

# Bridging centralized programming and distributed control planes



**Ryan Beckett**



**Ratul Mahajan**



**Todd Millstein**

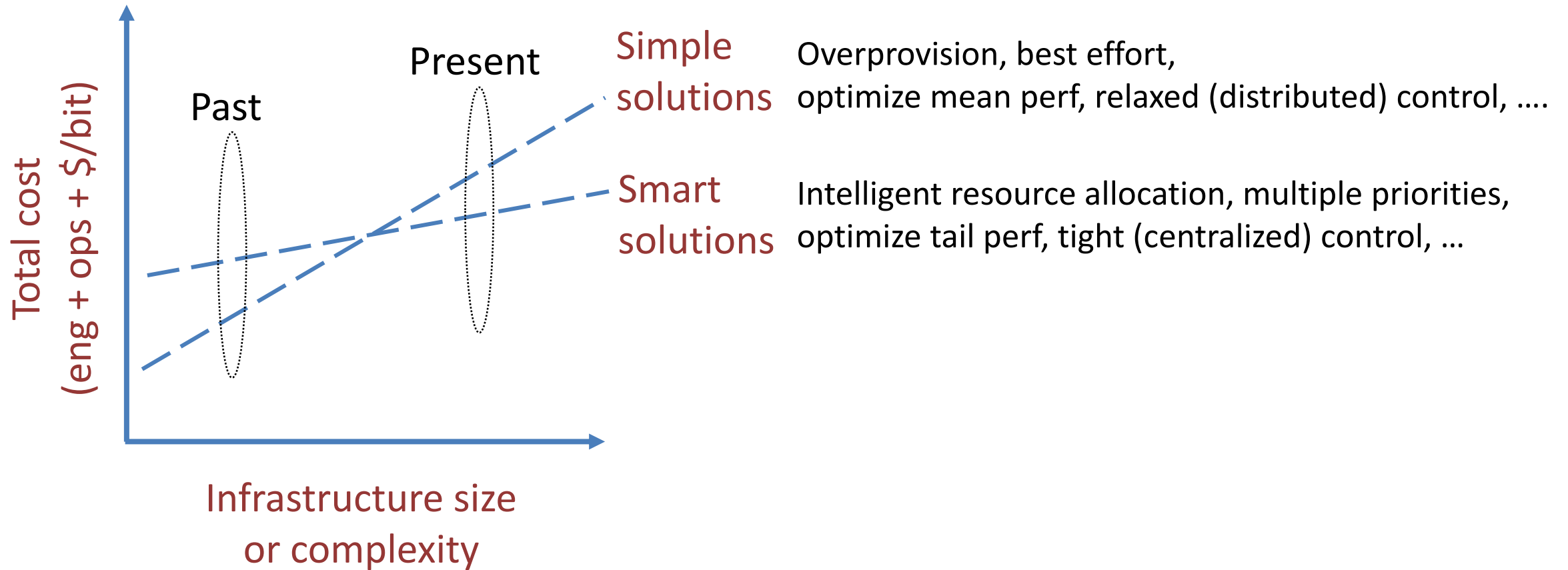


**Jitu Padhye**



**David Walker**

# Hypothesis on smart solutions in networks



# Network programming journey

	Distributed programming	Centralized programming
Distributed control plane	<div>+ Resilience</div> <div>– Programmability</div>	
Centralized Control plane		

# Programming (configuring) networks is error-prone

2/5/2016

China routing snafu briefly mangles internet

Log in | Sign up

GET YOUR SMOKE ONLY YOU CAN PREVENT WILDFIRE

DATA CENTER SOFTWARE NETWORKS SECURITY INFRASTRUCTURE DEV

Networks > Broadband

China routing snafu briefly mangles internet

Cockup, not conspiracy

9 Apr 2010 at 12:24, John Leyden

Bad routing information sourced from China has disrupted the internet for the second time in a week. Global BGP (Border Gateway Routing) lookup tables sucked in data from a small Chinese telecommunications company, apparently accidentally broadcast by state-owned carrier China Telecom. The company, IDG reports, ISPs including AT&T, France Telecom, Level3, Qwest and Telefonica accepted ill-thought out traffic routes as a result of the incident. BGP is a core routing protocol which maps options for the best available routes to the net. Several routing options are normally included. The China BGP incident is equivalent of TomTom publishing routes via Shanghai for motorists looking for a shortcut between London and Paris. IDC China Telecommunication published ill-conceived routes for between 32,000 about 10 per cent of the net - instead of the normal 40 or so routes, and this info viable routing options by many service providers for about 20 minutes early on Tuesday (time) after China Telecommunications republished it and before the mix-up was over. Asia would have been more likely to adopt the false routes as potentially viable, but the incident were recorded all over the world. BGPmon.net, a BGP monitoring service, has a detailed technical write-up of the incident described as a prefix hijack, here. Although it seems they [IDC China Telecommunication] have leaked a write-up about 10 per cent of these prefixes propagated outside of the Chinese network, it includes prefixes for popular websites such as dell.com, cnn.com, www.scoop.intel.com, www.rapidshare.com and www.geocities.jp. A large number of networks impacted this morning were actually Chinese. These include some popular Chinese website such as www.joy.cn, www.pconline.com.cn, www.huanqiu.com, www.tianya.cn and www.china2.com. A cock-up is suspected, rather than a conspiracy, at least by BGPmon.net. Given the large number of prefixes and short interval I don't believe this is a hijack. Most likely it's because of configuration issue, i.e. fat fingers. But speculation. The practical consequences of the screw-up are still being assessed but it could have dropped connections or, worse, traffic routed through unknown systems in China. One of the clearest illustrations of the security shortcomings of BGP, a somewhat nonetheless important network protocol. The China BGP global routing represents a rare but not unprecedented mix-up in network management. For example, just two weeks ago bad routing data resulted in the misrouting of internet traffic through a DNS (Domain Name System) server in China, as explained by a network monitoring firm Renesys here. Bad BGP routing information led to a major outage of the UK's largest search engine, as reported here. http://www.theregister.co.uk/2010/04/09/china\_bgp\_internetweb\_snafu/

2/5/2016

Internet-Wide Catastrophe—Last Year - Dyn Research | The New Home Of Renesys

Search

Dyn Research

THE NEW HOME OF renesys

HOME TOPICS PRESENTATION ABOUT

DECEMBER 24, 2005 COMMENTS (0) VIEWS: 3038

ENGINEERING TODD UNDERWOOD

Internet-Wide Catastrophe—Last Year

One year ago today TTNNet in Turkey (AS9121) pretended to be the entire Internet. And unfortunately for the rest of the Internet, many large network providers believed them (or at least believed them in part). As far as anyone knows, it was a mistake, not a malicious act. But the consequences were far from benign: for several hours a large number of Internet sites were unable to reach a large number of Internet sites. We can take a look at what happened, and whether it was the intervening time. In the morning 2004, TTNNet (AS9121) started announcing itself as the entire Internet. This was a major misconfiguration. The result was that traffic to many large websites was routed through TTNNet, which was not equipped to handle the traffic. This led to a widespread outage of many large websites, including Google, Yahoo, and Amazon. The outage lasted for several hours before being resolved. This incident highlighted the importance of proper BGP configuration and the potential for catastrophic failures in the Internet routing system.

https://www.thousandeyes.com/

Product Solutions Customers https://www.thousandeyes.com/customers/ About Login https://app.thousandeyes.com/ https://www.thousandeyes.com/search/ Sign Up https://www.thousandeyes.com/signup/

Blog Home 10

Time Warner Cable Outage Causes Widespread Routing and DNS Impacts

Posted by Edye Anderson https://blog.thousandeyes.com/author/edye/ on August 28, 2014

By now a lot of you have probably read about the Time Warner Cable (TWC) outage on August 27th. Yesterday morning I was greeted with a slew of alarms, names that wouldn't resolve, websites that wouldn't load and home office employees without any Internet access. It hadn't hit the news yet, but I could sense that a major outage was occurring and quickly opened up the ThousandEyes platform to get a handle on the situation.

Time Warner Outage

The alerts started coming in a little before 930 UTC (5:30 Eastern). I observed several different issues including inaccessible websites, DNS names failing to resolve, BGP reachability issues and agents losing access to the Internet. Companies that peer with Time Warner experienced degraded HTTP availability, affecting critical services such as supply chain portals (Figure 1).

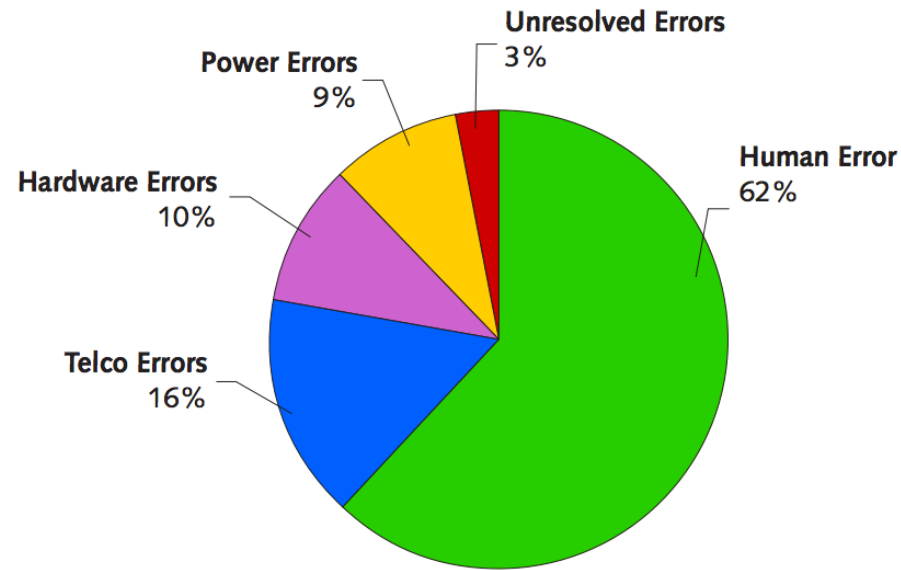


Figure 1. Supply chain portal with limited availability during Time Warner Cable outage.

I could see right away that users and networks that connect through Time Warner were unable to reach the supply chain portal, indicating the issue was in the Time Warner network. In this case, I'd expect to see a brief service interruption while all the traffic re-routed through their other upstream ISP AT&T. However, the availability issues continued for the entire duration of the outage. I was surprised to still see issues, so I took a look at the path visualization view to figure out exactly where traffic was getting dropped on the way to this site. Normally, two locations (Tokyo and Dallas) transit Road Runner (Time Warner) to reach this supply chain portal, while the rest go through AT&T (Figure 2).

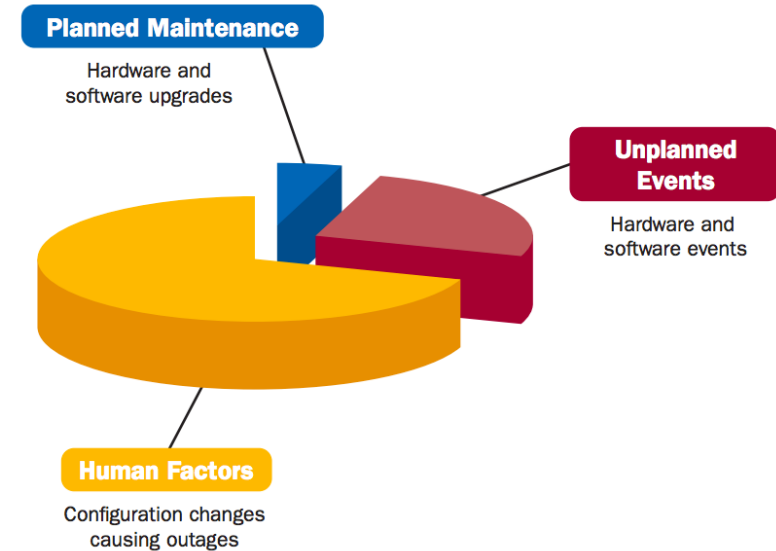
1/4

# Programming (configuring) networks is error-prone



60% of network downtime is caused by human error

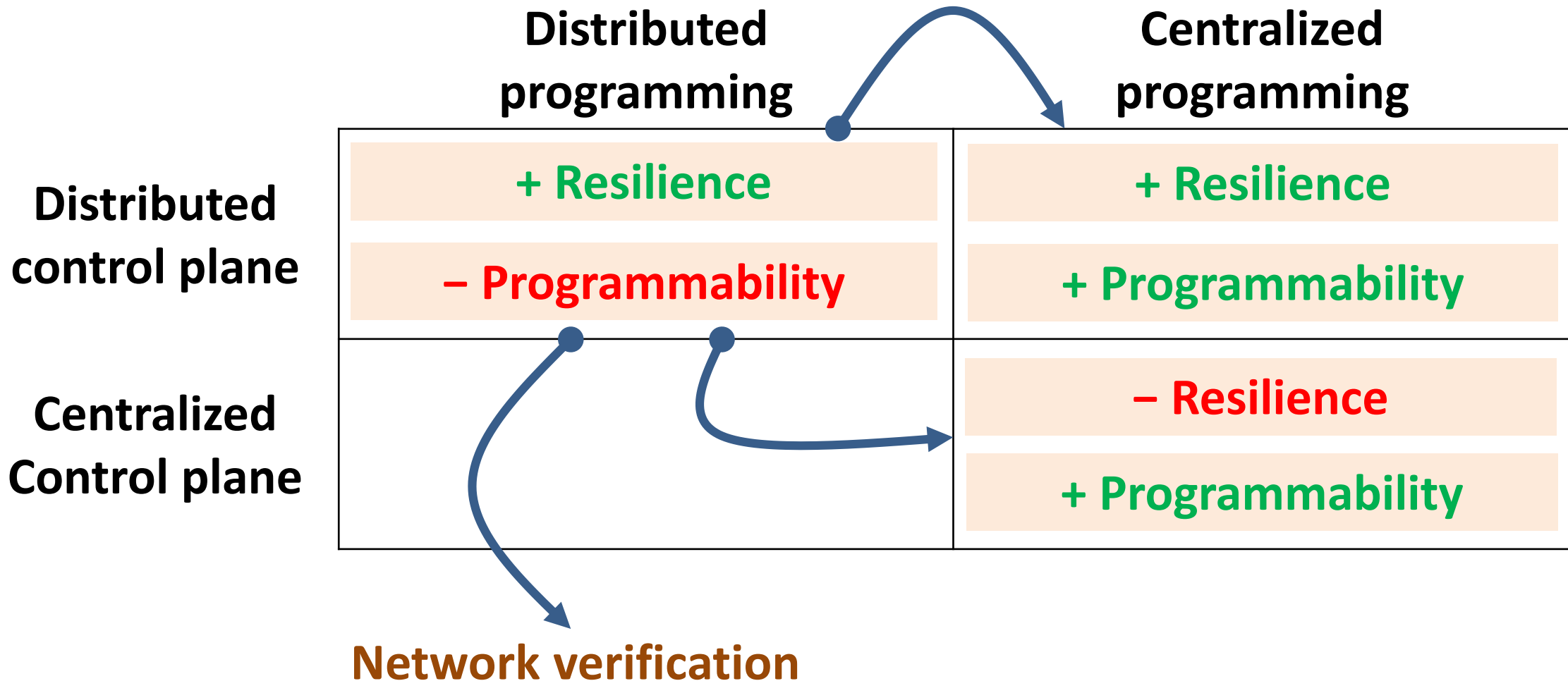
-Yankee group 2002



50-80% of outages are the result of human error

-Juniper 2008

# Network programming journey



# Programming distributed control planes is hard

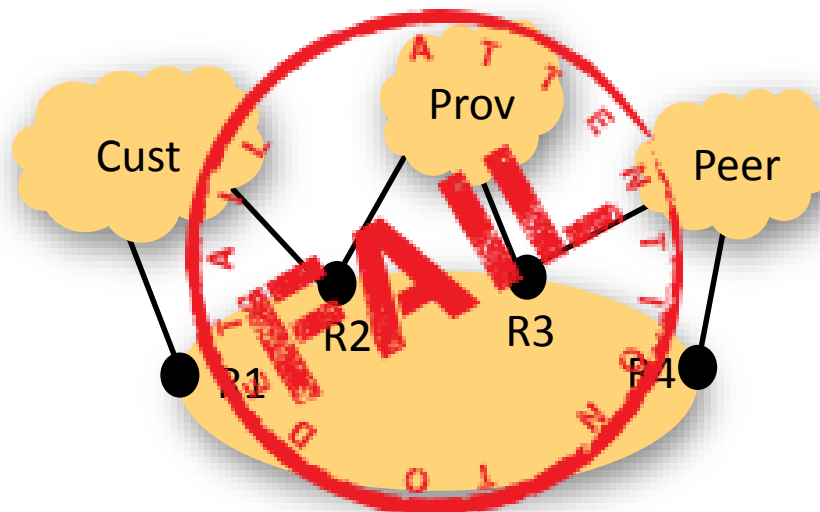
## Network-wide policies

- Prefer one neighbor over another
- Don't use my network as transit
- Keep traffic within a region
- Aggregate prefixes externally

