

# EN.601.414/614 Computer Networks

## Programmable Networks

Xin Jin

Spring 2019 (MW 3:00-4:15pm in Shaffer 301)



<https://github.com/xinjin/course-net>

Today, we will take a peek at the forefront of contemporary computer networking research...

- **An exciting journey**
  - One of the hottest topic in today's networking research
- **Yet, might be a little overwhelming**
  - Don't worry if you do not fully understand it
  - Raise your hand and ask questions

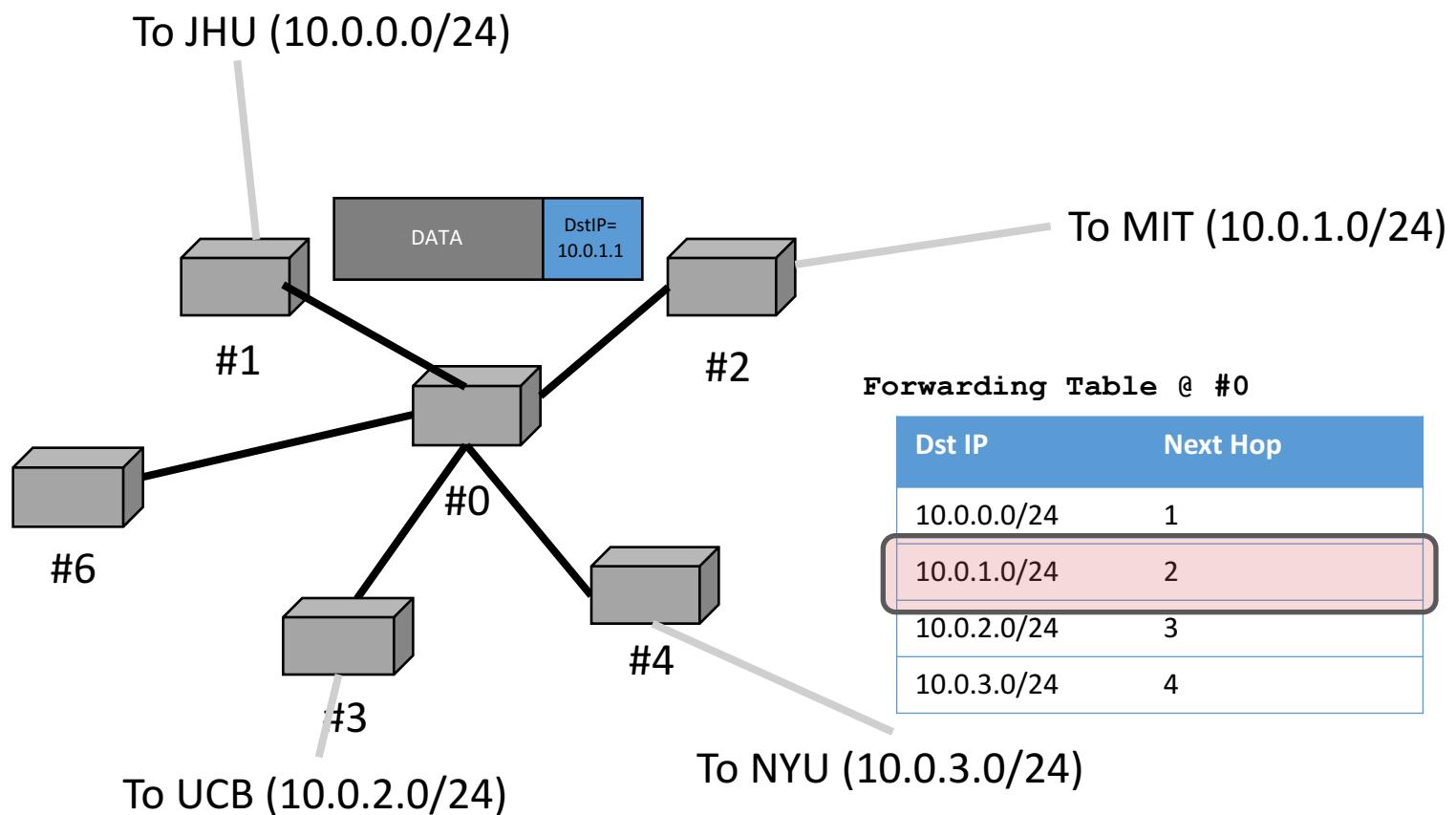
# Agenda

- **Programmable Networks**
- **Tutorial on Assignment 4**

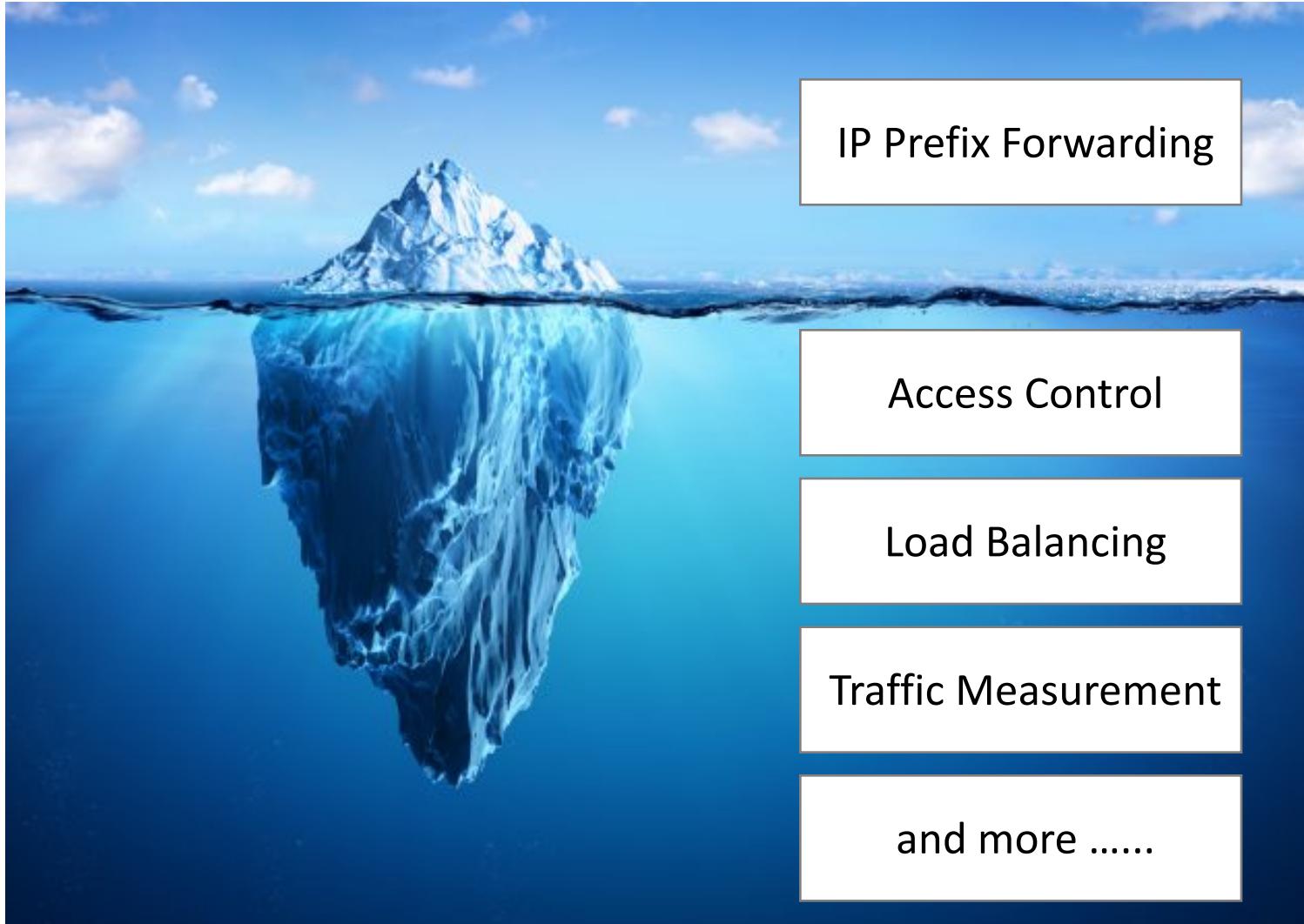
# Recap: Forwarding vs. routing

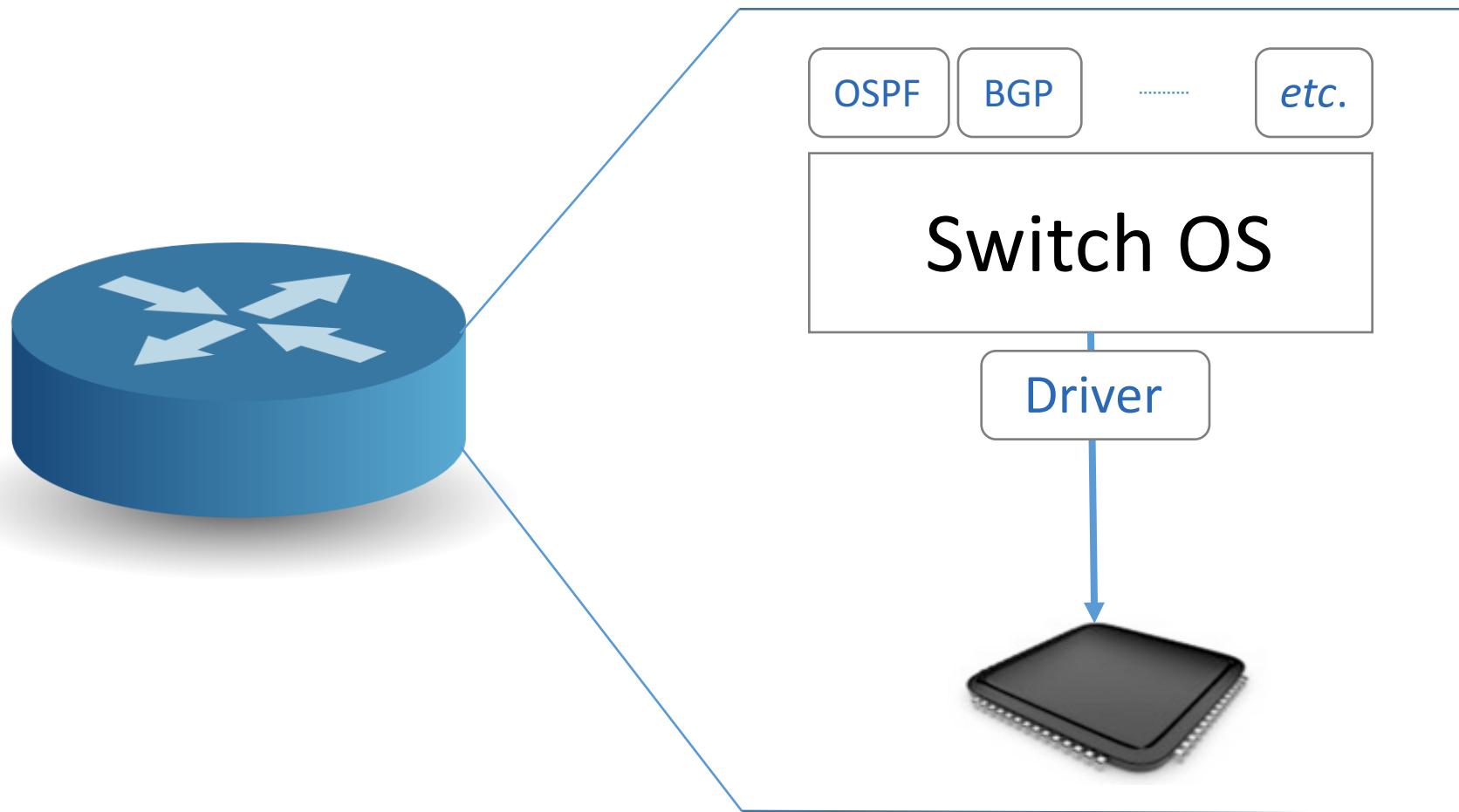
- **Forwarding:** “**data plane**”
  - Directing one data packet
  - Each router using local routing state
- **Routing:** “**control plane**”
  - Computing the forwarding tables that guide packets
  - Jointly computed by routers using a distributed algorithm
- **Very different timescales!**

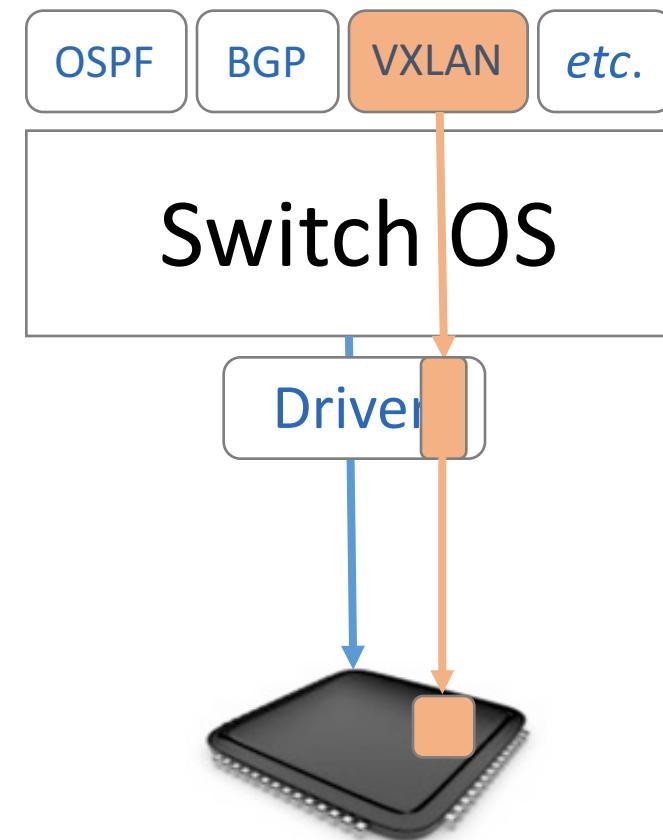
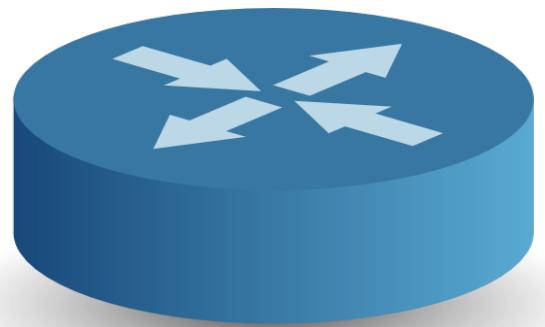
# Recap: Forwarding

