P4TC

Your Network Program Will Be P4 Scripted

Jamal Hadi Salim
Victor Nogueira
Pedro Tammela
Deb Chatterjee
Anjali Singhai Jain
Evangelos Haleplidis
And Many Others (see Credits)

Introduction

Community Historical Perspective

- Many informal hallway and online discussions (2016)
- Netdev 2.2 (Seoul, 2017)
  - Matty Kadosh, "P4 Offload", https://legacy.netdevconf.info/2.2/slides/salim-tc-workshop04.pdf
  - Prem Jonnalagadda, "Mapping tc to P4", https://legacy.netdevconf.info/2.2/slides/salim-tc-workshop06.pdf
- ONF 5th Workshop(Stanford, 2018)
  - Jamal Hadi Salim, "What P4 Can Learn From Linux Traffic Control", https://opennetworking.org/wp-content/uploads/2020/12/Jamal_Salim.pdf
- First ever P4 TC workshop, Intel, Santa Clara, 2018
  - Many Speakers (Barefoot, Intel, Cumulus, Melanox, Vmware, Mojatatu, and others) https://files.netdevconf.info/d/5aa8c0ca61ea4e96bb46/
- Netdev 0x12 (Montreal, 2018)
  - Antonin Bas and R. Krishnamoorthy, "Instrumenting P4 in the Kernel" https://www.files.netdevconf.info/d/9535fba900604dcd9c93/files/?p=/Instrumenting%20P4%20in%20the%20Linux%20kernel.pdf
- Netdev 0x13 (Prague, 2019)
  - Marian Pritsak and Matty Kadosh, "P4 Compiler Backend for TC", https://legacy.netdevconf.info/0x13/session.html?p4-compiler-backend-for-tc

Community Historical Perspective

- Call To Arms: Netdev 0x14(Virtual Land, 2020)
  - Nick Mckeown
    - Creating an End-to-End Programming Model for Packet Forwarding
      - https://legacy.netdevconf.info/0x14/session.html?keynote-mckeown

- And finally:..... Cut… Action! (2022)

P4 TC: Community Effort Going Forward

- P4TC is currently funded by Intel
  - Goal is to build up on the P4 ecosystem to grow network programmability
- P4TC is NOT an Intel-only Project
  - All Vendors and other community members welcome
    - Can participate in coding if want to contribute
    - There's a workshop in this conference
    - There's a periodic community meeting - come to the workshop for more details

What Is P4TC?

A ***Scriptable*** **Hardware offload of P4 MAT and Control**

- Both Control and Kernel Datapath are scripted (not compiled)

Caveat: P4 is a subset of what P4TC provides

P4 TC: Why P4?

- P4 has gained industry-wide acceptance as a datapath language
  - Currently _the_ Lingua Franca for describing hardware datapaths
    - Large consumers of NICs require at minimal P4 for datapath behavioral description if not implementation
  - To Each, Their Itch
    - Conway's Law: Organizations model their datapath based on their needs
    - Ossification challenges: It's not just about traditional TCP/IP
      - New+Novel protocols (see Calc example)
- Emergence of DPU/IPU NICs and higher IO rates
  - High Performance Network Programmability more accessible to the masses
  - Moore's law is done

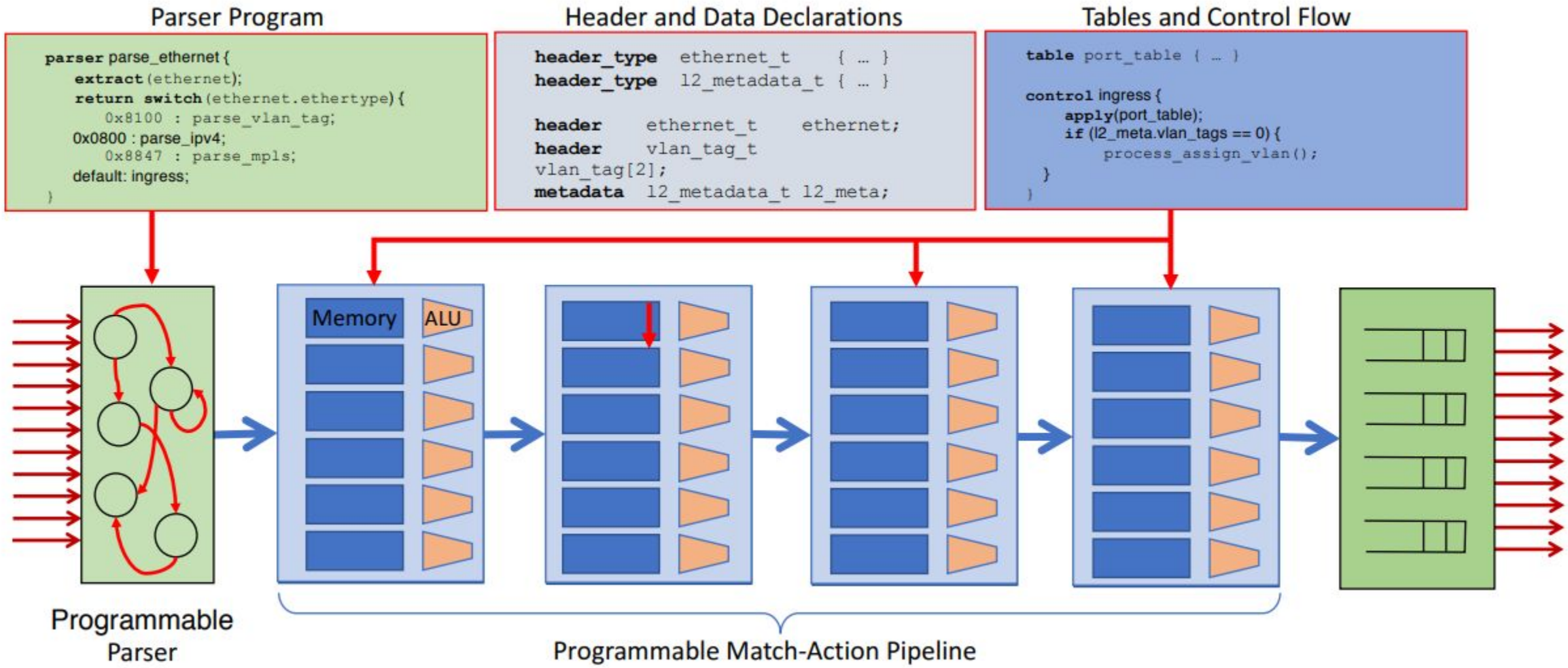
We need to make it real for the Linux Kernel

Show Me The Code!

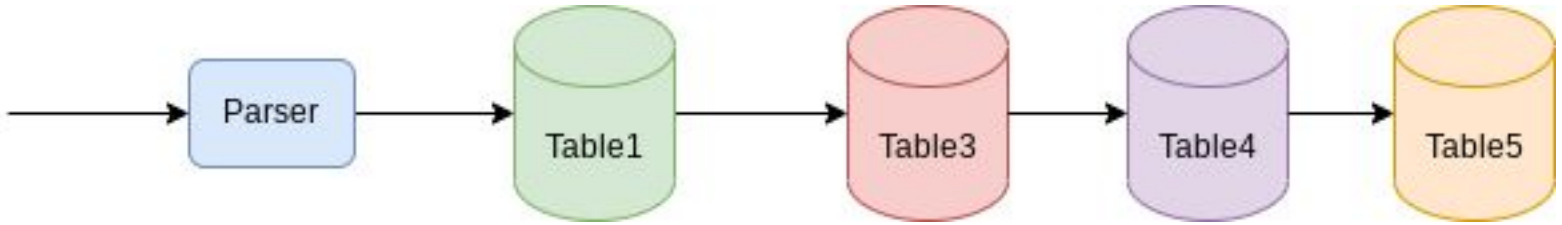
- Kernel Code:
  - https://github.com/p4tc-dev/linux-p4tc-pub
- iproute2 code:
  - https://github.com/p4tc-dev/iproute2-p4tc-pub
- project page:
  - https://p4tc.dev

High Level TC Perspective

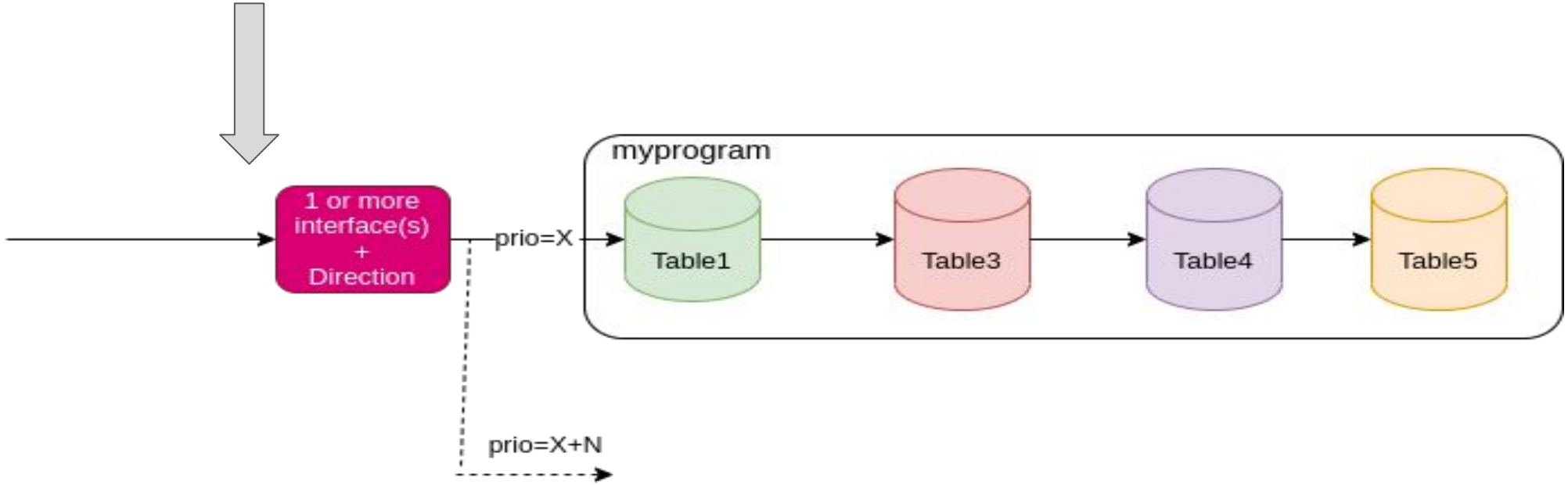
TC Perspective: What Is A P4 Program?



