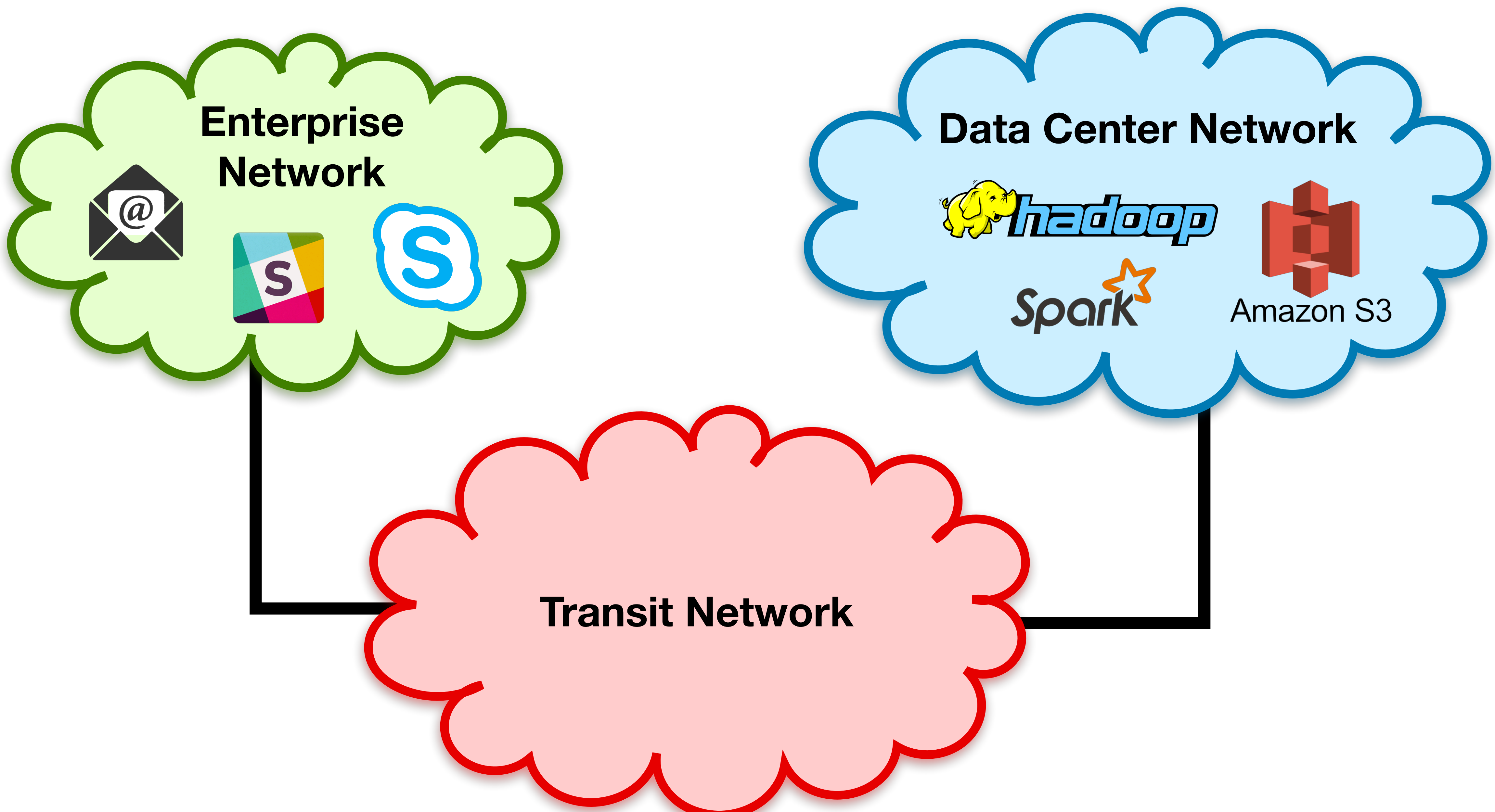
Stateful Programming of High-Speed Network Hardware

Mina Tahmasbi Arashloo

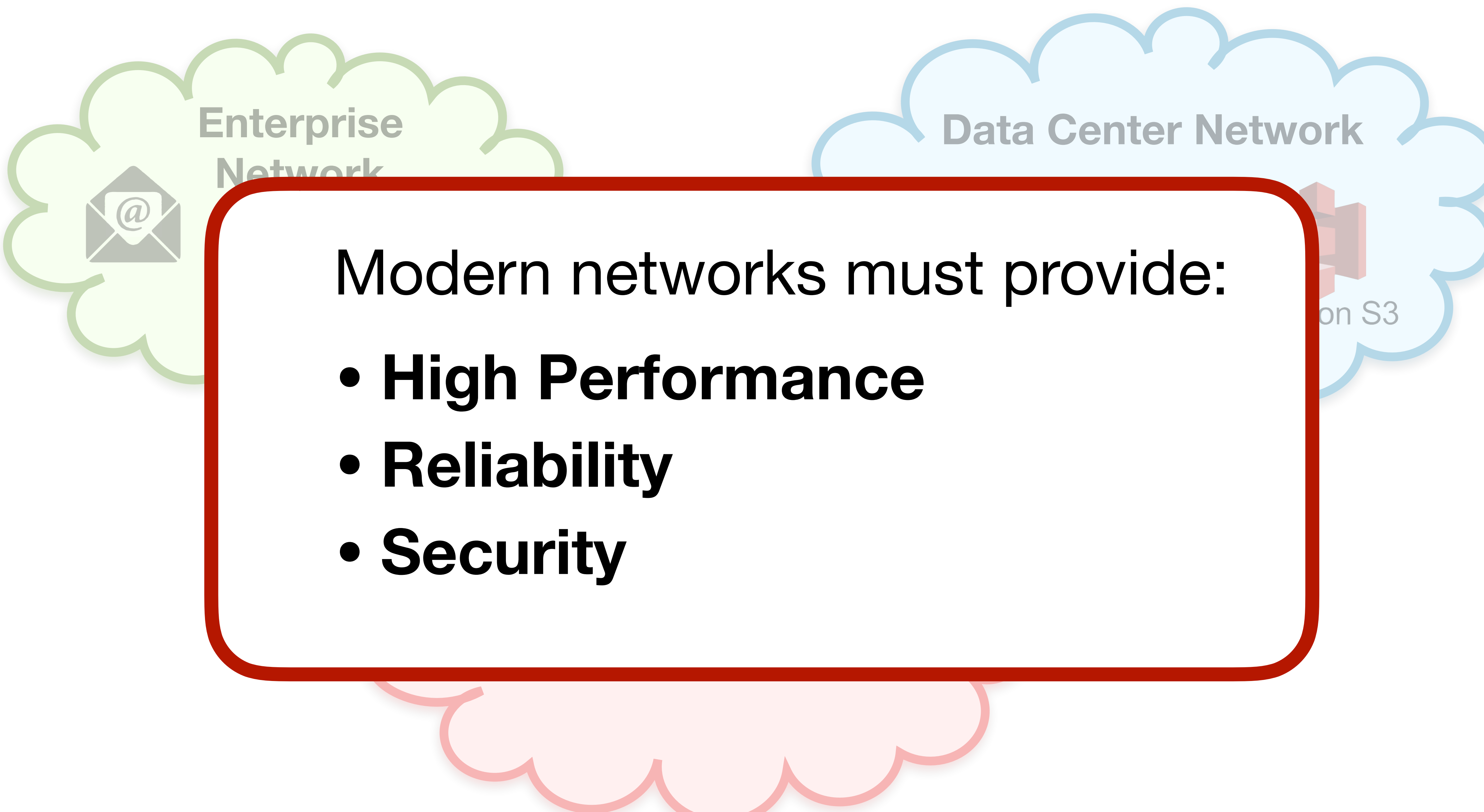
Final Public Oral Presentation

Advisor: Jennifer Rexford
Readers: David Walker, Arvind Krishnamurthy
Examiners: Nick Feamster, Michael Freedman

Networks of Unprecedented Diversity and Scale



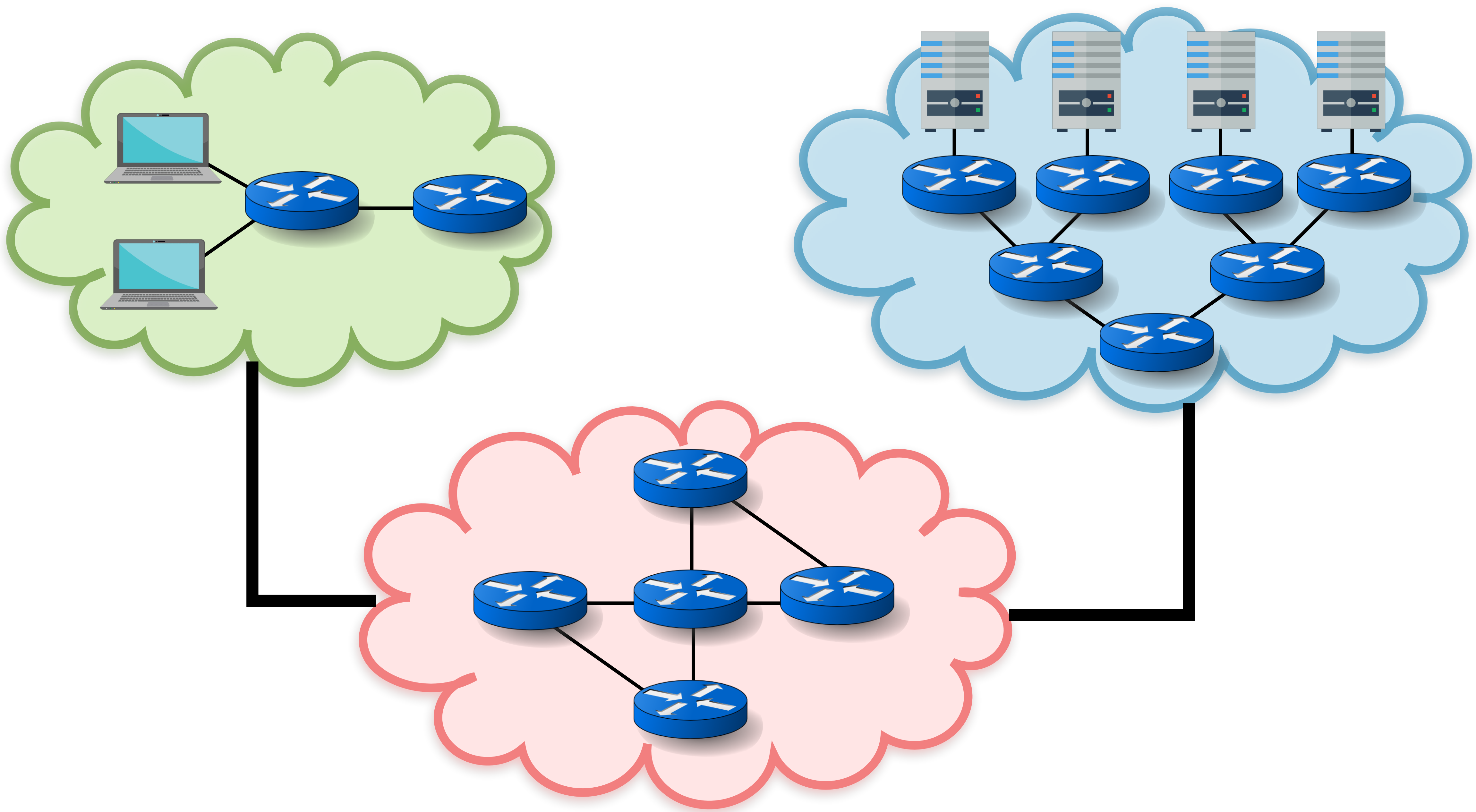
Networks of Unprecedented Diversity and Scale



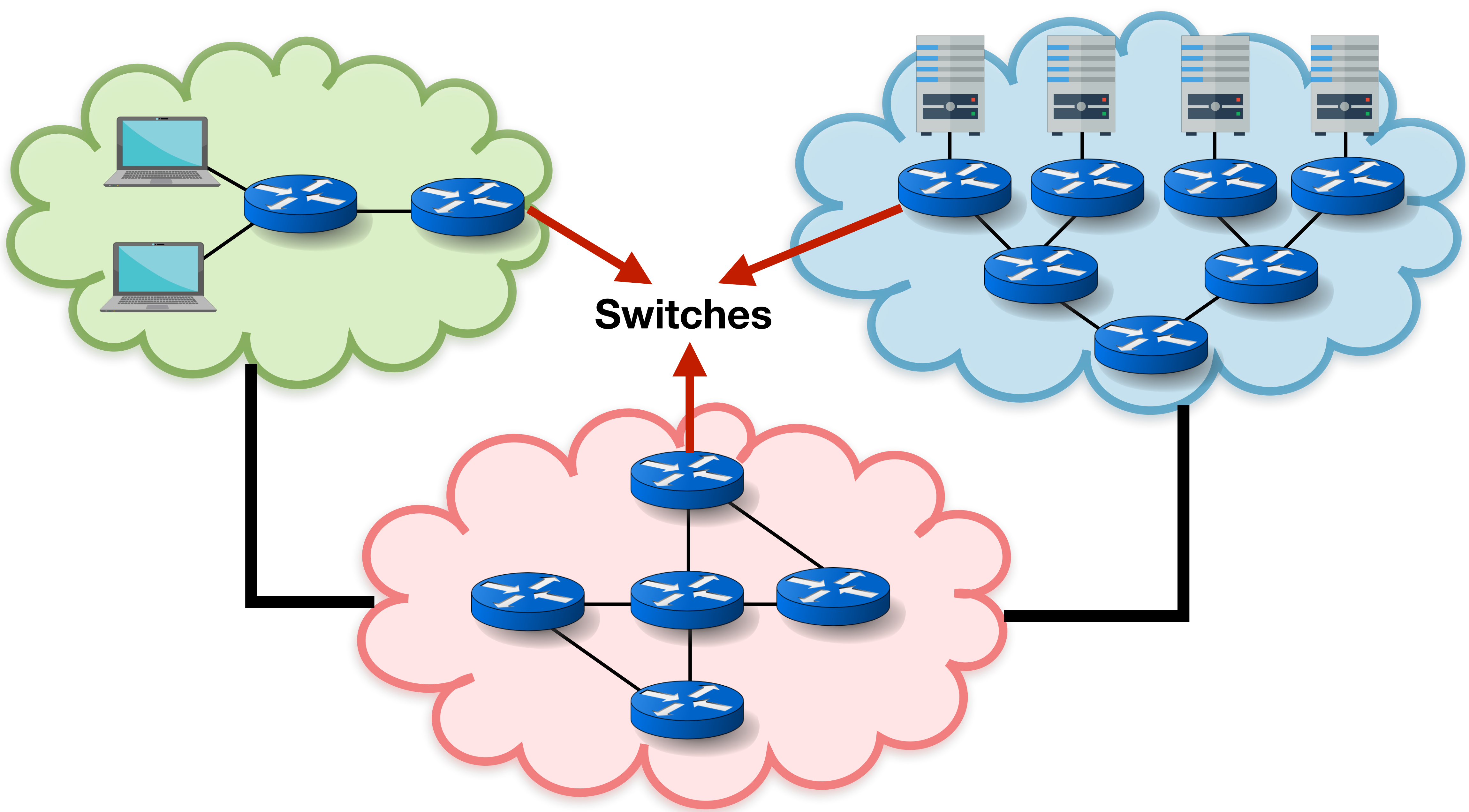
Modern networks must provide:

- **High Performance**
- **Reliability**
- **Security**

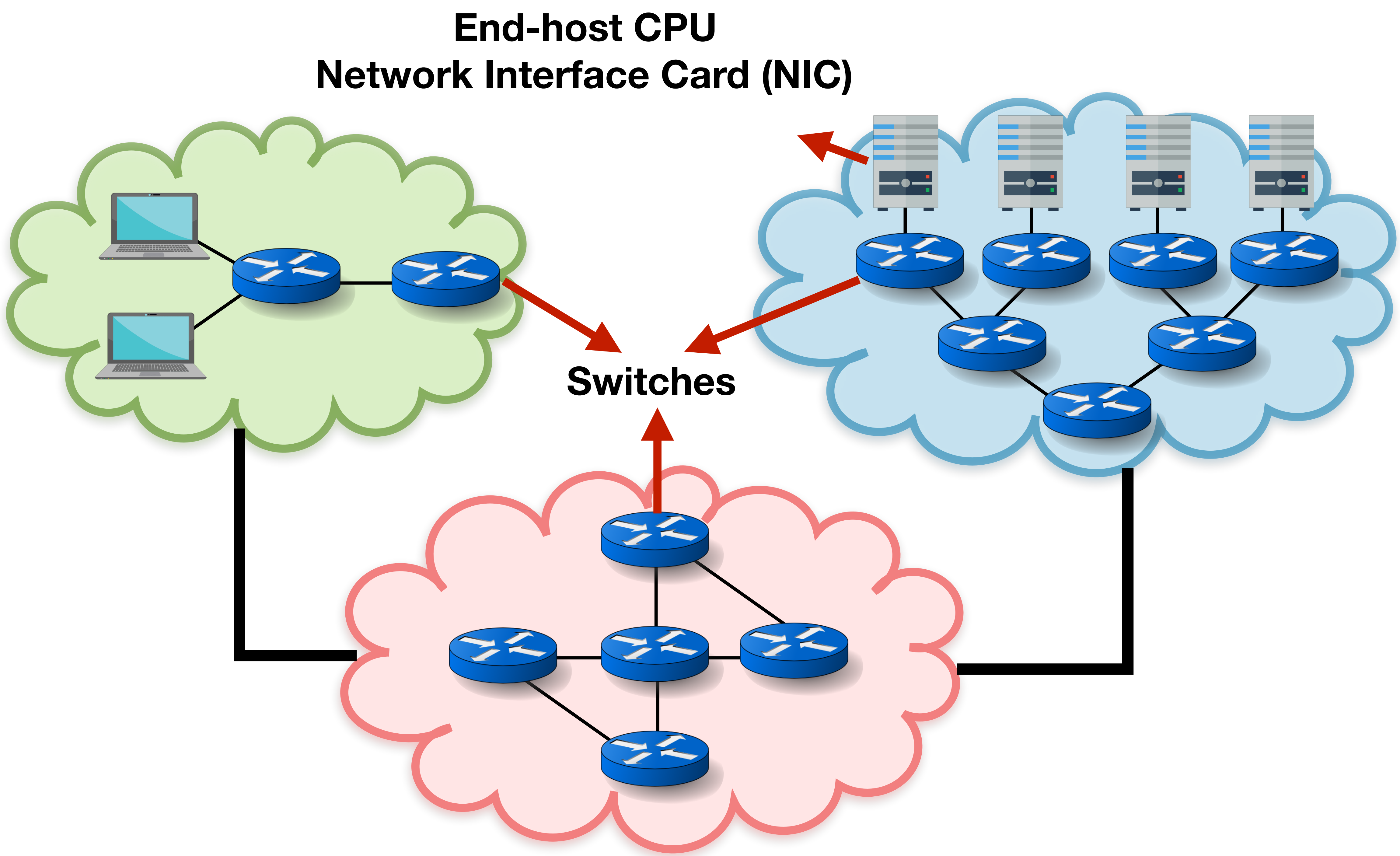
The Evolution of Network Hardware



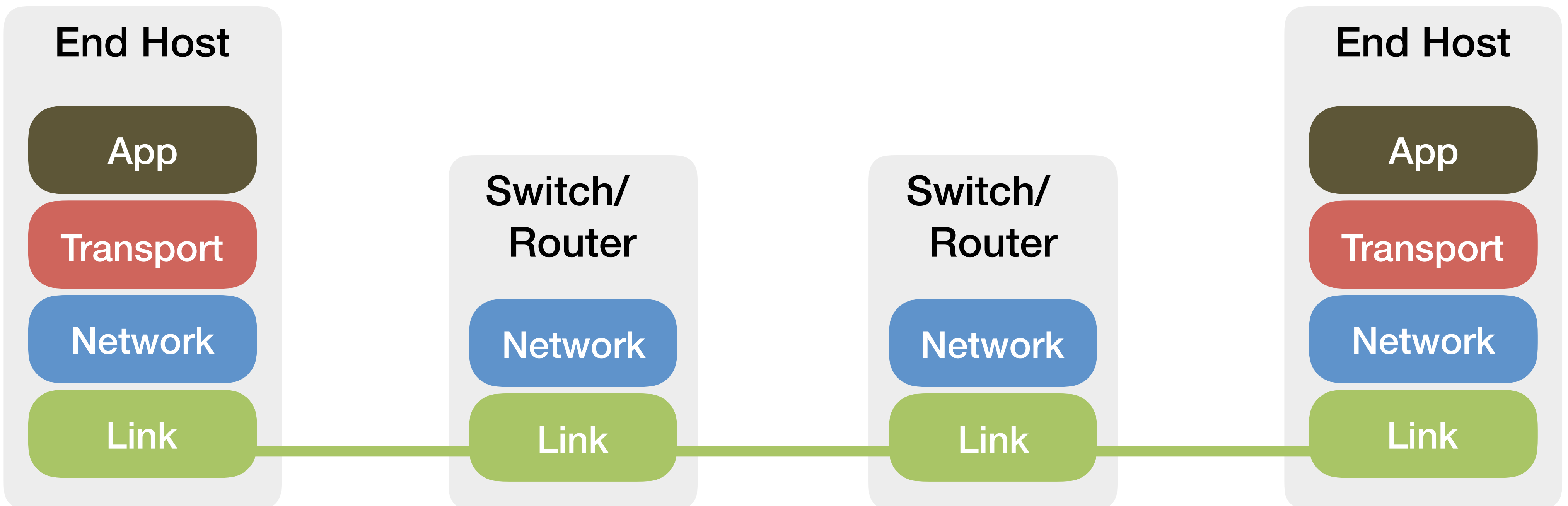
The Evolution of Network Hardware



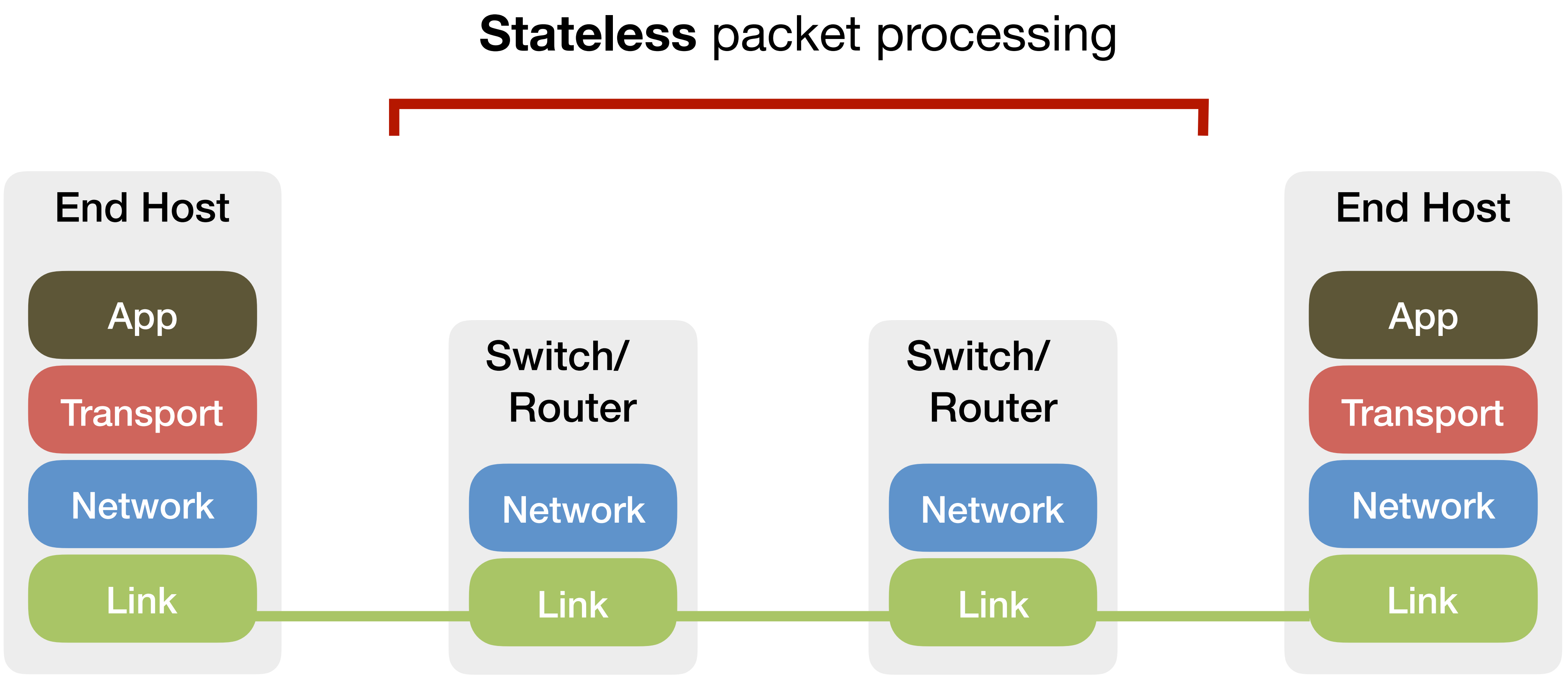
The Evolution of Network Hardware



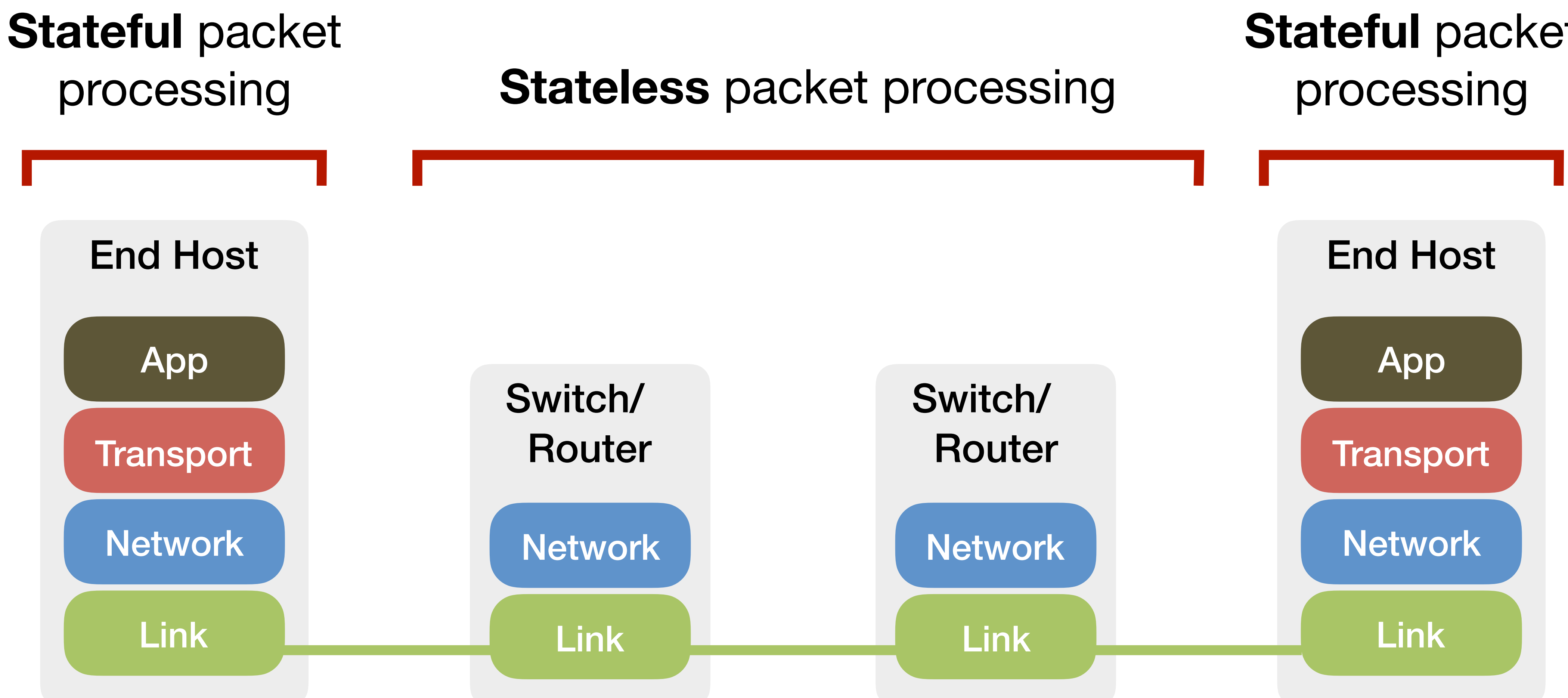
Early Networks: Stateful Edge, Stateless Core



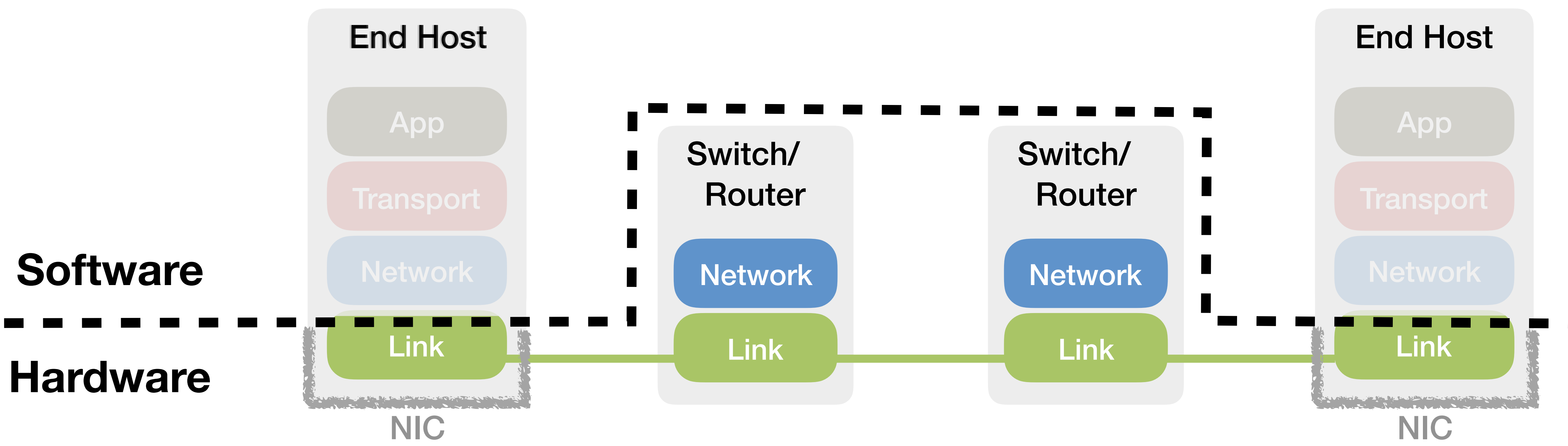
Early Networks: Stateful Edge, Stateless Core



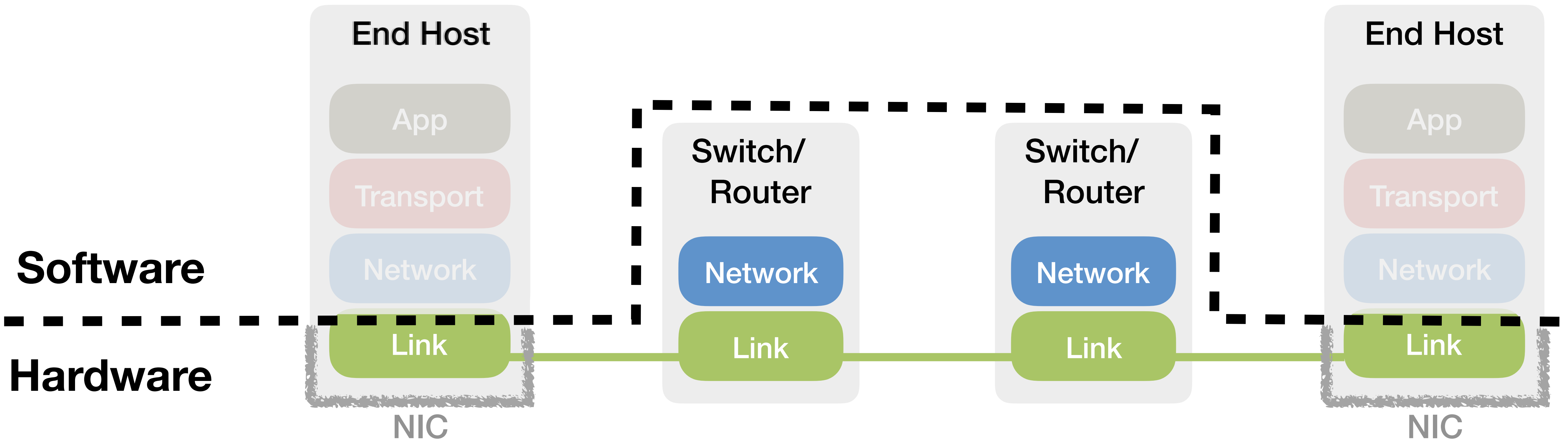
Early Networks: Stateful Edge, Stateless Core



Early Networks: Stateful Edge, Stateless Core



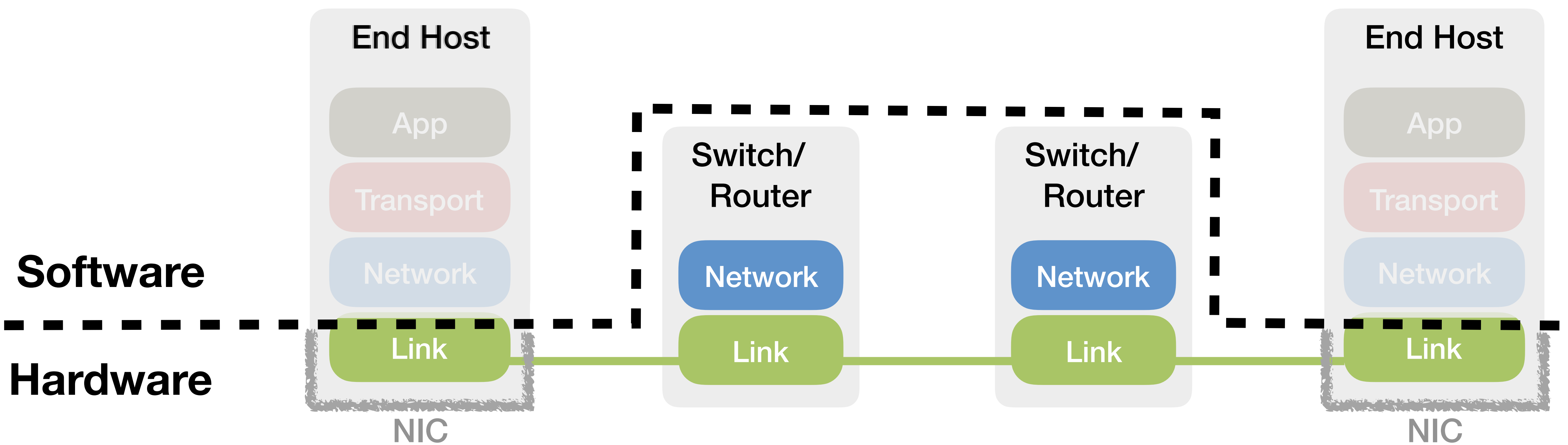
Early Networks: Stateful Edge, Stateless Core



Network Hardware

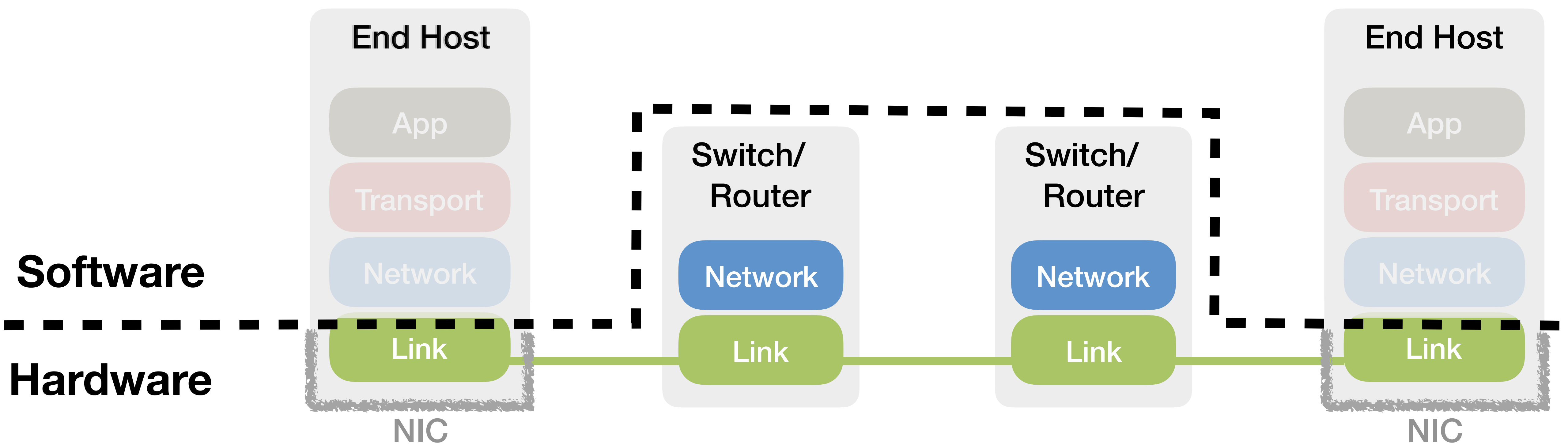
High-Speed

The Need for Stateful Processing in Hardware



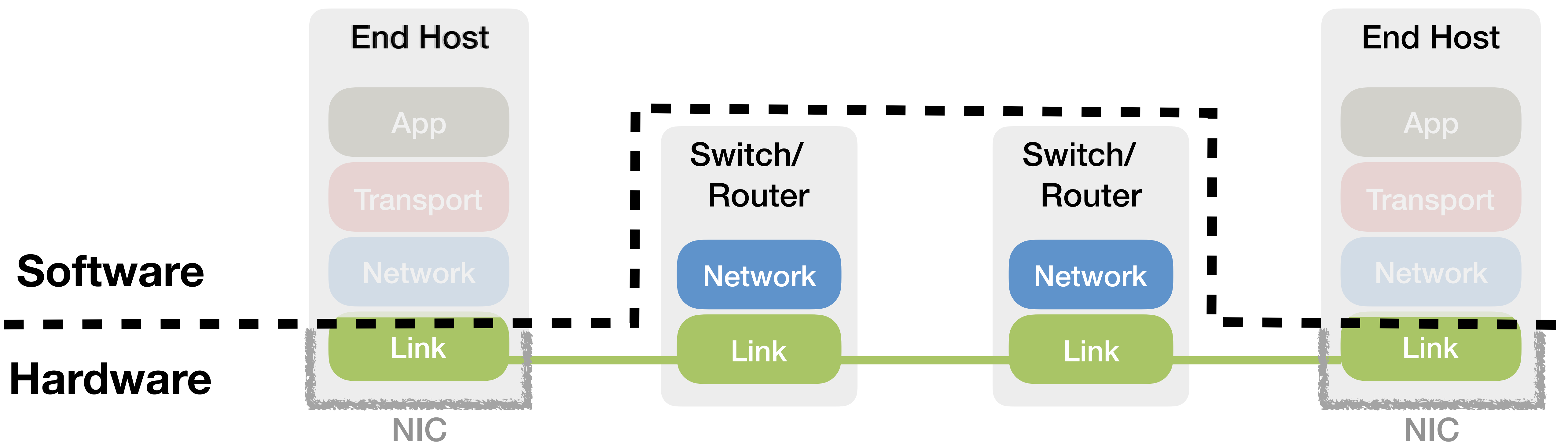
Network Hardware { High-Speed

The Need for Stateful Processing in Hardware

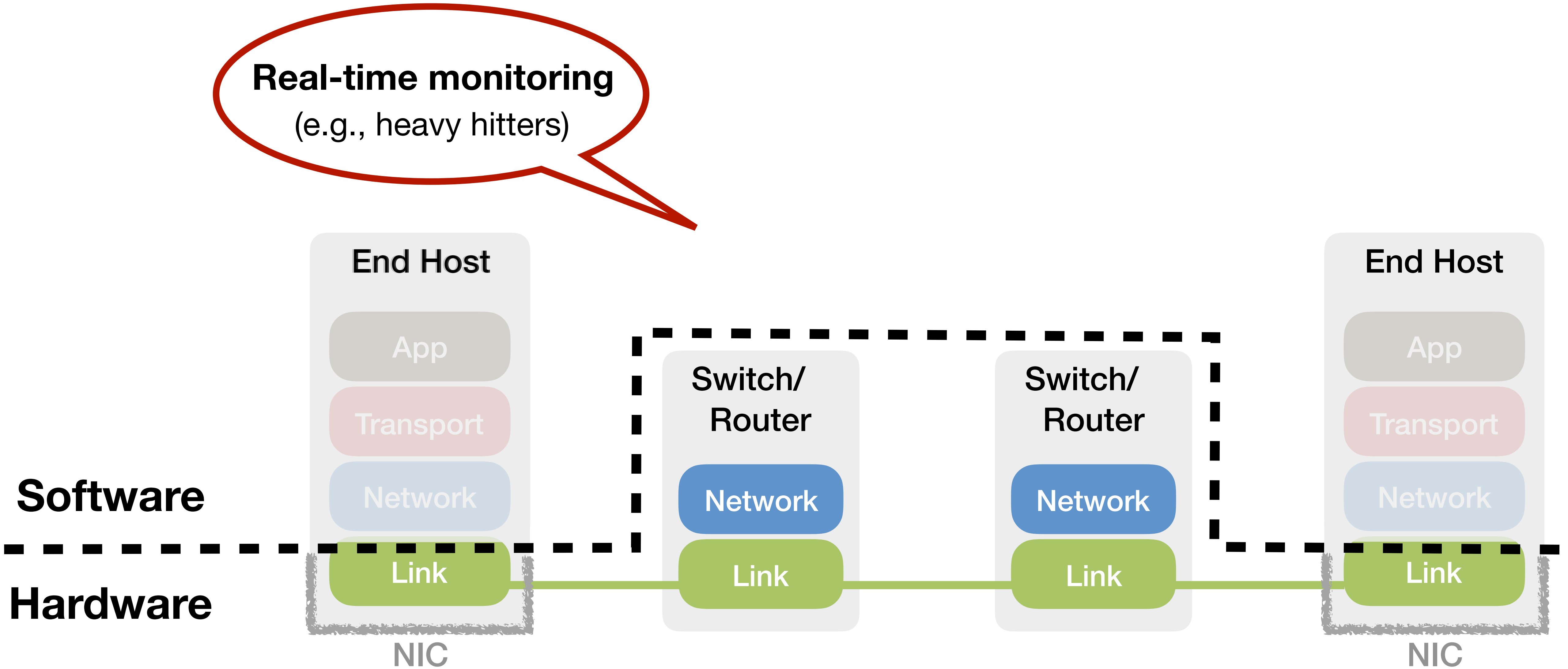


Network Hardware {
High-Speed
Stateful Packet Processing??

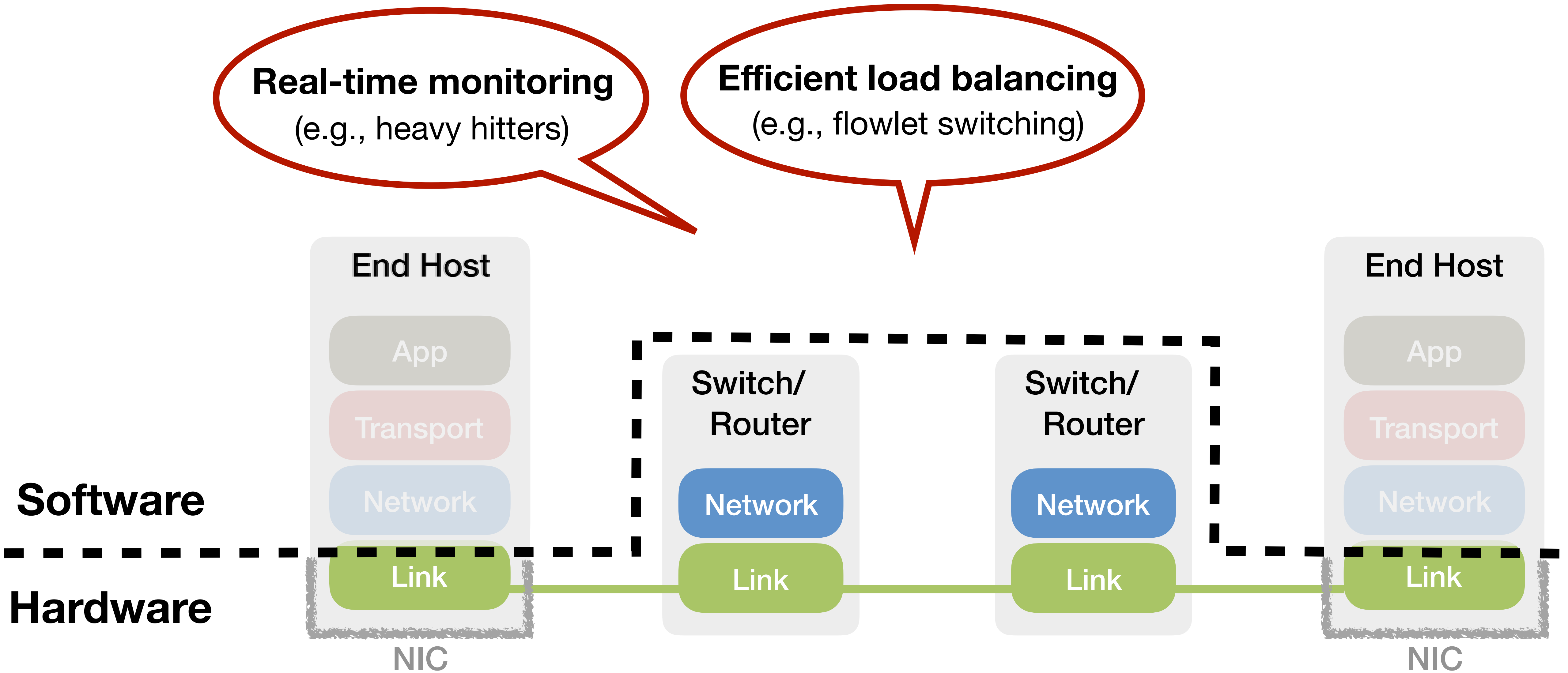
Trend #1: In-Network Stateful Processing



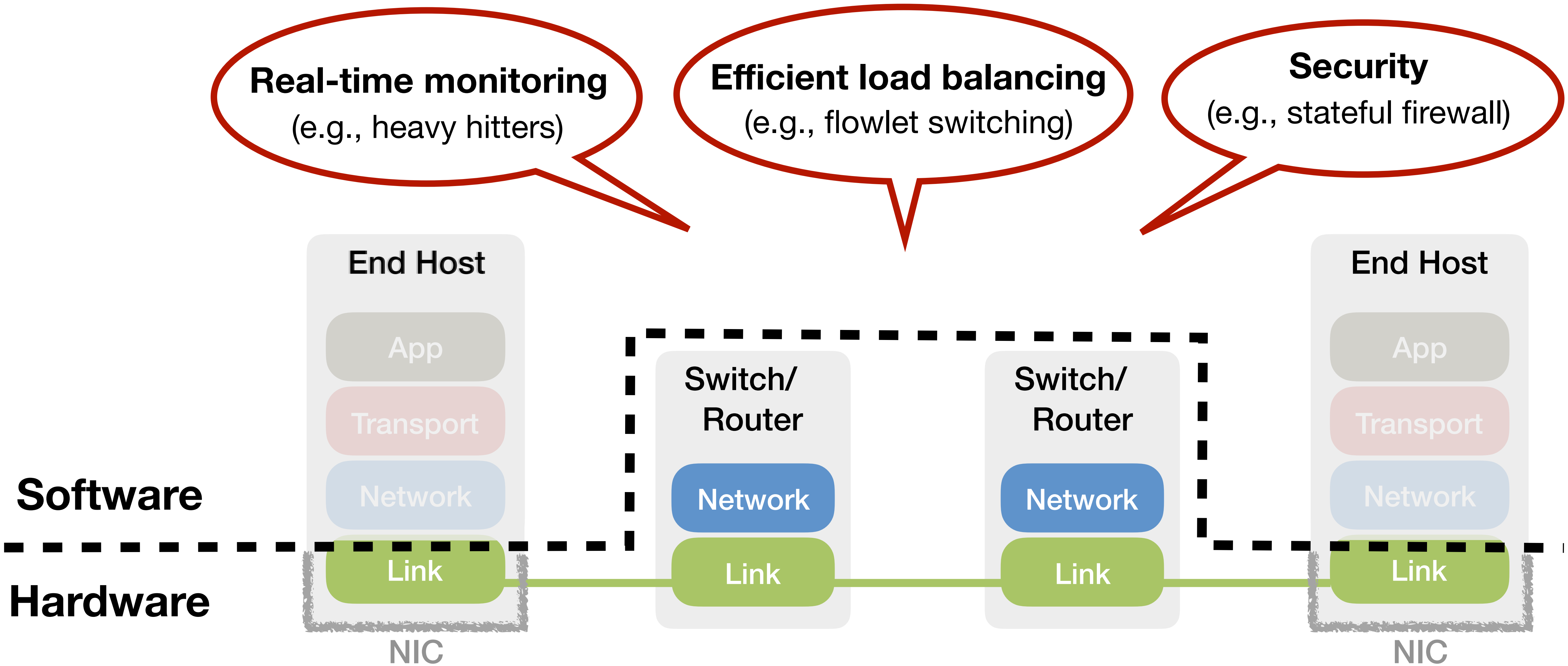
Trend #1: In-Network Stateful Processing



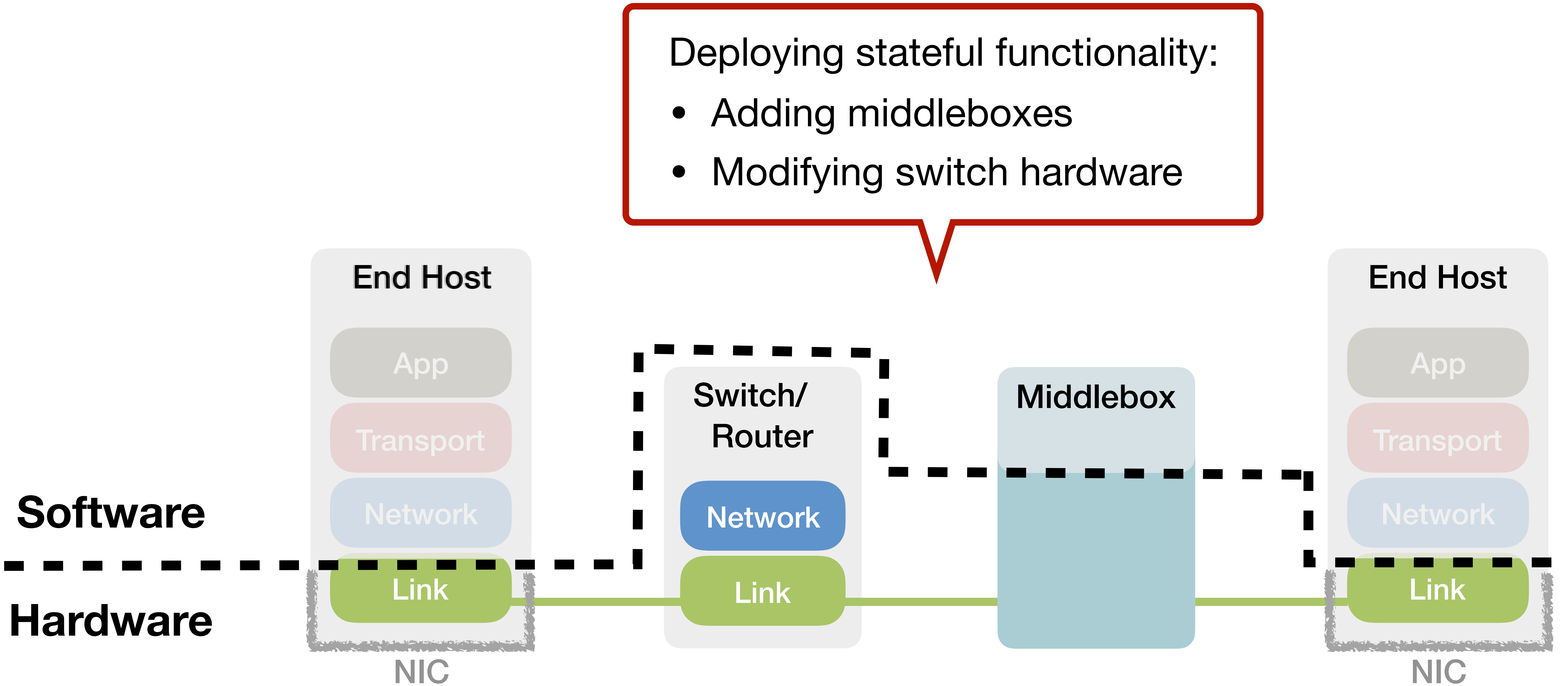
Trend #1: In-Network Stateful Processing



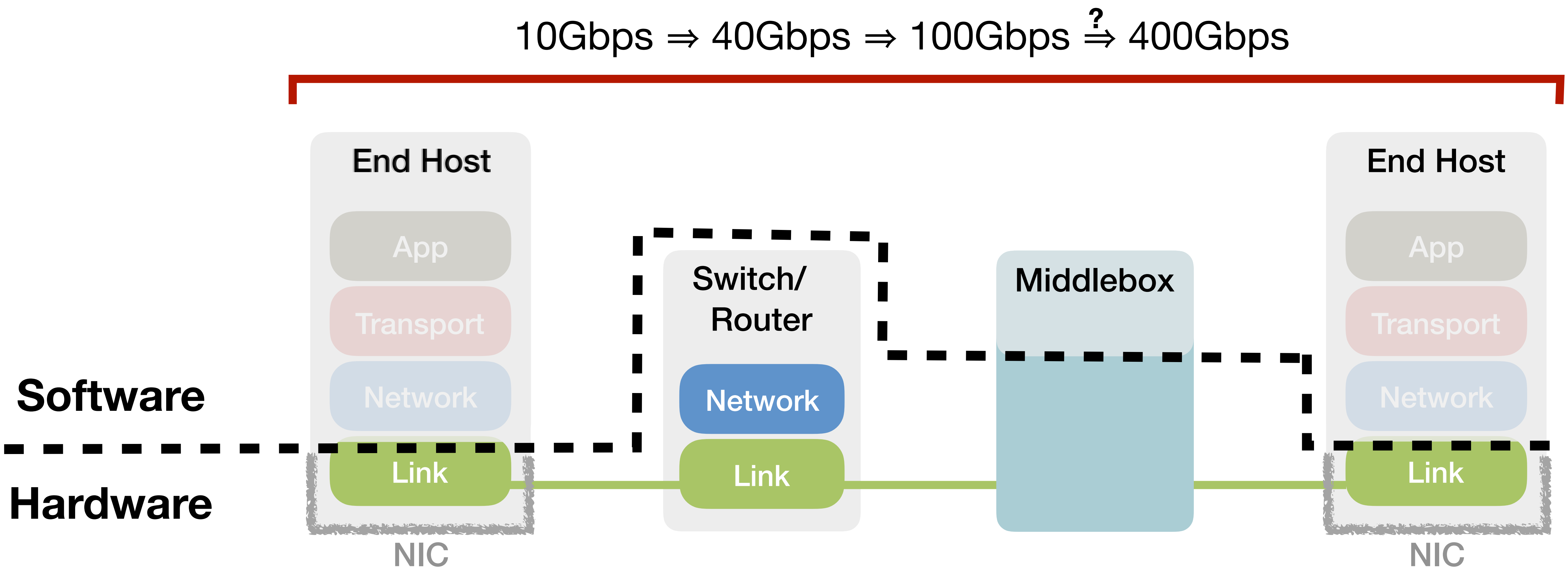
Trend #1: In-Network Stateful Processing



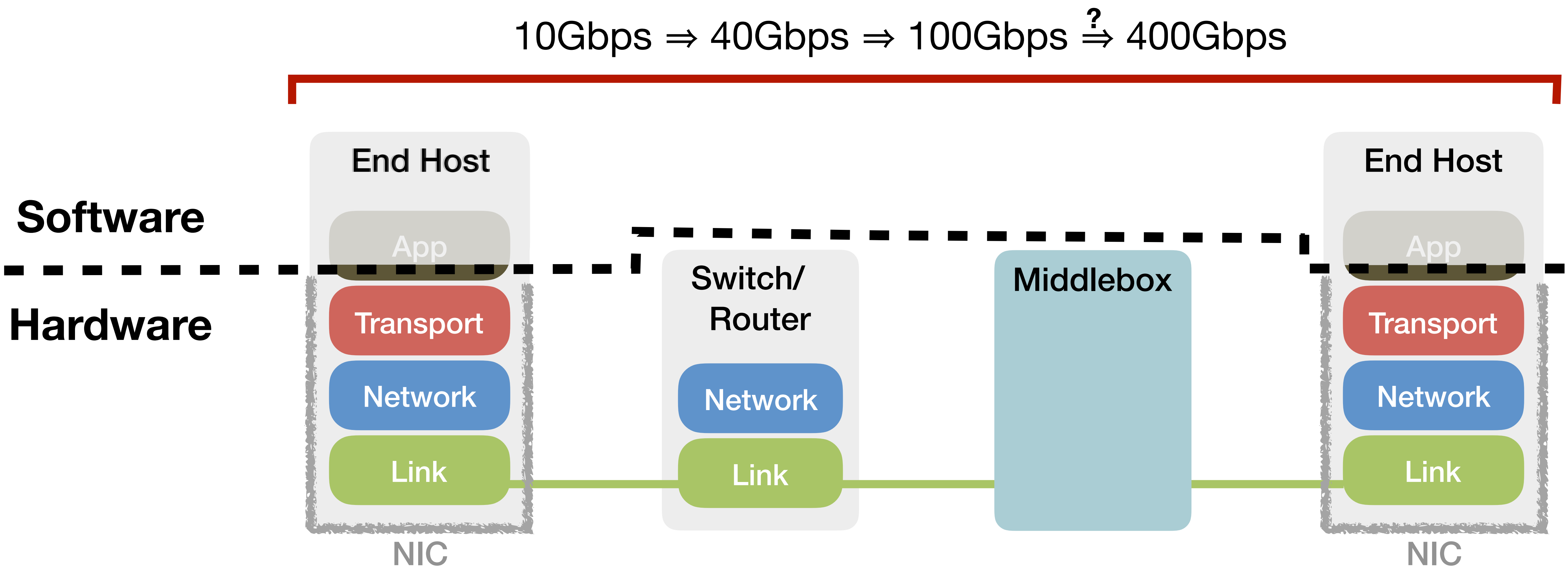
Trend #1: In-Network Stateful Processing



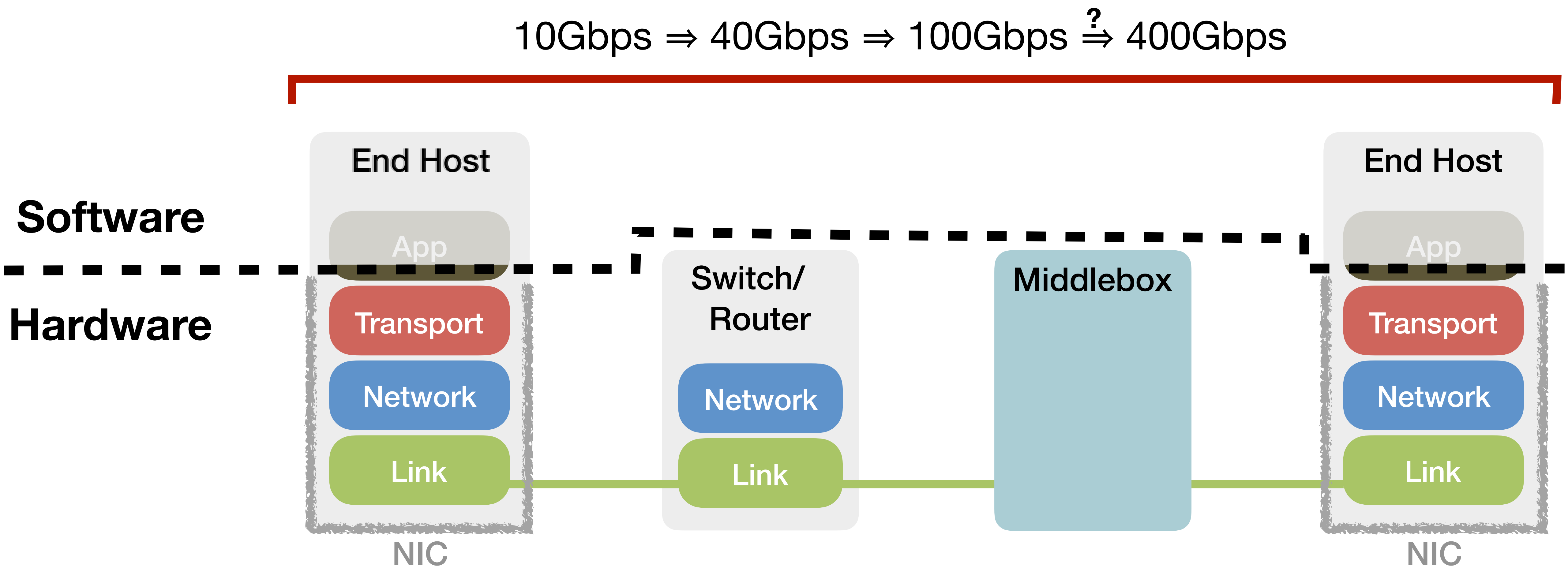
Trend #2: Increasing Link Speeds



Trend #2: Increasing Link Speeds

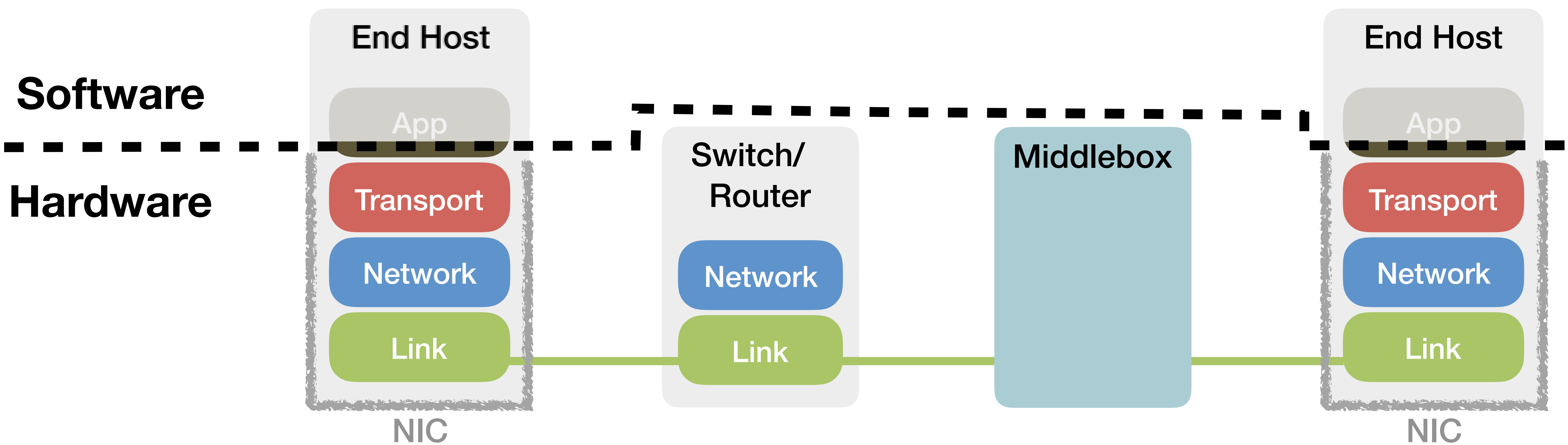


Trend #2: Increasing Link Speeds

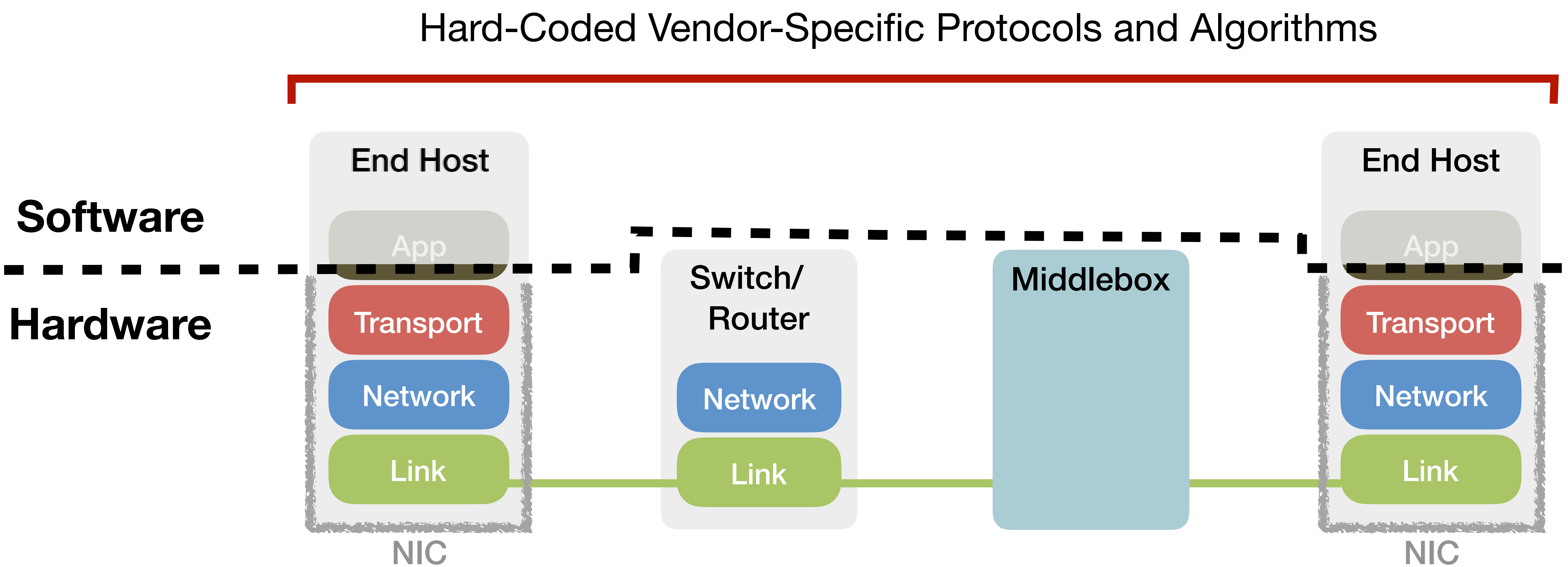


Network Hardware
- High-Speed
- Stateful Packet Processing

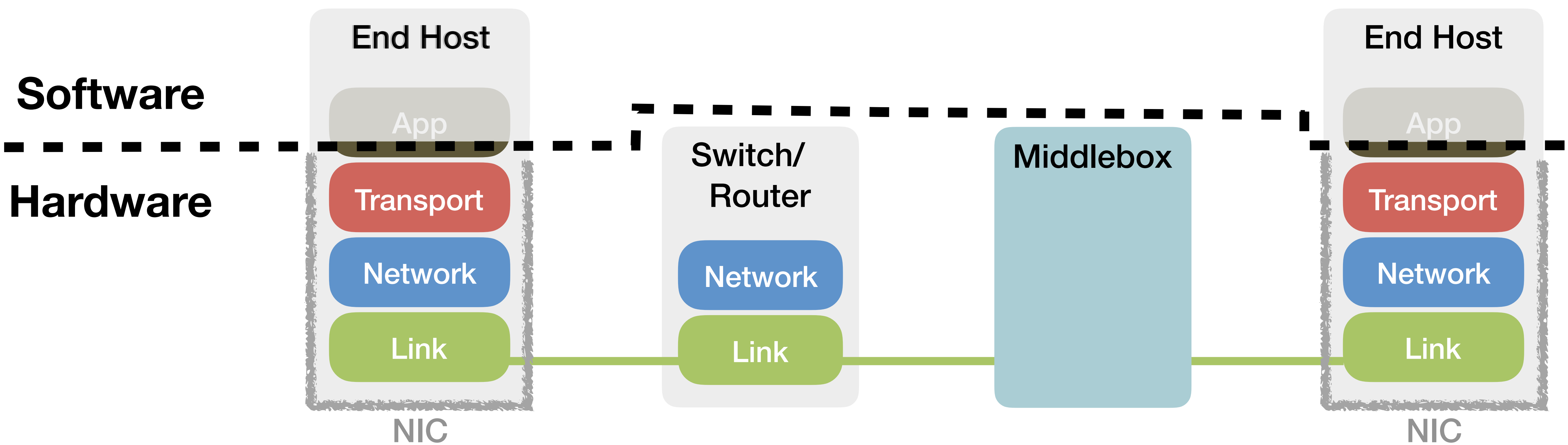
What about Flexibility?



