A Compiler and Run-time System for Network Programming Languages

Christopher Monsanto, Princeton

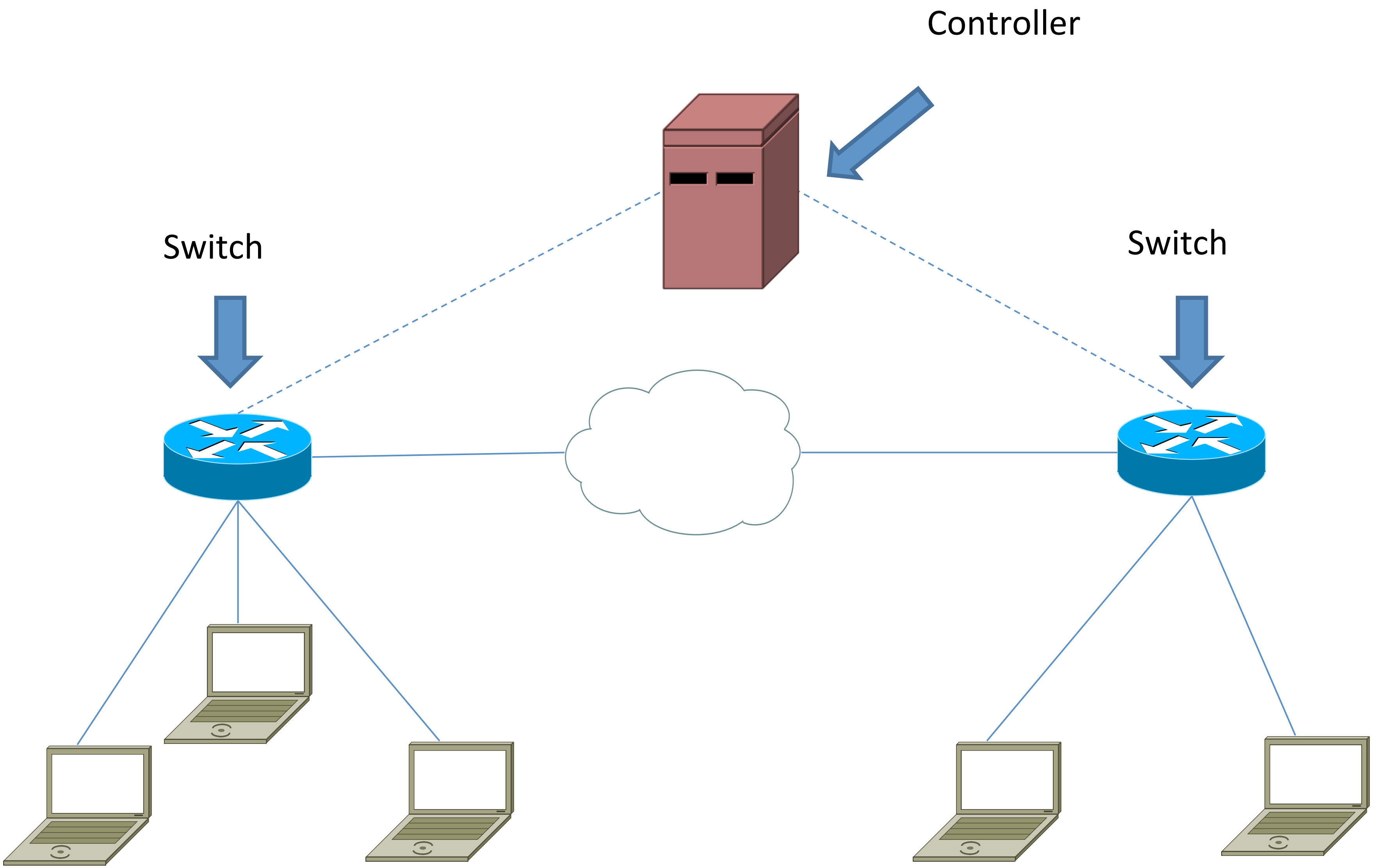
Nate Foster, Cornell

Rob Harrison, West Point

David Walker, Princeton

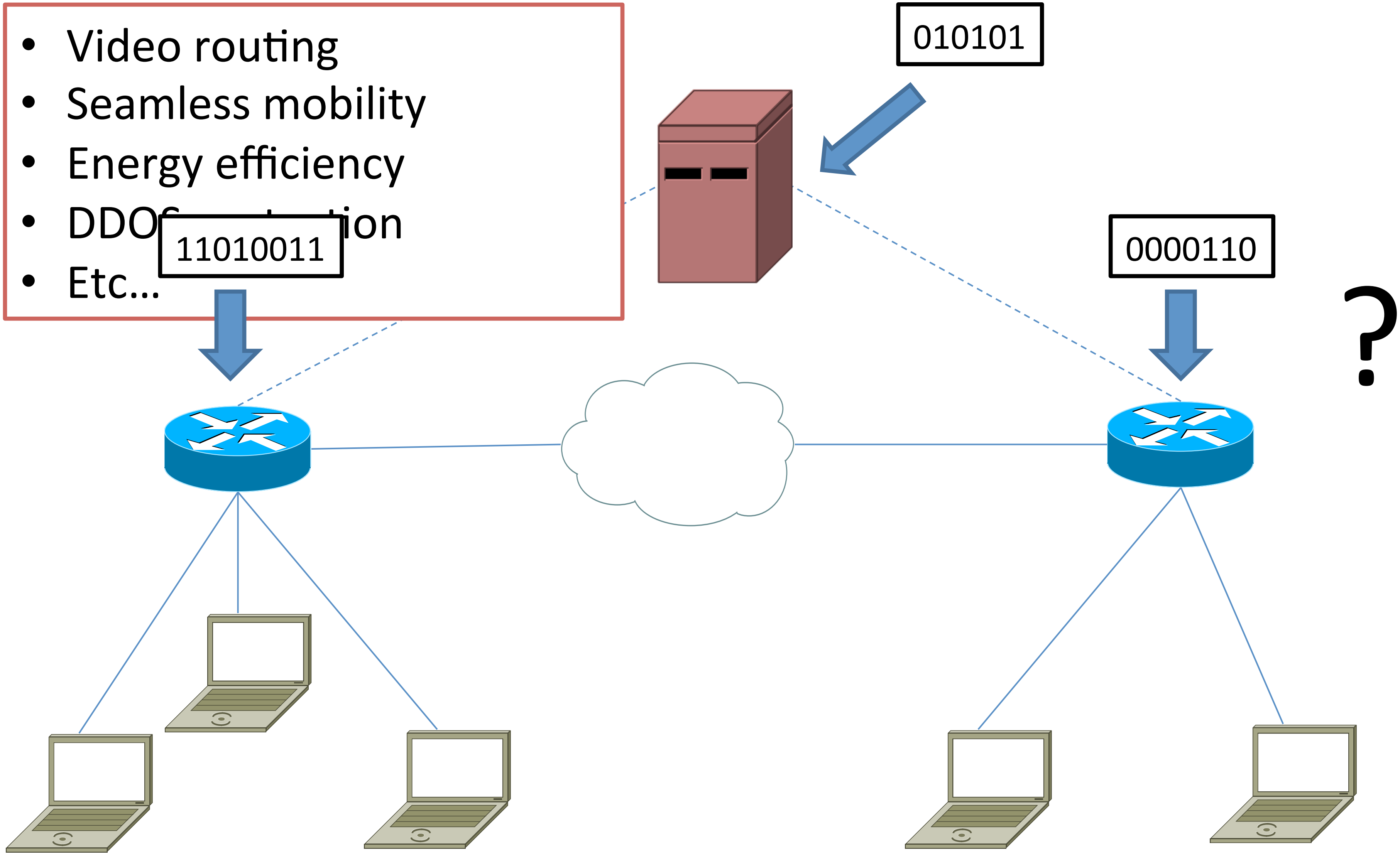
frenetic >>

Software-Defined Networks



Software-Defined Networks

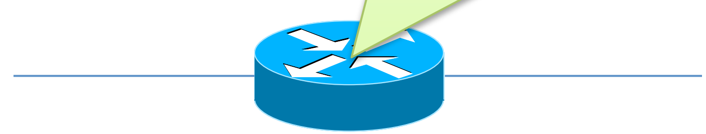
- Video routing
- Seamless mobility
- Energy efficiency
- DDOS detection
- Etc…



