Paving the Way for NFV: Simplifying Middlebox Modifications with StateAlyzr

Junaid Khalid, Aaron Gember-Jacobson,
Roney Michael, Archie Abhashkumar, Aditya Akella

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Middleboxes

Middleboxes

Perform sophisticated operations on network traffic

Middleboxes

Perform sophisticated operations on network traffic

Firewall

Middleboxes

Perform sophisticated operations on network traffic

Firewall

Intrusion detection system (IDS)

Middleboxes

Perform sophisticated operations on network traffic

Firewall

Intrusion detection system (IDS)

Caching proxy

Middleboxes

Perform sophisticated operations on network traffic

Firewall

Intrusion detection system (IDS)

Caching proxy

Maintain state about connections and hosts

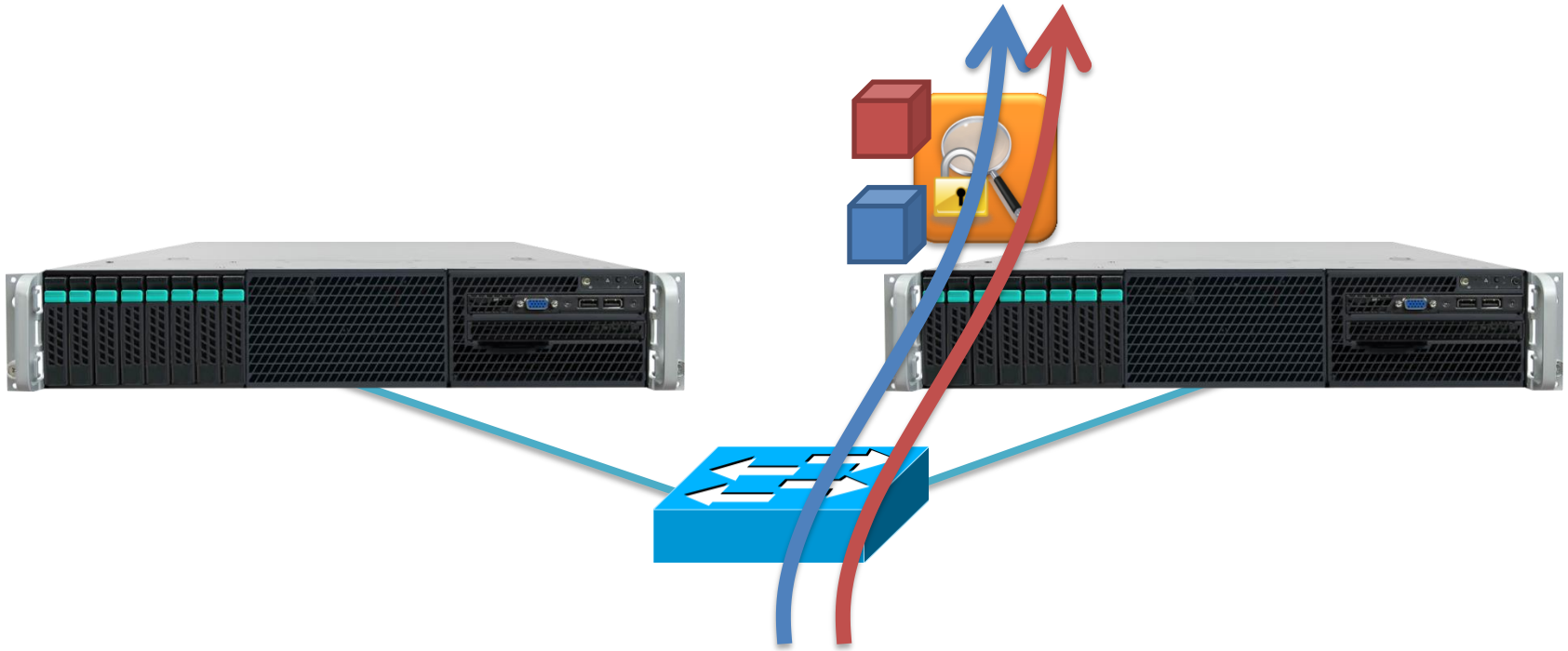
Network Function Virtualization (NFV)

Network Function Virtualization (NFV)

NFV enables ***elastic scaling*** and ***high availability***

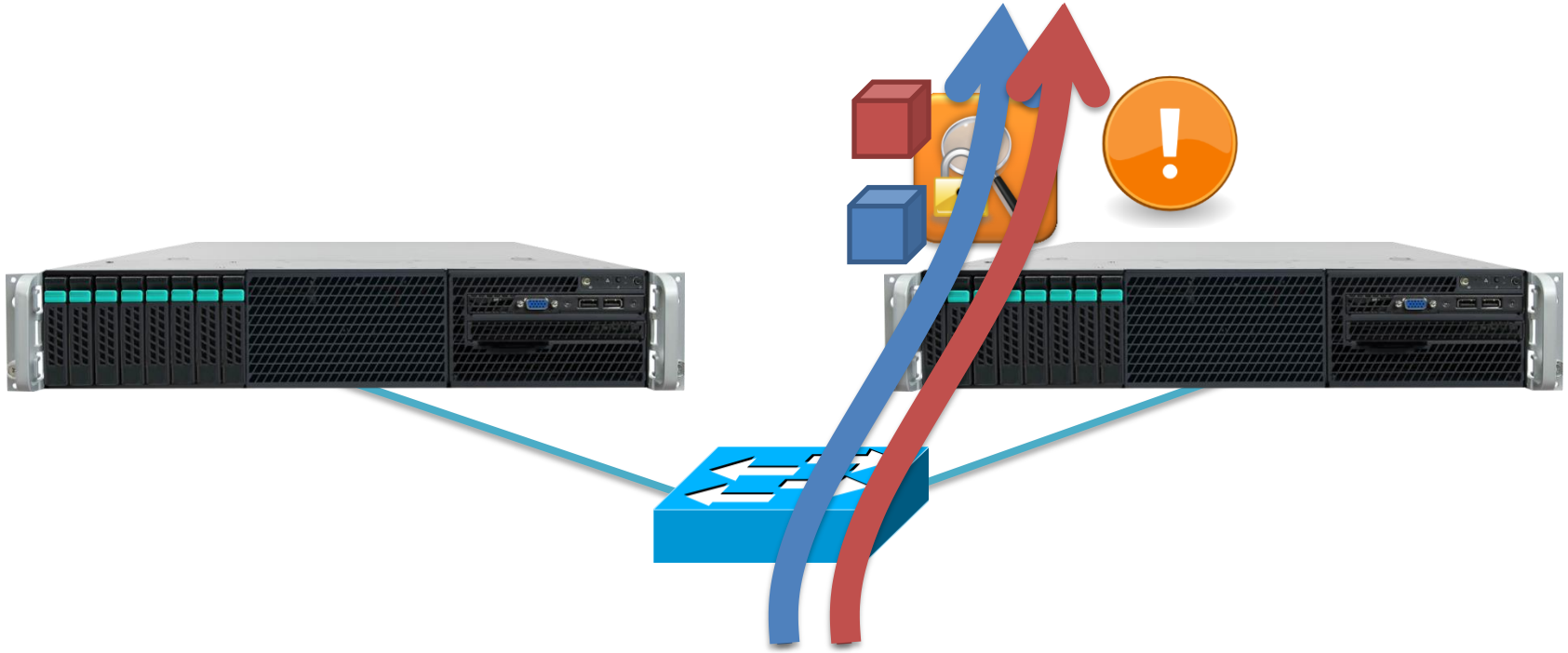
Network Function Virtualization (NFV)

NFV enables ***elastic scaling*** and ***high availability***



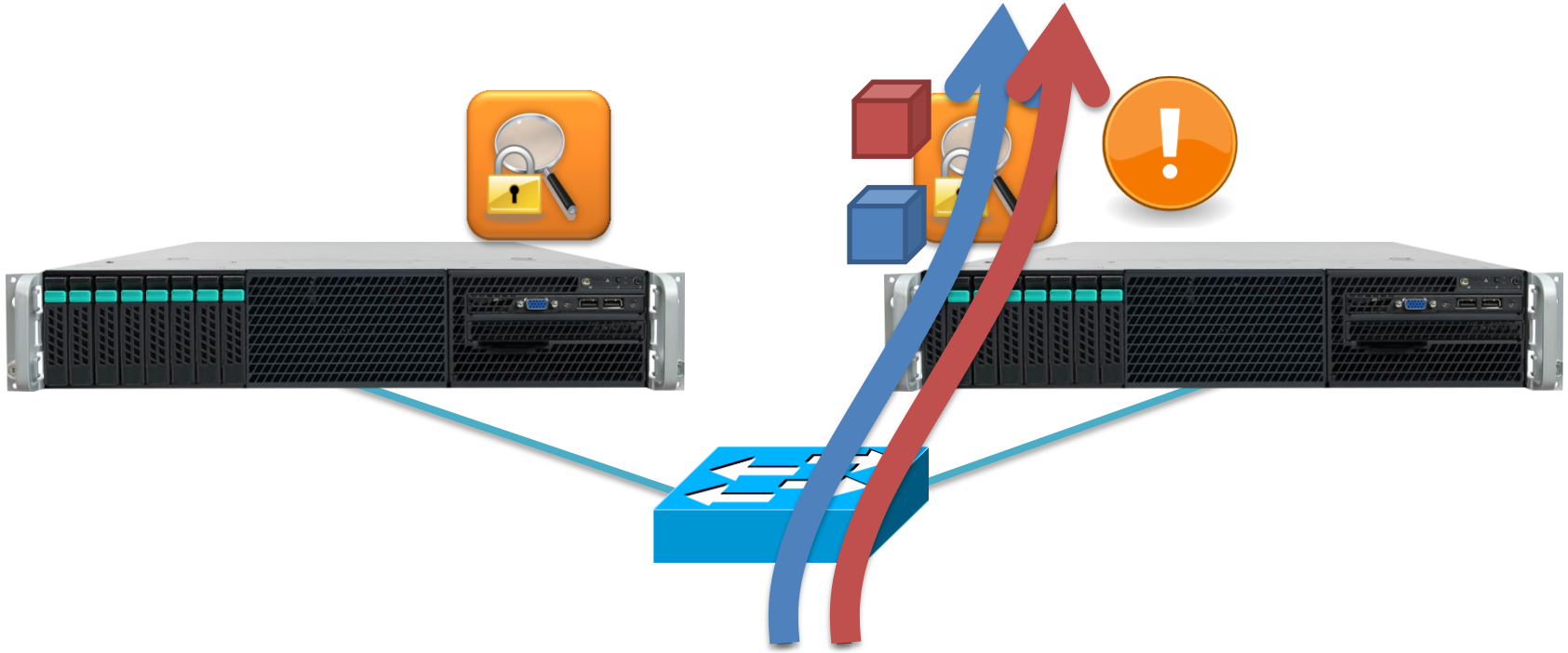
Network Function Virtualization (NFV)

NFV enables ***elastic scaling*** and ***high availability***



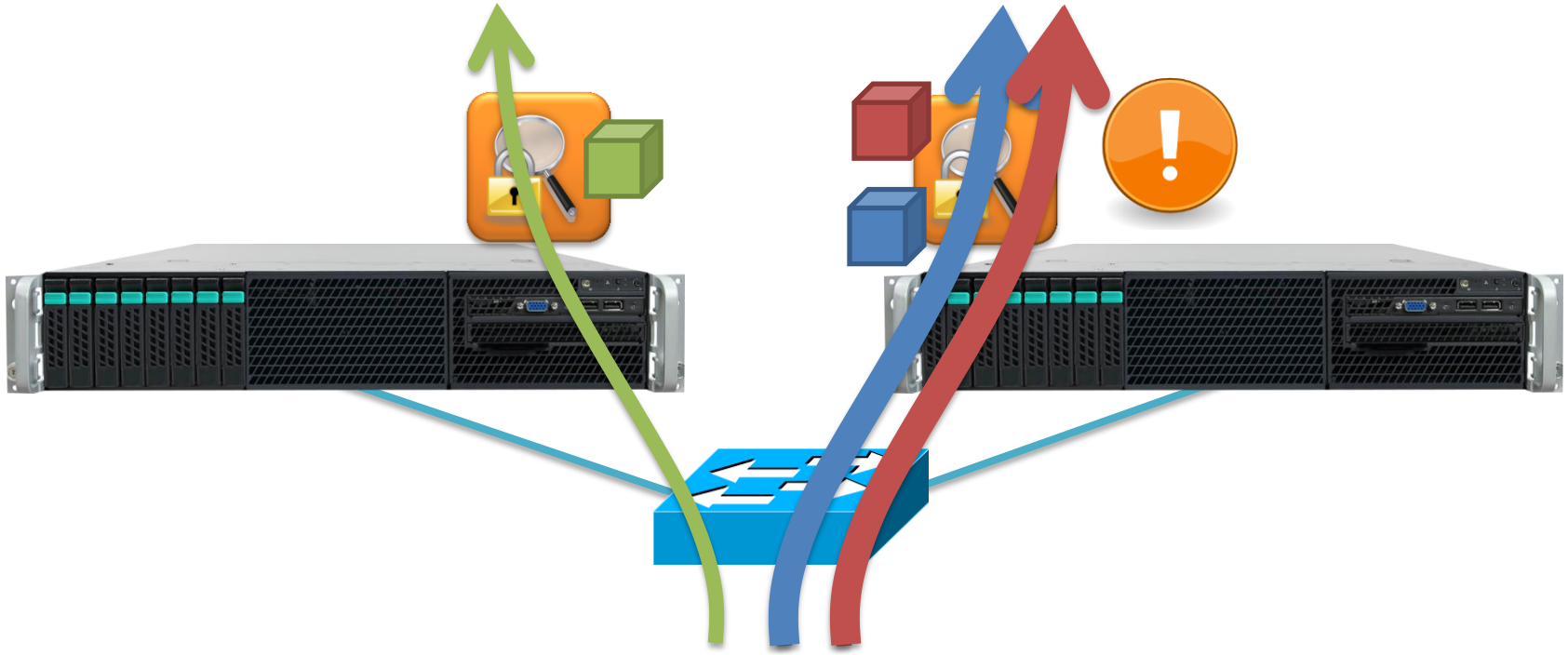
Network Function Virtualization (NFV)

NFV enables ***elastic scaling*** and ***high availability***



Network Function Virtualization (NFV)

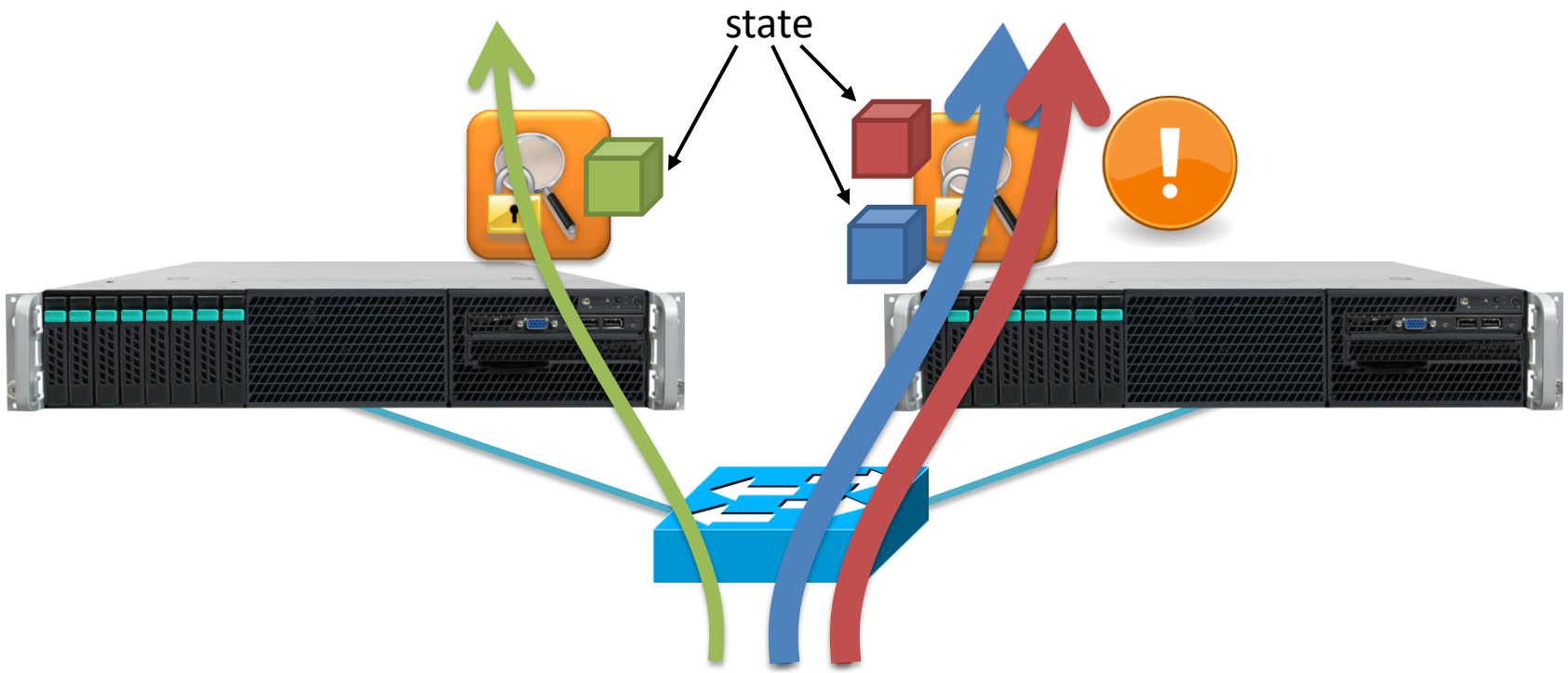
NFV enables ***elastic scaling*** and ***high availability***



Reroute new connections

Network Function Virtualization (NFV)

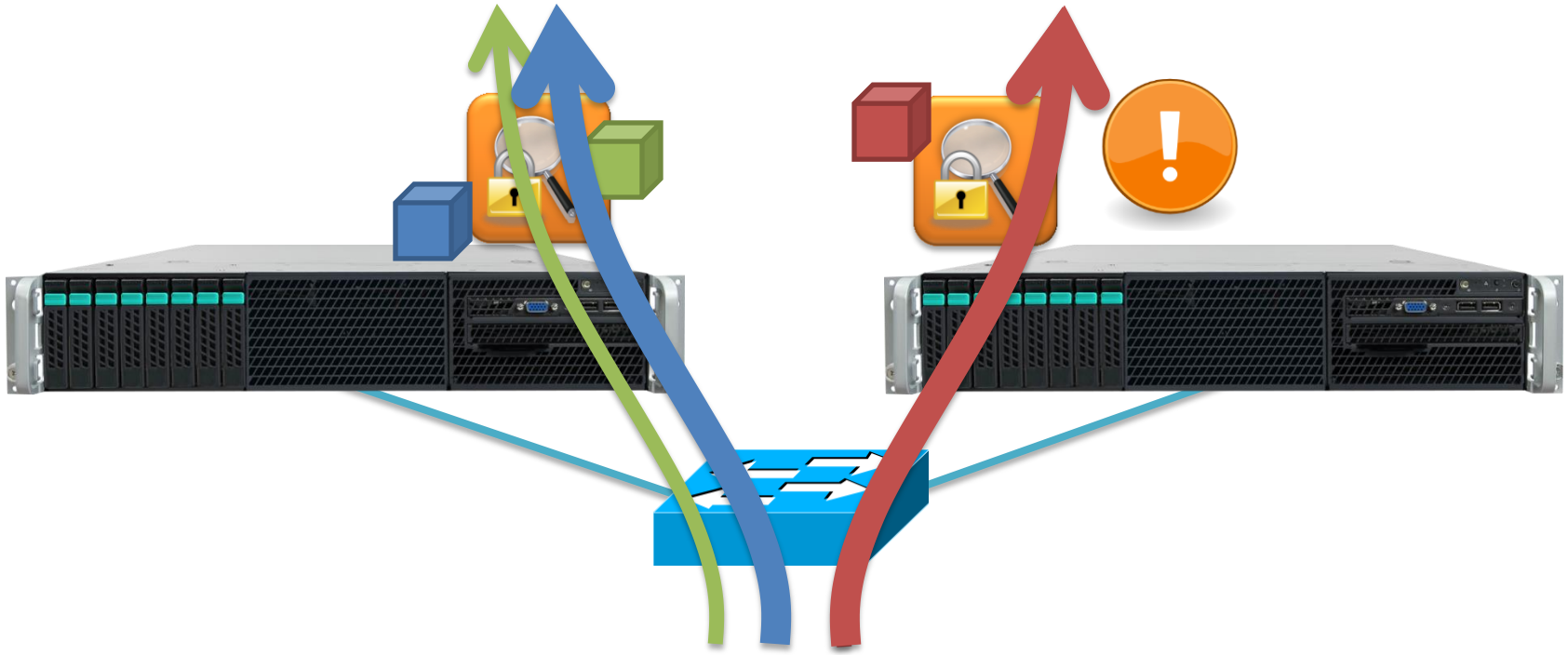
NFV enables ***elastic scaling*** and ***high availability***



Reroute new connections

Network Function Virtualization (NFV)

NFV enables ***elastic scaling*** and ***high availability***



Reroute ~~new~~ existing connections

State taxonomy

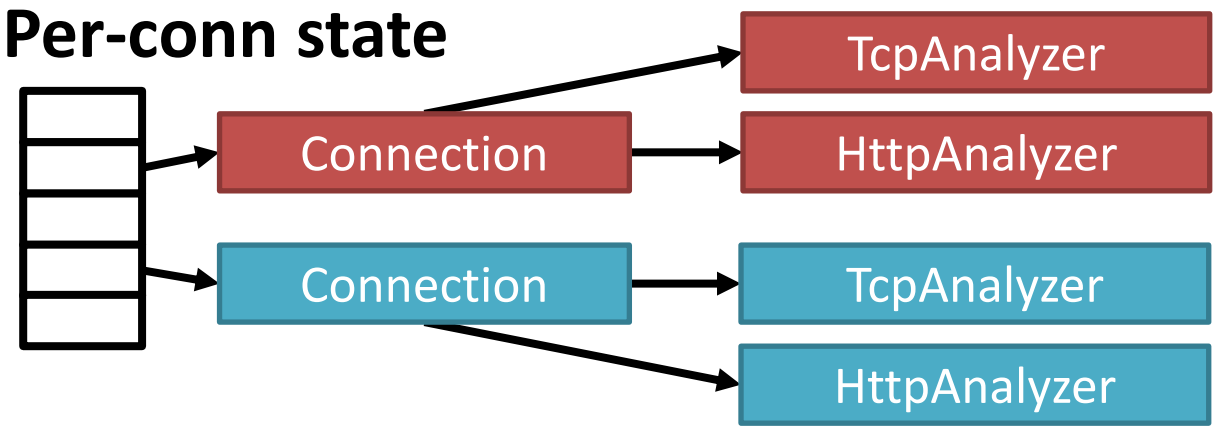
State created or updated by a middlebox applies to either a **single connection** or a **set of connections**

State taxonomy

State created or updated by a middlebox applies to either a **single connection** or a **set of connections**

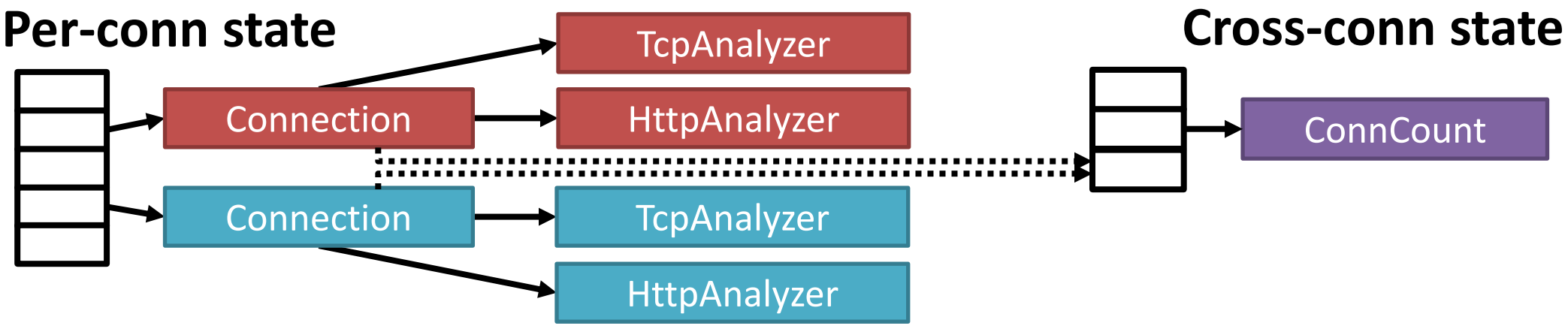
State taxonomy

State created or updated by a middlebox applies to either a **single connection** or a **set of connections**



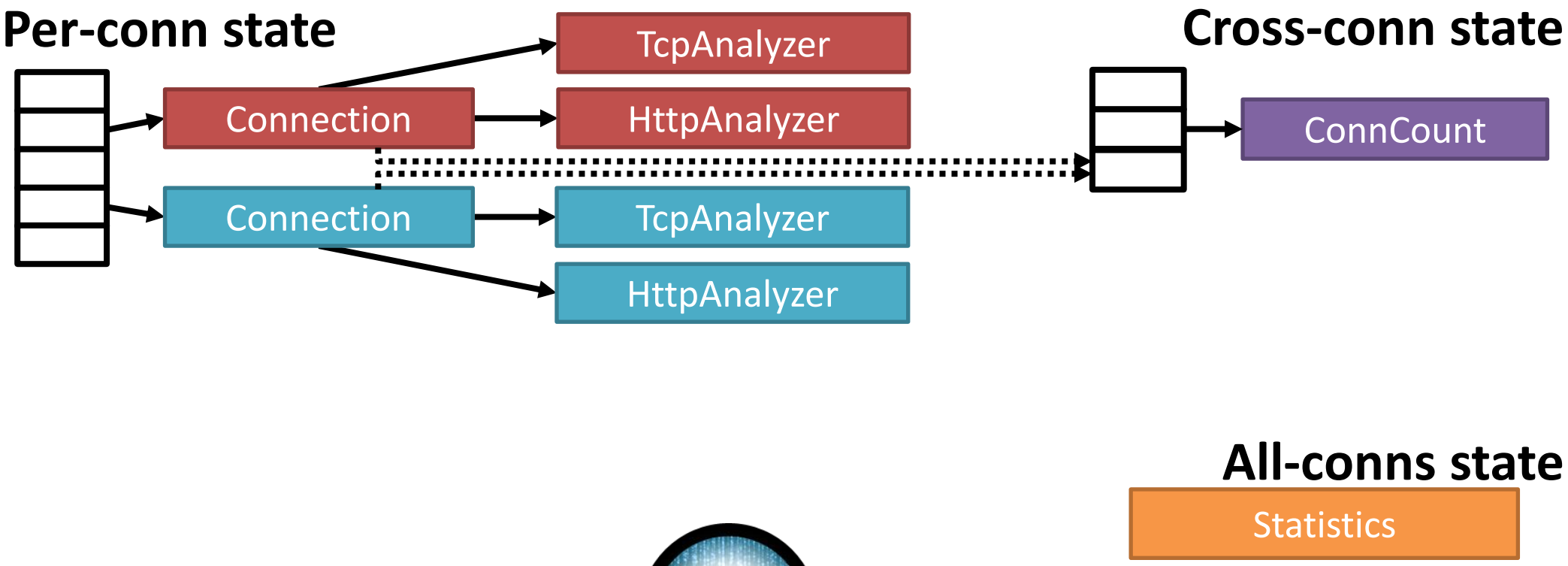
State taxonomy

State created or updated by a middlebox applies to either a **single connection** or a **set of connections**



