IEEE INFOCOM

Towards the Fairness of Traffic Policer

Danfeng Shan[1], Peng Zhang[1], Wanchun Jiang[2], Hao Li[1], Fengyuan Ren[3]

[1]Xi'an Jiaotong University, [2]Central South University, [3]Tsinghua University

西安交通大學 XI'AN JIAOTONG UNIVERSITY

中南大學 CENTRAL SOUTH UNIVERSITY

清華大學 Tsinghua University

Background

Background

YouTube

NETFLIX

Google

Content Providers

Background

YouTube

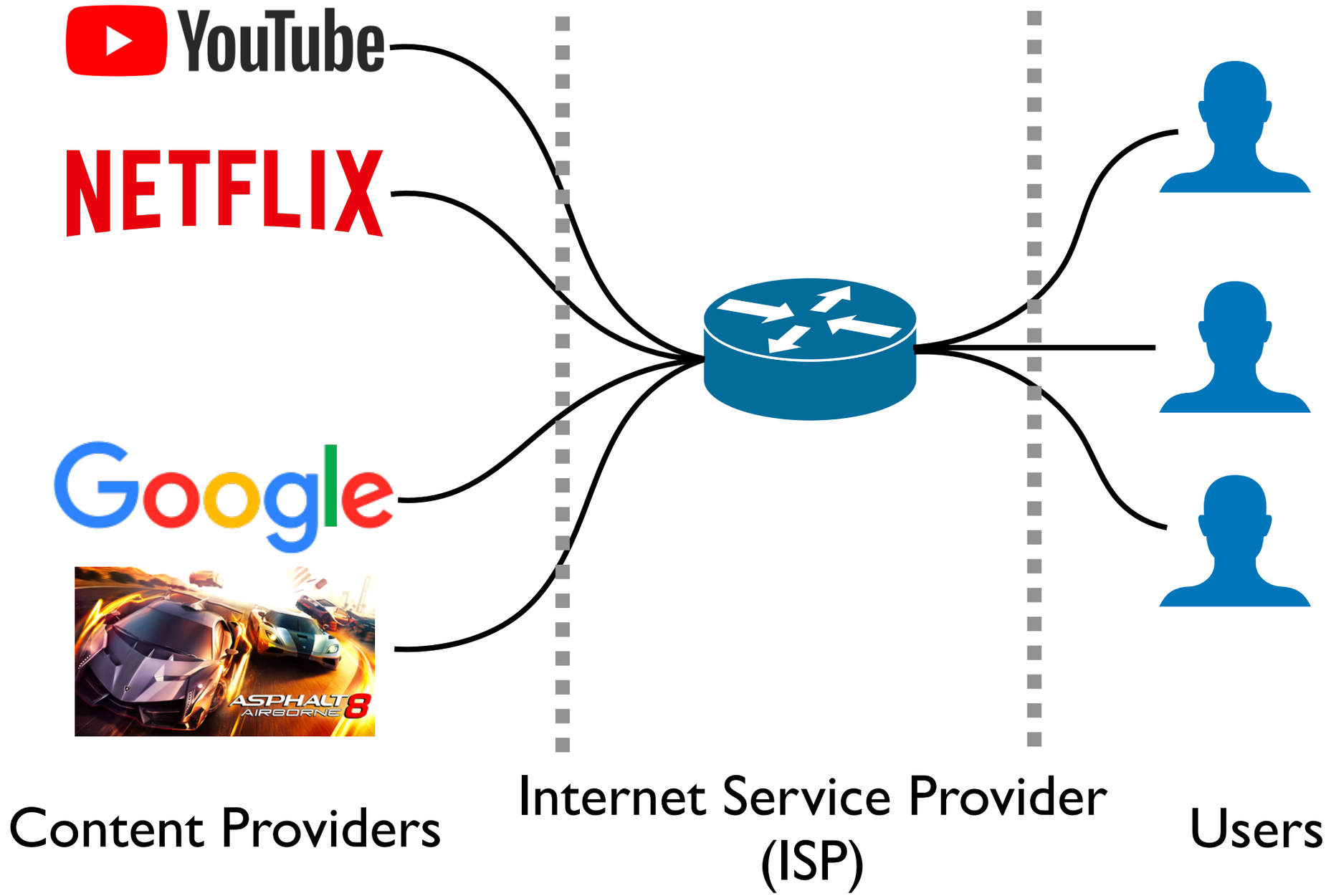
NETFLIX

Google

Content Providers

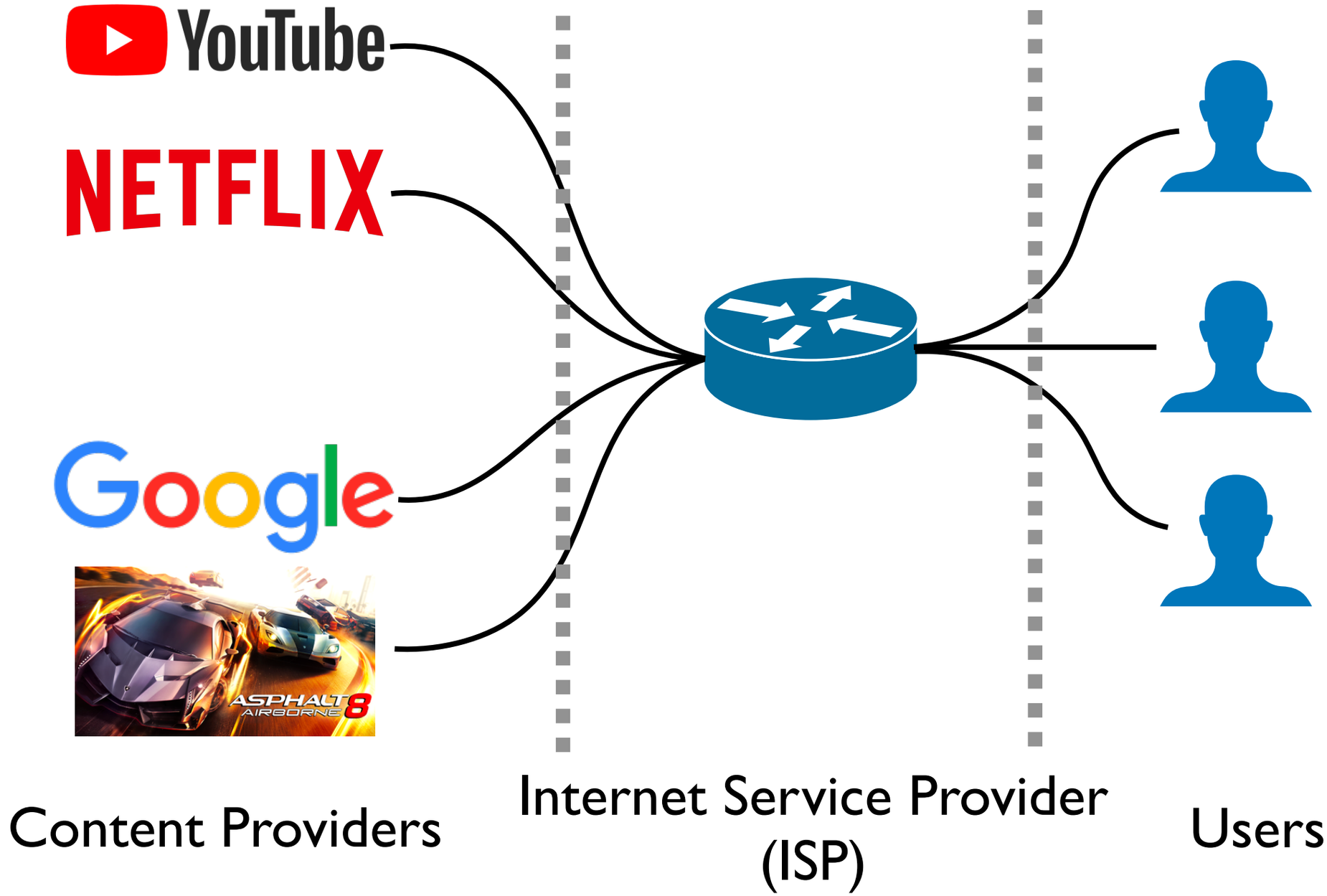
Users

Background


YouTube
NETFLIX
Google
ASPHALT 8 AIRBORNE
Content Providers
Internet Service Provider (ISP)
Users

Background

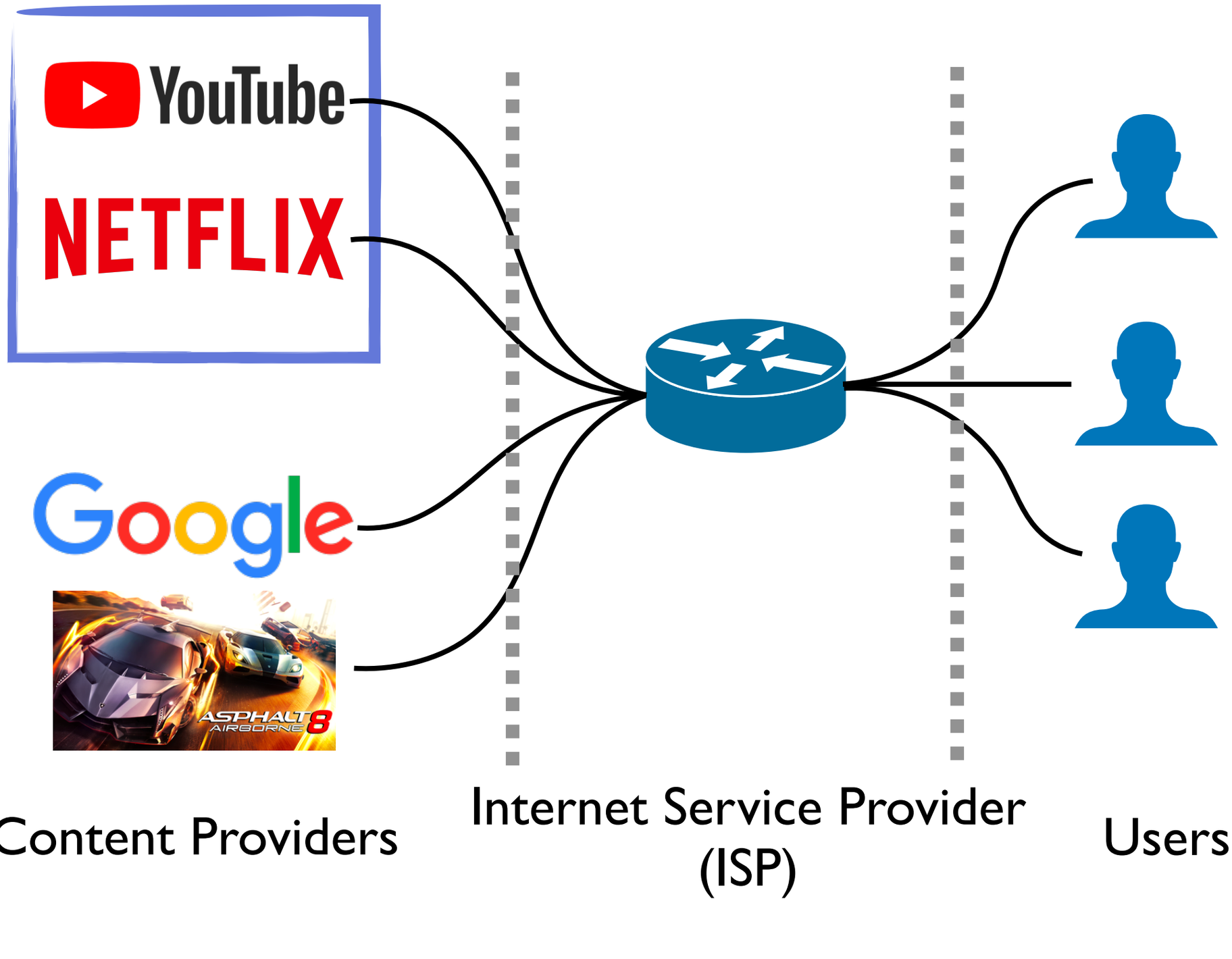
Application Requirements



Background

Application Requirements

Throughput Intensive



Content Providers

Internet Service Provider (ISP)

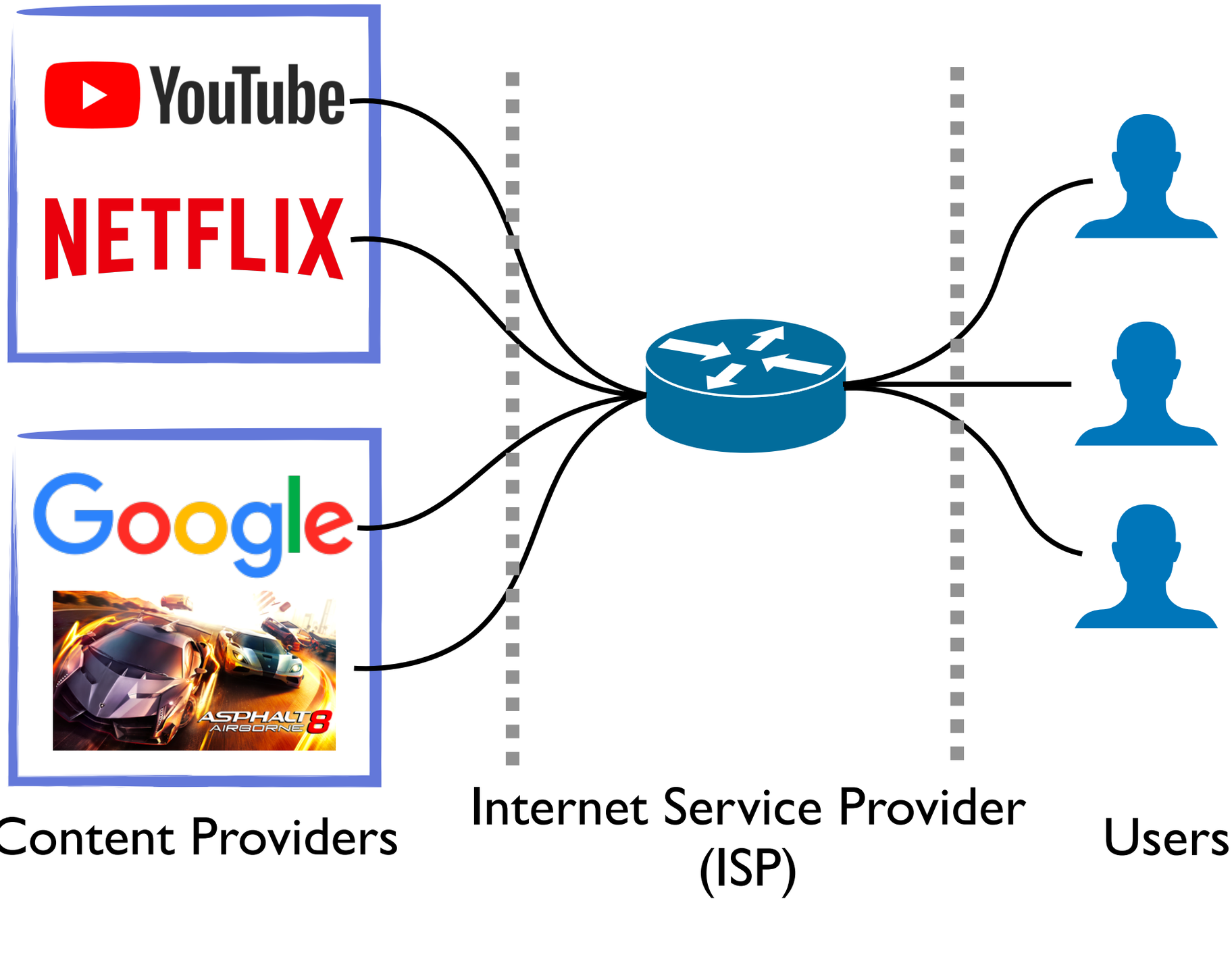
Users

Background

Application Requirements

Throughput Intensive

Latency Sensitive

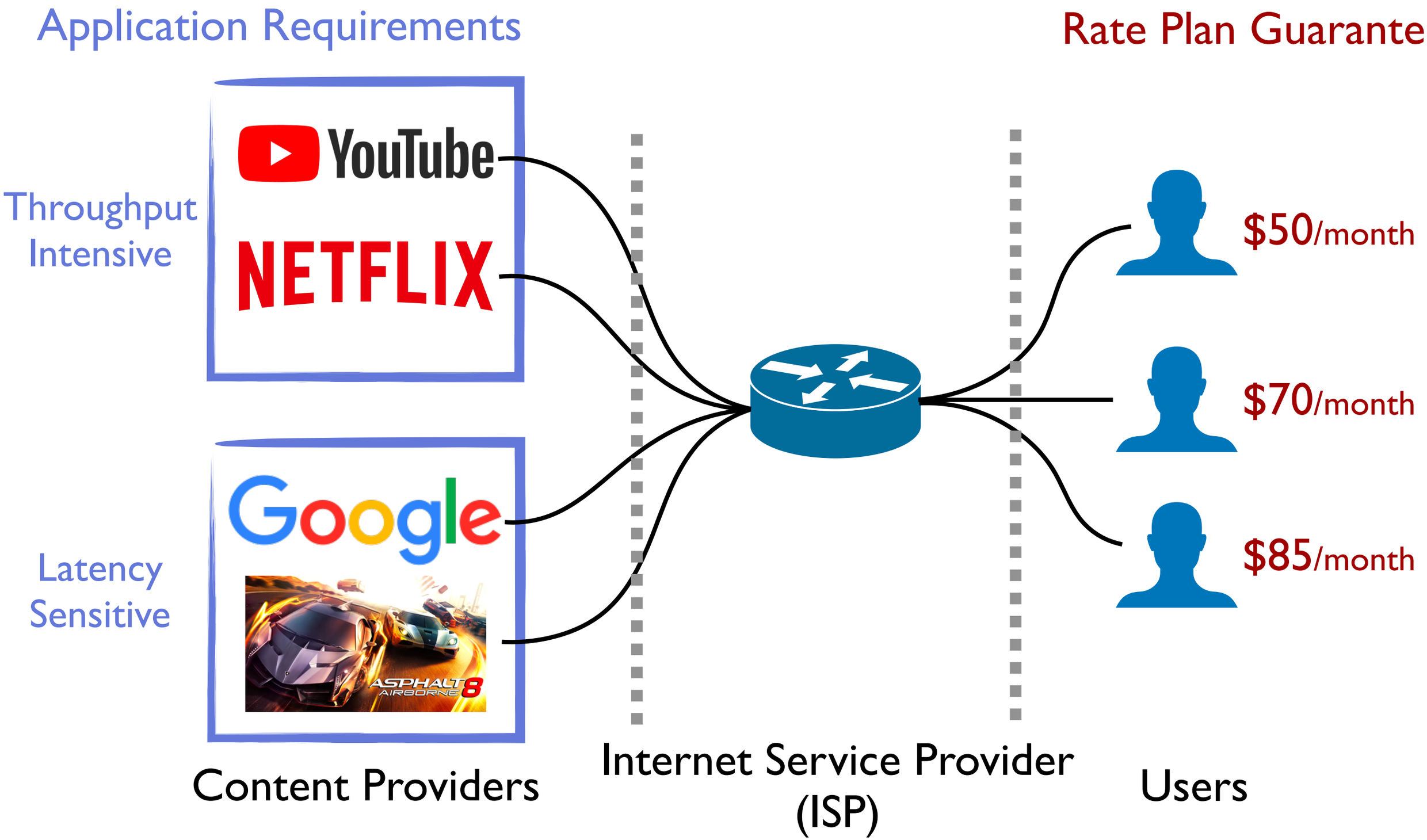


Content Providers

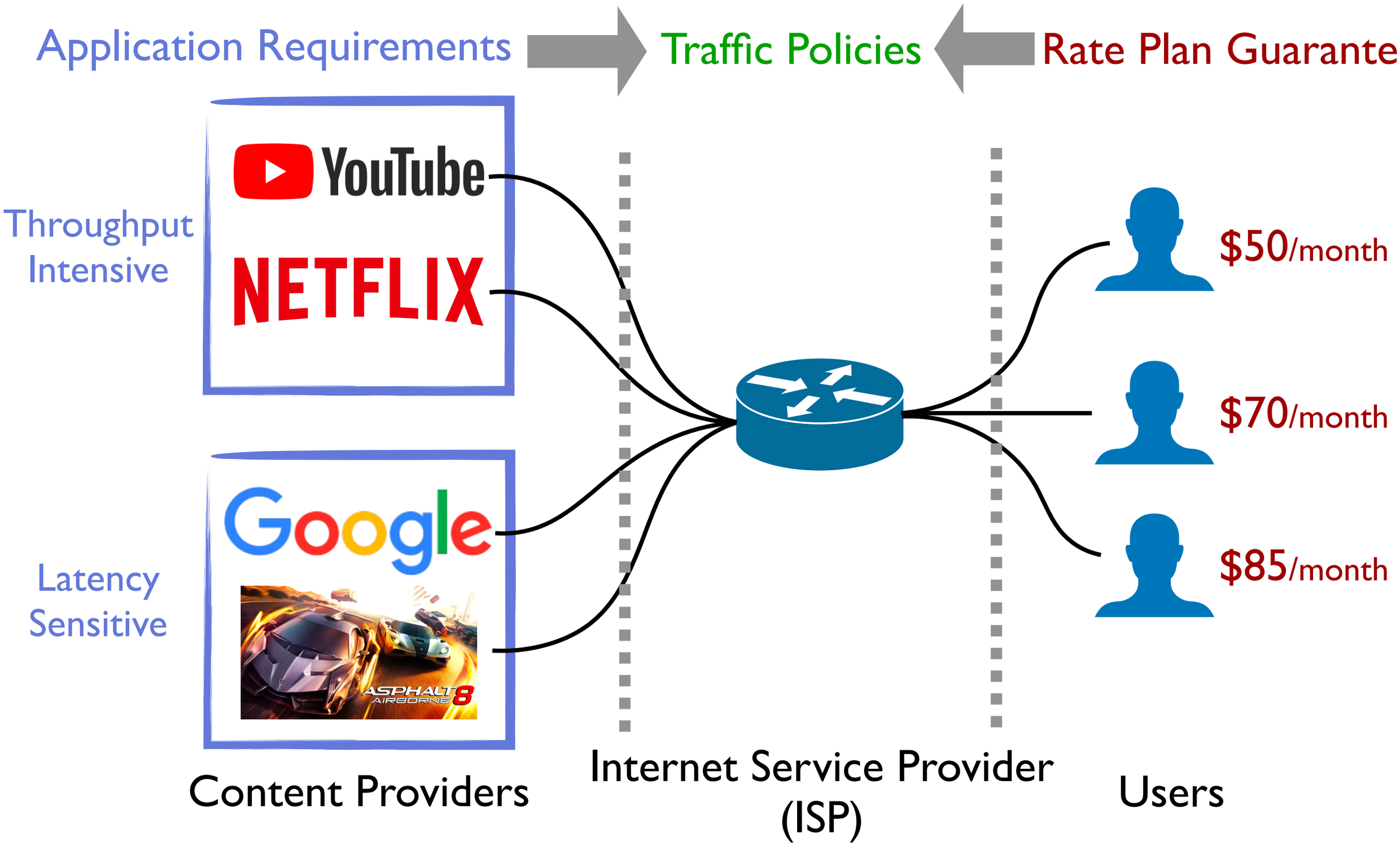
Internet Service Provider (ISP)

Users

Background


Application Requirements
Rate Plan Guarantee
Throughput Intensive
YouTube
NETFLIX
Latency Sensitive
Google
ASPHALT 8 AIRBORNE
$50/month
$70/month
$85/month
Content Providers
Internet Service Provider (ISP)
Users

Background


Application Requirements
Traffic Policies
Rate Plan Guarantee
Throughput Intensive
YouTube
NETFLIX
Latency Sensitive
Google
ASPHALT 8 AIRBORNE
$50/month
$70/month
$85/month
Content Providers
Internet Service Provider (ISP)
Users

Background

Enforce Traffic Policies
(Throttle Traffic Rate)

Background

Enforce Traffic Policies
(Throttle Traffic Rate)

Traffic Policing

Drop packets once reaching the limit

Background

Enforce Traffic Policies
(Throttle Traffic Rate)

Traffic Policing

Drop packets once reaching the limit

Traffic Shaping

Buffer packets after reaching the limit

Background

Enforce Traffic Policies
(Throttle Traffic Rate)

Traffic Policing

Drop packets once reaching the limit

- ✓ No RTT inflation
- ✓ Lower cost
- x Higher loss rate

Traffic Shaping

Buffer packets after reaching the limit

Background

Enforce Traffic Policies
(Throttle Traffic Rate)

Traffic Policing

Drop packets once reaching the limit

- ✓ No RTT inflation
- ✓ Lower cost
- x Higher loss rate

Traffic Shaping

Buffer packets after reaching the limit

- ✓ Lower loss rate
- x Require memory
- x RTT inflation

Background

Enforce Traffic Policies
(Throttle Traffic Rate)