MIND THE GAP

## Router-level mechanisms

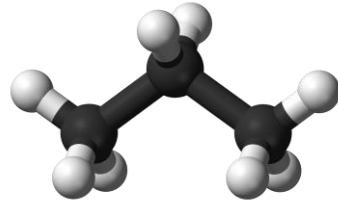
- Set consistent, per-link preferences
- Tag incoming routing info
- Program import and export filters based on various route attributes



# Propane: Centrally programming distributed control planes

A **language** for expressing of network-level objectives

- Path constraints and **relative preferences** (fallbacks)

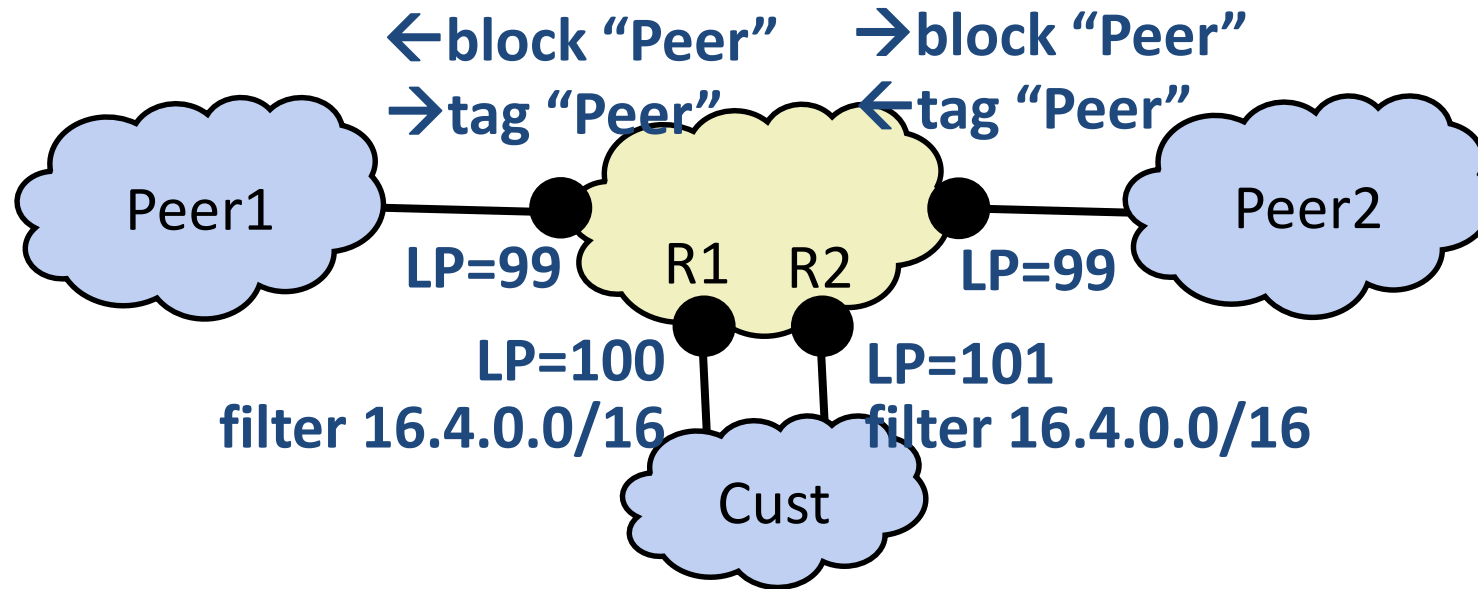


A **compiler** that configures router-level mechanisms

- Configurations are **policy-compliant under all failures**



# Example #1: A backbone network



## Goals

- No transit between peers
- Prefer  $R2 > R1 > \text{Peer}\{1,2\}$
- Limit Cust to 16.4.0.0/16

```
define notransit = {true => not transit({Peer1, Peer2})}  
define preference = {true => exit (R2>R1>{Peer1, Peer2})}  
define ownership = {16.4.0.0/16 => end(Cust)}  
define main = notransit and preference and ownership
```

