

**Propane**

# **Programming Distributed Control Planes**

**Ryan Beckett (Princeton & MSR)**

**Ratul Mahajan (MSR)**

**Todd Millstein (UCLA)**

**Jitu Padhye (MSR)**

**David Walker (Princeton)**

# Configuring Networks is Error-Prone

## Understanding BGP Misconfiguration

Ratul Mahajan

David Wetherall

Tom Anderson

{ratul,djw,tom}@cs.washington.edu  
Computer Science and Engineering  
University of Washington  
Seattle, WA 98195-2350

### ABSTRACT

It is well-known that simple, accidental BGP configuration errors can disrupt Internet connectivity. Yet little is known about the frequency of misconfiguration or its causes, except for the few spectacular incidents of widespread outages. In this paper, we present the first quantitative study of BGP misconfiguration. Over a three week period, we analyzed routing table advertisements from 23 vantage points across the Internet backbone to detect incidents of misconfiguration. For each incident we polled the ISP operators involved to verify whether it was a misconfiguration, and to learn the cause of the incident. We also actively probed the Internet to determine the impact of misconfiguration on connectivity.

Surprisingly, we find that configuration errors are pervasive, with 200-1200 prefixes (0.2-1.0% of the BGP table size) suffering from misconfiguration each day. Close to 3 in 4 of all new prefix advertisements were results of misconfiguration. Fortunately, the connectivity seen by end users is surprisingly robust to misconfigurations. While misconfigurations can substantially increase the update load on routers, only one in twenty five affects connectivity. While the causes of misconfiguration are diverse, we argue that most could be prevented through better router design.

### Categories and Subject Descriptors

C.2.3 [Communication Networks]: Operations—management;  
C.4 [Computer Systems]: Performance—reliability, availability, and serviceability

### General Terms

Human Factors, Management, Reliability

### 1. INTRODUCTION

As the Internet's inter-domain routing protocol, the Border Gateway Protocol (BGP) [34] is crucial to the overall reliability of the Internet. Faults in BGP implementations or mistakes in the way it is used have been known to disrupt large regions of the Internet. Recent studies have examined several kinds of BGP problems, including excessive churn due to implementation deficiencies [26],

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
*SIGCOMM'02*, August 19-23, 2002, Pittsburgh, Pennsylvania, USA.  
Copyright 2002 ACM 1-58113-570-X/02/0008 ...\$5.00.

delayed convergence [24], persistent oscillations due to policy interactions [18, 40], and instability caused by the propagation of worms [9].

In this paper we examine another source of unreliability: the misconfiguration of the routers that speak BGP. We know from numerous studies of highly reliable systems, such as aircraft, bank databases, and the telephone network, that human operator error can account for 20-70% of system failures [3, 6, 15, 21]. These studies have shown that as systems become more reliable, the human factor becomes increasingly important to overall reliability. We would expect the same to be true of the Internet. There is substantial anecdotal evidence that BGP configuration errors do occur, with serious consequences. The canonical example is the AS7007 incident [31], in which AS7007 accidentally announced routes to most of the Internet and disrupted connectivity for over two hours. Despite the publicity over this event, serious misconfigurations continue to occur. In April 2001, AS3561 propagated more than 5000 improper route announcements from one of its downstream customers [12], again leading to global connectivity problems.

In this paper, we complement the anecdotal study of infrequent large scale events with a microscopic study of more frequent “near misses” – globally visible BGP misconfigurations that occur many times per day but do not necessarily disrupt connectivity. As with the study of airplane near collisions, we hope that our study of microscopic events can help improve the design of systems to avoid future larger scale problems. To the best of our knowledge, this is the first systematic study of globally visible BGP misconfigurations; it complements other studies that have examined backbone failures in general [25]. Our goal is to answer four questions:

- *How frequently do these misconfigurations occur?*
- *What is their impact on global connectivity and routing load?*
- *Why do the misconfigurations occur?*
- *What can be done to reduce their frequency and impact?*

We consider two broad classes of faults that propagate across the backbone and hence are visible from our measurement points: *i*) the accidental injection of routes into global BGP tables, including address space hijacks; and *ii*) the accidental export of routes in violation of an ISP's policy. We focus on misconfigurations that are globally visible because they arguably have the potential to cause wider disruption than those that do not propagate across the Internet. We analyze the entire stream of BGP updates taken from 23 different vantage points around the Internet for a period of 21 days. We show that it is possible to identify misconfigurations with simple heuristics. To validate our results, we surveyed the ISP operators involved in each incident via email. We asked them whether the in-

**“Close to 3 in 4 of all new prefix advertisements were results of misconfiguration”**

## Causes:

- operator slips
- low-level details: filters, communities
- logical errors in plans
- misunderstanding of configuration semantics
- router bugs

# Configuring Networks is Error-Prone

2/5/2016 China routing snafu briefly mangles interweb

[Log in](#) | [Sign up](#)

**GET Y ONLY YOU**

(https://www.thousandeyes.com/)

← Blog Home



DATA CENTER SOFTWARE NETWORKS

**Networks ▶ Broadband**

## China routing snafu

Cockup, not conspiracy

9 Apr 2010 at 12:24, John Leyden

Bad routing information sourced from China

Global BGP (Border Gateway Routing) is a Telecommunication, apparently accidentally Telecoms, IDG reports. ISPs Qwest and Telefonica accepted ill-thought BGP is a core routing protocol which manages the net. Several routing options are now equivalent of TomTom publishing routes between London and Paris.

IDC China Telecommunications published about 10 per cent of the net - instead of viable routing options by many service providers (at the time) after China Telecommunications network Asia would have been more likely to affect incident were recorded all over the world. BGPmon.net, a BGP monitoring service described as a prefix hijack, [here](#).

Although it seems they [IDC China] about 10 per cent of these prefixes include prefixes for popular websites [www.rapidshare.com](#) and [www.getpano.com](#). A large number of networks impacted include some popular Chinese websites [www.huanqiu.com](#), [www.tianya.cn](#) and [www.sohu.com](#).

A cock-up is suspected, rather than a conspiracy.

Given the large number of prefixes hijacked. Most likely it's because of speculation.

The practical consequences of the screw-up dropped connections or, worse, traffic routes one of the clearest illustrations of the serious nonetheless important network protocol.

The China BGP global routing representation management. For example, just two weeks ago internet traffic through a DNS (Domain Name System) was taken down by a network mortem by internet monitoring firm Renesys. availability is a major concern. Look at the problem. Normally, two locations (Tokyo and Dallas) transit Road Runner while the rest go through AT&T (Figure 2).

24h 7d 1m 3m 6m 1y 3y 5y

DECEMBER 24, 2005

COM

ENGINEERING TODD UNDERWOOD

## Internet-Wide Catastrophe

8+

One year ago today TTNet in Taiwan took down the Internet. And unfortunately for the network providers believed their misconfiguration caused the problem. Telecom/YouTube incident was [Columnist Johnna Till Johnson](#)'s.

What is significant about the

« Previous Story

able to re

re can take

the intervening time.

morning 2004, TTNet (AS9121) started announcing

<http://research.dyn.com/2005/12/internetwide-nearcatastrophela/>

2/5/2016

Sign In | Register

**≡**

# NETWORKWORLD

## You Tube/Pakistan your site?

### Configuring BGP pro says

By Carolyn Duffy Marsan

Network World | Mar 10, 2008 1:00 AM

*In light of Pakistan Telecom/Youtube/Pakistan your web site victimized by such*

When Pakistan Telecom blocked international incident that wreaked RIPE NCC, the European registry during Pakistan Telecom's hijack attack.

We posed some questions to R. Here's what he had to say:

#### How frequently do hijacking

Misconfigurations of iBGP (inter Autonomous System) happen misconfiguration caused the Pakistan Telecom/YouTube incident was [Columnist Johnna Till Johnson](#)'s.

What is significant about the

« Previous Story

able to re

re can take

the intervening time.

morning 2004, TTNet (AS9121) started announcing

<http://www.bgpmon.net/news-and-events/>

2/5/2016 News and Press | BGPmon

**BGPMON** Now part of **OpenDNS**

HOME BLOG ABOUT US PRODUCTS AND SERVICES NEWS AND PRESS CLIENT PORTAL

At BGPmon we regularly blog and work with reporters and governments to investigate and report relevant issues. Below you'll find a list of recent press articles quoting or referencing our work. For more articles also see our [blog](#).

**2012**

- July 31 – mybroadband.co.za "Shocking" IPv6 revelation in South Africa

**2011**

- Feb 3 – Computerworld Egypt reverses 'kill switch' to restore Internet access
- Feb 2 – The Telegraph Egypt restores internet access
- Jan 28 – BBC News Egypt severs internet connection amid growing unrest
- Jan 28 – Der Spiegel Protests in Egypt
- Jan 28 – The Guardian Egypt cuts off internet access
- Jan 28 – CBS news Egypt Goes Offline
- Jan 28 – Networkworld Anatomy of an Internet blackout
- Jan 28 – The Wall Street Journal How Egypt Cut Itself off From the Net
- Jan 28 – CNN Reports say Egypt Web shutdown is coordinated

**2010**

- Sept 27 – itbusiness.ca Businesses, domain registrars dragging feet on IPv6 adoption
- July 8 – Heise.de Die Gefahr ist real
- Apr 9 – The Register China routing snafu briefly mangles interweb
- Apr 8 – CIO.com A Chinese ISP Momentarily Hijacks the Internet
- Apr 8 – The New York Times Chinese ISP Momentarily Hijacks the Internet

**2009**

- Jan 19 – pcworld Six Net Routing Nightmares

Copyright © 2016 BGPmon Network Solutions Inc. 2012. All rights reserved.

Country wide outage in Azerbaijan  
Large scale BGP hijack out of India  
How Hacking Team Helped Italian Special Operations Group with BGP Routing Hijack  
Massive route leak causes Internet slowdown  
BGP Optimizer Causes Thousands Of Fake Routes  
BGPMon Joins OpenDNS  
What caused the Google service interruption?  
BGP routing incidents in 2014, malicious or not?  
BGP hijack incident by Syrian Telecommunications Establishment  
Using BGP data to find Spammers  
What caused today's Internet hiccup  
The Canadian Bitcoin Hijack  
Hijack event today by Indosat  
Turkey Hijacking IP addresses for popular Global DNS providers  
Looking at the sparnaus DDoS from a BGP perspective  
Accidentally stealing the Internet  
Syria shuts down the Internet  
New version of BGPmon.net  
A BGP leak made in Canada  
Internet outage in Lebanon continues into second day  
How the Internet in Australia went down under  
F-Root DNS server moved to Beijing  
Internet Syria offline  
Facebook's detour through China and Korea  
Egypt Back Online



## **Objectives: Network-wide**

- Prefer traffic to go through AT&T over Verizon
- Enforce isolation between two locations
- Don't use our network as transit between A and B
- Traffic must stay within national boundaries
- Don't leak internal services to the outside world
- Adhere to policies in the good times and the bad (failures)

## **Mechanisms: Device-by-Device**

- Local decisions made independently on each device
- Several device-level actions together imply some higher-level behavior
- Failures interact with local decision-making algorithms in complex ways

# Recent Developments in Network Programming

## A new suite of SDN programming languages provide:

- Simpler, centralized programming models
- Network-wide abstractions like paths
- Compositional construction of complex policies from simpler parts
- Examples include:
  - Frenetic [ICFP '11], PANE [HotSDN '12], Fat Tire [HotSDN '13], Pyretic [NSDI '13], Merlin [CoNext '13], FlowLog [NSDI '14], NetKAT [POPL '14], ...

## But they aren't a panacea:

- They consider *intra*-domain routing but not *inter*-domain routing issues
- Programs don't express preferences ahead of time in case of failures
- Their implementations don't exploit existing network infrastructure
- One must still build controller infrastructure that is fault tolerant and scalable — issues often left undiscussed in academic projects

# **Propane:** **Programming a Distributed Control Plane**

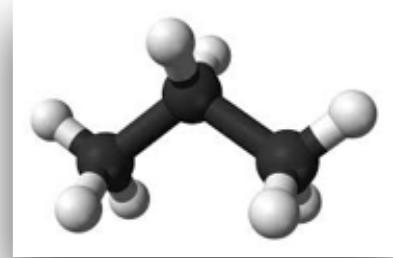
**A language for expression of high-level operator objectives with:**

- regular paths and relative preferences with fall-backs in case of failures
- uniform abstractions for intra- and inter-domain routing
- network-wide constraints that make some low-level errors impossible to express

**And a compiler that bridges the gap between high-level objectives and low-level distributed control plane mechanisms:**

- efficient algorithms to synthesize a set of policy-compliant BGP configs
  - reuse existing infrastructure
  - fault tolerant and scalable
- failure analysis guarantees *policy compliance* under all failures

# This Talk



## Two Examples (simplified but emblematic of the issues)

- Operator objectives and challenges
- Propane solutions

## The Propane Compiler

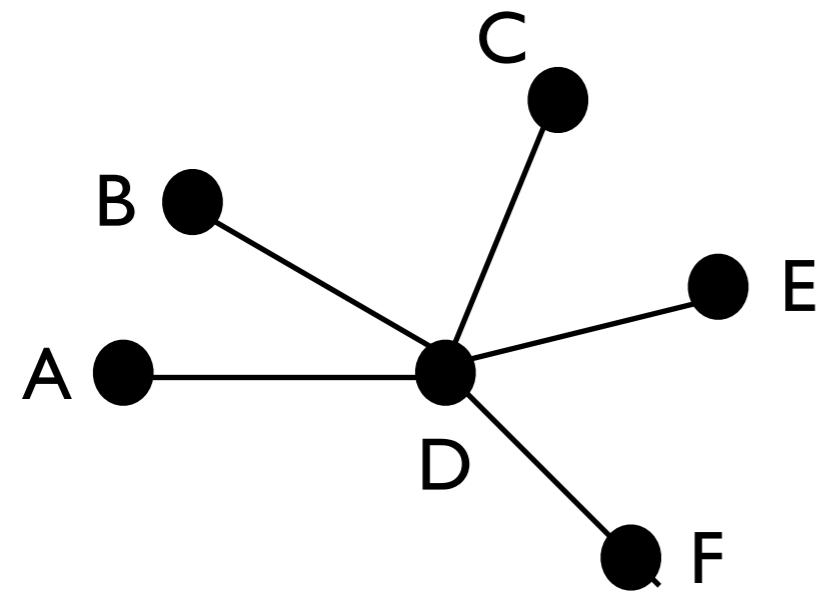
- Intermediate representations
- Compilation steps
- Failure analyses

## Wrap-up

- Future work
- Conclusions

# BGP Protocol

Prefix = 1.2.3.4/32

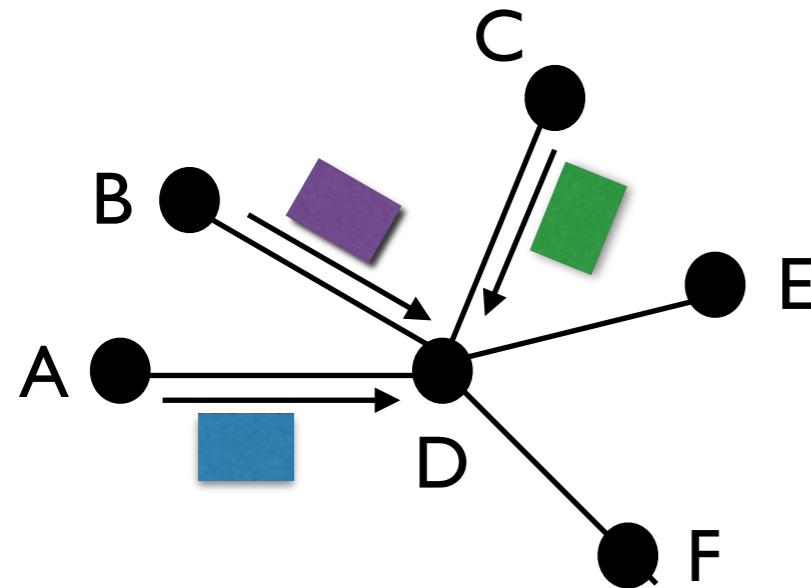


## BGP Nodes:

- receive and filter announcements
  - modifying attributes
- decide on the best route per prefix using announcement attributes:
  - local preference, AS path length, ...
- export and filter routes to neighbors

# BGP Protocol

Prefix = 1.2.3.4/32



## BGP Nodes:

- receive and filter announcements
  - modifying attributes
- decide on the best route per prefix using announcement attributes:
  - local preference, AS path length, ...
- export and filter routes to neighbors

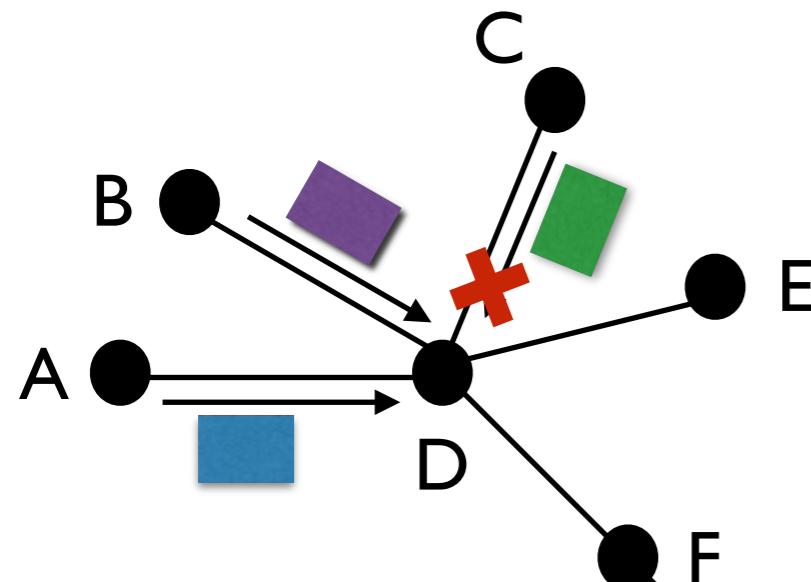
■ {Path=C X, Comm={}, ... }

■ {Path=BY X, Comm={6054:10}, ... }

■ {Path=A X, Comm={6054:10}, ... }

# BGP Protocol

Prefix = 1.2.3.4/32



## BGP Nodes:

- receive and filter announcements
  - modifying attributes
- decide on the best route per prefix using announcement attributes:
  - local preference, AS path length, ...
- export and filter routes to neighbors

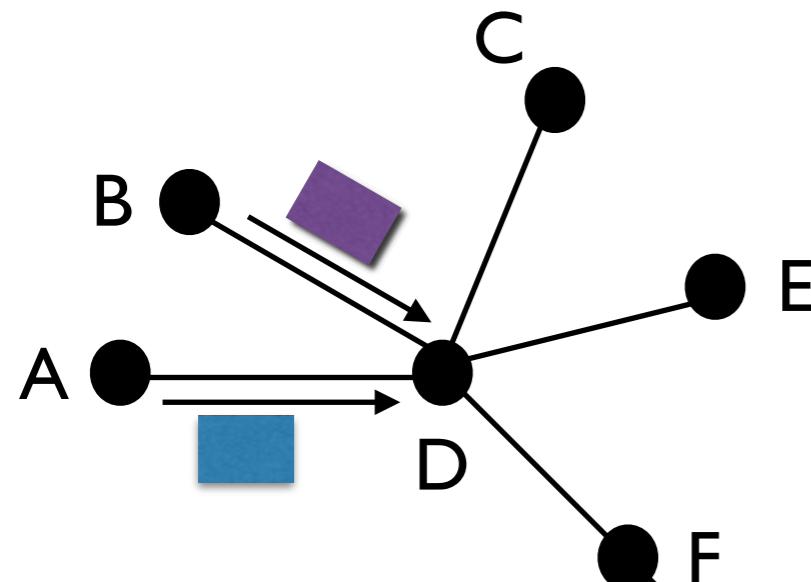
 {Path=C X, Comm={}, ...}

 {Path=B Y X, Comm={6054:10}, ...}

 {Path=A X, Comm={6054:10}, ...}

# BGP Protocol

Prefix = 1.2.3.4/32



## BGP Nodes:

- receive and filter announcements
  - modifying attributes
- decide on the best route per prefix using announcement attributes:
  - local preference, AS path length, ...
- export and filter routes to neighbors

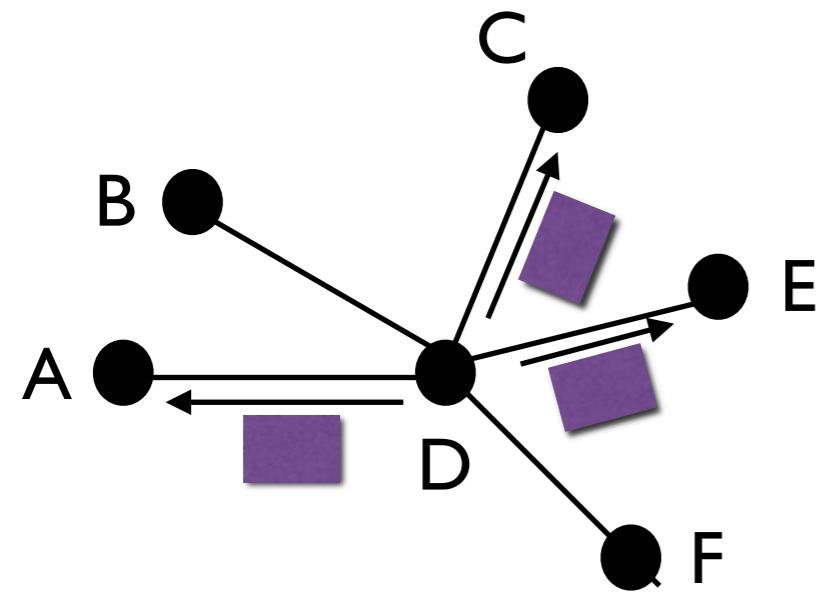
{Path=C X, Comm={}, ...}

{Path=B Y X, Comm={6054:10}, ...} ✓

{Path=A X, Comm={6054:10}, ...}

# BGP Protocol

Prefix = 1.2.3.4/32

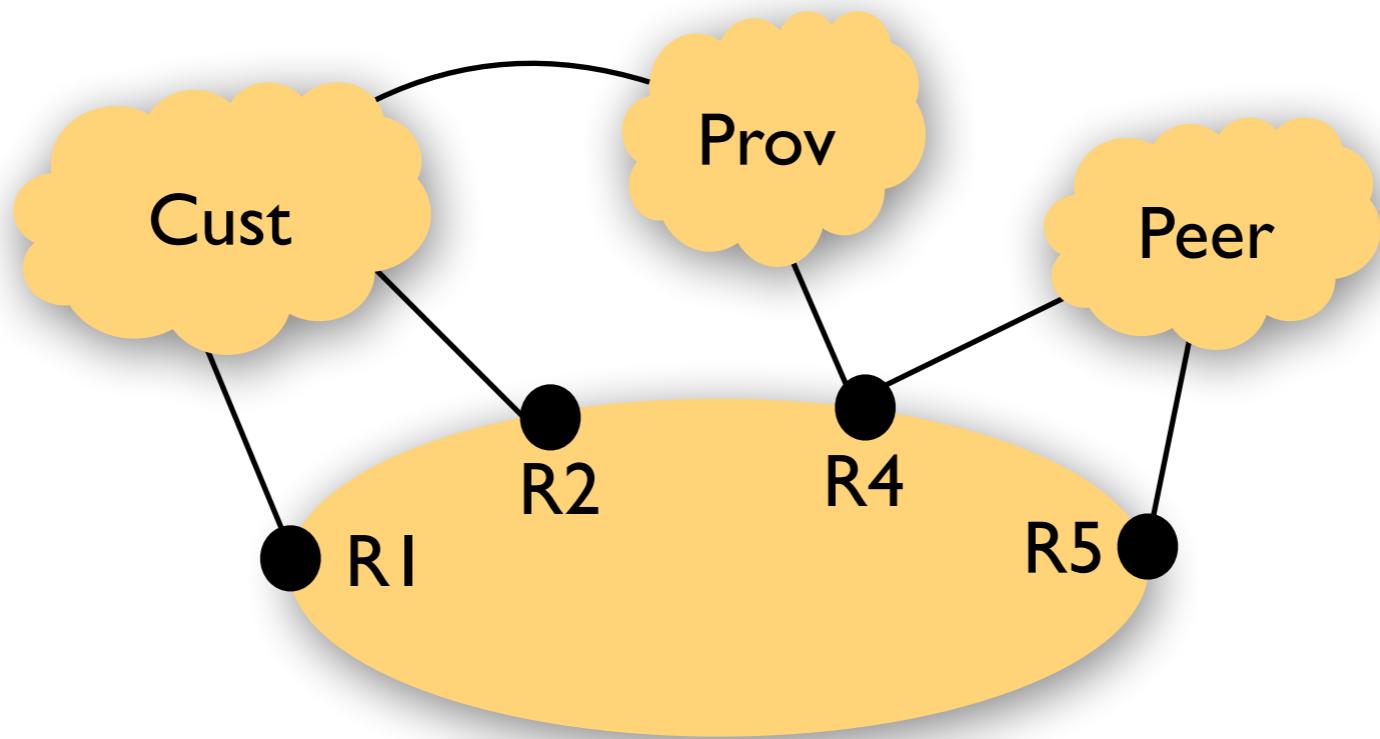


## BGP Nodes:

- receive and filter announcements
  - modifying attributes
- decide on the best route per prefix using announcement attributes:
  - local preference, AS path length, ...
- export and filter routes to neighbors

## **Example I: A Backbone Network**

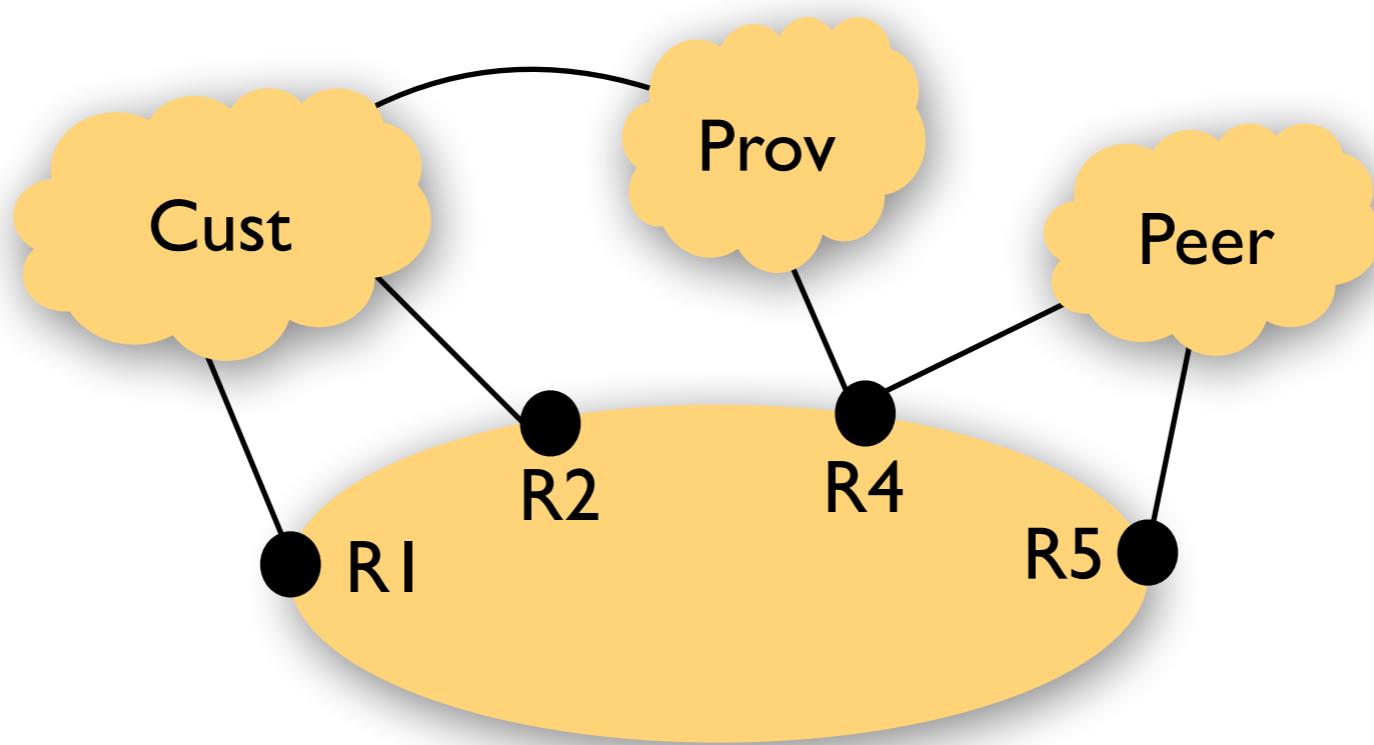
# A Backbone Network



## Goals:

- P1: Prefer to exit thru Cust > Peer > Prov
- P2: Disallow traffic between Prov and Peer
- P3: For Cust, prefer exit thru R1 > R2
- P4: Cust must be on path for its prefixes