Traffic Policing

Drop packets once reaching the limit

- ✓ No RTT inflation
- ✓ Lower cost
- x Higher loss rate

Traffic Shaping

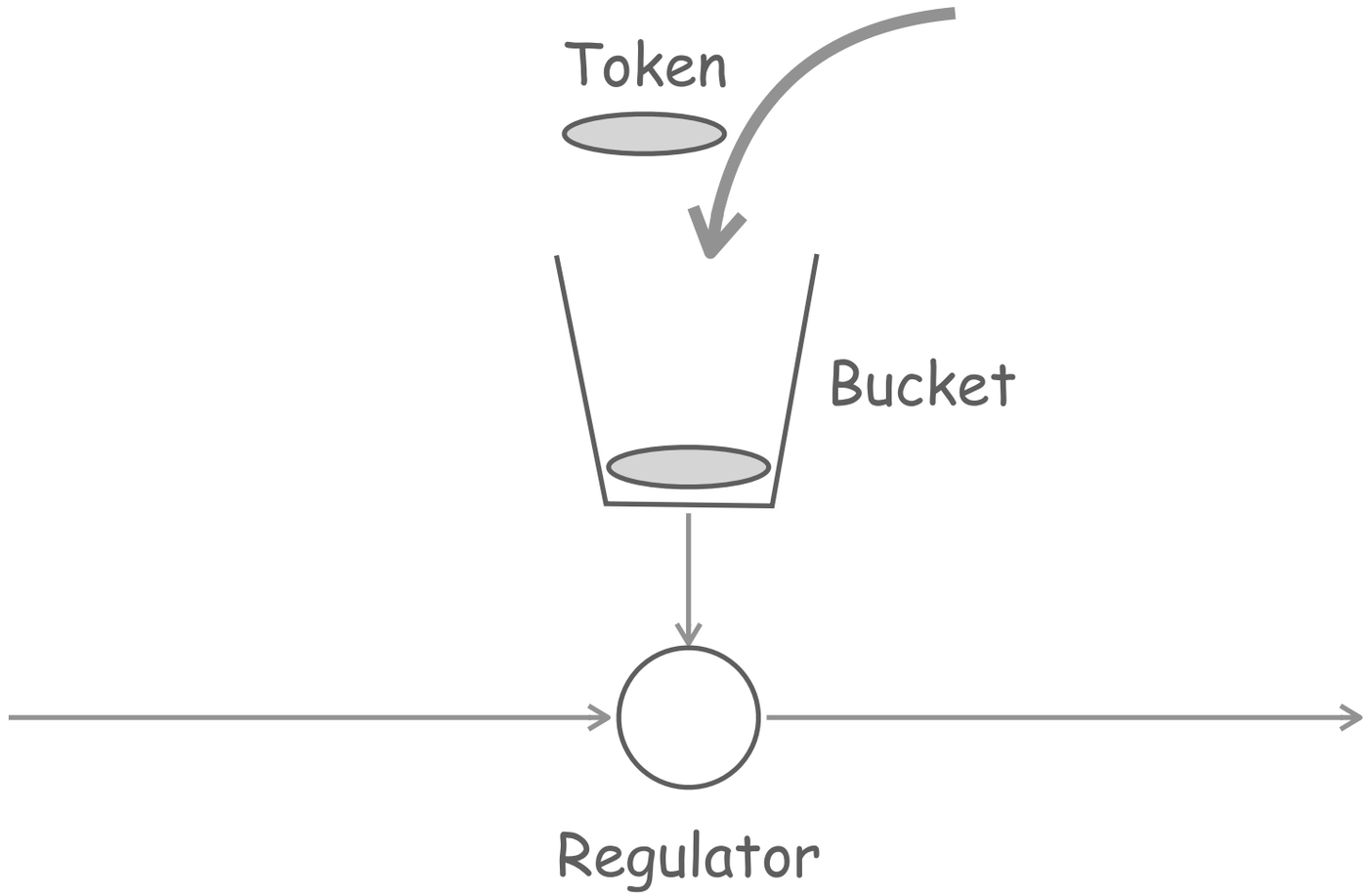
Buffer packets after reaching the limit

- ✓ Lower loss rate
- x Require memory
- x RTT inflation

Our Focus

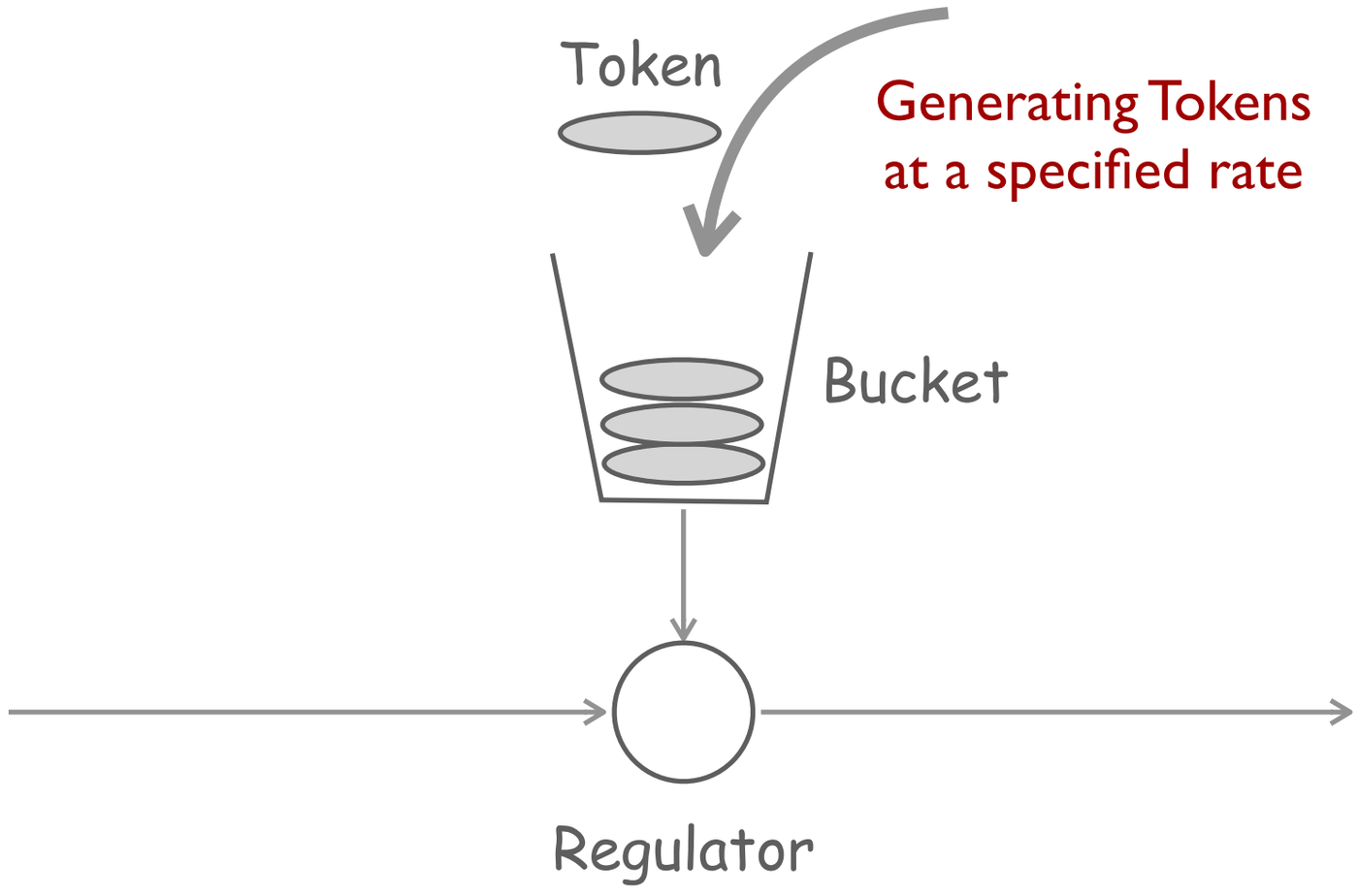
How Traffic Policer Works

Token Bucket Algorithm



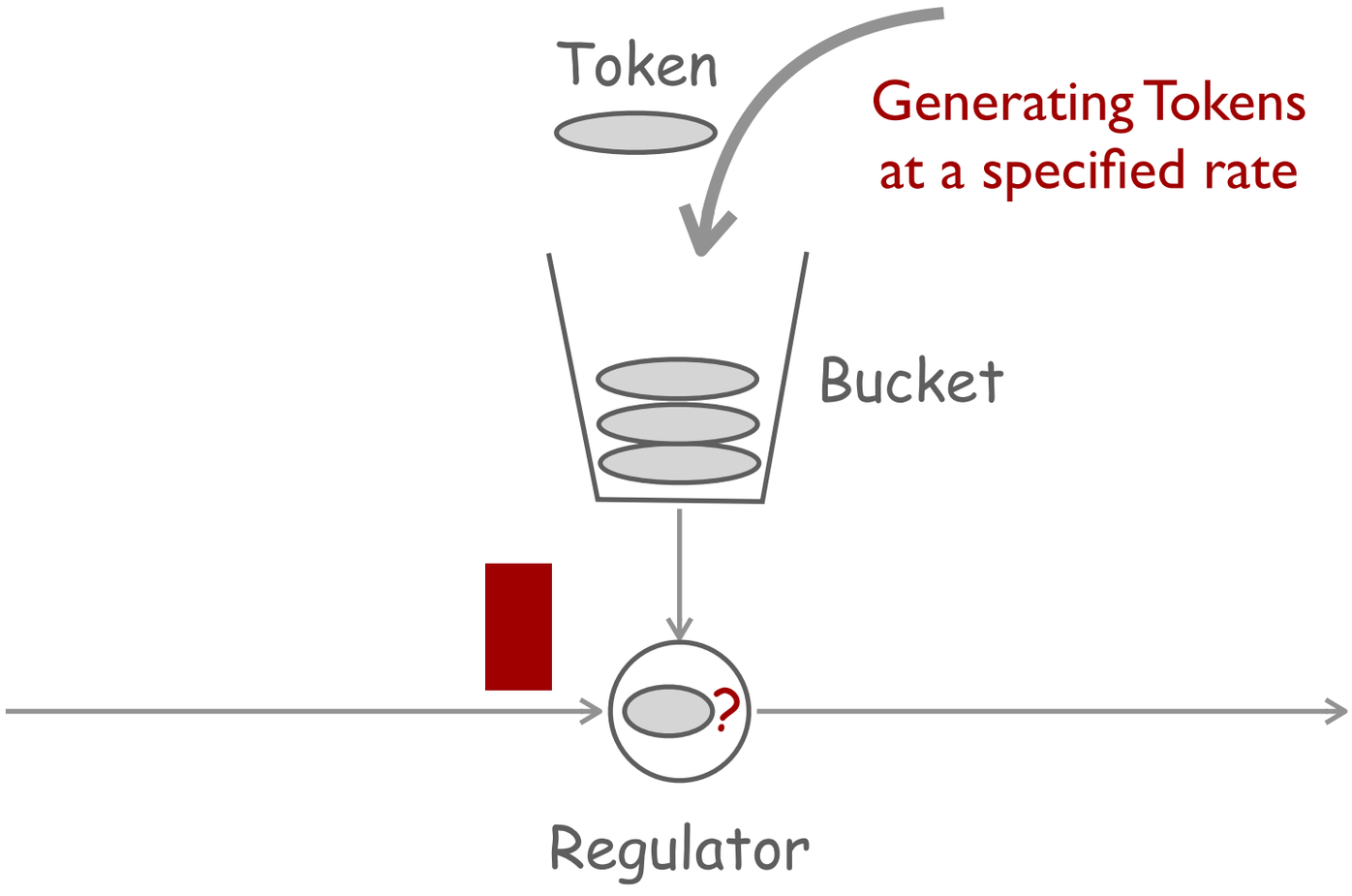
How Traffic Policer Works

Token Bucket Algorithm



How Traffic Policer Works

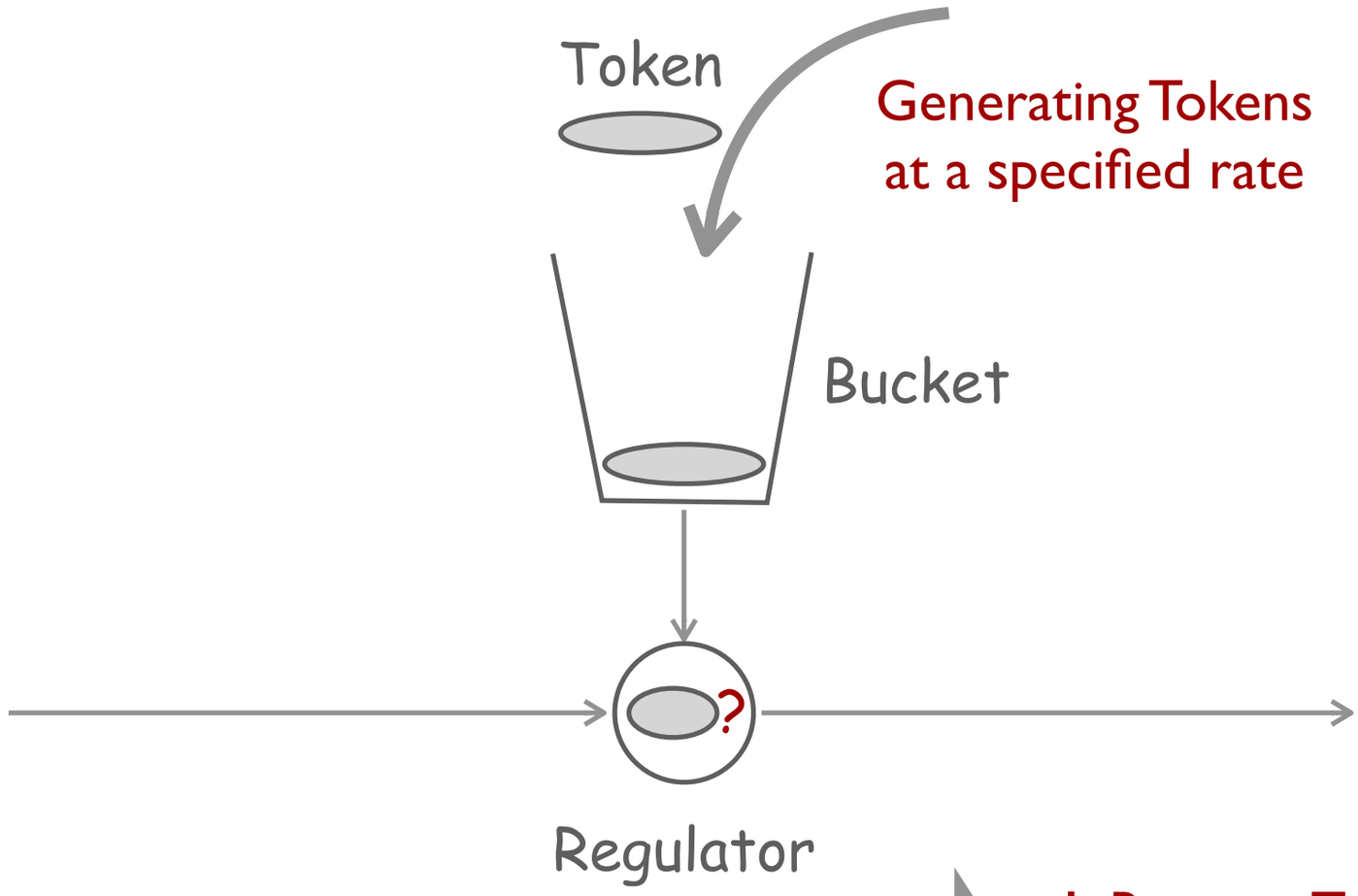
Token Bucket Algorithm



Token Num >= Packet Length

How Traffic Policer Works

Token Bucket Algorithm

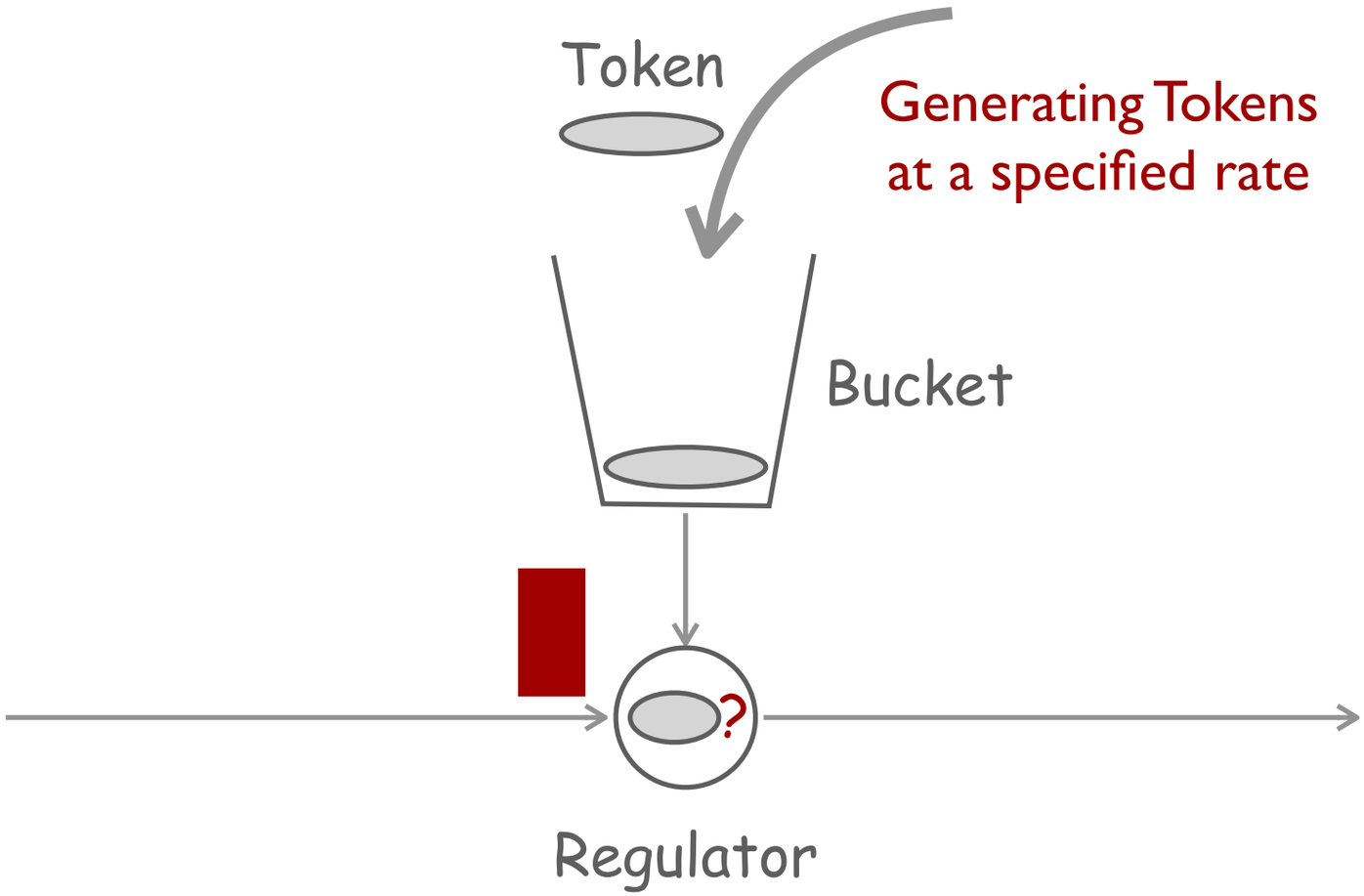


Token Num >= Packet Length

1. Remove Tokens from Bucket
2. Transmit Packet

How Traffic Policer Works

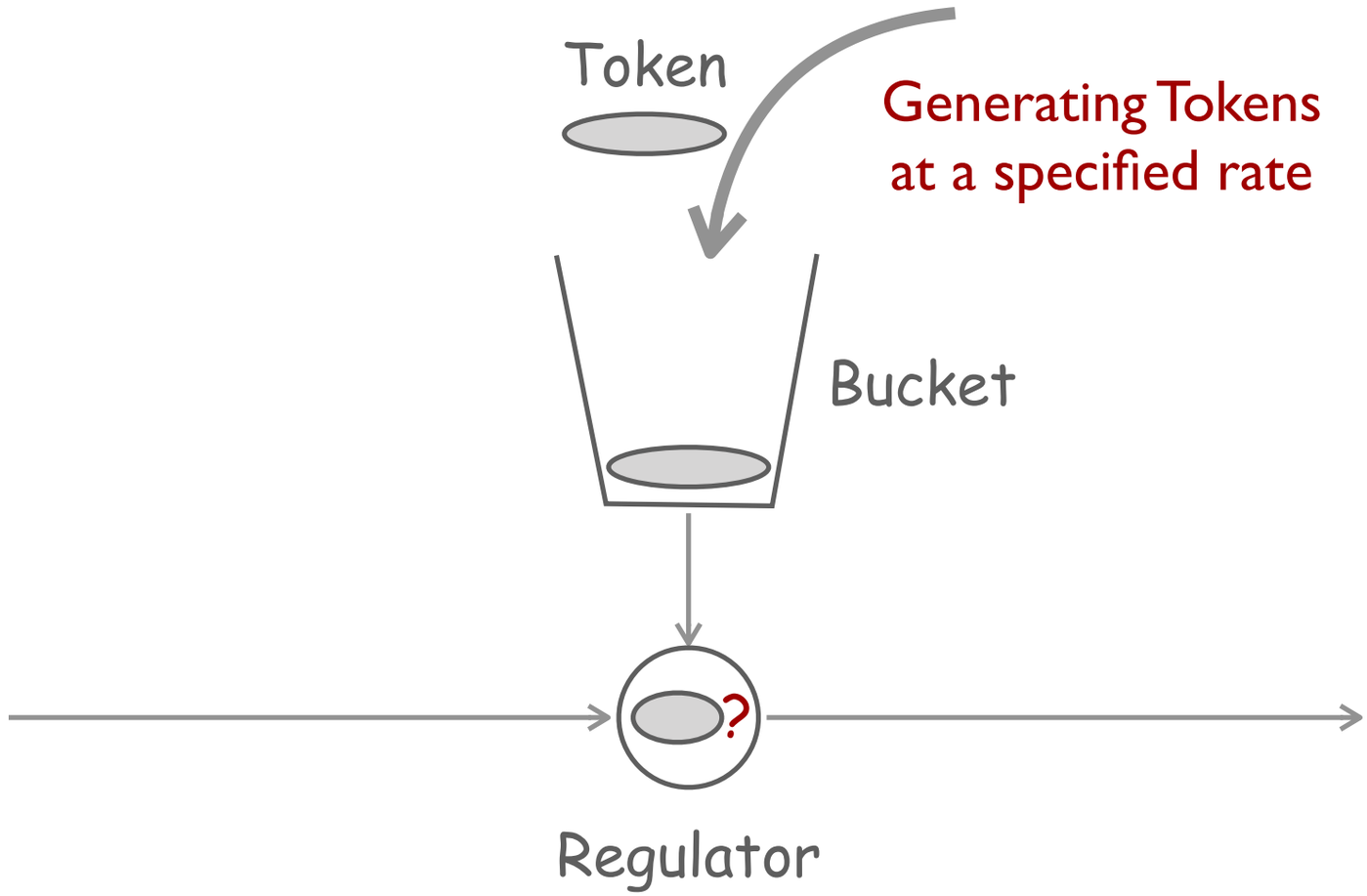
Token Bucket Algorithm



Token Num < Packet Length

How Traffic Policer Works

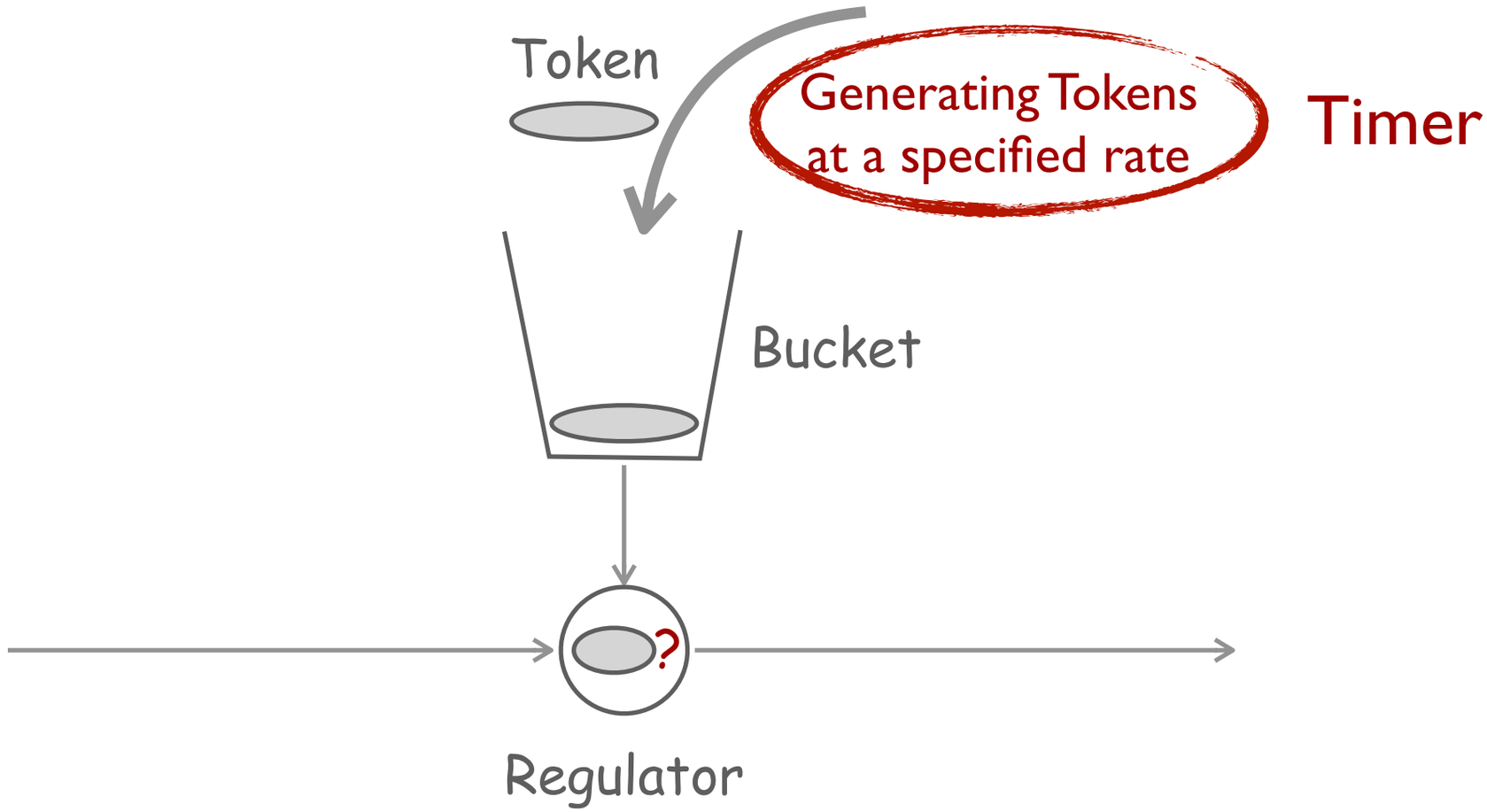
Token Bucket Algorithm



Token Num < Packet Length → Drop Packet

How Traffic Policer Works

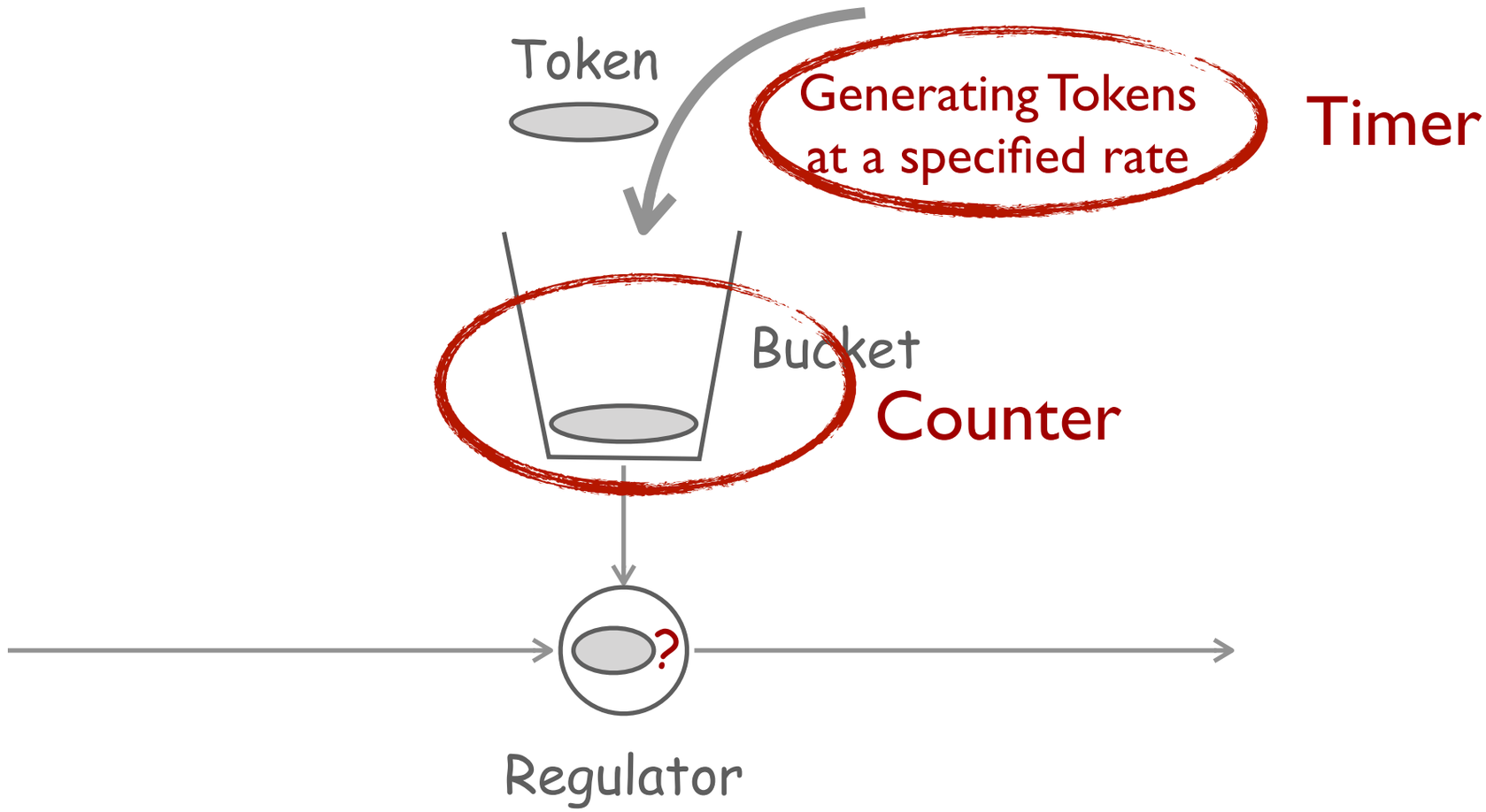
Token Bucket Algorithm



Token Num < Packet Length → Drop Packet

How Traffic Policer Works

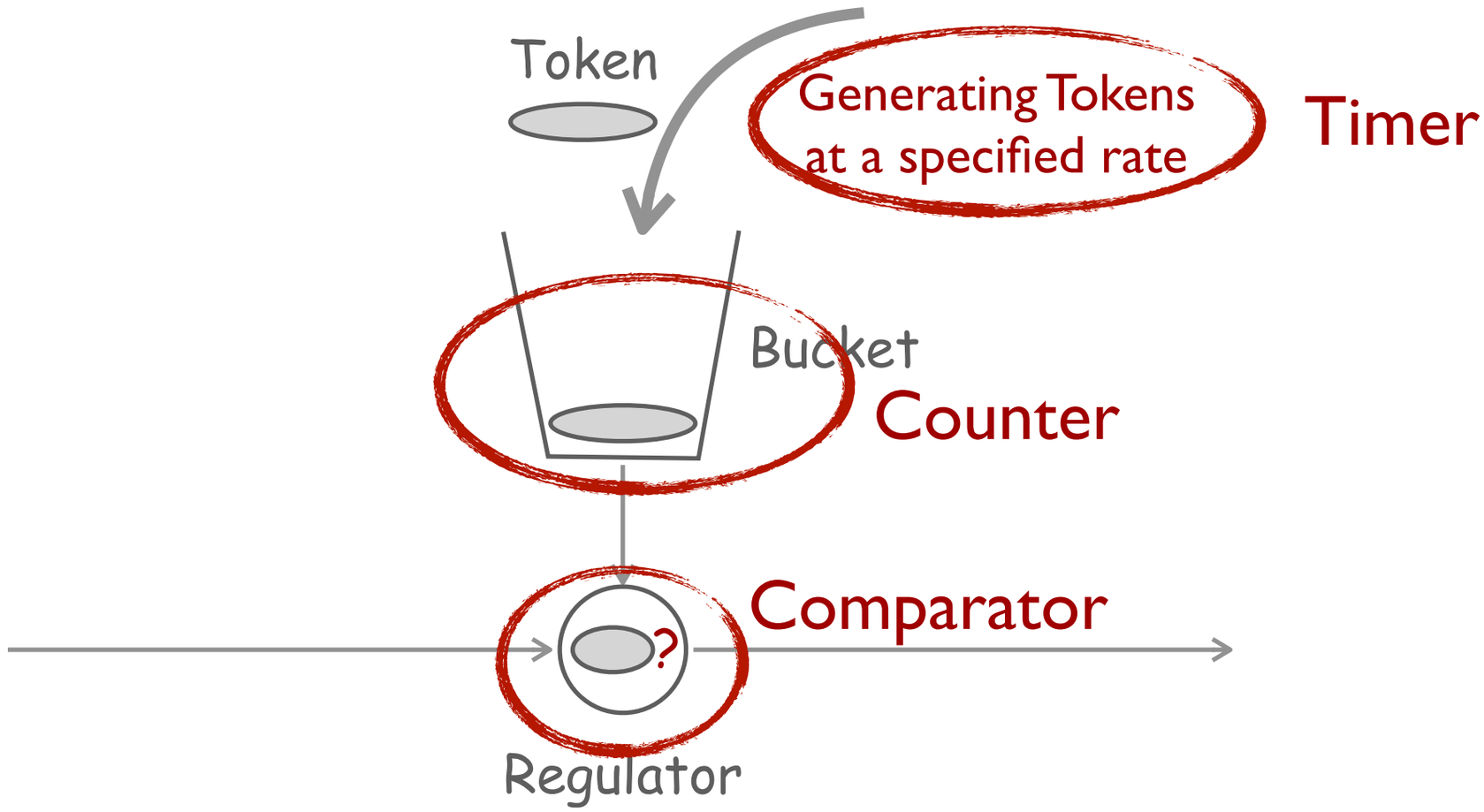
Token Bucket Algorithm



Token Num < Packet Length → Drop Packet

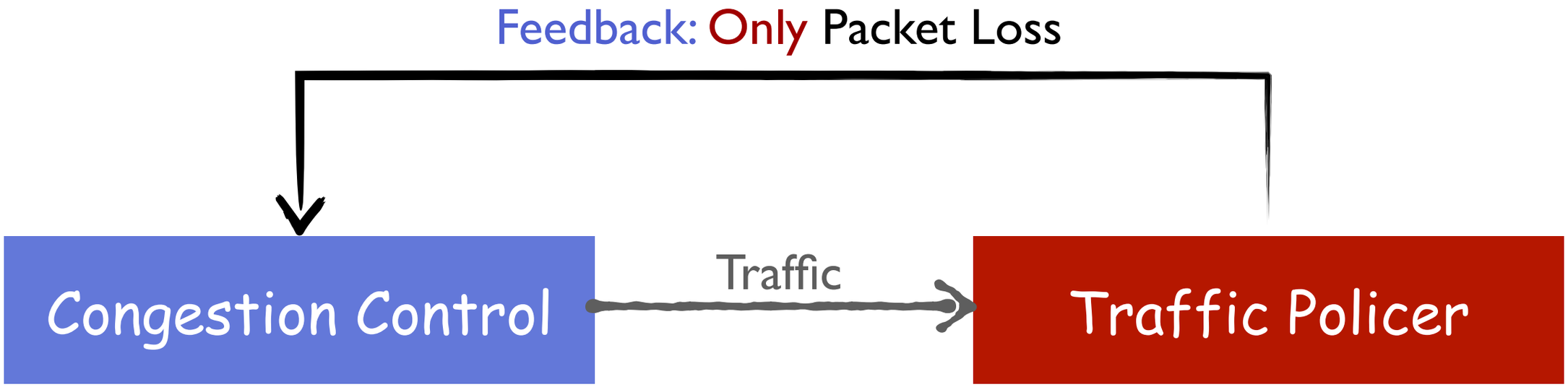
How Traffic Policer Works

Token Bucket Algorithm

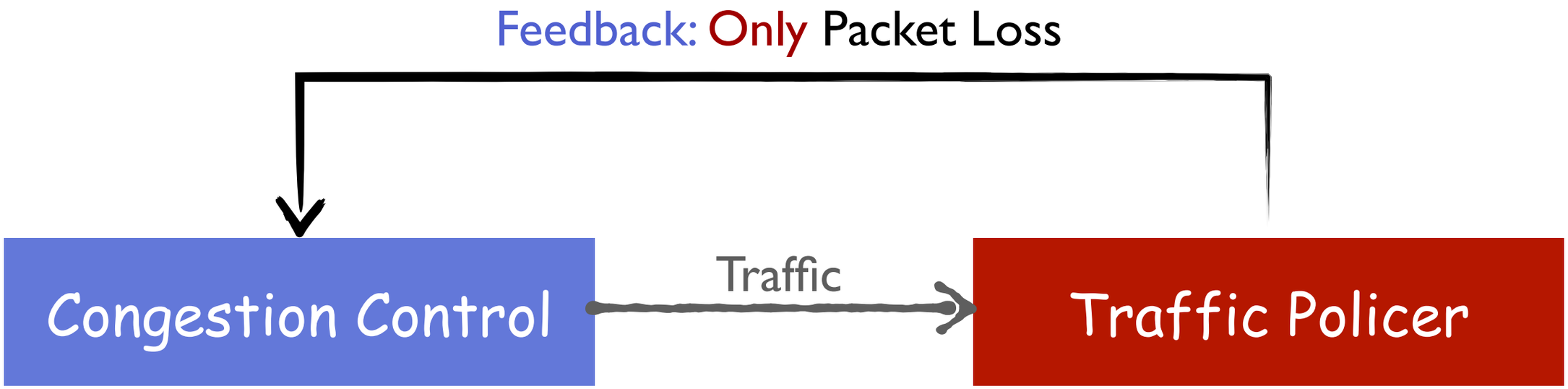


Token Num < Packet Length → Drop Packet

Congestion Control (CC) Interaction with Traffic Policer



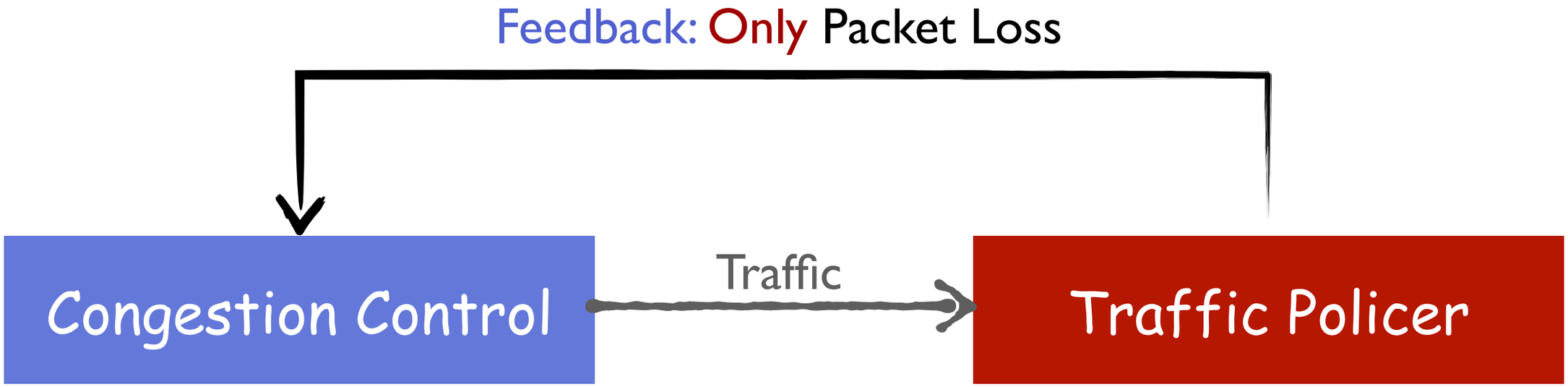
Congestion Control (CC) Interaction with Traffic Policer



