Introduction to P4 and an application

SING-lab HKUST
30/12/2017

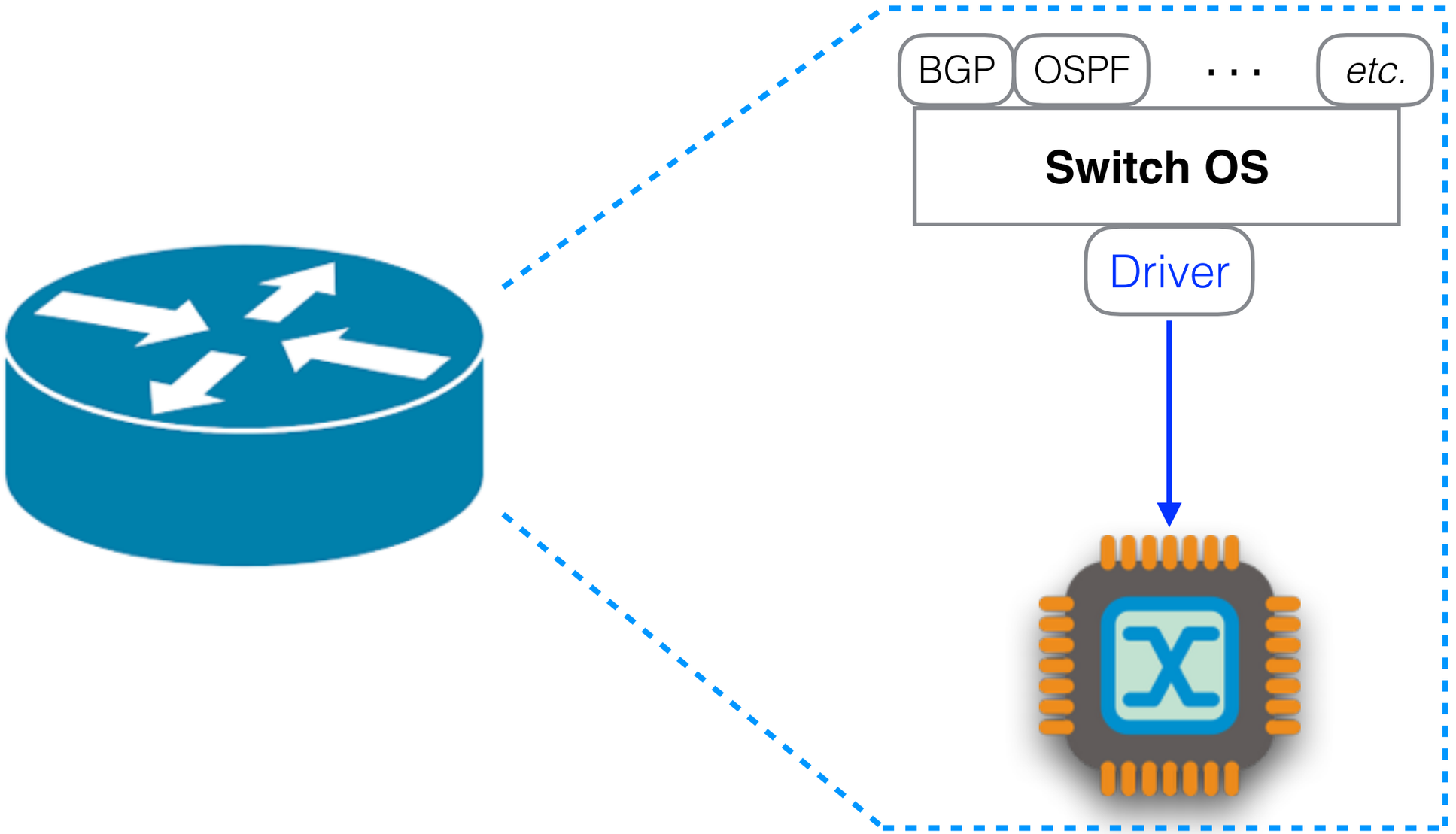
Outline

- Fixed-function *v.s.* Programmable dataplane
- Programmable Switch Overview
- P4 Overview
- An application: NetCache (SOSP 17’)

Outline

- Fixed-function *v.s.* Programmable dataplane
- Programmable Switch Overview
- P4 Overview
- An application: NetCache (SOSP 17')

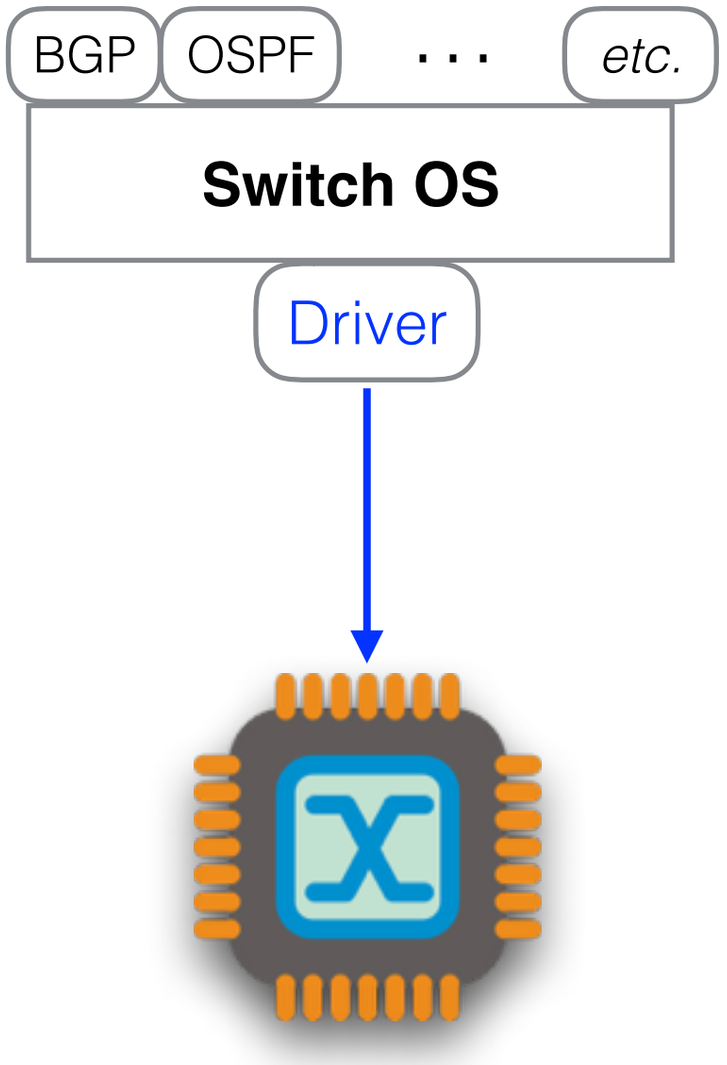
Switch General Architecture



Fixed-function: “Bottom Up”

