Efficient Network Stack with SDKLT, NPL

Venkat Pullela

ONF Connect 2019

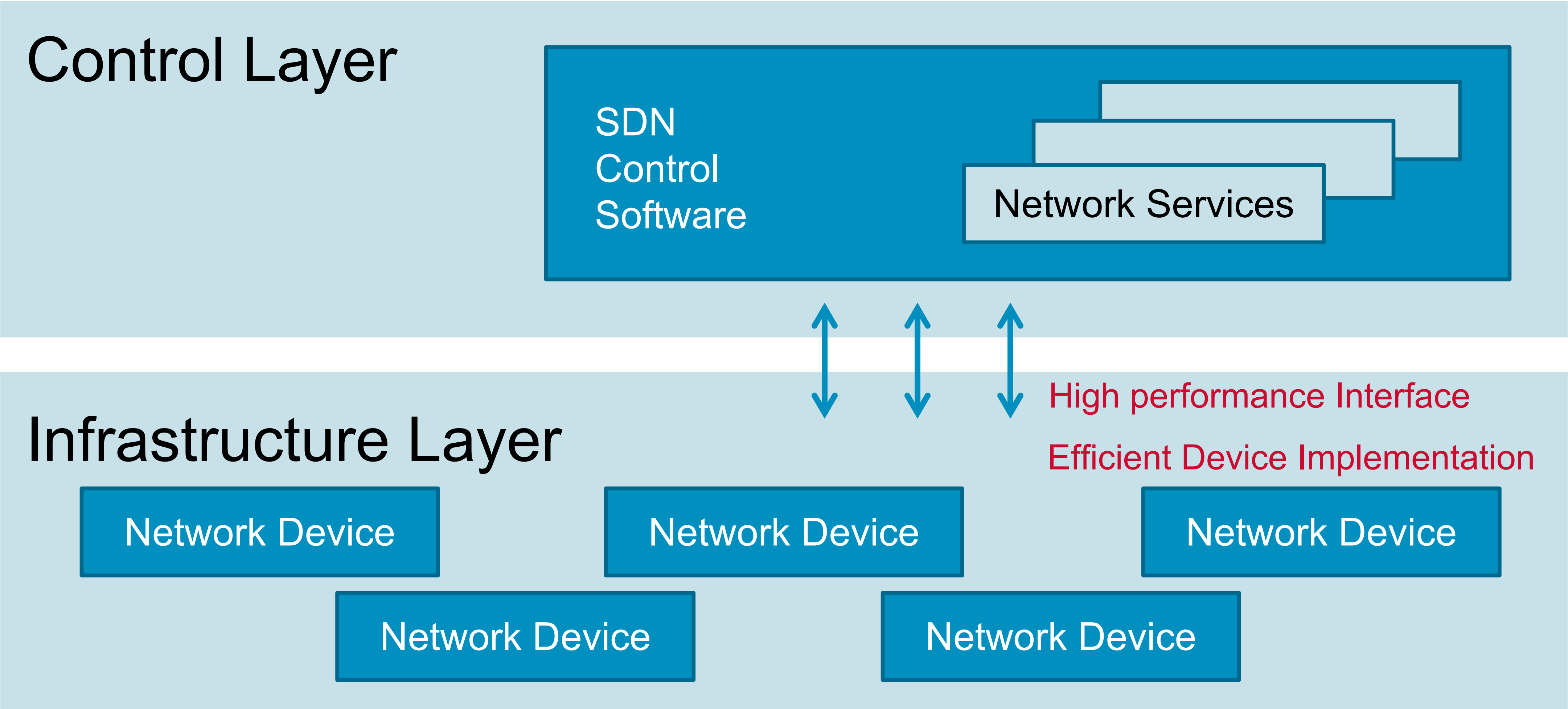
BROADCOM®

Agenda

- SDN based Network Model
- Efficient device interface
- Quick Introduction to NPL
- Efficient pipeline design
- Q&A