Traditional CC (CUBIC)

Loss-sensitive

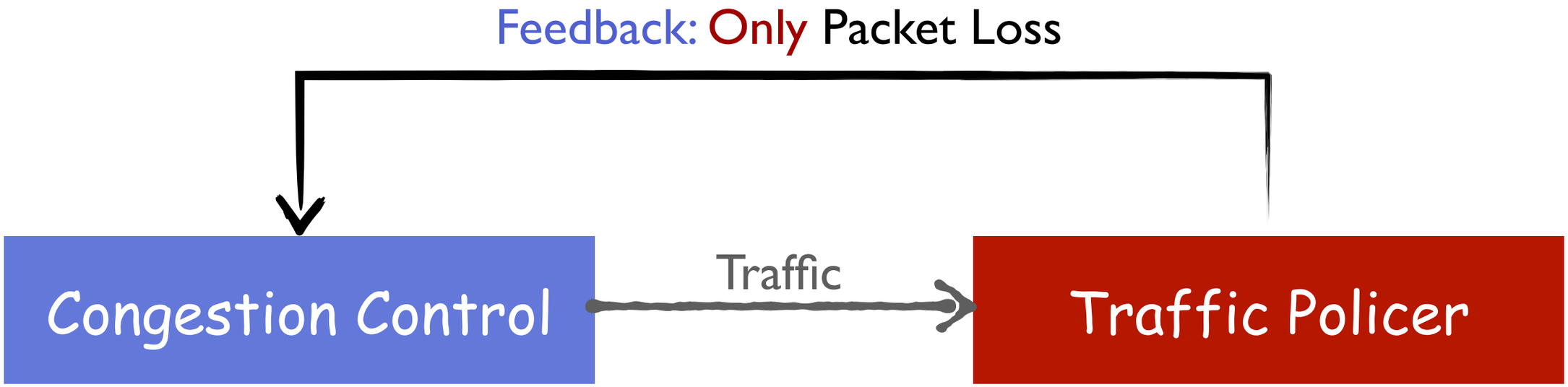
Congestion Control (CC) Interaction with Traffic Policer



Traditional CC (CUBIC)

Loss-sensitive

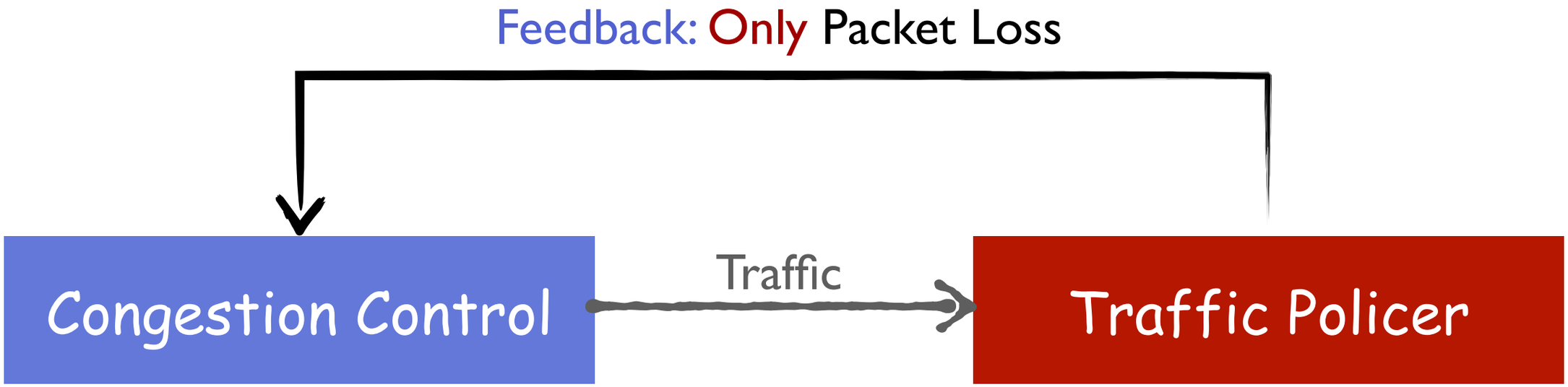
Congestion Control (CC) Interaction with Traffic Policer



Traditional CC (CUBIC)
Loss-sensitive

New CC (BBR, PCC)
Loss-resilient

Congestion Control (CC) Interaction with Traffic Policer



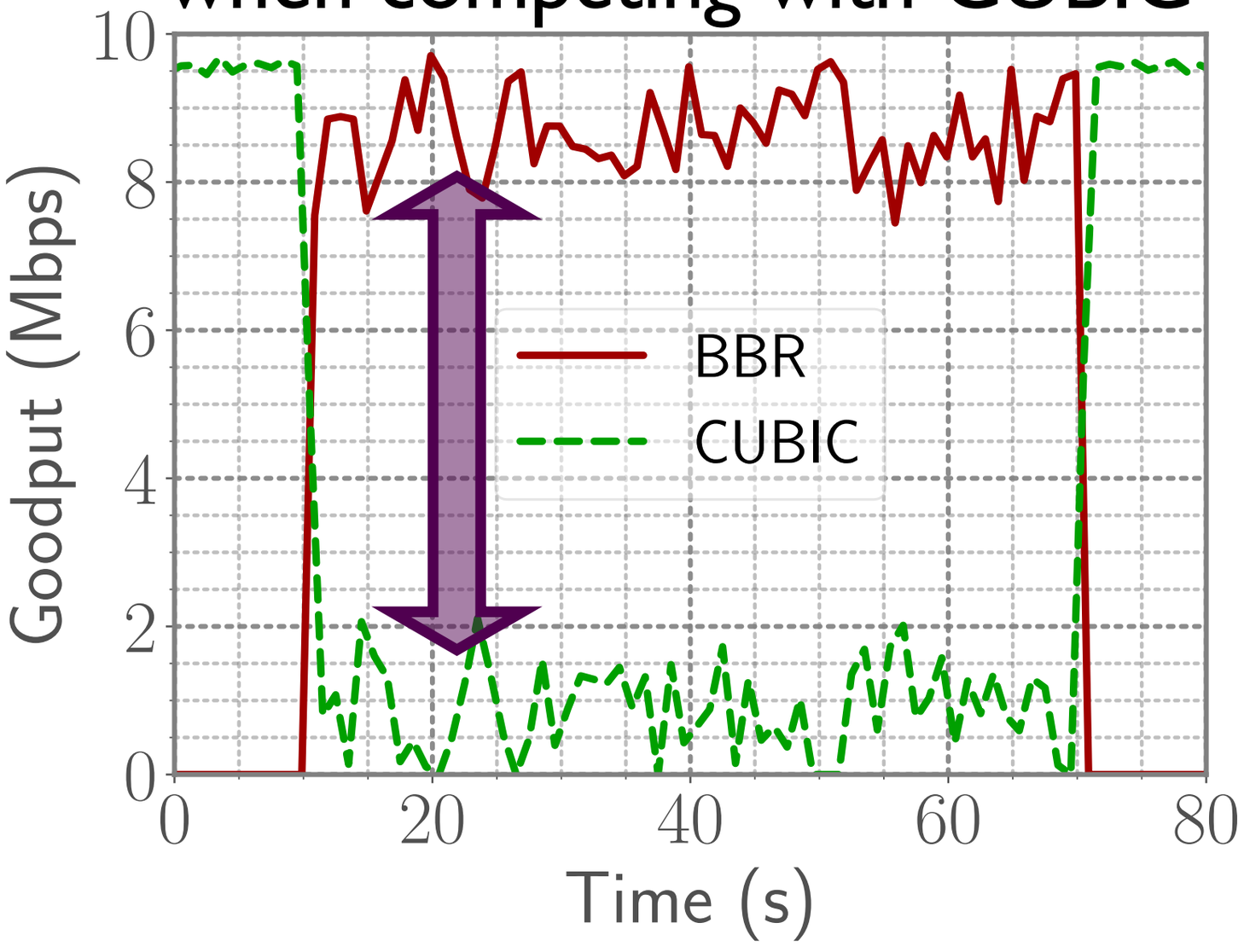
Traditional CC (CUBIC)
Loss-sensitive

New CC (BBR, PCC)
Loss-resilient

Traditional CC vs. new CC ⟶ Unfairness

The Unfairness Problem

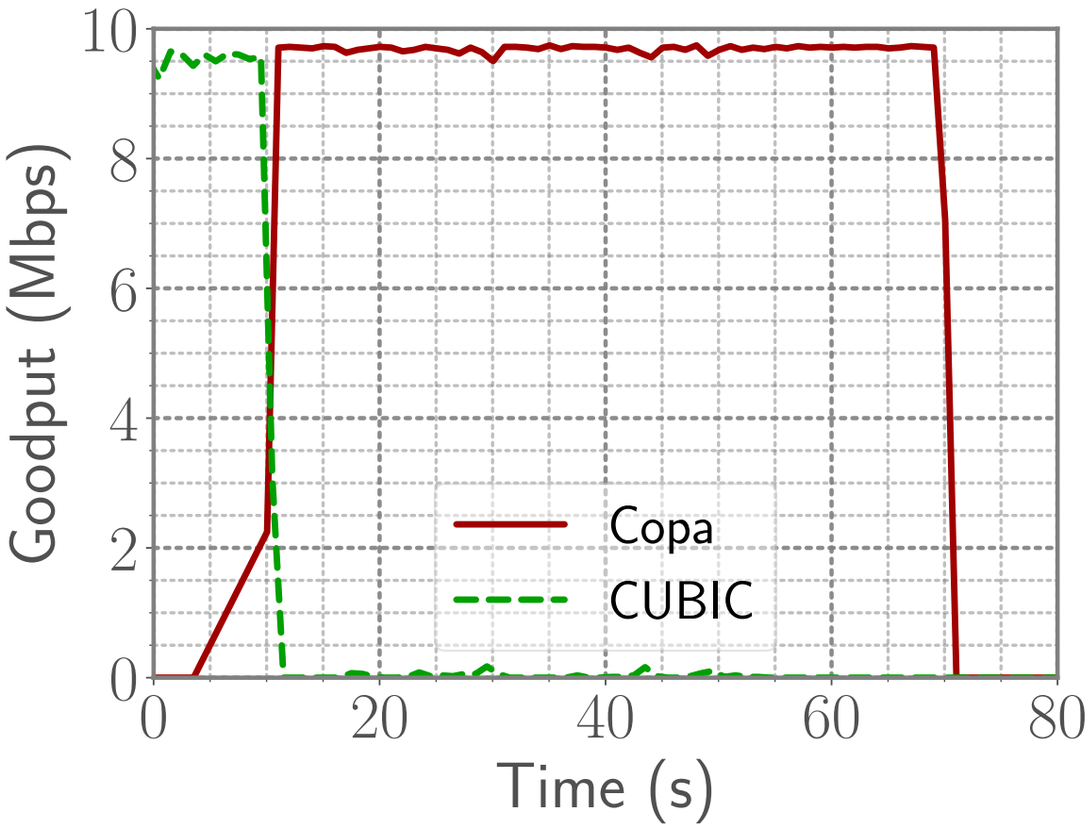
BBR occupies 90.6% bandwidth
when competing with CUBIC



1 CUBIC vs. 1 BBR

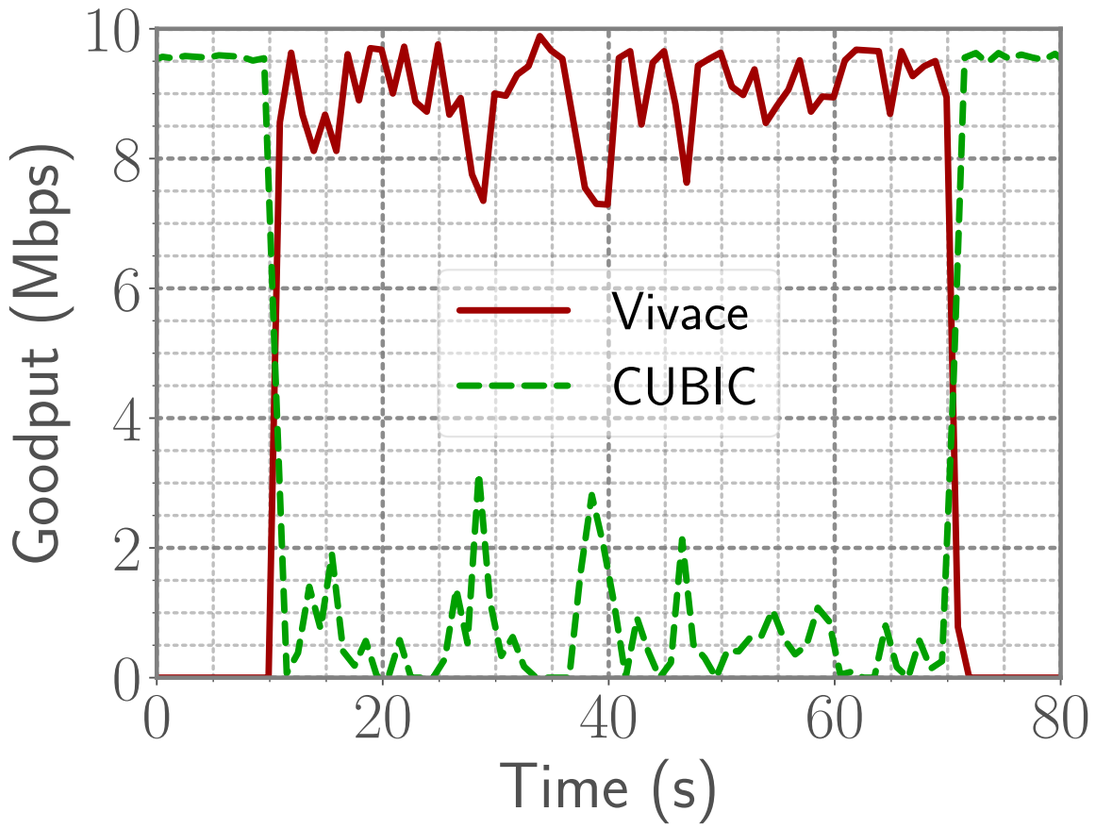
The Unfairness Problem

Copa occupies 99.8% bandwidth



1 CUBIC vs. 1 Copa

PCC Vivace occupies 93.8% bandwidth



1 CUBIC vs. 1 PCC Vivace

More in Paper: How new CC occupies the majority of bandwidth

How to Tackle the Unfairness Problem?

How to Tackle the Unfairness Problem?

Improve Congestion Control

How to Tackle the Unfairness Problem?

Improve Congestion Control

Not Practicable:
Content Providers want higher bandwidth

How to Tackle the Unfairness Problem?

Improve Congestion Control

Use Traffic Shaping

Not Practicable:
Content Providers want higher bandwidth

How to Tackle the Unfairness Problem?

Improve Congestion Control

Not Practicable:
Content Providers want higher bandwidth

Use Traffic Shaping

Inflate RTT
Increase Overhead

How to Tackle the Unfairness Problem?

Our Approach

Improve Congestion Control

Not Practicable:
Content Providers want higher bandwidth

Use Traffic Shaping

Inflate RTT
Increase Overhead

FairPolicer: Enforce Fairness in the Policer

Design of FairPolicer

Observation

Bandwidth is allocated in the unit of Tokens

Design of FairPolicer

Observation

Bandwidth is allocated in the unit of Tokens

↓

Goal

Fairly allocate tokens to flows

Design of FairPolicer

Observation

Bandwidth is allocated in the unit of Tokens

↓

Goal

Fairly allocate tokens to flows

↓

Basic Idea

Per-flow Token Buckets

Design of FairPolicer

Observation

Bandwidth is allocated in the unit of Tokens

↓

Goal

Fairly allocate tokens to flows

↓

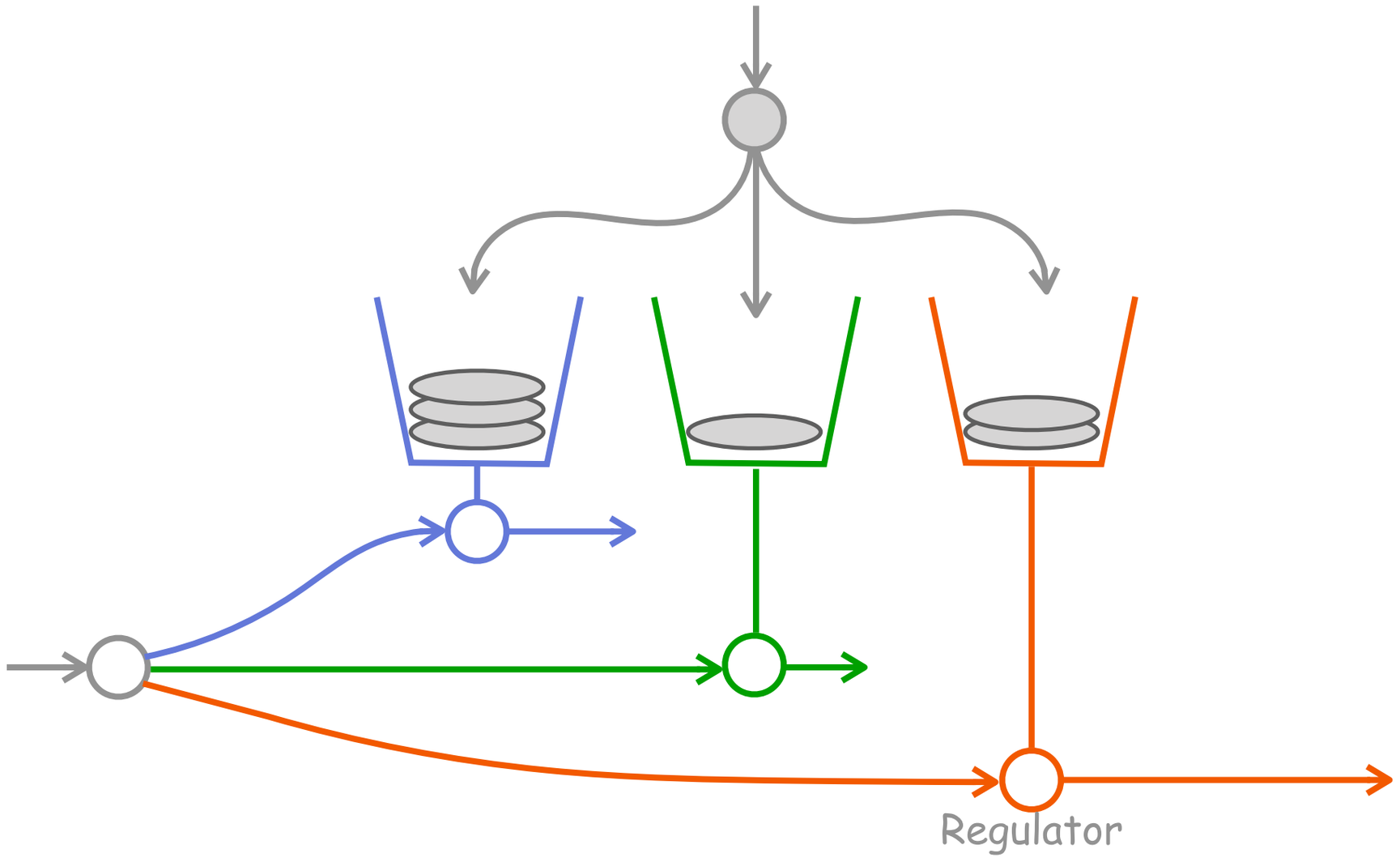
Basic Idea

Per-flow Token Buckets

↑

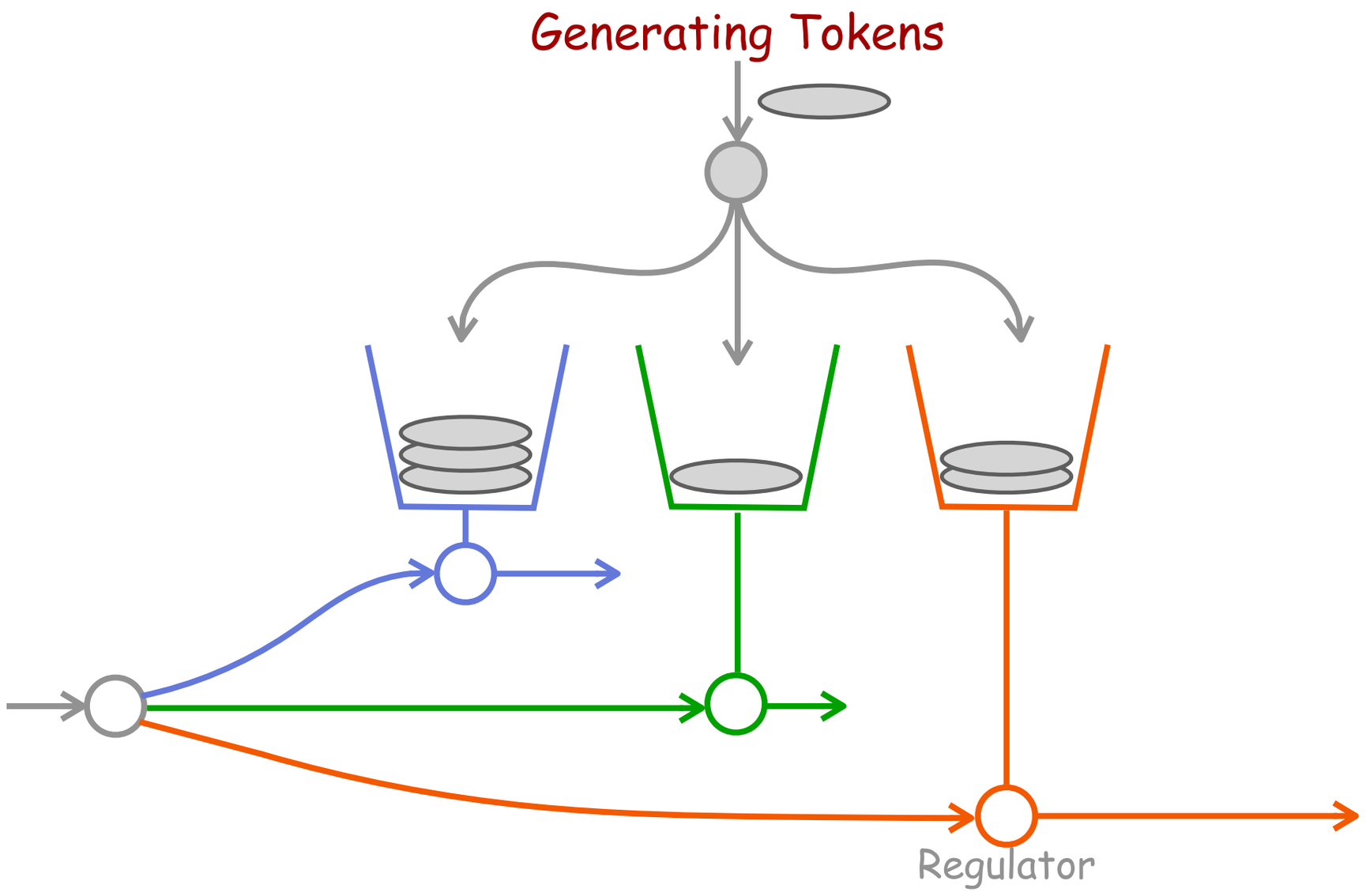
Token bucket is a very simple structure
(Counter)