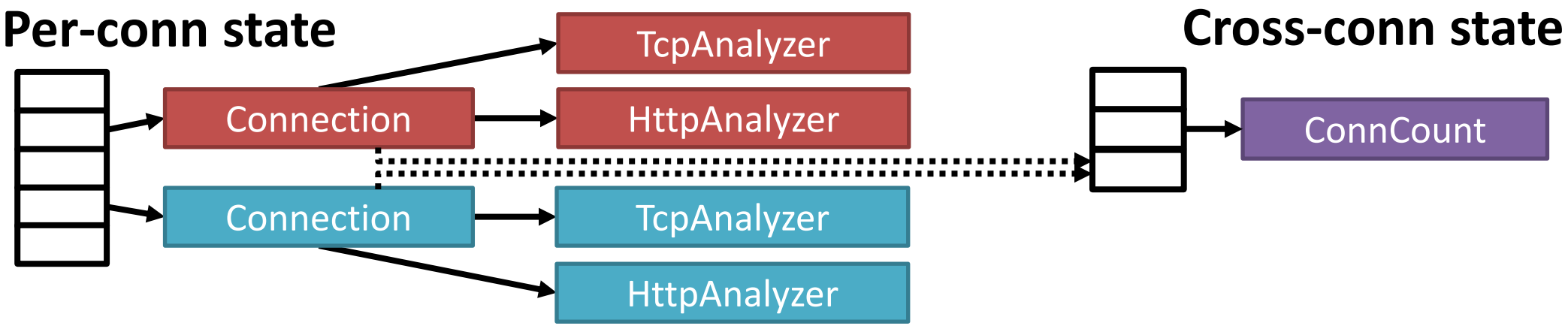
State taxonomy

State created or updated by a middlebox applies to either a **single connection** or a **set of connections**



State taxonomy

State created or updated by a middlebox applies to either a **single connection** or a **set of connections**



Input state

Config + Sig

All-conns state

Statistics

NFV state management -> middlebox modification

NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

NFV state management -> middlebox modification

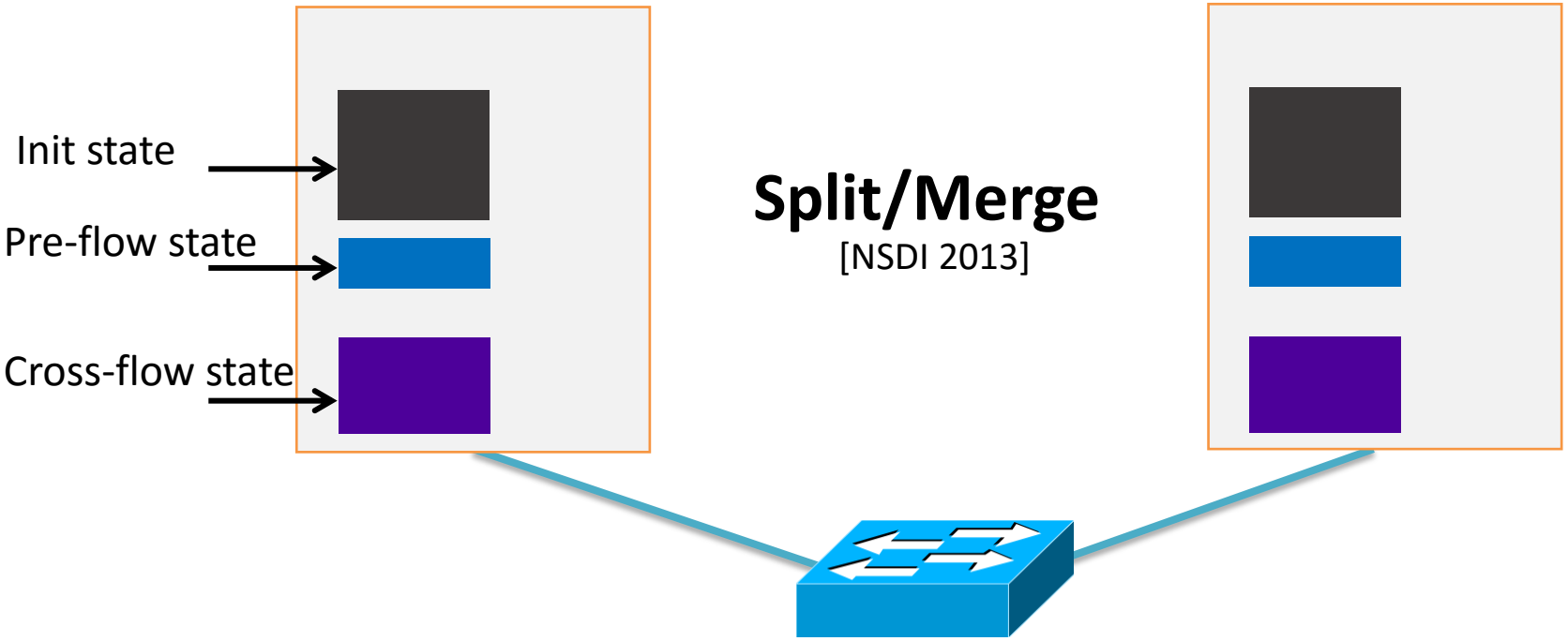
Frameworks for transferring, or sharing live middlebox state

- ***Require modifications*** or ***annotation*** to middlebox code

NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

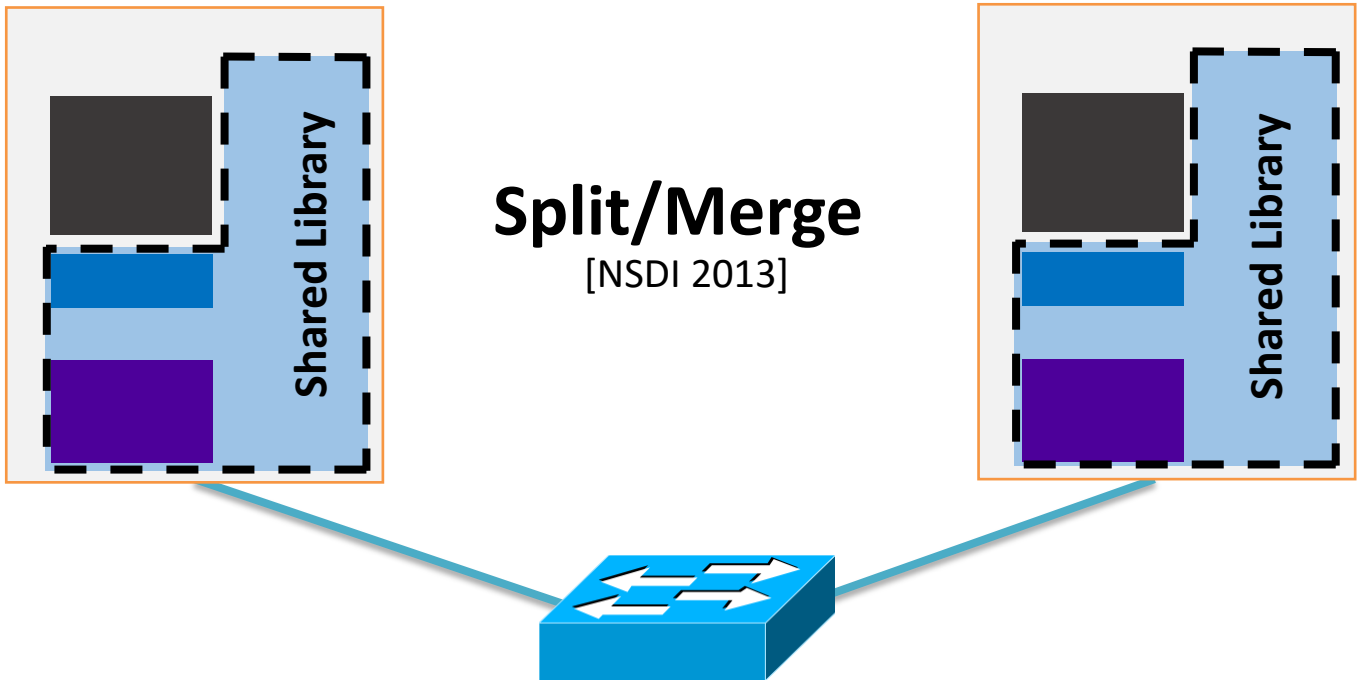
- ***Require modifications*** or ***annotation*** to middlebox code



NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

- ***Require modifications*** or ***annotation*** to middlebox code



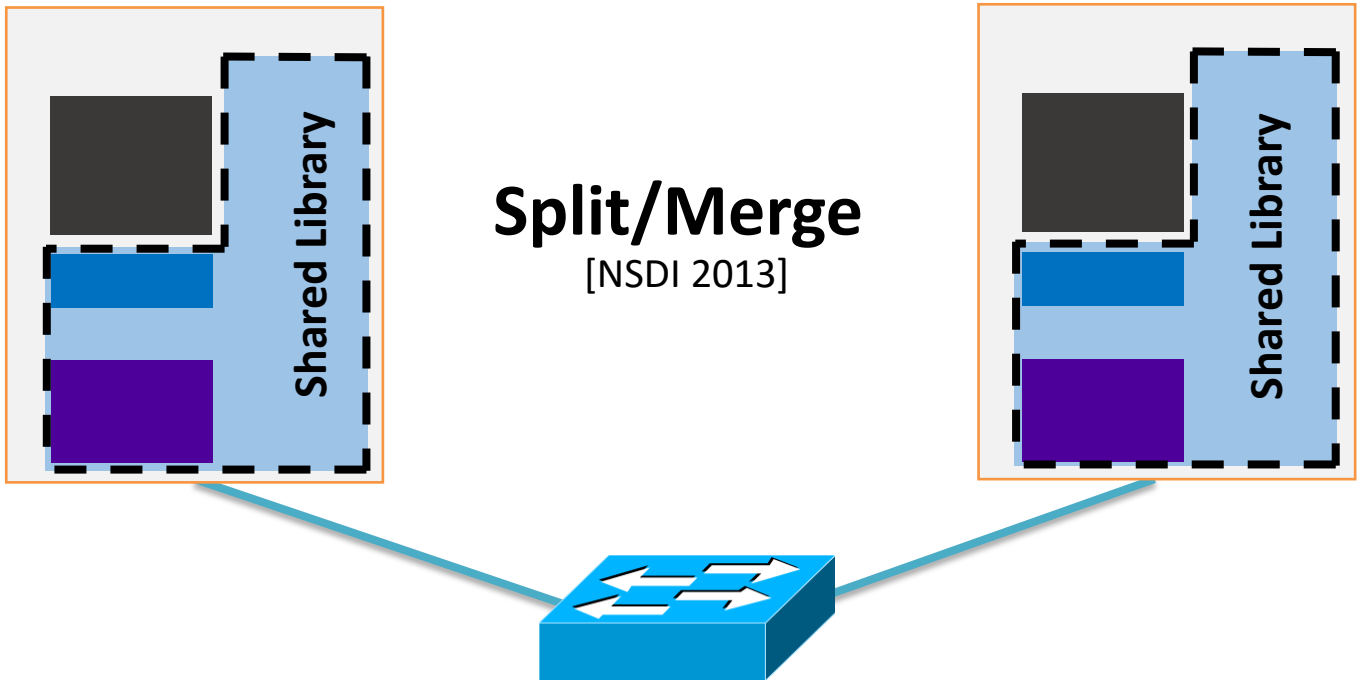
Required modifications:

1. State allocation

NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

- ***Require modifications*** or ***annotation*** to middlebox code



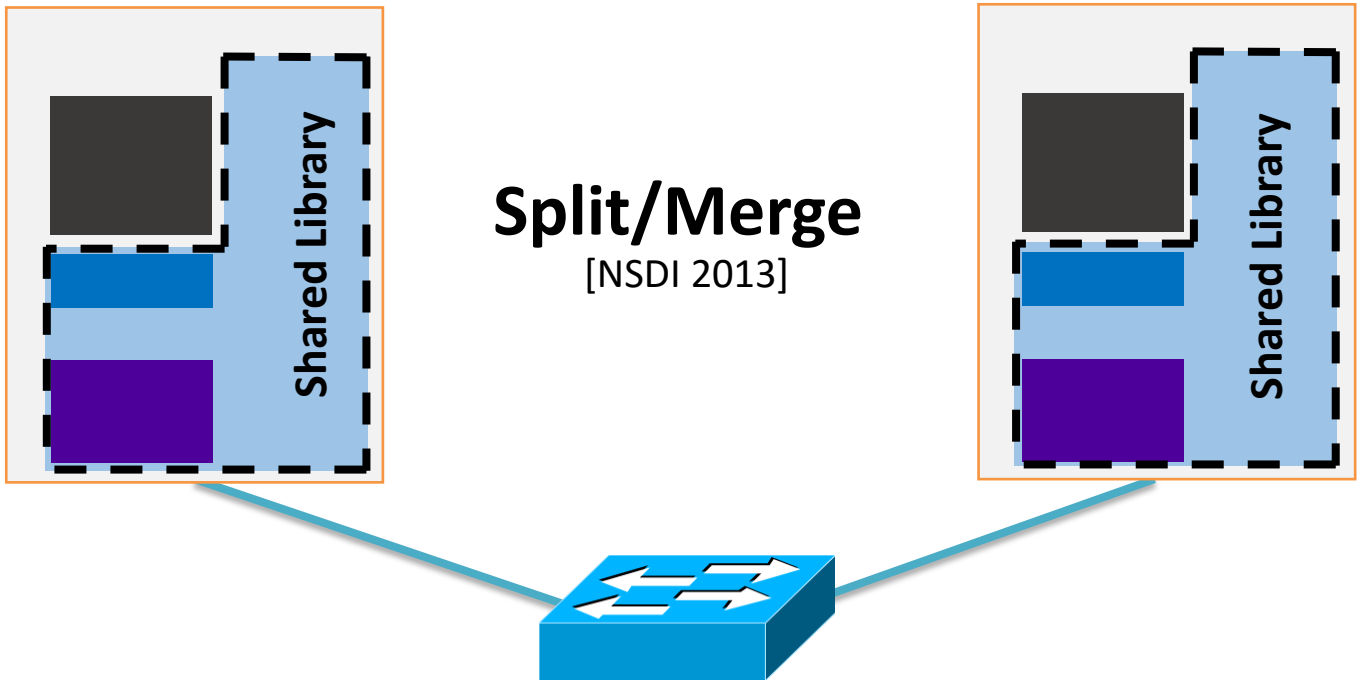
Required modifications:

1. State allocation
2. State access

NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

- ***Require modifications*** or ***annotation*** to middlebox code



Required modifications:

1. State allocation
2. State access
3. State merge

NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

- ***Require modifications*** or ***annotation*** to middlebox code

Framework	State Allocation	State Access	Serialization	Merge State
Split/Merge [NSDI 2013]	✓	✓		✓

NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

- ***Require modifications*** or ***annotation*** to middlebox code

Framework	State Allocation	State Access	Serialization	Merge State
Split/Merge [NSDI 2013]	✓	✓		✓
OpenNF [SIGCOMM 2014]			✓	✓

NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

- ***Require modifications*** or ***annotation*** to middlebox code

Framework	State Allocation	State Access	Serialization	Merge State
Split/Merge [NSDI 2013]	✓	✓		✓
OpenNF [SIGCOMM 2014]			✓	✓
FTMB [SIGCOMM 2015]		✓		

NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

- ***Require modifications*** or ***annotation*** to middlebox code

Framework	State Allocation	State Access	Serialization	Merge State
Split/Merge [NSDI 2013]	✓	✓		✓
OpenNF [SIGCOMM 2014]			✓	✓
FTMB [SIGCOMM 2015]		✓		
Pico Rep. [SoCC 2013]	✓	✓		

NFV state management -> middlebox modification

Frameworks for transferring, or sharing live middlebox state

- ***Require modifications*** or ***annotation*** to middlebox code

Framework	State Allocation	State Access	Serialization	Merge State
Split/Merge [NSDI 2013]	✓	✓		✓
OpenNF [SIGCOMM 2014]			✓	✓
FTMB [SIGCOMM 2015]		✓		
Pico Rep. [SoCC 2013]	✓	✓		
Stateless NF [HotMiddlebox 2015]	✓	✓		

Why is modifying a middlebox hard?

Why is modifying a middlebox hard?

Middleboxes are **complex**

MB	LOC (C/C++)	Classes/ Structs	Level of pointers	Number of Procedures
PRADS	10K	40	4	297
OpenVPN	62K	194	2	2023
HAProxy	63K	191	8	2560
Bro IDS	97K	1798	-	3034
Squid	166K	875	-	2133
Snort IDS	275K	898	10	4617

Why is modifying a middlebox hard?

Middleboxes are **complex**

MB	LOC (C/C++)	Classes/ Structs	Level of pointers	Number of Procedures
PRADS	10K	40	4	297
OpenVPN	62K	194	2	2023
HAProxy	63K	191	8	2560
Bro IDS	97K	1798	-	3034
Squid	166K	875	-	2133
Snort IDS	275K	898	10	4617

Why is modifying a middlebox hard?

Middleboxes are **complex**

MB	LOC (C/C++)	Classes/ Structs	Level of pointers	Number of Procedures
PRADS	10K	40	4	297
OpenVPN	62K	194	2	2023
HAProxy	63K	191	8	2560
Bro IDS	97K	1798	-	3034
Squid	166K	875	-	2133
Snort IDS	275K	898	10	4617

Why is modifying a middlebox hard?

Middleboxes are **complex**, **diverse** and have a **variety of state**

MB	LOC (C/C++)	Classes/ Structs	Level of pointers	Number of Procedures
PRADS	10K	40	4	297
OpenVPN	62K	194	2	2023
HAProxy	63K	191	8	2560
Bro IDS	97K	1798	-	3034
Squid	166K	875	-	2133
Snort IDS	275K	898	10	4617

Why is modifying a middlebox hard?

Middleboxes are **complex**, **diverse** and have a **variety of state**

MB	LOC (C/C++)	Classes/ Structs	Level of pointers	Number of Procedures
PRADS	10K	40	4	297
OpenVPN	62K	194	2	2023
HAProxy	63K	191	8	2560
Bro IDS	97K	1798	-	3034
Squid	166K	875	-	2133
Snort IDS	275K	898	10	4617

Why is modifying a middlebox hard?

Middleboxes are **complex**, **diverse** and have a **variety of state**

MB	LOC (C/C++)	Classes/ Structs	Level of pointers	Number of Procedures
PRADS	10K	40	4	297
OpenVPN	62K	194	2	2023
HAProxy	63K	191	8	2560
Bro IDS	97K	1798	-	3034
Squid	166K	875	-	2133
Snort IDS	275K	898	10	4617

Missing a change to some structure, class or function, ***may violate output equivalence.***

Why is modifying a middlebox hard?

Output equivalence: for any input the aggregate output of a dynamic set of instances should be equivalent to the output produced by single instance.

Middleboxes are **complex, diverse** and have a **variety of state**

	Size (C/C++)	Number of Structs	Level of pointers	Number of Procedures
PRADS	[illegible]	40	[illegible]	[illegible]
OpenVPN	[illegible]	194	[illegible]	[illegible]
HAProxy	63K	191	8	2560
Bro IDS	97K	1798	-	3034
Squid	166K	875	-	2133
Snort IDS	275K	898	10	4617

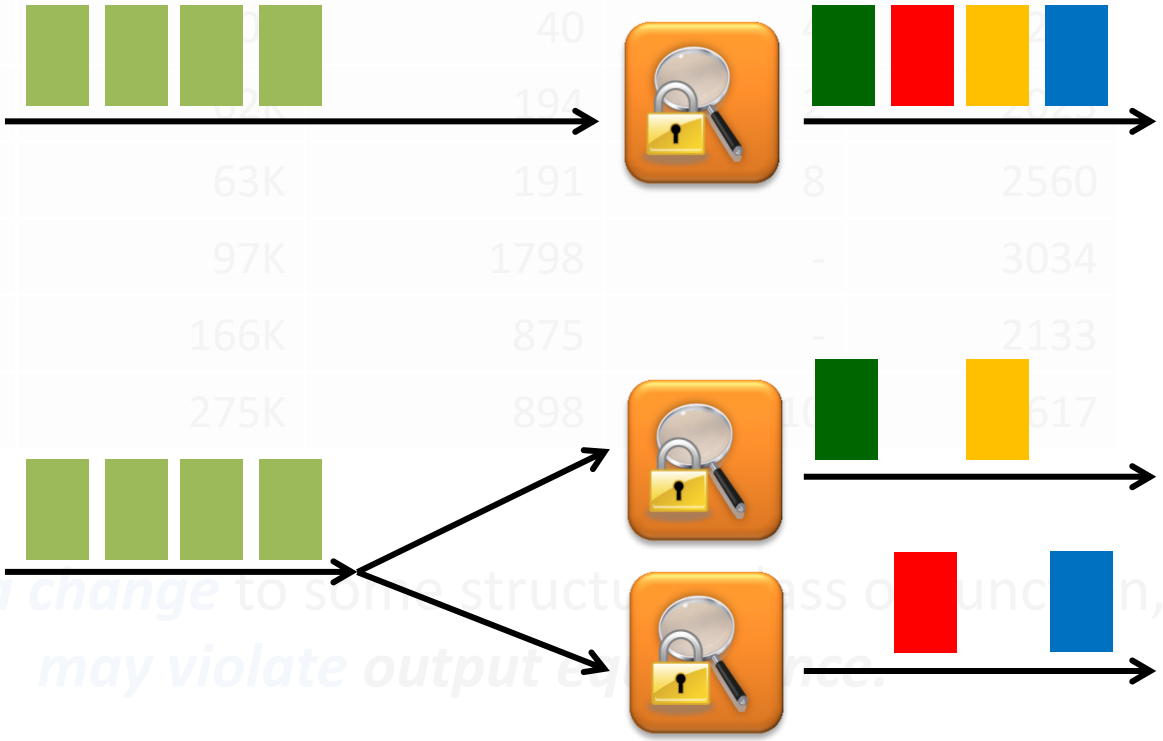
Missing a change to some structure, class or function, ***may violate output equivalence.***

Why is modifying a middlebox hard?

Output equivalence: for any input the aggregate output of a dynamic set of instances should be equivalent to the output produced by single instance.

Middleboxes are **complex, diverse** and have a **variety of state**

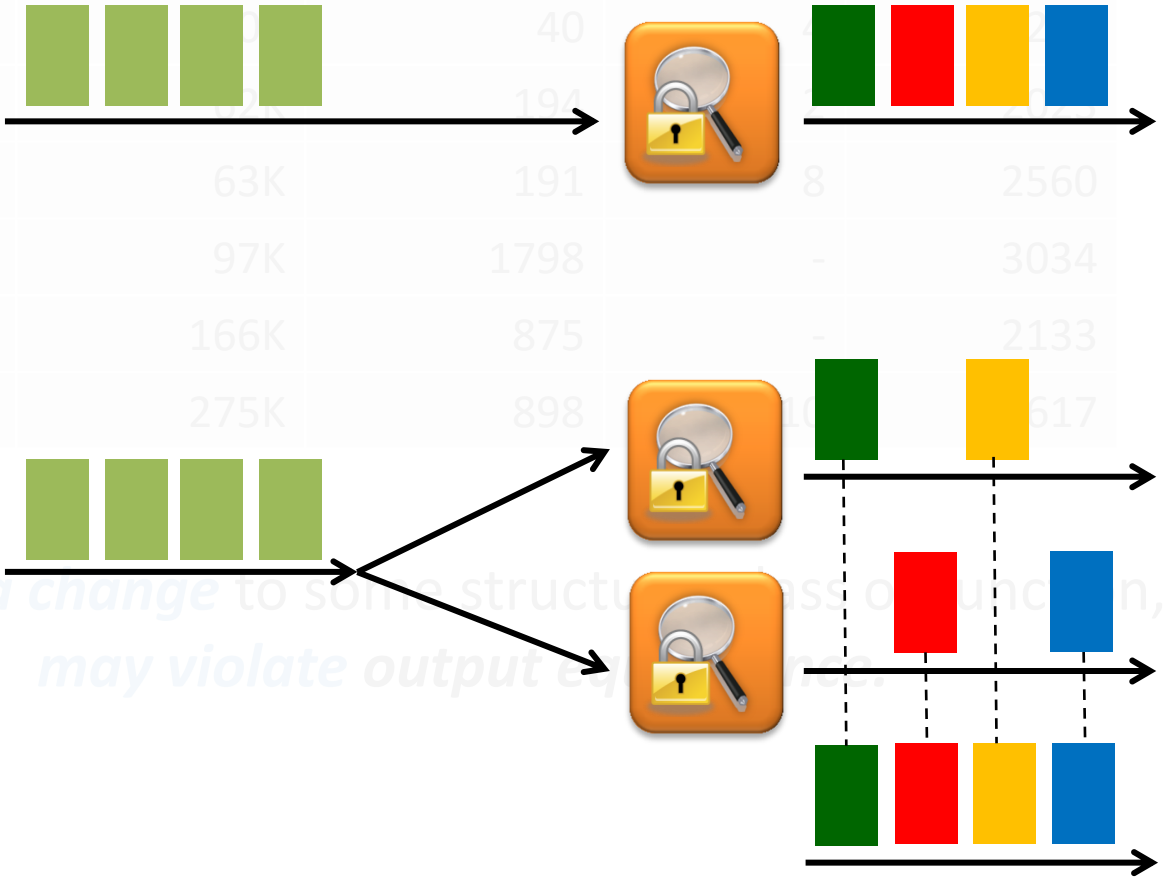
	(C/C++)	Structs	Level of pointers	Number of Procedures
PRADS	[illegible]	40	[illegible]	[illegible]
OpenVPN	[illegible]	194	[illegible]	[illegible]
HAProxy	63K	191	8	2560
Bro IDS	97K	1798	-	3034
Squid	166K	875	-	2133
Snort IDS	275K	898	10	617



Missing a change to some structure, class or function, ***may violate output equivalence.***

Why is modifying a middlebox hard?

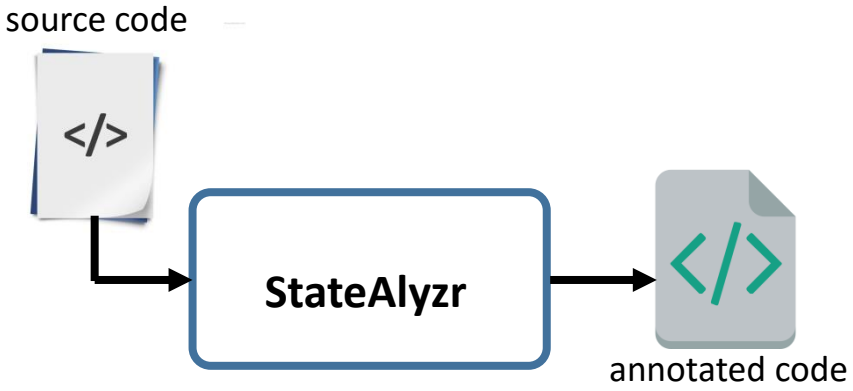
Output equivalence: for any input the aggregate output of a dynamic set of instances should be equivalent to the output produced by single instance.



