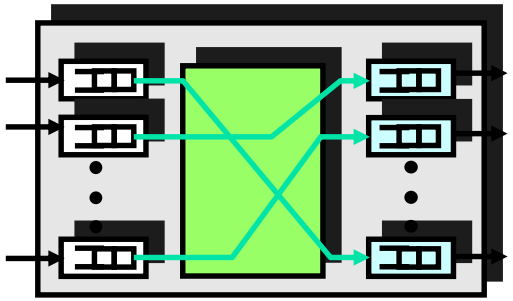


- Combined Input-Output Queued Architecture
  - Input/output speedup <= 2
- Input interface
  - Perform packet forwarding (and classification)
- Output interface
  - Perform packet (classification and) scheduling
- Backplane
  - Point-to-point (switched) bus; speedup N
  - Schedule packet transfer from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

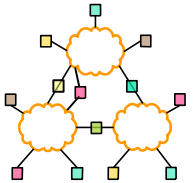
Backplane



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

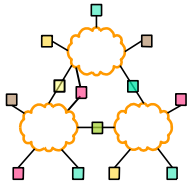


Cell transfer

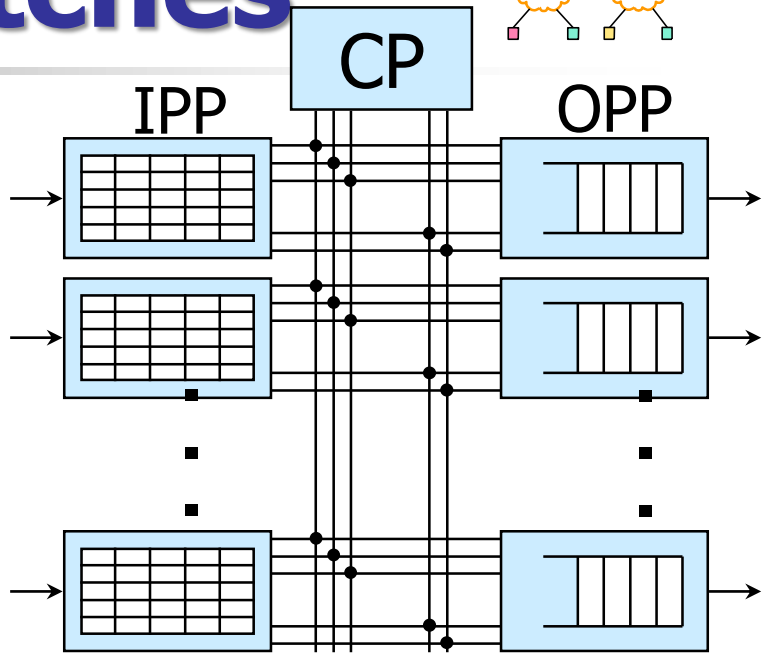


- Schedule:
  - Ideally: find the maximum number of input-output pairs:
    - Resolve input/output contentions
    - Avoid packet drops at outputs
    - Packets meet their time constraints (e.g., deadlines), if any
- Example
  - Assign cell preferences at inputs, e.g., their position in the input queue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

Bus-Based Switches

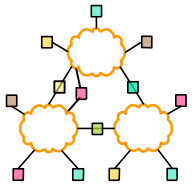


- *Input Port Processors* (IPP) do:
  - lookup, sync and format conversion and put cells on bus
- *Output Port Processors* (OPP)
  - Buffer cells awaiting transmission
- *Control Processor* (CP)
  - exchanges control messages with terminals and other CPs
  - configures connections by writing in IPP routing tables

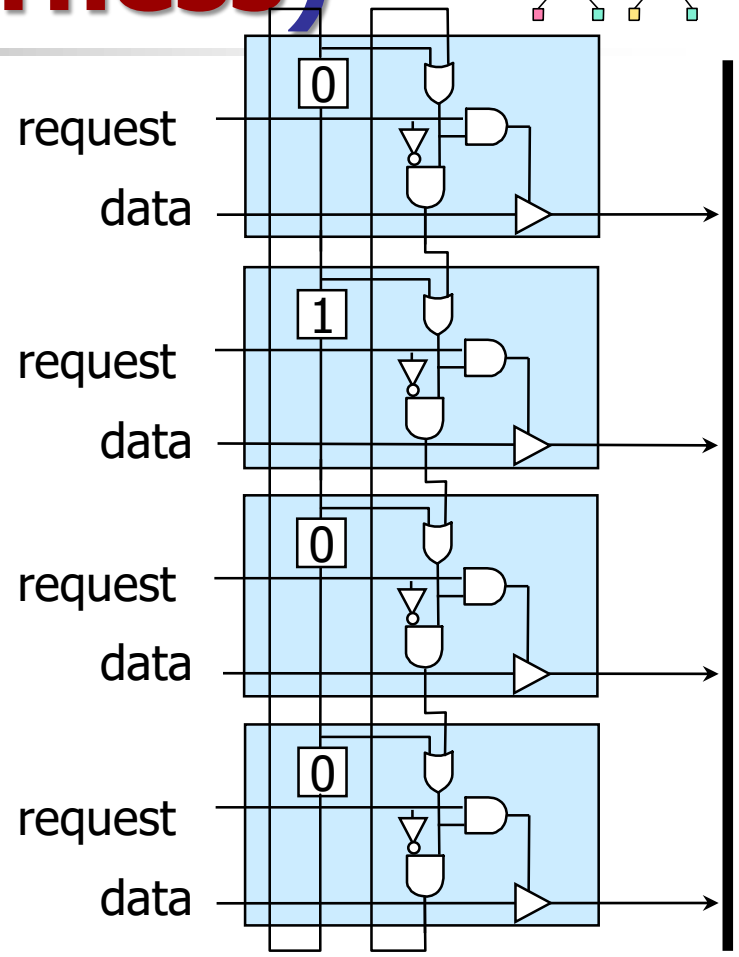


- Common bus interconnects various components.
  - » Requires bandwidth equal to sum of external link bandwidths;
  - » bus width must increase with number of links (scale?)
    - » **capacitive loading generally reduces clock rate as number of links grows**
- Problems?
  - Scale? Usually $O(n^2)$
  - OPP should operate with the rate of bus.
  - Fairness?

Bus Arbitration: Rotating Daisy Chain (Fairness)

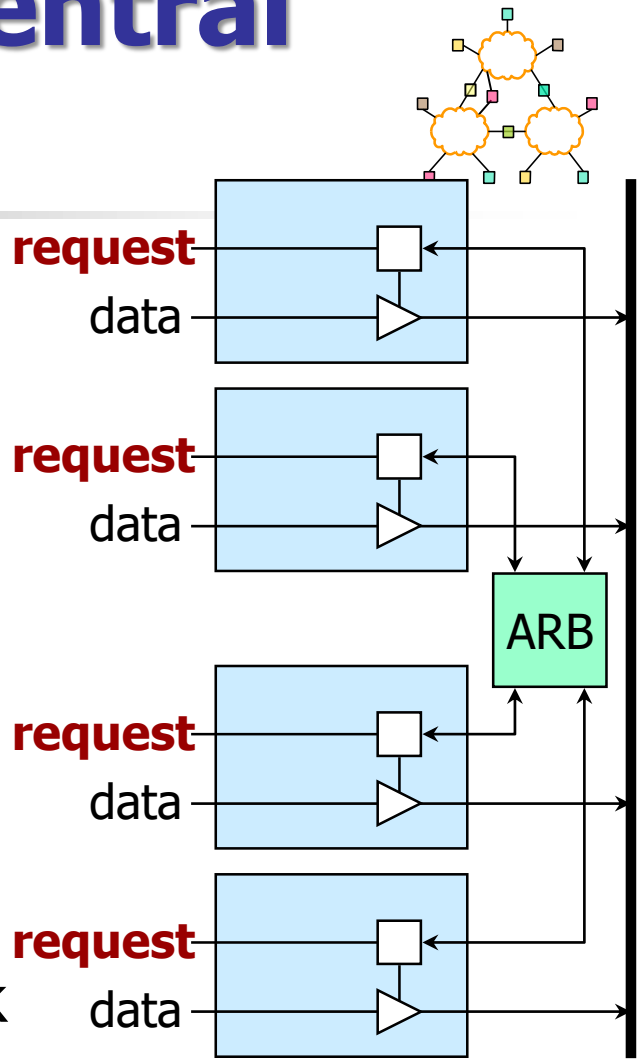


- In systems where link bandwidth is larger than bus bandwidth, need mechanism to regulate bus access
- Rotating *token* eliminates positional favoritism of static daisy chain
- Not always fair
  - » if two consecutive inputs are competing "second" one gets fewer bus cycles
- For "fair-sharing"
  - » advance token to winning IPP
- » **Limited functionality!**



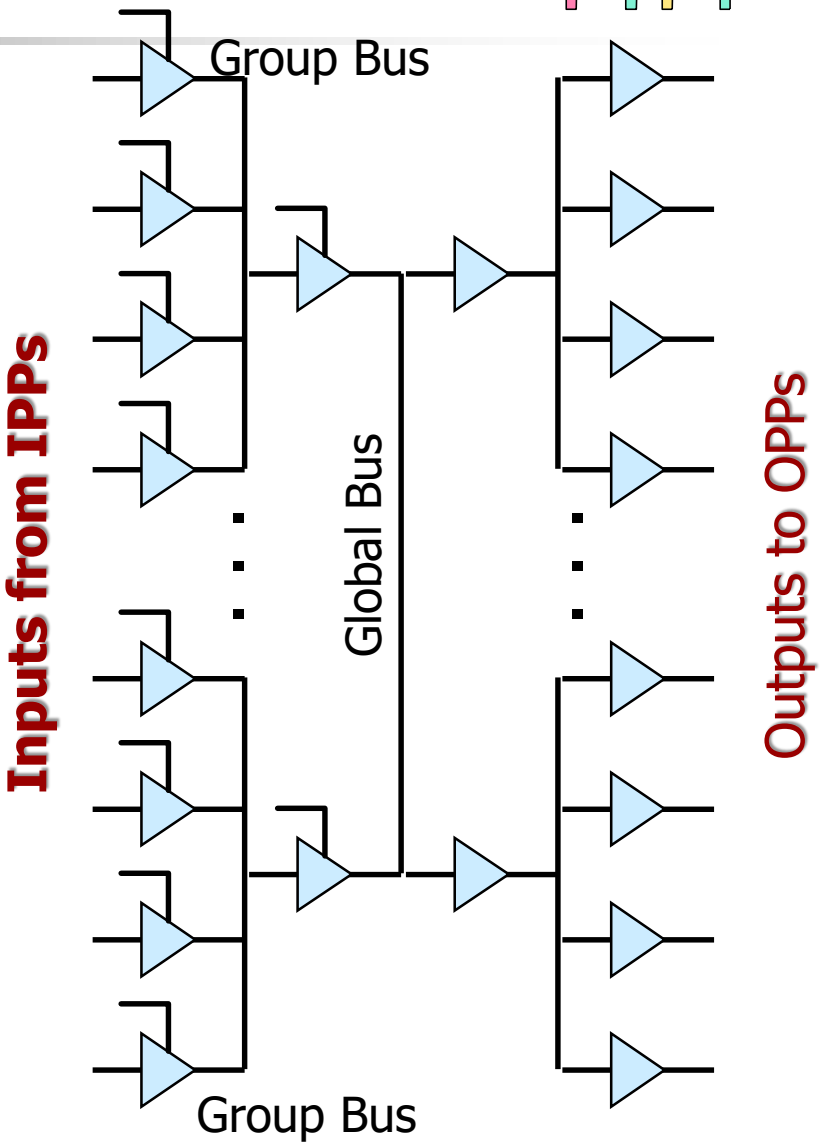
Bus Arbitration: Central Arbiter

- IPPs send requests to central arbiter which picks winner.
- Requests may include priority.
  - static "importance" of data to be sent
  - waiting time of data
  - length of IPP queue
  - combination
- Arbiter can become a performance bottleneck.
- Distributed version eliminates bottleneck
  - uses bit-serial maximum computation to select highest priority value

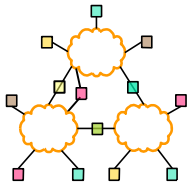


Two Level Bus Design(scale)

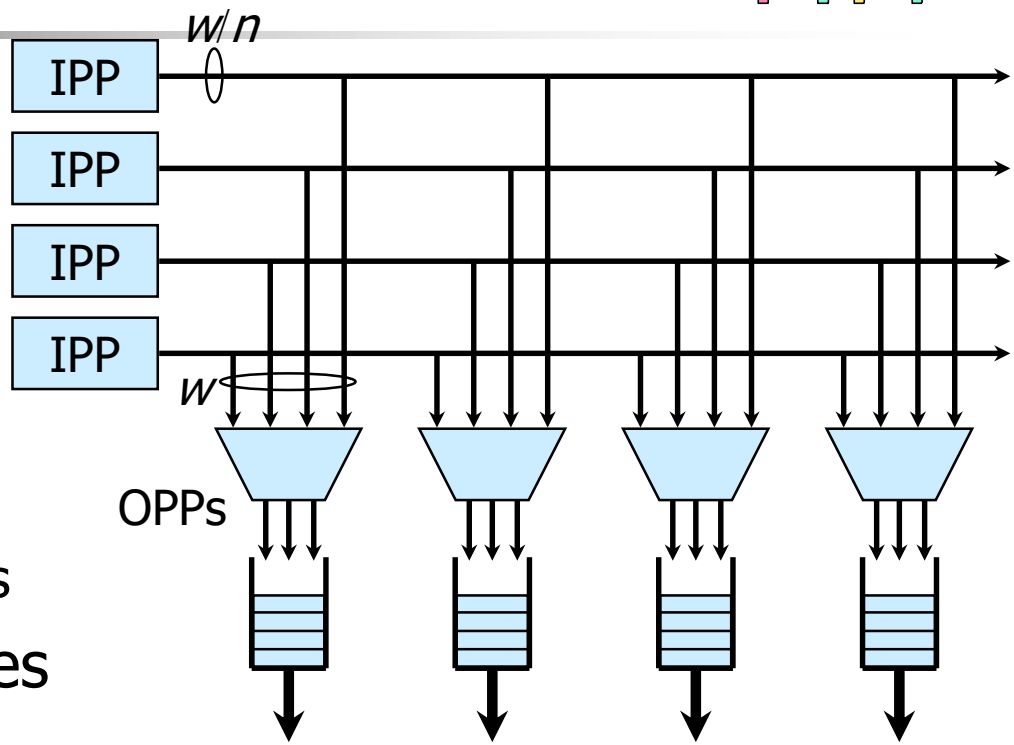
- In simple bus, every driver and receiver contributes to capacitive loading.
  - increases rise and fall times of signals, limiting frequency.
- In two level bus, lower capacitive loads per segment.
- Adds control complication and delay.
- Can be extended to more than two levels, but diminishing returns.



