Extending P4 to Realize a Scalable Flow Caching Mechanism

Angelo Tulumello
Network Software Engineer, Axbryd S.r.l.
PhD student, University of Rome Tor Vergata

Marco Bonola, Salvatore Pontarelli, **Axbryd S.r.l.**
Matty Kadosh, Yonatan Piasetzki, **NVIDIA Networking**

Motivations

- Programmable dataplanes bring the flexibility to reprogram the protocol parsing and match/action tables in switching ASICs
  - in the last few years reached the technological maturity (Tb/s)
  - P4 is the *de facto* standard
- A further step to enhance programmability is the addition of primitives that implement stateful functions in the dataplane
  - Problem: stateful NFs usually employ heavy state information, while switching ASICs have limited HW resources
  - but an increasing number of research works is proving its feasibility (e.g. [1][2])
- With this work we propose a programming abstraction for Stateful NFs by extending the P4 language

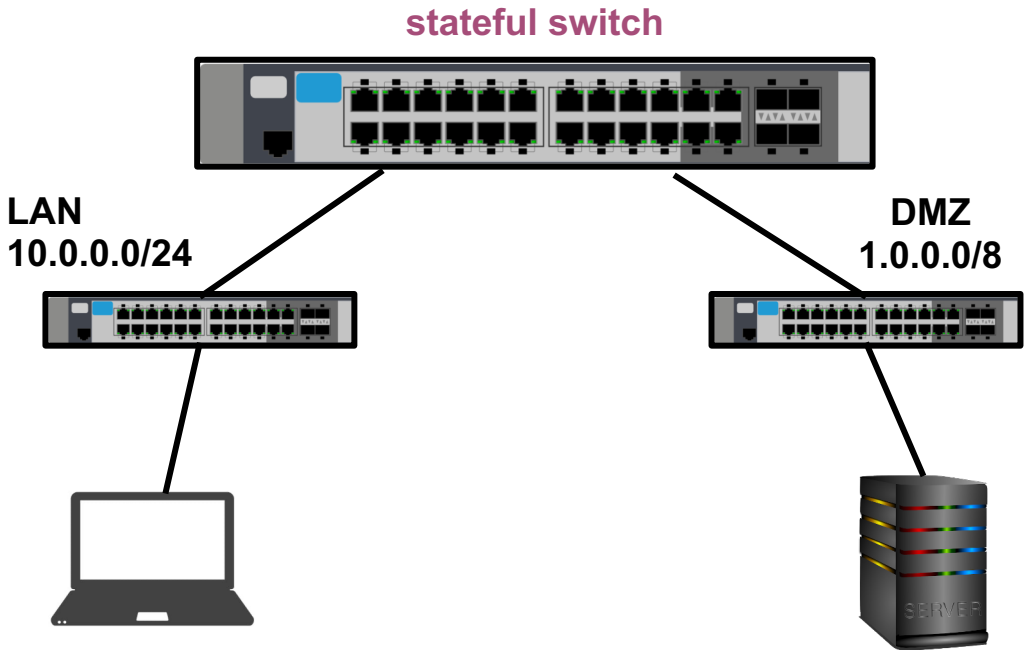
[1] *S. Pontarelli, et al.* ***"Flowblaze: Stateful packet processing in hardware."*** *NSDI '19.*
[2] *D. Kim, et al.* ***"Tea: Enabling state-intensive network functions on programmable switches."*** *SIGCOMM 2020.*

P4 extensions

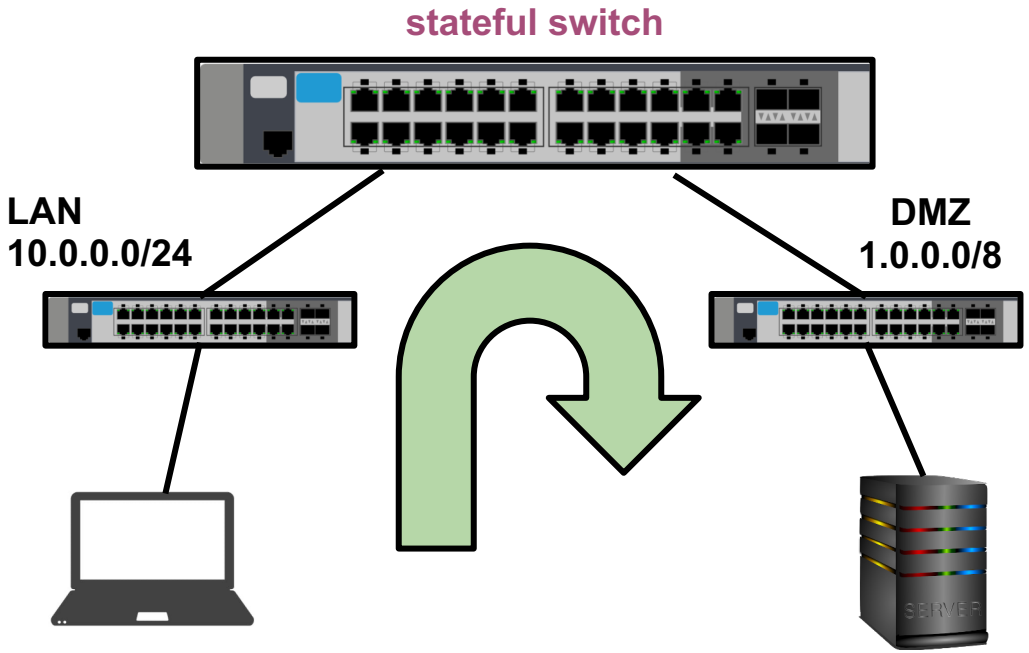
- ➔ Joint work with NVIDIA Networking → understand stateful primitives to be included in future NVIDIA/MLNX hardware and how to program them
  - ◆ Defined a pool of reference use cases
    - e.g. Stateful Firewall -- Flow Cache -- dynamic NAT -- etc.
  - ◆ Derive the functional/architectural requirements (keeping in mind HW)
    - e.g. **flow context table** -- LRU flow cache -- timers -- register stacks
  - ◆ Define the P4 language extensions
  - ◆ Extend bmv2 functional prototype
  - ◆ Validate the use cases implementation in bmv2

