Programmable Networks
Lecture 2 – P4 basics & lookups

Sándor Laki, PhD

Communication Networks Laboratory

Dept. of Information Systems, Faculty of Informatics

ELTE Eötvös Loránd University

lakis@elte.hu

http://lakis.web.elte.hu

Slides were inspired by (and are based on) related courses of Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich), Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

this week

- P4 environment
  - needed for programming in P4
- P4 language
  - Language constructs
- Fast lookup
  - LPM lookup in software and hardware, packet classification

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

P4 environment

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

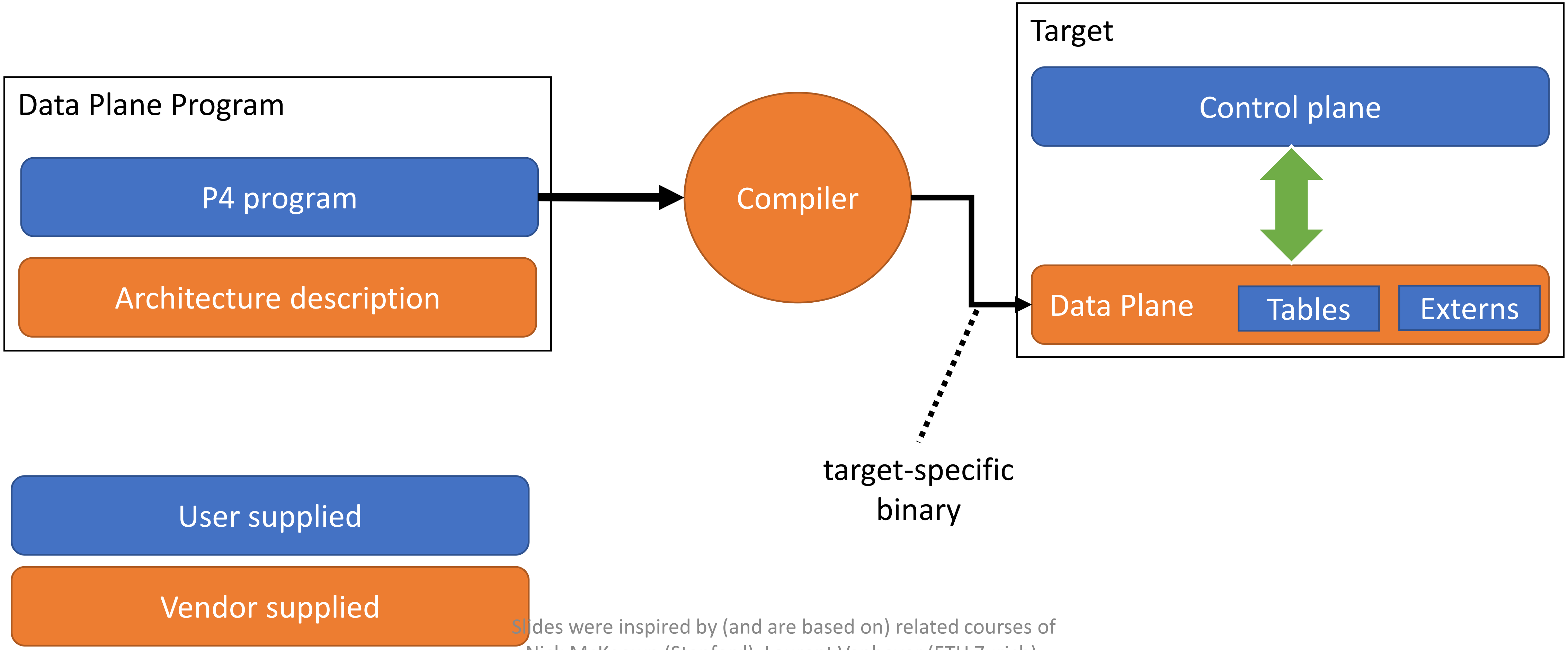
P4 history

- May 2013: Initial idea and the name “P4”
- July 2014: First paper (SIGCOMM CCR)
- Aug 2014: First P4-14 Draft Specification (v0.9.8)
- Sep 2014: P4-14 Specification released (v1.0.0)
- Jan 2015: P4-14 v1.0.1
- Mar 2015: P4-14 v1.0.2
- Nov 2016: P4-14 v1.0.3
- May 2017: P4-14 v1.0.4
- Apr 2016: P4-16 – first commits
- Dec 2016: First P4-16 Draft Specification
- **May 2017: P4-16 Specification released**

P4-16 introduces the concept of architecure

- P4 Target:
  - A model of a specific hardware implementation
  - The hardware backend running the compiled P4 code
- P4 Architecture:
  - An API to program a target
  - P4 programmable components, externs, fixed components

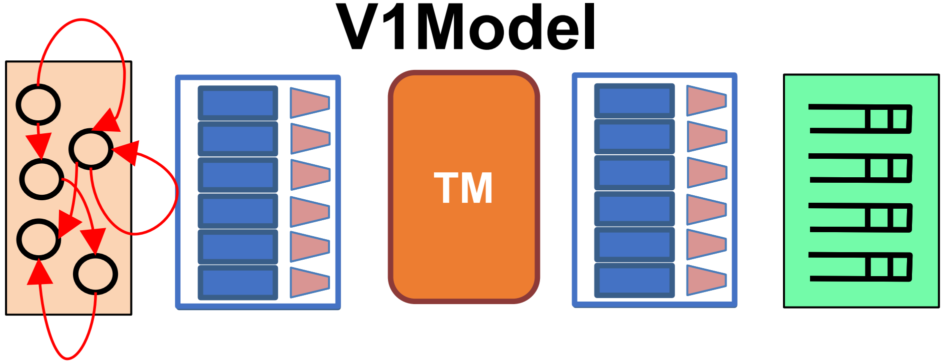
recap: how to program a P4 target

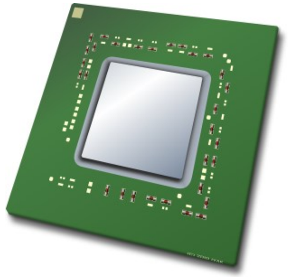


Slides were inspired by (and are based on) related courses of Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich), Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

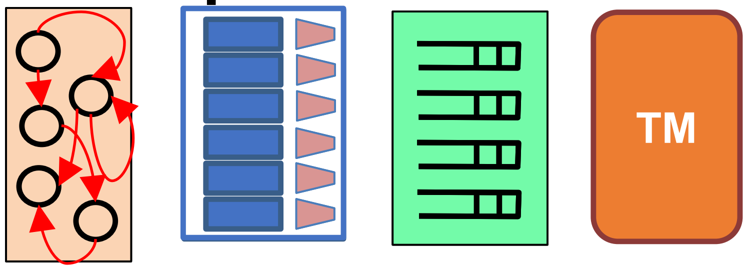
Example Architectures and Targets

V1Model

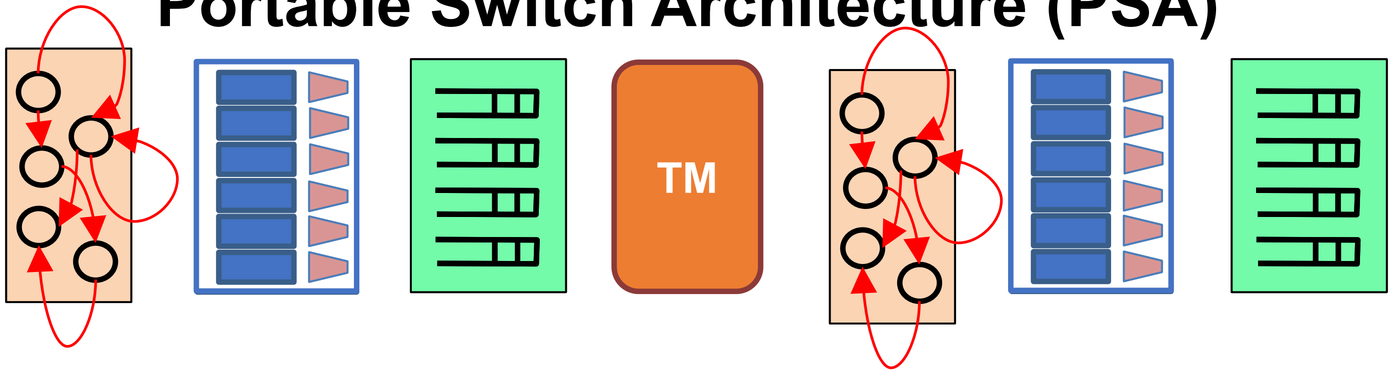




SimpleSumeSwitch

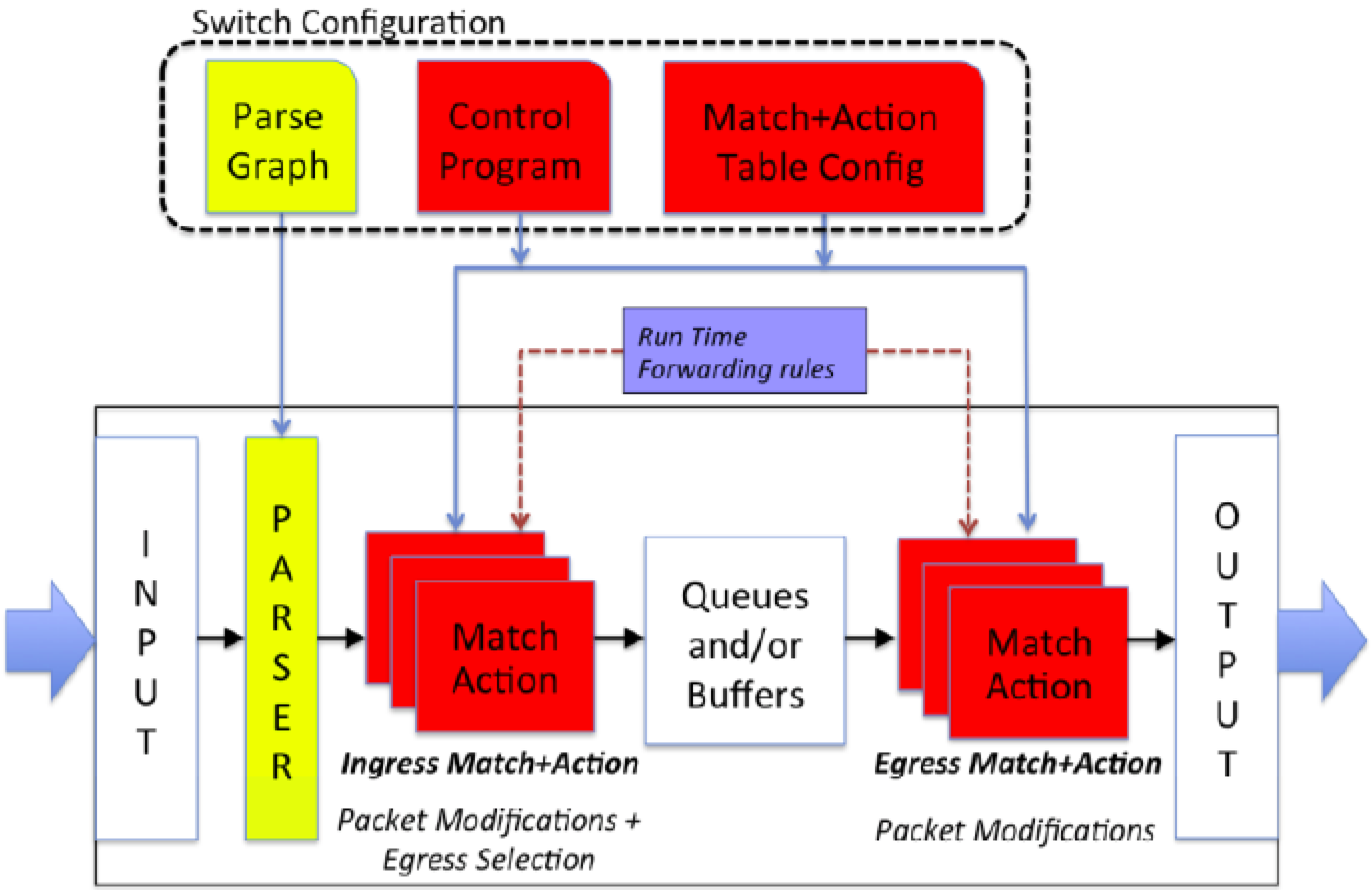


Portable Switch Architecture (PSA)



