

# Lecture 21:

# Router Design

---

CSE 123: Computer Networks

Aaron Schulman



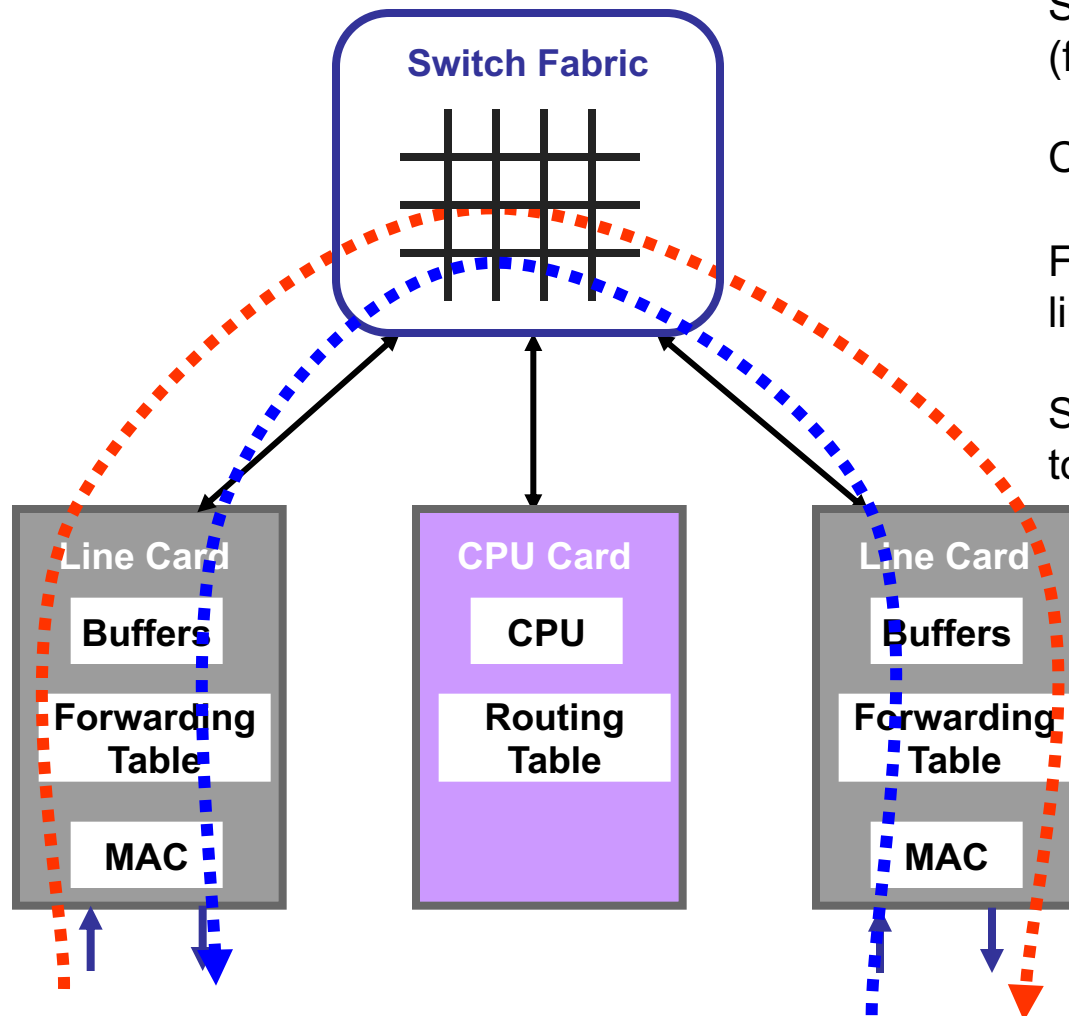


# Lecture 21

---

- Wrap up Interdomain routing
- Router implementation
  - ◆ Switching fabrics
  - ◆ Queueing

# Third Generation Routers (improved HW shared bus)



Shared interconnect  
(frequently crossbar)

Centralized scheduler

Full forwarding table in  
line card

Switches fixed-sized cells  
to improve performance

# What if input rate exceeds output rate?

## Output queuing



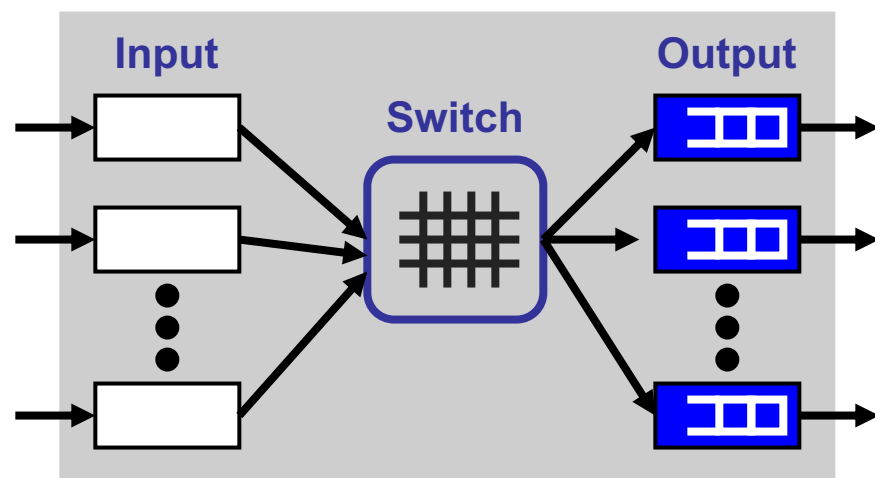
- Output interfaces buffer packets

- Pro

- ◆ Simple queuing algorithms
- ◆ Single congestion point

- Con

- ◆ N inputs may send to the same output
- ◆ Requires **speedup** of N
  - » Output ports must be N times faster than input ports



