intel®

connectivity academy by P4ica

intel
connectivity
academy
by P4ica

Control Plane APIs: The dark side of P4 Programming

Vladimir Gurevich, P4ica

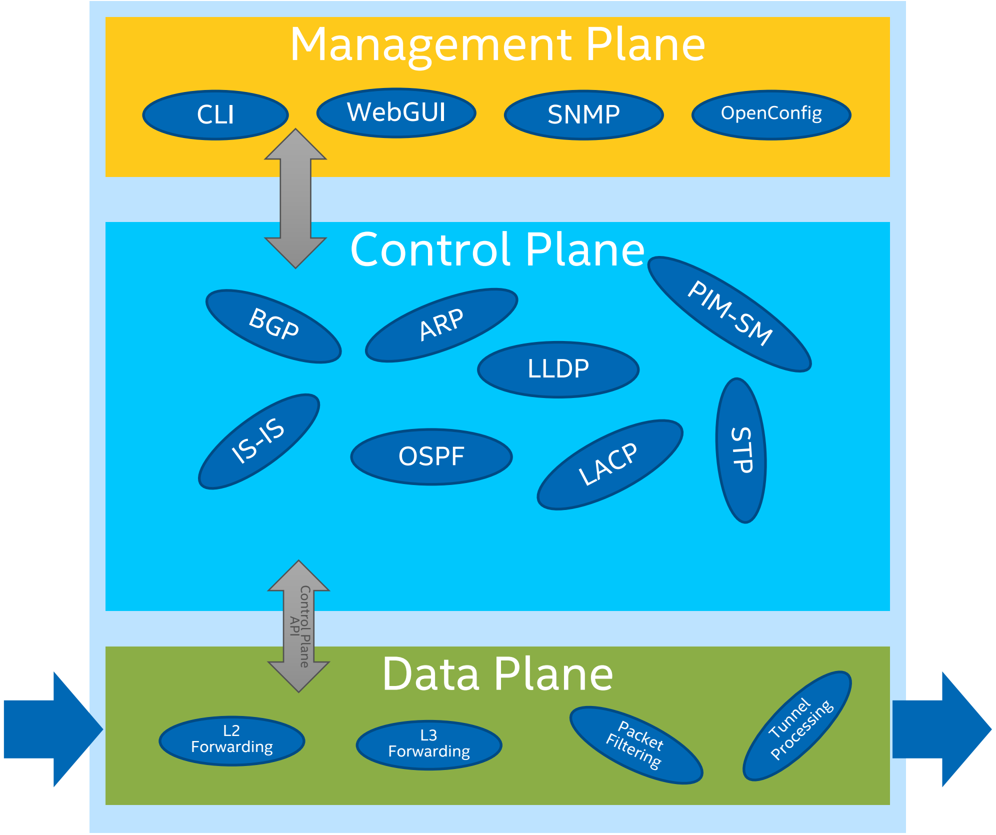
November 15, 2023

Agenda

- What are Control Plane APIs and why do we need them?
- How can we communicate with a data plane program?
- How should the APIs look like?
- Food for thought

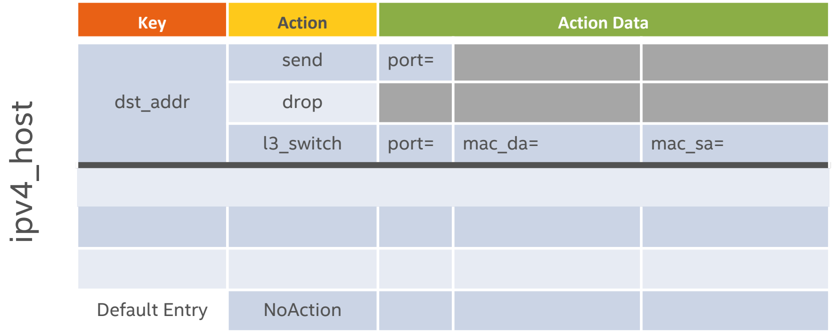
Standard Telecommunications System Architecture

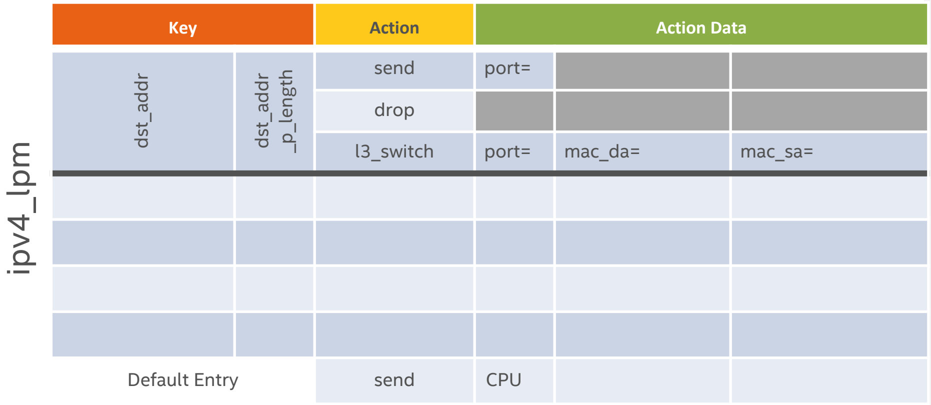
- **Three separate layers (planes)**
  - Data (Forwarding) Plane
  - Control Plane
  - Management (Configuration) Plane
- **What constitutes a "plane"**
  - The hardware
  - The algorithm
  - The interface
- **It is the data plane that ultimately determines the system performance and functionality**