Anything

we’ll rely on the simple „v1model”



https://p4.org/p4-spec/p4-14/v1.0.4/tex/p4.pdf

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

metadata

Each architecture defines the **metadata it supports**, including both **standard** and **intrinsic** ones

Intrinsic metadata: in addition to the standard metadata fields to offer more advanced features.

```
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;
    error parser_error;
    bit<48> ingress_global_timestamp;
    bit<48> egress_global_timestamp;
    bit<32> lf_field_list;
    bit<16> mcast_grp;
    bit<32> resubmit_flag;
    bit<16> egress_rid;
    bit<1> checksum_error;
    bit<32> recirculate_flag;
}
```

externs

Black-box functions implemented by the target whose interface is known:

- Most targets contain specialized components
  which cannot be expressed in P4 (e.g. complex computations)

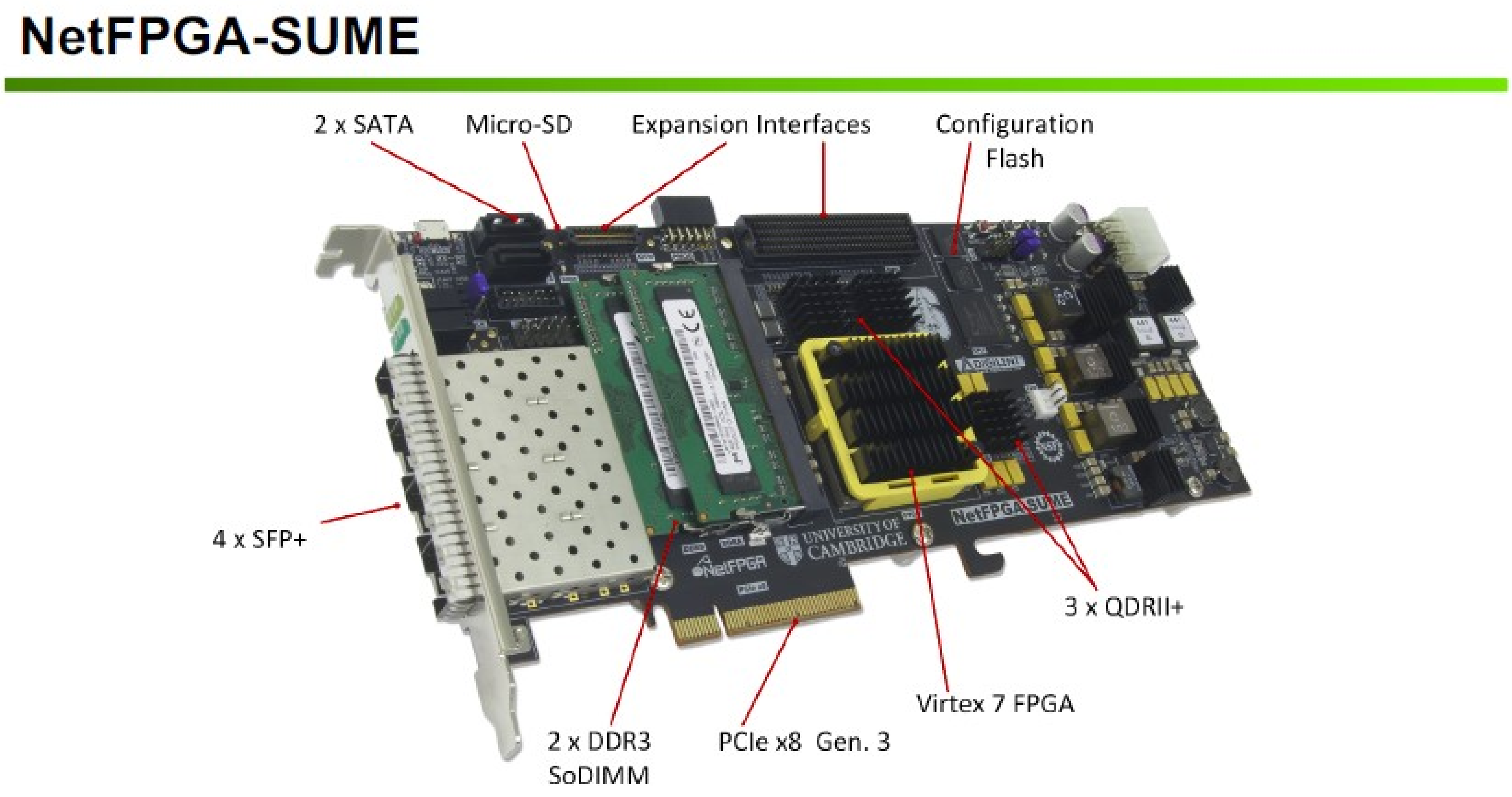
- but P4-16 should be target-independent
  in contrast to P4-14

- Externs are similar to Java interfaces
  only the signature is known, not the implementation

extern examples – v1model

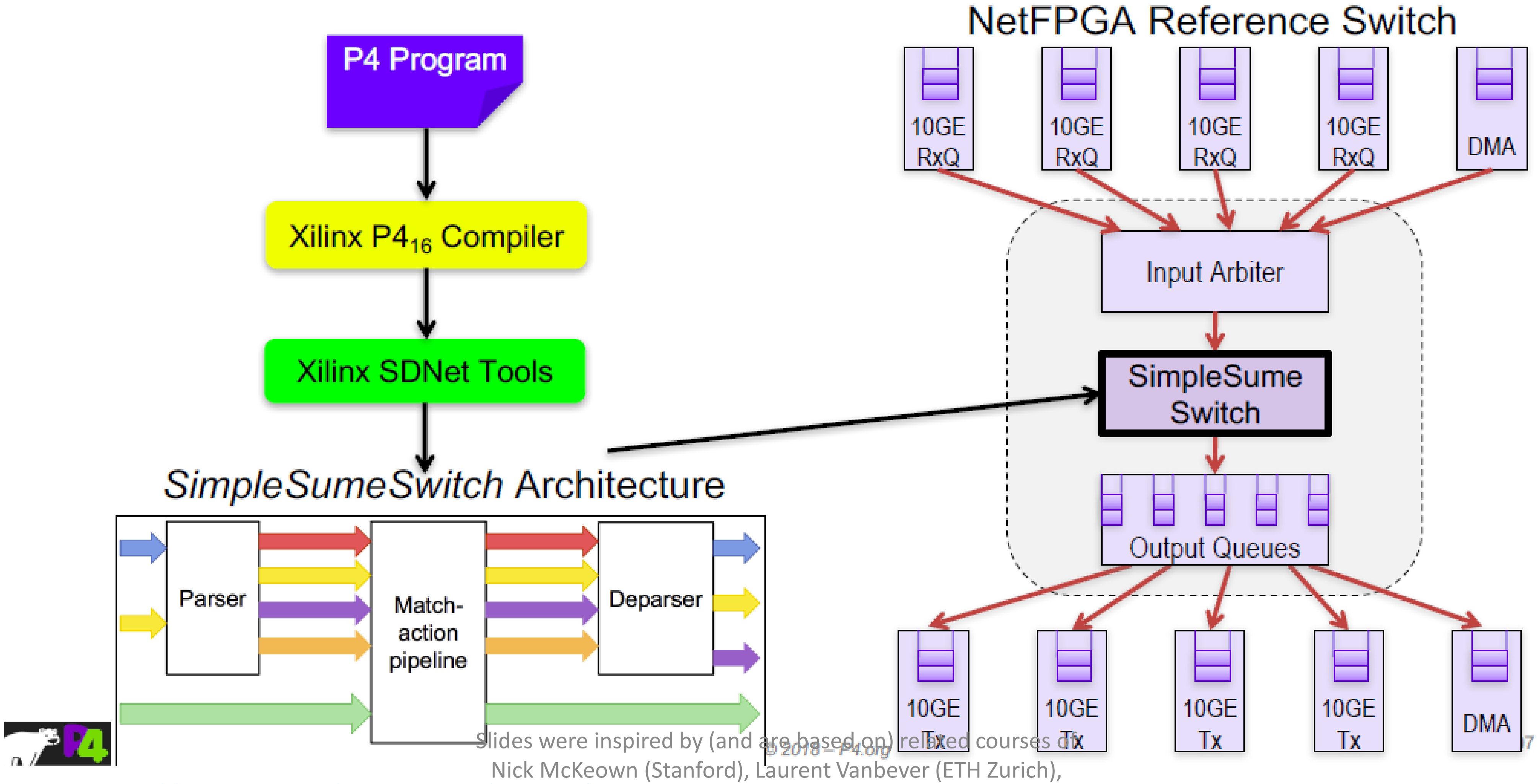
```
extern register<T> {
    register(bit<32> size);
    void read(out T result, in bit<32> index);
    void write(in bit<32> index, in T value);
}

extern void random<T>(out T result, in T lo, in T hi);

extern void hash<O, T, D, M>(out O result, in HashAlgorithm algo, in T base, in D data, in M max);

extern void update_checksum<T, O>(in bool condition, in T data, inout O checksum, HashAlgorithm algo);
```

For more visit: https://github.com/p4lang/p4c/blob/master/p4include/v1model.p4 architectures may have different metadata and externs



Source: http://isfpga.org/fpga2018/slides/FPGA-2018-P4-tutorial.pdf

P4→NetFPGA Compilation Overview



Source: http://isfpga.org/fpga2018/slides/FPGA-2018-P4-tutorial.pdf

Standard Metadata in SimpleSumeSwitch Architecture

```
/* standard sume switch metadata */
struct sume_metadata_t {
    bit<16> dma_q_size;
    bit<16> nf3_q_size;
    bit<16> nf2_q_size;
    bit<16> nf1_q_size;
    bit<16> nf0_q_size;
    bit<8> send_dig_to_cpu; // send digest_data to CPU
    bit<8> dst_port; // one-hot encoded
    bit<8> src_port; // one-hot encoded
    bit<16> pkt_len; // unsigned int
}
```

- `*_q_size` – size of each output queue, measured in terms of 32-byte words, when packet starts being processed by the P4 program
- `src_port/dst_port` – one-hot encoded, easy to do multicast
- `user_metadata/digest_data` – structs defined by the user

Source: http://isfpga.org/fpga2018/slides/FPGA-2018-P4-tutorial.pdf

P4→NetFPGA Extern Function library

- **Implement platform specific functions**
  - Black box to P4 program
- **Implemented in HDL**
- **Stateless – reinitialized for each packet**
- **Stateful – keep state between packets**
- **Xilinx Annotations**
  - `@Xilinx_MaxLatency()` – maximum number of clock cycles an extern function needs to complete
  - `@Xilinx_ControlWidth()` – size in bits of the address space to allocate to an extern function

Source: http://isfpga.org/fpga2018/slides/FPGA-2018-P4-tutorial.pdf

P4 language

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

```
#include <core.p4>
#include <v1model.p4>
const bit<16> TYPE_IPV4 = 0x800;
typedef bit<32> ip4Addr_t;
header ipv4_t {…}
struct headers {…}
```

Include libraries

Declarations

```
parser MyParser(…) {
        state start {…}
        state parse_ethernet {…}
        state parse_ipv4 {…}
}
```

Parse packet headers

```
control MyIngress(…) {

        action ipv4_forward(…) {…}

        table ipv4_lpm {…}

        apply {
                if (…) {…}
        }
}
```

Control flow to modify/forward packets

```
control MyDeparser(…) {…}
```

Assemble (modified) packet

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

main()

P4-16 is a statically-typed language with base types

bool	Boolean value
bit<W>	Bit-string of width W
int<W>	Signed integer of width W
varbit<W>	Bit-string of dynamic length ≤W
match_kind	describes ways to match table keys
error	used to signal errors
void	no values, used in few restricted circumstances
~~float~~	not supported
~~string~~	not supported

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

… and operators to derive composed ones

- Header
- Header stack
- Header union

- Struct

- Tuple

- Enum
- etc.

header

```
header Ethernet_h {
    bit<48> dstAddr;
    bit<48> srcAddr;
    bit<16> etherType;
}
```

Parsing a packet using **extract**() fills in the fields of the header from a network packet.

