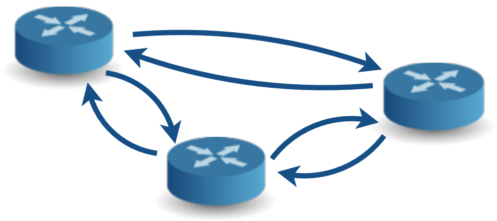
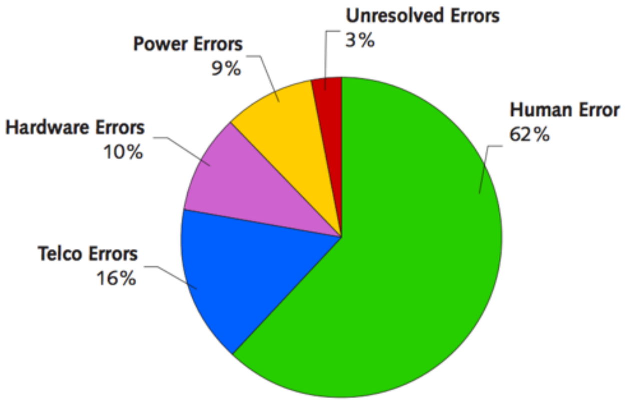
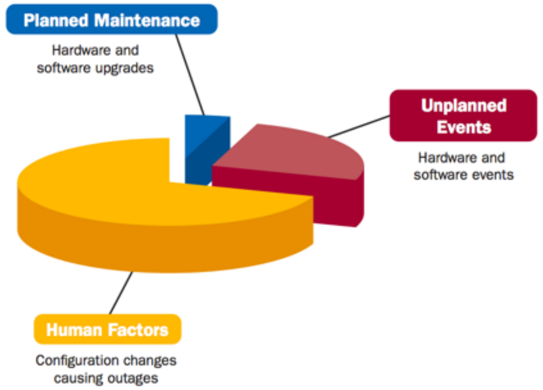
Don’t Mind the Gap:
Bridging Network-wide Objectives and Device-level Configurations

Ryan Beckett (Princeton, MSR)
Ratul Mahajan (MSR)
Todd Millstein (UCLA)
Jitu Padhye (MSR)
David Walker (Princeton)



Configuring Networks is Error-Prone





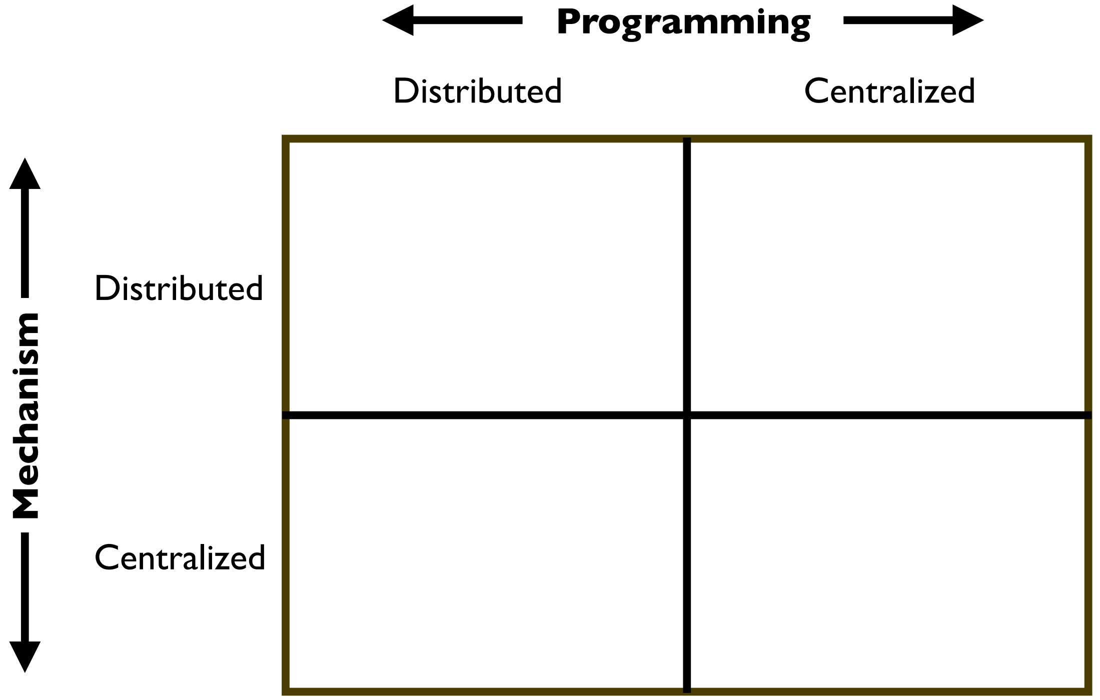
~60% of network downtime is caused by human error

-Yankee group 2002

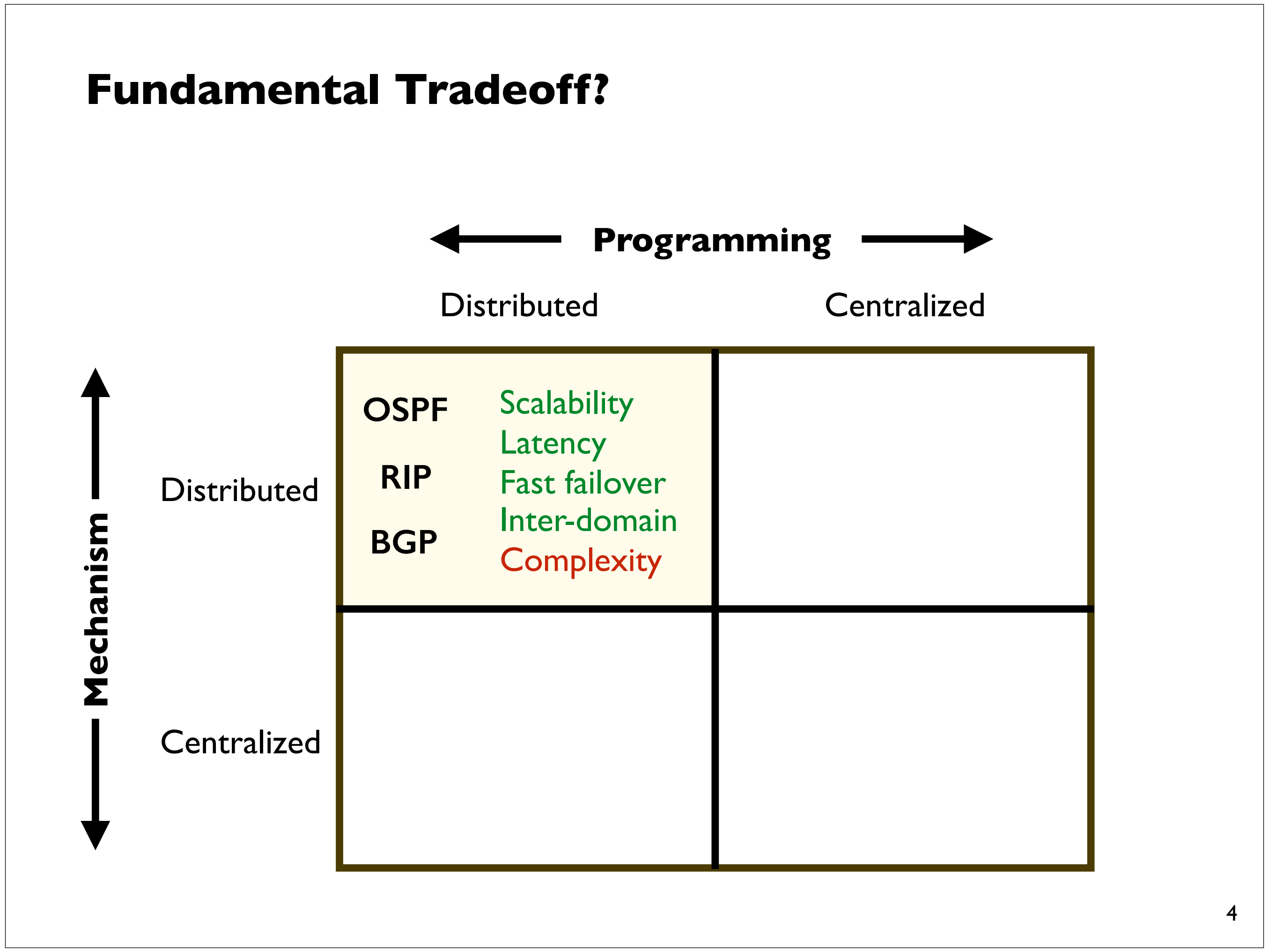
50-80% of outages from configuration changes

-Juniper 2008

Fundamental Tradeoff?



Distinguish what each means



Fundamental Tradeoff?

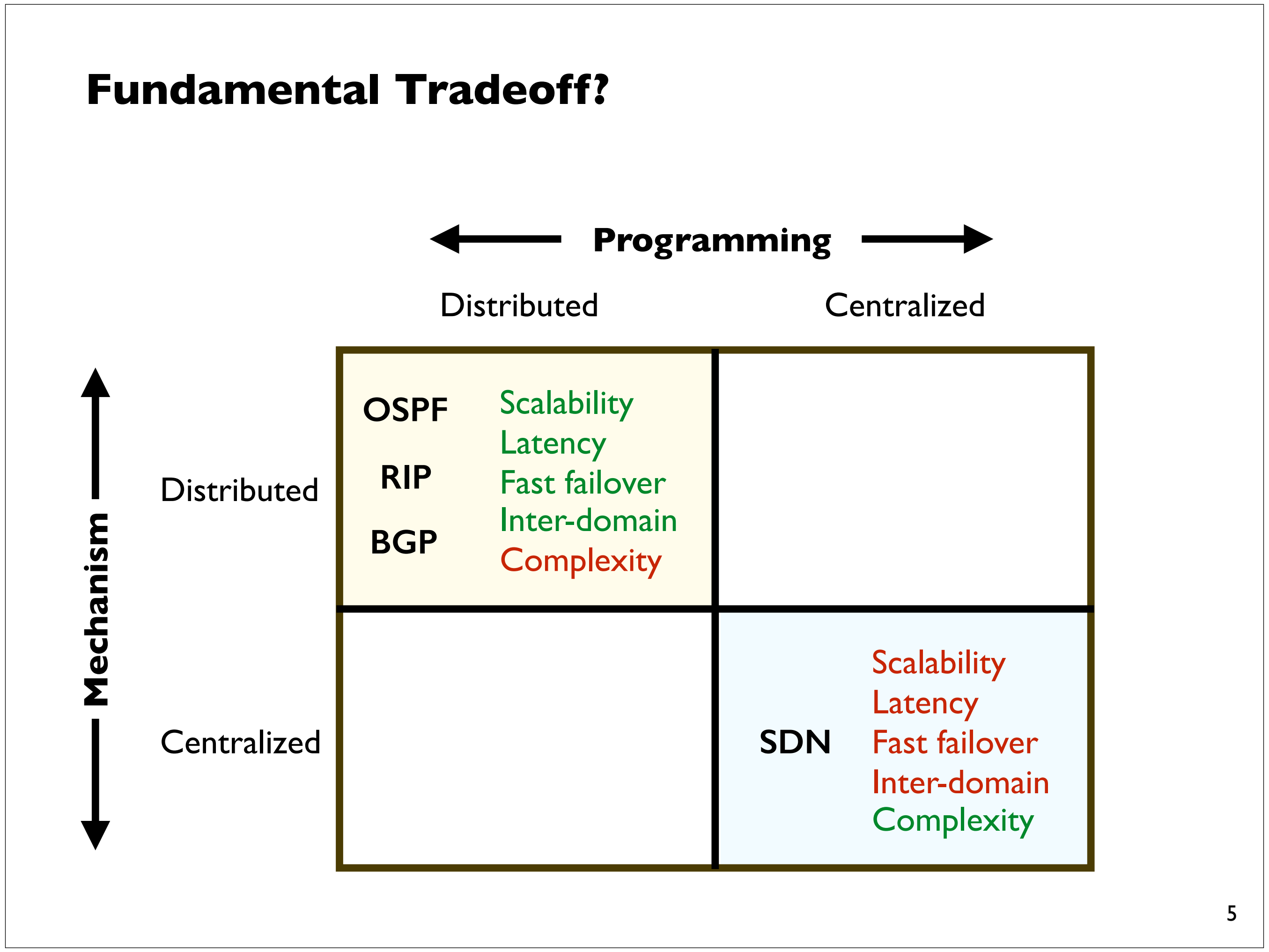
← Programming →

Mechanism	Distributed	Centralized
Distributed	OSPF RIP BGP Scalability Latency Fast failover Inter-domain Complexity	
Centralized		

Explain why BGP/OSPF fall into this category

Explain why SDN falls into this category (e.g., with concrete example: OpenDaylight)

Go through each property in detail for both cases



Fundamental Tradeoff?

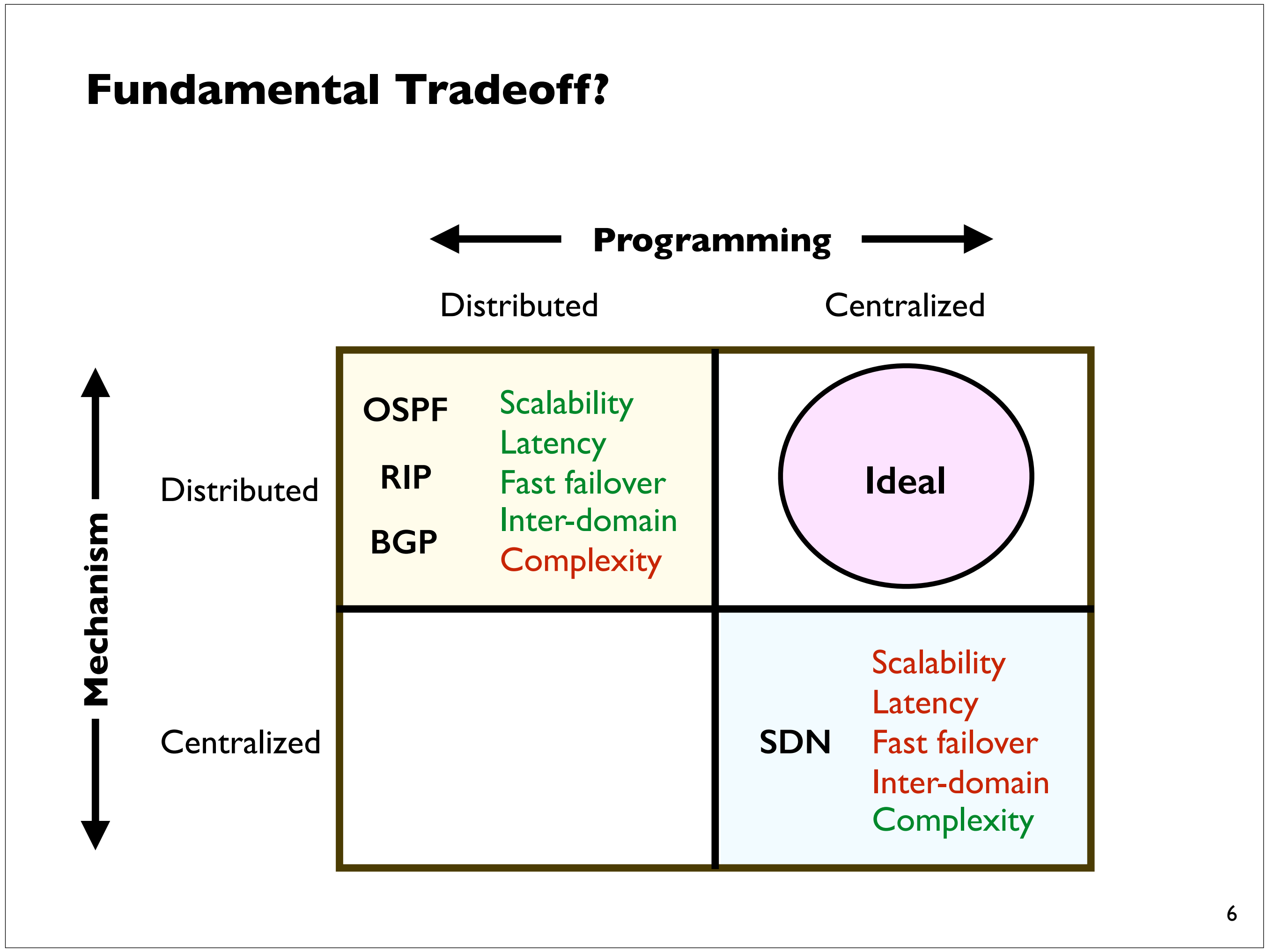
Programming

Mechanism	Distributed	Centralized
Distributed	OSPF RIP BGP Scalability Latency Fast failover Inter-domain Complexity	
Centralized		SDN Scalability Latency Fast failover Inter-domain Complexity

Explain why BGP/OSPF fall into this category

Explain why SDN falls into this category (e.g., OpenDaylight)

Go through each property in detail for both cases



Fundamental Tradeoff?