StateAlyzr: program analysis to the rescue

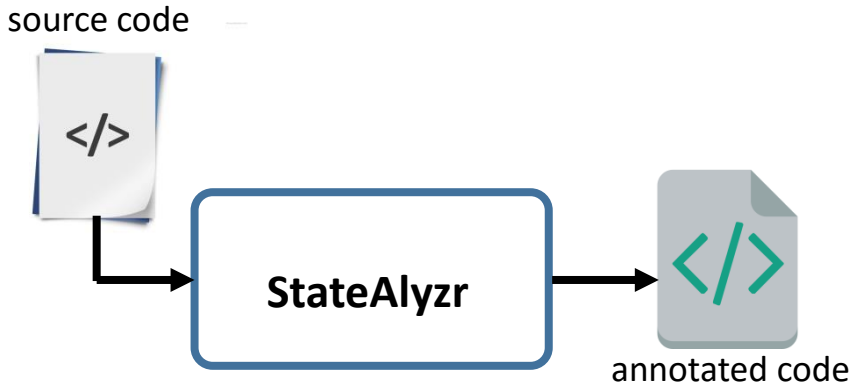
StateAlyzr: program analysis to the rescue

A system that relies on *data* and *control-flow analysis* to *automatically identify* state objects that need explicit handling



StateAlyzr: program analysis to the rescue

A system that relies on *data* and *control-flow analysis* to *automatically identify* state objects that need explicit handling

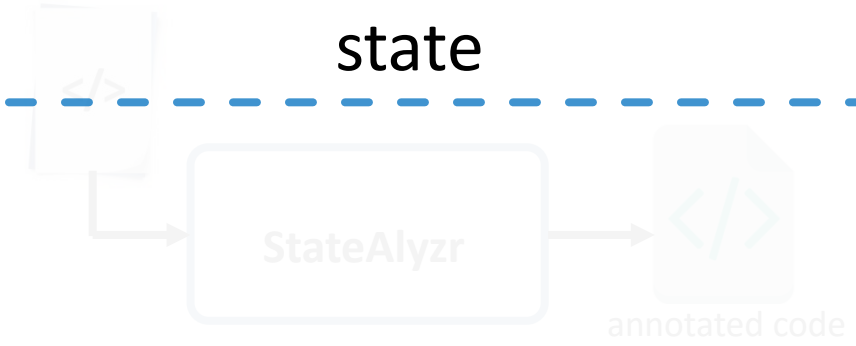


Leverage middlebox code structure to **improve precision** without **compromising soundness**

StateAlyzr: program analysis to the rescue

A system that relies on *data* and *control-flow analysis* to *automatically identify* state objects that need explicit handling

Soundness means that the system ***must not miss any critical*** types, storage locations, allocations, or uses of state

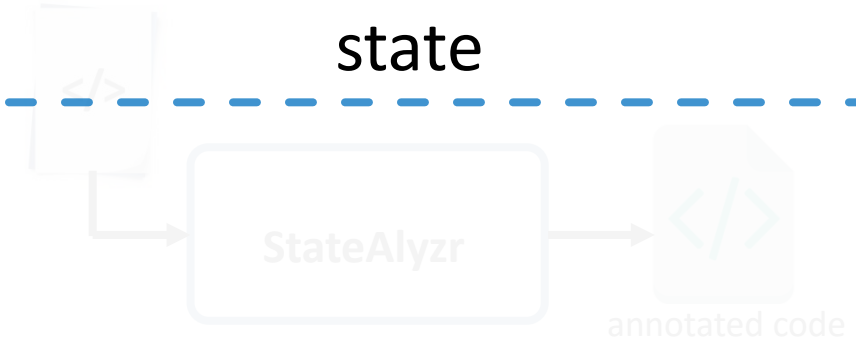


Leverage middlebox code structure to **improve precision** without **compromising soundness**

StateAlyzr: program analysis to the rescue

A system that relies on *data* and *control-flow analysis* to *automatically identify* state objects that need explicit handling

Soundness means that the system ***must not miss any critical*** types, storage locations, allocations, or uses of state



Leverage middlebox code structure to improve precision without compromising soundness

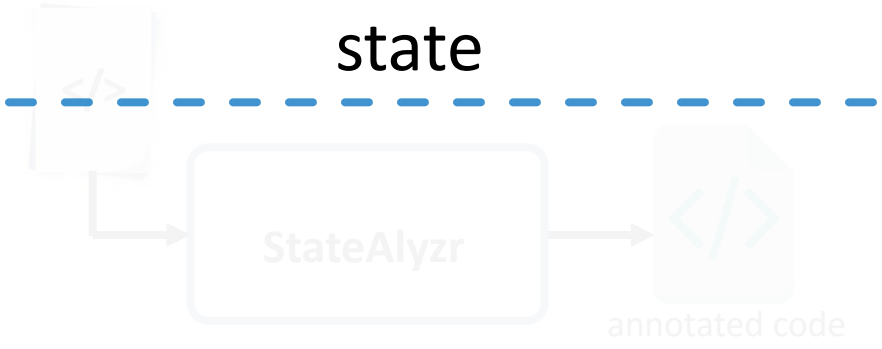
Precision means that the system identifies ***the minimal set of state*** that requires special handling.

StateAlyzr: program analysis to the rescue

required for **output equivalence**

A system that relies on *data* and *control-flow analysis* to *automatically identify* state objects that need explicit handling

Soundness means that the system ***must not miss any critical*** types, storage locations, allocations, or uses of state



Leverage middlebox code structure to improve precision without compromising soundness

Precision means that the system identifies ***the minimal set of state*** that requires special handling.

StateAlyzr: program analysis to the rescue

required for **output equivalence**

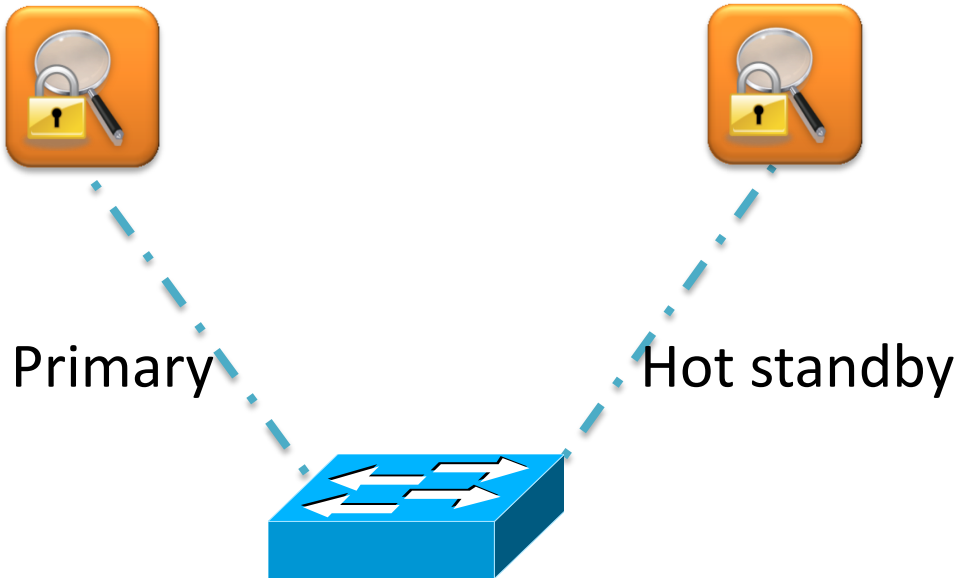
Soundness means that the system ***must not miss any critical*** types, storage locations, allocations, or uses of state

required for **performant state transfers**

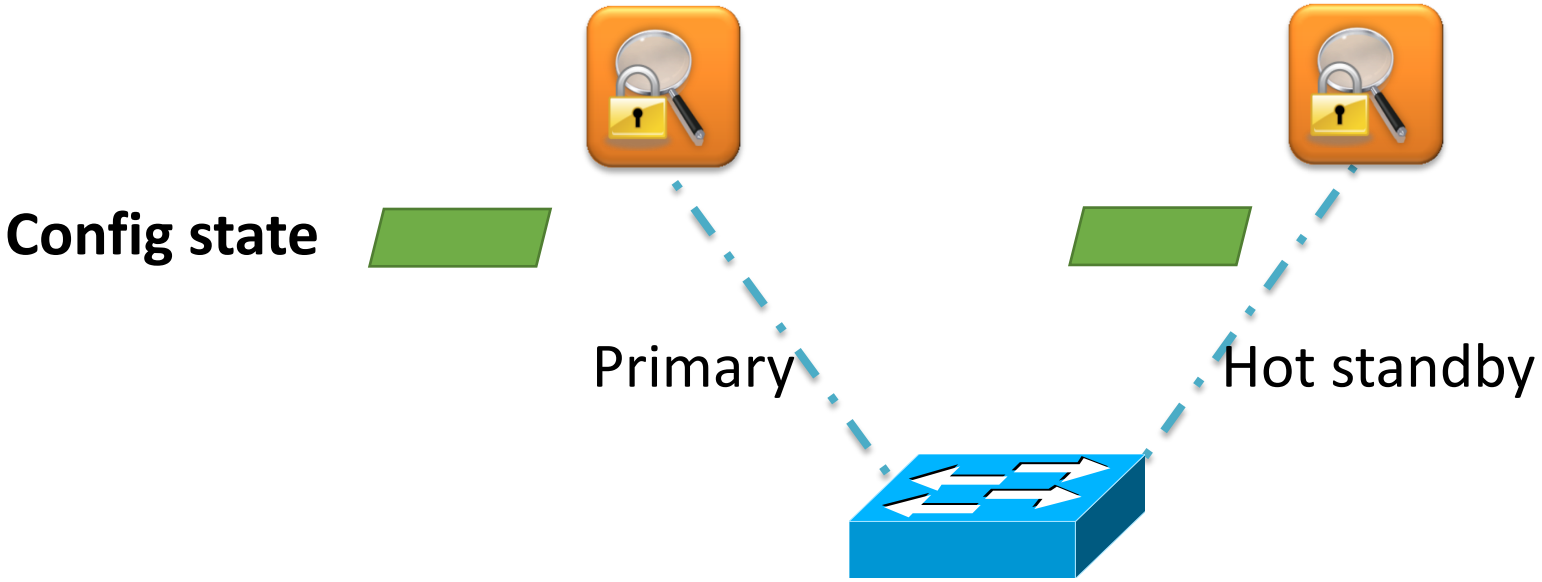
Precision means that the system identifies ***the minimal set of state*** that requires special handling.

Fault tolerance IDS

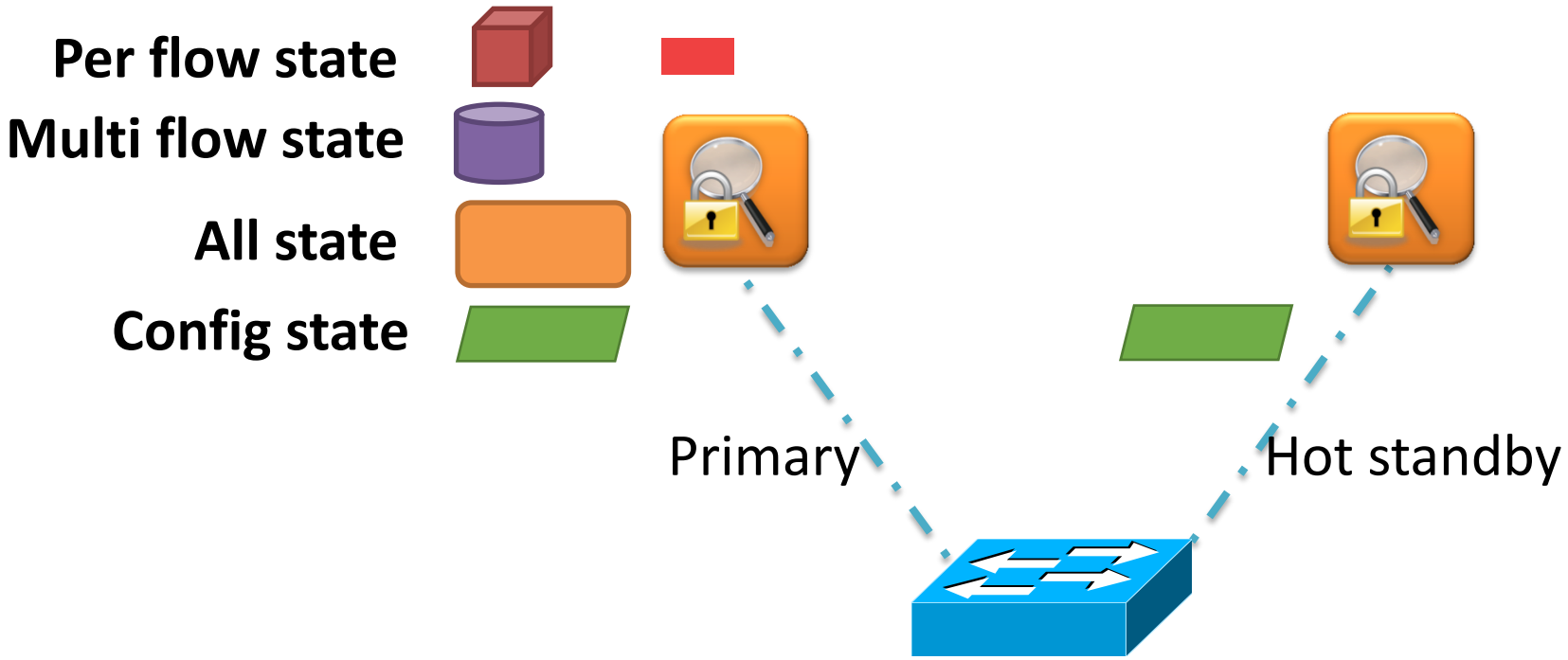
Fault tolerance IDS



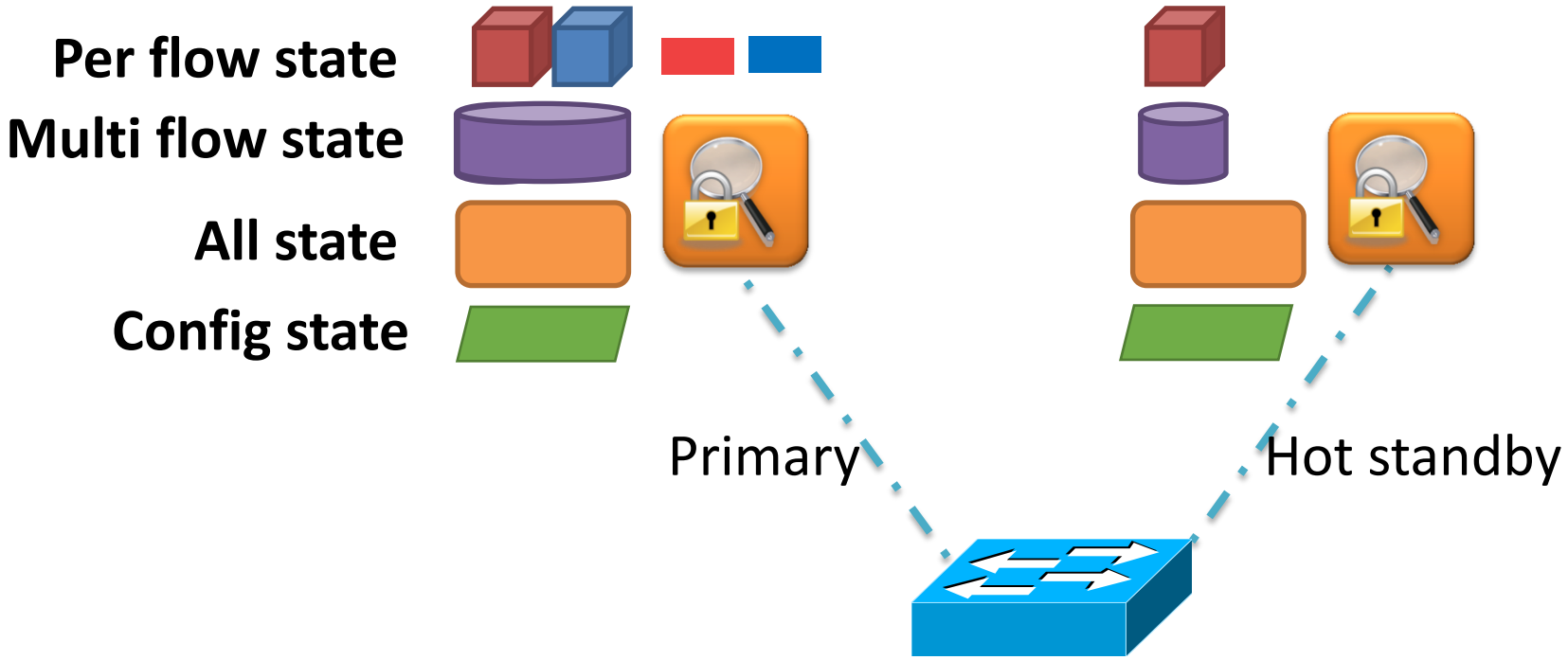
Fault tolerance IDS



Fault tolerance IDS

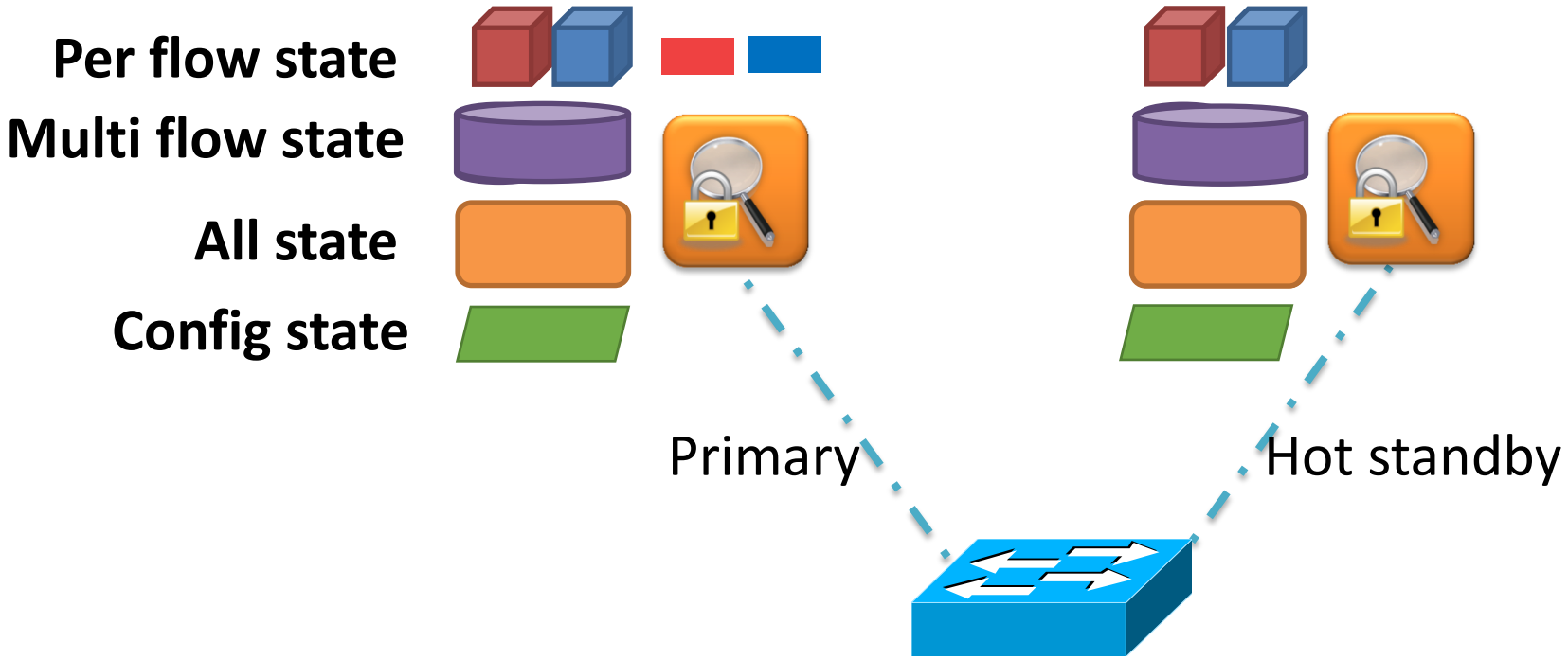


Fault tolerance IDS



The ***primary*** sends a copy of the ***updated*** state to the ***hot standby*** after each packet

Fault tolerance IDS



The ***primary*** sends a copy of the *updated* state to the ***hot standby*** after each packet

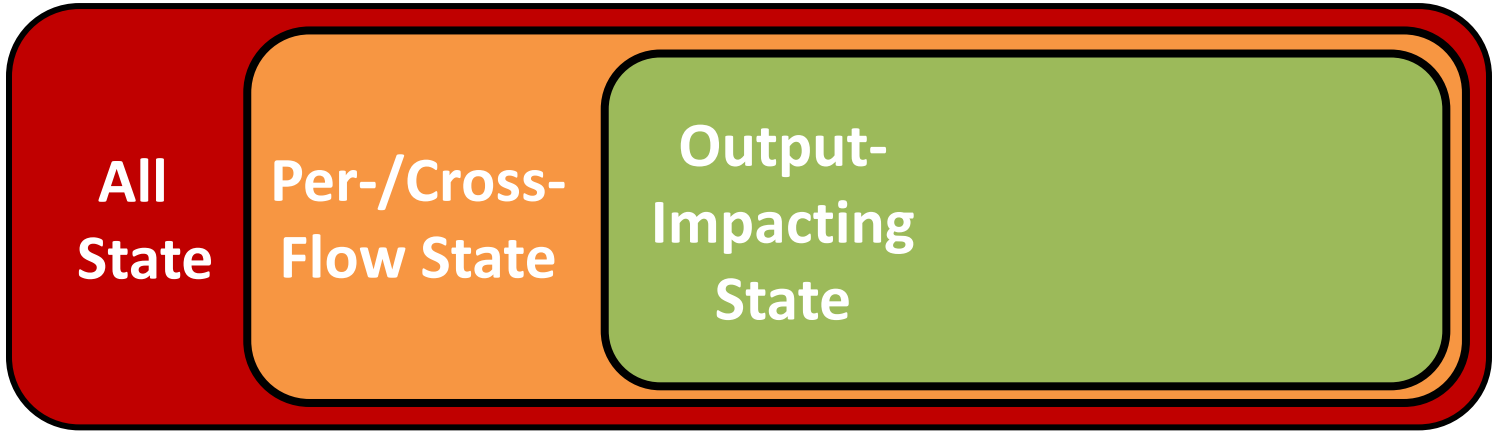
StateAlyzr

StateAlyzr

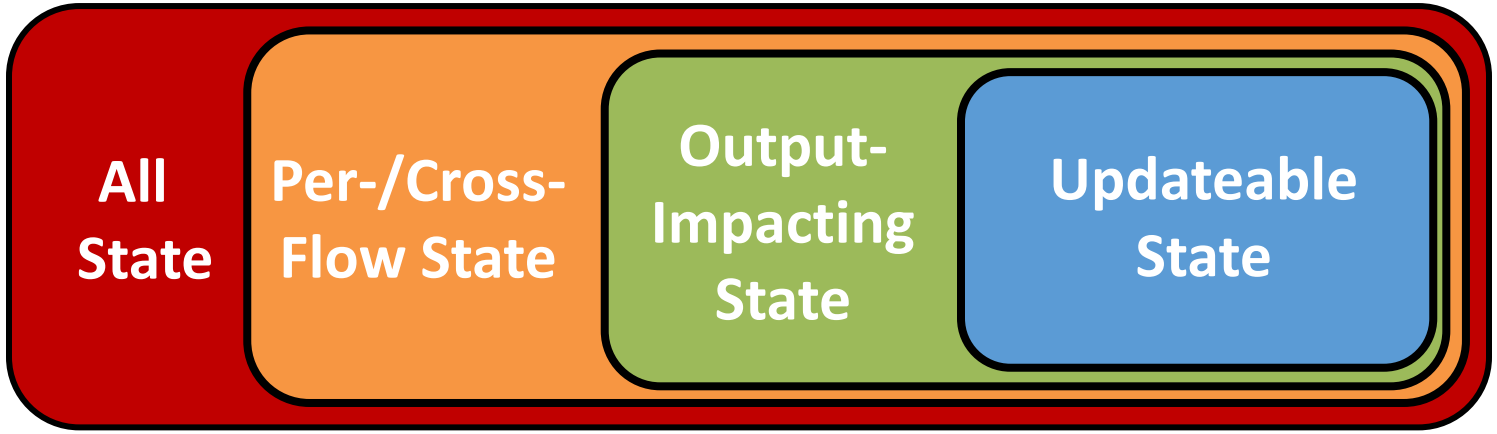
All
State

StateAlyzr

StateAlyzr



StateAlyzr

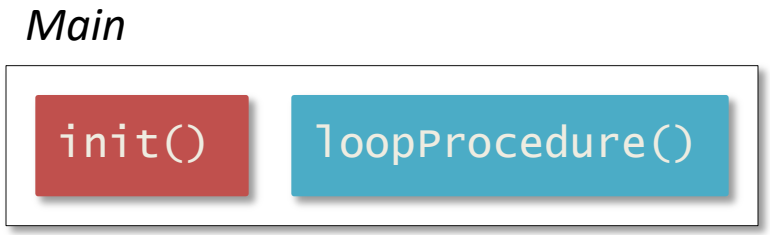

All State
Per-/Cross-Flow State
Output-Impacting State
Updateable State

Logical structure of middlebox code

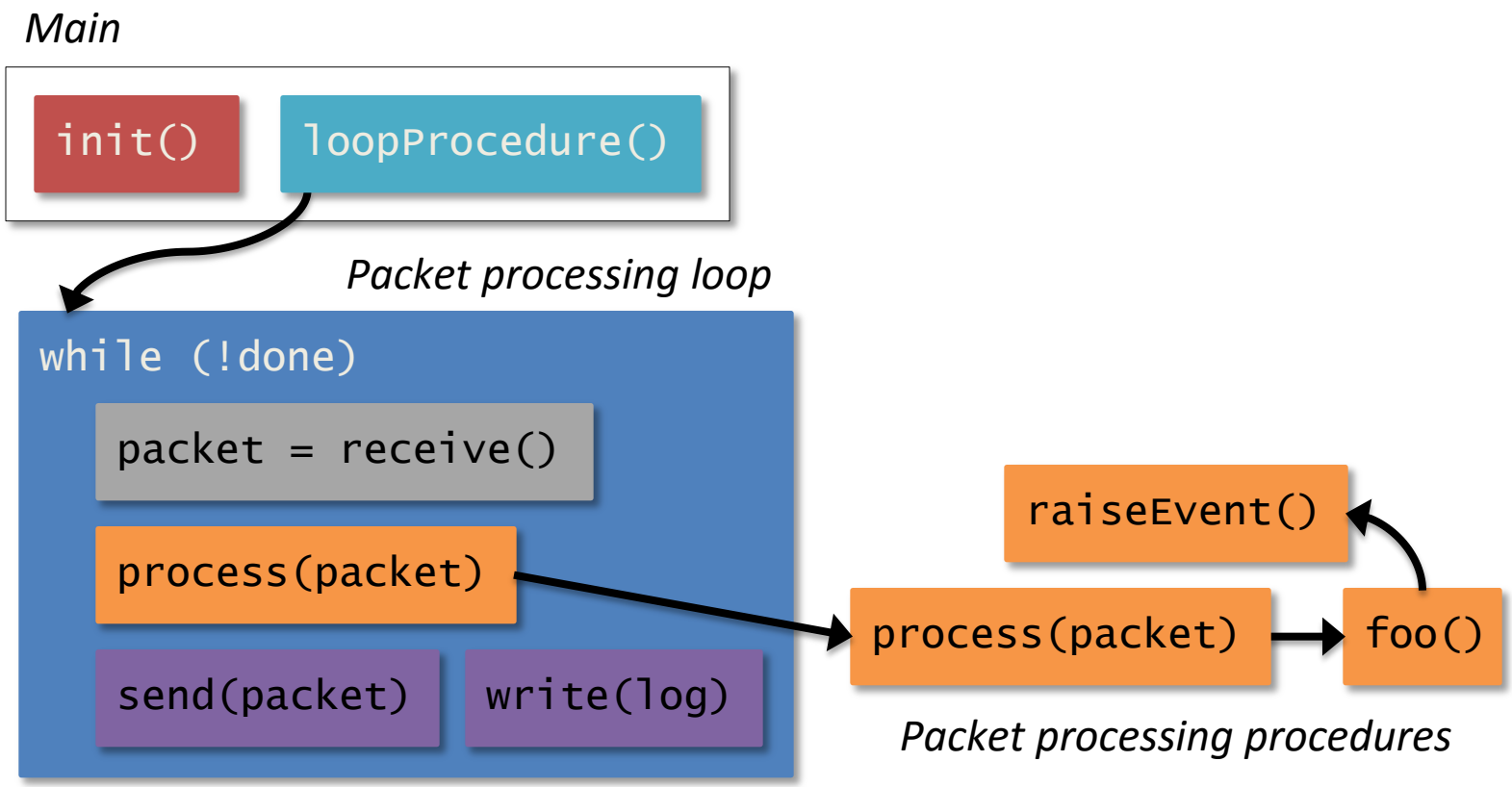
Logical structure of middlebox code

Main

Logical structure of middlebox code



Logical structure of middlebox code



1. Per-/cross-flow state identification

1. Per-/cross-flow state identification

Variables corresponding to per-/cross-flow state must be ***persistent***

1. Per-/cross-flow state identification

Variables corresponding to per-/cross-flow state must be ***persistent***

Persistent state can be stored in

1. Global variables

1. Per-/cross-flow state identification

Variables corresponding to per-/cross-flow state must be ***persistent***

Persistent state can be stored in

1. Global variables
2. Static variables

1. Per-/cross-flow state identification

Variables corresponding to per-/cross-flow state must be ***persistent***

Persistent state can be stored in

1. Global variables
2. Static variables
3. Local variables declared in loop proc.

```
int loopProcedure(int *threshold) {
 int count = 0;
 while(1) {
  struct pcap_pkthdr pcapHdr;
  char *pkt = pcap_next(extPcap, &pcapHdr);
  .
  .
```