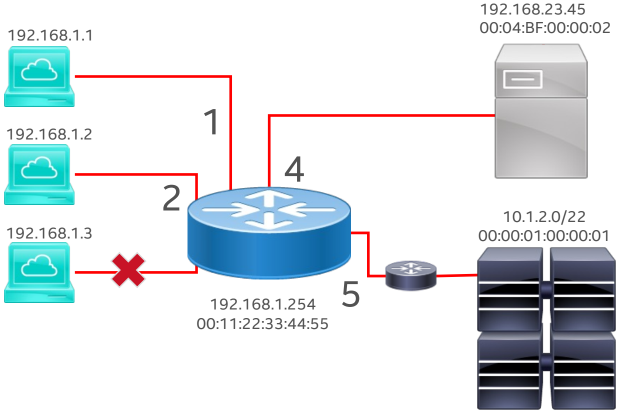
P4 programs only define table structure

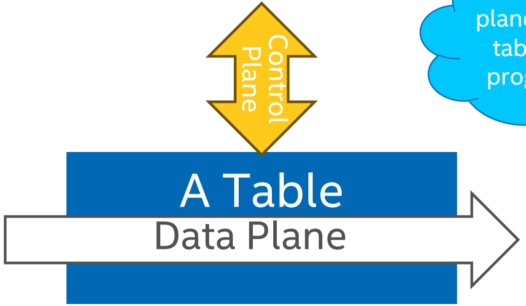
```
action send(PortId_t port) {
    ig_tm_md.ucast_egress_port = port;
}

action drop() {
    ig_dprsr_md.drop_ctl = 1;
}

action l3_switch(PortId_t port, bit<48> mac_da, bit<48> mac_sa)
{
    hdr.ethernet.dst_addr = mac_da;
    hdr.ethernet.src_addr = mac_sa;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    send(port);
}

table ipv4_host {
    key     = { hdr.ipv4.dst_addr : exact; }
    actions = { send; drop; l3_switch; }
    size    = 131072;
}

table ipv4_lpm {
    key     = { hdr.ipv4.dst_addr : lpm; }
    actions = { send; drop; l3_switch; }
    size    = 12288;
    default_action = send(CPU_PORT);
}
```

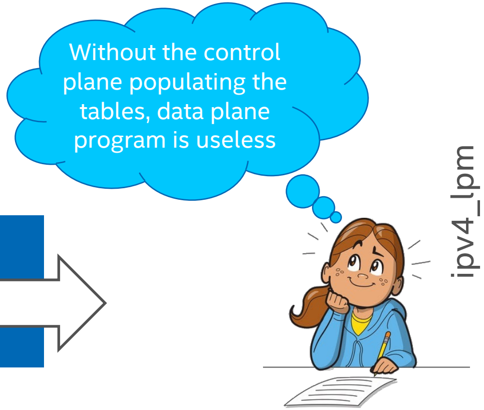




Tables are populated by the Control Plane







ipv4_host

Key	Action	Action Data		
dst_addr	send	port=		
	drop			
	l3_switch	port=	mac_da=	mac_sa=
192.168.1.1	send	1		
192.168.1.2	send	2		
192.168.1.3	drop			
Default Entry	NoAction			

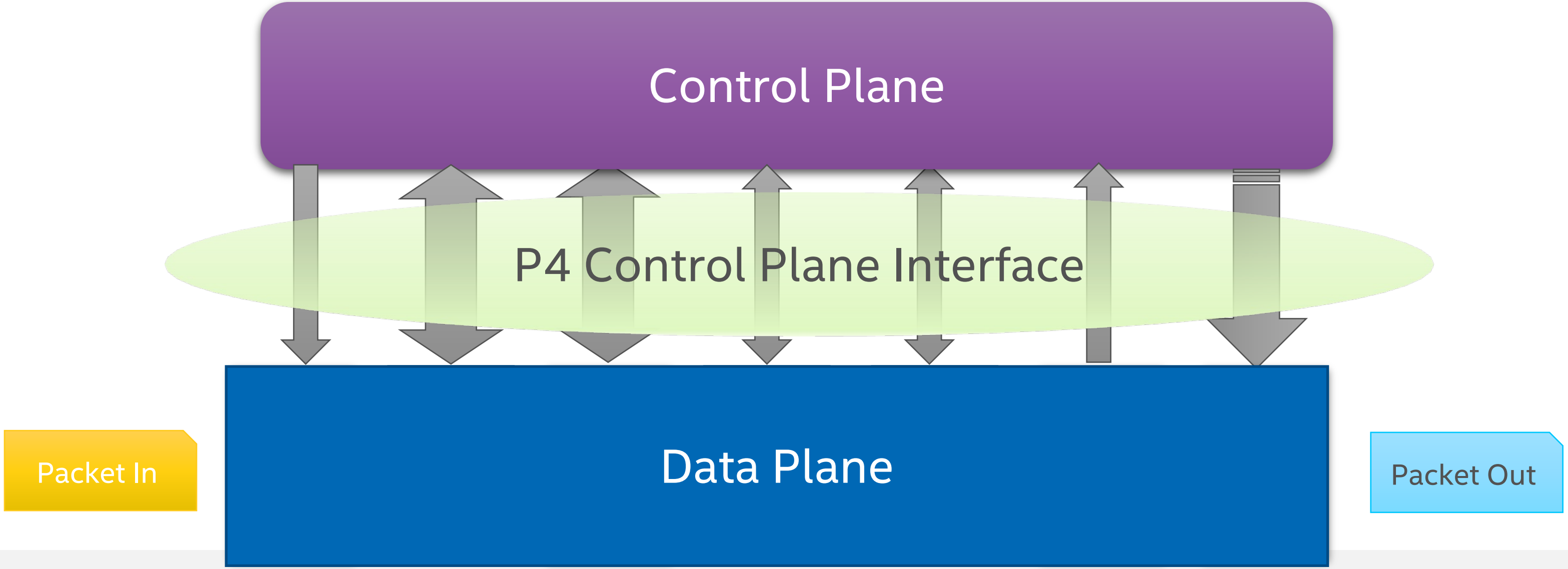
ipv4_lpm

Key		Action	Action Data		
dst_addr	dst_addr_p_length	send	port=		
		drop			
		l3_switch	port=	mac_da=	mac_sa=
192.168.1.1	24	send	CPU		
192.168.23.0	24	l3_switch	4	00:04:BF:00:00:02	00:11:22:33:44:55
10.1.2.0	22	l3_switch	5	00:00:01:00:00:01	00:11:22:33:44:55
Default Entry		send	CPU		

© P4ica, LLC 2023

Definition

- **P4 Control Program Interface is a set of methods that allow the Control Plane to manipulate or examine the state of**
  - All Stateful Objects, defined in P4-programmable blocks
  - All Stateful Objects in the fixed-logic (non-P4) blocks



Stateful Objects

P4 Objects

- **Tables**
- **Value Sets**
- **Externs**
  - Counters
  - Meters
  - Registers
  - Action Profiles
  - Action Selectors
  - Hashes and Hash Algorithms
  - …

Fixed Logic Blocks

- **Ports**
  - MAC
    - MAC counters
  - SerDes
- **Traffic Manager**
  - Memory Pools
  - Priority Groups
  - Queues
  - Schedulers
- **Replication Engine**
  - Multicast Groups