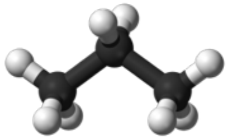
		Programming	
		Distributed	Centralized
Mechanism	Distributed	OSPF RIP BGP Scalability Latency Fast failover Inter-domain Complexity	Ideal
	Centralized		SDN Scalability Latency Fast failover Inter-domain Complexity

Explain why BGP/OSPF fall into this category

Explain why SDN falls into this category (e.g., OpenDaylight)

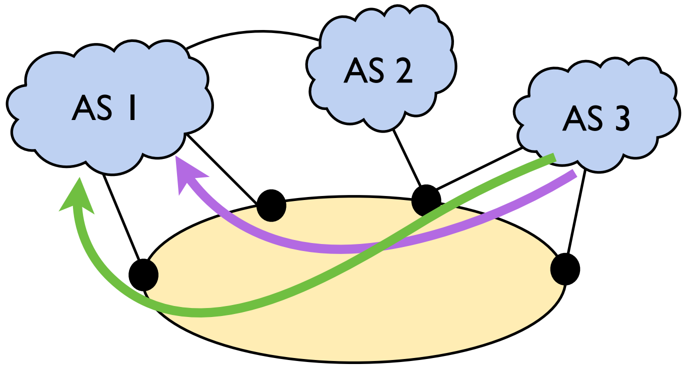
Go through each property in detail for both cases

Propane System

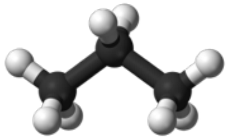


I) Language for expressing high-level operator objectives with:

- **Network-wide** programming abstraction
- Uniform abstractions for **intra**- and **inter**-domain routing
- Path **constraints** and **preferences** in case of failures



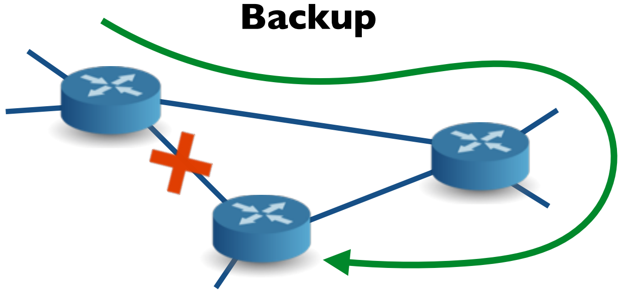
Propane System



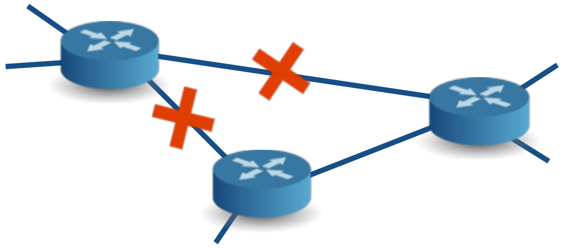
2) Compiler to generate a low-level distributed implementation

- Efficient algorithms to synthesize BGP configs
- Static analysis guarantees policy-compliance
- Policy-compliance holds under all failures

Backup



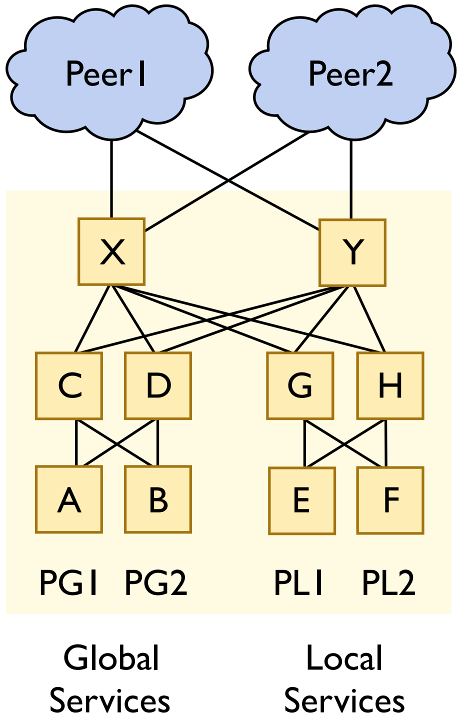
Network partition — drop

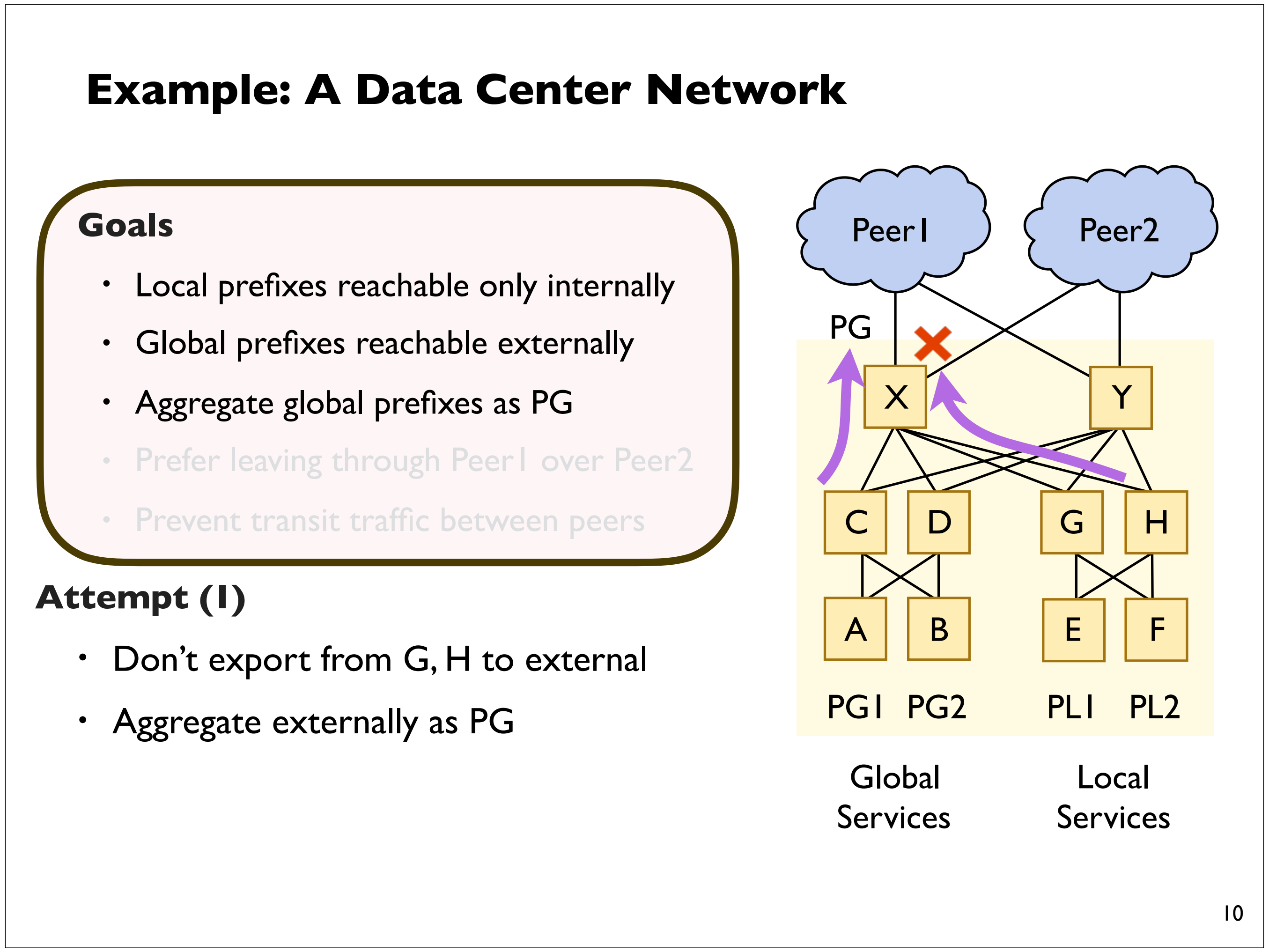


Example: A Data Center Network

Goals

- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as PG
- Prefer leaving through Peer1 over Peer2
- Prevent transit traffic between peers





Example: A Data Center Network

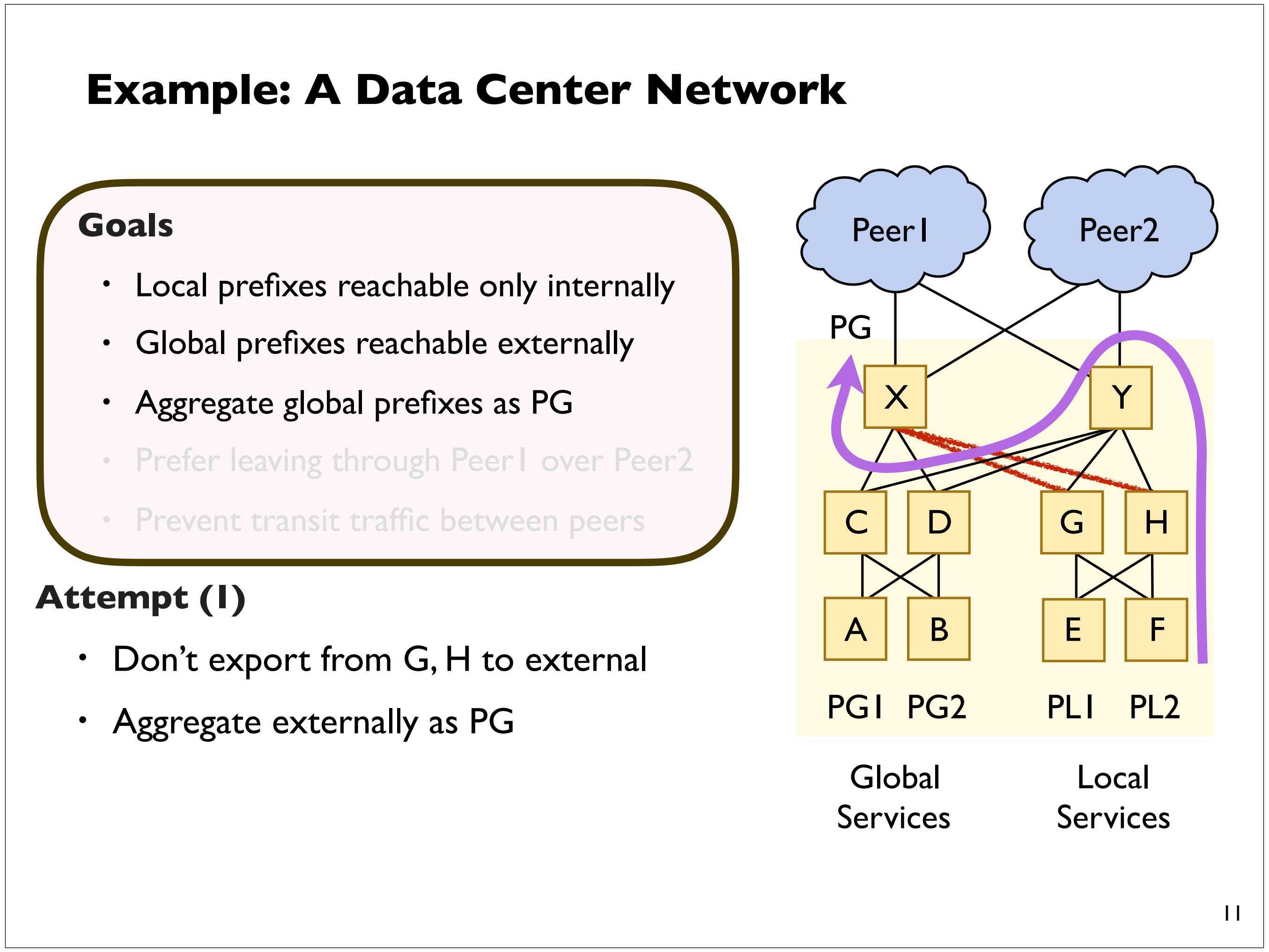
Goals

- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as PG
- Prefer leaving through Peer1 over Peer2
- Prevent transit traffic between peers

Attempt (1)

- Don’t export from G, H to external
- Aggregate externally as PG

Be clear about the direction of traffic vs. advertisement



Example: A Data Center Network

Goals

- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as PG
- Prefer leaving through Peer1 over Peer2
- Prevent transit traffic between peers

Attempt (1)

- Don't export from G, H to external
- Aggregate externally as PG

Be clear about the direction of traffic vs. advertisement

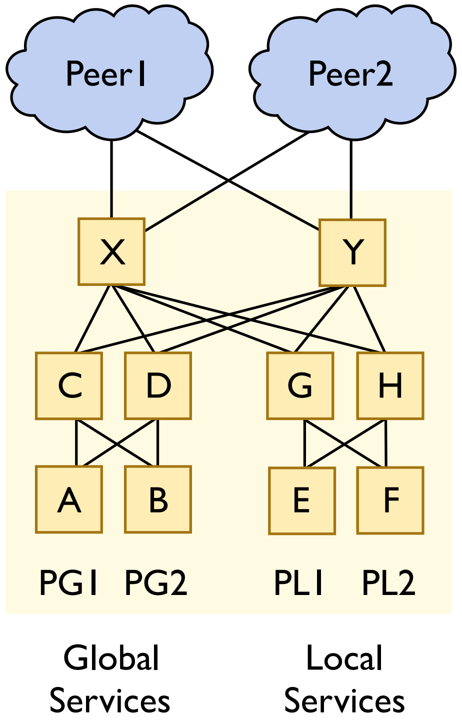
Example: A Data Center Network

Goals

- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as PG
- Prefer leaving through Peer1 over Peer2
- Prevent transit traffic between peers

Attempt (1)

- Don't export from G, H to external
- Aggregate externally as PG
- **X,Y block routes through each other**



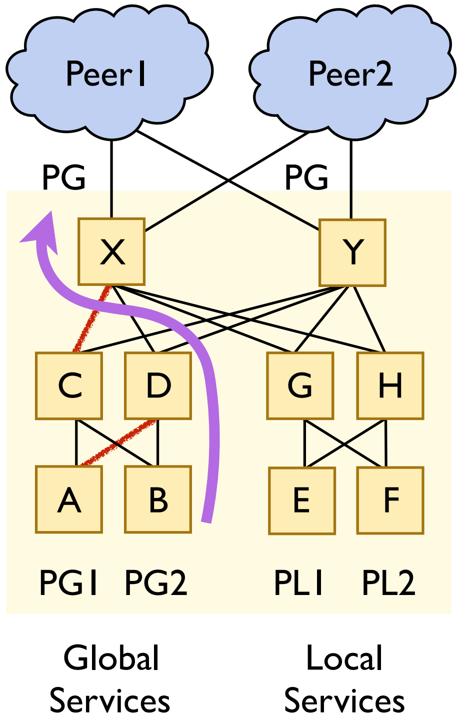
Example: A Data Center Network

Goals

- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as PG
- Prefer leaving through Peer1 over Peer2
- Prevent transit traffic between peers

Attempt (1)

- Don't export from G, H to external
- Aggregate externally as PG
- X,Y block routes through each other



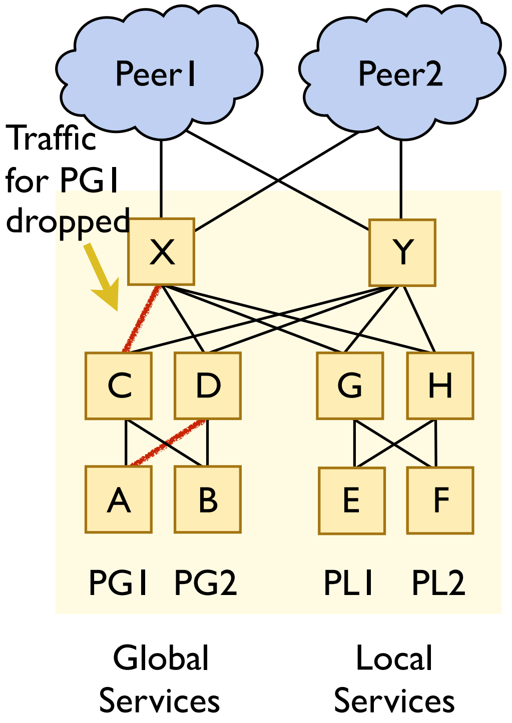
Example: A Data Center Network

Goals

- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as PG
- Prefer leaving through Peer1 over Peer2
- Prevent transit traffic between peers

Attempt (1)

- Don't export from G, H to external
- Aggregate externally as PG
- X,Y block routes through each other



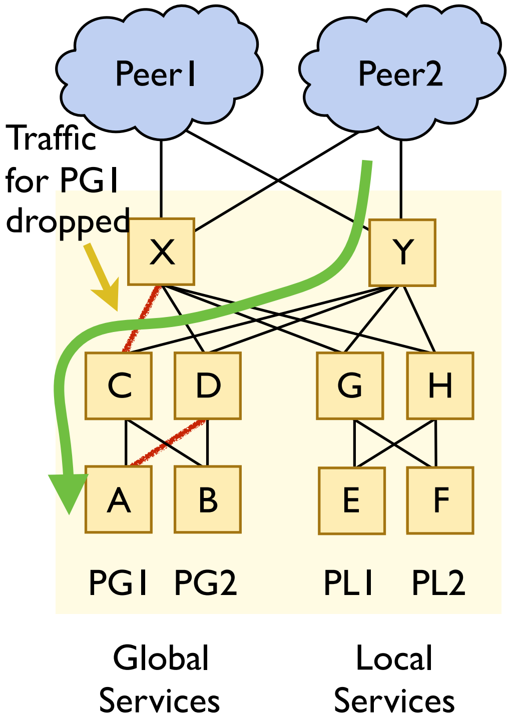
Example: A Data Center Network

Goals

- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as PG
- Prefer leaving through Peer1 over Peer2
- Prevent transit traffic between peers

Attempt (1)

- Don’t export from G, H to external
- Aggregate externally as PG
- X,Y block routes through each other

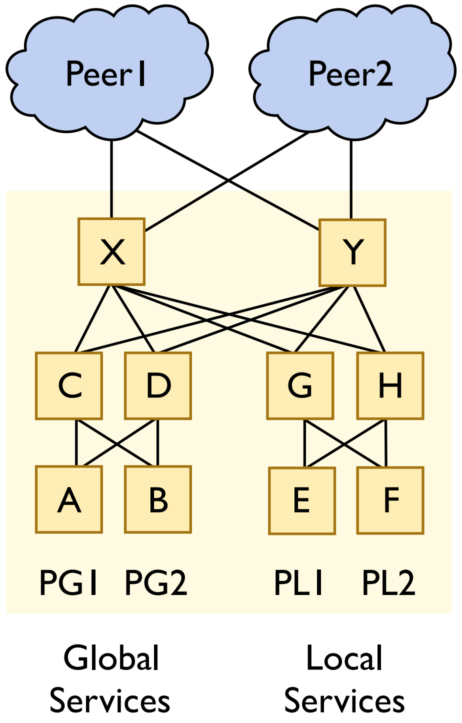


Aggregation-Induced Black Hole!

