Linux Traffic Control Classifier-Action Subsystem Architecture

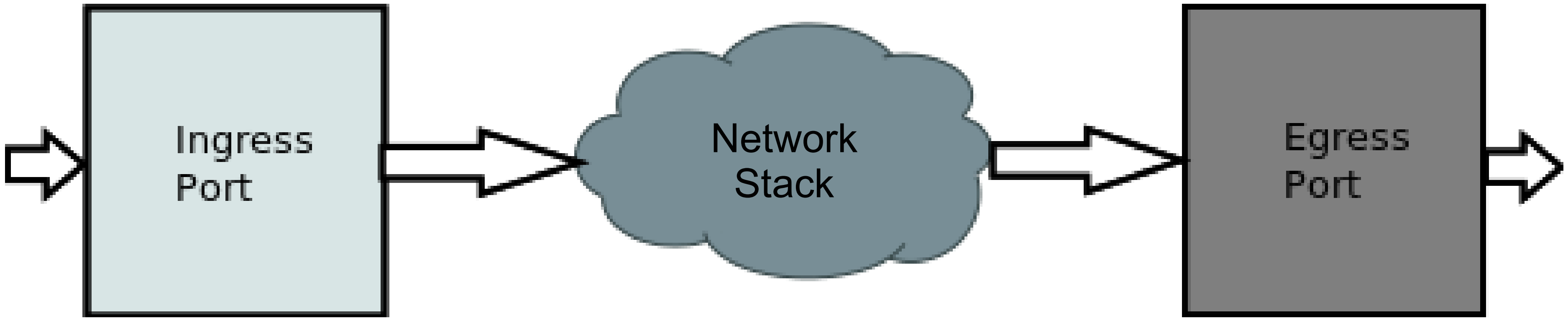
Jamal Hadi Salim

Netdev 0.1, Ottawa, On

Motivation

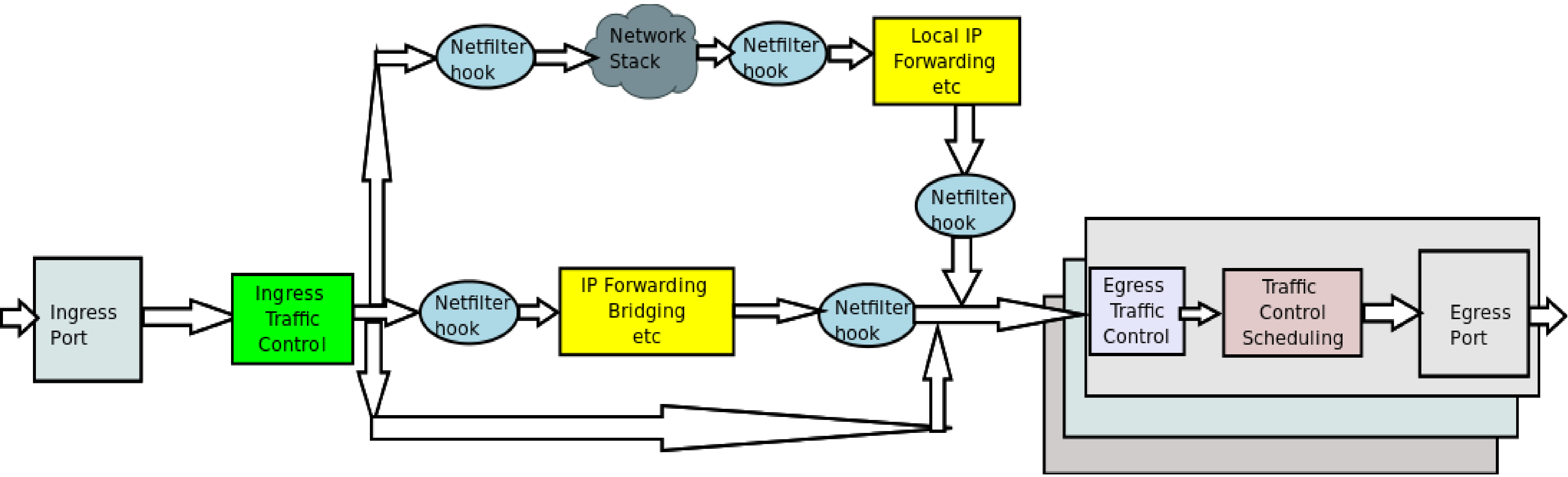
- Finally Document
- Hopefully have people use and build on top (as opposed to re-invent)

Life Starts With A Port...



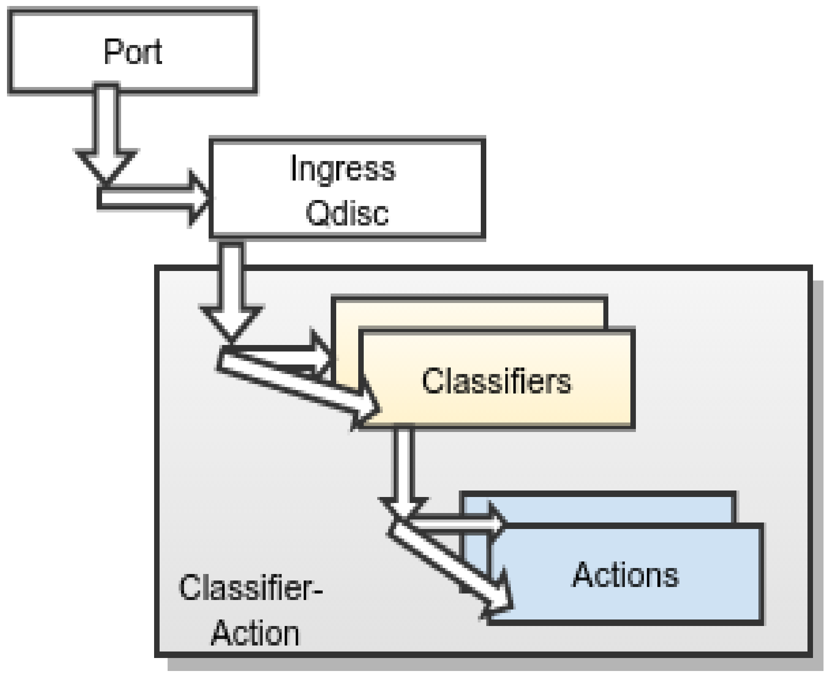
- And Packets cometh...
- And Packets goeth...

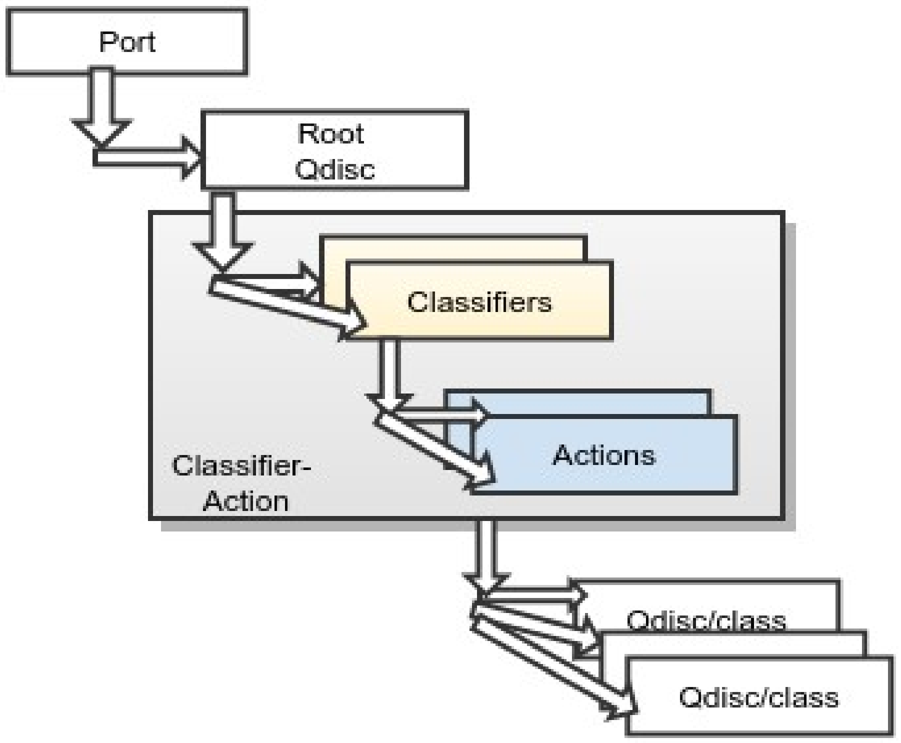
Linux Datapath



- The main packet mangling hooks are traffic control and netfilter
- We will focus on traffic control

Traffic Control Hierarchy





- Note: Ingress side does not have a class(queues)
- Our focus is on Classifiers and Actions
  - We will refer to those two as CA

