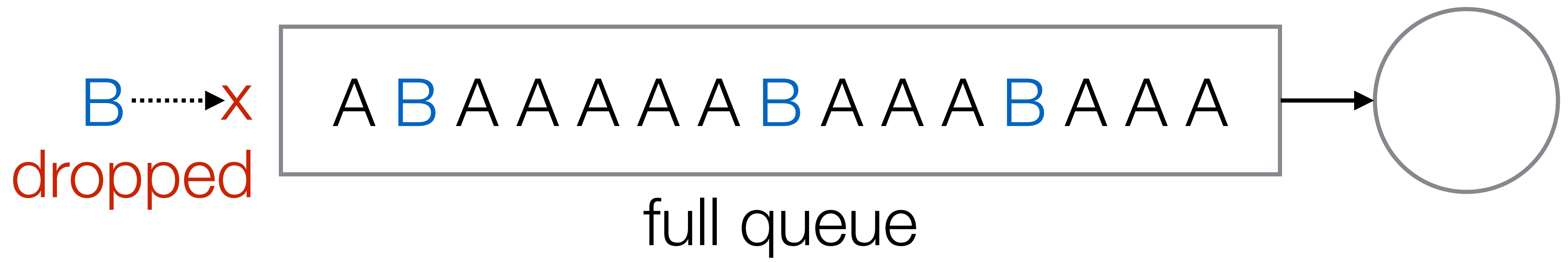
CS 925
Lecture 8
Queuing in Networks

Thursday, February 15, 2024

Prioritization of Flows

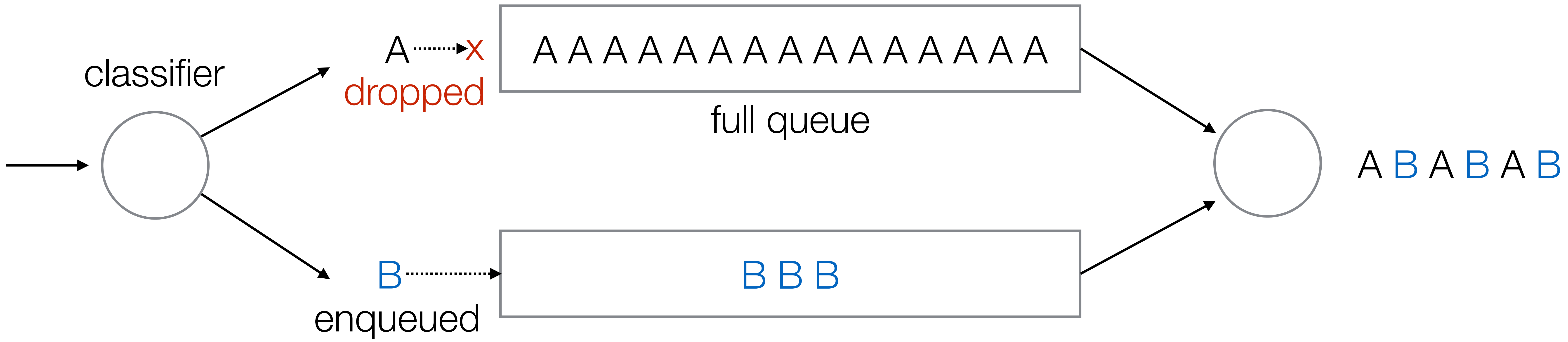
- Goals:
  - simple to implement
  - independent of packet length
  - prioritization
- Let's start with fair scheduling of equal-priority flows:
  - single queue (shared fate)