Example: A Data Center Network

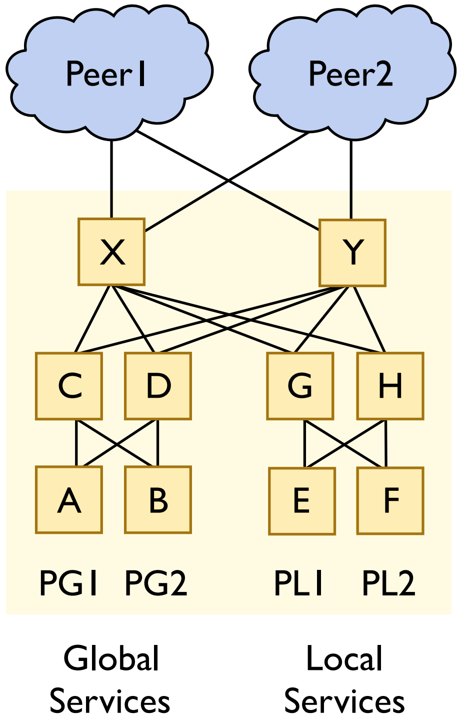
Goals

- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as PG
- Prefer leaving through Peer1 over Peer2
- Prevent transit traffic between peers



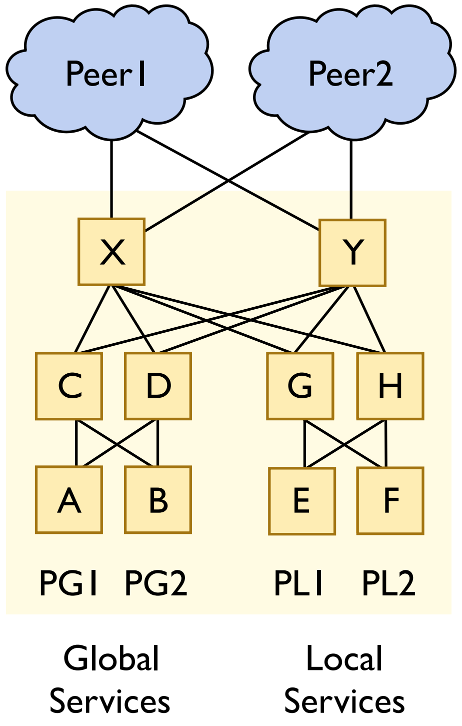
Example: A Data Center Network

```
define Ownership =
 {PG1  => end(A)
  PG2  => end(B)
  PL1  => end(E)
  PL2  => end(F)
  true => exit(Peer1 >> Peer2)}
```



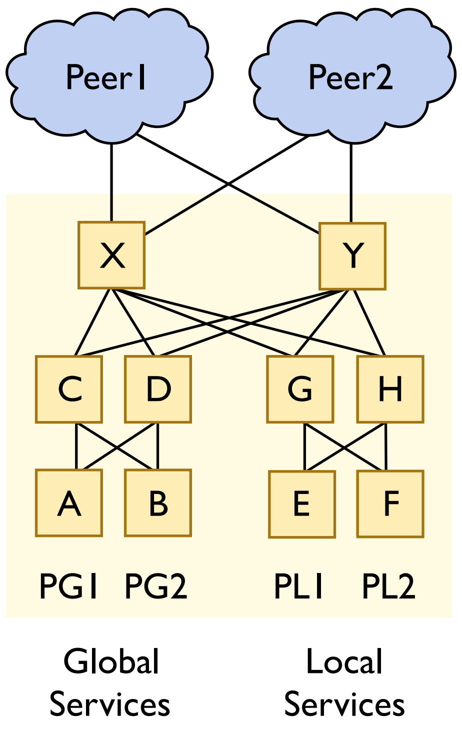
Example: A Data Center Network

```
define Ownership =
 {PG1  => end(A)
  PG2  => end(B)
  PL1  => end(E)
  PL2  => end(F)
  true => exit(Peer1 >> Peer2)}

define Locality =
 {PL1 | PL2 => internal}
```

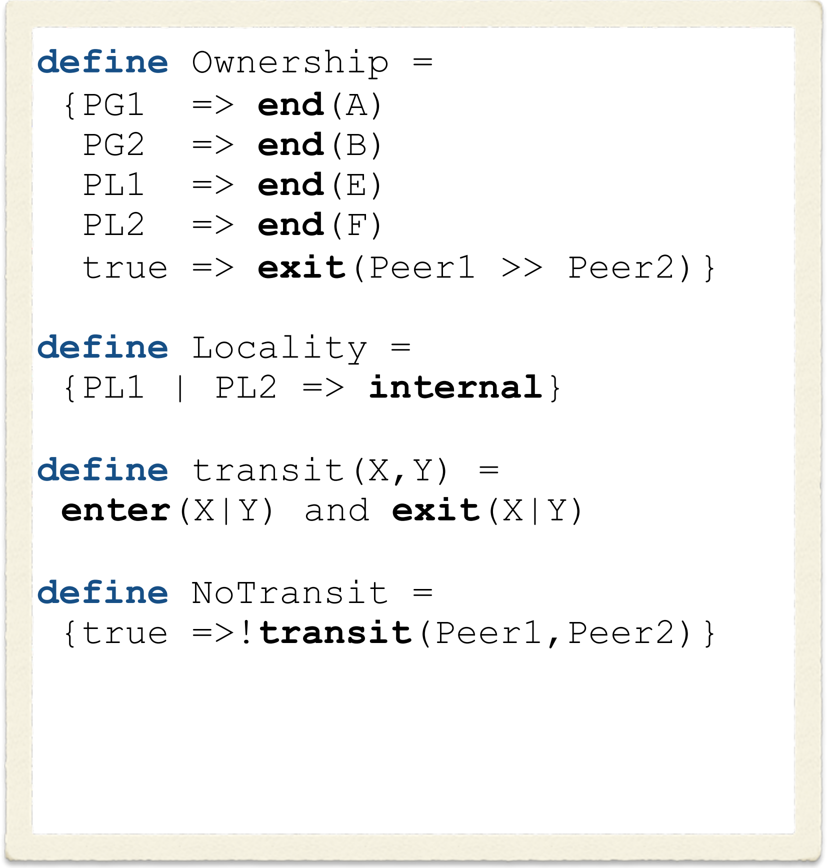


Example: A Data Center Network

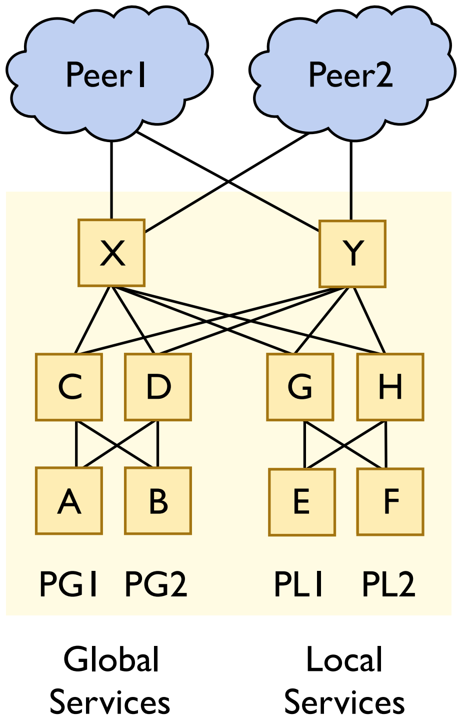
```
define Ownership =
 {PG1   => end(A)
  PG2   => end(B)
  PL1   => end(E)
  PL2   => end(F)
  true => exit(Peer1 >> Peer2)}

define Locality =
 {PL1 | PL2 => internal}

define transit(X,Y) =
 enter(X|Y) and exit(X|Y)
```



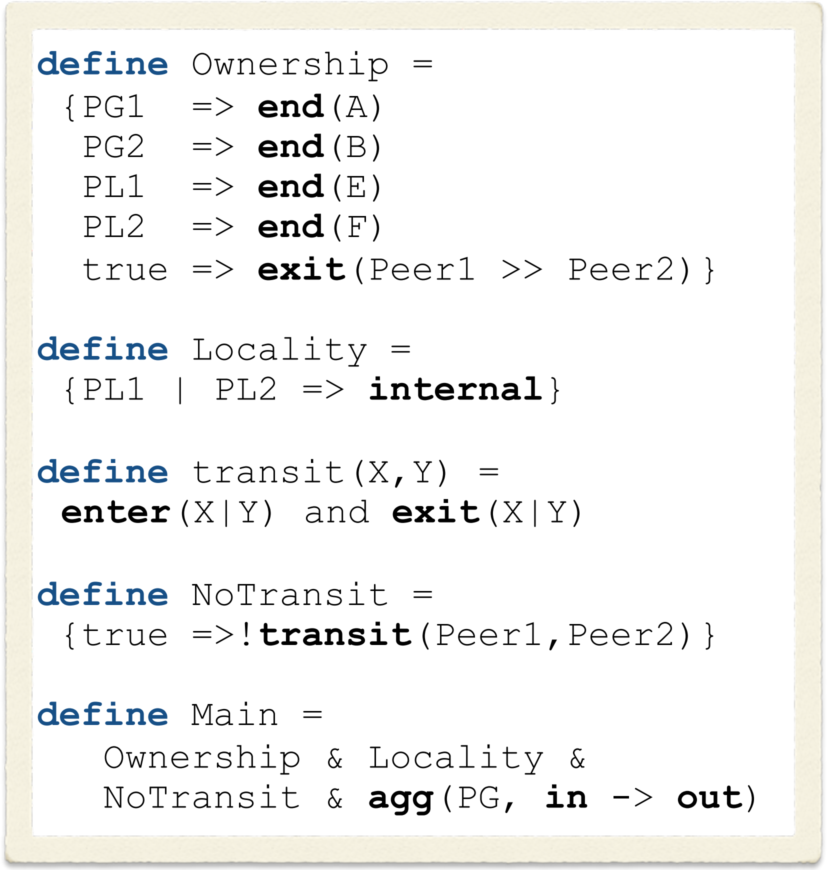
Example: A Data Center Network



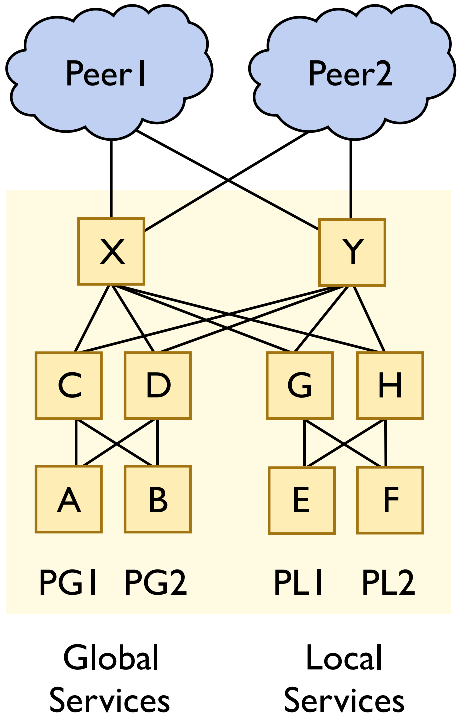
```
define Ownership =
 {PG1  => end(A)
  PG2  => end(B)
  PL1  => end(E)
  PL2  => end(F)
  true => exit(Peer1 >> Peer2)}

define Locality =
 {PL1 | PL2 => internal}

define transit(X,Y) =
 enter(X|Y) and exit(X|Y)

define NoTransit =
 {true =>!transit(Peer1,Peer2)}
```



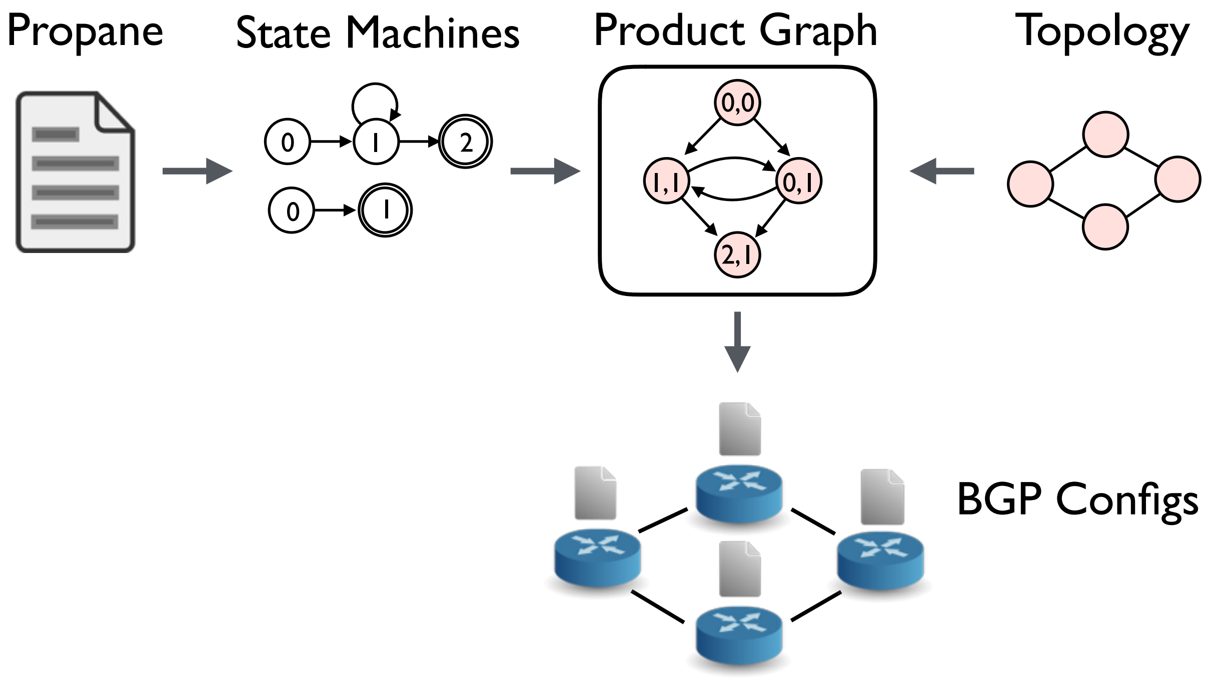
Example: A Data Center Network



```
define Ownership =
 {PG1  => end(A)
  PG2  => end(B)
  PL1  => end(E)
  PL2  => end(F)
  true => exit(Peer1 >> Peer2)}

define Locality =
 {PL1 | PL2 => internal}

define transit(X,Y) =
 enter(X|Y) and exit(X|Y)

define NoTransit =
 {true =>!transit(Peer1,Peer2)}

define Main =
   Ownership & Locality &
   NoTransit & agg(PG, in -> out)
```



