Alibaba Cloud
Worldwide Cloud Services Partner

Aquila: A Practically Usable Verification System for Production-Scale Programmable Data Planes

Bingchuan Tian, Jiaqi Gao, Mengqi Liu, Ennan Zhai, Yanqing Chen,
Yu Zhou, Li Dai, Feng Yan, Mengjing Ma, Ming Tang, Jie Lu, Xionglie Wei,
Hongqiang Harry Liu, Ming Zhang, Chen Tian and Minlan Yu

南京大學 NANJING UNIVERSITY
Alibaba Group 阿里巴巴集团
HARVARD UNIVERSITY

Alibaba leads the deployment of programmable switches in the industry

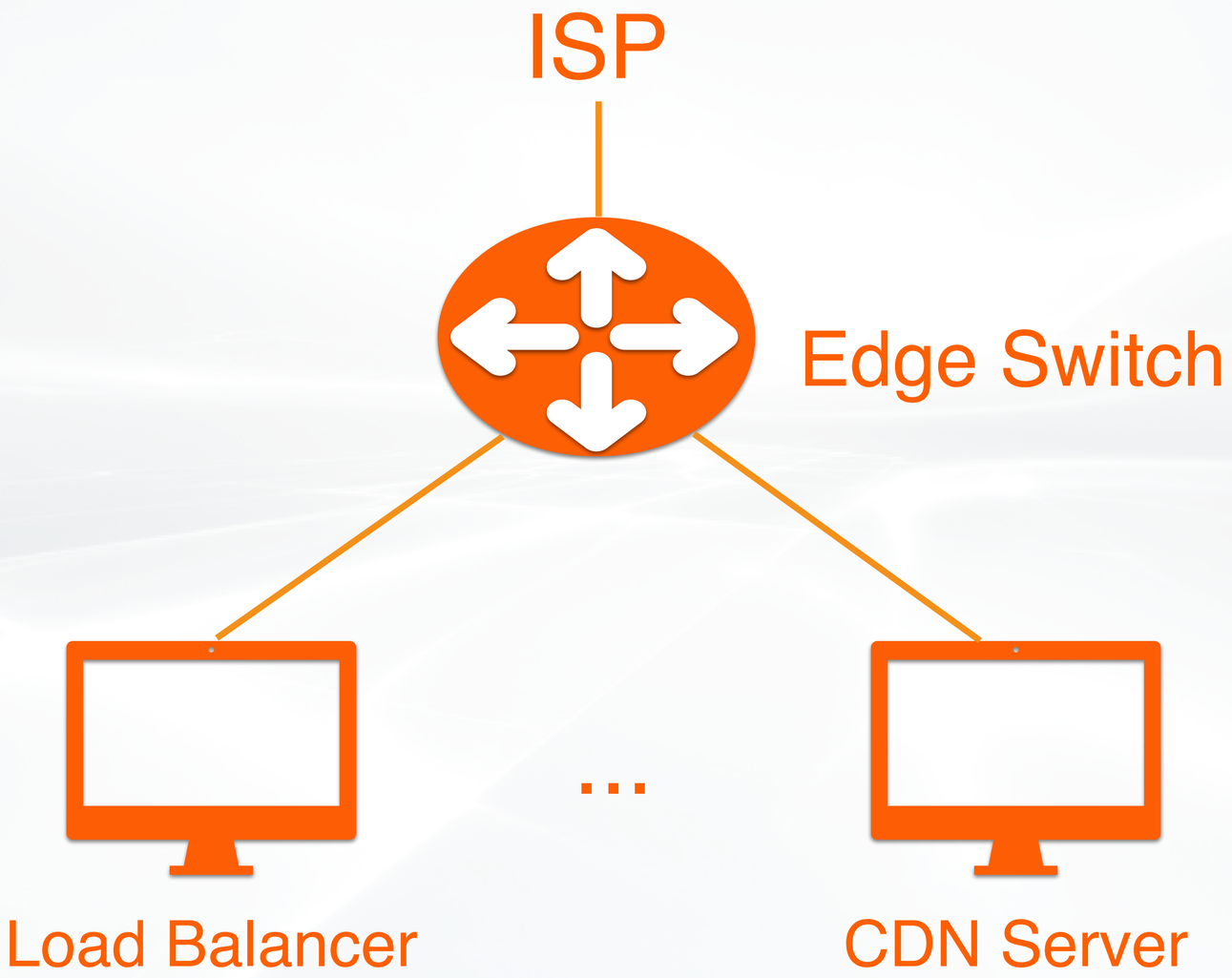
淘宝网
Taobao.com

天猫
TMALL

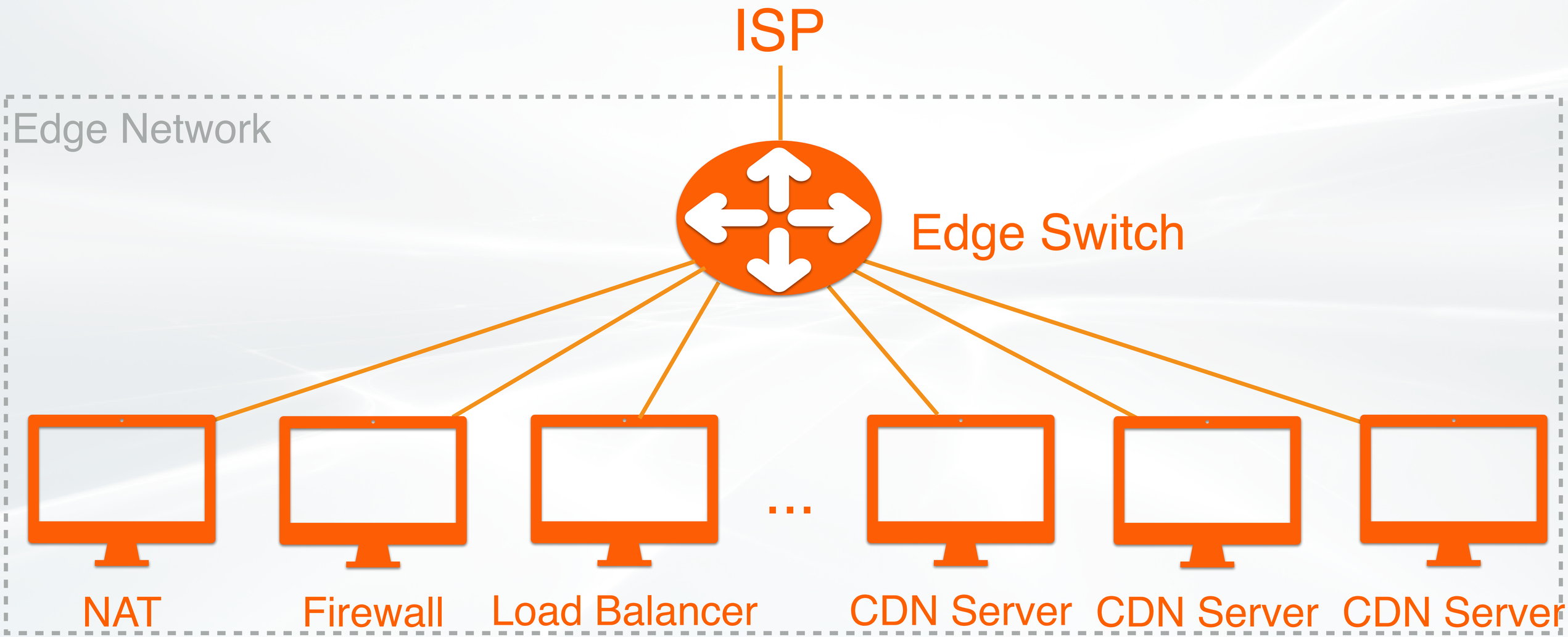
支付宝
ALIPAY

Alibaba Cloud

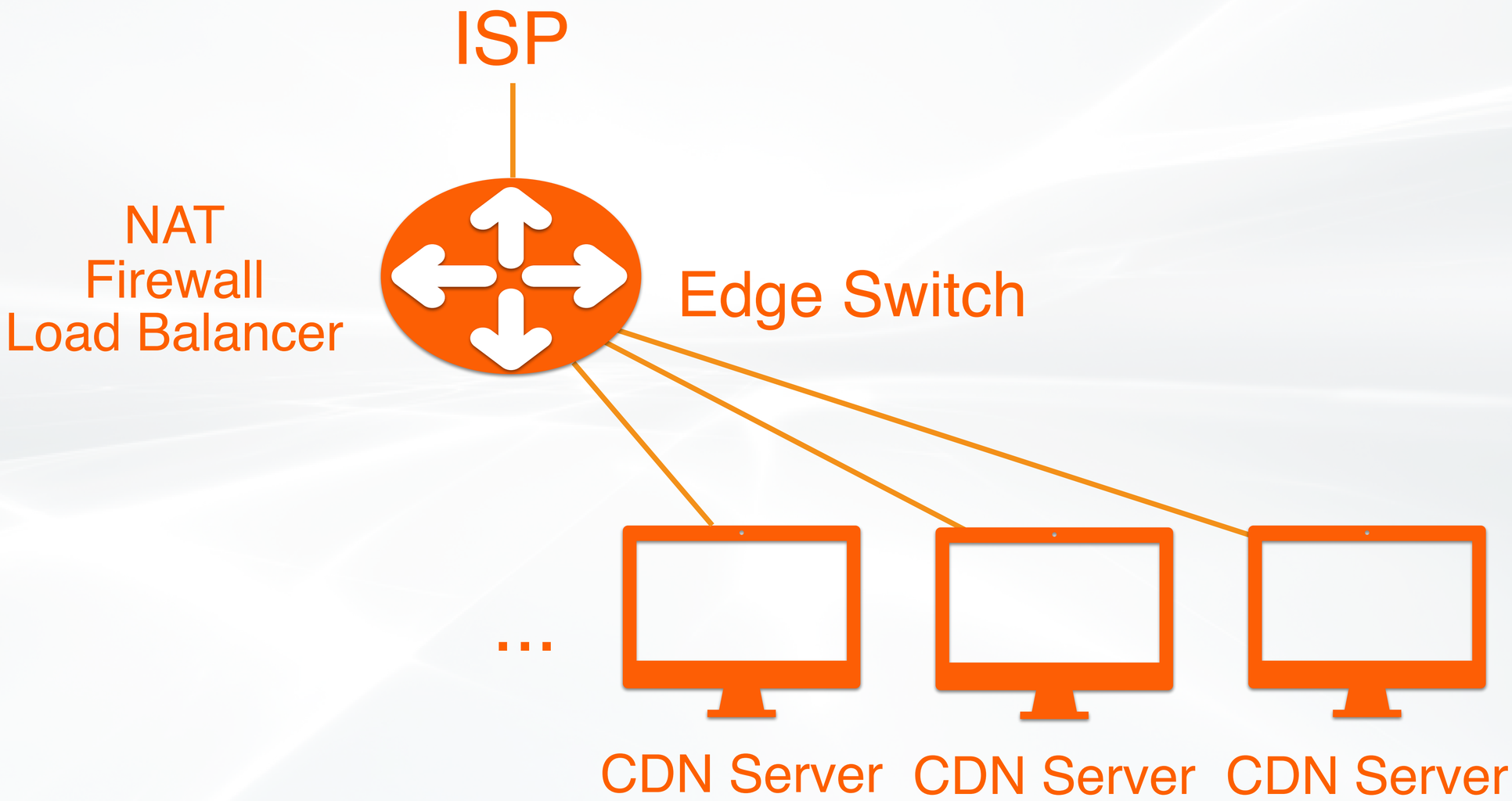
Alibaba leads the deployment of programmable switches in the industry