© P4ica, LLC 2023

Problems to Solve

- **How should the APIs look like?**
  - Program-dependent and program-independent approaches
  - Defining the APIs for everything that is not a table
- **How to access the target?**
  - PCIe primer
  - Locally callable APIs
  - Remotely callable APIs (RPC)

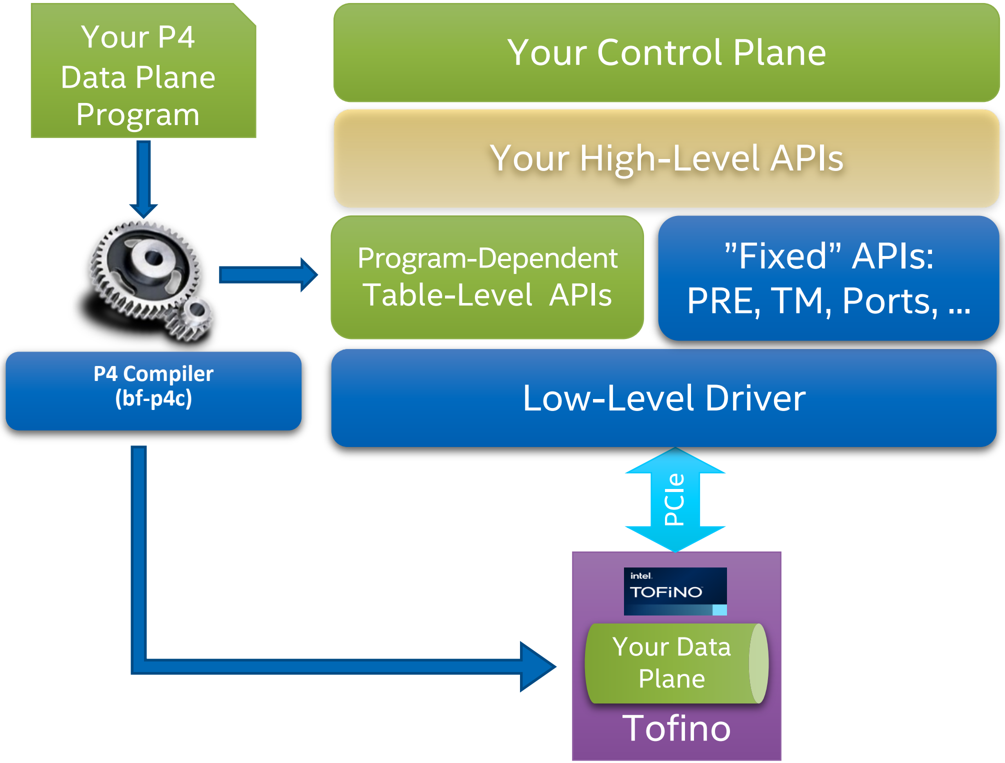
© P4ica, LLC 2023

intel connectivity academy by P4ica

Program-dependent and Program-independent APIs

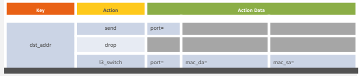
Program-Dependent (Autogenerated APIs)

- Device is accessed via PCIe
- Low-level driver provides basic access
- "Fixed" APIs are manually coded
- The compiler autogenerates APIs for each P4 program

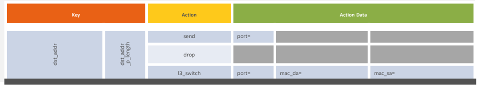


Automatically Generated Program-Dependent APIs (types)

```
action send(PortId_t port) {
    ig_tm_md.ucast_egress_port = port;
}

action drop() {
    ig_dprsr_md.drop_ctl = 1;
}

action l3_switch(PortId_t port, bit<48> mac_da, bit<48> mac_sa)
{
    hdr.ethernet.dst_addr = mac_da;
    hdr.ethernet.src_addr = mac_sa;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    send(port);
}

table ipv4_host {
    key     = { hdr.ipv4.dst_addr : exact; }
    actions = { send; drop; l3_switch; }
    size    = 131072;
}
```




```
table ipv4_lpm {
    key     = { hdr.ipv4.dst_addr : lpm; }
    actions = { send; drop; l3_switch; }
    size    = 12288;
    default_action = send(CPU_PORT);
}
```




```
/* Representing action data for each action */
typedef struct p4_pd_myprog_send_action_spec {
    uint16_t port;
} p4_pd_myprog_send_action_spec_t;

typedef struct p4_pd_myprog_drop_action_spec {
} p4_pd_myprog_drop_action_spec_t;

typedef struct p4_pd_myprog_l3_switch_action_spec {
    uint16_t port;
    uint64_t mac_da;
    uint64_t mac_sa;
} p4_pd_myprog_l3_switch_action_spec_t;


/* Representing table keys for each table */
typedef struct p4_pd_myprog_ipv4_host_match_spec {
    uint32_t dst_addr;
} p4_pd_myprog_ipv4_host_match_spec_t;

typedef struct p4_pd_myprog_ipv4_lpm_match_spec {
    uint32_t dst_addr;
    uint32_t dst_addr_p_length;
} p4_pd_myprog_ipv4_lpm_match_spec_t;
```

