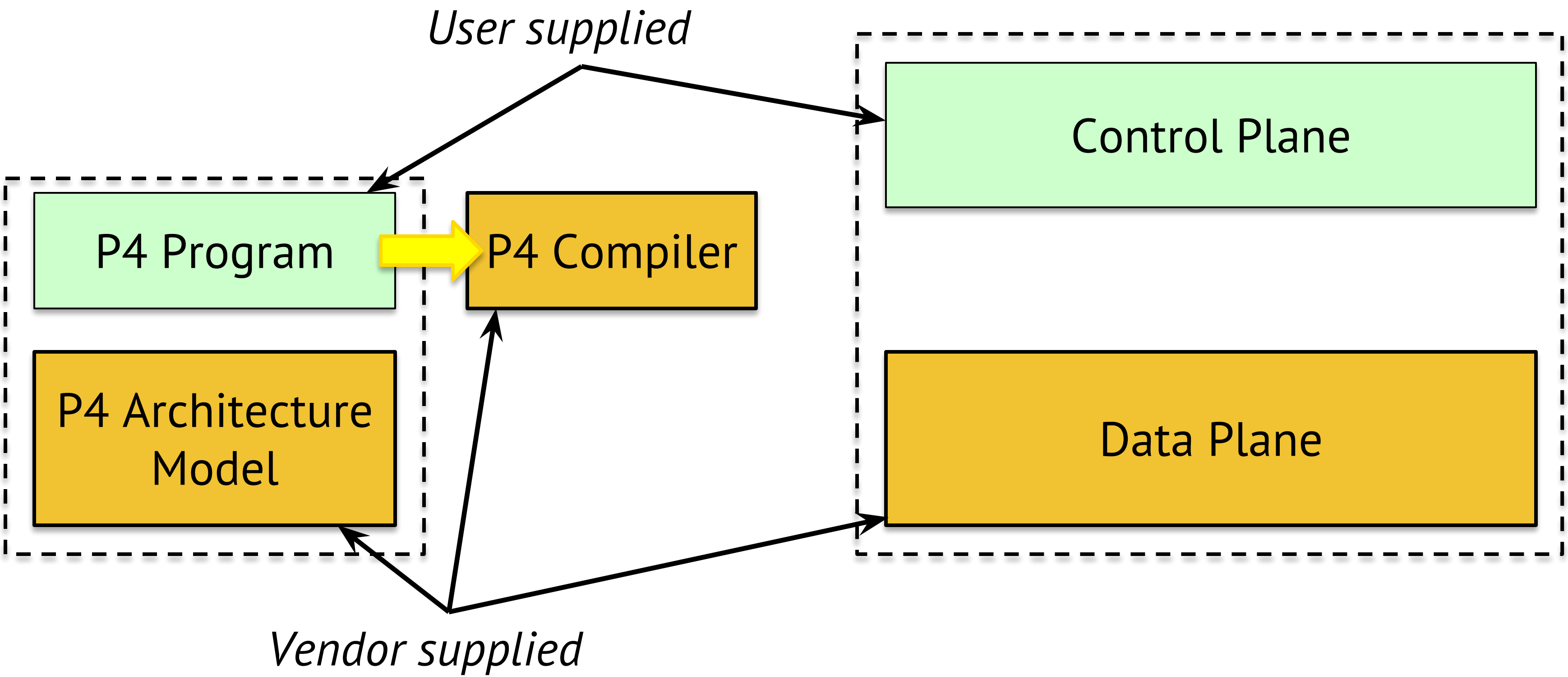
CS 6114

Match-Action Tables and P4 Runtime Control

Praveen Kumar

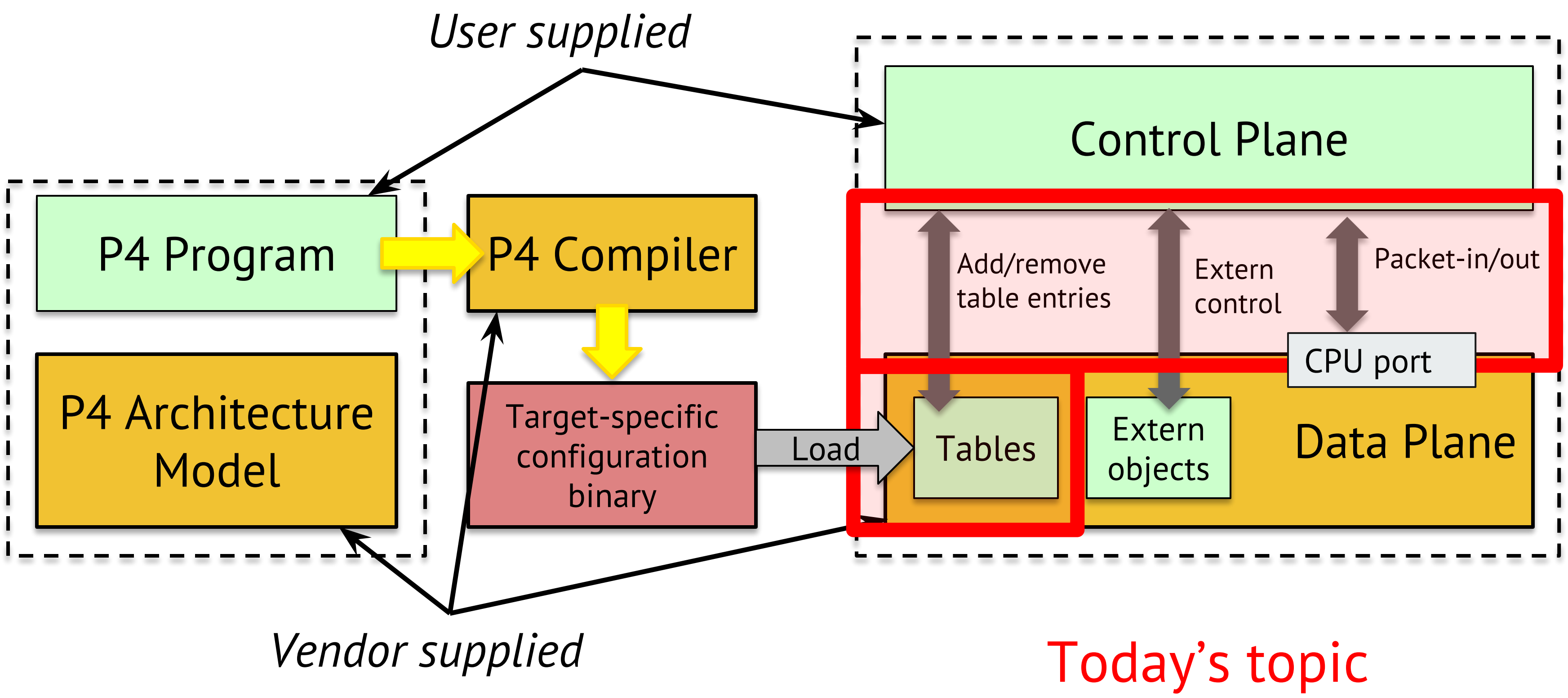
Fall 2018
9/27/2018

Components

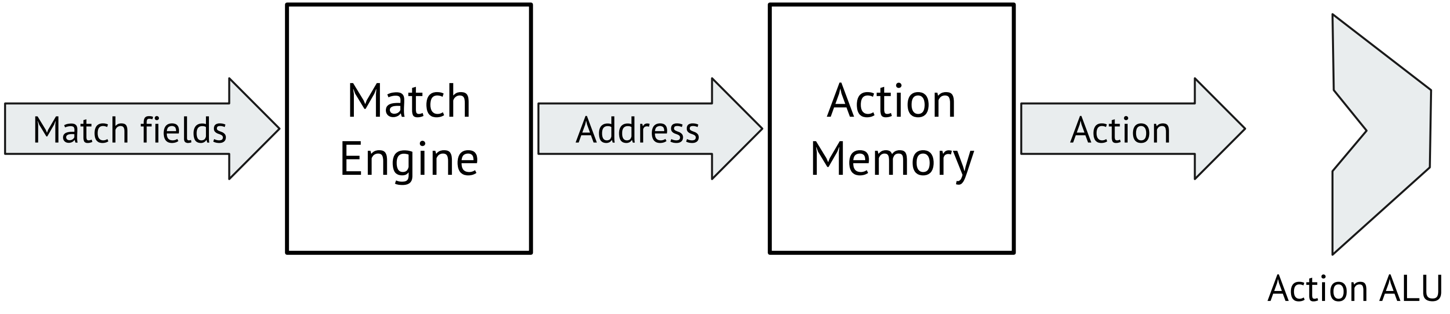


User supplied
Control Plane
P4 Program
P4 Compiler
P4 Architecture Model
Data Plane
Vendor supplied

Runtime control of P4 data planes



Match-Action Tables



Match-Action Tables

Match Types?

Match-Action Tables

Match Types

- Exact

ipv4.dstAddr	action
10.0.0.1	l2_switch

- Longest Prefix Match (LPM)

ipv4.dstAddr	action
10.0.0.0/24	l3_switch

- Ternary

ipv4.dstAddr	action
value=10.0.0.0 mask=0xFFFFFF00	l3_switch

How would you implement these? Fast Search - think terabits/s line rate.

Abstraction needed?

Match-Action Tables

Match Types

- Exact

ipv4.dstAddr	action
10.0.0.1	l2_switch

- Longest Prefix Match (LPM)

ipv4.dstAddr	action
10.0.0.0/24	l3_switch

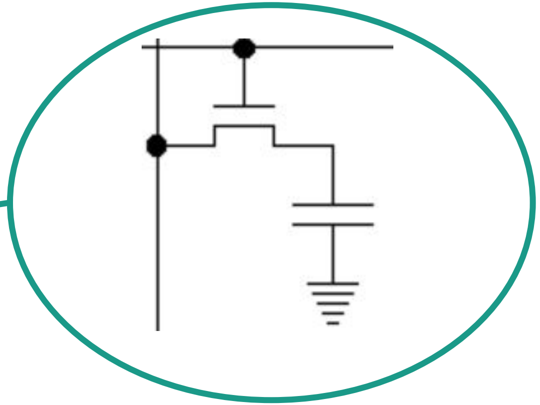
- Ternary

ipv4.dstAddr	action
value=10.0.0.0 mask=0xFFFFFF00	l3_switch

Need: **Input: match field value (data)**
Output: (address of) action

Random Access Memory

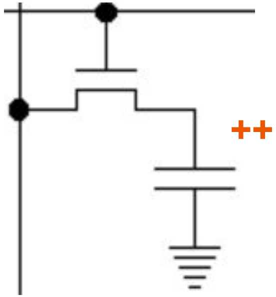
- **Dynamic RAM (DRAM)**
  - Slow
  - Cheap (1 transistor)
  - Example?
- **Static RAM (SRAM)**
  - Fast
  - Expensive (N transistors)
  - Example?