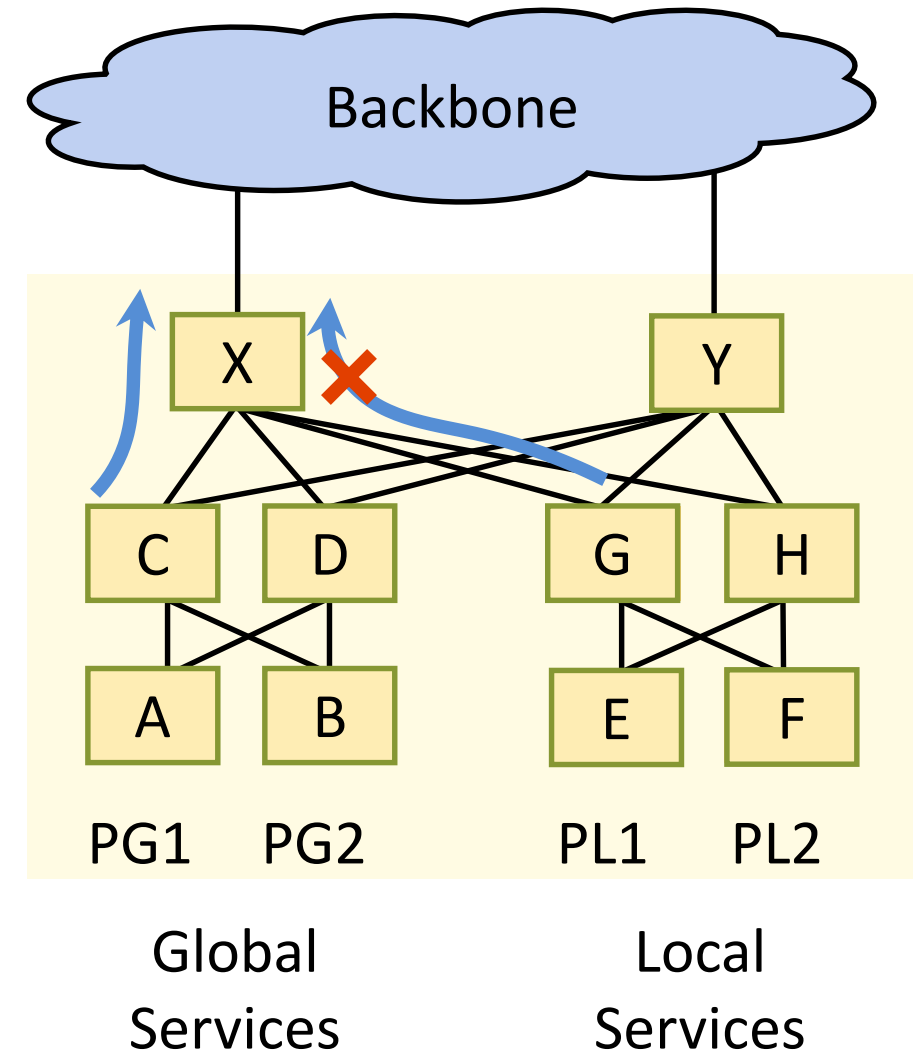
# Example #2: A data center network

## Goals

- Keep local prefixes internal
- Aggregate global prefixes as PG

## Attempt #1

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



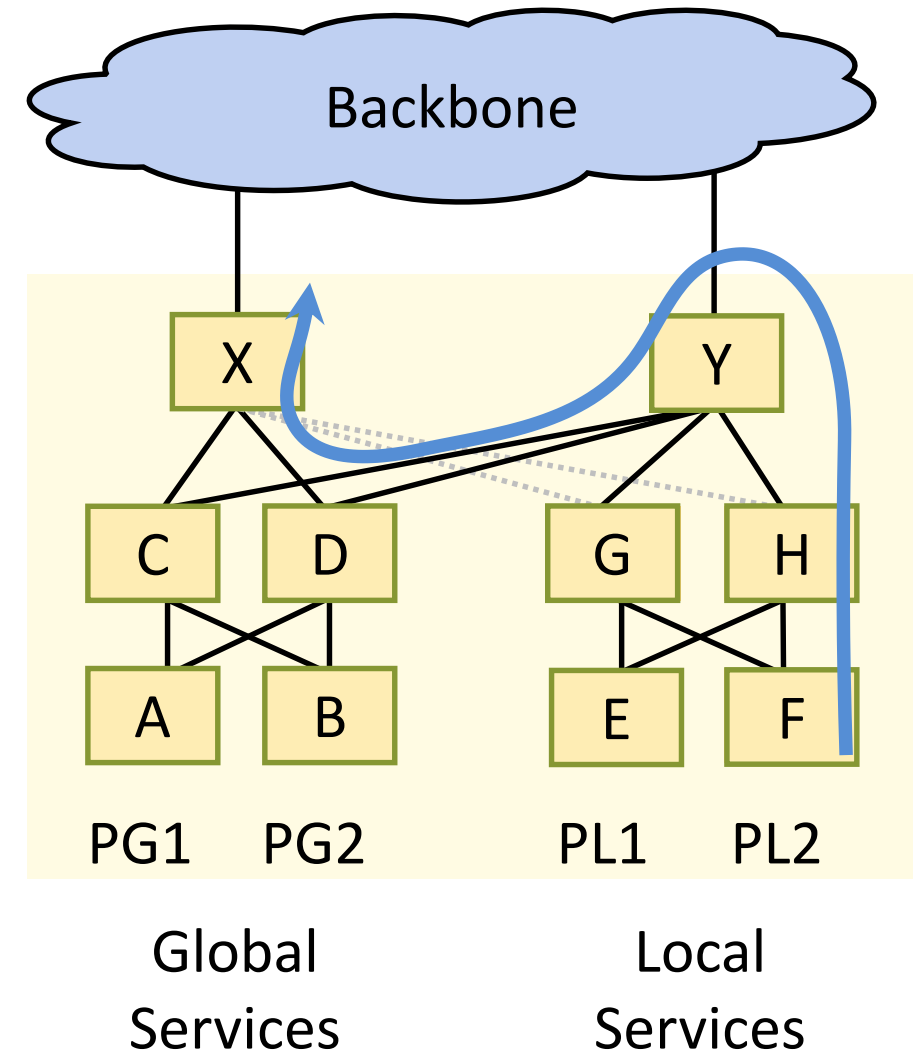
# Example #2: A data center network

## Goals

- Keep local prefixes internal
- Aggregate global prefixes as PG

## Attempt #1

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



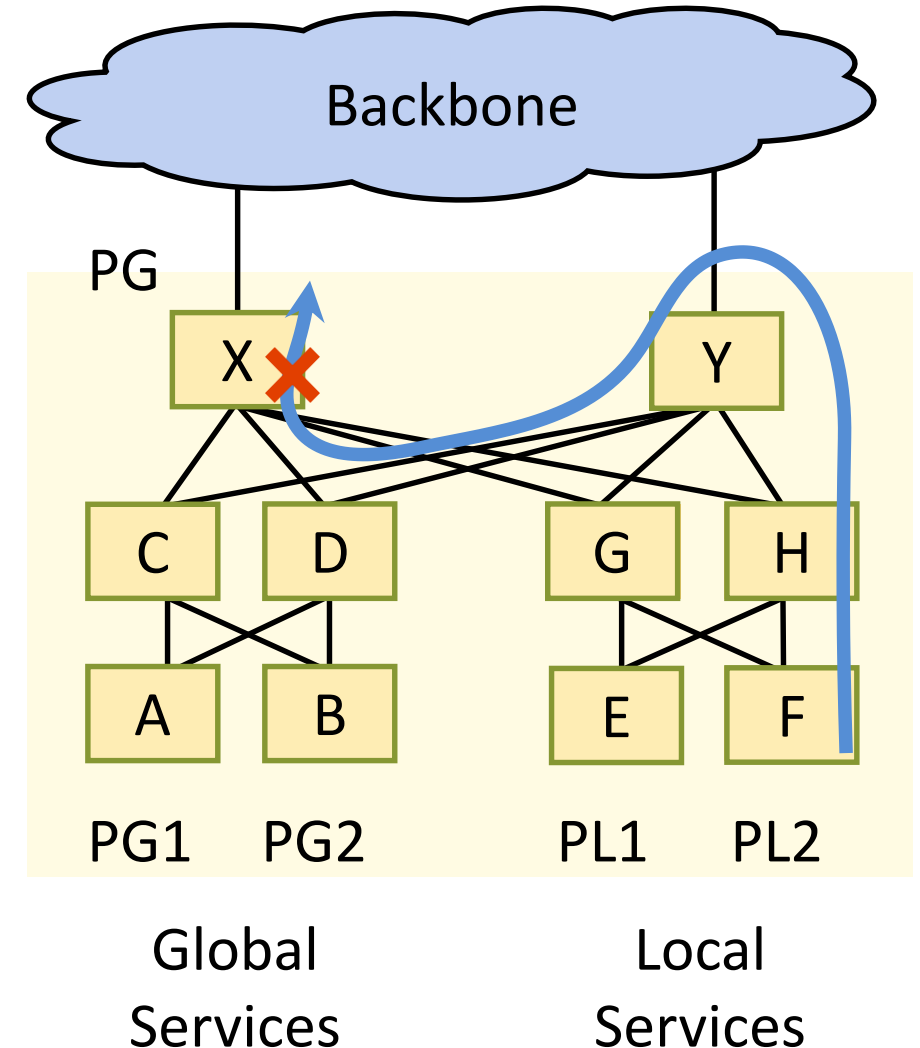
# Example #2: A data center network

## Goals

- Keep local prefixes internal
- Aggregate global prefixes as PG

## Attempt #2

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



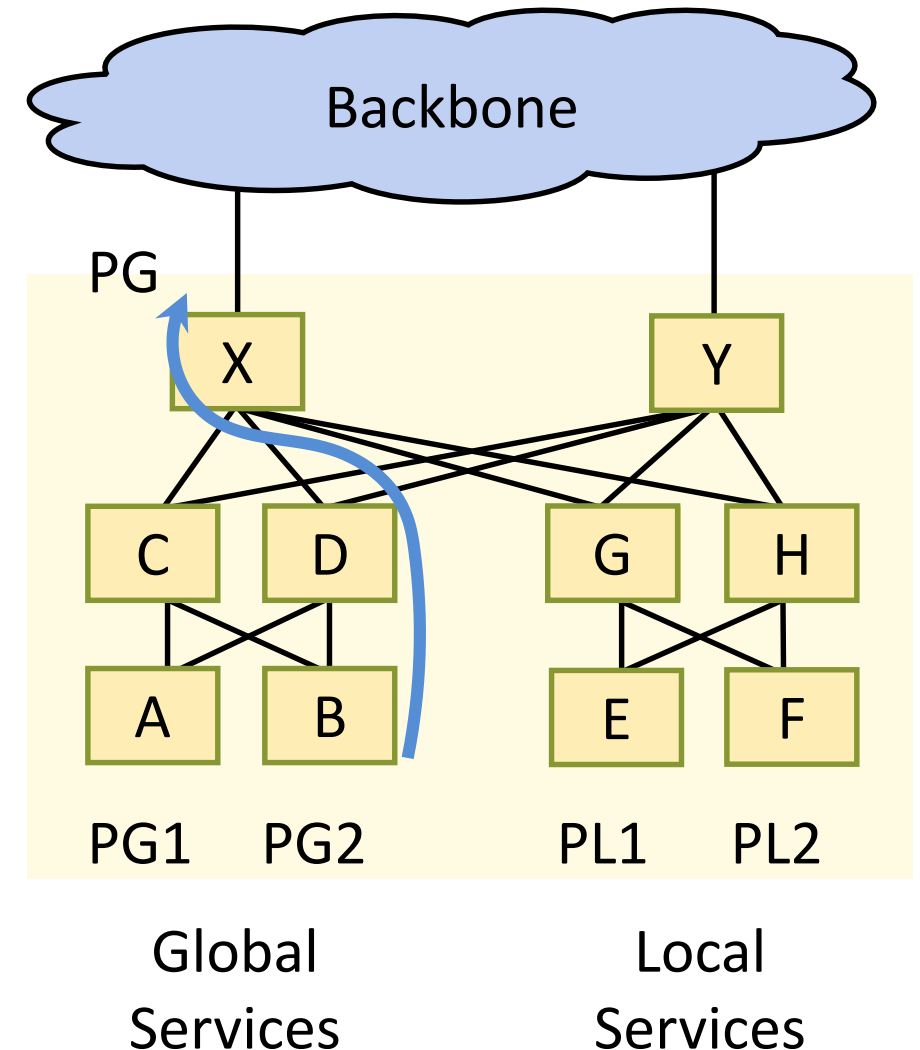
# Example #2: A data center network

## Goals

- Keep local prefixes internal
- Aggregate global prefixes as PG

## Attempt #2

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



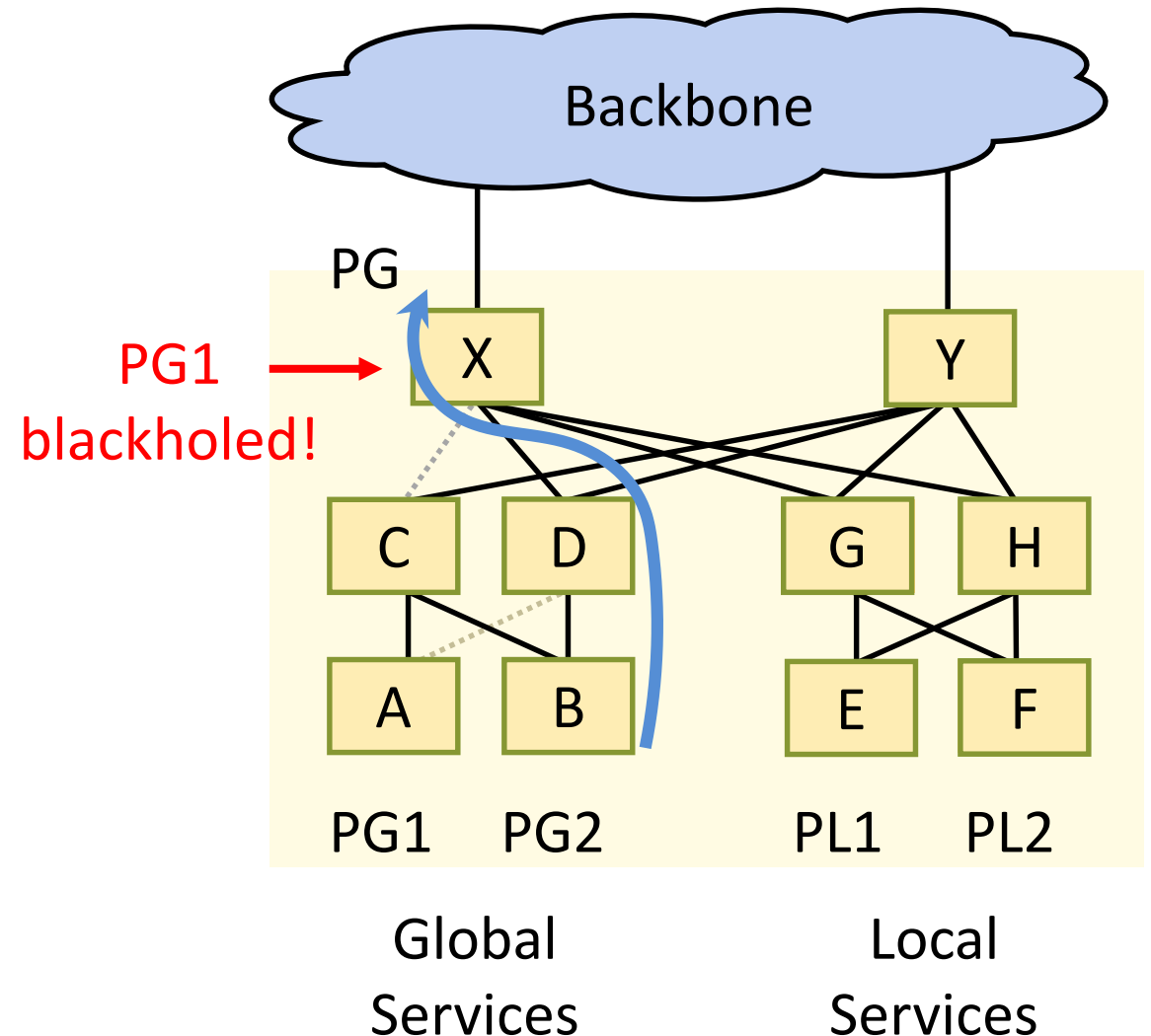
# Example #2: A data center network

## Goals

- Keep local prefixes internal
- Aggregate global prefixes as PG

## Attempt #2

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



# Example #2: A data center network

## Goals

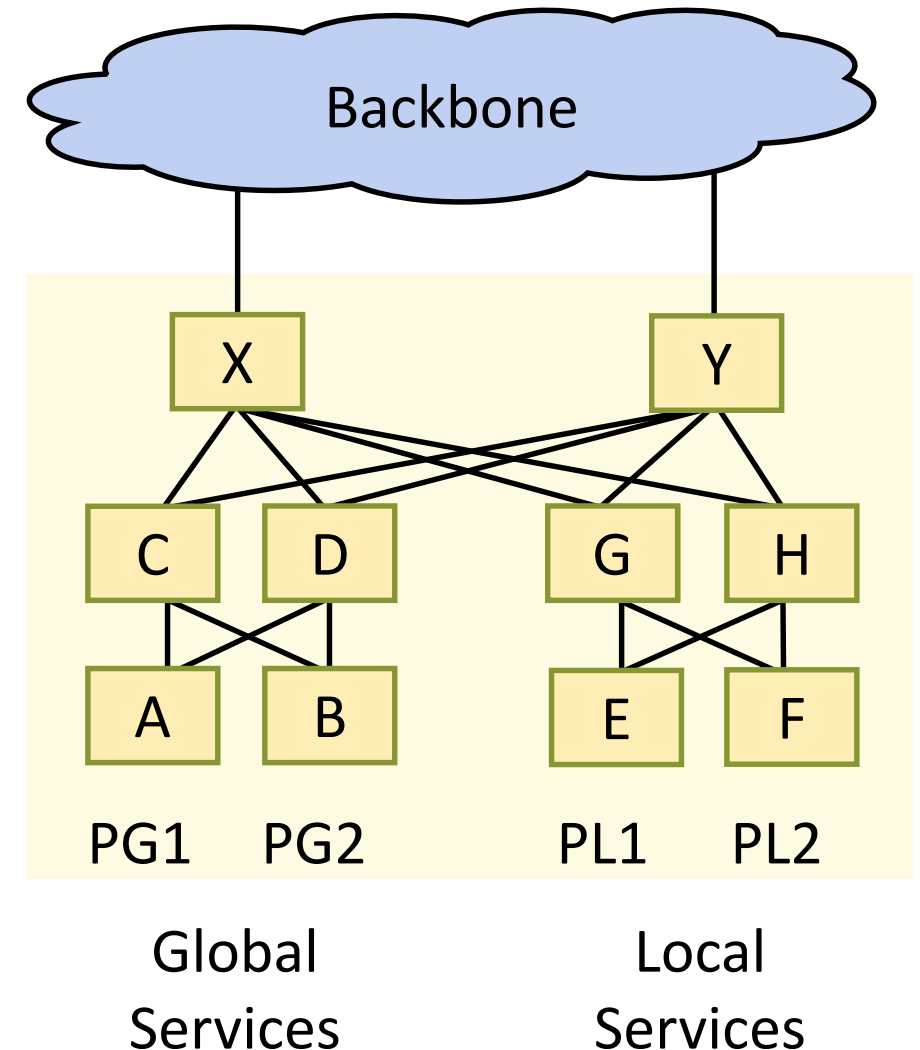
- Keep local prefixes internal
- Aggregate global prefixes as PG

```
define ownership = {PG1 => end(A),  
                    PG2 => end(B),  
                    PL1 => end(E),  
                    PL2 => end(F)}
```

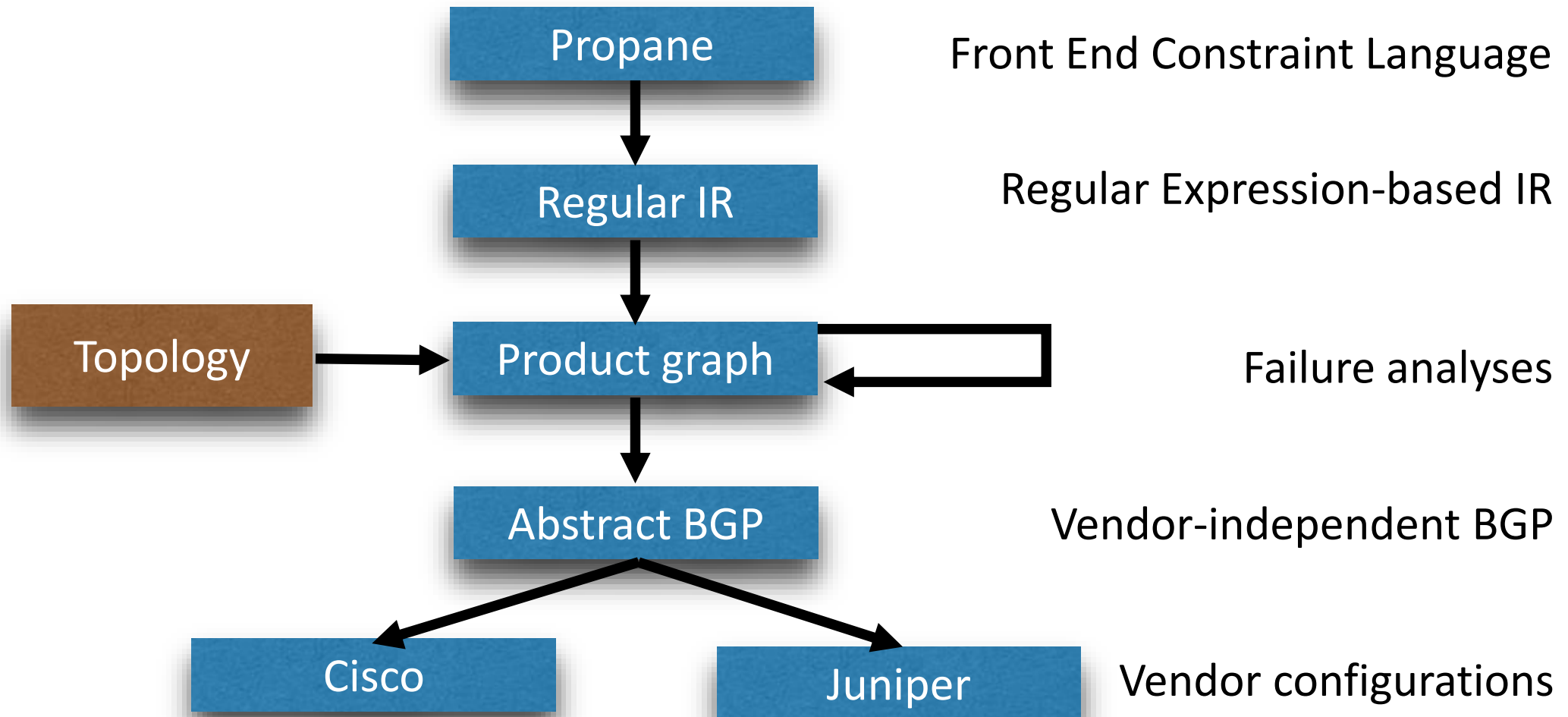
```
define locality = { {PL1, PL2} => always(in) }
```

```
control {aggregate(PG, in -> out)}
```

```
define main = routing and locality
```



# Propane compiler





# Propane Regular IR

Propane

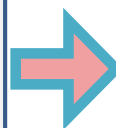


Regular  
IR

Step 1: Combine modular constraints

Prefix-by-prefix intersection of constraints

```
define ownership = {PG1 => end(A),  
                    PG2 => end(B),  
                    PL1 => end(E),  
                    PL2 => end(F)}  
define locality = { {PL1, PL2} => always(in)}  
control {aggregate(PG, in -> out)}  
define main = routing and locality
```



```
PG1  => end(A)  
PG2  => end(B)  
PL1  => always(in) and end(E)  
PL2  => always(in) and end(F)
```

# Propane Regular IR

Step 2: Expand constraints in to regular expressions

```
any = out*.in+.out*  
end(X) = ( $\Sigma^*$ .X)  
always(X) = (X)*  
exit(X) = (out*.in*. (X  $\cap$  in).out+) |  
          (out*.in+. (X  $\cap$  out).out*)  
start(X) = (X. $\Sigma^*$ )  
avoid(X) = (!X)*  
waypoint(X) = ( $\Sigma^*$ .X. $\Sigma^*$ )
```

Step 3: Reduced syntax

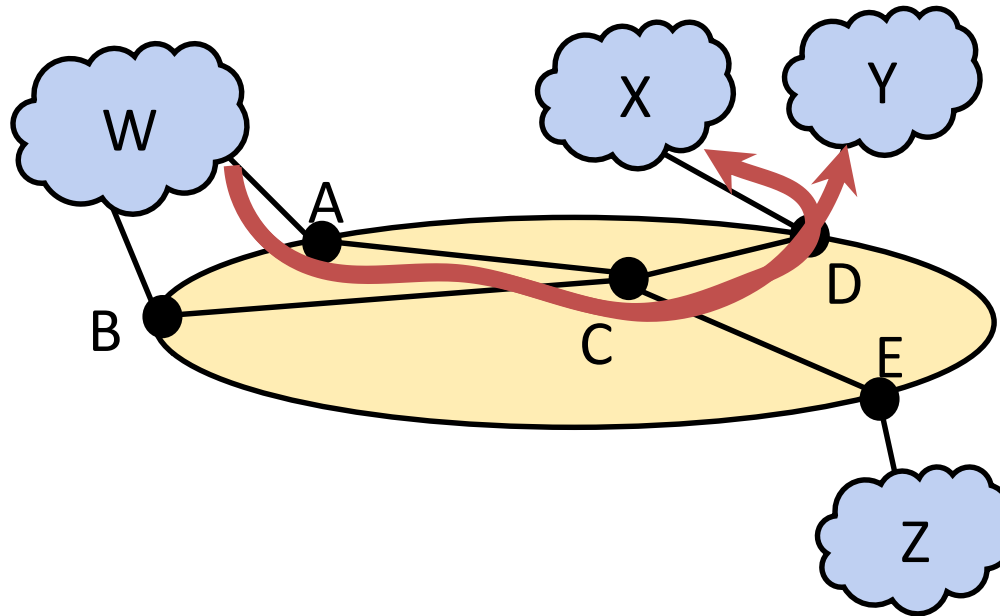
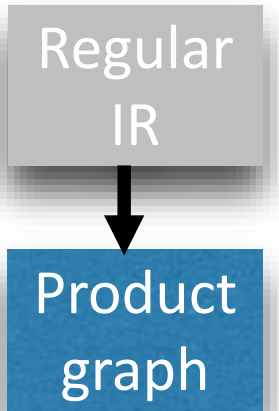
```
true => A. (X >> Y).out*  
true => (A.X.out*) >> (A.Y.out*)
```

