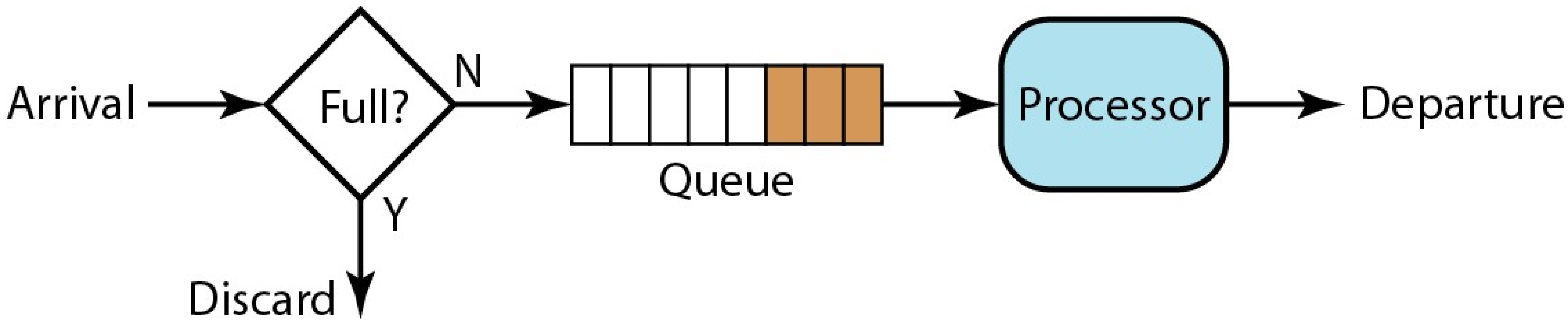
Scheduling and Traffic Shaping

Dept. of Computer Science and Engineering
Sogang University

Scheduling

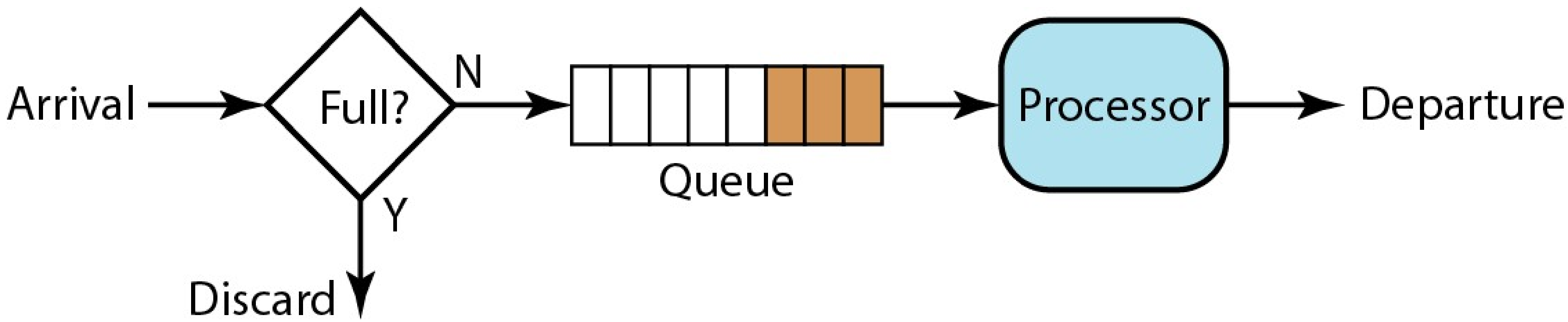
- deciding the order of packet processing

- FIFO (First-In First-Out) queue: basic type
  - packet arrived first is processed (and sent out) first



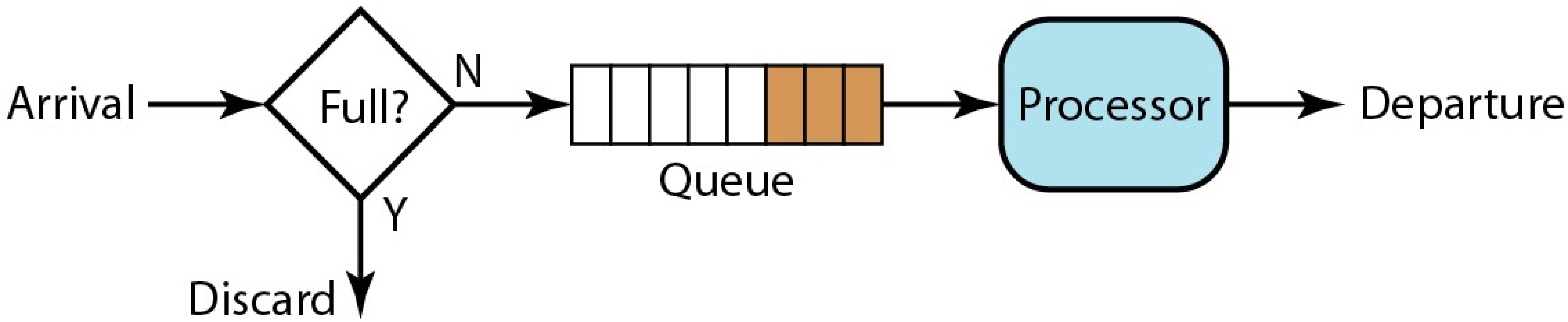
Scheduling

- FIFO queue
  - flows are not distinguished
  - no congestion control
  - if queue is full, the last packet is dropped (tail drop)
  - fair in terms of delay
  - bandwidth is not considered