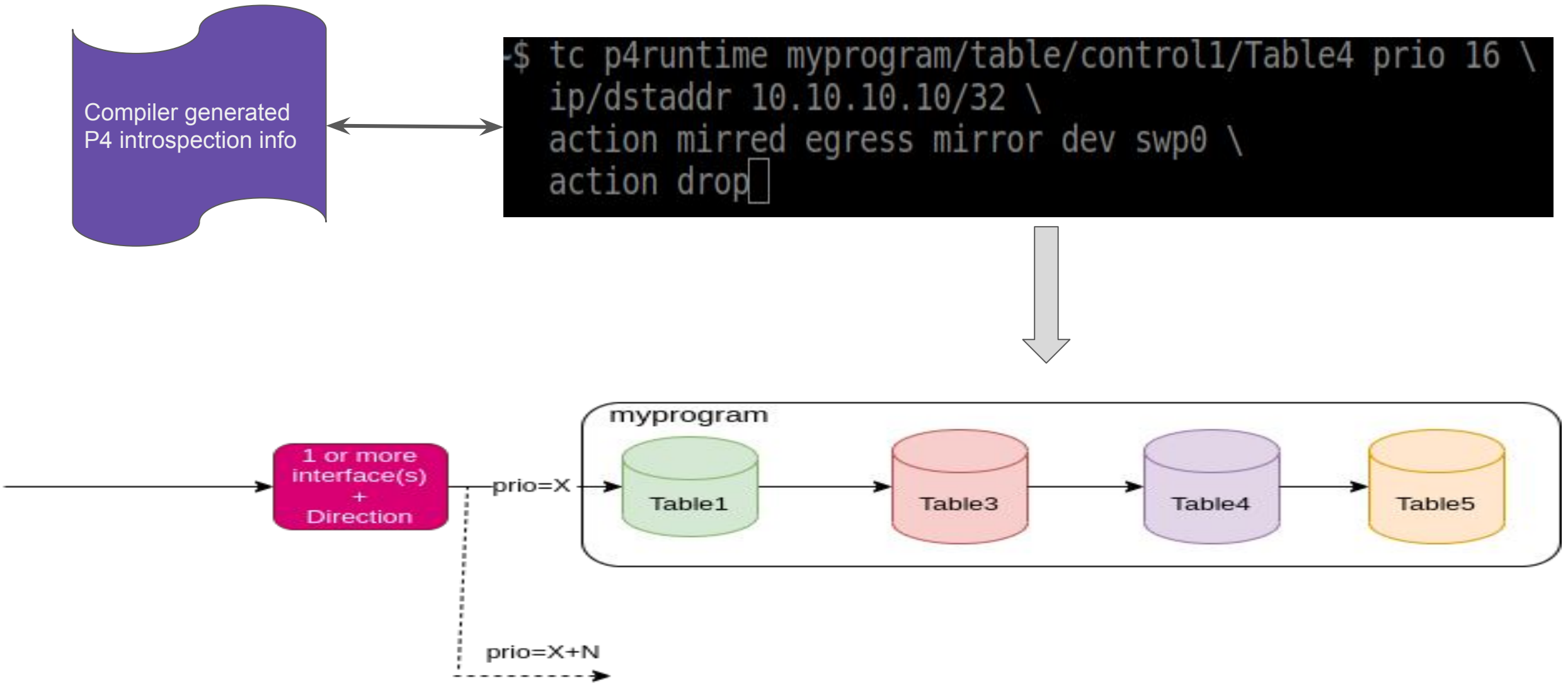
P4 Program Loading/Instantiation



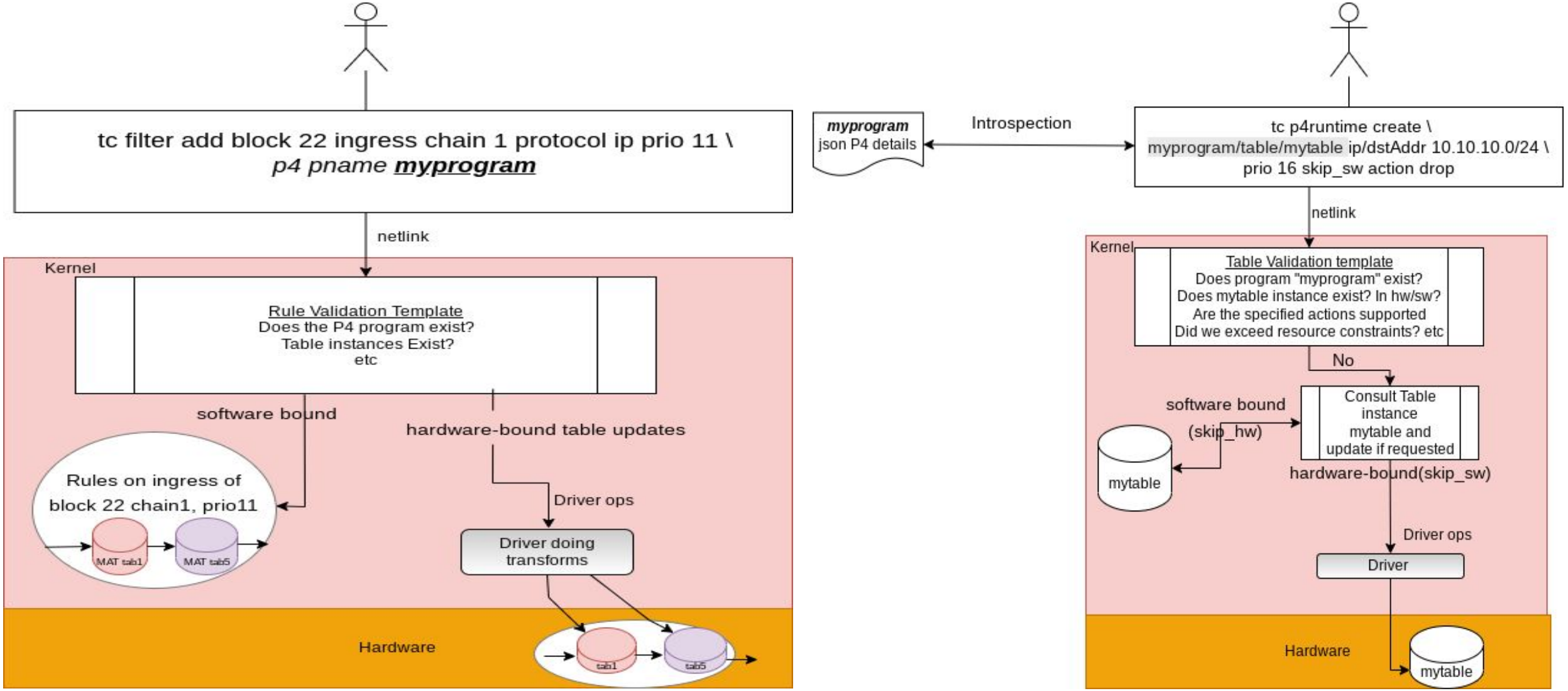

```
tc filter add dev eth0 <egress|ingress> protocol ip prio 1 p4 \
pname myprogram
```



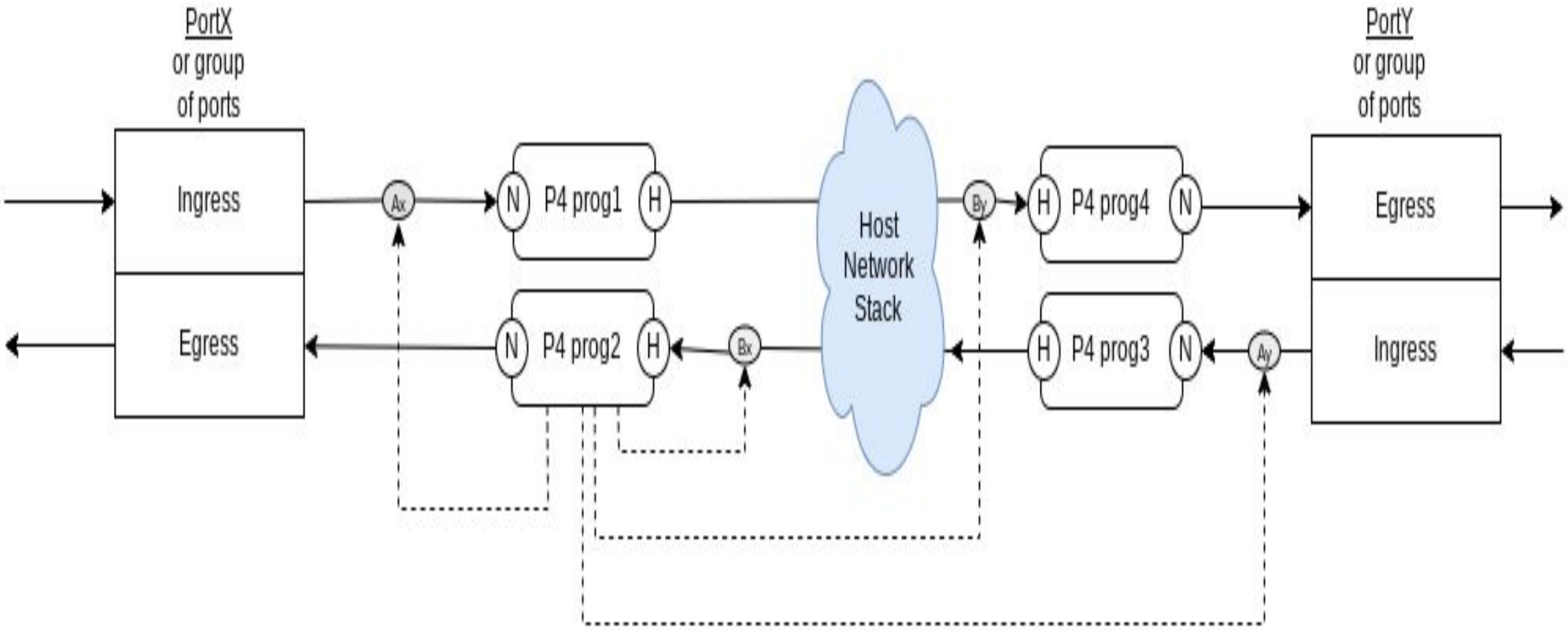
Runtime: P4 Match Action Table Control



Runtime: P4 Program Instantiation



TC Perspective: P4 Program Hooks



General Requirements, Motivations And Goals

So... how is this better than KDE?

Attributed to Rusty Russell

Who attributes it to Cort Dougan

s/**KDE**/[rust/ebpf/dpdk/vpp/ovs]/g

- All have P4 backends
- p4tc focus is hw offload with sw kernel equivalence
  - Same spirit as flower, u32, matchall, etc

P4 TC: Goals

- Offloading entirely to P4-capable hardware with *skip_sw*
- Running entirely in software infrastructures (VMs, baremetal, containers) via *skip_hw*
- Functional Equivalence in software to what is in hardware
- Hardware Independence for offload
  - Our abstraction is the P4 Logical definition
- Independence from changing any kernel or user space code with introduction of a new P4 program (achieved via "scriptability")

P4 TC: Goals

- *For Free* by virtue of using TC infra
  - Running multiple independent P4 programs across multiple independent hardware and/or software (using tc filter infrastructure)
  - Split datapath setup
    - Multi-tier programs
    - Part pipeline in h/w and part in s/w
    - part table in h/w and part in s/w

What Is Scriptability?

