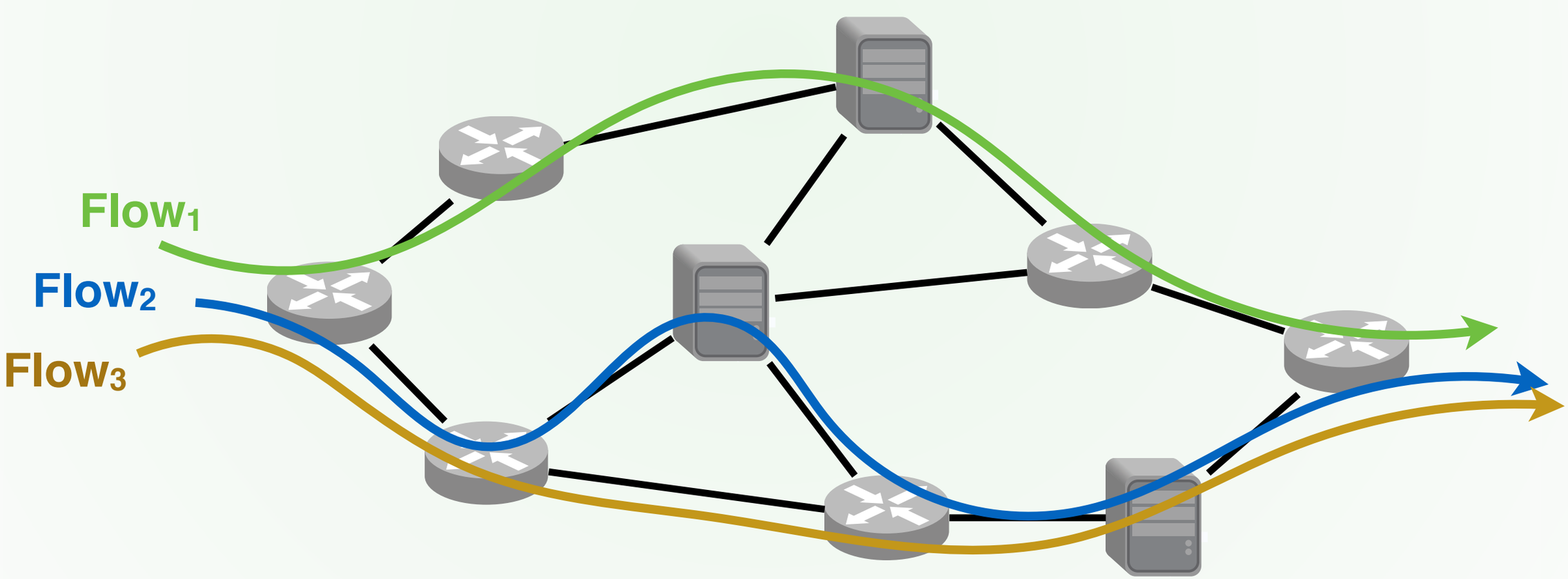
Concise Encoding of Flow Attributes in SDN Switches

Robert MacDavid*, Rüdiger Birkner†, Ori Rottenstreich*,
Arpit Gupta*, Nick Feamster*, Jennifer Rexford*