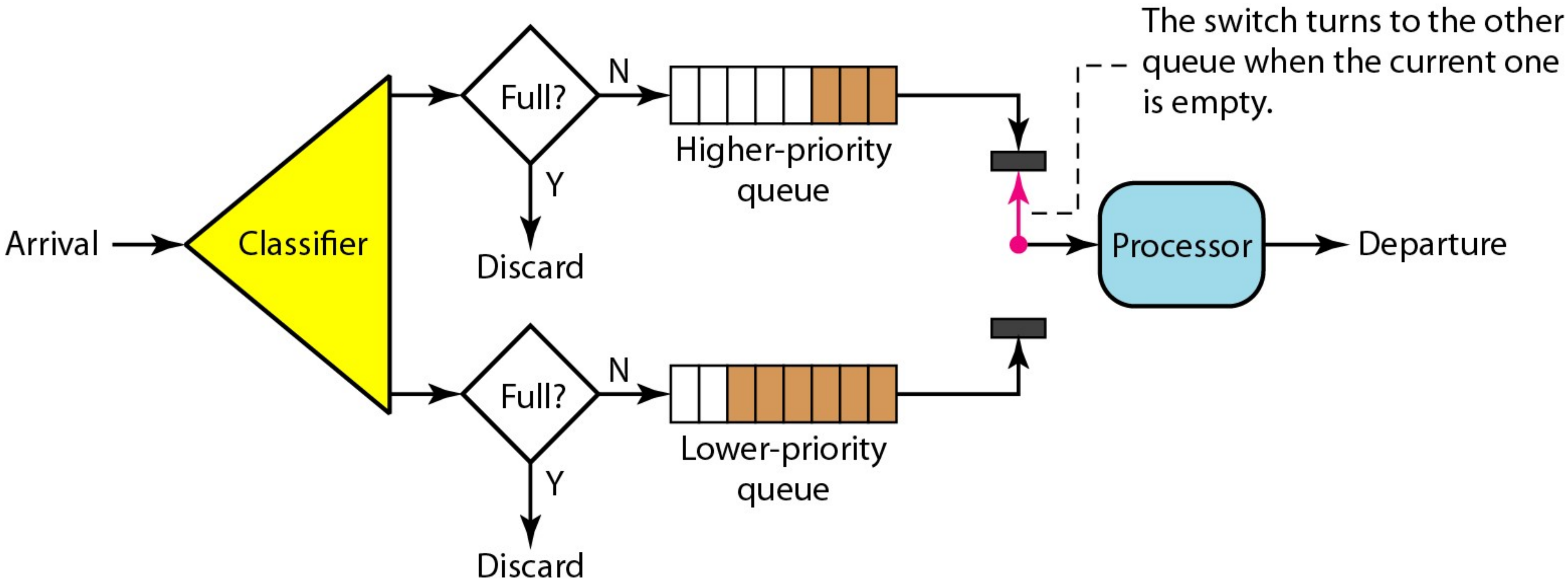
Scheduling

- FIFO queue: problem
  - If a flow sends more packets, the flow is served more
    - promotes selfish behavior 패킷을 많이 보내는 쪽이 서비스를 많이 받는다
    - can be used as attack 라우터를 다운 시키기 위해 의도적으로 패킷을 계속 보내서 점령
  - Packets may need different types of QoS, but cannot be implemented in FIFO queue