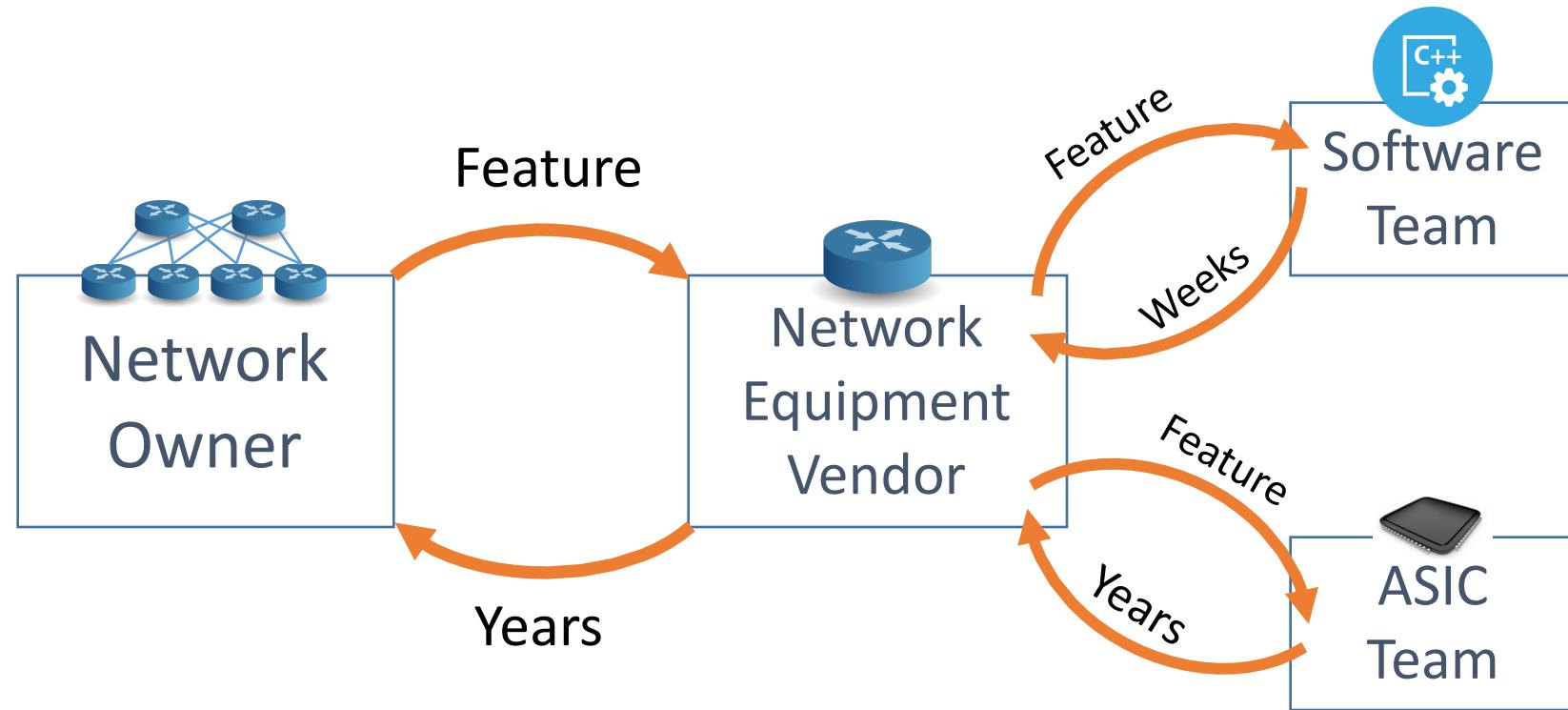


# Many functionalities in the data plane







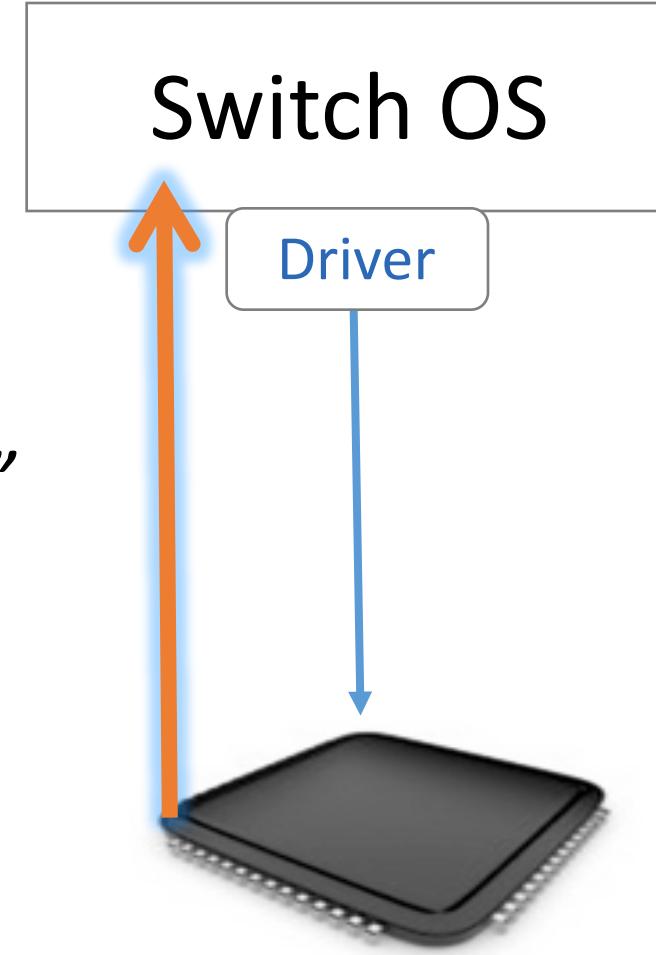
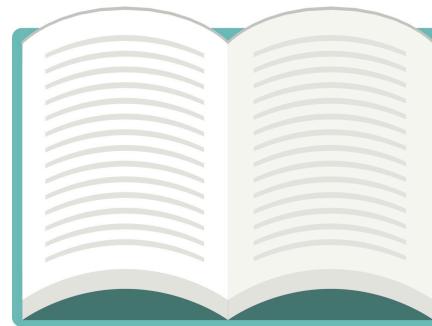


# When you need a new feature...

1. Equipment vendor can't just send you a software upgrade
2. New forwarding features take years to develop
3. By then, you've figured out a kludge to work around it
4. Your network gets more complicated, more brittle
5. Eventually, when the upgrade is available, it either
  - No longer solves your problem, or
  - You need a fork-lift upgrade, at huge expense.

# Network systems are built “bottoms-up”

*“This is how I process packets ...”*



Fixed-function switch

# Network systems are starting to be programmed “top-down”

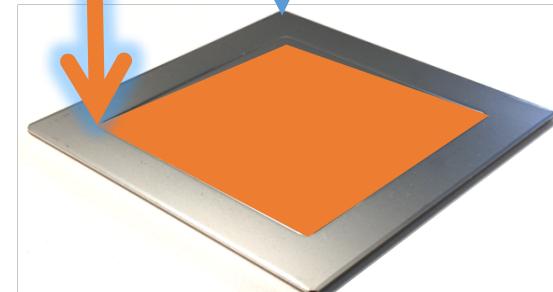
*“This is precisely how you must process packets”*

```
table int_table {
    reads {
        ip.protocol;
    }
    actions {
        export_queue_latency;
    }
}
```

```
action export_queue_latency (sw_id) {
    add_header(int_header);
    modify_field(int_header.kind, TCP_OPTION_INT);
    modify_field(int_header.len, TCP_OPTION_INT_LEN);
    modify_field(int_header.sw_id, sw_id);
    modify_field(int_header.g_latency,
                 intrinsic_metadata.deg_timedelta);
    add_to_field(tcp.dataOffset, 2);
    add_to_field(ipv4.totalLen, 8);
    subtract_from_field(ingress_metadata.tcpLength,
                       12);
}
```

Switch OS

Driver

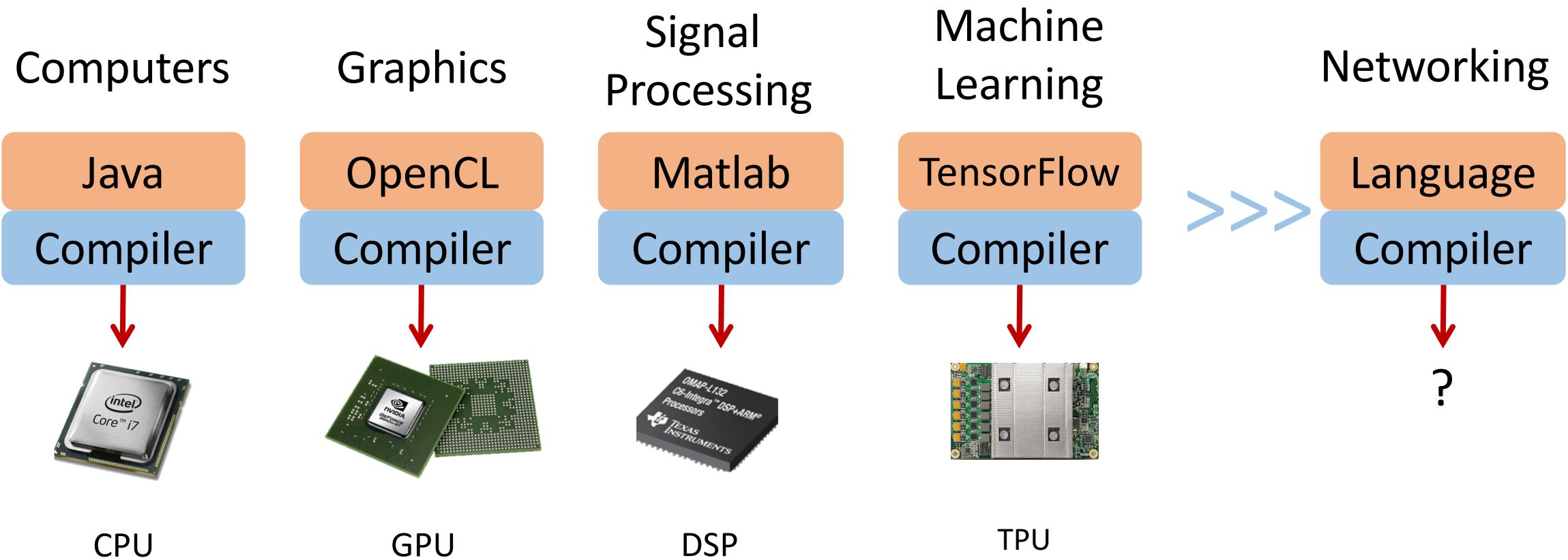


Programmable Switch

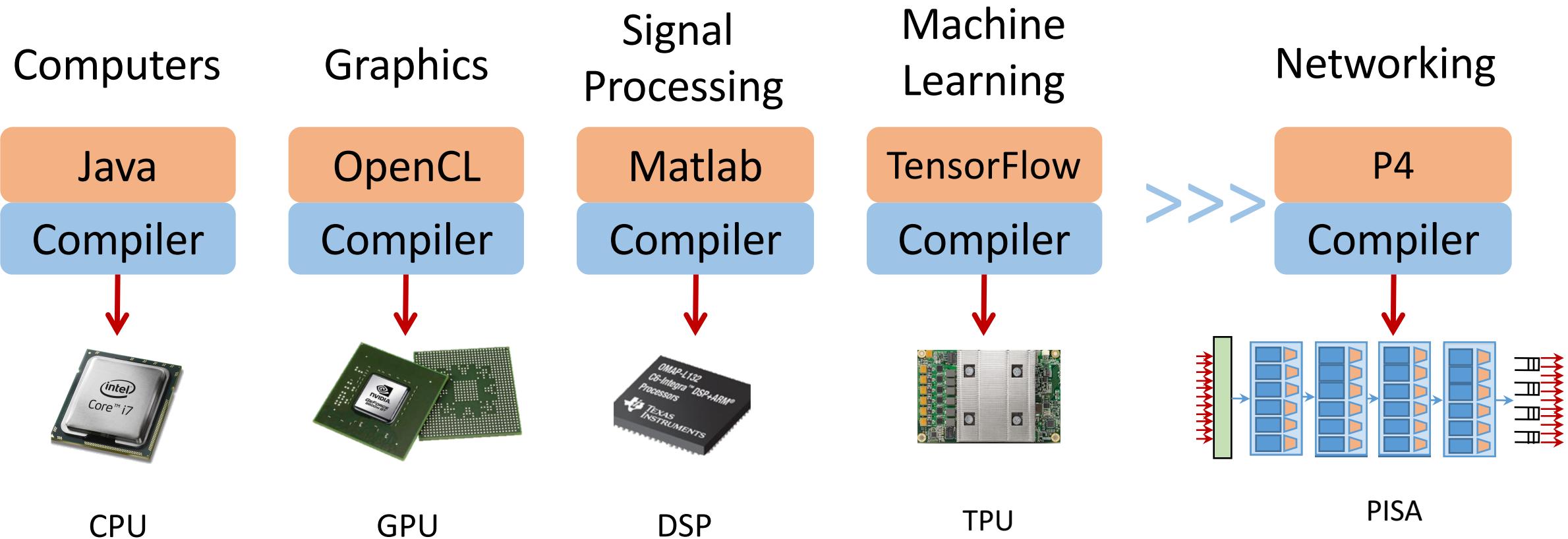
# Outline

1. Why programmability is happening now
2. How programmability is being used
  - Subtract features: Reducing complexity
  - Add proprietary features: Invent, differentiate, own
  - Silicon independence: Breaking a lock-in
  - Telemetry and measurement
3. Why programmability now from an ASIC technology viewpoint

# Domain Specific Processors



# Domain Specific Processors



# Many functionalities in the data plane



# Data plane functionalities have a common style: match-action packet processing

## IP Prefix Forwarding

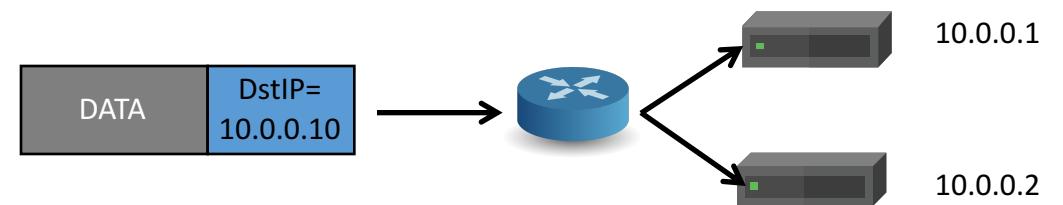
Match	Action
DstIP = 10.0.0.0/24	fwd(1)
DstIP = 10.0.1.0/24	fwd(2)
DstIP = 10.0.2.0/24	fwd(3)
DstIP = 10.0.3.0/24	fwd(4)

## Access Control

Match	Action
DstPort = 80	drop
DstIP = 10.0.0.1	drop
DstIP = 10.0.0.0/24	pass
DstIP = 10.0.0.0/16	drop

## Load Balancing

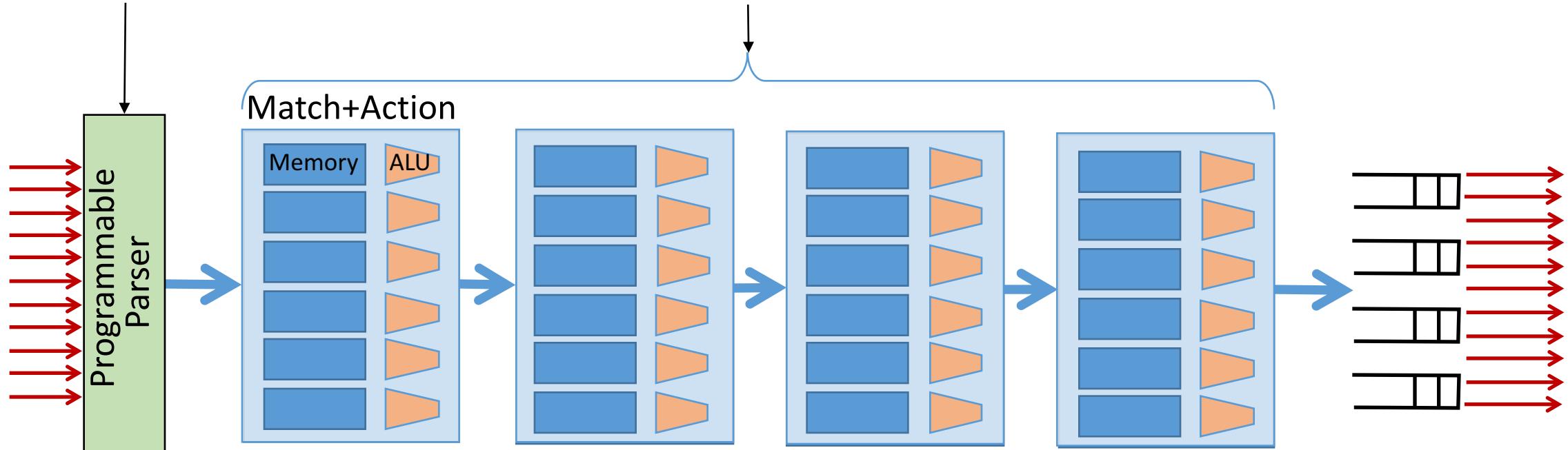
Match	Action
DstIP = 10.0.0.10	DstIP = random(10.0.0.1, 10.0.0.2)
DstIP = 10.0.1.10	DstIP = random(10.0.1.1, 10.0.1.2)



# PISA: Protocol Independent Switch Architecture

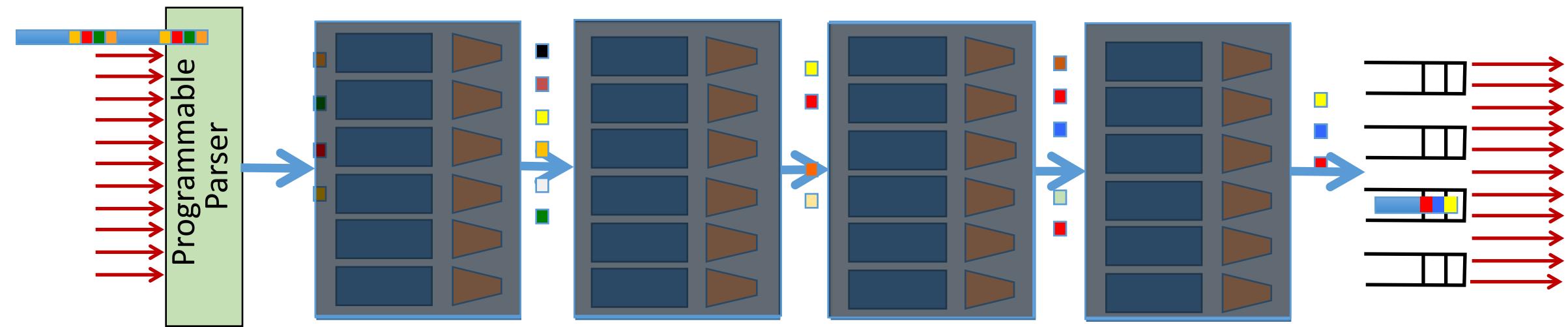
Programmer declares which headers are recognized

Programmer declares what tables are needed and how packets are processed



All stages are identical – makes PISA a good “compiler target”

# PISA: Protocol Independent Switch Architecture



# Tofino 6.5Tb/s Switch

December 2016



65 x 100GE (or 260 x 25GE)

Same power and cost as fixed-function switches.