*Princeton University, †ETH Zürich

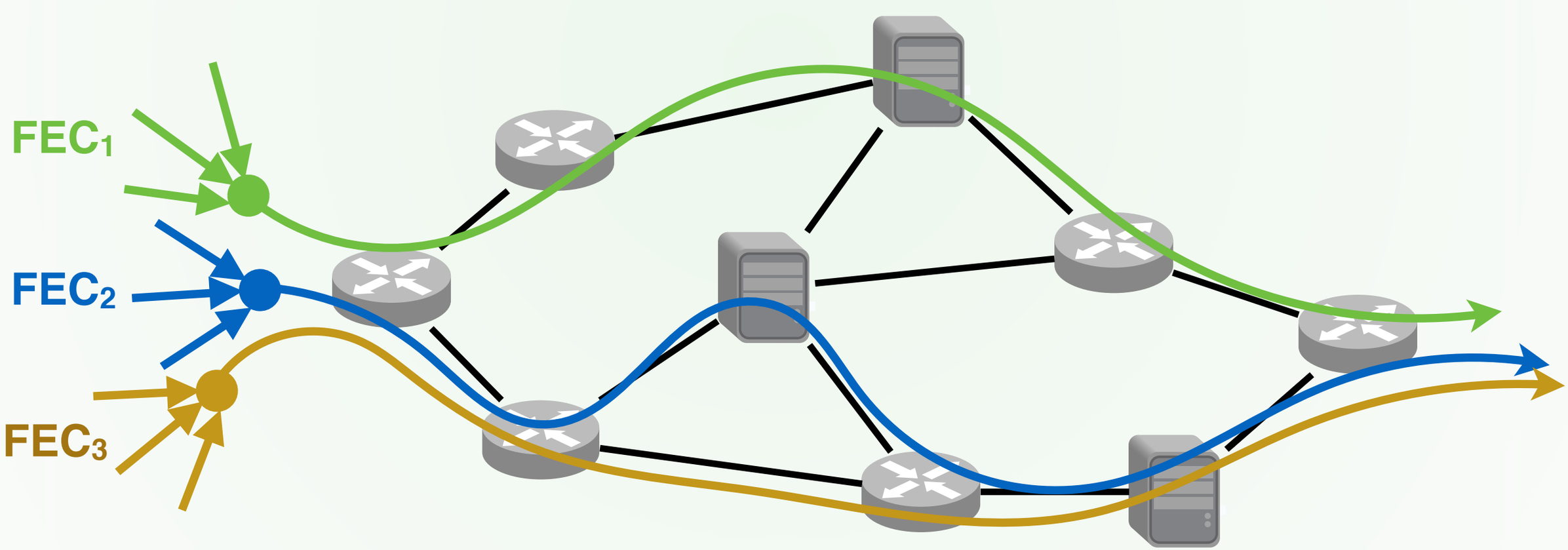
Motivation

Traffic forwarding is becoming increasingly complex



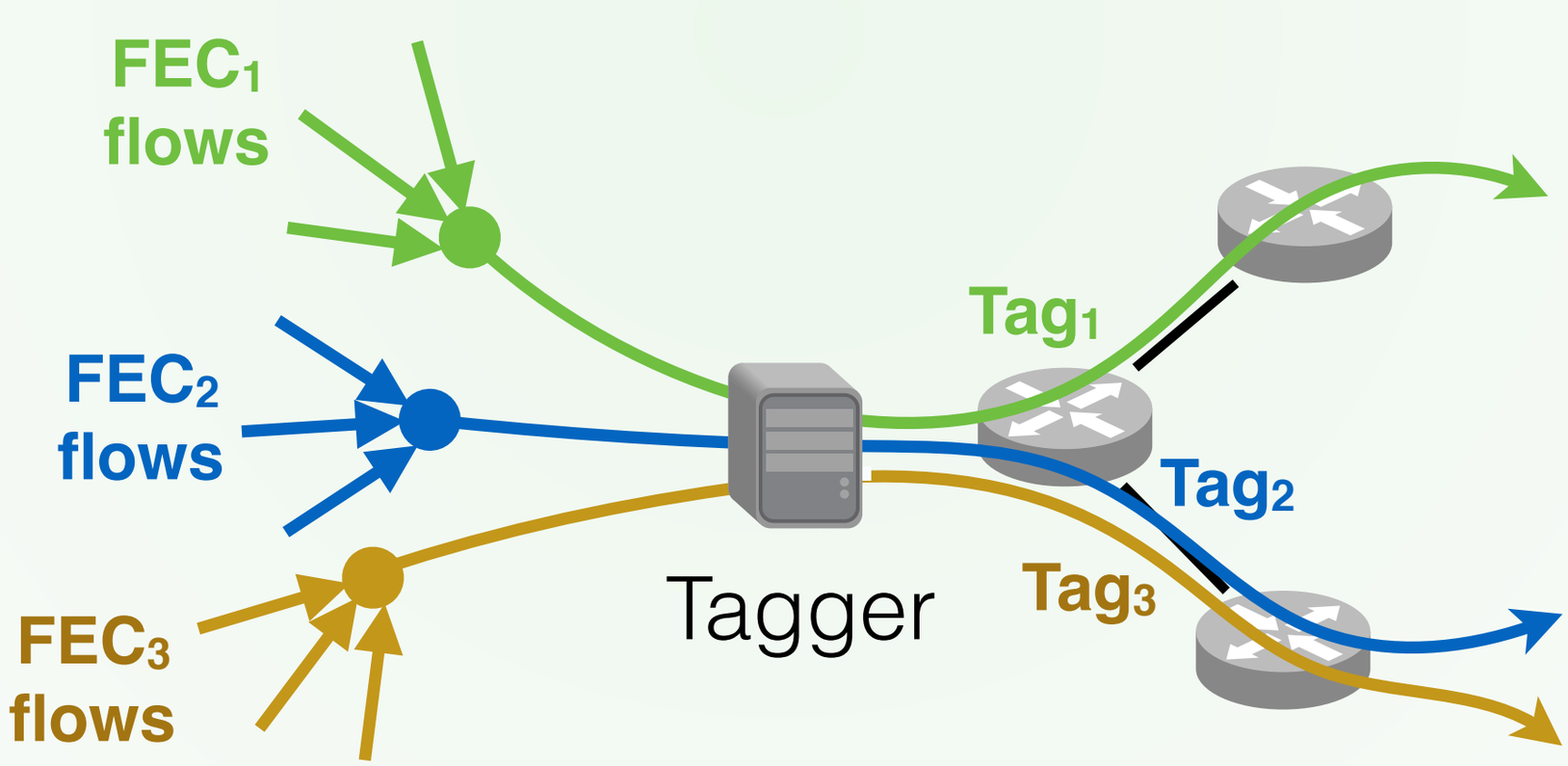
Forwarding Equivalence Classes (FECs)

Many flows have the same forwarding actions



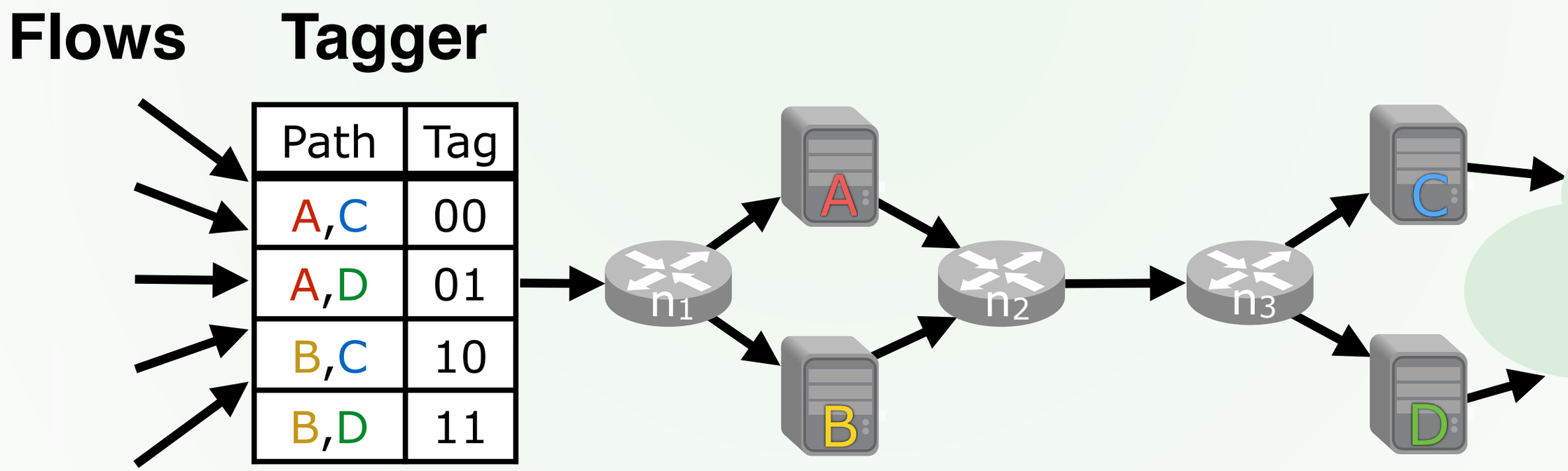
Fewer Rules with FEC Tags

- Reclassifying flows at each switch is costly
- Attach FEC tag to each packet
- Use Cases: Forwarding, Load Balancing, Billing, Policy Enforcement