Compilation



Propane

State Machines

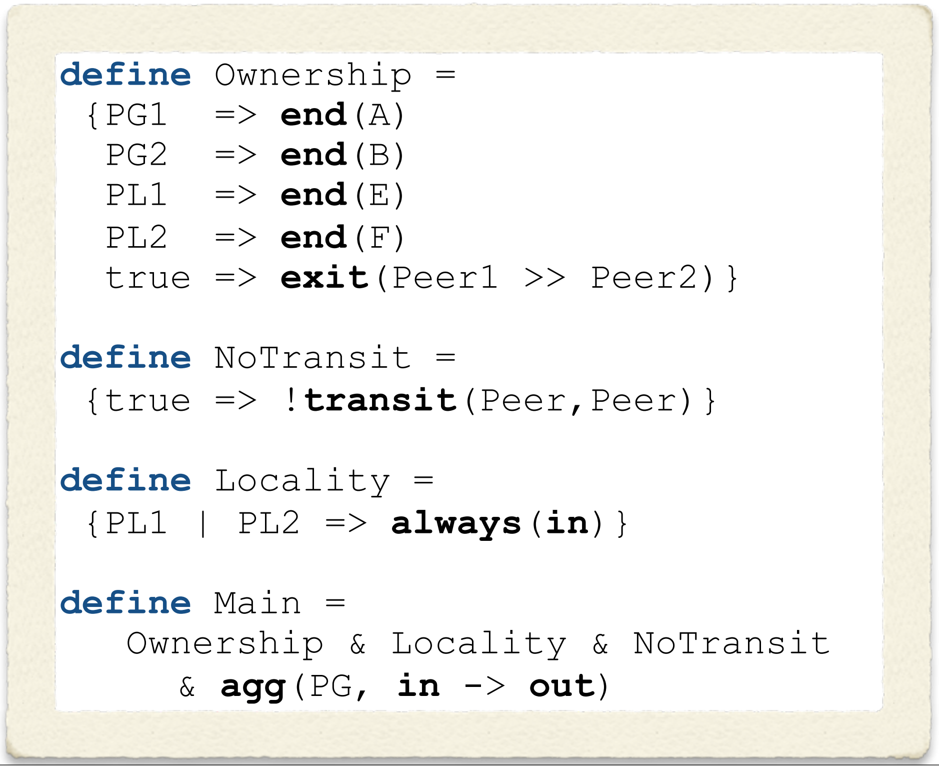
Product Graph

Topology

BGP Configs

Propane Regular IR

Step 1: Combine modular constraints



```
define Ownership =
 {PG1   => end(A)
  PG2   => end(B)
  PL1   => end(E)
  PL2   => end(F)
  true  => exit(Peer1 >> Peer2)}

define NoTransit =
 {true => !transit(Peer,Peer)}

define Locality =
 {PL1 | PL2 => always(in)}

define Main =
   Ownership & Locality & NoTransit
     & agg(PG, in -> out)
```

Propane Regular IR

Prefix-by-prefix intersection of constraints

```
PG1  => !transit(Peer,Peer) & end(A)
PG2  => !transit(Peer,Peer) & end(B)
PL1  => !transit(Peer,Peer) & internal & end(E)
PL2  => !transit(Peer,Peer) & internal & end(F)
true => !transit(Peer,Peer) & exit(Peer1 >> Peer2)
```

Propane Regular IR

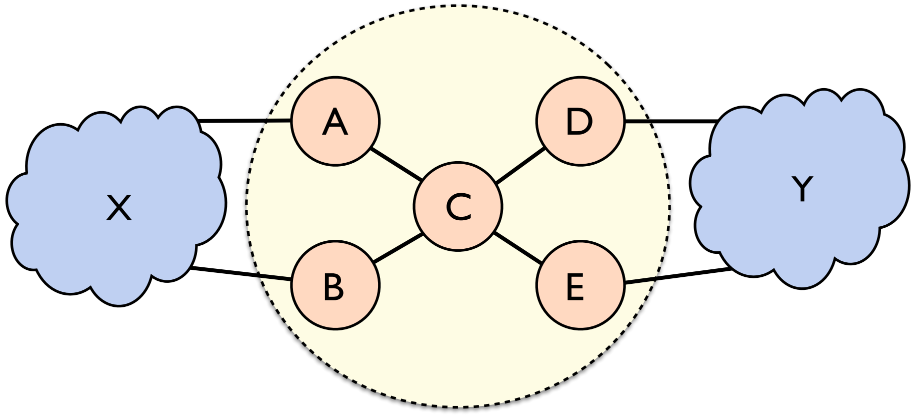
TODO: yellow obscures text

Single prefix

Prefix-by-prefix intersection of constraints

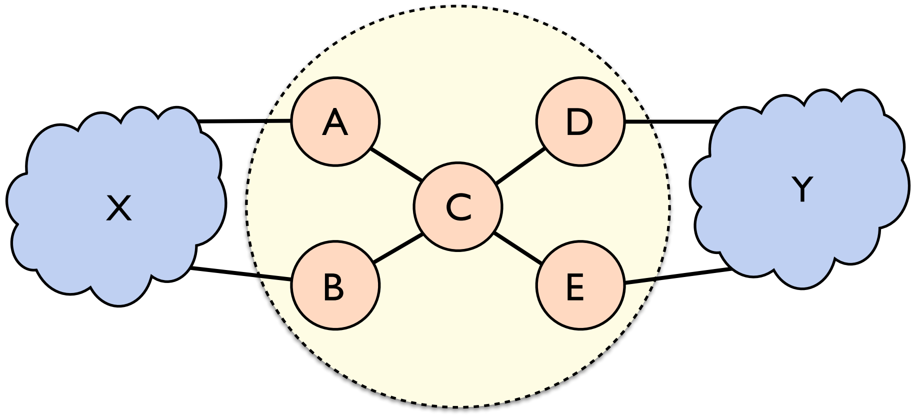
```
PG1  => !transit(Peer,Peer) & end(A)
PG2  => !transit(Peer,Peer) & end(B)
PL1  => !transit(Peer,Peer) & internal & end(E)
PL2  => !transit(Peer,Peer) & internal & end(F)
true => !transit(Peer,Peer) & exit(Peer1 >> Peer2)
```

Compilation: A simple Example



```
end(Y) & (path(A,C,D) >> any)
```

Compilation: A simple Example

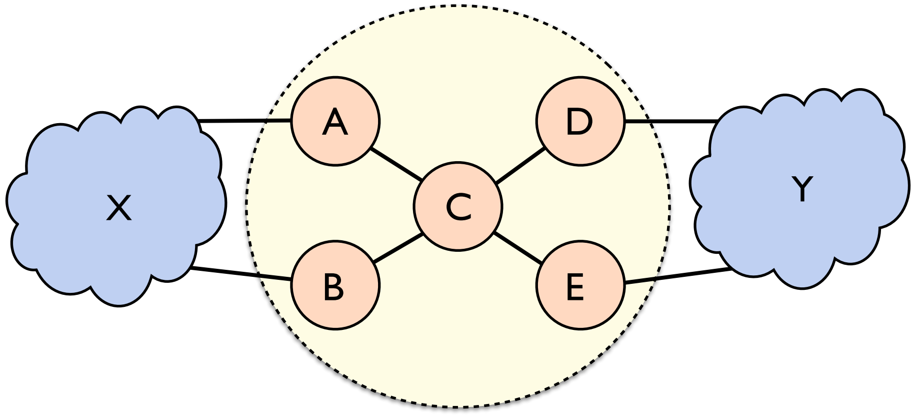


```
end(Y) & (path(A,C,D) >> any)
```

Distribute Preferences

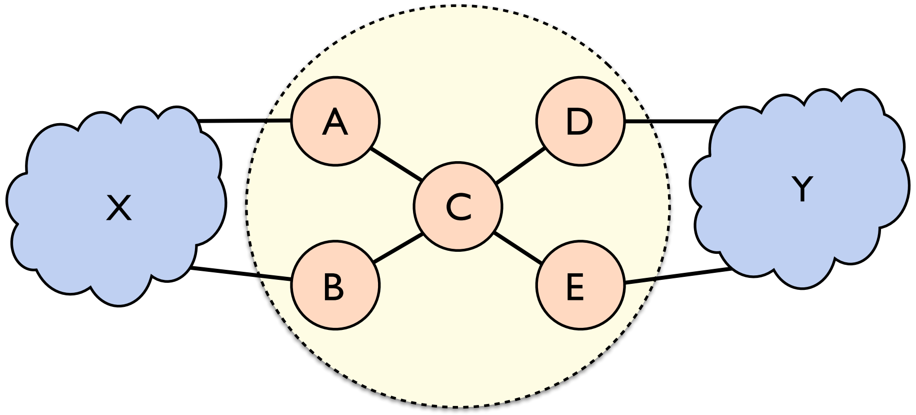
```
(end(Y) & path(A,C,D)) >> (end(Y) & any)
```

Compilation: A simple Example



```
(end(Y) & path(A,C,D)) >> (end(Y) & any)
```

Compilation: A simple Example

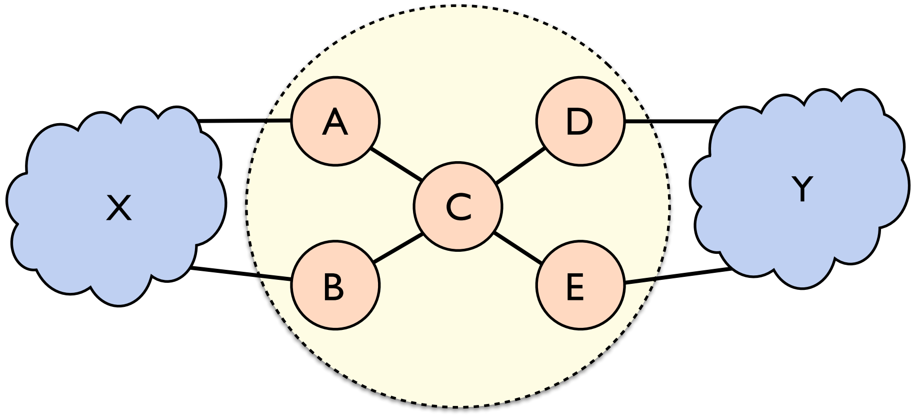


```
(end(Y) & path(A,C,D)) >> (end(Y) & any)
```

Convert to Regex

```
XACDY >> (Σ*)Y
```

Reversed Automata from Policies



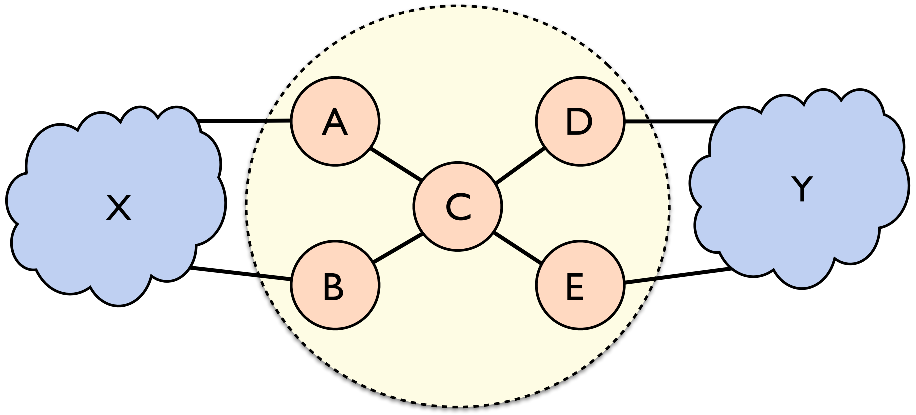
Policy:

1. XACDY

2. (Σ*)Y

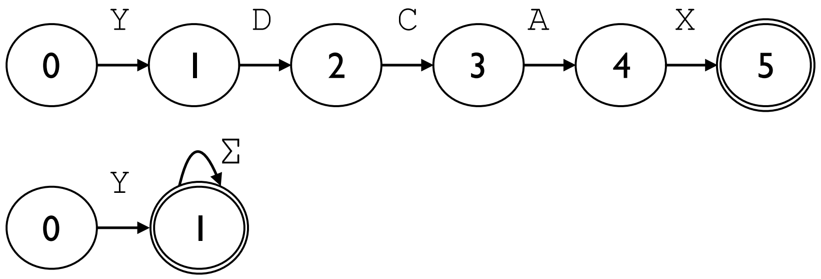
Be clear about the direction of traffic vs. advertisement

Reversed Automata from Policies

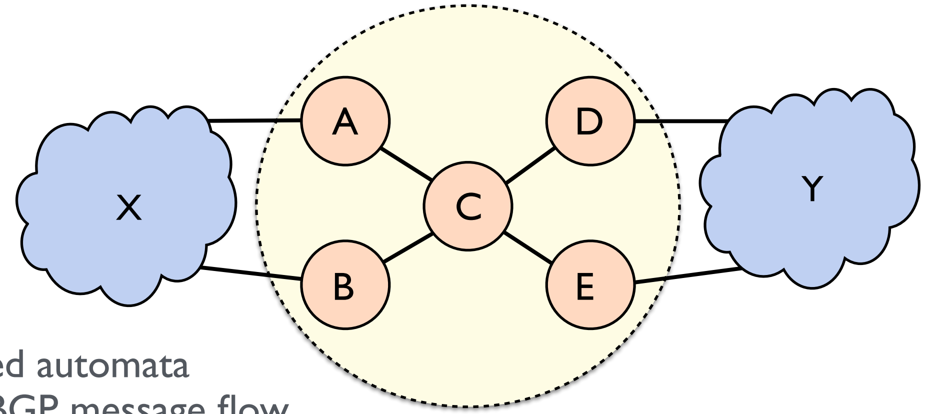


Policy:

```
1.  XACDY

2. (Σ*)Y
```



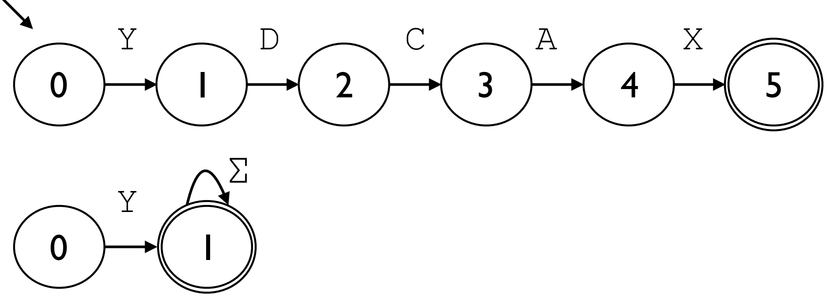
Reversed Automata from Policies



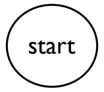
Reversed automata tracks BGP message flow

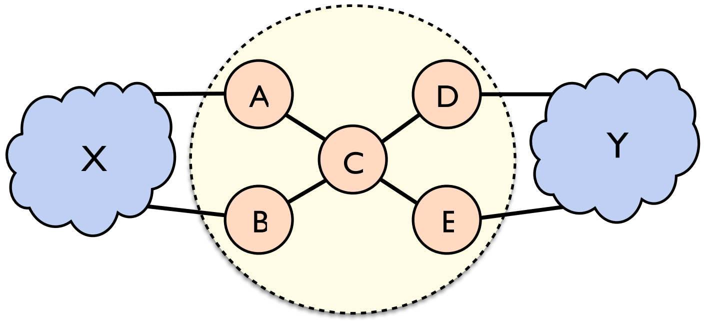
Policy:

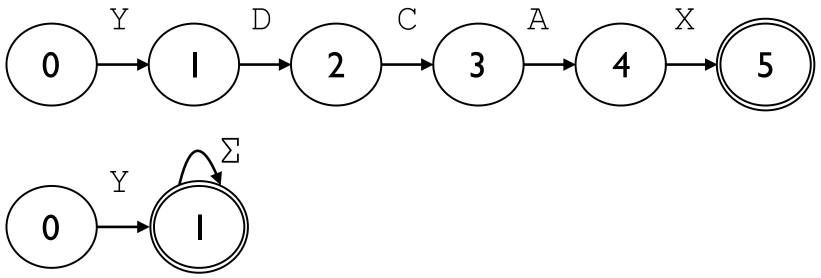
```
1.  XACDY

2. (Σ*)Y
```



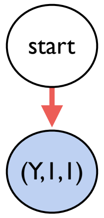
Constructing the Product Graph (PG)

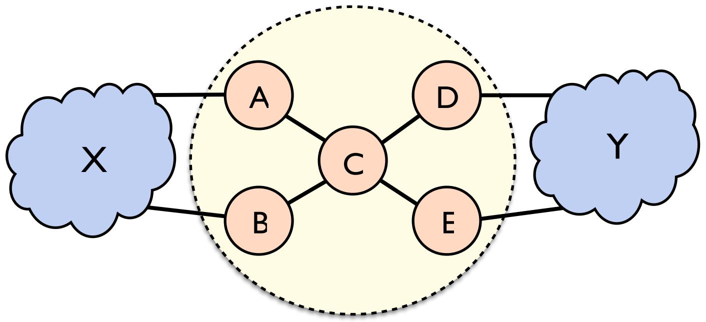


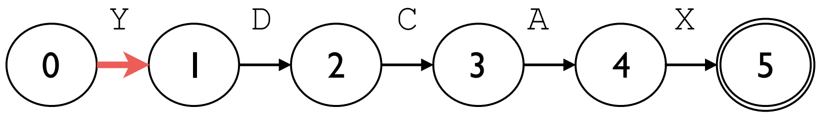


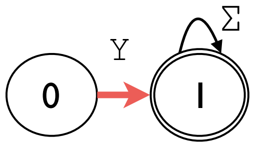


Constructing the Product Graph (PG)

