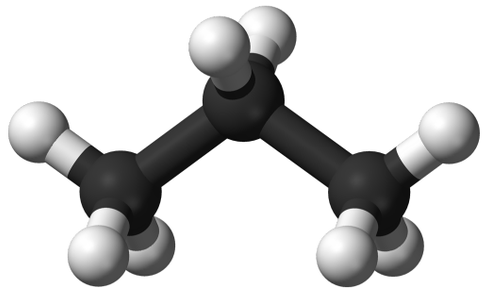
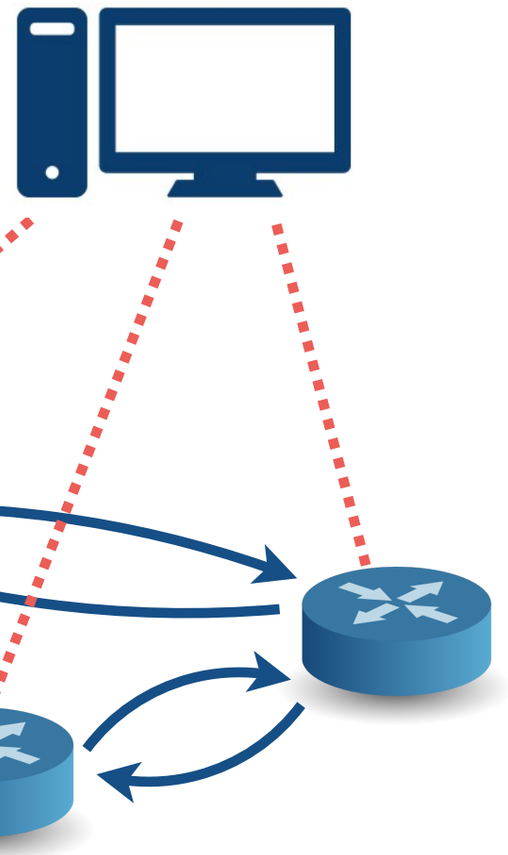
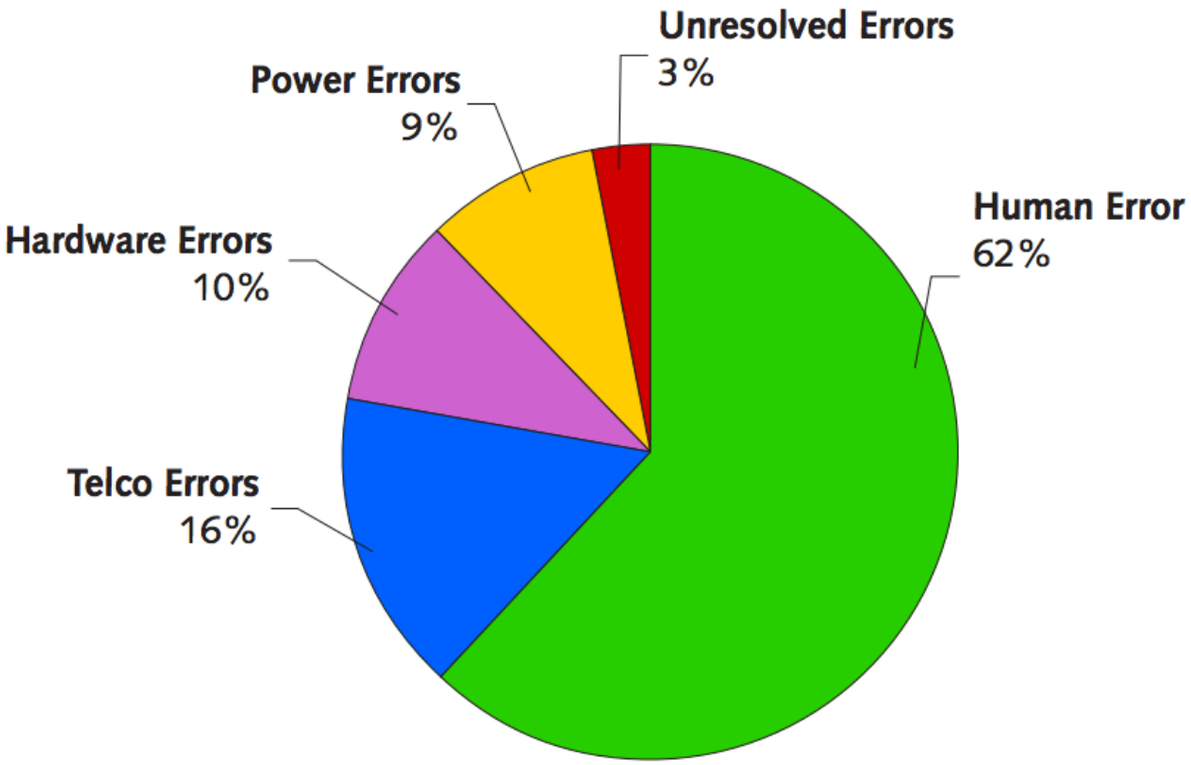
Propane: Programming Distributed Control Planes

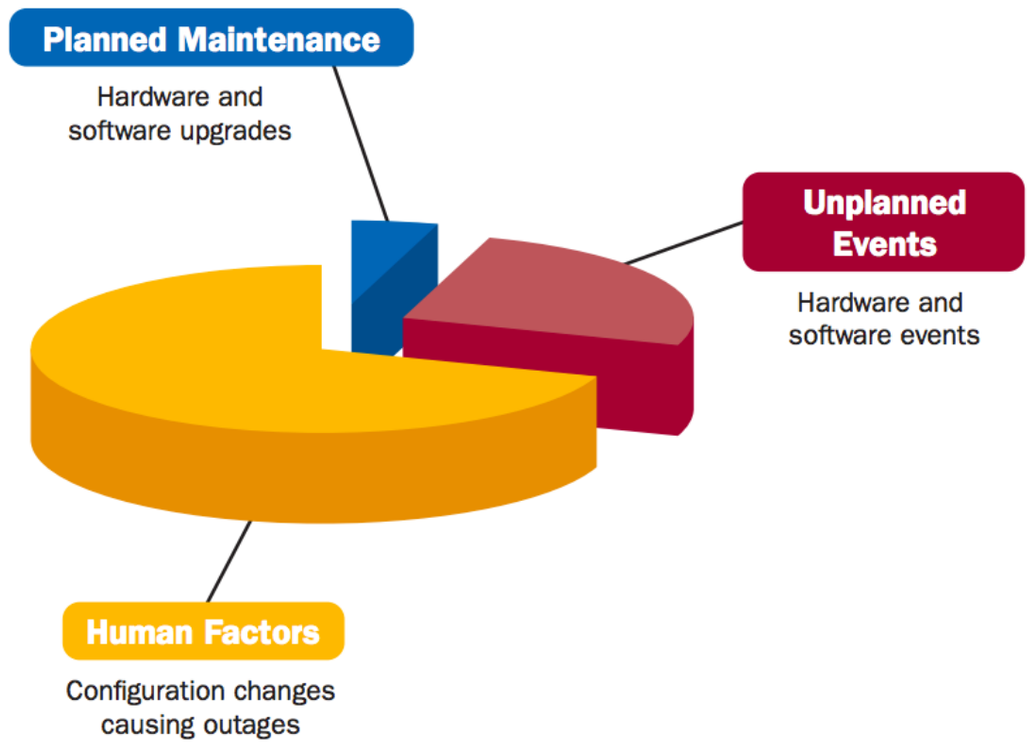


Ryan Beckett (Princeton, MSR)
Ratul Mahajan (MSR)
Todd Millstein (UCLA)
Jitu Padhye (MSR)
David Walker (Princeton)



Configuring Networks is Error-Prone





~60% of network downtime is caused by human error

-Yankee group 2002

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

-Juniper 2008

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

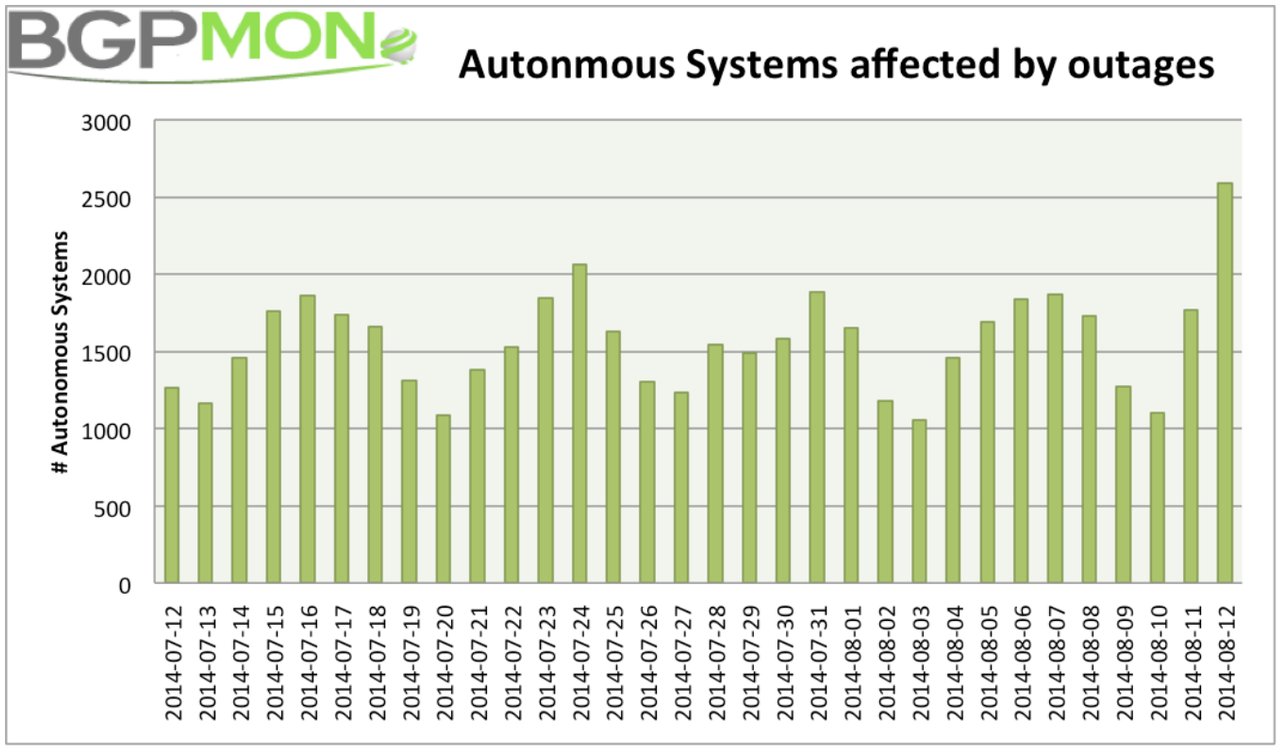
In this paper we examine another source of unreliability: the misconfiguration of the routers that speak BGP. We know from numerous studies of highly reliable systems, such as aircraft, bank databases, and the telephone network, that human operator error can account for 20-70% of system failures [3, 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-

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.



"Close to 3 in 4 of all new prefix advertisements were results of misconfiguration"

~1500 ASes affected by outages every day

Configuring Networks is Error-Prone

(https://www.thousandeyes.com)

Product Solution
About Login (http
Sign Up (https://ww

← Blog Home (/)

Time Warner Cable Outage
Impacts

Posted by Pete Anderson (https://blog.thousande

By now a lot of you have probably read about the Tim
morning I was greeted with a slew of alarms, names t
home office employees without any Internet access.
outage was occurring and quickly opened up the Tho

Time Warner Outage

The alerts started coming in a little before 930 UTC (
inaccessible websites, DNS names failing to resolve,
Internet. Companies that peer with Time Warner expe
services such as supply chain portals (Figure 1).

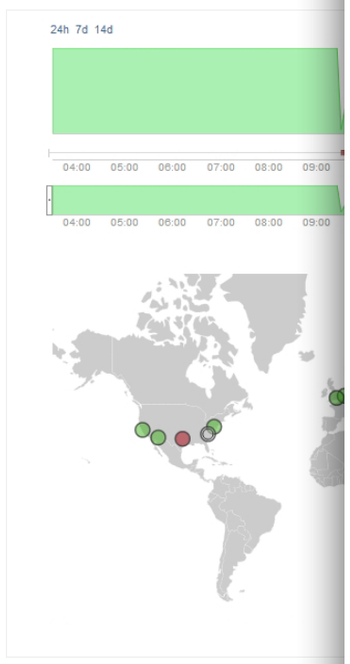


Figure 1: Supply chain portal with limite

I could see right away that users and networks that c
supply chain portal, indicating the issue was in the Ti
service interruption while all the traffic re-routed thro
availability issues continued for the entire duration of
look at the path visualization view to figure out exactl
Normally, two locations (Tokyo and Dallas) transit Road Runner (Time Warner) to reach this supply chain porta
while the rest go through AT&T (Figure 2).

2/5/2016 China routing snafu briefly mangles interweb • The Register

Log in | Sign up Cash'n'Carrion | Whitepapers | The Channel | The Next Platform

DATA CENTER SOFTWARE NETWORKS

Networks ▸ Broadband

China routing snafu b

Cockup, not conspiracy

9 Apr 2010 at 12:24, John Leyden

Bad routing information sourced from China h

Global BGP (Border Gateway Routing) lookup
Telecommunication, apparently accidentally b
Telecommunications, IDG reports. ISPs includ
Qwest and Telefonica accepted ill-thought out

BGP is a core routing protocol which maps op
the net. Several routing options are normally i
equivalent of TomTom publishing routes via S
between London and Paris.

IDC China Telecommunication published ill-co
about 10 per cent of the net - instead of the no
viable routing options by many service provide
time) after China Telecommunications republi
Asia would have been more likely to adopt the
incident were recorded all over the world.

BGPmon.net, a BGP monitoring service, has
described as a prefix hijack, here.

**Although it seems they [IDC China Telec
about 10 per cent of these prefixes propa
include prefixes for popular websites su
www.rapidshare.com and www.geocities
A large number of networks impacted th
include some popular Chinese website s
www.huanqiu.com, www.tianya.cn and w**

A cock-up is suspected, rather than a conspira

**Given the large number of prefixes and s
hijack. Most likely it's because of config
speculation.**

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

The China BGP global routing represents a ra
management. For example, just two weeks ag
internet traffic through a DNS (Domain Name
mortem by internet monitoring firm Renesys h

http://www.theregister.co.uk/2010/04/09/china_bgp_

2/5/2016 Internet-Wide Catastro

Dyn Research

THE NEW HOME OF ren

HOME TOPICS PRESE

DECEMBER 24, 2005 COMMENTS (0)

ENGINEERING TODD UNDERWOOD

Internet-Wide Catastrophe—La

One year ago today TTNet in Turkey (AS9121)
Internet. And unfortunately for the rest of the I
network providers believed them (or at least b
far as anyone knows, it was a mistake, not a m
consequences were far from benign: for sever

« Previous Story

able to reach a large nu
e can take a look at wha
the intervening time.

orning 2004, TTNet (AS9121) started announcing

http://research.dyn.com/2005/12/internetwide-nearcatastrophela/ 1/5

Sign In | Register

NETWORKWORLD

YouTube/Pakistan incident: Could something similar whack your site?

Configuring BGP properly is key to avoidance, 'Net registry official says

By Carolyn Duffy Marsan
Network World | Mar 10, 2008 1:00 AM PT

In light of Pakistan Telecom/YouTube incident, Internet registry official explains how you can avoid having your web site victimized by such an attack.

When Pakistan Telecom blocked YouTube's traffic one Sunday evening in February, the ISP created an international incident that wreaked havoc on the popular video site for more than two hours.

RIPE NCC, the European registry for Internet addresses, has conducted an analysis of what happened during Pakistan Telecom's hijacking of YouTube's traffic and the steps that YouTube took to stop the attack.

We posed some questions to RIPE NCC's Chief Scientist Daniel Karrenberg about the YouTube incident. Here's what he had to say:

How frequently do hijacking incidents like the Pakistan Telecom/YouTube incident happen?

Misconfigurations of iBGP (internal BGP, the protocol used between the routers in the same Autonomous System) happen regularly and are usually the result of an error. One such misconfiguration caused the Pakistan Telecom/YouTube incident. It appears that the Pakistan Telecom/YouTube incident was not an "attack" as some have labeled it, but a configuration error. (See Columnist Johna Till Johnson's take on the topic.)

What is significant about the YouTube incident?

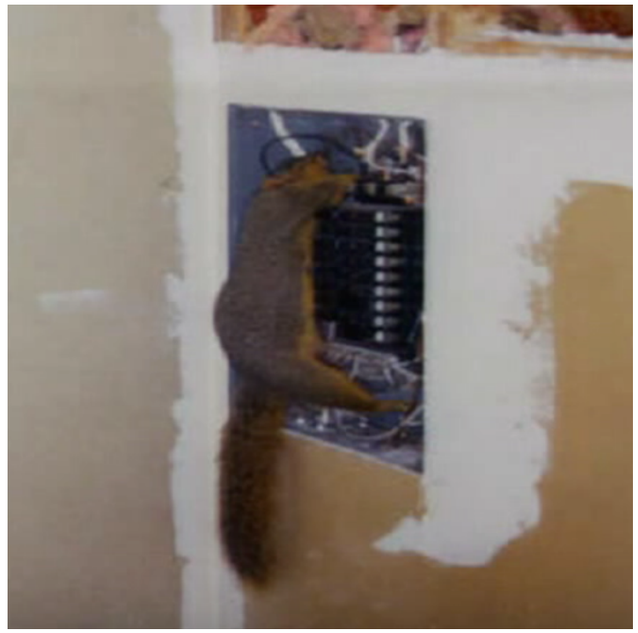
Objectives: Network-wide

- Prefer traffic to go through AT&T over Sprint
- Don't use our network as transit between A and B
- Traffic must stay within national boundaries
- Adhere to policies even when failures occur

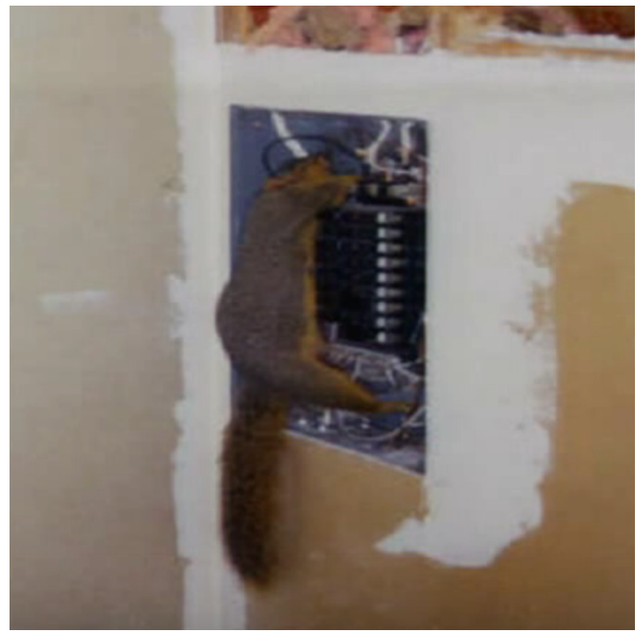
Objectives GAP **Mechanisms**

Objectives: Network-wide

- Prefer traffic to go through AT&T over Sprint
- Don't use our network as transit between A and B
- Traffic must stay within national boundaries
- Adhere to policies even when failures occur



Objectives GAP **Mechanisms**



Objectives: Network-wide

- Prefer traffic to go through AT&T over Sprint
- Don't use our network as transit between A and B
- Traffic must stay within national boundaries
- Adhere to policies even when failures occur

Mechanisms: Device-by-Device

- Local decisions made independently on each device
- Several device-level actions together imply some higher-level behavior

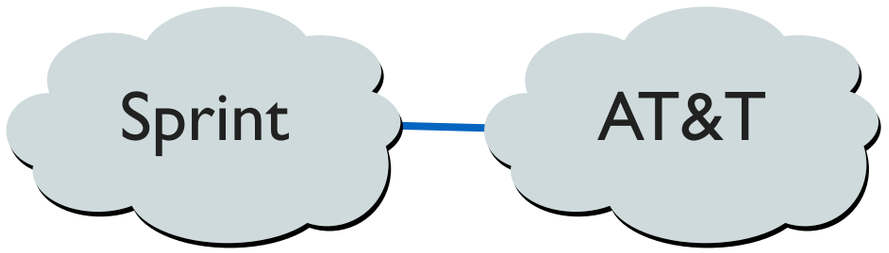
> *"a large backbone network may have a thousand different policies configured across hundreds of routers"* [Feamster 2005]

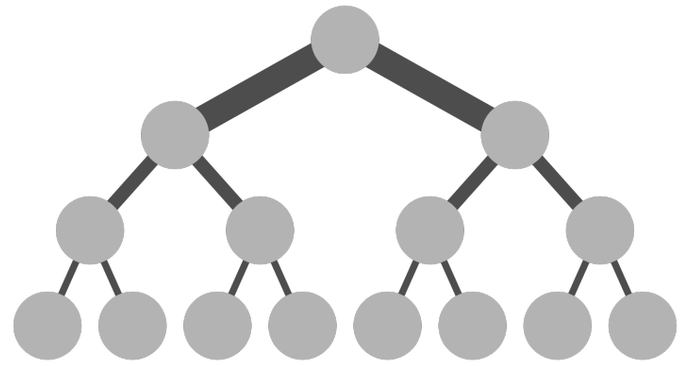
- Failures interact with local decision-making algorithms in complex ways

Why use BGP?

BGP in the Wild

- Inter-domain routing between Autonomous Systems (AT&T, Sprint)
- Intra-domain routing in data centers





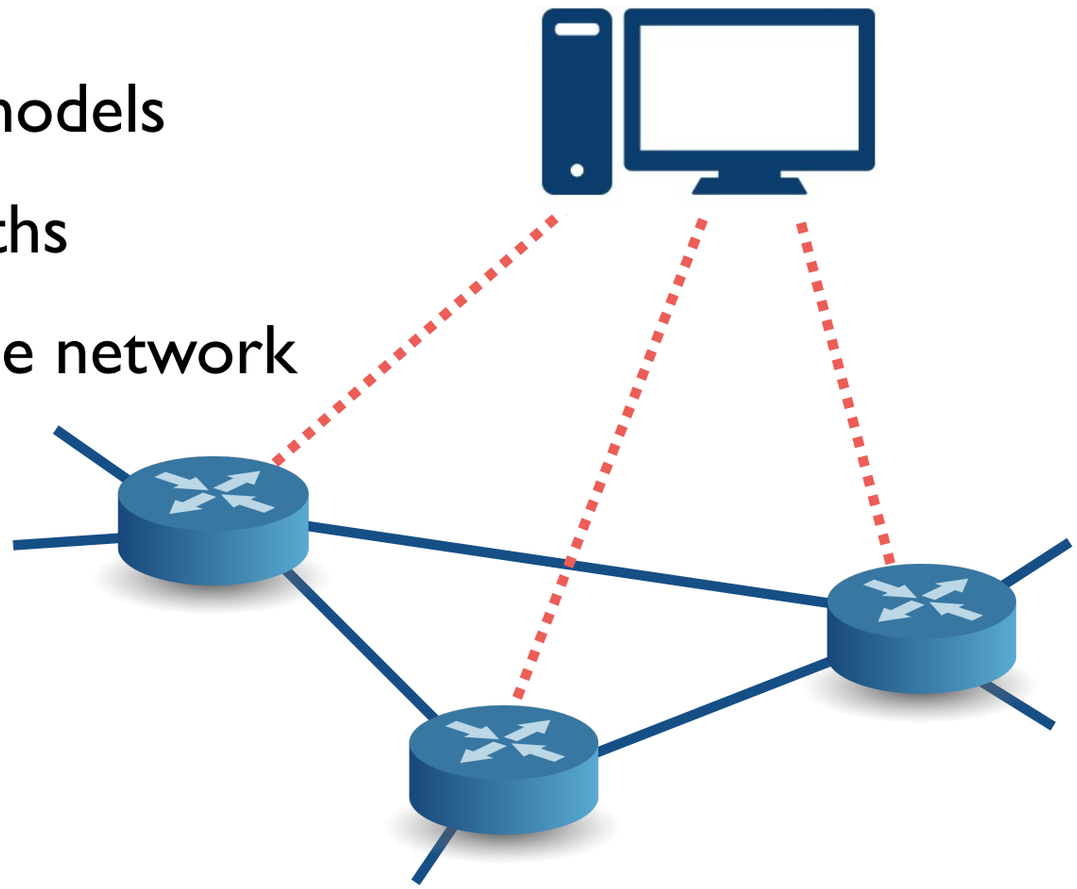
Advantages

- Allows for *local* policy among nodes
- Massively *scalable* — the only routing protocol that scales to hundreds of thousands of IP prefixes
- Fully *distributed* — does not require global knowledge of network state

What about SDN?

Software Defined Networks

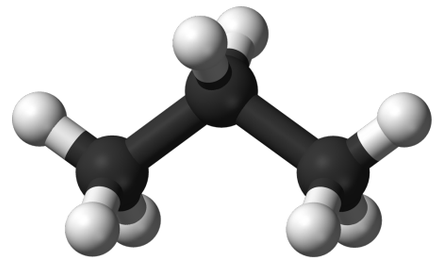
- Simpler, centralized programming models
- Network-wide abstractions like paths
- Centralized controller programs the network



But they are not a panacea:

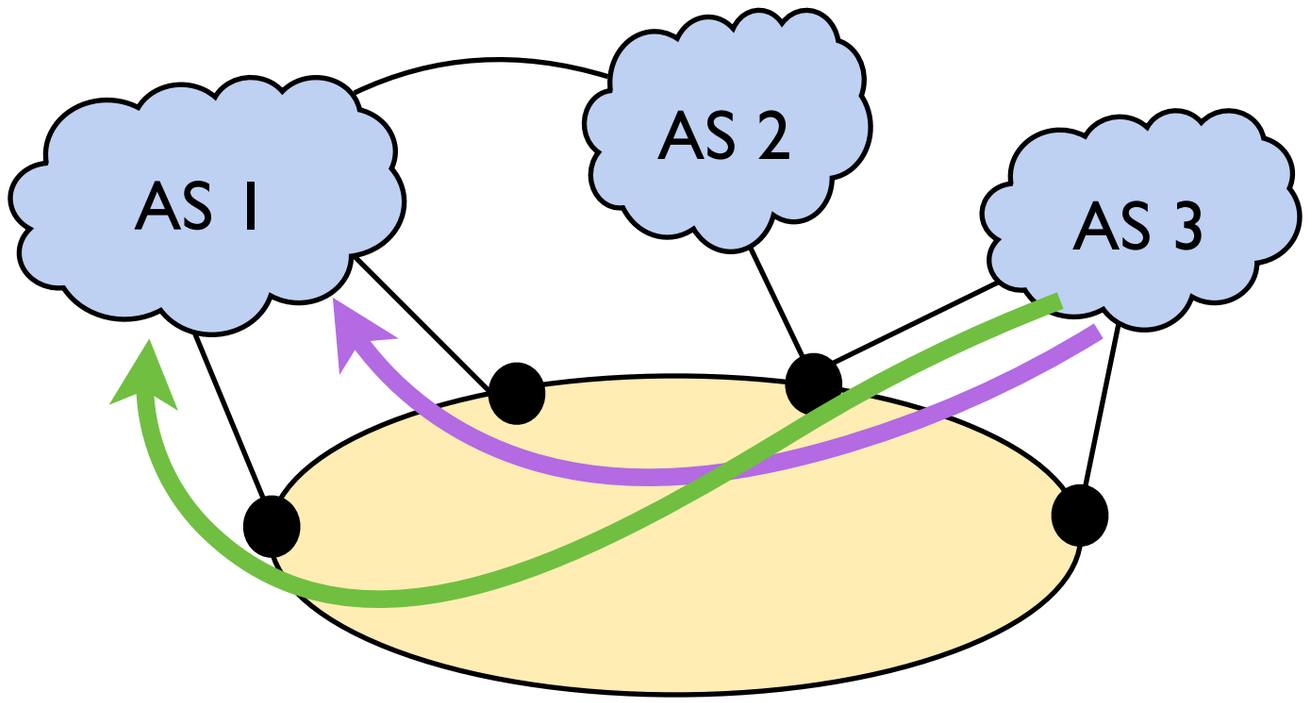
- Do not usually help with inter-domain routing
- Latency, and scalability can be problematic for large networks — often requires bringing back distributed control planes.
- Require careful design and engineering to be robust to failures
- Their implementations don't exploit existing network infrastructure

Propane: Programming a Distributed Control Plane

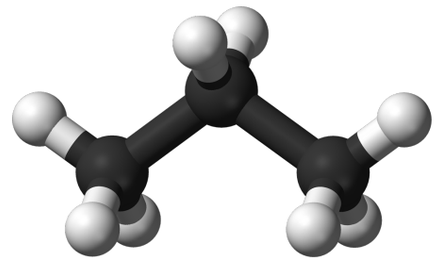


I) Language for expressing high-level operator objectives with:

- Network-wide programming abstraction
- Uniform abstractions for intra- and inter-domain routing
- Paths constraints and relative preferences with fall-backs in case of failures



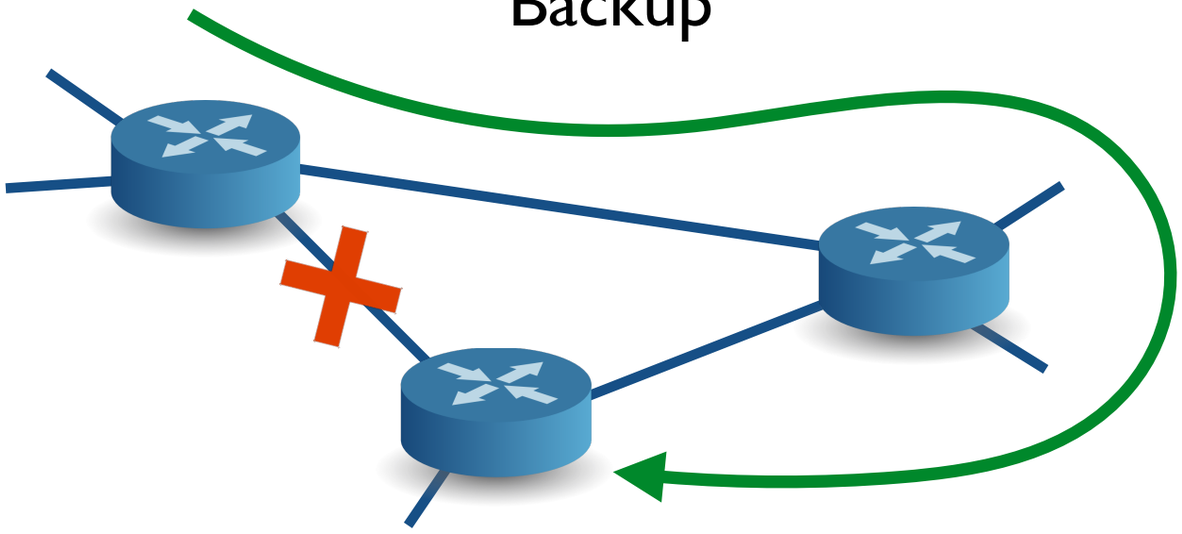
Propane: Programming a Distributed Control Plane



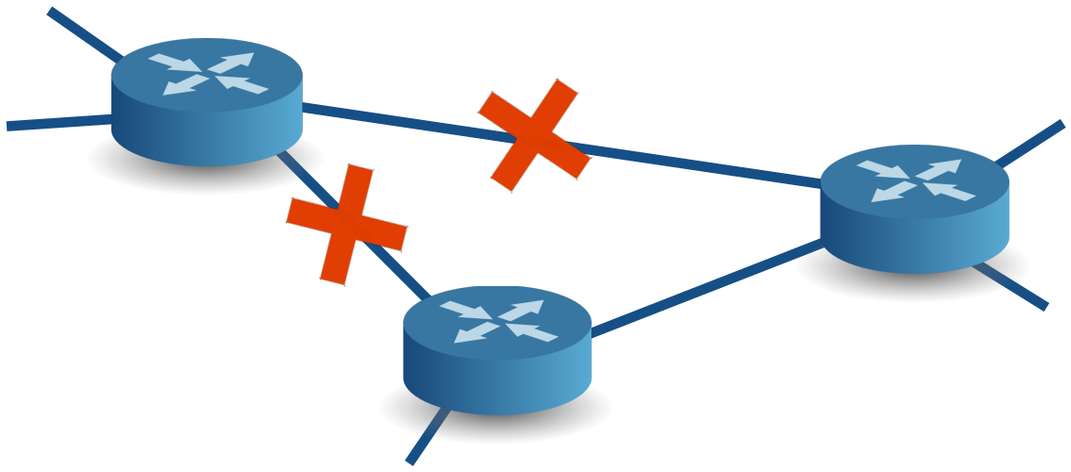
2) Compiler to generate a low-level distributed implementation:

- Efficient algorithms to synthesize a set of *policy-compliant* BGP configs
- Static analysis guarantees policy compliance under all failures

Backup



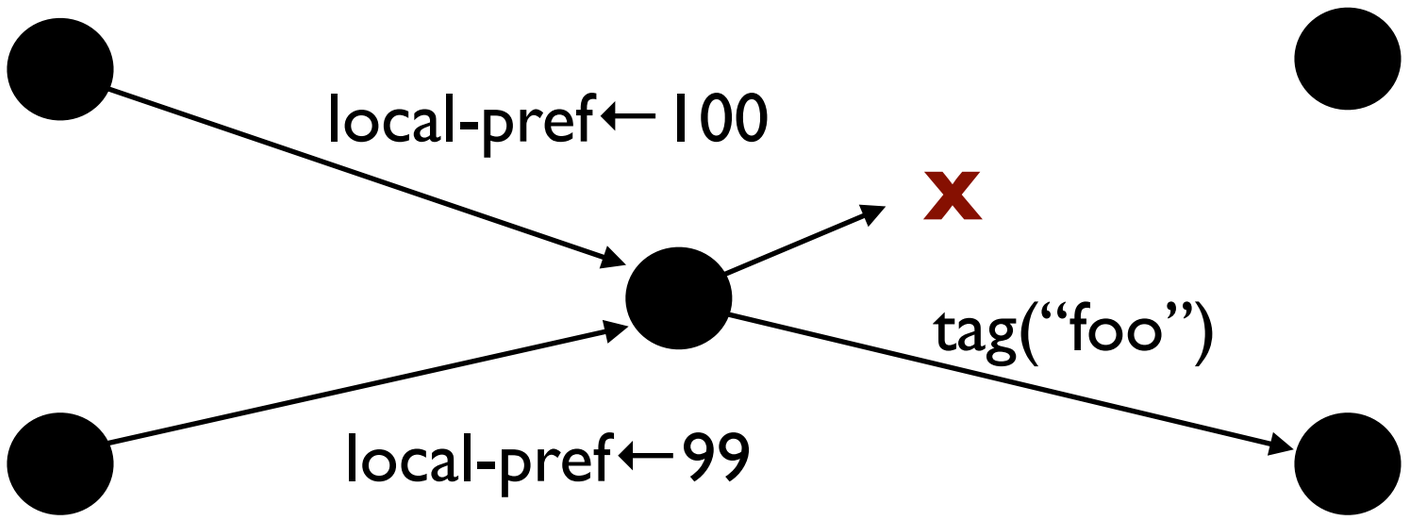
Network partition — drop



Border Gateway Protocol (BGP)

- Routers send advertisements for a given prefix (e.g., 16.4.0.0/16)
- Advertisements contain the AS path traffic would take
- Each router applies local policy to choose best route