Think tc u32 classifier

- Parse and match on arbitrary packet headers with zero kernel or user space changes

Think pedit tc action

- Edit arbitrary packet headers with zero kernel or user space changes

Think a programmable tc skbedit

- RW arbitrary packet metadata with zero kernel or user space changes

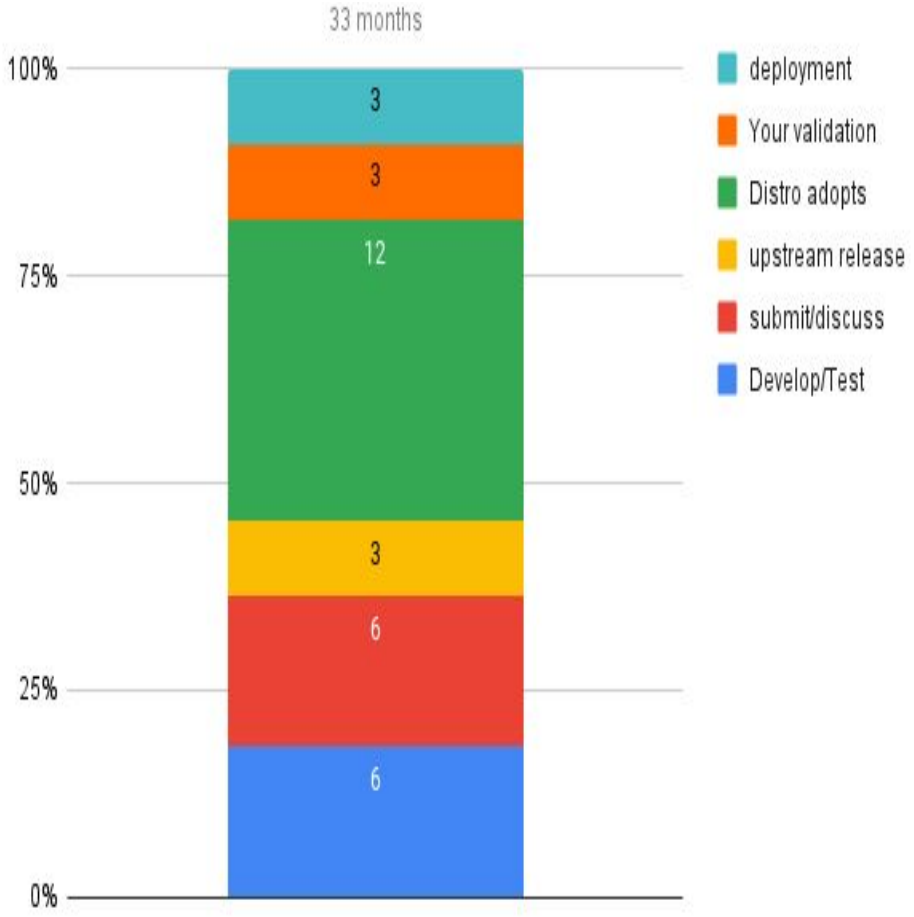
Next:

…Think of all those designed from day 1 with intention to define arbitrary datapath and associated processing as defined by P4

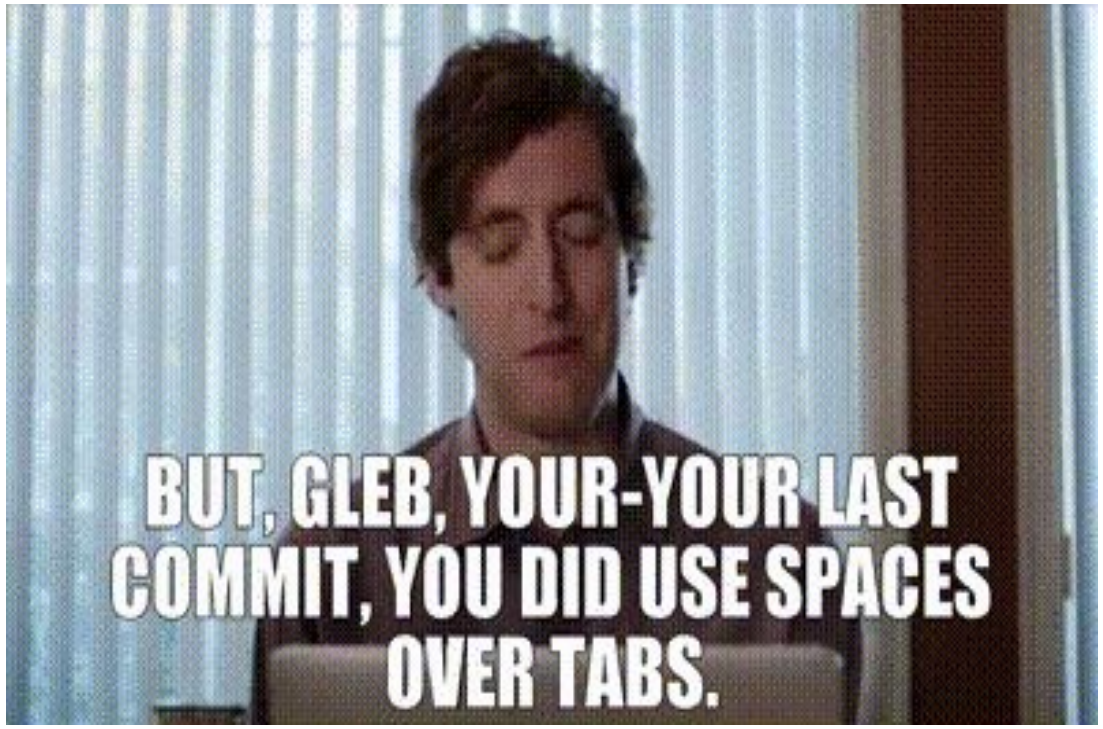
⇒ That is what P4TC is about….

Why Scriptability?

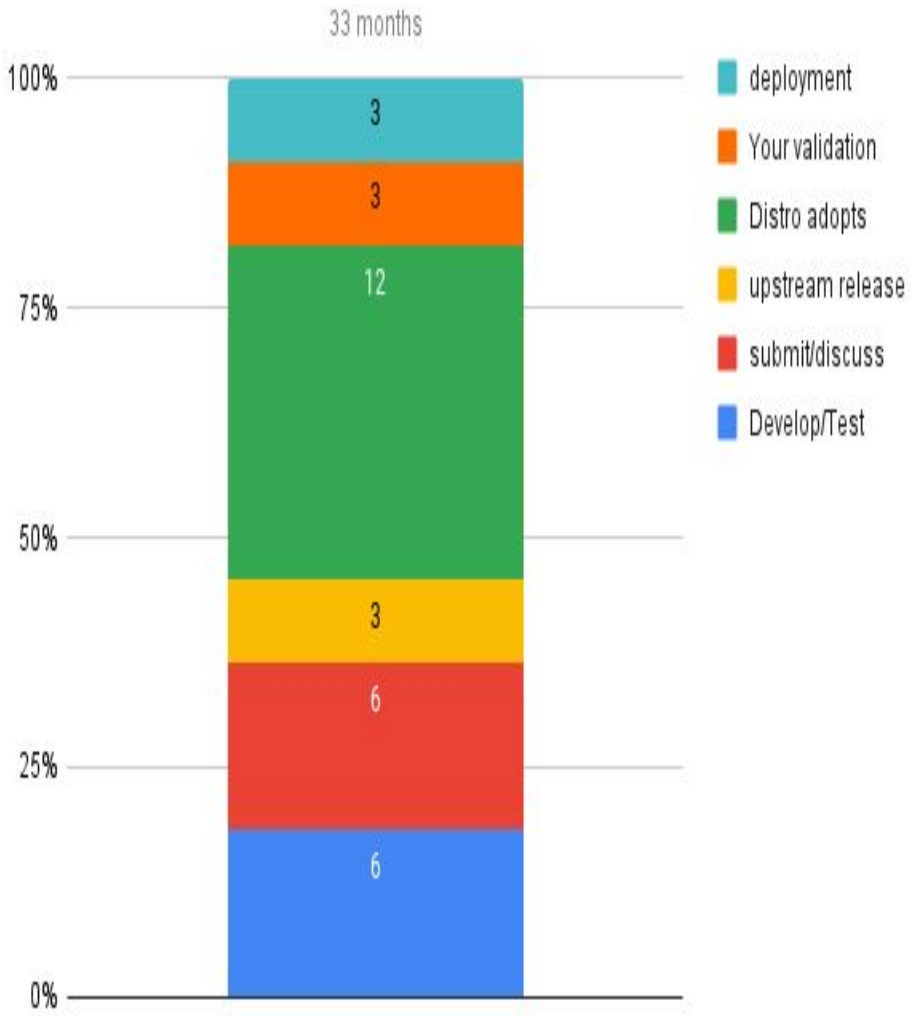
Time Cost of Adding offload of a basic header match



- The tragedy of current kernel offloads: $$$$$$
  - Process takes too long
    - Think of just that one more header field for flower



Why Scriptability?



- The tragedy of current kernel offloads: $$$$$$
  - Process takes too long
    - Solution? keep around a team of domain experts (kernel, etc)
- Every network is different
  - meaning different datapath requirements
    - Gone are the days of selling black boxes
    - Feature velocity as a requirement
    - Consistent API
- Compiling is restricted for some DC environments

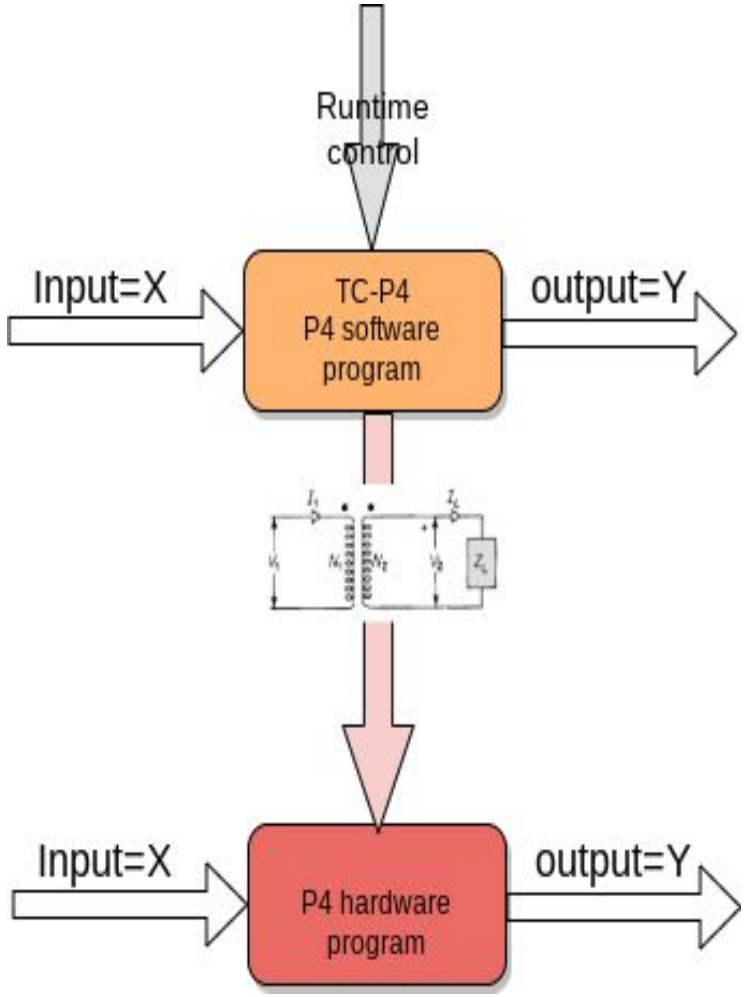
Upstream Sayeth: Thou Shalt Have A Software Twin

Upstream Requirements

- SW and HW are functionally equivalent
  - **In SW given choice between performance and equivalence choose equivalence**
- 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
  - Using exactly the same code and interfaces in s/w as in h/w
- Use P4 in new infrastructure experiments
  - Accelerate when needed



Target Audience

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

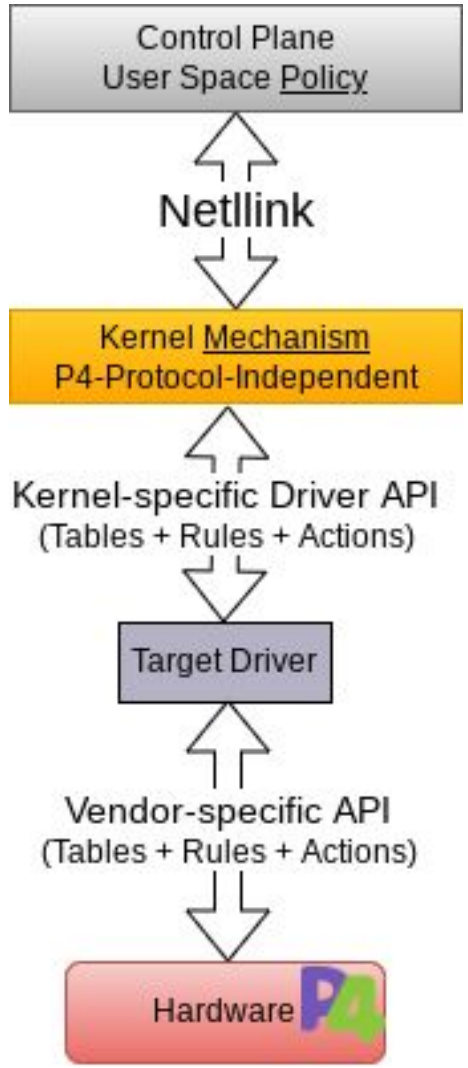
Core Design Principle: Templates And Scriptability