What about Flexibility?



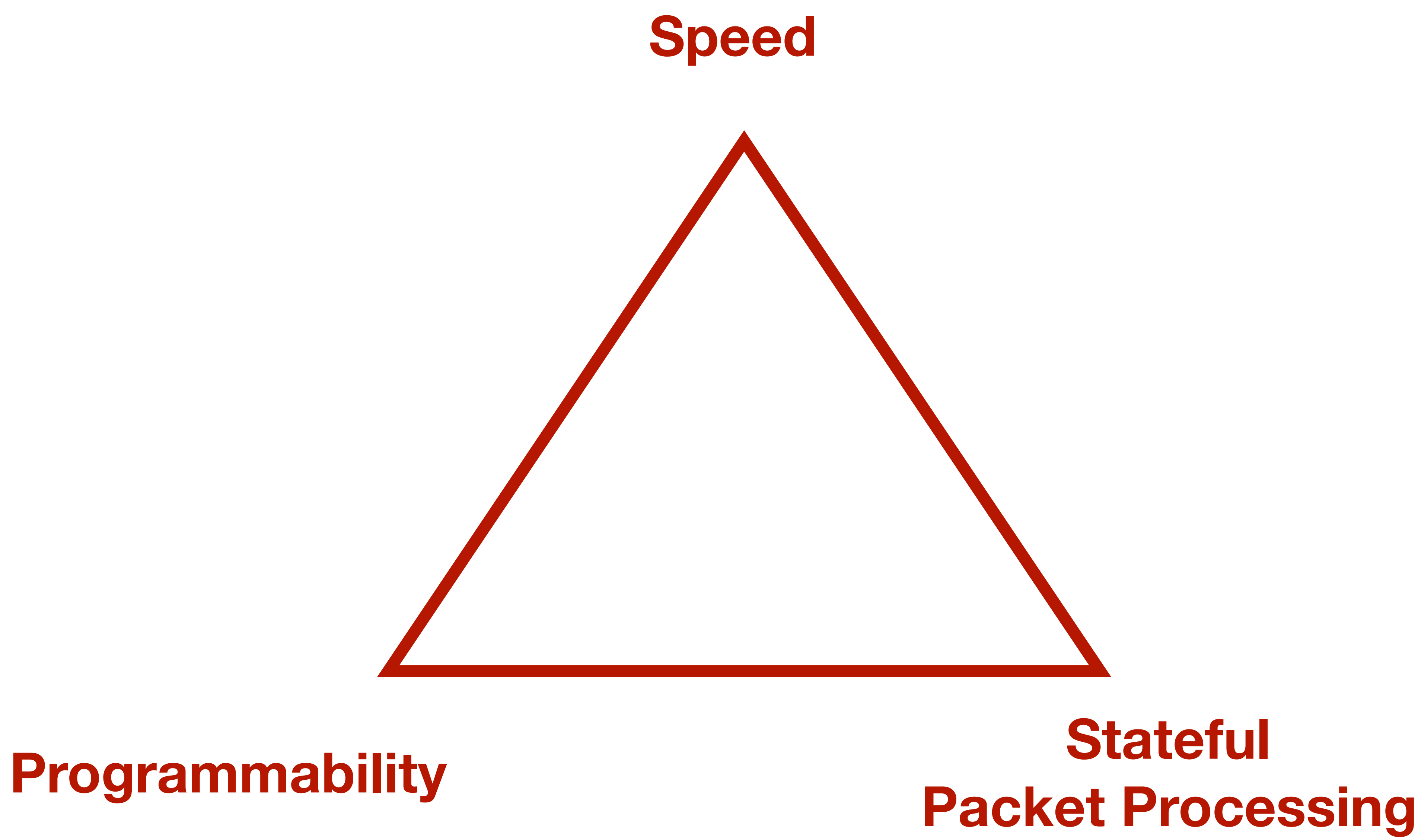
Requirements of Today's Network Hardware



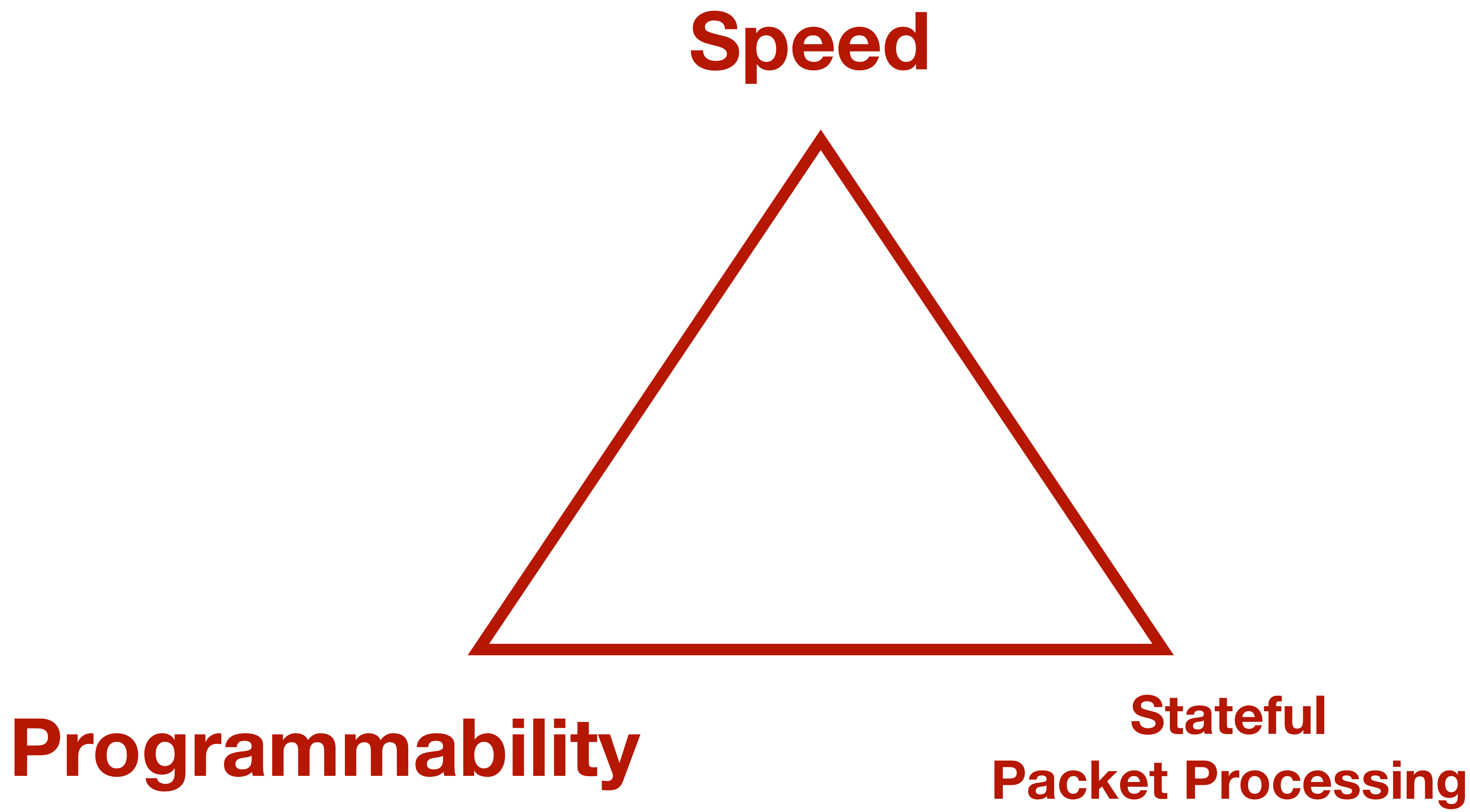
Network Hardware

- High-Speed
- Stateful Packet Processing
- Programmable

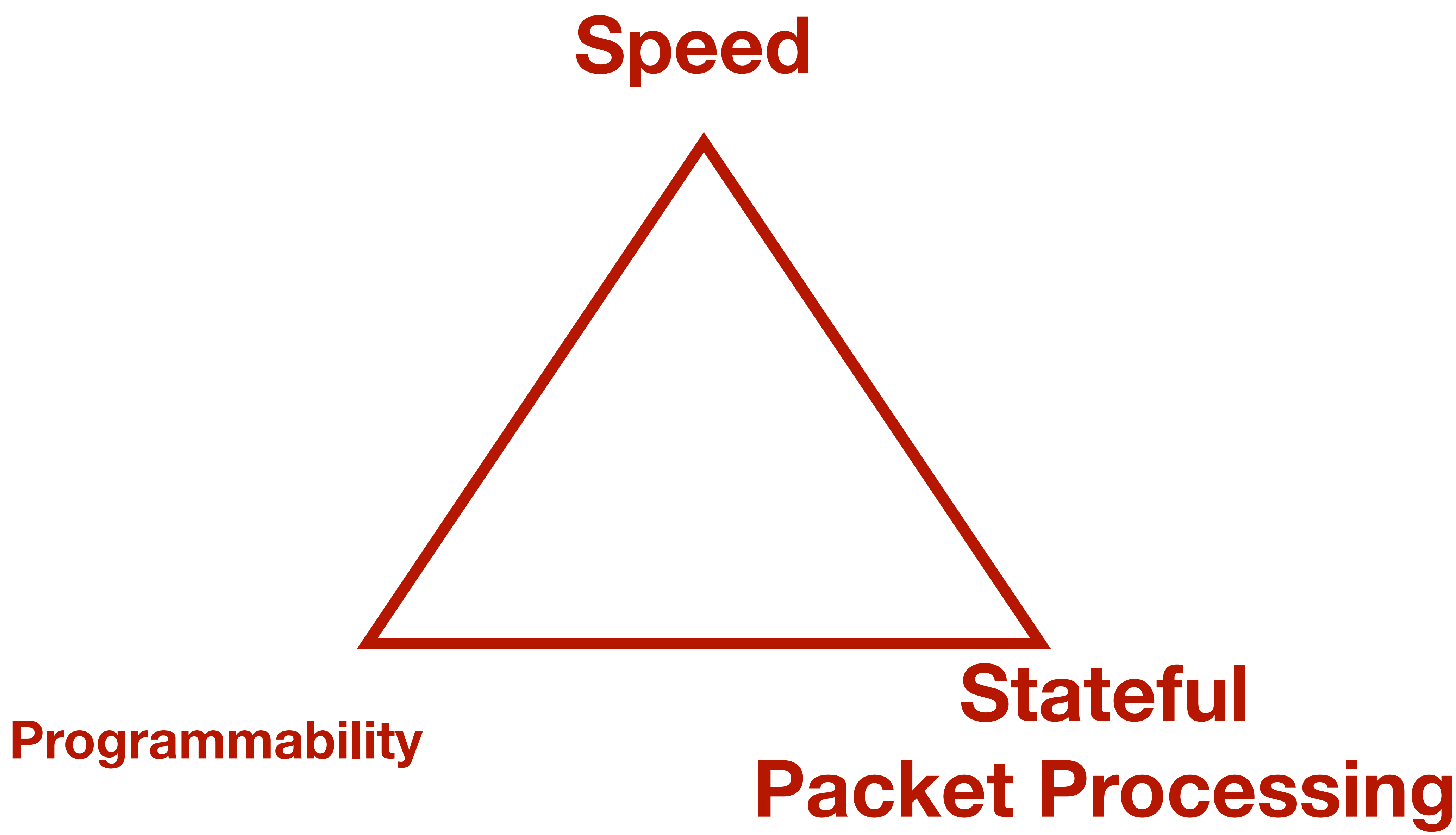
Network Hardware Design Space



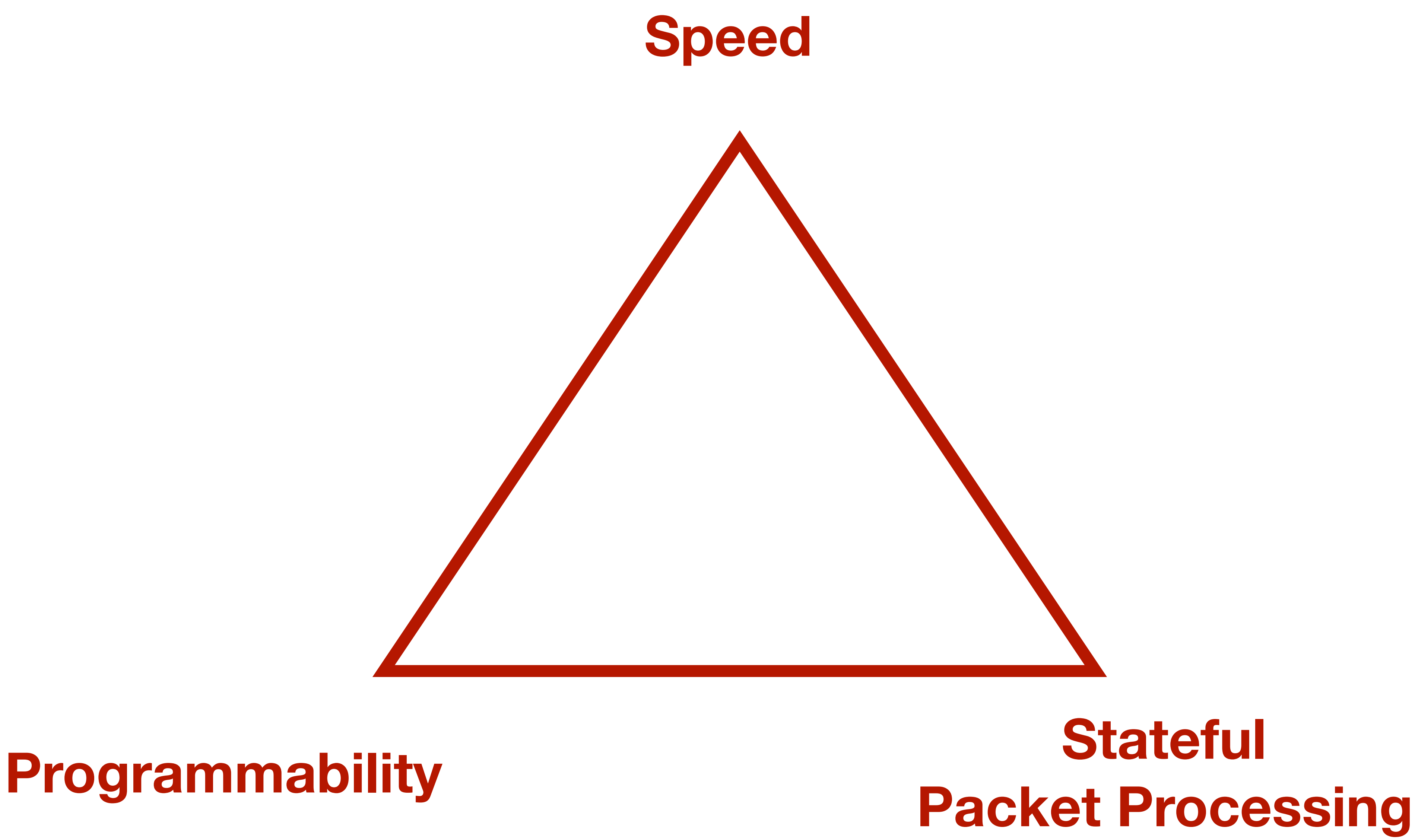
Network Hardware Design Space



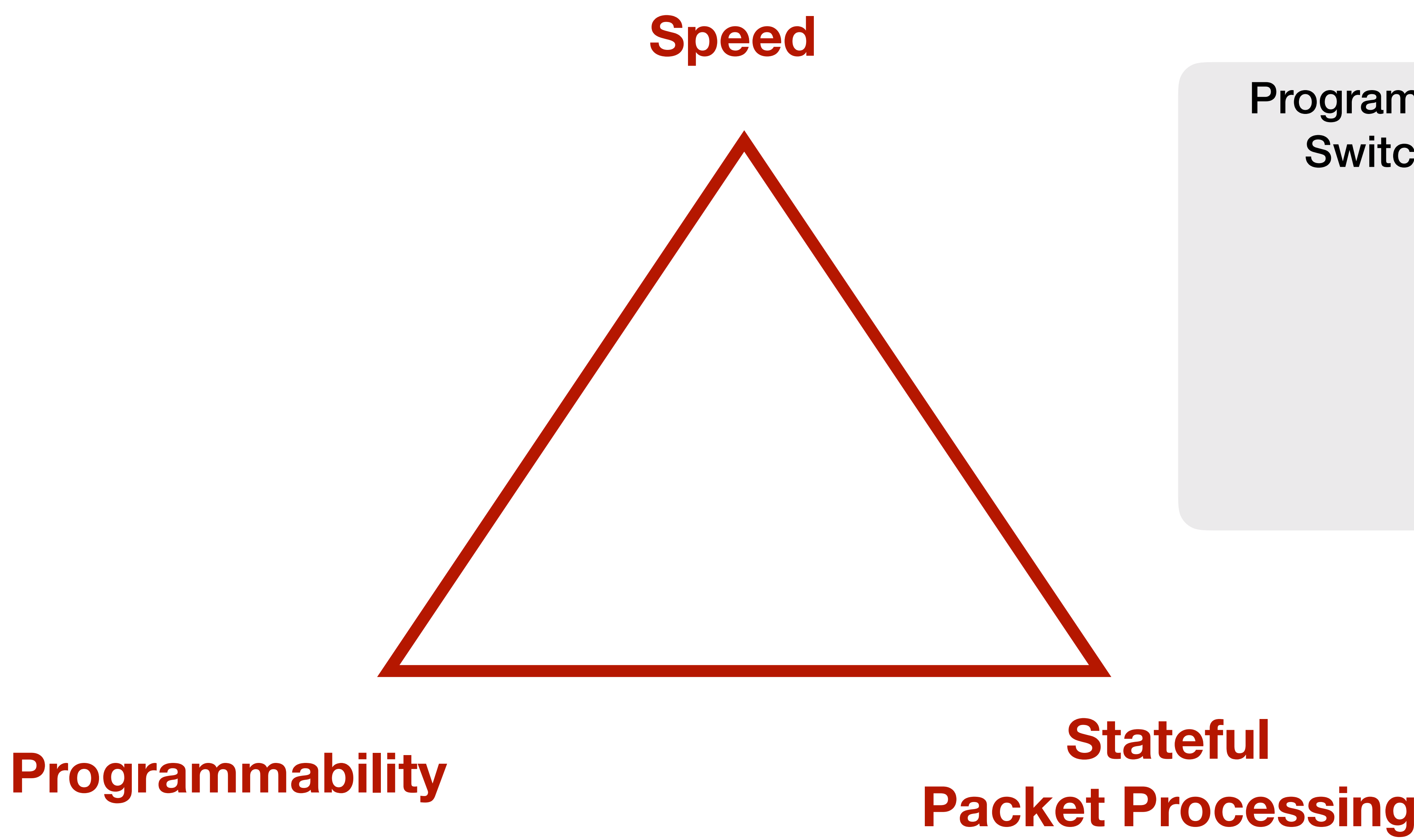
Network Hardware Design Space



Network Hardware Design Space

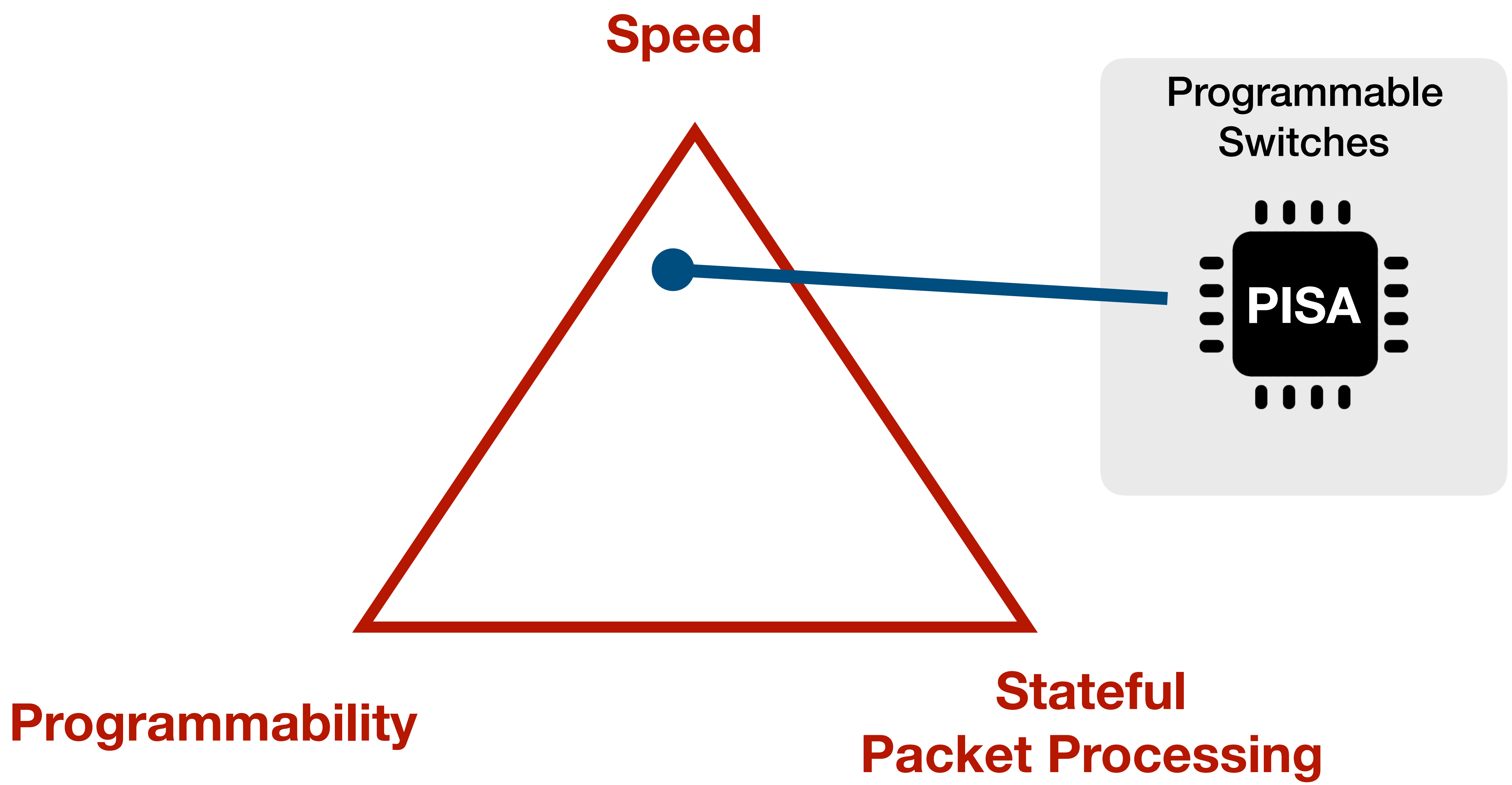


Network Hardware Design Space

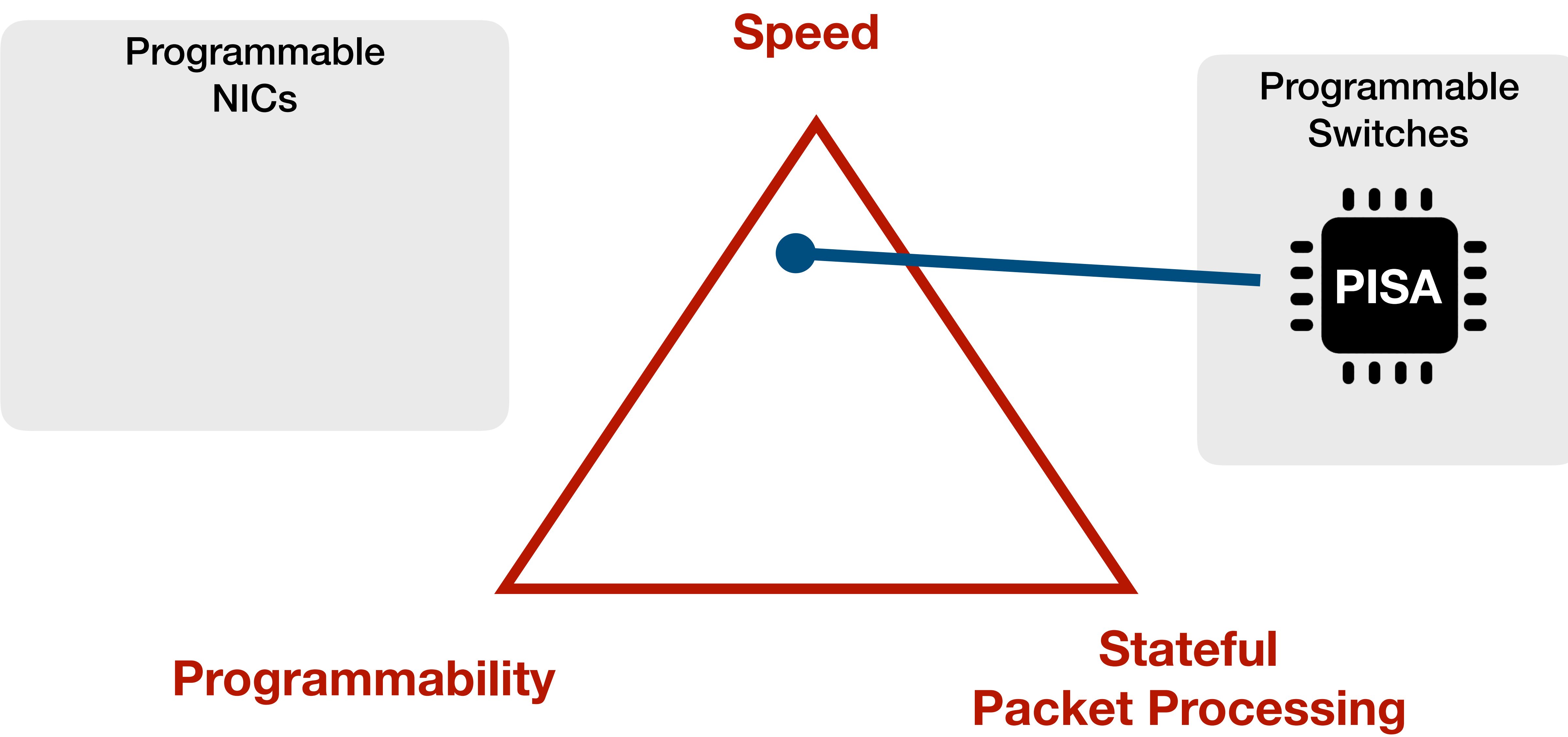


Programmable
Switches

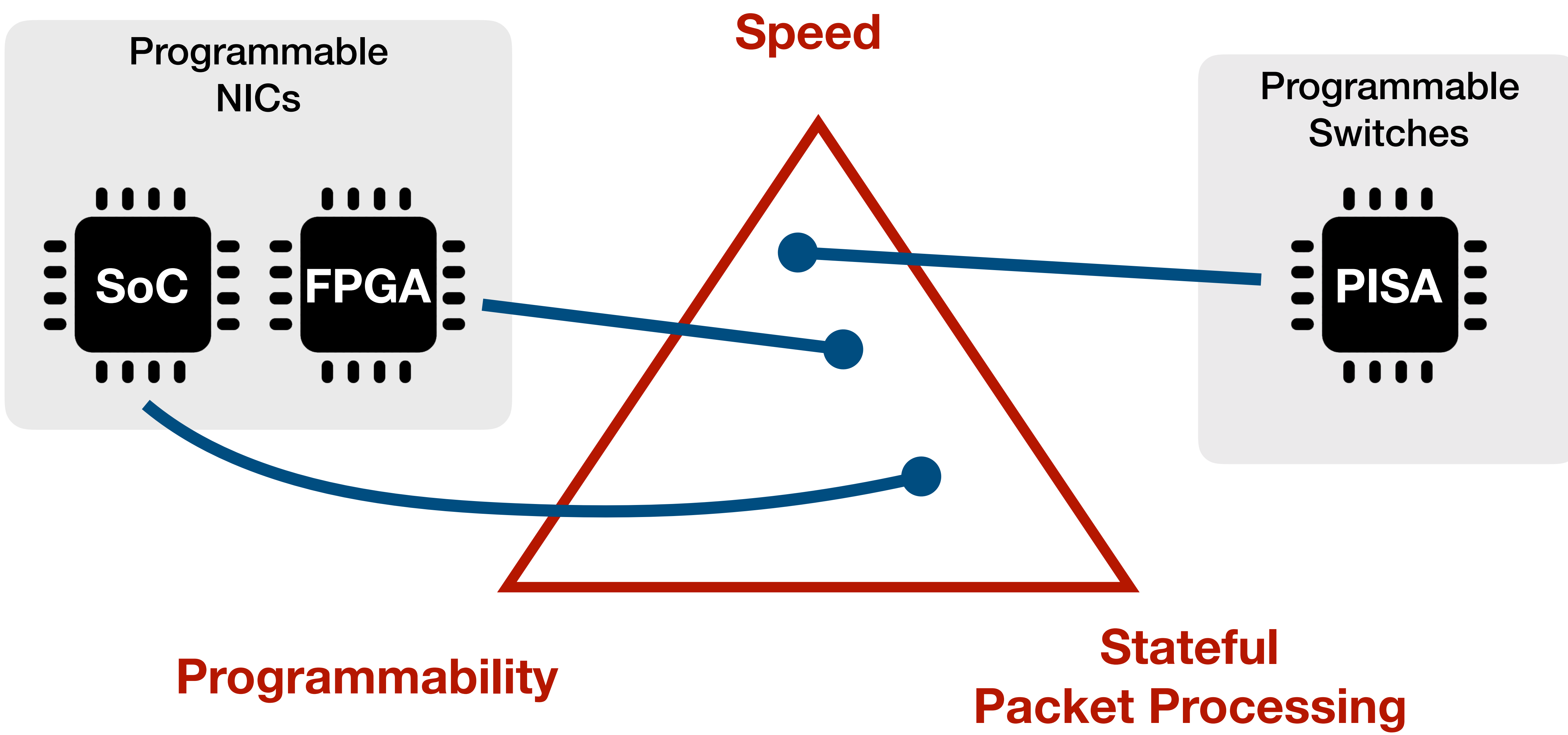
Network Hardware Design Space



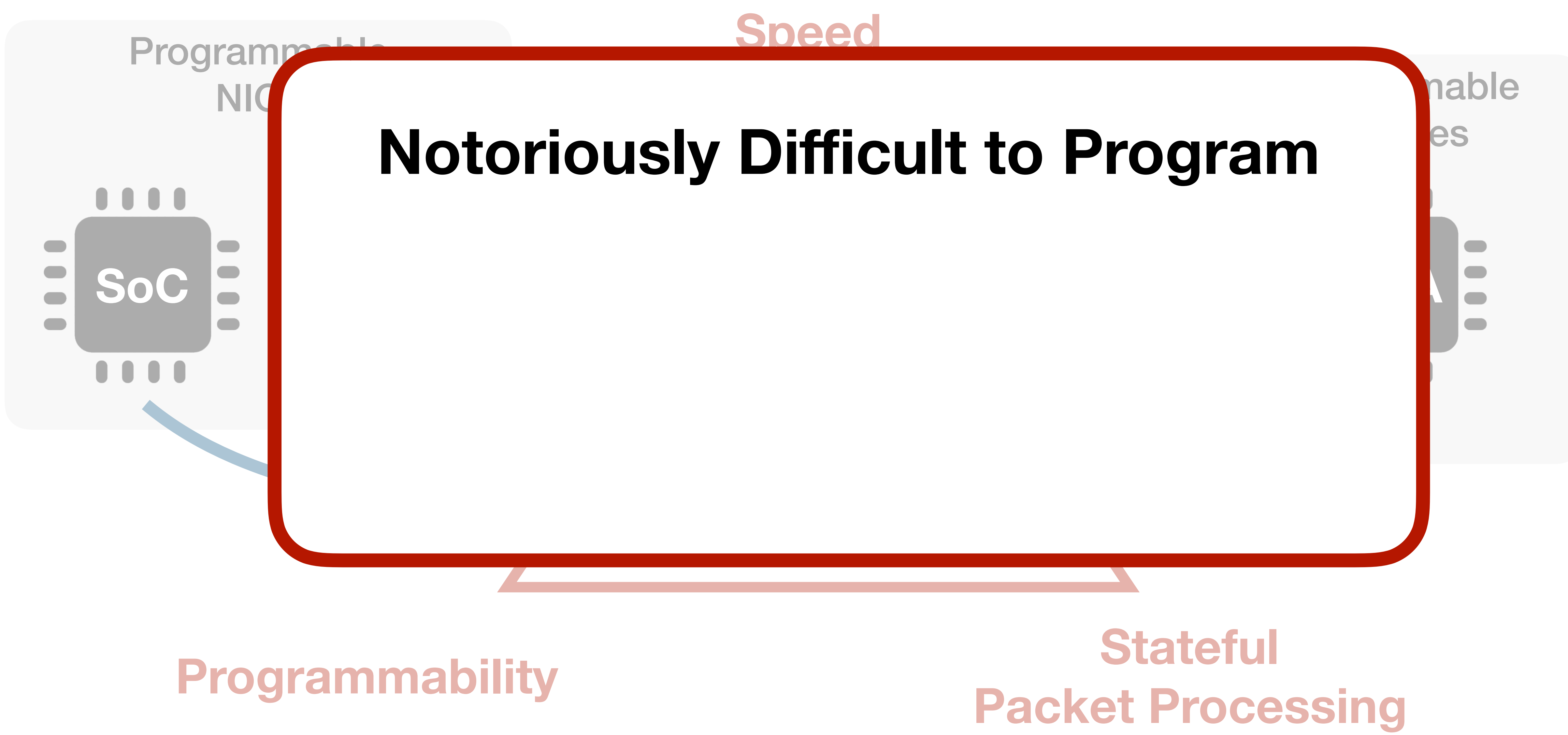
Network Hardware Design Space



Network Hardware Design Space



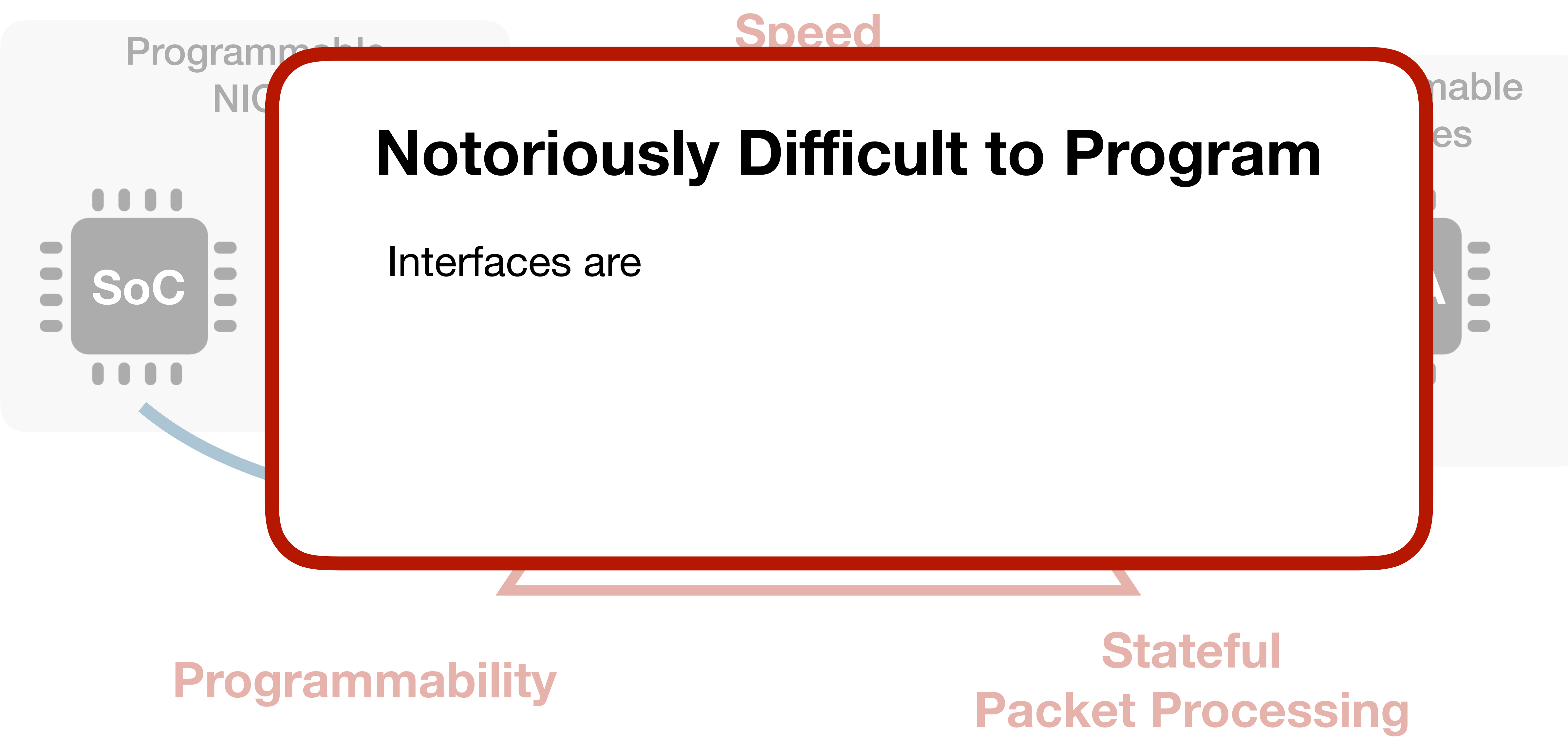
Network Hardware Design Space



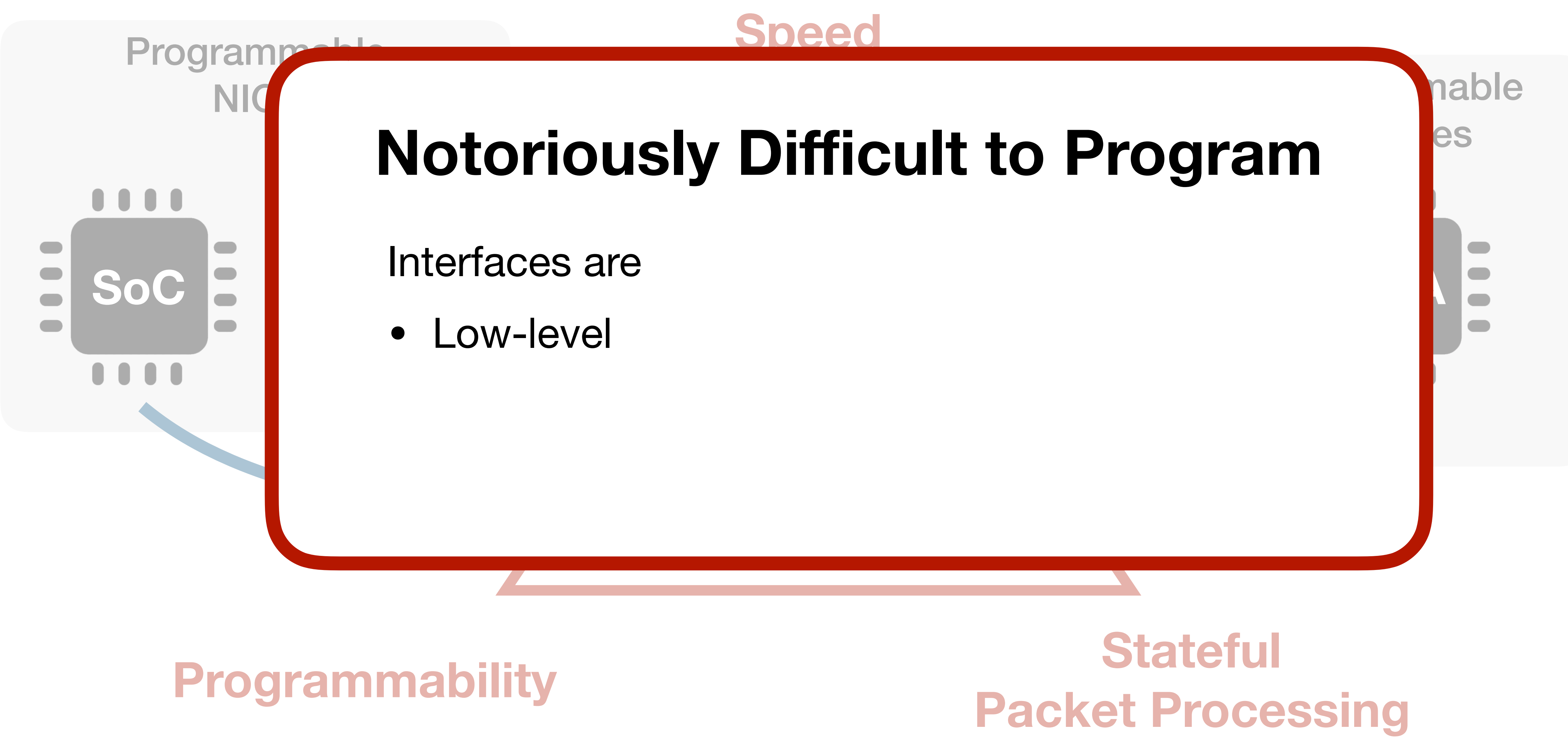
Network Hardware Design Space

Notoriously Difficult to Program

Interfaces are



Network Hardware Design Space

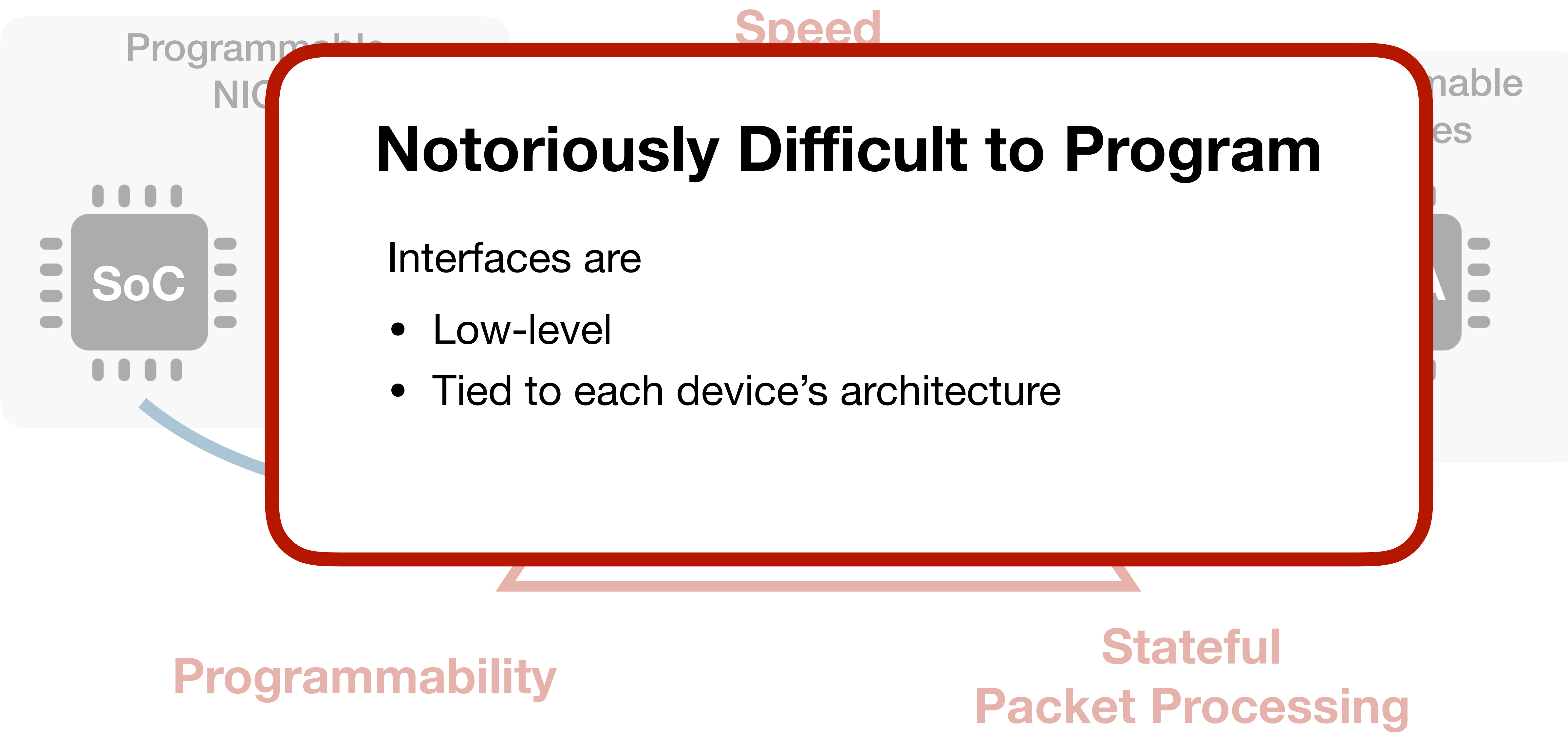


Notoriously Difficult to Program

Interfaces are

- Low-level

Network Hardware Design Space

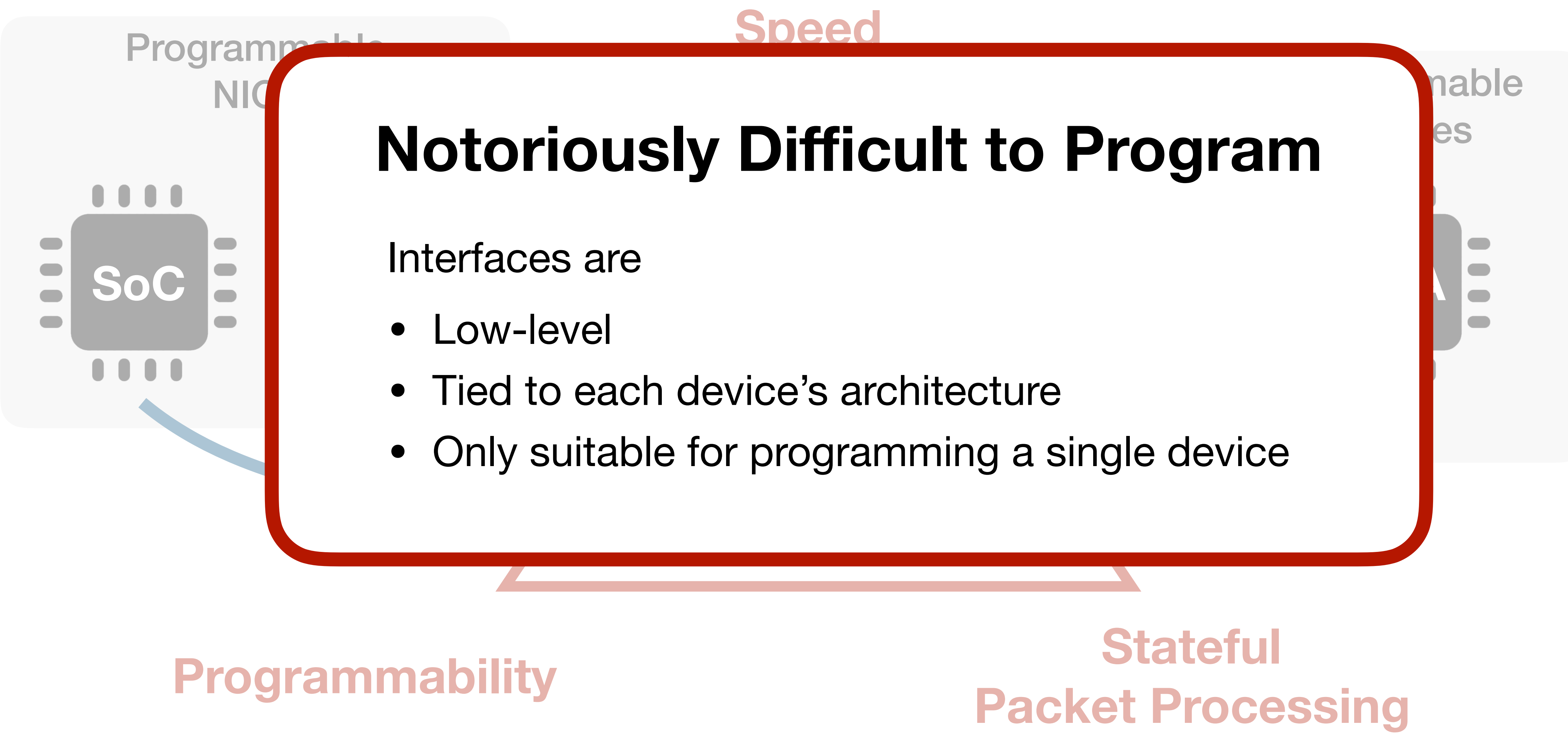


Notoriously Difficult to Program

Interfaces are

- Low-level
- Tied to each device's architecture

Network Hardware Design Space



Notoriously Difficult to Program

Interfaces are

- Low-level
- Tied to each device’s architecture
- Only suitable for programming a single device

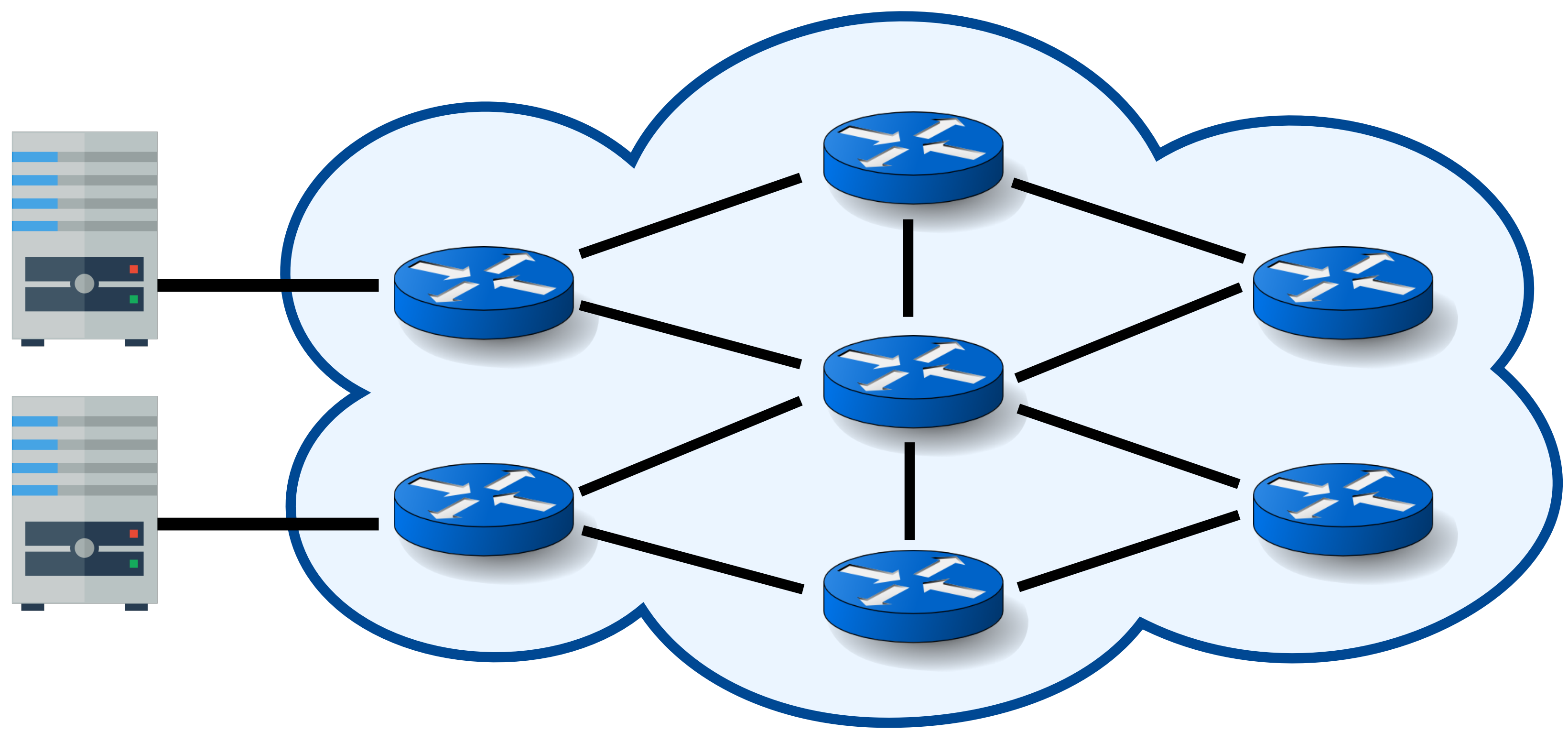
This Dissertation

Design and implementation of

***modular and high-level** programming abstractions*

*for **stateful** programming of **high-speed** network hardware*

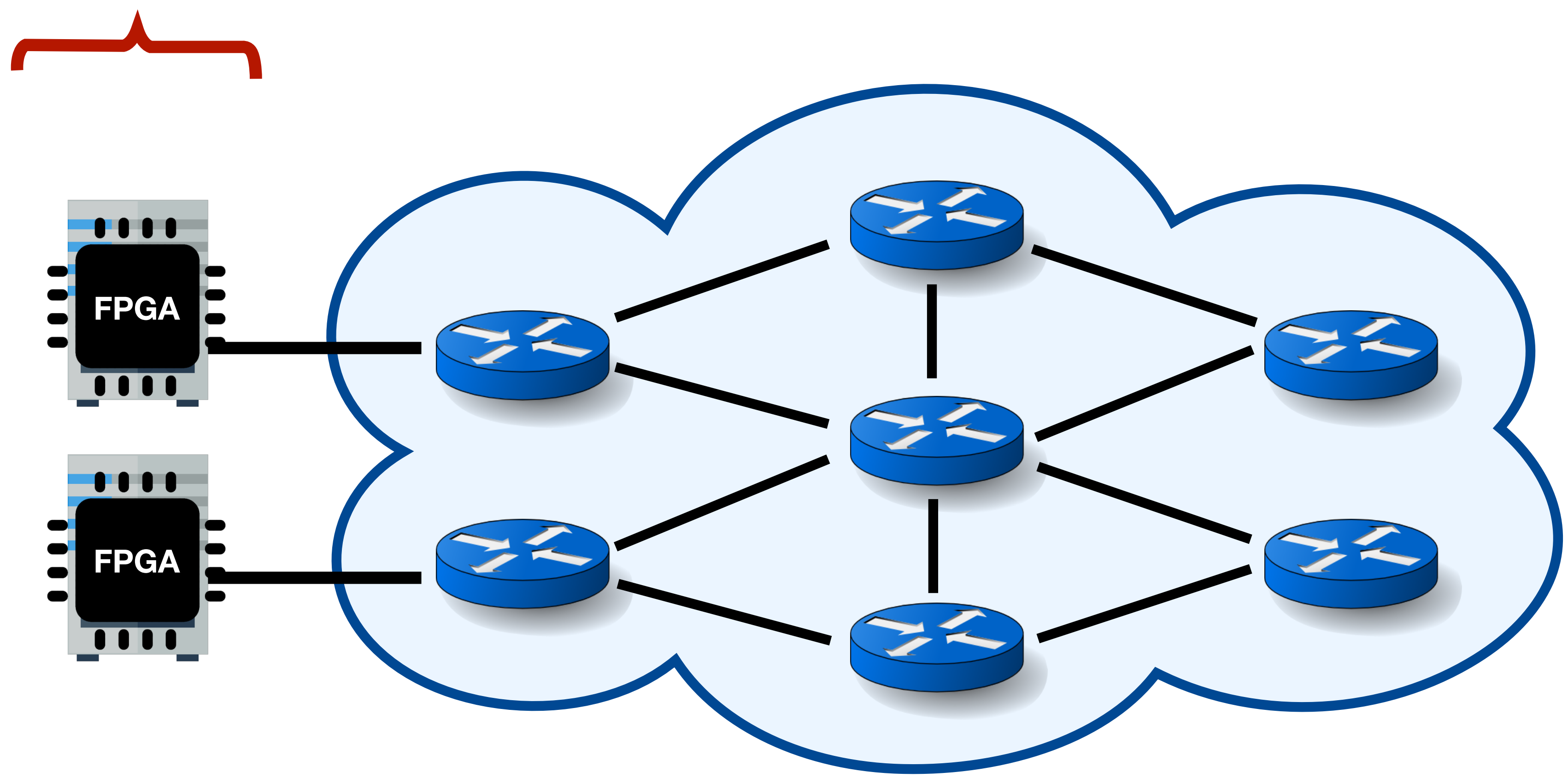
This Dissertation



This Dissertation

Tonic

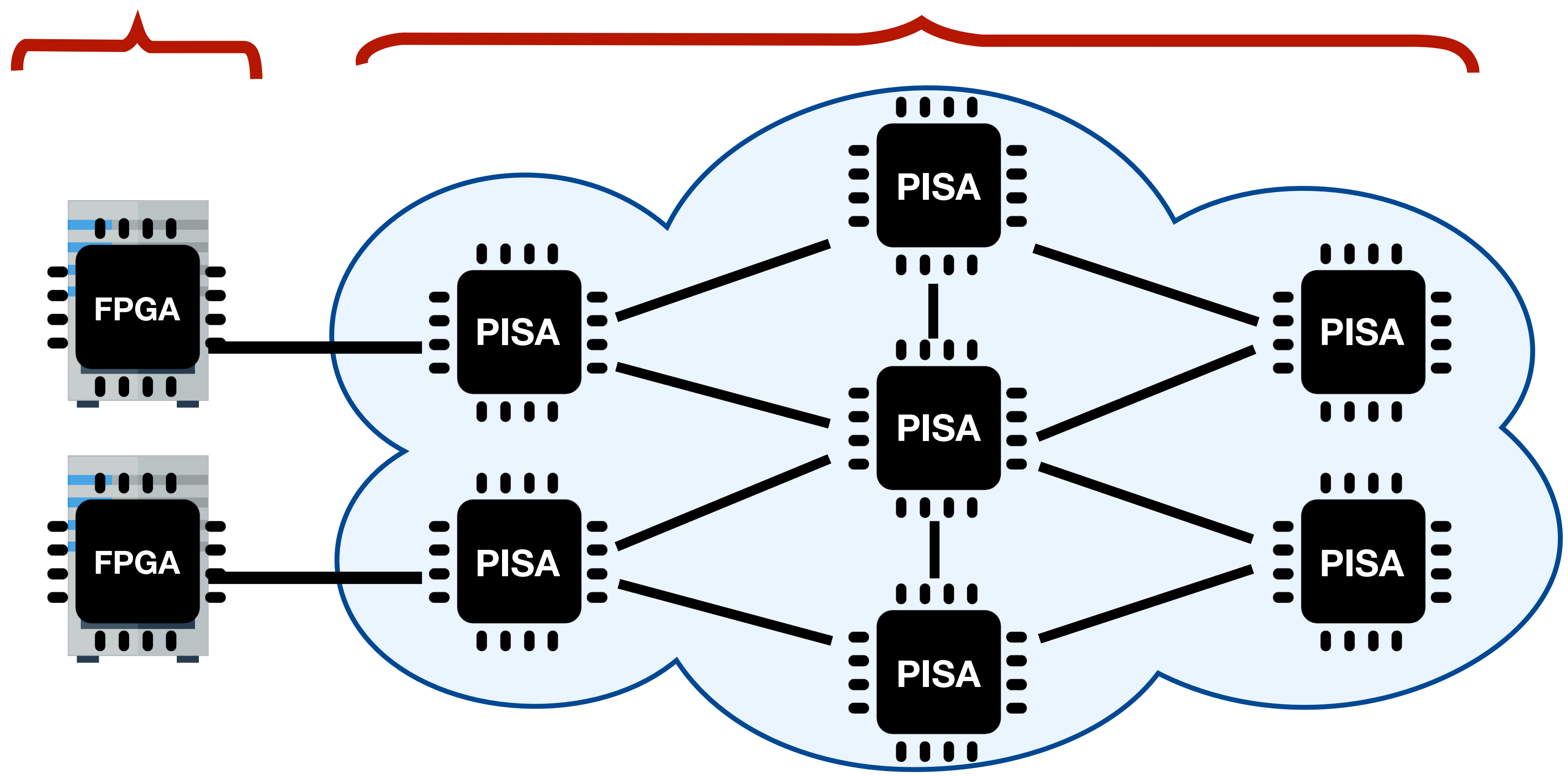
[Under revision for NSDI'20]



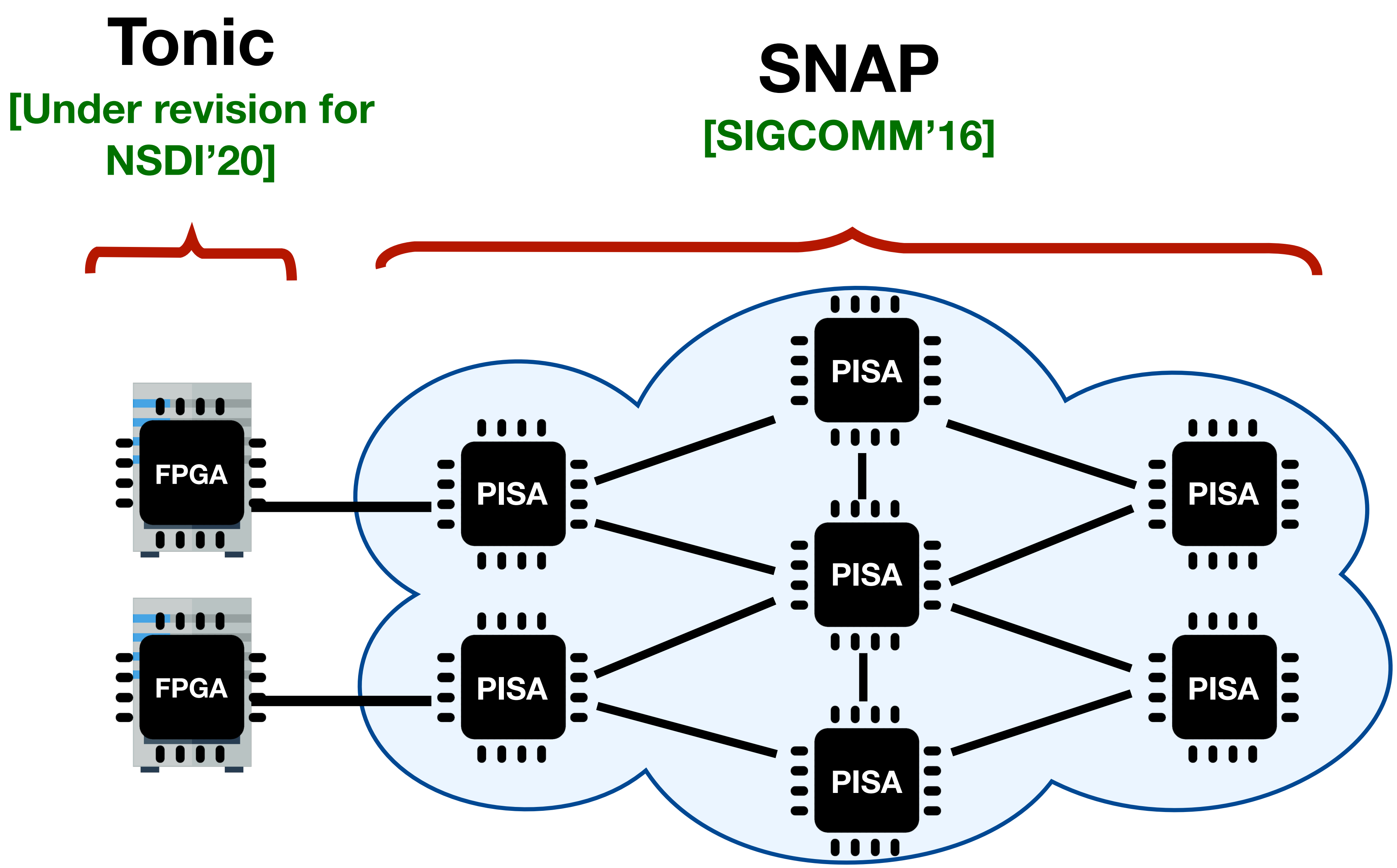
This Dissertation

Tonic
[Under revision for NSDI'20]

SNAP
[SIGCOMM'16]

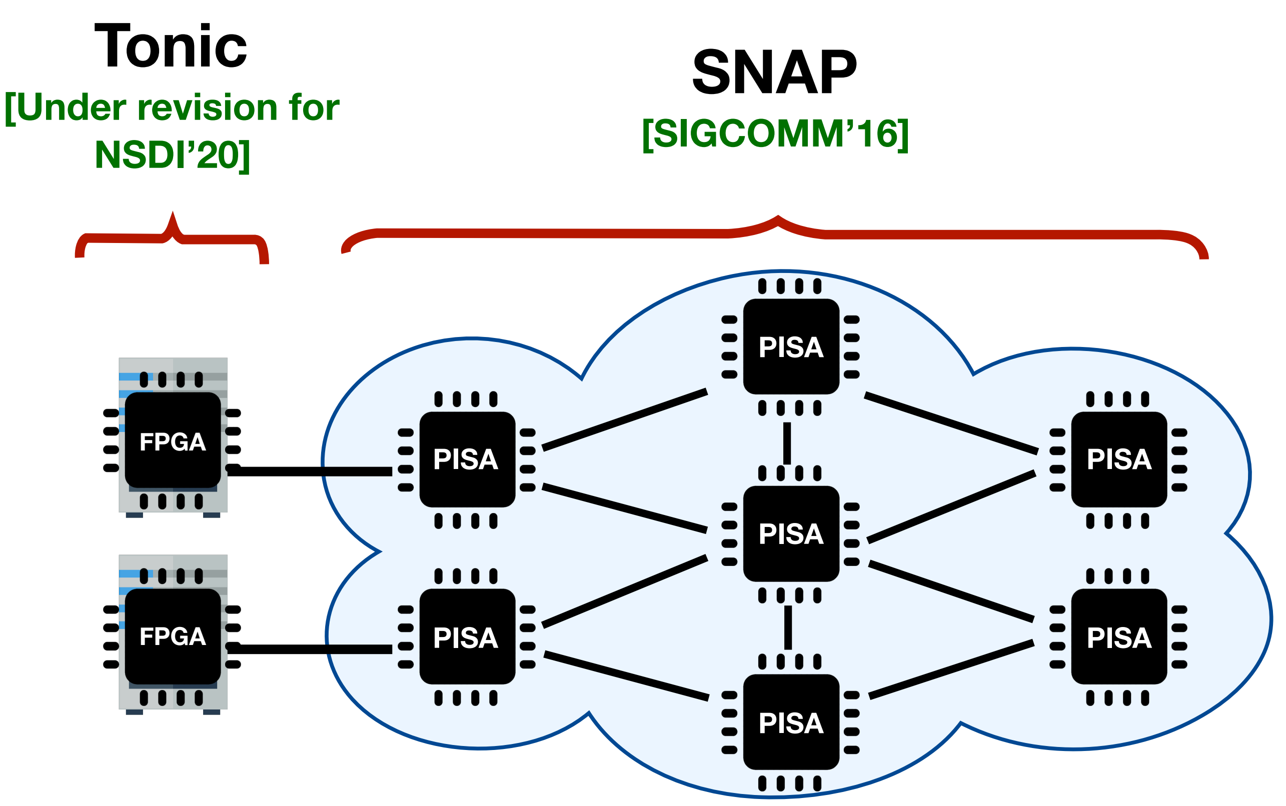


This Dissertation



This Dissertation

- With an emphasis on
  - modularity
  - minimizing development effort



Enabling Programmable Transport Protocols on High-Speed NICs

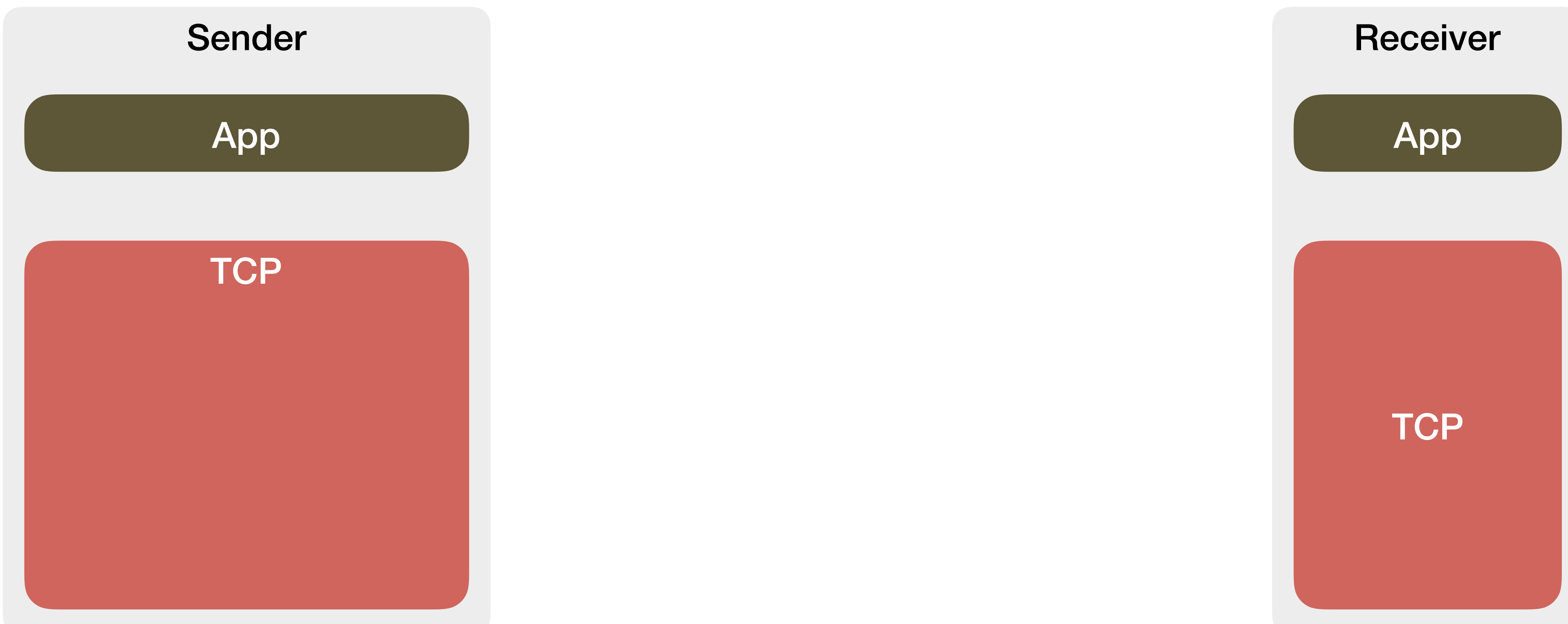
Mina Tahmasbi Arashloo[1], Alexey Lavrov[1],
Manya Ghobadi[2], Jennifer Rexford[1],
David Walker[1], and David Wentzlaff[1]

