Testing Compilers for Programmable Switches Through Switch Hardware Simulation

Michael D. Wong, Aatish Kishan Varma, Anirudh Sivaraman

Programmable Switches

- Switch programmability
  - ✓ Power
  - ✓ Cost
  - ✓ Performance

Programmable
Switching ASIC

BAREFOOT NETWORKS

CISCO

BROADCOM

Programming Languages

Lyra: A Cross-Platform Language and Compiler for Data Plane Programming on Heterogeneous ASICs

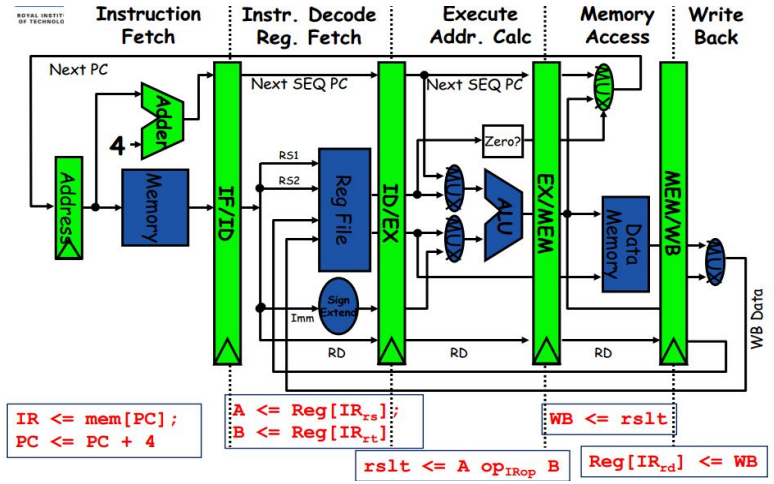
Jiaqi Gao[†§], Ennan Zhai[†], Hongqiang Harry Liu[†], Rui Miao[†], Yu Zhou[†◇], Bingchuan Tian[†★], Chen Sun[†]
Dennis Cai[†], Ming Zhang[†], Minlan Yu[§]
[†]*Alibaba Group* [§]*Harvard University* [◇]*Tsinghua University* [★]*Nanjing University*

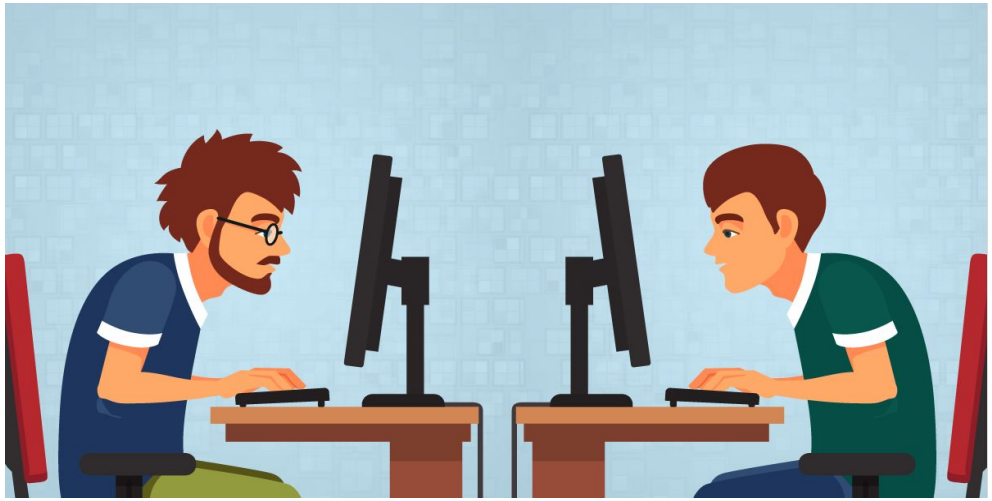
Packet Transactions: High-Level Programming for Line-Rate Switches

Anirudh Sivaraman[¶], Alvin Cheung[‡], Mihai Budiu[§*], Changhoon Kim[†], Mohammad Alizadeh[¶], Hari Balakrishnan[¶],
George Varghese[++], Nick McKeown[+], Steve Licking[†]
[¶]MIT CSAIL, [‡]University of Washington, [§]VMWare Research, [†]Barefoot Networks, [++]Microsoft Research, [+]Stanford University

Building Switch Compilers

- Compiler development in general is hard
  - Need knowledge of underlying hardware architecture
  - Requires significant engineering effort





- **Switch compiler development introduces new challenges**

Switch Compilation Challenges

- Limited hardware resources
  - Limited pipeline stages, memory, ALUs, etc.

- Pipeline architecture
  - Computations must be mapped to pipeline stages that respect dependencies

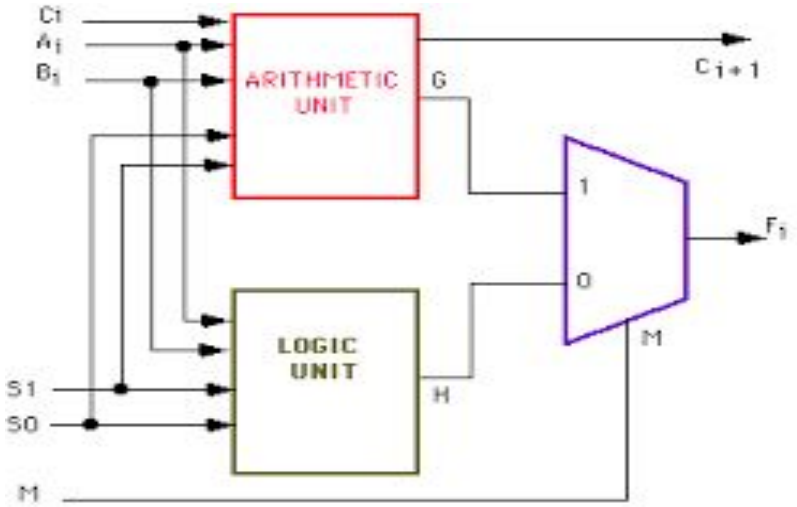
- All-or-nothing nature
  - A program runs at line-rate if it can be mapped to the pipeline or it is rejected

Druzhba

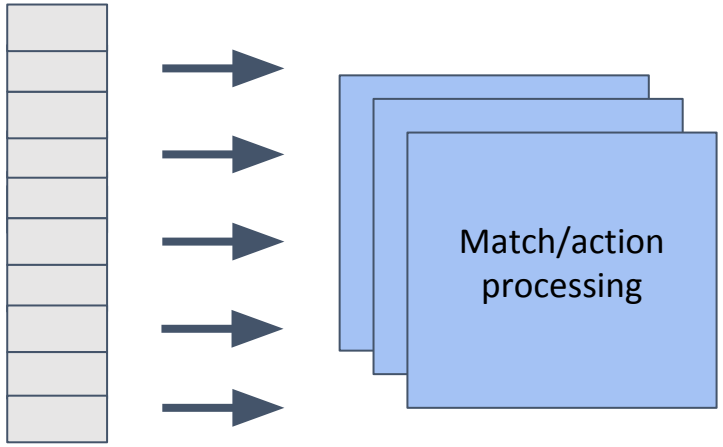
- Low-level switch pipeline simulator
- Executes compiler-generated machine code programs
- Compiler developers observe the correctness of compiler mappings

Druzhba Machine Model

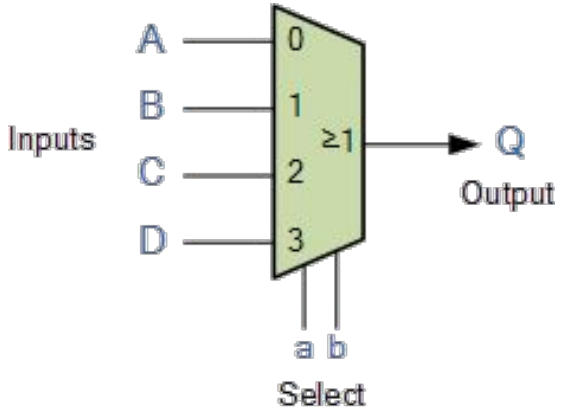
- Druzhba models the hardware details of the pipeline architecture
  - **Details of arithmetic logic units (ALUs)**
  - Packet header vectors (PHVs) instead of packets
  - Input and output multiplexers