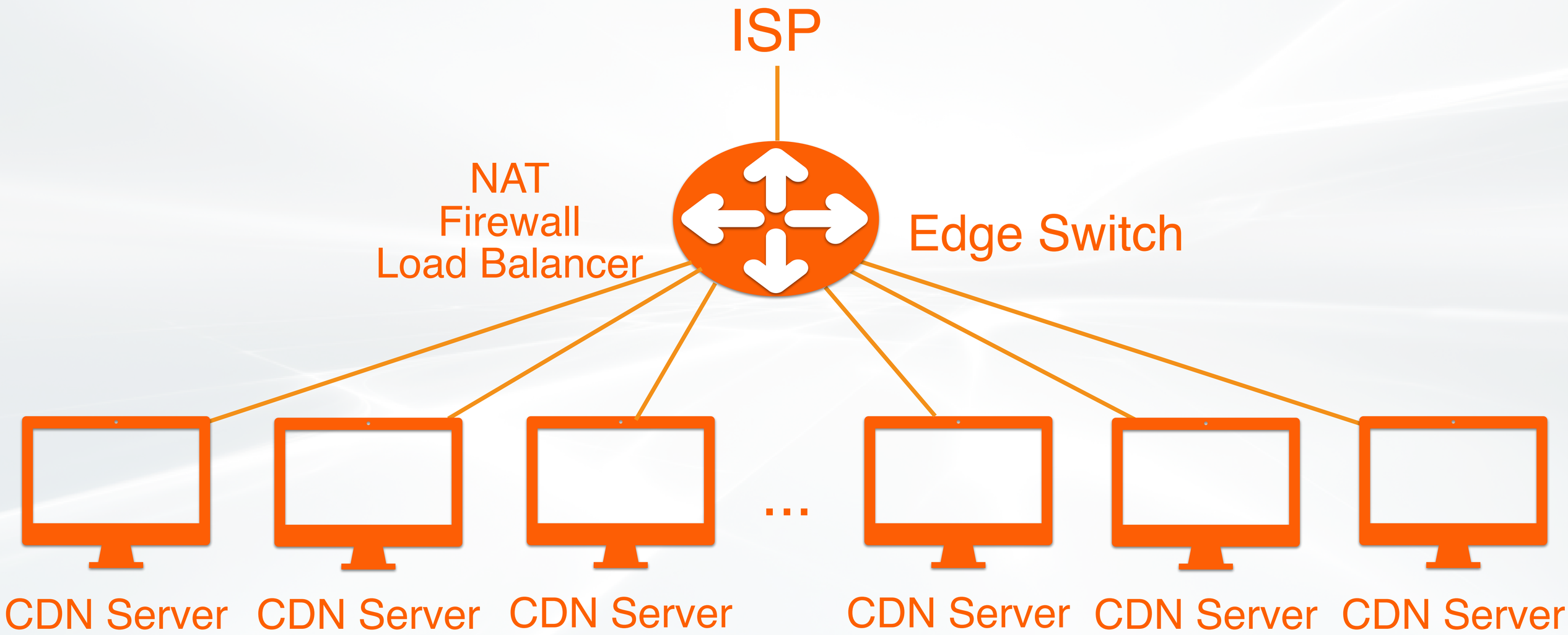
Alibaba leads the deployment of programmable switches in the industry



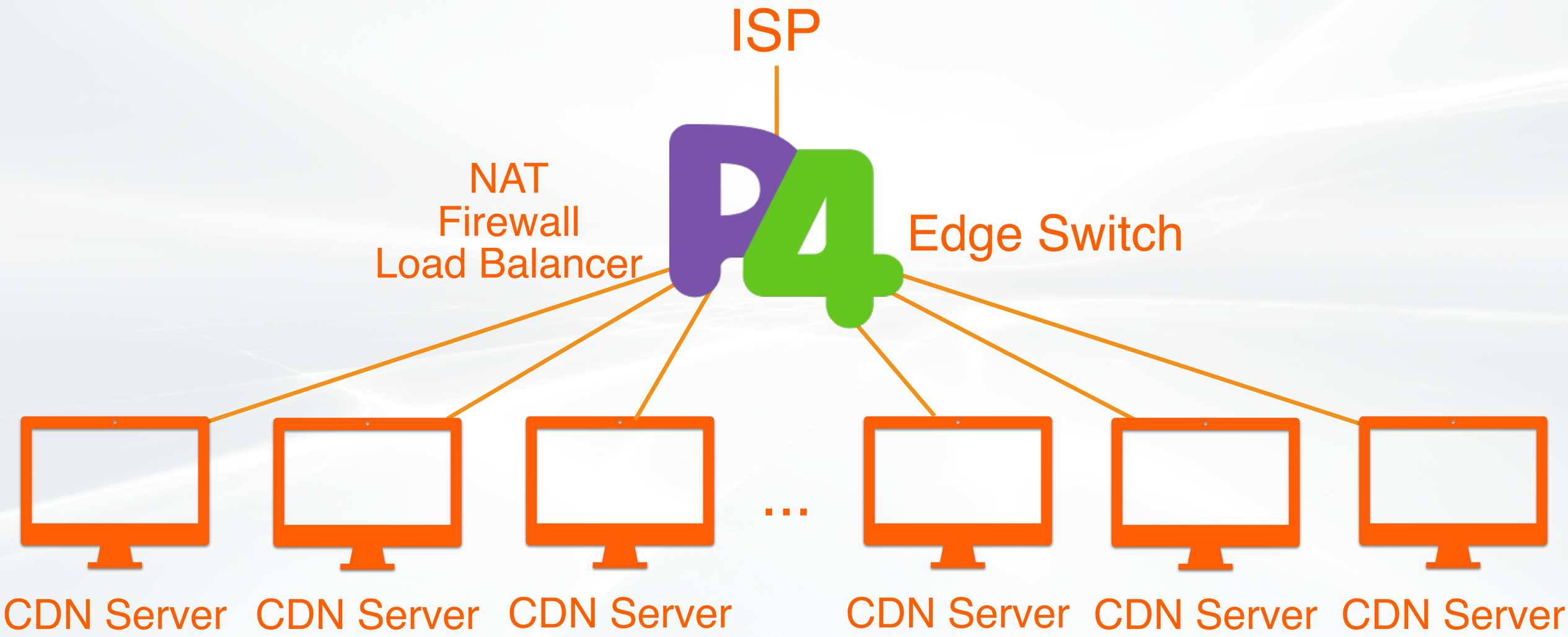
Alibaba leads the deployment of programmable switches in the industry



Alibaba leads the deployment of programmable switches in the industry



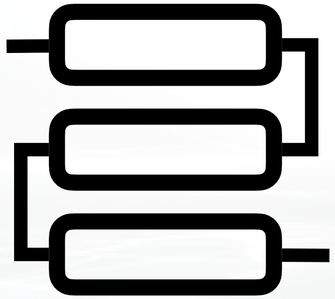
Alibaba leads the deployment of programmable switches in the industry



Alibaba leads the deployment of programmable switches in the industry

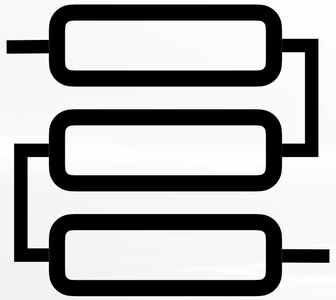
- Functionality
- Efficiency
- Flexibility

Ensuring the correctness of data plane programs is challenging

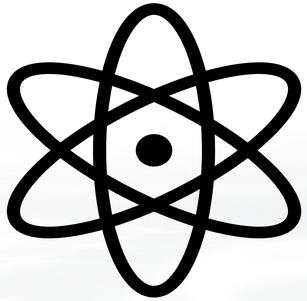


Multiple Pipelines

Ensuring the correctness of data plane programs is challenging



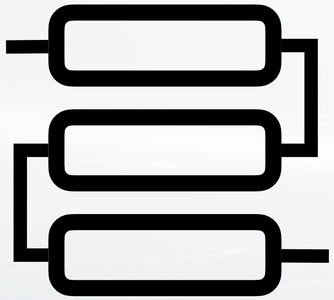
Multiple Pipelines



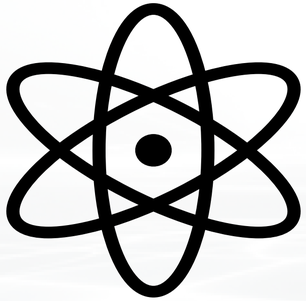
Many Network Functions

* NAT
* Load Balancer
* DDoS Defense

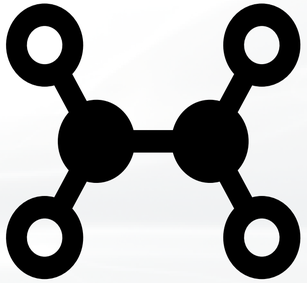
Ensuring the correctness of data plane programs is challenging



Multiple Pipelines

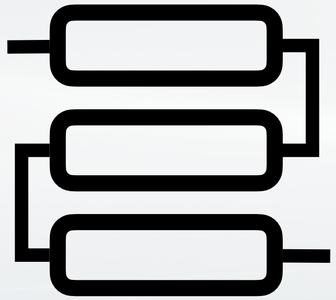


Many Network Functions

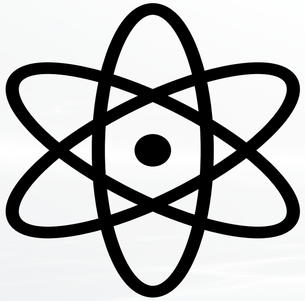


Various Packet Paths

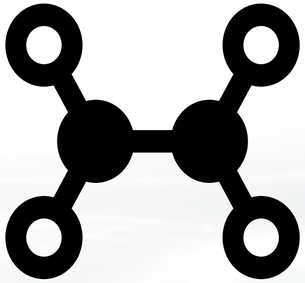
Ensuring the correctness of data plane programs is challenging



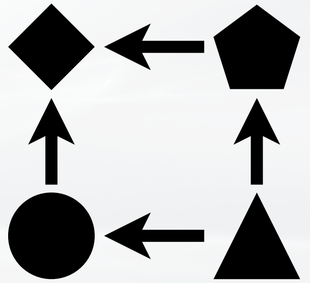
Multiple Pipelines



Many Network Functions



Various Packet Paths



Complex Function Chain

Ensuring the correctness of data plane programs is challenging

- ✓ Functionality
- ✓ Efficiency
- ✓ Flexibility
- ✗ Correctness

Ensuring the correctness of data plane programs is challenging

+

Formal Verification

- Functionality
- Efficiency
- Flexibility
- Correctness

Alibaba decided to build a practically usable data plane program verification system

Goal 1: Simple Intent Language → Challenge 1

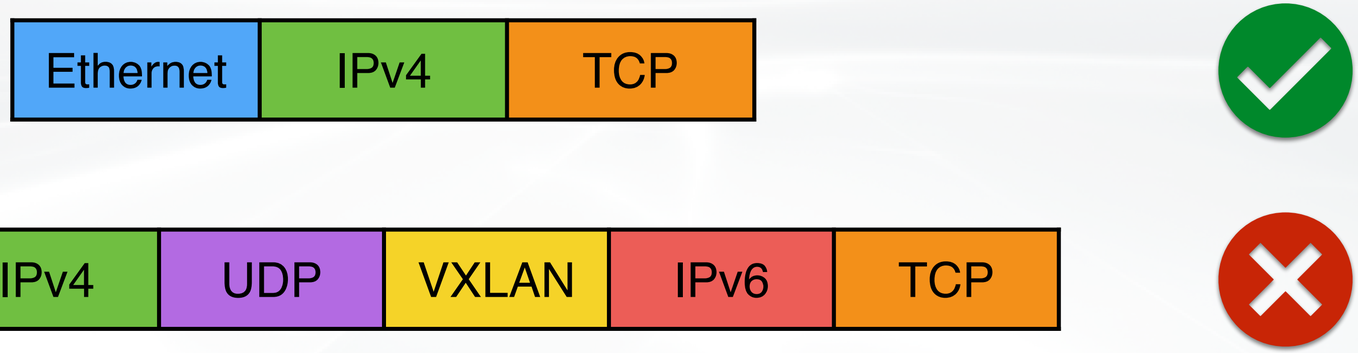
Goal 2: Scalable Verification Algorithm → Challenge 2

Goal 3: Accurate Bug Localization → Challenge 3

Goal 4: Verifier's Self Validation → Challenge 4

Challenge 1: Intent Complexity

For each TCP packet, we should not change its TCP header.