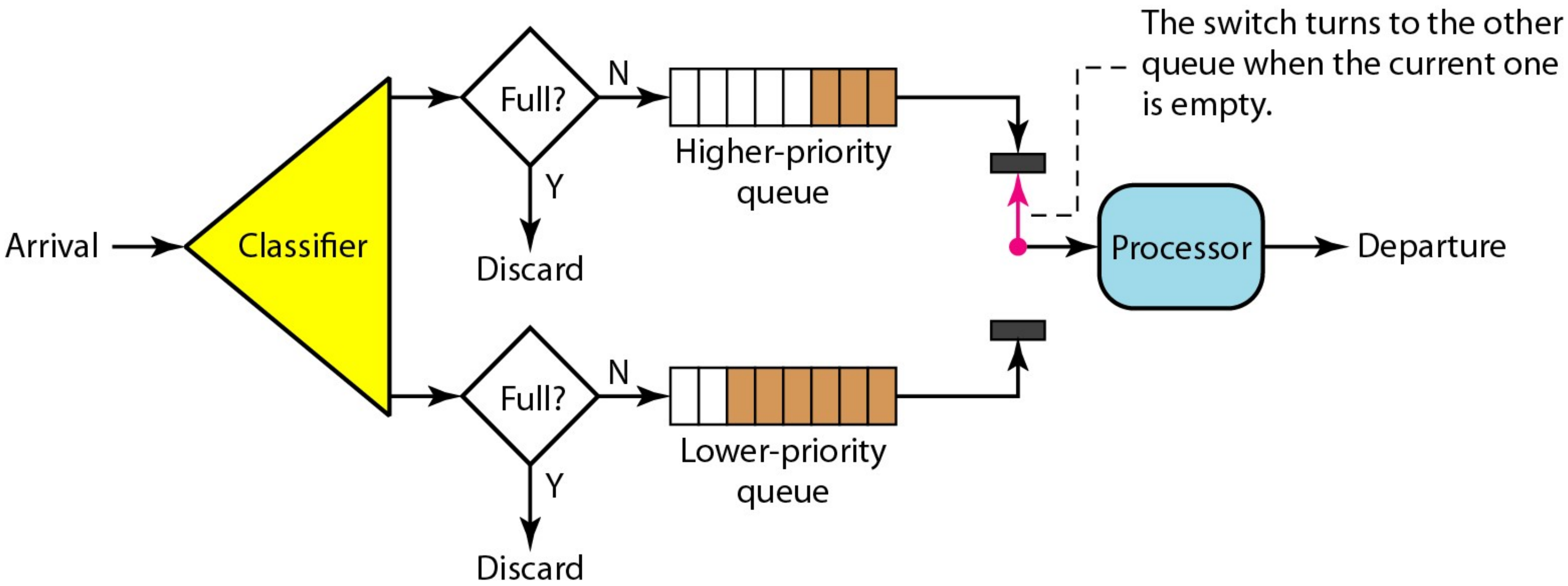
Scheduling

- Priority Queue
  게임, 스트리밍 같은 즉각적인 서비스가 필요한 걸 우선순위로 둠
  - Process packet depending on priority
    - Even if a packet arrived late, it is processed first if it has the highest priority



Scheduling

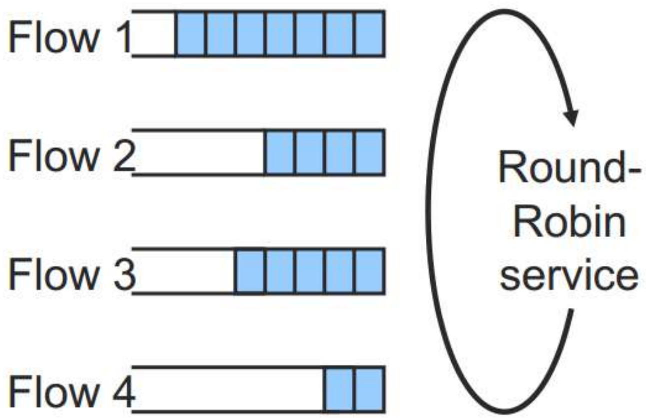
- Priority Queue: problem
  - low-priority flow can be starved
    - cannot send packets for long time (starvation)



Scheduling

round robin에서는 패킷 단위로 차례대로 서비스하니까 서비스를 많이 받고 싶으면 패킷을 크게 하면 그 덩어리를 처리

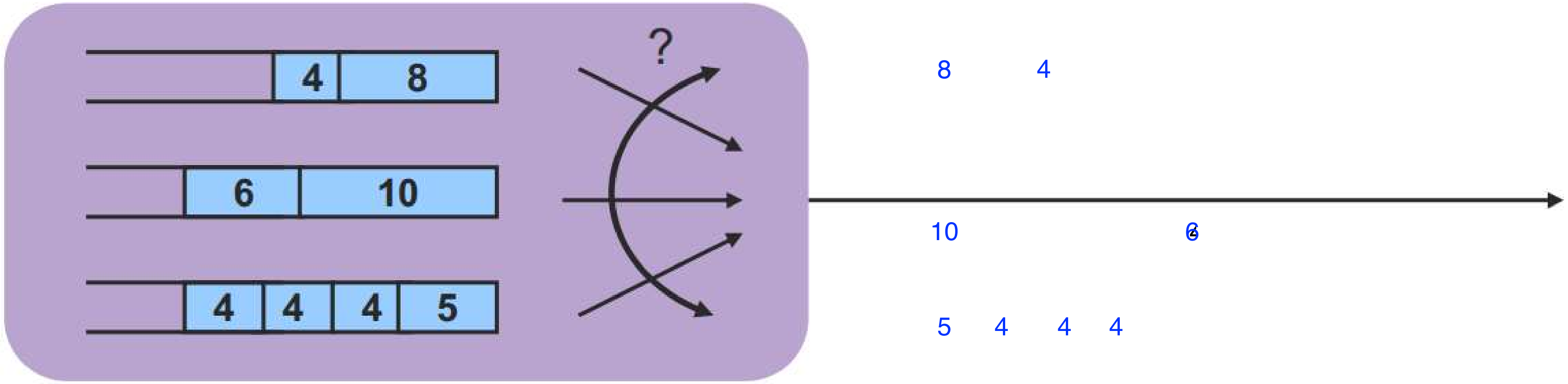
- Round Robin
  - A queue exists for each flow
  - Takes turns to go around each queue and process one packet per each queue
  - Compared with FIFO, a flow does not receive more service by sending more packets
  - Flows sending large packets gets more service



Scheduling

- Fair queuing
  - A queue exists for each flow
  - Schedule packets so that the flows are given fair service
    - Scheduling has to be fair even if packet sizes are different

