Linux Network Programming with P4

Linux Plumbers 2018

Fabian Ruffy, William Tu, Mihai Budiu

VMware Inc. and University of British Columbia

Outline

- Introduction to P4
- XDP and the P4 Compiler
- Testing
- Example
- Performance Results
- Discussion

Fabian

William

What is P4?

- High-level programming language for network data planes
  - Allows for protocol flexibility
  - Specifies a packet processing pipeline
- Compiled and loaded into target platform
- Open and standardized

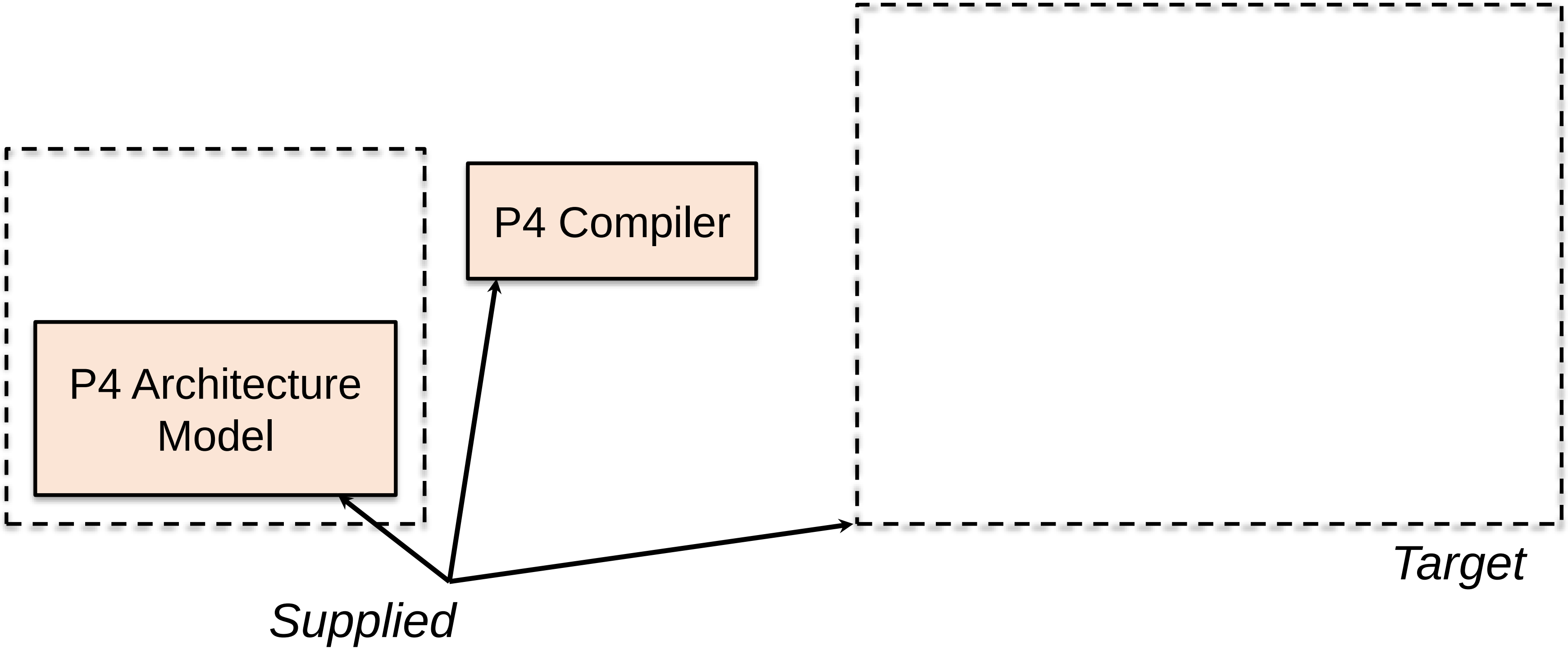
[P4: Programming Protocol-Independent Packet Processors]

Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, David Walker *ACM SIGCOMM Computer Communications Review (CCR). Volume 44, Issue #3 (July 2014)*

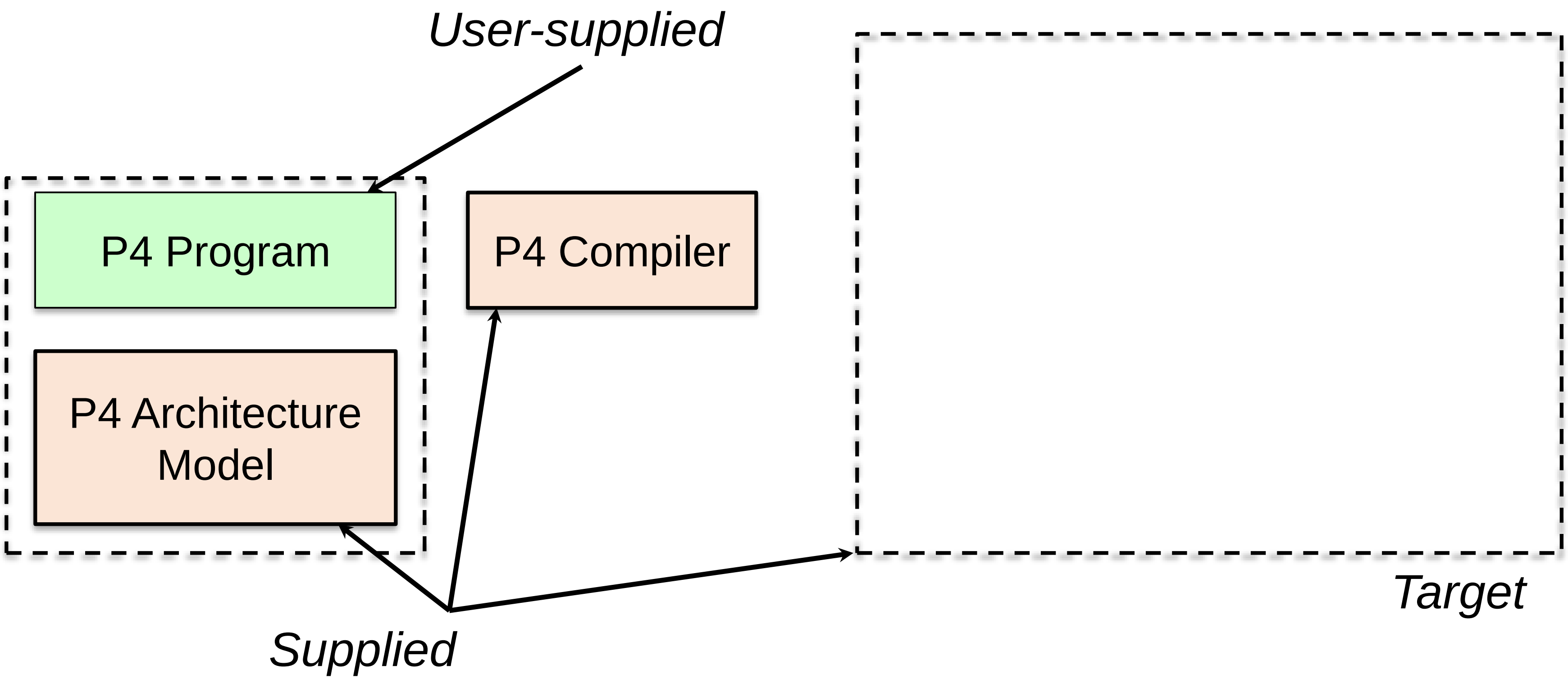
P4 Essentials

- C-like, strongly typed language
- Type and memory-safe (no pointers)
- Bounded execution (no loops)
- Statically allocated (no malloc, no recursion)
- Spec:
  http://github.com/p4lang/p4-spec
- Reference compiler implementation:
  http://github.com/p4lang/p4c (Apache 2 license)

