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

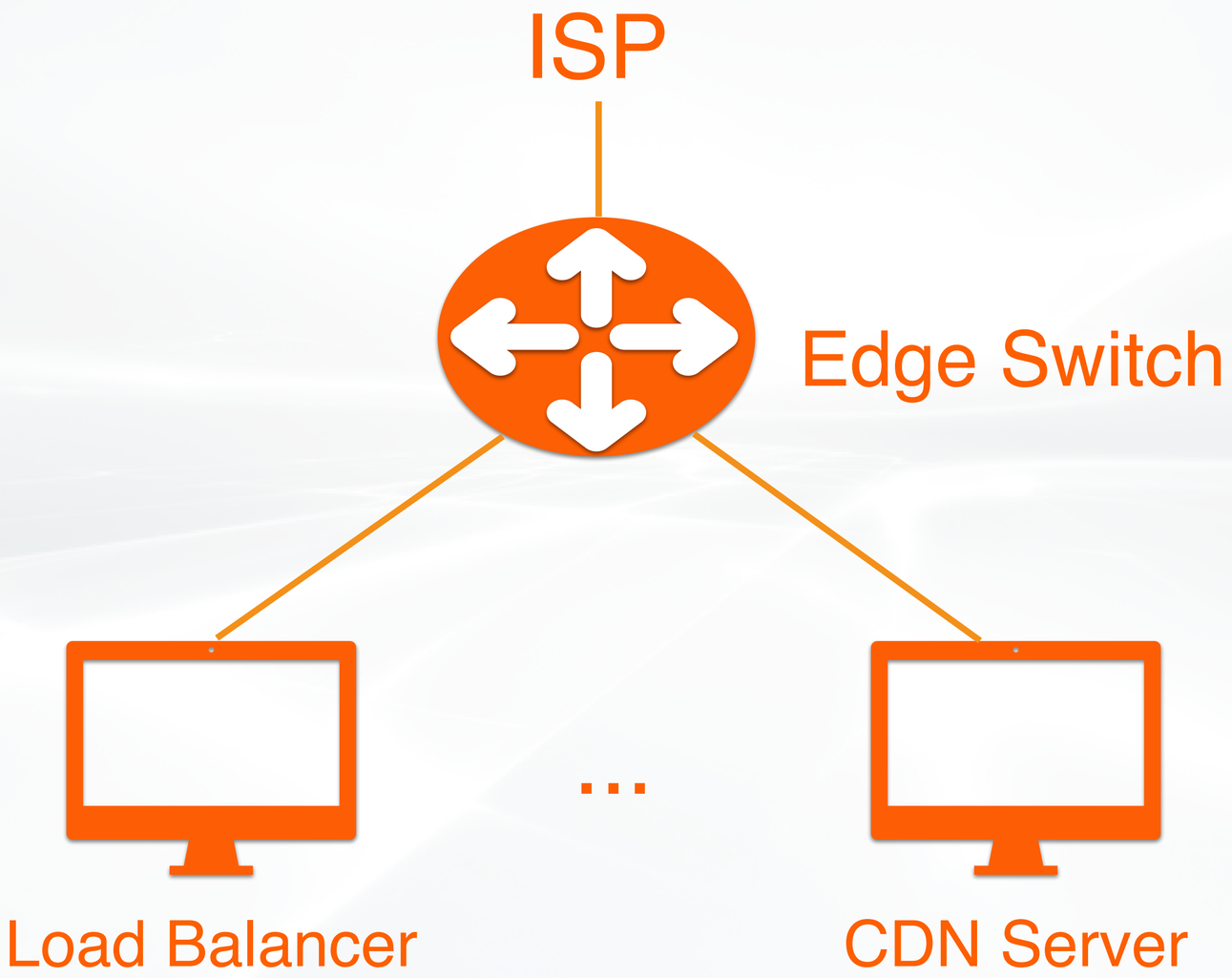
淘宝网
Taobao.com

天猫
TMALL

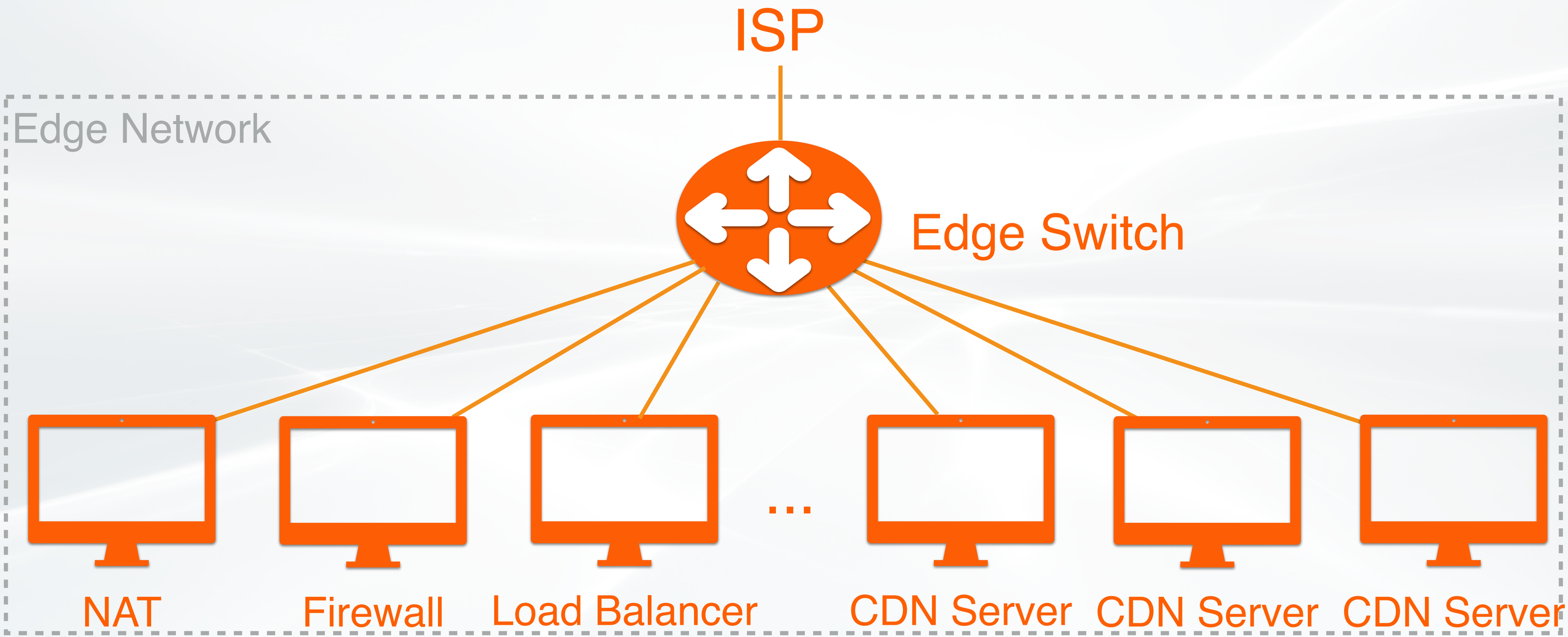
支付宝
ALIPAY

Alibaba Cloud

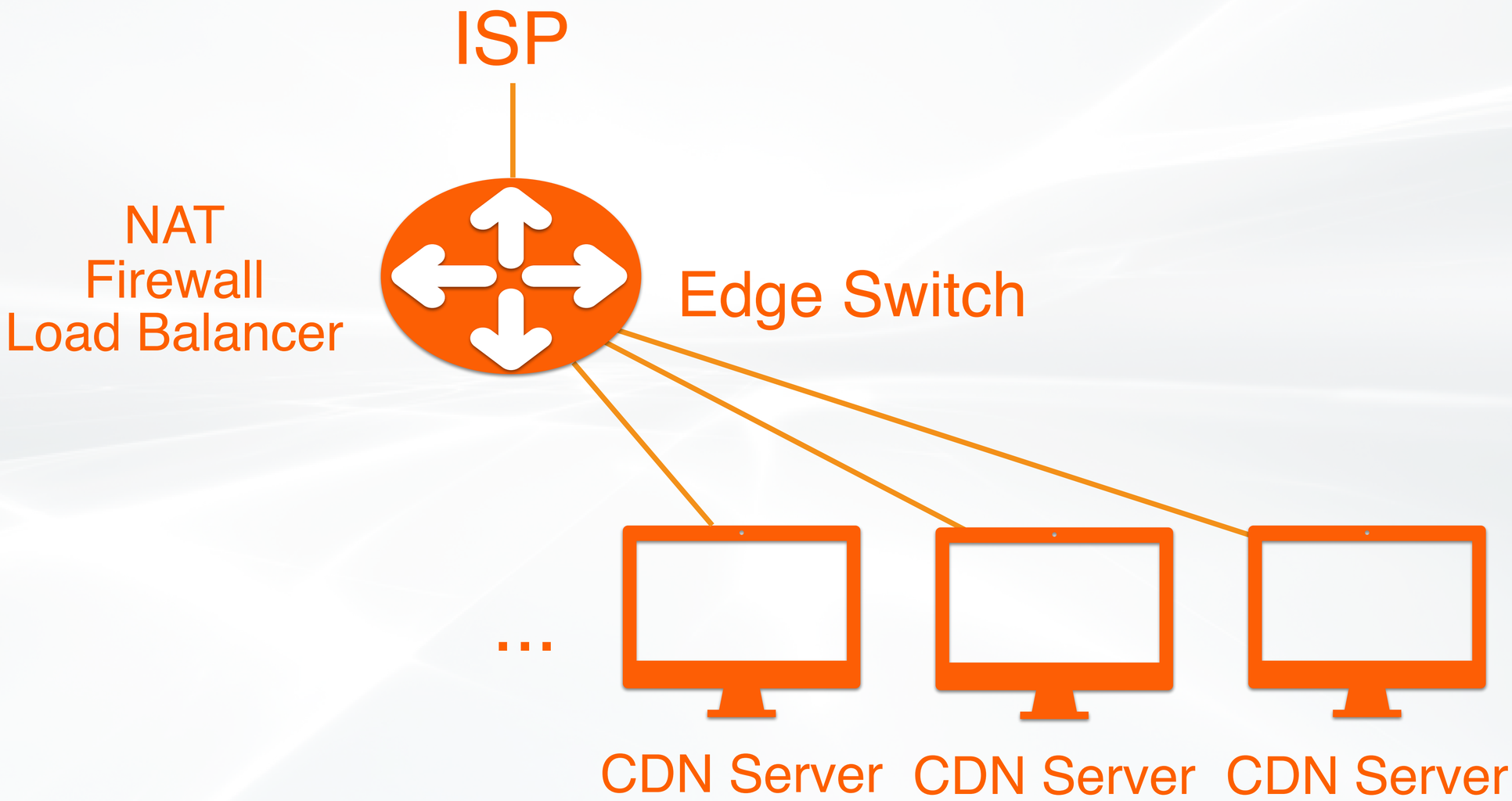
Alibaba leads the deployment of programmable switches in the industry



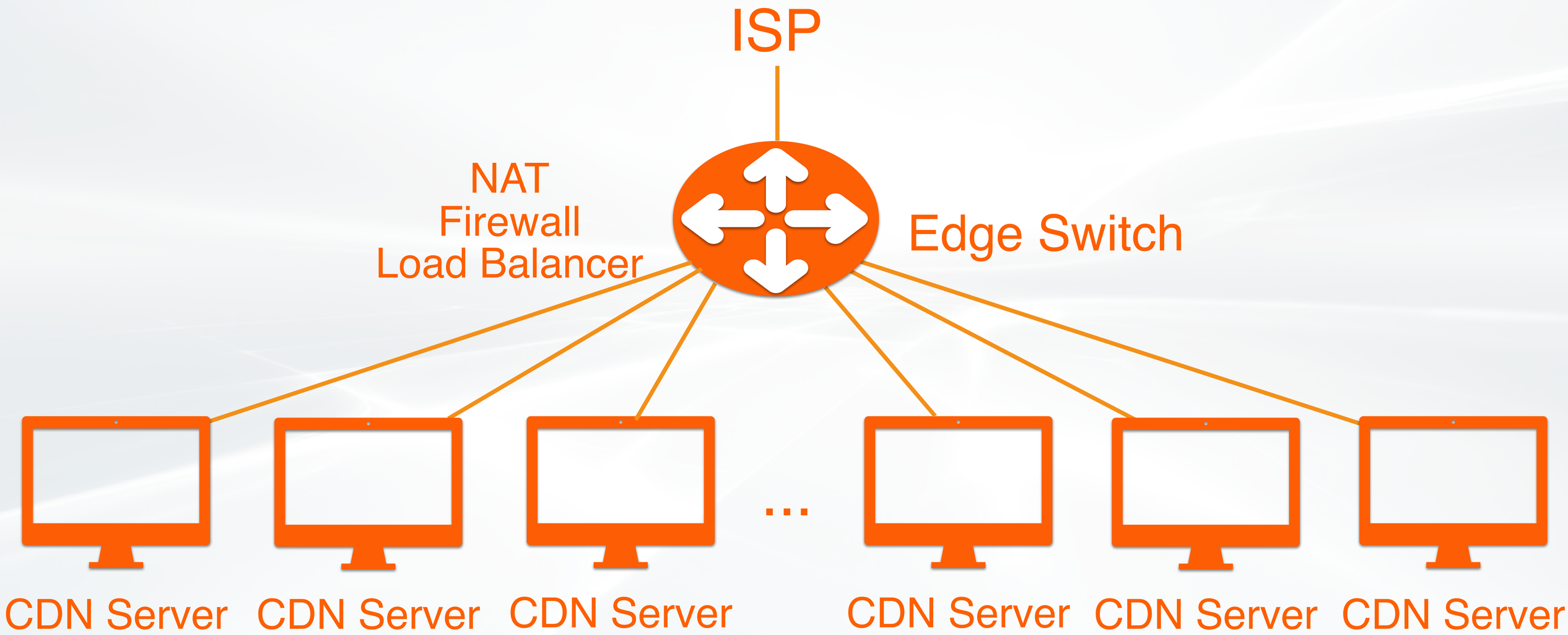
Alibaba leads the deployment of programmable switches in the industry



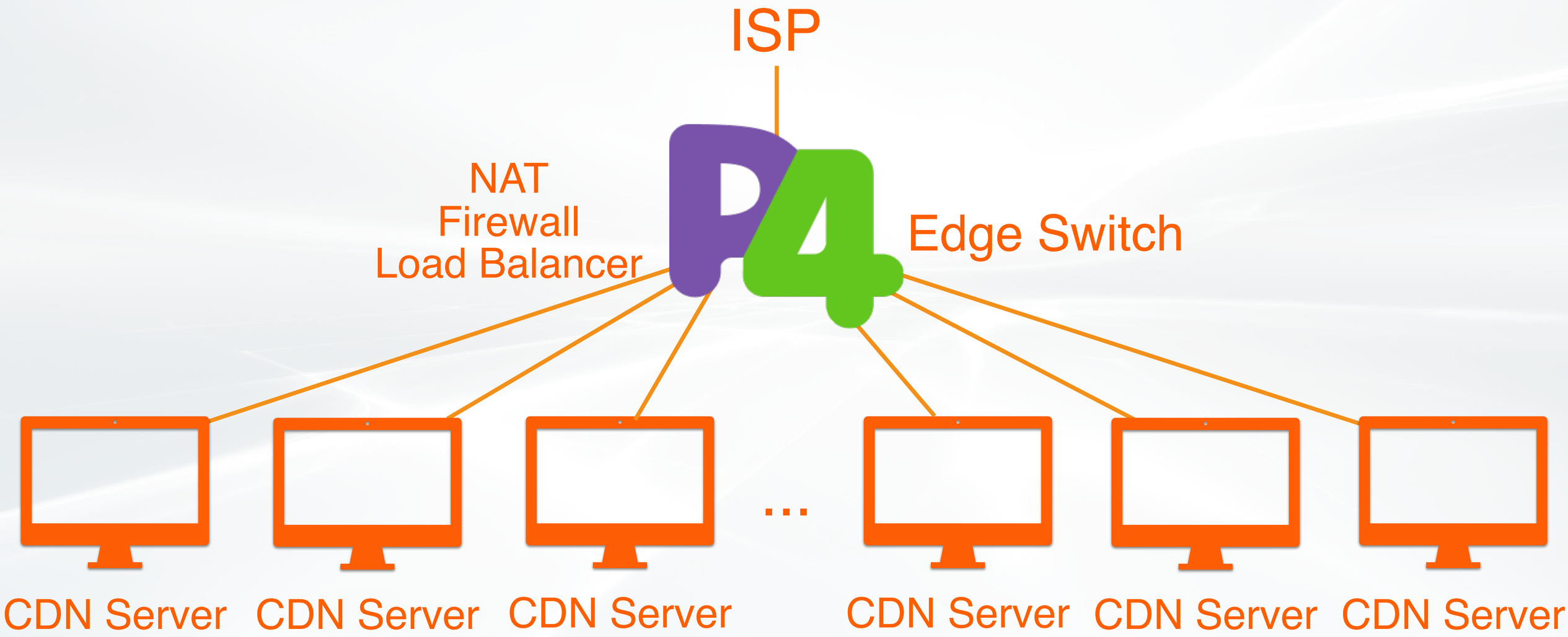
Alibaba leads the deployment of programmable switches in the industry