[1] Princeton University, [2] MIT

TCP 101

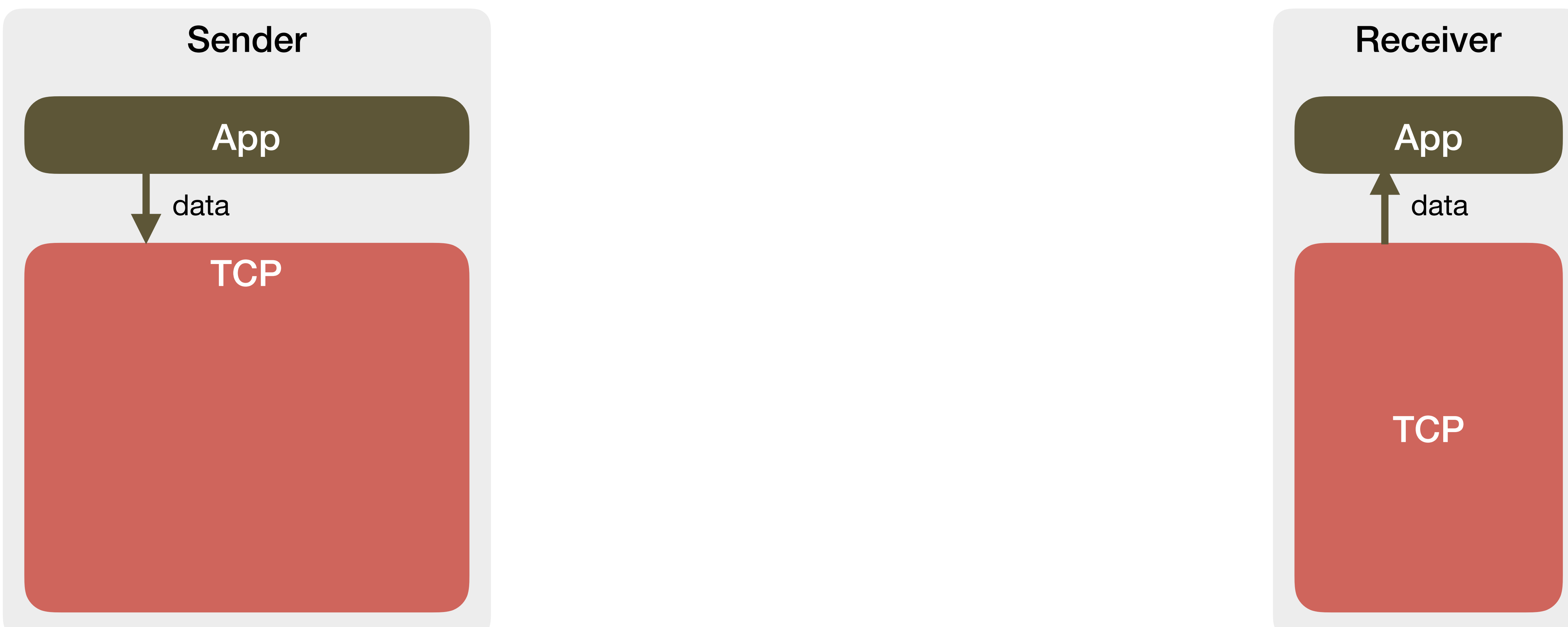
TCP 101

- The most common transport protocol



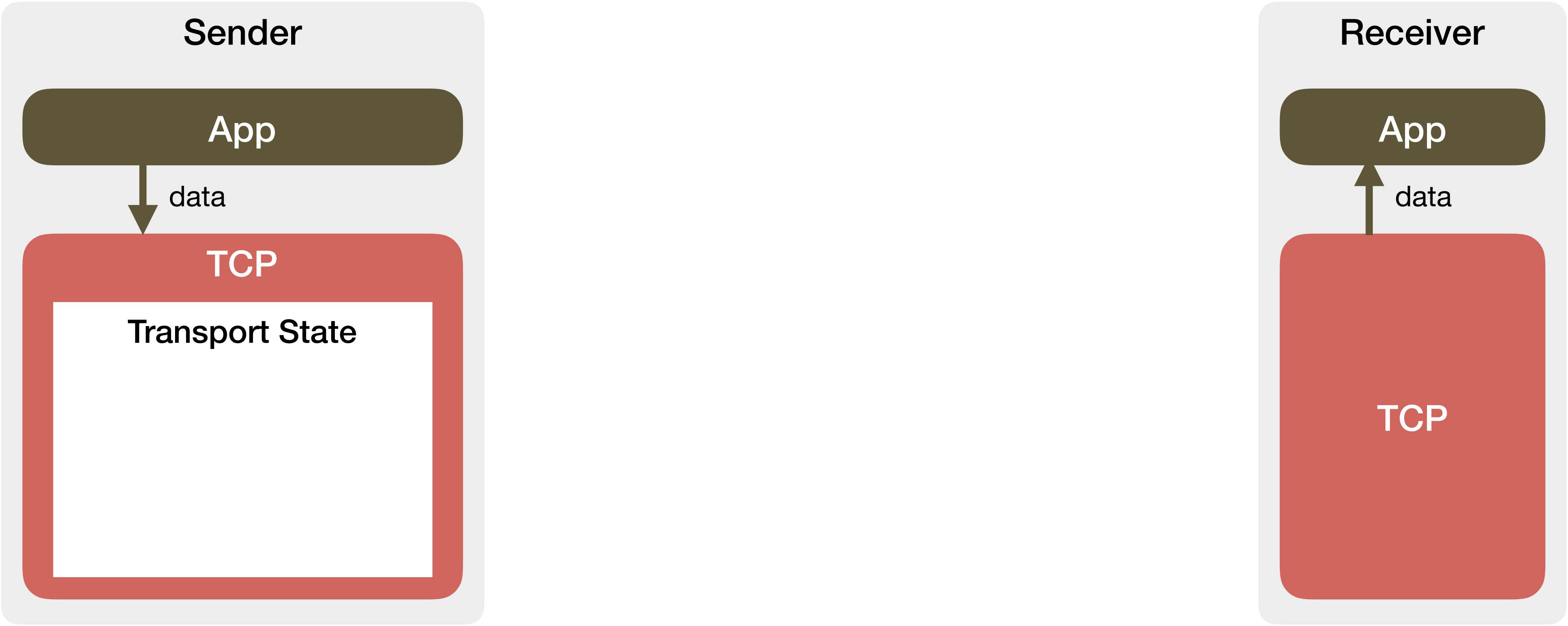
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**



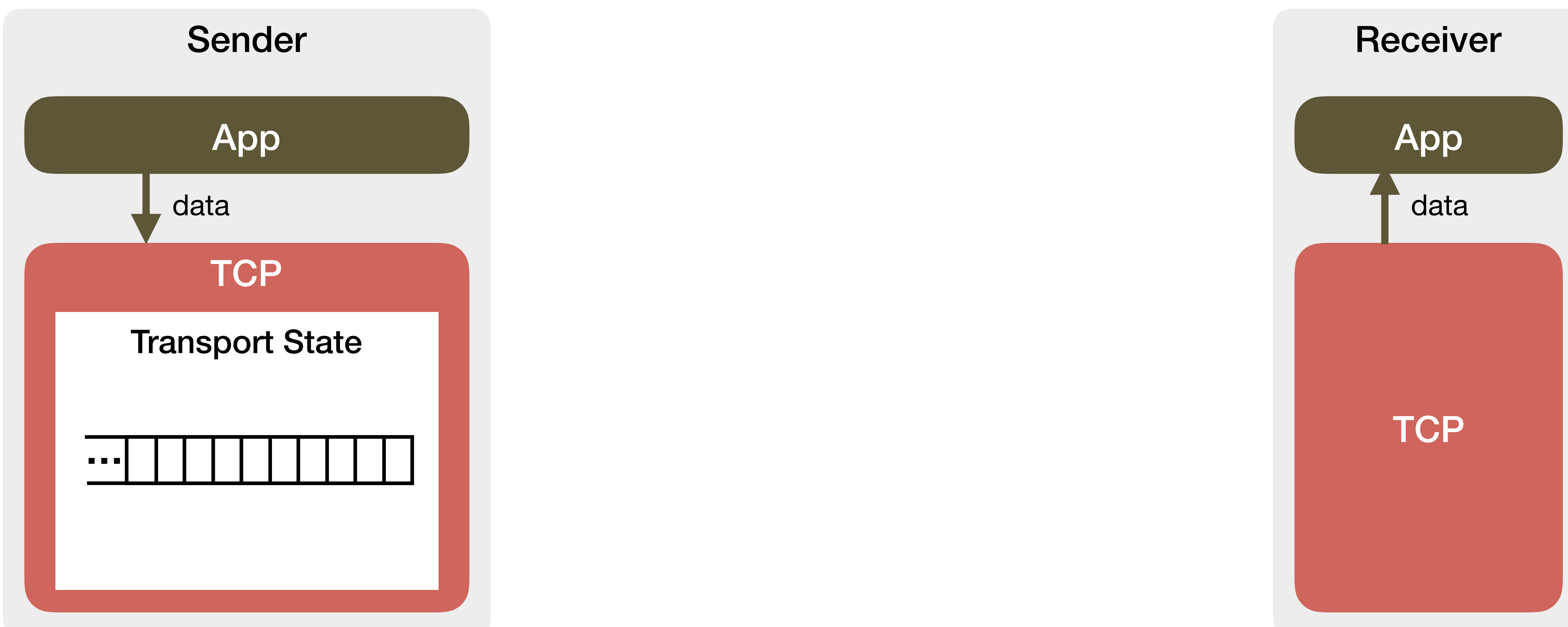
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**



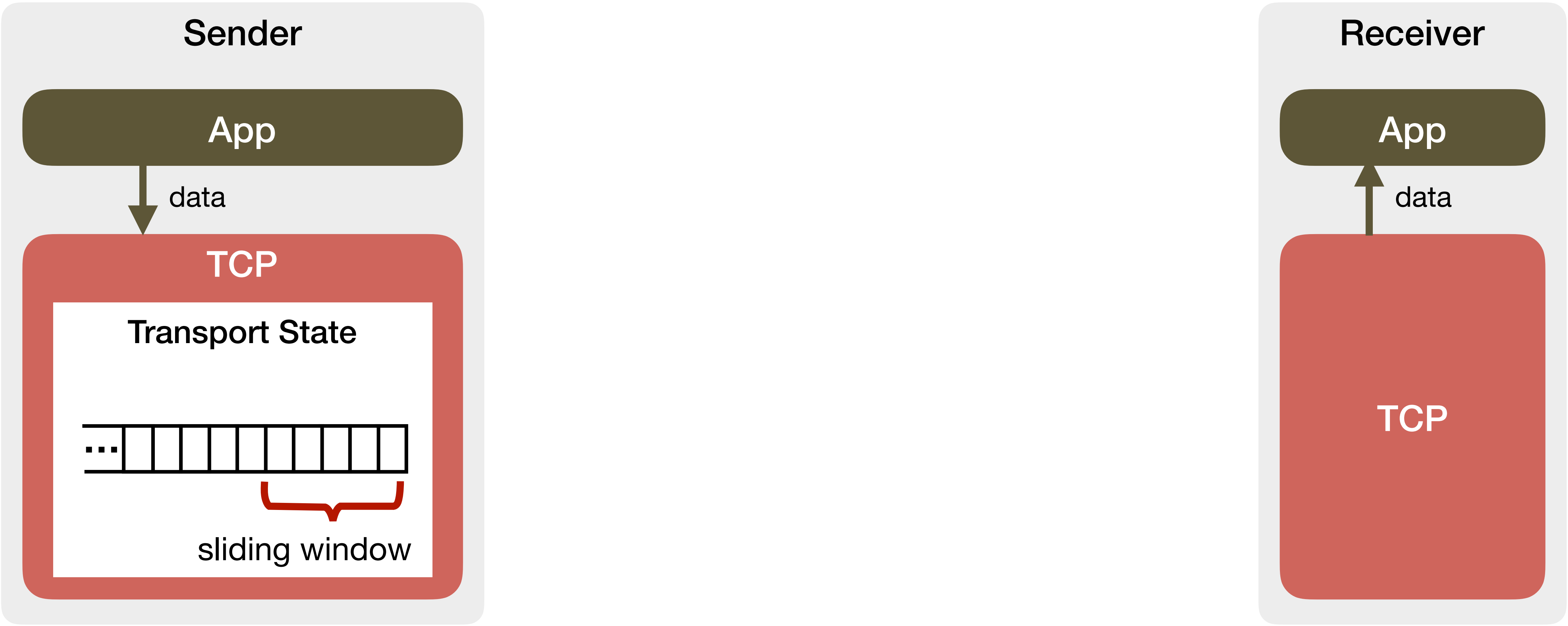
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**



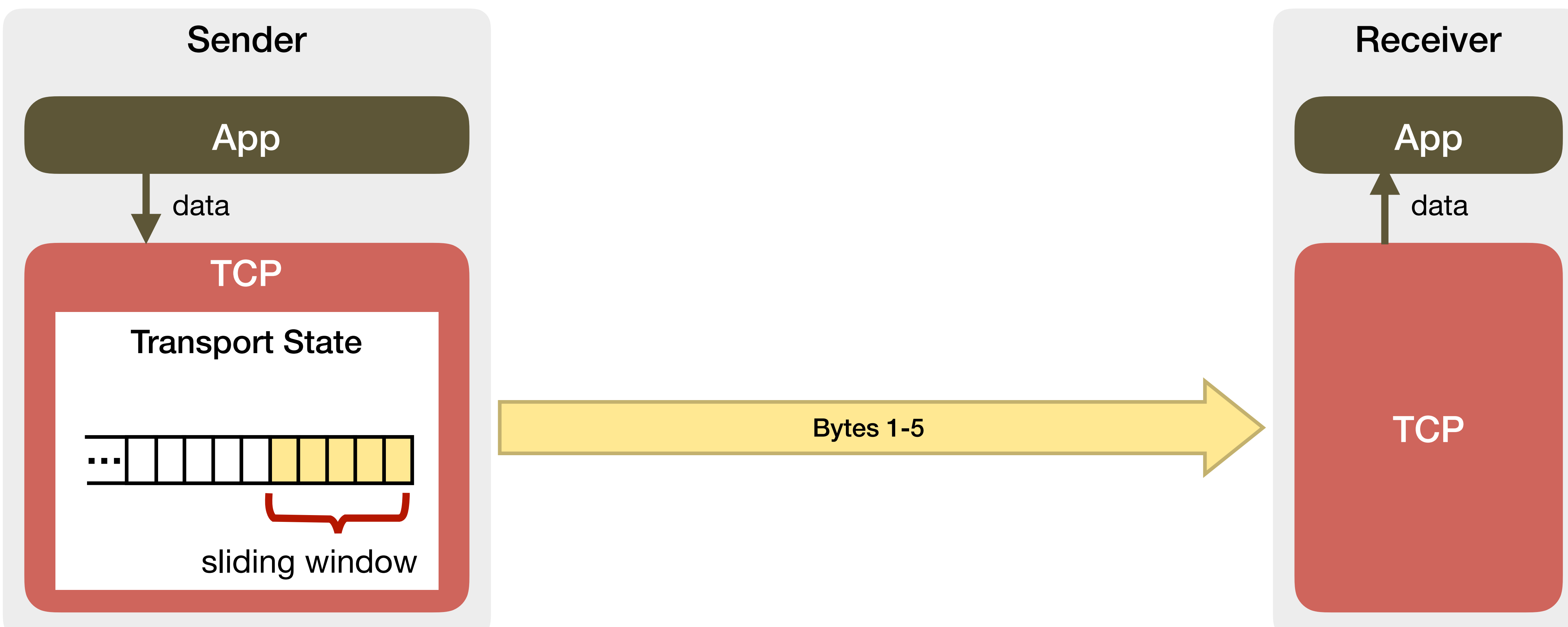
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**



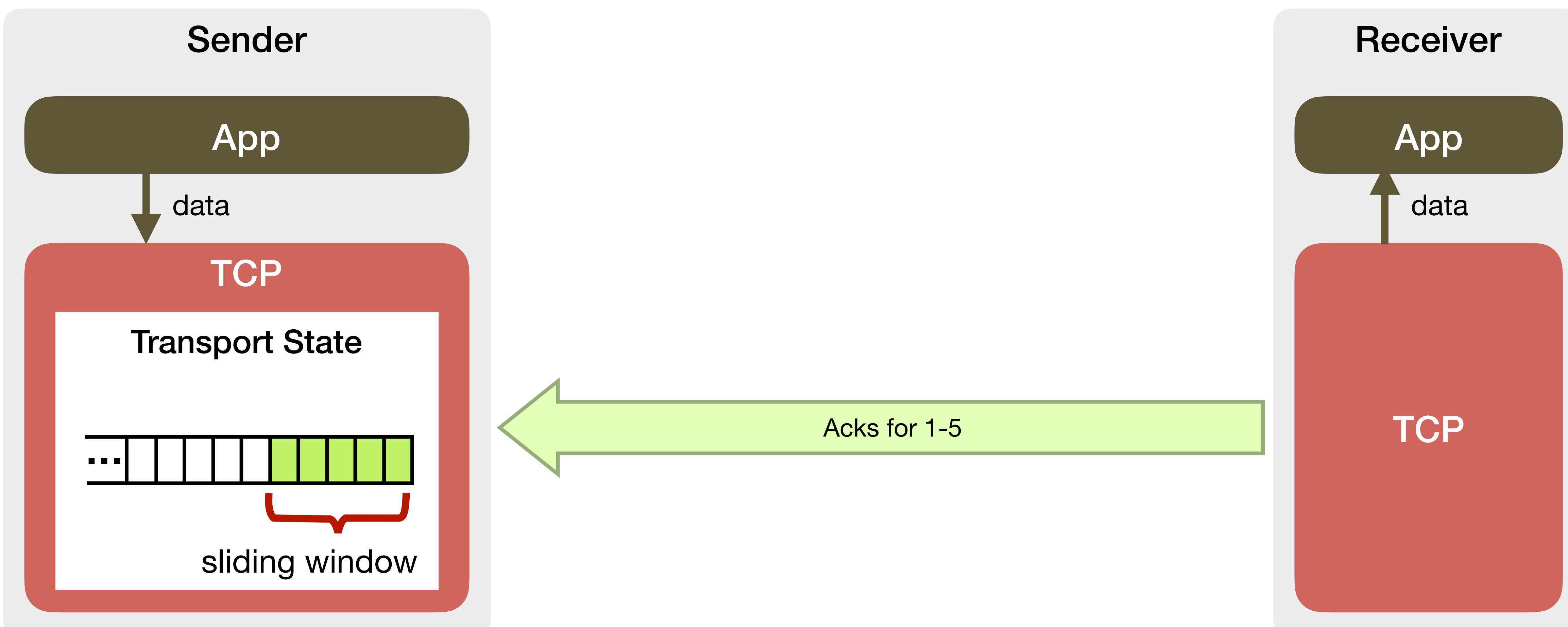
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**



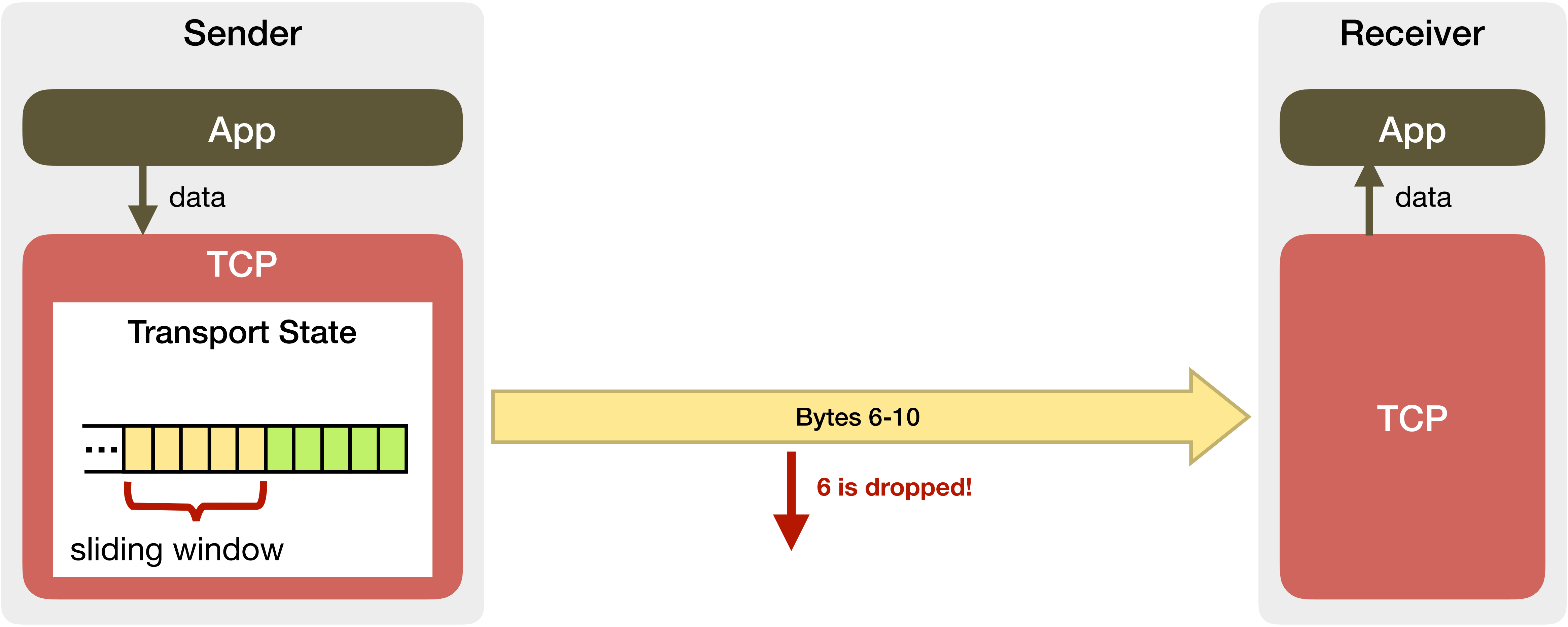
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**



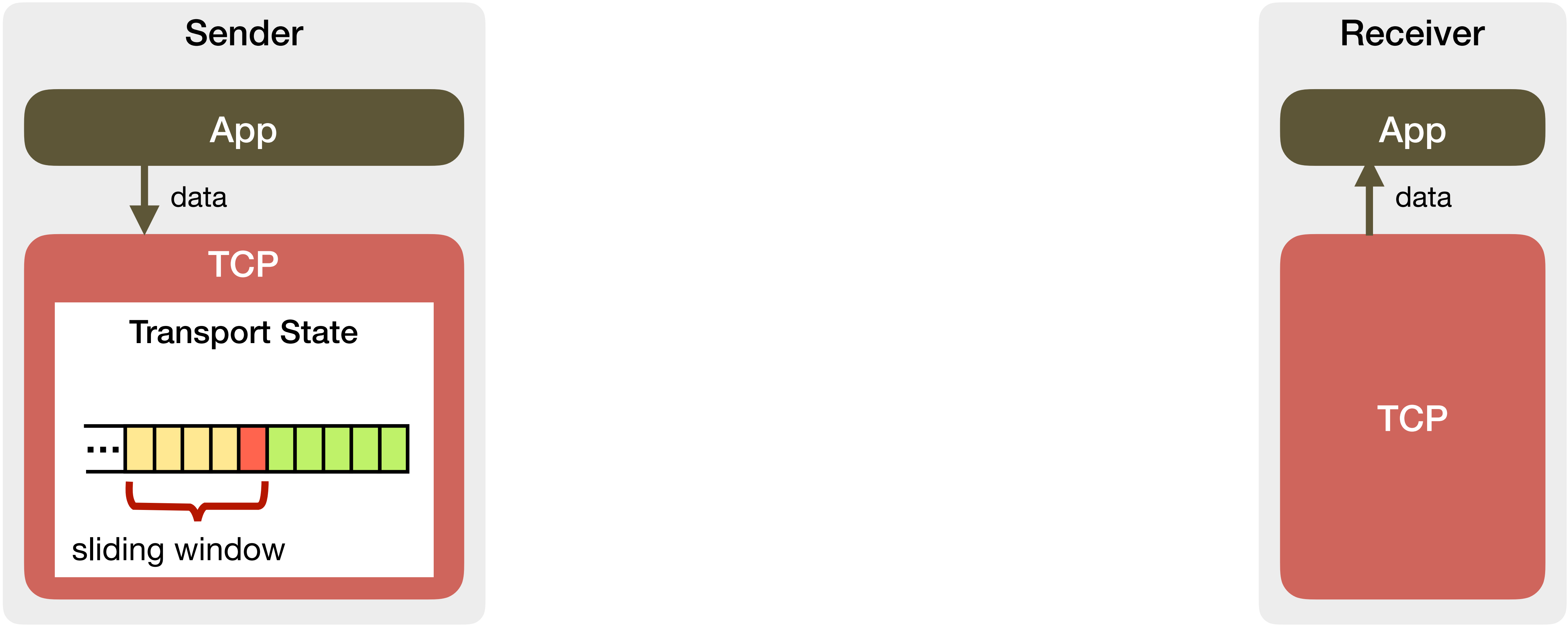
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**



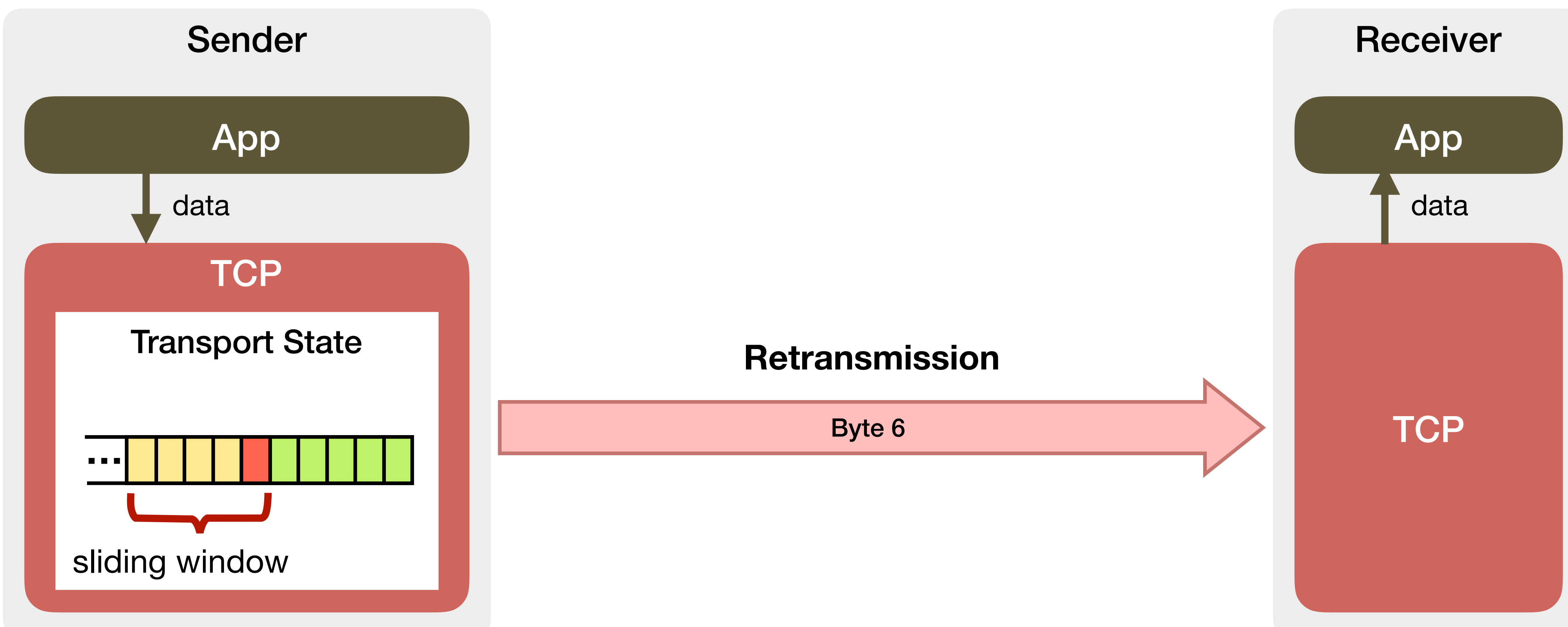
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**



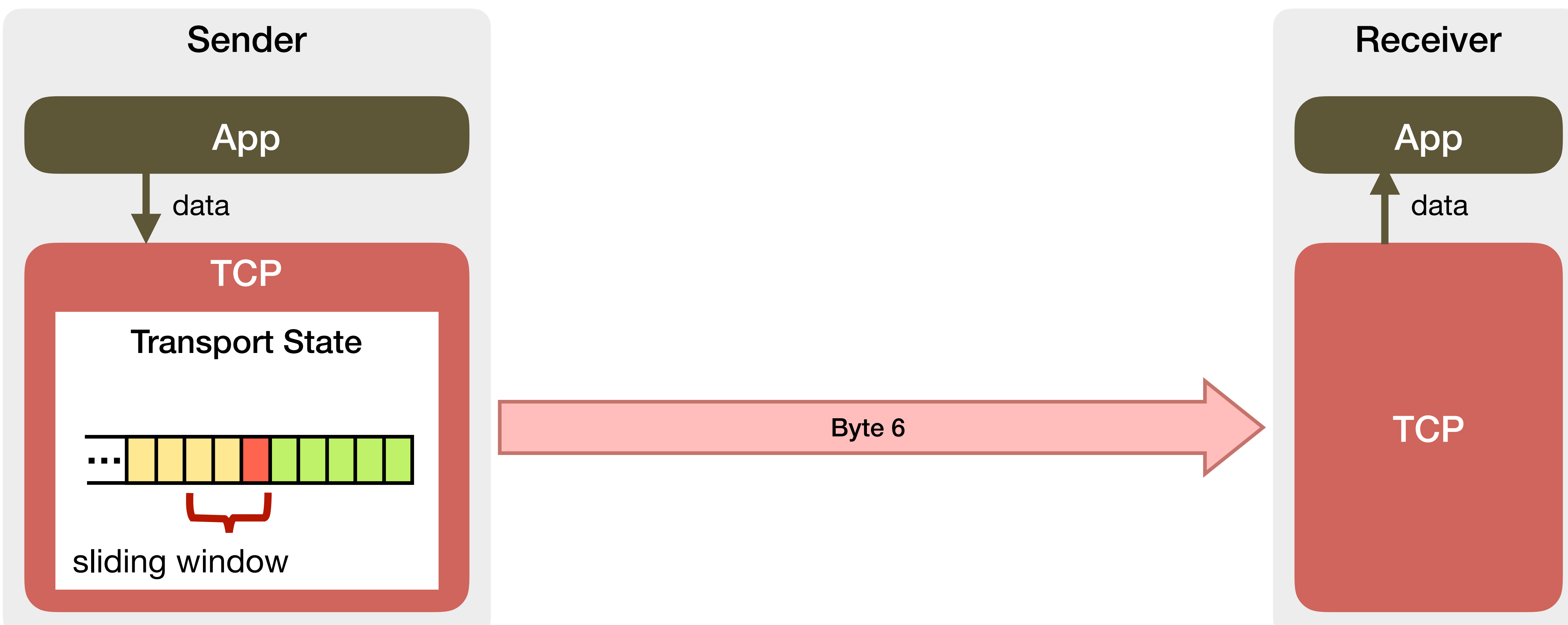
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**



TCP 101

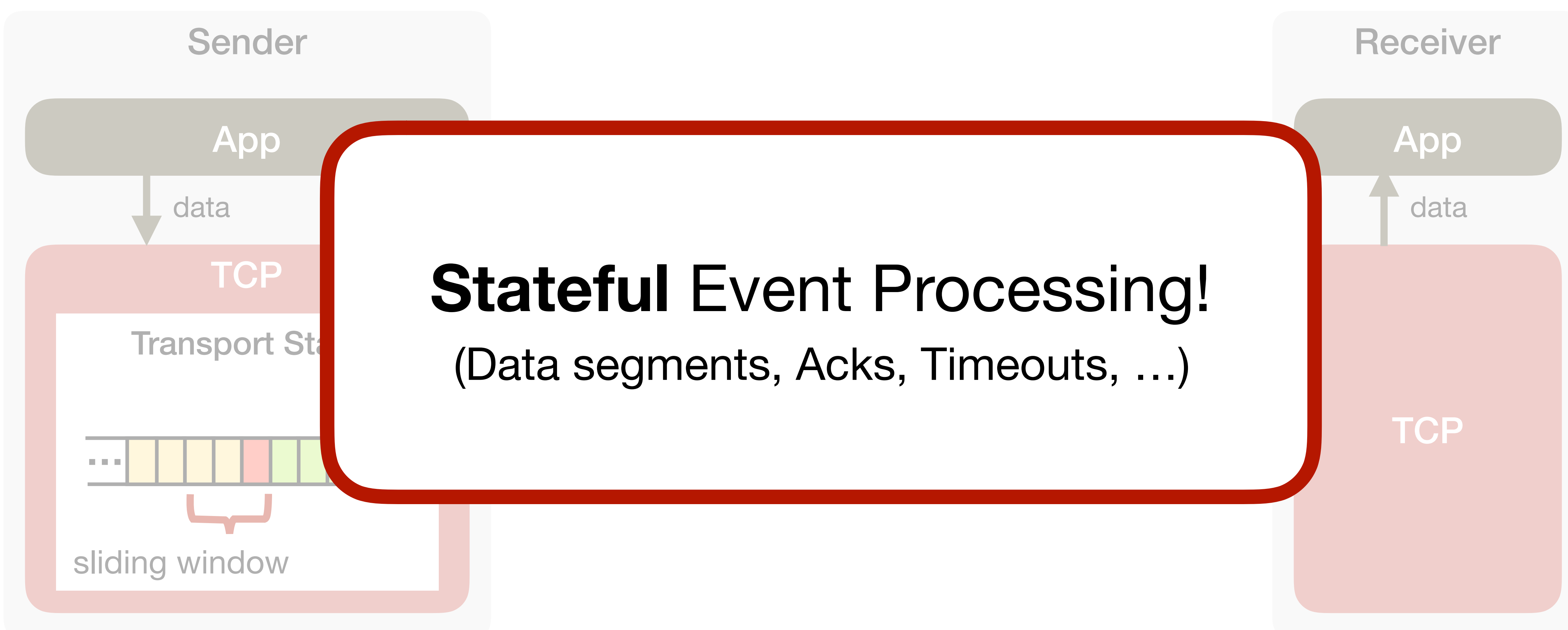
- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**



Window Adjustment

TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**



Constant Innovation in Transport Protocols

Constant Innovation in Transport Protocols

NDP
XCP
Homa
Karuna
PCC
DCTCP
D3
DCQCN
New Reno
Reno
IRN
pHost
CUBIC
QUIC
Sprout
TIMELY
BIC
Tahoe
RCP
BBR
TCP Vegas
PCC Vivace

Network Stacks in Data Centers

Network Stacks in Data Centers

Target	CPU Overhead	Transport Programmability

Network Stacks in Data Centers

	Target	CPU Overhead	Transport Programmability
Software	kernel user space	30-40%	✔

Network Stacks in Data Centers

	Target	CPU Overhead	Transport Programmability
Software	**kernel** **user space**	**30-40%**	✔
Fixed-Function Hardware	**NIC**	**~none**	✘

Network Stacks in Data Centers

	Target	CPU Overhead	Transport Programmability
Software	kernel user space	30-40%	✔
Fixed-Function Hardware	NIC	~none	✘
Programmable Hardware (Tonic)	NIC	~none	✔

Challenges of Hardware Programming for High-Speed NICs

Challenges of Hardware Programming for High-Speed NICs

- **Timing Constraints**
  - Median packet size in data centers is 200 bytes
  - At 100 Gbps, one 128-byte packet every ~10 ns
    - Back-to-back stateful event processing

Challenges of Hardware Programming for High-Speed NICs

- **Timing Constraints**
  - Median packet size in data centers is 200 bytes
  - At 100 Gbps, one 128-byte packet every ~10 ns
    - Back-to-back stateful event processing
- **Memory Constraints**
  - A few megabytes of high-speed memory
  - More than a thousand active flows
  - A few kilobits of per-flow state

Challenges of Hardware Programming for High-Speed NICs

- Timing Constraints
  - M
  - A
- Me
  - A
  - M
  - A few kilobits of per-flow state

Tonic

- A **programmable** hardware architecture
  - running at **100** Gbps
  - within **memory limits of commodity NICs**
- to implement transport protocols
  - with **modest development effort**

Main Observation

Main Observation

Common transport patterns as reusable components

Main Observation

Common transport patterns as reusable components

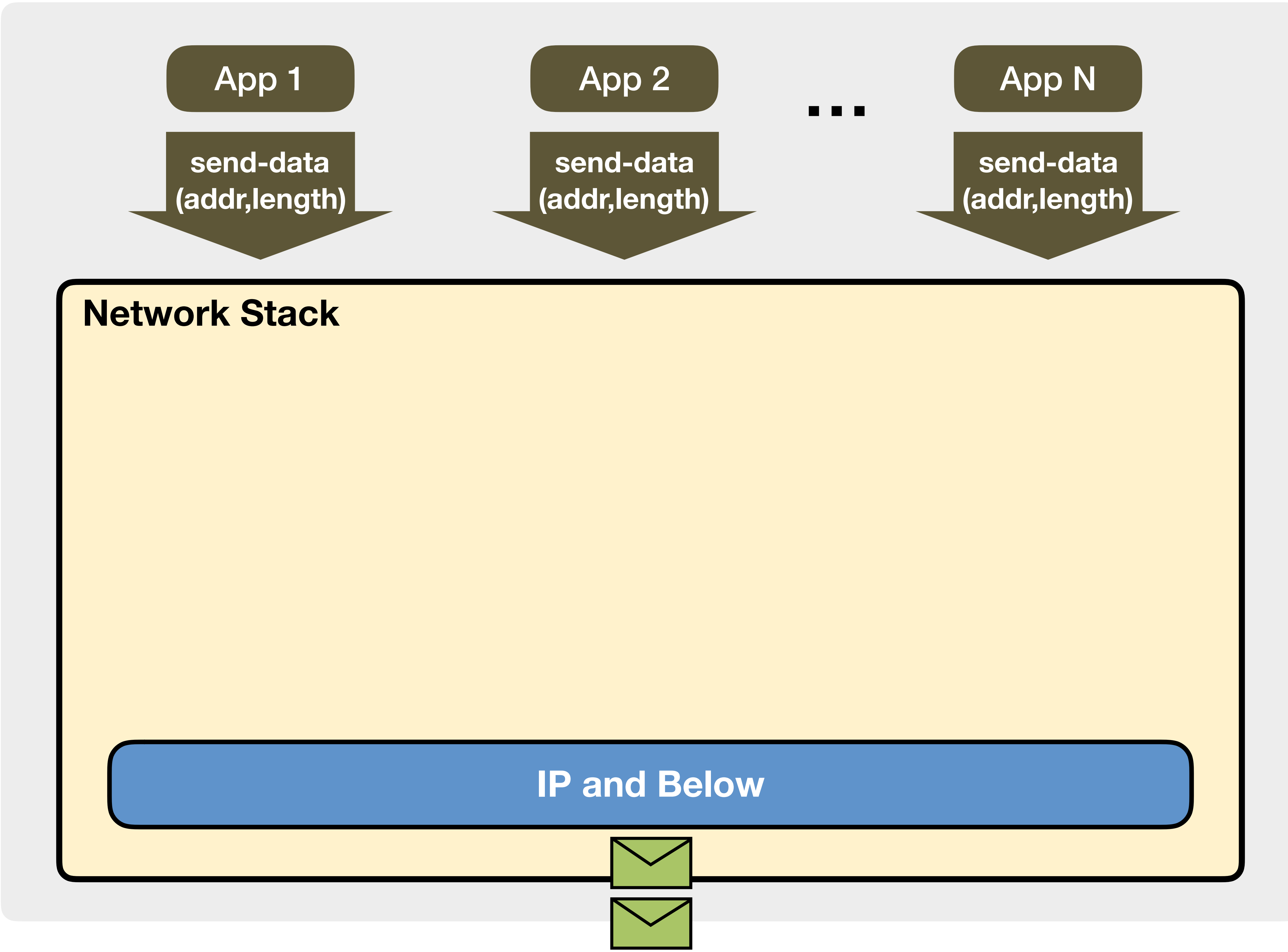
- drive the design of an efficient hardware “template” for transport logic

Main Observation

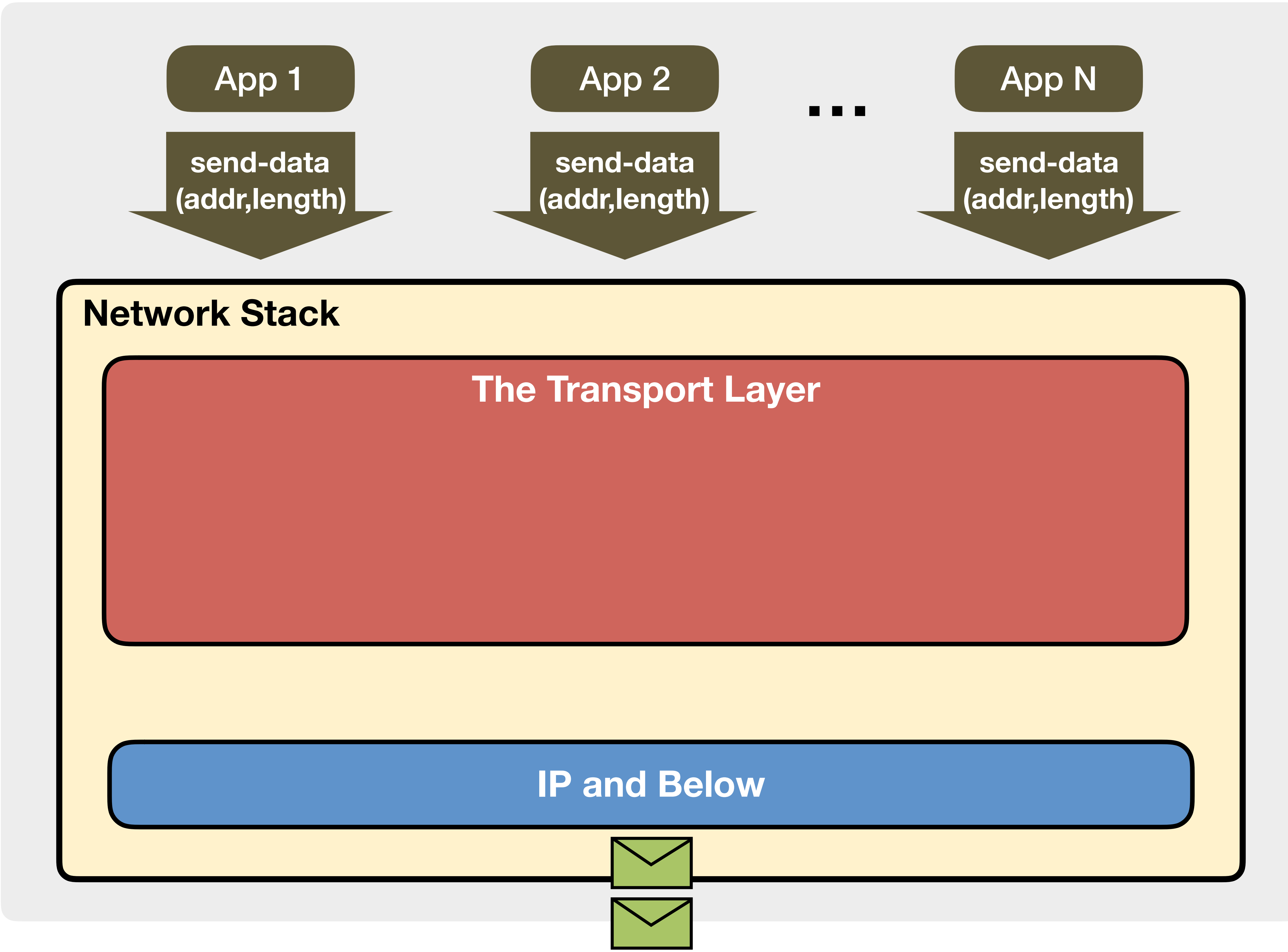
Common transport patterns as reusable components

- drive the design of an efficient hardware “template” for transport logic
- reduce the functionality users must specify