SDN Network model

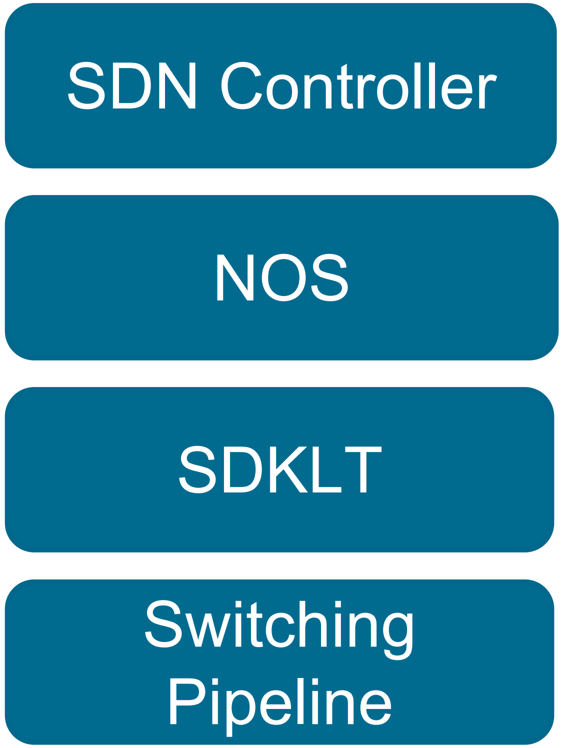
SDN Layers



Courtesy SDx Central

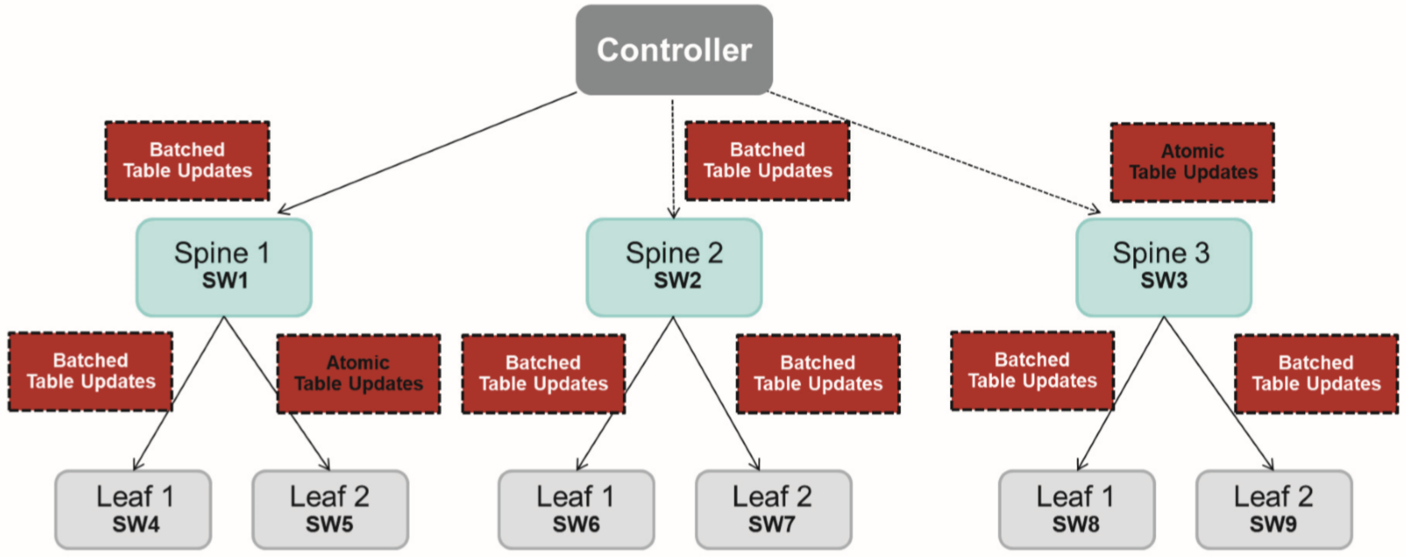
Efficient Network Stack

- Architect the System instead of components
- Layered architecture to scale
- Efficient design at each layer
- Align the interfaces and abstractions across the layers

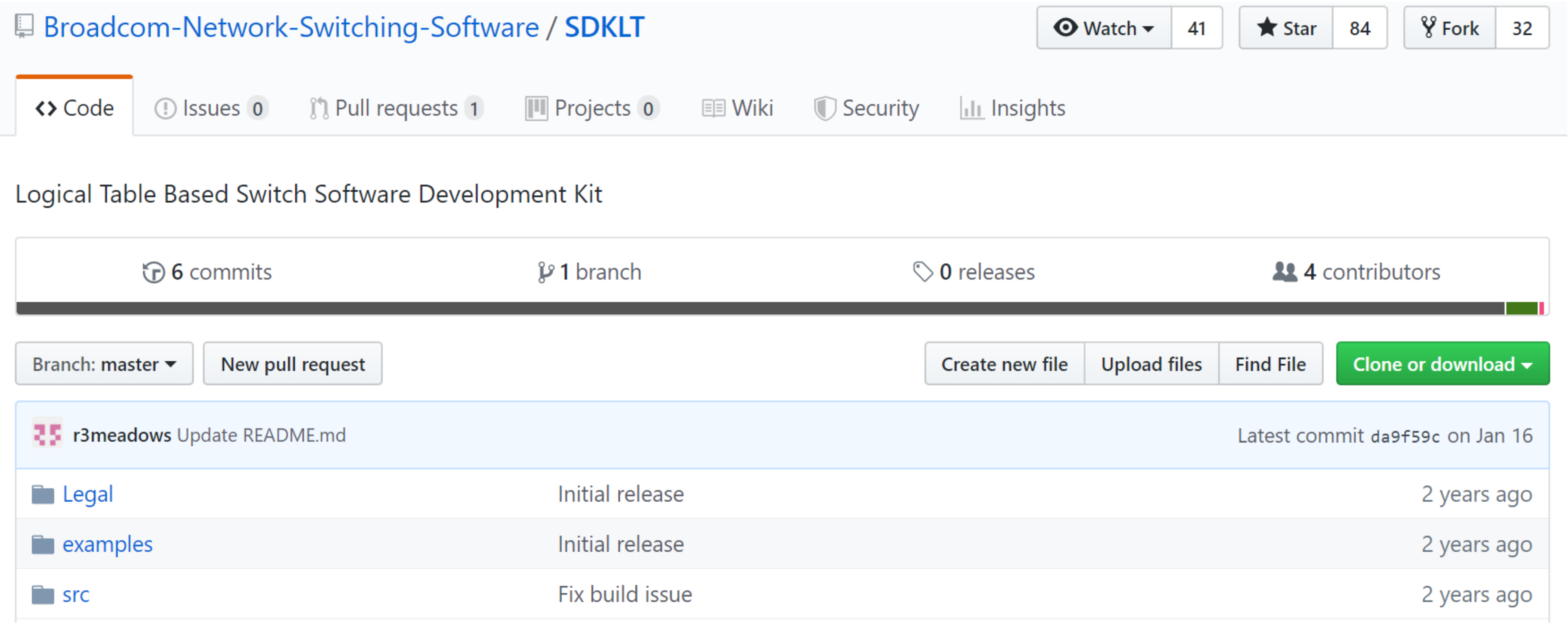


SDN Controllers

- Have a larger foot print and impact radius
- Have to manage a large number of heterogeneous devices
- Need Device Data reliably and quickly to assess network state
- Need quick Device re-config to steer traffic to avoid area of fault
- Need high performance SDN Controllers



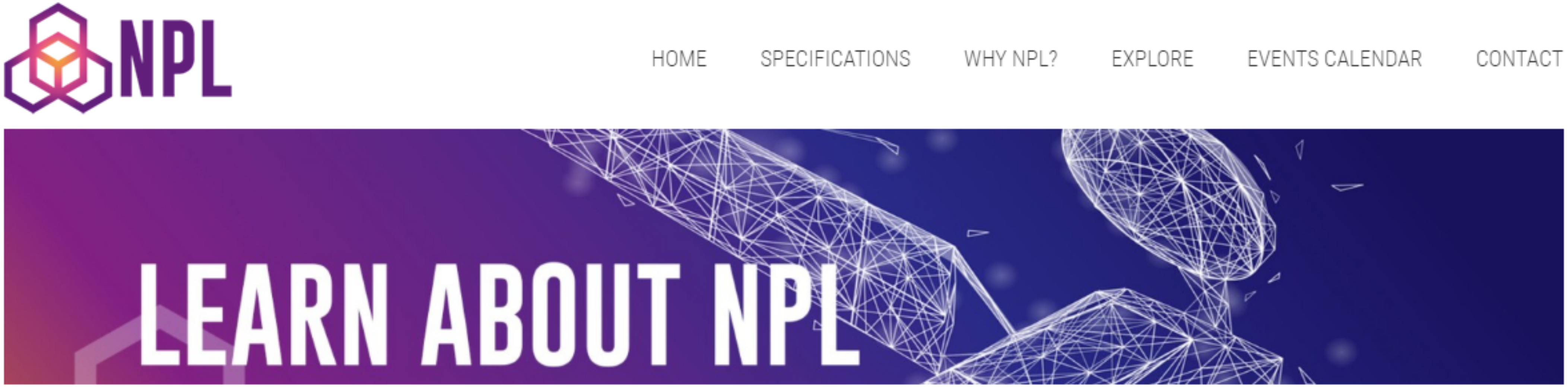
SDKLT



Git Hub : https://github.com/Broadcom-Network-Switching-Software/SDKLT

Product Brief and White Paper “Benefits of Logical Table APIs in DC” : https://www.broadcom.com/products/ethernet-connectivity/software/sdklt