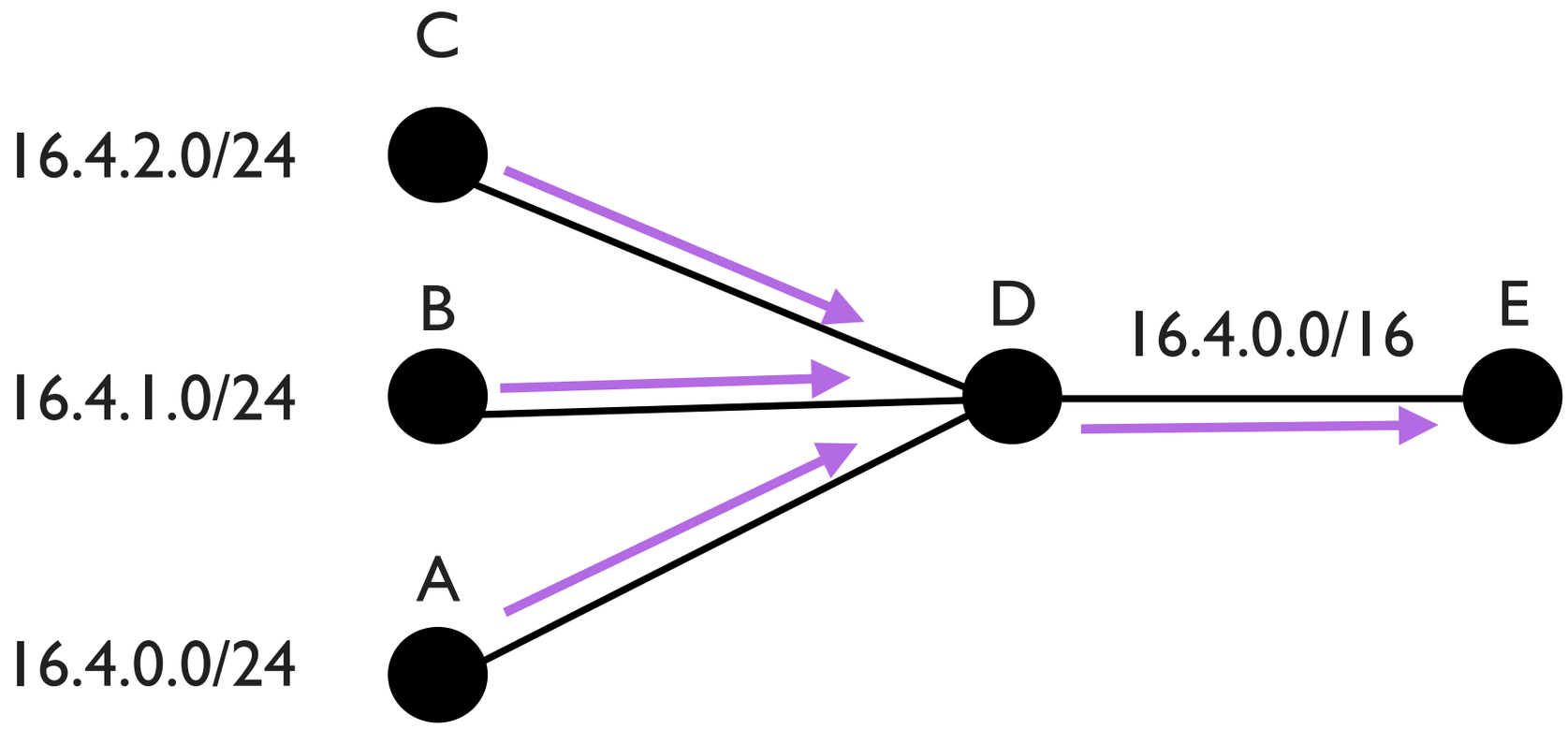
1. Import filters drop advertisements or modify attributes
2. Highest preference is chosen. Path length as a tie breaker
3. Export filters drop advertisements or modify attributes



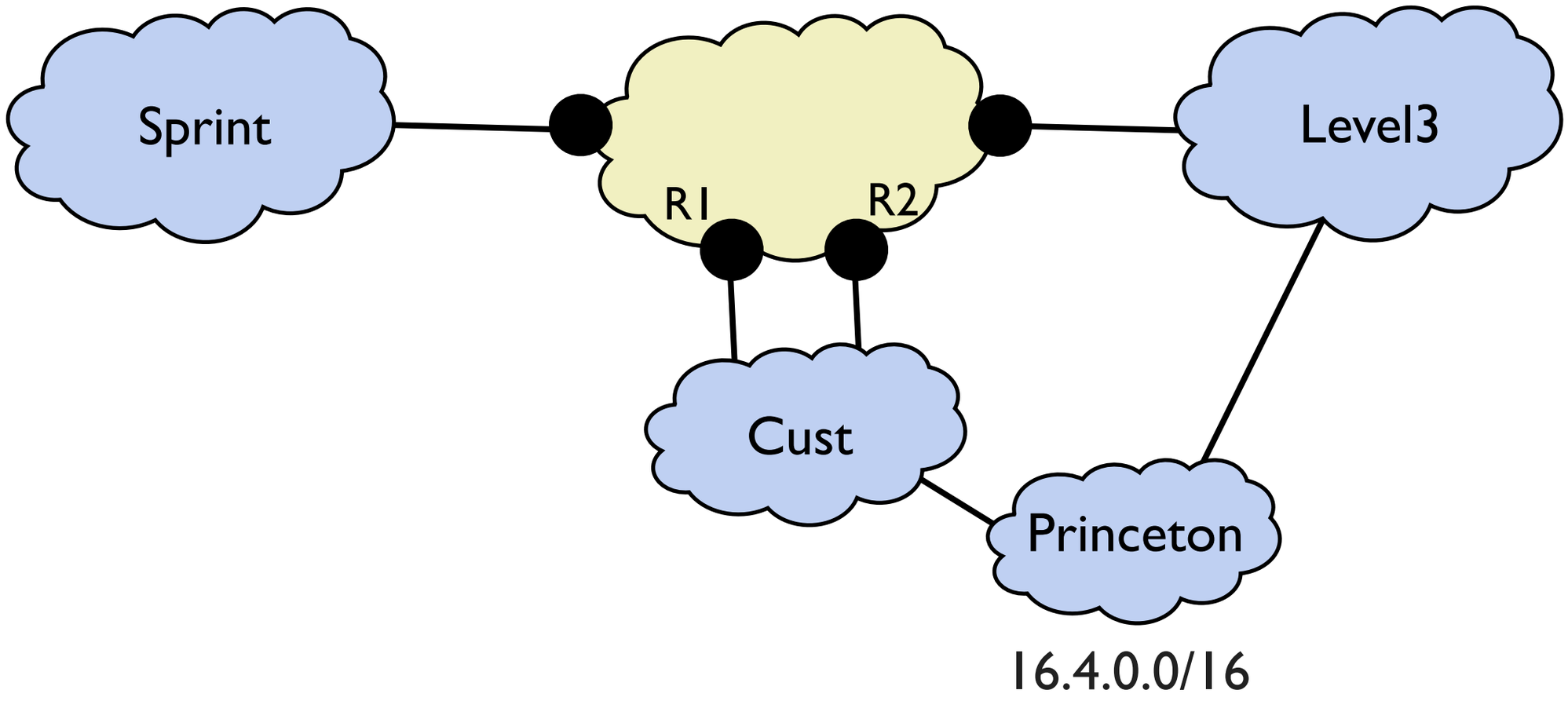
Border Gateway Protocol (BGP)

Aggregation (Route summarization)

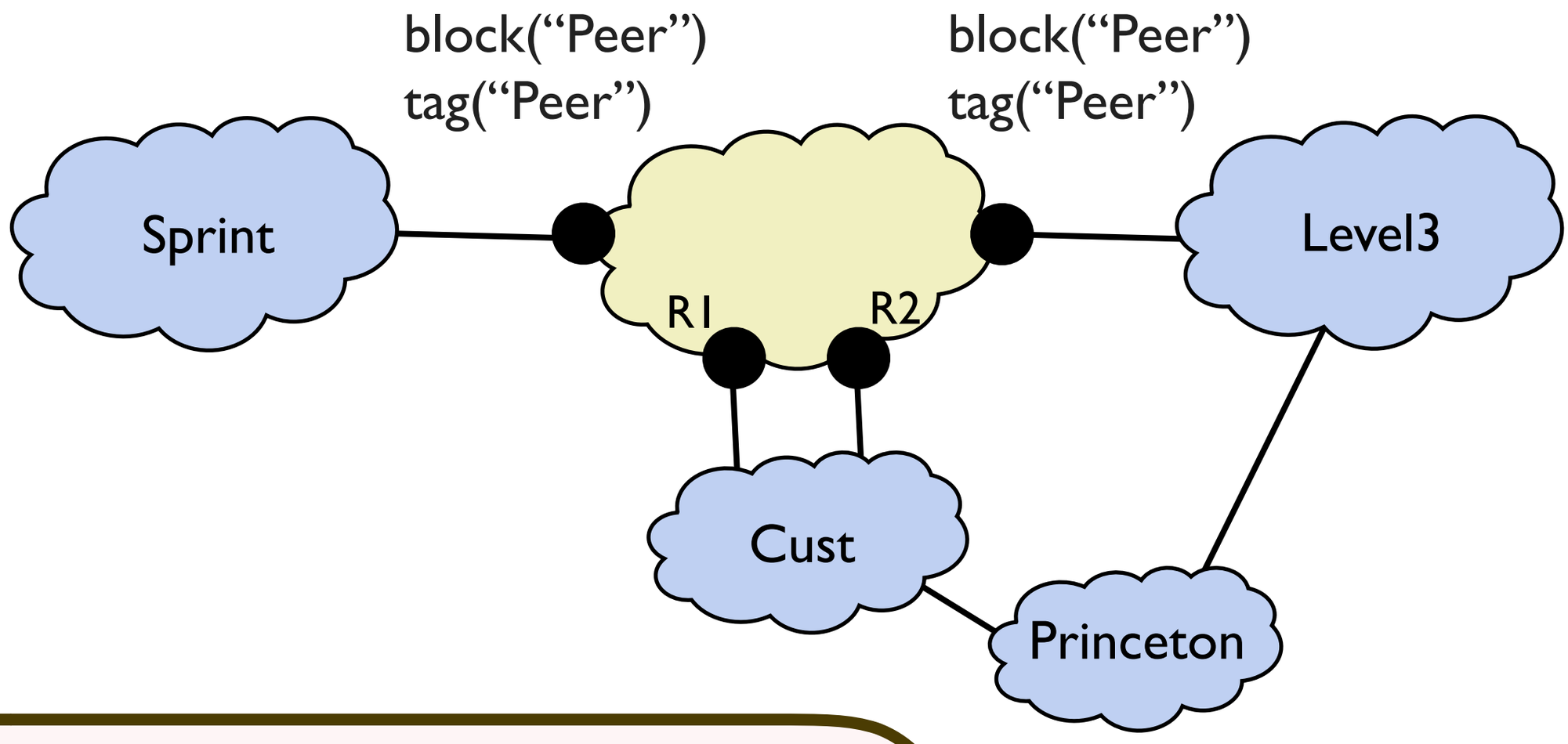
- Advertise a more general subnetwork
- Information hiding for routers
- Reduces router table size and improves network stability



Example 1: Backbone Network



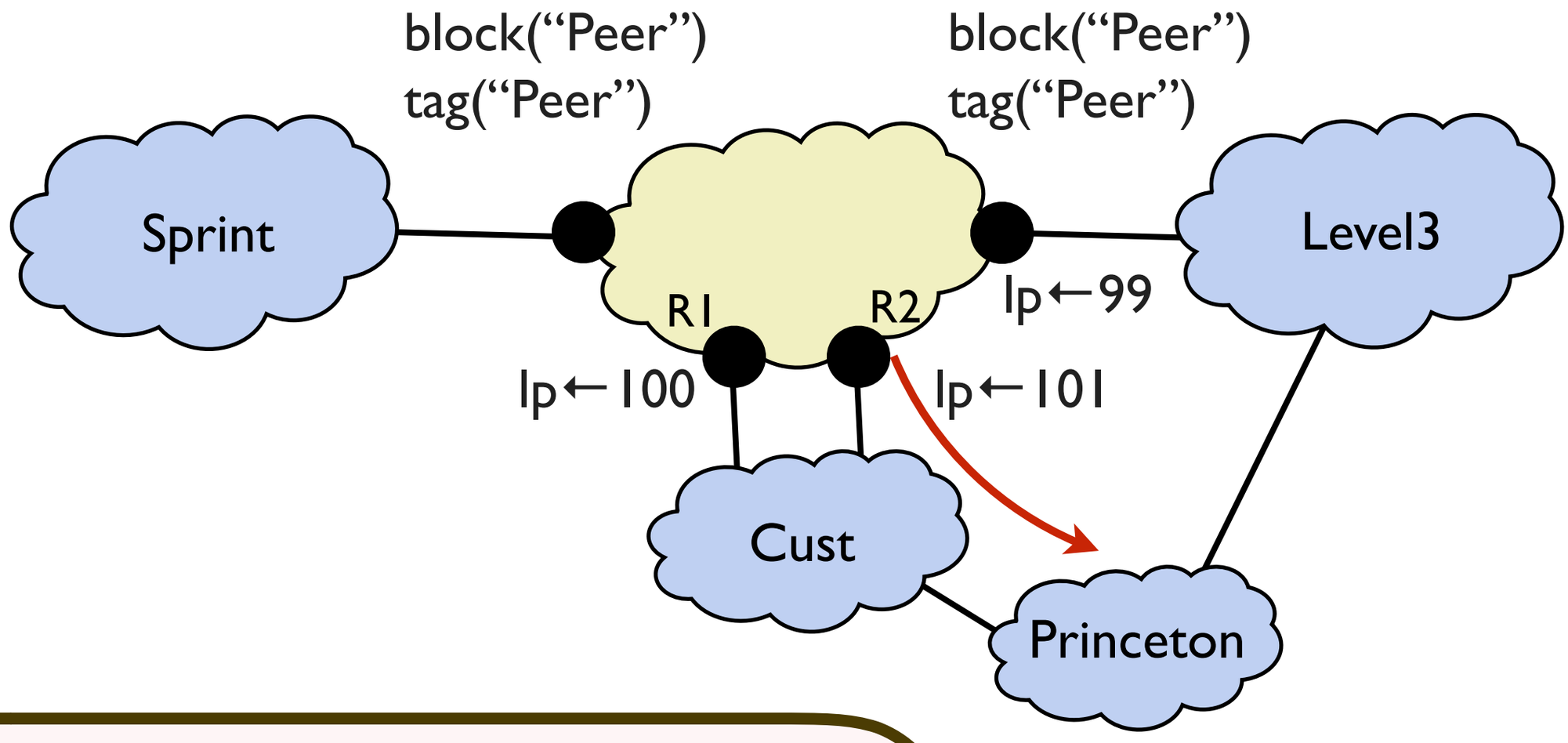
Example 1: Backbone Network



Specification

- Prevent transit between peers ($$$)

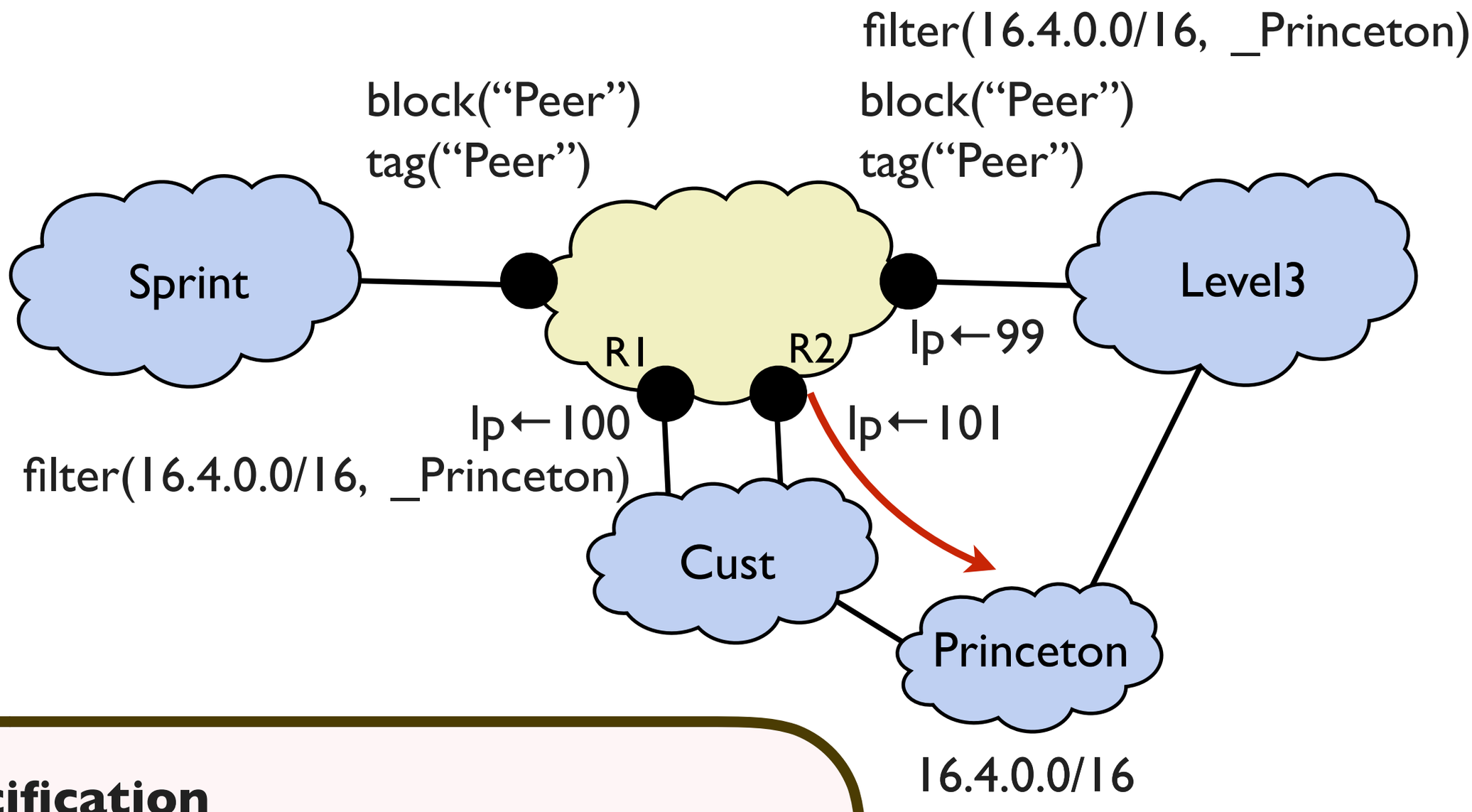
Example 1: Backbone Network



Specification

- Prevent transit between peers ($$$)
- Prefer R2 > R1 > Peer ($$$)

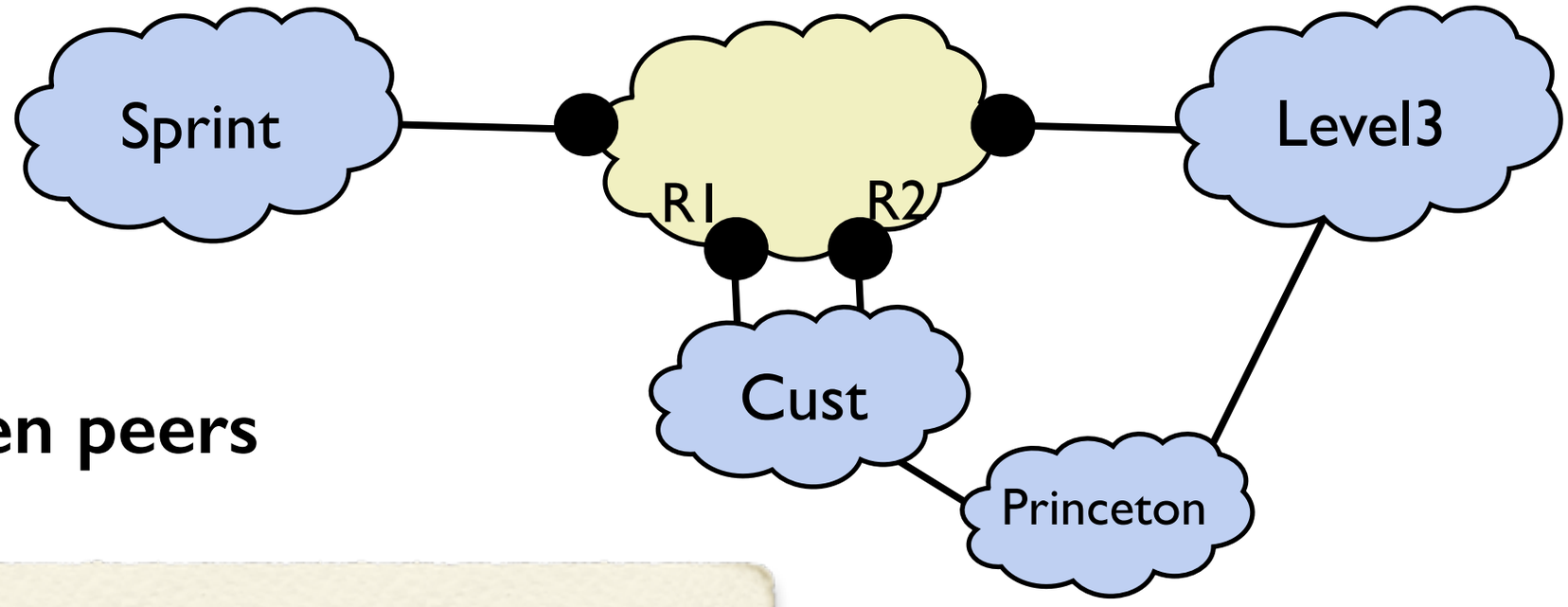
Example 1: Backbone Network



Specification

- Prevent transit between peers ($$$)
- Prefer R2 > R1 > Peer ($$$)
- Filter customer by prefix (security)

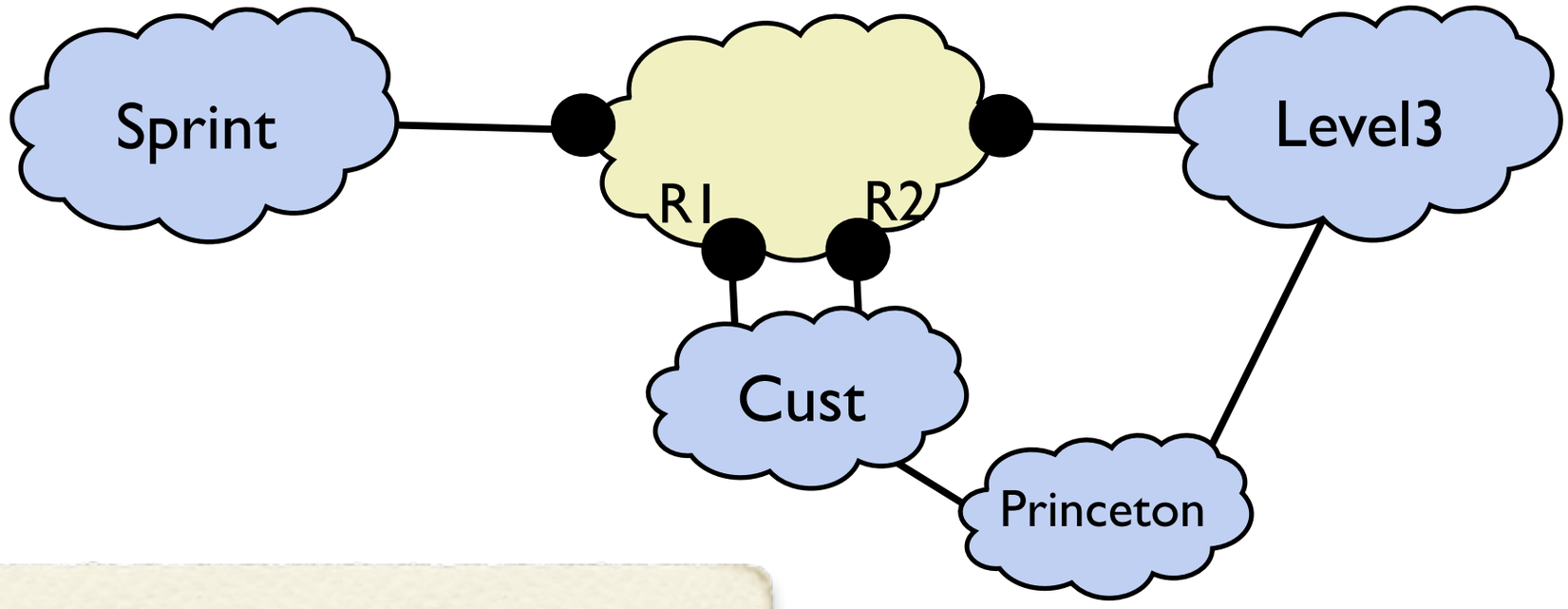
Example 1: Backbone Network



Prevent transit between peers

```
define Peer = {Sprint, Level3}

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

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

Example 1: Backbone Network

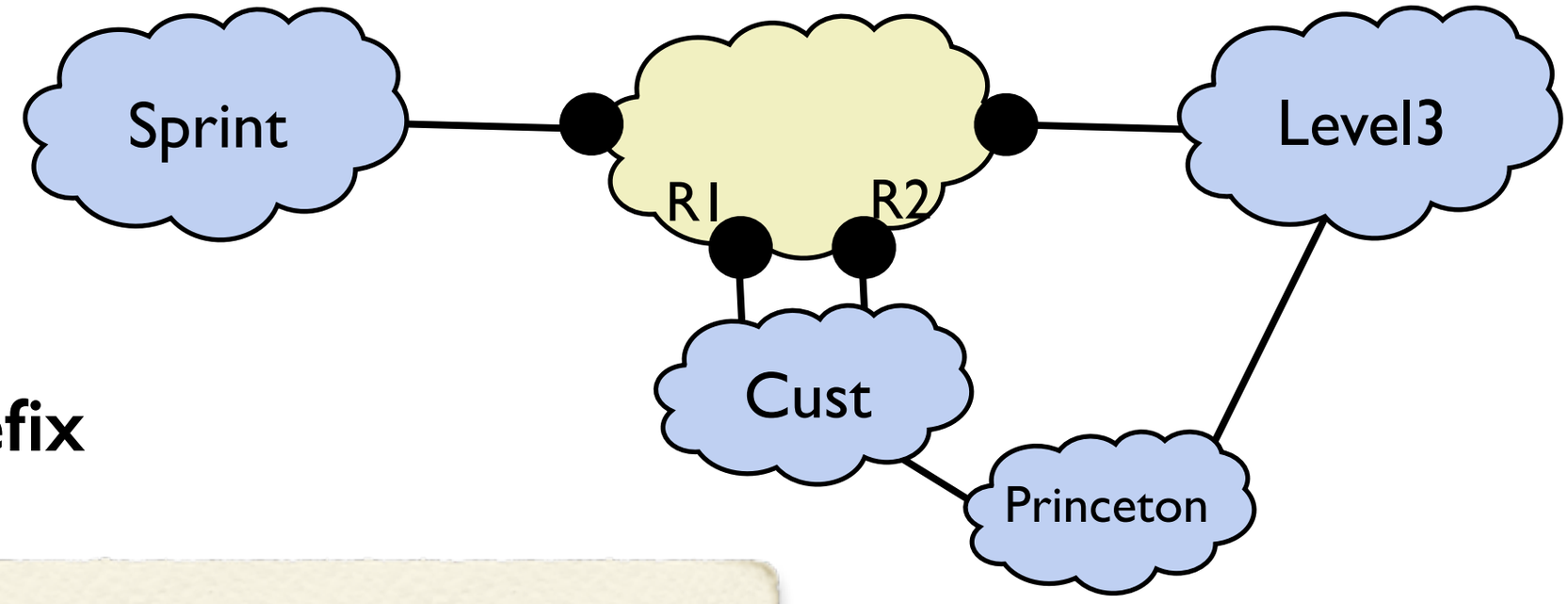


Prefer R2 > R1 > Peer

```
...

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

16.4.0.0/16

Example 1: Backbone Network



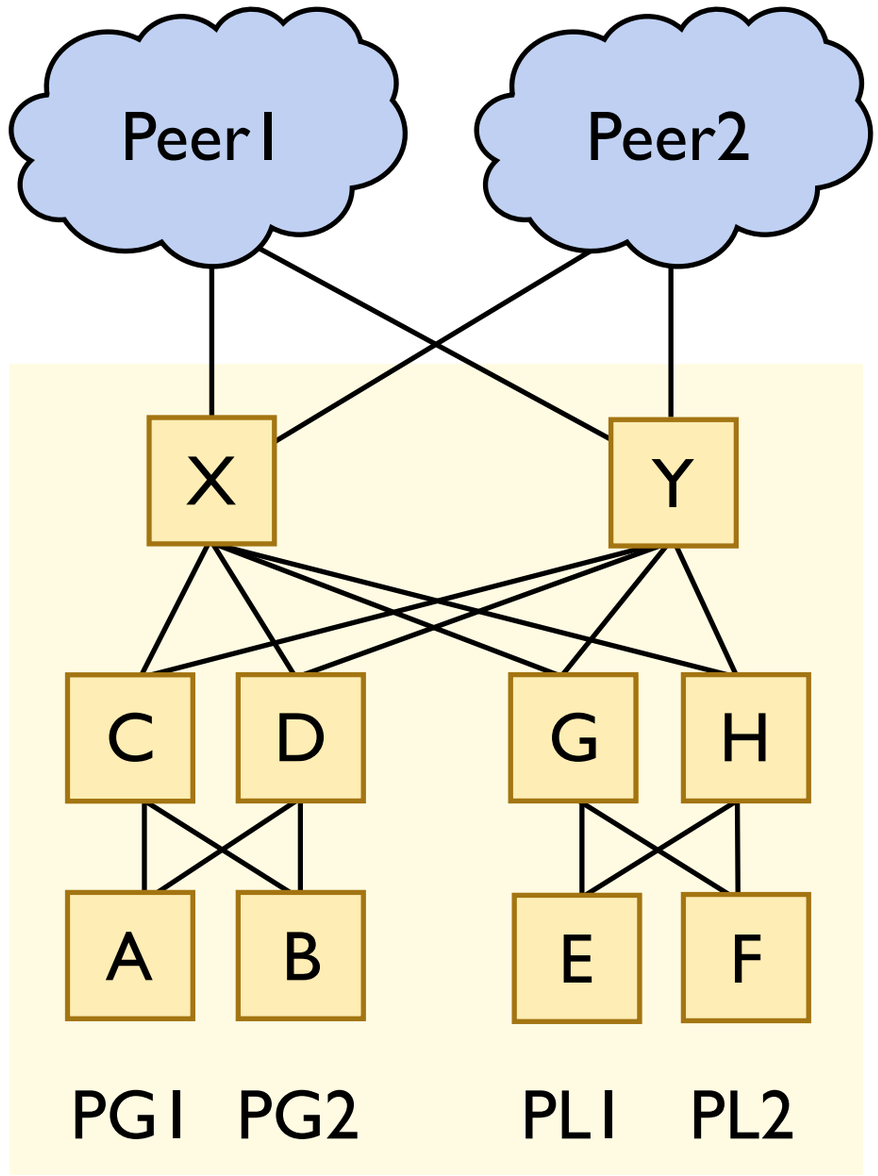
Filter customer by prefix

```
...

define Ownership = {
      16.4.0.0/16  => end(Princeton),
}

define Main =
      Preferences & Ownership & NoTransit
```

Example 2: A Data Center Network

Specification

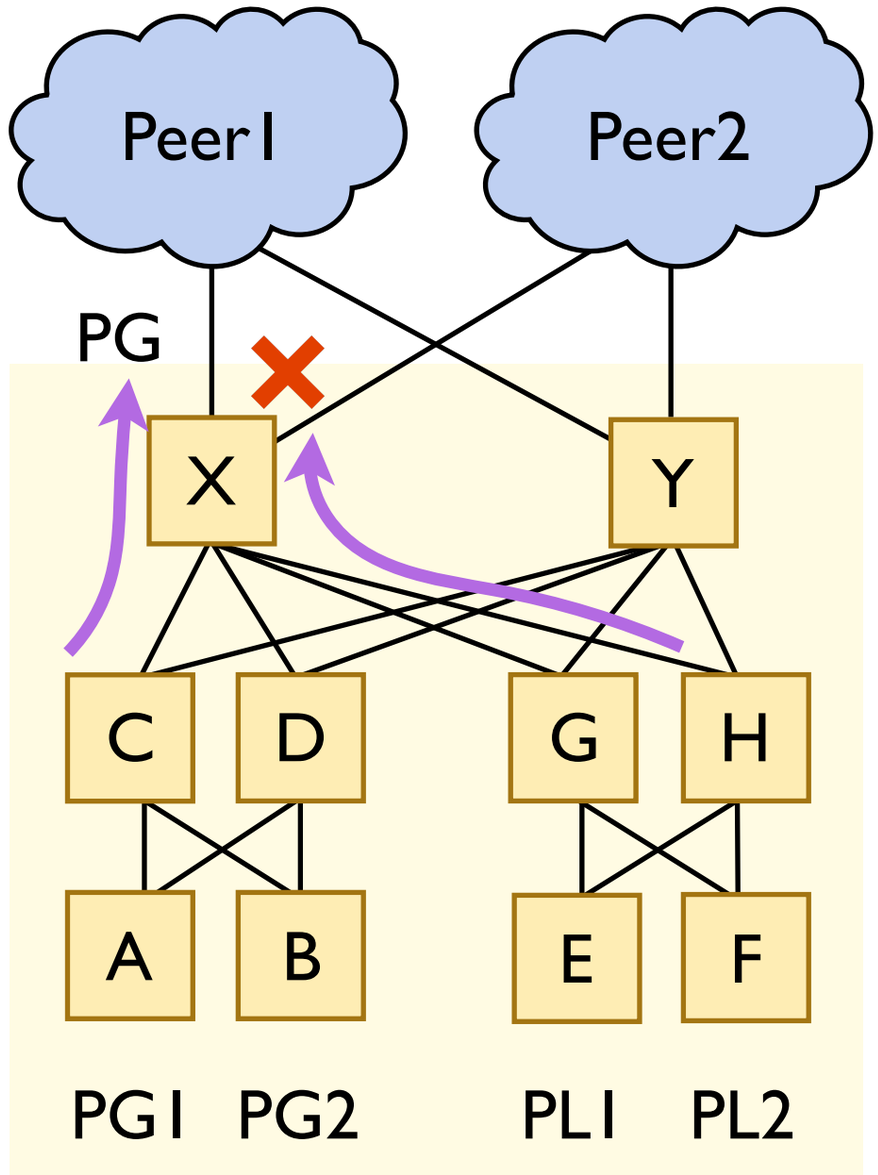
- Local prefixes reachable only internally
- Global prefixes reachable externally
- Aggregate global prefixes as PG
- Prefer leaving through Peer1 over Peer2
- Prevent transit traffic between peers



Example 2: A Data Center Network

Specification

- 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



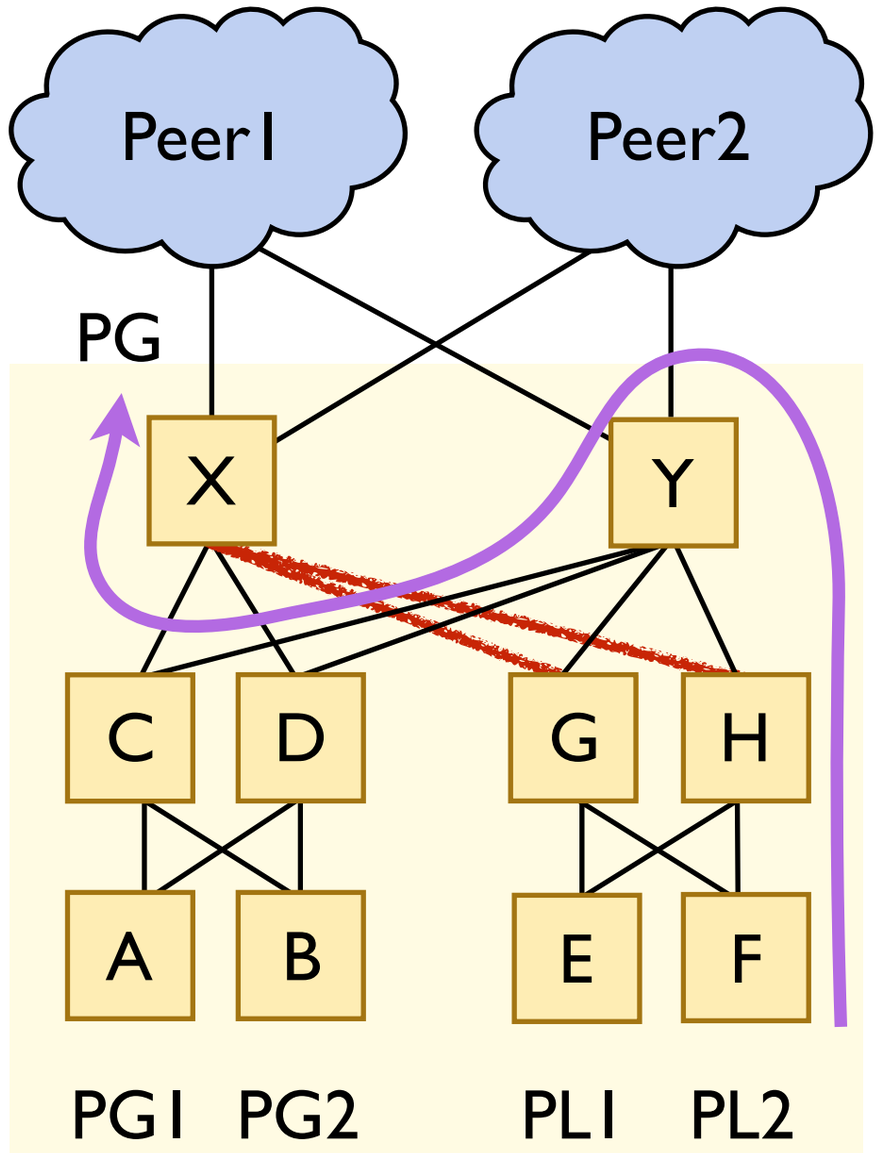
Attempt (1)