Alibaba leads the deployment of programmable switches in the industry



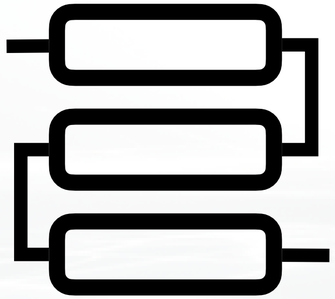
Alibaba leads the deployment of programmable switches in the industry



Alibaba leads the deployment of programmable switches in the industry

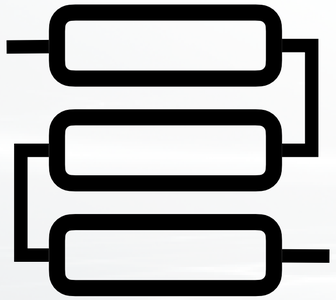
- Functionality
- Efficiency
- Flexibility

Ensuring the correctness of data plane programs is challenging

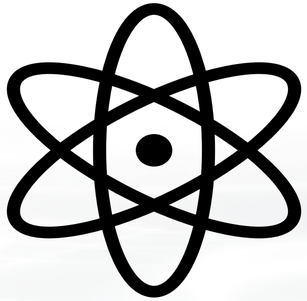


Multiple Pipelines

Ensuring the correctness of data plane programs is challenging



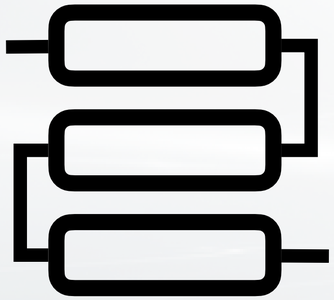
Multiple Pipelines



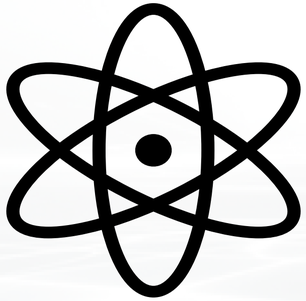
Many Network Functions

* NAT
* Load Balancer
* DDoS Defense

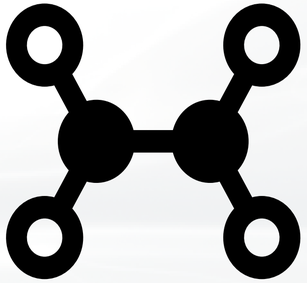
Ensuring the correctness of data plane programs is challenging



Multiple Pipelines

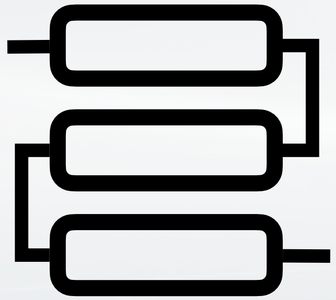


Many Network Functions

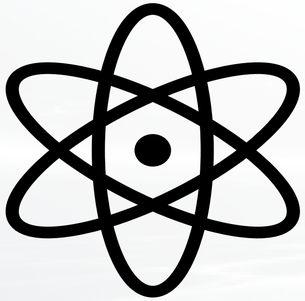


Various Packet Paths

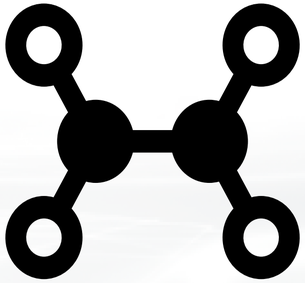
Ensuring the correctness of data plane programs is challenging



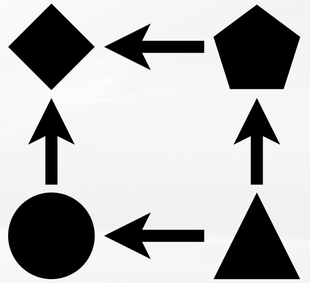
Multiple Pipelines



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

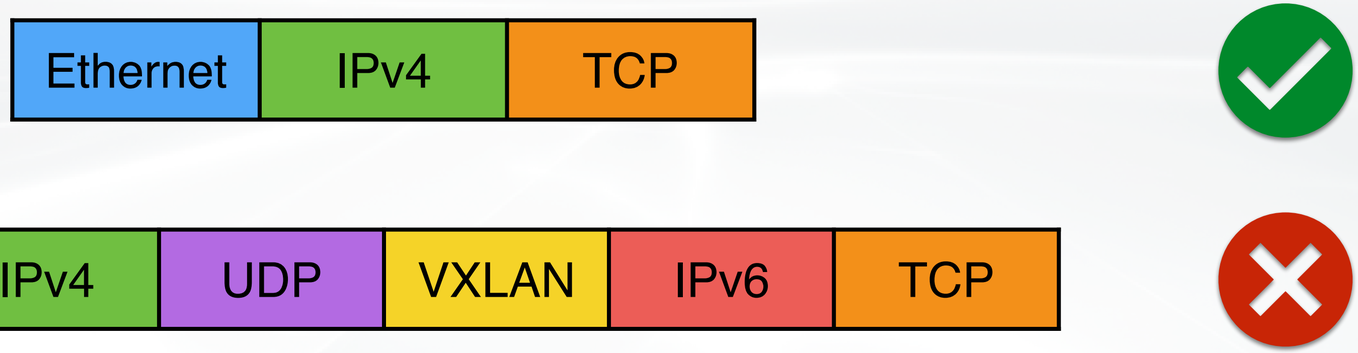
Goal 2: Scalable Verification Algorithm → Challenge 2