Software Defined Networks: Switches

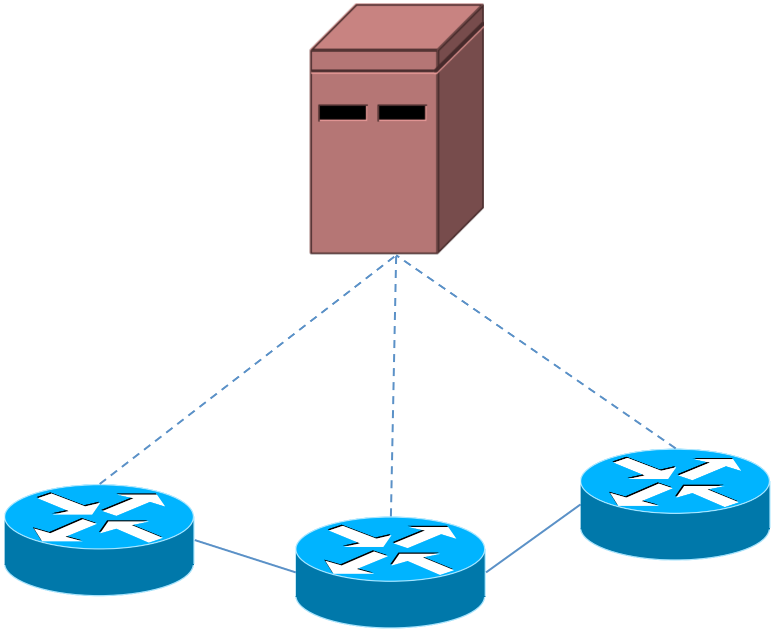
- Processes packets according to classifier
  - Limited pattern matching capabilities
  - Actions: forwarding, dropping, sending to controller
- As fast as it gets

Dest. IP	Dest. Port	Action
191.*.*.*	80	Forward A
191.*.*.*	*	Drop
191.1.1.0	22	Forward B
*	*	Controller



SDN: Controllers

- Capable of arbitrary computation
- Orders of magnitude slower
- Installs and uninstalls rules from switch classifiers



Goals

1. A simple, declarative language…
   – That is, we specify the functionality, not the concrete rules that go on the switches, or the explicit install and uninstall commands the controller must issue
2. … that is mathematically guaranteed to be correct and efficient.

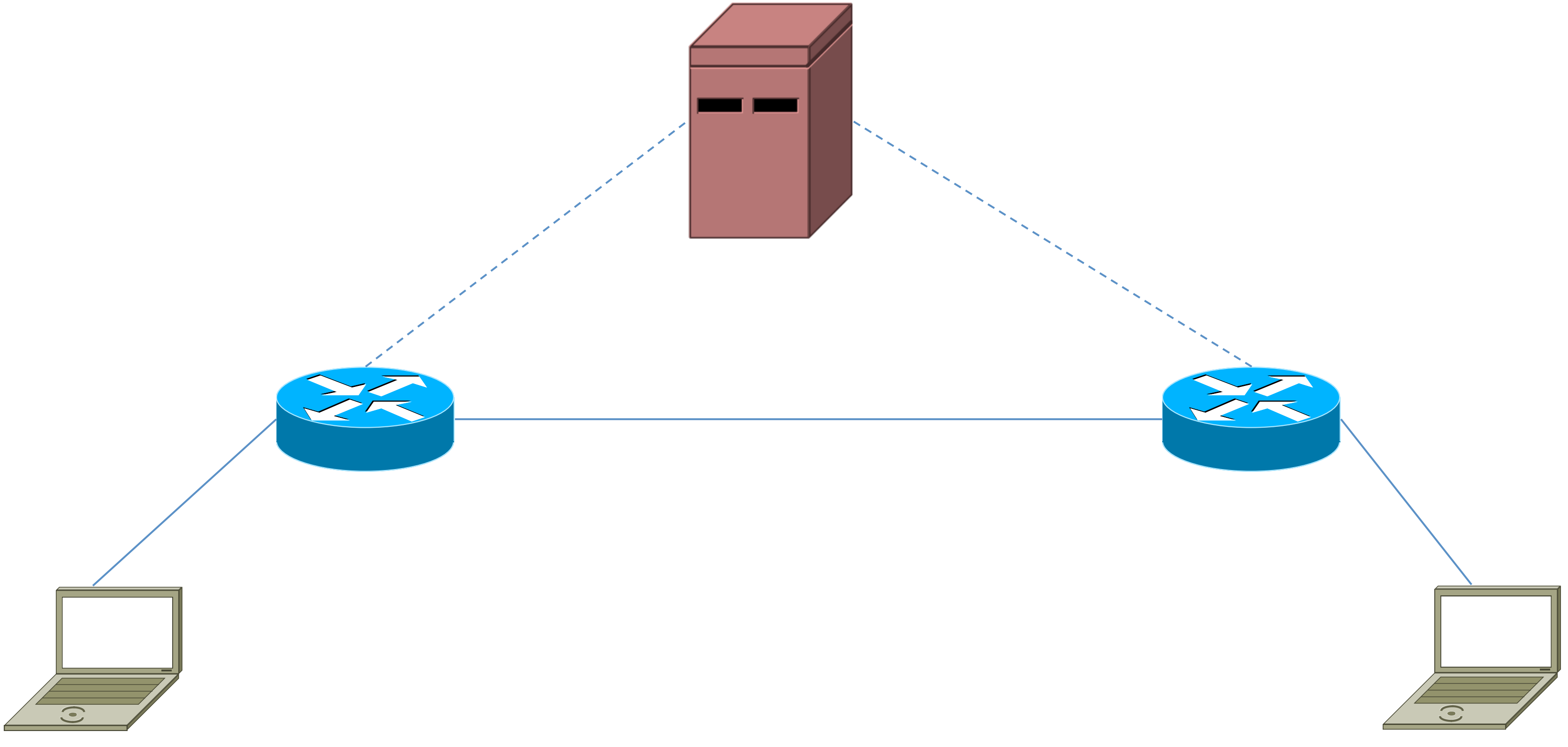
Goals

1. A simple, declarative language…
   – That is, we specify the functionality, not the concrete rules that go on the switches, or the explicit install and uninstall commands the controller must issue
2. … that is mathematically guaranteed to be correct and *efficient*.

???