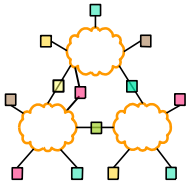
Subdivided Bus with Knockout Concentrators



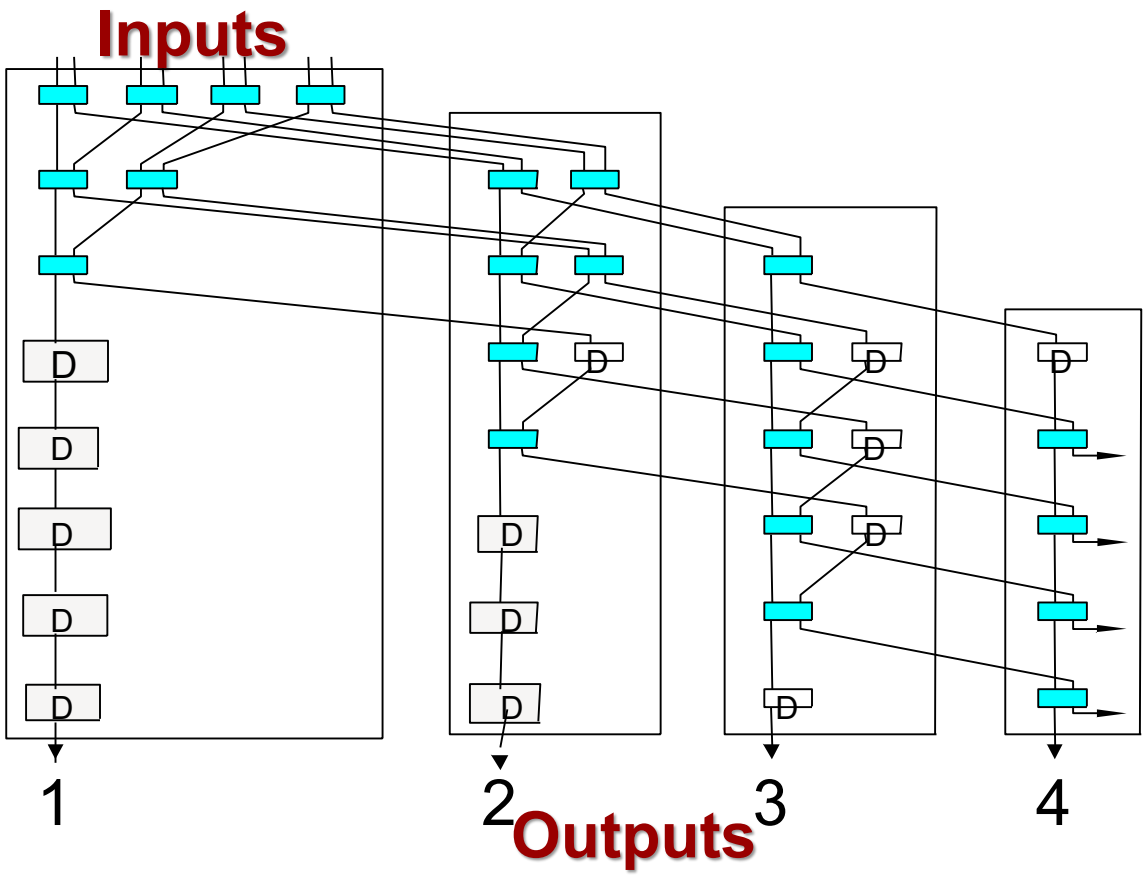
- Split bus into n "minibuses" with w/n wires each
- Each minibus driven by just one IPP.
  - cuts capacitive loading in half
  - adding fanout components allows higher clock frequencies
- OPPs concentrate n minibuses onto $L<n$ outputs (optional)
- OPPs must each be able to buffer up to L cells in parallel
- Parallel reception complicates control somewhat
- Concentration reduces required OPP memory bandwidth



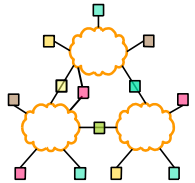
Knockout Switch



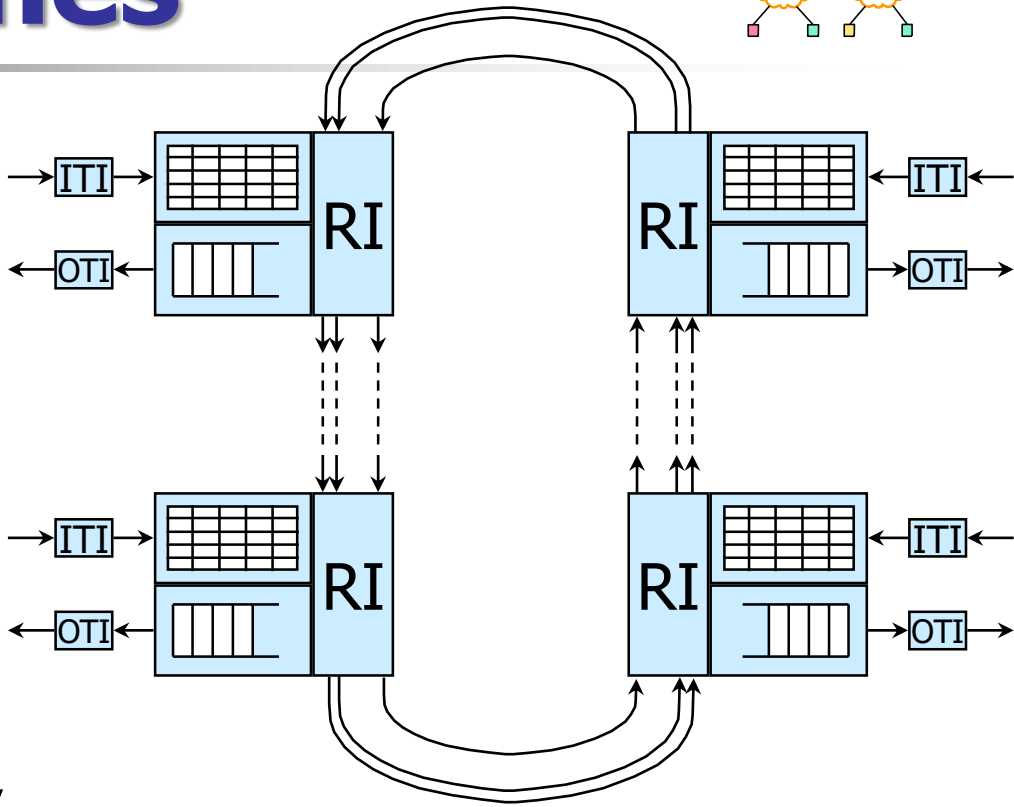
- Concentrator
  - select l of n packets
- Complexity: o(n^2)



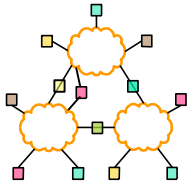
Ring Switches



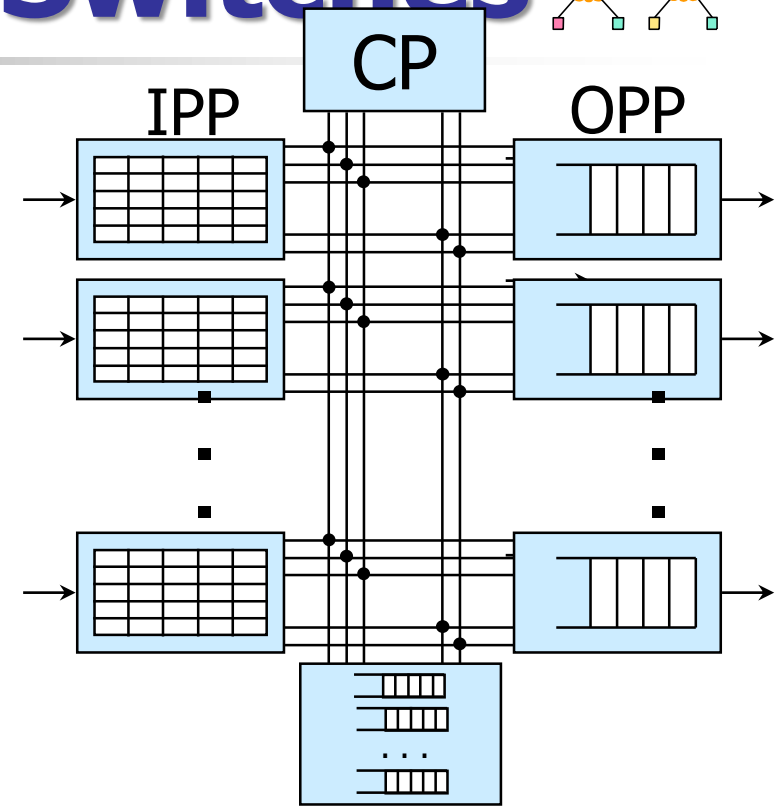
- *Ring Interfaces* (RI) connect IPP and OPP to ring.
- Ring avoids capacitive loading of bus, allowing higher clock frequencies.
- Latency of 1-4 clock ticks per RI.
- Same overall bandwidth requirements and complexity characteristics as bus.
- Common control mechanisms
  - token passing
  - slotted ring with busy bit
  - delay-insertion



Shared Buffer Switches



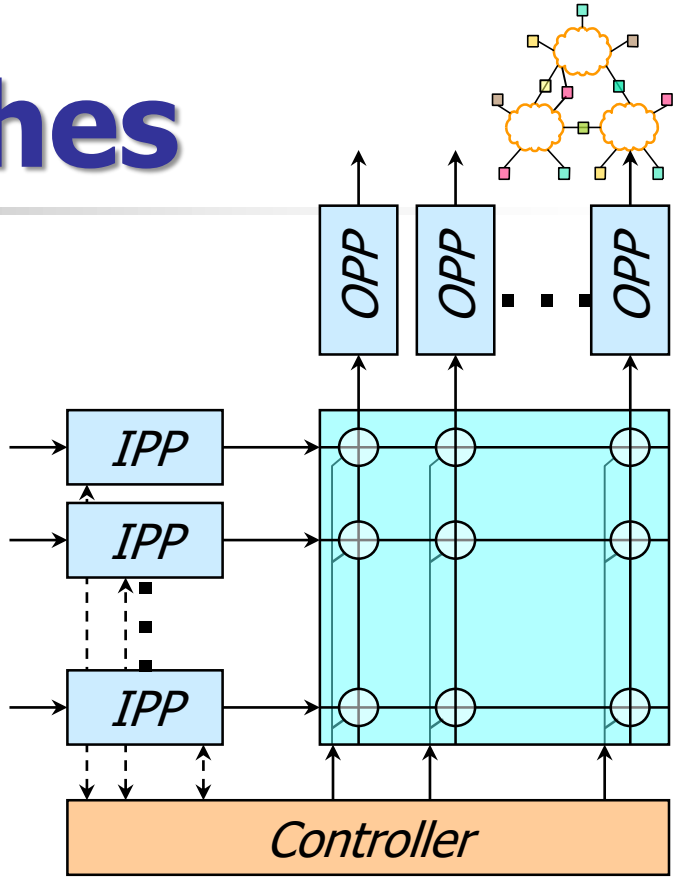
- Queues are rarely full, then, memory for queues is rarely fully used.
- Share memory among all queues.
- Requires a central memory with bandwidth equal to twice the external link bandwidth. Bus bandwidth must also be doubled.
- Per output or per flow queues typically implemented as linked lists.