# Outline

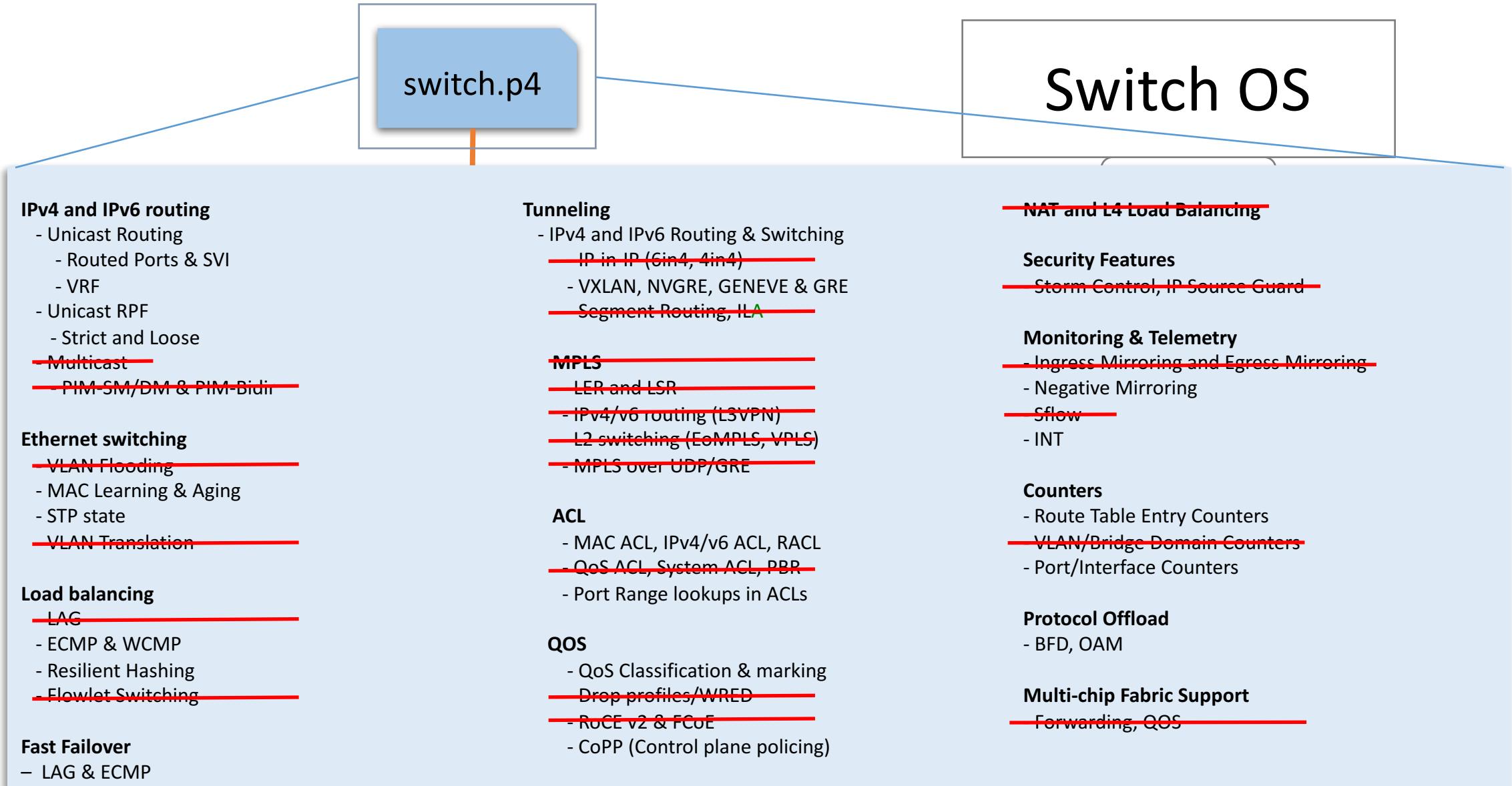
1. Why programmability is happening now
2. How programmability is being used
  - Subtract features: Reducing complexity
  - Add proprietary features: Invent, differentiate, own
  - Silicon independence: Breaking a lock-in
  - Telemetry and measurement
3. Why programmability now from an ASIC technology viewpoint

# How programmability is being used

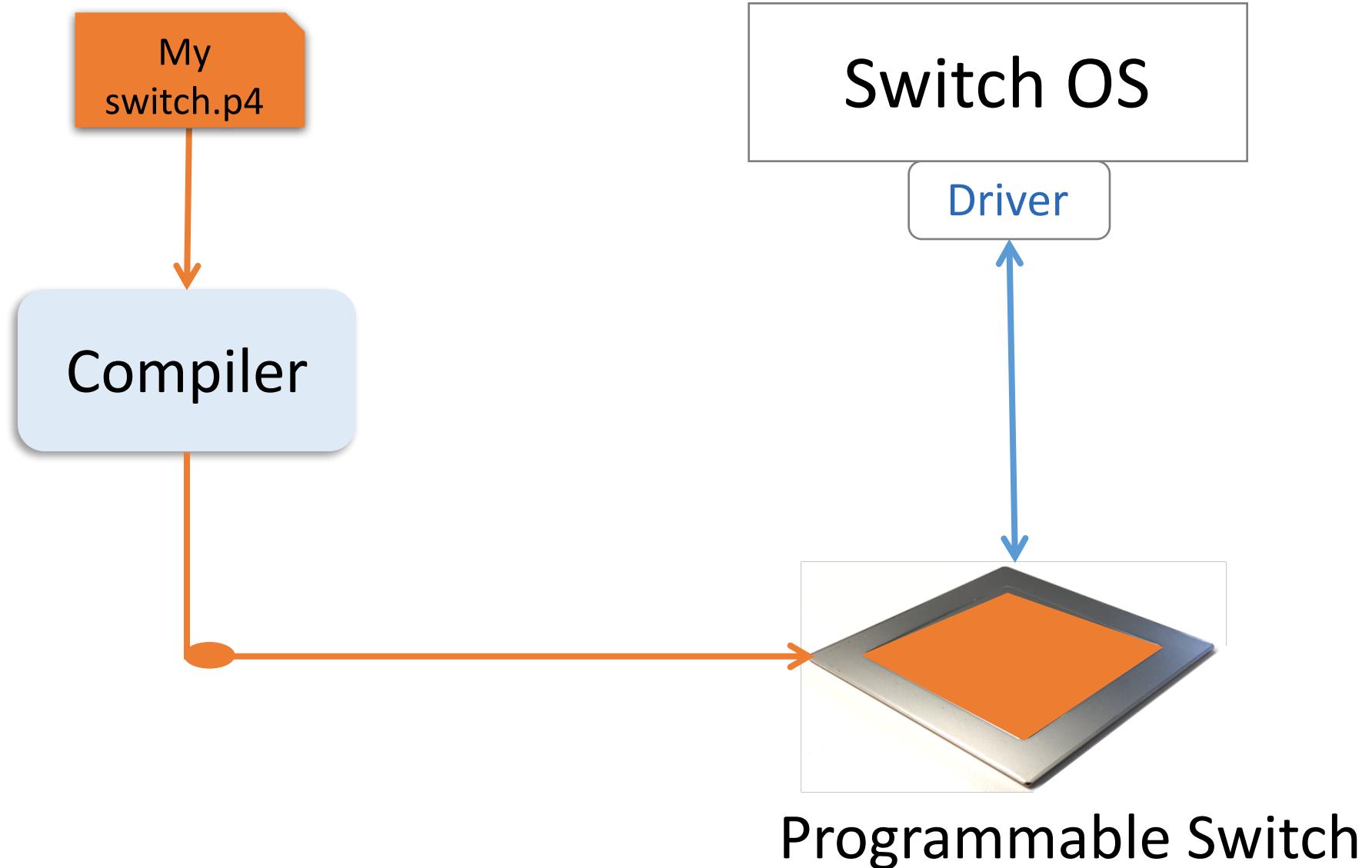
1

## Reducing complexity

# Reducing complexity



# Reducing complexity

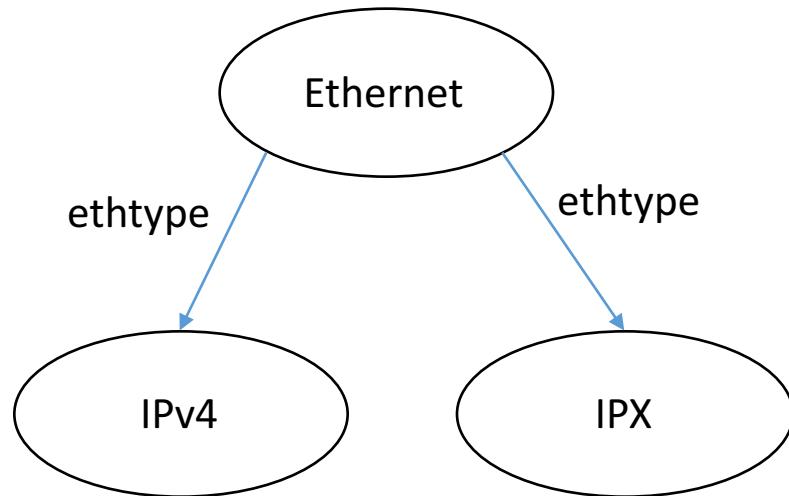


# How programmability is being used

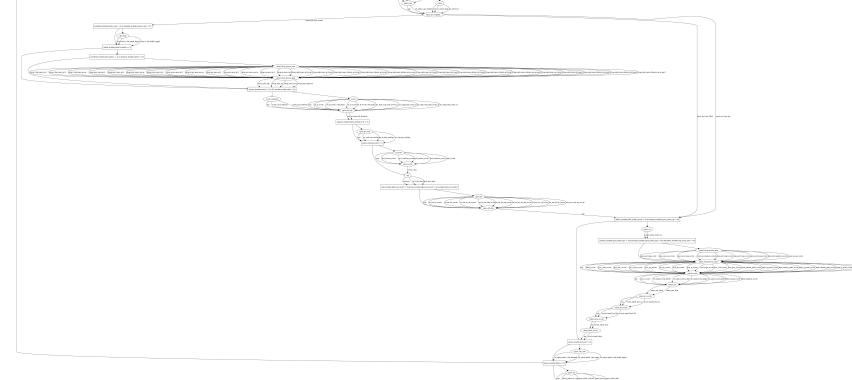
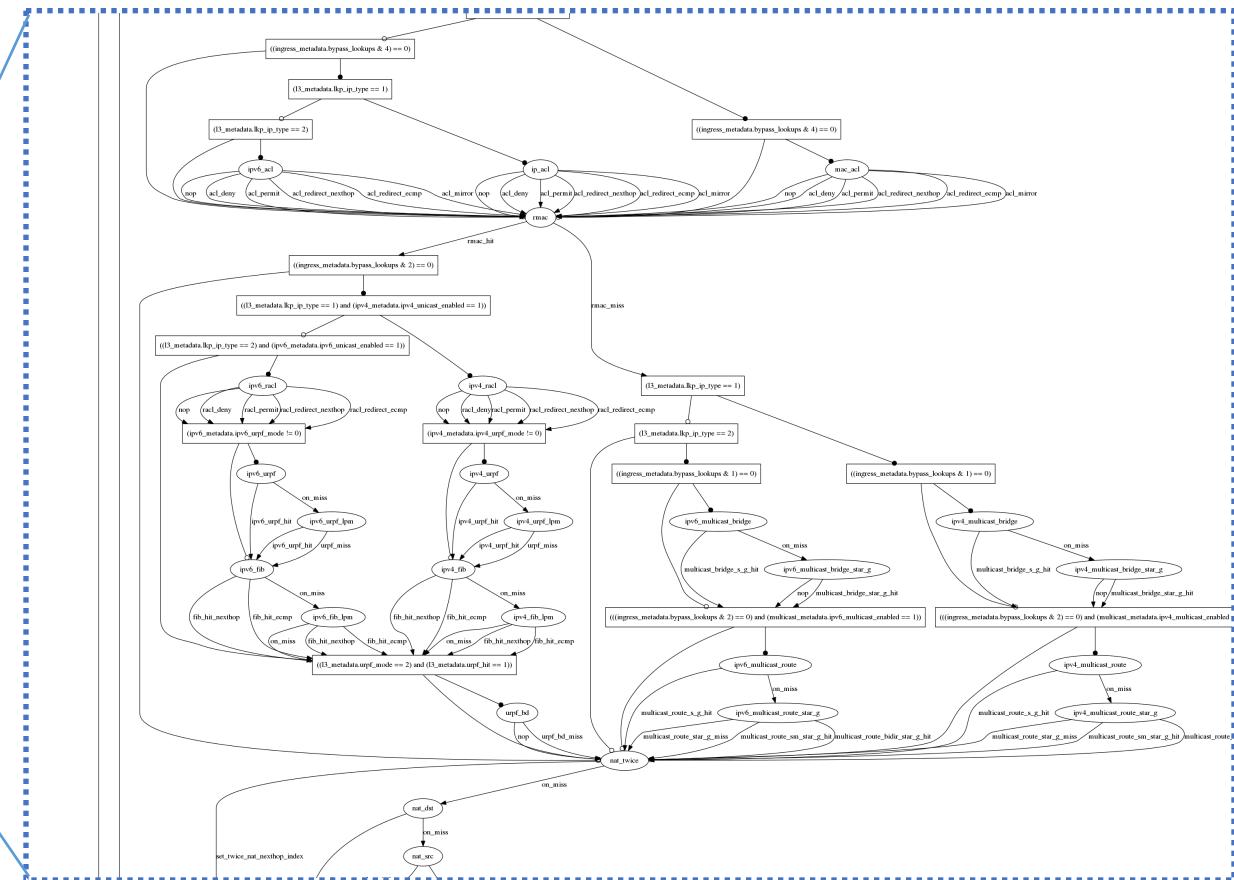
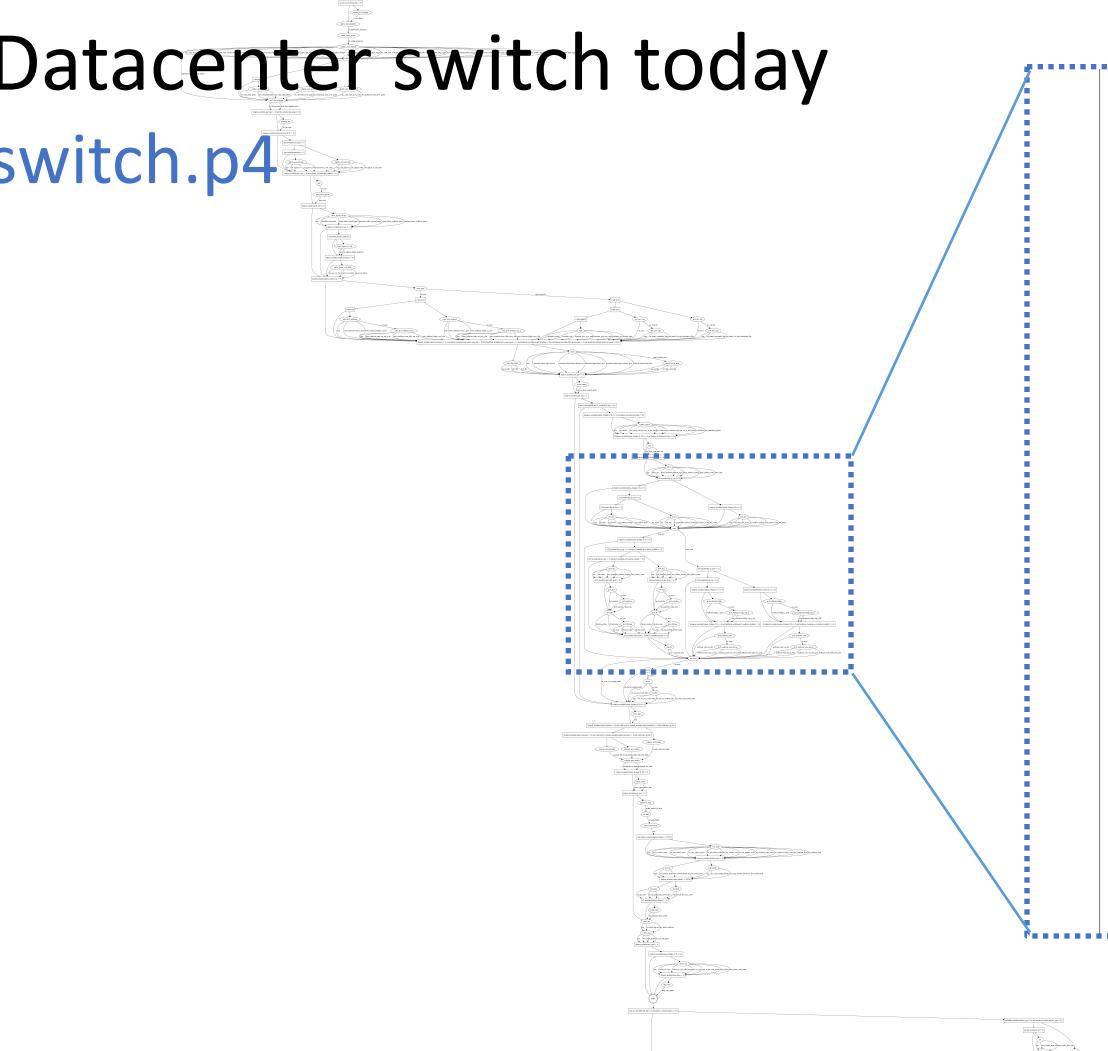
2

## Adding proprietary features

# Protocol complexity 20 years ago



# Datacenter switch today switch.p4



# Adding features: Some examples so far

1. New encapsulations and tunnels
2. New ways to tag packets for special treatment
3. New approaches to routing: e.g. source routing in MSDCs
4. New approaches to congestion control
5. New ways to process packets: e.g. processing ticker-symbols

# New applications: Some examples so far

## 1. Layer-4 Load Balancer<sup>1</sup>

- Replace 100 servers or 10 dedicated boxes with one programmable switch
- Track and maintain mapping for 5-10 million http flows

## 2. Fast stateless firewall

- Add/delete and track 100s of thousands of new connections per second

## 3. Cache for Key-value store<sup>2</sup>

- Memcache in-network cache for 100 servers
- 1-2 billion operations per second

[1] “SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs.” Rui Miao et al. Sigcomm 2017.

