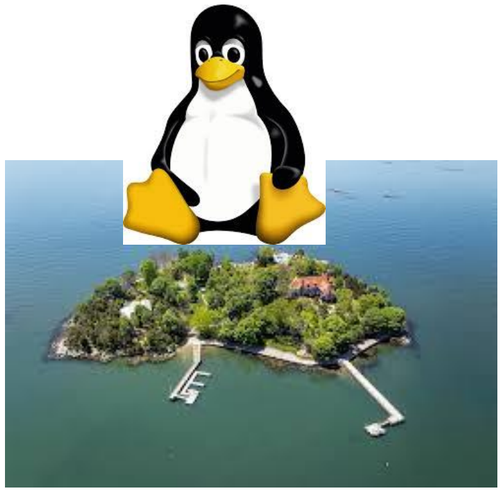
Compiler-assisted kernel-based P4 pipeline offloading using Intel IPU

Deb Chatterjee
Neha Singh

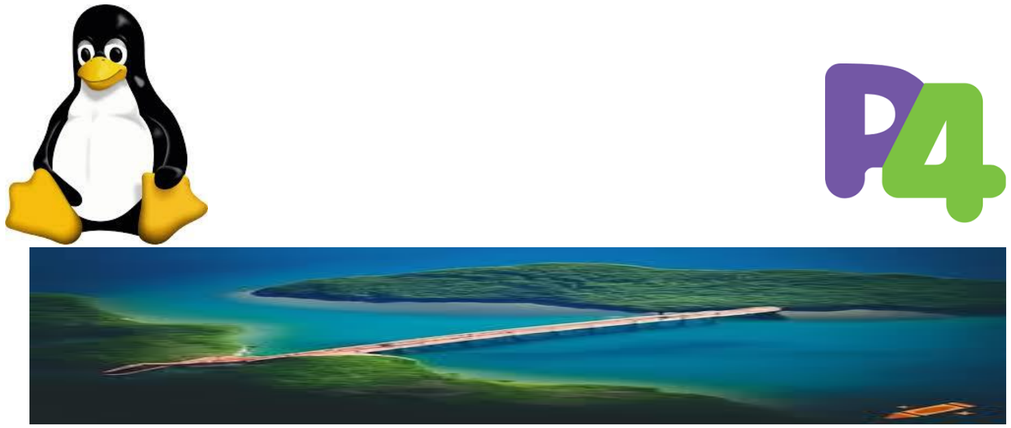
Two very important technologies, so far unconnected



- A new synonym for “Ubiquitous”
  - If a computing device is doing something useful, it is more than likely running Linux
- Mature API for HW offloads (via vendor driver)
- Powerful TC abstraction
  - Consistent regardless of deployment in SW or HW

- Standardized language for describing datapaths, whether in HW or SW
- Commoditization happening with native P4 support on xPUS (Intel and AMD)
  - Intel IPU ES 2100 support available
- Large consumers of NICs value the role of a minimal P4 for implementation as well as behavioral description of datapath
- New use cases are emerging, such as Microsoft DASH

We built a bridge!



P4TC is the bridge, for growing the Network Programmability ecosystem

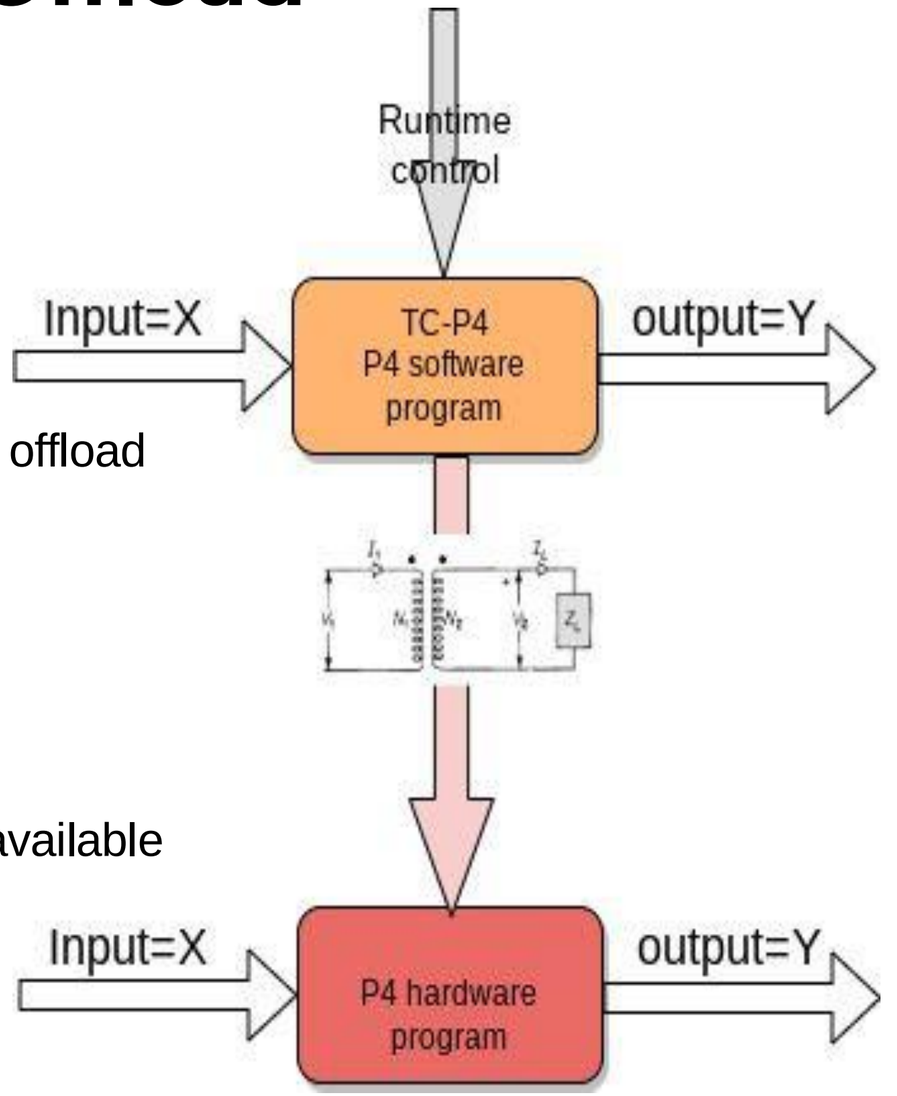
- Datapath definition using P4
  - Linux kernel native P4 implementation
  - Mundane developer knowledge automated into compiler
    - knowledge shift to system (and P4) from kernel skills
    - Zero upstream effort
- Same interfaces for either SW or HW datapaths
  - TC offload functionality
  - Intel IPU ES 2100 implementation to be discussed in this presentation