Template-driven architecture

- map P4 Program to mechanisms in the kernel
- user space(control) and kernel(datapath) independence

A P4 program is constructed via template scripts from user space

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

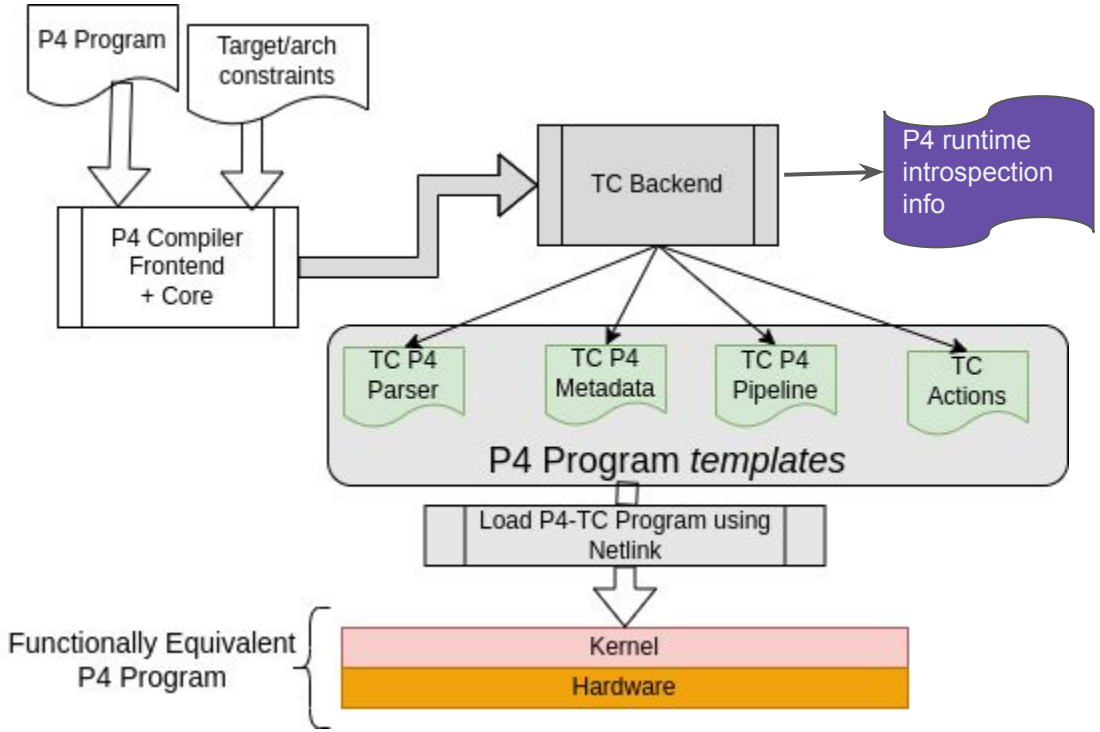


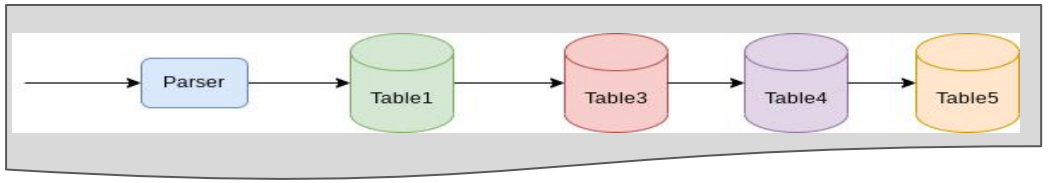
General Design Goals

- 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

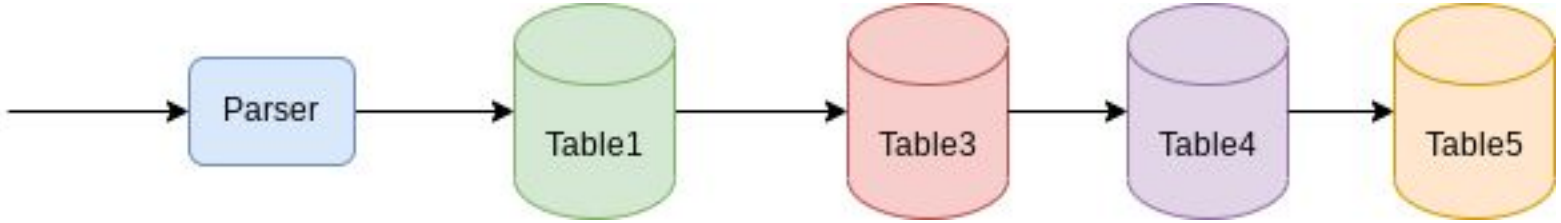




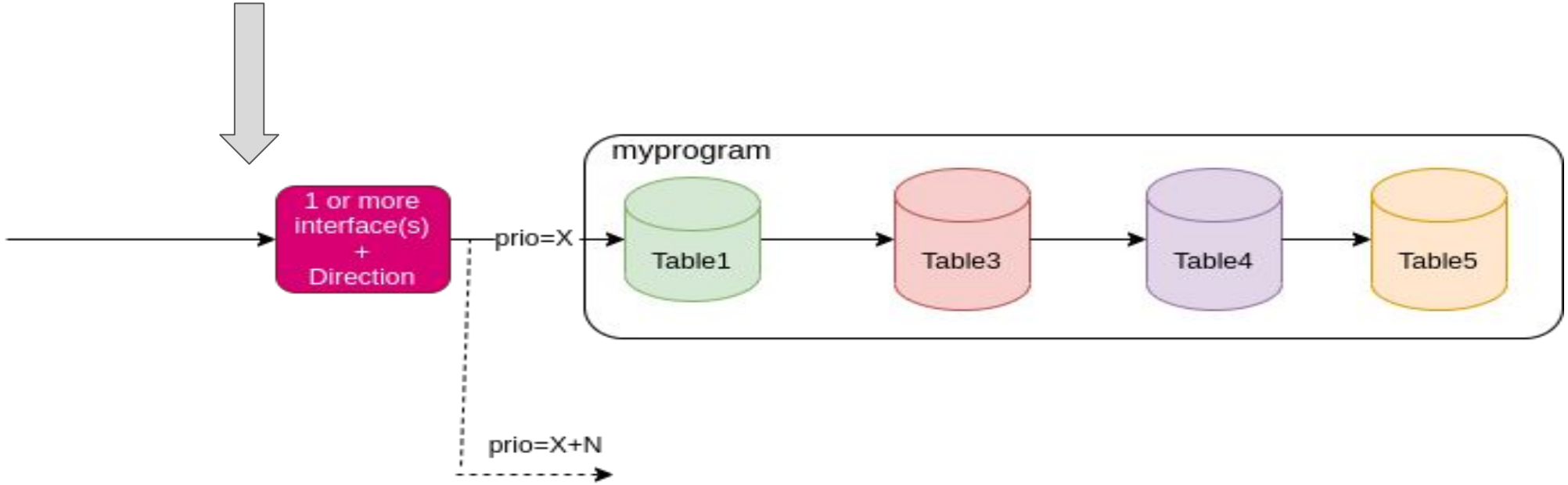
P4 Dev approach
1. Author writes the P4 program
2. Build P4 program (using a compiler) targeting TC and/or hardware
3. Author/dev executes the tc template scripts to “install P4 program”

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

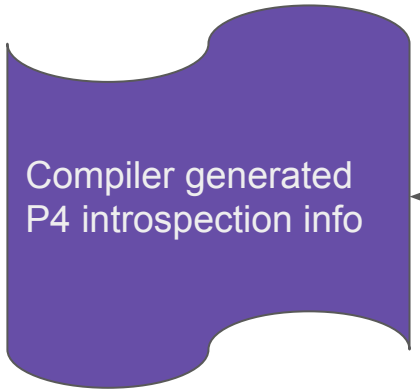
Runtime: P4 Program Instantiation




```
tc filter add dev eth0 <egress|ingress> protocol ip prio 1 p4 \
pname myprogram
```

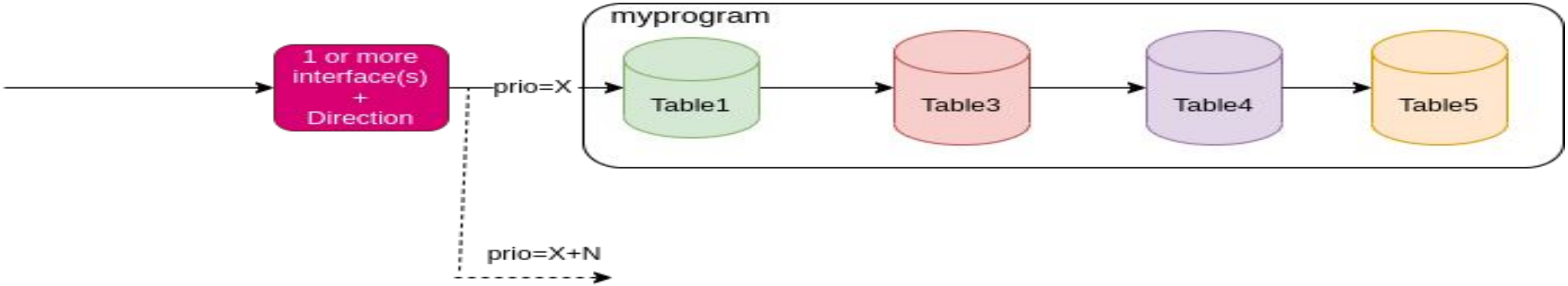


Runtime: P4 Match Action Table Control




```
$ tc p4runtime myprogram/table/control1/Table4 prio 16 \
  ip/dstaddr 10.10.10.10/32 \
  action mirred egress mirror dev swp0 \
  action drop
```