  - idea: a queue per flow

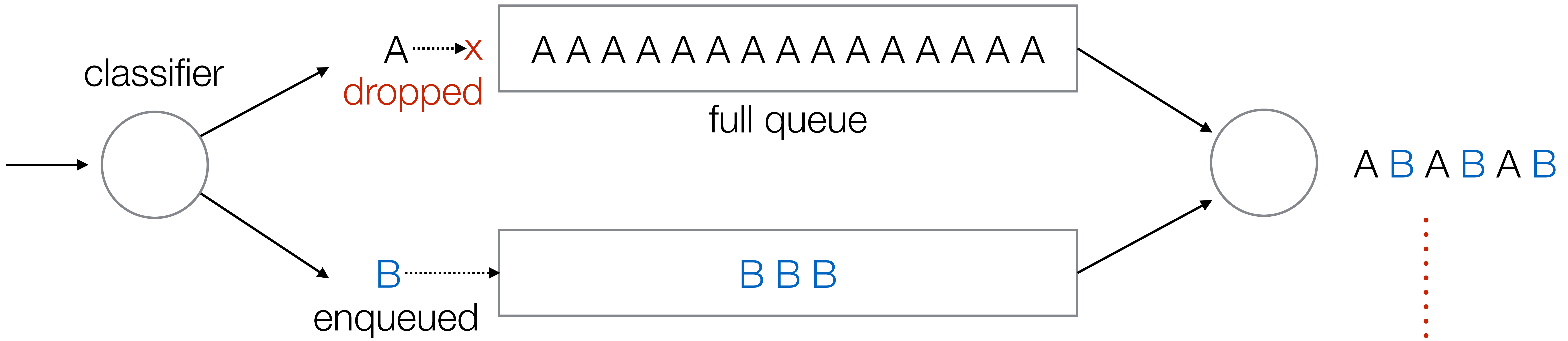
Prioritization of Flows

- Equal-priority flows:
  - multiple per-flow queues served in round-robin fashion

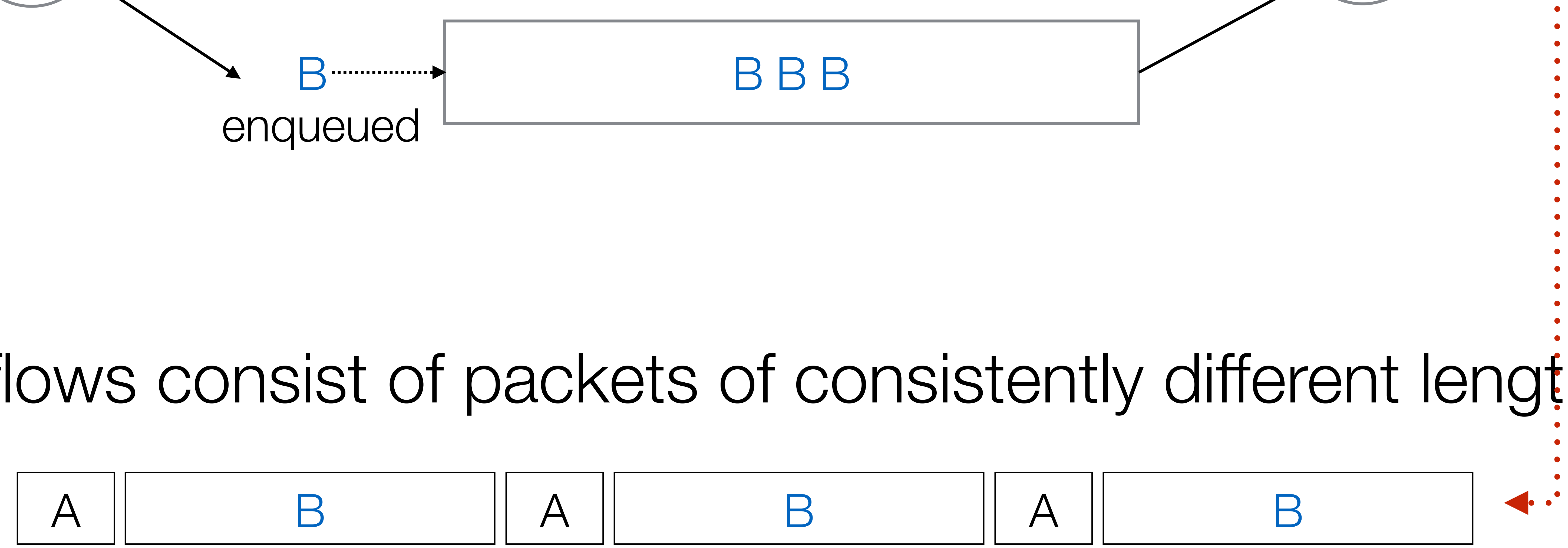


Prioritization of Flows

- Equal-priority flows:
  - multiple per-flow queues served in round-robin fashion



  - what if flows consist of packets of consistently different lengths?