- X and Y allow adv. from C,D to peers
- X and Y do not allow adv. from G,H to peers
- Aggregate to PG externally

Example 2: A Data Center Network

Specification

- 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



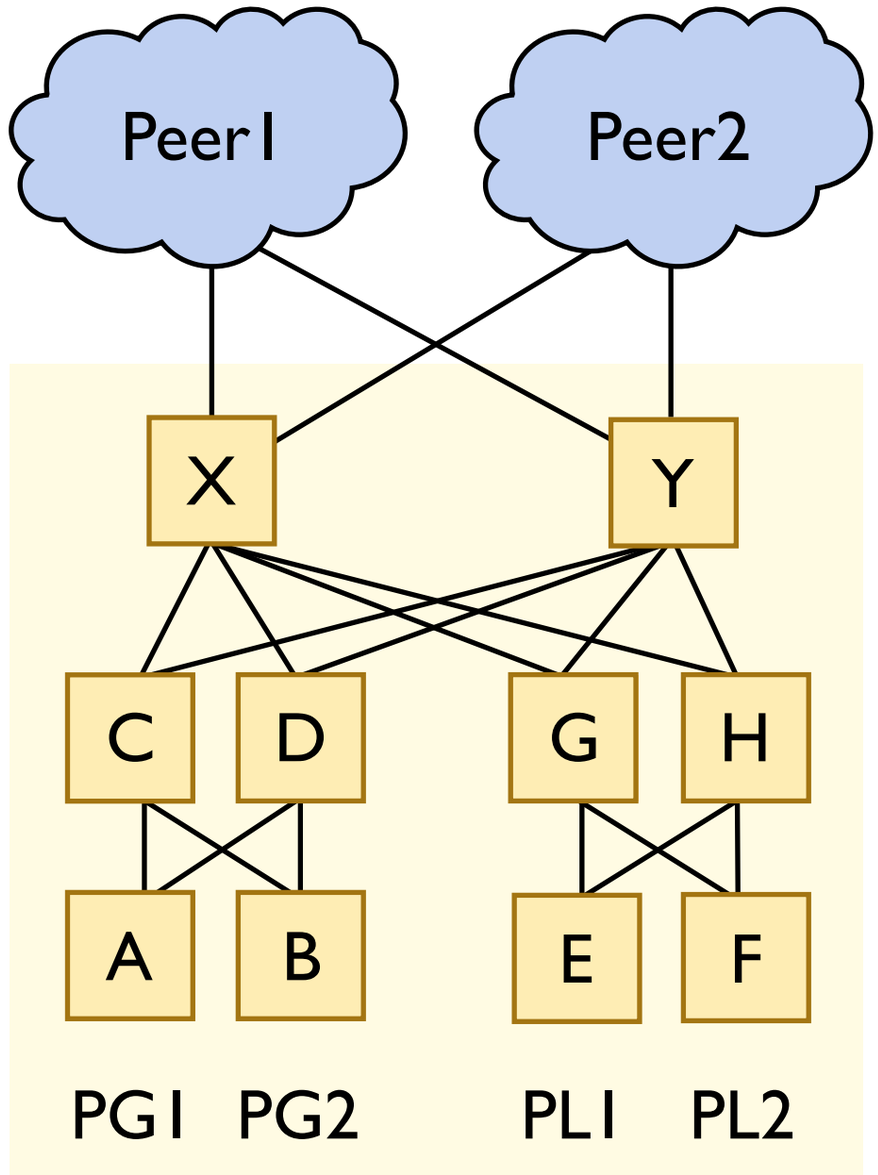
Attempt (1)

- X and Y allow adv. from C,D to peers
- X and Y do not allow adv. from G,H to peers
- Aggregate to PG externally

Example 2: A Data Center Network

Specification

- 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



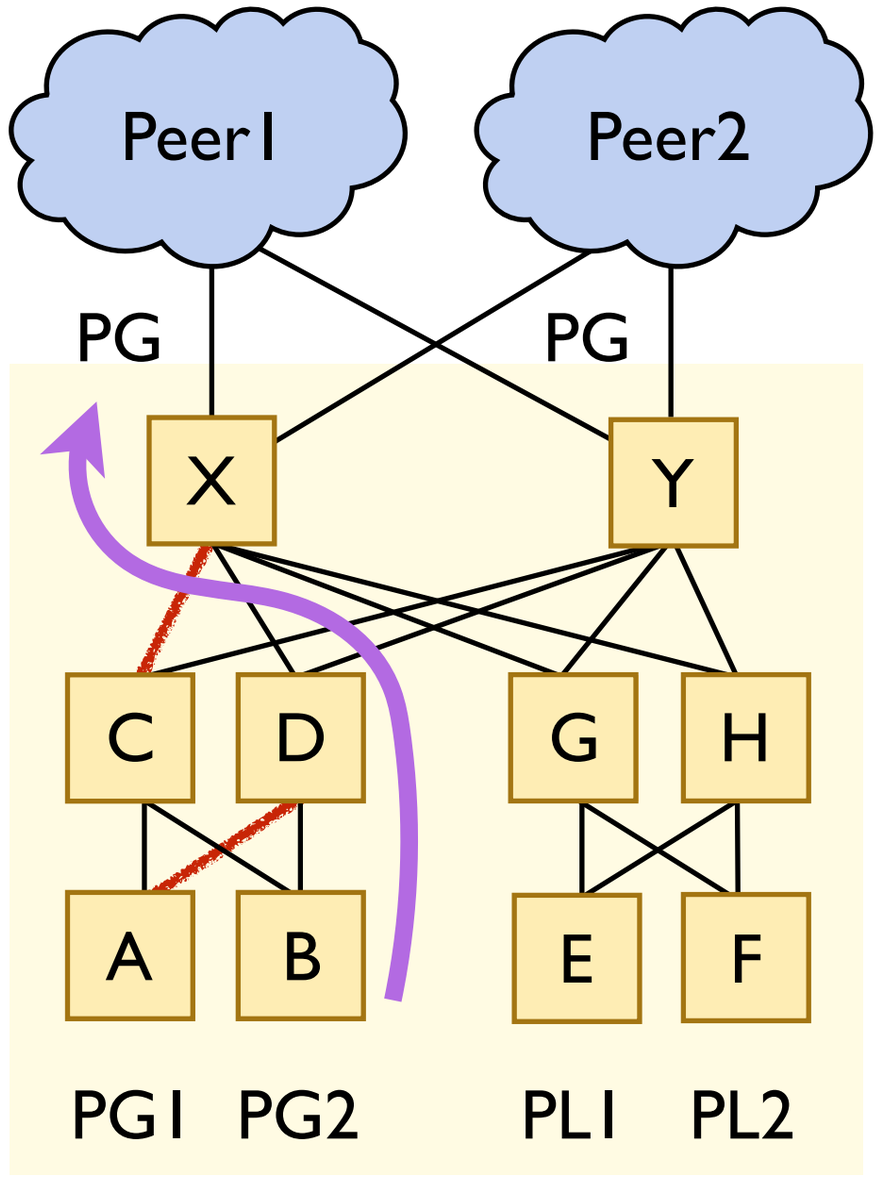
Attempt (1)

- X and Y allow adv. from C,D to peers
- X and Y do not allow adv. from G,H to peers
- **X and Y drop routes through each other**
- Aggregate to PG externally

Example 2: A Data Center Network

Specification

- 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



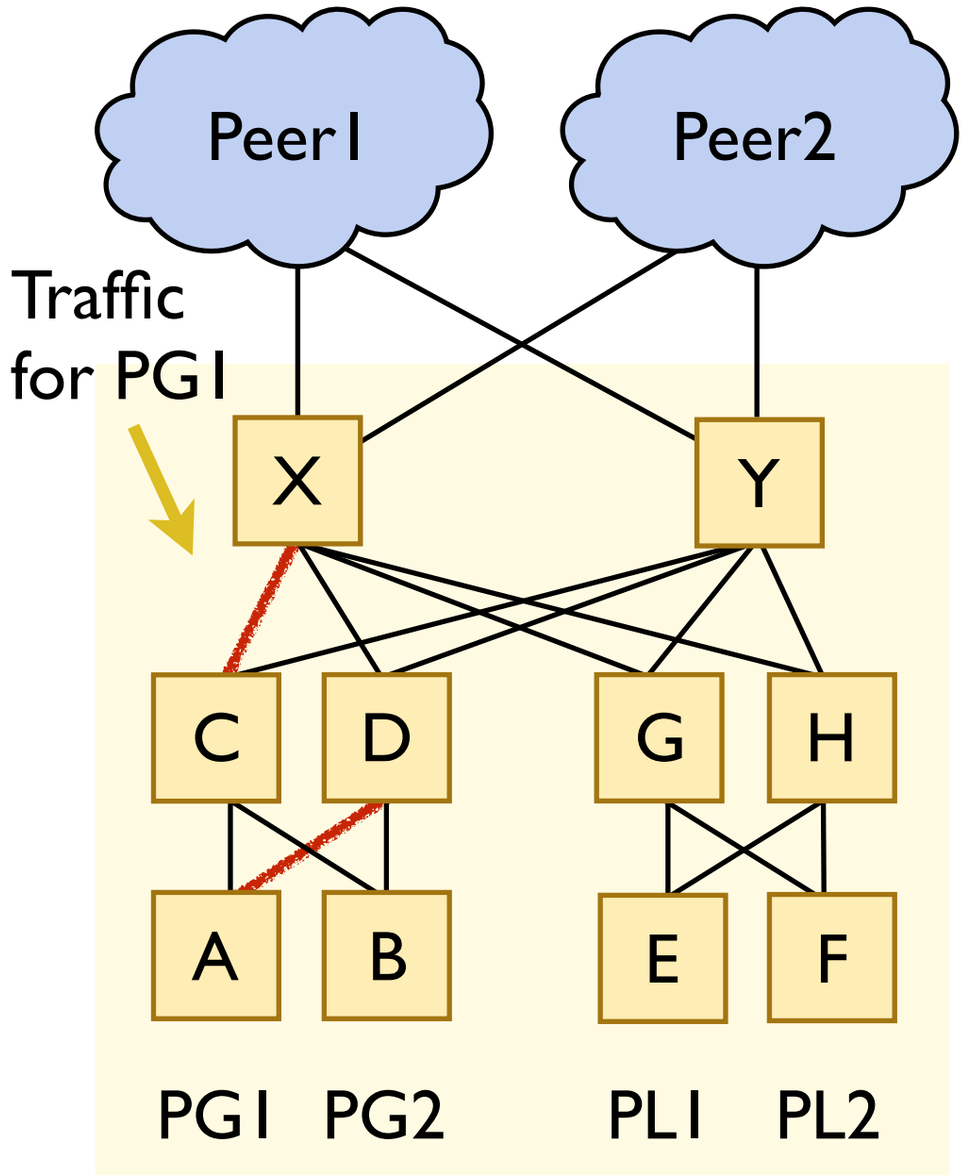
Attempt (1)

- X and Y allow adv. from C,D to peers
- X and Y do not allow adv. from G,H to peers
- **X and Y drop routes through each other**
- Aggregate to PG externally

Example 2: A Data Center Network

Specification

- 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)

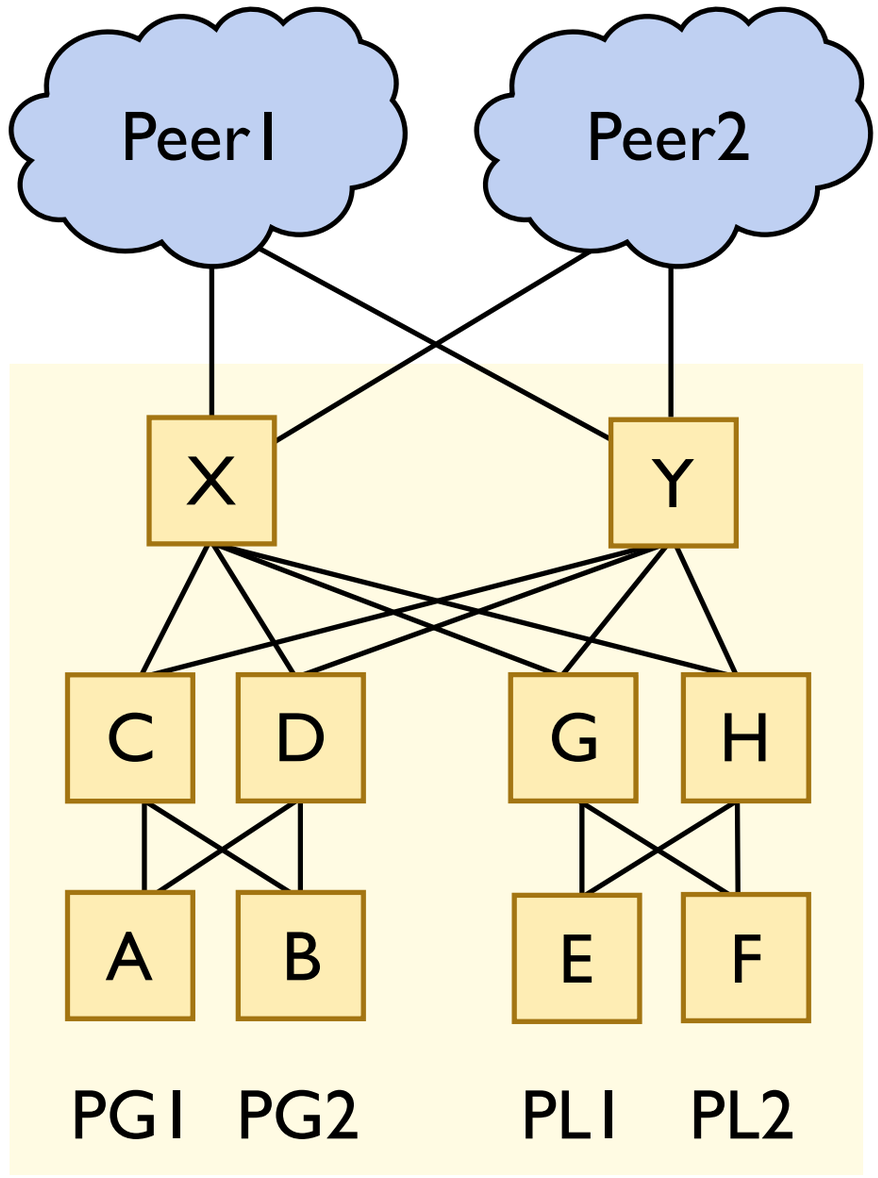
- X and Y allow adv. from C,D to peers
- X and Y do not allow adv. from G,H to peers
- **X and Y drop routes through each other**
- Aggregate to PG externally

Aggregation Black hole!

Example 2: A Data Center Network

Basic reachability for prefixes

```
define PL1 = 1.0.0.0/24
...

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



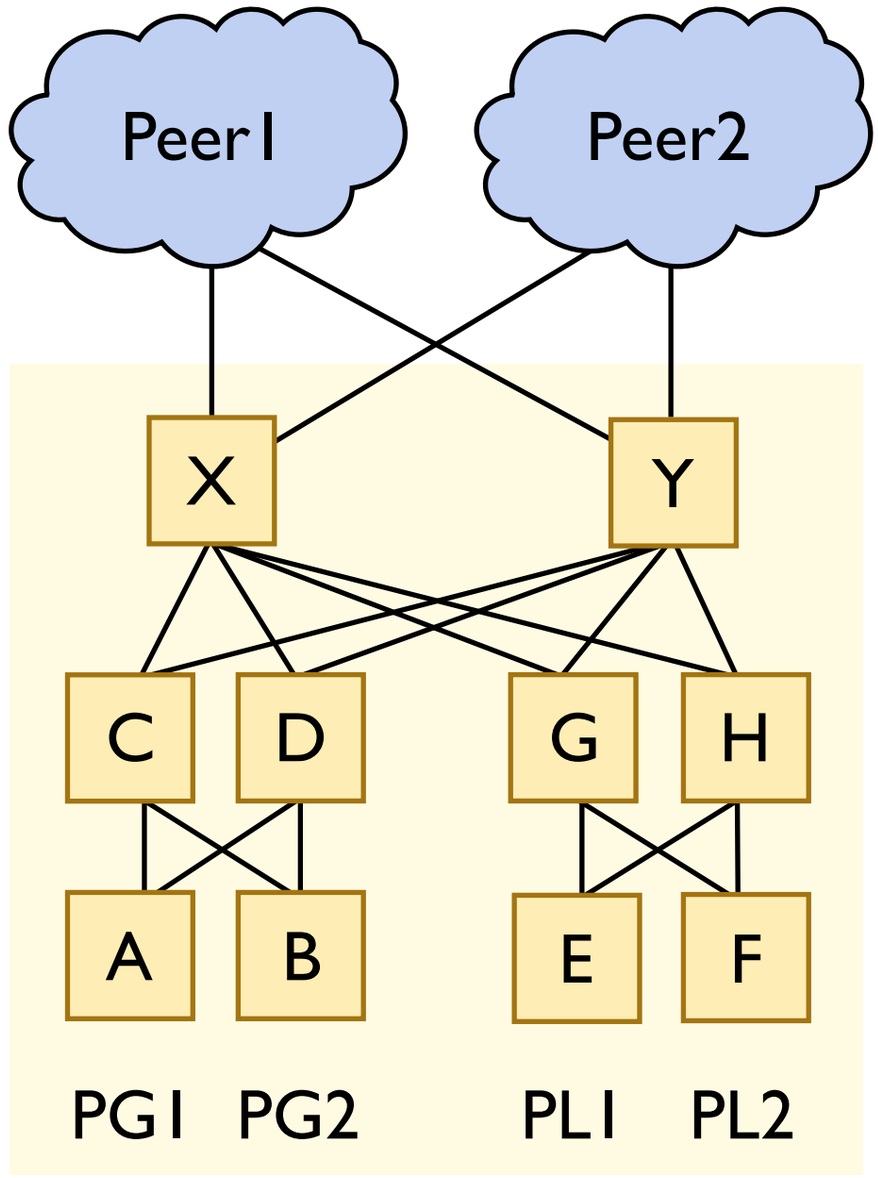
Global Services

Local Services

Example 2: A Data Center Network

Local services stay internal

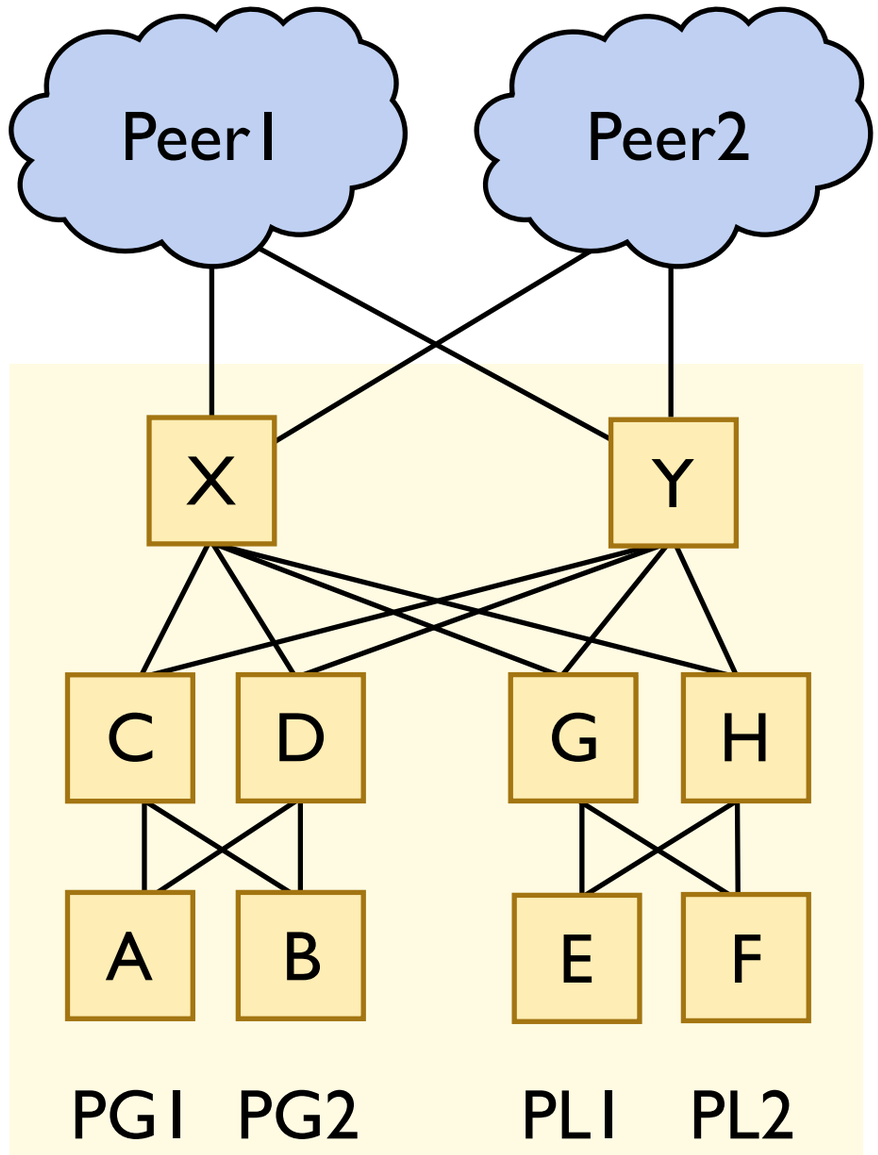
```
...

# define in = A | B | C | D | E | F | G | H
define Locality = {
    PL1 or PL2 => always(in)
}
```



Example 2: A Data Center Network

Prevent transit traffic

```
...

# define in = A | B | C | D | E | F | G | H
define Locality = {
      PL1 or PL2 => always(in)
}

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



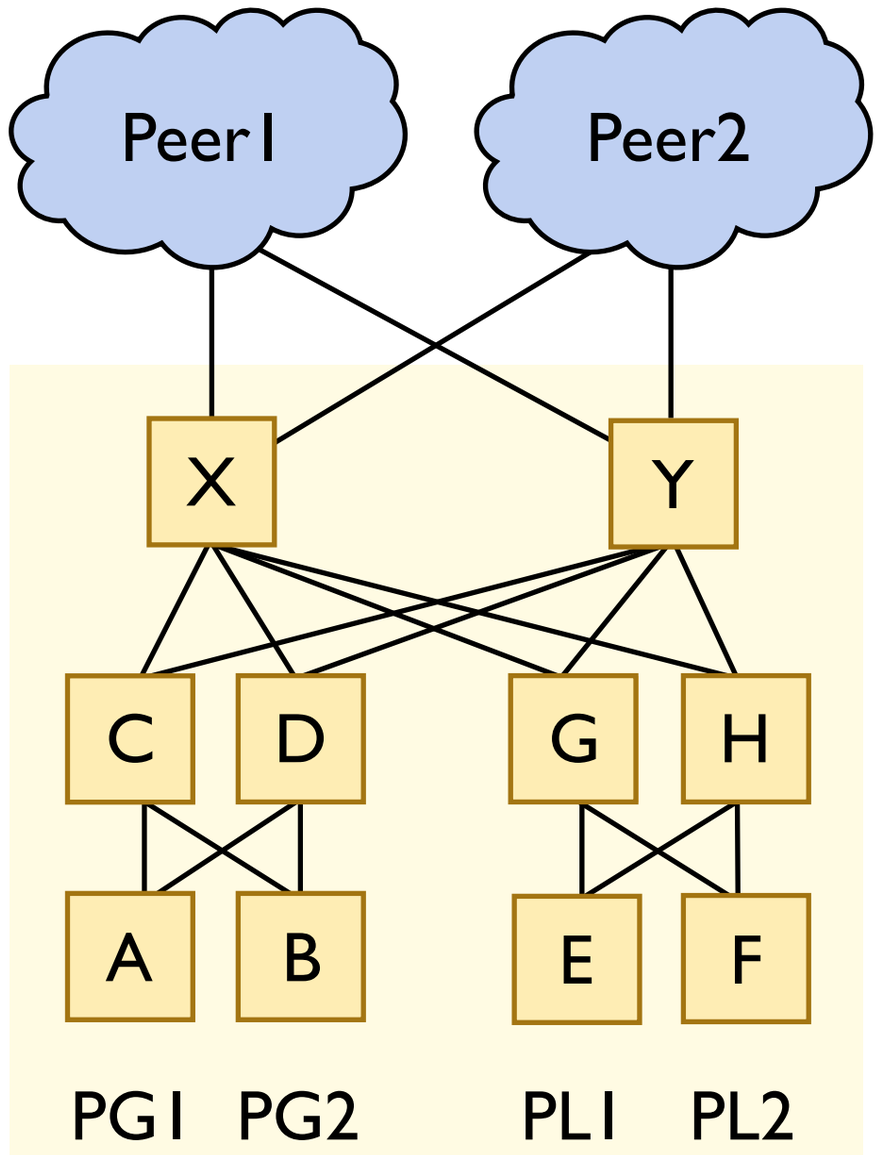
Global Services

Local Services

Example 2: A Data Center Network

Prevent transit traffic

```
...

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



Global Services

Local Services

Propane Compiler

dc.pro — dc (git: master)

```
1  define PL1 = 2.0.1.0/24
2  define PL2 = 2.0.2.0/24
3  define PG1 = 1.0.0.0/24
4  define PG2 = 1.0.1.0/24
5  define PG  = 1.0.0.0/16
6
7  define Peer = {Peer1, Peer2}
8
9  define routing = {
10     PG1 => end(A),
11     PG2 => end(B),
12     PL1 => end(E),
13     PL2 => end(F),
14     true => exit(Peer1) >> exit(Peer2)
15 }
16
17 define locality = {
18     PL1 or PL2 => always(in)
19 }
20
21 define transit(X,Y) = enter(X) and exit(Y)
22
23 define notransit = {
24     true => not transit(Peer, Peer)
25 }
26
27 define main = routing and locality and notransit
28
29 control {
30     aggregate(PG, in -> out)
31 }
```

Line: 31:2 | Propane | Tab Size: 4

dc — -bash — 82×21

```
beckett:dc ryanbeckett$ propane -pol:dc.pro -topo:dc.xml -out:config -parallel:on
--------------------------------------------------------------------------------
dc/dc.pro

Warning: Could only prove aggregation black-hole safety for up to 1 failures.
         It may be possible to disconnect the prefix 1.0.1.0/24 ge 24 le 32 at
         location B from the aggregate prefix 1.0.0.0/16 ge 16 le 32 at X after
         2 failures. Consider using the -failures:n flag to specify a tolerable
         failure level.
--------------------------------------------------------------------------------
beckett:dc ryanbeckett$
```

Propane Compiler

config.ir — config (git: master)

```
28
29  Router G
30    allow:   0.0.0.0/0 ge 0 le 32, peer=*, comm=1 [export peer<-*]
31    allow:   0.0.0.0/0 ge 0 le 32, peer=*, comm=2 [export peer<-*] (lp=99)
32
33  Router H
34    allow:   0.0.0.0/0 ge 0 le 32, peer=*, comm=1 [export peer<-*]
35    allow:   0.0.0.0/0 ge 0 le 32, peer=*, comm=2 [export peer<-*] (lp=99)
36
37  Router X
38    control: aggregate(1.0.0.0/16 ge 16 le 32, Peer2)
39    control: aggregate(1.0.0.0/16 ge 16 le 32, Peer1)
40    allow:   1.0.0.0/24 ge 24 le 32, peer=in [export peer<-*]
41    deny:    1.0.0.0/24 ge 24 le 32
42    allow:   1.0.1.0/24 ge 24 le 32, peer=in [export peer<-*]
43    deny:    1.0.1.0/24 ge 24 le 32
44    allow:   2.0.1.0/24 ge 24 le 32, peer=in [export peer<-in]
45    deny:    2.0.1.0/24 ge 24 le 32
46    allow:   2.0.2.0/24 ge 24 le 32, peer=in [export peer<-in]
47    deny:    2.0.2.0/24 ge 24 le 32
48    allow:   0.0.0.0/0 ge 0 le 32, peer=Peer1 [export peer<-in, comm<-1]
49    allow:   0.0.0.0/0 ge 0 le 32, peer=in, comm=1 [export peer<-in]
50    allow:   0.0.0.0/0 ge 0 le 32, peer=Peer2 [export peer<-in, comm<-2] (lp=99)
51    allow:   0.0.0.0/0 ge 0 le 32, peer=in, comm=2 [export peer<-in] (lp=99)
52
```

Line: 47:34 | Plain Text | Tab Size: 4

Propane Compiler

config.ir — config (git: master)

```
28
29  Router G
30    allow:   0.0.0.0/0 ge 0 le 32, peer=
31    allow:   0.0.0.0/0 ge 0 le 32, peer=
32
33  Router H
34    allow:   0.0.0.0/0 ge 0 le 32, peer=
35    allow:   0.0.0.0/0 ge 0 le 32, peer=
36
37  Router X
38    control: aggregate(1.0.0.0/16 ge 16
39    control: aggregate(1.0.0.0/16 ge 16
40    allow:   1.0.0.0/24 ge 24 le 32, pee
41    deny:    1.0.0.0/24 ge 24 le 32
42    allow:   1.0.1.0/24 ge 24 le 32, pee
43    deny:    1.0.1.0/24 ge 24 le 32
44    allow:   2.0.1.0/24 ge 24 le 32, pee
45    deny:    2.0.1.0/24 ge 24 le 32
46    allow:   2.0.2.0/24 ge 24 le 32, pee
47    deny:    2.0.2.0/24 ge 24 le 32
48    allow:   0.0.0.0/0 ge 0 le 32, peer=
49    allow:   0.0.0.0/0 ge 0 le 32, peer=
50    allow:   0.0.0.0/0 ge 0 le 32, peer=
51    allow:   0.0.0.0/0 ge 0 le 32, peer=
52
```

Line: 47:34 | Plain Text | Tab Size: 4

as109.quagga — quagga (git: master)

as109.quagga

```
19    neighbor 12.12.12.1 route-map rm-export-1 out
20  !
21  ip prefix-list pl-1 permit 1.0.0.0/24 ge 24 le 32
22  ip prefix-list pl-2 deny 1.0.0.0/24 ge 24 le 32
23  ip prefix-list pl-3 permit 1.0.1.0/24 ge 24 le 32
24  ip prefix-list pl-4 deny 1.0.1.0/24 ge 24 le 32
25  ip prefix-list pl-5 permit 2.0.1.0/24 ge 24 le 32
26  ip prefix-list pl-6 deny 2.0.1.0/24 ge 24 le 32
27  ip prefix-list pl-7 permit 2.0.2.0/24 ge 24 le 32
28  ip prefix-list pl-8 deny 2.0.2.0/24 ge 24 le 32
29  ip prefix-list pl-9 permit 0.0.0.0/0 ge 0 le 32
30  !
31  ip community-list standard cl-1 permit 100:1
32  ip community-list standard cl-2 permit 100:2
33  ip community-list standard cl-3 permit 100:3
34  ip community-list standard cl-4 permit 100:4
35  !
36  ip as-path access-list path-1 permit (^107_ | ^106_ | ^103_ | ^102_)
37  ip as-path access-list path-2 permit (^111_ | ^110_)
38  ip as-path access-list path-3 permit ^110_
39  ip as-path access-list path-4 permit ^111_
40  !
41  route-map rm-in permit 10
42    match ip address prefix-list pl-1
43    match as-path path-1
44    set community additive 200:1
45  !
46  route-map rm-in permit 20
47    match ip address prefix-list pl-2
48  !
49  route-map rm-in permit 30
50    match ip address prefix-list pl-3
51    match as-path path-1
52    set community additive 200:1
53  !
```

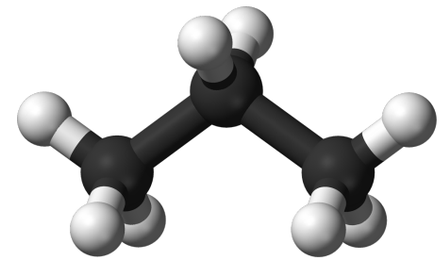
Line: 58:36 | Plain Text | Tab Size: 4

Propane Compiler

ryanbeckett — -bash — 86×24