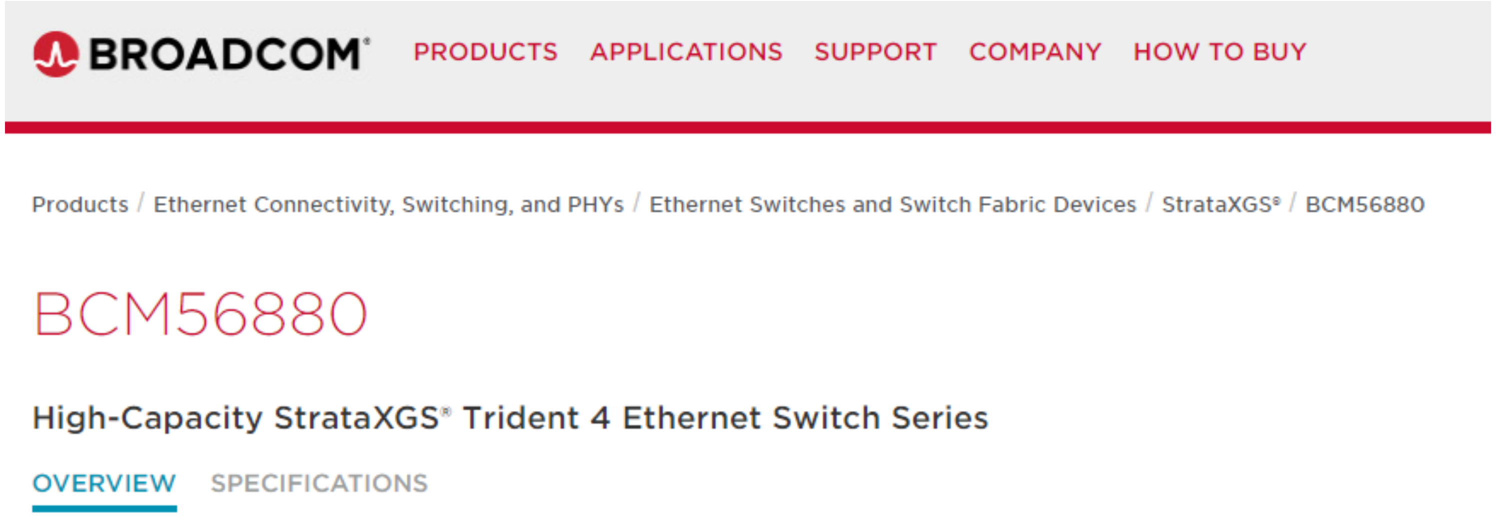
Network Programming Language(NPL)

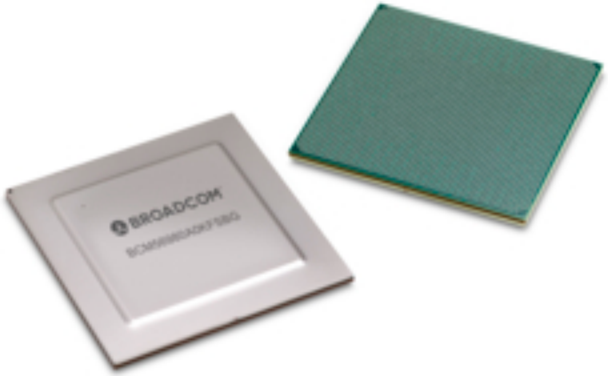


- www.NPLang.org
- NPL 1.3 Language Specification
- Frontend compiler
- Behavioral Model Generator
- Example programs (L2_switch, L3_App) and Tutorials

BROADCOM

Trident 4, Tomahawk





- TD4 - https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56880-series
- TH - https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56960-series
- Architecture, Features, Applications

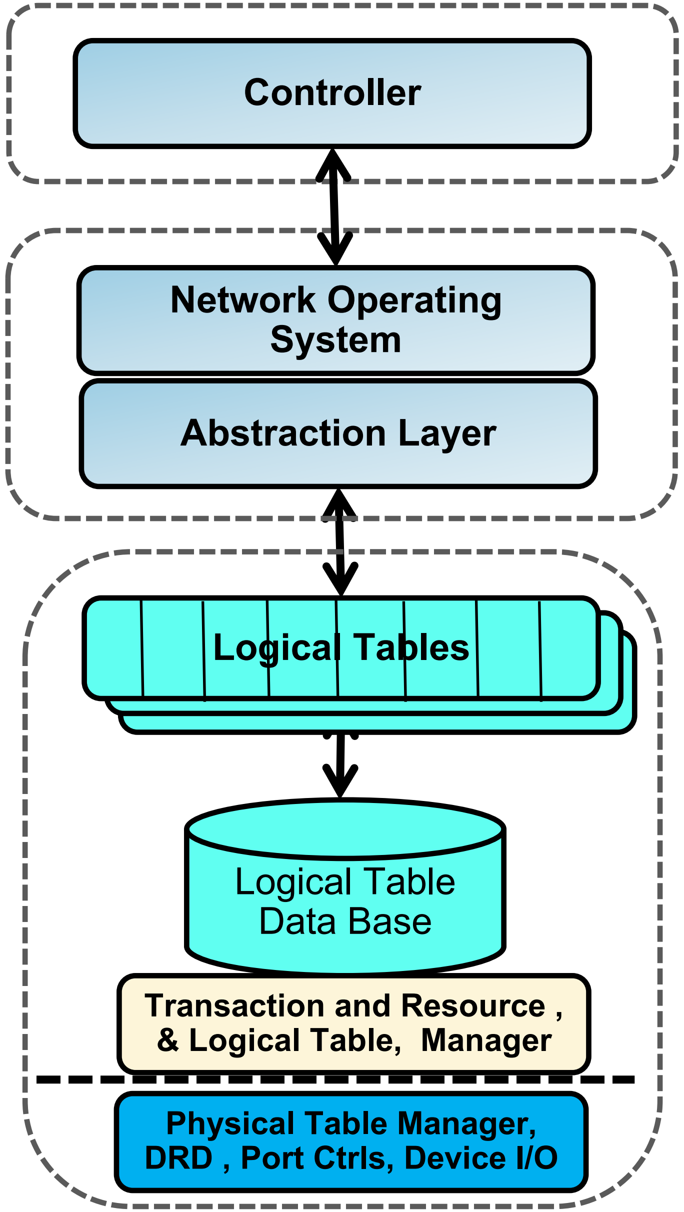
BROADCOM

High performance Interface

Logical Table SDK (SDKLT)