# What if input rate exceeds output rate?

## Input queuing



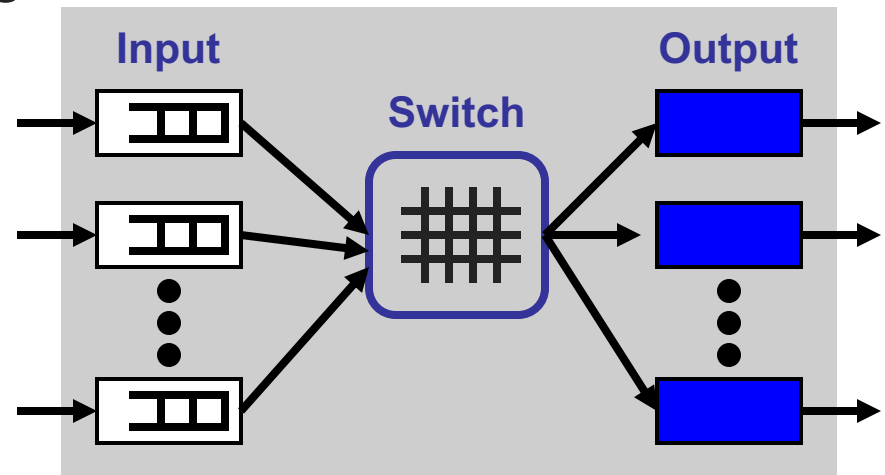
- Input interfaces buffer packets

- Pro

- ◆ Single congestion point
- ◆ Simple to design algorithms

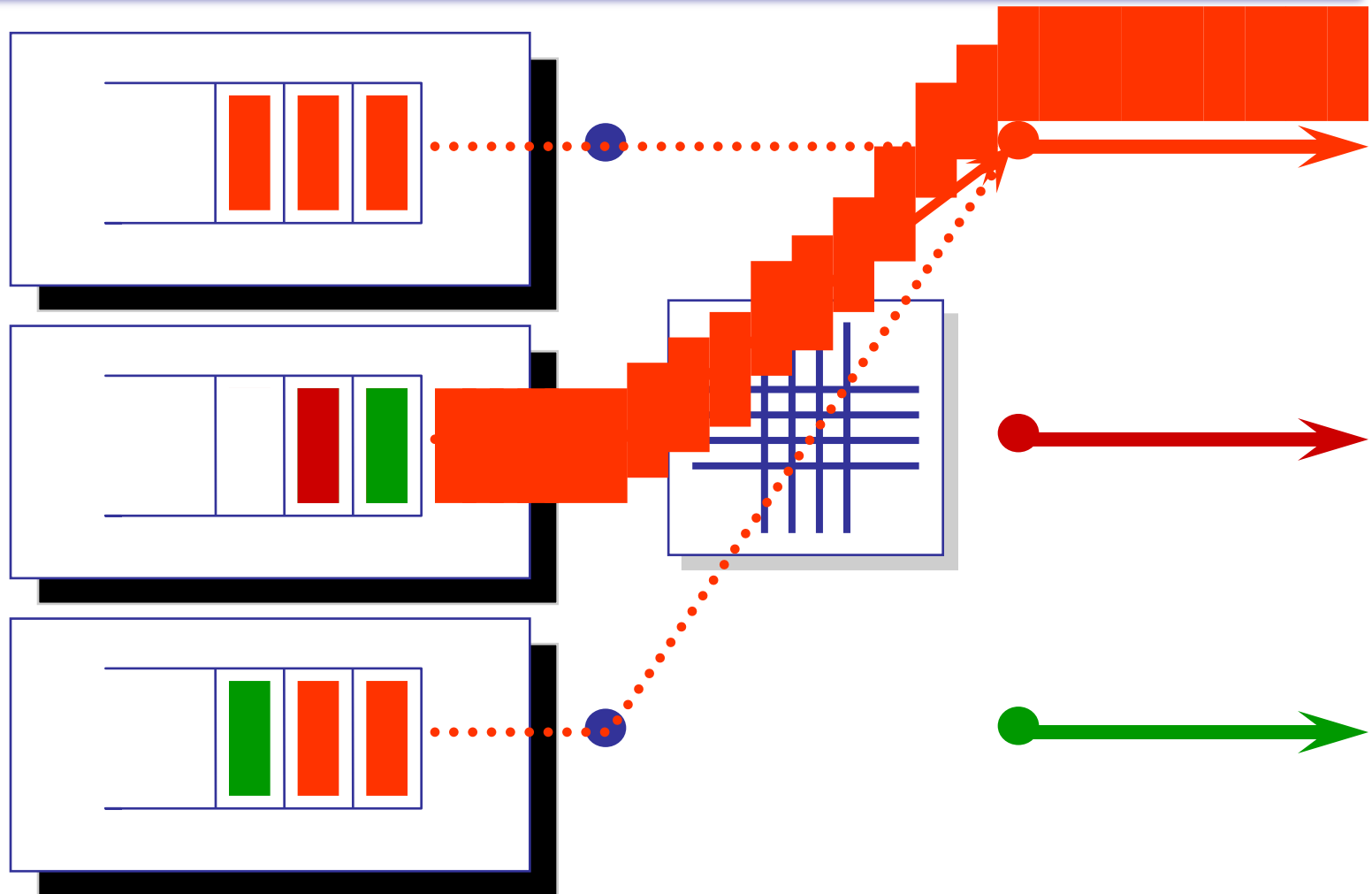
- Con

- ◆ Must implement flow control to outputs
- ◆ Low output utilization due to Head-of-Line (HoL) Blocking





# Head-of-Line Blocking



# What if input rate exceeds output rate?

## IQ + Virtual Output Queuing



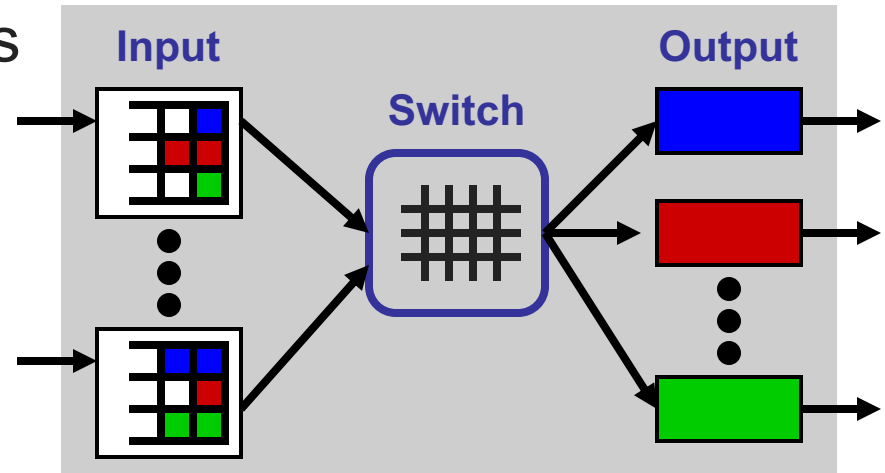
- Input interfaces buffer packets in per-output virtual queues