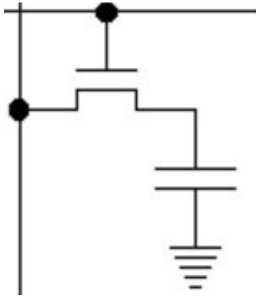
Abstraction (Read):

Input: address

Output: data



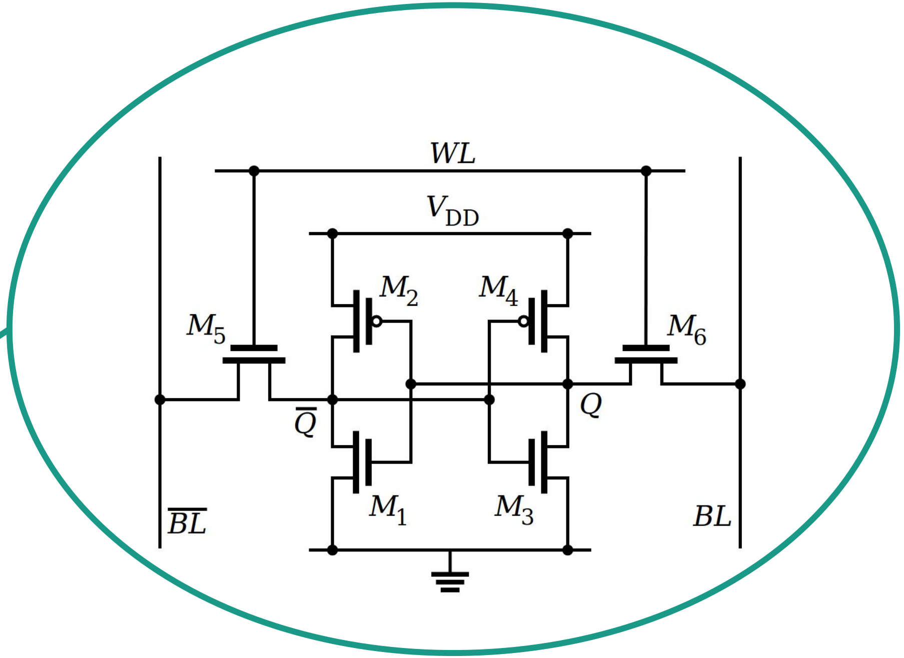
Stored bit = 1



Stored bit = 0

Random Access Memory

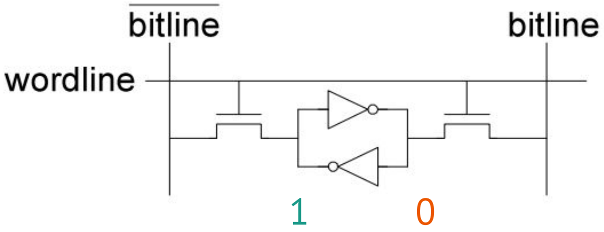
- **Dynamic RAM (DRAM)**
  - Slow
  - Cheap (1 transistor)
  - Main memory
- **Static RAM (SRAM)**
  - Fast
  - Expensive (4-6 transistors)
  - CPU registers

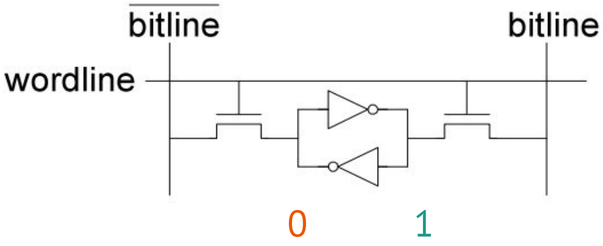


Abstraction (Read):

Input: address

Output: data





source: wikipedia

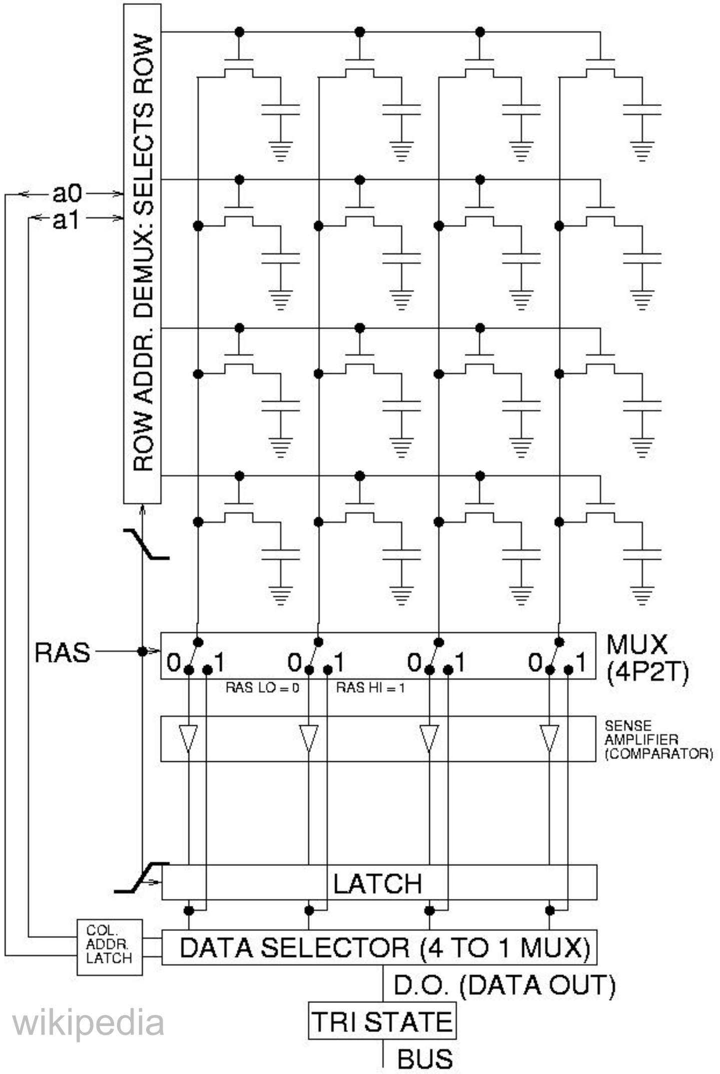
Random Access Memory

- **Dynamic RAM (DRAM)**
  - Slow
  - Cheap (1 transistor)
  - Main memory
- **Static RAM (SRAM)**
  - Fast
  - Expensive (4-6 transistors)
  - CPU registers

Abstraction (Read):

Input: address

Output: data



Source: wikipedia

Implementing Exact Match

- **Given: arrays of SRAM**
- **Want:**
    - In: match-key data
    - Out: action
- How do we get the match-action entry from the match-key?

```
lookup(10.0.0.2)
```

?

ipv4.dstAddr	**action**
10.0.0.1	l3_switch
10.0.0.2	l2_switch
10.0.0.3	l2_switch
DEFAULT	drop

Implementing Exact Match

- **Given: arrays of SRAM**
- **Want:**
  - In: match-key data
  - Out: action
- How do we get the match-action entry from the match-key?

Linear Scan

`lookup(10.0.0.2)` ?

Lookup time?

ipv4.dstAddr	**action**
10.0.0.1	l3_switch
10.0.0.2	l2_switch
10.0.0.3	l2_switch
DEFAULT	drop

Implementing Exact Match

- **Given arrays of on-chip SRAM**
- **Want:**
  - In: match-key data
  - Out: action
- **Solution**
  - Hash-based binary match

```
lookup(10.0.0.2): hash(10.0.0.2)
```

→

ipv4.dstAddr	action
10.0.0.1	l3_switch
10.0.0.2	l2_switch
10.0.0.3	l2_switch
DEFAULT	drop

Lookup time?

Implementing Exact Match

- **Given arrays of on-chip SRAM**
- **Want:**
  - In: match-key data
  - Out: action
- **Solution**
  - Hash-based binary match

```
lookup(10.0.0.2): hash(10.0.0.2)
```

→

ipv4.dstAddr	**action**
10.0.0.1	l3_switch
10.0.0.2	l2_switch
10.0.0.3	l2_switch
DEFAULT	drop

- **Collisions?**
  - Linear probing, chaining, etc.

Lookup time?
Average / (expected) Worst-case
Need: O(1)

Cuckoo Hashing

- Hash table with:

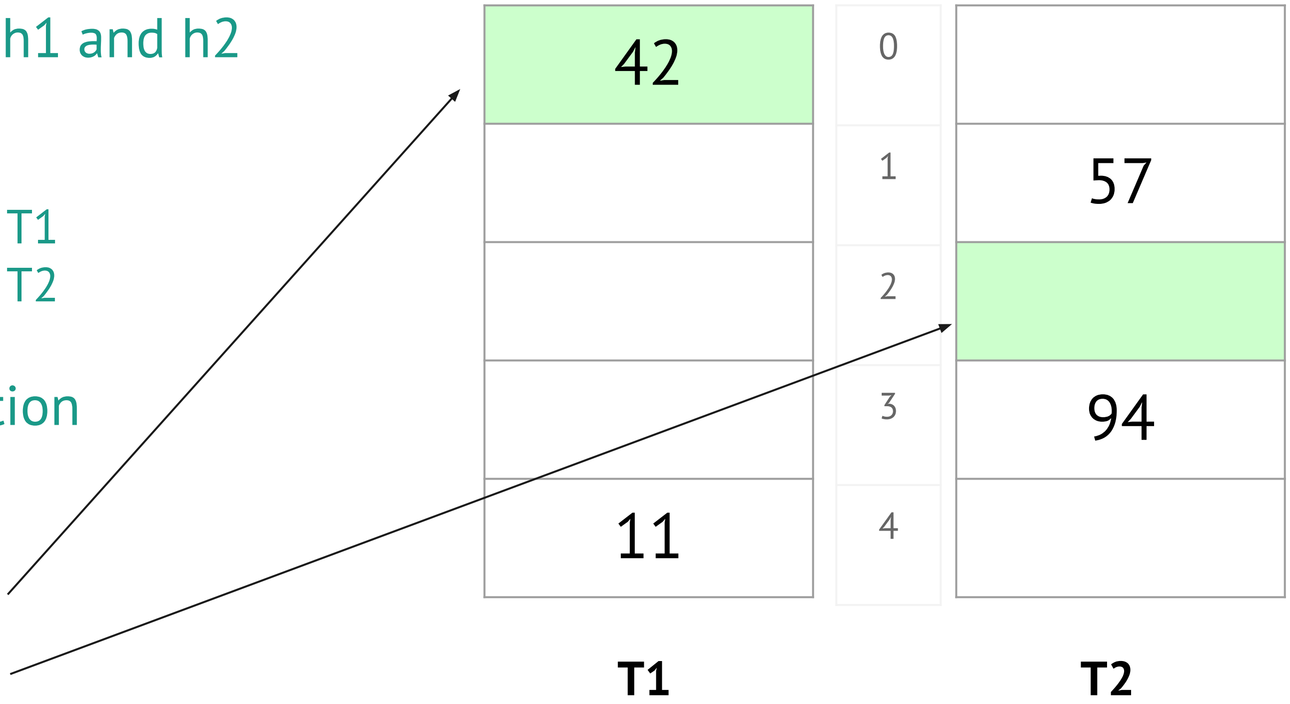
Worst-case lookup	**O(1)**
Worst-case delete	O(1)
Average-case insertion	O(1)