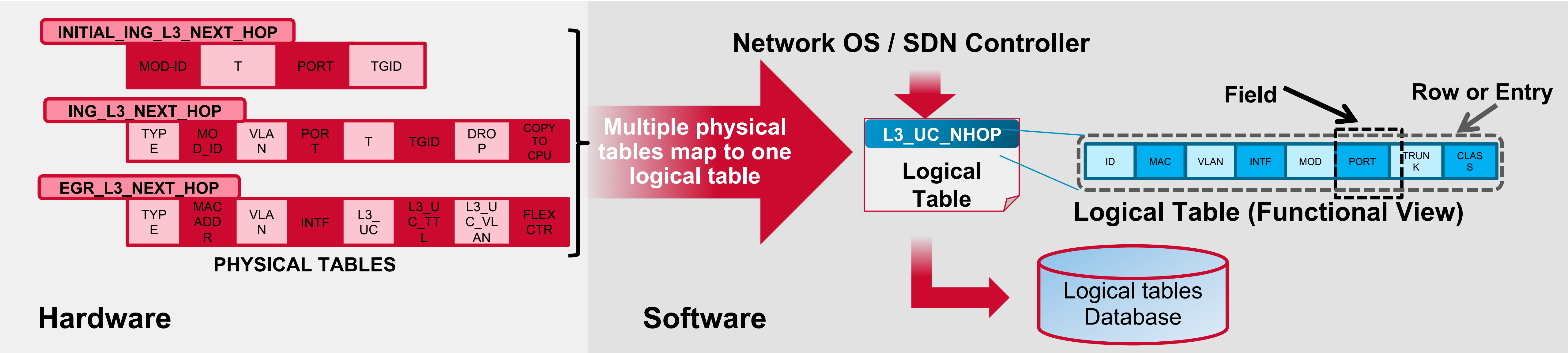
What is SDKLT?

- Data Driven programming model
  - New approach to SDK where the Logical Tables APIs enable table-driven chip programming
  - Simple table-programming APIs write into logical tables, data base
  - Logical Tables (LT) Map into physical tables on silicon
  - SDK 6 is, in comparison, is focused on traditional action-oriented APIs
- Logical table (LT) APIs
  - Structured and easy to use
  - Direct control over functions
  - Direct control of devices
  - Support Async and Sync operations
  - Support Atomic and Simple transactions



BROADCOM

Logical Table and Physical Tables



- A Logical Table provides a user's view of device feature or functionality (except Packet IO)
  - Pipeline (L2, VLAN, IPMC, ECMP, etc.), Port/PHY (topology, configuration, status, etc.), MMU,..
- The physical table controls device behavior for a specific function
- Physical Table is one to one representation of a table and register in chip hardware
- Add/Commit, Lookup, Update, Delete and Traverse

High Performance Interface

- Data Driven
- Efficiency
- High Performance
- High Availability
- Quality
- Ease of Development
- Visibility
- Simplicity

High Performance Interface

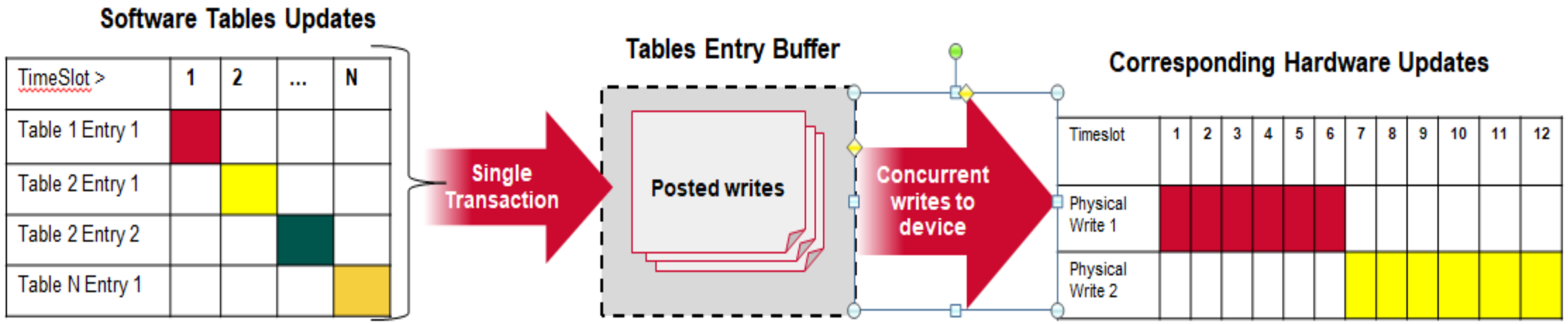
- Data Driven
- Efficiency
- High Performance
- High Availability
- Quality
- Ease of Development
- Visibility
- Simplicity

- Logical Table driven programming
- Clean separation of code and data
- Insert/Commit, Lookup, Update, Delete and traverse
- APIs to operate on Table, Entry, Field or a transaction

High Performance Interface

- Data Driven
- Efficiency
- High Performance
- High Availability
- Quality
- Ease of Development
- Visibility
- Simplicity

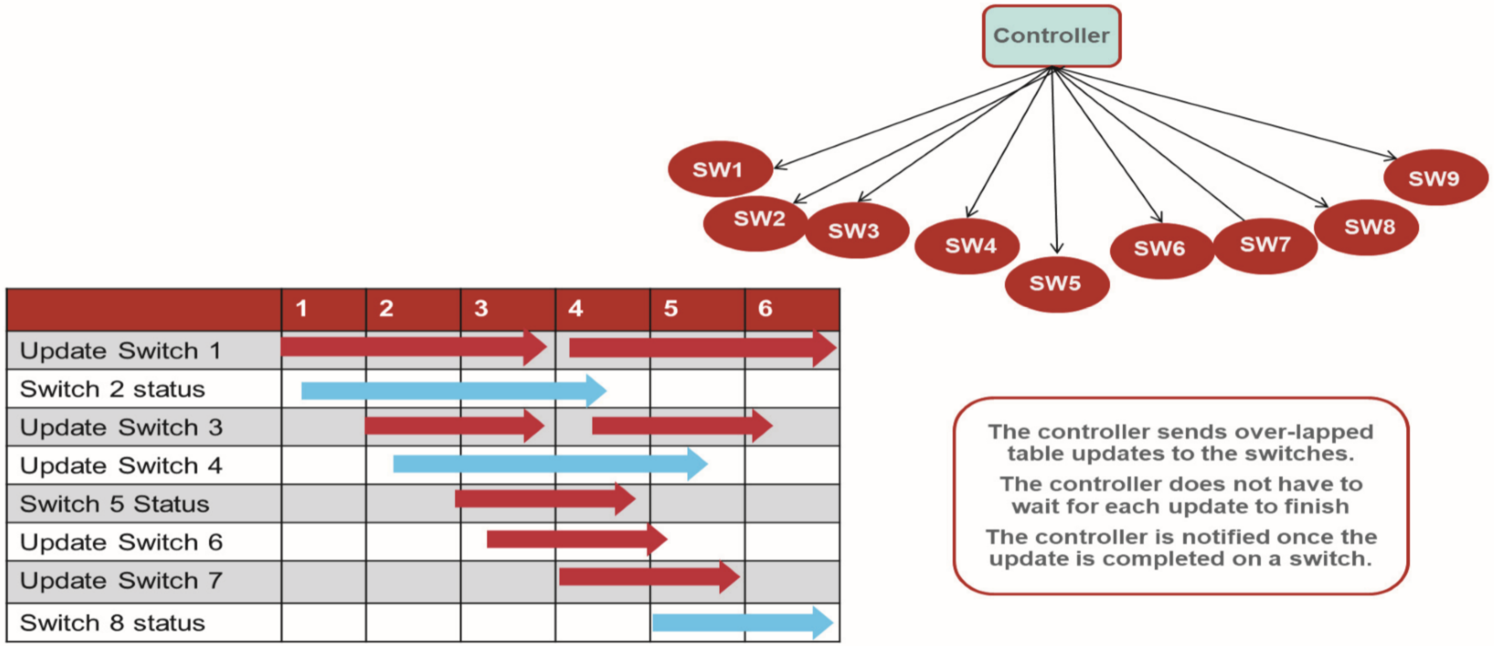
- Bulk read/write to quickly load new layout
- KNET, DMA based packet IO for fast turn around for control plane request/response
- Combine multiple operations in to transactions
- Software modeled tables for faster writes