© P4ica, LLC 2023

Automatically Generated Program-Dependent APIs (Entry Add)

```
/* Representing action data for each action */
typedef struct p4_pd_myprog_send_action_spec {
    uint16_t port;
} p4_pd_myprog_send_action_spec_t;

typedef struct p4_pd_myprog_drop_action_spec {
} p4_pd_myprog_drop_action_spec_t;

typedef struct p4_pd_myprog_l3_switch_action_spec {
    uint16_t port;
    uint64_t mac_da;
    uint64_t mac_sa;
} p4_pd_myprog_l3_switch_action_spec_t;


/* Representing table keys for each table */
typedef struct p4_pd_myprog_ipv4_host_match_spec {
    uint32_t dst_addr;
} p4_pd_myprog_ipv4_host_match_spec_t;

typedef struct p4_pd_myprog_ipv4_lpm_match_spec {
    uint32_t dst_addr;
    uint32_t dst_addr_p_length;
} p4_pd_myprog_ipv4_lpm_match_spec_t;
```

```
/* Table ipv4_host */
p4_pd_status_t p4_pd_myprog_ipv4_host_table_add_with_send(
    p4_pd_dev_target_t                        device_number,
    const p4_pd_myprog_ipv4_host_match_spec_t * key,
    const p4_pd_myprog_send_action_spec_t     * data);

p4_pd_status_t p4_pd_myprog_ipv4_host_table_add_with_drop(
    p4_pd_dev_target_t                        device_number,
    const p4_pd_myprog_ipv4_host_match_spec_t * key);

p4_pd_status_t p4_pd_myprog_ipv4_host_table_add_with_l3_switch(
    p4_pd_dev_target_t                           device_number,
    const p4_pd_myprog_ipv4_host_match_spec_t  * key,
    const p4_pd_myprog_l3_switch_action_spec_t * data);

/* Table ipv4_lpm */
p4_pd_status_t p4_pd_myprog_ipv4_lpm_table_add_with_send(
    p4_pd_dev_target_t                        device_number,
    const p4_pd_myprog_ipv4_lpm_match_spec_t  * key,
    const p4_pd_myprog_send_action_spec_t     * data);

p4_pd_status_t p4_pd_myprog_ipv4_lpm_table_add_with_drop(
    p4_pd_dev_target_t                        device_number,
    const p4_pd_myprog_ipv4_lpm_match_spec_t  * key);

p4_pd_status_t p4_pd_myprog_ipv4_lpm_table_add_with_l3_switch(
    p4_pd_dev_target_t                           device_number,
    const p4_pd_myprog_ipv4_lpm_match_spec_t   * key,
    const p4_pd_myprog_l3_switch_action_spec_t * data);
```

© P4ica, LLC 2023

Adding and deleting a table entry

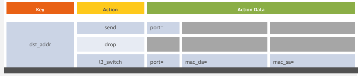
```
#include "p4_pd.h"

p4_pd_myprog_ipv4_host_match_spec_t key;
p4_pd_myprog_send_action_spec_t     data;

/* Prepare the key */
memset(&key, 0, sizeof(key));
key.dst_addr = 0xc0a80101;       // 192.168.1.1

/* Prepare the action data */
memset(&data, 0, sizeof(data));
data.port = 5;

/* Add an entry */
result =
    p4_pd_myprog_ipv4_host_table_add_with_send(0, &key, &data);

/* Delete an entry */
result =
    p4_pd_myprog_ipv4_host_table_delete(0, &key);
```

- **Very easy to use, compact APIs**
- **A lot of compile-time checks**
  - No way to specify a non-existing table
  - No way to specify a non-existing action
  - No way to specify wrong key or action data field
  - No way to specify incorrect action data for a given action
- **Retrieving an entry might be challenging**
  - We do not know what action data data structure to return

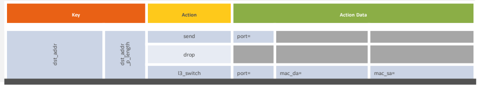
© P4ica, LLC 2023

Program-Independent APIs

```
action send(PortId_t port) {
    ig_tm_md.ucast_egress_port = port;
}

action drop() {
    ig_dprsr_md.drop_ctl = 1;
}

action l3_switch(PortId_t port, bit<48> mac_da, bit<48> mac_sa)
{
    hdr.ethernet.dst_addr = mac_da;
    hdr.ethernet.src_addr = mac_sa;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    send(port);
}

table ipv4_host {
    key     = { hdr.ipv4.dst_addr : exact; }
    actions = { send; drop; l3_switch; }
    size    = 131072;
}
```




```
table ipv4_lpm {
    key     = { hdr.ipv4.dst_addr : lpm; }
    actions = { send; drop; l3_switch; }
    size    = 12288;
    default_action = send(CPU_PORT);
}
```




```
#include "p4.h"

p4_key_t  key;      /* Abstract, opaque type */
p4_data_t data;     /* Abstract, opaque type */

p4_key_init(key, "ipv4_host");
p4_key_field_set_exact(key, "dst_addr", 0xc0a80101);

p4_data_init(data, "ipv4_host", "send");
p4_data_field_set_exact(data, 5);

/* Adding an entry */
result = p4_table_entry_add(0, "ipv4_host", key, data);

/* Deleting an entry */
result = p4_table_entry_del(0, "ipv4_host", "port", key);

/* Retrieving an entry */
result = p4_table_entry_get(0, "ipv4_host", key, &data);

printf("Action: %s\n", p4_data_action_get(data));
for (int i = 0; i < p4_action_param_num_get(data); i++) {
    printf("%s: %d\n",
           p4_action_param_name_get(data, i),
           p4_action_param_value_get(data, i));
}
```

© P4ica, LLC 2023

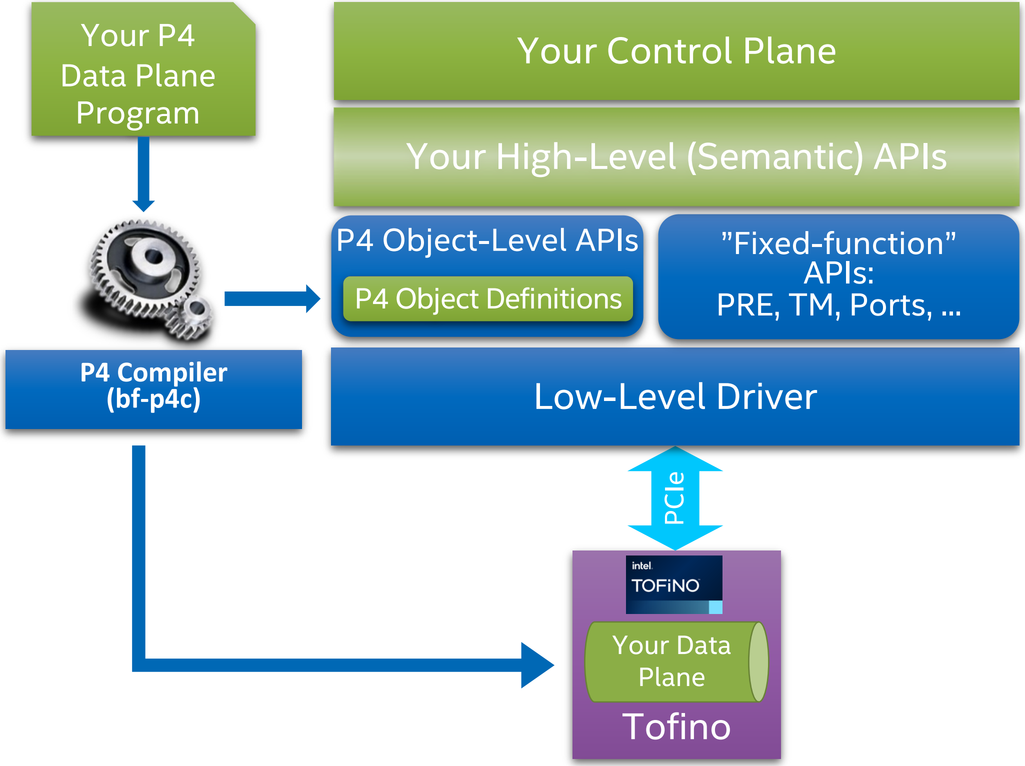
Program-Independent APIs (Analysis)

```
#include "p4.h"

p4_key_t  key;      /* Abstract, opaque type */
p4_data_t data;     /* Abstract, opaque type */

p4_key_init(key, "ipv4_host");
p4_key_field_set_exact(key, "dst_addr", 0xc0a80101);

p4_data_init(data, "ipv4_host", "send");
p4_data_field_set_exact(data, "port", 5);

/* Adding an entry */
result = p4_table_entry_add(0, "ipv4_host", key, data);

/* Deleting an entry */
result = p4_table_entry_del(0, "ipv4_host", key);

/* Retrieving an entry */
result = p4_table_entry_get(0, "ipv4_host", key, &data);

printf("Action: %s\n", p4_data_action_get(data));
for (int i = 0; i < p4_action_param_num_get(data); i++) {
    printf("%s: %d\n",
           p4_action_param_name_get(data, i),
           p4_action_param_value_get(data, i));
}
```