Introduction to P4-TC

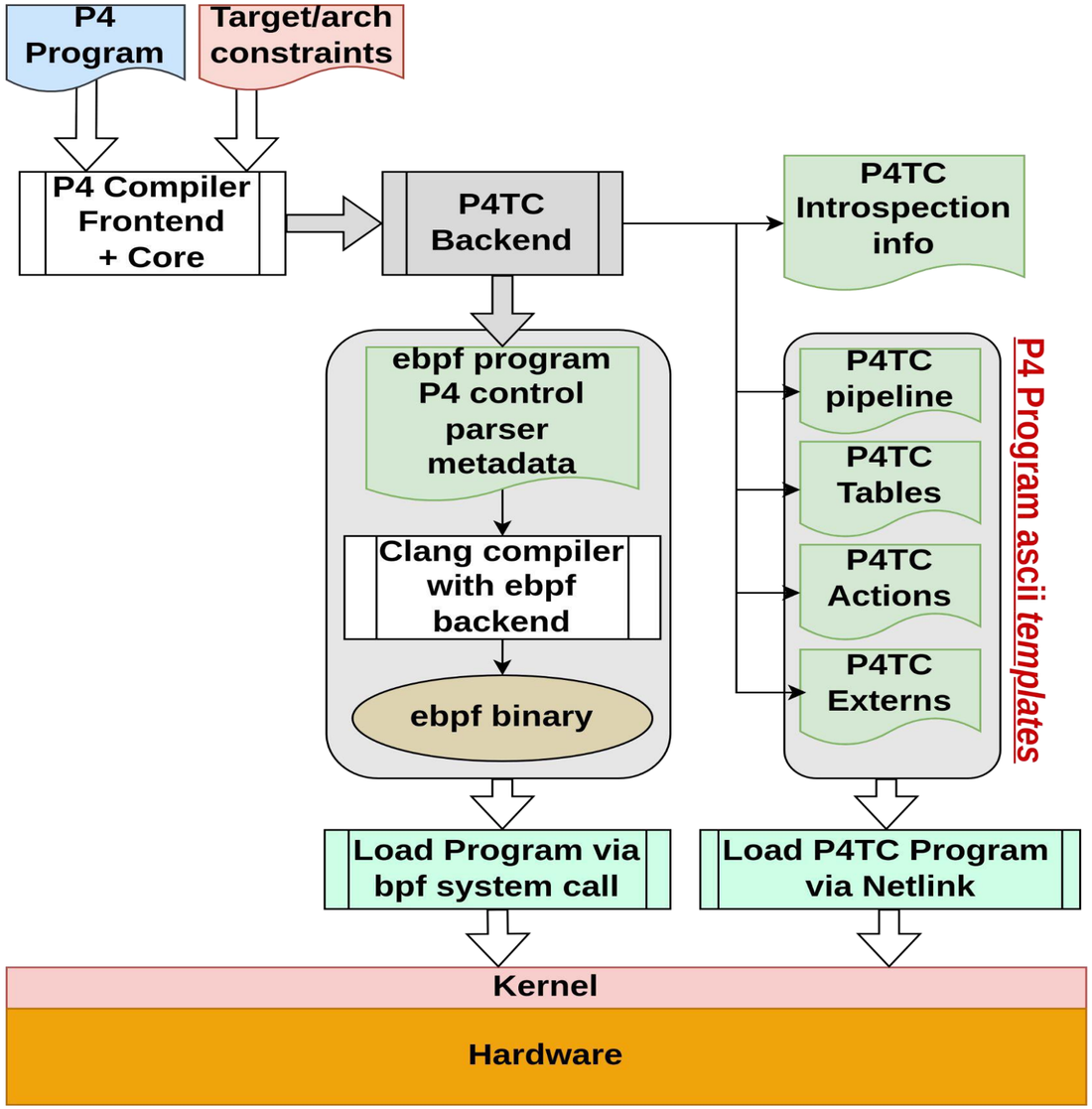
- TC based kernel-native P4 implementation
- Learn from previous experiences (tc flower, u32, switchdev, etc) and scale
  - Kernel independence
  - Control plane transaction rate and latency
- P4 Architecture Independence
  - Currently PNA with some extra “constructs”
    - Not hard to add other architectures
    - This is about progressing network programmability in addition to expanding P4 reach
- Use of P4 compiler backend
  - The P4-TC compiler backend generates script files for the TC implementation in Linux kernel (from version 6.3.x onwards).
  - The P4-TC compiler backend reuses code from the existing p4c-ebpf backend
- Vendor Independent interfacing
  - No need to deal with multiple vendor abstraction transformations (and multiple indirections)