P4 extensions walkthrough with a simple use case

Simple example: Stateful Firewall

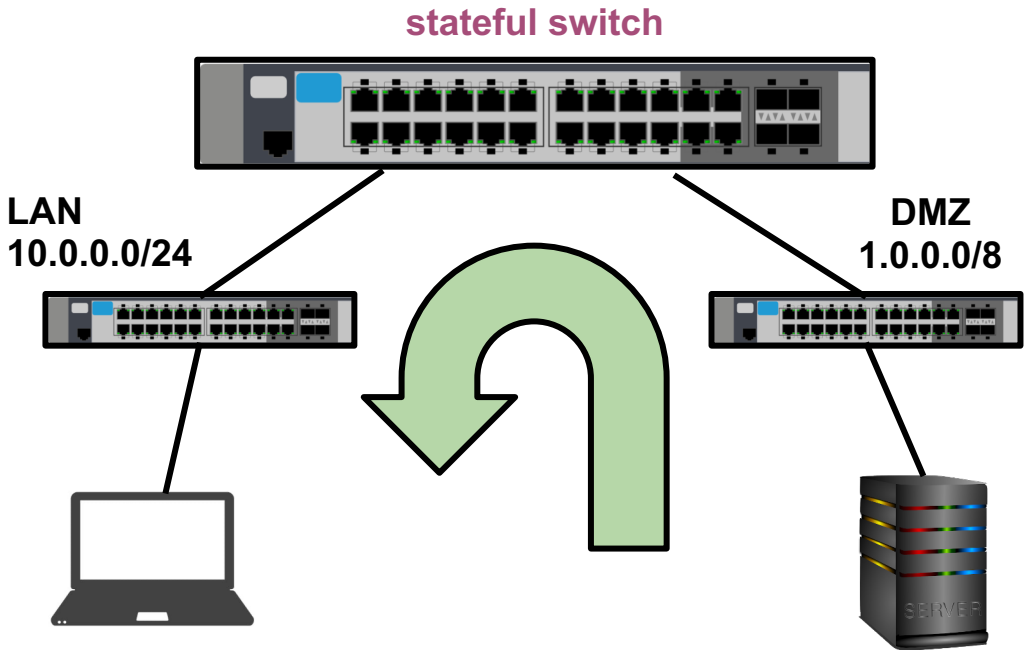


Simple example: Stateful Firewall



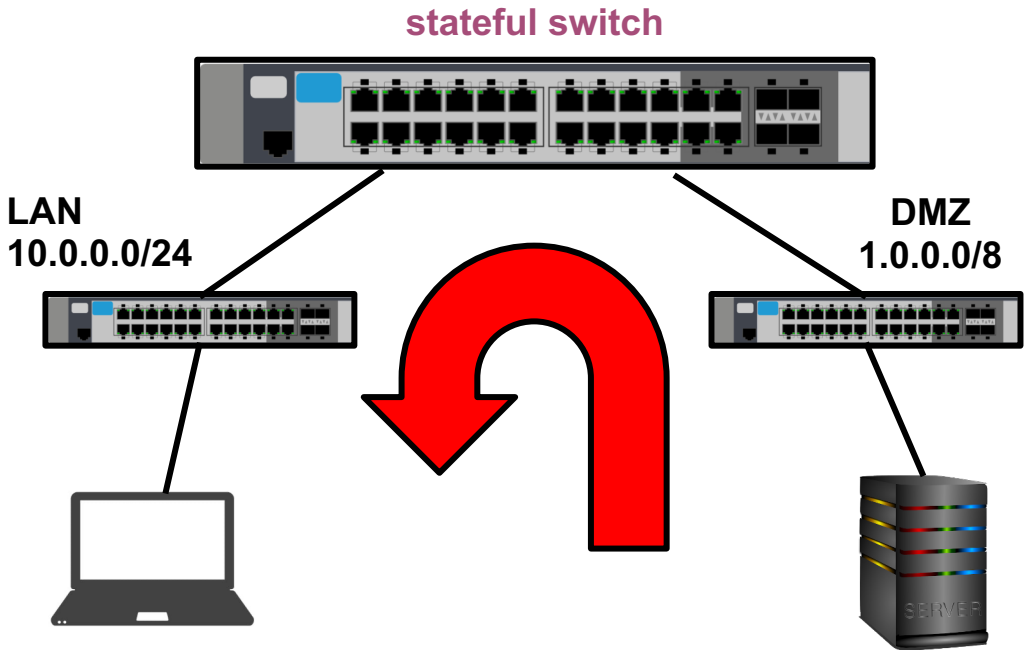
Packets from LAN: ALLOW

Simple example: Stateful Firewall



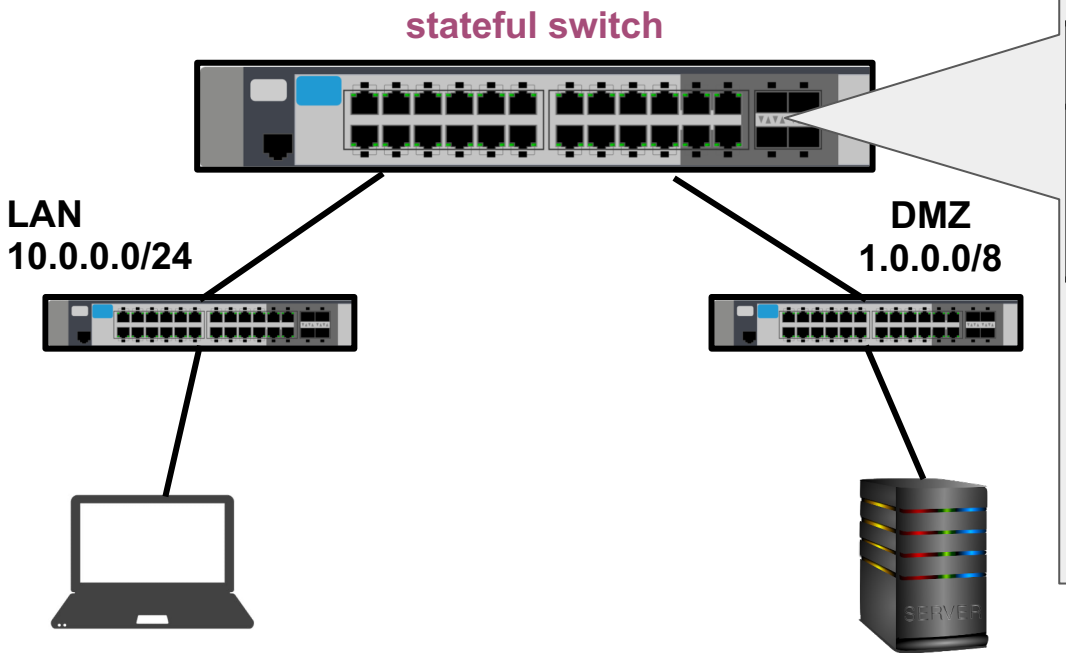
Response packet from DMZ: ALLOW

Simple example: Stateful Firewall



Flow initiated from DMZ: DROP

Stateful Firewall - Flow Context table



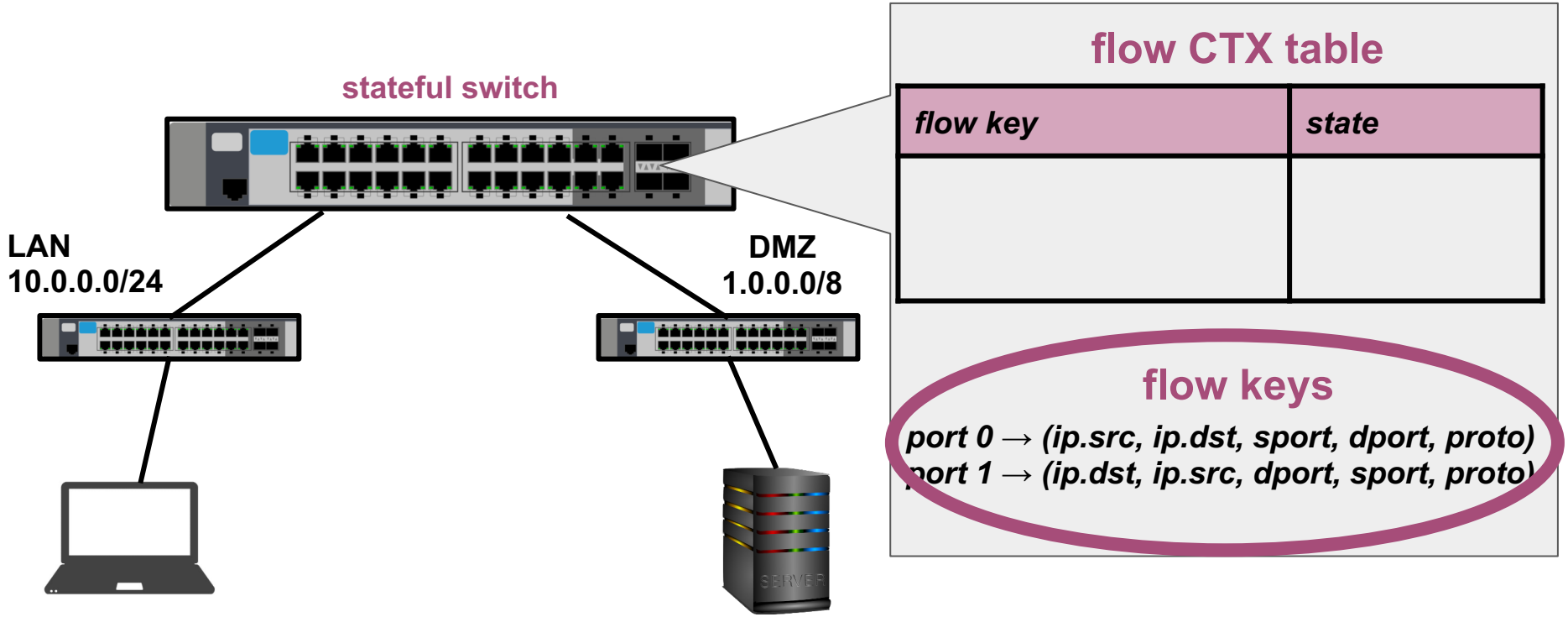
flow CTX table

flow key	state

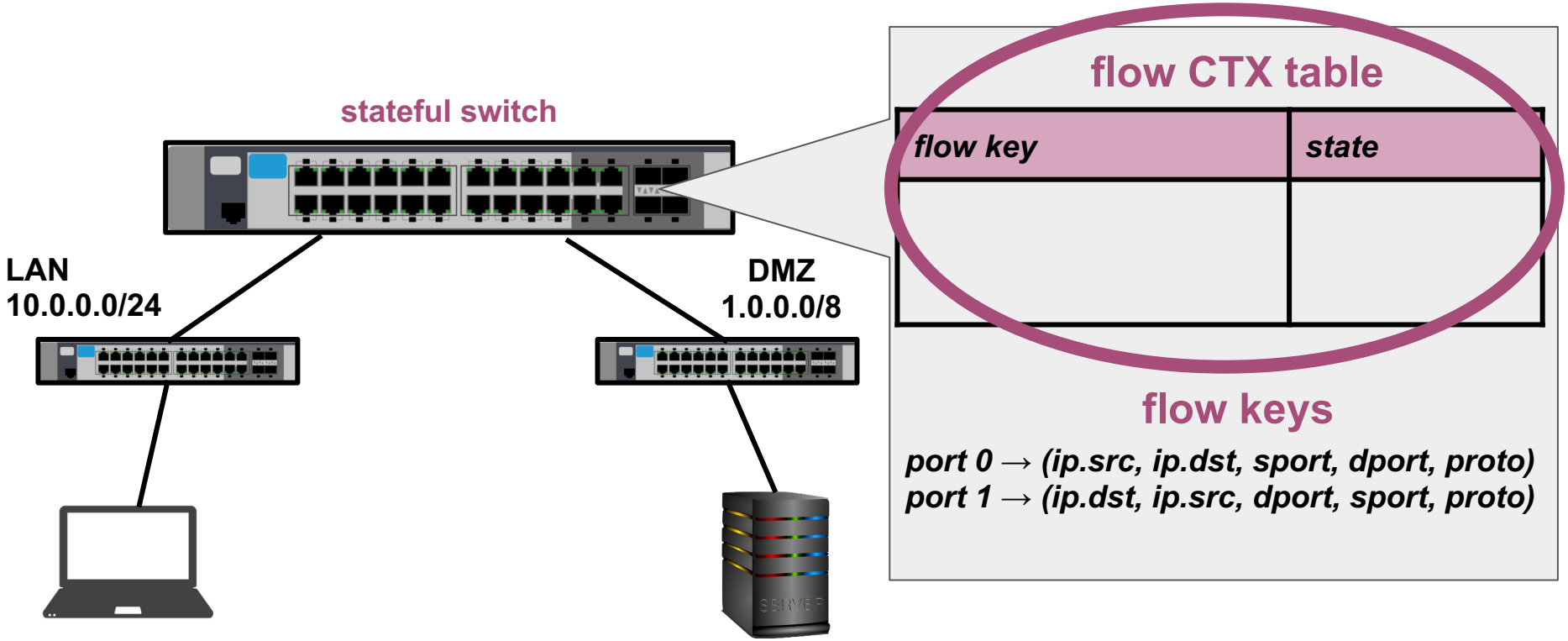
flow keys

port 0 → (ip.src, ip.dst, sport, dport, proto)
port 1 → (ip.dst, ip.src, dport, sport, proto)

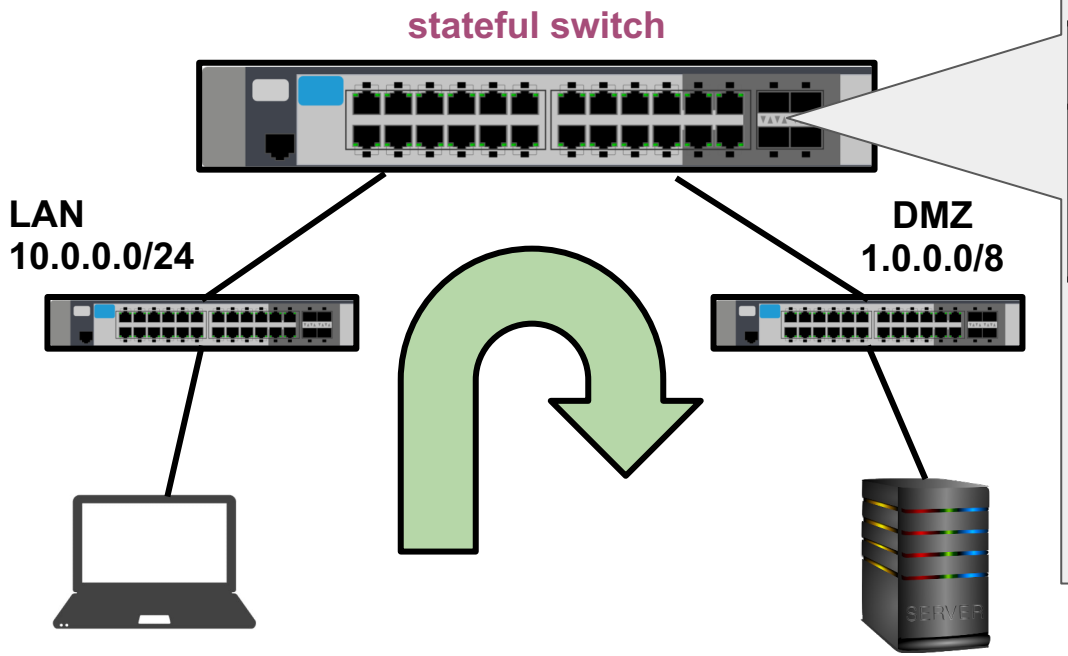
Stateful Firewall - Flow Context table



Stateful Firewall - Flow Context table



Stateful Firewall - Flow Context table



flow CTX table

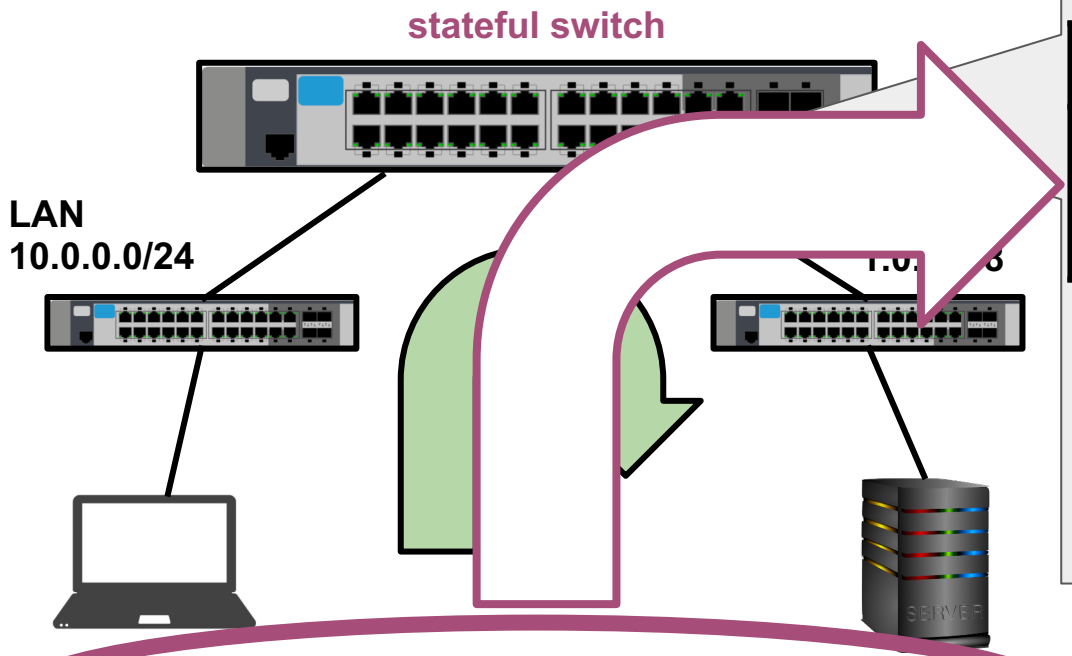
flow key	*state*

flow keys

port 0 → (ip.src, ip.dst, sport, dport, proto)
port 1 → (ip.dst, ip.src, dport, sport, proto)