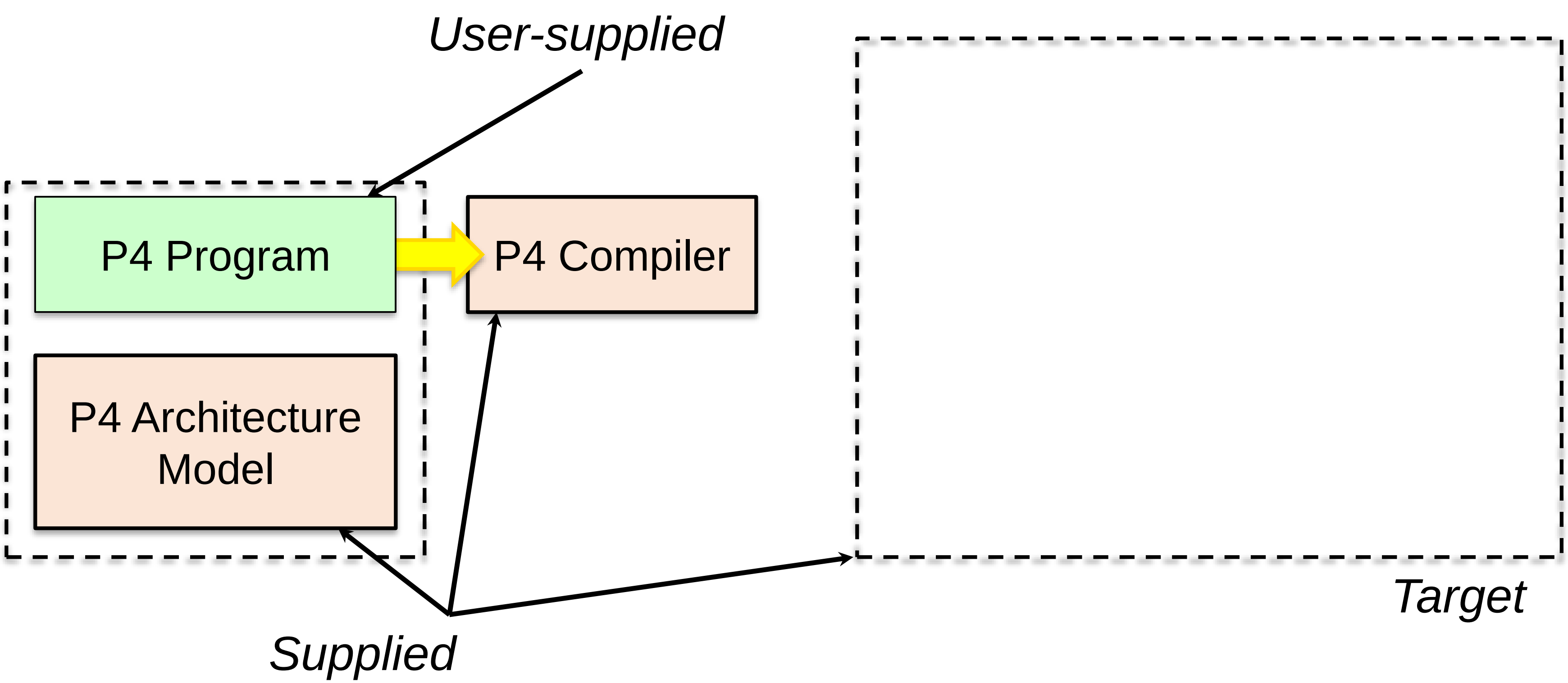
P4 Software Workflow



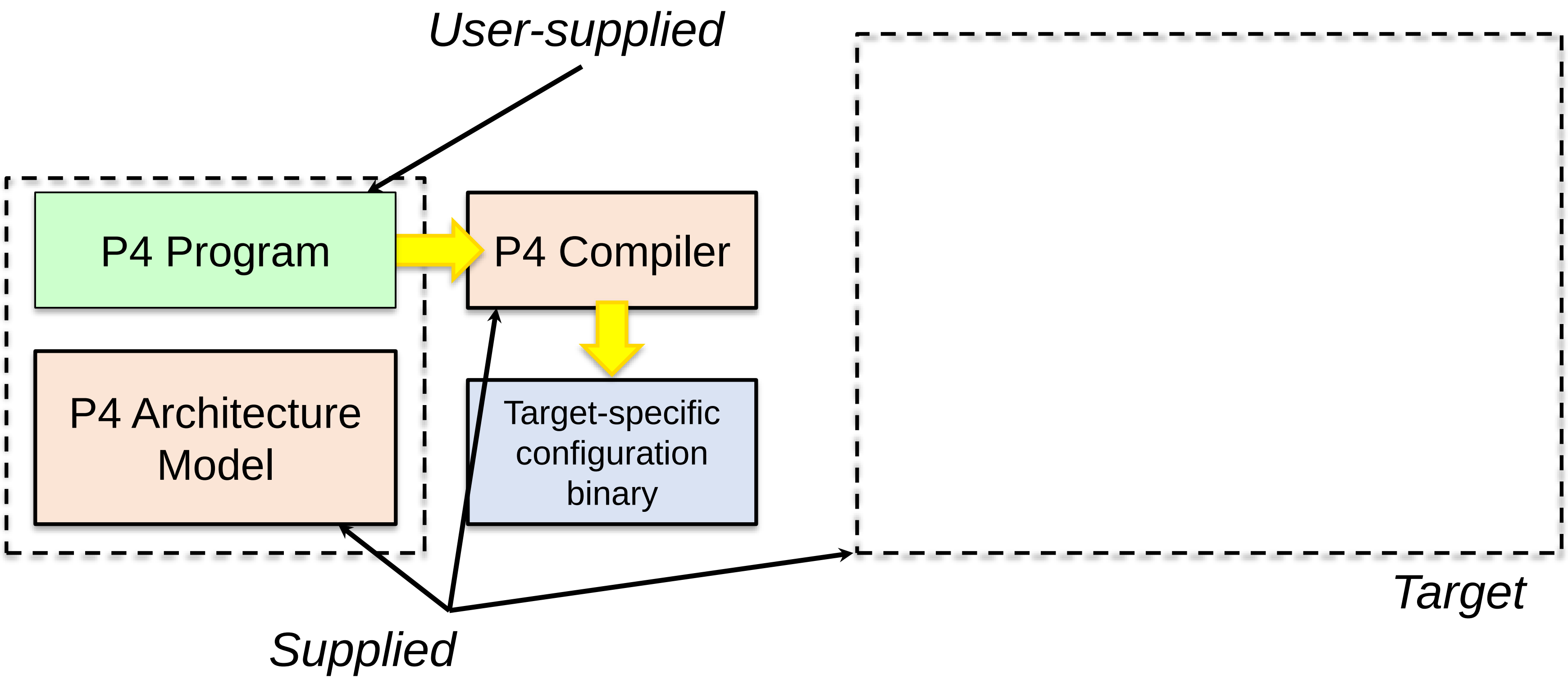
P4 Software Workflow



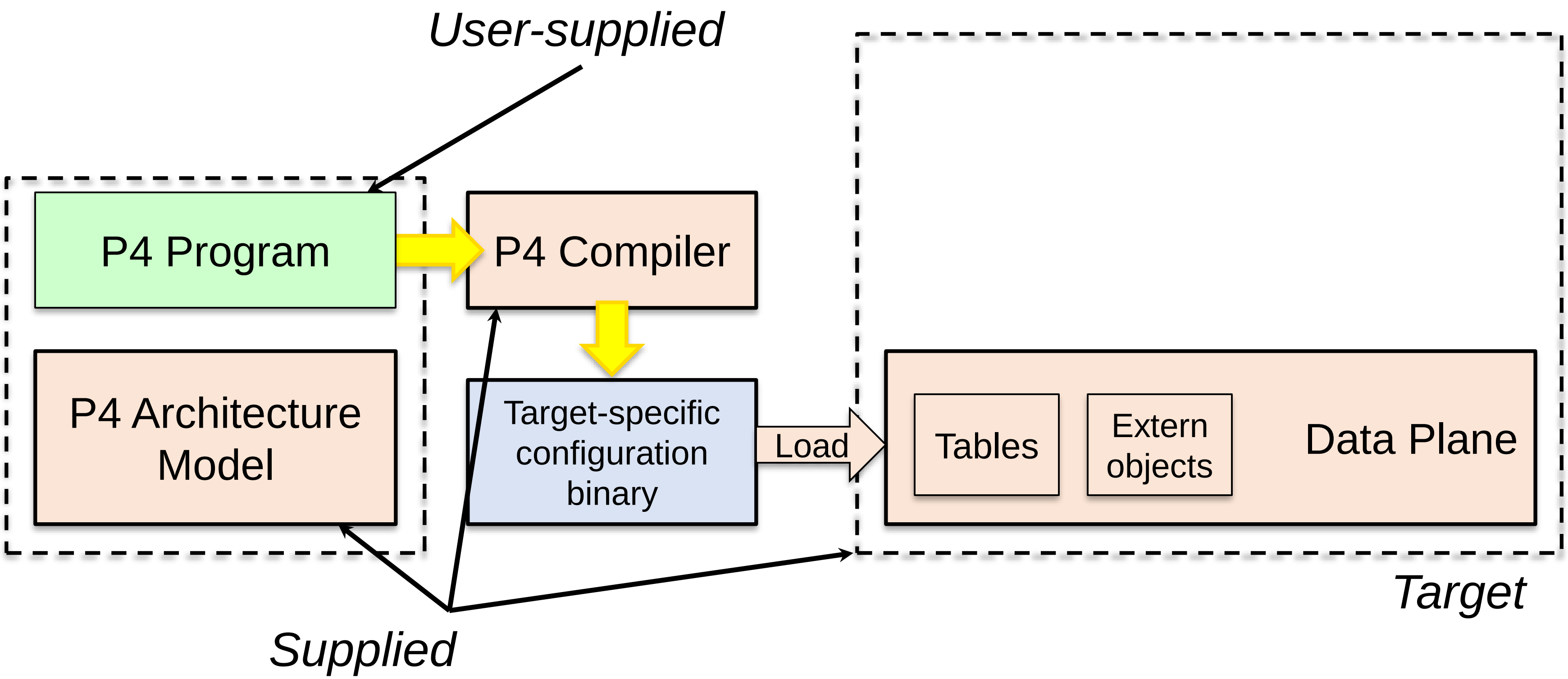
P4 Software Workflow



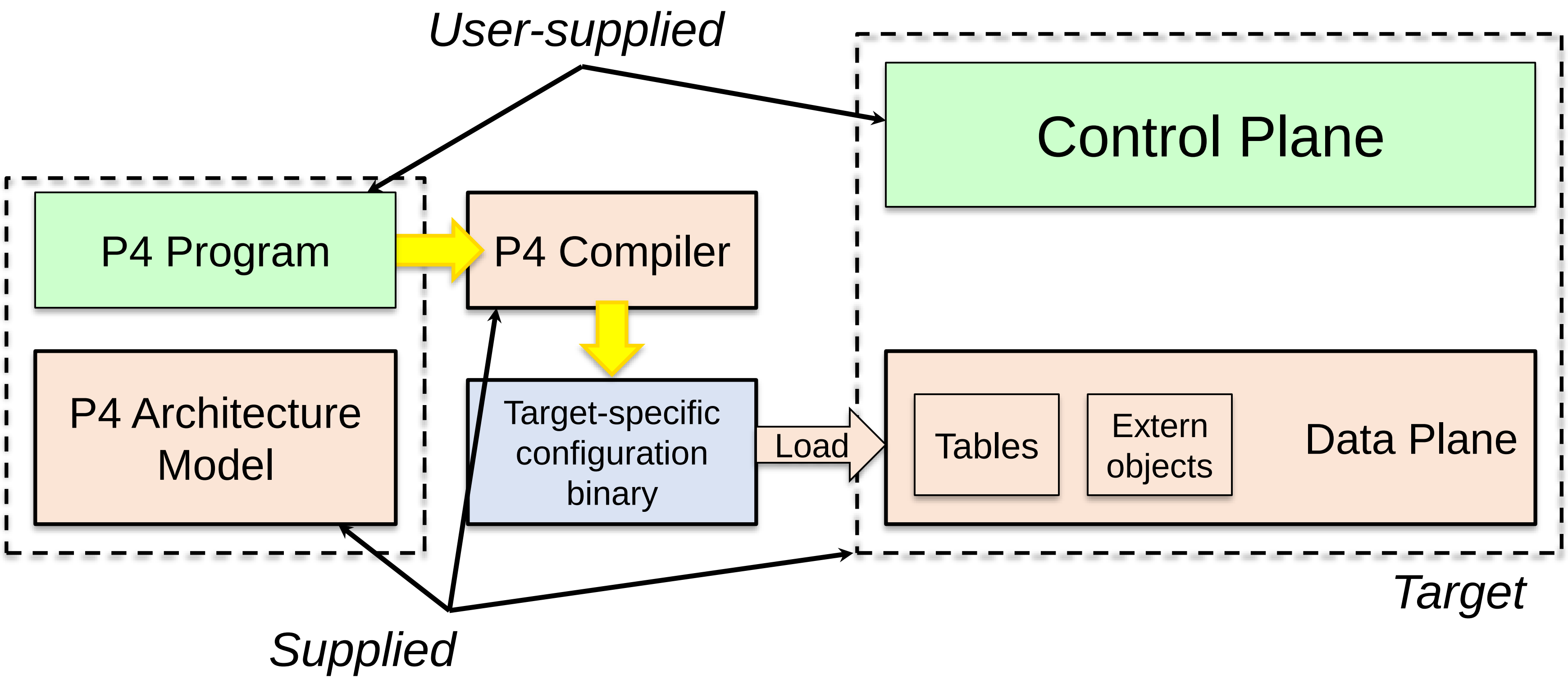
P4 Software Workflow



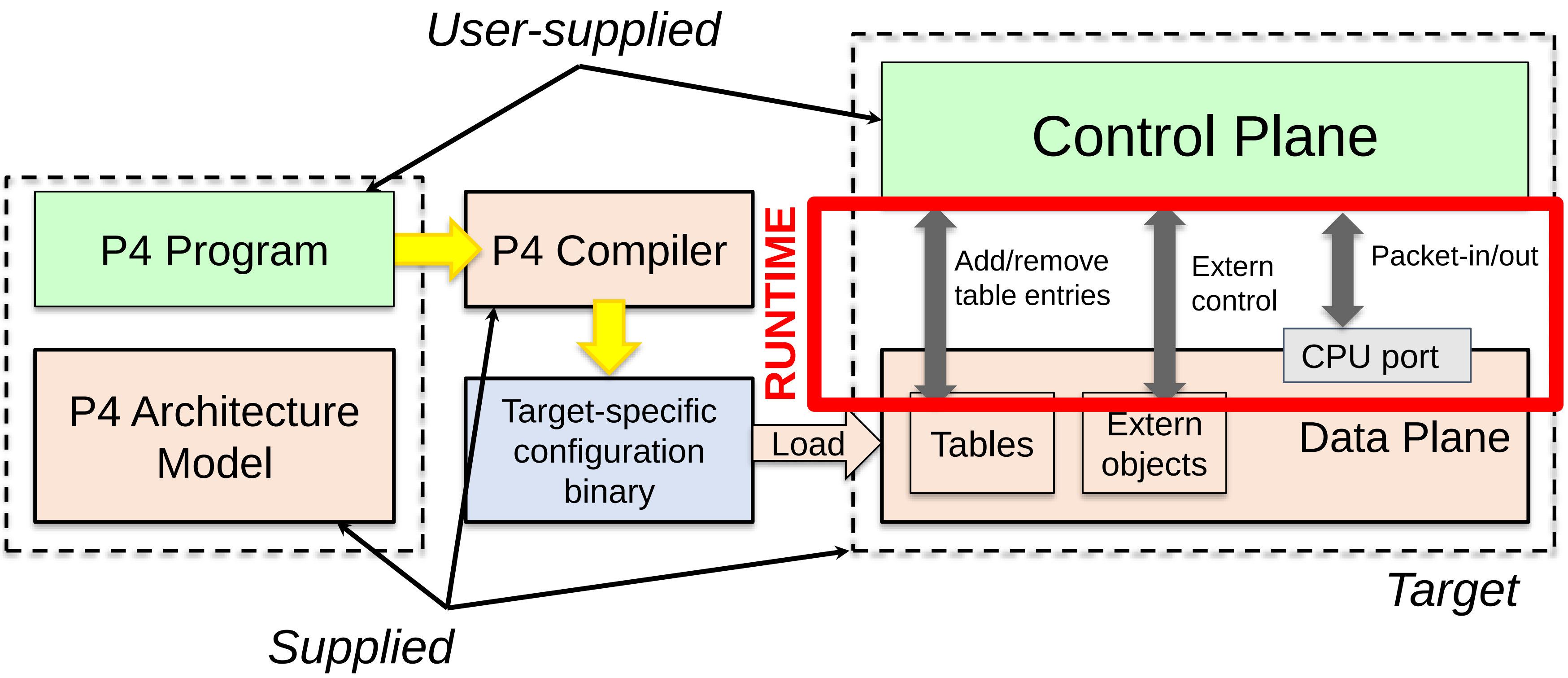
P4 Software Workflow



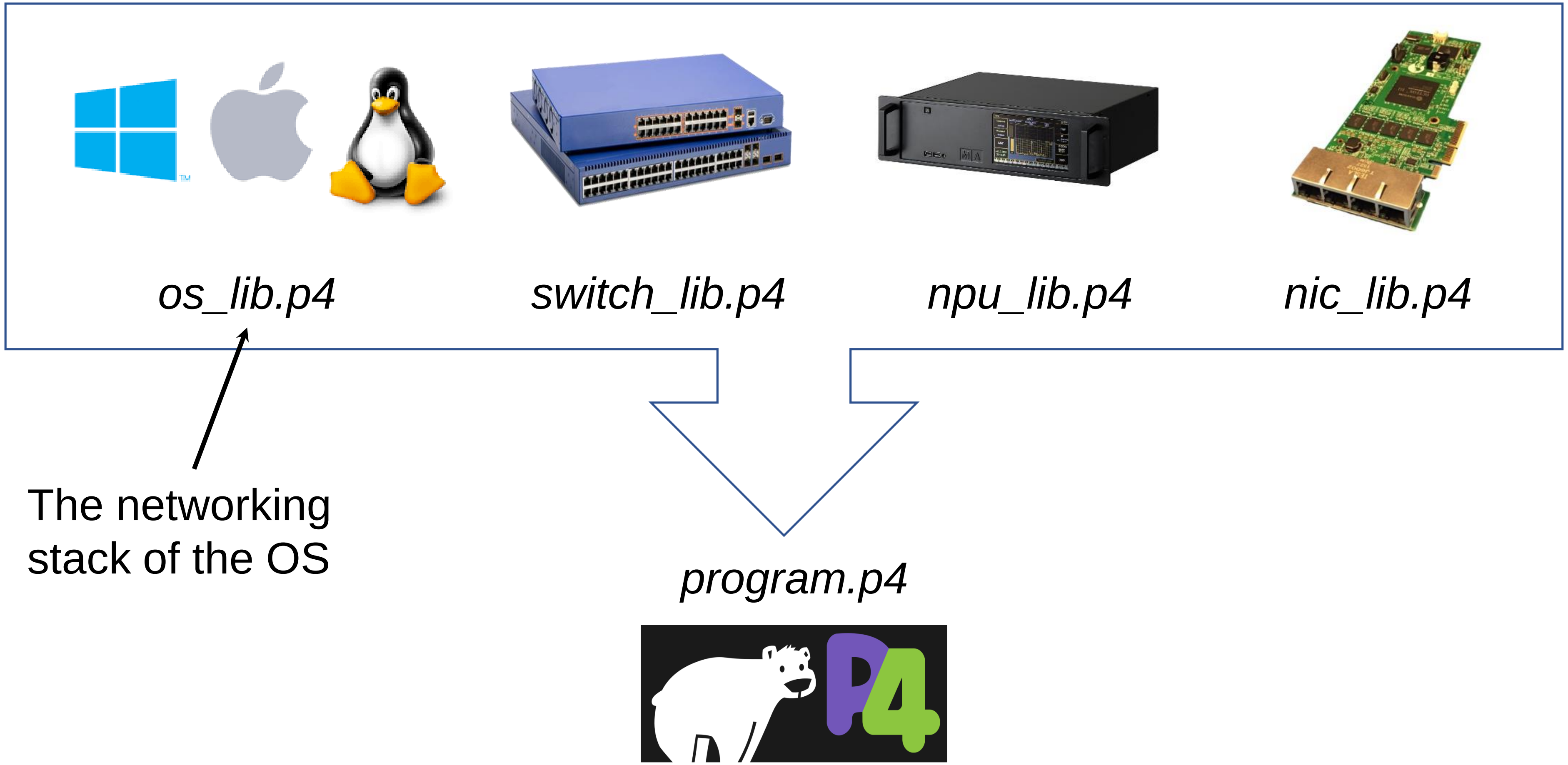
P4 Software Workflow



P4 Software Workflow



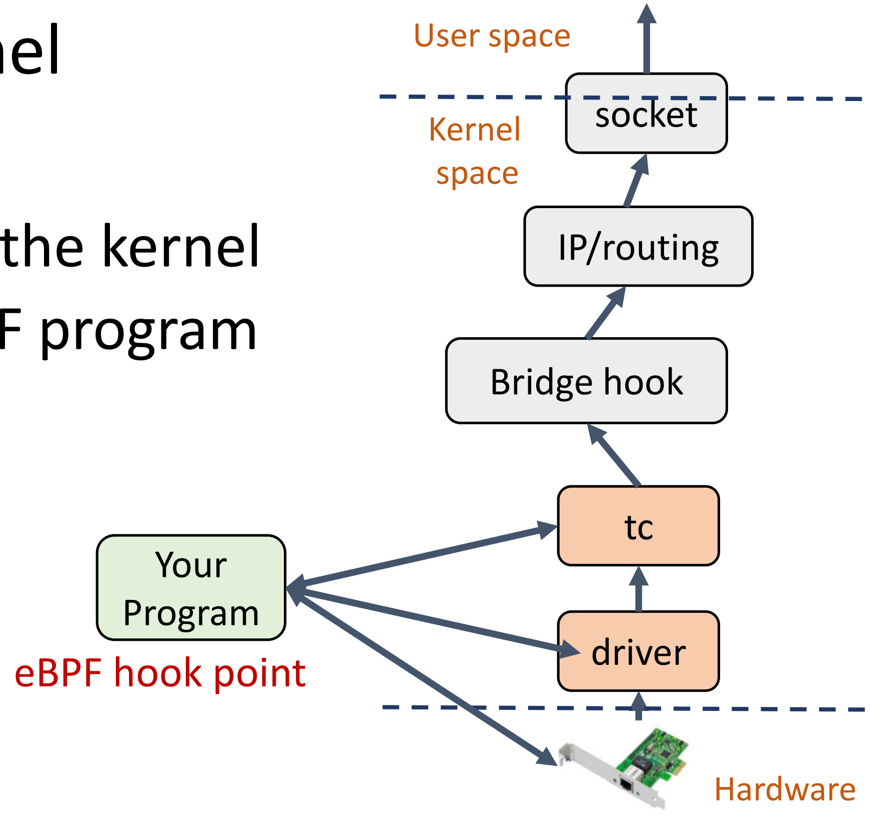
$P4_{16}$ generic data-plane model



P4 and XDP

eBPF/XDP

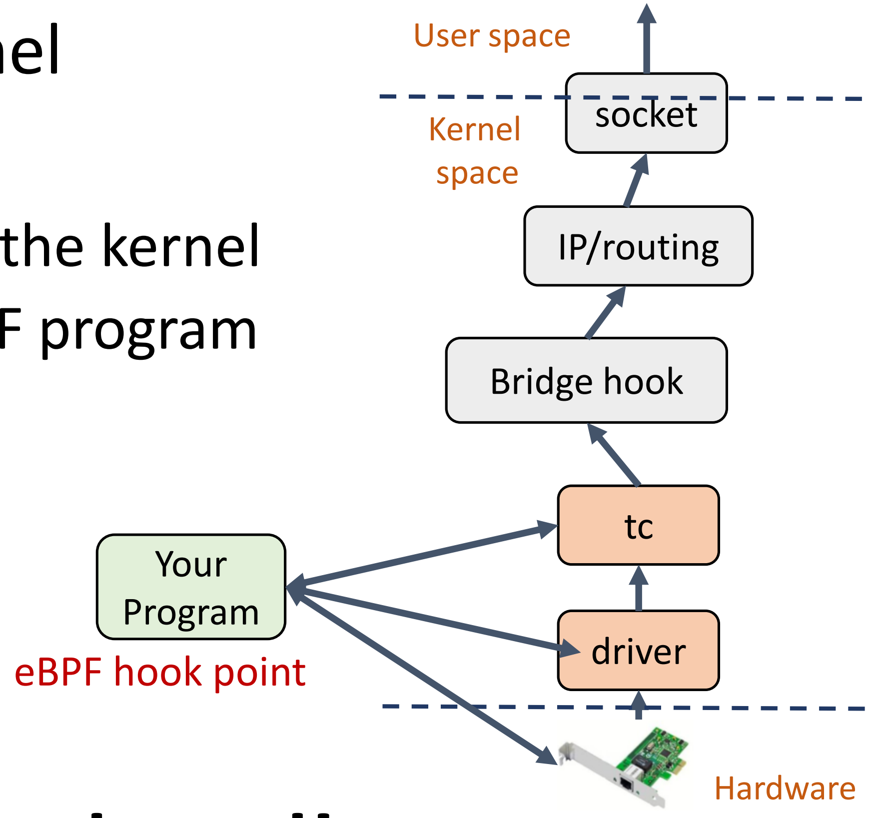
- Virtual machine running in the Linux kernel
- Provides:
  - The ability to write restricted C and run it in the kernel
  - A set of kernel hook points invoking the eBPF program
- Extensible, safe and fast
- Alternative to user-space networking



Example of TC+eBPF

eBPF/XDP

- Virtual machine running in the Linux kernel
- Provides:
  - The ability to write restricted C and run it in the kernel
  - A set of kernel hook points invoking the eBPF program
- Extensible, safe and fast
- Alternative to user-space networking