- Easy to use. More "verbose" APIs
- No compile time checks
  - Easy to specify a non-existing table/action/key/data field
- Not as efficient (string searches)
  - Fixed with assigned IDs
    - Also helps with typos
- Implementing generic algorithms becomes easy
- No need to have NxM functions
- No need to recompile control plane code
- Can be used to generate PD APIs
  - But not vice-versa

Program-Independent (Autogenerated Information) APIs

- Device is accessed via PCIe
- Low-level driver provides basic access
- “Fixed” APIs are manually coded
- APIs for P4 objects are generic
- The compiler autogenerates P4 object descriptions



© P4ica, LLC 2023

intel. connectivity academy by P4ica

What about other objects?

Specialized Program-Dependent APIs

```
Meter<bit<10>>(1024, MeterType_t.BYTES) acl_meter;


typedef struct p4_pd_meter_byte_spec {
  uint64_t cir_rate_bps;
  uint64_t cir_burst_kbits;
  uint64_t pir_rate_bps;
  uint64_t pir_burst_kbits;
} p4_pd_meter_byte_spec_t;
```

```
p4_pd_status_t p4_pd_myprog_meter_set_acl_meter(
    p4_pd_dev_target_t          device_number,
    uint32_t                    meter_index,
    p4_pd_meter_byte_spec_t     meter_byte_spec);

p4_pd_status_t p4_pd_myprog_meter_get_acl_meter(
    p4_pd_dev_target_t          device_number,
    uint32_t                    meter_index,
    p4_pd_meter_byte_spec_t   * meter_byte_spec);
```

© P4ica, LLC 2023

Specialized Program-Dependent APIs

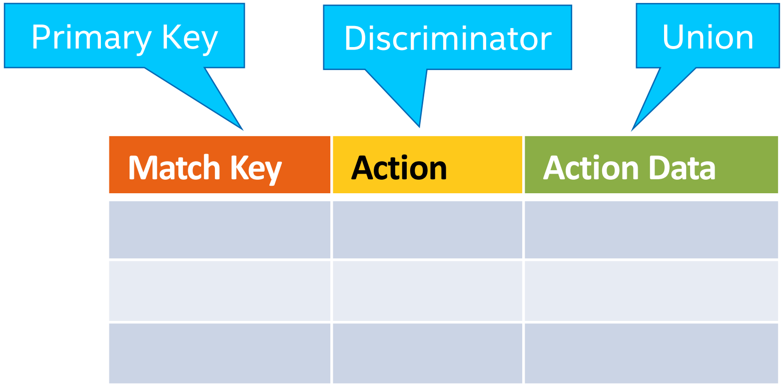
```
Counter<bit<64>, bit<12>>(4096,
      CounterType_t.PACKETS_AND_BYTES) ipv4_stats;


typedef struct p4_pd_counter_value {
  uint64_t packets;
  uint64_t bytes;
} p4_pd_counter_value_t;
```

```
p4_pd_status_t p4_pd_myprog_counter_set_ipv4_stats (
    p4_pd_dev_target_t       device_number,
    uint32_t                 counter_index,
    p4_pd_counter_value_t    counter_value);

p4_pd_status_t p4_pd_myprog_counter_get_ipv4_stats (
    p4_pd_dev_target_t       device_number,
    uint32_t                 counter_index,
    p4_pd_counter_value_t  * counter_value);

p4_pd_status_t p4_pd_myprog_counter_set_range_ipv4_stats (
    p4_pd_dev_target_t       device_number,
    uint32_t                 counter_index,
    uint32_t                 num_entries,
    p4_pd_counter_value_t    counter_value);

p4_pd_status_t p4_pd_myprog_counter_get_range_ipv4_stats (
    p4_pd_dev_target_t       device_number,
    uint32_t                 counter_index,
    uint32_t                 num_entries,
    p4_pd_counter_value_t  * counter_value);

p4_pd_status_t p4_pd_myprog_counter_clear_ipv4_stats (
    p4_pd_dev_target_t       device_number);


p4_pd_status_t p4_pd_myprog_counter_sync_ipv4_stats (
    p4_pd_dev_target_t       device_number);
```