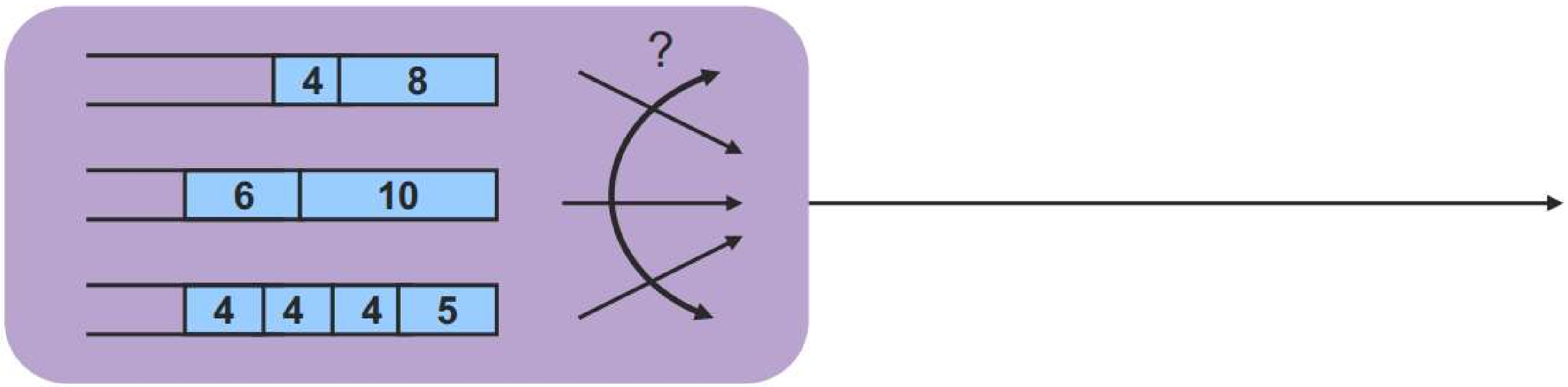
첫 시작은 tie breaking 조건에 따름



Scheduling

- Fair queuing with variable packet length: algorithm
  - S_i = amount of data flow i has sent
  - process flow with minimum S_i, if it has packets to process
  - Update S_i after processing packet
    - $S_i = S_i + P$ (packet length)

Si 값이 낮은 거 먼저함



Scheduling

- Fair queuing: problem
  - If a flow does not have data for long time and come back
  - The flow will have a very small S_i → will receive service alone for a long time

- Solution
  - "use it or lose it"
  - S_{min} = min(S_i such that queue i is not empty)
  - If queue j is empty, set $S_j = S_{min}$

어떤 최소한의 값으로 설정해둠

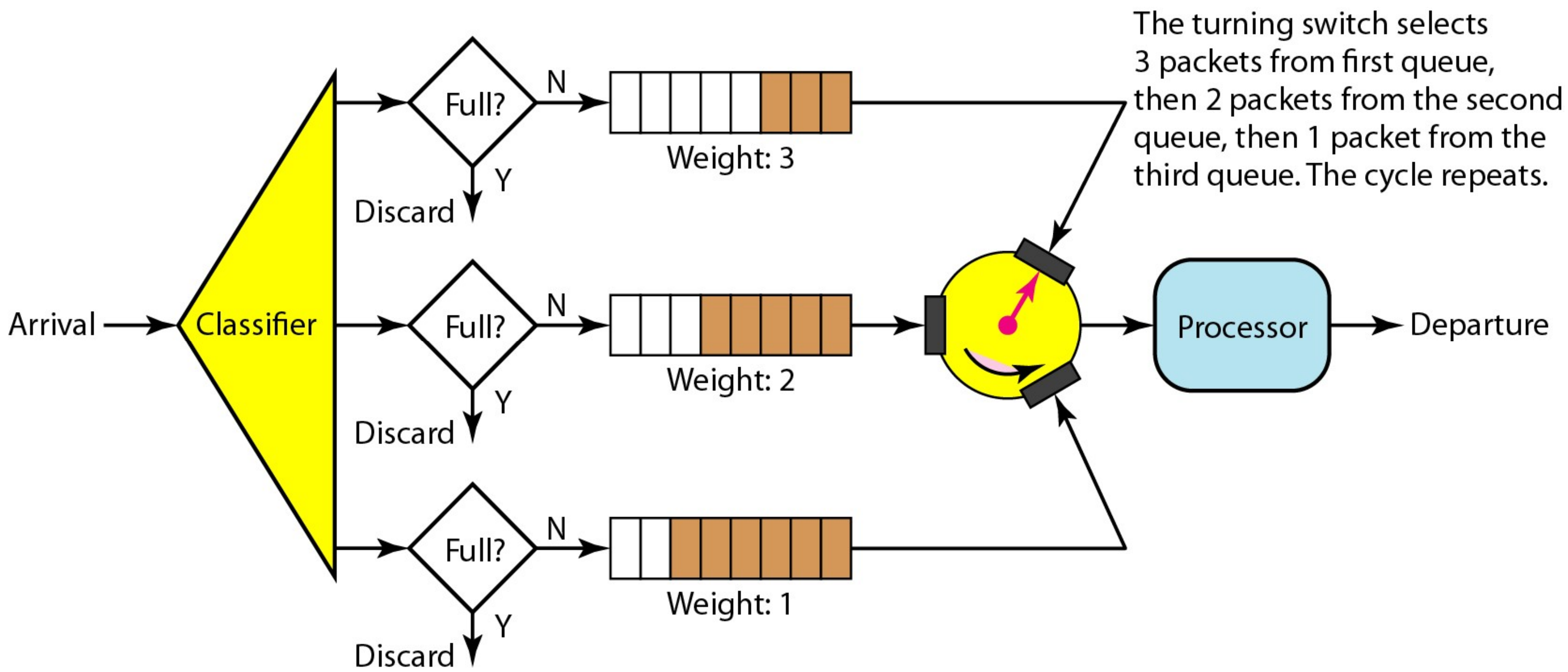
Scheduling

- Weighted Fair Queuing
  - Similar to fair queuing, but flows have different priorities
  - Each flow has a weight, w_i: weight of flow i
  - S_i is computed in the following way
    - $S_i = S_i + P / w_i$ 우선순위 가중치

wi가 높을 수록 Si가 작아지니까 우선순위 높아짐

Scheduling

- Weighted Fair Queuing



Scheduling

- Weighted fair queuing: pros
  - fair scheduling based on flow priorities
  - avoid starvation