P4TC: Building On TC Offload

- Datapath definition using P4
  - Generate the datapath for both s/w and vendor h/w
    - Functional equivalence between sw and hw
- P4 Linux kernel-native implementation
  - Kernel TC-based software datapath and Kernel-based HW datapath offload
    - Understood Infra tooling which already has deployments
  - Seamless software and hardware symbiosis
  - Functional equivalence whether offloading or s/w datapaths
    - Bare Metal, VMs, or Containers
  - Ideal for datapath specification
    - test in s/w container, VM, etc) then offload when hardware is available



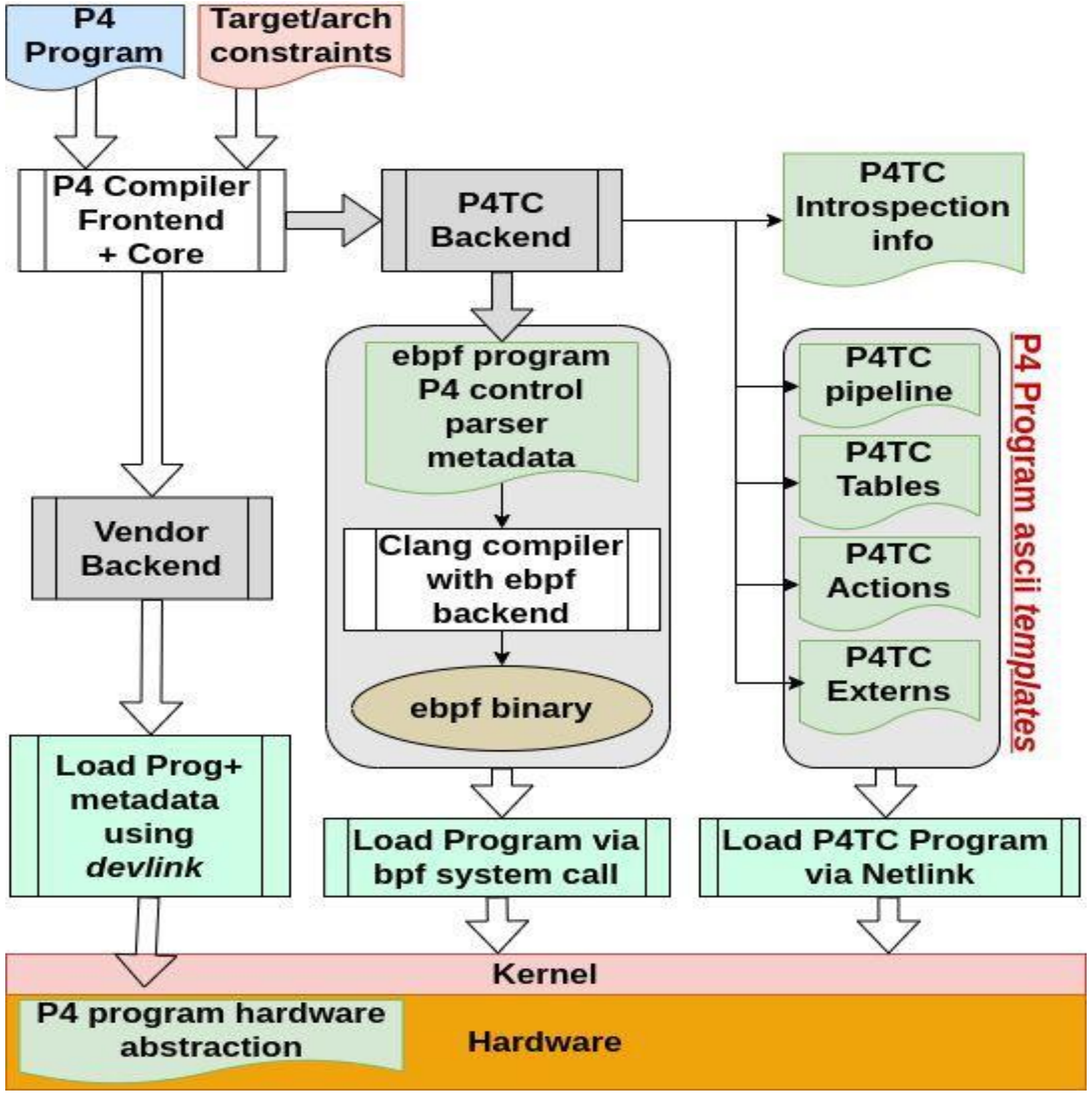
P4TC Software Datapath Workflow



Generated

1. P4TC Template (Loaded via generated) script
2. P4TC Introspection json (used by CP)
3. eBPF s/w datapath (at tc and/or xdp level)
   *Per packet execution engine
   (compiled and loaded when instantiating)