© P4ica, LLC 2023

Logical Representation of Match-Action Tables

```
action send(PortId_t port) {
    ig_tm_md.ucast_egress_port = port;
}

action drop() {
    ig_dprsr_md.drop_ctl = 1;
}

action l3_switch(PortId_t port, bit<48> mac_da, bit<48> mac_sa)
{
    hdr.ethernet.dst_addr = mac_da;
    hdr.ethernet.src_addr = mac_sa;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    send(port);
}

table ipv4_host {
    key     = { hdr.ipv4.dst_addr : exact; }
    actions = { send; drop; l3_switch; }
    size    = 131072;
}

table ipv4_lpm {
    key     = { hdr.ipv4.dst_addr : lpm; }
    actions = { send; drop; l3_switch; }
    size    = 12288;
    default_action = send(CPU_PORT);
}
```



ipv4_host entry format

Key	Action	Action Data		
dst_addr	send	port=		
	drop			
	l3_switch	port=	mac_da=	mac_sa=

ipv4_lpm entry format

Key		Action	Action Data		
dst_addr	dst_addr_p_length	send	port=		
		drop			
		l3_switch	port=	mac_da=	mac_sa=

© P4ica, LLC 2023

Representing Specialized Objects (Indirect Externs)

```
Counter<bit<64>, bit<12>>
        (4096, CounterType_t.PACKETS_AND_BYTES) ipv4_stats;

Meter<bit<10>>(ACL_METER_SIZE, MeterType_t.BYTES) acl_meter;
```

- **The index is the primary key**
  - Like an exact match table
- **Action might be optional**
- **All entries might already in the table**
  - No add/delete

Key	(Volatile) Entry Data	
$COUNTER_INDEX	$COUNTER_SPEC_BYTES	$COUNTER_SPEC_PKTS

The same approach can be used to represent any configuration!

Key	Entry Data			
$METER_INDEX	$METER_SPEC_CIR_KBPS	$METER_SPEC_CIR_KBITS	$METER_SPEC_PIR_KBPS	$METER_SPEC_PIR_KBITS

That's exactly the idea behind Barefoot Runtime Interface (BRI)

Representing Fixed-Function Components

port.port

Key	Entry Data								
$DEV_PORT	$SPEED=	$FEC=	$N_LANES=	$ENABLE=	$AUTONEG=	$TX_MTU=	$RX_MTU	$PORT_UP	...

This field is volatile

mirror.cfg

Key	Action	Entry Data						
$sid	normal	$session_enable	$direction	$ucast_egress_port				
	coalescing	$session_enable	$direction	$ucast_egress_port				

tf1.tm.queue.sched_cfg

Key		Entry Data					
pg_id	pg_queue	min_priority	min_rate_enable	dwrr_weight	max_priority	max_rate_enable	scheduling_enable

- **Fixed-function components are represented as tables too**

© P4ica, LLC 2023

Ascribing an API to an arbitrary extern

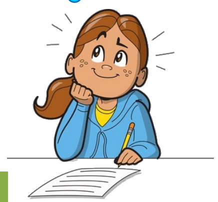
```
enum e1_t { e1_value1, e1_value2 }
enum e2_t { e2_value1, e2_value2, e2_value3 }

extern ext<S> {
    ext(bit<32> param1, e1_t param2);
    e2_t method1(in S param1, in e2_t param2);
    e2_t method1(in S param1);
}

enum MeterType_t  { PACKETS, BYTES }
enum MeterColor_t { RED, GREEN, YELLOW }

extern Meter<S> {
    Meter(bit<32> n_meters, MeterType_t type);
    MeterColor_t execute(in S index, in MeterColor_t color);
    MeterColor_t execute(in S index);
}
```

Key	Entry Data			
$METER_INDEX	$METER_SPEC_CIR_KBPS	$METER_SPEC_CIR_KBITS	$METER_SPEC_PIR_KBPS	$METER_SPEC_PIR_KBITS

Key	Action	Entry Data			
$INDEX	byte_metering	$CIR	$CBS	$PIR	$PBS
	packet_metering	$CIR	$CBS	$PIR	$PBS

- What is the table format for the extern “ext”?
- Oh, that’s a Meter!

We can not currently derive the APIs or table formats based on P4 code alone



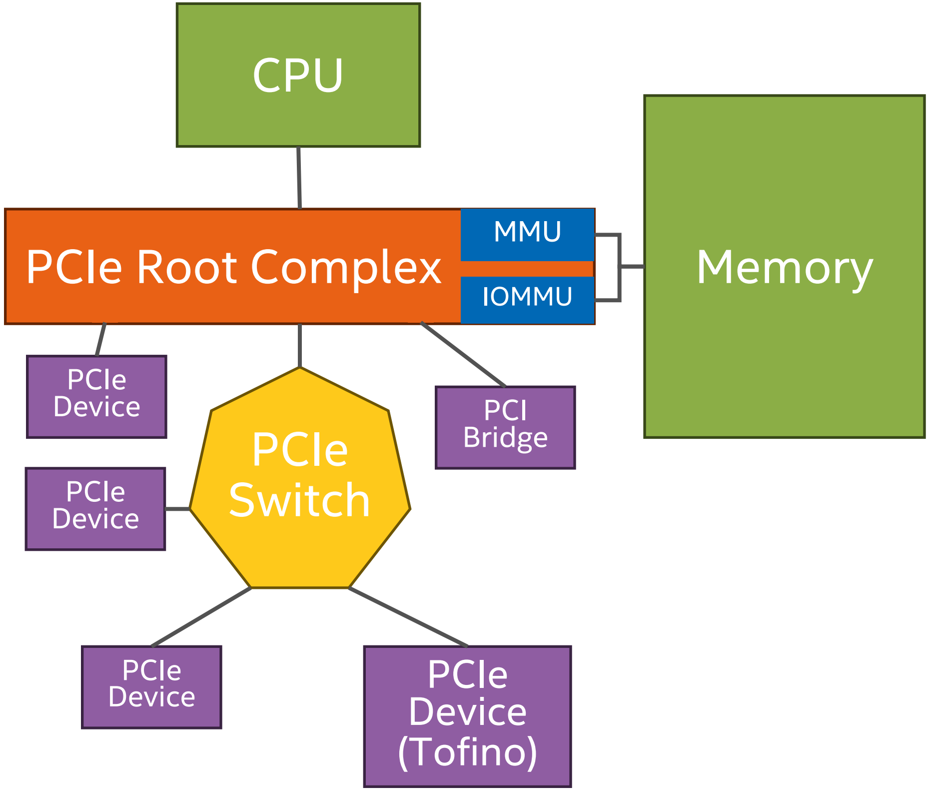
We need to invent a new specialized (sub) language to go with P4!