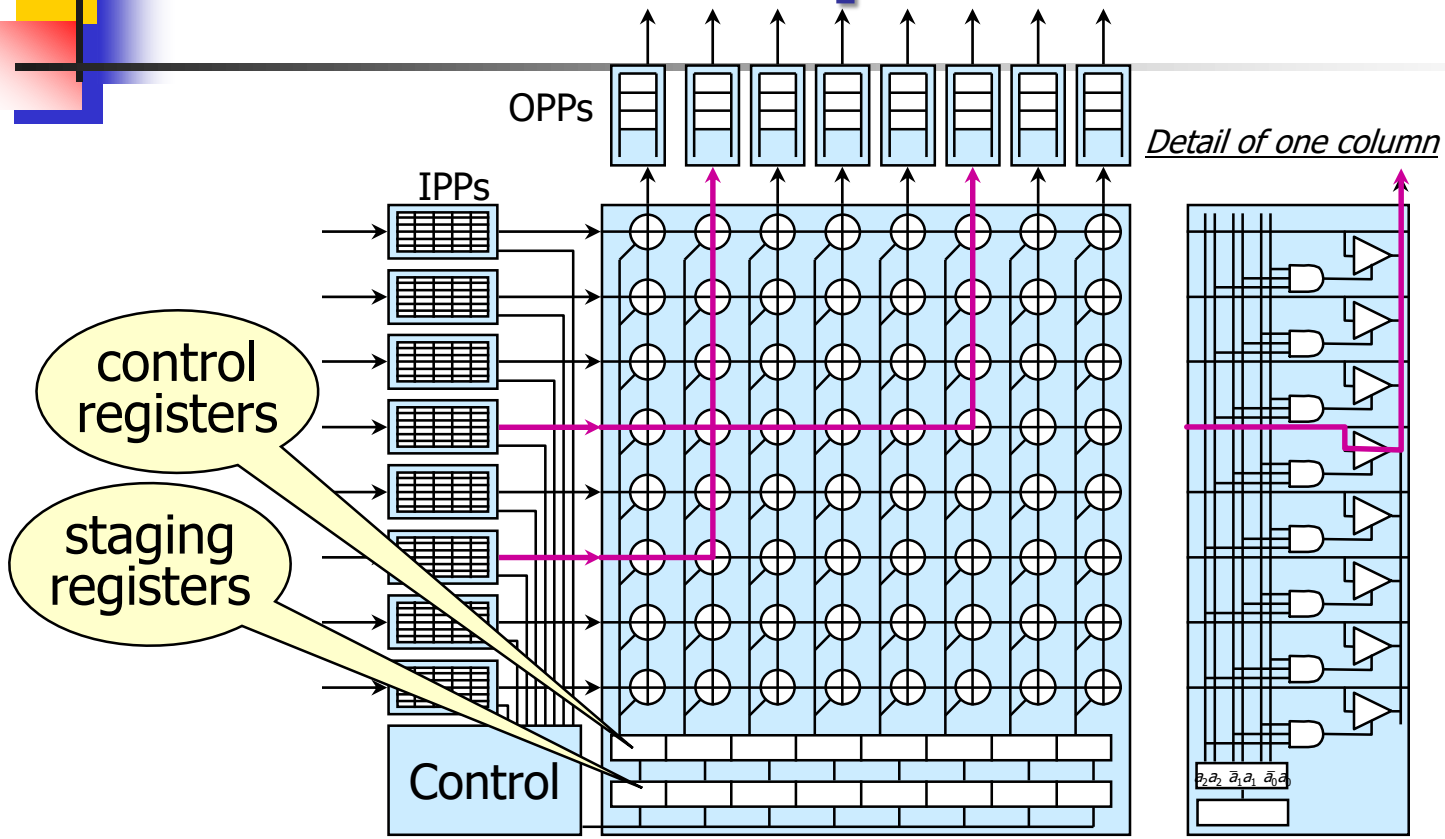
- Size of Memory can be estimated by taking the *convolution* of the queue length distributions of individual output queues.
- For switches with 10 or more links, can reduce required memory by up to an order of magnitude.
- Depends on independence of queuing processes at different outputs.

Crossbar Switches

- Crossbar allows multiple cells to pass in parallel to distinct outputs
  - » use of point-to-point transmission cuts capacitive loading at circuit board level
  - » parallelism allows smaller data path widths at IPPs, OPPs
- Control circuit arbitrates access to outputs
  - » for each output can use any standard bus arbitration mechanism
    - – daisy-chain with optional rotating starting point
    - – dynamic priority arbitration mechanism
  - » alternative approach is time-slotted arbitration ring
- Retains quadratic complexity, but concentrates it within chip or chip set, reducing impact on system cost

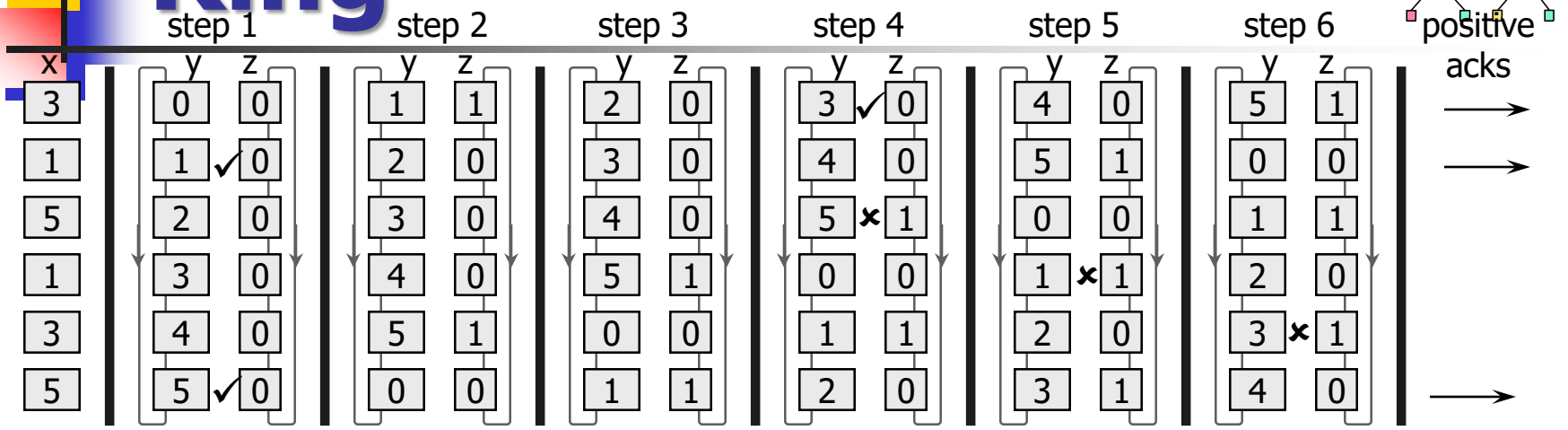


Crossbar Implementation



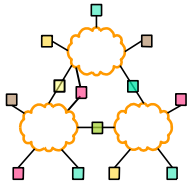
- Control registers specify connected input.
- Crosspoints decode specified "row number".
- New values are pre-loaded into staging registers.
- One input can connect to multiple outputs.

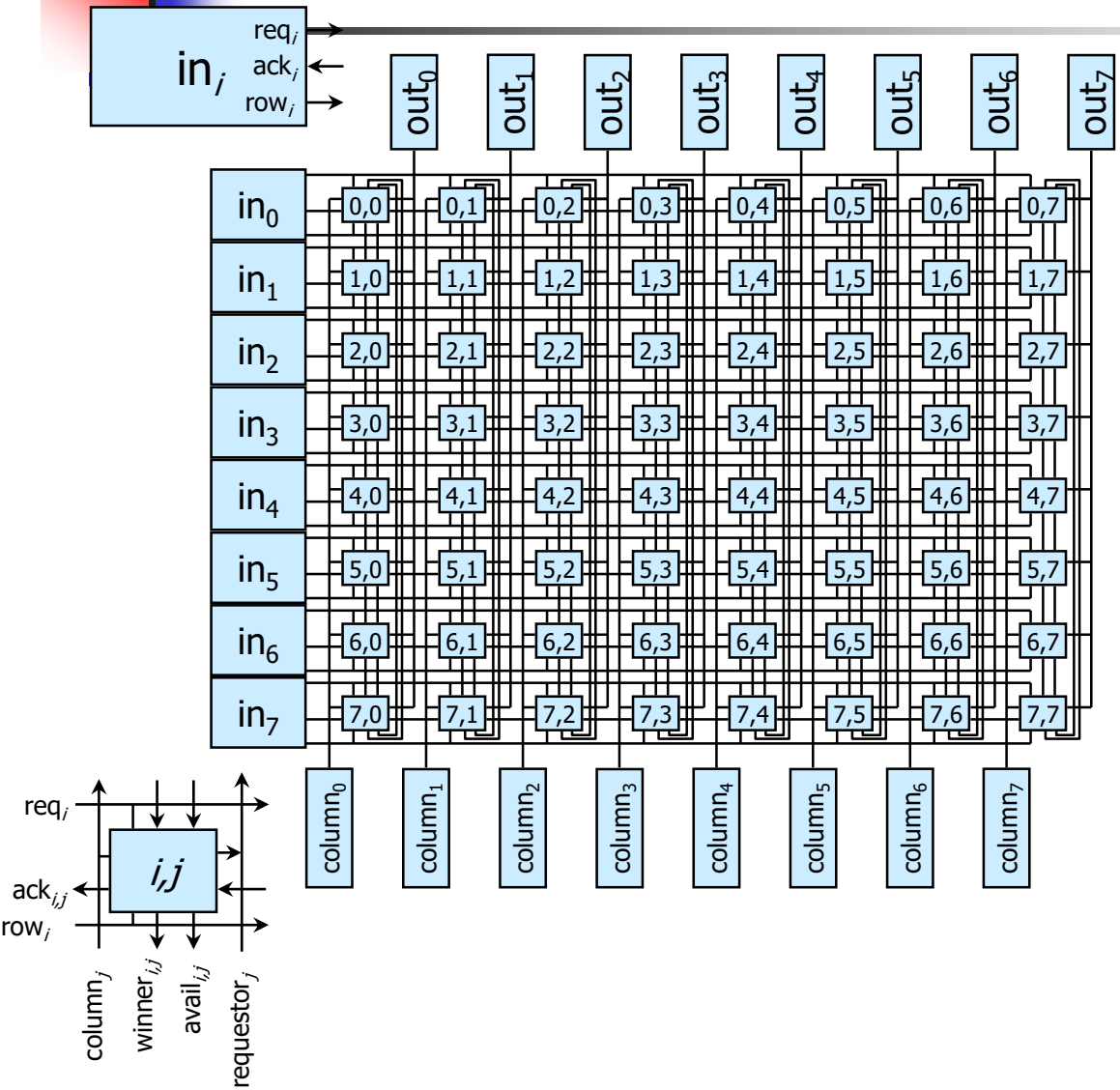
Time Slotted Arbitration Ring



- For each input, there is a register x_i containing desired output, a register y_i and a flip flop z_i, linked in circular shift register; initially $y_i = i$, $z_i = 0$.
- Shift register shifts between each arbitration step, so y_i , z_i are assigned previous values of y_{i+1}, z_{i+1}.
- During arbitration step, x_i and y_i are compared; if equal and if $z_i = 0$, then waiting cell has "won" and z_i is set.
- Make accessibility even by initializing y_i to i+offset at start of each cycle, where offset is incremented by 1 modulo n before first step.
- Arbitration time is proportional to n.

Scalable Crossbar Controller

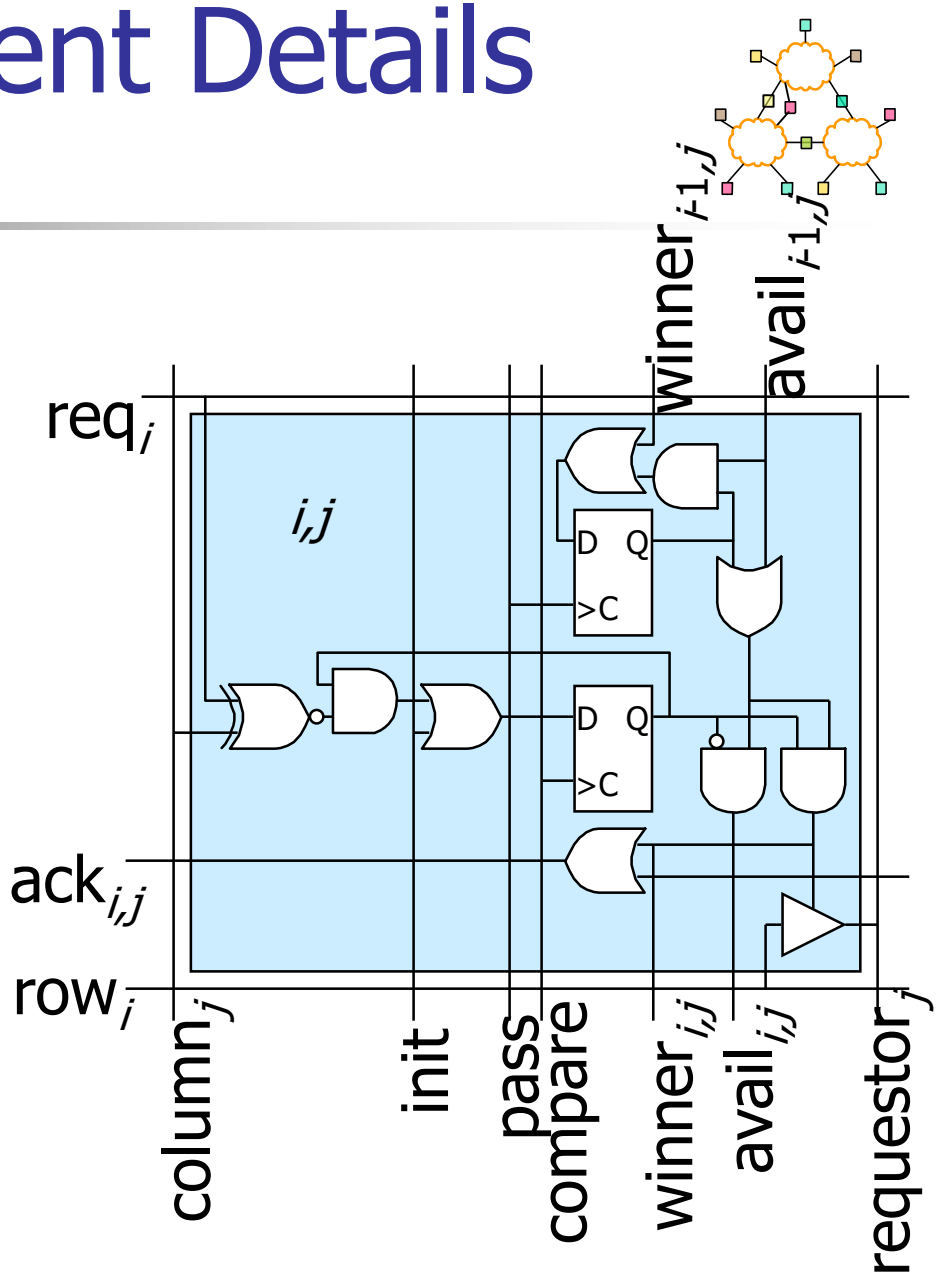




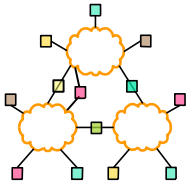
- Inputs send requests bit-serially along rows.
- *Arbitration Elements* compare request bits to column number bits.
- One AE per column holds *token* for rotating daisy-chain arbitration.
- Token passed to AE following winner at end of cycle.
- Approximately 25 gate-equivalents per AE.
  - 64 port crossbar controller requires 100K gates

Arbitration Element Details

- Upper flip flop holds *token* for arbitration process.
  - passed at end of cycle to AE following winner
  - if no winner in previous cycle, stays in same location
- Lower flip flop & associated gates implement serial comparator.
  - flip flop initialized to 1 and cleared on first mismatched bit
- Acknowledgement returned to input, after arbitration.
- Input passes row number through enabled tri-state driver to output.