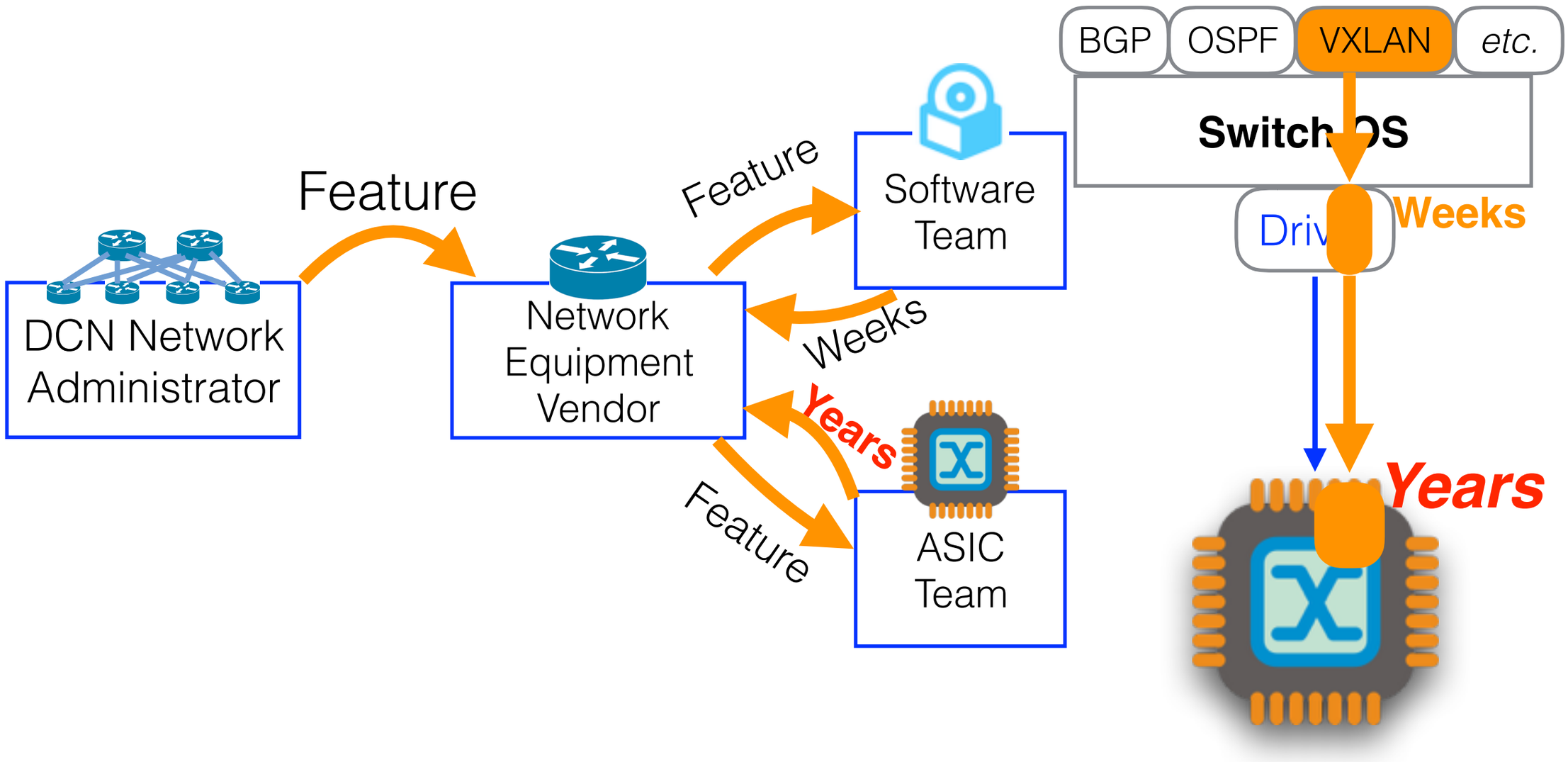
Network Admin use features

Chip makers designed features



Fixed-function ASIC

Fixed-function **Problem**

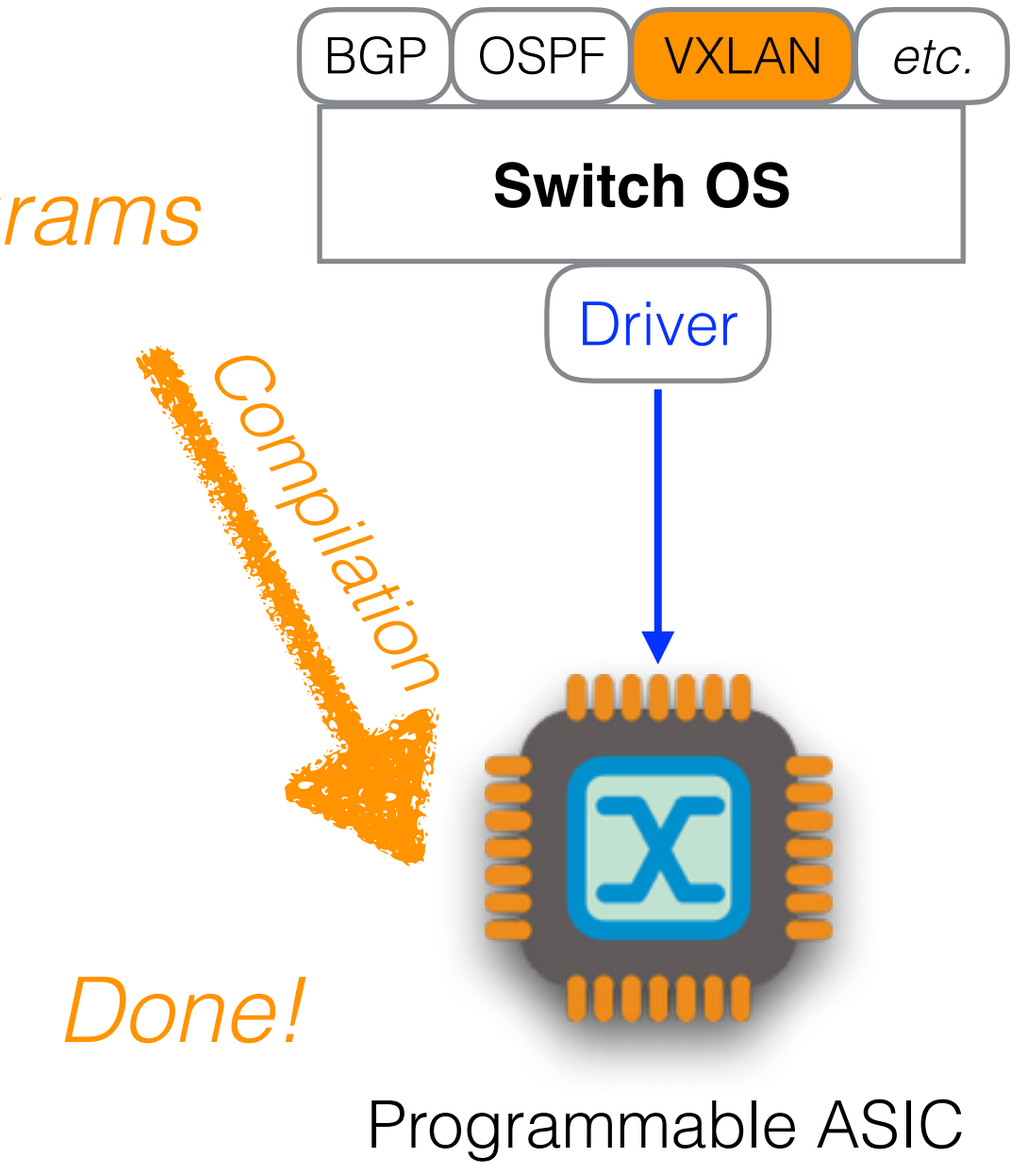


Programmable: “Top Down”

Network Admin write programs

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

```
parser MyParser(packet_in packet,
        out headers hdr,
        inout metadata meta,
        inout standard_metadata_t std_meta) {
  state start {
    packet.extract(hdr.ethernet);
    transition accept;
} }
```



Why not all switches built in a programmable manner?

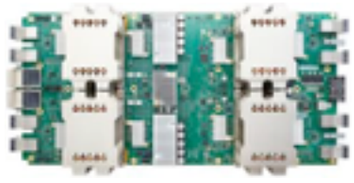
“Programmable switches are **10-100x slower** than fixed-function switches. They cost more and consume more power.”

Belief in network community for long

Why rule is broken?

Domain Specific Processors

Computers	Graphics	Machine Learning	Networking
Java Compiler	**OpenCL Compiler**	**TensorFlow Compiler**	***P4* Compiler**



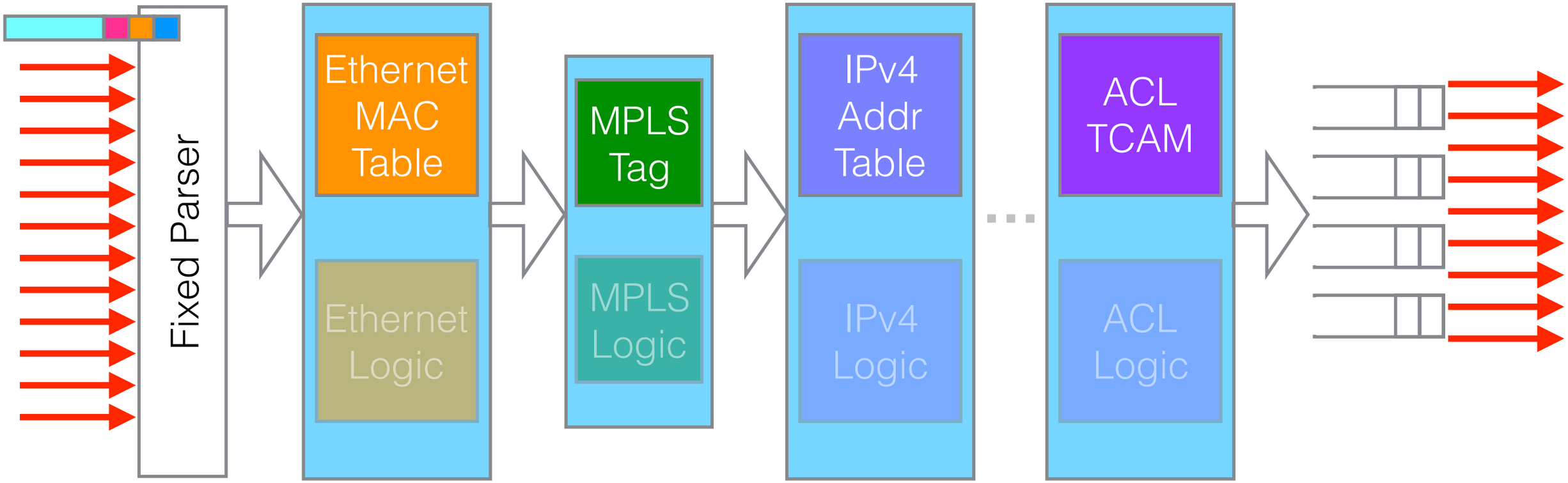
PISA (Tofino)

Outline

- Fixed-function *v.s.* Programmable dataplane
- Programmable Switch Overview
- P4 Overview
- An application: NetCache (SOSP 17')

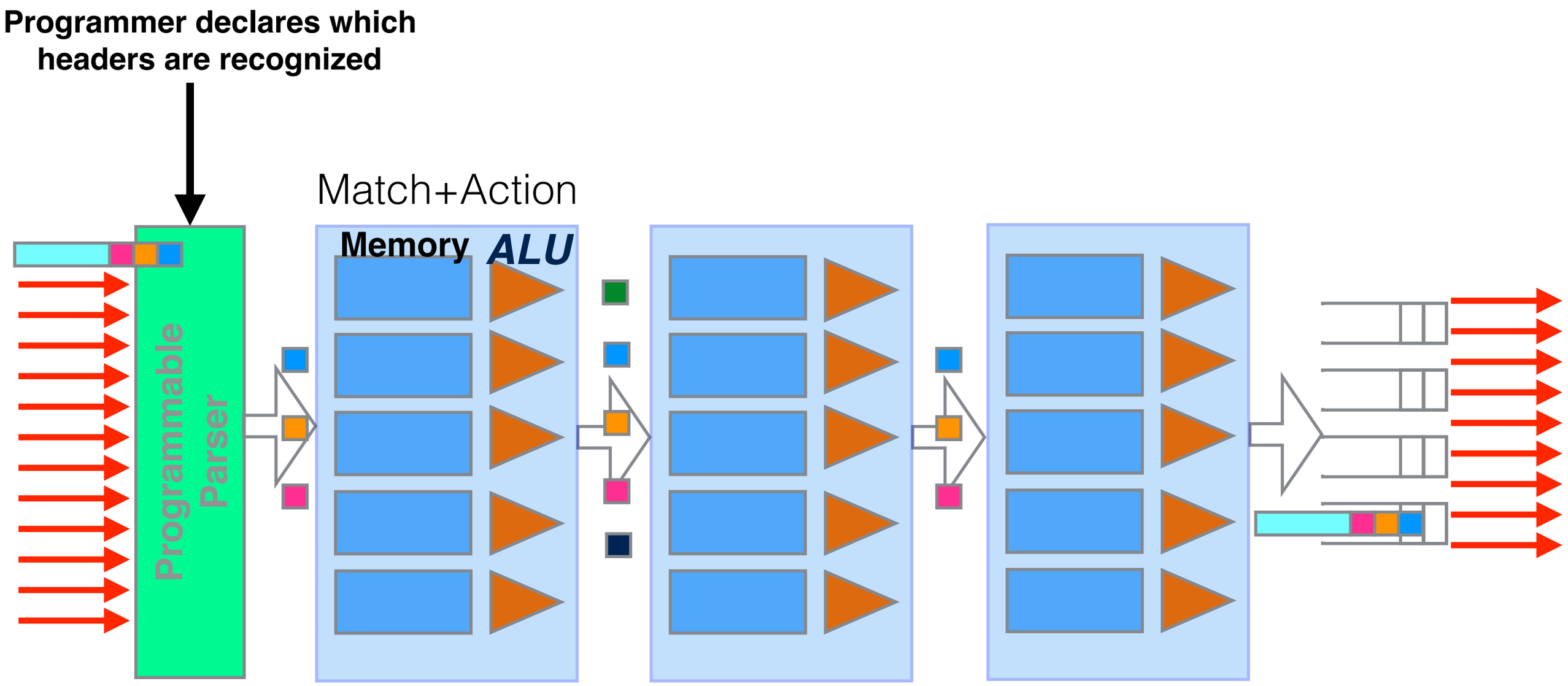
Fix-Function Switch

Same architecture for 20 years



Features and table-sizes are baked in at design time

PISA (Protocol Independent Switch Architecture)