- Key Idea:
  - Maintain two hash tables with different hash functions
  - A key can be in one of the **only two possible locations**

Cuckoo Hashing - Lookup

- Tables T1 and T2
- Hash functions: h1 and h2
- Lookup key *k*:
  - Check:
    - h1(k) in T1
    - h2(k) in T2
  - Constant time
- Similar for deletion

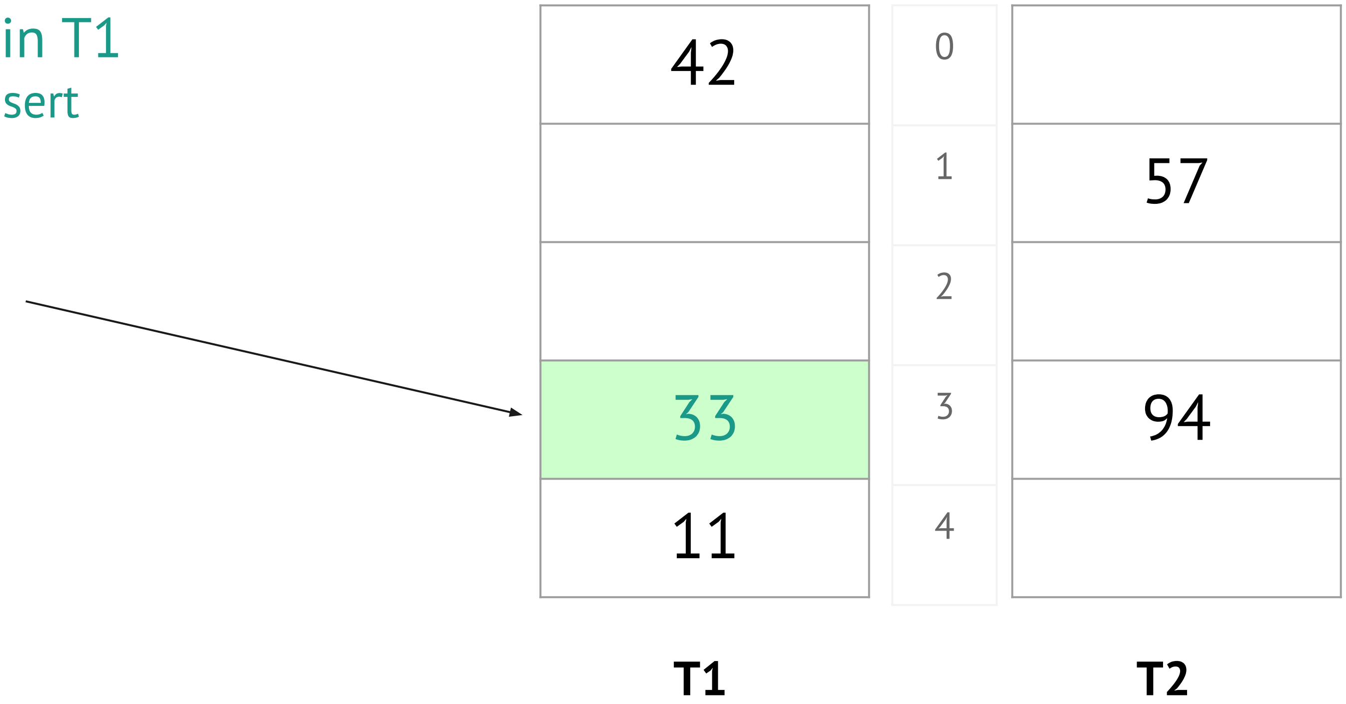
- Example:
  - `h1(42) = 0`
  - `h2(42) = 2`



Cuckoo Hashing - Insertion

- Insert (k)
- Check `h1(k)` in T1
  - If empty, insert

- `h1(33) = 3`

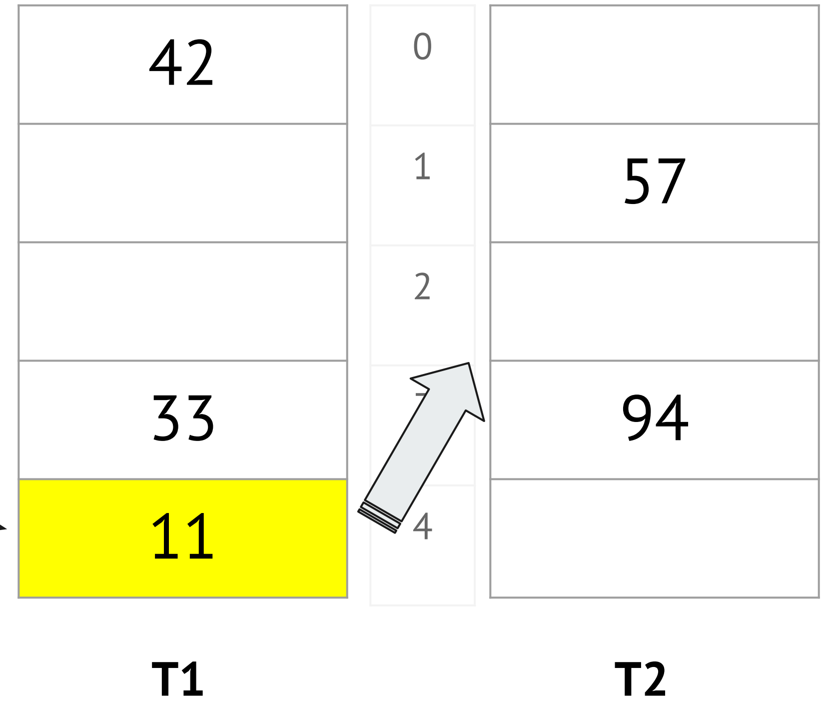


Cuckoo Hashing - Insertion

- Insert (k)
- Check h1(k) in T1
  - If empty, insert
  - Else, evict the current occupant to its position in T2 and insert

- h1(52) = 4

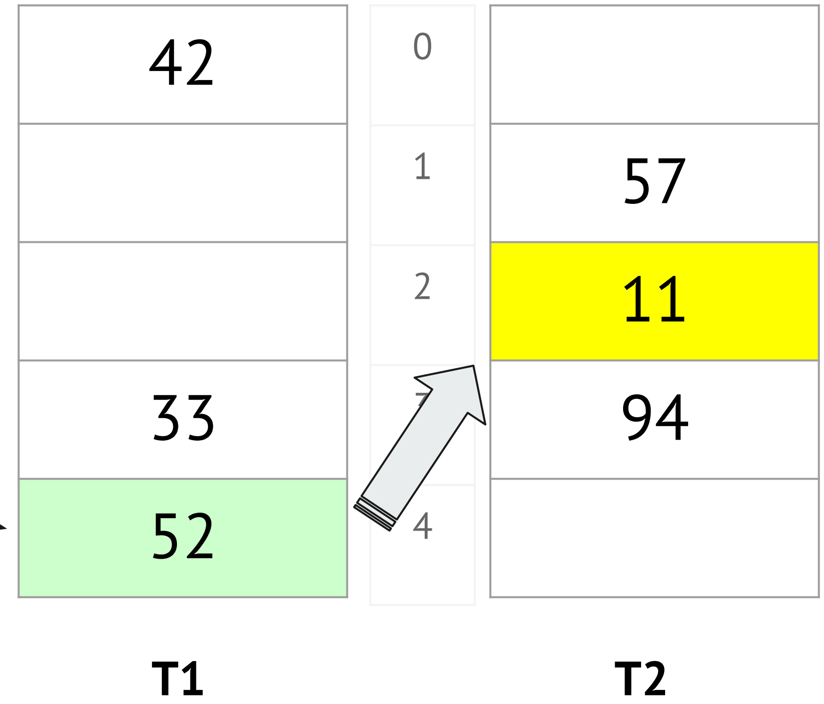
- h1(11) = 4, h2(11) = 2



Cuckoo Hashing - Insertion

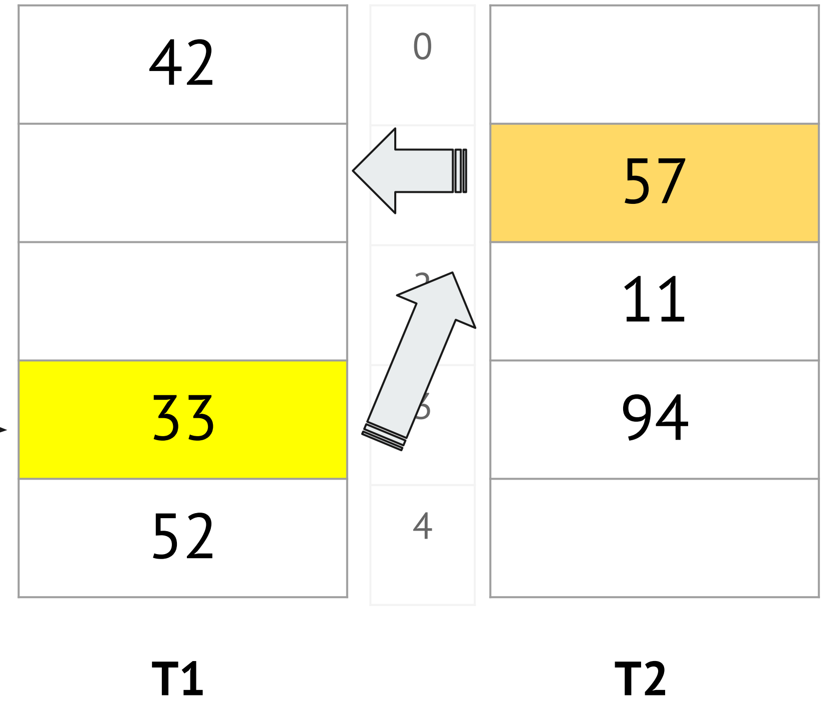
- Insert (k)
- Check h1(k) in T1
  - If empty, insert
  - Else, evict the current occupant to its position in T2 and insert