High Performance Interface

- Data Driven
- Efficiency
- High Performance
- High Availability
- Quality
- Ease of Development
- Visibility
- Simplicity

- ASYNC operations allow multiple updates to multiple devices in parallel



High Performance Interface

- Data Driven
- Efficiency
- High Performance
- **High Availability**
- Quality
- Ease of Development
- Visibility
- Simplicity

- Committed and acknowledged transactions can be recovered through HA Database
- Play-by-play recording and replay
- Warmboot and ISSU
- Rollback support for Atomic transactions

High Performance Interface

- Data Driven
- Efficiency
- High Performance
- High Availability
- **Quality**
- Ease of Development
- Visibility
- Simplicity

- Auto generated code, tooling and documentation
- Automated framework for functionality and performance validation
- Simple and consistent APIs for observability
- Action reply to help debug

High Performance Interface

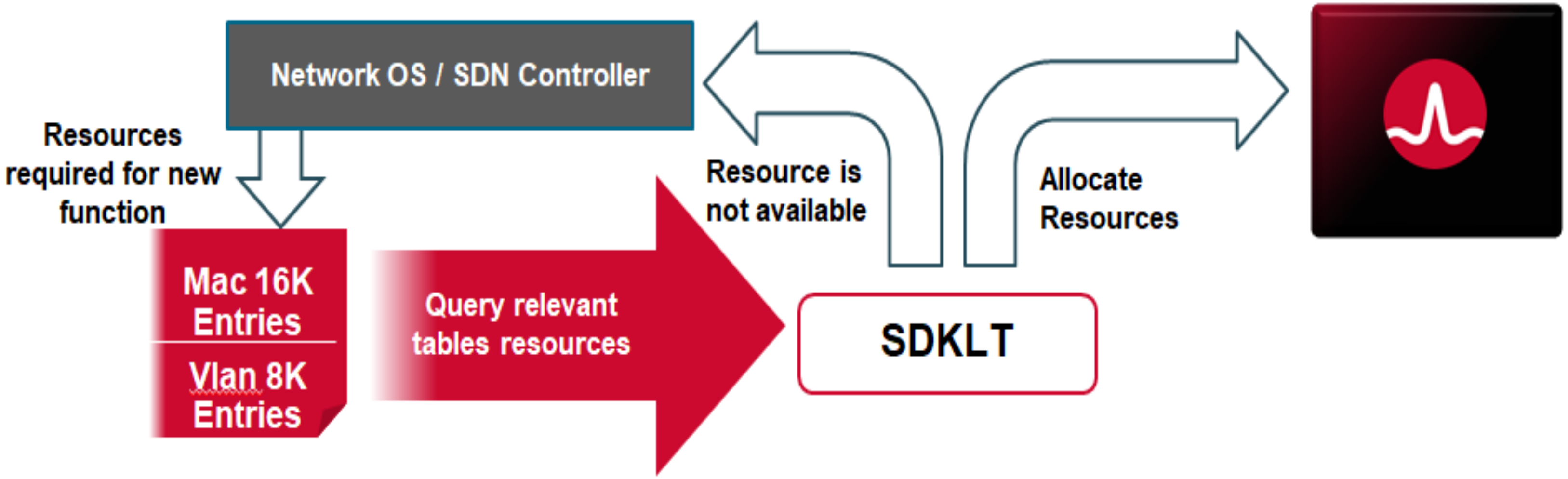
- Data Driven
- Efficiency
- High Performance
- High Availability
- Quality
- Ease of Development
- Visibility
- Simplicity

- Diag Shell access to Logical and Physical Tables
- C-Interpreter to enable script development

BROADCOM

High Performance Interface

- Data Driven
- Efficiency
- High Performance
- High Availability
- Quality
- Ease of Development
- **Visibility**
- Simplicity



- Visibility in to resource utilization
- Resource reservation, monitor and control
- LT view simplifies use of resources