Challenge 1: Intent Complexity

For each TCP packet, we should not change its TCP header.

p4v spec [SIGCOMM'18]

```
parse_eth:
    last := eth
parse_vlan:
    assume last == eth
    last := vlan
parse_ipv4:
    assume last == eth || last == vlan
    last := ipv4
parse_ipv6:
    assume last == eth || last == vlan
    last := ipv4
parse_tcp:
    assume last == ipv4 || last == ipv6
    last := tcp
    @tcp.src_port := tcp.src_port
    @tcp.dst_port := tcp.dst_port
    ...
parse_udp:
    last := udp

assume last == tcp
assert tcp.src_port == @tcp.src_port
assert tcp.dst_port == @tcp.dst_port
...
```

Vera spec [SIGCOMM'18]

```
InstructionBlock(
    CreateTag("START", 0),
    Call("router.generator.eth.ipv4.tcp"),
    res.initFactory(switchInstance)
)
InstructionBlock(
    CreateTag("START", 0),
    Call("router.generator.eth.ipv6.tcp"),
    res.initFactory(switchInstance)
)
InstructionBlock(
    CreateTag("START", 0),
    Call(router.generator.eth.vlan.ipv4.tcp),
    res.initFactory(switchInstance)
)
InstructionBlock(
    CreateTag("START", 0),
    Call(router.generator.eth.vlan.ipv6.tcp),
    res.initFactory(switchInstance)
)

AF(Constrain(tcp.src_port, Eq(Original.tcp.src_port)))
AF(Constrain(tcp.dst_port, Eq(Original.tcp.dst_port)))
...
```

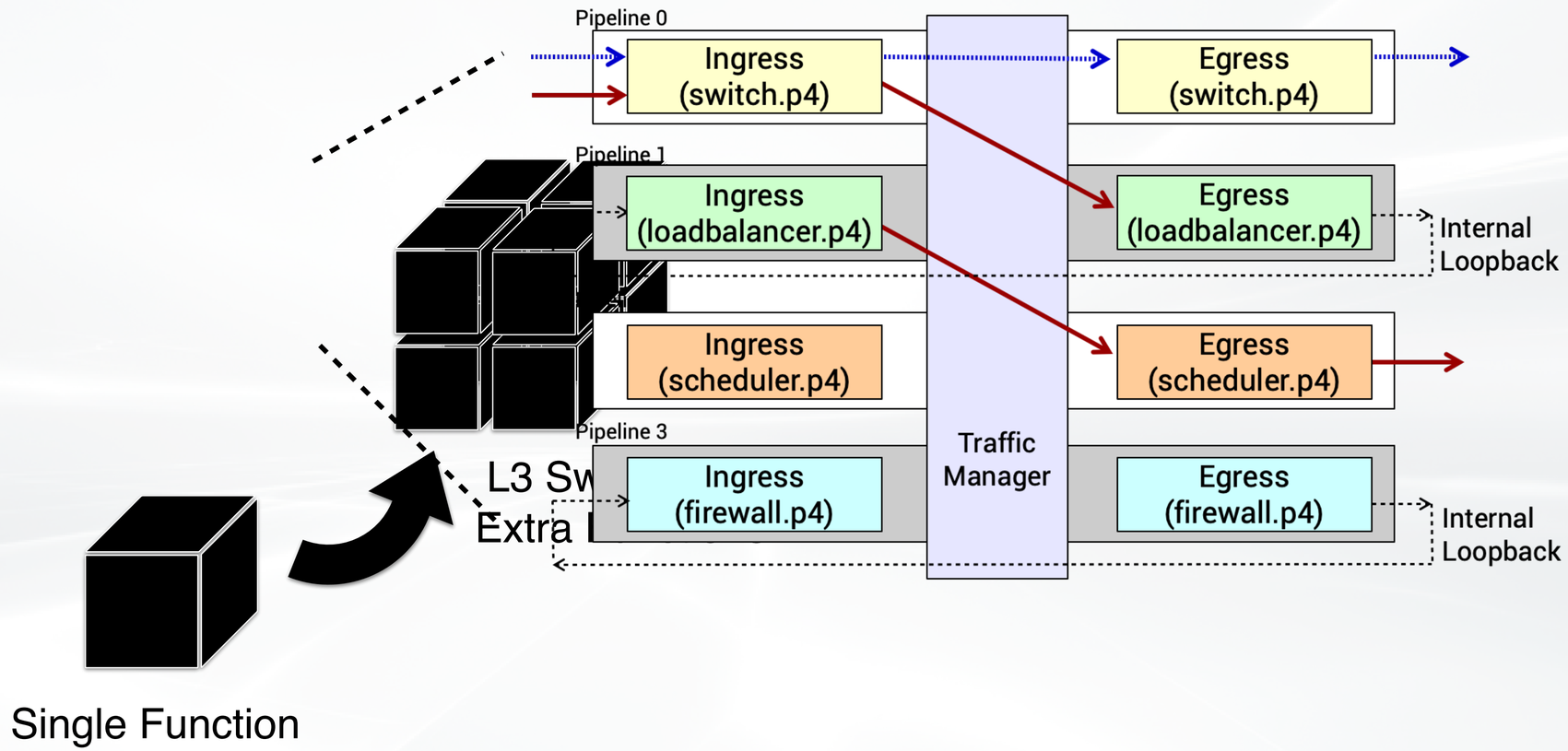
Describing header order constraints

Enumerating header orders

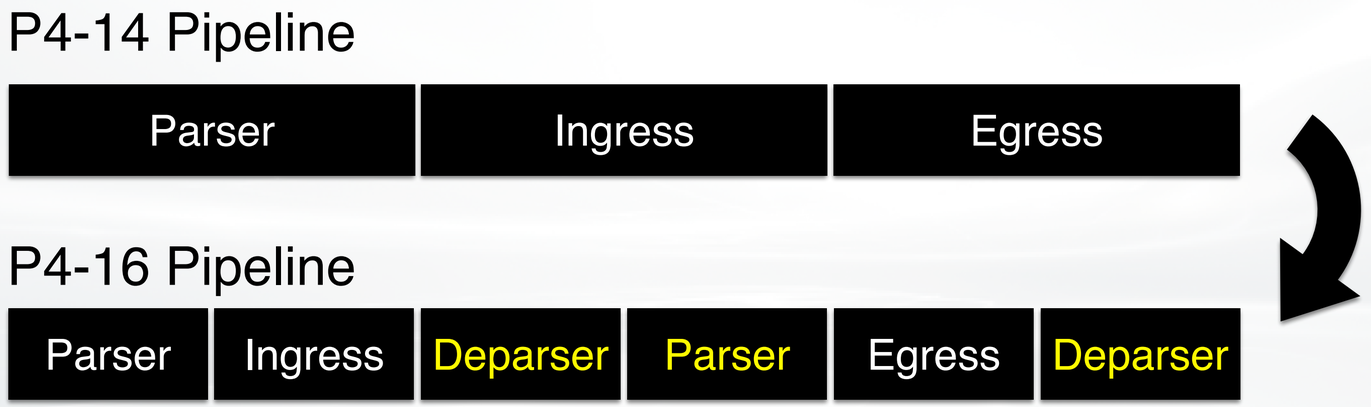
@pkt.$order == <eth [vlan] (ipv4|ipv6) tcp> => keep(tcp)

Preferred

Challenge 2: Verification Scalability



Challenge 2: Verification Scalability



Existing work is not scalable to our production P4-16 programs!

Challenge 3: Bug Localization

Fail

Counterexample:
ipv4.src_addr = 0x0a000001
ipv4.dsr_addr = 0x0b000001
…

Trace:
[1] validate.p4(10) validate_eth
[2] validate.p4(50) validate_ipv4
[3] l3.p4(20) ipv4_fib
[4] acl.p4(30) ipv4_acl
[5] acl.p4(80) system_acl
...

Existing work

Fail

Counterexample:
ipv4.src_addr = 0x0a000001
ipv4.dsr_addr = 0x0b000001
…

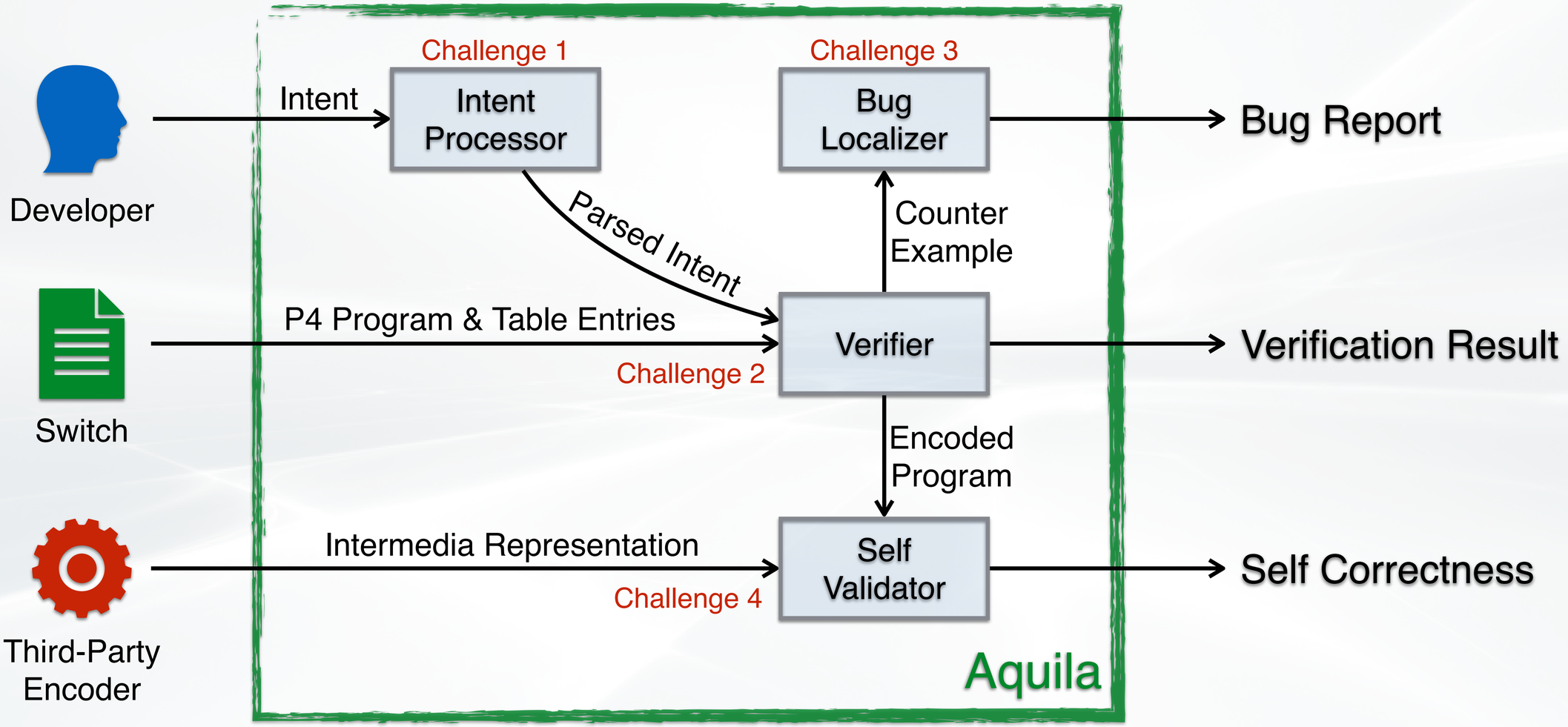
Trace:
[1] validate.p4(10) validate_eth
[2] validate.p4(50) validate_ipv4
[3] l3.p4(20) ipv4_fib
[4] acl.p4(30) ipv4_acl
[5] acl.p4(80) system_acl
...