A successful **extract**() sets to true the **validity** bit of the extracted header

Operations on header instances in the control blocks: **isValid**(), **setValid**() and **setInvalid**()

```
Ethernet_h ethernetHeader;
```

Declaration

Similar to struct in C containing the different fields plus a hidden "validity" field

header

```
header Ethernet_h {
    bit<48> dstAddr;
    bit<48> srcAddr;
    bit<16> etherType;
}
```

```
header Mpls_h {
    bit<20> label;
    bit<3> tc;
    bit<1> bos;
    bit<8> ttl;
}

Mpls_h[10] mpls;
```

Array of up to 10 MPLS headers

```
header_union IP_h {
    IPv4_h v4;
    IPv6_h v6;
}
```

Either IPv4 or IPv6 header
(only one alternative)

struct & tuple

```
struct standard_metadata_t {
    bit<9> ingress_port;
    bit<9> egress_spec;
    bit<9> egress_port;
    …
}
```

Unordered collection of named members

```
tuple<bit<32>, bool> x;
x = { 10, false };
```

Unordered collection of unnamed members

others

enum Priority {High, Low};

typedef bit<48> macAddr_t;

extern …

parser …

control …

package …

operations similar to C

- arithemtic operations +, -, *
- bitwise operations ~, &, |, ^, >>, <<
- non-standard bit operations
  [a:b] bit-slicing
  ++ bit-string concatenation
- No division and modulo ~~/, %~~
  - Division by constant is possible for integers

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

constants and variables

```
bit<8> v = 42;

typedef bit<32> MyType;
MyType v2;
v2 = 42;

const bit<8> c = 42;
const MyType c2 = 8899;
```

important

variables cannot be used to maintain state between different network packets

To maintain states:

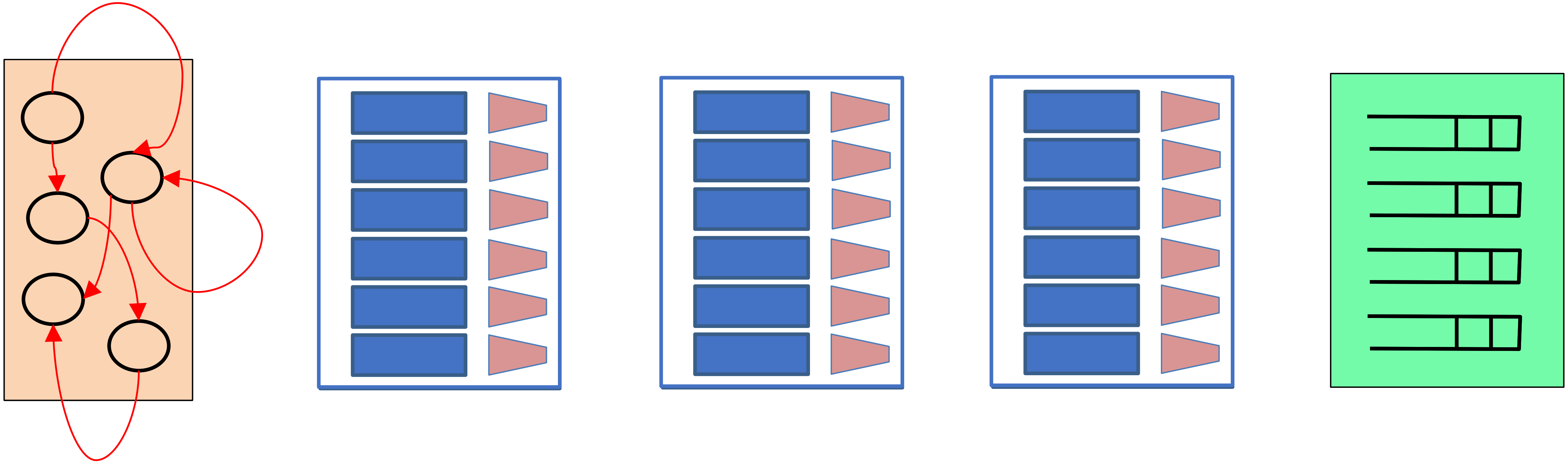
- **tables** that can be modified by the control plane
- **extern objects** like registers that can be modified by both control and data plane

statements

return	terminates the execution of the action or control containing it
exit	terminates the execution of all the blocks currently executing
conditions	if (v==42) { … } else { … } cannot be used in parsers

switch
onyl in control b.
	switch (t.apply().action_run) {
	action_1 : { … }
	action_2 : { … }
	}

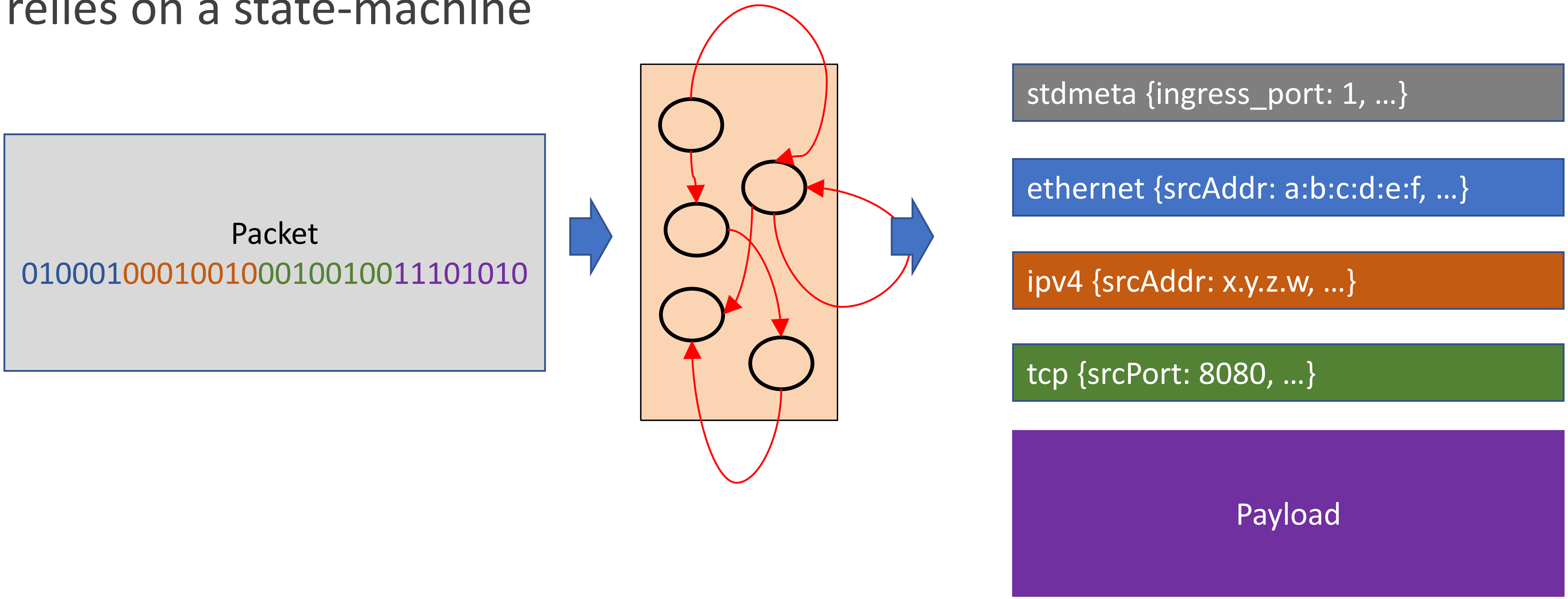
parsing + match-actions + deparsing



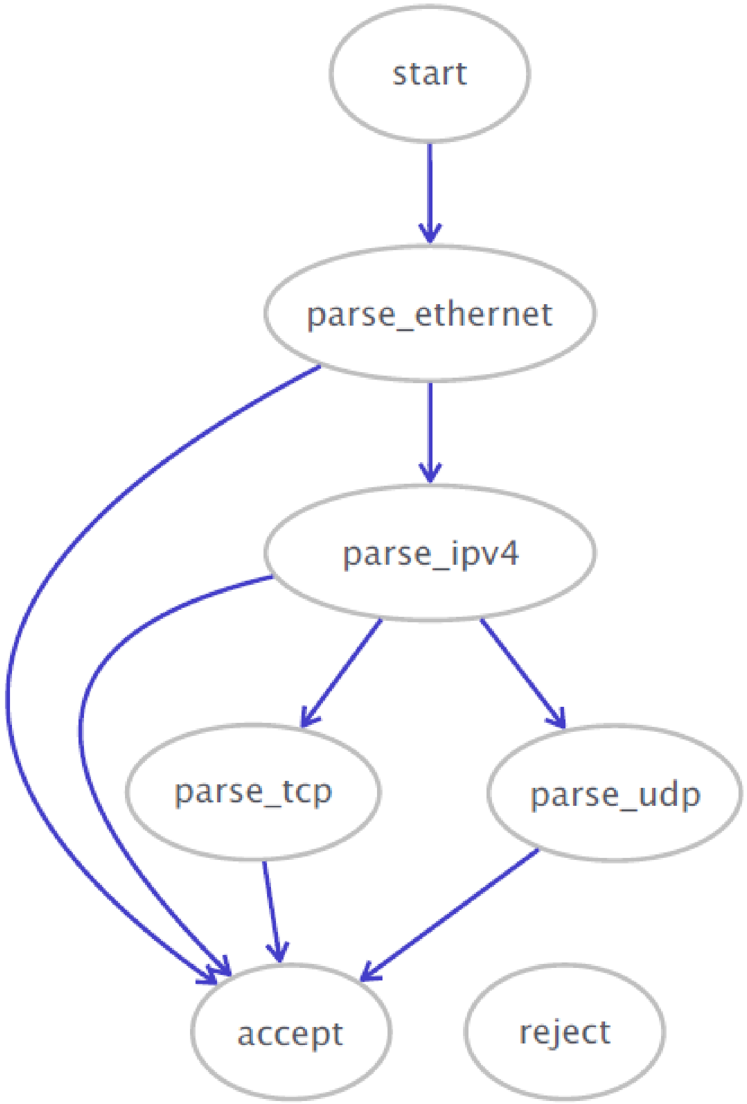
Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

parser

relies on a state-machine



Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).




```
parser MyParser(…) {

 state start {
  transition parse_ethernet;
 }

 state parse_ethernet {
  packet.extract(hdr.ethernet);
  transition select(hdr.ethernet.etherType) {
    0x800: parse_ipv4;
    default: accept;
  }
 }

 state parse_ipv4 {
  packet.extract(hdr.ipv4);
  transition select(hdr.ipv4.protocol) {
   6: parse_tcp;
   17: parse_udp;
   default: accept;
  }
 }

 state parse_tcp {
  packet.extract(hdr.tcp);
   transition accept;
 }

 state parse_udp {
  packet.extract(hdr.udp);
   transition accept;
 }
}
```

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

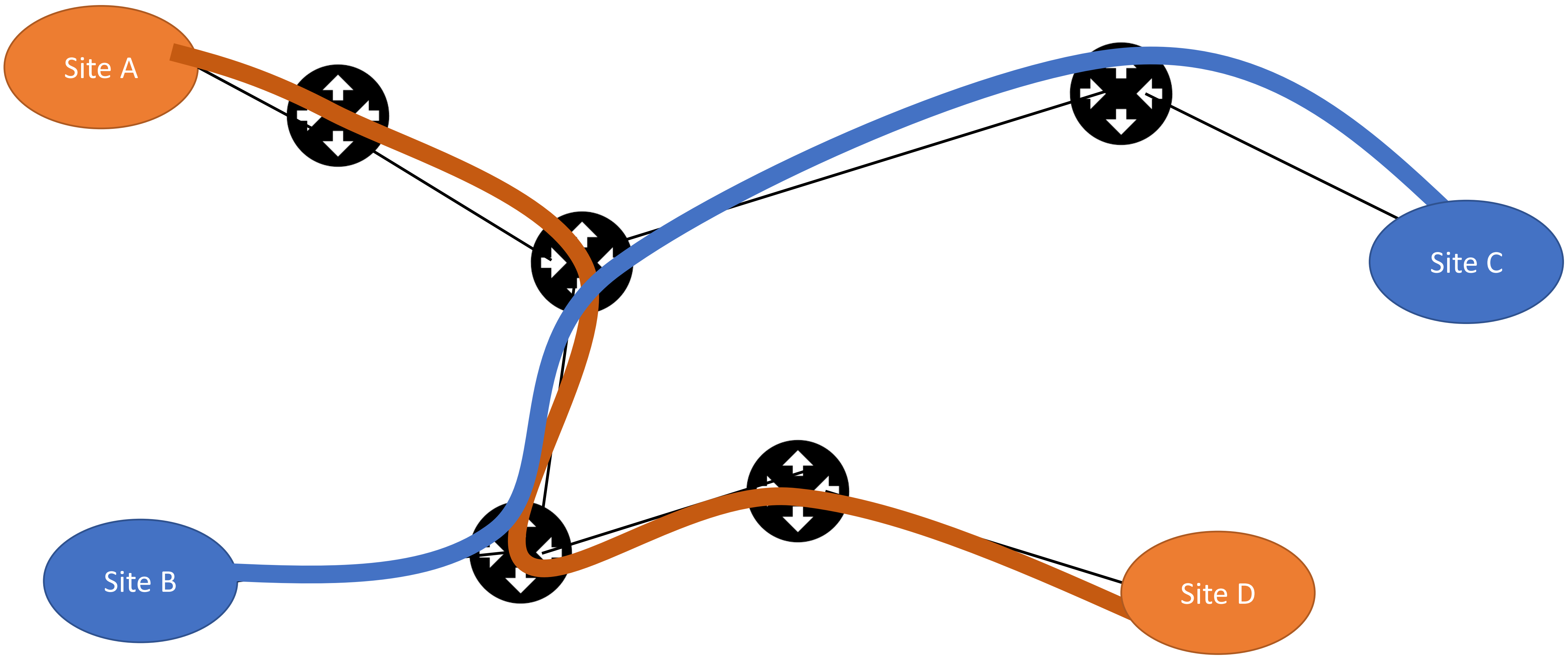
the parser is a state machine

```
state start {
    transition parse_ethernet;
}

state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
        0x800: parse_ipv4;
        default: accept;
    }
}
```

