UNIVERSITY OF WATERLOO

CS 456/656 Computer Networks

Lecture 18: Router/Switch Architecture

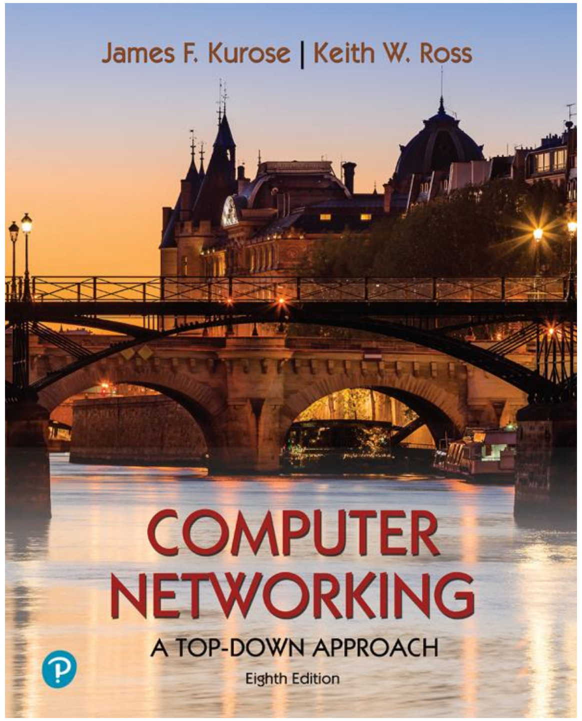
Mina Tahmasbi Arashloo and Bo Sun

Fall 2024

A note on the slides

Adapted from the slides that accompany this book.

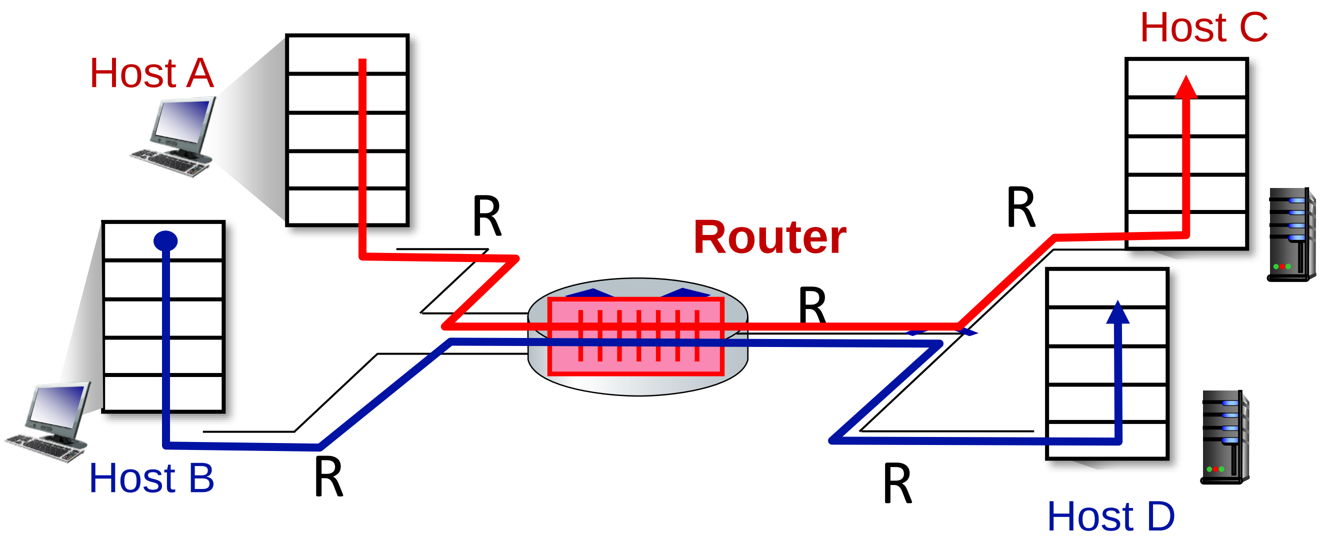
And lecture notes from Anirudh Sivaraman, NYU



Computer Networking: A Top-Down Approach
8^{th} edition
Jim Kurose, Keith Ross
Pearson, 2020

