Language and Open Source Update: Where are we now and what's next?

P4 Design Working Group

On behalf of Leo Alterman, Gordon Brebner, Mihai Budiu, Chris Dodd, Mukesh Hira, Raja Jayakumar, Naga Katta, Yan Luo, Peter Newman, Ben Pfaff, Satyam Sinha, Anirudh Sivaraman, Haoyu Song, Dan Talayco, Joe Tardo, Tom Tofigh, Johann Tonsing, Awanish Verma, and more.

2nd P4 Workshop
November 2015

Overview

- What's new in P4 v1.1?
- Post v1.1 goals and approach
- Update on open-source contributions
- Update on advanced use cases

New additions to P4 v1.1

- **Feature enhancement**
  - *set_metadata*( ) taking expression
    - Enables TLV-style header parsing
  - *modify_field*( ) taking expression
    - Avoids proliferation of primitive actions, keeping the language clean and simple
  - Proper data types and type-checking system
    - Action parameters are now typed
- **Unified way of embracing functional heterogeneity**
  - *extern* types and instances
- **Improved clarity and understandability of the spec**
  - Sequential-execution semantics

Concepts we reviewed, but didn't add to v1.1

- Architecture-language separation
  - Unified way of embracing architectural heterogeneity
  - Identify P4-programmable modules (*whiteboxes*) and declare their signatures
- Standard library
  - Primitive actions
  - Standard *extern* types
    - Stateful objects (counter, meter, and register)
    - Other objects that are subject to compile-time resource allocation
- Support for de-parser specification
  - Inverse of packet parsing

Primary goals for post-1.1 activity

- **Architecture-language separation**
  - Reuse the same compiler for new targets
- **Portability**
  - Reuse the same P4 code for new targets
- **Composability**
  - Write P4 code (library) once and reuse it many times

How?

- **Architecture-language separation**
  - Introduce architecture-modeling constructs in P4
- **Portability**
  - Standard architecture
  - Standard library
- **Composability**
  - Introduce new constructs for namespace and parameterization

Sample: Architecture-language separation

Switch Architecture Specification

```
// “arch.p4”
// Architecture declaration
parser P<H>(in packet_in packet,
              out H headers);
control Ingress<H>(
        inout H headers,
        in intrinsic_metadata_in imi,
        out intrinsic_metadata_out imo
);
control Deparser<H>(in H headers,
                        out packet_out packet);
package Switch<H>(Parser<H> p,
                       Ingress<H> ingress,
                       Deparser<H> deparser);
```