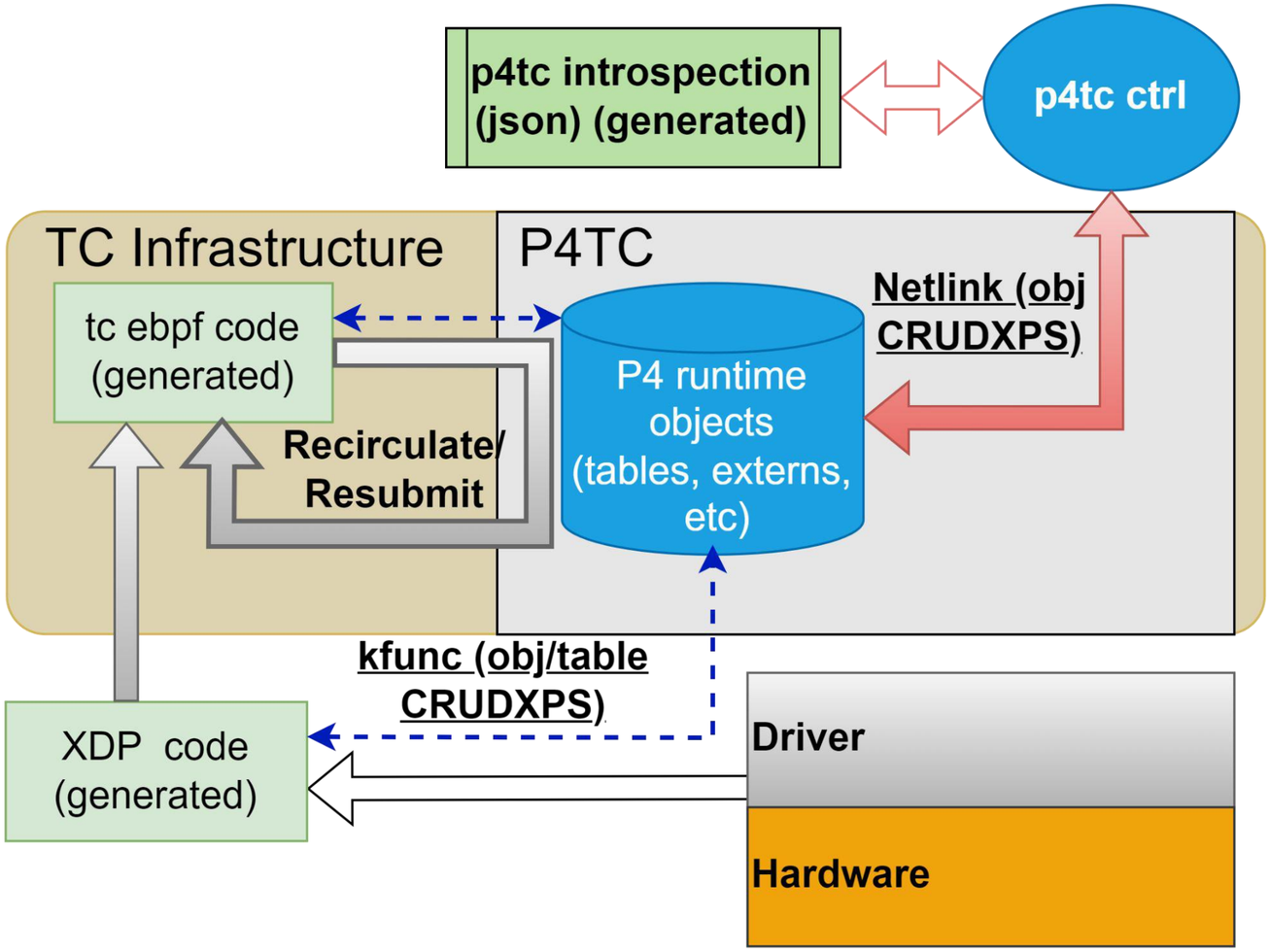
P4TC Workflow With HW offload



HW offload path also generates:

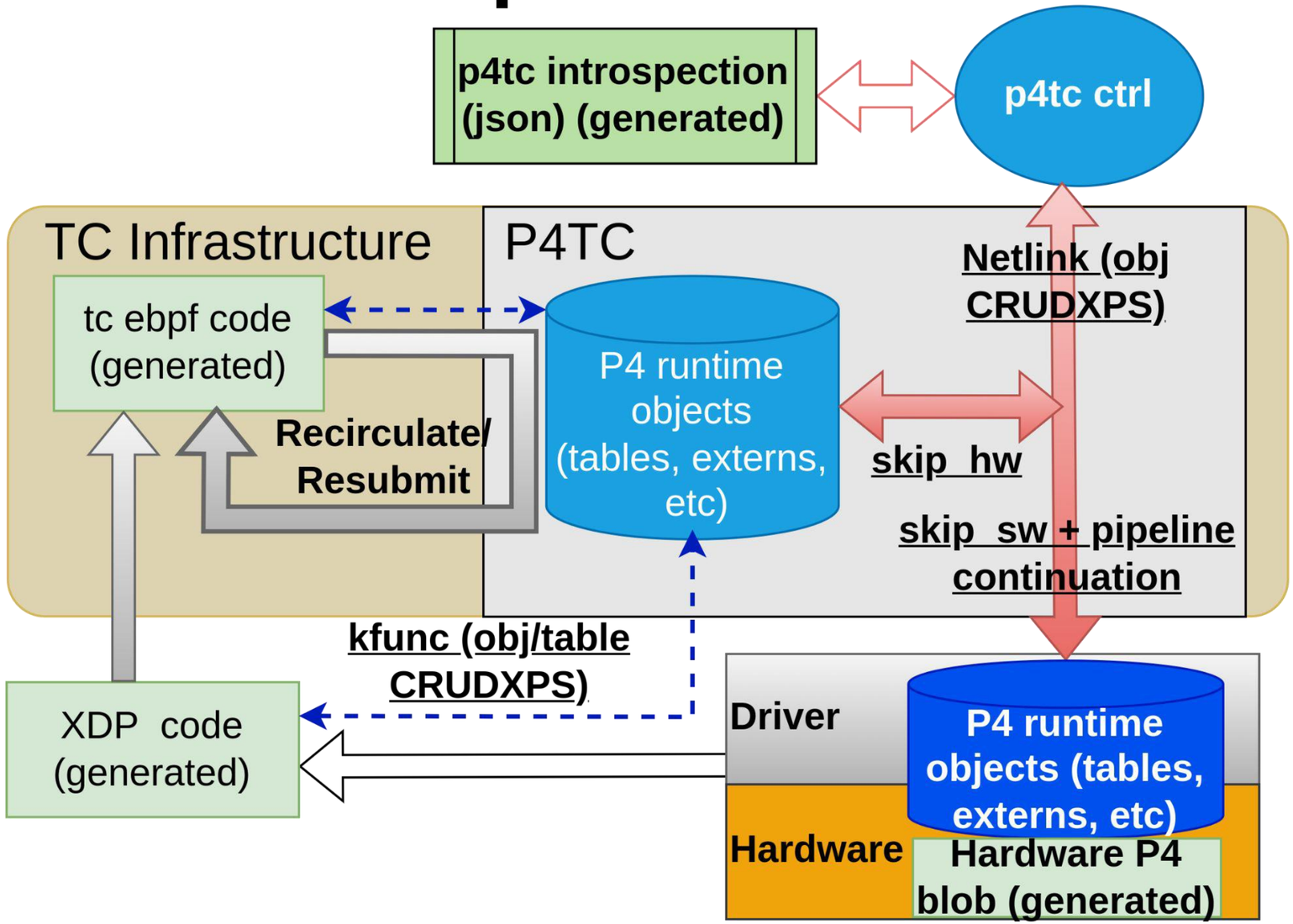
- **Binary hardware blob**
  - Compatible with vendor hardware
  - Loaded via firmware upload mechanisms