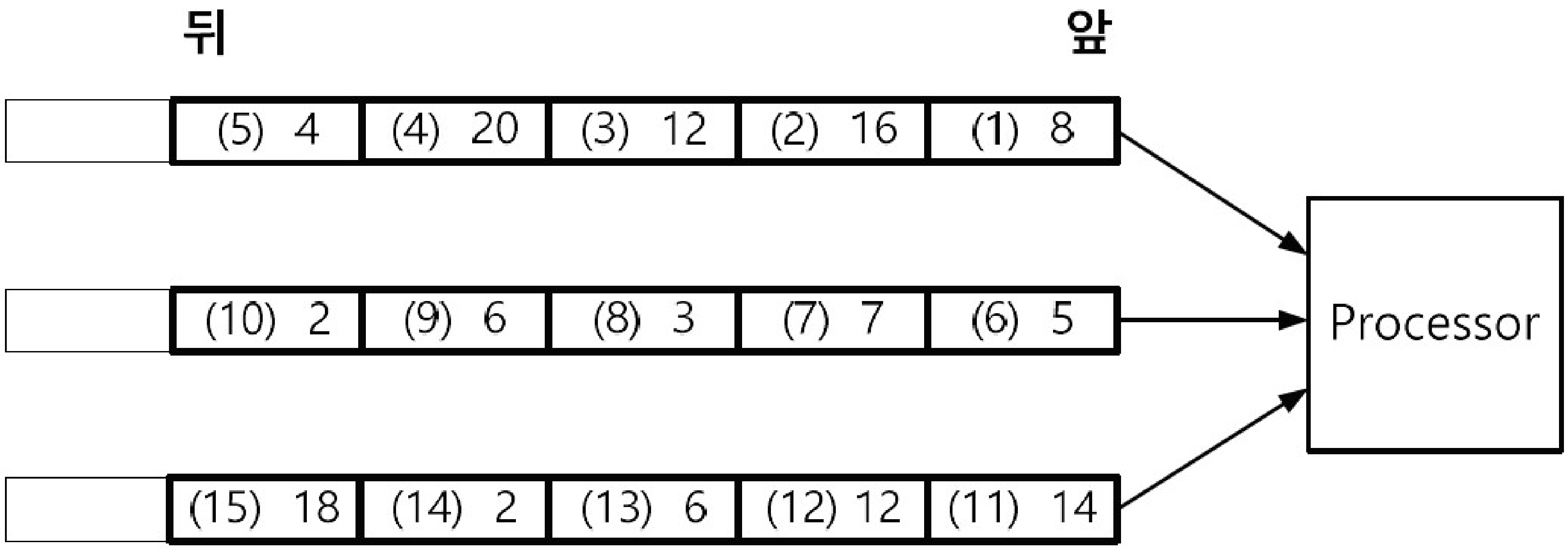
- Weighted fair queuing: cons
  - complex state
    - a queue for each flow
  - complex computation
    - flow separation
    - packet sorting
    - processing when a flow is added or removed

Scheduling

맨 처음에는 s 값이 같으니까 어떠한 tie breaking rule을 따름

- S값이 같은 경우의 우선순위: A > B > C
  - 모든 flow의 weight가 같은 경우, 패킷 처리 순서는?
  - $W_A = 4, W_B = 1, W_C = 2$일때 패킷 처리 순서는?

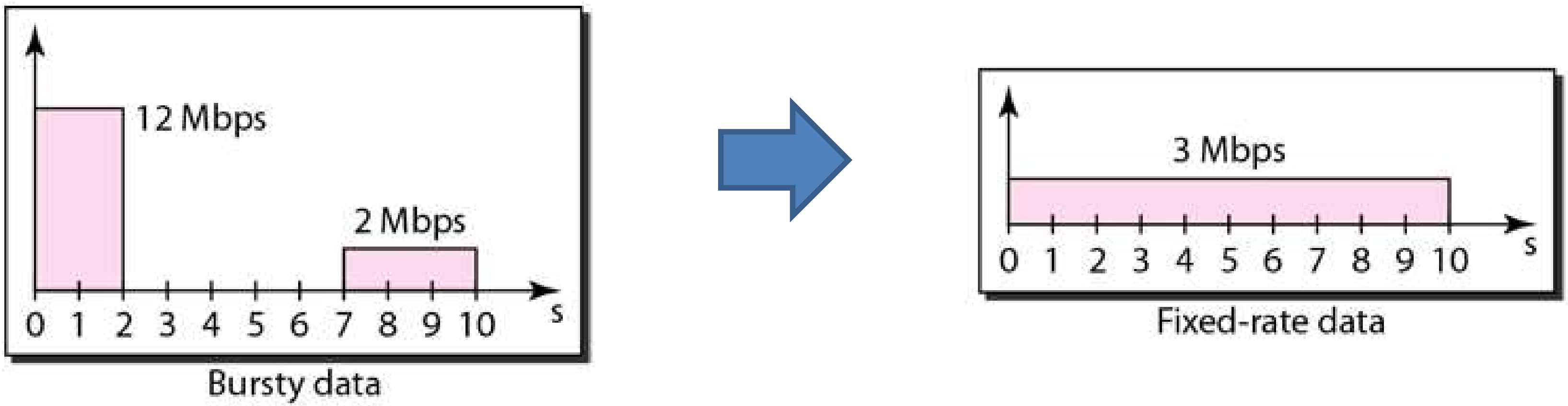
1, 6, 11, 2, 7, 3, 12, 4, 8, 13, 5, 9, 14, 10, 15