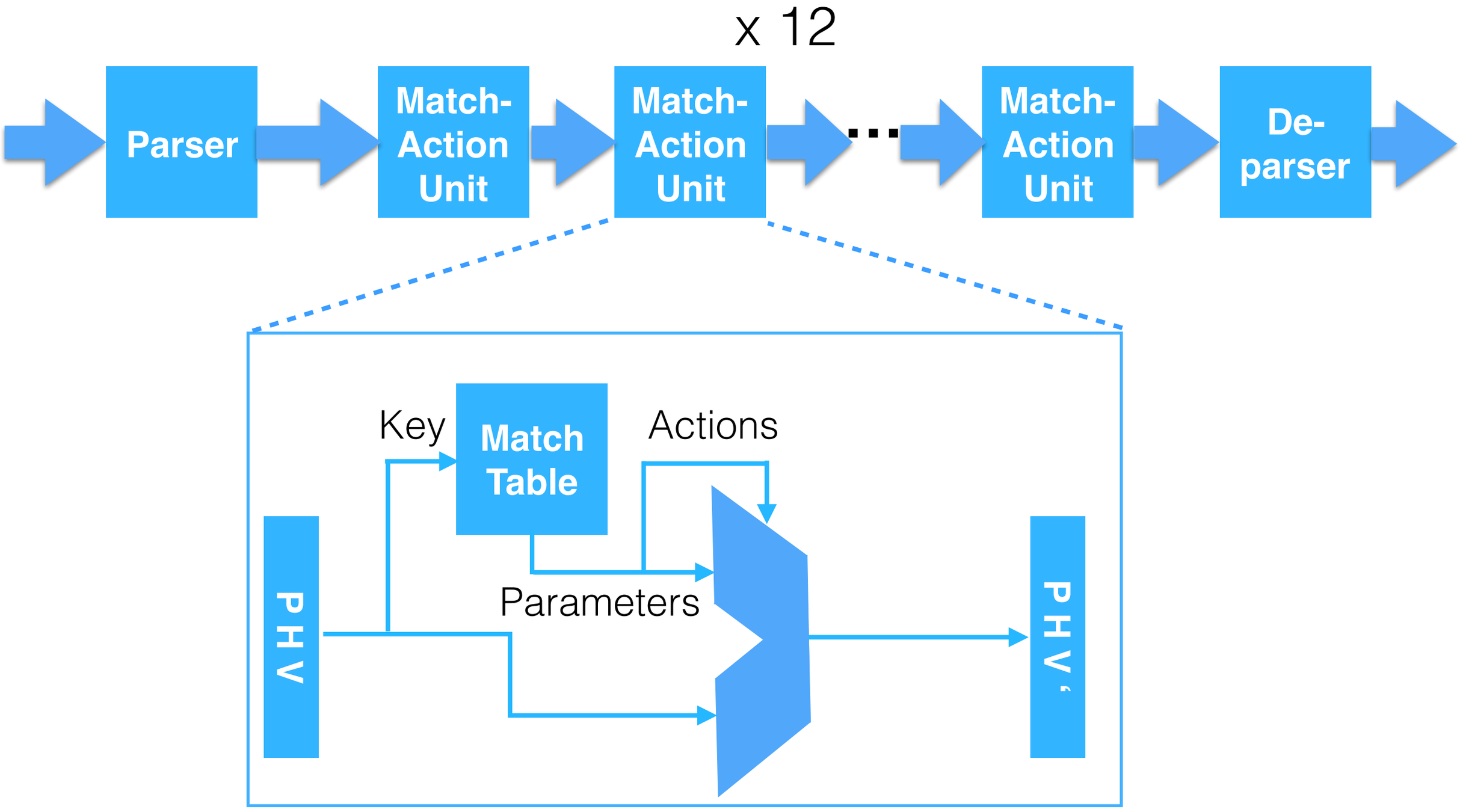


Switch Overview

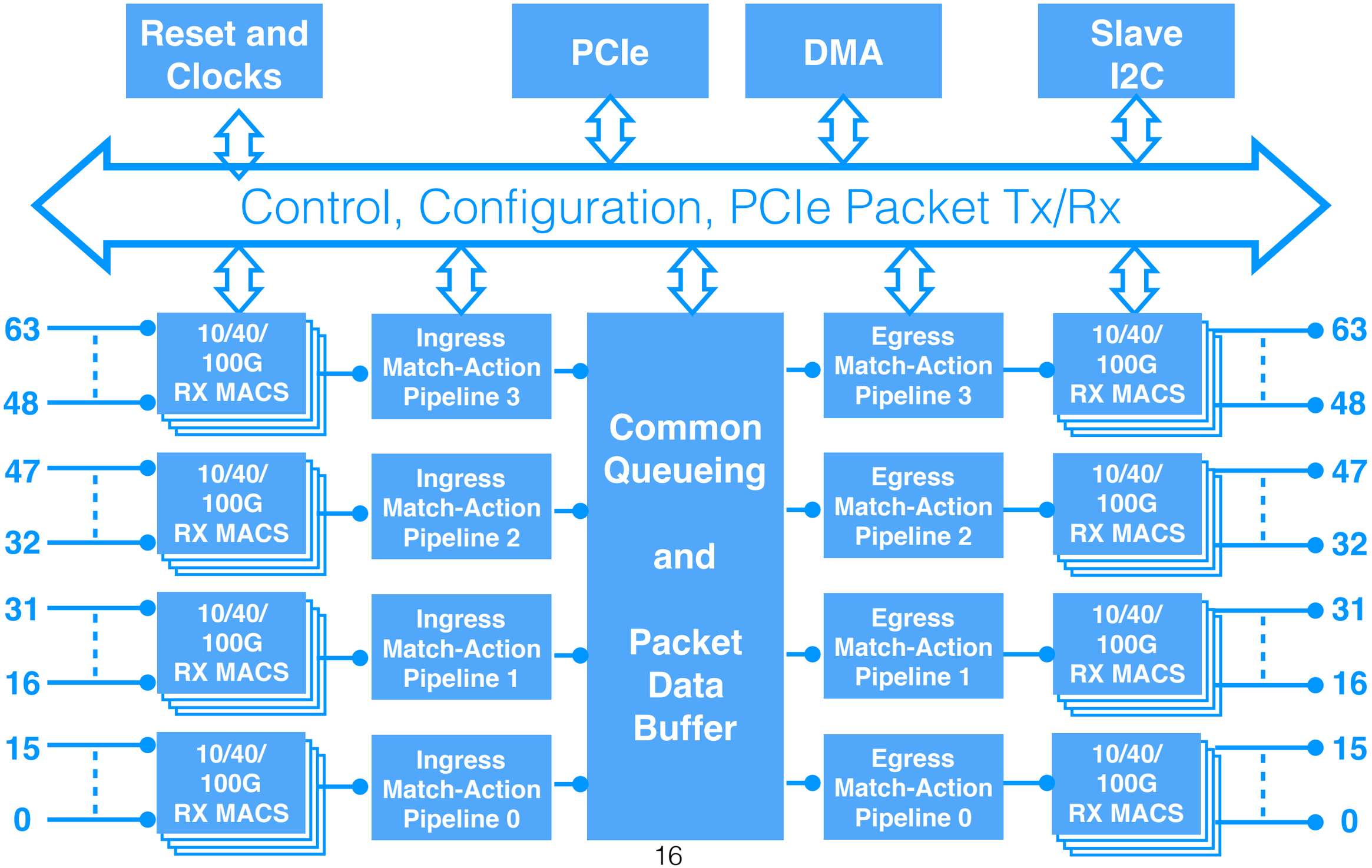
Barefoot Networks 10k family first utilize the Match-Action-Pipeline architecture

- 6.4Tbps of non-blocking throughput
- 4.4 billion packets/second of forwarding capacity
- High-speed CPU interface
- 22 MB unified packet buffer

Match-Action Pipeline



10k series switch architecture



Feature Overview

Unified Traffic Manager

- 22MB on-chip packet buffer
- Up to 9k jumbo frame
- Support for 256 ports
  - **8** Ingress priority accounting
  - **8** Egress queues
- 64k entry multicast table

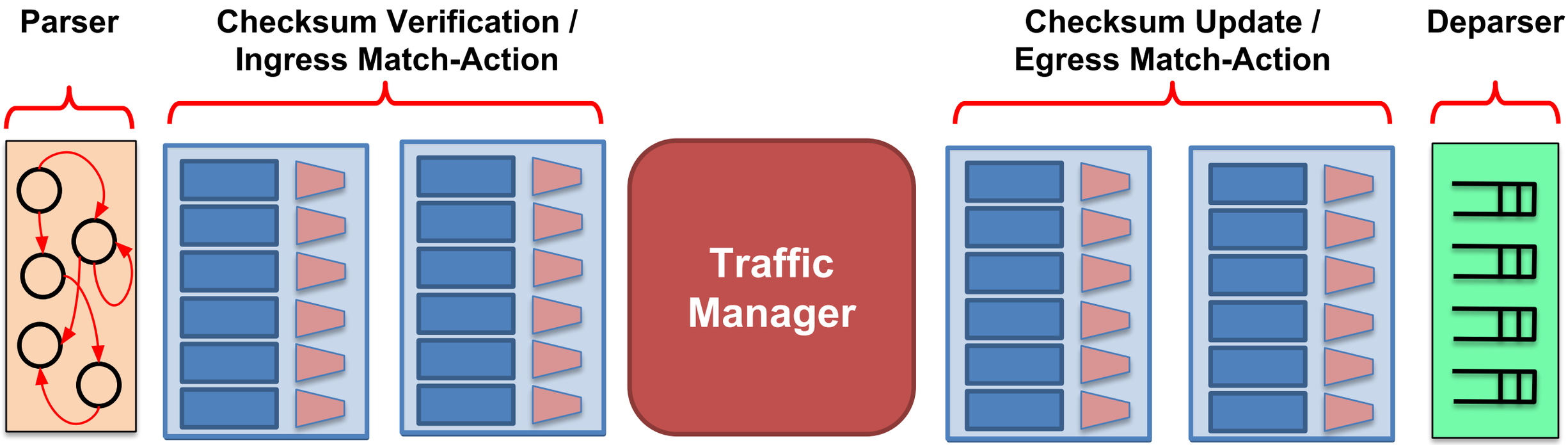
Hight Speed Management

- PCIe Gen3 x4 interface
- SPI interface for PCIe firmware
- 4 x 25Gbps SerDes for up to 100Gbps packet transfer