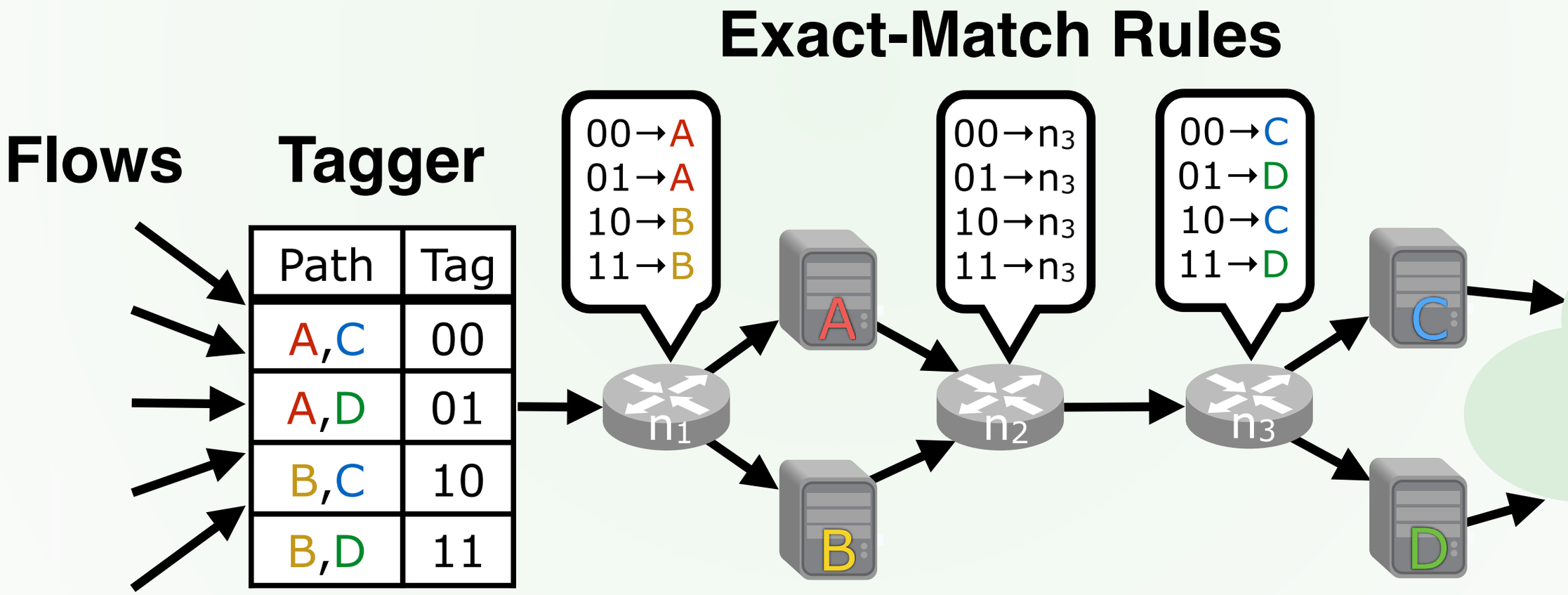
Example: Service Chaining

- Routing packet through middlebox paths



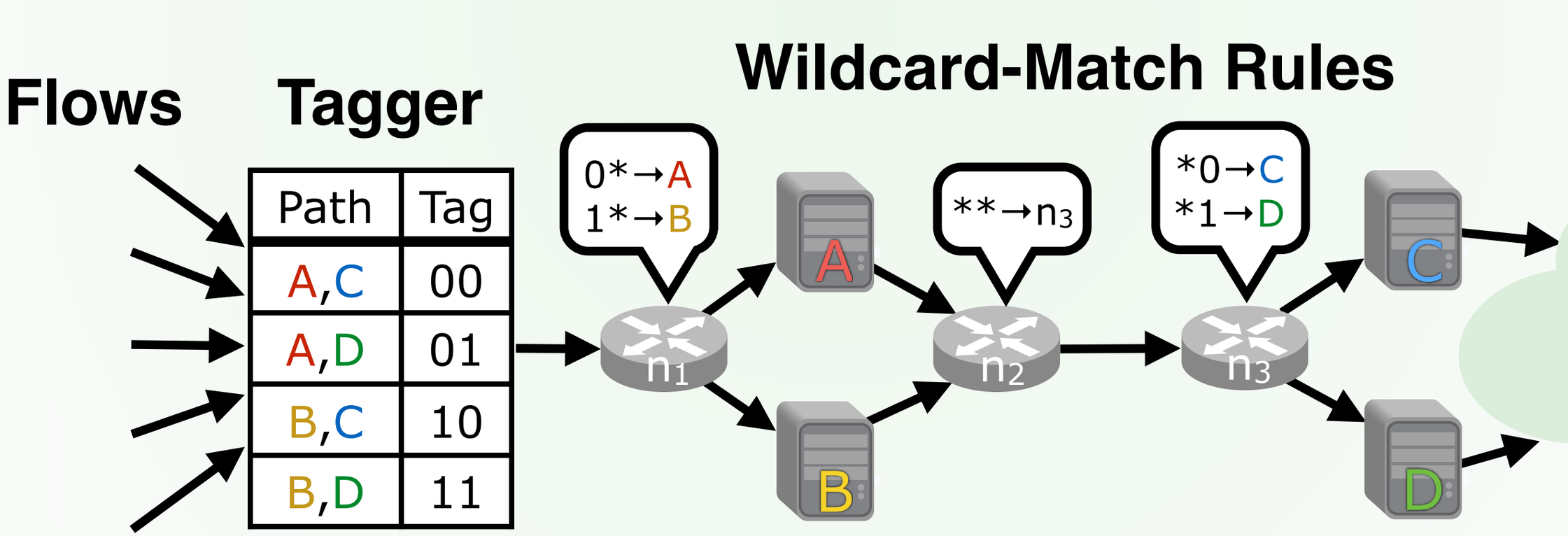
Example: Service Chaining

- Routing packet through middlebox paths



Example: Service Chaining

- Routing packet through middlebox paths



Tagging Limitations

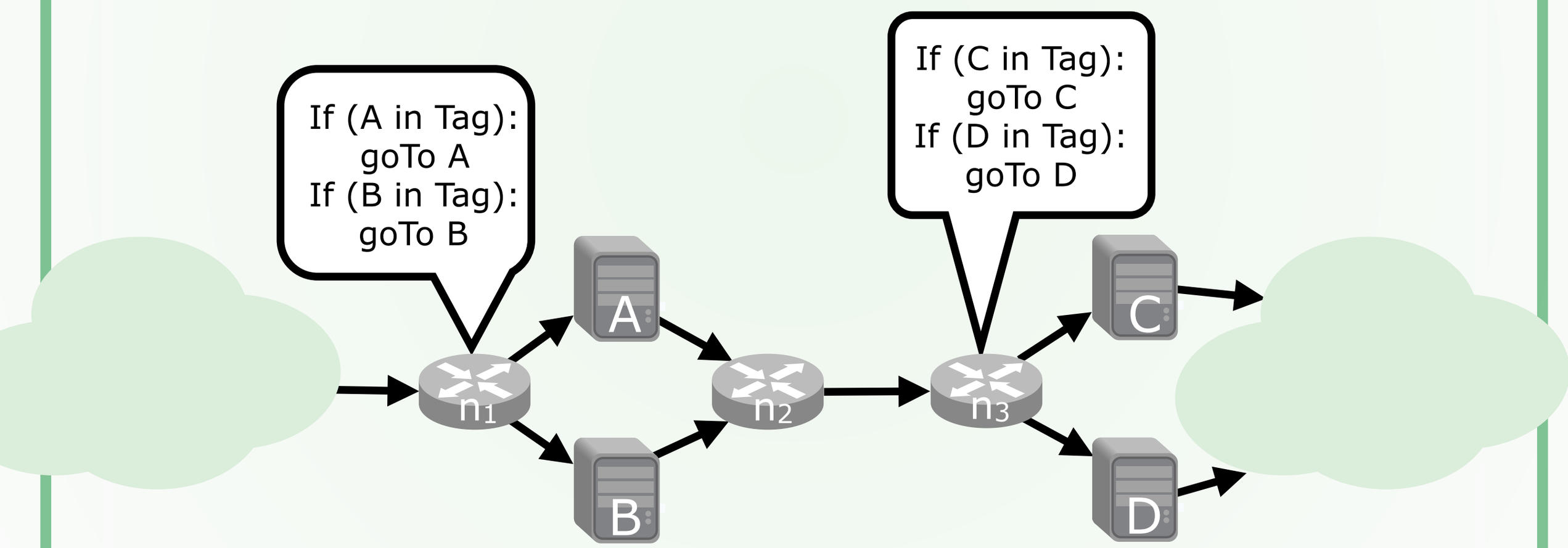
- *Flat Tags* need switch tables linear in the number of tags

- Cannot make aggregate actions across flat tags

- Modern switches can wildcard match on packets, can exploit to build better tags

Attribute-Encoding Tags

Switch actions often depend on one *attribute*



Attribute-Encoding Tags

Any tagging problem is composed of two parts:

Attribute-Encoding Tags

Any tagging problem is composed of two parts:

1. A Tag for every FEC

FEC	Attributes	Tag
1	hit Mbox A, hit Mbox C	00
2	hit Mbox A, hit Mbox D	01
3	hit Mbox B, hit Mbox C	10
4	hit Mbox B, hit Mbox D	11

Attribute-Encoding Tags

Any tagging problem is composed of two parts:

1. A Tag for every FEC

FEC	Attributes	Tag
1	hit Mbox A, hit Mbox C	00
2	hit Mbox A, hit Mbox D	01
3	hit Mbox B, hit Mbox C	10
4	hit Mbox B, hit Mbox D	11

2. Wildcard strings to match on FEC attributes

Attribute	Match Condition