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)))
...
```

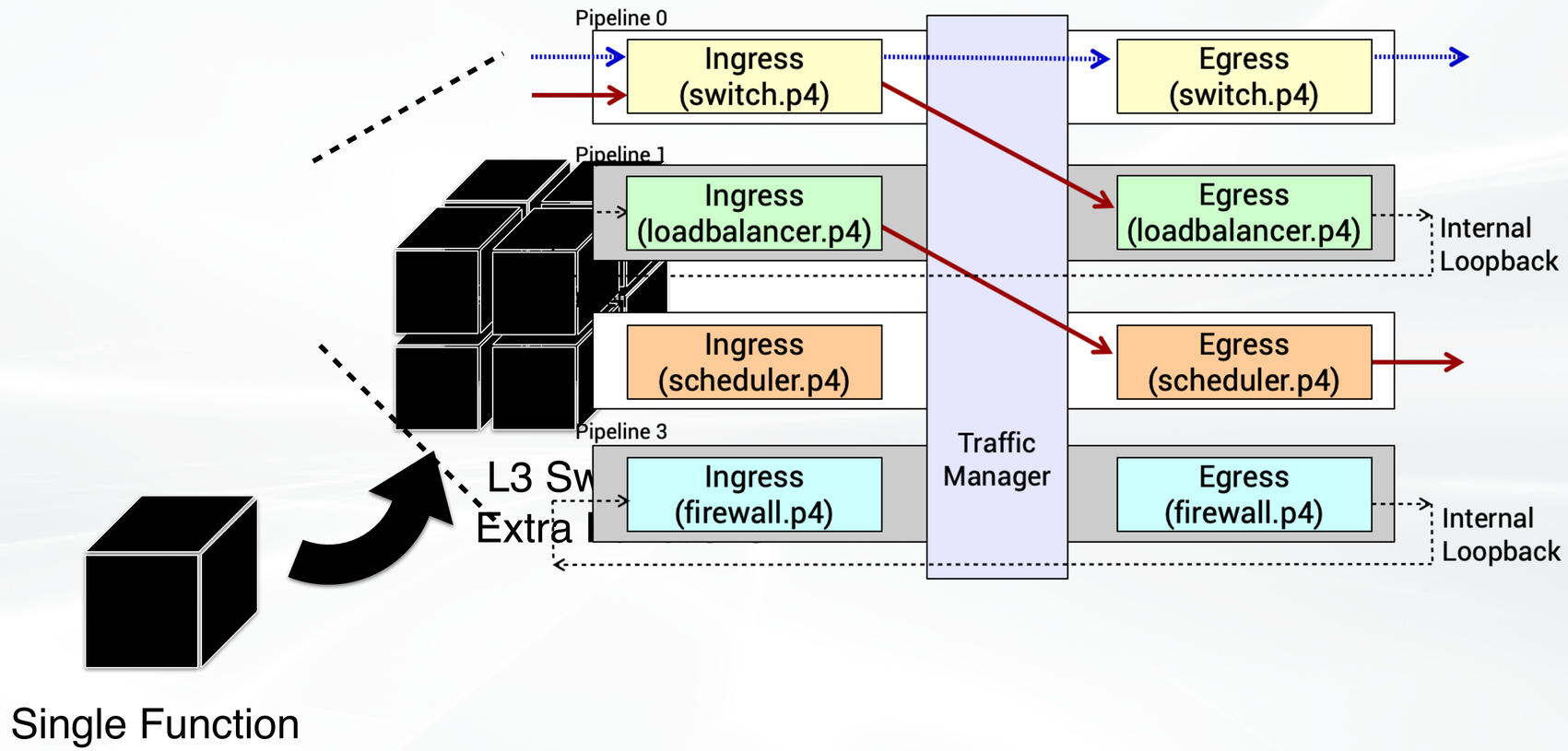
Describing header order constraints

Enumerating header orders

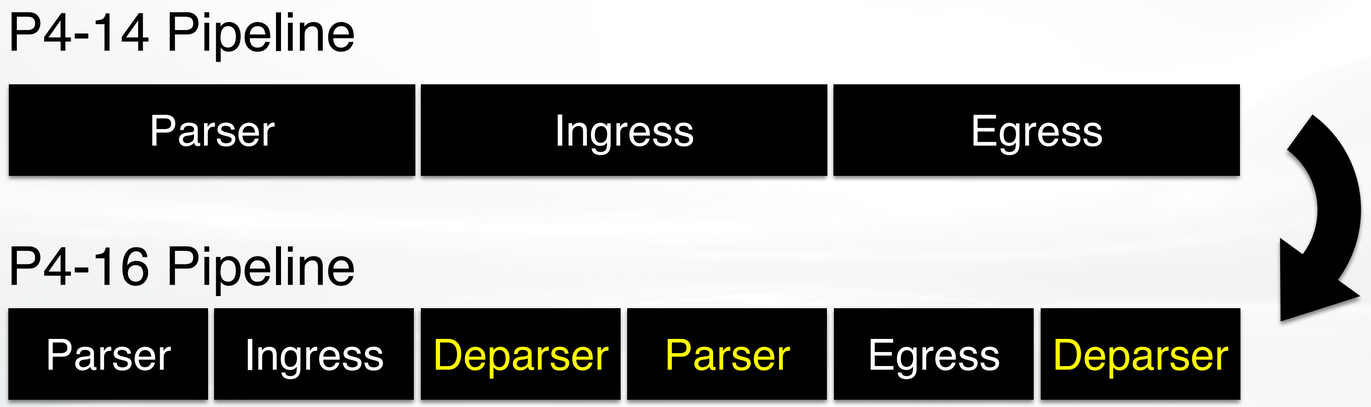
@pkt.$order == <eth [vlan] (ipv4|ipv6) tcp> => keep(tcp)

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
…

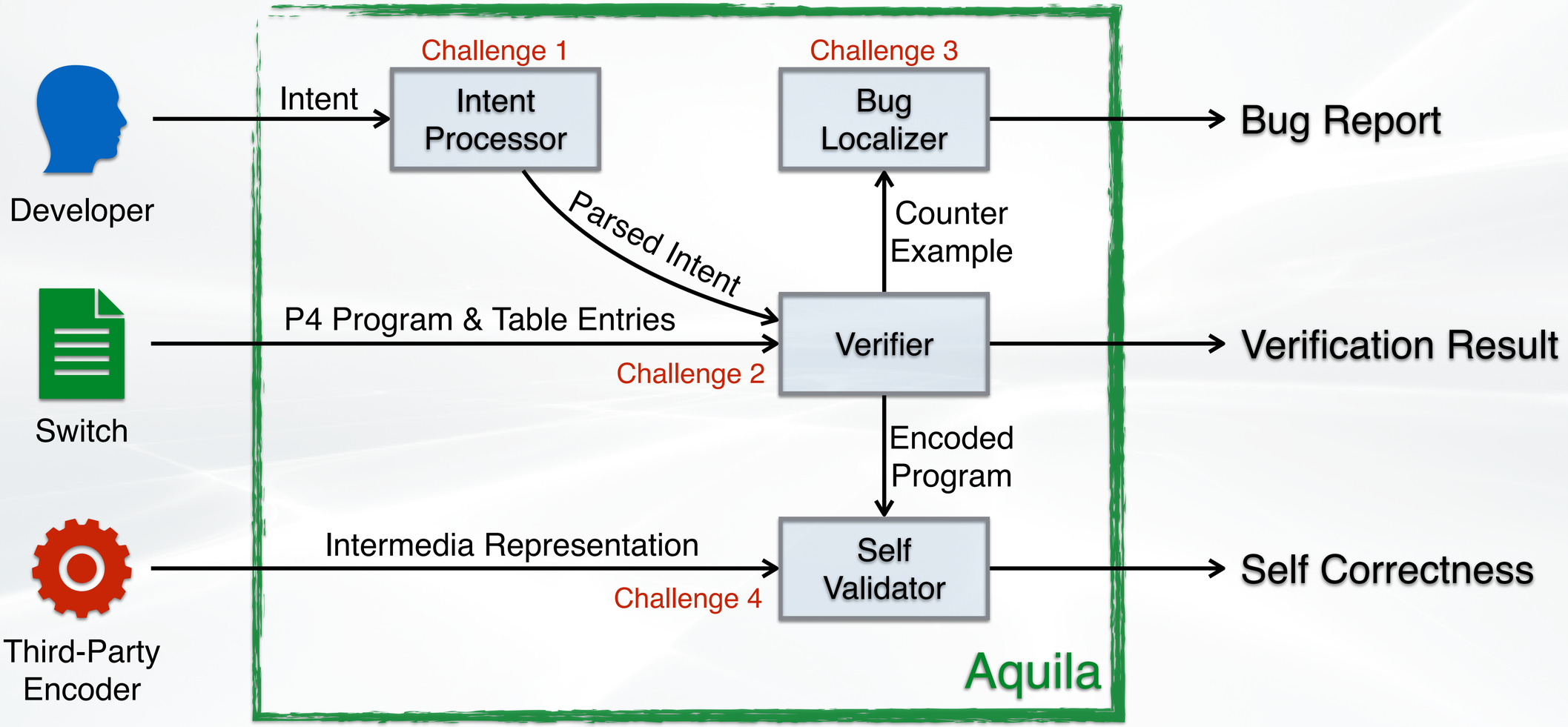
Trace:
[1] validate.p4(10) validate_eth
[2] validate.p4(50) validate_ipv4
[3] l3.p4(20) ipv4_fib
[4] acl.p4(30) ipv4_acl
[5] acl.p4(80) system_acl
...

Preferred

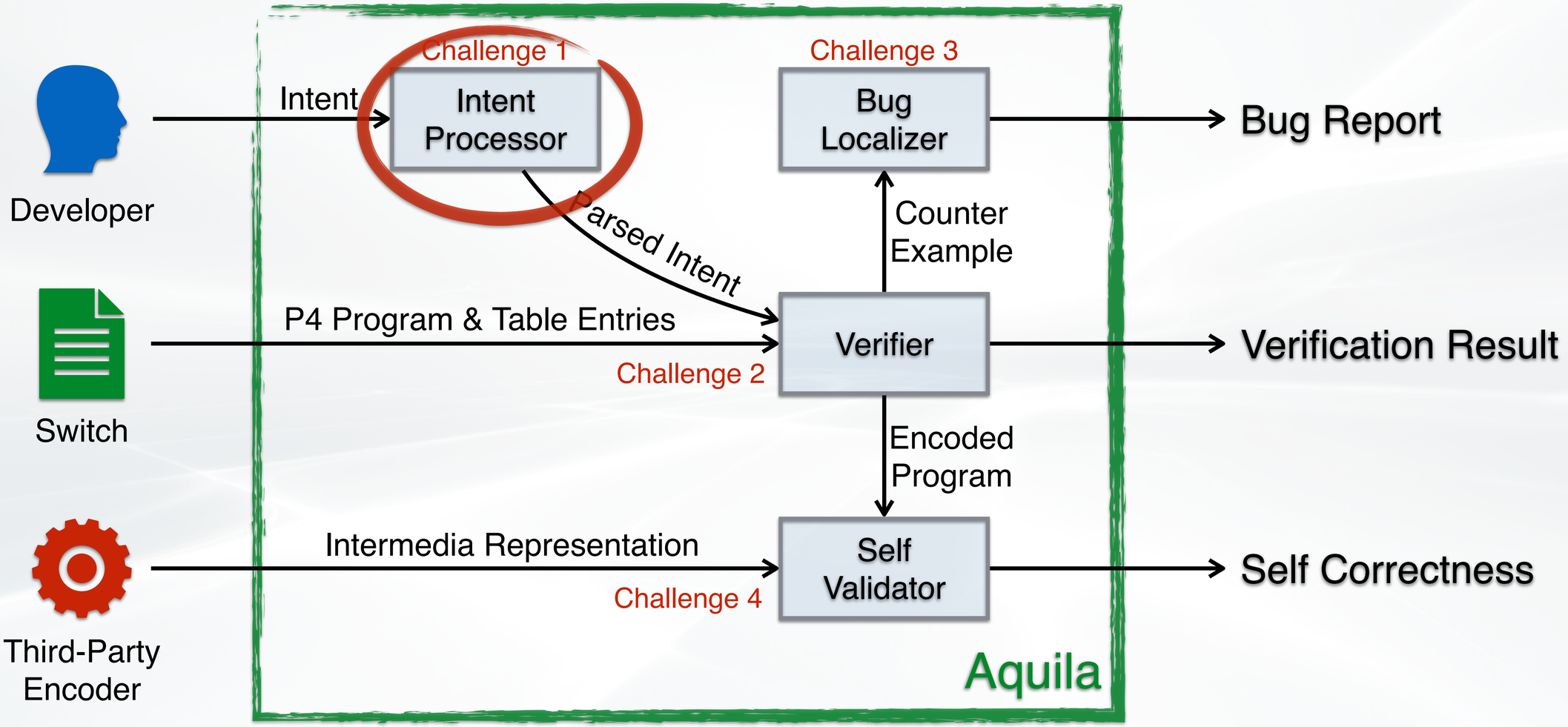
Challenge 4: Reliability of Verifier

No existing work can do self-validation

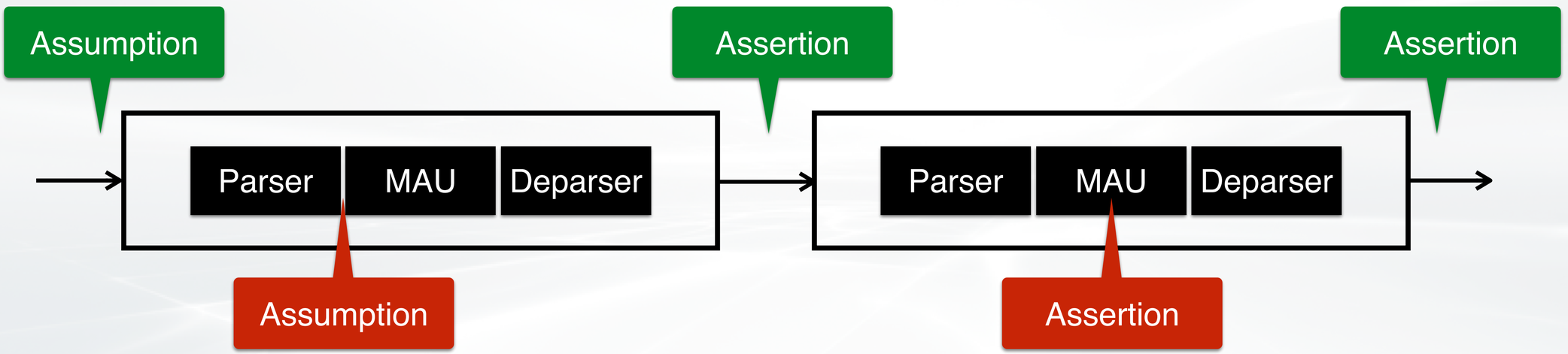
How to make a P4 verifier practically usable?



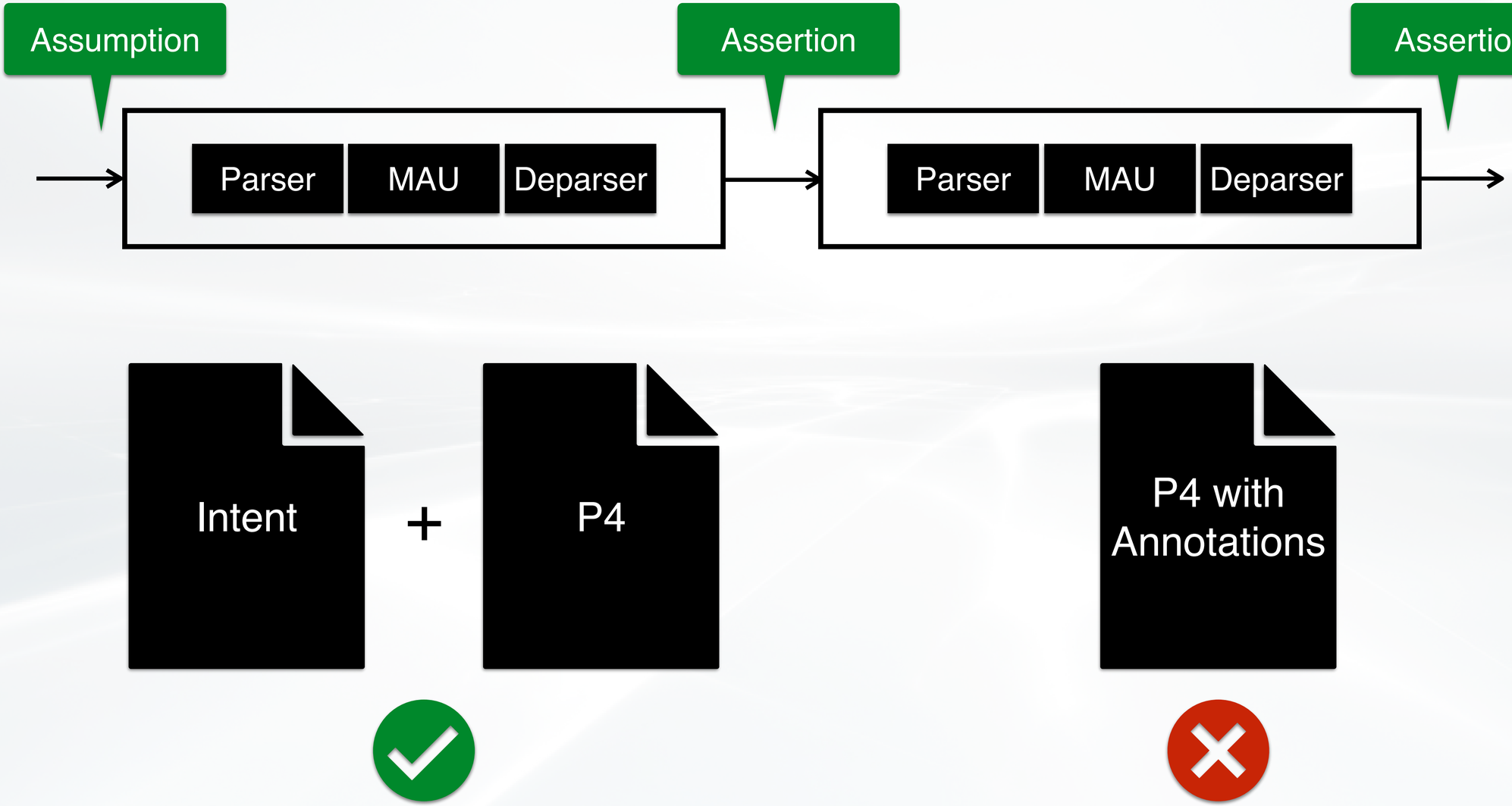
Aquila Architecture



Insights of Intent Language Design



Insights of Intent Language Design



Intent Language

```
program {
  assume(init);
  call(a.ingress);
  assert(ig);
  #quit = ig_md.drop > 0
      || ig_md.to_cpu > 0;
  if (!#quit) {
    call(a.egress);
    assert(eg);
  }
}

pipeline {
  a.ingress {
    id = 0, 2;
    snapshot = "a.ig.json";
  }
  a.egress {
    id = 0, 2;
    snapshot = "a.eg.json";
  }
}
```

```
assumption {
  init {
    ig_md.ingress_port < 8;
    pkt.$order == <eth ipv4 (tcp | udp)>;
    pkt.ipv4.dst_ip == 10.0.0.0/24;
  }
}

assertion {
  ig {
    if (@pkt.ipv4.prototol == 6)
      pkt.ipv4.dst_ip == 10.0.0.1;
    if (match(fwd, send))
      modified(pkt.ipv4.dst_ip);
  }
  eg { ... }
}

config {
  source = "forward.p4";
  ...
}
```

Intent Language

```
program {
  assume(init);
  call(a.ingress);
  assert(ig);
  #quit = ig_md.drop > 0
      ll ig_md.to_cpu > 0;
  if (!#quit) {
    call(a.egress);
    assert(eg);
  }
}
```

Execution Paths

```
pipeline {
  a.ingress {
    id = 0, 2;
    snapshot = "a.ig.json";
  }
  a.egress {
    id = 0, 2;
    snapshot = "a.eg.json";
  }
}
```

```
assumption {
  init {
    ig_md.ingress_port < 8;
    pkt.$order == <eth ipv4 (tcp l udp)>;
    pkt.ipv4.dst_ip == 10.0.0.0/24;
  }
}
```

```
assertion {
  ig {
    if (@pkt.ipv4.prototol == 6)
      pkt.ipv4.dst_ip == 10.0.0.1;
    if (match(fwd, send))
      modified(pkt.ipv4.dst_ip);
  }
  eg { ... }
}
```

```
config {
  source = "forward.p4";
  ...
}
```

Intent Language

```
program {
  assume(init);
  call(a.ingress);
  assert(ig);
  #quit = ig_md.drop > 0
      ll ig_md.to_cpu > 0;
  if (!#quit) {
    call(a.egress);
    assert(eg);
  }
}

pipeline {
  a.ingress {
    id = 0, 2;
    snapshot = "a.ig.json";
  }
  a.egress {
    id = 0, 2;
    snapshot = "a.eg.json";
  }
}
```

Execution Paths

```
assumption {
  init {
    ig_md.ingress_port < 8;
    pkt.$order == <eth ipv4 (tcp l udp)>;
    pkt.ipv4.dst_ip == 10.0.0.0/24;
  }
}

assertion {
  ig {
    if (@pkt.ipv4.prototol == 6)
      pkt.ipv4.dst_ip == 10.0.0.1;
    if (match(fwd, send))
      modified(pkt.ipv4.dst_ip);
  }
  eg { ... }
}

config {
  source = "forward.p4";
  ...
}
```

Intent Language

```
program {
  assume(init);
  call(a.ingress);
  assert(ig);
  #quit = ig_md.drop > 0
     || ig_md.to_cpu > 0;
  if (!#quit) {
    call(a.egress);
    assert(eg);
  }
}

pipeline {
  a.ingress {
    id = 0, 2;
    snapshot = "a.ig.json";
  }
  a.egress {
    id = 0, 2;
    snapshot = "a.eg.json";
  }
}
```

Execution Paths

```
assumption {
  init {
    ig_md.ingress_port < 8;
    pkt.$order == <eth ipv4 (tcp | udp)>;
    pkt.ipv4.dst_ip == 10.0.0.0/24;
  }
}

assertion {
  ig {
    if (@pkt.ipv4.prototol == 6)
      pkt.ipv4.dst_ip == 10.0.0.1;
    if (match(fwd, send))
      modified(pkt.ipv4.dst_ip);
  }
  eg { ... }
}

config {
  source = "forward.p4";
  ...
}
```

Intent Language

```
program {
  assume(init);
  call(a.ingress);
  assert(ig);
  #quit = ig_md.drop > 0
      || ig_md.to_cpu > 0;
  if (!#quit) {
    call(a.egress);
    assert(eg);
  }
}

pipeline {
  a.ingress {
    id = 0, 2;
    snapshot = "a.ig.json";
  }
  a.egress {
    id = 0, 2;
    snapshot = "a.eg.json";
  }
}
```

Execution Paths

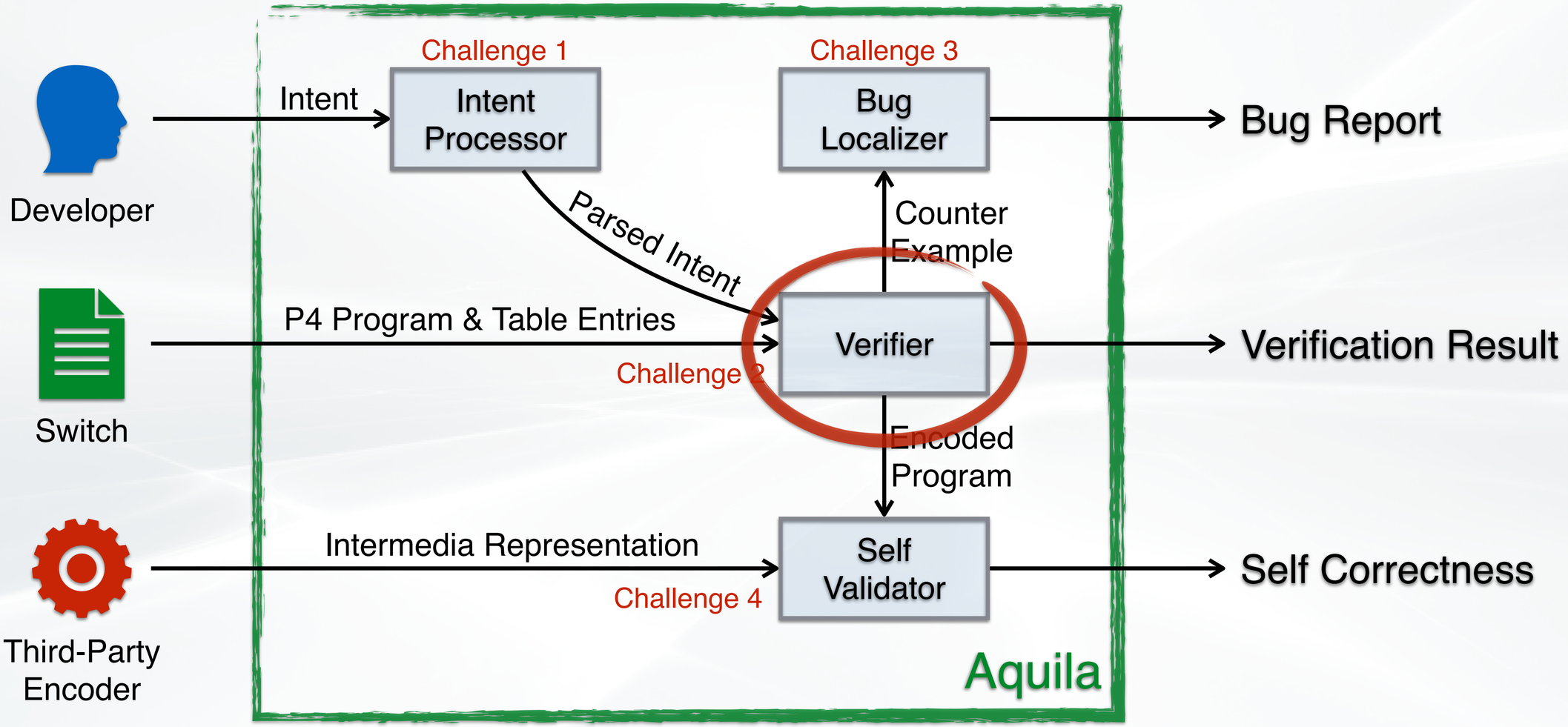
```
assumption {
  init {
    ig_md.ingress_port < 8;
    pkt.$order == <eth ipv4 (tcp | udp)>;
    pkt.ipv4.dst_ip == 10.0.0.0/24;
  }
}