Design of FairPolicer



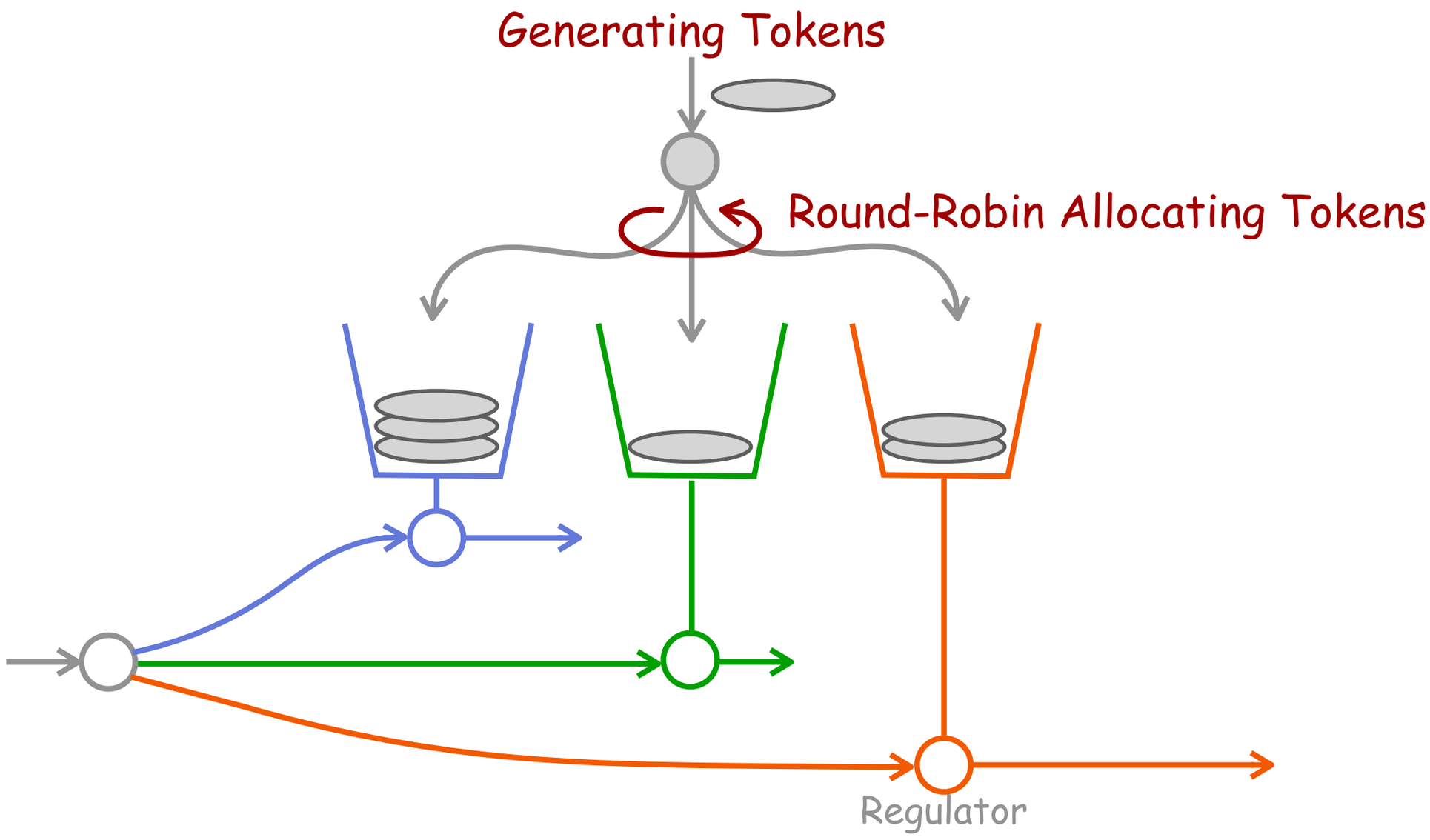
Basic Idea

Design of FairPolicer



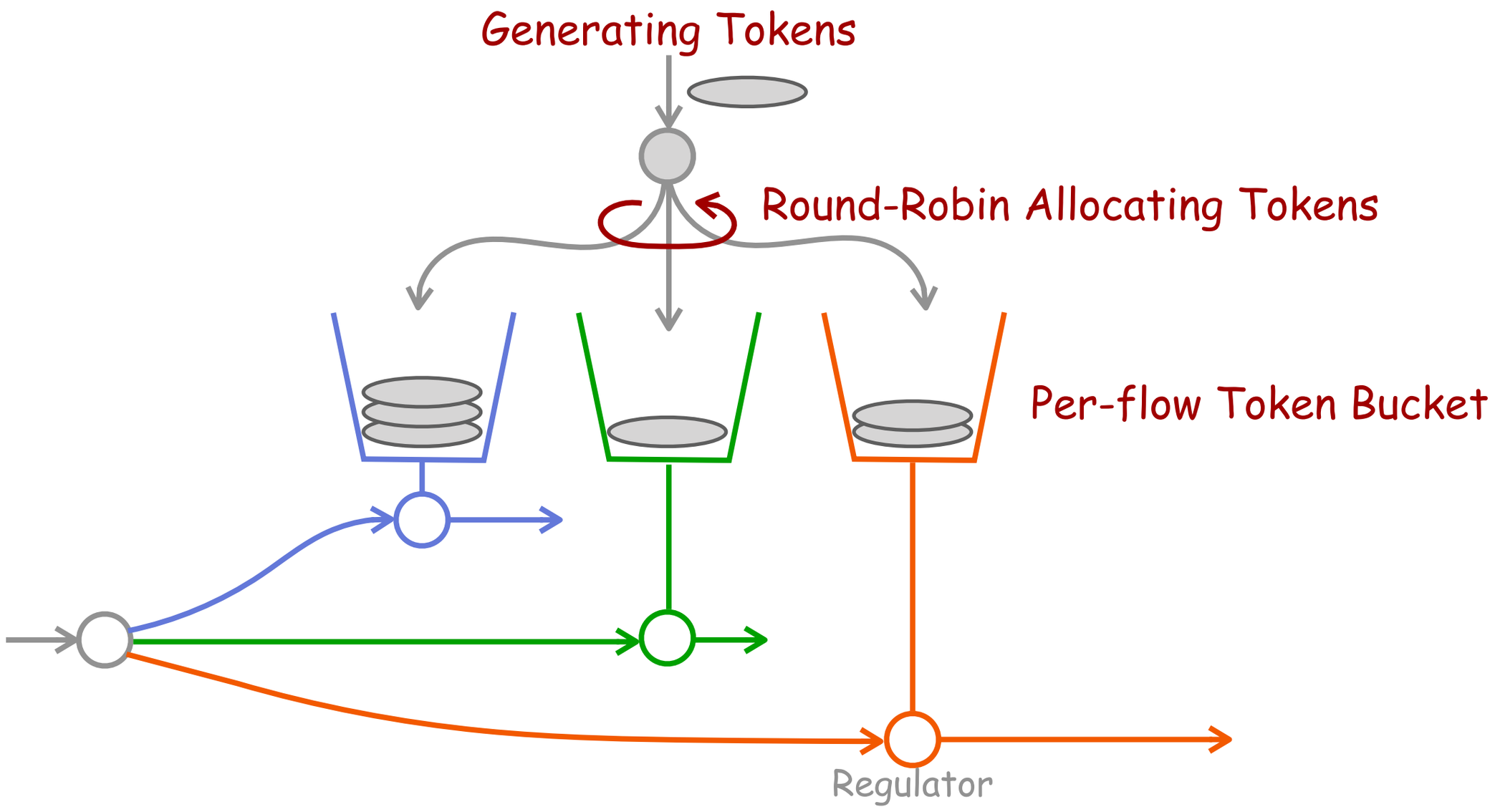
Basic Idea

Design of FairPolicer



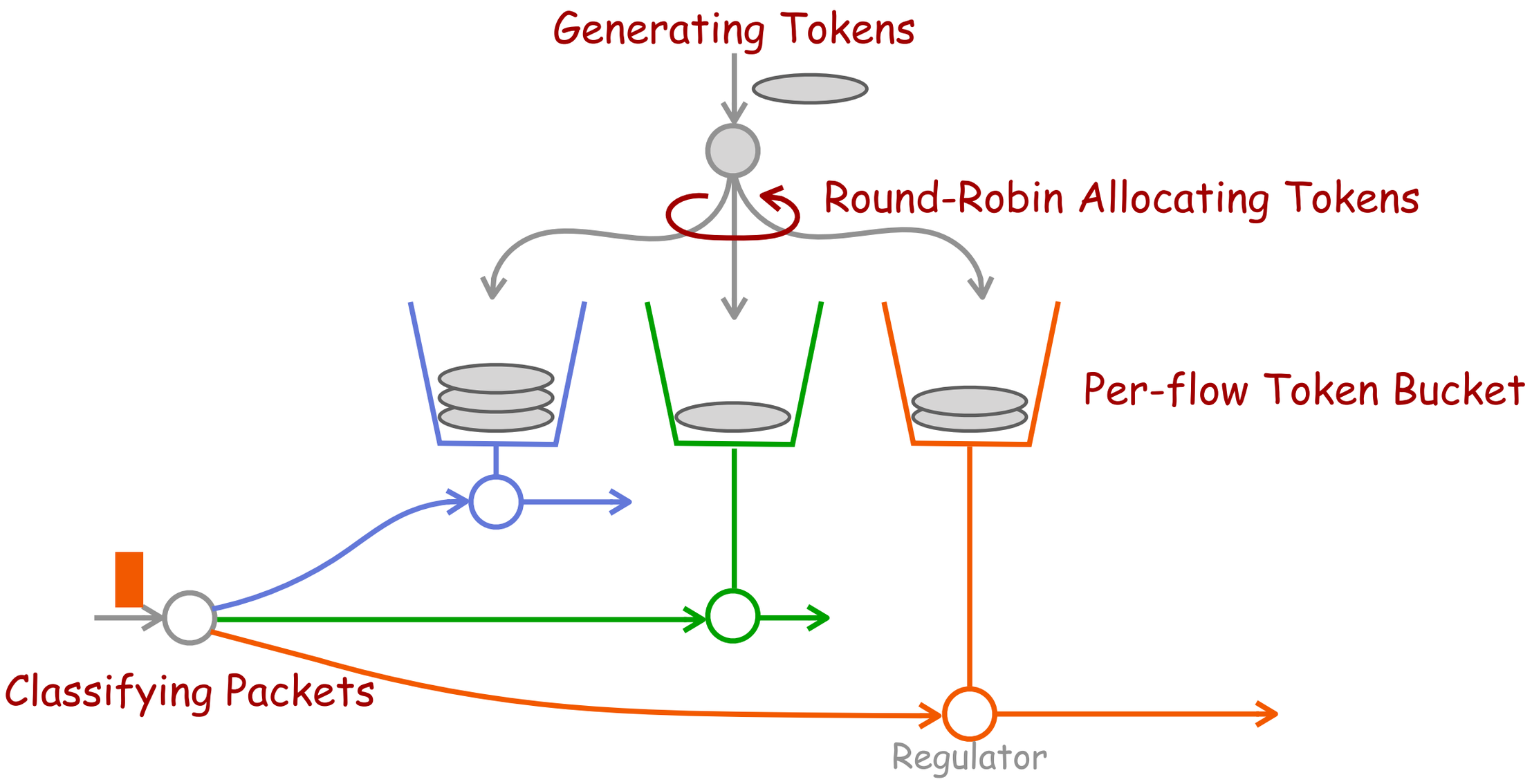
Basic Idea

Design of FairPolicer



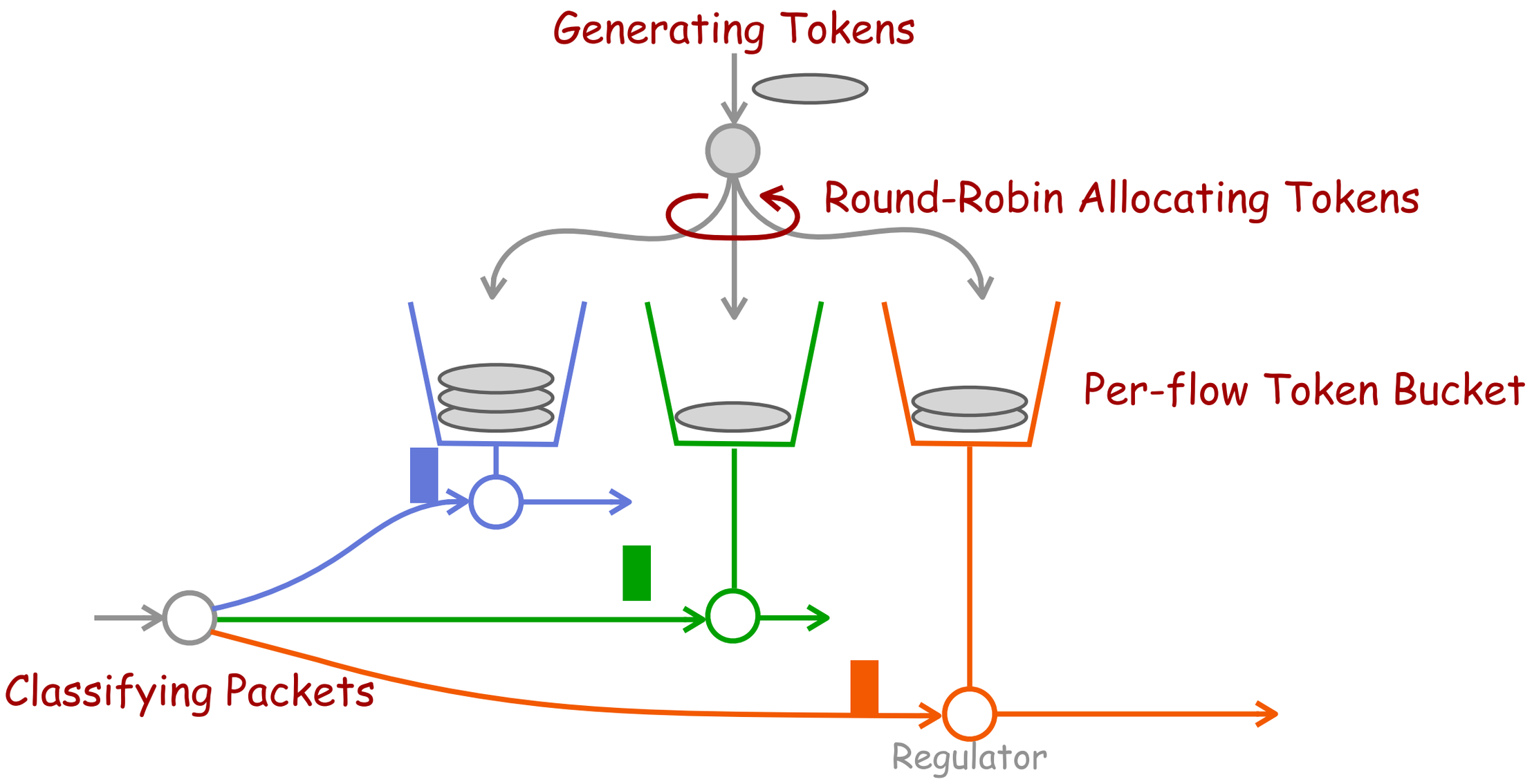
Basic Idea

Design of FairPolicer



Basic Idea

Design of FairPolicer

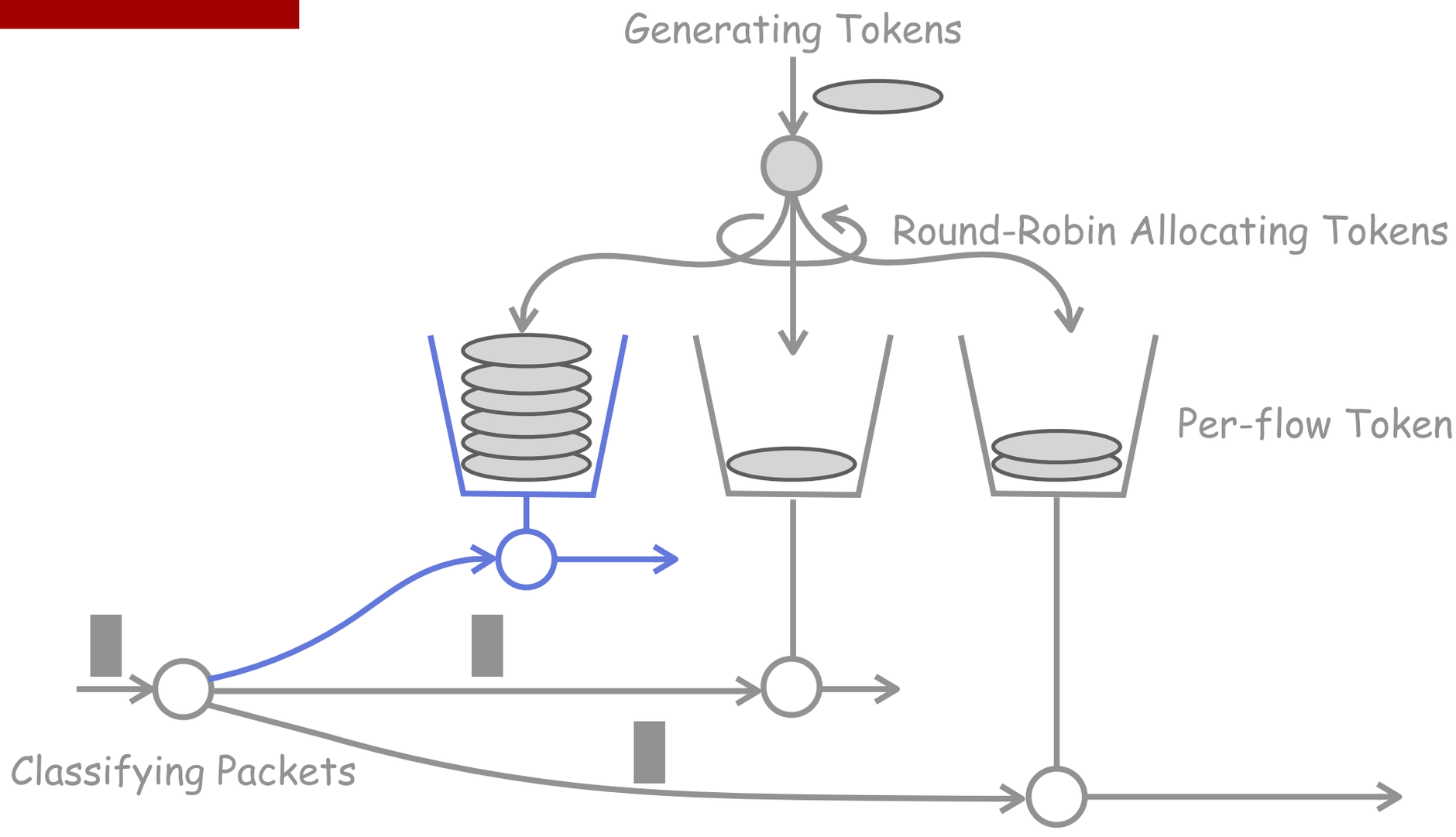


Basic Idea

Design of FairPolicer

Challenges #1: Flows Come and Go

Flow becomes inactive

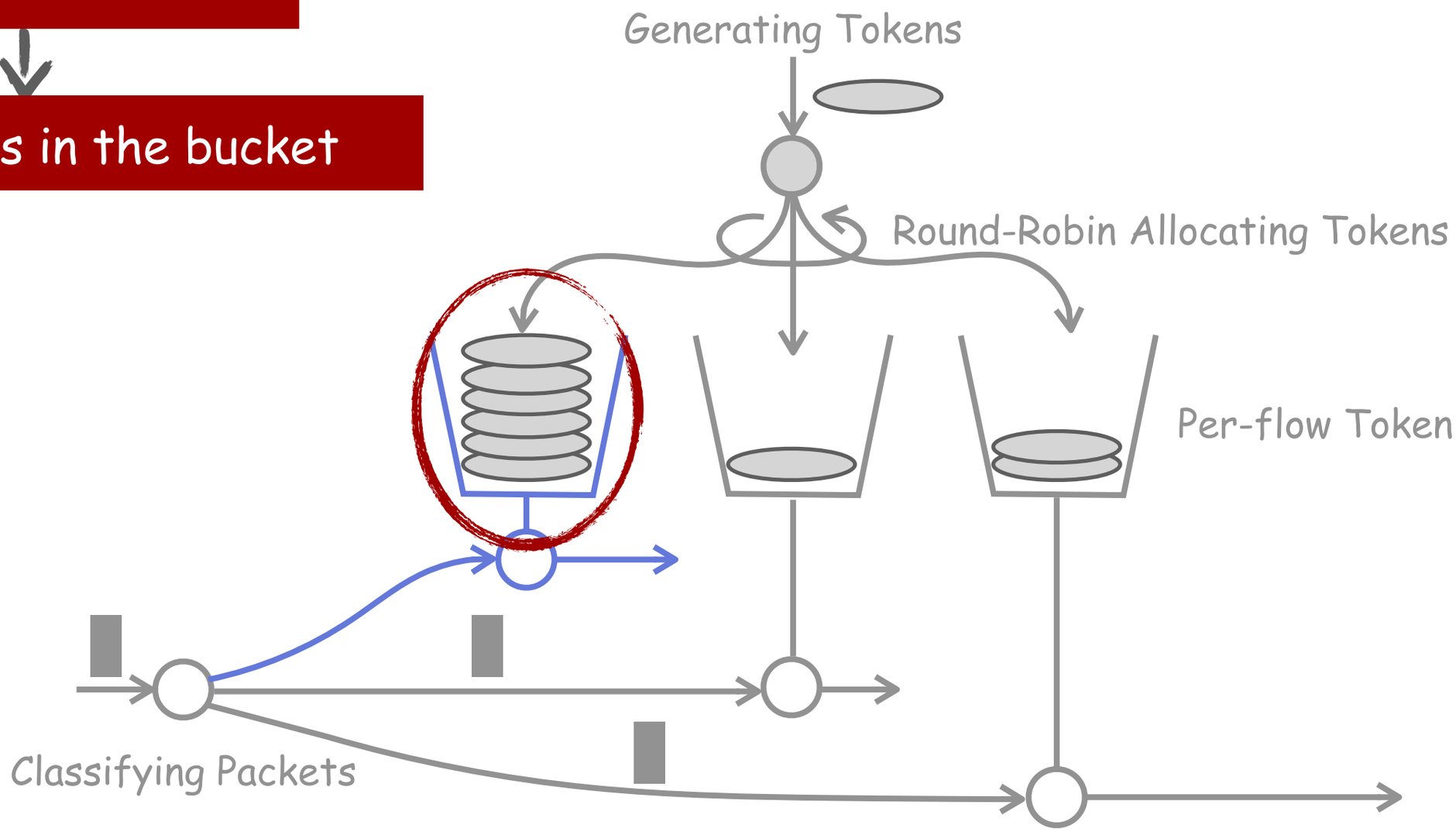


Design of FairPolicer

Challenges #1: Flows Come and Go

Flow becomes inactive

Full of tokens in the bucket



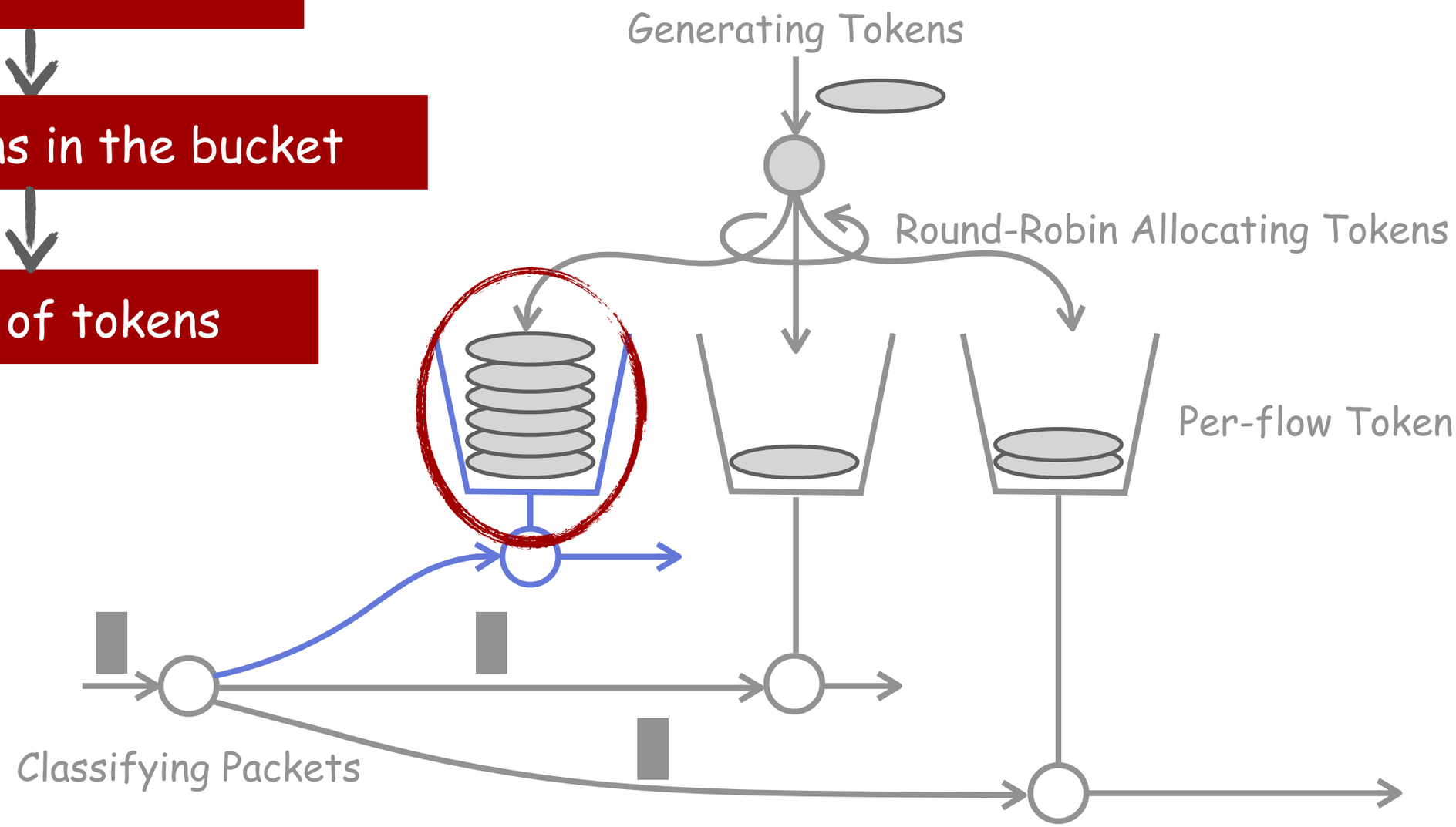
Design of FairPolicer

Challenges #1: Flows Come and Go

Flow becomes inactive

Full of tokens in the bucket

Waste of tokens



Design of FairPolicer

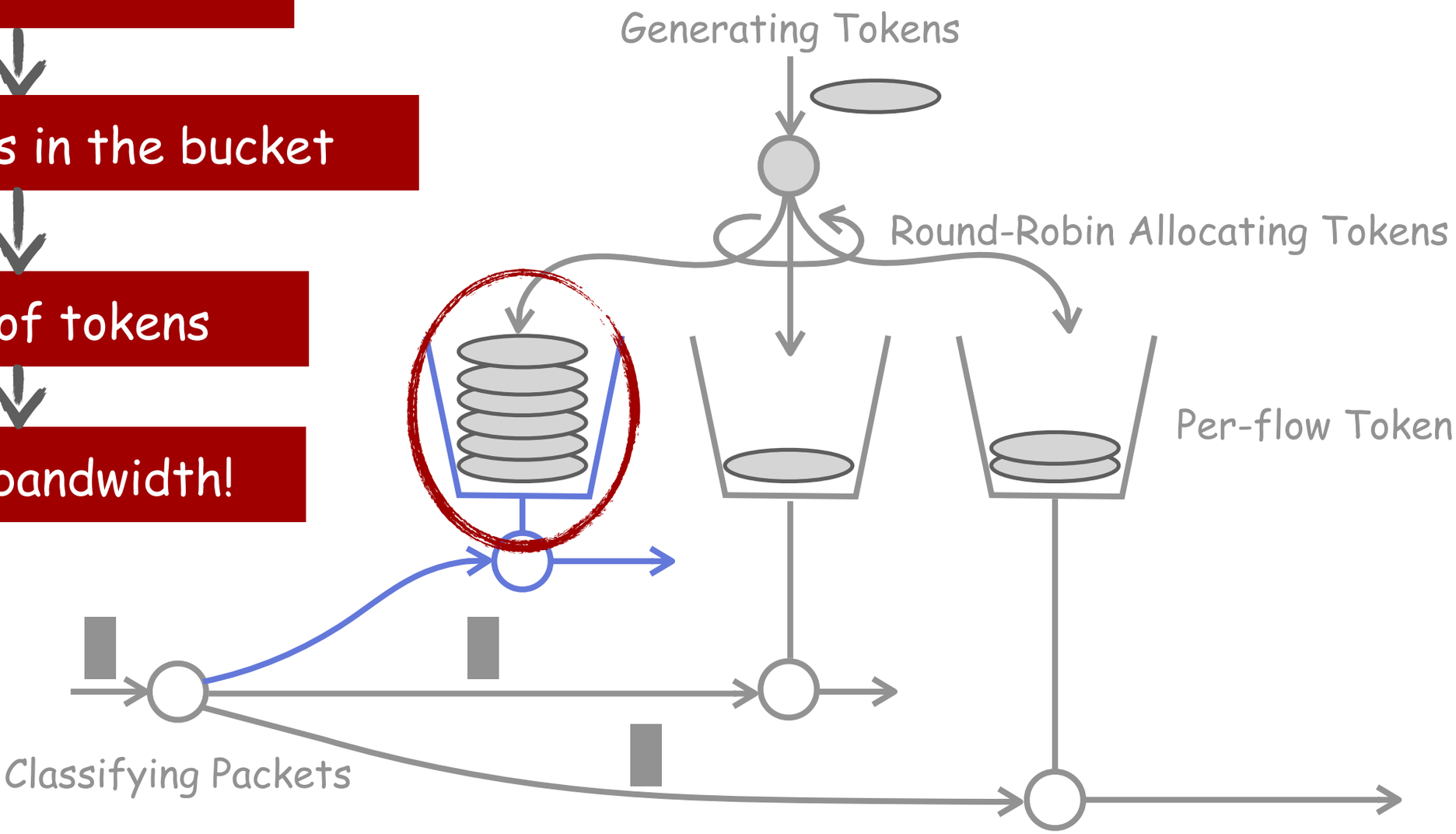
Challenges #1: Flows Come and Go

Flow becomes inactive

Full of tokens in the bucket

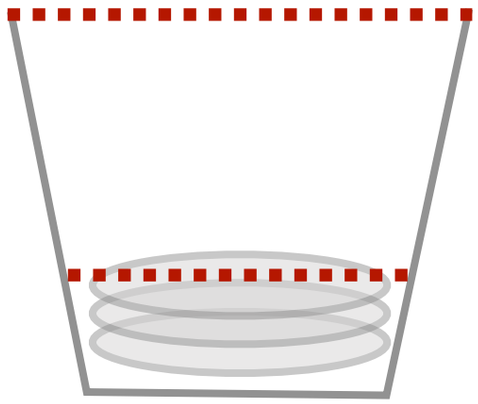
Waste of tokens

Waste of bandwidth!