Output Contention in Crossbars



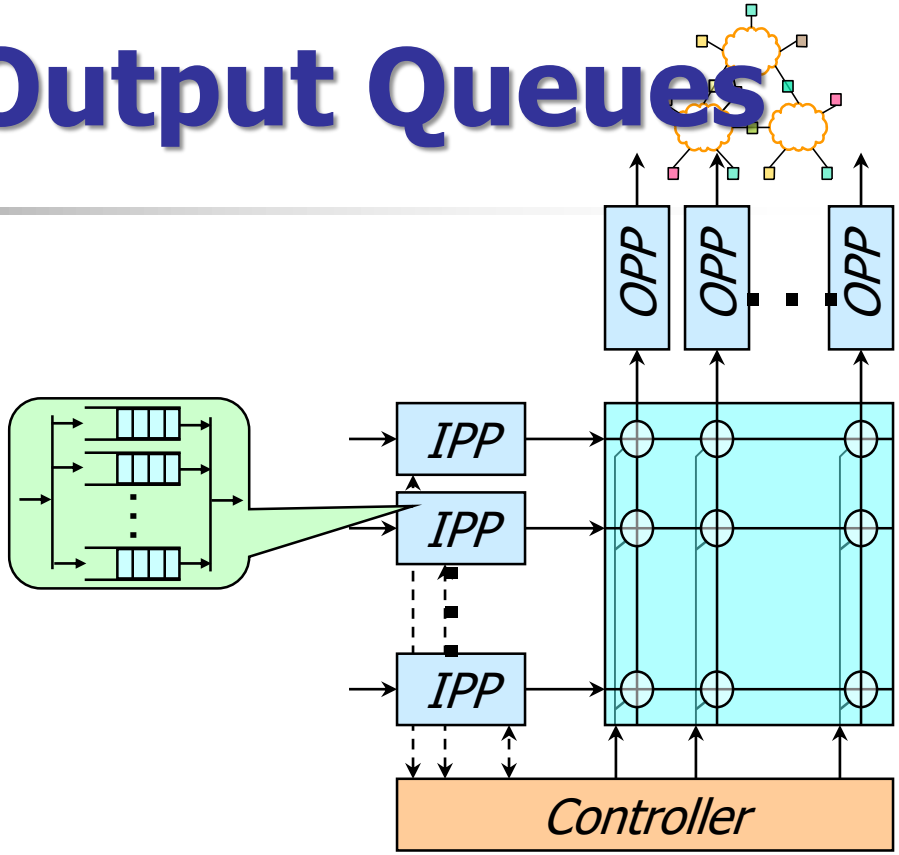
- Different inputs can compete for one output.
- Can alleviate by increasing crossbar bandwidth
- Simple analysis
  - assume each crossbar input has cell with probability p, each input cell is independent and output addresses are equiprobable.
  - for any given output, probability that i cells are addressed to it is

$$\binom{n}{i}(p/n)^i(1-p/n)^{n-i}$$

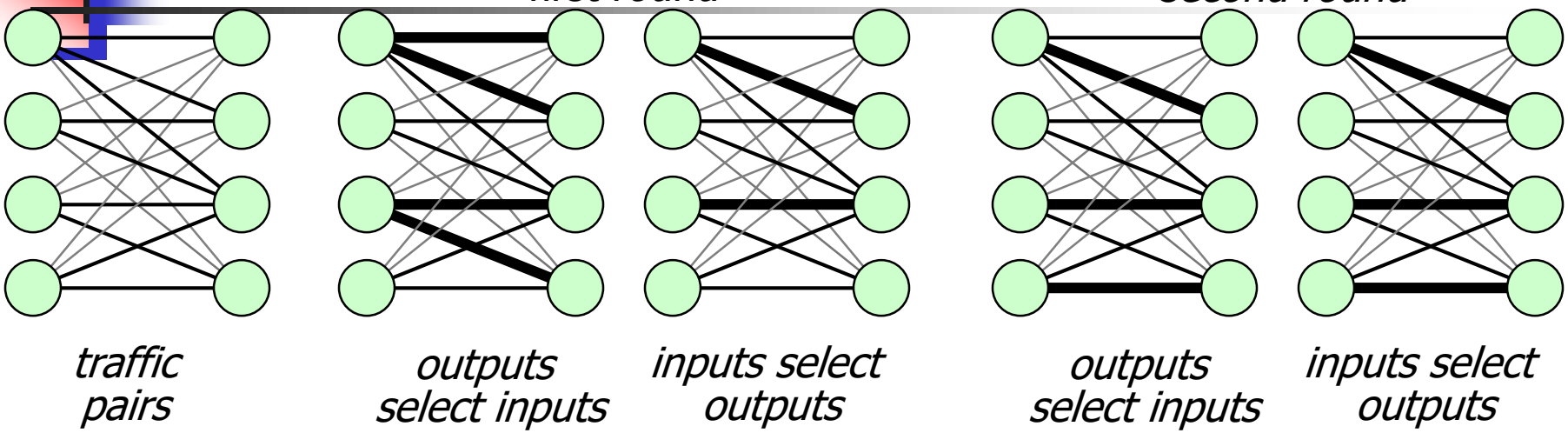
  - » on average, there are $n(1-p/n)^n \approx ne^{-p}$ outputs for which there are no cells.
  - » so, out of expected pn input cells, $\approx (1-e^{-p})n$ leave.
  - » for $p=1$, about $.63n$ can leave, implying limit on usable crossbar capacity.
- More precise analysis shows that for large n, maximum crossbar throughput closer to $.58n$ cells per cycle.

Crossbar, Virtual Output Queues

- Separate *virtual output queues* at each input.
  - queues implemented using linked lists in common memory
- Controller seeks to match inputs to outputs.
  - keep outputs busy
  - emulate queuing behavior of "ideal switch"
  - best algorithms work well for arbitrary input traffic
  - some speedup still needed
- Does not extend readily to multicast.
  - cannot associate cell with single VOQ

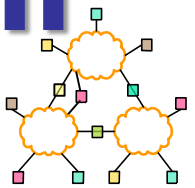


Iterative Matching with Random Selection

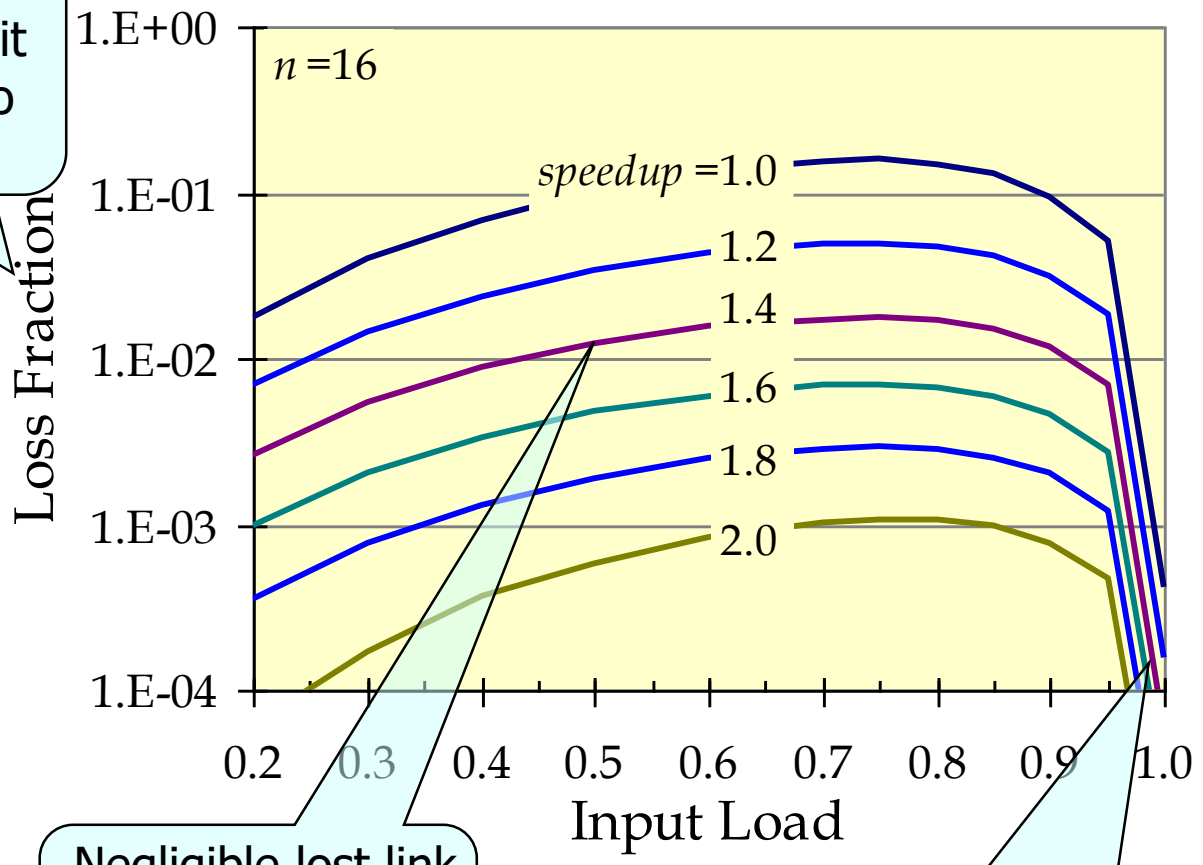


- While there is an "unscheduled output" with waiting cells,
  - for each unscheduled output with waiting cells, randomly select an input with cells to send
  - for each selected input, randomly pick one of the selecting outputs
    - call these (input,output) pairs, *matched pairs*
- May produce as few as half the ideal number of matched pairs.

Performance on Random Traffic



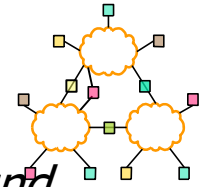
Ratio of number of "lost" transmit opportunities to input load

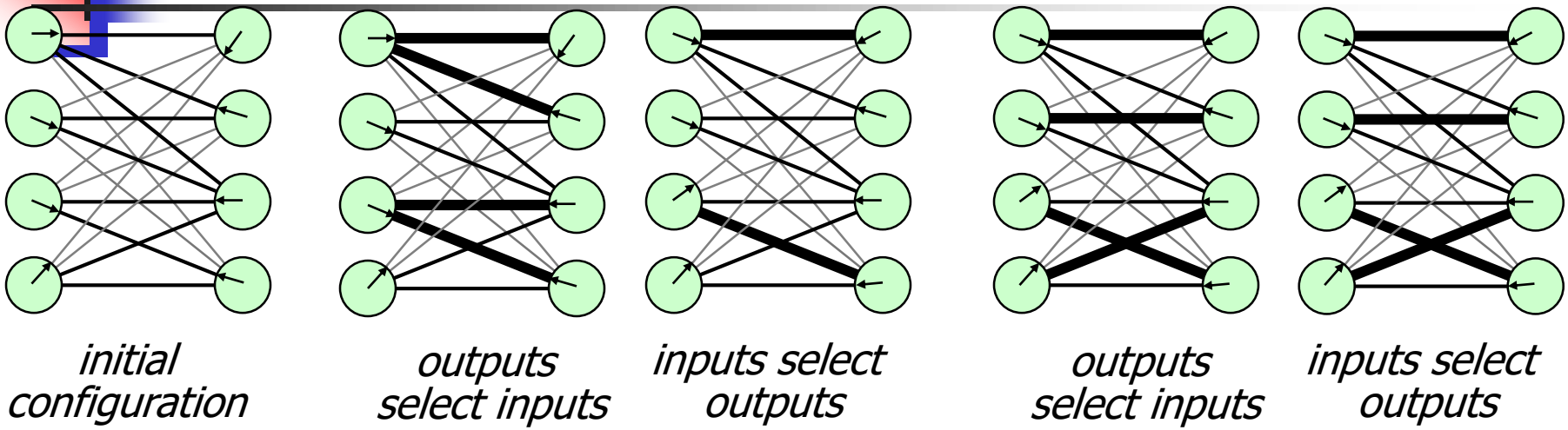


Negligible lost link capacity for speedups >1.5

Loss drops at heavy loads, since output queues rarely empty.