Propane



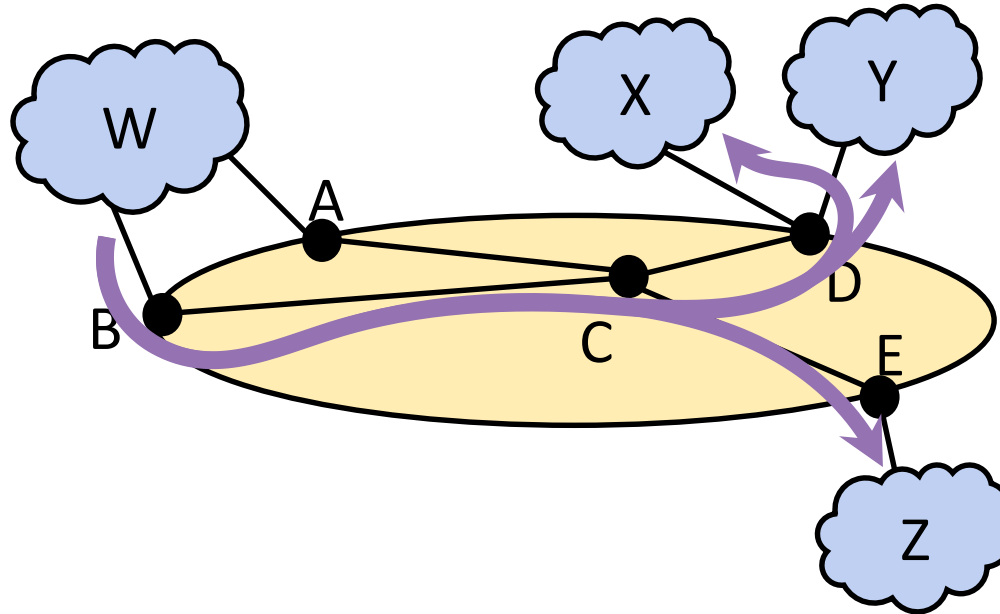
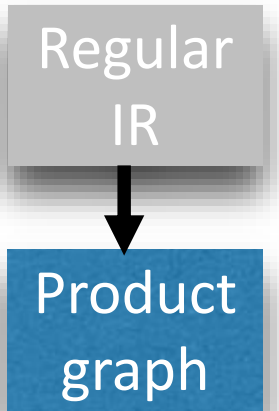
Regular  
IR

# PG construction: An Example



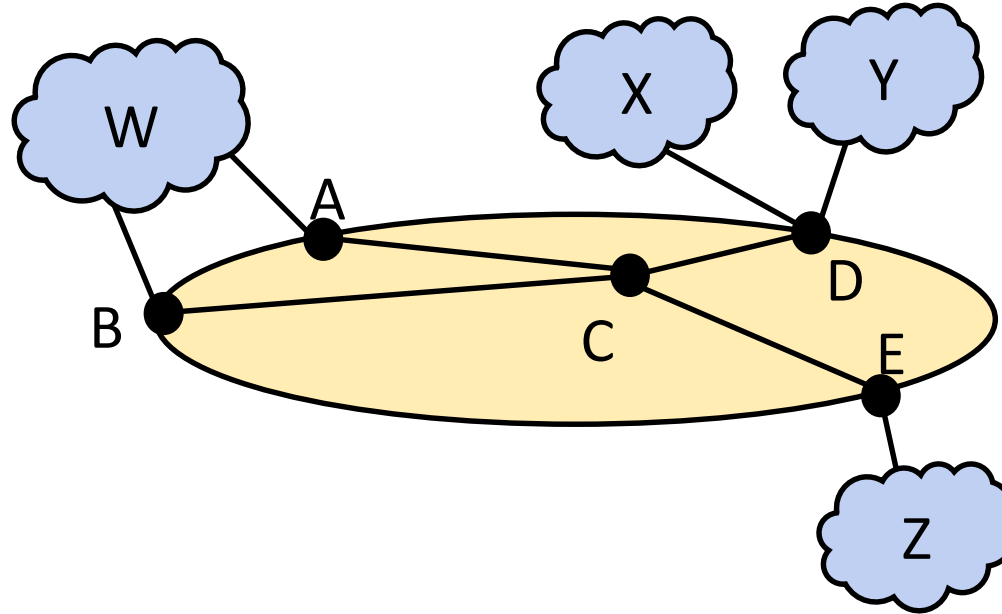
Policy:  $(W.A.C.D.out) \gg (W.B.in+.out)$

# PG construction: An Example

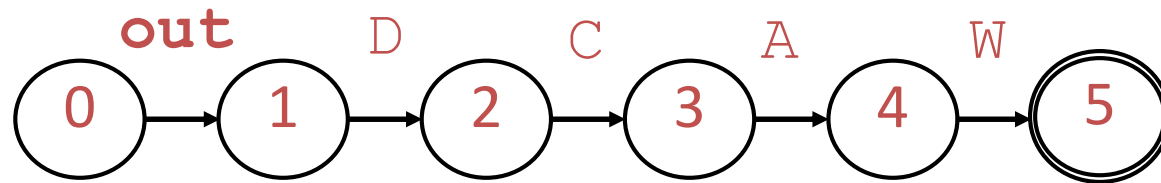


Policy:  $(W.A.C.D.out) \gg (W.B.in+.out)$

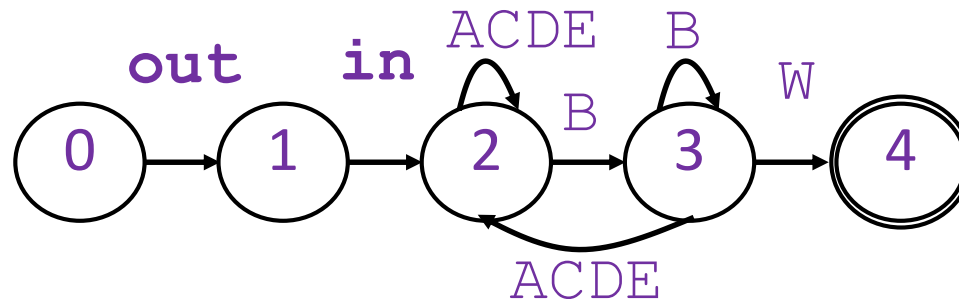
# PG construction: Reversed policy automata



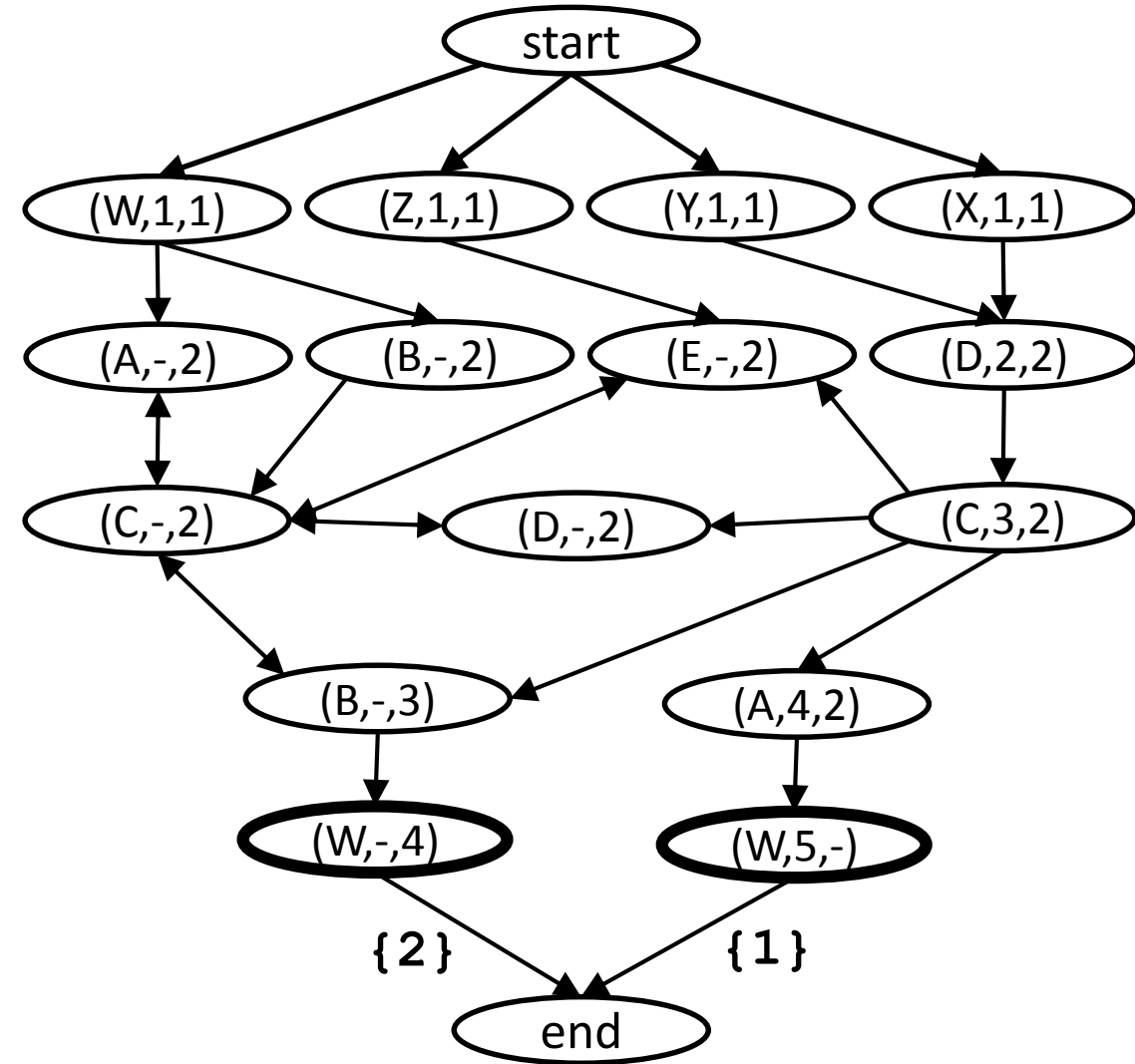
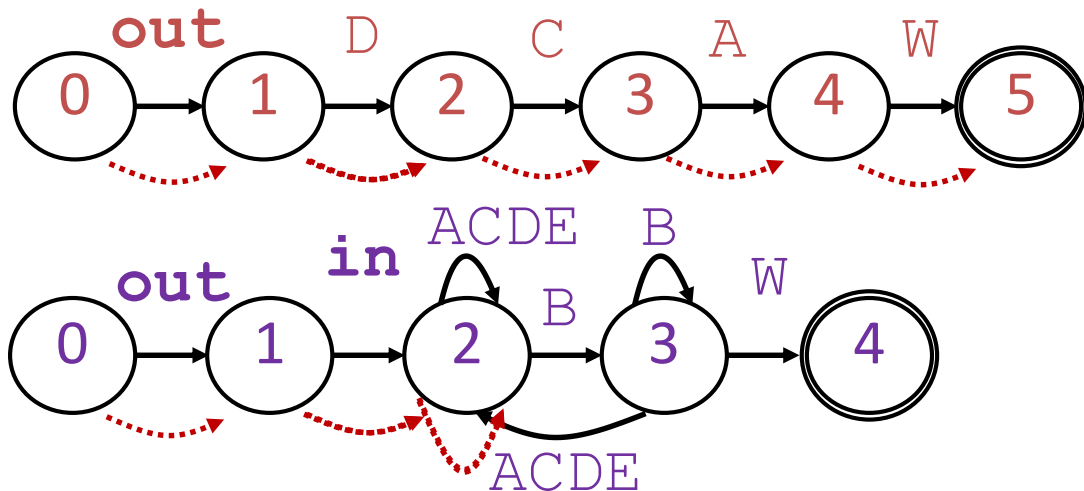
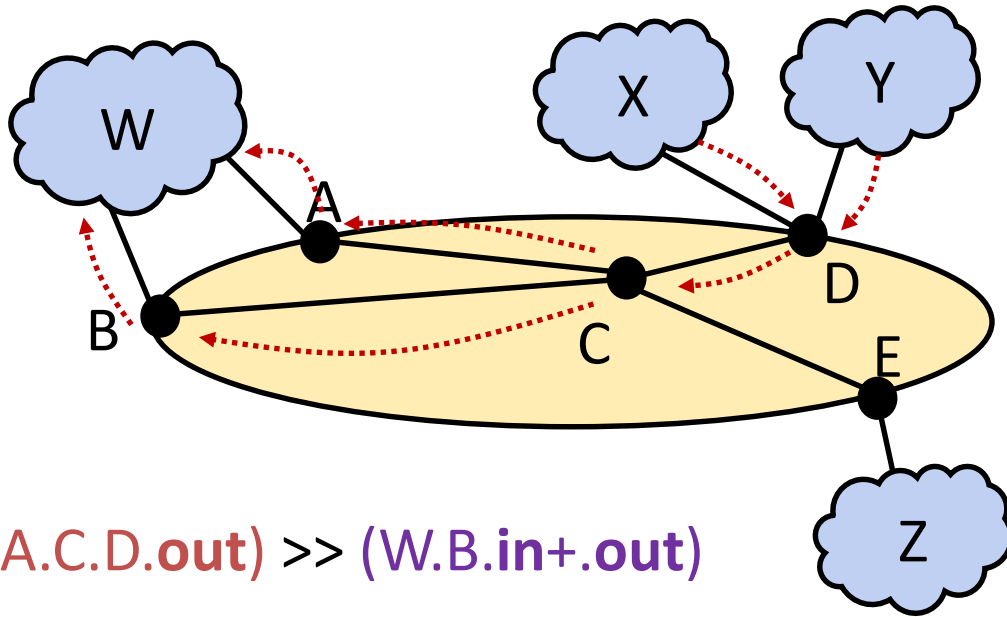
(W.A.C.D.out)



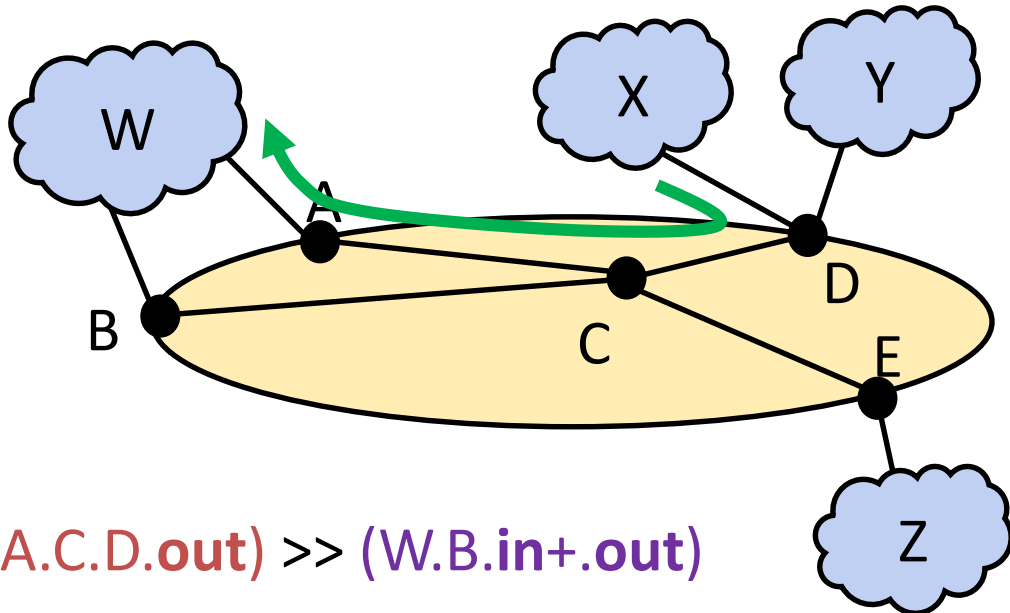
(W.B.in+.out)