Preferred

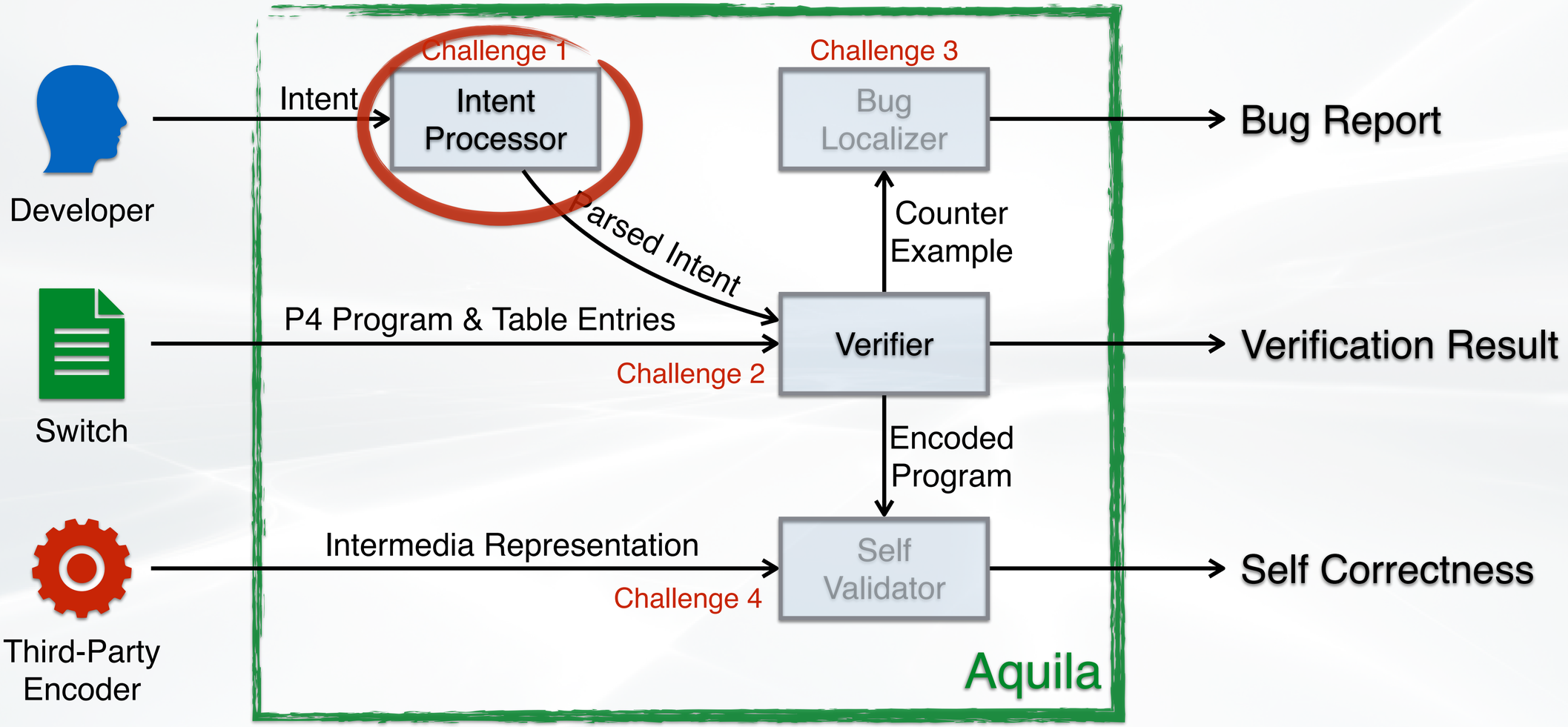
Challenge 4: Reliability of Verifier

No existing work can do self-validation

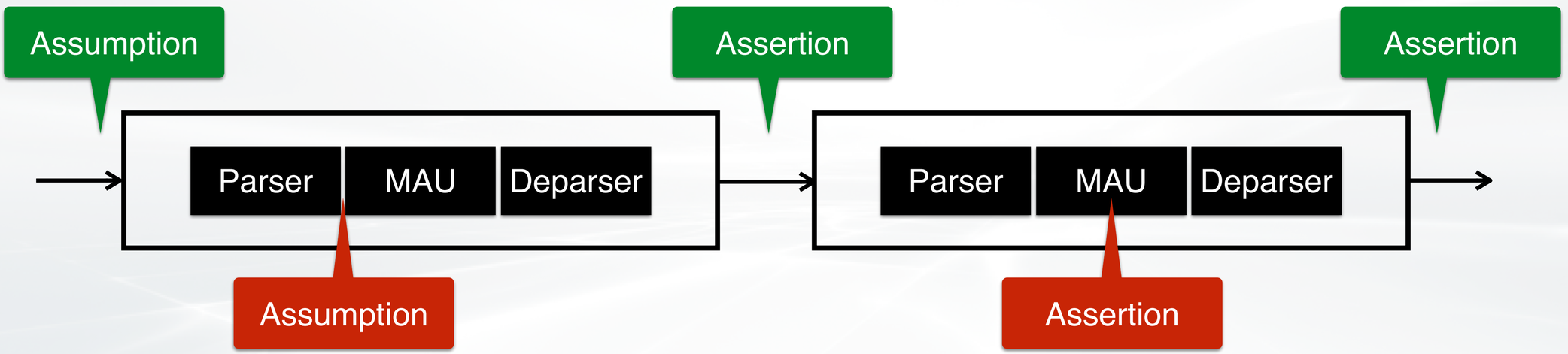
How to make a P4 verifier practically usable?



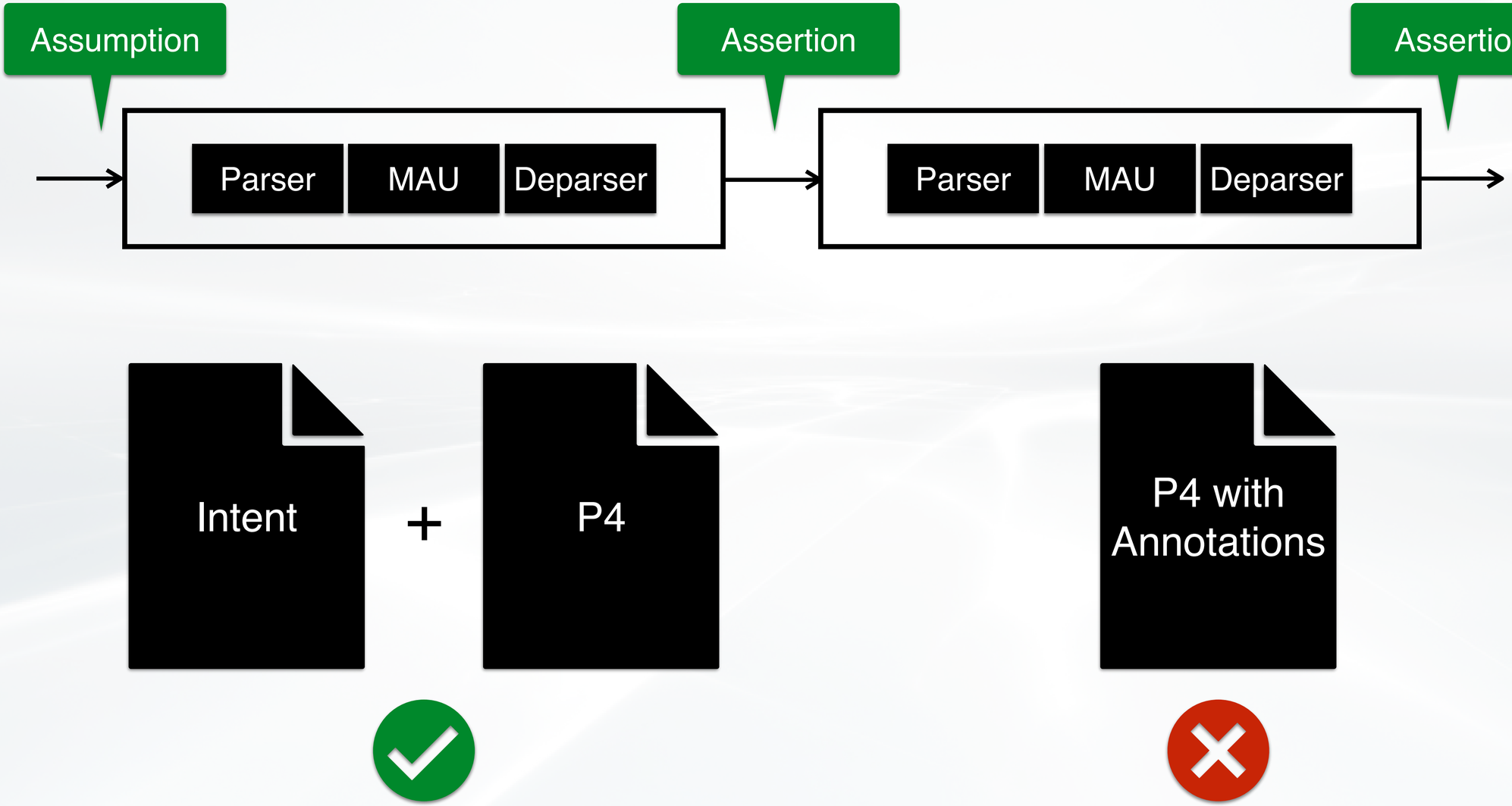
Aquila Architecture



Insights of Intent Language Design



Insights of Intent Language Design



Intent Language Overview

Experience + Abstraction

Parsers

- ☑ Header Order
- ☑ Header Parsing
- ☑ …

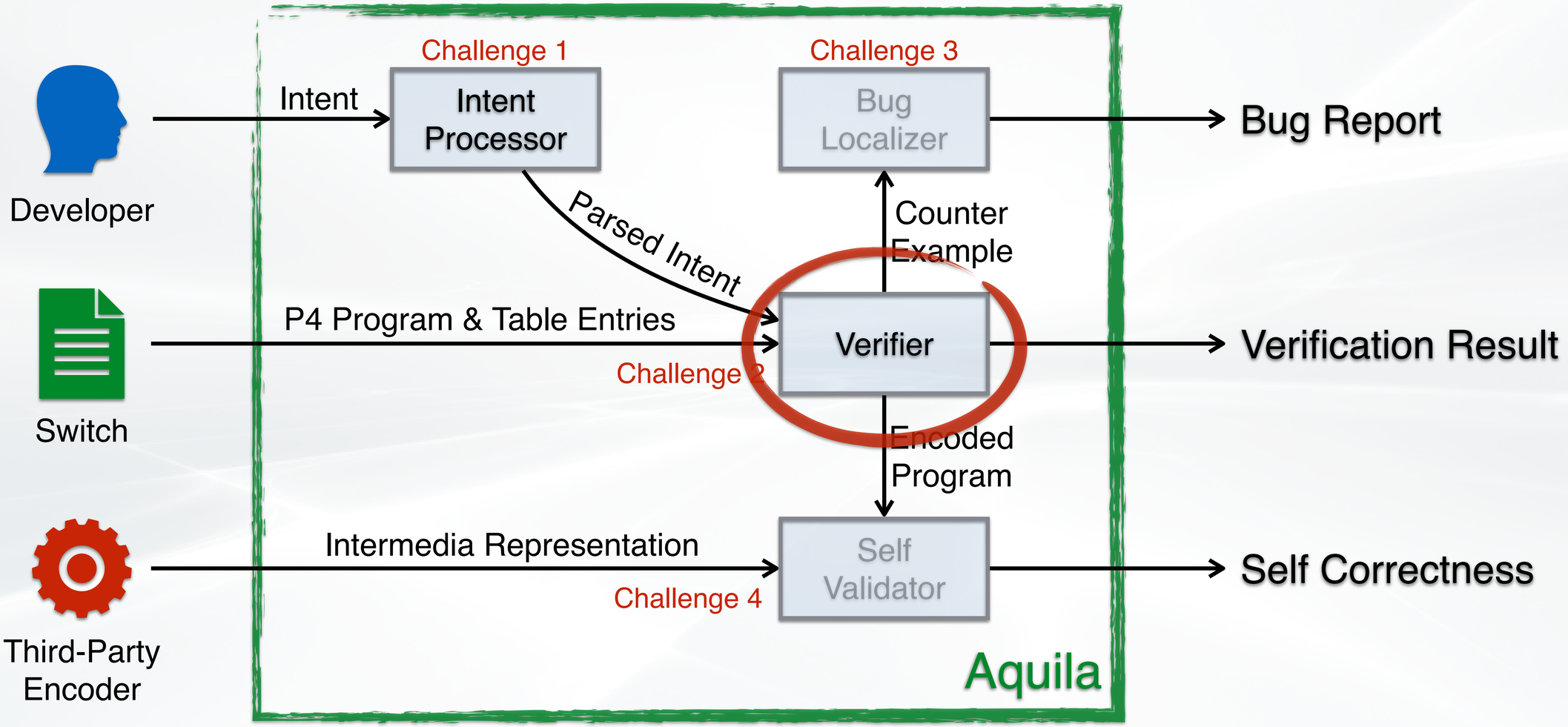
Match-Action Units

- ☑ Header Validity
- ☑ Payload Correctness
- ☑ Entry Correctness
- ☑ Deparsing
- ☑ …