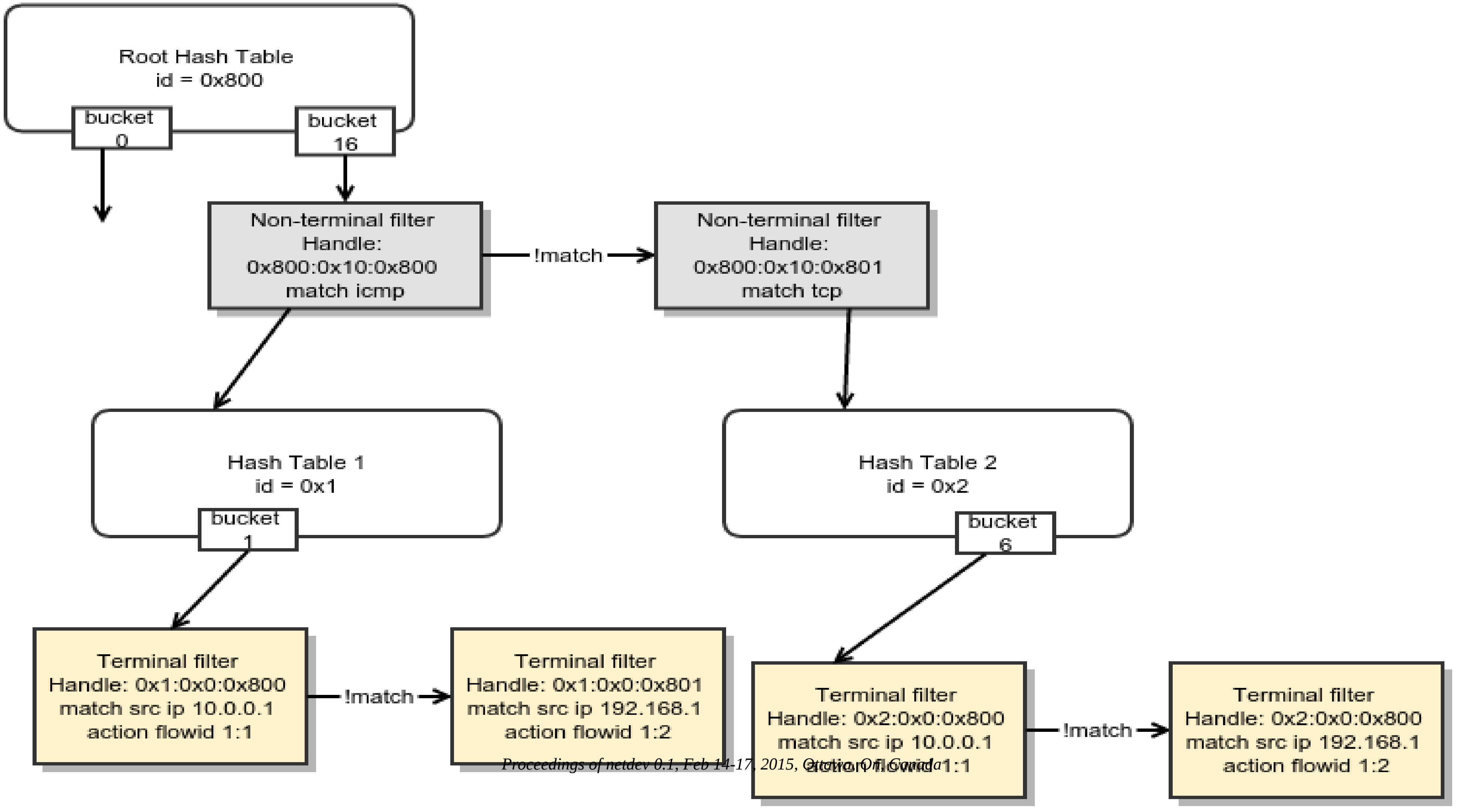
Early History

- Alexey Kuznetsov is the originator of TC and most of the architecture as it stands right now
  - Much of the flexibility and beauty
  - Initial patches around kernel 2.1
- Werner Almesberger did a *lot* of formative work (many things: classifiers, qdiscs, general education)
- Jamal created the “A” part of “CA” (and current maintainer)
- DaveM who was actively involved in those days

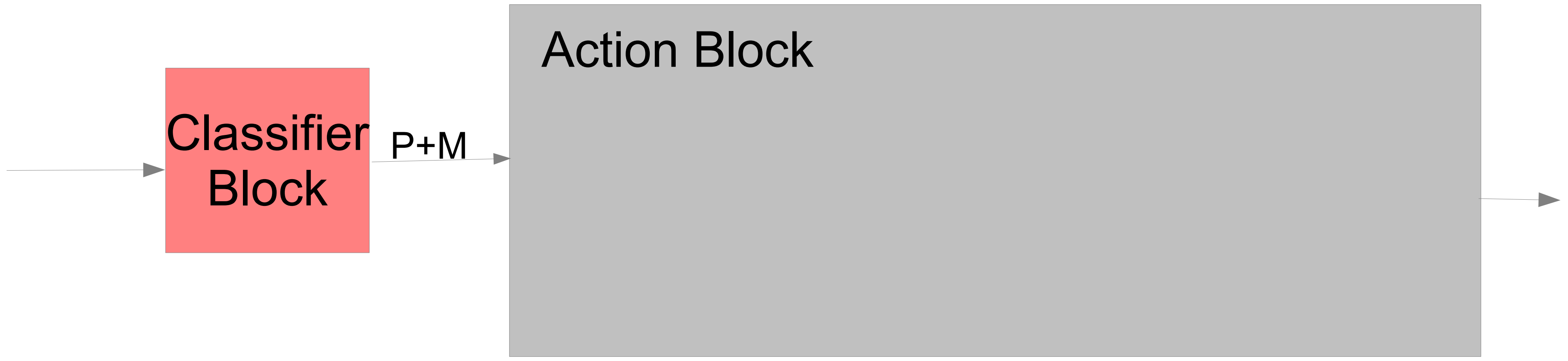
Classifiers

- Classifiers hold filters which segregate traffic
  - Built-in default classifier based on protocol
- Many different types of classifiers
  - No such thing as a universal classifier
  - Each does something they are good at
    - Unix philosophy
  - Types can be mixed and matched when creating policies
- Example of classifiers
  - U32, fw, route, rsvp, basic, bpf, flow, openflow, etc
    - Example u32 could be used to build an efficient tree for packet lookup based on chunks of 32-bit packet blocks
    - Route is efficient with IP based route attributes