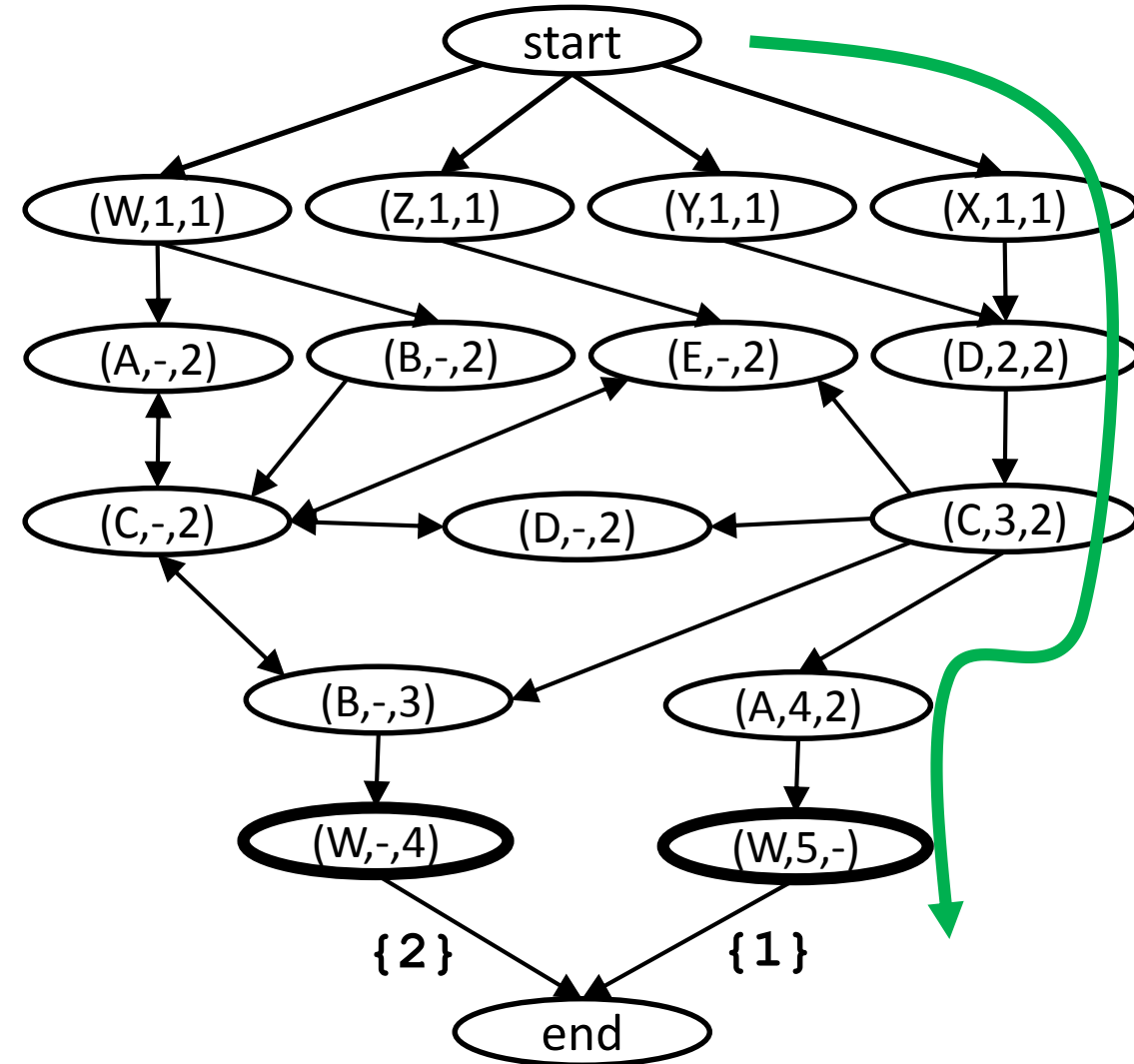
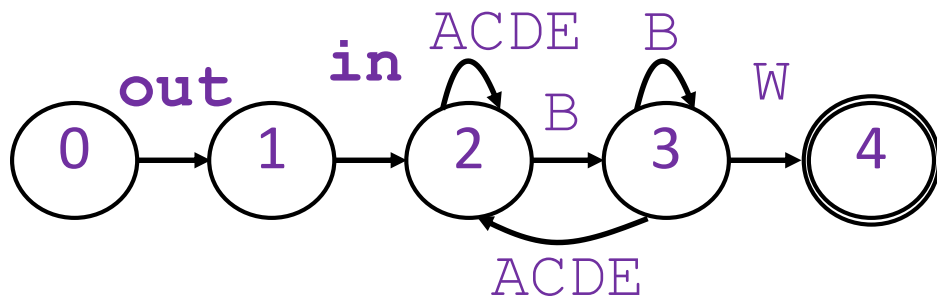
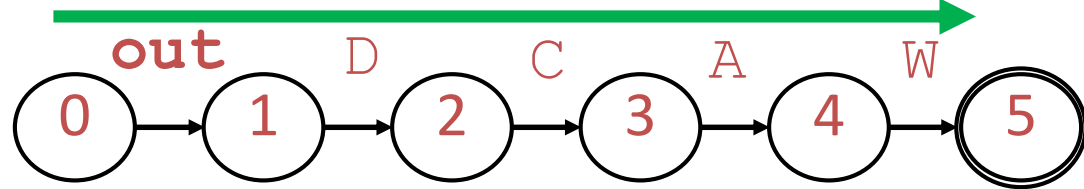
# PG construction: Graph generation



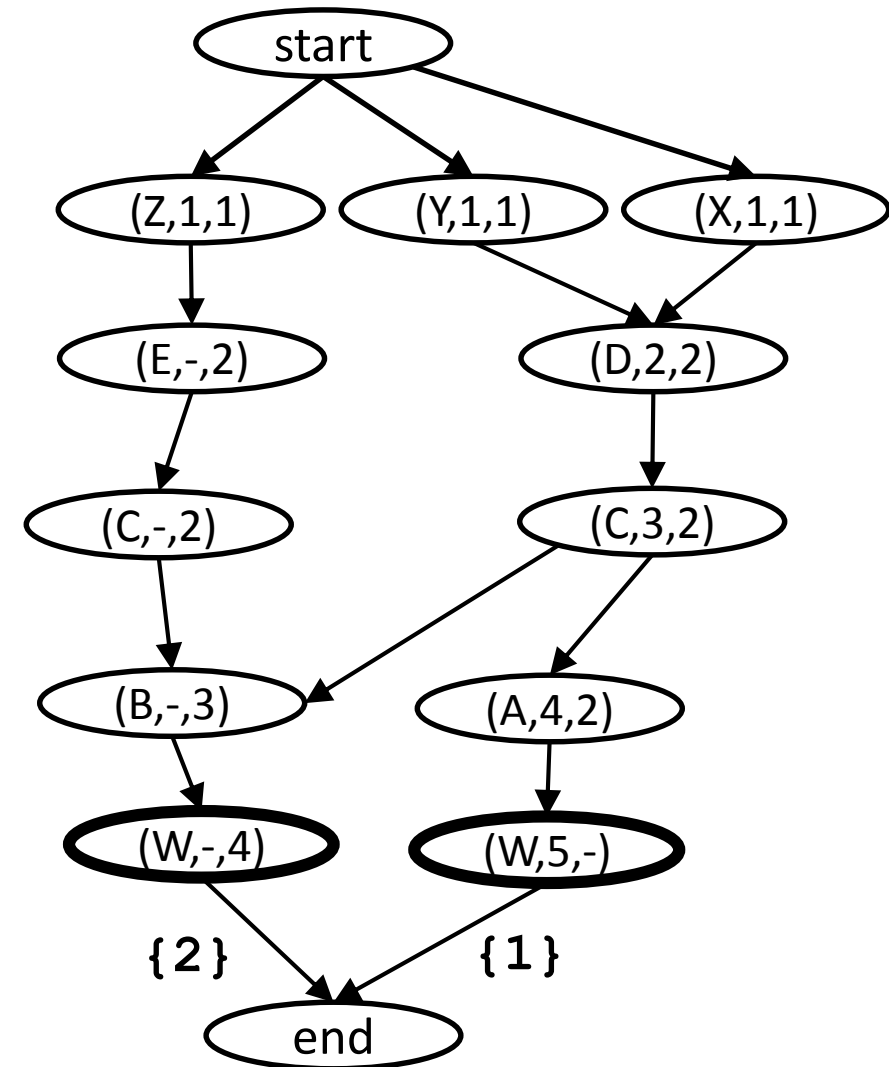
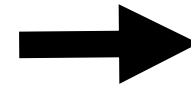
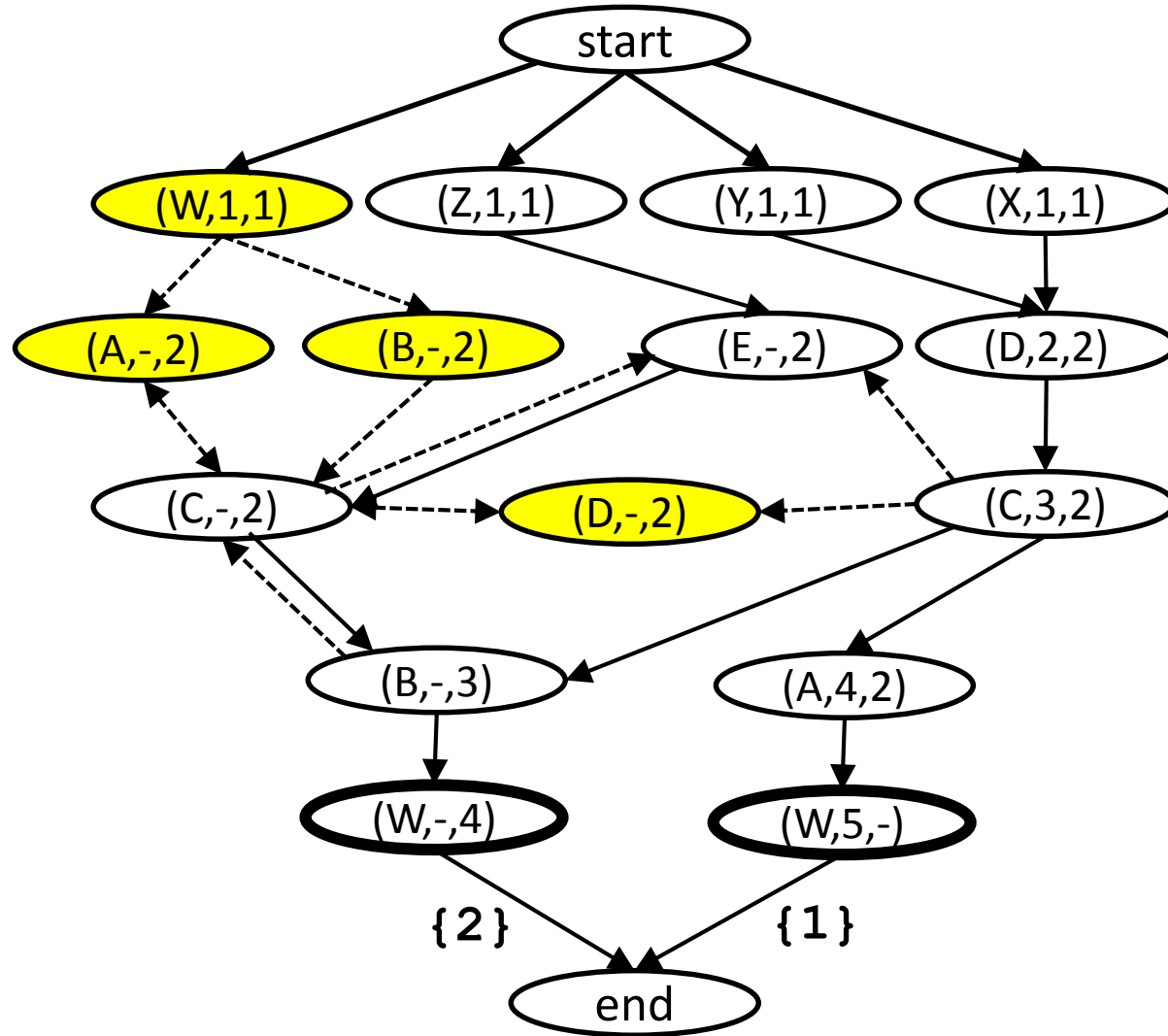
# PG construction: Graph generation



$(W.A.C.D.out) \gg (W.B.in+.out)$

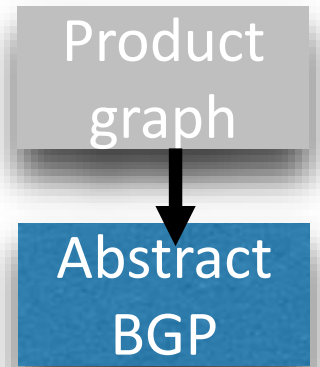
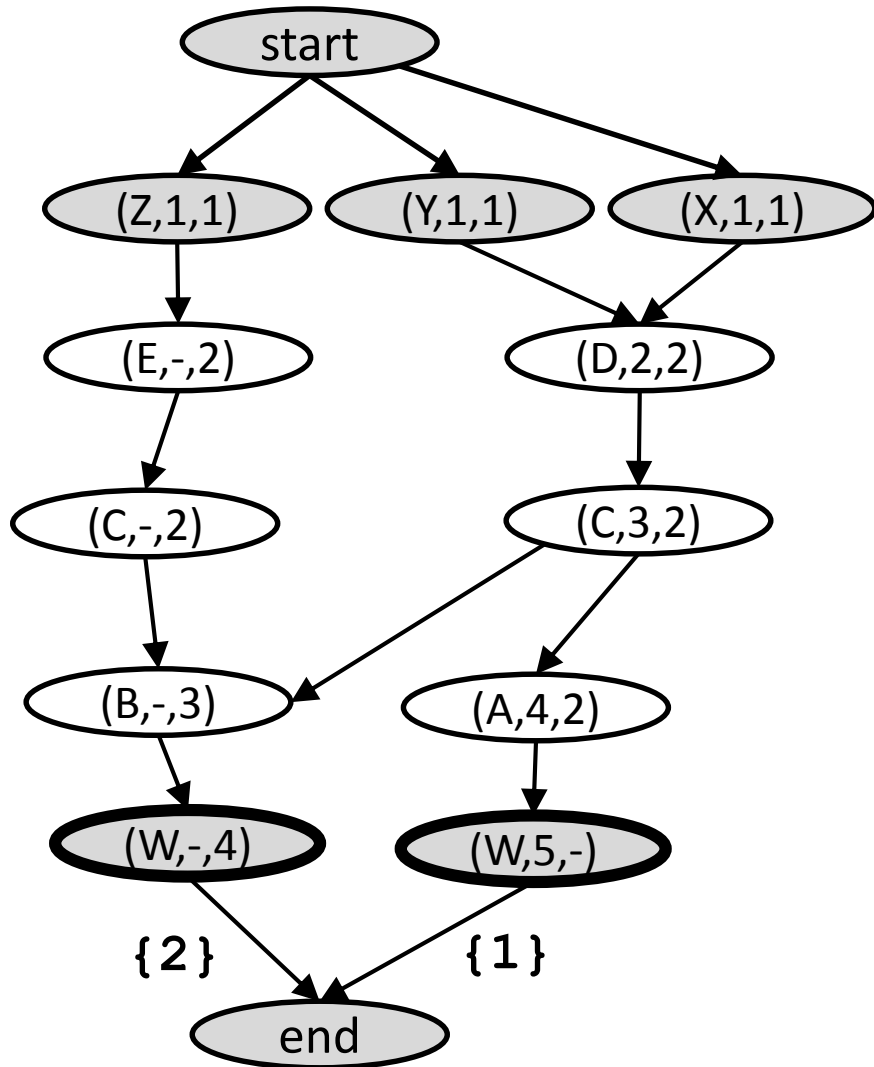


