2022 P4 Workshop
May 24-26th

P4 Over Linux TC

Scripting Away At P4

Jamal Hadi Salim (Mojatatu Networks)
Victor Nogueira (Mojatatu Networks)
Pedro Tammela (Mojatatu Networks)
Deb Chatterjee (Intel)
Anjali Singhai Jain (Intel)
Evangelos Haleplidis (Mojatatu Networks)

Goal: Extend P4 Ecosystem To The Linux Kernel

Core Requirement: **Scriptable Hardware offload of P4 MAT**

Target personas:

- The P4 developer
  - Knowledgeable in P4 - No interest in kernel internals
- Traditional Linux (non-P4) ops
  - Knowledgeable in Linux - No interest in P4 internals

Why P4 TC In The Linux Kernel?

- Grow P4 ecosystem to a much larger audience of devs and ops
  - Take advantage of existing, widely used infrastructure
    - Containers, VMs, baremetal, servers, middle boxes, etc
- TC infrastructure maps to P4 MAT with built-in HW full/partial offload
  - Very little core kernel changes required to support new P4 infrastructure code
- Why not DPDK or other user space datapaths?
  - Kernel provides a singular interface
    - No vendor-specific fragmentation in DPDK based APIs
  - DPDK selling point performance
    - Our view: Performance is achieved by offloading
  - Gain usability by using well understood kernel interfaces and tooling
- Why not ebpf?
  - Hardware offload and scriptability is a core requirement
  - ebpf still in active development (compiler, verifier, kernel, etc)
  - Challenges in writing complex programs (turing completeness issues)
  - Ops entry engagement point is steep

Motivation and Design

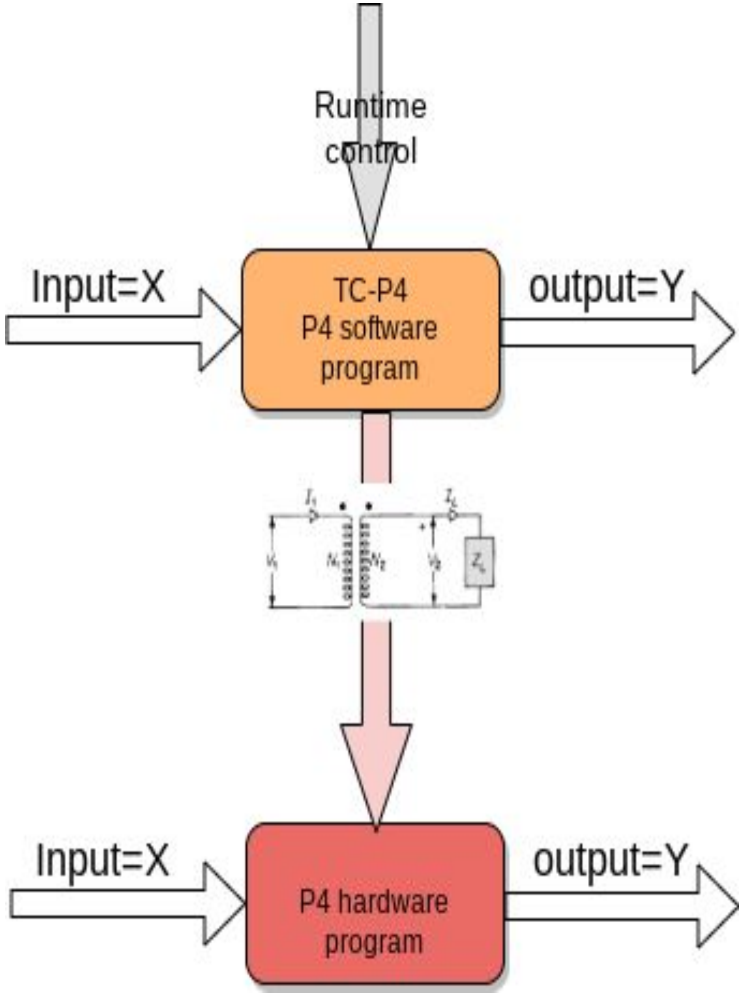
Upstream Sayeth: Thou Shalt Have A Software Twin

Upstream Requirements

- SW and HW are functionally equivalent
- P4 program runs in absence of offloadable HW
- Partial and/or full offload
    - Pipeline exists in both HW and SW

Benefits

- Test your P4 program in the kernel
    - Baremetal, VM, container, etc
    - When ready toggle policy knob to use HW offload
- Build realistic digital twin to mimic deployed offload
- Use P4 in new infrastructure experiments
    - Accelerate when needed