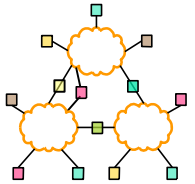
Iterative Round Robin with iSLIP



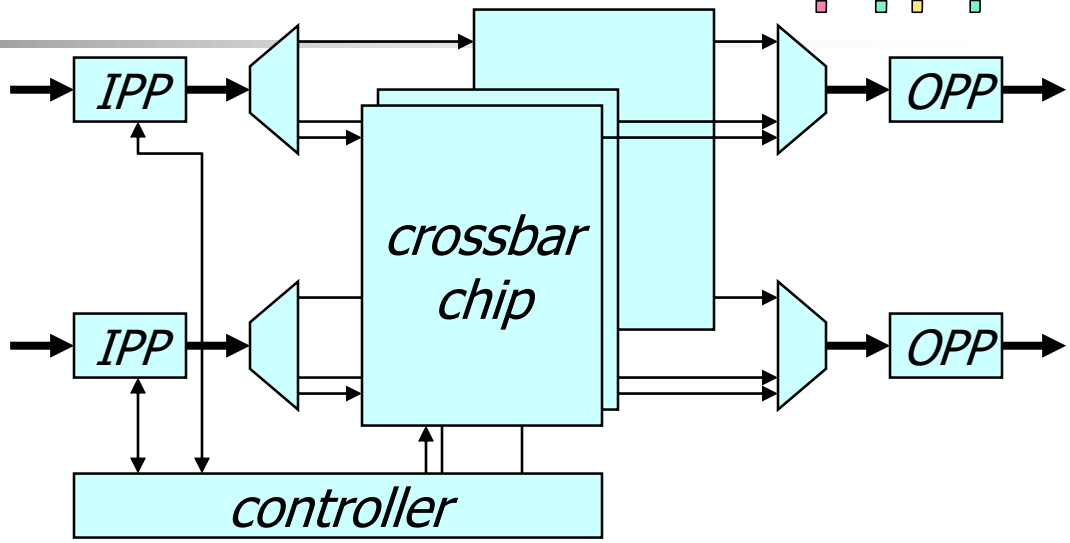


- *i*-SLIP algorithm seeks to desynchronize priorities.
  - update priorities only for those outputs whose selections are "confirmed" in second half of round
  - update priorities only in first round
  - provides good performance for random traffic with minimal speedup
  - works well even with limit on iterations

High Performance Crossbars

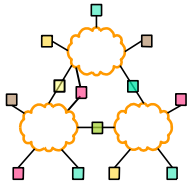


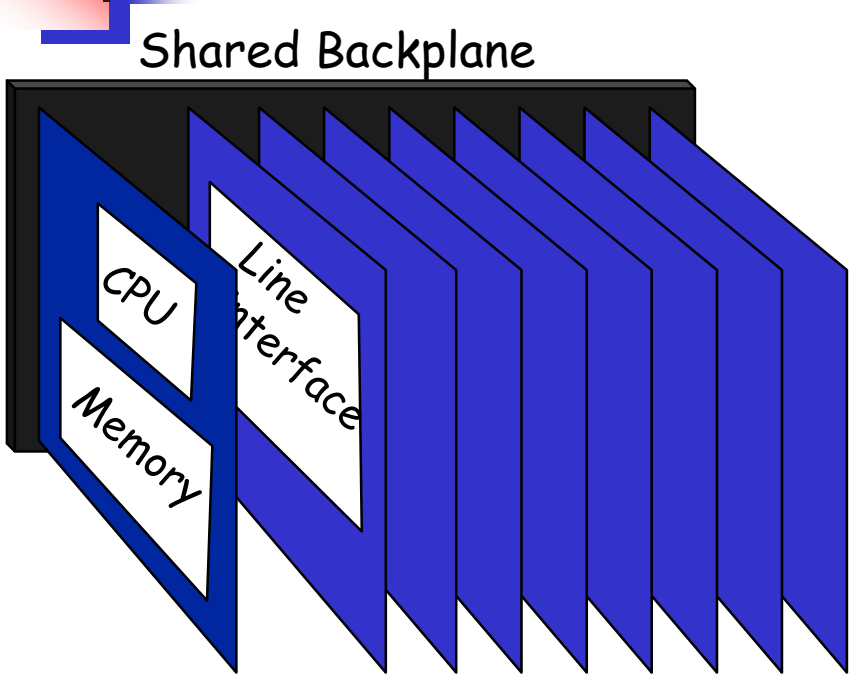
- Need parallelism for high performance systems.
  - 32×10G system needs total bandwidth over 1 Tb/s
  - single ICs now limited to under under 100 Gb/s
    - 32 gigabit serial links at 2.5 Gb/s each

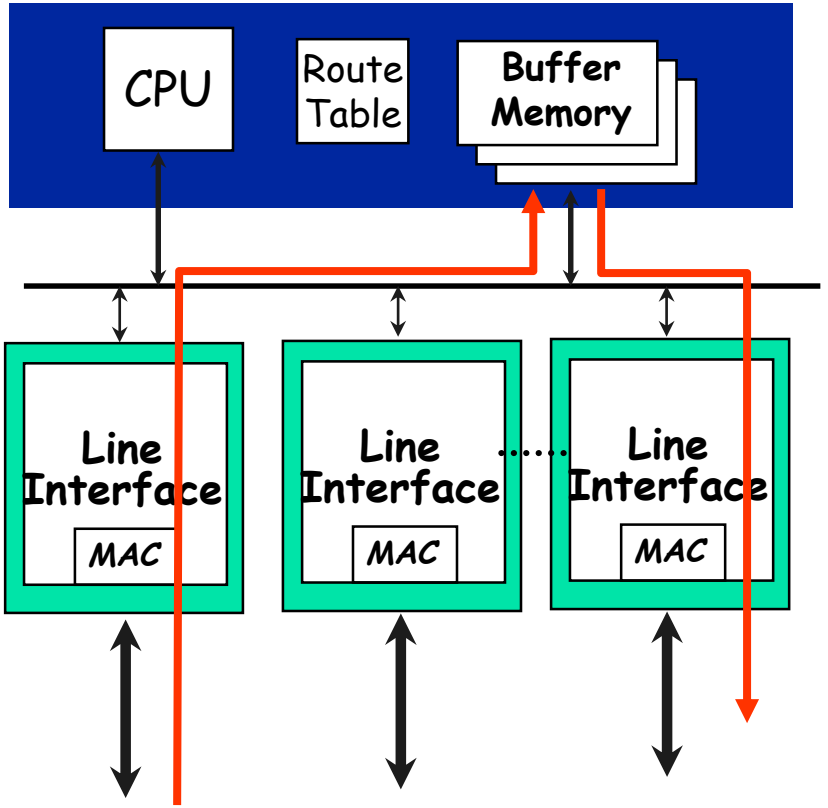


- "Bit-sliced" design makes best use of IC bandwidth.
  - » mux/demux convert single fast datapath to/from many small ones
    - » divides each cell into smaller pieces, or
    - » send each cell through just one crossbar - allows control to be parallelized too, but may get cells out of order
- Can also build large crossbars by *tiling* smaller components.
  - » leads to complex and high cost systems (relative to alternatives)
  - » often done, but almost never a good idea