1. Per-/cross-flow state identification

Variables corresponding to per-/cross-flow state must be ***persistent***

Persistent state can be stored in

1. Global variables
2. Static variables
3. Local variables declared in loop proc.
4. Formal Params of loop proc.

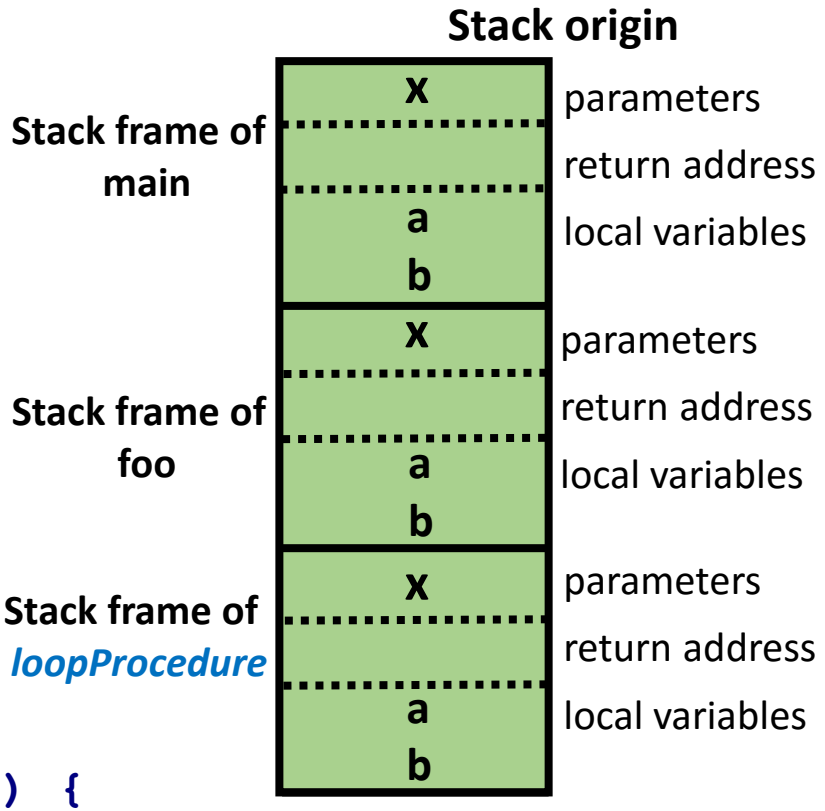
```
int loopProcedure(int *threshold) {
 int count = 0;
 while(1) {
  struct pcap_pkthdr pcapHdr;
  char *pkt = pcap_next(extPcap, &pcapHdr);
  .
  .
```

1. Per-/cross-flow state identification

Variables corresponding to per-/cross-flow state must be ***persistent***

Persistent state can be stored in

1. Global variables
2. Static variables
3. Local variables declared in loop proc.
4. Formal Params of loop proc.



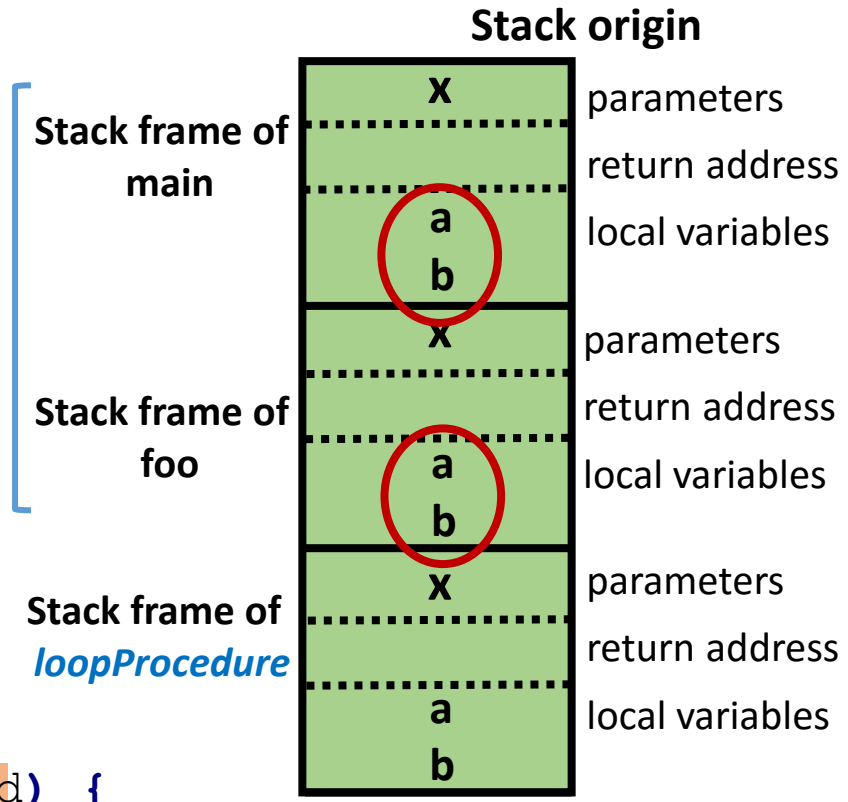

```
int loopProcedure(int *threshold) {
 int count = 0;
 while(1) {
  struct pcap_pkthdr pcapHdr;
  char *pkt = pcap_next(extPcap, &pcapHdr);
  .
  .
```

1. Per-/cross-flow state identification

Variables corresponding to per-/cross-flow state must be ***persistent***

Persistent state can be stored in

1. Global variables
2. Static variables
3. Local variables declared in loop proc.
4. Formal Params of loop proc.



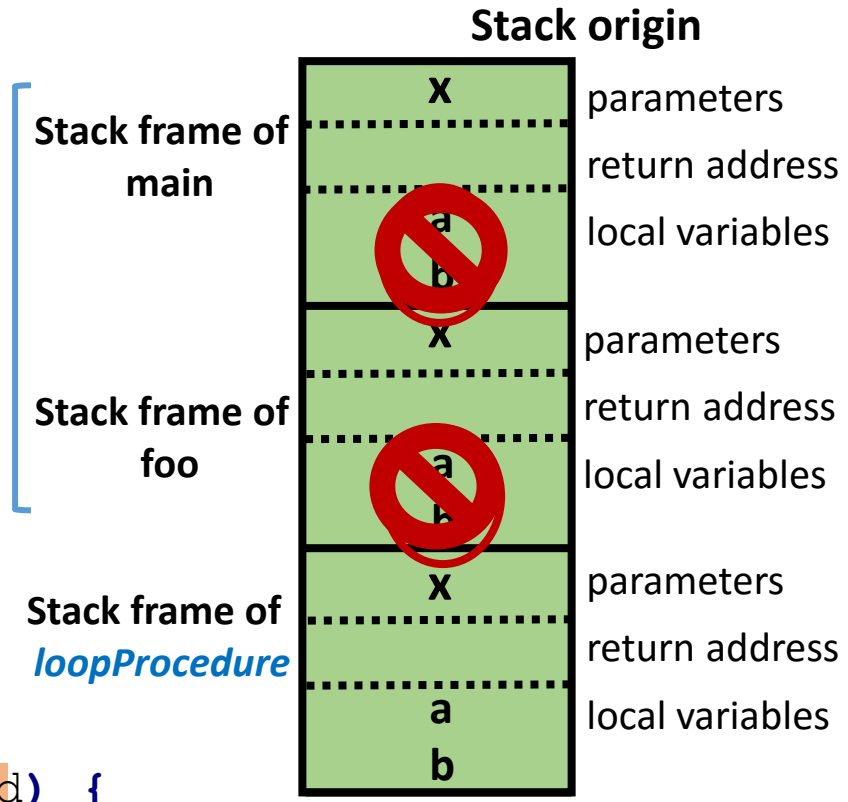

```
int loopProcedure(int *threshold) {
 int count = 0;
 while(1) {
  struct pcap_pkthdr pcapHdr;
  char *pkt = pcap_next(extPcap, &pcapHdr);
  .
  .
```

1. Per-/cross-flow state identification

Variables corresponding to per-/cross-flow state must be ***persistent***

Persistent state can be stored in

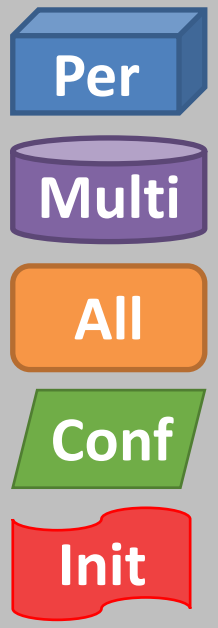
1. Global variables
2. Static variables
3. Local variables declared in loop proc.
4. Formal Params of loop proc.



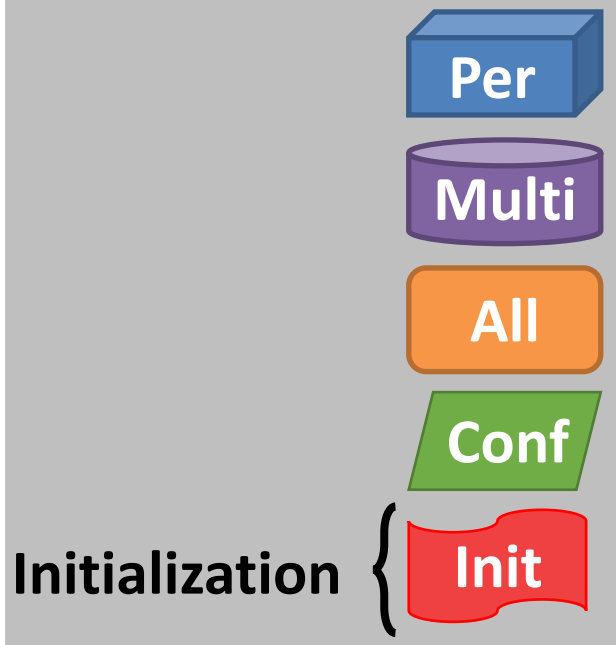

```
int loopProcedure(int *threshold) {
 int count = 0;
 while(1) {
  struct pcap_pkthdr pcapHdr;
  char *pkt = pcap_next(extPcap, &pcapHdr);
  .
  .
```

1. Per-/cross-flow state identification

1. Per-/cross-flow state identification

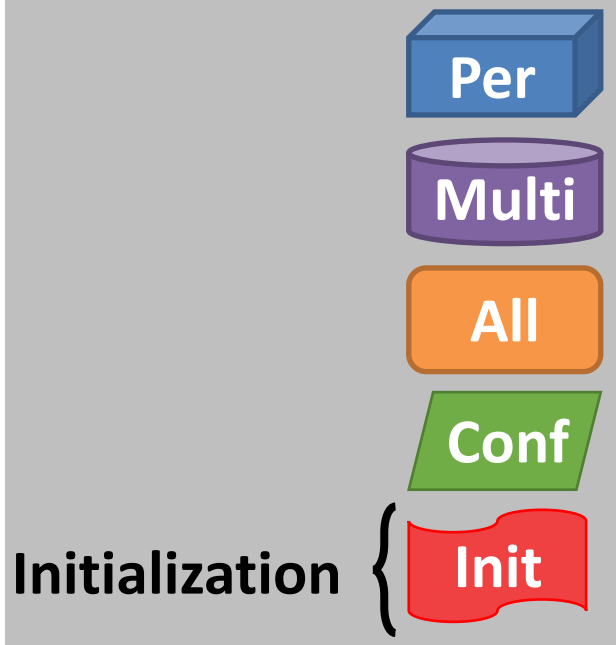


1. Per-/cross-flow state identification



1. Per-/cross-flow state identification

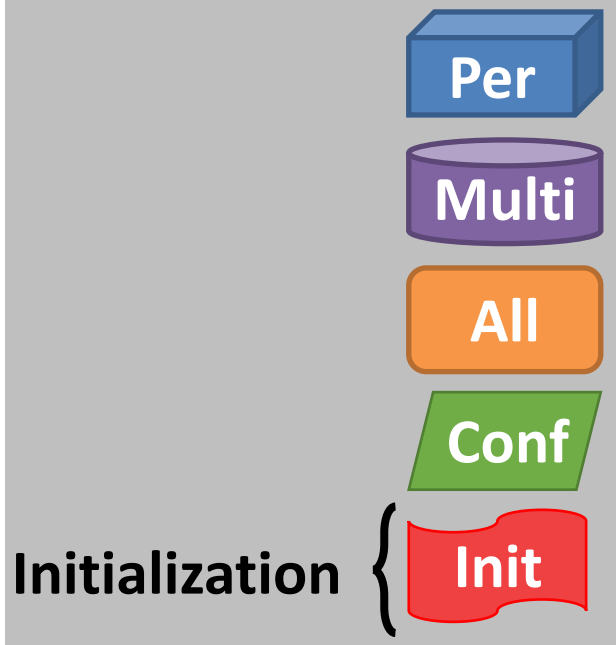
Improve precision by considering variables which are ***used*** in ***packet processing code***



How to identify packet processing code?

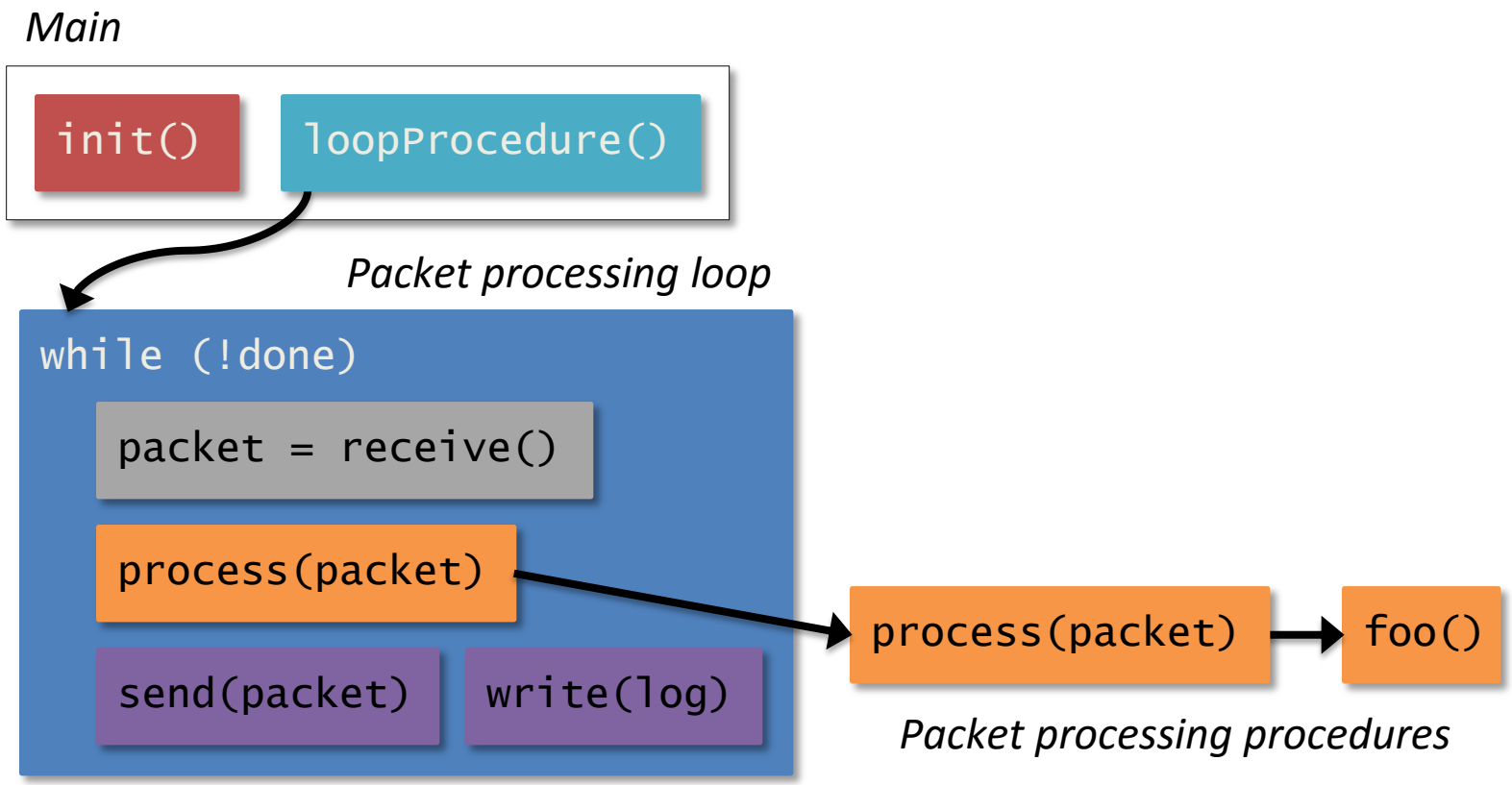
1. Per-/cross-flow state identification

Improve precision by considering variables which are ***used*** in ***packet processing code***



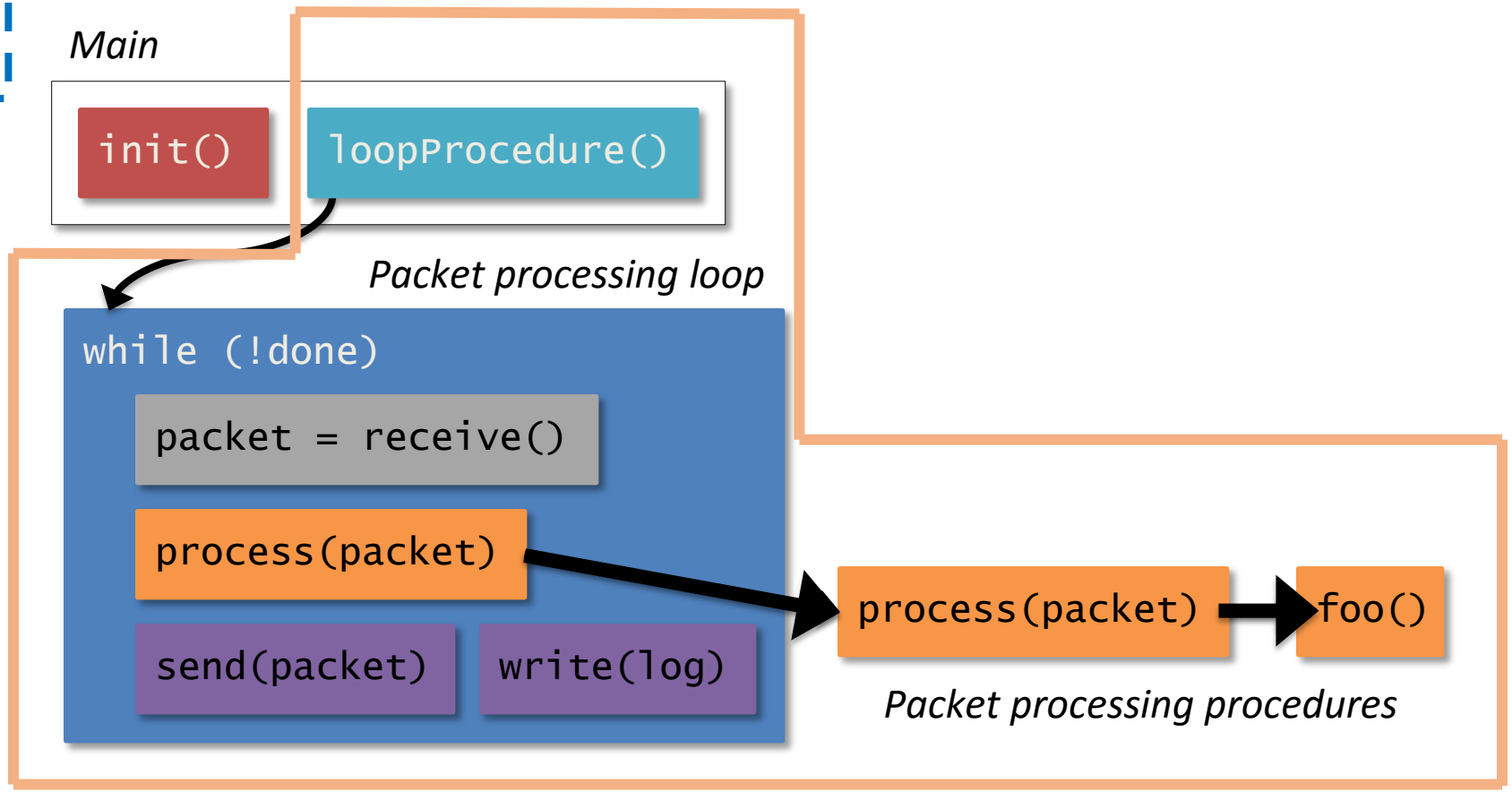
1. Per-/cross-flow state identification

How to identify packet processing code?



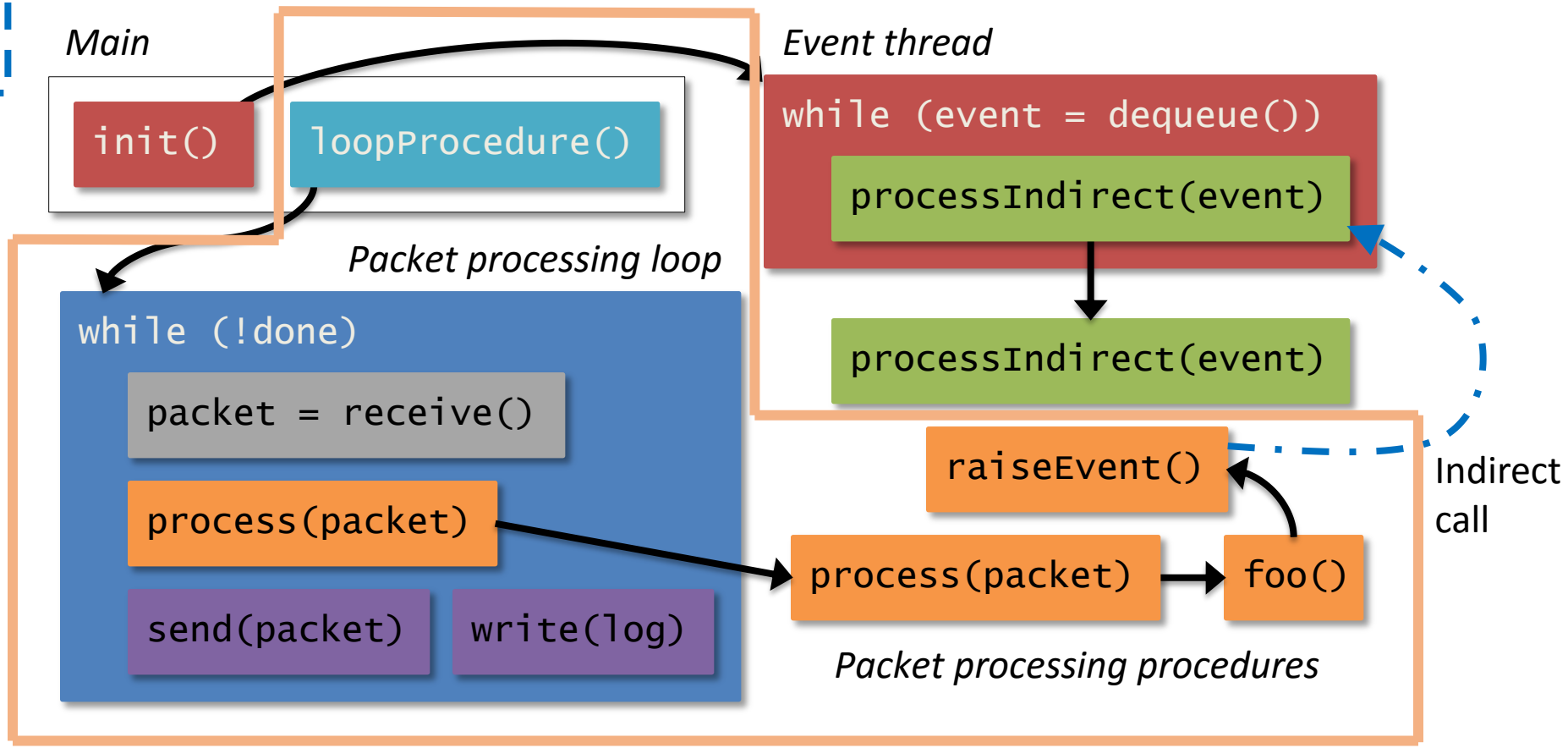
1. Per-/cross-flow state identification

How to identify packet processing code?



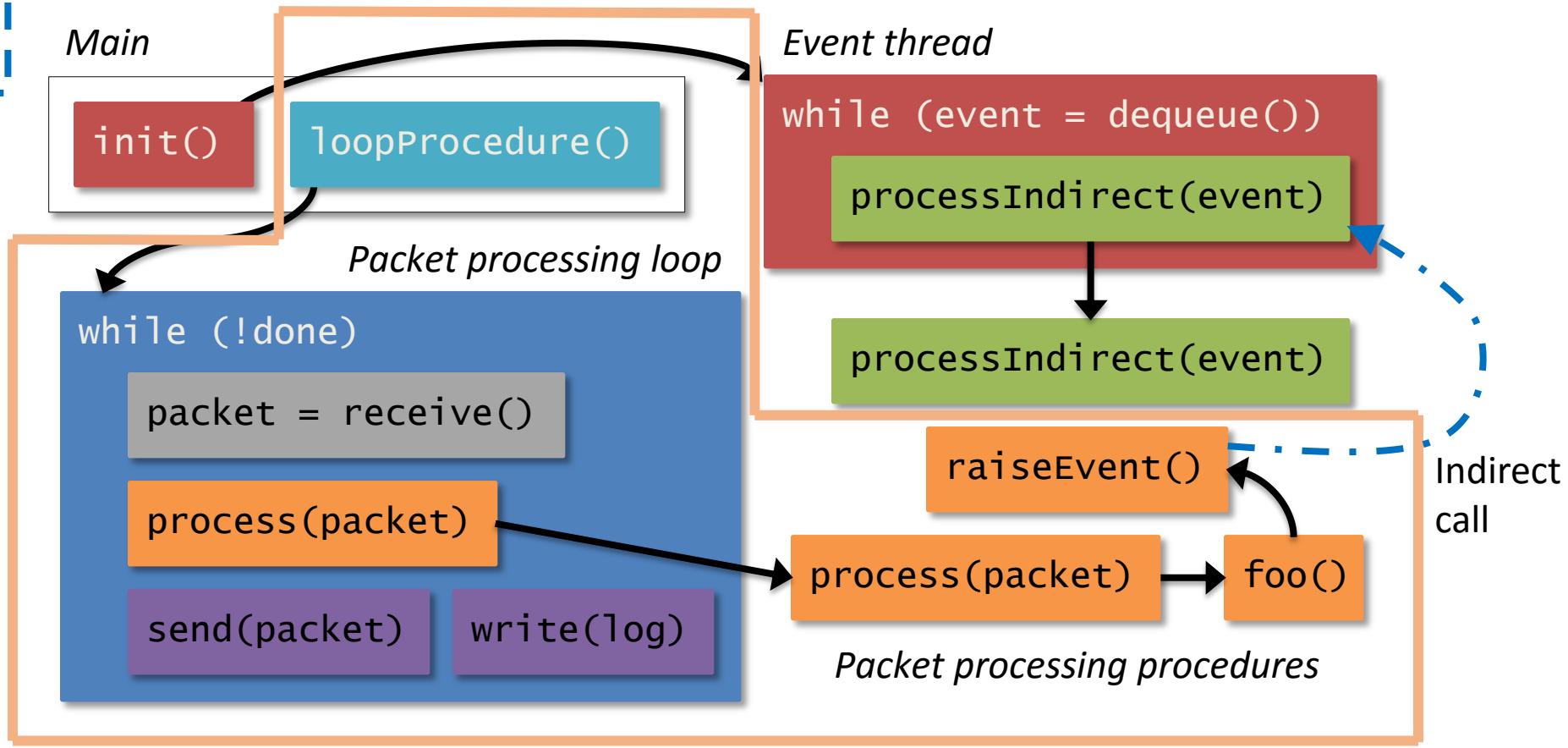
1. Per-/cross-flow state identification

How to identify packet processing code?