Outline

- Fixed-function *v.s.* Programmable dataplane
- Programmable Switch Overview
- P4 Overview
- An application: NetCache (SOSP 17')

V1Model Architecture



P4$_{16}$ 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$_{16}$ 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$_{16}$ 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;
```

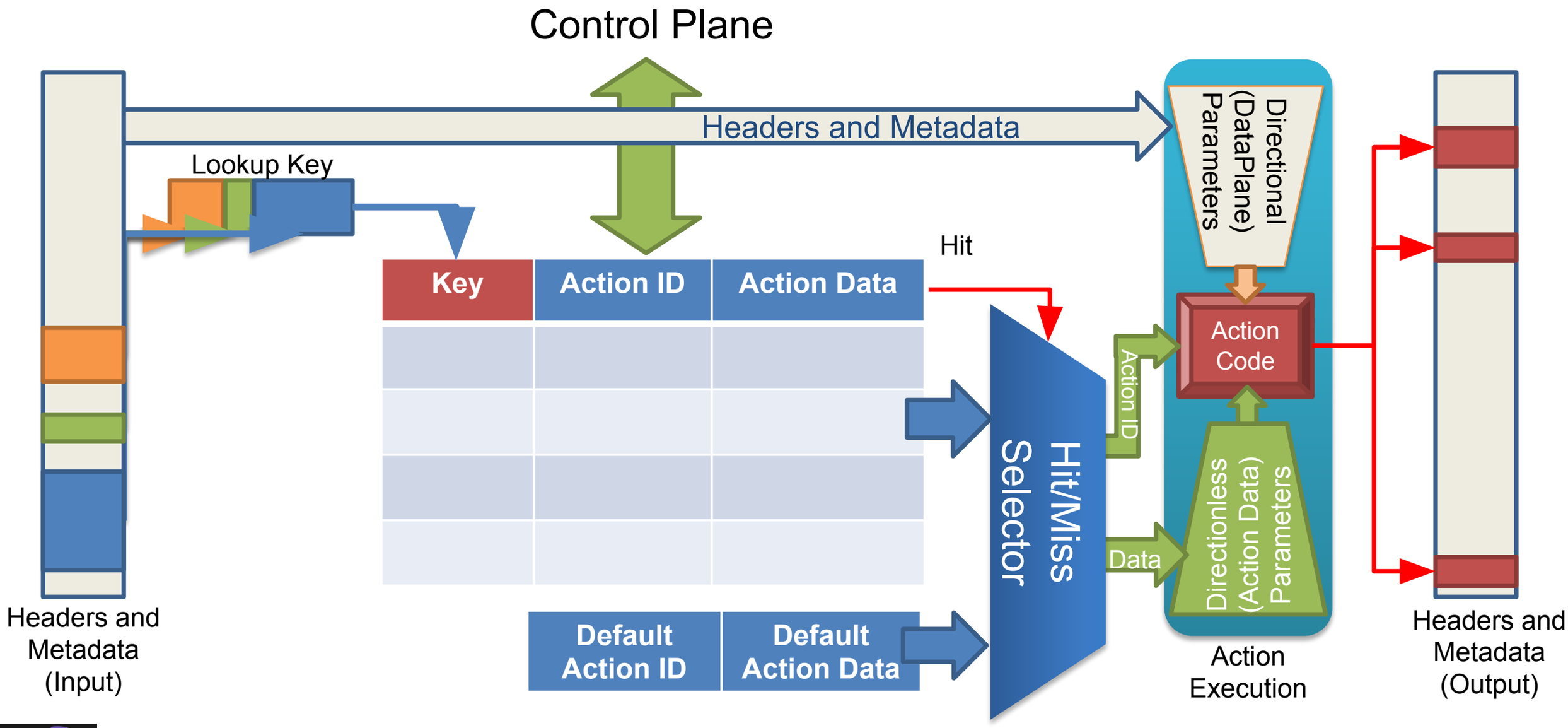
P4$_{16}$ Hello World (V1Model)

```
#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;
    }
    table forward {
        key = { standard_metadata.ingress_port: exact; }
        actions = {
            set_egress_spec;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }
    apply {   forward.apply();   }
}
```

```
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 Name	Action Data
1	set_egress_spec	2
2	set_egress_spec	1

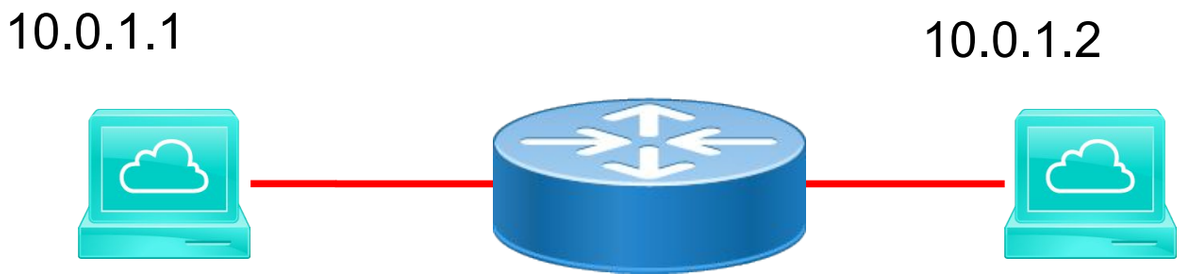
Tables: Match-Action Processing



IPv4_LPM Table

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

Example: IPv4_LPM Table



Key	Action	Action Data
10.0.1.1/32	ipv4_forward	dstAddr=00:00:00:00:01:01 port=1
10.0.1.2/32	drop	
*`	NoAction	

- **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

Outline

- Fixed-function *v.s.* Programmable dataplane
- Programmable Switch Overview
- P4 Overview
- An application: NetCache (SOSP 17’)

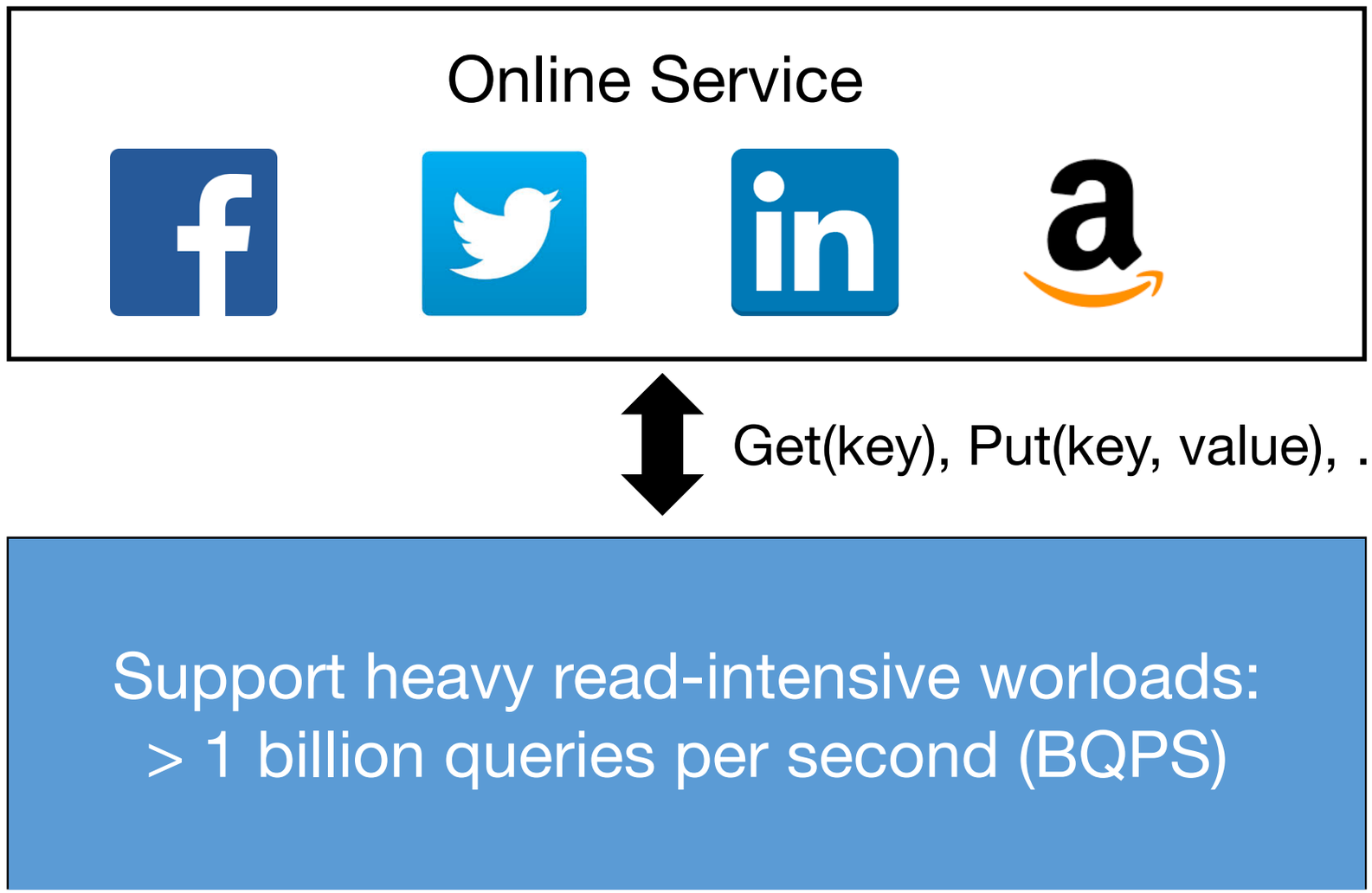
NetCache: Balancing Key-Value Stores with Fast In-Network Caching