assertion {
  ig {
    if (@pkt.ipv4.prototol == 6)
      pkt.ipv4.dst_ip == 10.0.0.1;
    if (match(fwd, send))
      modified(pkt.ipv4.dst_ip);
  }
  eg { ... }
}

config {
  source = "forward.p4";
  ...
}
```

Intent Language

```
program {
  assume(init);
  call(a.ingress);
  assert(ig);
  #quit = ig_md.drop > 0
      ll ig_md.to_cpu > 0;
  if (!#quit) {
    call(a.egress);
    assert(eg);
  }
}

pipeline {
  a.ingress {
    id = 0, 2;
    snapshot = "a.ig.json";
  }
  a.egress {
    id = 0, 2;
    snapshot = "a.eg.json";
  }
}
```

```
assumption {
  init {
    ig_md.ingress_port < 8;
    pkt.$order == <eth ipv4 (tcp l udp)>;
    pkt.ipv4.dst_ip == 10.0.0.0/24;
  }
}

assertion {
  ig {
    if (@pkt.ipv4.prototol == 6)
      pkt.ipv4.dst_ip == 10.0.0.1;
    if (match(fwd, send))
      modified(pkt.ipv4.dst_ip);
  }
  eg { ... }
}

config {
  source = "forward.p4";
  ...
}
```

Aquila Architecture



A P4-16 Program

```
header eth_t {
  bit<48> dstAddr;
  bit<48> srcAddr;
  bit<16> type;
}

header ipv4_t {
  bit<4> version;
  ...
  bit<32> dstAddr;
}

struct headers {
  eth_t eth;
  ipv4_t ipv4;
  ...
}
```

```
parser Parser(pkt_in pkt, headers hdr) {
  state start {
    pkt.extract(hdr.eth);
    transition select (hdr.eth.type) {
      0x0800: parse_ipv4;
      default: accept;
    }
  }
  state parse_ipv4 {
    pkt.extract(hdr.ipv4);
    transition accept;
  }
}
control Depsr(pkt_out pkt, headers hdr) {
  apply {
    pkt.emit(hdr.eth);
    pkt.emit(hdr.ipv4);
  }
}
```

```
control MAU(headers hdr, metadata_t md) {
  action drop() { md.drop = 1; }
  action forward(bit<9> port) {
    md.egress_spec = port;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
  }
  table routing {
    key = { hdr.ipv4.dstAddr: lpm; }
    actions = { forward; drop; }
  }
  apply {
    if (hdr.ipv4.isValid() &&
        hdr.ipv4.ttl > 0) {
      routing.apply();
    } else {
      drop();
    }
  }
}
```

Headers & Metadata Parsers & Deparsers Match-Action Units

Key Complexity of P4-16 Encoding

```
header eth_t {
  bit<48> dstAddr;
  bit<48> srcAddr;
  bit<16> type;
}

header ipv4_t {
  bit<4> version;
  ...
  bit<32> dstAddr;
}

struct headers {
  eth_t eth;
  ipv4_t ipv4;
  ...
}
```

```
parser Parser(pkt_in pkt, headers hdr) {
  state start {
    pkt.extract(hdr.eth);
    transition select (hdr.eth.type) {
      0x0800: parse_ipv4;
      default: accept;
    }
  }
  state parse_ipv4 {
    pkt.extract(hdr.ipv4);
    transition accept;
  }
}
control Depsr(pkt_out pkt, headers hdr) {
  apply {
    pkt.emit(hdr.eth);
    pkt.emit(hdr.ipv4);
  }
}
```

(1) State-Machine Language