© P4ica, LLC 2023

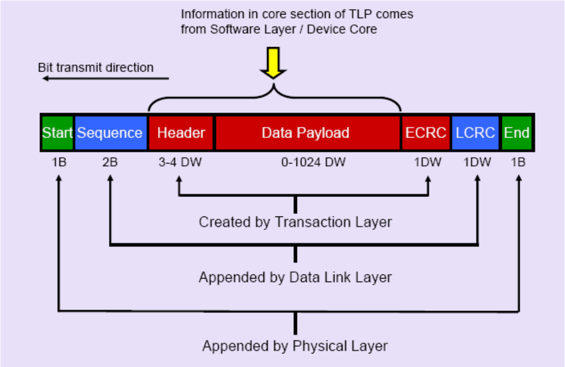
intel. connectivity academy by P4ica

Building a full API stack

PCI Express – the network inside your computer

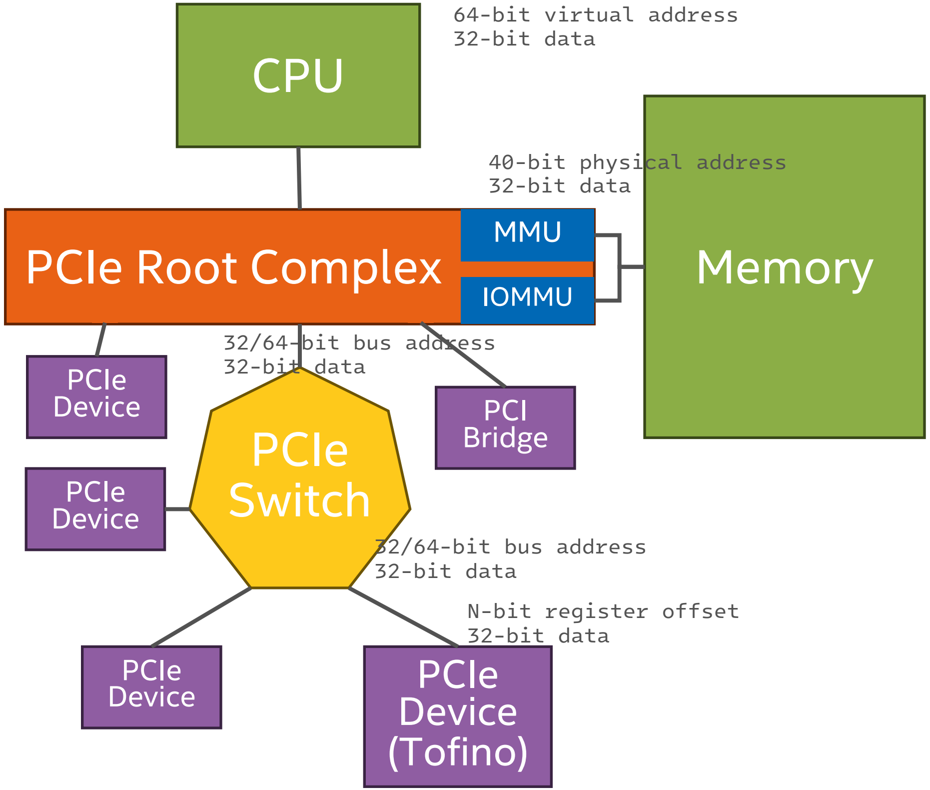


Version	Line code	Transfer rate per lane	Throughput (GB/s)		
			x1	x4	x16
1.0	8b/10b	2.5 GT/s	0.250	1.000	4.000
2.0		5.0 GT/s	0.500	2.000	8.000
3.0	128b/130b	8.0 GT/s	0.985	3.938	15.754
4.0		16.0 GT/s	1.969	7.877	31.508
5.0		32.0 GT/s	3.938	15.754	63.015