Flow Key: (10.0.0.100, 1.0.0.1, 1000, 2000, tcp)

Stateful Firewall - Flow Context table



Flow Key: (10.0.0.100, 1.0.0.1, 1000, 2000, tcp)

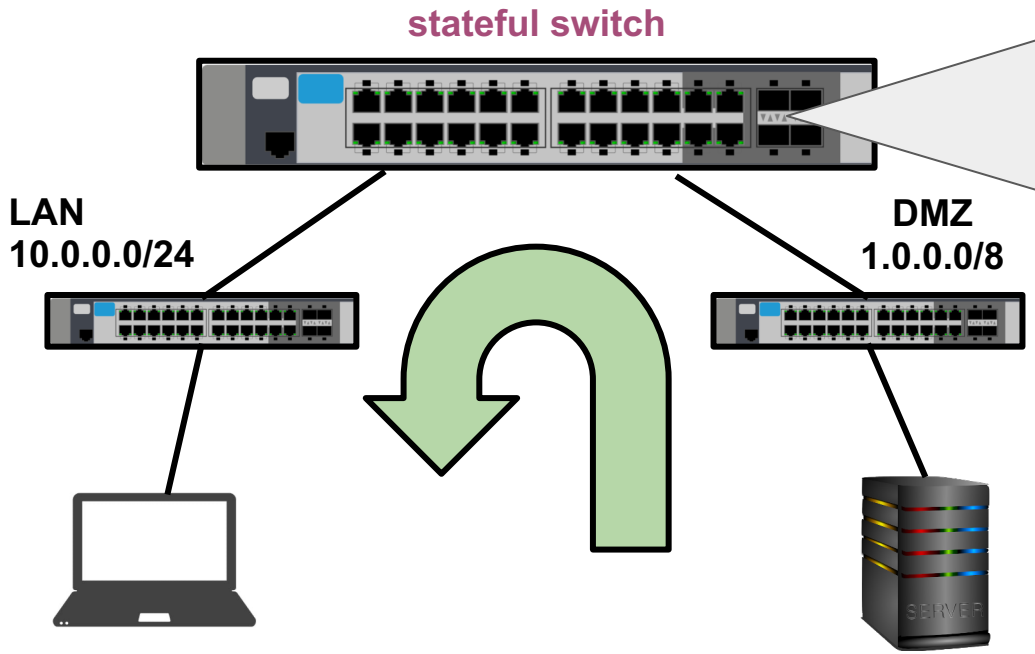
flow CTX table

flow key	*state*
(10.0.0.100, 1.0.0.1, 1000, 2000, tcp)	OPEN

flow keys

port 0 → (ip.src, ip.dst, sport, dport, proto)
port 1 → (ip.dst, ip.src, dport, sport, proto)

Stateful Firewall - Flow Context table



flow CTX table

flow key	*state*
(10.0.0.100, 1.0.0.1, 1000, 2000, tcp)	OPEN

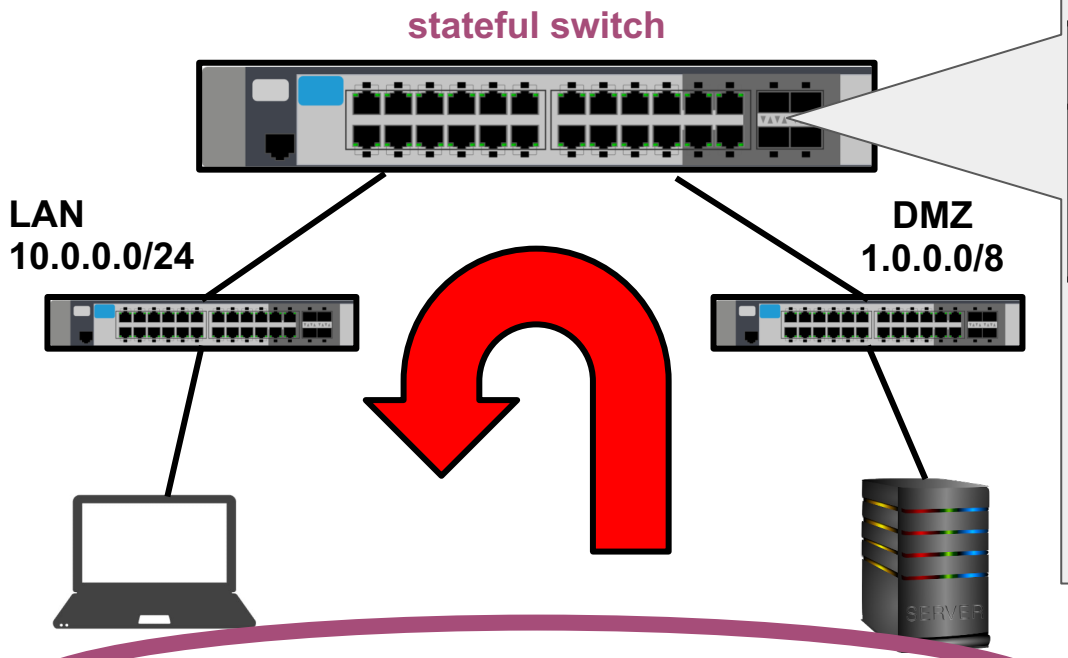
flow keys

port 0 → (ip.src, ip.dst, sport, dport, proto)
port 1 → (ip.dst, ip.src, dport, sport, proto)

Flow Key: (10.0.0.100, 1.0.0.1, 1000, 2000, tcp)

Flow from DMZ in state OPEN? ⇨ **ALLOW**

Stateful Firewall - Flow Context table



New Flow: (10.0.0.100, 1.0.0.1, 3000, 4000, tcp)

flow CTX table

flow key	*state*
(10.0.0.100, 1.0.0.1, 1000, 2000, tcp)	OPEN

flow keys

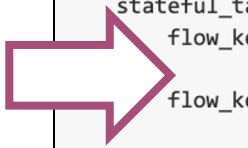
port 0 → (ip.src, ip.dst, sport, dport, proto)
port 1 → (ip.dst, ip.src, dport, sport, proto)

flow from DMZ not in the flow CTX table? ⇨ **DROP!!**

Stateful Firewall - P4(ext.) implementation

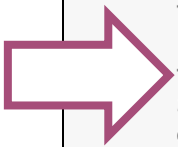
```
control IngressPipeImpl (inout parsed_headers_t hdr,
                         inout local_metadata_t local_metadata,
                         inout standard_metadata_t standard_metadata) {

    stateful_table stage_0 {
        flow_key[0] = {hdr.ipv4.src, hdr.ipv4.dst, hdr.ipv4.ip_proto,
                       local_metadata.l4_src_port, local_metadata.l4_dst_port};
        flow_key[1] = {hdr.ipv4.dst, hdr.ipv4.src, hdr.ipv4.ip_proto,
                       local_metadata.l4_dst_port, local_metadata.l4_src_port};
        flow_ctx = ctx_0(8);
        idle_timeout = 30000;
        eviction_policy = LRU;
        size = 4096;
        graph = graph_0(flow_ctx, hdr, standard_metadata,
                        local_metadata);
    }

    apply {
        if (standard_metadata.ingress_port == LAN) {
            stage_0.apply(0);
        } else {
            stage_0.apply(1);
        }
    }
}
```

Stateful Firewall - P4(ext.) implementation



```
control IngressPipeImpl (inout parsed_headers_t hdr,
                         inout local_metadata_t local_metadata,
                         inout standard_metadata_t standard_metadata) {

    stateful_table stage_0 {
        flow_key[0] = {hdr.ipv4.src, hdr.ipv4.dst, hdr.ipv4.ip_proto,
                       local_metadata.l4_src_port, local_metadata.l4_dst_port};
        flow_key[1] = {hdr.ipv4.dst, hdr.ipv4.src, hdr.ipv4.ip_proto,
                       local_metadata.l4_dst_port, local_metadata.l4_src_port};
        flow_ctx = ctx_0(8);
        idle_timeout = 30000;
        eviction_policy = LRU;
        size = 4096;
        graph = graph_0(flow_ctx, hdr, standard_metadata,
                        local_metadata);
    }

    apply {
        if (standard_metadata.ingress_port == LAN) {
            stage_0.apply(0);
        } else {
            stage_0.apply(1);
        }
    }
}
```

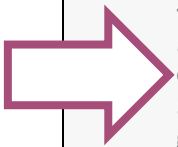
Stateful Firewall - P4(ext.) implementation



```
control IngressPipeImpl (inout parsed_headers_t hdr,
                         inout local_metadata_t local_metadata,
                         inout standard_metadata_t standard_metadata) {

    stateful_table stage_0 {
        flow_key[0] = {hdr.ipv4.src, hdr.ipv4.dst, hdr.ipv4.ip_proto,
                       local_metadata.l4_src_port, local_metadata.l4_dst_port};
        flow_key[1] = {hdr.ipv4.
                       local_met
        flow_ctx = ctx_0(8);
        idle_timeout = 30000;
        eviction_policy = LRU;
        size = 4096;
        graph = graph_0(flow_ctx, hdr, standard_metadata,
                        local_metadata);
    }

    apply {
        if (standard_metadata.ingress_port == LAN) {
            stage_0.apply(0);
        } else {
            stage_0.apply(1);
        }
    }
}
```

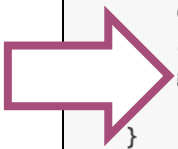
```
state_context ctx_0(bit<8> state_size) {
    bit<32> state;
}
```

Stateful Firewall - P4(ext.) implementation



```
control IngressPipeImpl (inout parsed_headers_t hdr,
                         inout local_metadata_t local_metadata,
                         inout standard_metadata_t standard_metadata) {

    stateful_table stage_0 {
        flow_key[0] = {hdr.ipv4.src, hdr.ipv4.dst, hdr.ipv4.ip_proto,
                       local_metadata.l4_src_port, local_metadata.l4_dst_port};
        flow_key[1] = {hdr.ipv4.dst, hdr.ipv4.src, hdr.ipv4.ip_proto,
                       local_metadata.l4_dst_port, local_metadata.l4_src_port};
        flow_ctx = ctx_0(8);
        idle_timeout = 30000;
        eviction_policy = LRU;
        size = 4096;
        graph = graph_0(flow_ctx, hdr, standard_metadata,
                        local_metadata);
    }

    apply {
        if (standard_metadata.ingress_port == LAN) {
            stage_0.apply(0);
        } else {
            stage_0.apply(1);
        }
    }
}
```