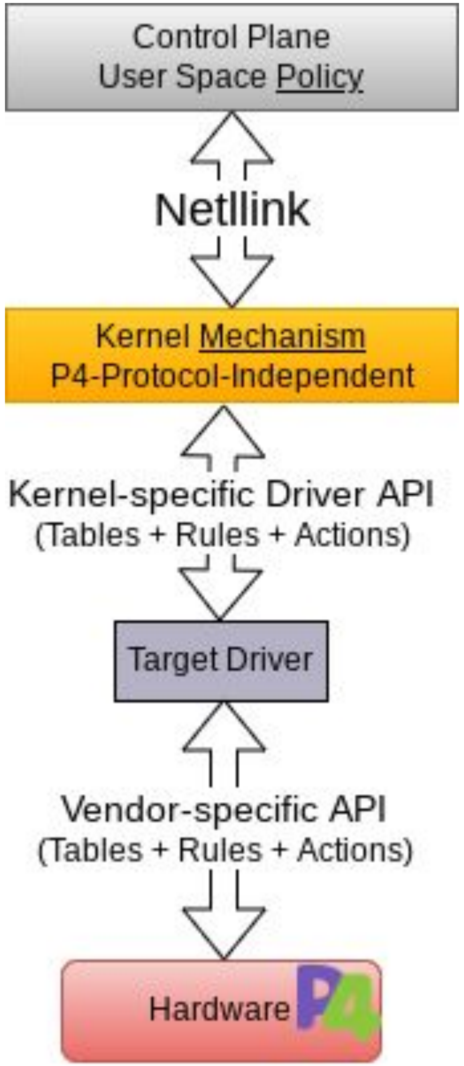
Core Design Principle: <u>Scriptability</u>

<u>Template-driven</u> architecture

- map P4 Program to mechanisms in the kernel
- kernel independence

A P4 program is <u>constructed</u> via scripts from user space

- <u>A new P4 program does not need kernel or user space changes</u>
- Prior art: *u32* classifier, *pedit* packet editor, *skbedit* metadata editor



Motivation: Kernel independence

- Challenges
  - High effort investment for upstreaming (or any kernel coding)
    - Specialized social and technical skills required
  - 2-3 years cycles post-upstreaming to a supported distro (RHEL etc)
    - Speeding up of this process means NRE to the distro vendor
      - Still 6 months vetting release time frames
  - Some DCs: Upto a year cycle if newly coded feature requires compilation
    - Compiled(binary blob) change require vigorous validation
    - No such restrictions in operational scripting
- Side-Benefits
  - Lowering entry point for defining a new datapath or protocols
  - Less kernel code => reduce kernel maintenance overhead
  - No need to battle newer kernels, compilers, tools, skills, etc
    - No need for inhouse gurus

Scriptability Roadmap

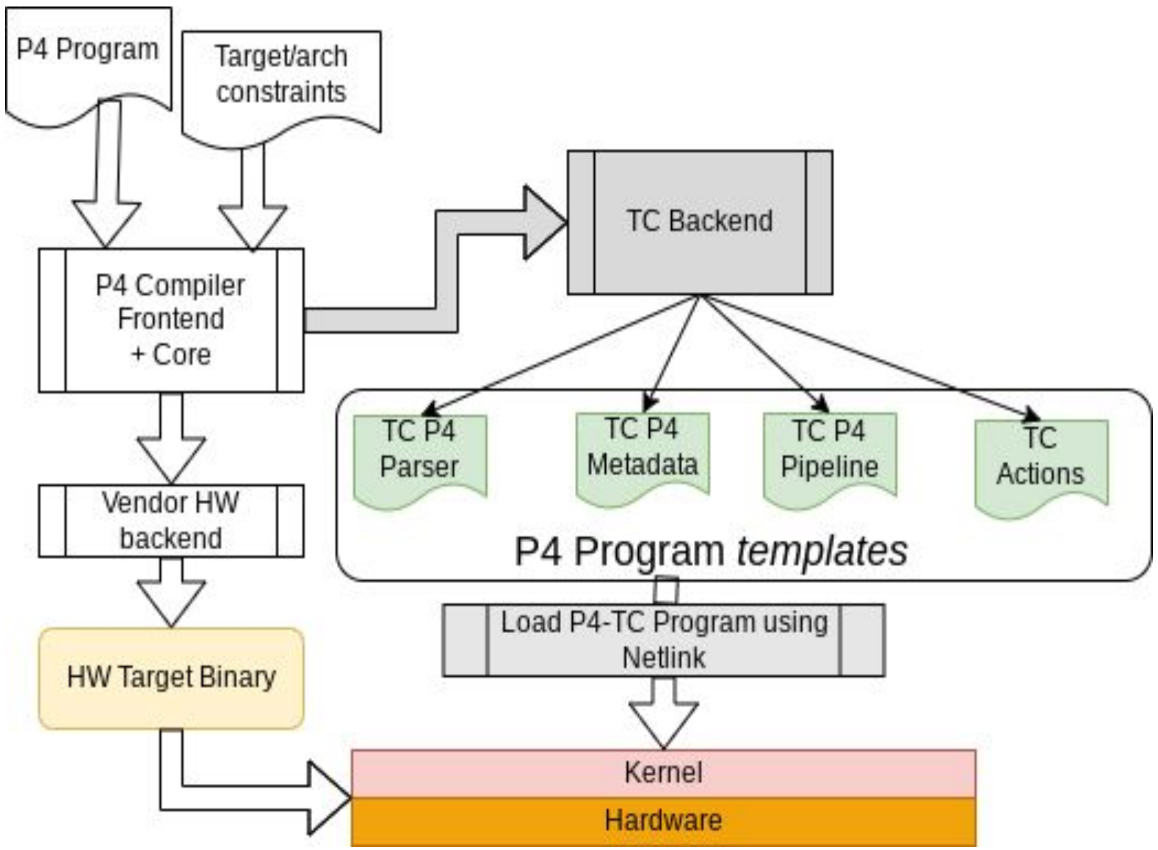
- Upstream Once
  - Add all features needed into the kernel
    - Add any missing functionality in TC to support P4 generically
  - Any subsequent changes to the kernel will be bug fixes
- No Kernel or user space code changes for any P4 prog
  - No code generation or compiling of loadable blobs
    - i.e no kernel modules, ebpf, etc.
- No change to user space/control *iproute2::tc* control/provision
  - Scriptable Introspection (as opposed to a DSL)