Program Architecture

- ☑ Multi-Pipeline
- ☑ ASIC Behavior
- ☑ Register
- ☑ …

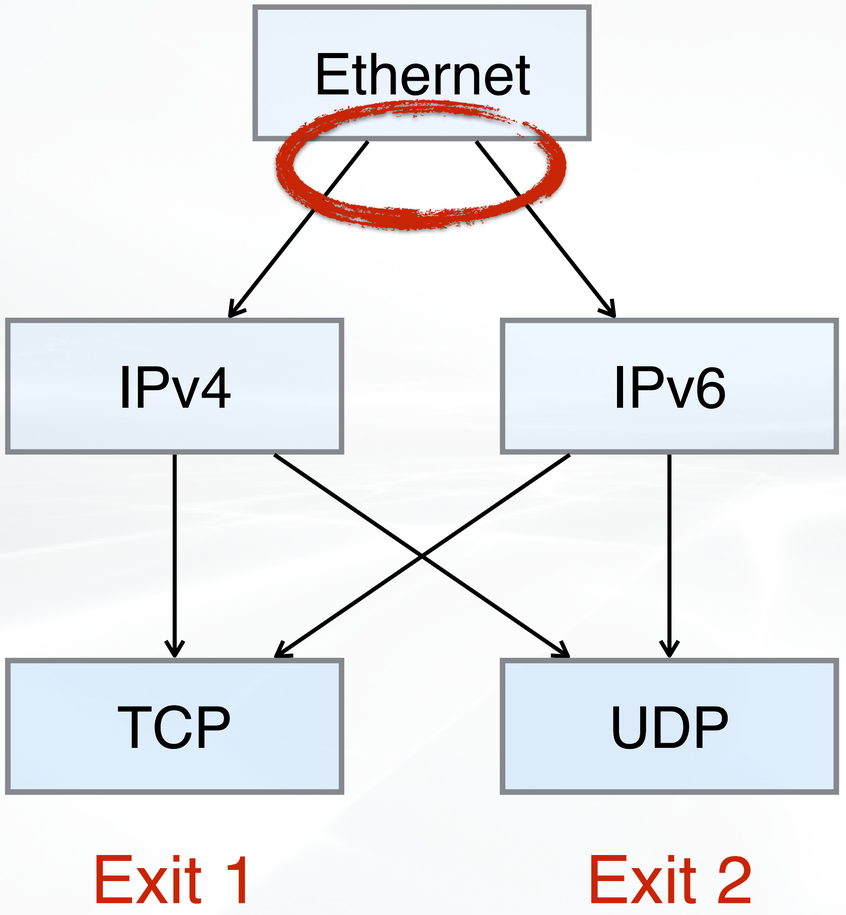
Aquila Architecture



We omit Challenge 3 and Challenge 4 in this talk

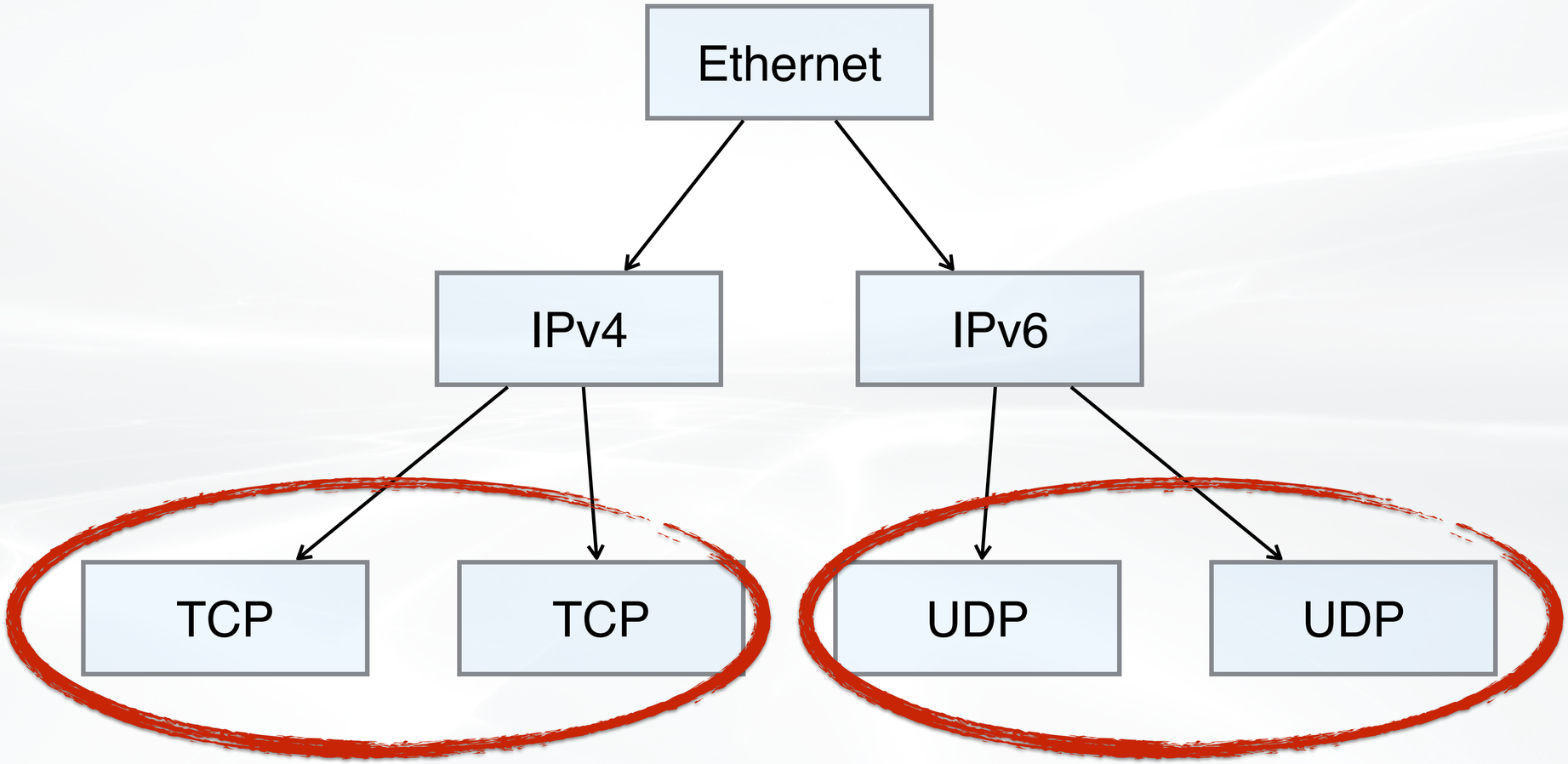
Parser State Explosion

```
Ethernet;
if (...) {
  Ipv4;
} else {
  Ipv6;
}
if (...) {
  TCP;
} else {
  UDP;
}
```



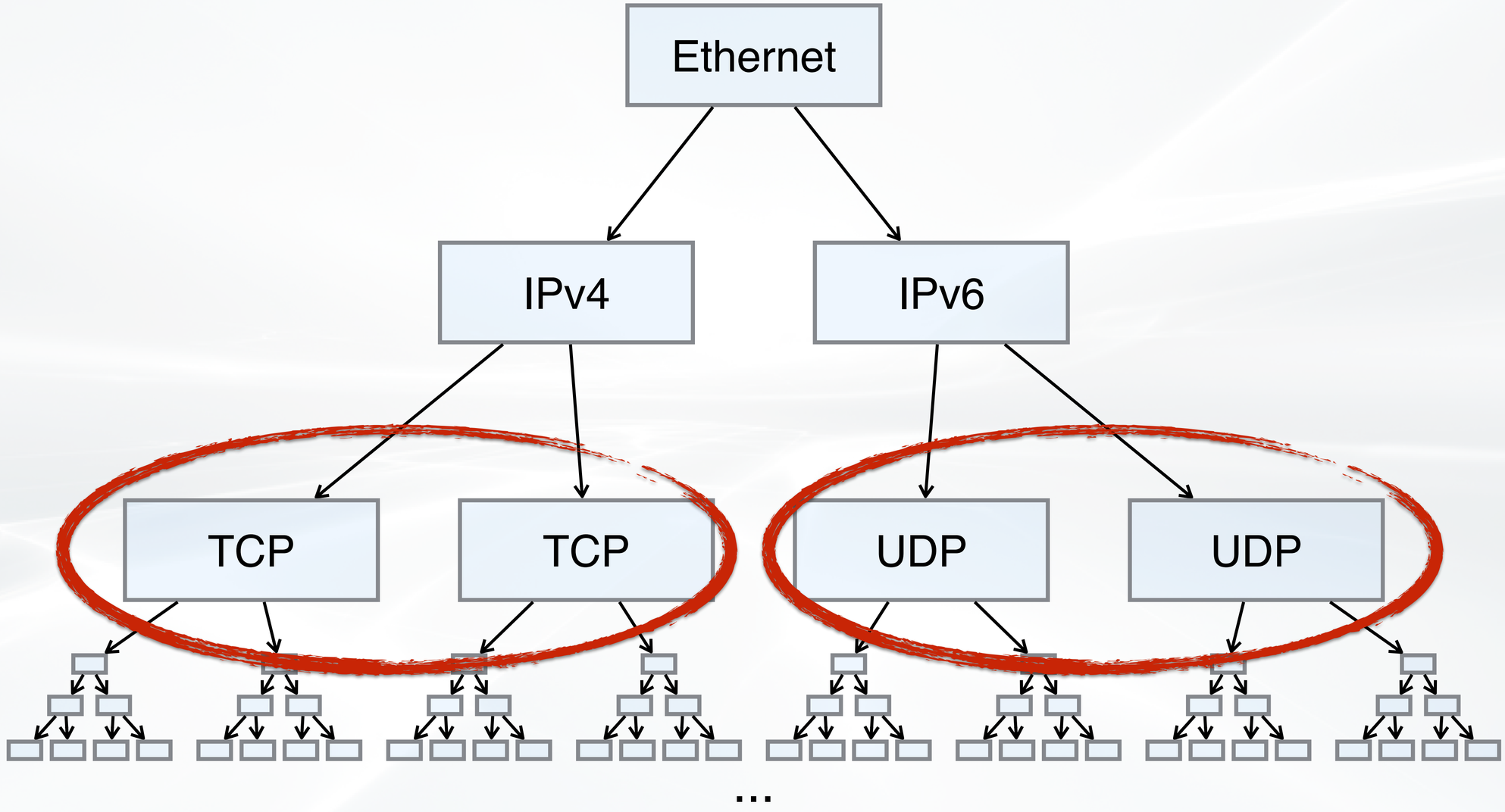
Parser State Explosion

```
Ethernet;
if (...) {
  Ipv4;
  if (...) {
    TCP;
  } else {
    UDP;
  }
} else {
  Ipv6;
  if (...) {
    TCP;
  } else {
    UDP;
  }
}
```

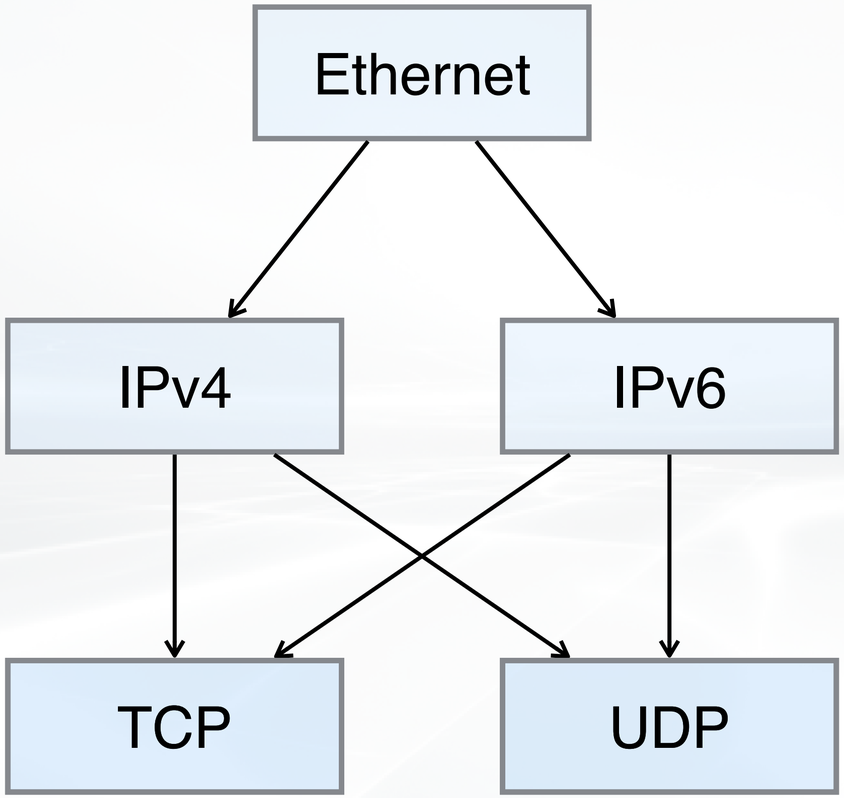


Parser State Explosion

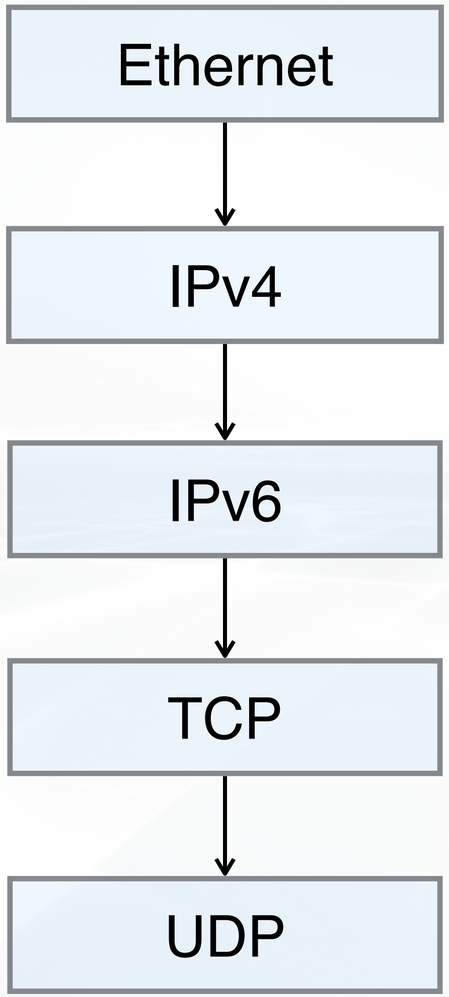
```
Ethernet;
if (...) {
  Ipv4;
  if (...) {
    TCP;
  } else {
    UDP;
  }
} else {
  Ipv6;
  if (...) {
    TCP;
  } else {
    UDP;
  }
}
```