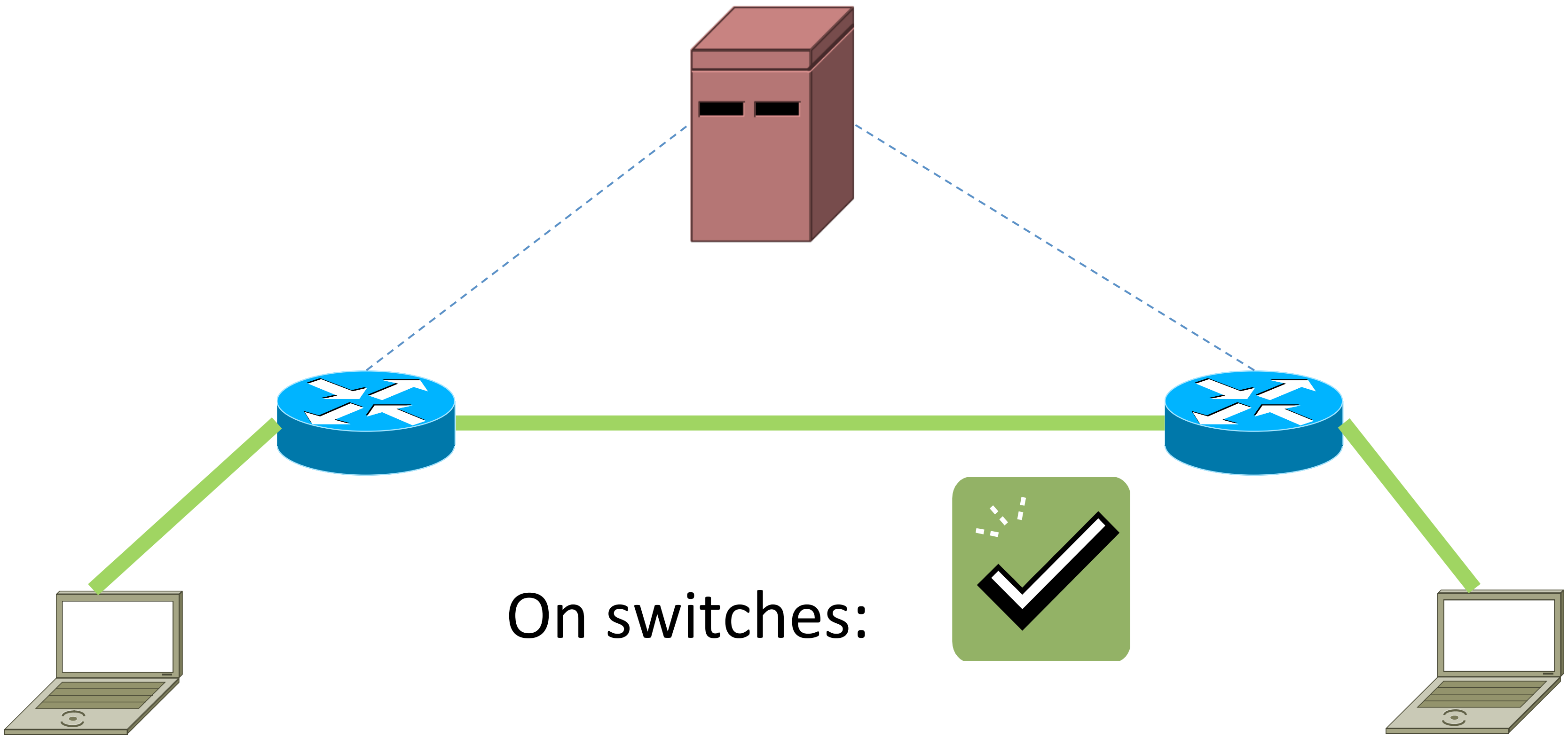
Goals

- … guaranteed to be *efficient*



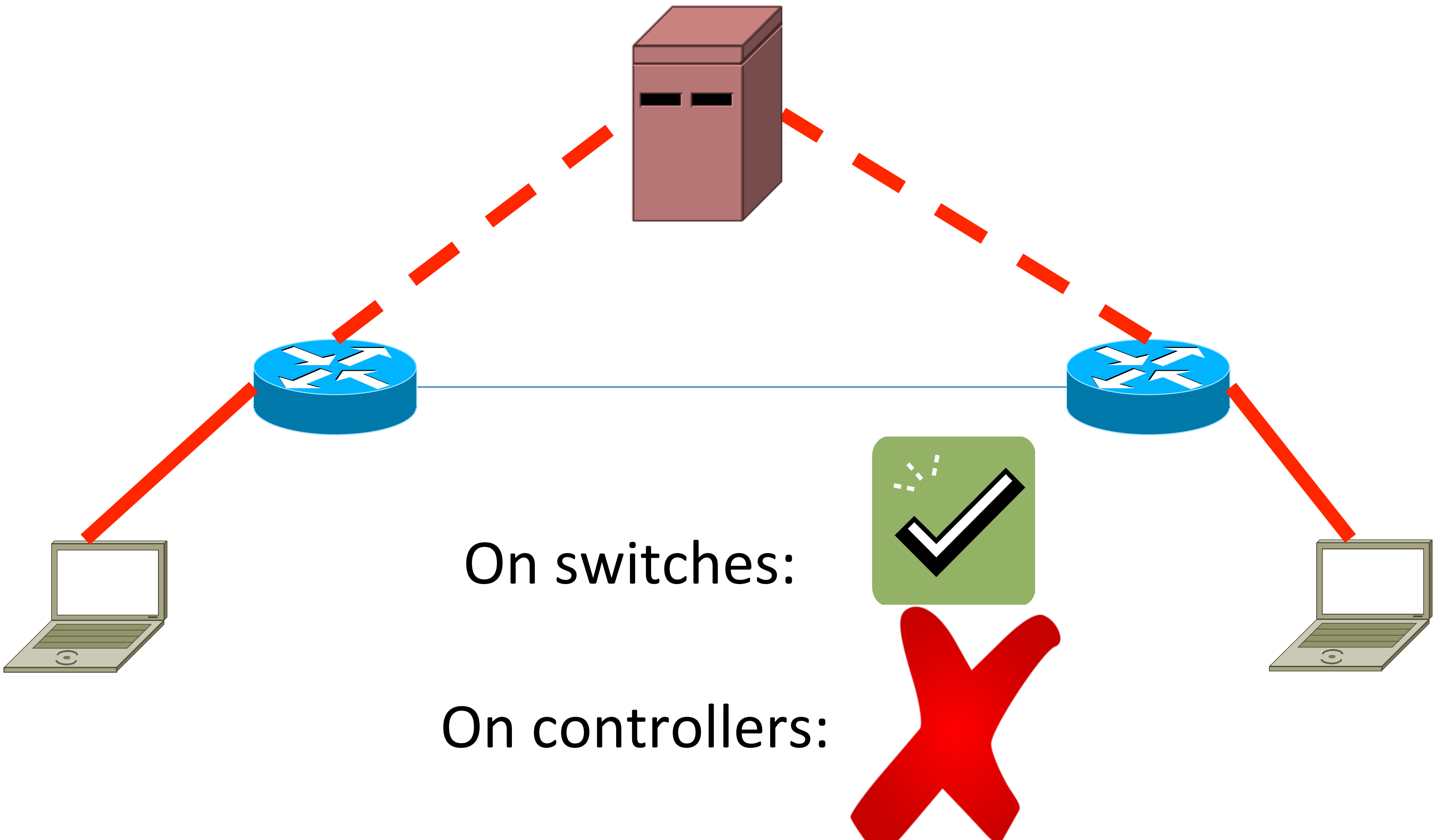
Goals

- … guaranteed to be *efficient*



Goals

- … guaranteed to be *efficient*

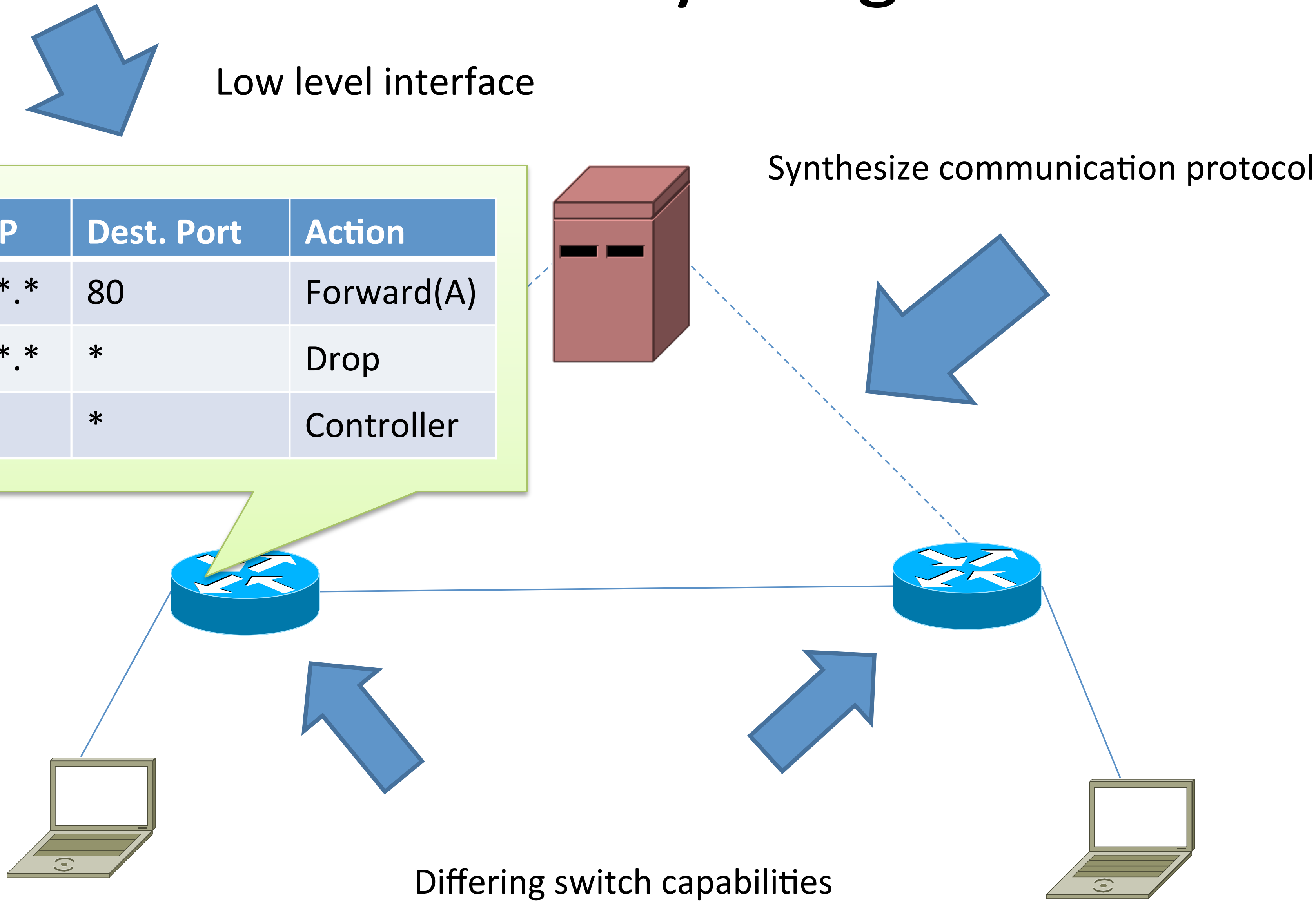


On switches:

On controllers:

Obstacles Faced By Programmers

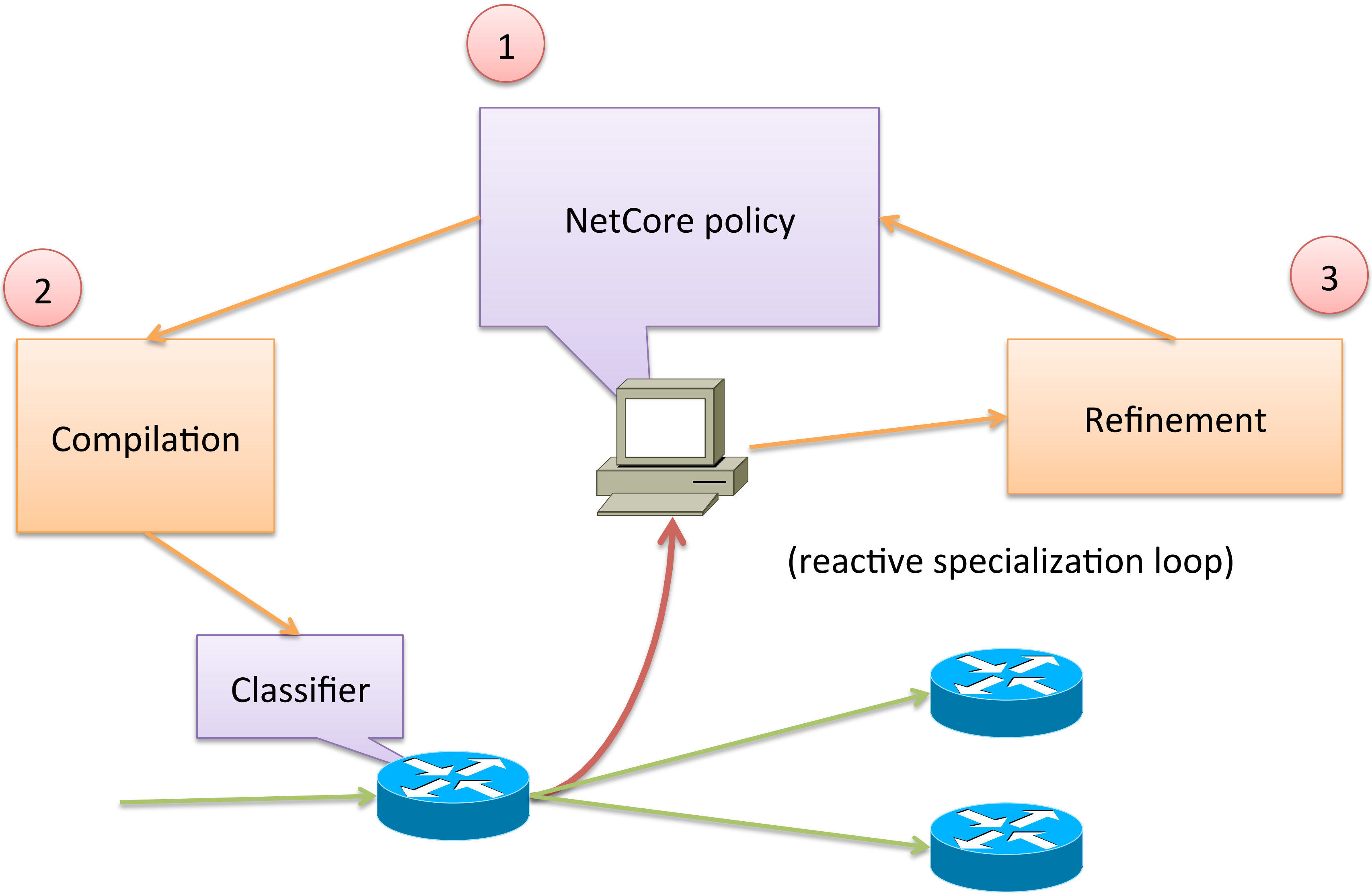
Dest. IP	Dest. Port	Action
191.*.*.*	80	Forward(A)
191.*.*.*	*	Drop
*	*	Controller

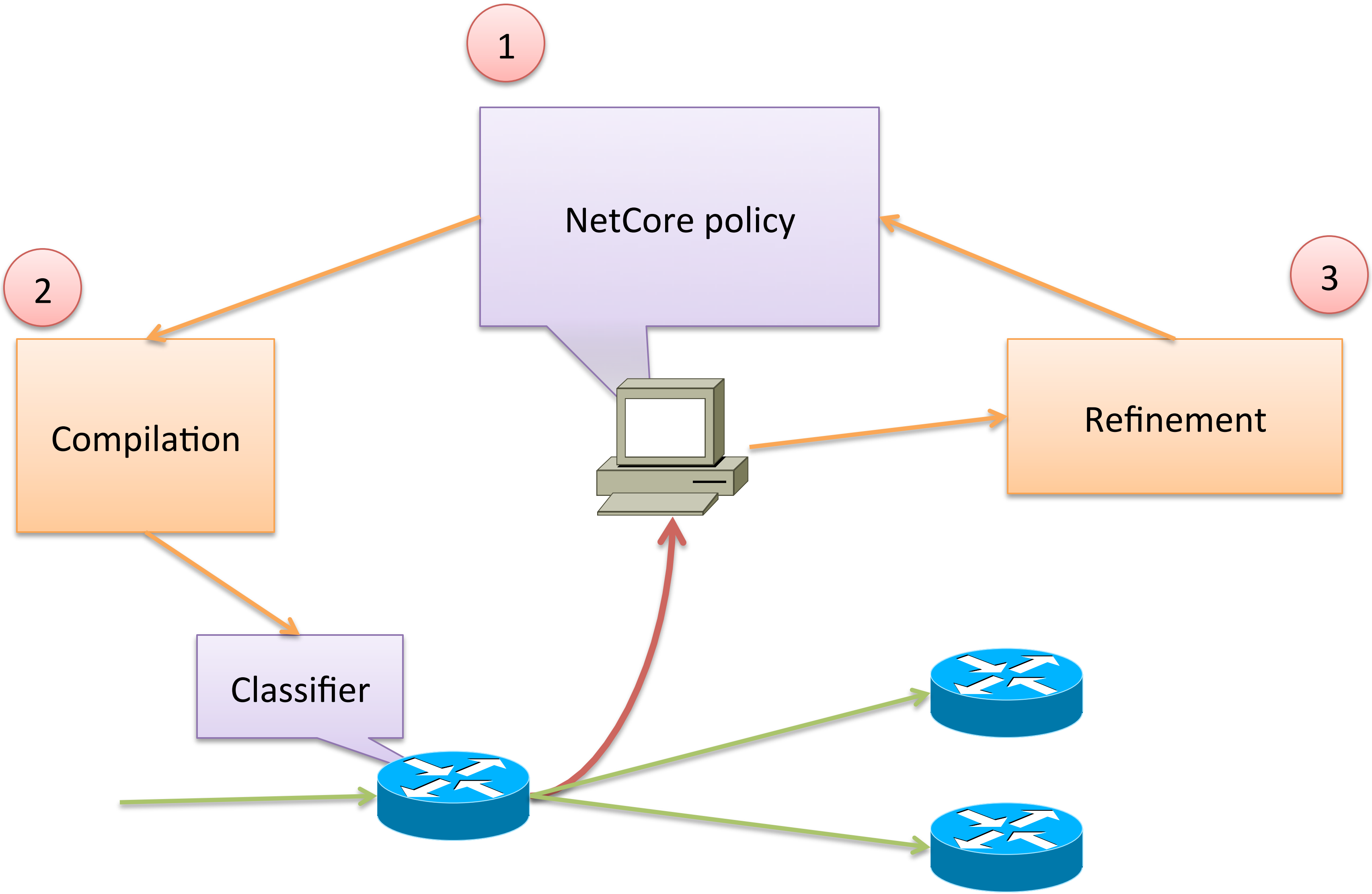


Our Real Enemy: Complexity

- Managing these rules is complicated
- Tendency to fall back to using the simplest kind of rule: microflows (exact match rules)
- Very inefficient: many packets go to the controller!