Stateful Firewall - P4(ext.) implementation



```
control IngressPipeImpl (inout parsed_headers_t hdr,
                         inout local_metadata_t local_metadata,
                         inout standard_metadata_t standard_metadata) {

    stateful_table stage_0 {
        flow_key[0] = {hdr.ipv4.src, hdr.ipv4.dst, hdr.ipv4.ip_proto,
                       local_metadata.l4_src_port, local_metadata.l4_dst_port};
        flow_key[1] = {hdr.ipv4.dst, hdr.ipv4.src, hdr.ipv4.ip_proto,
                       local_metadata.l4_dst_port, local_metadata.l4_src_port};
        flow_ctx = ctx_0(8);
        idle_timeout = 30000;
        eviction_policy = LRU;
        size = 4096;
        graph = graph_0(flow_ctx, hdr, standard_metadata,
                        local_metadata);
    }

    apply {
        if (standard_metadata.ingress_port == LAN) {
            stage_0.apply(0);
        } else {
            stage_0.apply(1);
        }
    }
}
```

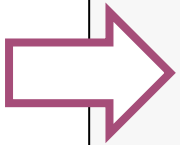
Stateful Firewall - P4(ext.) implementation

```
state_graph graph_0(inout state_context flow_ctx, inout headers_t hdr,
                    inout standard_metadata_t standard_metadata) {
    state start {
        if (standard_metadata.ingress_port == LAN) {
            standard_metadata.egress_spec = DMZ;
            transition established;
        }
        else if (standard_metadata.ingress_port == DMZ) {
            mark_to_drop();
        }
    }

    state established {
        if (standard_metadata.ingress_port == LAN) {
            standard_metadata.egress_spec = DMZ;
        }
        else if (standard_metadata.ingress_port == DMZ) {
            standard_metadata.egress_spec = LAN;
        }
    }
}
```

Stateful Firewall - P4(ext.) implementation

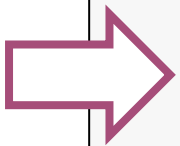
```
state_graph graph_0(inout state_context flow_ctx, inout headers_t hdr,
                    inout standard_metadata_t standard_metadata) {
    state start {
        if (standard_metadata.ingress_port == LAN) {
            standard_metadata.egress_spec = DMZ;
            transition established;
        }
        else if (standard_metadata.ingress_port == DMZ) {
            mark_to_drop();
        }
    }

    state established {
        if (standard_metadata.ingress_port == LAN) {
            standard_metadata.egress_spec = DMZ;
        }
        else if (standard_metadata.ingress_port == DMZ) {
            standard_metadata.egress_spec = LAN;
        }
    }
}
```

Stateful Firewall - P4(ext.) implementation



```
state_graph graph_0(inout state_context flow_ctx, inout headers_t hdr,
                    inout standard_metadata_t standard_metadata) {
    state start {
        if (standard_metadata.ingress_port == LAN) {
            standard_metadata.egress_spec = DMZ;
            transition established;
        }
        else if (standard_metadata.ingress_port == DMZ) {
            mark_to_drop();
        }
    }

    state established {
        if (standard_metadata.ingress_port == LAN) {
            standard_metadata.egress_spec = DMZ;
        }
        else if (standard_metadata.ingress_port == DMZ) {
            standard_metadata.egress_spec = LAN;
        }
    }
}
```

Stateful Firewall - P4(ext.) implementation

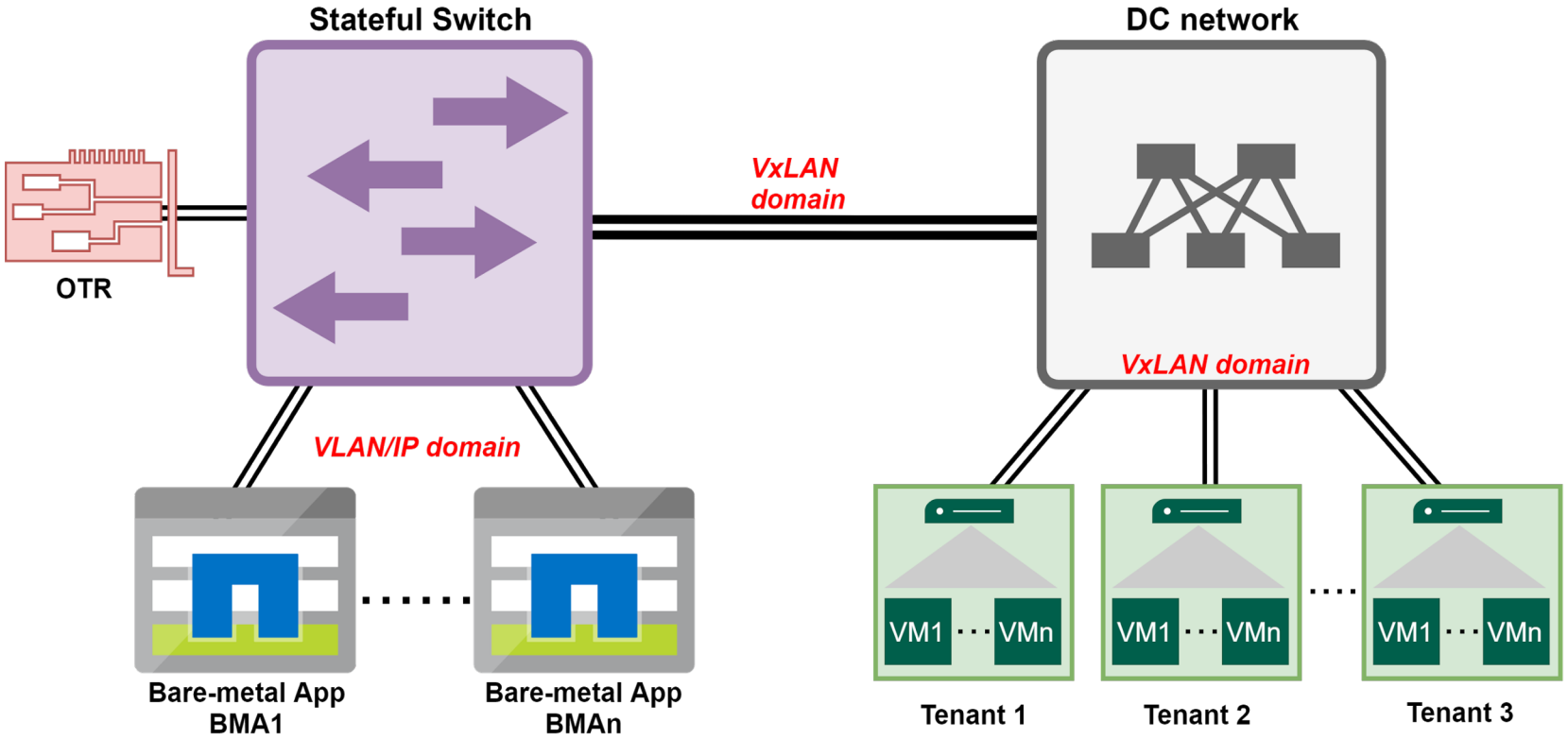


```
state_graph graph_0(inout state_context flow_ctx, inout headers_t hdr,
                    inout standard_metadata_t standard_metadata) {
    state start {
        if (standard_metadata.ingress_port == LAN) {
            standard_metadata.egress_spec = DMZ;
            transition established;
        }
        else if (standard_metadata.ingress_port == DMZ) {
            mark_to_drop();
        }
    }

    state established {
        if (standard_metadata.ingress_port == LAN) {
            standard_metadata.egress_spec = DMZ;
        }
        else if (standard_metadata.ingress_port == DMZ) {
            standard_metadata.egress_spec = LAN;
        }
    }
}
```