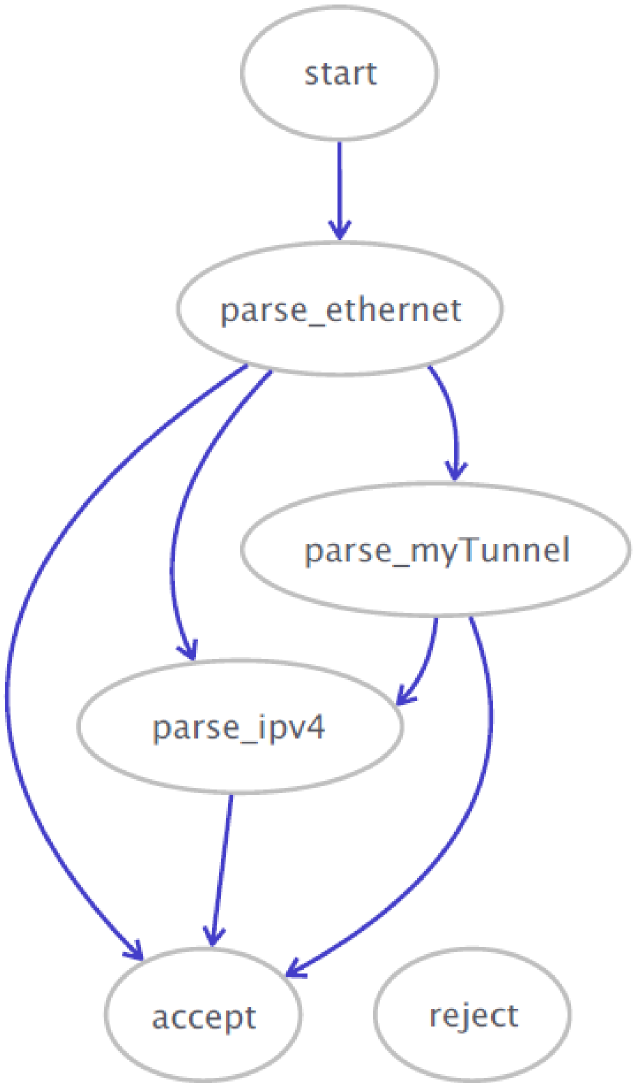
Next state depends on etherType

implement your own protocol

Simple tunneling



Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).




```
header myTunnel_t {
    bit<16> proto_id;
    bit<16> dst_id;
}

struct headers {
    ethernet_t   ethernet;
    myTunnel_t   myTunnel;
    ipv4_t       ipv4;
}

parser MyParser(…) {

 state start {…}

 state parse_ethernet {
  packet.extract(hdr.ethernet);
  transition select(hdr.ethernet.etherType) {
   0x1212: parse_myTunnel;
   0x800: parse_ipv4;
   default: accept;
  }
 }

 state parse_myTunnel {
  packet.extract(hdr.myTunnel);
  transition select(hdr.myTunnel.proto_id) {
   TYPE_IPV4: parse_ipv4;
   default: accept;
  }
 }

 state parse_ipv4 {…}
}
```

fixed vs variable length packet fields

```
header IPv4_no_options_h {
        ...
        bit<32> srcAddr;
        bit<32> dstAddr;
}
header IPv4_options_h {
        varbit<320> options;
}
...
parser MyParser(...) {
        ...
        state parse_ipv4 {
                packet.extract(headers.ipv4);
                transition select (headers.ipv4.ihl) {
                        5: dispatch_on_protocol;
                        default: parse_ipv4_options;
                }
        }
        state parse_ipv4_options {
                packet.extract(headers.ipv4options, (headers.ipv4.ihl - 5) << 2 );
                transition dispatch_on_protocol;
        }
}
```

Variable width field (only one field in a header)

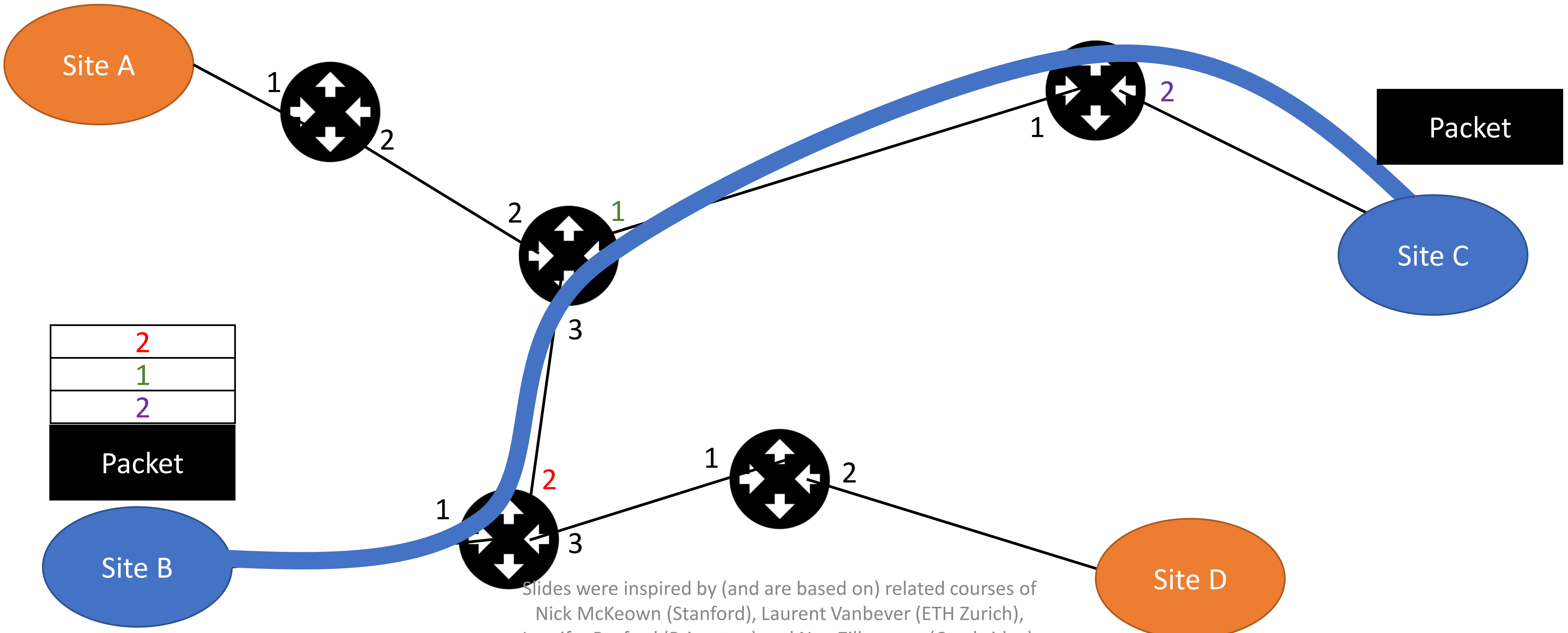
ihl determines the length of field options

note: ihl is the number of words (bit<32>) in the IP packet

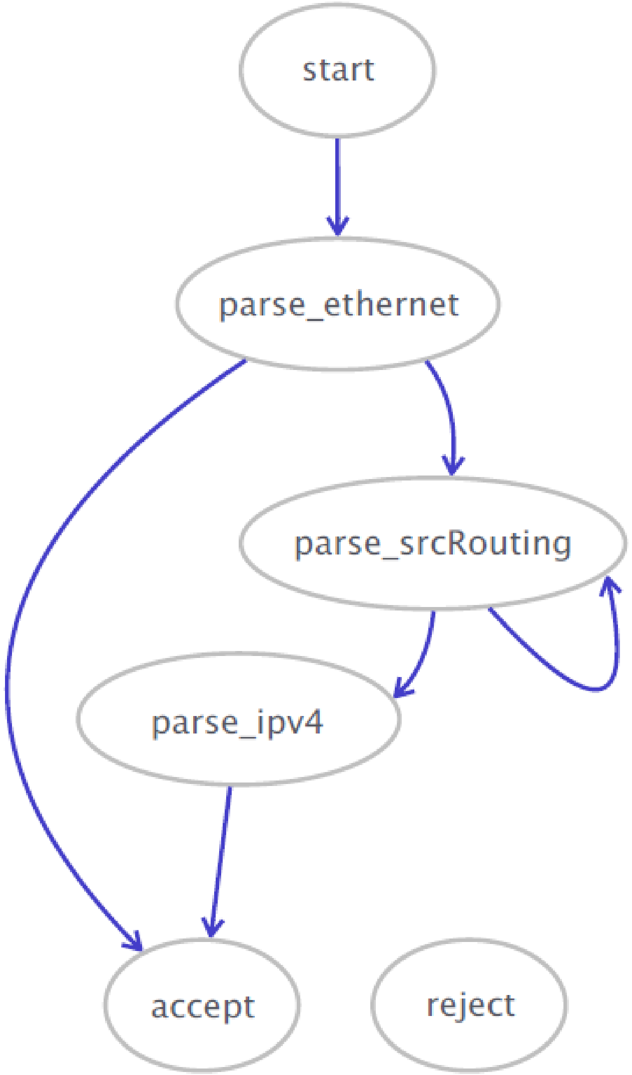
Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

parsing a header stack

requires loops – the only case when loop is possible in P4



Slides were inspired by (and are based on) related courses of Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich), Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).




```
header srcRoute_t {
 bit<1>     bos;
 bit<15>    port;
}

struct headers {
 ethernet_t                ethernet;
 srcRoute_t[MAX_HOPS]      srcRoutes;
 ipv4_t                    ipv4;
}

parser MyParser(...) {
 state parse_ethernet {
  packet.extract(hdr.ethernet);
  transition select(hdr.ethernet.etherType) {
   TYPE_SRCROUTING: parse_srcRouting;
   default: accept;
  }
 }

 state parse_srcRouting {
  packet.extract(hdr.srcRoutes.next);
  transition select(hdr.srcRoutes.last.bos) {
   1: parse_ipv4;
   default: parse_srcRouting;
  }
 }
}
```

more advanced parser constructions

extern void verify(in bool condition, in error err);

a form of error handling

hdr.lookahead<T>();

access bits that have not been parsed yet

value_set<T>(size) pvs;

Sub-parsers like subrutines

```
parser callee(packet_in packet, out IPv4 ipv4) { ...}
parser caller(packet_in packet, out Headers h) {
    callee() subparser;  // instance of callee
    state subroutine {
        subparser.apply(packet, h.ipv4);  // invoke sub-parser
        transition accept;  // accept if sub-parser ends in accept state
    }
}
```