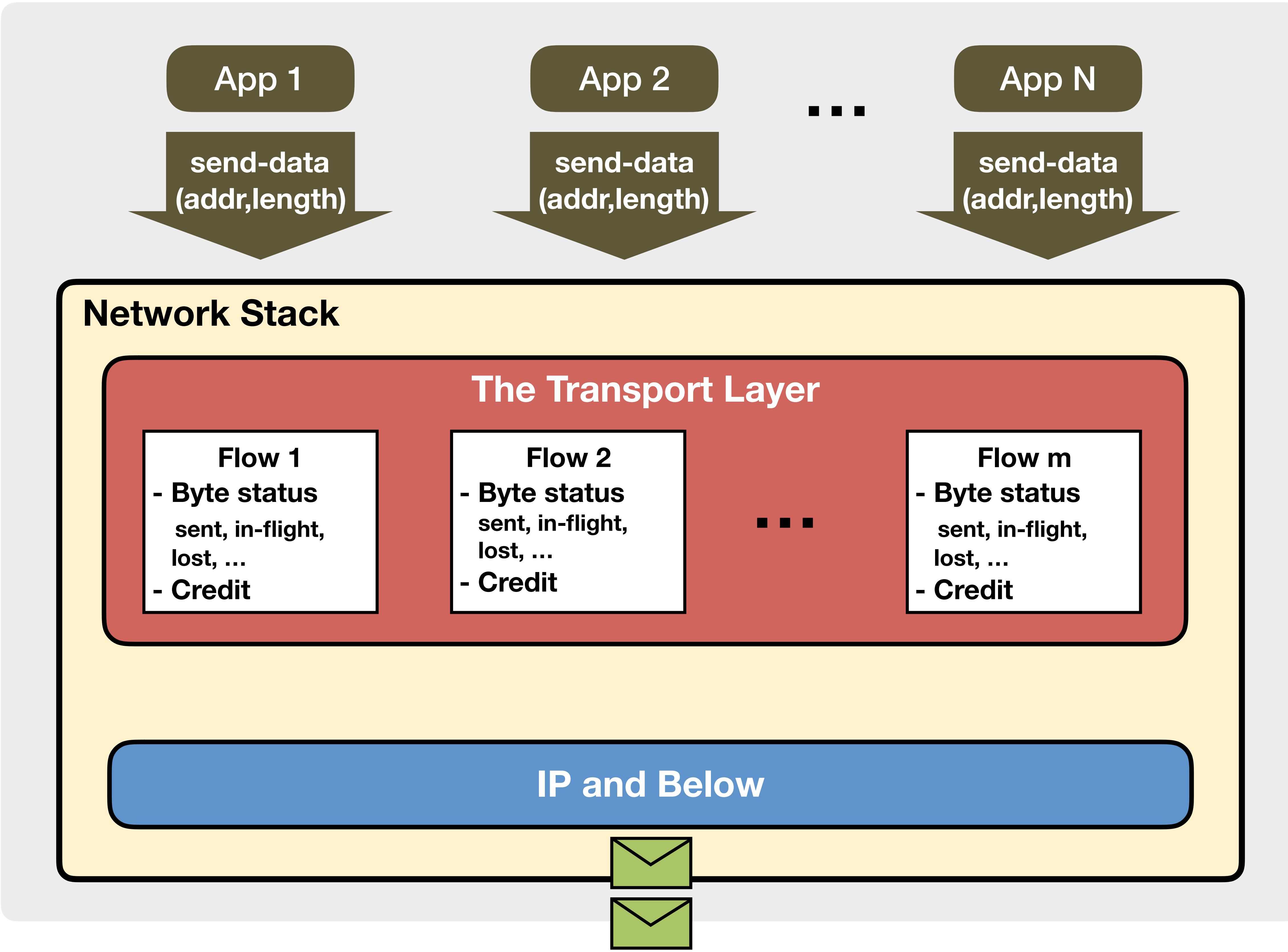
The Transport Layer



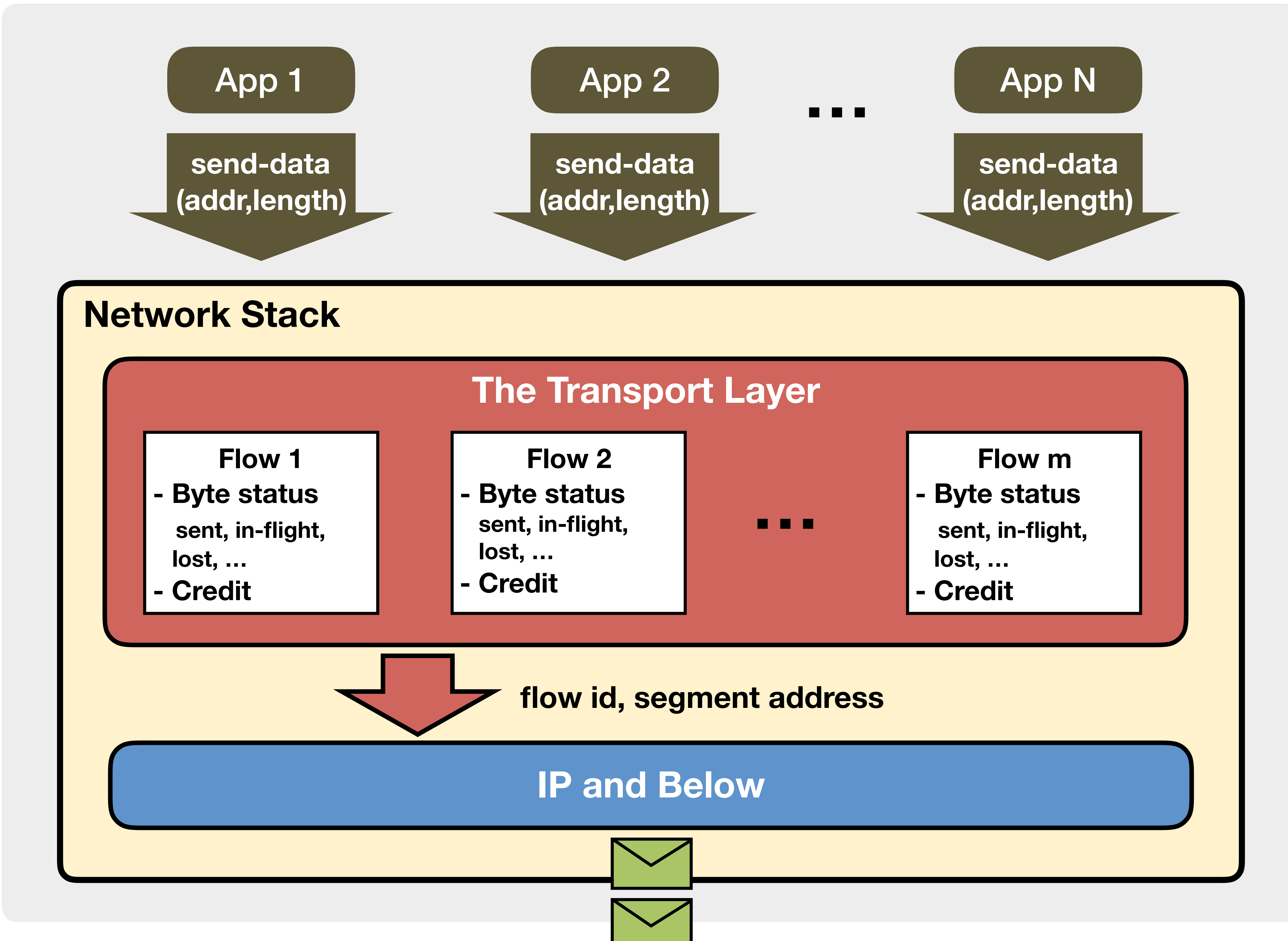
The Transport Layer



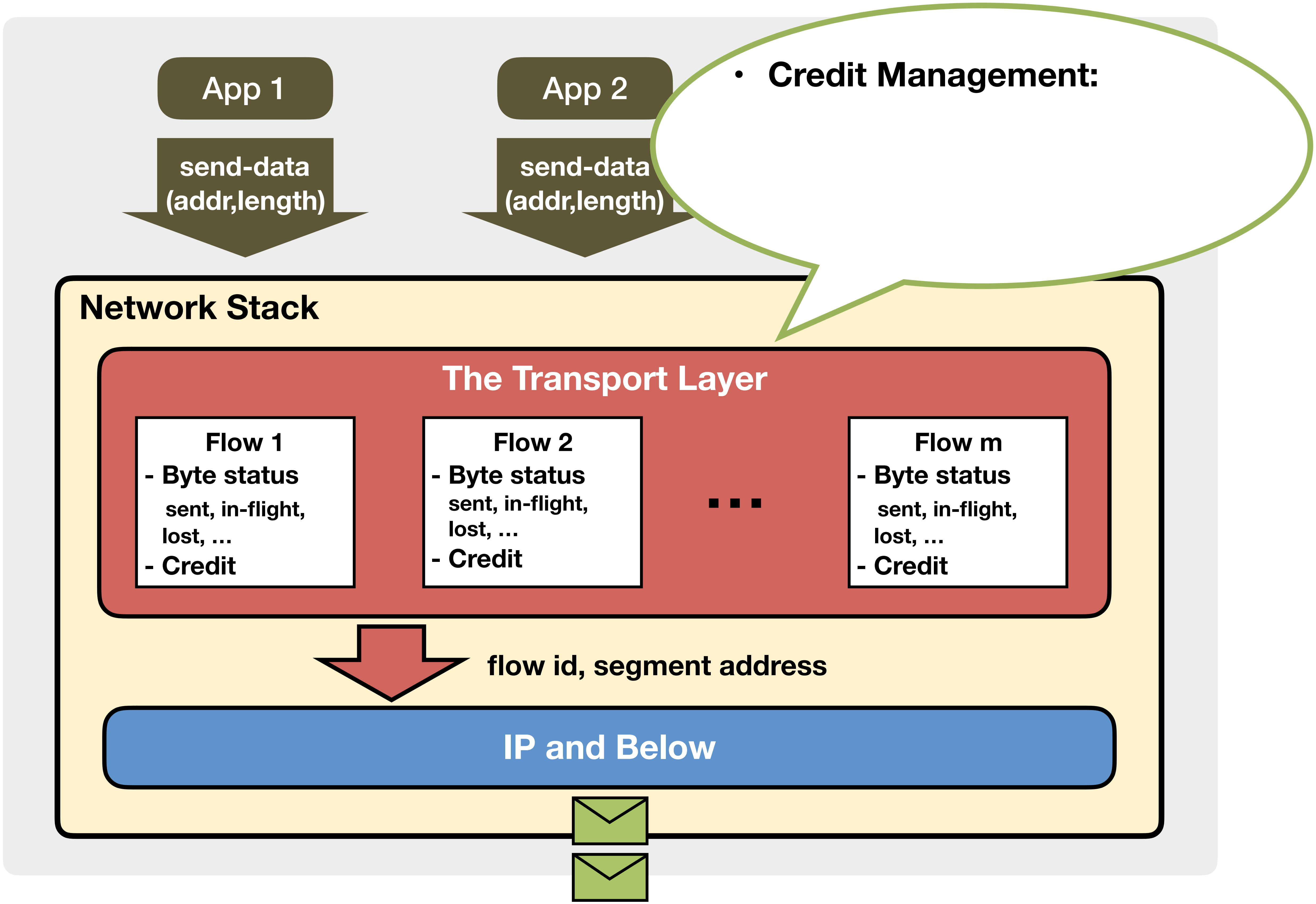
The Transport Layer



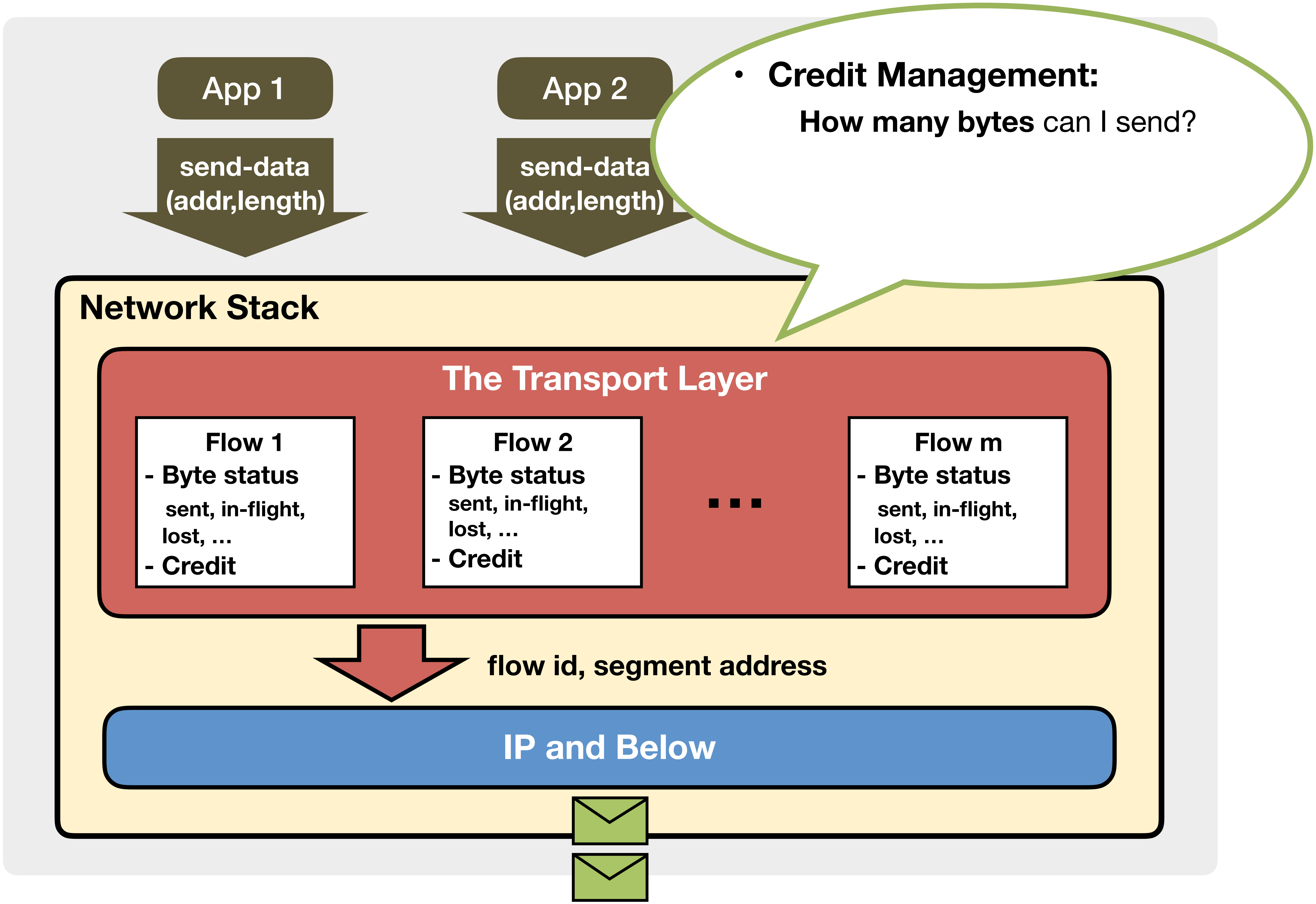
The Transport Layer



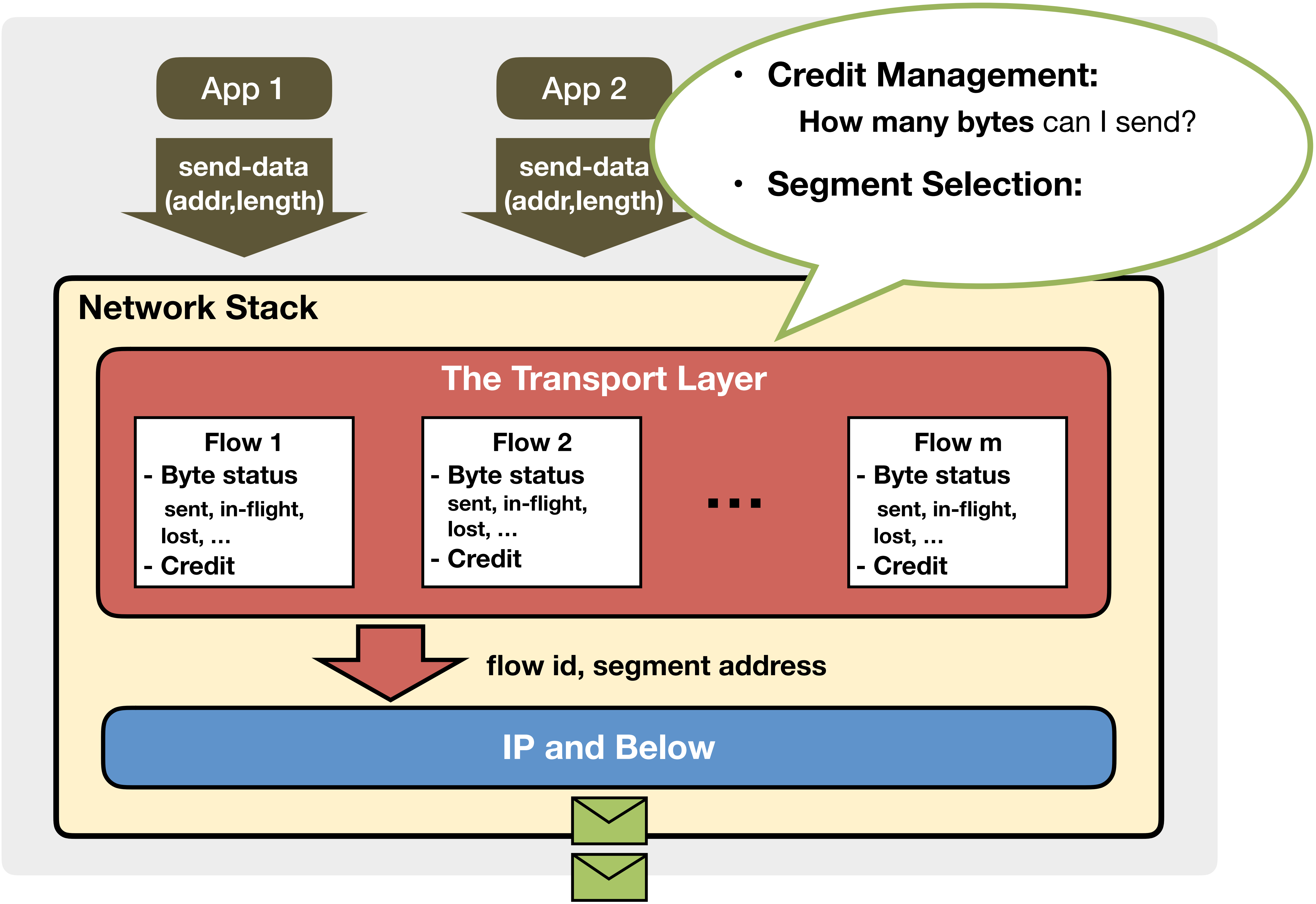
The Transport Layer



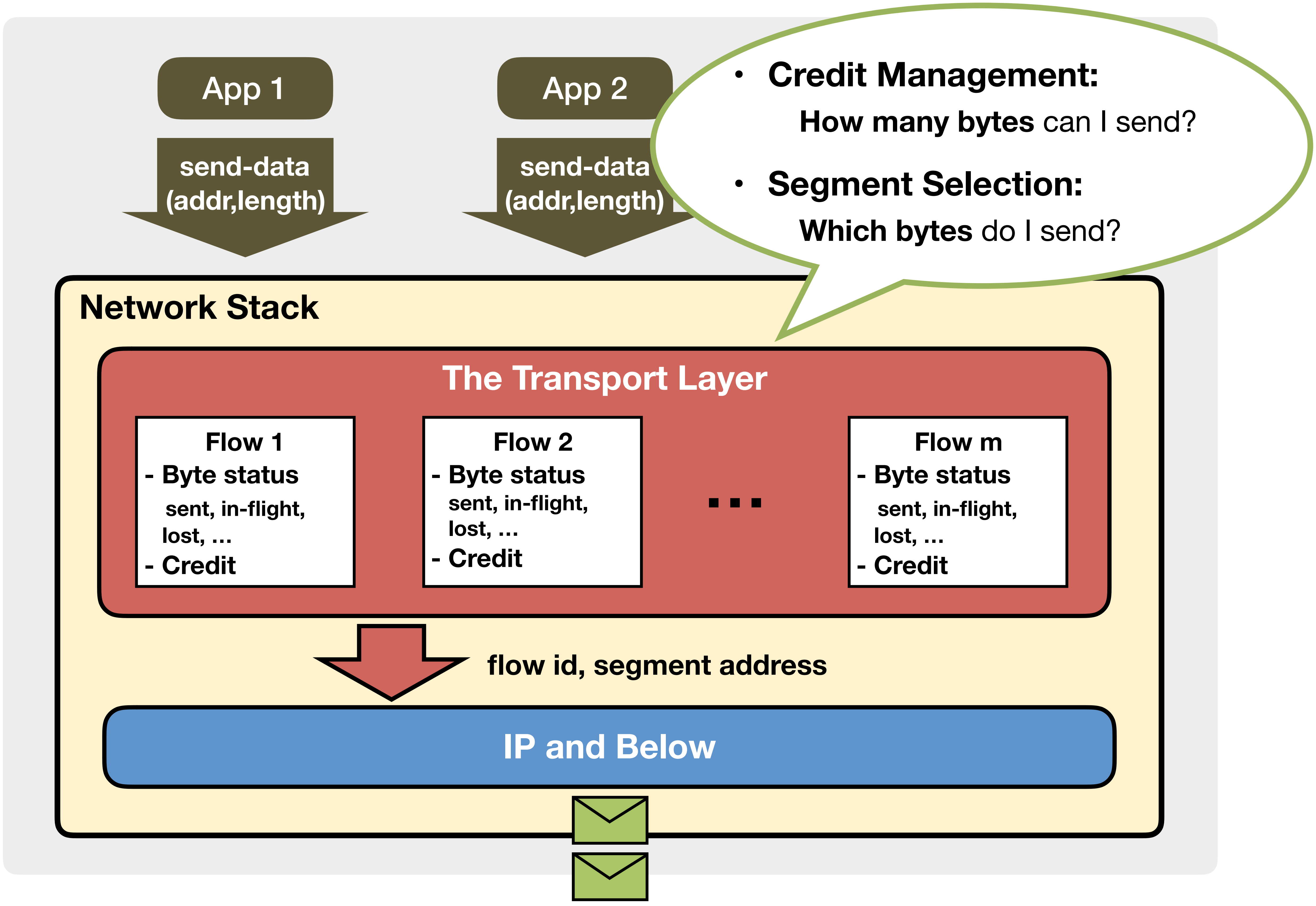
The Transport Layer



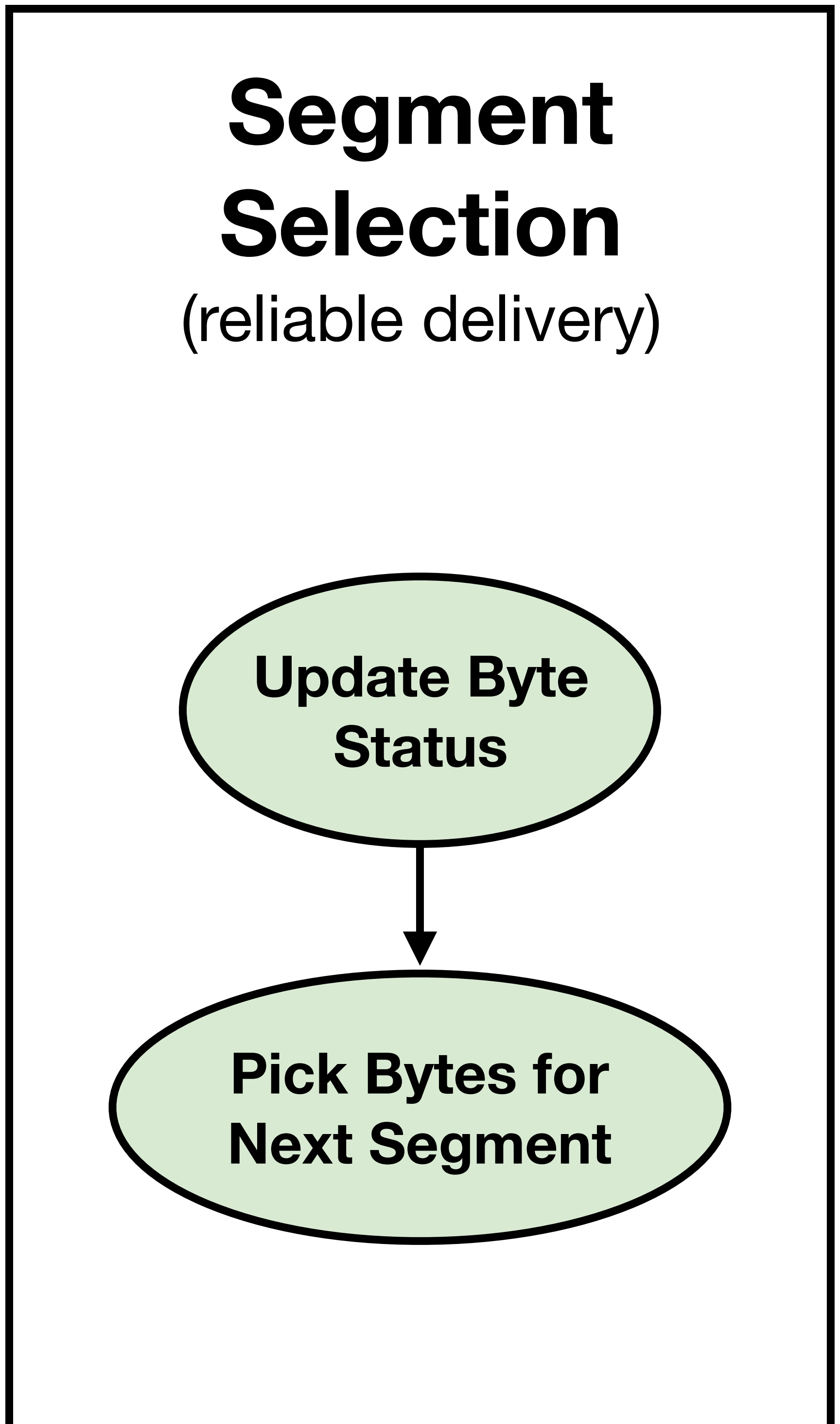
The Transport Layer



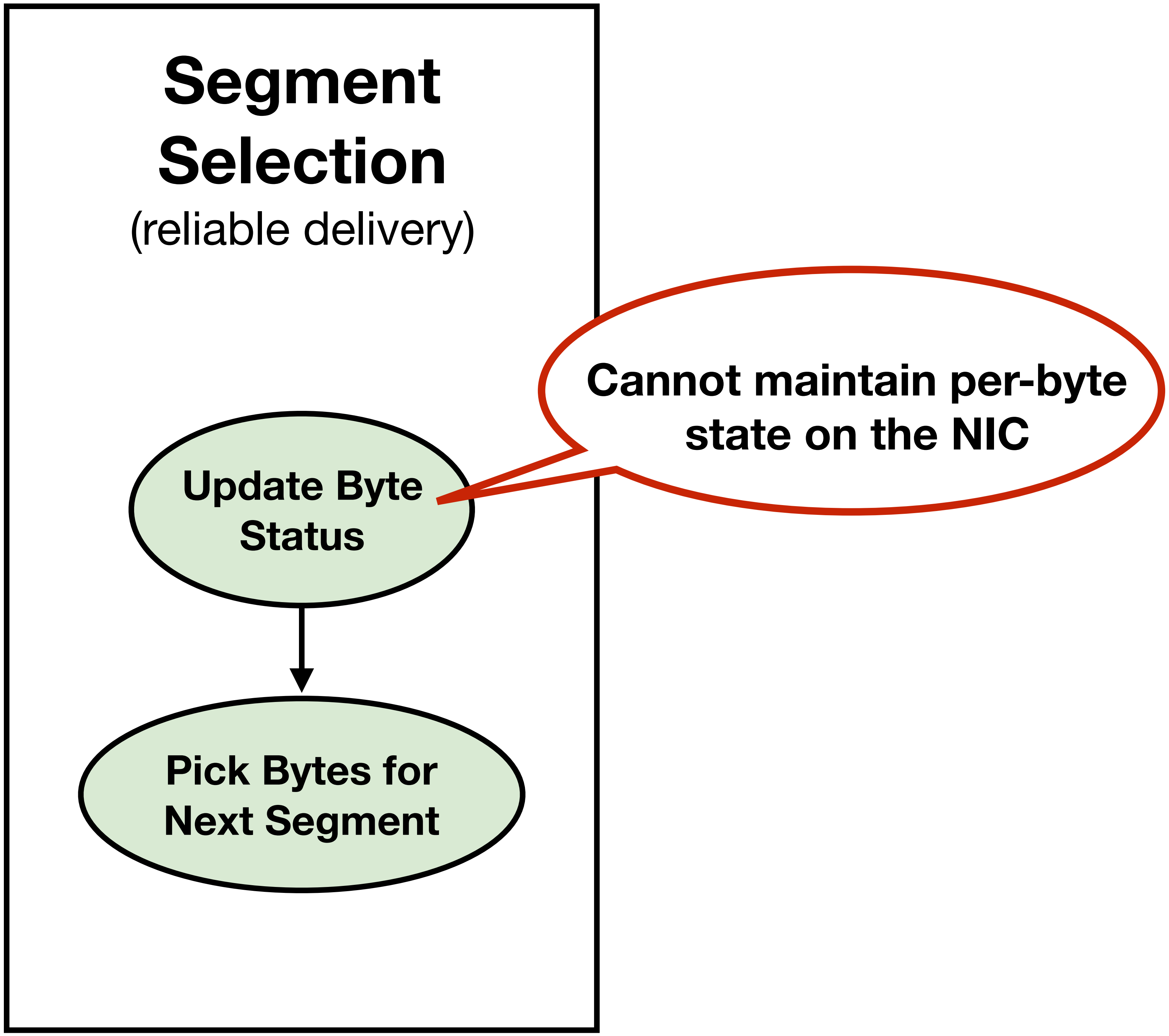
The Transport Layer



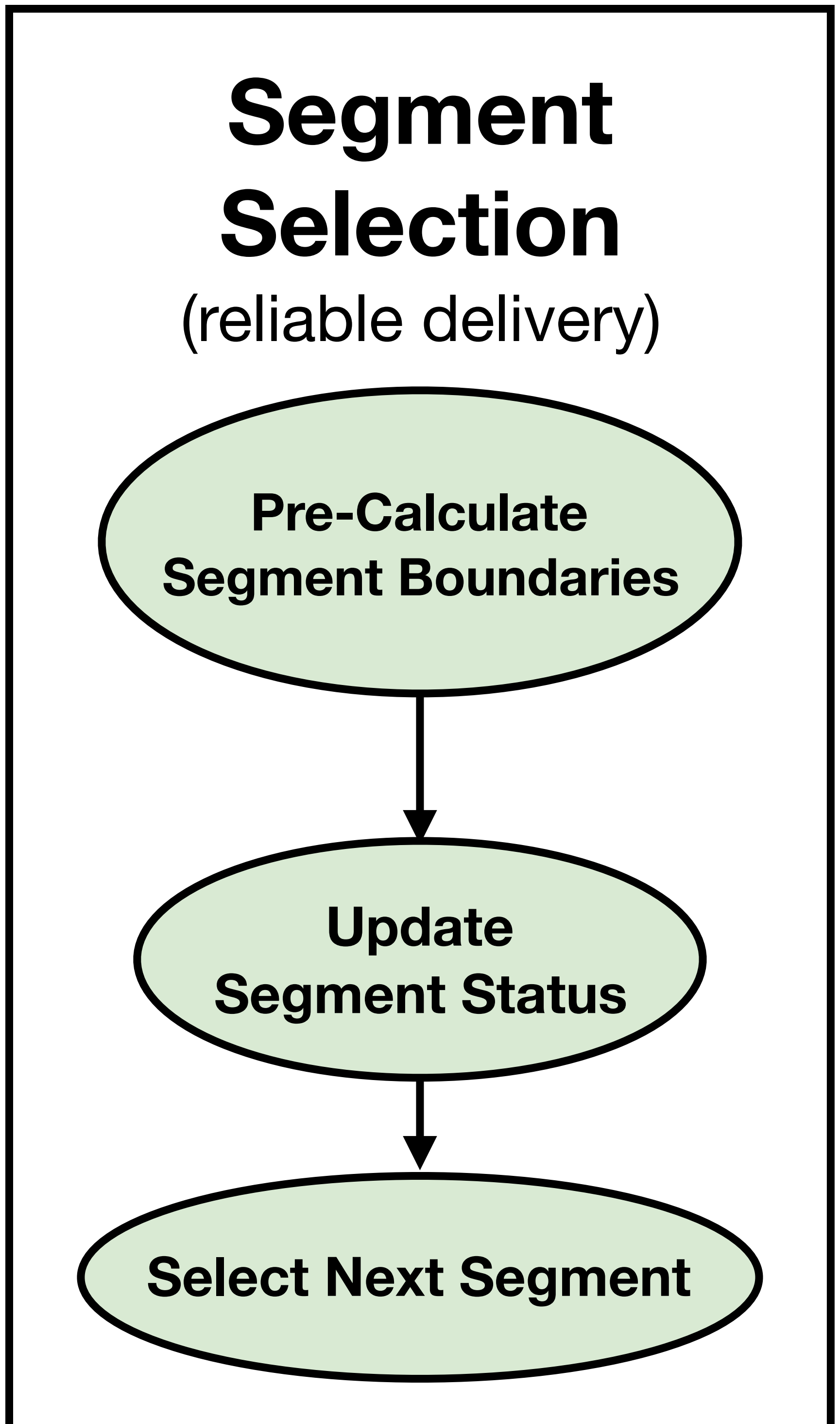
Segment Selection Patterns



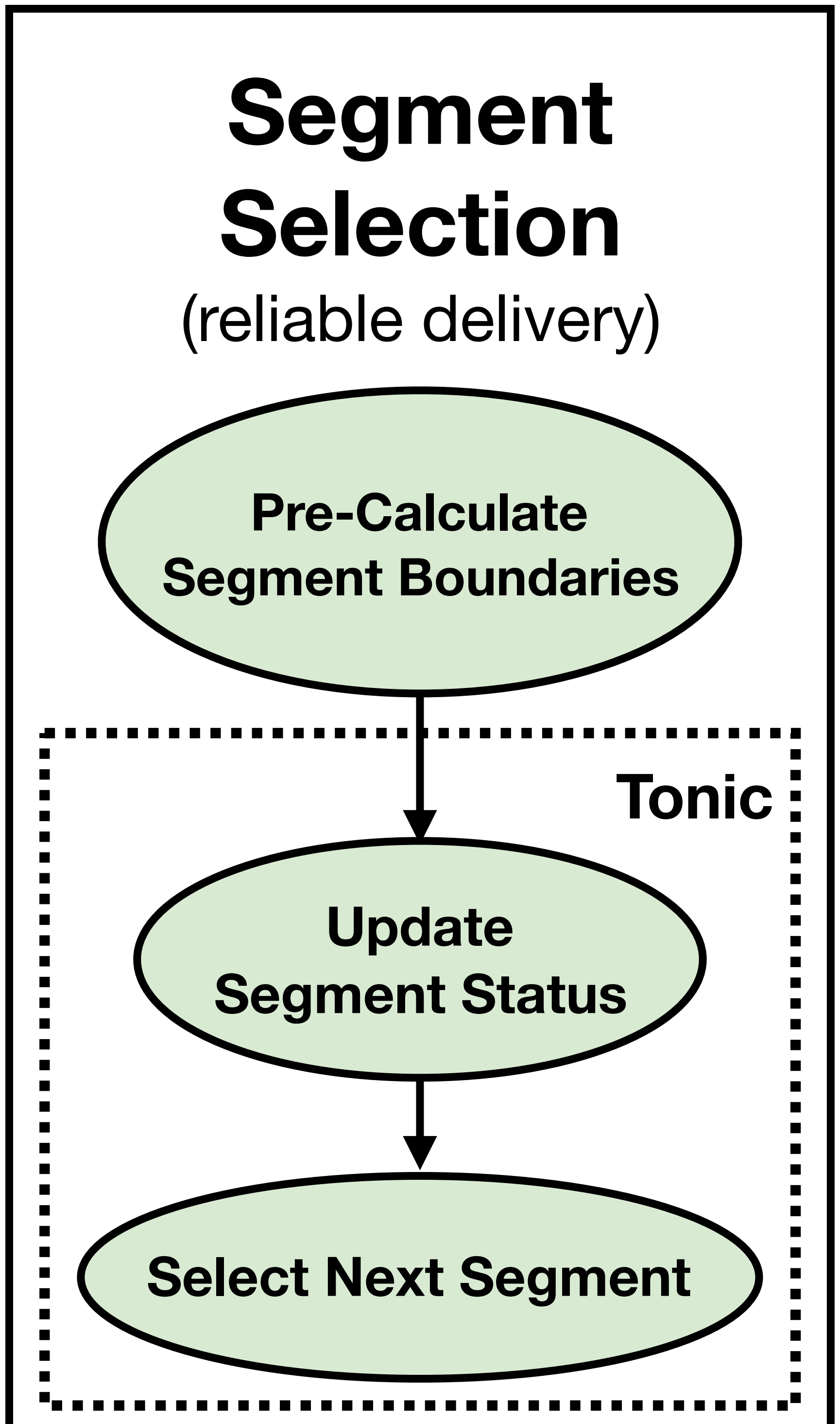
Segment Selection Patterns



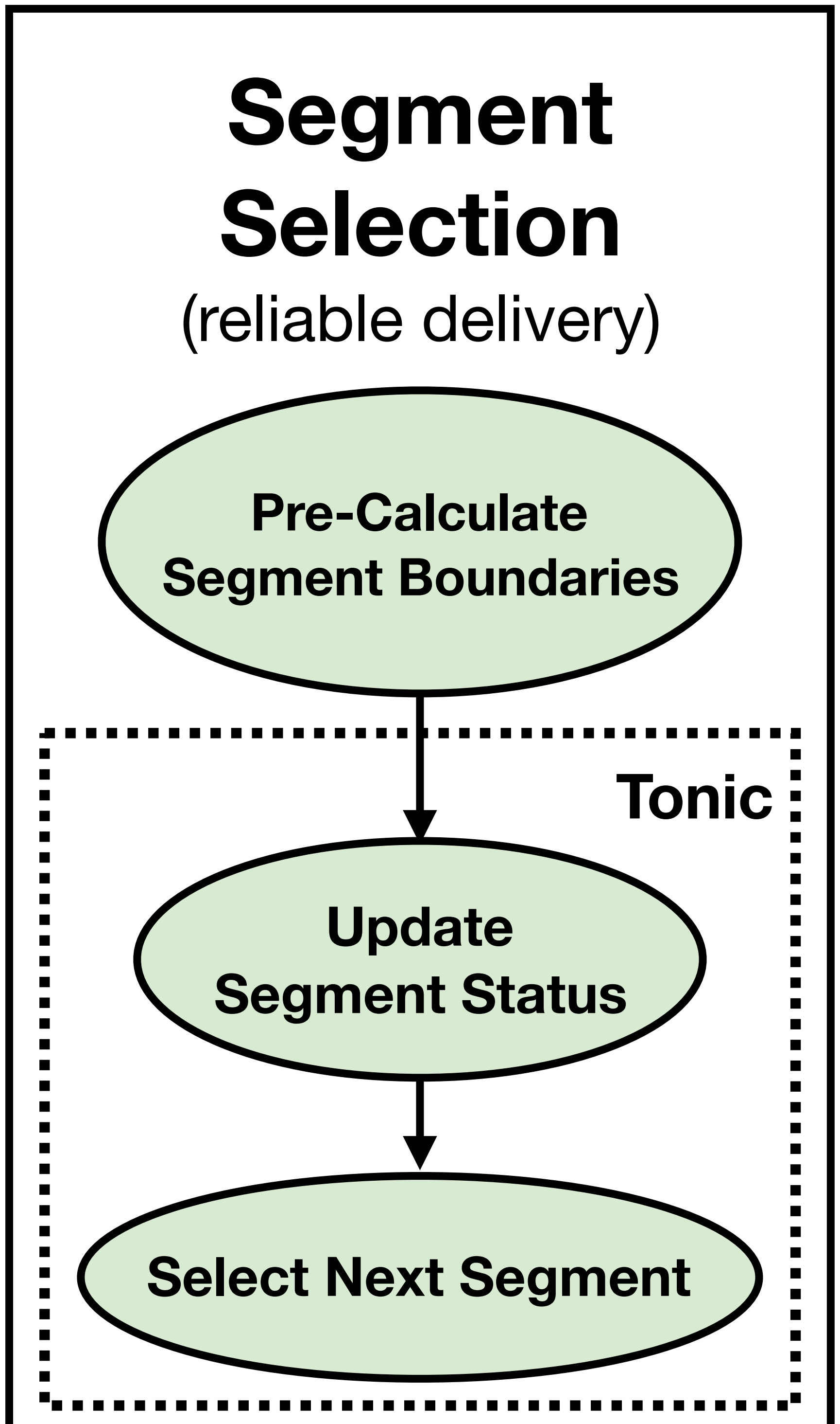
Segment Selection Patterns



Segment Selection Patterns

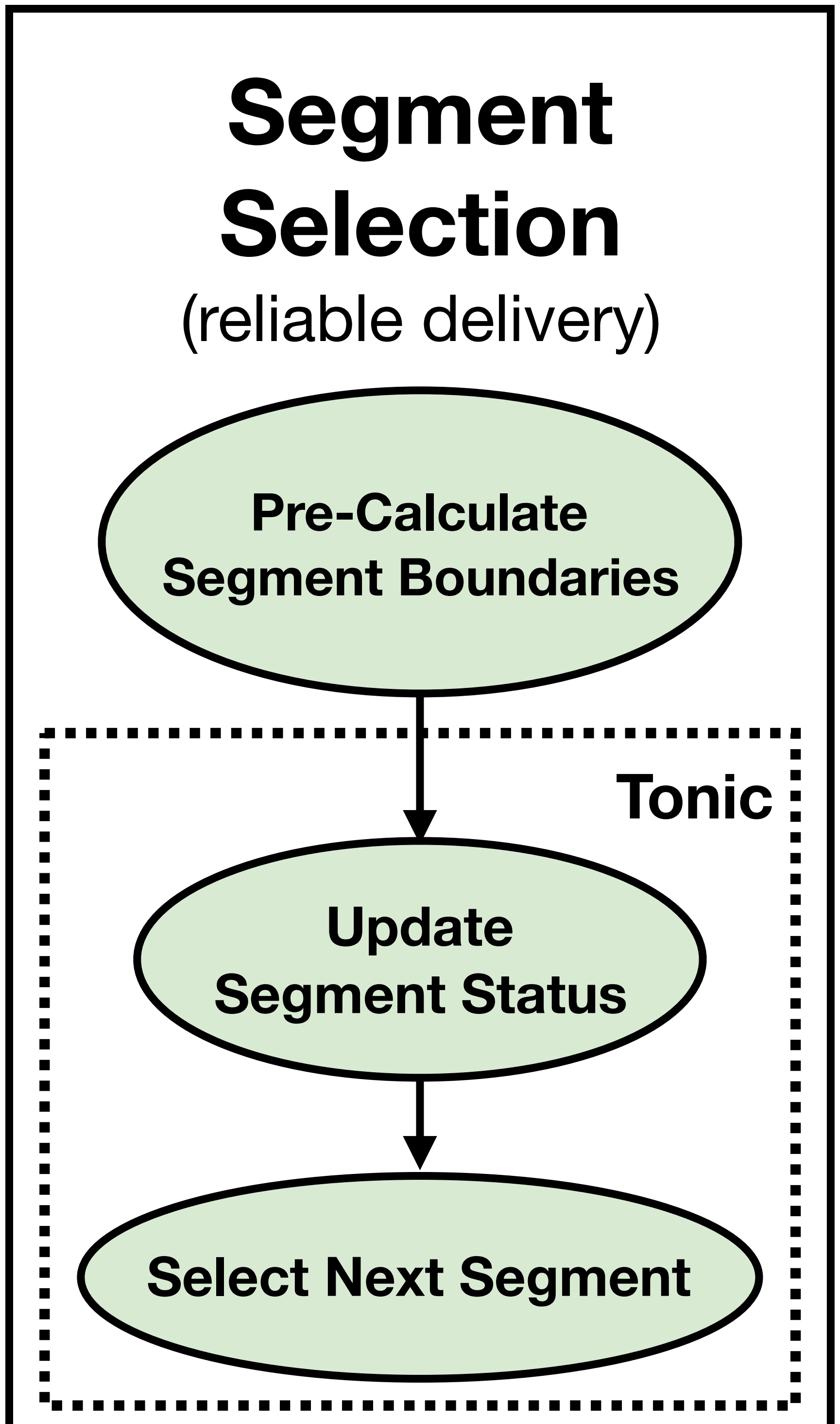


Segment Selection Patterns



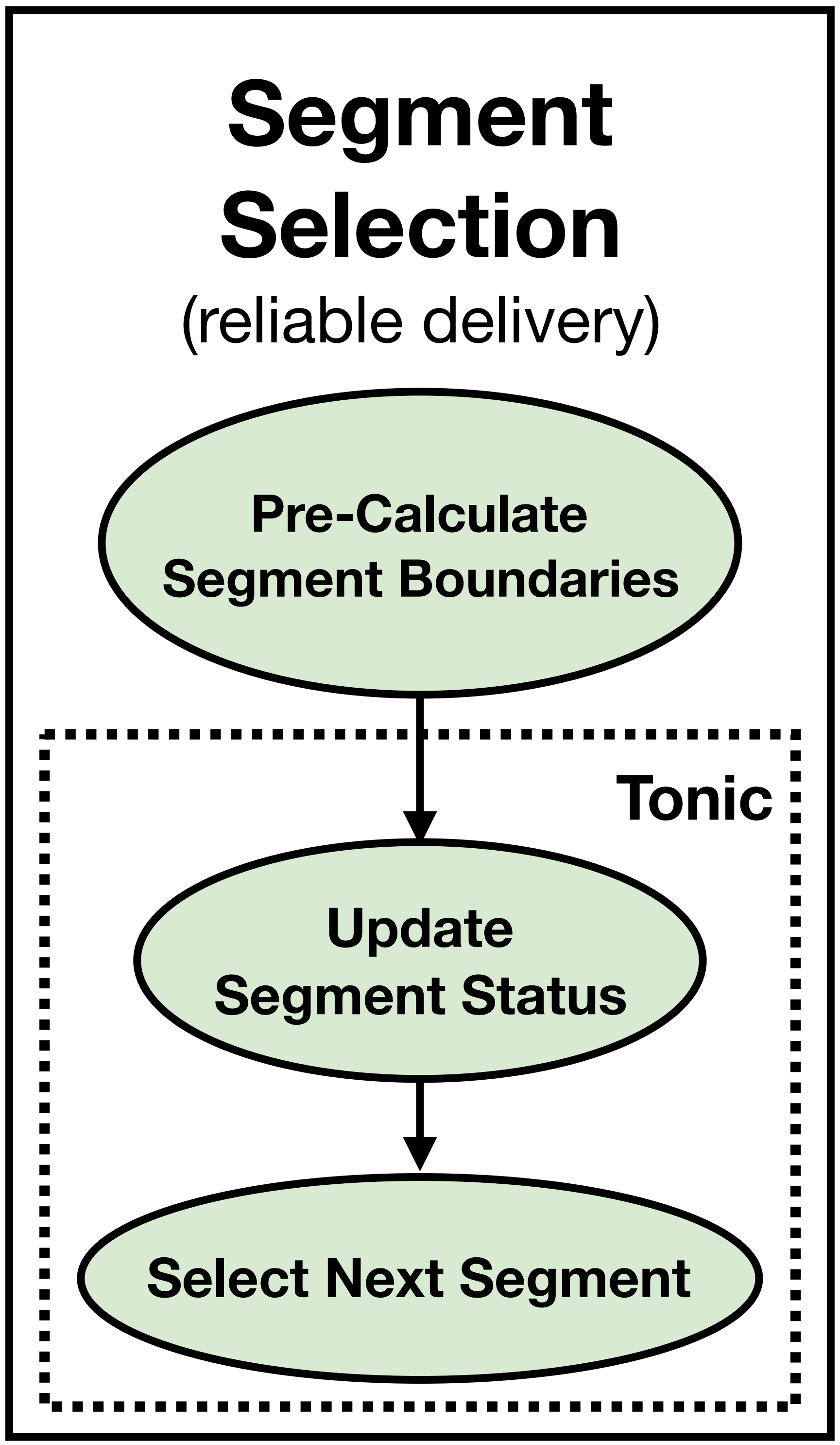
1. **Only a few bits of state per segment**

Segment Selection Patterns



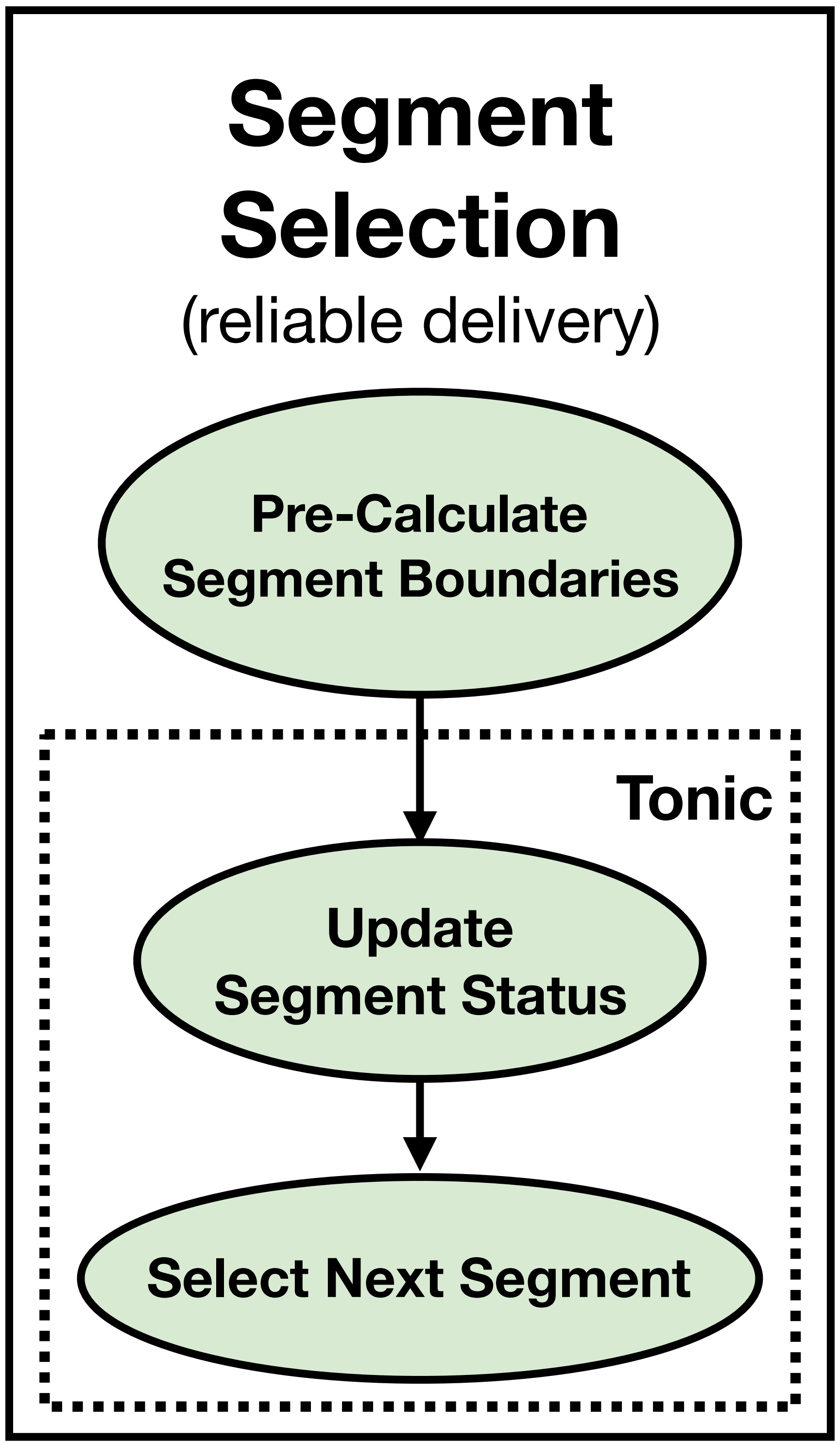
1. **Only a few bits of state per segment**
   - acked, rtxed, lost

Segment Selection Patterns



1. **Only a few bits of state per segment**
   - acked, rtxed, lost
   - fixed function modules for common state updates

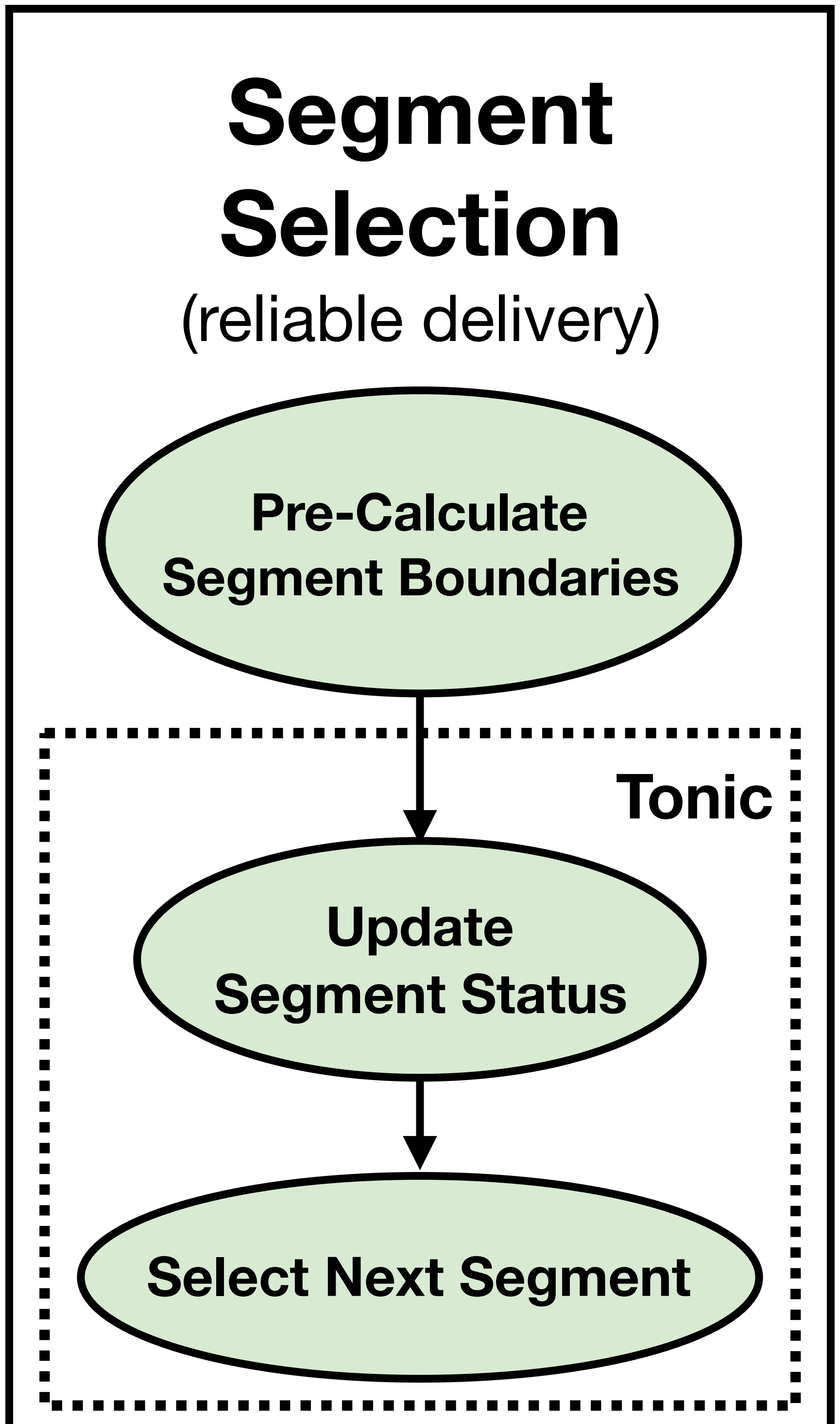
Segment Selection Patterns



1. Only a few bits of state per segment

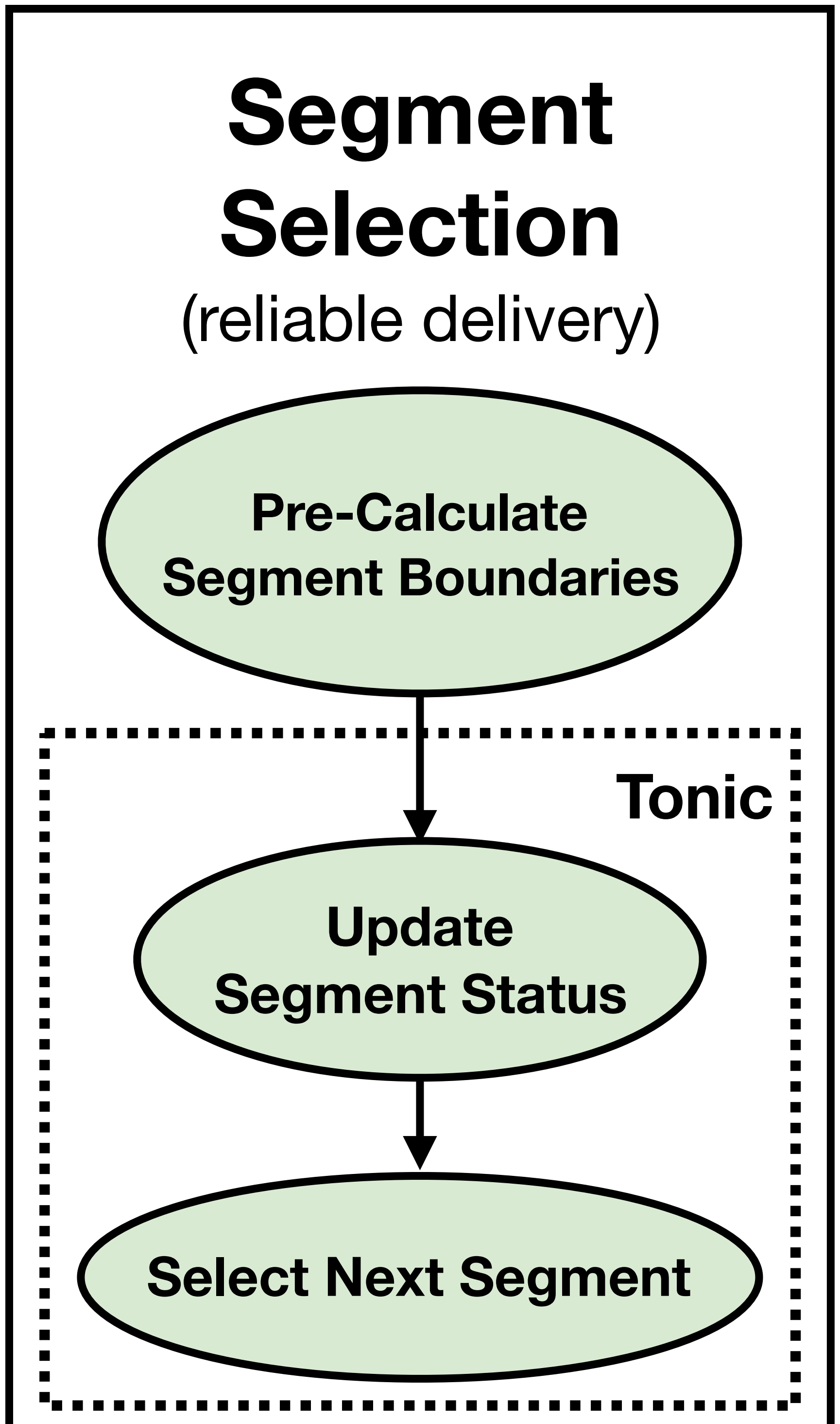
- acked, rtxed, lost
- fixed function modules for common state updates
- programmable modules only for loss detection

Segment Selection Patterns



1. **Only a few bits of state per segment**
    - acked, rtxed, lost
    - fixed function modules for common state updates
    - programmable modules only for loss detection
2. **Loss detection: acks and timeouts**

Segment Selection Patterns



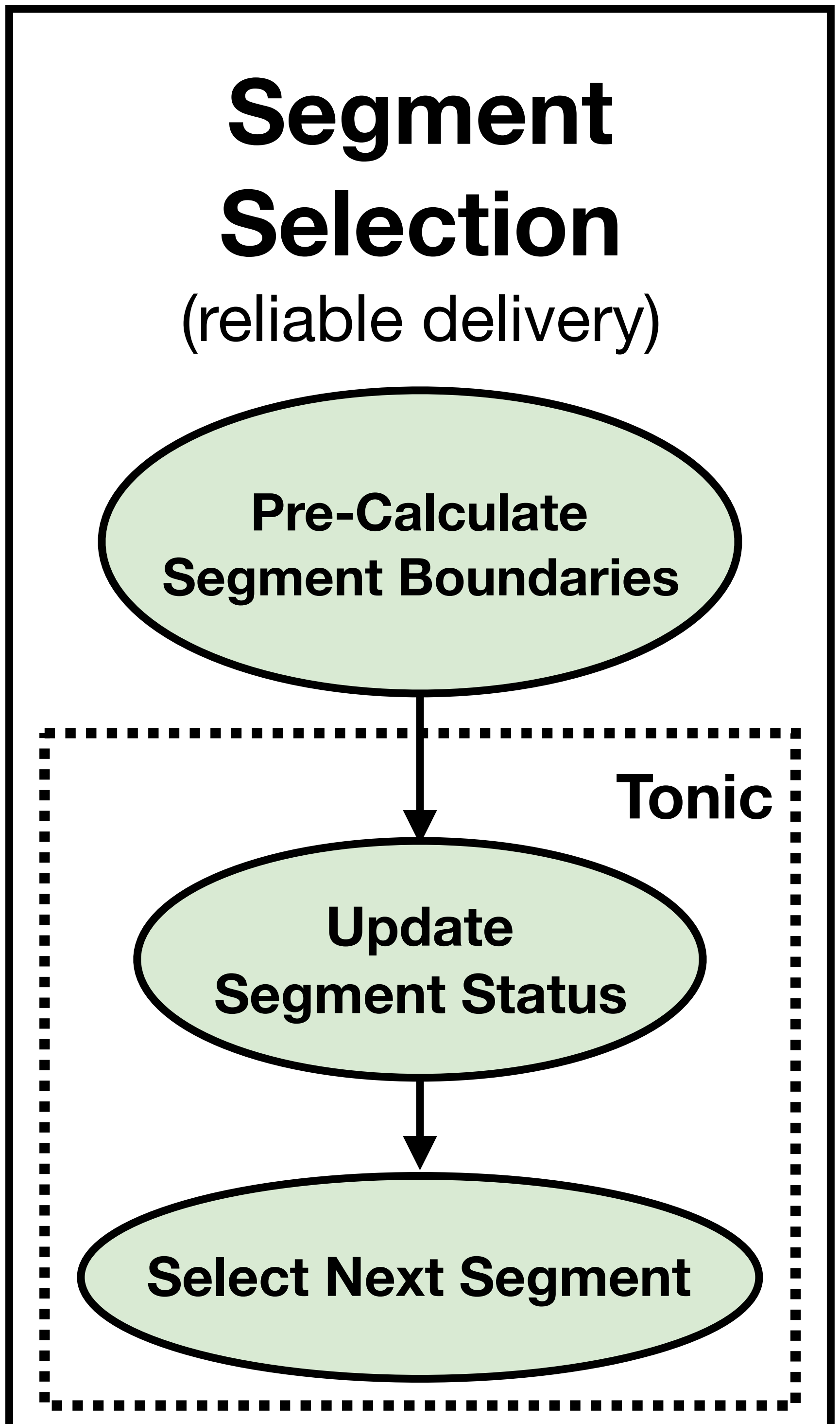
1. Only a few bits of state per segment

- acked, rtxed, lost
- fixed function modules for common state updates
- programmable modules only for loss detection

2. Loss detection: acks and timeouts

- only two programmable modules

Segment Selection Patterns



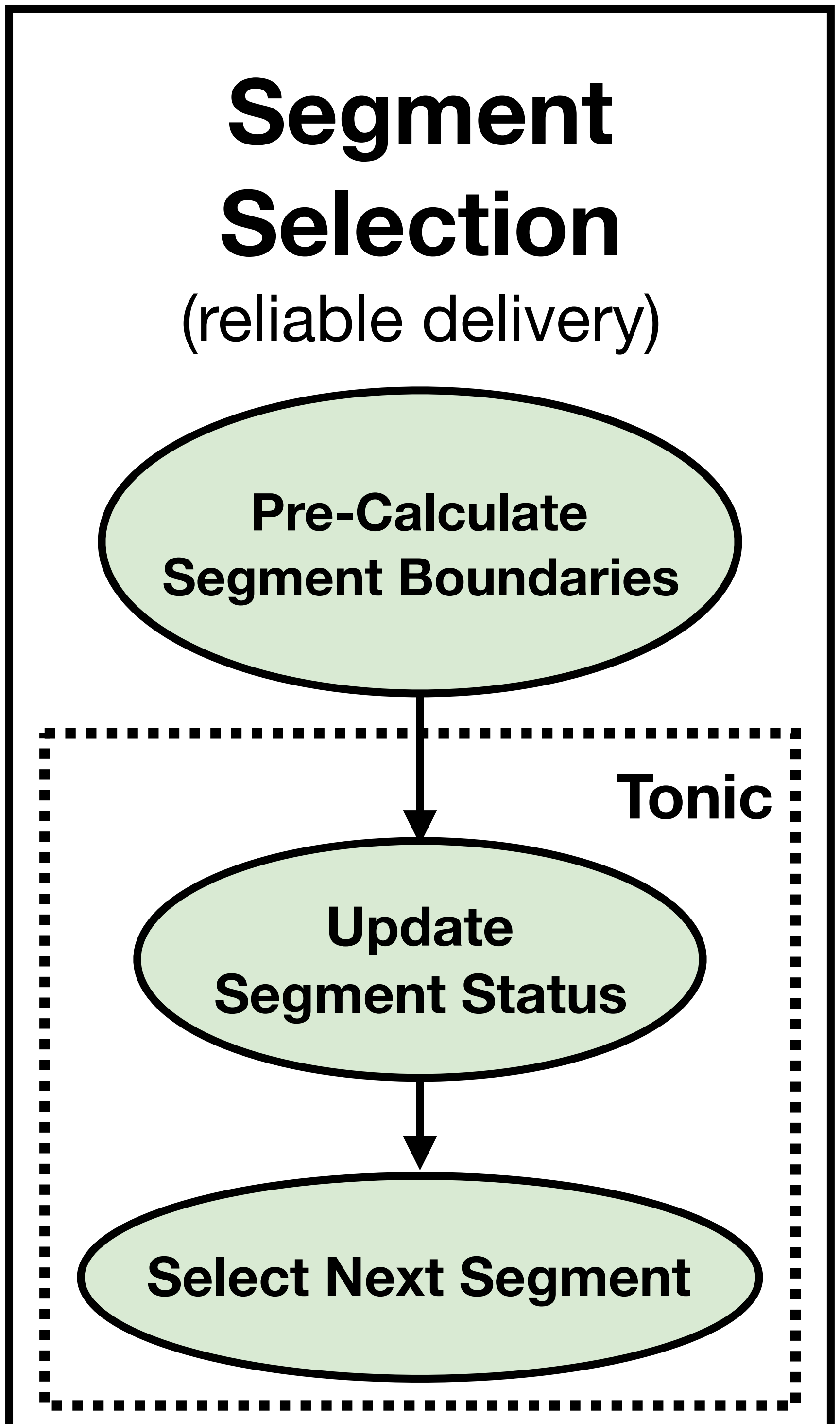
1. Only a few bits of state per segment

- acked, rtxed, lost
- fixed function modules for common state updates
- programmable modules only for loss detection

2. Loss detection: acks and timeouts

- only two programmable modules
- mutually exclusive → fewer concurrent state updates

Segment Selection Patterns



1. Only a few bits of state per segment

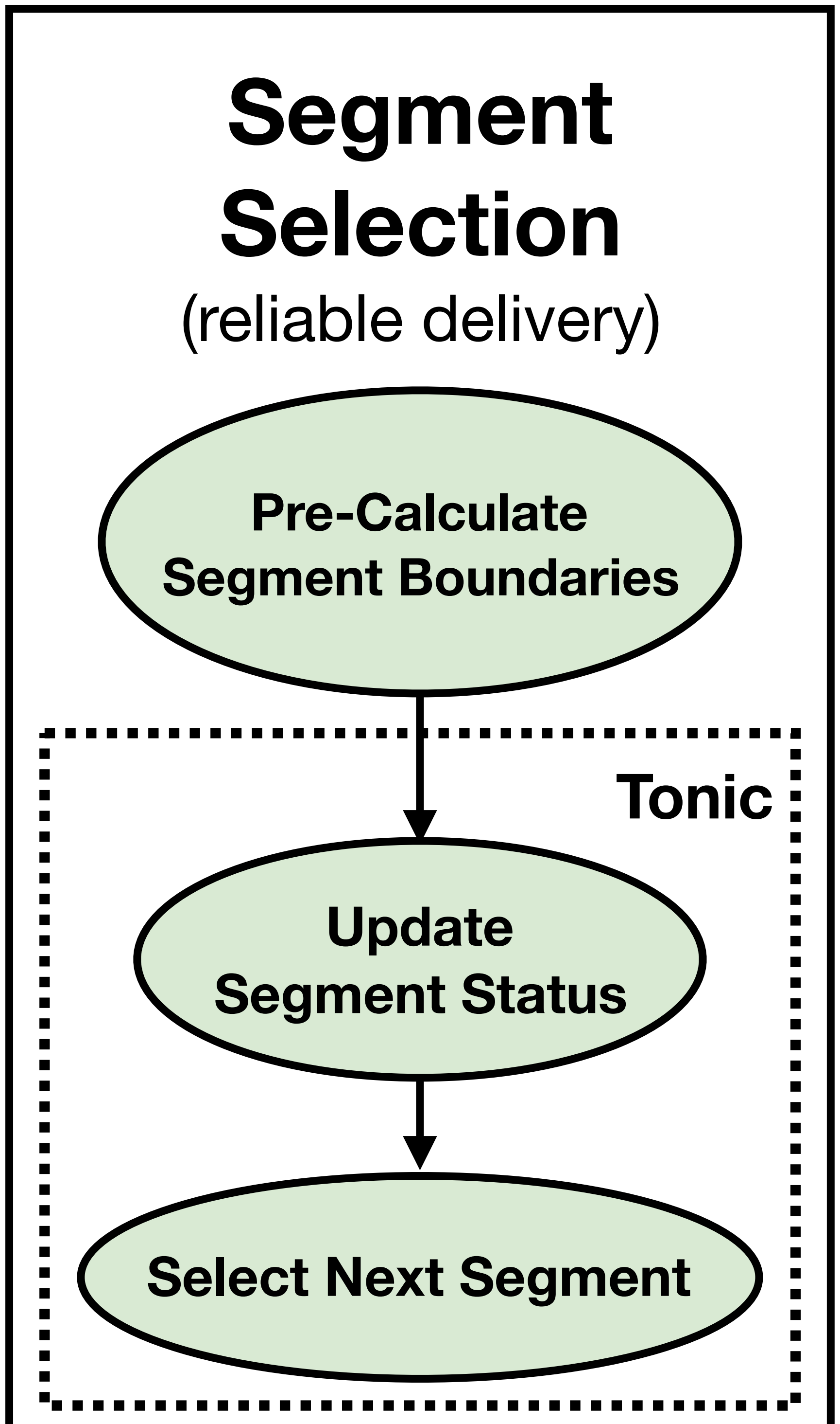
- acked, rtxed, lost
- fixed function modules for common state updates
- programmable modules only for loss detection

2. Loss detection: acks and timeouts

- only two programmable modules
- mutually exclusive → fewer concurrent state updates

3. Lost segments first, new segments next

Segment Selection Patterns



1. Only a few bits of state per segment

- acked, rtxed, lost
- fixed function modules for common state updates
- programmable modules only for loss detection

2. Loss detection: acks and timeouts

- only two programmable modules
- mutually exclusive → fewer concurrent state updates