Xin Jin

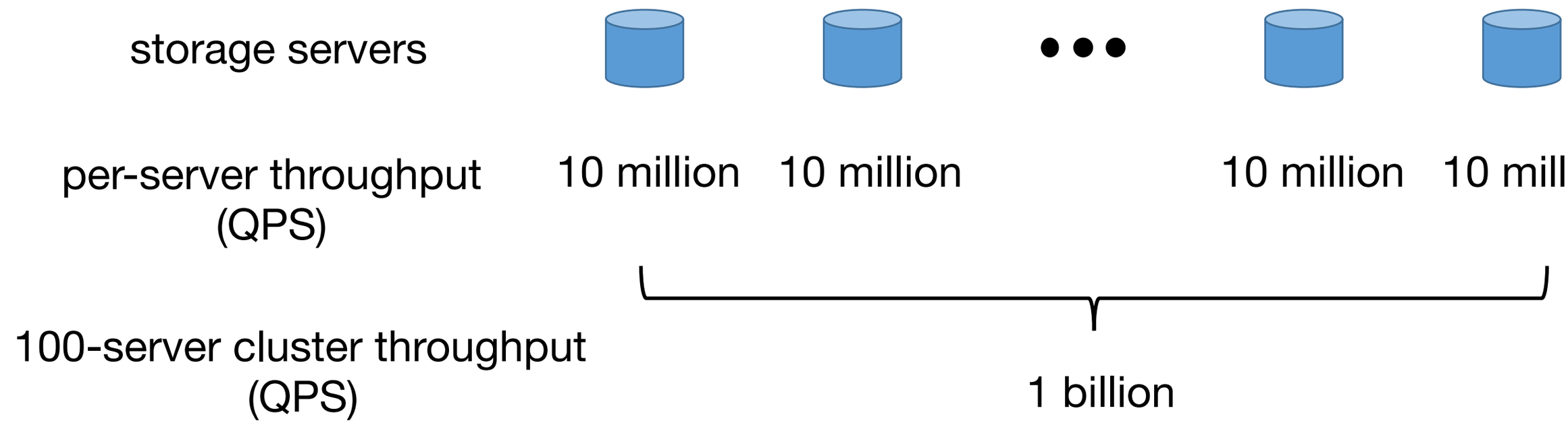
Xiaozhou Li, Haoyu Zhang, Robert Soule, Jeongkeun Lee,
Nate Foster, Changhoon Kim, Ion Stoica

Berkeley UNIVERSITY OF CALIFORNIA
JOHNS HOPKINS UNIVERSITY
BAREFOOT NETWORKS
PRINCETON UNIVERSITY
Università della Svizzera italiana
CORNELL UNIVERSITY FOUNDED A.D. 1865

Key-value stores power online services



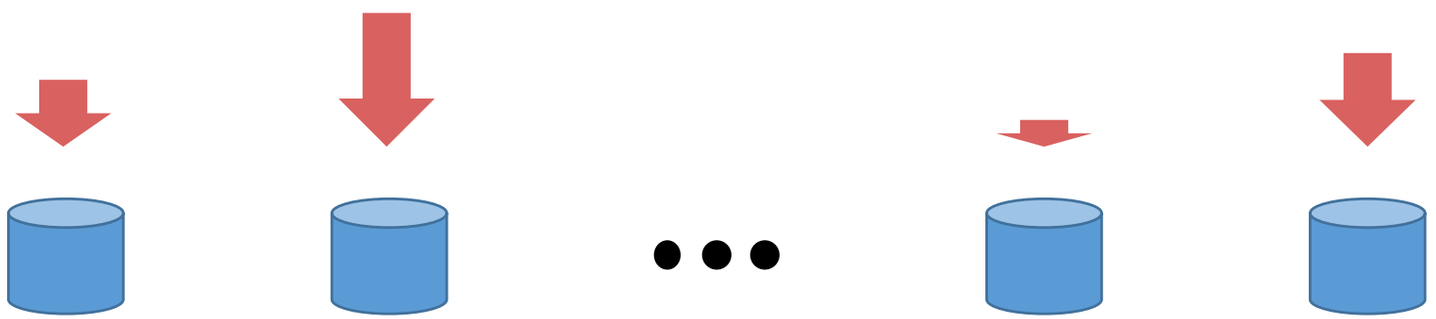
Scale out key-value stores for high-performance



Key challenge: Dynamic load balancing

load

storage servers



per-server throughput (QPS)

~~10 million~~ 5 million

~~10 million~~ 10 million

~~10 million~~ 2 million

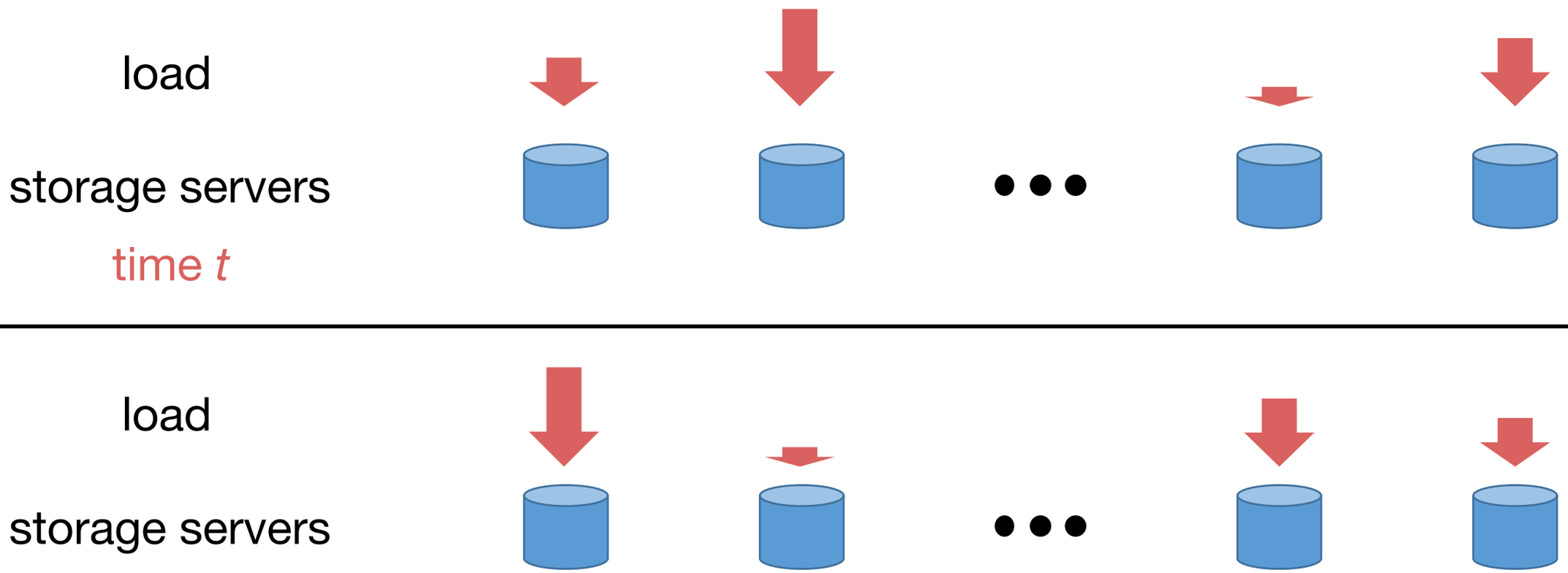
~~10 million~~ 7 million

100-server cluster throughput (QPS)

~~1 billion~~ 0.2 billion

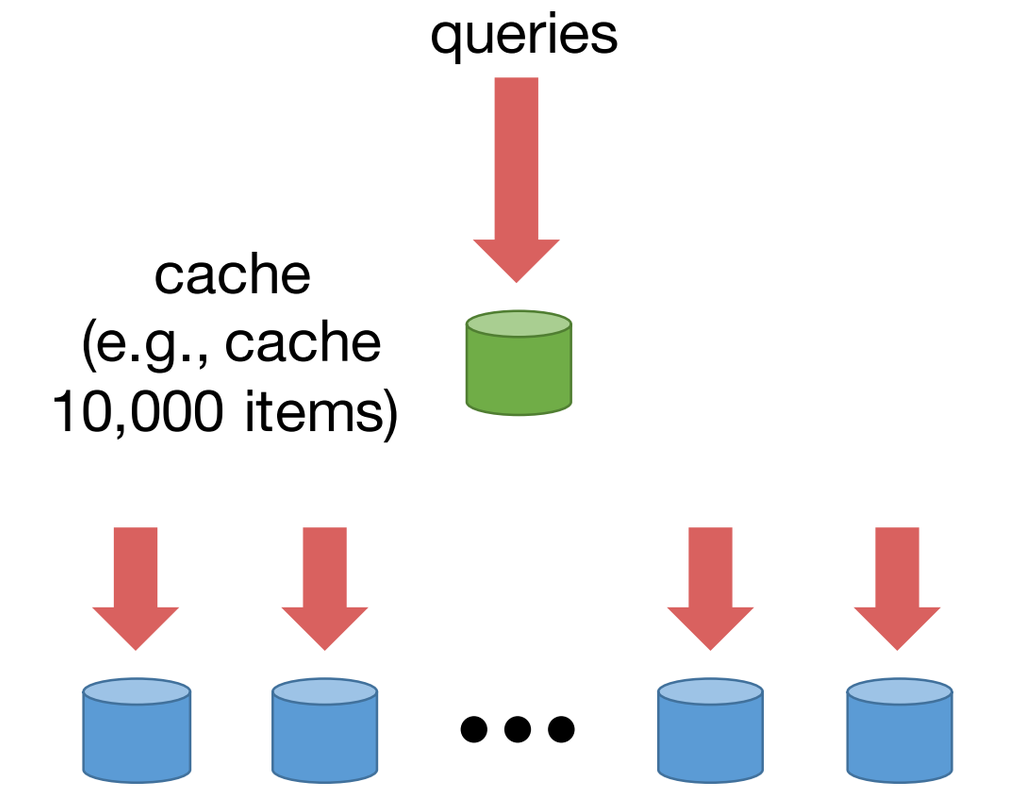
low throughput + high tail latency

Key challenge: Dynamic load balancing



How to handle highly-skewed and rapidly-changing workloads?

Fast, small cache for load balancing



- Cache **O(*N* log *N*)** items [Fan, SOCC'11]
  - N: number of servers
- **Performance guarantee**
  - Throughput: N·T
    - T: per-server throughput
  - Latency: bounded queue length (no server receives more than T load)
  - Regardless of workload **skewness**
- **Requirement**
  - **Cache throughput ≈ N·T**

Towards in-memory key-value stores

cache layer

O(10) MQPS

?