- Pro

- ◆ Solves blocking problem

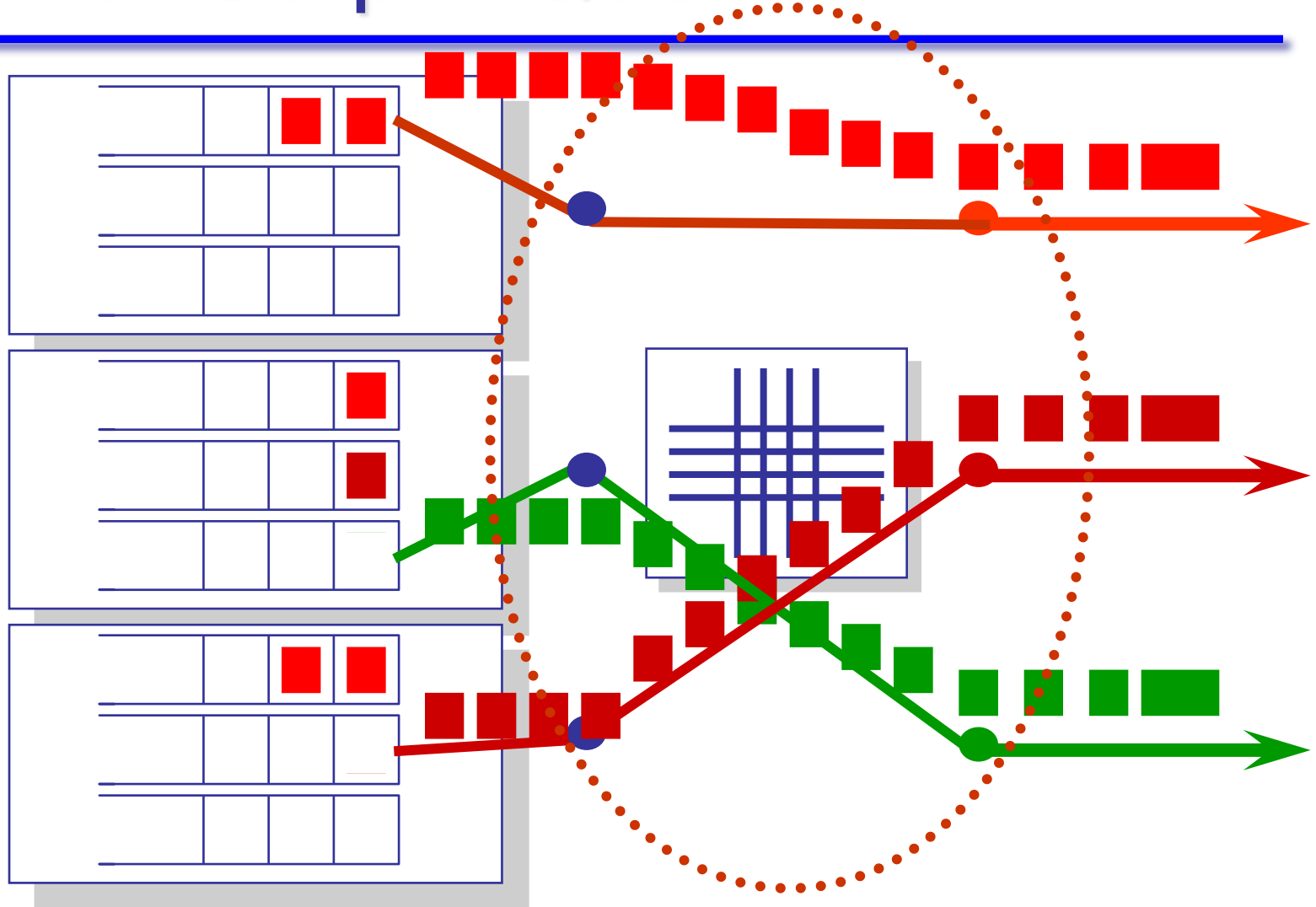
- Con

- ◆ More resources per port
- ◆ Complex arbiter at switch
- ◆ Still limited by input/output contention (scheduler)





# Virtual Output Queues





# Switch scheduling

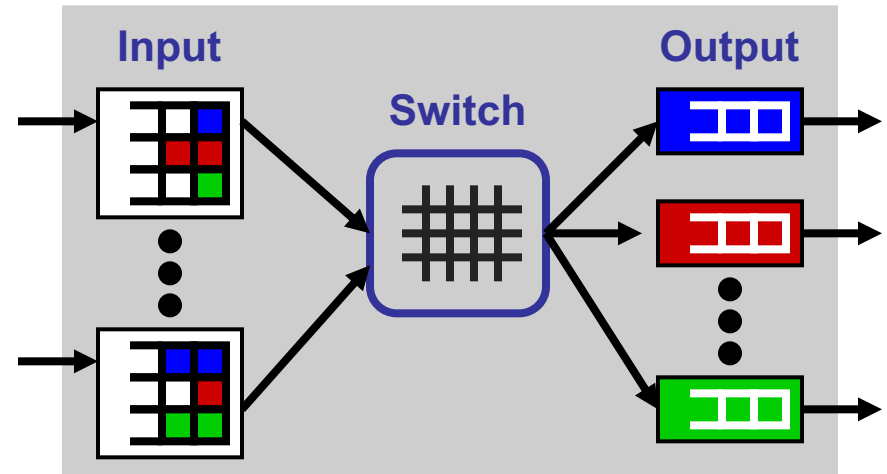
---

- Problem
  - ◆ Match inputs and outputs
  - ◆ Resolve contentions, no packet drops
  - ◆ Maximize throughput
  - ◆ Do it in constant time...
- If traffic is uniformly distributed its easy
  - ◆ Lots of algorithms (approximate matching)
- Seminal result (Dai et al, 2000)
  - ◆ Maximal size matching + *speedup* of two guarantees  
100% utilization for most traffic assumptions



# Typical high-performance router

- IQ + VoQ + OQ
  - ◆ Central scheduler
  - ◆ Fixed-sized internal pkt cells
- Pro
  - ◆ Can achieve utilization of 1
  - ◆ Can scale to  $> \text{Tb/s}$
- Con
  - ◆ Multiple congestion points
  - ◆ Complexity

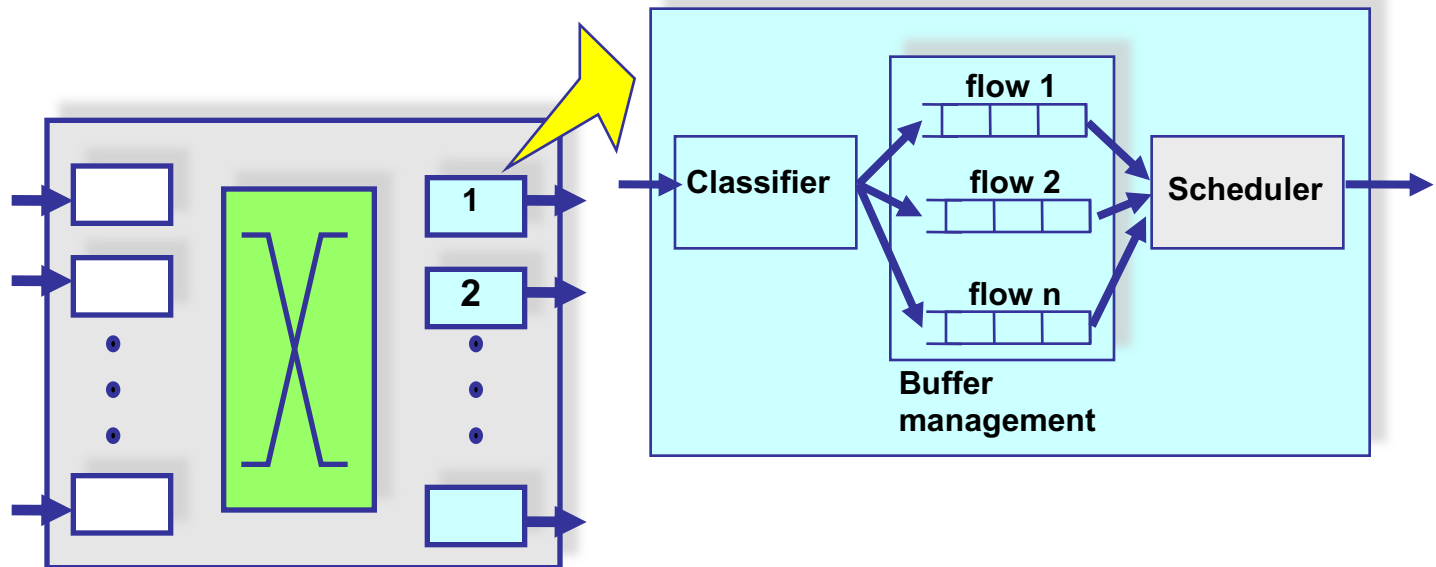


- But what do we do when queues are full (links are saturated)



# Key Router Challenges

- **Scheduling**: which packet to transmit next?
- **Buffer management**: which packet to drop when?
  - ◆ We only have finite-length queues





# Basic Buffer Management

---

- **FIFO + drop-tail**
  - ◆ Simplest choice
  - ◆ Used widely in the Internet
- **FIFO (first-in-first-out)**
  - ◆ Implies single class of traffic
- **Drop-tail**
  - ◆ Arriving packets get dropped when queue is full regardless of flow or importance
- **Important distinction:**
  - ◆ FIFO: scheduling discipline
  - ◆ Drop-tail: drop policy



# FIFO/Drop-Tail Problems

---

- Leaves responsibility of preventing congestion completely to the edges
  - ◆ Transport protocols will need to make sure queues in routers are not so full that they drop, (called congestion control)
- Does not separate between different flows
  - ◆ Packet can be dropped regardless of what TCP or UDP flow it is part of
- **No policing:** send more packets → get more service
- Synchronization: end hosts react to same problems

# Active Queue Management

---



- Design active router queue management to aid in reducing the congestion at routers
- Why?
  - ◆ Router has unified view of queuing behavior
  - ◆ Routers see actual queue occupancy (distinguish queue delay and propagation delay.
  - ◆ Routers can see if congestion is transient



# Design Objectives

---

- Keep throughput high and delay low
  - ◆ High power (throughput/delay)
- Accommodate bursts
- Queue size should reflect ability to accept bursts rather than steady-state queuing (that's why they are there!)
- Help with transport protocol (e.g., TCP) performance with minimal hardware changes in router