# A Backbone Network



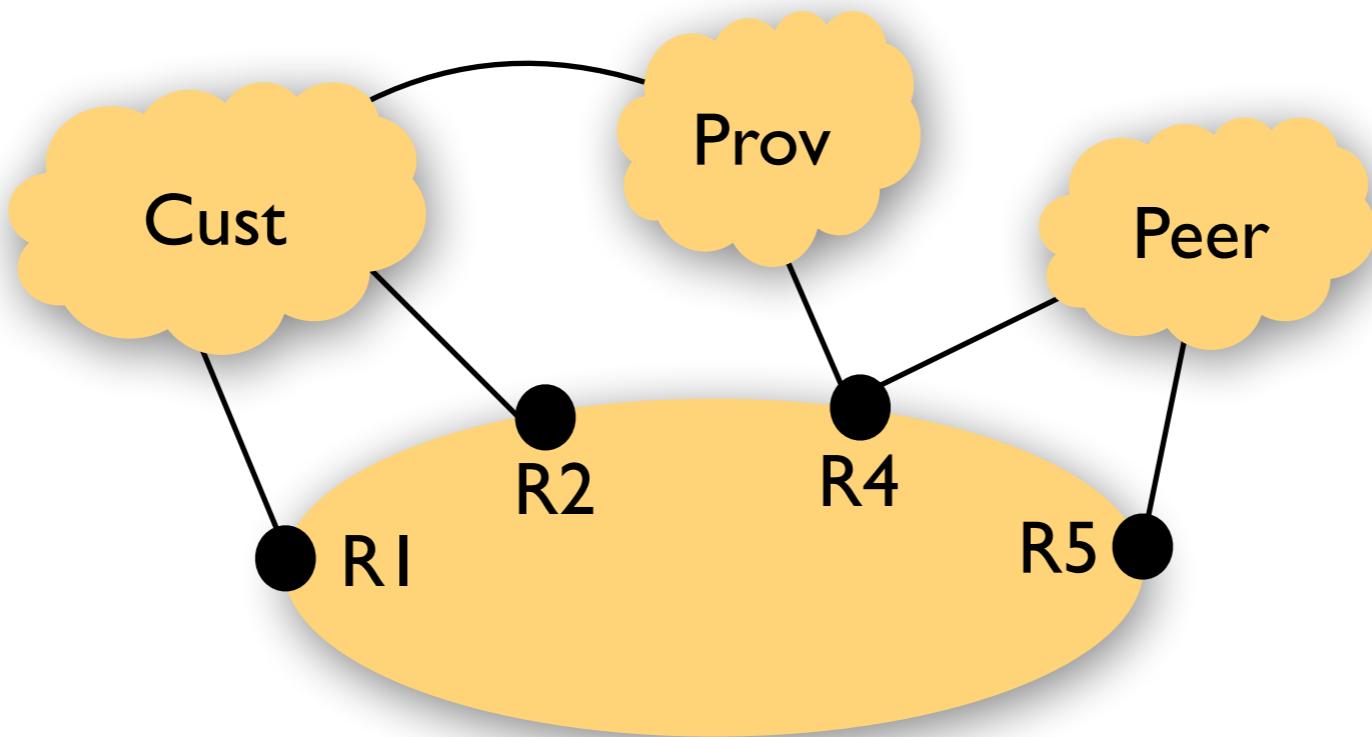
## Goals:

- P1: Prefer to exit thru Cust > Peer > Prov
- P2: Disallow traffic between Prov and Peer
- P3: For Cust, prefer exit thru R1 > R2
- P4: Cust must be on path for its prefixes

## Implementation Techniques:

- Compute and set local preferences consistently at all Peer-facing, Cust-facing, Prov-facing interfaces
- For Cust, ensure R2 local pref is lower than R1

# A Backbone Network



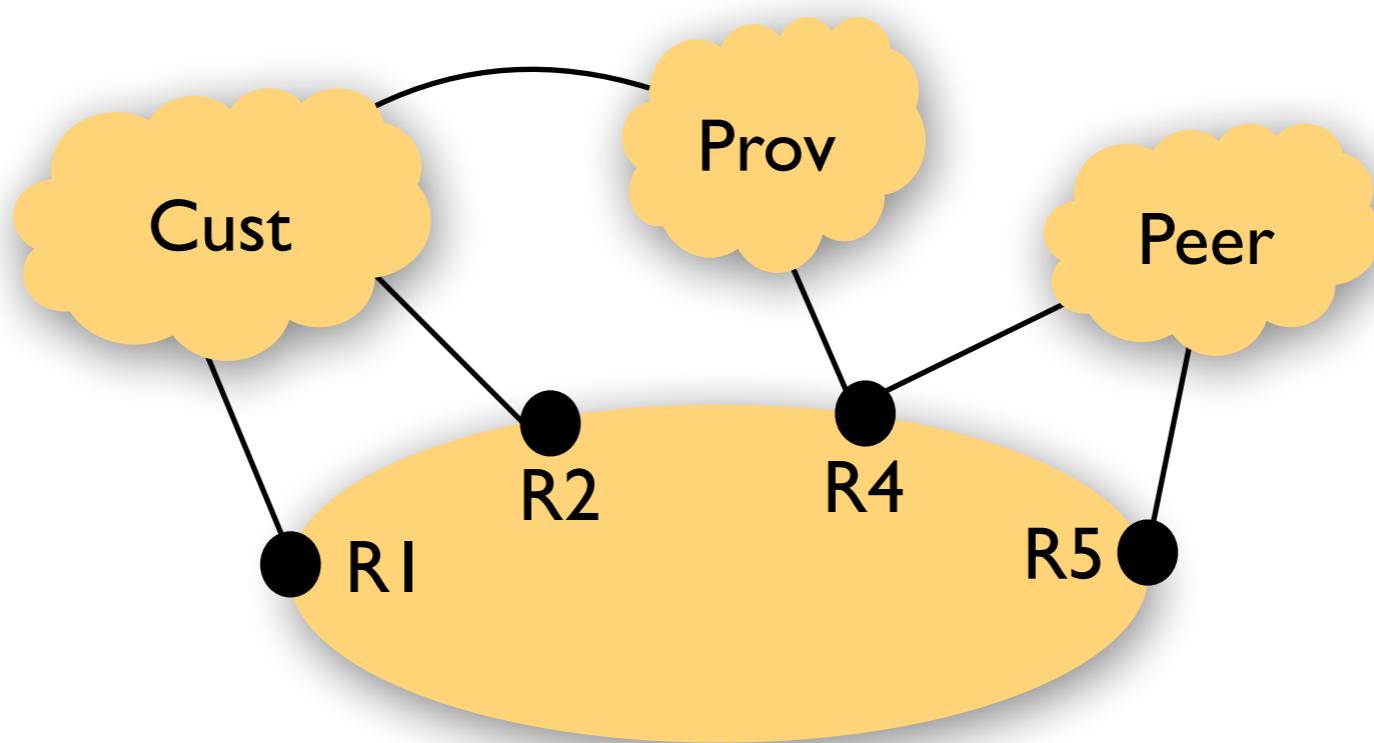
## Goals:

- P1: Prefer to exit thru Cust > Peer > Prov
- P2: Disallow traffic between Prov and Peer
- P3: For Cust, prefer exit thru R1 > R2
- P4: Cust must be on path for its prefixes

## Implementation Techniques:

- Use communities to mark the location an announcement entered the network

# A Backbone Network



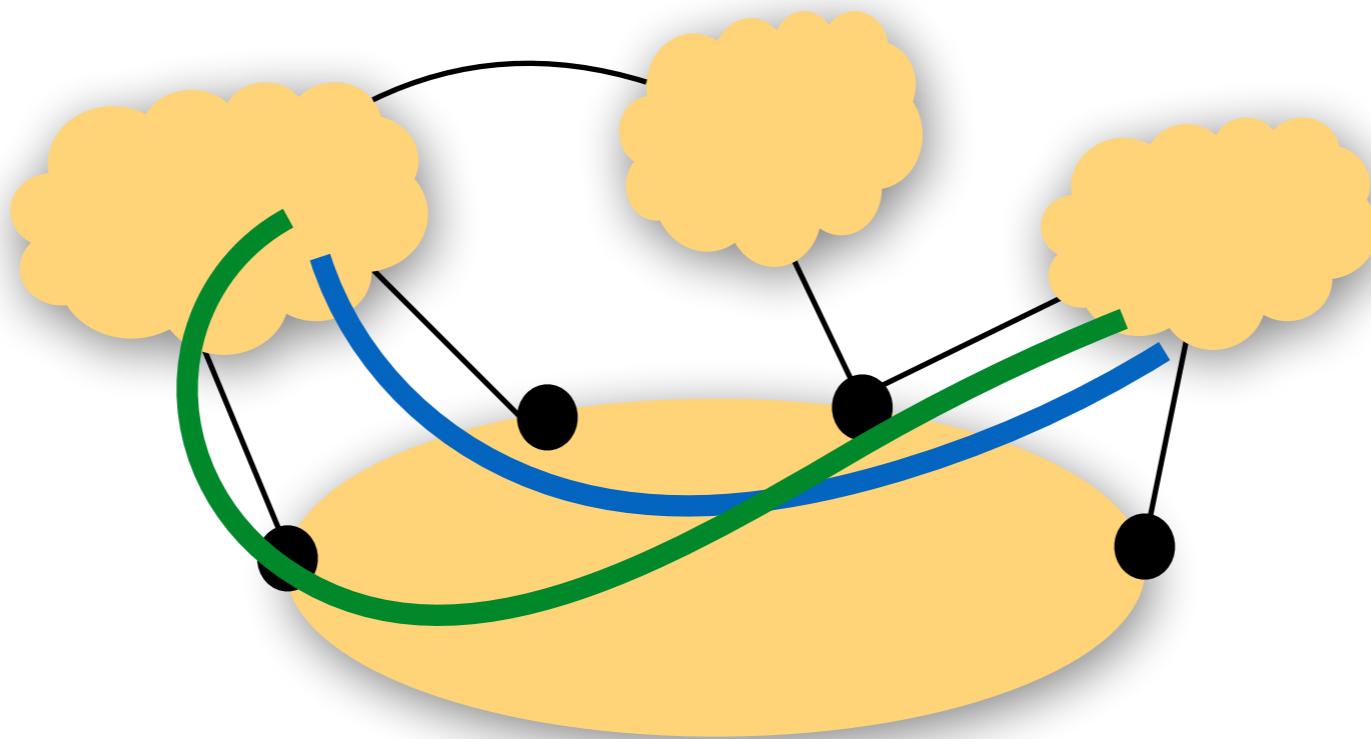
## Goals:

- P1: Prefer to exit thru Cust > Peer > Prov
- P2: Disallow traffic between Prov and Peer
- P3: For Cust, prefer exit thru R1 > R2
- P4: Cust must be on path for its prefixes

## Implementation Techniques:

- More filters to drop undesirable routes

# Propane



A **Propane policy** associates **sets of ranked paths** with prefixes:

- Such paths define the desired flow of traffic
- Paths stretch from other networks, through our network, back out to neighbors
- Preferences between sets of paths express desired behavior when faults occur

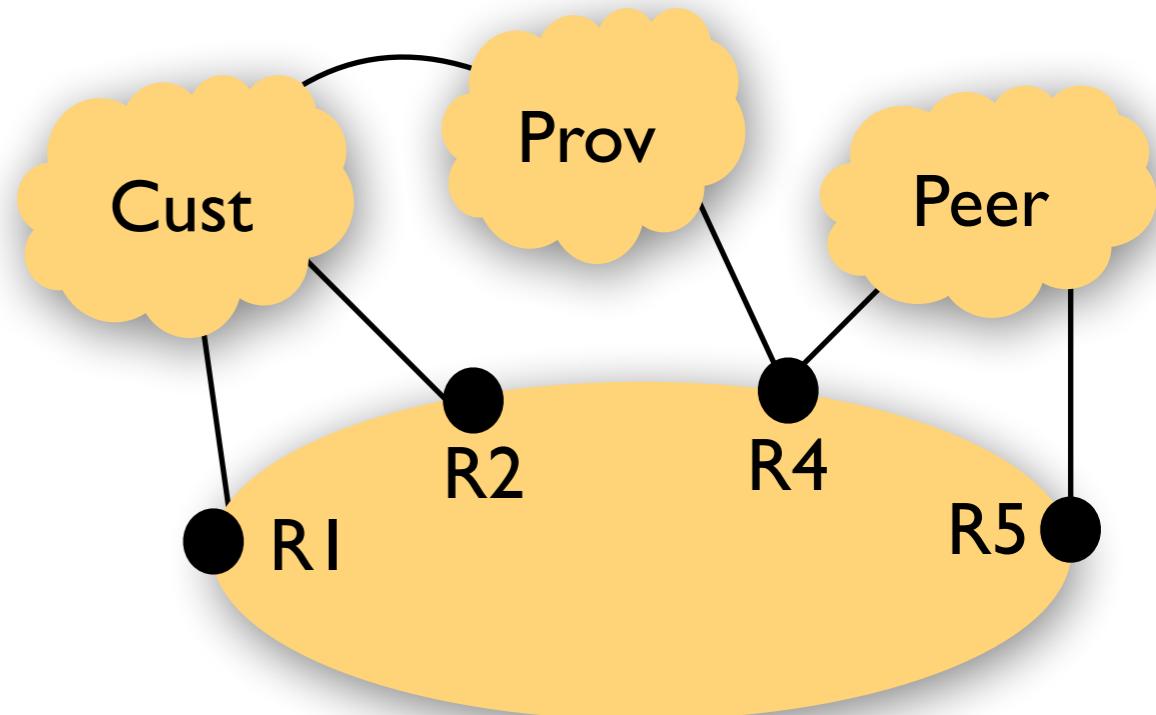
# Propane

P1: Prefer to exit thru Cust > Peer > Prov

P2: Disallow traffic between Prov and Peer

P3: For Cust, prefer exit thru R1 > R2

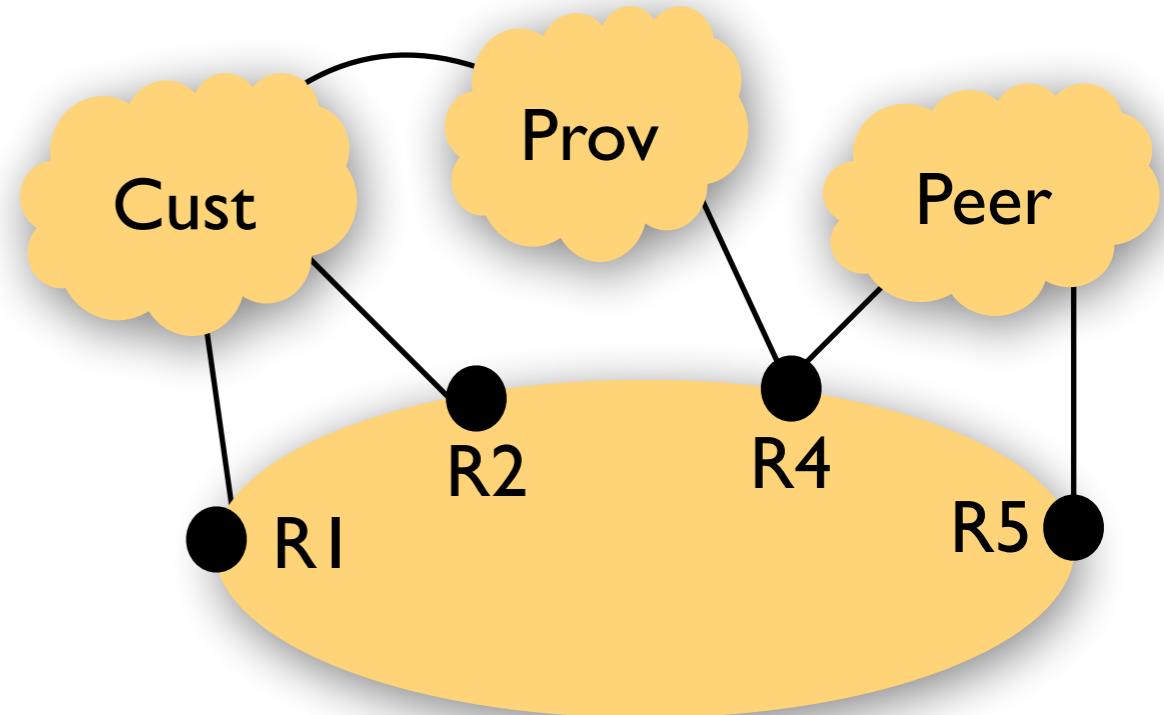
P4: Cust must be on path for its prefixes



```
define Routing = {
    true  => exit(R1 >> R2 >> Peer >> Prov)
}
```

# Propane

- P1: Prefer to exit thru Cust > Peer > Prov
- P2: Disallow traffic between Prov and Peer
- P3: For Cust, prefer exit thru R1 > R2
- P4: Cust must be on path for its prefixes



```
define Routing = {  
    true => exit(R1 >> R2 >> Peer >> Prov)  
}
```

```
define Ownership = {  
    PCust => later(Cust)  
}
```

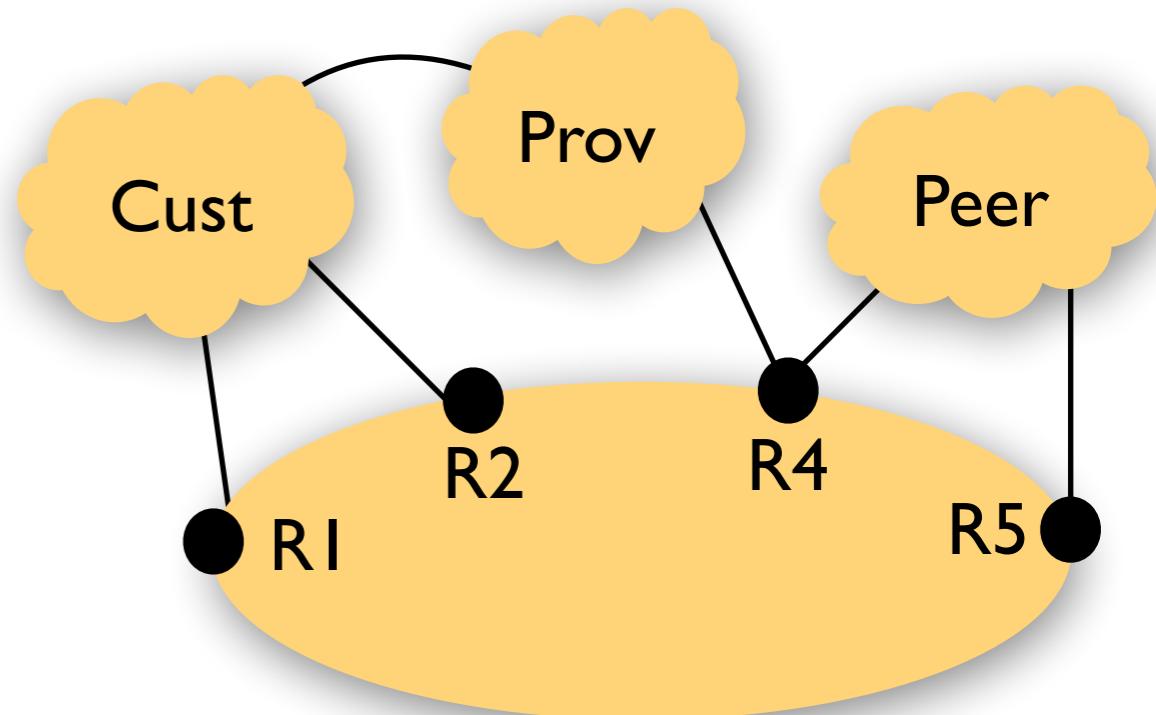
# Propane

P1: Prefer to exit thru Cust > Peer > Prov

P2: Disallow traffic between Prov and Peer

P3: For Cust, prefer exit thru RI > R2

P4: Cust must be on path for its prefixes



```
define transit(X) = enter(X) & exit(X)
```

```
define NoTransit = {  
    true => !transit(Peer|Prov)  
}
```

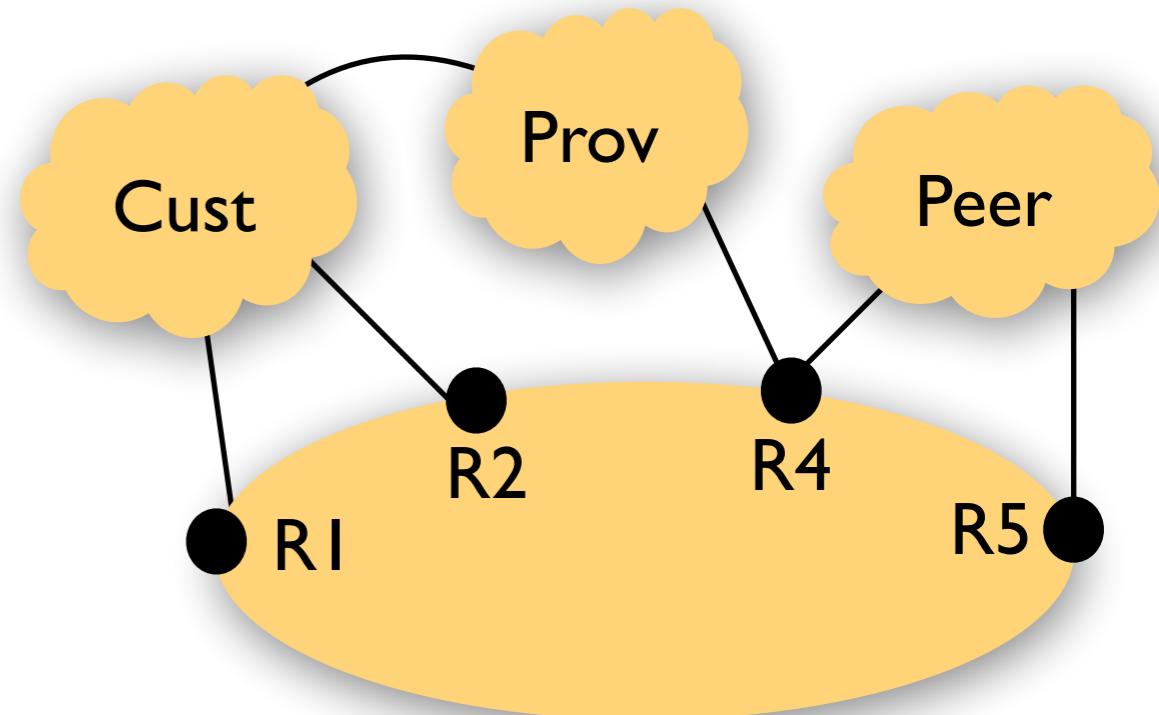
# Propane

P1: Prefer to exit thru Cust > Peer > Prov

P2: Disallow traffic between Prov and Peer

P3: For Cust, prefer exit thru RI > R2

P4: Cust must be on path for its prefixes



```
define transit(X) = enter(X) & exit(X)
```

```
define NoTransit = {
    true => !transit(Peer|Prov)
}
```

```
define Main = Routing & Ownership & NoTransit
```

## **Example II: A Data Center Network**

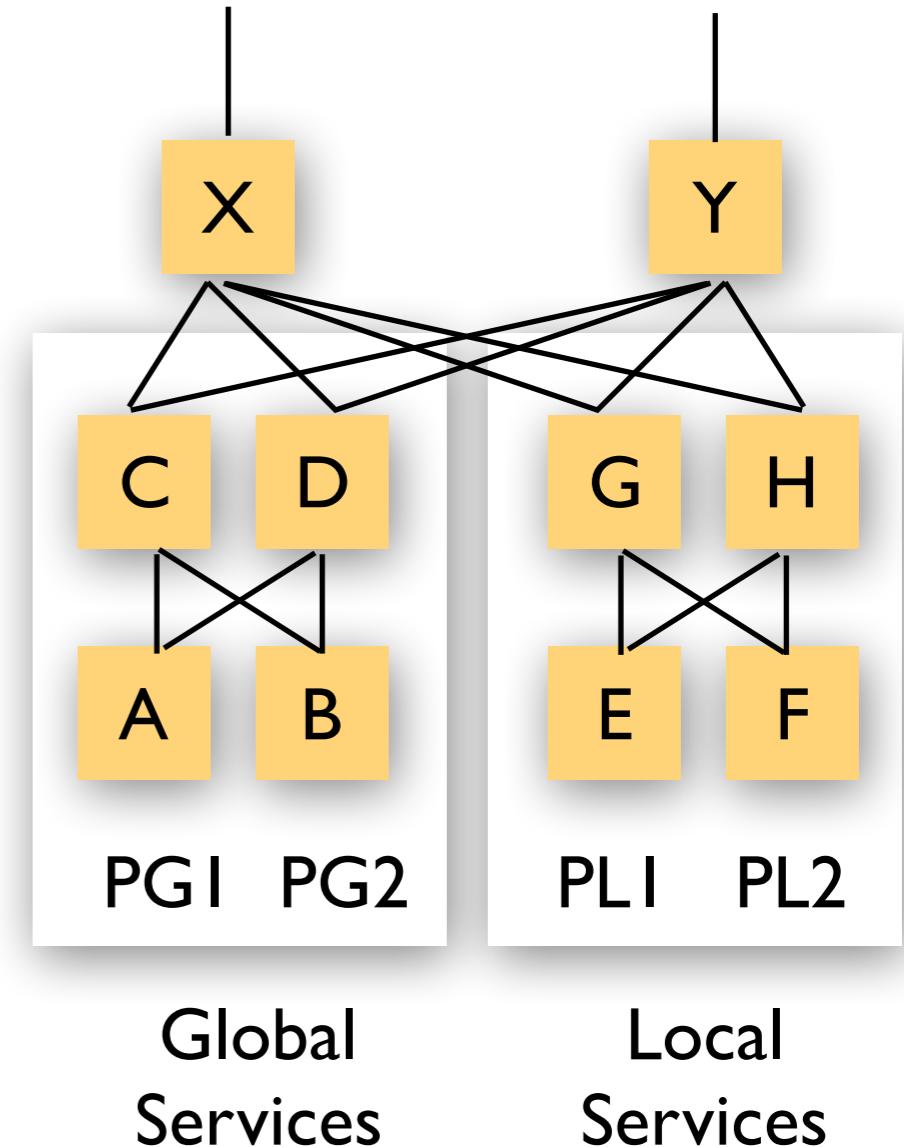
# A Data Center Network

## Goals:

- P1: Announce global services externally as the aggregate PG
- P2: Do not announce local services externally

## Implementation Techniques for X, Y:

- do export announce's from C, D outside
- do *not* export announce's from G,H outside
  - appeal: X,Y do not need to know which prefixes are local vs global
- aggregate to PG if announce is subset of PG



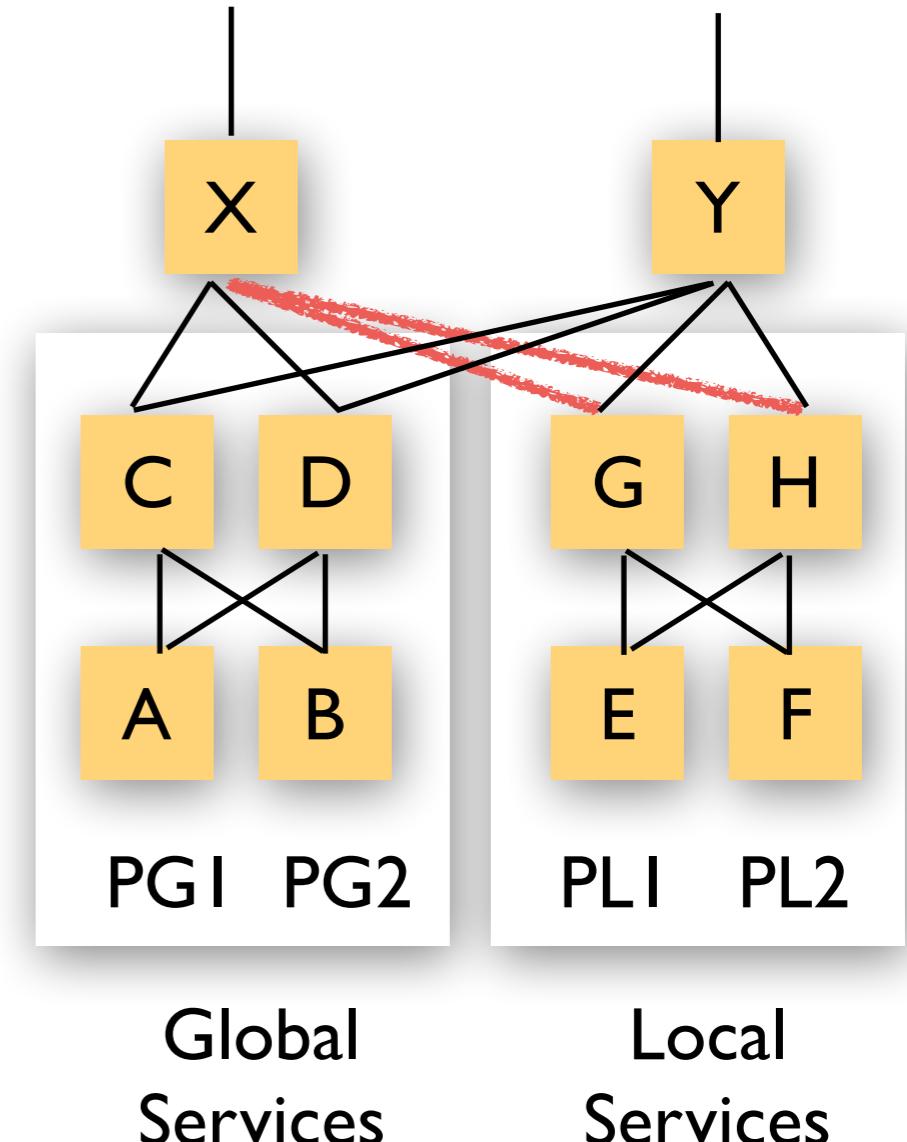
# A Data Center Network

## Goals:

- P1: Announce global services externally as the aggregate PG
- P2: Do not announce local services externally

## Implementation Techniques for X, Y:

- do export announce's from C, D outside
- do *not* export announce's from G,H outside
  - appeal: X,Y do not need to know which prefixes are local vs global
- aggregate to PG if announce is subset of PG



## Consider X-G, X-H Failure:

- PL\* announcements travel H-Y-D-X
- PL\* announcements are then leaked
- Without failure, X selects direct H-X path, which is shorter