BROADCOM

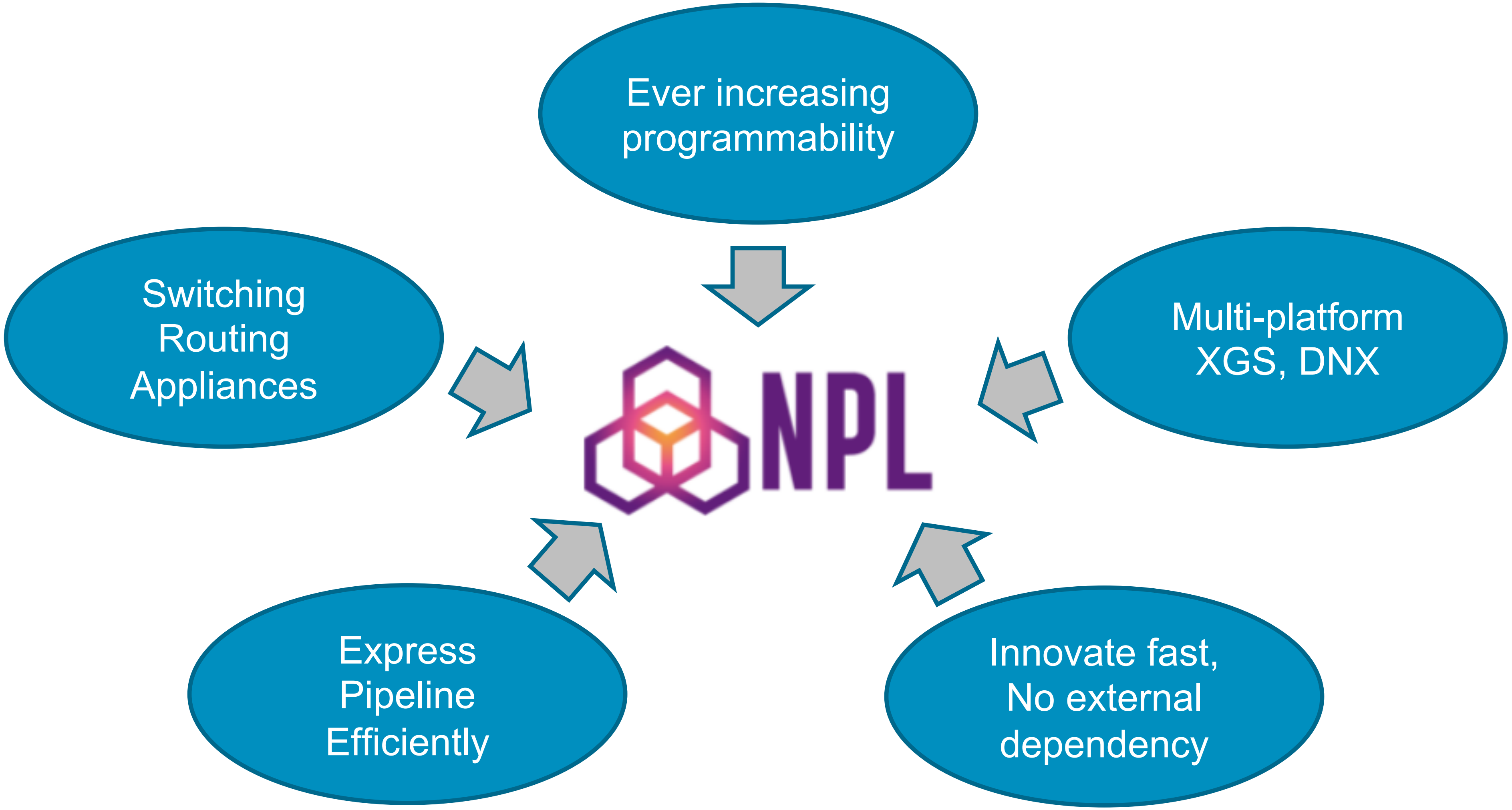
High Performance Interface

- Data Driven
- Efficiency
- High Performance
- High Availability
- Quality
- Ease of Development
- Visibility
- Simplicity

- Few consistent set of APIs
- Hides device idiosyncrasies

Network Programming Language (NPL)

NPL Motivation



NPL Program

```
program l3_app ()   {
    memory_init();
    parse_begin (start);
    port_table.lookup (0);
    parse_continue(ethernet);
    do_vid_assign();
    vlan_table.lookup (0);
    if (ing_pkt.l2_grp.l2._PRESENT) {
        mac_table.lookup(0);
        mac_table.lookup(1);
    }
    if (cmd_bus.l3_enable) {
        do_l3_forwarding();
        do_packet_edits();
        do_checksum_update();
    }
}
```

Start Parsing

Lookup

Continue parsing

Multi-Lookup

Switching Logic

NPL Data Types

```
bit       cfi;       // specify single bit
bit[12]   vid;       // specify 12 bit vid
varbit[64] options;    // up to 64b wide

struct vlan_t {      // Header definition
    fields {
        bit[16]  tci;
        bit[16]  ethertype;
    }
    overlays {
        pcp : tci[15:13];
        dei : tci[12:12];
        vid : tci[11:0];
        full_tag : tci <> ethertype;
    }
}
```

```
struct l2_group_t {    // Header Group
    fields {
        l2_t    l2;
        vlant_t vlan;
    }
}
struct l3_packet_t {          // Packet
    fields {
        l2_group_t l2_grp;
        l3_group_t l3_grp;
    }
}
packet l3_packet_t ingress_pkt;

bus cmd_bus_t cmd_bus;  // Global Bus
```


Logical Table Example

Define a TCAM table

```
logical_table my_station_hit
{
     table_type : tcam;
     maxsize : 512;
     minsize : 512;
     keys {
         bit[48] macda;
         bit[12] vid;
         bit[8]  src_modid;
     }
     fields {
         bit[2] mpls_tunnel_type;
         bit    local_l3_host;
     }
     key_construct() {
         macda     = ing_pkt.l2_grp.l2.macda;
         vid       = obj_bus.vlan_id;
         src_modid = obj_bus.source_logical_port;
     }
     fields_assign() {
         l3_cmd_bus.local_l3_host = local_l3_host;
         . . .
     }
}
```

Multiple Lookups of the same logical Table

```
key_construct() {
    if (_LOOKUP0==1) {
         macda = ing_pkt.l2_grp.l2.da;
    }
    if (_LOOKUP1==1) {
        macsa = ing_pkt.l2_grp.l2.sa;
    }
}

fields_assign() {
    if (_LOOKUP0==1) { //e.g. Entry 100
        obj_bus.dst = port;
        obj_bus.dst_discard = dst_discard;
    }
    if (_LOOKUP1==1) { //e.g. Entry 200
        temp_bus.src_port = port;
        obj_bus.src_discard = src_discard;
    }
}
```


Function Examples

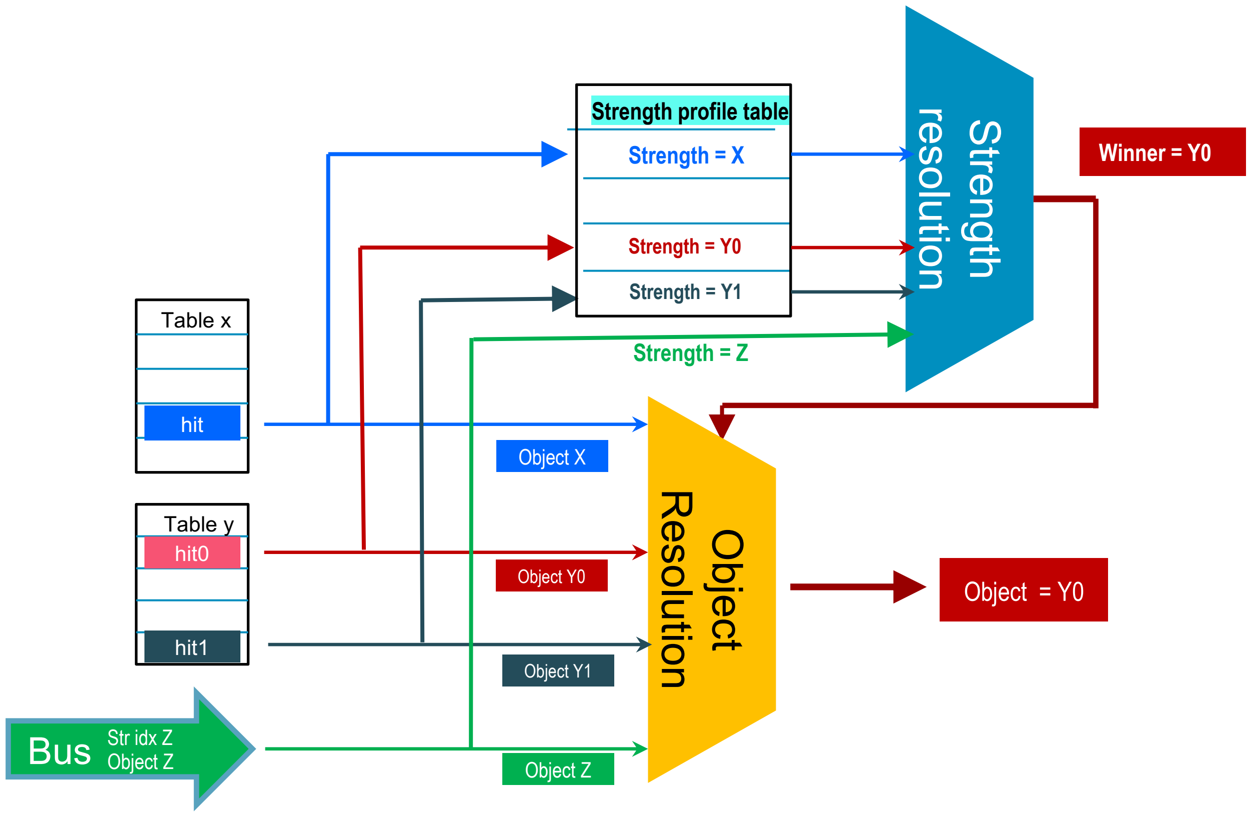
Example 1 – Simple VLAN Assignment

```
function do_vid_assign(){
    // Input is packet header
    if (ing_pkt.vlan._PRESENT) {
        // Output is global bus field
        obj_bus.vid = ing_pkt.vlan.vid;
    }
}
```

```
function do_l3_forwarding()
{
    local_var.no_l3_switch = 0;
    cmd_bus.l3_routable = 0;

    if (cmd_bus.do_l3_lookup &
        cmd_bus.my_stn_routing_enable) {
        if ((ingress_pkt.ipv4.ttl == 0) &&
            (obj_bus.local_address == 0)) {
            local_var.no_l3_switch = 1;
        }
        if ((ingress_pkt.ipv4.ttl == 1) &&
            (obj_bus.local_address == 0)) {
            local_var.no_l3_switch = 1;
        }
        if (ingress_pkt.ipv4.option != 0) {
            local_var.no_l3_switch = 1;
        }
        if (local_var.no_l3_switch == 0) {
            // Output to global bus
            cmd_bus.l3_routable = 1;
        }
    }
}
```