# Random Early Detection (RED)

---



- Detect incipient congestion
- **Assume hosts respond to lost packets**
  - ◆ We know they will retransmit based on losses
  - ◆ Soon we will see **losses will also make them slow down!**
- Avoid window/end host synchronization
  - ◆ Randomly mark packets
- Do not bias against bursty traffic

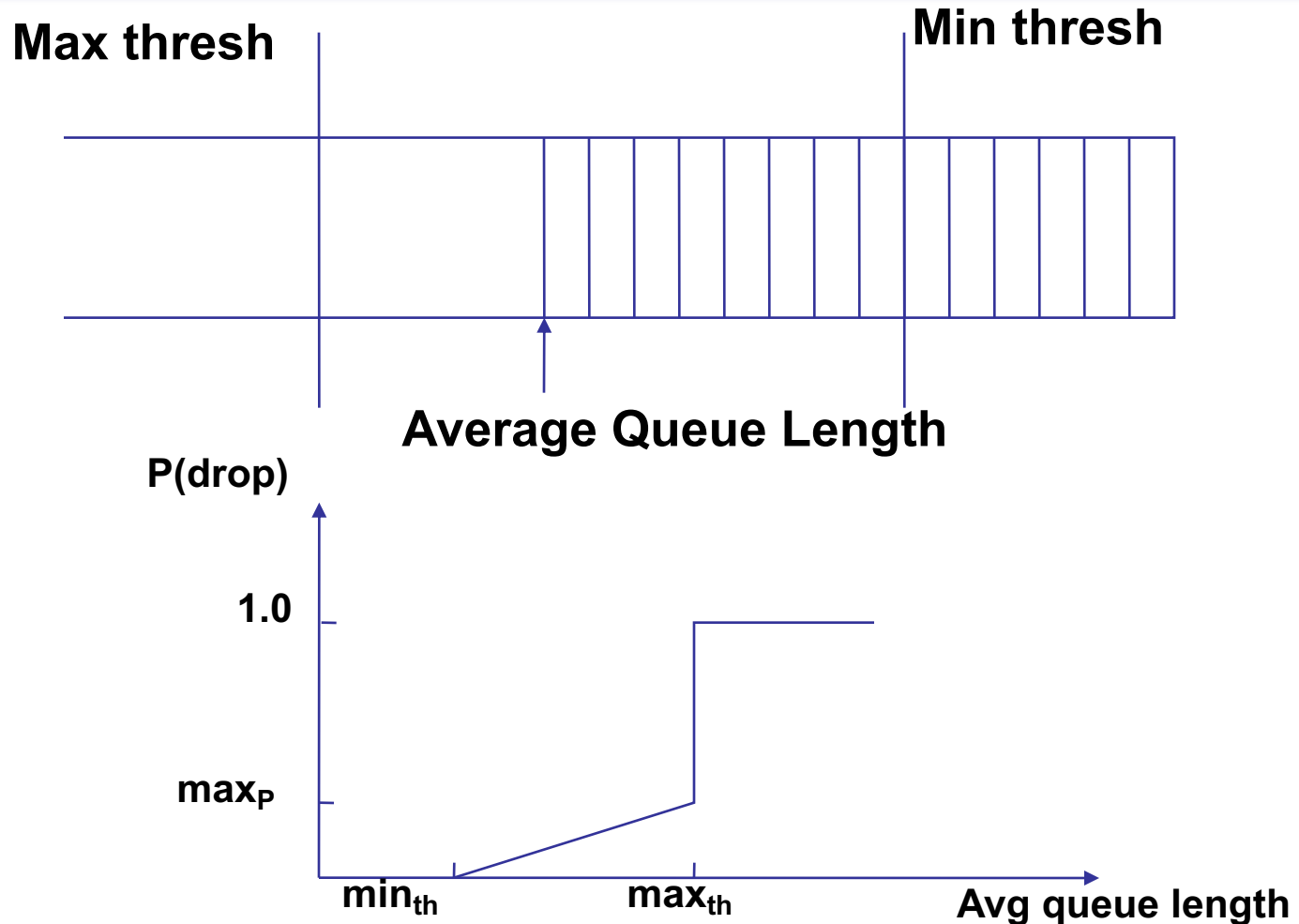


# RED Algorithm

---

- Maintain running average of queue length in router
- If  $\text{avg} < \text{min}_{\text{th}}$  do nothing
  - ◆ Low queuing, send packets through
- If  $\text{avg} > \text{max}_{\text{th}}$ , drop packet
  - ◆ Protection from misbehaving sources
- Else drop/mark packet in a manner proportional to queue length
  - ◆ Dropping notifies sources of incipient congestion
  - ◆ Q: Dropping vs Marking (Explicit Congestion Notification).  
Why drop to notify of congestion rather than mark in header?

# RED Operation



# Scheduling

---



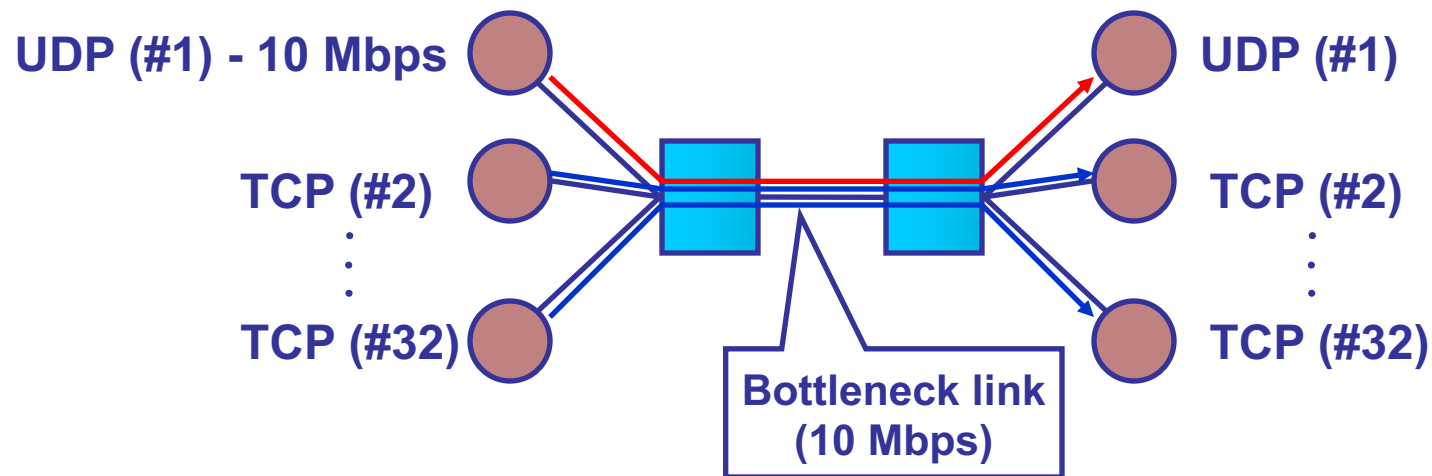
- So far we've done **traffic policing**
  - ◆ Limit the rate of flows regardless of the load in the network
- Routers also need **scheduling**
  - ◆ Dynamically allocate resources when multiple flows compete
  - ◆ Give each “flow” (or src/destination pair) own queue (at least theoretically)
- Weighted fair queuing
  - ◆ Proportional-share scheduling
  - ◆ Schedule round-robins among queues in proportion to some weight parameter