P4TC Runtime SW Datapath

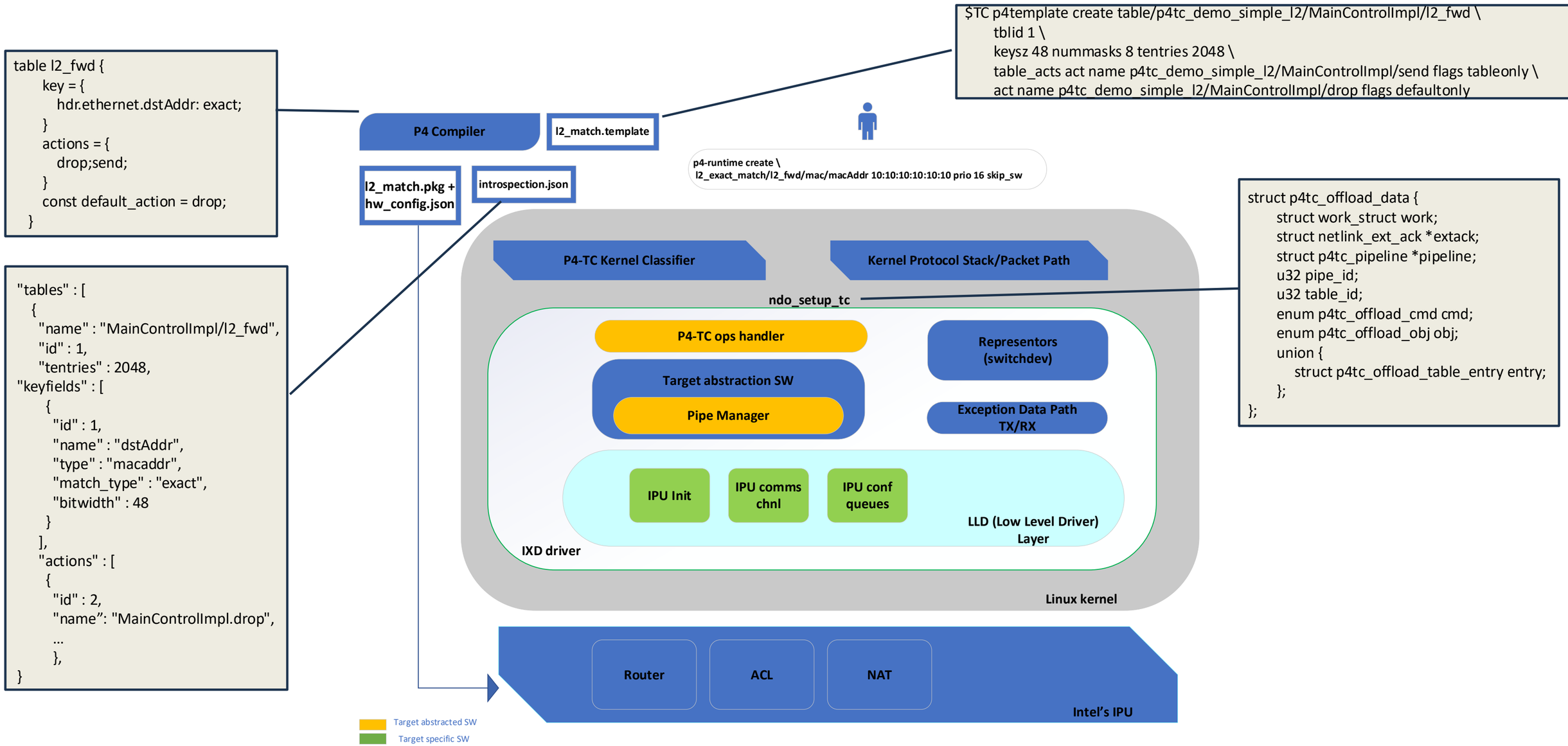


- eBPF serves as per packet exec engine
  - Parser, control block and deparser
- P4 objects that require control state reside in TC domain (attached to netns)
  - Actions, externs, pipeline, tables and their attributes (default hit/miss actions, etc)
  - Kfunc to access them from ebpf when needed