```
beckett:~ ryanbeckett$ propane

Usage: propane [options]
-pol                    Policy file
-topo                   Topology file
-out                    Output file
-failures:any|n         Failure safety for aggregation (default any)
-anycast:on|off         Allow anycast (default off)
-med:on|off             Use MED attribute (default off)
-prepending:on|off      Use AS path prepending (default off)
-no-export:on|off       Use no-export community (default off)
-ibgp:on|off            Assume IBGP configuration (default off)
-minimize:on|off        Minimize configuration (default on)
-parallel:on|off        Parallelize compilation (default on)
-target:none|ir|templ   Compilation target
-stats:on|off           Display performance statistics to stdout (default off)
-checkenter:on|off      Check traffic entry conditions to network
-debug:on:off           Log/Save Debugging information (default off)
-debug-dir              Debugging directory (default 'debug')
-test                   Run unit tests
-bench                  Generate benchmark files
-help                   Display this message

beckett:~ ryanbeckett$
```

Propane: Summary



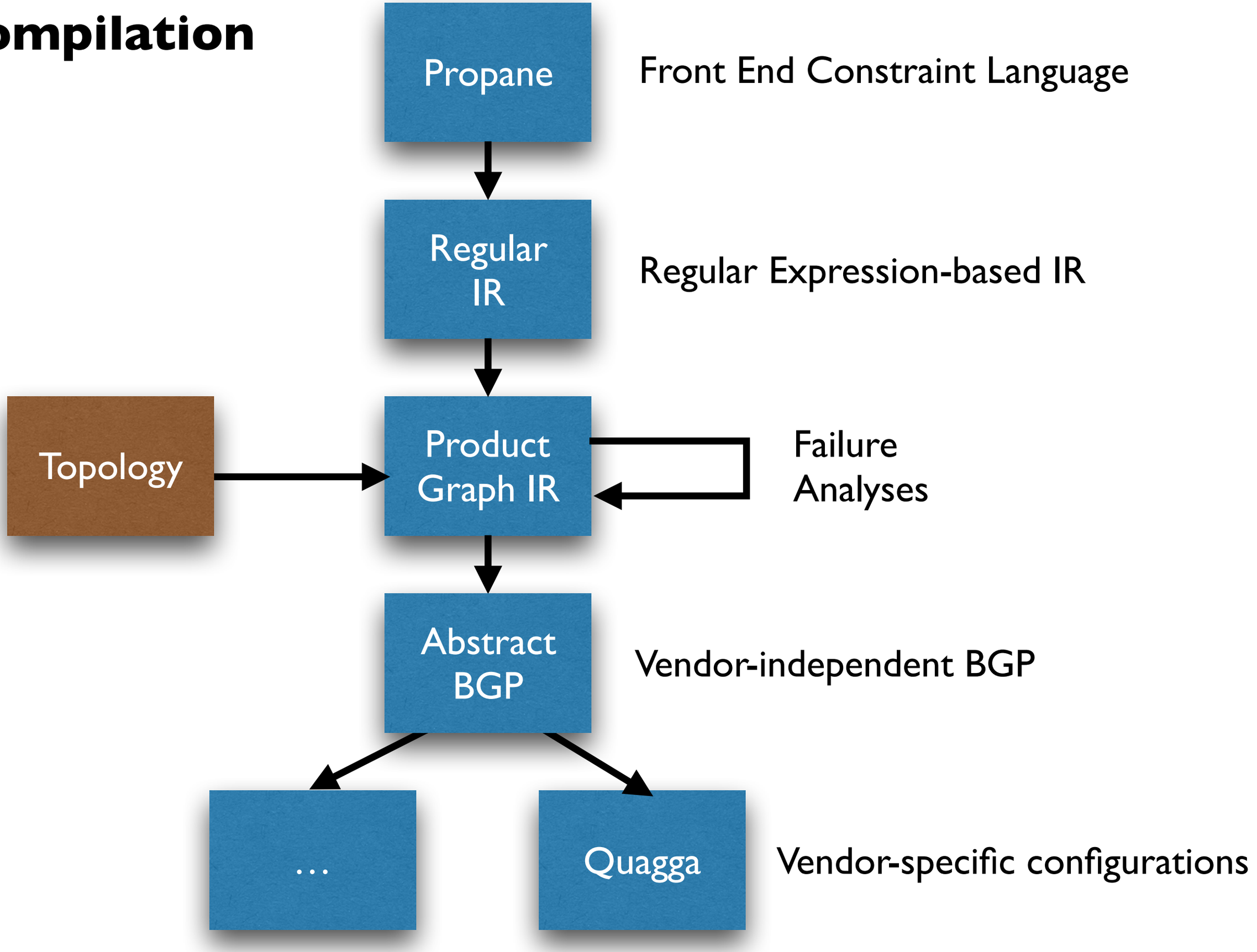
High-level language

- Single, network-wide policy
- Program using constraints and preferences
- Create new, reusable abstractions (e.g., transit traffic)
- Configure the network modularly

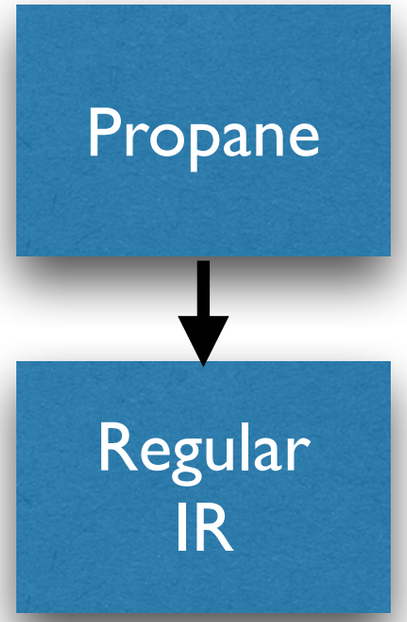
Compiler

- Automatically generates filters, preferences, community values, etc
- Static analysis guarantees policy compliance under all failures
- Lower bound on failures for aggregation-induced black holes

Compilation

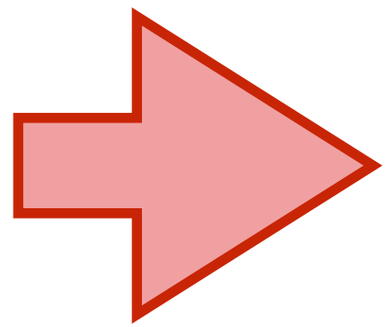


Propane Regular IR

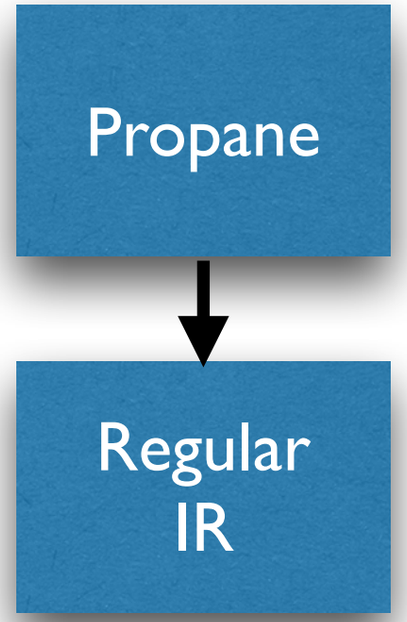


Step 1: Combine modular constraints

```
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)
```



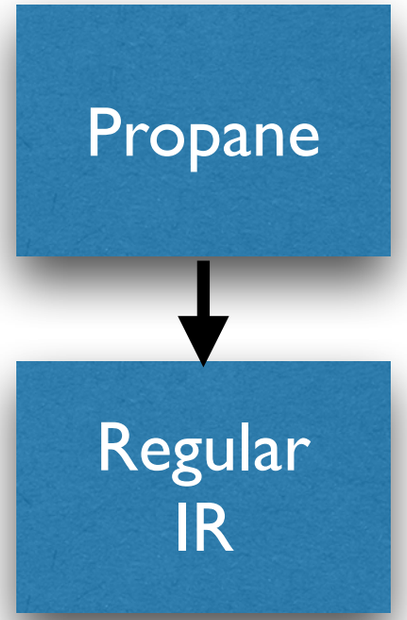
Propane Regular IR



Prefix-by-prefix intersection of constraints

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

Propane Regular IR



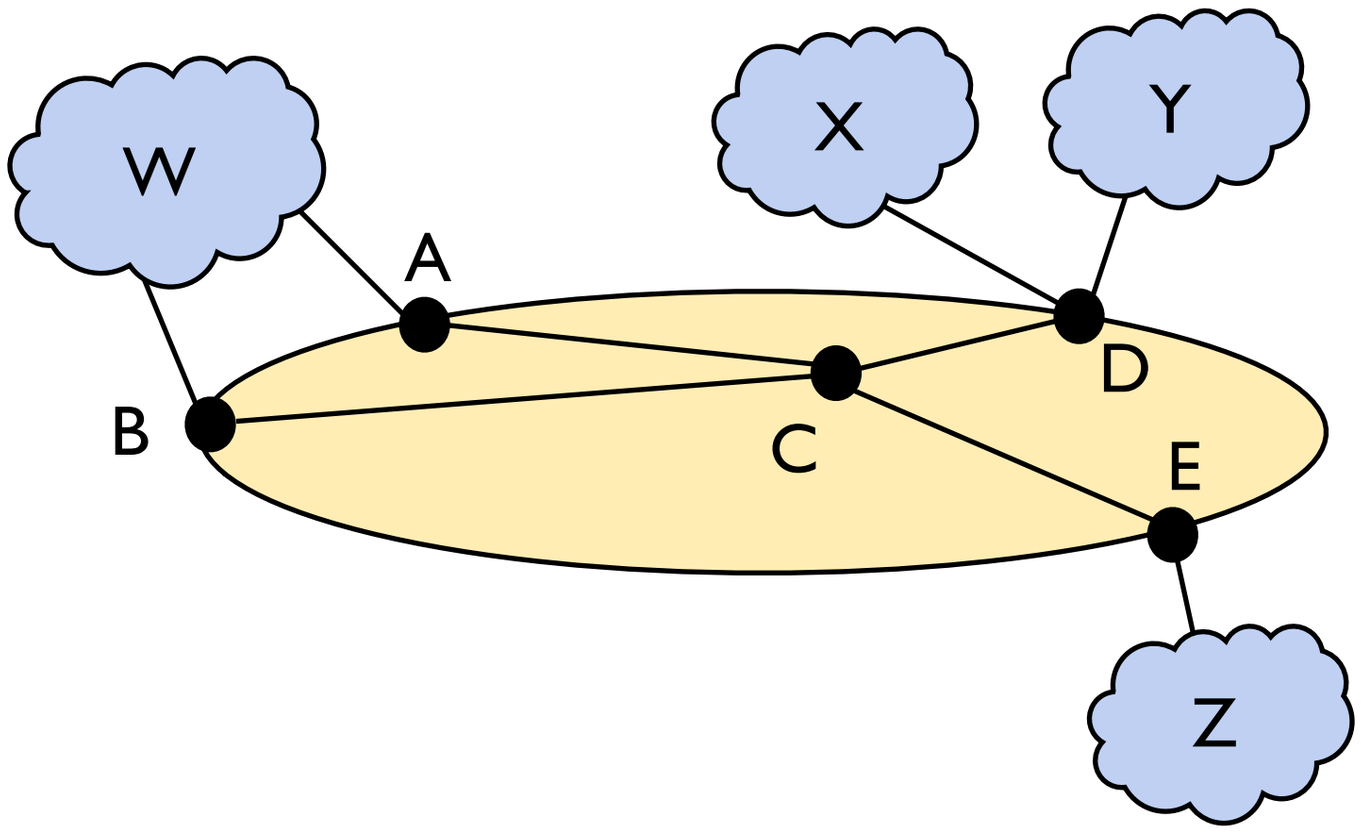
Step 2: Expand constraints in to regular expressions

```
      any = out*.in+.out*
      end(X) = any ∩ (Σ*.X)
   always(X) = any ∩ (X)*
     exit(X) = (out*.in*.(X ∩ in).out+) |
               (out*.in+.(X ∩ out).out*)
      end(X) = ...
    avoid(X) = ...
 waypoint(X) = ...
```

Step 3: Reduced syntax

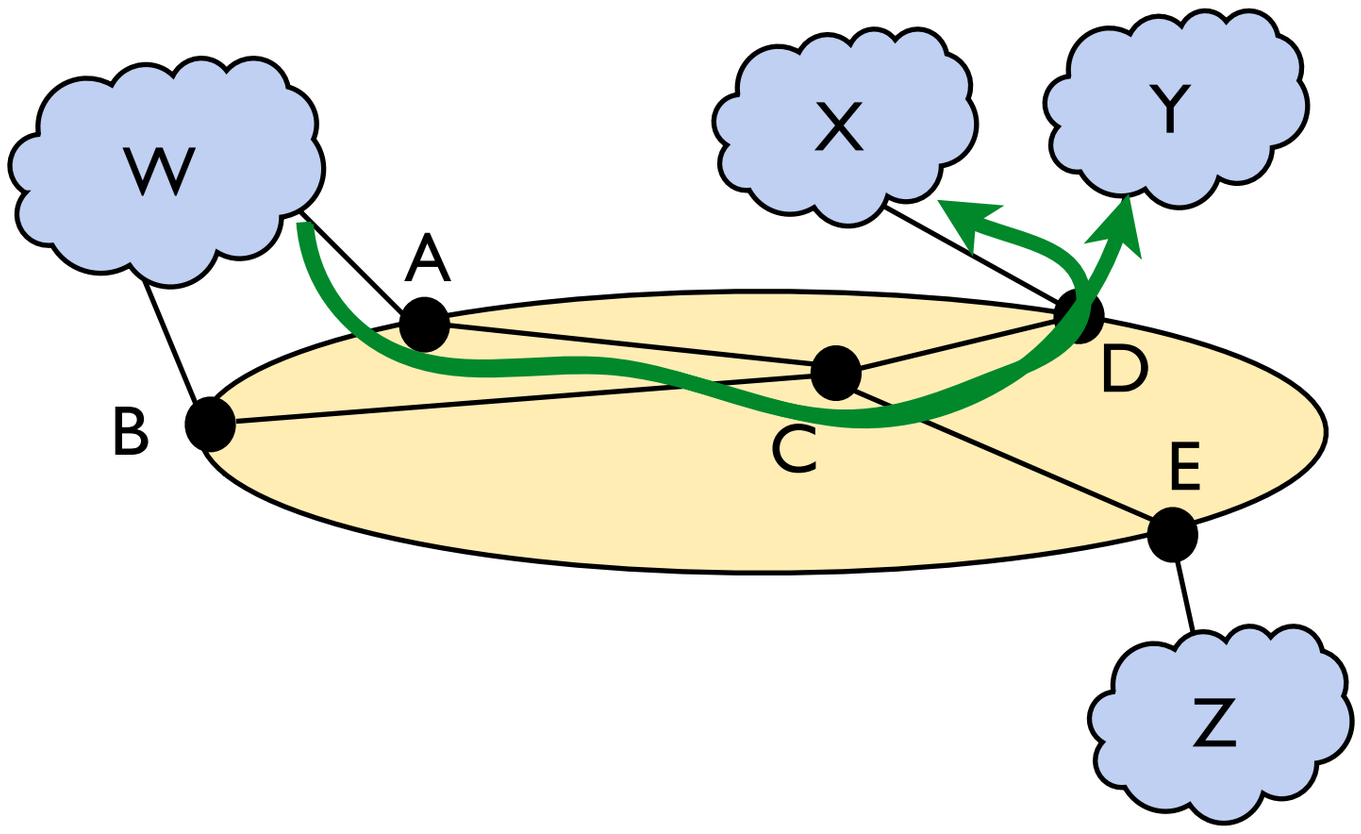
```
true => A.(X >> Y).out*
true => (A.X.out*) >> (A.Y.out*)
```

Compilation: An Idealized Example



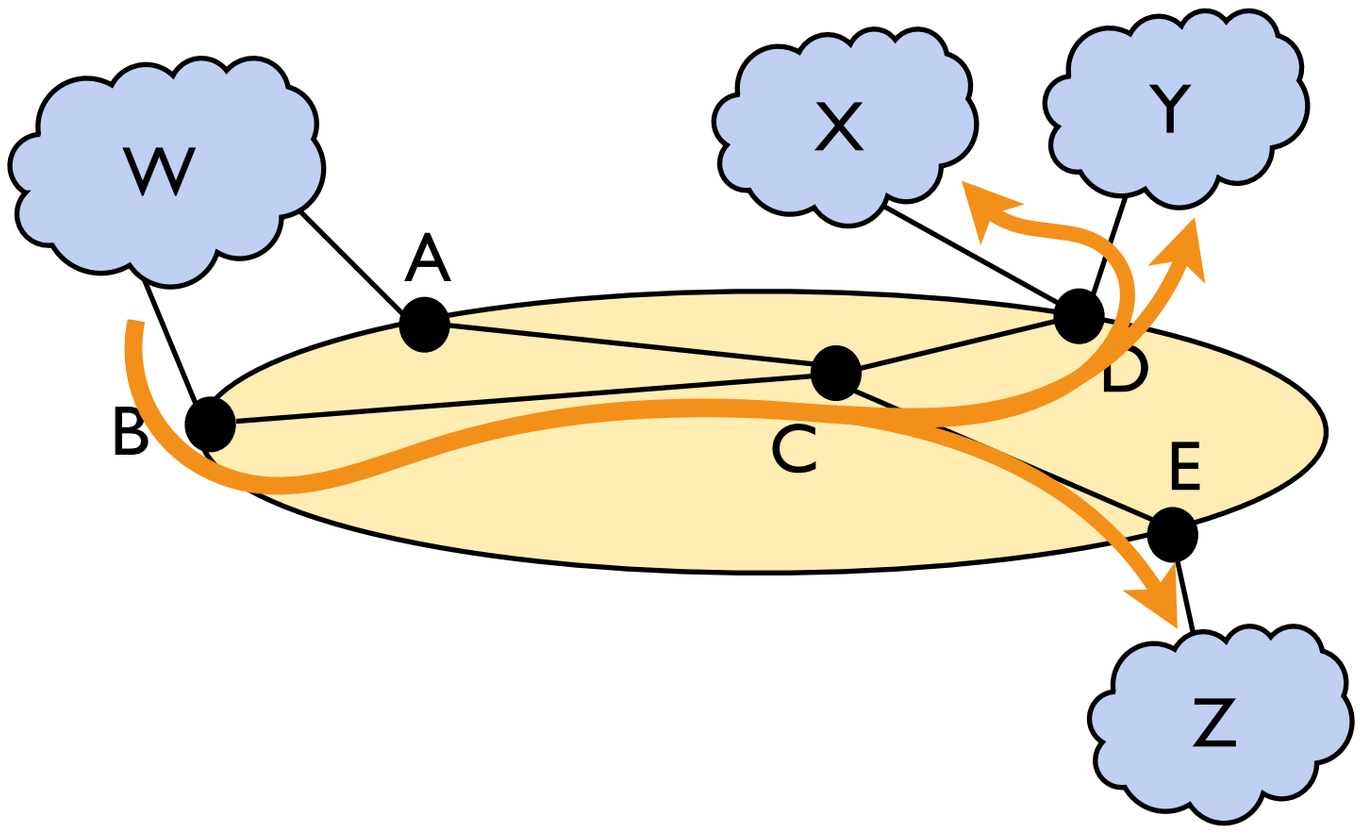
Policy: `(W.A.C.D.`**`out`**`) >> (W.B.`**`in`**`+.`**`out`**`)`

Compilation: An Idealized Example



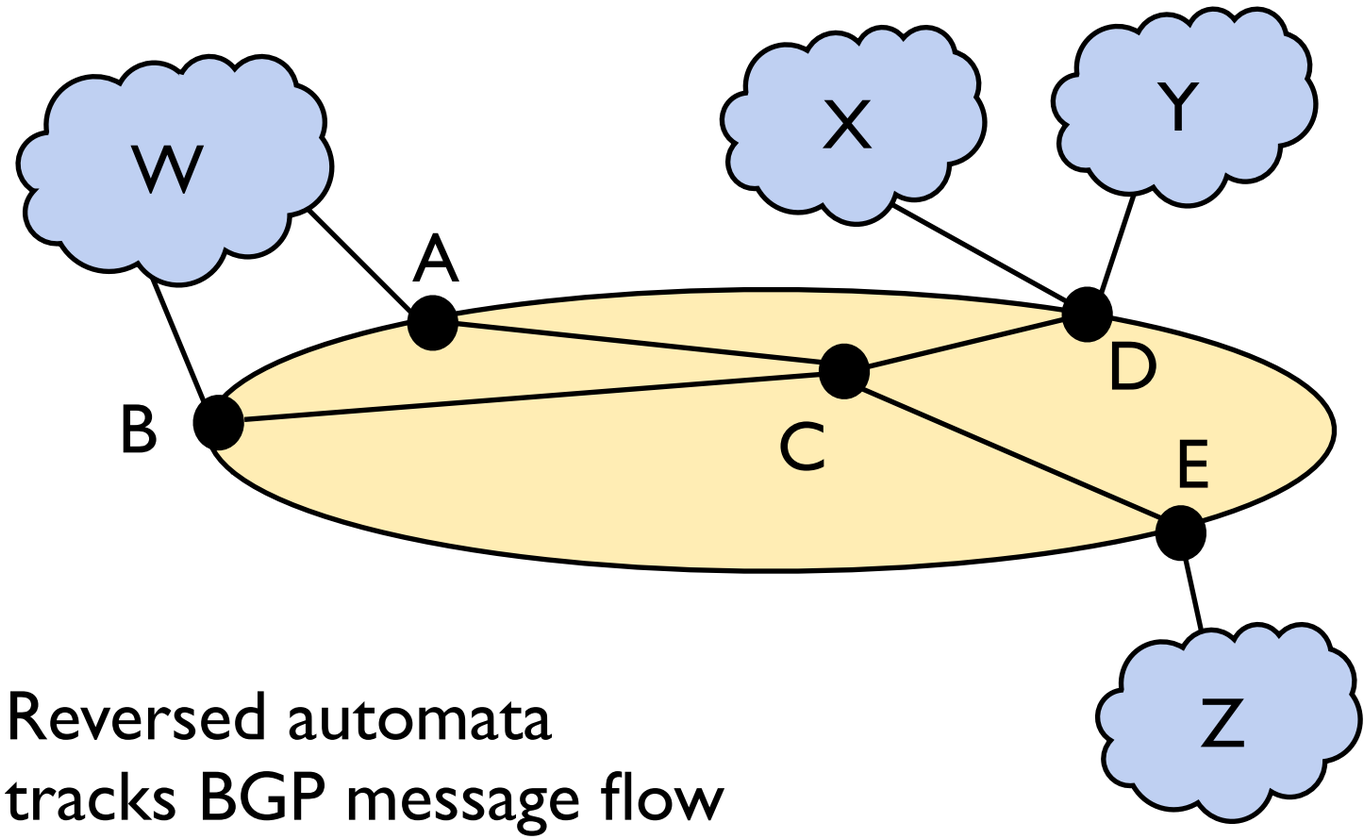
Policy: `(W.A.C.D.out) >> (W.B.in+.out)`

Compilation: An Idealized Example



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

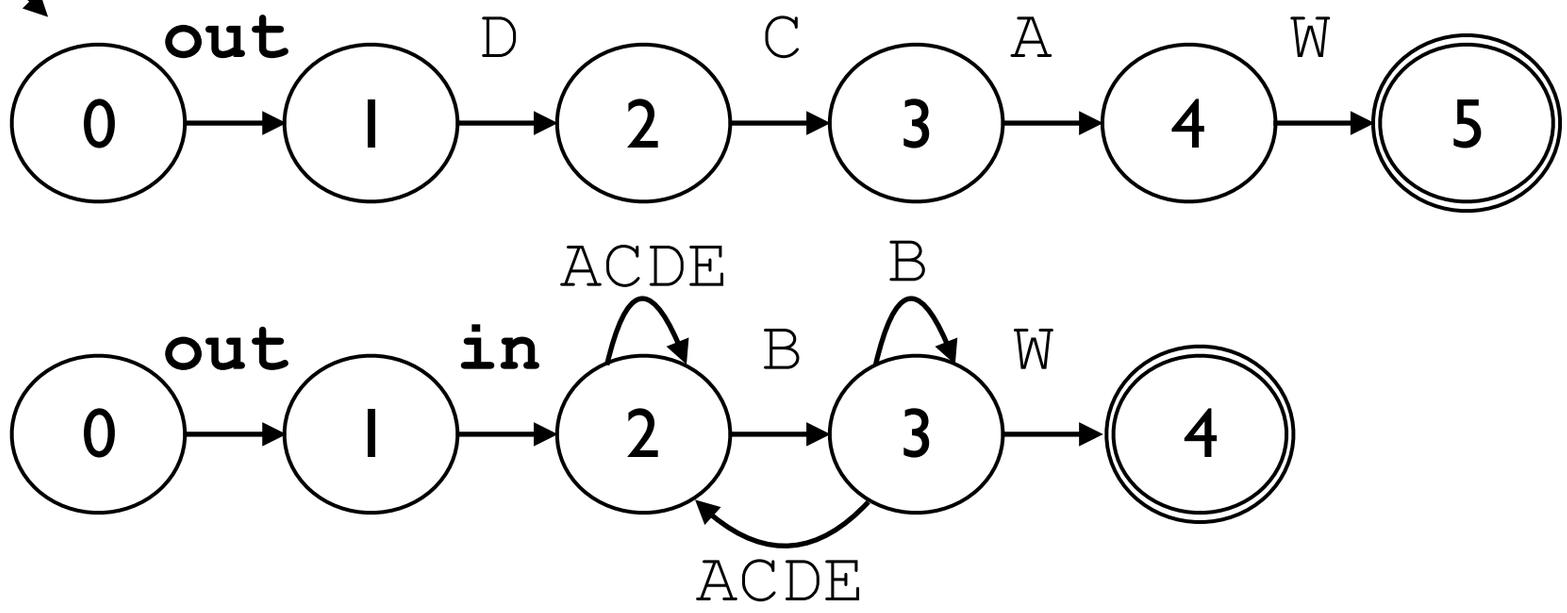
Reversed Automata from Policies



Reversed automata
tracks BGP message flow

Policy:

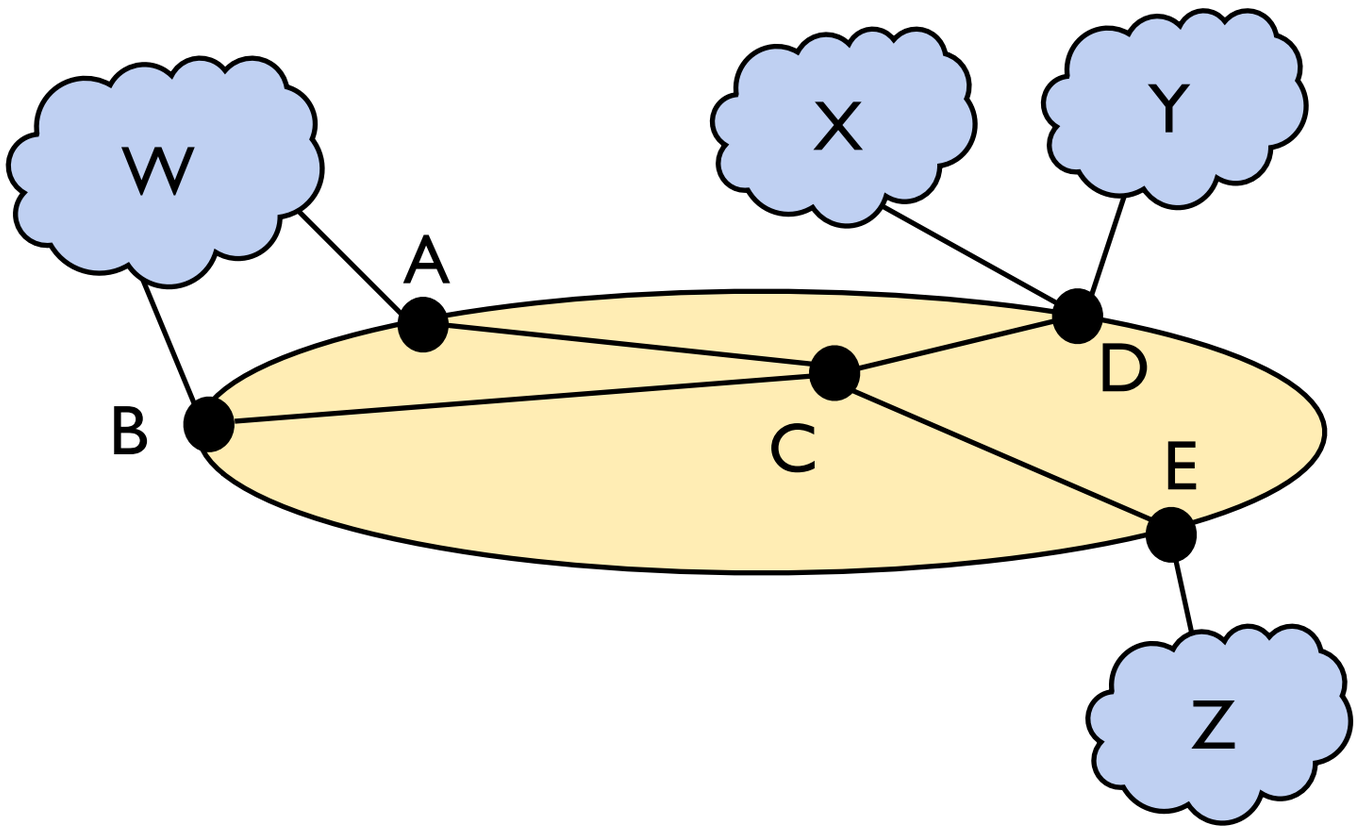
```
1.  (W.A.C.D.out)

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

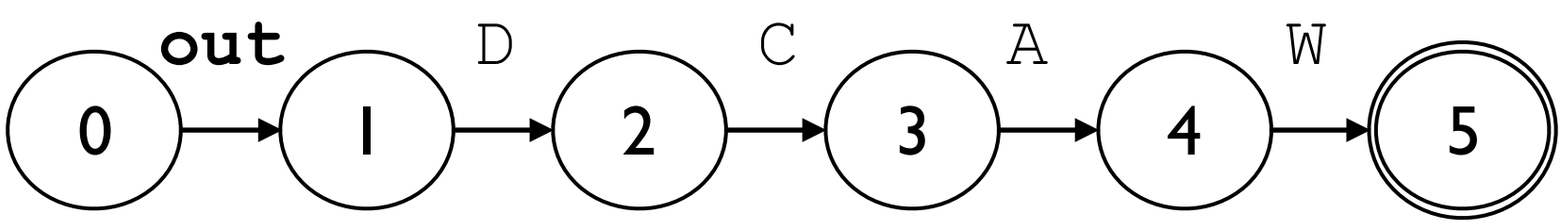


Constructing the Product Graph (PG)

start

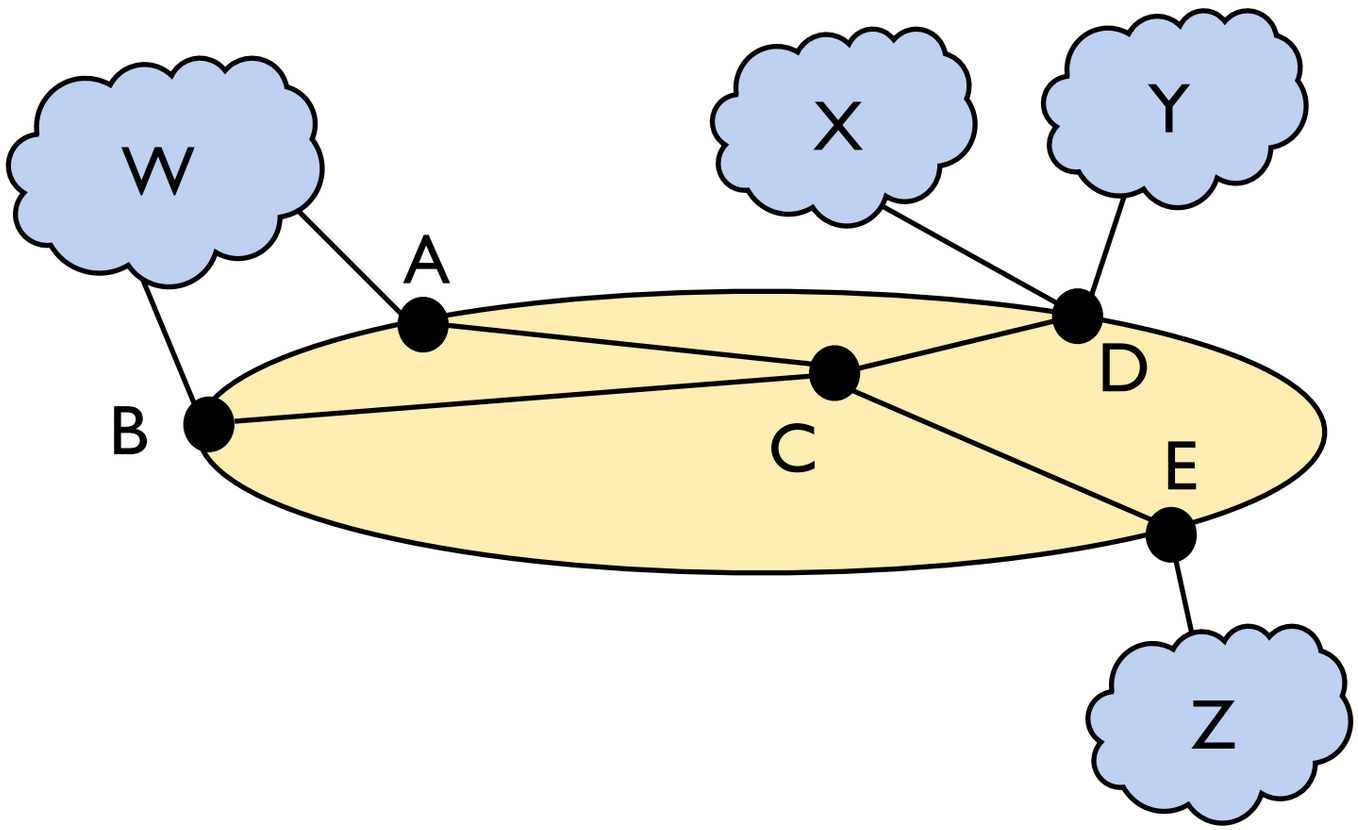
`(W.A.C.D.`**`out`**`) >> (W.B.`**`in`**`+.`**`out`**`)`

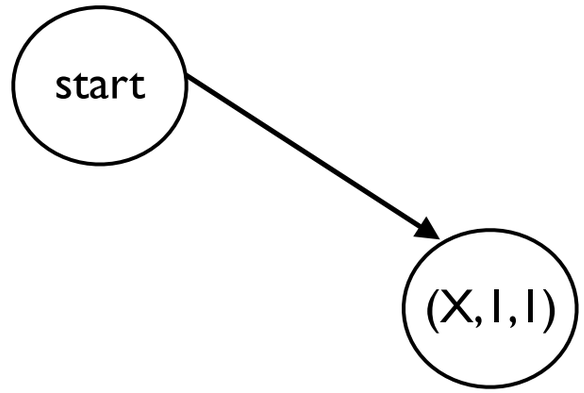


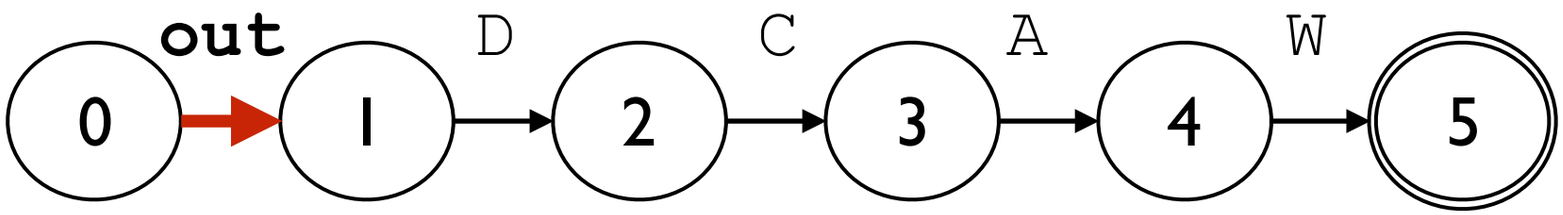


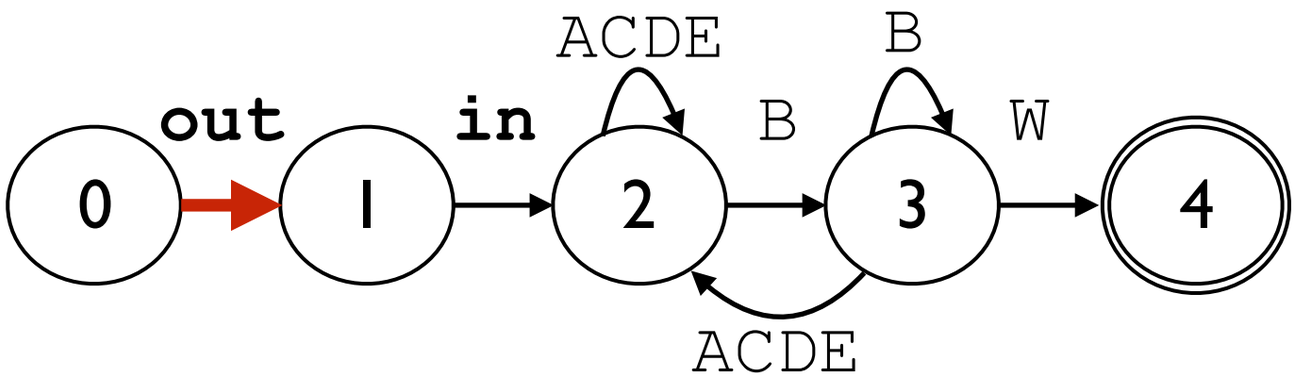
Constructing the Product Graph (PG)