Arithmetic logic unit (ALU)



Packet header vector (PHV)

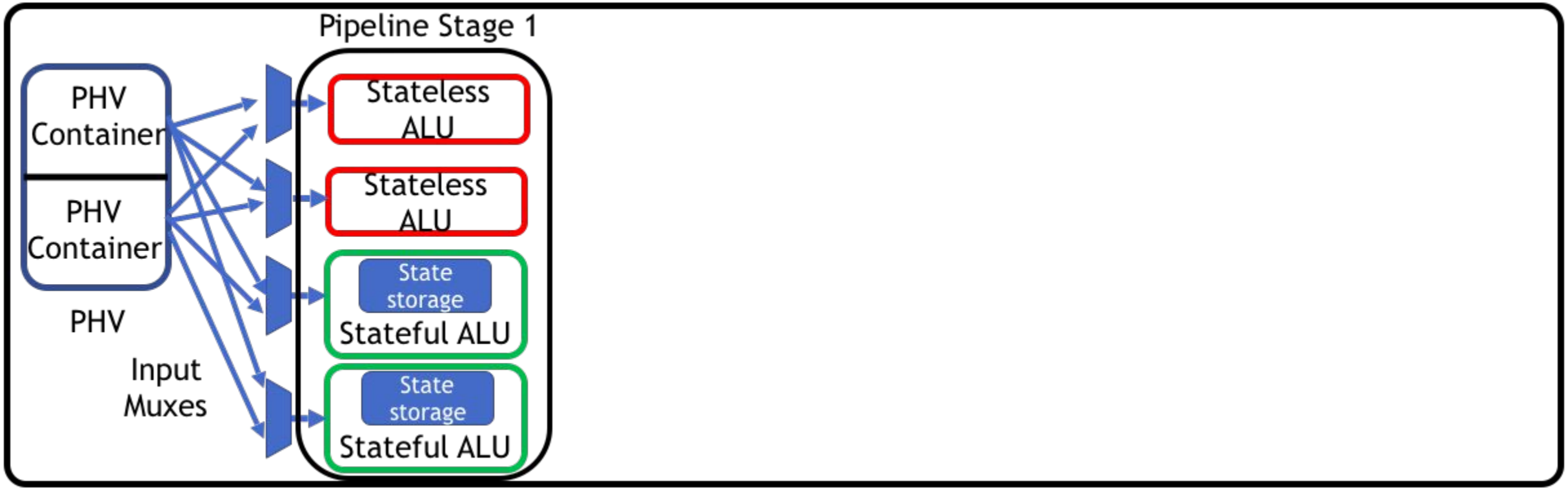


Muxes

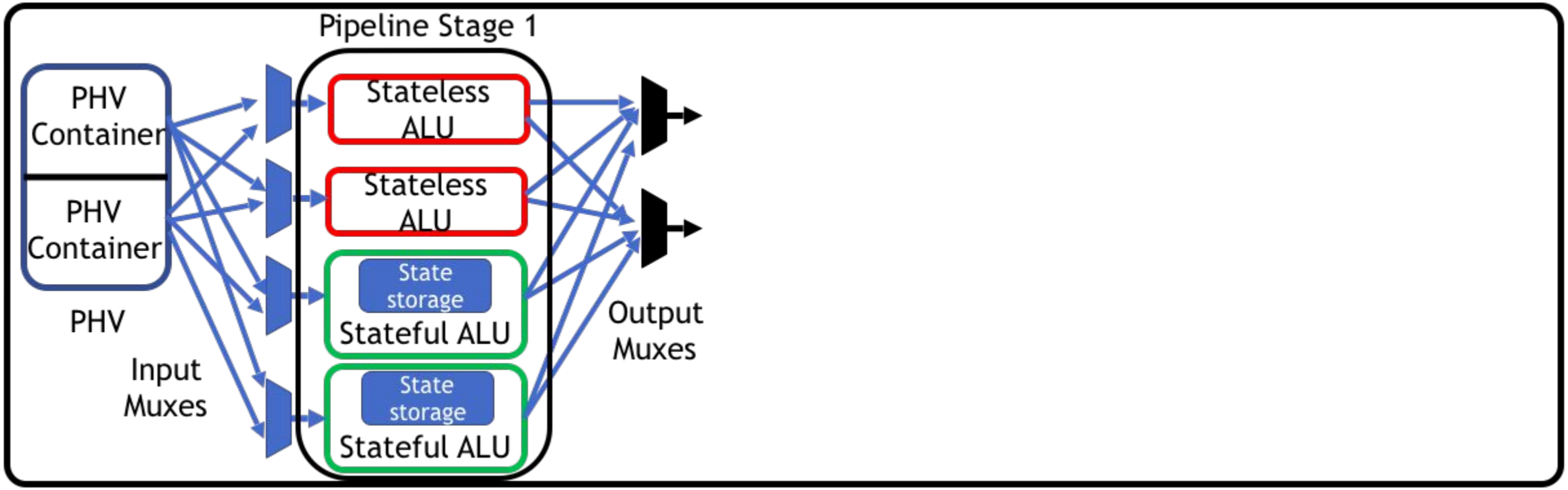
Druzhba Machine Model

Druzhba Machine Model

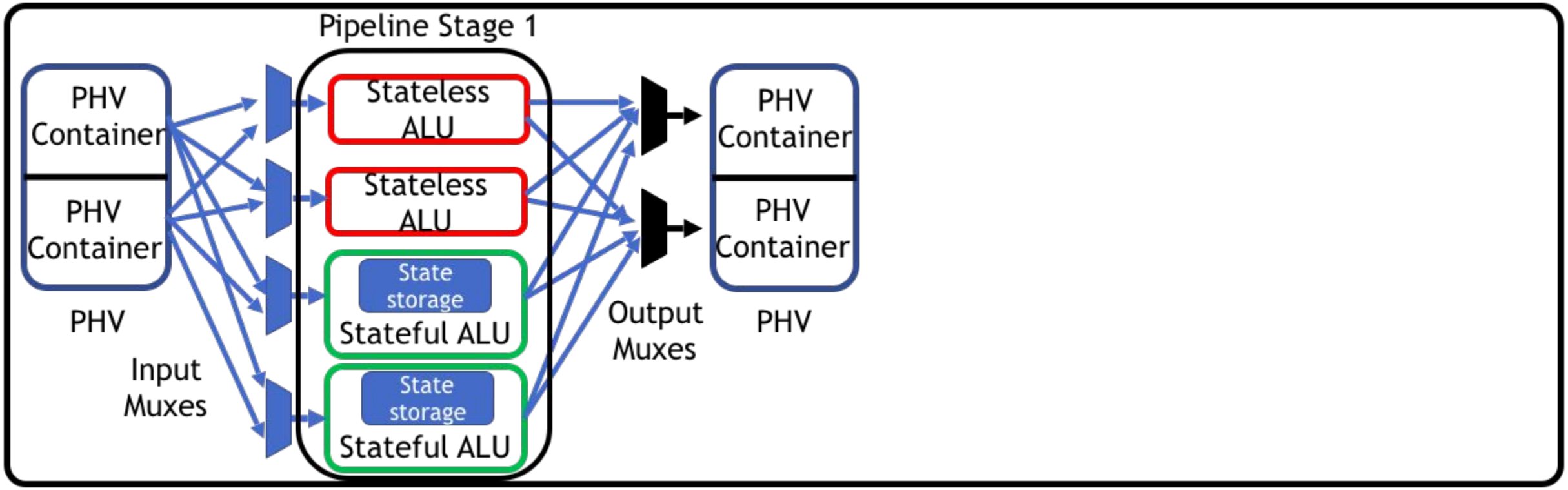
Druzhba Machine Model



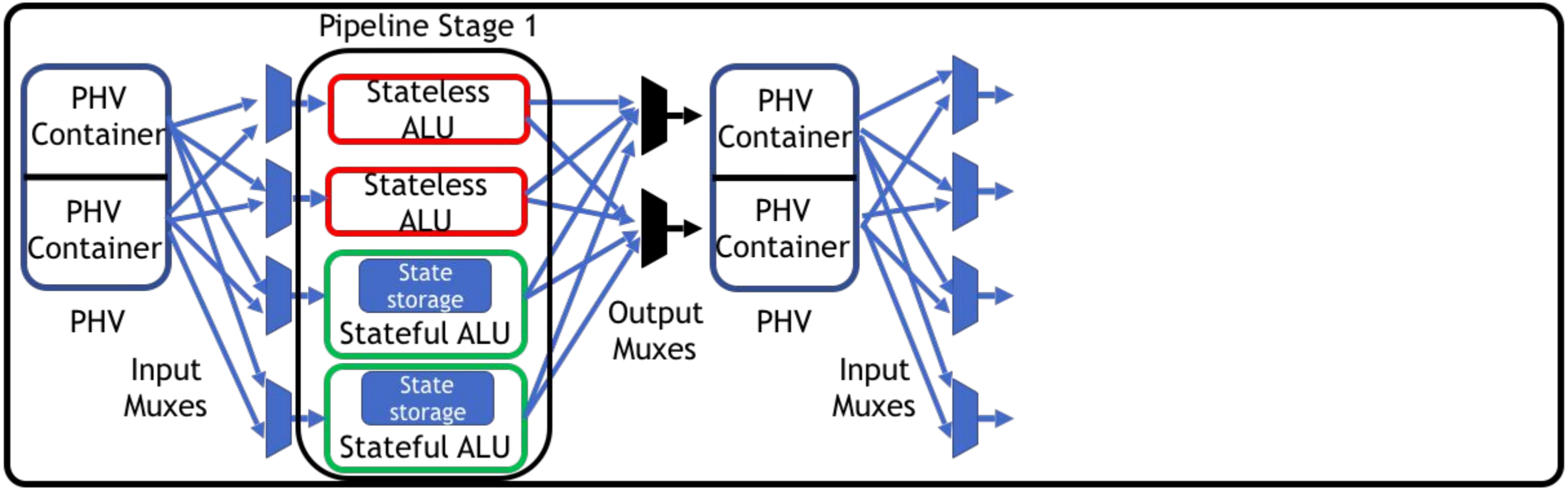
Druzhba Machine Model



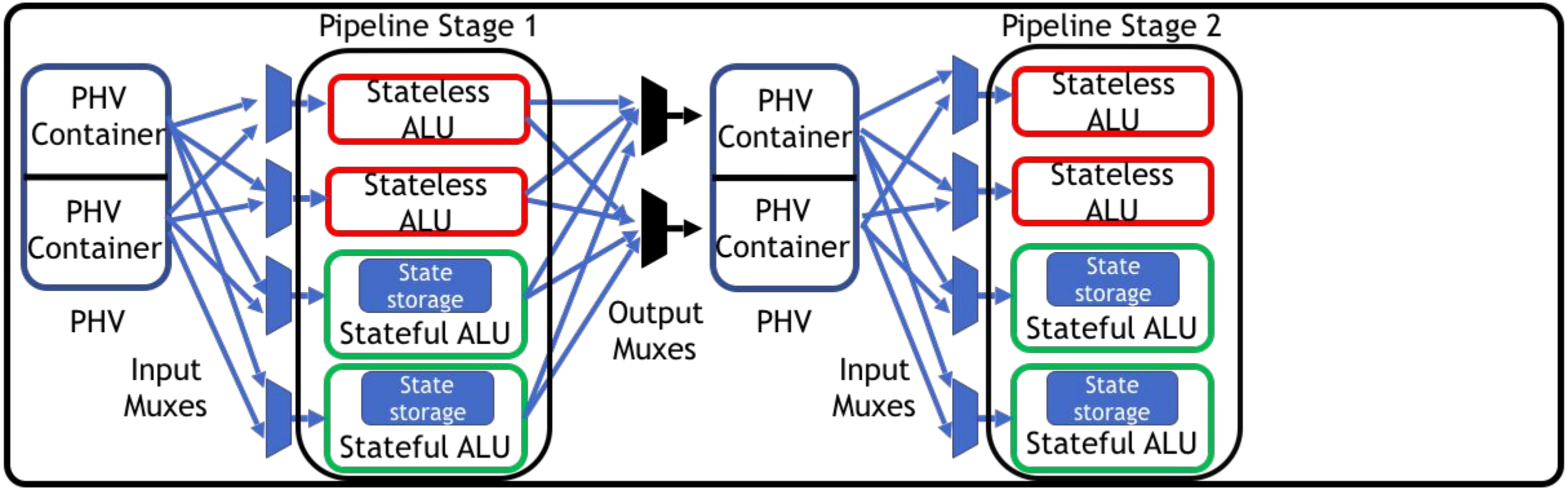
Druzhba Machine Model