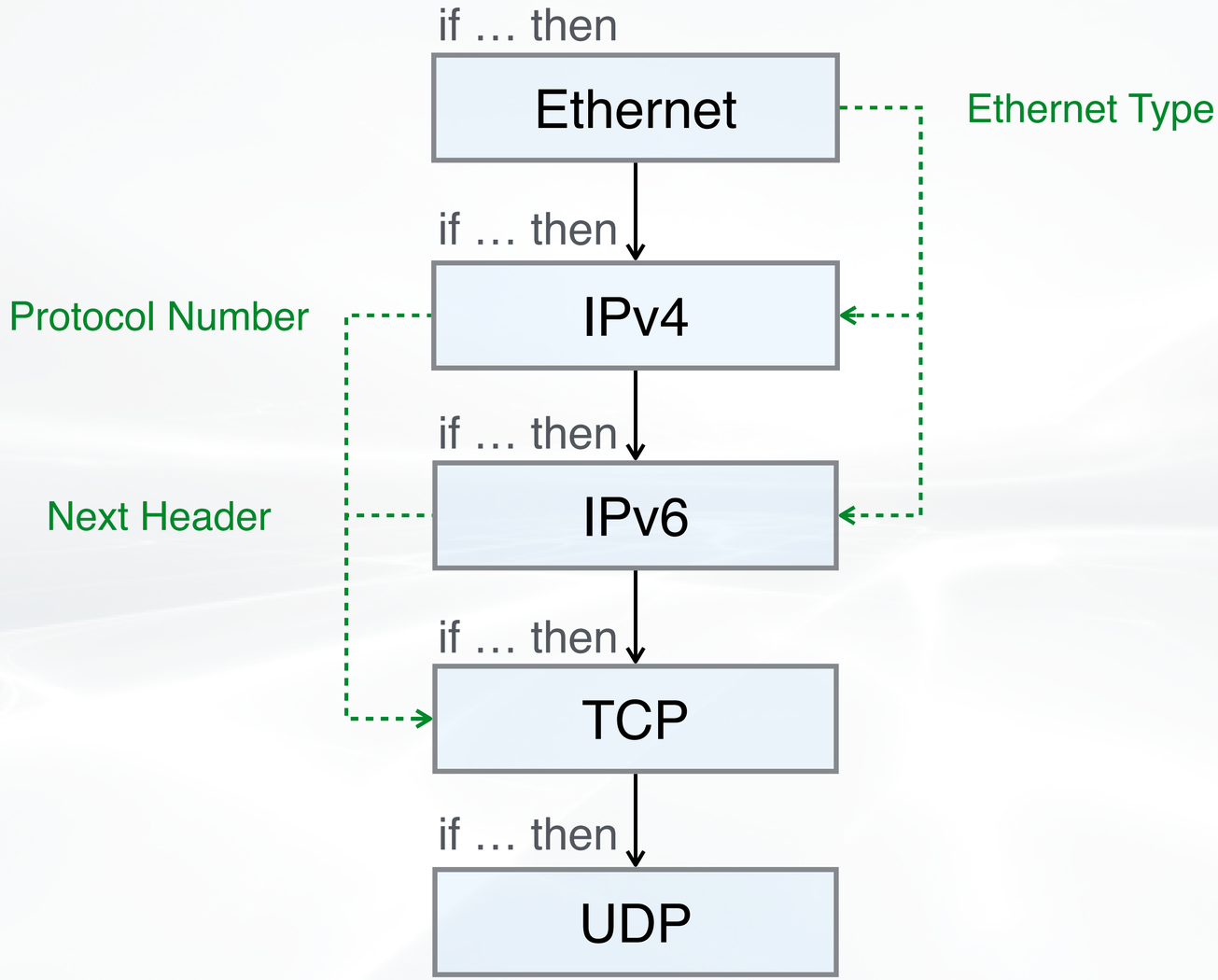
Our Solution: Sequential Encoding



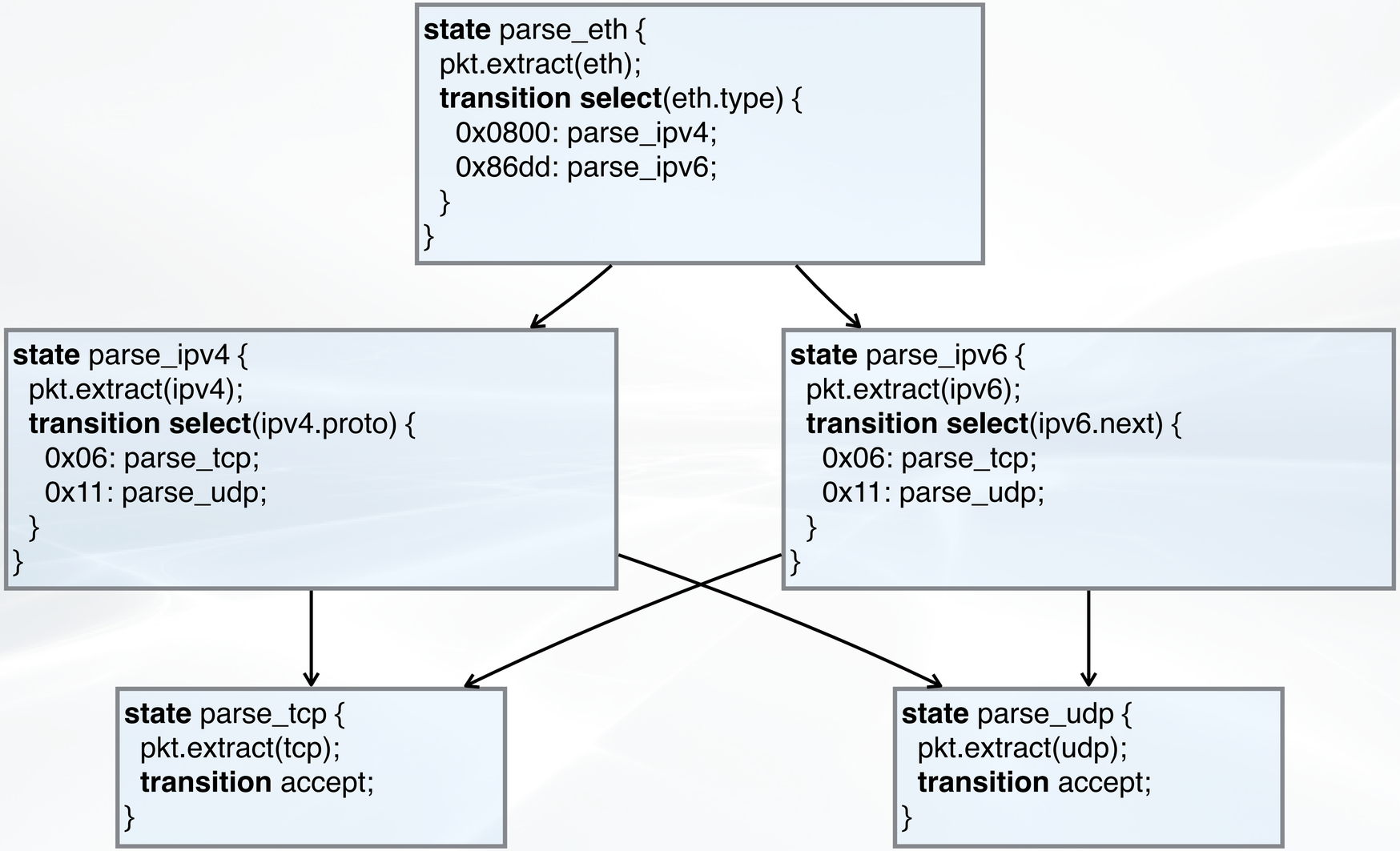
Our Solution: Sequential Encoding



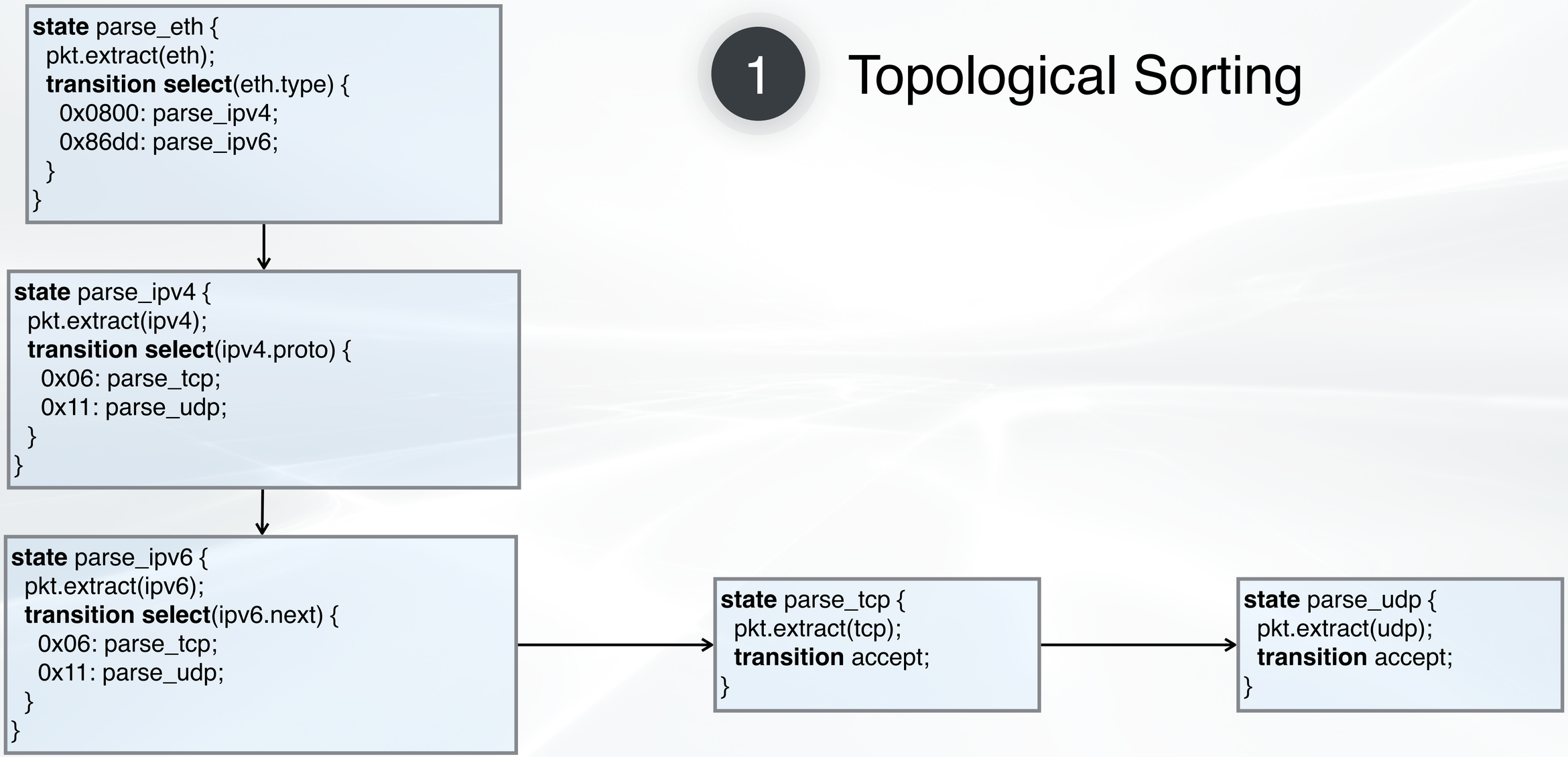
Our Solution: Sequential Encoding



Example of Sequential Encoding

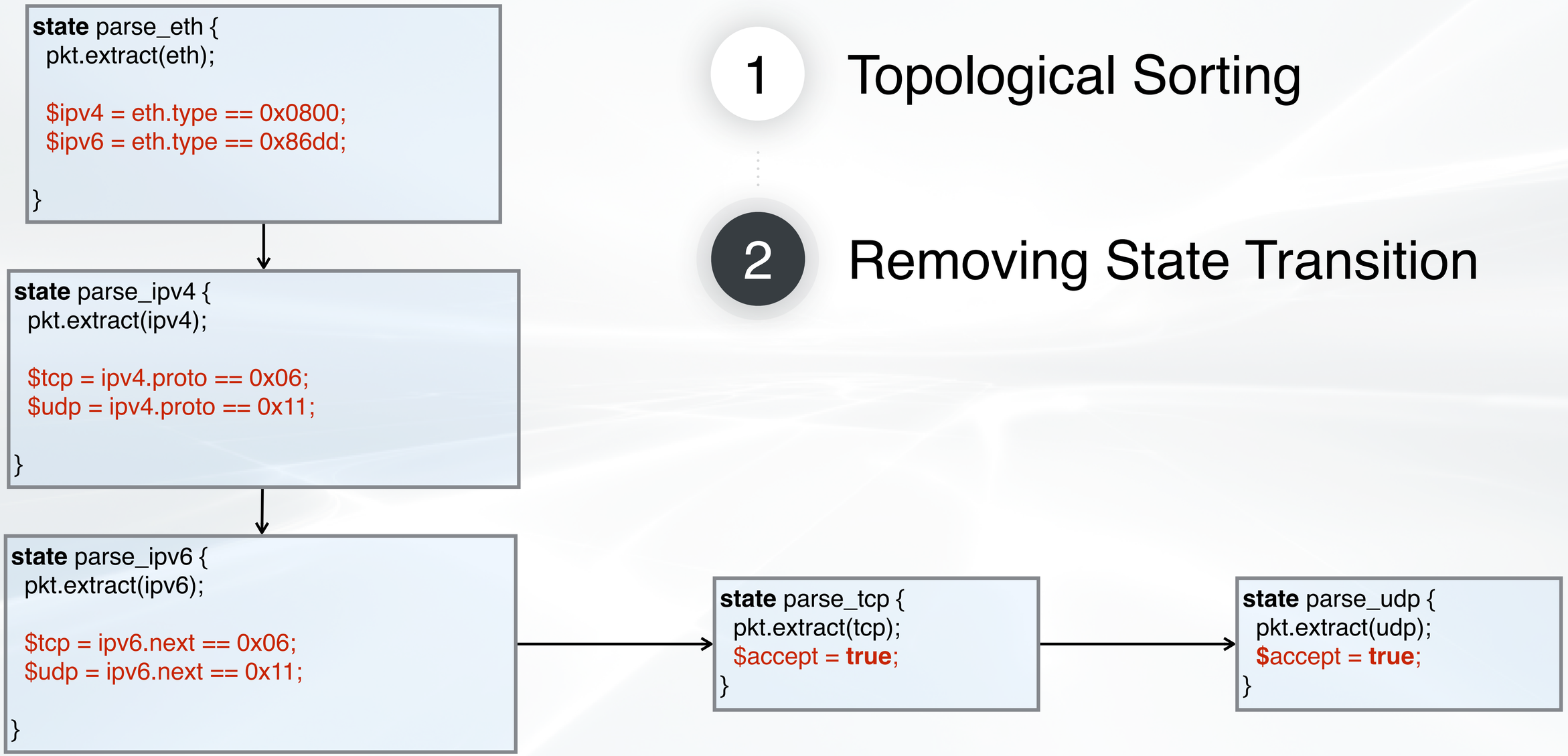


Example of Sequential Encoding



1 Topological Sorting

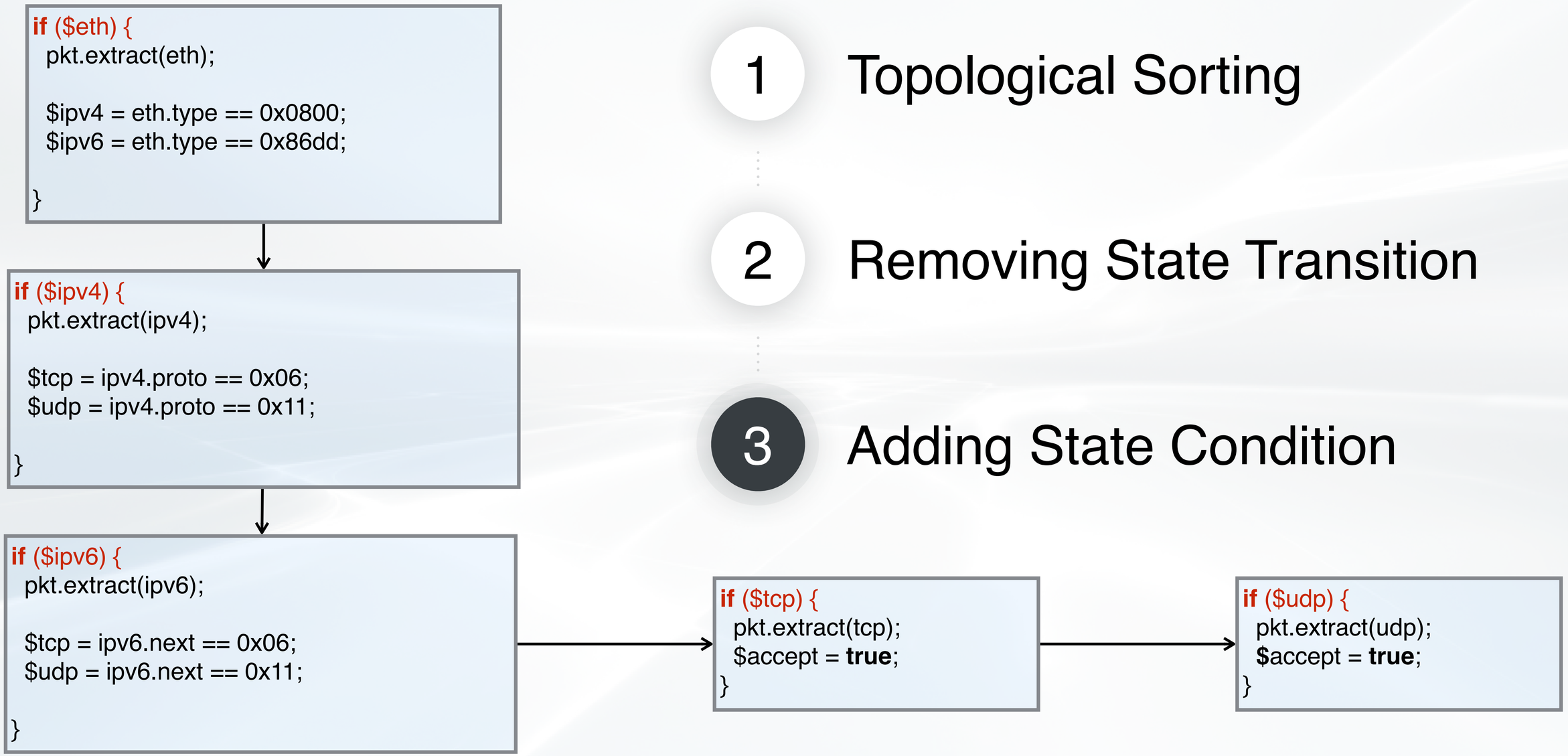
Example of Sequential Encoding



1 Topological Sorting

2 Removing State Transition

Example of Sequential Encoding



Example of Sequential Encoding

```
$eth = true;
if ($eth) {
  pkt.extract(eth);
  $ipv4 = eth.type == 0x0800;
  $ipv6 = eth.type == 0x86dd;
}
if ($ipv4) {
  pkt.extract(ipv4);
  $tcp = ipv4.proto == 0x06;
  $udp = ipv4.proto == 0x11;
}
if ($ipv6) {
  pkt.extract(ipv6);
  $tcp = ipv6.next == 0x06;
  $udp = ipv6.next == 0x11;
}
if ($tcp) {
  pkt.extract(tcp);
  $accept = true;
}
if ($udp) {
  pkt.extract(udp);
  $accept = true;
}
```

1. Topological Sorting
2. Removing State Transition
3. Adding State Condition