```
control MAU(headers hdr, metadata_t md) {
  action drop() { md.drop = 1; }
  action forward(bit<9> port) {
    md.egress_spec = port;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
  }
  table routing {
    key = { hdr.ipv4.dstAddr: lpm; }
    actions = { forward; drop; }
  }
  apply {
    if (hdr.ipv4.isValid() &&
        hdr.ipv4.ttl > 0) {
      routing.apply();
    } else {
      drop();
    }
  }
}
```

(2) Extremely Branchy Entries

Headers & Metadata

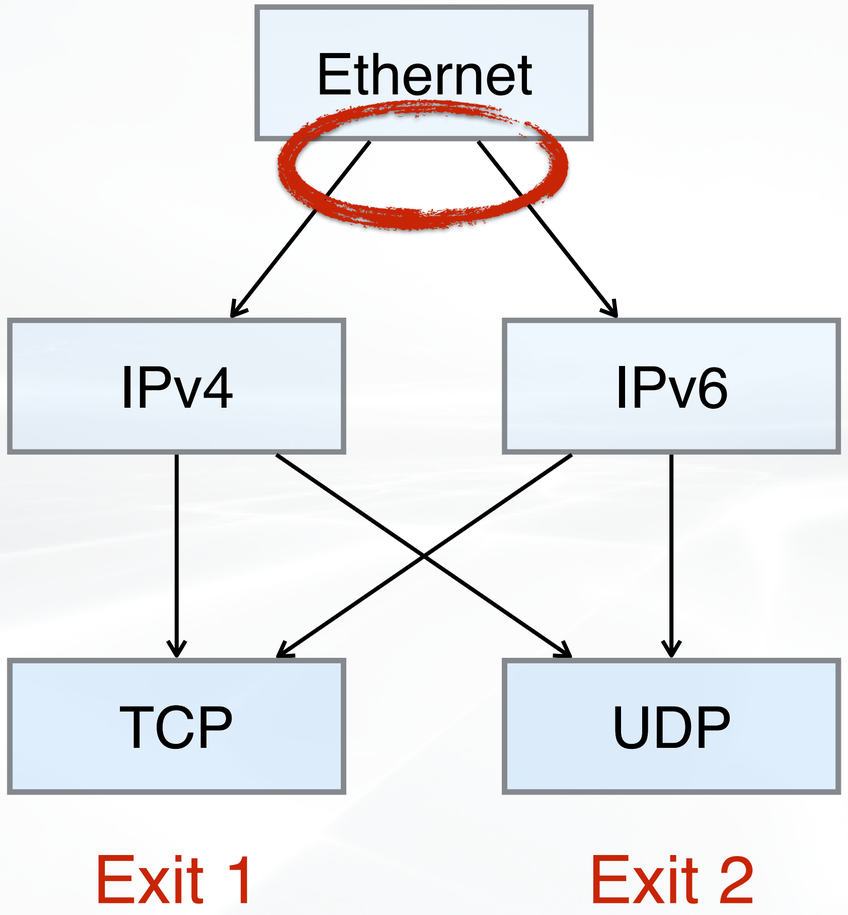
Parsers & Deparsers

Match-Action Units

(3) Connection Between Multiple Pipelines

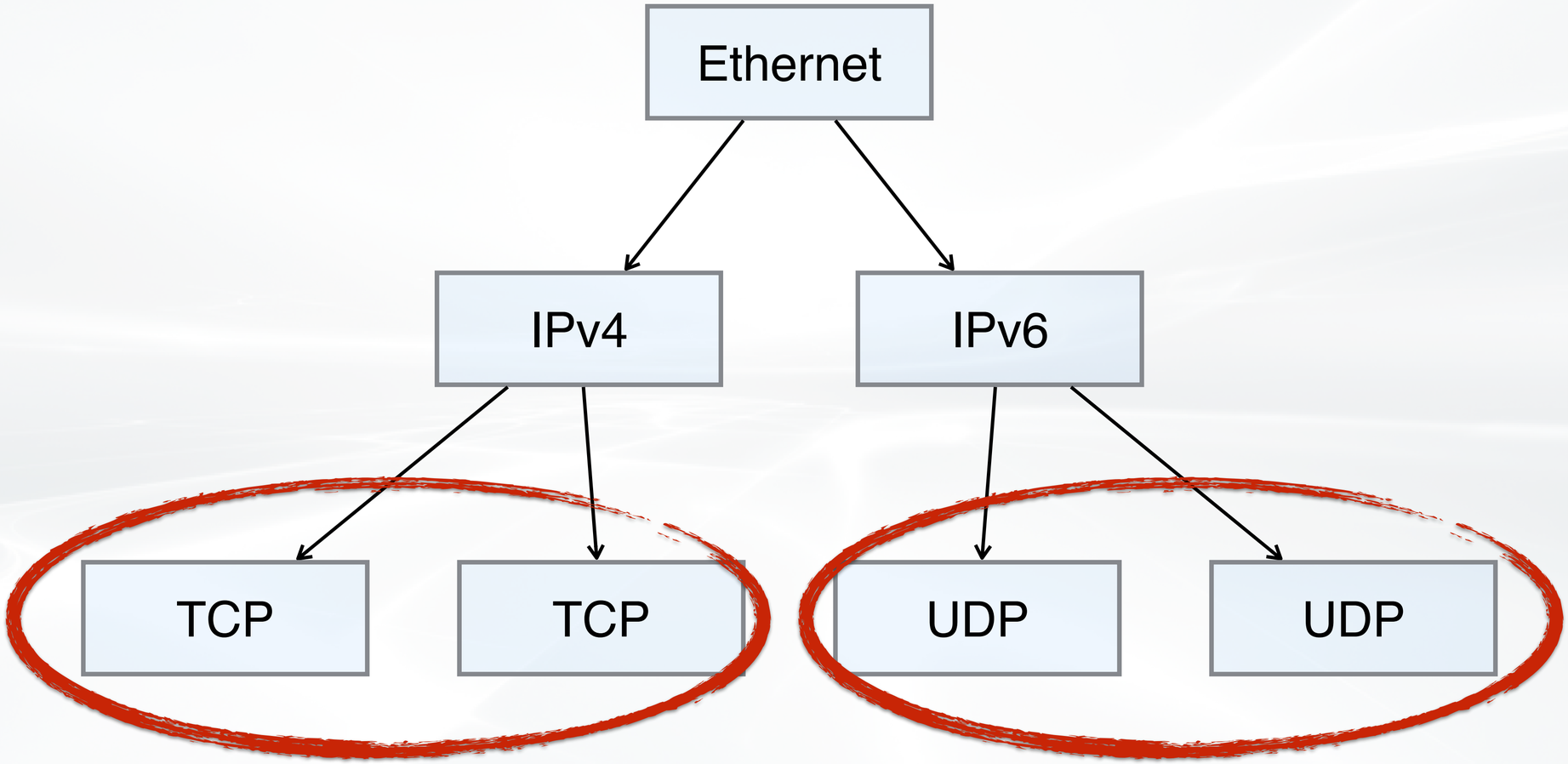
Parser State Explosion

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



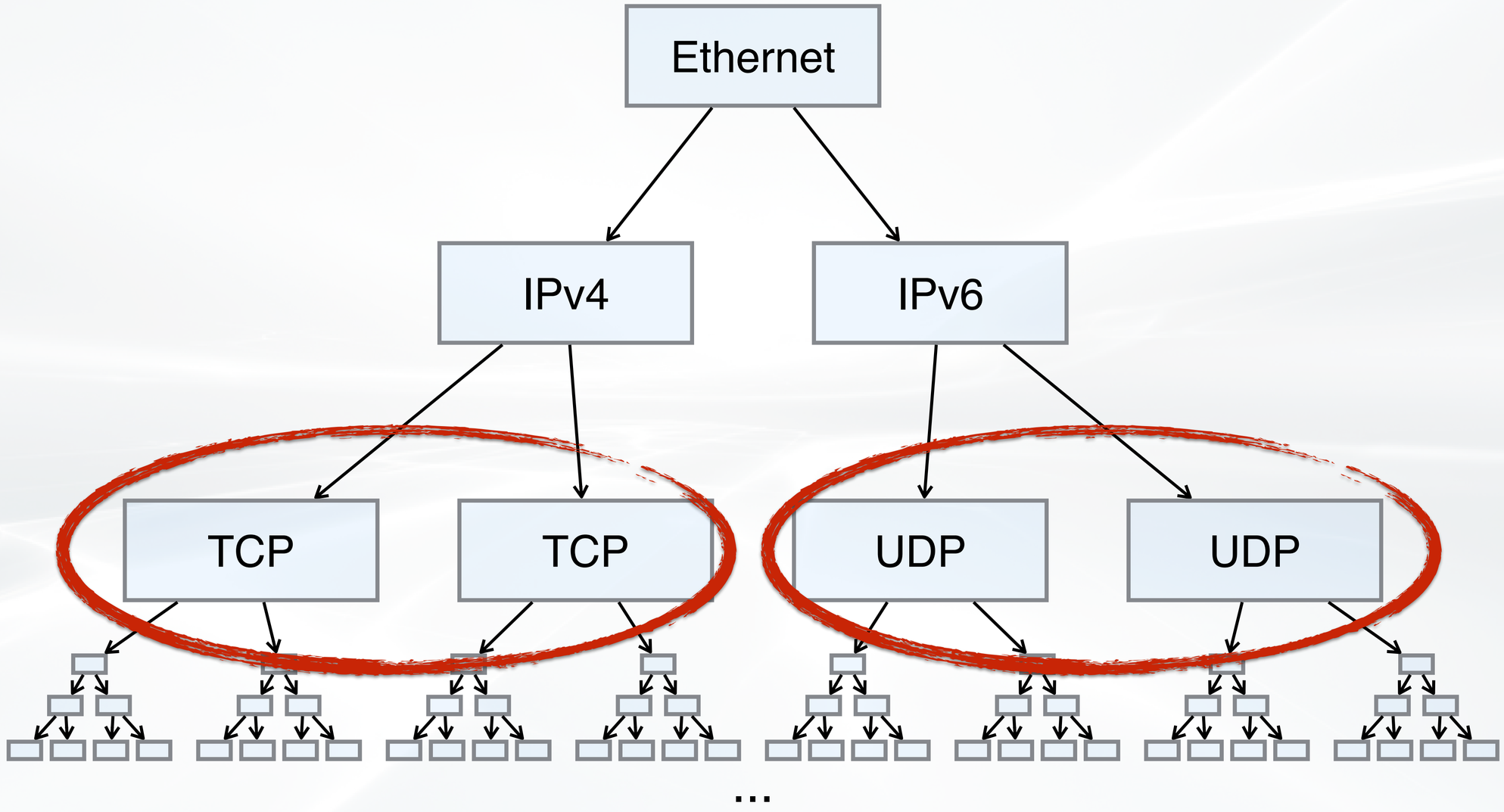
Parser State Explosion

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

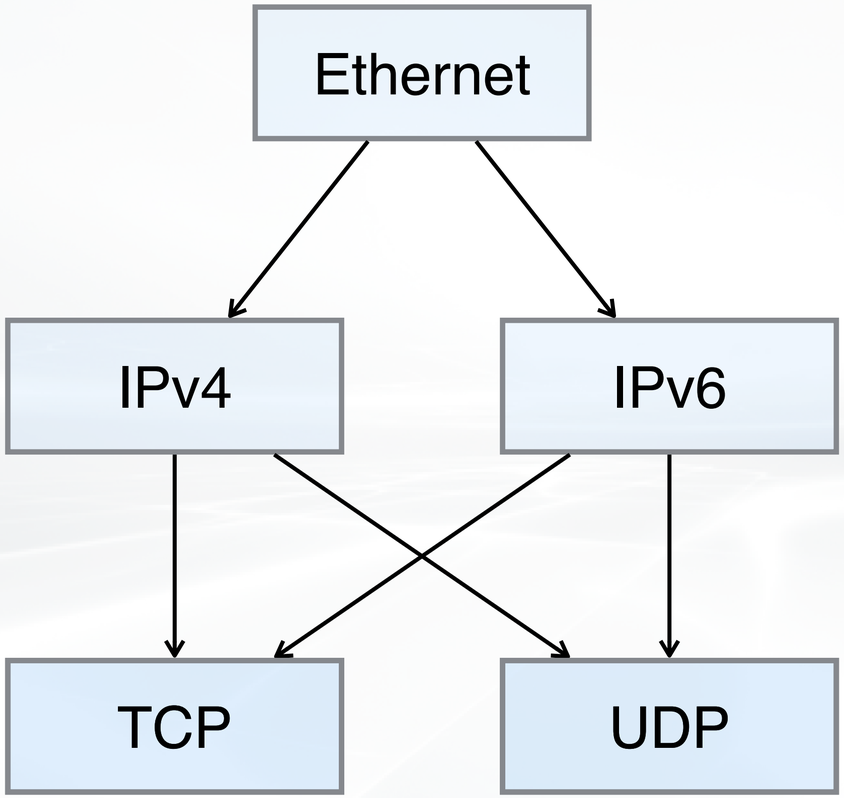


Parser State Explosion

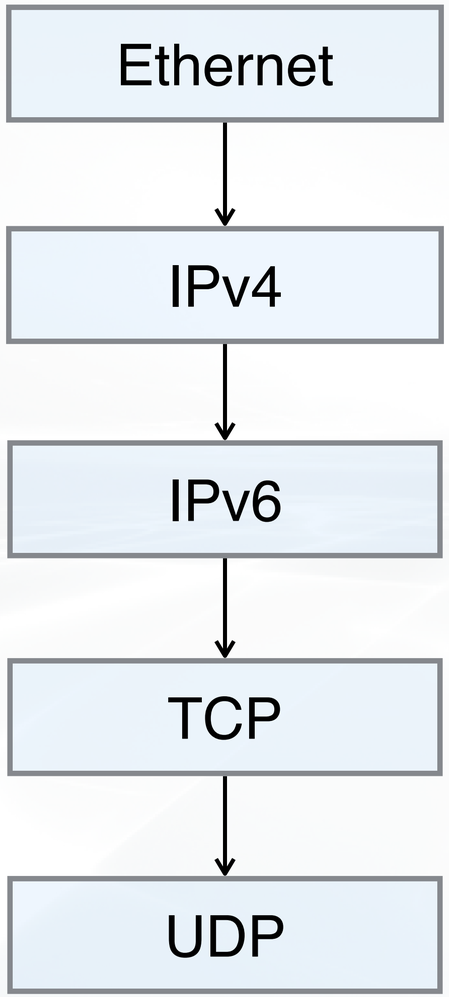
```
Ethernet;
if (...) {
  Ipv4;
  if (...) {
    TCP;
  } else {
    UDP;
  }
} else {
  Ipv6;
  if (...) {
    TCP;
  } else {
    UDP;
  }
}
```



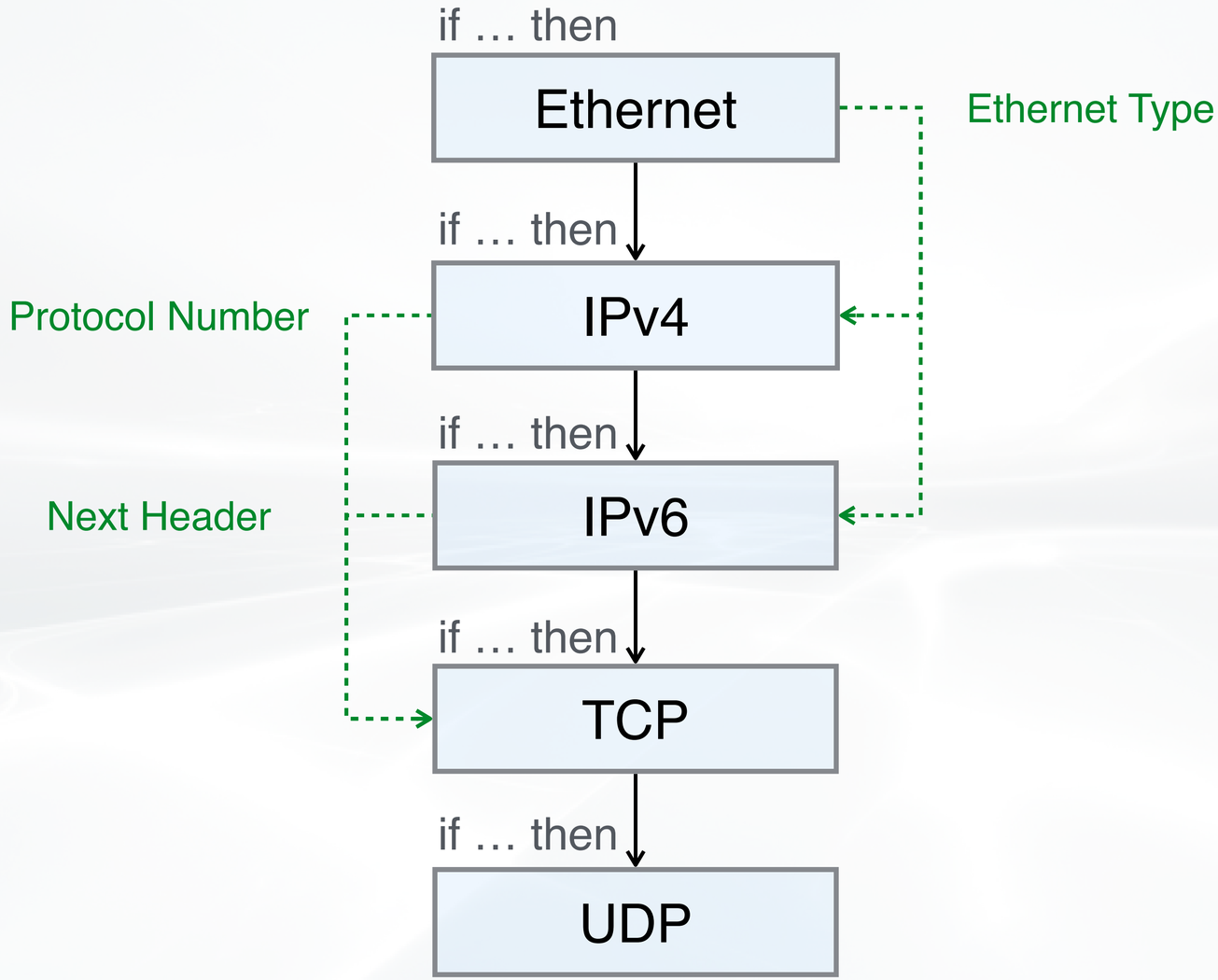
Our Solution: Sequential Encoding



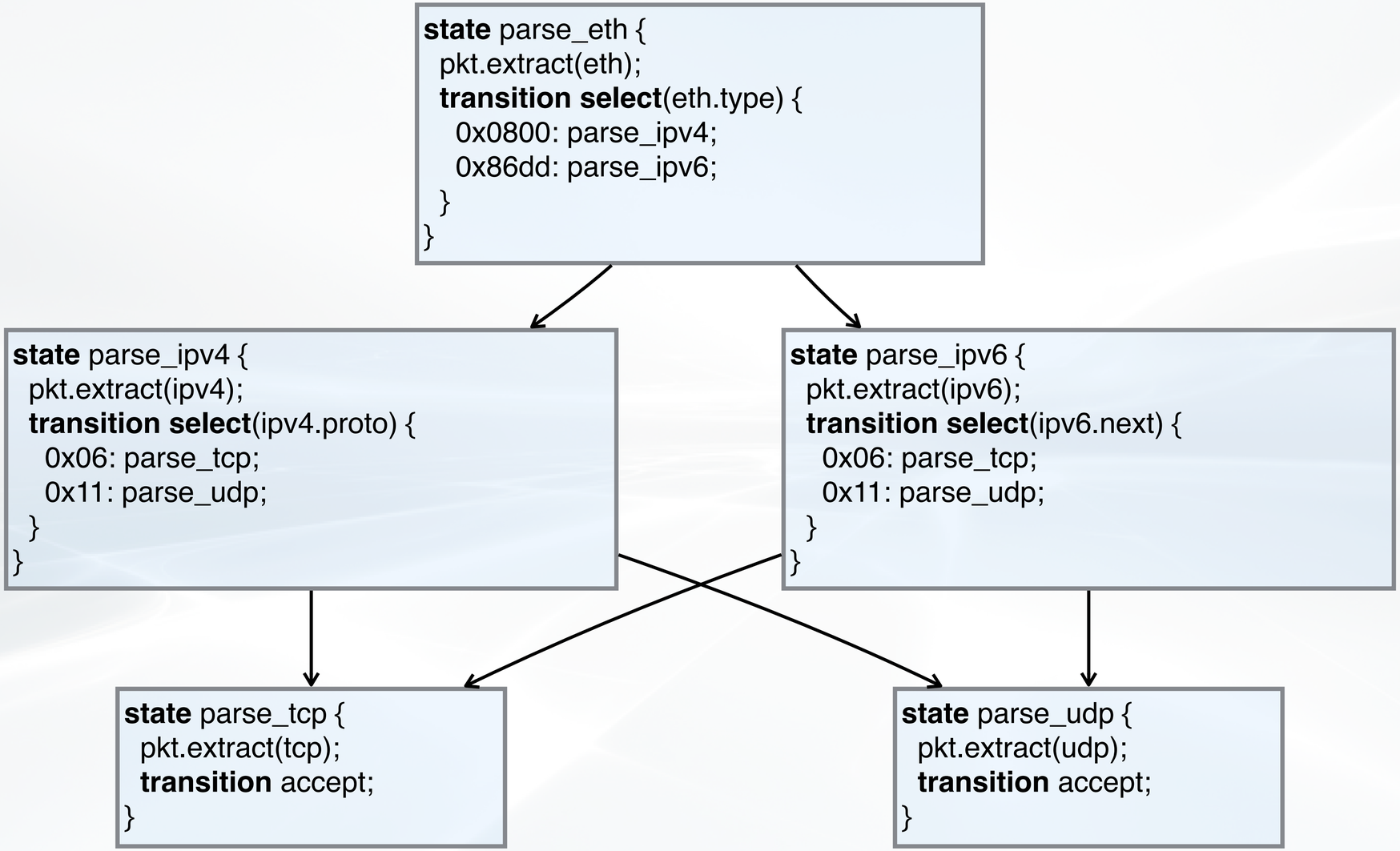
Our Solution: Sequential Encoding



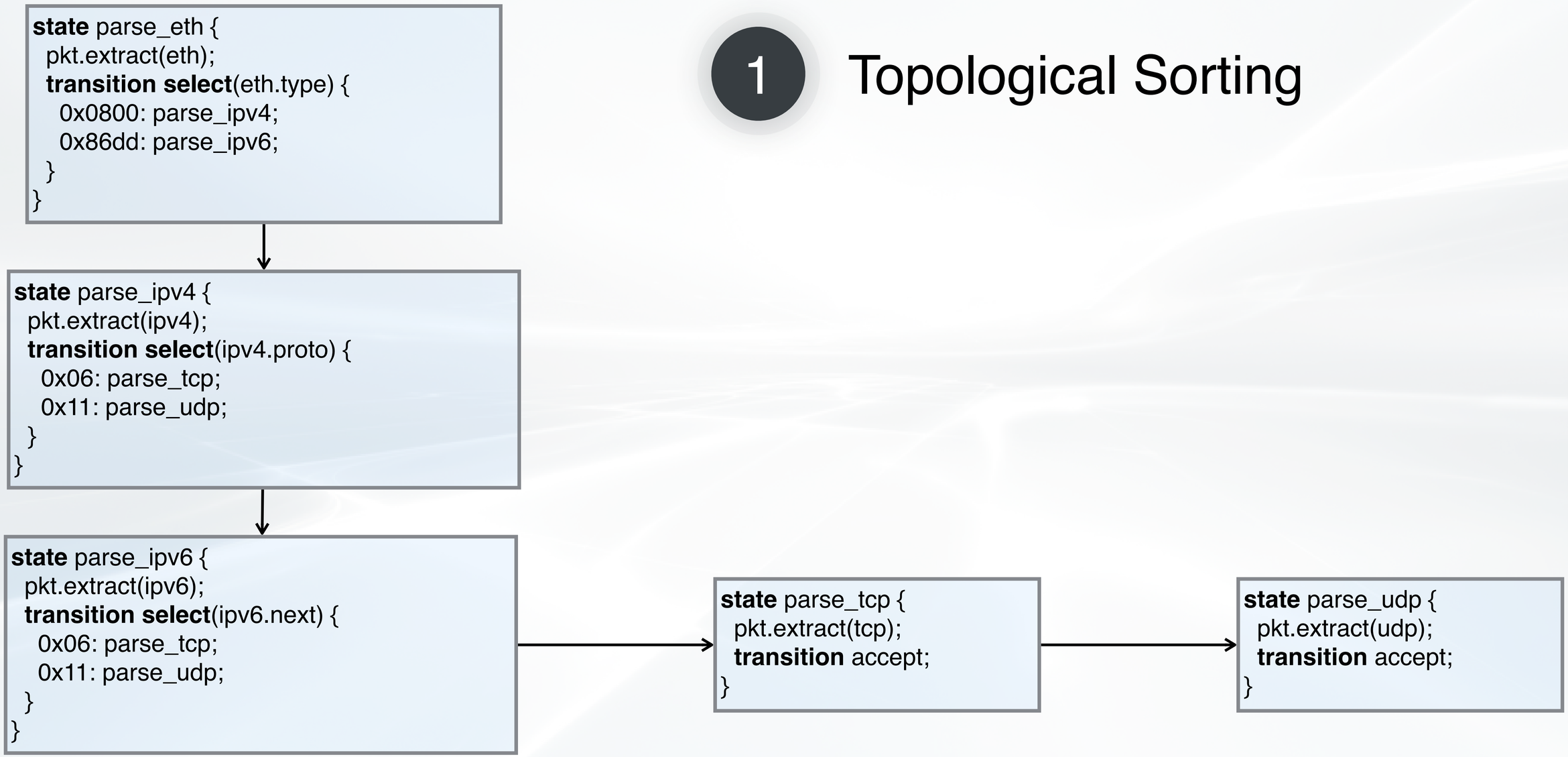
Our Solution: Sequential Encoding



Example of Sequential Encoding

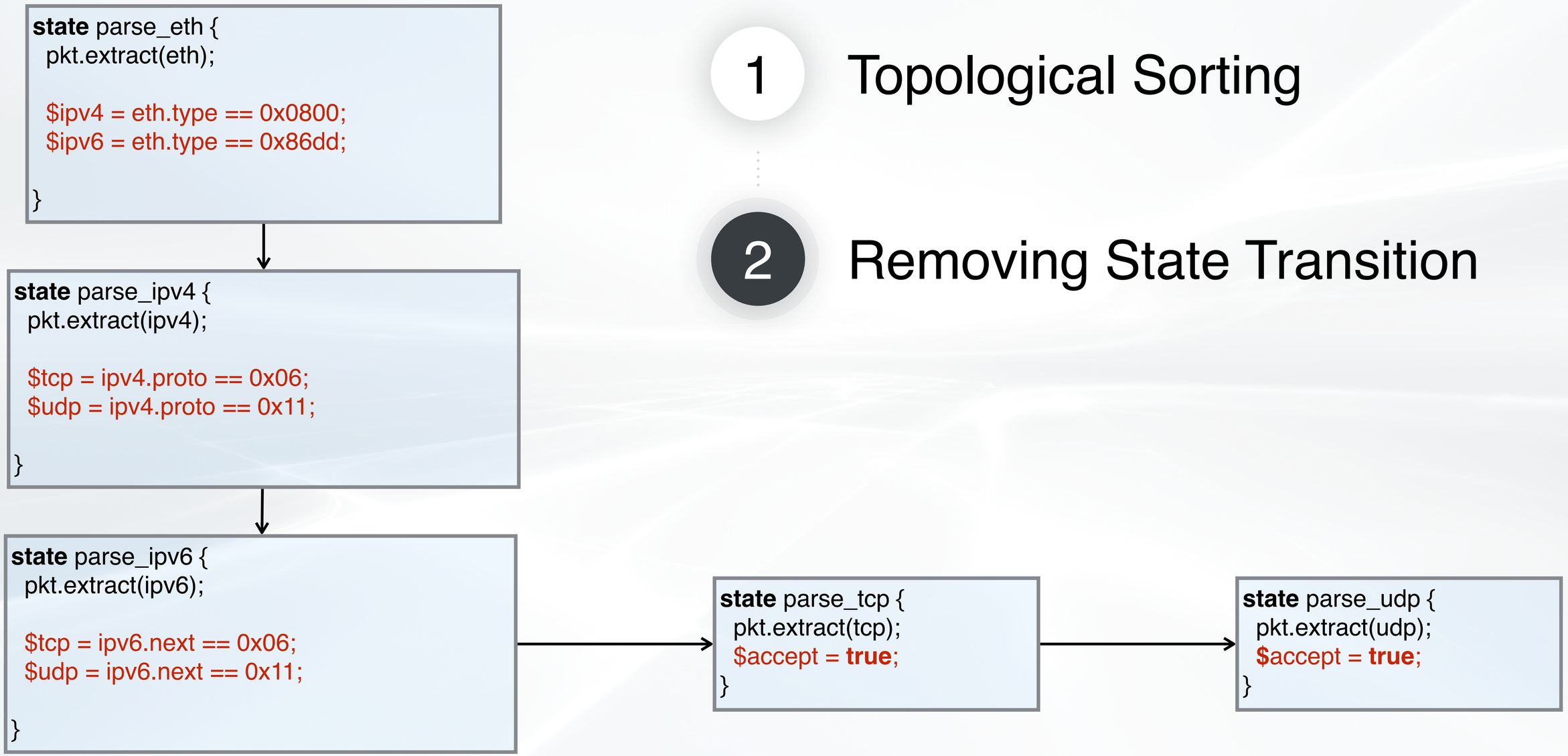


Example of Sequential Encoding



1 Topological Sorting

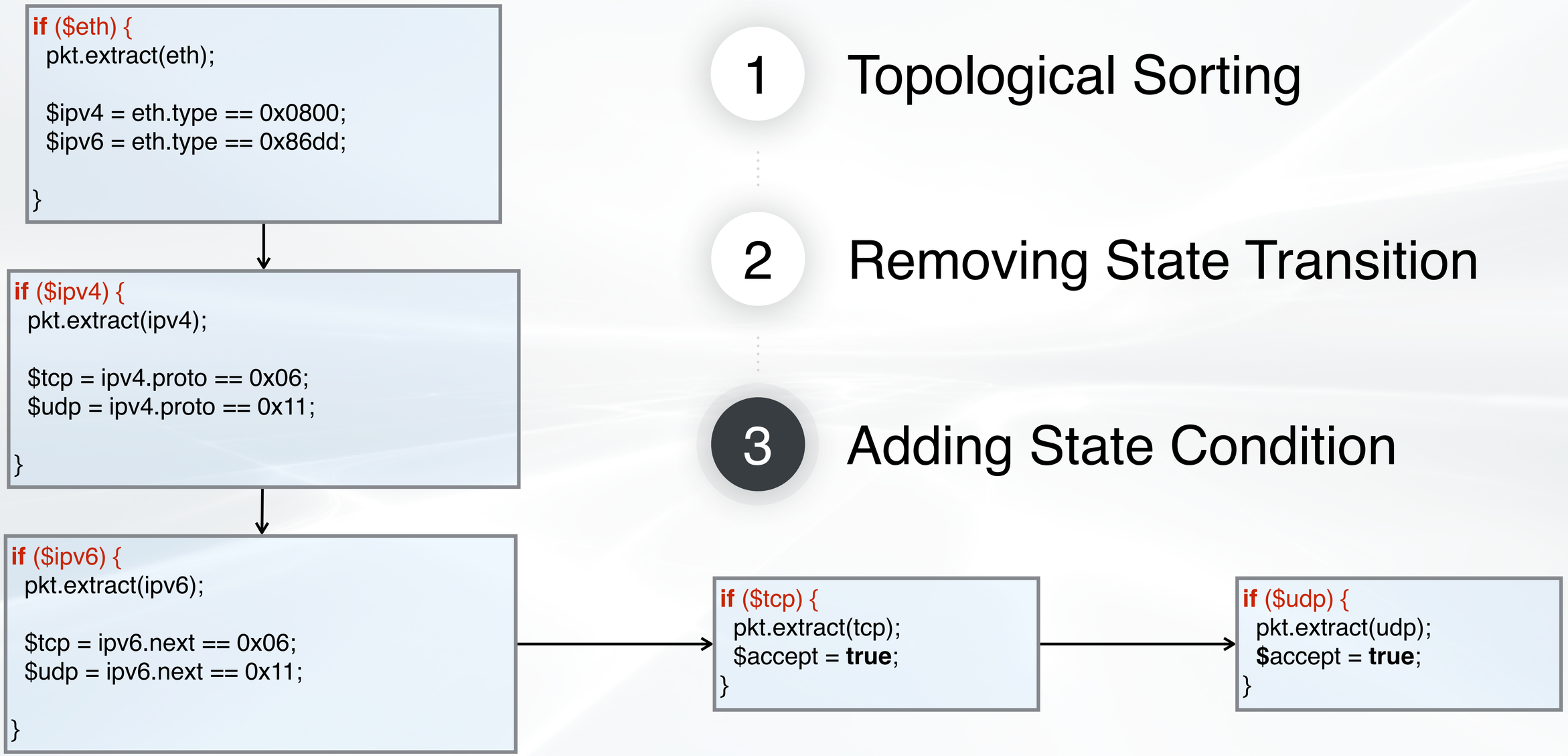
Example of Sequential Encoding



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

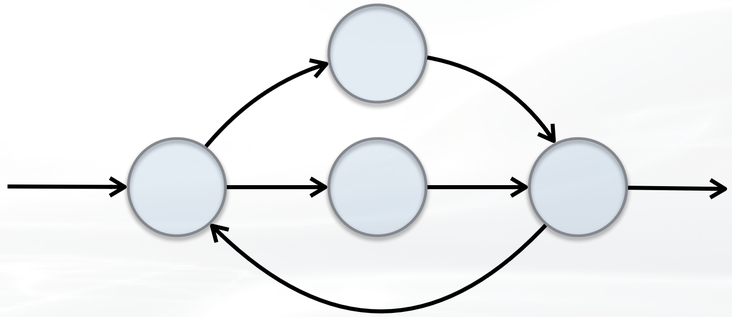
```
$eth = true;
if ($eth) {
  pkt.extract(eth);
  $ipv4 = eth.type == 0x0800;
  $ipv6 = eth.type == 0x86dd;
}
if ($ipv4) {
  pkt.extract(ipv4);
  $tcp = ipv4.proto == 0x06;
  $udp = ipv4.proto == 0x11;
}
if ($ipv6) {
  pkt.extract(ipv6);
  $tcp = ipv6.next == 0x06;
  $udp = ipv6.next == 0x11;
}
if ($tcp)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