# PG construction: minimization (loop analysis)





# Compilation to ABGP



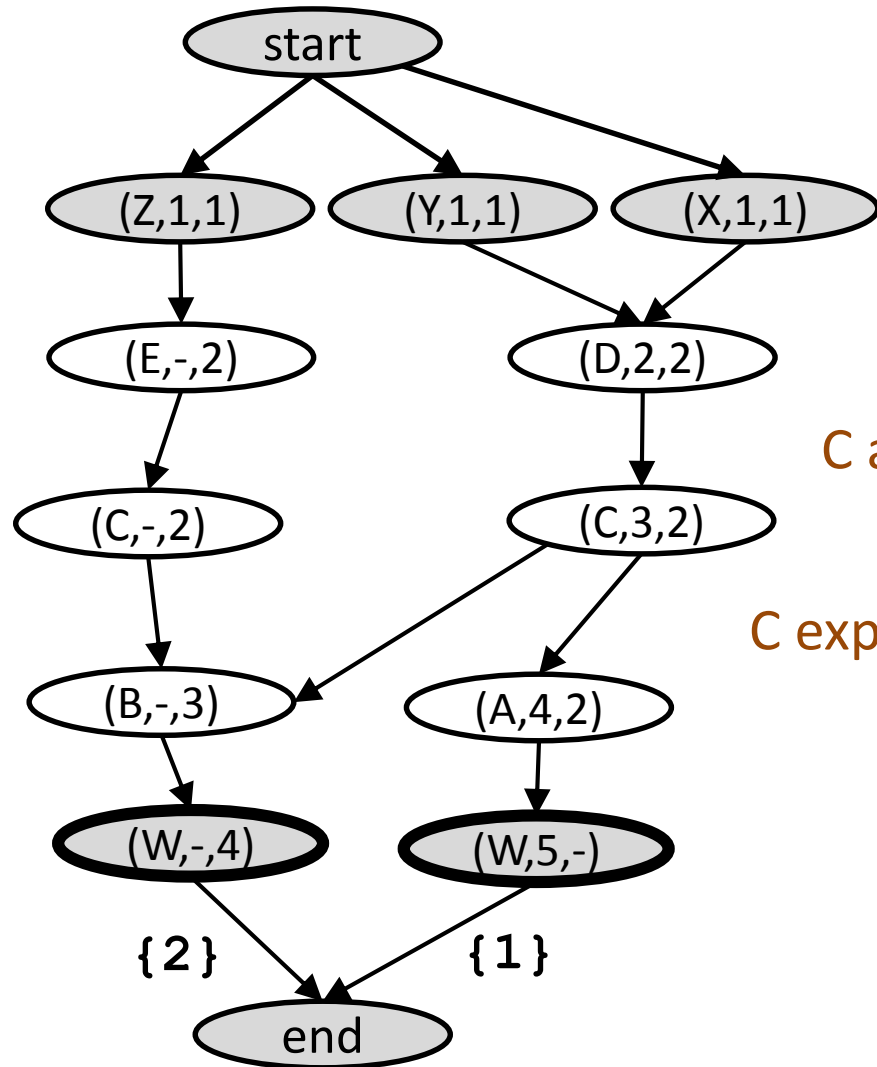
Idea 1: Restrict advertisements to PG edges

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



Let BGP find **some allowed** path dynamically

# Compilation to BGP



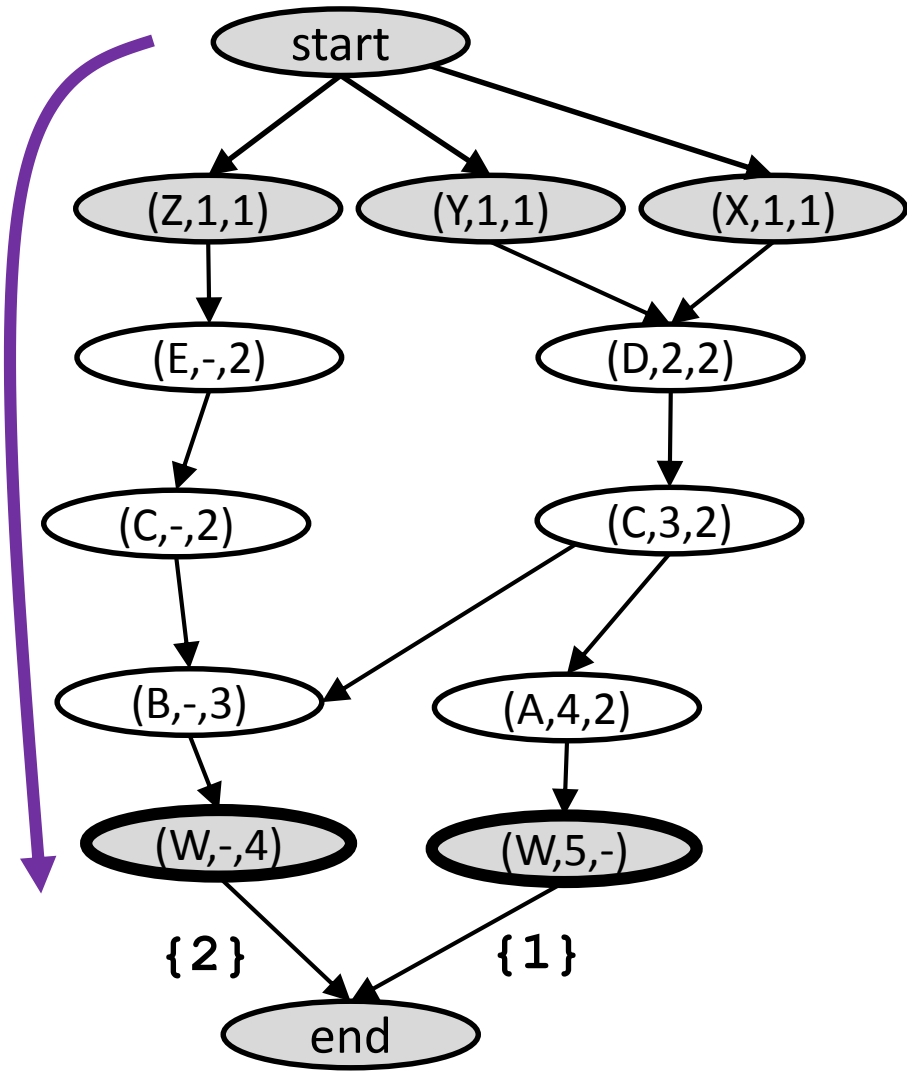
C allows import from D with tag (2,2)

C exports to A,B with tag (3,2)

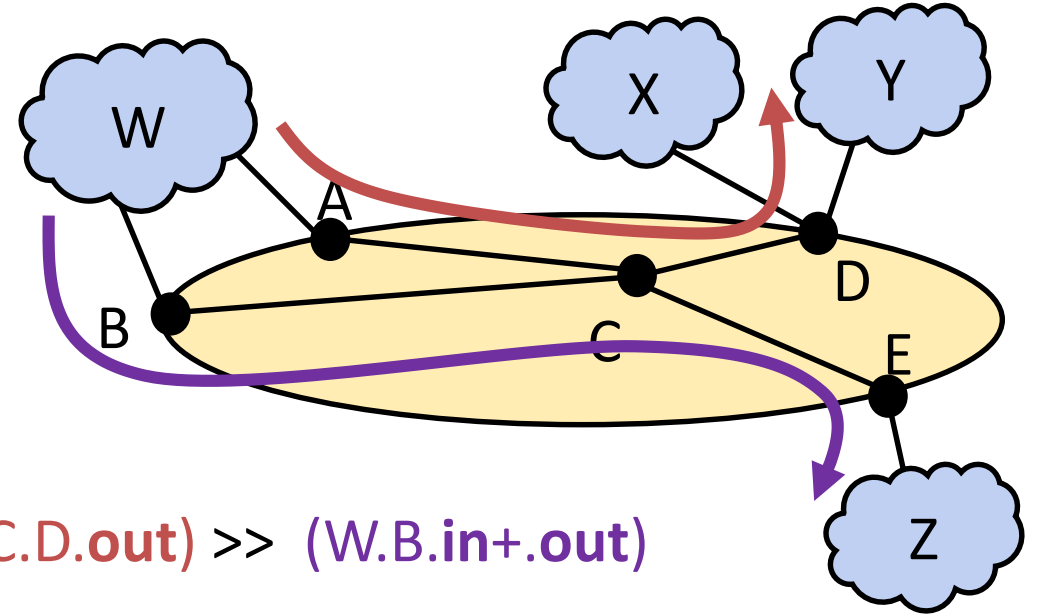
Product  
graph

Abstract  
BGP

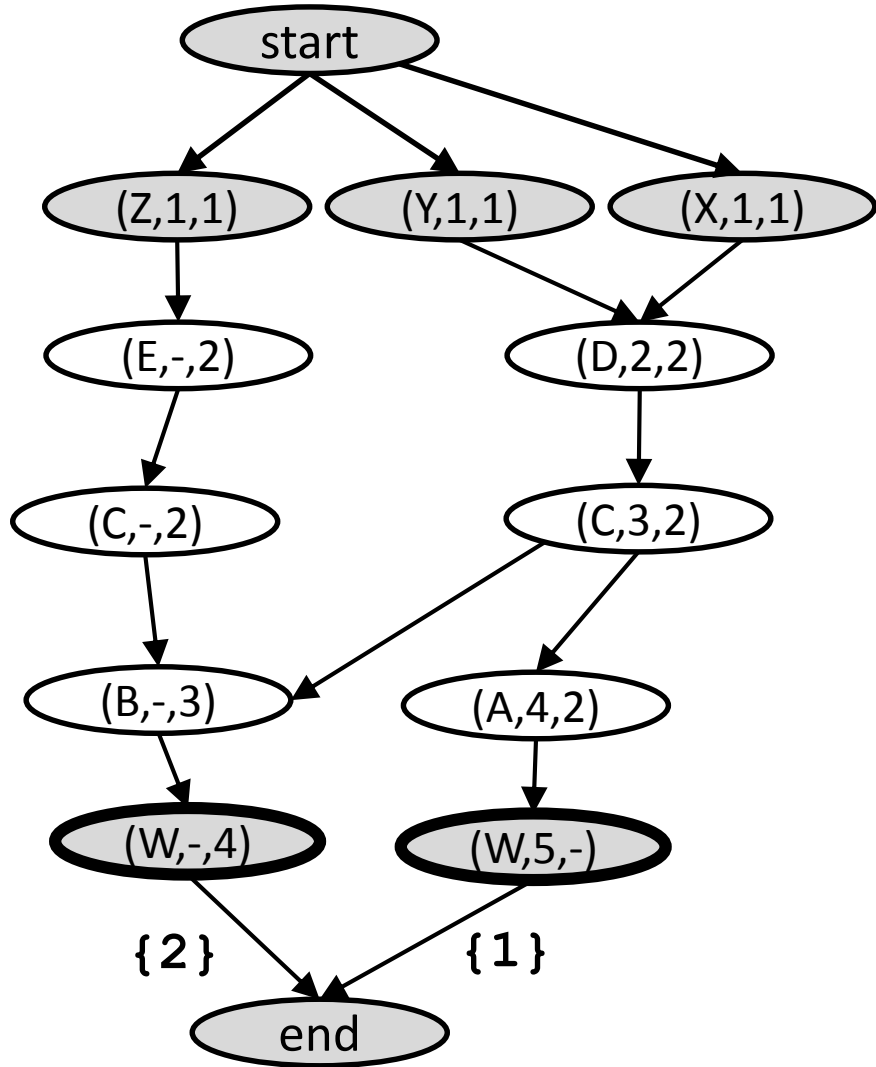
# Compilation to BGP



A better path exists in the network, but is not used!



# Compilation to BGP



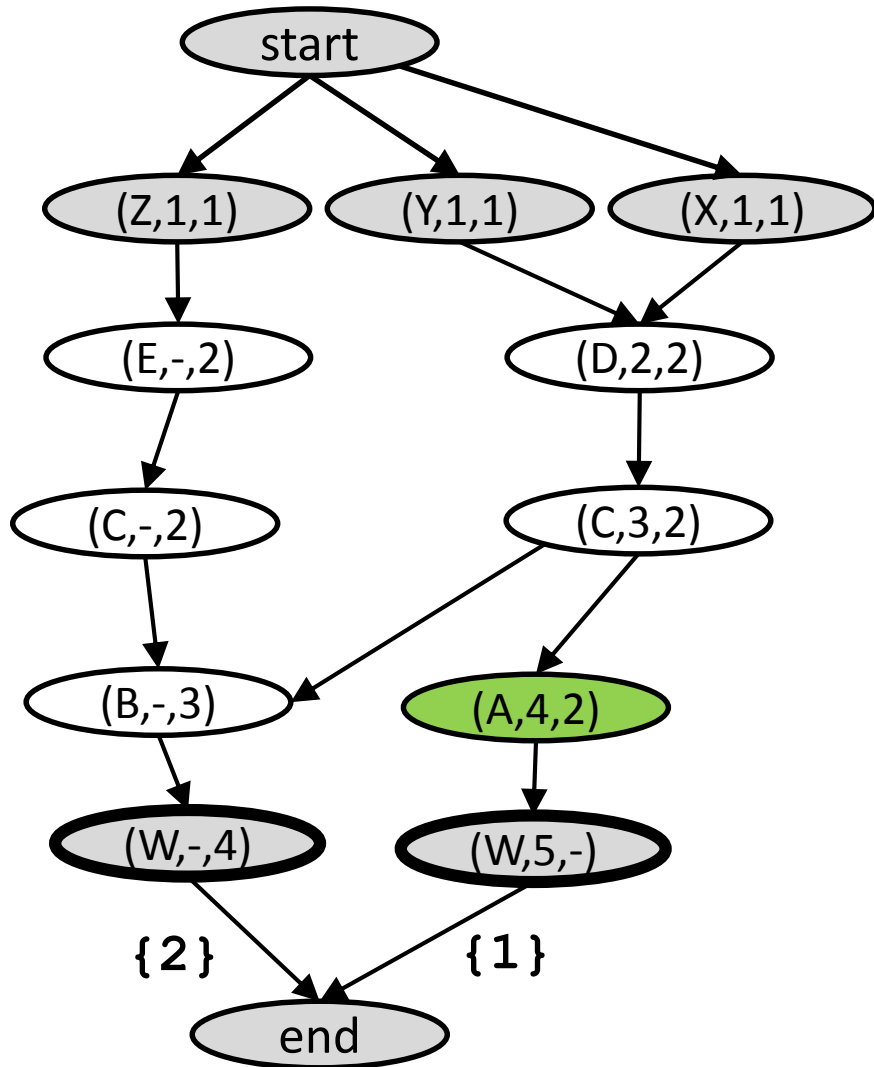
Idea 2: Synthesize local preferences

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



Let BGP find **the best allowed** path dynamically

# Compilation to BGP



Router A

```
match peer=C comm=(3,2)
export peer←W, comm←(4,2),
      comm← noexport, MED←80
```

Router B

```
match peer=C
export peer←W, comm←(-,3),
      comm←noexport, MED←81
```

Router C

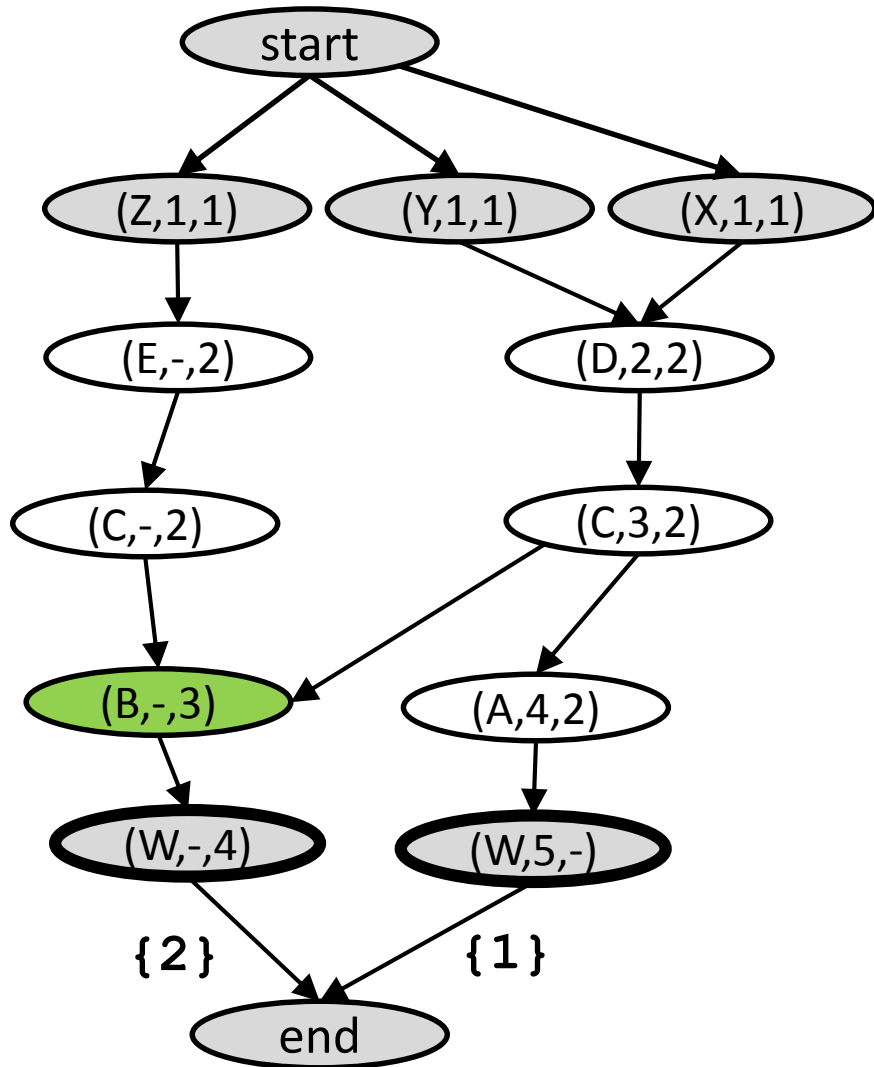
```
match[lp=99] peer=E, comm=(-,2)
export peer←B, comm←(-,2)
match[lp=100] peer=D, comm=(2,2)
export peer←A,B, comm←(3,2)
```

Router D

```
match regex=(X + Y)
export peer←C, comm←(2,2)
```

...