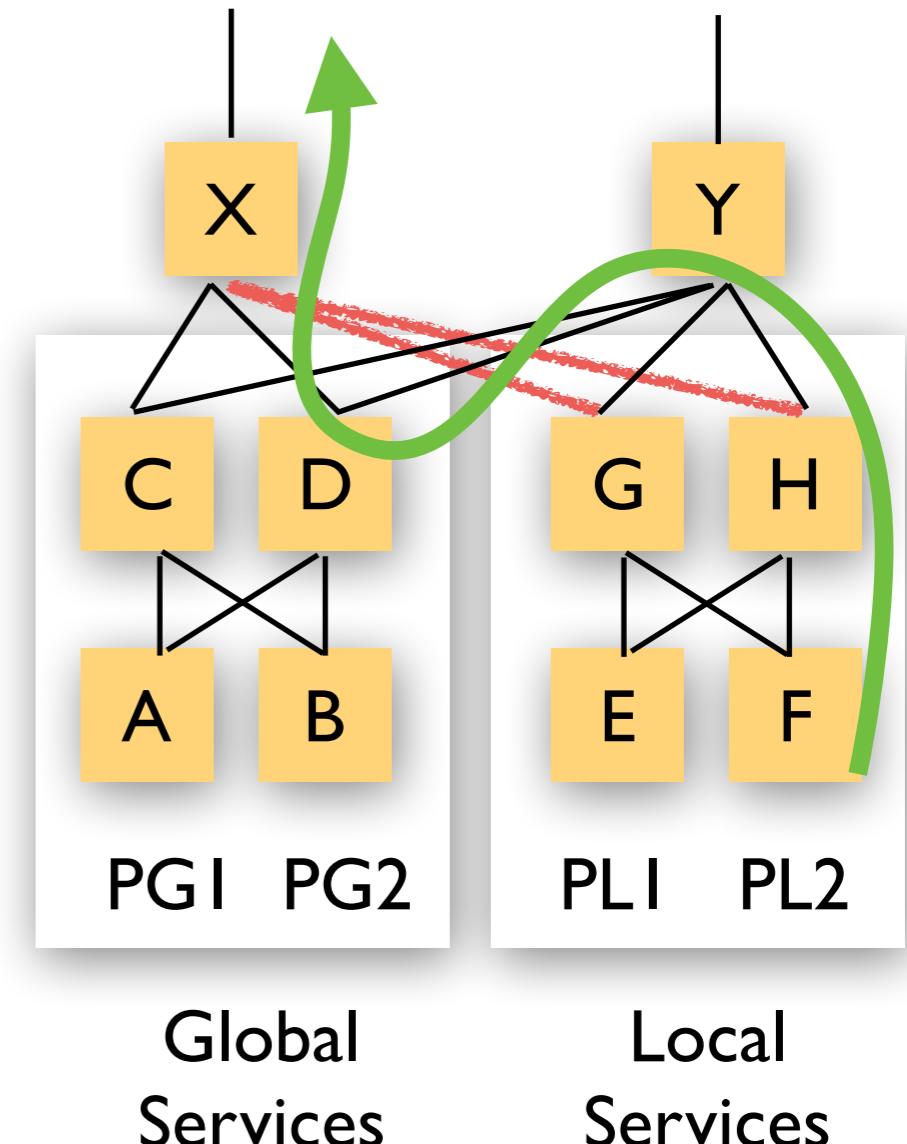
# A Data Center Network

## Goals:

- P1: Announce global services externally as the aggregate PG
- P2: Do not announce local services externally

## Implementation Techniques for X, Y:

- do export announce's from C, D outside
- do *not* export announce's from G,H outside
  - appeal: X,Y do not need to know which prefixes are local vs global
- aggregate to PG if announce is subset of PG



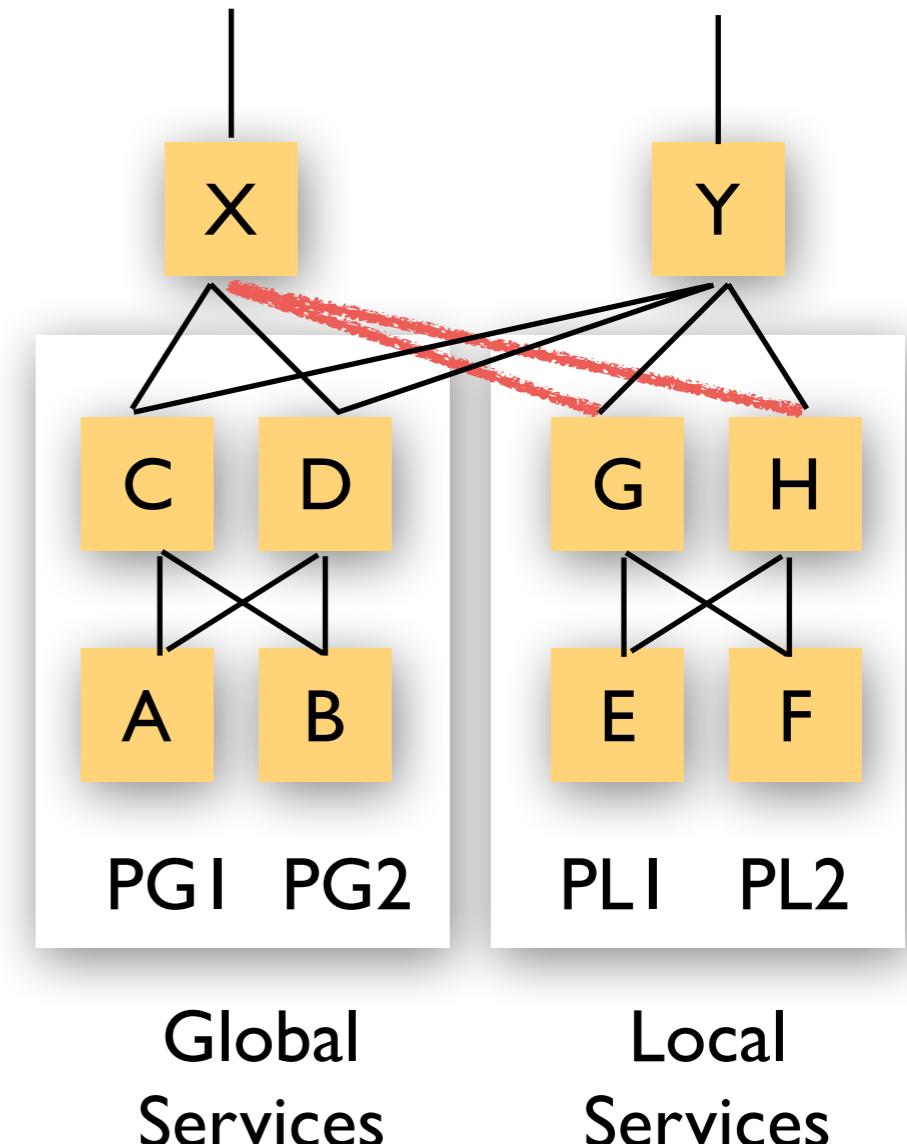
## Consider X-G, X-H Failure:

- PL\* announcements travel H-Y-D-X
- PL\* announcements are then leaked

# A Data Center Network

## Implementation Techniques for X, Y:

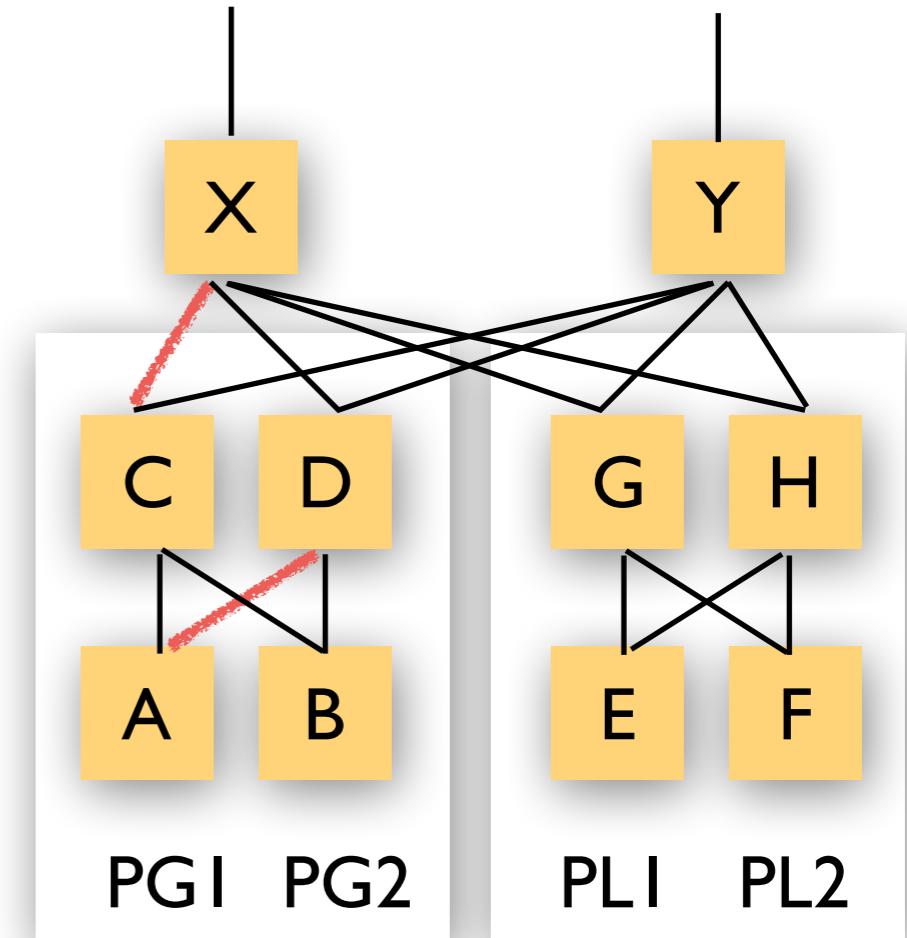
- do export announce's from C, D outside
- do *not* export announce's from G,H outside
  - appeal: X,Y do not need to know which prefixes are local vs global
- aggregate to PG if announce is subset of PG
- disallow “valley” paths



# A Data Center Network

## Implementation Techniques for X, Y:

- do export announce's from C, D outside
- do *not* export announce's from G,H outside
  - appeal: X,Y do not need to know which prefixes are local vs global
- aggregate to PG if announce is subset of PG
- disallow “valley” paths



## Consider D-A, X-C Failure:

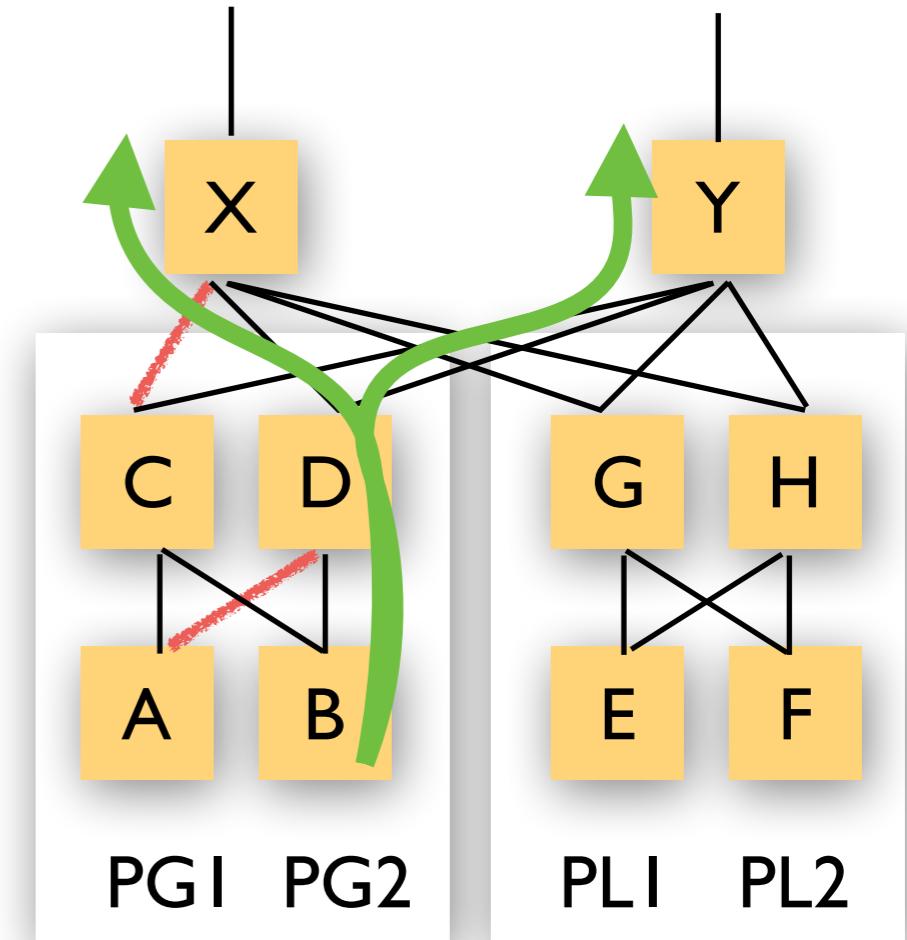
Global  
Services

Local  
Services

# A Data Center Network

## Implementation Techniques for X, Y:

- do export announce's from C, D outside
- do *not* export announce's from G,H outside
  - appeal: X,Y do not need to know which prefixes are local vs global
- aggregate to PG if announce is subset of PG
- **disallow “valley” paths**



## Consider D-A, X-C Failure:

- X and Y will hear PG2

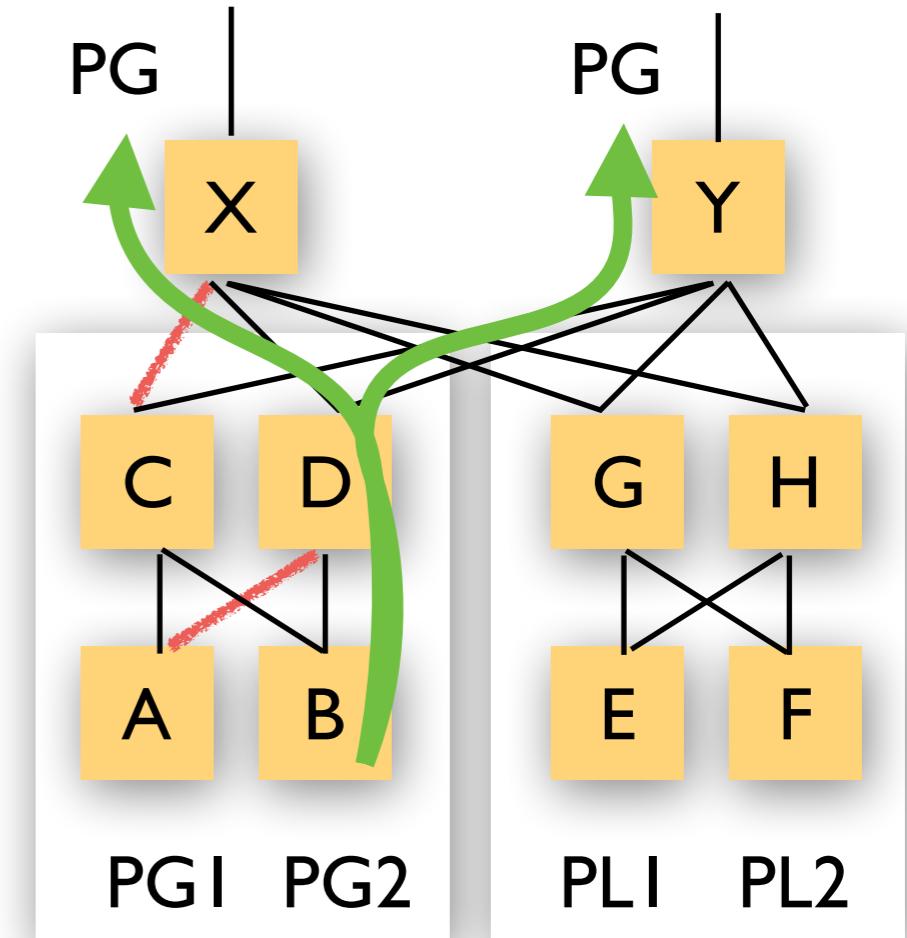
Global  
Services

Local  
Services

# A Data Center Network

## Implementation Techniques for X, Y:

- do export announce's from C, D outside
- do *not* export announce's from G,H outside
  - appeal: X,Y do not need to know which prefixes are local vs global
- aggregate to PG if announce is subset of PG
- **disallow “valley” paths**



## Consider D-A, X-C Failure:

- X and Y will hear PG2
- X and Y will announce aggregate PG

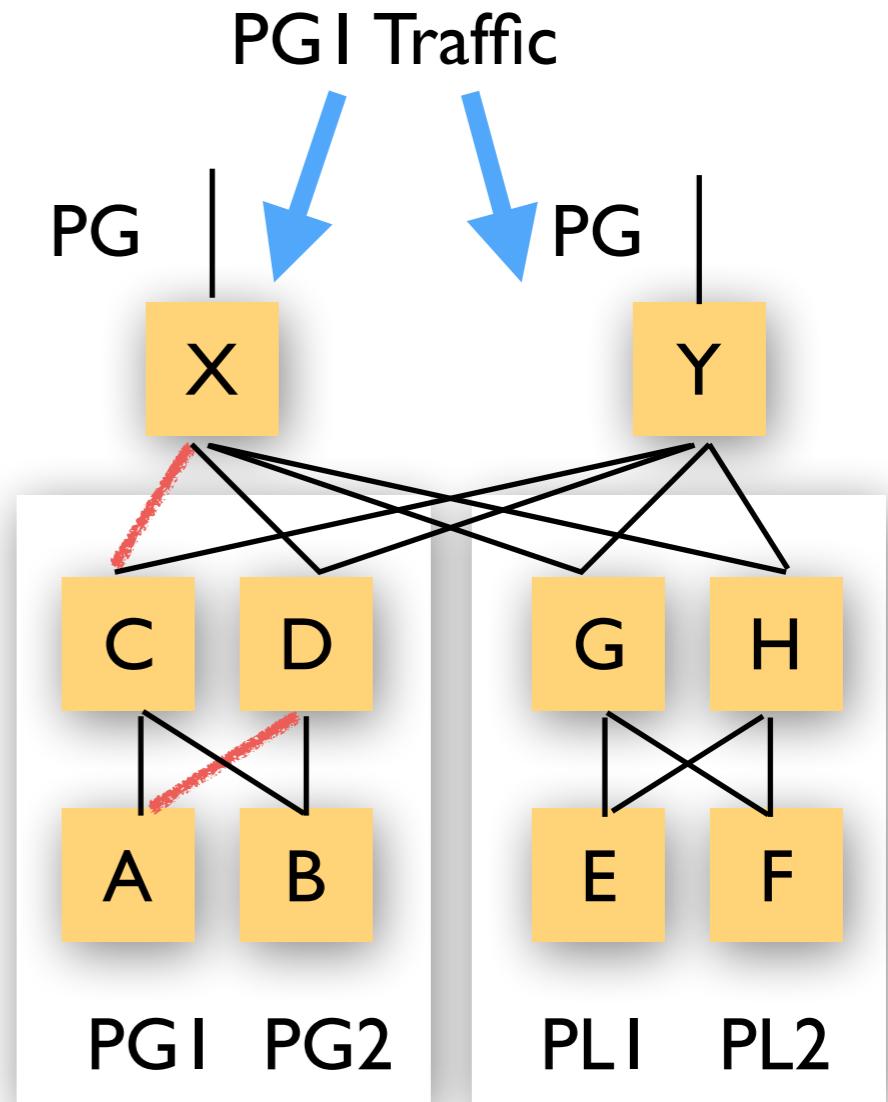
Global  
Services

Local  
Services

# A Data Center Network

## Implementation Techniques for X, Y:

- do export announce's from C, D outside
- do *not* export announce's from G,H outside
  - appeal: X,Y do not need to know which prefixes are local vs global
- aggregate to PG if announce is subset of PG
- **disallow “valley” paths**



## Consider D-A, X-C Failure:

- X and Y will hear PG2
- X and Y will announce aggregate PG
- But PGI is inaccessible through X because there is no valley routing
- An aggregation-induced black hole is created [See Le et al, CoNext '11]

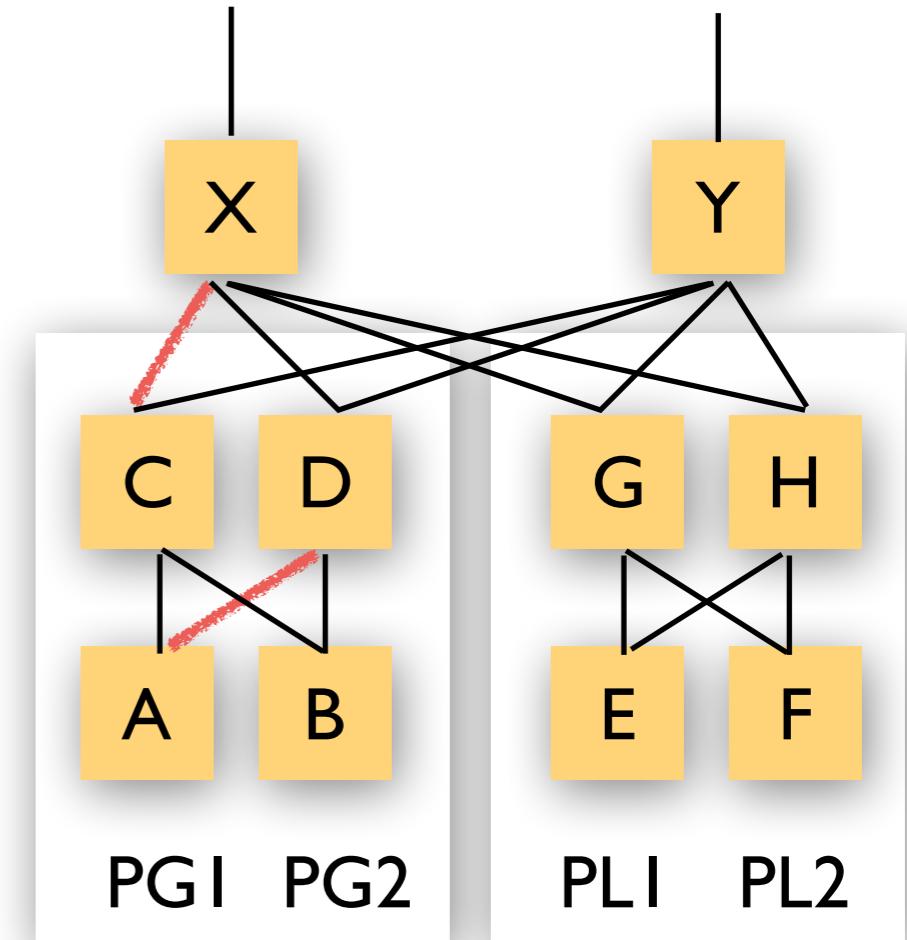
Global  
Services

Local  
Services

# A Data Center Network

## Implementation Techniques for X, Y:

- do export announce's from C, D outside
- do *not* export announce's from G,H outside
  - appeal: X,Y do not need to know which prefixes are local vs global
- aggregate to PG if announce is subset of PG
- disallow “valley” paths



## Moral:

- Reasoning about the interactions between failures and the device-level configs is hard
- Not a good idea for humans to be doing that

Global  
Services

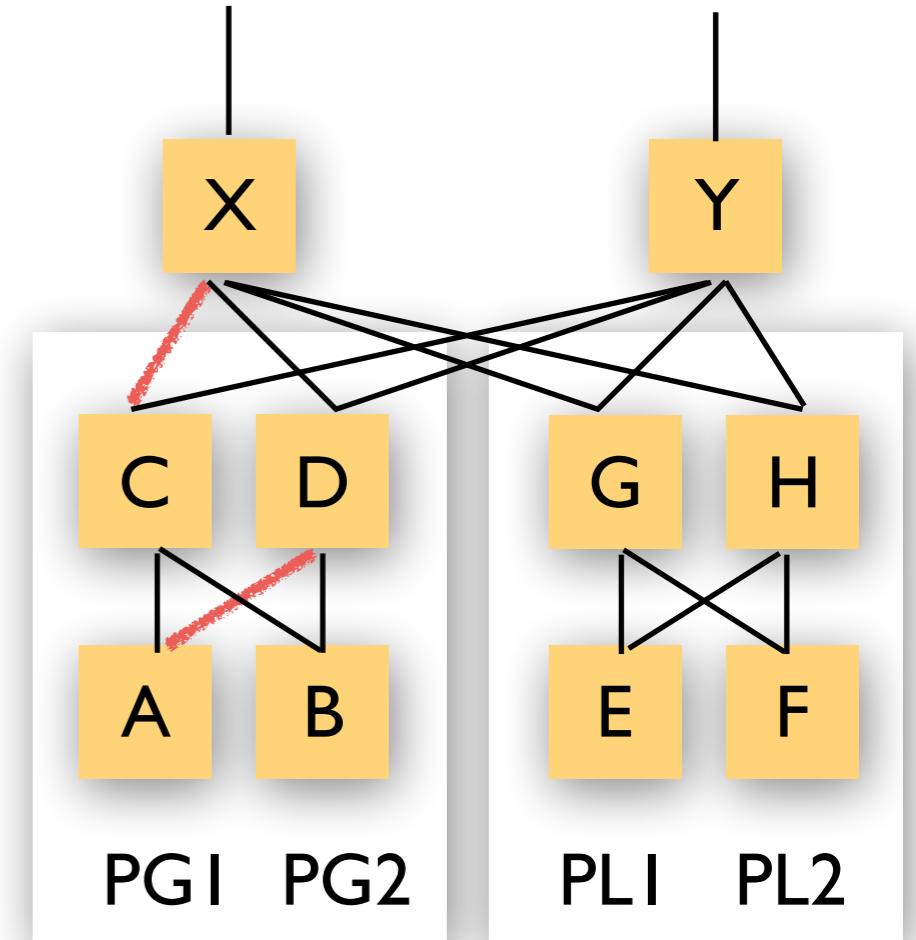
Local  
Services

# A Data Center Network

P1: Traffic for prefixes for each TOR router must reach the router

P2: Do not announce local services externally

P3: Aggregation must be performed on global prefixes to reduce churn



Global  
Services

Local  
Services

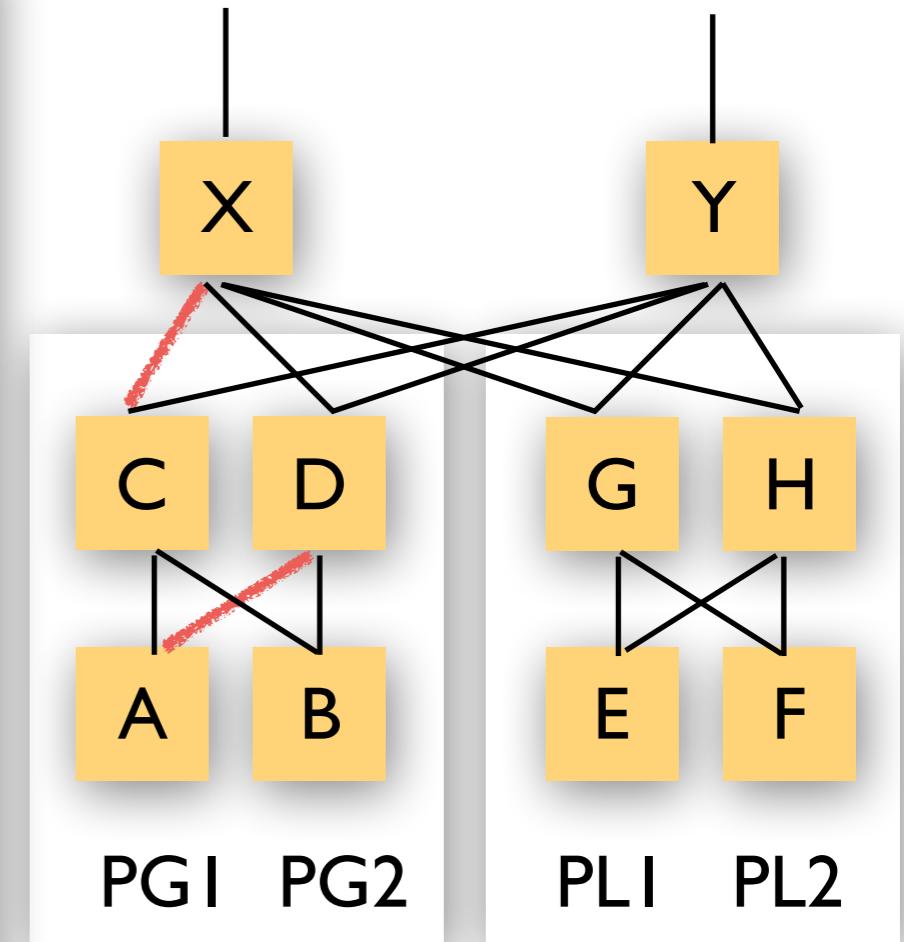
# A Data Center Network

P1: Traffic for prefixes for each TOR router must reach the router

P2: Do not announce local services externally

P3: Aggregation must be performed on global prefixes to reduce churn

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



Global  
Services

Local  
Services

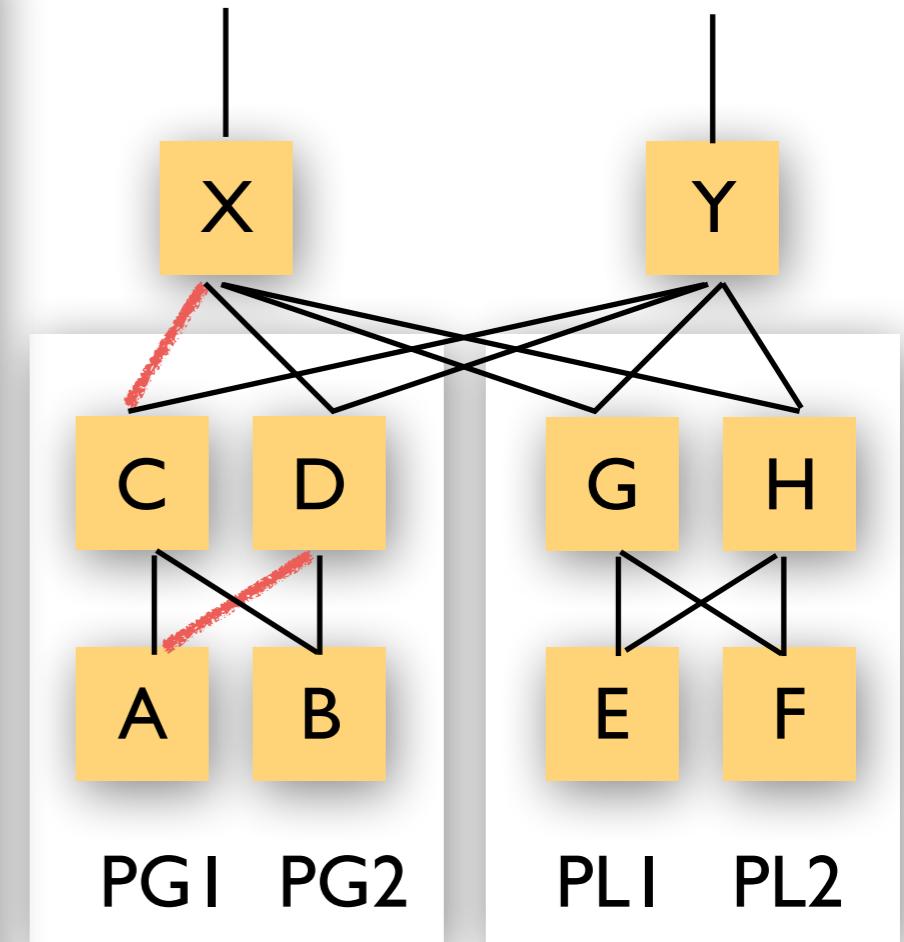
# A Data Center Network

P1: Traffic for prefixes for each TOR router must reach the router

P2: Do not announce local services externally

P3: Aggregation must be performed on global prefixes to reduce churn

```
define Ownership = {  
    PG1 => end(A)  
    PG2 => end(B)  
    PL1 => end(E)  
    PL2 => end(F)  
}  
  
define Routing = {  
    PL1 | PL2 => always(in)  
}
```