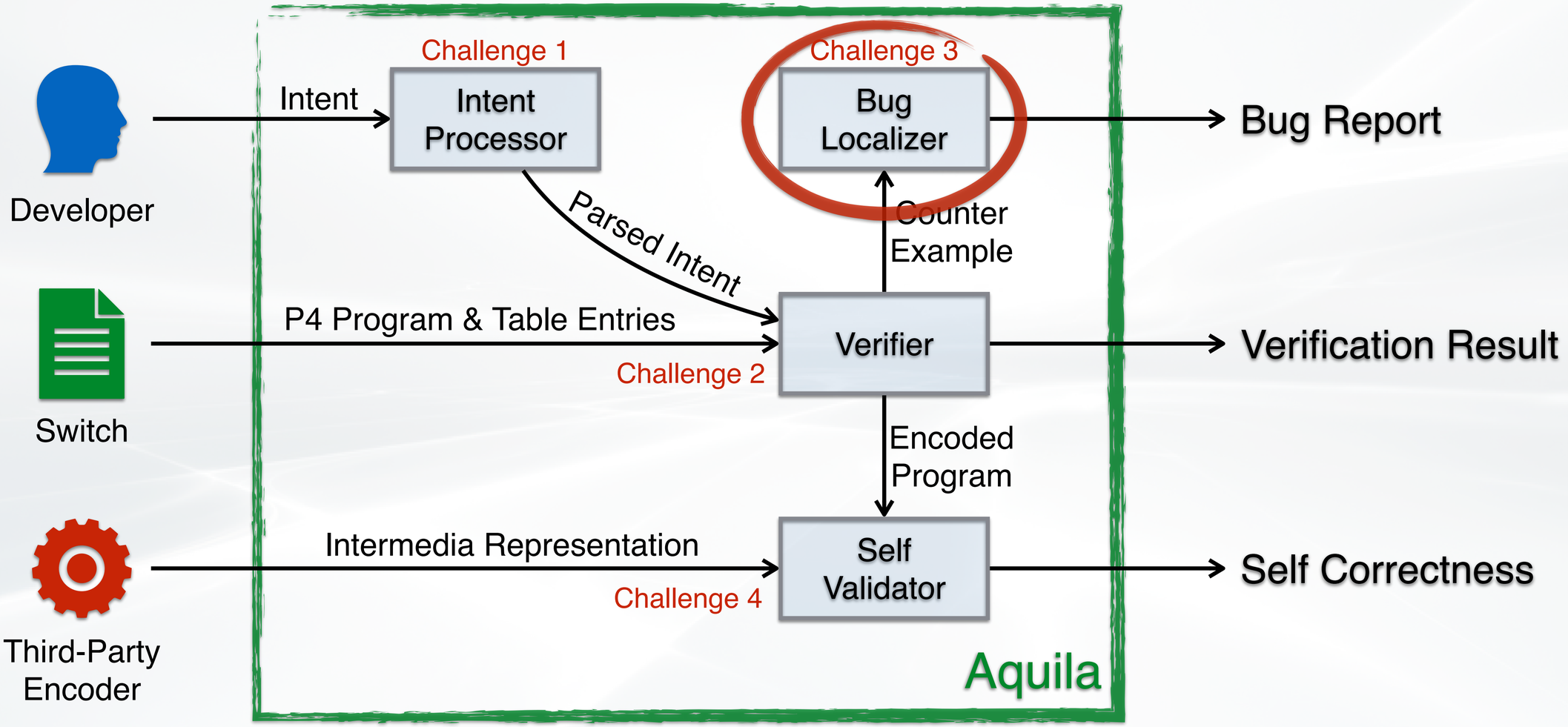
How to encode external functions?



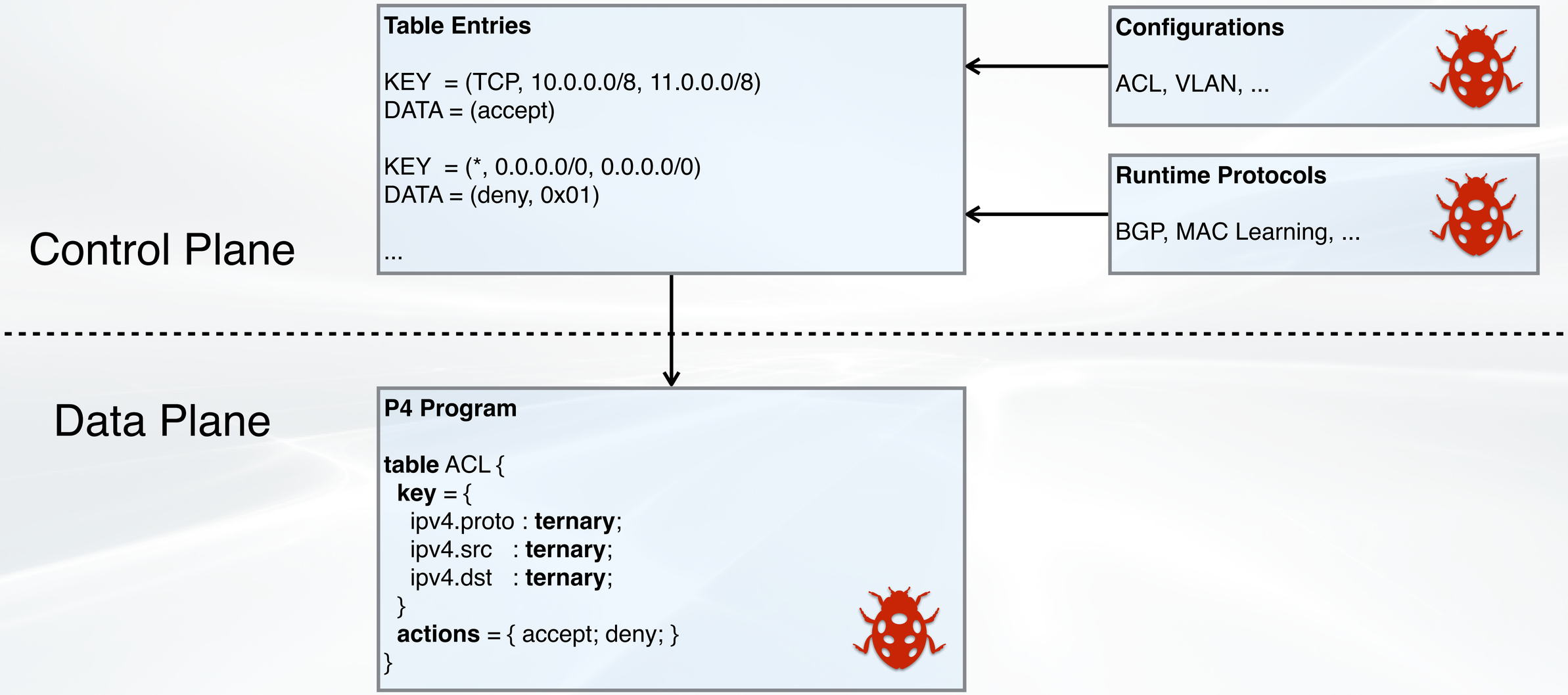
How to handle parser loops?

Please see our paper for more details!