Example of TC+eBPF

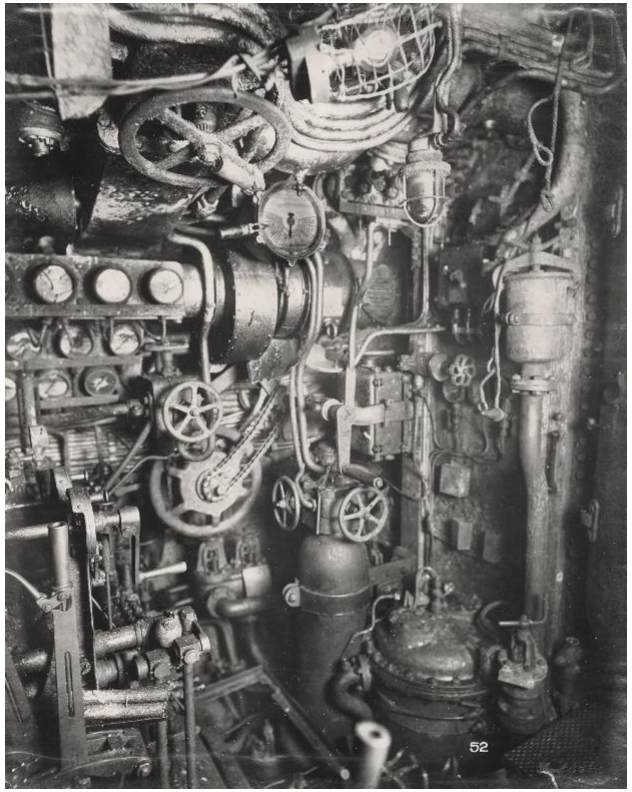
A programmable data plane in the Linux kernel!

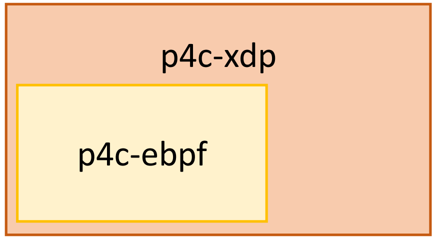
P4 vs eBPF/XDP

Feature	P4	eBPF/XDP
Level	High	Low
Safe	Yes	Yes
Safety	Type system	Verifier
Loops	In parsers	Tail calls (dynamic limit)
Resources	Statically allocated	Statically allocated
Policies	Tables (match+action)	Maps (tables)
Extern helpers	Target-specific	Hook-specific
Control-plane API	Synthesized by compiler	eBPF maps

The P4 eBPF backends

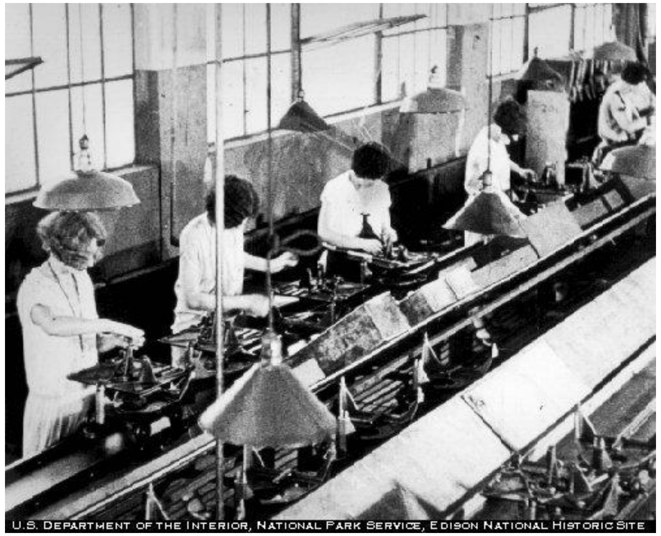
- p4c-ebpf is part of the open-source distribution
  - http://github.com/p4lang/p4c/backends/ebpf
- p4c-xdp is a separate open-source project
  - http://github.com/vmware/p4c-xdp
  - Extension of the p4c compiler
  - Reuses much of the code
- Not production-ready
  - Needs more work
  - Known bugs and limitations
  - Generated not efficient yet





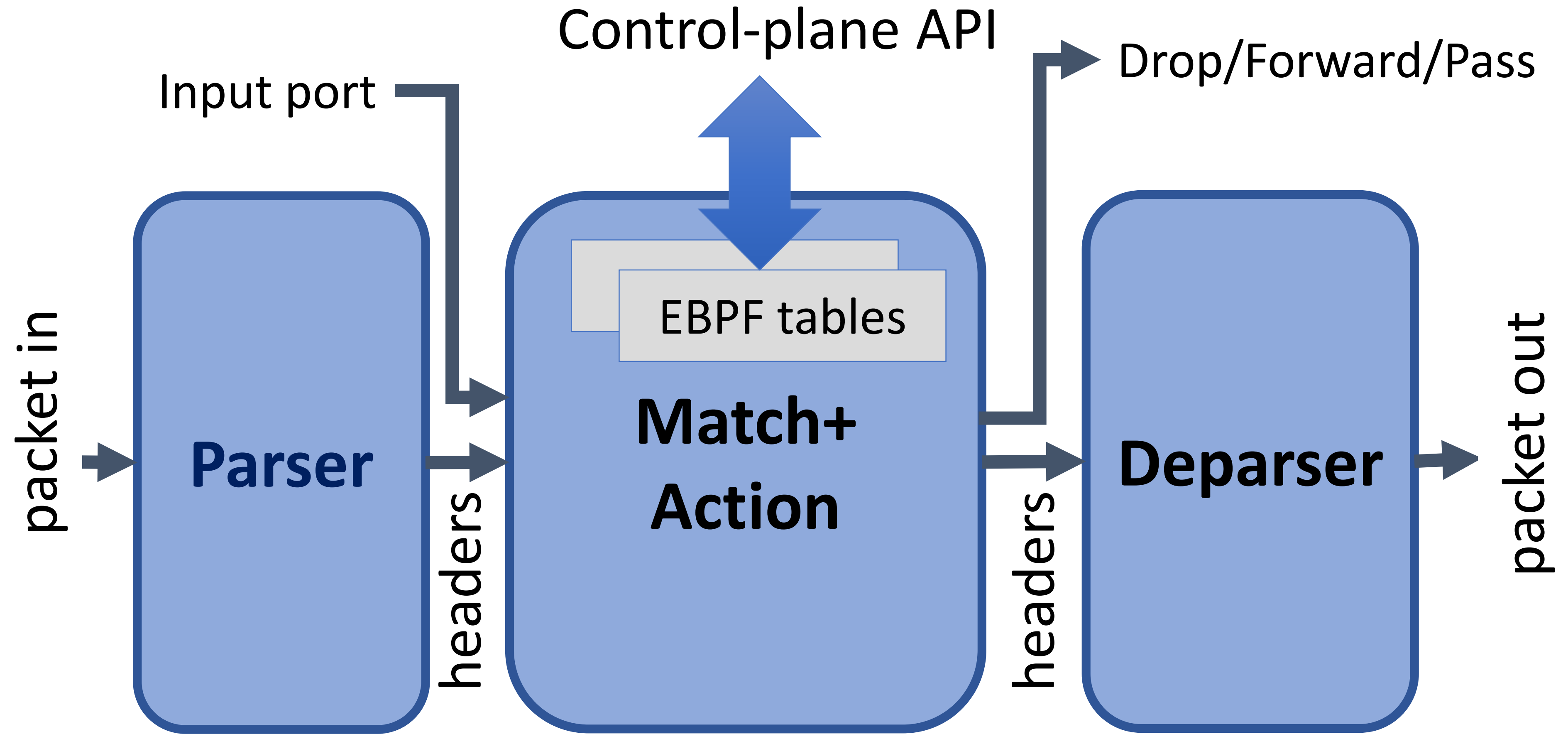
Generating XDP code

P4$_{16}$ -> C -> eBPF/XDP



- Generates stylized C
- No tail calls yet, all data on stack
- eBPF tables control/data-plane communication
- Can do filtering, forwarding, encapsulation
- Relies on Linux TC for forwarding
  - We plan on switching to libbpf