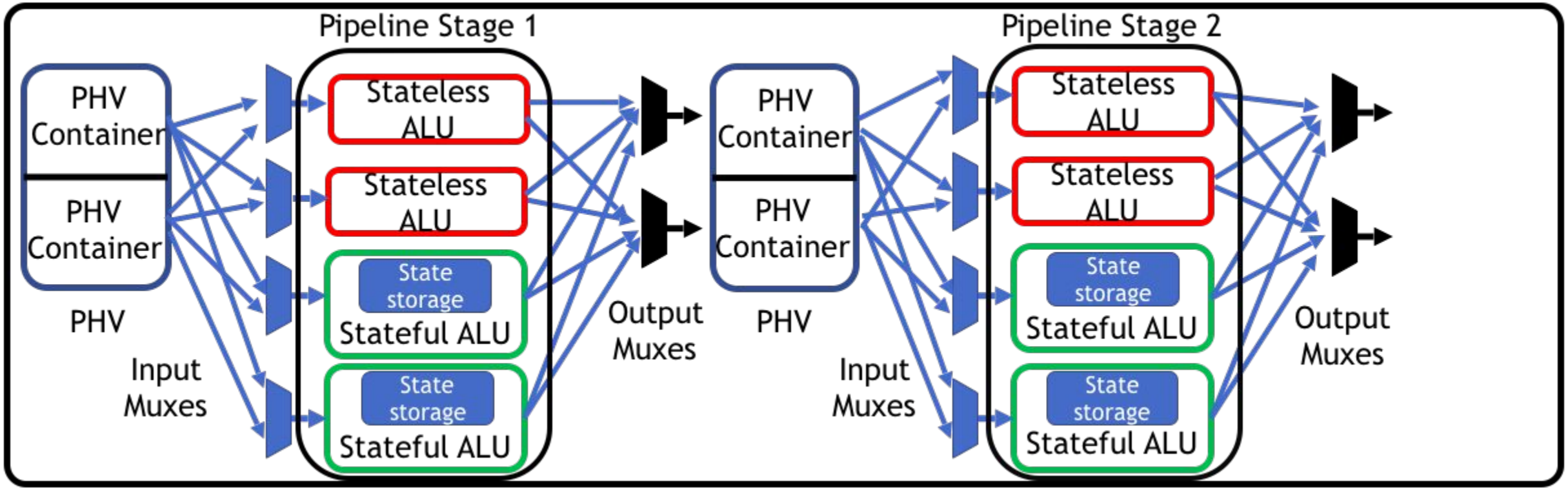
Druzhba Machine Model



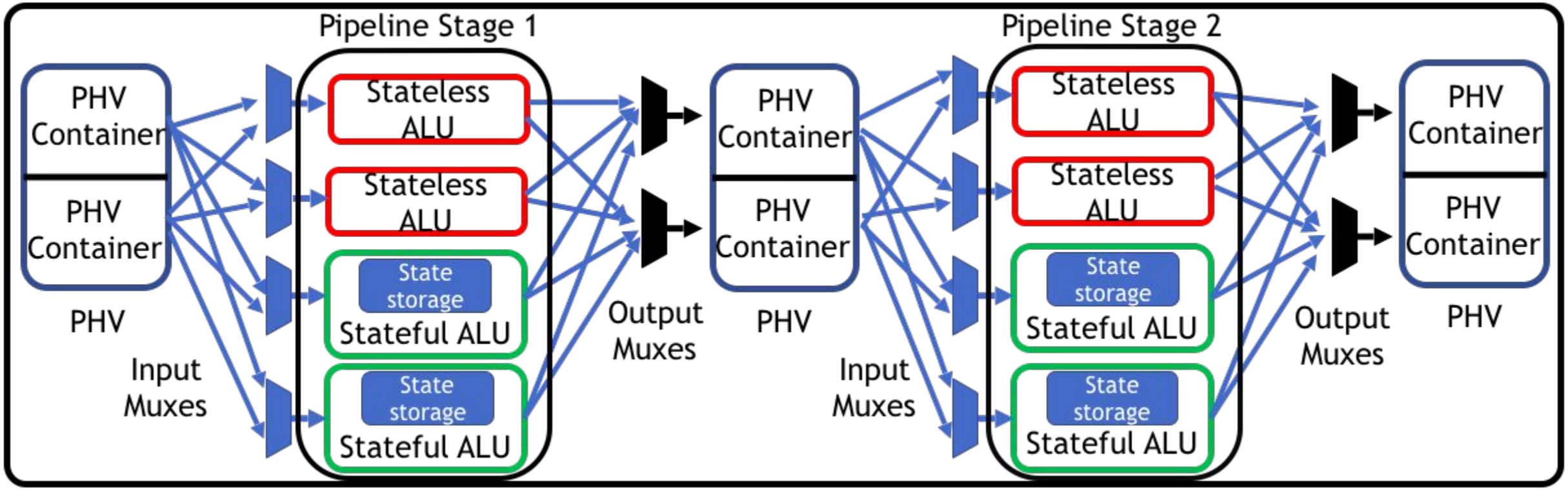
Druzhba Machine Model



Druzhba Machine Model



Druzhba Machine Model



Limitations: does not model matching, parsing, and scheduling

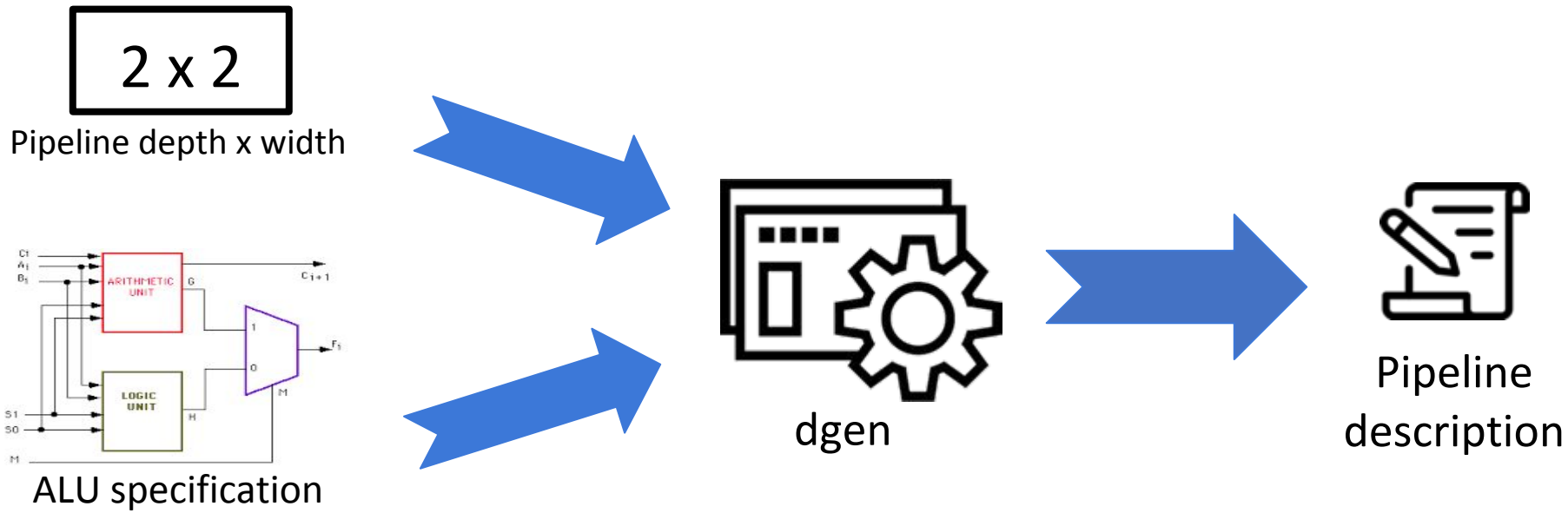
Druzhba Overview

- **dgen** pipeline generator
  - Generates simulation program to represent hardware details of the pipeline