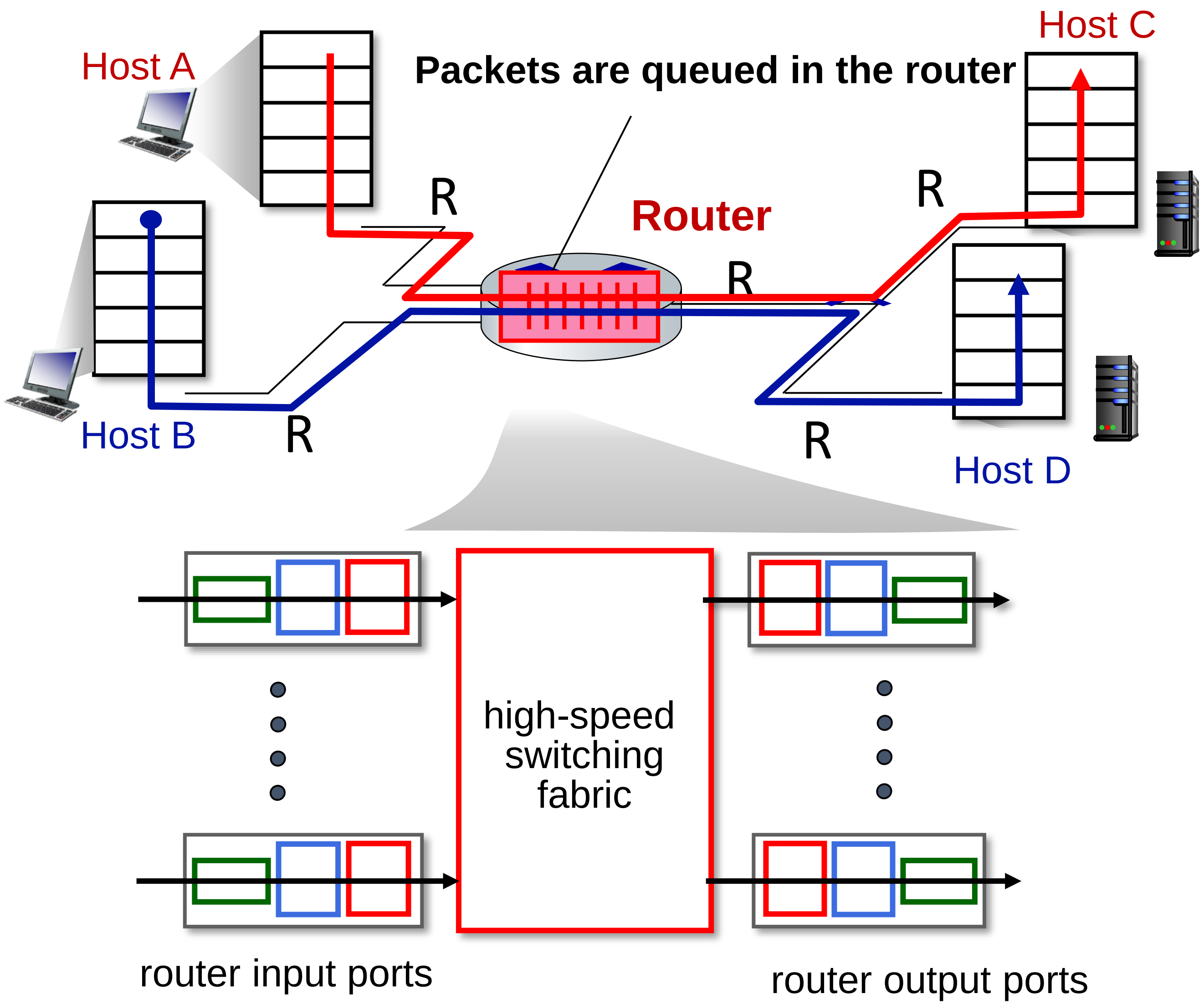
What we discussed before

- Packets can be buffered in routers
  - delay and loss
  - network congestion



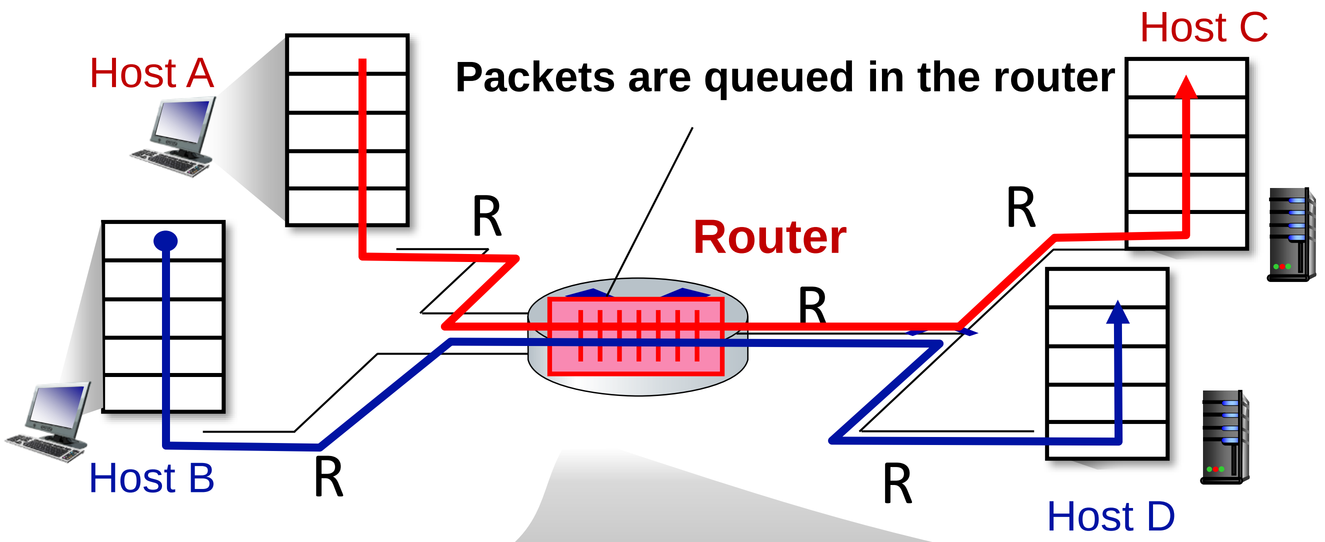
What we discussed before

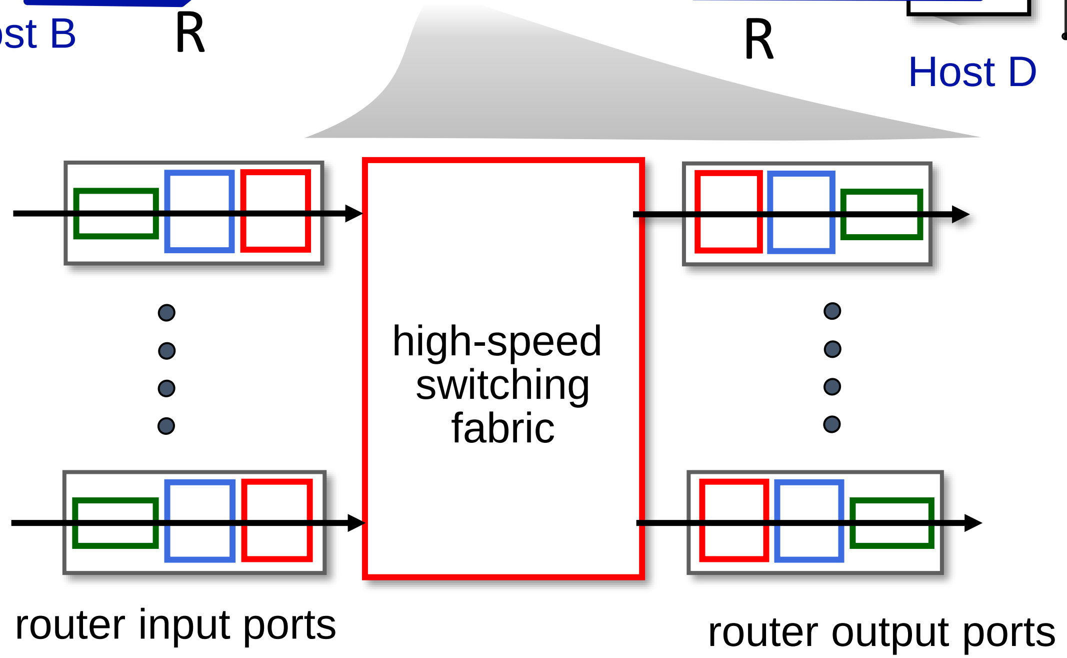
- Packets can be buffered in routers
  - delay and loss
  - network congestion
- A high-level view of a router architecture.
  - multiple input ports
  - multiple output ports
  - a switching fabric



What we discussed before

- Packets can be buffered in routers
  - delay and loss
  - network congestion
- A high-level view of a router architecture.
  - multiple input ports
  - multiple output ports
  - a switching fabric





This Lecture: How are packets buffered and managed in routers?

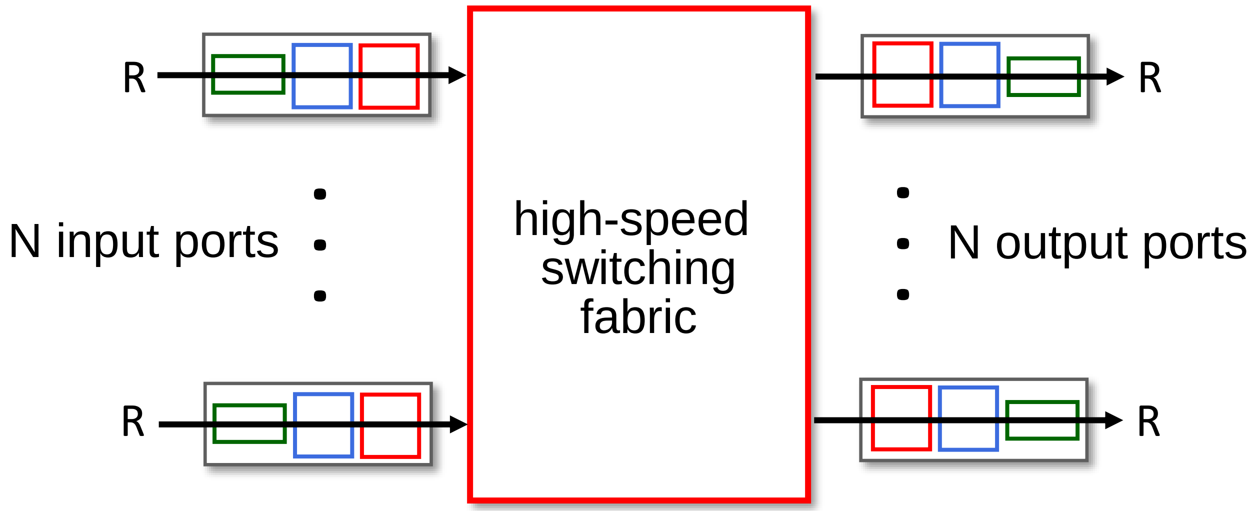
Router architecture and buffer management

- Router architecture
  - Shared memory
  - Output queueing
  - Input queueing

- Buffer management and scheduling

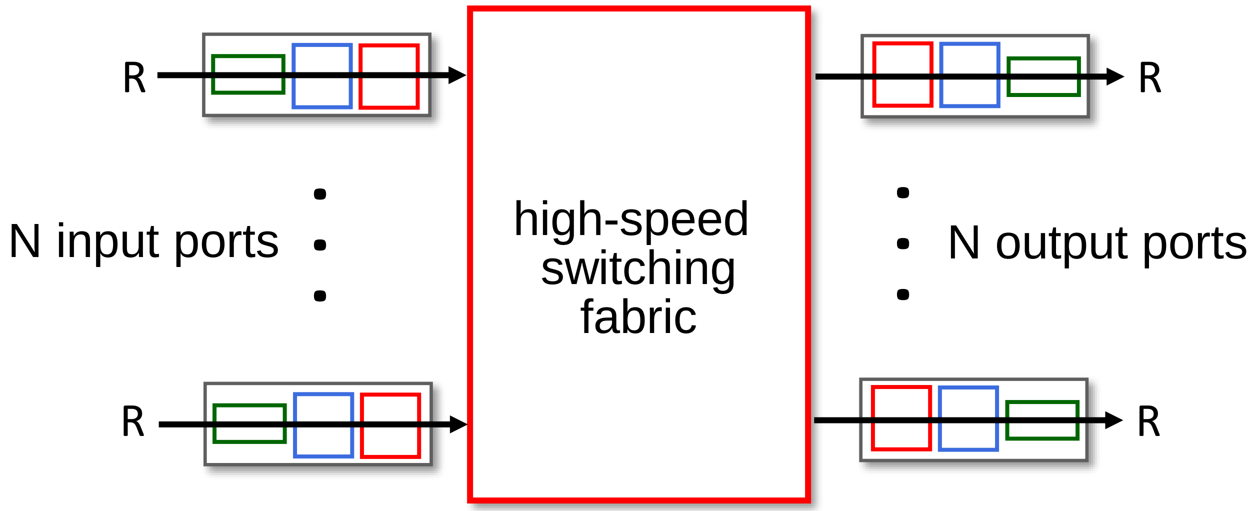
Switching fabrics

- transfer packets from input links to appropriate output links
- Suppose
  - All packets are of the same size
  - Define the time it takes to send/receive a packet on a port as our time unit, and call it a *tick* (today, that's usually a few nanoseconds!)



Switching fabrics

- In each tick, we can
  - receive at most a packet on each input port (up to N ports)
  - send at most a packet on each output port (up to N ports)



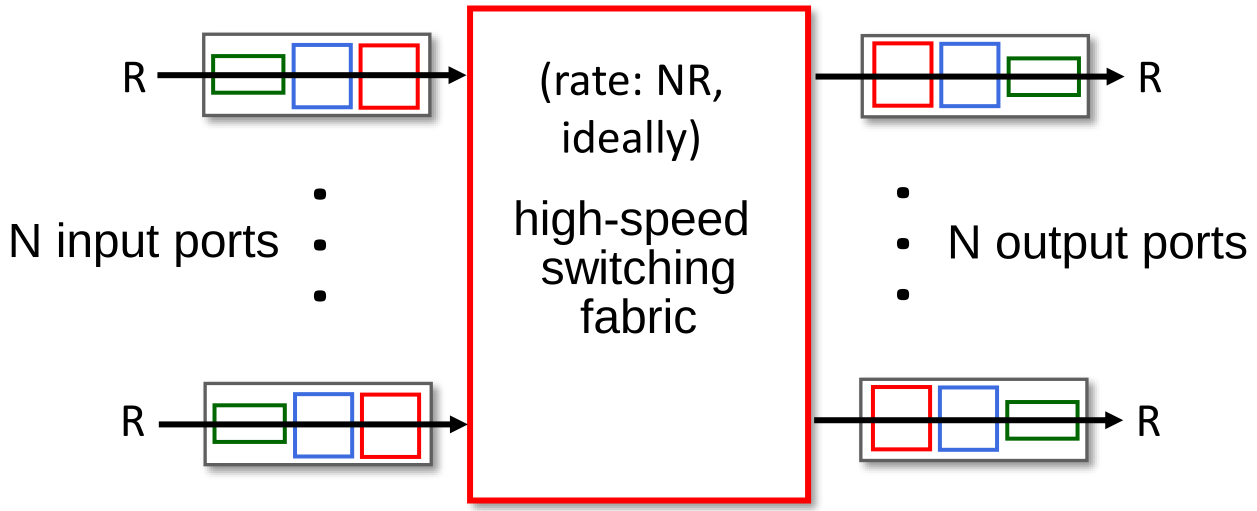
Switching fabrics

On each tick, we can

receive a packet on each input port

send a packet on each output port

- Ideally, the switching fabric can move N packets in each tick
  - If the link rates are R, an ideal switching fabric moves packets at rate NR.



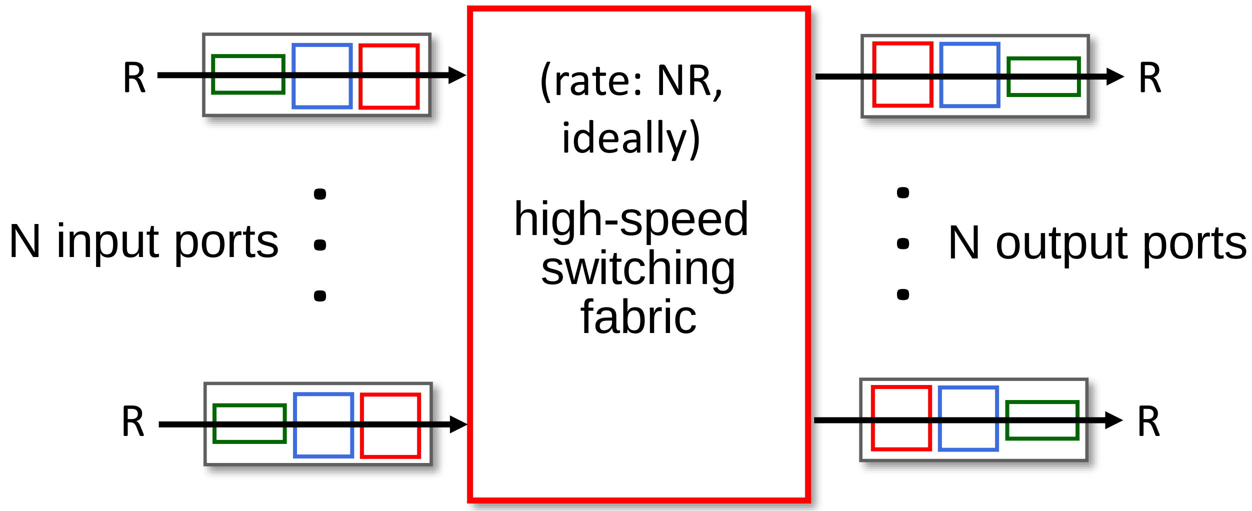
Switching fabrics

On each tick, we can

receive a packet on each input port

send a packet on each output port

- If two or more input ports have a packet destined to the same output port
  - only one can go out in the next tick(s)
  - the rest have to wait somewhere