U32 Classifier

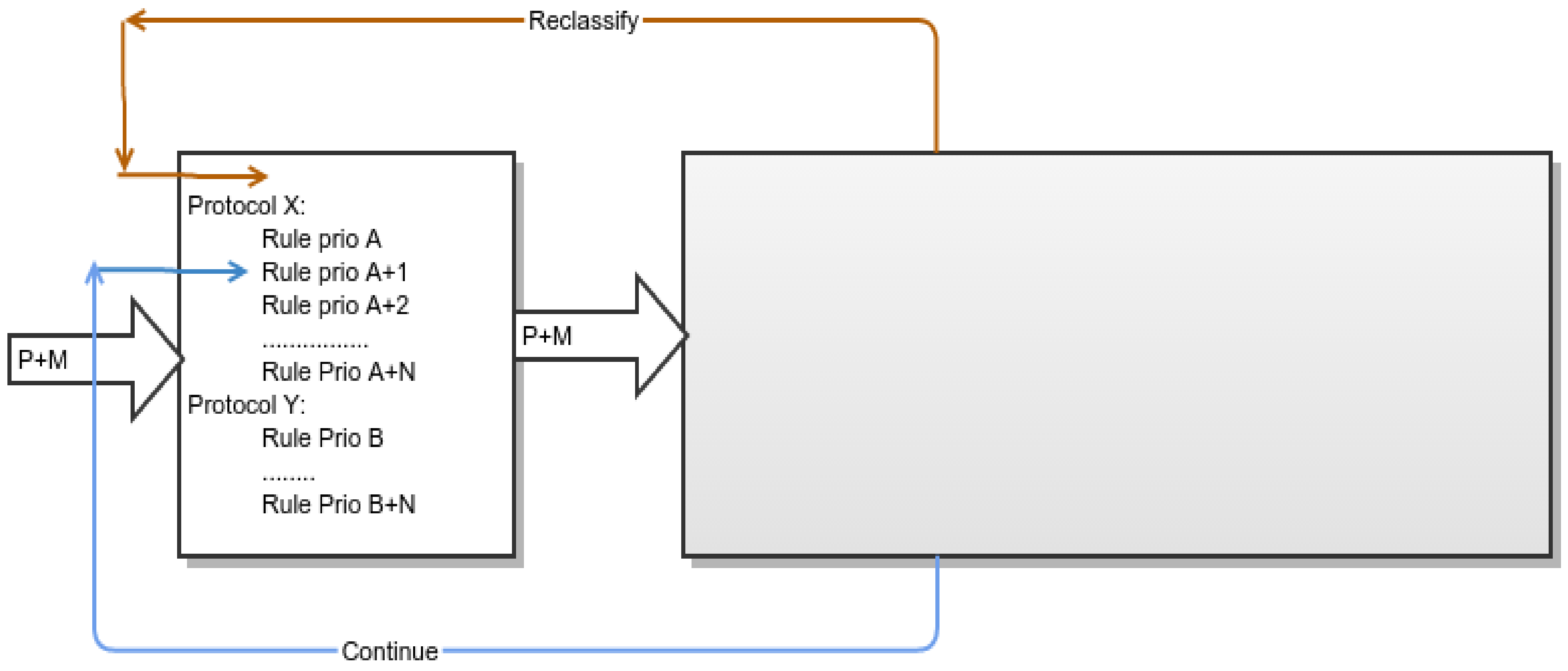


TC Classifier-Actions



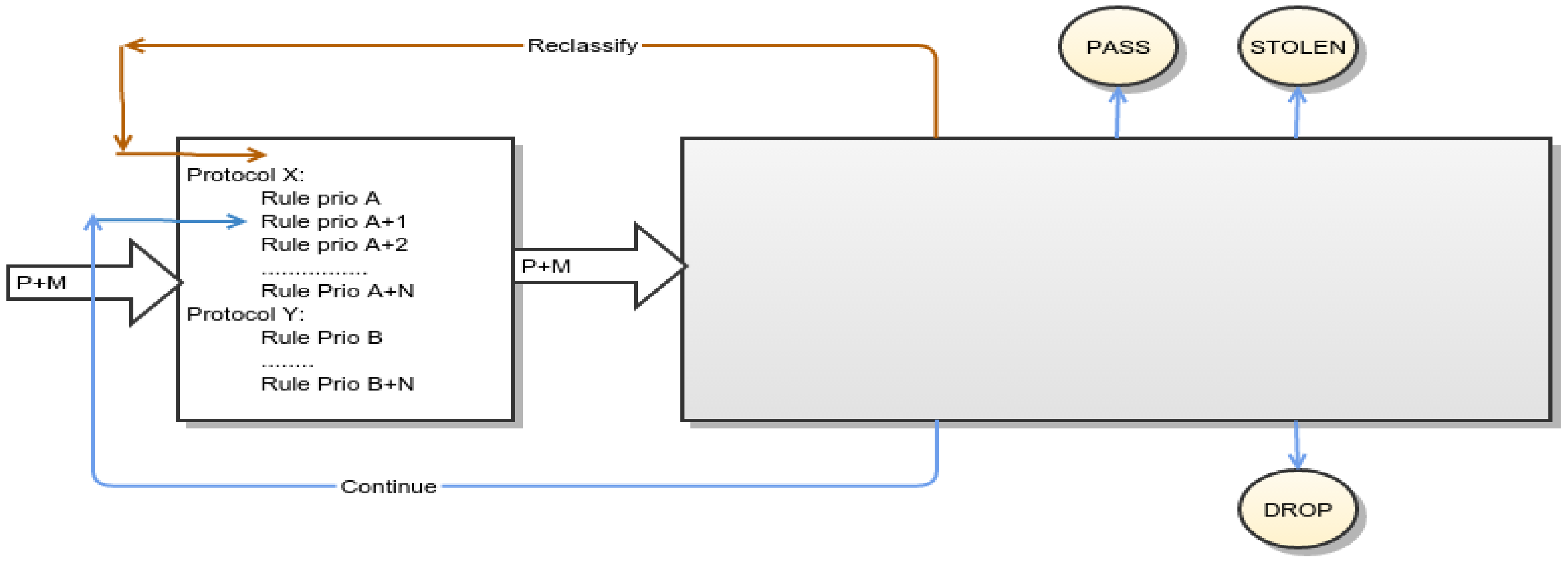
- Packet + Metadata exchanged between the 2 blocks
- Can create a policy graph made of filters and actions
  - Graph flow is programmable at both blocks
    - Programming Constructs and flow control:
      statement, if, else, while, goto, continue, end

CA Programmatic Flow Control



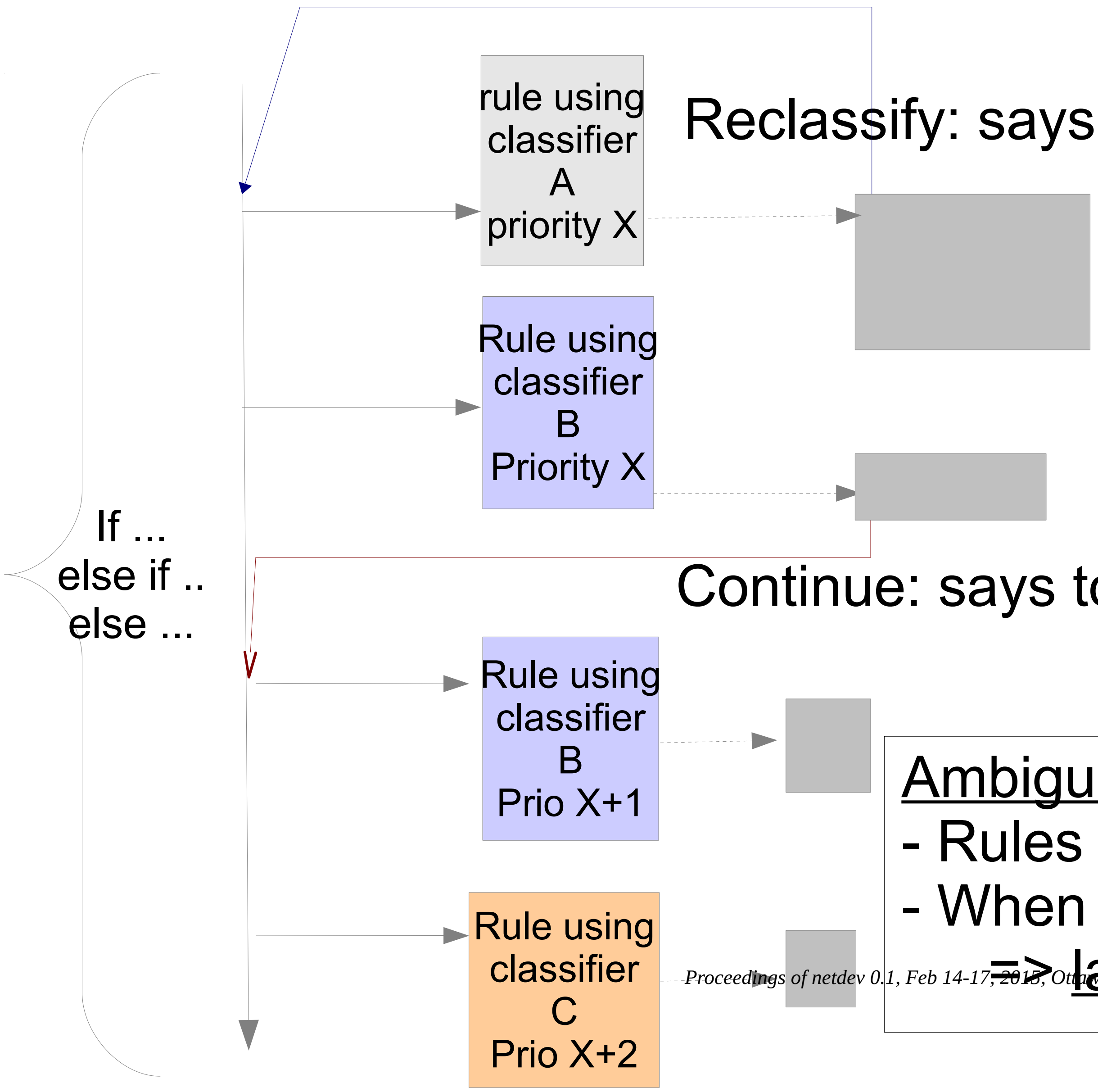
- Priority arrangement of rule predicates is equivalent to *if/else if/else*
- Rules of the same protocol are grouped by priority
- Each rule maybe a totally different classifier algorithm

Classifier Flow Control



- *Continue* construct (contributes to *if/else* branching)
  - Essentially *continue onto next classifier rule*
    - Useful for having default policies and overriding rules
- *reclassify* construct *(jump-back operation)*
  - Useful for adding or removing tunnel headers
  - It means *start the classification again*
- All other constructs(Accept/Drop/Steal) terminate the pipeline