- `h1(52) = 4`
- `h1(11) = 4, h2(11) = 2`



Cuckoo Hashing - Insertion

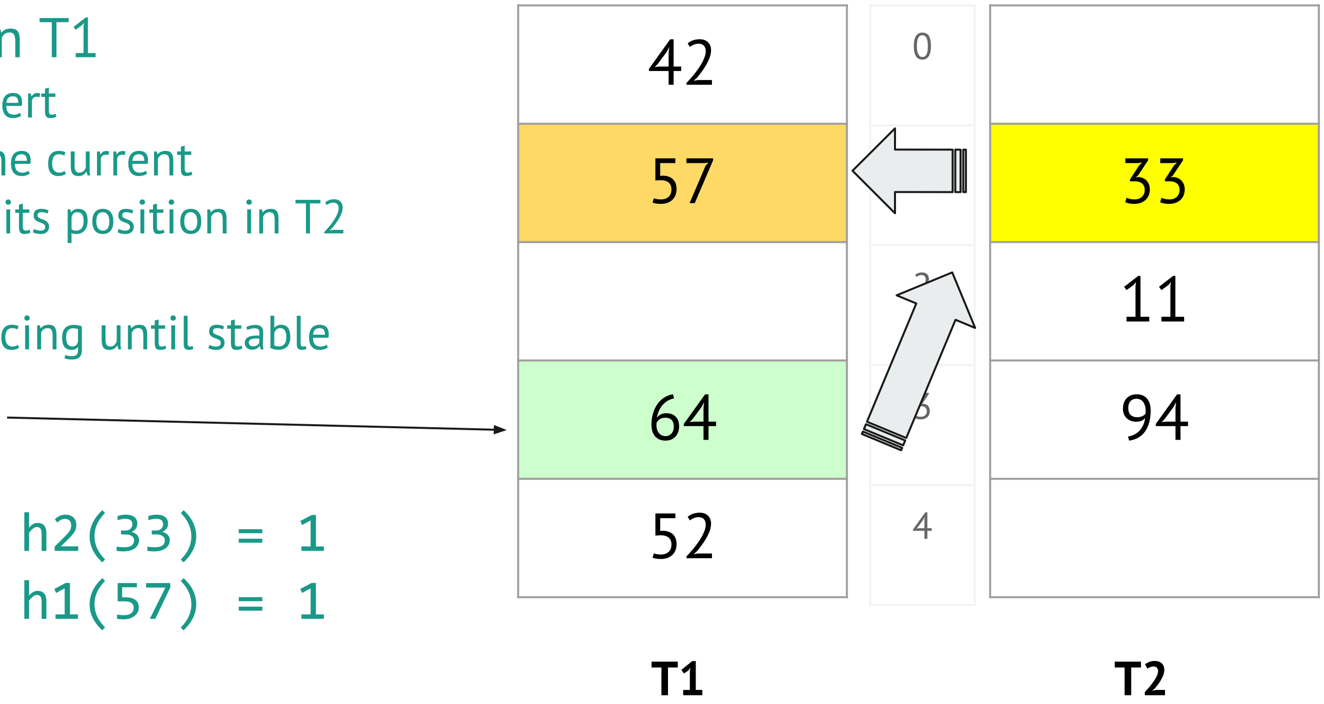
- Insert (k)
- Check h1(k) in T1
  - If empty, insert
  - Else, evict the current occupant to its position in T2 and insert
  - Iterate bouncing until stable
- `h1(64) = 3`
- `h1(33) = 3, h2(33) = 1`
- `h2(57) = 1, h1(57) = 1`



Cuckoo Hashing - Insertion

- Insert (k)
- Check h1(k) in T1
  - If empty, insert
  - Else, evict the current occupant to its position in T2 and insert
  - Iterate bouncing until stable
- h1(64) = 3

- h1(33) = 3, h2(33) = 1
- h2(57) = 1, h1(57) = 1



Match-Action Tables

Matches

- Exact
- **LPM**
- Ternary

ipv4.dstAddr	**action**
10.0.0.1/32	l3_switch
10.0.0.0/24	l2_switch
10.0.0.0/16	l2_switch
DEFAULT	drop

Want:

Input: match field value (data)
Output: (address of) action

LPM Table

Entries (can also be written using wildcards)

key	action
0*	a1
1*	a2
10*	a3
111*	a4
101*	a5

LPM Table

Entries

key	action
0*	a1
1*	a2
10*	a3
111*	a4
101*	a5

Attempt 1

- Check all entries
- Select longest match

1010 :
{1*, 10*, 101*} → 101*

Lookup time?

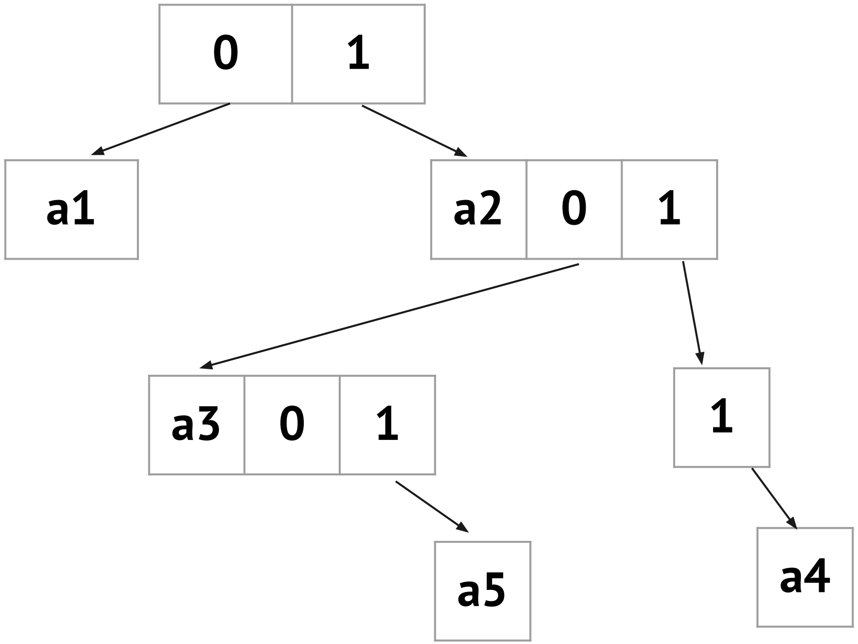
LPM Table

Entries

key	action
0*	a1
1*	a2
10*	a3
111*	a4
101*	a5

Attempt 2

Trie (radix tree / prefix tree)

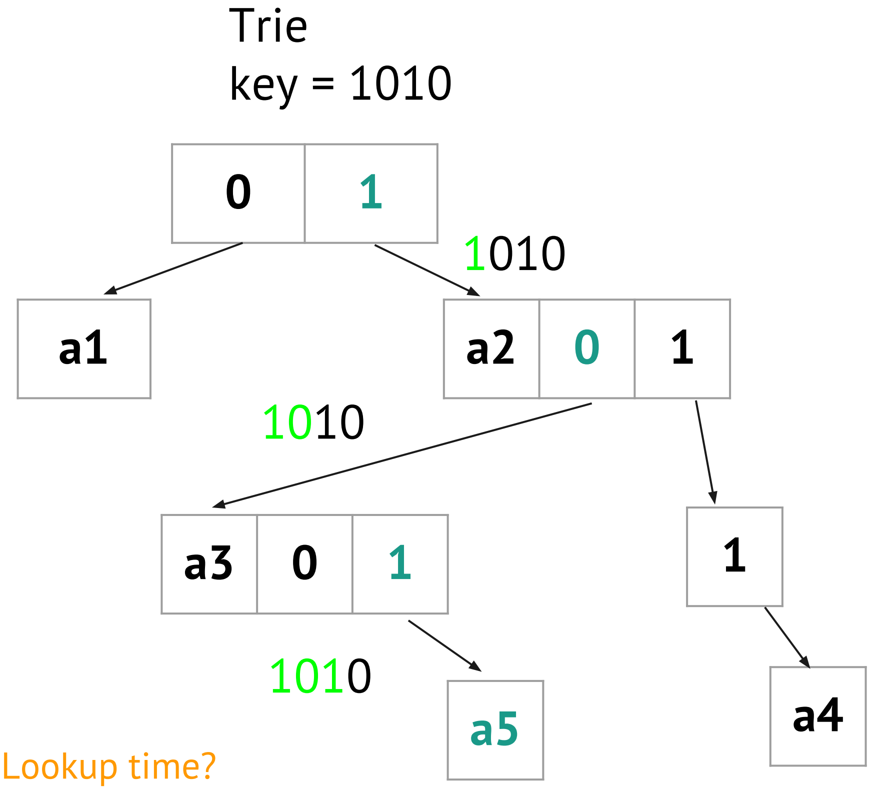


LPM Table

Entries

key	action
0*	a1
1*	a2
10*	a3
111*	a4
101*	a5

Attempt 2



Lookup time?

Match-Action Tables

Match Types

- Exact

- Longest Prefix Match (LPM)

- Ternary

RAM Abstraction (Read):

Input: address
Output: data

Need: **Input: match field value (data)**
Output: (address of) action

How would you implement these? Fast Search - think terabits/s line rate.

Abstraction needed?

Match-Action Tables

Match Types

- Exact
- Longest Prefix Match (LPM)
- Ternary

RAM Abstraction (Read):

Input: address

Output: data

Need: **Input: match field value (data)**

Output: (address of) action

Content Addressable Memory (CAM)

How would you implement these? Fast Search - think terabits/s line rate.

Abstraction needed?

Content Addressable Memory (CAM)

- **Think of CAM as**

  “ massively parallel lookup engine ”

- **Search all entries in parallel**

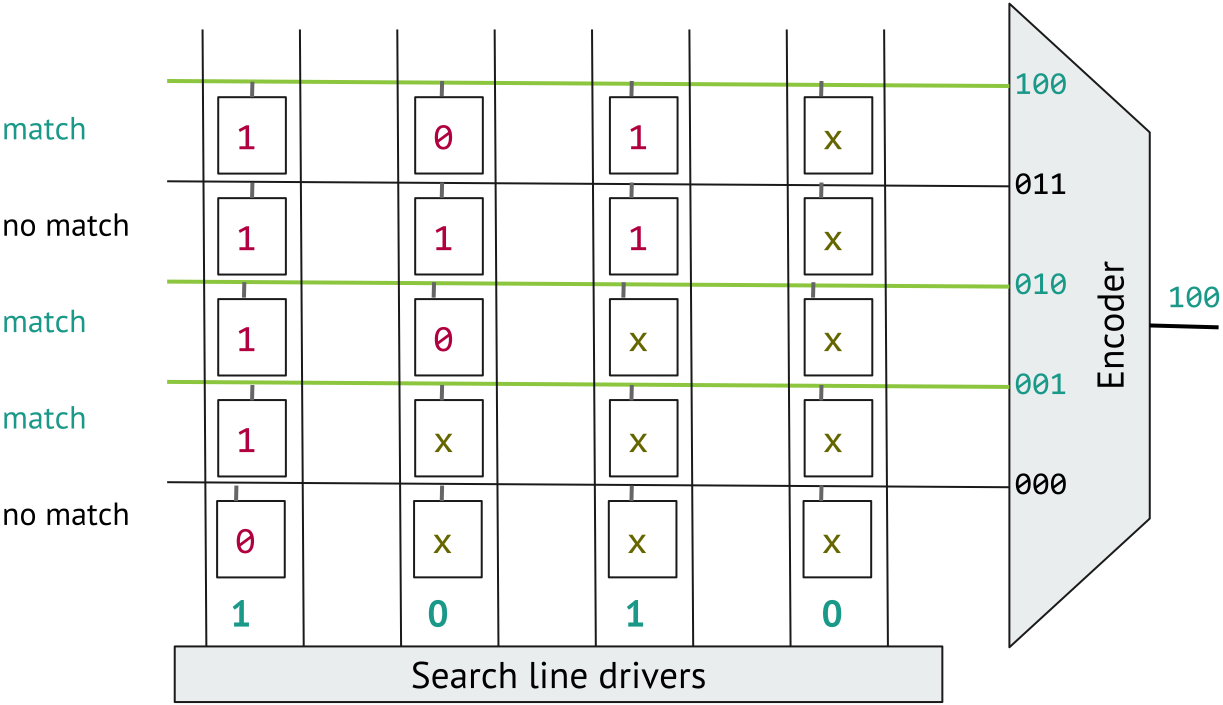
- **Select the best match in constant time**