# Example with contending hosts

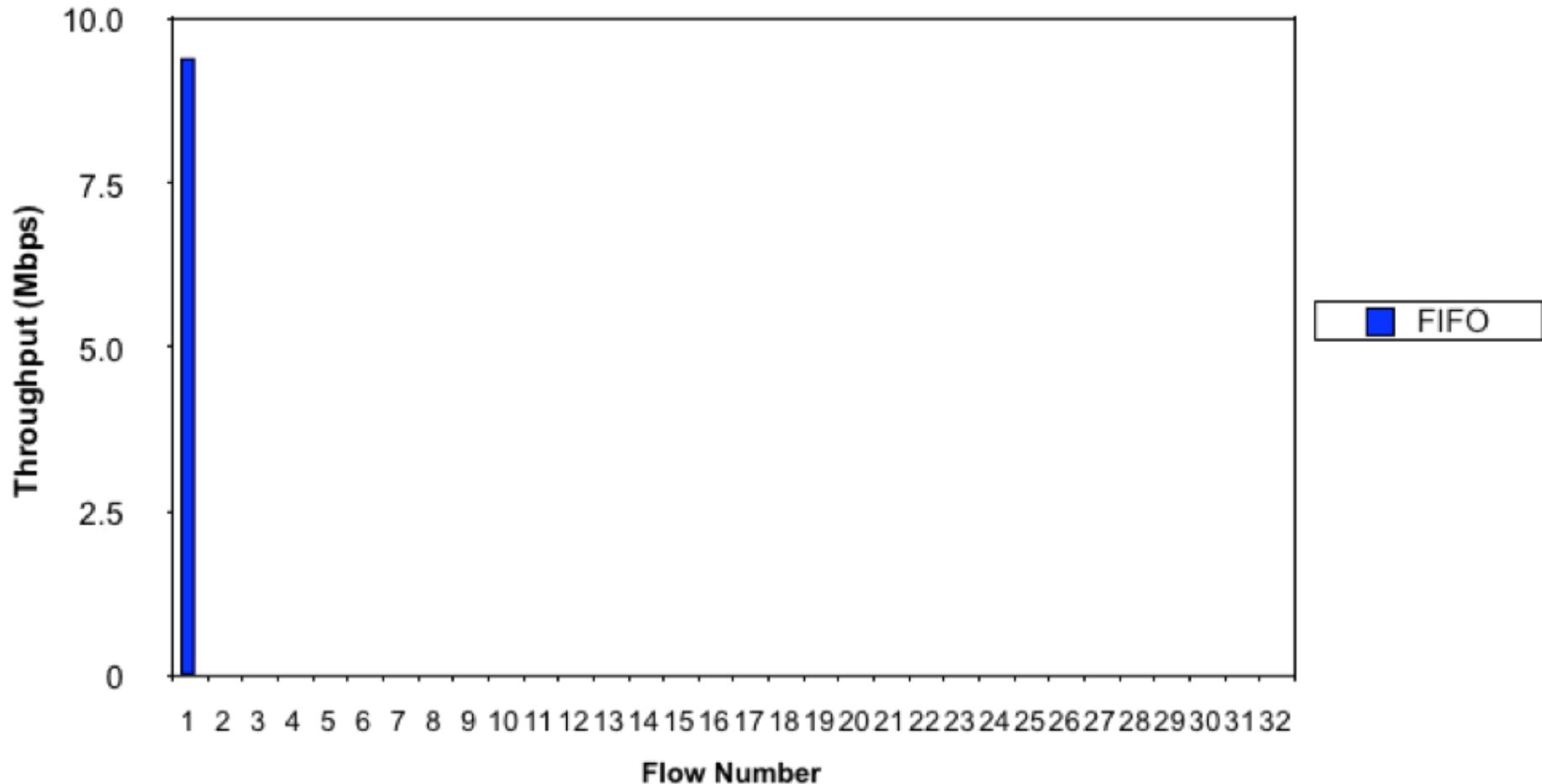
1 UDP (10 Mbps) and 31 TCPs sharing 10 Mbps  
(Note: *TCP will throttle due to congestion control*)