Anatomy of a Classifier Block Branching

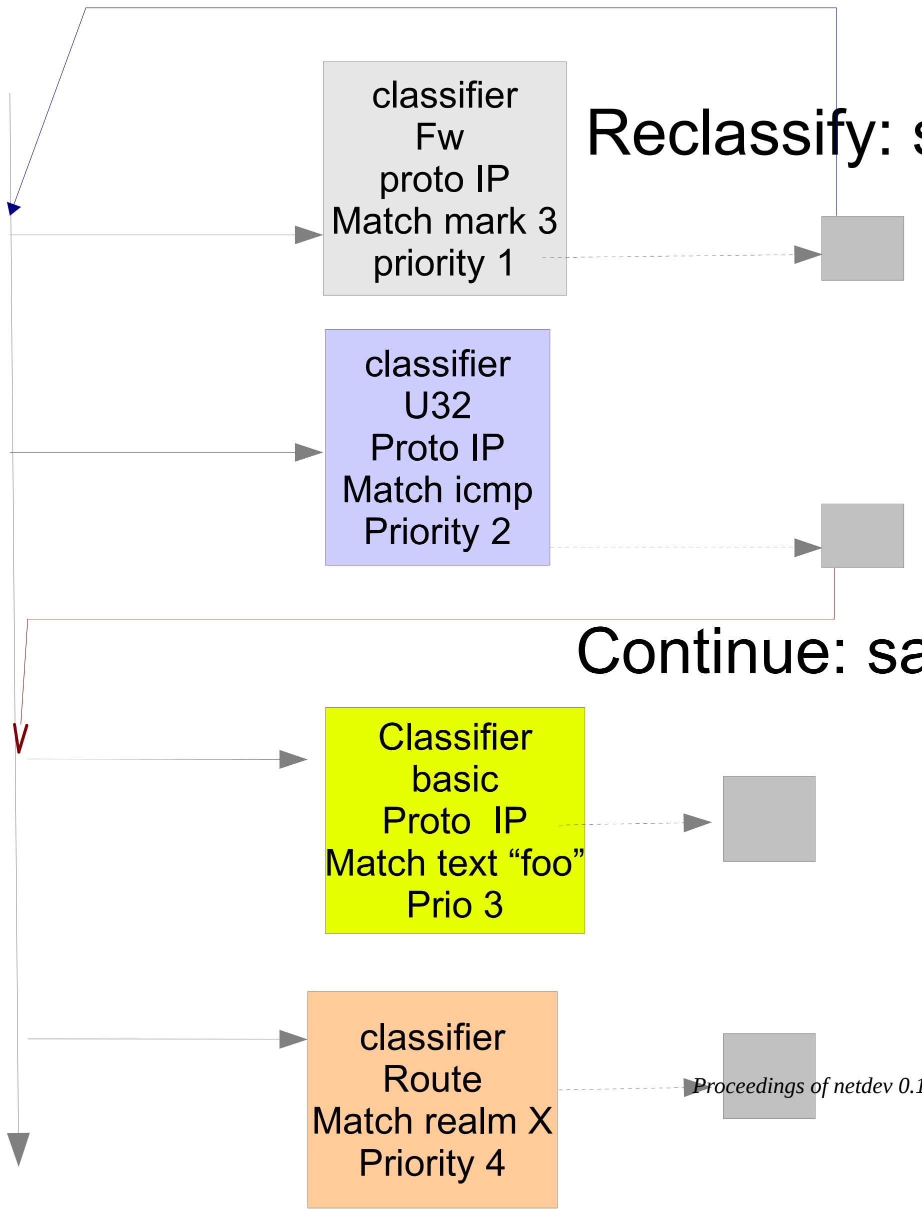

rule using classifier A priority X
Rule using classifier B Priority X
If ... else if .. else ...
Rule using classifier B Prio X+1
Rule using classifier C Prio X+2

Reclassify: says to restart the classification

Continue: says to continue the classification

Ambiguity resolution upto to admin
- Rules are sorted by priorities
- When priority equal then
  => last entered rule more important

Example classifier branching



Reclassify: says to restart the classification

Continue: says to continue the classification

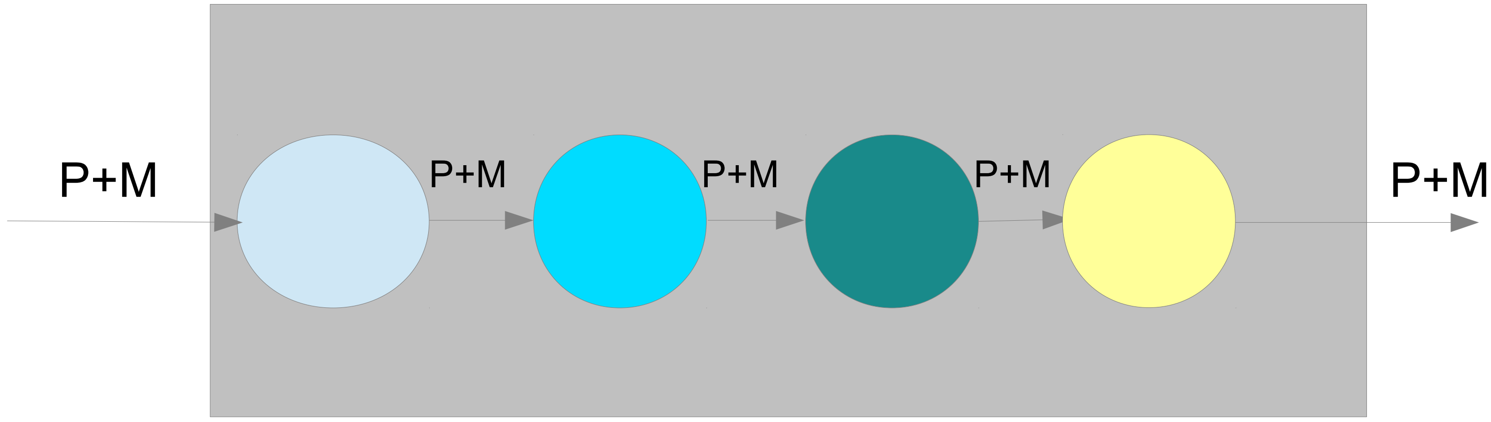
Actions

- Do one small thing they are good at
  - Unix philosophy
- Typically the attributes of each instance of a specific action sit in a table row
  - Creation from the control plane is equivalent to adding a table row

Actions

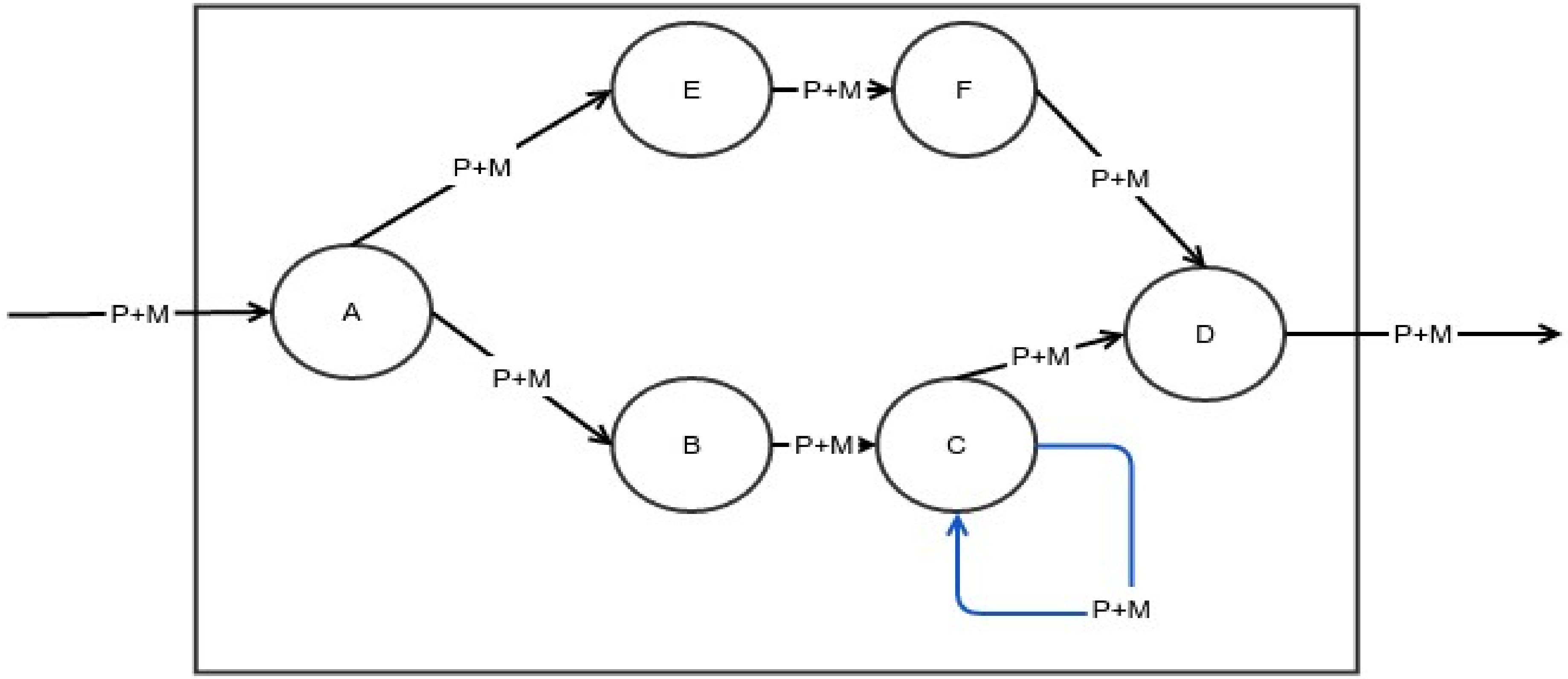
- Many actions exist
  - nat, checksum, TBF policing, generic action (drop/accept), arbitrary packet editor, mirroring, redirect, etc
- Each action instance maintains its own private state which is typically updated by arriving packets
- Each action instance carries attributes and statistics
- An action instance can be shared across more than one service graph