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



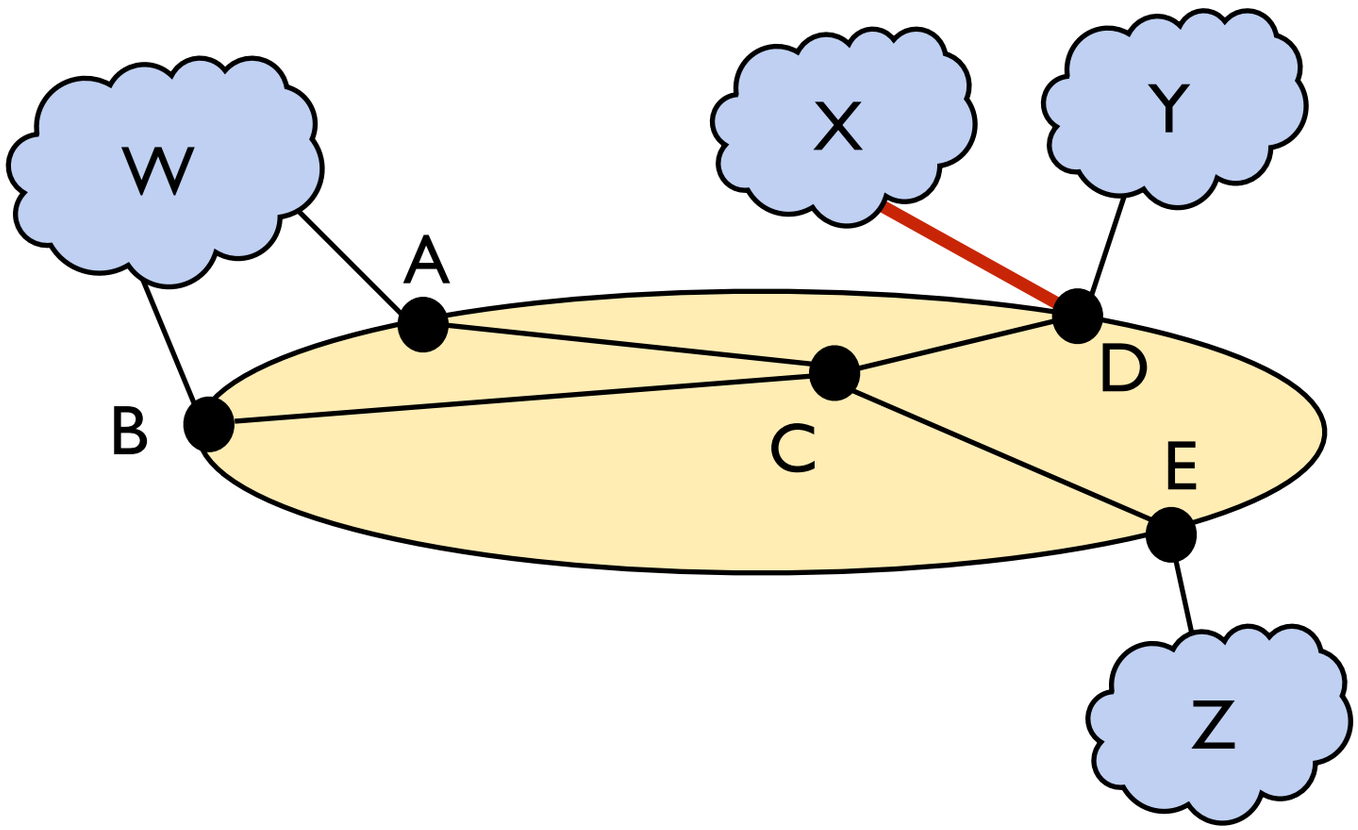


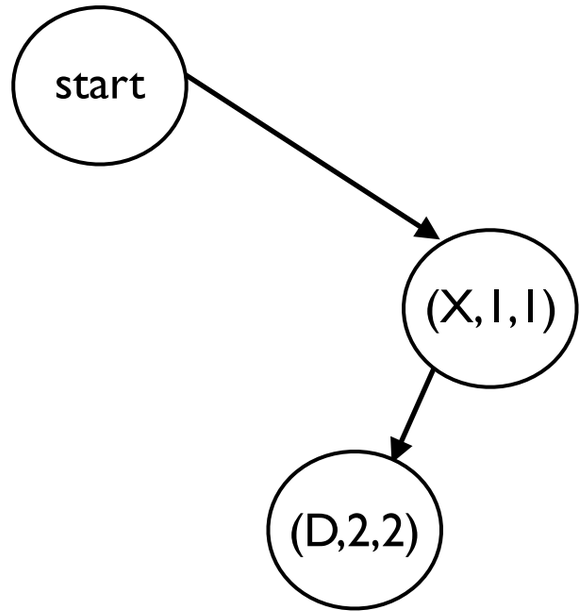


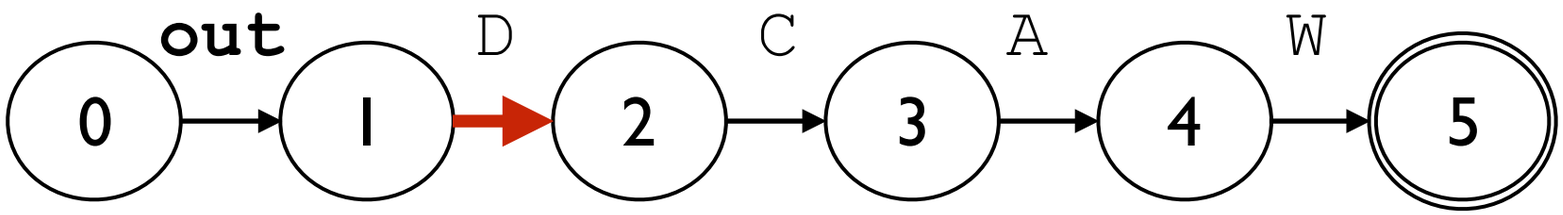


Constructing the Product Graph (PG)

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

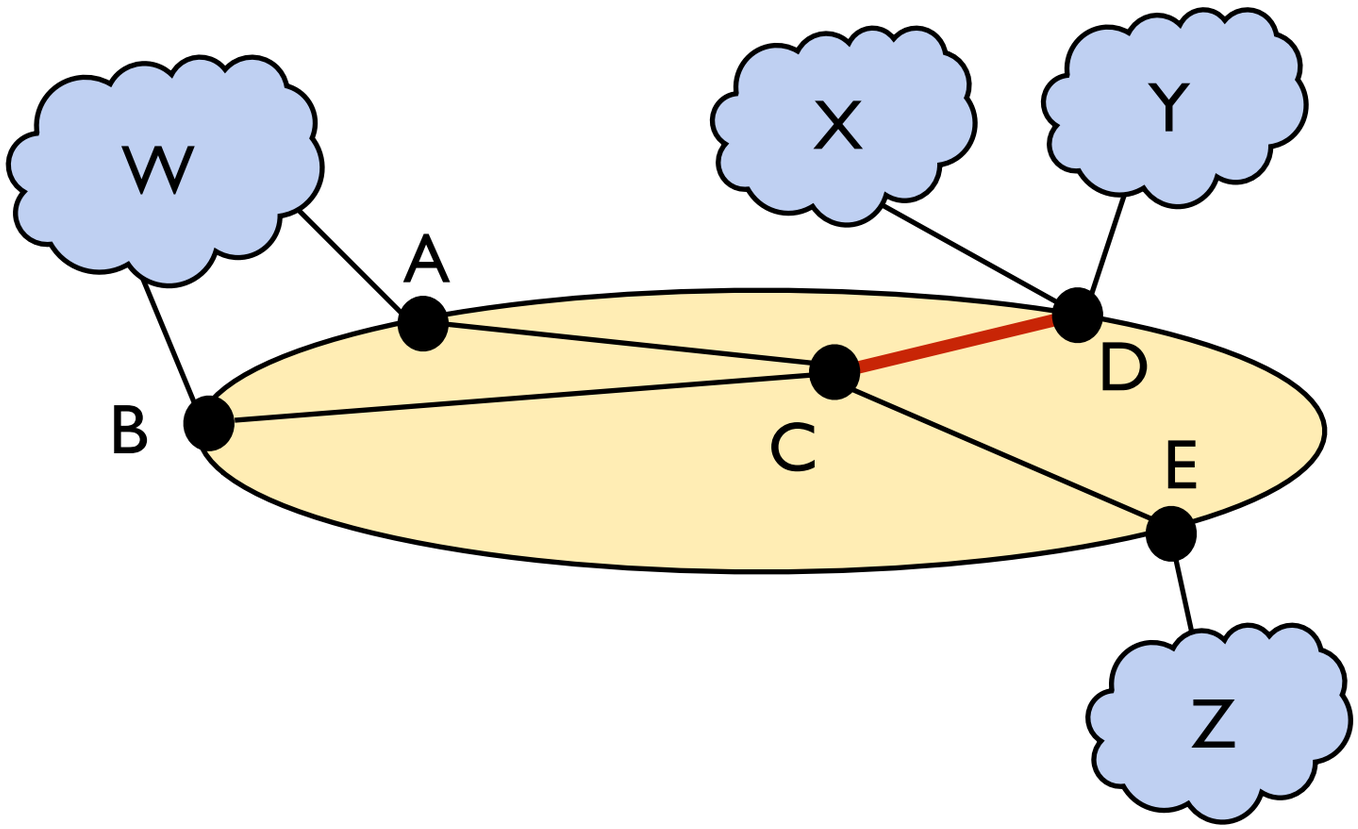


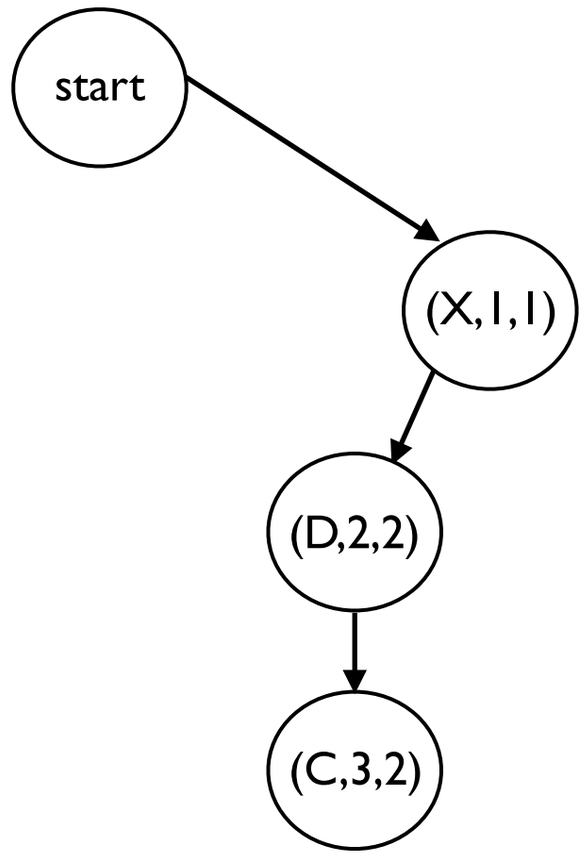


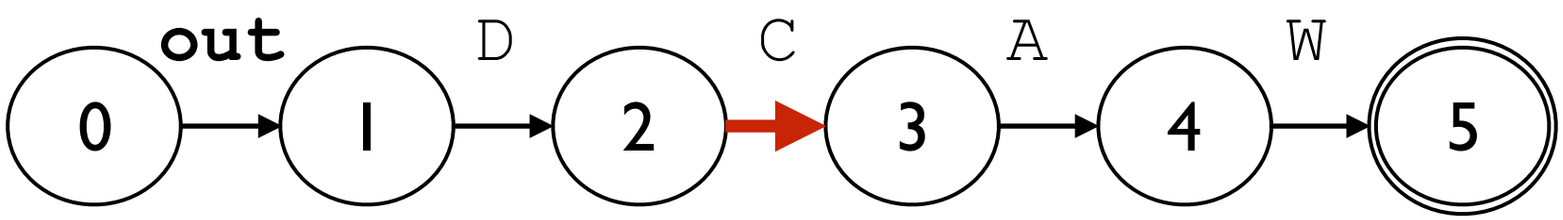


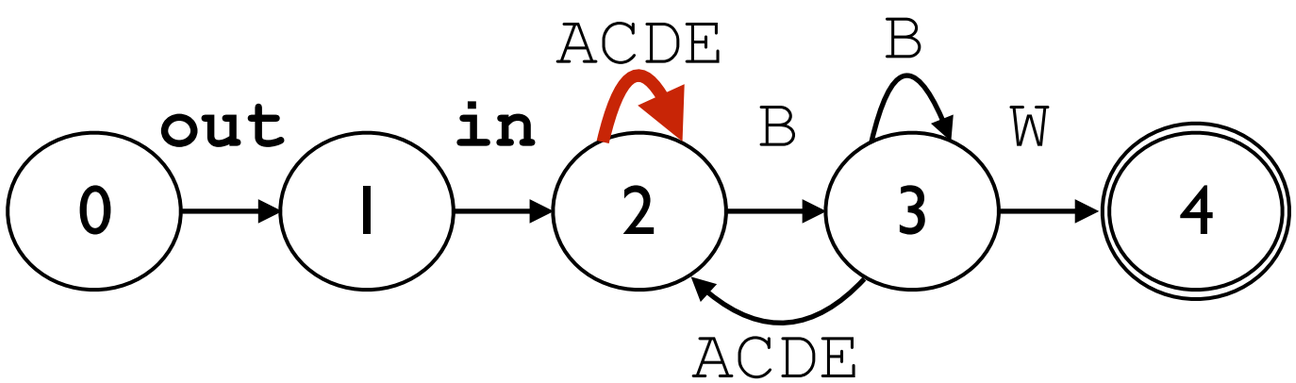
Constructing the Product Graph (PG)

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



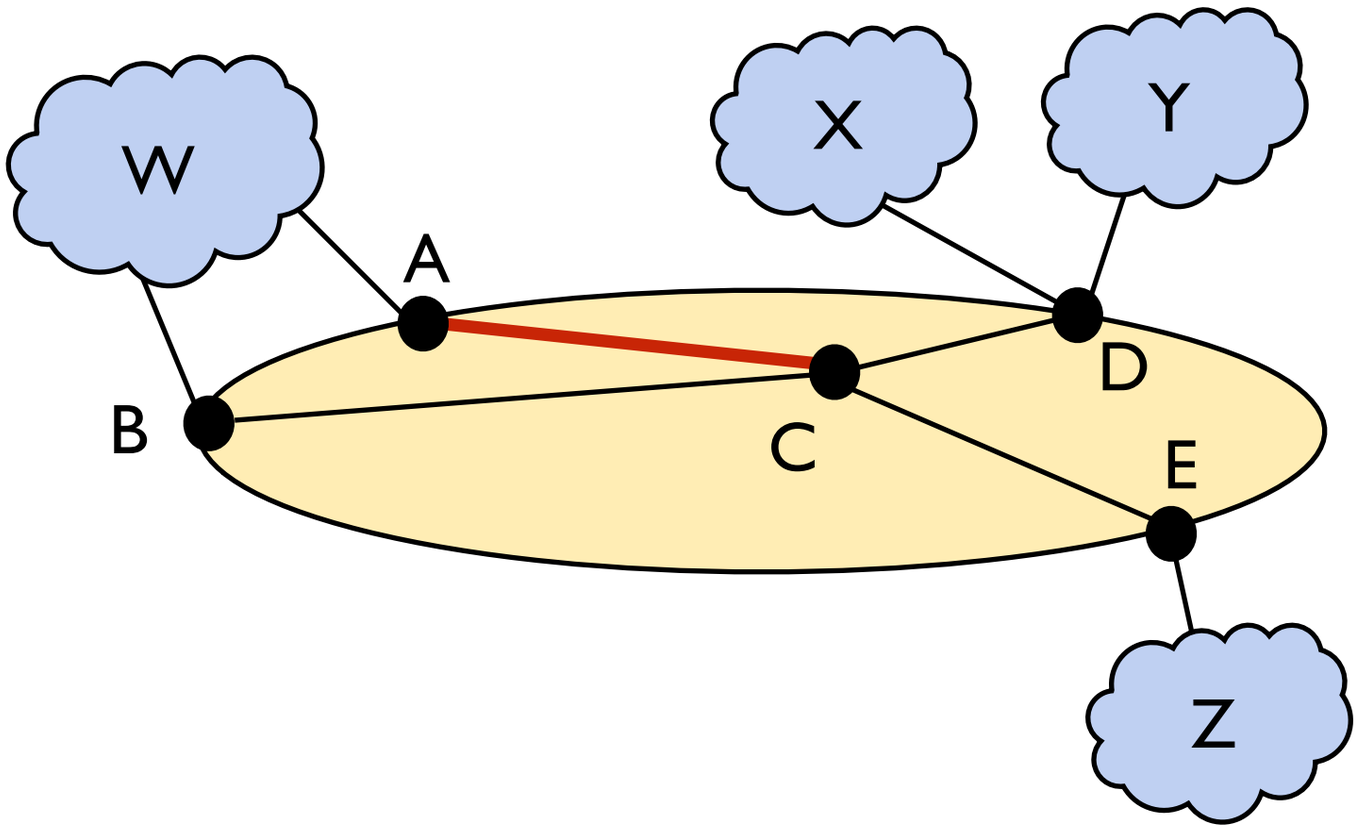


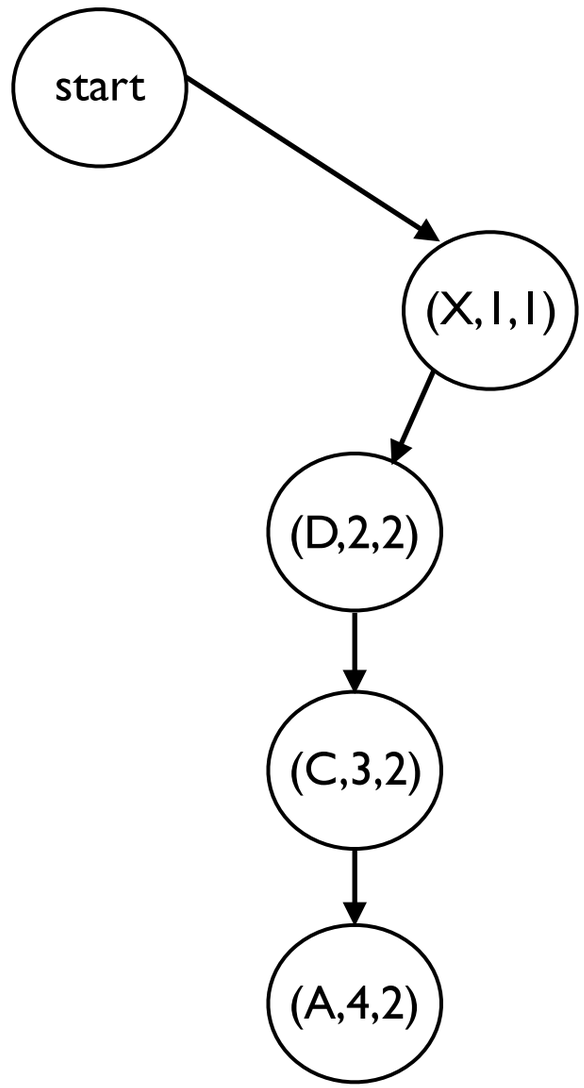


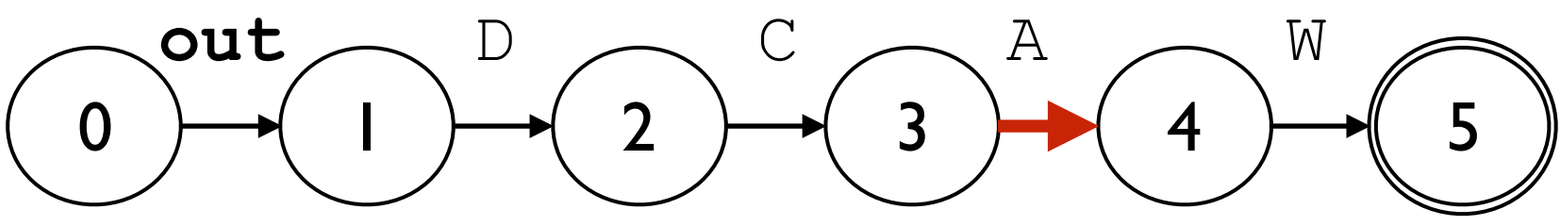


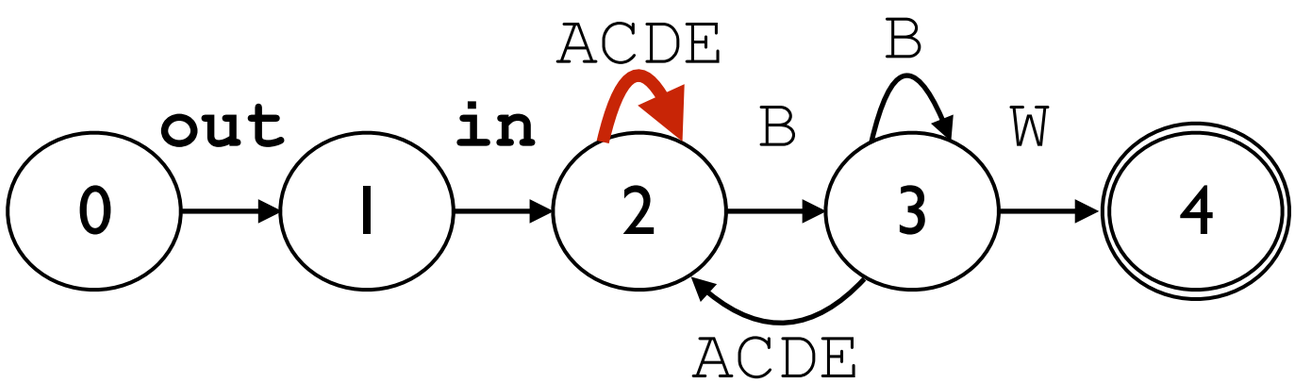
Constructing the Product Graph (PG)

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



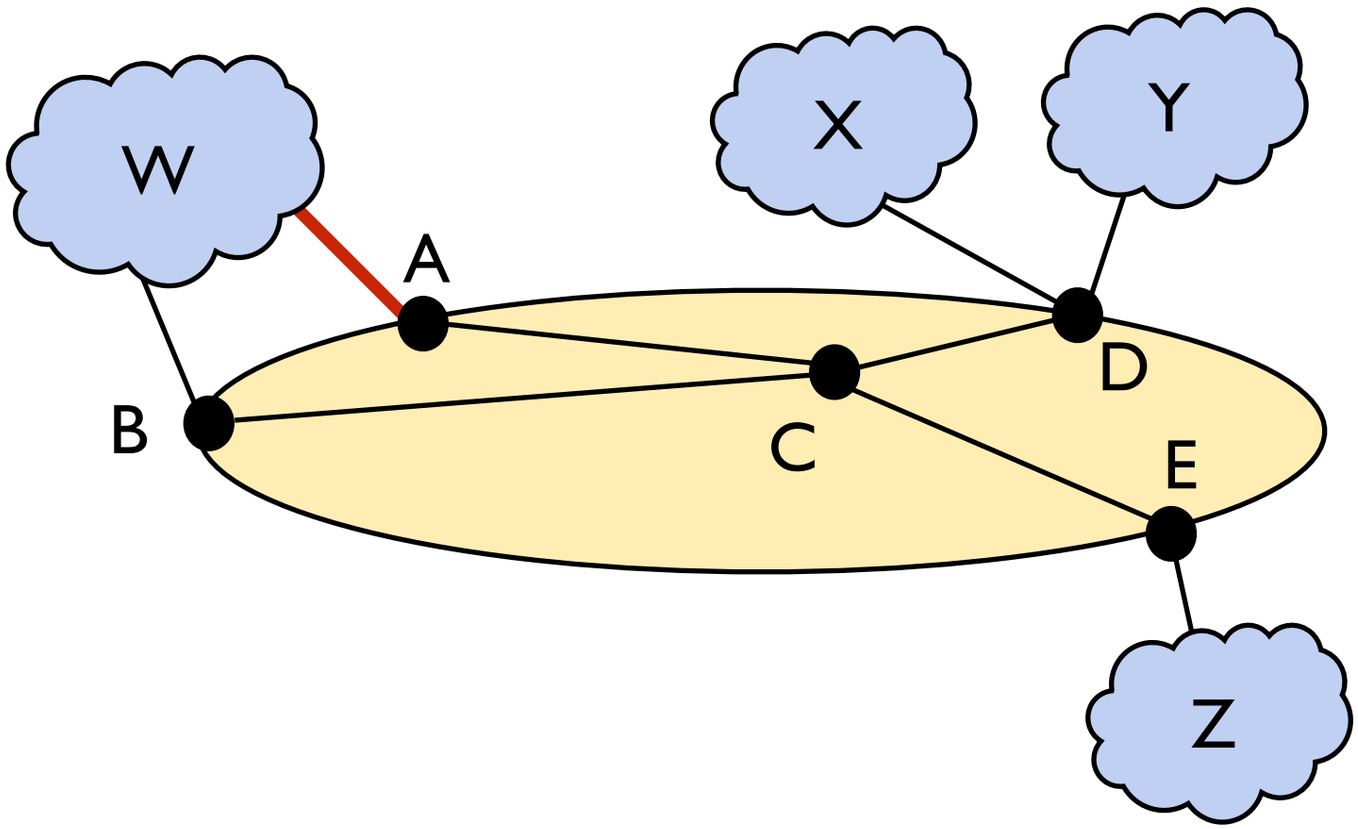


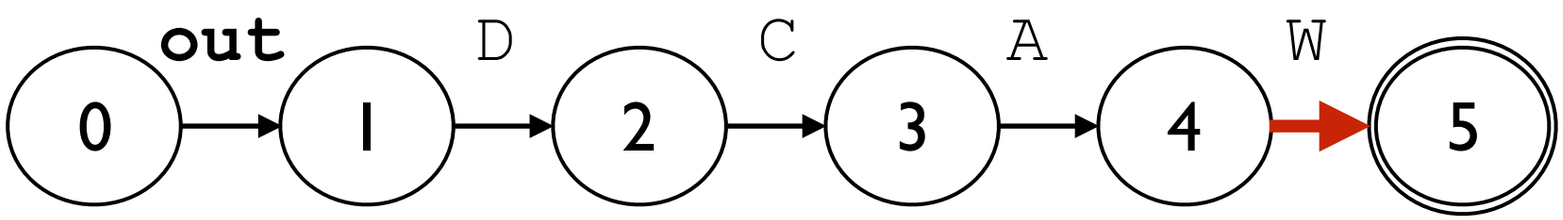


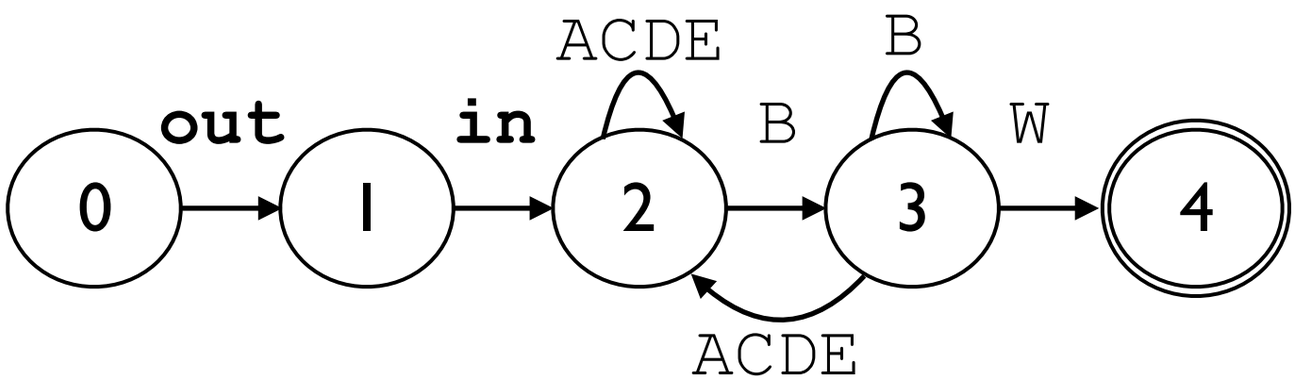


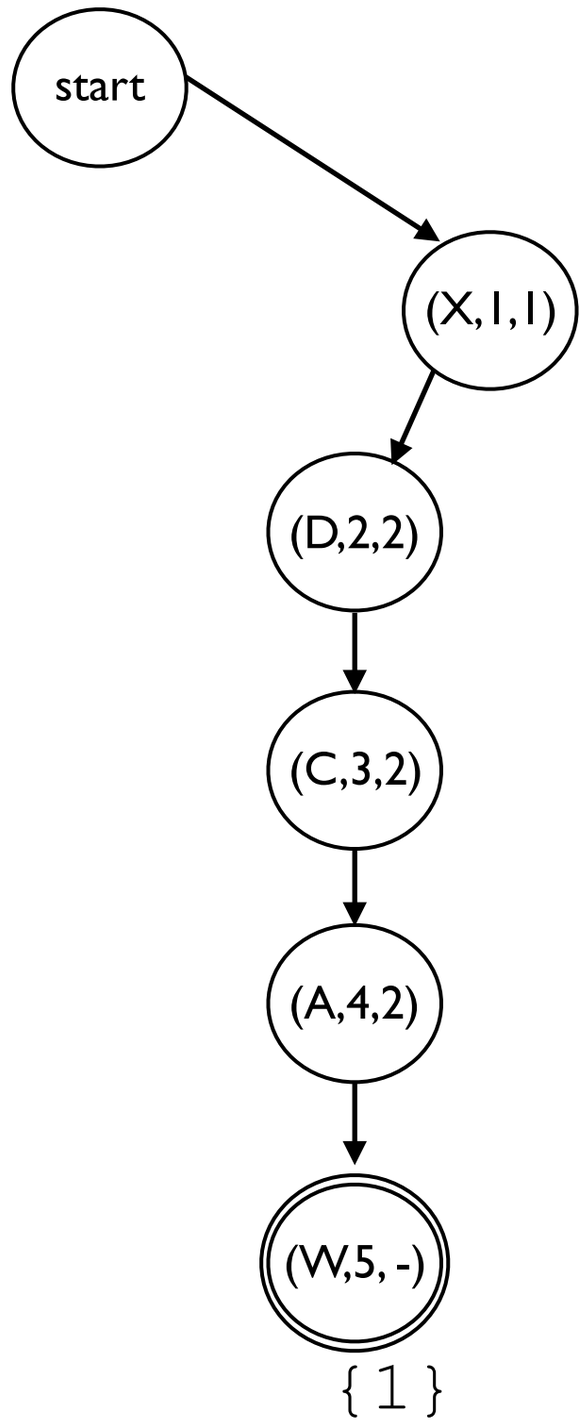
Constructing the Product Graph (PG)

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





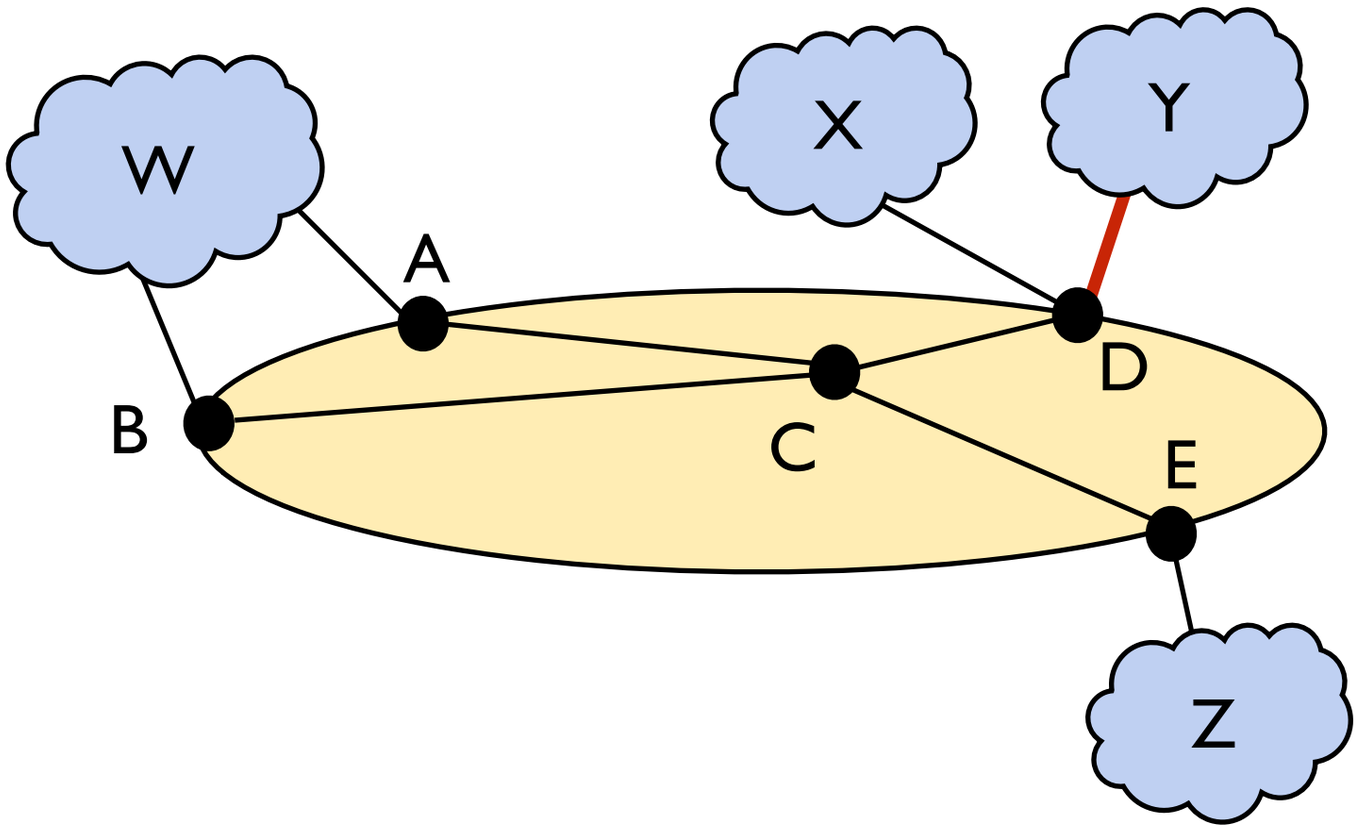


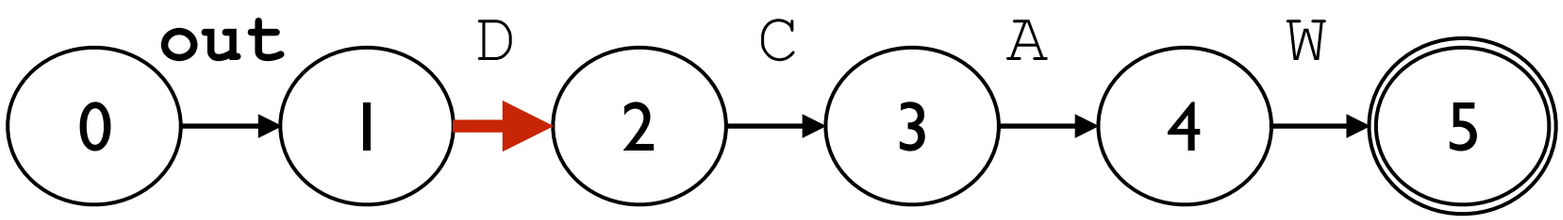


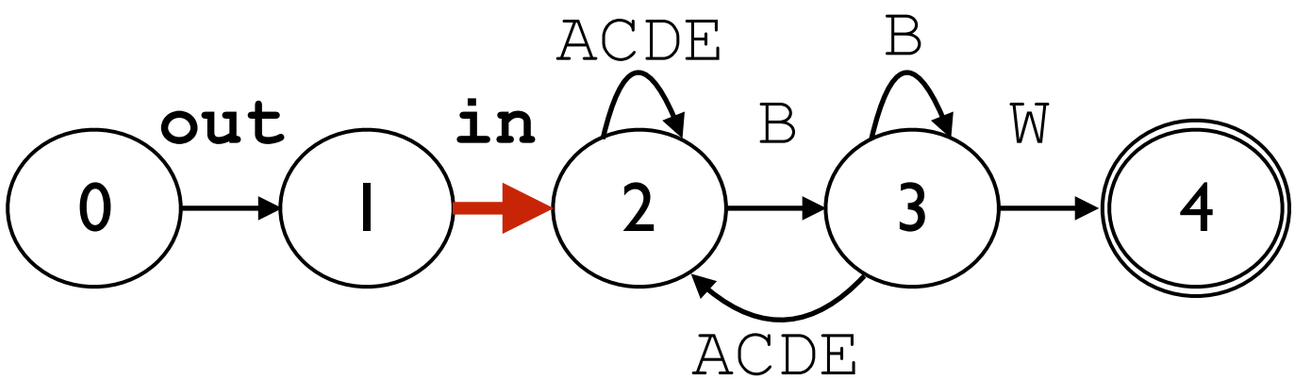
{1}

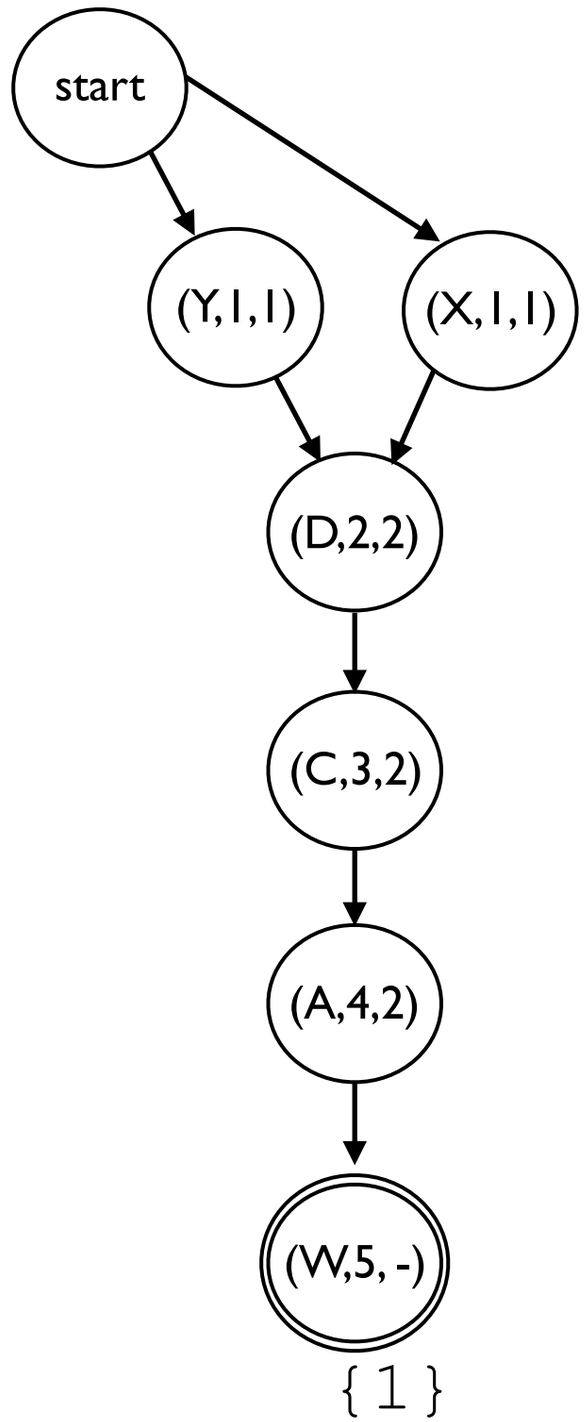
Constructing the Product Graph (PG)