Switching fabrics

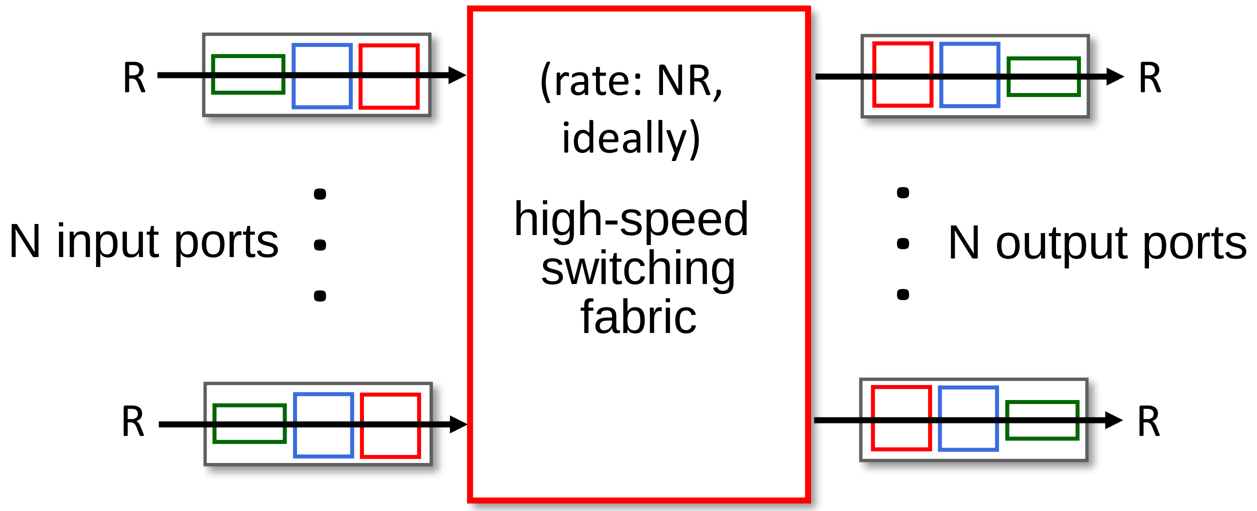
On each tick, we can

receive a packet on each input port

send a packet on each output port

- If two or more input ports have a packet destined to the same output port
  - only one can go out in the next tick(s)
  - the rest have to wait somewhere

Queued in a buffer (*where?*)



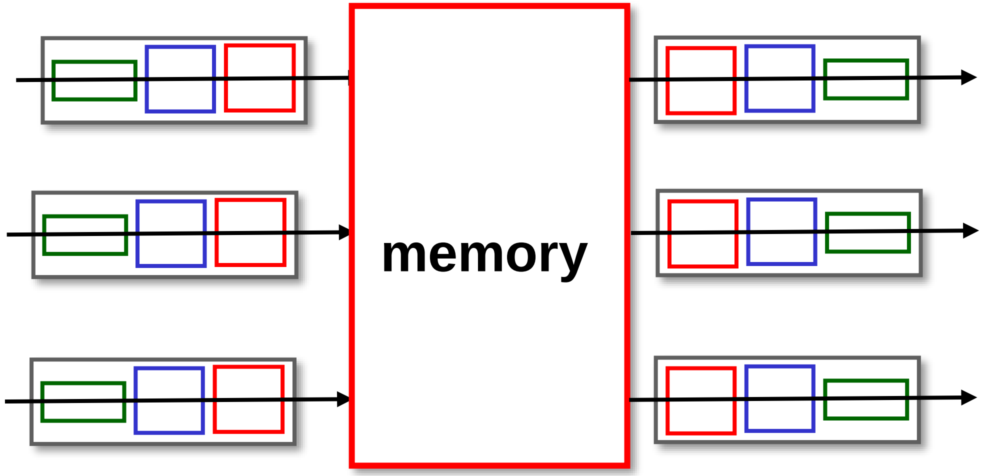
Router architecture and buffer management

- Router architecture
  - Shared memory
  - Output queueing
  - Input queueing

- Buffer management and scheduling

Option 1 – Shared memory

- A single pool of shared memory between all input and output ports.
- Input port receives a packet, and then puts it in some memory region
- Output port pulls the next packet it is supposed to send out from the same memory



Option 1 – Shared memory

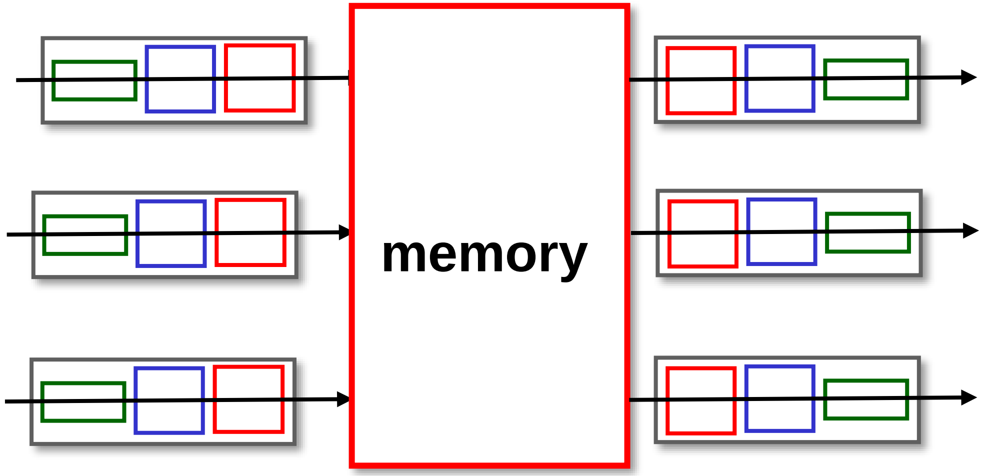
On each tick, we can

receive a packet on each input port

send a packet on each output port

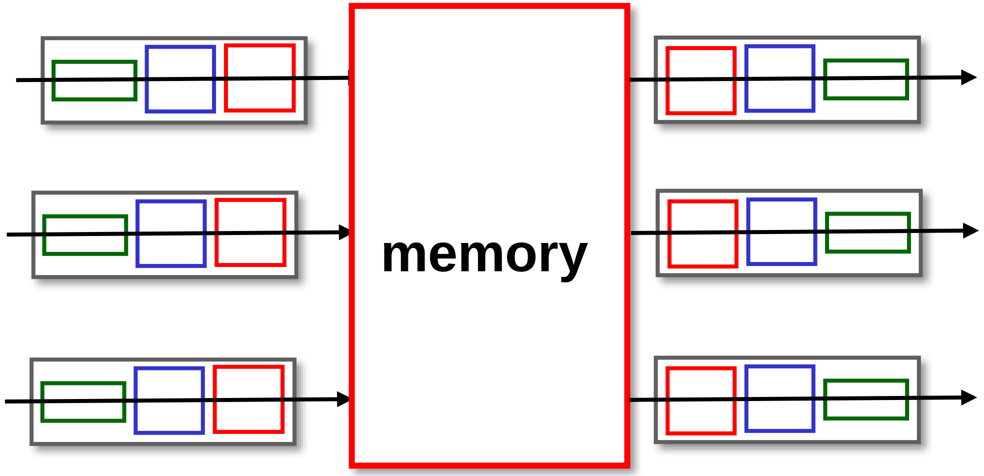
For every tick:

- memory needs to support *N enqueues*
  - Each input port may receive a packet and has to put it in a queue
- memory needs to support *N dequeues*
  - Each output port may have outstanding packets to send



Option 1 – Shared memory

- Pros: dynamically allocate more or less memory to ports depending on current traffic demands
- Cons: difficult to have a *large high-speed* memory that can do N enqueues and dequeues in every tick
  - Remember, tick ~= a few nsecs



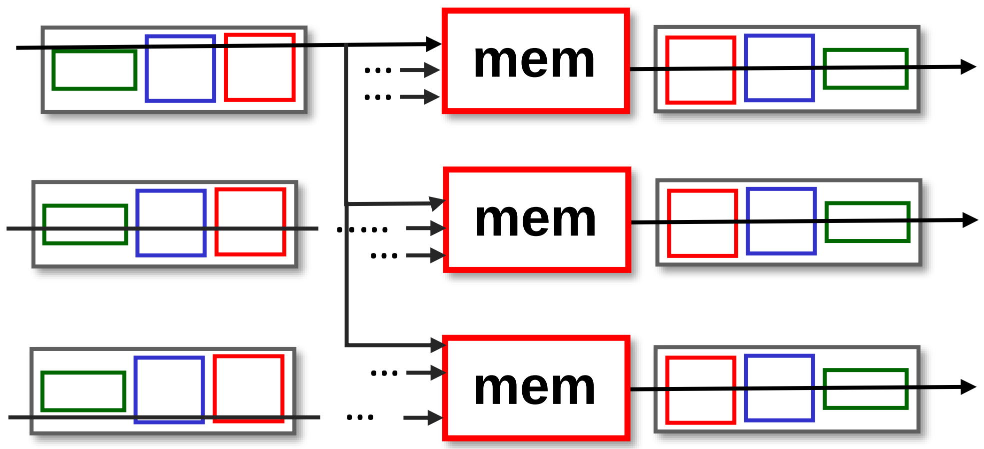
Router architecture and buffer management

- Router architecture
  - Shared memory
  - Output queueing
  - Input queueing

- Buffer management and scheduling

Option 2 – Output queuing

- N separate memories, one for each output port
- Input port receives a packet, and then puts it in the memory of the output port it is supposed to exit from
- Output port pulls the next packet from its corresponding memory