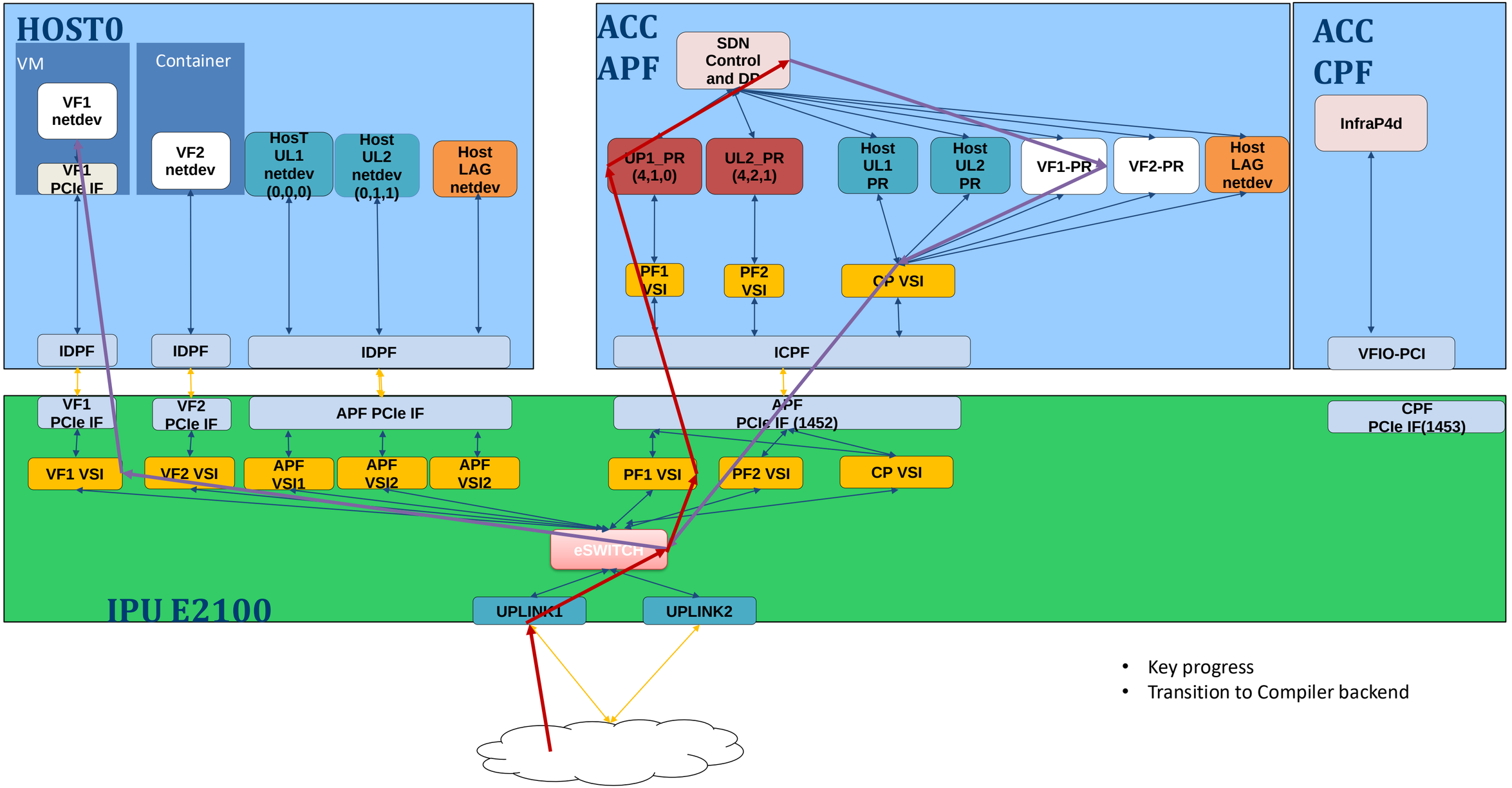
P4TC New Datapath With HW offload



P4TC driver architecture for Intel devices

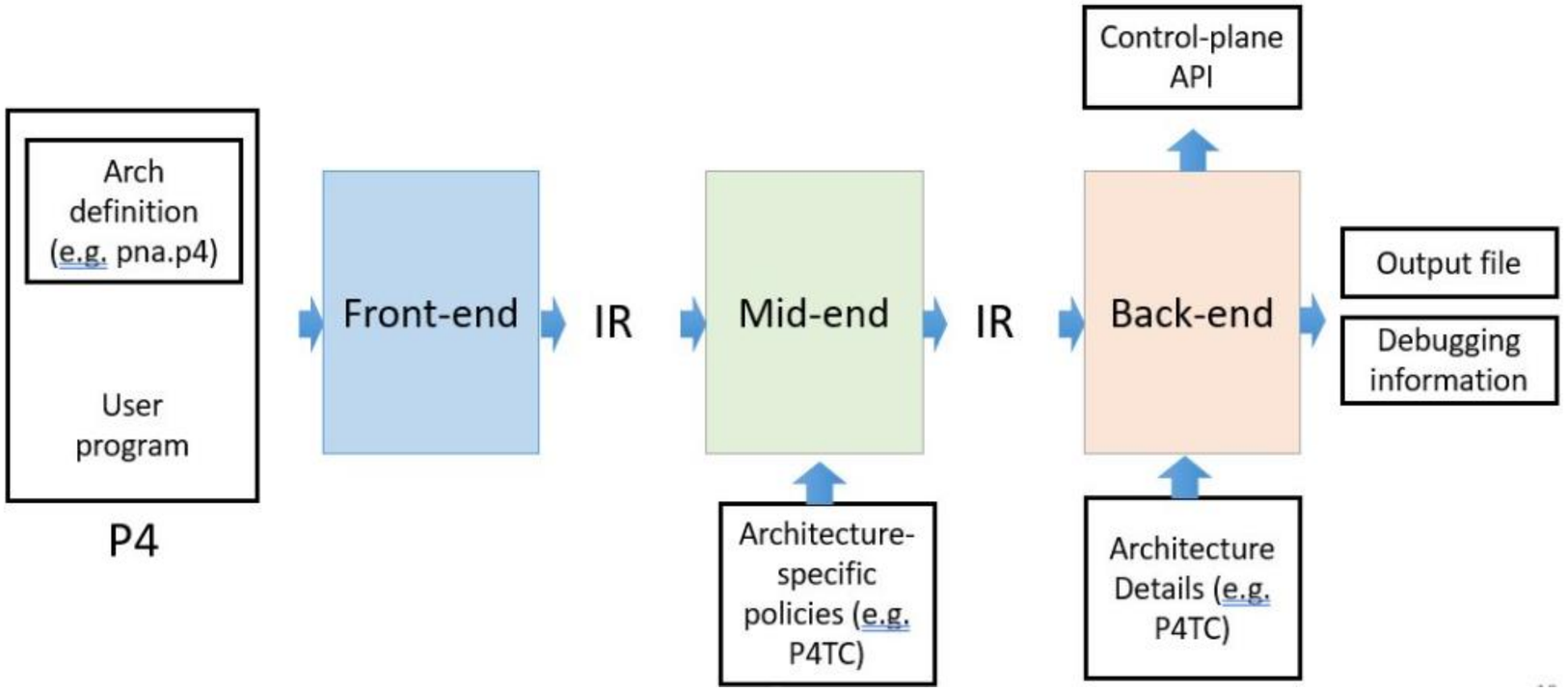


Switchdev Mode with 2 UL PRs On Host0(Uplink to VM)