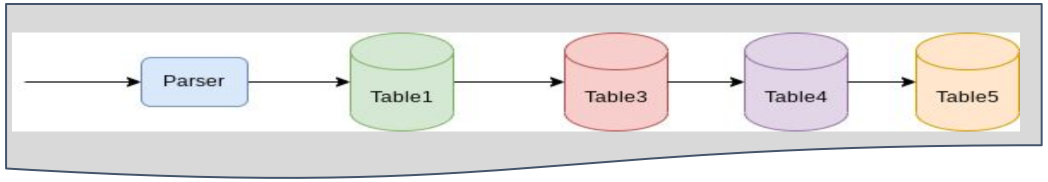
General Design Goals

- Desire to be P4-architecture independent
  - i.e support PSA (for switches) or PNA in NIC environment, etc
- Desire support multi-vendor implementations of P4
- Debuggability
  - Ease developer and operator runtime troubleshooting
- Admission Control for hardware resources
  - Generated Template “Program loading” tells the kernel about resource limits
  - Kernel keeps state of in-use resources
- Efficient Control Plane Table CRUD Netlink Messaging
  - Lower Latency for Individual transactions
  - Higher Throughput for batching

Workflow
Program Installation And Runtime

Workflow: Creating And Installing The P4 Program

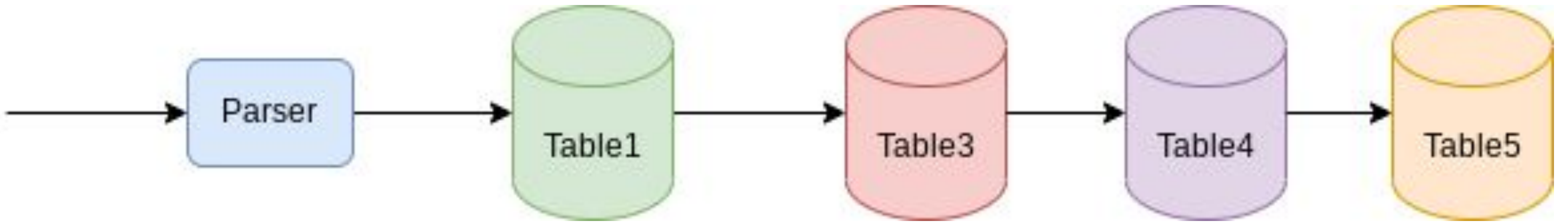




P4 Dev approach
1. Author writes the P4 program
2. Build P4 program (using a compiler) targeting TC and/or hardware
3. Author/dev executes the tc template scripts to "install P4 program"