# Compilation to BGP



Router A

match peer=C comm=(3,2)  
export peer←W, comm←(4,2),  
comm←noexport, MED←80

Router B

match peer=C  
export peer←W, comm←(-,3),  
comm←noexport, MED←81

Router C

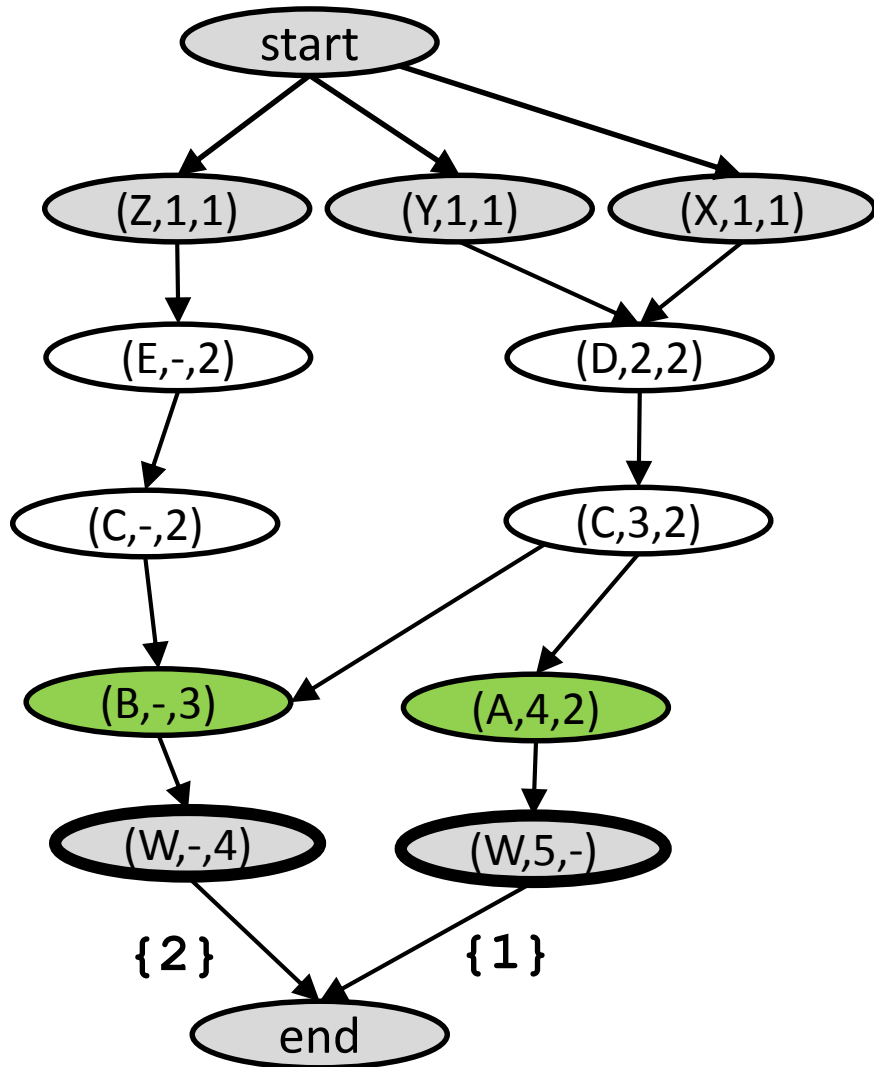
match[ip=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[ip=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)

Router D

match regex=(X + Y)  
export peer←C, comm←(2,2)

...

# Compilation to BGP



Router A

match peer=C comm=(3,2)  
export peer←W, comm←(4,2),  
comm←noexport, MED←80

Router B

match peer=C  
export peer←W, comm←(-,3),  
comm←noexport, MED←81

Router C

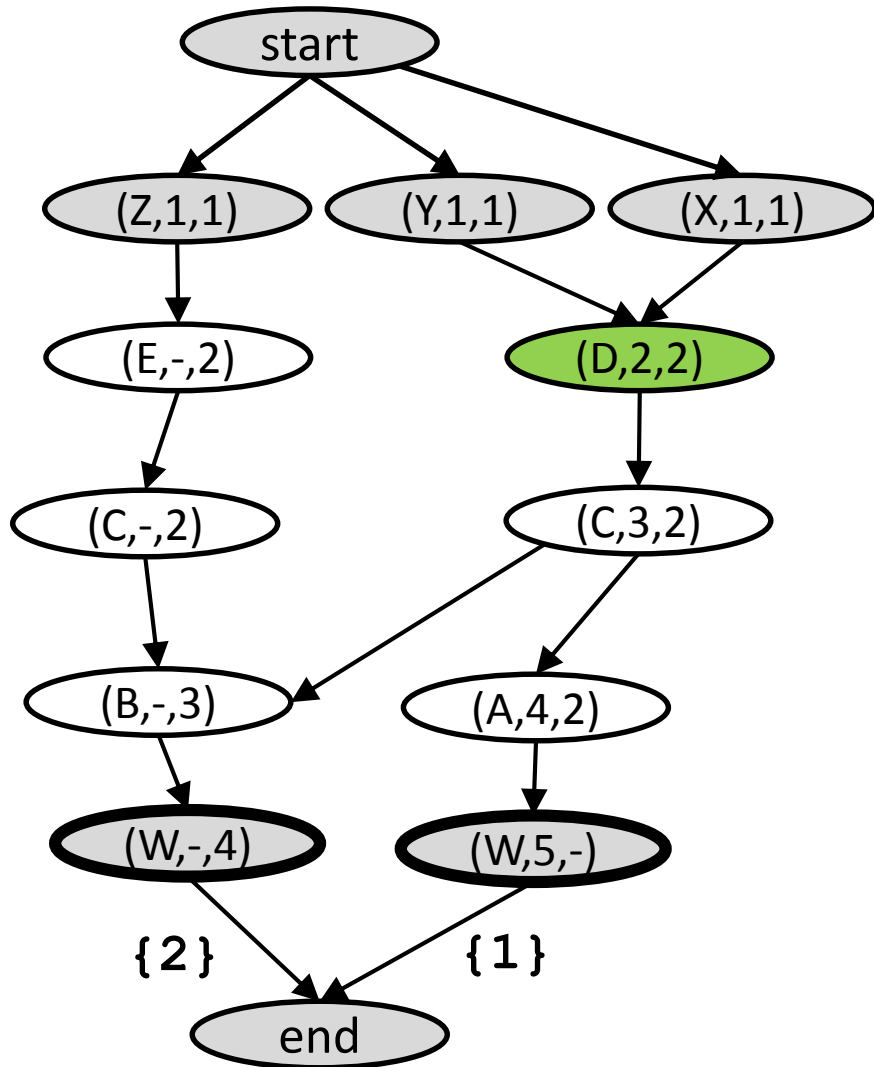
match[ip=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[ip=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)

Router D

match regex=(X + Y)  
export peer←C, comm←(2,2)

...

# Compilation to BGP



Router A

match peer=C comm=(3,2)  
export peer←W, comm←(4,2),  
comm← noexport, MED←80

Router B

match peer=C  
export peer←W, comm←(-,3),  
comm←noexport, MED←81

Router C

match[ip=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[ip=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)

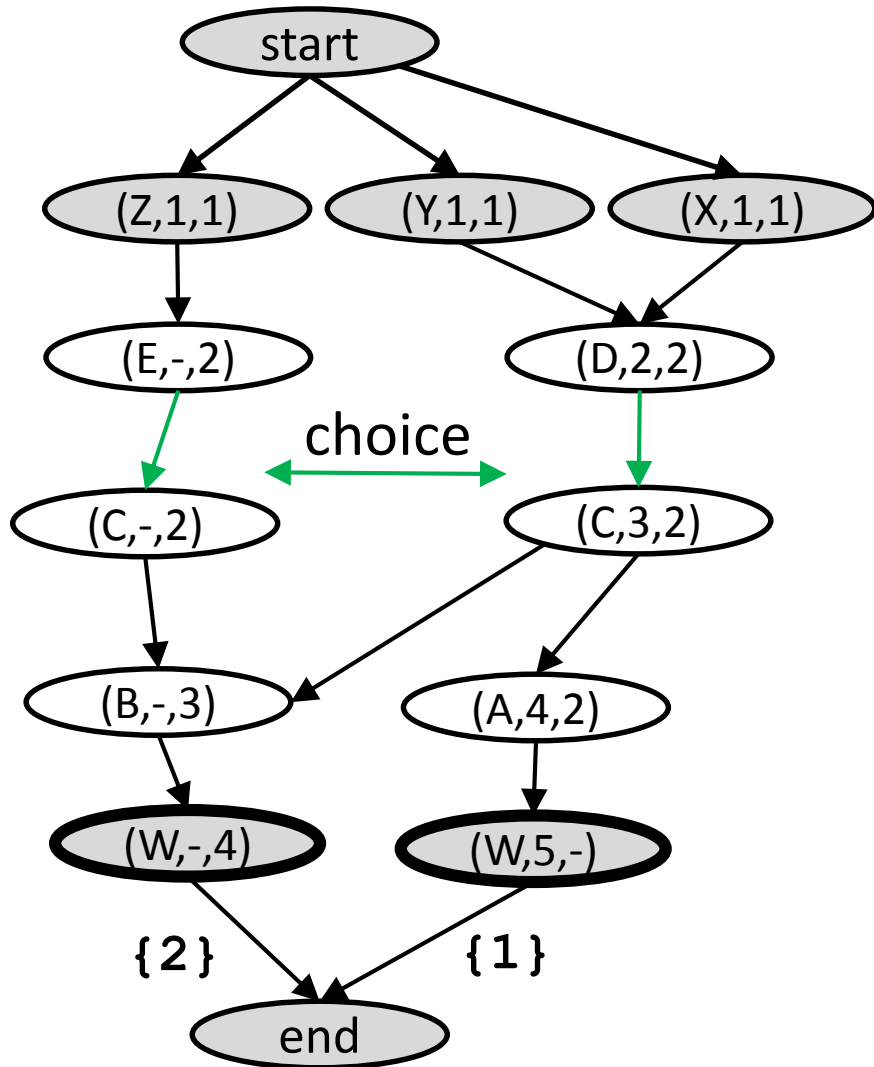
Router D

match regex=(X + Y)  
export peer←C, comm←(2,2)

...



# Compilation to BGP



Router A

```
match peer=C comm=(3,2)
export peer←W, comm←(4,2),
comm← noexport, MED←80
```

Router B

```
match peer=C
export peer←W, comm←(-,3),
comm←noexport, MED←81
```

Router C

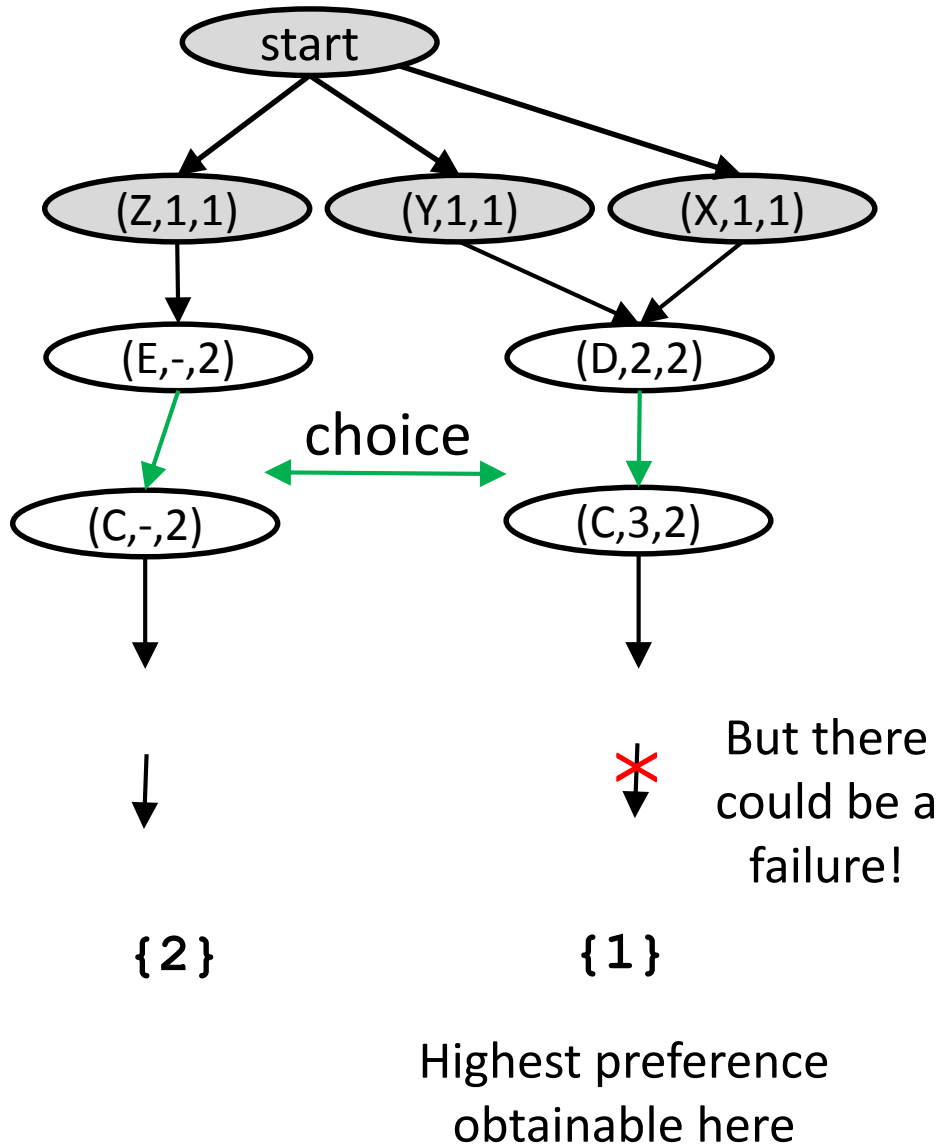
```
match[lp=99] peer=E, comm=(-,2)
export peer←B, comm←(-,2)
match[lp=100] peer=D, comm=(2,2)
export peer←A,B, comm←(3,2)
```

Router D

```
match regex=(X + Y)
export peer←C, comm←(2,2)
```

...