Global  
Services

Local  
Services

# A Data Center Network

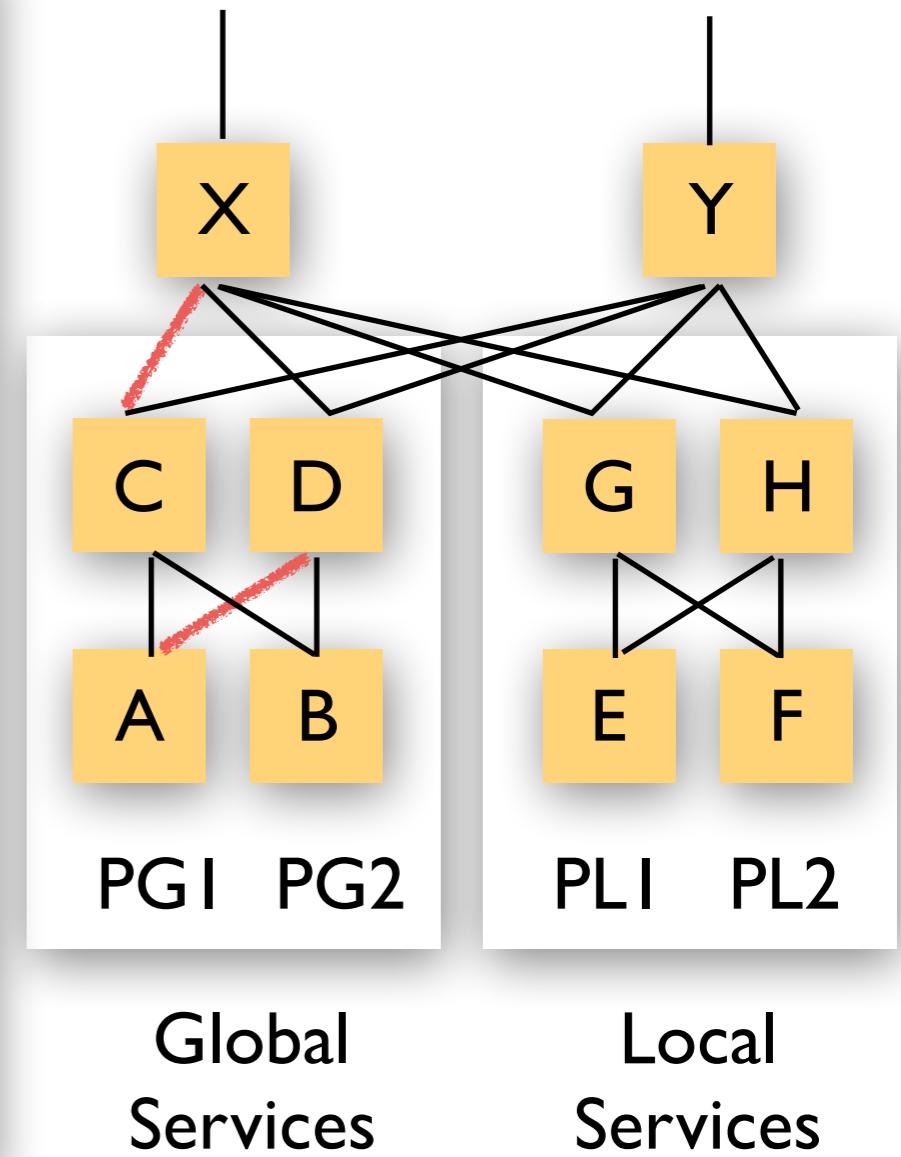
P1: Traffic for prefixes for each TOR router must reach the router

P2: Do not announce local services externally

P3: Aggregation must be performed on global prefixes to reduce churn

```
define Ownership = {  
    PG1 => end(A)  
    PG2 => end(B)  
    PL1 => end(E)  
    PL2 => end(F)  
}  
  
define Routing = {  
    PL1 | PL2 => always(in)  
}  
  
define Main =  
    Ownership & Routing &  
    agg(PG, in -> out)
```

compiler  
discovers  
lower bound  
on # failures  
required to  
induce a  
black hole



# Propane Language Summary

## Programmers express high-level objectives via:

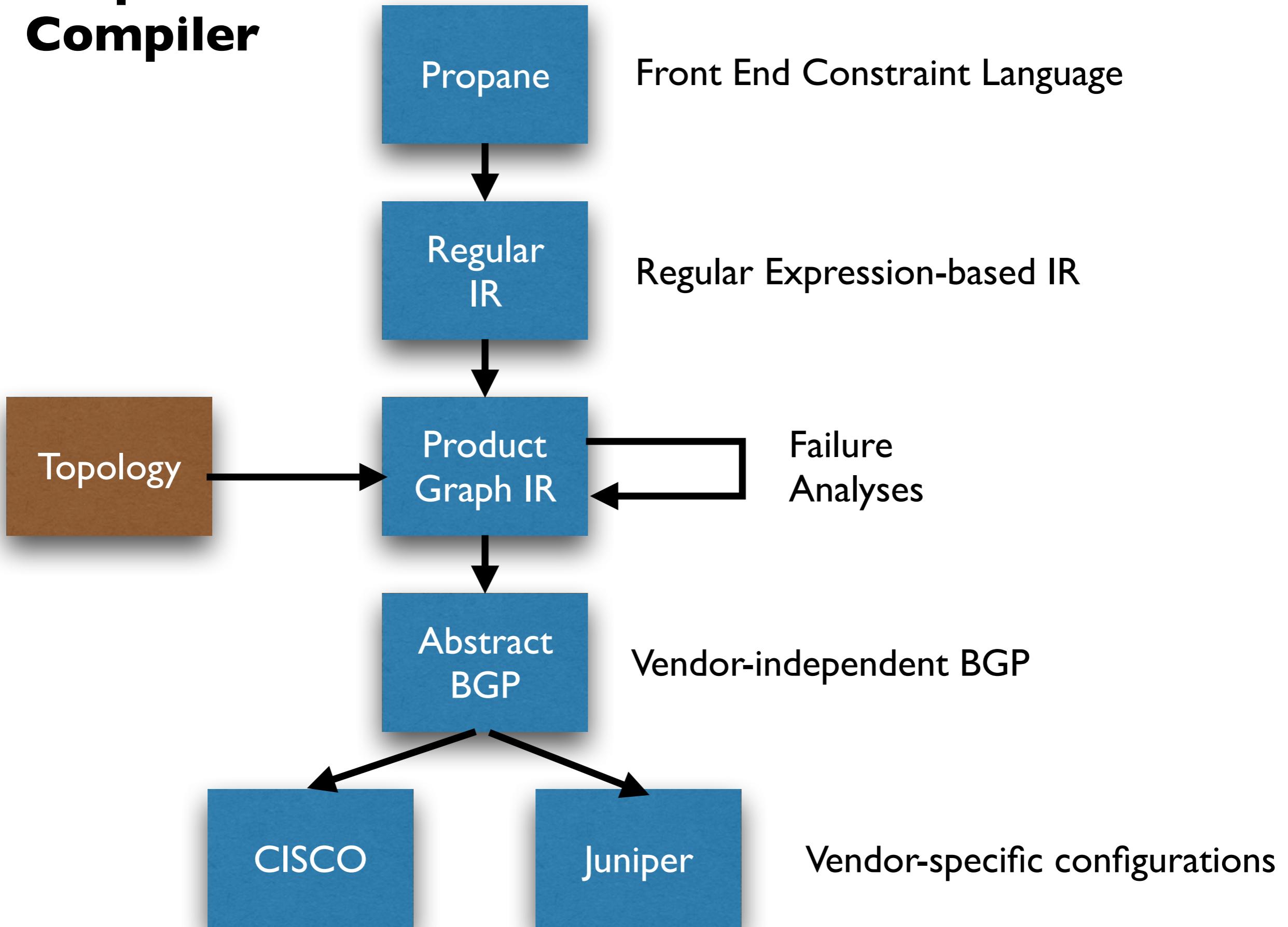
- Sets of inter- and intra-domain paths
- ... defined by high-level abstractions: `enter`, `exit`, `always`, `end`, ...
- ... as well as preferences between those paths for failures

## Programmers may also use:

- a small number of control constraints (`agg`)
- ... while counting on the compiler for failure analysis
- ... and combine constraints together using natural logical combinators

# **Compiling Propane**

# Propane Compiler



# Propane Regular IR

Propane

**Expand constraints in to regular expressions. EG:**

Regular  
IR

**any** = **out**<sup>\*</sup>.**in**<sup>+</sup>.**out**<sup>\*</sup>

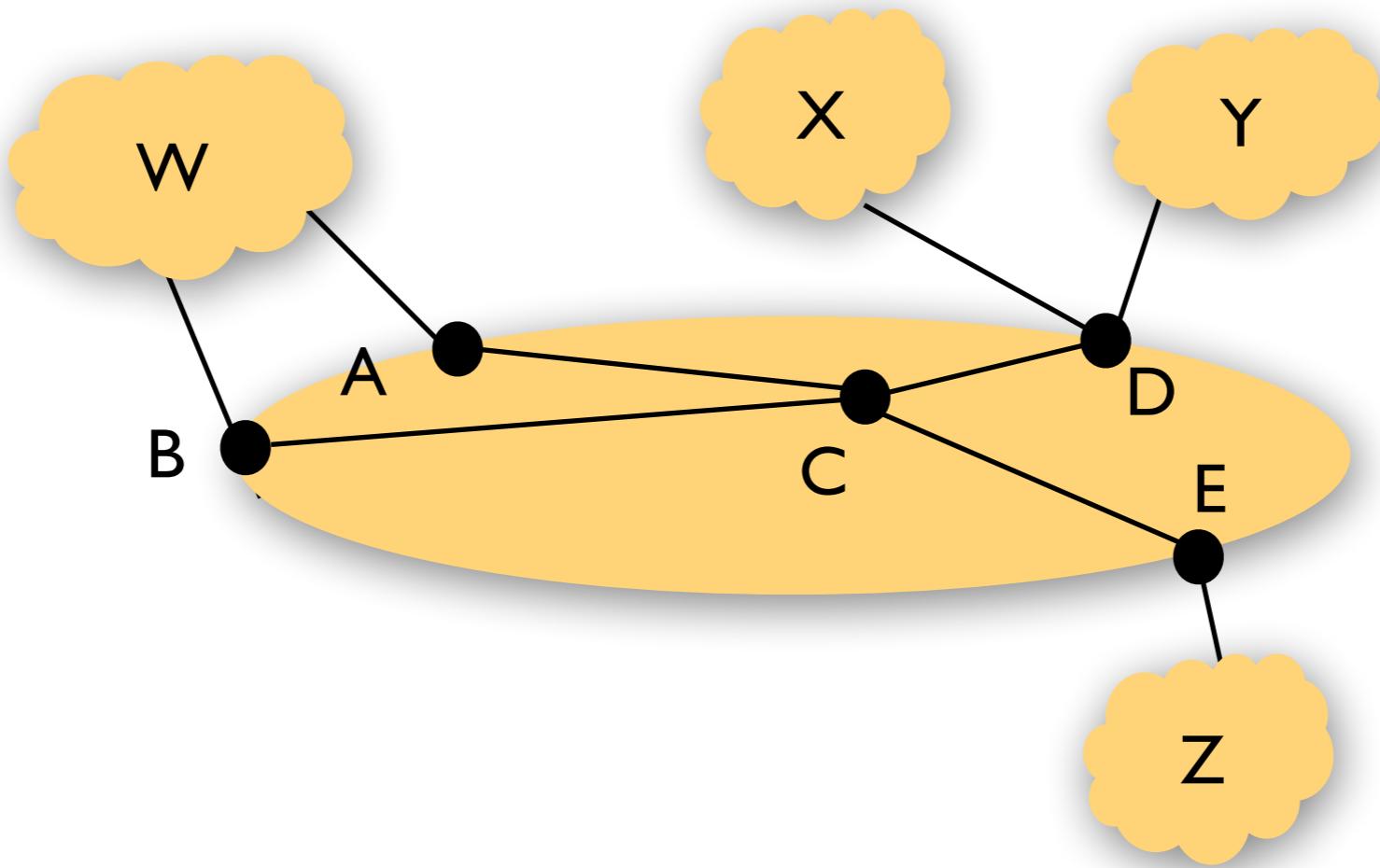
**end**(X) = **any**  $\cap$  ( $\Sigma^*$ .X)

**exit**(X) = (**out**<sup>\*</sup>.**in**<sup>\*</sup>.(X  $\cap$  **in**).**out**<sup>+</sup>)  $\cup$   
(**out**<sup>\*</sup>.**in**<sup>+</sup>.(X  $\cap$  **out**).**out**<sup>\*</sup>)

**A few other simple transformations:**

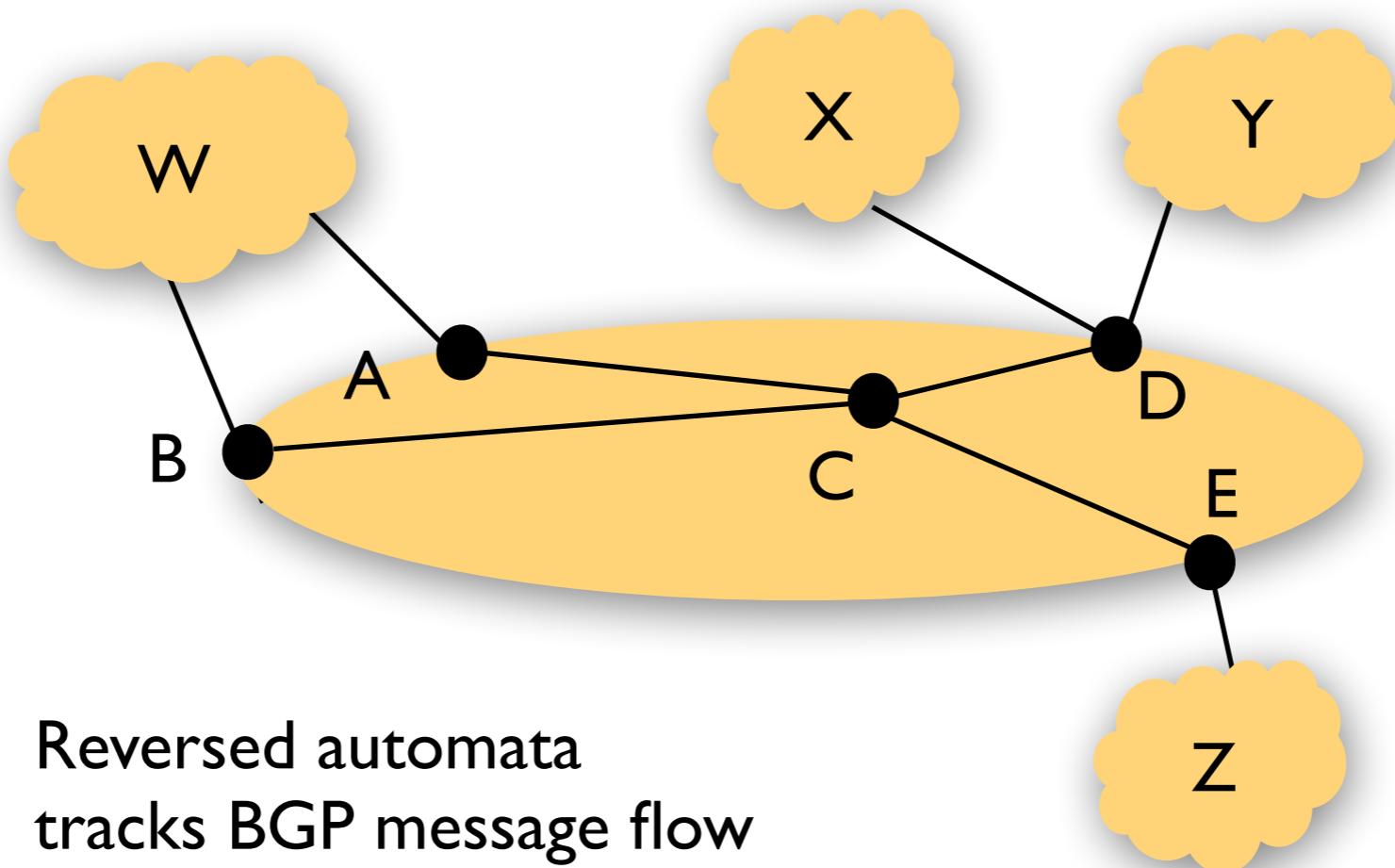
- conjunction of constraints ==> intersection of regular expressions
- conjunction of policies ==> prefix-by-prefix intersection
- nested preferences lifted: (x >> y) . z ==> (x.z) >> (y.z)

# Compilation: An Idealized Example



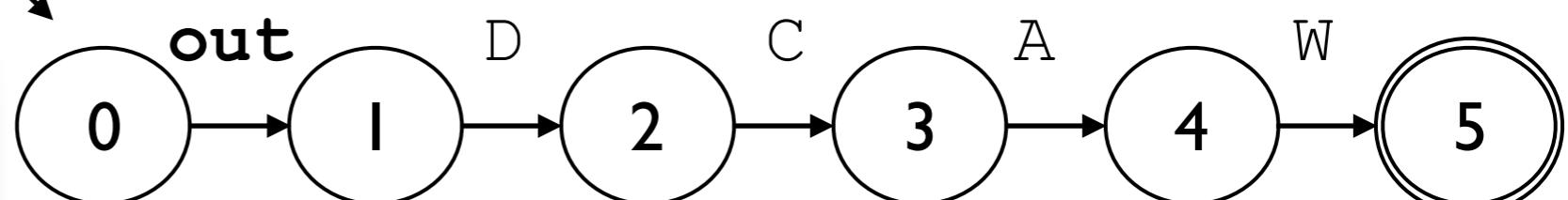
```
(W.A.C.D.out) >> (W.B.in+.out)
```

# Reversed Automata from Policies

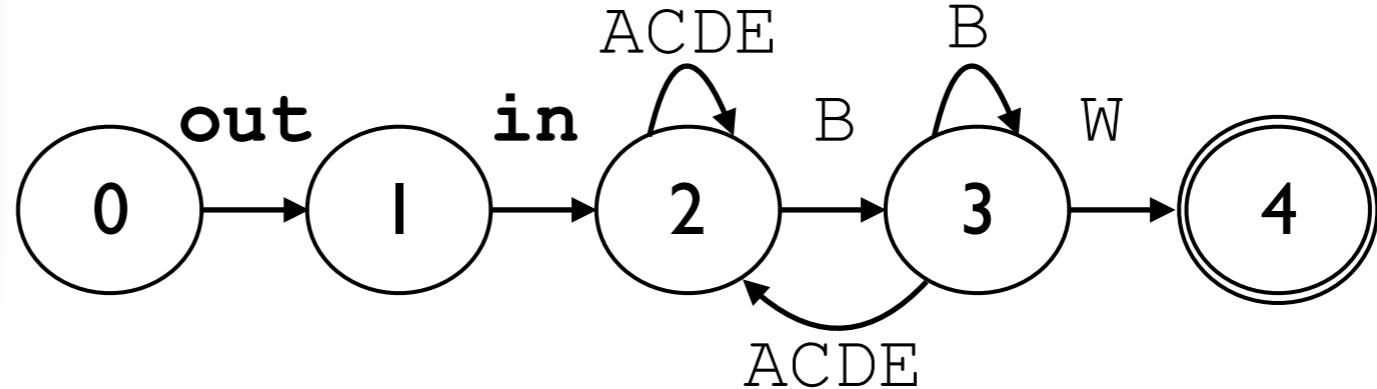


Policies

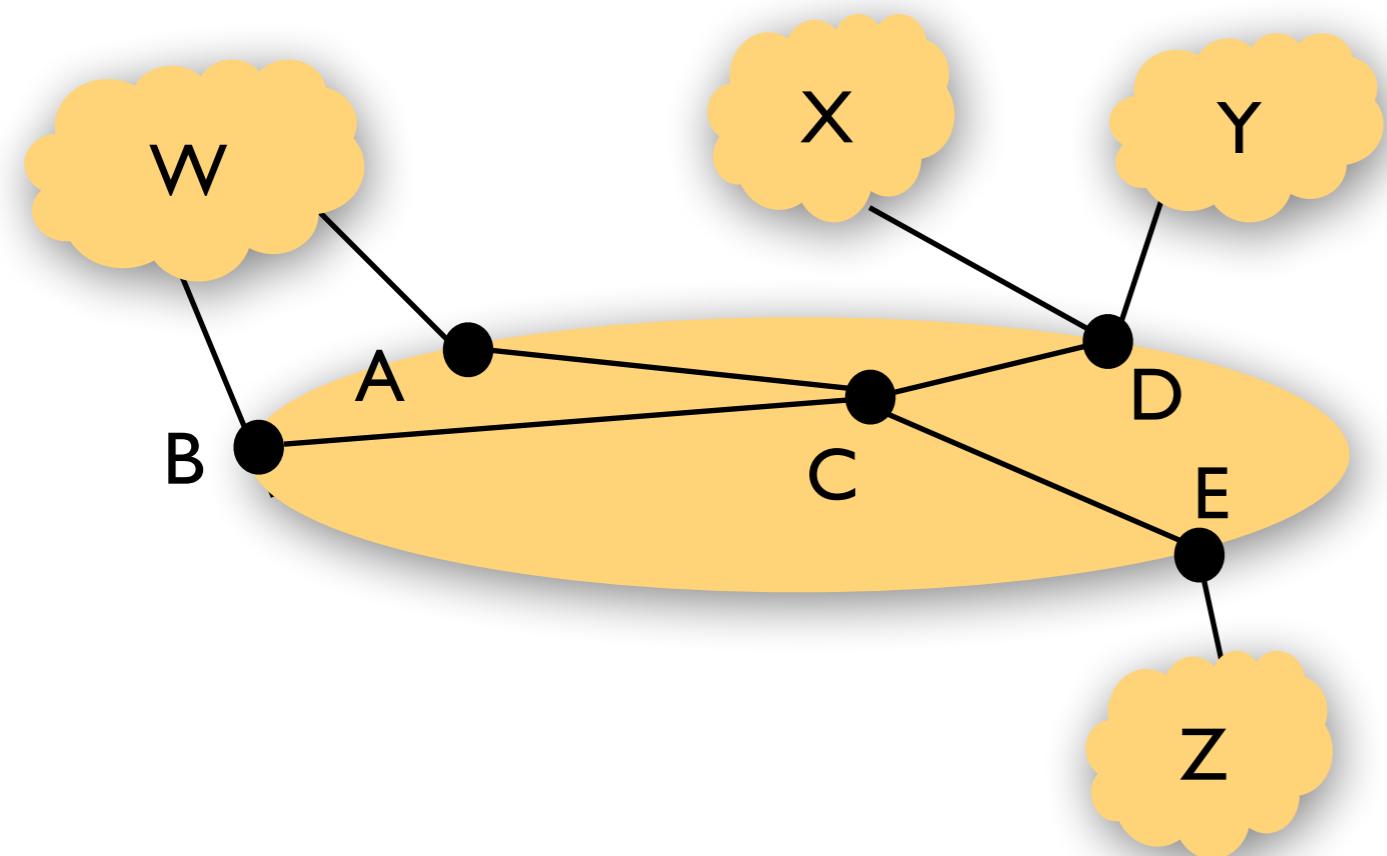
1. (W.A.C.D.out)



2. (W.B.in+.out)



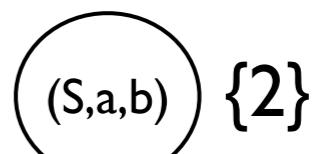
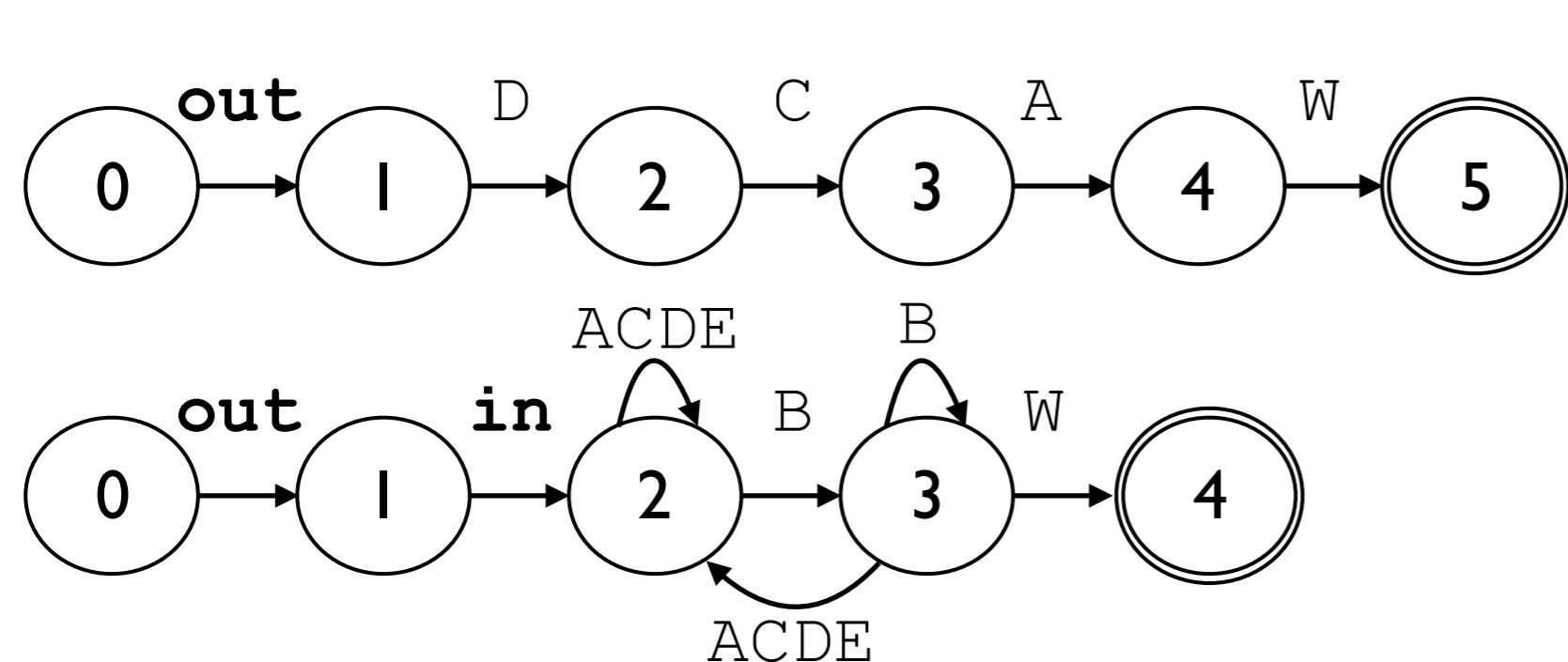
# Constructing the Product Graph (PG)



General Idea:  
PG represents locations  
reachable in the topology  
while following the policy