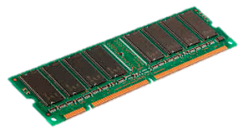
O(1) BQPS

storage layer

flash/disk

per: O(100) KQPS
total: O(10) MQPS

towards
BQPS
key-value
stores

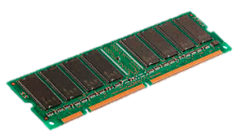


in-memory

per: O(10) MQPS
total: O(1) BQPS

Towards in-memory key-value stores

cache layer


in-memory

O(10) MQPS


in-network

O(1) BQPS

storage layer


flash/disk

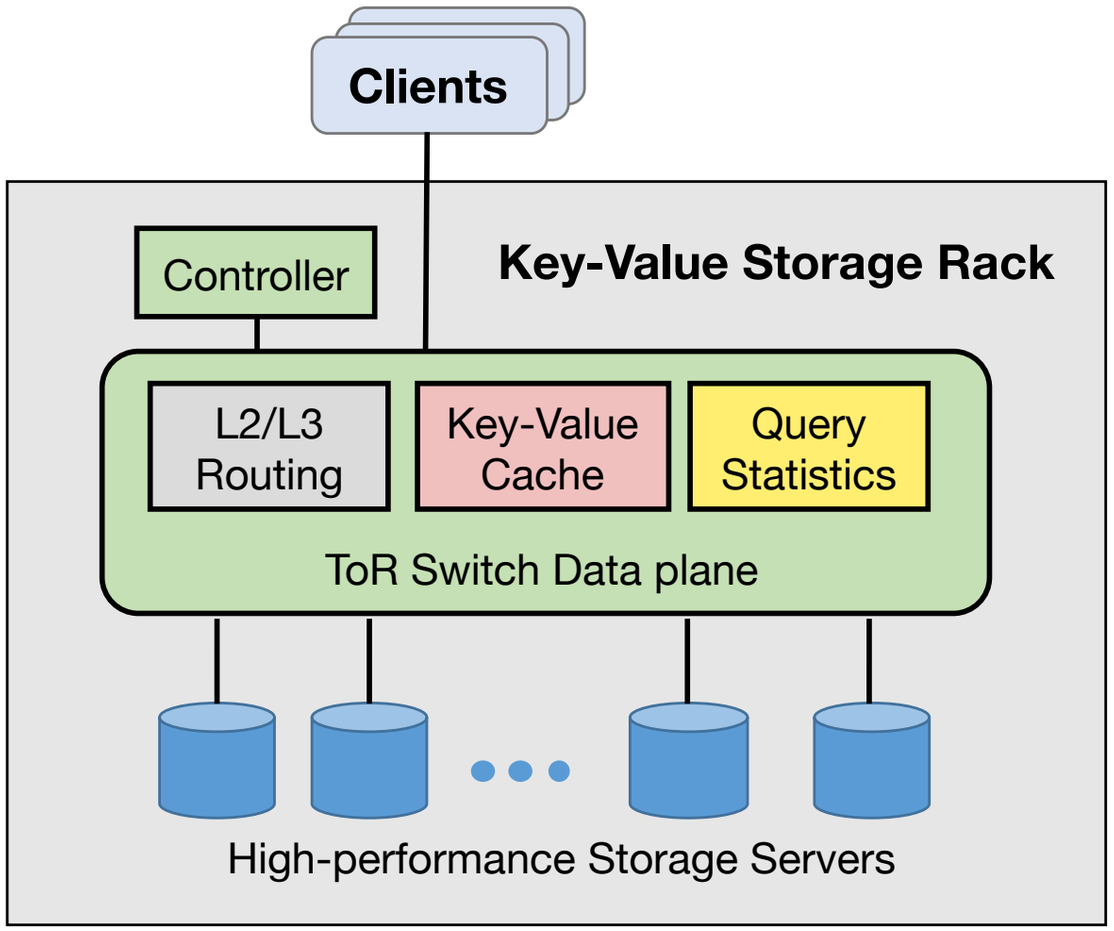
per: O(100) KQPS
total: O(10) MQPS

towards
BQPS
key-value
stores


in-memory

per: O(10) MQPS
total: O(1) BQPS

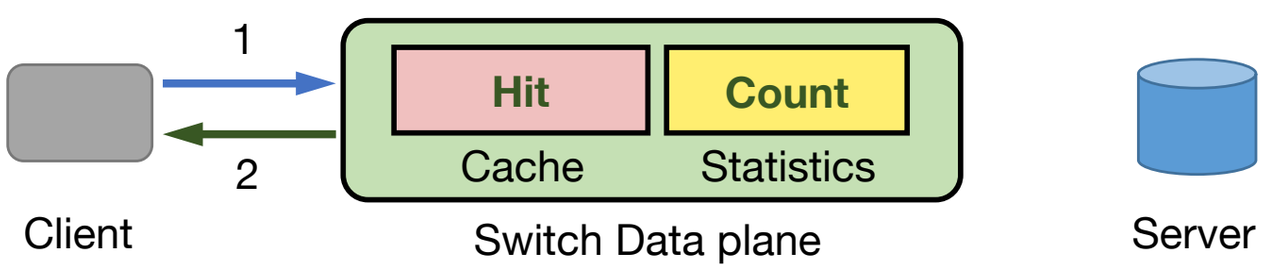
NetCache Architecture



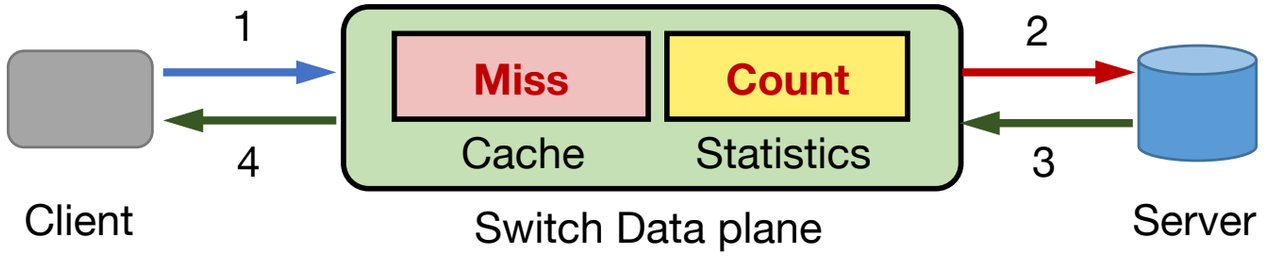
- **Performance guarantee**
  - BQPS throughput with bounded latency with a single rack
  - Regardless of workload skewness
- **Data plane**
  - Unmodified routing
  - Key-value cache to serve hot items
  - Query statistics to detect hot items
- **Control plane**
  - Update cache with hot items
  - Handle dynamic workloads