- **dsim** pipeline simulator
  - Executes a machine code program using dgen's generated pipeline

Hardware Specification

- dgen uses the hardware specification to generate a file with the pipeline configuration details

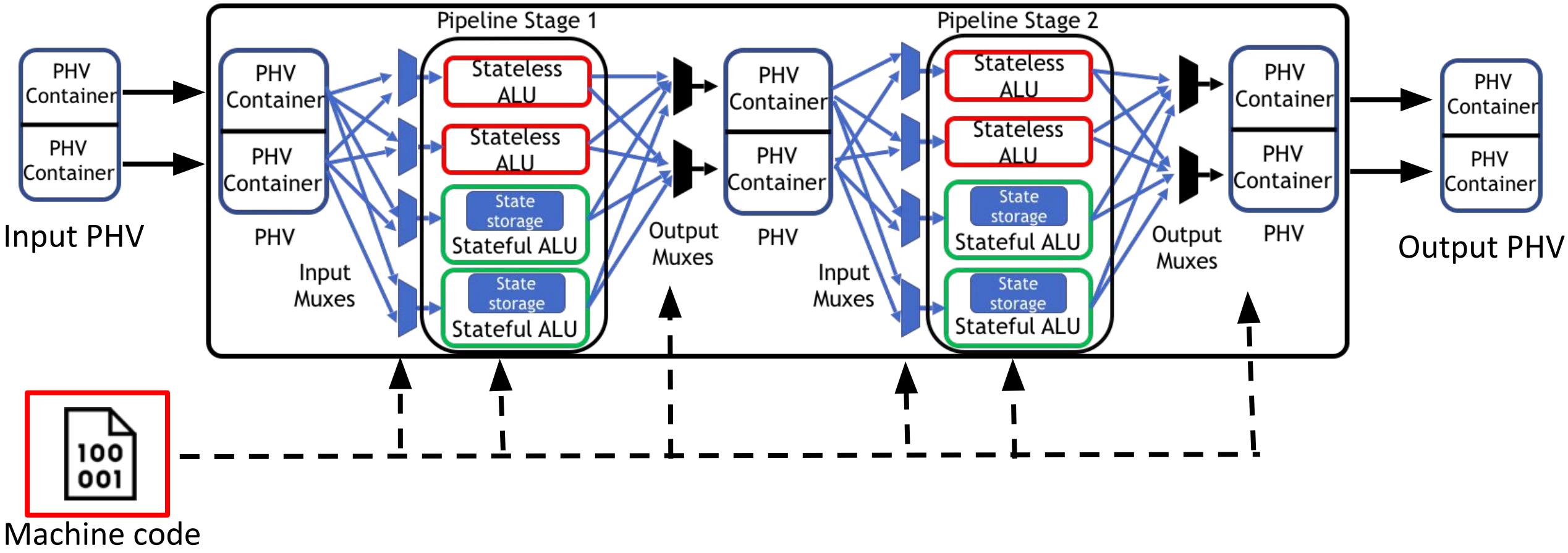


- Optimizations applied to the pipeline description to reduce overall simulation time