Will discuss this later

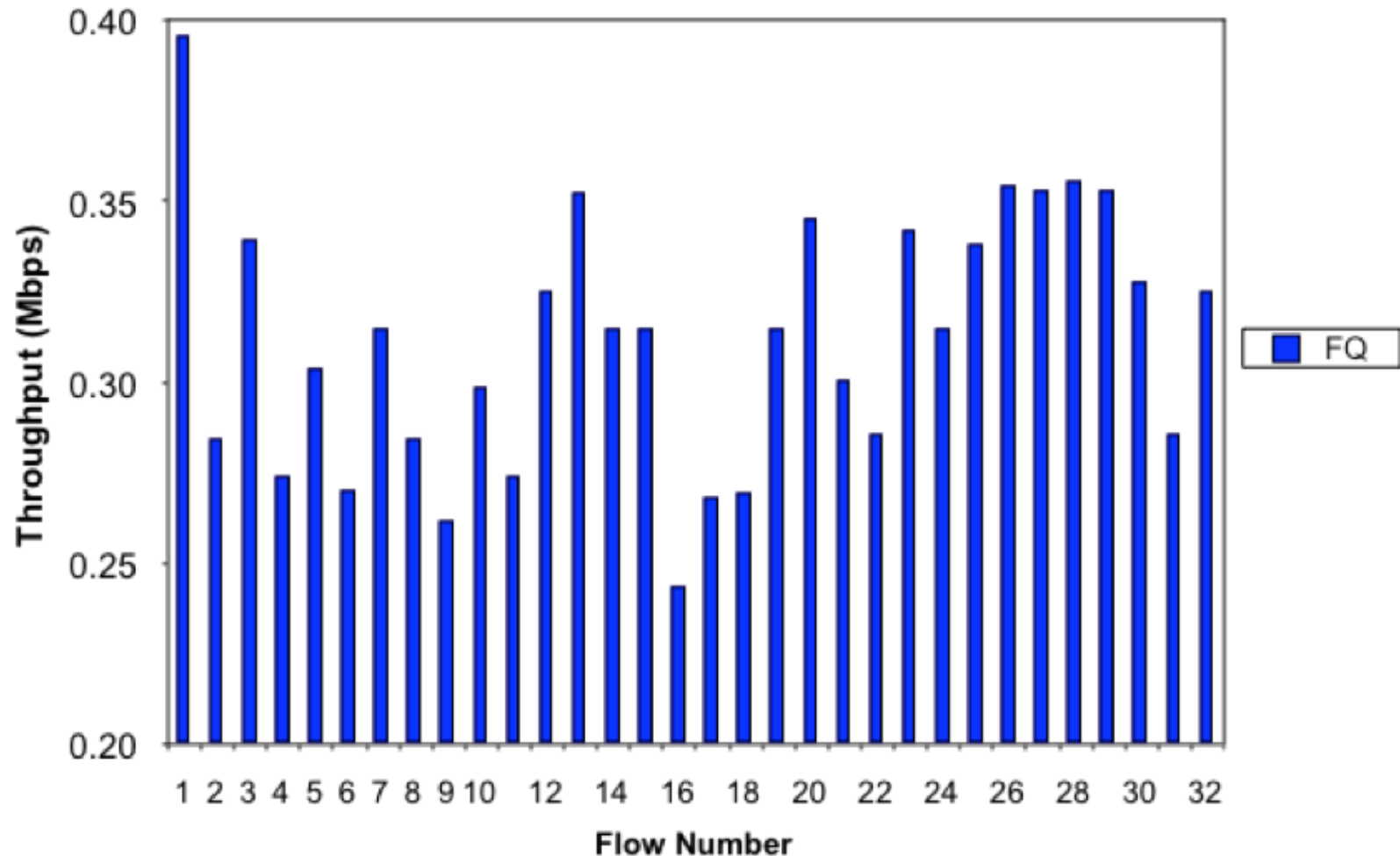




# UDP vs. TCP w/FIFO

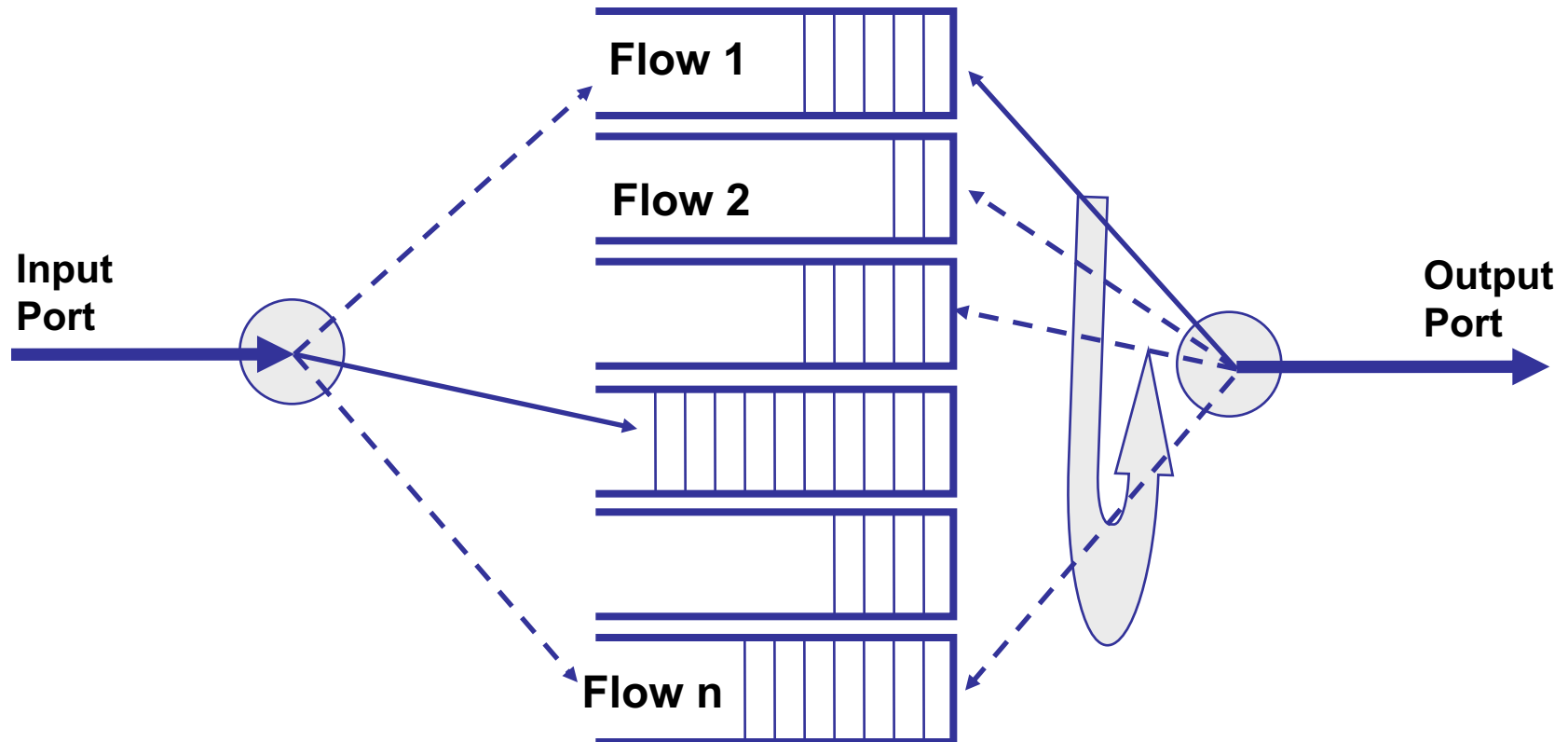


# TCP vs. UDP w/Fair Queuing





# (Weighted) Fair Queuing





# Fair Queuing

---

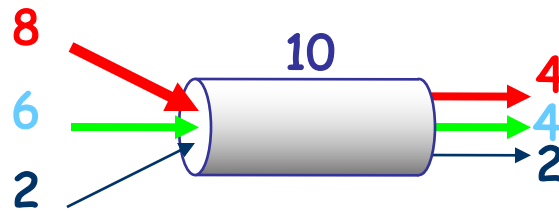
- Maintain a queue for each flow
  - ◆ What is a flow?
    - » We approximate this with the **5 tuple**
    - » (IP src, IP dst, port src, port dst, protocol)
- Implements **max-min fairness**: each flow receives  $\min(r_i, f)$ , where
  - ◆  $r_i$  – flow arrival rate
  - ◆  $f$  – link fair rate (see next slide)
- **Weighted Fair Queuing** (WFQ) – associate a weight with each flow to divvy bandwidth up non-equally



# Fair Rate Computation

- If link congested, compute  $f$  such that

$$\sum_i \min(r_i, f) = C$$



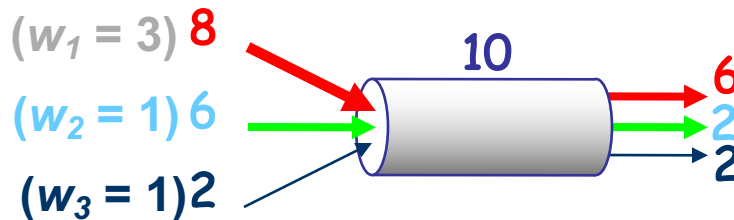
$f = 4$ :  
 $\min(8, 4) = 4$   
 $\min(6, 4) = 4$   
 $\min(2, 4) = 2$



# Weighted Fair Queuing

- Associate a weight  $w_i$  (*priority*) with each flow  $I$
- If link congested, compute  $f$  such that

$$\sum_i \min(r_i, f \times w_i) = C$$



$f = 2$ :

$$\begin{aligned} \min(8, 2 \times 3) &= 6 \\ \min(6, 2 \times 1) &= 2 \\ \min(2, 2 \times 1) &= 2 \end{aligned}$$

Flow  $i$  is guaranteed to be allocated a rate  $\geq w_i \cdot C / (\sum_k w_k)$



If  $\sum_k w_k \leq C$ , flow  $i$  is guaranteed to be allocated a rate  $\geq w_i$

# WFQ Implementation: Fluid Flow

---

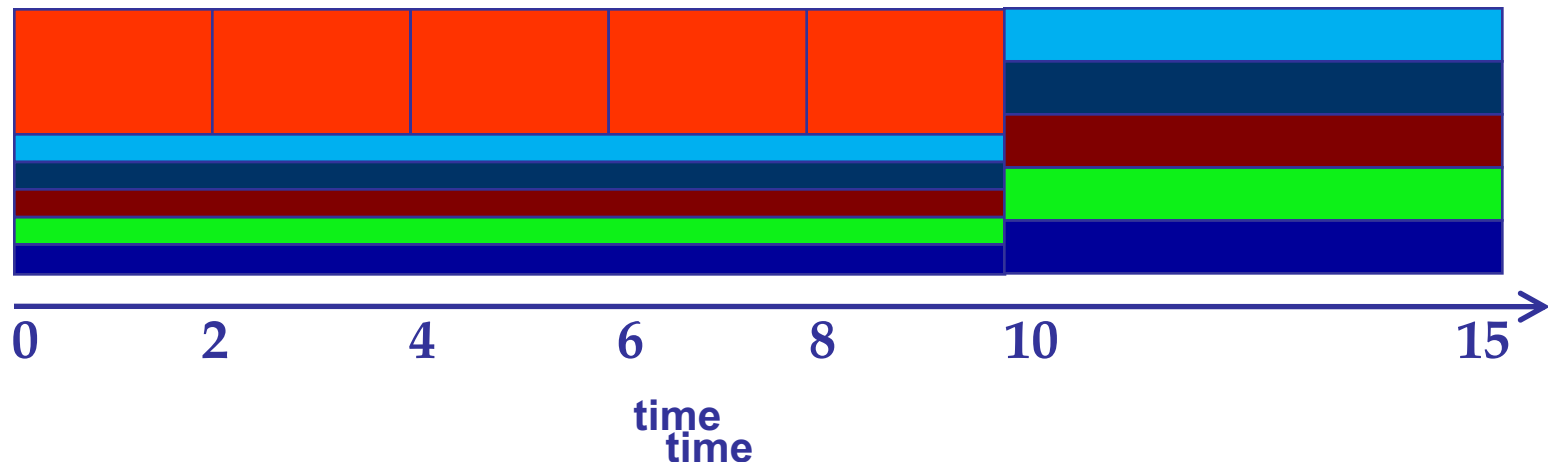
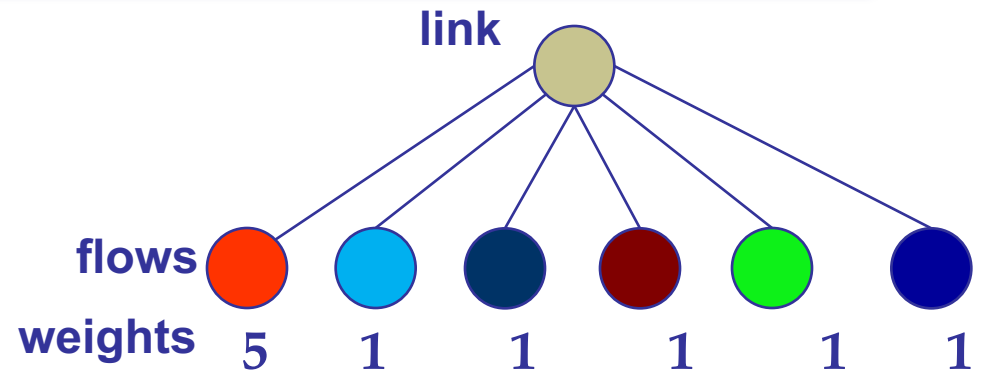