TC Actions: Simple chain



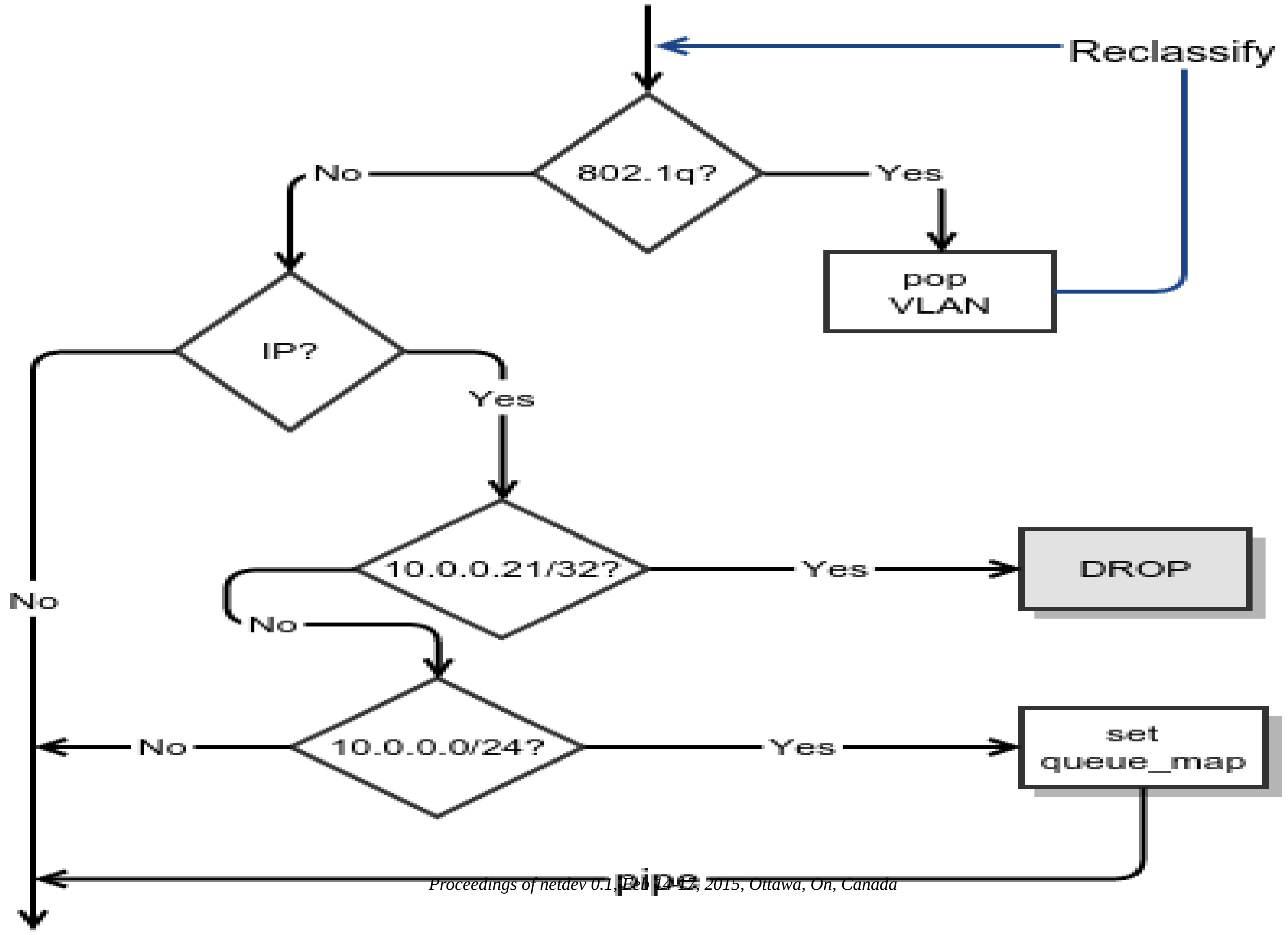
- Actions policy chain using using *pipe* construct (emulating the *unix* | *operator*)
  - i.e *pipe a packet across actions*
- As in Unix *pipe* chain can conditionally be terminated earlier by any action
  - Action state, packet *Drop*, Packet *Acceptance*, Packet *stealing*

Actions: Branching Control

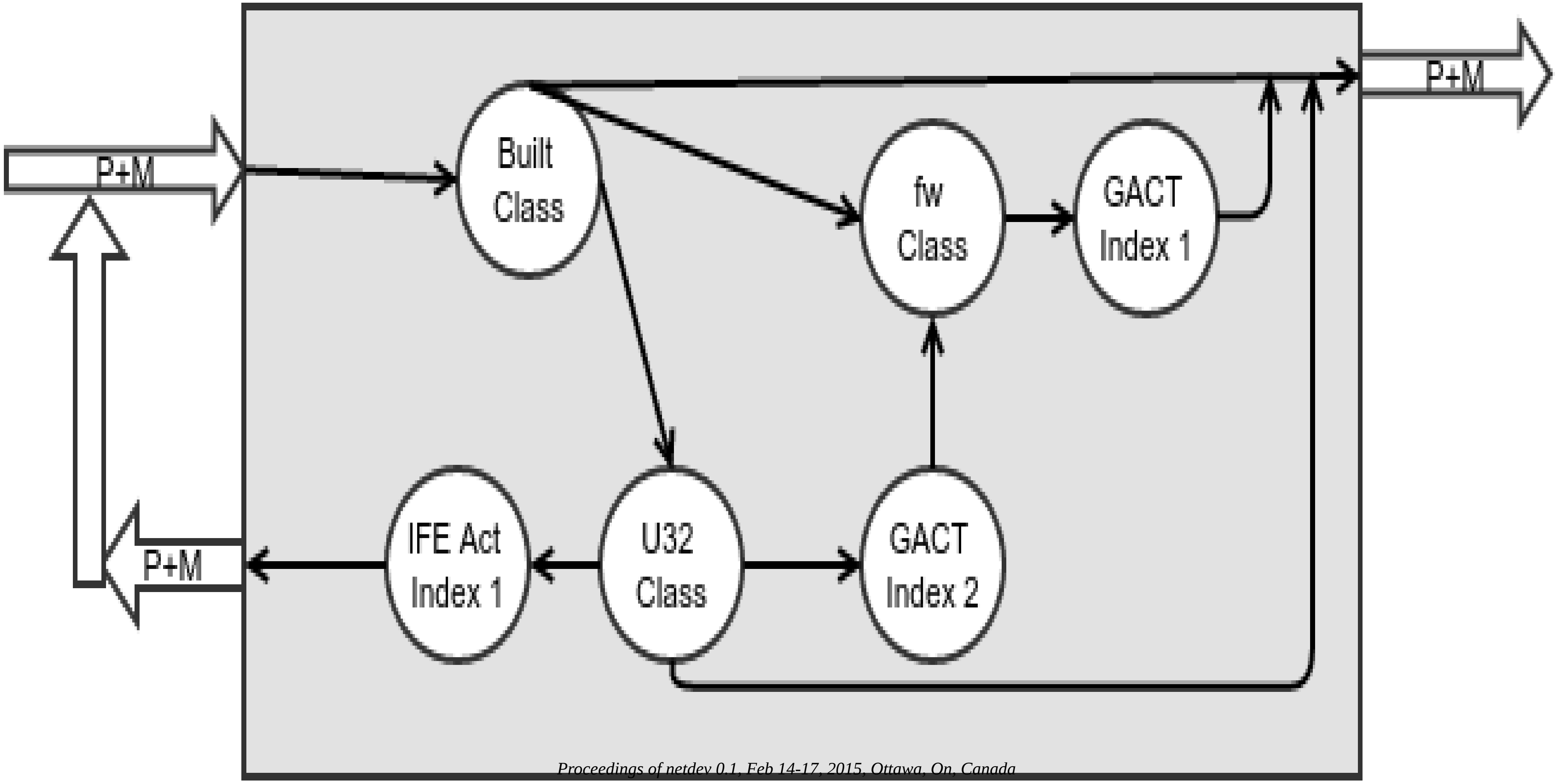


- *if* and *else* conditions programmed in action instance
- Any action could conditionally <u>repeat</u> (REPEAT)
  - Loop construct