© P4ica, LLC 2023

PCI Express – the network inside your computer

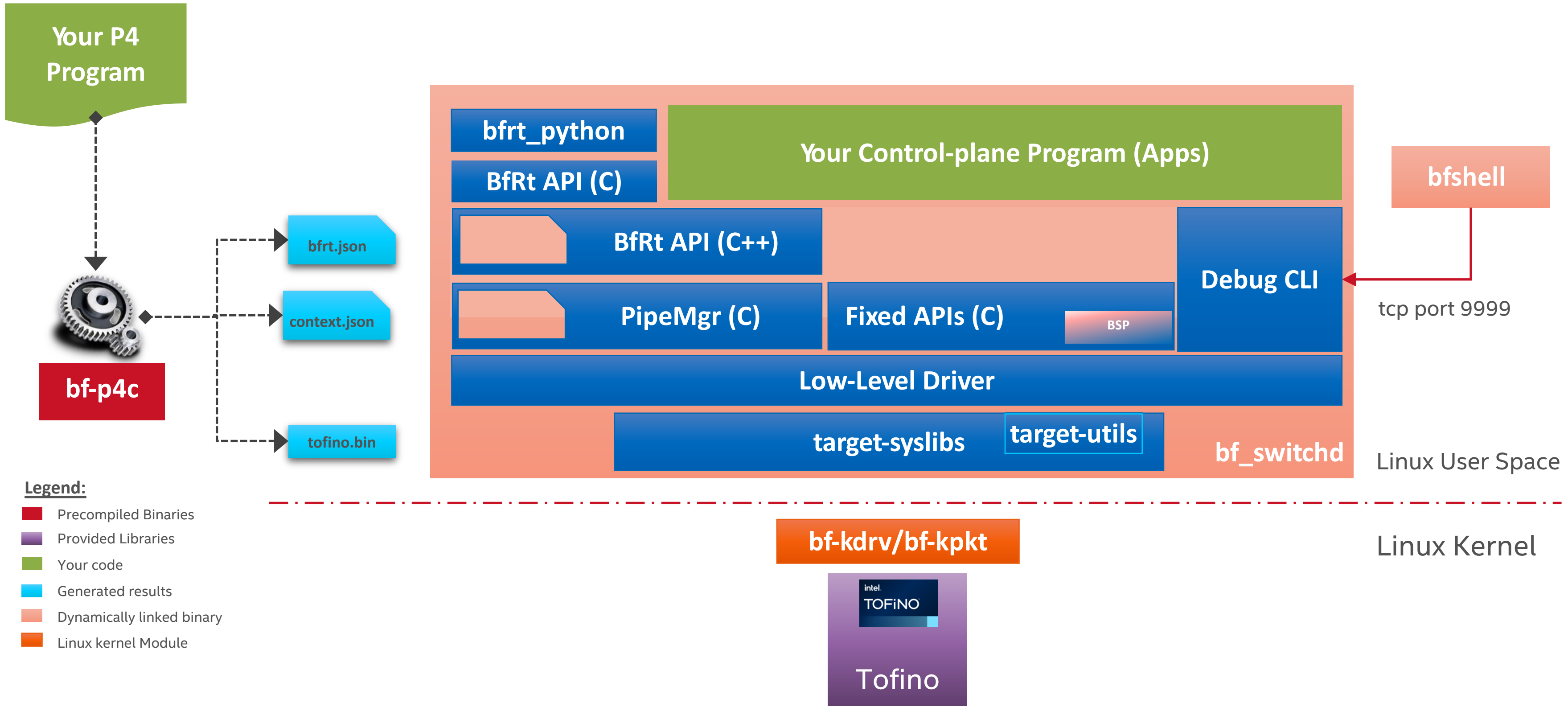


- Both memory and device registers are accessed using the same address space
- Pointers hold the addresses
- Based on the address the packet is routed to the proper place

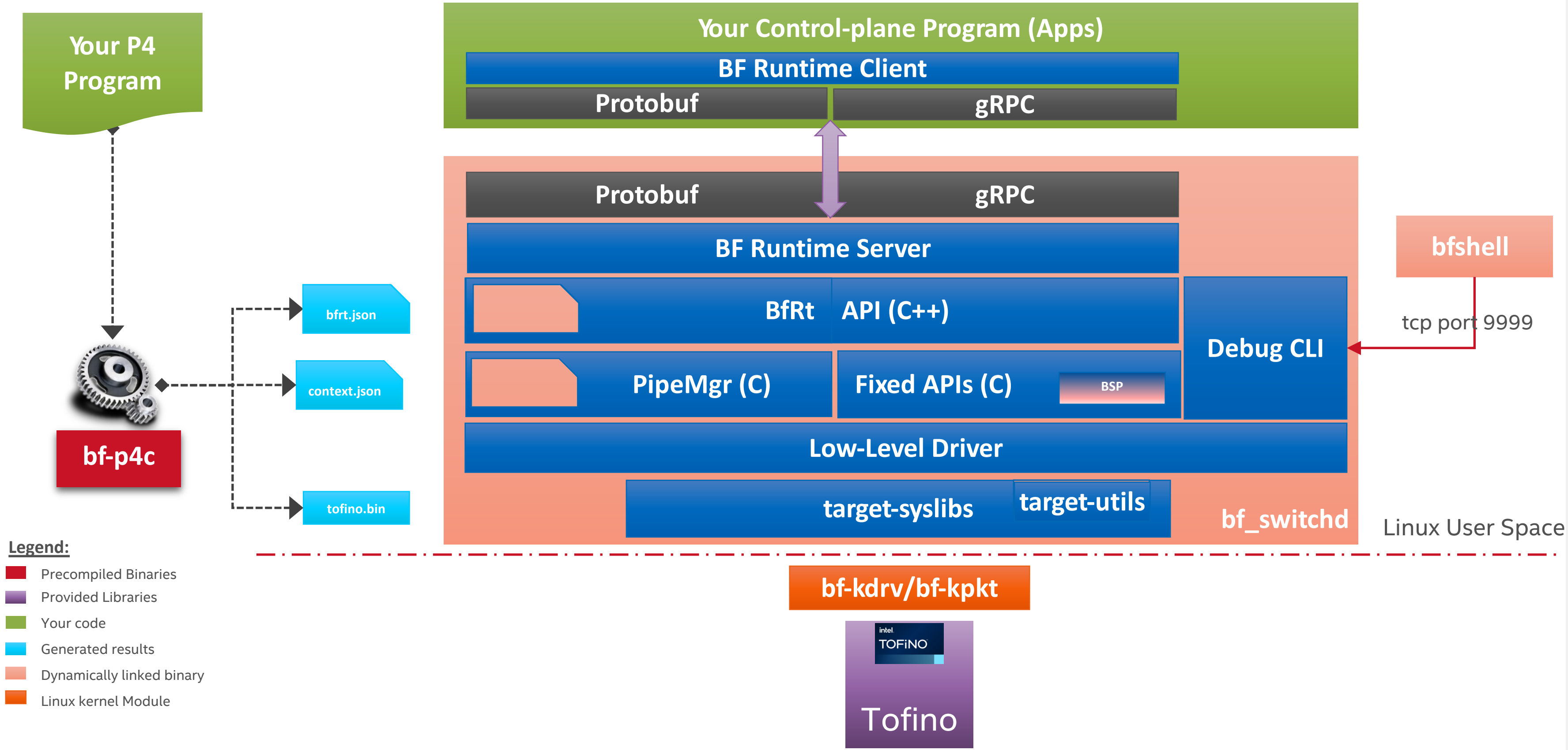
```c
#include <stdint.h>

typedef uint64_t my_reg_t;
typedef volatile my_reg_t * my_reg_addr_t;

void my_reg_write(my_reg_addr_t addr, my_reg_t val)
{
    *addr = val;    // movl %esi, (%rdi)
}

my_reg_t my_reg_read(my_reg_addr_t addr)
{
    return *addr;  //  movl (%rdi), %eax
}
```

© P4ica, LLC 2023

Core Components (Tofino). Single Process



Full Stack (Tofino ASIC). Remote Program Load



What Have We Learned?

- **P4 is just a part of the solution**
- **Control plane APIs are essential part of the data plane**
- **There are many ways to define control plane APIs**
  - Program-dependent vs. program-independent
  - Specialized vs. generic (table-like)
  - Locally and remotely callable
- **We still need to design a methodology for true automatic API generation for P4**

© P4ica, LLC 2023

Useful Materials (Click on the Links)

- **P4ica**
  - Website (under construction)
  - Support Portal (Course Materials and Recordings)
  - Eventbrite (Course signups)
- **P4.org**
  - Website Forum Github YouTube
- **Intel Connectivity Research Program**
  - Website Forum P4 Paper Collection (no longer updated)
- **Selected public videos of Vladimir, teaching P4**
- **A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research**

© P4ica, LLC 2023

Q &A

Intel Confidential . © Intel 2017-2022, P4ica LLC, 2023-

Thank You!

Intel Confidential . © Intel 2017-2022, P4ica LLC, 2023-

intel®
connectivity
academy
by P4ica