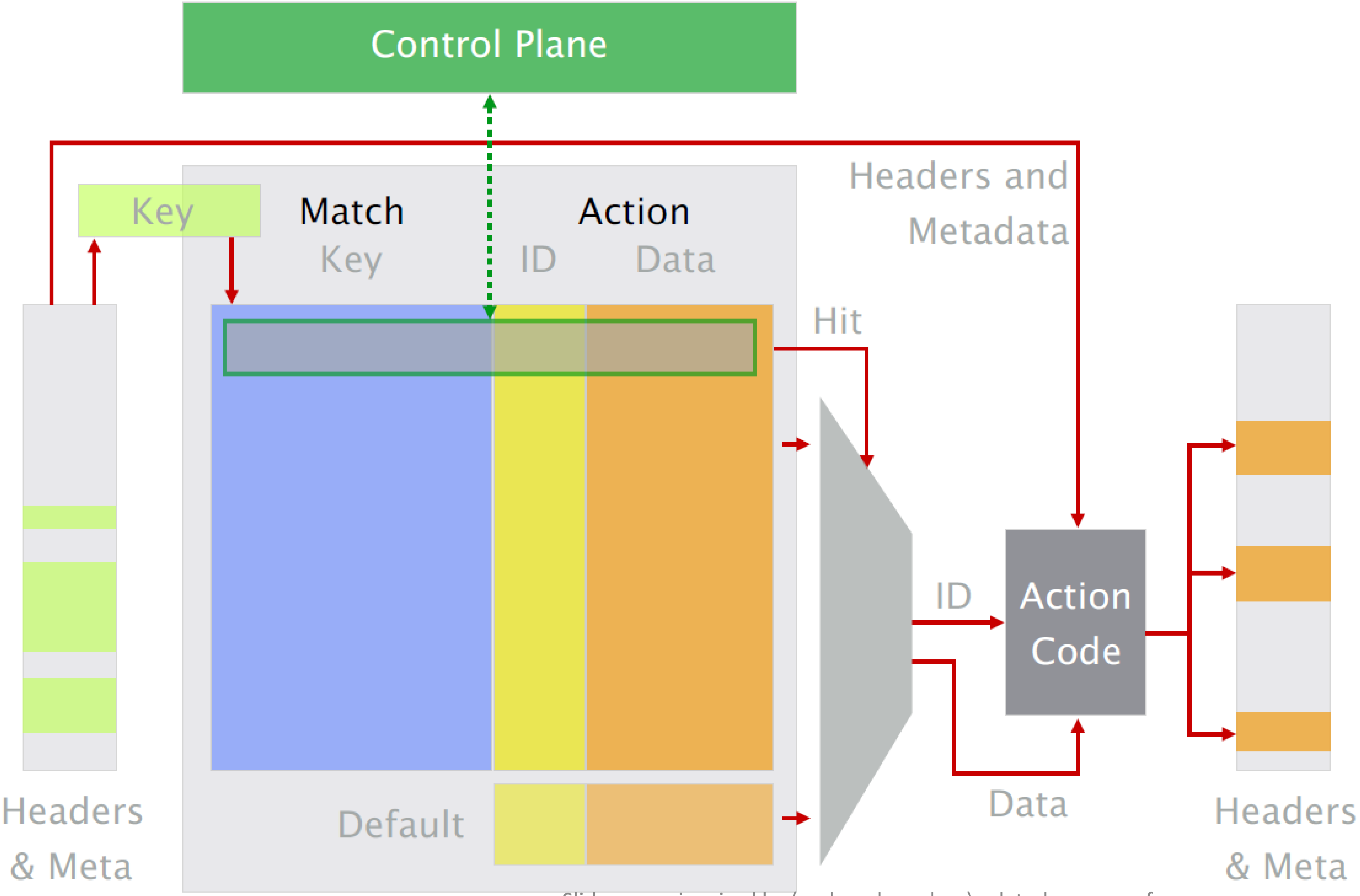
```
ParserModel.verify(bool condition, error err)
{
    if (condition == false) {
        ParserModel.parserError = err;
        goto reject;
    }
}
```

```
state start {
    transition select(hdr.lookahead<bit<8>>()) {
        0: parse_tcp_option_end;
        1: parse_tcp_option_nop;
        2: parse_tcp_option_ss;
        3: parse_tcp_option_s;
        5: parse_tcp_option_sack;
    }
}
```

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

control blocks

tables	match a key and return an action
actions	similar to functions in C
control flow	similar to C but without loops


Control Plane
Key
Match
Key
Action
ID
Data
Headers and
Metadata
Hit
ID
Action
Code
Data
Default
Headers
& Meta
Headers
& Meta

tables

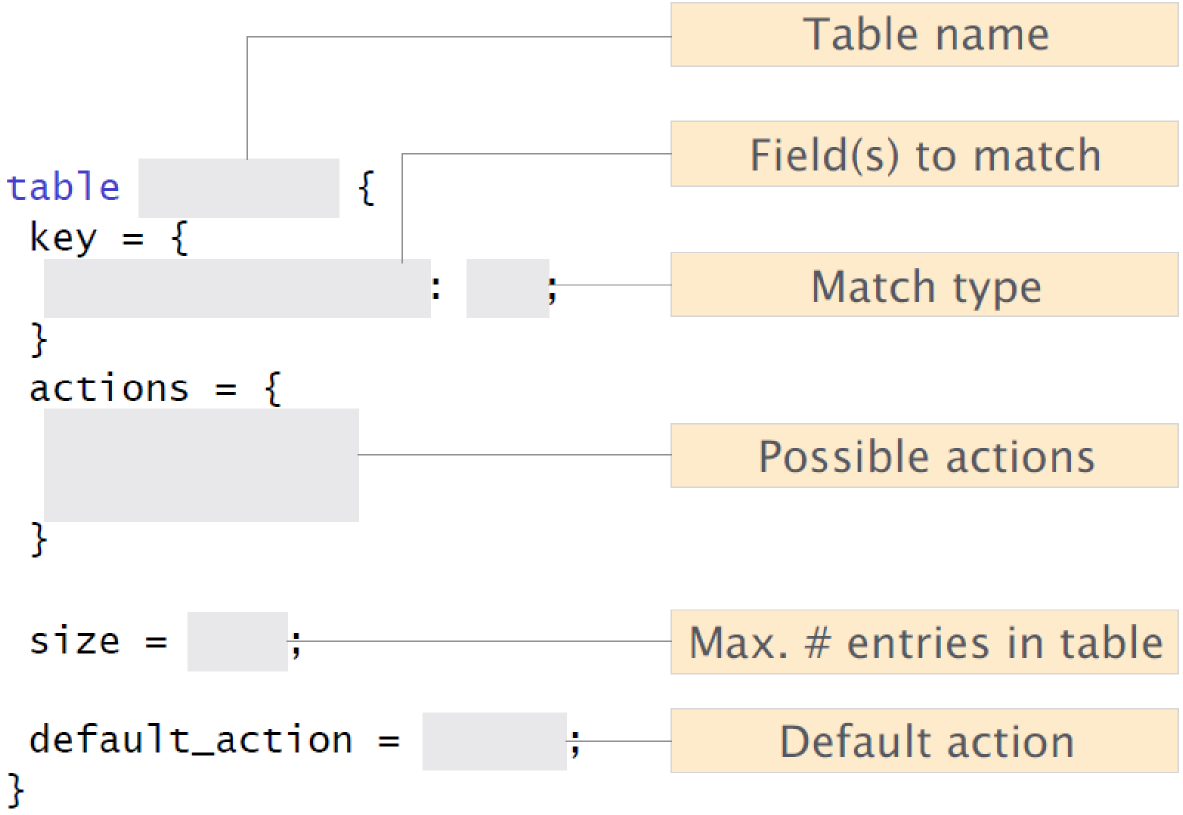


table {
key = {
: ;
}
actions = {
}
size = ;
default_action = ;
}
Table name
Field(s) to match
Match type
Possible actions
Max. # entries in table
Default action

tables

```
table ipv4_lpm {
 key = {
  hdr.ipv4.dstAddr: lpm;
 }
 actions = {
  ipv4_forward;
  drop;
 }

 size = 1024;

 default_action = drop();
}
```

- Table name
- Destination IP address
- Longest prefix match
- Possible actions
- Max. # entries in table
- Default action

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

match on one or **multiple** keys in different ways

```
table Fwd {
    key = {
        hdr.ipv4.dstAddr : ternary;
        hdr.ipv4.version : exact;
    }
    ...
}
```

Fields to match

Match kind

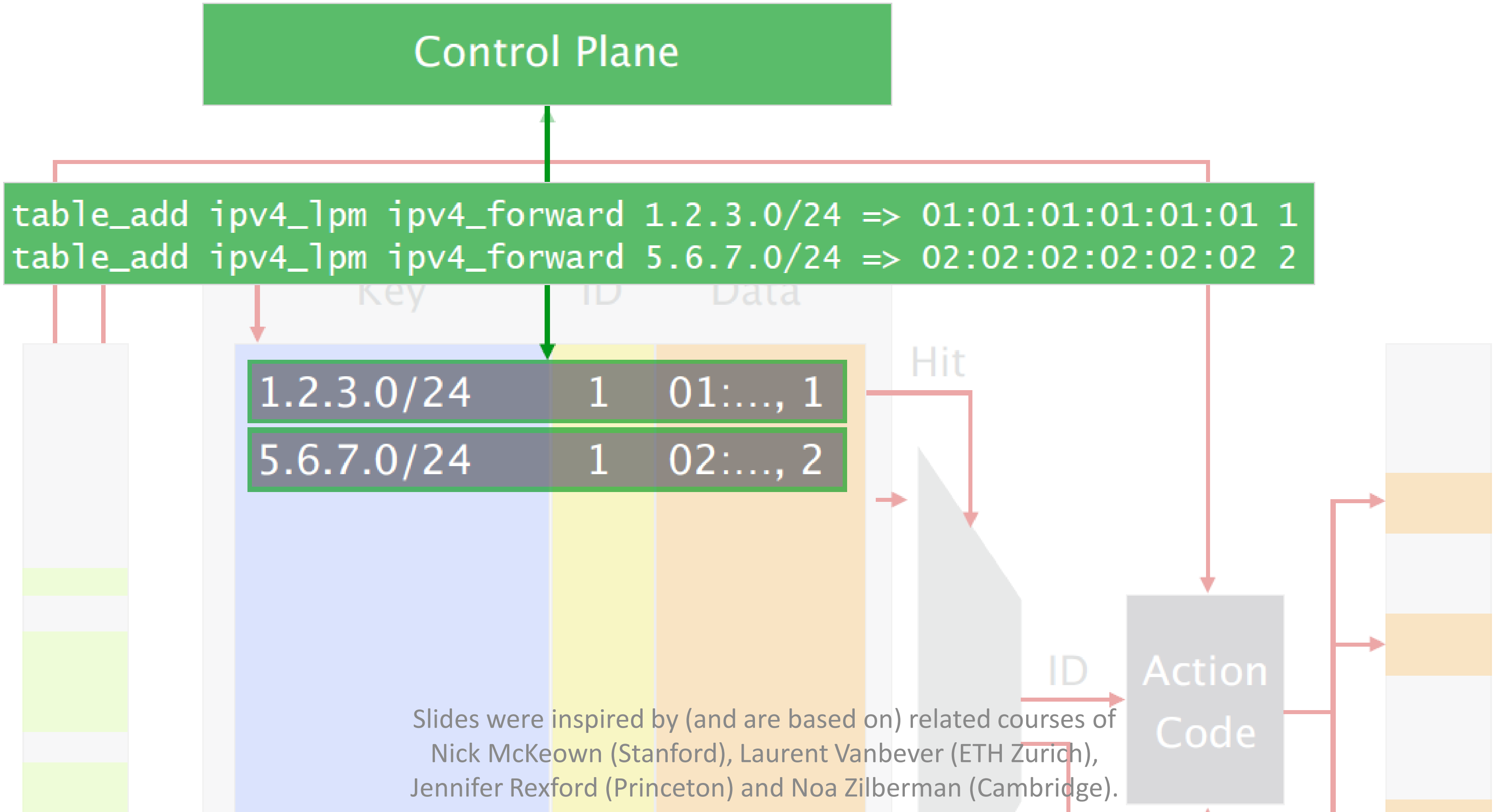
Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

match kinds are specified in P4-core and in the archs

Match kind	Description	Example	Specified in
`exact`	exact comparison	`0x01020304`	core.p4
`ternary`	compare with mask	`0x01020304 & 0x0F0F0F0F`	core.p4
`lpm`	longest prefix match	`0x01020304/24`	core.p4
`range`	check if in range	`0x01020304 – 0x010203FF`	v1model.p4
`...`			v1model.p4
`...`			other architecture

Slides were inspired by (and are based on) related courses of Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich), Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

Table entries are added through the control plane in runtime



Slides were inspired by (and are based on) related courses of Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich), Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

actions

Block of statements that can modify the packet

Usually take directional parameters:

in read only inside the action
like parameters to a function

out uninitialized, write inside the action
like return values

inout combination of in and out
like “call by reference”