New Frenetic Run-time Architecture





NETCORE

14

NetCore Grammar

Predicates | **Headers h, wildcards w**

- e ::= $h : w$ — primitive match
  - | switch s — match packets at switch
  - | inspect f — arbitrary function
  - | $e_1 \cup e_2$ — set union
  - | $e_1 \setminus e_2$ — set difference
  - | $e_1 \cap e_2$ — set intersection
  - | $\neg e$ — set negation

Policies

- τ ::= $e \to \{s_1, \ldots, s_n\}$ — action
  - | $\tau_1 \cup \tau_2$ — policy union
  - |

NetCore Grammar

Predicates — **Headers h, wildcards w**

- e ::= $h : w$ — primitive match
  - | `switch` s — match packets at switch
  - | `inspect` f — arbitrary function
  - | $e_1 \cup e_2$ — set union
  - | $e_1 \setminus e_2$ — set difference
  - | $e_1 \cap e_2$ — set intersection
  - | $\neg e$ — set negation

Policies

- τ ::= $e \rightarrow \{s_1, \ldots, s_n\}$ — action
  - | $\tau_1 \cup \tau_2$ — policy union
  - |

NetCore Grammar

Predicates | **Headers h, wildcards w**

• e ::=	$h : w$	primitive match
|	switch s	
	`DestIP: 10.*.1.*`	
|	inspect f	arbitrary function
|	$e_1 \cup e_2$	set union
|	$e_1 \setminus e_2$	set difference
|	$e_1 \cap e_2$	set intersection
|	$\neg e$	set negation

Policies

• τ ::=	$e \rightarrow \{s_1, \ldots, s_n\}$	action
|	$\tau_1 \cup \tau_2$	policy union
|		

NetCore Grammar

Predicates **Headers h, wildcards w**

- e ::= h:w primitive match
  | switch s match packets at switch
  | inspect f arbitrary function
  | $e_1 \cup e_2$ set union
  | $e_1 \setminus e_2$ set difference
  | $e_1 \cap e_2$ set intersection
  | $\neg e$ set negation

Policies

- τ ::= $e \rightarrow \{s_1, \ldots, s_n\}$ action
  | $\tau_1 \cup \tau_2$ policy union
  |

NetCore Grammar

Predicates — **Headers h, wildcards w**

• e ::= $h:w$	primitive match
| `switch` s	match packets at switch
| `inspect` f	arbitrary function
| $e_1 \cup e_2$	set union
| $e_1 \setminus e_2$	set difference
| $e_1 \cap e_2$	set intersection
| $\neg e$	set negation

Policies

• τ ::= $e \rightarrow \{s_1, \ldots, s_n\}$	action
| $\tau_1 \cup \tau_2$	policy union

NetCore Grammar

Predicates — Headers h, wildcards w

- e ::= $h:w$ — primitive match
  - | switch s — match packets at switch
  - | inspect f — arbitrary function

```
badWebsite p = addr `elem` blacklist
 where
  headers = parseHTTP p
  addr = headers ! “address”
```

Policies

- τ ::= $e \rightarrow \{s_1, \ldots, s_n\}$ — action
  - | $\tau_1 \cup \tau_2$ — policy union

NetCore Grammar

Predicates — Headers h, wildcards w

- e ::= $h:w$ — primitive match
  - | switch s — match packets at switch
  - | inspect f — arbitrary function

```
badWebsite p = addr `elem` blacklist
 where
  headers = parseHTTP p
  addr = headers ! “address”
```

| ¬e — set negation

Policies

```
inspect (not ∘ badWebsite)
```

- τ ::= $e \rightarrow \{s_1, \ldots, s_n\}$ — action
  - | $\tau_1 \cup \tau_2$ — policy union

NetCore Grammar

Predicates — **Headers h, wildcards w**

- e ::= $h : w$ — primitive match
  - | `switch` s — match packets at switch
  - | `inspect` f — arbitrary function
  - | $e_1 \cup e_2$ — set union
  - | $e_1 \setminus e_2$ — set difference
  - | $e_1 \cap e_2$ — set intersection
  - | $\neg e$ — set negation

Policies

- τ ::= $e \to \{s_1, \ldots, s_n\}$ — action
  - | $\tau_1 \cup \tau_2$ — policy union
  - |

NetCore Grammar

Predicates — Headers h, wildcards w

- e ::= $h : w$ — primitive match
  - | switch s — match packets at switch
  - | inspect f — arbitrary function
  - | $e_1 \cup e_2$ — set union
  - | $e_1 \setminus e_2$ — set difference
  - | $e_1 \cap e_2$ — set intersection

```
inspect (not ∘ badWebSite) ∪
DestIP: 10.10.50.20
```

Policies

- τ ::= $e \rightarrow \{s_1, \ldots, s_n\}$ — action
  - | $\tau_1 \cup \tau_2$ — policy union

NetCore Grammar

Predicates **Headers h, wildcards w**

- e ::= $h : w$ primitive match
  | switch s match packets at switch
  | inspect f arbitrary function
  | $e_1 \cup e_2$ set union
  | $e_1 \setminus e_2$ set difference
  | $e_1 \cap e_2$ set intersection
  | $\neg e$ set negation

Policies

- τ ::= $e \rightarrow \{s_1, \ldots, s_n\}$ action
  | $\tau_1 \cup \tau_2$ policy union
  |

NetCore Grammar

Predicates | **Headers h, wildcards w**

- e ::= $h : w$ — primitive match
  - | switch s — match packets at switch
  - | inspect f — arbitrary function
  - | $e_1 \cup e_2$ — set union
  - | $e_1 \setminus e_2$ — set difference
  - | $e_1 \cap e_2$ — set intersection
  - | $\neg e$ — set negation

Policies

- τ ::

`inspect (not ∘ badWebSite) → {A}`

  - | $\tau_1 \cup \tau_2$ — policy union

NetCore Grammar

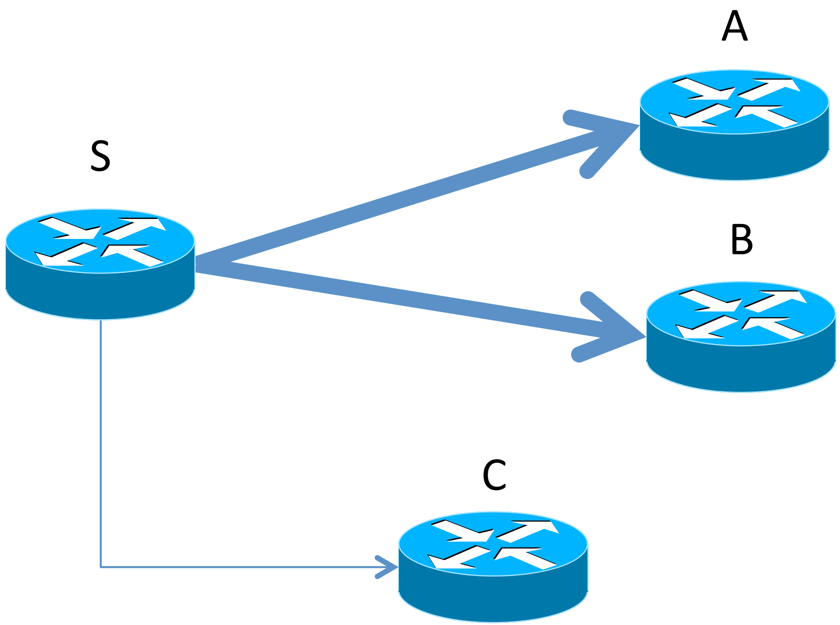
Predicates | **Headers h, wildcards w**

- e ::= $h : w$ — primitive match
  | `switch` s — match packets at switch
  | `inspect` f — arbitrary function
  | $e_1 \cup e_2$ — set union
  | $e_1 \setminus e_2$ — set difference
  | $e_1 \cap e_2$ — set intersection
  | $\neg e$ — set negation

Policies

- τ ::= $e \rightarrow \{s_1, \ldots, s_n\}$ — action
  | $\tau_1 \cup \tau_2$ — policy union
  |

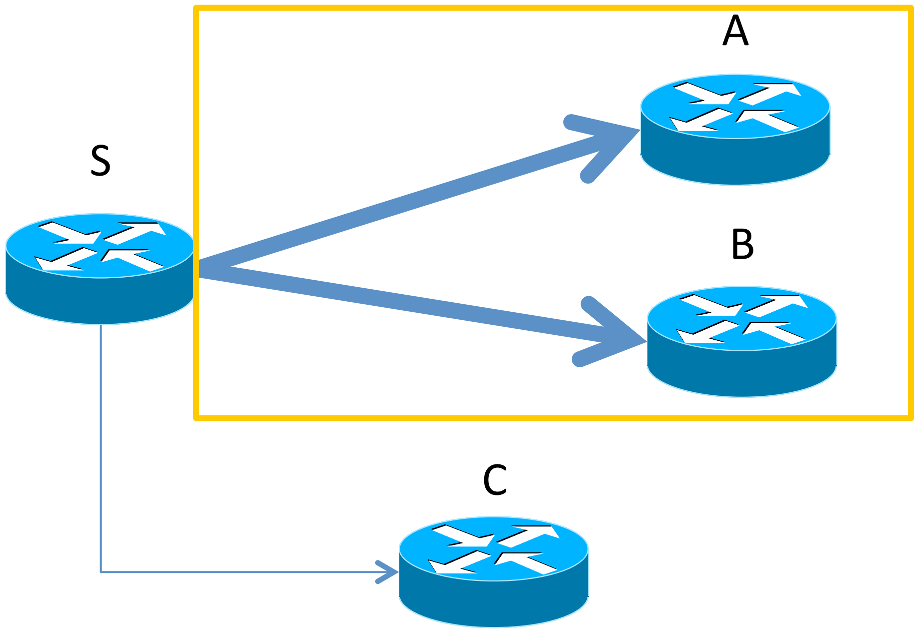
NetCore Example



- Load balanced, fast path
- Slow path

```
DestIP: *0 → {A} ∪ DestIP: *1 → {B}
```