First Generation Routers

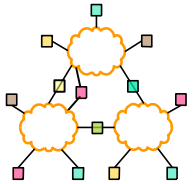


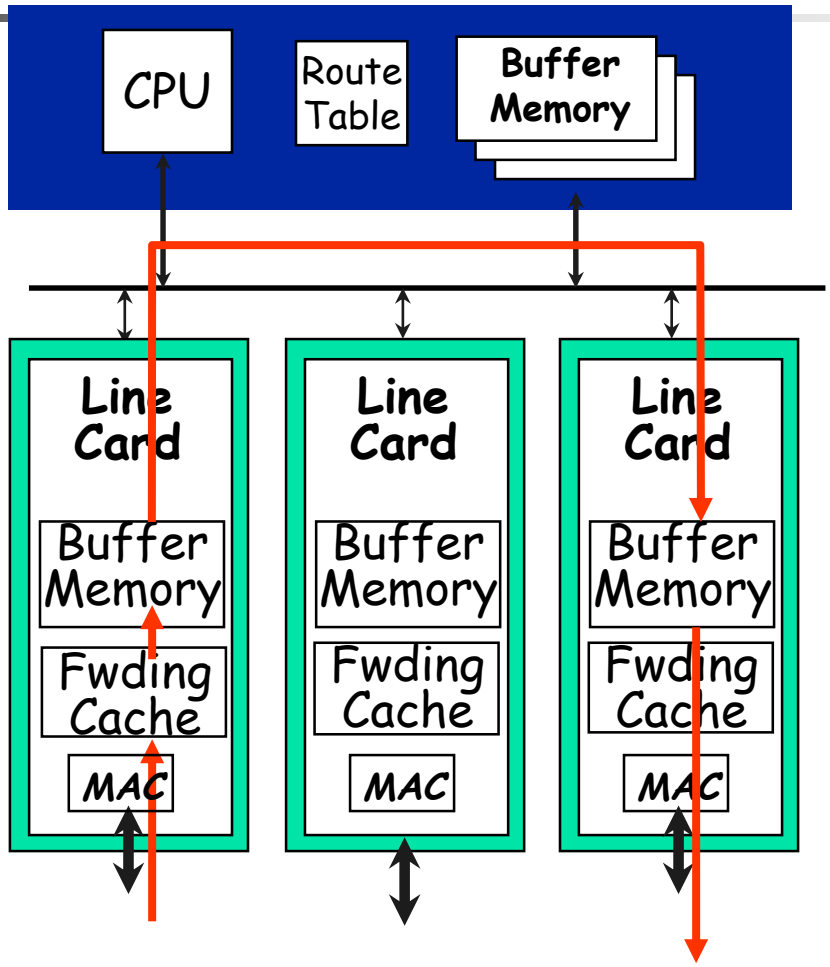




Typically <20 Gb/s aggregate capacity

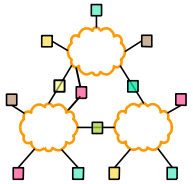
Second Generation Routers



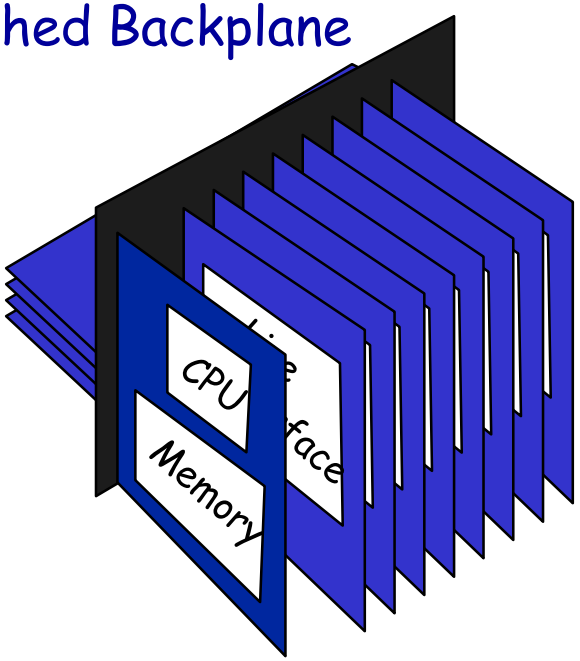


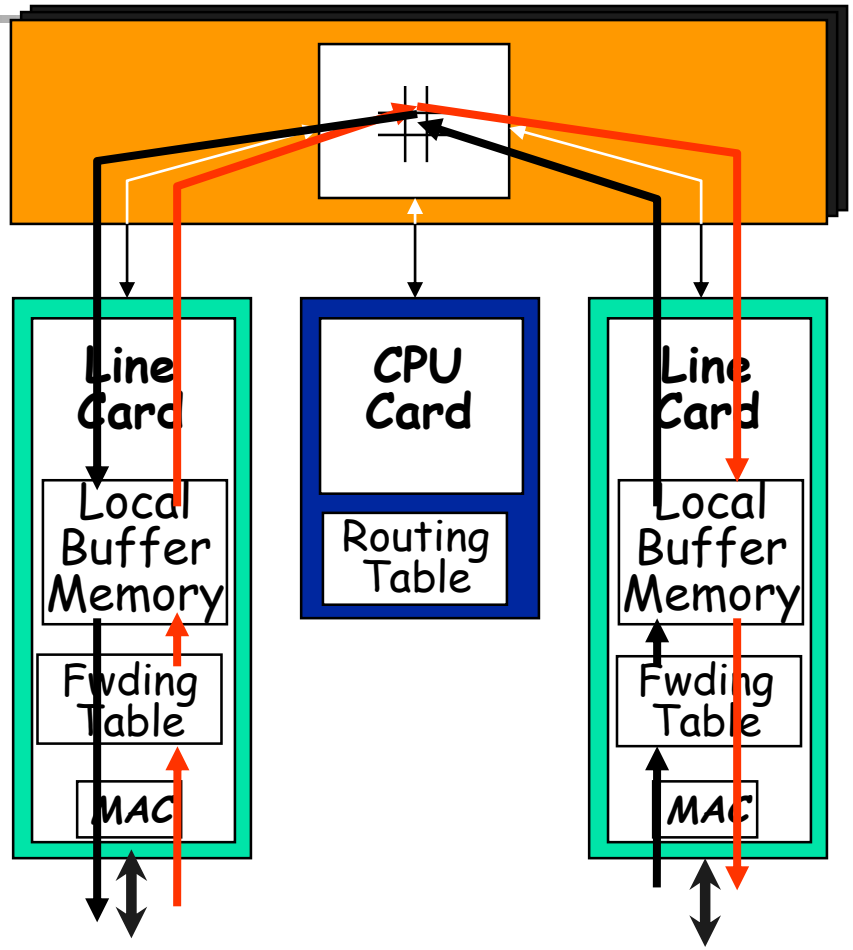
Typically < 120 Gb/s aggregate capacity

Third Generation Routers



Switched Backplane

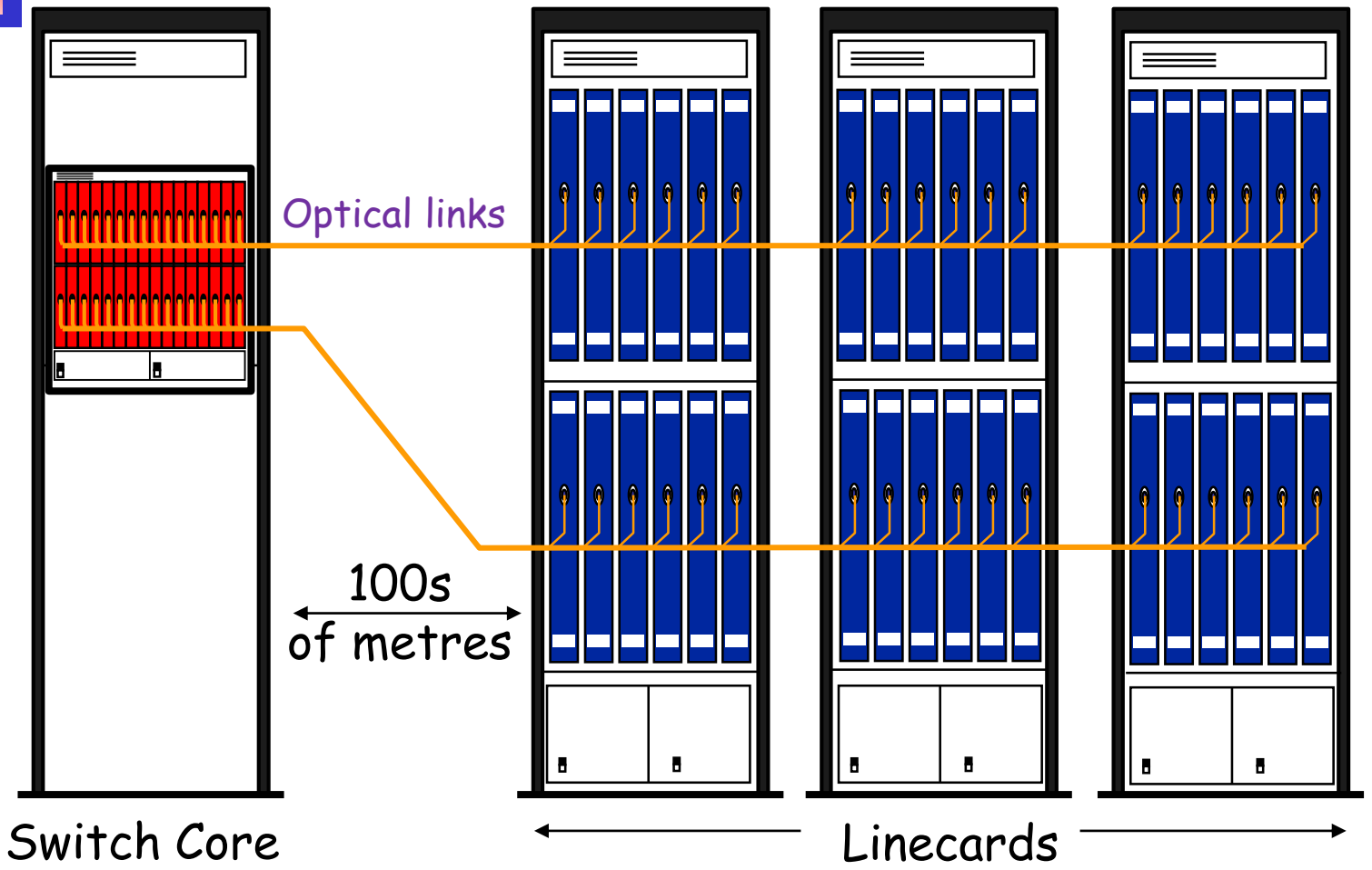




Typically < 640 Gb/s aggregate capacity

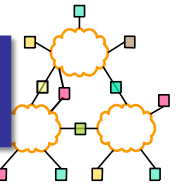
Fourth Generation Routers/Switches

Optics inside a router for the first time



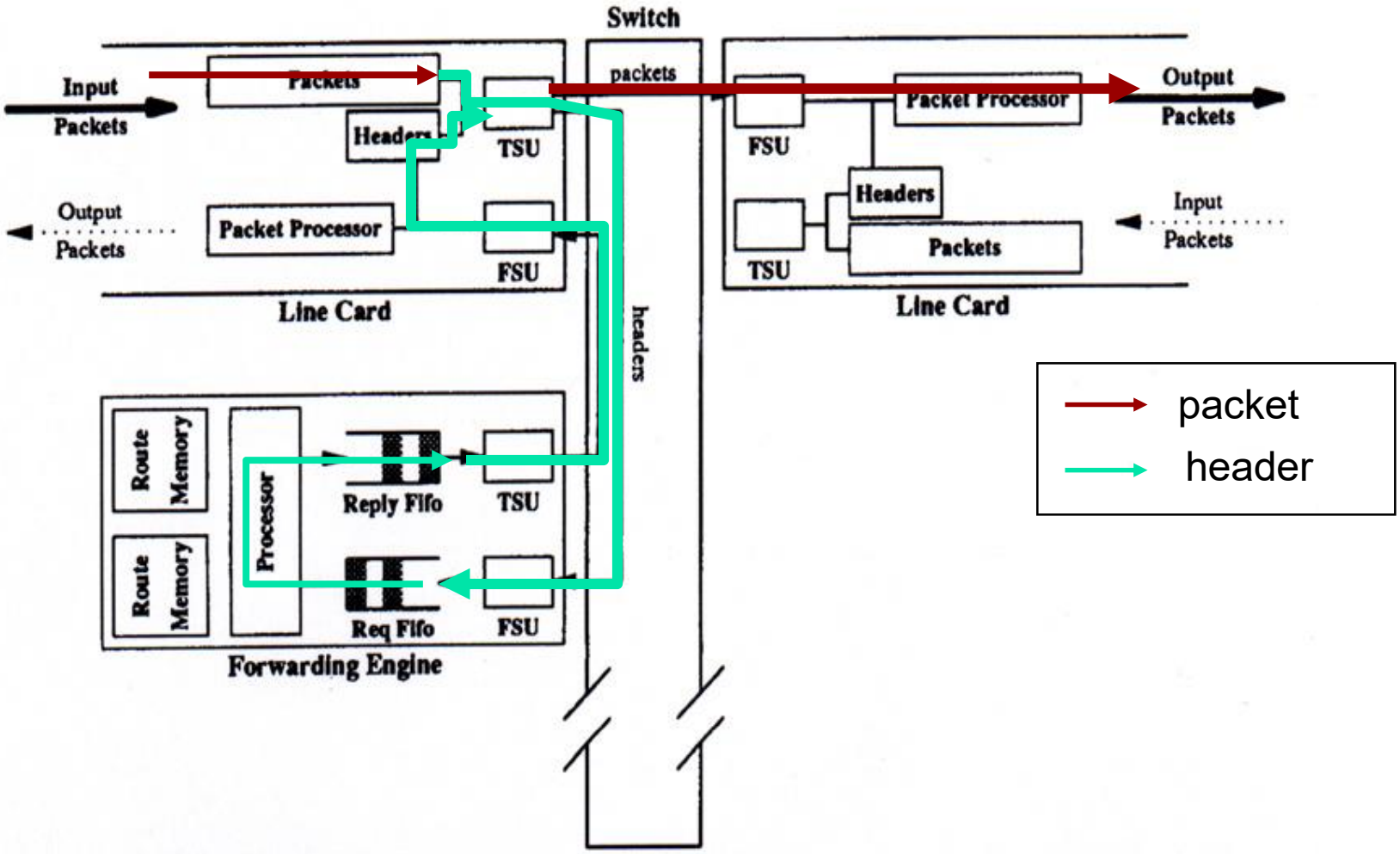
0.3 - 10Tb/s routers in development

A Case Study [Part., '98]

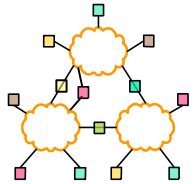


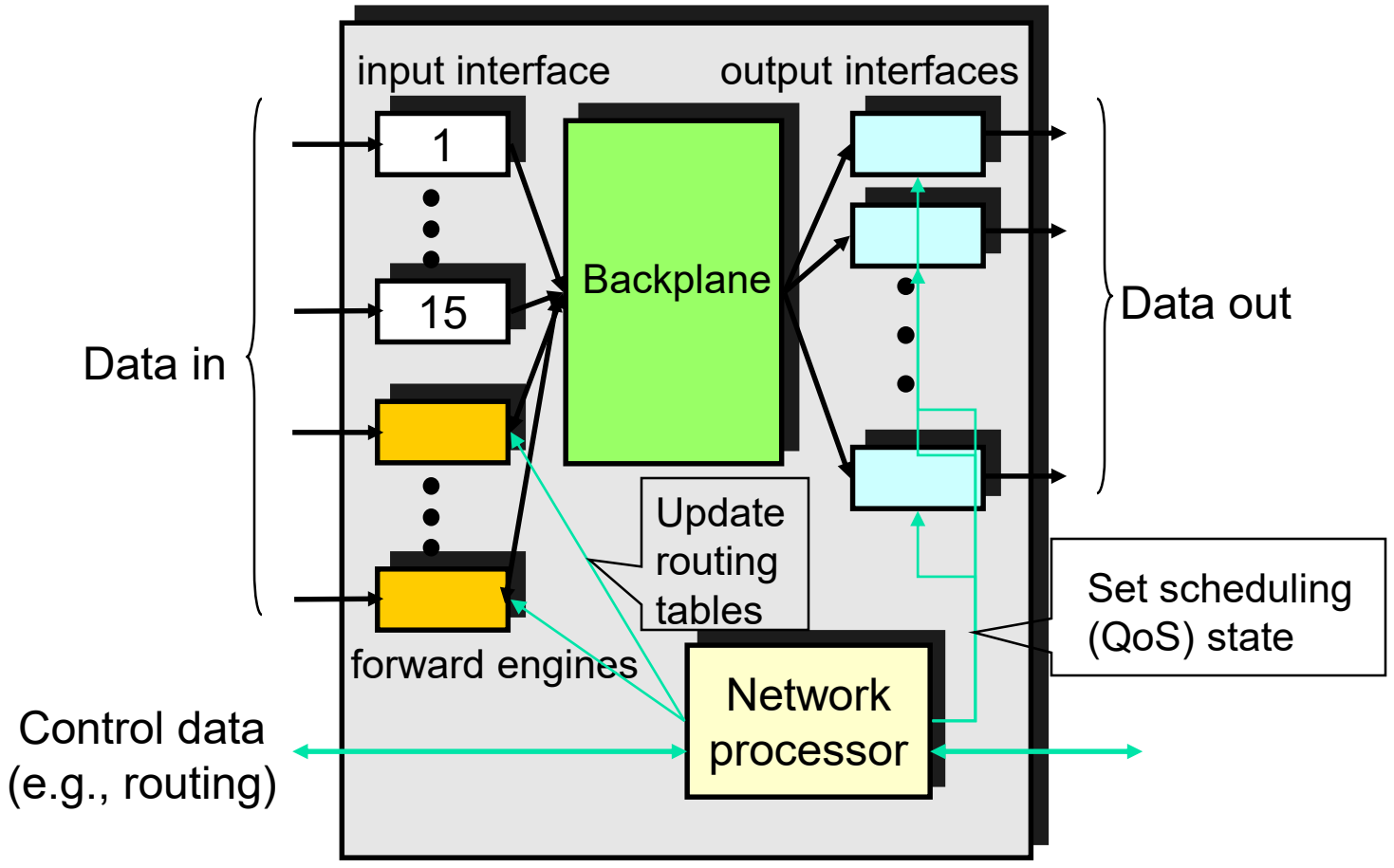
- Goal: show that routers can keep pace with improvements of transmission link bandwidths
- Architecture
  - A CIOQ router
  - 15 (input/output) line cards: C = 2.4 Gbps
    - Each input card can handle up to 16 (input/output) interfaces
    - Separate forward engines (FEs) to perform routing
  - Backplane: Point-to-point (switched) bus, capacity B = 50 Gbps (32 MPPS)
    - B/C = 20, but 25% of B lost to overhead (control) traffic

Router Architecture

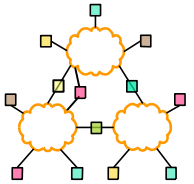


Architecture



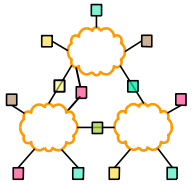


Data Plane



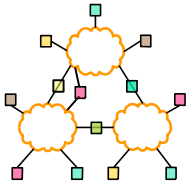
- Line cards
  - Input processing: can handle input links up to 2.4 Gbps (3.3 Gbps including overhead)
  - Output processing: use a 52 MHz FPGA; implements QoS
- Forward engine:
  - 415-MHz DEC Alpha 21164 processor, three level cache to store recent routes
    - Up to 12,000 routes in second level cache (96 kB); ~ 95% hit rate
    - Entire routing table in tertiary cache (16 MB divided in two banks)

Control Plane



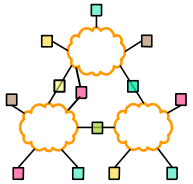
- Network processor: 233-MHz 21064 Alpha running NetBSD 1.1
  - Update routing
  - Manage link status
  - Implement reservation
- Backplane Allocator: implemented by an FPGA
  - Schedule transfers between input/output interfaces

Checksum



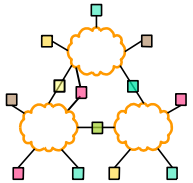
- Takes too much time to verify checksum
  - Requires 17 instructions with min of 14 cycles, Increases forwarding time by 21%
- Take an optimistic approach: just incrementally update it
  - Safe operation: if checksum was correct it remain correct
  - If checksum bad, it will be anyway caught by end-host
- Note: IPv6 does not include a header checksum anyway!