The XDP Switching Model



XDP Data Plane

Flow

app.p4

User space

BPF system call

Kernel space

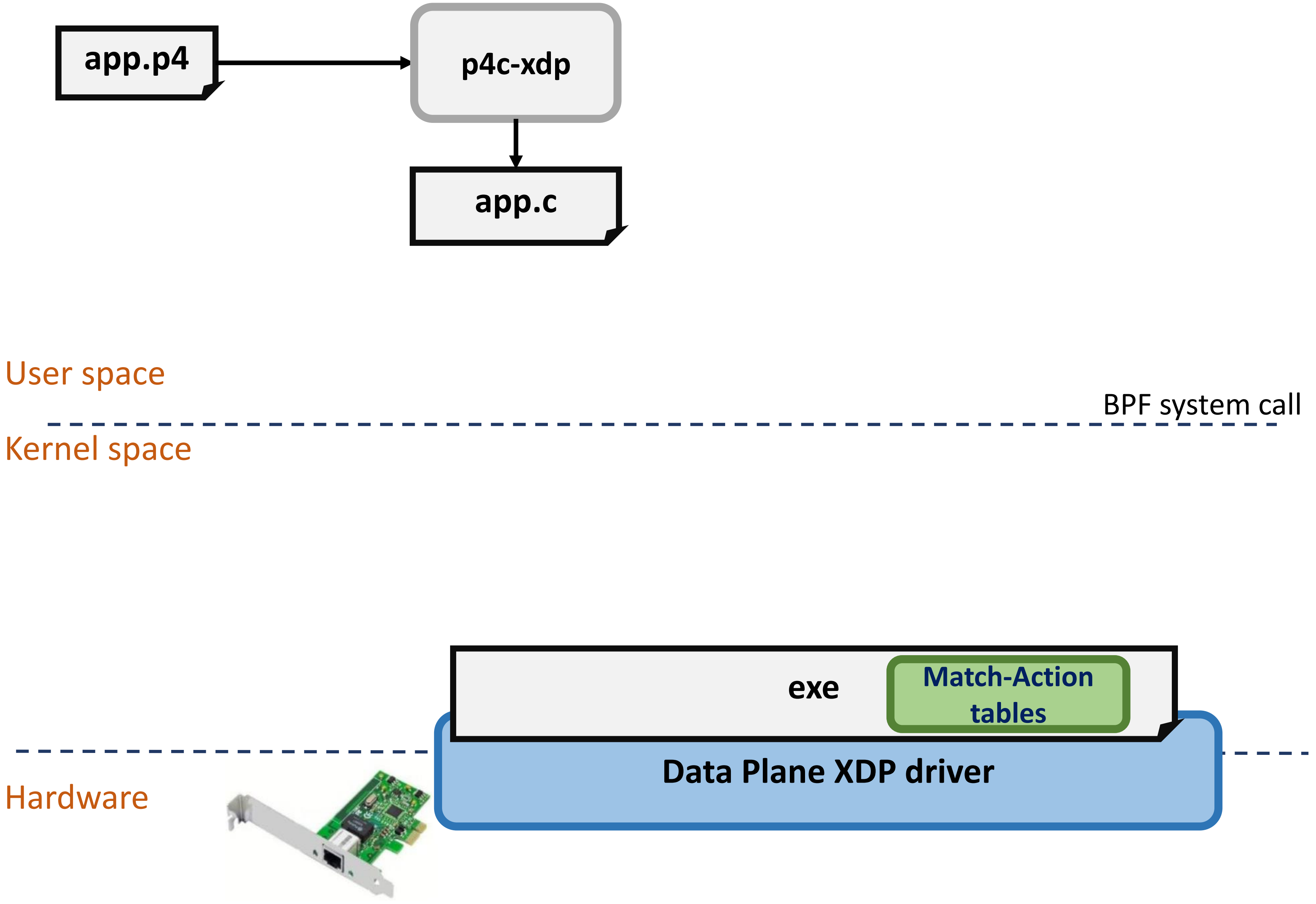
exe

Match-Action tables

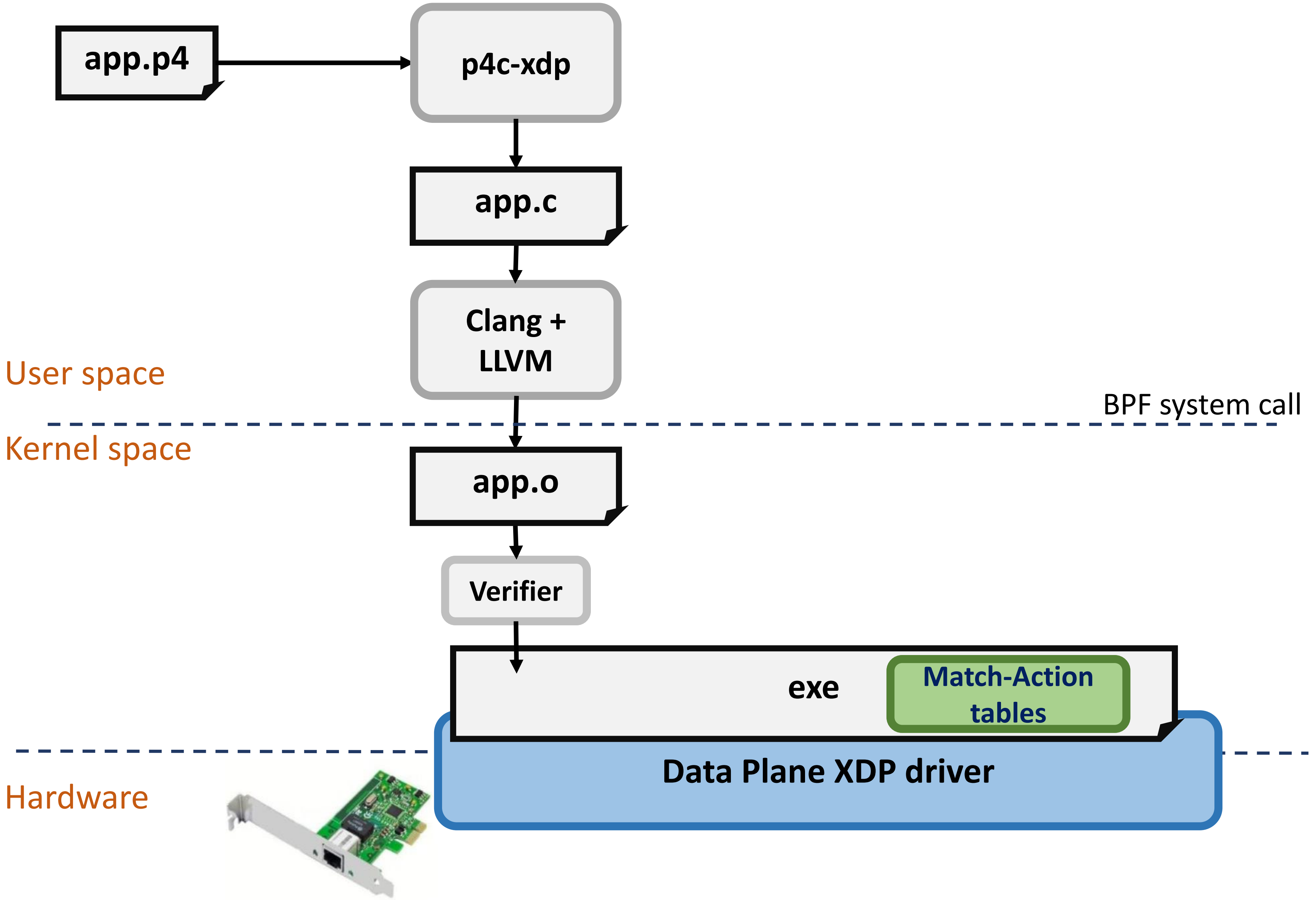
Data Plane XDP driver

Hardware

Flow

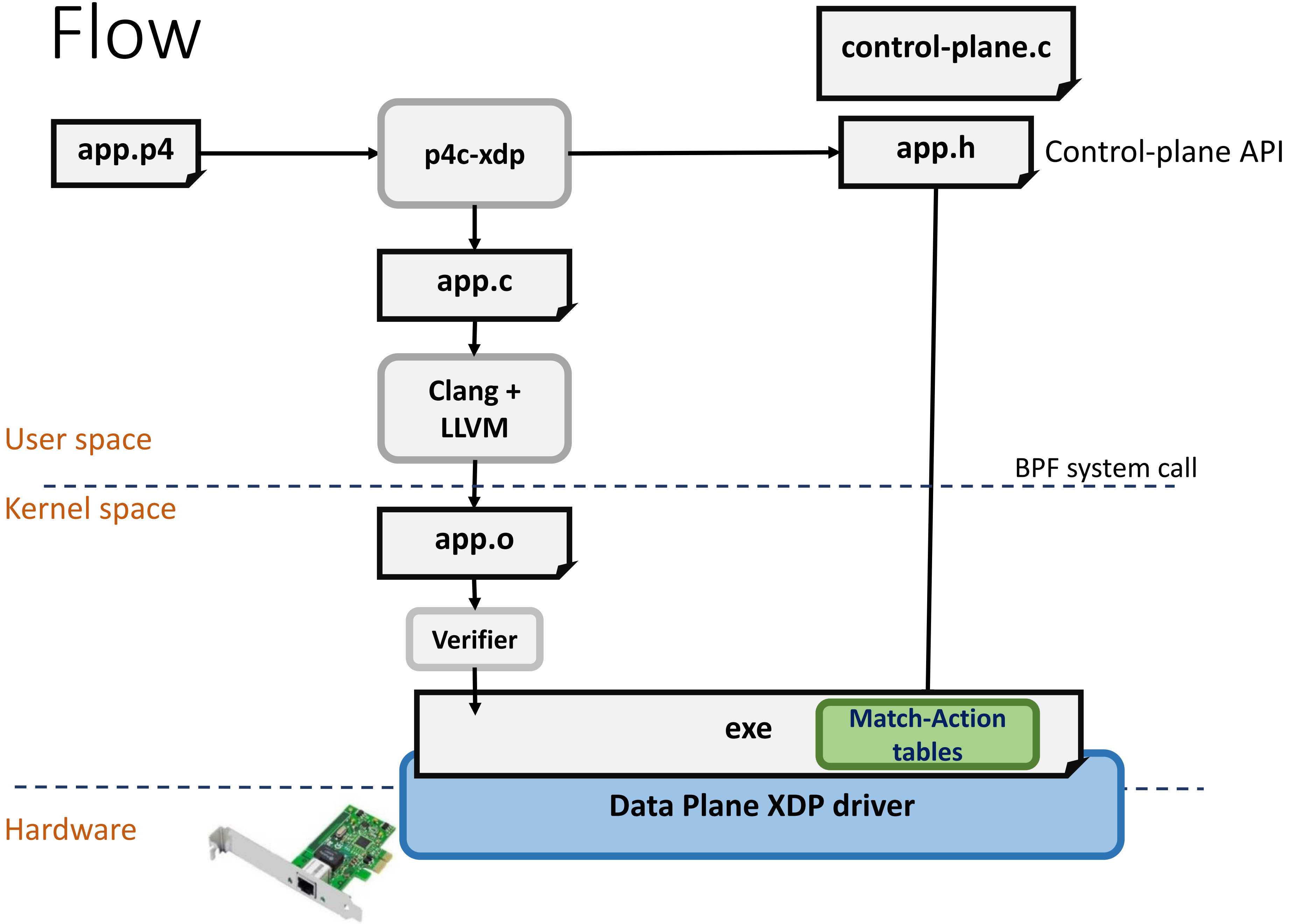

app.p4
p4c-xdp
app.c
User space
BPF system call
Kernel space
exe
Match-Action tables
Data Plane XDP driver
Hardware

Flow



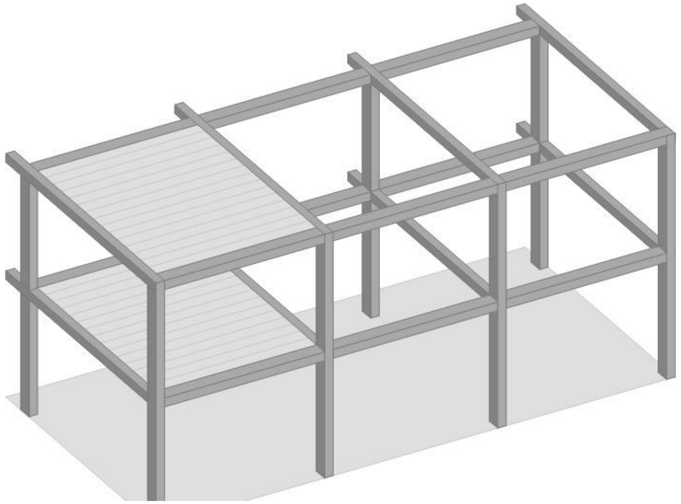
app.p4
p4c-xdp
app.c
Clang +
LLVM
User space
BPF system call
Kernel space
app.o
Verifier
exe
Match-Action
tables
Data Plane XDP driver
Hardware

Flow


control-plane.c
app.p4
p4c-xdp
app.h
Control-plane API
app.c
Clang +
LLVM
User space
BPF system call
Kernel space
app.o
Verifier
exe
Match-Action
tables
Data Plane XDP driver
Hardware

Testing P4-XDP code

Test Frameworks



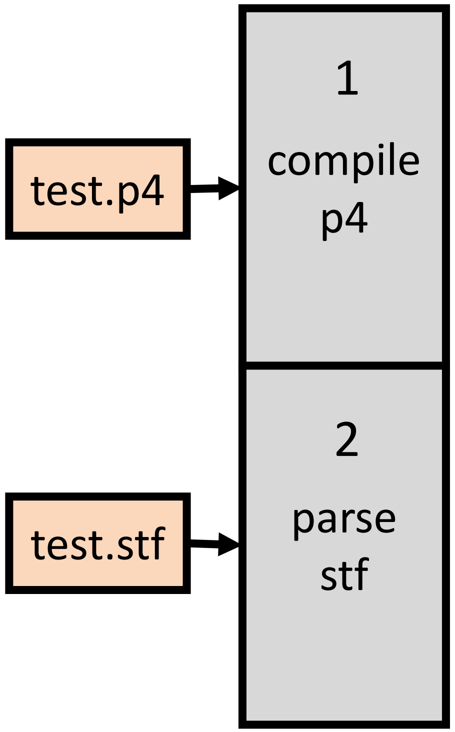
- User-space testing
  - Isolates specification from implementation
  - Validates correctness of generated code
  - User-space wrappers around eBPF tables and APIs
  - Reads and writes packets from capture files
- Kernel-space testing
  - Loads eBPF program into kernel
  - I/O connected to virtual interfaces
  - Writes capture files to interfaces in user-space
  - Records output using tcpdump