Design of FairPolicer

Address Challenges #1



Token Bucket

Design of FairPolicer

Address Challenges #1

Token Bucket

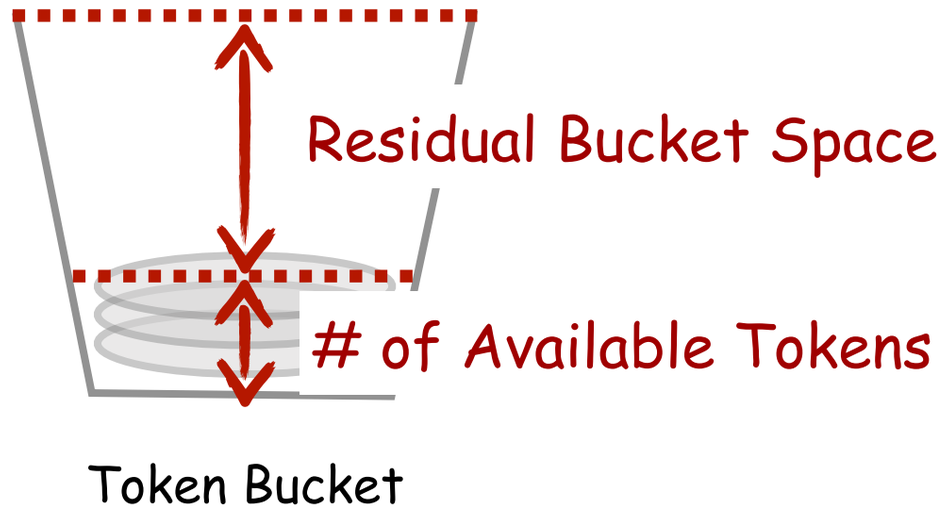
Design of FairPolicer

Address Challenges #1

Token Bucket

Design of FairPolicer

Address Challenges #1



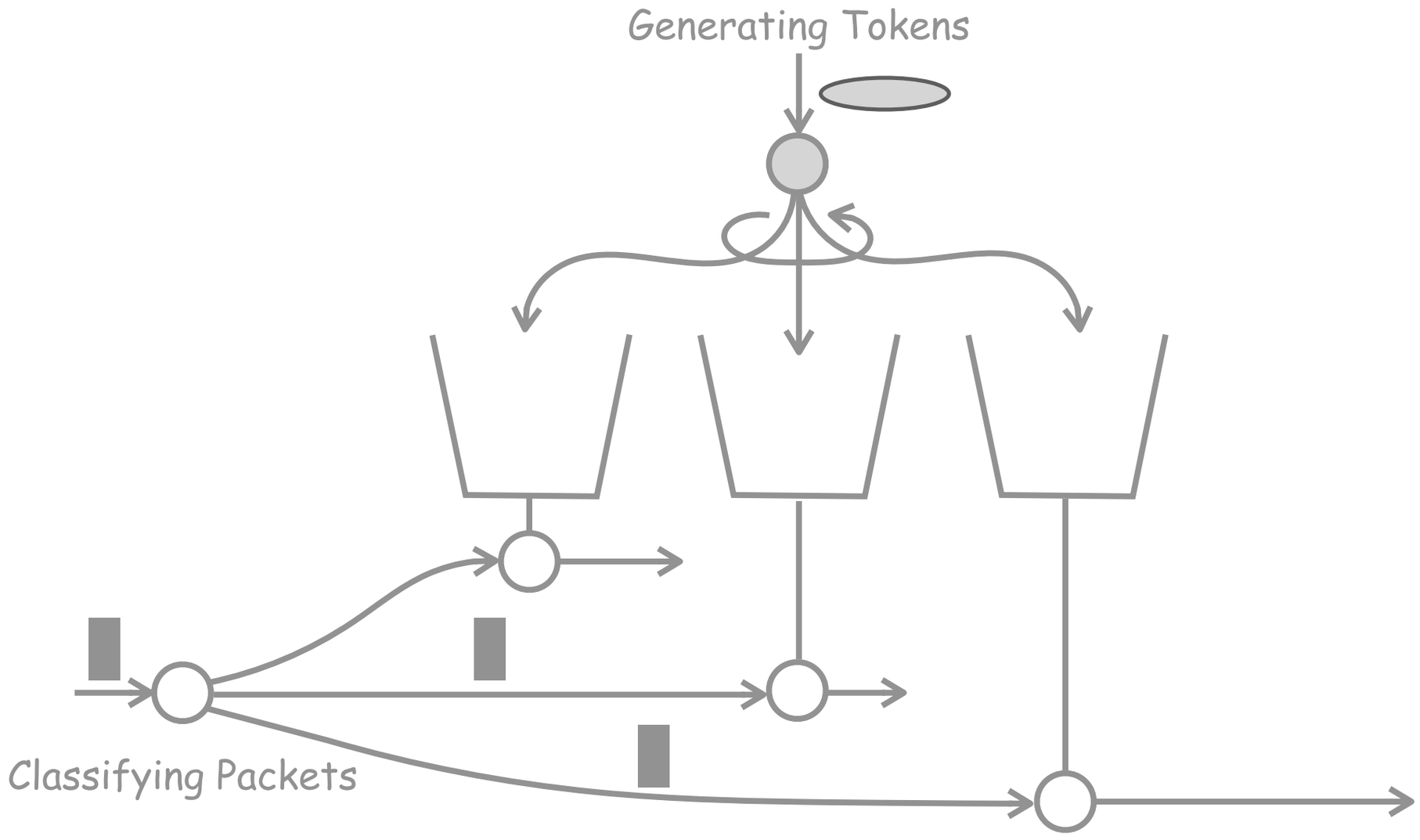
Token Bucket

Token Bucket Full ⟷ Residual Bucket Space = 0

Residual bucket space instead of # of available tokens

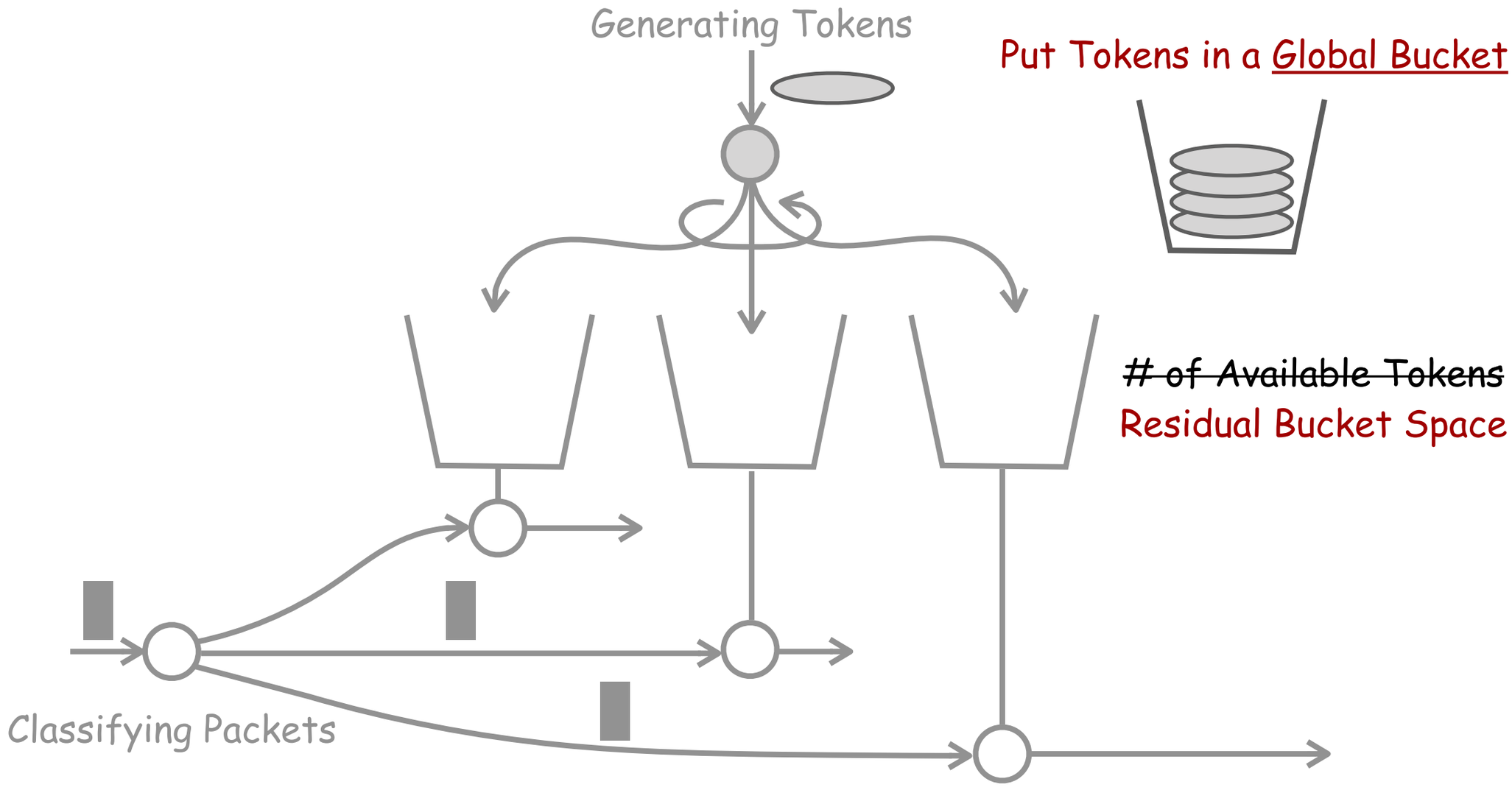
Design of FairPolicer

Address Challenges #1



Design of FairPolicer

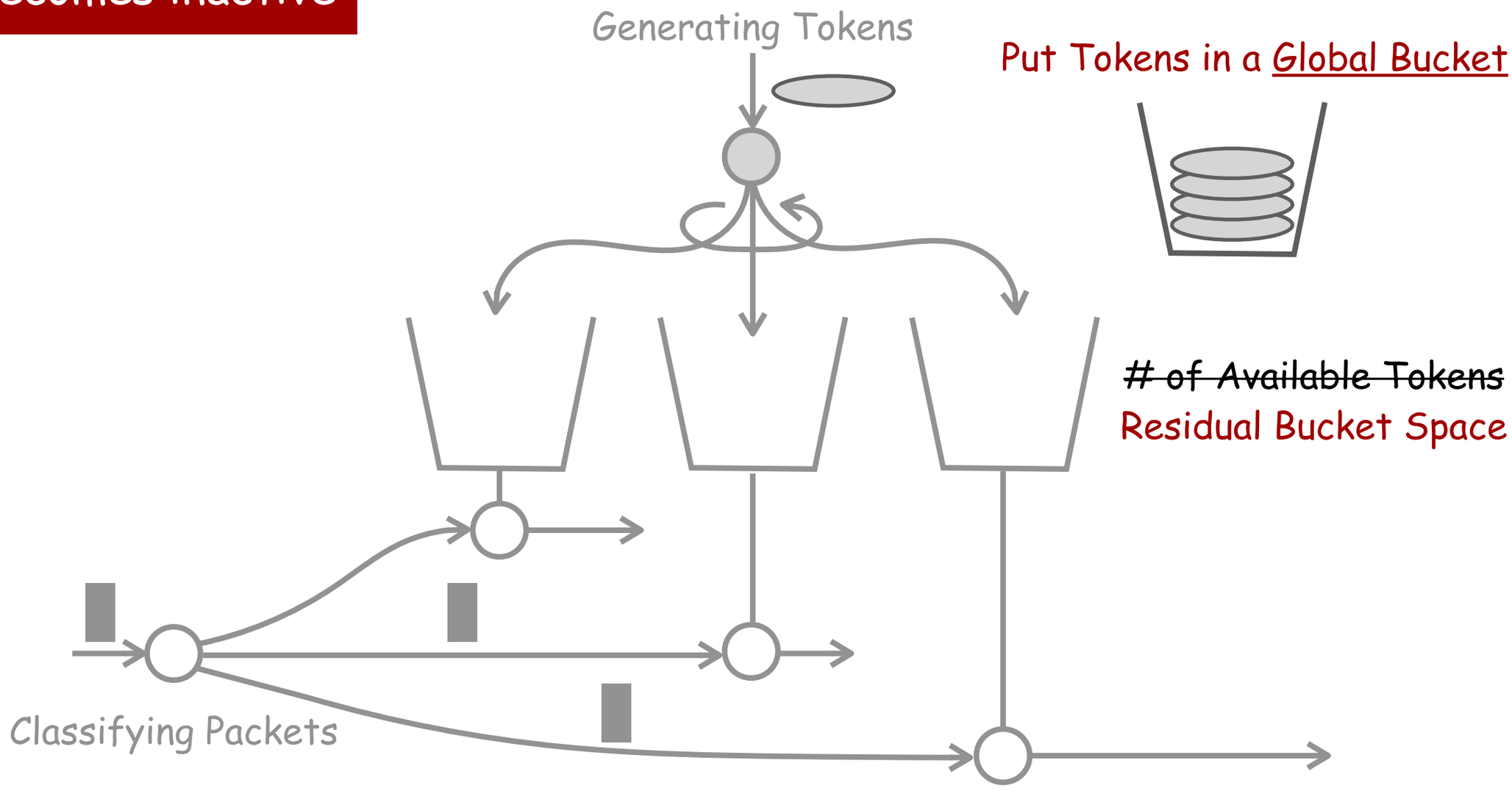
Address Challenges #1



Design of FairPolicer

Address Challenges #1

Flow becomes inactive



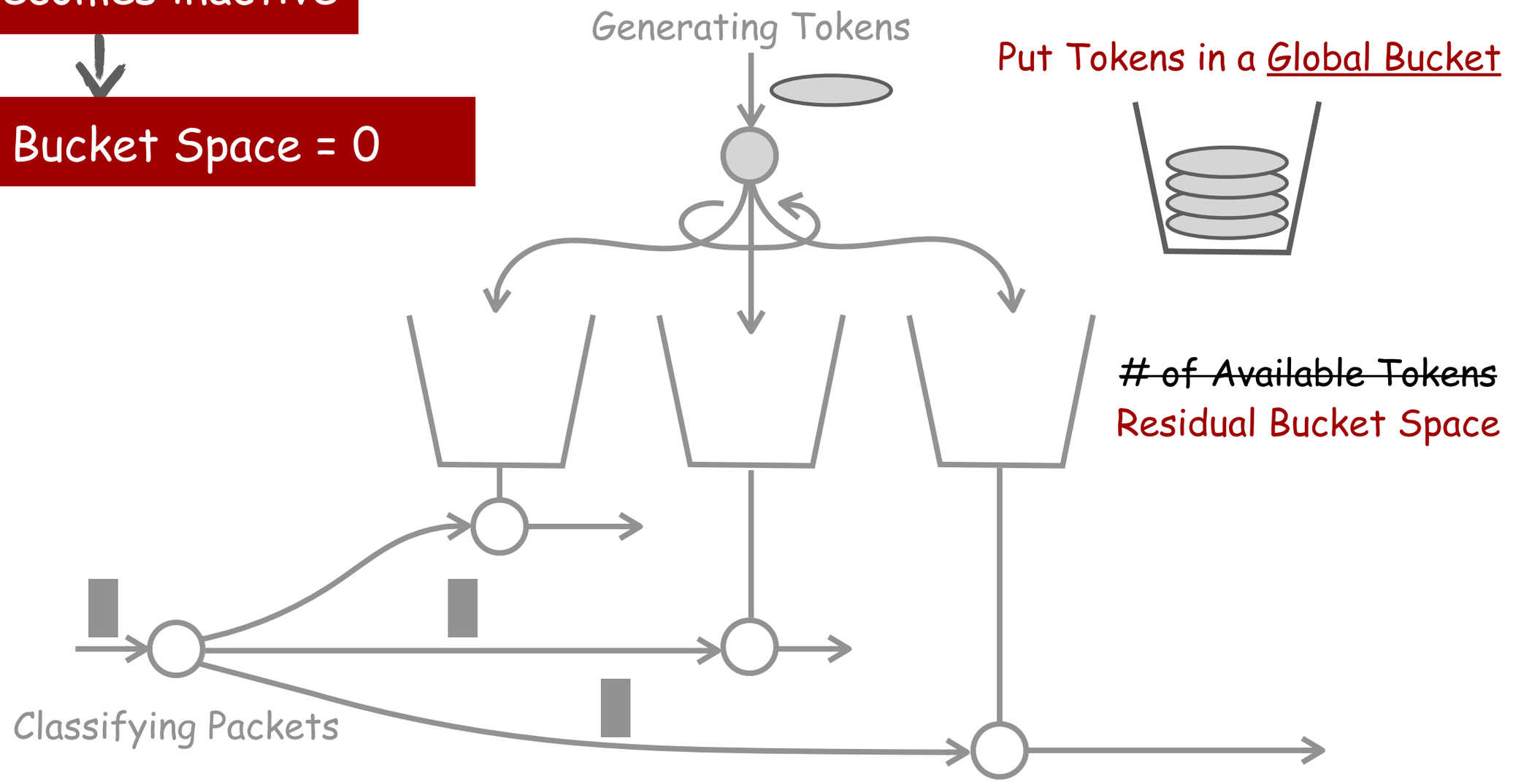
Design of FairPolicer

Address Challenges #1

Flow becomes inactive

↓

Residual Bucket Space = 0



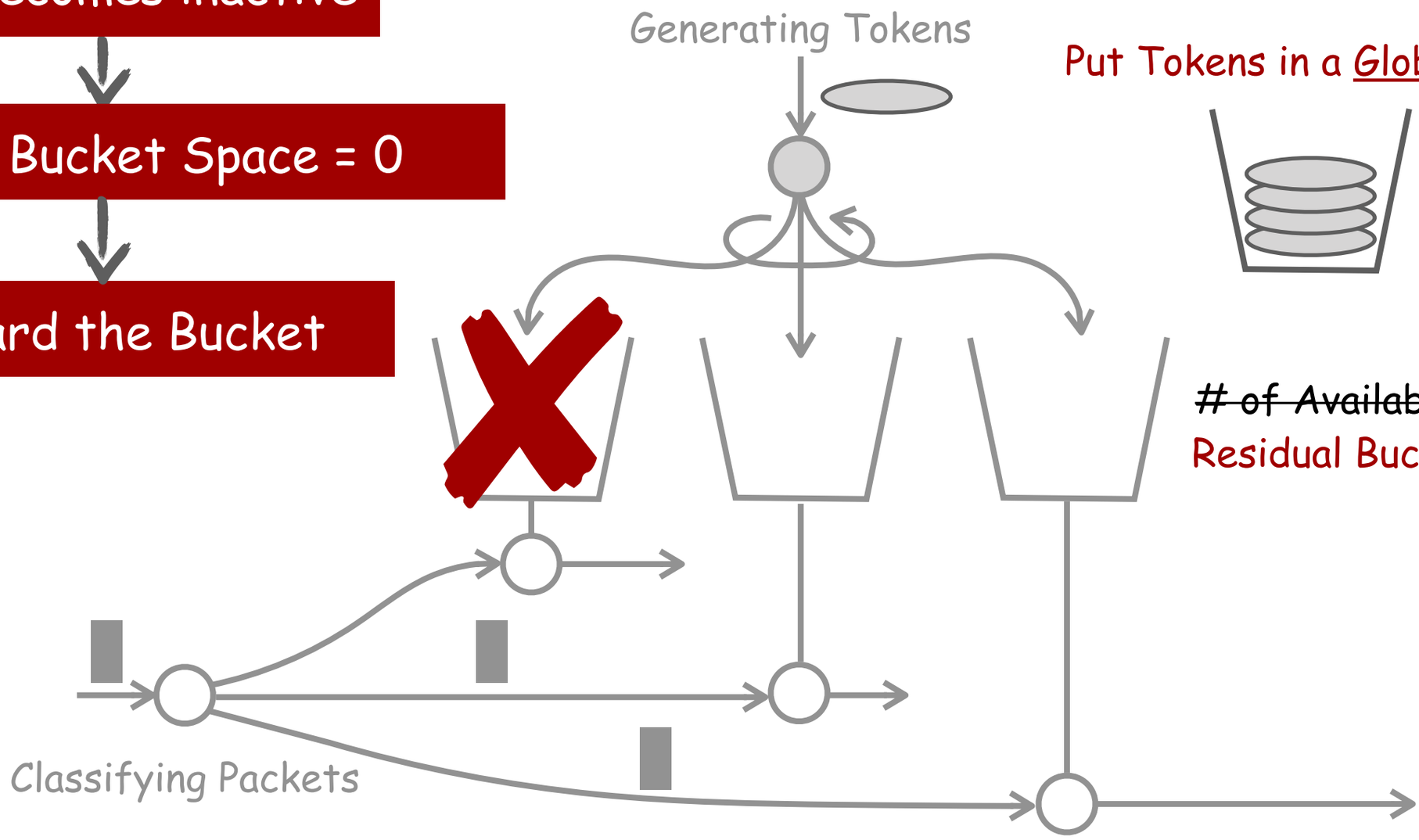
Design of FairPolicer

Address Challenges #1

Flow becomes inactive

Residual Bucket Space = 0

Discard the Bucket



Put Tokens in a Global Bucket

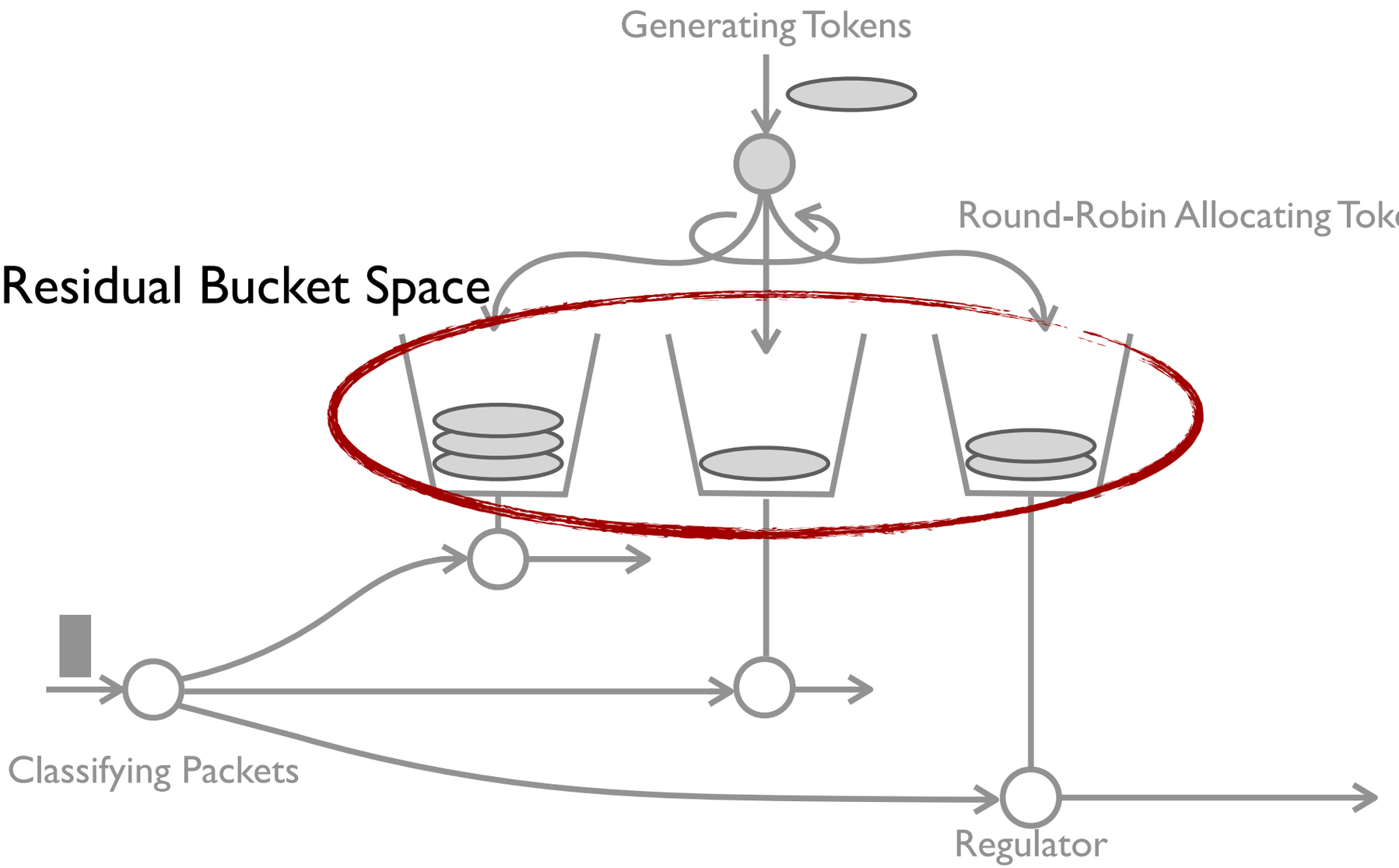
~~# of Available Tokens~~

Residual Bucket Space

Design of FairPolicer

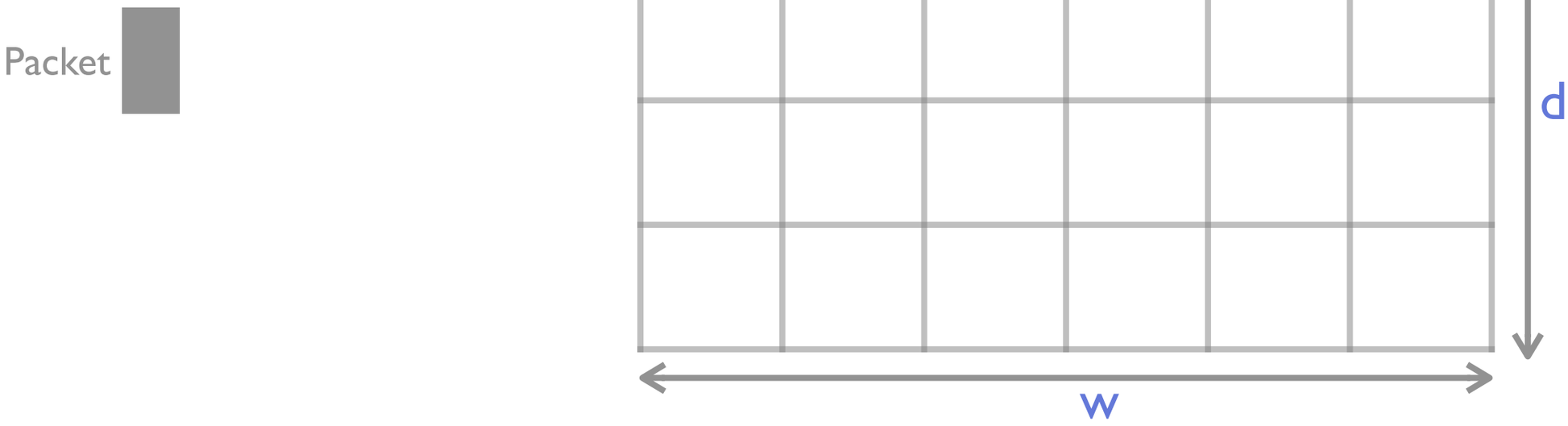
Challenges #2: Maintain Per-flow Data

Per-flow Residual Bucket Space



Design of FairPolicer

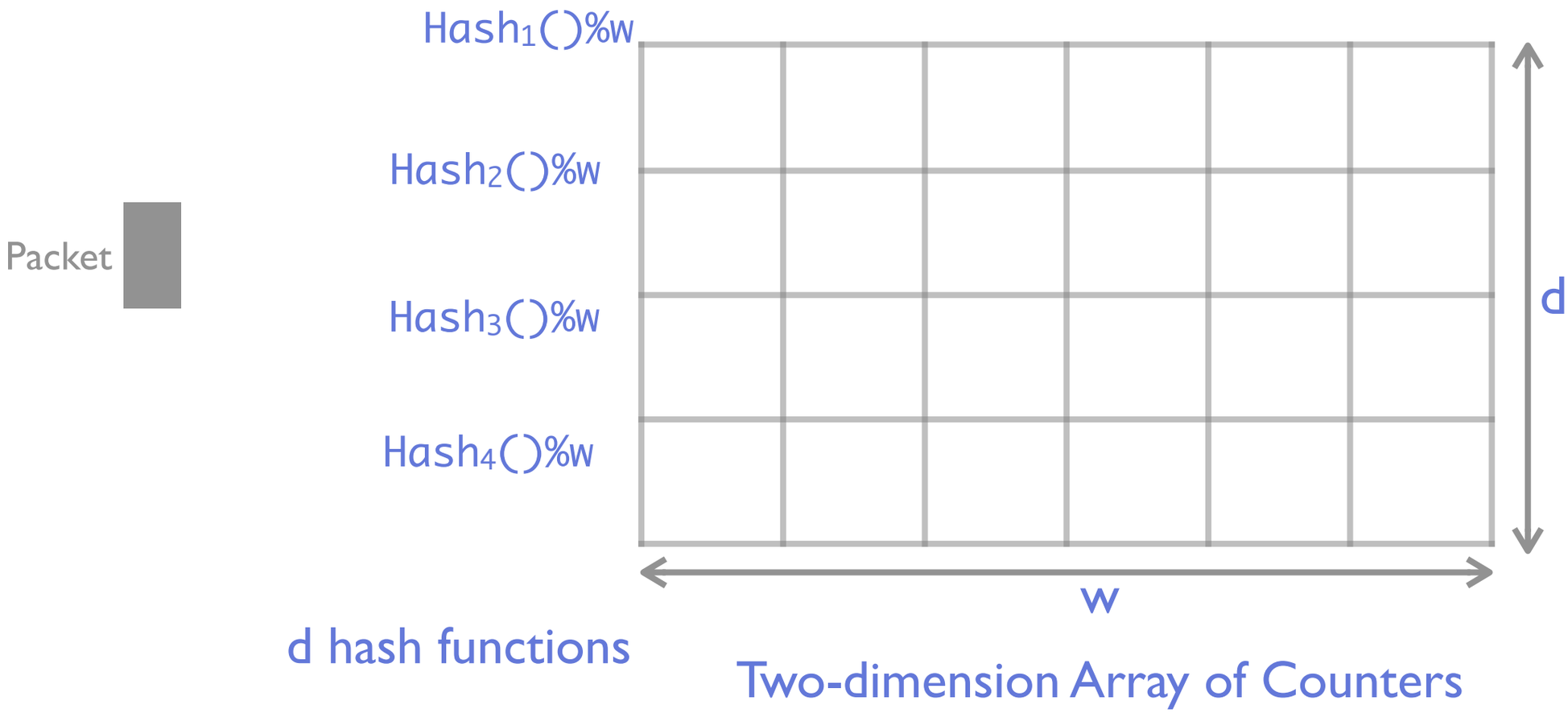
Address Challenges #2: Count-min Sketch



Two-dimension Array of Counters

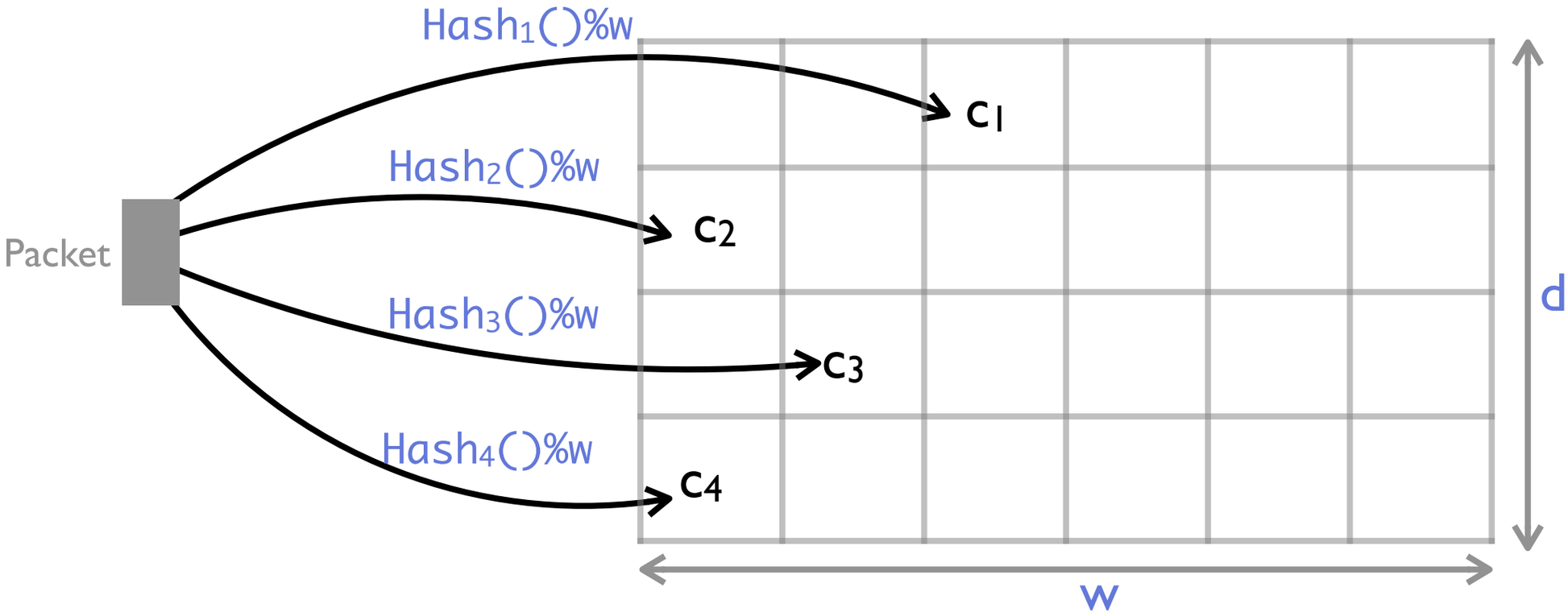
Design of FairPolicer

Address Challenges #2: Count-min Sketch



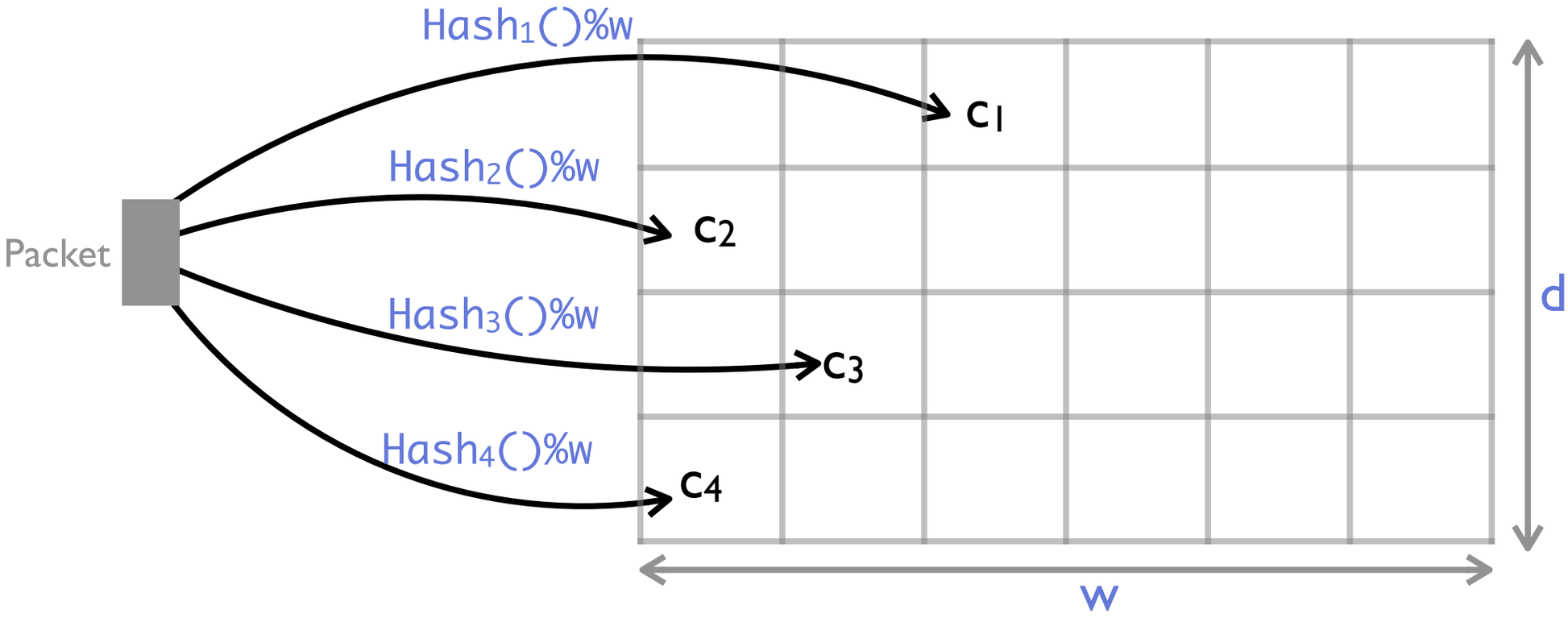
Design of FairPolicer

Address Challenges #2: Count-min Sketch



Design of FairPolicer

Address Challenges #2: Count-min Sketch

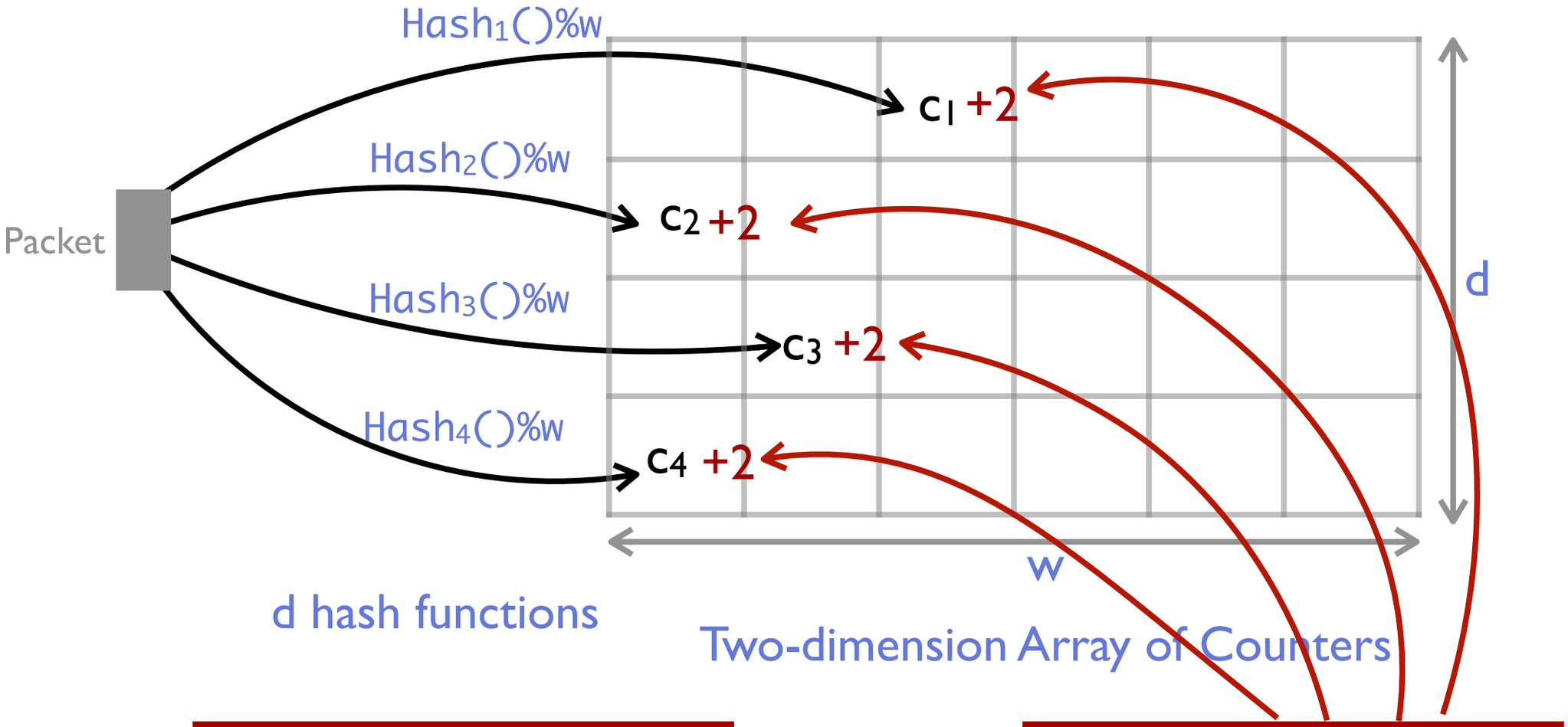