hit Mbox A	Compare Tag to 0*
hit Mbox B	Compare Tag to 1*
hit Mbox C	Compare Tag to *0
hit Mbox D	Compare Tag to *1

Attribute-Encoding Tags

Any tagging problem is composed of two parts:

1. A Tag for every FEC

FEC	Attributes	Tag
1	hit Mbox A, hit Mbox C	00
2	hit Mbox A, hit Mbox D	01
3	hit Mbox B, hit Mbox C	10
4	hit Mbox B, hit Mbox D	11

2. Wildcard strings to match on FEC attributes

Attribute	Match Condition
hit Mbox A	Compare Tag to 0*
hit Mbox B	Compare Tag to 1*
hit Mbox C	Compare Tag to *0
hit Mbox D	Compare Tag to *1

Tradeoff: Tag width VS complexity of match conditions

Tagging Applications

Application	Existing Solution	Attributes	Tag Field	Tag Conveyed By
SDN-Enabled IXP	iSDX	Advertising Peers	Destination MAC	ARP
Service Chaining	FlowTags	Middleboxes	IP Fragment Field	First Middlebox
Policy Enforcement	Alpaca	Host Permissions	IP Source Address	DHCP

PathSets Outline

1. Construct tagging scheme for unordered sets of attributes

2. Extend scheme to support ordered sequences of attributes

3. Using prefix codes to reduce tag size

PathSets Outline

1. **Construct tagging scheme for unordered sets of attributes**
2. Extend scheme to support ordered sequences of attributes
3. Using prefix codes to reduce tag size