3. Lost segments first, new segments next

- fixed-function module for segment generation

Concurrent State Update

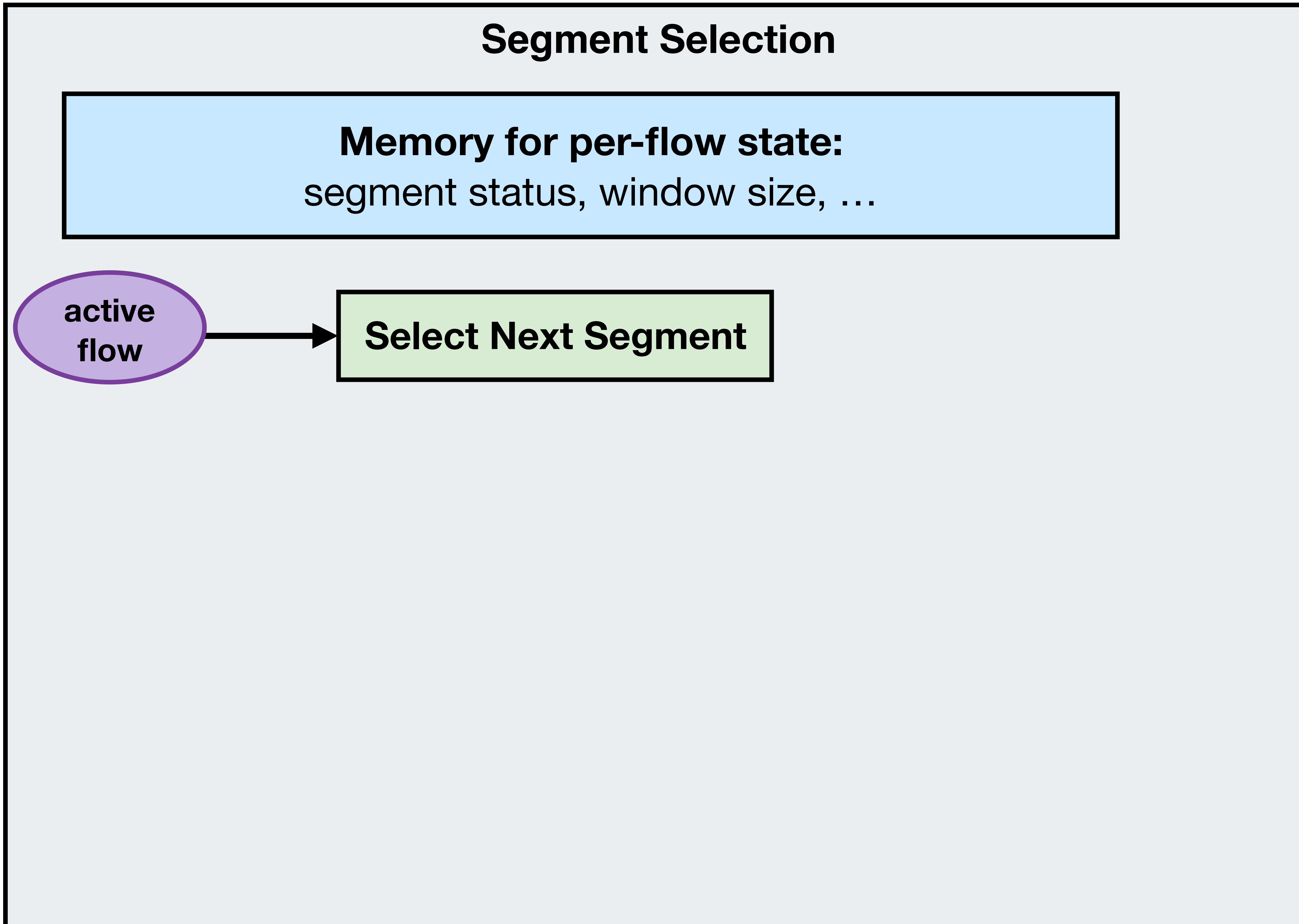
Segment Selection

Concurrent State Update

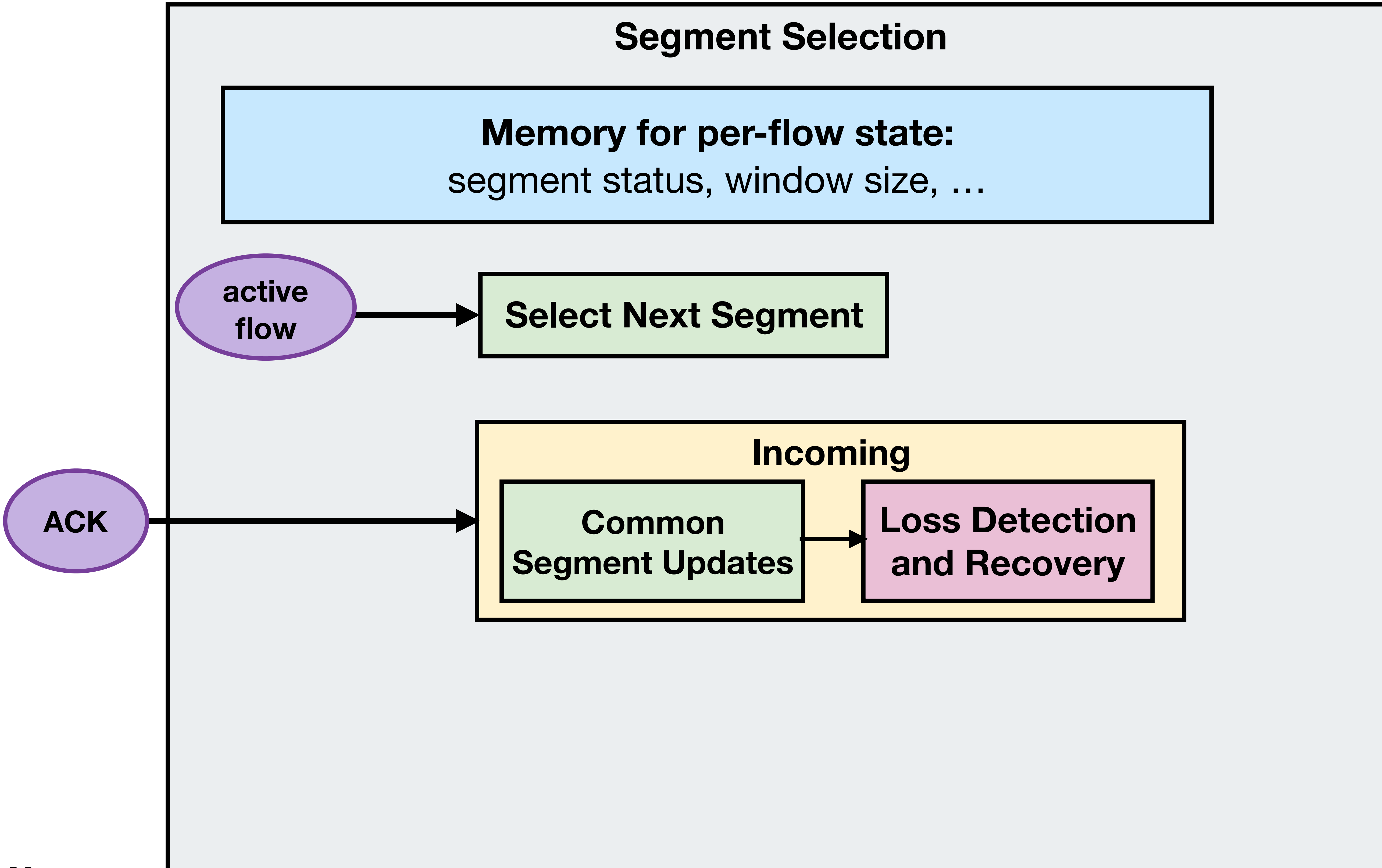
Segment Selection

Memory for per-flow state:
segment status, window size, …

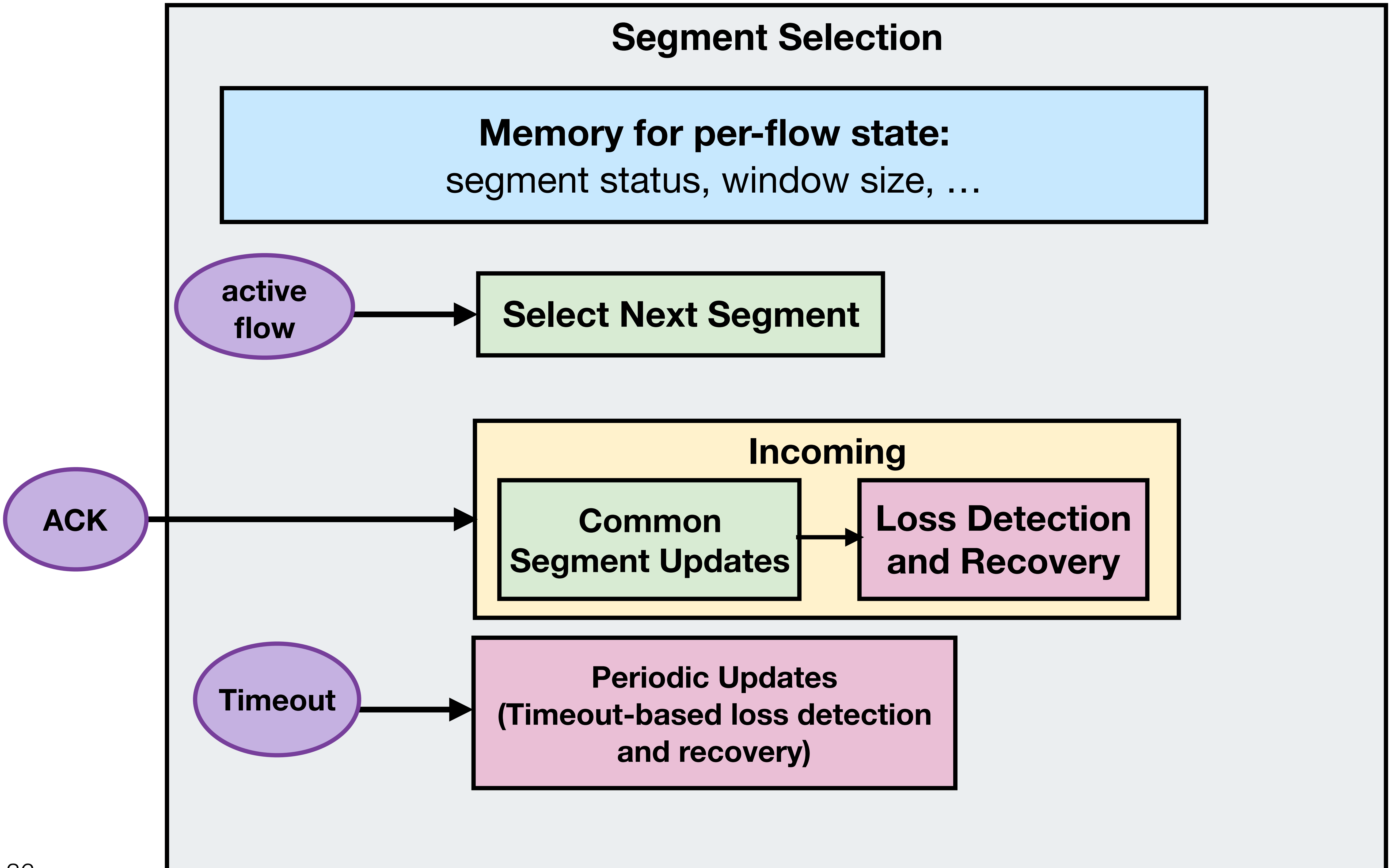
Concurrent State Update



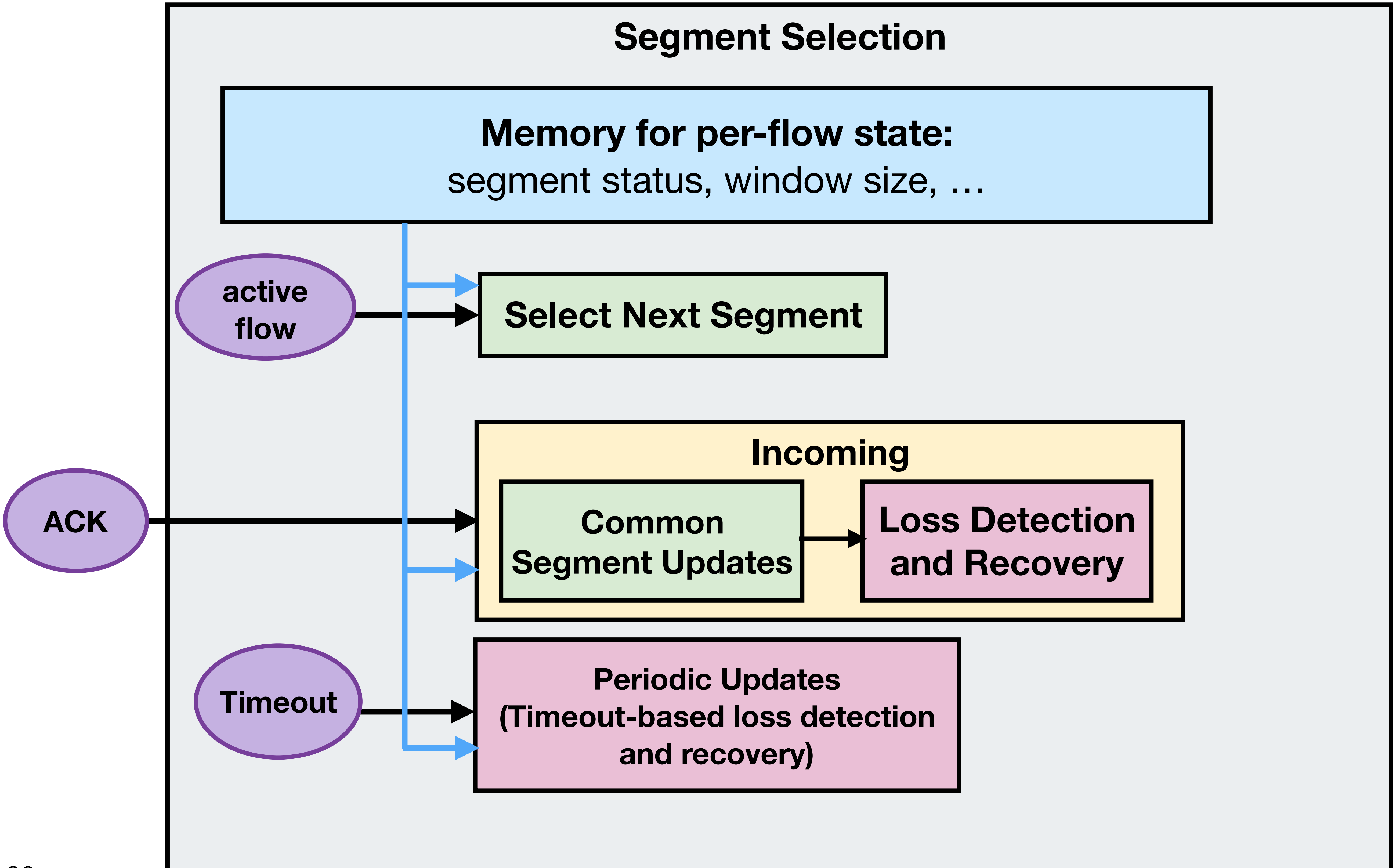
Concurrent State Update



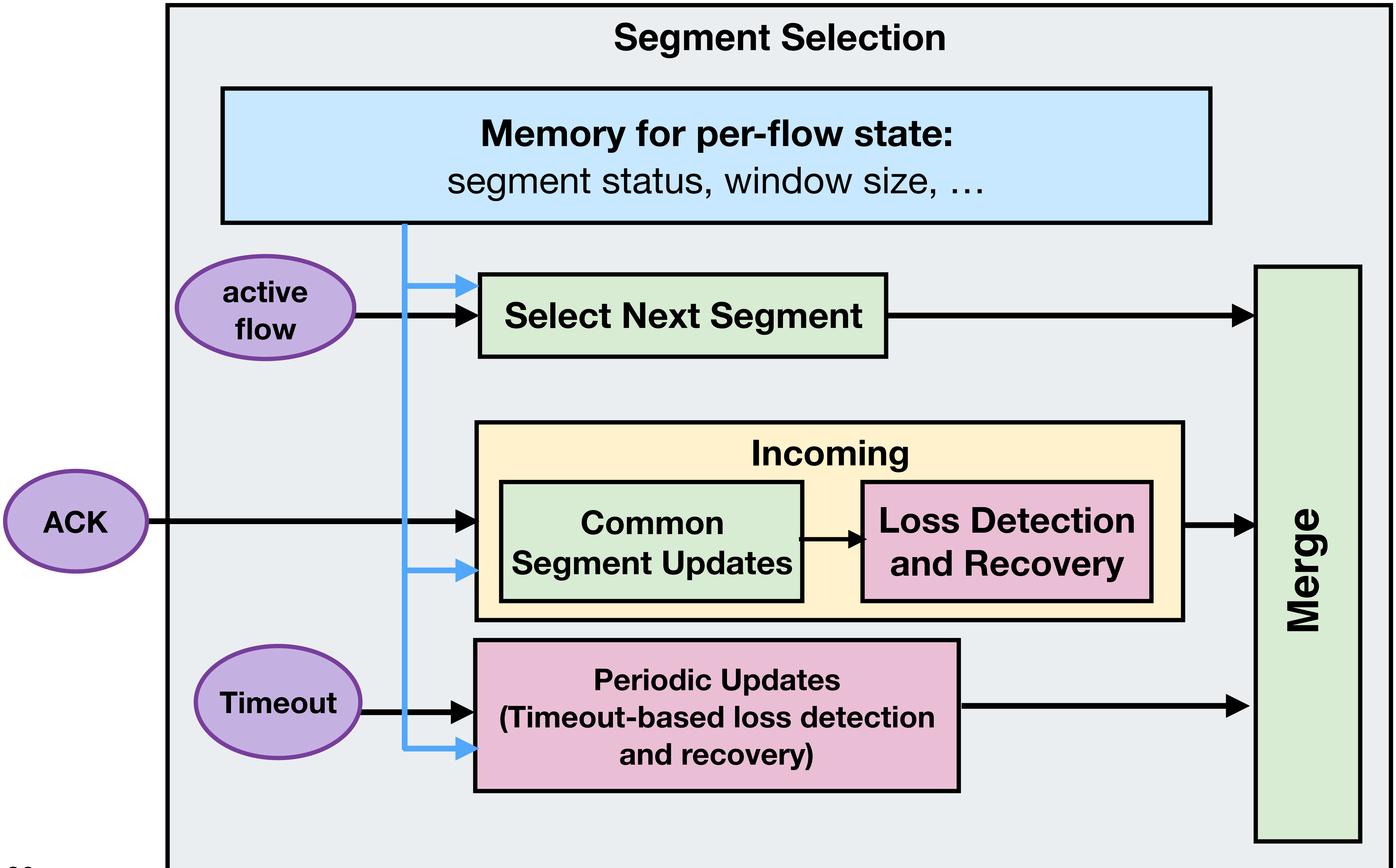
Concurrent State Update



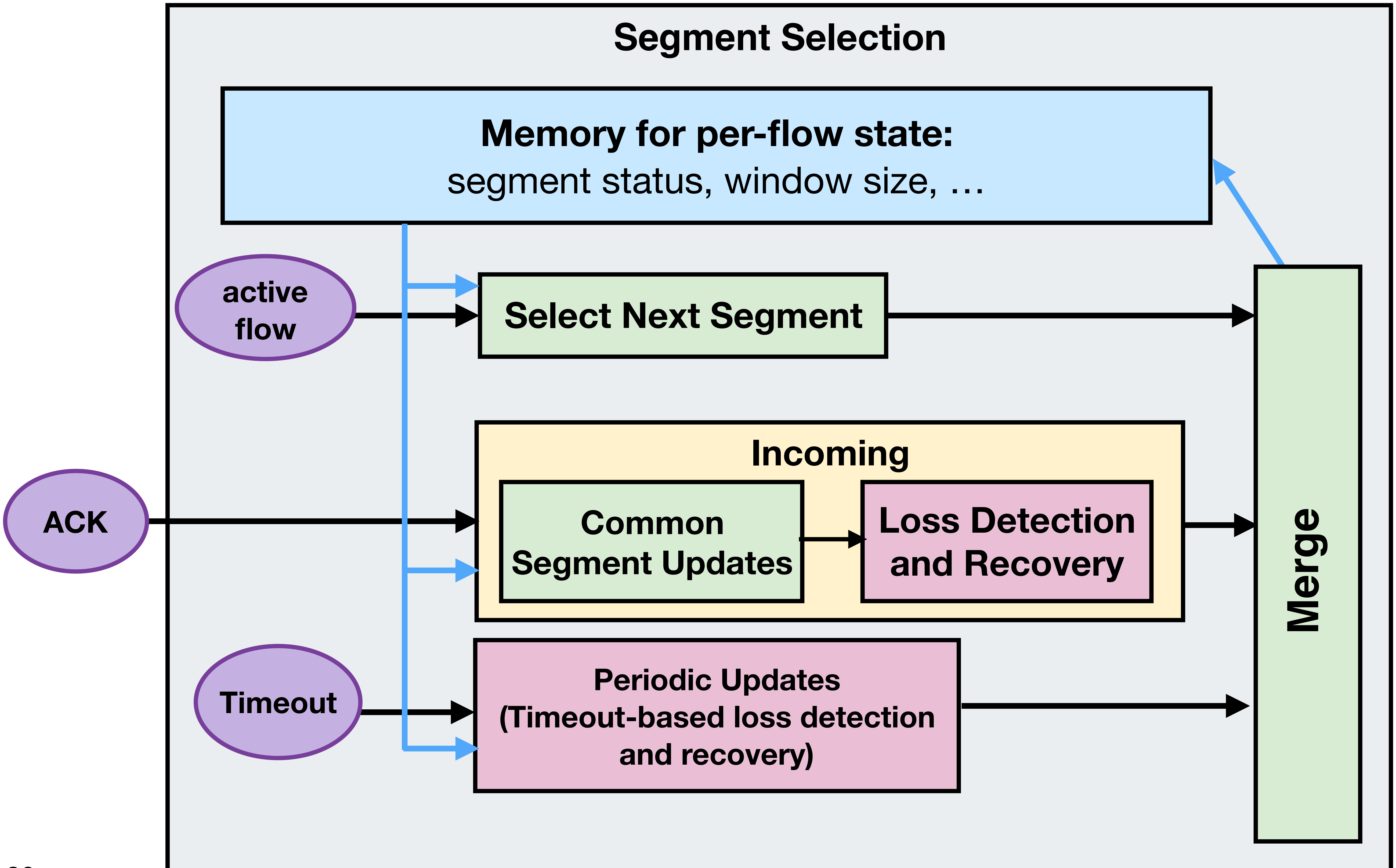
Concurrent State Update



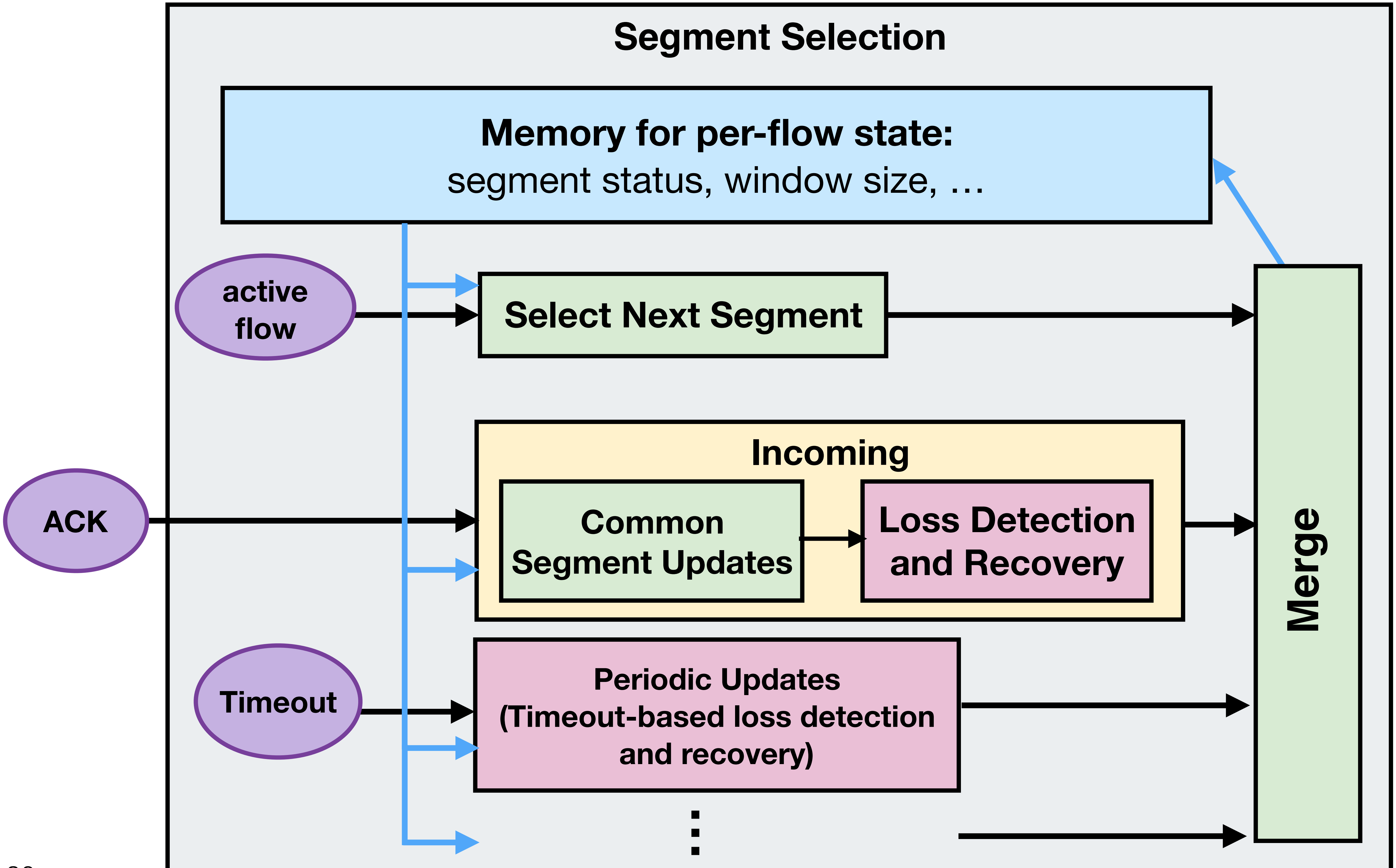
Concurrent State Update



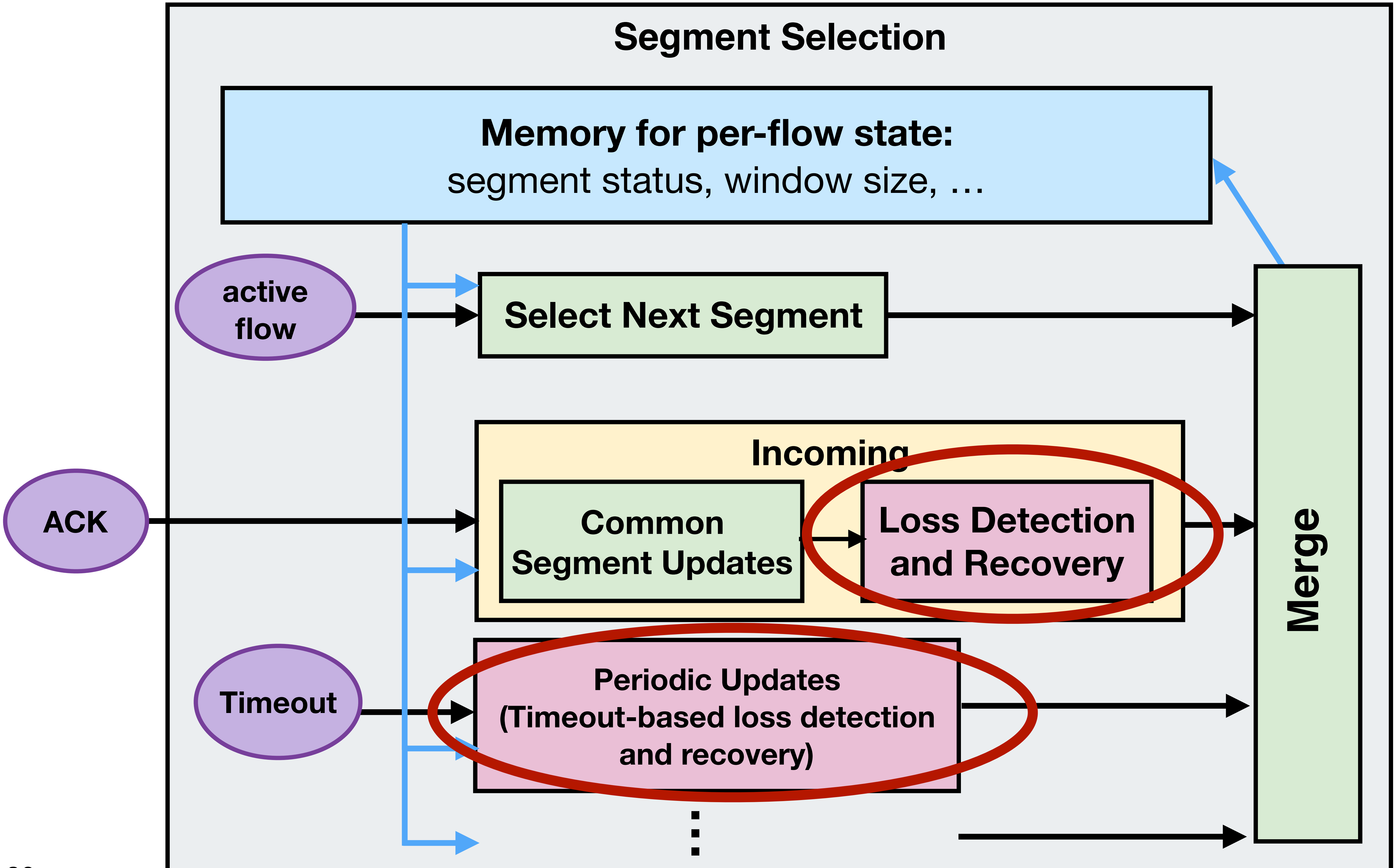
Concurrent State Update



Concurrent State Update



Concurrent State Update

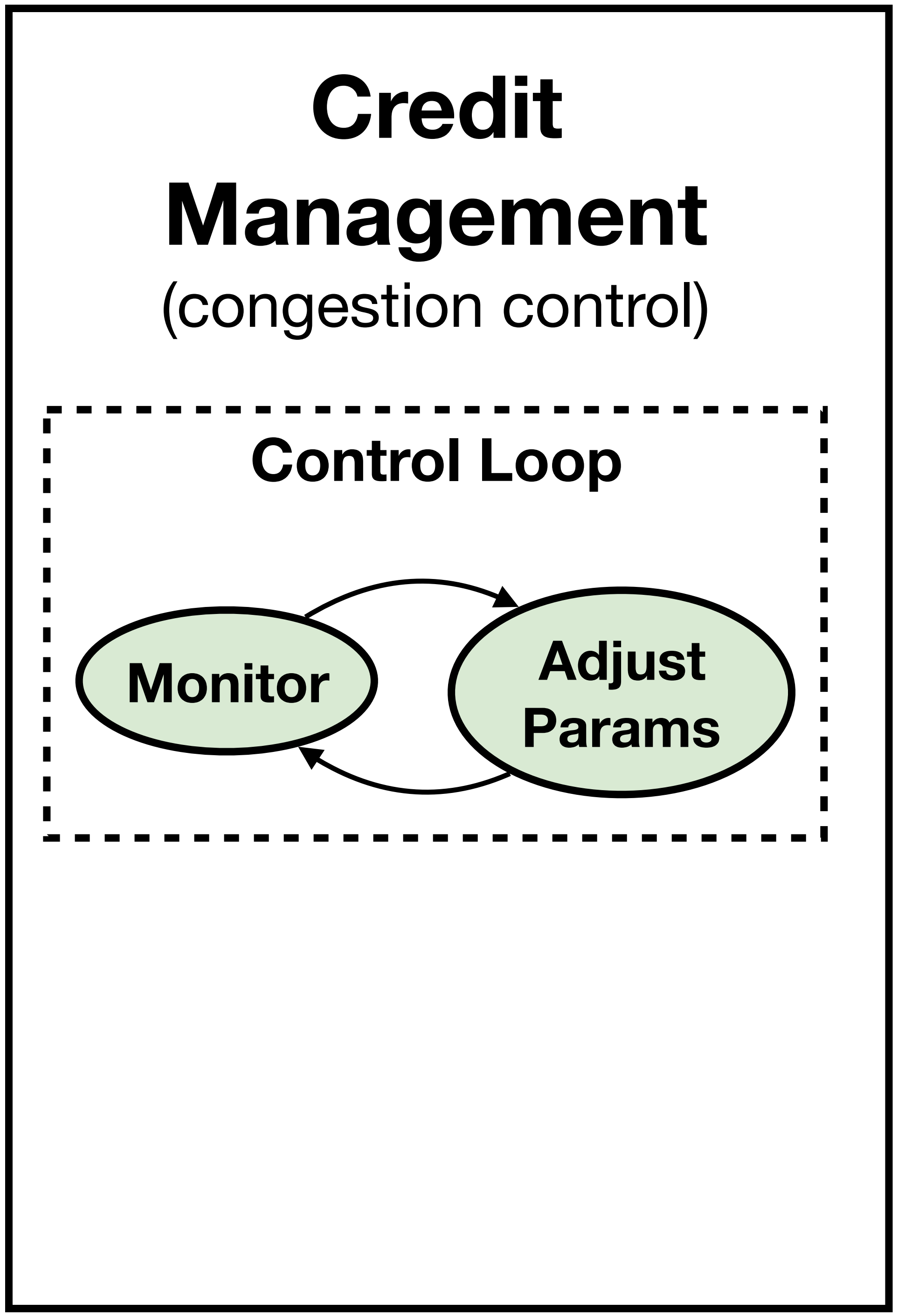


Credit Management Patterns

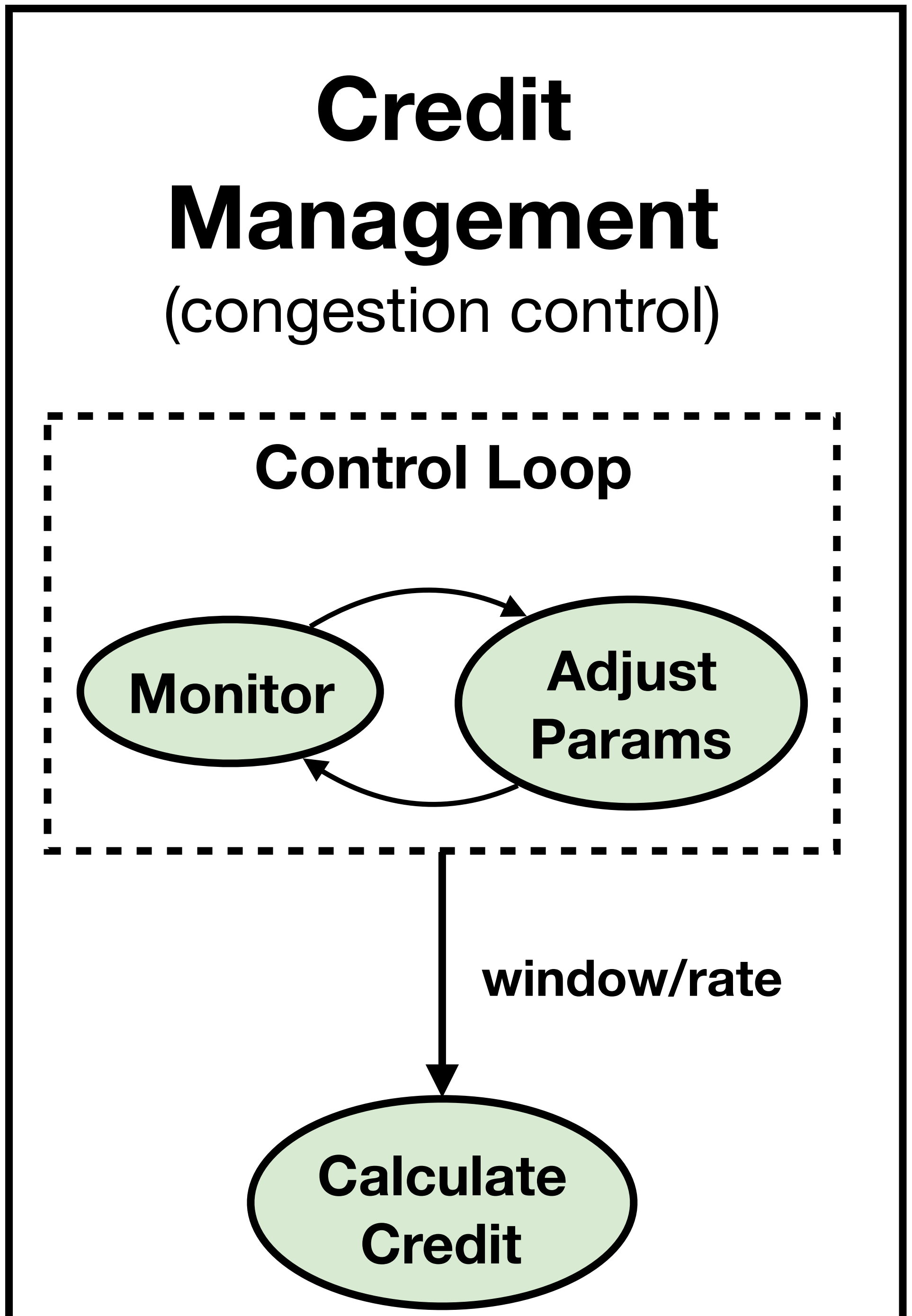
Credit Management Patterns

Credit Management
(congestion control)

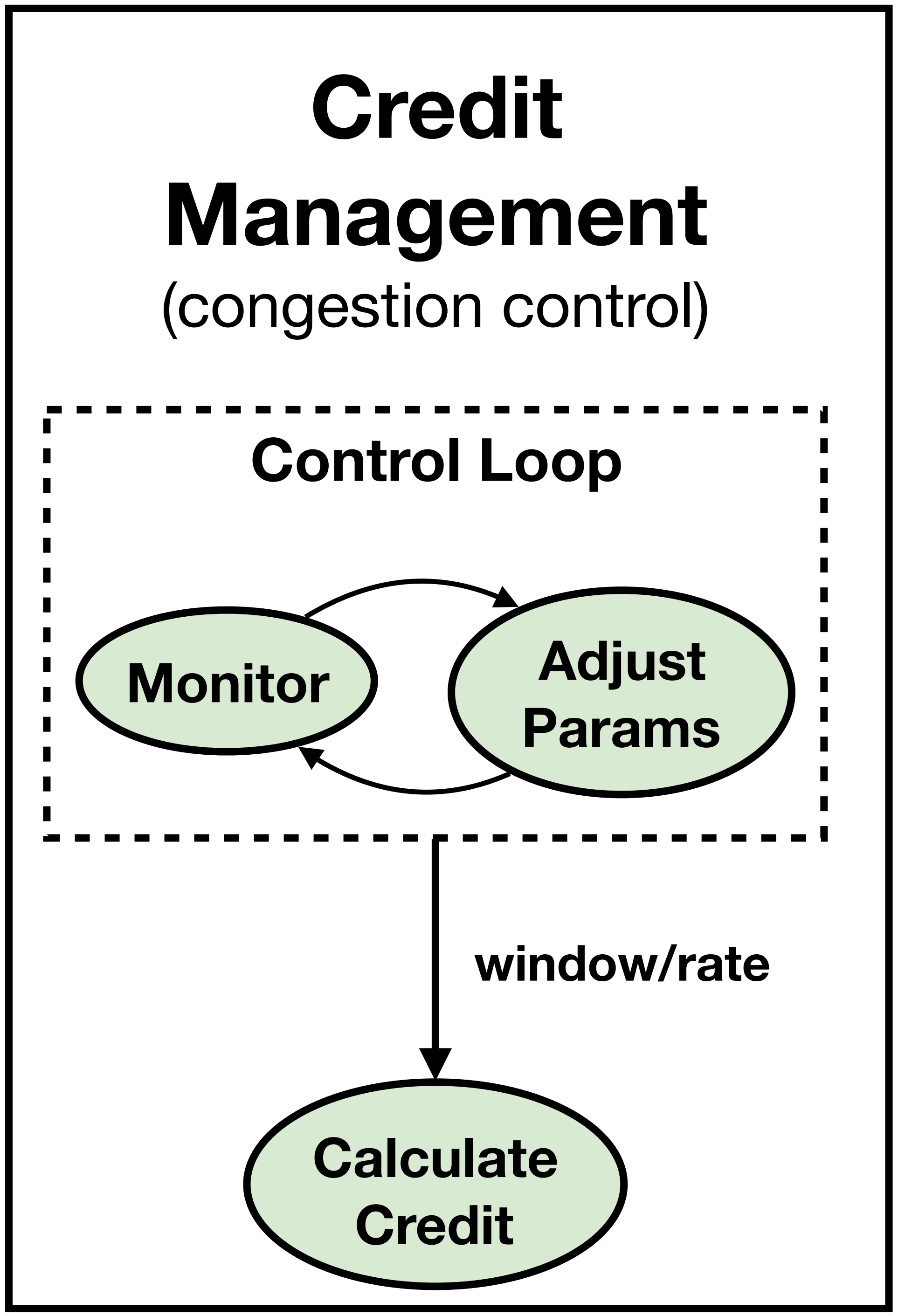
Credit Management Patterns



Credit Management Patterns

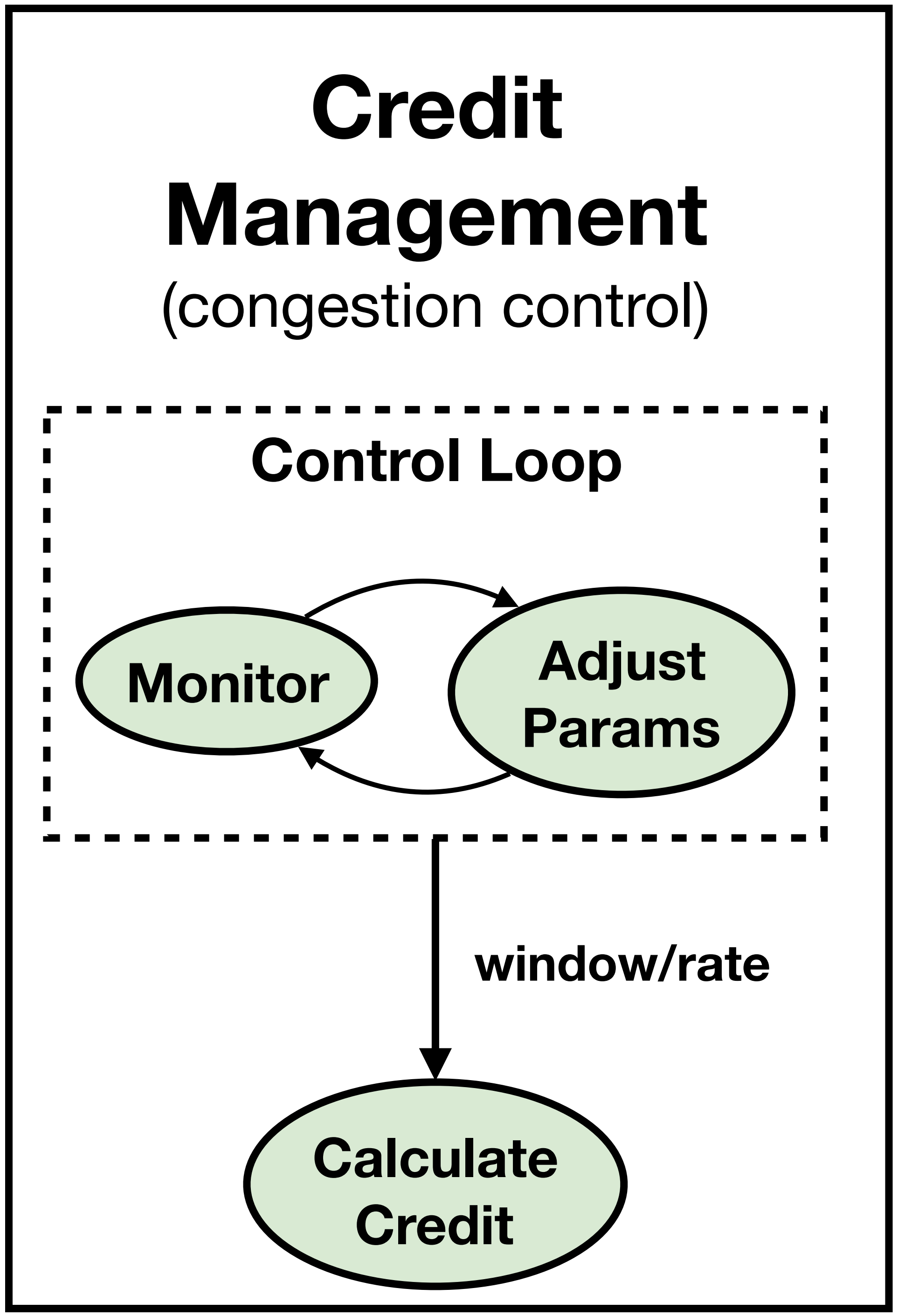


Credit Management Patterns



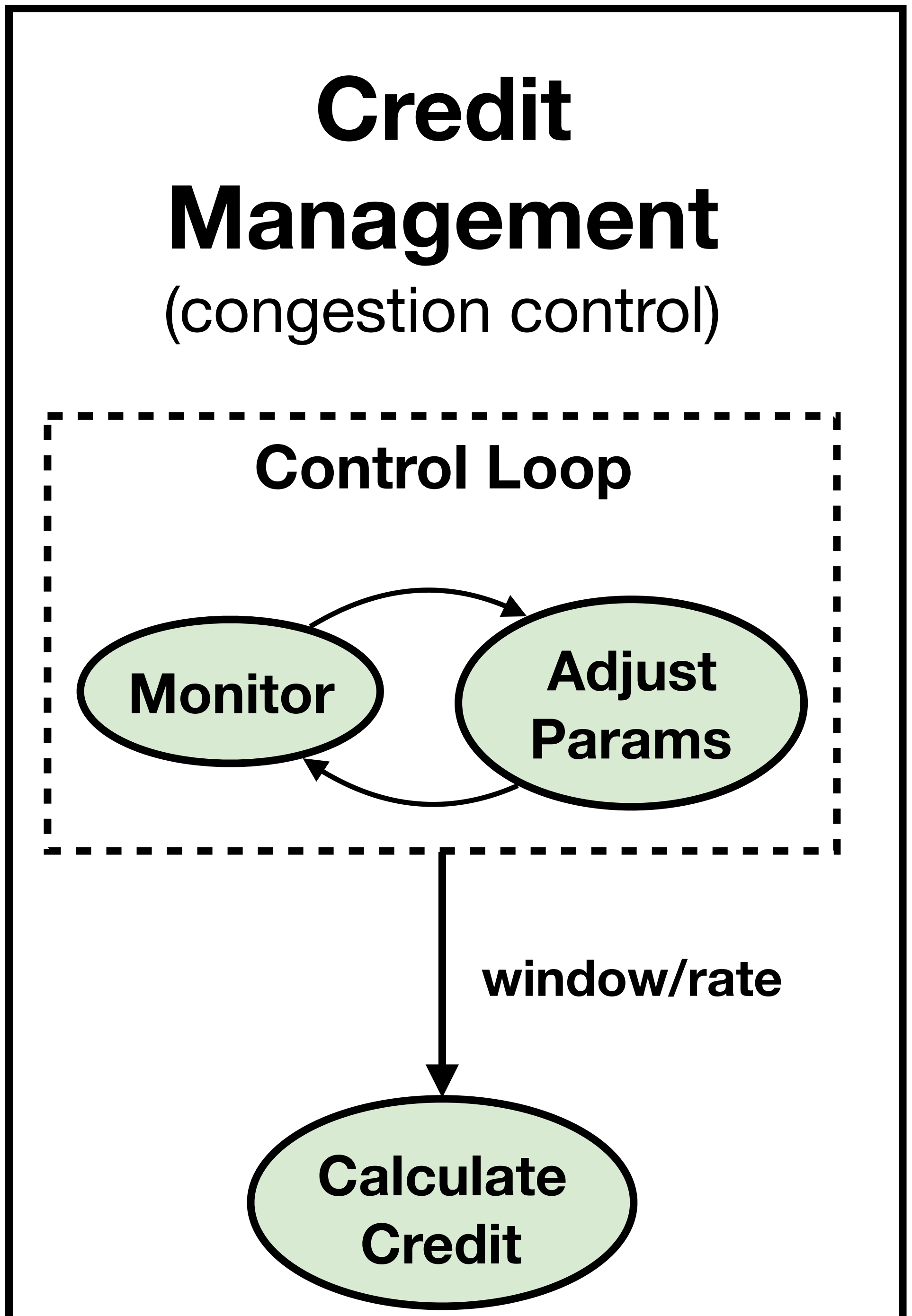
1. **Three common credit calculation schemes**

Credit Management Patterns



1. **Three common credit calculation schemes**
   - congestion window, rate, grant tokens

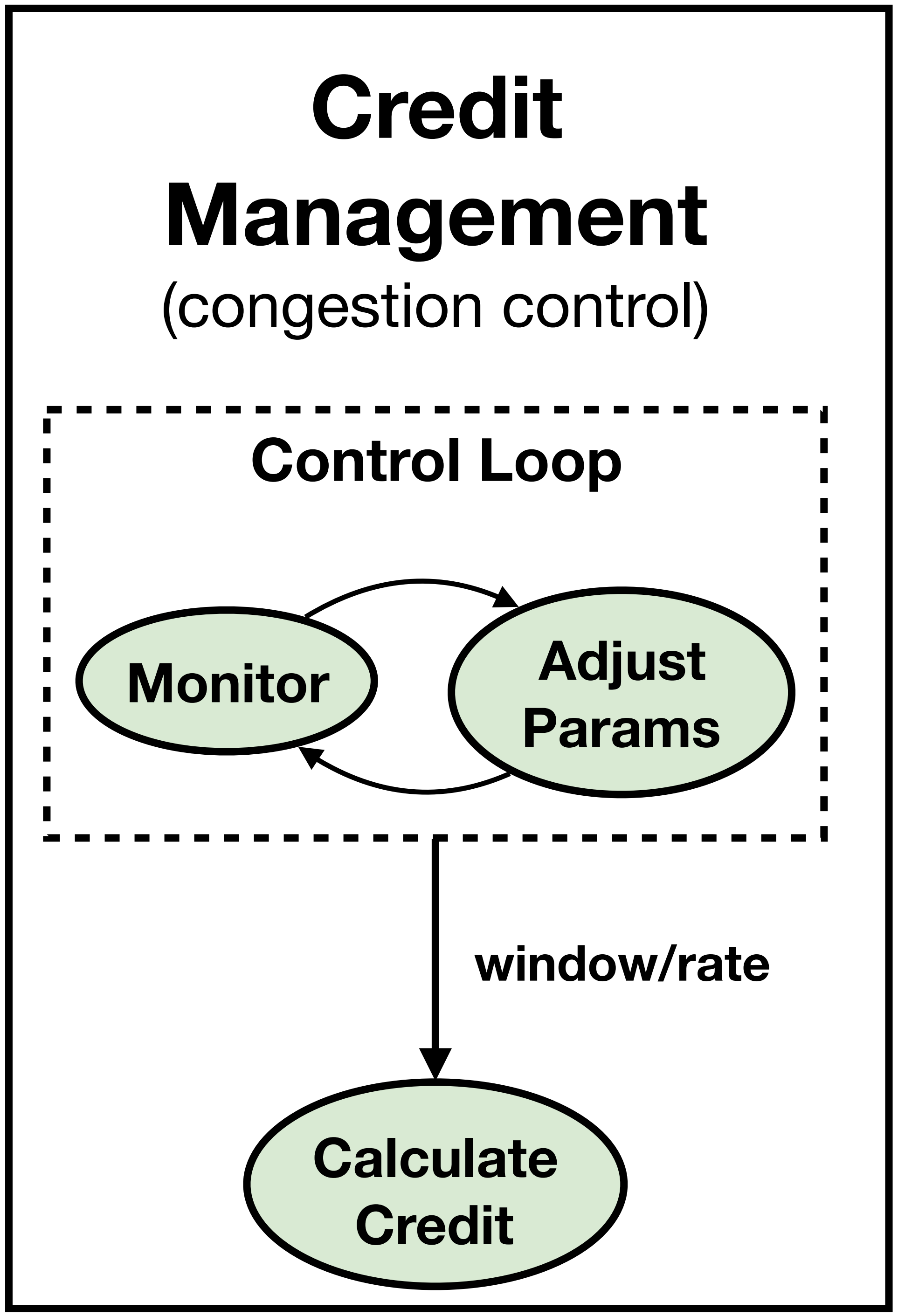
Credit Management Patterns



1. **Three common credit calculation schemes**
   - congestion window, rate, grant tokens

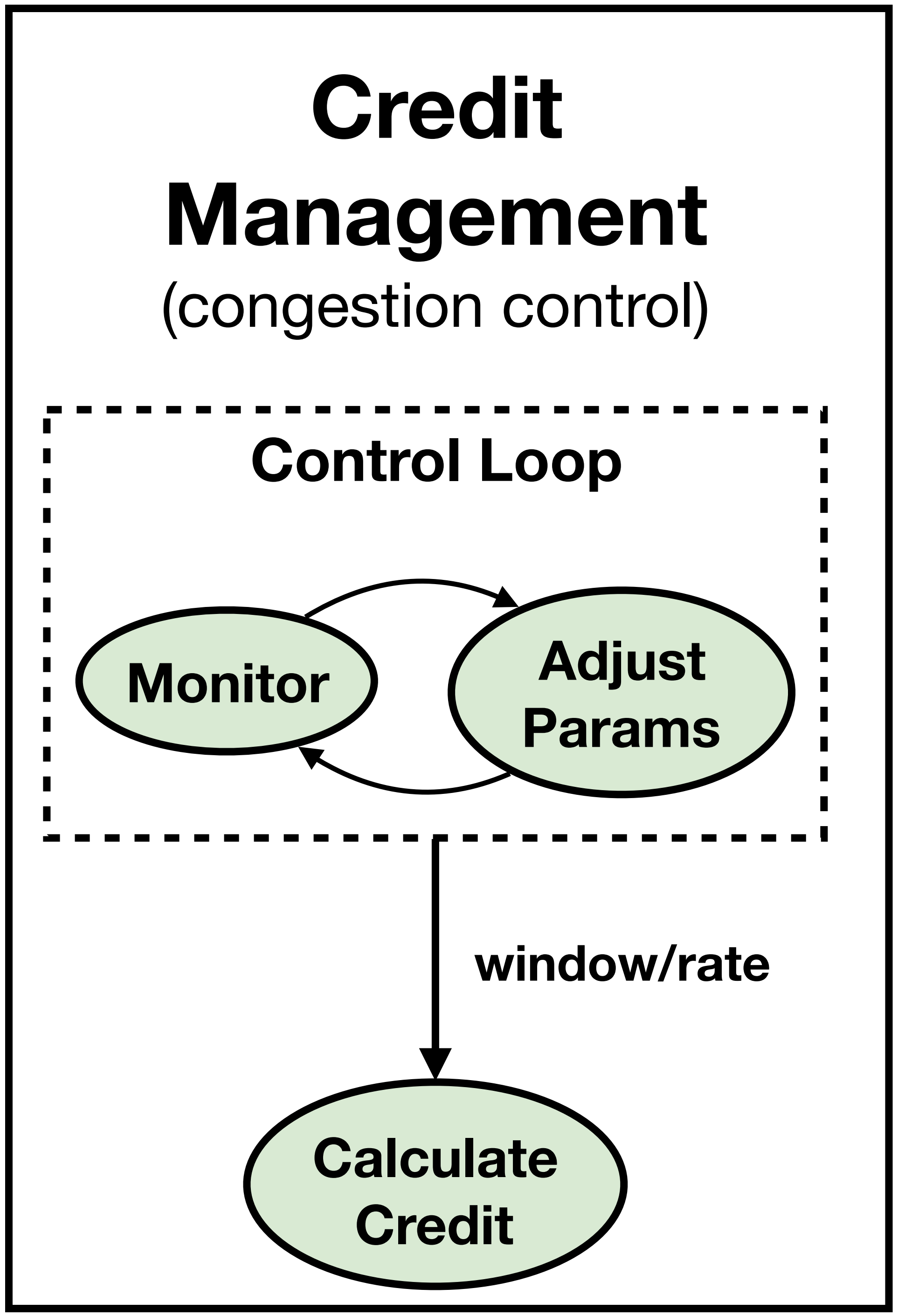
2. **Two main parameter adjustment signals**

Credit Management Patterns



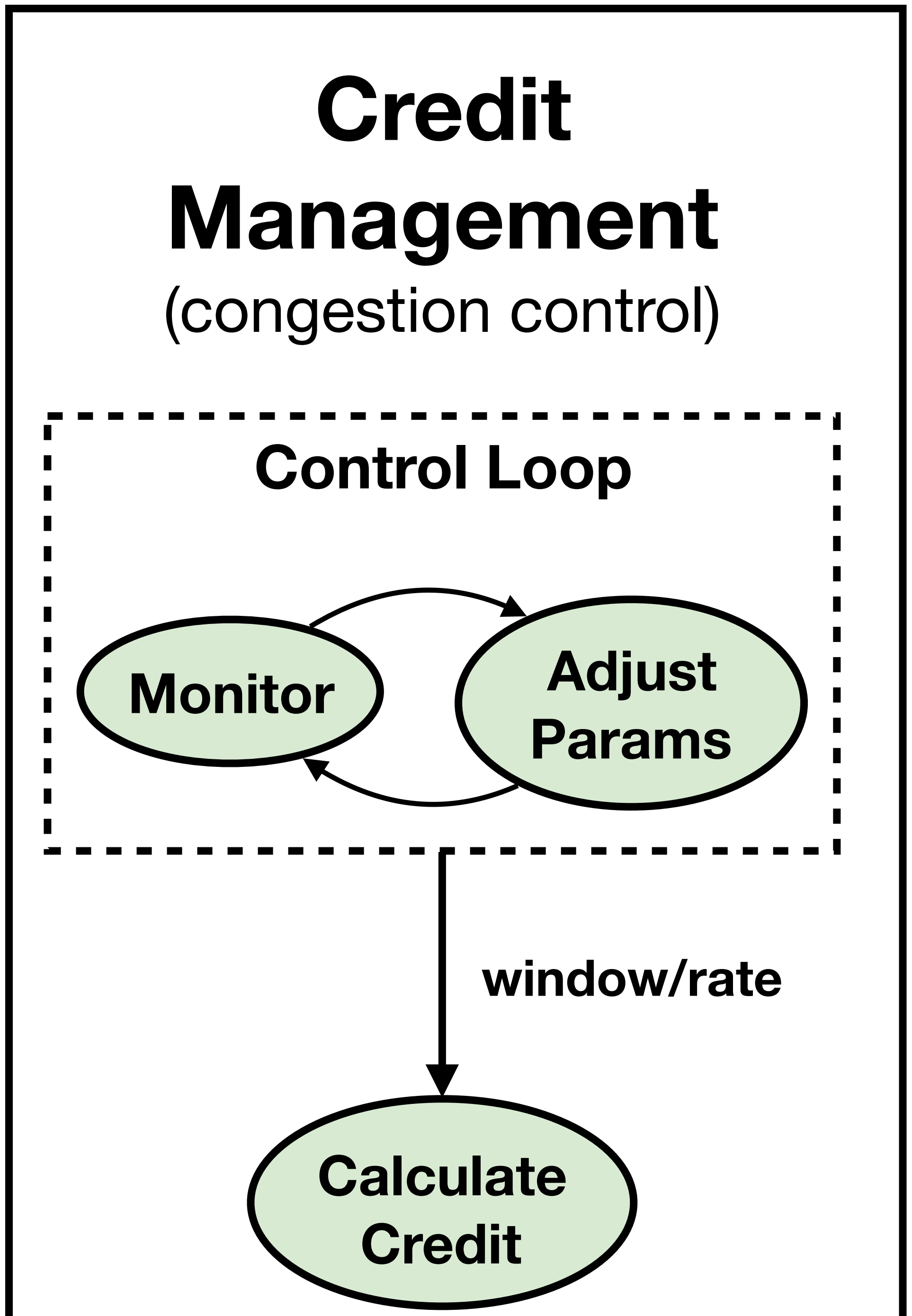
1. **Three common credit calculation schemes**
   - congestion window, rate, grant tokens
2. **Two main parameter adjustment signals**
   - external signals, e.g., acks and CNPs

Credit Management Patterns



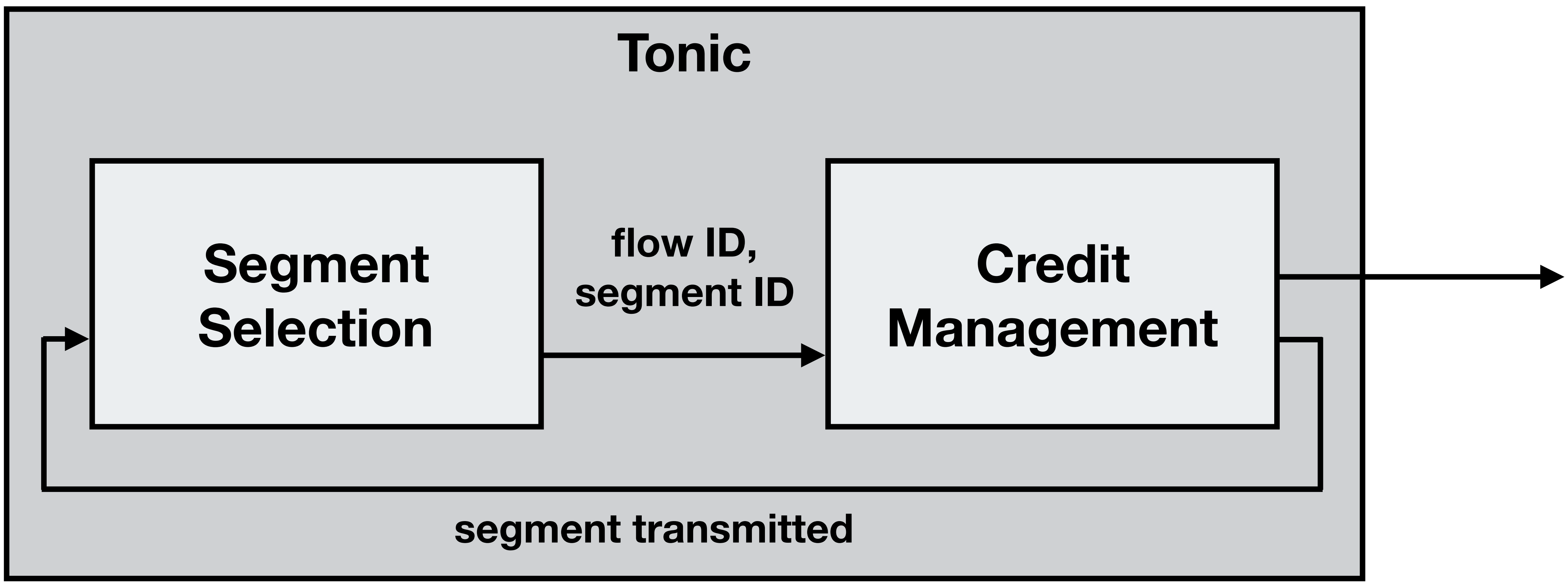
1. **Three common credit calculation schemes**
   - congestion window, rate, grant tokens
2. **Two main parameter adjustment signals**
   - external signals, e.g., acks and CNPs
   - periodic internal signals, .e.g., counters

Credit Management Patterns

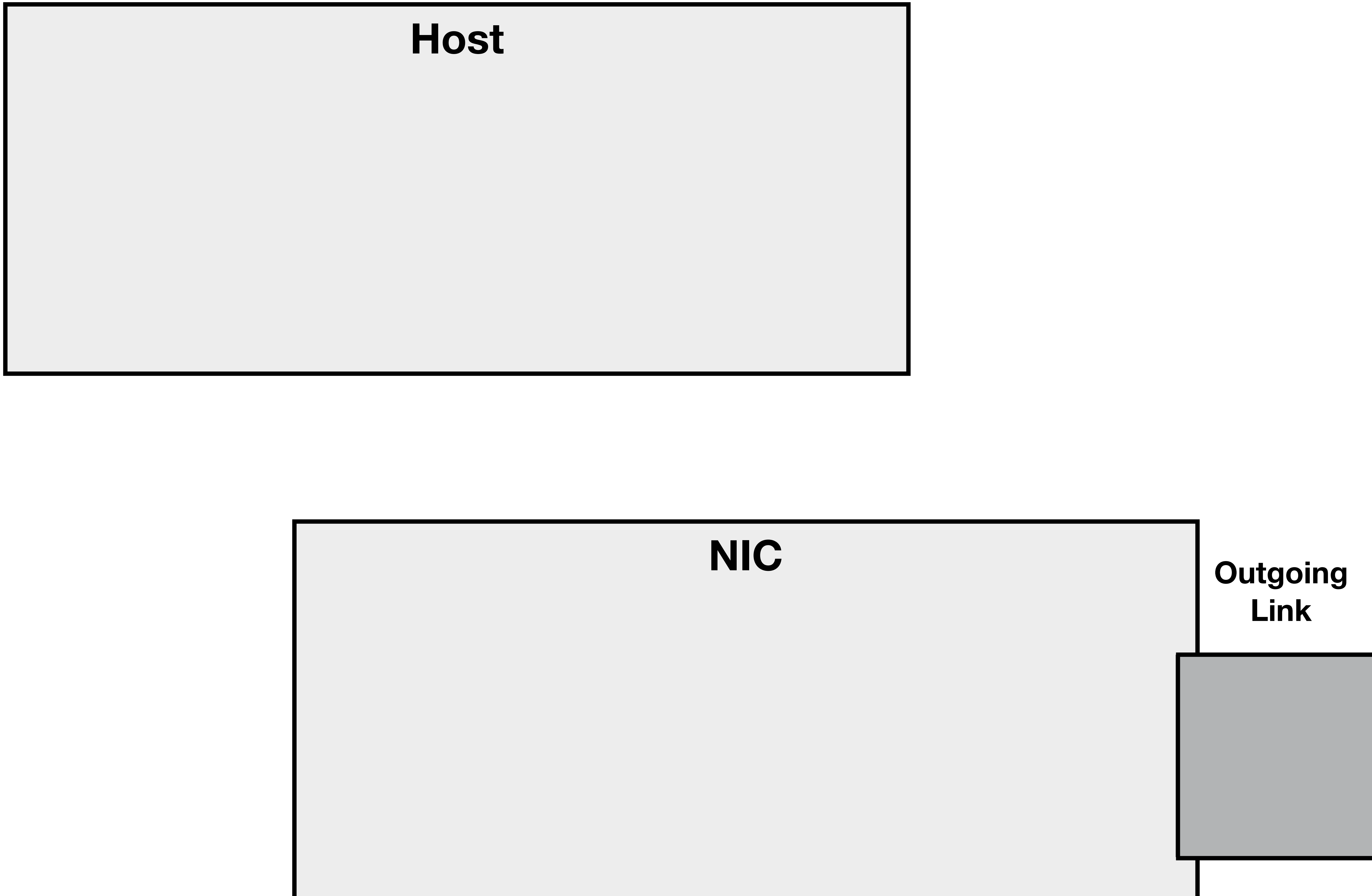


1. **Three common credit calculation schemes**
    - congestion window, rate, grant tokens
2. **Two main parameter adjustment signals**
    - external signals, e.g., acks and CNPs
    - periodic internal signals, .e.g., counters
    - aligns with existing programmable modules for segment selection