Query Handling

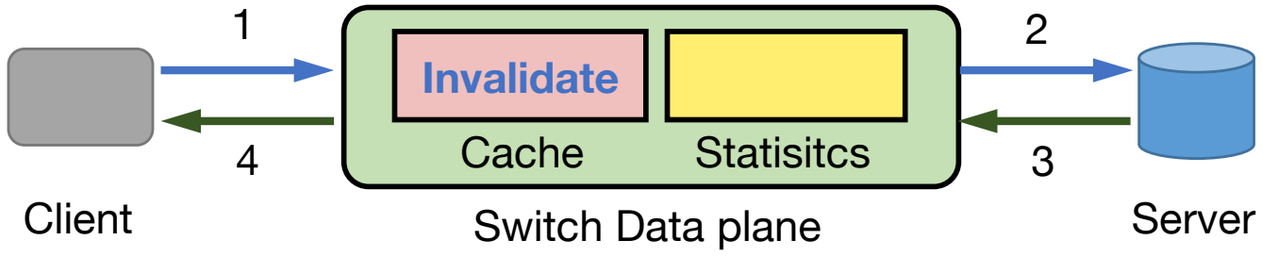
Read Query:
Cache Hit



Read Query:
Cache Miss

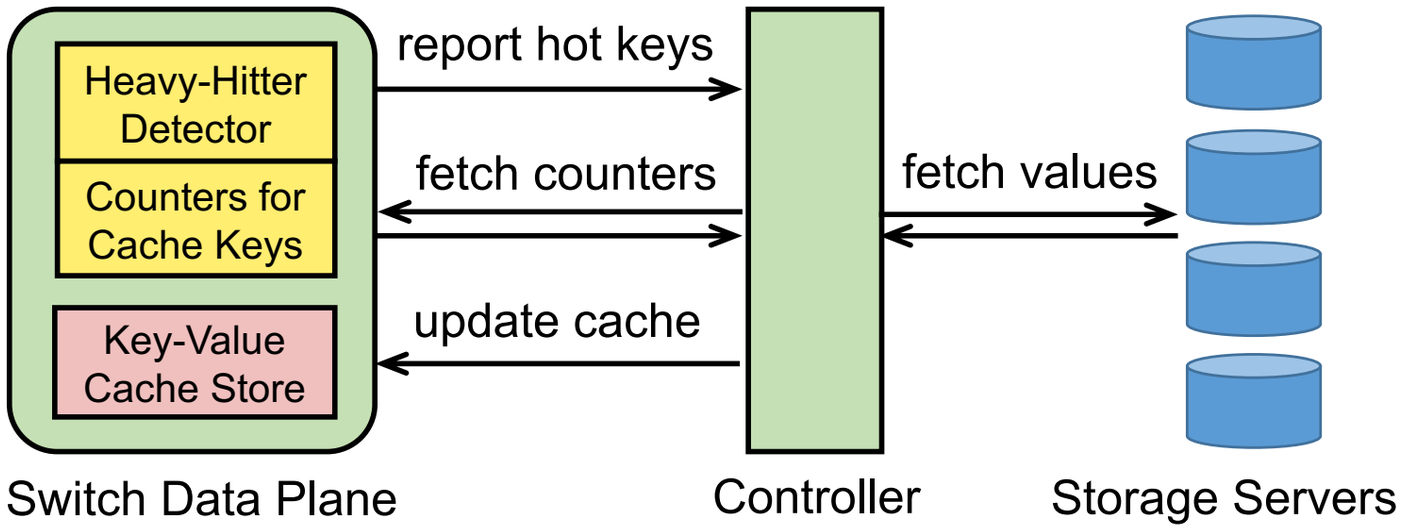


Write Query



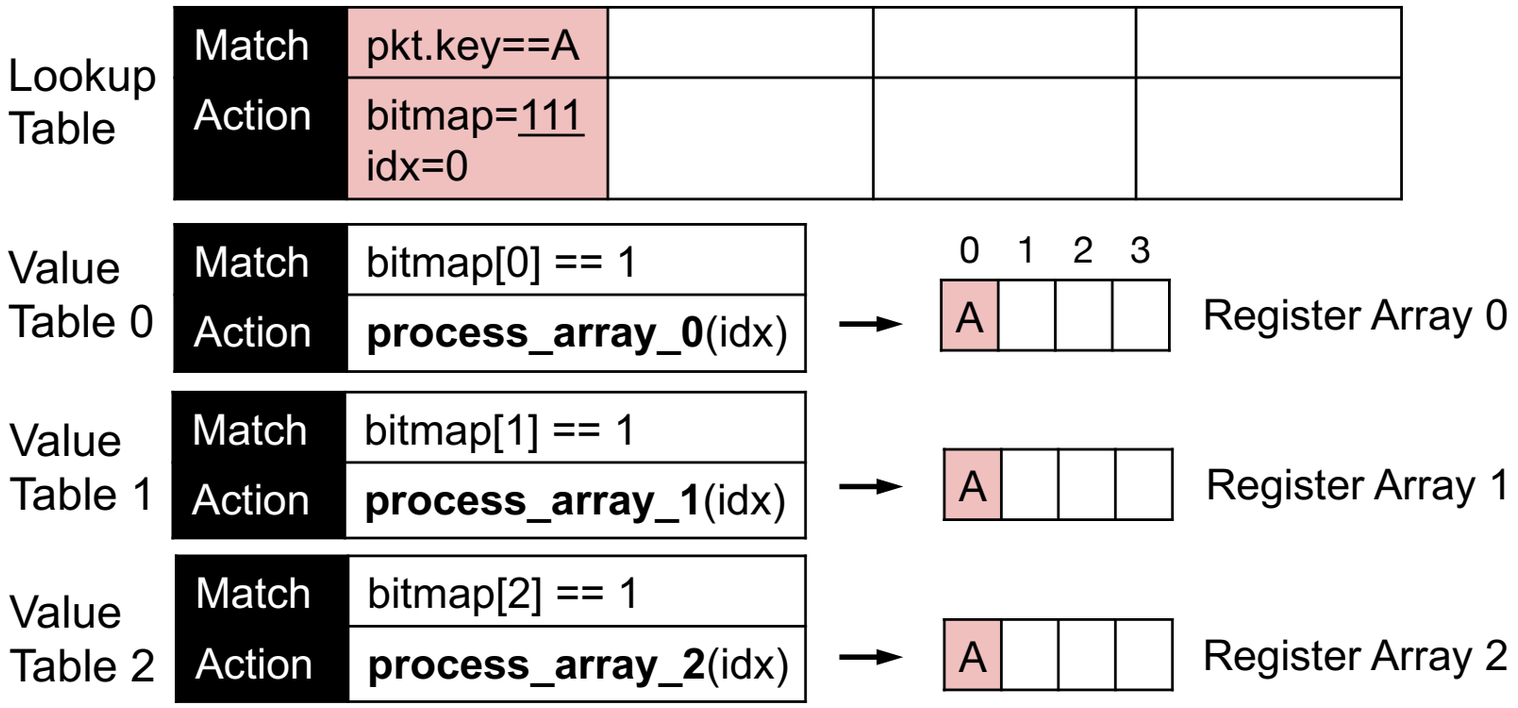
Cache coherence: write-through in the data plane

Cache Update



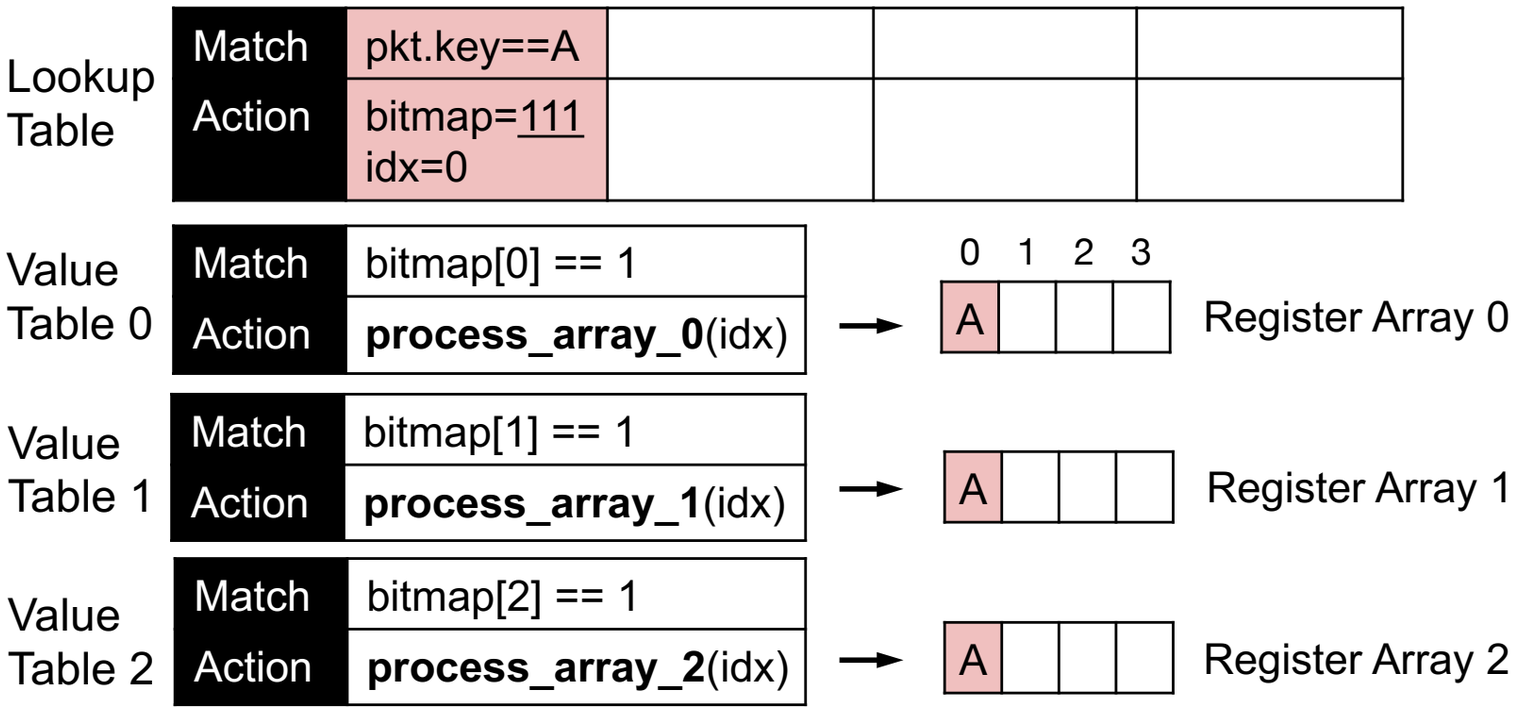
- Compare counters of **new hot keys** and **cached keys**
- Use **sampling** to avoid fetch counters of all cached keys

Variable-Length On-Chip Key-Value Cache



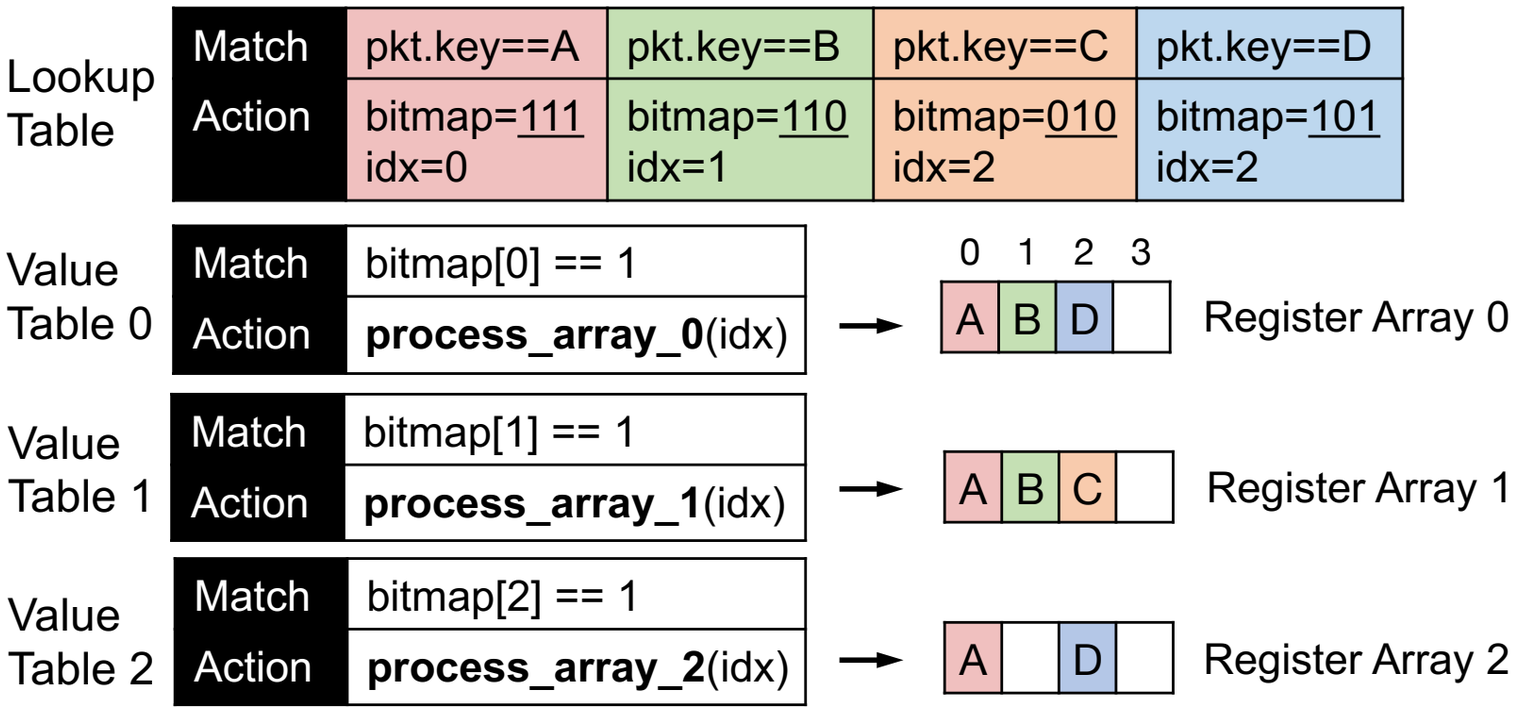
- Lookup table: map a key to a bitmap and an index
- Value table: store value in register arrays