Prioritization of Flows

- Need a method to account for the amount of traffic sent and serve queues in an order that reflects the amount of traffic sent
  - this is easier said than done
- The solution is based on *theoretical* approach called Processor Sharing (PS):
  - assuming packets can be fragmented into small equal-size fragments
  - if we serve packet fragments in round-robin fashion, there is no packet size bias:
  - except that we cannot break packets into small fragments (!)

A	B1	A	B2	A	B3

Fair Queuing (FQ)

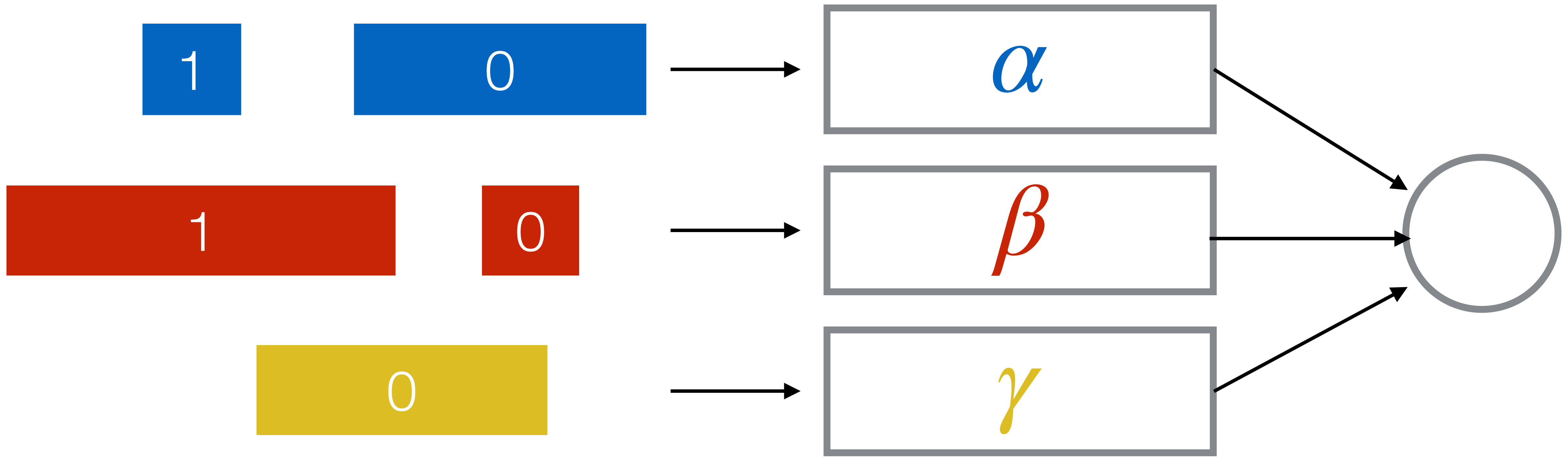
- Simulate Processor Sharing to find the order of finish times of packet transmissions
- Schedule (full) packets in that order
  - surprisingly, this can be done by a simple algorithm

Processor Sharing (PS)

- Definitions
  - $N(t)$ - number of non empty queues at time t
  - $R(t)$ - number of rounds at time t
  - P_i^α - the length of packet i in queue α
  - τ_i^α - arrival time of packet i in queue α
  - S_i^α - round when the transmission of packet i in queue α started
  - F_i^α - round when the transmission of packet i in queue α was over (the last bit was sent in the previous one)
- Then the packet schedule can be computed using:
  - $F_i^\alpha = S_i^\alpha + P_i^\alpha$ and $S_i^\alpha = \max\left[F_{i-1}^\alpha, R(\tau_i^\alpha)\right]$