1. Per-/cross-flow state identification

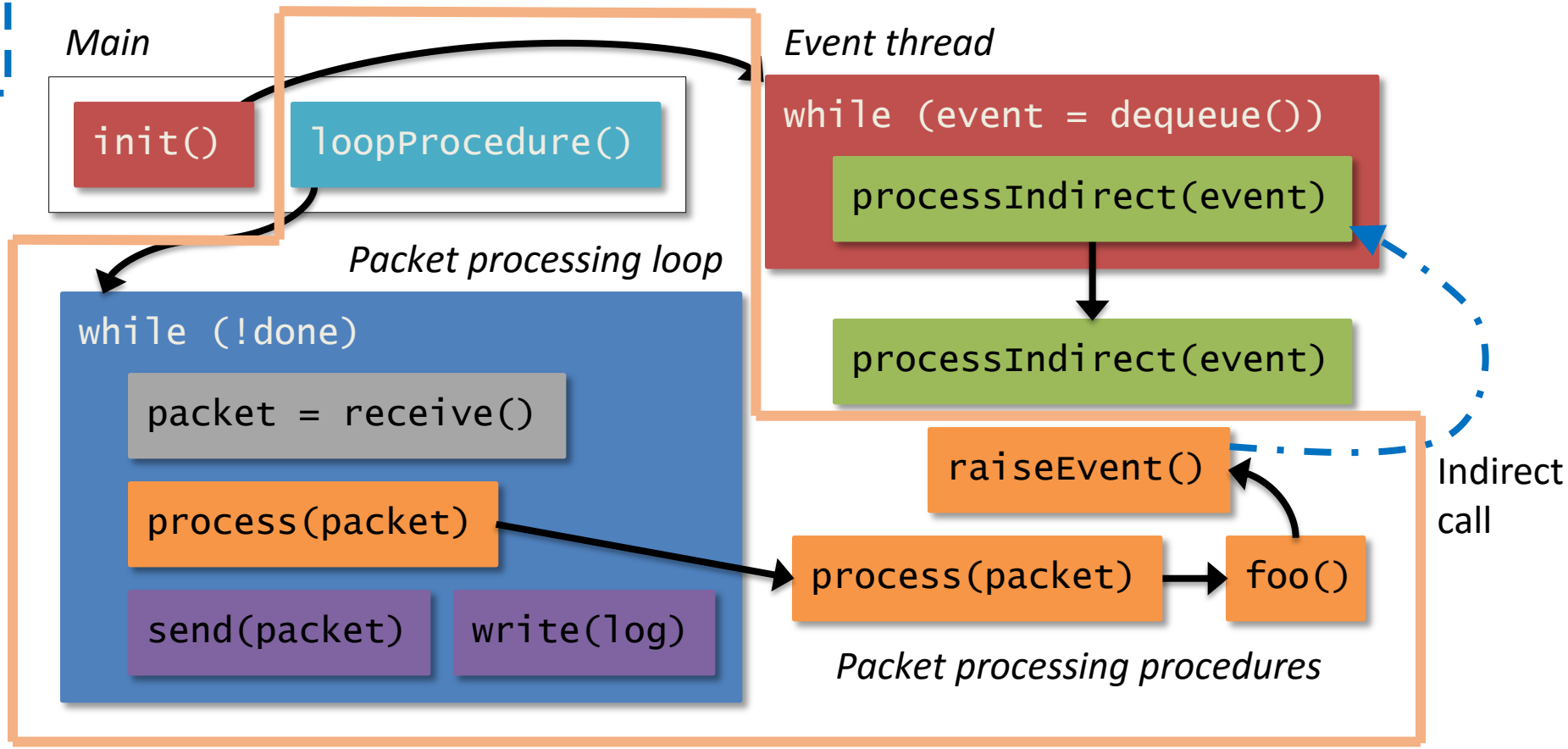
How to identify packet processing code?




```
struct pktHdr *pkt = recv(extPcap);
src_ip = pkt->ip_src_addr;
packet_count ++;
index = src_ip + offset
```

1. Per-/cross-flow state identification

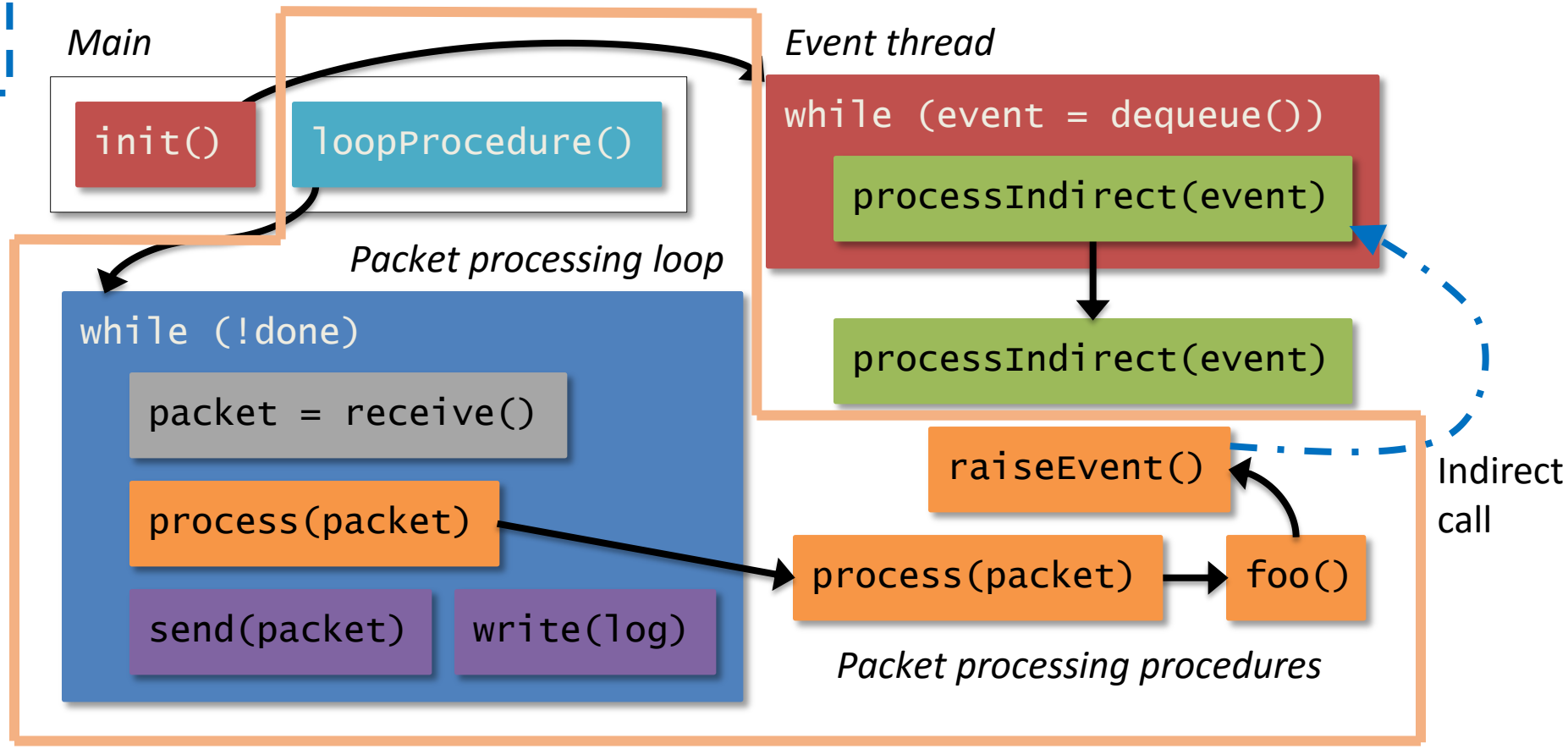
How to identify packet processing code?




```
struct pktHdr *pkt = recv(extPcap);
src_ip = pkt->ip_src_addr;
packet_count ++;
index = src_ip + offset
```

1. Per-/cross-flow state identification

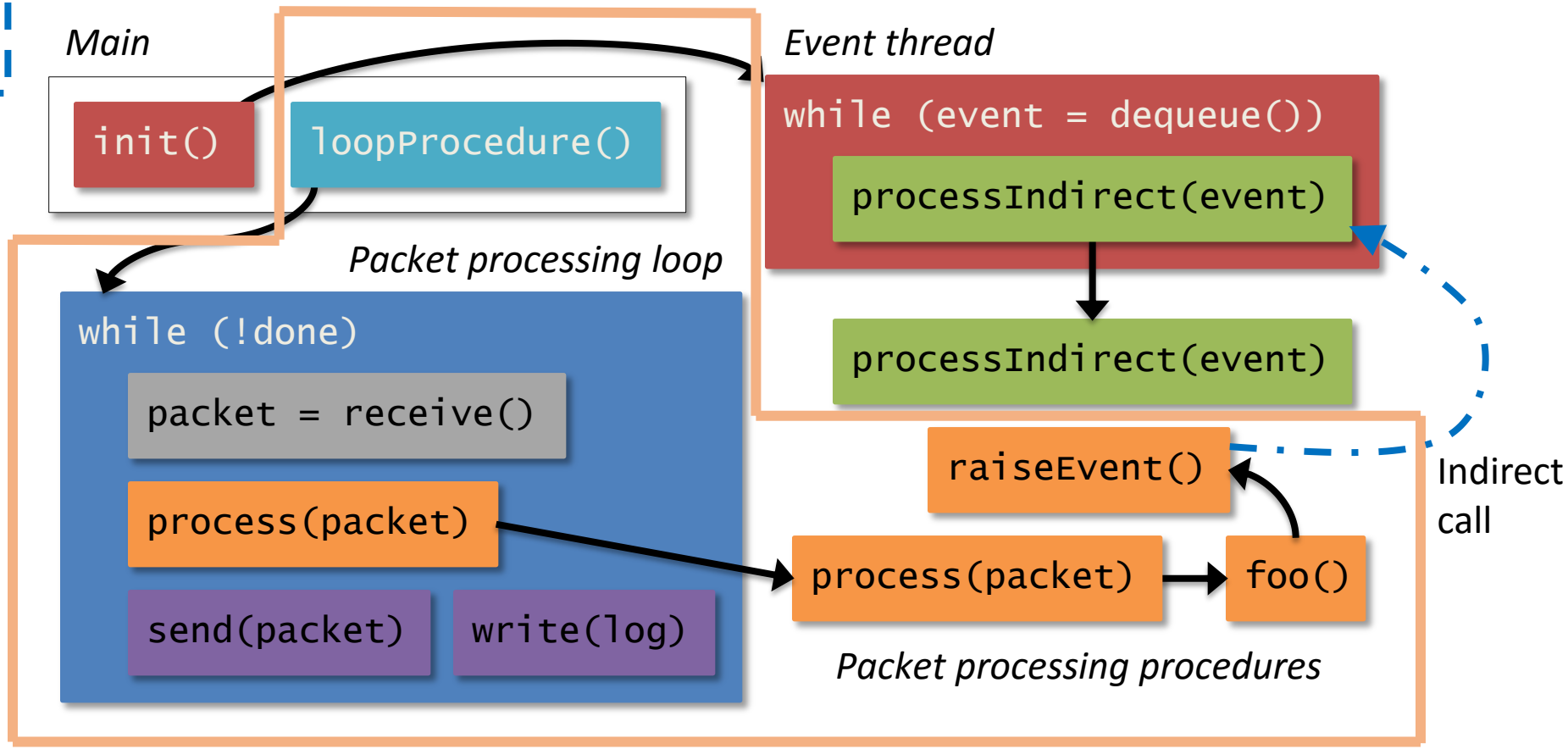
How to identify packet processing code?




```
struct pktHdr *pkt = recv(extPcap);
src_ip = pkt->ip_src_addr;
packet_count ++;
index = src_ip + offset
```

1. Per-/cross-flow state identification

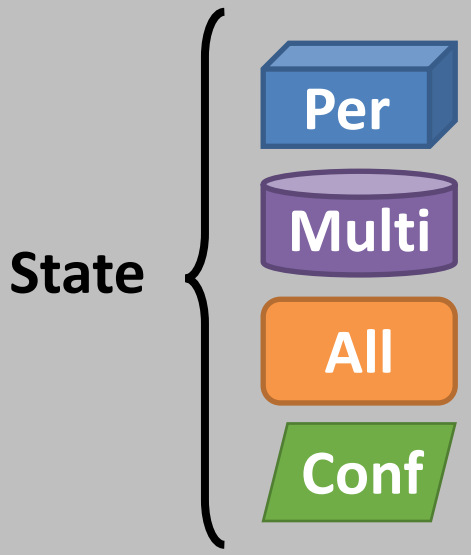
How to identify packet processing code?



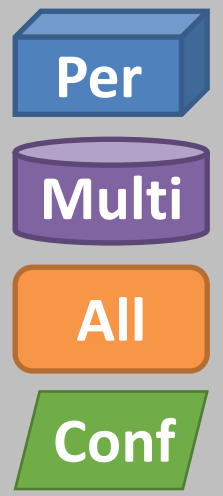
Computes a ***forward slice*** from packet recv function. Any procedure appearing ***in the slice*** is considered as ***packet processing procedure***.

2. Identify updateable state

2. Identify updateable state



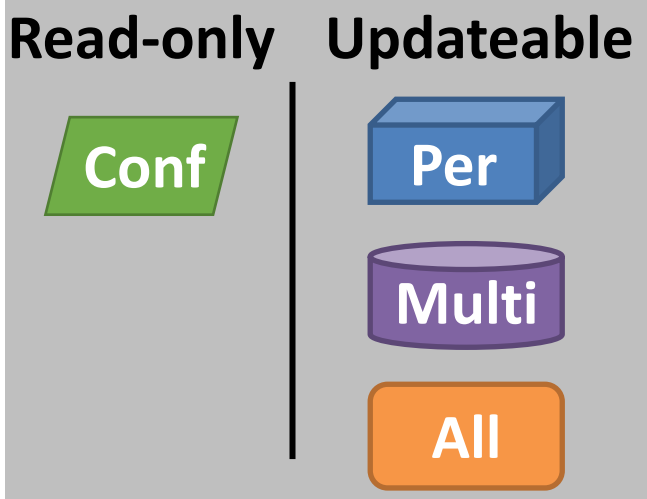
2. Identify updateable state



Read-only | Updateable

2. Identify updateable state

Whether the state is updated while processing the packet ?



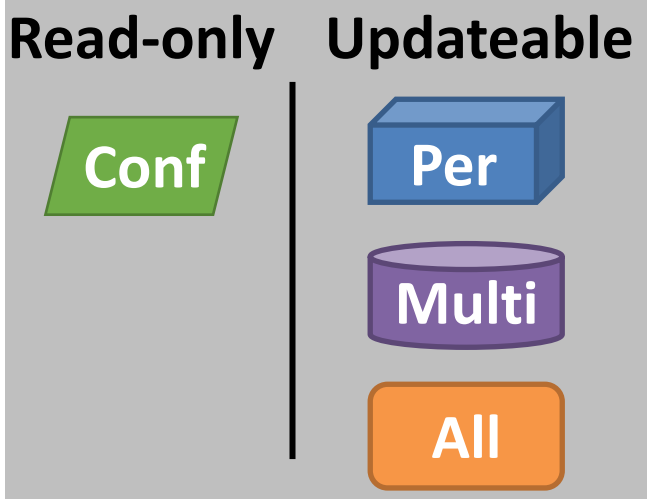
2. Identify updateable state

Whether the state is updated while processing the packet ?

- Strawman approach
  - Identify top-level variable on the left-hand-side(LHS) of assignment statement

per-/cross-flow var

```
in_port = pkt.src_port;
```

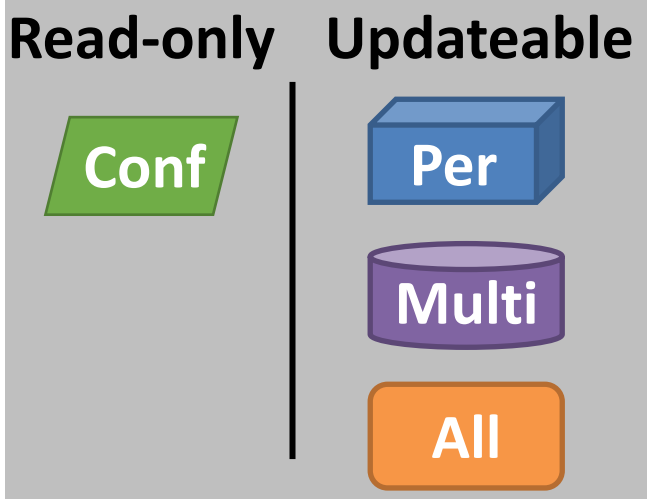


2. Identify updateable state

Whether the state is updated while processing the packet ?

- Strawman approach
  - Identify top-level variable on the left-hand-side(LHS) of assignment statement

Falls short due to *aliasing*



2. Identify updateable state

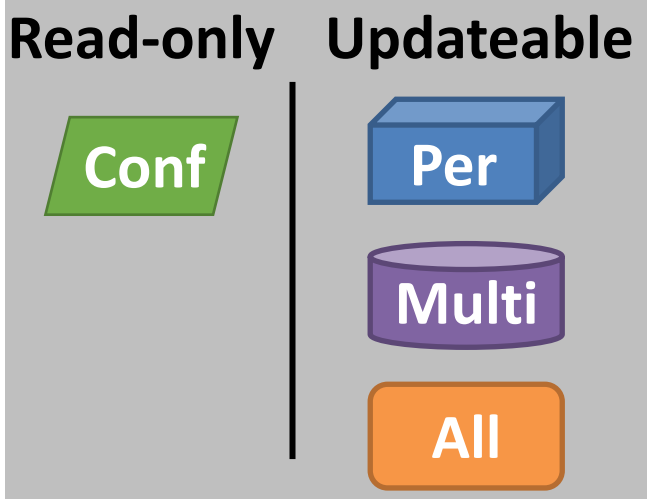
Whether the state is updated while processing the packet ?

- Strawman approach
  - Identify top-level variable on the left-hand-side(LHS) of assignment statement

per-/cross-flow var

```
int *index = &tail;
*index = (*index + 1)%100;
```

Falls short due to *aliasing*

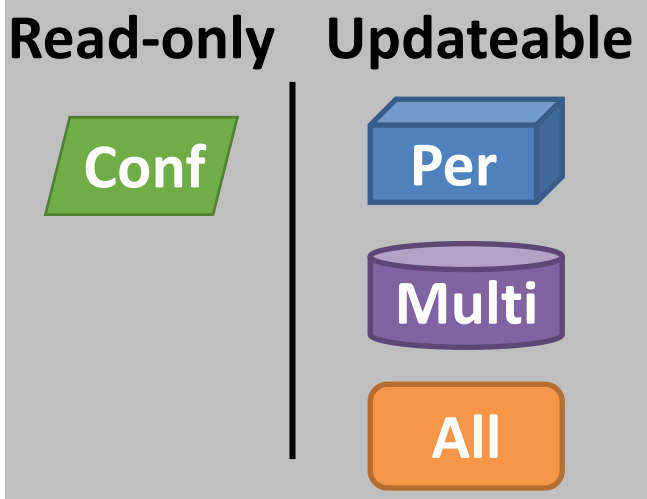


2. Identify updateable state

Whether the state is updated while processing the packet ?

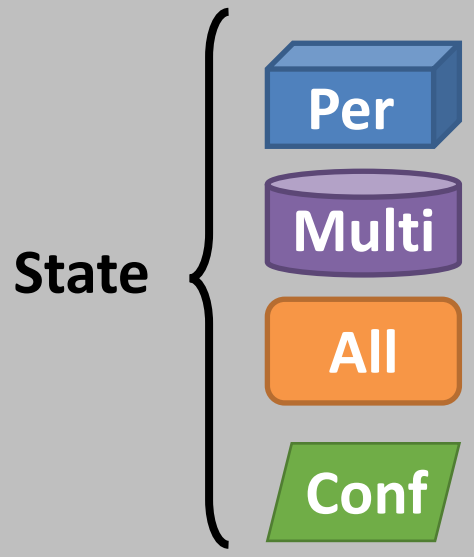
- Strawman approach
  - Identify top-level variable on the left-hand-side(LHS) of assignment statement

StateAlyzr employs flow-, context-, and field-insensitive ***pointer analysis*** to identify updateable variables

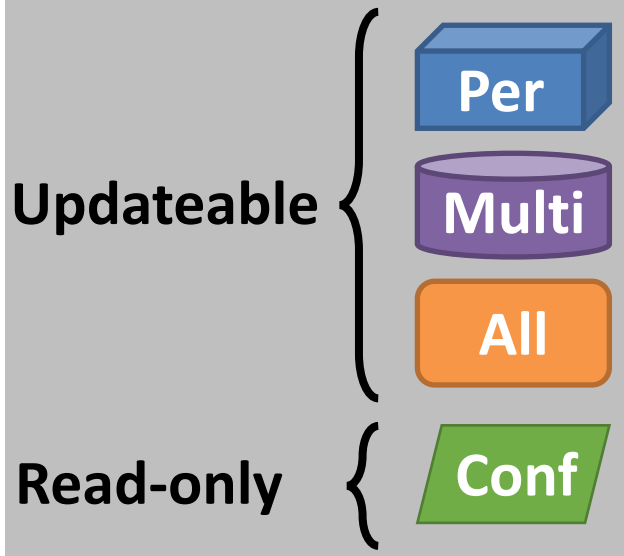


3. Identify states' flowspace dimensions

3. Identify states' flowspace dimensions

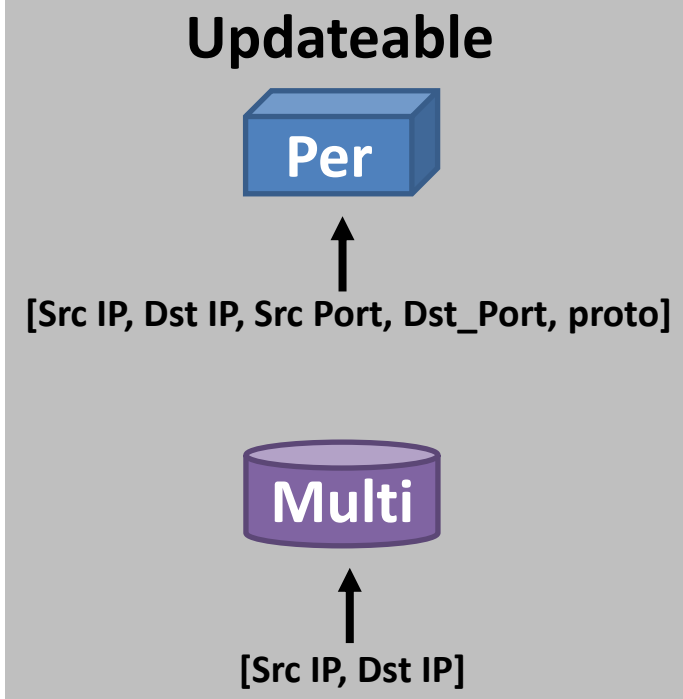


3. Identify states' flowspace dimensions



3. Identify states' flowspace dimensions

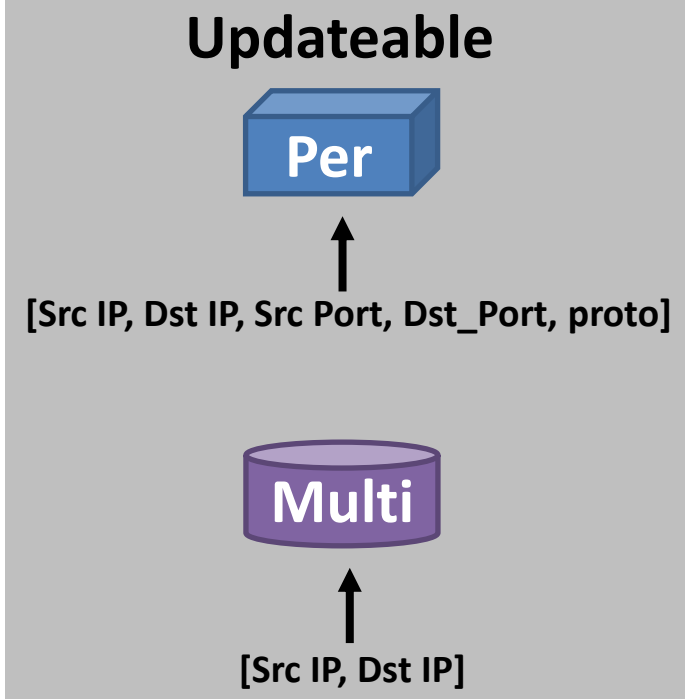
Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state



3. Identify states' flowspace dimensions

Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state

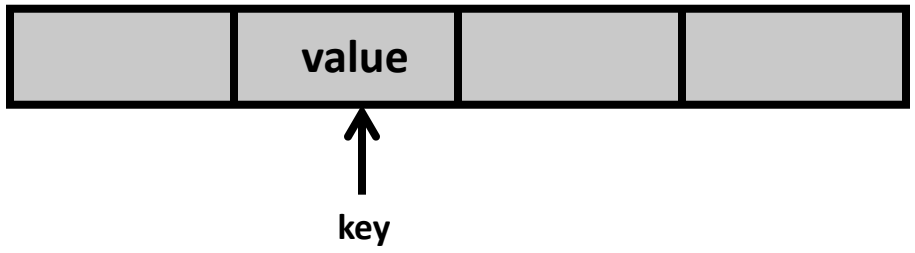
Common access patterns



3. Identify states' flowspace dimensions

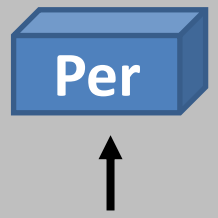
Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state

Hashtable

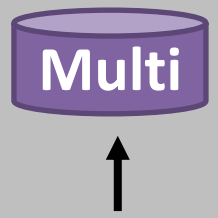


Common access patterns

Updateable



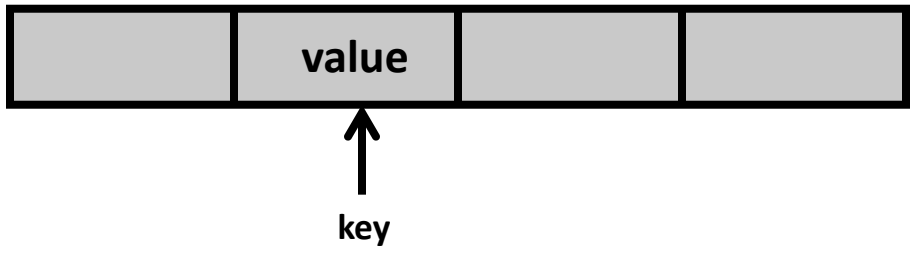
[Src IP, Dst IP, Src Port, Dst_Port, proto]



[Src IP, Dst IP]

3. Identify states' flowspace dimensions

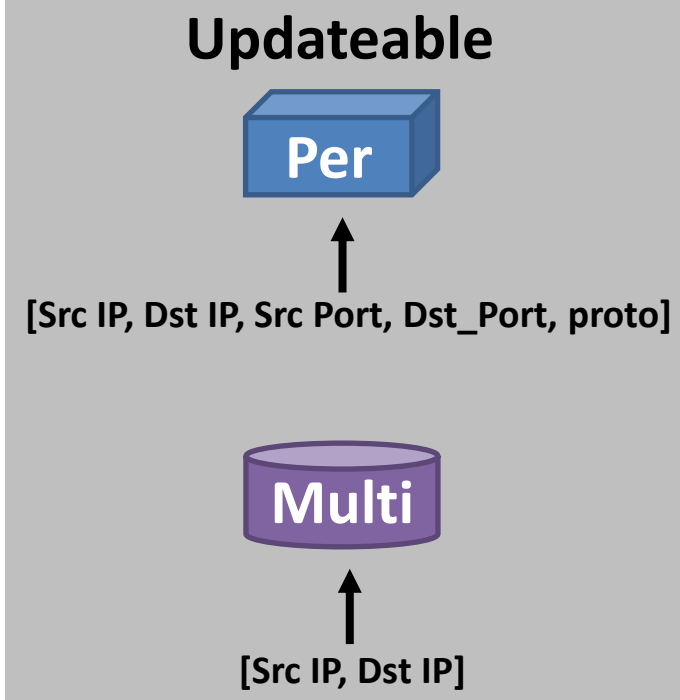
Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state