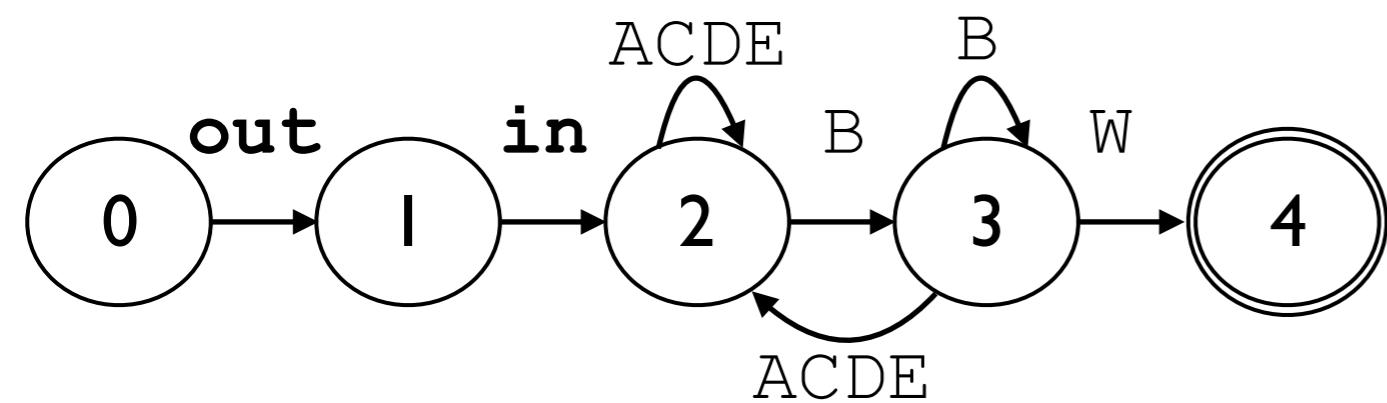
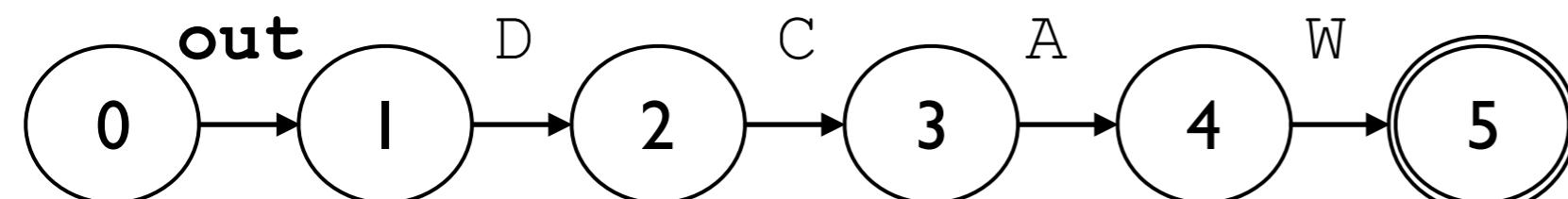
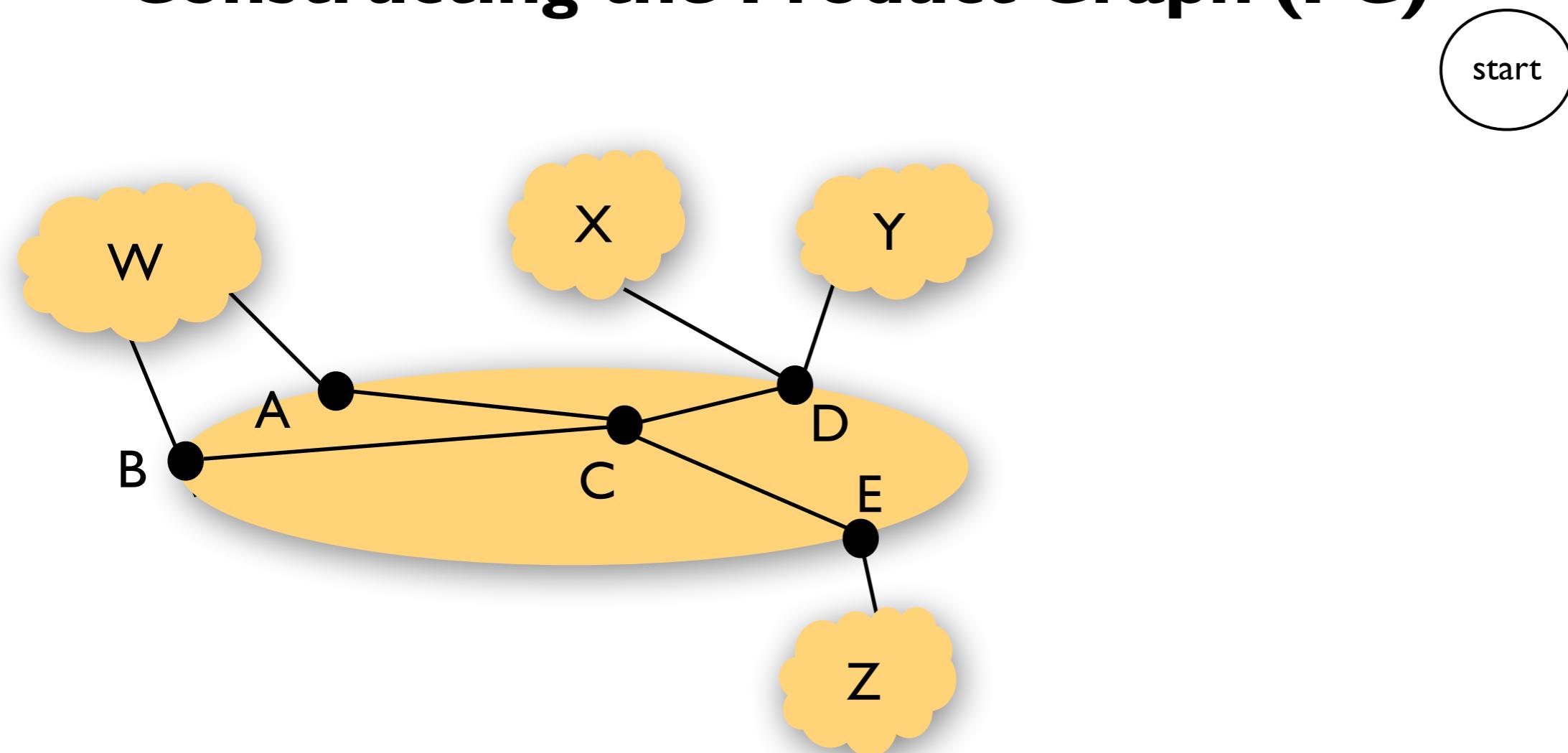
Each PG node contains:

- topology node (S)
- state of automaton 1 (a)
- state of automaton 2 (b)
- set of preferences achieved

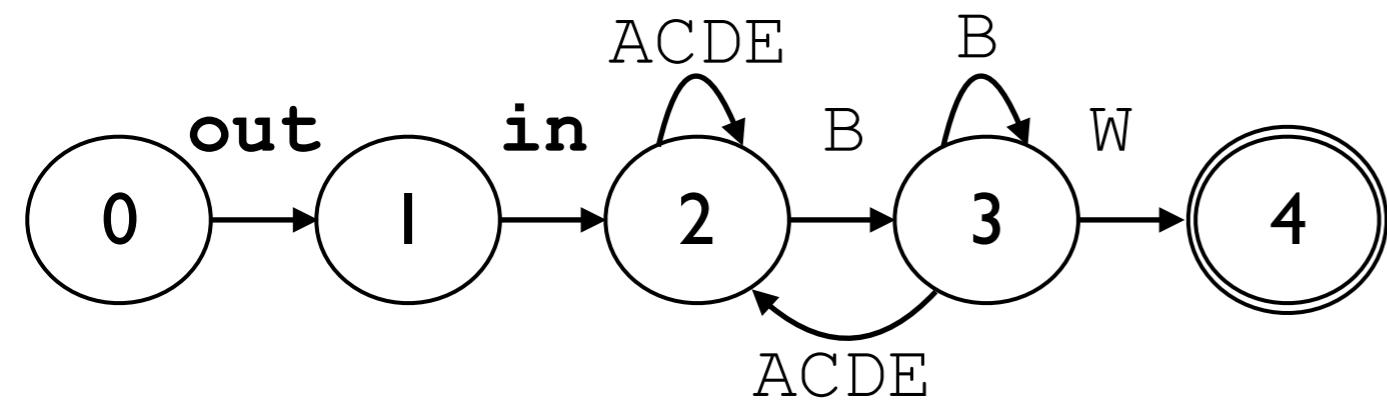
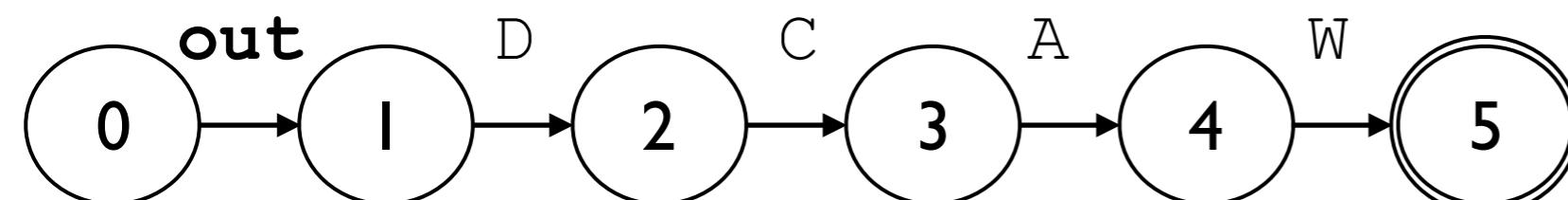
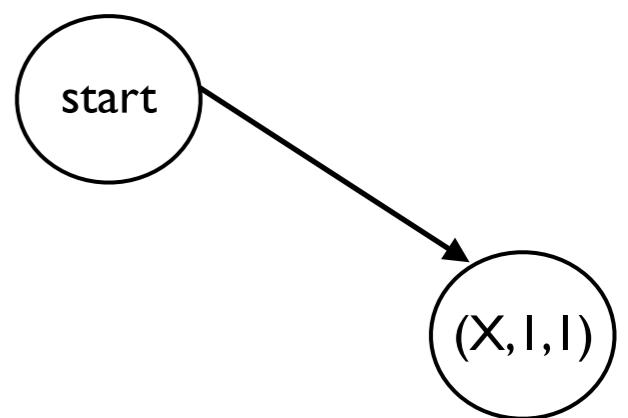
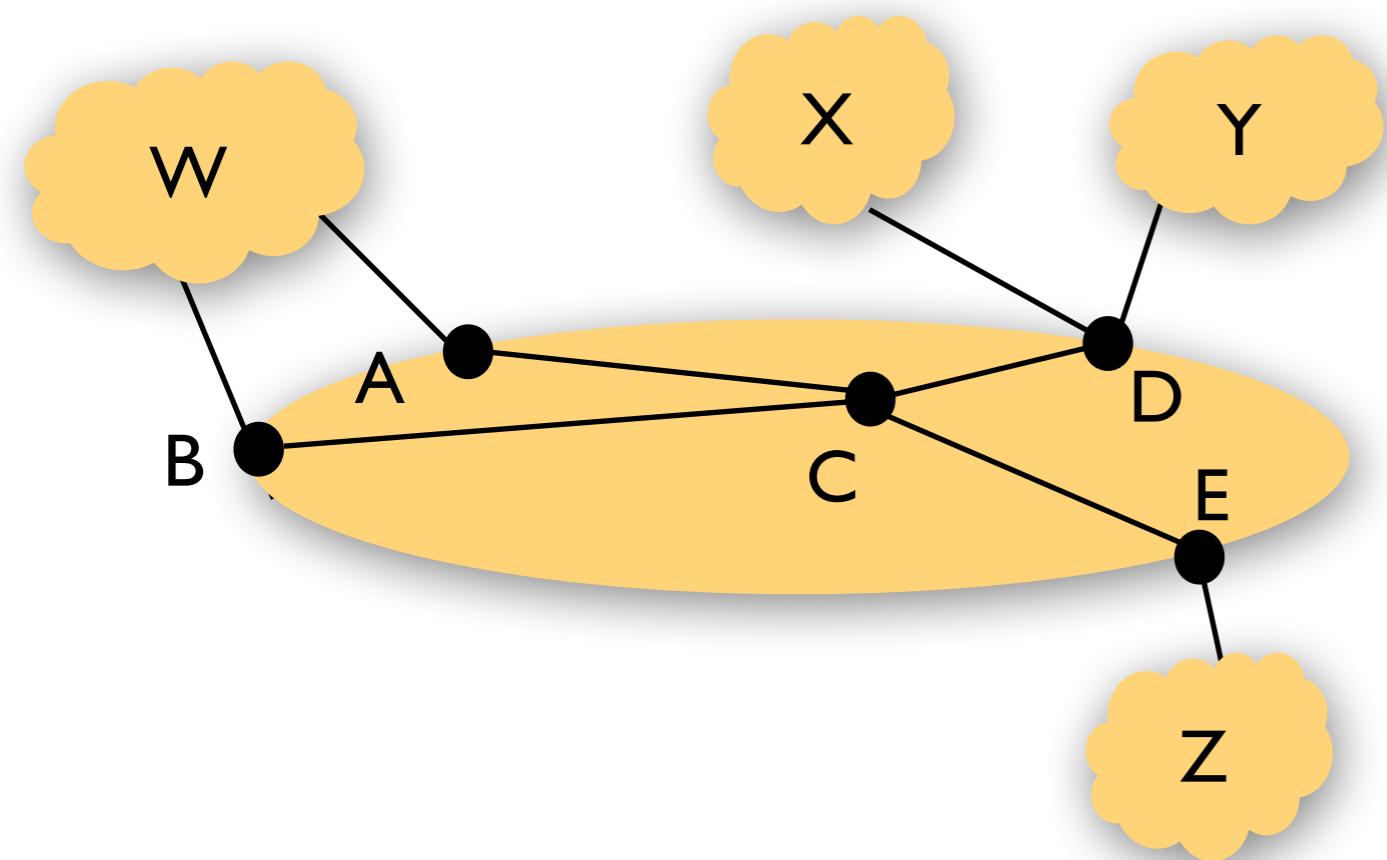


Two PG nodes are connected  
if topology nodes are connected  
and the automata make the  
specified transition

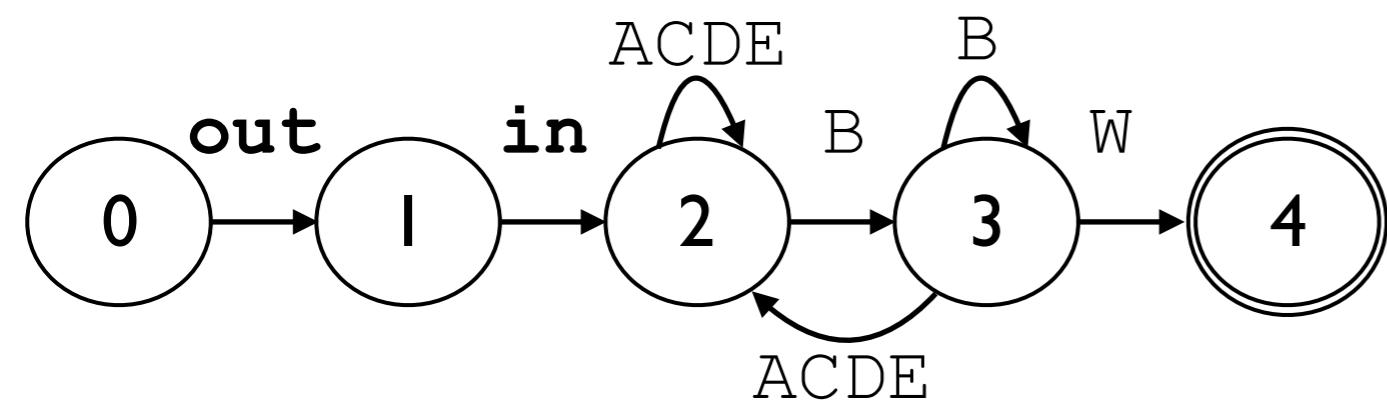
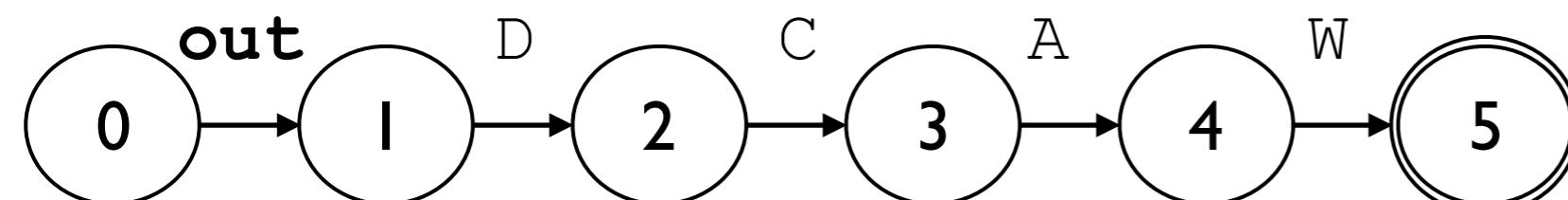
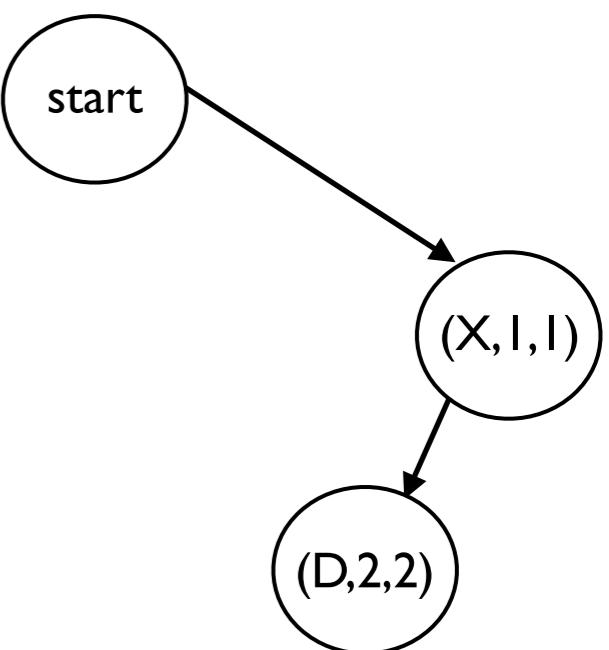
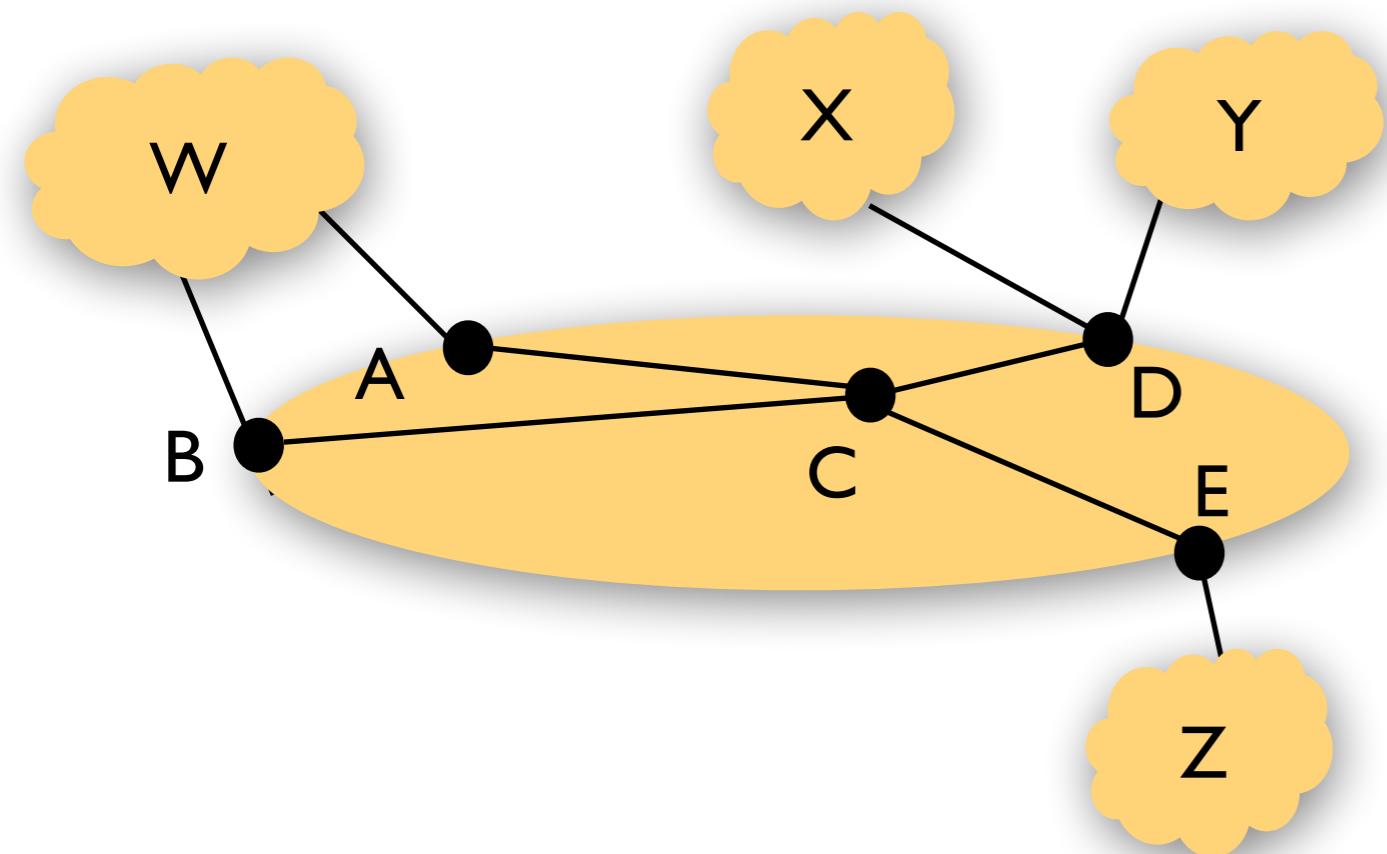
# Constructing the Product Graph (PG)



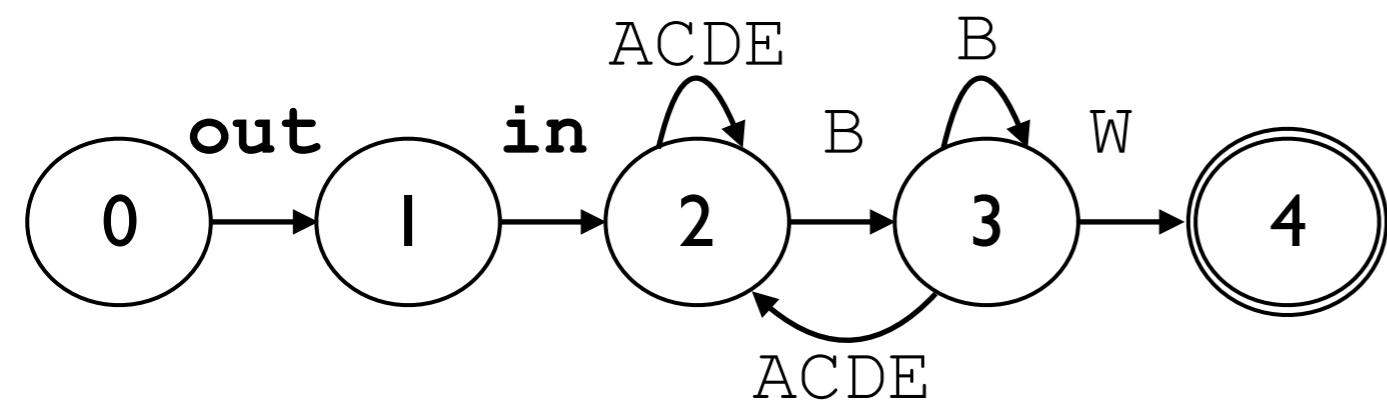
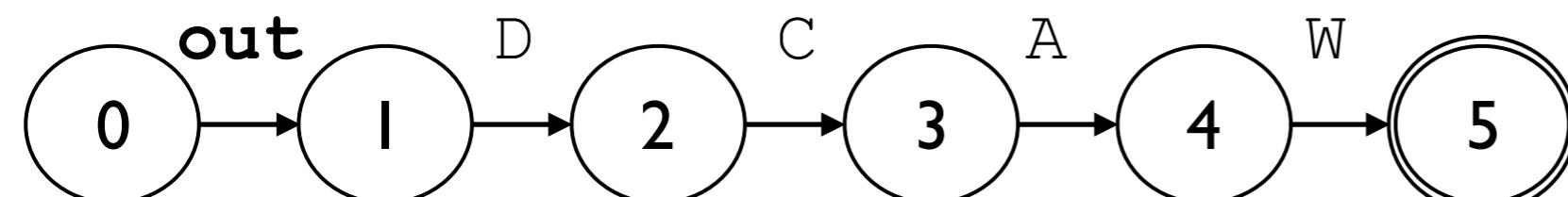
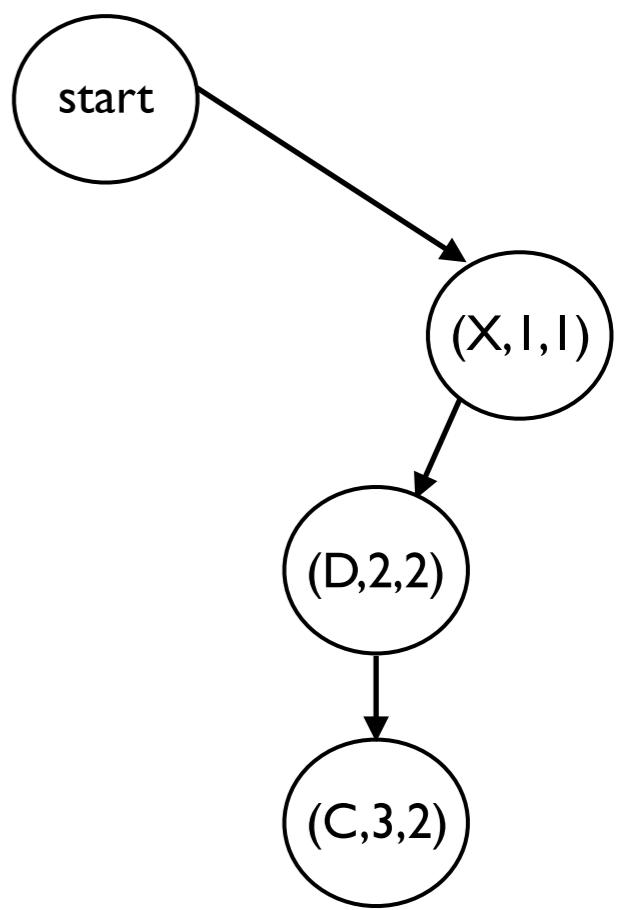
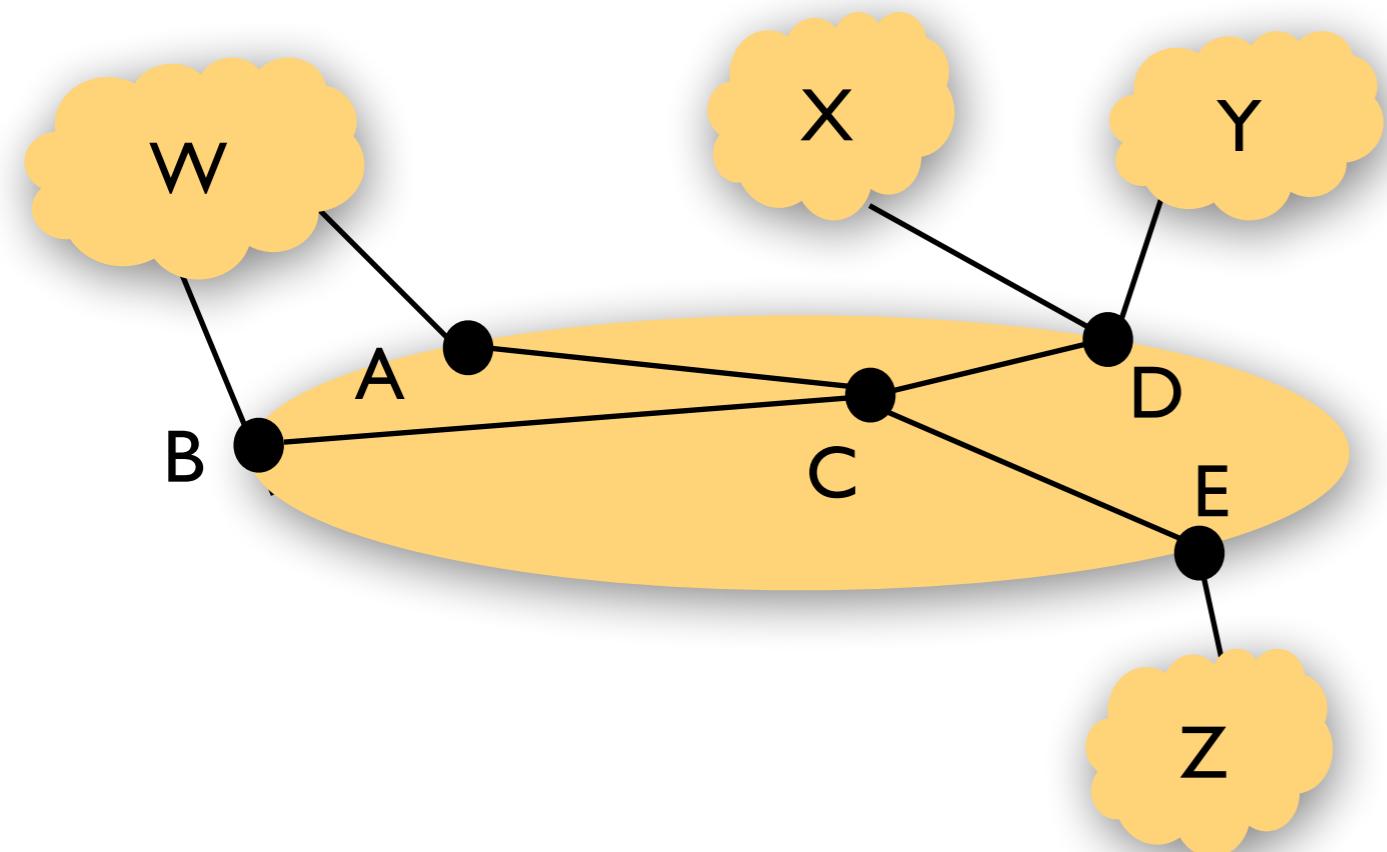
# Constructing the Product Graph (PG)