[2] “NetCache: Balancing Key-Value Stores with Fast In-Network Caching”, Xin Jin et al. To appear at SOSP 2017

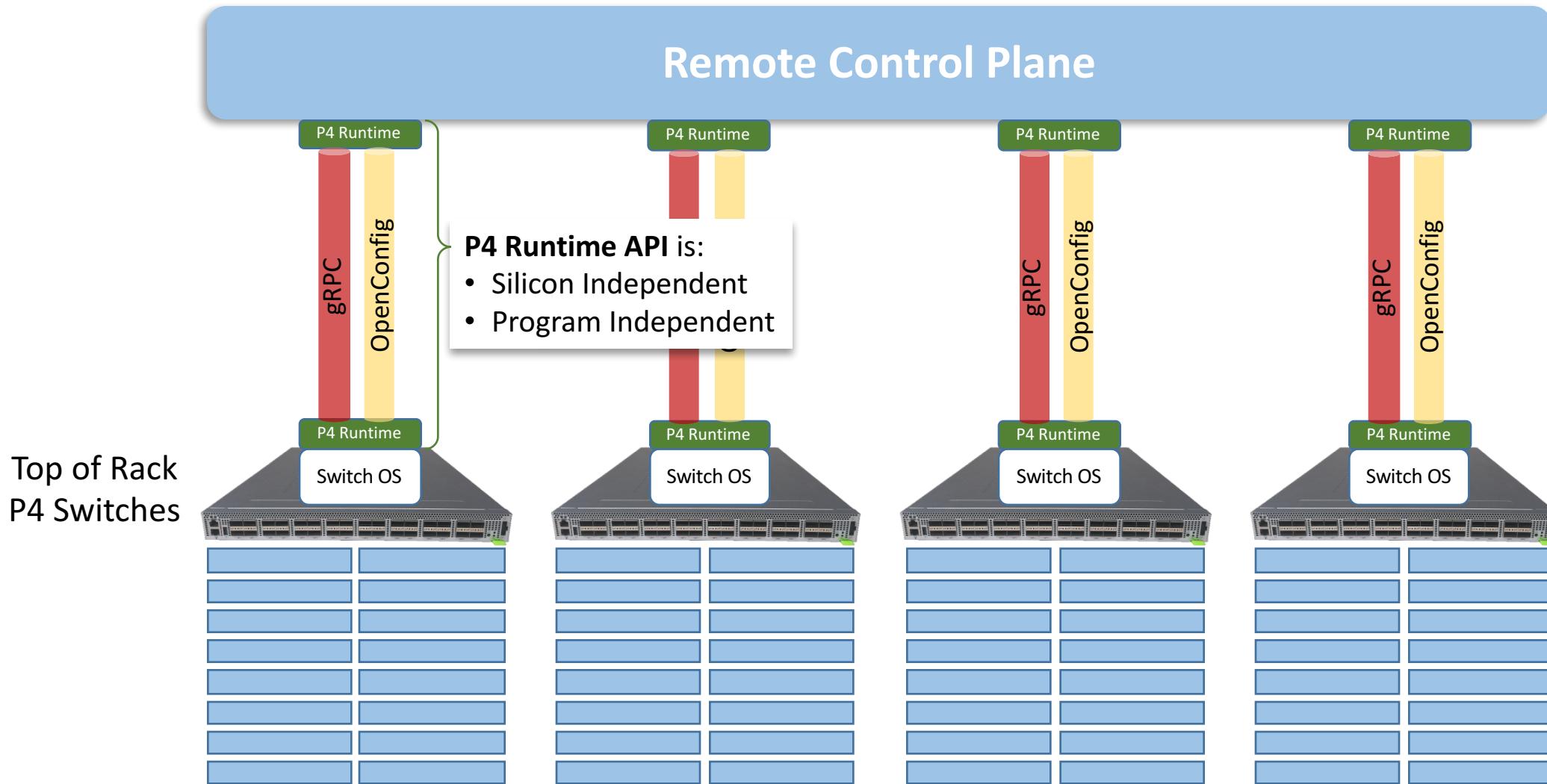
# How programmability is being used

③

## Silicon independence

# P4 Runtime

Open-source project to remotely control P4 switches<sup>1</sup>

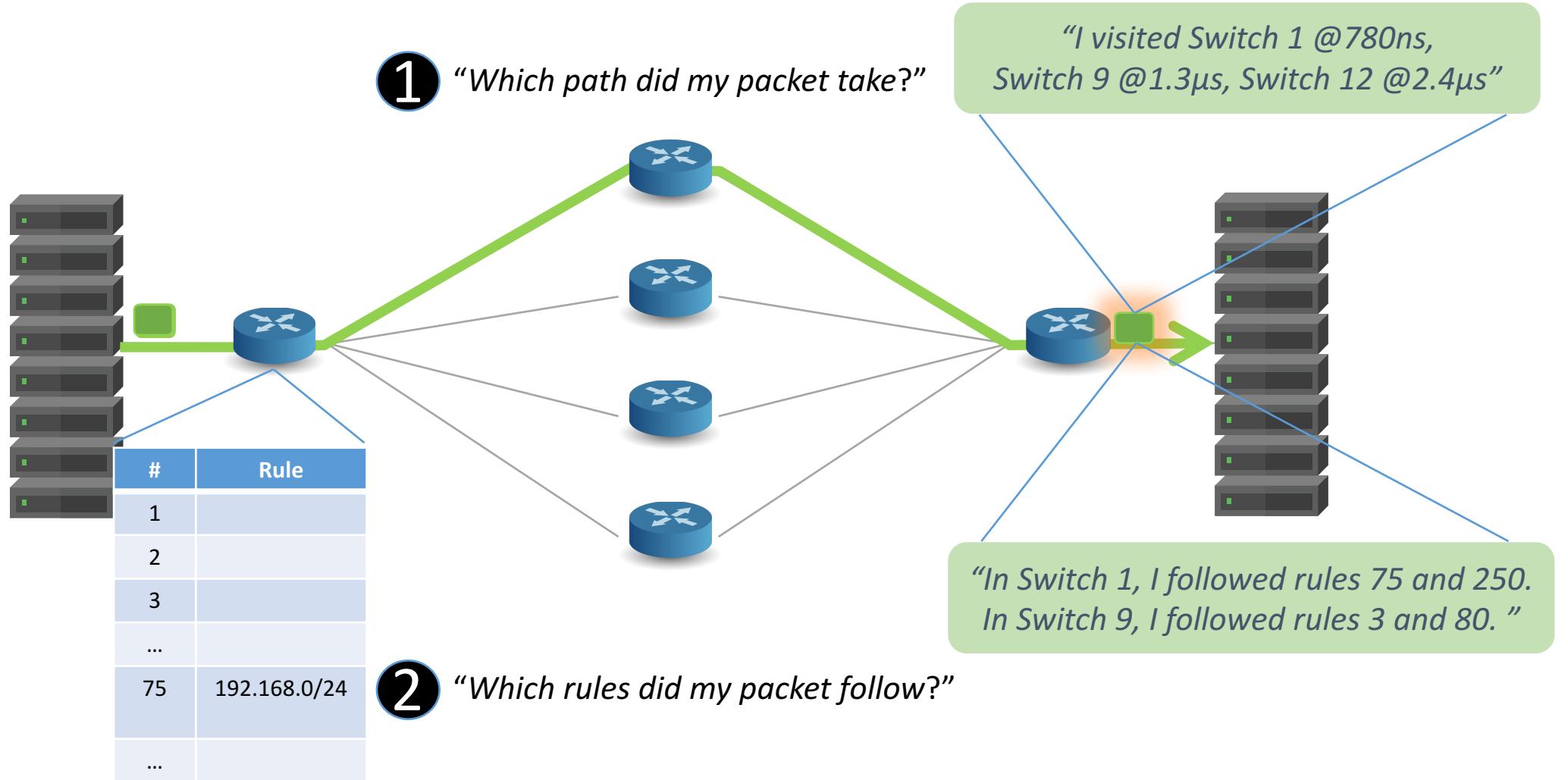


[1] "P4 Program-dependent Controller Interface for SDN Applications", Samar Abdi et al (Google), P4 Workshop 2017