Traffic Shaping

- A method to control amount of data the source sends to the network

bursty traffic - 불규칙하게 어느 타이밍에 쏠리게 들어오는 트래픽

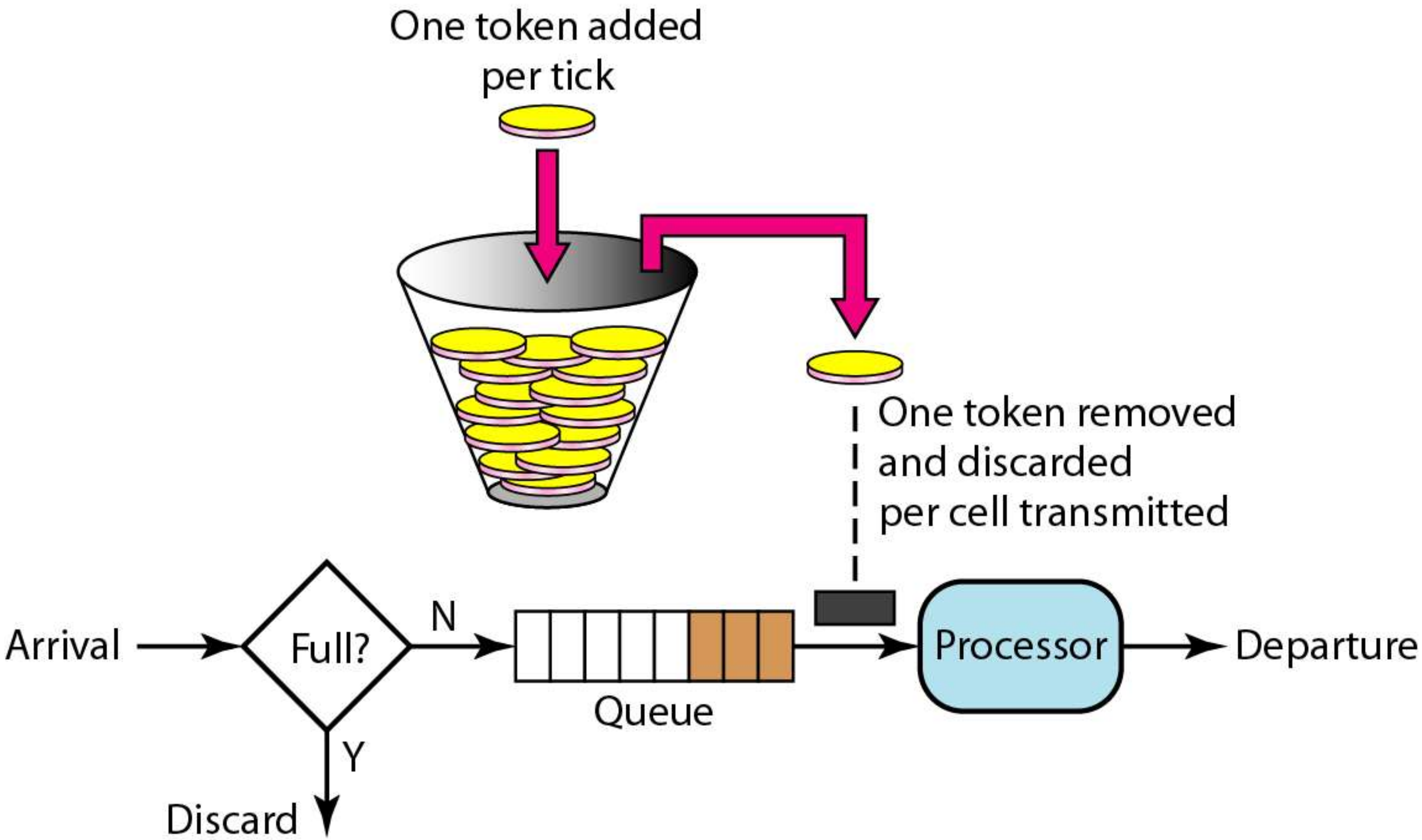


Traffic Shaping

- Token bucket filter
  - Token is accumulated in a bucket
    - If the bucket is full, token is no longer accumulated
  - When the sender sends data traffic, it also consumes tokens from the bucket
  - The sender does not send traffic is there is no token in the bucket