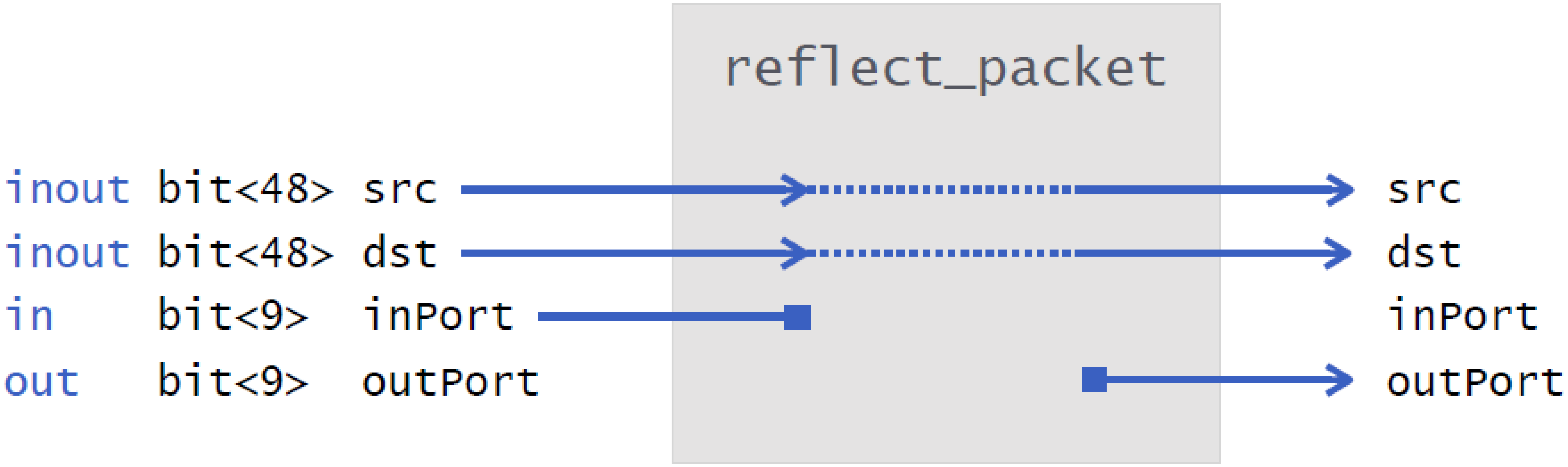
Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

example

```
action reflect_packet( inout bit<48> src,
                       inout bit<48> dst,
                       in bit<9> inPort,
                       out bit<9> outPort
                     ) {
 bit<48> tmp = src;
 src = dst;
 dst = tmp;
 outPort = inPort;
}
```

Parameter with direction

```
reflect_packet( hdr.ethernet.srcAddr,
                hdr.ethernet.dstAddr,
                standard_metadata.ingress_port,
                standard_metadata.egress_spec
              );
```



Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

actions for table lookups

Parameter
without direction

```
action set_egress_port(bit<9> port) {
 standard_metadata.egress_spec = port;
}
```

control flow

Interacting with tables from the control flow

Applying a table	`ipv4_lpm.apply();`
Checking if there was a hit	`if (ipv4_lpm.apply().hit) {...}` `else {...}`

Check which action was executed	`switch (ipv4_lpm.apply().action_run) {`
	`    ipv4_forward: { ... }`
	`}`

l3fwd with multiple tables

IP Packet →

ipv4_lpm	
1.1.1.0	1
2.2.2.0	1
3.3.3.0	2
4.4.4.0	3

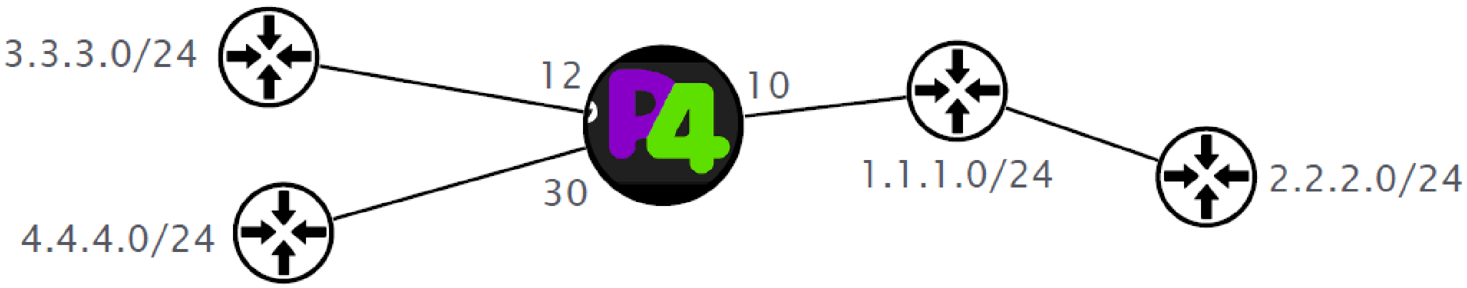
Map a prefix to a next hop index

→

forward	
1	10
2	12
3	30

Map a next hop index to an egress port

→



Slides were inspired by (and are based on) related courses of Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich), Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

l3fwd with multiple tables

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

```
table forward {
 key = {
  meta.nhop_index: exact;
 }
 actions = {
  _forward;
  NoAction;
 }

 size = 64;
 default_action = NoAction();
}
```

control flow – applying tables in a seq.

```
control MyIngress(...) {
 action drop() {...}
 action set_nhop_index(...}
 action _forward(...}
 table ipv4_lpm {...}
 table forward {...}

 apply {
  if (hdr.ipv4.isValid()){
   if (ipv4_lpm.apply().hit) {
    forward.apply();
   }
  }
 }
}
```

Check if IPv4 packet

Apply ipv4_lpm table and check if there was a hit

apply forward table

Slides were inspired by (and are based on) related courses of Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich), Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

checksum validation and recomputation

```
extern void verify_checksum<T, O>( in bool condition,
                                   in T data,
                                   inout O checksum,
                                   HashAlgorithm algo
                                 );

extern void update_checksum<T, O>( in bool condition,
                                   in T data,
                                   inout O checksum,
                                   HashAlgorithm algo
                                 );
```

v1model.p4

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

example - checksum recomputation

```
control MyComputeChecksum(...) {
 apply {
  update_checksum(
   hdr.ipv4.isValid(),
   { hdr.ipv4.version,
     hdr.ipv4.ihl,
     hdr.ipv4.diffserv,
     hdr.ipv4.totalLen,
     hdr.ipv4.identification,
     hdr.ipv4.flags,
     hdr.ipv4.fragOffset,
     hdr.ipv4.ttl,
     hdr.ipv4.protocol,
     hdr.ipv4.srcAddr,
     hdr.ipv4.dstAddr },
     hdr.ipv4.hdrChecksum,
   HashAlgorithm.csum16);
 }
}
```

pre-condition

fields list

checksum field

algorithm

More concepts

cloning packets	create a clone of a packet
sending packets to control plane	using dedicated Ethernet port, or target-specific mechanisms (e.g. digests)
recirculating	send packet through pipeline multiple times

Lookups & packet classification

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

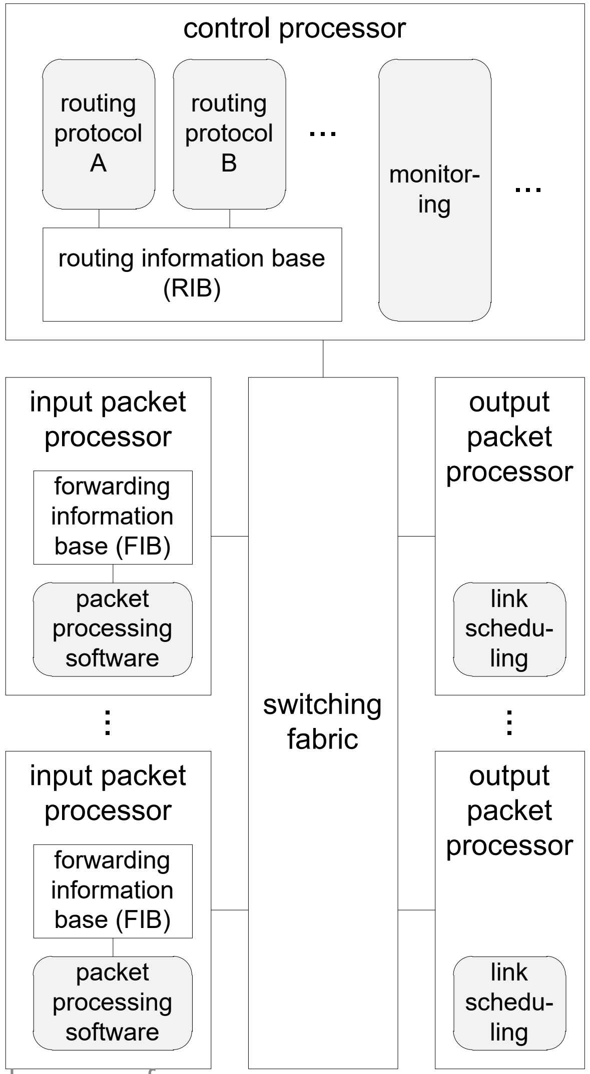
ECE 671 – Lecture 12

Routers

Prefix lookup

Prefix lookups for packet forwarding

- Match of IP destination address with prefixes specified in FIB
  - Longest matching prefix
- Typical core router
  - Hundreds of thousands of prefixes
  - Millions of lookups per second
- Efficient data structures and algorithms essential for lookup



LPF Thoughts

- Given N prefixes K_i of up to W bits, find the longest match with input K of W bits.
- 3 prefix notations: slash, mask, and wildcard. 192.255.255.255 /31 or 1*
- N =1M (ISPs) or as small as 5000 (Enterprise). W can be 32 (IPv4), 64 (multicast), 128 (IPv6).
- For IPv4, CIDR makes all prefix lengths from 8 to 28 common, density at 16 and 24

Example prefixes

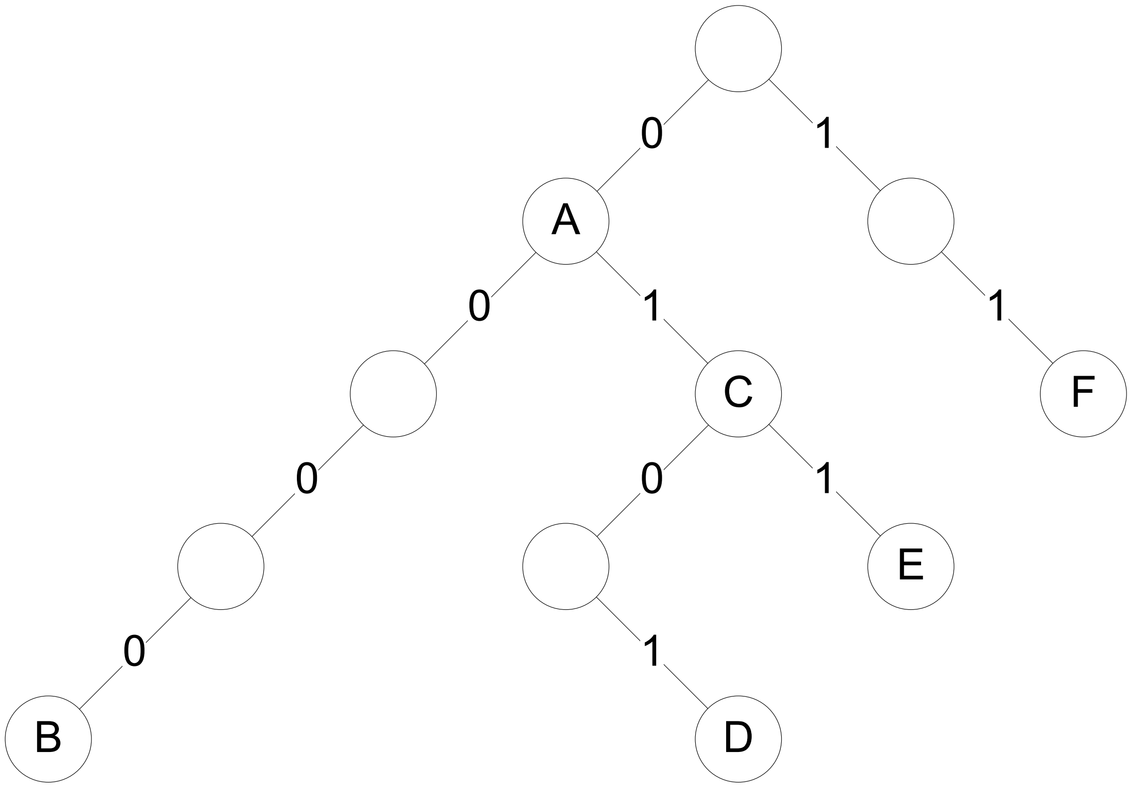
- Prefixes used for example data structures

Prefix name	Binary notation
A	0/1
B	0000/4
C	01/2
D	0101/4
E	011/3
F	11/2

- How to find match for an address (e.g., 01001111)?