Note: no mention of "dev" in the command line



P4 Objects Kernel Abstractions

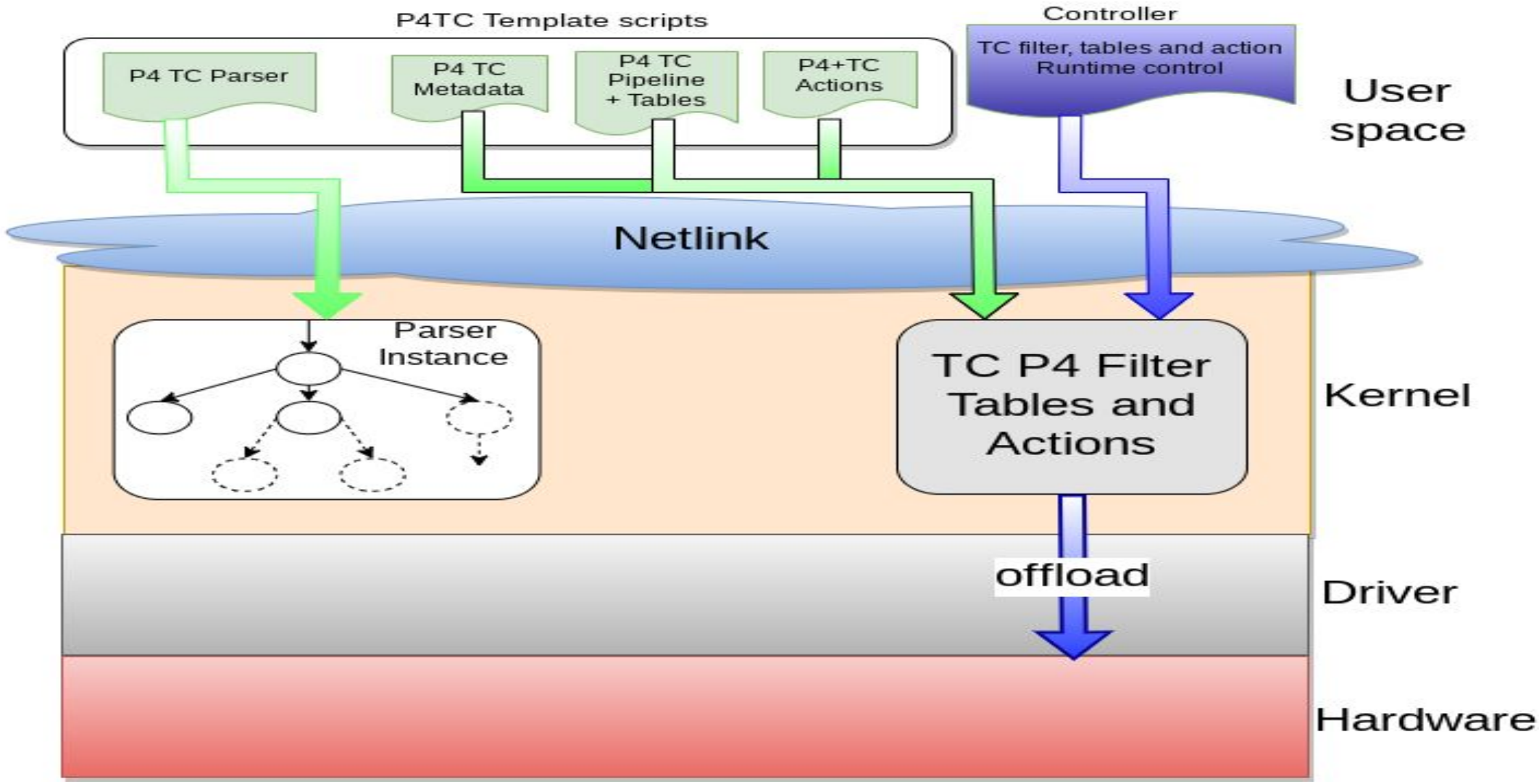
P4 Objects In The Kernel

- Pipeline
- Parser
- Header Fields
- Actions (extended to add dynamic actions)
- Tables and Keys
- Metadata
- Externs
- Registers
- Note: Counters and meters are supported in TC already (25 years old!)
  - We are looking at reusing them

P4 Kernel/User Interfaces

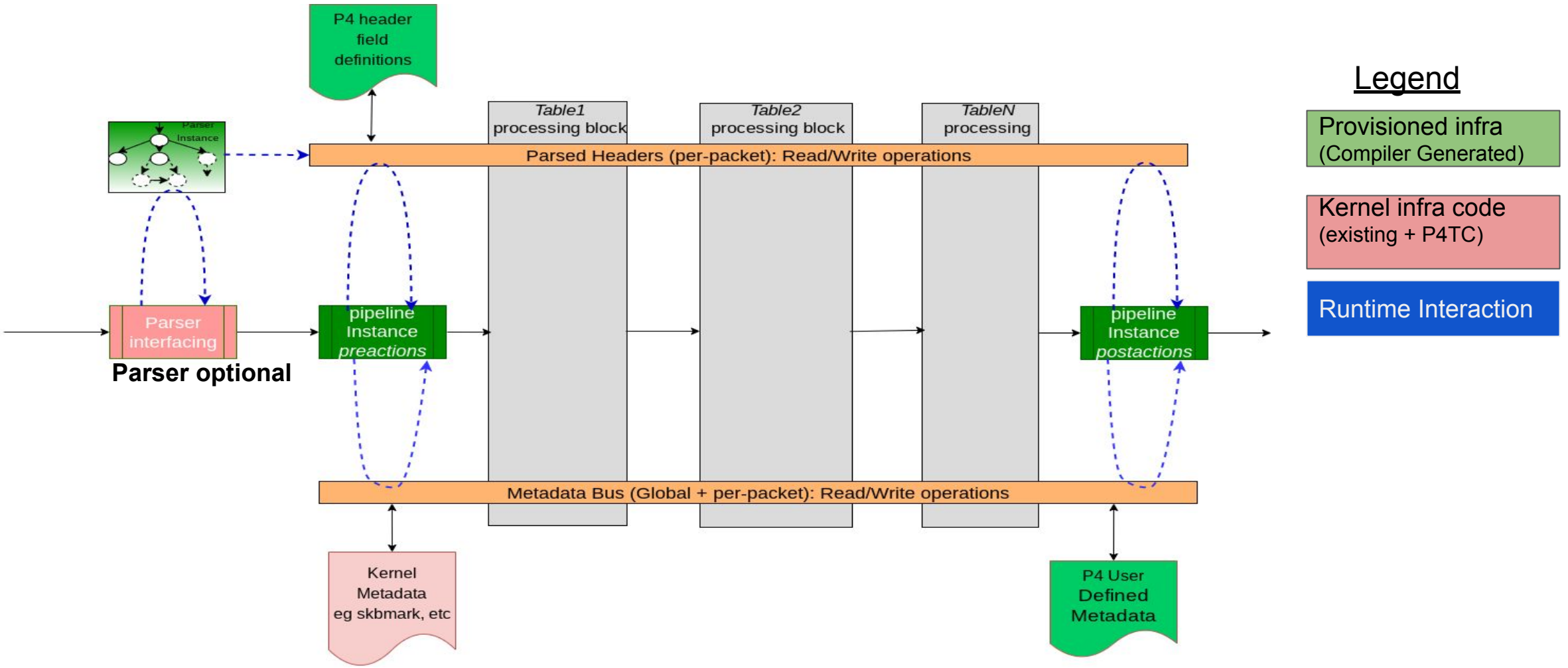
- tc p4template ..<CRUD on constructing P4 program>
- tc p4runtime .. <CRUD on P4 program/pipeline object>
  - Will likely change to “tc filter … p4” to reduce number of options
- tc filter add on .. p4 *myp4programname*

Compiler Generated Template Hooks Vs Runtime

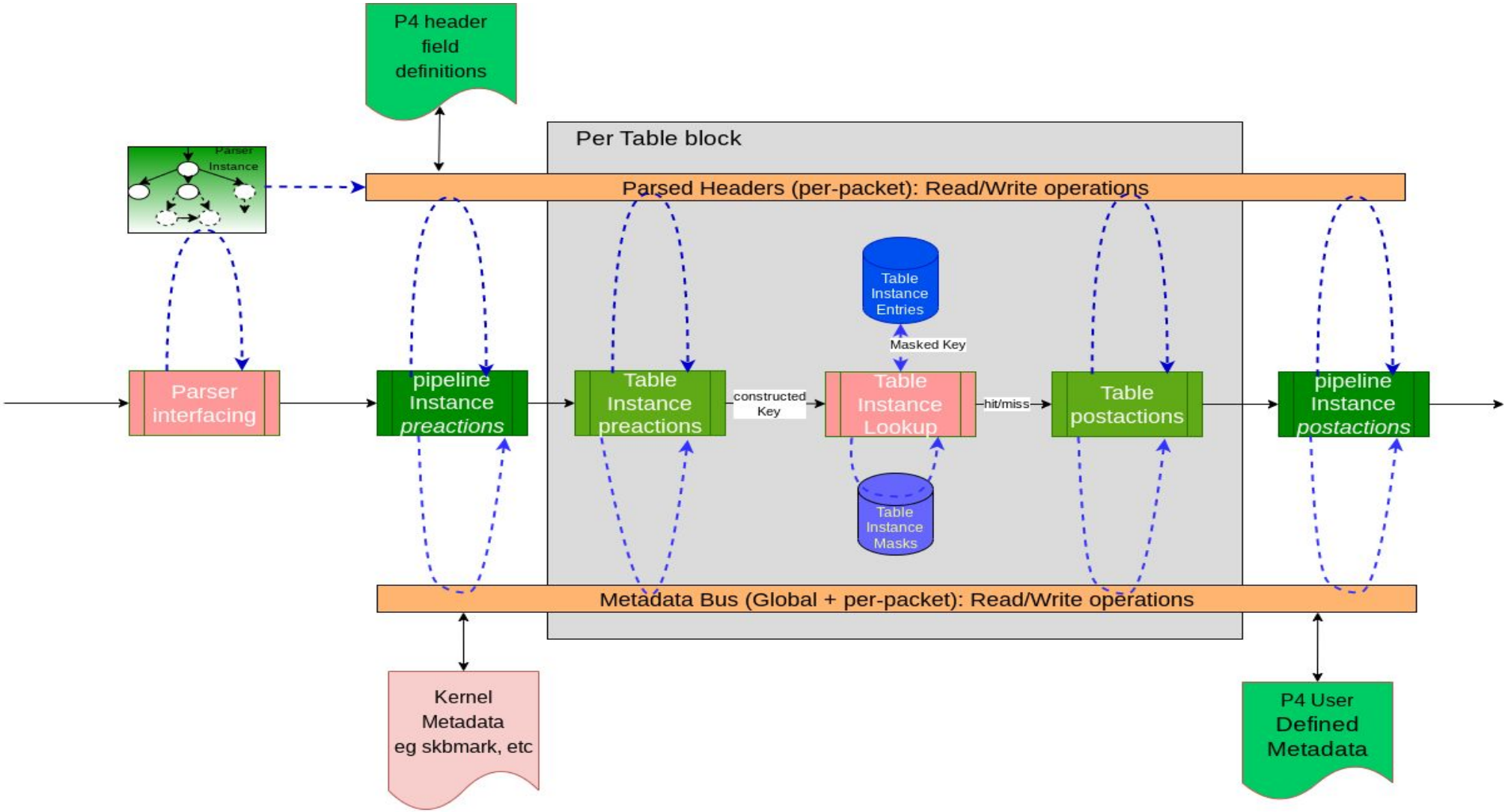


Runtime View: Anatomy of TC P4 filter

Aka Pipeline "instance"



Runtime View: Single Table Pipeline



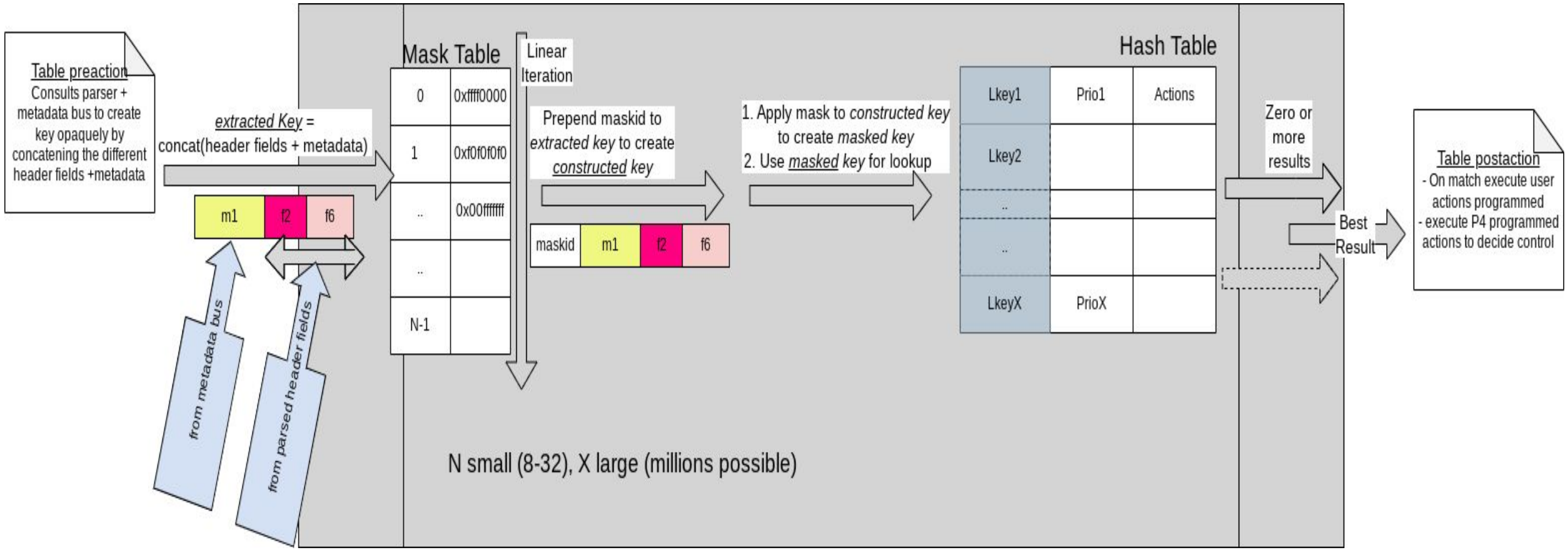
Legend

Provisioned infra (Compiler Generated)

Kernel infra code (existing + P4TC)

Runtime data + control

Table Abstraction: Algorithmic TCAM



Actions

- TC predefined/existing Actions
- P4 user defined actions
  - Observation is that a lot of action behavior is repeatable, example
    - Read/write of metadata and packet headers (essentially u32/skbedit/pedit behavior)
    - Arithmetic (eg decr ttl)
  - The templating phase defines the combination of these operations that are required for a specific action
    - Compiler takes care of this..