Option 2 – Output queuing

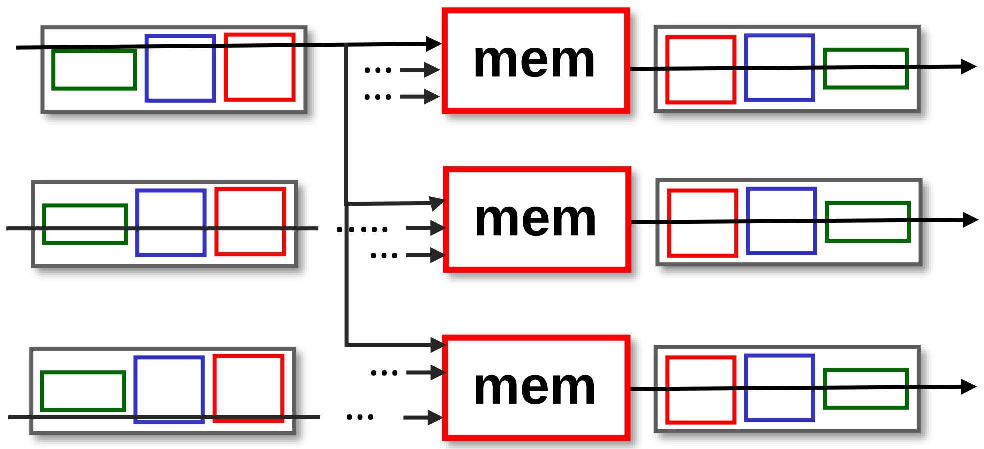
On each tick, we can

receive a packet on each input port

send a packet on each output port

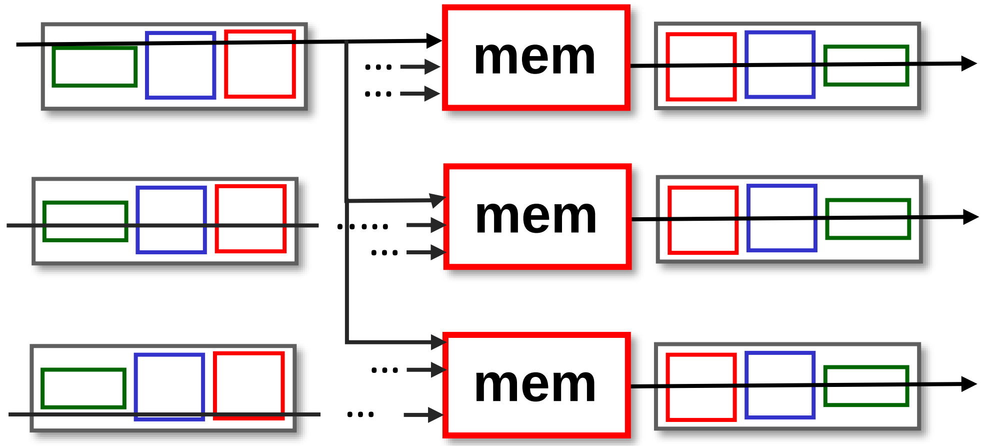
For every tick:

- each memory needs to support *N enqueues*
  - all input ports may receive packets going to the same output port
- each memory only needs to support 1 dequeue per tick



Option 2 – Output queuing

- Pros: For each memory, 1 dequeue per tick
  - as opposed to N dequeues per tick in shared memory
- Cons:
  - Static allocation of memory to output ports: if port 1 is not getting too much traffic and port 2 is, can't give port 1's unused memory to port 2
  - Each memory still has to support N enqueues per tick



Router architecture and buffer management

- Router architecture
  - Shared memory
  - Output queueing
  - Input queueing

- Buffer management and scheduling

Option 3 – Input queuing

- N separate memory pools, one for each input port.
- Input port receives a packet, puts it in the dedicated memory for that input port.
  - 1 enqueue per tick
- Output port gets the next packet from the memory of one of the input ports that have packets destined to it
  - 1 dequeue per tick

Option 3 – Input queuing

At most one enqueue in each memory per tick

Option 3 – Input queuing

At most one dequeue from each memory per tick

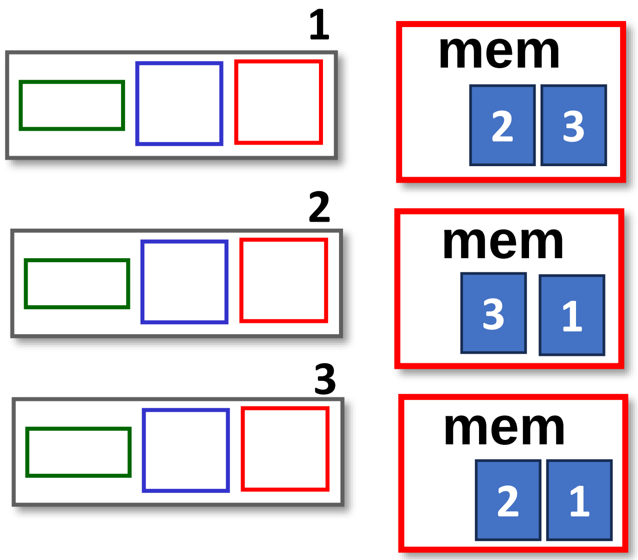
Option 3 – Input queuing

At most one dequeue from each memory per tick

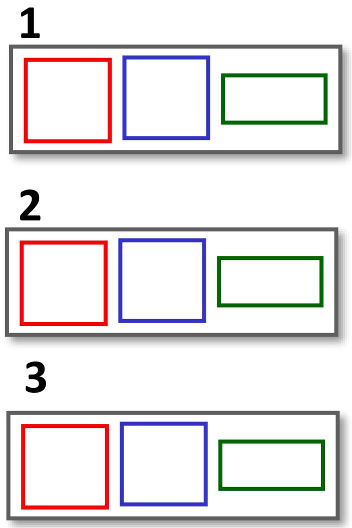
- Will an input queue ever need to send the same packet to multiple output queues (hence the need for >1 dequeue per tick?
  - for a rarely used capability called multicast (not covered in this course).

Option 3 – Input queuing

How do we coordinate packets between inputs and outputs?



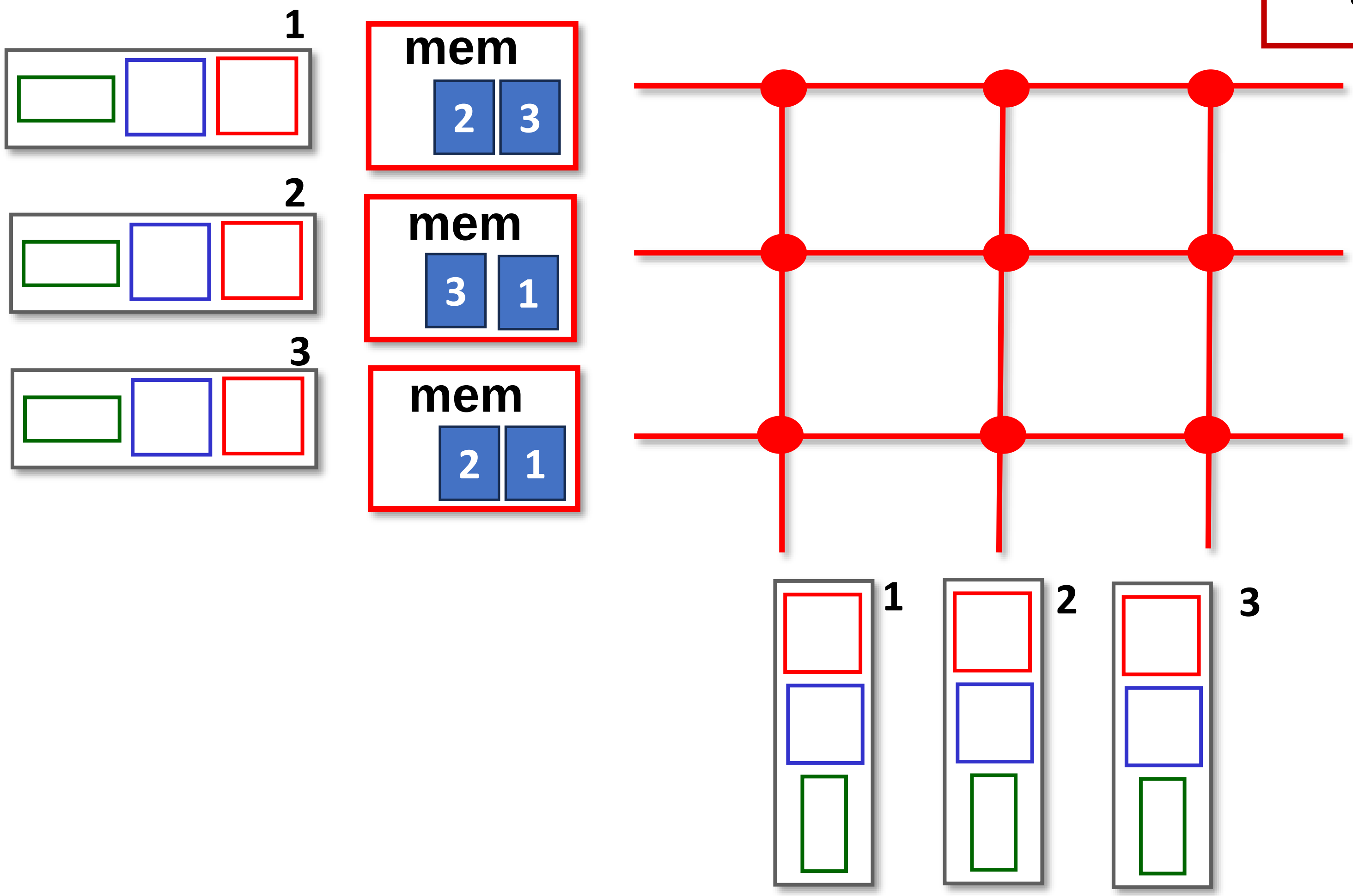
In every tick
- Each input sends to at most one output
- Each output receives from at most each input