# Compilation to BGP



Router A

```
match peer=C comm=(3,2)
export peer←W, comm←(4,2),
comm← noexport, MED←80
```

Router B

```
match peer=C
export peer←W, comm←(-,3),
comm←noexport, MED←81
```

Router C

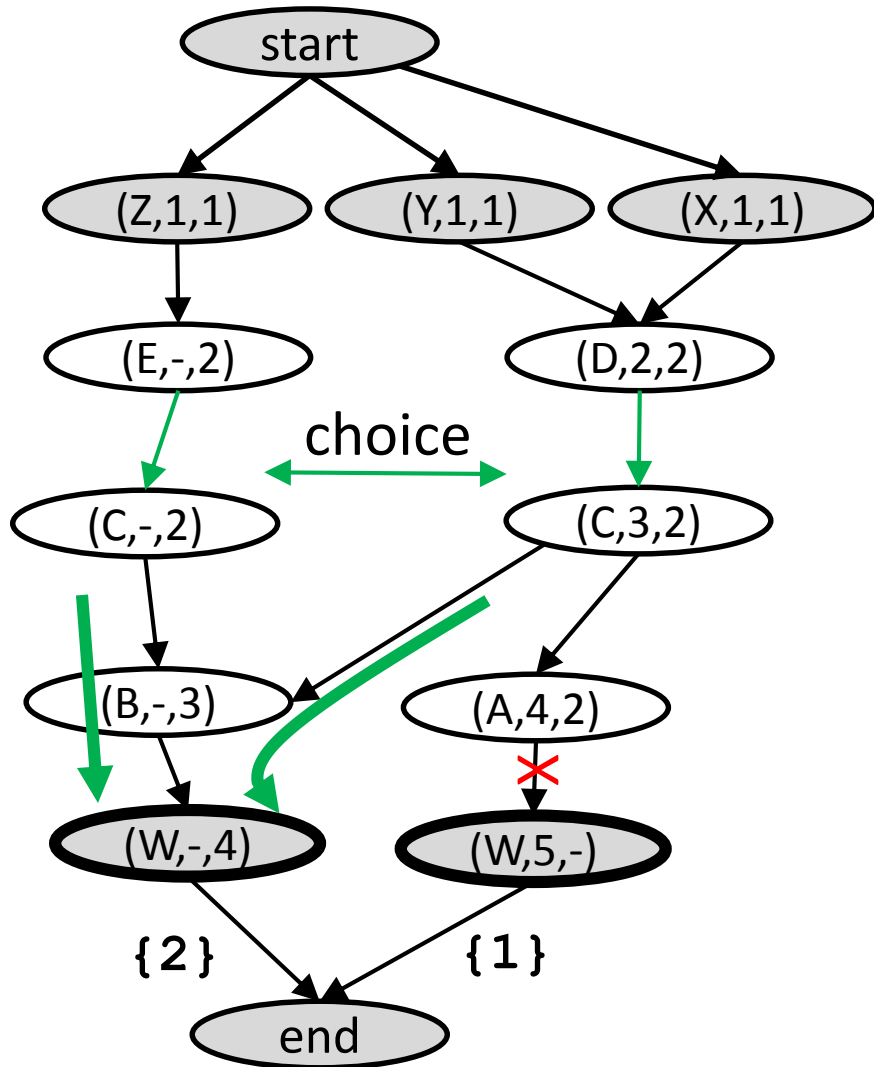
```
match[lp=99] peer=E, comm=(-,2)
export peer←B, comm←(-,2)
match[lp=100] peer=D, comm=(2,2)
export peer←A,B, comm←(3,2)
```

Router D

```
match regex=(X + Y)
export peer←C, comm←(2,2)
```

...

# Compilation to BGP



Router A

match peer=C comm=(3,2)  
export peer←W, comm←(4,2),  
comm← noexport, MED←80

Router B

match peer=C  
export peer←W, comm←(-,3),  
comm←noexport, MED←81

Router C

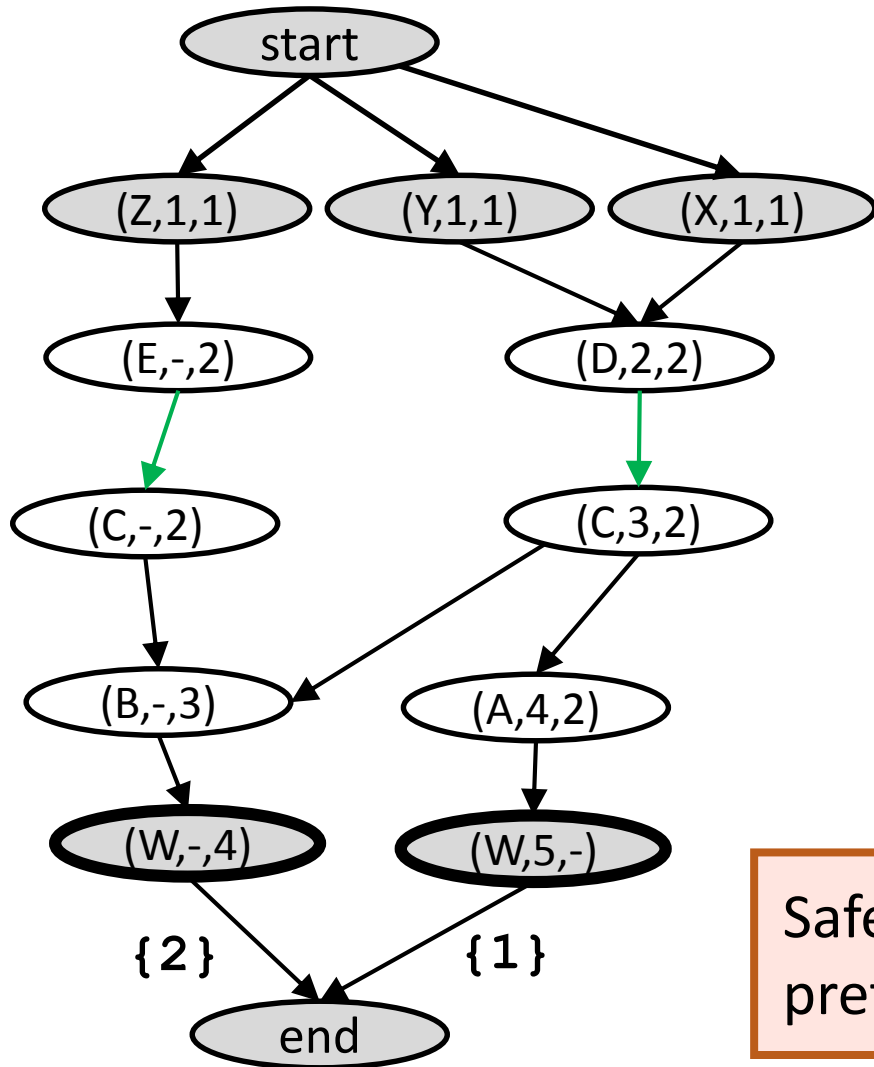
match[ip=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[ip=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)

Router D

match regex=(X + Y)  
export peer←C, comm←(2,2)

...

# Compilation to BGP



Safe to  
prefer D

Router A

match peer=C comm=(3,2)  
export peer←W, comm←(4,2),  
comm←noexport, MED←80

Router B

match peer=C  
export peer←W, comm←(-,3),  
comm←noexport, MED←81

Router C

match[ip=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[ip=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)

Router D

match regex=(X + Y)  
export peer←C, comm←(2,2)

...

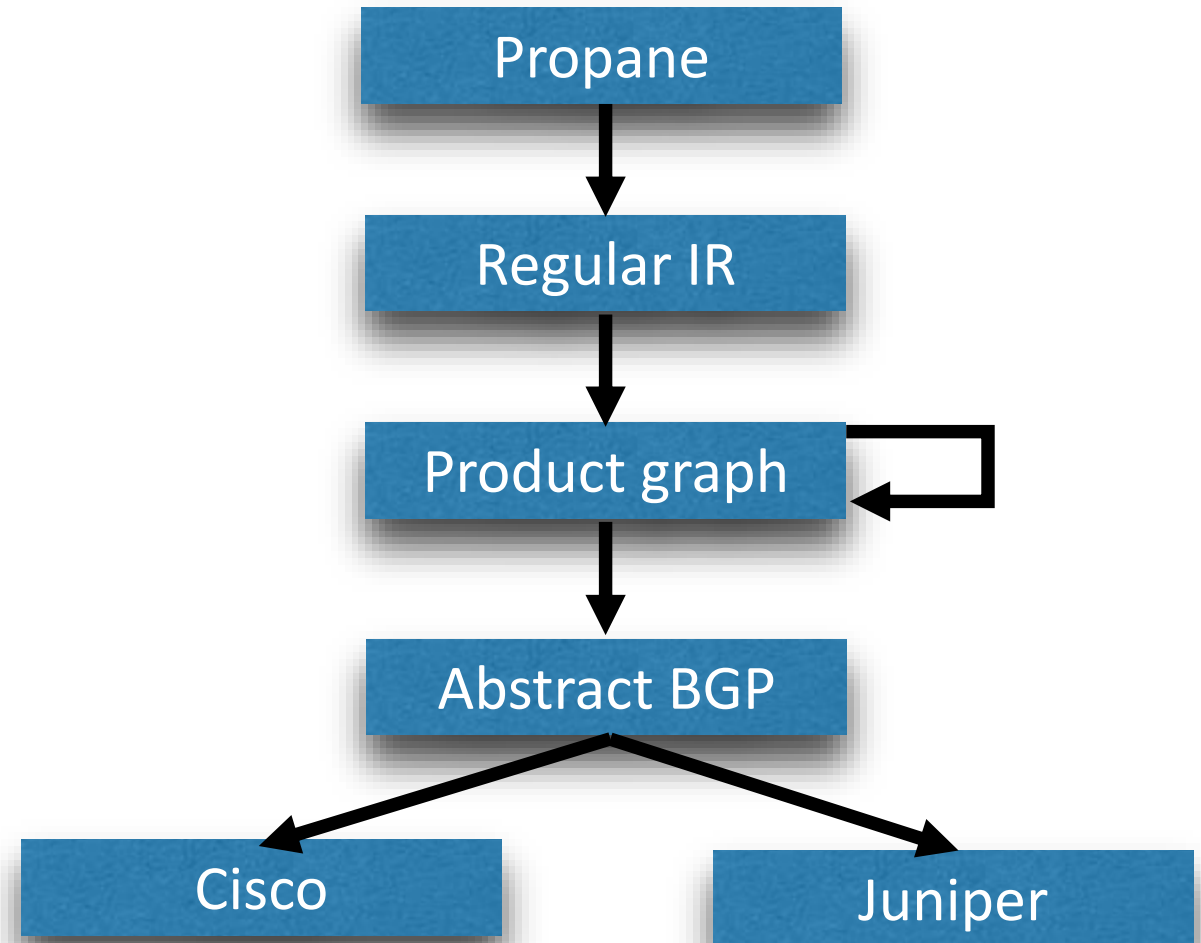
# Propane compiler implementation

Efficient graph algorithms

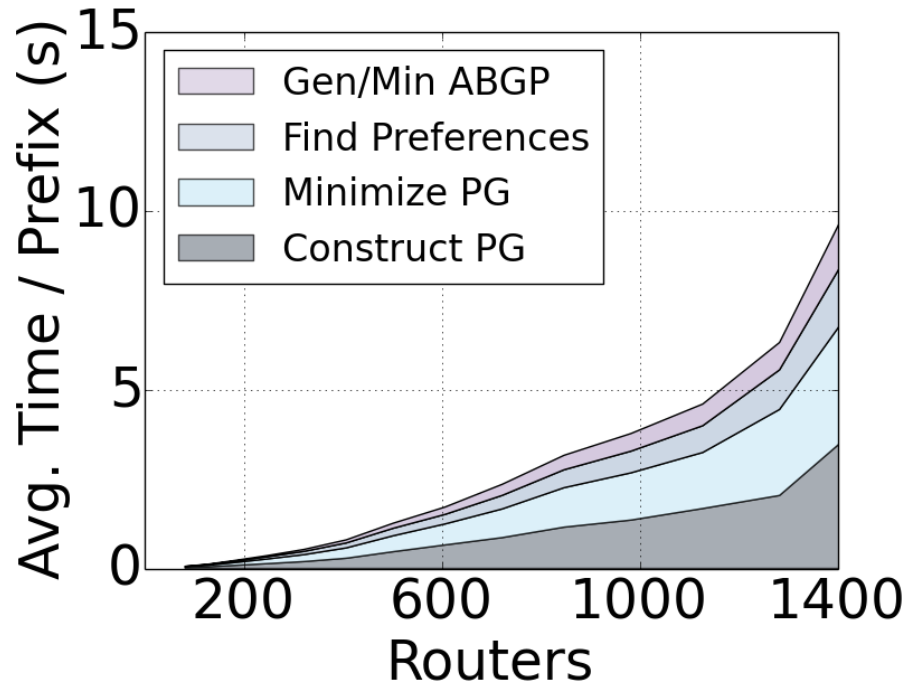
- Minimization
- Failure safety
- Aggregation blackholes

Config minimization

5500 LoC (F#)

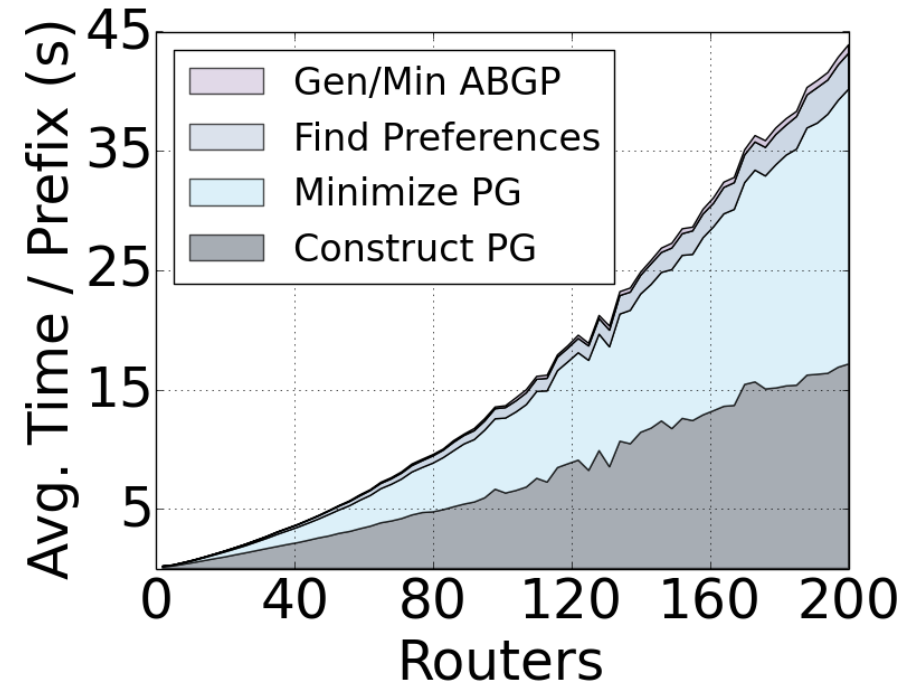


# Evaluation on Microsoft network policies



Data center networks

- 31 lines of Propane
- 9 mins for 1400 routers



Backbone networks

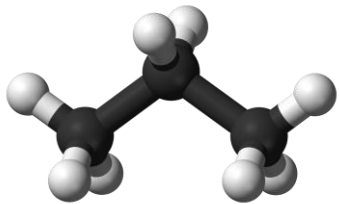
- 43 lines of Propane
- 3 mins for 200 routers

# Summary

Centralized programming of  
distributed control planes



Resilient **and** programmable  
networks



**Propane**

[github.com/rabeckett/propane](https://github.com/rabeckett/propane)

Generates BGP configurations from high-level policies  
using a **product graph abstraction** of control plane