Slow Path Processing



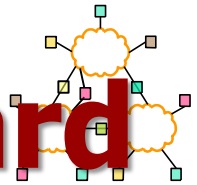
1. Headers whose destination misses in the cache
2. Headers with errors
3. Headers with IP options
4. Datagrams that require fragmentation
5. Multicast datagrams
   - Requires multicast routing which is based on source address and inbound link as well
   - Requires multiple copies of header to be sent to different line cards

Backplane Allocator



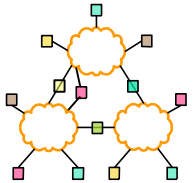
- Time divided in epochs
  - An epoch consists of 16 ticks of data clock (8 allocation clocks)
- Transfer unit: 64 B (8 data click ticks)
- During one epoch, up to 15 simultaneous transfers in an epoch
  - One transfer: two transfer units (128 B of data + 176 auxiliary bits)
- Minimum of 4 epochs to schedule and complete a transfer but scheduling is pipelined.
  1. Source card signals that it has data to send to the destination card
  2. Switch allocator schedules transfer
  3. Source and destination cards are notified and told to configure themselves
  4. Transfer takes place
- Flow control through inhibit pins

The Switch Allocator Card



- Takes connection requests from function cards
- Takes inhibit requests from destination cards
- Computes a transfer configuration for each epoch
- 15X15 = 225 possible pairings with 15! Patterns

Allocator Algorithm



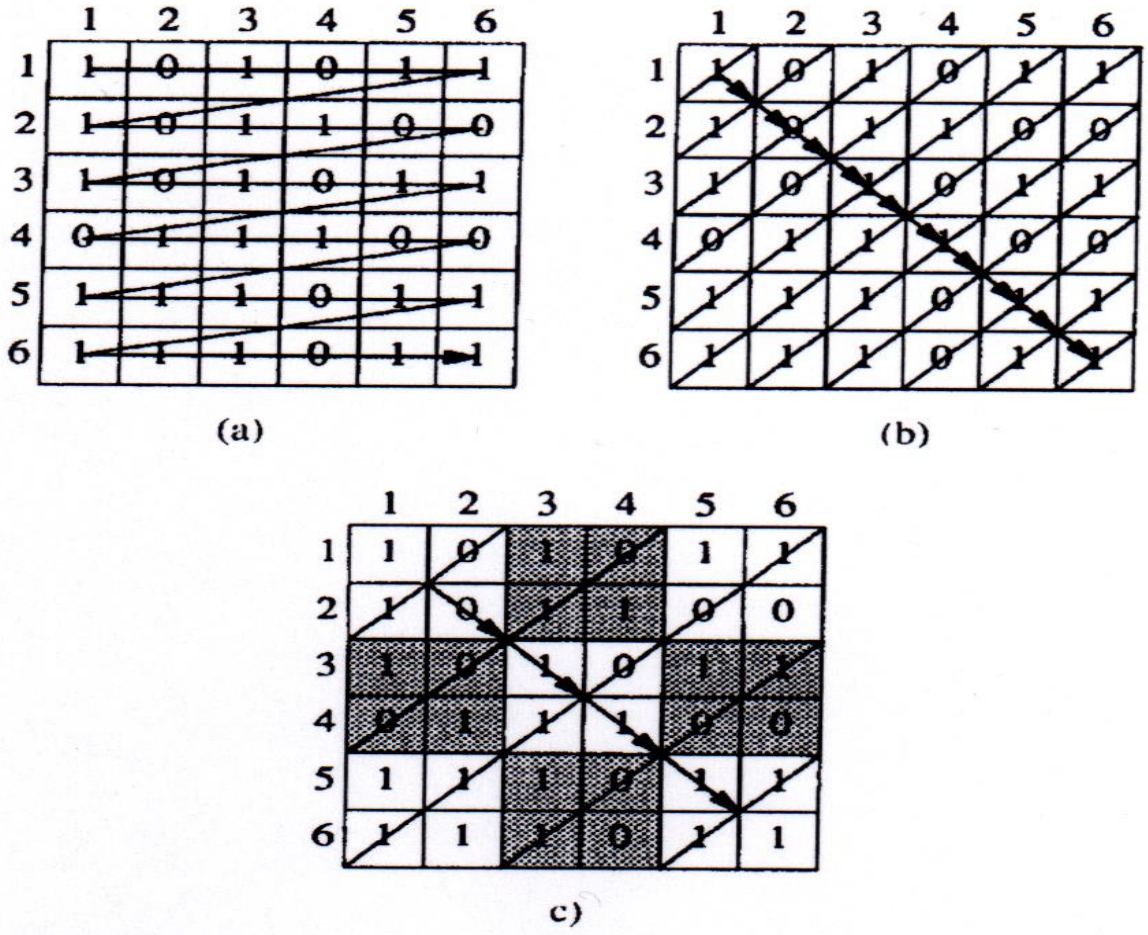
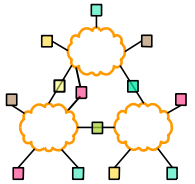


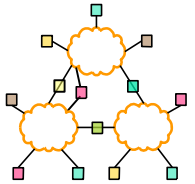
Fig. 4. Simple and wavefront allocators. (a) Simple. (b) Wavefront. (c) Group wavefront.

The Switch Allocator



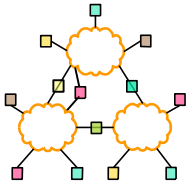
- Disadvantages of the simple allocator
  - Unfair: there is a preference for low-numbered sources
  - Requires evaluating 225 positions per epoch, which is too fast for an FPGA
- Solution to unfairness problem: Random shuffling of sources and destinations
- Solution to timing problem: Parallel evaluation of multiple locations
- Priority to requests from forwarding engines over line cards to avoid *header contention* on line cards

But…



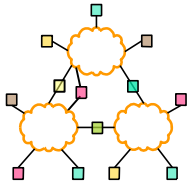
- Remember that if you want per flow processing the performance need to increase at a faster rate than the link capacity!
- If link capacity increases by n, two effects:
  - The time to process a packet decreases by n
  - The number of flows increase (by n?), thus the per packet processing also increases

Challenges



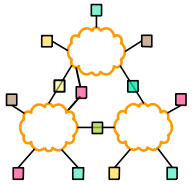
- Build an optimal allocator that makes decisions in constant time
- Packet classification.
- Packet scheduling.

Fast path of the code



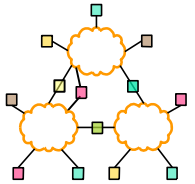
- Stage 1:
  1. Basic error checking to see if header is from a IP datagram
  2. Confirm packet/header lengths are reasonable
  3. Confirm that IP header has no options
  4. Compute hash offset into route cache and load the route
  5. Start loading of next header

Fast path of the code



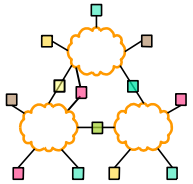
- Stage 2
  1. Check if cached route matches destination of the datagram
  2. If not then do an extended lookup in the route table in Bcache
  3. Update TTL and CHECKSUM fields
- Stage 3
  1. Put updated TTL, checksum and the route information into IP hdr along with link layer info from the forwarding table

Some datagrams not handled in fast path



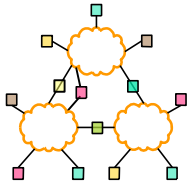
1. Headers whose destination misses in the cache
2. Headers with errors
3. Headers with IP options
4. Datagrams that require fragmentation
5. Multicast datagrams
   - Requires multicast routing which is based on source address and inbound link as well
   - Requires multiple copies of header to be sent to different line cards

Instruction set



- 27% of them do bit, byte or word manipulation due to extraction of various fields from headers
- The above instructions can only be done in E0, resulting in contention (checksum verifying)
- Floating point instructions account for 12% but do not have any impact on performance as they only set SNMP values and can be interleaved
- There is a minimum of loads(6) and stores(4)

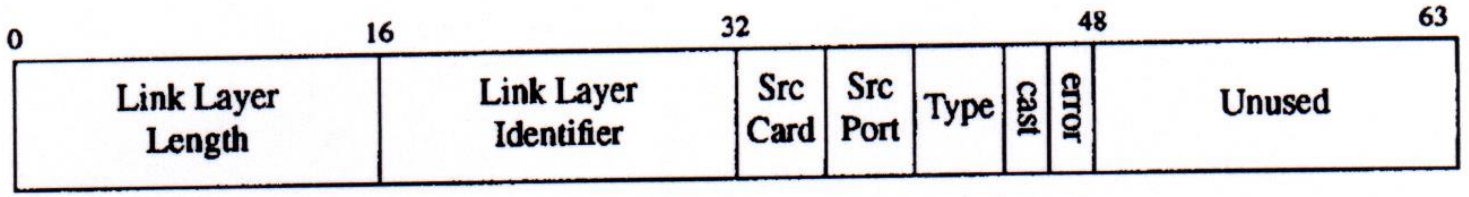
Issues in forwarding design



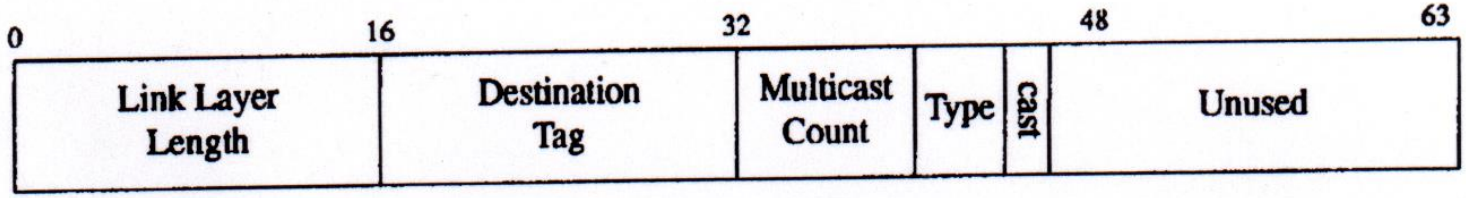
- *Why not use an ASIC in place of the engine ?*
  - Since IP protocol is stable, why not do it ?
  - Answer depends on where the router will be deployed: corporate LAN or ISP's backbone?
- *How effective is a route cache ?*
  - A full route lookup is 5 times more expensive than a cache hit. So we need modest hit rates.
  - And modest hit rates seem to be assured because of packet trains

Abstract link layer header

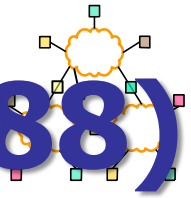
- Designed to keep the forwarding engine and its code simple



(a)

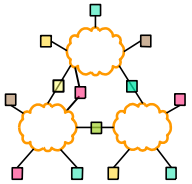


Forwarding Engine (P+88)



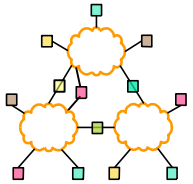
- General purpose processor + software
- 8KB L1 Icache
  - Holds full forwarding code
- 96KB L2 cache
  - Forwarding table cache
- 16MB L3 cache
  - Full forwarding table x 2 - double buffered for updates

Network Processor



- Runs routing protocol and downloads forwarding table to forwarding engines
  - Two forwarding tables per engine to allow easy switchover
- Performs "slow" path processing
  - Handles ICMP error messages
  - Handles IP option processing

Line Card Interconnect (P+88)

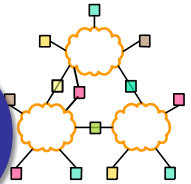


- Virtual output buffering
  - Maintain per output buffer at input
  - Solves head of line blocking problem
  - Each of MxN input buffer places bid for output
- Crossbar connect
- Challenge: map of bids to schedule for crossbar

Switch Scheduling (P+88)

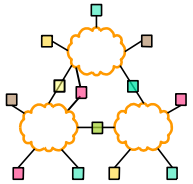
- Schedule for 128 byte slots
  - Greedy mapping of inputs to outputs
- Fairness
  - Order of greedy matching permuted randomly
  - Priority given to forwarding engine in schedule (why?)
- Parallelized
  - Check independent paths simultaneously

Summary: Design Decisions (Innovations)



1. Each FE has a complete set of the routing tables
2. A switched fabric is used instead of the traditional shared bus
3. FEs are on boards distinct from the line cards
4. Use of an abstract link layer header
5. Include QoS processing in the router

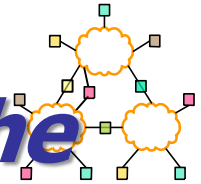
Why Faster Routers?