Option 3 – Input queuing

In every tick

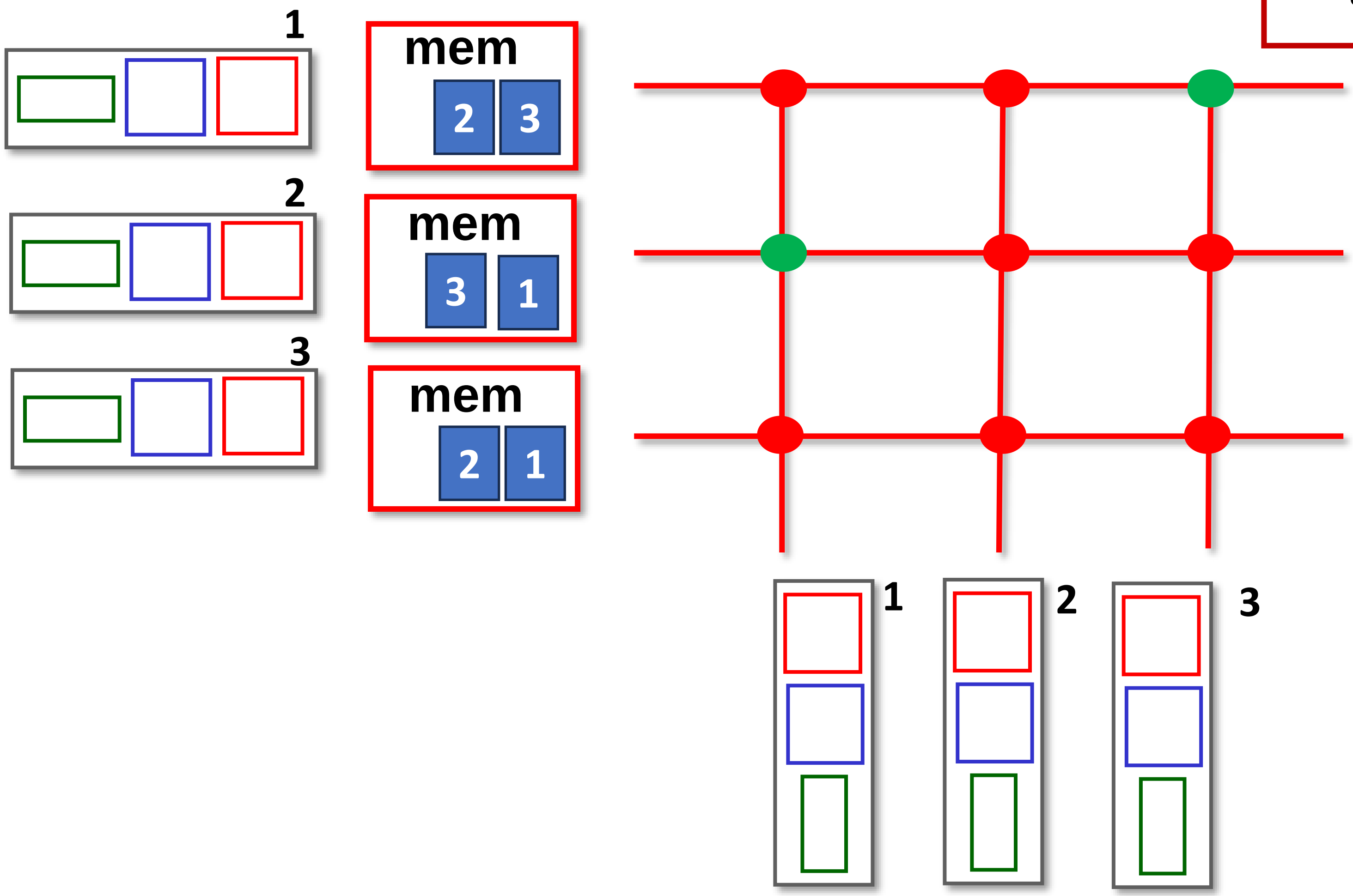
- Each input sends to at most one output
- Each output receives from at most each input



Option 3 – Input queuing

In every tick

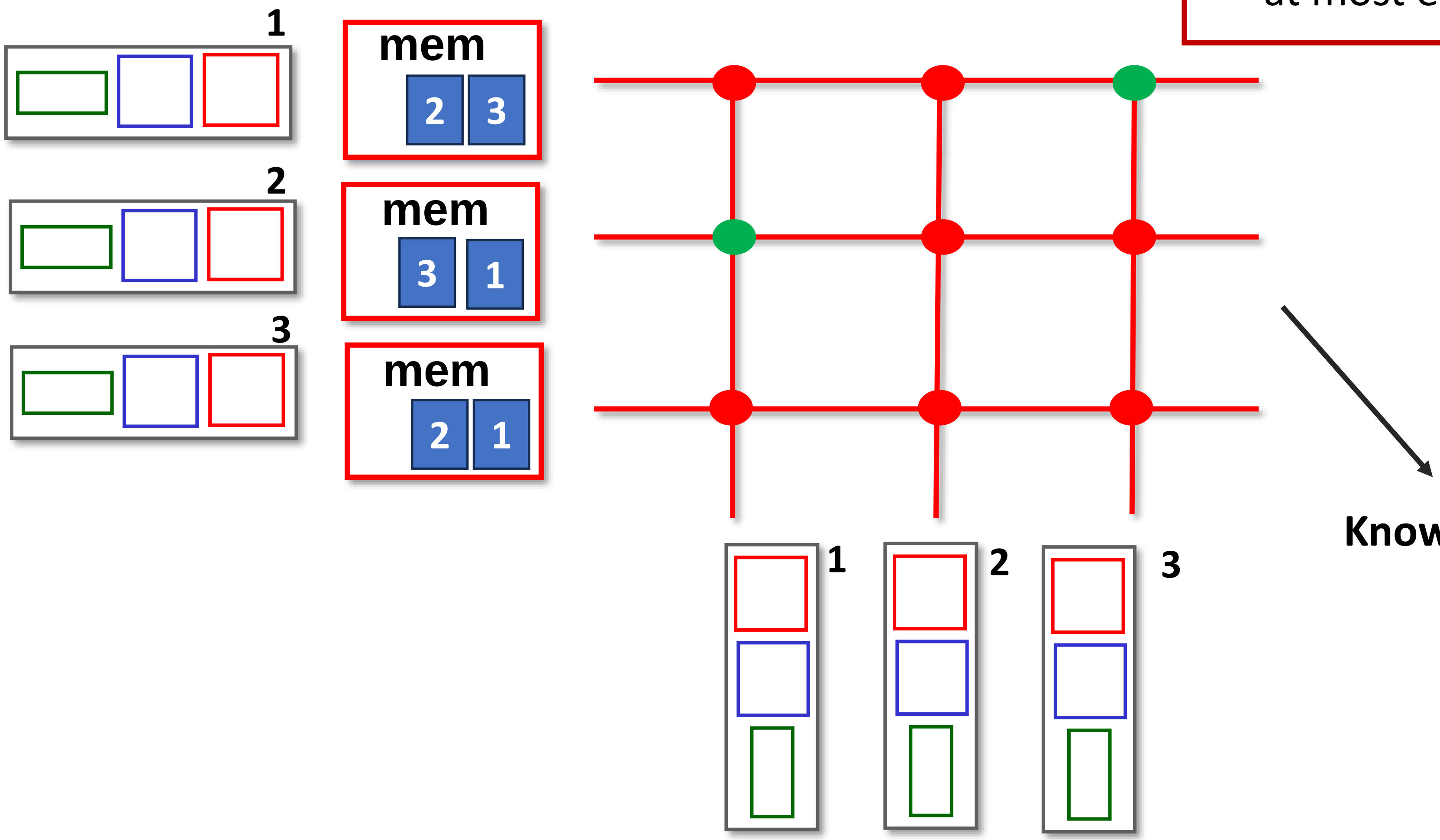
- Each input sends to at most one output
- Each output receives from at most each input



Option 3 – Input queuing

In every tick

- Each input sends to at most one output
- Each output receives from at most each input