A Simple Program



A Simple Program: Functional View



Summary: Classifier-Action Pipeline

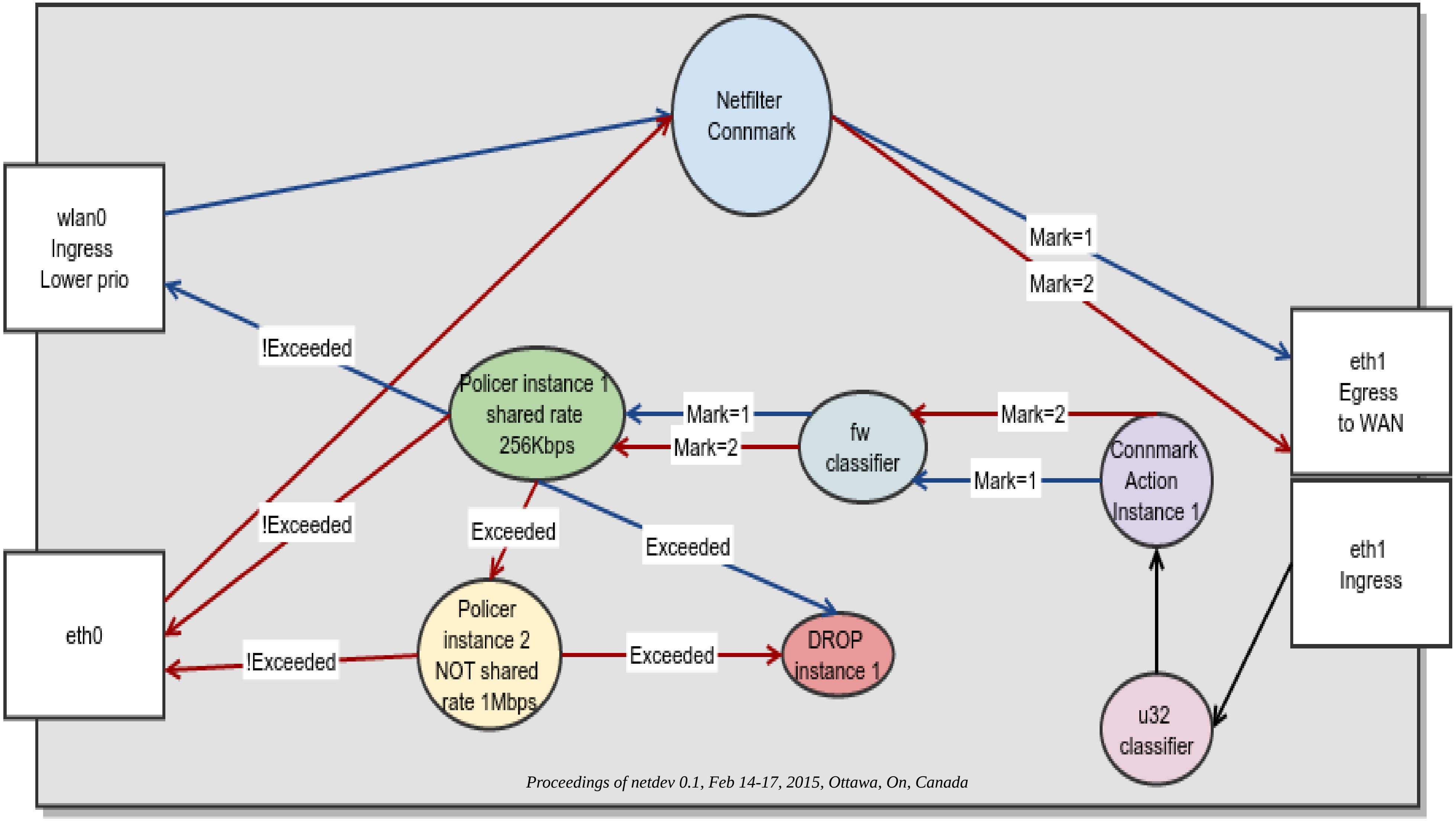
Classifier Programmatic control

- CONTINUE (*iterate* next rule)
- RECLASSIFY (*restart* pipeline)
- All others (*end* CA pipeline)

Action Programmatic Control

- *Stolen/Queued (end **CA** pipeline)*
- *DROP (end **CA** pipeline)*
- *ACCEPT (end **CA** pipeline)*
- PIPE (*iterate* next action)
- *CONTINUE (end **A**ction pipeline)*
- *RECLASSIFY (end **A**ction pipeline)*
- REPEAT (*restart* action processing)
- JUMPx (*jump* X actions in pipeline)

Sharing Actions: IMQ



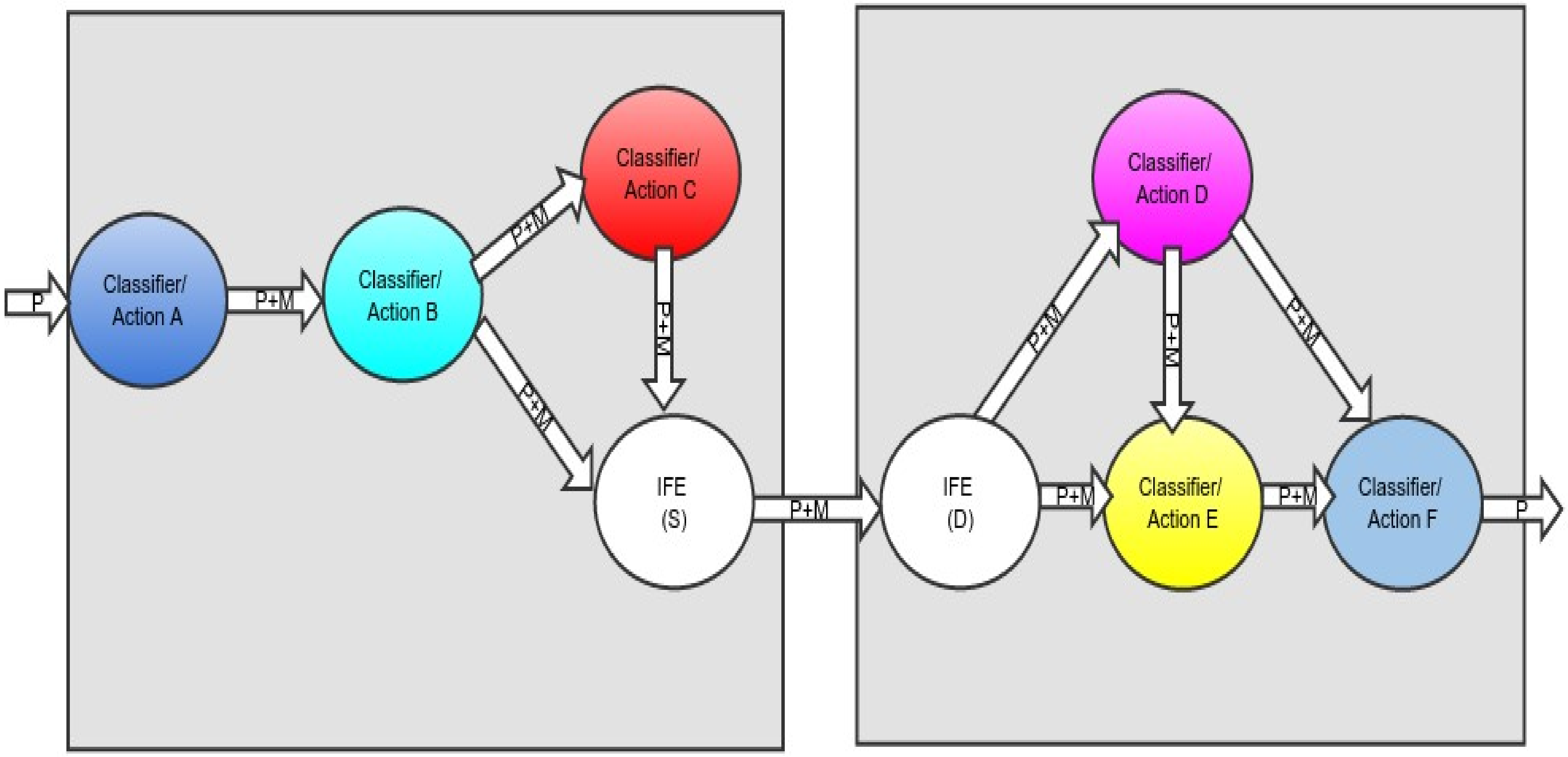
Aging of Policies

- All Actions keep track of when they were installed and last used
- Control side can use this info to implement aging algorithms