Common access patterns

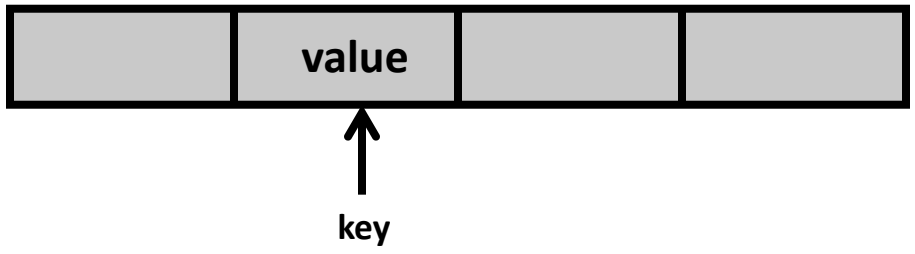
1. Square brackets

```
entry = table[index];
```



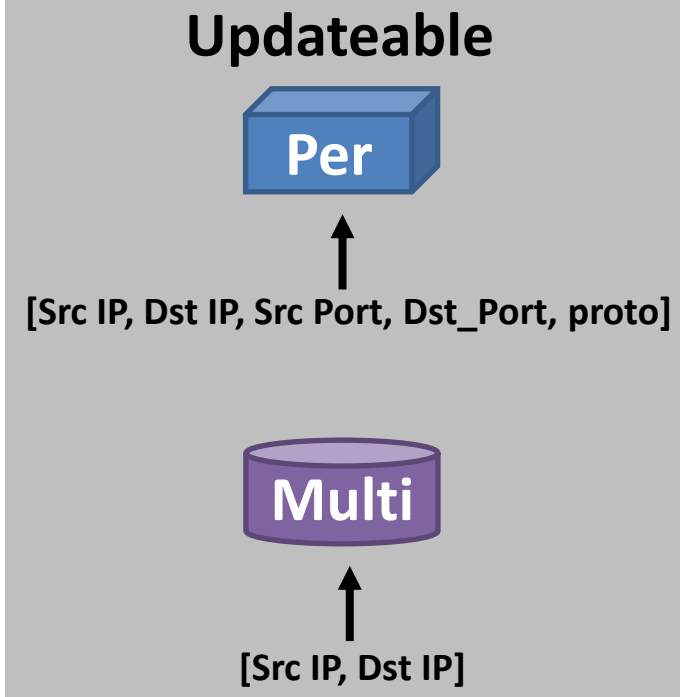
3. Identify states' flowspace dimensions

Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state



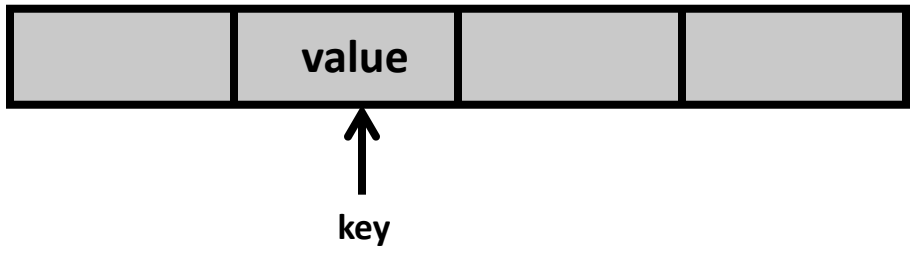
Common access patterns

1. Square brackets



3. Identify states' flowspace dimensions

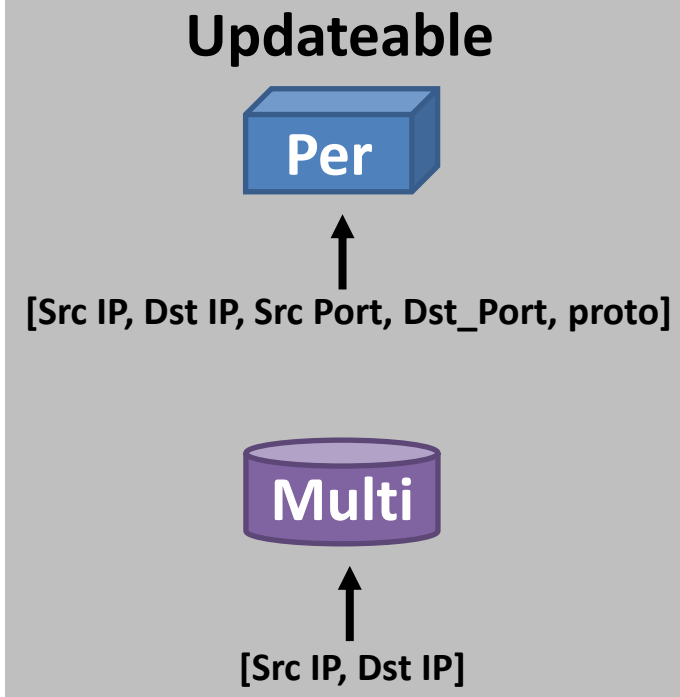
Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state



Common access patterns

1. Square brackets
2. Pointer arithmetic

```
entry = head + offset;
```

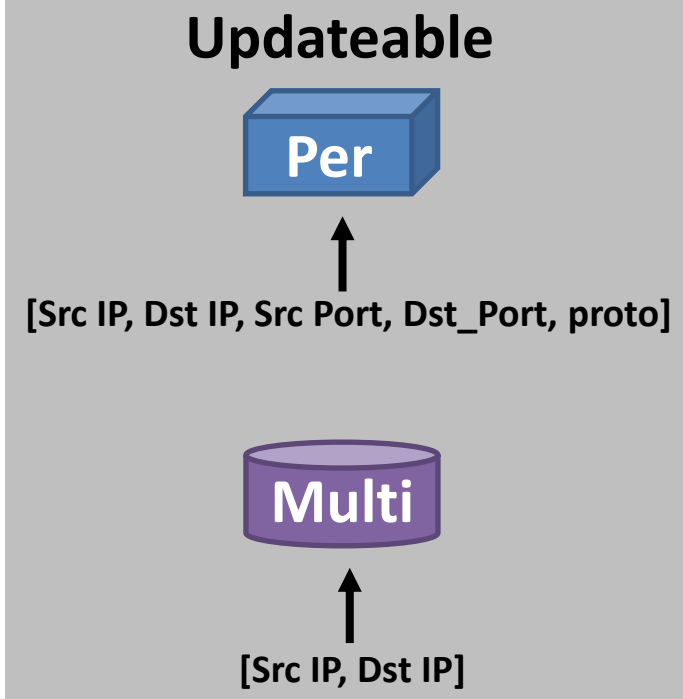


3. Identify states' flowspace dimensions

Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state

Common access patterns

1. Square brackets
2. Pointer arithmetic

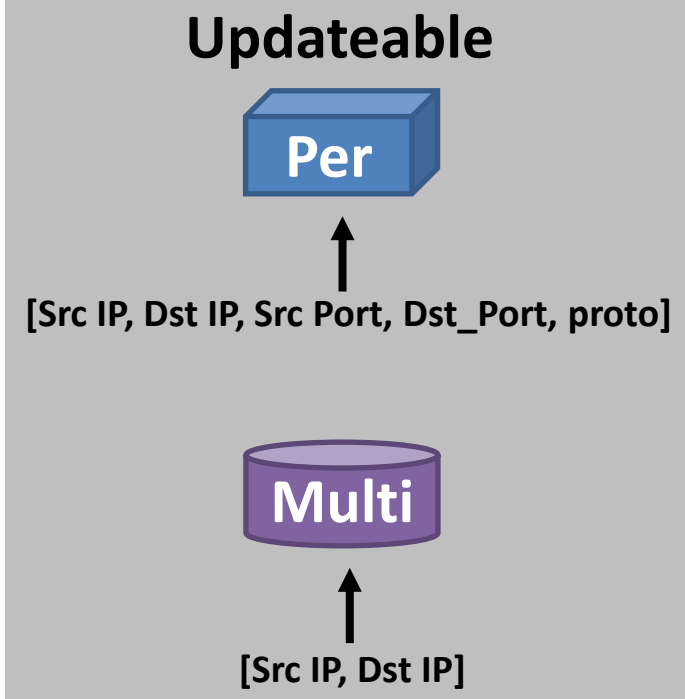


3. Identify states' flowspace dimensions

Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state

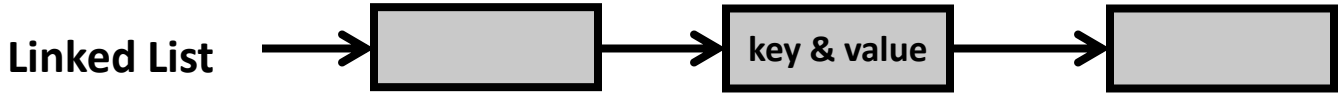
Common access patterns

1. Square brackets
2. Pointer arithmetic
3. Iteration



3. Identify states' flowspace dimensions

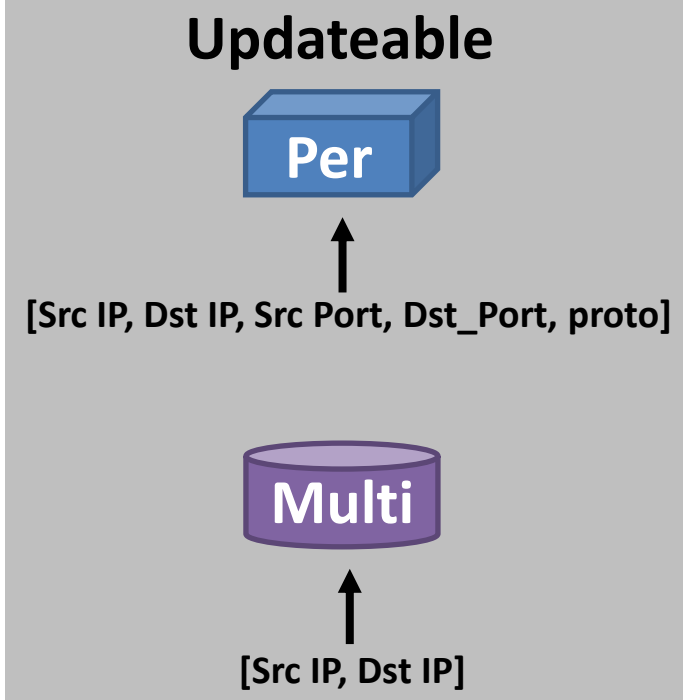
Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state



Common access patterns

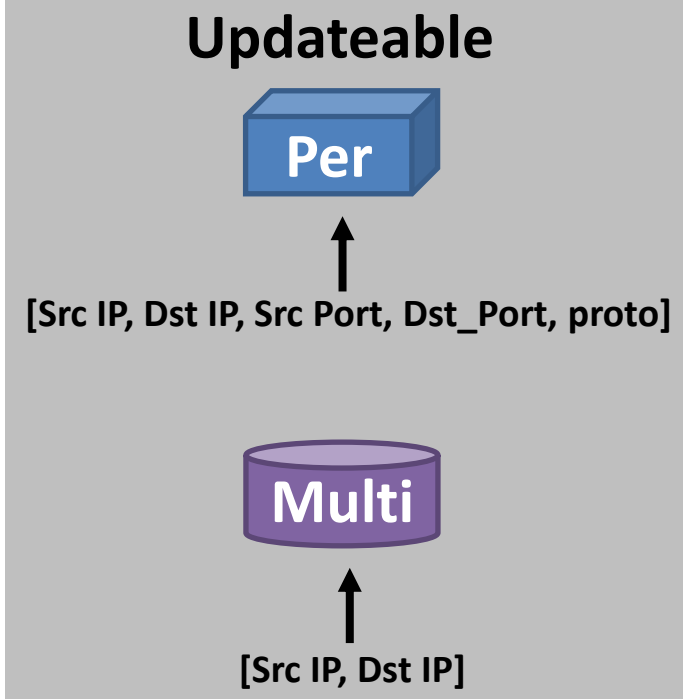
1. Square brackets
2. Pointer arithmetic
3. Iteration

```
struct host *lookup(uint ip) {
 struct host *curr = hosts;
 while (curr != NULL) {
  if (curr->ip == ip)
   return curr;
  curr = curr->next;
 }
}
```



3. Identify states' flowspace dimensions

Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state




```
entry = host_map[index]
```

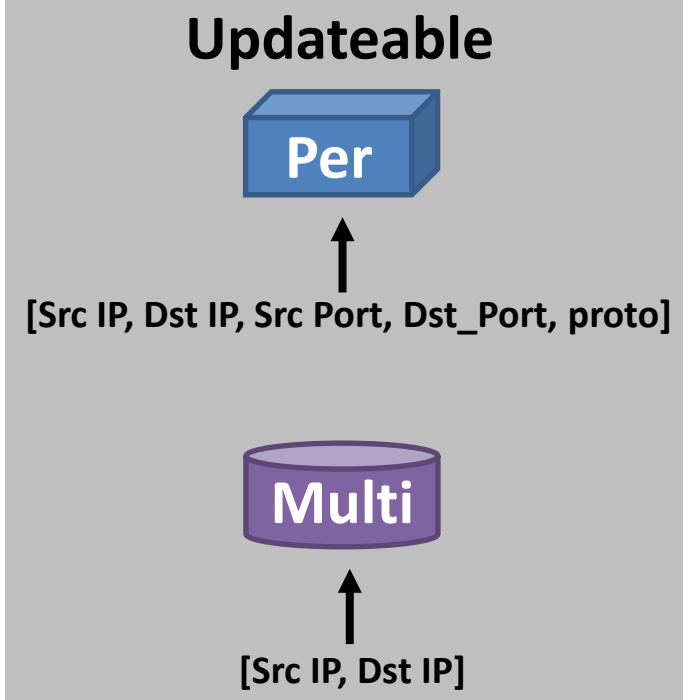
3. Identify states' flowspace dimensions

Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state

Program ***chopping*** to determine relevant ***header fields***

```
struct pktHdr *pkt = recv(extPcap);
```

```
entry = host_map[index]
```

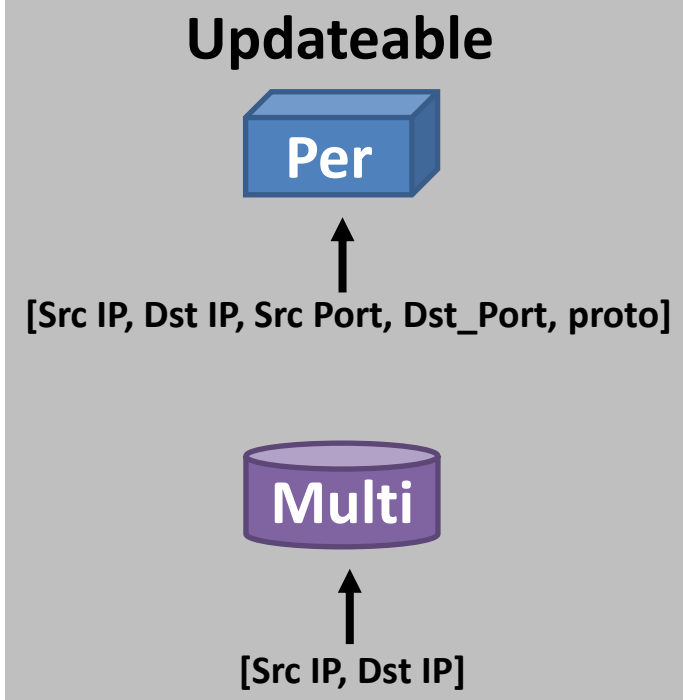


3. Identify states' flowspace dimensions

Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state

Program ***chopping*** to determine relevant ***header fields***

```
struct pktHdr *pkt = recv(extPcap);
src_ip = pkt->ip_src_addr;
packet_count ++;
index = src_ip + offset
entry = host_map[index]
```

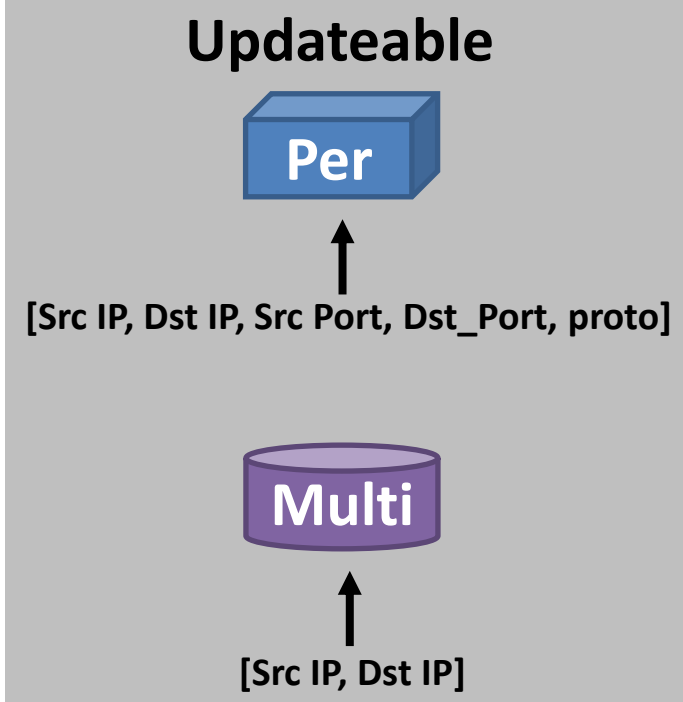


3. Identify states' flowspace dimensions

Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state

Program ***chopping*** to determine relevant ***header fields***

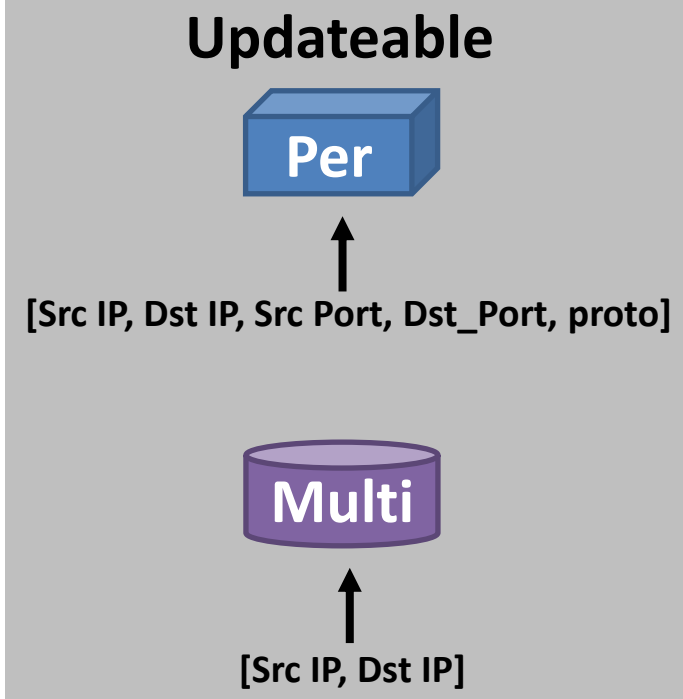
```
struct pktHdr *pkt = recv(extPcap);
src_ip = pkt->ip_src_addr;
packet_count ++;
index = src_ip + offset
entry = host_map[index]
```



3. Identify states' flowspace dimensions

Identify a set of ***packet header*** *fields* that delineate the subset of traffic that relates to the state