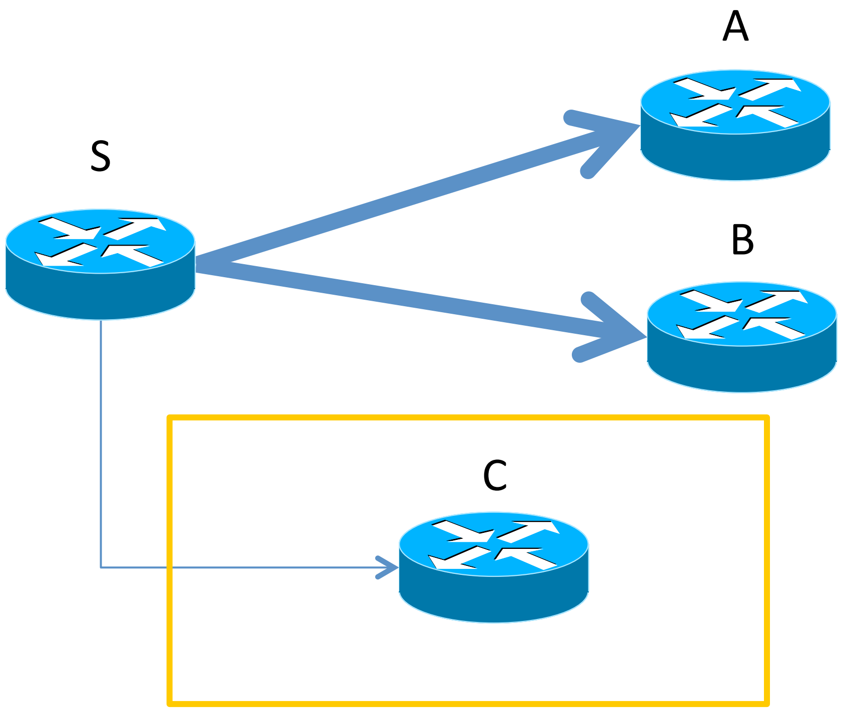
NetCore Example



- Load balanced, fast path
  - Internal traffic (90.80.*.*)
  - SSH traffic
- Slow path

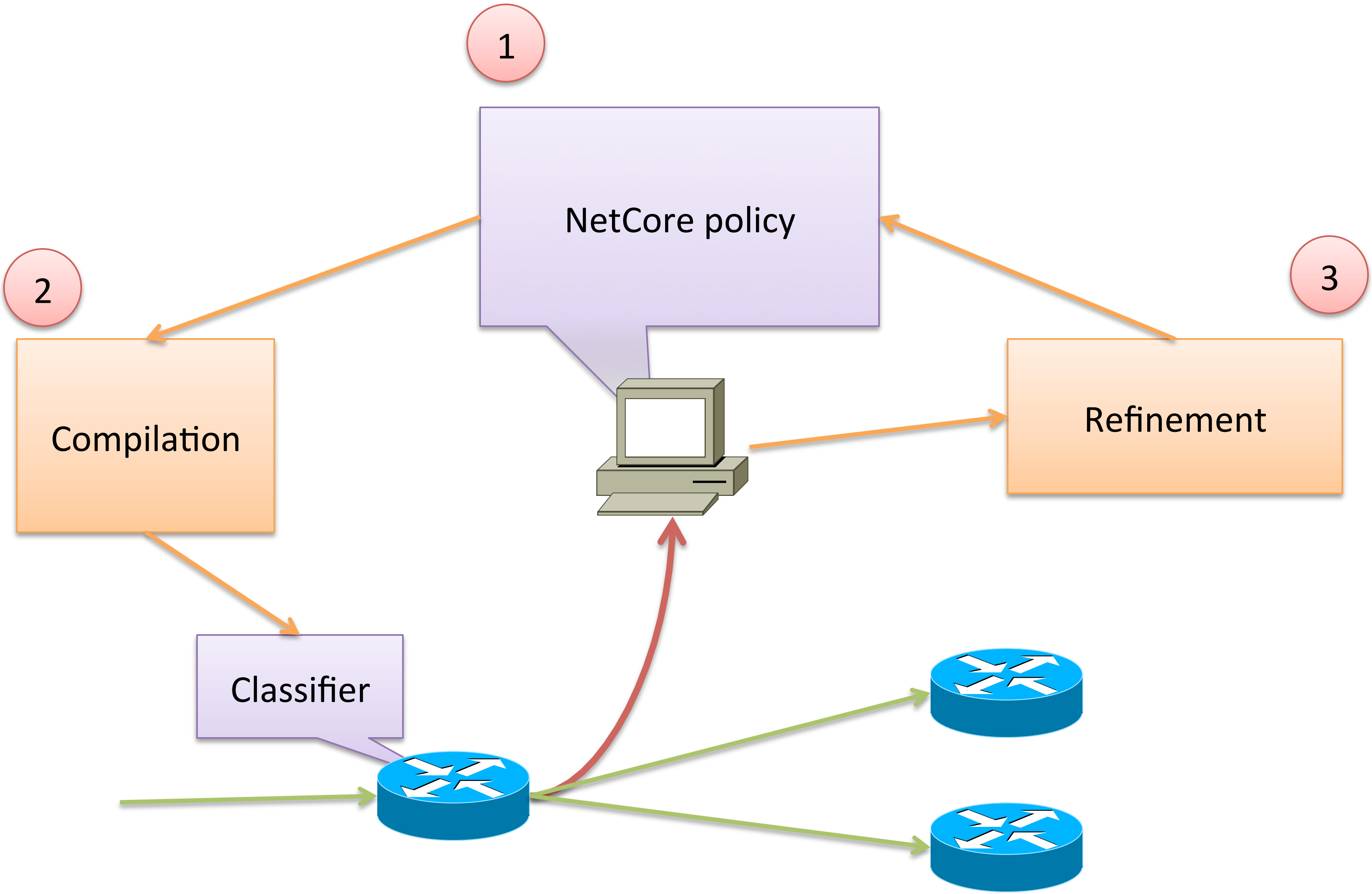
```
(DestIP:90.80.*.* ∪ DestPort: 22) ∩
  (DestIP: *0 → {A} ∪ DestIP: *1 → {B})
```

NetCore Example



- Load balanced, fast path
  - Internal traffic (90.80.*.*)
  - SSH traffic
- Slow path
  - External traffic

```
((DestIP:90.80.*.* ∪ DestPort: 22) ∩
  (DestIP: *0 → {A} ∪ DestIP: *1 → {B})) ∪
(¬DestIP:90.80.*.*) → {C}
```



COMPILATION

30

Compilation

```
P1 = DestIP:90.80.*.* → {A}
```

Compiles to:

Dest. IP	Dest. Port	Action
90.80.*.*	*	Forward A

Compilation: Approximations

Non prefix match!

```
P2 = DestIP:90.*.70.60 → {B}
```

Compiles to:

Dest. IP	Dest. Port	Action
90.*.*.*	*	Controller

Overapproximation

Compilation

Run programs in parallel:

```
P3 = P1 ∪ P2
```

Compiles to:

Dest. IP	Dest. Port	Action
90.80.70.60	*	Forward A, B
90.80.*.*	*	Forward A
90.*.*.*	*	Controller

Compilation

Run programs in parallel:

```
P3 = P1 ∪ P2
```

Subtle interaction: where did B come from?