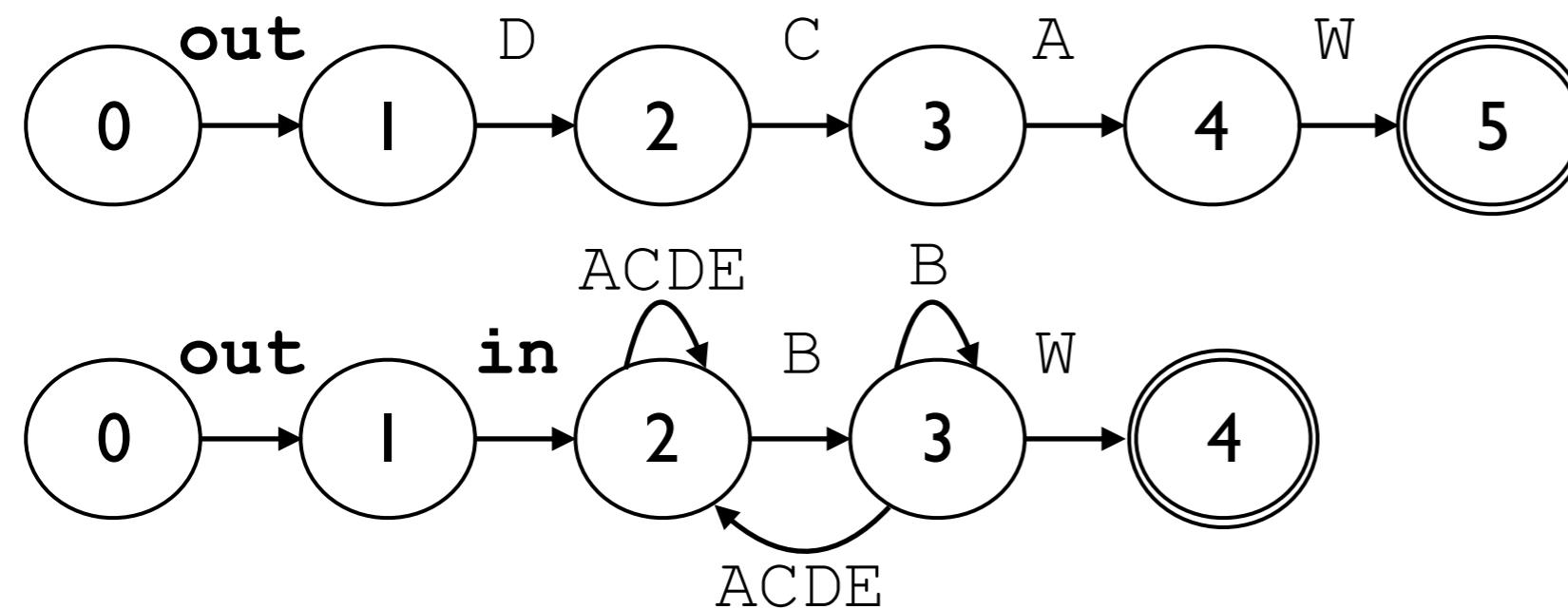
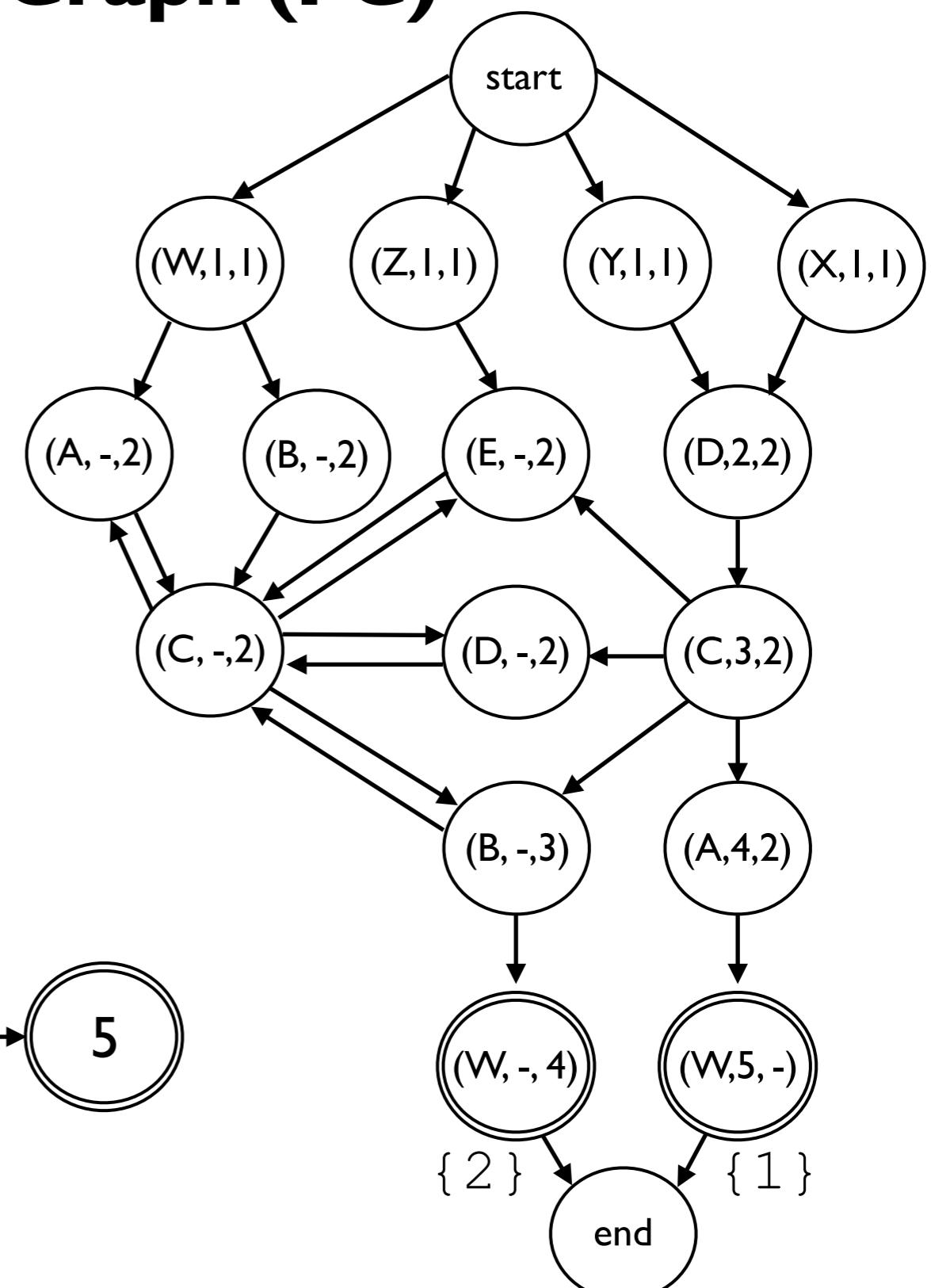
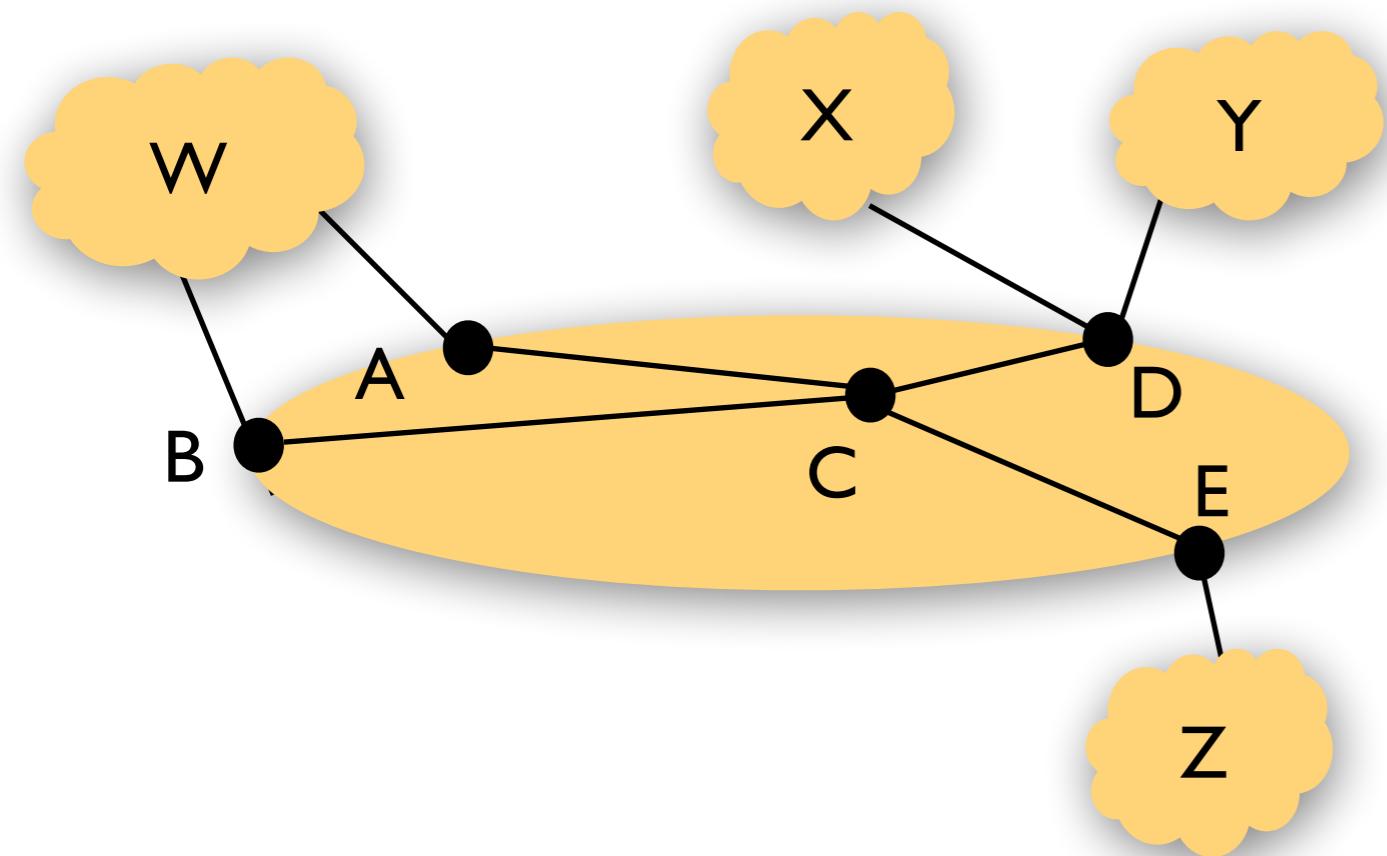
# Constructing the Product Graph (PG)



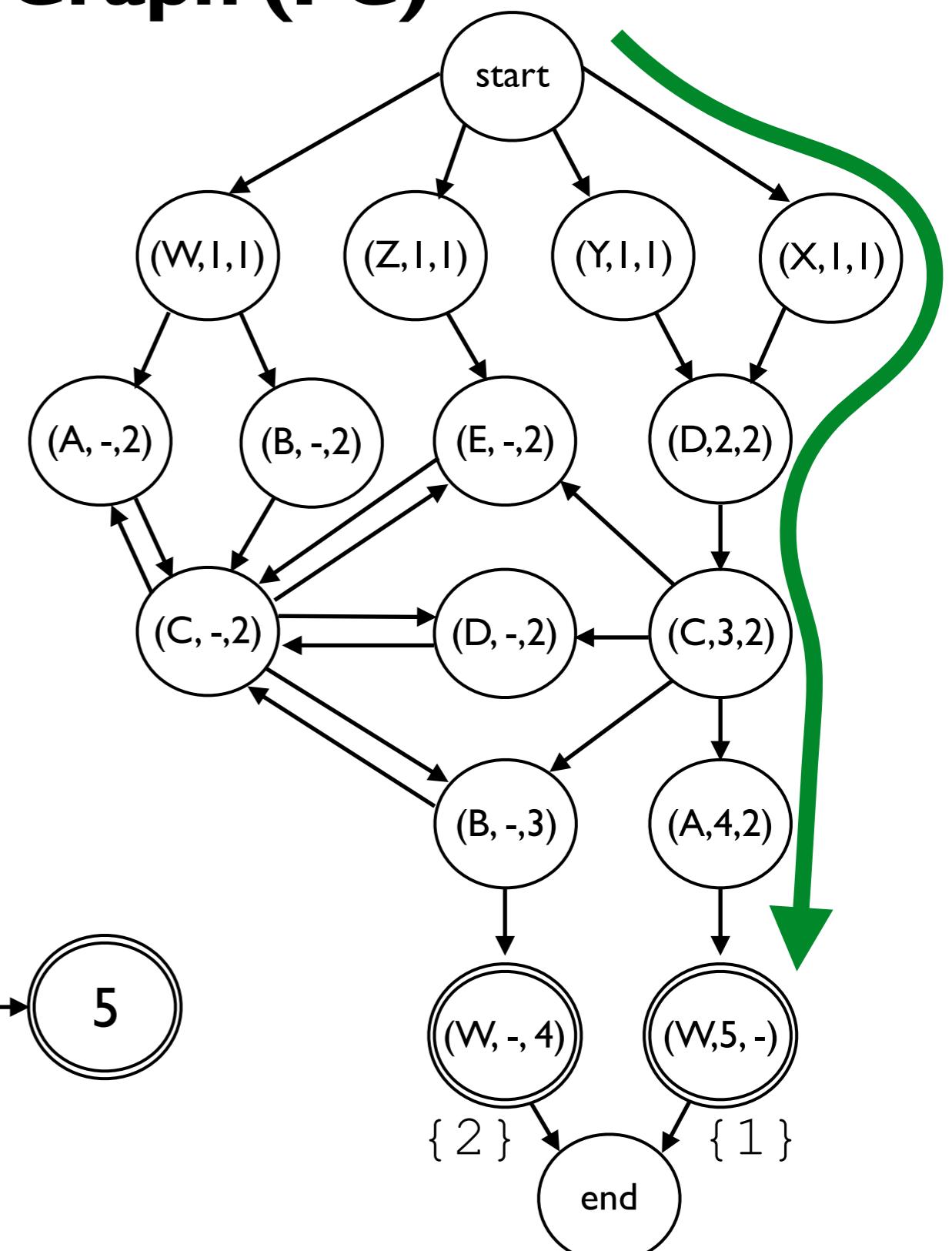
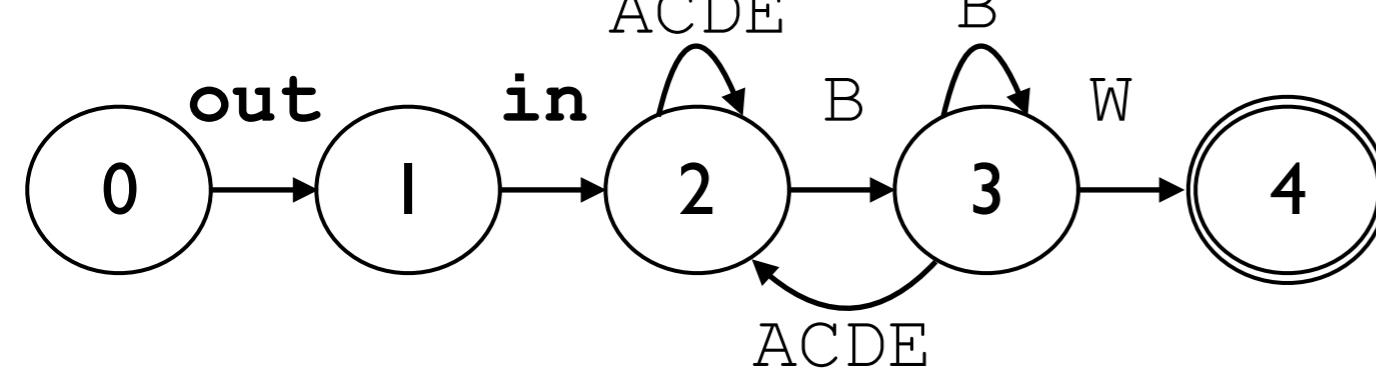
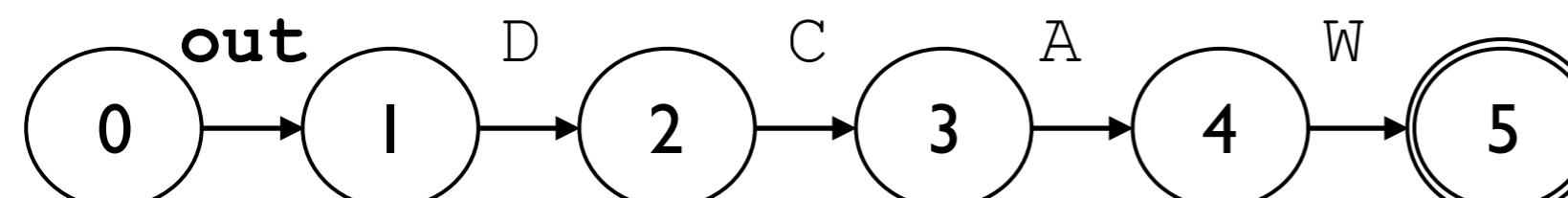
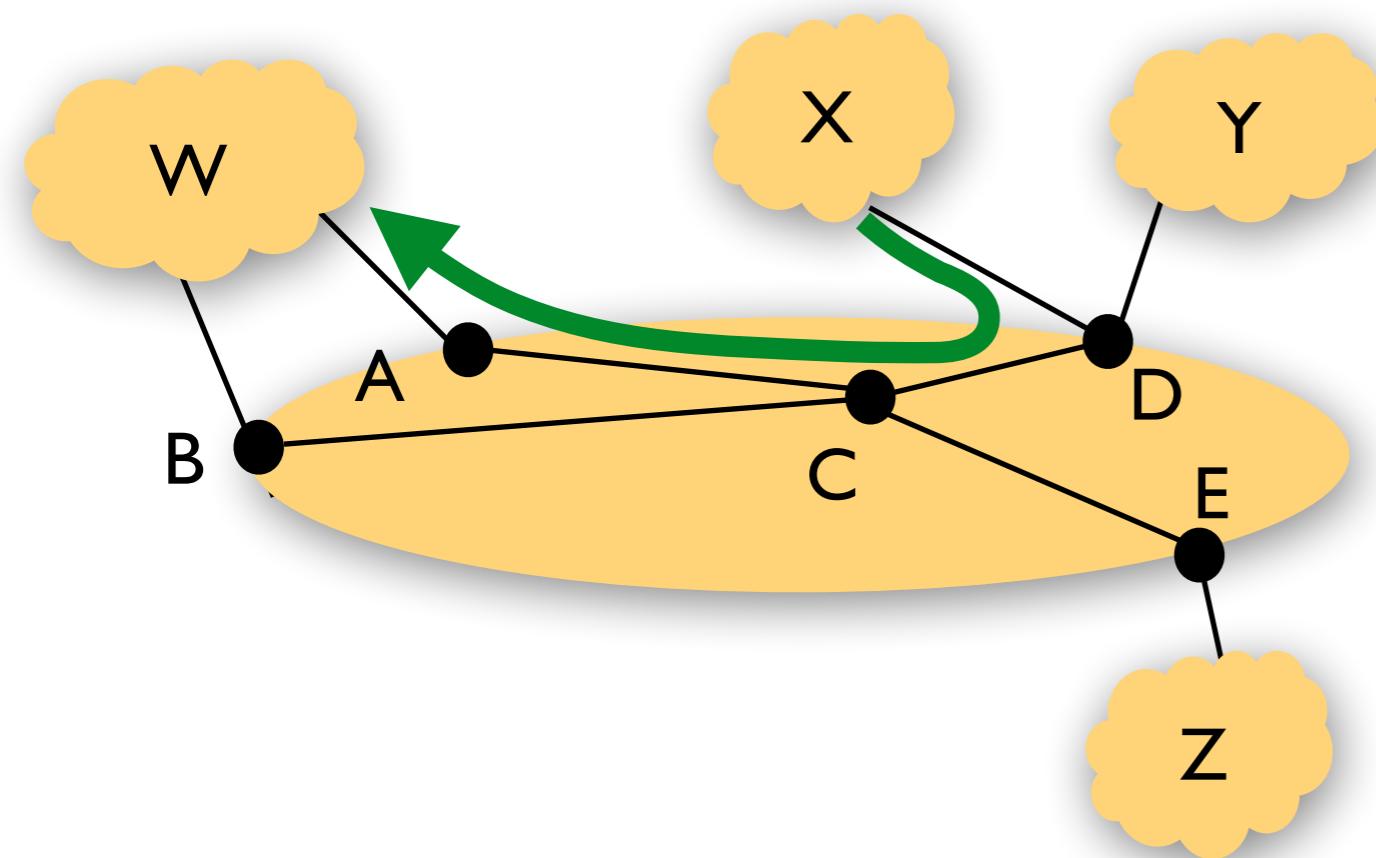
# Constructing the Product Graph (PG)



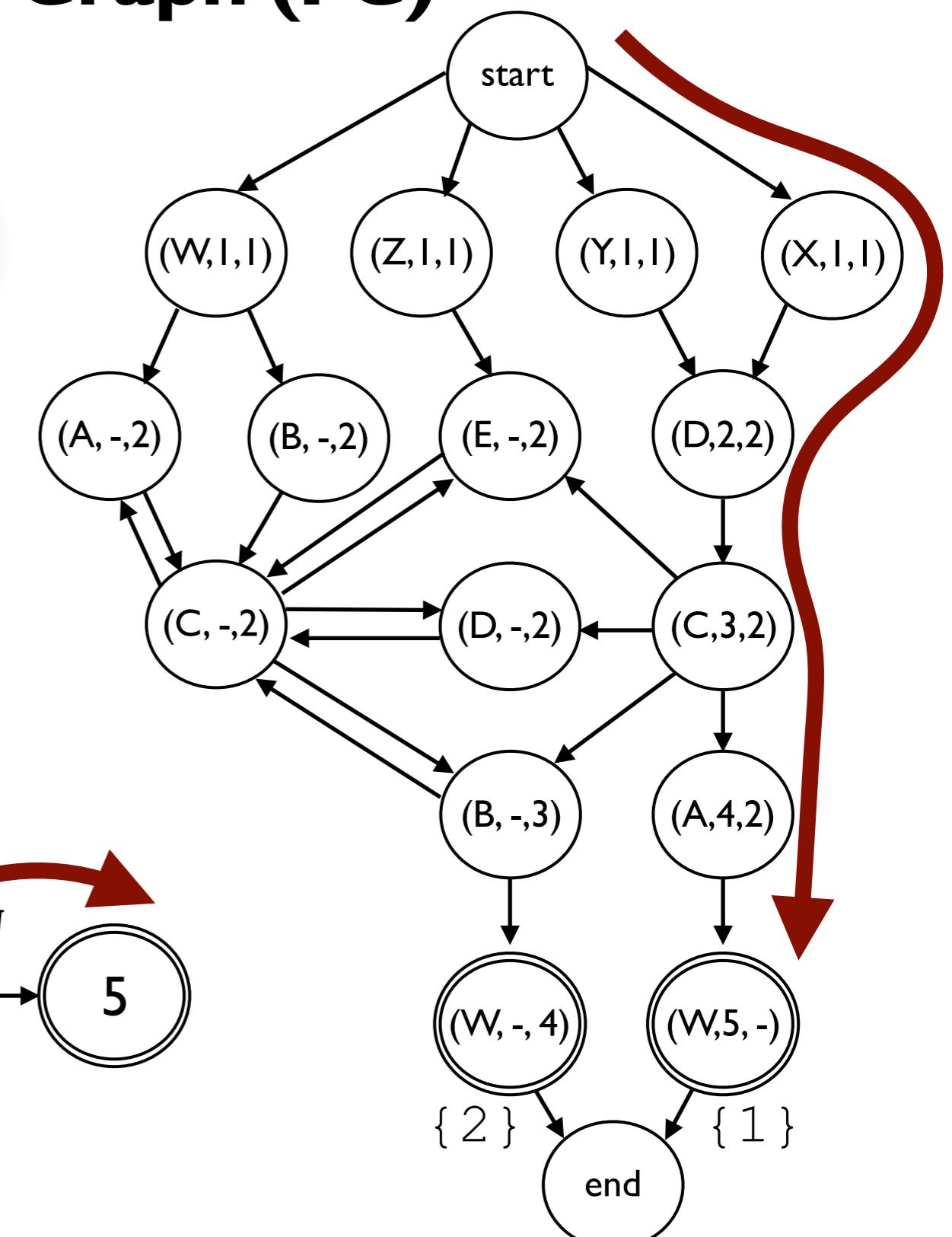
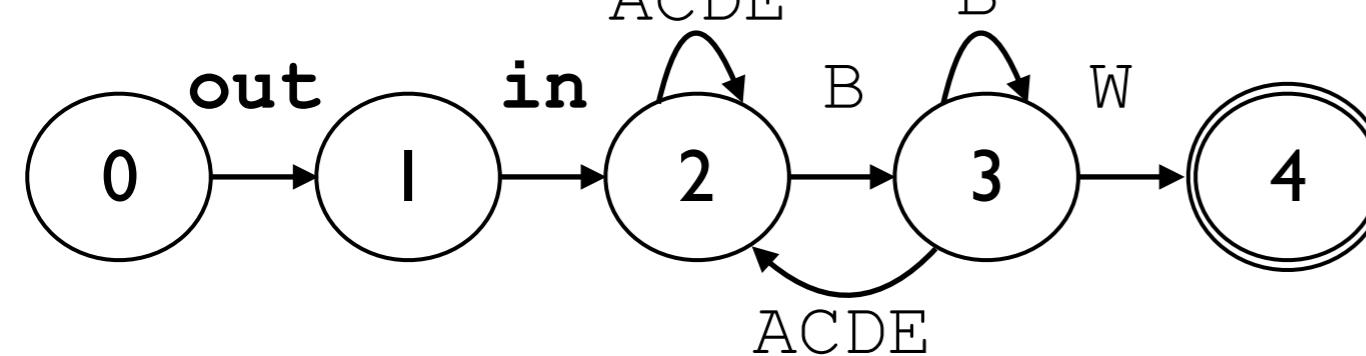
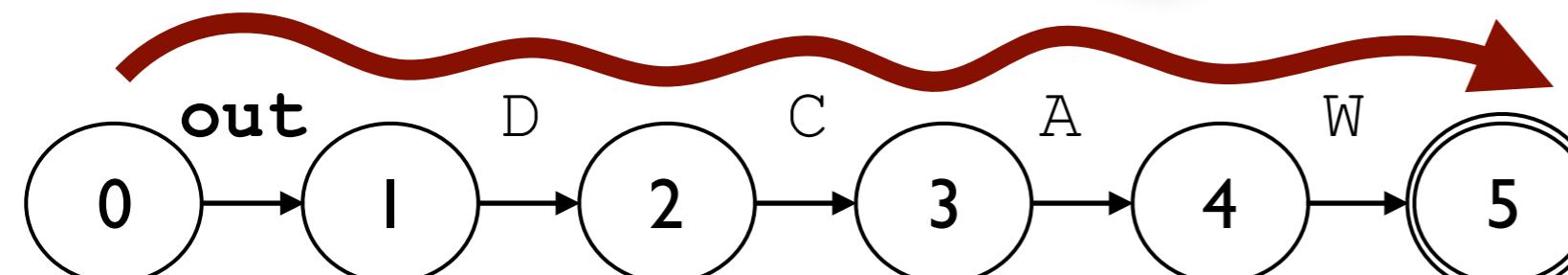
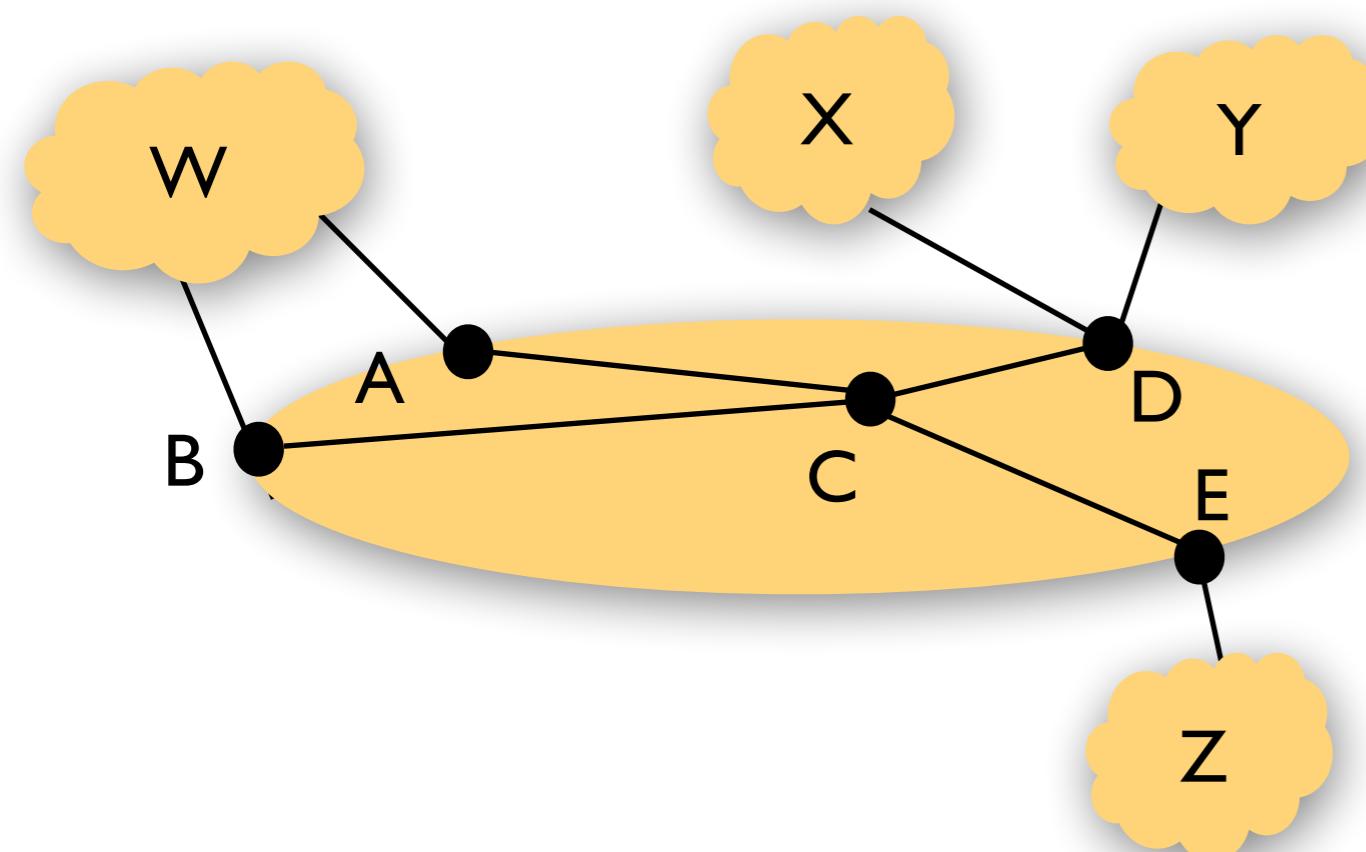
# Constructing the Product Graph (PG)