key	action
0*	a1
1*	a2
10*	a3
111*	a4
101*	a5

Key: 1010

{1*, 10*, 101*} → 101*

TCAM Design



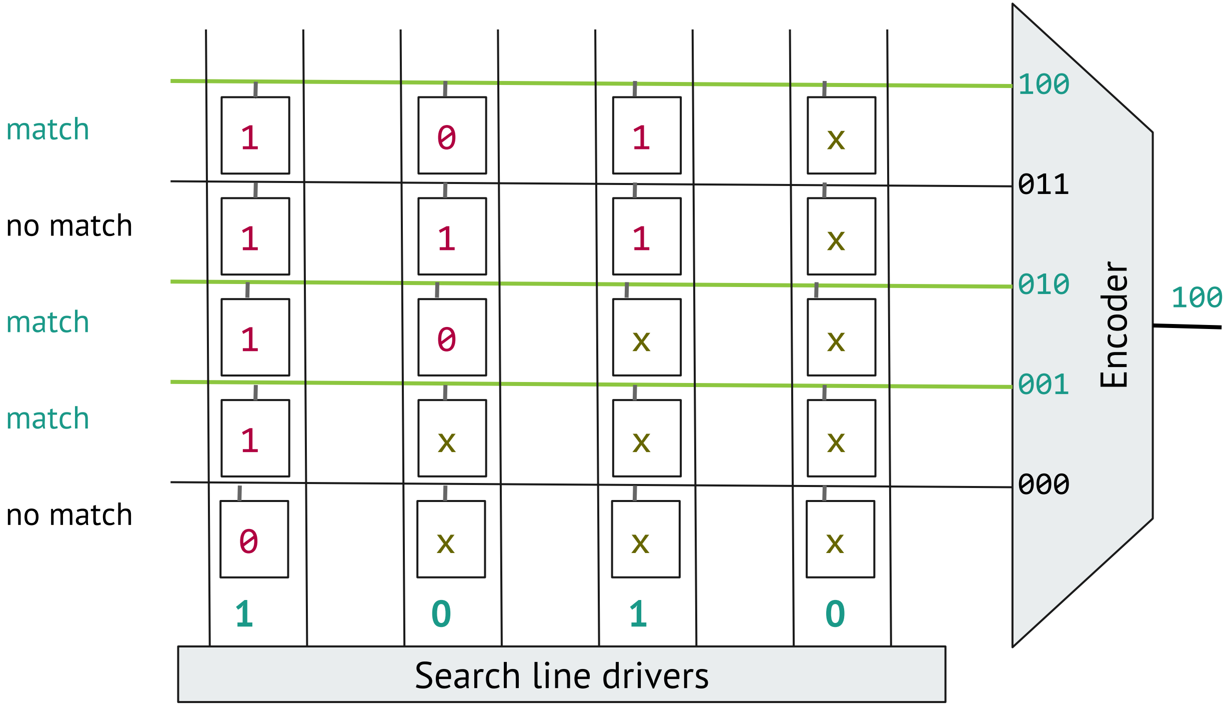
	key	action
100	101*	a5
011	111*	a4
010	10*	a3
001	1*	a2
000	0*	a1

Key: 1010

{1*, 10*, 101*} → 101*

Ternary CAM (TCAM) and *Binary* CAM (BCAM)

TCAM Design



	key	action
100	101*	a5
011	111*	a4
010	10*	a3
001	1*	a2
000	0*	a1

Key: 1010

{1*, 10*, 101*} → 101*

Ternary CAM (TCAM): Great for LPM and Ternary matches

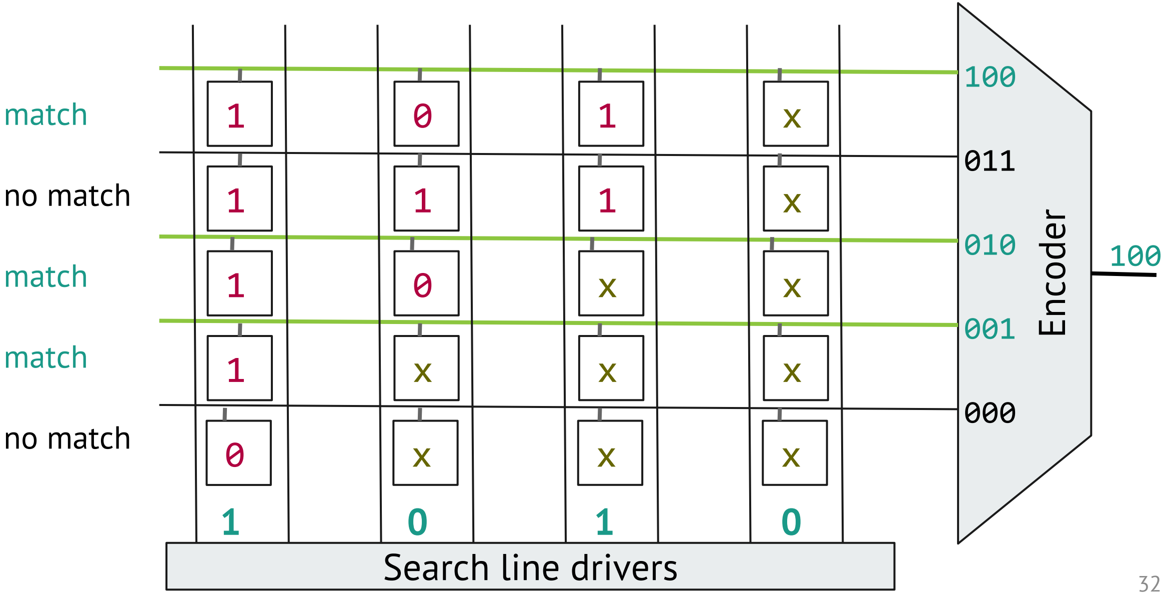
Memory Costs

TCAM compared to SRAM

- 6X more power
- 6-7X more area on chip
- 2-4X higher latency

Refer RMT (SIGCOMM 2013) paper for updated numbers

Trade-off: Efficiency vs Cost
How would you choose?

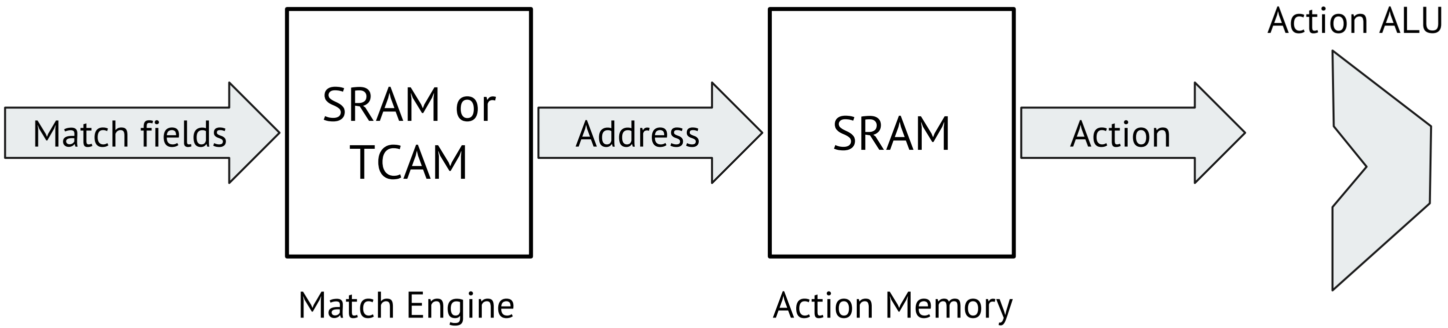


Bringing it all together

- **Exact match**
  - Cuckoo Hashing with SRAM
- **Wildcard match (LPM, Ternary)**
  - TCAM
- **Optimized for read; expensive writes**
  - Need for consistent updates (later)

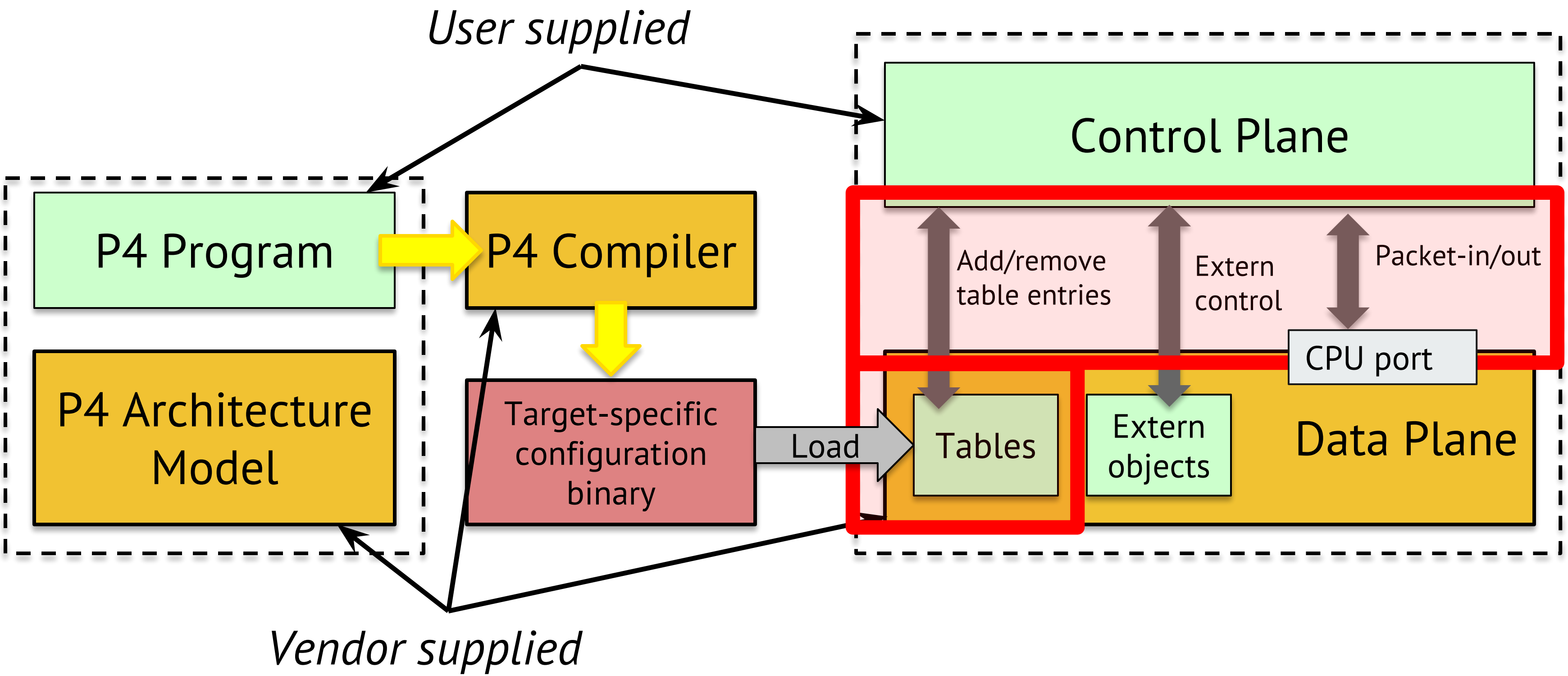
Refer RMT (SIGCOMM 2013) paper for more details