Templating User Defined Actions

```
action ipv4_forward(bit<48> dstAddr,bit<9> port)
{
    standard_metadata.egress_spec = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}
```

```
tc p4template create action/myprog1/mycontrol/ipv4_forward \
    param dstAddr type bit48 param port type bit9 \
    cmd set standard_metadata/egress_spec param/port \
    cmd set hdr/ethernet/srcAddr hdr/ethernet/dstAddr \
    cmd set hdr/ethernet/dstAddr param/dstAddr \
    cmd decr hdr/ipv4/ttl
```

Using User Defined Actions

Method1: Direct action (by value)

```
~$ tc p4runtime create myprog/table/mycontrol/table1 srcaddr 10.10.10.0/24

  prio 15 \
  action ipv4_forward dstaddr AA:BB:CC:DD:EE:01 port 9
```

Method2: Indirect action (by reference)

```
tc actions add \
action ipv4_forward dstAddr AA:BB:CC:DD:EE:01 port 9 \
index 23
```

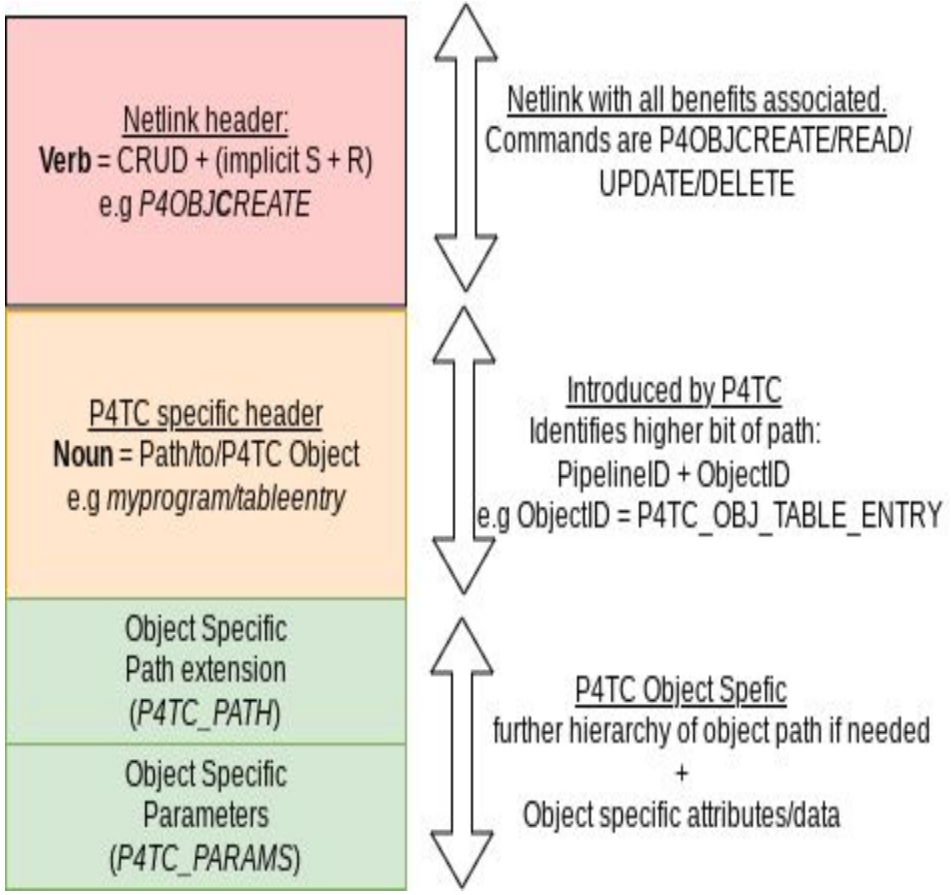
```
$ tc p4runtime create myprog/table/mycontrol/table1 srcaddr 10.10.10.0/24 \
  prio 15 \
  action ipv4_forward index 23
```

Externs

- A source of portability challenges within P4 spec
  - Ex: See the small description of PNA mirror/send_to_port in the workshop
- We will implement all specified externs within PSA and PNA
- Add a few more Linux kernel Helpers (linux externs)
- Teach the compiler to sort it out
- We still may not solve all the issues

Control Plane Interaction

Control Plane Runtime CRUD Interface



Goal: Very High throughput and Low Latency interface

<VERB> < <NOUN> [OPTIONAL DATA] >+

single entry:

```
tc p4runtime get ptables/table/control1/mytable \
    ip/dstaddr 1.1.1.1/32
```

vs dump the whole table

```
tc p4runtime get ptables/table/control1/mytable
```

single entry:

```
tc p4runtime create ptables/table/control1/mytable \
   ip/dstaddr 10.10.10.0/24 prio 16 action drop
```

Data

vs batch:

```
tc p4runtime create ptables/table/control1/mytable \
   ip/dstaddr 10.10.10.0/24 prio 16 action drop \
   ip/dstaddr 1.1.1.1/32 prio 32 action drop \
   ip/dstaddr 8.8.8.8/32 prio 64 action drop
```

Runtime: Offload Hardcoding Be Gone

```
struct flow_rule {
        struct flow_match       match;
        struct flow_action      action;
};
```

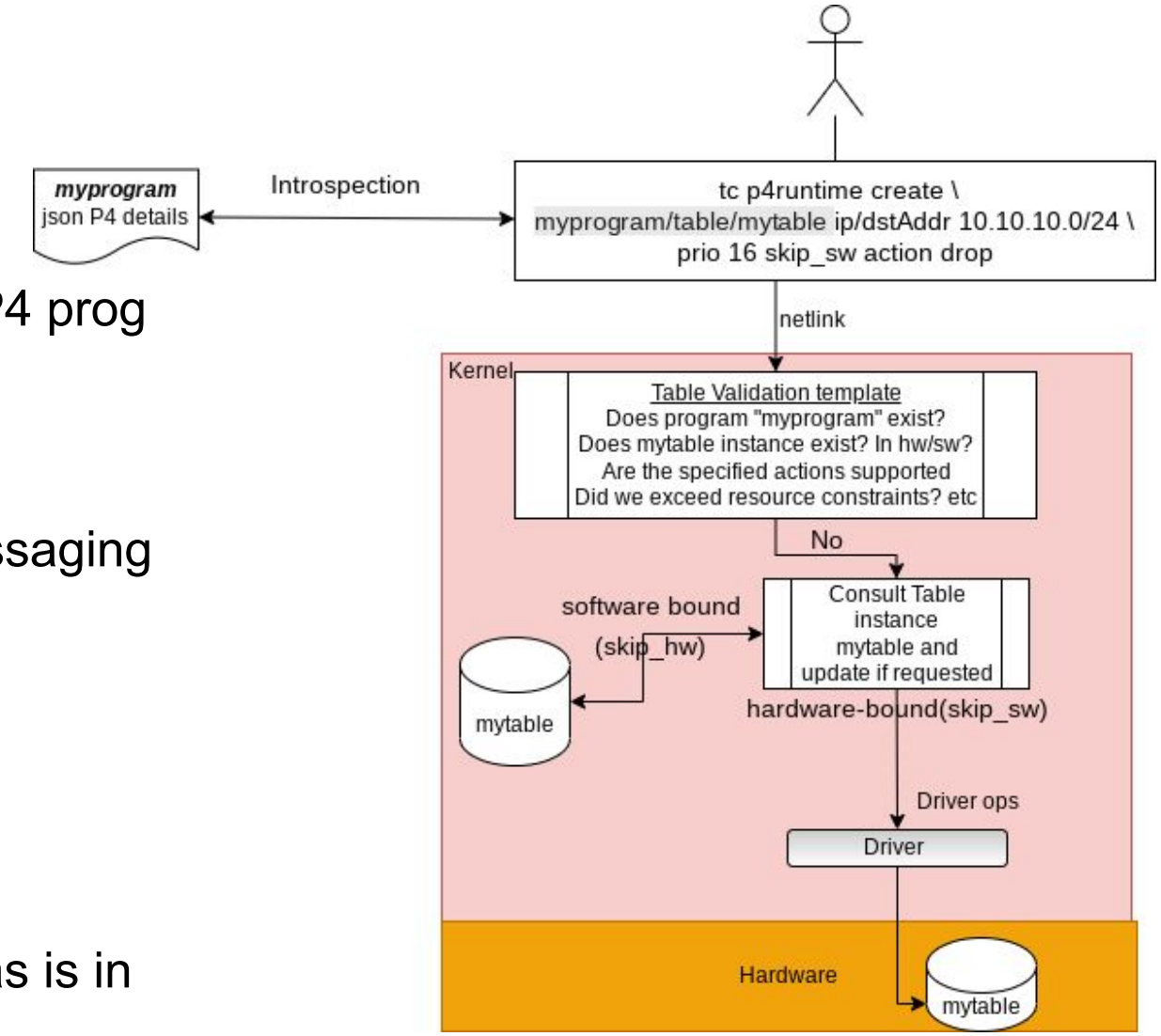
```
struct flow_match {
        struct flow_dissector   *dissector;
        void                    *mask;
        void                    *key;
};
```