Program ***chopping*** to determine relevant ***header fields***

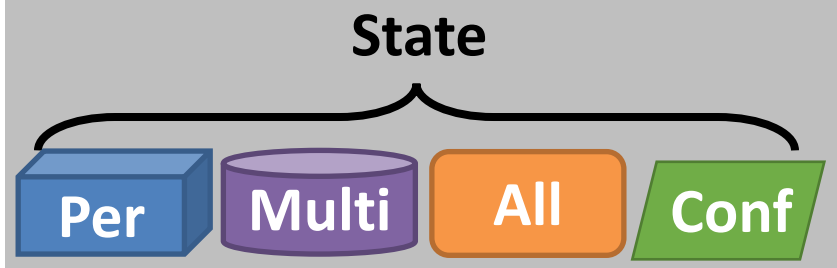


StateAlyzr steps

StateAlyzr steps

1. **Identify Per-/Cross-flow state**

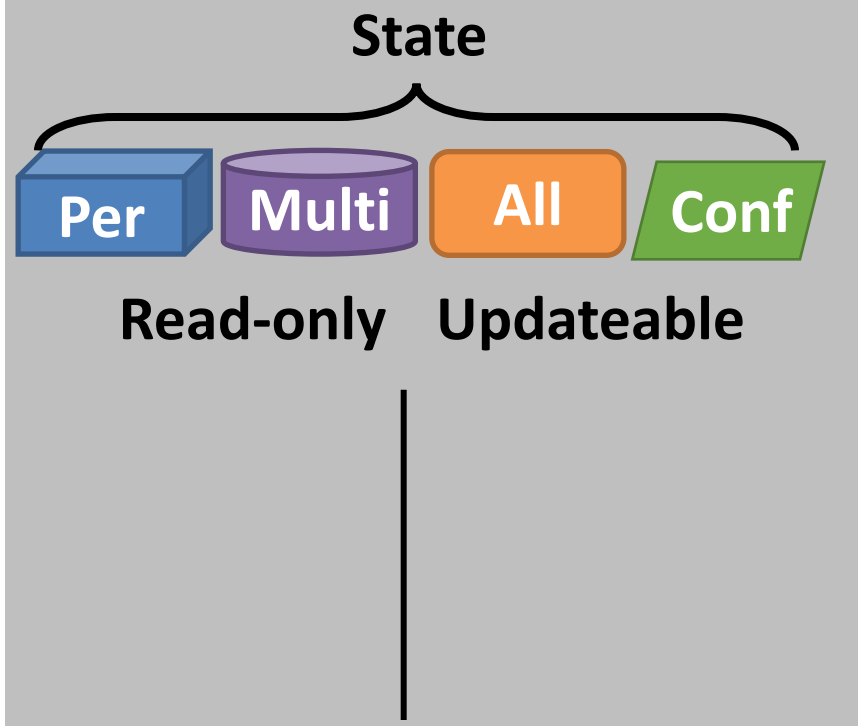
StateAlyzr steps



1. Identify Per-/Cross-flow state

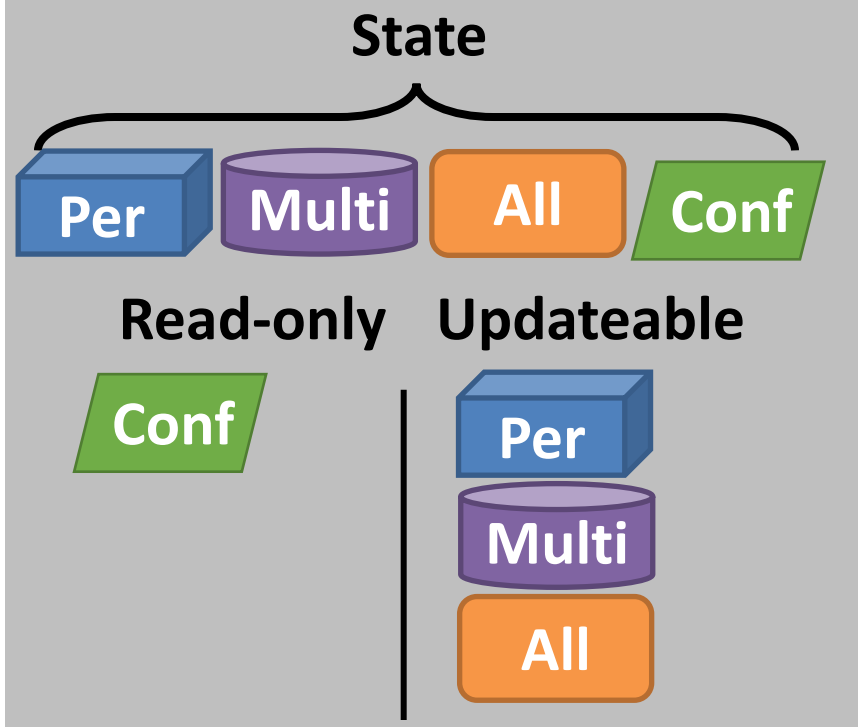
StateAlyzr steps

1. Identify Per-/Cross-flow state
2. Identify Updateable State



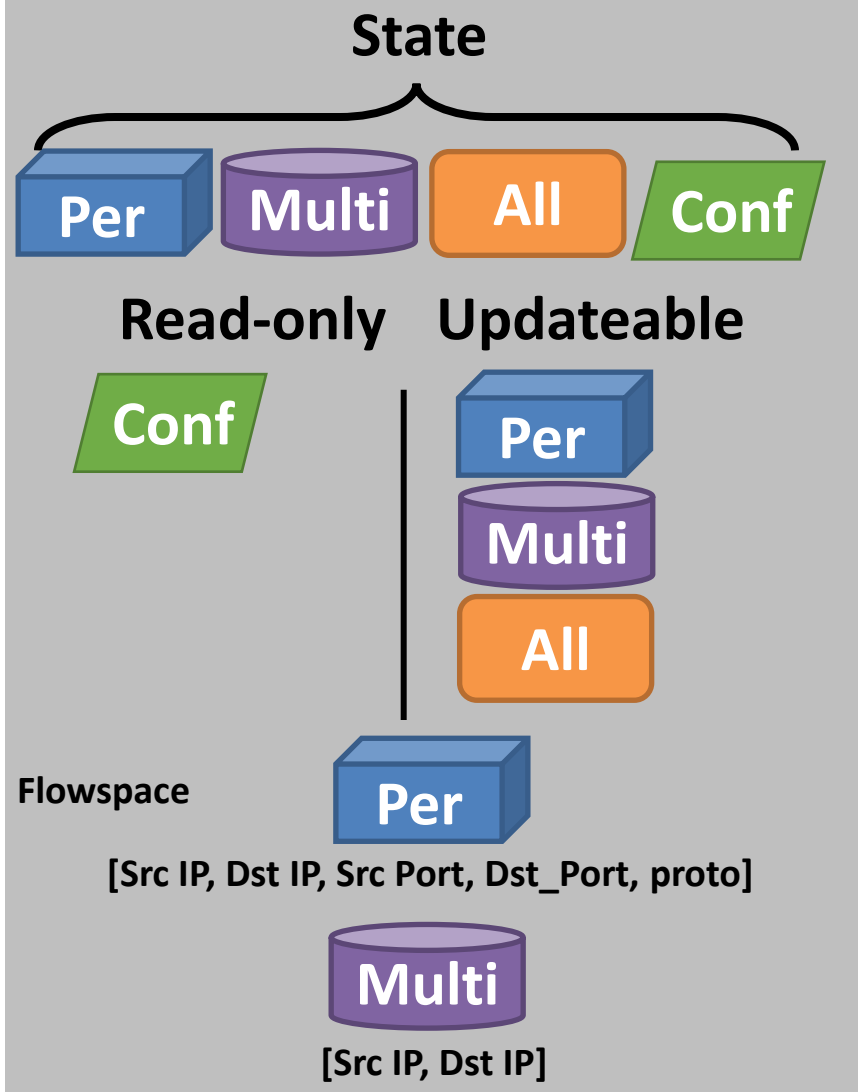
StateAlyzr steps

1. Identify Per-/Cross-flow state
2. Identify Updateable State



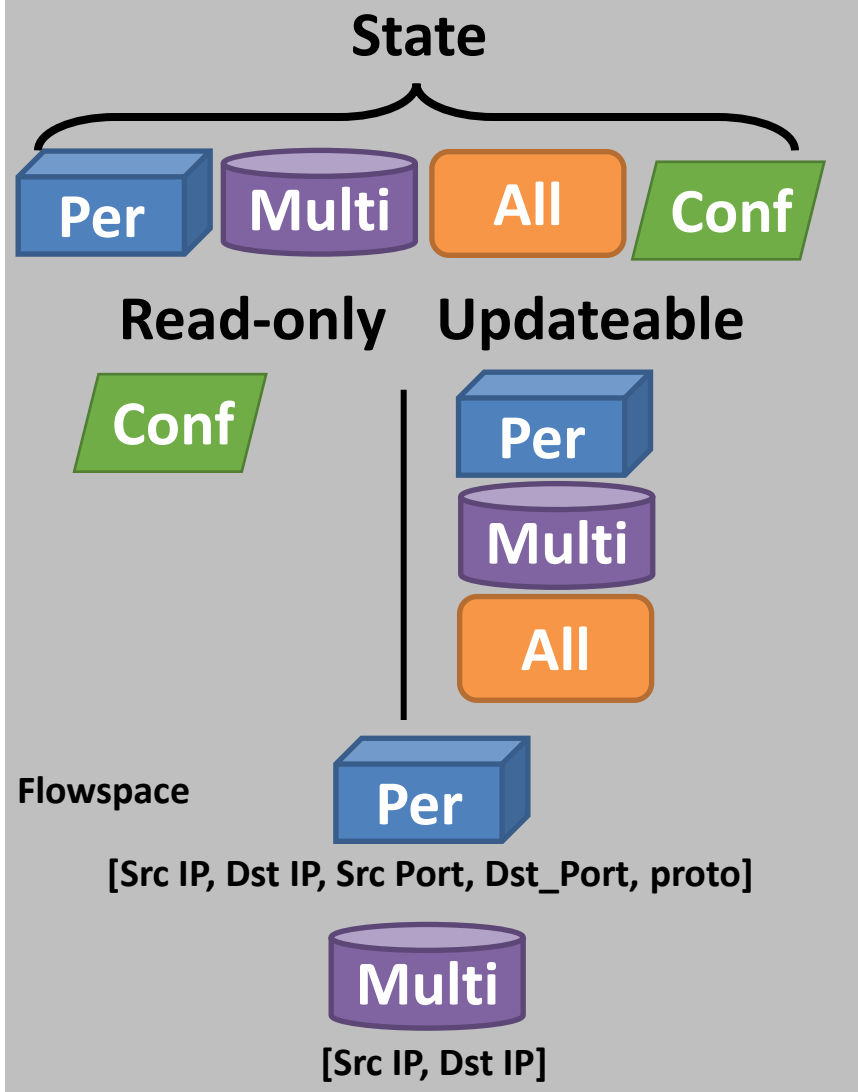
StateAlyzr steps

1. Identify Per-/Cross-flow state
2. Identify Updateable State
3. Identify States' Flowspace Dimensions



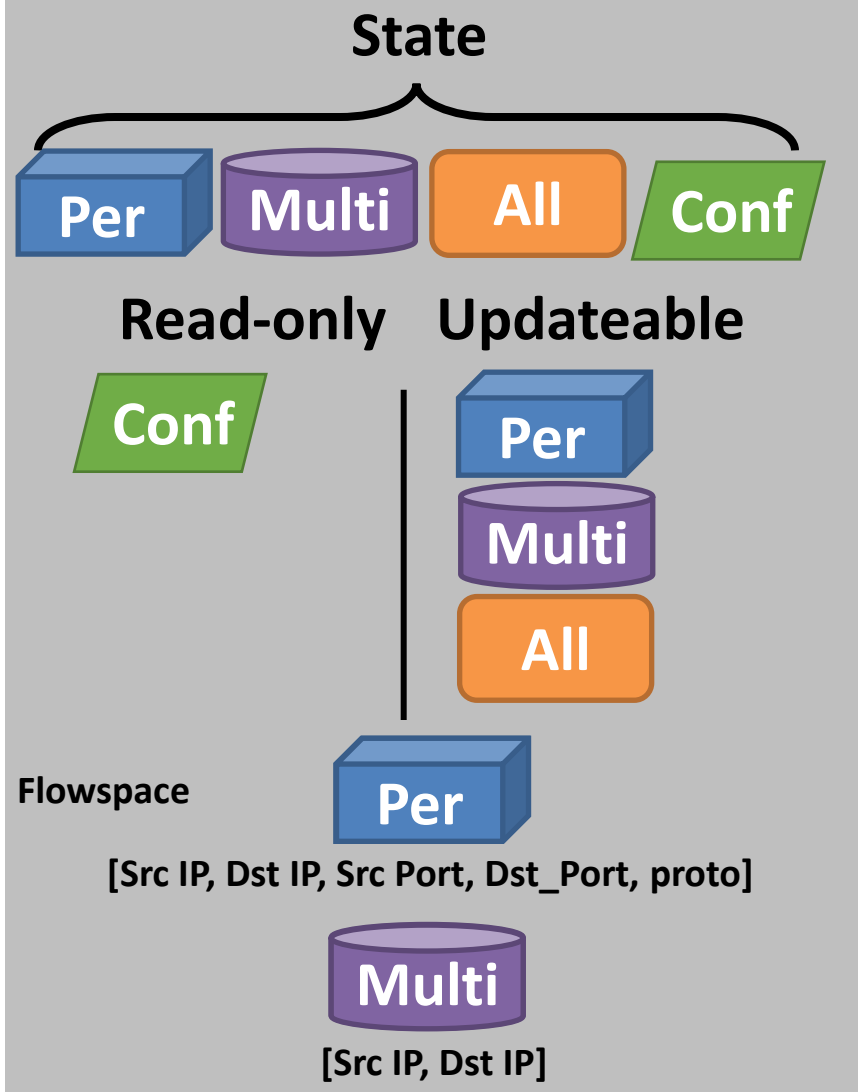
StateAlyzr steps

1. Identify Per-/Cross-flow state
2. Identify Updateable State
3. Identify States' Flowspace Dimensions
4. Output Impacting State



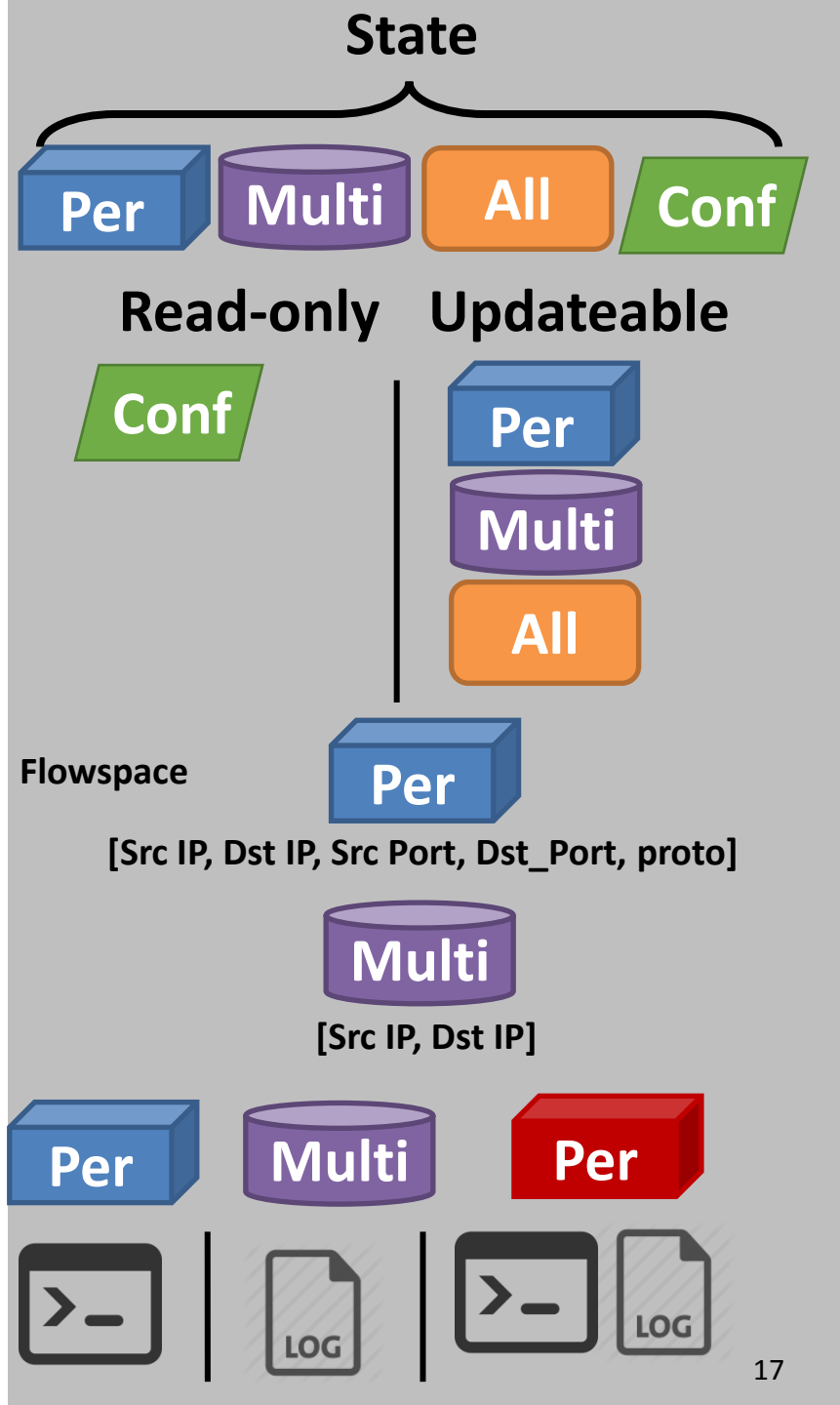
StateAlyzr steps

1. **Identify Per-/Cross-flow state**
2. **Identify Updateable State**
3. **Identify States' Flowspace Dimensions**
4. **Output Impacting State**
   - Identify the type of output (log or packet) that updateable state affects



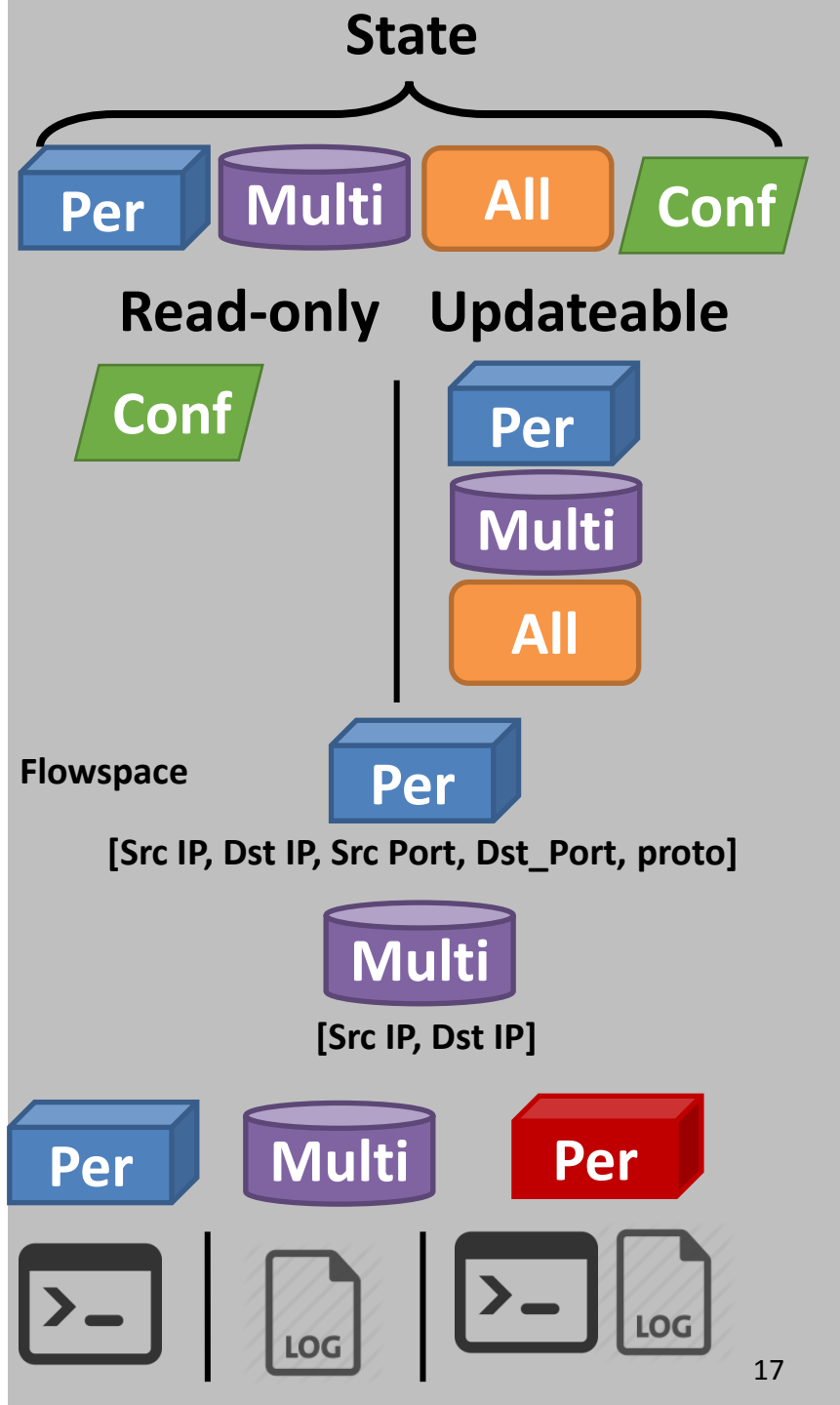
StateAlyzr steps

1. **Identify Per-/Cross-flow state**
2. **Identify Updateable State**
3. **Identify States' Flowspace Dimensions**
4. **Output Impacting State**
   - Identify the type of output (log or packet) that updateable state affects



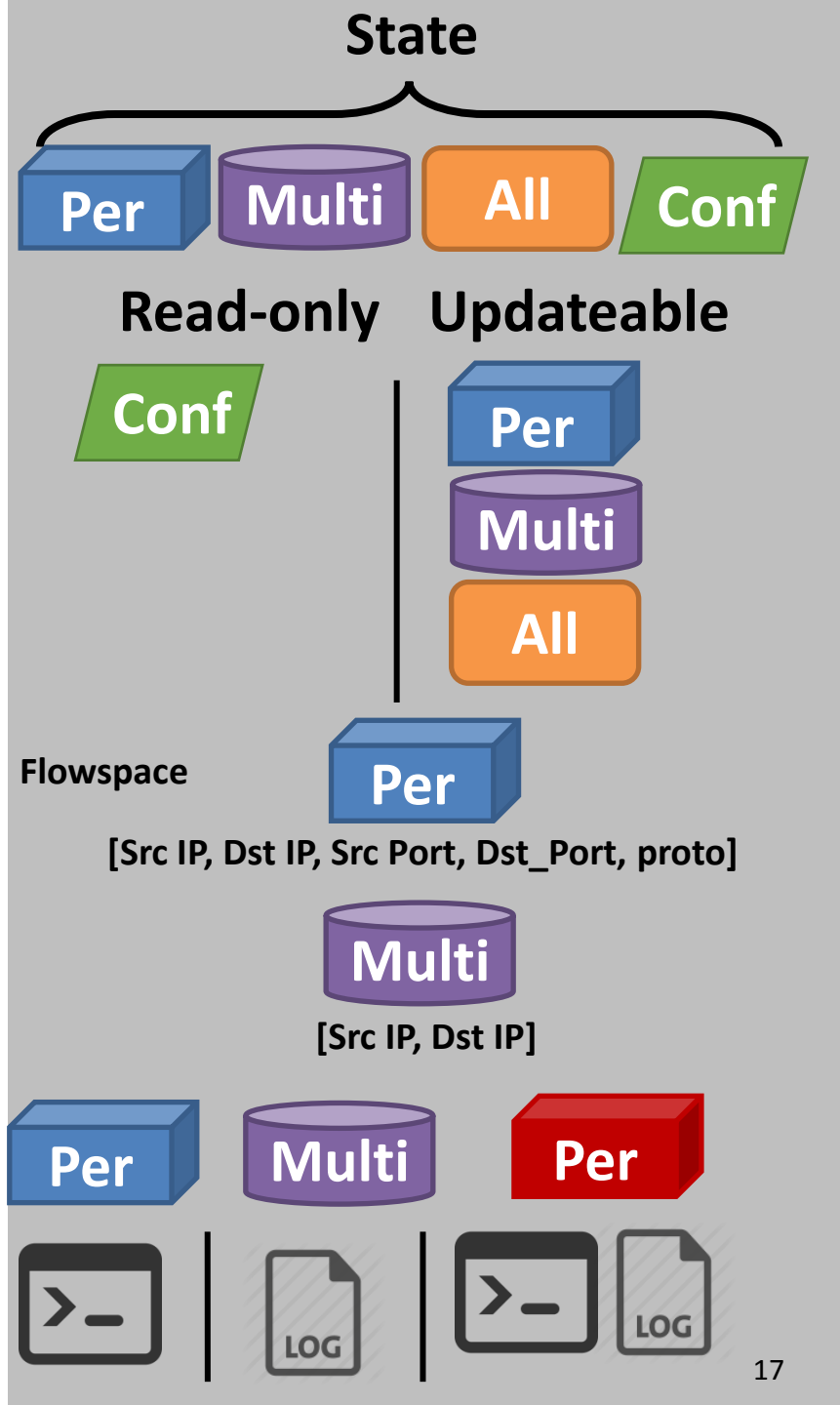
StateAlyzr steps

1. **Identify Per-/Cross-flow state**
2. **Identify Updateable State**
3. **Identify States' Flowspace Dimensions**
4. **Output Impacting State**
   - Identify the type of output (log or packet) that updateable state affects
5. **Tracking Run-time Update**



StateAlyzr steps

1. **Identify Per-/Cross-flow state**
2. **Identify Updateable State**
3. **Identify States' Flowspace Dimensions**
4. **Output Impacting State**
   - Identify the type of output (log or packet) that updateable state affects
5. **Tracking Run-time Update**
   - Insert statements to do run time monitoring to track whether a variable is updated



Implementation

Implementation

Used CodeSurfer to implement StateAlyzr

Implementation

Used CodeSurfer to implement StateAlyzr

- CodeSurfer has built-in support for

Implementation

Used CodeSurfer to implement StateAlyzr

- CodeSurfer has built-in support for
  - Control flow graph construction

Implementation

Used CodeSurfer to implement StateAlyzr

- CodeSurfer has built-in support for
  - Control flow graph construction
  - Flow and context-insensitive pointer analysis

Implementation

Used CodeSurfer to implement StateAlyzr

- CodeSurfer has built-in support for
  - Control flow graph construction
  - Flow and context-insensitive pointer analysis
  - Forward/backward slice and chop computation

Implementation

Used CodeSurfer to implement StateAlyzr

- CodeSurfer has built-in support for
  - Control flow graph construction
  - Flow and context-insensitive pointer analysis
  - Forward/backward slice and chop computation

Analyzed four open-source middleboxes

Implementation

Used CodeSurfer to implement StateAlyzr

- CodeSurfer has built-in support for
  - Control flow graph construction
  - Flow and context-insensitive pointer analysis
  - Forward/backward slice and chop computation

Analyzed four open-source middleboxes

1. PRADS – a monitoring middlebox

Implementation

Used CodeSurfer to implement StateAlyzr

- CodeSurfer has built-in support for
  - Control flow graph construction
  - Flow and context-insensitive pointer analysis
  - Forward/backward slice and chop computation

Analyzed four open-source middleboxes

1. PRADS – a monitoring middlebox
2. Snort – an IDS

Implementation

Used CodeSurfer to implement StateAlyzr

- CodeSurfer has built-in support for
  - Control flow graph construction
  - Flow and context-insensitive pointer analysis
  - Forward/backward slice and chop computation

Analyzed four open-source middleboxes

1. PRADS – a monitoring middlebox
2. Snort – an IDS
3. HAProxy – a load balancing proxy

Implementation

Used CodeSurfer to implement StateAlyzr

- CodeSurfer has built-in support for
  - Control flow graph construction
  - Flow and context-insensitive pointer analysis
  - Forward/backward slice and chop computation

Analyzed four open-source middleboxes

1. PRADS – a monitoring middlebox
2. Snort – an IDS
3. HAProxy – a load balancing proxy
4. OpenVPN – a VPN gateway

Evaluation

Evaluation

- Precision

Evaluation

- Precision
- Performance benefits at run time

Evaluation: effectiveness

	Step 0		Step 1	Step 2
MB	**All variables**	**Persistent variables**	**per-/cross-flow variables**	**Updateable variables**
PRADS	1529	61	29	10
Snort IDS	18393	507	333	148
HAproxy	7876	272	176	115
OpenVPN	8704	156	131	106

Evaluation: effectiveness

	Step 0		Step 1	Step 2
MB	**All variables**	**Persistent variables**	**per-/cross-flow variables**	**Updateable variables**
PRADS	1529	61	29	10
Snort IDS	18393	507	333	148
HAproxy	7876	272	176	115
OpenVPN	8704	156	131	106

Evaluation: effectiveness

MB	Step 0 All variables	Persistent variables	Step 1 per-/cross-flow variables	Step 2 Updateable variables
PRADS	1529	61	29	10
Snort IDS	18393	507	333	148
HAproxy	7876	272	176	115
OpenVPN	8704	156	131	106

StateAlyzr offers useful ***improvements*** in ***precision***

Evaluation: effectiveness

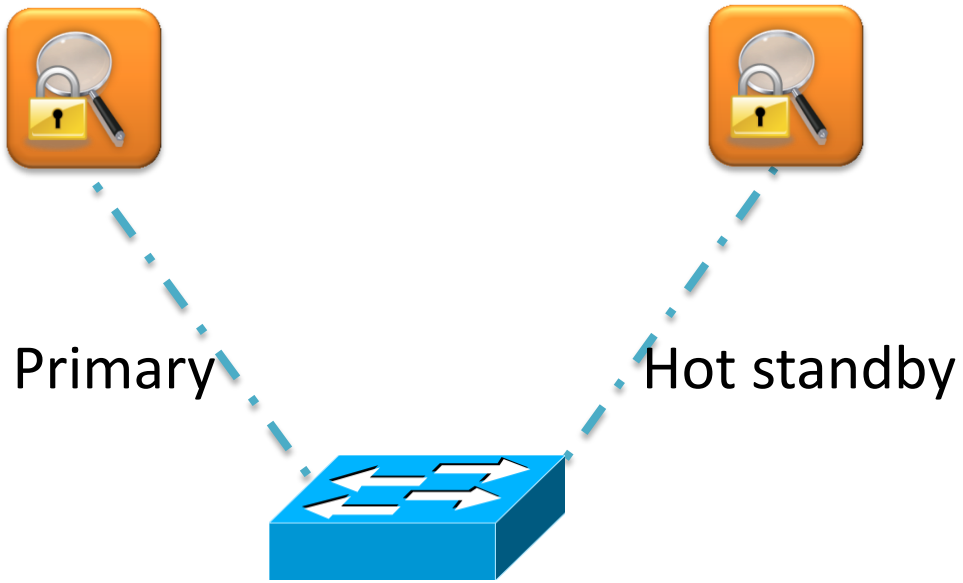
	Step 0		Step 1	Step 2
MB	**All variables**	**Persistent variables**	**per-/cross-flow variables**	**Updateable variables**
PRADS	1529	61	29	10
Snort IDS	18393	507	333	148
HAproxy	7876	272	176	115
OpenVPN	8704	156	131	106

StateAlyzr offers useful ***improvements*** in ***precision***

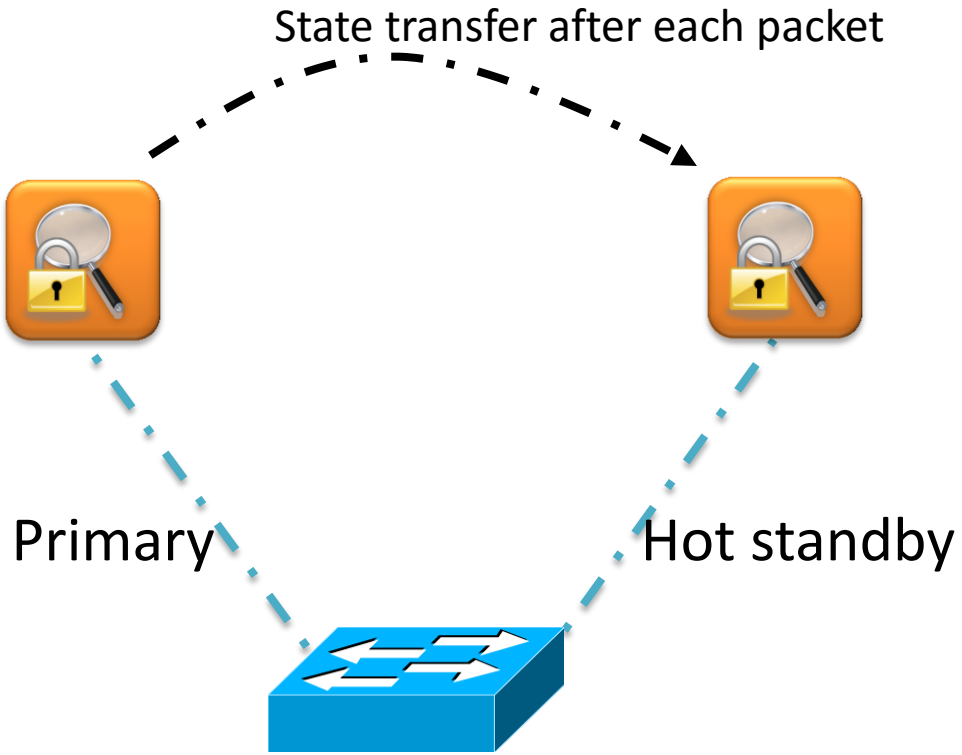
Theoretically ***proved*** the ***soundness*** of our algorithms