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



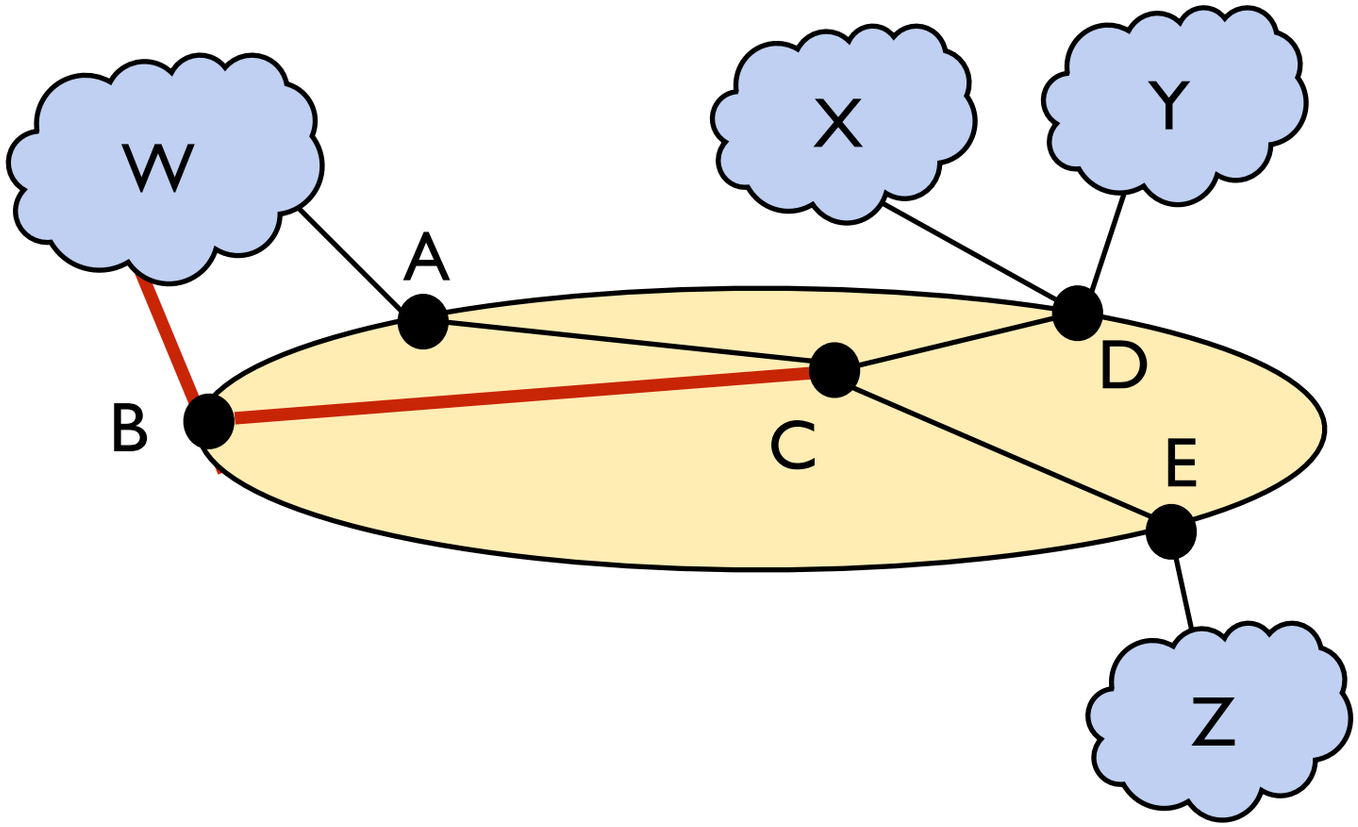


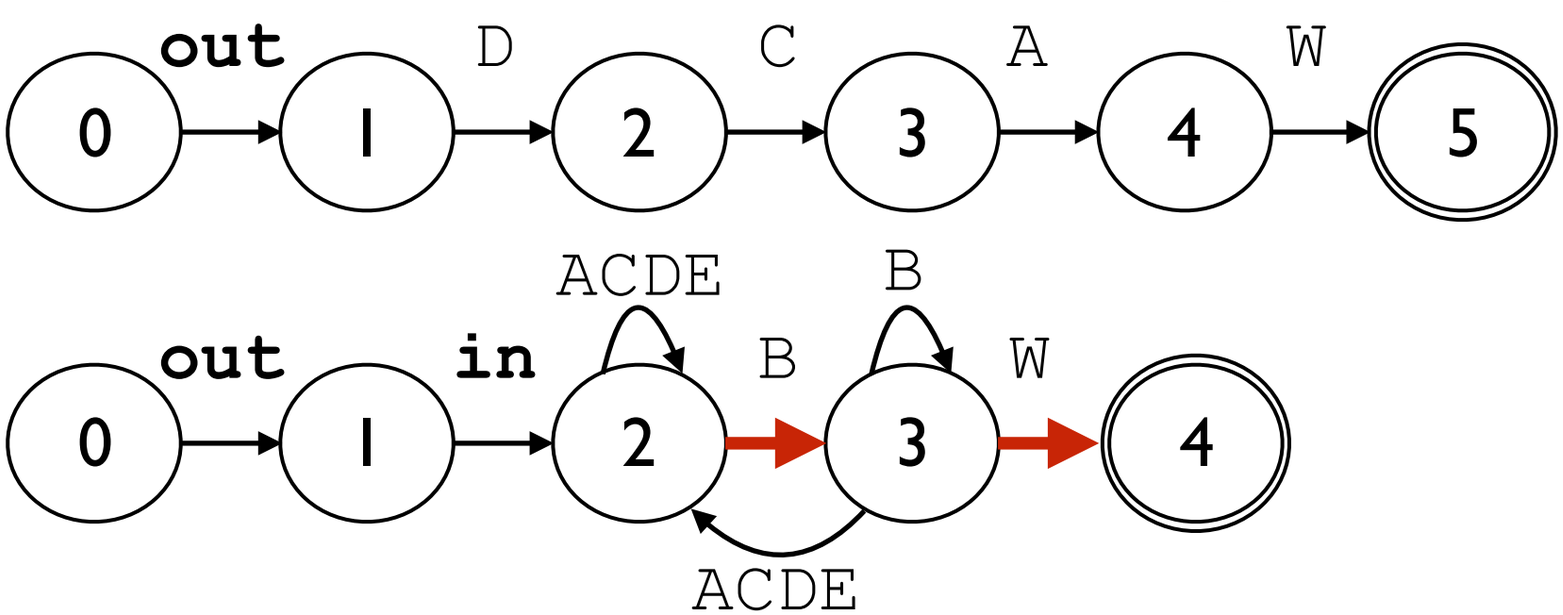


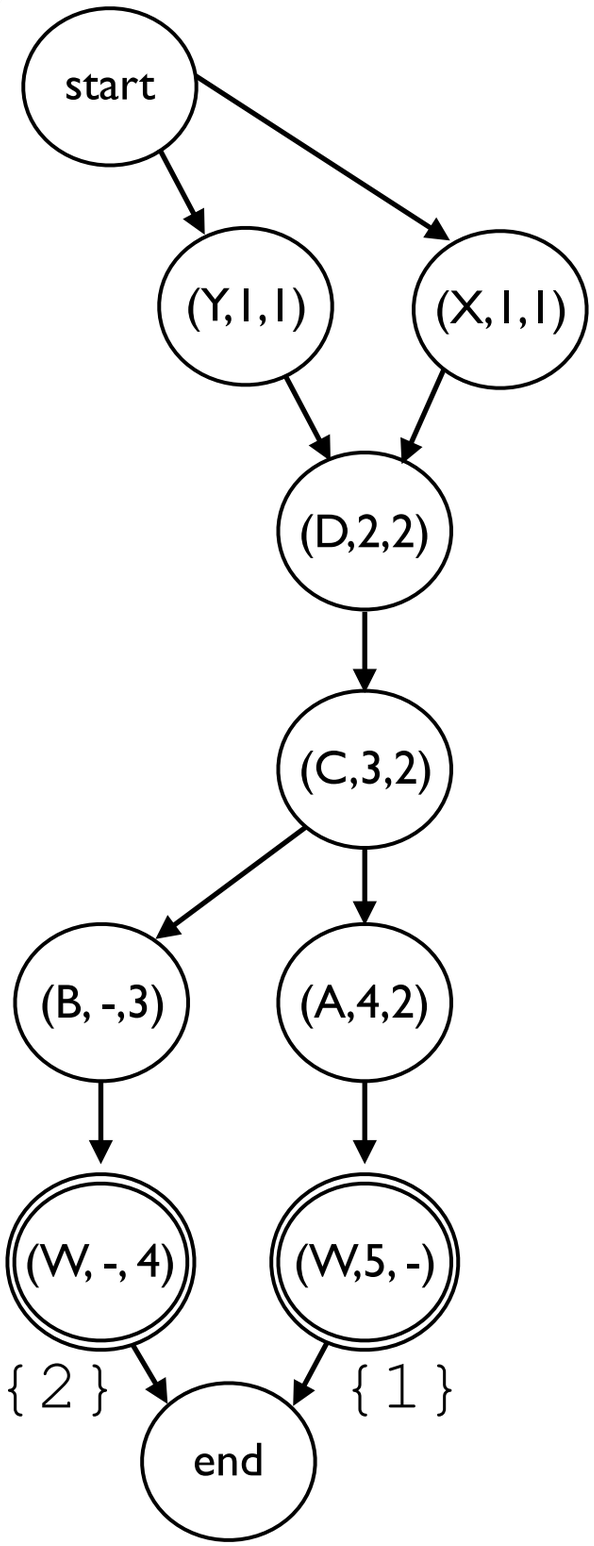


Constructing the Product Graph (PG)

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

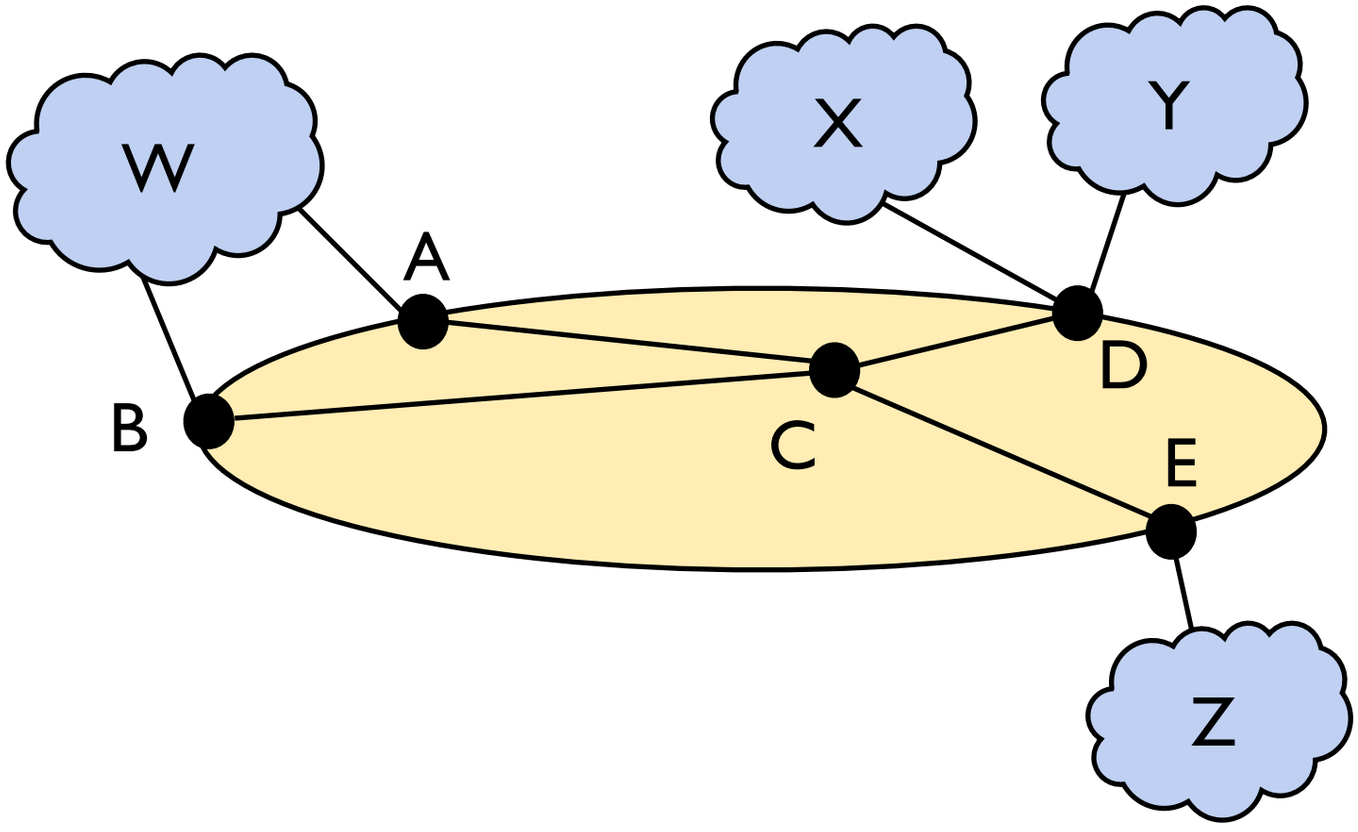


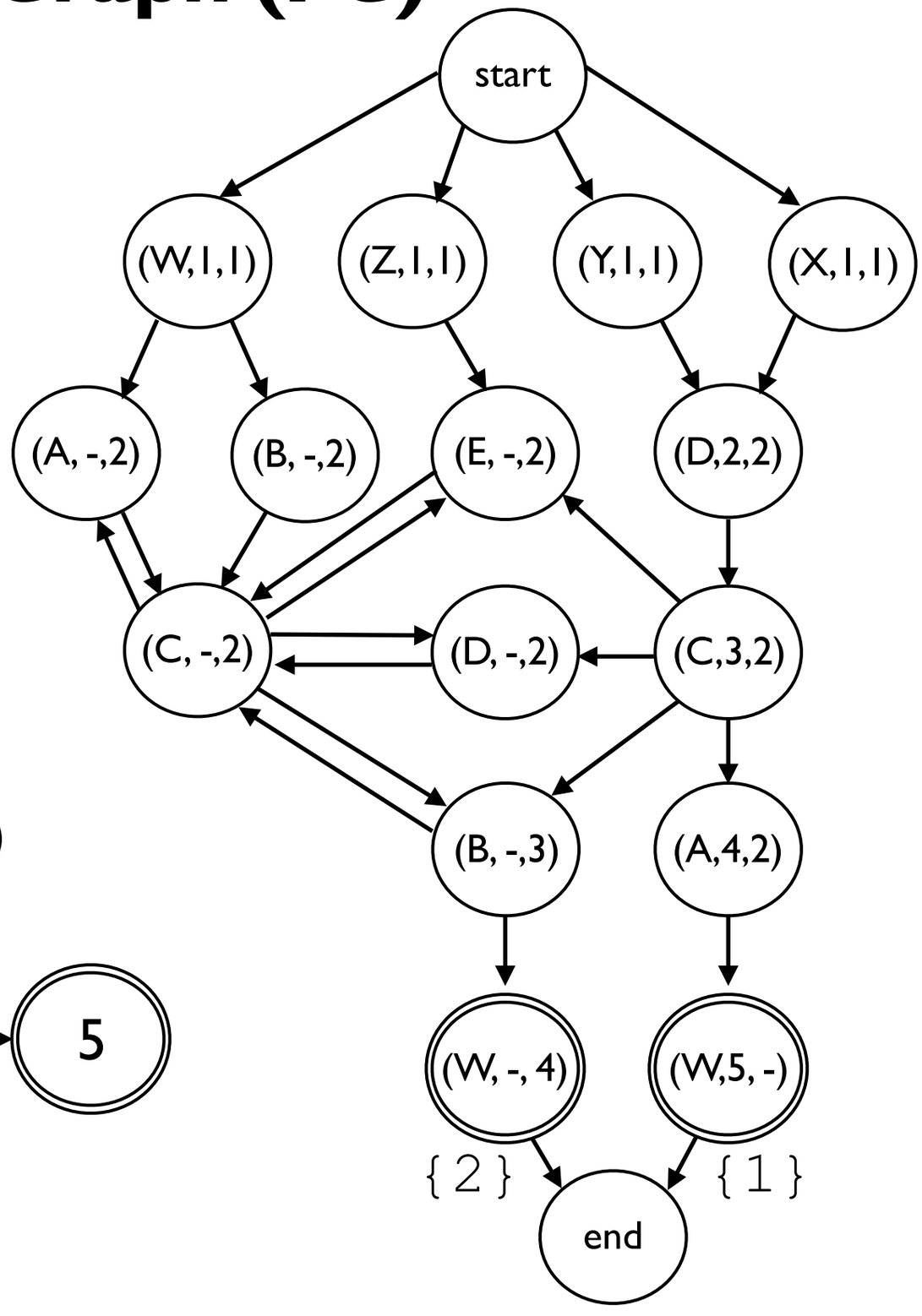


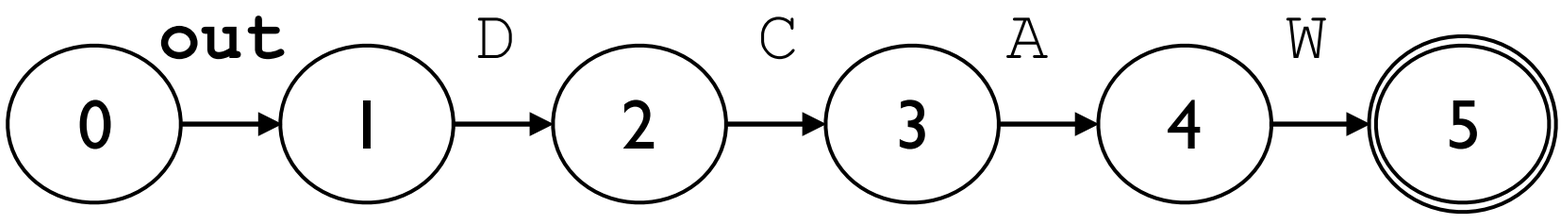


Constructing the Product Graph (PG)

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

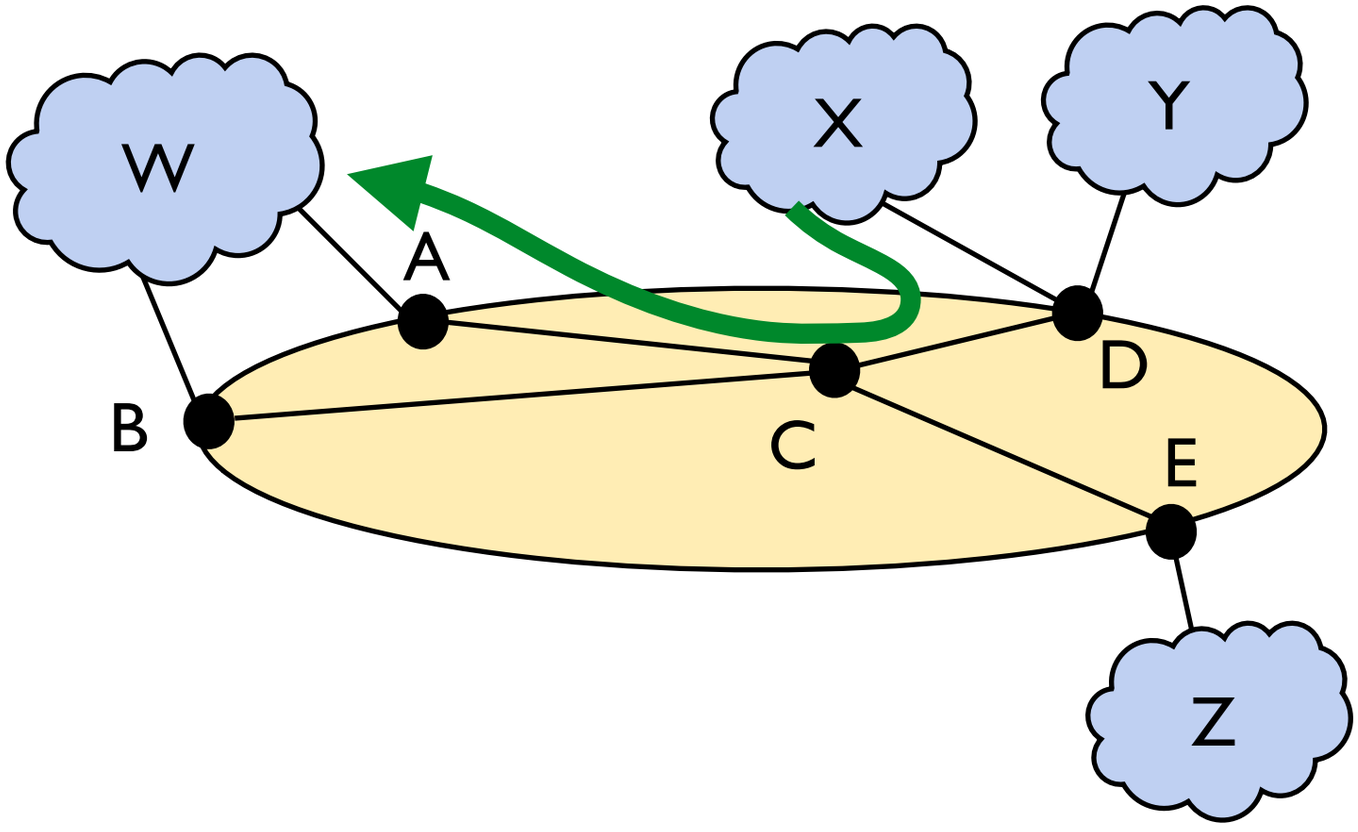


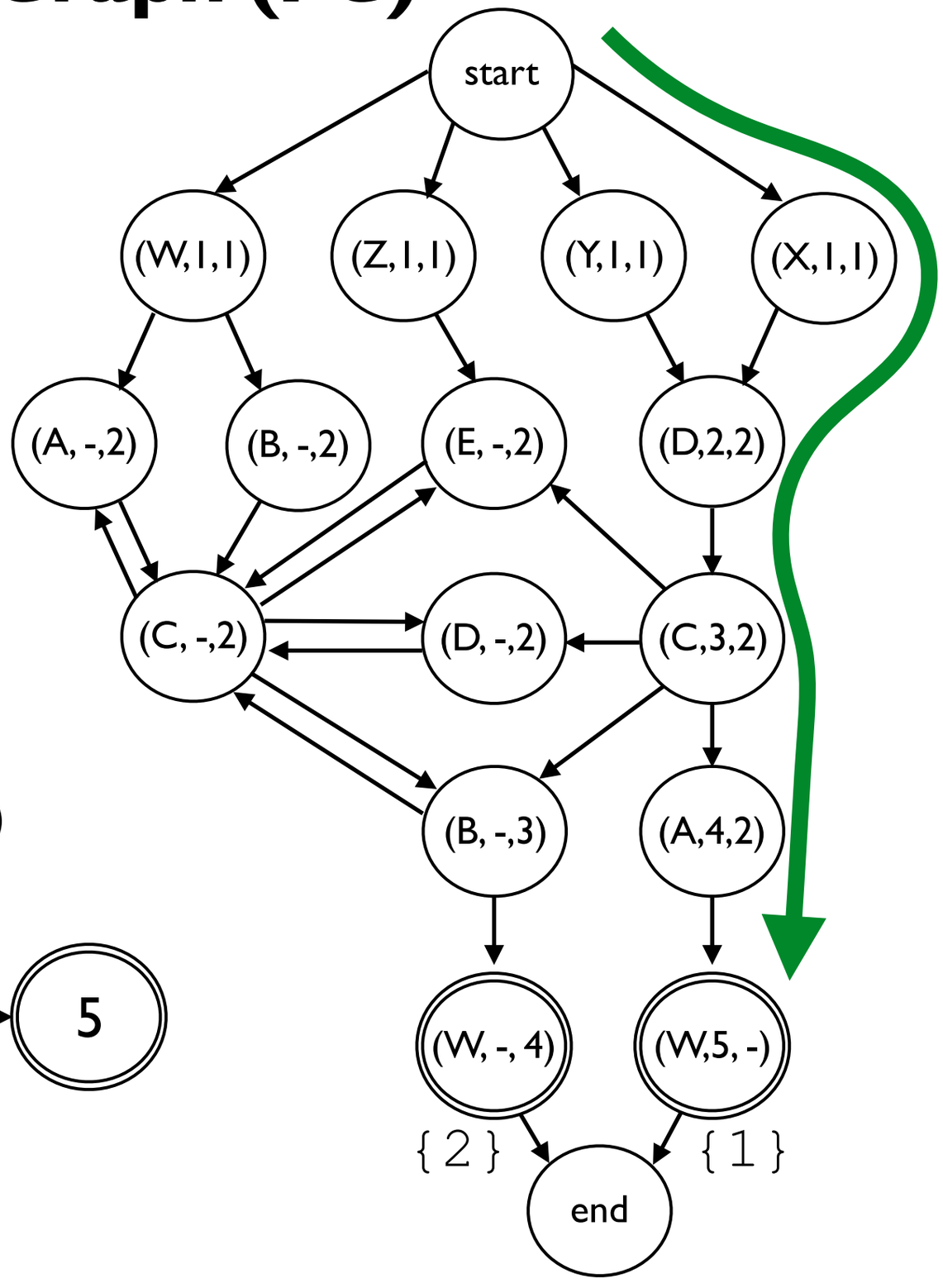


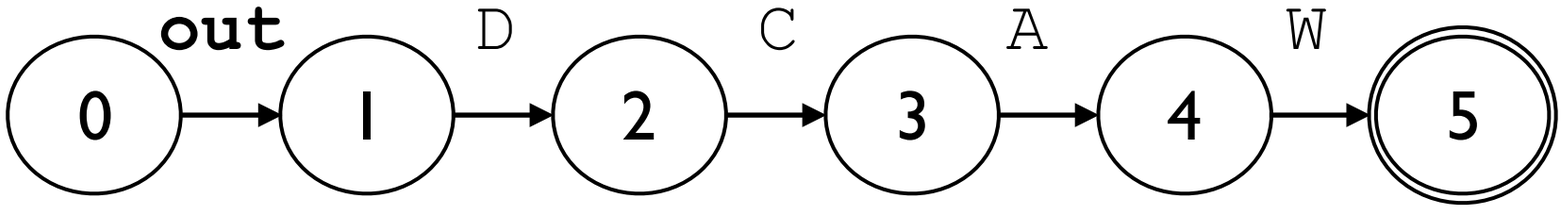


Constructing the Product Graph (PG)

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

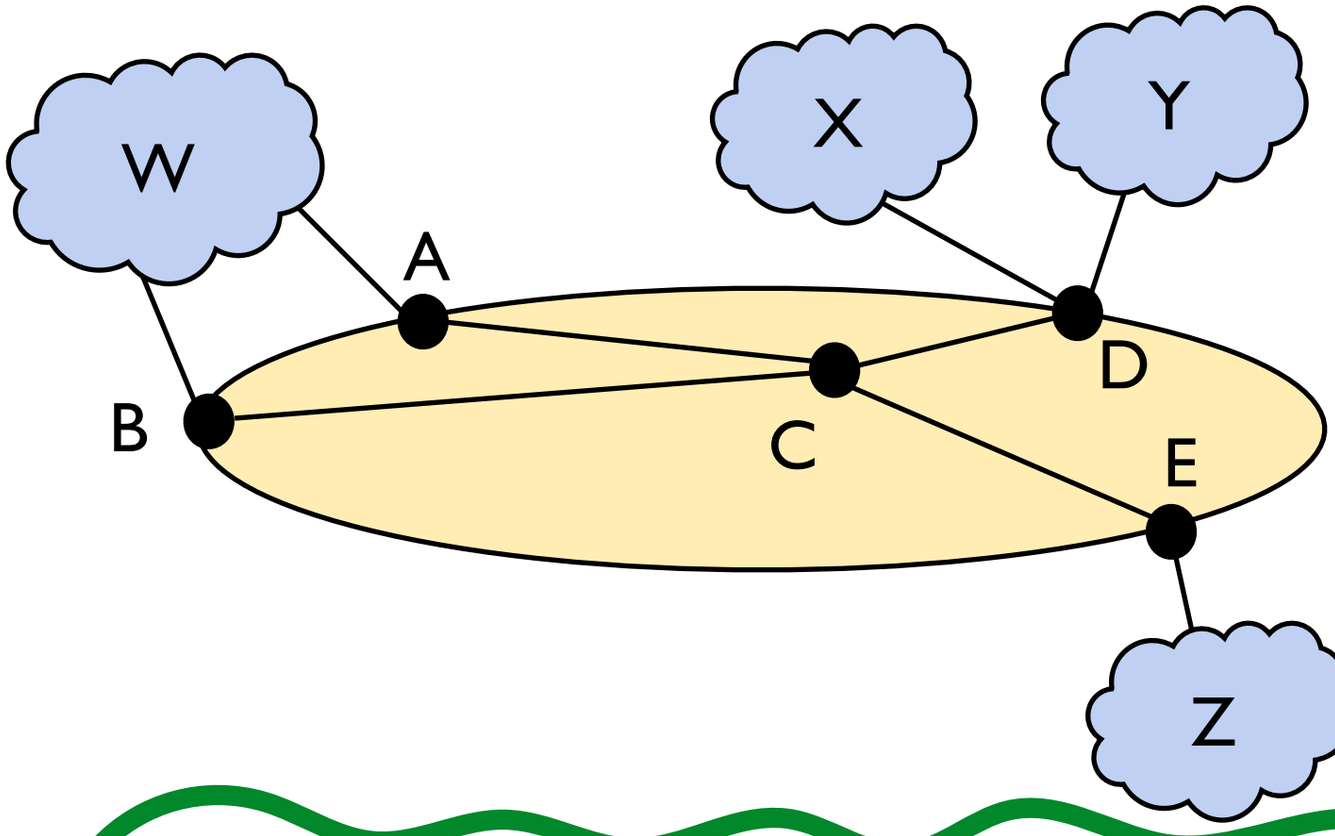


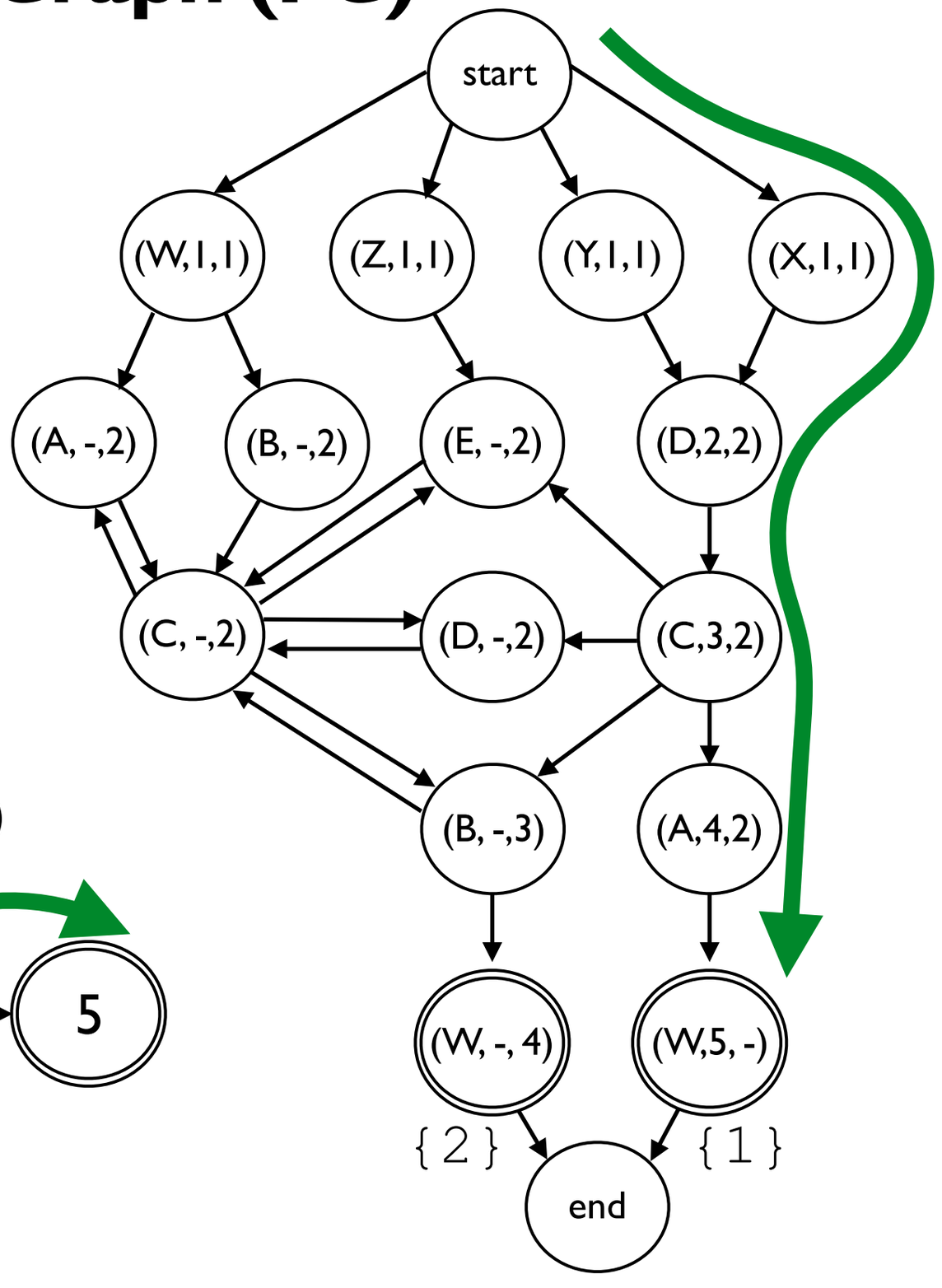




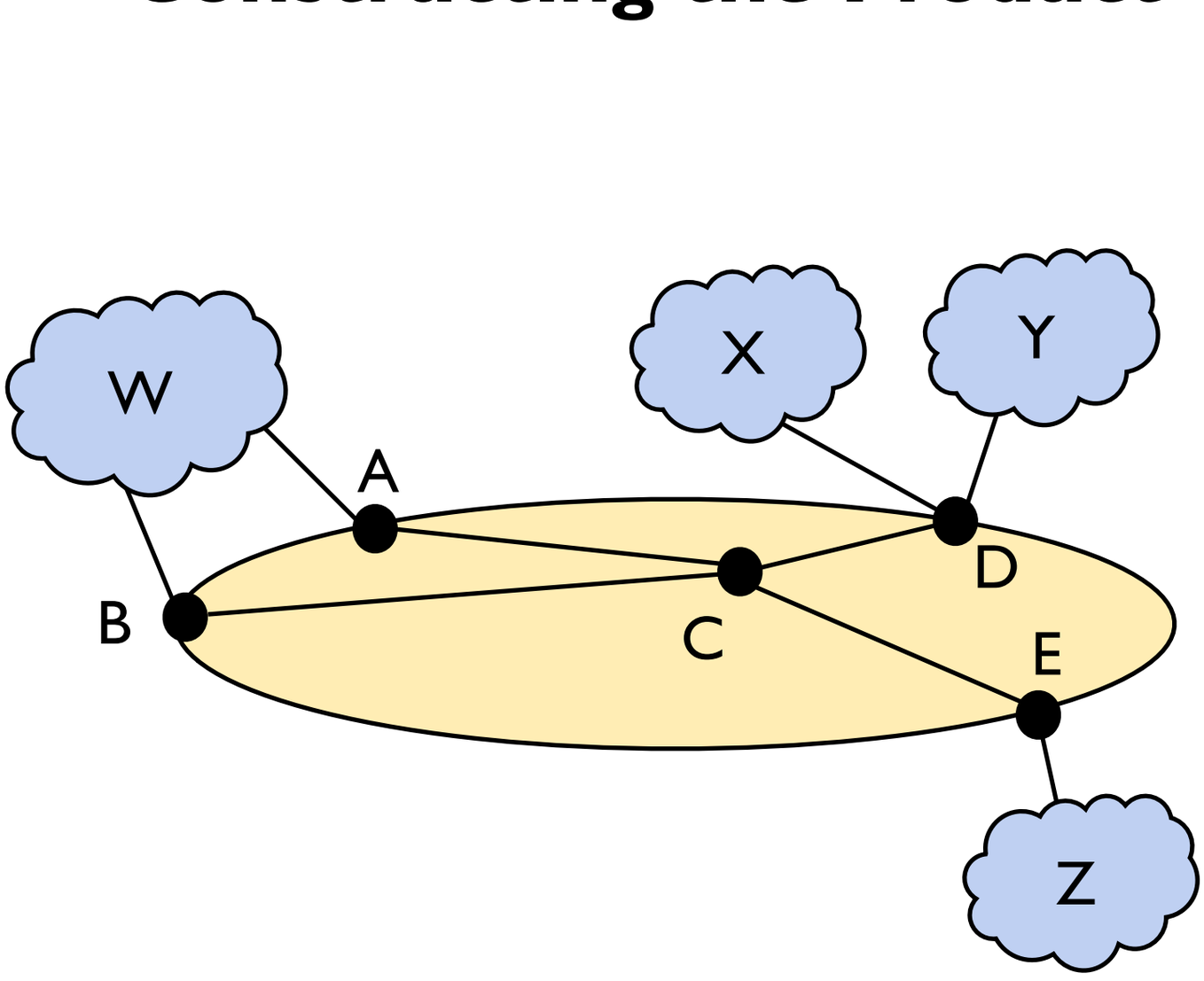
Constructing the Product Graph (PG)