Hardware Simulation

- dsim simulates the hardware design specified in the pipeline description
- dsim takes in as input the machine code from the compiler
  - The machine code configures the behavior for ALUs and muxes
- A compiler is tested by fuzzing a compiler-generated program with random PHVs

Hardware Simulation



Compiler Testing Workflow

1. Provide transactional specification that captures intended behavior

2. Fuzz both the dsim pipeline and the transactional specification

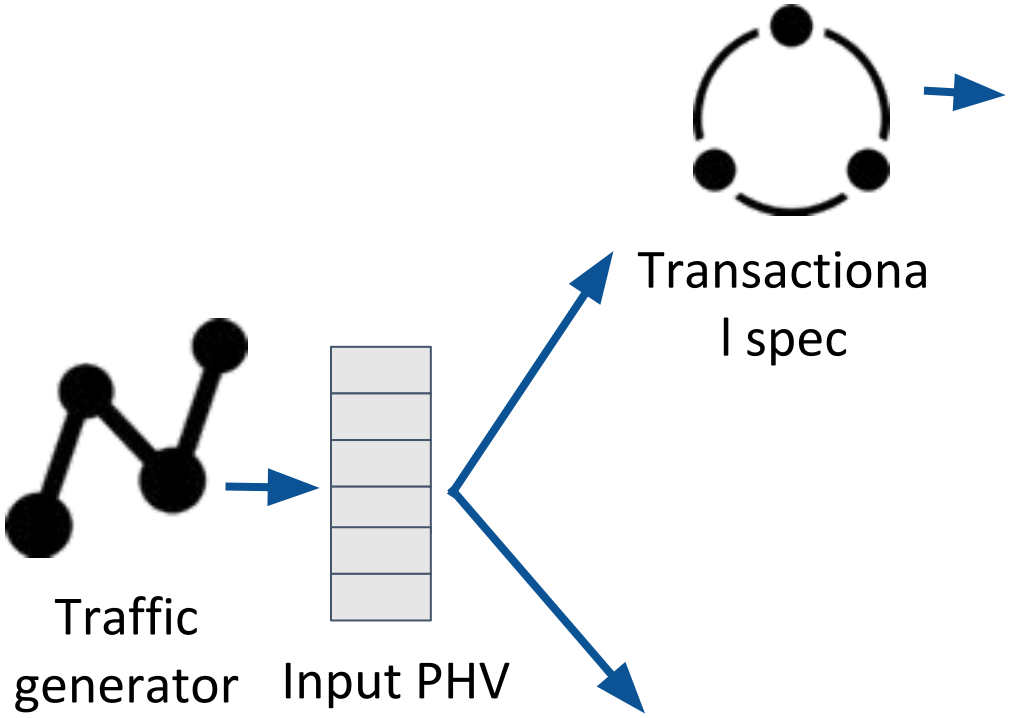
3. Check that the dsim pipeline and the transactional specification produce the same behavior

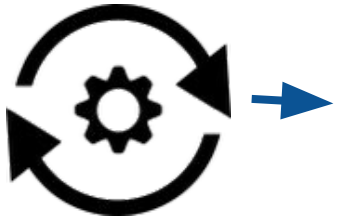
Correctness

- Correct compiler mapping
  - The output PHVs from both the dsim pipeline and the transactional specification are equal

- Erroneous compiler mapping
  - An input PHV yields two different results from the dsim pipeline and transactional specification