# How programmability is being used

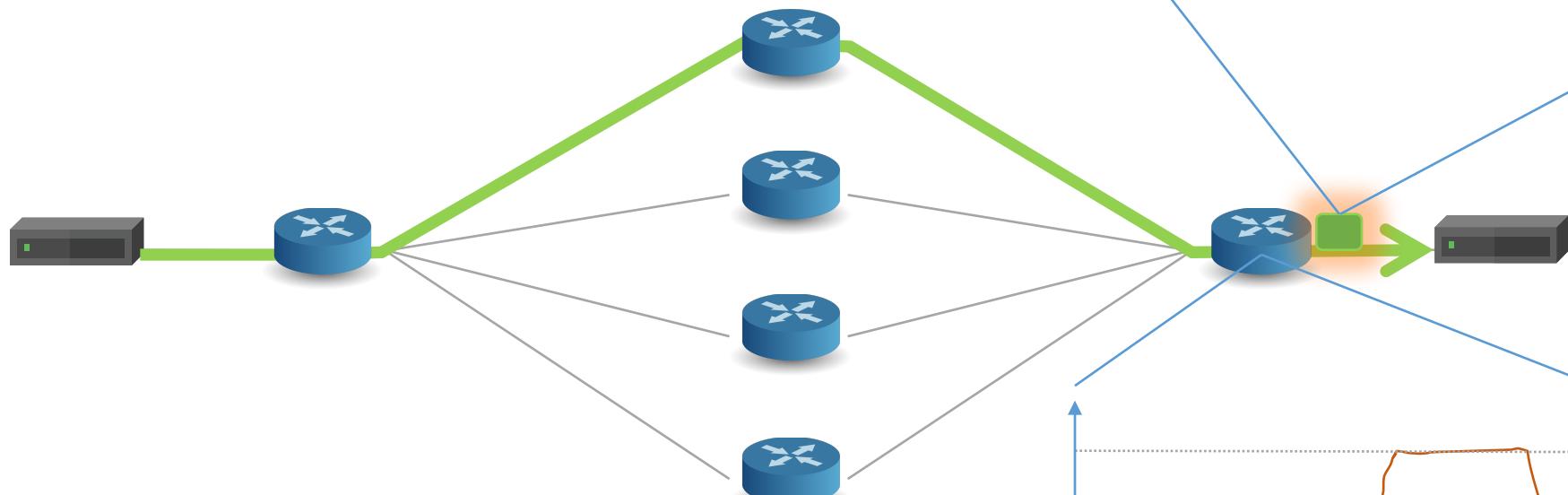
4

## Network telemetry



3

*"How long did my packet queue at each switch?"*



*"Delay: 100ns, 200ns, 19740ns"*

Queue

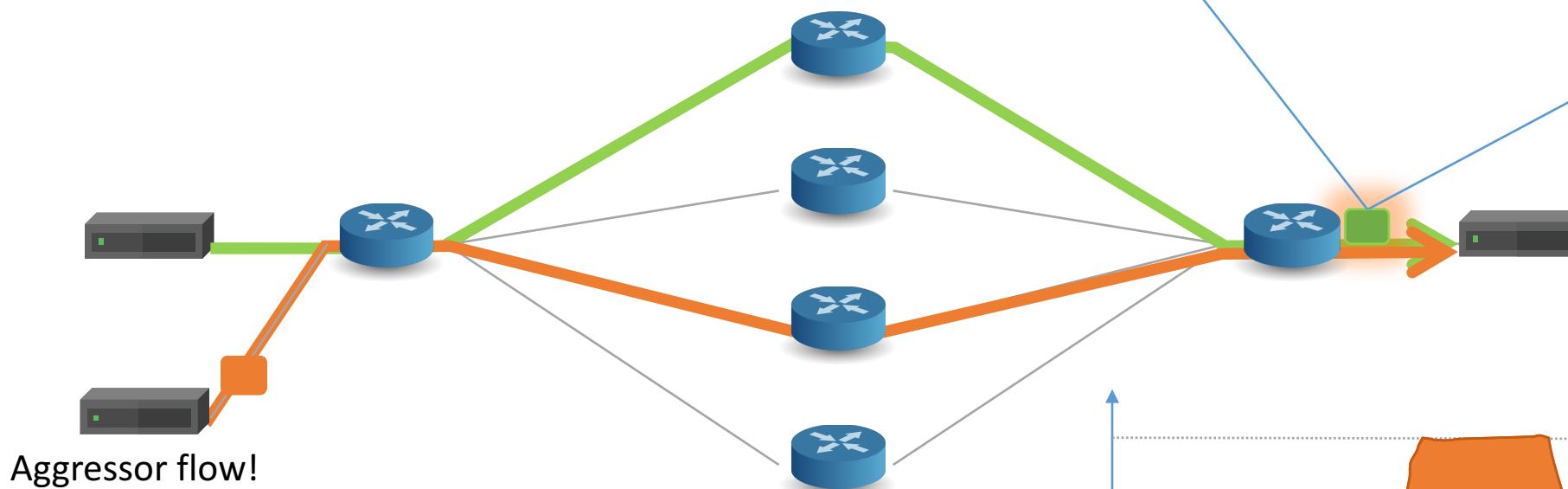
Time

4

*"Who did my packet share the queue with?"*

3

*"How long did my packet queue at each switch?"*



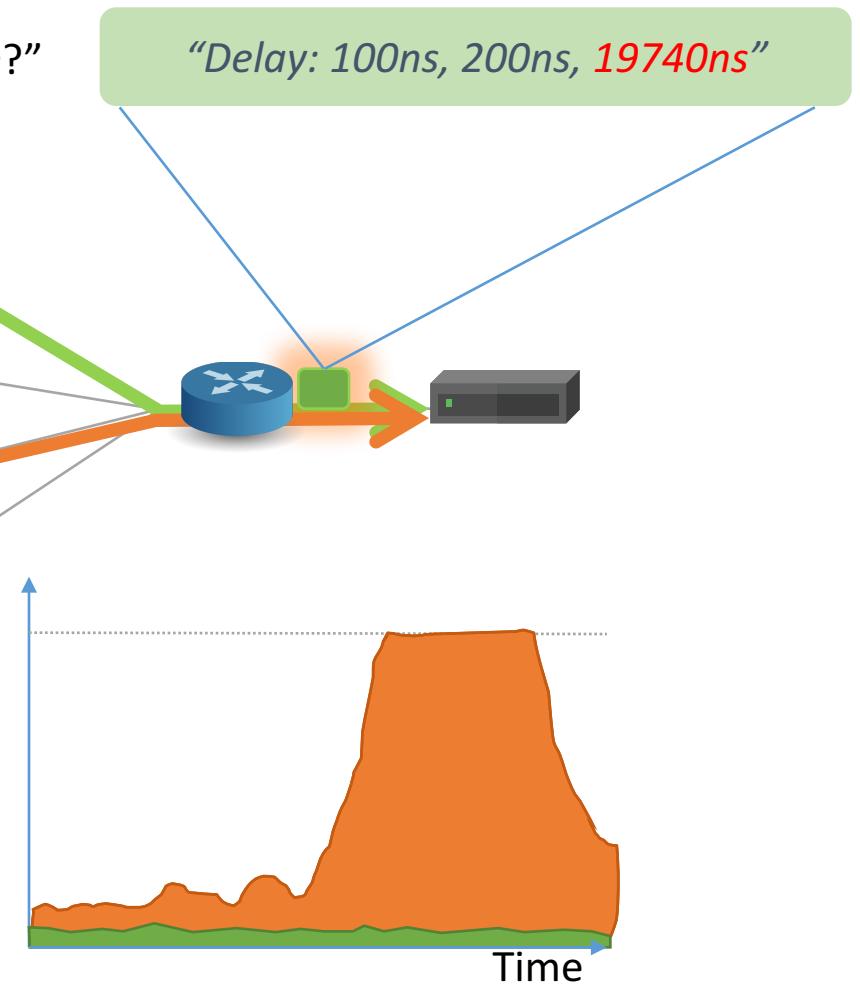
*"Delay: 100ns, 200ns, 19740ns"*

Queue

Time

4

*"Who did my packet share the queue with?"*

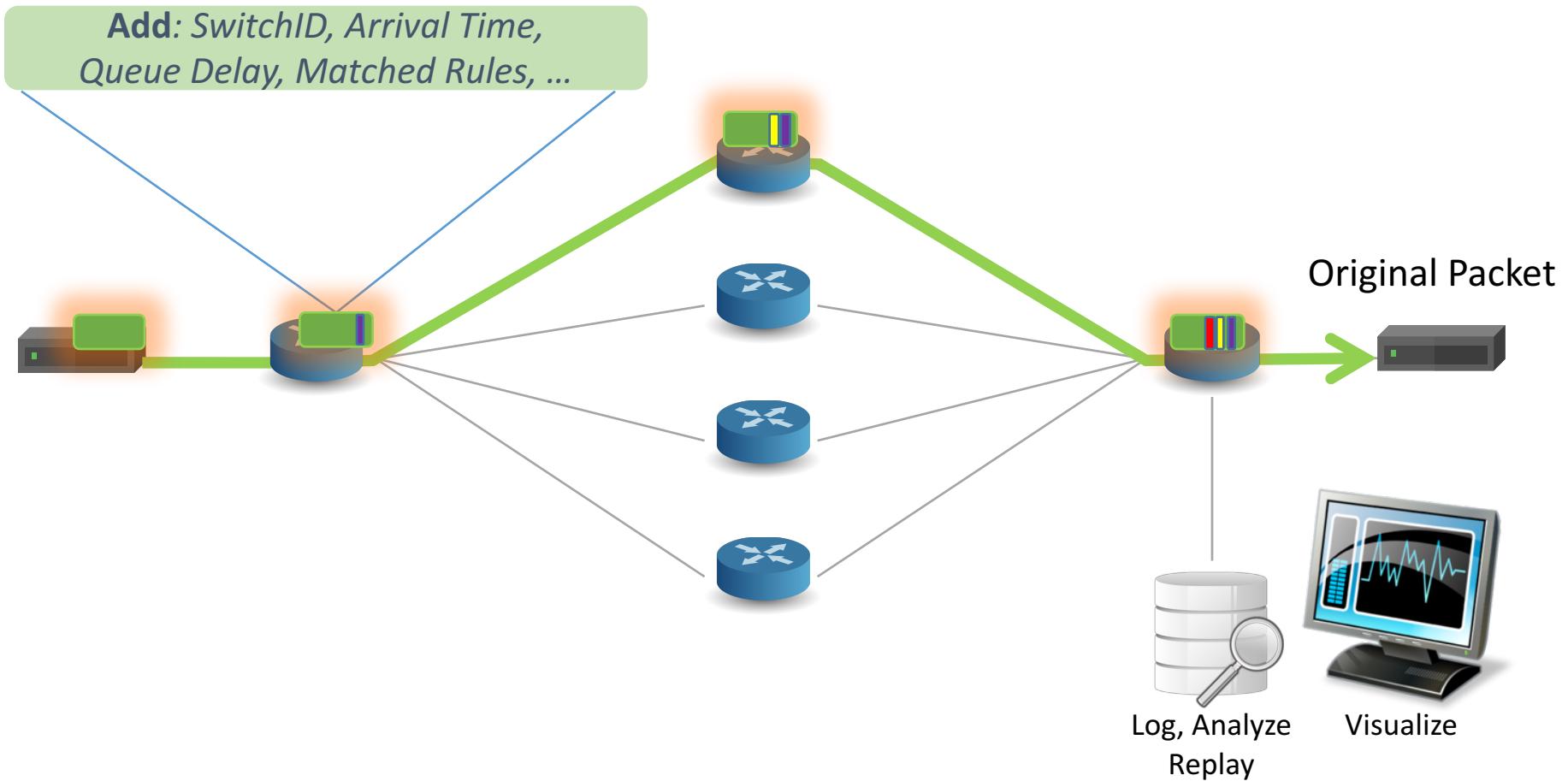


# We'd like the network to answer these questions

- 1 “Which path did my packet take?”
- 2 “Which rules did my packet follow?”
- 3 “How long did it queue at each switch?”
- 4 “Who did it share the queues with?”

A programmable device can potentially answer all four questions at line rate.

# INT: Inband Network Telemetry

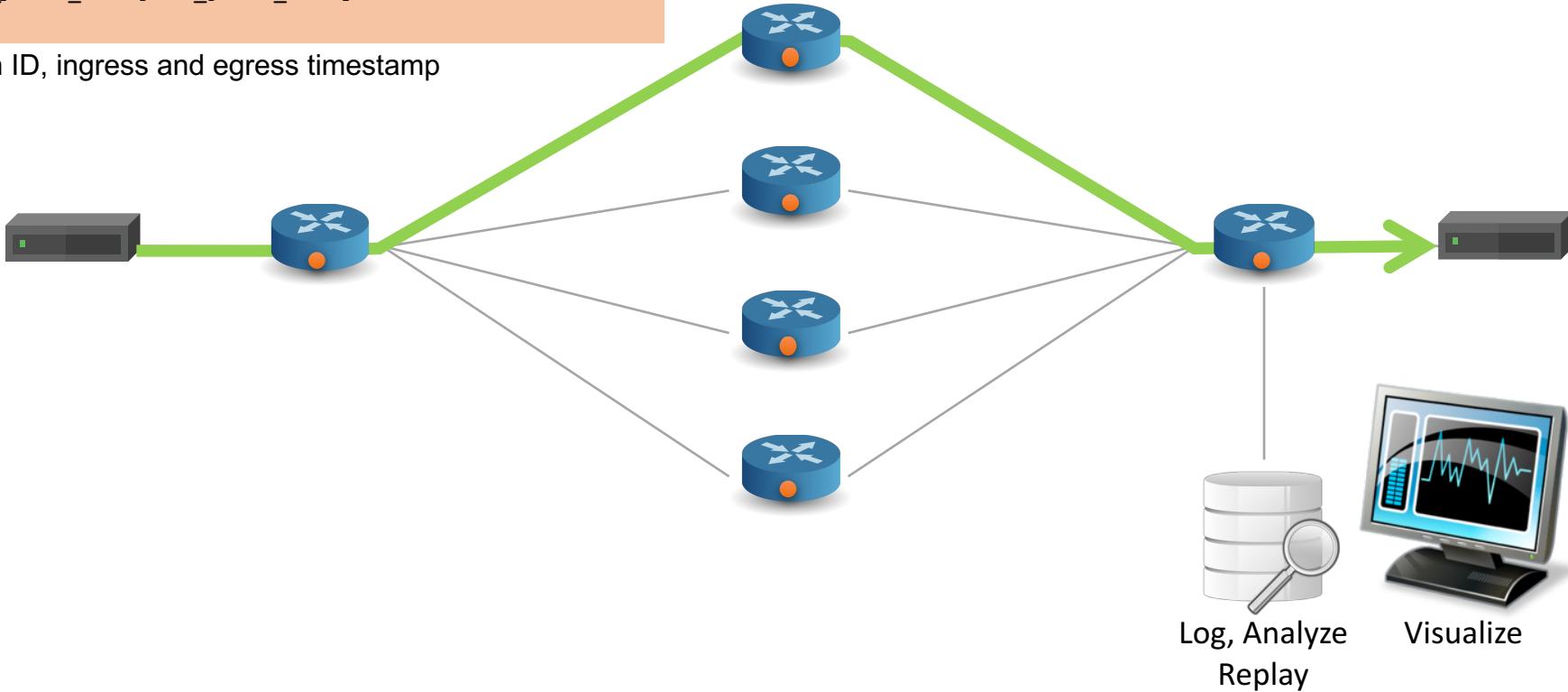


```
/* INT: add switch id */
action int_set_header_0() {
    add_header(int_switch_id_header);
    modify_field(int_switch_id_header.switch_id,
                 global_config_metadata.switch_id);
}

/* INT: add ingress timestamp */
action int_set_header_1() {
    add_header(int_ingress_tstamp_header);
    modify_field(int_ingress_tstamp_header.ingress_tstamp, i2e_metadata.ingress_tstamp);
}

/* INT: add egress timestamp */
action int_set_header_2() {
    add_header(int_egress_tstamp_header);
    modify_field(int_egress_tstamp_header.egress_tstamp,
                 eg_intr_md_from_parser_aux.egress_global_tstamp);
}
```

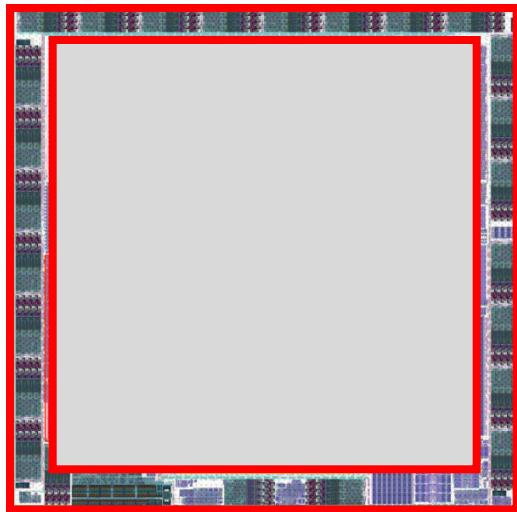
P4 code snippet: Insert switch ID, ingress and egress timestamp



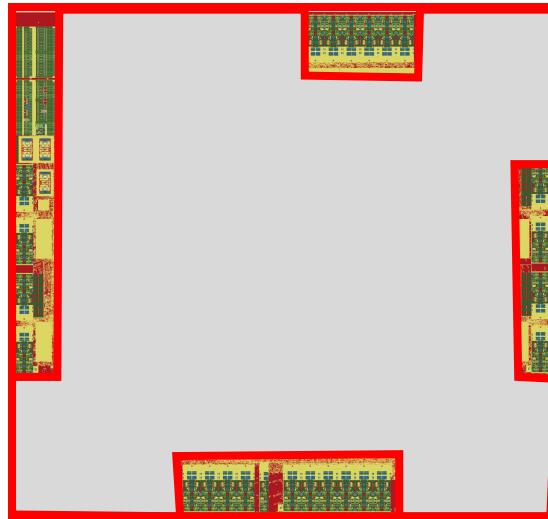
# Outline

1. Why programmability is happening now
2. How programmability is being used
  - Subtract features: Reducing complexity
  - Add proprietary features: Invent, differentiate, own
  - Silicon independence: Breaking a lock-in
  - Telemetry and measurement
3. Why programmability now from an ASIC technology viewpoint

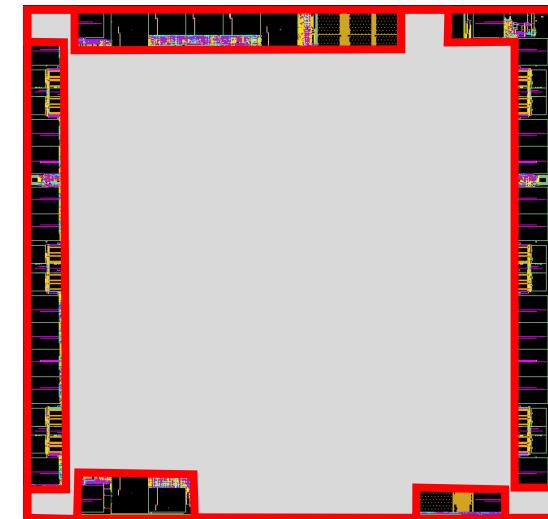
# Serial I/O: About 30% of switch chip area



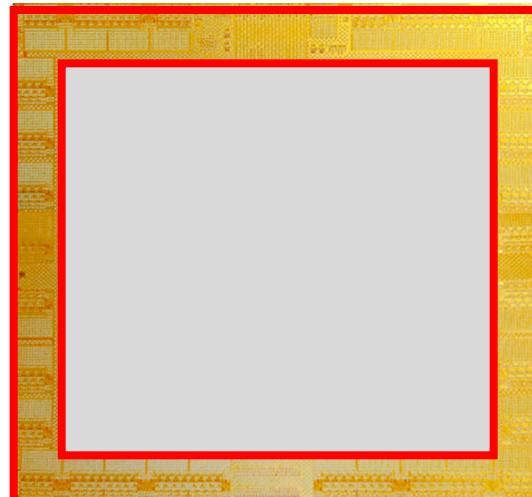
Intel Alta (2011)



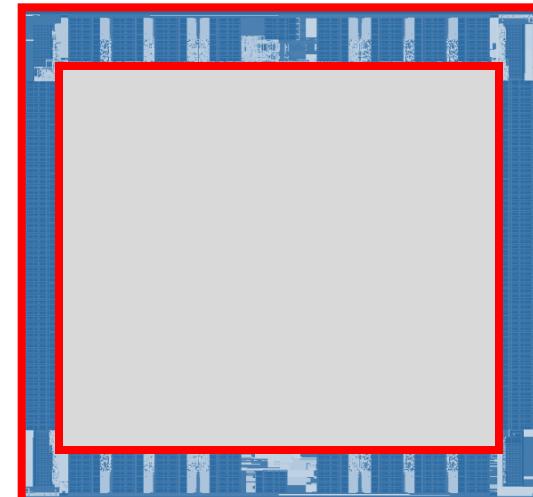
Cisco (2011)



Ericsson Spider (2011)

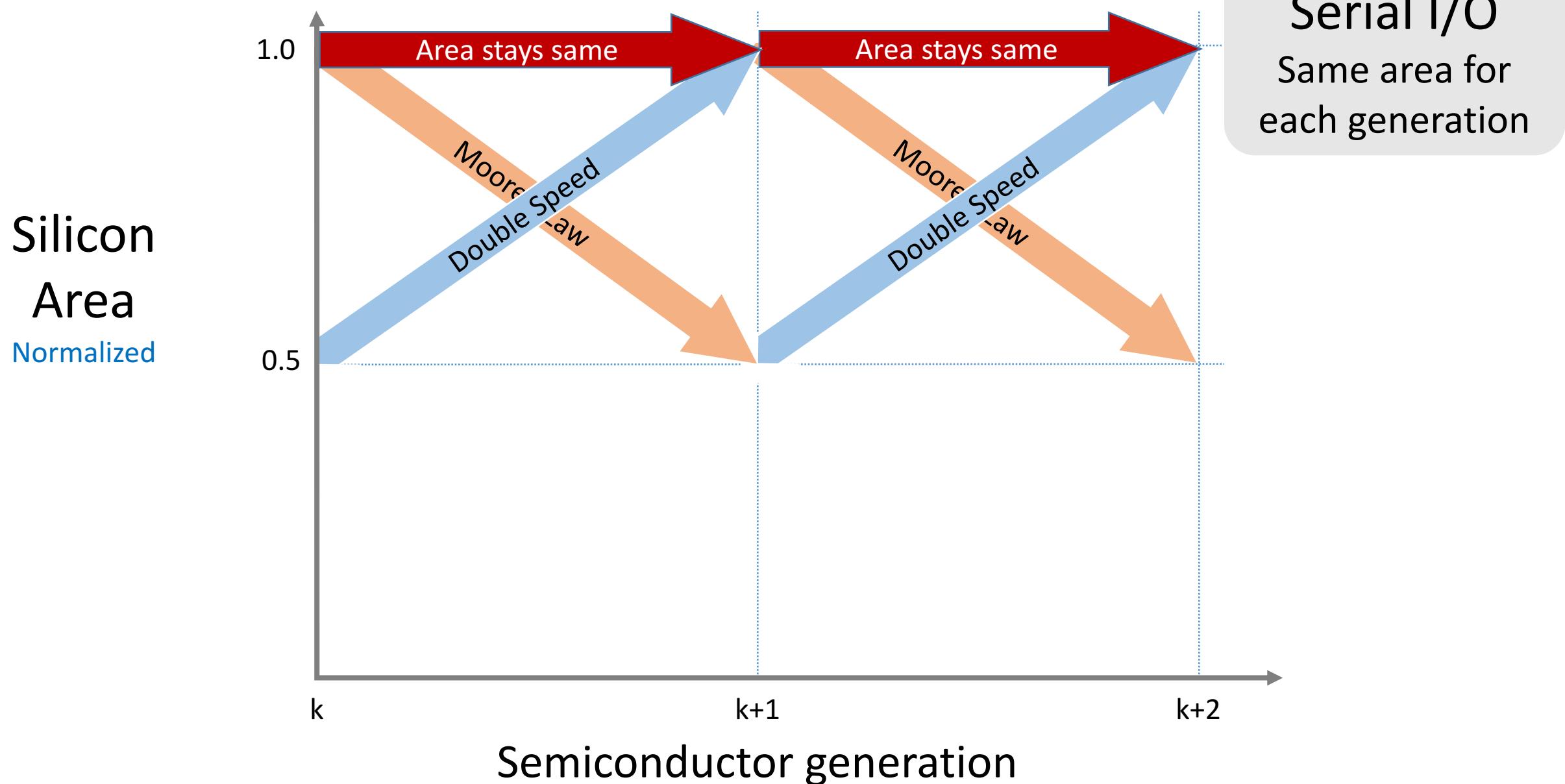


Broadcom Tomahawk (2014)