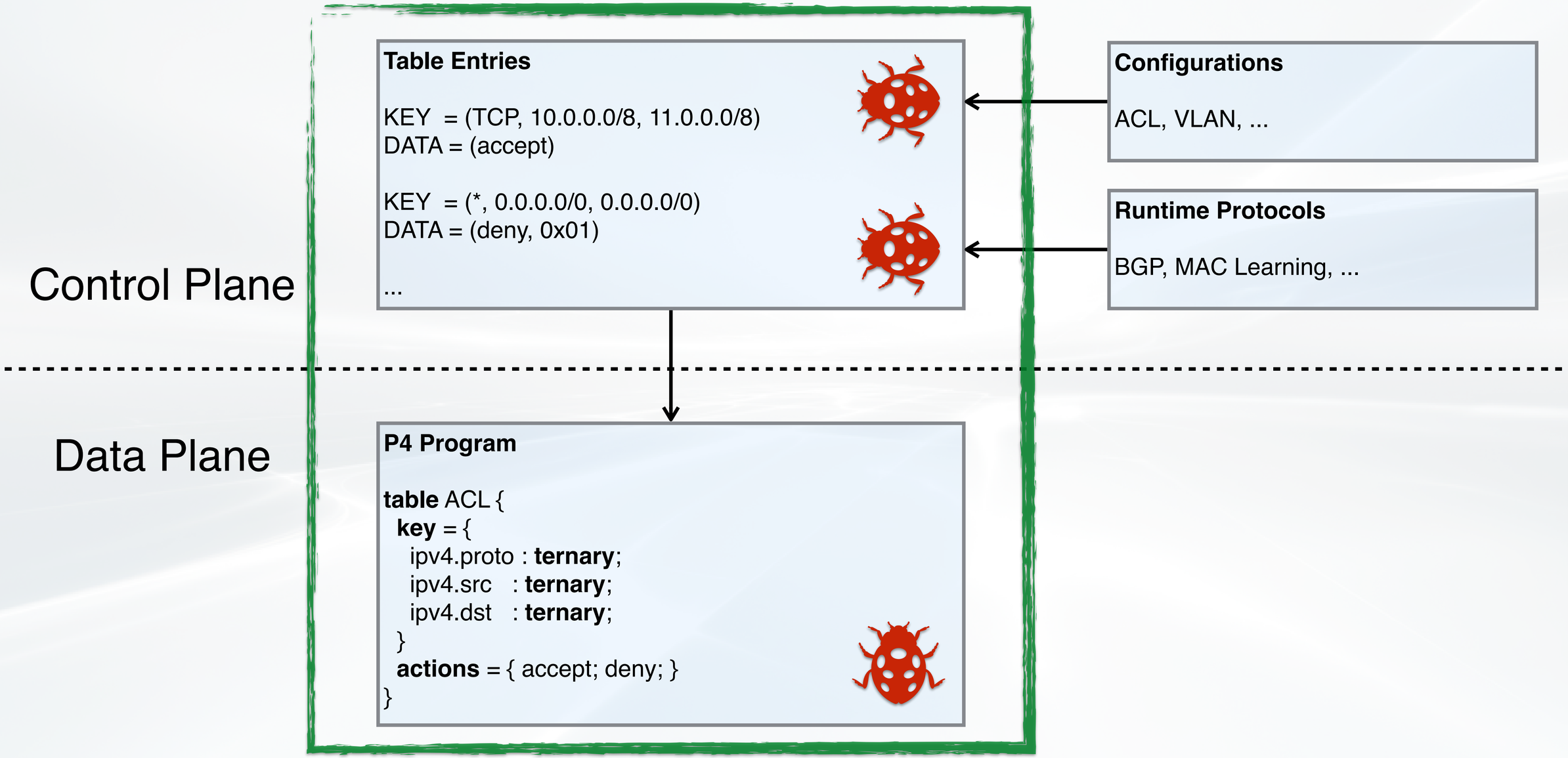
Aquila Architecture



Where is the bug?



Where is the bug?

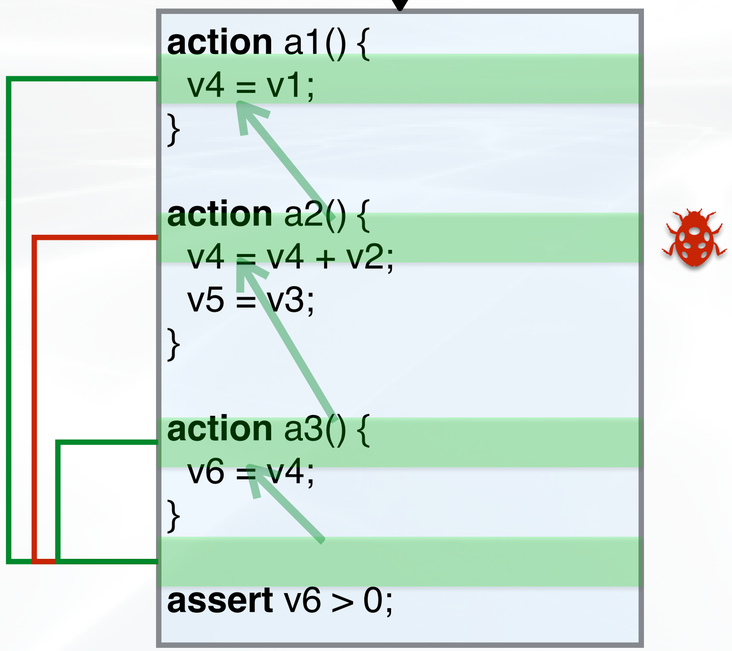


How to localize a data plane bug?

```
action read() { ig_md.ttl = ipv4.ttl; ... }
action drop() {.ig_md.drop = 1; }
action rewrite() { ipv4.ttl = ig_md.ttl; ... }
apply {
  read();
  if (ig_md.ttl == 0) { drop(); }
  // ig_md.ttl = ig_md.ttl - 1; // Bug: code missing
  ...
  rewrite();
}
```

How to localize a data plane bug?

```
action read() { ig_md.ttl = ipv4.ttl; ... }
action drop() {.ig_md.drop = 1; }
action rewrite() { ipv4.ttl = ig_md.ttl; ... }
apply {
  read();
  if (ig_md.ttl == 0) { drop(); }
  // ig_md.ttl = ig_md.ttl - 1; // Bug: code missing
  ...
  rewrite();
}
// Intent: assert ipv4.ttl == @ipv4.ttl - 1
```



Fast Filter: Find Suspicious Variables

```
action a1() {
  v4 = v1;
}

action a2() {
  v4 = v4 + v2;
  v5 = v3;
}

action a3() {
  v6 = v4;
}

assert v6 > 0;
```

Step	Suspicious Variables
assert	v6

Fast Filter: Find Suspicious Variables

```
action a1() {
  v4 = v1;
}

action a2() {
  v4 = v4 + v2;
  v5 = v3;
}

action a3() {
  v6 = v4;
}

assert v6 > 0;
```

Step	Suspicious Variables
assert	v6
a3	v4

Fast Filter: Find Suspicious Variables

```
action a1() {
  v4 = v1;
}

action a2() {
  v4 = v4 + v2;
  v5 = v3;
}

action a3() {
  v6 = v4;
}

assert v6 > 0;
```

Step	Suspicious Variables
assert	v6
a3	v4
a2	v4 v2

Fast Filter: Find Suspicious Variables

```
action a1() {
  v4 = v1;
}

action a2() {
  v4 = v4 + v2;
  v5 = v3;
}

action a3() {
  v6 = v4;
}

assert v6 > 0;
```

Step	Suspicious Variables
assert	v6
a3	v4
a2	v4 v2
a1	v1 v2

Fast Filter: Find Suspicious Variables

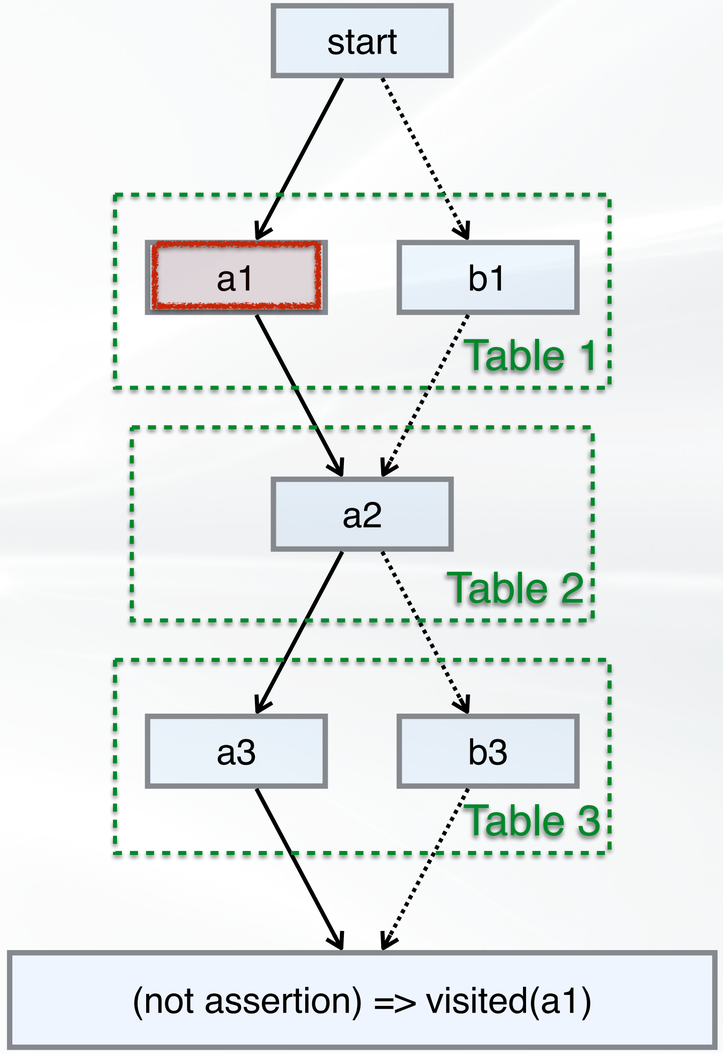
```
action a1() {
  v4 = v1;
}

action a2() {
  v4 = v4 + v2;
  v5 = v3;
}

action a3() {
  v6 = v4;
}

assert v6 > 0;
```

Step	Suspicious Variables
assert	v6
a3	v4
a2	v4 ~~v2~~
a1	~~v1~~ ~~v2~~

Slow Verification: Check Causality

```
action a1() {
  v4 = v1;
}

action a2() {
  v4 = v4 + v2;
  v5 = v3;
}

action a3() {
  v6 = v4;
}

assert v6 > 0;
```

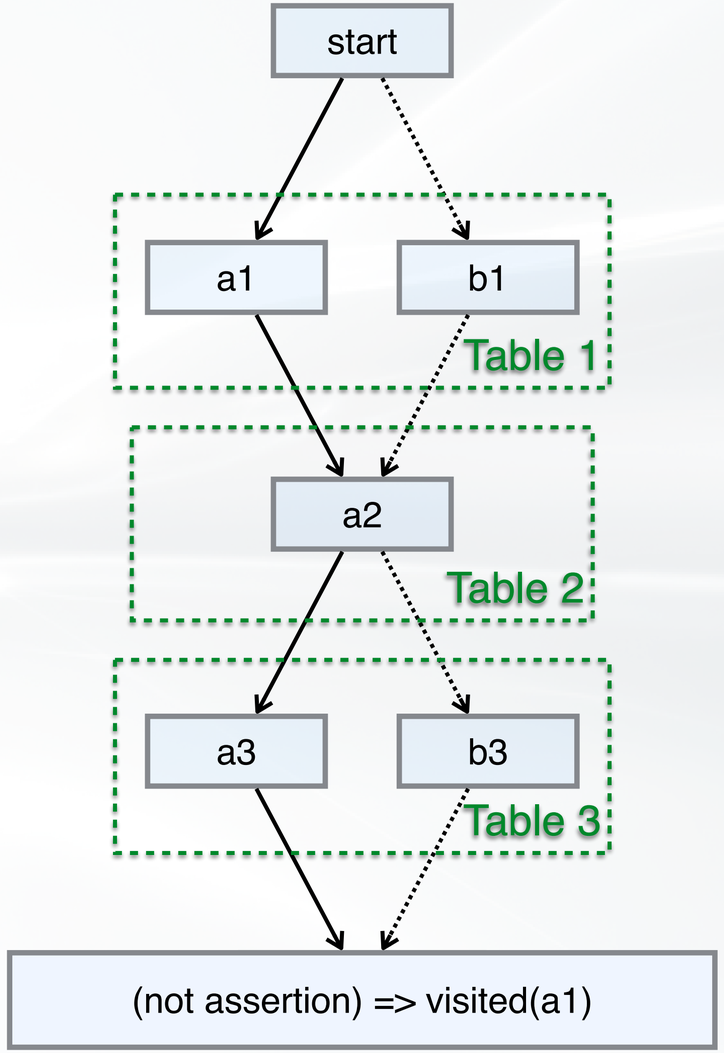
Step	Suspicious Variables
assert	v6
a3	v4
a2	v4 ~~v2~~
a1	~~v1~~ ~~v2~~



Slow Verification: Check Causality

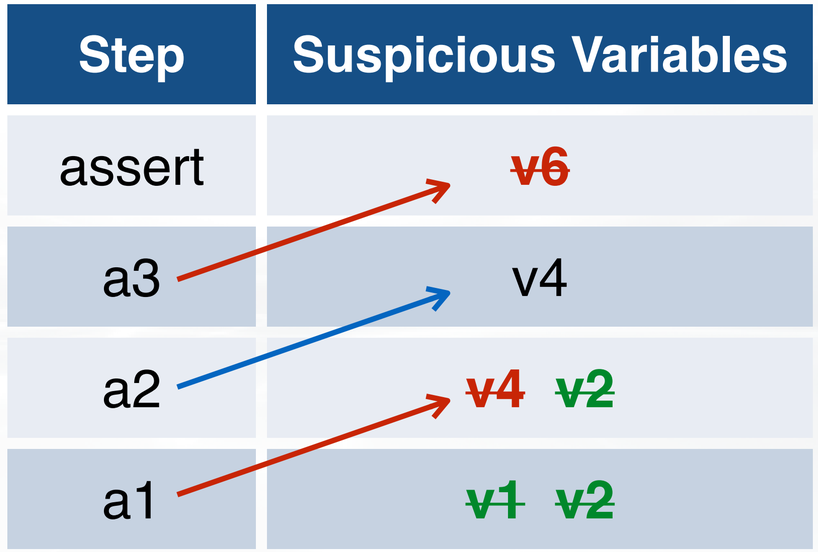
```
action a1() {
  v4 = v1;
}

action a2() {
  v4 = v4 + v2;
  v5 = v3;
}

action a3() {
  v6 = v4;
}

assert v6 > 0;
```

Step	Suspicious Variables
assert	v6
a3	v4
a2	~~v4~~ ~~v2~~
a1	~~v1~~ ~~v2~~



Slow Verification: Check Causality

```
action a1() {
  v4 = v1;
}

action a2() {
  v4 = v4 + v2;
  v5 = v3;
}

action a3() {
  v6 = v4;
}

assert v6 > 0;
```



Slow Verification: Check Causality

```
action a1() {
  v4 = v1;
}

action a2() {
  v4 = v4 + v2;
  v5 = v3;
}

action a3() {
  v6 = v4;
}

assert v6 > 0;
```

Step	Suspicious Variables
assert	~~v6~~
a3	v4
a2	~~v4~~ ~~v2~~
a1	~~v1~~ ~~v2~~

Variable v4 gets wrong just before a3!