Strawman Approach

Attribute Sets

FEC	Attributes
S_1	B, C
S_2	B, C, D
S_3	D
S_4	D, E
S_5	D, E, F

Attribute Vectors

FEC	Attributes
S_1	B C _ _ _
S_2	B C D _ _
S_3	_ _ D _ _
S_4	_ _ D E _
S_5	_ _ D E F

Strawman Approach

Attribute Vectors

FEC	Attributes
S_1	B C _ _ _
S_2	B C D _ _
S_3	_ _ D _ _
S_4	_ _ D E _
S_5	_ _ D E F

Subsets of [B,C,D,E,F] →

Vector Bitmasks

FEC	Bitmask
S_1	11000
S_2	11100
S_3	00100
S_4	00110
S_5	00111

Strawman Approach

Very simple match rules!

Tags

Set	Bitmask
B,C	11000
B,C,D	11100
D	00100
D,E	00110
D,E,F	00111

Match Patterns

Attribute	Match String
B	1****
C	*1***
D	**1**
E	***1*
F	****1

Strawman Approach

Problem: Tag size is linear in the number of attributes to encode. Scales poorly

Set	Bitmask
B,C	11000
B,C,D	11100
D	00100
D,E	00110
D,E,F	00111

Partial Masks

Attribute Vectors

FEC	Attributes
S_1	B C _ _ _
S_2	B C D _ _
S_3	_ _ D _ _
S_4	_ _ D E _
S_5	_ _ D E F

Subsets of [B,C,D,E,F] →

Vector Bitmasks

FEC	Bitmask
S_1	11000
S_2	11100
S_3	00100
S_4	00110
S_5	00111

Partial Masks

	Attributes
S_1	B C _ _ _
S_2	B C D _ _
S_3	_ _ D _ _
S_4	_ _ D E _
S_5	_ _ D E F

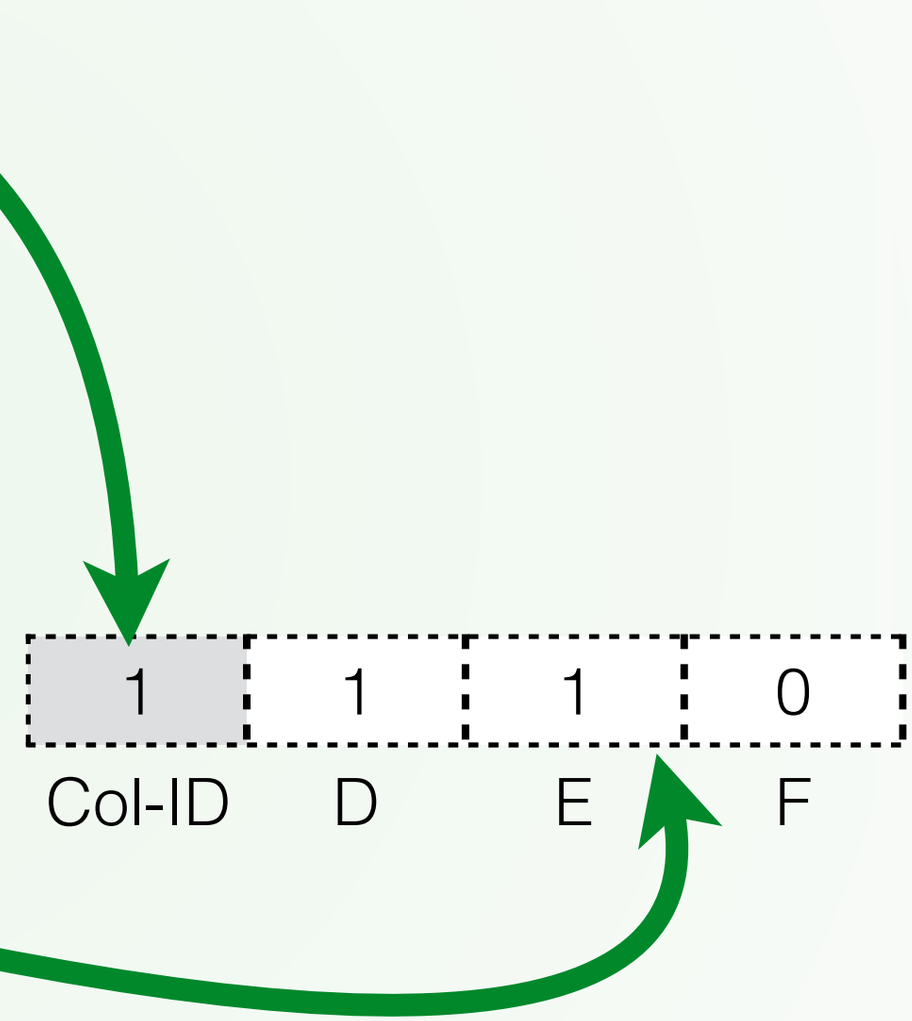
Subsets of [B,C,D]

Subsets of [D,E,F]