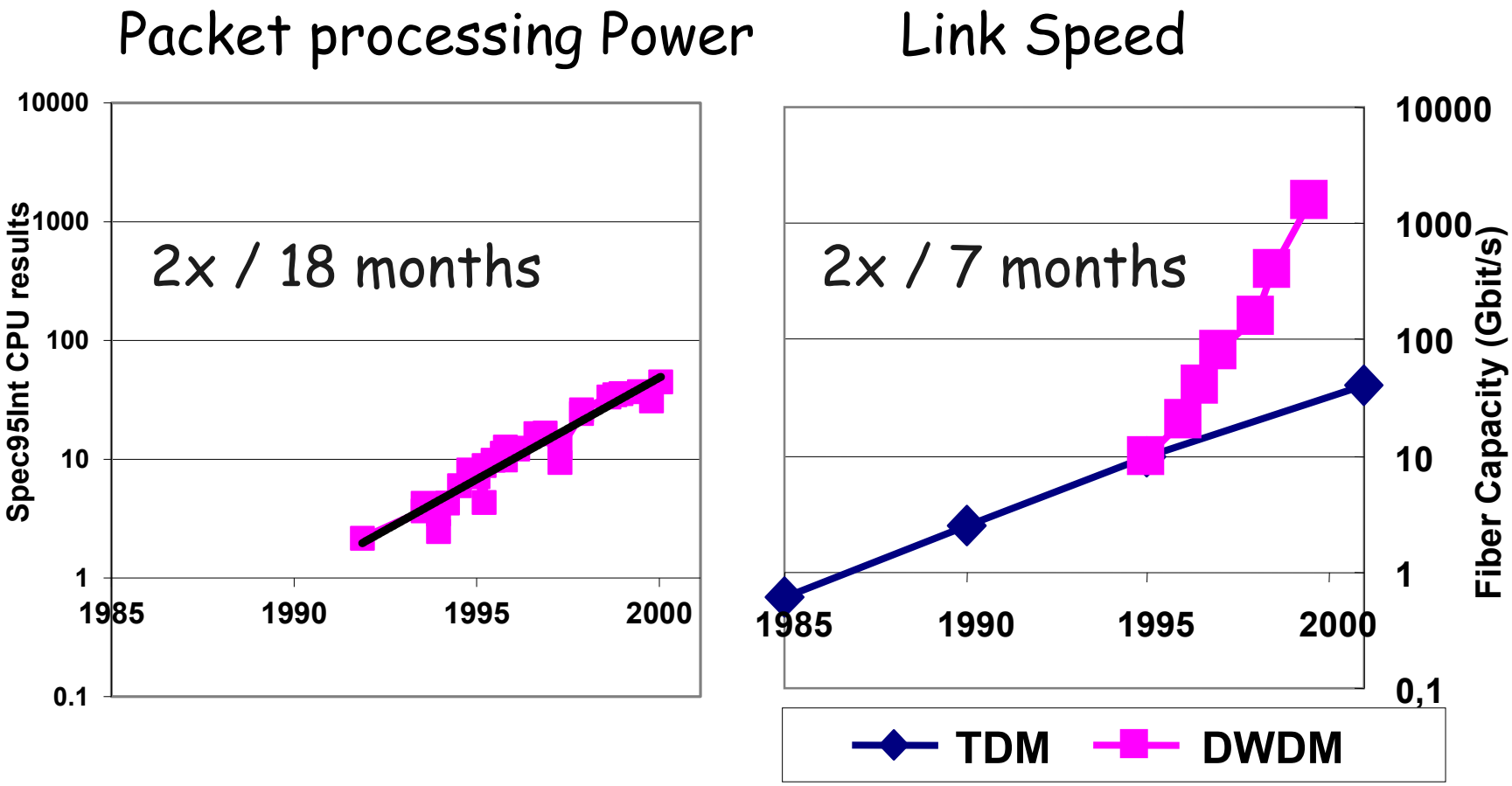


1. To prevent routers becoming the bottleneck in the Internet.
2. To increase POP capacity, and to reduce their cost, size and power.

Why Faster Routers?

1: *To prevent routers from being the bottleneck*

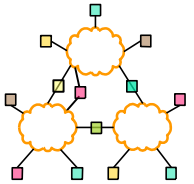




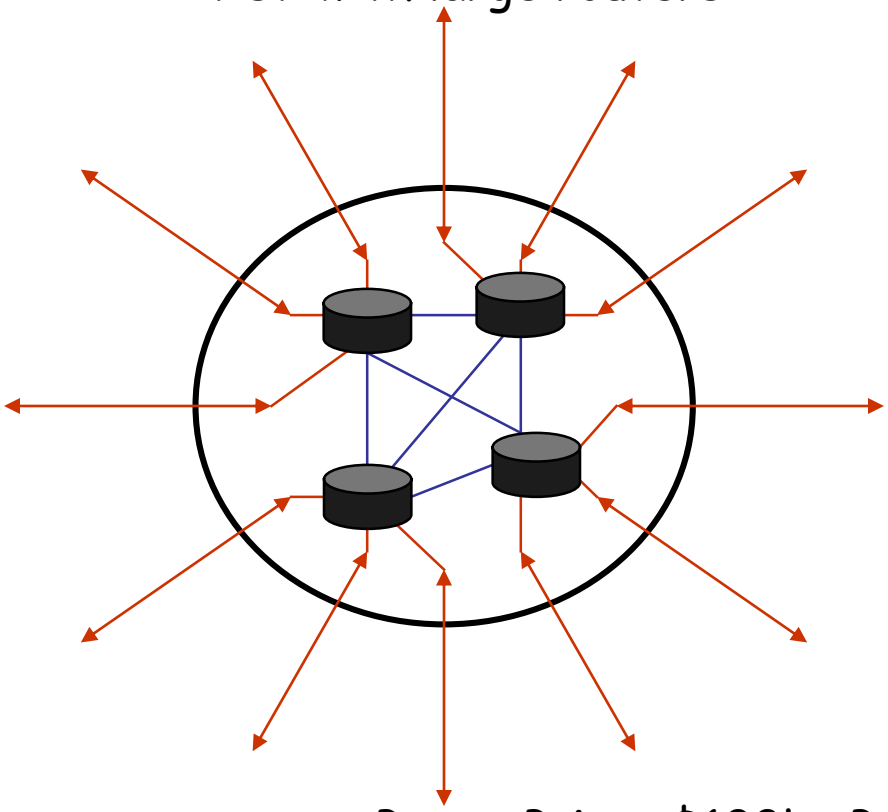
Source: SPEC95Int & David Miller, Stanford.

Why Faster Routers?

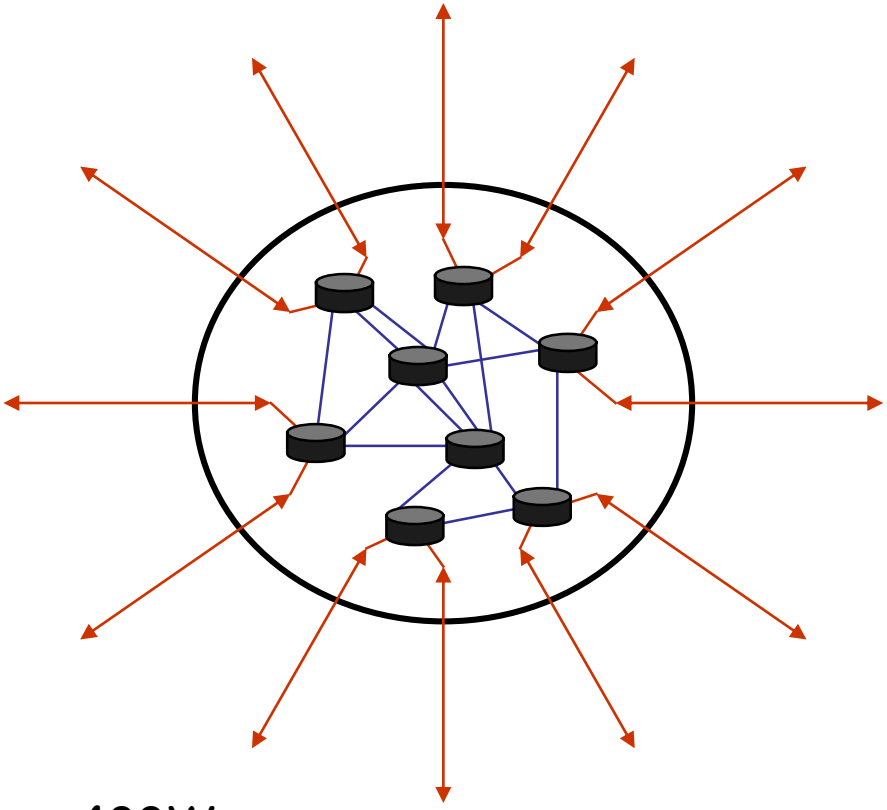
2: *To reduce cost, power & complexity of POPs*



POP with large routers

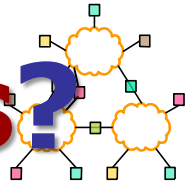


POP with smaller routers



- Ports: Price >$100k, Power > 400W.
- It is common for 50-60% of ports to be for interconnection.

Fast Routers Difficulties?

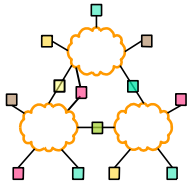


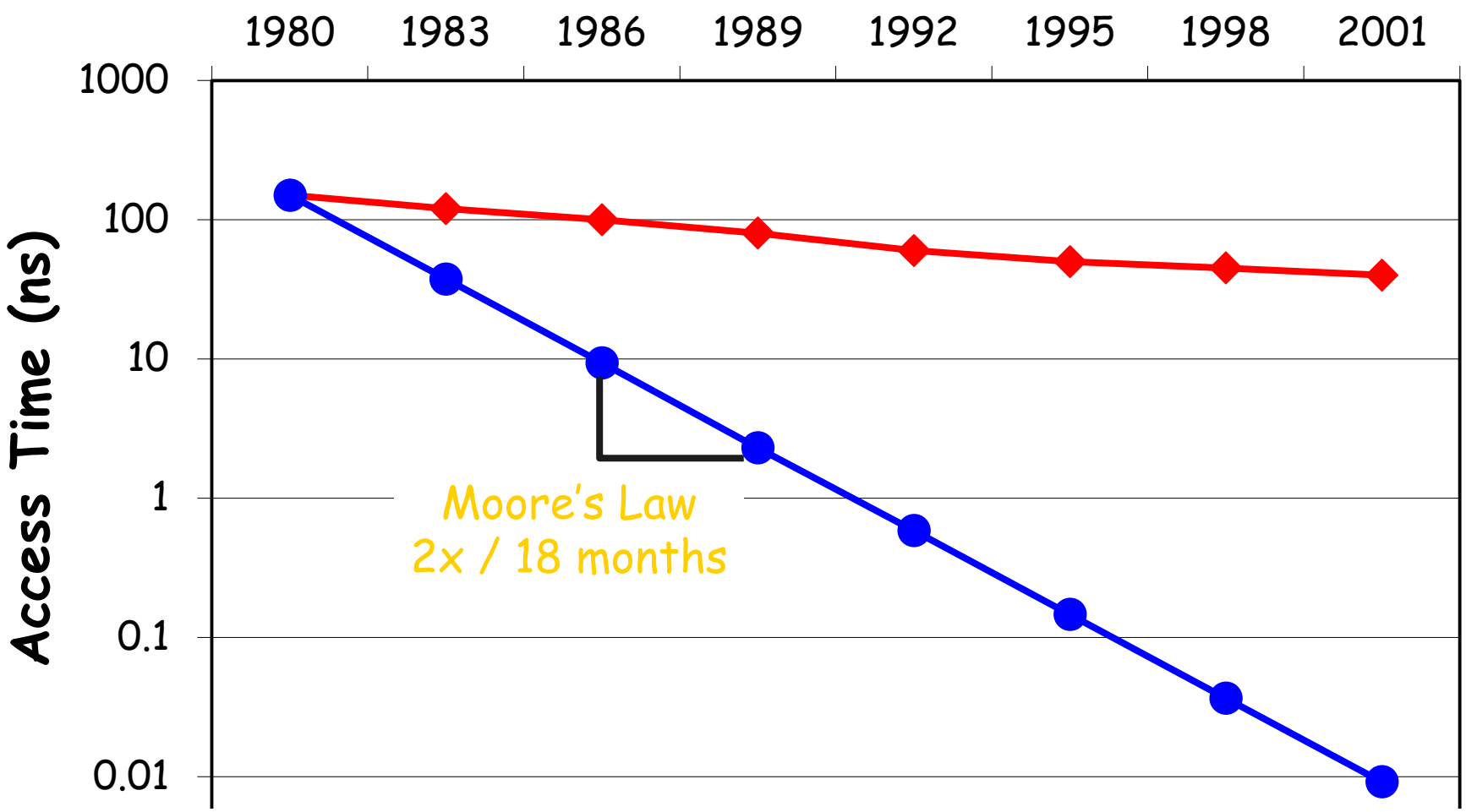
1. **It's hard to keep up with Moore's Law:**
   - The bottleneck is memory speed.
   - Memory speed is not keeping up with Moore's Law.

1.1x / 18 months

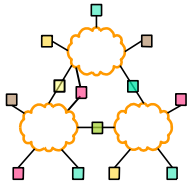
Fast Routers Difficulties?

Speed of Commercial DRAM



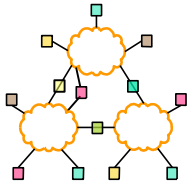


Fast Routers Difficulties?



1. **It's hard to keep up with Moore's Law:**
   - The bottleneck is memory speed.
   - Memory speed is not keeping up with Moore's Law.
2. **Moore's Law is too slow:**
   - Routers need to improve *faster* than Moore's Law.

Router Performance Exceeds Moore's Law

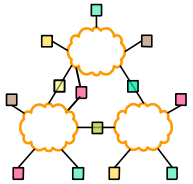


Growth in capacity of commercial routers:

- Capacity 1992 ~ 2Gb/s
- Capacity 1995 ~ 10Gb/s
- Capacity 1998 ~ 40Gb/s
- Capacity 2001 ~ 160Gb/s
- Capacity 2003 ~ 640Gb/s

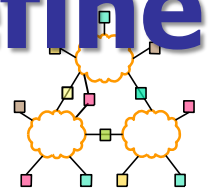
Average growth rate: 2x / 18 months.

Conclusions



- It is feasible to implement IP routers at very high speeds
- Today:
  - Input link: 100 Gbps
  - Backplane: 4 Tbps

Next-lecture: software define Networking



- What is SDN?
- Why wee need SDN?
- SDN Architecture.
- Problems with SDN.