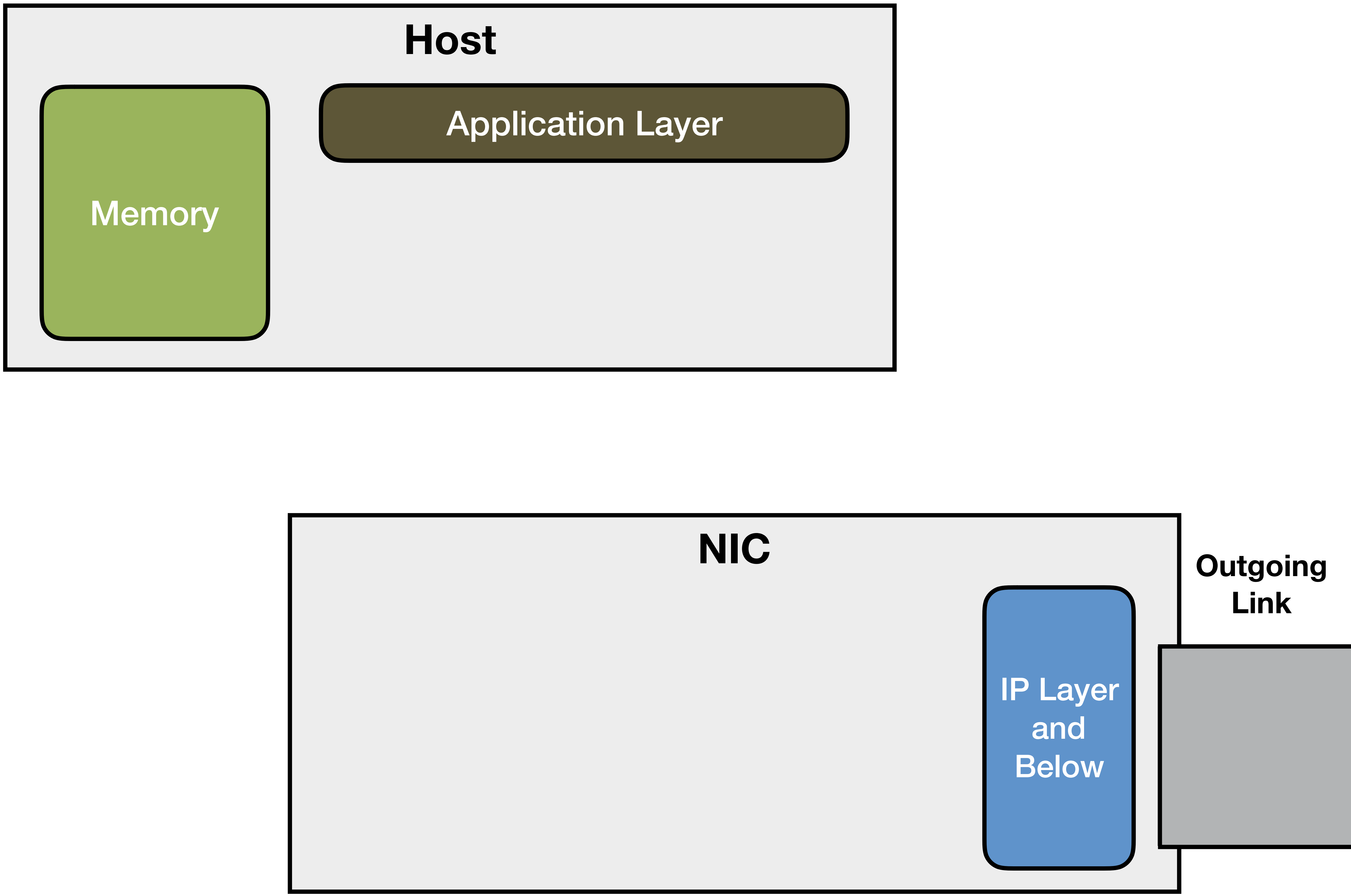
The Two Engines



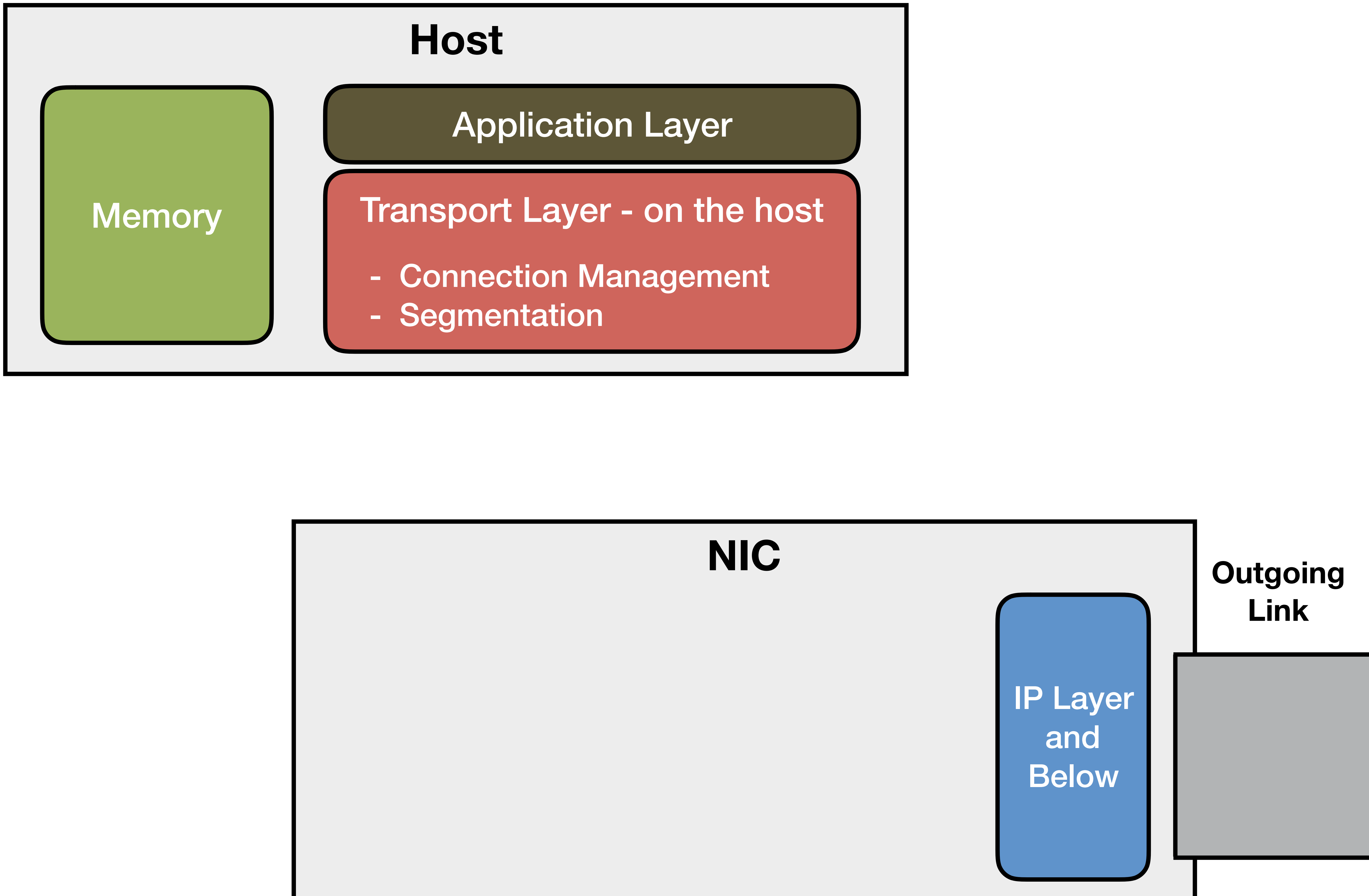
Integration into the Network Stack



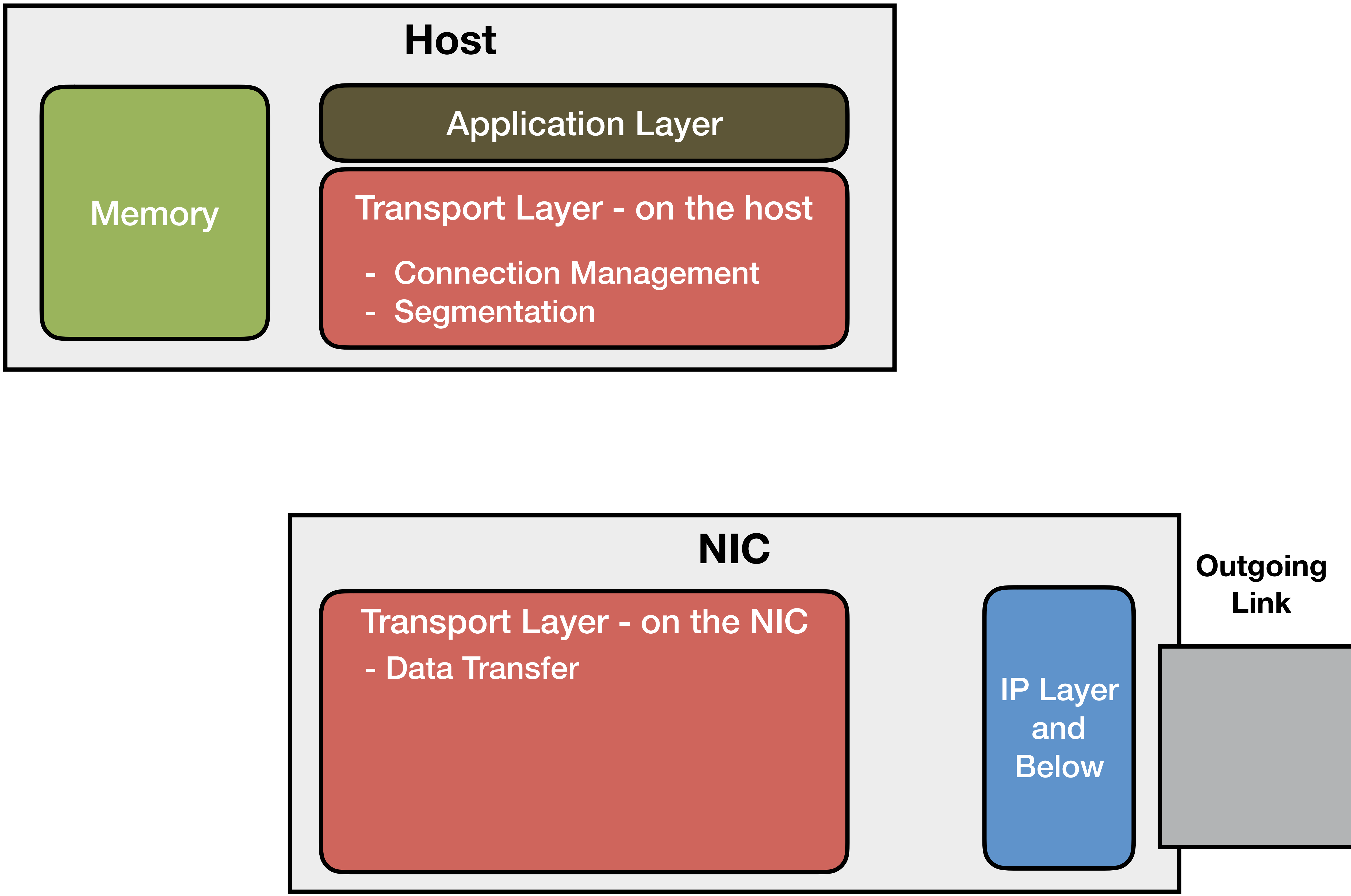
Integration into the Network Stack



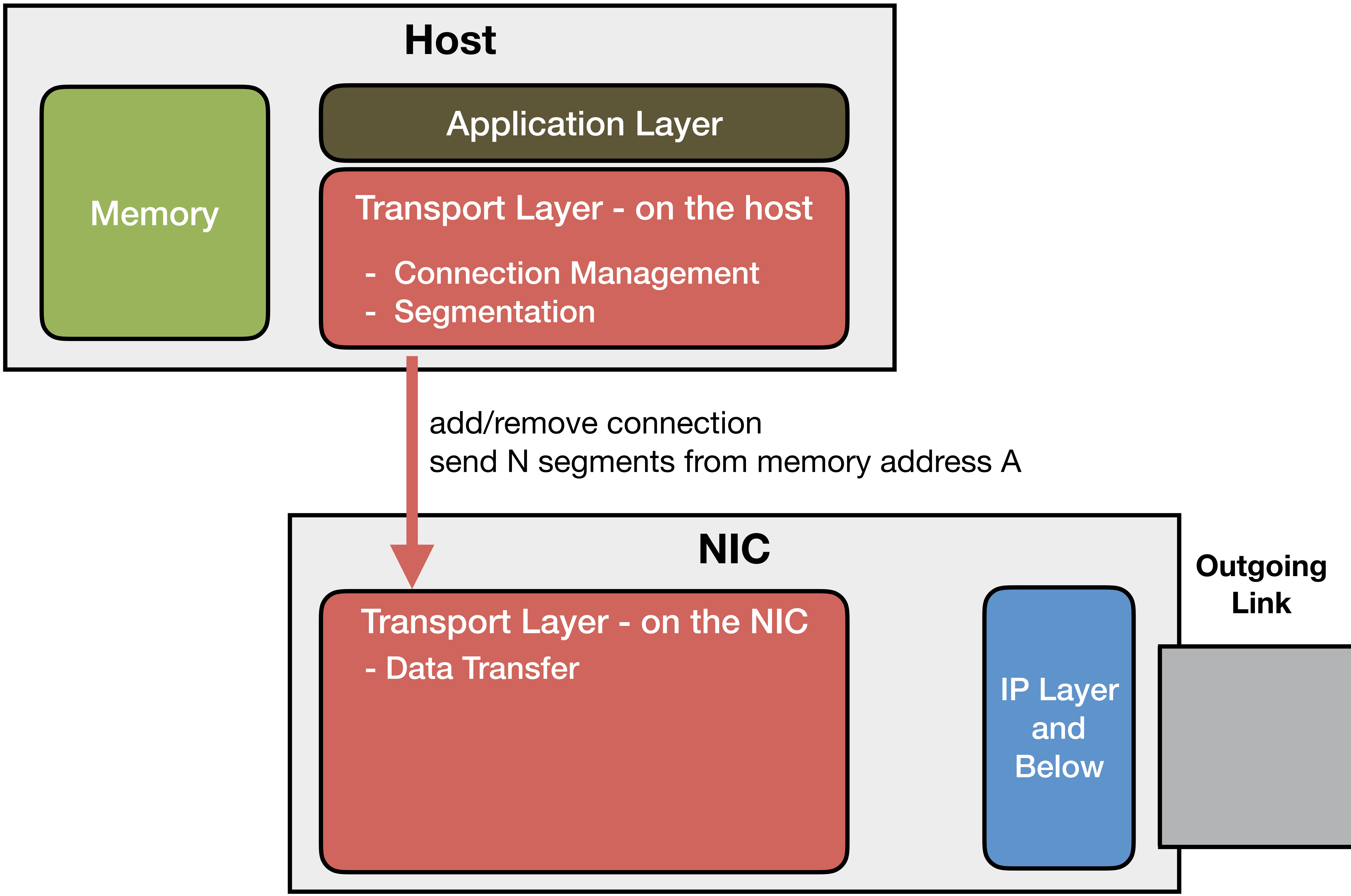
Integration into the Network Stack



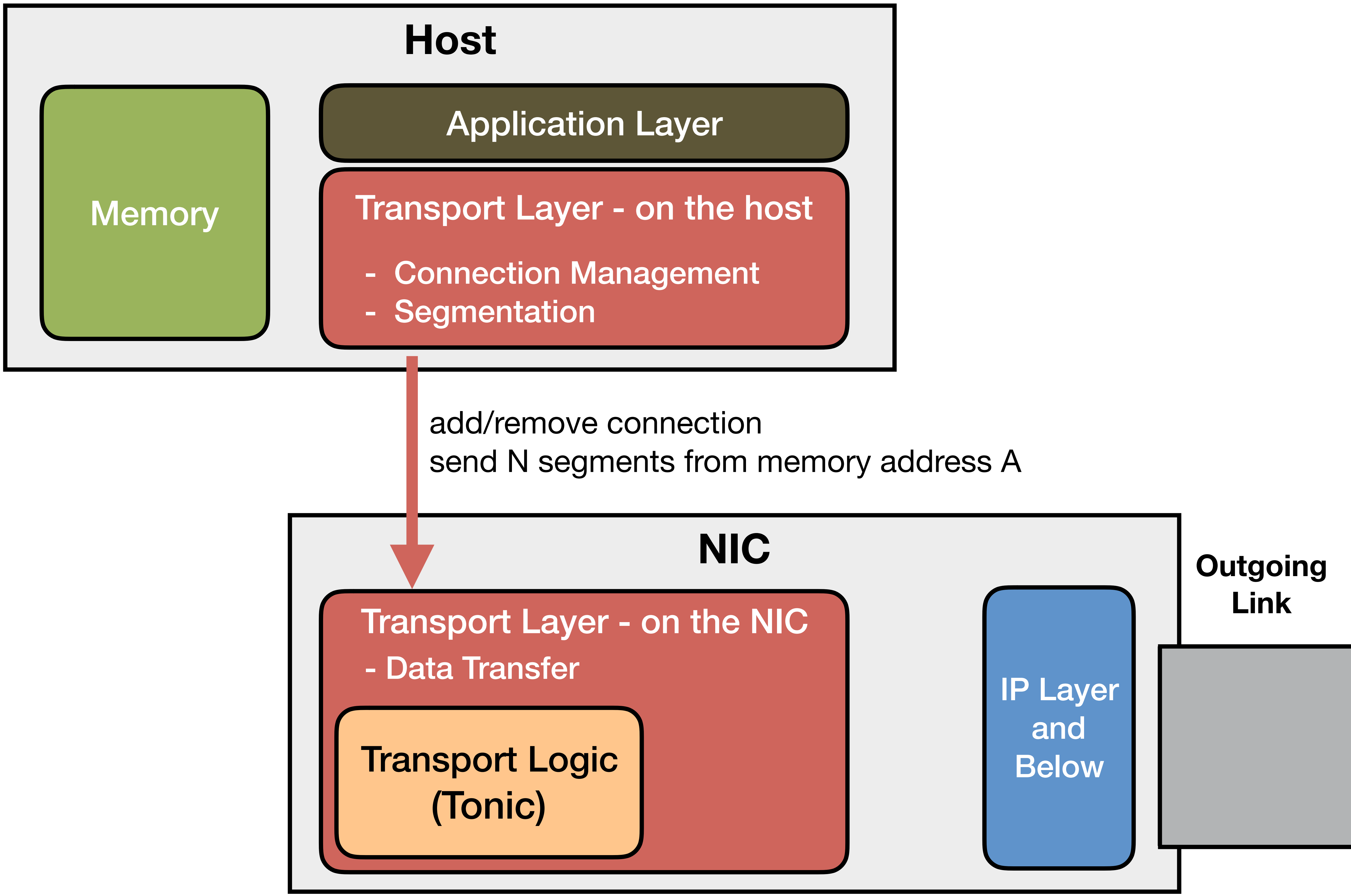
Integration into the Network Stack



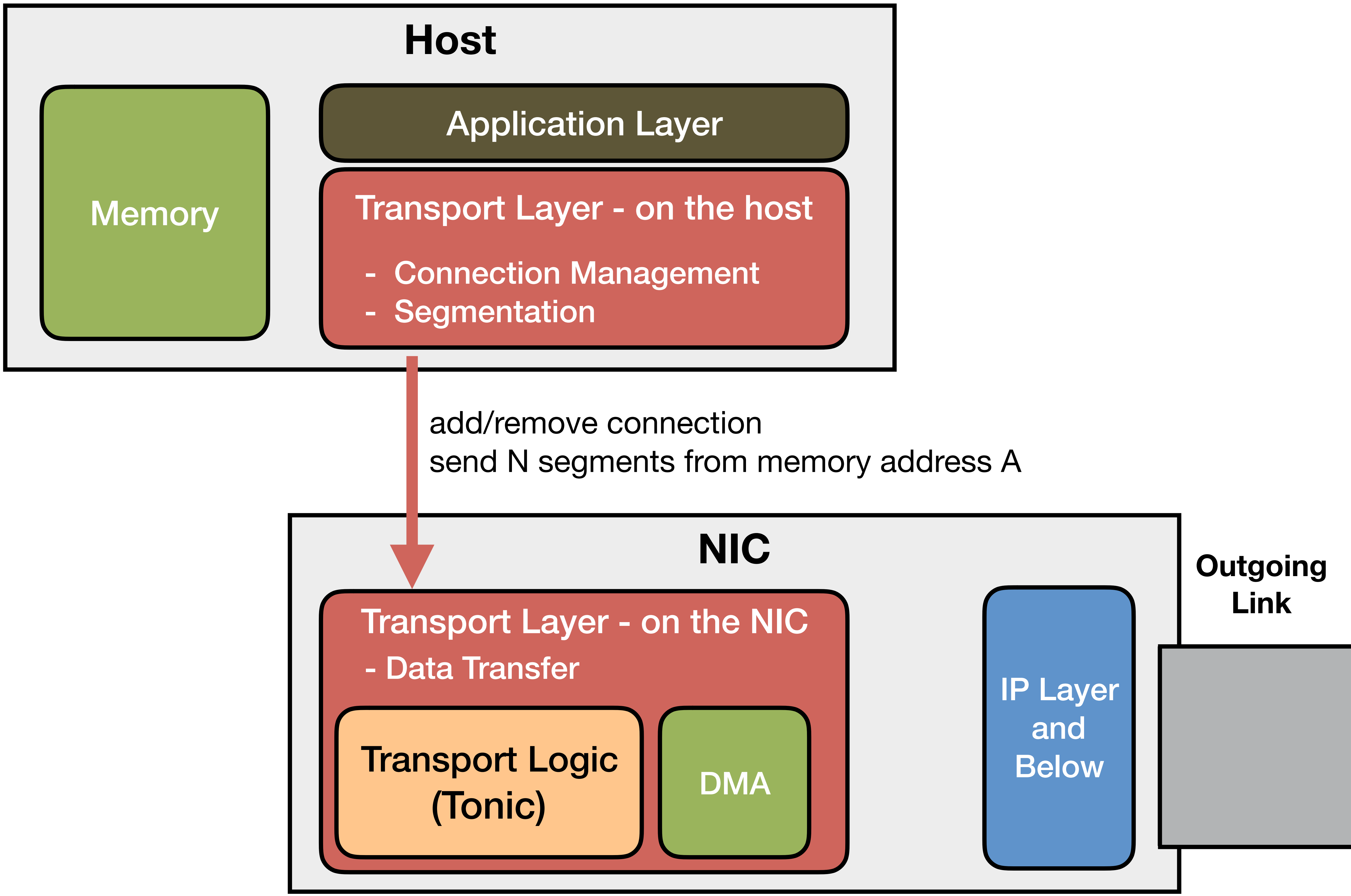
Integration into the Network Stack



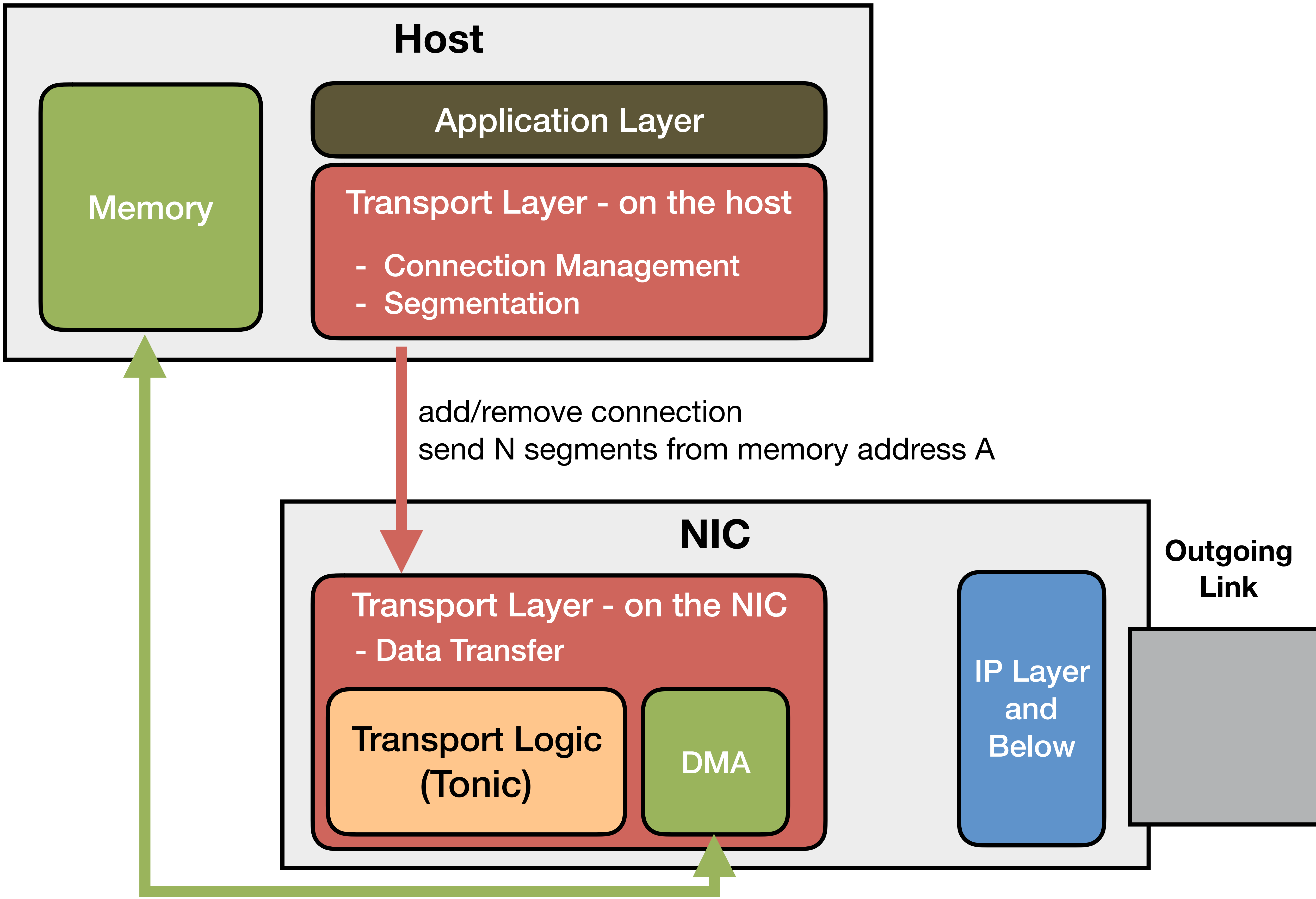
Integration into the Network Stack



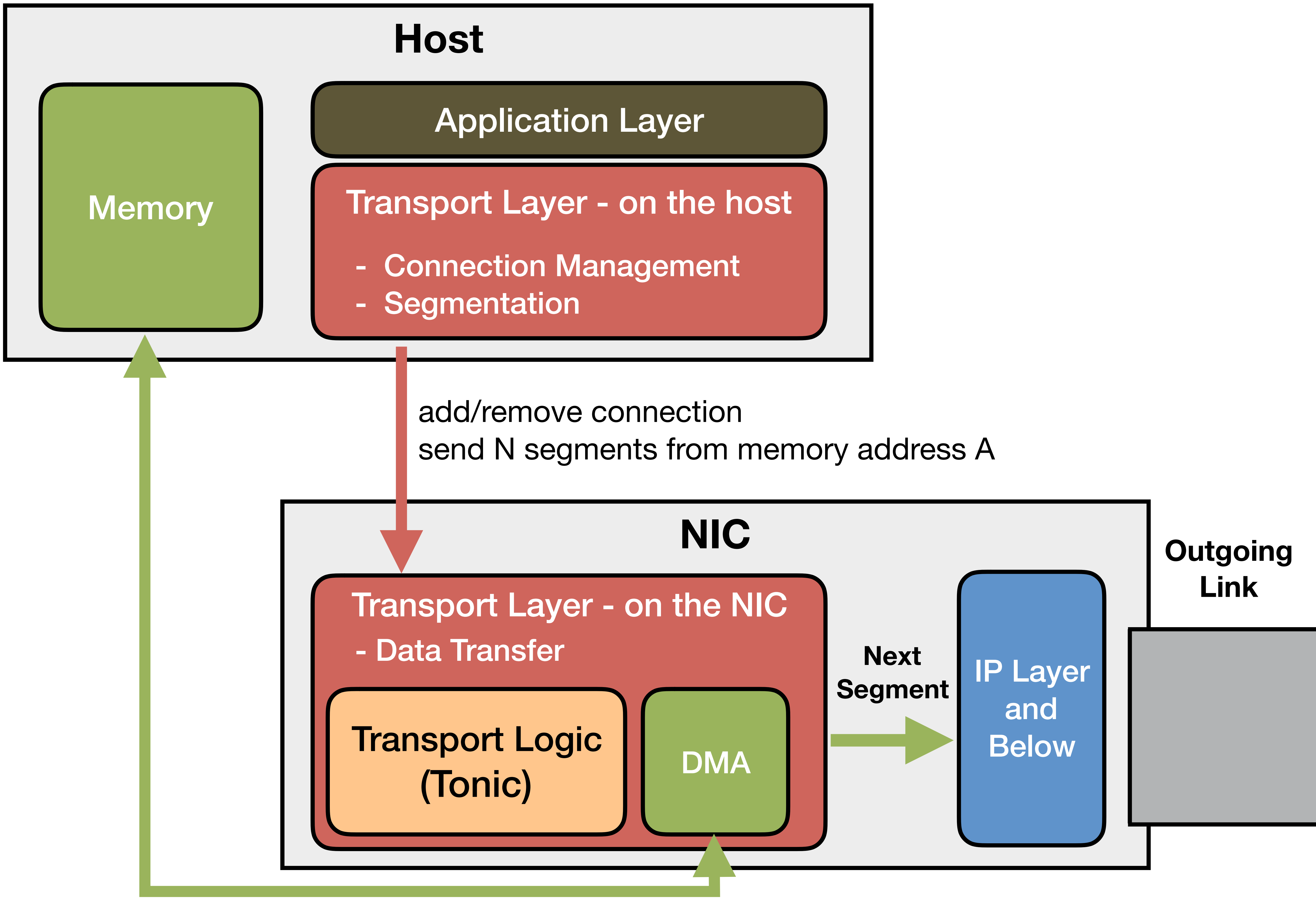
Integration into the Network Stack



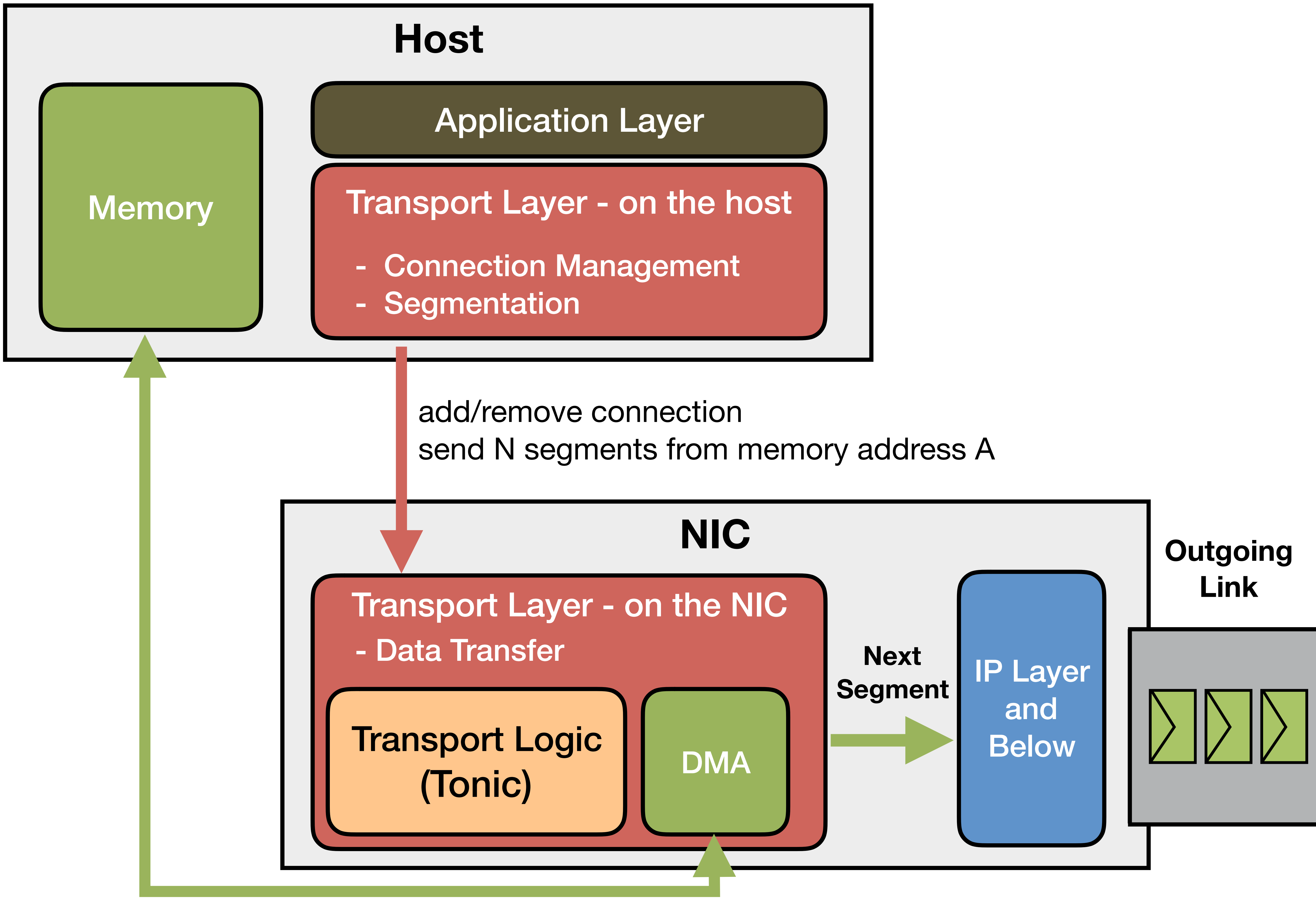
Integration into the Network Stack



Integration into the Network Stack



Integration into the Network Stack



Evaluation - Programmability

Evaluation - Programmability

- Implemented six representative protocols

Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno

Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)

Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)

Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)
  - DCQCN, IRN (Improved RoCE NIC)

Evaluation - Programmability

- Implemented six representative protocols
    - Reno, New Reno
    - SACK (Selective ACK)
    - NDP (Receiver-driven data-center transport)
    - DCQCN, IRN (Improved RoCE NIC)
- All meet timing for 100 Gpbs (10-ns clock)

Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)
  - DCQCN, IRN (Improved RoCE NIC)
- All meet timing for 100 Gpbs (10-ns clock)
- Implemented within 200 lines of *Verilog* code

Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)
  - DCQCN, IRN (Improved RoCE NIC)
- All meet timing for 100 Gpbs (10-ns clock)
- Implemented within 200 lines of *Verilog* code
  - uses 0.5% of total logic resources

Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)
  - DCQCN, IRN (Improved RoCE NIC)
- All meet timing for 100 Gpbs (10-ns clock)
- Implemented within 200 lines of *Verilog* code
  - uses 0.5% of total logic resources
- Re-usable modules are 8K lines of *Verilog* code

Evaluation - Programmability

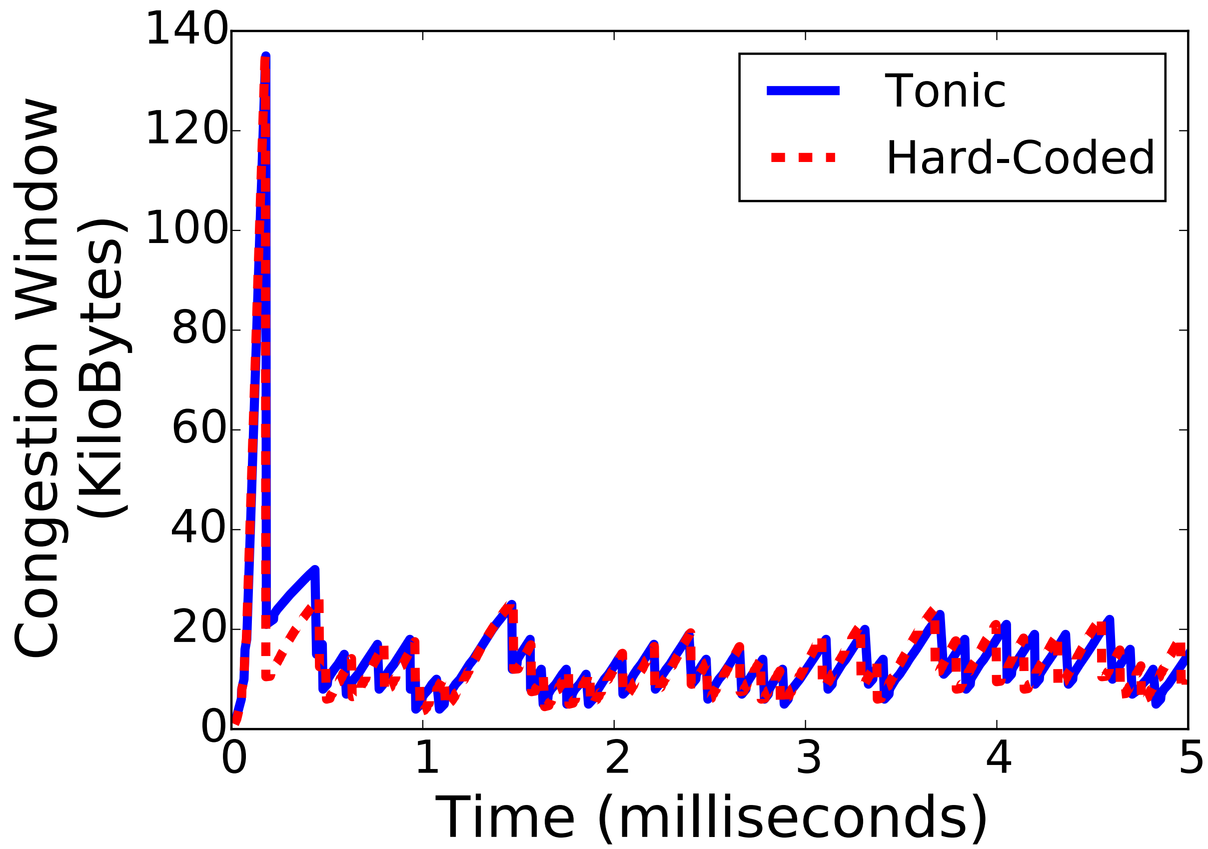
- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)
  - DCQCN, IRN (Improved RoCE NIC)
- All meet timing for 100 Gpbs (10-ns clock)
- Implemented within 200 lines of *Verilog* code
  - uses 0.5% of total logic resources
- Re-usable modules are 8K lines of *Verilog* code
  - uses 35% of total logic resources

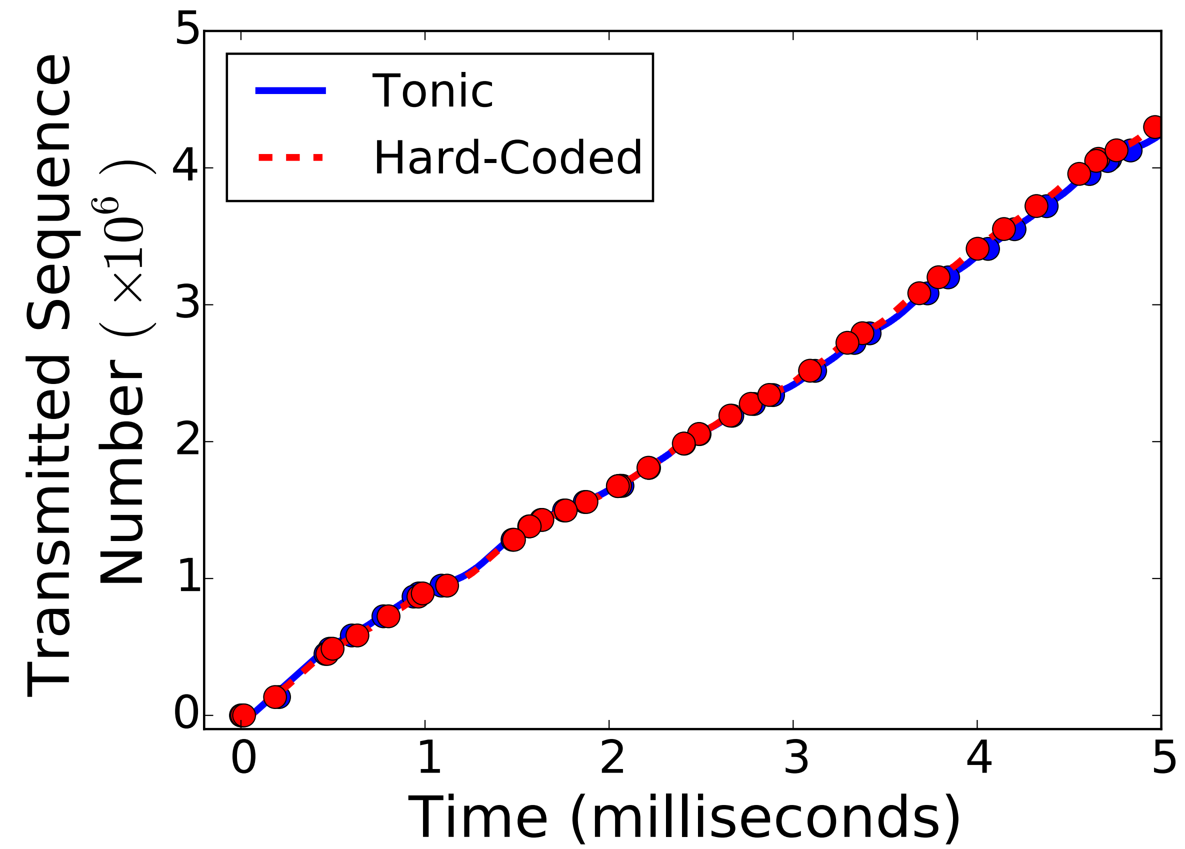
Evaluation - Scalability

	Metric	**Results**	
Complexity of User-Defined Logic	logic levels	$(0, 31]$	meets timing
		$(31, 42]$	depends on operations
		$(42, 65]$	violates timing
User-Defined State	bytes	256	grant token
		340	rate
		448	congestion window
Window Size	segments	256	
Concurrent Flows	count	2048	

Evaluation - End-to-End Simulations

- Cycle-accurate hardware simulator for Tonic within NS3
- Compared existing protocols with Tonic implementations
  - TCP New Reno (plots shown below) and DCQCN





Summary

Summary

- Tonic is a **programmable hardware** architecture

Summary

- Tonic is a **programmable hardware** architecture
- Enables implementing transport protocols at high-speed
  - with **modest development effort**

Summary

- Tonic is a **programmable hardware** architecture
- Enables implementing transport protocols at high-speed
  - with **modest development effort**
- Exploits domain-specific optimizations
  - Implementing common transport patterns as re-usable modules

SNAP: Stateful Network-Wide Abstractions for Packet Processing

Mina Tahmasbi Arashloo[1], Yaron Koral[1], Michael Greenberg[2], Jennifer Rexford[1], and David Walker[1]

[1] Princeton University, [2] Pomona College

In-Network Stateful Applications

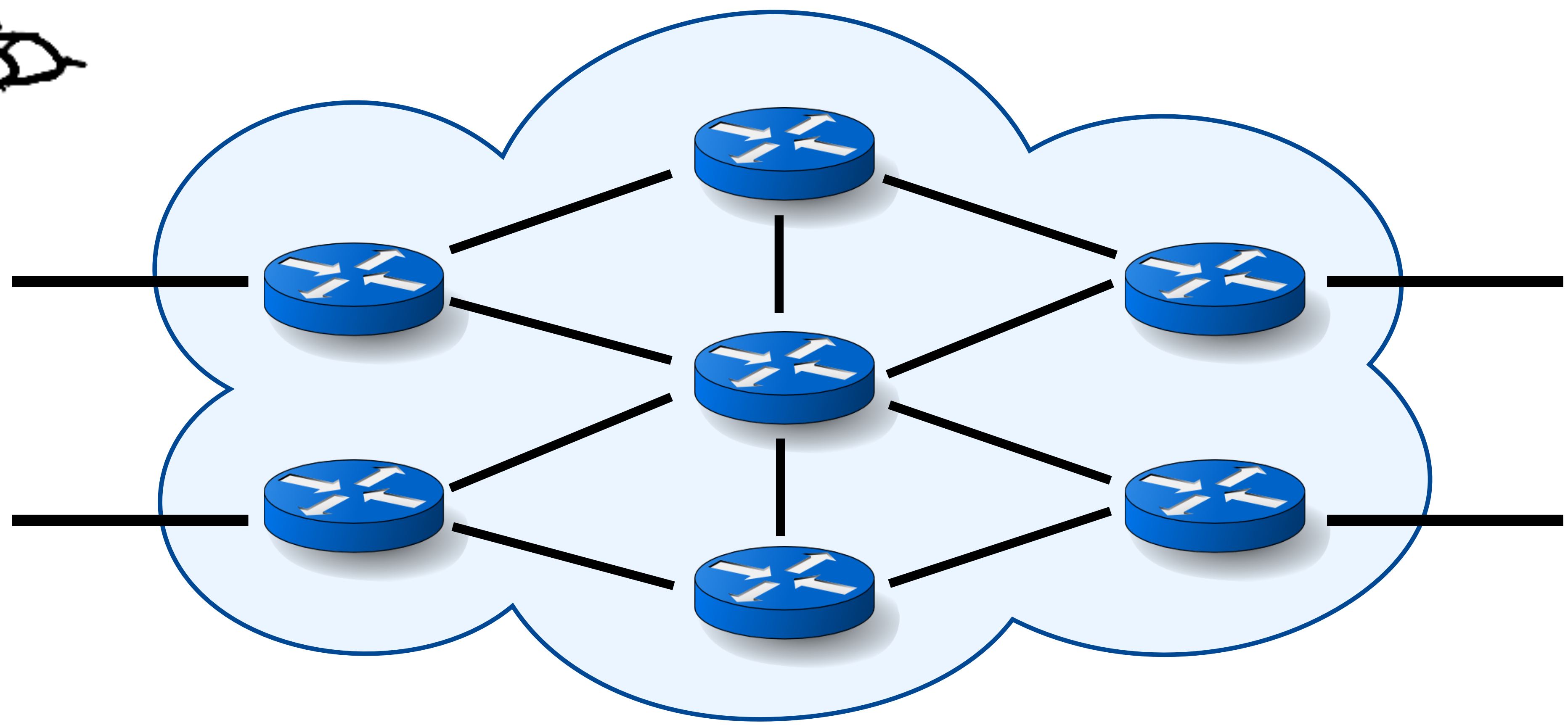
Source	Application
Chimera (USENIX Security'12)	Number of domains sharing the same IP address
	Number of distinct IP addresses under the same domain
	DNS TTL change tracking
	DNS tunnel detection
	Sidejack detection
	Phishing/spam detection
FAST (HotSDN'14)	Stateful firewall
	FTP monitoring
	Heavy-hitter detection
	Super-spreader detection
	Sampling based on flow size
	Selective packet dropping (MPEG frames)
	Connection affinity
Bohatei (USENIX Security'15)	SYN flood detection
	DNS reflection (and amplification) detection
	UDP flood mitigation
	Elephant flows detection
Others	Bump-on-the-wire TCP state machine
	Snort flowbits

In-Network Stateful Applications

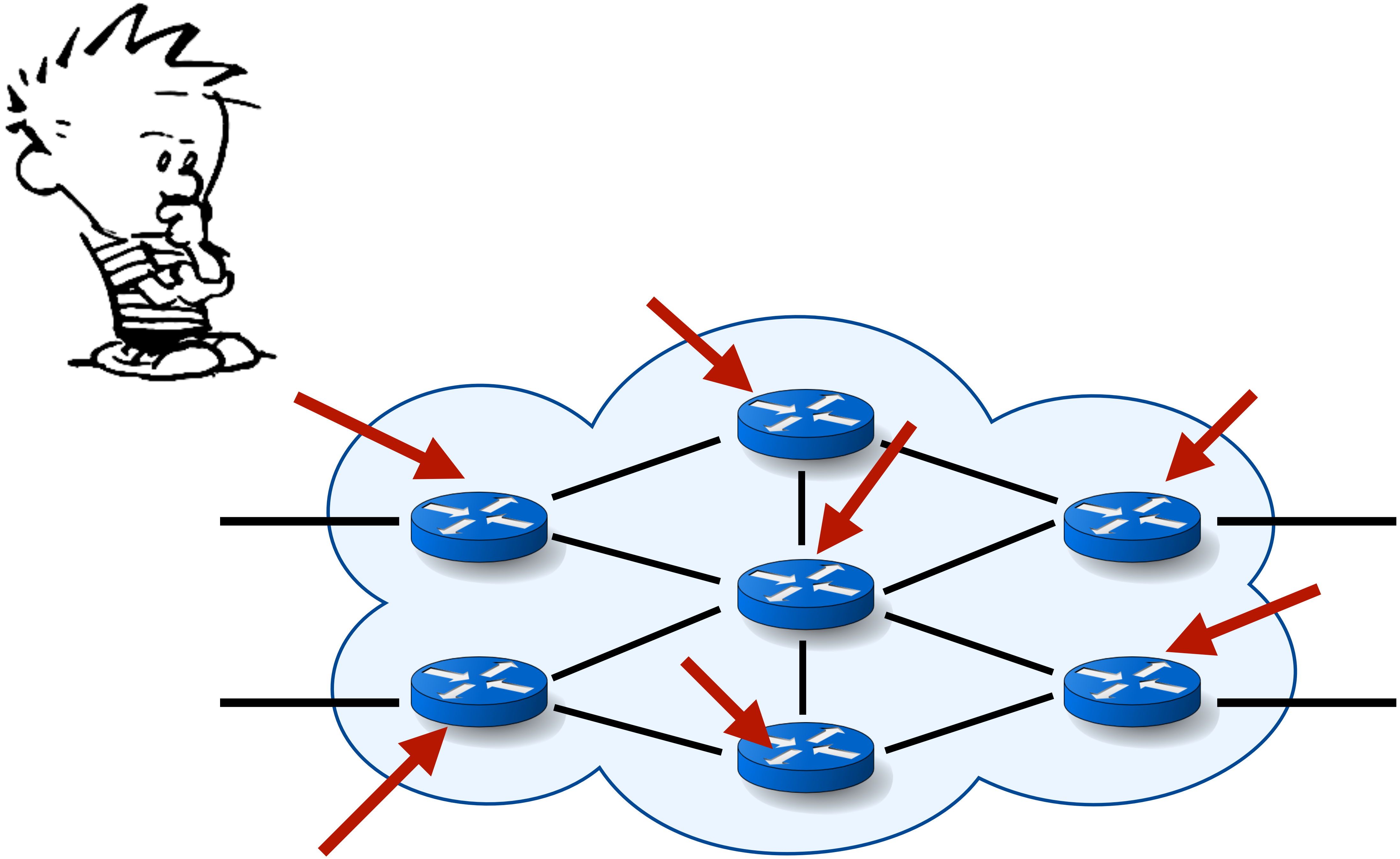
Source	Application
Chimera (USENIX Security'12)	Number of domains sharing the same IP address
	Number of distinct IP addresses under the same domain
	DNS TTL change tracking
	DNS tunnel detection
(H	
Bohatei (USENIX Security'15)	SYN flood detection
	DNS reflection (and amplification) detection
	UDP flood mitigation
	Elephant flows detection
Others	Bump-on-the-wire TCP state machine
	Snort flowbits

Programmable switches expose **data plane state** through a programming interface!

Distributed Stateful Programming is Challenging

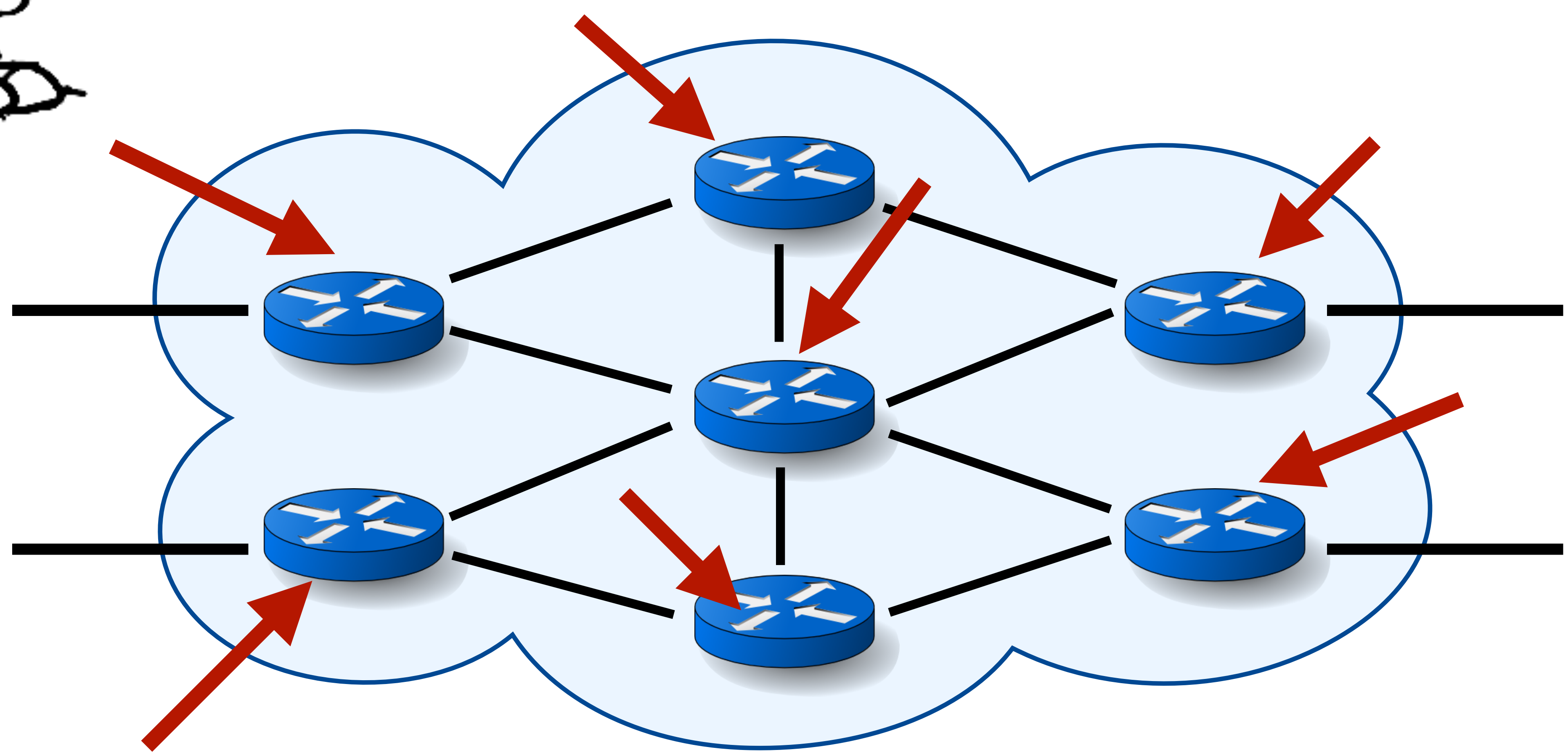


Distributed Stateful Programming is Challenging



Distributed Stateful Programming is Challenging

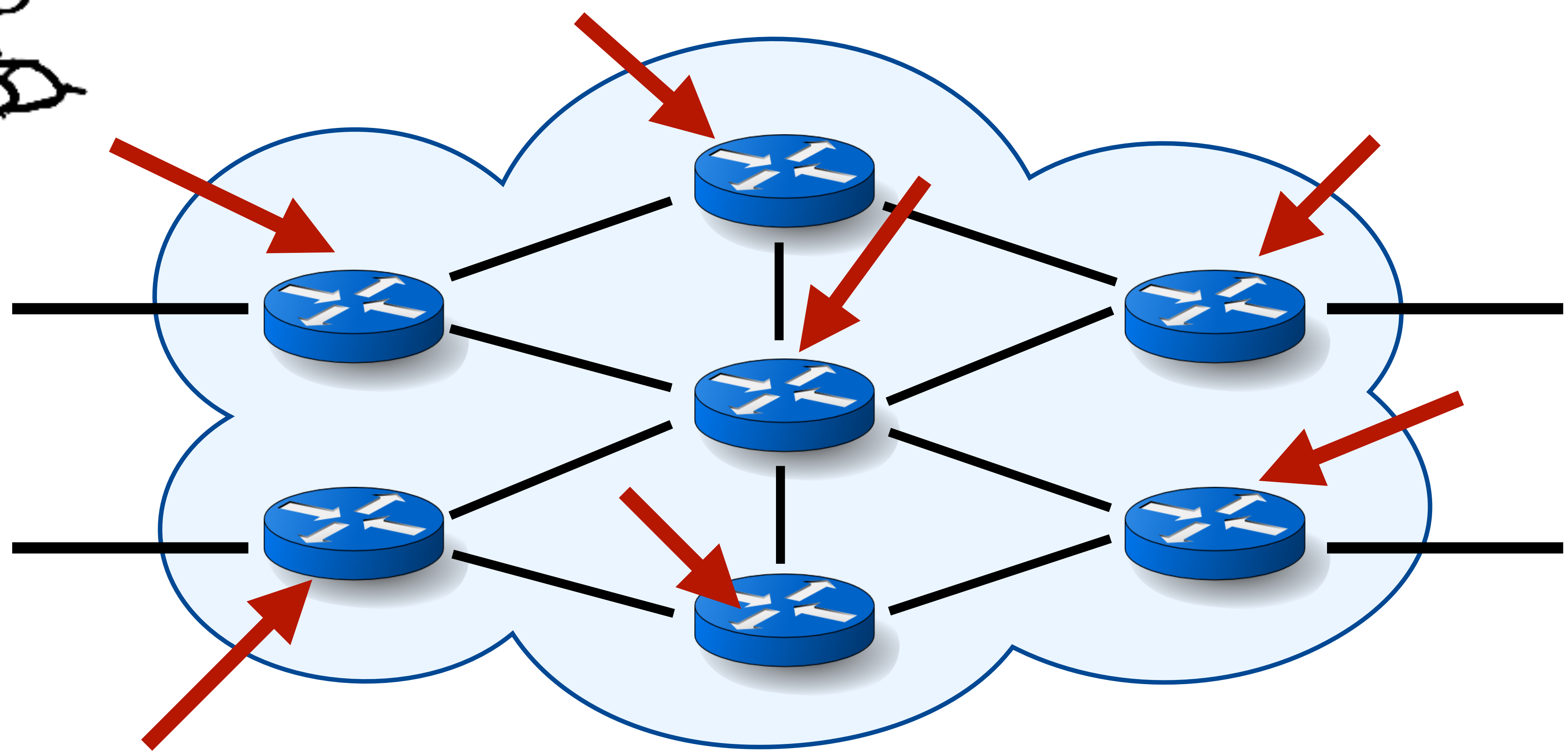
Use one switch?



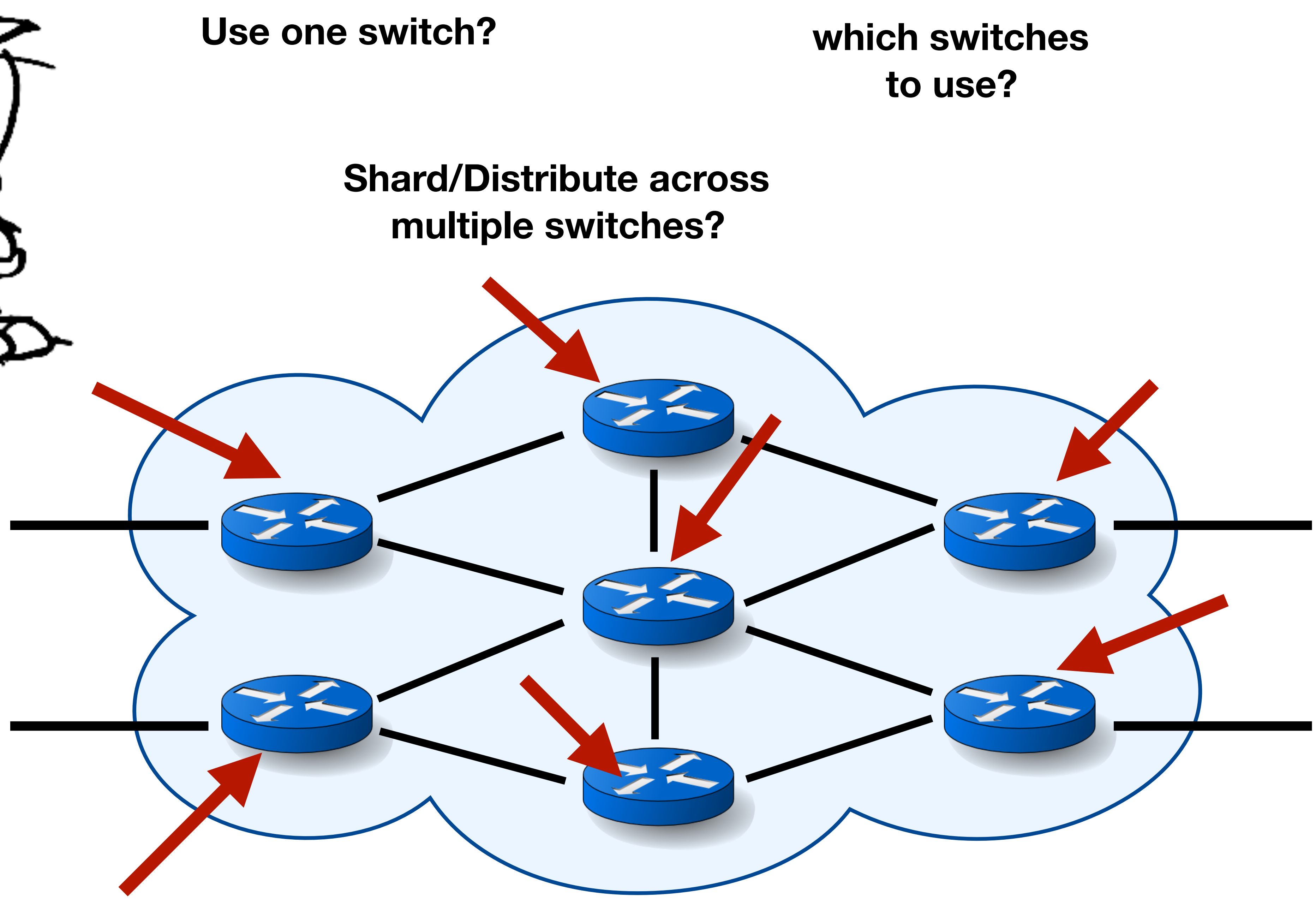
Distributed Stateful Programming is Challenging

Use one switch?

Shard/Distribute across
multiple switches?



Distributed Stateful Programming is Challenging



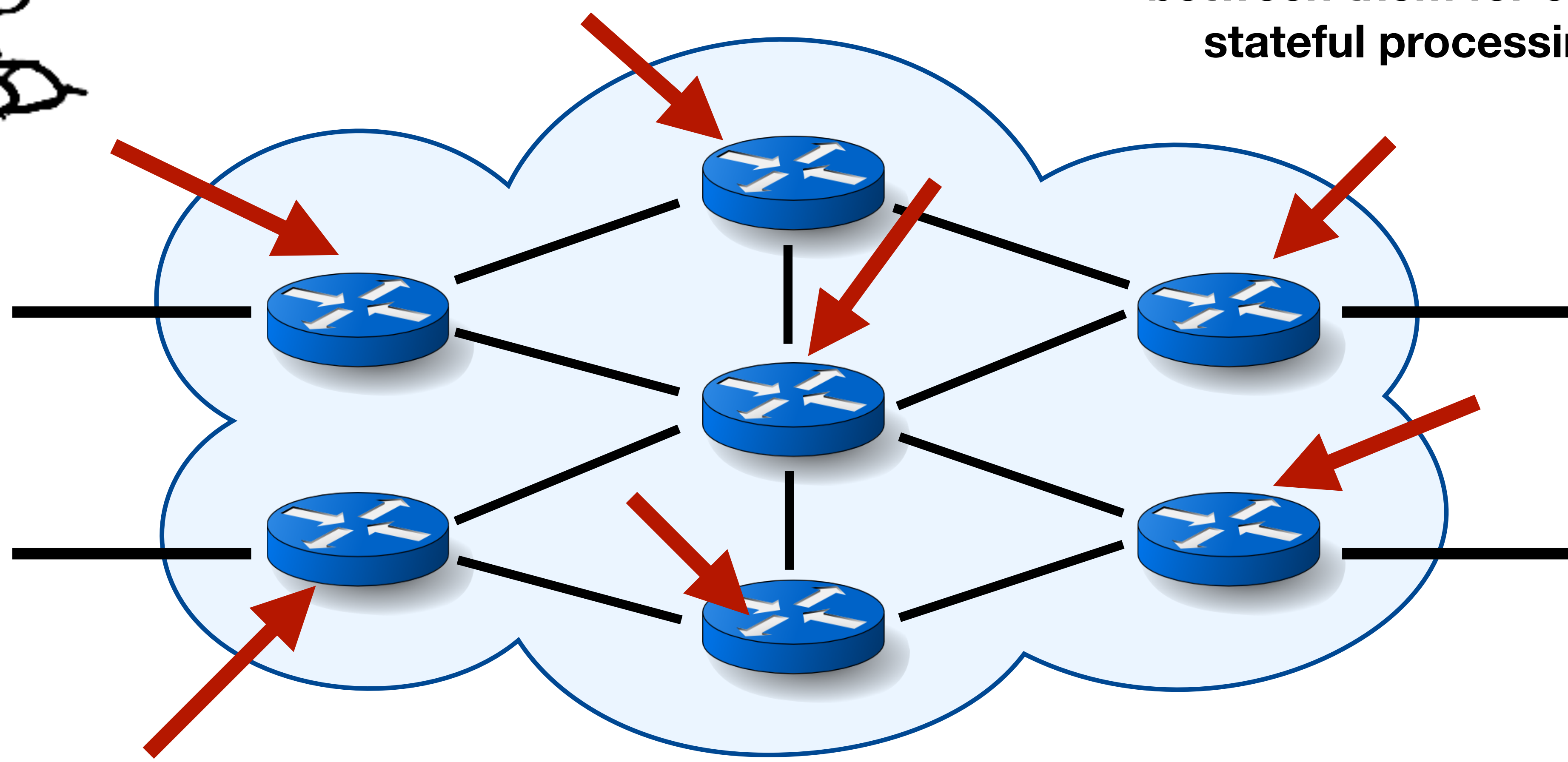
Distributed Stateful Programming is Challenging

Use one switch?

which switches
to use?

Shard/Distribute across
multiple switches?

How to coordinate
between them for correct
stateful processing?



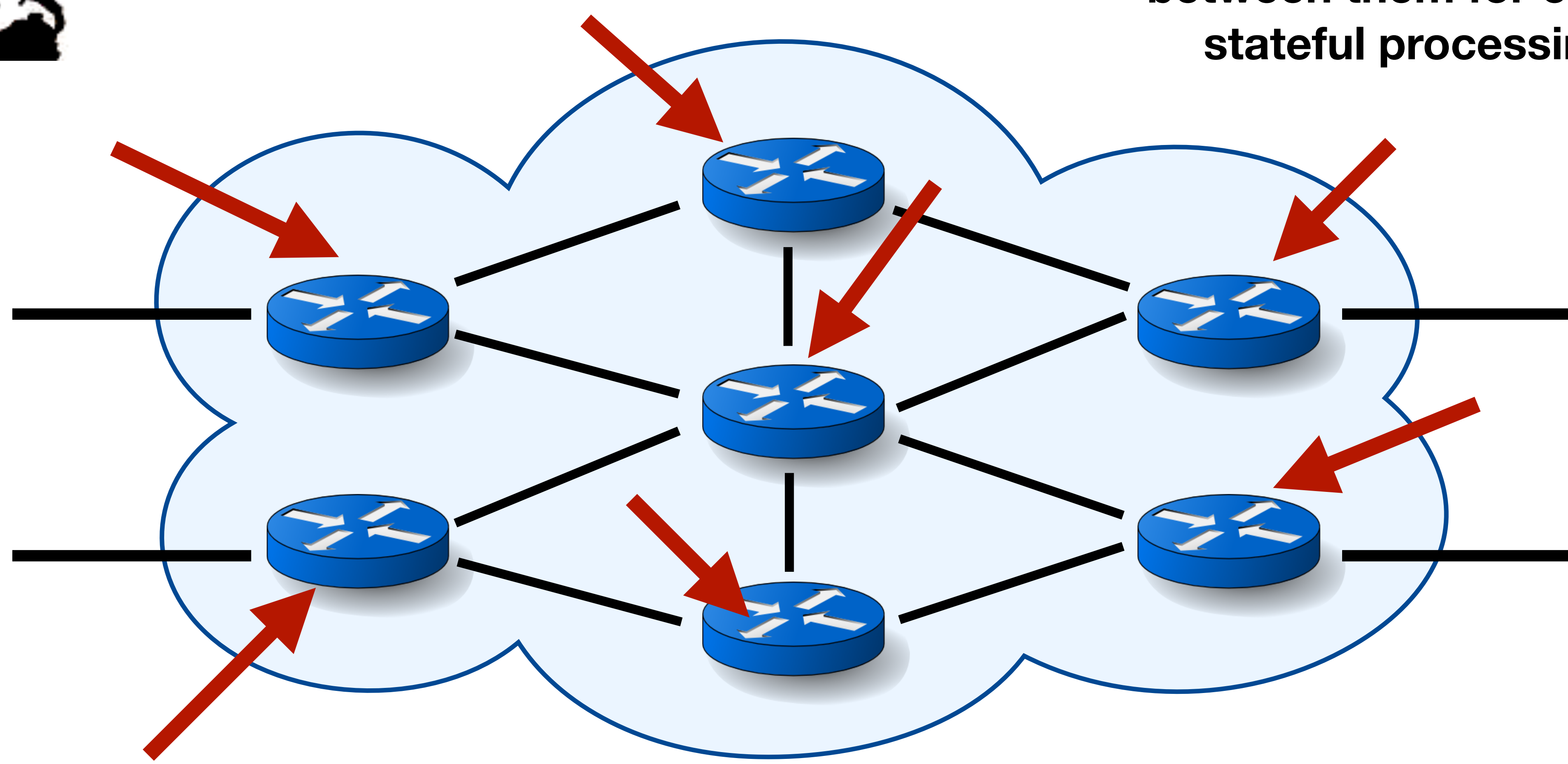
Distributed Stateful Programming is Challenging

Use one switch?

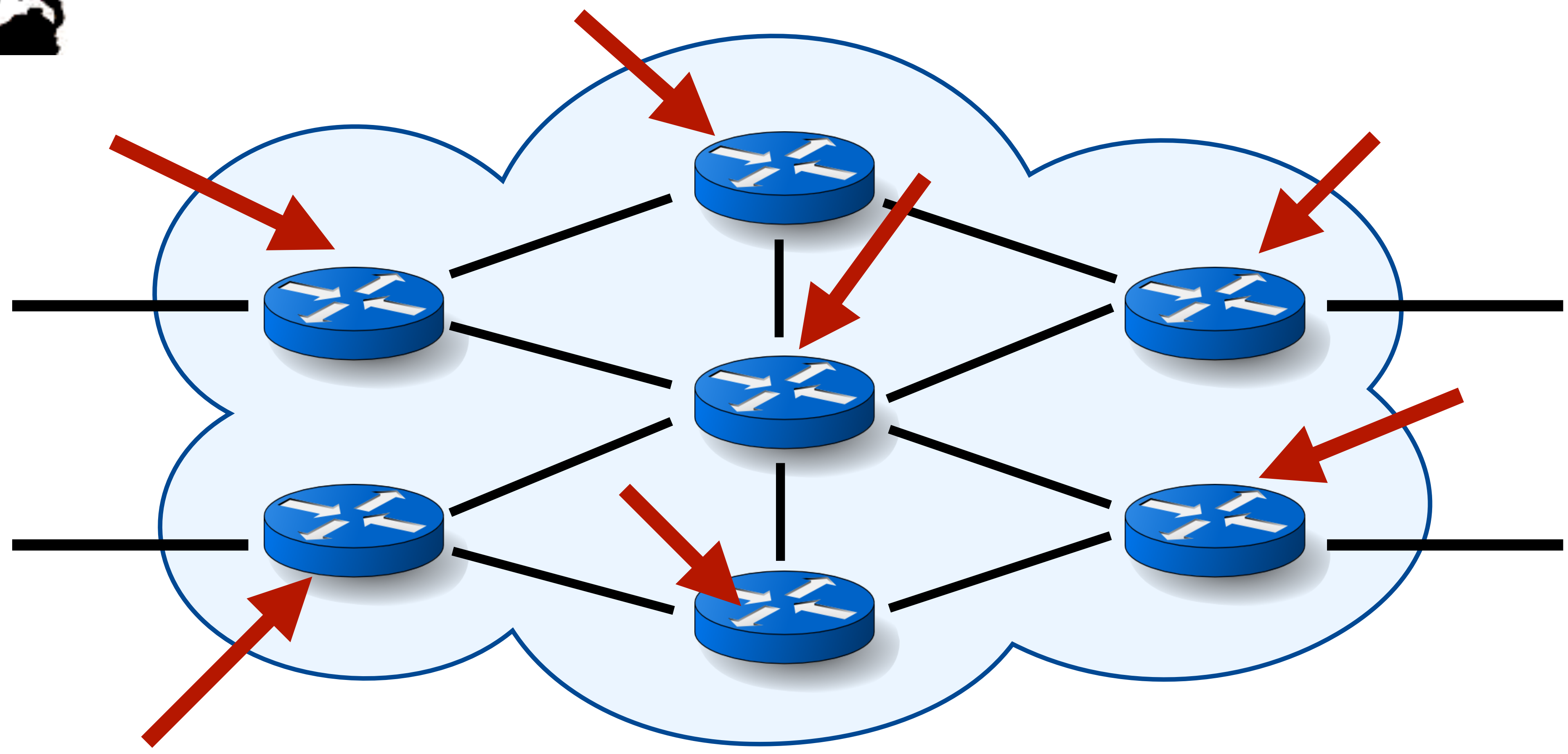
which switches to use?

Shard/Distribute across multiple switches?

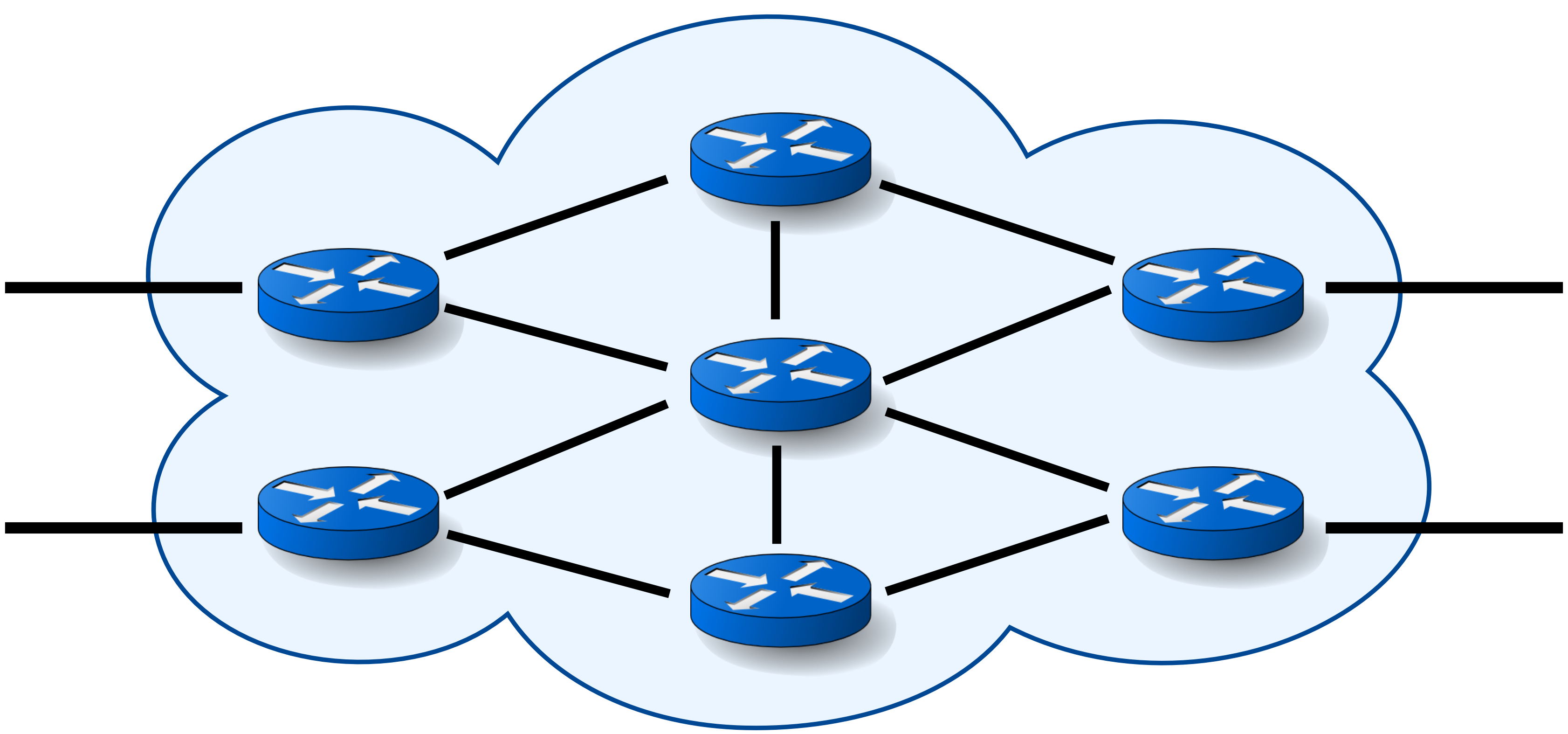
How to coordinate between them for correct stateful processing?