4. Simple Program

Other Encoding Details

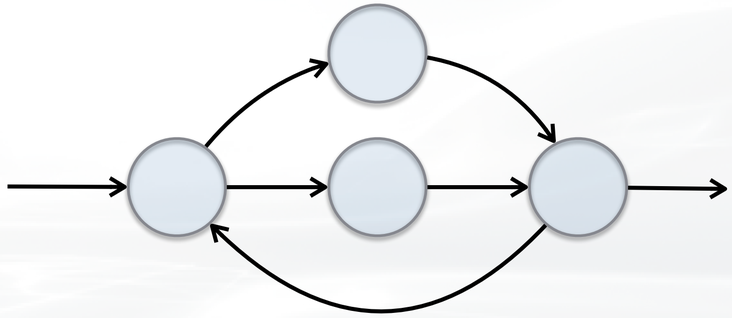
```
$eth = true;
if ($eth) {
  pkt.extract(eth);
  $ipv4 = eth.type == 0x0800;
  $ipv6 = eth.type == 0x86dd;
}
if ($ipv4) {
  pkt.extract(ipv4);
  $tcp = ipv4.proto == 0x06;
  $udp = ipv4.proto == 0x11;
}
if ($ipv6) {
  pkt.extract(ipv6);
  $tcp = ipv6.next == 0x06;
  $udp = ipv6.next == 0x11;
}
if ($tcp) {
  pkt.extract(tcp);
  $accept = true;
}
if ($udp) {
  pkt.extract(udp);
  $accept = true;
}
```

How to encode external functions?

Other Encoding Details

```
$eth = true;
if ($eth) {
  pkt.extract(eth);
  $ipv4 = eth.type == 0x0800;
  $ipv6 = eth.type == 0x86dd;
}
if ($ipv4) {
  pkt.extract(ipv4);
  $tcp = ipv4.proto == 0x06;
  $udp = ipv4.proto == 0x11;
}
if ($ipv6) {
  pkt.extract(ipv6);
  $tcp = ipv6.next == 0x06;
  $udp = ipv6.next == 0x11;
}
if ($tcp) {
  pkt.extract(tcp);
  $accept = true;
}
if ($udp) {
  pkt.extract(udp);
  $accept = true;
}
```

How to encode external functions?



How to handle parser loops?

Please see our paper for more details!