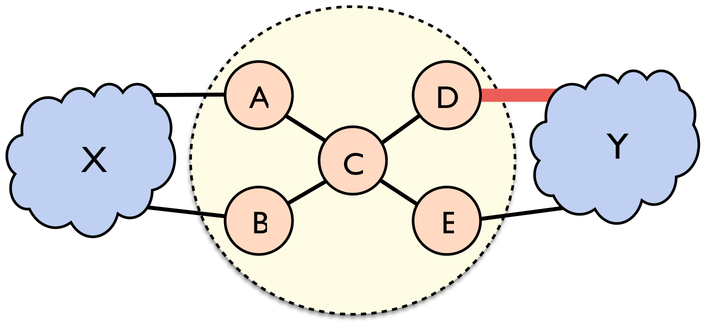


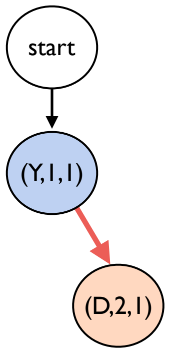


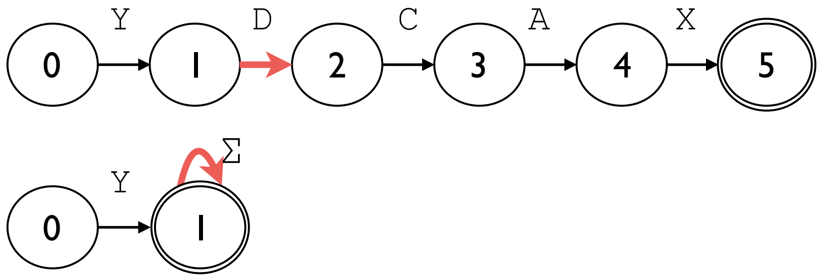




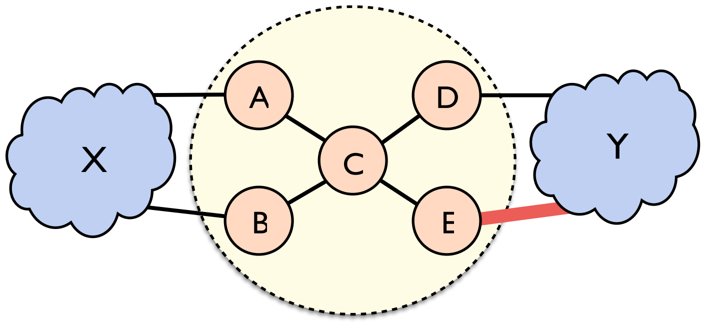
Constructing the Product Graph (PG)

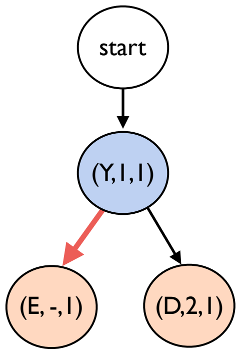


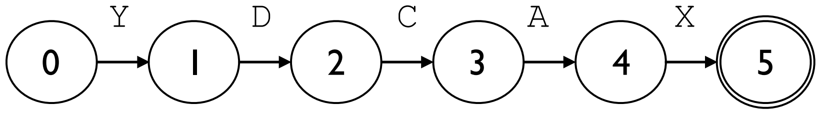


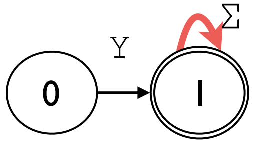


Constructing the Product Graph (PG)

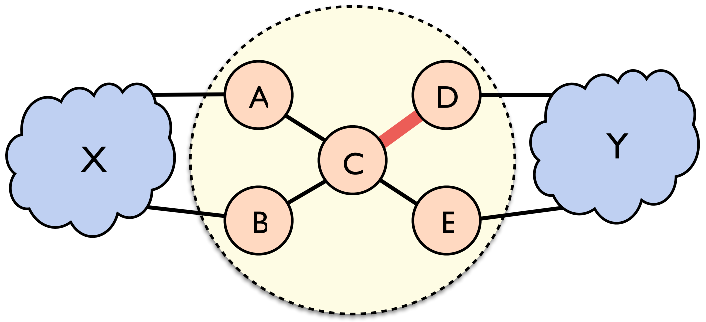


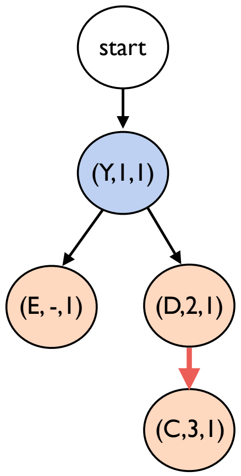


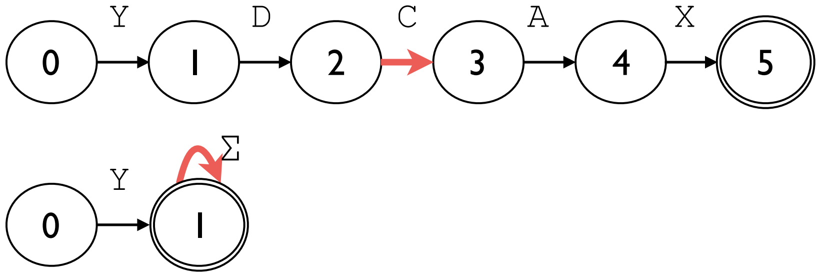




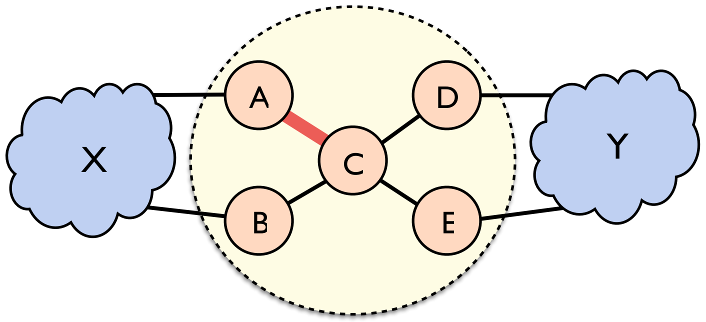
Constructing the Product Graph (PG)

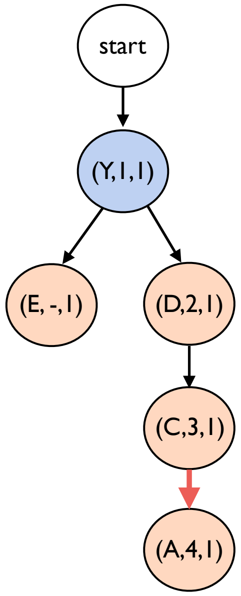


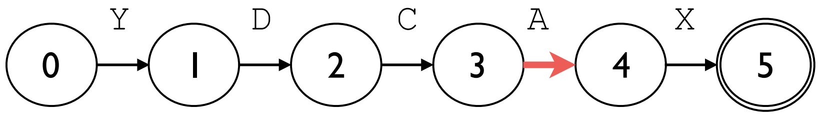


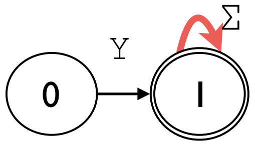


Constructing the Product Graph (PG)

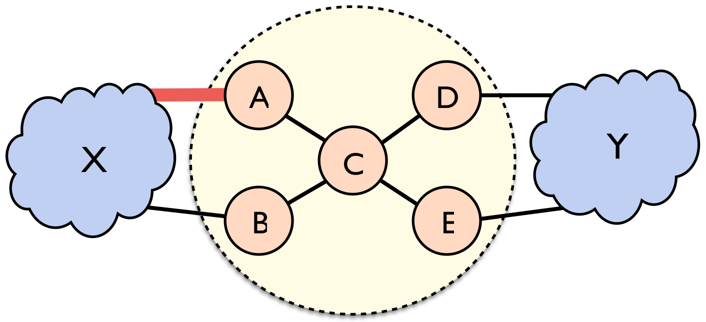


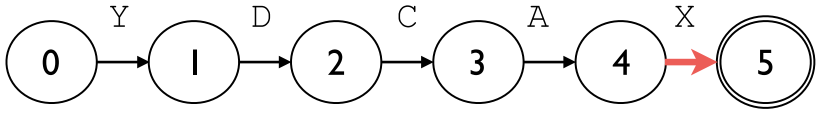


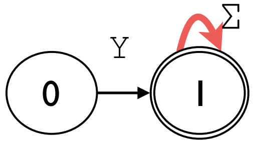


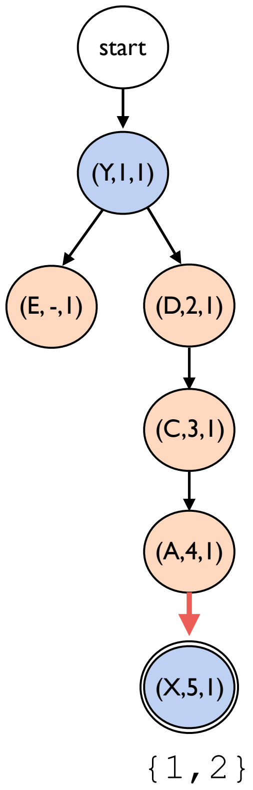


Constructing the Product Graph (PG)

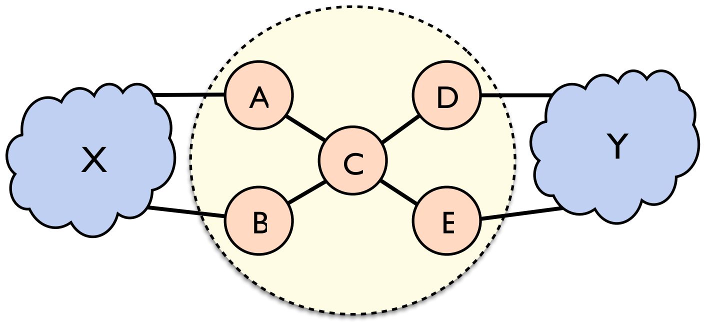


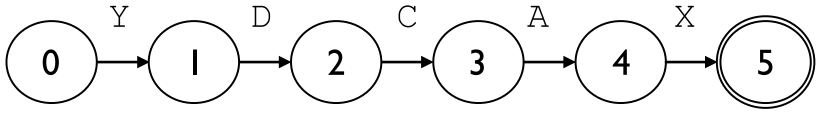


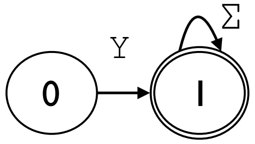


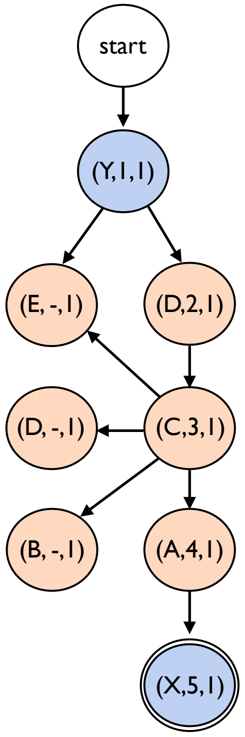


Constructing the Product Graph (PG)



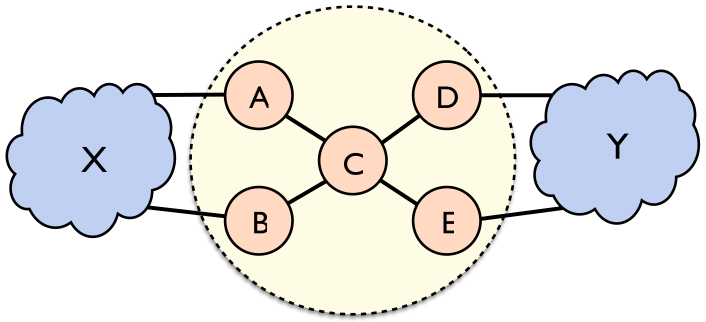


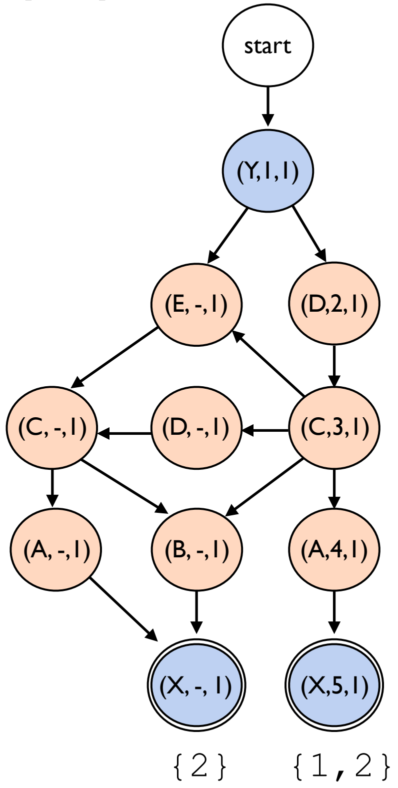


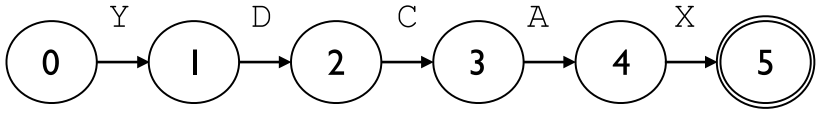


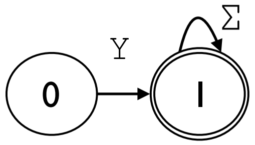
{1,2}

Constructing the Product Graph (PG)

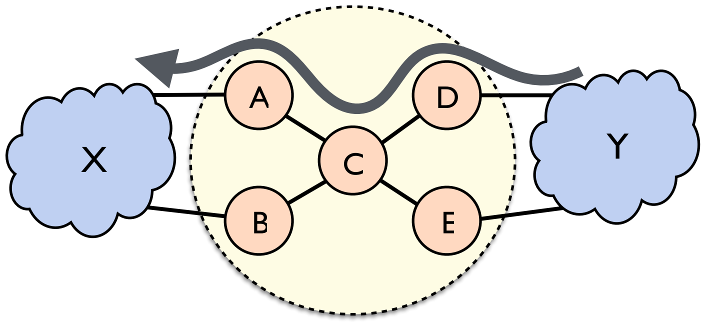


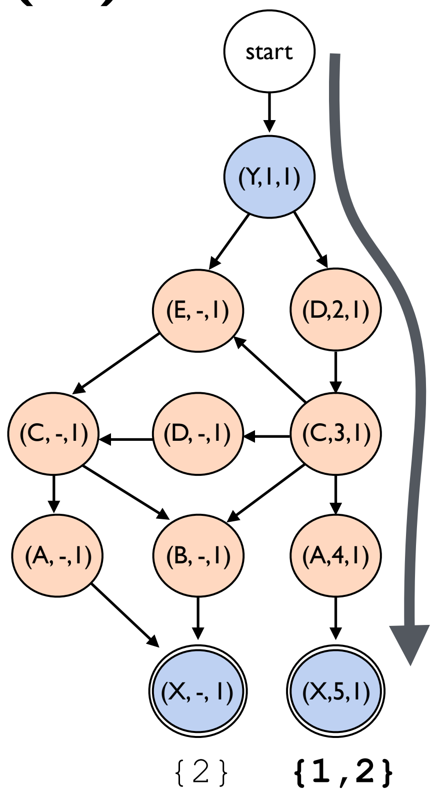


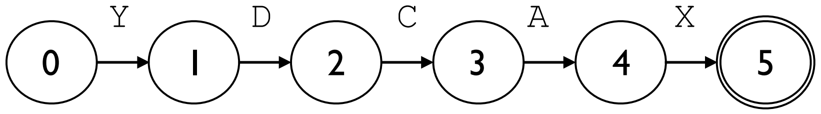


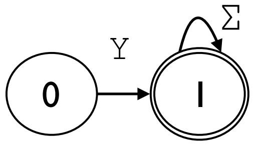


Constructing the Product Graph (PG)