Update Per-flow Data

Query Per-flow Data

Design of FairPolicer

Address Challenges #2: Count-min Sketch

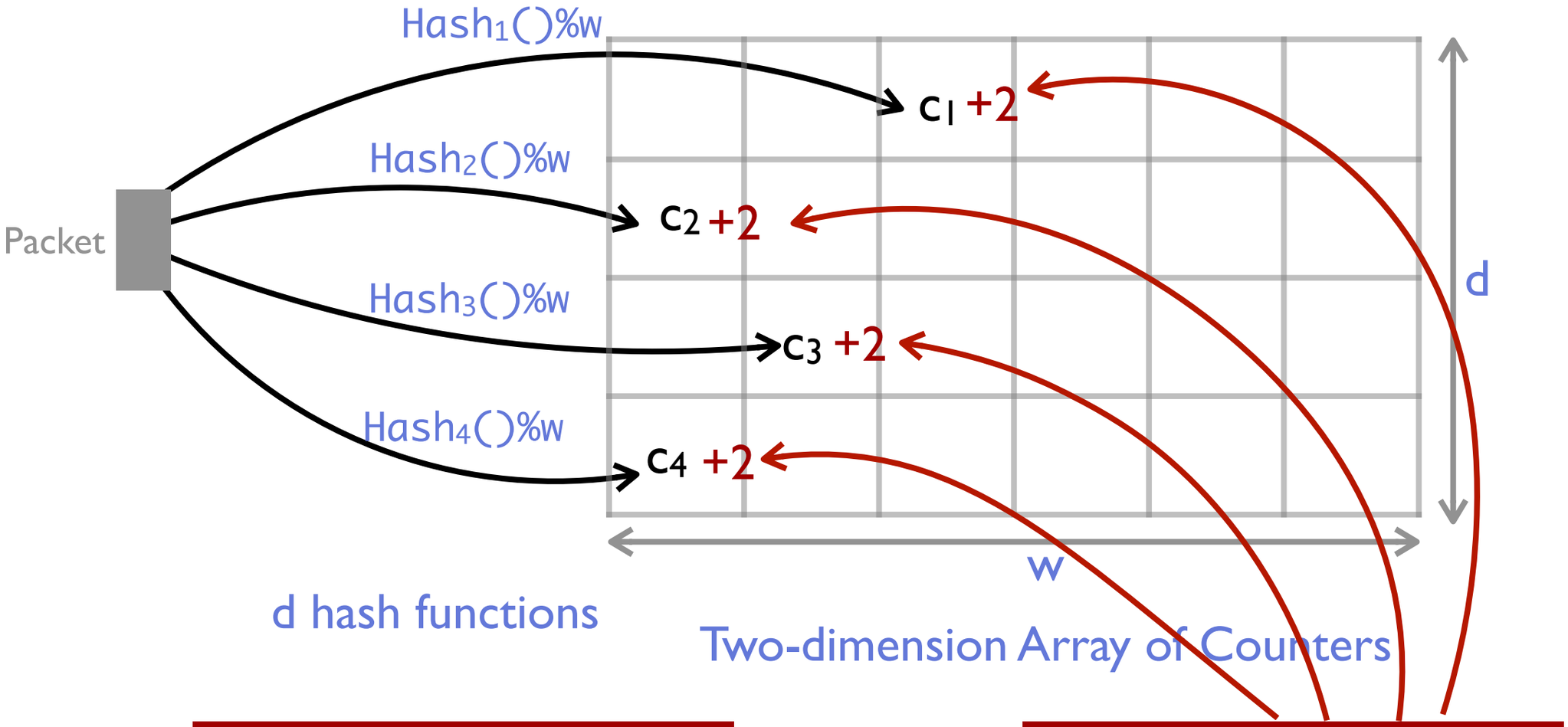


Update Per-flow Data → Update All Counters

Query Per-flow Data

Design of FairPolicer

Address Challenges #2: Count-min Sketch

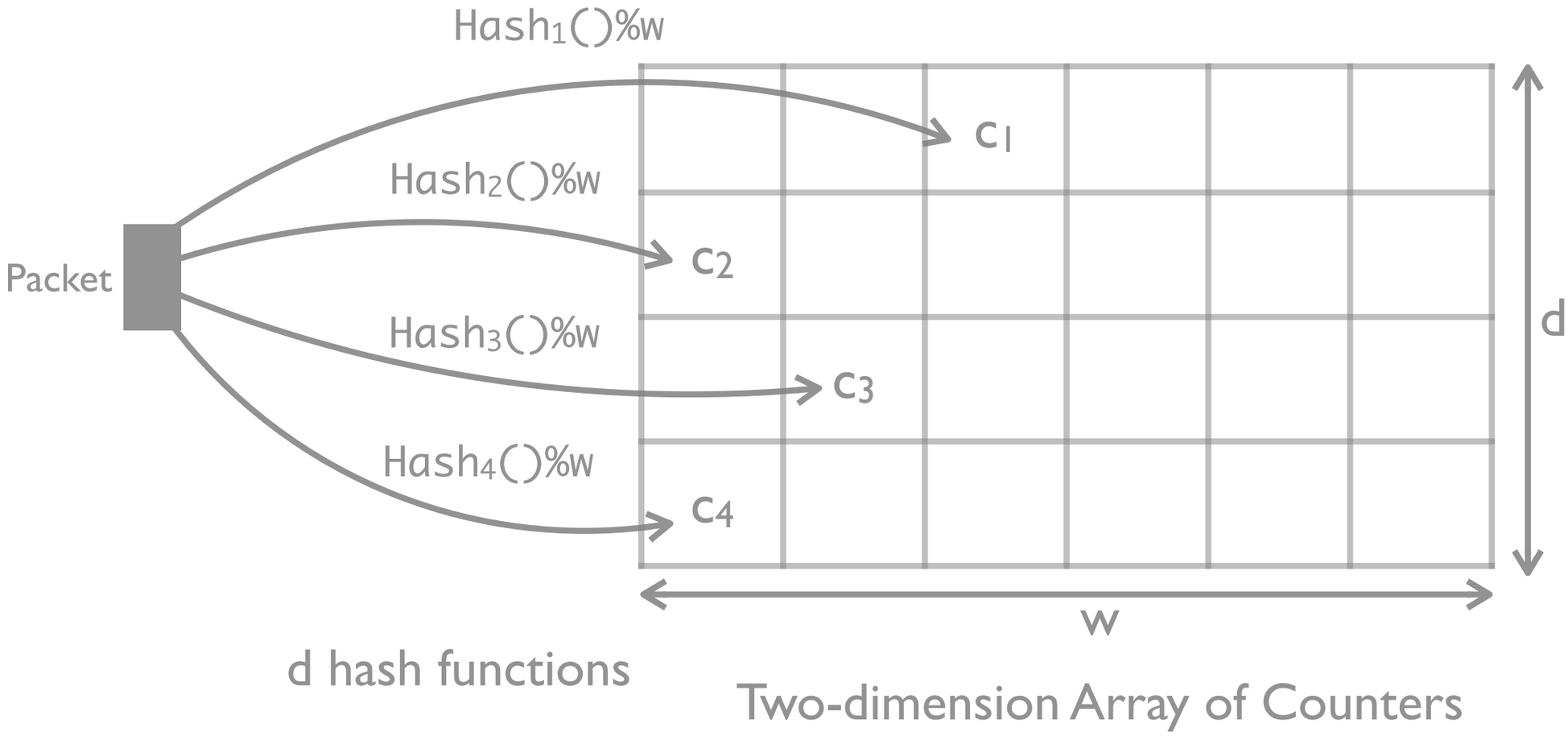


Update Per-flow Data → Update All Counters

Query Per-flow Data → Return the Minimum

Design of FairPolicer

Address Challenges #2: Count-min Sketch



Accuracy of Count-min Sketch with lots of flows?

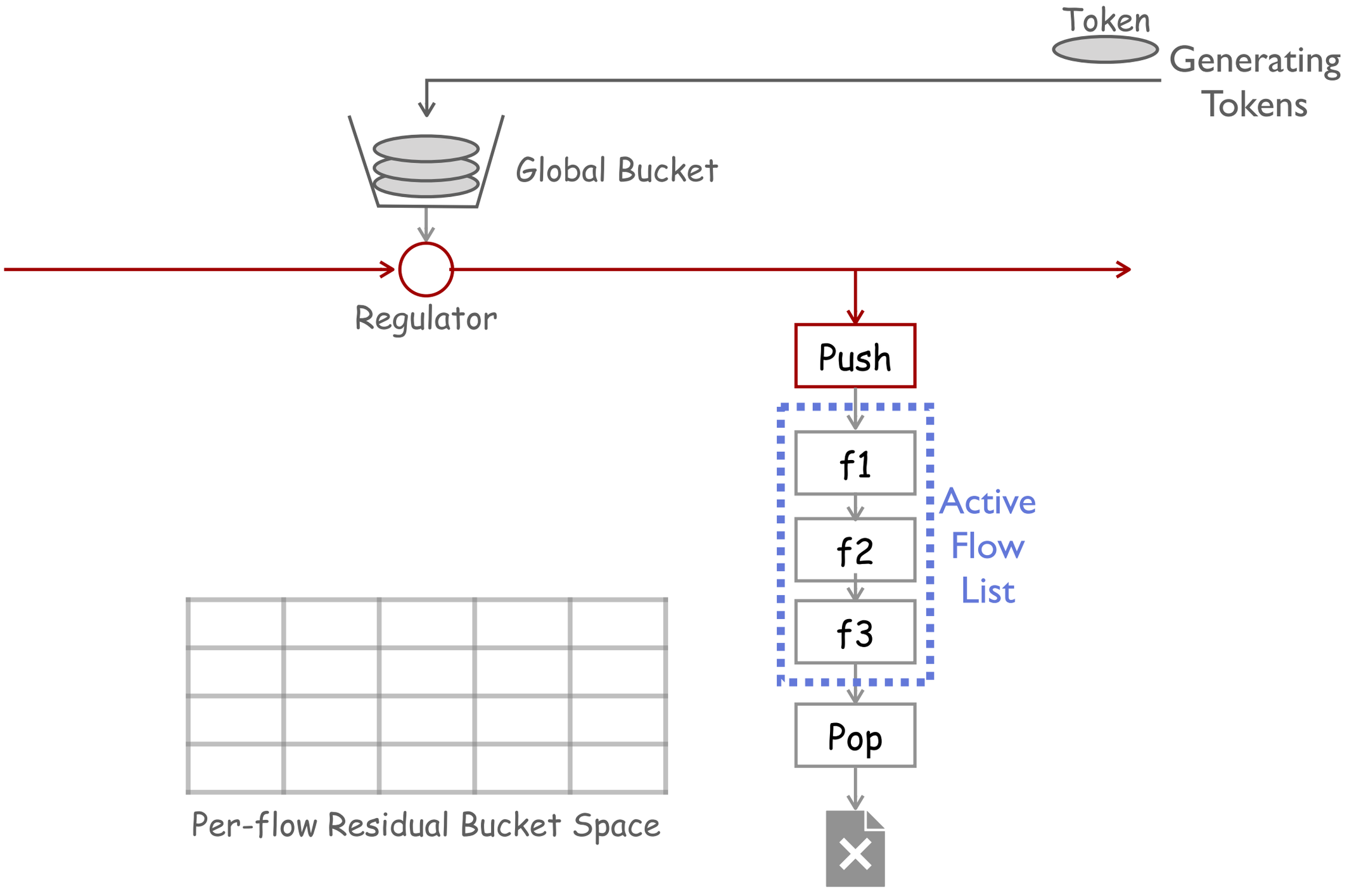
Accuracy of Count-Min Sketch

- The estimation error is bounded

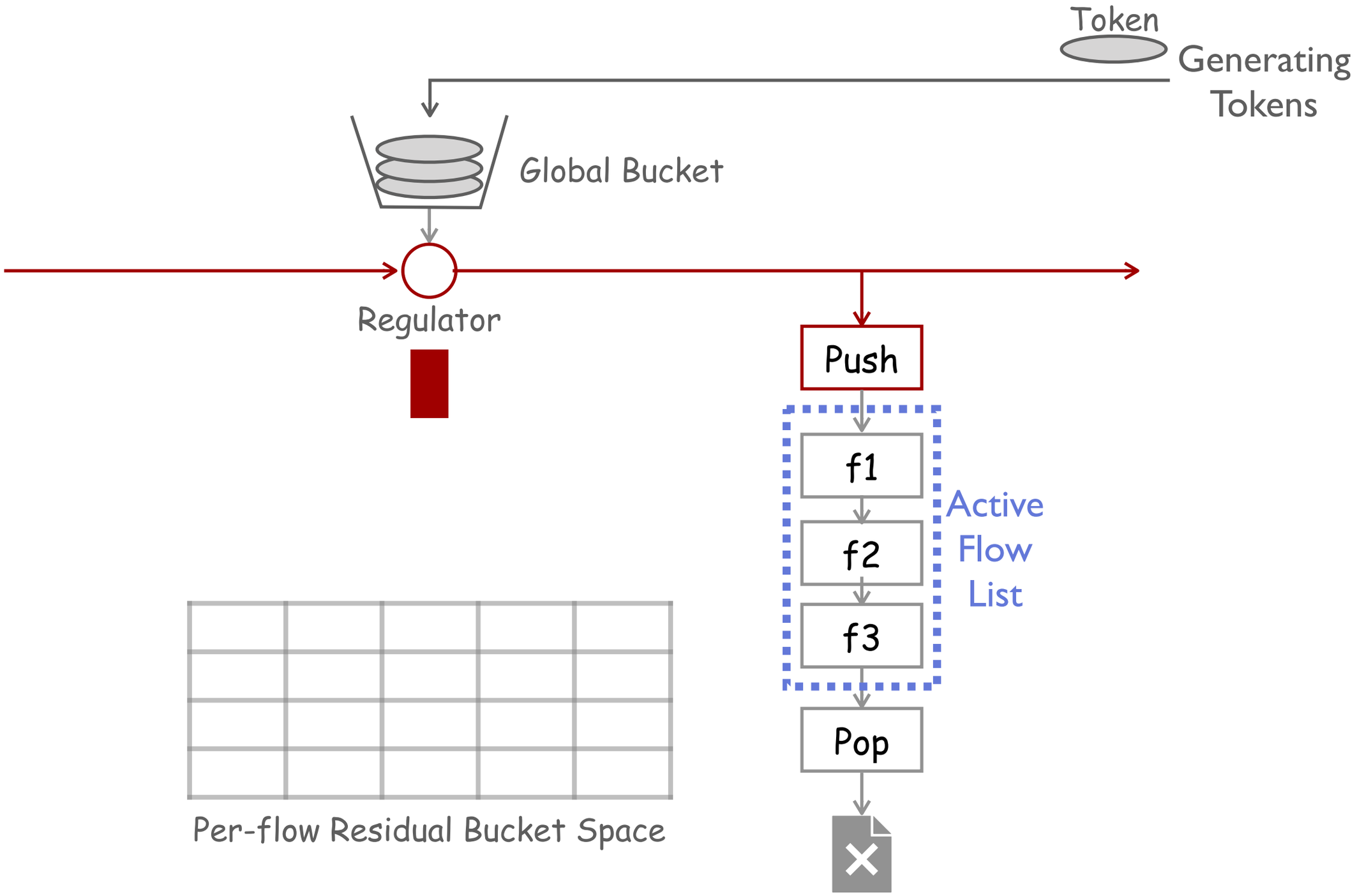
Theorem 1. *The estimation error is within ϵB with a probability of $1-\delta$, where $\epsilon = e/w$ and $\delta = 1/e^d$.*

- Employing a large Count-Min Sketch is not exorbitant
  - A counter is no larger than B (bucket size)
  - A counter only needs $\Theta(log_2 B)$ bits
  - E.g., 100KB bucket, 4×4096 sketch
    - 2B memory for a counter (40B granularity)
    - 32KB memory for the total sketch
    - Commercial switching chip: MBs memory

FairPolicer: On Packet Arrival



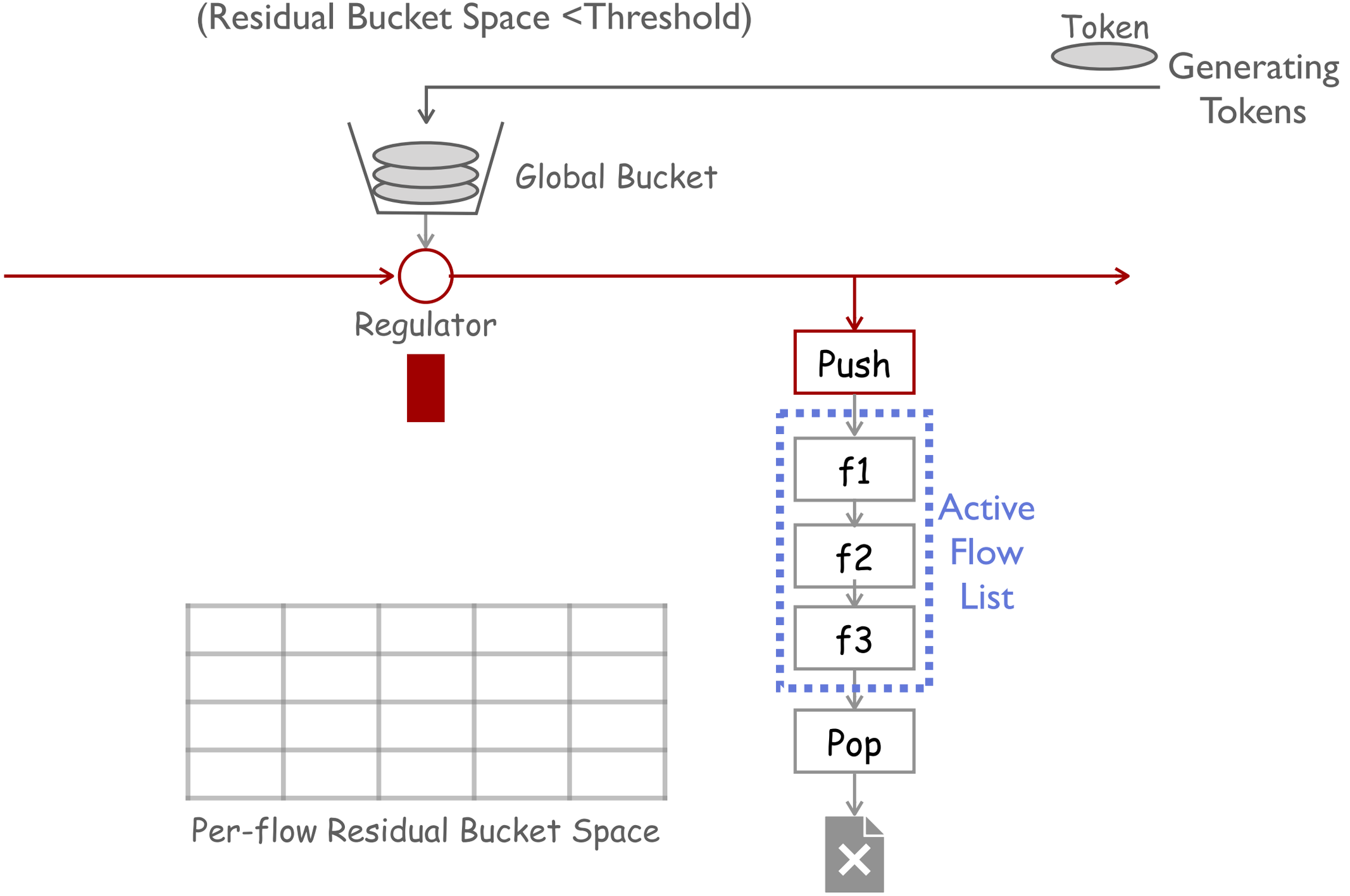
FairPolicer: On Packet Arrival



FairPolicer: On Packet Arrival

Case 1: Enough Tokens in the Bucket

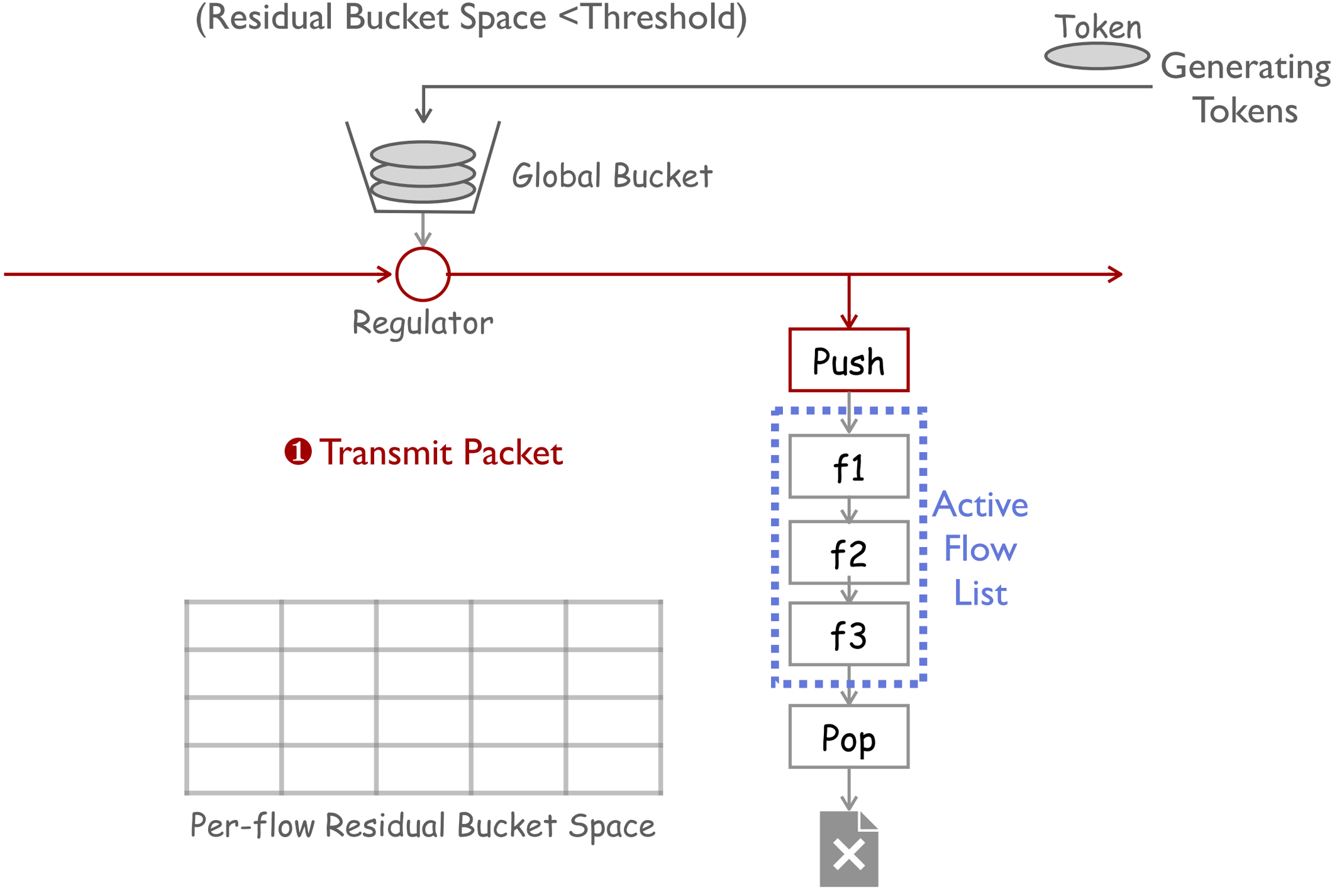
(Residual Bucket Space <Threshold)



FairPolicer: On Packet Arrival

Case 1: Enough Tokens in the Bucket

(Residual Bucket Space <Threshold)

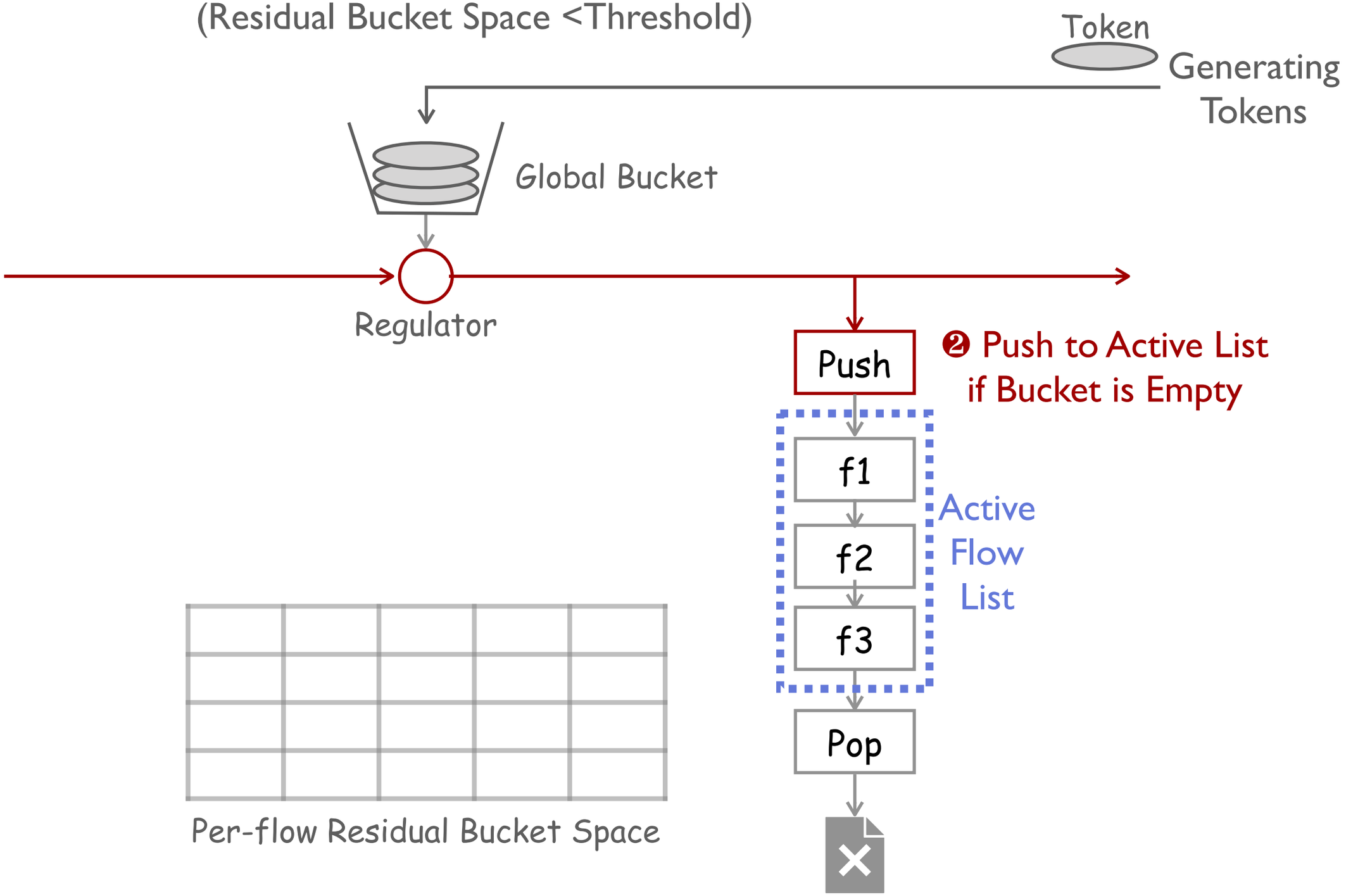


❶ Transmit Packet

FairPolicer: On Packet Arrival

Case 1: Enough Tokens in the Bucket

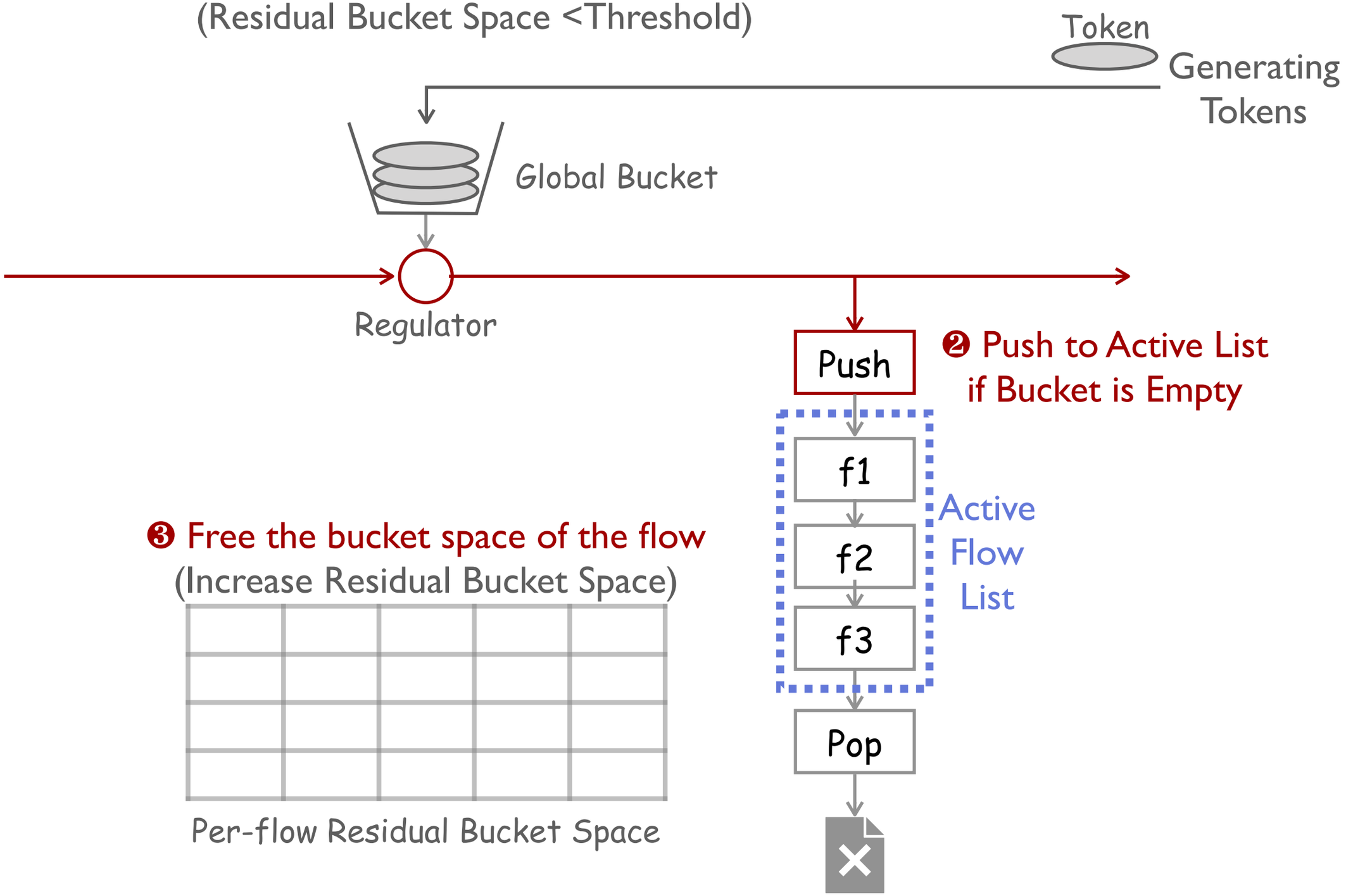
(Residual Bucket Space <Threshold)