Traffic Shaping

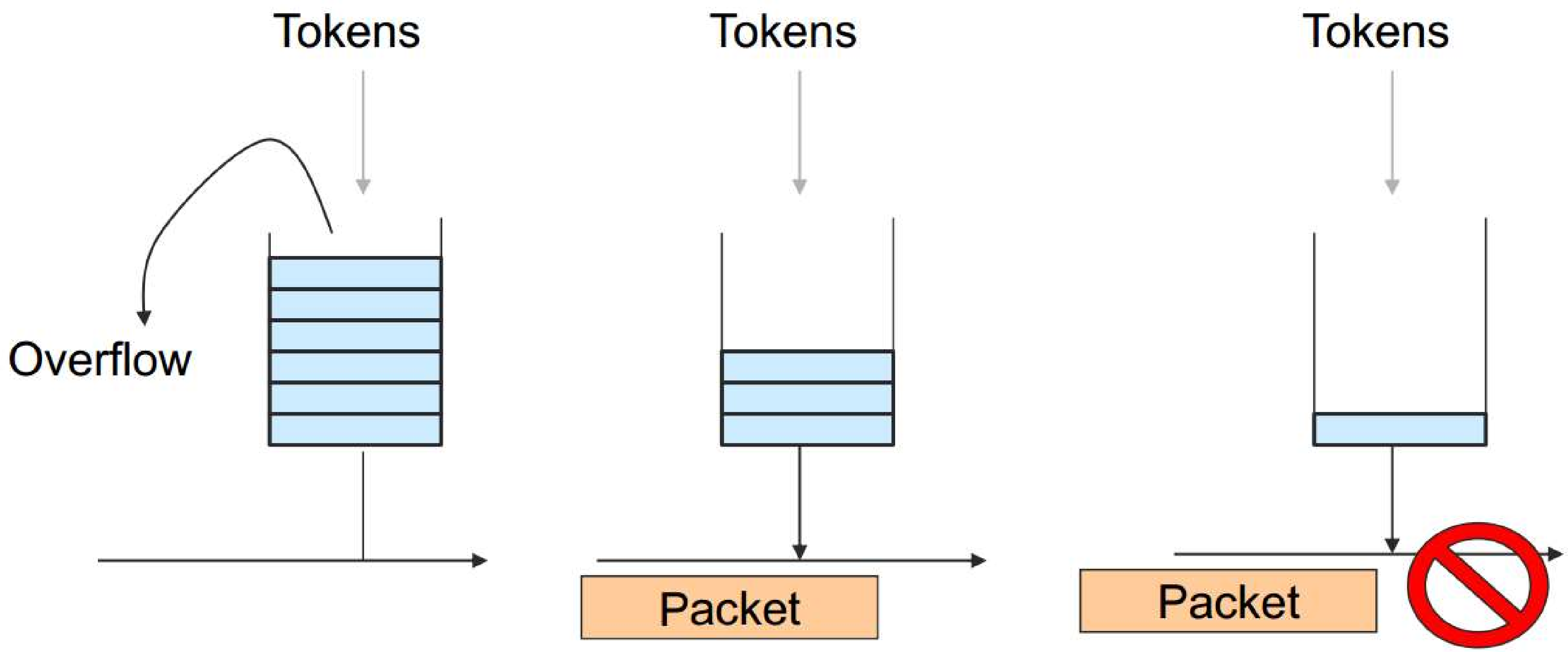
- Token bucket



Token Bucket

- Traffic characterization
  - Token rate: r
  - Bucket depth: B
- Use
  - 1 token is needed to send 1 byte
    - for n bytes, n tokens needed
  - Initially, no token is in the bucket
  - r tokens are accumulated in one seconds
  - number of tokens cannot exceed B

Token Bucket



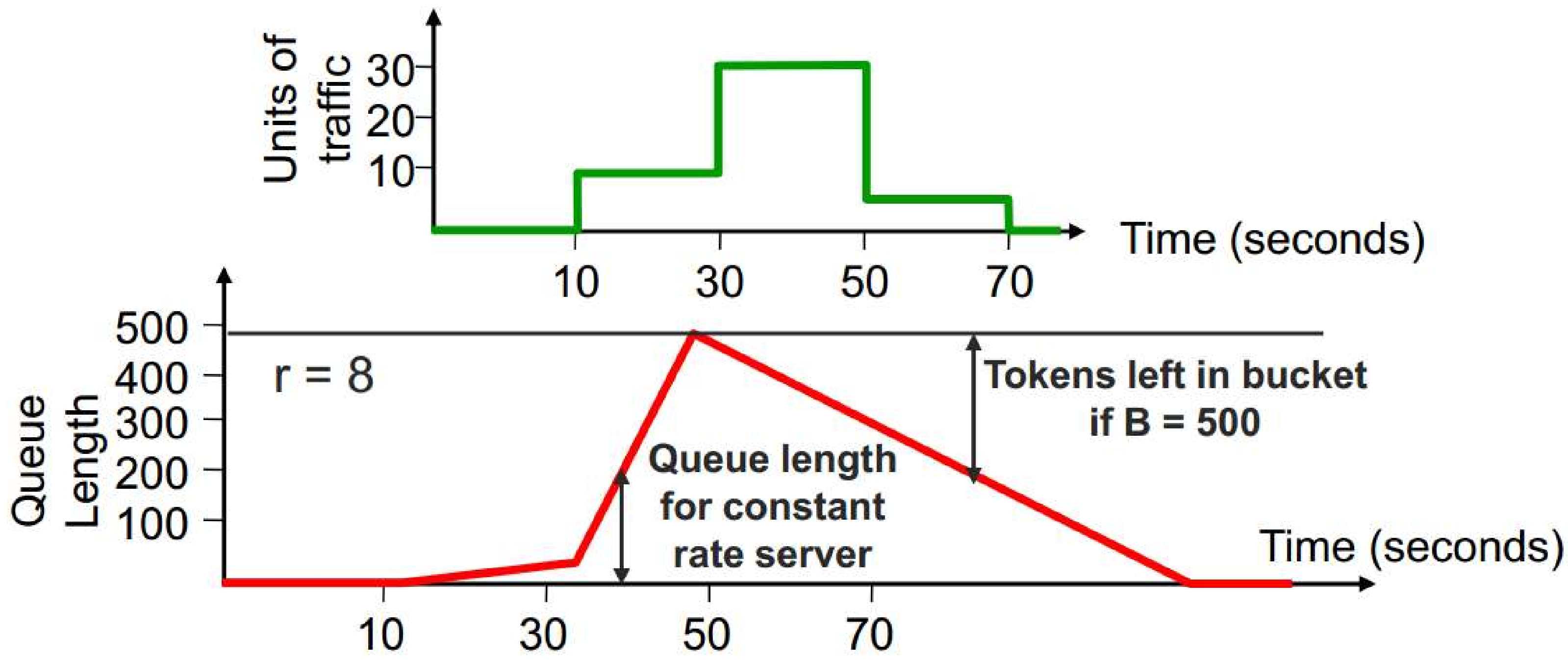
Buffer tokens up to capacity of bucket

Enough tokens → packet goes through, tokens removed

Not enough tokens → wait for tokens to accumulate

Token Bucket

- Look at the trace of input traffic, and find the minimum bucket size so that the traffic is not delayed