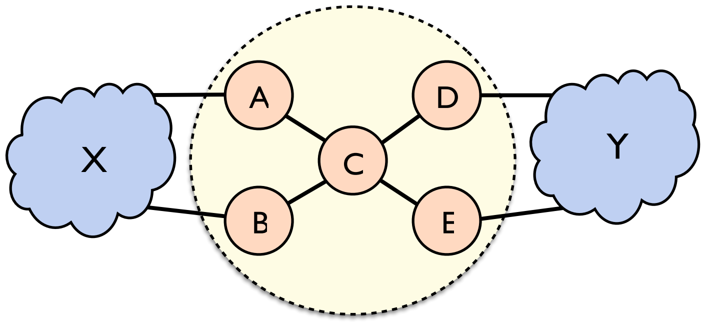


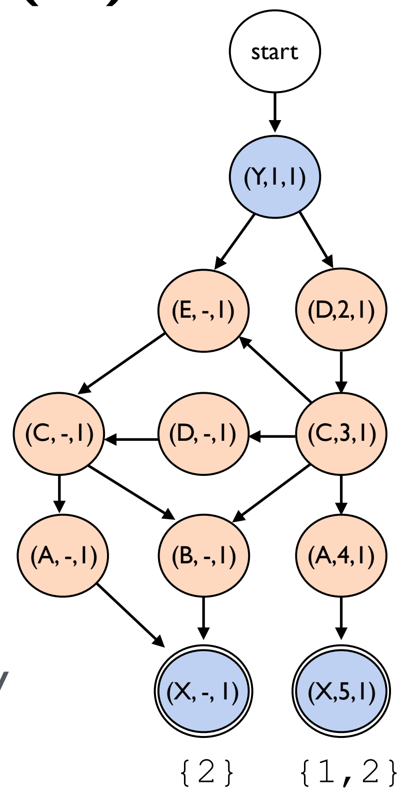




Accept: Y D C A X

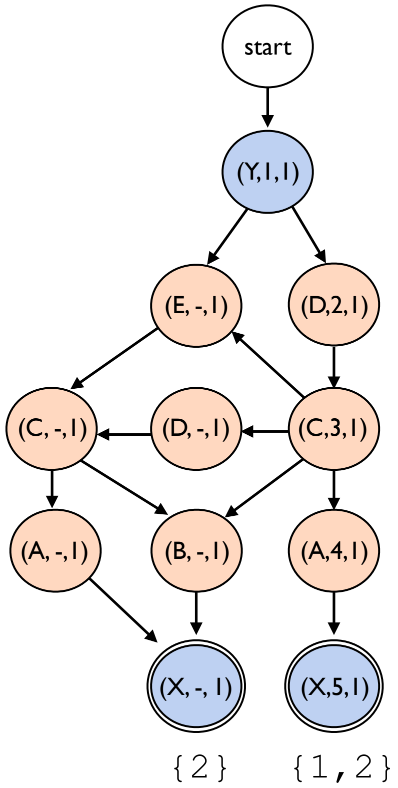
Constructing the Product Graph (PG)





Compact representation of all policy-compliant paths through the topology

Compilation to BGP:



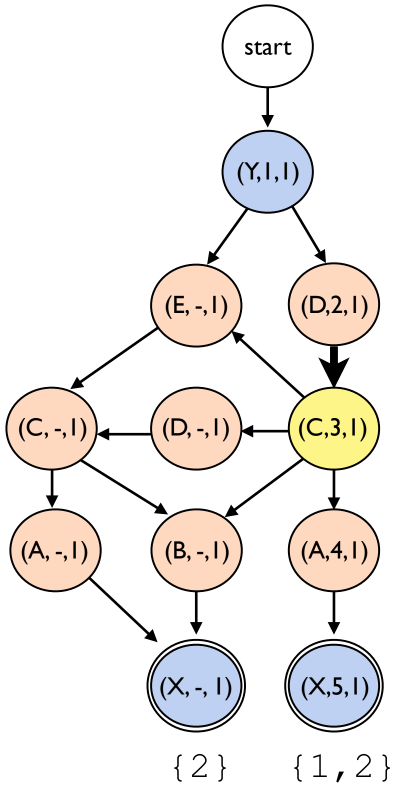
Idea 1: Restrict advertisements to PG edges

- Encode PG state in community tag
- Incoming edges — import filters
- Outgoing edges — export filters

↓

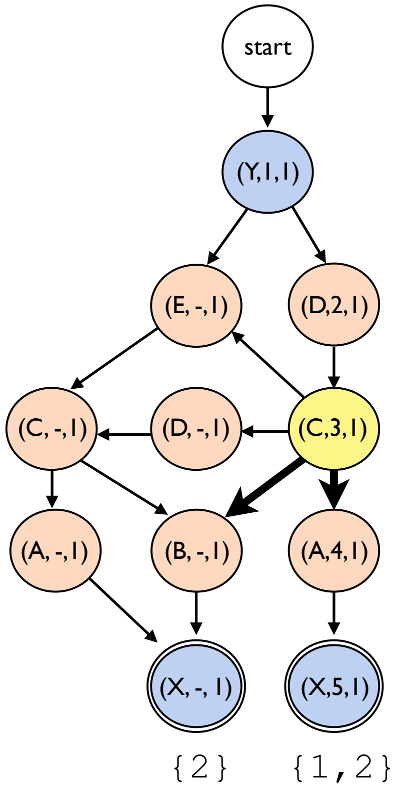
Let BGP find **some allowed** path dynamically

Compilation to BGP:



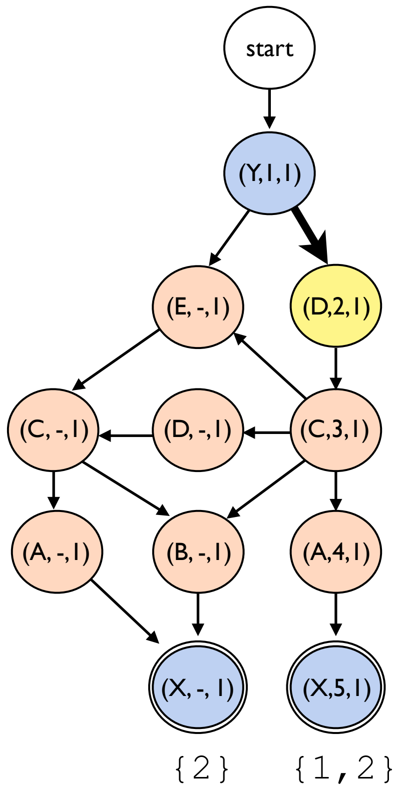
C allows import from D with tag (2,1)

Compilation to BGP:



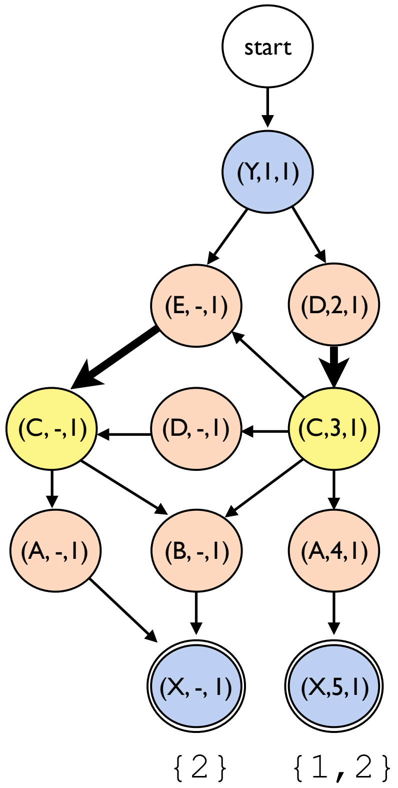
C exports to A,B with tag (3,1)

Compilation to BGP:



BGP regex filter: Y

Compilation to BGP:

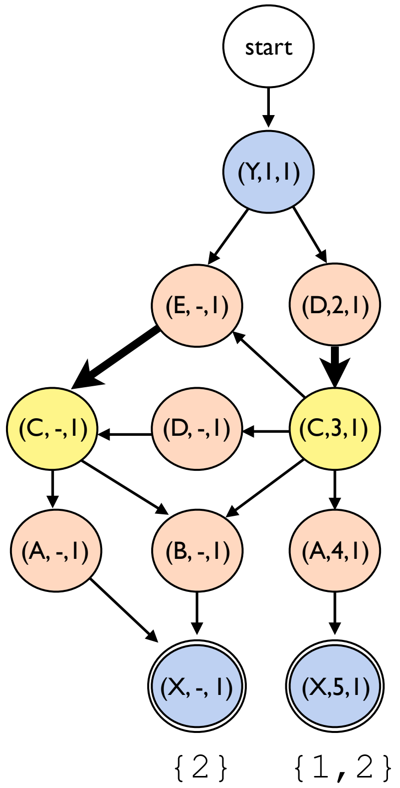


Idea 2: Find preferences

- Direct BGP towards best path
- Under all combinations of failures

Let BGP find **the best** path dynamically

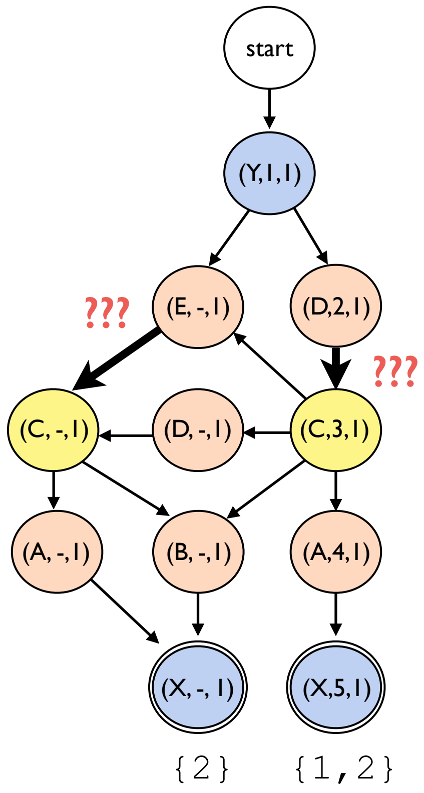
Compilation to BGP:



Router C

```
  match peer = D …

  match peer = E …
```

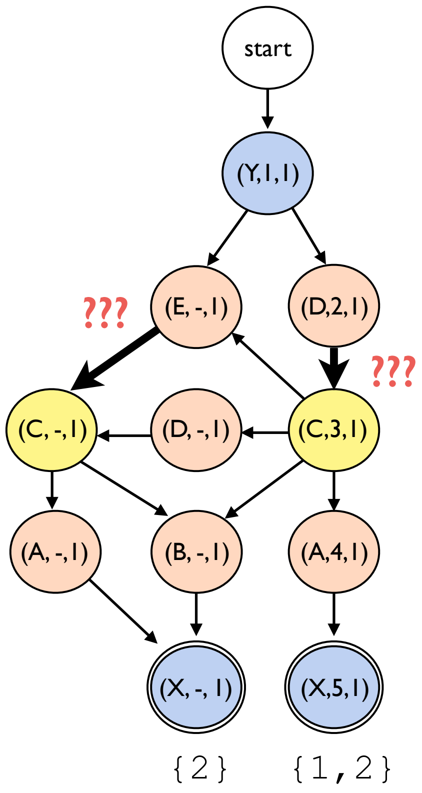
Compilation to BGP:



Router C

```
  match peer = D …
    local-pref ← ???
  match peer = E …
    local-pref ← ???
```

Compilation to BGP:

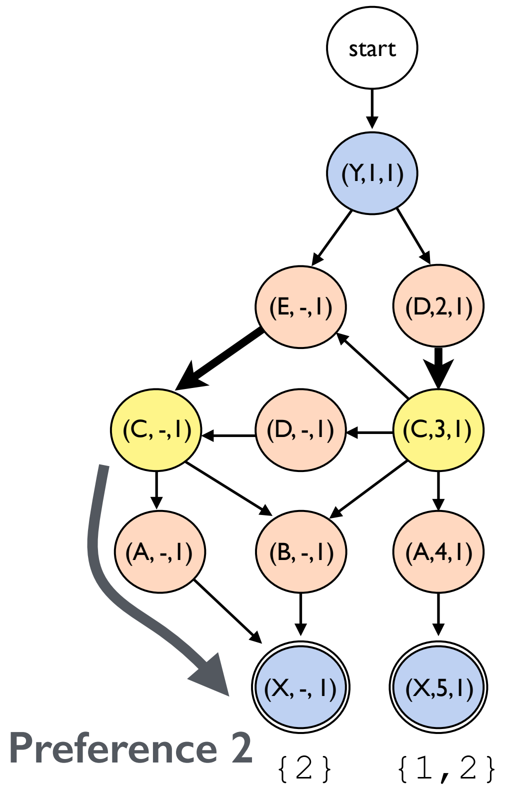


Router C

```
match peer = D …
  local-pref ← ???
match peer = E …
  local-pref ← ???
```

Should we prefer messages from E over D?

Compilation to BGP:

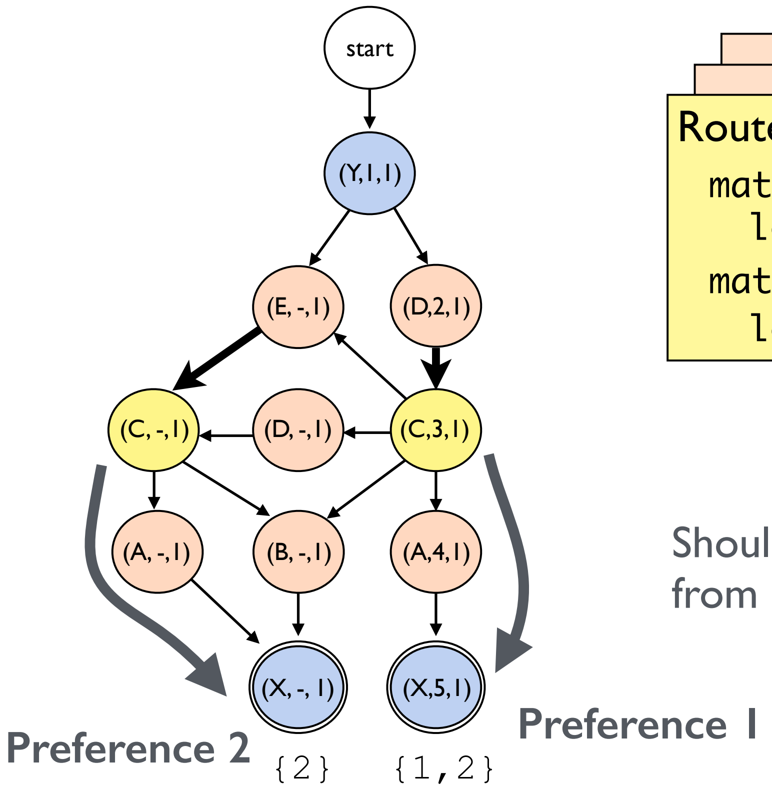


Router C

```
match peer = D …
  local-pref ← ???
match peer = E …
  local-pref ← ???
```

Should we prefer messages from E over D?

Compilation to BGP:

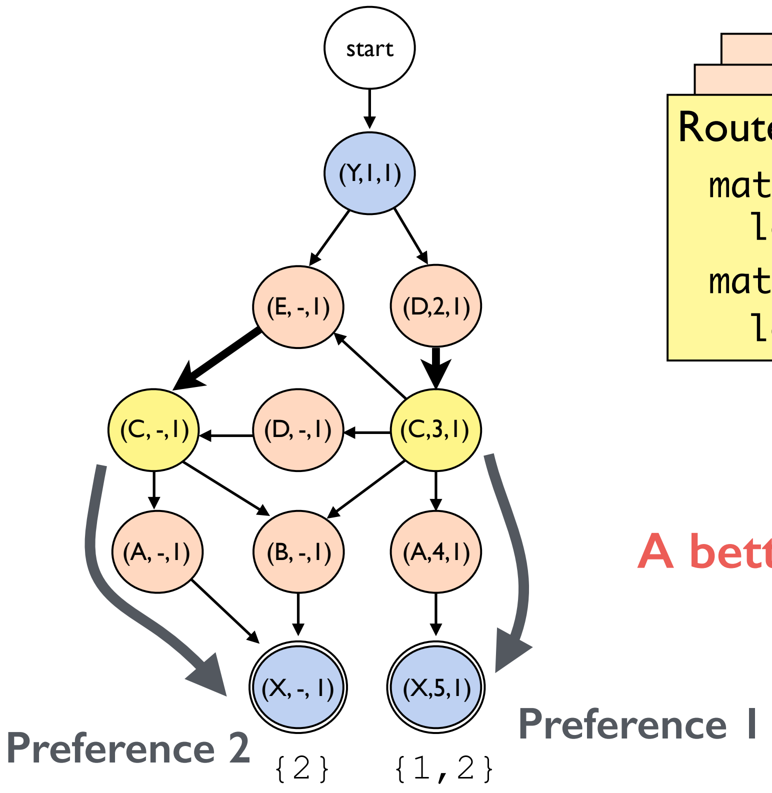


Router C

```
match peer = D …
  local-pref ← ???
match peer = E …
  local-pref ← ???
```

Should we prefer messages from E over D?

Compilation to BGP:

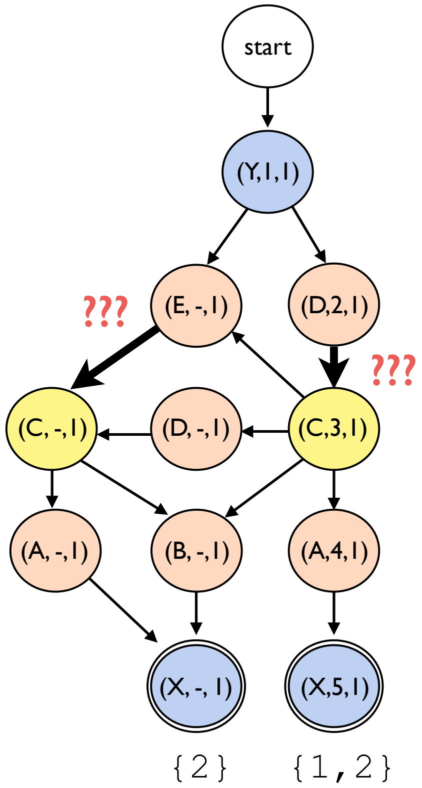


Router C

```
match peer = D …
  local-pref ← ???
match peer = E …
  local-pref ← ???
```

A better path goes unused!