- It's tricky to implement WFQ when there are different packet sizes in different queues.
  - ◆ Can only serve one packet at a time.
- Simplification: Flows can be served one bit at a time
- WFQ can be implemented using bit-by-bit weighted round robin
  - ◆ During each round from each flow that has data to send, send a number of bits equal to the flow's weight



# Fluid Flow Example

- **Red flow** has packets backlogged between time 0 and 10
- Other flows have packets continuously backlogged
- All packets have the same size



# Packet-Based Implementation

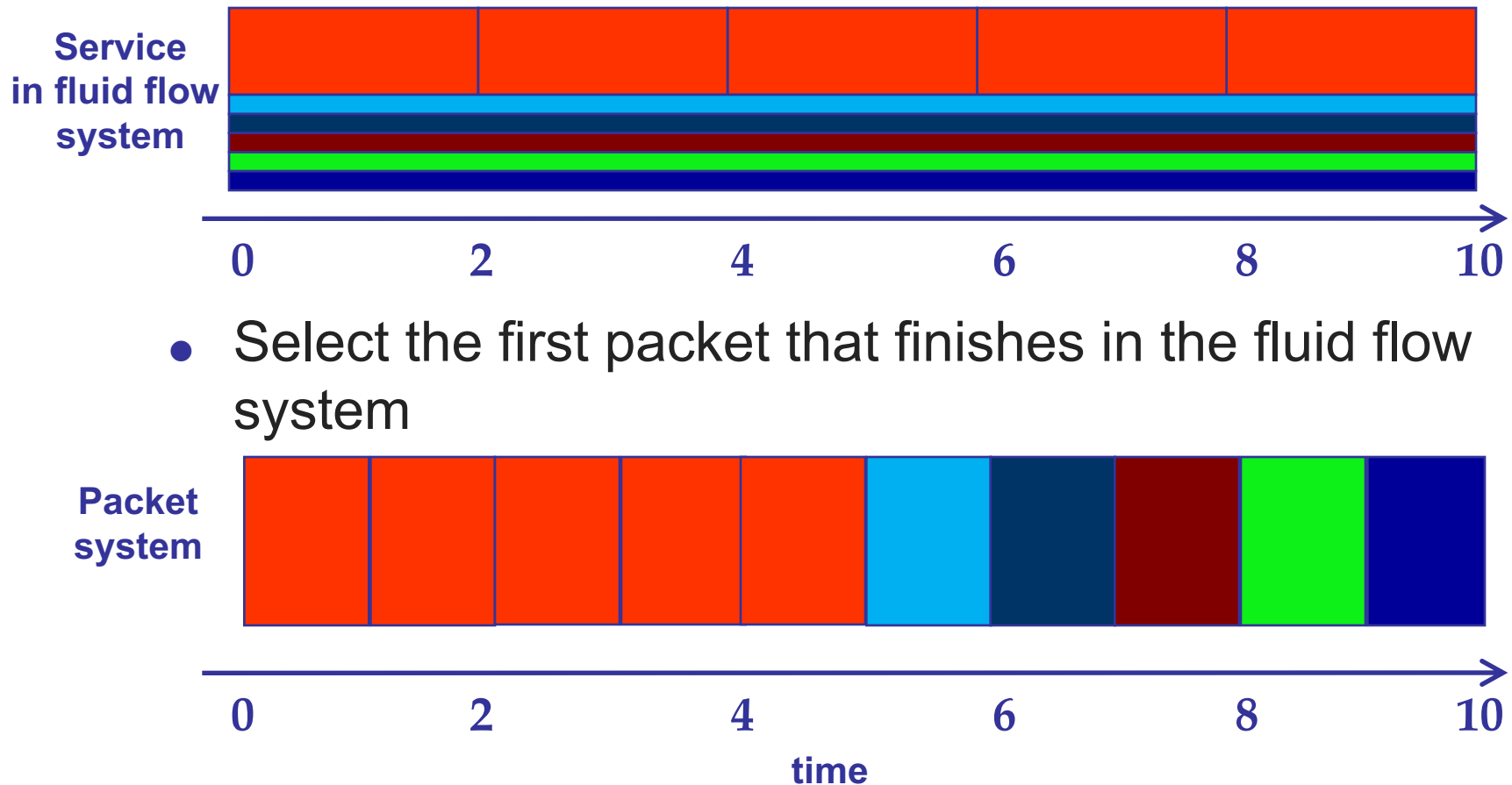
---



- Packet (real) system can't do bit-by-bit splitting: packet transmission cannot be preempted. Why?
- Issue: flows can get more bandwidth by sending bigger packets
- Solution: serve packets in the order in which they would have finished being transmitted in the fluid flow system



# Packet-Based Example

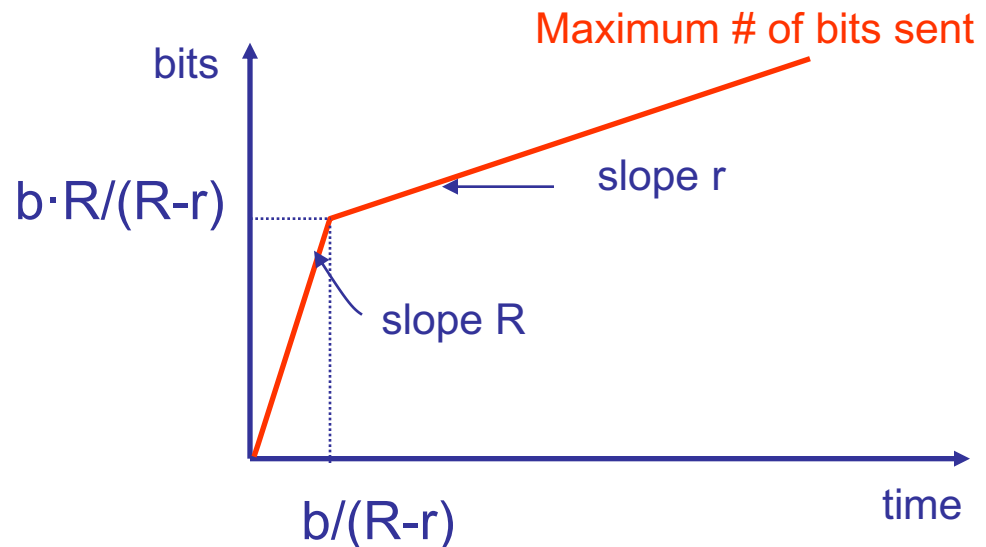
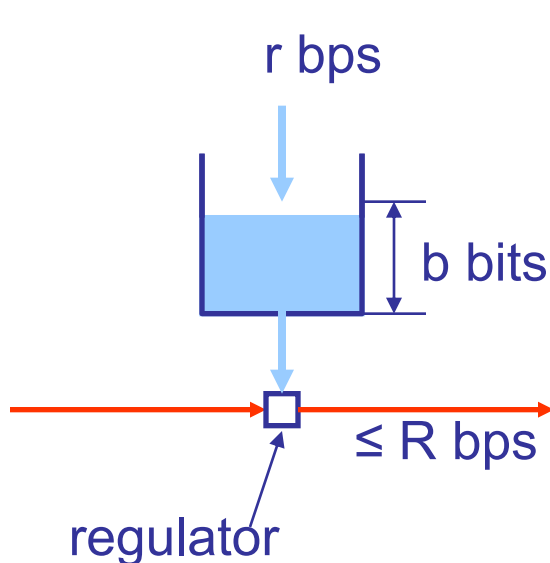