	BCD	DEF
S_1	110	
S_2	111	
S_3	001	100
S_4		110
S_5		111

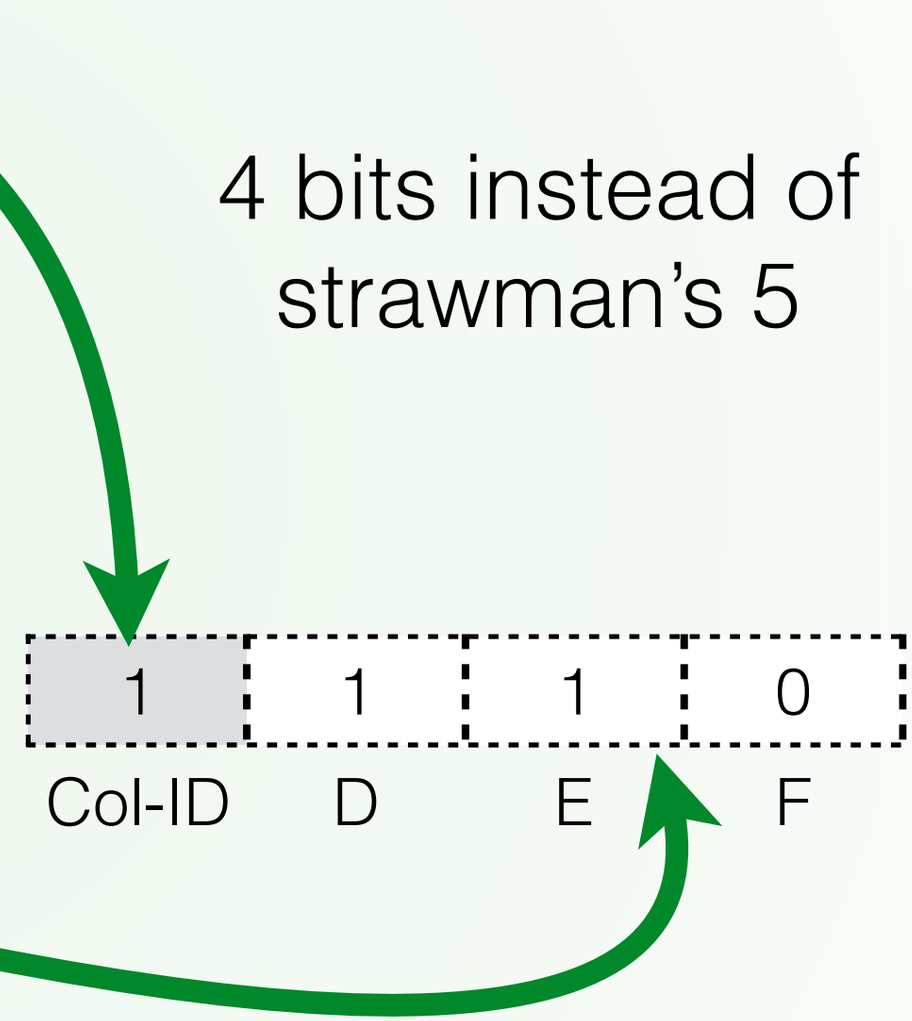
Tag for Set S_4

	Col-0	Col-1
	BCD	**DEF**
S_1	110	
S_2	111	
S_3	001	100
S_4		110
S_5		111



Tag for Set S_4

	Col-0	Col-1
	BCD	**DEF**
S_1	110	
S_2	111	
S_3	001	100
S_4		110
S_5		111

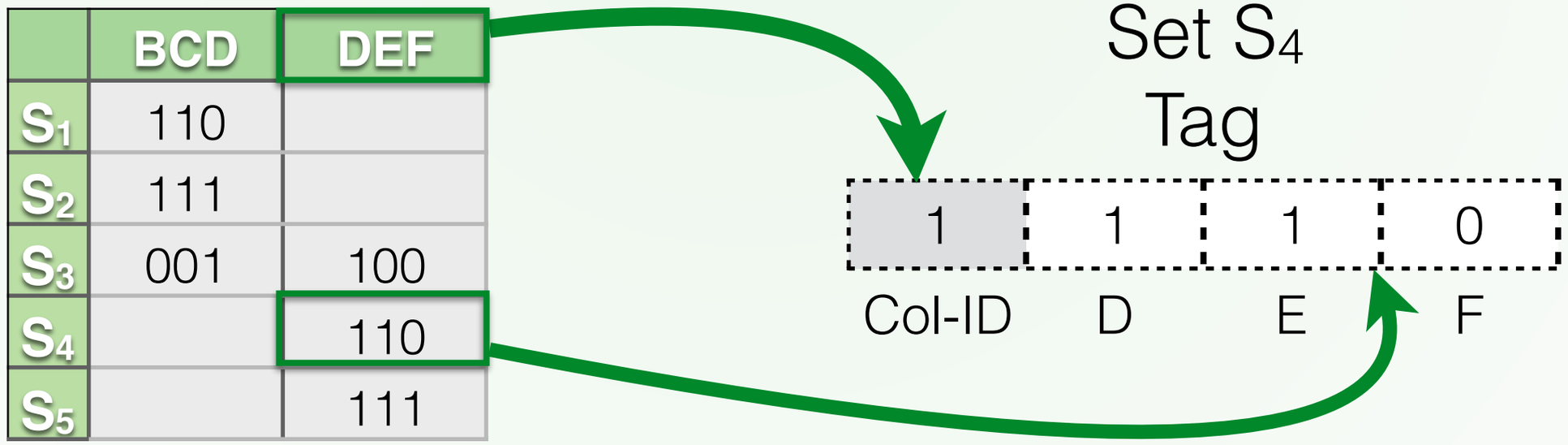


4 bits instead of strawman's 5

Partial Mask Size

- Bits required are now
  Size of Column ID + Mask Size

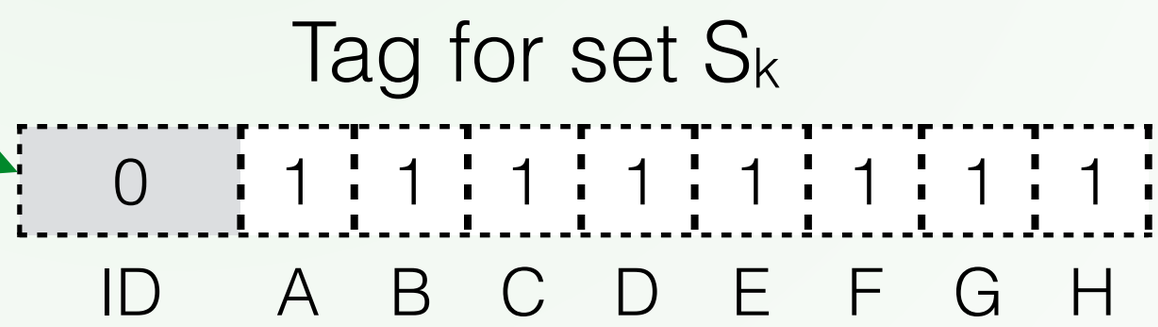
- Now scales to more attributes! (with a catch)



Min Partial Mask Size

- The minimum mask size is the maximum number of attributes in a single set
- Limits applications

	Attributes
.	.
.	.
S_k	ABCDEFGH
.	.
.	.