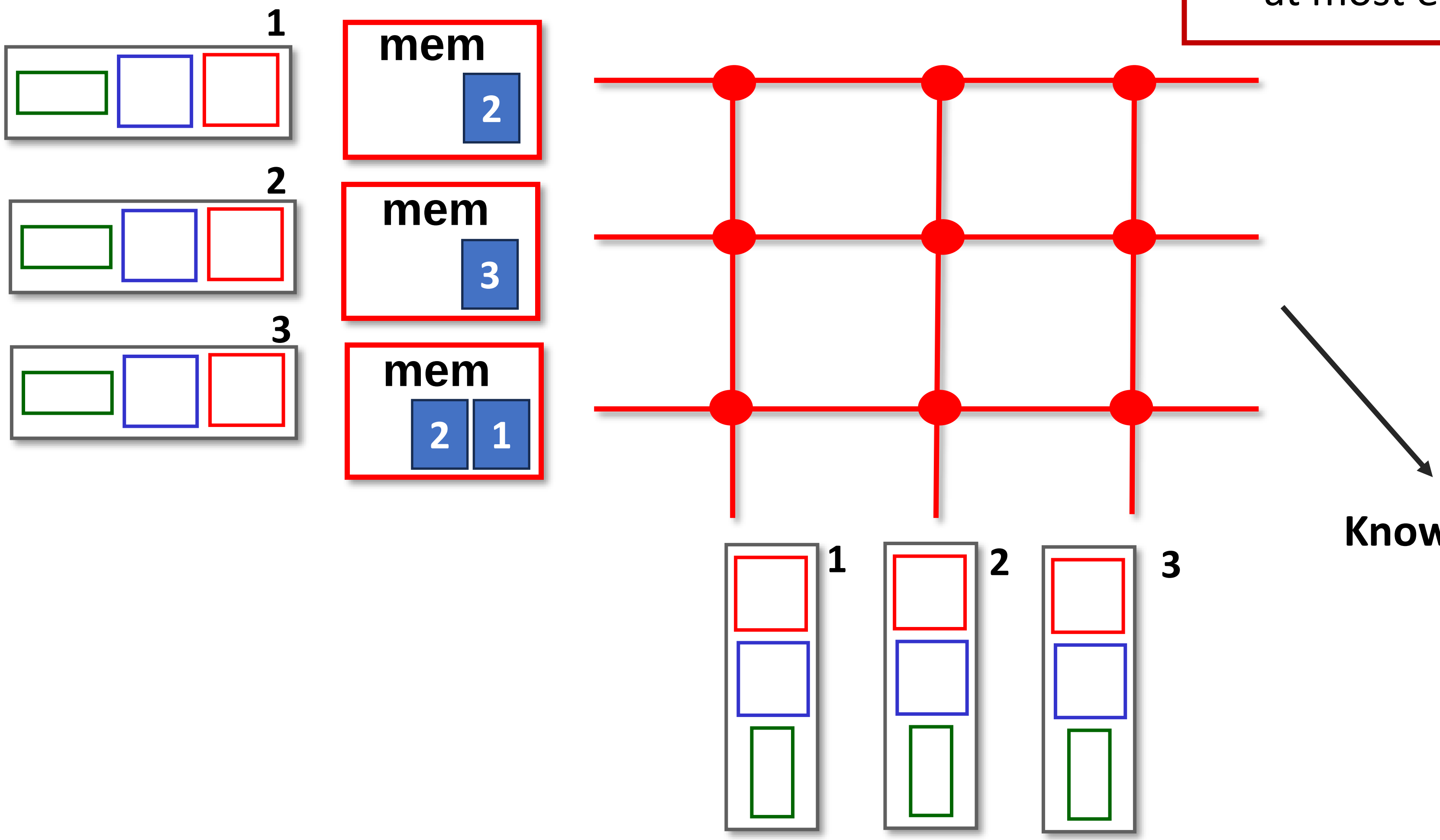


Known as the crossbar

Option 3 – Input queuing

In every tick

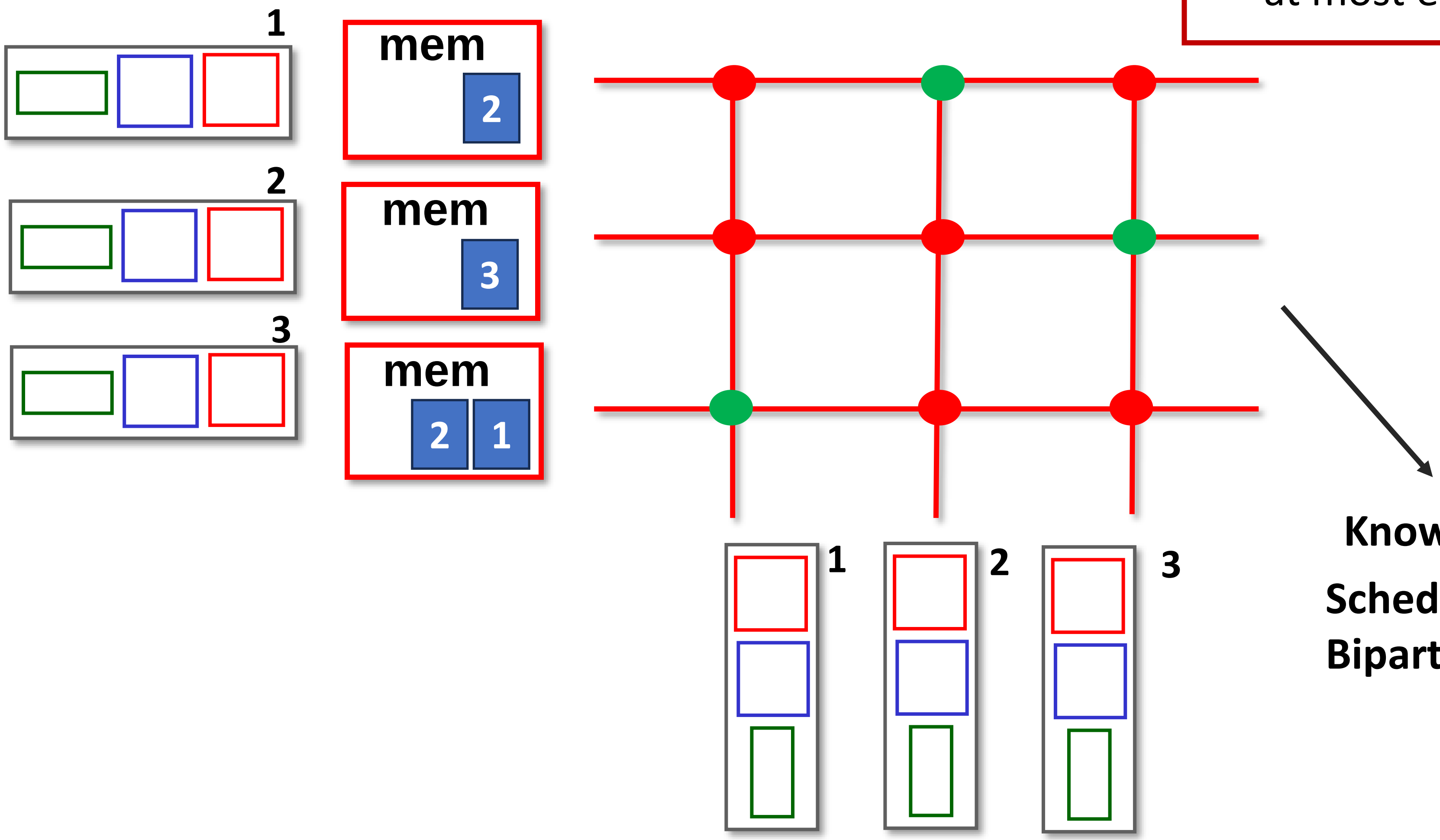
- Each input sends to at most one output
- Each output receives from at most each input



Known as the crossbar

Option 3 – Input queuing

In every tick
- Each input sends to at most one output
- Each output receives from at most each input

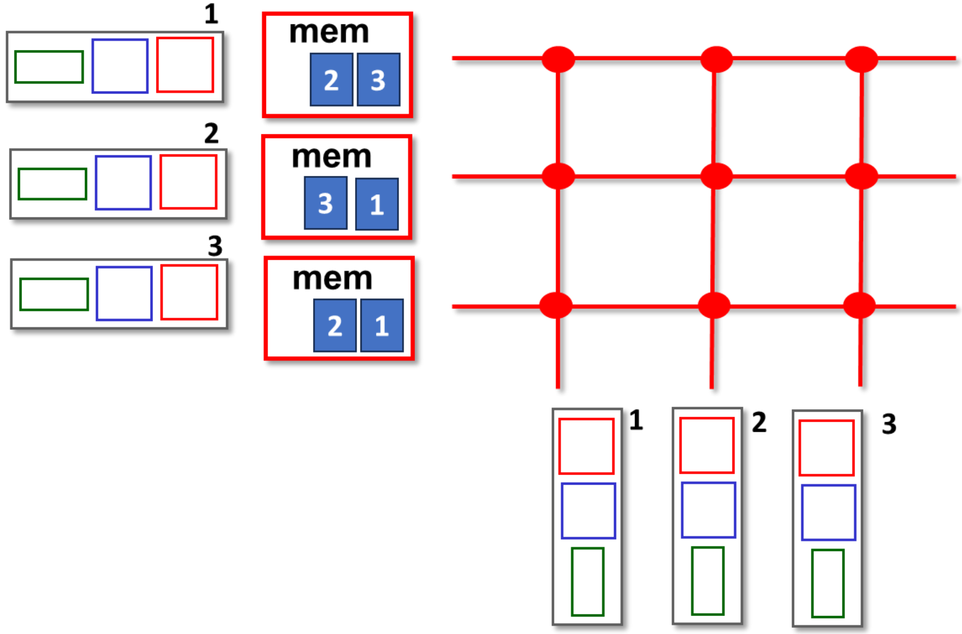


Known as the crossbar

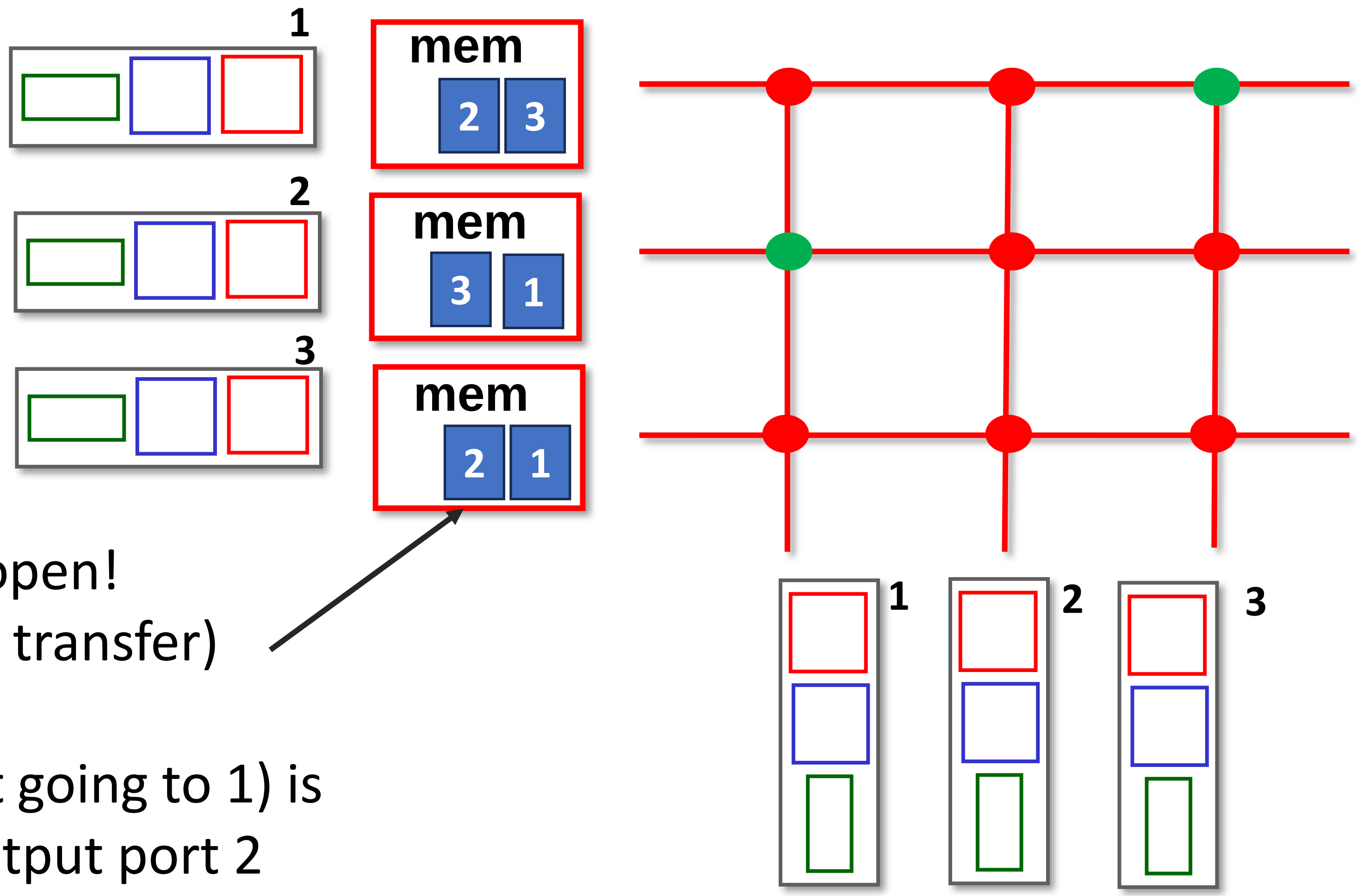
Scheduling the crossbar: Bipartite matching

Option 3 – Input queuing

- Pros:
  - For each memory, 1 enqueue and 1 dequeue per tick
- Cons:
  - Static allocation of memory resources to input ports
  - *Head of line blocking* (HoL blocking)