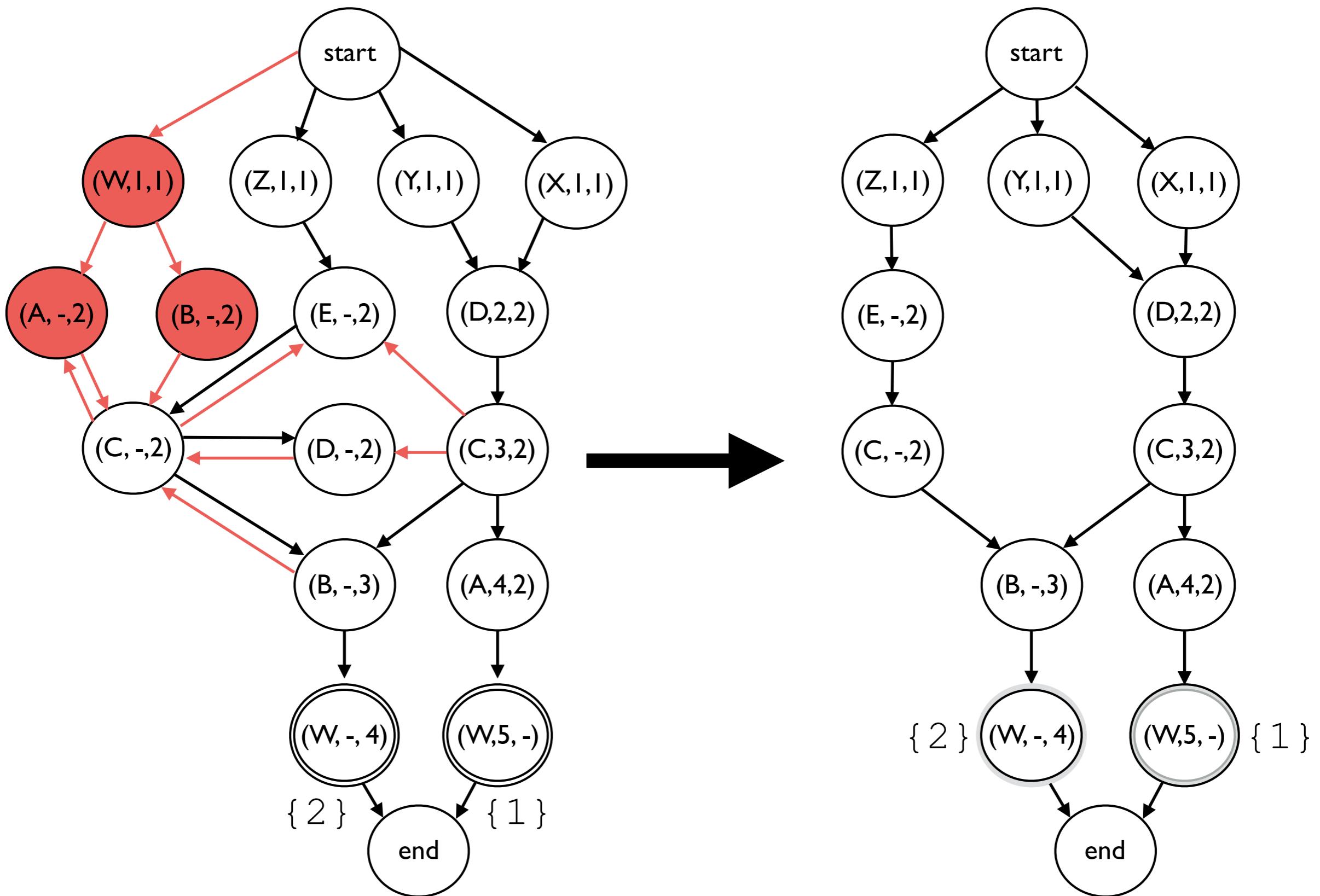
# Constructing the Product Graph (PG)



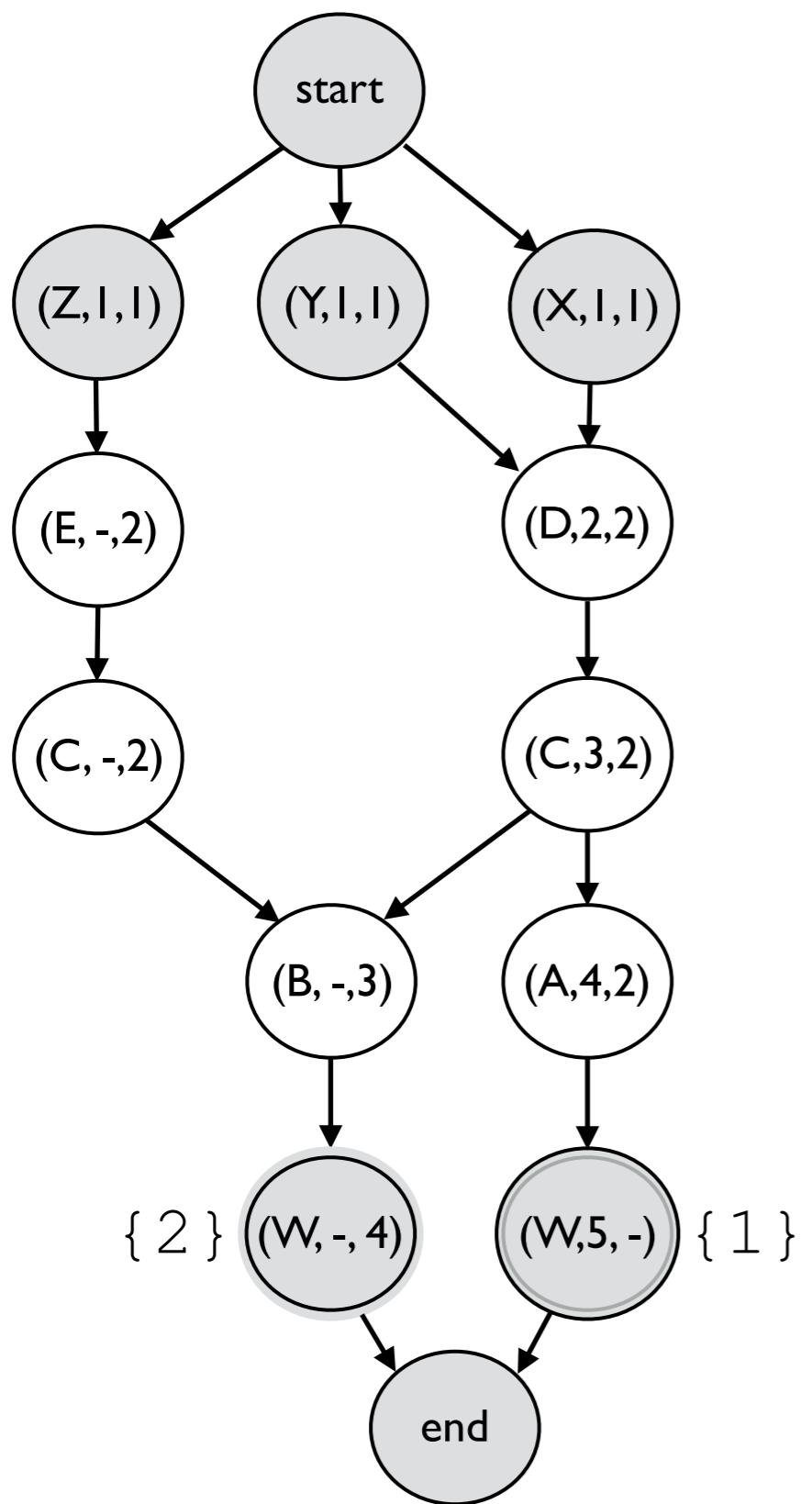
# Constructing the Product Graph (PG)



# Product Graph Minimization



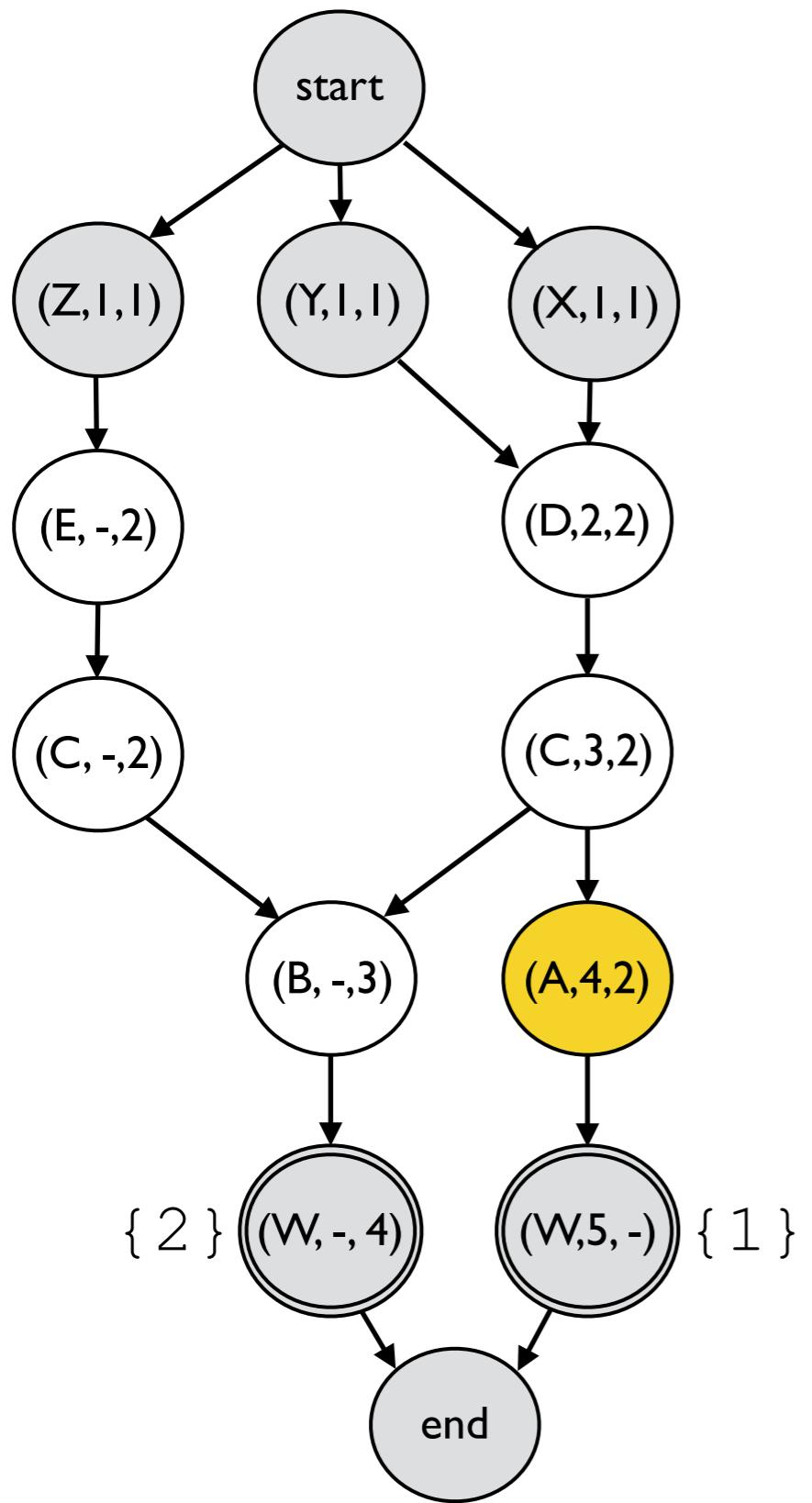
# Compilation to BGP:



Idea:

- Filter import messages according to incoming PG edges.
- For each internal location, decide which announcements to prefer, forward messages along PG edges
- Use a community value to tag the state of the automata
- For each external location, do nothing

# 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←(-,2),  
comm←noexport, MED←81
```

## Router C

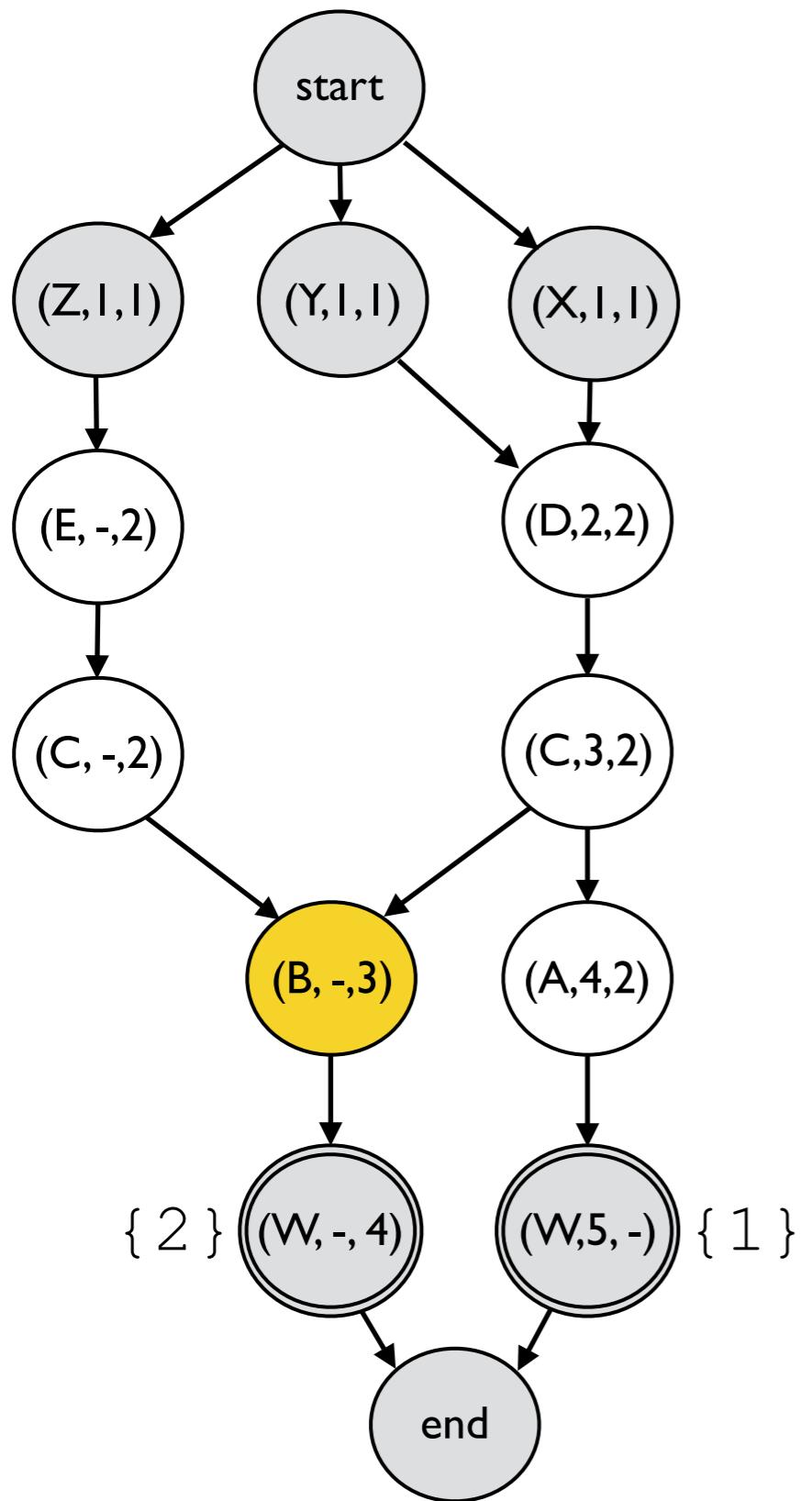
```
match[lp=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[lp=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)
```

## 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

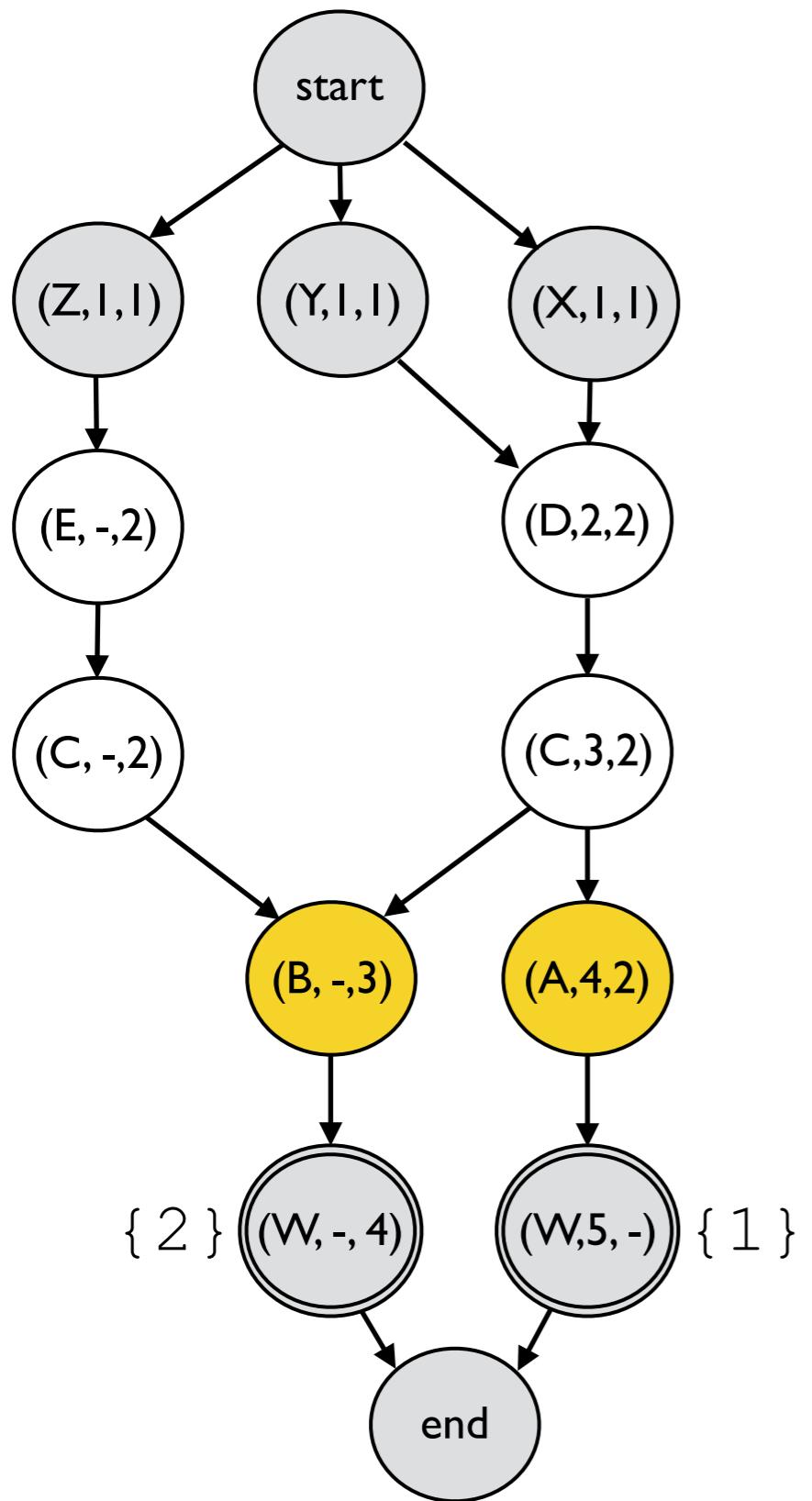
```
match[lp=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[lp=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)
```

## 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

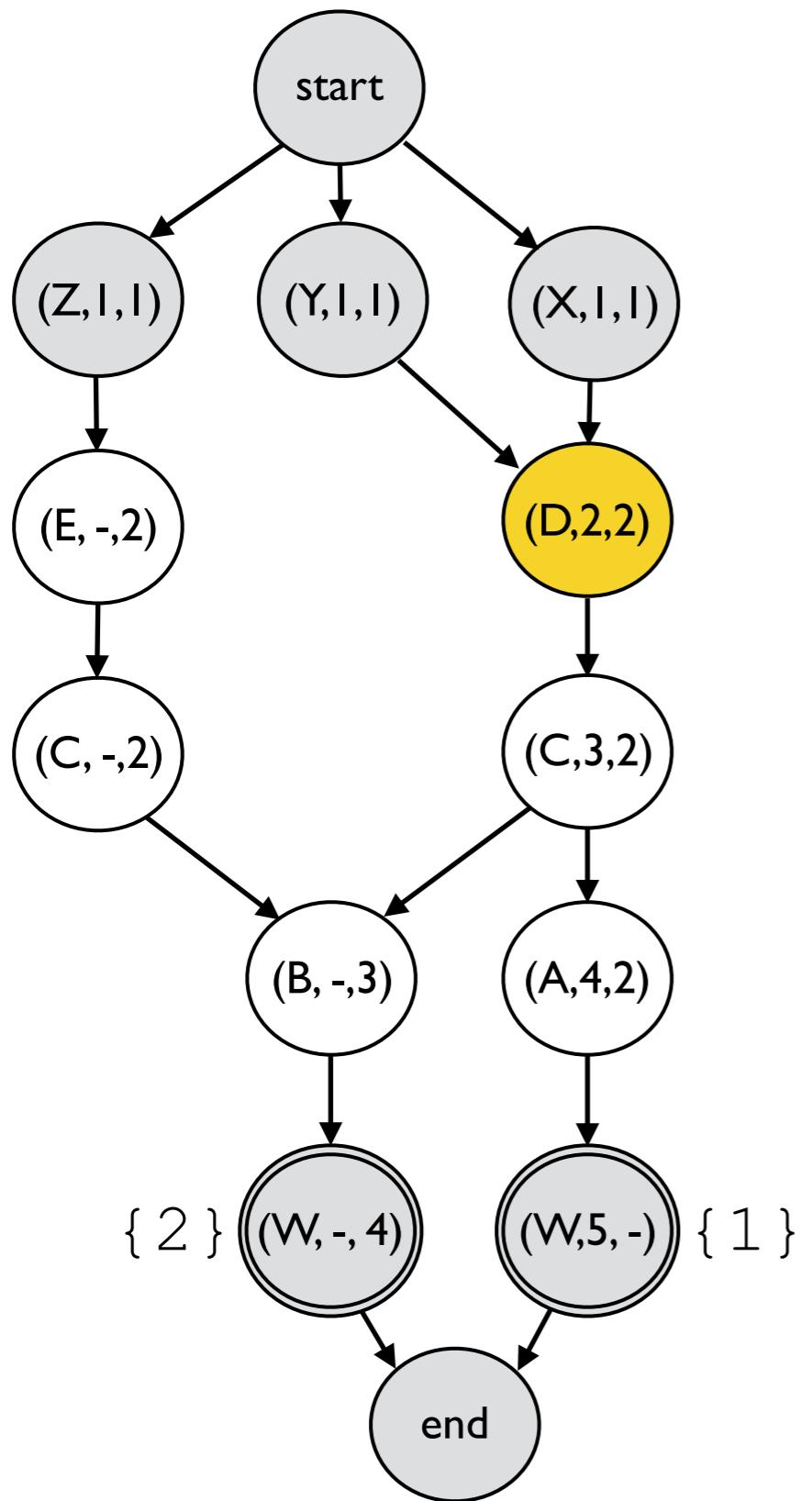
```
match[lp=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[lp=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)
```

## 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

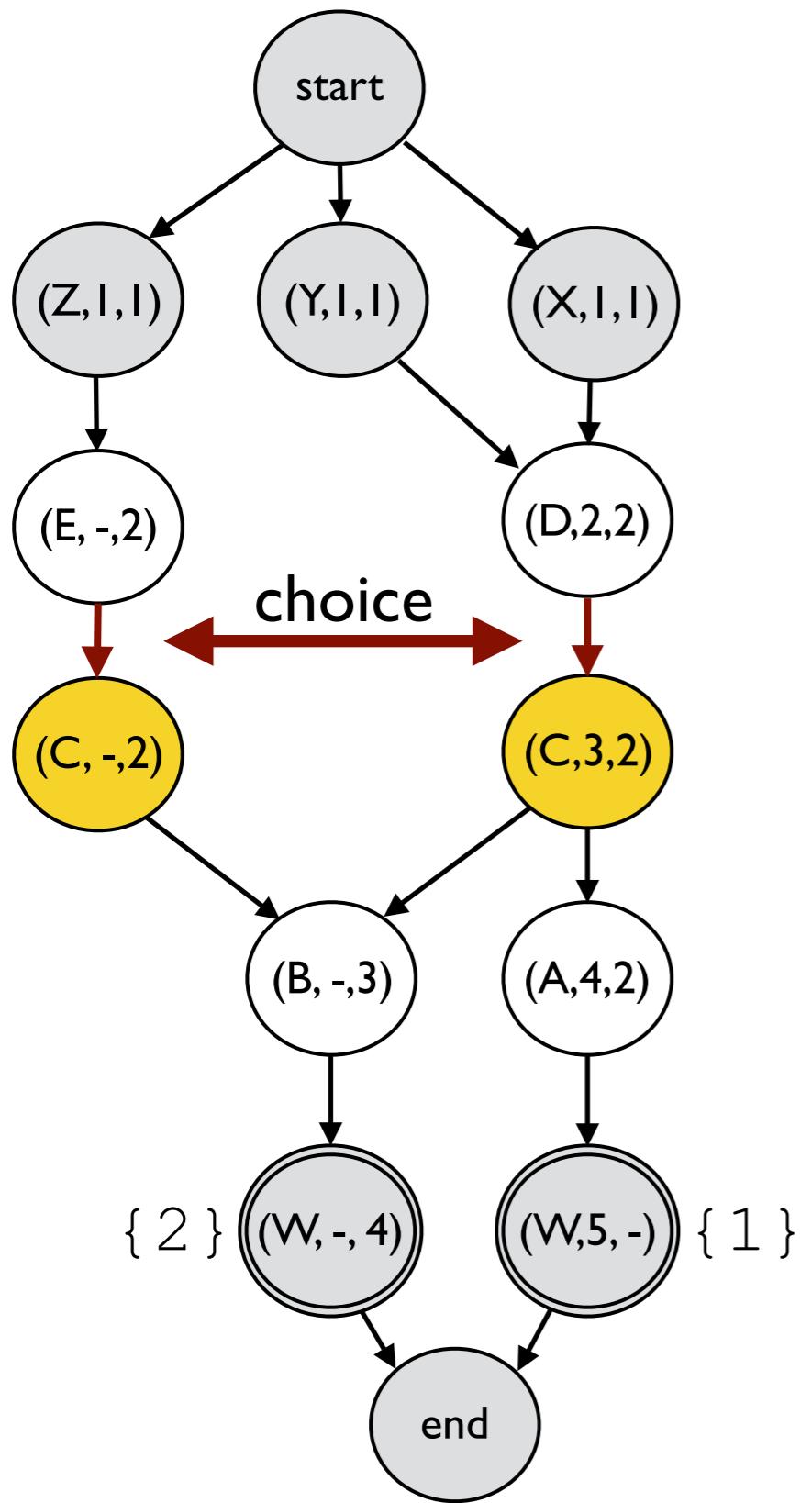
```
match[lp=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[lp=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)
```

## 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

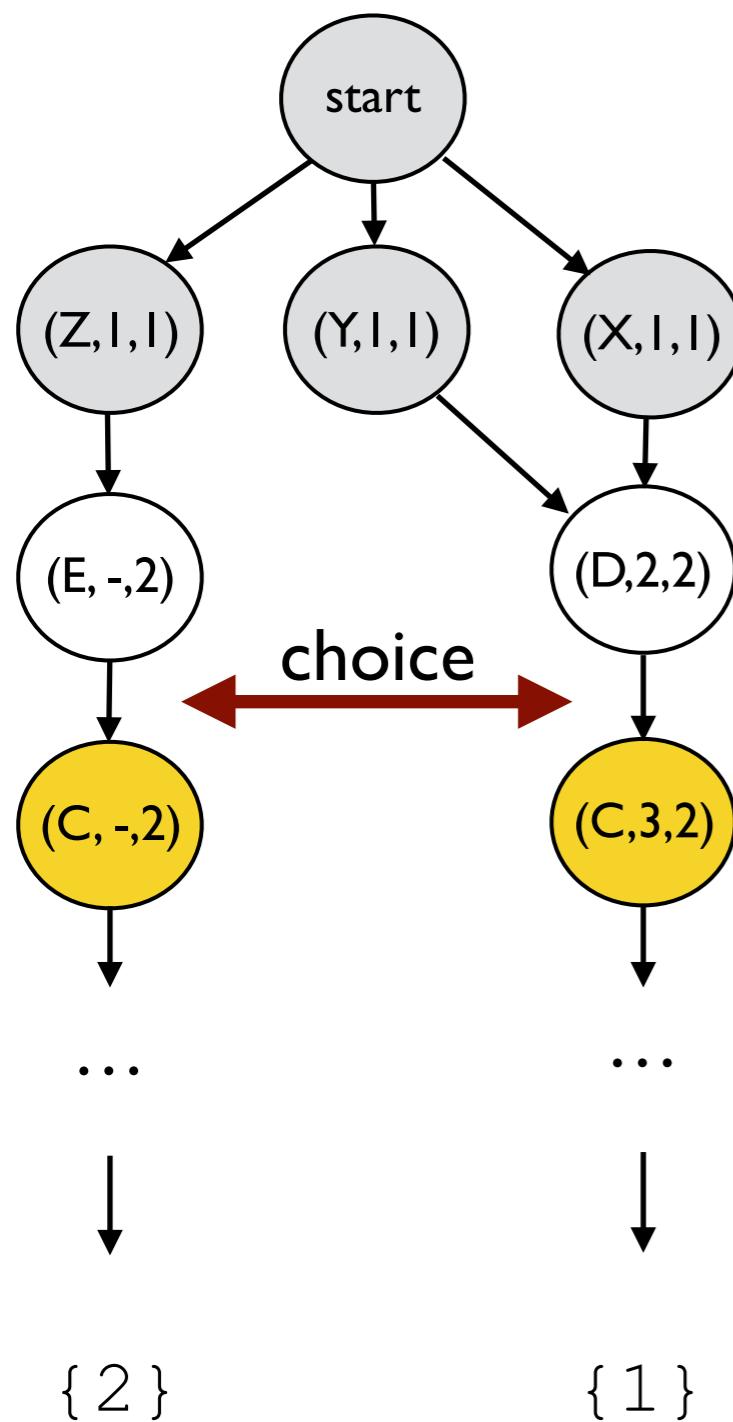
```
match[lp=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[lp=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)
```

## 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←8I
```