Barefoot Tofino (2016)

# Switch chip area: Serial I/O



30% Serial I/O

50%  
Packet Processing

20%  
Packet Buffer  
& TM

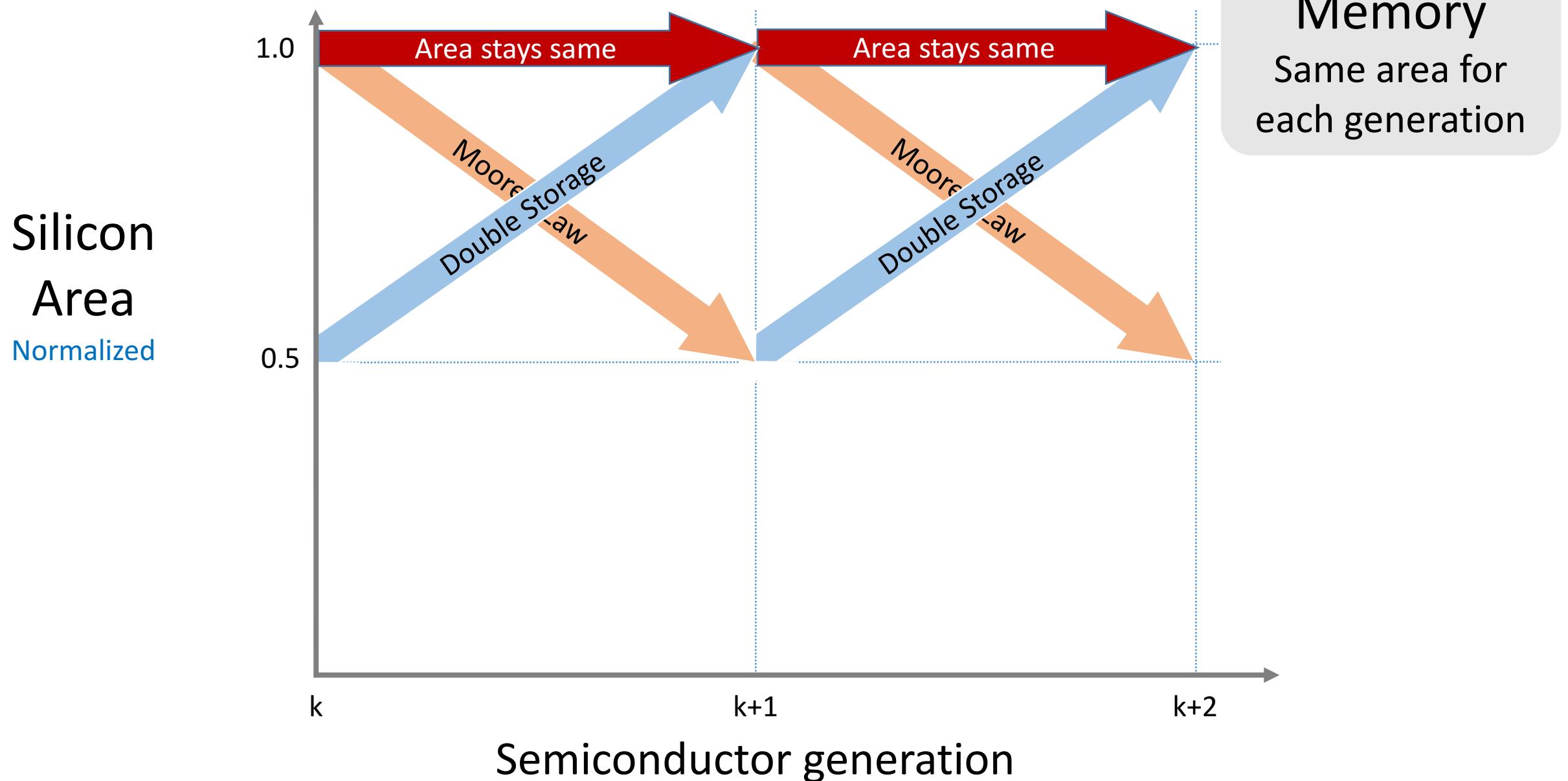
30% Serial I/O

25% Memory  
(Lookup tables)

25% Logic  
(Packet Processing)

20%  
Packet Buffer  
& TM

# Switch chip area: Memory (Tables and Buffer)



30% Serial I/O

25% Memory  
(Lookup tables)

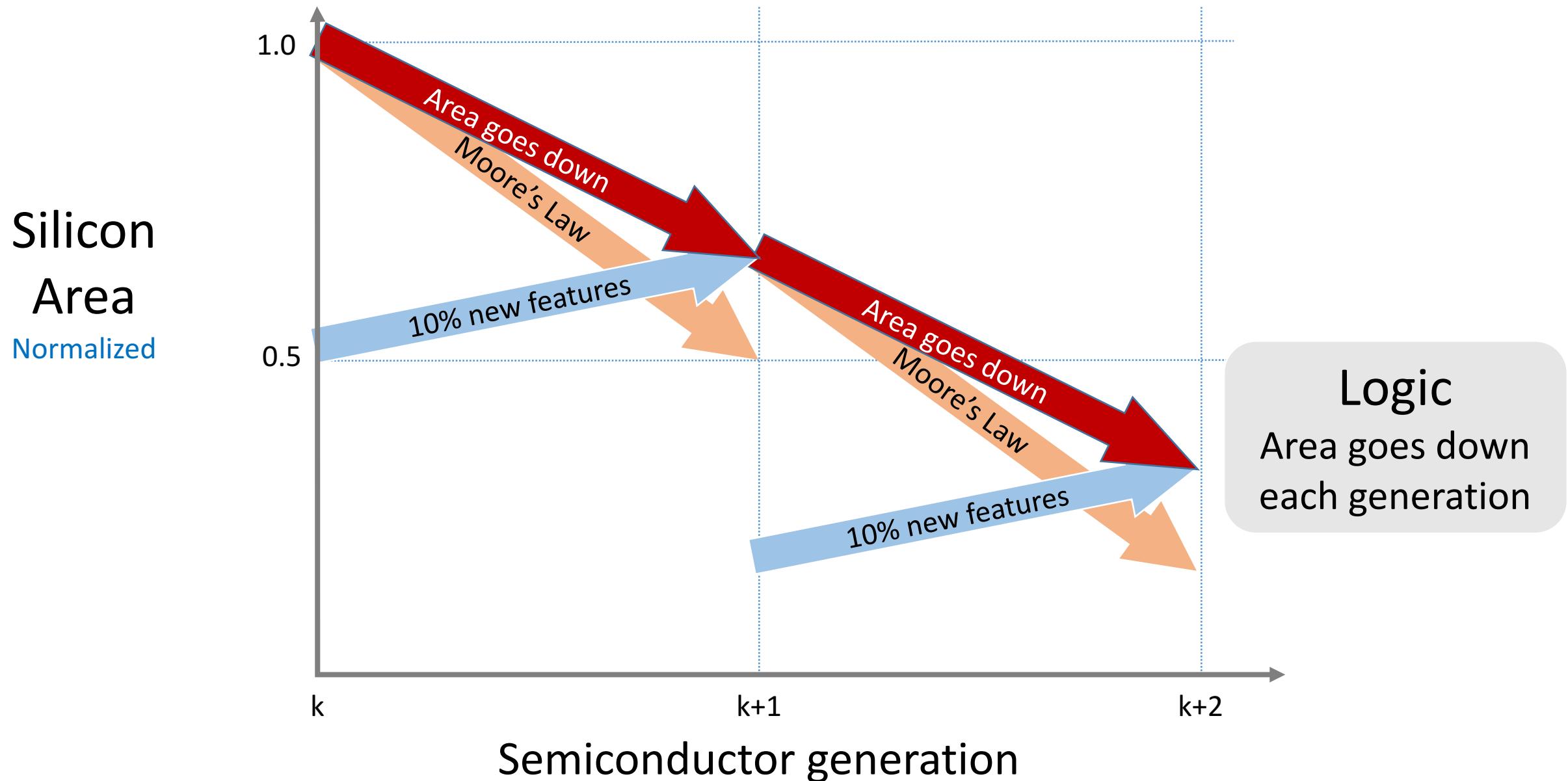
20%  
Packet Buffer  
& TM

25% Logic  
(Packet Processing)

Logic dictates whether  
“fixed function” or  
“programmable”



# Switch chip area: Packet Processing Logic



30% Serial I/O

25% Memory  
(Lookup tables)

20%  
Packet Buffer  
& TM

25% Logic  
(Packet Processing)

Logic dictates whether  
“fixed function” or  
“programmable”



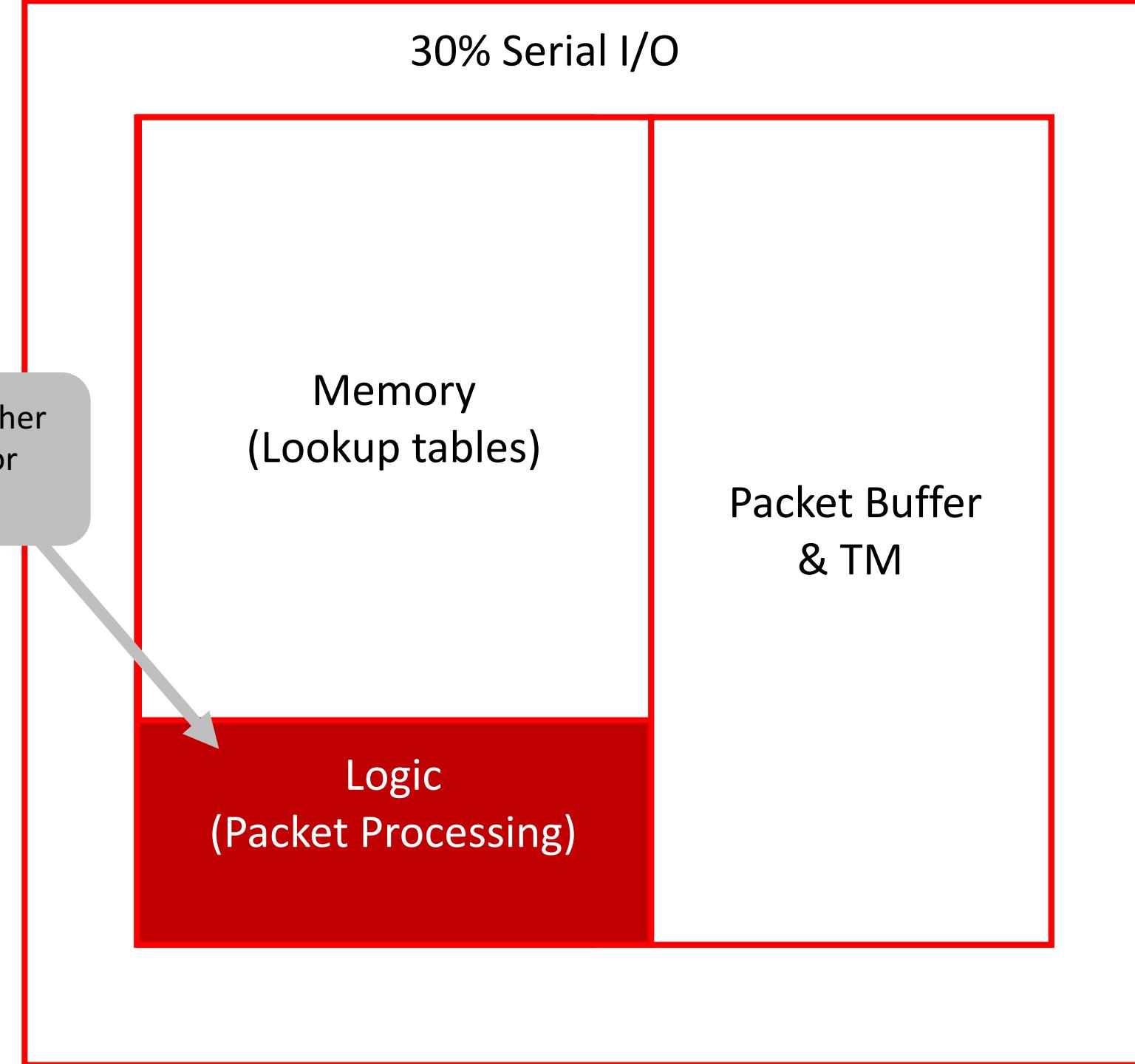
## 30% Serial I/O

Logic dictates whether  
“fixed function” or  
“programmable”

Memory  
(Lookup tables)

Packet Buffer  
& TM

Logic  
(Packet Processing)



30% Serial I/O

Logic dictates whether  
“fixed function” or  
“programmable”

Memory  
(Lookup tables)

Packet Buffer  
& TM

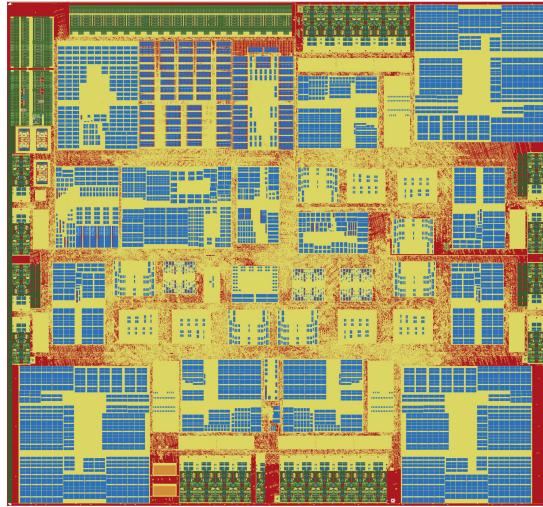
Logic  
(Packet Processing)

Memory  
(Lookup tables)

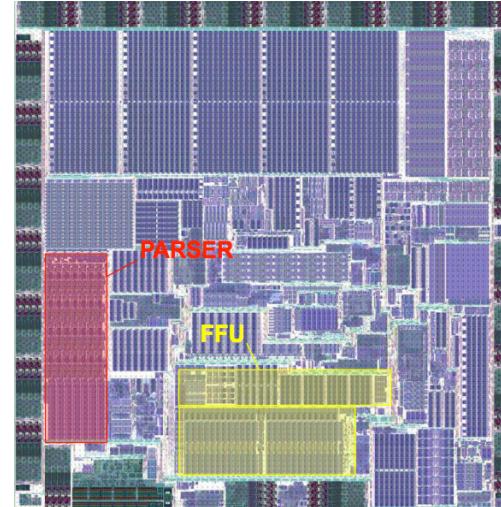
Packet Buffer  
& TM

Logic  
(Packet Processing)

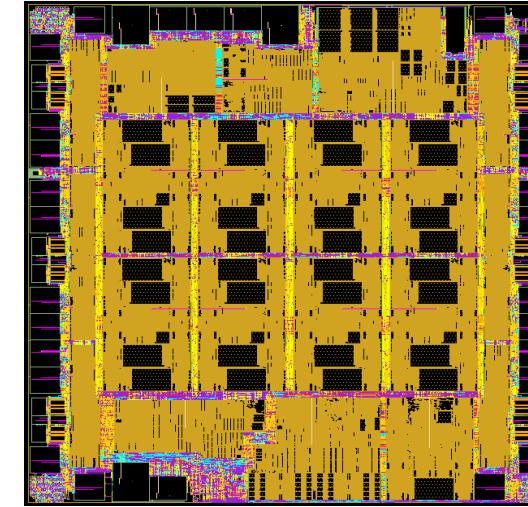
# Programmable switch chips are more uniform



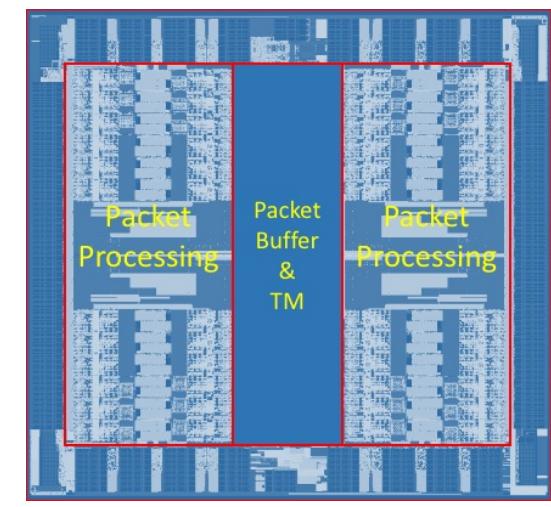
Cisco



Intel Alta



Ericsson Spider



Barefoot Tofino

More programmable



# In summary

1. **Chip speed:** We can now make programmable switch chips as fast as fixed ones.
2. **Chip technology:** The difference in chip area and power between “programmable” and “fixed function” is going away.
3. **Chip complexity:** There are now too many protocols to correctly hard-code in silicon.
4. **New ideas:** Beautiful new ideas are owned by the programmer, not the chip designer.
5. **Level playing field:** Lets us create a solid platform, an abstraction layer, upon which more will be built