Compilation to BGP:

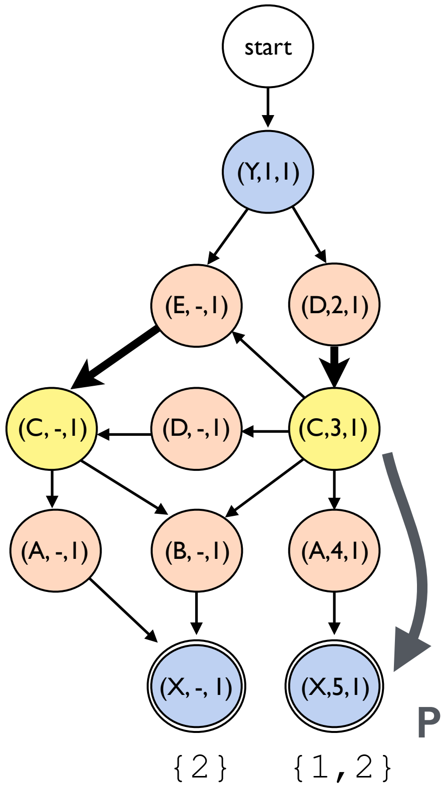


Router C

```
match peer = D …
  local-pref ← ???
match peer = E …
  local-pref ← ???
```

What about preferring messages from D over E?

Compilation to BGP:



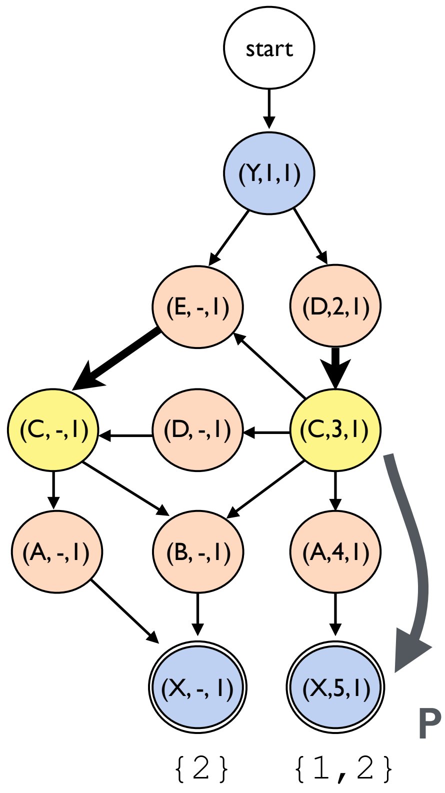
Preference 1

Router C

```
match peer = D …
   local-pref ← ???
match peer = E …
   local-pref ← ???
```

What about preferring messages from D over E?

Compilation to BGP:



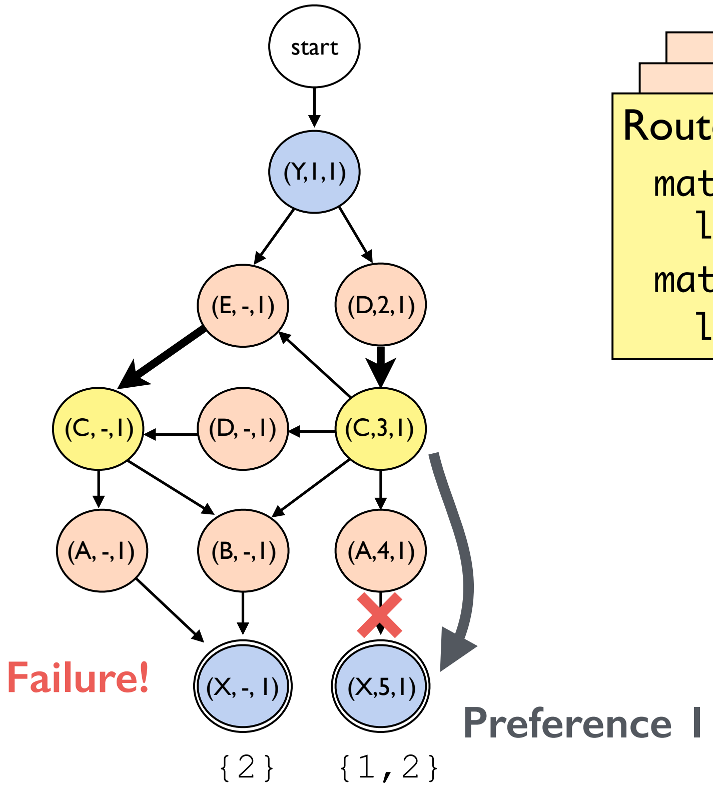
Preference 1

Router C

```
match peer = D …
  local-pref ← ???
match peer = E …
  local-pref ← ???
```

No worse than preferring E

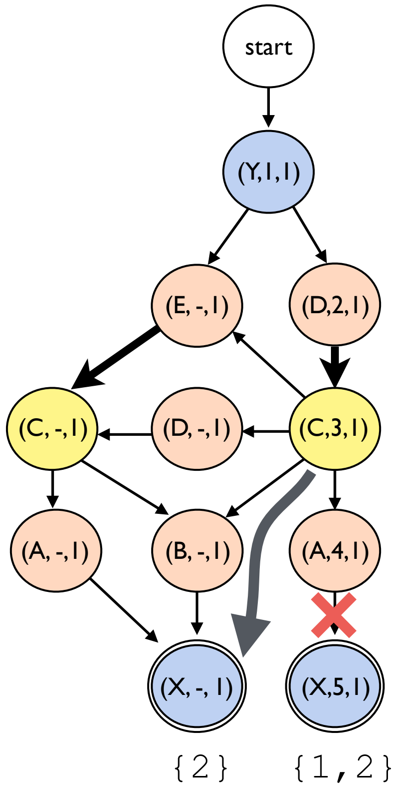
Compilation to BGP:



Router C

```
match peer = D …
  local-pref ← ???
match peer = E …
  local-pref ← ???
```

Compilation to BGP:



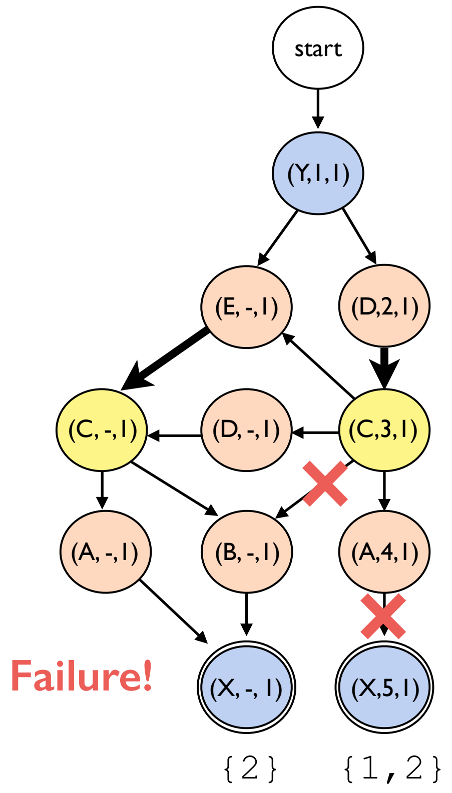
Preference 2

Router C

```
match peer = D …
  local-pref ← ???
match peer = E …
  local-pref ← ???
```

No worse than preferring E

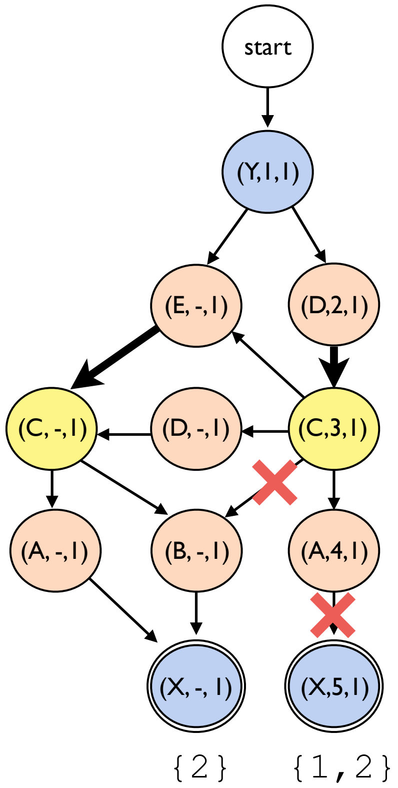
Compilation to BGP:



Router C

```
match peer = D …
  local-pref ← ???
match peer = E …
  local-pref ← ???
```

Compilation to BGP:

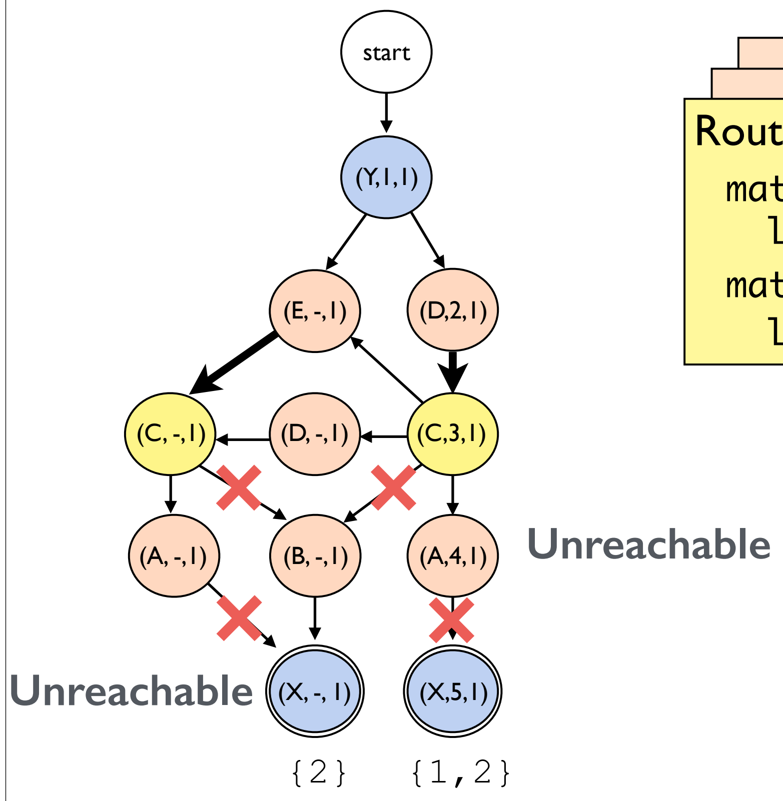


Unreachable

Router C

```
match peer = D …
   local-pref ← ???
match peer = E …
   local-pref ← ???
```

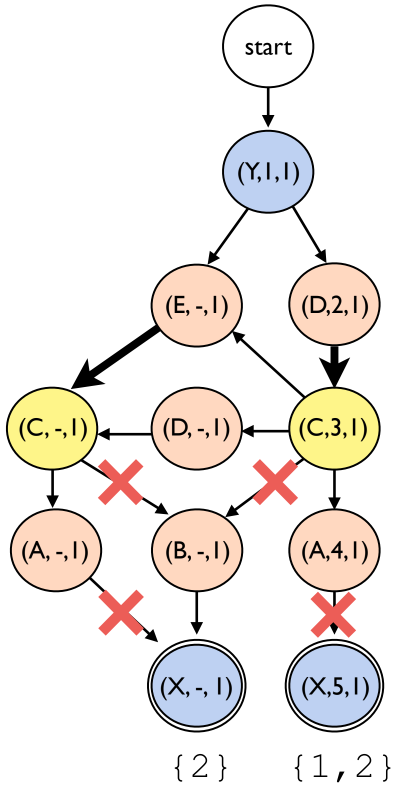
Compilation to BGP:



Router C

```
match peer = D …
  local-pref ← ???
match peer = E …
  local-pref ← ???
```

Compilation to BGP:

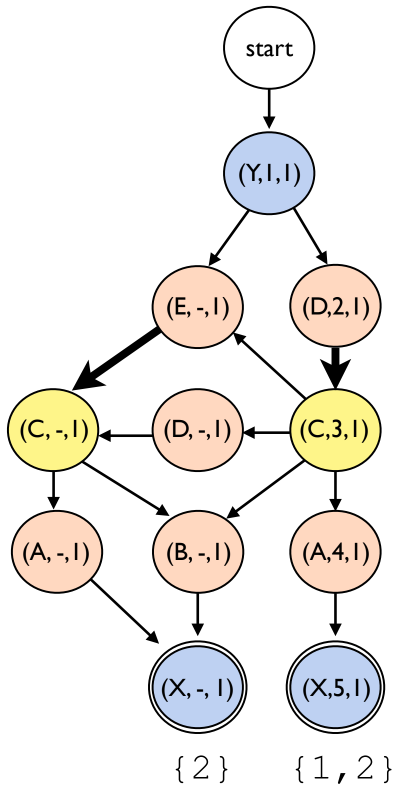


Router C

```
match peer = D …
  local-pref ← ???
match peer = E …
  local-pref ← ???
```

No worse than preferring E

Compilation to BGP:



Router C

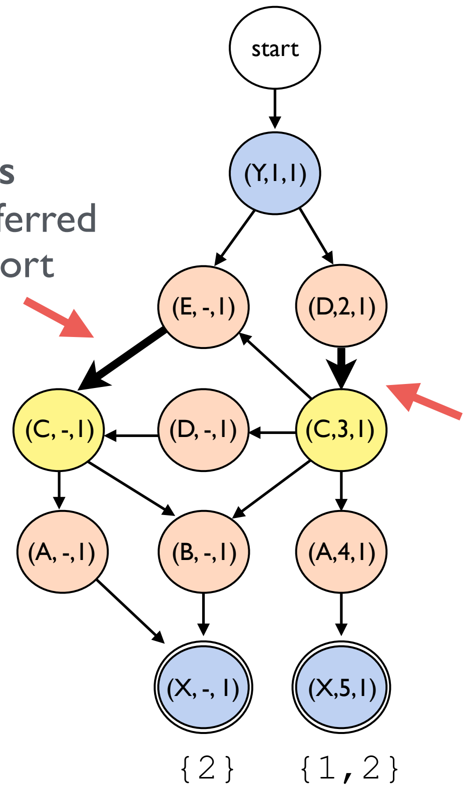
```
match peer = D …
  local-pref ← ???
match peer = E …
  local-pref ← ???
```

All failures checked efficiently by a greedy graph algorithm

See paper for details!