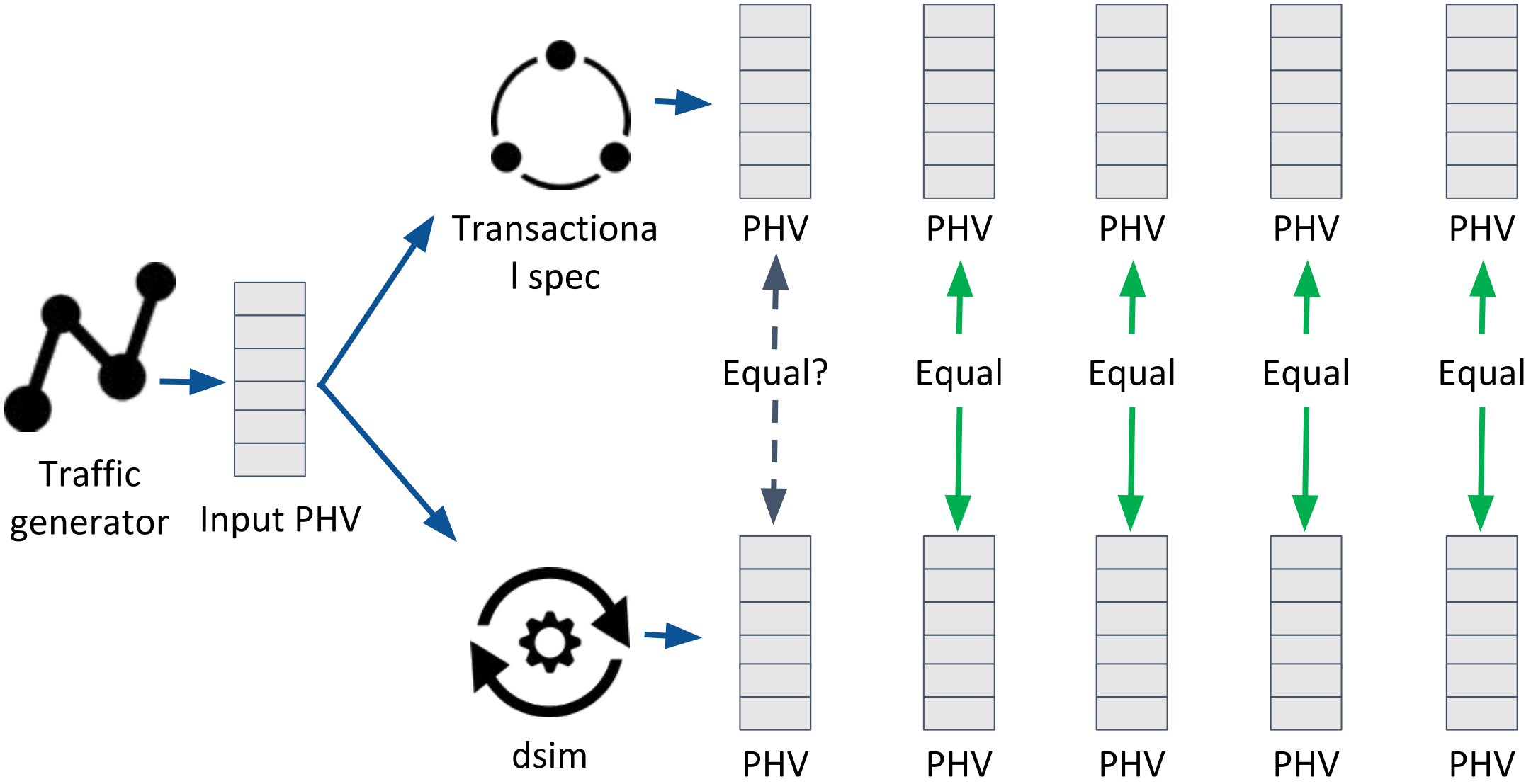
Compiler Testing Example



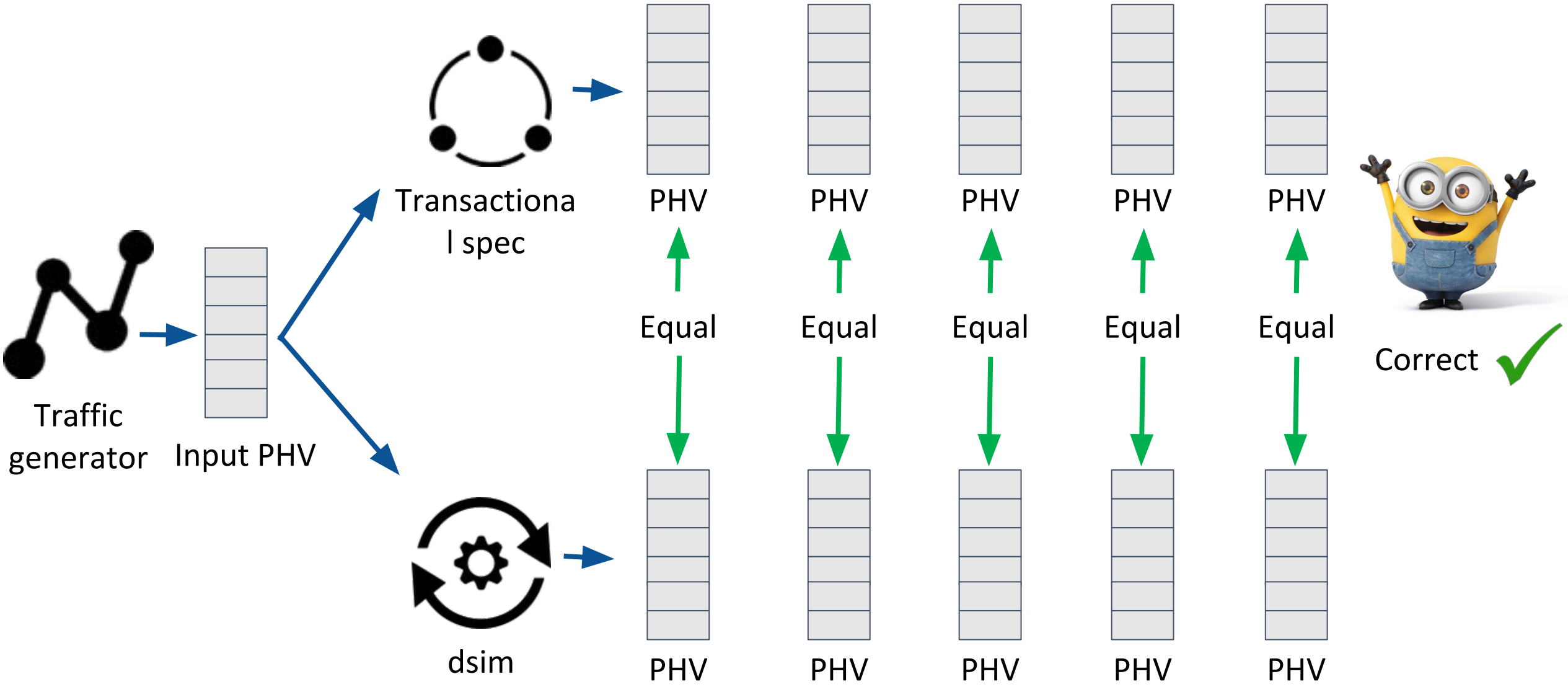


dsim

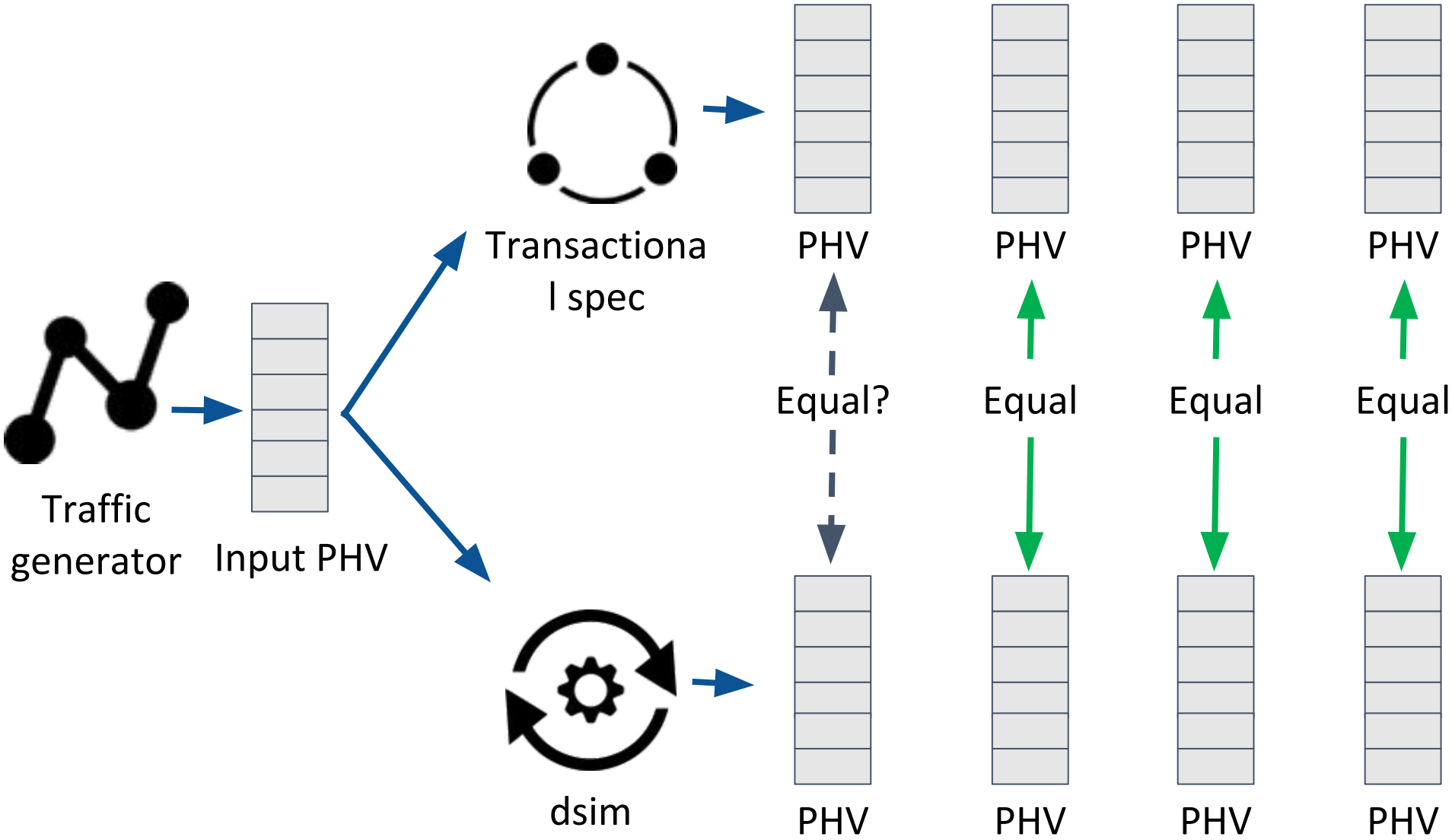
Compiler Testing Example



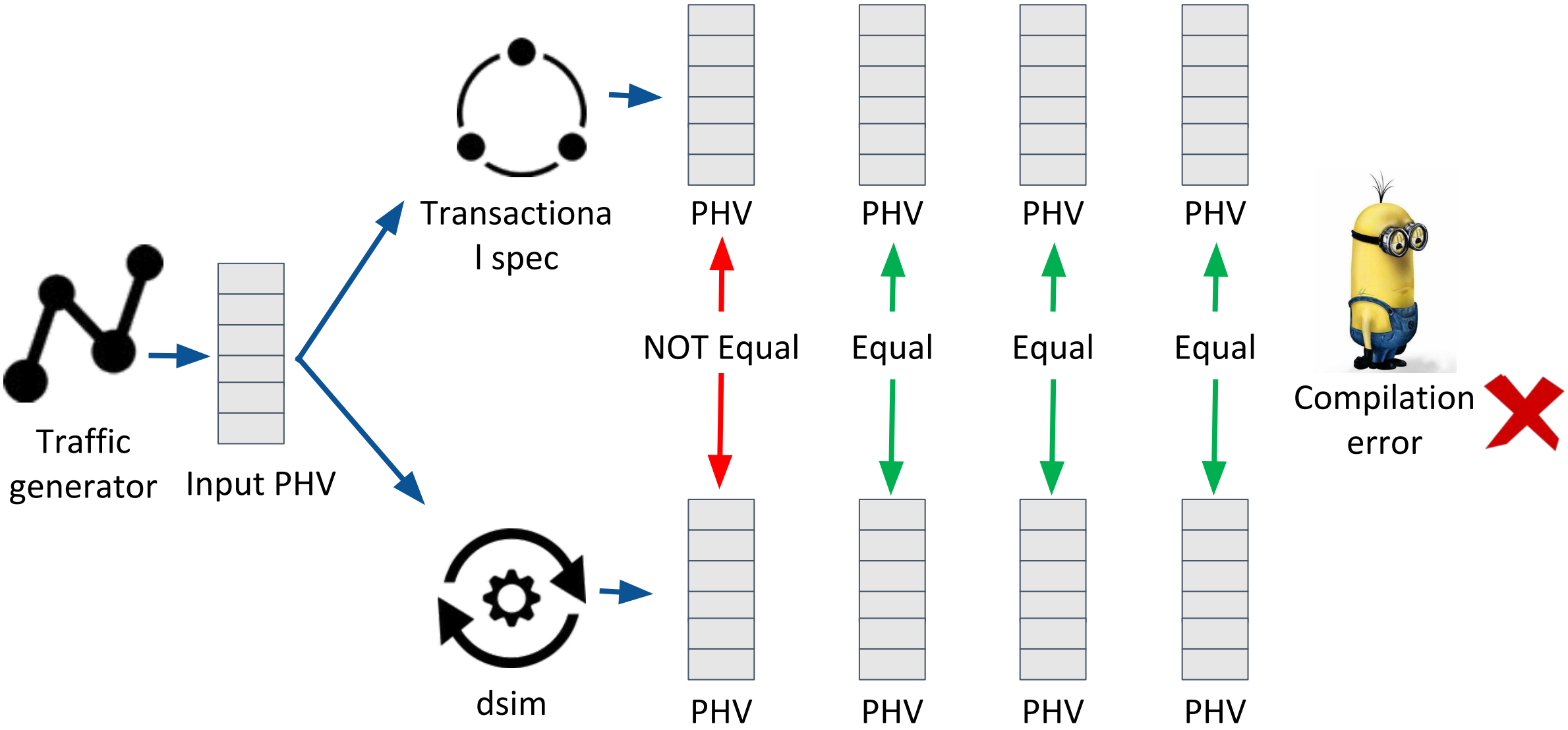
Compiler Testing Example



Compiler Testing Example



Compiler Testing Example



Testing Chipmunk

- Druzhba tested Chipmunk, a compiler for the Domino programming language

- The compilations of 120+ Chipmunk programs were validated

Conclusion

- We can model the low-level hardware details of the switch chip
- Druzhba can simulate compiler-generated machine code programs
- Compiler developers observe correctness of compiler mappings
- Code: https://github.com/chipmunk-project/druzhba-simulator

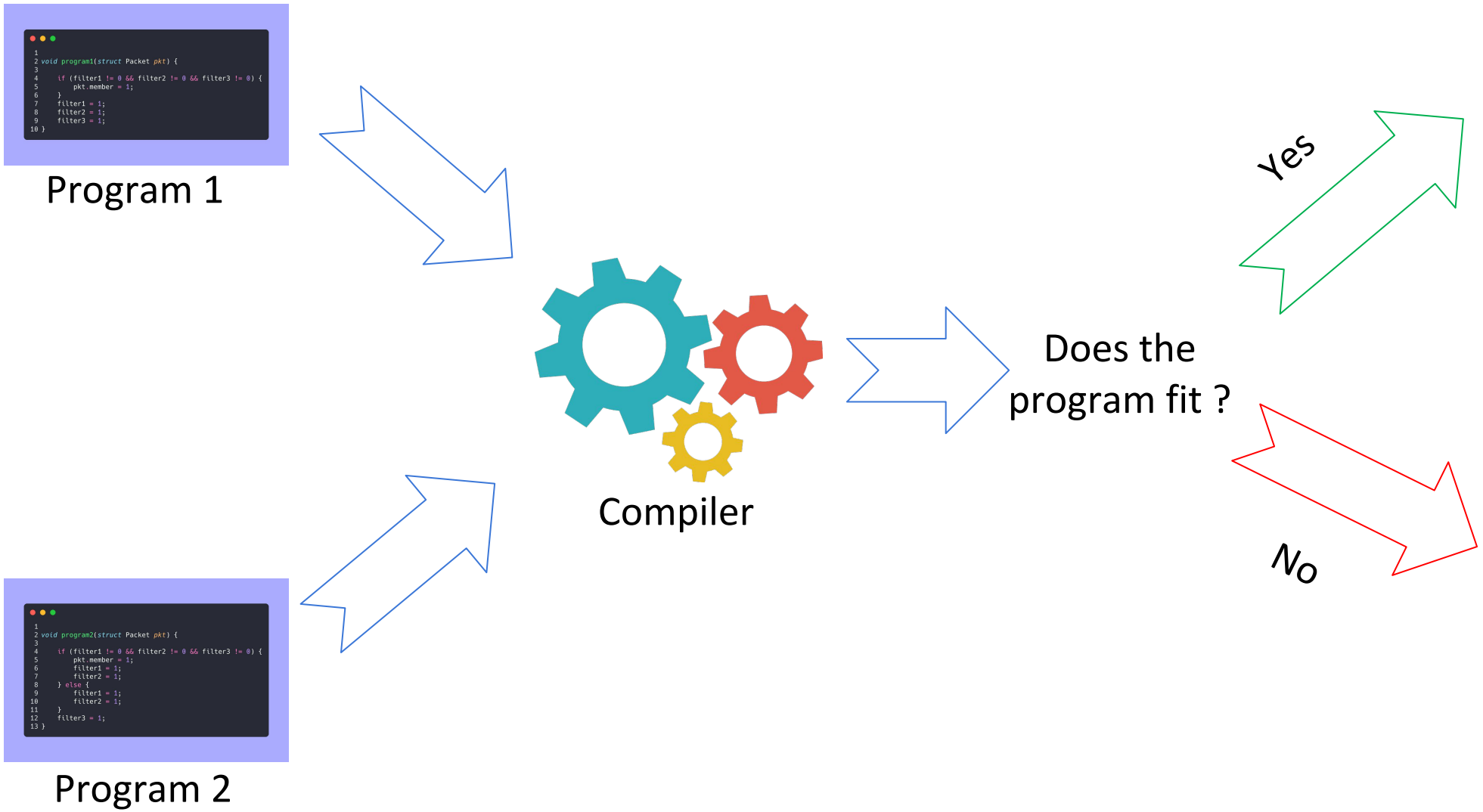
back up slides

False Positives

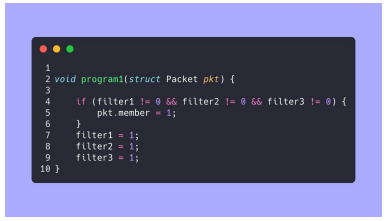
```
1
2 void program1(struct Packet pkt) {
3
4     if (filter1 != 0 && filter2 != 0 && filter3 != 0) {
5         pkt.member = 1;