- **Forwarding Diagram**



Runtime Control

Runtime control of P4 data planes



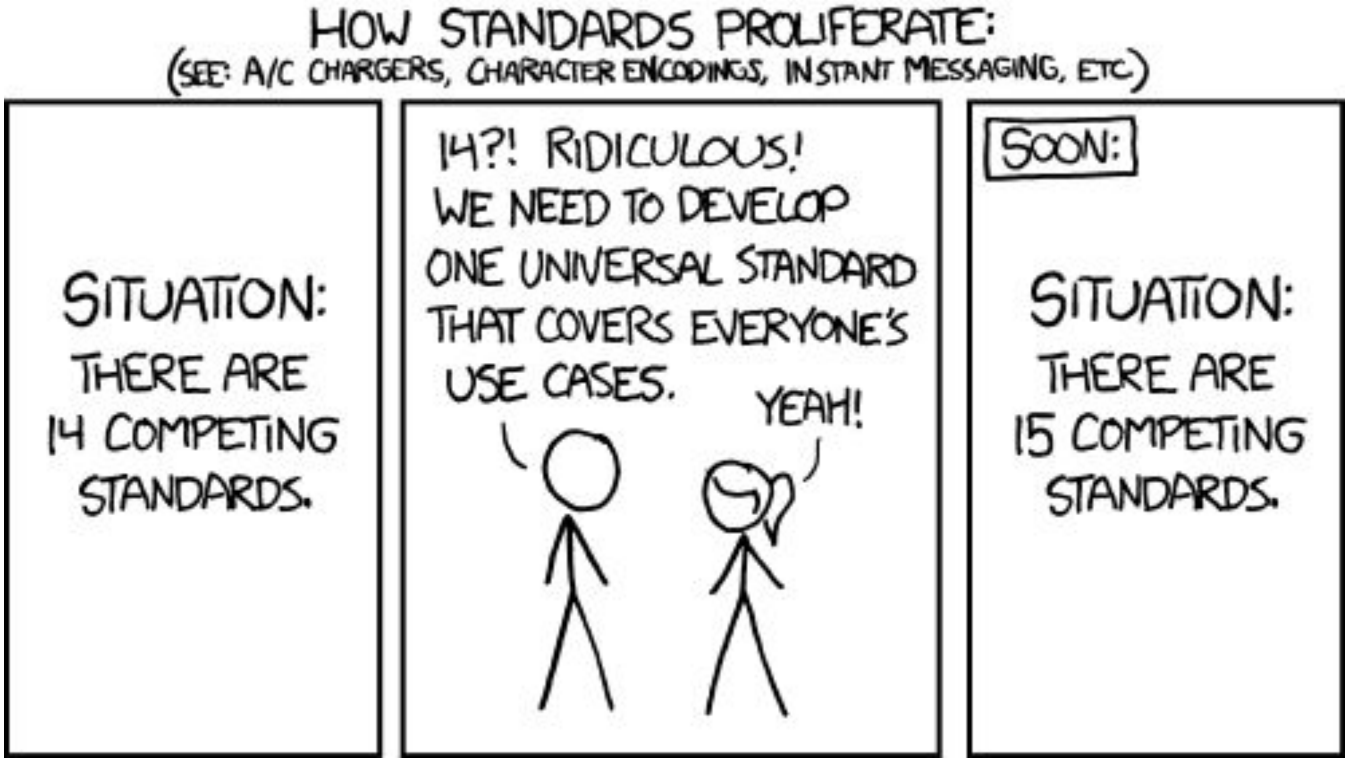
Existing approaches to runtime control

- **P4 compiler auto-generated runtime APIs**
  - Program-dependent -- hard to provision new P4 program without restarting the control plane!
- **BMv2 CLI**
  - Program-independent, but target-specific -- control plane not portable!
- **OpenFlow**
  - Quiz

Existing approaches to runtime control

- **P4 compiler auto-generated runtime APIs**
  - Program-dependent -- hard to provision new P4 program without restarting the control plane!
- **BMv2 CLI**
  - Program-independent, but target-specific -- control plane not portable!
- **OpenFlow**
  - Target-independent, but protocol-dependent -- protocol headers and actions baked in the specification!
- **OCP Switch Abstraction Interface (SAI)**
  - Target-independent, but protocol-dependent

Why do we need another data plane control API?



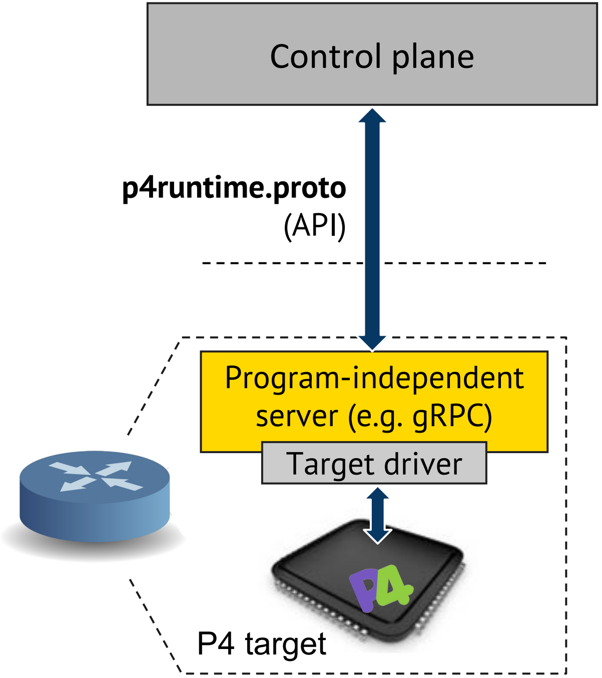
Source: https://xkcd.com/927/

Properties of a runtime control API

API	Target-independent	Protocol-independent
P4 compiler auto-generated	✔	✘
BMv2 CLI	✘	✔
OpenFlow	✔	✘
SAI	✔	✘
P4Runtime	✔	✔

What is P4Runtime?

- **Framework for runtime control of P4 targets**
  - Open-source API + server implementation
    - https://github.com/p4lang/PI
  - Initial contribution by Google and Barefoot
- **Work-in-progress by the p4.org API WG**
  - Draft of version 1.0 available
- **Protobuf-based API definition**
  - p4runtime.proto
  - gRPC transport
- **P4 program-independent**
  - API doesn't change with the P4 program
- **Enables field-reconfigurability**
  - Ability to push new P4 program without recompiling the software stack of target switches



Protocol Buffers Basics

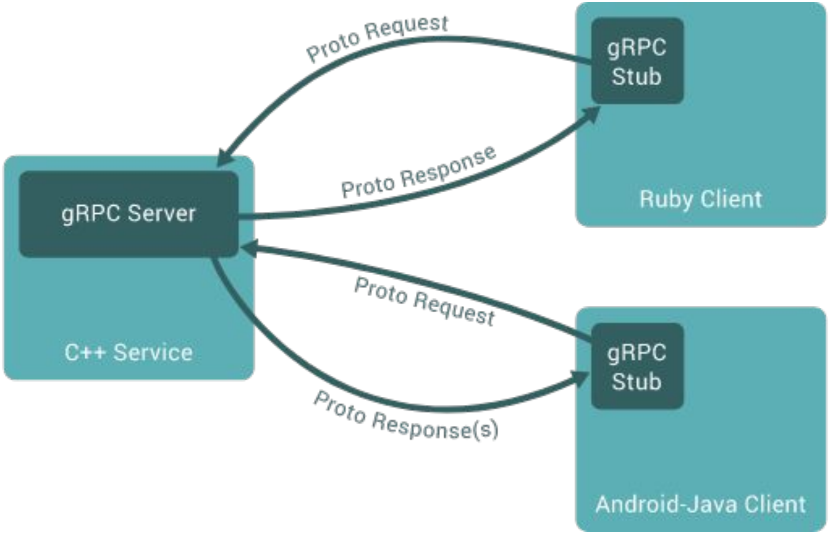
- Language for describing data for serialization in a structured way
- Common binary wire-format
- Language-neutral
  - Code generators for: *Action Script, C, C++, C#, Clojure, Lisp, D, Dart, Erlang, Go, Haskell, Java, Javascript, Lua, Objective C, OCaml, Perl, PHP, Python, Ruby, Rust, Scala, Swift, Visual Basic, ...*
- Platform-neutral
- Extensible and backwards compatible
- Strongly typed

```
syntax = "proto3";

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    string number = 1;
    PhoneType type = 2;
  }

  repeated PhoneNumber phone = 4;
}
```

https://developers.google.com/protocol-buffers/docs/overview

gRPC Basics

- Use Protocol Buffers to define service API and messages
- Automatically generate native stubs in:
  - C / C++
  - C#
  - Dart
  - Go
  - Java
  - Node.js
  - PHP
  - Python
  - Ruby



- Transport over HTTP/2.0 and TLS
  - Efficient single TCP connection implementation that supports bidirectional streaming

gRPC Service Example

```
// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

P4Runtime Service

Enables a local or remote entity to load the pipeline/program, send/receive packets, and read and write forwarding table entries, counters, and other chip features.