Late Binding

- Action instances can be created
- Later bound to policies

Distributing CA



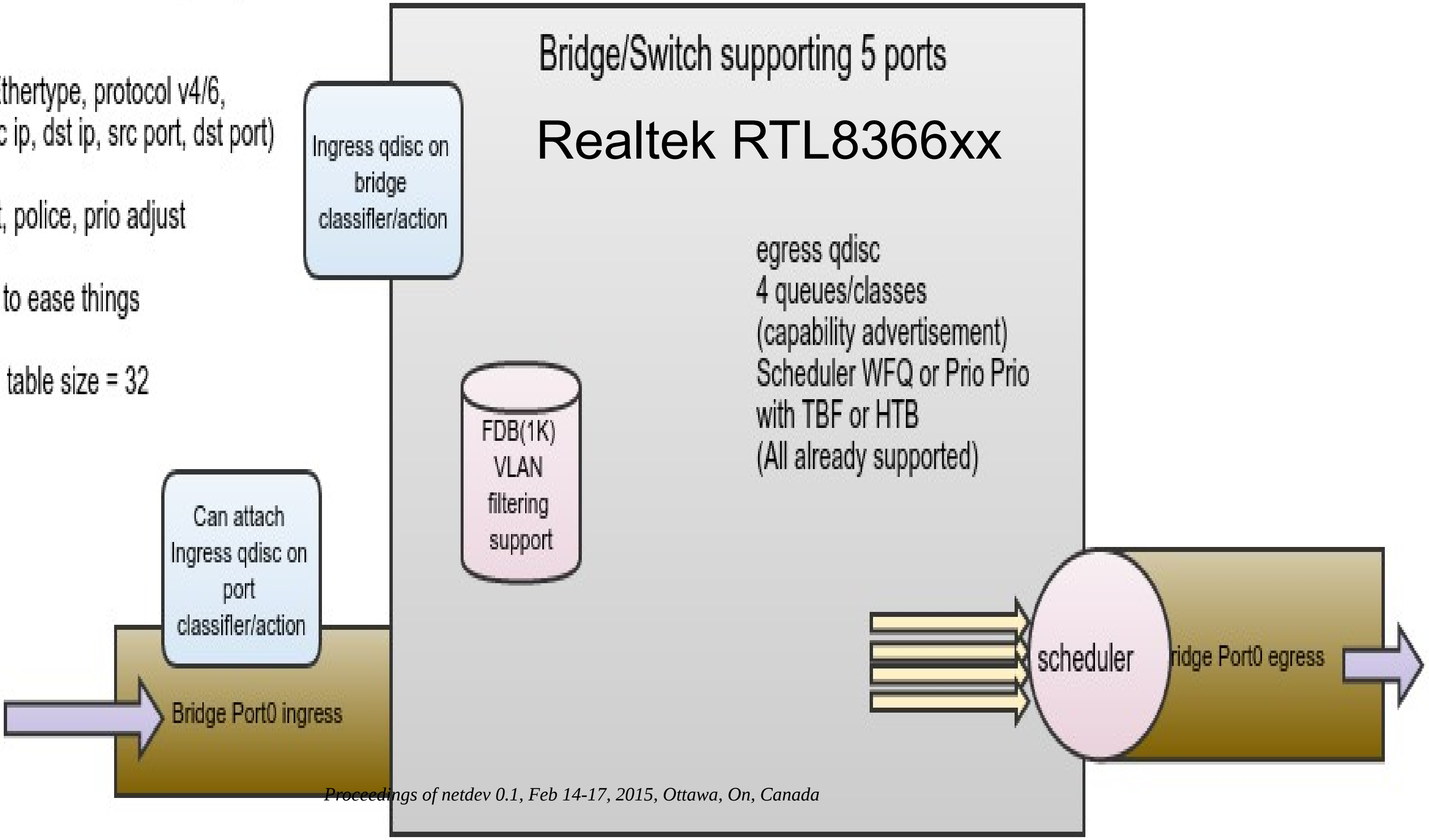
Future Work

- More Classifiers and Actions of course
- Functional discovery
- Usability
  - tcng effort by Werner
  - Programmability extension into higher level language (python, lua etc)

Future Work: Hardware Offload

** Essentially ingress qdisc attached to bridge or port
32 ACL rules:
 Classifier:
   (src MAC, Dst MAC, Ethertype, protocol v4/6,
    ip proto = tcp/udp, src ip, dst ip, src port, dst port)
Actions:
   DROP, mirror, redirect, police, prio adjust

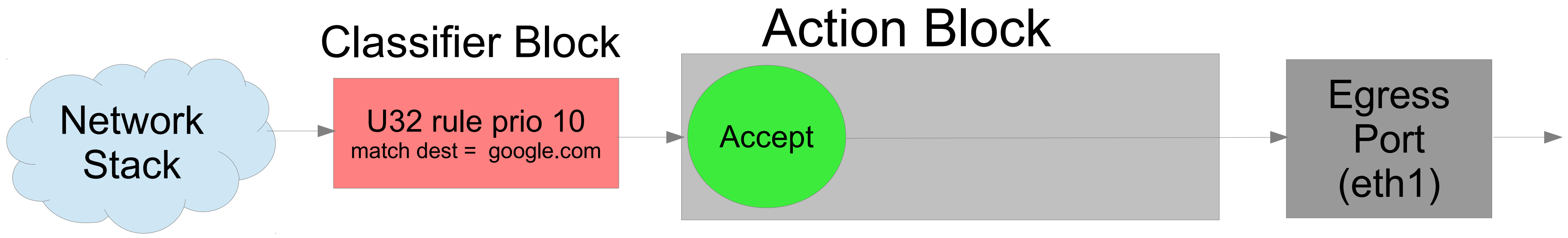
New tc classifier needed to ease things
no new actions needed
capability advertisement: table size = 32



Bridge/Switch supporting 5 ports
Realtek RTL8366xx
Ingress qdisc on bridge classifier/action
egress qdisc
4 queues/classes
(capability advertisement)
Scheduler WFQ or Prio Prio
with TBF or HTB
(All already supported)
FDB(1K) VLAN filtering support
Can attach Ingress qdisc on port classifier/action
Bridge Port0 ingress
scheduler
ridge Port0 egress