Binary tree

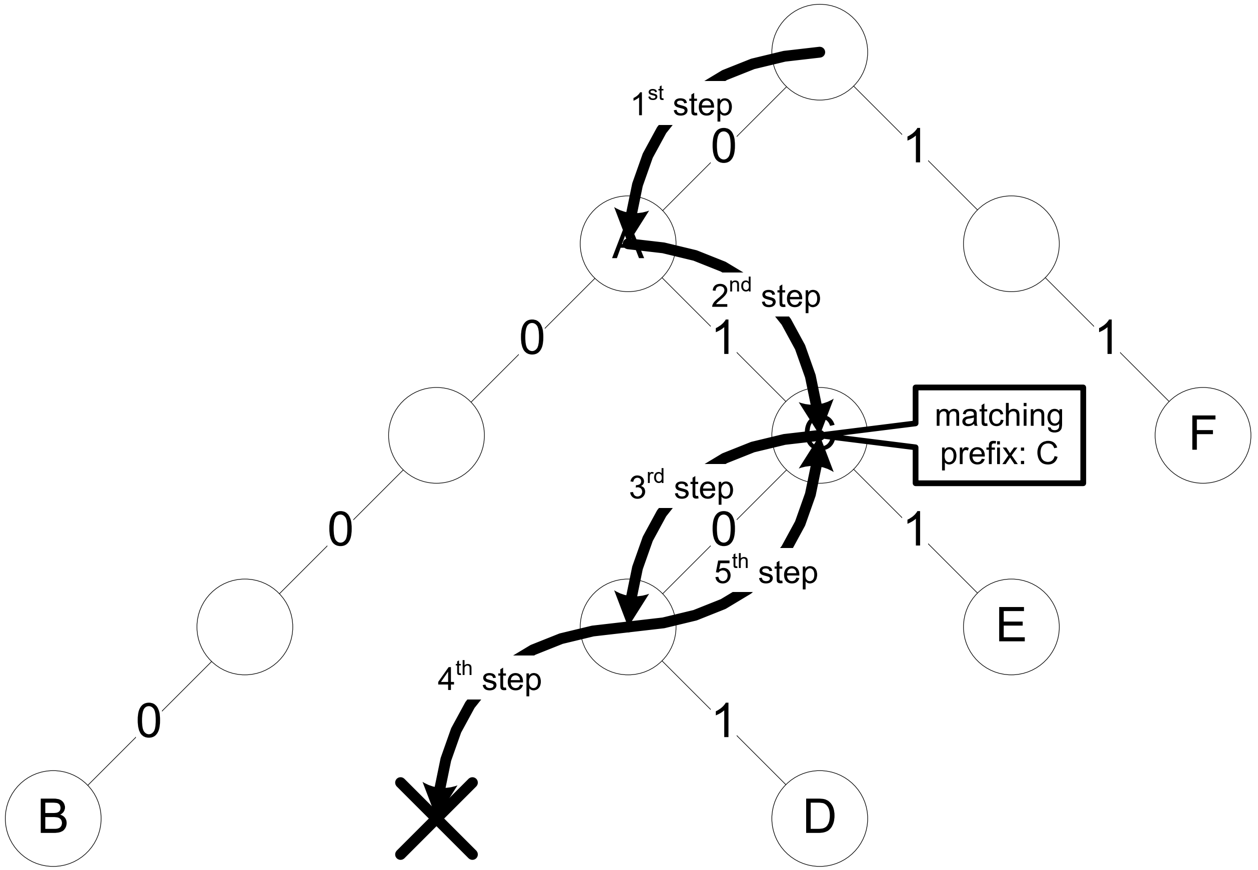
- One bit per level



- How to do lookup?

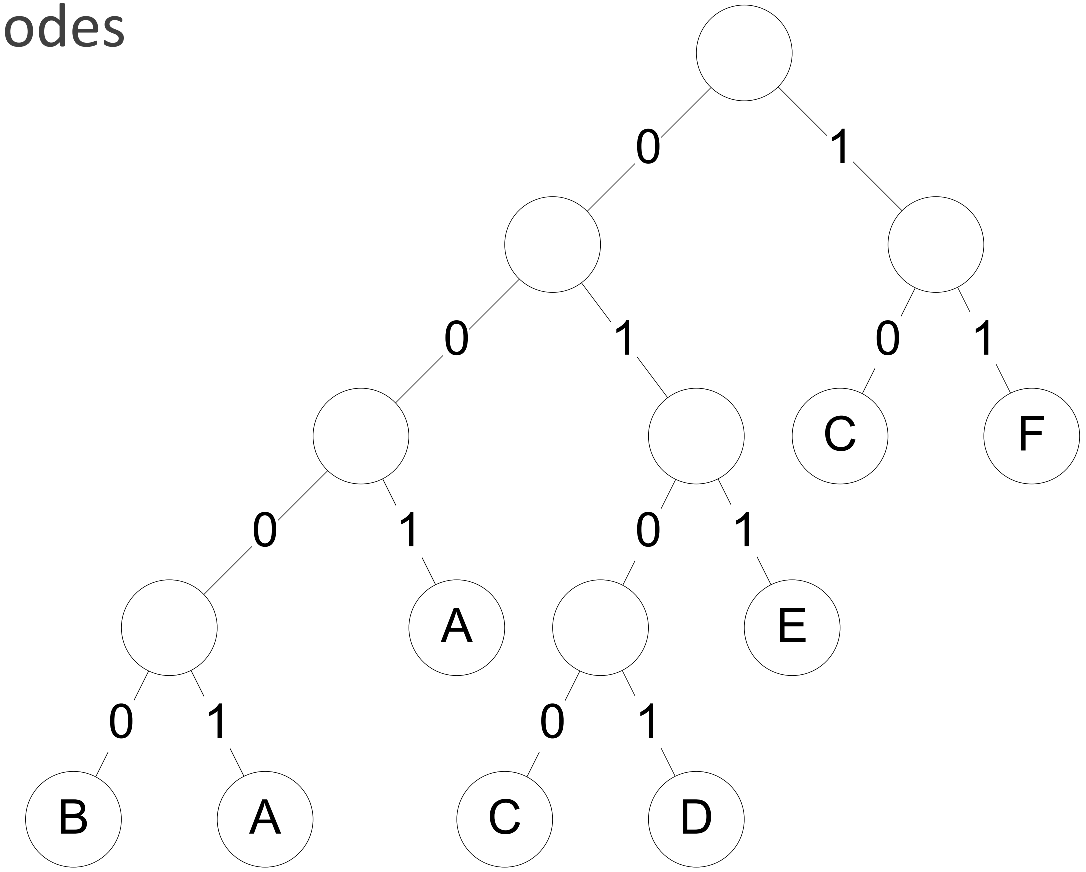
Binary tree

- Lookup may require backtracking (or memory):



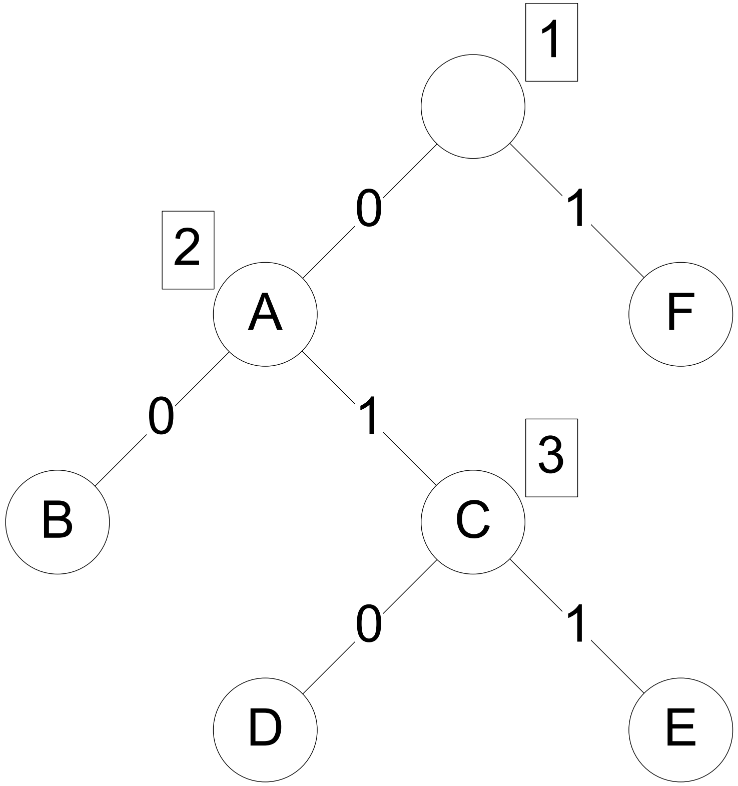
Leaf pushing

- Disjoint prefix binary tree
  - All matches in leaf nodes



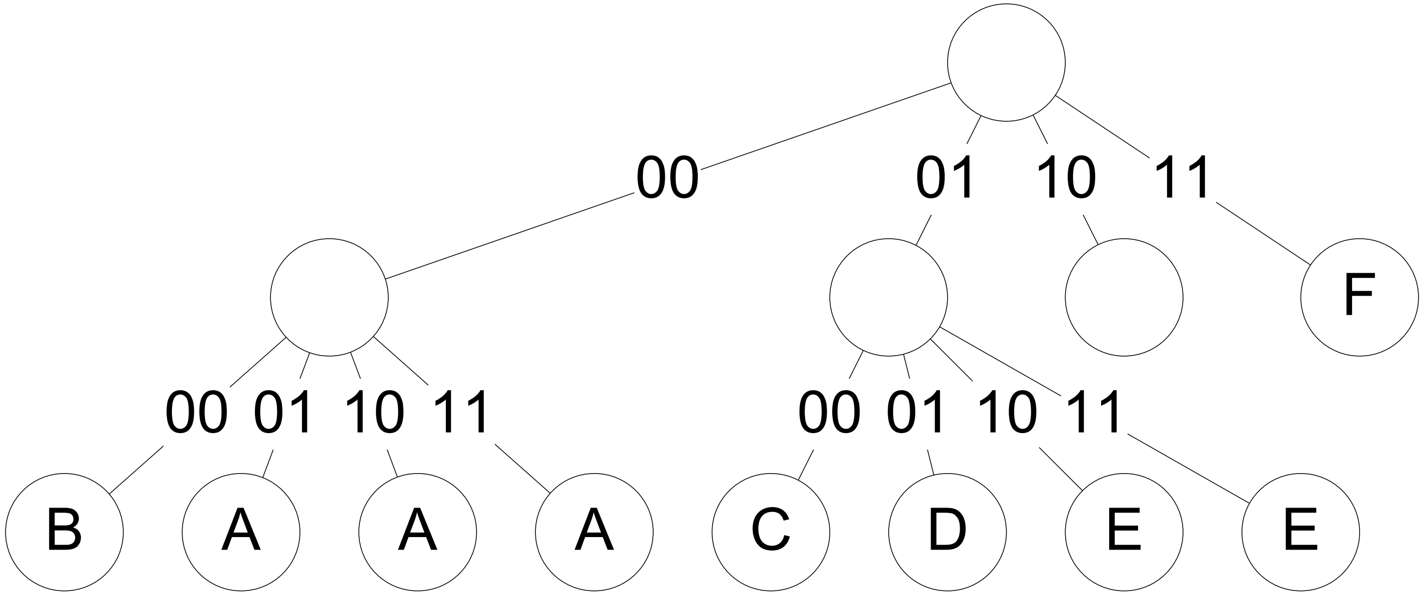
Path compression

- Path-compressed binary tree
  - Avoids long branches with only one node
  - Annotation to determine which bit to compare
  - Final node needs to be checked – otherwise backtracking



Tries

- Check multiple bits per step



Content Addressable Memory (CAM)