Head of line blocking

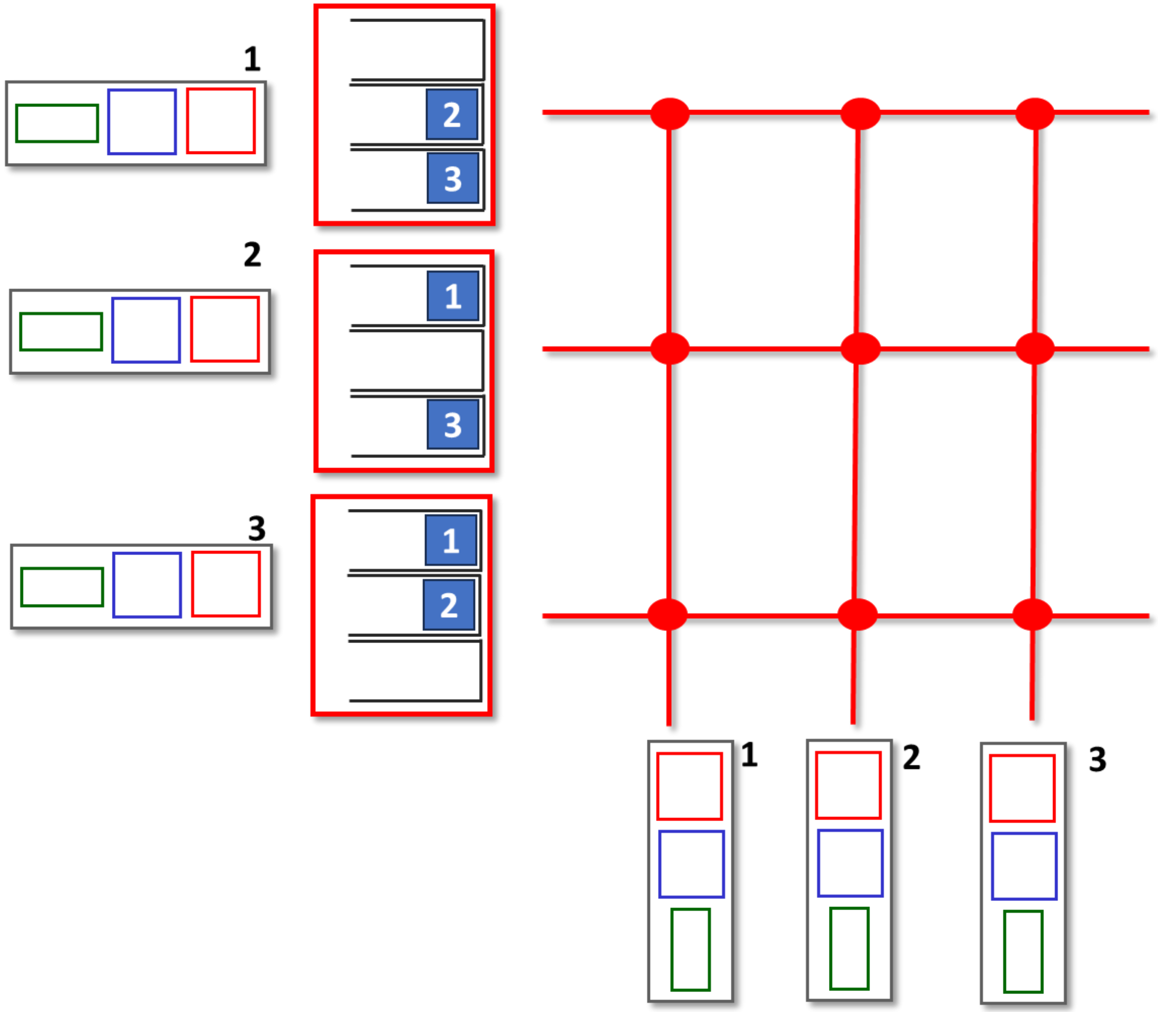


The “path” between 3 and 2 is open!
(does not collide with any other transfer)

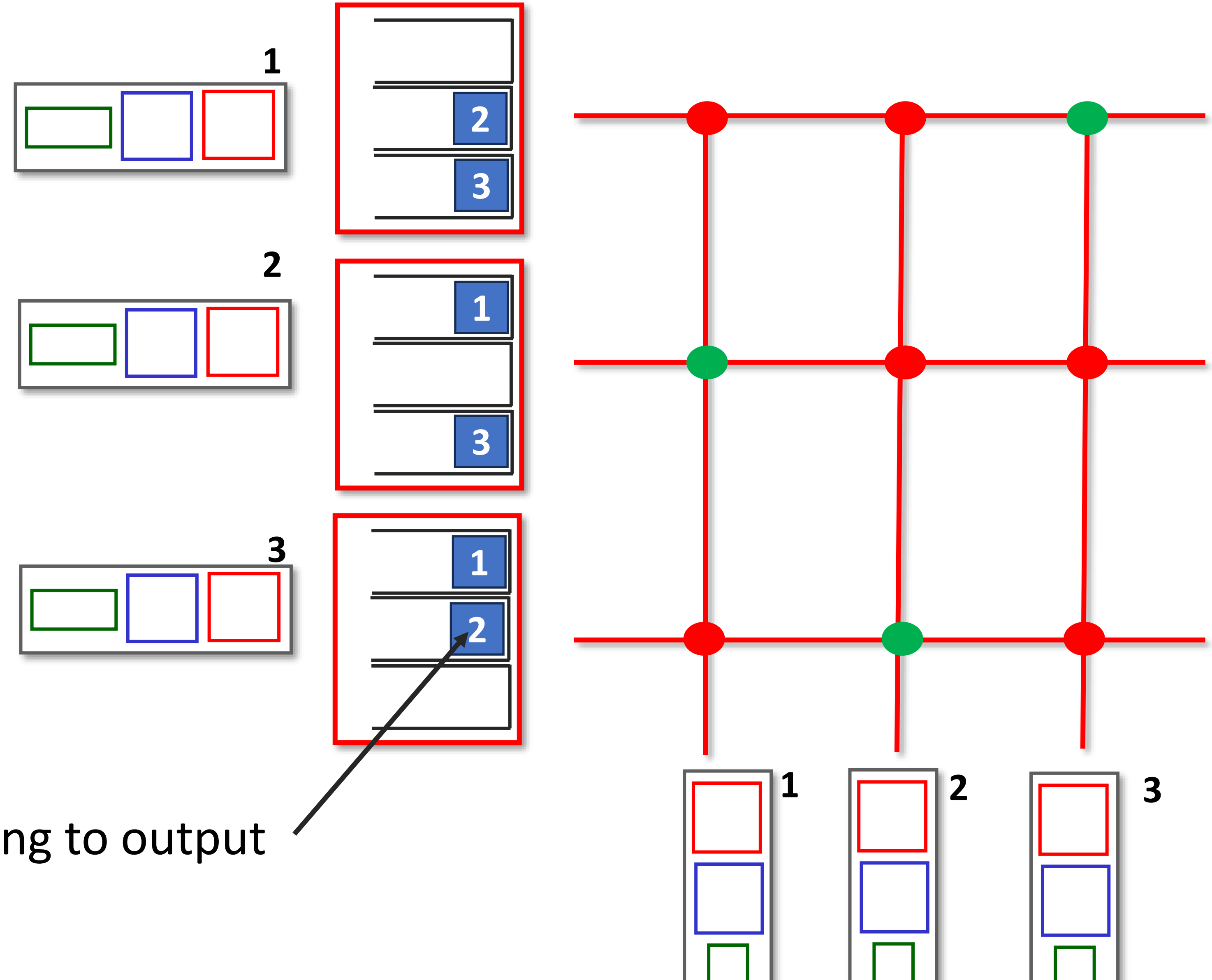
But, the *head of the line* (packet going to 1) is *blocking* this packet to get to output port 2

Revised Option 3 – Virtual output queuing

- Instead of one queue at each input port, have N queues at each input port, one for the packets destined to one output port.
- *Virtual output queue* (VOQ) j at input port i
  - is located in input port i's memory
  - buffers the packets entering from input port i and destined towards output port j

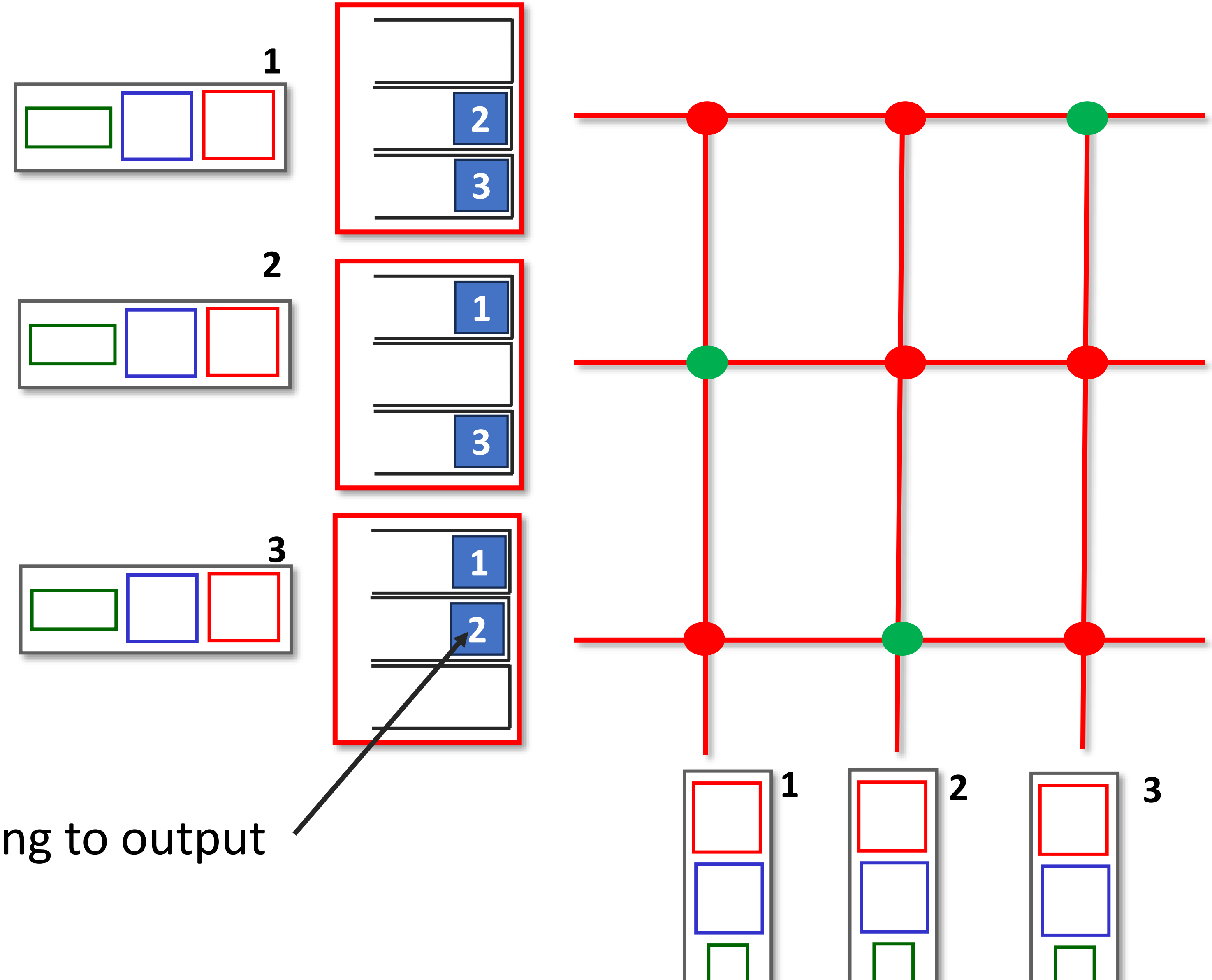


Revised Option 3 – Virtual output queuing



Not stuck behind the packet going to output port 1 anymore

Revised Option 3 – Virtual output queuing



Not stuck behind the packet going to output port 1 anymore

Make sure you know

- When queues form
- The differences of shared memory, output-queueing and input-queueing
- The pros and cons of each approach
- For input queueing approach
  - How crossbar works?
  - What head of line blocking is?
  - What virtual output queues are and how they work in conjunction with the crossbar?

Architecture trends

- Early architectures were shared memory
- Then moved towards output-queued architectures
- Then came input-queued architectures.

Architecture trends

- Today, there is a renewed interest in output-queued and shared memory architectures
- Data centers have *many* switches (100s of thousands)
- To keep the costs down, vendors have reduced the amount of memory available for buffering in these switches
  - Easier, e.g., compared to a WAN, to keep the queues shorter in DCs, specially with the help of congestion control algorithms.
- Easier to make smaller high-speed memory with multiple enqueues and/or dequeues per tick
- With output-queued or shared-memory architectures, no need for dealing with efficient scheduling of a crossbar.