Highly available PRADS

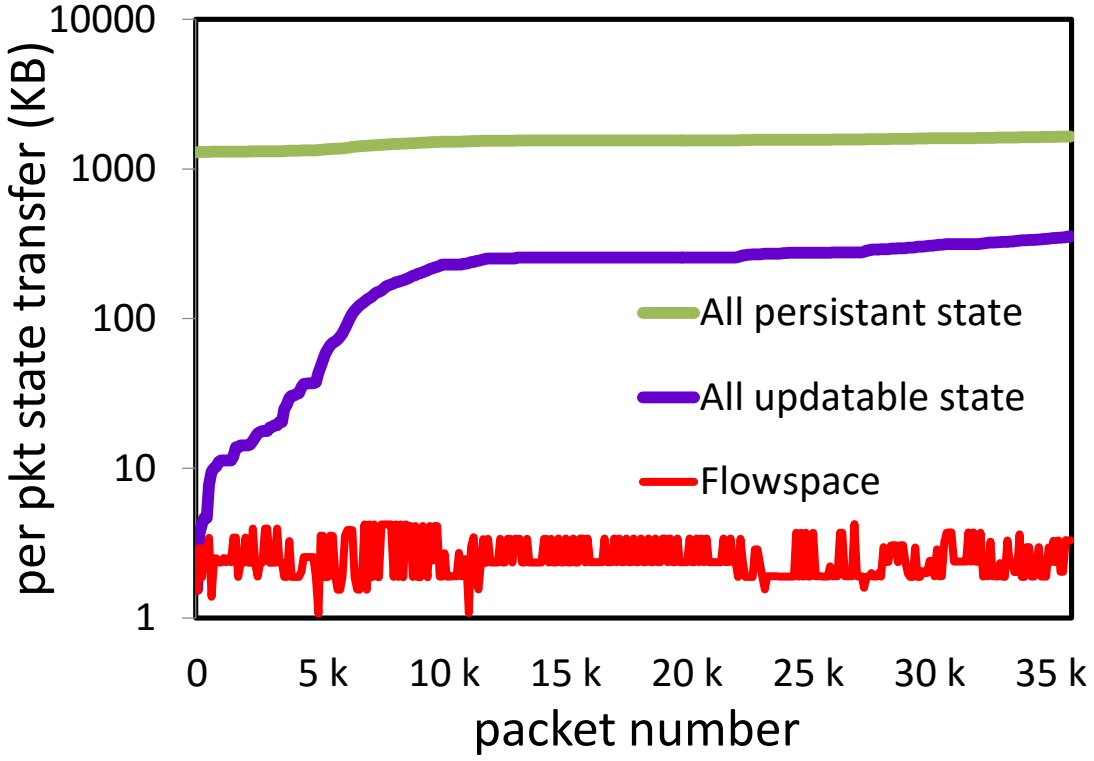
Highly available PRADS

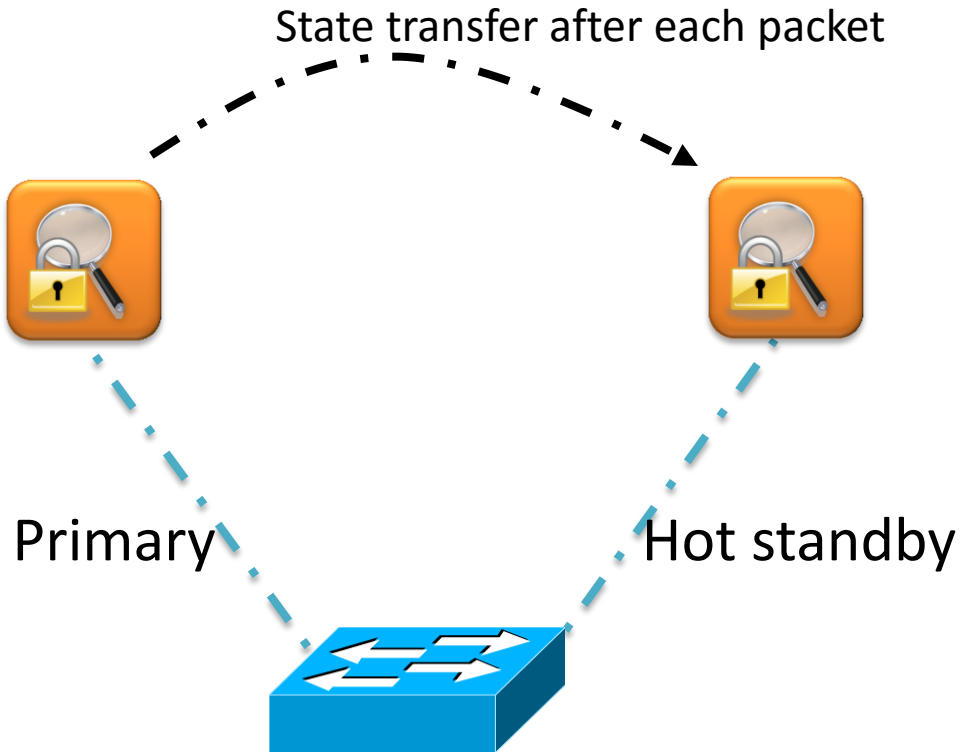


Highly available PRADS

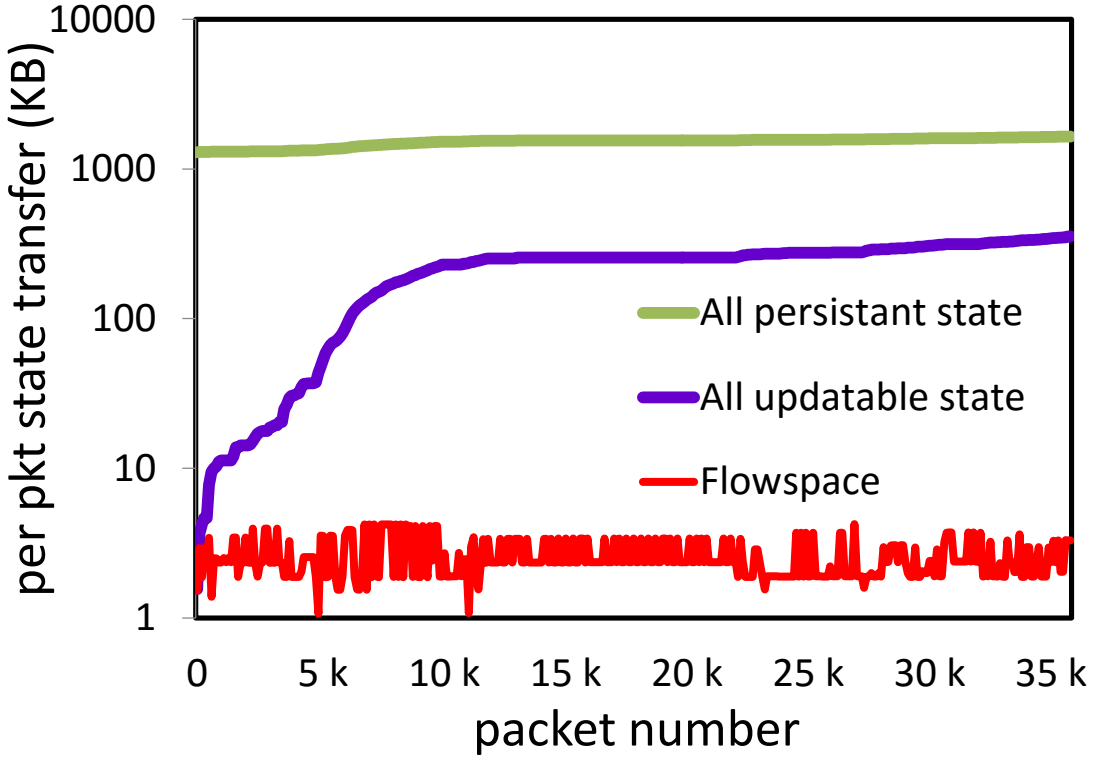


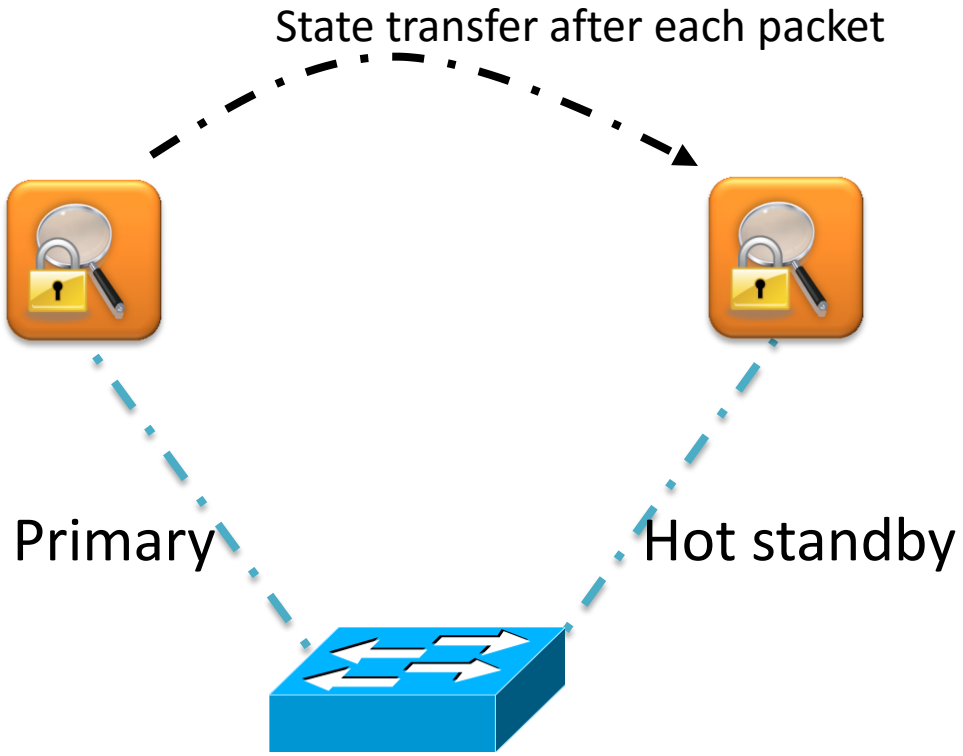
Highly available PRADS



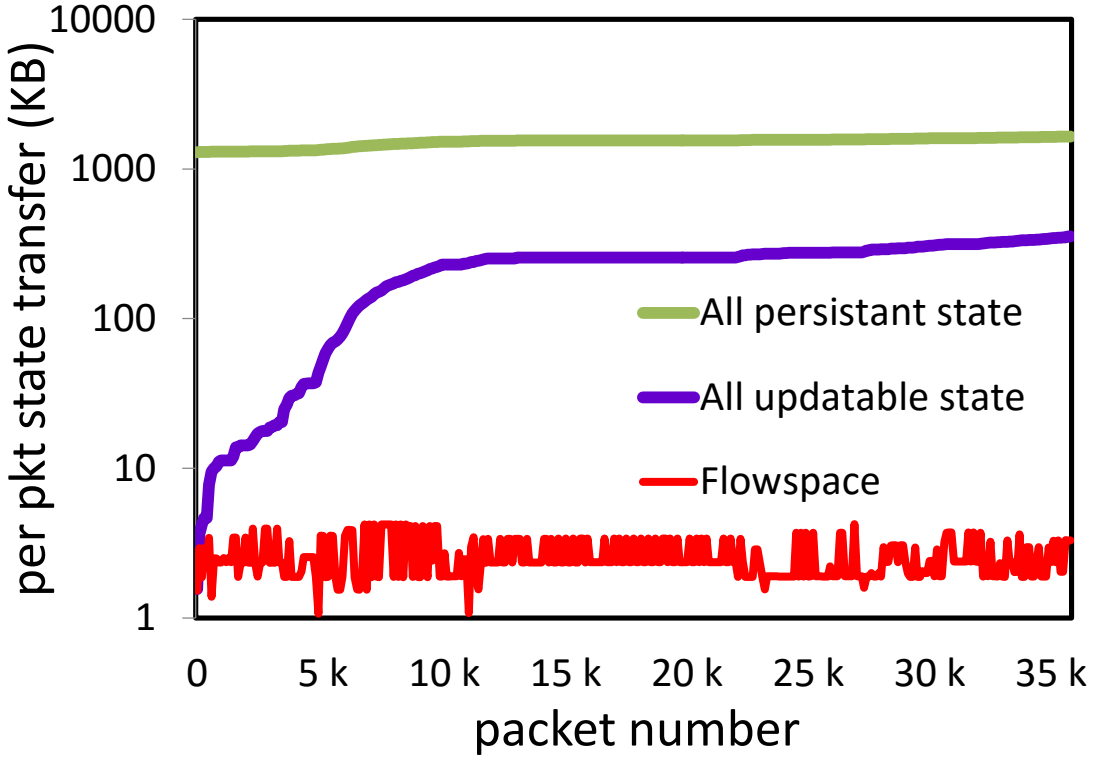


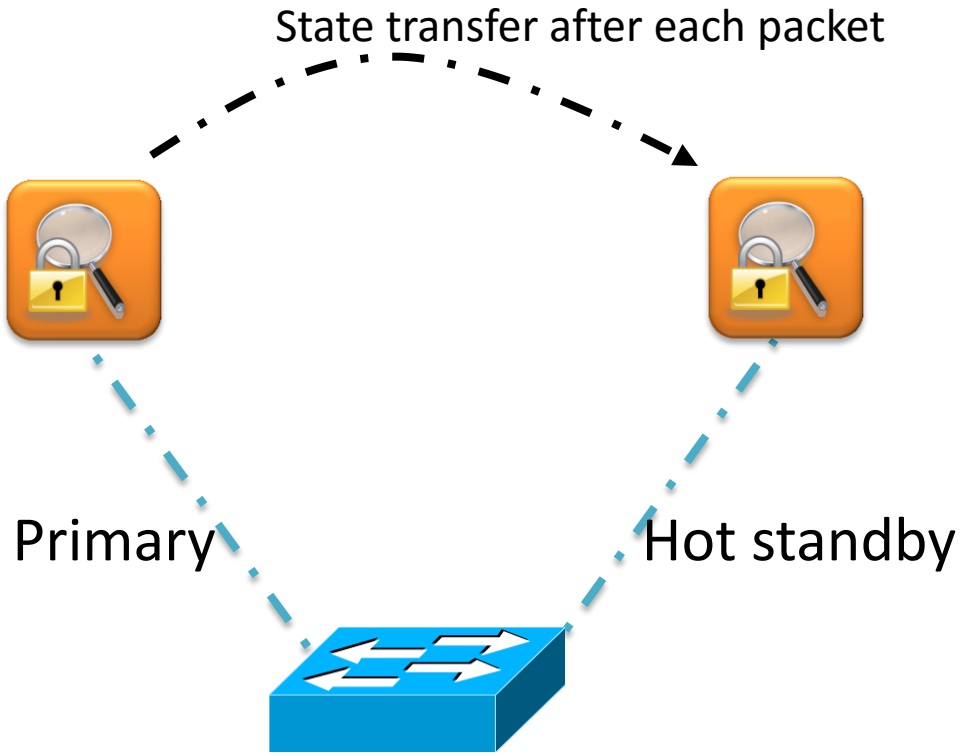
Highly available PRADS



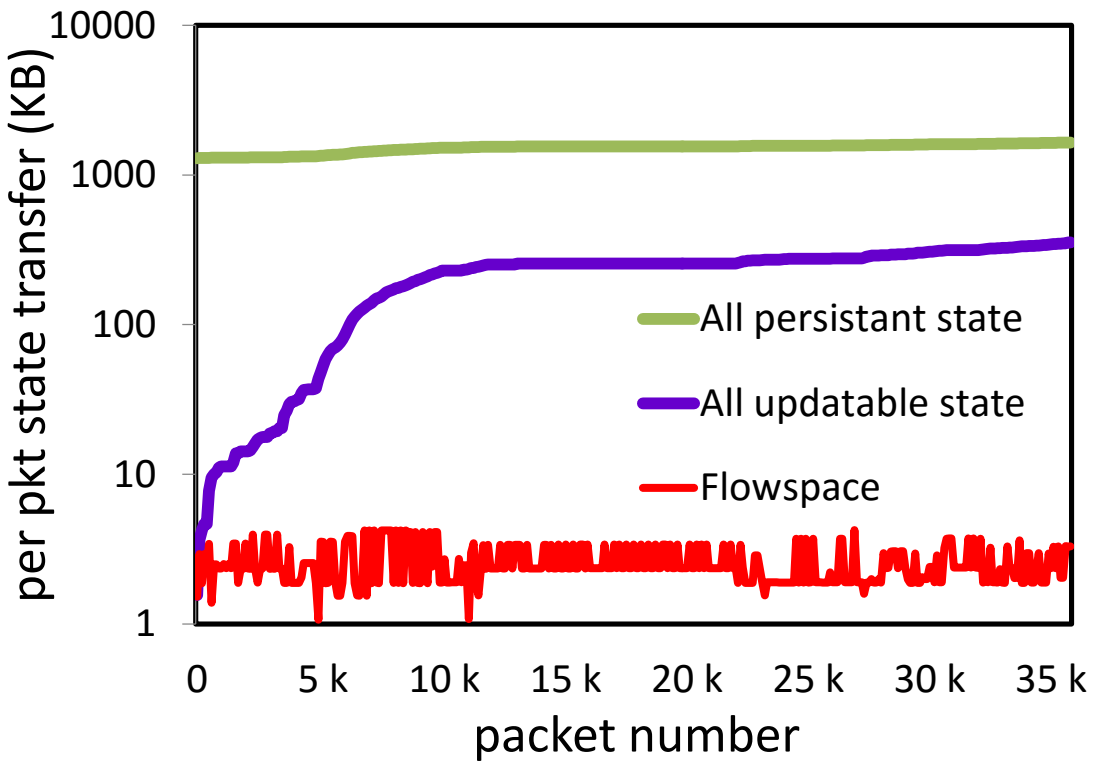


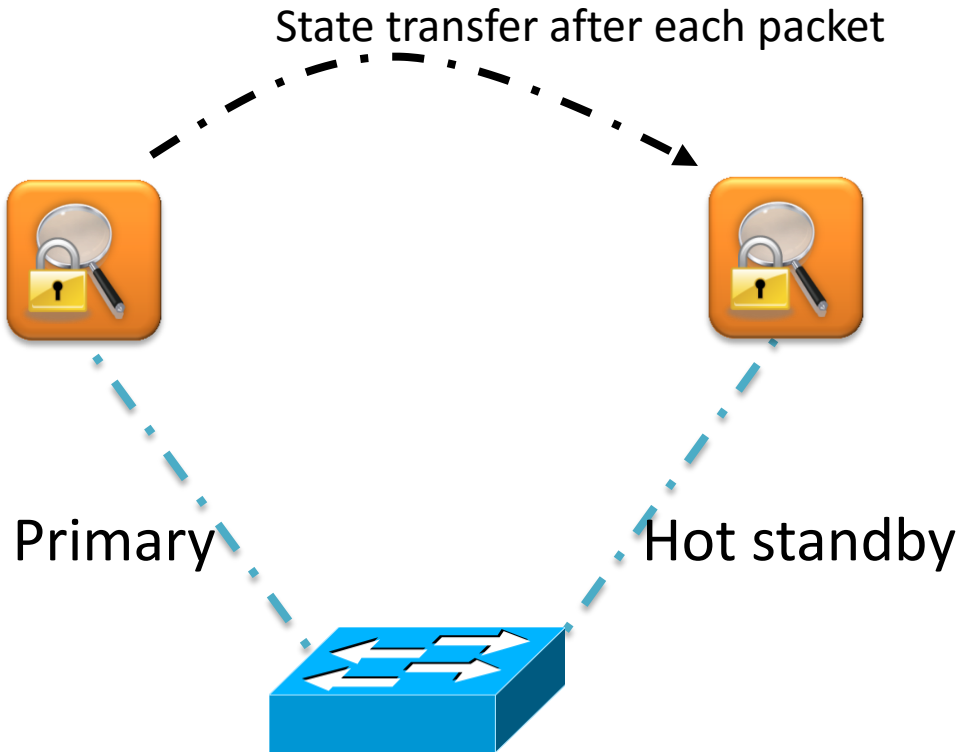
Highly available PRADS





Highly available PRADS



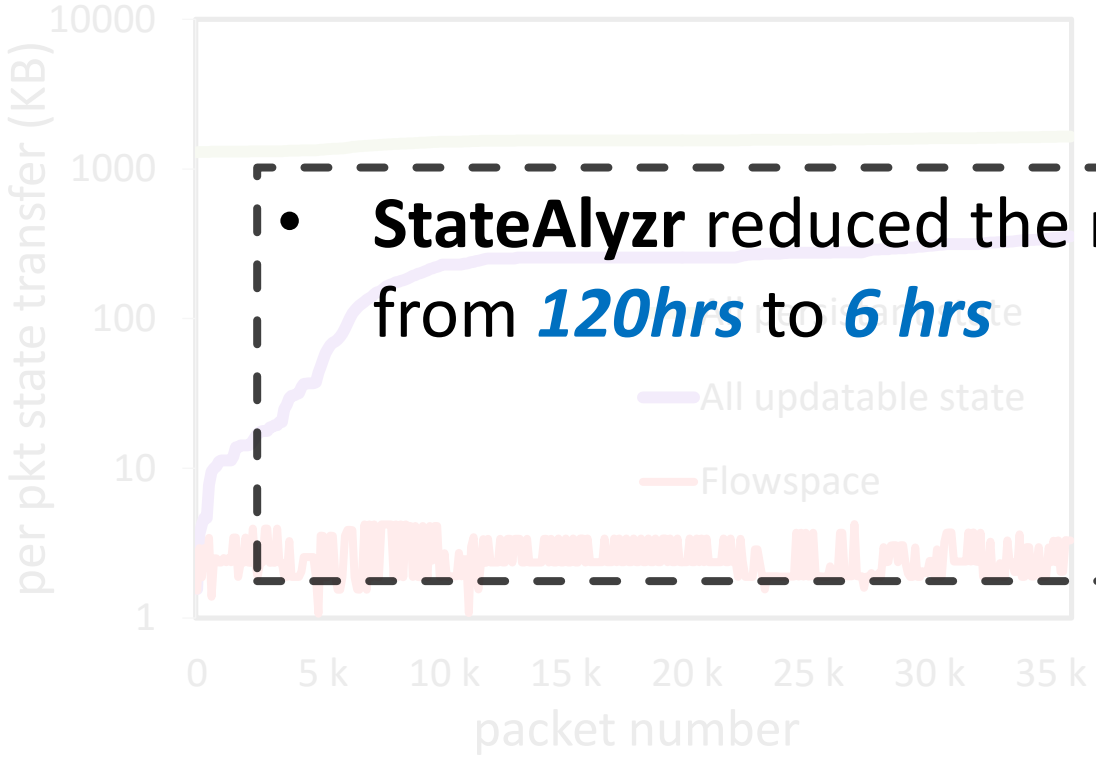


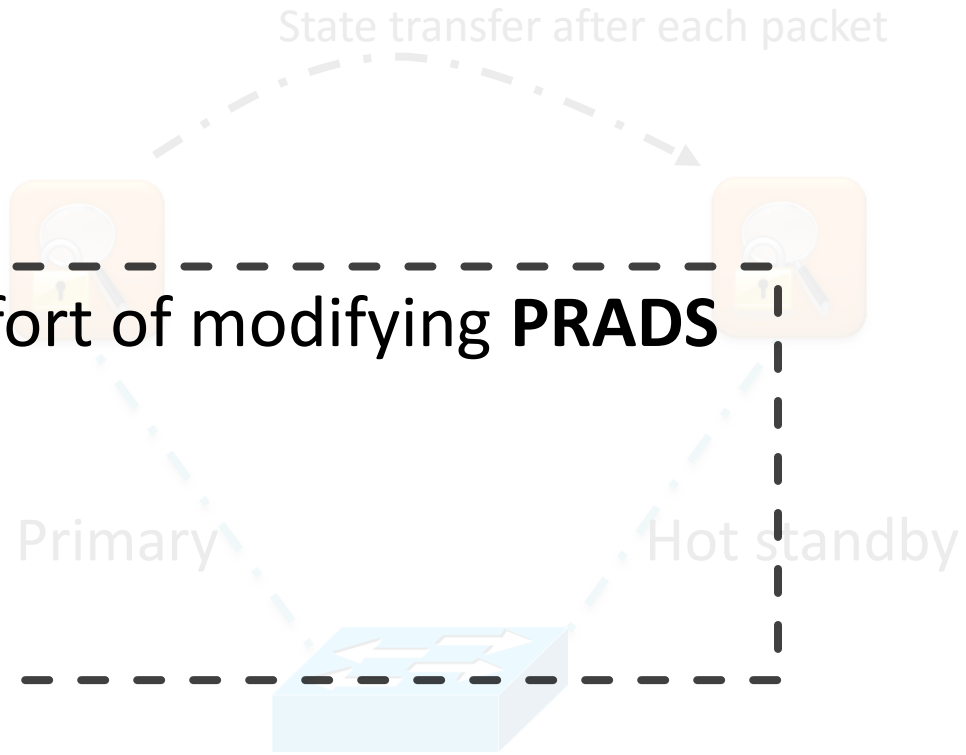
Reduction in the state transfer by *305x*

Highly available PRADS

Reduction in the state transfer by *305x*

Highly available PRADS

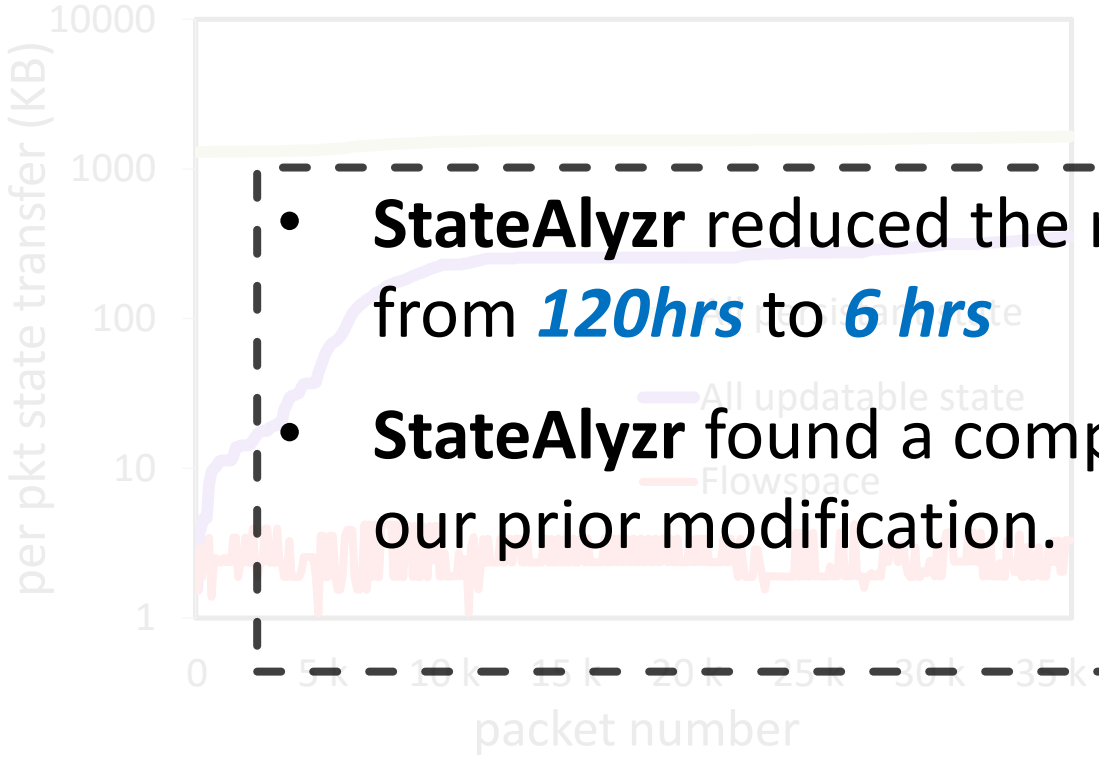


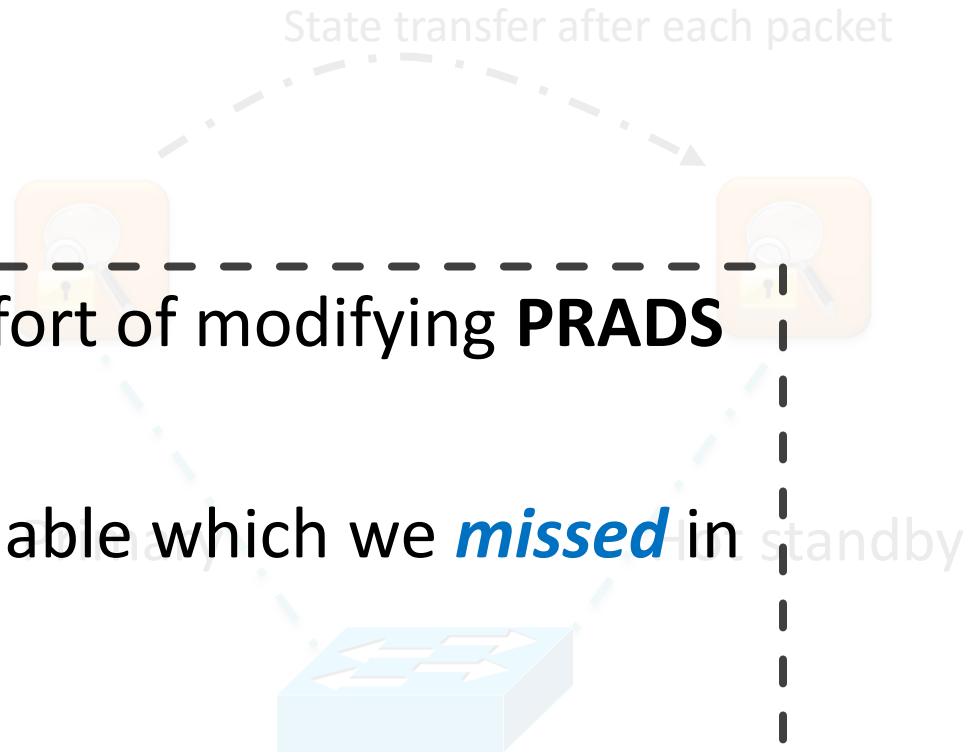


- **StateAlyzr** reduced the manual effort of modifying **PRADS** from ***120hrs*** to ***6 hrs***

Reduction in the state transfer by *305x*

Highly available PRADS





- **StateAlyzr** reduced the manual effort of modifying **PRADS** from ***120hrs*** to ***6 hrs***
- **StateAlyzr** found a compound variable which we ***missed*** in our prior modification.

Reduction in the state transfer by *305x*

Summary

Summary

- Goal is to aid middlebox developers to identify state objects that need explicit handling

Summary

- Goal is to aid middlebox developers to identify state objects that need explicit handling
- Novel state characterization algorithms that adapt standard program analysis tools

Summary

- Goal is to aid middlebox developers to identify state objects that need explicit handling
- Novel state characterization algorithms that adapt standard program analysis tools
- Ensure soundness and high precision

Summary

- Goal is to aid middlebox developers to identify state objects that need explicit handling
- Novel state characterization algorithms that adapt standard program analysis tools
- Ensure soundness and high precision
- Ultimate goal is to fully automate the process