Variable-Length On-Chip Key-Value Cache



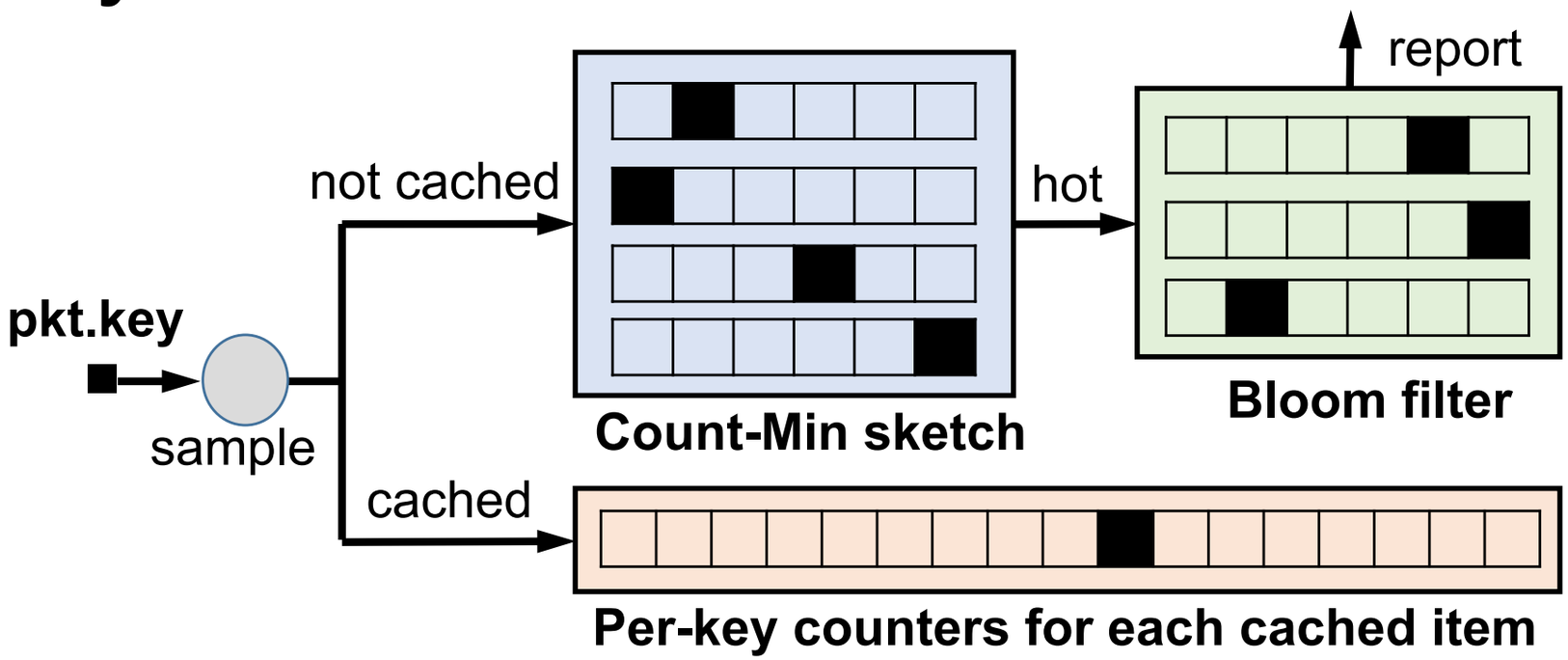
- Lookup table: map a key to a bitmap and an index
- Value table: store value in register arrays

Variable-Length On-Chip Key-Value Cache



- Lookup table: map a key to a bitmap and an index
- Value table: store values in register arrays

Query Statistics



- New hot key
  - Count-Min sketch: report new hot keys
  - Bloom filter: remove duplicate hot key reports
- Cached key: per-key counter array
- Sample: reduce memory usage

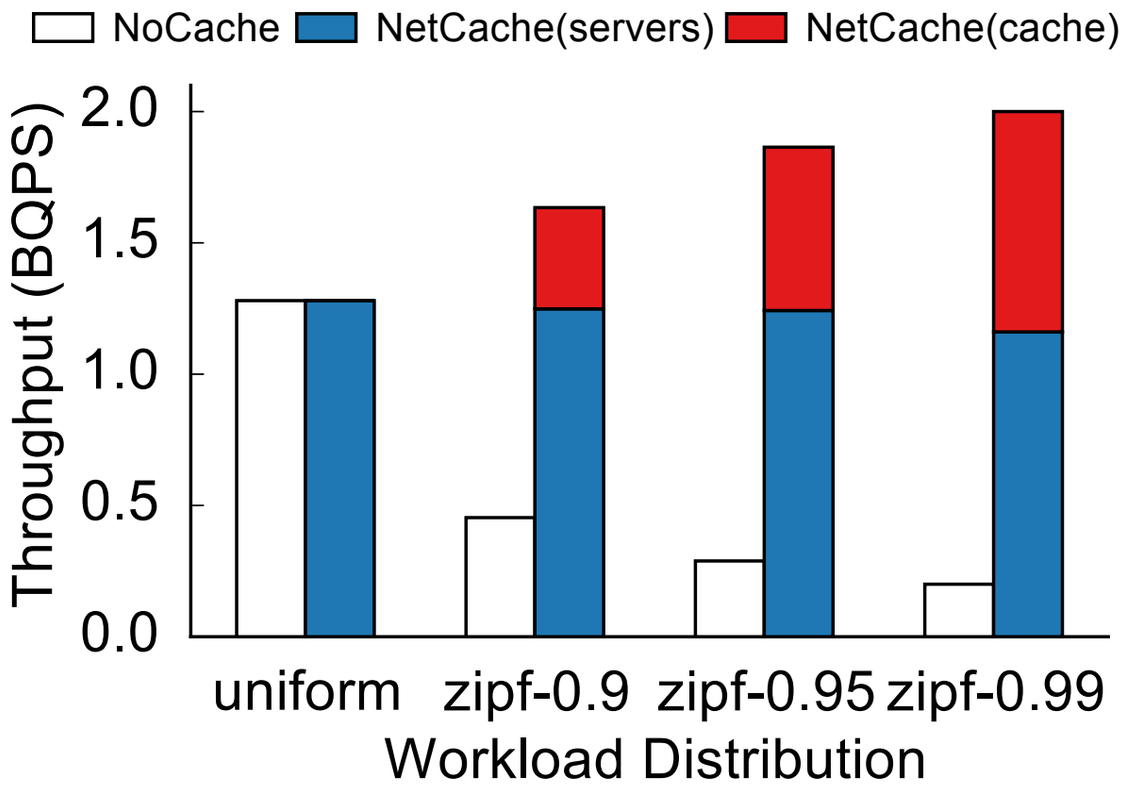
Implementation

- **Switch: Barefoot Tofino**
  - Throughput: 6.5 Tbps, 4+ bpps; Latency: <1 us
  - Routing: standard L3 routing
  - Key-value cache: 64K items with 16-byte keys and 128-byte values
  - Query statistics: 256K entries for Count-Min sketch, 768K entries for Bloom filter

- **Storage Server**
  - 16-core Intel Xeon E5-2630, 128 GB memory, 40Gbps Intel XL710 NIC
  - Intel DPDK for optimized IO, TommyDS for in-memory key-value store
  - Throughput: 10 MQPS; Latency: 7 us

Evaluation: System Performance

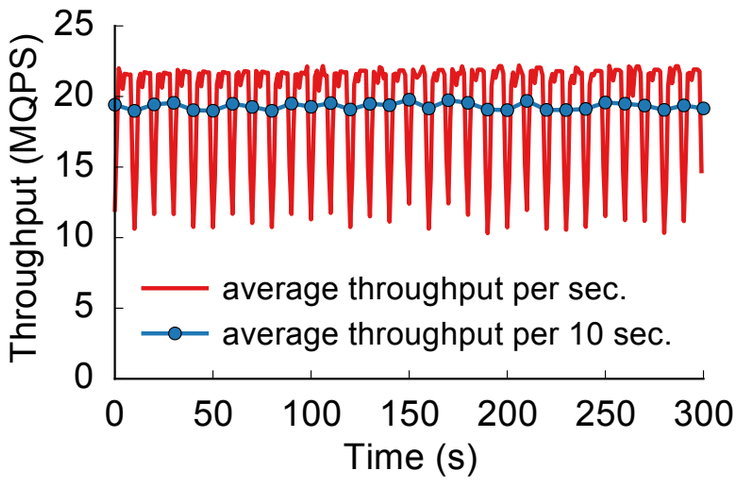
Throughput of a key-value storage rack with one Tofino switch and 128 storage servers.



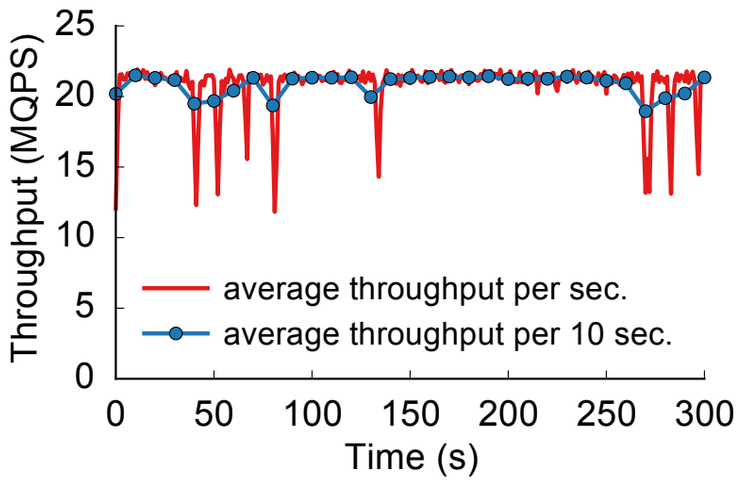
NetCache provides **3-10x throughput improvements**.

Evaluation: Handling Workload Dynamics

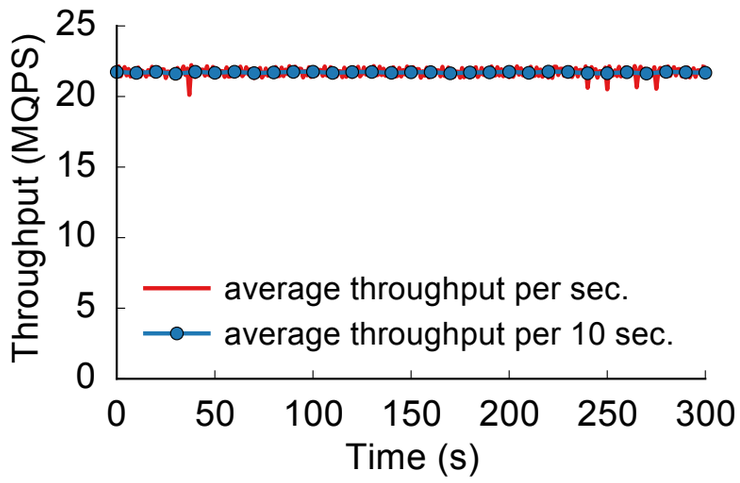
hot-in workload
(radical change)



random workload
(moderate change)



hot-out workload
(small change)



NetCache **quickly and effectively reacts** to a wide range of workload dynamics.

Conclusion

- **NetCache** is a new key-value store architecture that uses **in-network caching** to balance in-memory key-value stores.

- NetCache exploits programmable switches to efficiently **detect, index, cache and serve** hot items **in the data plane**

- NetCache provides high performance even under **highly-skewed and rapidly-changing** workloads

Discussion