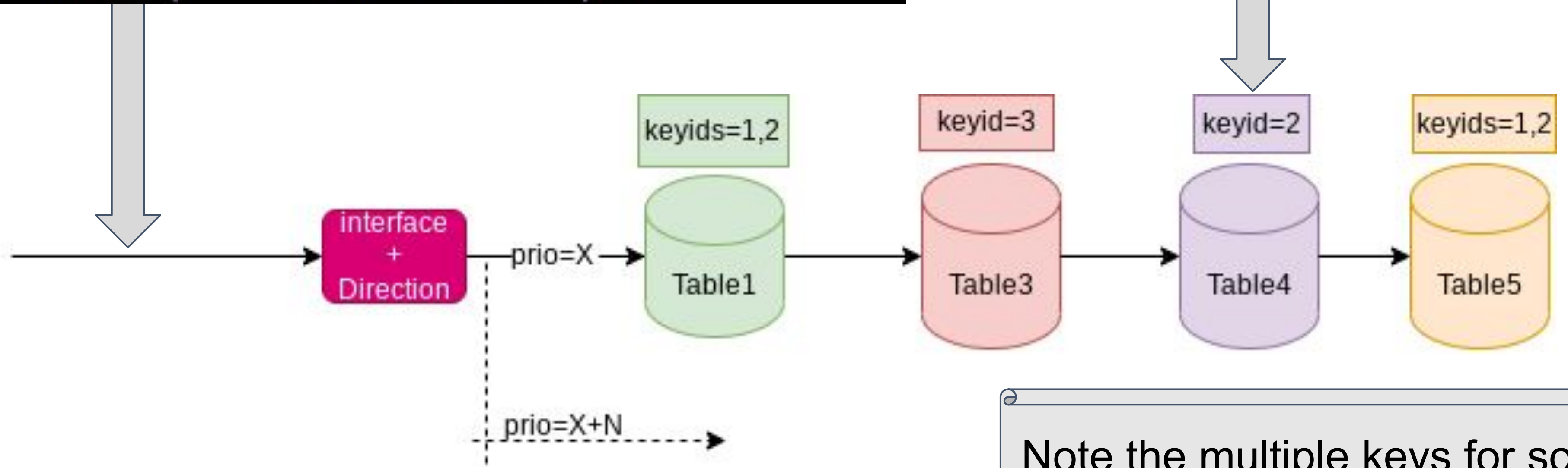
Linux Ops approach
1. Manually create tc scripts
2. Execute them to install

Runtime: P4 Program Mapping to TC Infrastructure



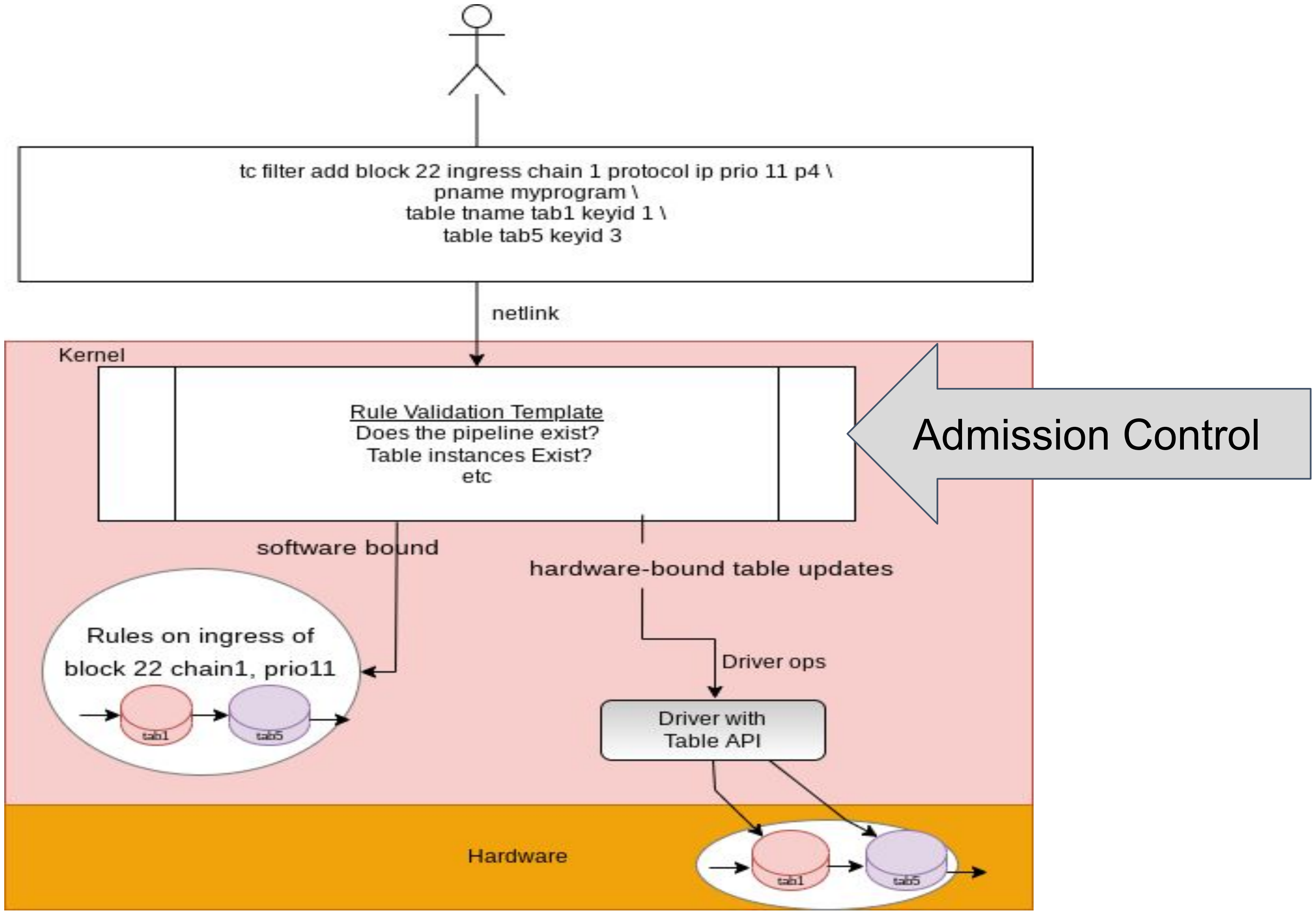

```
tc filter add dev eth0 ingress protocol all prio 5 p4 \
pname myprog table tname table1 keyid 1 table tname table3 keyid 3 \
table tname table4 keyid 2 table tname table5 keyid 1
```

```
tc p4 create myprog/table/table4 ip/dstaddr 192.168.0.0/16 prio 17 \
skip_sw \
action drop
```