Runtime: Offload Hardcoding Be Gone

```
struct flow_action_entry {
        enum flow_action_id             id;
        u32                             hw_index;
        enum flow_action_hw_stats       hw_stats;
        action_destr                    destructor;
        void                            *destructor_priv;
        union {
                u32                     chain_index;    /* FLOW_ACTION_GOTO */
                struct net_device       *dev;           /* FLOW_ACTION_REDIRECT */
                struct {                                /* FLOW_ACTION_VLAN */
                        u16             vid;
                        __be16          proto;
                        u8              prio;
                } vlan;
                struct {                                /* FLOW_ACTION_VLAN_PUSH_ETH */
                        unsigned char dst[ETH_ALEN];
                        unsigned char src[ETH_ALEN];
                } vlan_push_eth;
                struct {                                /* FLOW_ACTION_MANGLE */
                                                        /* FLOW_ACTION_ADD */
                        enum flow_action_mangle_base htype;
                        u32             offset;
                        u32             mask;
                        u32             val;
                } mangle;
                struct ip_tunnel_info   *tunnel;        /* FLOW_ACTION_TUNNEL_ENCAP */
                u32                     csum_flags;     /* FLOW_ACTION_CSUM */
                u32                     mark;           /* FLOW_ACTION_MARK */
                u16                     ptype;          /* FLOW_ACTION_PTYPE */
                u32                     priority;       /* FLOW_ACTION_PRIORITY */
                struct {                                /* FLOW_ACTION_QUEUE */
                        u32             ctx;
```

```
enum flow_action_id {
        FLOW_ACTION_ACCEPT                = 0,
        FLOW_ACTION_DROP,
        FLOW_ACTION_TRAP,
        FLOW_ACTION_GOTO,
        FLOW_ACTION_REDIRECT,
        FLOW_ACTION_MIRRED,
        FLOW_ACTION_REDIRECT_INGRESS,
        FLOW_ACTION_MIRRED_INGRESS,
        FLOW_ACTION_VLAN_PUSH,
        FLOW_ACTION_VLAN_POP,
        FLOW_ACTION_VLAN_MANGLE,
        FLOW_ACTION_TUNNEL_ENCAP,
        FLOW_ACTION_TUNNEL_DECAP,
        FLOW_ACTION_MANGLE,
        FLOW_ACTION_ADD,
        FLOW_ACTION_CSUM,
        FLOW_ACTION_MARK,
        FLOW_ACTION_PTYPE,
        FLOW_ACTION_PRIORITY,
        FLOW_ACTION_WAKE,
        FLOW_ACTION_QUEUE,
        FLOW_ACTION_SAMPLE,
        FLOW_ACTION_POLICE,
        FLOW_ACTION_CT,
        FLOW_ACTION_CT_METADATA,
        FLOW_ACTION_MPLS_PUSH,
        FLOW_ACTION_MPLS_POP,
        FLOW_ACTION_MPLS_MANGLE,
        FLOW_ACTION_GATE,
        FLOW_ACTION_PPPOE_PUSH,
        FLOW_ACTION_JUMP
```

Runtime: HW Object Mapping



P4 Object IDs are created by compiler based on P4 prog
- Name to ID translation per generated json file
- These IDs are passthrough all the way to HW

iproute2::tc uses them and encodes in netlink messaging
- human readable Name to ID translation via introspection

Kernel code uses them to ID objects

Driver has the same IDs

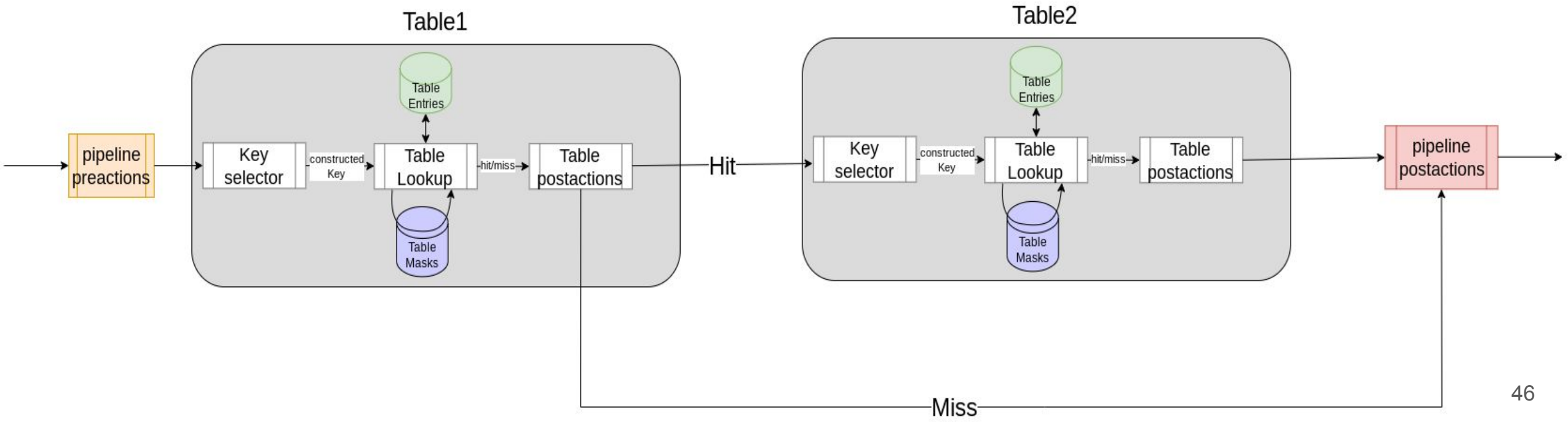
no need for hardcoding IDs in the kernel as is in TC offload today