6     }
7     filter1 = 1;
8     filter2 = 1;
9     filter3 = 1;
10 }
```

```
1
2 void program2(struct Packet pkt) {
3
4     if (filter1 != 0 && filter2 != 0 && filter3 != 0) {
5         pkt.member = 1;
6         filter1 = 1;
7         filter2 = 1;
8     } else {
9         filter1 = 1;
10        filter2 = 1;
11    }
12    filter3 = 1;
13 }
```

False Positives

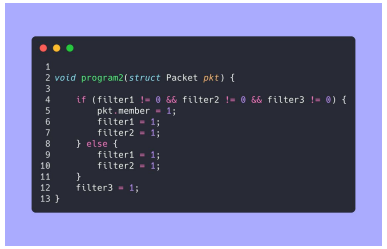


False Positives



```
1
2 void program1(struct Packet pkt) {
3
4     if (filter1 != 0 && filter2 != 0 && filter3 != 0) {
5         pkt.member = 1;
6     }
7     filter1 = 1;
8     filter2 = 1;
9     filter3 = 1;
10 }
```

Program 1



```
1
2 void program2(struct Packet pkt) {
3
4     if (filter1 != 0 && filter2 != 0 && filter3 != 0) {
5         pkt.member = 1;
6         filter1 = 1;
7         filter2 = 1;
8     } else {
9         filter1 = 1;
10        filter2 = 1;
11    }
12    filter3 = 1;
13 }
```

Program 2

Compiler

Does the program fit ?