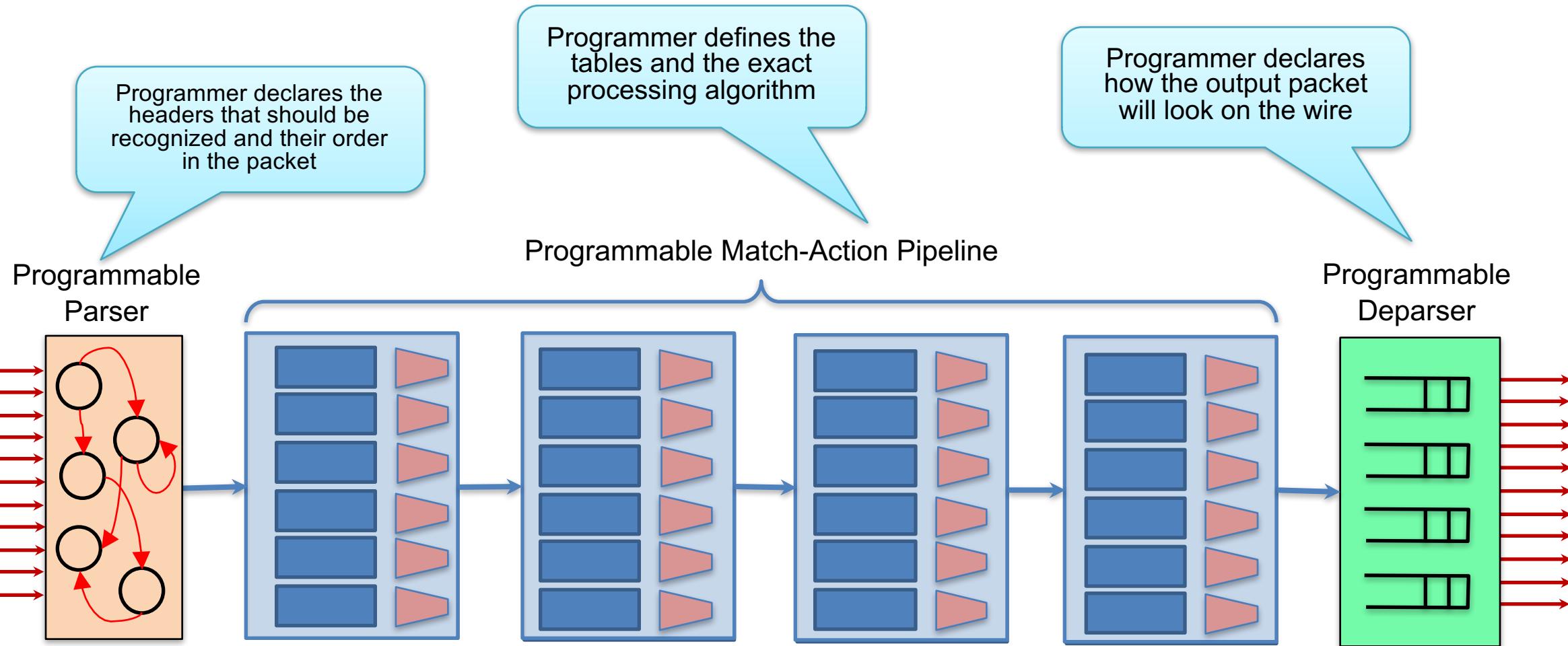
# Agenda

- **Programmable Networks**
- **Tutorial on Assignment 4**

# Tutorial on Assignment 4

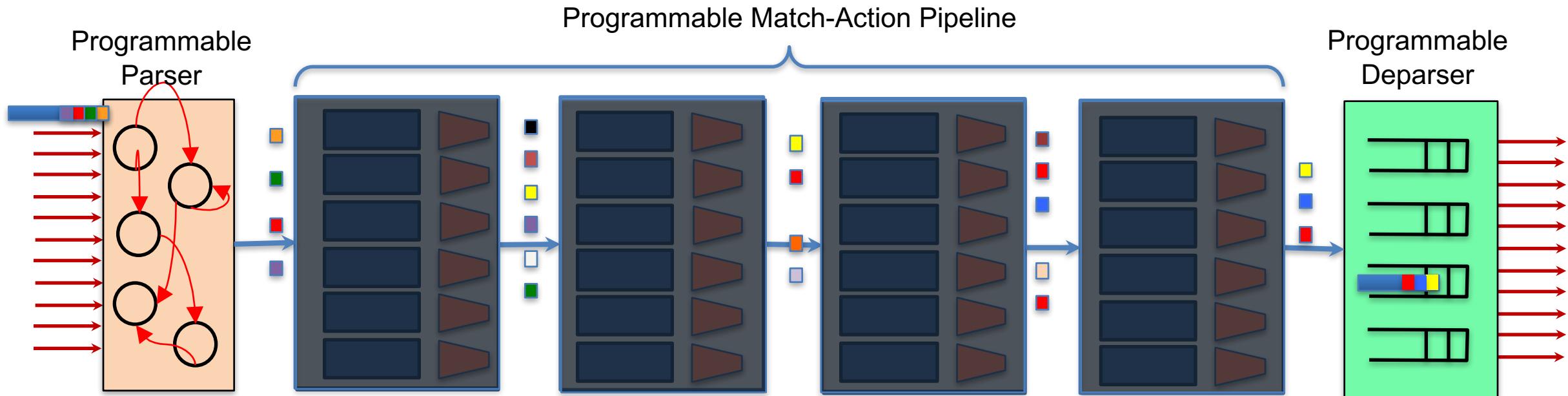
- Exercise 1: Access control list (ACL)
  - Write P4 code to implement access control list (acl.p4)
  - Add rules to your table to block certain destination port and IP (s1-acl.json)
- Exercise 2: Load balancing
  - Write P4 code to implement load balancing (lb.p4)
  - Add rules to your table to block certain destination port and IP (s1-lb.json)

# PISA: Protocol-Independent Switch Architecture

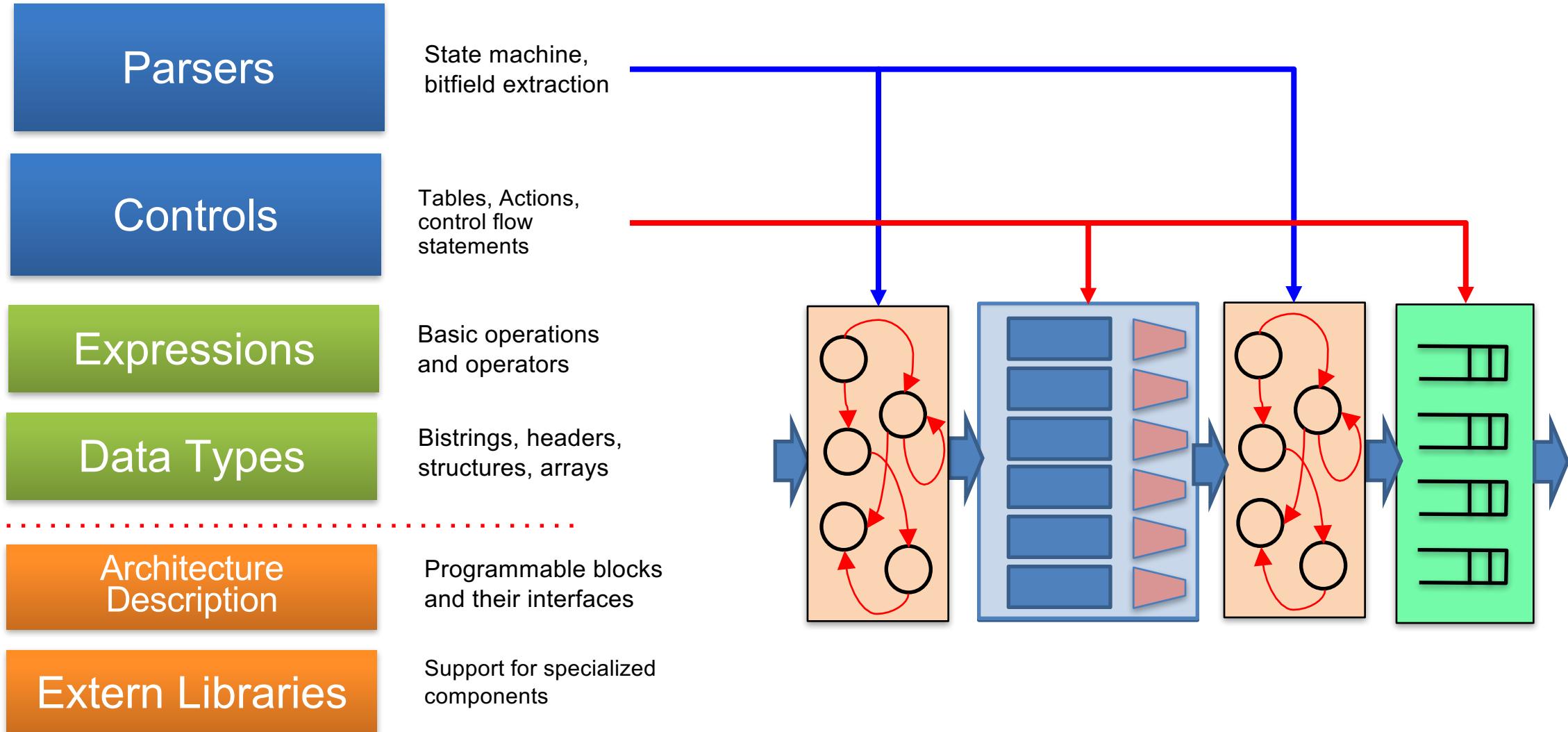


# PISA in Action

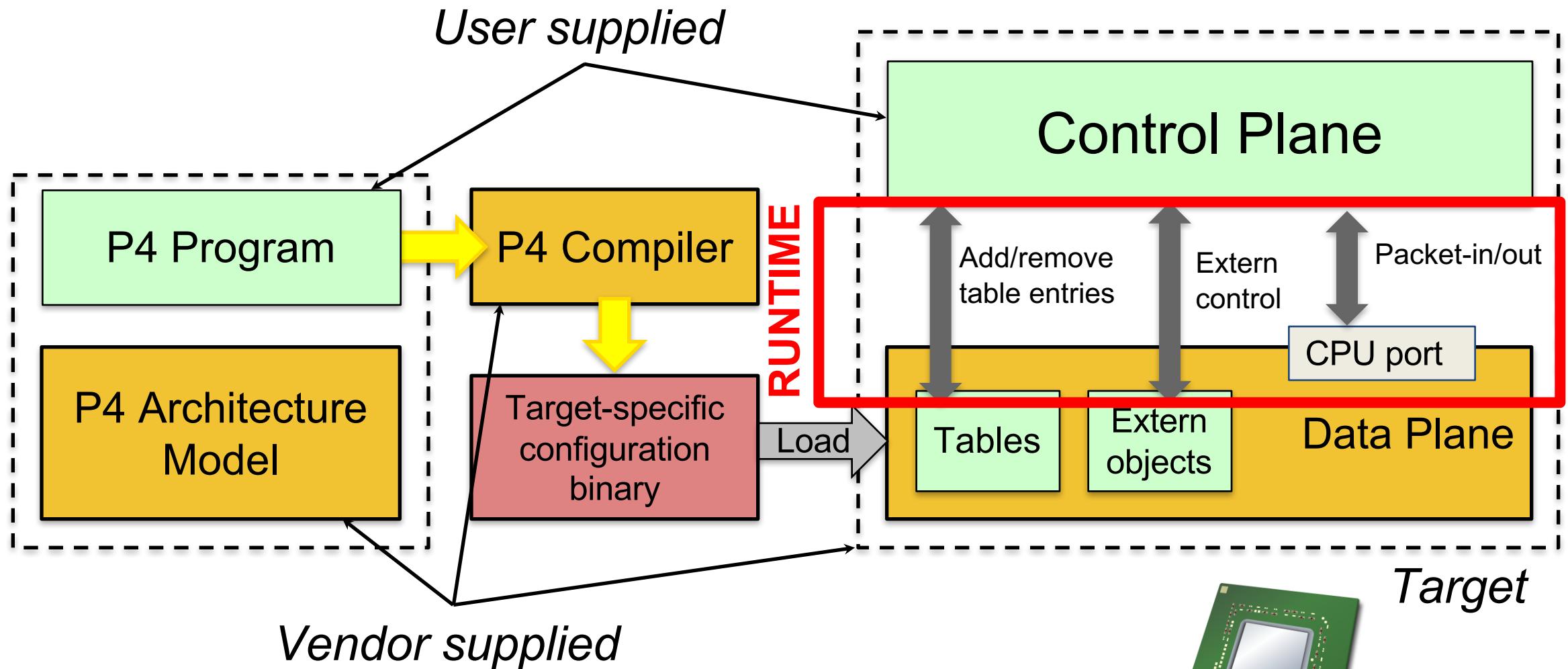
- Packet is parsed into individual headers (parsed representation)
- Headers and intermediate results can be used for matching and actions
- Headers can be modified, added or removed
- Packet is deparsed (serialized)



# P4<sub>16</sub> Language Elements

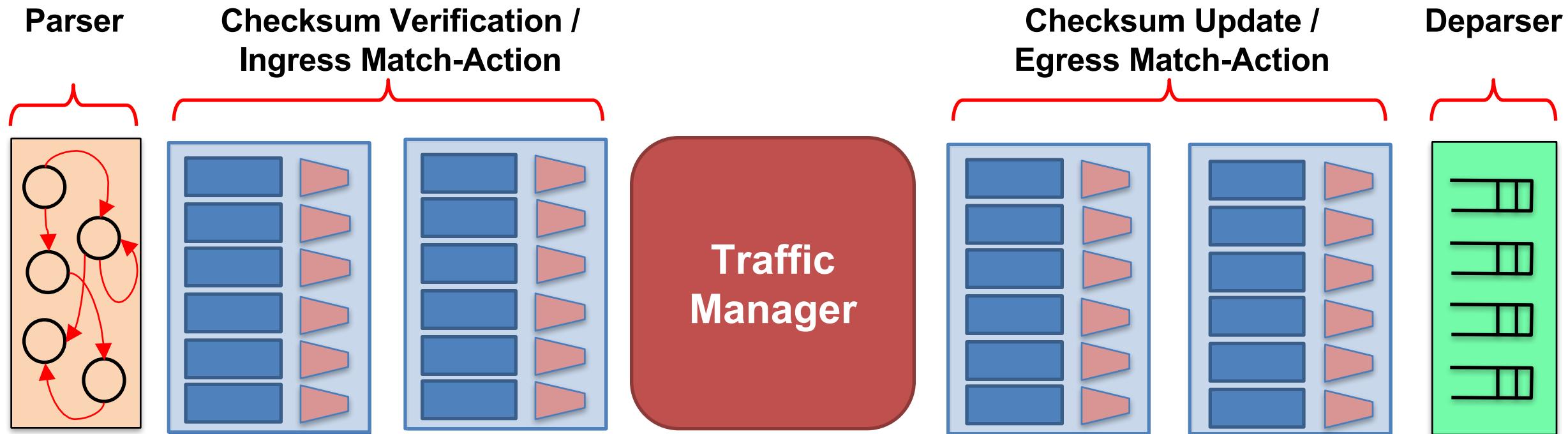


# Programming a P4 Target



# V1 Model Architecture

- Implemented on top of Bmv2's simple\_switch target
- Bmv2 is a software switch that can run on your laptop



# P4<sub>16</sub> Program Template (V1Model)

```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
    ethernet_t    ethernet;
    ipv4_t         ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
                 out headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t smeta) {
    ...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(in headers hdr,
                         inout metadata meta) {
    ...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
    ...
}
```

```
/* EGRESS PROCESSING */
control MyEgress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
    ...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
    ...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
                    inout metadata meta) {
    ...
}
/* SWITCH */
V1Switch(
    MyParser(),
    MyVerifyChecksum(),
    MyIngress(),
    MyEgress(),
    MyComputeChecksum(),
    MyDeparser()
) main;
```

# P4<sub>16</sub> Hello World (V1Model)

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet,
    out headers hdr,
    inout metadata meta,
    inout standard_metadata_t standard_metadata) {

    state start { transition accept; }

control MyVerifyChecksum(inout headers hdr, inout
metadata meta) { apply { } }

control MyIngress(inout headers hdr,
    inout metadata meta,
    inout standard_metadata_t standard_metadata) {
apply {
    if (standard_metadata.ingress_port == 1) {
        standard_metadata.egress_spec = 2;
    } else if (standard_metadata.ingress_port == 2) {
        standard_metadata.egress_spec = 1;
    }
}
```