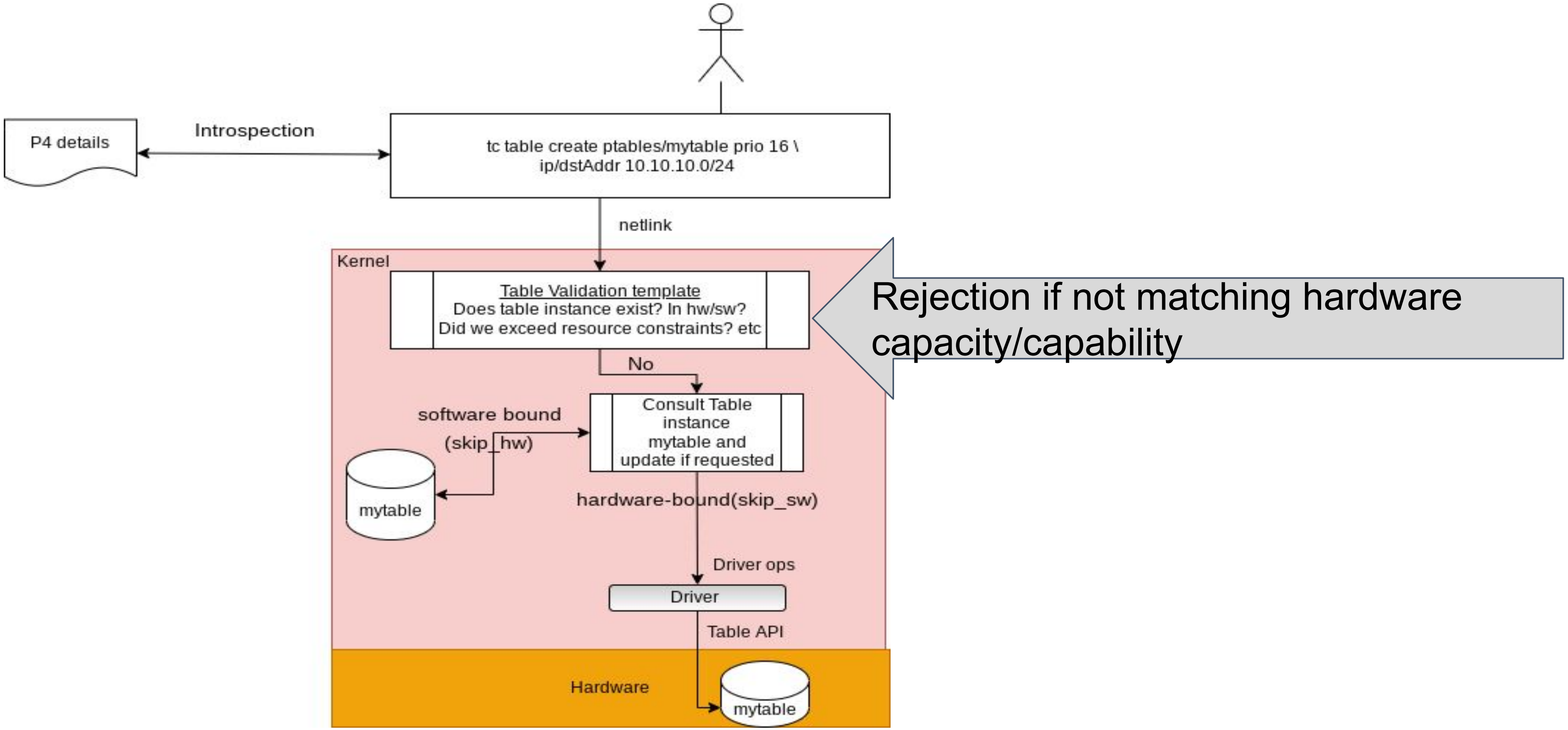


Note the multiple keys for some tables..