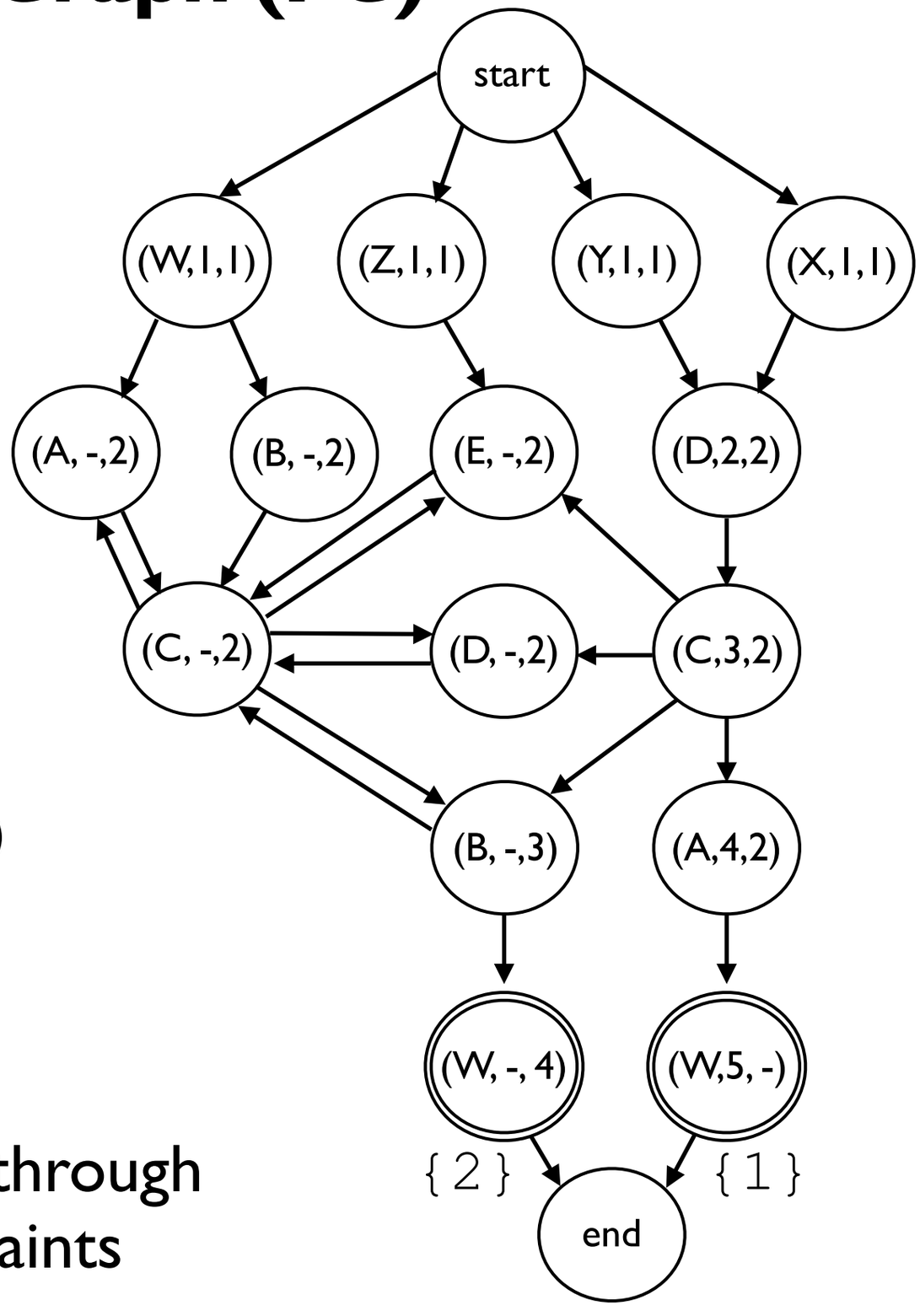
```
(W.A.C.D.out) >> (W.B.in+.out)
```





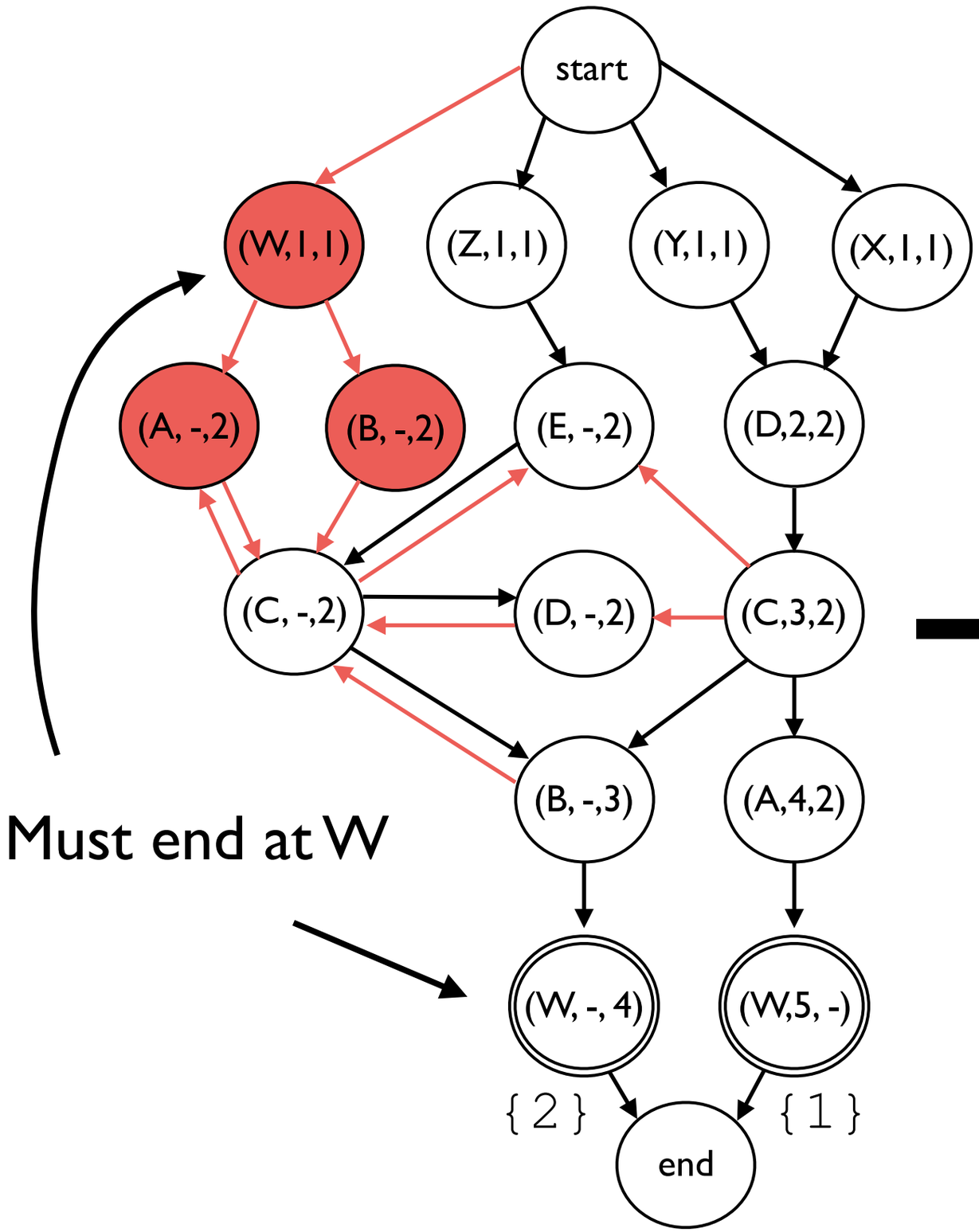
Constructing the Product Graph (PG)



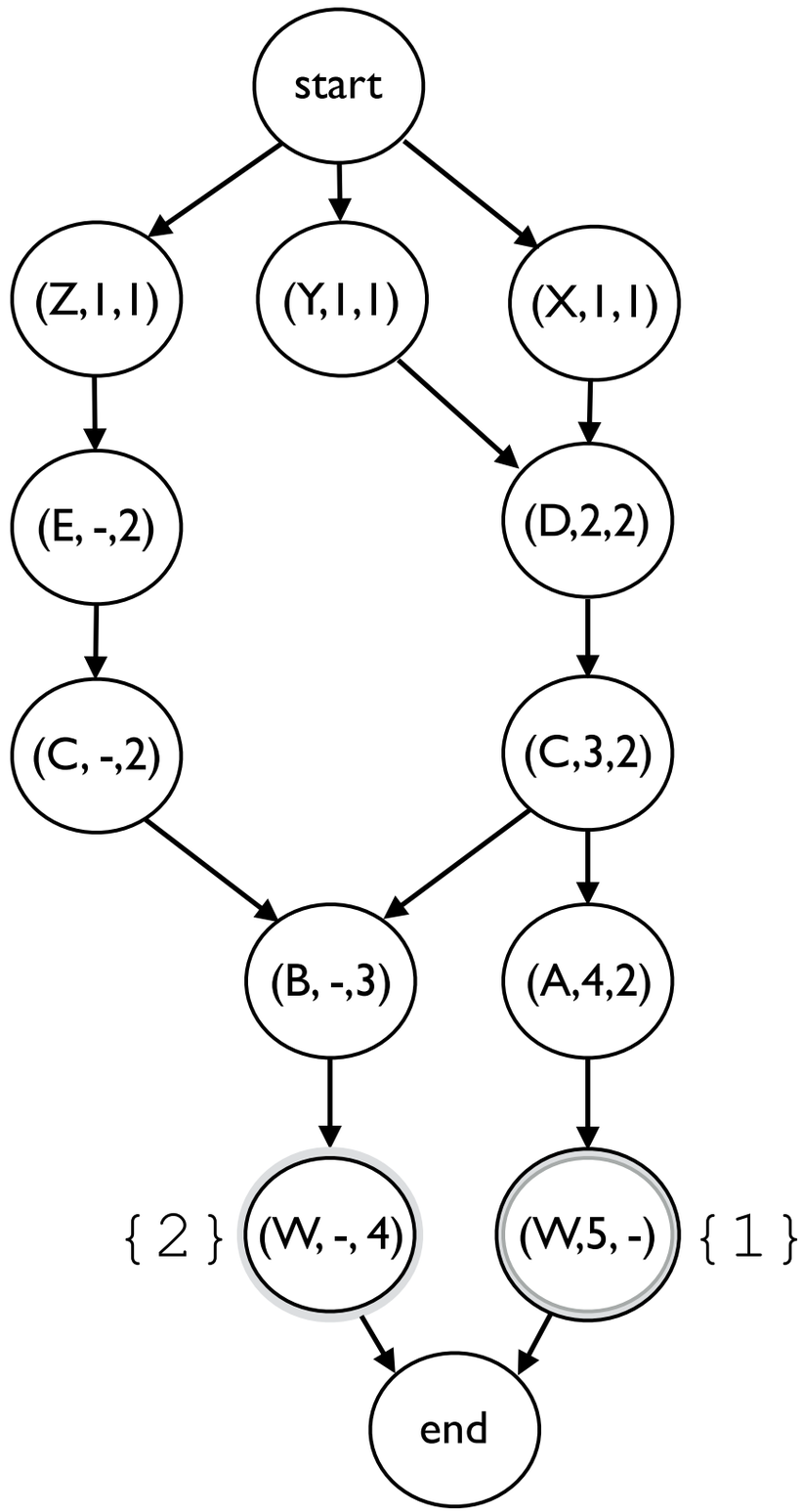


Compact representation of all paths through the topology subject to policy constraints

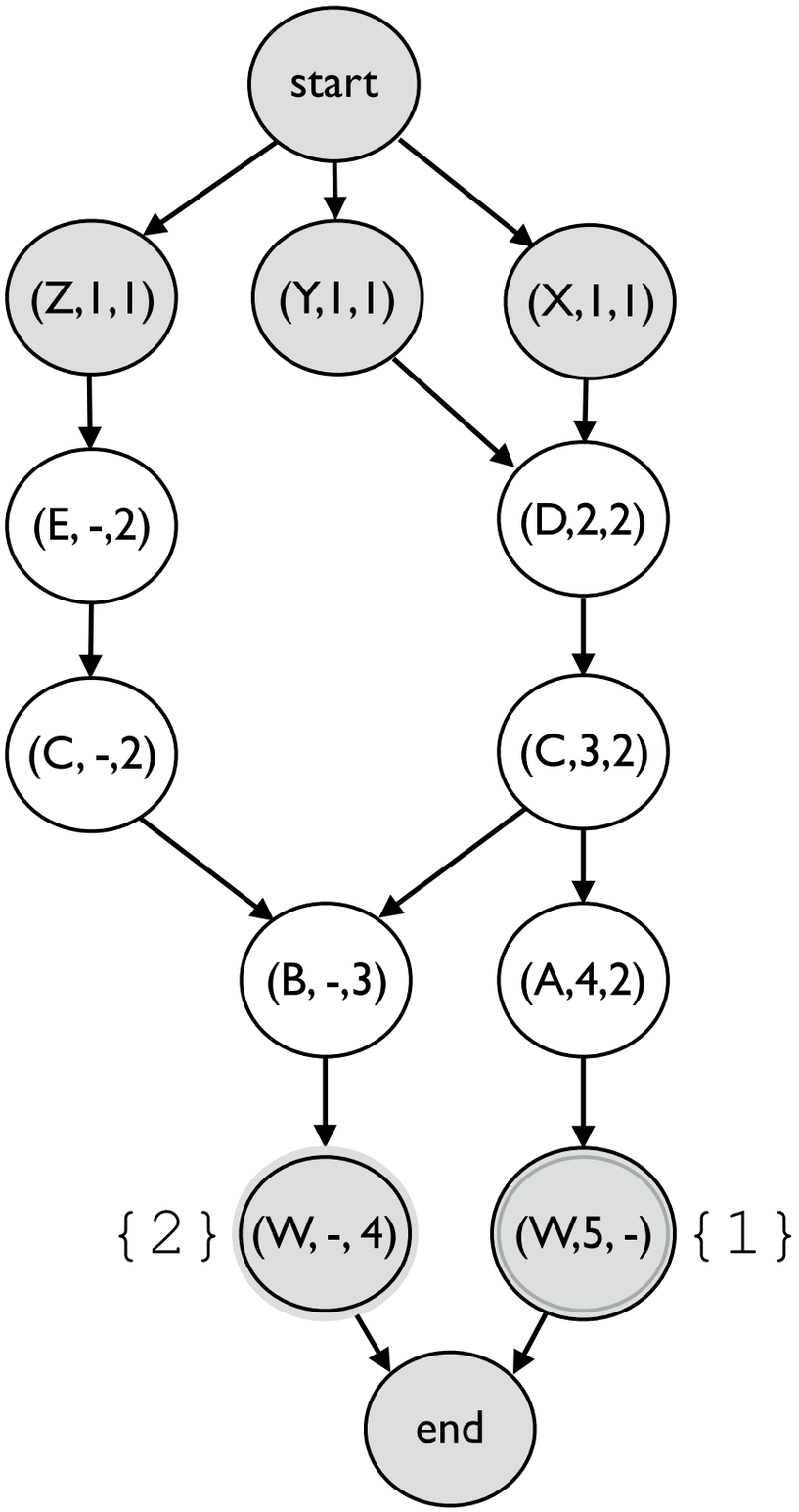
PG Minimization (Loop analysis)



Must end at W



Compilation to BGP:

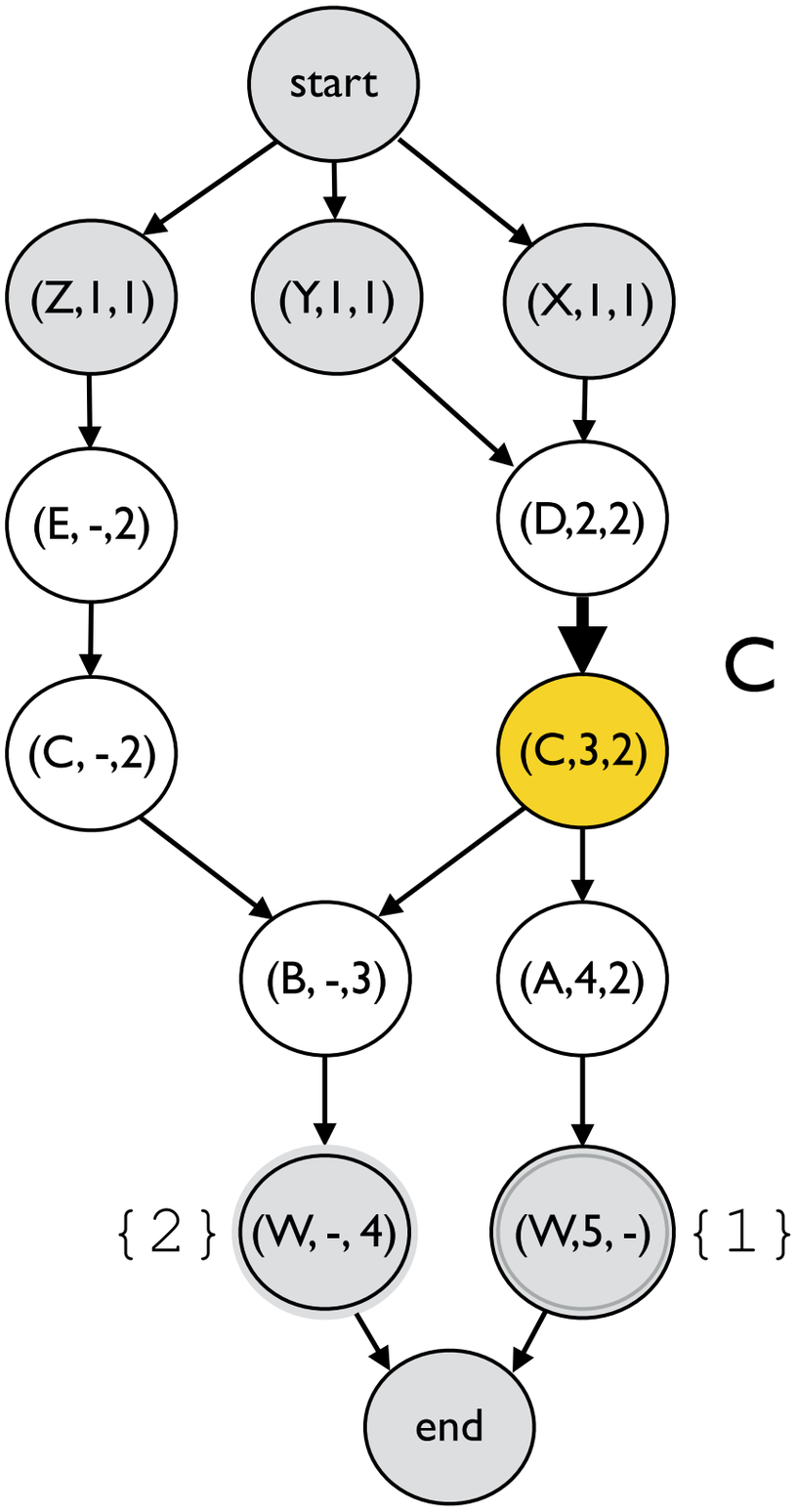


Idea 1: Restrict advertisements to PG edges

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

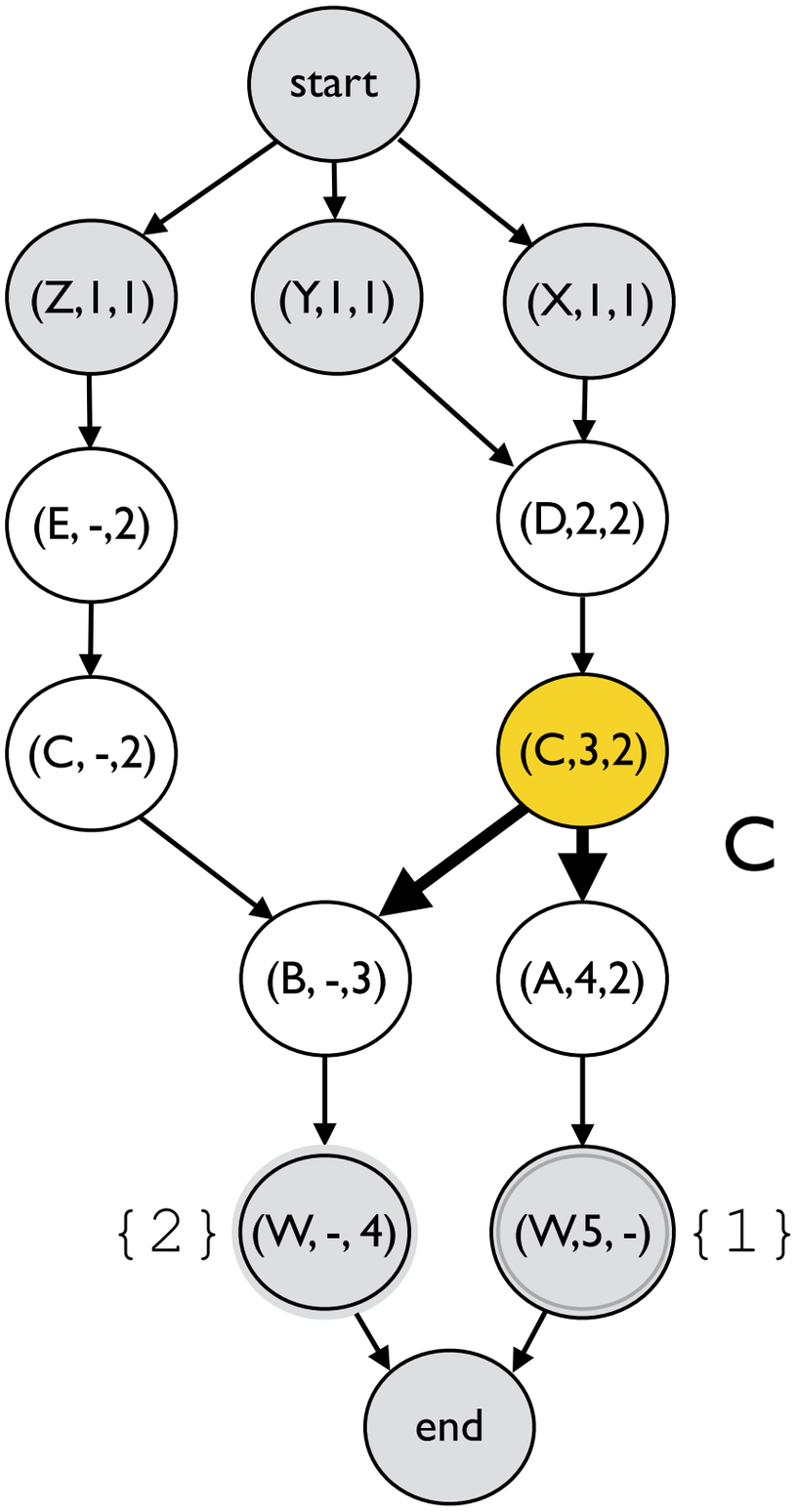
Let BGP find **some allowed** path dynamically

Compilation to BGP:



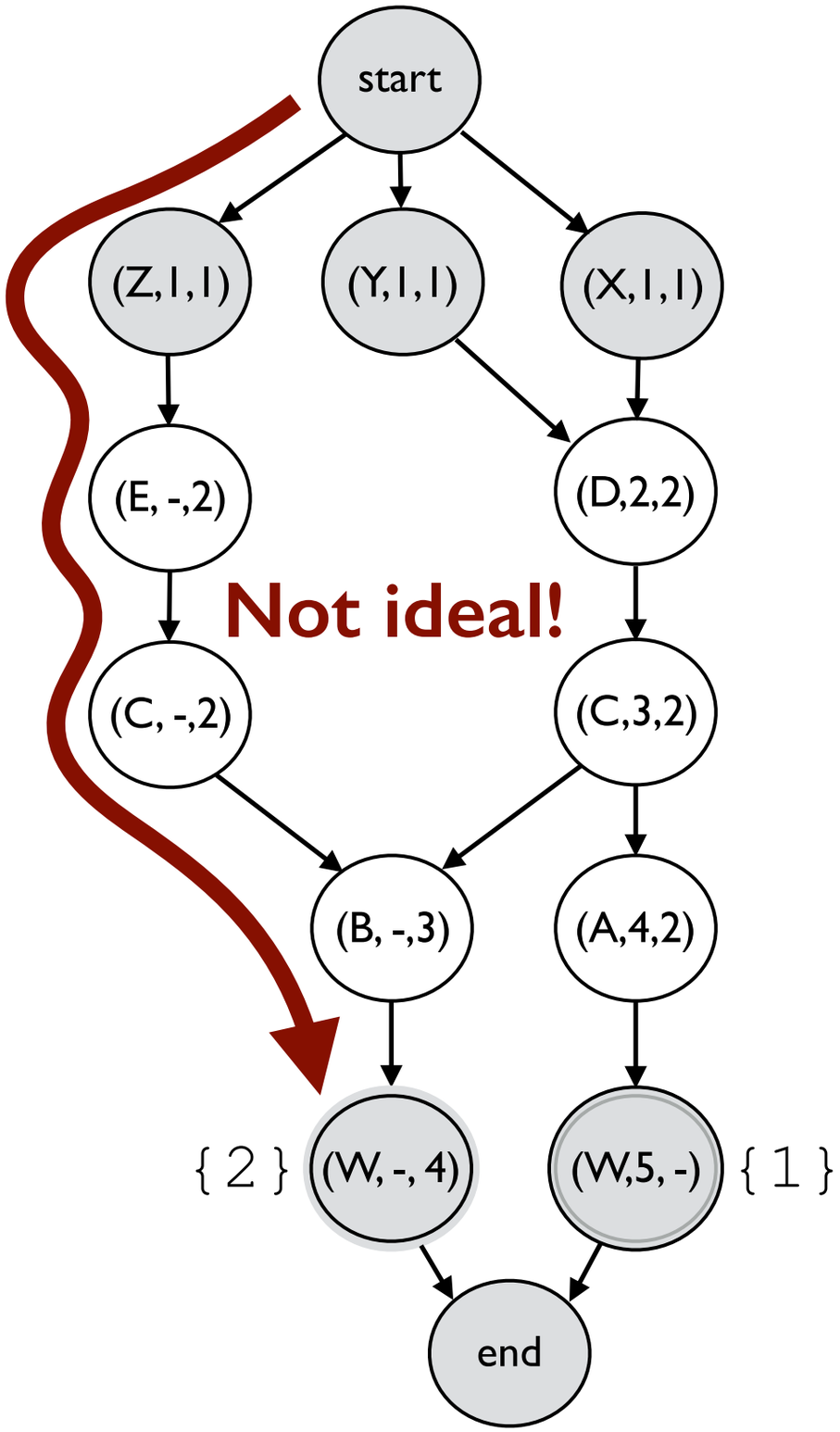
C allows import from D with tag (2,2)

Compilation to BGP:



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

Compilation to BGP:

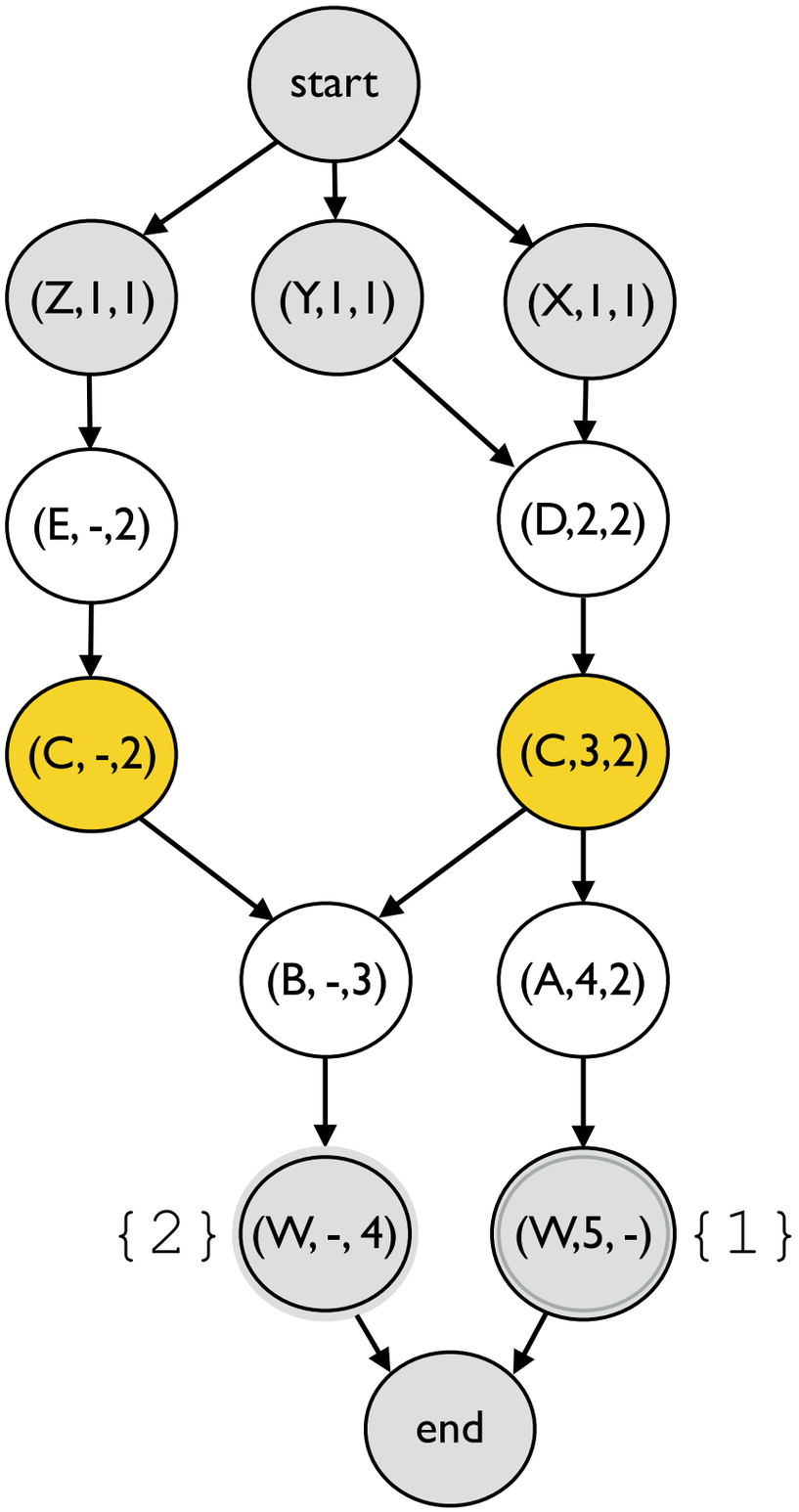


Idea 1: Restrict advertisements to PG edges

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

Let BGP find **some allowed** path dynamically

Compilation to BGP:

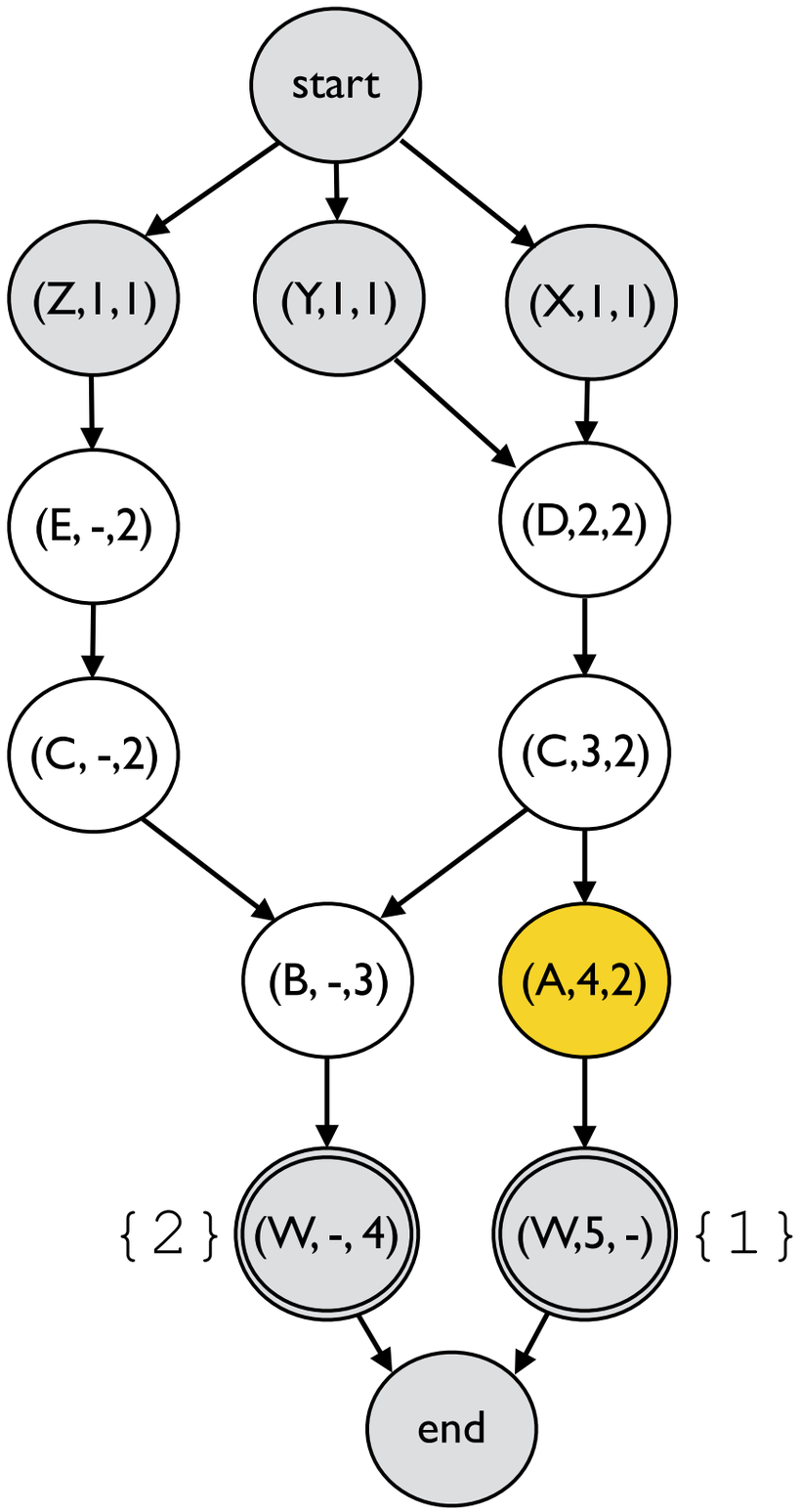


Idea 2: Find local preferences

- Direct BGP towards best path
- Under all combinations of failures
- Frame in terms of graph algorithms