```
control MyEgress(inout headers hdr,
    inout metadata meta,
    inout standard_metadata_t standard_metadata) {
    apply { }
}

control MyComputeChecksum(inout headers hdr, inout metadata
meta) {
    apply { }
}

control MyDeparser(packet_out packet, in headers hdr) {
    apply { }
}

V1Switch(
    MyParser(),
    MyVerifyChecksum(),
    MyIngress(),
    MyEgress(),
    MyComputeChecksum(),
    MyDeparser()
) main;
```

# P4<sub>16</sub> Hello World (V1Model)

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet, out headers hdr,
    inout metadata meta,
    inout standard_metadata_t standard_metadata) {
    state start { transition accept; }
}

control MyIngress(inout headers hdr, inout metadata meta,
    inout standard_metadata_t standard_metadata) {
    action set_egress_spec(bit<9> port) {
        standard_metadata.egress_spec = port;
    }
}






```

```
control MyEgress(inout headers hdr,
    inout metadata meta,
    inout standard_metadata_t standard_metadata) {
    apply {    }
}

control MyVerifyChecksum(inout headers hdr, inout metadata meta) {    apply {    }    }

control MyComputeChecksum(inout headers hdr, inout metadata meta) {    apply {    }    }

control MyDeparser(packet_out packet, in headers hdr) {
    apply {    }
}

V1Switch( MyParser(), MyVerifyChecksum(), MyIngress(),
MyEgress(), MyComputeChecksum(), MyDeparser() ) main;
```

Key	Action ID	Action Data
1	set_egress_spec ID	2
2	set_egress_spec ID	1

# Example: IPv4\_LPM Table



Key	Action	Action Data
10.0.1.1/32	ipv4_forward	port=1
10.0.1.2/32	ipv4_forward	port=2
*	drop	

- **Data Plane (P4) Program**
  - Defines the format of the table
    - Key Fields
    - Actions
    - Action Data
  - Performs the lookup
  - Executes the chosen action
- **Control Plane (IP stack, Routing protocols)**
  - Populates table entries with specific information
    - Based on the configuration
    - Based on automatic discovery
    - Based on protocol calculations

# Example: IPv4\_LPM Table

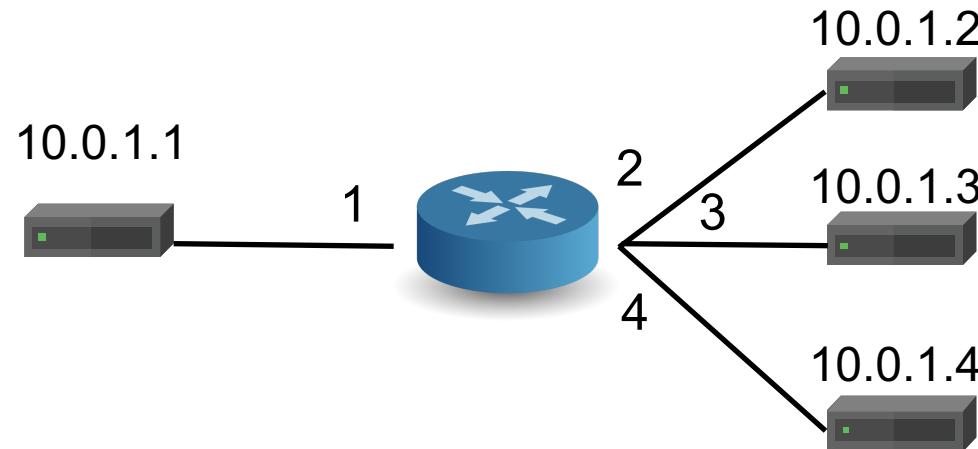
```
control MyIngress(inout headers hdr, inout metadata meta,  
    inout standard_metadata_t standard_metadata) {  
  
    action drop() {  
        mark_to_drop();  
    }  
    action ipv4_forward(egressSpec_t port) {  
        standard_metadata.egress_spec = port;  
    }  
    table ipv4_lpm{  
        key = {  
            hdr.ipv4.dstAddr: lpm;  
        }  
        actions = {  
            ipv4_forward;  
            drop;  
        }  
        size = 1024;  
        default_action = drop();  
    }  
  
    apply {  
        if (hdr.ipv4.isValid()) {  
            ipv4_lpm.apply();  
        }  
    }  
}
```

Key	Action	Action Data
10.0.1.1/32	ipv4_forward	port=1
10.0.1.2/32	ipv4_forward	port=2
*	drop	

Specify rules in a JSON file

```
{  
    "target": "bmv2",  
    "p4info": "build/acl.p4info",  
    "bmv2_json": "build/acl.json",  
    "table_entries": [  
        {  
            "table": "MyIngress.ipv4_lpm",  
            "match": {  
                "hdr.ipv4.dstAddr": ["10.0.1.1", 32]  
            },  
            "action_name": "MyIngress.ipv4_forward",  
            "action_params": {  
                "port": 1  
            },  
            .....  
        }  
    ]  
}
```

# Exercise 1: Access Control List for UDP Traffic



Key	Action	Action Data
dstIP = *, dstPort = 80	drop	
dstIP = 10.0.1.4, dstPort = *	drop	
*	NoAction	

- **Data Plane (P4) Program**
  - Defines the format of the table
    - Key Fields:
      - `hdr.ipv4.dstAddr`: ternary
      - `hdr.udp.dstPort`: ternary
    - Actions:
      - drop
      - NoAction
    - Action Data: None

- **Control Plane**
  - Populates table entries with specific information
    - Use the two rules in the example

# Exercise 1: Access Control List for UDP Traffic

```
control MyIngress(inout headers hdr, inout metadata meta,  
    inout standard_metadata_t standard_metadata) {  
  
    table udp_acl {  
        key = {  
            hdr.ipv4.dstAddr: ternary;  
            hdr.udp.dstPort: ternary;  
        }  
        actions = {  
            drop;  
            NoAction;  
        }  
        size = 1024;  
        default_action = NoAction();  
    }  
}
```

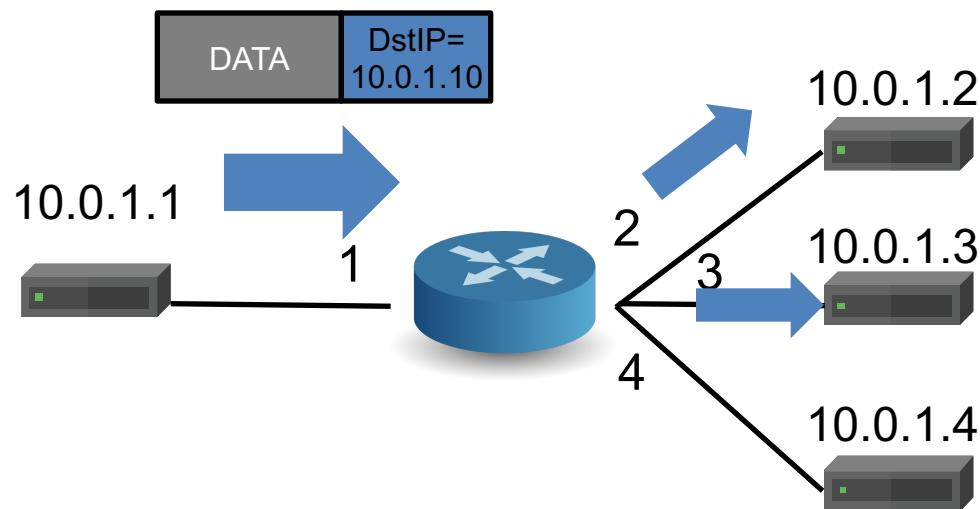
```
apply {  
    if (hdr.ipv4.isValid()) {  
        ipv4_lpm.apply();  
    if (hdr.udp.isValid()) {  
        udp_acl.apply();  
    }  
}
```

Key	Action	Action Data
dstIP = *, dstPort = 80	drop	
dstIP = 10.0.1.4, dstPort = *	drop	
*	NoAction	

Specify rules in a JSON file

```
{  
    "target": "bmv2",  
    "p4info": "build/acl.p4info",  
    "bmv2_json": "build/acl.json",  
    "table_entries": [  
        {  
            "table": "MyIngress.udp_acl",  
            "match": {  
                "hdr.udp.dstPort": [80, 65535]},  
            "action_name": "MyIngress.drop",  
            "action_params": {}  
        },  
        .....  
    ]  
}
```

# Exercise 2: Load Balancing



Key	Action	Action Data
dstIP = 10.0.1.10	select_nhop	0, 2
*	NoAction	

Key	Action	Action Data
meta.nhop = 0	set_ip_egress	10.0.1.2, 2
meta.nhop = 1	set_ip_egress	10.0.1.3, 3
*	NoAction	

- **Data Plane (P4) Program**
  - The first table picks one server
  - The second table updates the destination IP address and egress port

- **Control Plane**
  - Populates table entries with specific information
    - Use the rules in the example

# Exercise 2: Load Balancing

```
control MyIngress(inout headers hdr, inout metadata meta,  
    inout standard_metadata_t standard_metadata) {  
  
    action set_nhop(bit<16> base, bit<16> count) {  
        hash(meta.nhop, HashAlgorithm.crc16, base,  
            {hdr.ipv4.srcAddr, hdr.ipv4.dstAddr,  
             hdr.ipv4.protocol, hdr.udp.srcPort,  
             hdr.udp.dstPort}, count);  
    }  
  
    table lb_select {  
        key = {  
            hdr.ipv4.dstAddr: exact;  
        }  
        actions = {  
            set_nhop;  
            NoAction;  
        }  
        size = 1024;  
        default_action = NoAction();  
    }  
}
```

Key	Action	Action Data
dstIP = 10.0.1.10	select_nhop	0, 2
*	NoAction	

Specify rules in a JSON file

```
{  
    "target": "bmv2",  
    "p4info": "build/load_balance.p4info",  
    "bmv2_json": "build/load_balance.json",  
    "table_entries": [  
        {  
            "table": "MyIngress.lb_select",  
            "match": {  
                "hdr.ipv4.dstAddr": ["10.0.1.10", 32]  
            },  
            "action_name": "MyIngress.set_nhop",  
            "action_params": {  
                "base": 0,  
                "count": 2  
            },  
            .....  
        }  
    ]  
}
```

# Exercise 2: Load Balancing

```
control MyIngress(inout headers hdr, inout metadata meta,  
    inout standard_metadata_t standard_metadata) {  
  
    action set_ip_egress(bit<32> ip, egressSpec_t port) {  
        hdr.ipv4.dstAddr = ip;  
        standard_metadata.egress_spec = port;  
    }  
  
    table lb_set {  
        key = {  
            meta.nhop: exact;  
        }  
        actions = {  
            set_ip_egress;  
            NoAction;  
        }  
        size = 2;  
        default_action = NoAction();  
    }  
  
    apply {  
        if (hdr.udp.isValid()) {  
            lb_select.apply();  
            lb_set.apply();  
        }  
    }  
}
```

Key	Action	Action Data
meta.nhop = 0	set_ip_egress	10.0.1.2, 2
meta.nhop = 1	set_ip_egress	10.0.1.3, 3
*	NoAction	

Specify rules in a JSON file

```
{  
    "target": "bmv2",  
    "p4info": "build/load_balance.p4info",  
    "bmv2_json": "build/load_balance.json",  
    "table_entries": [  
        {  
            "table": "MyIngress.ecmp_set",  
            "match": {  
                "meta.nhop": 0},  
            "action_name": "MyIngress.set_ip_egress",  
            "action_params": {  
                "ip": "10.0.1.2"  
                "Count": 2}  
            },  
            .....  
        }  
    ]  
}
```

# Summary

- **Programmable networks: programmable switches enable fast development and deployment of new data plane functionalities**
- **Next lecture: link layer**

Thanks!  
Q&A

# V1 Model Standard Metadata

```
struct standard_metadata_t {  
    bit<9>  ingress_port;  
    bit<9>  egress_spec;  
    bit<9>  egress_port;  
    bit<32> clone_spec;  
    bit<32> instance_type;  
    bit<1>   drop;  
    bit<16> recirculate_port;  
    bit<32> packet_length;  
    bit<32> enq_timestamp;  
    bit<19> enq_qdepth;  
    bit<32> deq_timedelta;  
    bit<19> deq_qdepth;  
    bit<48> ingress_global_timestamp;  
    bit<32> lf_field_list;  
    bit<16> mcast_grp;  
    bit<1>  resubmit_flag;  
    bit<16> egress_rid;  
    bit<1>  checksum_error;  
}
```

- **ingress\_port** - the port on which the packet arrived
- **egress\_spec** - the port to which the packet should be sent to
- **egress\_port** - the port that the packet will be sent out of (read only in egress pipeline)

# P4<sub>16</sub> Types (Basic and Header Types)

```
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;
header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16> etherType;
}

header ipv4_t {
    bit<4> version;
    bit<4> ihl;
    bit<8> diffserv;
    bit<16> totalLen;
    bit<16> identification;
    bit<3> flags;
    bit<13> fragOffset;
    bit<8> ttl;
    bit<8> protocol;
    bit<16> hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}
```

## Basic Types

- **bit<n>**: Unsigned integer (bitstring) of size n
- **bit** is the same as **bit<1>**
- **int<n>**: Signed integer of size n ( $>= 2$ )
- **varbit<n>**: Variable-length bitstring

## Header Types: Ordered collection of members

- Can contain **bit<n>**, **int<n>**, and **varbit<n>**
- Byte-aligned
- Can be valid or invalid
- Provides several operations to test and set validity bit:  
**isValid()**, **setValid()**, and **setInvalid()**

## Typedef: Alternative name for a type

# P4<sub>16</sub> Types (Other Types)

```
/* Architecture */
struct standard_metadata_t {
    bit<9> ingress_port;
    bit<9> egress_spec;
    bit<9> egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1> drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    ...
}

/* User program */
struct metadata {
    ...
}
struct headers {
    ethernet_t    ethernet;
    ipv4_t        ipv4;
}
```

## Other useful types

- **Struct:** Unordered collection of members (with no alignment restrictions)
- **Header Stack:** array of headers
- **Header Union:** one of several headers

# P4<sub>16</sub> Controls

---

- **Similar to C functions (without loops)**
- **Can declare variables, create tables, instantiate externs, etc.**
- **Functionality specified by code in apply statement**
- **Represent all kinds of processing that are expressible as DAG:**
  - Match-Action Pipelines
  - Deparsers
  - Additional forms of packet processing (updating checksums)
- **Interfaces with other blocks are governed by user- and architecture-specified types (typically headers and metadata)**

# P4<sub>16</sub> Tables

---

- **The fundamental unit of a Match-Action Pipeline**
  - Specifies what data to match on and match kind
  - Specifies a list of *possible* actions
  - Optionally specifies a number of table **properties**
    - Size
    - Default action
    - Static entries
    - etc.
- **Each table contains one or more entries (rules)**
- **An entry contains:**
  - A specific key to match on
  - A **single** action that is executed when a packet matches the entry
  - Action data (possibly empty)

# Example: Simple Actions

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {

    action swap_mac(inout bit<48> src,
                     inout bit<48> dst) {
        bit<48> tmp = src;
        src = dst;
        dst = tmp;
    }

    apply {
        swap_mac(hdr.ethernet.srcAddr,
                  hdr.ethernet.dstAddr);
        std_meta.egress_spec = std_meta.ingress_port;
    }
}
```

- **Very similar to C functions**
- **Can be declared inside a control or globally**
- **Parameters have type and direction**
- **Variables can be instantiated inside**
- **Many standard arithmetic and logical operations are supported**
  - +, -, \*
  - ~, &, |, ^, >>, <<
  - ==, !=, >, >=, <, <=
  - No division/modulo
- **Non-standard operations:**
  - Bit-slicing: [m:l] (works as l-value too)
  - Bit Concatenation: ++

# IPv4\_LPM Table

```
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

# Match Kinds

```
/* core.p4 */
match_kind {
    exact,
    ternary,
    lpm
}

/* v1model.p4 */
match_kind {
    range,
    selector
}

/* Some other architecture */
match_kind {
    regexp,
    fuzzy
}
```

- **The type `match_kind` is special in P4**
- **The standard library (`core.p4`) defines three standard match kinds**
  - Exact match
  - Ternary match
  - LPM match
- **The architecture (`v1model.p4`) defines two additional match kinds:**
  - range
  - selector
- **Other architectures may define (and provide implementation for) additional match kinds**

# Defining Actions for L3 forwarding

```
/* core.p4 */
action NoAction() {
}

/* basic.p4 */
action drop() {
    mark_to_drop();
}

/* basic.p4 */
action ipv4_forward(macAddr_t dstAddr,
                     bit<9> port) {
    ...
}
```

- Actions can have two different types of parameters
  - Directional (from the Data Plane)
  - Directionless (from the Control Plane)
- Actions that are called directly:
  - Only use directional parameters
- Actions used in tables:
  - Typically use directionless parameters
  - May sometimes use directional parameters too



# Applying Tables in Controls

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    table ipv4_lpm {
        ...
    }
    apply {
        ...
        ipv4_lpm.apply();
        ...
    }
}
```

# FAQs

---

- **Can I apply a table multiple times in my P4 Program?**
  - No (except via resubmit / recirculate)
- **Can I modify table entries from my P4 Program?**
  - No (except for direct counters)
- **What happens upon reaching the reject state of the parser?**
  - Architecture dependent
- **How much of the packet can I parse?**
  - Architecture dependent