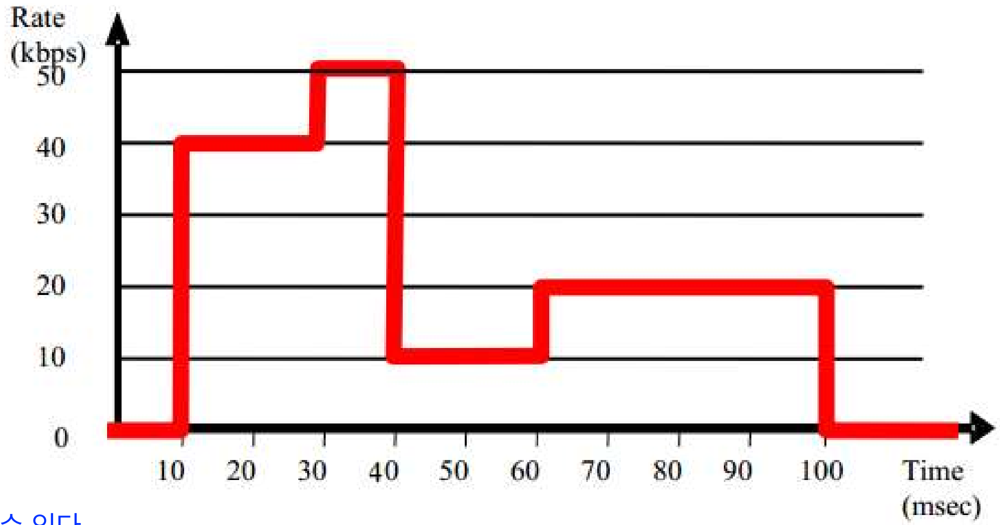
Token Bucket: exercise

- We want to find the minimum bucket size so that traffic is not delayed

10-30ms 500소비
30-40 350소비
40-60 100버킷
60-100 200소비
B = 950(500+350-100+200)

= 15bit ms

- r = 15kbps 토큰이 차는 속도
- input traffic →



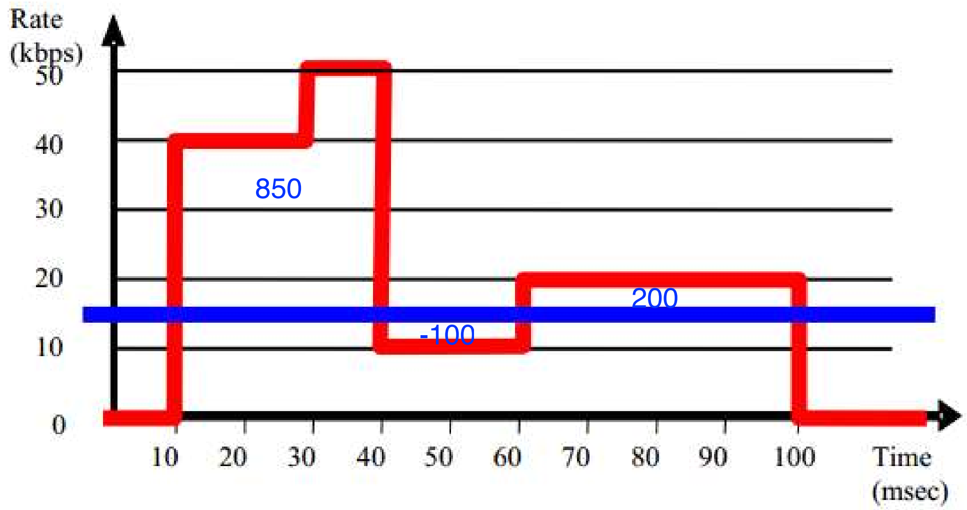
최대 버킷 사이즈만큼 토큰이 쌓있다

프래픽이 몰리면 버킷이 차지 않으면서 15kbps로만 보낼 수 있다

- What is the minimum bucket size (B)?

Token Bucket: exercise

- r = 15kbps



이 선 위로는 버킷을 소비
밑으로는 버킷에 저장

0 time에서 950bit을 가지면 100 time이 될 때 0으로 다 소비되어 딱 맞음

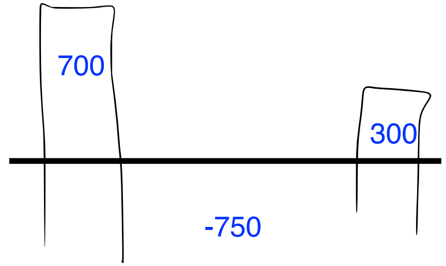
- min B =

  (40 – 15) * 20 + (50 – 15) * 10 – (15 – 10) * 20 + (20 – 15) * 40 = 950 bits

- Start from the left, and find the maximum data that is can be accumulated in the buffer

Token Bucket: exercise

B = 700



- what is the minimum B when
  - r = 55kbps?
  - r = 45kbps?
  - r = 35kbps?
  - r = 25kbps? 300 + 250
  - r = 5kbps?

300 - 400 + 750이 아니라
300 - 400 = 0 + 750 = 750이다
빼서 음수가 되지 않고 최소 0
계산할 때 시간순으로 더하면서 음수가 되면 0으로 생각하고 계속

B = 750