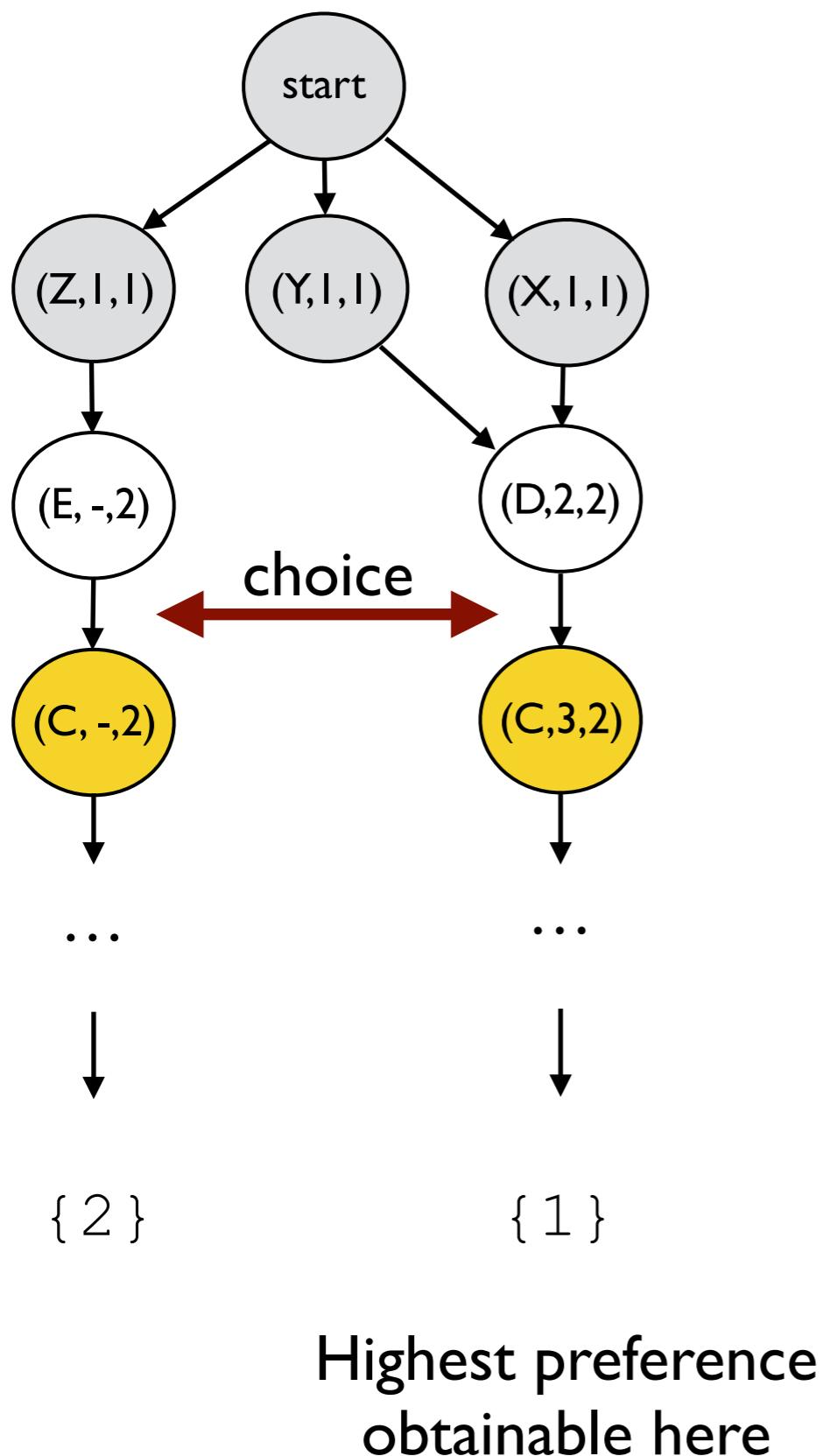
# Router C

```
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
```

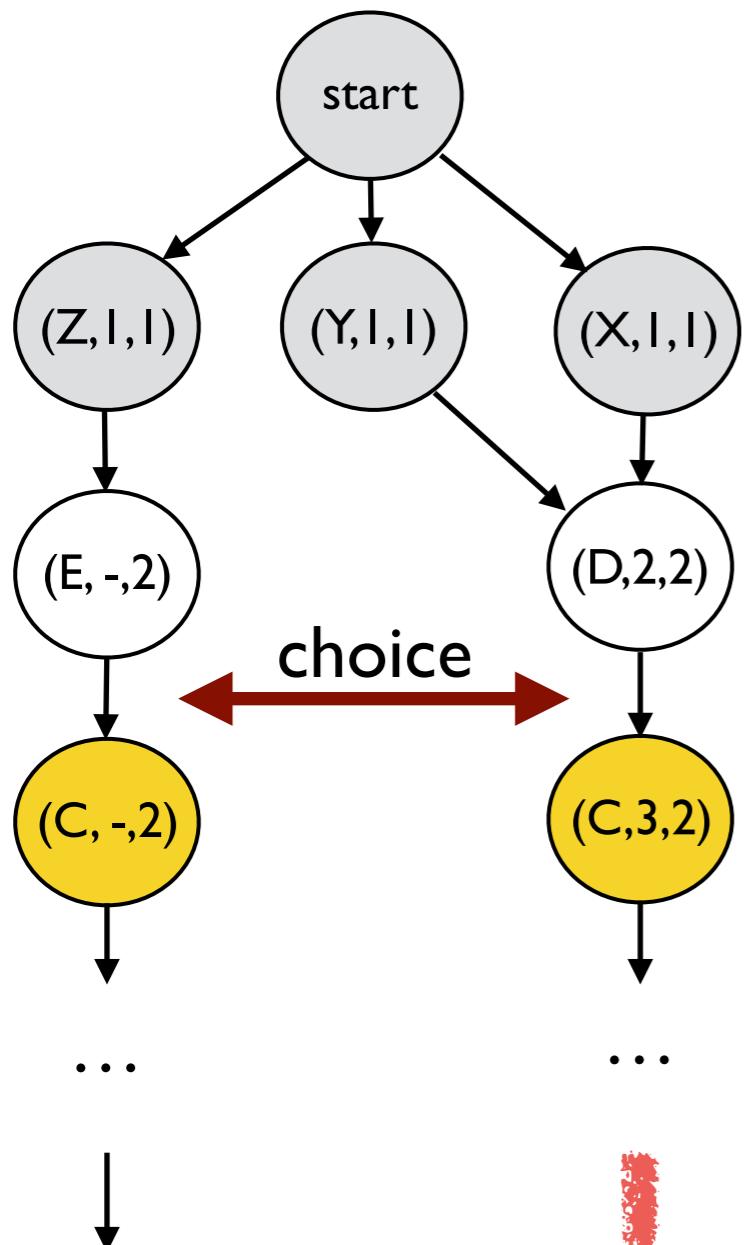
## Router C

```
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:



{ 2 }

{ 1 }

But there  
could be a  
failure!

## 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

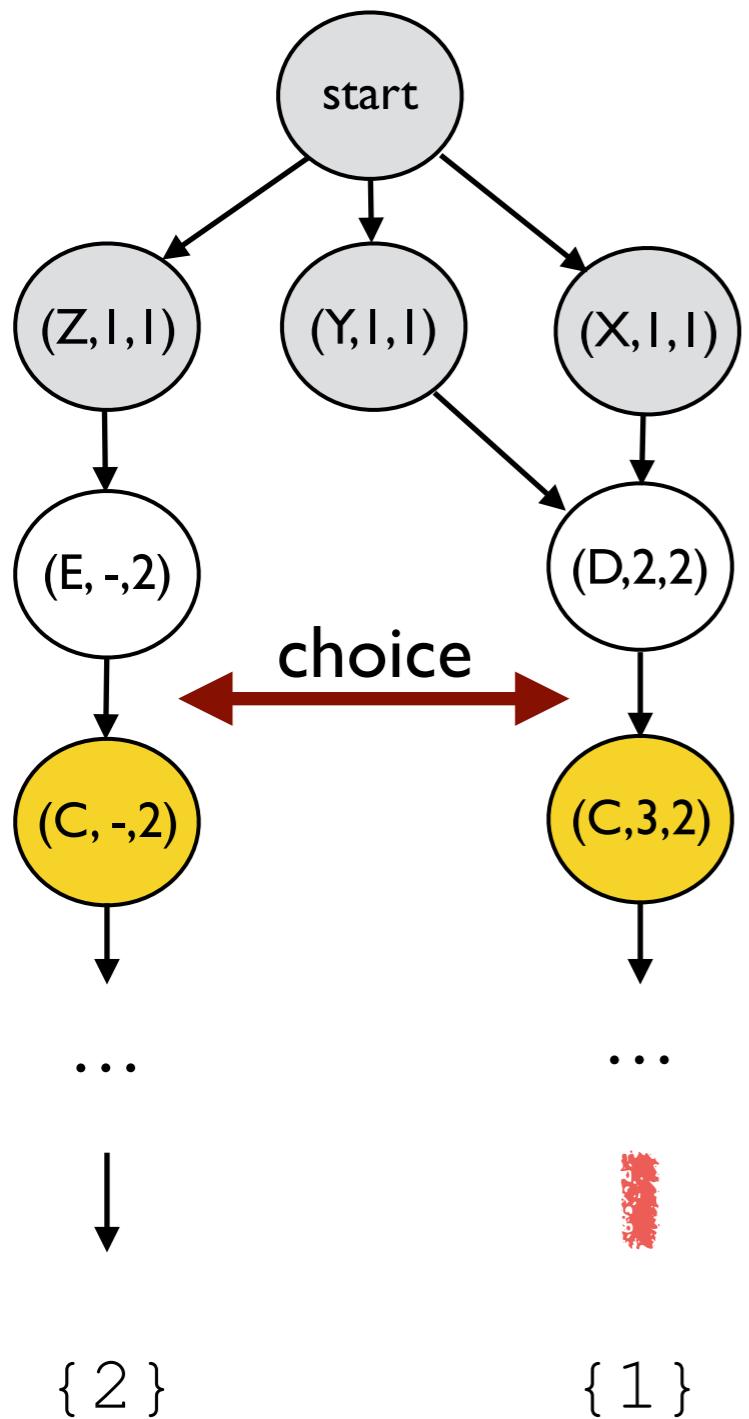
```
match[lp=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[lp=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)
```

## Router D

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

...

# Compilation to BGP:



*Sometimes there is  
no best choice*

## 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

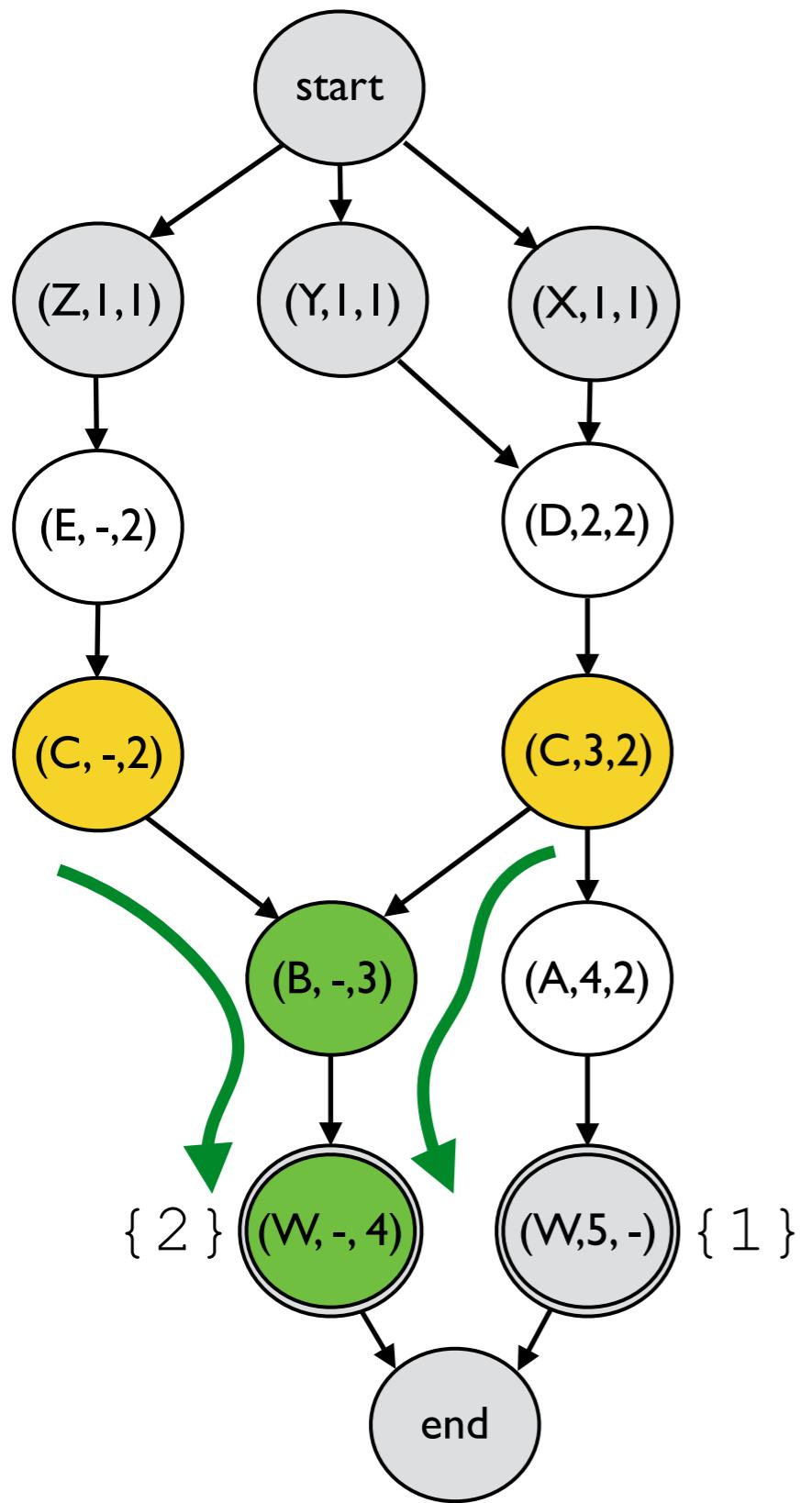
```
match[lp=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[lp=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)
```

## 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**

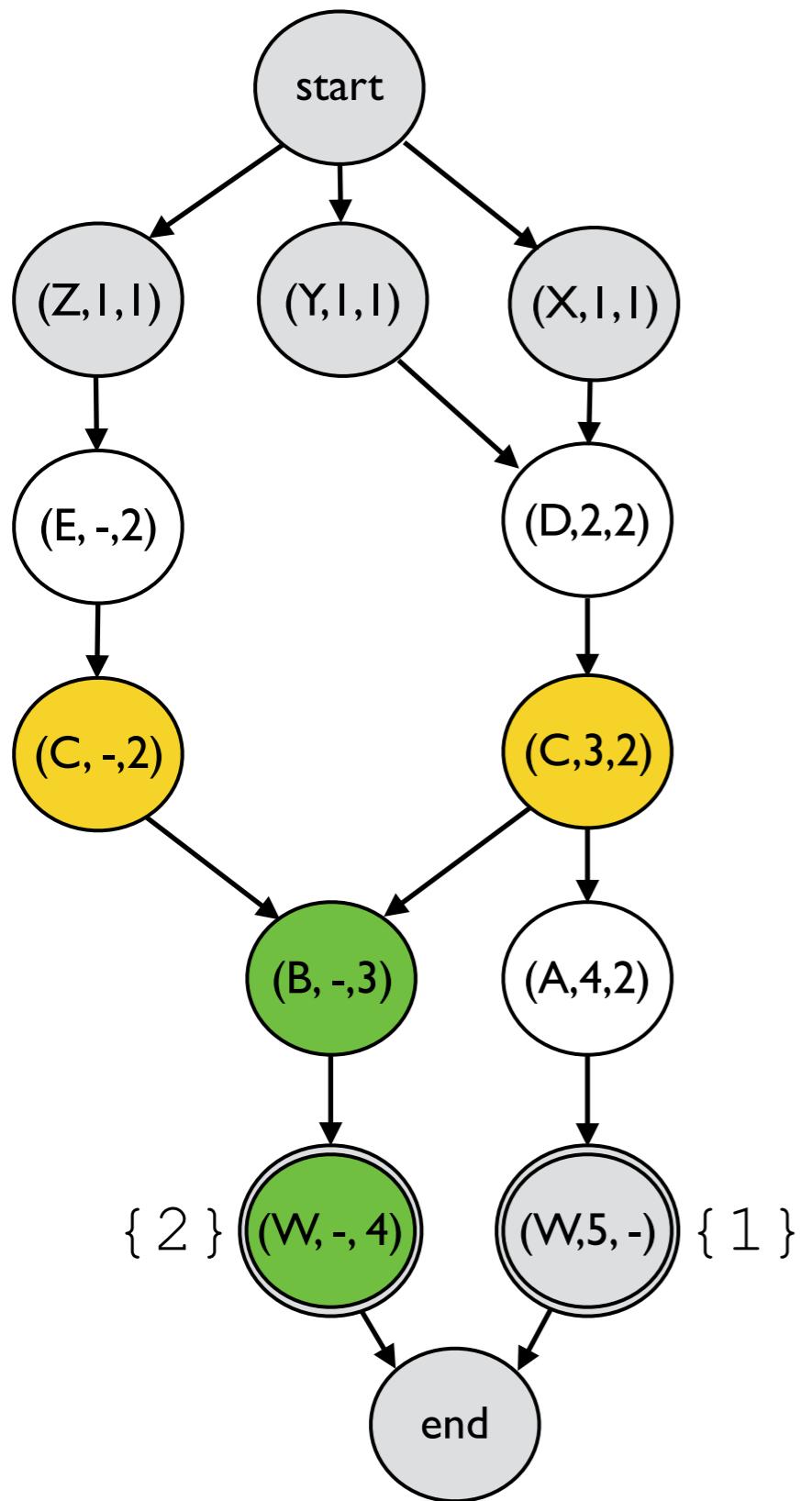
```
match[lp=99] peer=E, comm=(-,2)  
export peer←B, comm←(-,2)  
match[lp=100] peer=D, comm=(2,2)  
export peer←A,B, comm←(3,2)
```

**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[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)
    
```

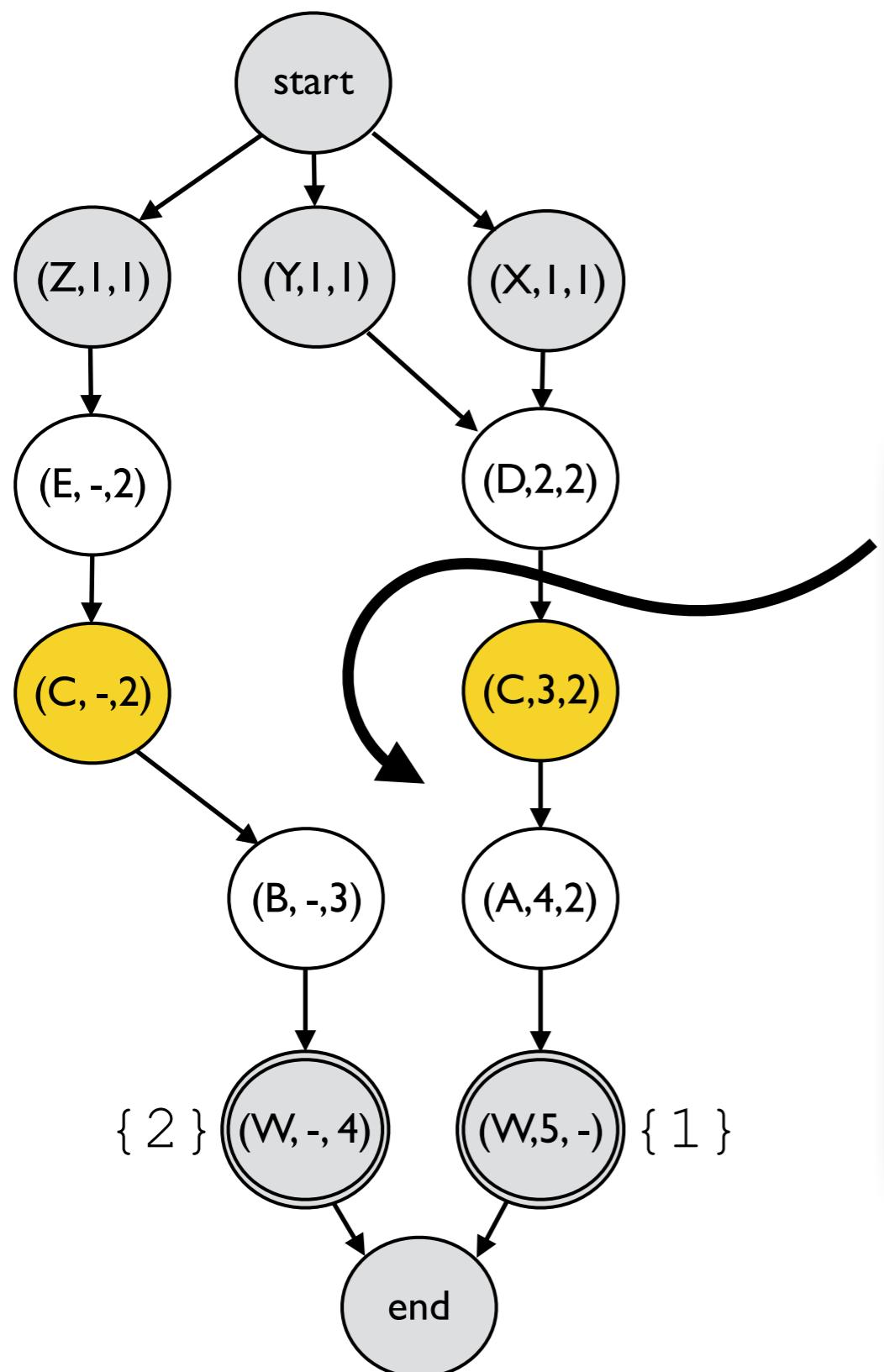
Prefer D's  
announce

**Router D**

```

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

# Compilation to BGP:



If we remove this edge,  
the policy becomes unimplementable.

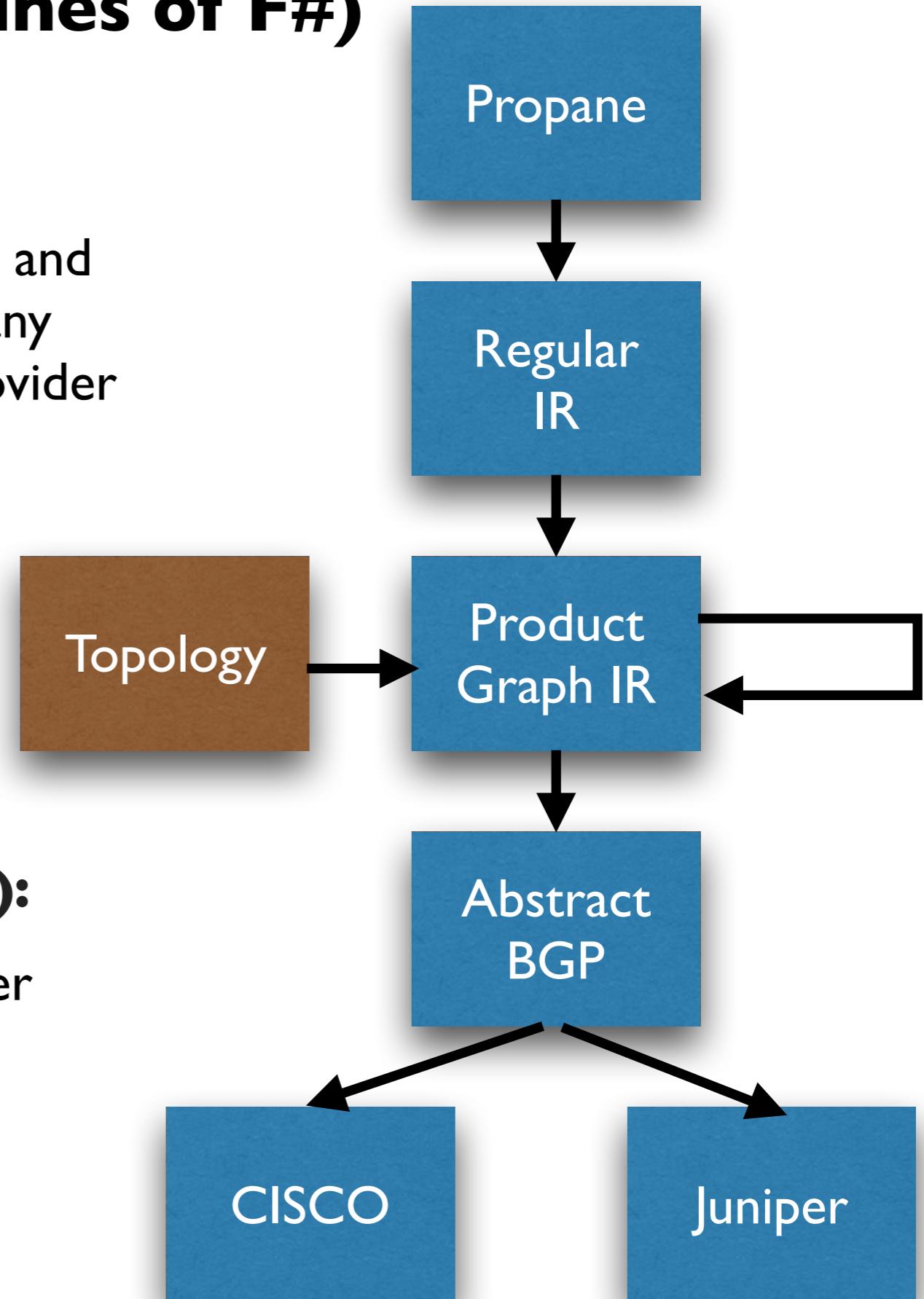
If we choose D's announcement and forward to A, the A-W link may fail

If we choose E's announcement and forward to B, the B-W link may fail (or the A-W link may not fail and we will have a sub-optimal route)

# Implementation (5,500 lines of F#)

## Benchmarks:

- data center policies (~1600 routers) and backbone policies (~200 routers, many peers/router) from a large cloud provider
- policy from English docs
- Ignoring prefix, customer group and ownership definitions:
  - 31 lines for data center
  - 43 lines for backbone



## Scaling (8 core Windows machine):

- 10s/pfx (mean) for largest data center
- 45s/pfx (mean) for largest backbone
- 3 minutes total for the backbone
- 9 minutes total for the data center

# Summary

Propane users express high-level objectives.

Objectives involve both intra- and inter-domain constraints.

Failure analysis guarantees strong safety properties.

Compiler bridges the gap between objectives and device-by-device control plane configurations.