Five Testing Stages

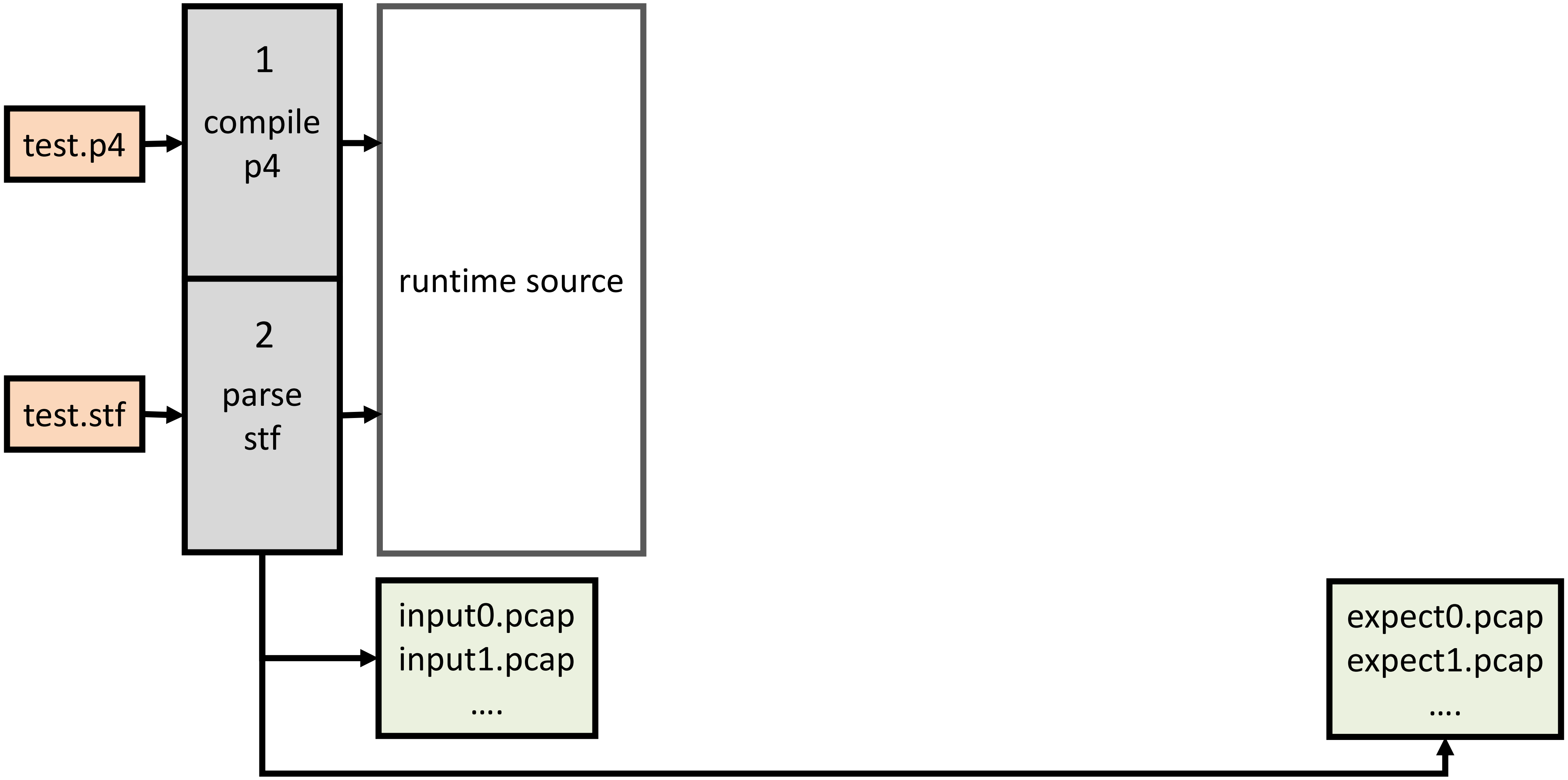
test.p4

test.stf

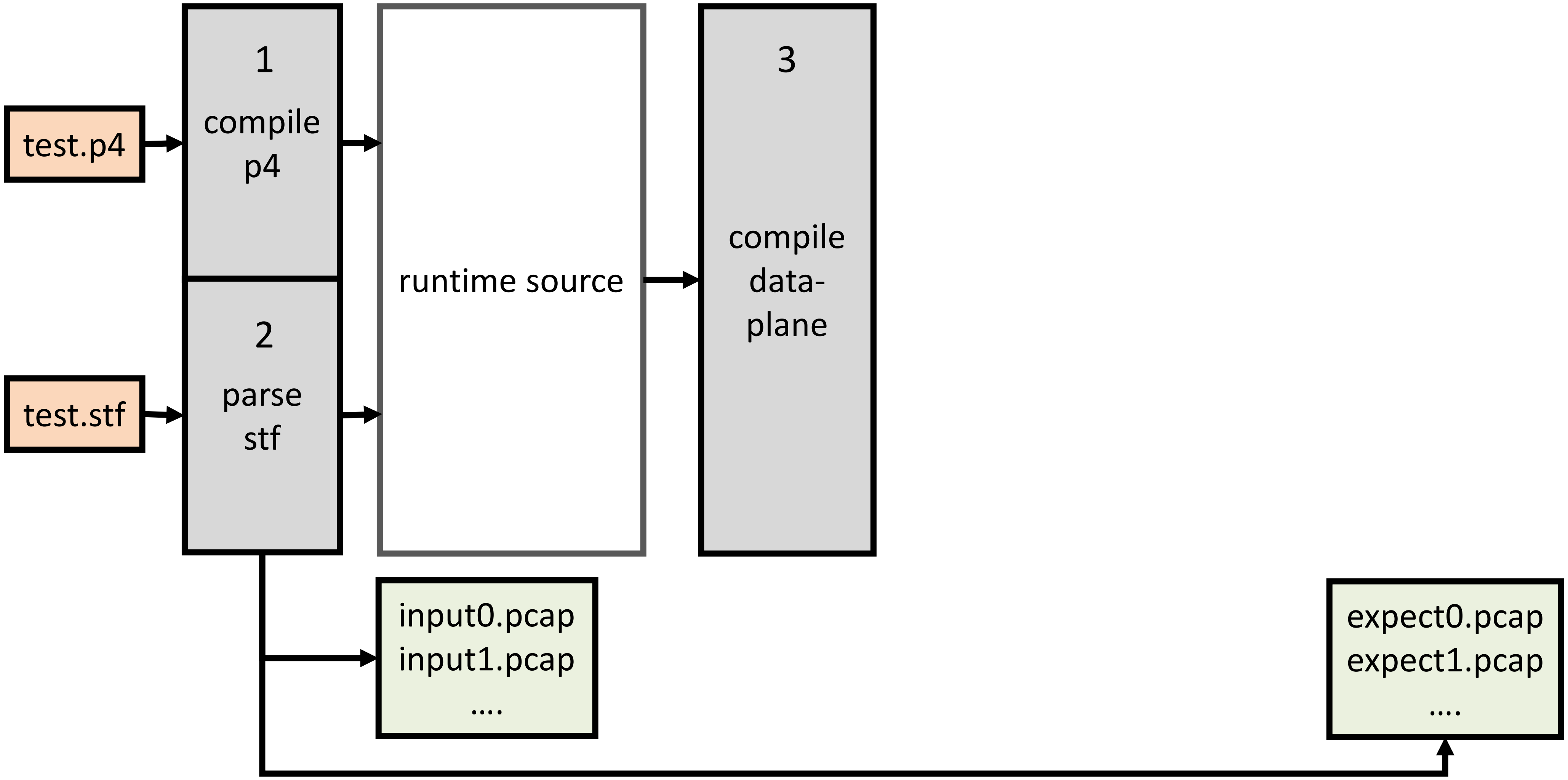
Five Testing Stages



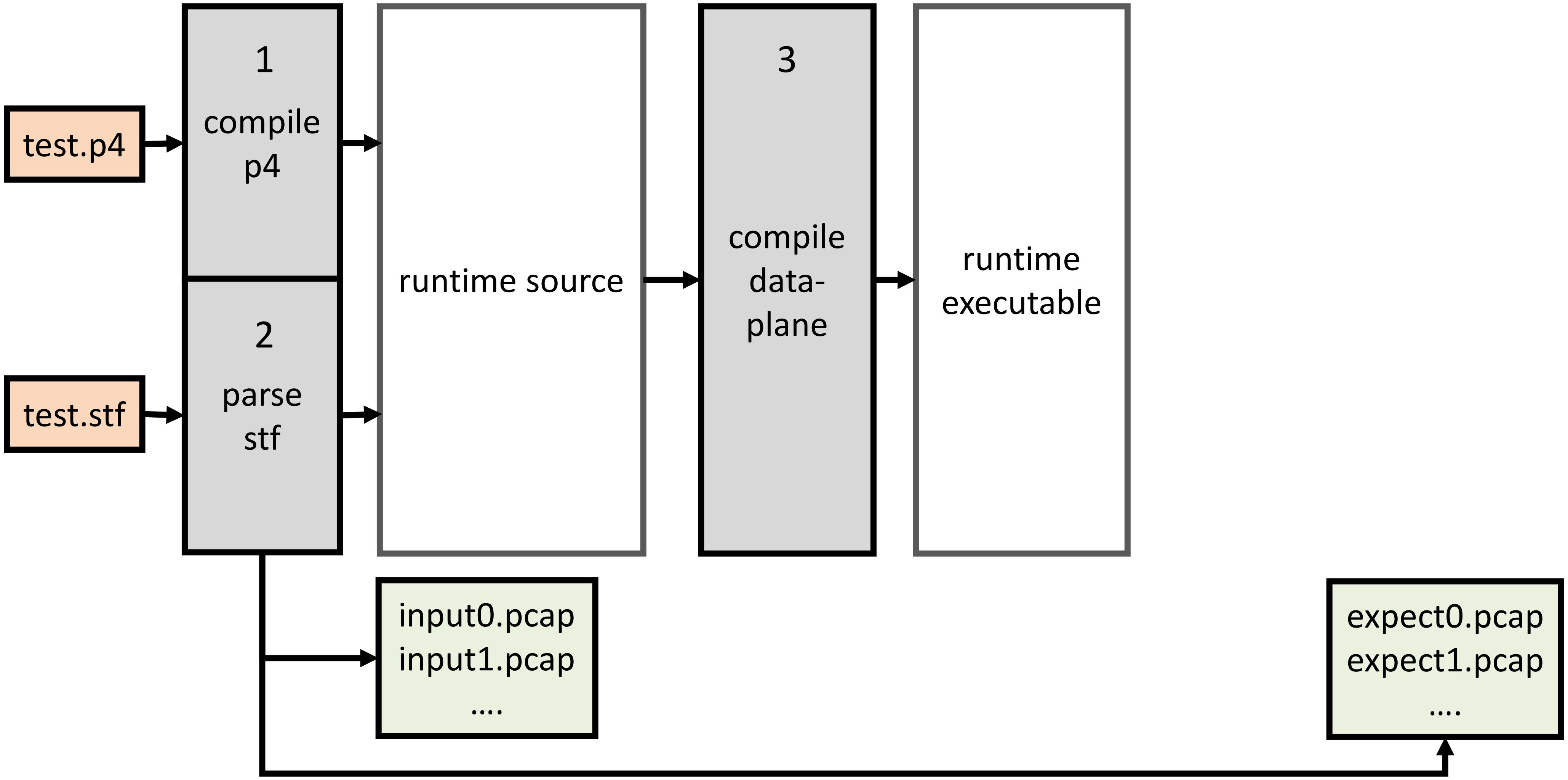
Five Testing Stages



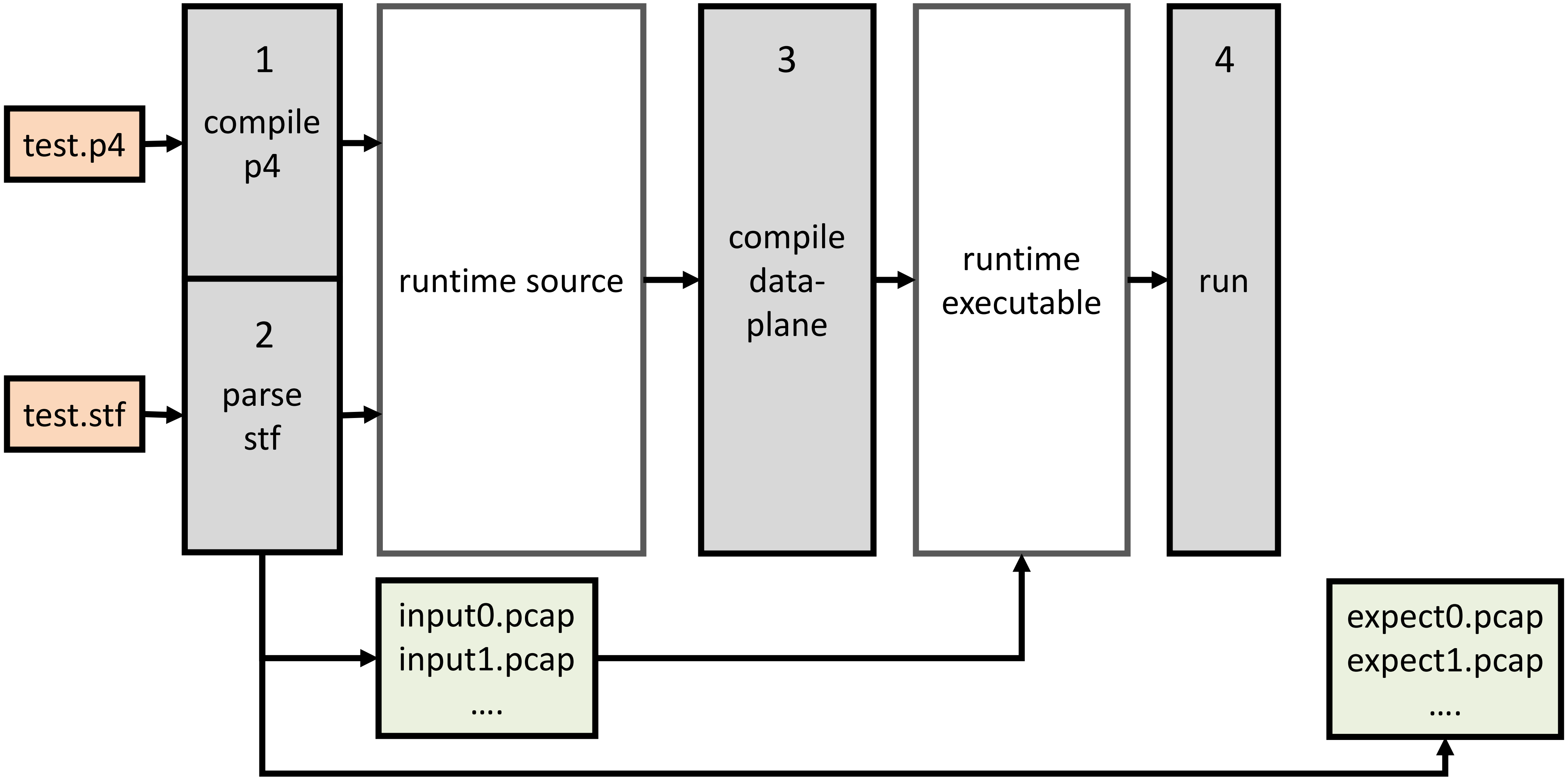
Five Testing Stages



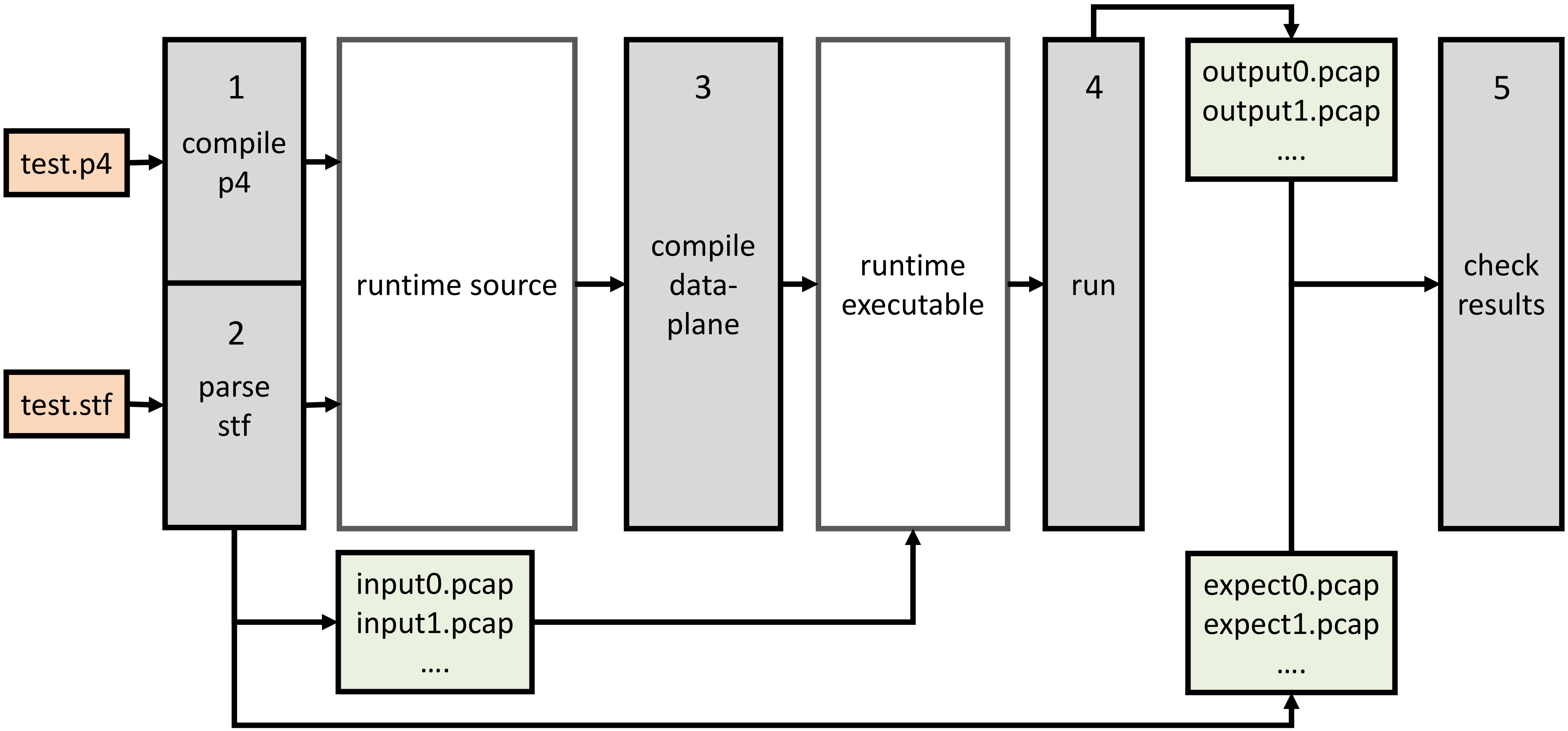
Five Testing Stages



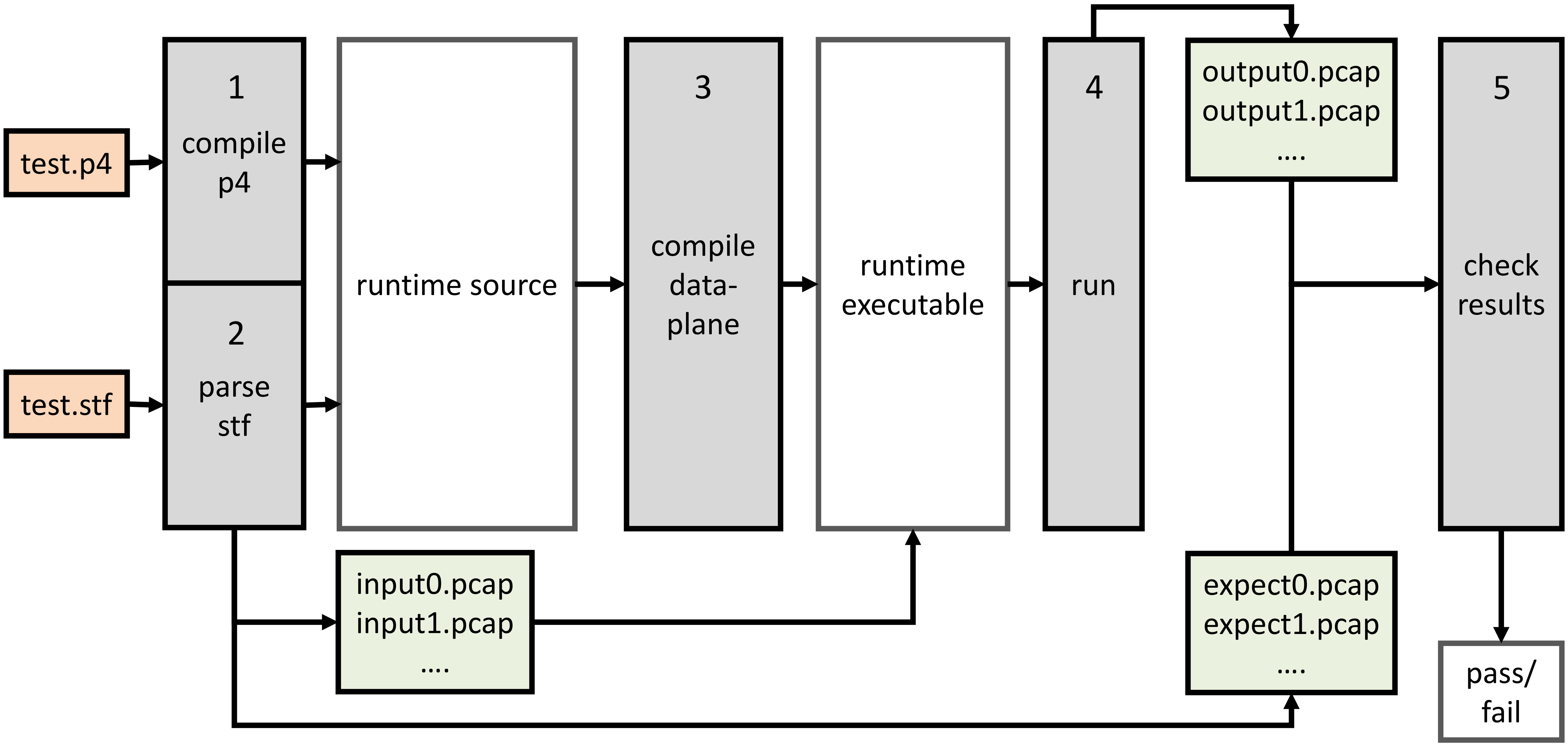
Five Testing Stages



Five Testing Stages



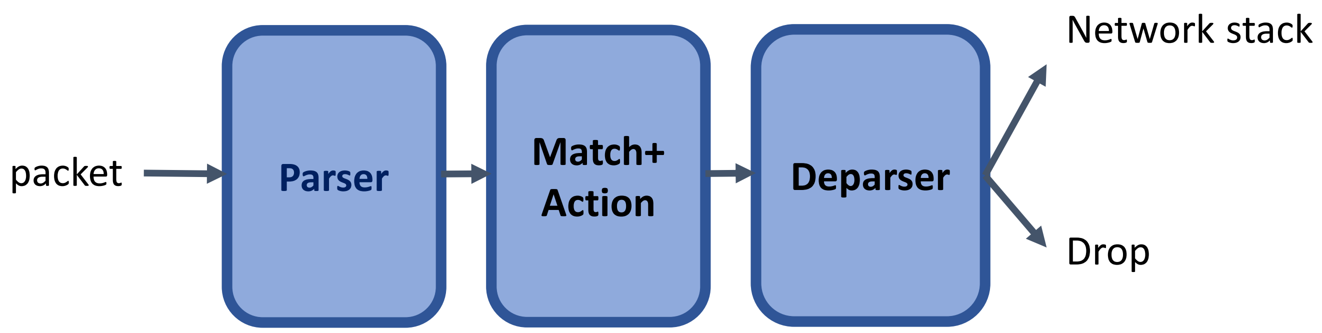
Five Testing Stages



A sample P4-XDP program

Forwarding an IPv4 Packet

- Parse Ethernet and IPv4 header
- Lookup a table using Ethernet’s destination as **key**
- Based on Ethernet’s destination address, execute one **action**:
  - Drop the packet (**XDP_DROP**)
  - Pass the packet to network stack (**XDP_PASS**)



P4 Headers

```
header Ethernet {
        bit<48> source;
        bit<48> dest;
        bit<16> protocol;
}
header IPv4{
        bit<4> version;
        bit<4> ihl;
        bit<8> diffserv;
        …
}
struct Headers {
        Ethernet eth;
        IPv4   ipv4;
}
```

P4 Headers

```
header Ethernet {
        bit<48> source;
        bit<48> dest;
        bit<16> protocol;
}

header IPv4{
        bit<4> version;
        bit<4> ihl;
        bit<8> diffserv;
        …
}

struct Headers {
        Ethernet eth;
        IPv4   ipv4;
}
```

p4c-xdp

```
struct Ethernet{
        u8 source[6];
        u8 destination[6];
        u16 protocol;
        u8 ebpf_valid;
}

struct IPv4 {
        u8 version[6]; /* bit<4> */
        u8 ihl[6];     /* bit<4> */
        u8 diffserv;   /* bit<8> */
```

C struct + valid bit

- Currently each header field is re-aligned
- Inefficient design

P4 Protocol Parser

```
parser Parser(packet_in packet, out Headers hd) {

   state start {
       packet.extract(hd.ethernet);
           transition select(hd.ethernet.protocol) {
               16w0x800: parse_ipv4;
               default: accept; }

   state parse_ipv4 {
       packet.extract(hd.ipv4);
           transition accept; }}
```

P4 Protocol Parser

```
parser Parser(packet_in packet, out Headers hd) {

   state start {
       packet.extract(hd.ethernet);
           transition select(hd.ethernet.protocol) {
               16w0x800: parse_ipv4;
               default: accept; }

   state parse_ipv4 {
       packet.extract(hd.ipv4);
           transition accept; }}
```

p4c-xdp

```
struct Headers hd = {};
      …
if (end < start + header_size)
      goto reject;