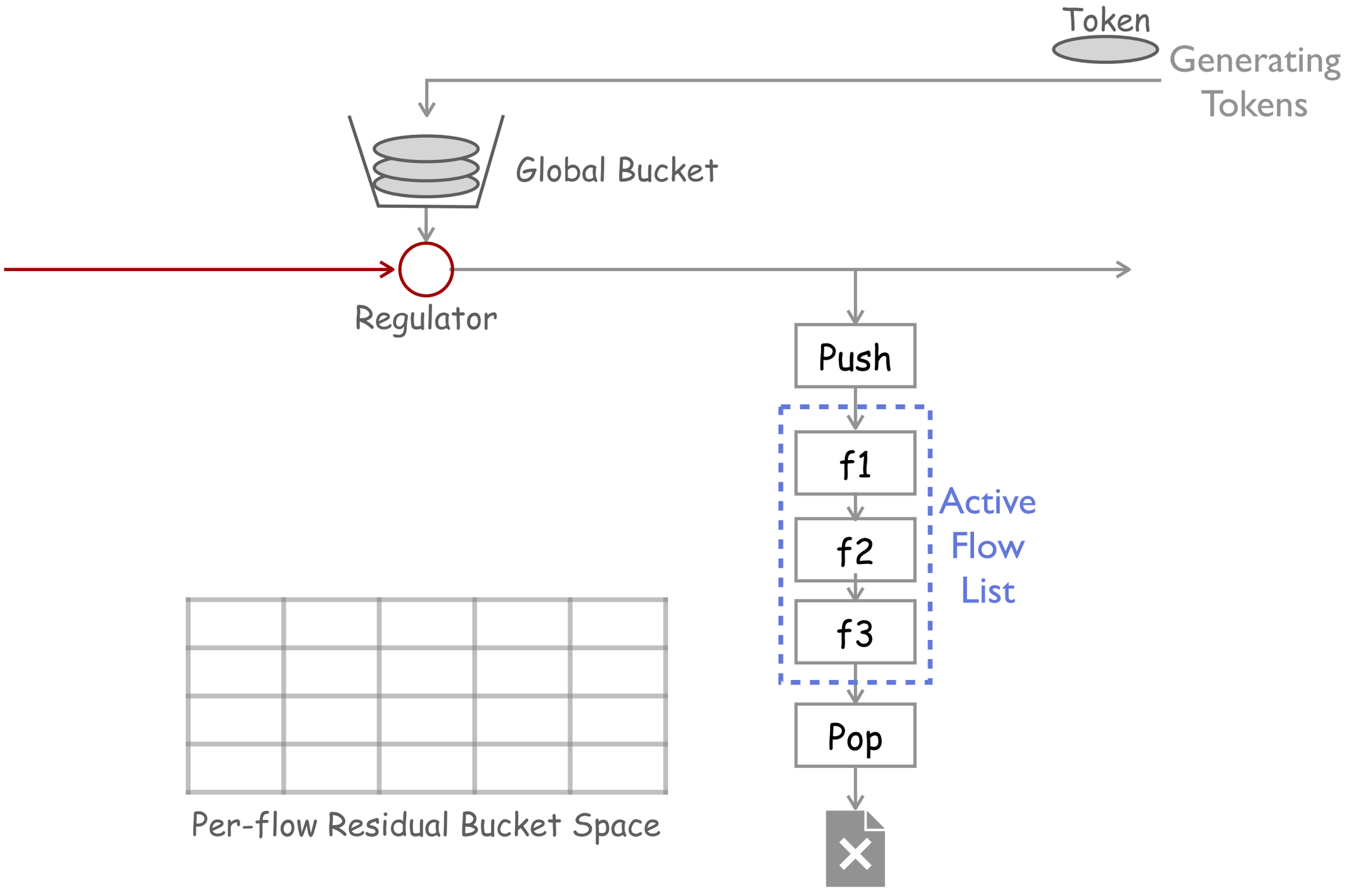
FairPolicer: On Packet Arrival

Case 1: Enough Tokens in the Bucket

(Residual Bucket Space <Threshold)



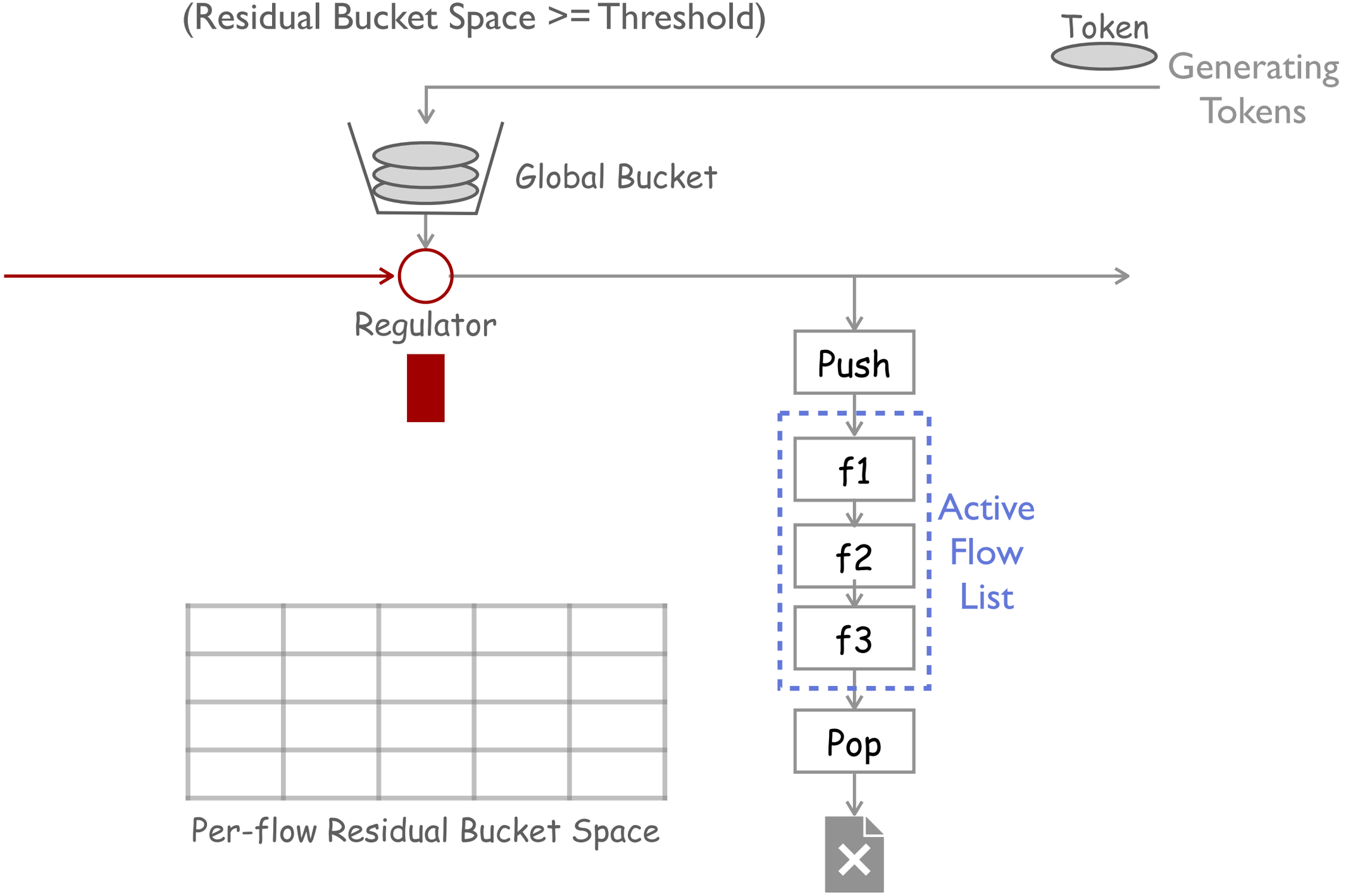
FairPolicer: On Packet Arrival



FairPolicer: On Packet Arrival

Case 2: Not Enough Tokens in the Bucket

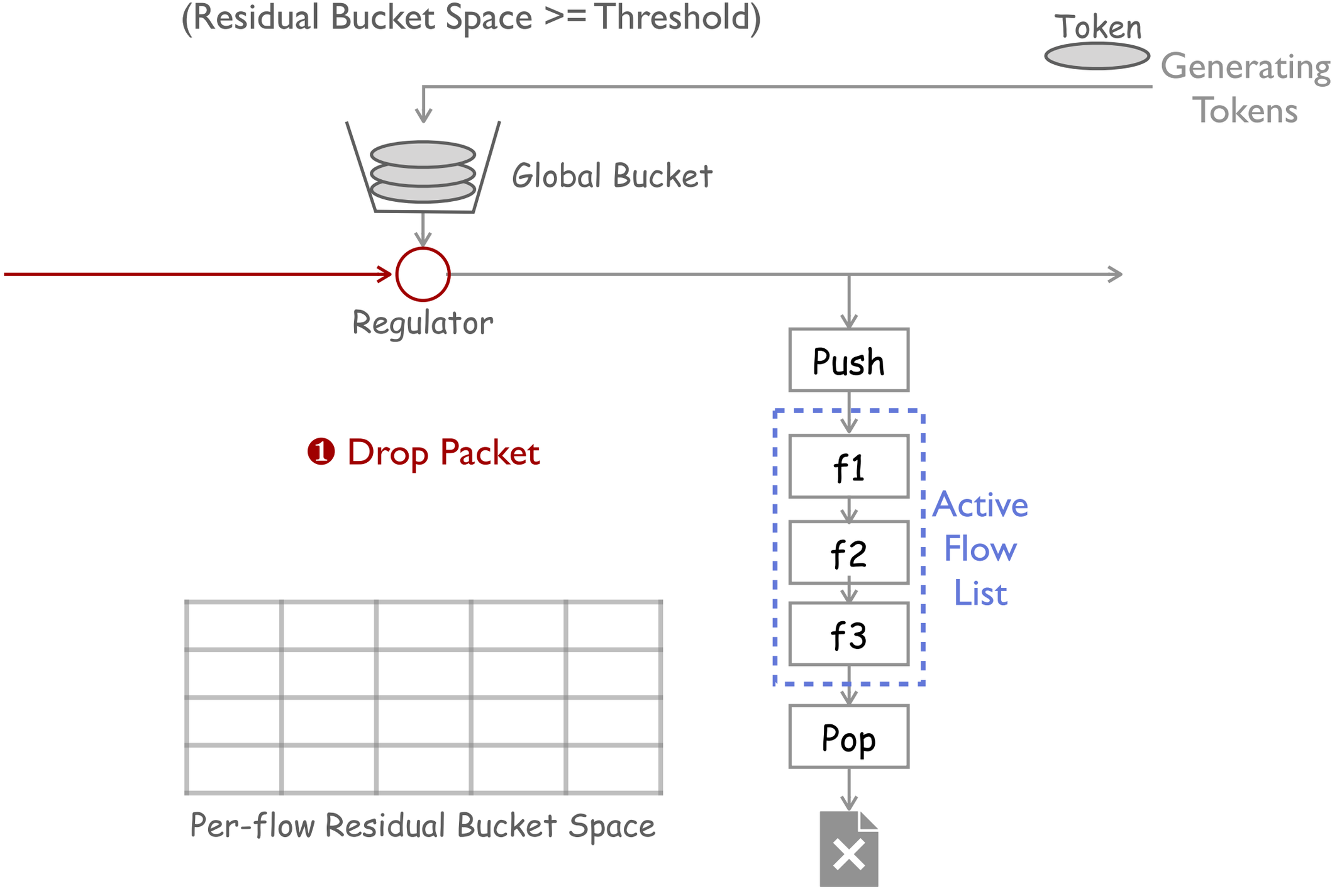
(Residual Bucket Space >= Threshold)



FairPolicer: On Packet Arrival

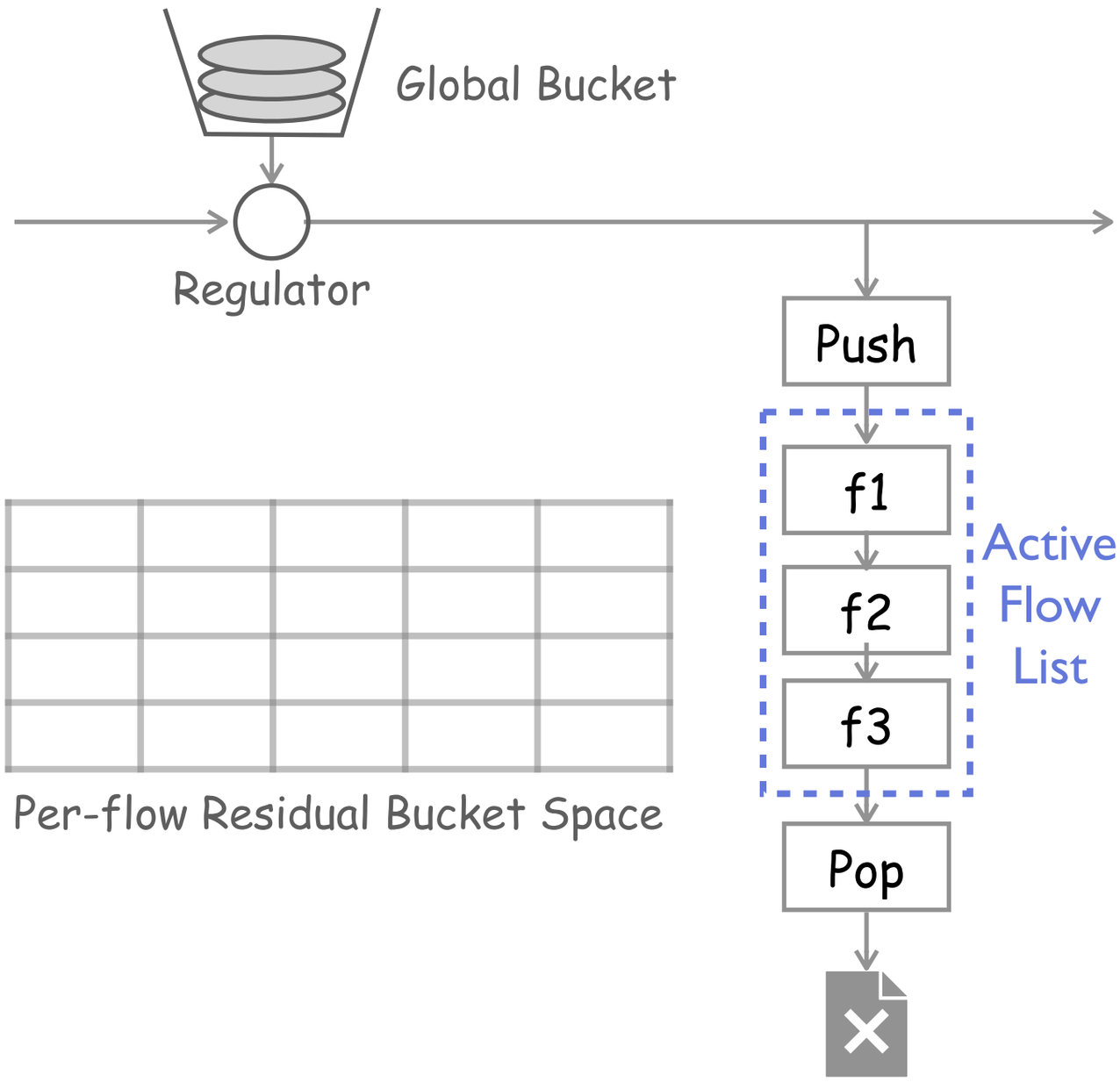
Case 2: Not Enough Tokens in the Bucket

(Residual Bucket Space >= Threshold)