Compiles to:

Dest. IP	Dest. Port	Action
90.80.70.60	*	Forward A, B
90.80.*.*	*	Forward A
90.*.*.*	*	Controller

Compilation: Exponential Growth

DestIP: 1.*.*.* $\rightarrow$ {A}

Dest. IP	Dest. Port	Action
1.*.*.*	*	Forward A

Compilation: Exponential Growth

DestIP: 1.*.*.* $\rightarrow$ {A} $\cup$
DestIP: *.2.*.* $\rightarrow$ {B}

Dest. IP	Dest. Port	Action
1.2.*.*	*	Forward A, B
.2..*	*	Forward B
1.*.*.*	*	Forward A

Compilation: Exponential Growth

DestIP: 1.*.*.* $\rightarrow$ {A} $\cup$
DestIP: *.2.*.* $\rightarrow$ {B} $\cup$
DestIP: *.*.3.* $\rightarrow$ {C}

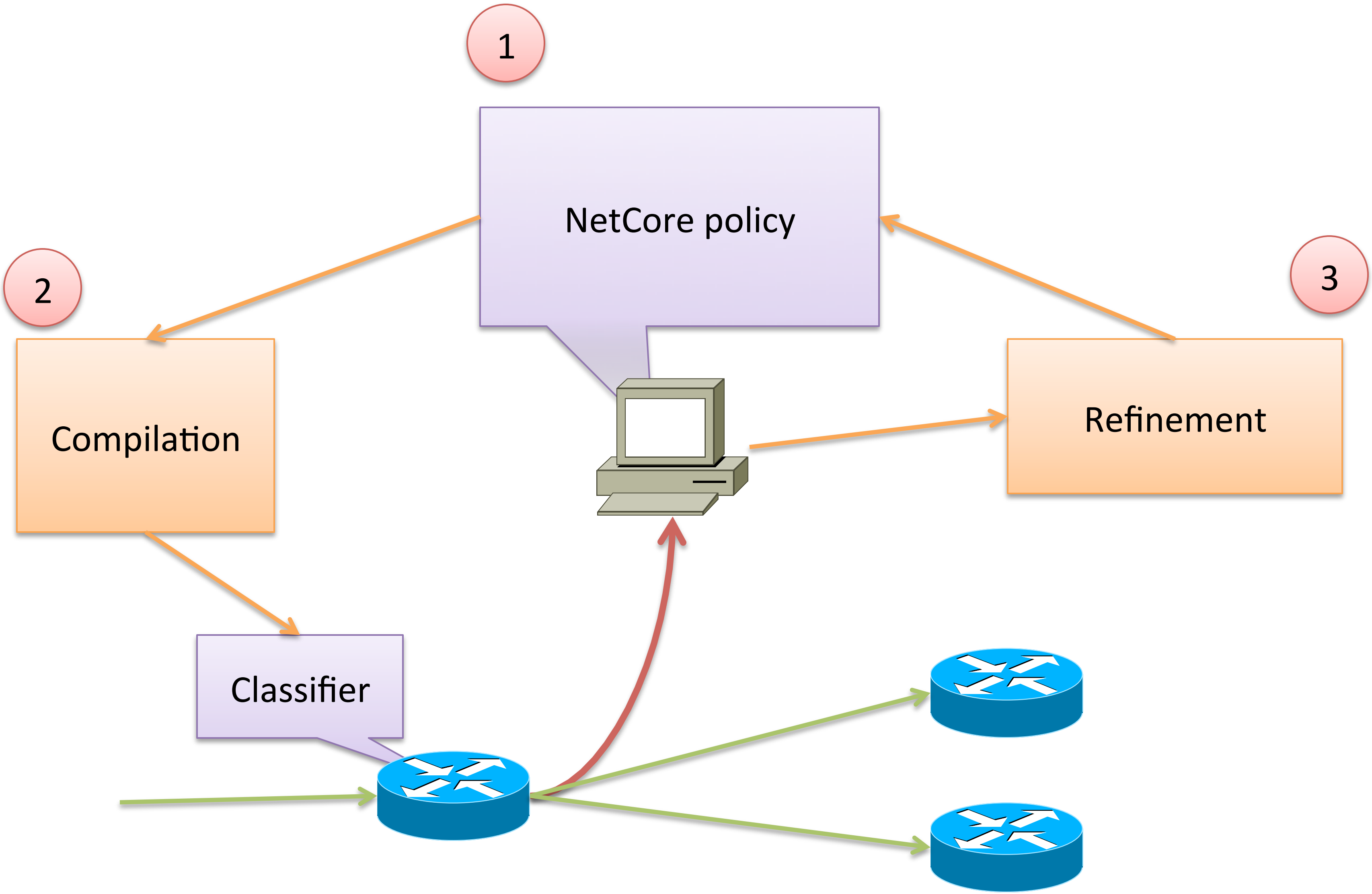
Dest. IP	Dest. Port	Action
1.2.3.*	*	Forward A, B, C
.2.3.	*	Forward B, C
1.*.3.*	*	Forward A, C
..3.*	*	Forward C
1.2.*.*	*	Forward A, B
.2..*	*	Forward B
1.*.*.*	*	Forward A

Com… …th

Dest. IP	Dest. Port	Action
1.2.3.4	*	Forward A, B, C, D
*.2.3.4	*	Forward B, C, D
1.*.3.4	*	Forward A, C, D
..3.4	*	Forward C, D
1.2.*.4	*	Forward A, B, D
.2..4	*	Forward B, D
1.*.*.4	*	Forward A, D
..*.4	*	Forward D
1.2.3.*	*	Forward A, B, C
.2.3.	*	Forward B, C
1.*.3.*	*	Forward A, C
..3.*	*	Forward C
1.2.*.*	*	Forward A, B
.2..*	*	Forward B
1.*.*.*	*	Forward A

Compilation: Takeaways

- Switch classifiers are not compositional – they do not contain enough information
  - Compiler uses an intermediate form which makes use of high-level information
- Classifiers that implement high-level specifications grow in size quickly
  - Compiler uses optimized algorithms and classifier minimization techniques



REACTIVE SPECIALIZATION

40

Reactive Specialization

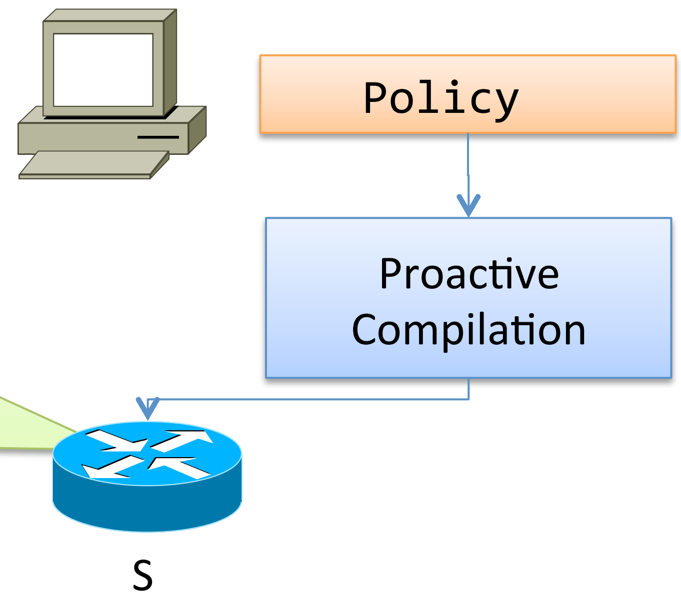
```
DestIP: 90.80.*.* → {A} ∪
DestIP: 90.*.70.60 → {B}
```

- Can't compile this classifier exactly without using many rules
  - Most switches only support prefix matches
  - Worst case can take billions and billions of rules

R. Specialization: Base Compilation

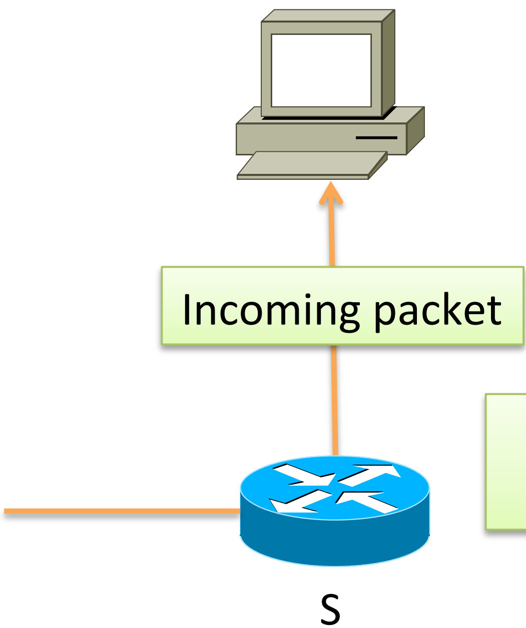
- Step 1: install a “base” classifier
  - Proactively handle packets on switch if possible
  - Send other packets to the controller

Dest. IP	Action
90.80.70.60	Forward A, B
90.80.*.*	Forward A
90.*.*.*	Controller



R. Specialization: Refinement

- Packet {DestIP=90.2.60.70, DestPort=80} comes in
- Use packet to generate semantically equivalent, but structurally different policy



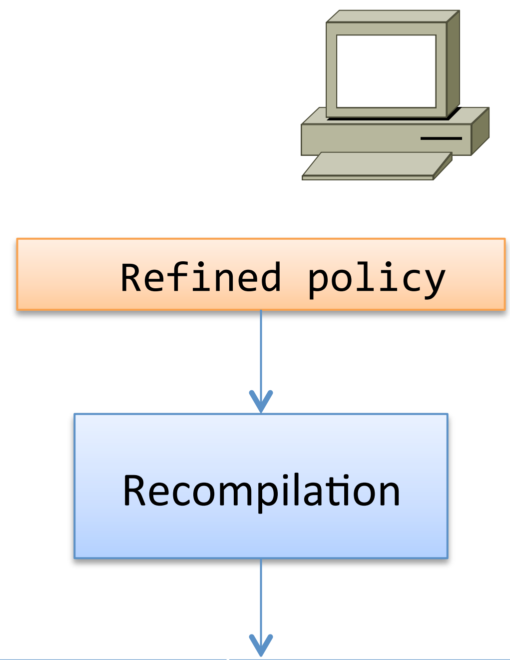

```
DestIP: 90.80.*.* → {A} ∪
DestIP: 90.*.70.60 → {B}
```