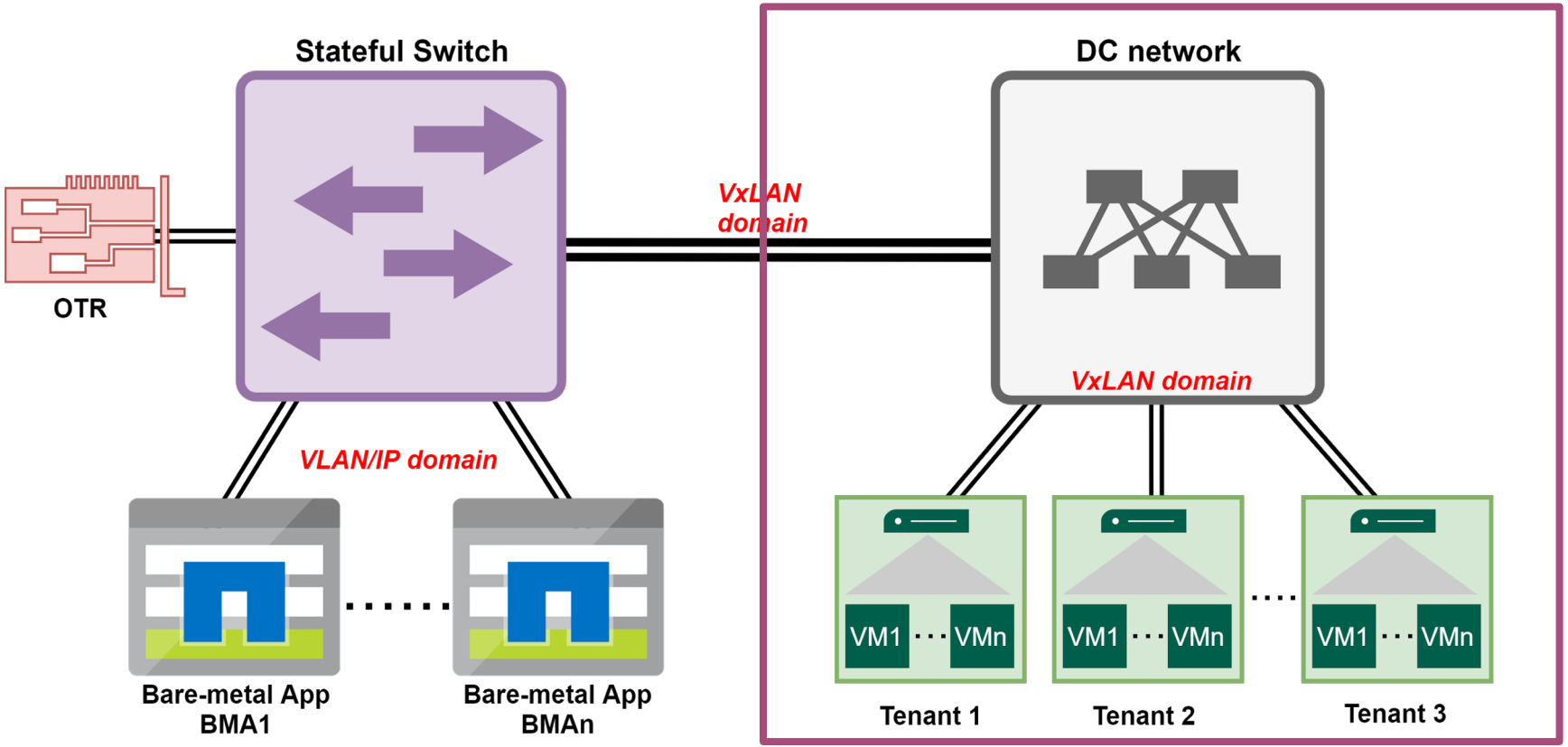
The Flow Cache use case

repository: *https://github.com/axbryd/p4-flow-cache*

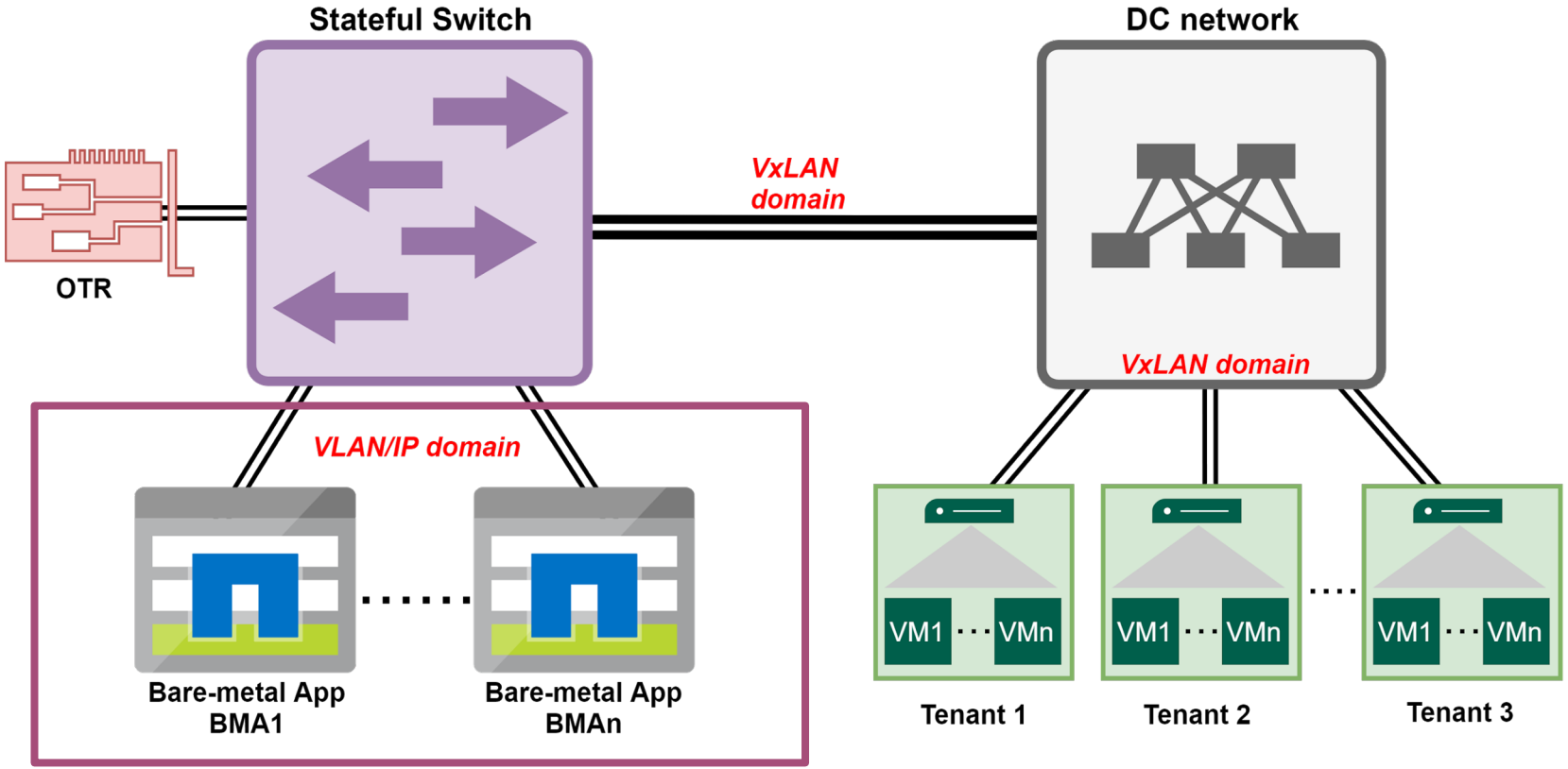
Flow Cache


Stateful Switch
DC network
OTR
VxLAN
domain
VxLAN domain
VLAN/IP domain
Bare-metal App
BMA1
Bare-metal App
BMAn
VM1
VMn
VM1
VMn
VM1
VMn
Tenant 1
Tenant 2
Tenant 3

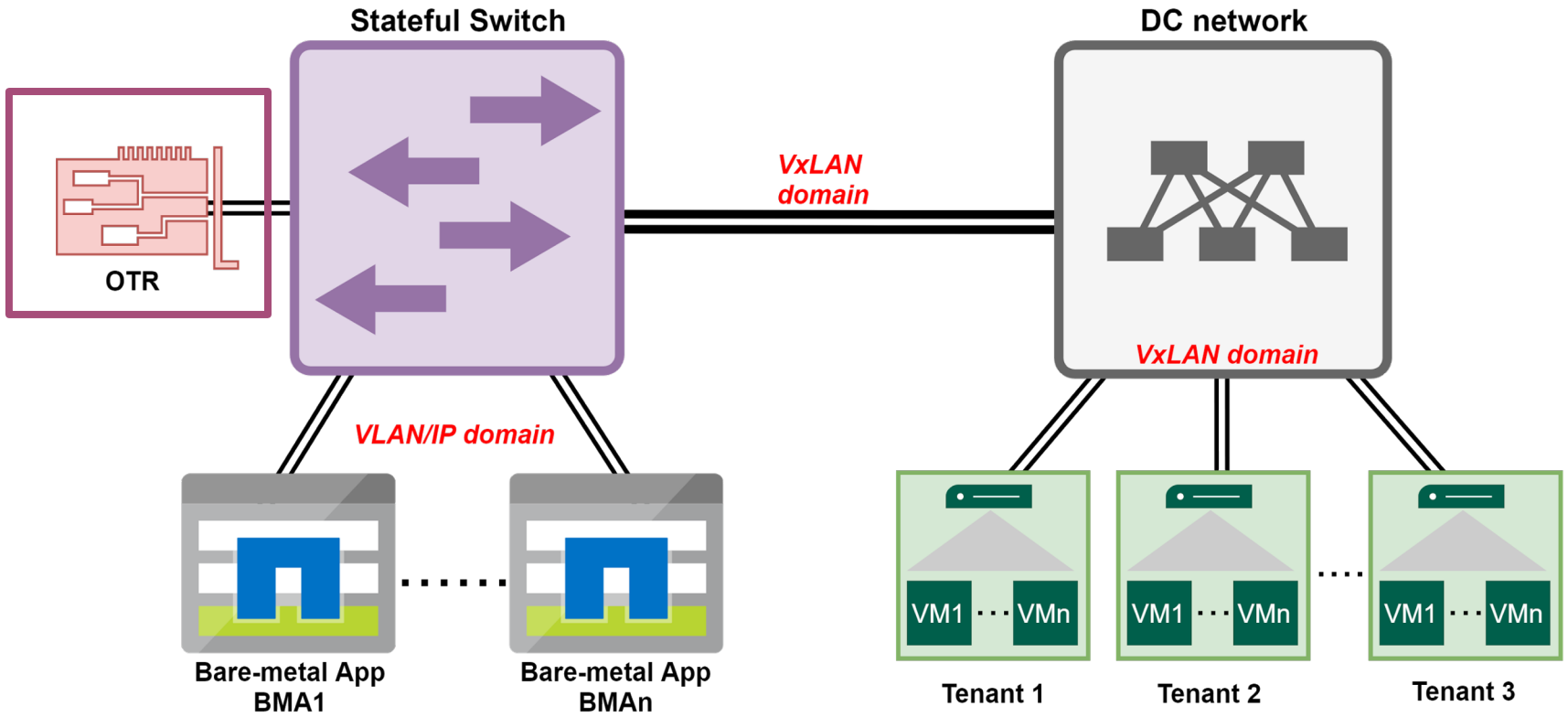
Flow Cache


Stateful Switch
DC network
OTR
VxLAN
domain
VxLAN domain
VLAN/IP domain
VM1
VMn
VM1
VMn
VM1
VMn
Bare-metal App
BMA1
Bare-metal App
BMAn
Tenant 1
Tenant 2
Tenant 3

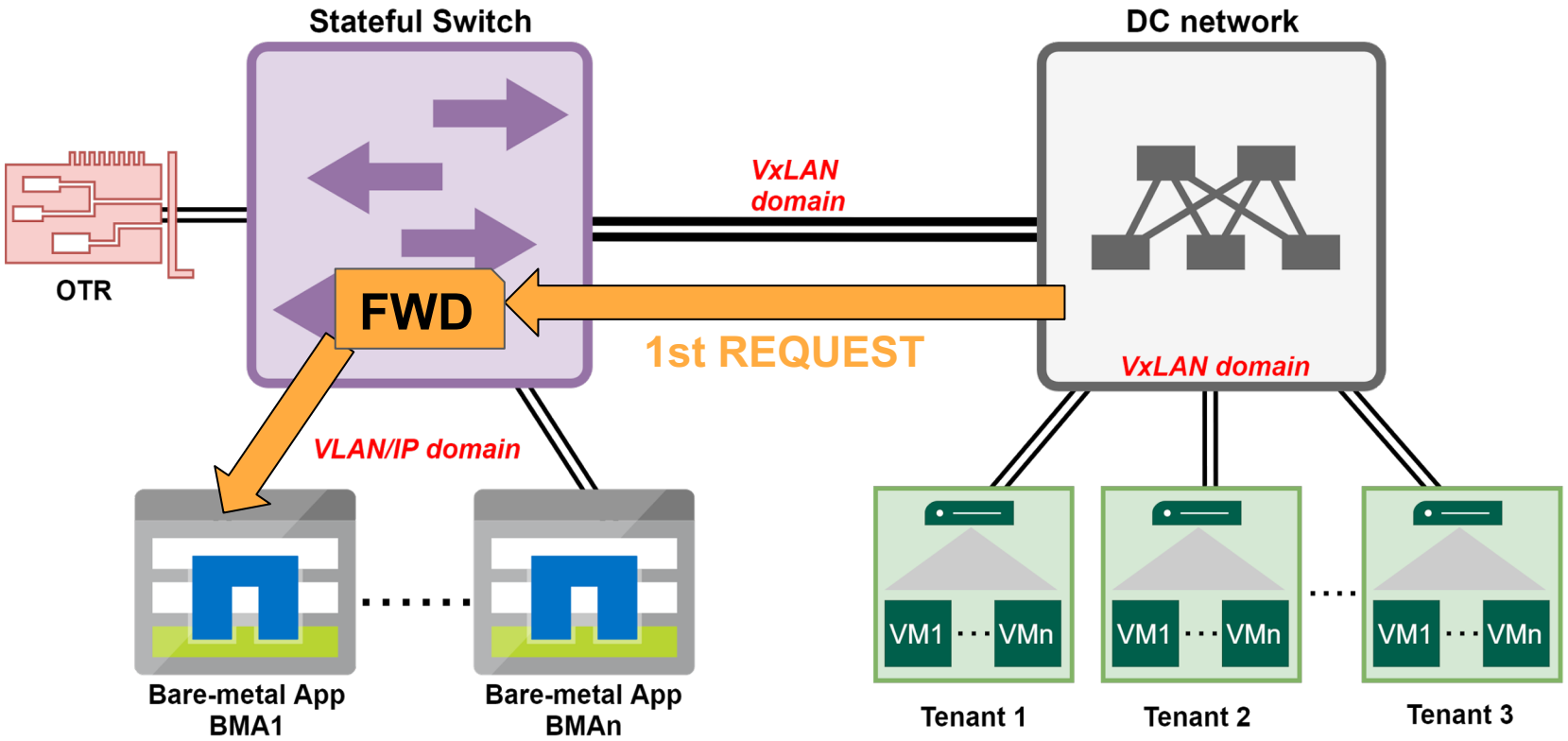
Flow Cache


Stateful Switch
DC network
OTR
VxLAN
domain
VxLAN domain
VLAN/IP domain
Bare-metal App
BMA1
Bare-metal App
BMAn
VM1
VMn
VM1
VMn
VM1
VMn
Tenant 1
Tenant 2
Tenant 3