Yes

No

Execute on switch

False Positives

```
1
2 void program1(struct Packet pkt) {
3
4     if (filter1 != 0 && filter2 != 0 && filter3 != 0) {
5         pkt.member = 1;
6     }
7     filter1 = 1;
8     filter2 = 1;
9     filter3 = 1;
10 }
```

Program 1

```
1
2 void program2(struct Packet pkt) {
3
4     if (filter1 != 0 && filter2 != 0 && filter3 != 0) {
5         pkt.member = 1;
6         filter1 = 1;
7         filter2 = 1;
8     } else {
9         filter1 = 1;
10        filter2 = 1;
11    }
12    filter3 = 1;
13 }
```

Program 2

Compiler

Does the program fit ?

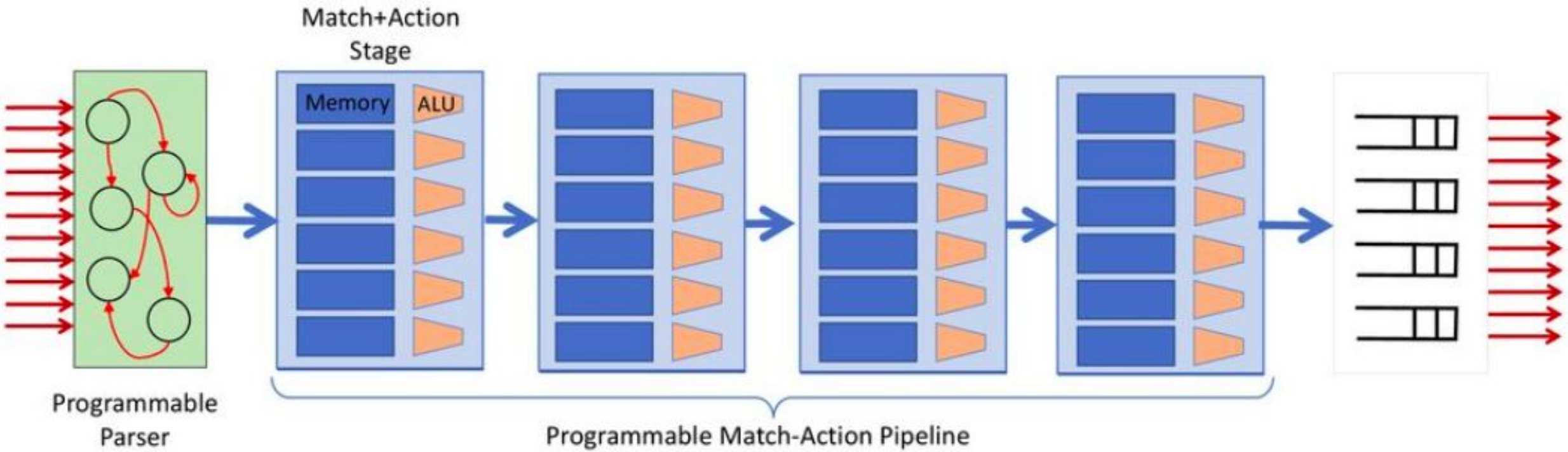
Yes

Execute on switch

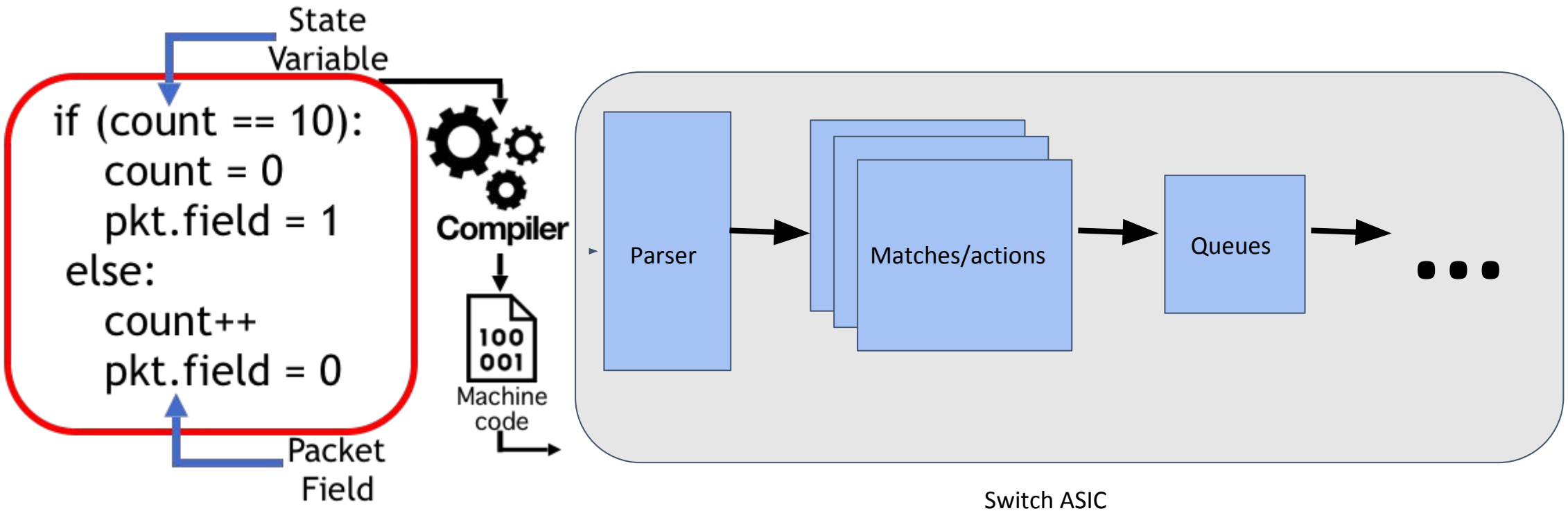
No

Rejected

Switch Architecture



Compilation to pipeline



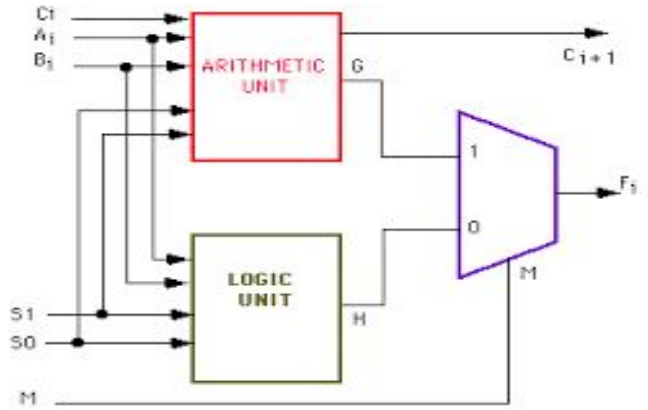
Hardware Specification

- Pipeline dimensions (depth, width)
  - Number of total stages and number of ALUs per stage

- ALU specifications
  - ALU DSL used to specify capabilities of switch ALUs

2 x 2

Pipeline depth x width



ALU specification

Optimizations

- Sparse conditional constant propagation
  - Constant propagation of machine code values followed by interpretation of control flow

- Function inlining
  - Replace pipeline description

Compiler Testing Workflow