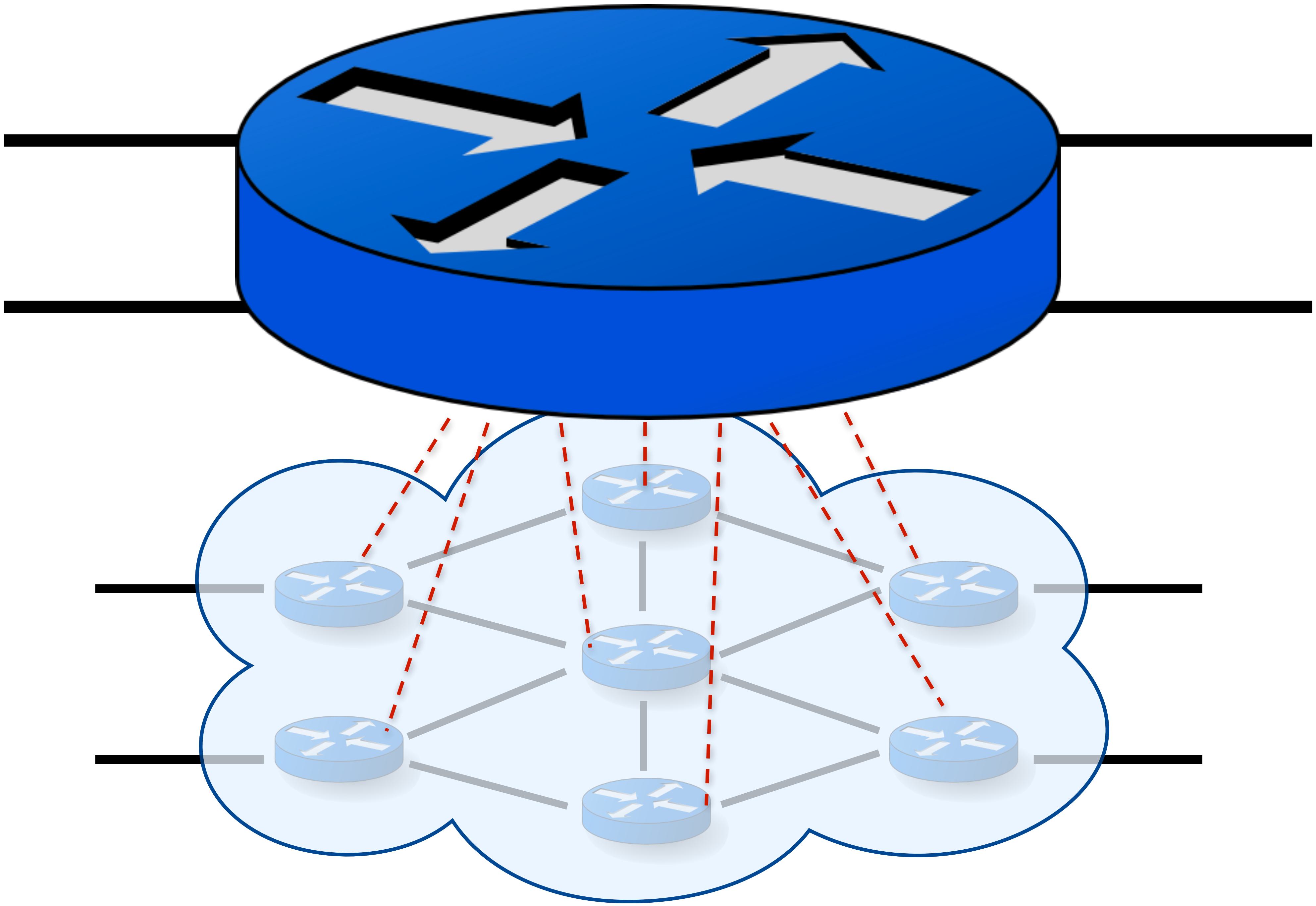
SNAP: Stateful Network-Wide Abstractions for Packet Processing



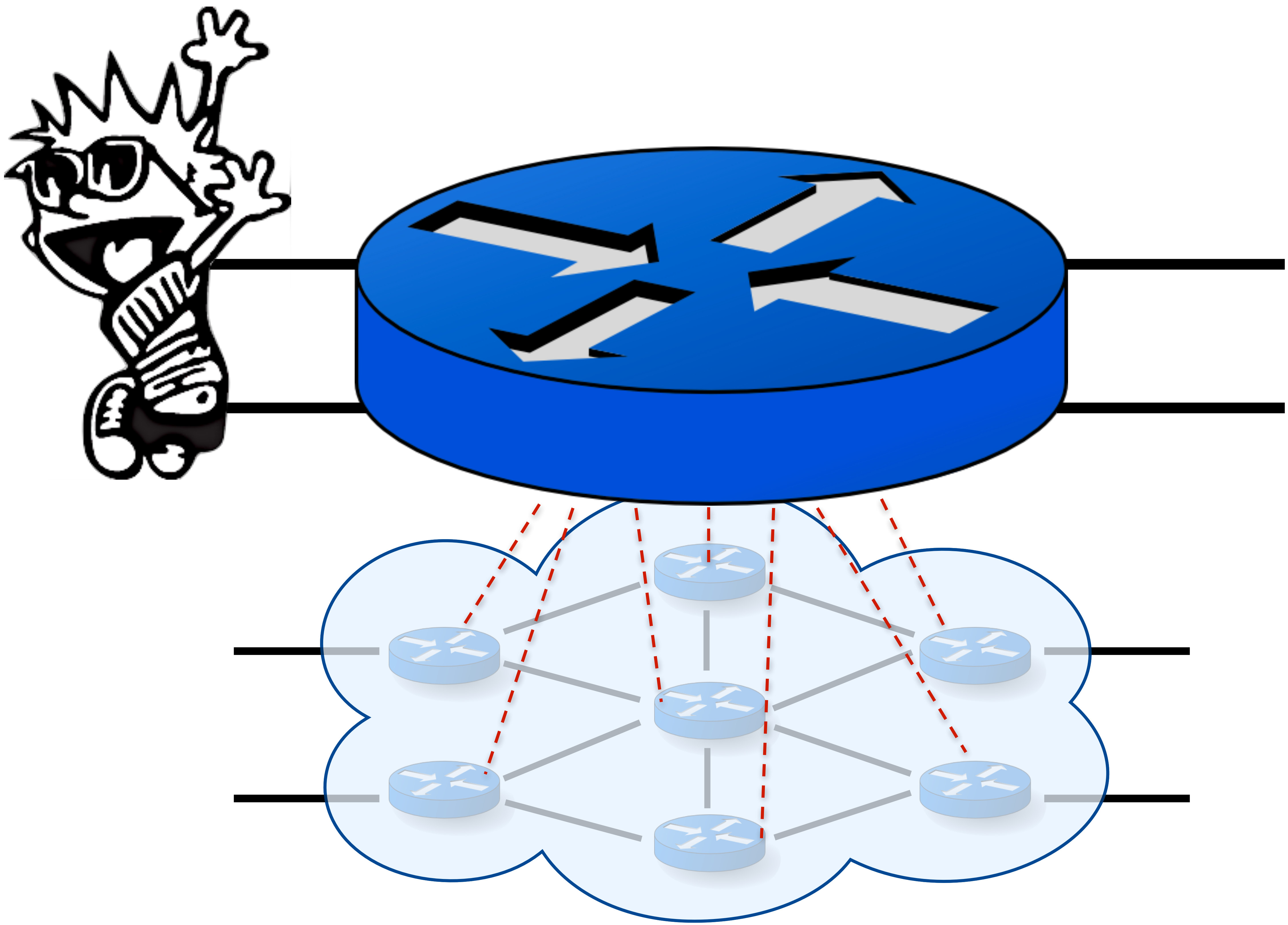
SNAP: Stateful Network-Wide Abstractions for Packet Processing



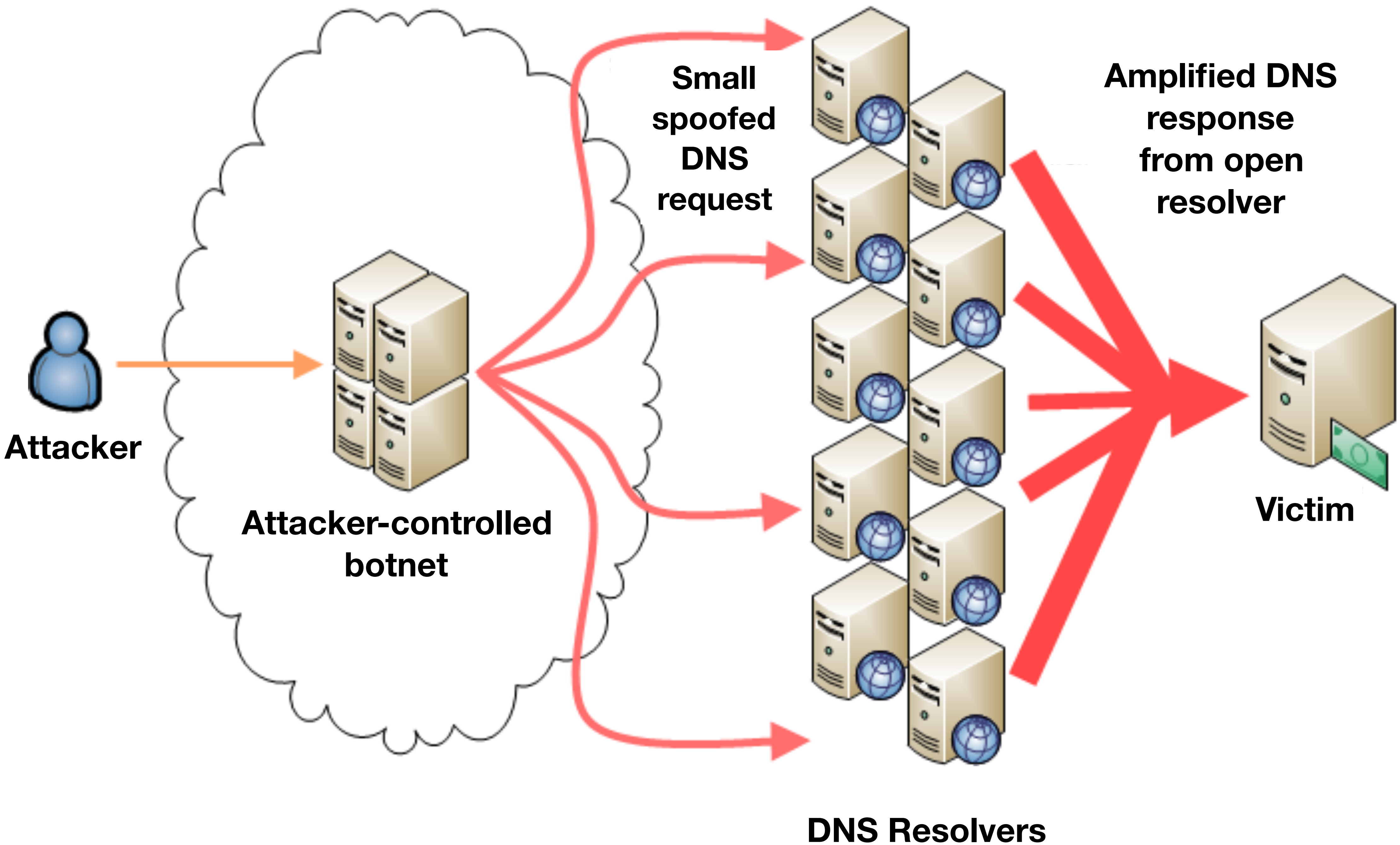
SNAP: Stateful Network-Wide Abstractions for Packet Processing



SNAP: Stateful Network-Wide Abstractions for Packet Processing

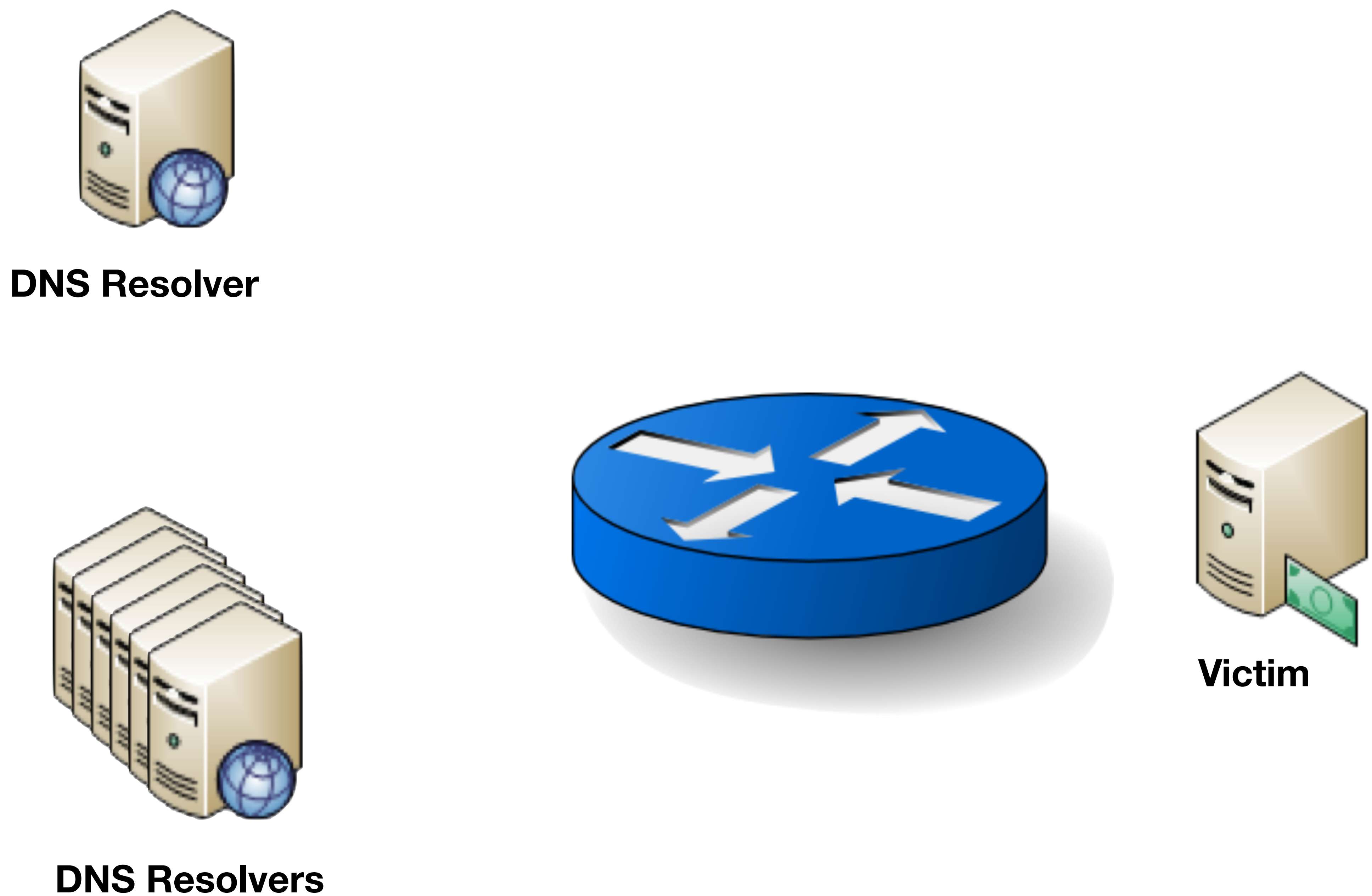


Example - Detecting DNS Reflection Attacks



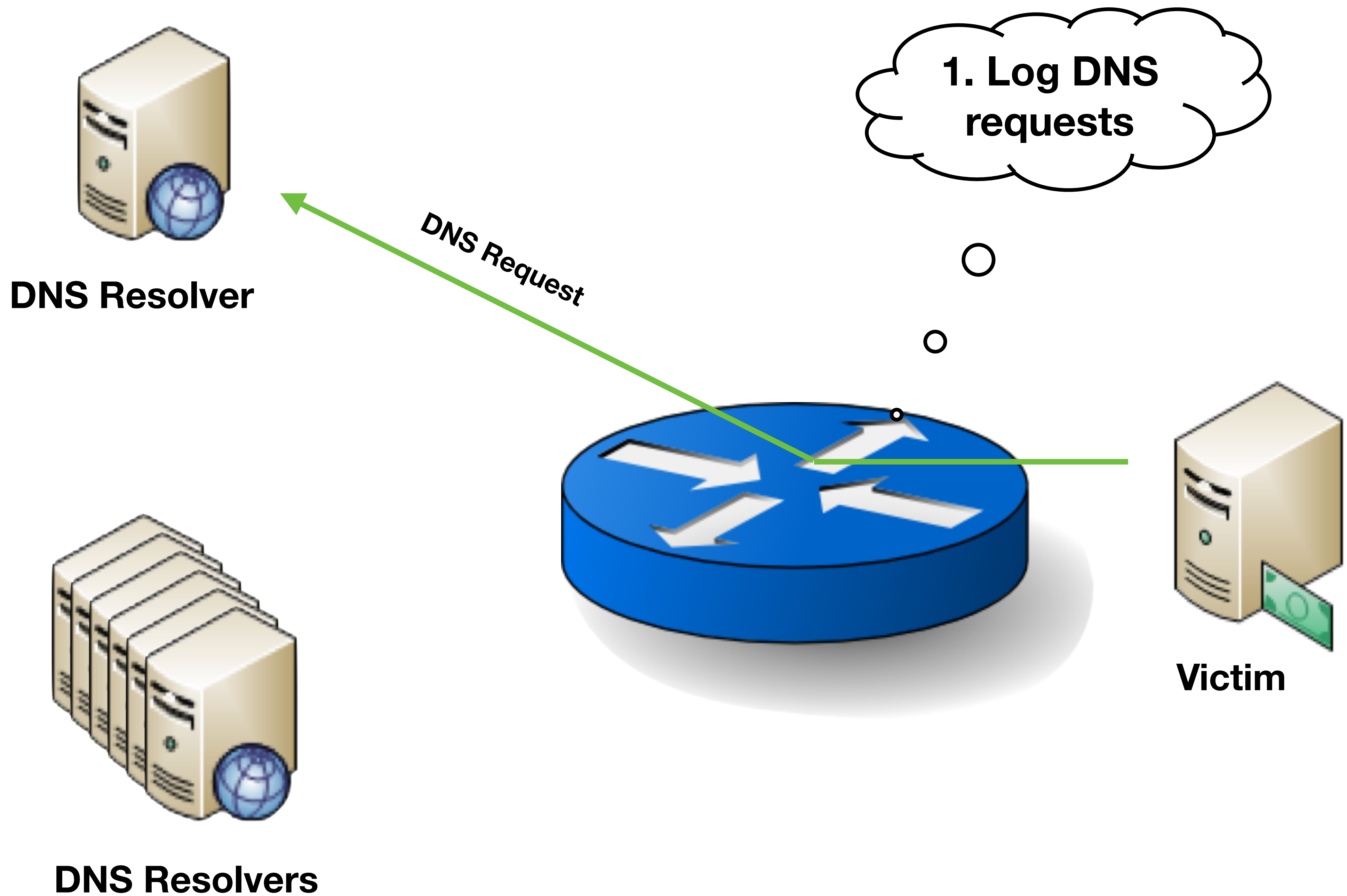
http://ddosandbotnets.blogspot.com

Example - Detecting DNS Reflection Attacks



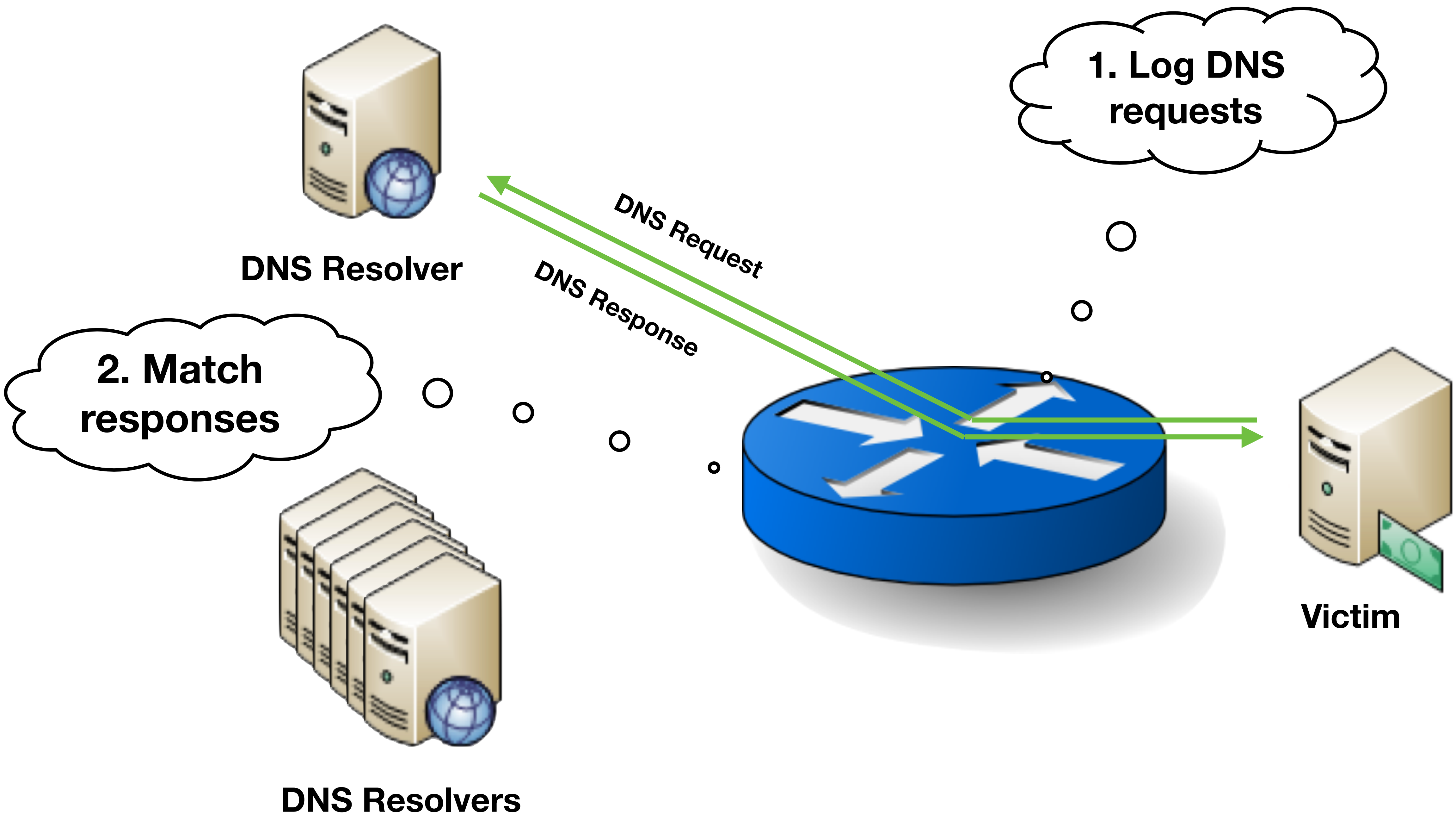
Bohatei: flexible and elastic DDoS defense, Fayaz et.al., USENIX SECURITY 15

Example - Detecting DNS Reflection Attacks



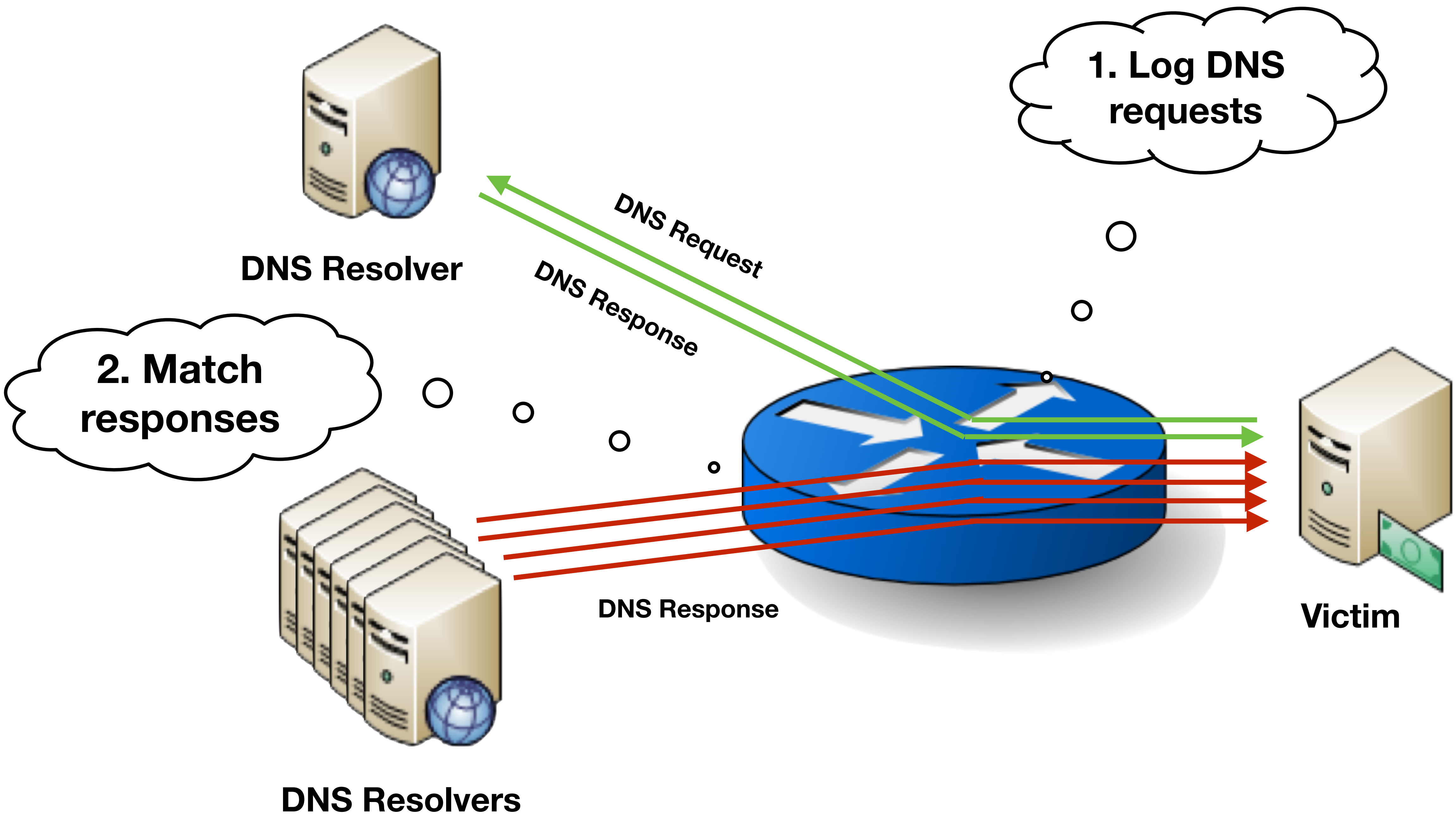
Bohatei: flexible and elastic DDoS defense, Fayaz et.al., USENIX SECURITY 15

Example - Detecting DNS Reflection Attacks



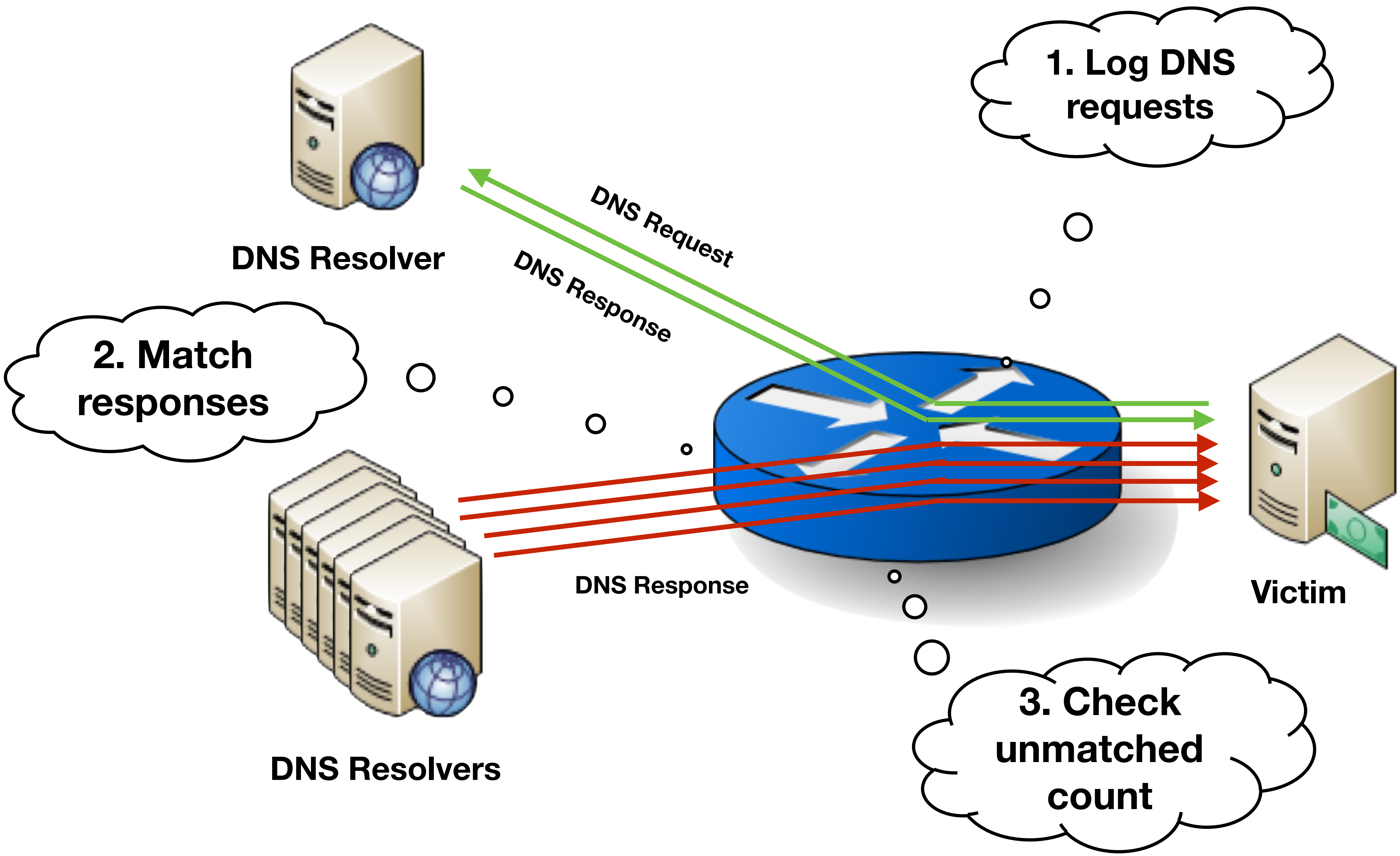
Bohatei: flexible and elastic DDoS defense, Fayaz et.al., USENIX SECURITY 15

Example - Detecting DNS Reflection Attacks



Bohatei: flexible and elastic DDoS defense, Fayaz et.al., USENIX SECURITY 15

Example - Detecting DNS Reflection Attacks

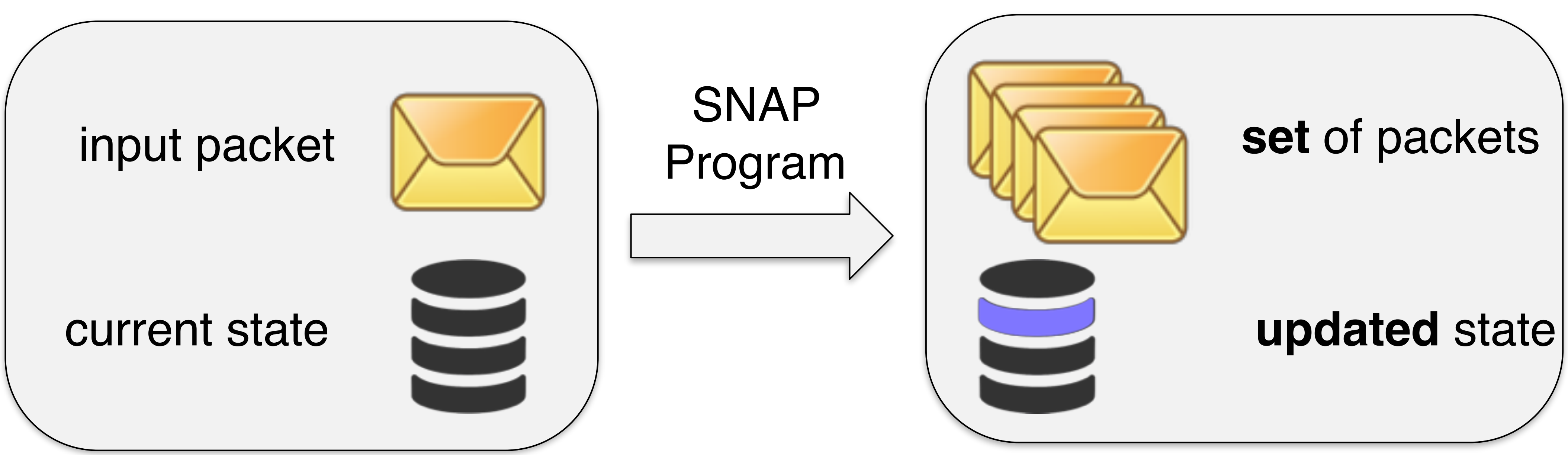


Bohatei: flexible and elastic DDoS defense, Fayaz et.al., USENIX SECURITY 15

SNAP Language

Stateful Packet Processing Functions

- A function specifying
  - how to process each packet
  - based on its **fields** and the **program state**



DNS Reflection Detection in SNAP

```
if (scrip in CSNET) & (dstport = DNS) then
   seen[srcip][dns.id] ← True
else if (dstip in CSNET) & (srcport = DNS) then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++
   if unmatched[dstip] = THRESH then
      susp[dstip] ← True
   else id
else id
```

DNS Reflection Detection in SNAP

```
if (scrip in CSNET) & (dstport = DNS) then
   seen[srcip][dns.id] ← True
else if (dstip in CSNET) & (srcport = DNS) then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++
   if unmatched[dstip] = THRESH then
      susp[dstip] ← True
   else id
else id
```

DNS Reflection Detection in SNAP

```
if (scrip in CSNET) & (dstport = DNS) then
   seen[srcip][dns.id] ← True
else if (dstip in CSNET) & (srcport = DNS) then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++
   if unmatched[dstip] = THRESH then
      susp[dstip] ← True

   else id
else id
```

DNS Reflection Detection in SNAP

```
if (scrip in CSNET) & (dstport = DNS) then
   seen[srcip][dns.id] ← True
else if (dstip in CSNET) & (srcport = DNS) then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++
   if unmatched[dstip] = THRESH then
      susp[dstip] ← True
   else id
else id
```

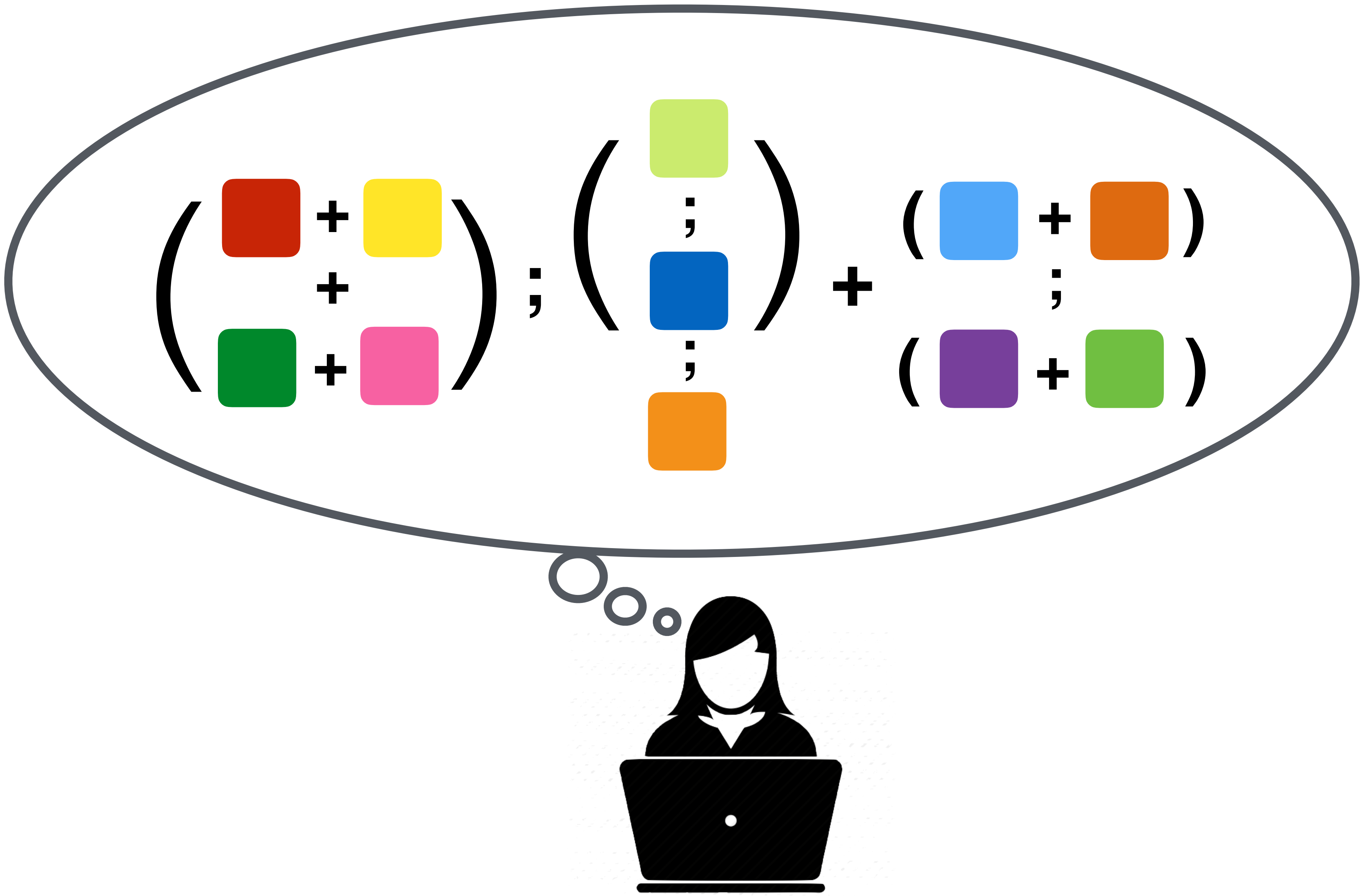
DNS Reflection Detection in SNAP

```
if (scrip in CSNET) & (dstport = DNS) then
   seen[srcip][dns.id] ← True
else if (dstip in CSNET) & (srcport = DNS) then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++
   if unmatched[dstip] = THRESH then
      susp[dstip] ← True
   else id
else id
```

DNS Reflection Detection in SNAP

```
if (scrip in CSNET) & (dstport = DNS) then
   seen[srcip][dns.id] ← True
else if (dstip in CSNET) & (srcport = DNS) then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++
   if unmatched[dstip] = THRESH then
      susp[dstip] ← True
   else id
else id
```

Program Composition



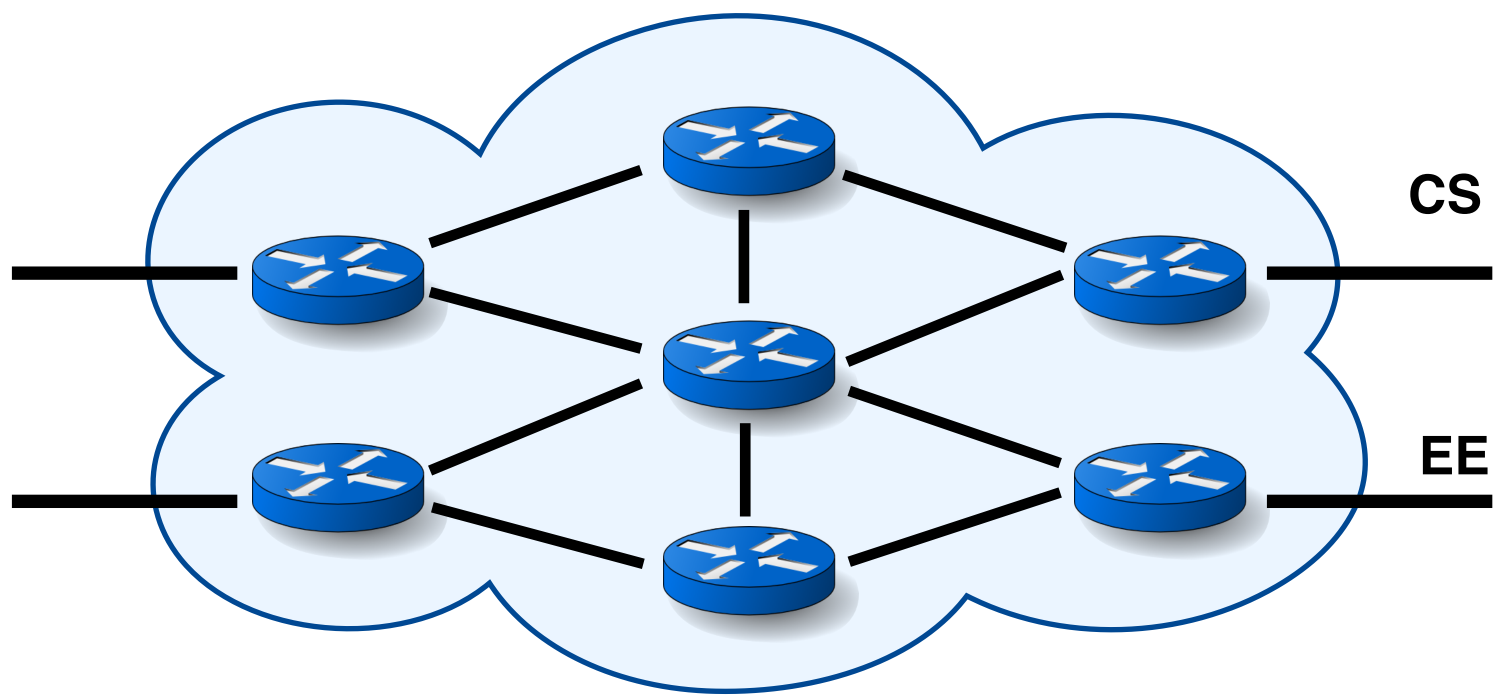
SNAP Compiler

Where to Place State Variables?

```
if srcip in CSNET & dstport = 53 then
   seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++;
      if unmatched[dstip] = threshold then
         susp[dstip] ← True
   else id
else id
```

;

```
if dstip = CSNET then outport ← CS
else if dstip = EENET then outport ← EE
else if dstip = ISP1NET then outport ← ISP1
else if dstip = ISP2NET then outport ← ISP2
else drop
```

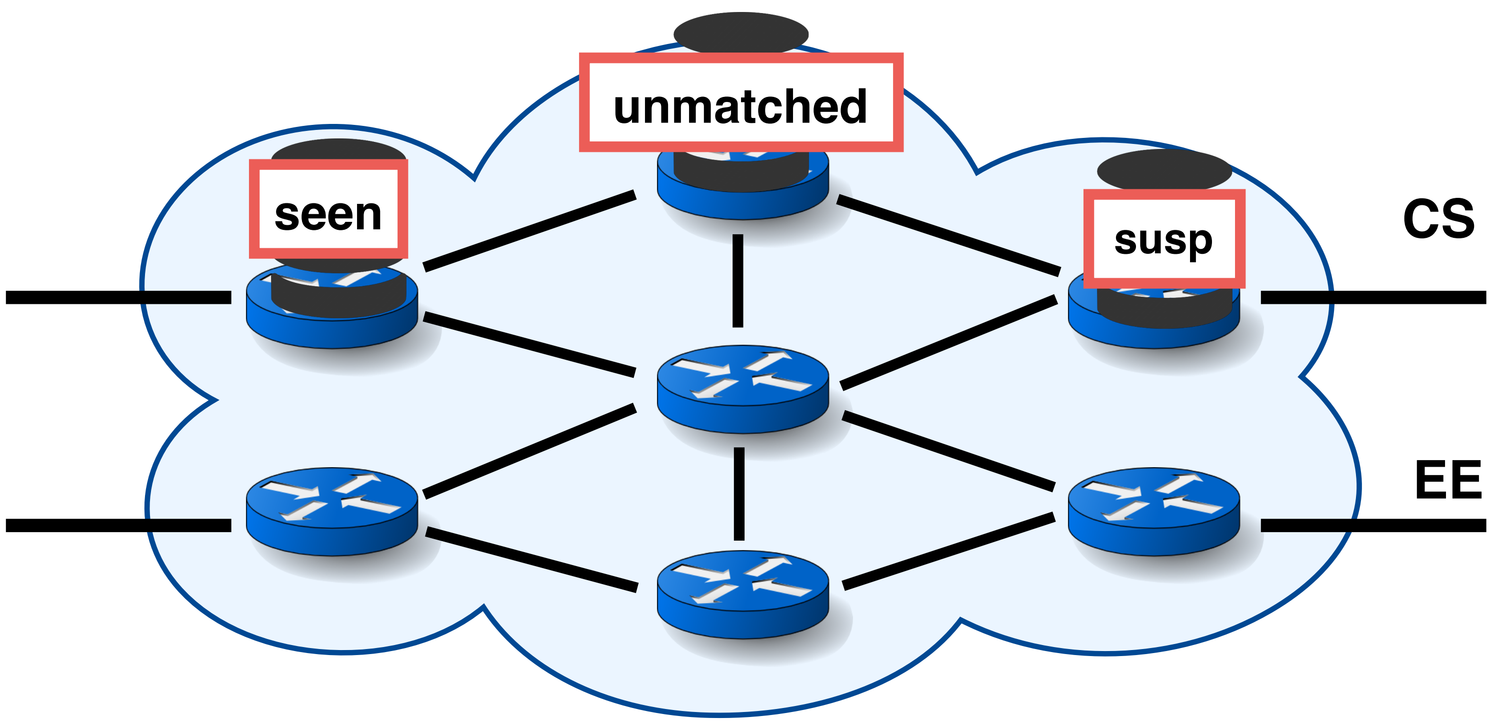


How to Forward Packets through State Variables?

```
if srcip in CSNET & dstport = 53 then
   seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++;
      if unmatched[dstip] = threshold then
         susp[dstip] ← True
   else id
else id
```

;

```
if dstip = CSNET then outport ← CS
else if dstip = EENET then outport ← EE
else if dstip = ISP1NET then outport ← ISP1
else if dstip = ISP2NET then outport ← ISP2
else drop
```

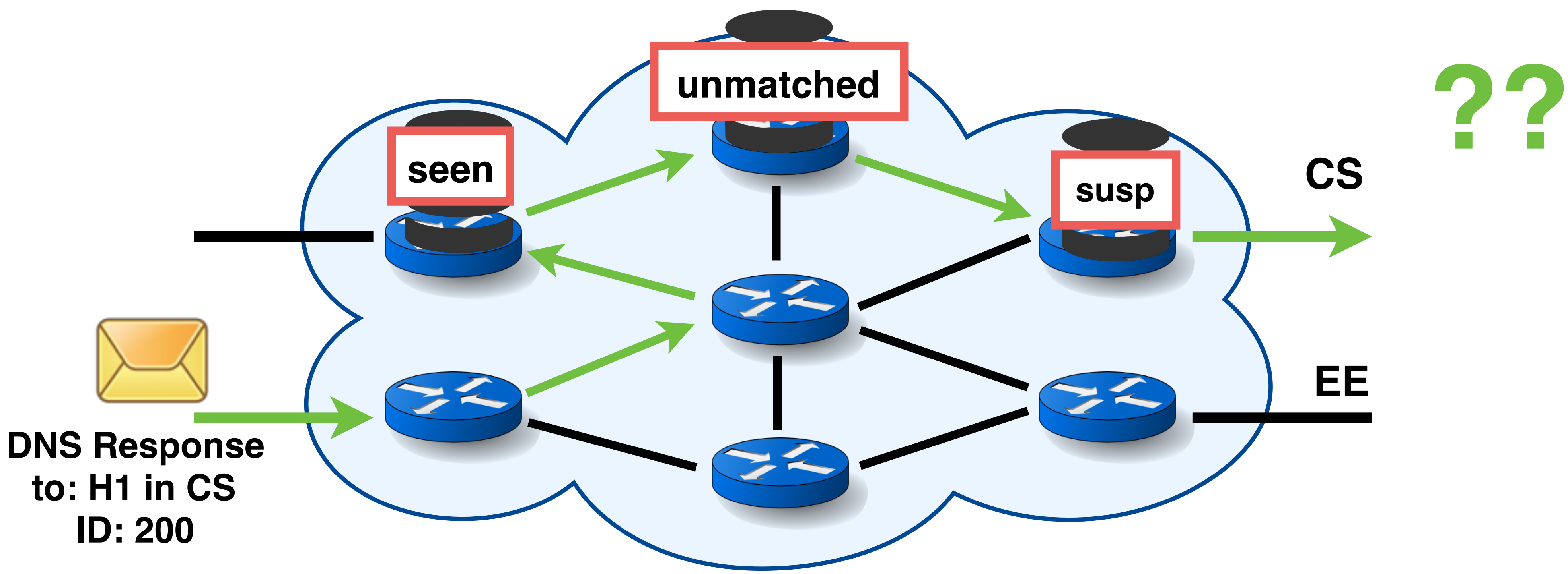


How to Forward Packets through State Variables?

```
if srcip in CSNET & dstport = 53 then
   seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++;
      if unmatched[dstip] = threshold then
         susp[dstip] ← True
   else id
else id
```

;

```
if dstip = CSNET then outport ← CS
else if dstip = EENET then outport ← EE
else if dstip = ISP1NET then outport ← ISP1
else if dstip = ISP2NET then outport ← ISP2
else drop
```



Program Analysis

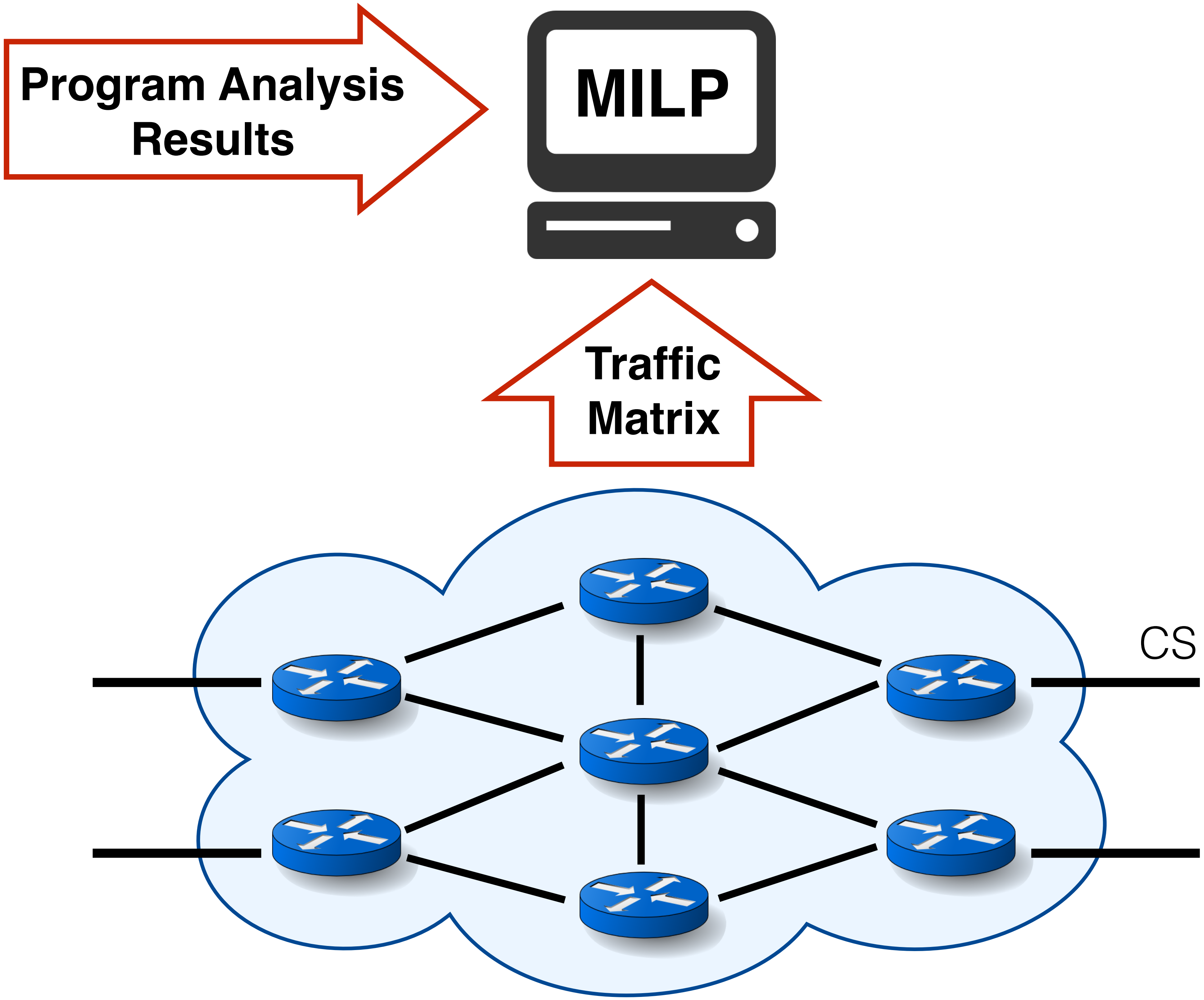
Program Analysis

- For each flow, find
  - all the state variables that it needs
  - the order in which the state variables should be visited

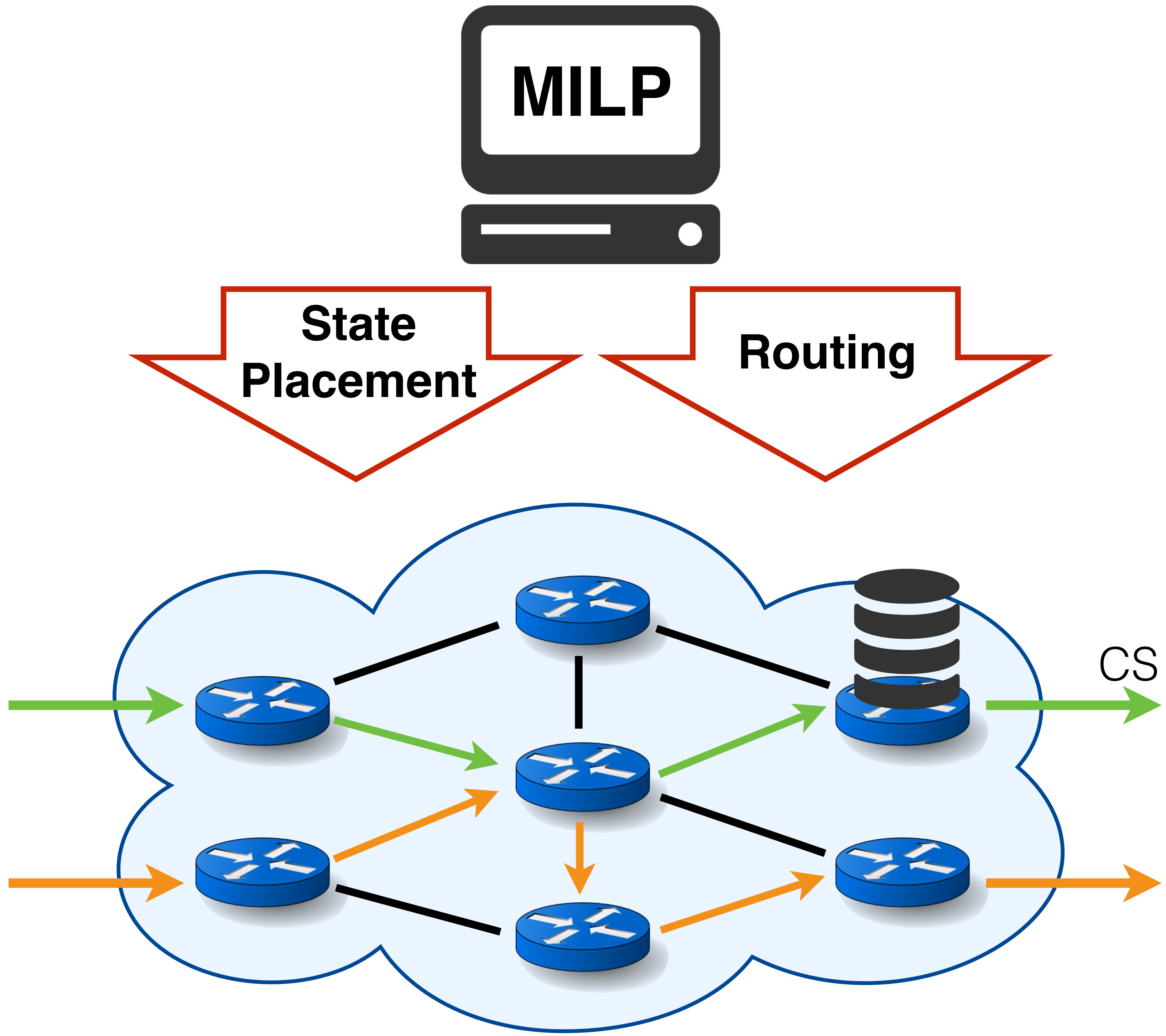
Program Analysis

- For each flow, find
  - all the state variables that it needs
  - the order in which the state variables should be visited
- A flow can be defined at any granularity
  - As long as its traffic statistics is available
  - E.g., All DNS packets to the CS subnet need all three state variables
  - E.g., All packets from the CS subnet need `seen`.

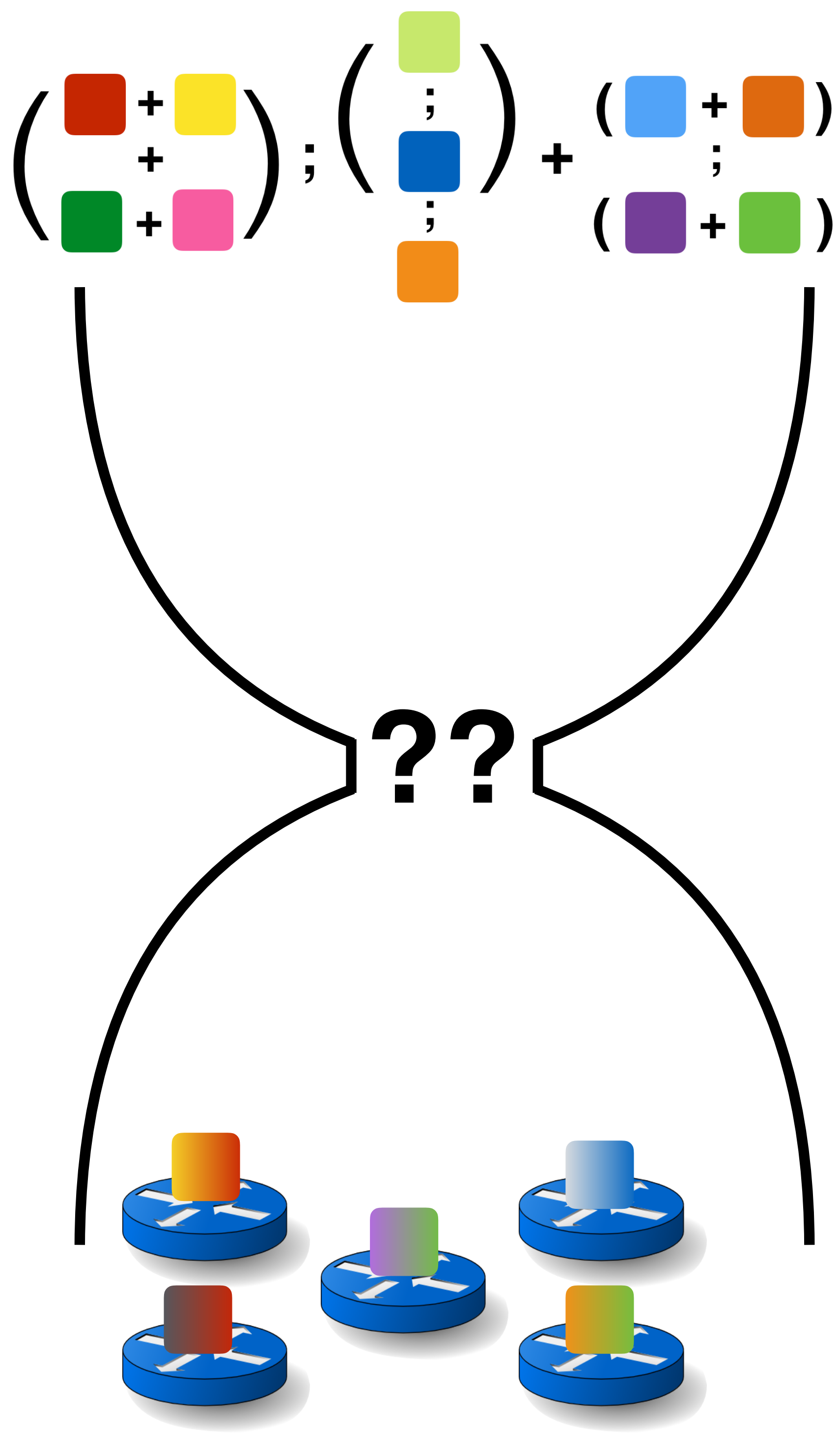
Mixed-Integer Linear Program (MILP)



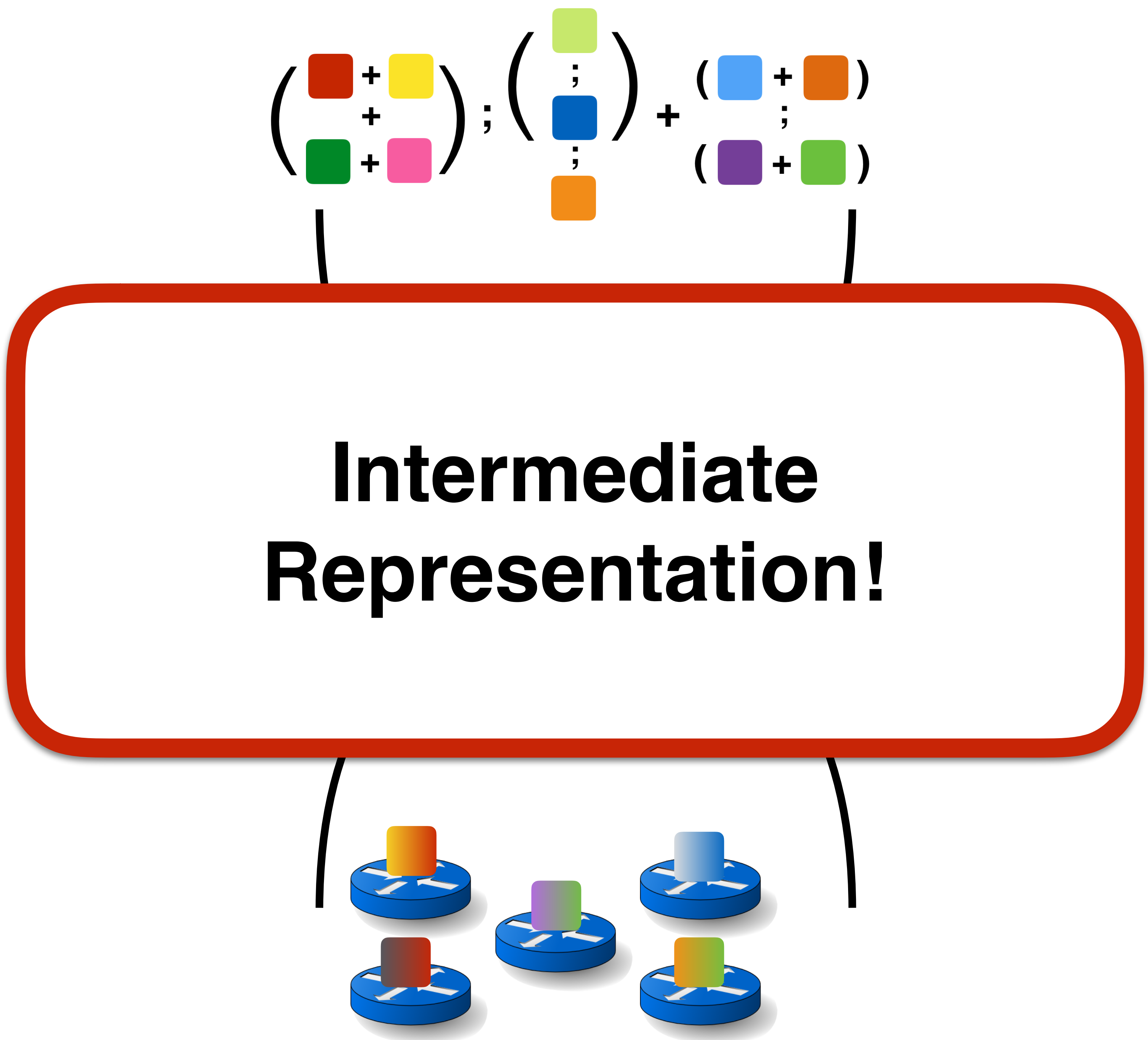
Mixed-Integer Linear Program (MILP)



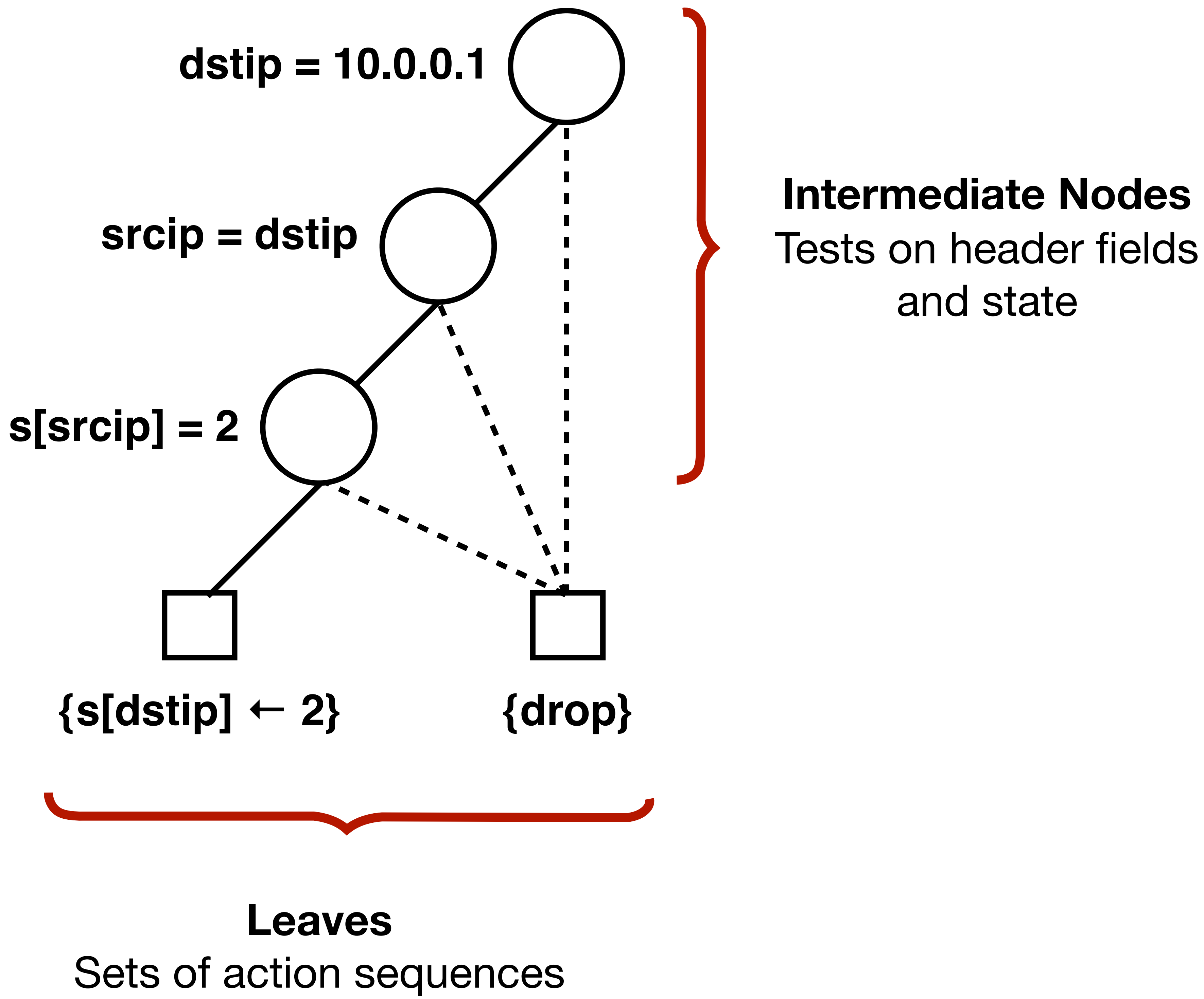
How to distribute a SNAP program?



How to distribute a SNAP program?