hd.ethernet.destination[0] = load_byte(…);
…
```

Match-Action

```
control Ingress (inout Headers hdr,
                 in xdp_input xin, out xdp_output xout) {
   action Drop_action() { xout.output_action = xdp_action.XDP_DROP; }
   action Fallback_action() { xout.output_action = xdp_action.XDP_PASS; }
   table mactable {
                 key = {hdr.ethernet.destination : exact; }
                 actions = {
                         Fallback_action;
                         Drop_action;
                 }
                 implementation = hash_table(64); } … }
```

Match-Action

```
control Ingress (inout Headers hdr,
                 in xdp_input xin, out xdp_output xout) {
   action Drop_action() { xout.output_action = xdp_action.XDP_DROP; }
   action Fallback_action() { xout.output_action = xdp_action.XDP_PASS; }
   table mactable {
              key = {hdr.ethernet.destination : exact; }
              actions = {
                      Fallback_action;
                      Drop_action;
              }
              implementation = hash_table(64); } … }
```

p4c-xdp

```
struct mactable_key {
        u8 field0[6];
}
enum mactable_actions {
        Fallback_action,
        Drop_action,
}
```

```
struct mactable_value {
        enum mactable_actions action;
        union {
                struct {
                } Fallback_action;
                struct {
                } Drop_action;
        } u;
}
```

Control-plane API in C

Generated by compiler

```
#include ”xdp1.h”
int main () {
        int fd = bpf_obj_get(MAP_PATH);
        …
        struct mactable_key key;
        memcpy(key.field0, MACADDR, 6);
        struct mactable_value value;
        value.action = Fallback_action;

        bpf_update_elem(fd, &key, &value, BPF_ANY);
}
```

Deparser: Update the Packet

```
control Deparser(in Headers hdrs,
packet_out packet) {
   apply {
      packet.emit(hdrs.ethernet);
      packet.emit(hdrs.ipv4); }}
```

Deparser: Update the Packet

```
control Deparser(in Headers hdrs,
packet_out packet) {
   apply {
      packet.emit(hdrs.ethernet);
      packet.emit(hdrs.ipv4); }}
```

p4c-xdp

```
bpf_xdp_adjust_head(skb, offset);
ebpf_byte = ((char*)(&hd.ethernet.destination))[0];
write_byte(ebpf_packetStart, BYTES(ebpf_packetOffsetInBits) + 0, ebpf_byte);
…
ebpf_packetOffsetInBits += 48;
```

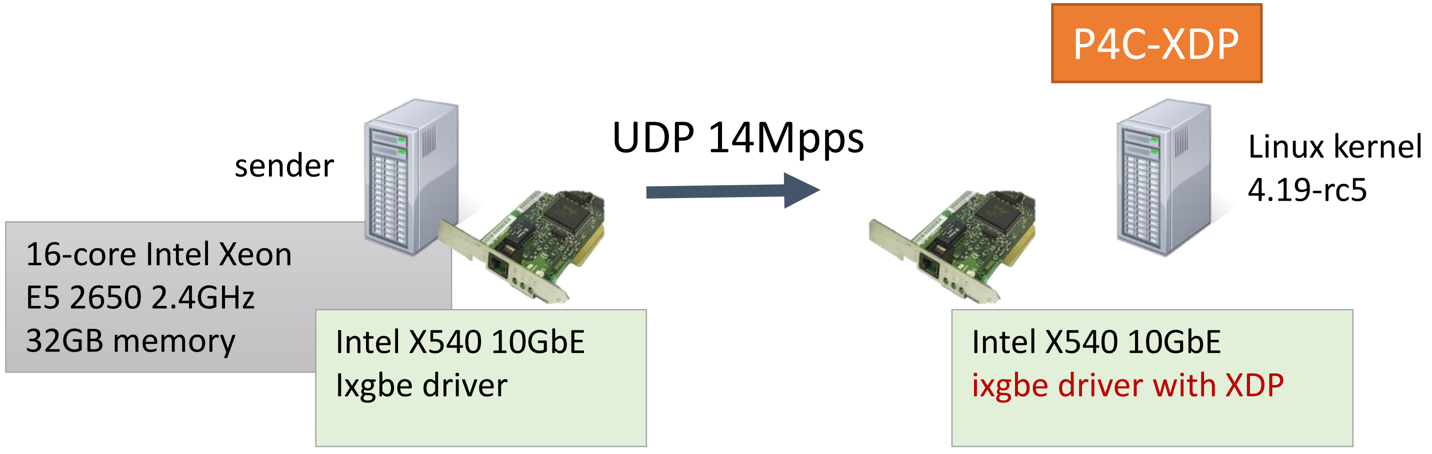
Complete C program structure

```
SEC("prog")
int ebpf_filter(struct xdp_md *skb) {