Strength Resolution Construct




```
/*
 * Strength resolution function
 *   There are 3 strength sources for strength resoluion:
 *     a. local_bus as default cos
 *     b. pri_cos_mapping_table for tagged packets
 *     c. dscp_cos_mapping_table for ipv4 packets
 *   Each source provide strength_index into cos_profile_table
 *   Entries of cos_profile_table is used in strength resolution
 */

strength_resolve( local_bus.cos,            // field taking the assignment
        local_bus.cos_strength,             // strength for value on bus
        { pri_cos_mapping_table._LOOKUP0, NULL, cos_strength_profile_table.cos_strength, pri_cos_mapping_table.cos},
        { dscp_cos_mapping_table._LOOKUP0, NULL, cos_strength_profile_table.cos_strength, dscp_cos_mapping_table.cos});
```


BROADCOM

Efficient Pipeline Design

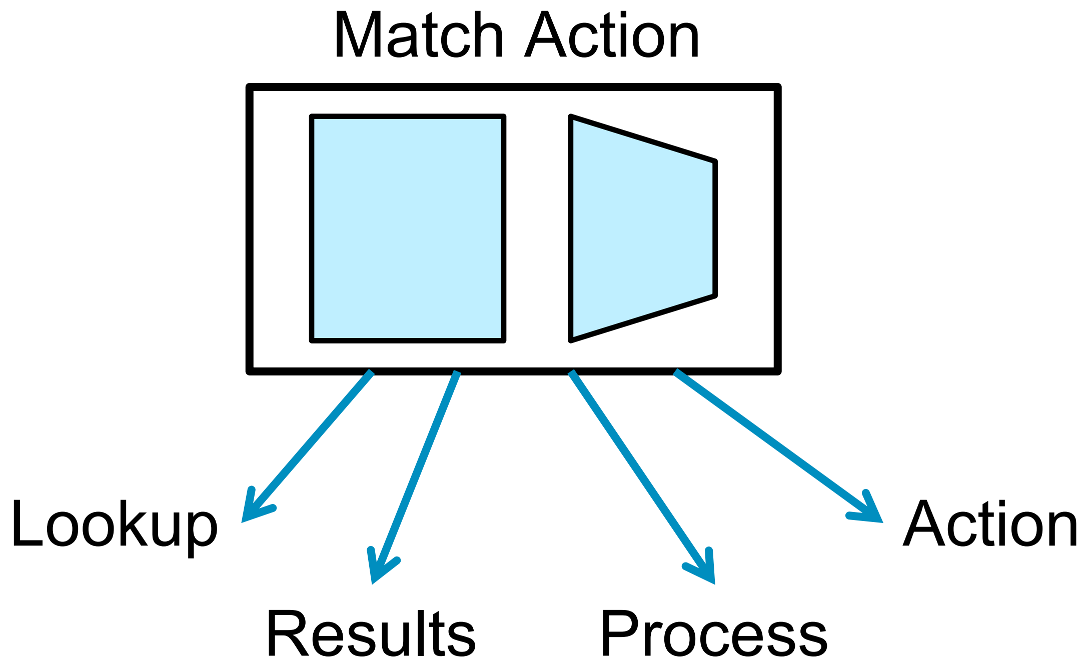
Efficient Packet Processing

- Throughput
- Capacity
- Concurrency
- Low Latency
- Efficiency
- Visibility
- Runtime
- Simplicity

Efficient Packet Processing

- Throughput
- Capacity
- Concurrency
- Low Latency
- Efficiency
- Visibility
- Runtime
- Simplicity

- Light weight processing stages
- Separation of Match from Actions



Efficient Packet Processing

- Throughput
- Capacity
- Concurrency
- Low Latency
- Efficiency
- Visibility
- Runtime
- Simplicity

- Multi-Lookup Tables
- Switch Dst MAC, Learn Dst MAC
- Route Dst IP, RPF check Src IP