Program Abstraction

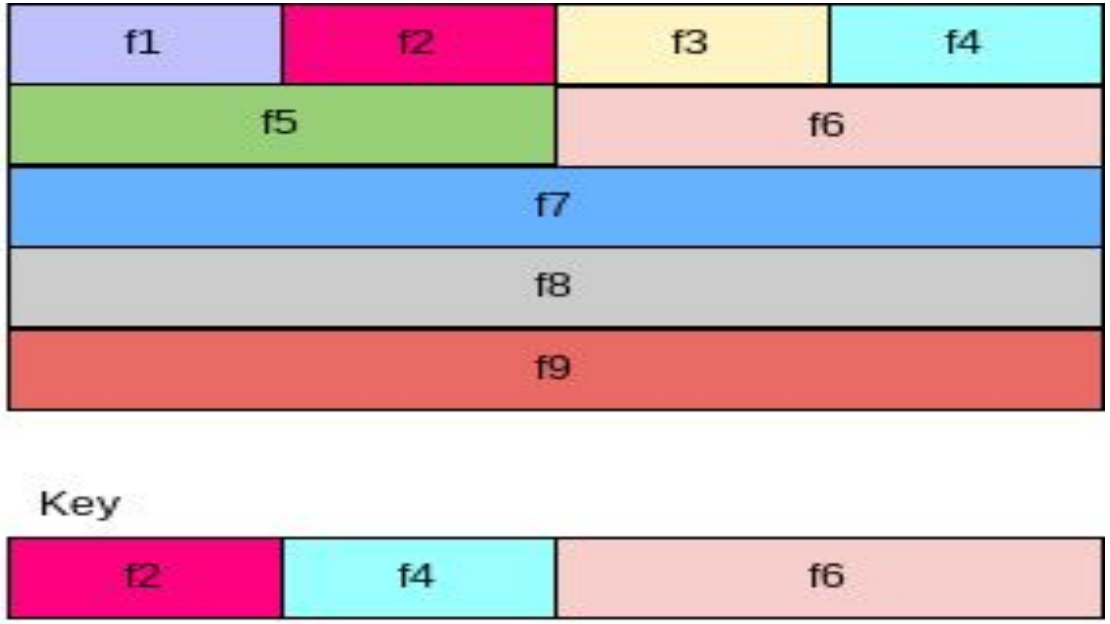
```
...
table table1 {
        key = {
                hdr.ipv4.srcAddr: lpm;
        }
        actions = {
                ...
        }
        size = 8192;
        ...
}

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

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



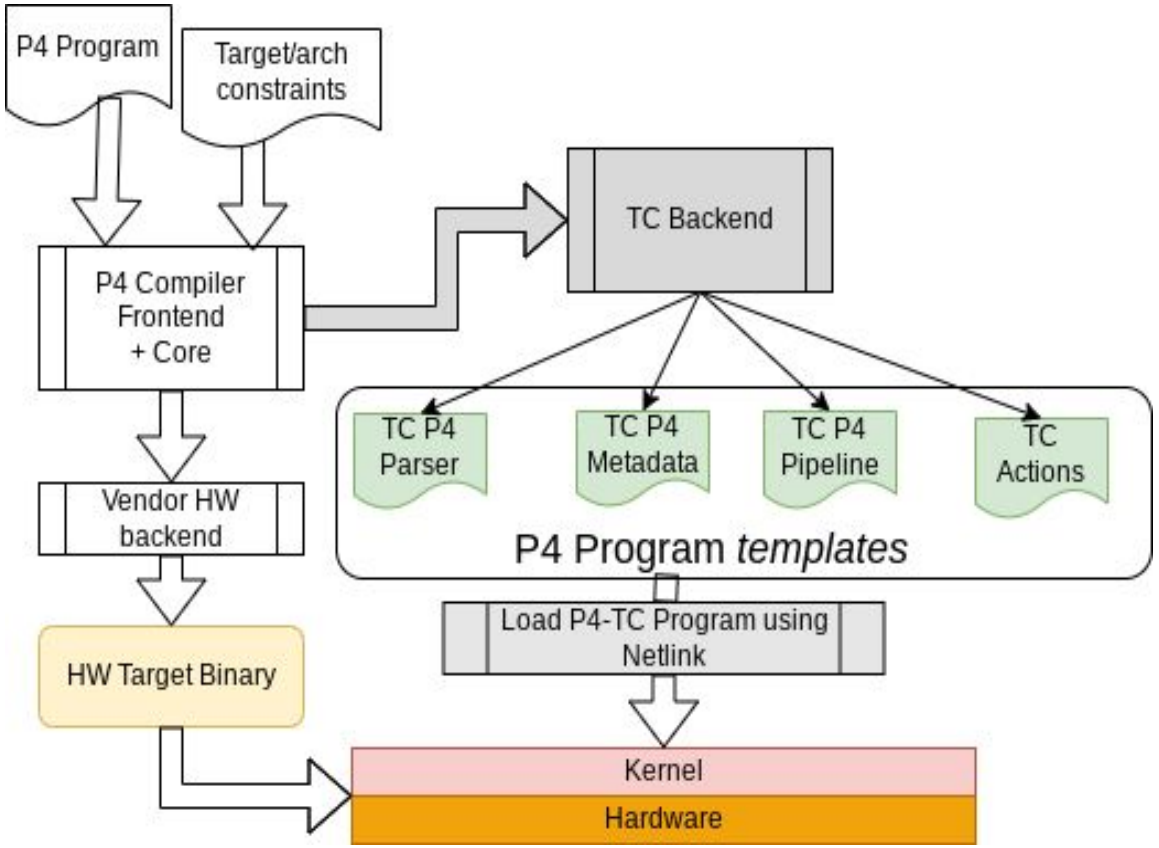
Example Headers for An In-network-computing Calculator



Sample Action
Read op from f5 (+, -, etc),
Operand A from f7,
Operand B from f8
Write A Op B into f9

Offload: To Devlink Or Not

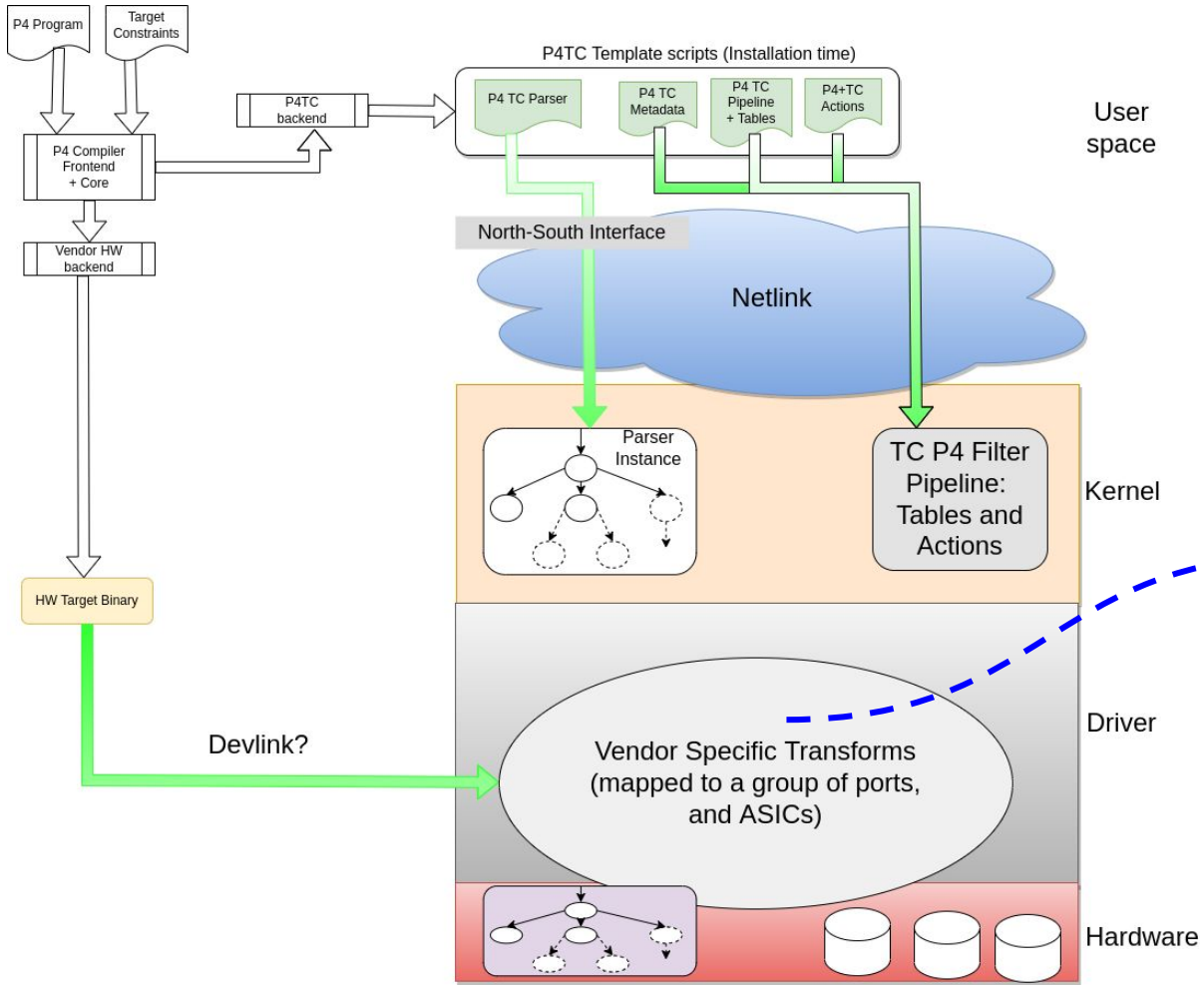
Workflow: Target Binary Loading



Sideband Loading vs Kernel syscall Direct?

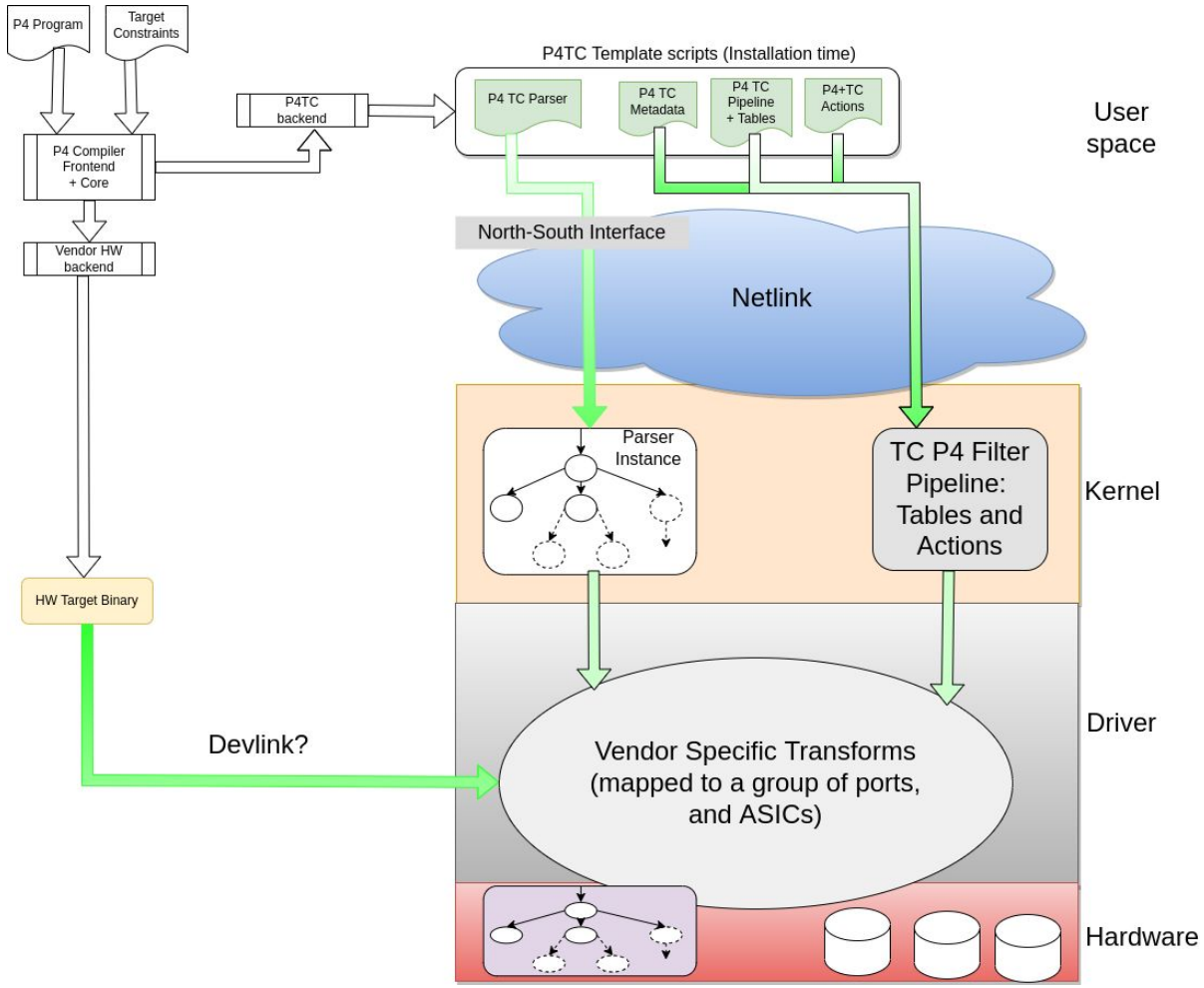
Ongoing discussions (come to workshop)

Model 1: Separate Loading



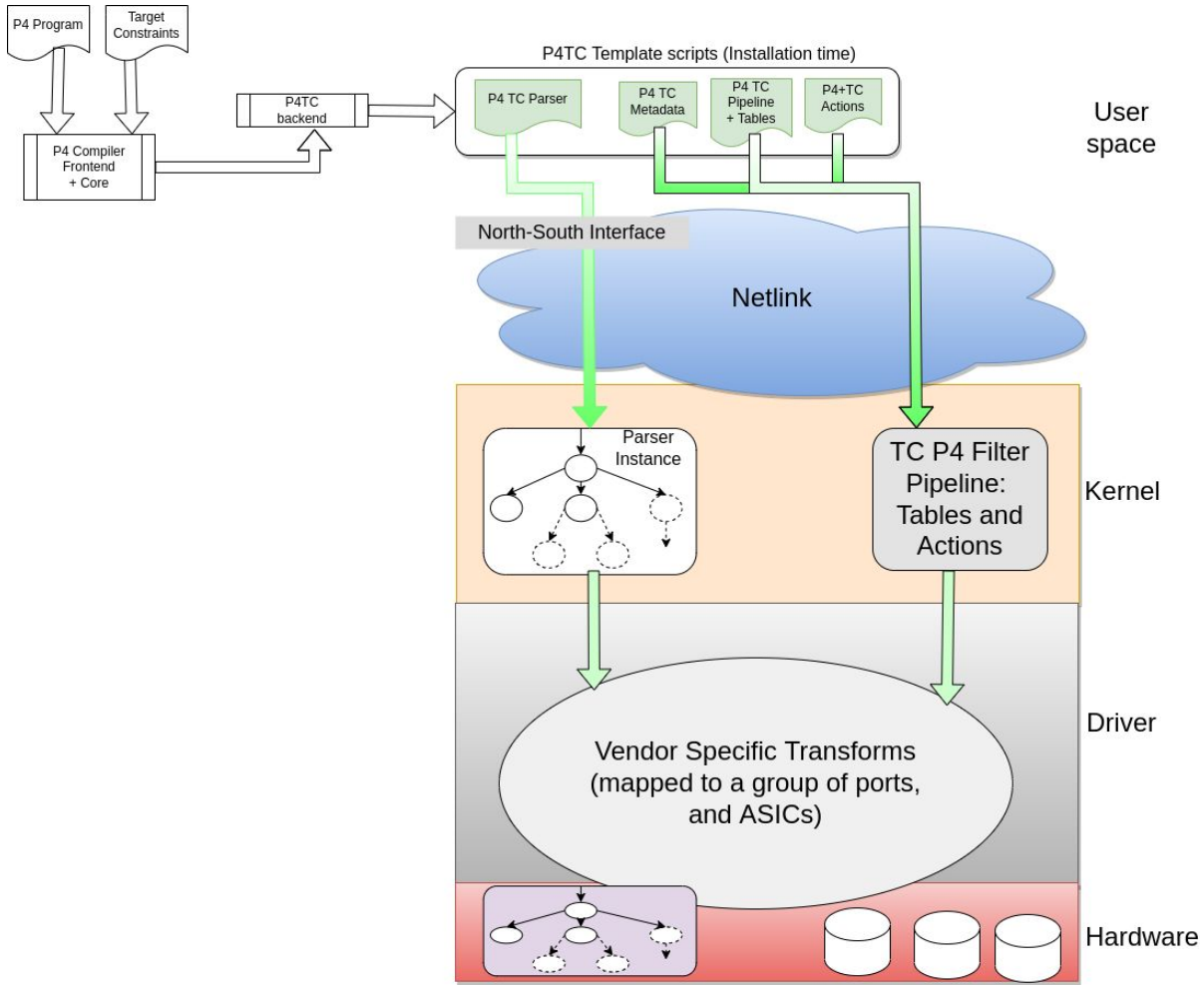
- HW-equivalent loaded via sideband
  - Package includes one or more of
    - Binary
    - Extra configuration for vendor
- SW-equivalent install time verifies that the two programs are the same
- Vendor driver is responsible for "*transforming*" P4 abstractions
  - Table reordering, merge, sort, etc
  - LPM, TCAM, SRAM etc
  - P4 pipeline to hardware pipeline mapping
    - RMT vs DRMT
    - Mesh processing/cross-bar

Model 2: Hybrid Loading



- Parts of HW-equivalent loaded via direct system calls in conjunction with SW-equivalent
- Parts loaded on sideband
- Driver handles P4 transforms

Model 3: Joint Loading



- Both HW-equivalent and SW-equivalent loaded via direct system calls
  - Assumes all objects are <u>provisionable</u> in hw
    - Consensus so far says: This is IMpossible
- Driver transforms P4 abstraction

Credits

- Balachander Sambasivam
- Sosutha Sethuramapandian
- Namrata Limaye
- Neha Singh
- James Choi
- Mohammed Arif Khan
- Sathya Narayana Pottimurthy
- Santosh Karthik Prasad Vajghala
- Sai Krishna Gogineni
- Michael Parker
- Keith Kong
- Aravindkumar MG
- Dan Talayco
- Andy Fingerhut

Credits

- spaces vs tabs gif
  - https://getyarn.io/yarn-clip/2dcf7166-cb68-4e0b-a867-fd99a13a9dc7Multi-tier programs

More Info

visit: p4tc.dev

Back Slides

Taking Advantage Of Other TC Features

- Multi-tier programs
  - Program 1: ACL
  - Program 2: LB
  - Program 3: Forwarding, NAT etc
- Pipeline Splitting
  - Two Use cases
    - Two different programs: Prog1 in h/w chain 10, Prog2 in s/w chain 5
    - Same program: Split table into h/w chain 10, s/w chain 5

TC Perspective: Stateful Tables