```
service P4Runtime {
 rpc Write(WriteRequest) returns (WriteResponse) {}
 rpc Read(ReadRequest) returns (stream ReadResponse) {}
 rpc SetForwardingPipelineConfig(SetForwardingPipelineConfigRequest)
     returns (SetForwardingPipelineConfigResponse) {}
 rpc GetForwardingPipelineConfig(GetForwardingPipelineConfigRequest)
     returns (GetForwardingPipelineConfigResponse) {}
 rpc StreamChannel(stream StreamMessageRequest)
     returns (stream StreamMessageResponse) {}
}
```

P4Runtime Service

Protobuf Definition:

https://github.com/p4lang/p4runtime/blob/master/proto/p4/v1/p4runtime.proto

Service Specification:

Working draft of version 1.0 is available now

https://p4.org/p4-spec/docs/P4Runtime-v1.0.0.pdf

P4Runtime Write Request

```
message WriteRequest {
 uint64 device_id = 1;
 uint64 role_id = 2;
 uint128 election_id = 3;
 repeated Update updates = 4;
}
```

```
message Update {
 enum Type {
   UNSPECIFIED = 0;
   INSERT = 1;
   MODIFY = 2;
   DELETE = 3;
 }
 Type type = 1;
 Entity entity = 2;
}
```

```
message Entity {
 oneof entity {
   ExternEntry extern_entry = 1;
   TableEntry table_entry = 2;
   ActionProfileMember
          action_profile_member = 3;
   ActionProfileGroup
          action_profile_group = 4;
   MeterEntry meter_entry = 5;
   DirectMeterEntry direct_meter_entry = 6;
   CounterEntry counter_entry = 7;
   DirectCounterEntry direct_counter_entry = 8;
   PacketReplicationEngineEntry
          packet_replication_engine_entry = 9;
   ValueSetEntry value_set_entry = 10;
   RegisterEntry register_entry = 11;
 }
}
```

P4Runtime Table Entry

p4runtime.proto simplified excerpts:

```
message TableEntry {
  uint32 table_id;
  repeated FieldMatch
match;
  Action action;
  int32 priority;
  ...
}
```

```
message Action {
  uint32 action_id;
  message Param {
    uint32 param_id;
    bytes value;
  }
  repeated Param params;
}
```

```
message FieldMatch {
  uint32 field_id;
  message Exact {
    bytes value;
  }
  message Ternary {
    bytes value;
    bytes mask;
  }
  ...
  oneof
field_match_type {
    Exact exact;
    Ternary ternary;
    ...
  }
}
```

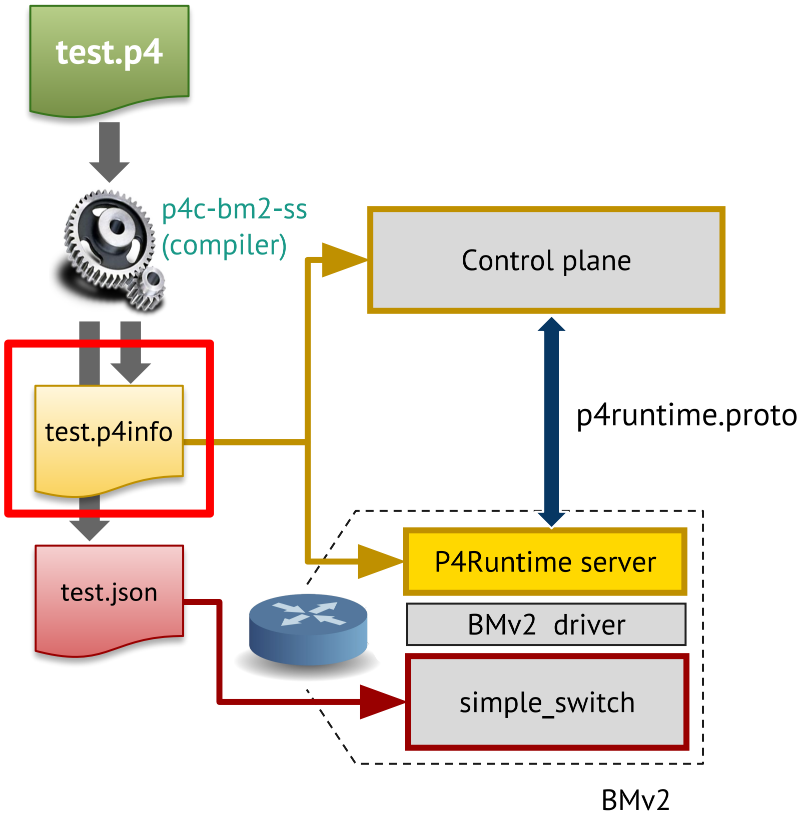
To add a table entry, the control plane needs to know:

- **IDs of P4 entities**
  - Tables, field matches, actions, params, etc.
- **Field matches for the particular table**
  - Match type, bitwidth, etc.
- **Parameters for the particular action**
- **Other P4 program attributes**

P4Runtime workflow

P4Info

- **Captures P4 program attributes needed at runtime**
  - IDs for tables, actions, params, etc.
  - Table structure, action parameters, etc.
- **Protobuf-based format**
- **Target-independent compiler output**
  - Same P4Info for BMv2, ASIC, etc.



Full P4Info protobuf specification:
https://github.com/p4lang/PI/blob/master/proto/p4/config/v1/p4info.proto

P4Info example

basic_router.p4

```
...

action ipv4_forward(bit<48> dstAddr,
                    bit<9> port) {
   /* Action implementation */
}

...

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

P4 compiler

basic_router.p4info

```
actions {
  id: 16786453
  name: "ipv4_forward"
  params {
    id: 1
    name: "dstAddr"
    bitwidth: 48
    ...
    id: 2
    name: "port"
    bitwidth: 9
  }
}
...
tables {
  id: 33581985
  name: "ipv4_lpm"
  match_fields {
    id: 1
    name: "hdr.ipv4.dstAddr"
    bitwidth: 32
    match_type: LPM
  }
  action_ref_id: 16786453
}
```

P4Runtime Table Entry Example

basic_router.p4

```
action ipv4_forward(bit<48> dstAddr,
                    bit<9>  port) {
   /* Action implementation */
}
table ipv4_lpm {
   key = {
       hdr.ipv4.dstAddr: lpm;
   }
   actions = {
       ipv4_forward;
       ...
   }
   ...
}
```

Logical view of table entry

```
hdr.ipv4.dstAddr=10.0.1.1/32
      -> ipv4_forward(00:00:00:00:00:10, 7)
```

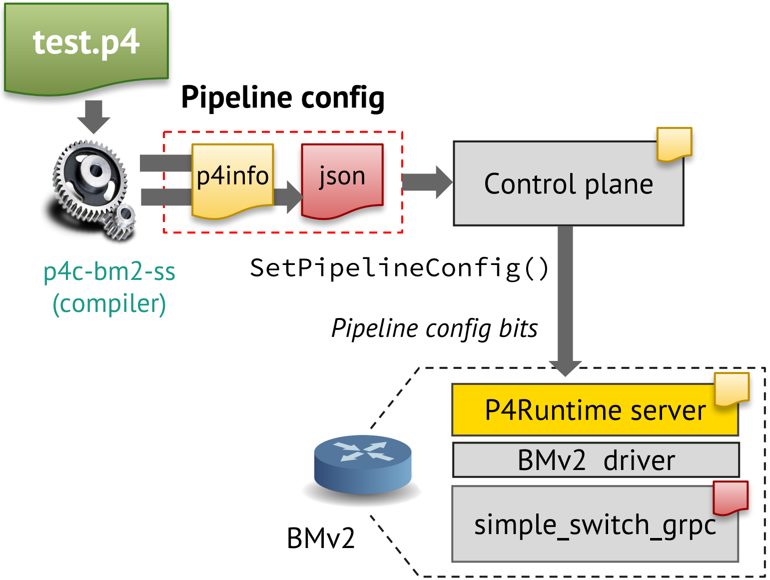
Control plane generates

Protobuf message

```
table_entry {
  table_id: 33581985
  match {
    field_id: 1
    lpm {
      value: "\n\000\001\001"
      prefix_len: 32
    }
  }
  action {
    action_id: 16786453
    params {
      param_id: 1
      value: "\000\000\000\000\000\n"
    }
    params {
      param_id: 2
      value: "\000\007"
    }
  }
}
```

P4Runtime SetPipelineConfig

```
message SetForwardingPipelineConfigRequest {
 enum Action {
   UNSPECIFIED = 0;
   VERIFY = 1;
   VERIFY_AND_SAVE = 2;
   VERIFY_AND_COMMIT = 3;
   COMMIT = 4;
   RECONCILE_AND_COMMIT = 5;
 }
 uint64 device_id = 1;
 uint64 role_id = 2;
 uint128 election_id = 3;
 Action action = 4;
 ForwardingPipelineConfig config = 5;
}
```




```
message ForwardingPipelineConfig {
 config.P4Info p4info = 1;
 // Target-specific P4 configuration.
 bytes p4_device_config = 2;
}
```

P4Runtime StreamChannel

```
message StreamMessageRequest {
 oneof update {
   MasterArbitrationUpdate
         arbitration = 1;
   PacketOut packet = 2;
 }
}
```

```
// Packet sent from the controller to the switch.
message PacketOut {
 bytes payload = 1;
 // This will be based on P4 header annotated as
 // @controller_header("packet_out").
 // At most one P4 header can have this annotation.
 repeated PacketMetadata metadata = 2;
}
```

```
message StreamMessageResponse {
 oneof update {
   MasterArbitrationUpdate
         arbitration = 1;
   PacketIn packet = 2;
 }
}
```

```
// Packet sent from the switch to the controller.
message PacketIn {
 bytes payload = 1;
 // This will be based on P4 header annotated as
 // @controller_header("packet_in").
 // At most one P4 header can have this annotation.
 repeated PacketMetadata metadata = 2;
}
```

P4Runtime Common Parameters

- `device_id`
  - Specifies the specific forwarding chip or software bridge
  - **Set to 0 for single chip platforms**
- `role_id`
  - Corresponds to a role with specific capabilities (i.e. what operations, P4 entities, behaviors, etc. are in the scope of a given role)
  - Role definition is currently agreed upon between control and data planes offline
  - **Default role_id (0) has full pipeline access**
- `election_id`
  - P4Runtime supports mastership on a per-role basis
  - Client with the highest election ID is referred to as the "master", while all other clients are referred to as "slaves"
  - **Set to 0 for single instance controllers**