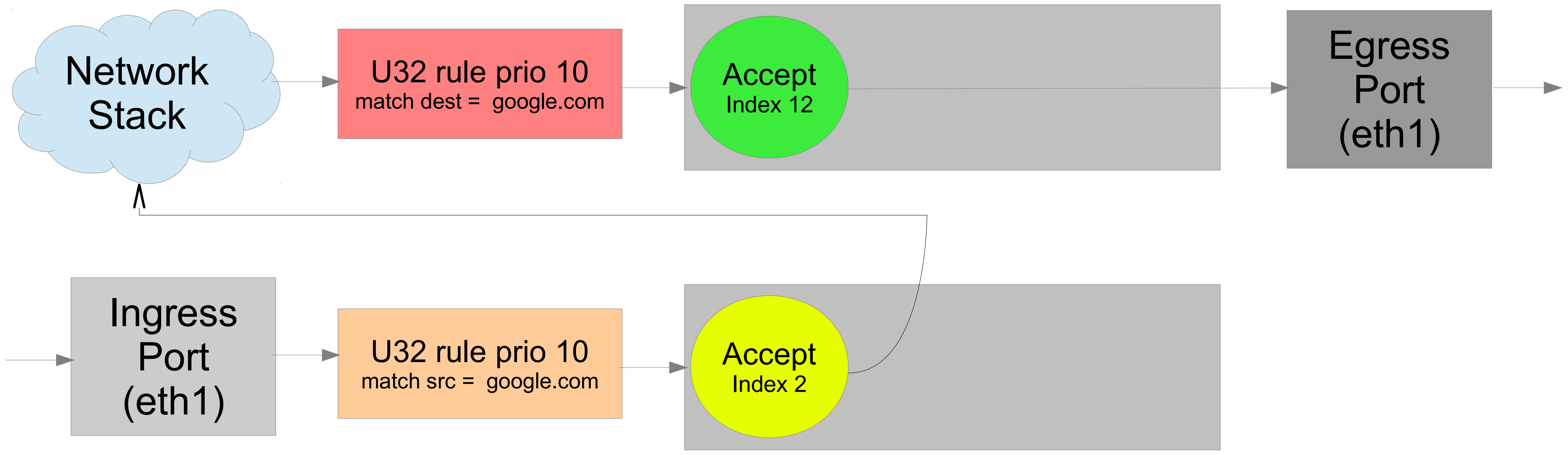
Lets Write Some Programs

Counting Packets To A Host



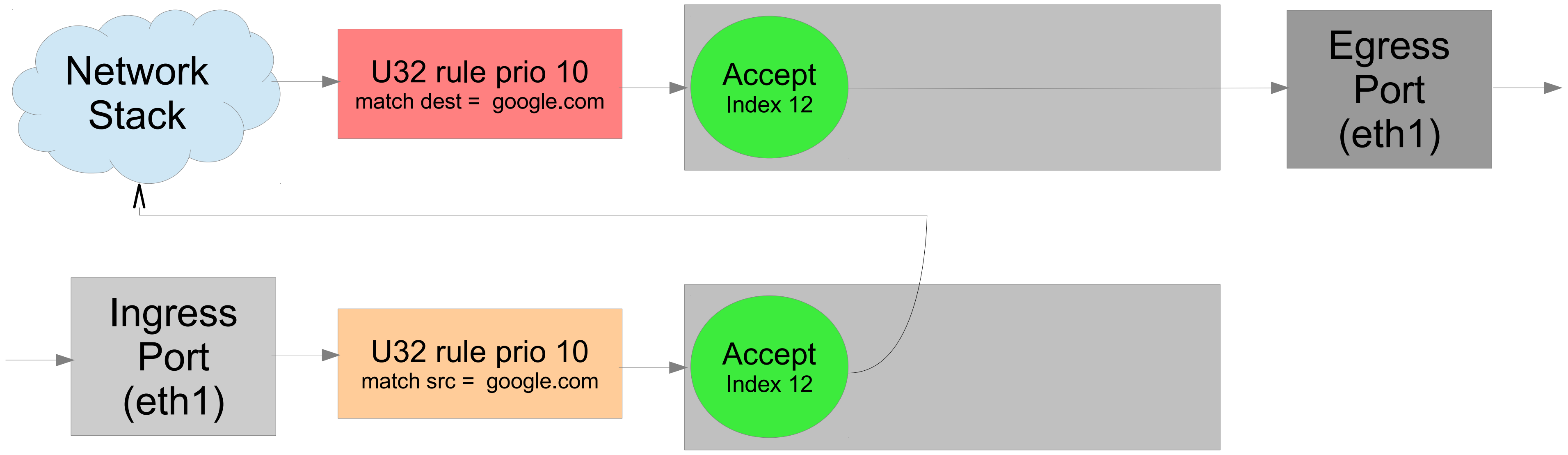
- Goal: get acquinted with the control setup via CLI
- Ping google.com
- Show statistics

Counting Packets To/From A Host



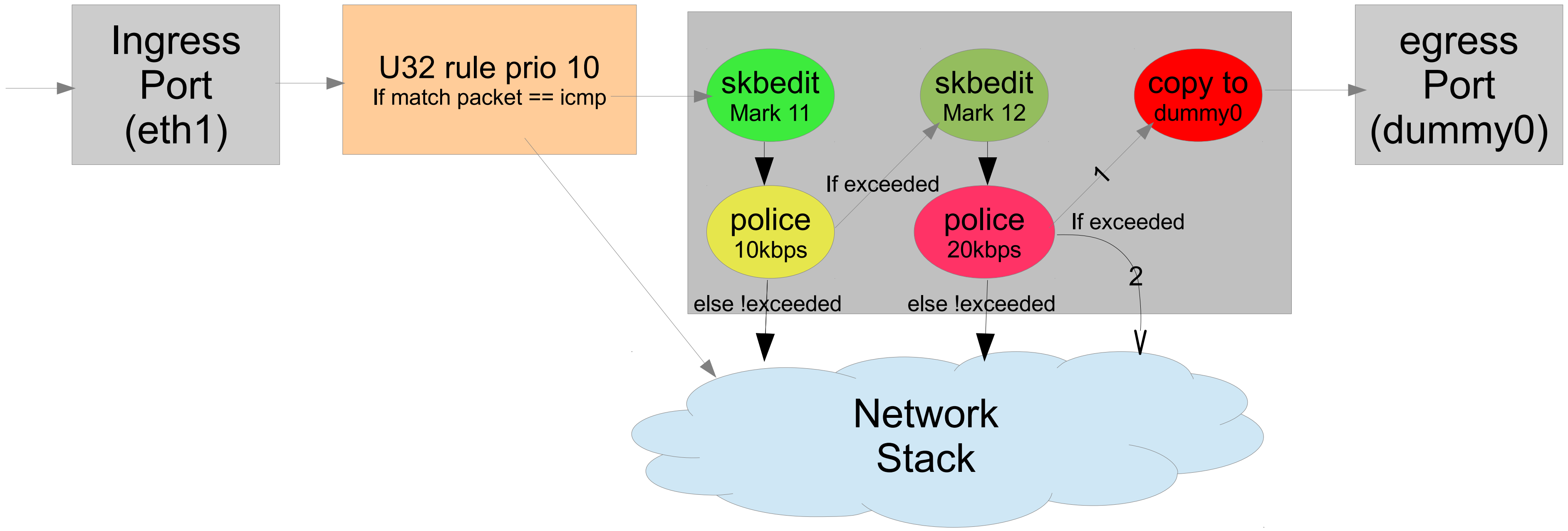
- Goal: get acquinted with the control setup via CLI
- Ping google.com
- Show statistics

Counting Packets To/From A Host Shared Action Instance



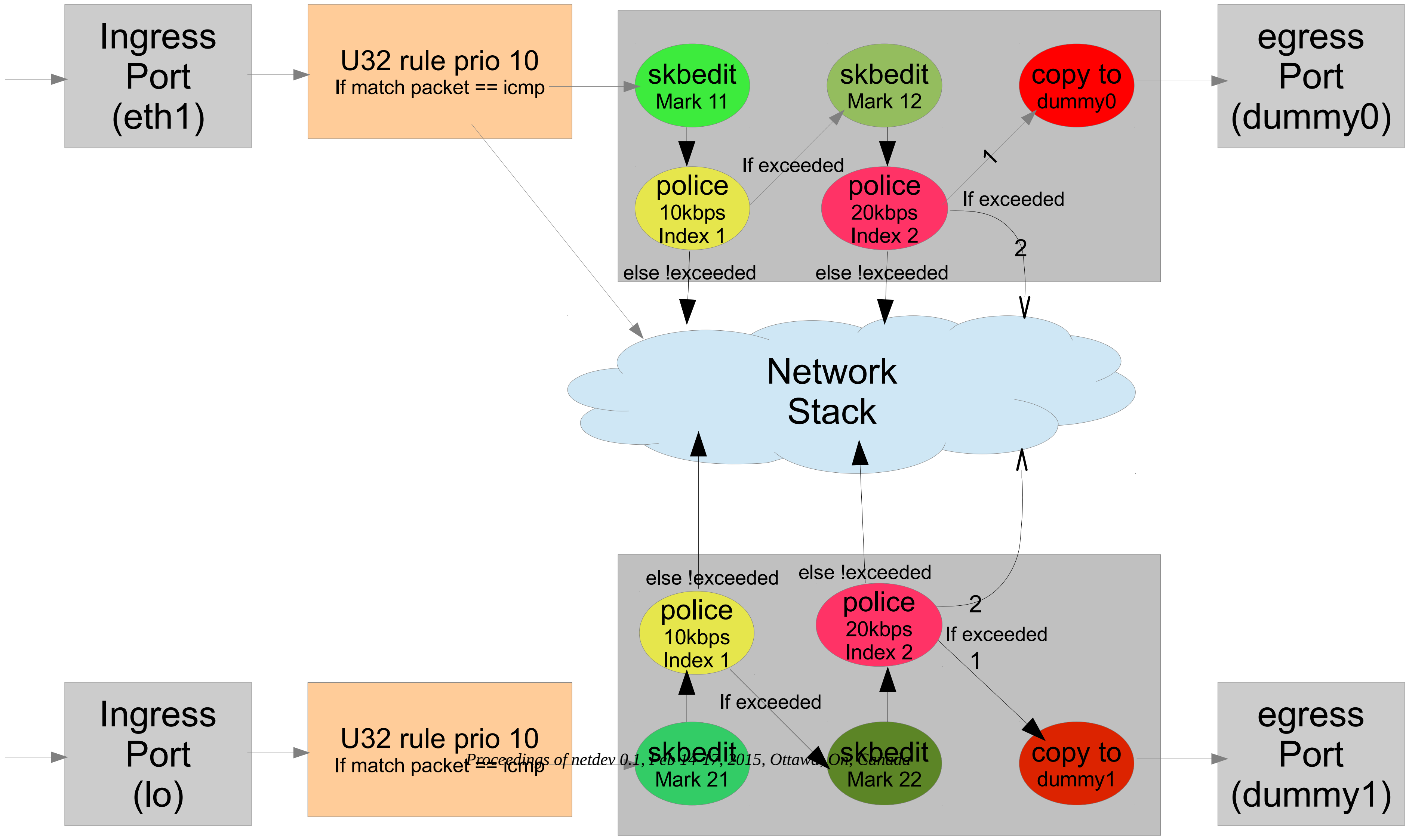
- Goal: A little more complex setup (sharing action instance)
- Ping google.com and show statistics
- Broken for ubuntu shipped kernels and iproute2

More Complex Service



- Goal: Illustrate a more complex service
  - More complex action graph
- Broken for ubuntu shipped kernels and iproute2

More Complex Service Shared Rate control


Ingress Port (eth1)
U32 rule prio 10
If match packet == icmp
skbedit Mark 11
police 10kbps Index 1
If exceeded
skbedit Mark 12
police 20kbps Index 2
copy to dummy0
1
If exceeded
2
else !exceeded
else !exceeded
egress Port (dummy0)
Network Stack
else !exceeded
else !exceeded
police 10kbps Index 1
police 20kbps Index 2
2
If exceeded
1
If exceeded
Ingress Port (lo)
U32 rule prio 10
If match packet == icmp
skbedit Mark 21
skbedit Mark 22
copy to dummy1
egress Port (dummy1)