Router architecture and buffer management

- Router architecture
  - Shared memory
  - Output queueing
  - Input queueing

- Buffer management and scheduling

Queue/Buffer management and scheduling

- Independent of where the queues are in the router architecture, there are some important questions:
  - *Buffer size*: How large should a buffer be?
  - *Queue management*: When the queue is full, which packet do we drop? What do we do when the queue starts building up?
  - *Packet scheduling*: Which packet in the queue gets dequeued first? Should it be first-in first-out? Something else?

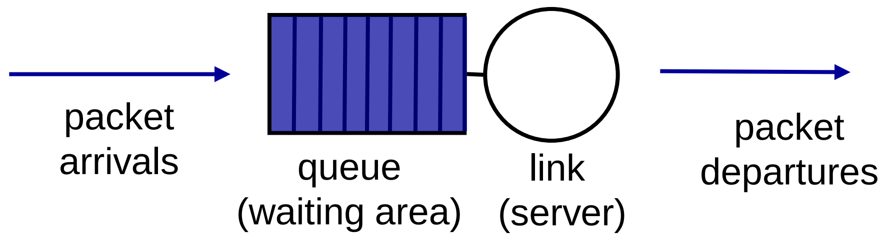
How large should a buffer be?

There is no easy answer:

- Too small: can't absorb bursts, keeps dropping packets
- Too large: can hurt performance
  - *buffer bloat:* When the buffer is too large, it will take a long time to fill up before a packet is dropped (however, TCP only realizes there is congestion when a packet is dropped and will not decrease its sending rate). In the meantime, all packets will experience increasing queueing delay.
  - Delay-based congestion control algorithms do better here.

Queue management – Drop policy

- When a new packet arrives to a full queue, which packet do we drop?
- *Tail drop*: drop arriving packet
- *Priority*: drop/remove based on priority
  - E.g., if the incoming packet has higher priority than a packet already in the queue, drop the lower priority packet and insert the incoming packet into the queue.



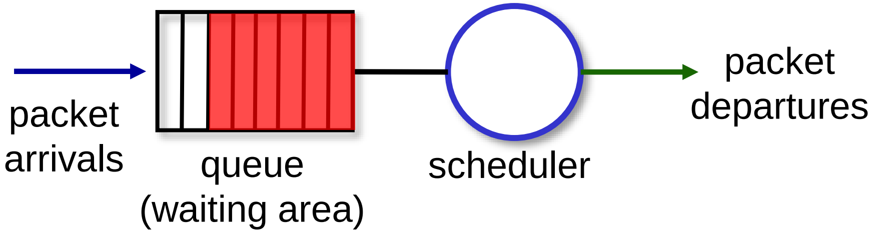
Queue management – Marking

- When the queue starts filling up, one strategy is to mark packets to signal the onset of congestion to the end points
- Recall our discussion about Early Congestion Notification (ECN) and its role in congestion control
- When should we start/stop marking packets?
- Which packets do we mark?
  - All packets after the queue size passes a threshold?
  - From the flow with the most packets in the queue?
  - …

Packet scheduling - FIFO

- So far, we have assumed that our queues are *first in first out (FIFO)*

Abstraction: queue



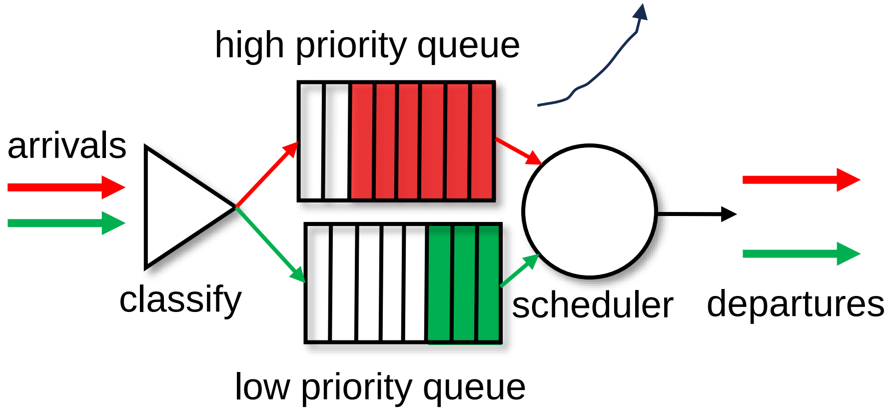
- But, there are other packet scheduling algorithms as well.

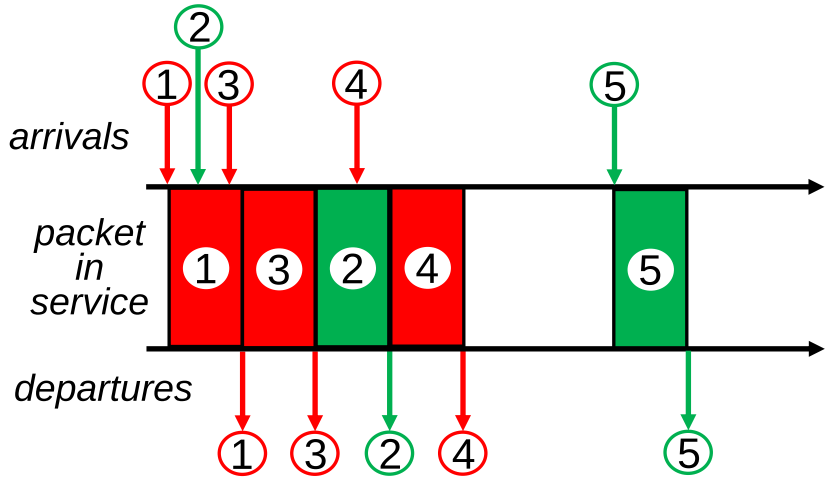
Packet scheduling: priority

Instead of one queue for all packets going to the same output port, there are two queues, one for each priority

Priority scheduling:

- arriving traffic classified to figure out priority
  - any header fields can be used for classification
- send packet from highest priority queue that has buffered packets
  - FIFO within priority class

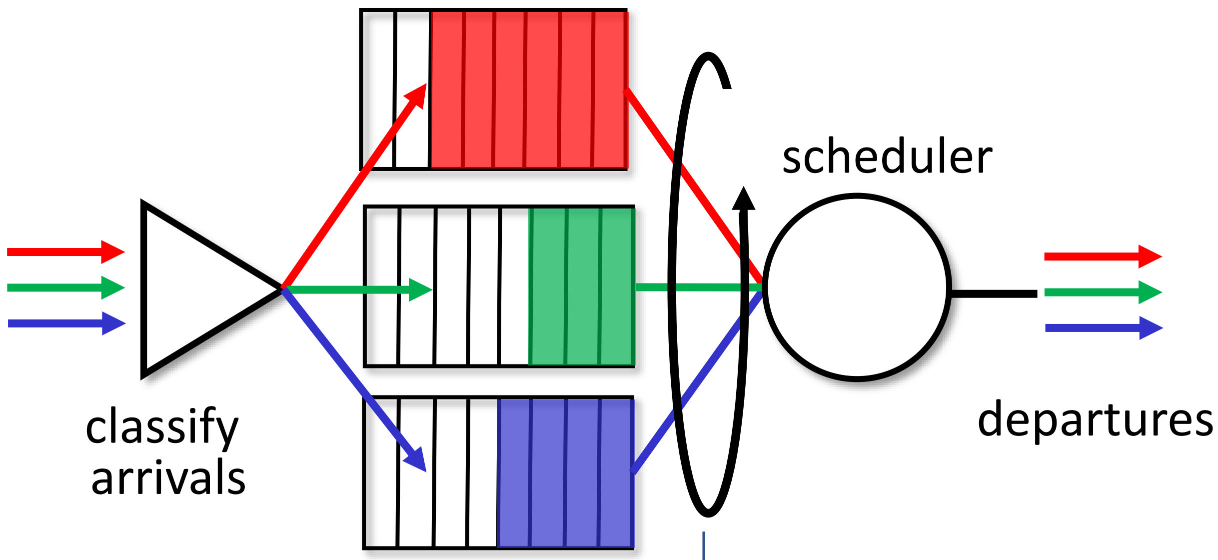




Packet scheduling: round robin

Round Robin (RR) scheduling:

- arriving traffic classified, queued by class
  - any header fields can be used for classification
- scheduler cyclically, repeatedly scans class queues, sending one complete packet from each class (if available) in turn



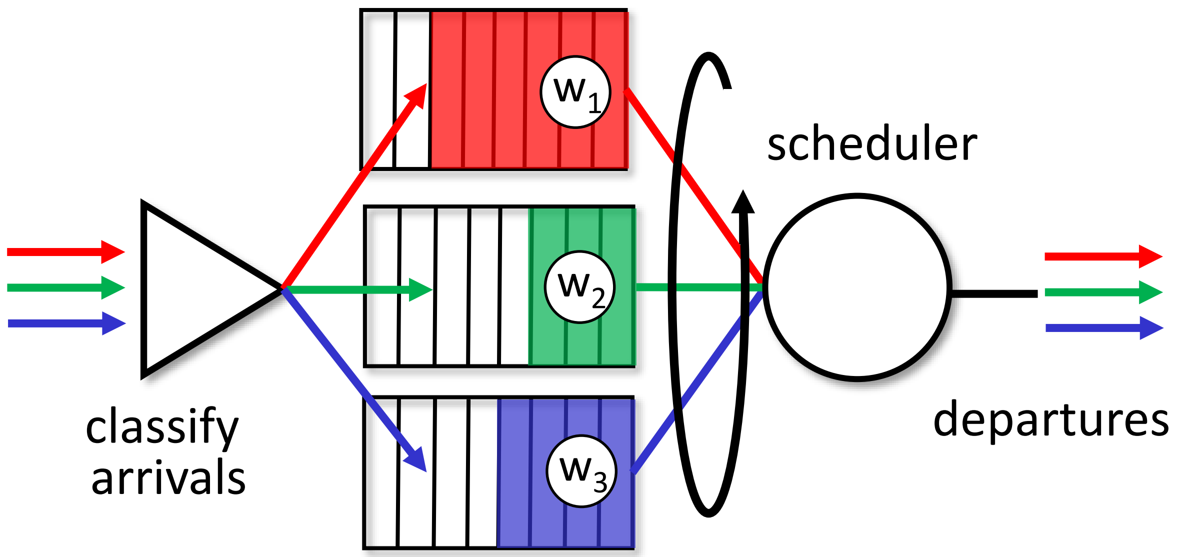
Packet scheduling: weighted fair queueing

Weighted Fair Queuing (WFQ):

- generalized Round Robin
- each class, *i*, has weight, w_i, and gets weighted amount of service in each cycle:

$$\frac{w_i}{\Sigma_j w_j}$$

- minimum bandwidth guarantee (per-traffic-class)



Make sure you know

- For exam purposes
  - No need to know much about architecture trends and strategies for setting queue sizes
- Know the space of different strategies for queue management
  - That is, the fact that there are different options for drop and marking policies
- Understand how FIFO, priority, and round robin packet scheduling.
  - Given a sequence of packets and the scheduling policy, you should be able to figure out the departures.