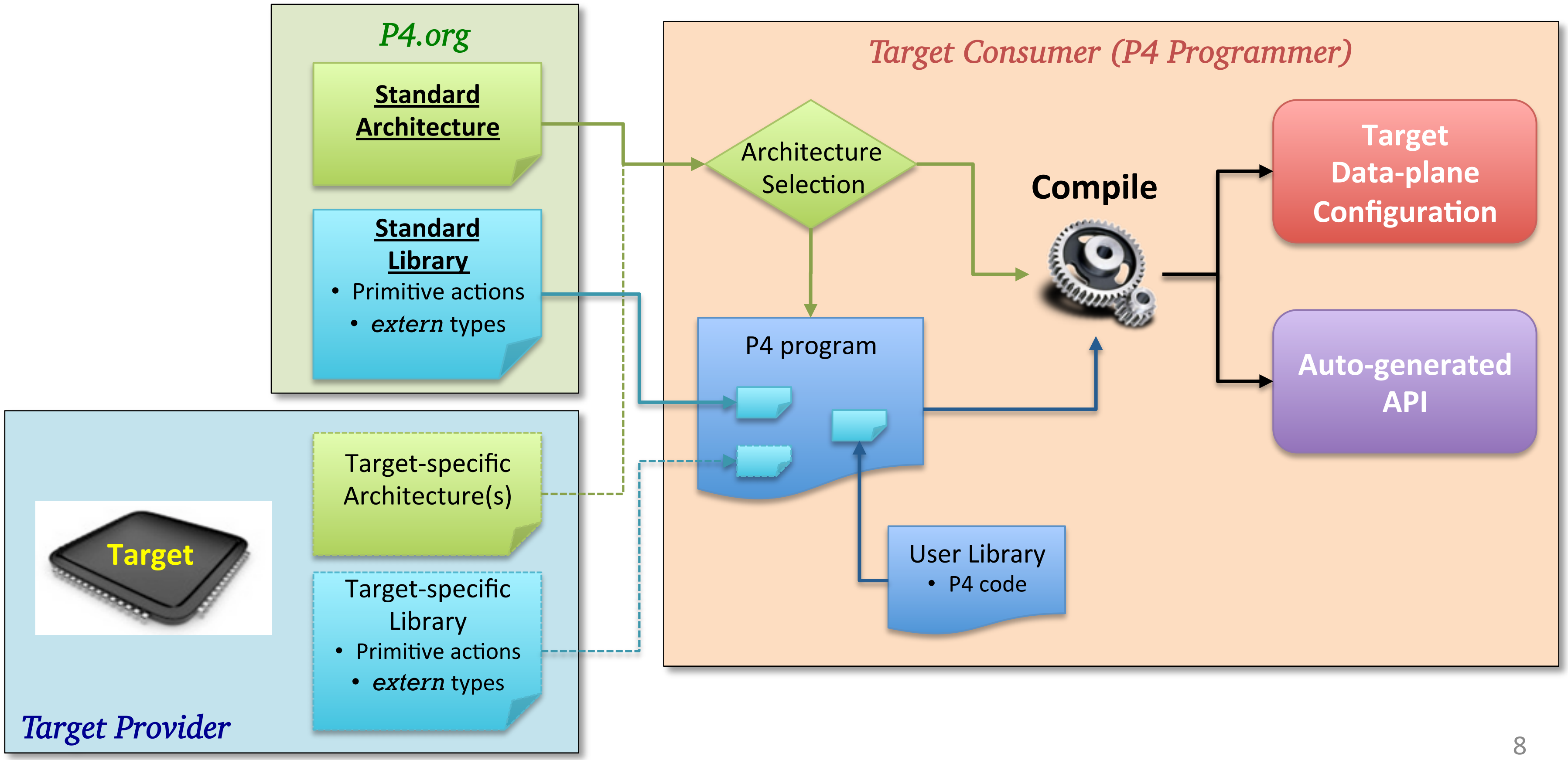
Switch Implementation (by user)

```
// Program written by user
#include “arch.p4”

parser MyParser(...) { ... }
control MyIngress(...) { ... }
control MyDeparser(...) { ... }

// Top-level element instantiation
Switch(MyParser(),
        MyIngress(),
        MyDeparser()) MySwitch;
```

Fitting all these together



Other goals for post v1.1

- Support for …
  - Incremental parsing
  - Deparser specification
  - Compile-time table population
  - Compile-time default-action specification
- Common control-plane API generation convention

Sub-group approach for p4-design

- **Potential sub-groups**
  - Language / Standard library
  - Standard architecture
  - API-generation convention
- Each sub-group could work with its own schedule and logistics
  - Conf calls, in-person meetings, or both
- Monthly in-person plenary meetings
- Seeking representatives who'd like to lead sub-group activities

Update on open-source contributions

- **Behavioral Model v2 (BMv2)**
  - Re-configurable fixed code; no code generation
  - Easier to add features, maintain, and understand
  - Architecture independent
  - Better logging and great test coverage
  - Decent performance
- **Packet Test Framework (PTF)**
  - Replaces OF Test Framework
  - Light weight, more features (network-level testing, etc.)

Update on advanced use cases

- In-band Network Telemetry
  - Full spec & prototype available
    - http://p4.org/p4/inband-network-telemetry/
    - P4 code, mininet-based test framework, and real-time data visualizer
- More in the pipeline
  - L4 load balancing, in-network Paxos, utilization-aware routing, etc.
- P4 code examples (assignments from P4 tutorials)
  - Source routing, flowlet switching, etc.