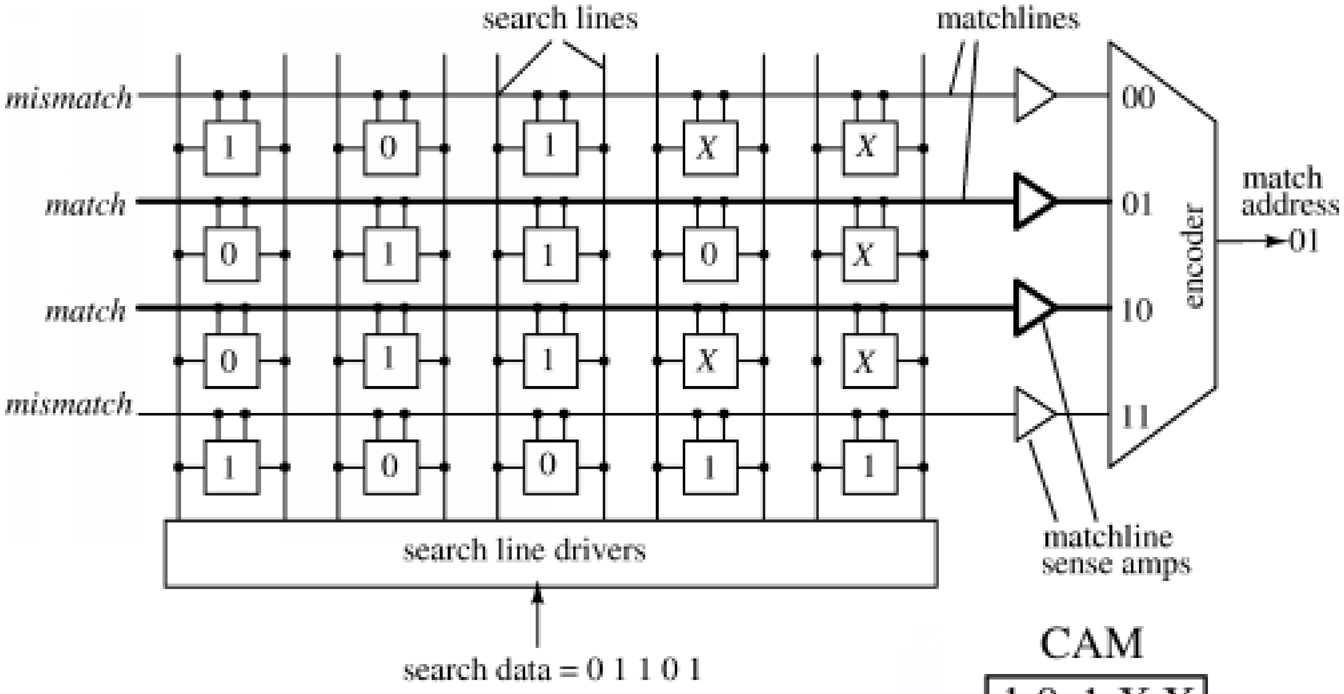
- Uses hardware to complete search in a single cycle
- O(1)
- Fast massively parallel lookup engine
- Large power consumption due to large amount of comparison circuitry
- Binary (0, 1) and Ternary (0, 1, X) CAMs. Latter most popular due to LPF.

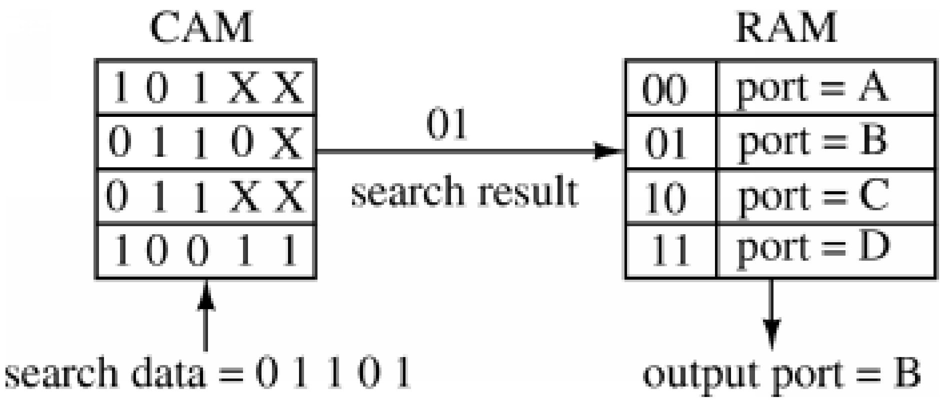
TCAM Example

Line No.	Address (Binary)	Output Port
1	101XX	A
2	0110X	B
3	011XX	C
4	10011	D

- Lookup *01101*.

TCAM





https://www.pagiamtzis.com/cam/camintro/

Hardware implementation

- Ternary content-addressable memory (TCAM)
  - Parallel lookup across all entries
  - 'x' indicates "don't care"

search word

TCAM entries

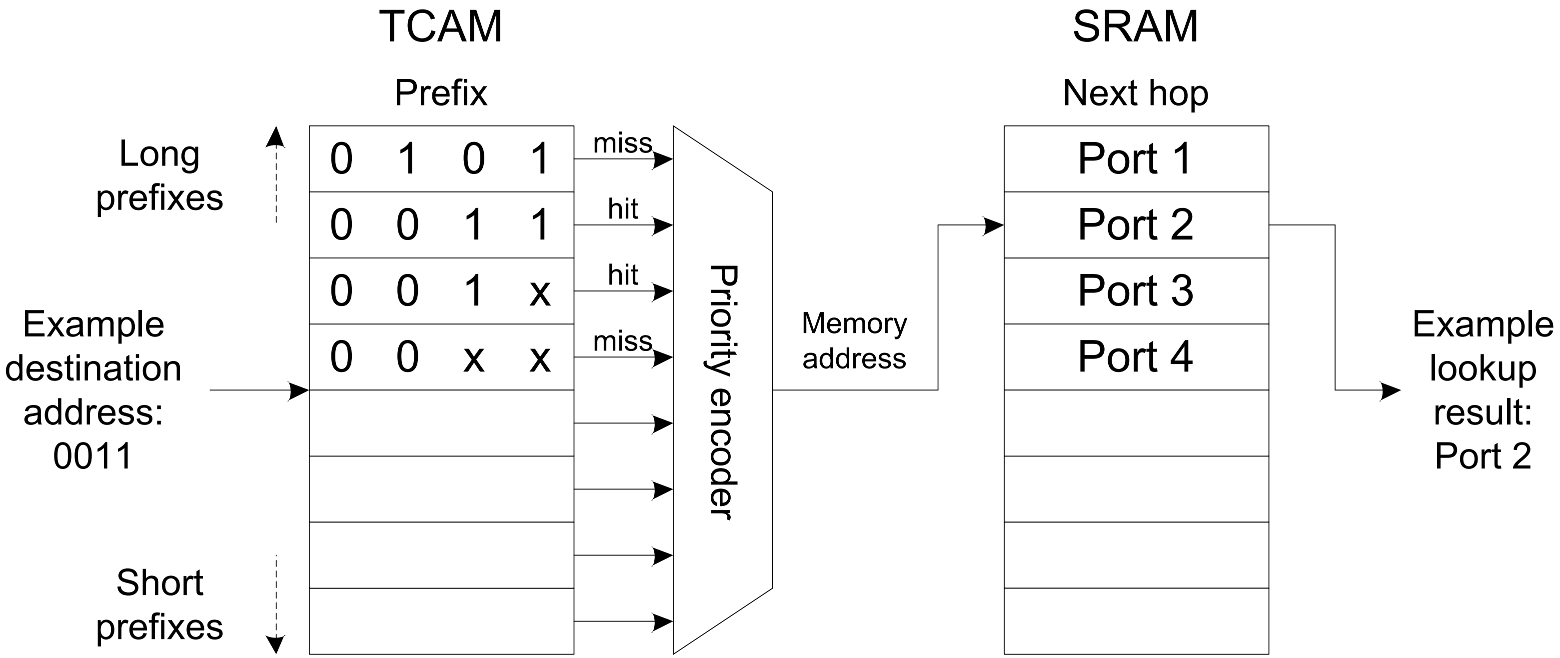
0	0	0	0	x	x	x	x	B
0	1	0	1	x	x	x	x	D
0	1	1	x	x	x	x	x	E
0	1	x	x	x	x	x	x	C
1	1	x	x	x	x	x	x	F
0	x	x	x	x	x	x	x	A

data word

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

Hardware implementation

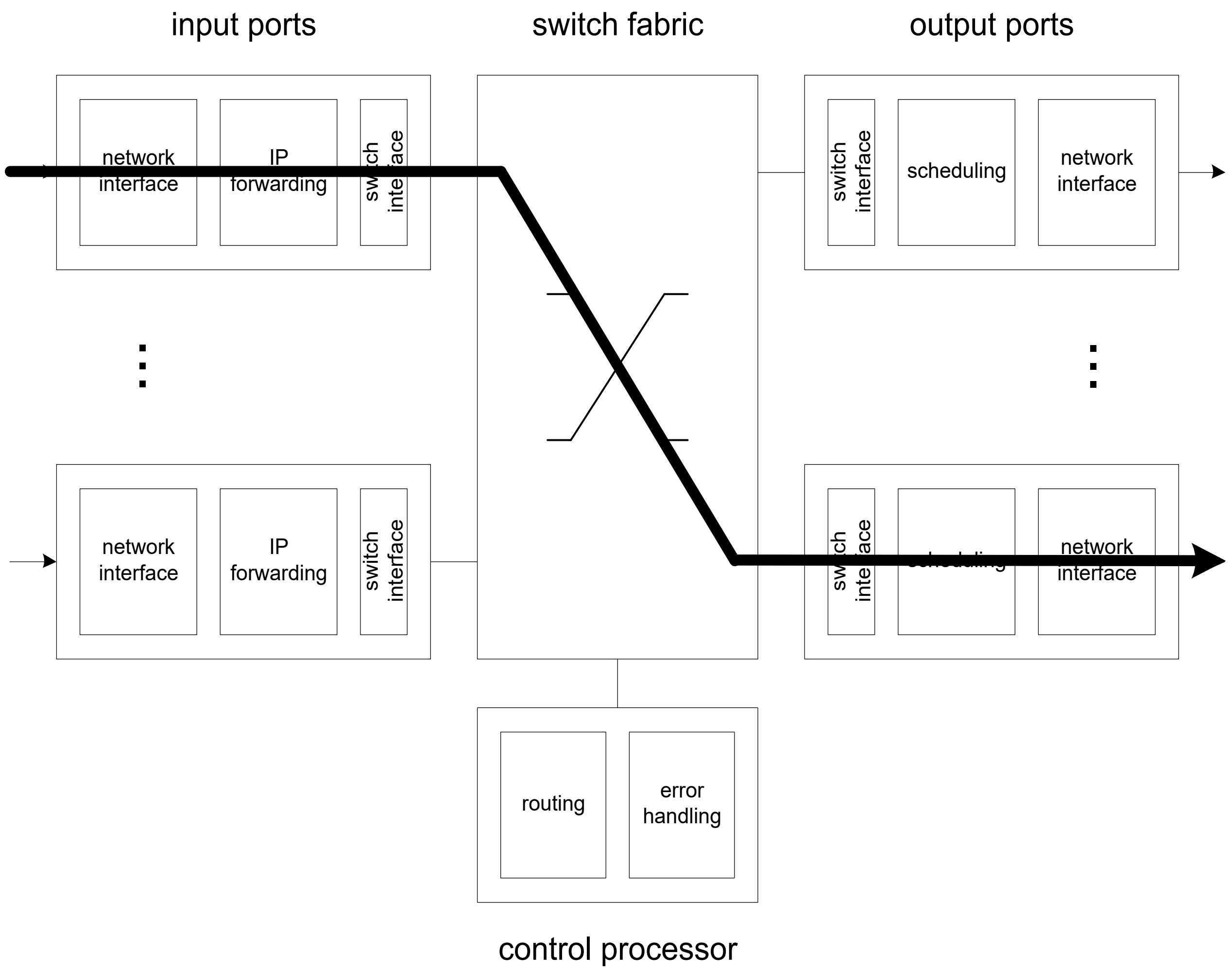
- TCAM operation



Prefix lookup issues

- Performance concerns
  - Lookups per second
  - Memory requirements
  - Power requirements
  - Ability to handle updates
- Lots of research in past years
  - Many specialized solutions

Router wrap-up



Slides were inspired by (and are based on) related courses of Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich), Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).

Slides were inspired by (and are based on) related courses of
Nick McKeown (Stanford), Laurent Vanbever (ETH Zurich),
Jennifer Rexford (Princeton) and Noa Zilberman (Cambridge).