   struct Headers hd = {};
   …
   /* parser */
   if (end < start + header_size)
       goto reject;
   hd.ethernet.destination[0] = load_byte(…);
   …
   /* match+action*/
   value = bpf_map_lookup_elem(key);
   switch(value->action) {
       case Drop_action:
       …
   }
   /* deparser */
   xdp_adjust_head(amount);
   // update packet header
   return xout.xdp_output;
}
```

- Parser:
  - Check packet access boundary.
  - Walk through the protocol graph.
  - Save in "struct Headers hd."
- Match+Action:
  - Extract key from struct Headers
  - Lookup BPF hash map
  - Execute the correponding action
- Deparser
  - Convert headers back into a byte stream.
  - Only valid headers are emitted.

Performance Benchmarks

Performance Evaluation



- P4C-XDP binary
  - #./p4c-xdp --target xdp -o <**output_file**> <**input p4**>
  - Sample code at tests/xdp*.p4
  - Load to driver by: `ip link set dev eth0 xdp obj xdp1.o`
- Measure packet rate in Mpps
  - Packet drop rate (XDP_DROP) and transmit rate (XDP_TX)

Sample P4 Program Performance

- SimpleDrop: return XDP_DROP
- xdp1.p4: parse Ethernet/IPv4 header, deparse it, and drop.
- xdp3.p4: parse Ethernet/IPv4 header, lookup a MAC address in a map, deparse it, and drop.
- xdp6.p4: parse Ethernet/IPv4 header, lookup and get a new TTL value from eBPF map, set to IPv4 header, deparse it, and drop.
- Possible Optimization: avoid byte-order translation and unnecessary (de-)parsing

P4 Program	Performance (Mpps)	Possible Optimization
SimpleDrop	14.4	NA
xdp1	8.1	14
xdp3	7.1	13
xdp6	2.5	12

Limitations

Fundamental Limitations

Feature	P4	XDP
Loops	Parsers	Tail call
Nested headers	Bounded depth	Bounded depth
Multicast/broadcast	External	No
Packet segmentation	No	No
Packet reassembly	No	No
Timers/timeouts/aging	No	No
Queues	No	No
Scheduling	No	No
State	Registers/counters	Maps
Linear scans	No	No

Limitations of XDP

- No multi-/broadcast support
  - No ability to clone packets in XDP
- The stack size is too small
  - Complex pipelines are rejected by the verifier
- Generic XDP and TCP
  - TCP is ignored by the generic XDP driver
- eBPF maps cannot be pinned in network namespaces

Conclusion

- P4 is a language that defines data-path behavior
  - It generalizes to different architectures
  - Including the Linux kernel
- P4 can express XDP
  - High-level abstraction to C code
  - Generated code is performant but not optimal
  - Many future optimizations are possible
- P4 and XDP have similar limitations