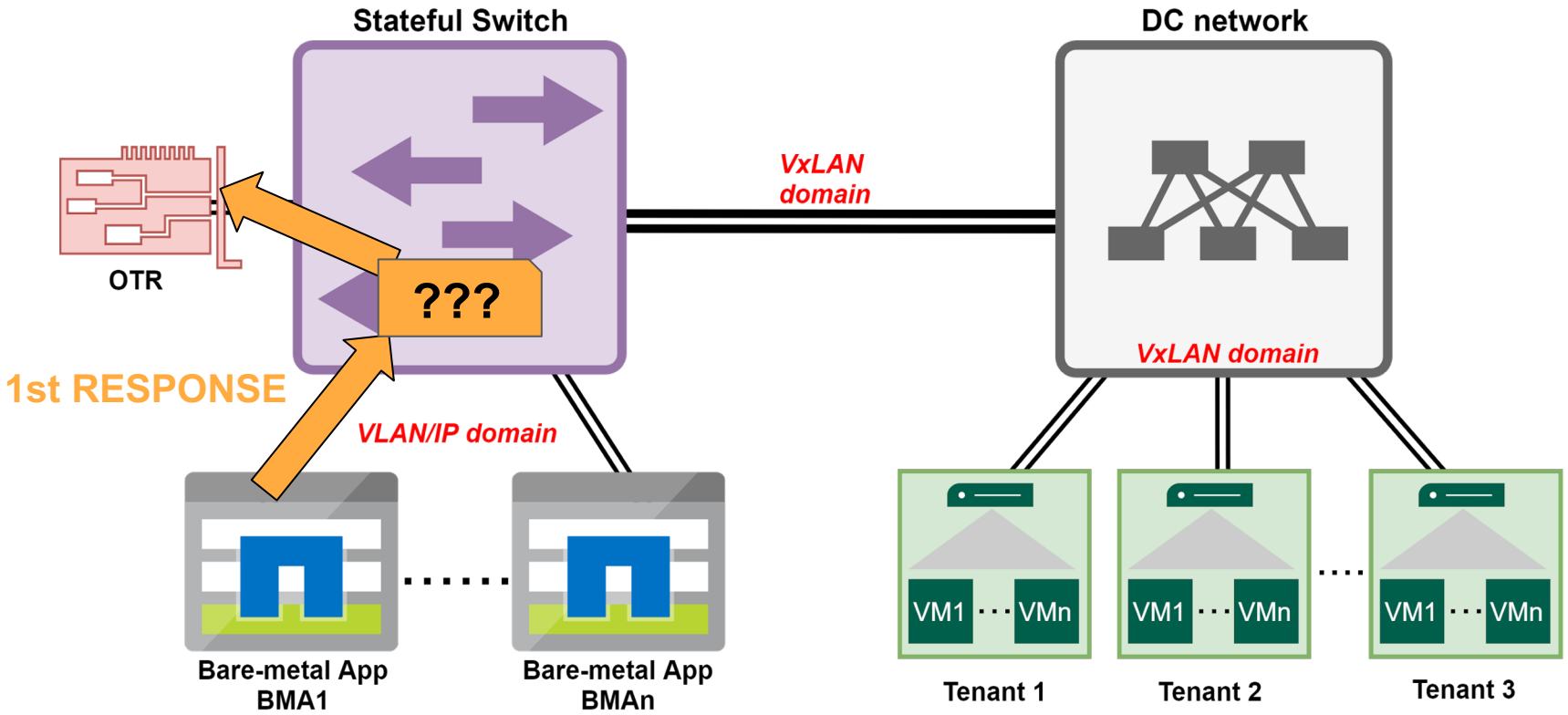
Flow Cache


Stateful Switch
DC network
OTR
VxLAN
domain
VxLAN domain
VLAN/IP domain
VM1
VMn
VM1
VMn
VM1
VMn
Bare-metal App
BMA1
Bare-metal App
BMAn
Tenant 1
Tenant 2
Tenant 3

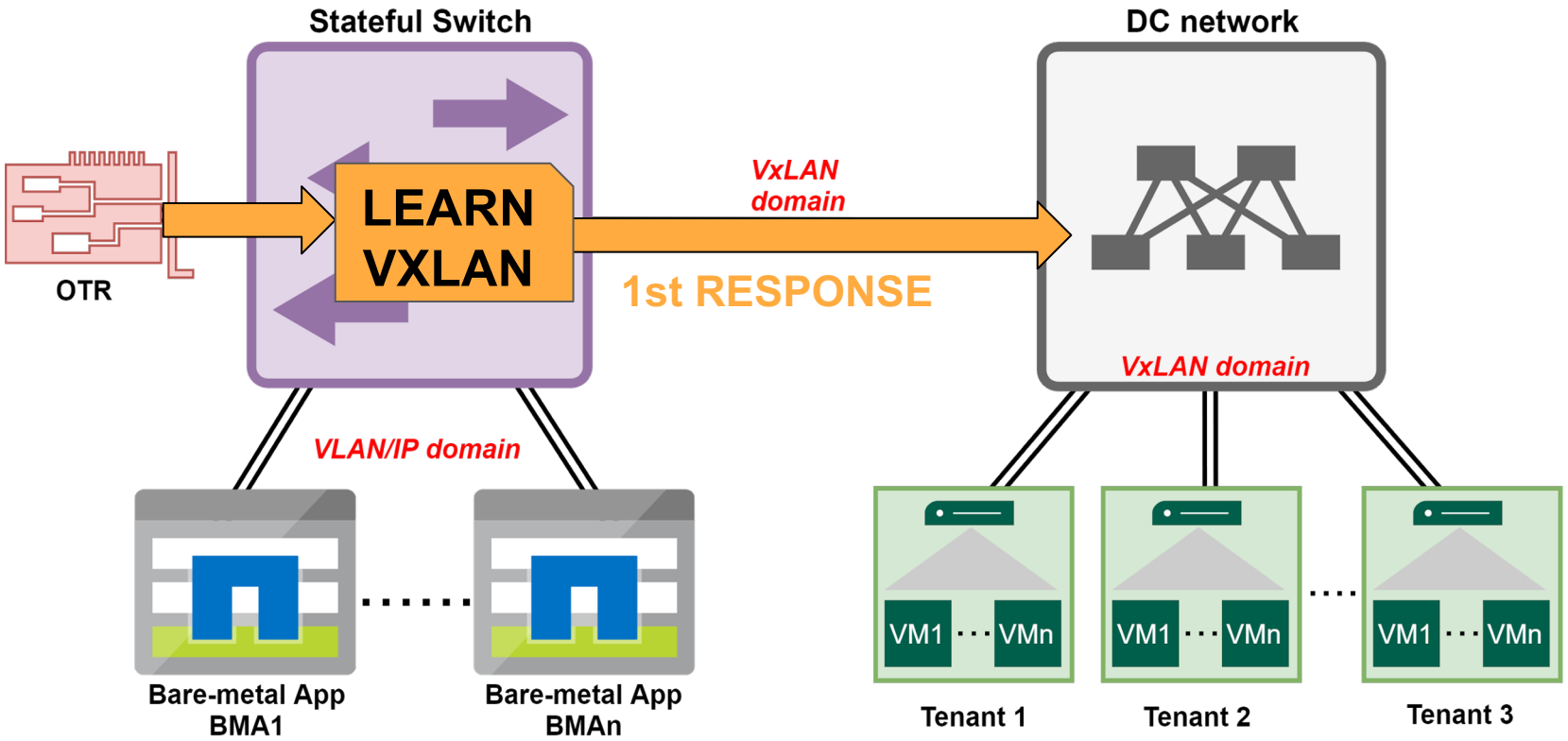
Flow Cache


Stateful Switch
DC network
OTR
VxLAN
domain
FWD
1st REQUEST
VxLAN domain
VLAN/IP domain
VM1
VMn
VM1
VMn
VM1
VMn
Bare-metal App
BMA1
Bare-metal App
BMAn
Tenant 1
Tenant 2
Tenant 3

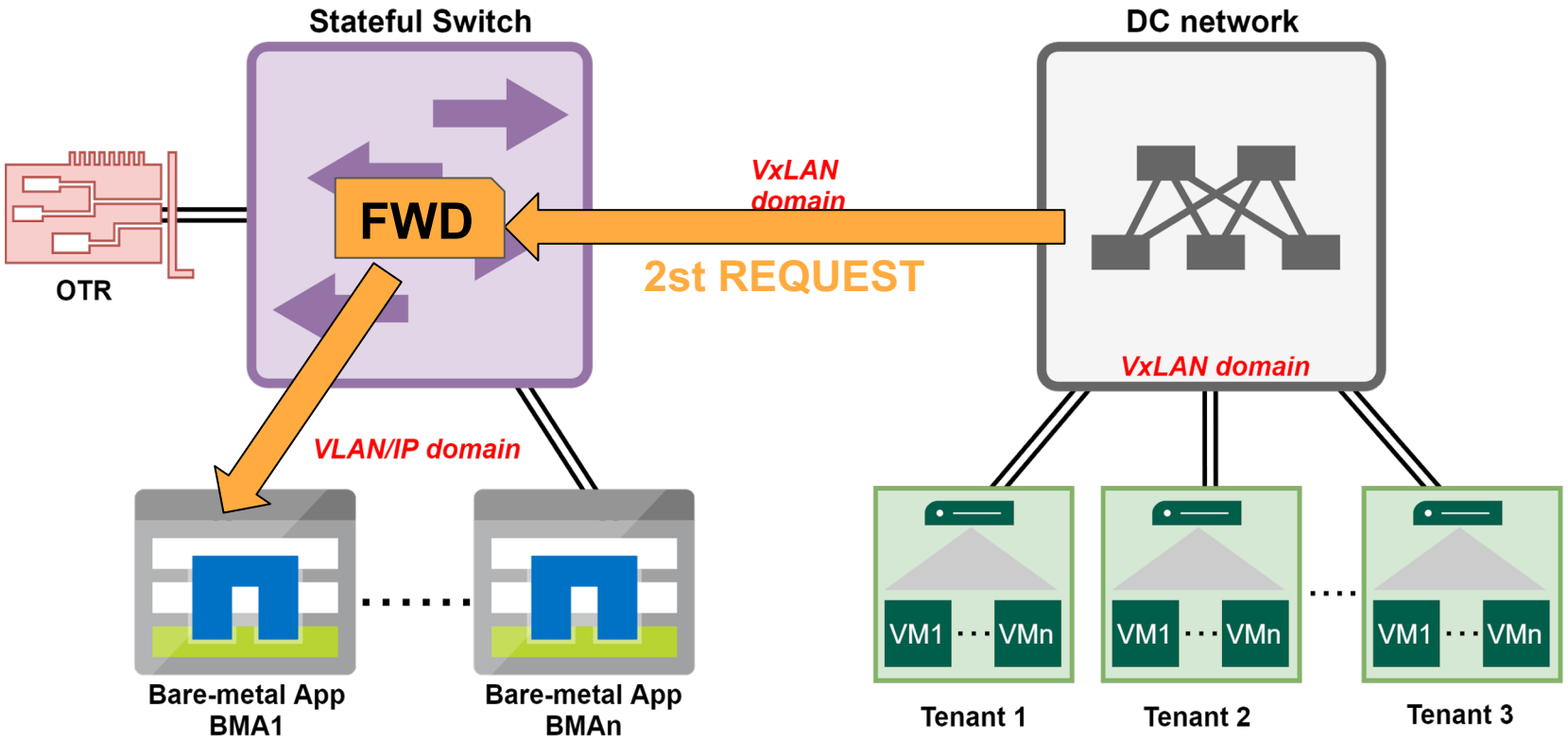
Flow Cache


Stateful Switch
DC network
VxLAN
domain
OTR
???
1st RESPONSE
VLAN/IP domain
VxLAN domain
Bare-metal App
BMA1
Bare-metal App
BMAn
VM1
VMn
VM1
VMn
VM1
VMn
Tenant 1
Tenant 2
Tenant 3