Refinement

```
DestIP: 90.80.*.* → {A} ∪
(DestIP: 90.*.70.60 ∪ DestIP: 90.2.60.70) → {B}
```

R. Specialization: Recompilation

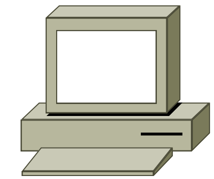
- Compile refined policy
  - Extra structural information handles similar packets to the one we handled

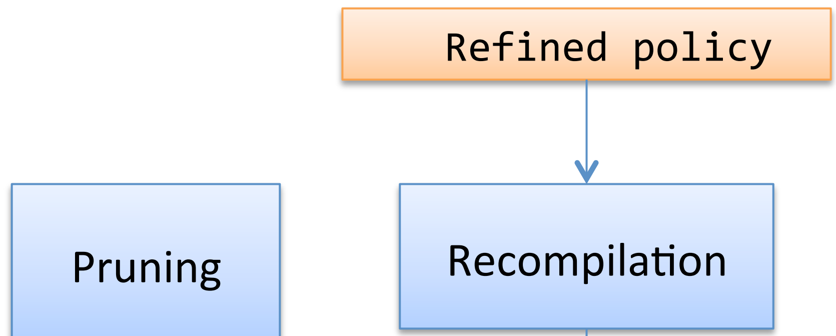


Dest. IP	Action
90.2.70.60	Forward B
90.80.70.60	Forward A, B
90.80.*.*	Forward A
90.*.*.*	Controller

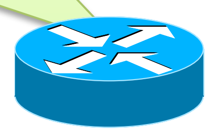
R. Specialization: Pruning

- Remove controller rules and rules that don’t match packet we handled
- Take result and place on switches





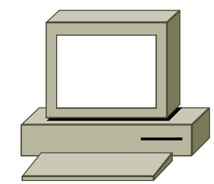
Dest. IP	Action
90.80.70.60	Forward A, B
90.80.*.*	Forward A
90.*.*.*	Controller



Dest. IP	Action
90.2.70.60	Forward B
90.80.70.60	Forward A, B
90.80.*.*	Forward A
90.*.*.*	Controller

R. Specialization: Pruning

- Remove controller rules and rules that don't match packet we handled
- Take result and place on switches

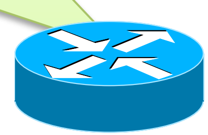


Refined policy

Recompilation

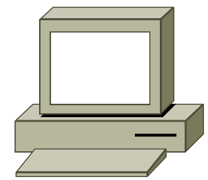
Dest. IP	Action
90.2.70.60	Forward B
90.80.70.60	Forward A, B
90.80.*.*	Forward A
90.*.*.*	Controller

Dest. IP	Action
90.2.70.60	Forward B
90.80.70.60	Forward A, B
90.80.*.*	Forward A
90.*.*.*	Controller



R. Specialization: Second Packet

- Consider handling a second packet, {DestIP=90.100.70.60, DestPort=21}



Refinement

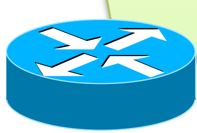
```
DestIP: 90.80.*.* → {A} ∪
(DestIP: 90.*.70.60 ∪ DestIP: 90.100.70.60) → {B}
```

Recompilation

Dest. IP	Action
90.100.70.60	Forward B
90.80.70.60	Forward A, B
90.80.*.*	Forward A
90.*.*.*	Controller

Pruning

Dest. IP	Action
90.100.70.60	Forward B
90.2.70.60	Forward B
90.80.70.60	Forward A, B
90.80.*.*	Forward A
90.*.*.*	Controller



THEORETICAL & EMPIRICAL RESULTS

Functional Correctness

- Define idealized machine that handles all traffic at the controller

Theorem (Functional Correctness):

NetCore machines bisimulate idealized machines.

- Statement of correctness more subtle when considering queries

Quiescence

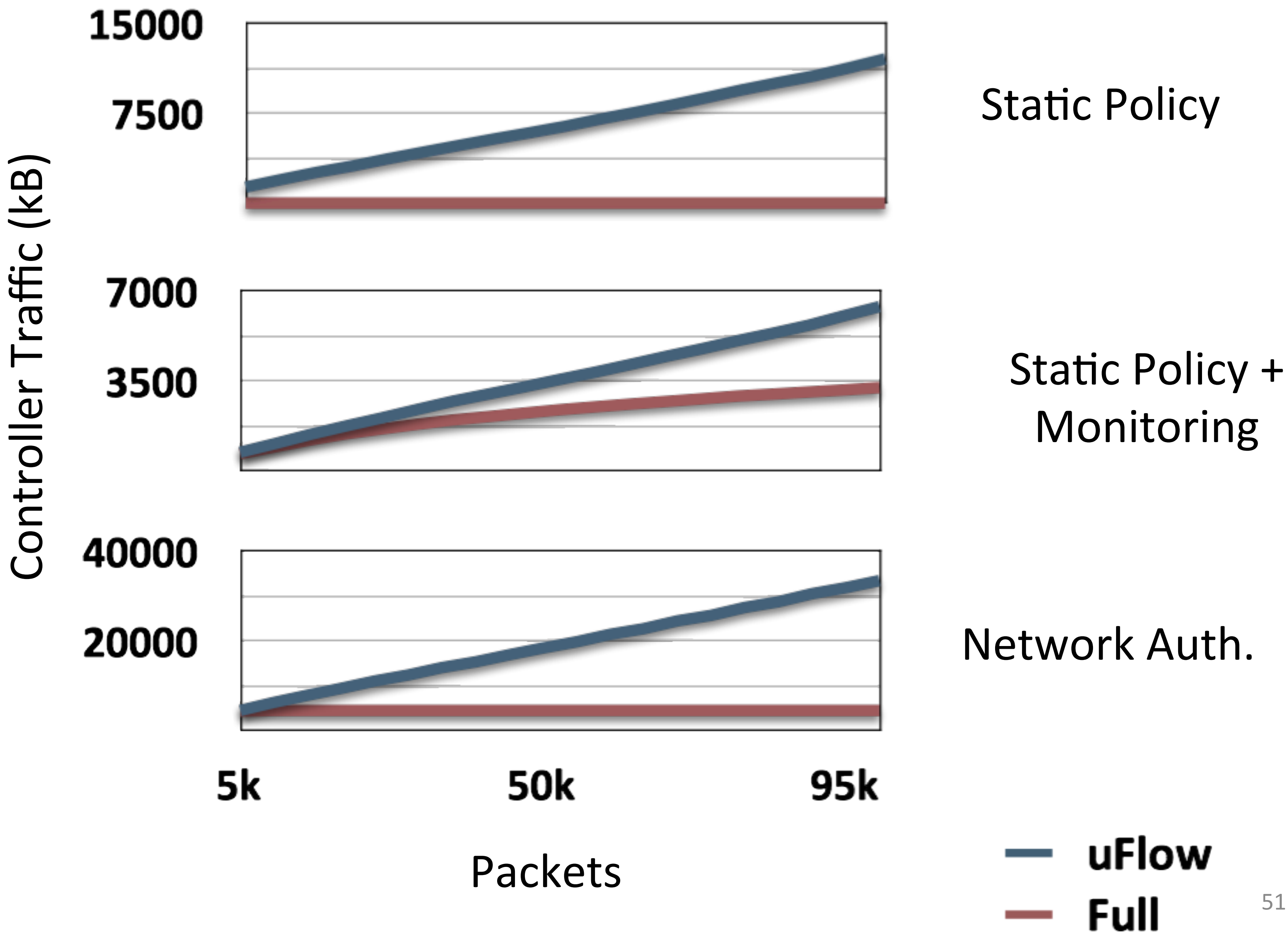
- Recall: Programs are efficient if they handle traffic on the switches, not the controller.

Theorem (Quiescence):

Under reasonable circumstances, NetCore programs eventually handle all traffic on the (fast) switches, instead of on the (slow) controller

- Special case of cost theorem: identify packets sent to controller

Benchmarks: Controller Traffic



Conclusion

- This talk was short; we didn't talk about:
  - Switch heterogeneity & lattice theory
  - Arbitrary Haskell functions inside policies
  - Queries
  - How compilation and specialization actually work
- So:
  - Check out the paper!
  - Talk to Chris, Dave, or Nate!
  - Check out the compiler at frenetic-lang.org!