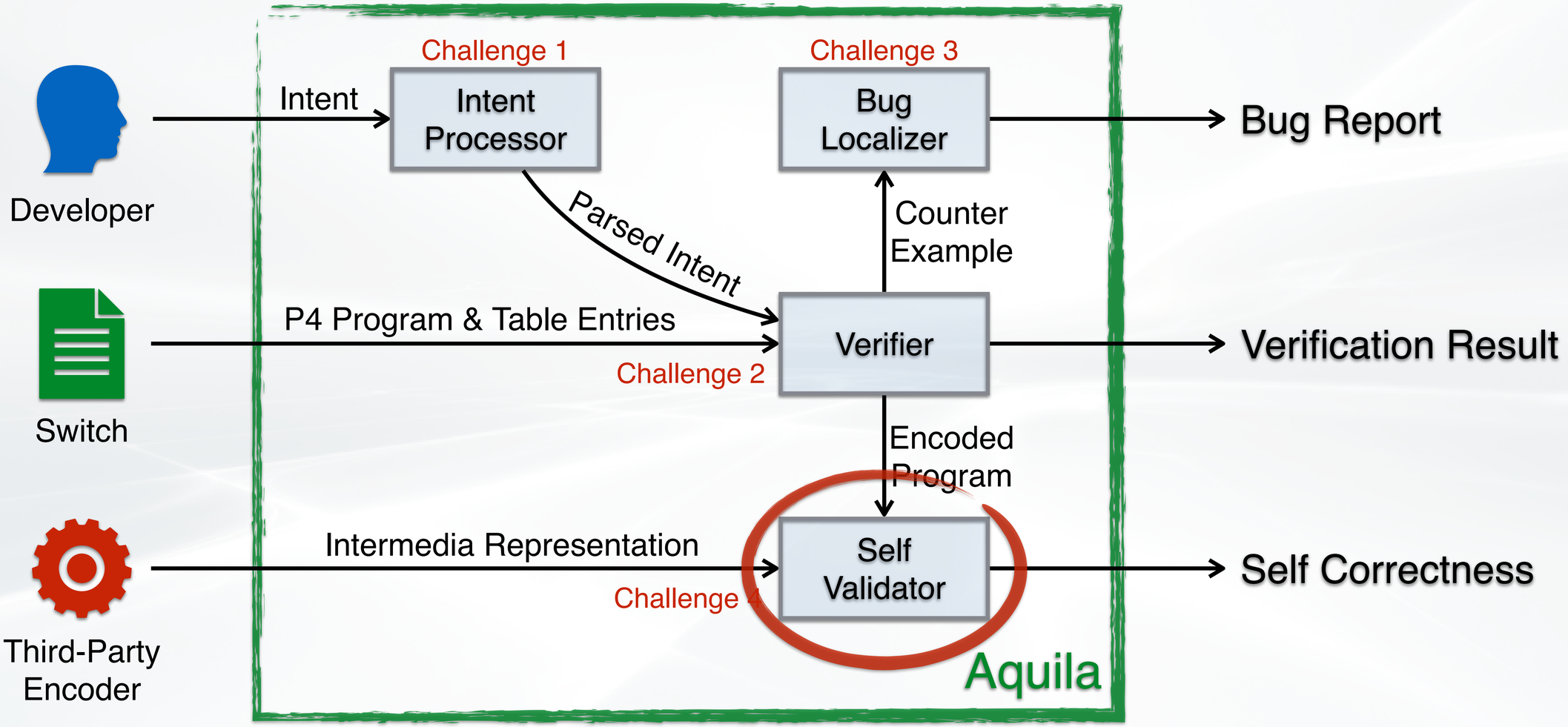
Slow Verification: Check Causality

```
action a1() {
  v4 = v1;
}

action a2() {
  v4 = v4 + v2;
  v5 = v3;
}

action a3() {
  v6 = v4;
}

assert v6 > 0;
```

Step	Suspicious Variables
assert	~~v6~~
a3	v4
a2	~~v4~~ ~~v2~~
a1	~~v1~~ ~~v2~~

Variable v4 gets wrong just before a3!

Aquila Architecture



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

Multi-Pipeline Overhead

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

Parser State Overhead

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!

Bug Localization

	Wrong Table Entry		Code Missing		Code Error	
	Time (s)	Precision	Time (s)	Precision	Time (s)	Precision
Small	17.1	100%	68.6	94.8%	1.47	100%
Medium	21.2	100%	83.2	95.7%	2.92	100%
Large	31.2	100%	164	96.0%	3.01	100%

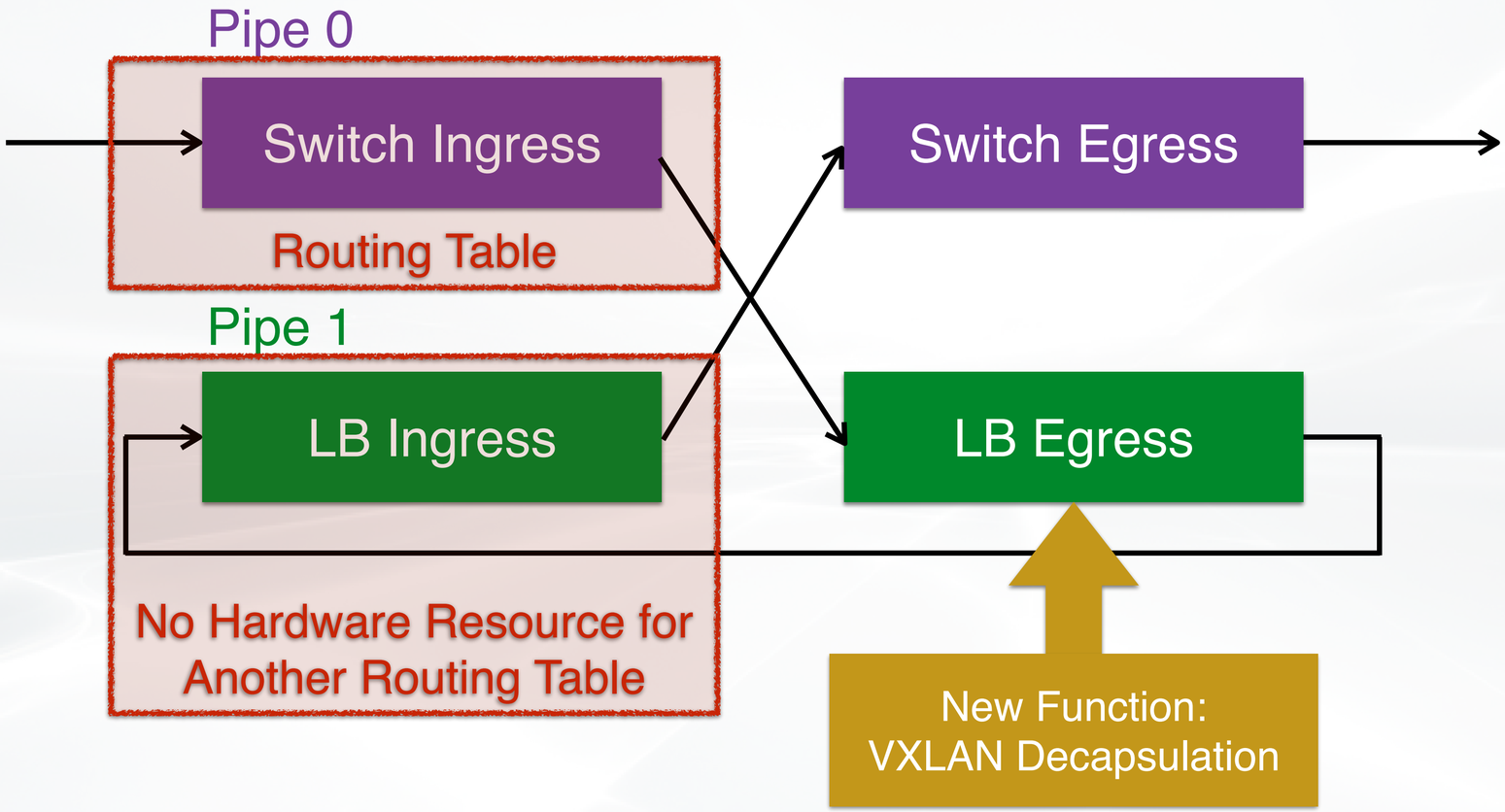
$$\text{Precision} = 1 - \frac{\text{False Positive Lines}}{\text{Total Lines}}$$

Bug Localization

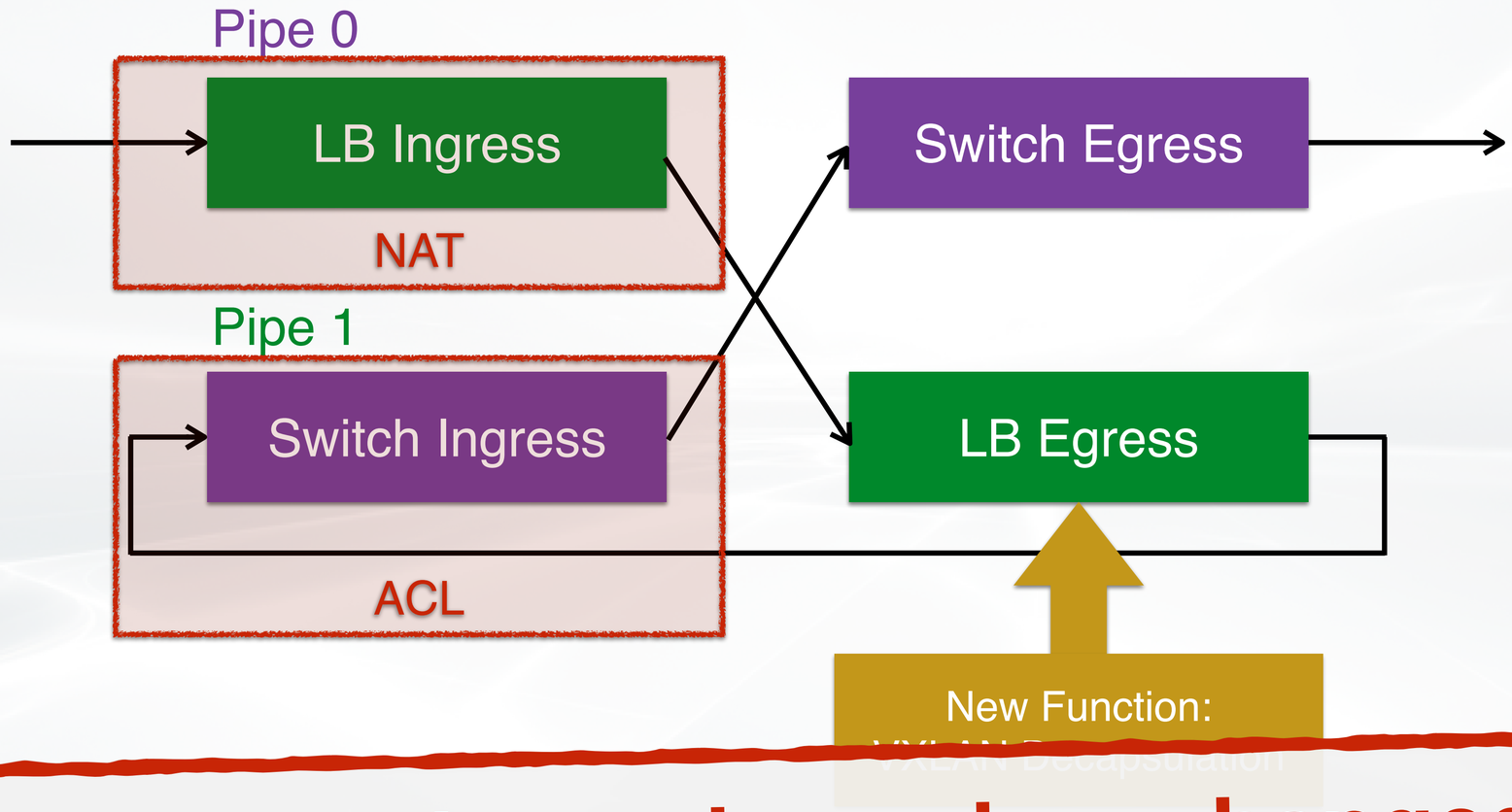
	Wrong Table Entry		Code Missing		Code Error	
	Time (s)	Precision	Time (s)	Precision	Time (s)	Precision
Small	17.1	100%	68.6	94.8%	1.47	100%
Medium	21.2	100%	83.2	95.7%	2.92	100%
Large	31.2	100%	164	96.0%	3.01	100%

Aquila narrows down the suspects!

Experience: Swapping Two Pipelines

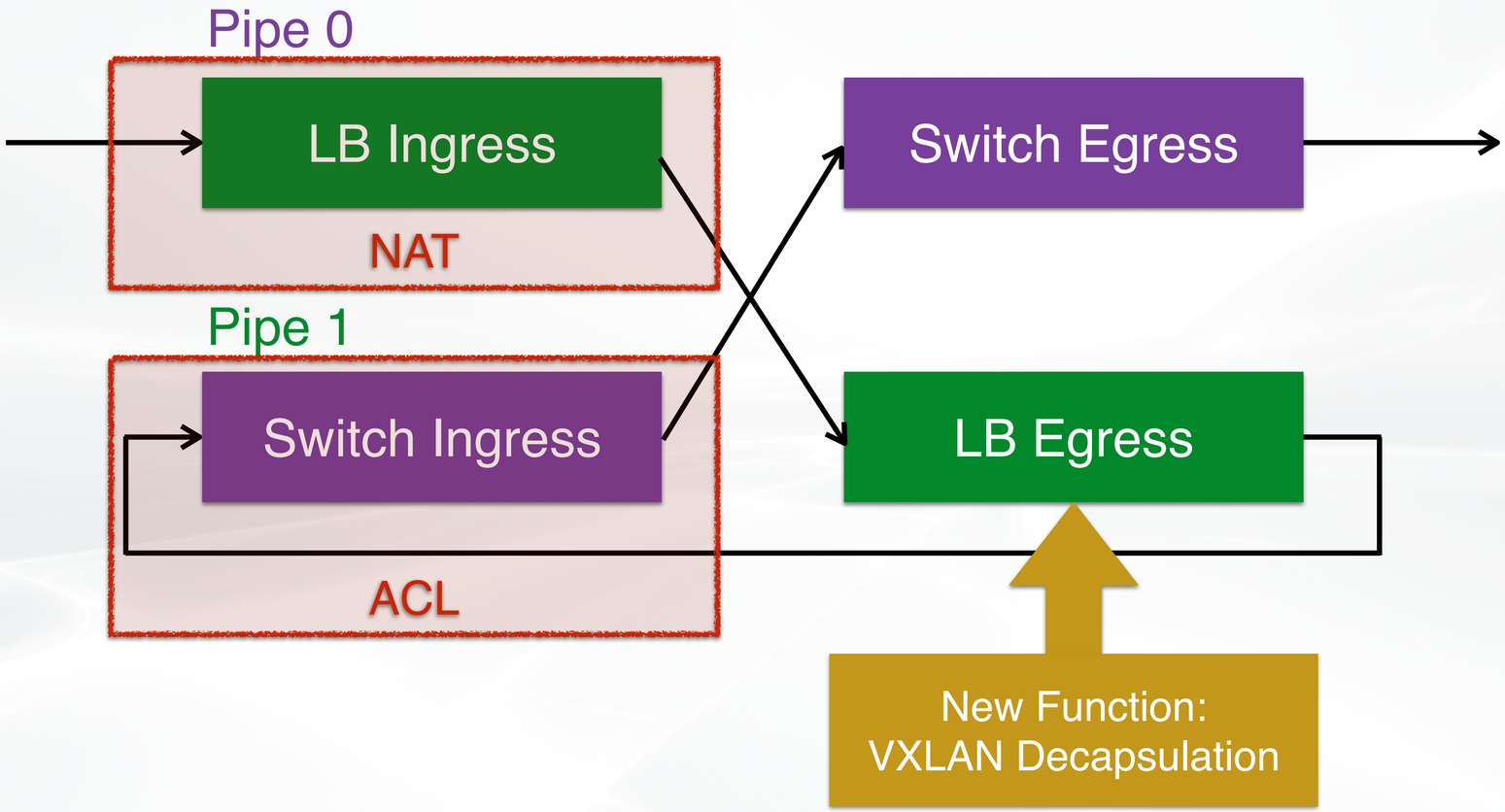


Experience: Swapping Two Pipelines



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.
- Experience learnt from Aquila:
  - Verification should not be a burden
  - Time consumption determines the scenario and frequency of use
  - Developers need detailed information to fix a bug
  - Verifier itself can also have bugs

Conclusion

- Aquila is the first practically usable verification system for production-scale programmable data planes.
- Experience learnt from Aquila:
  - Verification should not be a burden
  - Time consumption determines the scenario and frequency of use
  - Developers need detailed information to fix a bug
  - Verifier itself can also have bugs
- Open problems:
  - Distributed data plane verification with heterogeneous ASICs
  - Interactive bug localization or bug repair
  - ...

Thanks!