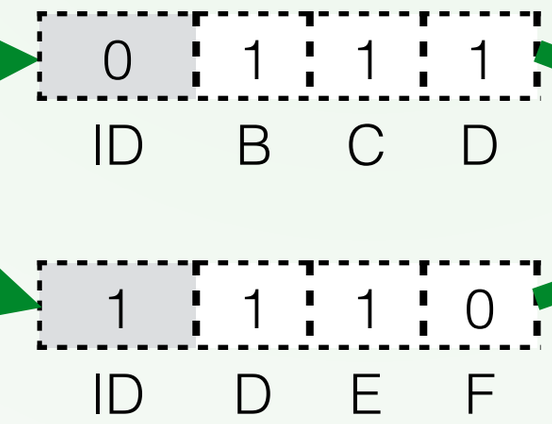


Matching not as easy

- If X appears in multiple columns, then multiple match patterns needed for X

- # match patterns depends upon columns

	BCD	DEF
S1	110	
S2	111	
S3	001	100
S4		110
S5		111



Att	Match
B	01**
C	0*1*
D	0**1 OR 11**
E	1*1*
F	1**1

Matching not as easy

- If X appears in multiple columns, then multiple match patterns needed for X

- # match patterns depends upon columns

6 patterns (strawman had 5)

	BCD	DEF
S1	110	
S2	111	
S3	001	100
S4		110
S5		111

0	1	1	1
ID	B	C	D

1	1	1	0
ID	D	E	F

Att	Match
B	01**
C	0*1*
D	0**1 OR 11**
E	1*1*
F	1**1

PathSets Outline

1. Construct tagging scheme for unordered sets of attributes

2. **Extend scheme to support ordered sequences of attributes**

3. Using prefix codes to reduce tag size

Ordered Attributes

Scheme currently does not support priority matching

Att	Match
B	01**
C	0*1*
D	0**1 OR 11**
E	1*1*
F	1**1

For Tag 0111, any of these three could match!

Ordered Attributes

Only wildcard attributes that come after the attribute of interest -> ordered matching

Order = B<C<D<E<F

Att	Match
B	01**
C	0**0**1*
D	0**00**1 OR 11**
E	1**0**1*
F	1**00**1

For Tag 0111, only the first will match

Attribute Sequences

If all sequences adhere to some *supersequence*, the extension is straightforward

Supersequence	ID
A→B→C→D→E	-

Sequence	Tag
A→B→C	11100
C→D→E	00111
B→C→E	01101
A→C→E	10101

Att.	Match String
A	“1****”
B	“01***”
C	“001**”
D	“0001*”
E	“00001”

Partial Ordering

If the input sequences do not perfectly follow a total ordering, this approach does not work.

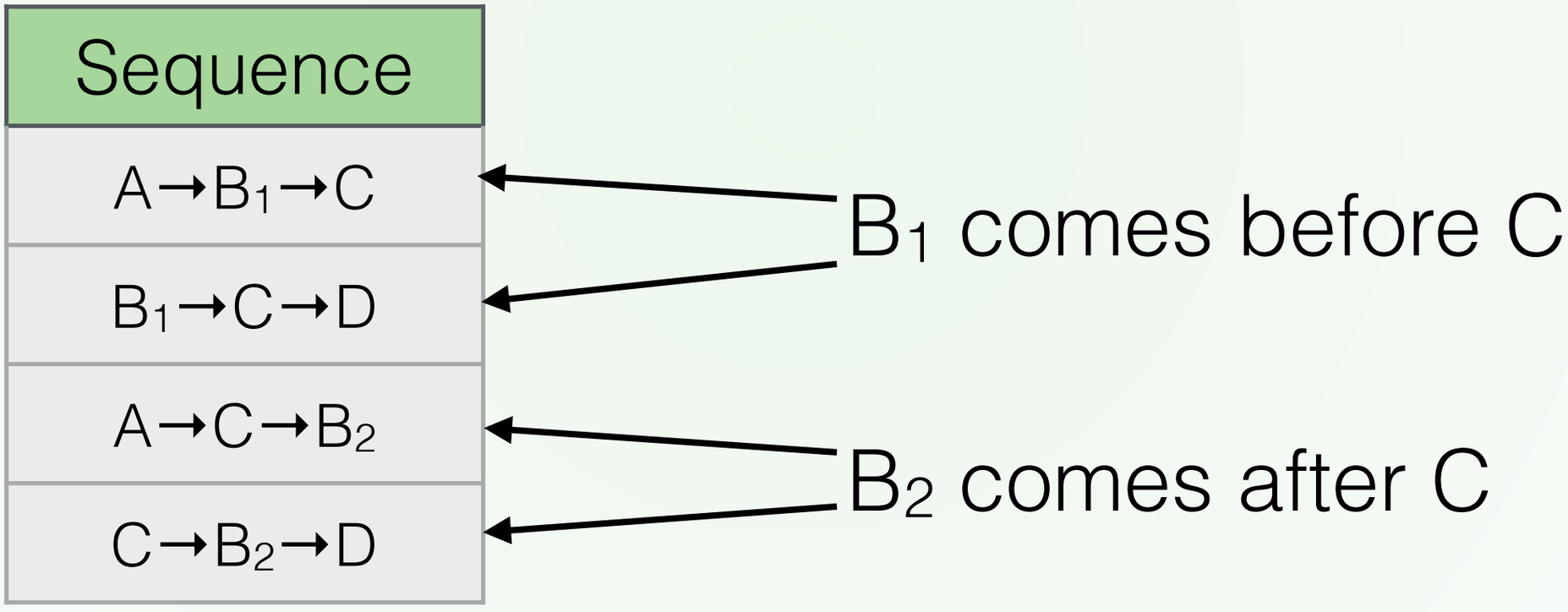
Attribute Sequence
A→B→C
B→C→D
A→C→B
C→B→D