Compilation to BGP:

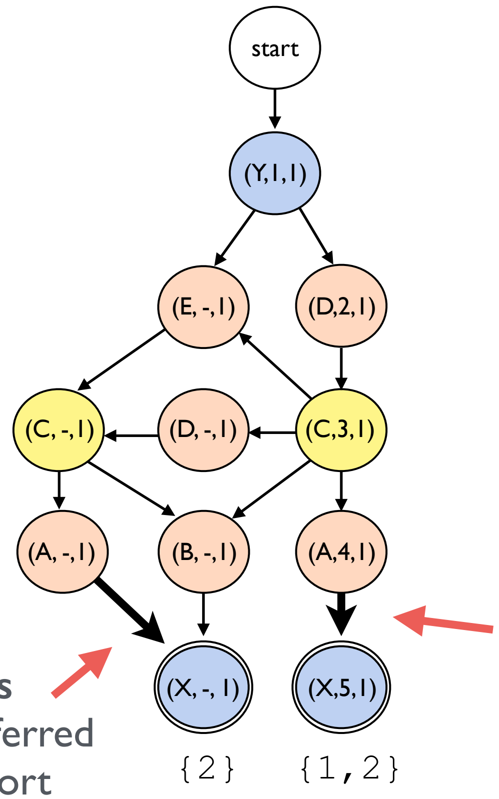
Less preferred import



Implementation: Local preference

More preferred import

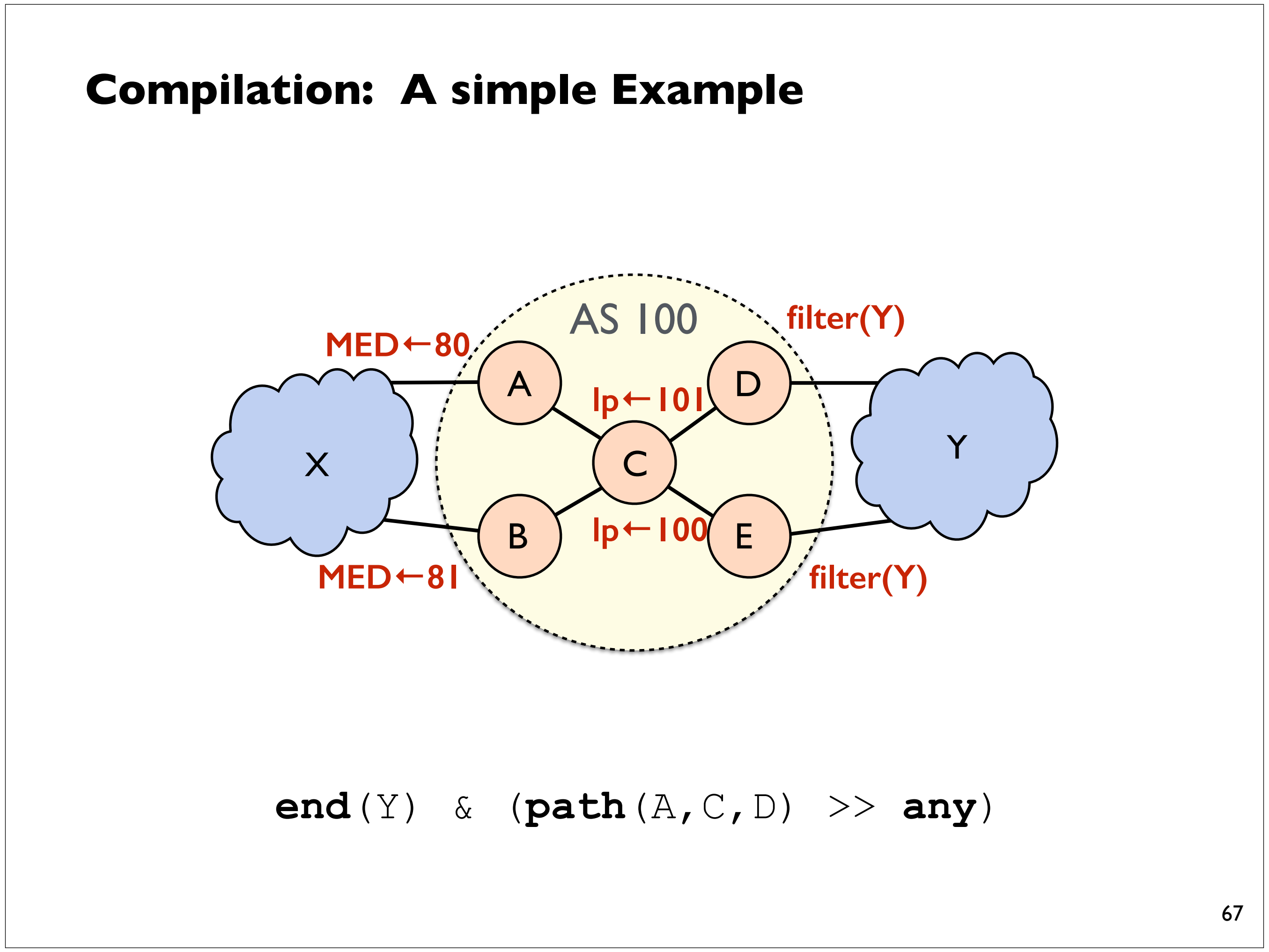
Compilation to BGP:



Implementation: MED/Prepending

More
preferred
import

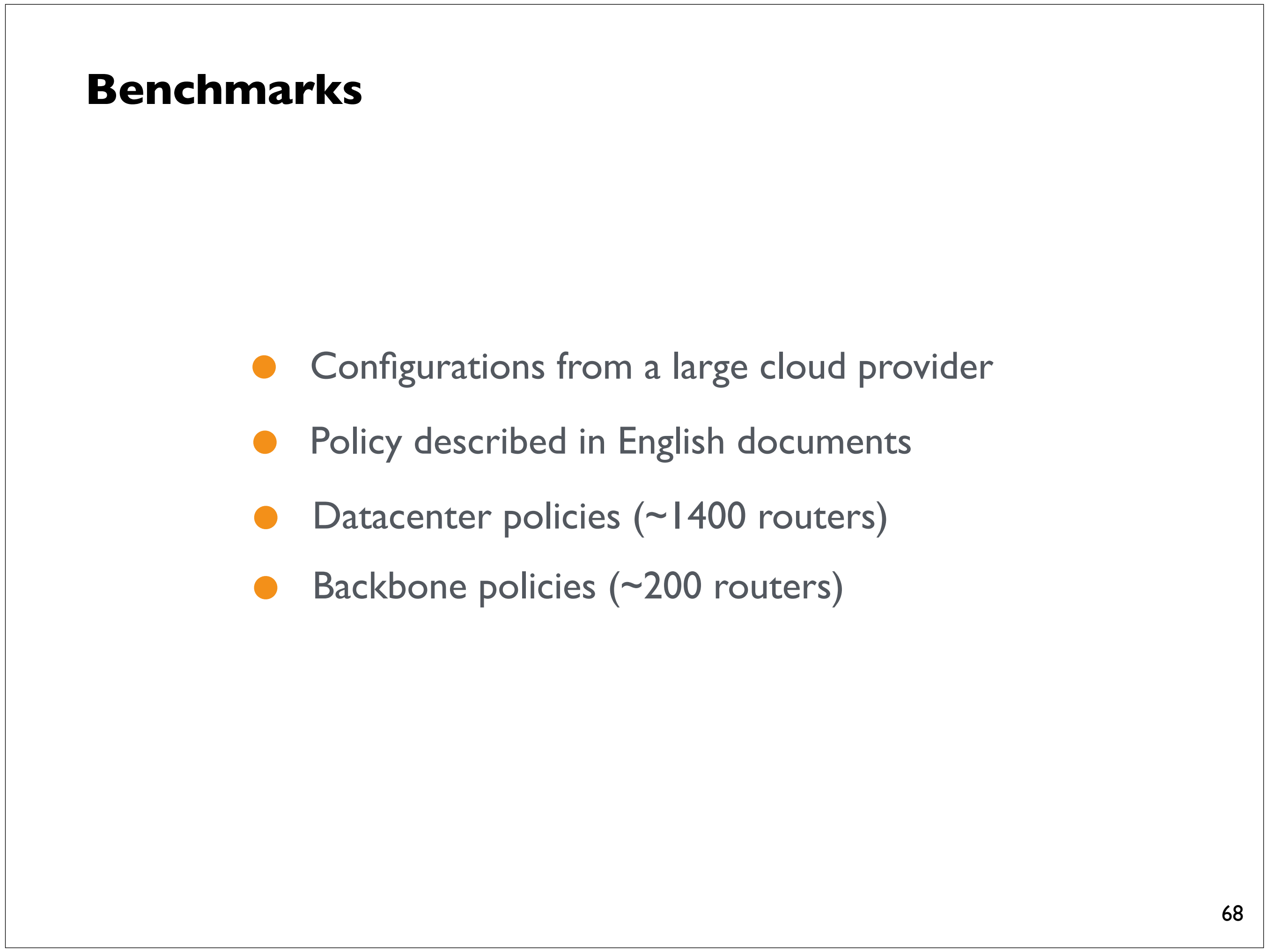
Less
preferred
import



Compilation: A simple Example

```
end(Y) & (path(A,C,D) >> any)
```

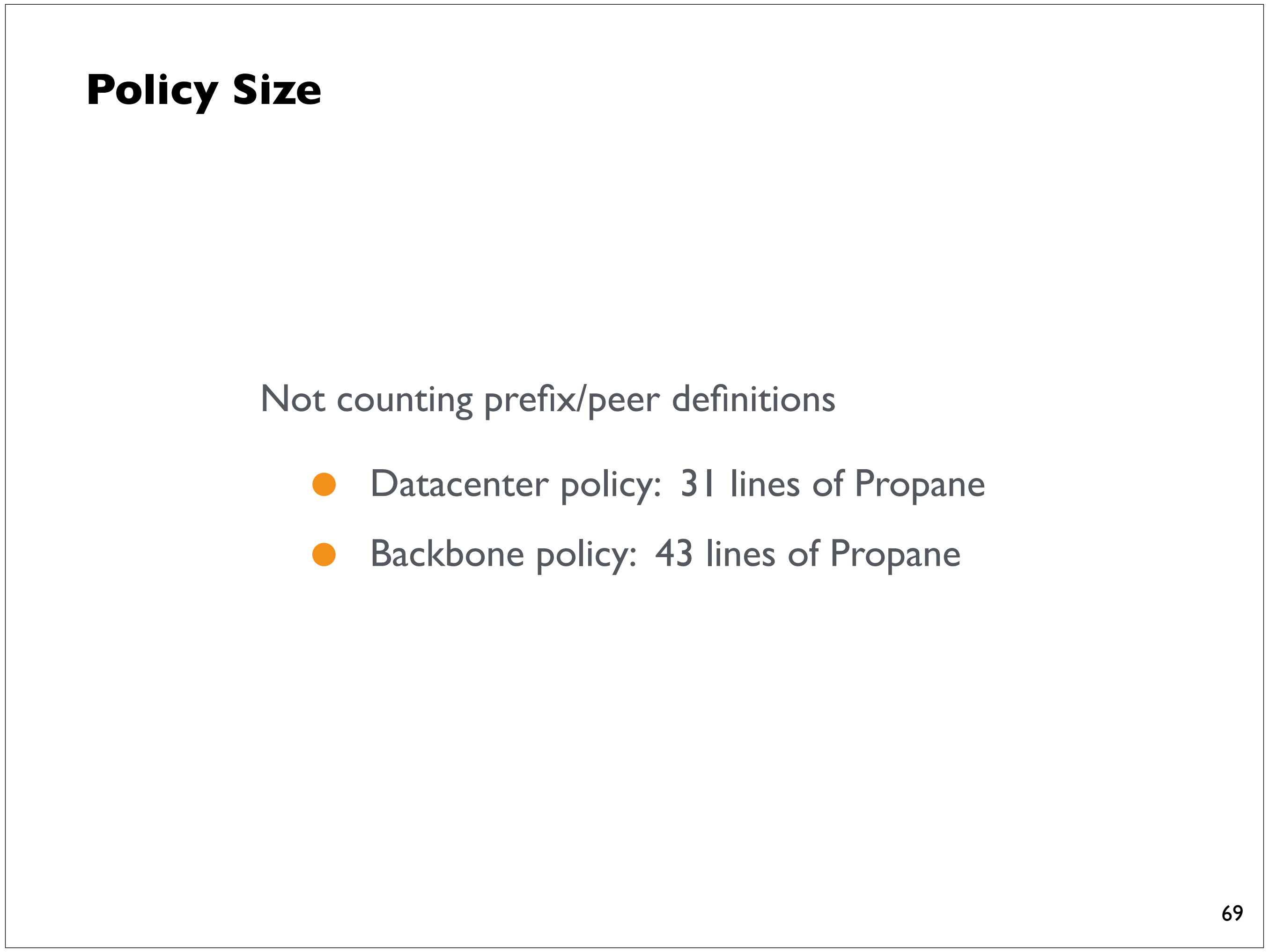
Mention demo & poster here (unless it is before, which I think it might be)



Benchmarks

- Configurations from a large cloud provider
- Policy described in English documents
- Datacenter policies (~1400 routers)
- Backbone policies (~200 routers)

We translated the documents into Propane



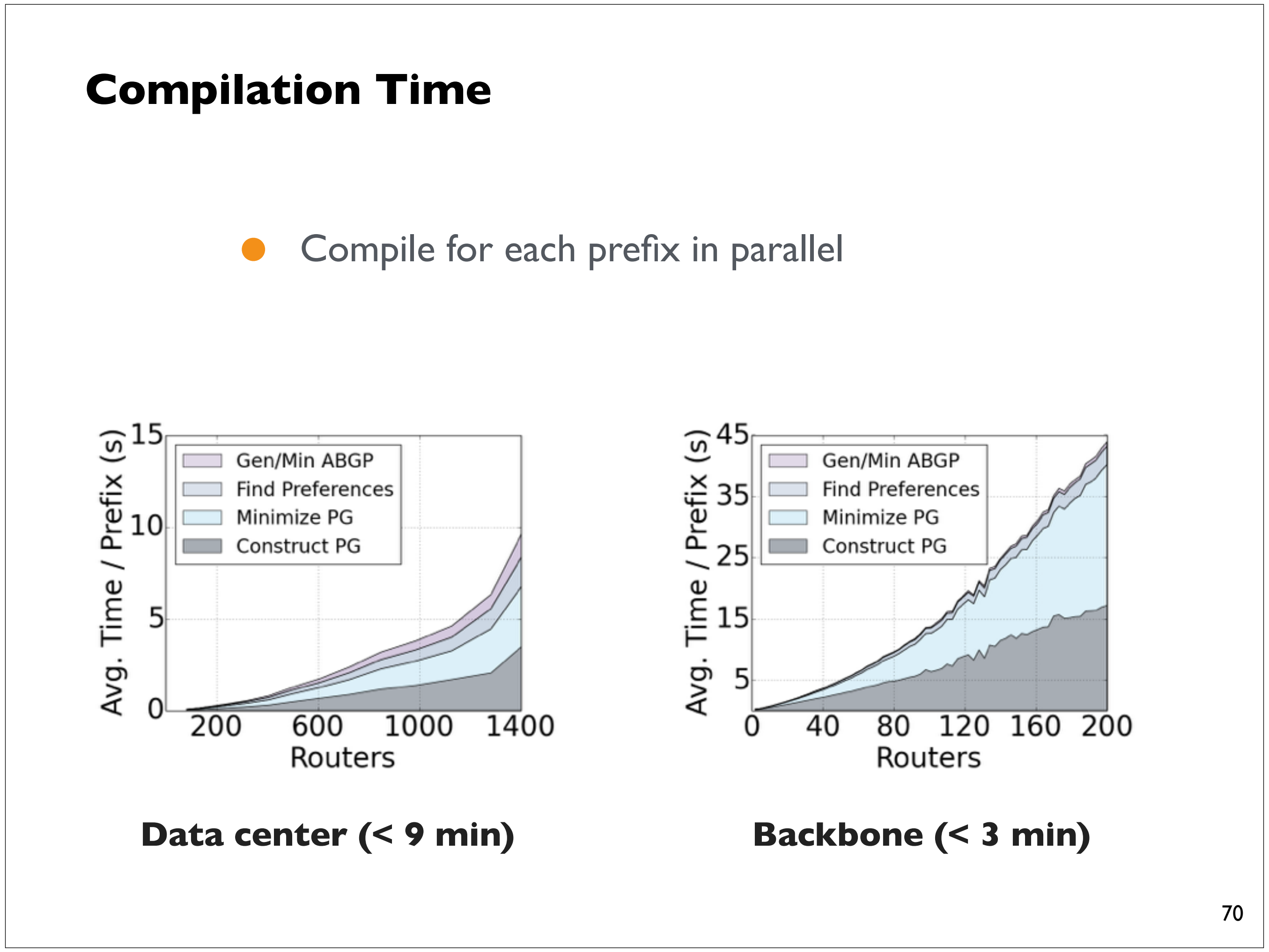
Policy Size

Not counting prefix/peer definitions

- Datacenter policy: 31 lines of Propane
- Backbone policy: 43 lines of Propane

Say something dramatic here (thousands of lines normally), orders of magnitude smaller

Biggest takeaway of the talk



Compilation Time

- Compile for each prefix in parallel

Data center (< 9 min)

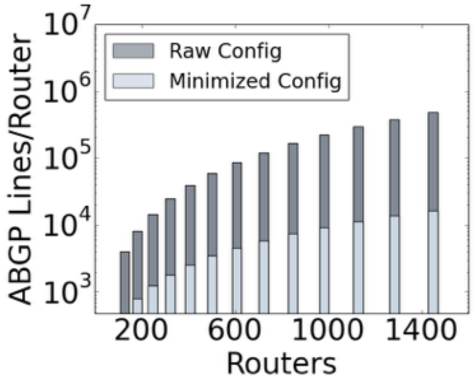
Backbone (< 3 min)

Say what the axes are

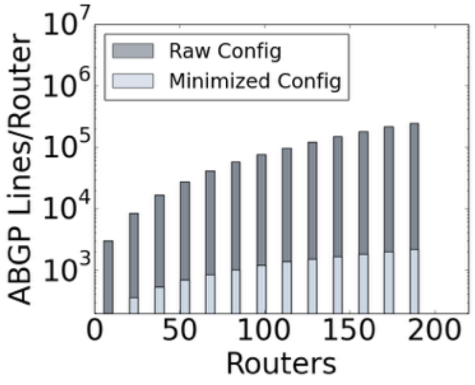
Say what the colors are (can read more about it in the paper)

Configuration Size

- Avoid using community tags when choices unambiguous
- Fall-through elimination of route maps

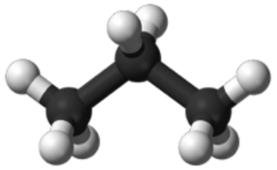


Data center



Backbone

Propane: Summary



High-level language

- Centralized network programmability
- Uniform abstractions for Inter- and Intra-domain routing
- Constraints specify preferred paths and backups in case of failure
- Core policy in 30-50 lines of Propane vs. 1000s

Compiler

- Distributed implementation via BGP
- Generates filters, preferences, community values, MEDs, etc.
- Static analysis guarantees policy compliance for all failures
- Scales to reasonably sized network topologies