Example

Queue	α		β		γ	
Packet	Arrival	Length	Arrival	Length	Arrival	Length
0	0	3	1	1	3	3
1	2	1	2	4	-	-



Fair Queuing (FQ)

- For every packet calculate
  - $F_i^\alpha = S_i^\alpha + P_i^\alpha$ and $S_i^\alpha = \max\left[F_{i-1}^\alpha,\ R(\tau_i^\alpha)\right]$
- Schedule them in the increasing order of their F_i^α
  - per-queue packet order is preserved ($F_i^\alpha < F_{i+1}^\alpha$)
  - smaller F_i^α when compared to other queues indicates “lower utilization” by that queue

FQ

$$F_i^\alpha = S_i^\alpha + P_i^\alpha \qquad S_i^\alpha = \max[F_{i-1}^\alpha, R(\tau_i^\alpha)]$$

	Queue Alpha		Queue Beta		Queue Gamma
	Packet 0	Packet 1	Packet 0	Packet 1	Packet 0
Arrival (tau)	0	2	1	2	3
Length (P)	3	1	1	4	3
Start round (S)	0	3	1	2	2
End round (F)	3	4	2	6	5

Time τ	Round $R(\tau)$	Queue Alpha Packet 0	Queue Alpha Packet 1	Queue Beta Packet 0	Queue Beta Packet 1	Queue Gamma Packet 0
0	0	A 1				
1	1	2		A		
2			A	1	A	
3		3		4		A
4	2				1	
5			5			1
6			1			
7	3				2	
8						2 8
9	4				3	
10						3
11	5				4	
12	6				12	

Total wait 32

FIFO

$$F_i^{\alpha} = S_i^{\alpha} + P_i^{\alpha} \qquad S_i^{\alpha} = \max\left[F_{i-1}^{\alpha}, R(\tau_i^{\alpha})\right]$$

	Queue Alpha		Queue Beta		Queue Gamma
	Packet 0	Packet 1	Packet 0	Packet 1	Packet 0
Arrival (tau)	0	2	1	2	3
Length (P)	3	1	1	4	3
Start round (S)					
End round (F)					

		Queue Alpha		Queue Beta		Queue Gamma
Time	Round	Packet 0	Packet 1	Packet 0	Packet 1	Packet 0
0		(A)				
1				(A)		
2			(A)		(A)	
3		3				(A)
4				4		
5						
6						
7						
8					8	
9			9			
10						
11						
12						12

total WAIT
36

Weighed Fair Queueing

- Fair Queuing does not support prioritization
- Idea:
  - Generalized Processor Sharing (GPS): adjust fragment sizes of PS to reflect priority of the flow
  - Weighted Fair Queuing (WFQ): use simulation of GPS to schedule packets

Deficit Round Robin

- An improvement on WFQ
- Each queue has a deficit counter
- Queues with deficit counter values higher than the packet length are served in round robin fashion (and the deficit counter is reduced accordingly)
- A *quantum* is added to deficit counter of a queue that is skipped
- Complexity: $O(1)$ vs $O(\log N)$ for WFQ

Active Queue Management

- A method used by routers or switches of preemptive dropping (or marking) of packets before queues become full with the intent of:
  - avoiding congestion
  - reducing end-to-end latency

Random Early Detection

- TCP flow control
  - packet loss triggers back-off (rate reduction)
  - it takes time to recognize that packet was lost (network latency, timeouts)
- Possible outcome network synchronization
  - periods of congestion followed by periods of low load caused by a TCP flows backing off
- Solution: 1993 Sally Floyd
  - RED (Random Early Detection)

RED - Goals

- Goals:
  - Avoid congestion and global synchronization
  - Avoid bias against bursty traffic
  - Bound on queuing delay
- Method
  - calculate *average queue size*
  - set two thresholds (TH_{max} and TH_{min}) within the queue size
  - enqueue or drop packets based the relation between the average queue size and the thresholds

RED - Details

- Average queue size
  - use exponentially weighted average
  - RED uses low weights (0.002)
- Determining packets to discard:
  - discards should be regular (so burst are not targeted)
  - … but not too regular (because strict regularity is also undesirable