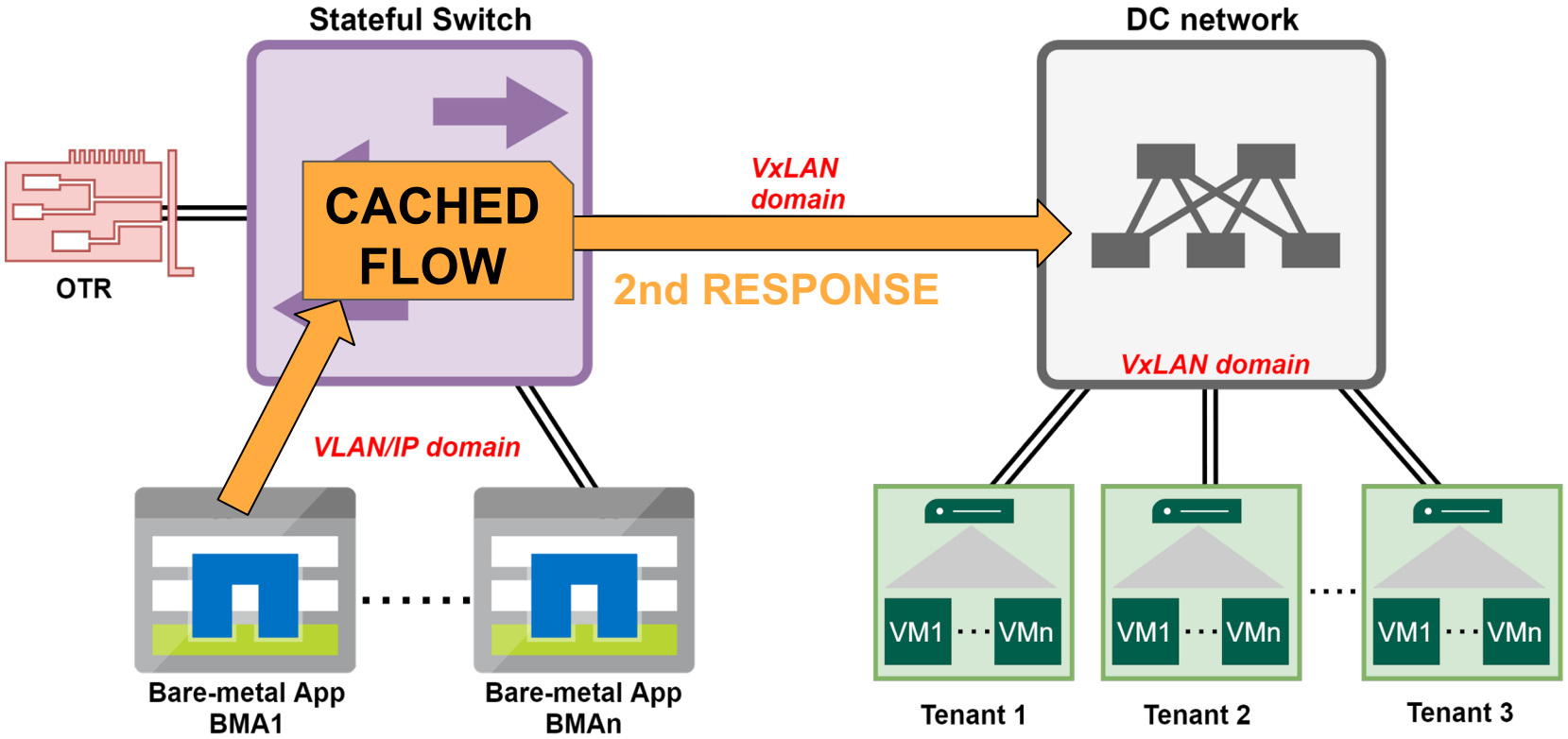
Flow Cache


Stateful Switch
DC network
OTR
LEARN
VXLAN
VxLAN
domain
1st RESPONSE
VxLAN domain
VLAN/IP domain
Bare-metal App
BMA1
Bare-metal App
BMAn
VM1
VMn
VM1
VMn
VM1
VMn
Tenant 1
Tenant 2
Tenant 3

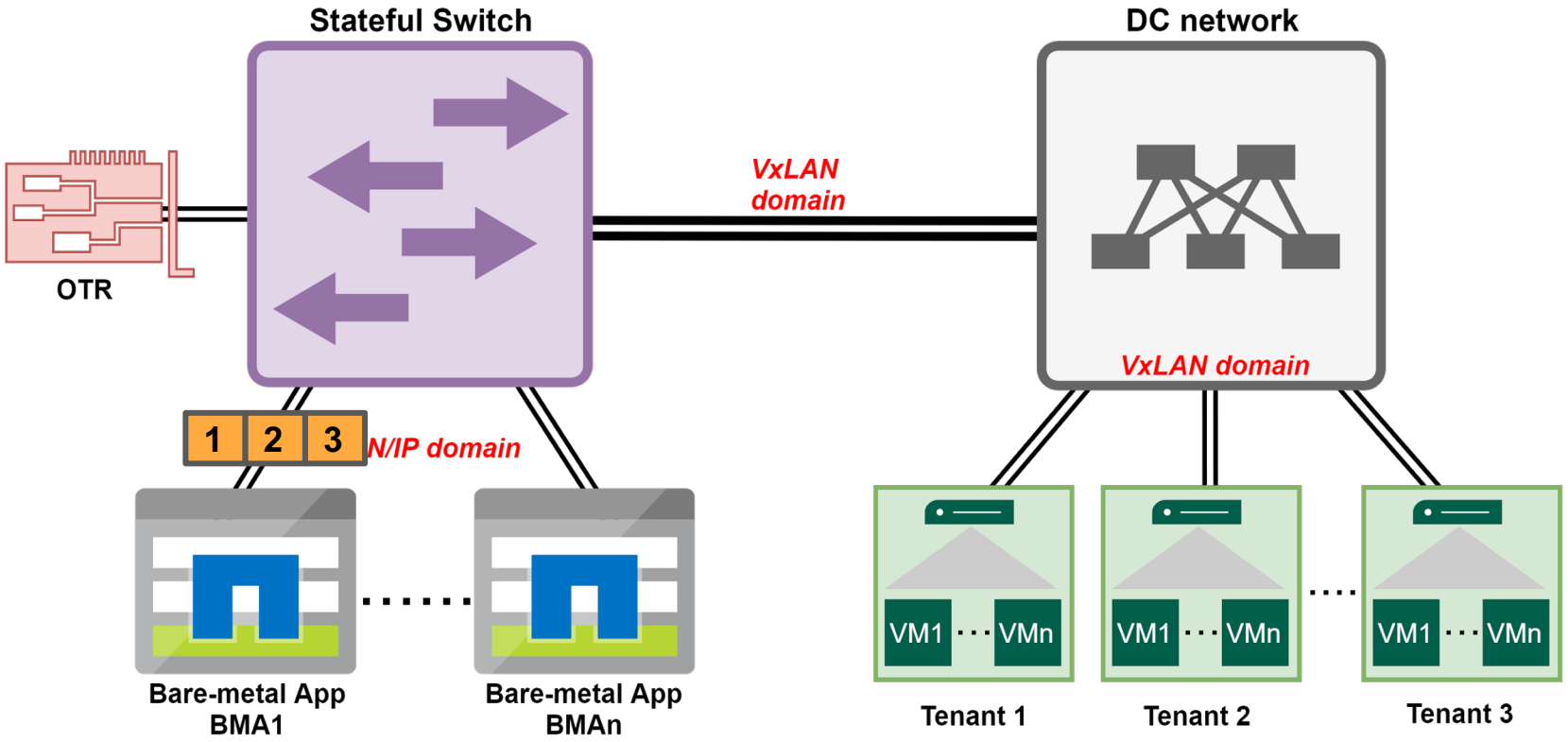
Flow Cache


Stateful Switch
DC network
FWD
OTR
VxLAN
domain
2st REQUEST
VxLAN domain
VLAN/IP domain
Bare-metal App
BMA1
Bare-metal App
BMAn
VM1
VMn
VM1
VMn
VM1
VMn
Tenant 1
Tenant 2
Tenant 3

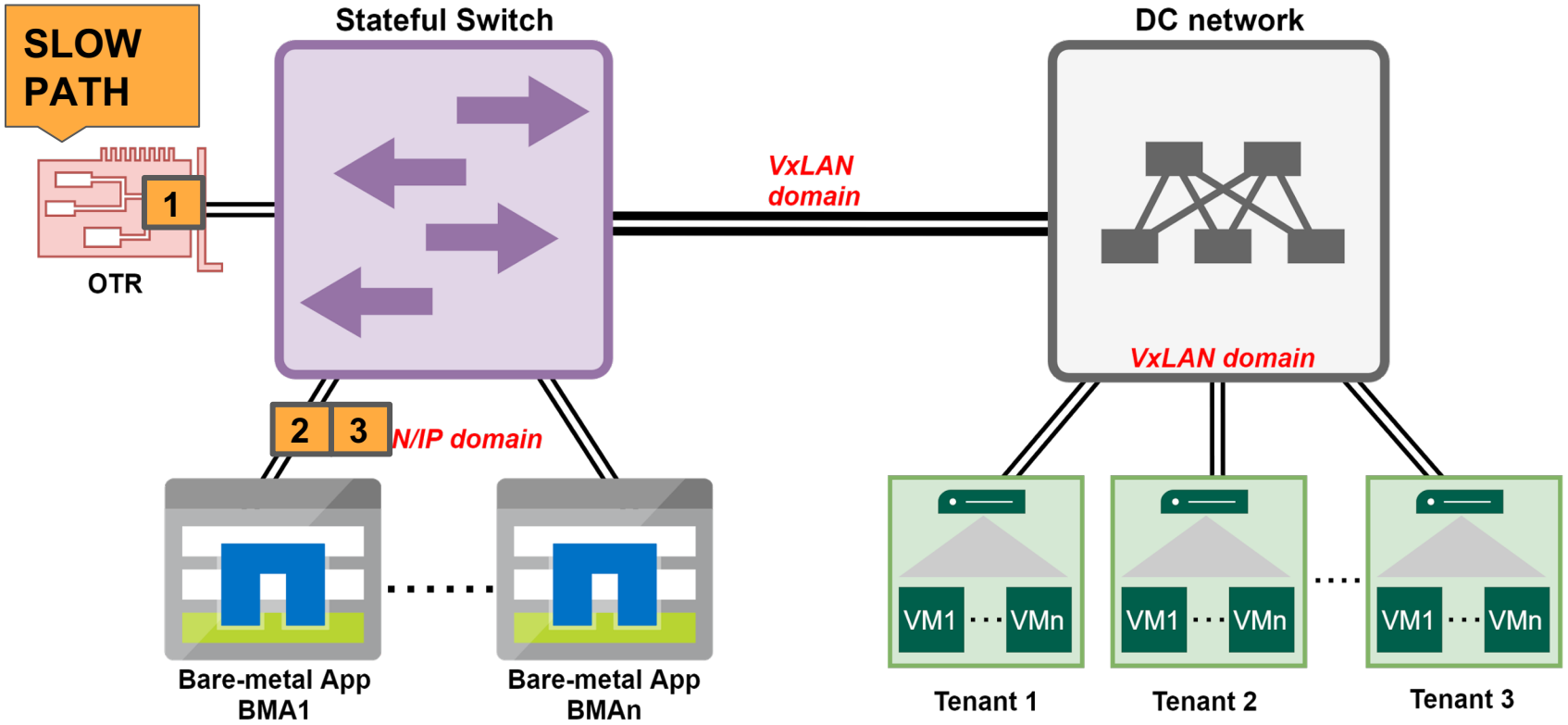
Flow Cache


Stateful Switch
DC network
OTR
CACHED FLOW
VxLAN domain
2nd RESPONSE
VxLAN domain
VLAN/IP domain
Bare-metal App BMA1
Bare-metal App BMAn
VM1
VMn
VM1
VMn
VM1
VMn
Tenant 1
Tenant 2
Tenant 3