Experiments & Experience

Benchmark: Invalid Header Access

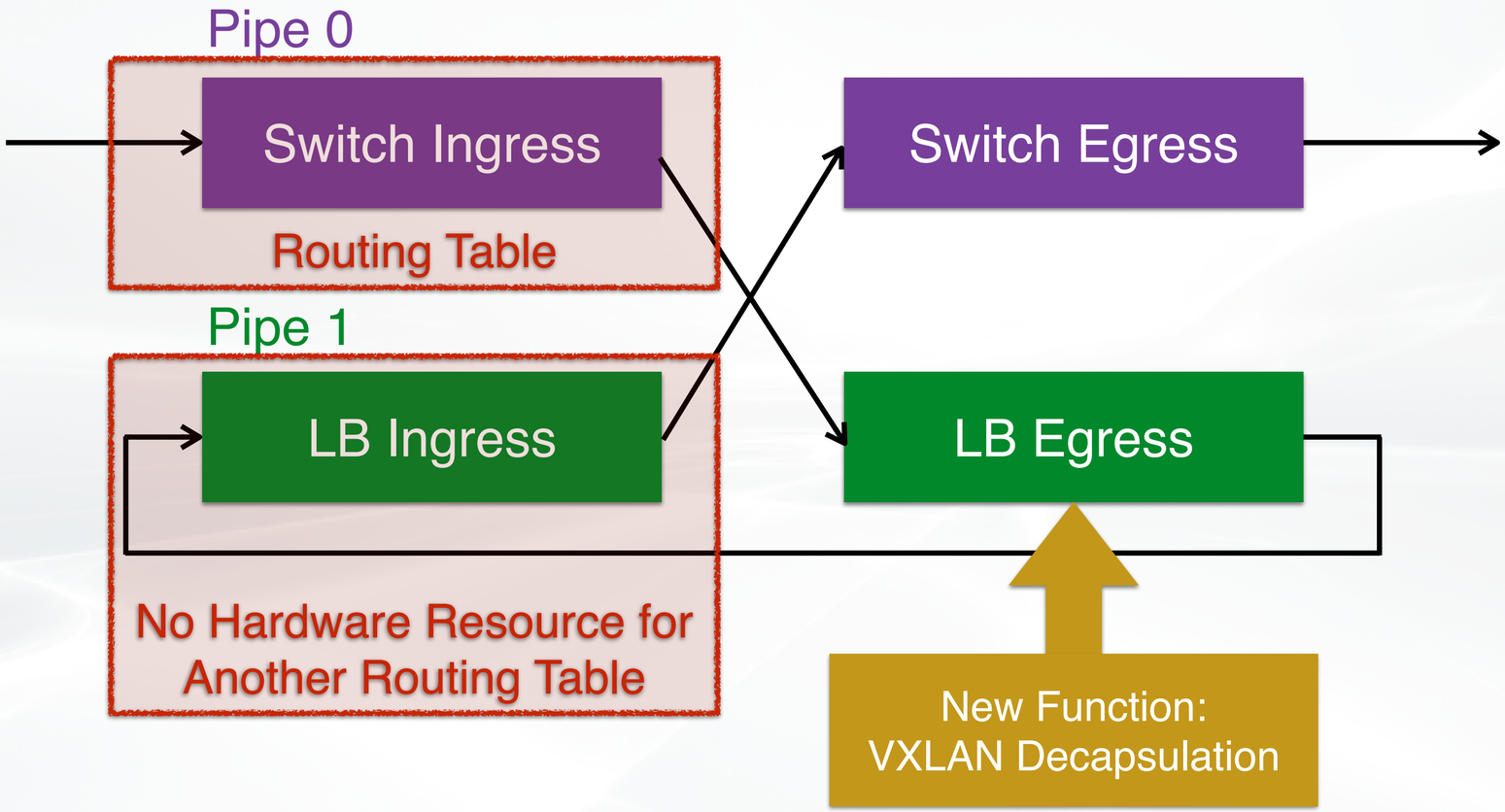
	LoC	Pipelines	Parser States	Find First		Find All	
				Aquila	Vera	Aquila	Vera
BMv2 Switch (w/o INT)	~ 5.0k	1	59	1 s	13 s	5 min	> 2 hr
BMv2 Switch	~ 5.5k	1	64	1 s	4 min	6 min	> 2 hr
Switch from Vendor	~ 5.5k	2	30	20 s	Error	21 min	Error
Production Program 1	> 6.0k	4	41	24 s	Error	9 min	Error
Production Program 2	> 6.0k	4	47	25 s	Error	12 min	Error
Production Program 3	> 2.0k	6	114	41 s	Error	60 min	Error

Benchmark: Invalid Header Access

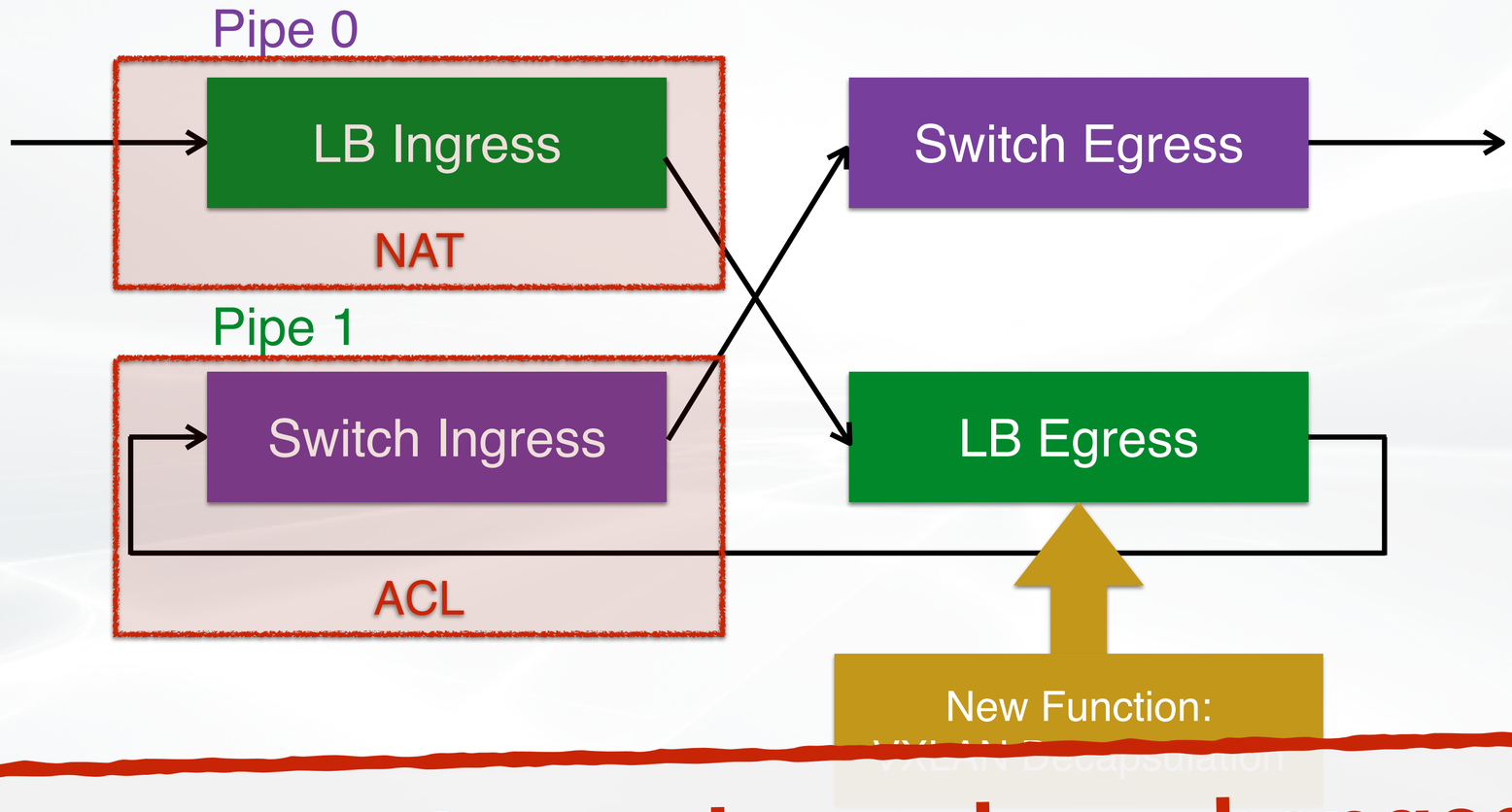
	LoC	Pipelines	Parser States	Find First		Find All	
				Aquila	Vera	Aquila	Vera
BMv2 Switch (w/o INT)	~ 5.0k	1	59	1 s	13 s	5 min	> 2 hr
BMv2 Switch	~ 5.5k	1	64	1 s	4 min	6 min	> 2 hr
Switch from Vendor	~ 5.5k	2	30	20 s	Error	21 min	Error
Production Program 1	> 6.0k	4	41	24 s	Error	9 min	Error
Production Program 2	> 6.0k	4	47	25 s	Error	12 min	Error
Production Program 3	> 2.0k	6	114	41 s	Error	60 min	Error

Aquila scales well!

Experience: Swapping Two Pipelines

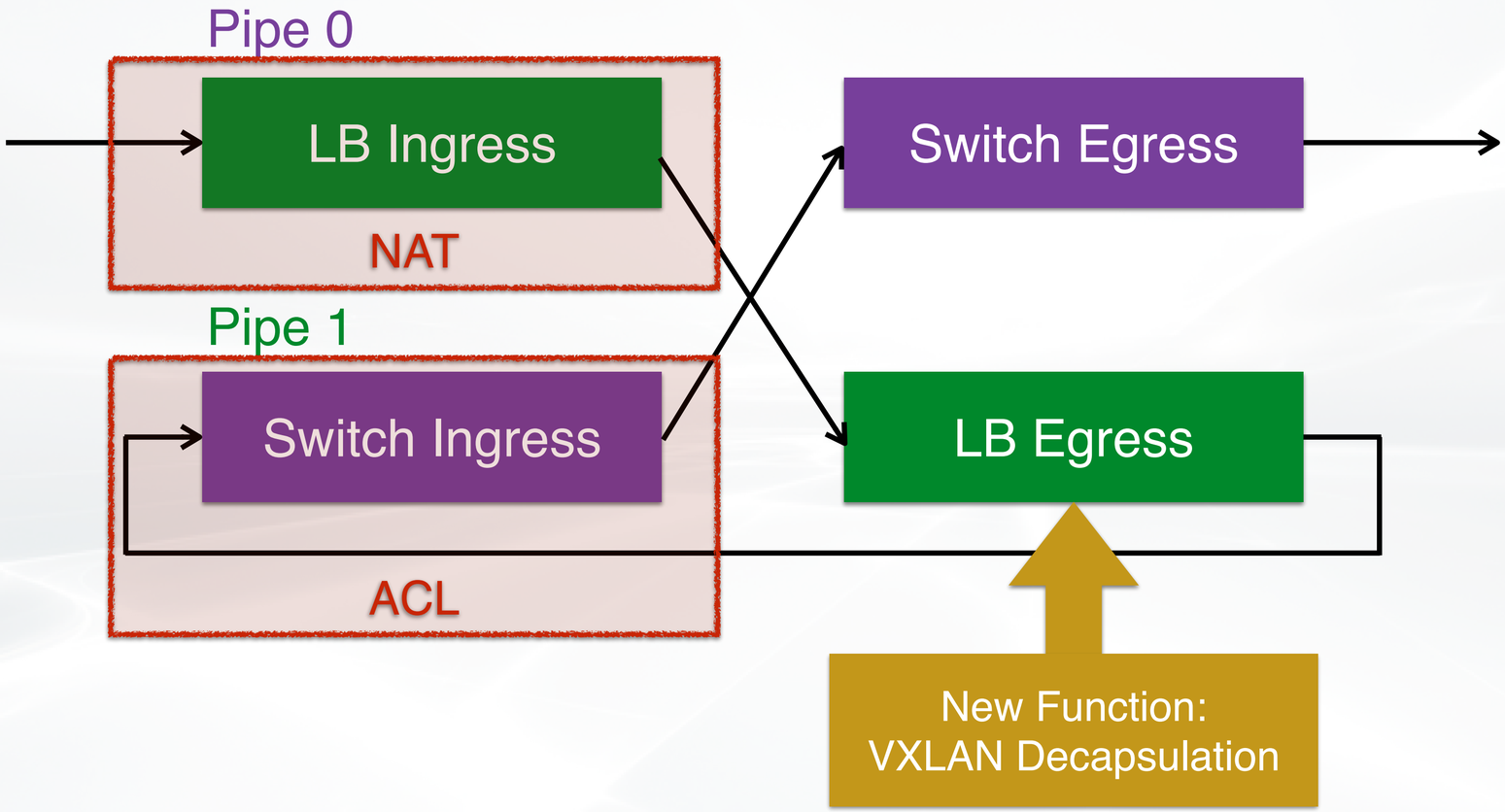


Experience: Swapping Two Pipelines



Pipeline dependency has changed!

Experience: Swapping Two Pipelines



Aquila is practically usable!

Conclusion

- Aquila is the first practically usable verification system for production-scale programmable data planes.
- Aquila provides a high-level intent language, a fast and scalable verifier, a bug localizer, and self-correctness guarantee.
- Aquila has been used in Alibaba for more than one year, and found several bugs in data planes programs.

Thanks!