Runtime: P4 Pipeline Control

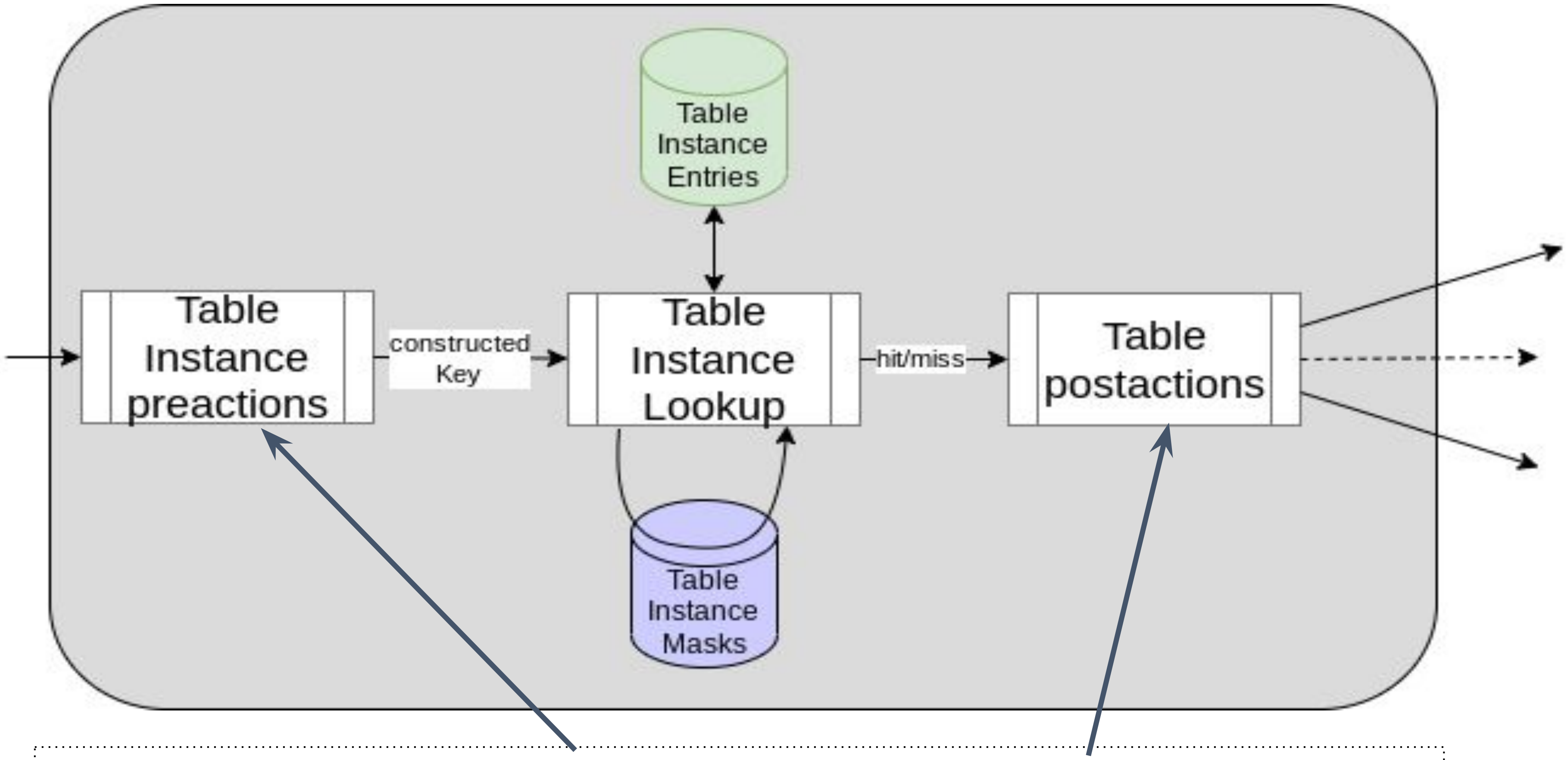


Runtime: P4 Table Control



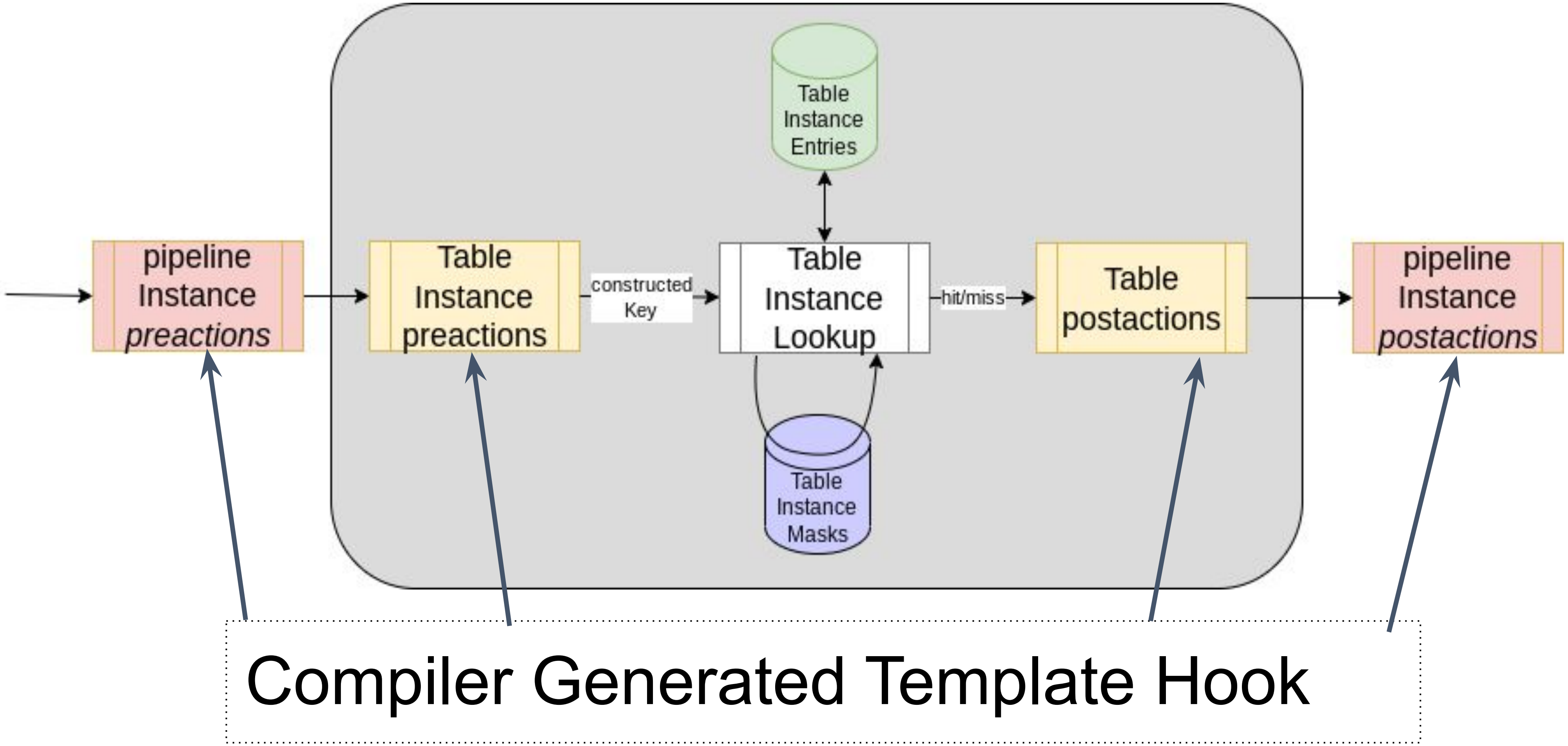
P4 Objects Kernel Abstractions

Compiler Generated Template Hooks: Table Instance *Preactions* And *Postactions*



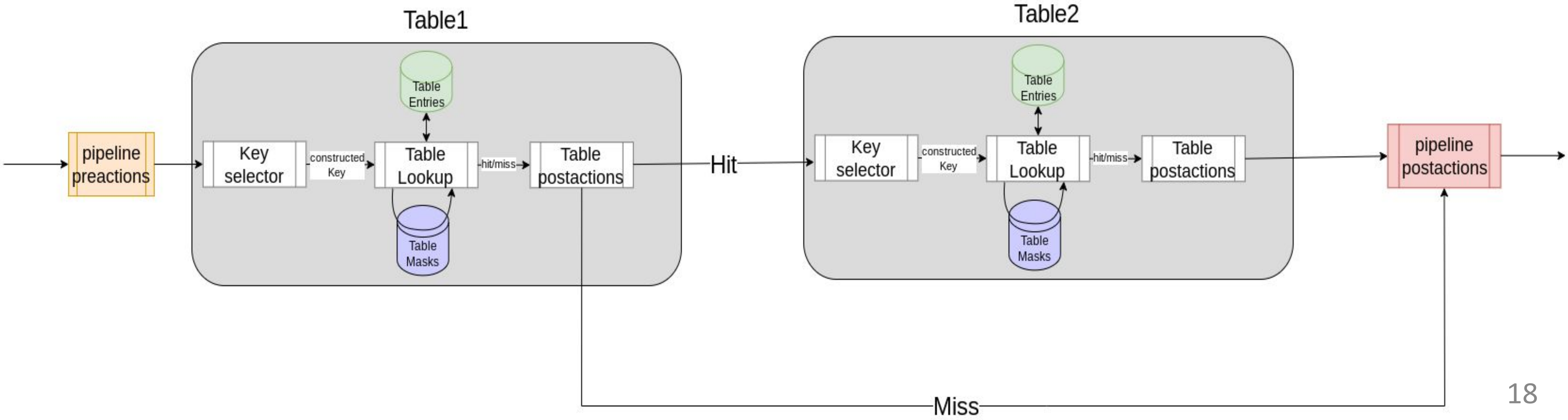
Compiler Generated Template hook

Compiler Generated Template Hooks: Pipeline *Pre* and *Post* Actions



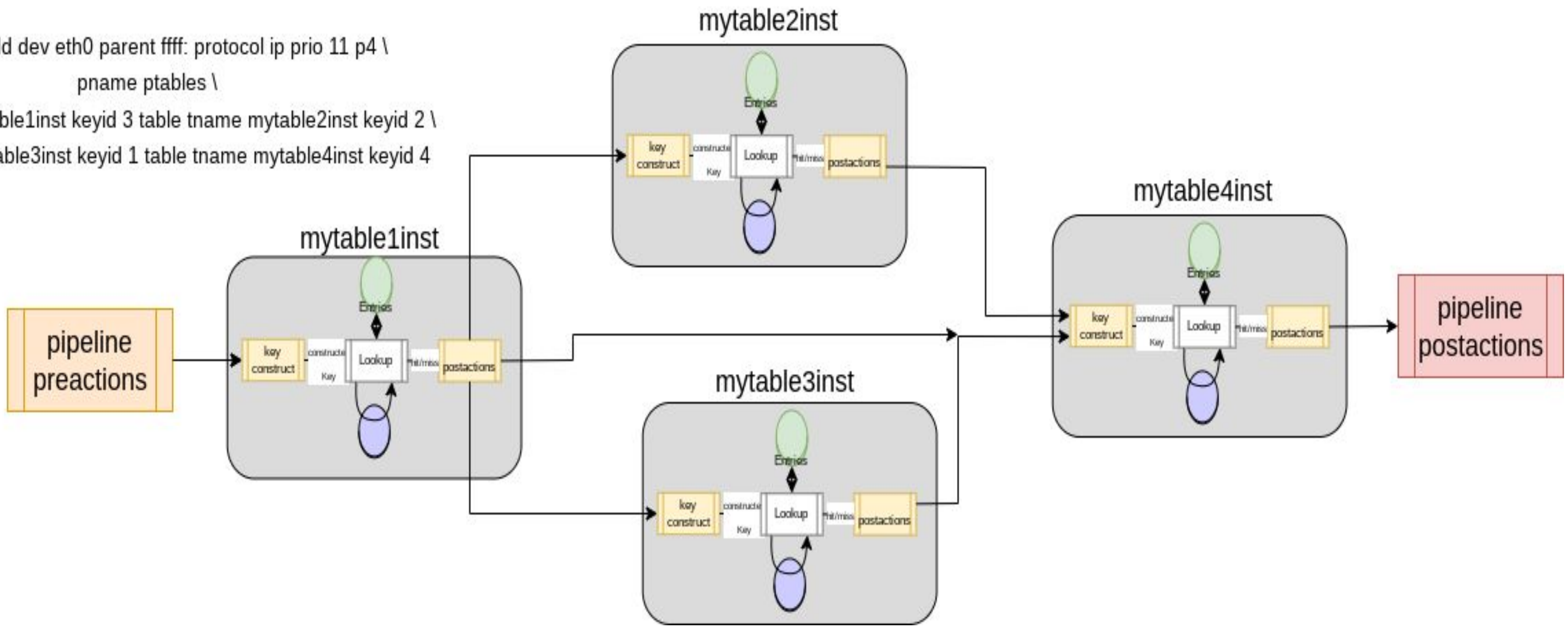
Program Abstraction

```
...
table table1 {
        key = {
                hdr.ipv4.srcAddr: lpm;
        }
        actions = {
                ...
        }
        size = 8192;
        ...
}

table table2 {
        key = {
                hdr.ipv4.dstAddr: lpm;
        }
        actions = {
                ...
        }
        size = 8192;
        ...
}

...
table1.apply()
if (results.hit)
  table2.apply()
...
```



Slightly Complex Program

tc filter add dev eth0 parent ffff: protocol ip prio 11 p4 \
pname ptables \
table tname mytable1inst keyid 3 table tname mytable2inst keyid 2 \
table tname mytable3inst keyid 1 table tname mytable4inst keyid 4



TC Kernel Gaps

- Kernel
  - Action block need revamp to add P4 action selectors
  - Externs - anything that the compiler can see we should be able to handle
- P4
  - Multiple keys
  - Multiple table + pipeline instance
  - Deparser behavioral consequences
    - Inline in SW vs at end of pipeline for HW

Thank You

- Project status: Coding in progress
- Mailing list, to subscribe:
  - send email with subject “subscribe” to: p4tc-discussions@netdevconf.org
- Biweekly meetings (join the mailing list to find out)
- Project is incubating
  - Serious contributors welcome
  - Hope to have packets passing in about 6 months