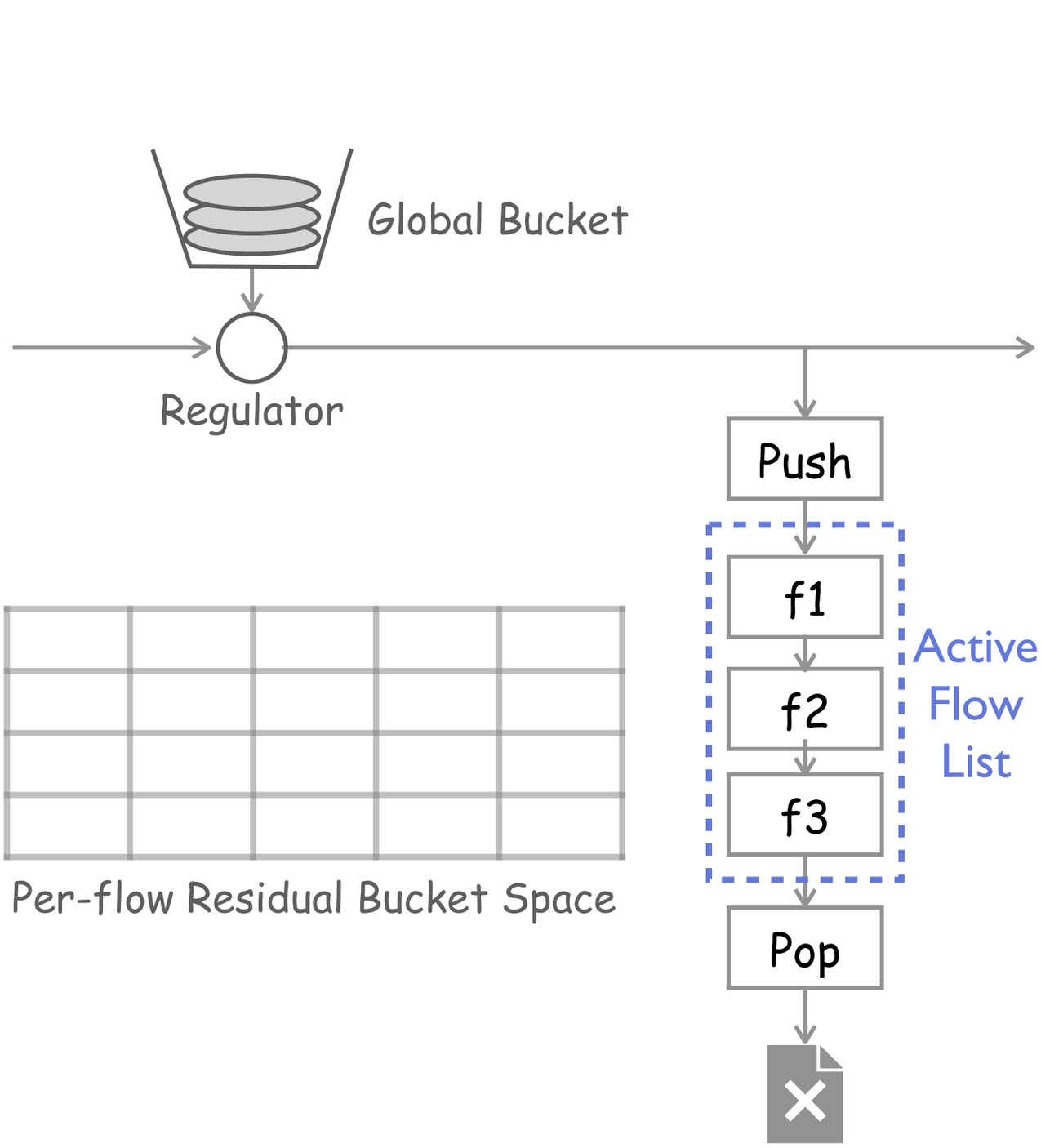


❶ Drop Packet

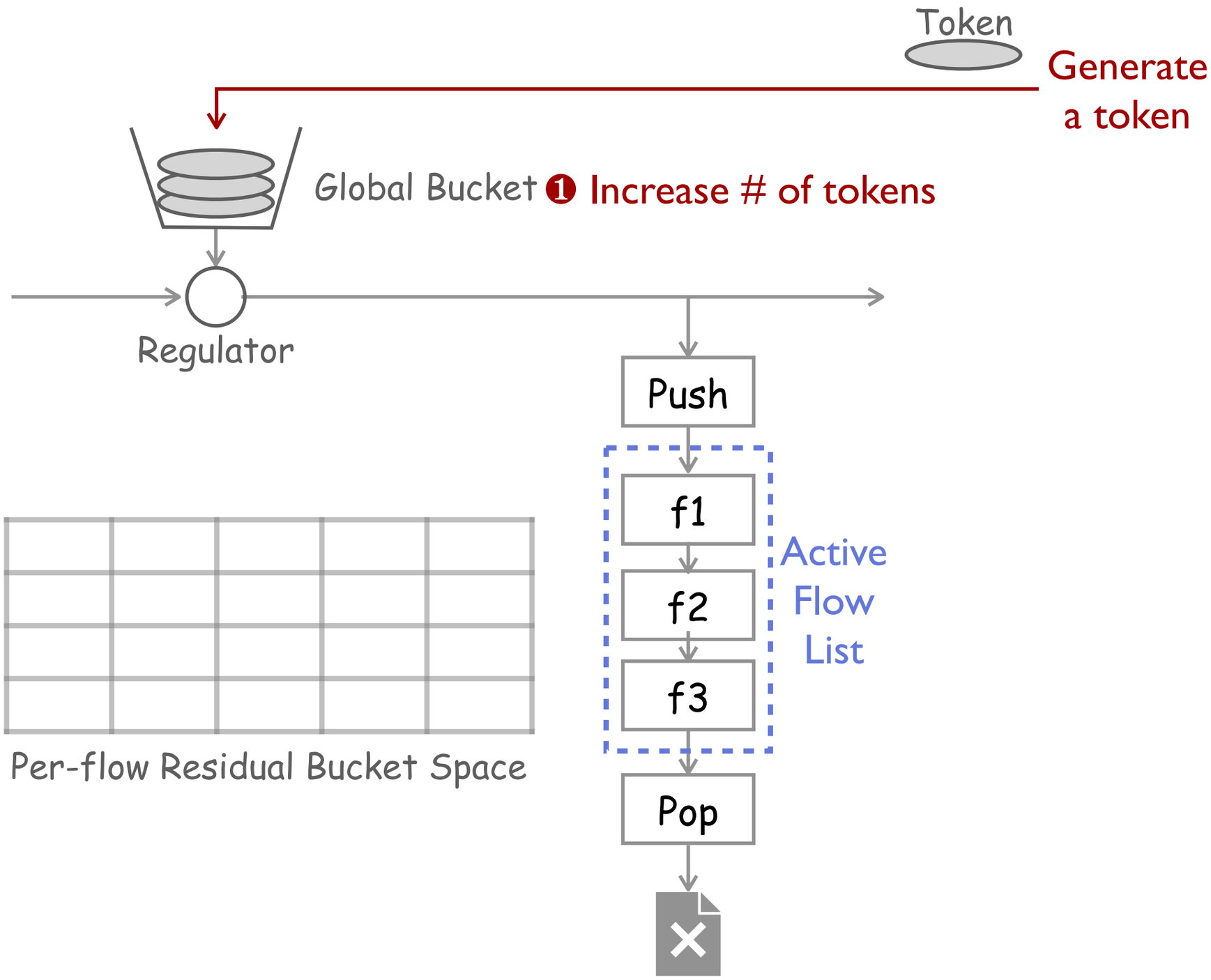
FairPolicer: On Token Generation



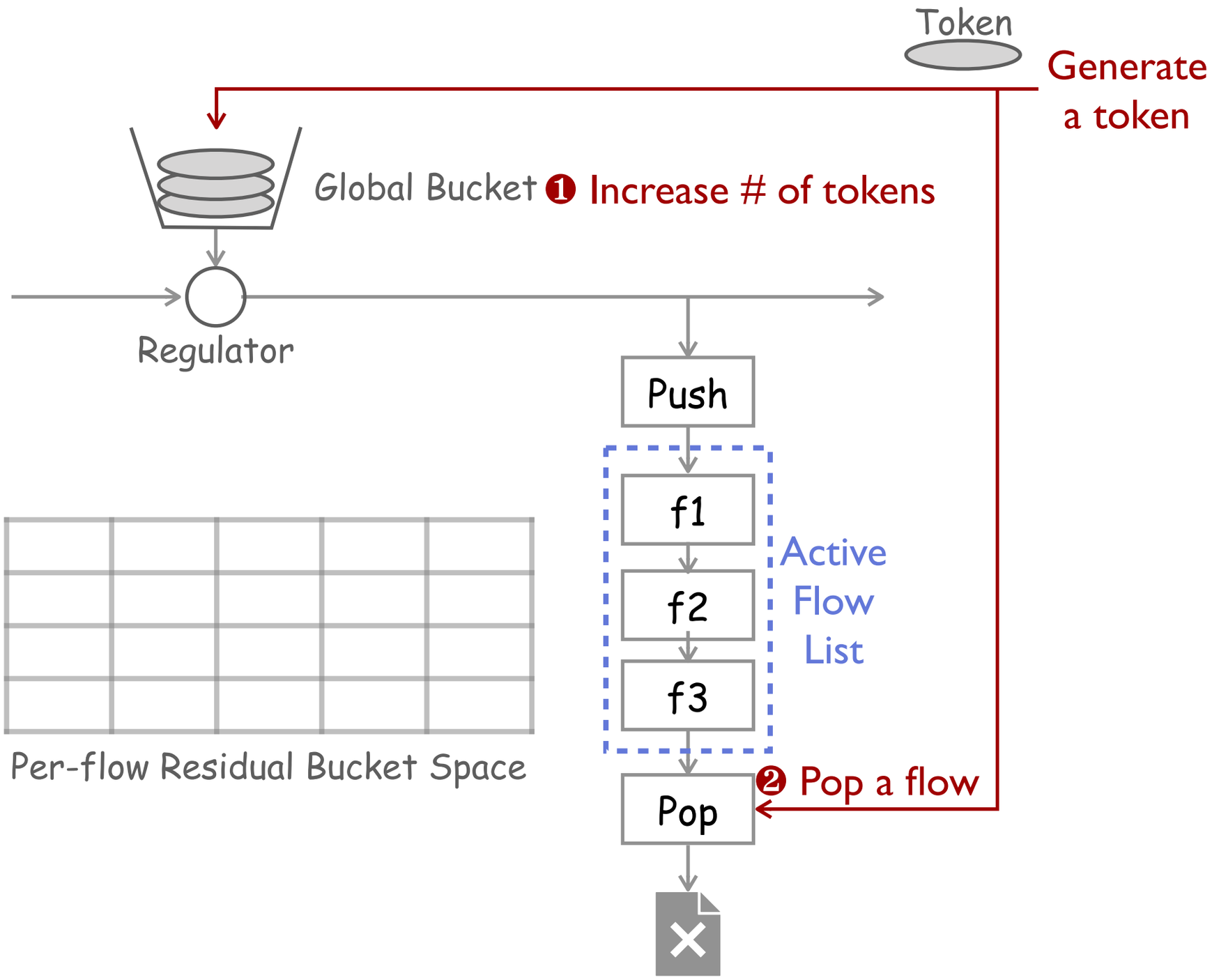
FairPolicer: On Token Generation



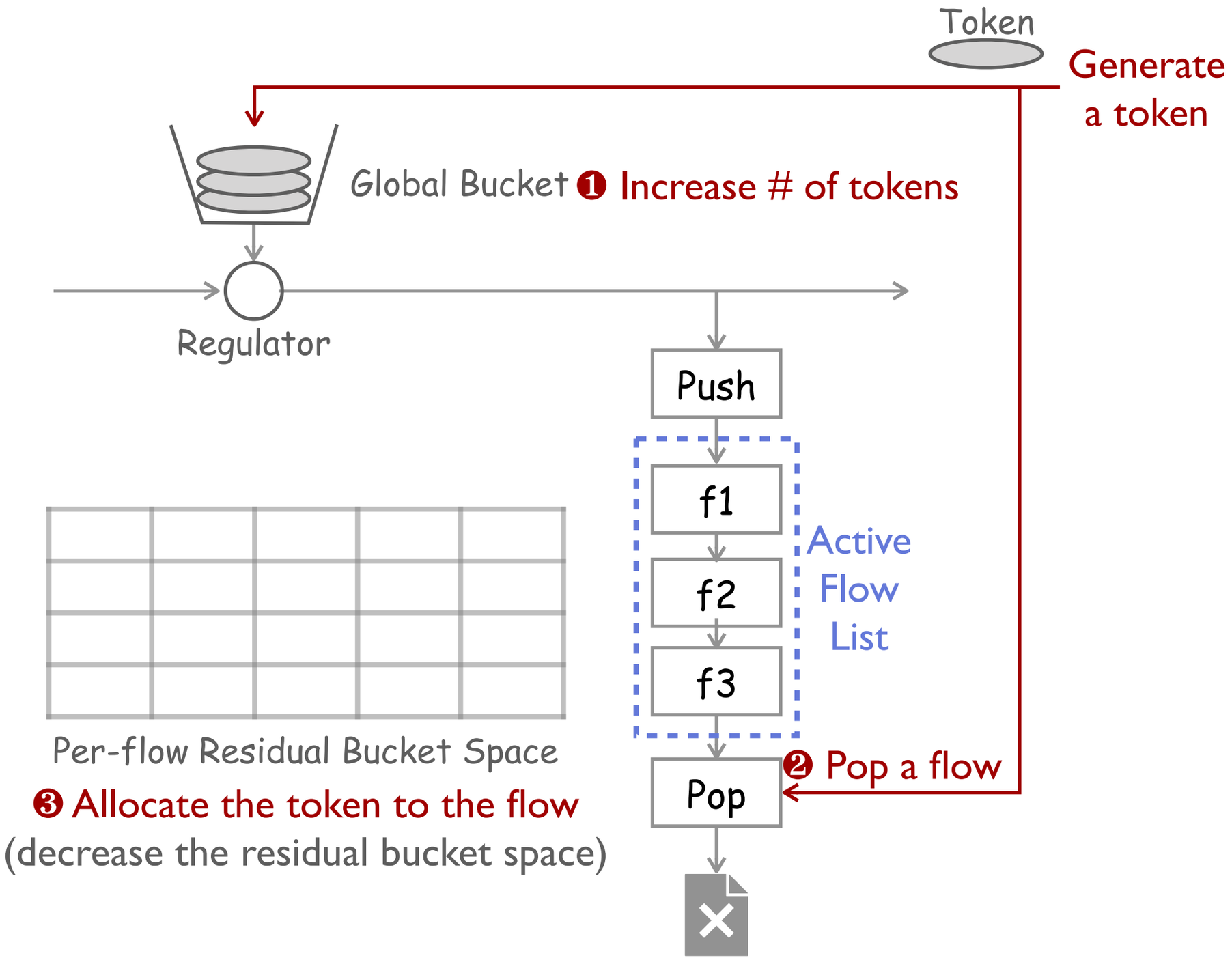
FairPolicer: On Token Generation



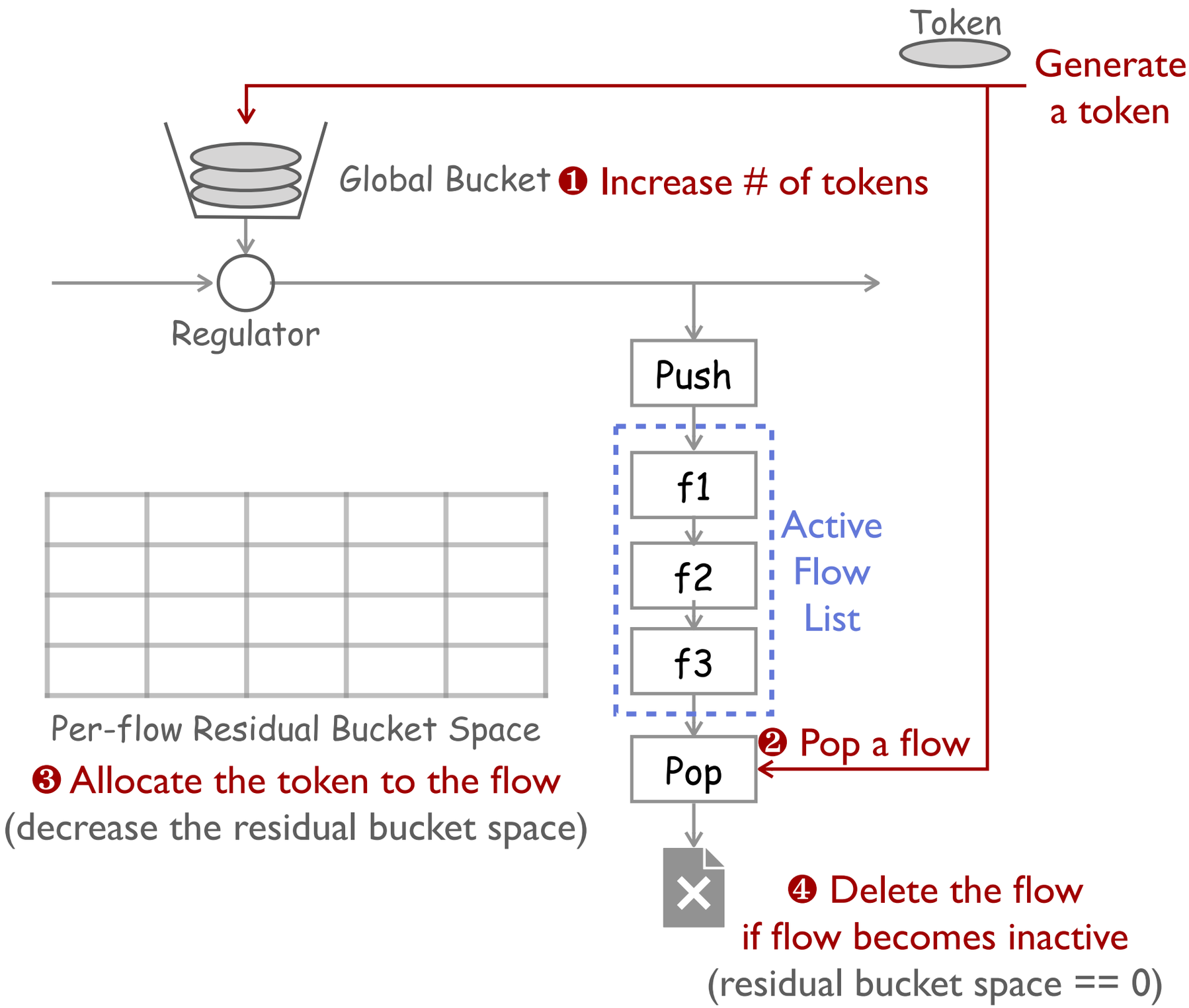
FairPolicer: On Token Generation



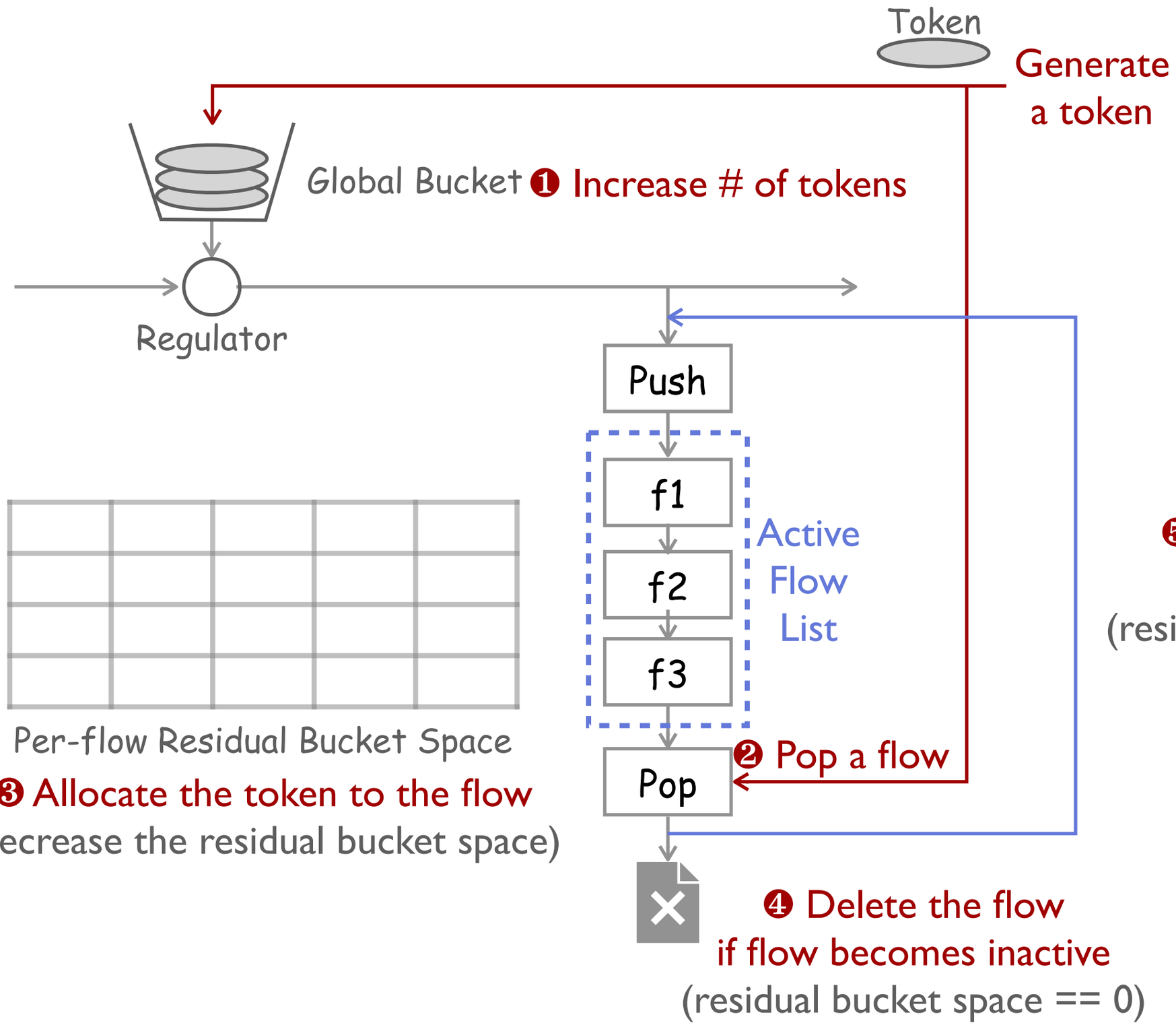
FairPolicer: On Token Generation



FairPolicer: On Token Generation



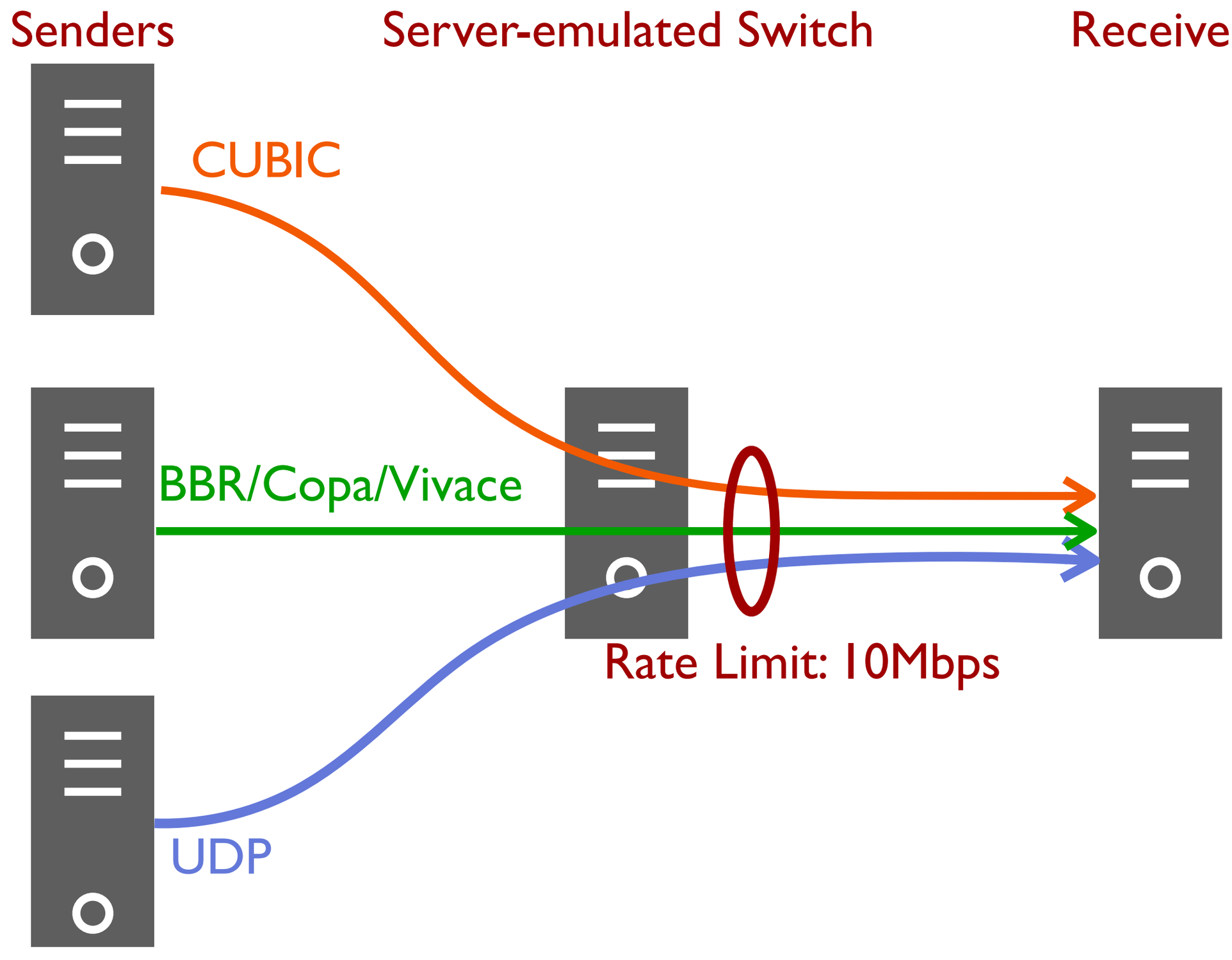
FairPolicer: On Token Generation



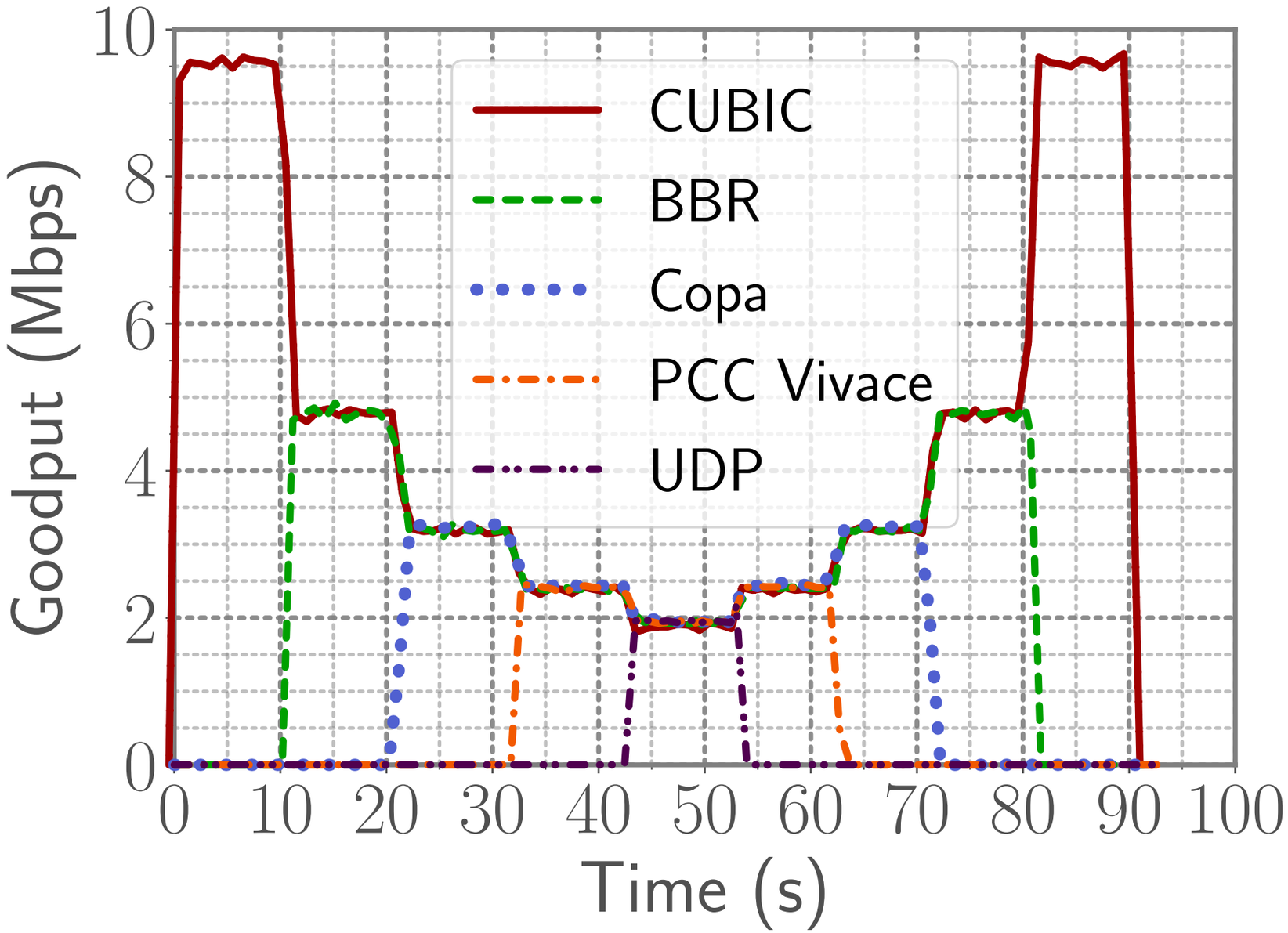
Design of FairPolicer

- More details in the paper
  - Dynamically adjust the per-flow bucket space
  - Parameter settings

Evaluation — Testbed Setup



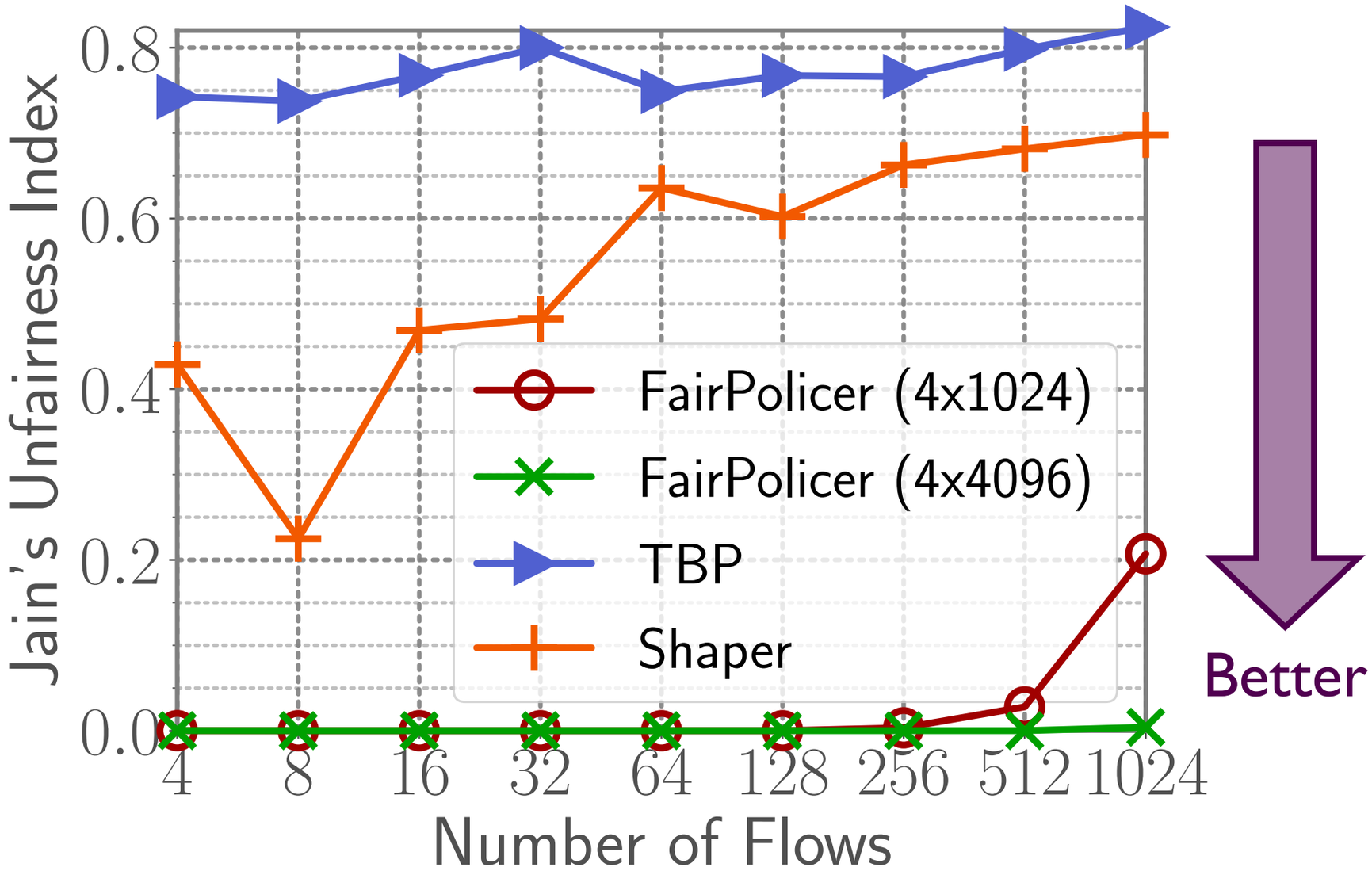
Evaluation — Fairness



Fairness and Convergence

Ensure fairness regardless of CC algorithms

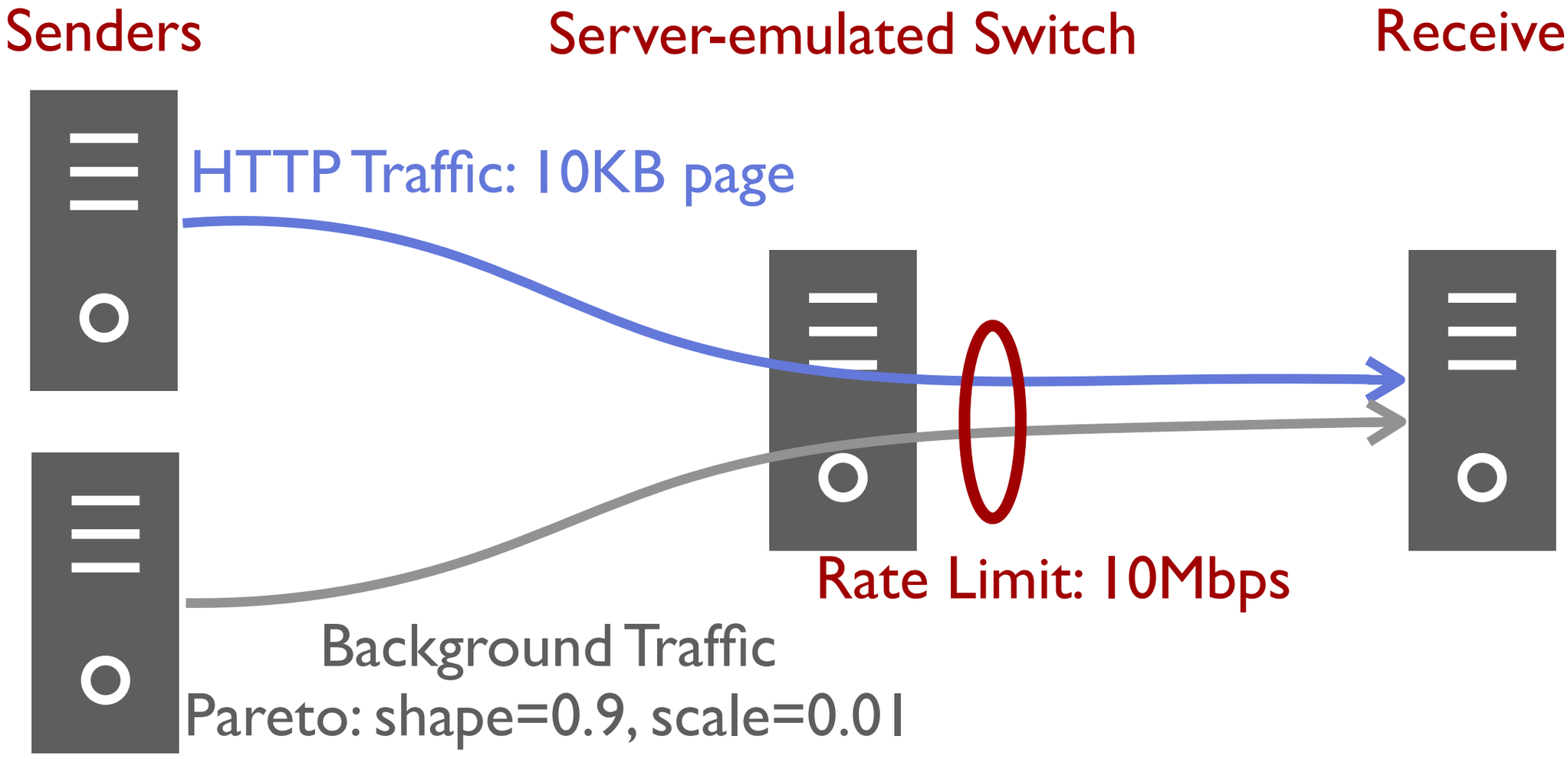
Evaluation — Fairness



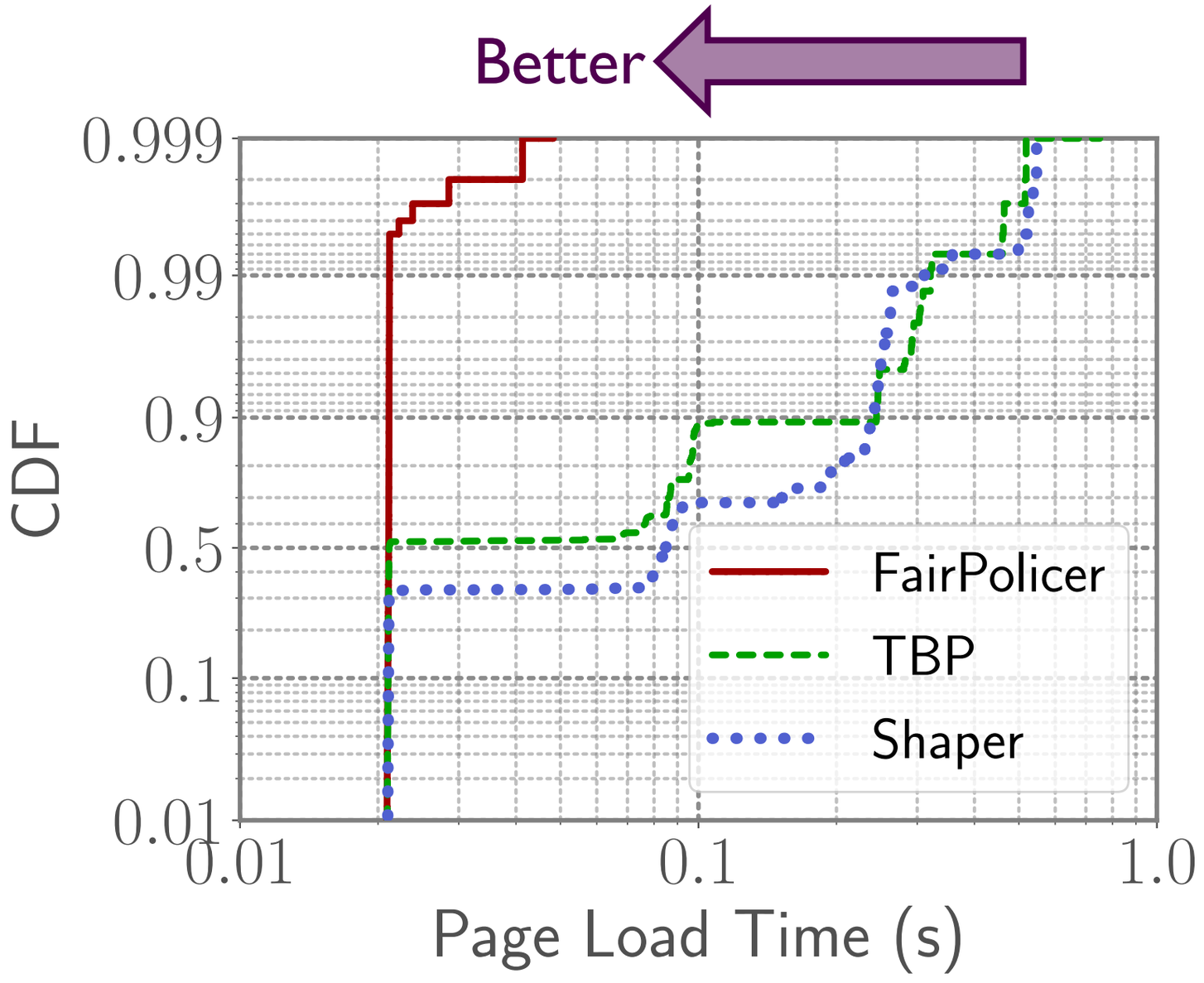
Unfairness Index

Unfairness Index is within 0.004 with 4x4096 sketch size (32KB memory)

Evaluation — Latency



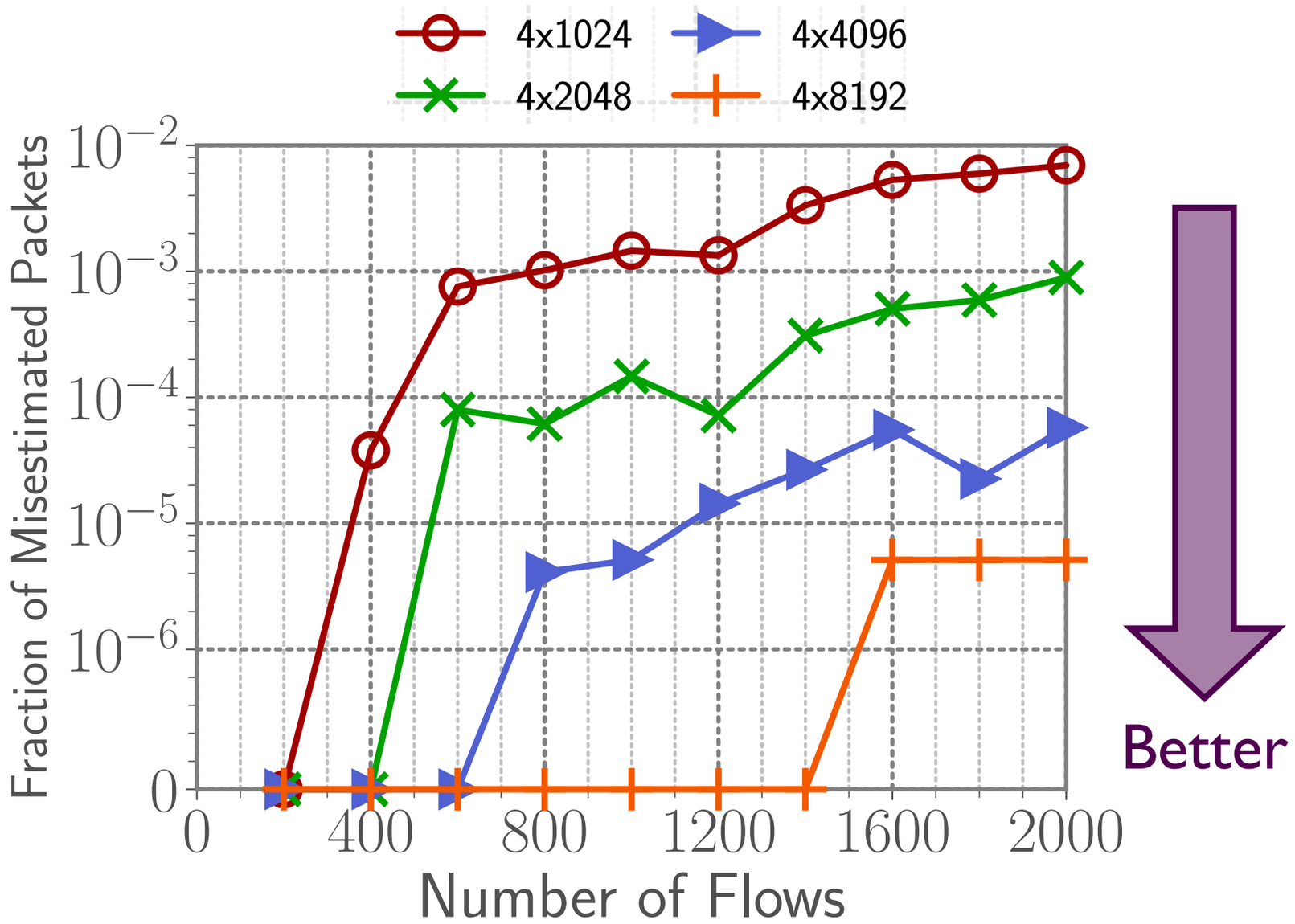
Evaluation — Latency



Web Page Load Time

Reduce the avg. page load time by 14.0x

Evaluation — Accuracy of Count-min Sketch



Accuracy

Scale to 2K flows with a sketch size of 4x4096 (32KB memory)

Conclusion

- Discover and validate the unfairness problem
- Propose FairPolicer that can fairly allocate bandwidth regardless of CC algorithms
- Prototype and evaluate FairPolicer in a testbed

Thank you!

Q&A

Source code: https://github.com/ants-xjtu/fairpolicer