# Token Bucket

(similar to WFQ, but doesn't allocate spare capacity)



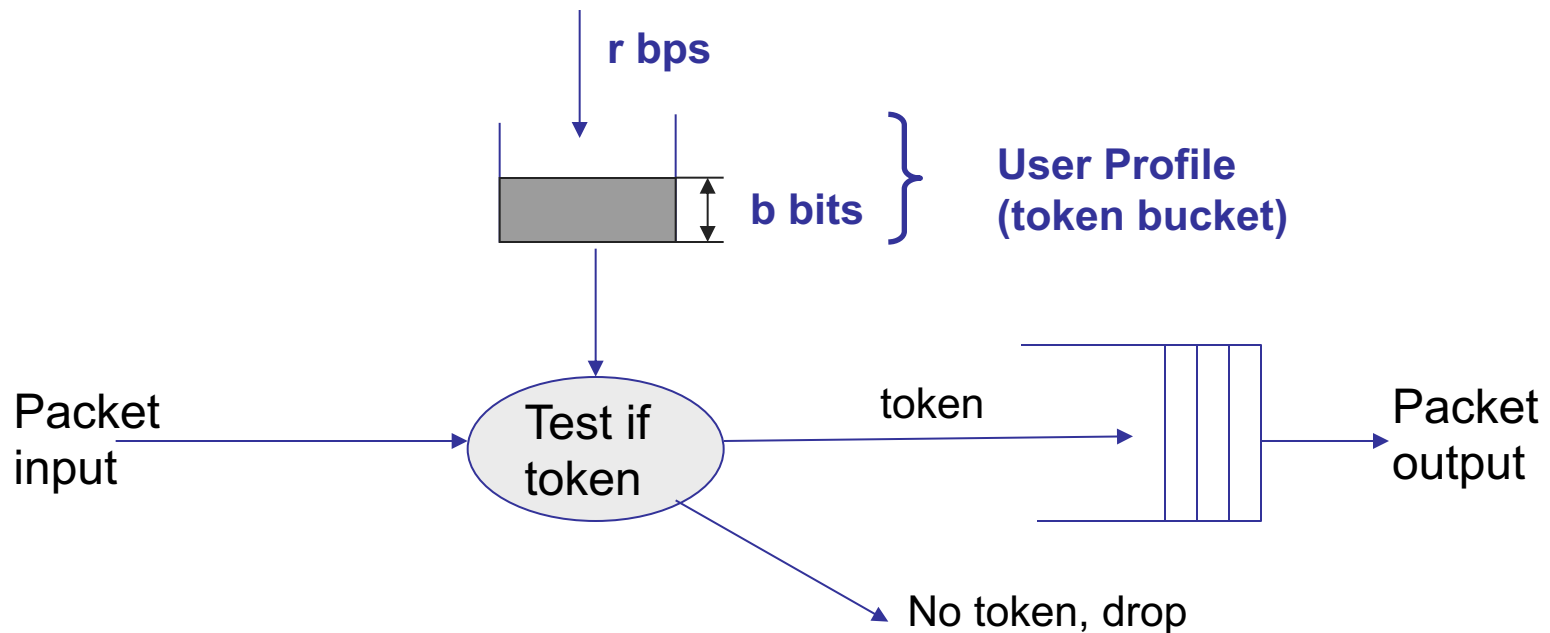
- Parameters
  - ◆  $r$  – average rate, i.e., rate at which tokens fill the bucket
  - ◆  $b$  – bucket depth (limits size of burst)
  - ◆  $R$  – maximum link capacity or peak rate (optional parameter)
- A bit can be transmitted only when a token is available





# Traffic Policing

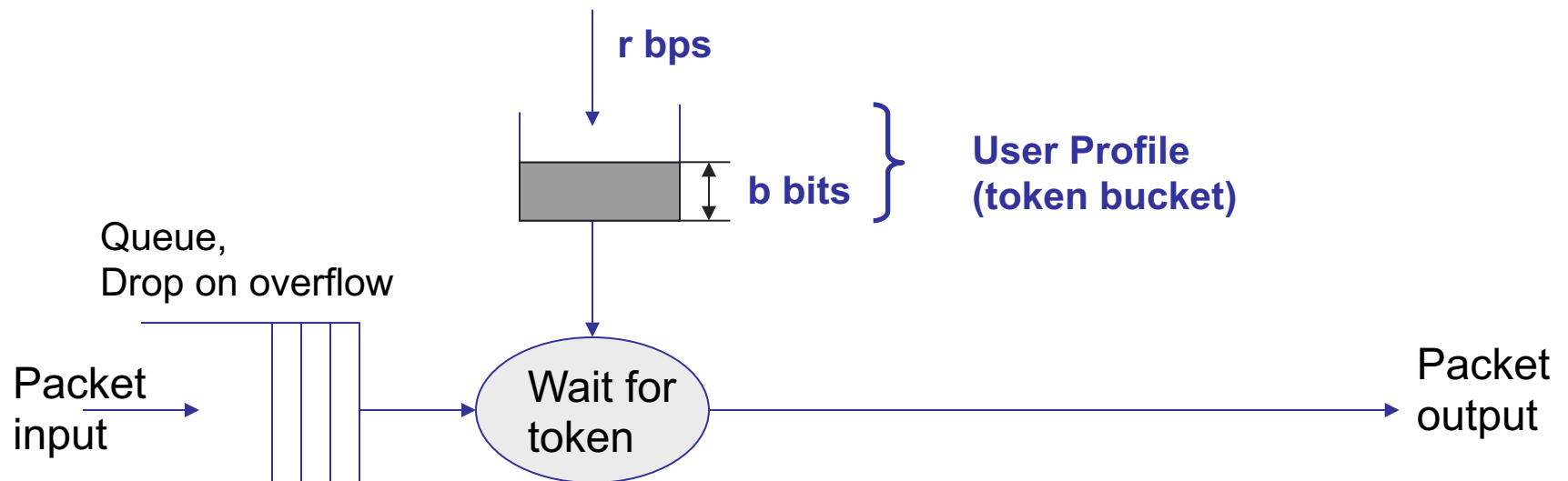
- Drop packets that don't meet **flow policy**
- Output limited to average of  $r$  bps and bursts of  $b$





# Traffic Shaping

- Shape (queue) packets according to flow policy
- Output limited to average of  $r$  bps and bursts of  $b$





# For Next Time

---

- Read P&D 6.2
- Get started on Project 4!