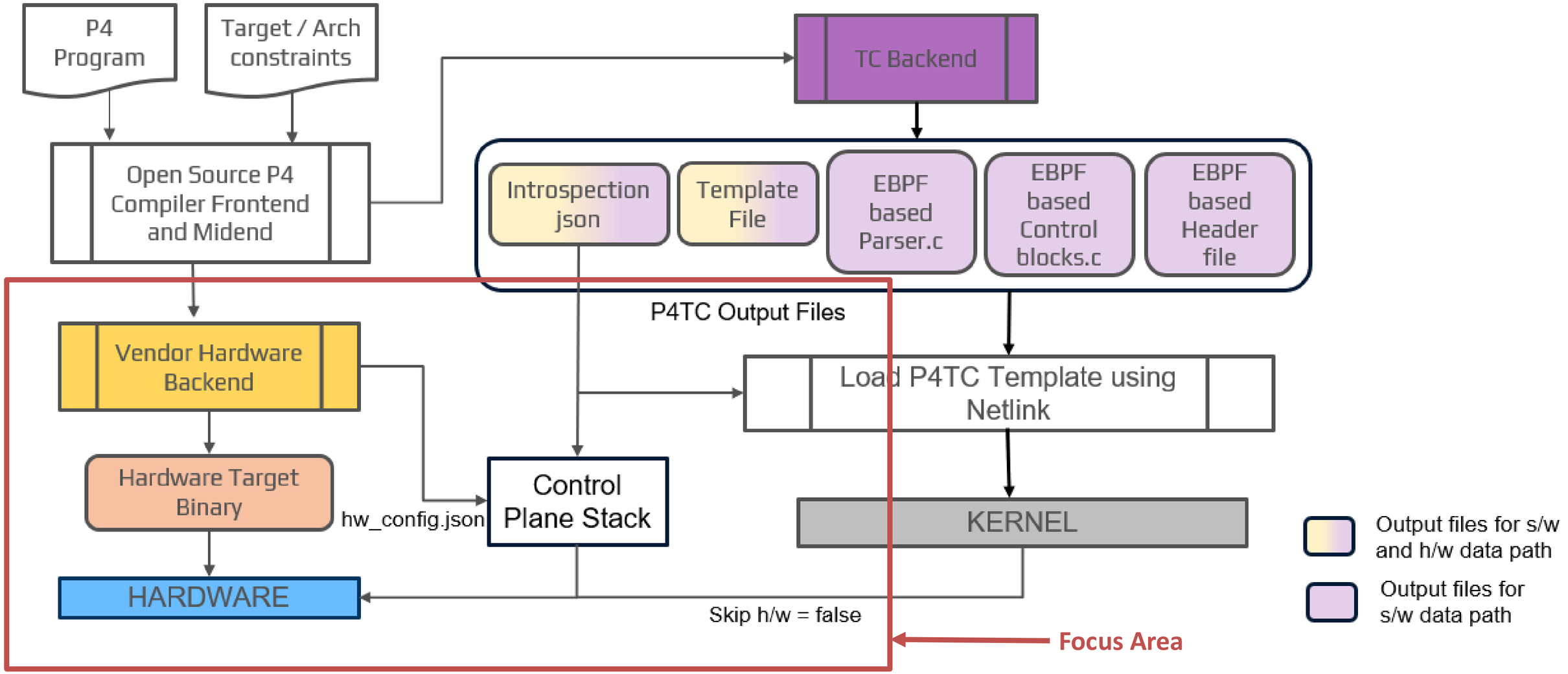


- Key progress
- Transition to Compiler backend

P4 compiler workflow

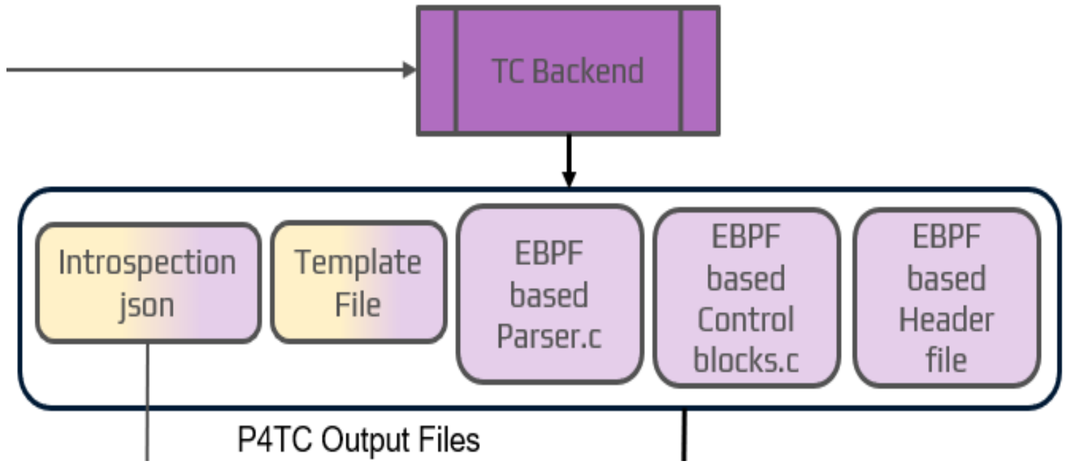


P4 compilation workflow for P4TC



Output files

- **Introspection json file -** Introspection json file is used for control plane programming.
- **Template File** – This file is a shell script that forms template definitions for various P4 objects (e.g. tables, actions).
- **EBPF based Parser C file** – This C file represents the parser definition.
- **EBPF based Control blocks C file** – This C files defines rest of the software datapath (i.e. control blocks and deparser)
- **EBPF Header file** – The header File defines all the structure definitions, include files.



Thank You