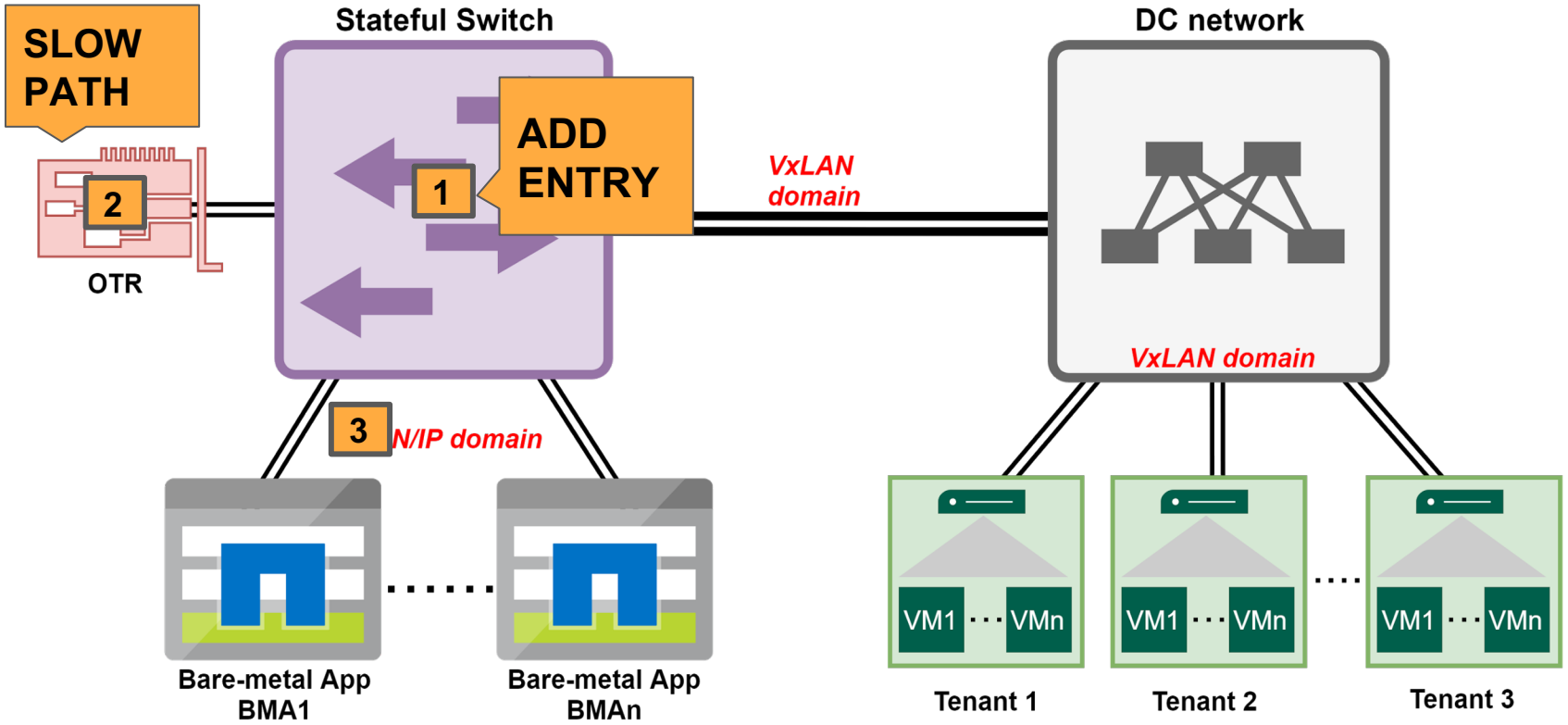
Flow Cache (out-of-order problem)



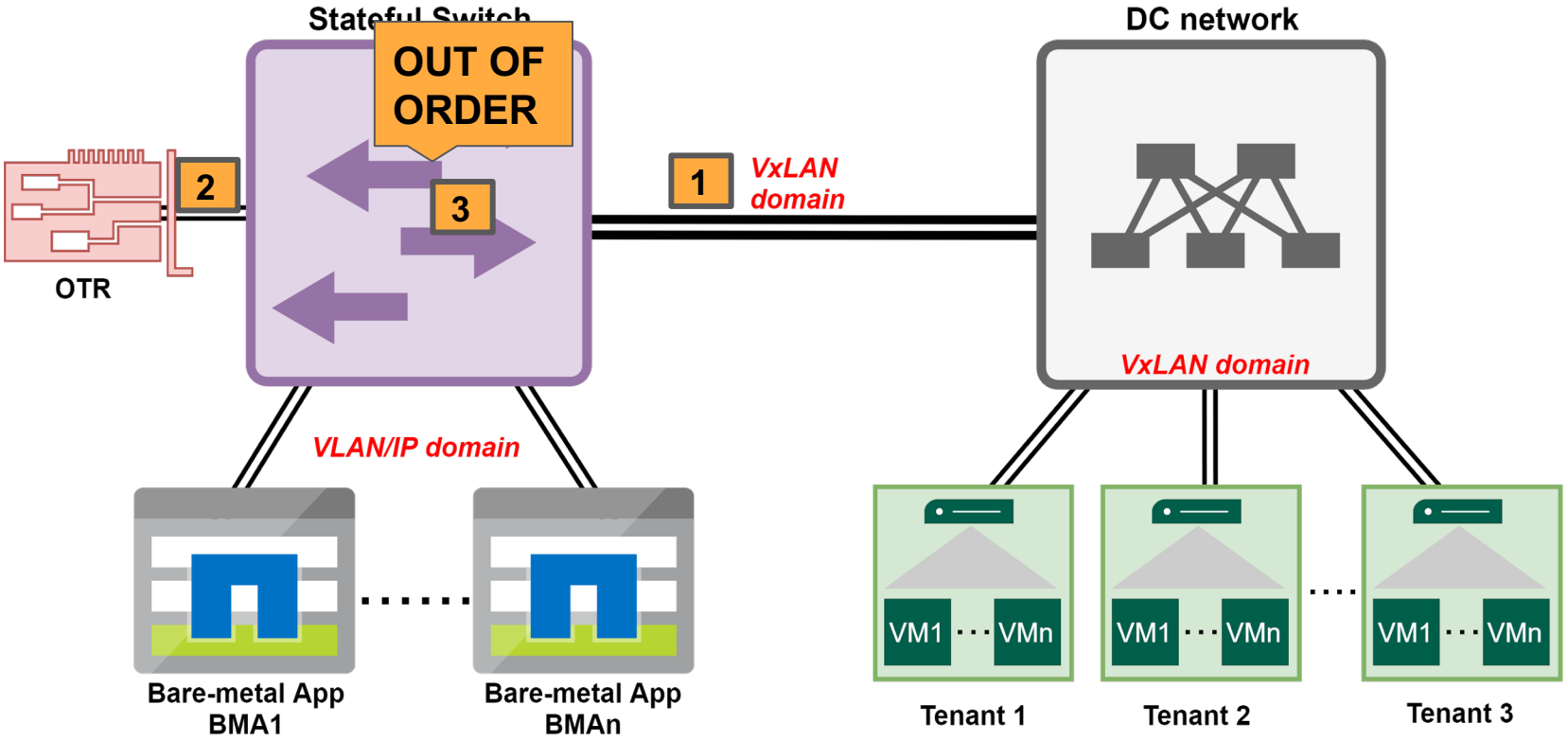
Flow Cache (out-of-order problem)



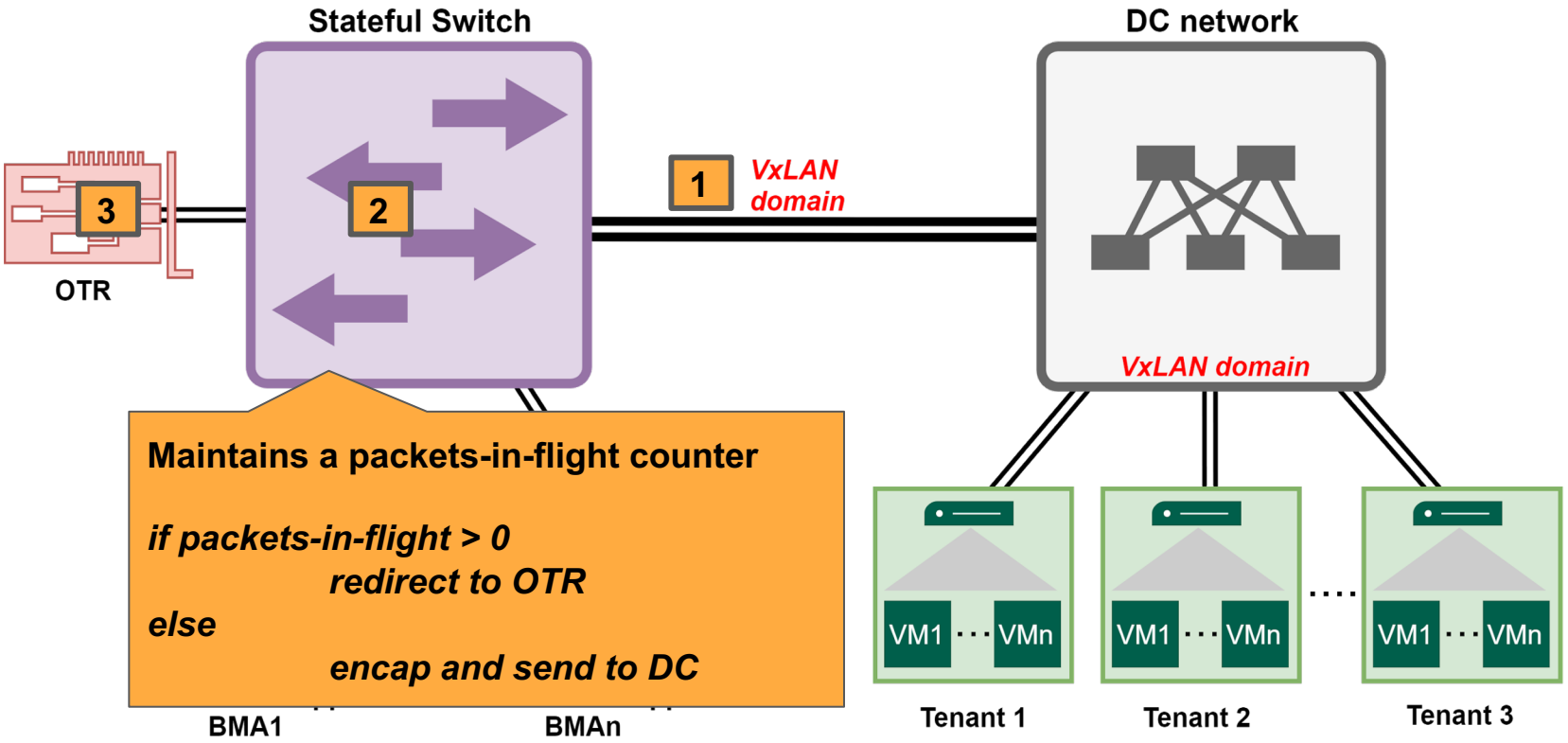
Flow Cache (out-of-order problem)



Flow Cache (out-of-order problem)



Flow Cache (out-of-order solution)



DEMO

https://youtu.be/6U54E-7Lzq0

Conclusions

- We defined the P4 language extensions to define a new programming abstraction for stateful Network Functions
  - implemented a prototype in bmv2
- Functional assessment of a set of real-world use cases
- *Next steps…*
  - Front-end compiler
  - Support for other use cases (e.g. In-network computation, etc.)
  - Map the language extensions to HW design (NVIDIA)

Thank You

tulumello@axbryd.com
www.axbryd.com