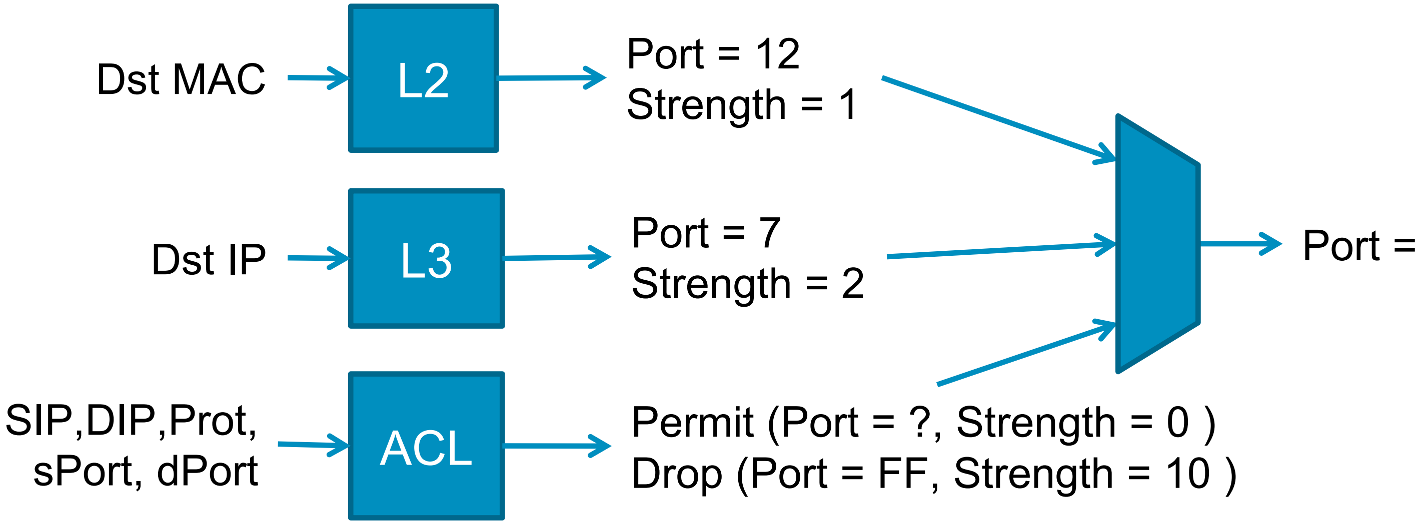
```
if (ing_pkt.l2_grp.l2._PRESENT) {
    mac_table.lookup(0);
    mac_table.lookup(1);
}
```


BROADCOM

Efficient Packet Processing

- Throughput
- Capacity
- **Concurrency**
- Low Latency
- Efficiency
- Visibility
- Runtime
- Simplicity

- Strength based dependency resolution



Efficient Packet Processing

- Throughput
- Capacity
- Concurrency
- **Low Latency**
- Efficiency
- Visibility
- Runtime
- Simplicity

- Drive Parallelism
- Hardware constructs for parallelism
- Speculative Lookups

Efficient Packet Processing

- Throughput
- Capacity
- Concurrency
- Low Latency
- **Efficiency**
- Visibility
- Runtime
- Simplicity

- Efficient resource utilization
- Special Functions
- Usage Modes

Efficient Packet Processing

- Throughput
- Capacity
- Concurrency
- Low Latency
- Efficiency
- Visibility
- Runtime
- Simplicity

- Instrumentation
- Trace points
- Counters

Efficient Packet Processing

- Throughput
- Capacity
- Concurrency
- Low Latency
- Efficiency
- Visibility
- Runtime
- Simplicity

- Re-prioritization
- Runtime selection of in/outputs, Mode
- Dynamic Tables

BROADCOM

Efficient Packet Processing

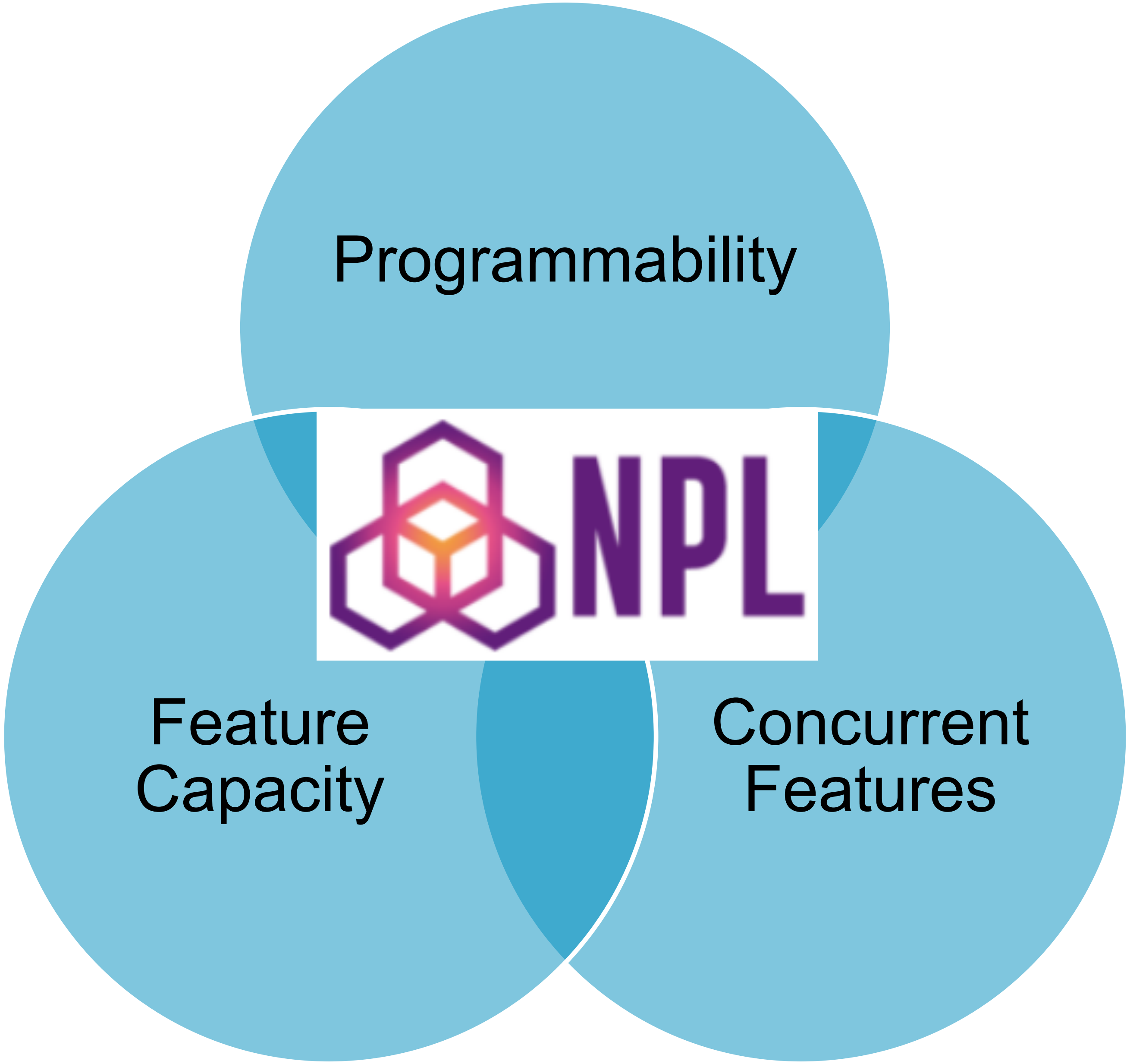
- Throughput
- Capacity
- Concurrency
- Low Latency
- Efficiency
- Visibility
- Runtime
- Simplicity

- Native support of switching logic

BROADCOM

NPL Design decisions

- Multi-Stage, re-entrant parser
- Separation of Lookups from Actions
- Multiple table types
- Design for parallelism
- Switching Logic is first class citizen
- Runtime re-configuration, re-prioritization, and linking
- Multi-platform support
- Support power users


Programmability
NPL
Feature Capacity
Concurrent Features

Thank You