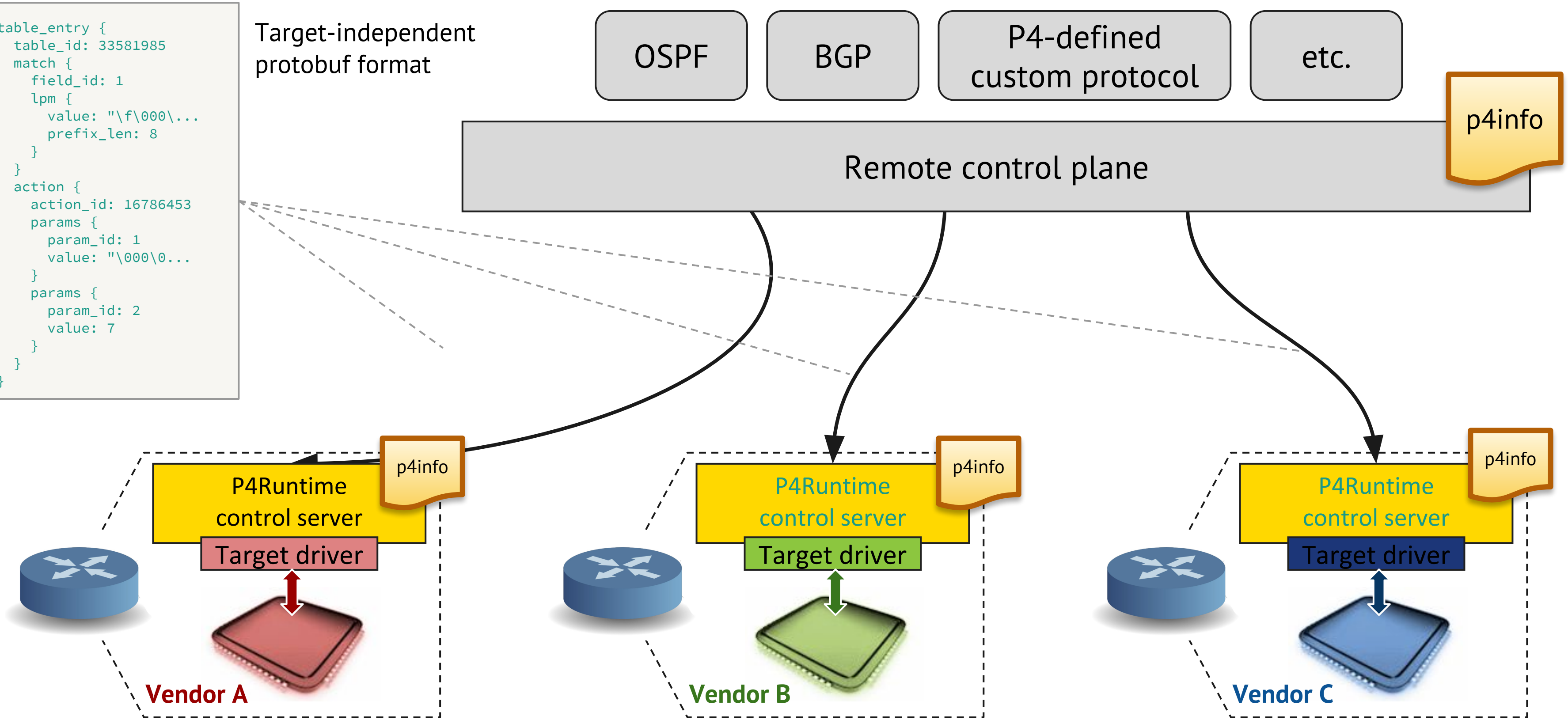
Mastership Arbitration

- **Upon connecting to the device, the client (e.g. controller) needs to open a `StreamChannel`**
- **The client must advertise its `role_id` and `election_id` using a `MasterArbitrationUpdate` message**
  - If `role_id` is not set, it implies the default role and will be granted full pipeline access
  - The `election_id` is opaque to the server and determined by the control plane (can be omitted for single-instance control plane)
- **The switch marks the client for each role with the highest `election_id` as master**
- **Master can:**
  - Perform `Write` requests
  - Receive `PacketIn` messages
  - Send `PacketOut` messages

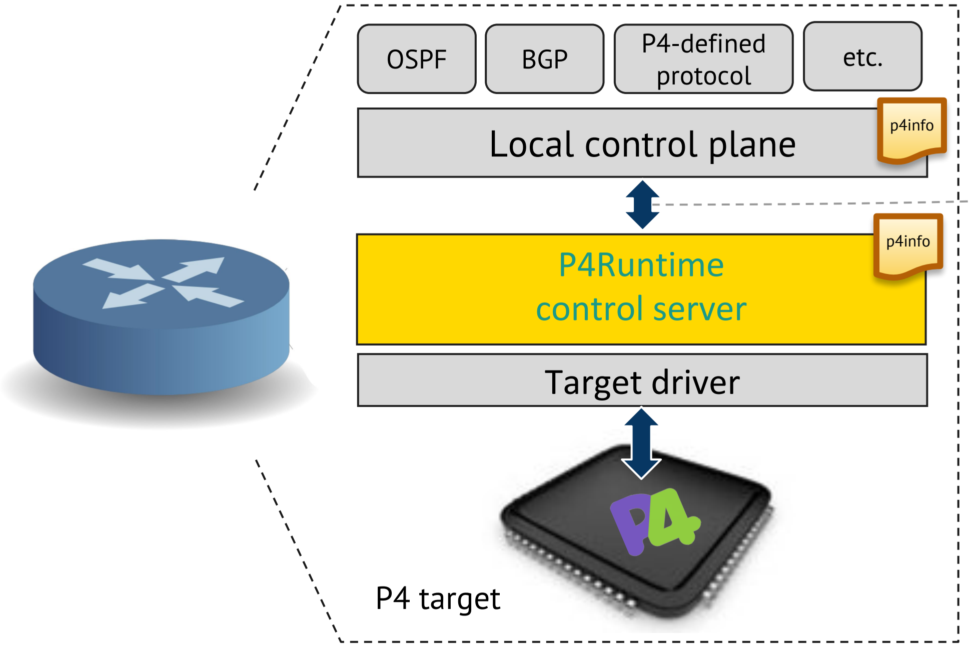
Remote control

```
table_entry {
  table_id: 33581985
  match {
    field_id: 1
    lpm {
      value: "\f\000\...
      prefix_len: 8
    }
  }
  action {
    action_id: 16786453
    params {
      param_id: 1
      value: "\000\0...
    }
    params {
      param_id: 2
      value: 7
    }
  }
}
```

Target-independent protobuf format



Local control



Same target-independent protobuf format

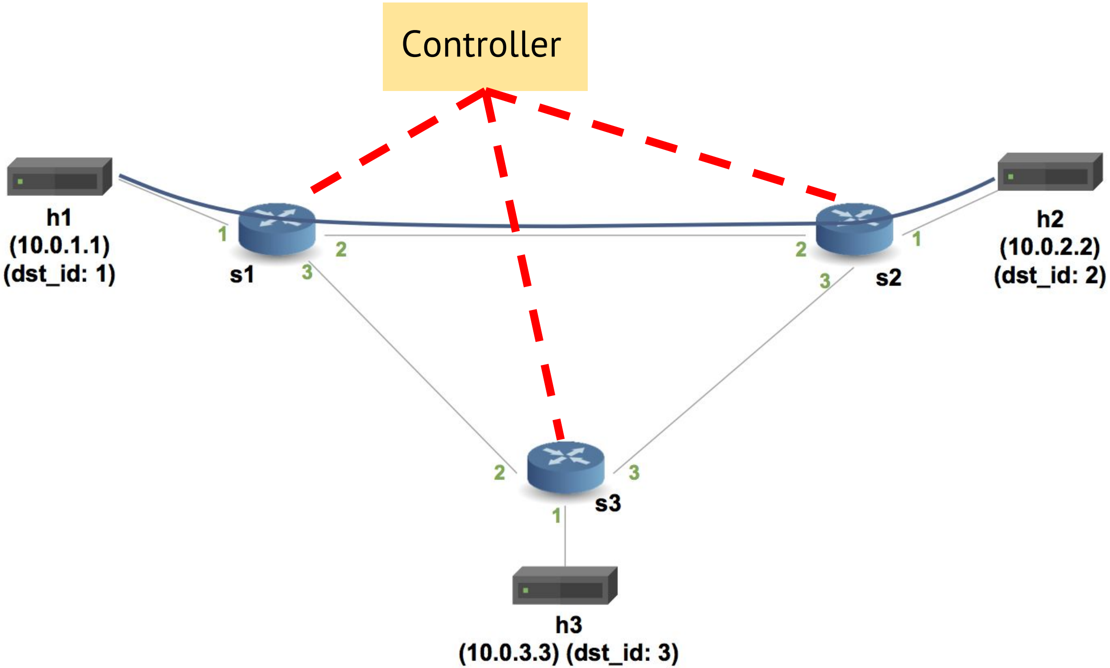
```
table_entry {
  table_id: 33581985
  match {
    field_id: 1
    lpm {
      value: "\f\000\...
      prefix_len: 8
    }
  }
  action {
    action_id: 16786453
    params {
      param_id: 1
      value: "\000\0...
    }
    params {
      param_id: 2
      value: 7
    }
  }
}
```

P4Runtime can be used equally well by a remote or local control plane

Demo

Implementing a Control Plane using P4Runtime

https://github.com/p4lang/tutorials/tree/master/exercises/p4runtime



P4Runtime API recap

Things we covered:

- **P4Runtime definition**
- **P4Info**
- **Table entries**
- **Set pipeline config**
- **Controller replication**
  - Via mastership arbitration

What we didn't cover:

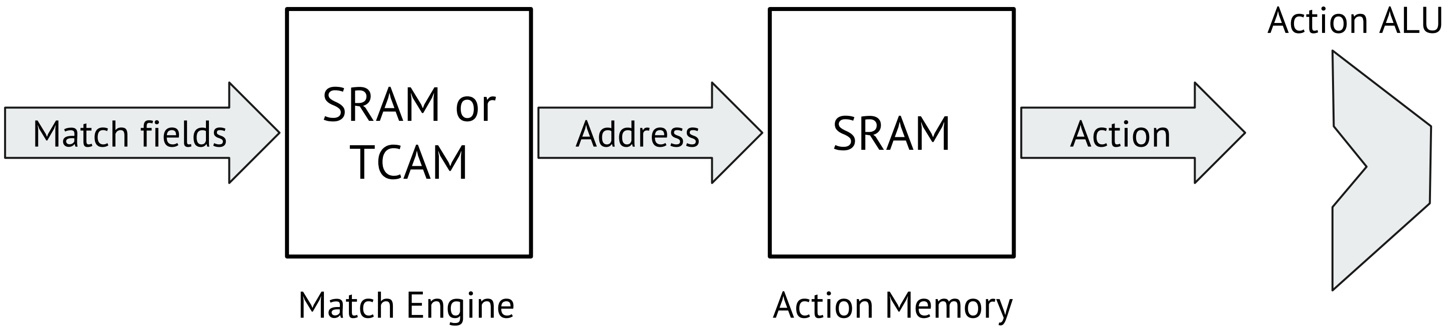
- **How to control other P4 entities**
  - Externs, counters, meters
- **Batched reads/writes**
- **Switch configuration**
  - Outside the P4Runtime scope
  - Achieved with other mechanisms
  - e.g., OpenConfig and gNMI

Summary

Match-Action Tables
- Exact: Cuckoo Hashing with SRAM
- Wildcard: TCAM

Optimized for read; expensive writes
- Need for consistent updates (later)

Forwarding Diagram



P4 Runtime
- Controlling P4 devices
- Protocol Independent
- Target Independent