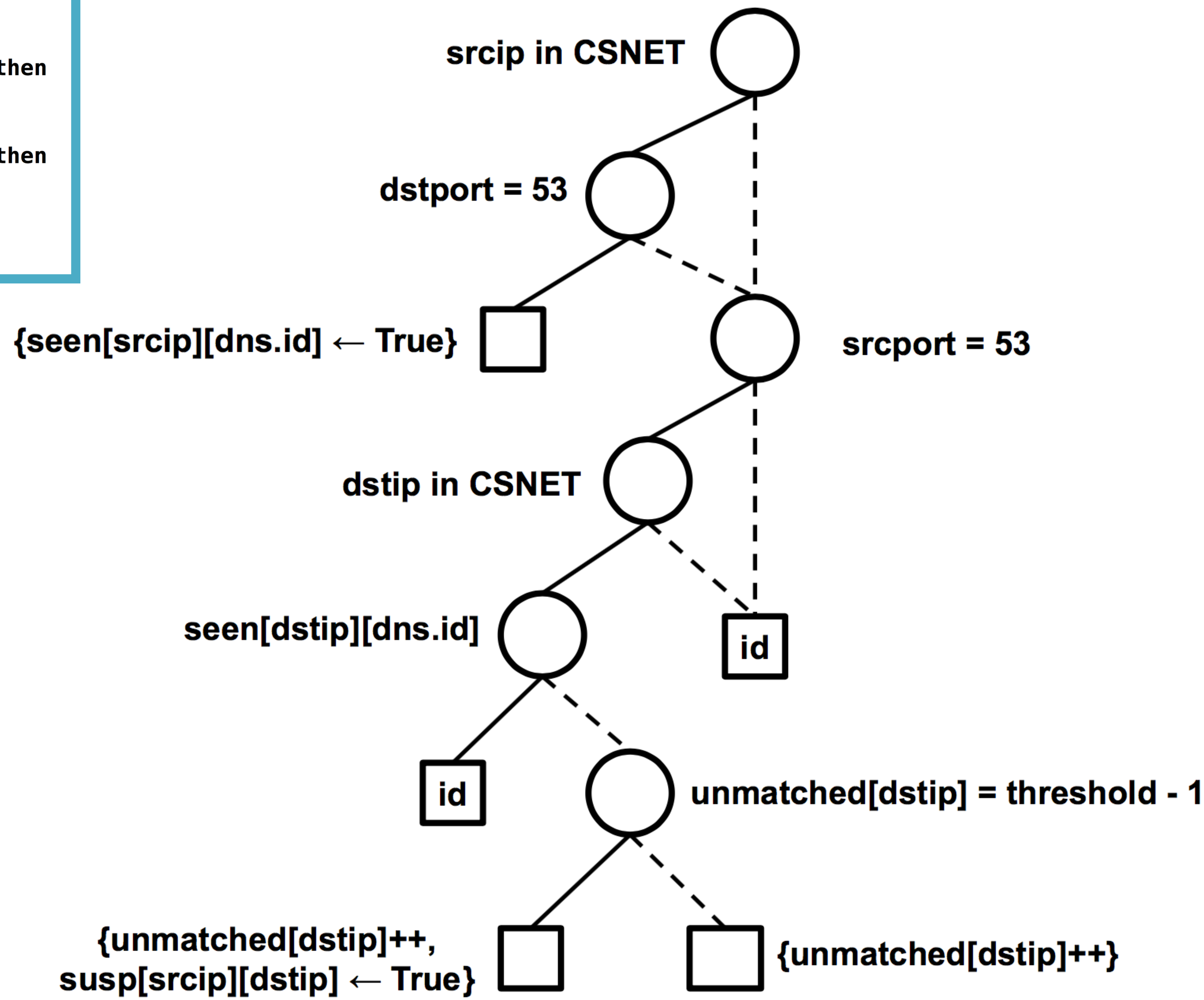
Extended Forwarding Decision Diagrams (xFDDs)



69

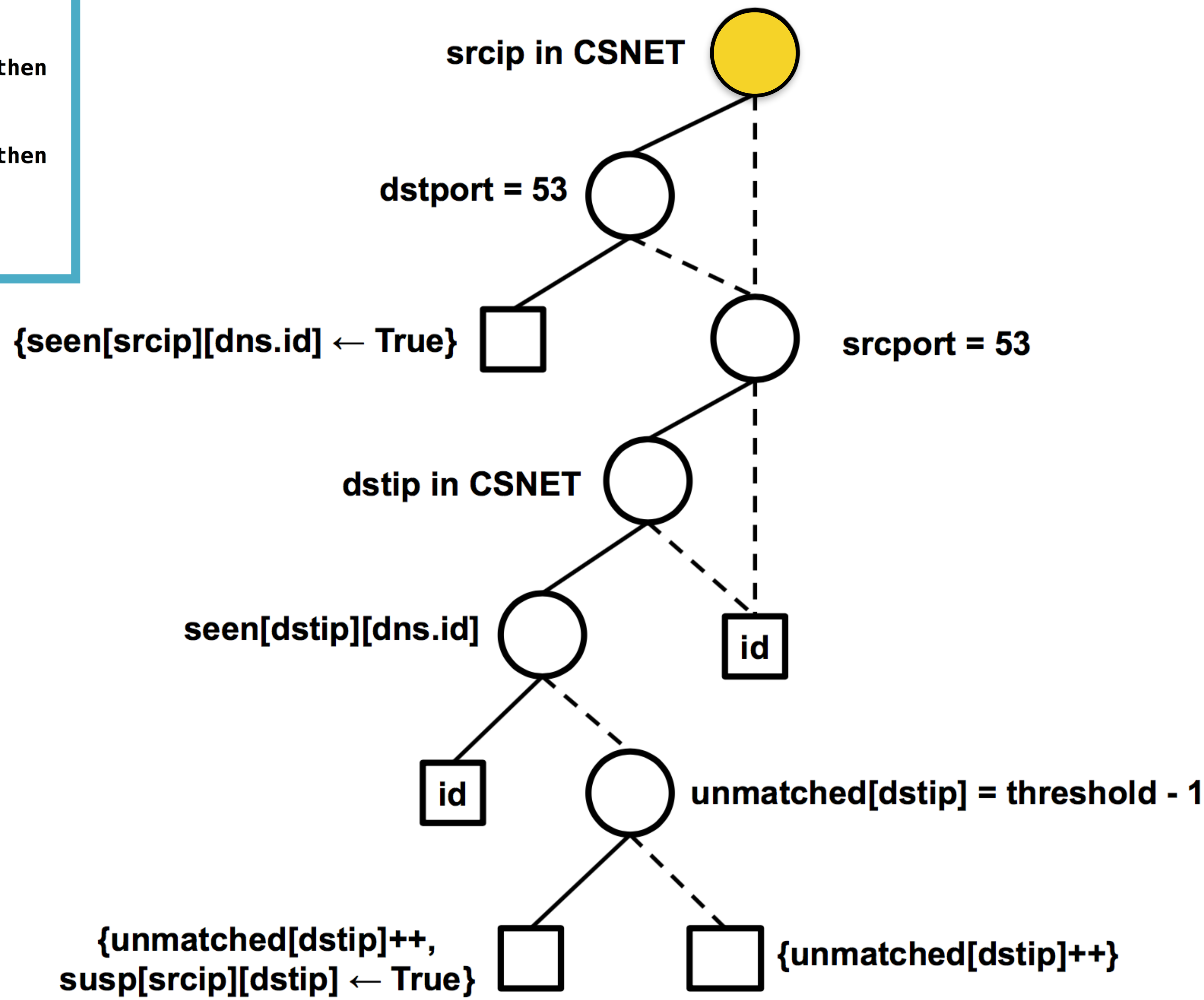
xFDD for DNS Reflection Detection

```
if srcip in CSNET & dstport = 53 then
   seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++;
      if unmatched[dstip] = threshold then
         susp[dstip] ← True
   else id
else id
```



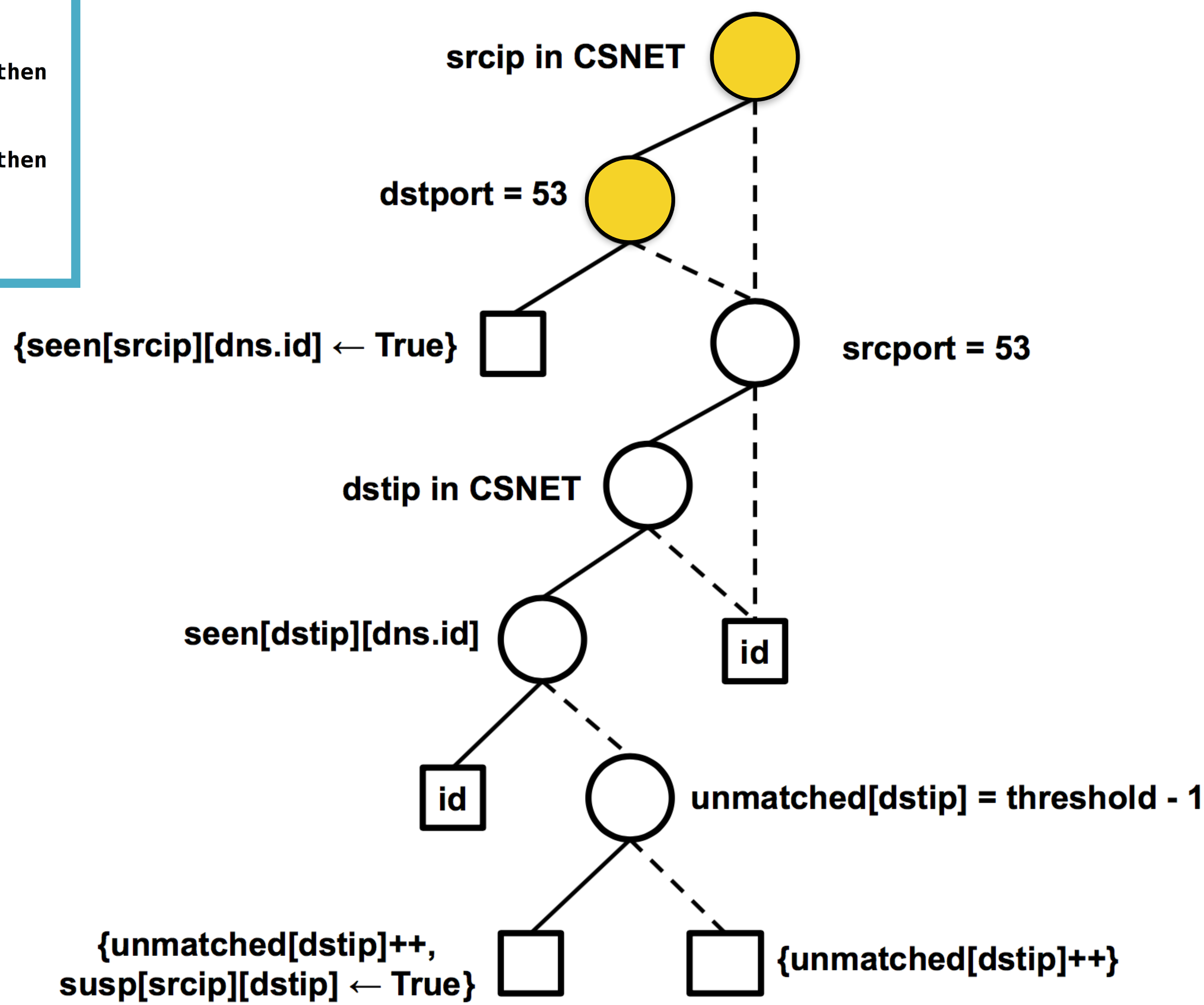
xFDD for DNS Reflection Detection

```
if srcip in CSNET & dstport = 53 then
   seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++;
      if unmatched[dstip] = threshold then
         susp[dstip] ← True
   else id
else id
```



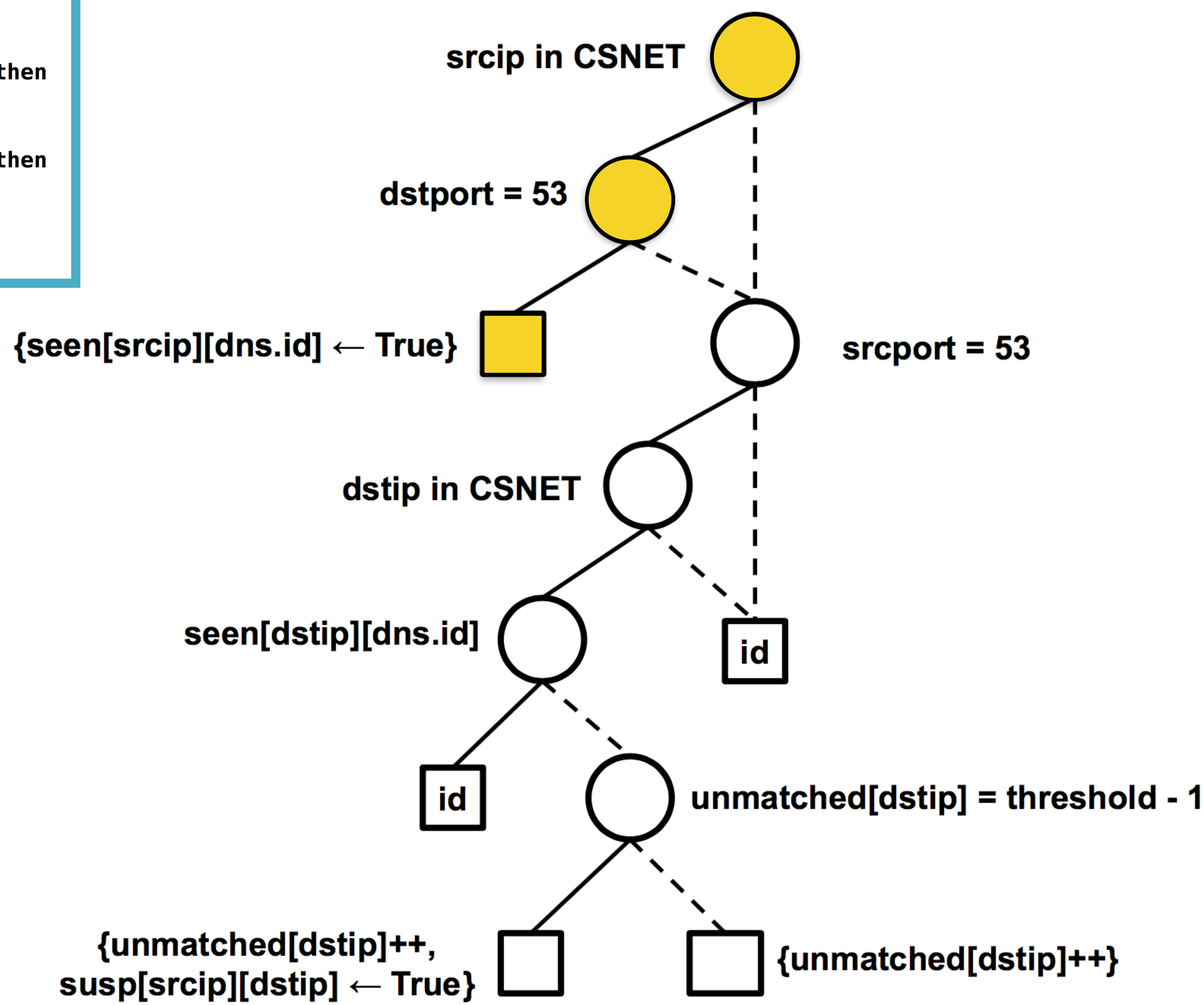
xFDD for DNS Reflection Detection

```
if srcip in CSNET & dstport = 53 then
   seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++;
      if unmatched[dstip] = threshold then
         susp[dstip] ← True
   else id
else id
```

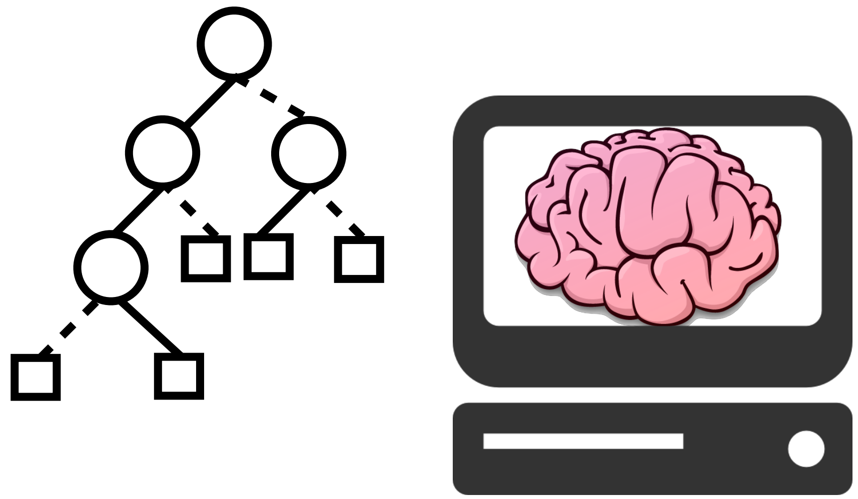


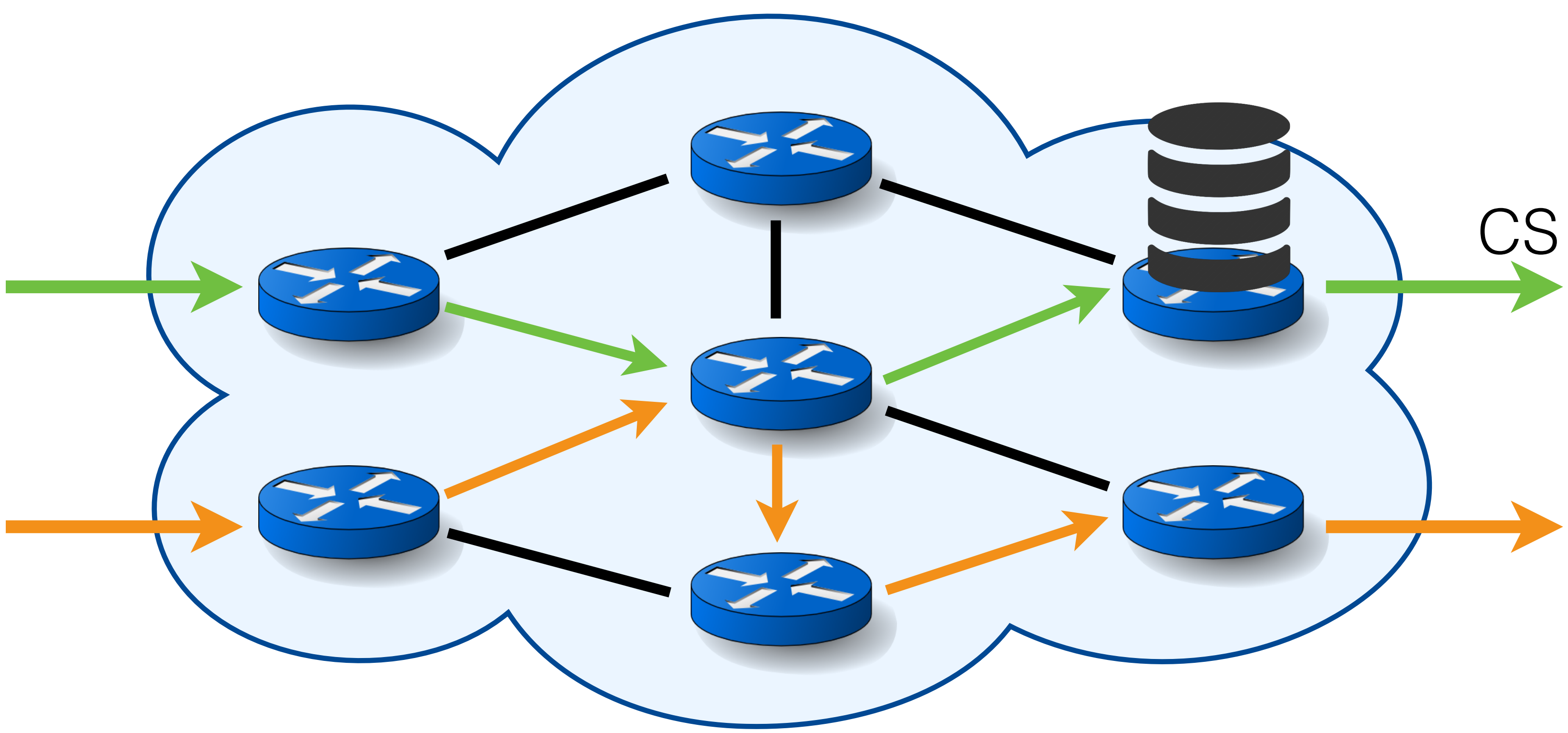
xFDD for DNS Reflection Detection

```
if srcip in CSNET & dstport = 53 then
   seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
   if ~seen[dstip][dns.id] then
      unmatched[dstip]++;
      if unmatched[dstip] = threshold then
         susp[dstip] ← True
   else id
else id
```

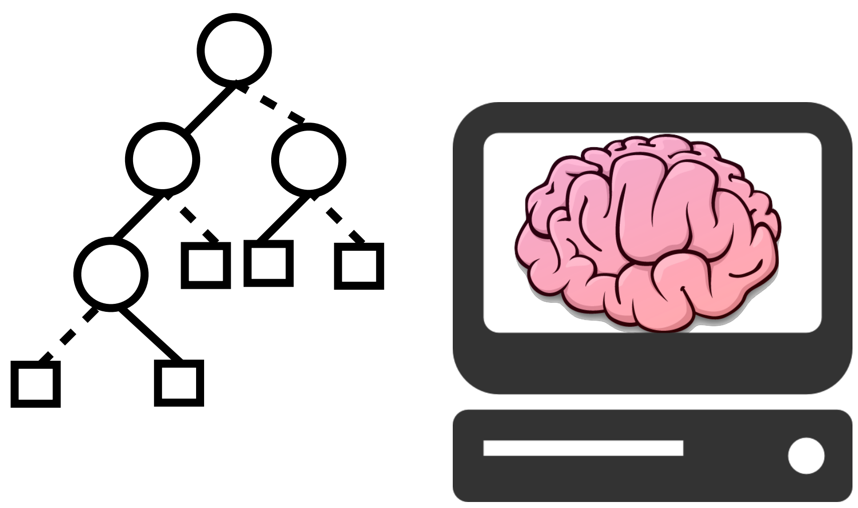


Partitioning and Distribution of the xFDD

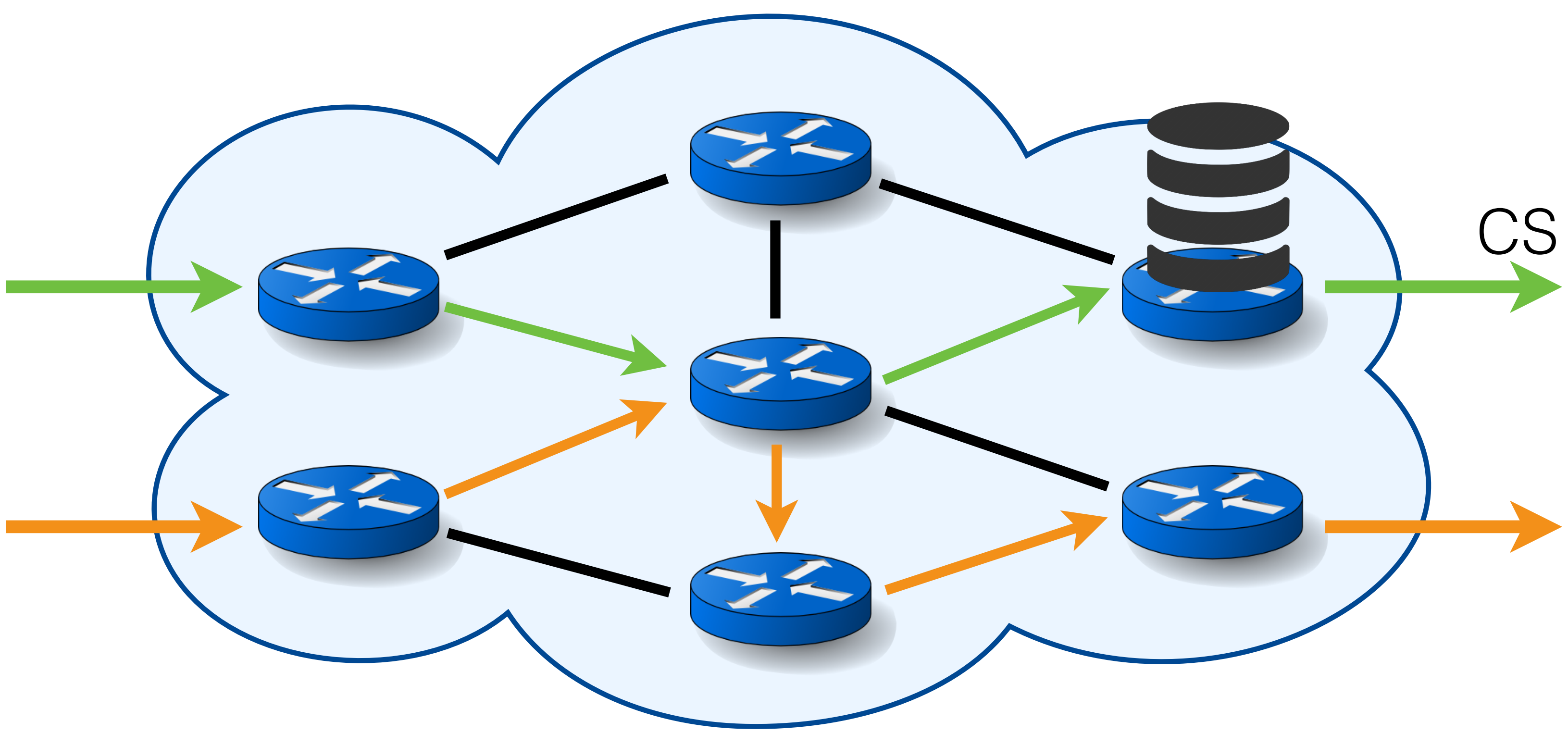




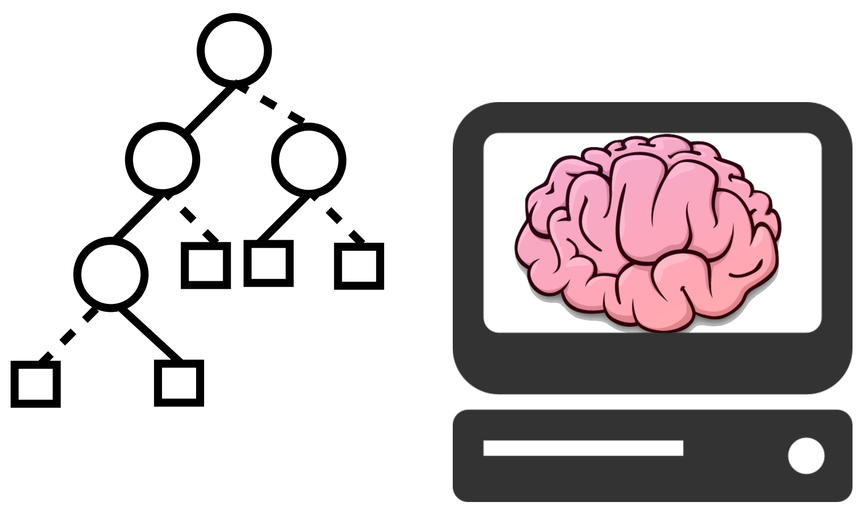
Partitioning and Distribution of the xFDD



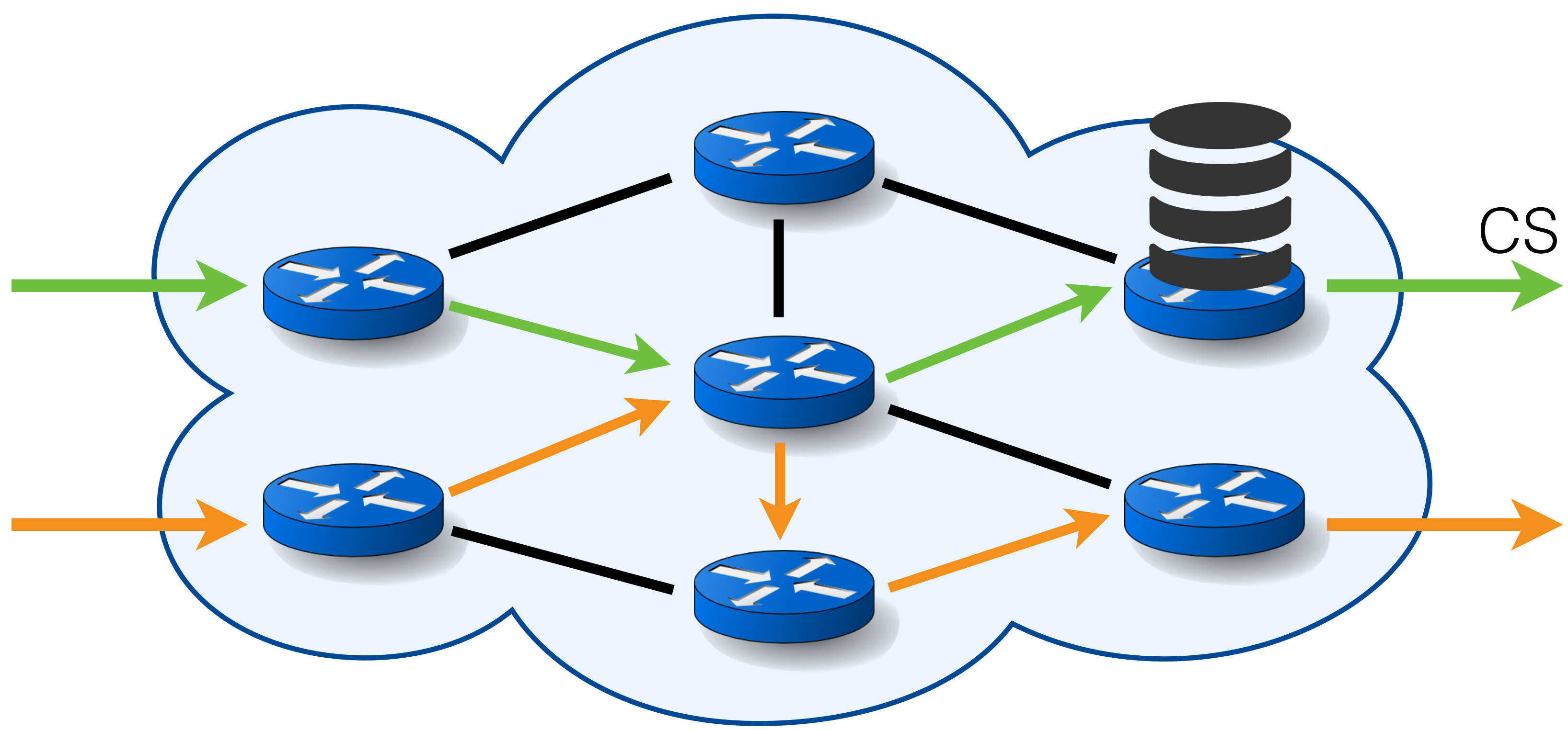
- The stateless tests and actions are at the top.



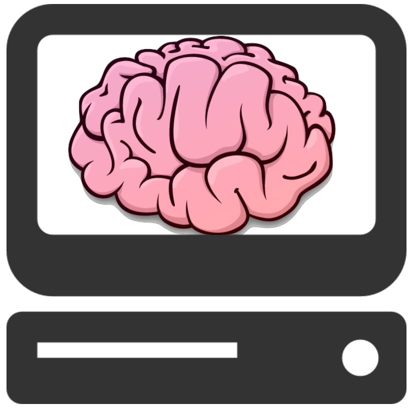
Partitioning and Distribution of the xFDD



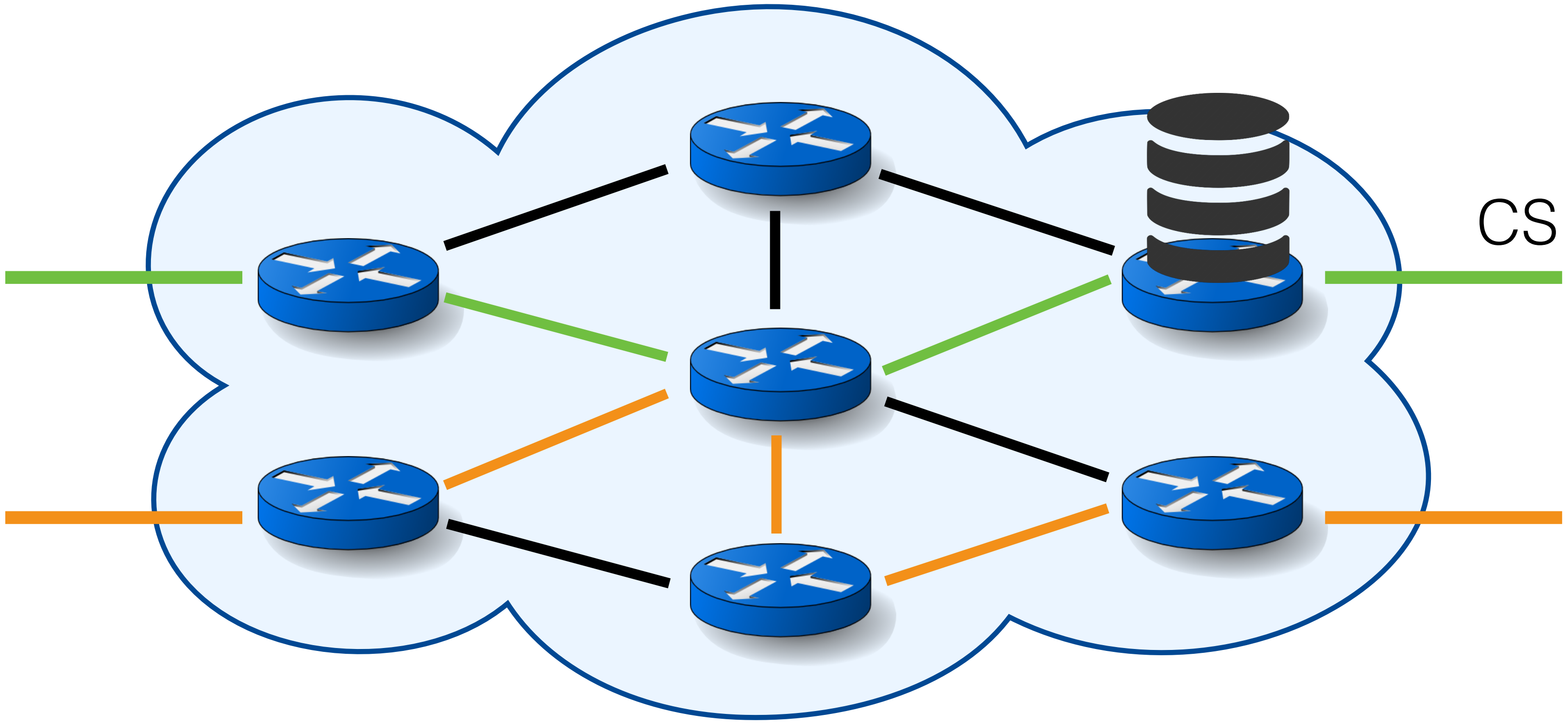
- The stateless tests and actions are at the top.
- Stateful tests and actions on the same variable form subtrees.



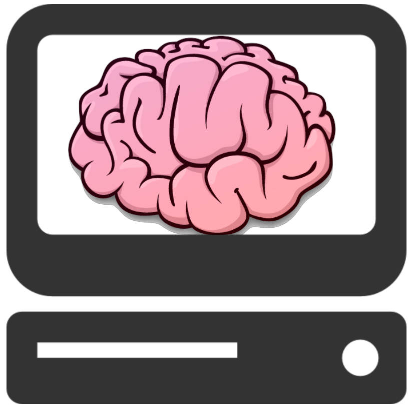
Partitioning and Distribution of the xFDD



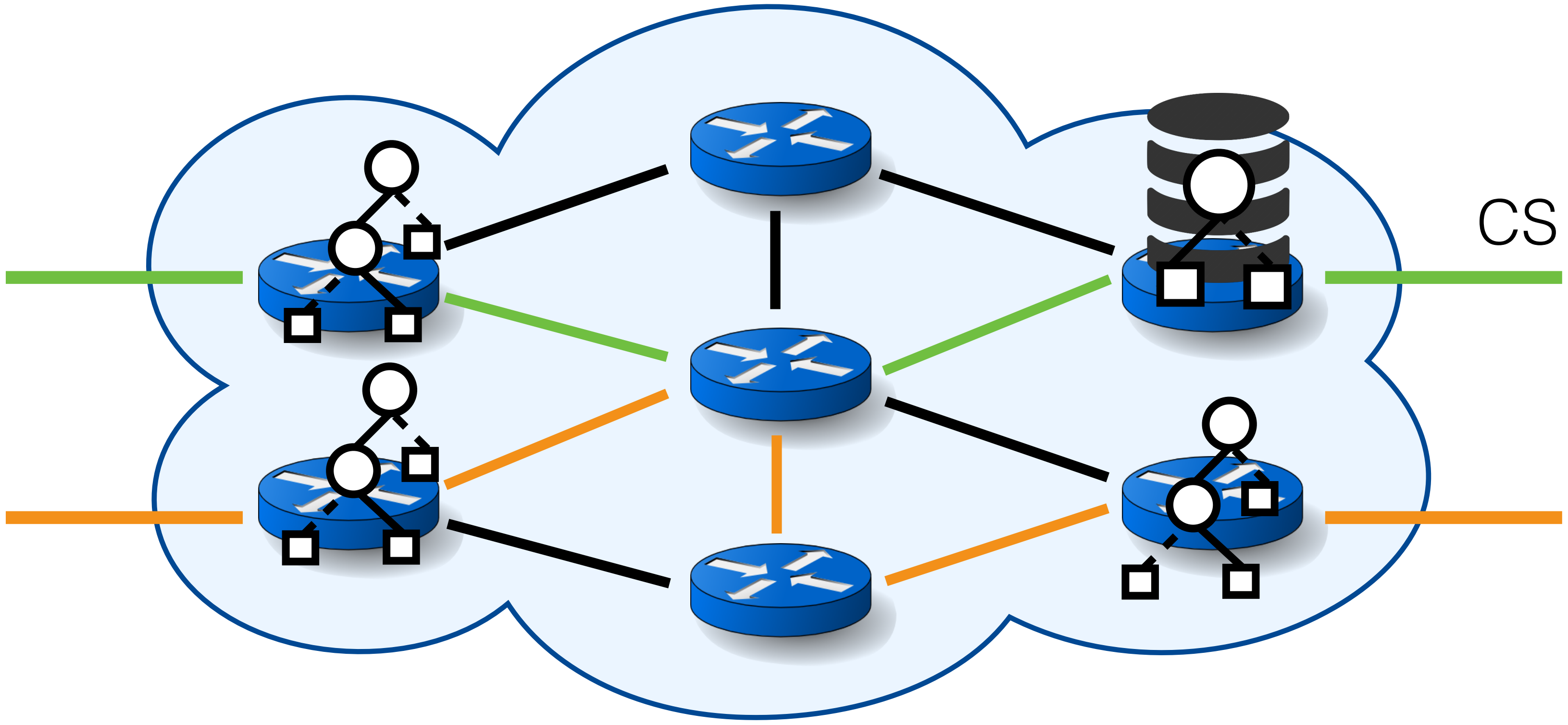
- The stateless tests and actions go to the edge.
- Subtrees of a stateful variable go the switch storing it.



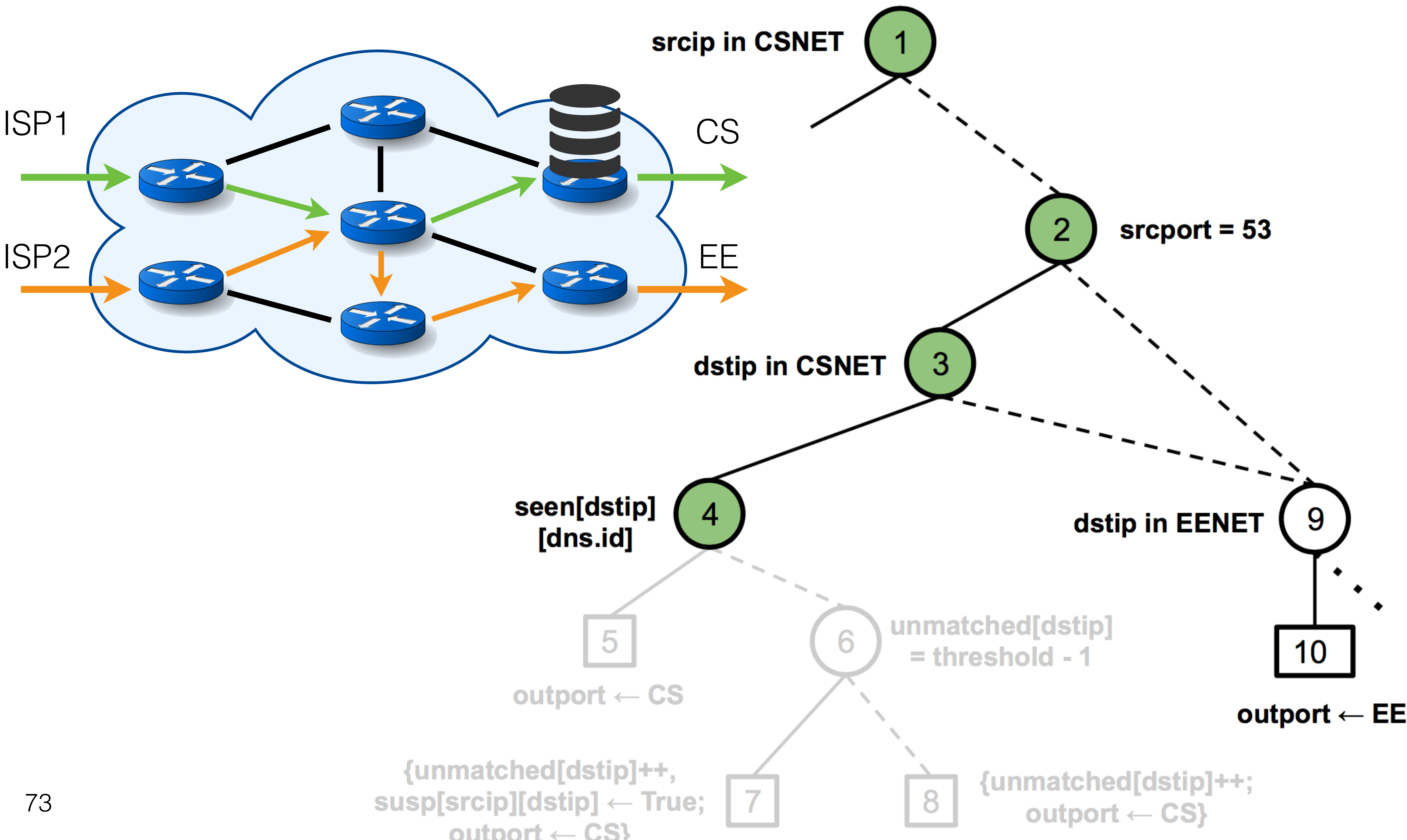
Partitioning and Distribution of the xFDD



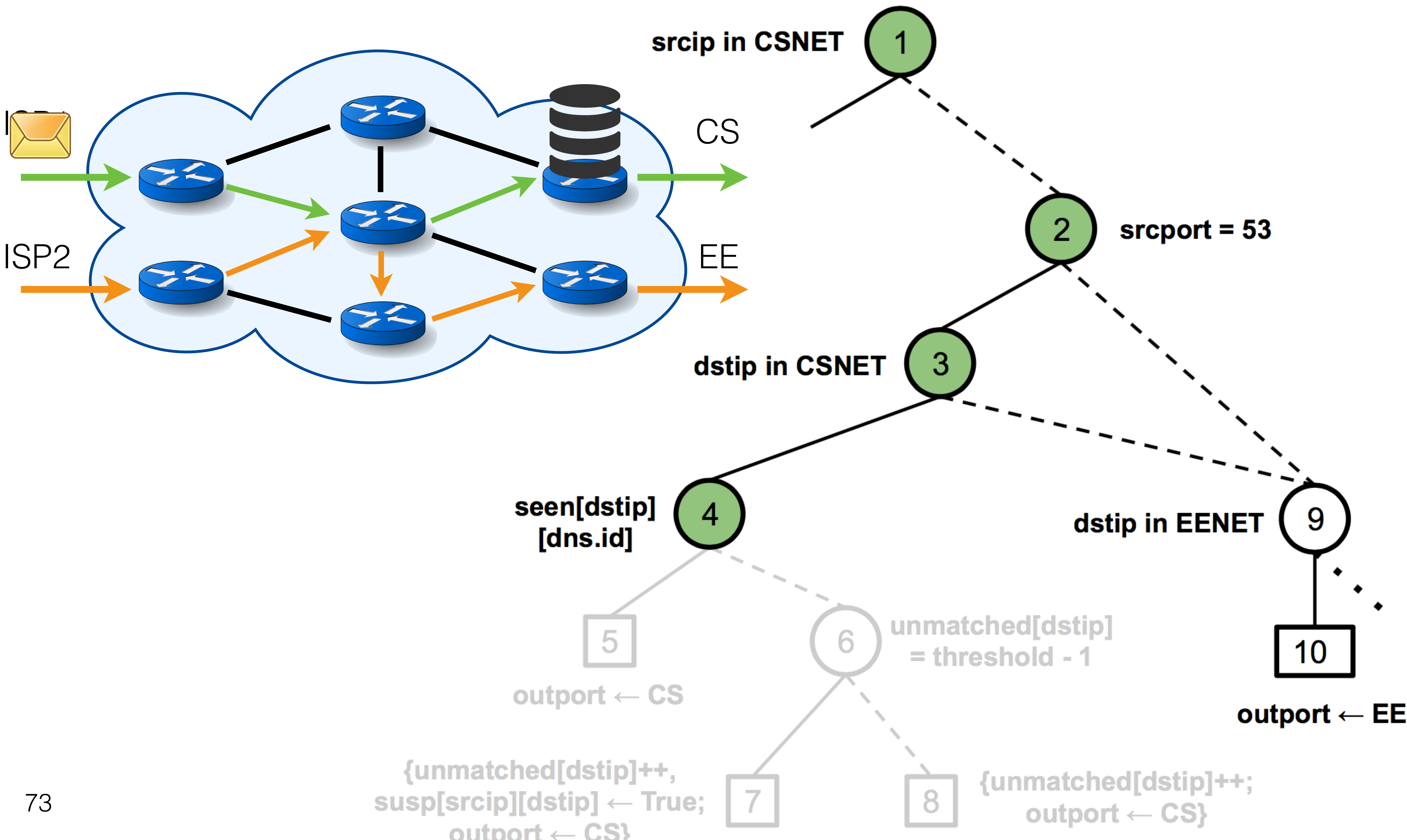
- The stateless tests and actions go to the edge.
- Subtrees of a stateful variable go the switch storing it.



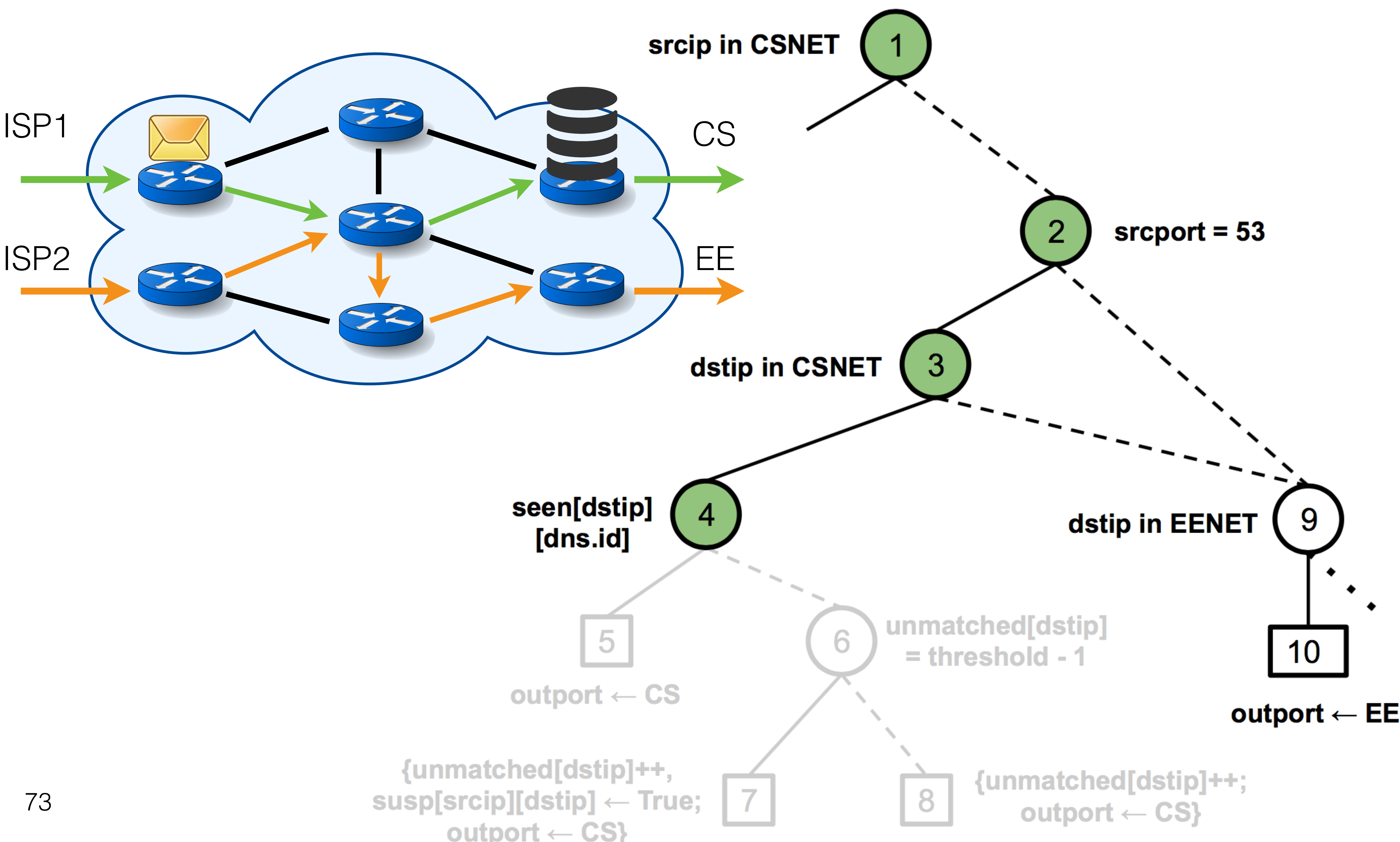
Putting It All Together



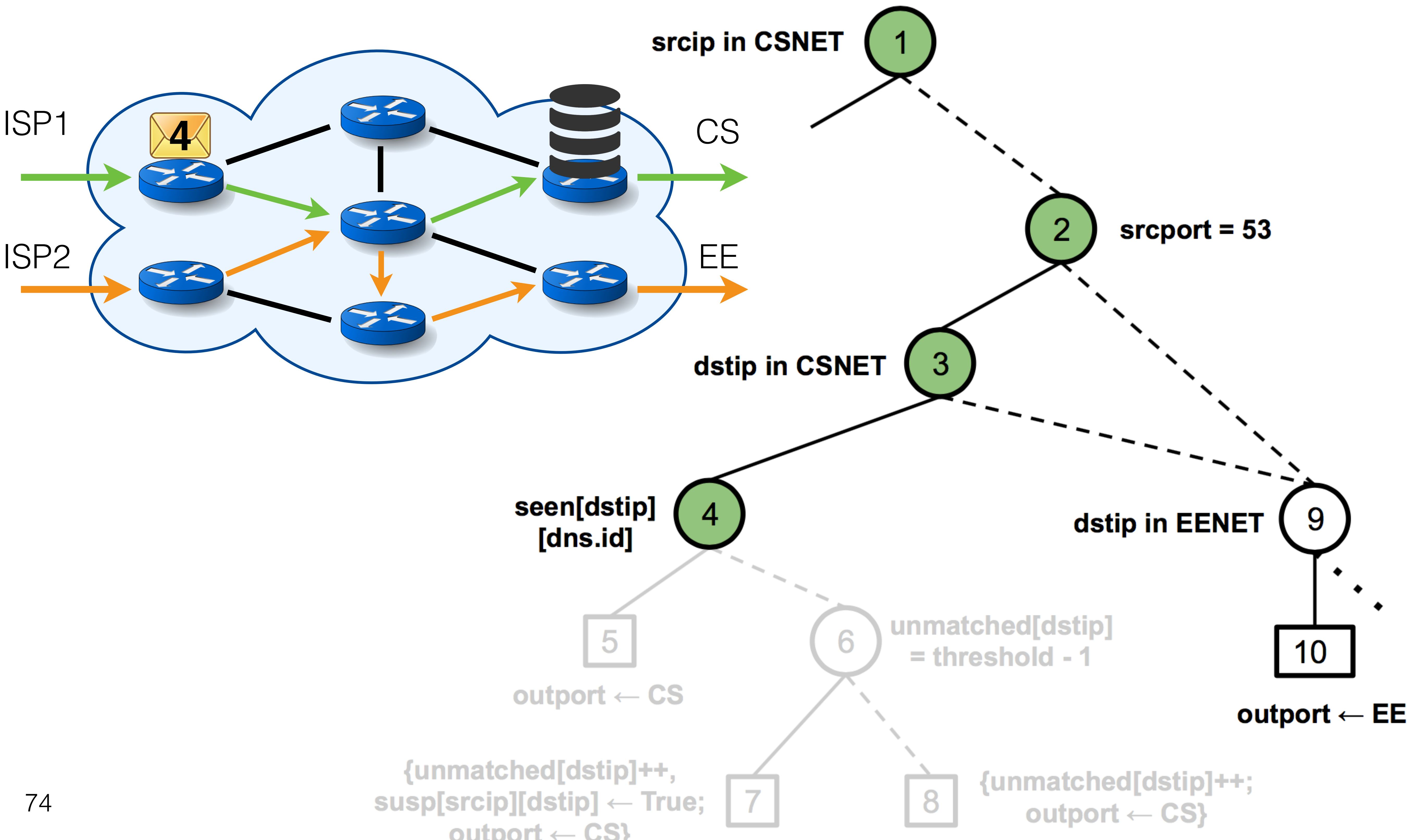
Putting It All Together



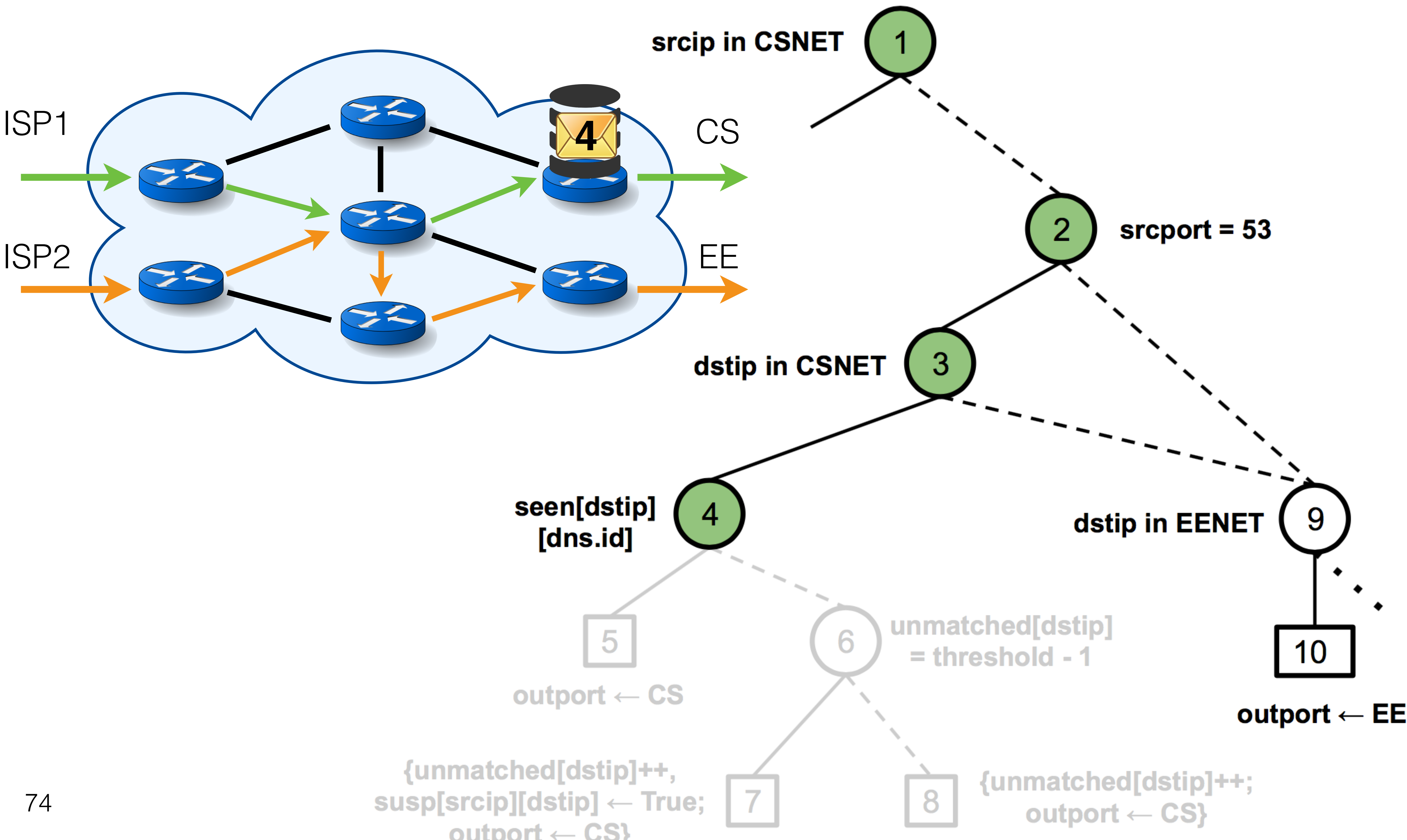
Putting It All Together



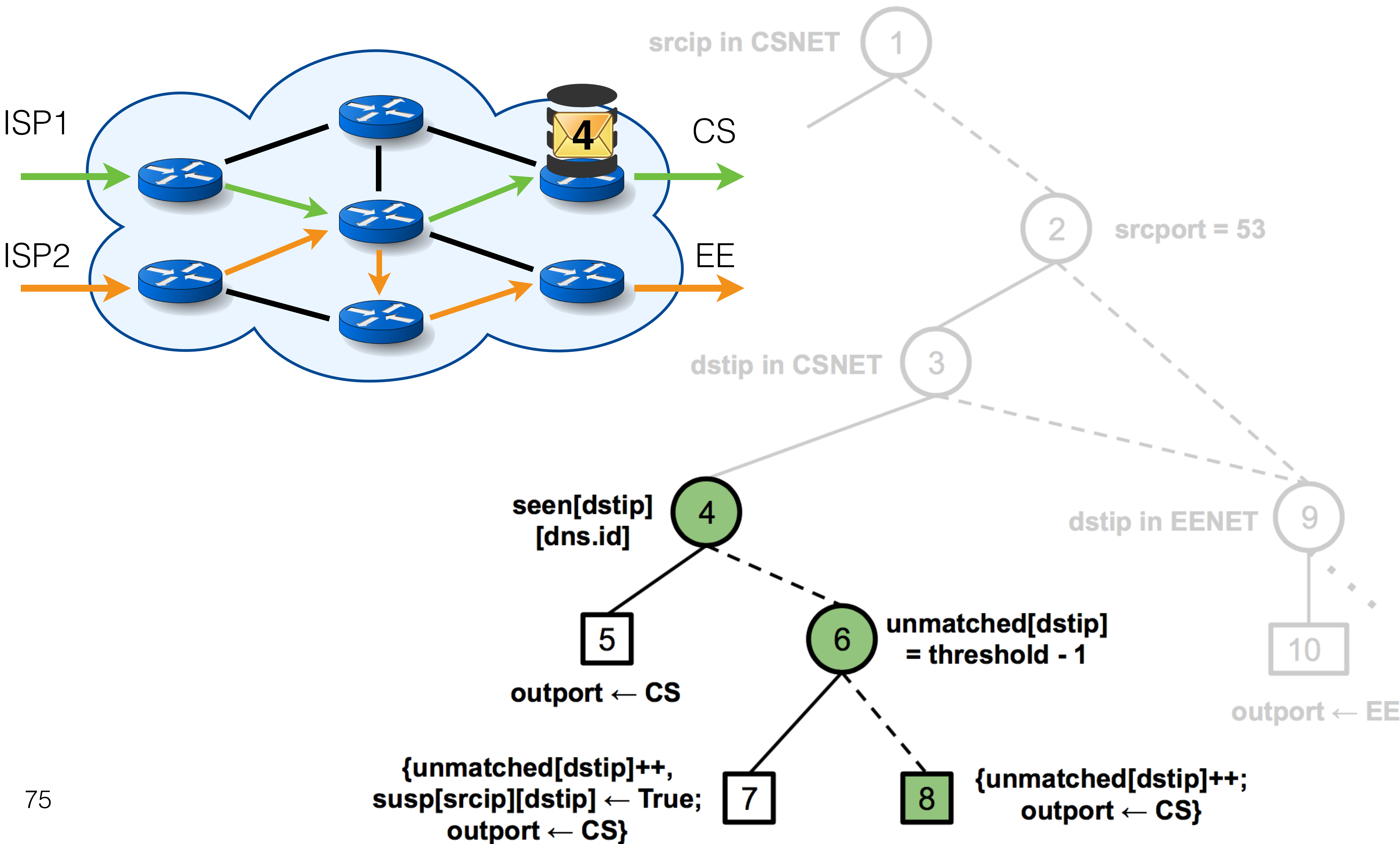
Putting It All Together



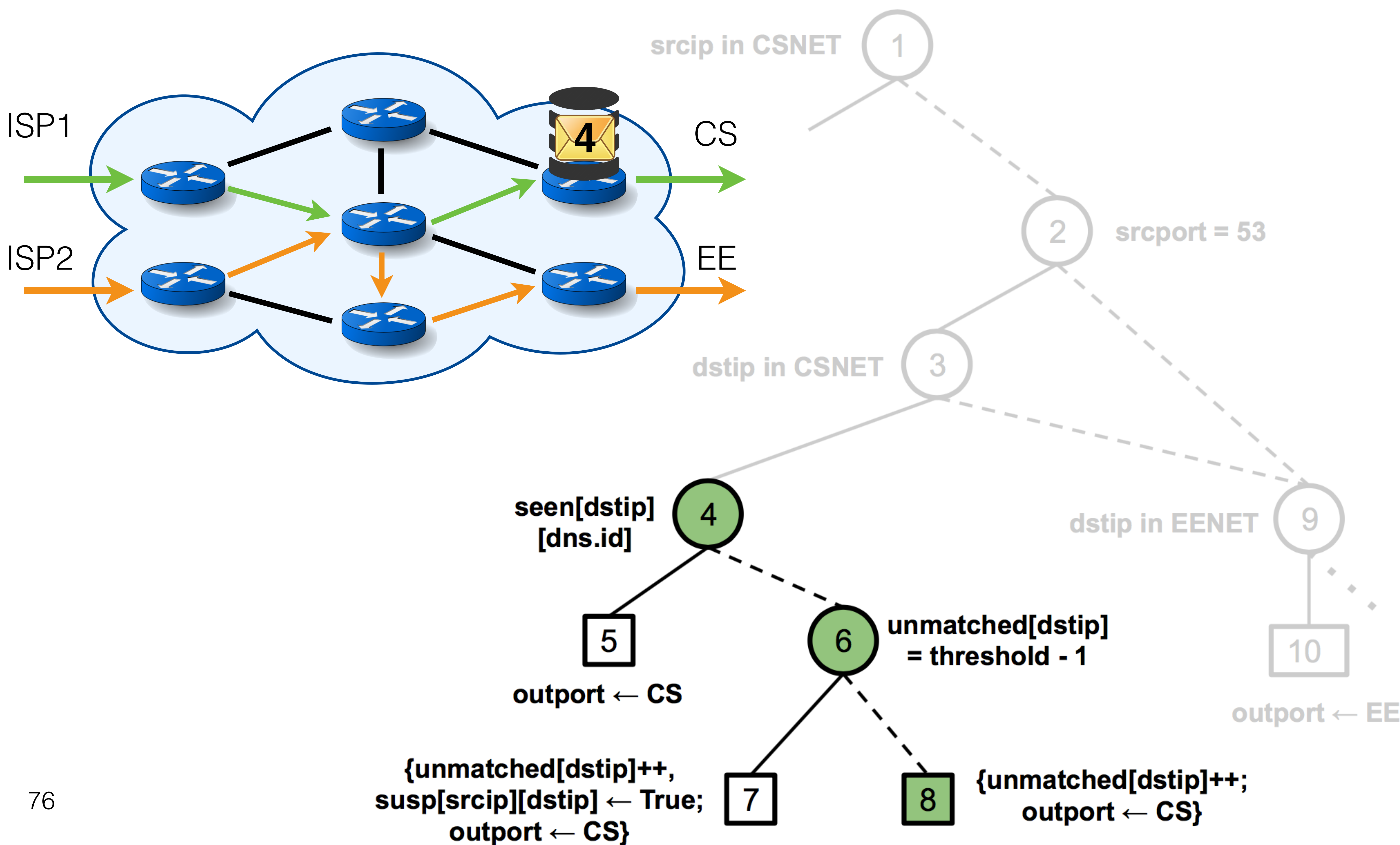
Putting It All Together



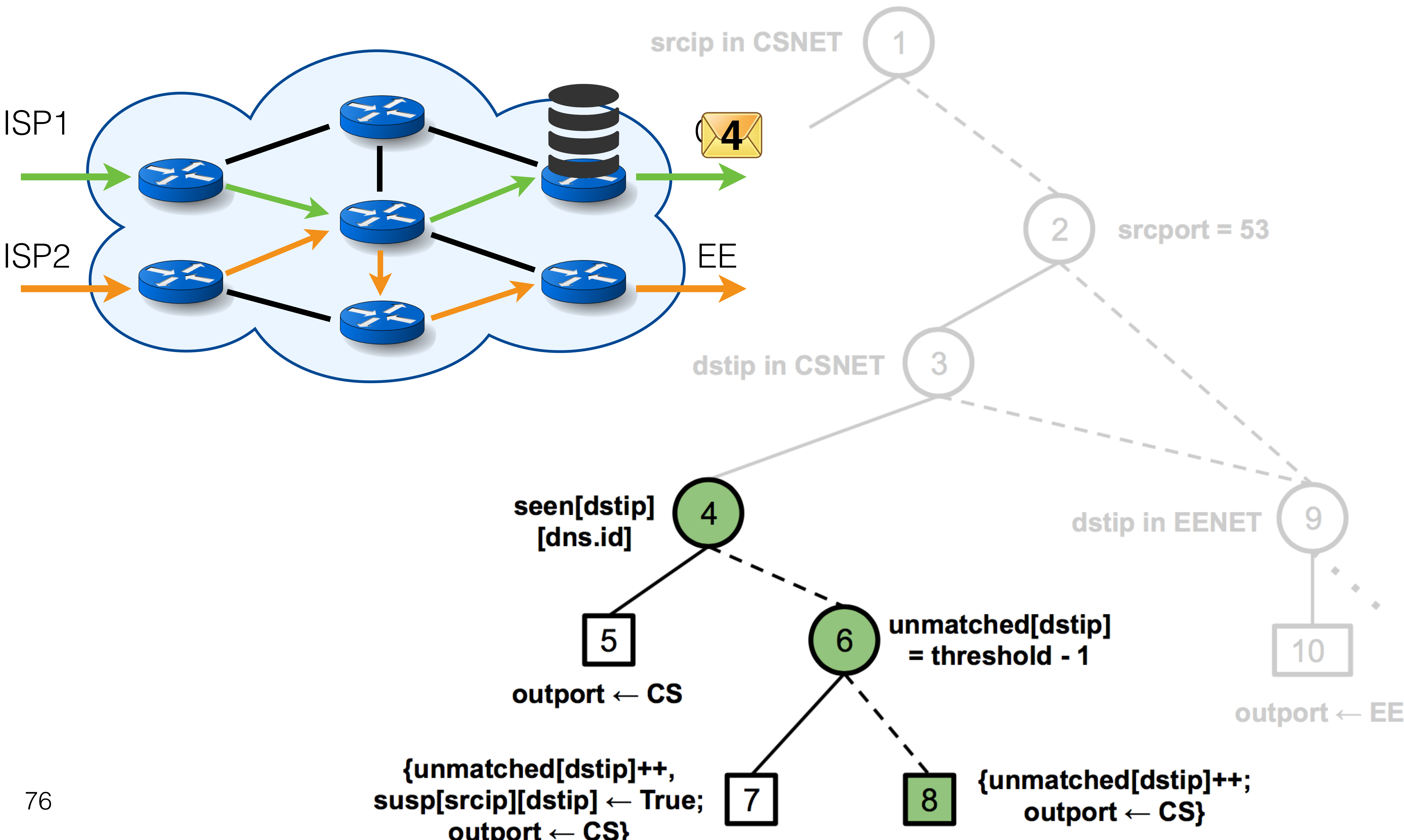
Putting It All Together



Putting It All Together



Putting It All Together



Compiler Evaluation

Compiler Evaluation

- 7 Campus and ISP topologies

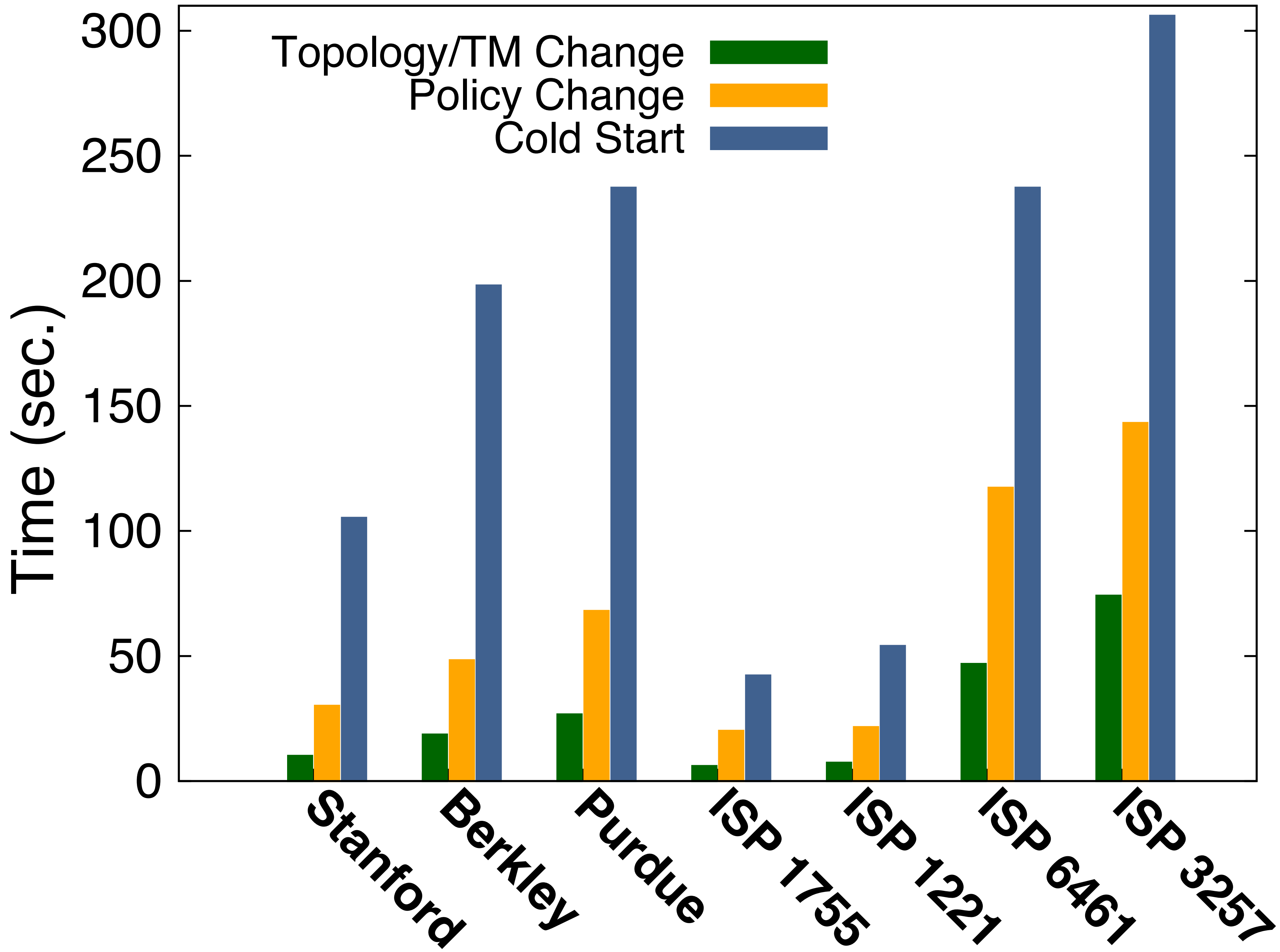
Compiler Evaluation

- 7 Campus and ISP topologies
- Order of 100s of switches and links

Compiler Evaluation

- 7 Campus and ISP topologies
- Order of 100s of switches and links
- Scenarios
  - Cold start (freq. weeks)
  - Policy change (freq. days)
  - Topology/TM change (freq. minutes)

Compiler Evaluation - Results



Summary

Summary

- SNAP is a **network-wide programmable** platform for **stateful** packet processing

Summary

- SNAP is a **network-wide programmable** platform for **stateful** packet processing
- SNAP Language
  - One big stateful switch abstraction
  - Intuitive and flexible composition

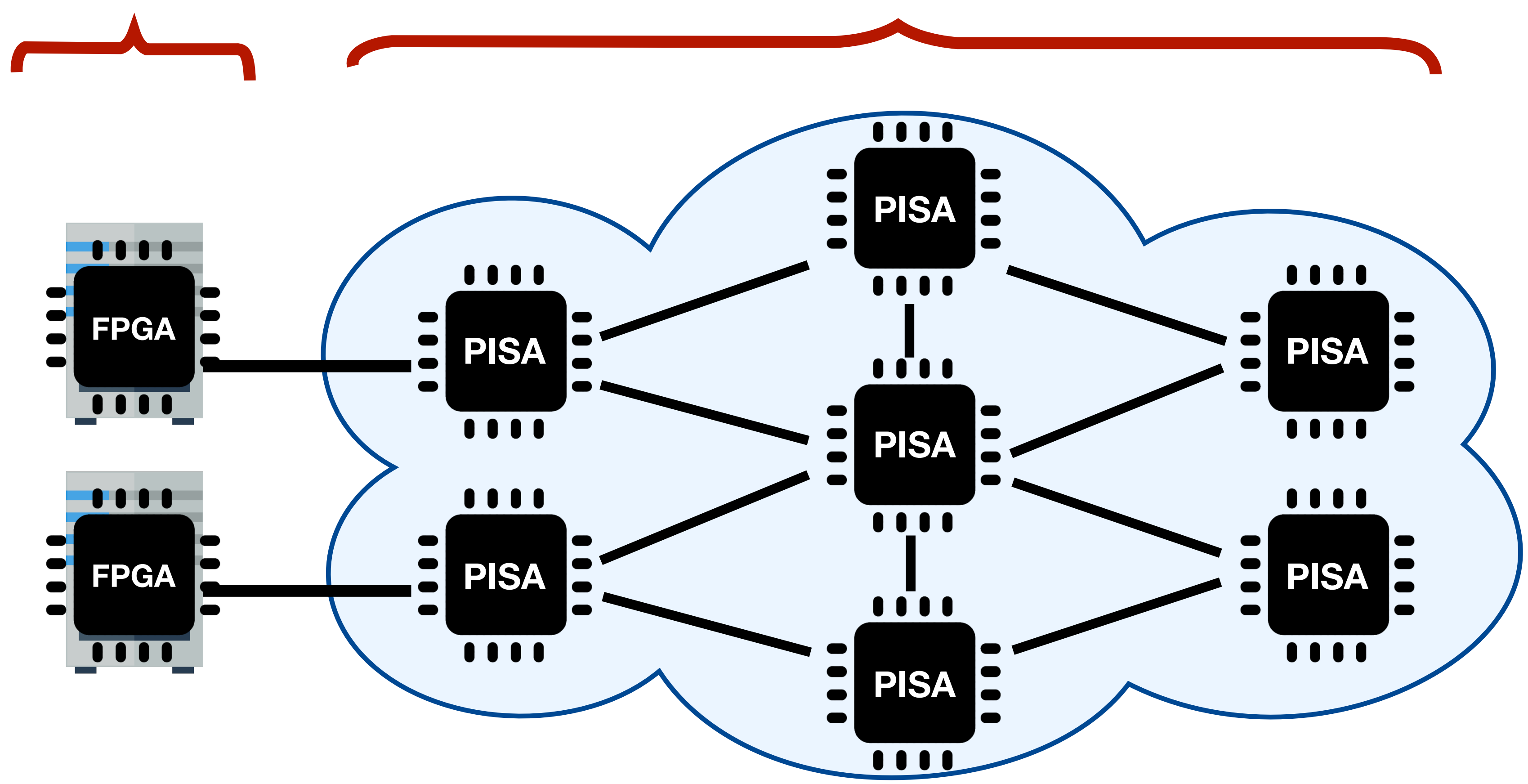
Summary

- SNAP is a **network-wide programmable** platform for **stateful** packet processing
- SNAP Language
  - One big stateful switch abstraction
  - Intuitive and flexible composition
- SNAP Compiler
  - Decides state placement and routing
  - Distributes an intermediate representation of the program across the network

Stateful Programming of High-Speed Network Hardware

Tonic
[Under revision for NSDI'19]

SNAP
[SIGCOMM'16]



Thank You!