B comes before C

B comes after C

Partial Ordering

To resolve this, we can “split” conflicting attributes into multiple copies, for which a total order exists



Total Order = $A < B_1 < C < B_2 < D$

Partial Ordering

Splitting attributes increases the number of match strings necessary for that attribute

Path	Tag
$A \rightarrow B_1 \rightarrow C$	11100
$B_1 \rightarrow C \rightarrow D$	01101
$A \rightarrow C \rightarrow B_2$	10110
$C \rightarrow B_2 \rightarrow D$	00111

Att	Match
A	1****
B	**01*** OR 0001***
C	001**
D	00001

Total Order = $A < B_1 < C < B_2 < D$

Partial Ordering Resolution

How do we systematically identify and resolve conflicts efficiently?

Partial Ordering Resolution

In general, if attributes repeat, problem known as Shortest Common Supersequence (SCS) problem. Known to be NP-Hard even for only two attributes.

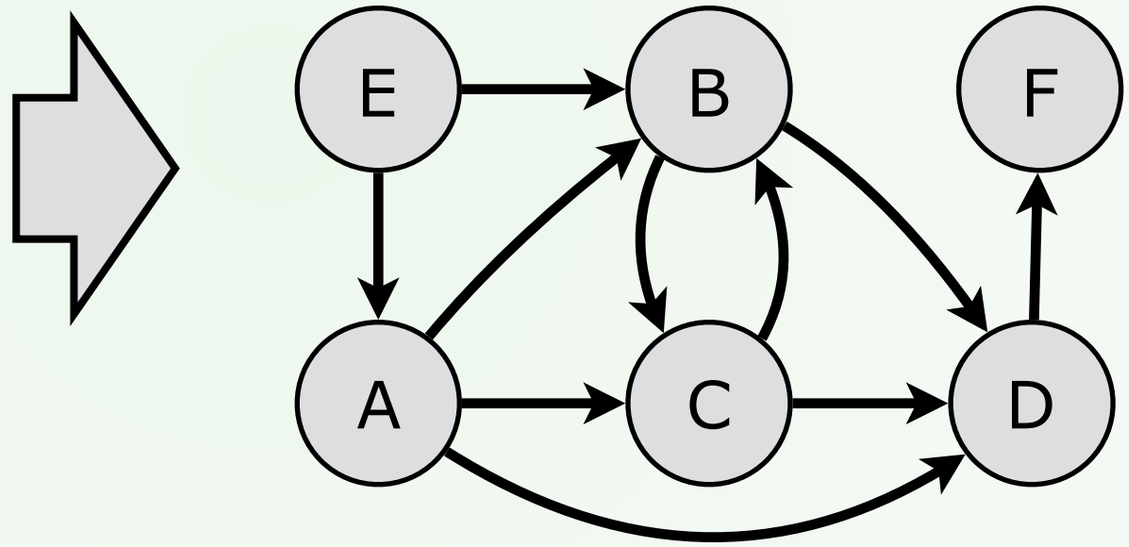
Our case has no repeats and we assume low number of conflicts. We use a heuristic to quickly find a nearly-optimal solution.

SCS Heuristic

Each sequence can be viewed as a simple path through some graph

Sequences
A→C→B→D
E→A→B
D→F
A→B→C→D

Sequence Graph



Create a vertex for each attribute, add edge (X→Y) if X appears before Y in any sequence. Use existing graph theory to identify and resolve conflicts

PathSets Outline

1. Construct tagging scheme for unordered sets of attributes
2. Extend scheme to support ordered sequences of attributes
3. **Using prefix codes to reduce tag size**

Fixed-Length IDs are Inefficient

Fixed-Length IDs are Inefficient

Tags can vary in size

ID	Sets
00	[A,B,C]
01	[C,D]
10	[E,F]
11	[W,X,Y,Z]

Shortest Tag = 4 Bits
Longest Tag = 6 Bits

Fixed-Length IDs are Inefficient

Tags can vary in size

ID	Sets
00	[A,B,C]
01	[C,D]
10	[E,F]
11	[W,X,Y,Z]

Shortest Tag = 4 Bits
Longest Tag = 6 Bits

Many identifiers can be left unused

ID	Sets
000	[E,F]
001	[F,H,I]
010	[L,M]
011	[M,N,O,P]
100	[R,S,T]
101	-
110	-
111	-

Prefix-Free Codes as IDs

Observation: Tags are uniquely decodable iff no ID is a prefix of any other

Can use this to create variable-length identifiers

Use theory of Kraft's Inequality to build identifiers

ID	Sets
1	[A,B,C,D]
01	[E,F,G]
001	[H,I]
0001	[J]
0000	-

Shortest Tag = 5 Bits
Longest Tag = 5 Bits

Empirical Analysis

- Evaluated Scheme for two Applications:

  1. Software-Defined IXP Case (Unordered Sets)

  2. Middlebox paths (Ordered Sets)

SDN-IXP Evaluation

- Used AMS-IX RIPE RIB dumps (633k prefixes, 63 attached networks)
- Generated 1k random SDN policies, computed switch table size

Statistics

- Balancing tradeoffs, tags required 37 bits, used <2k switch entries
- Flat tagging requires ~18 bits, >200k entries

Service Chaining Evaluation

- 800 random paths using Markov chains over some hidden super sequence
- 5% chance of pairwise reordering
- 40 distinct middlebox types

Statistics

- Flat tagging needs 10 bits, ~150 entries per switch
- PathSets needs <19 bits, ~75 entries

Conclusions

- Our tagging scheme is general enough to be used by many applications

- A first step for non-flat attribute tagging

- Code publicly available:
  github.com/PrincetonUniversity/PathSets

Thank You!