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)

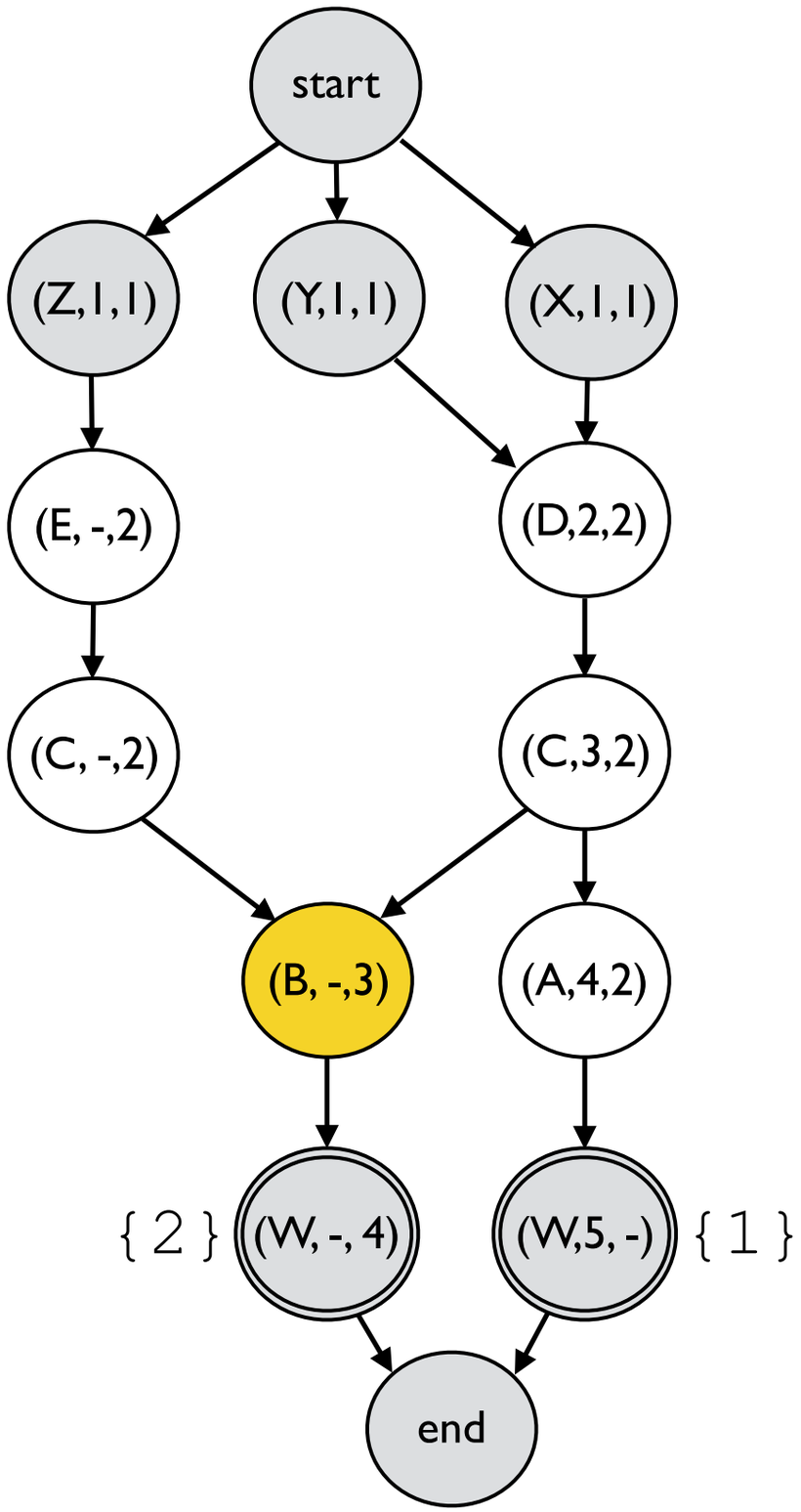
Router D

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[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)

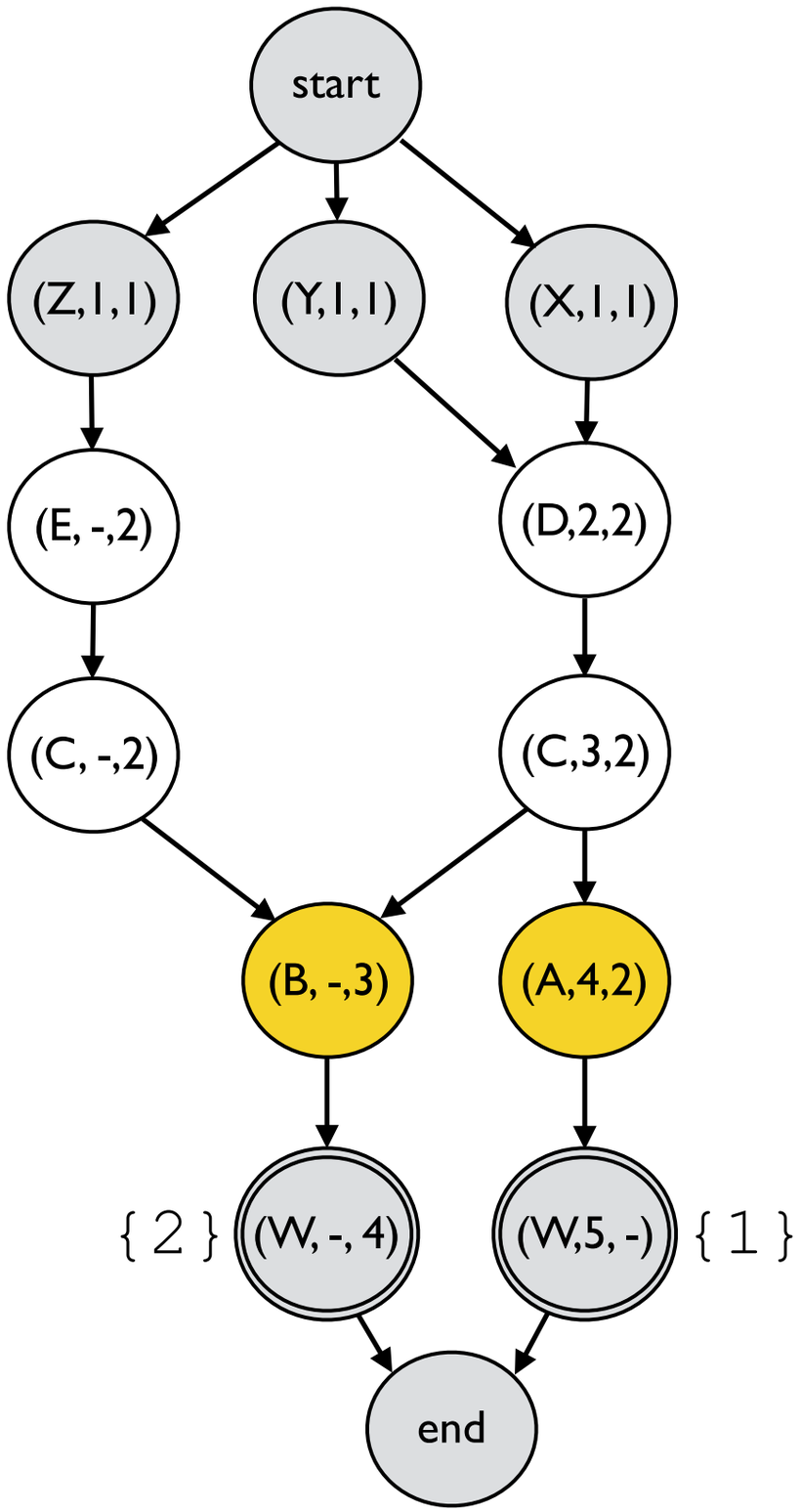
Router D

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)

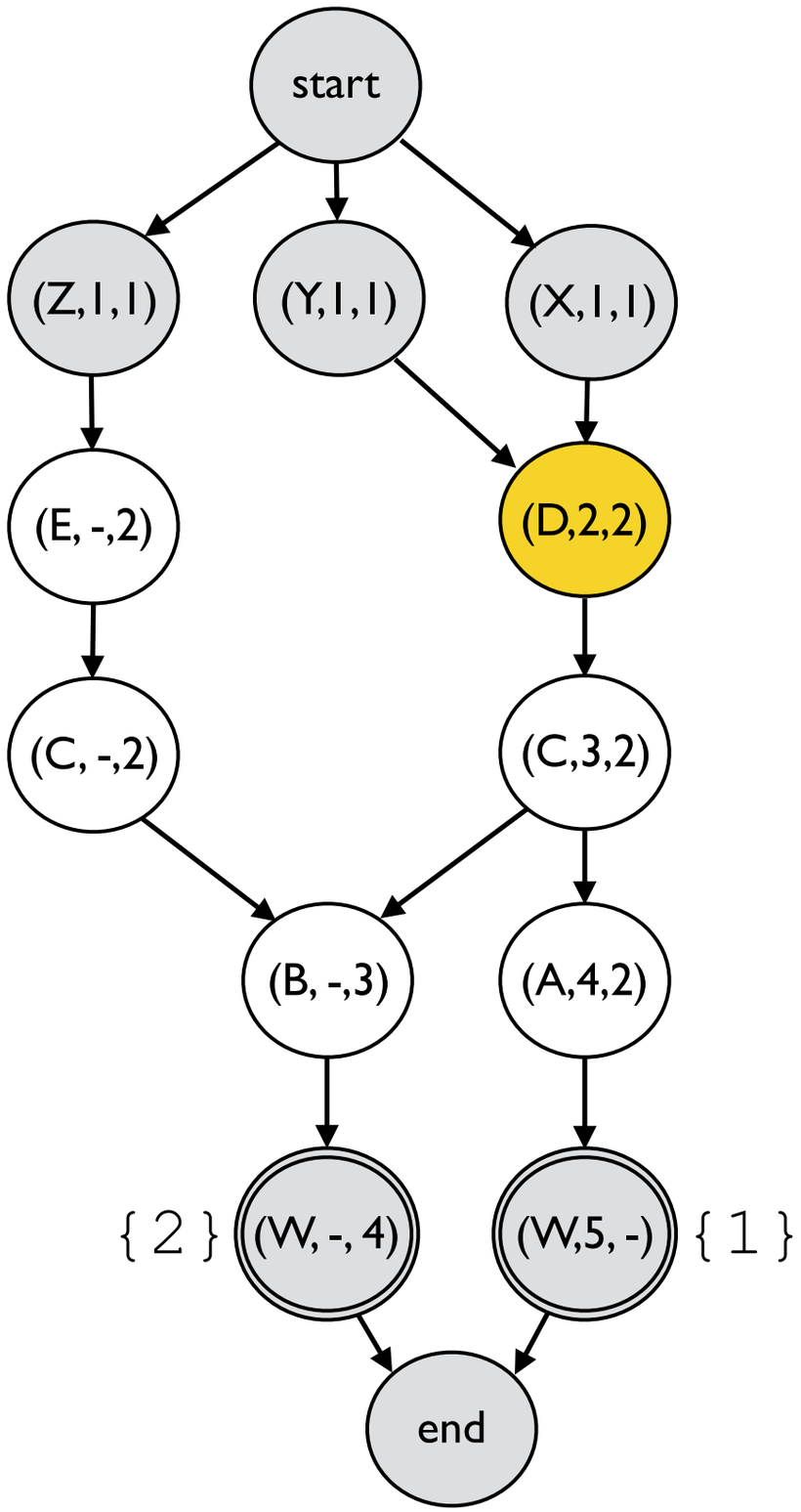
Router D

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)

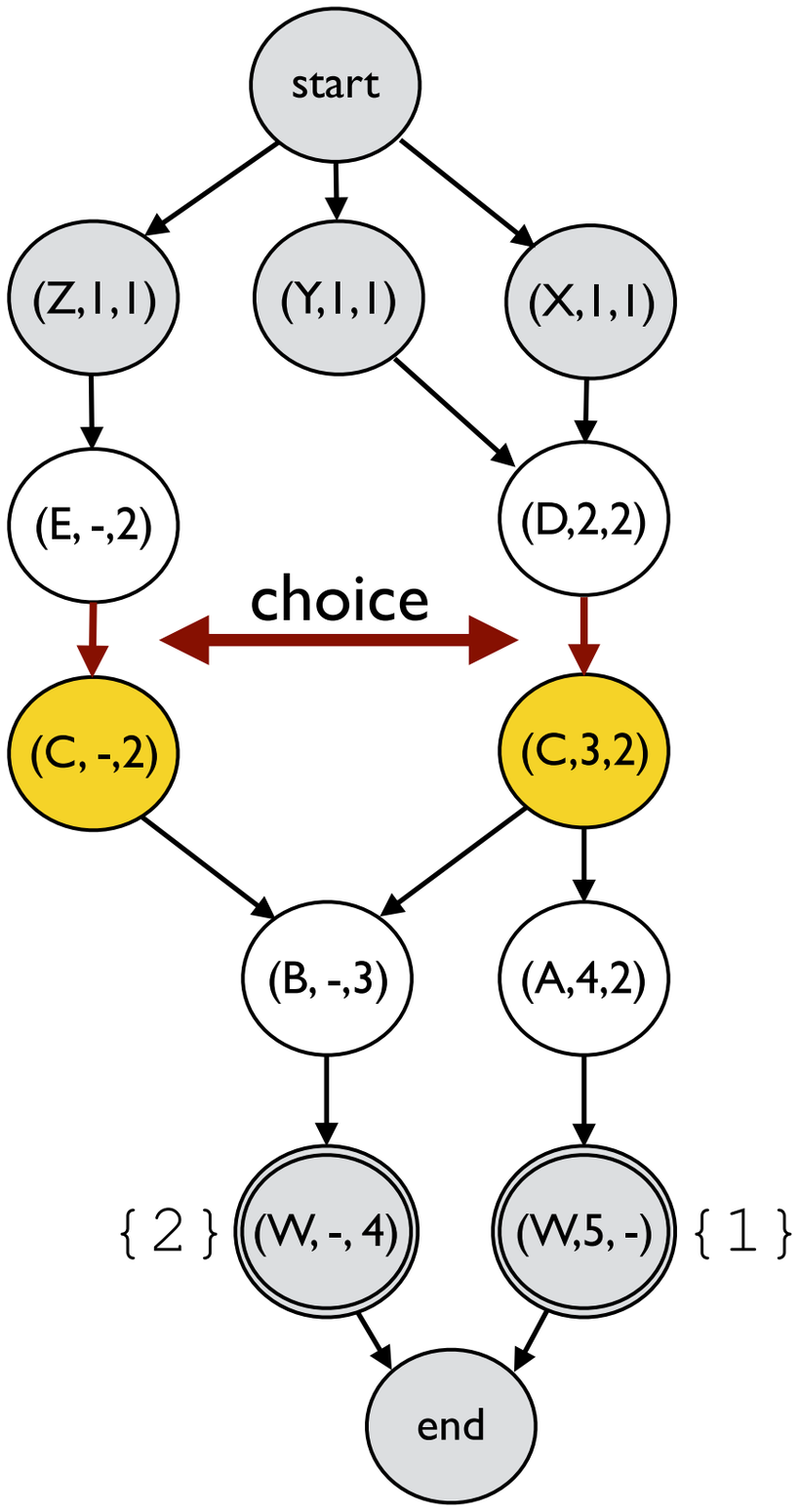
Router D

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)

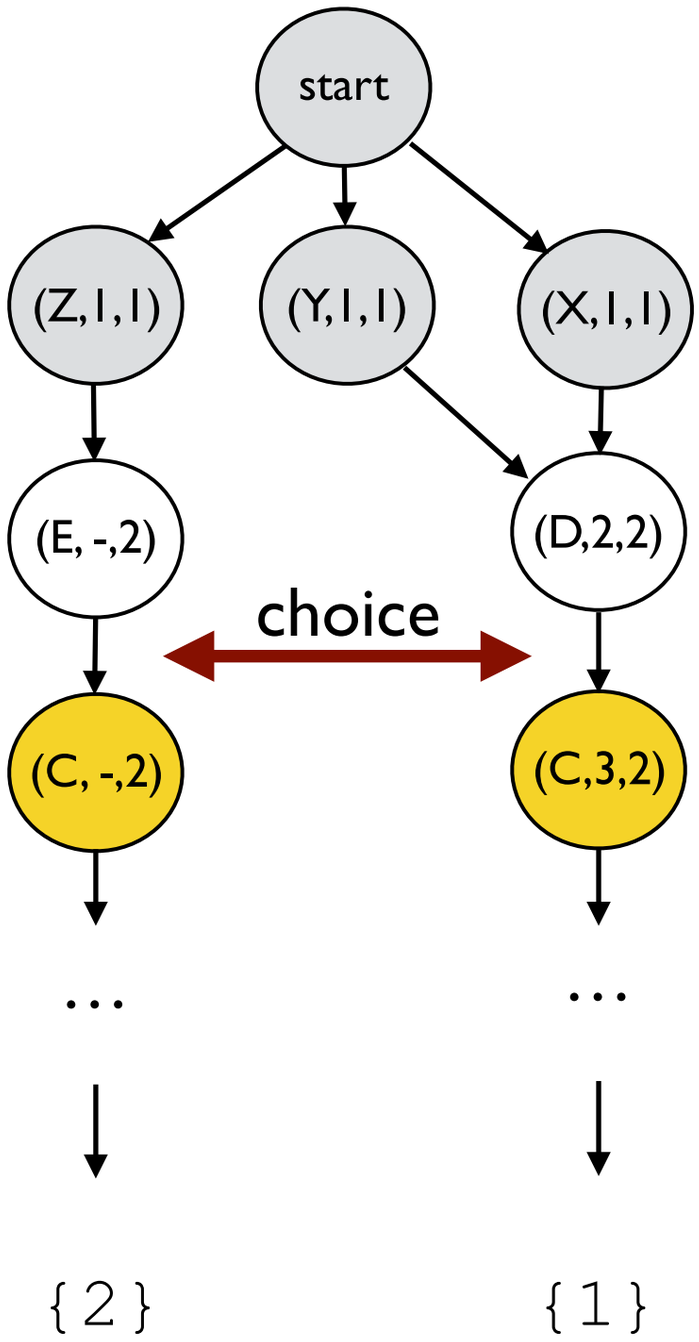
Router D

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)

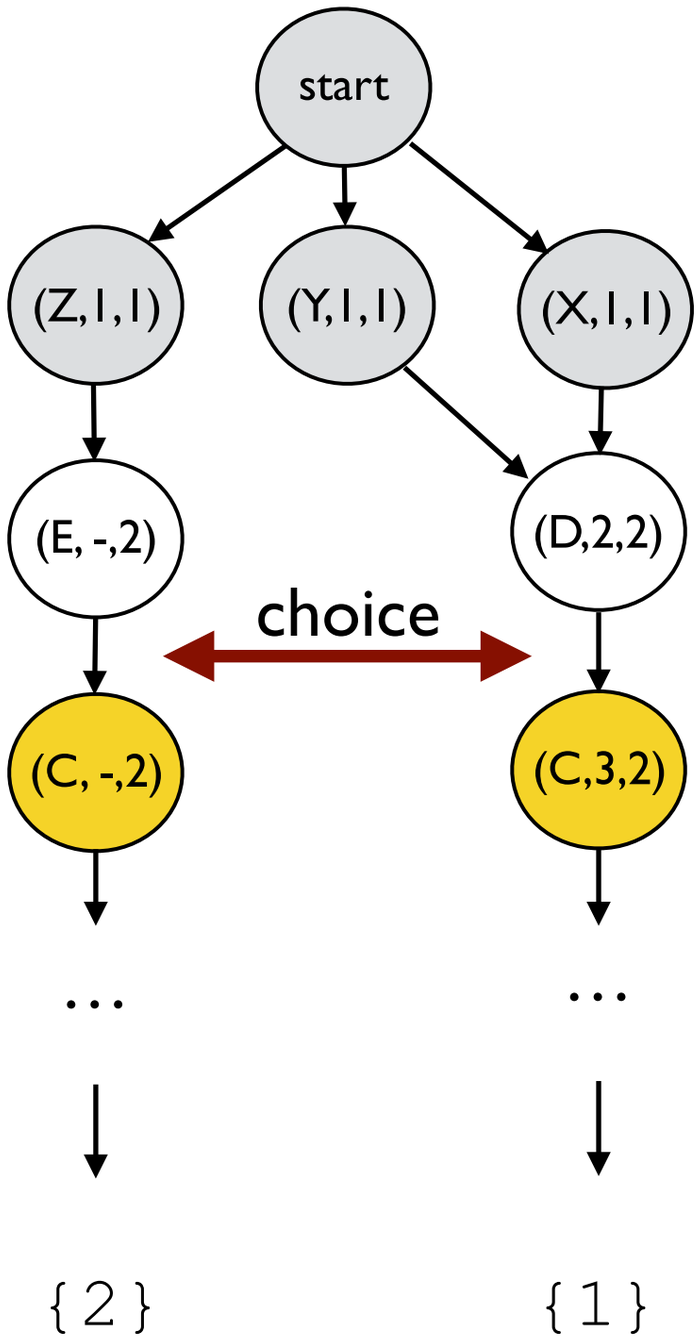
Router D

match regex=(X + Y)

export peer←C, comm←(2,2)

…

Compilation to BGP:



Highest preference obtainable here

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)

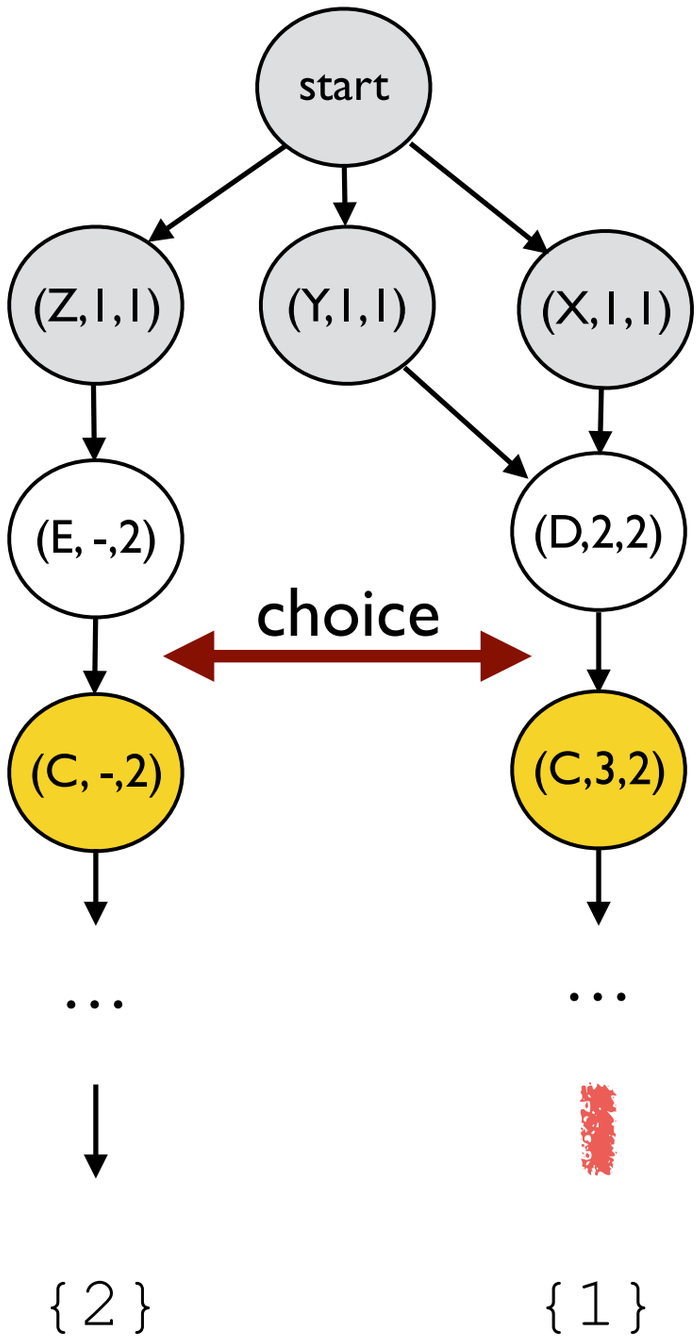
Router D

match regex=(X + Y)

export peer←C, comm←(2,2)

…

Compilation to BGP:



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)

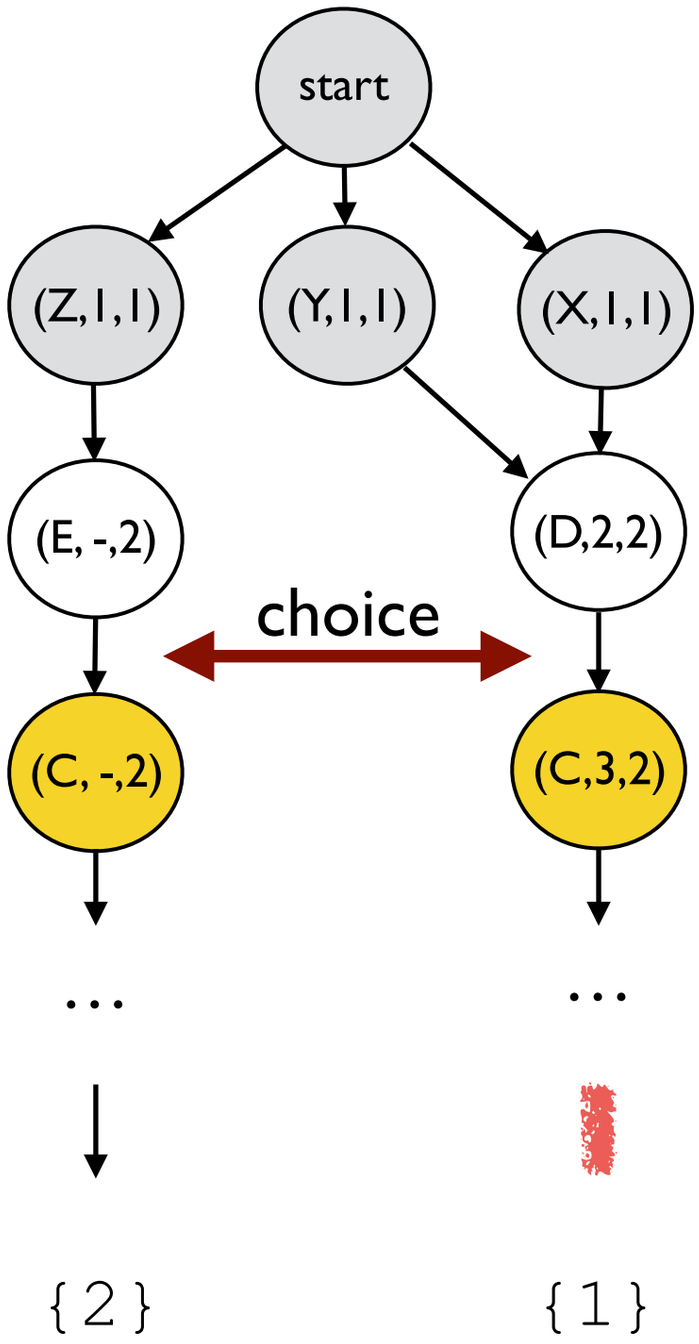
Router D

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)

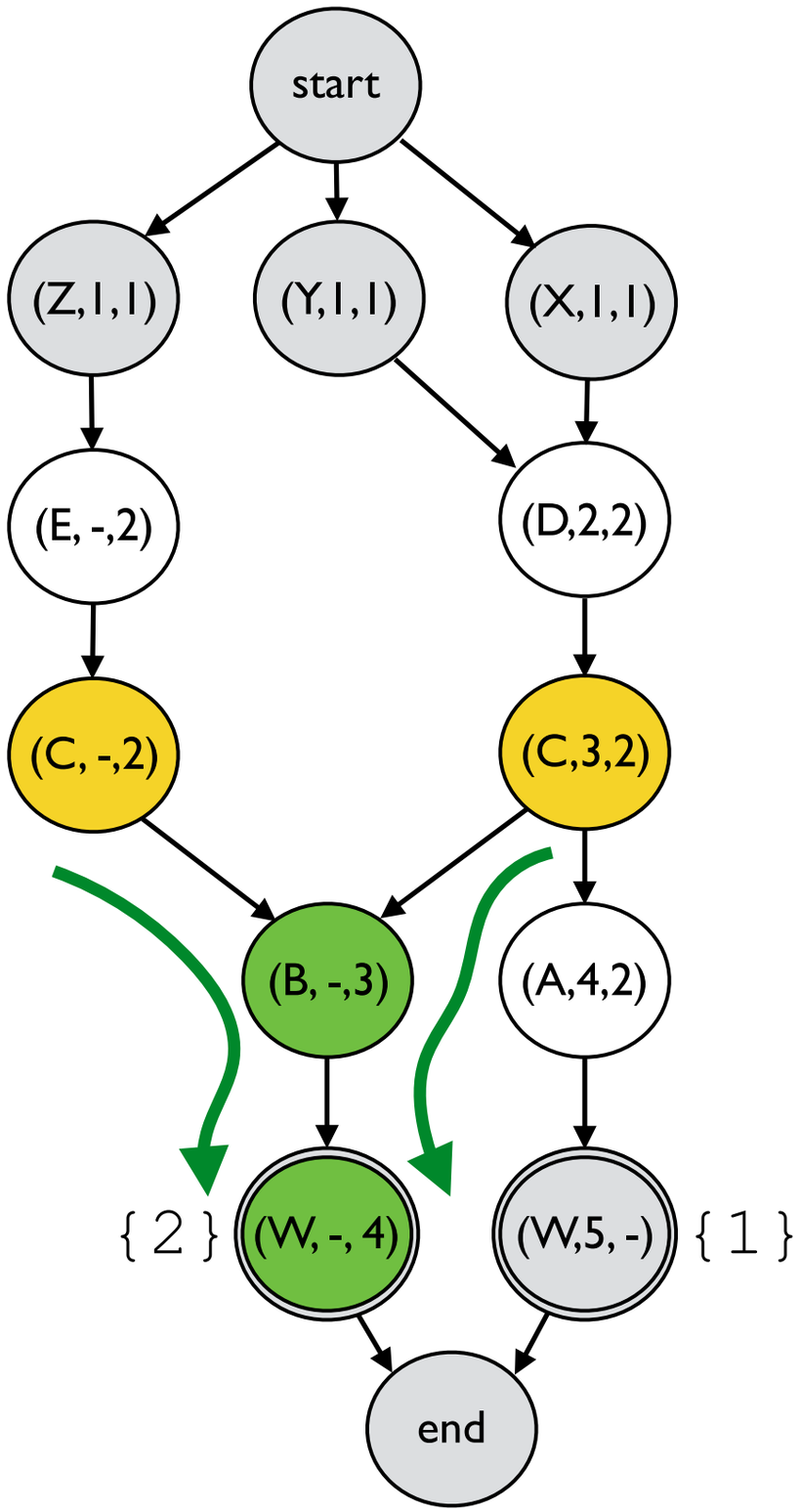
Router D

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)

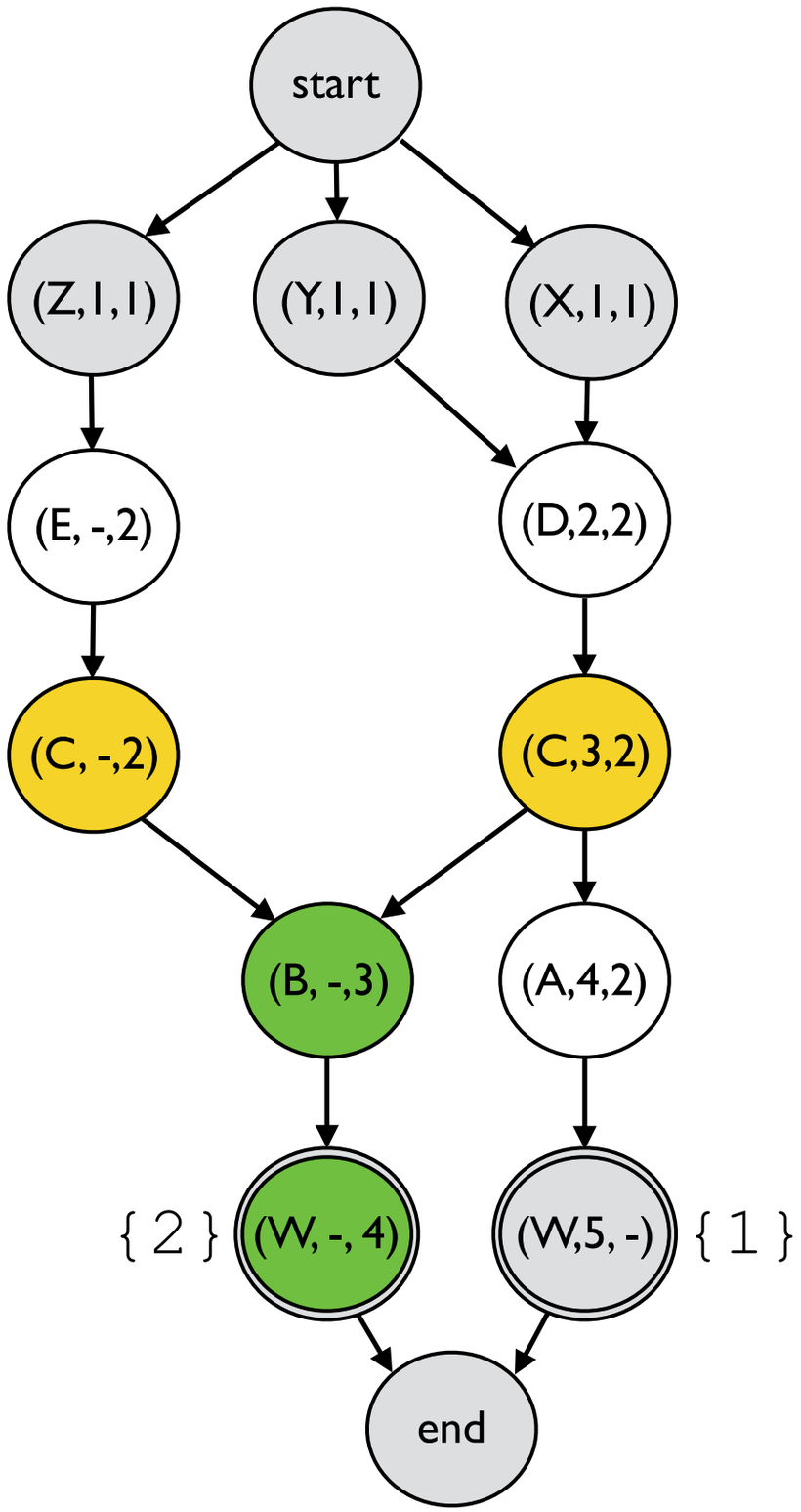
Router D

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)

Router D

match regex=(X + Y)

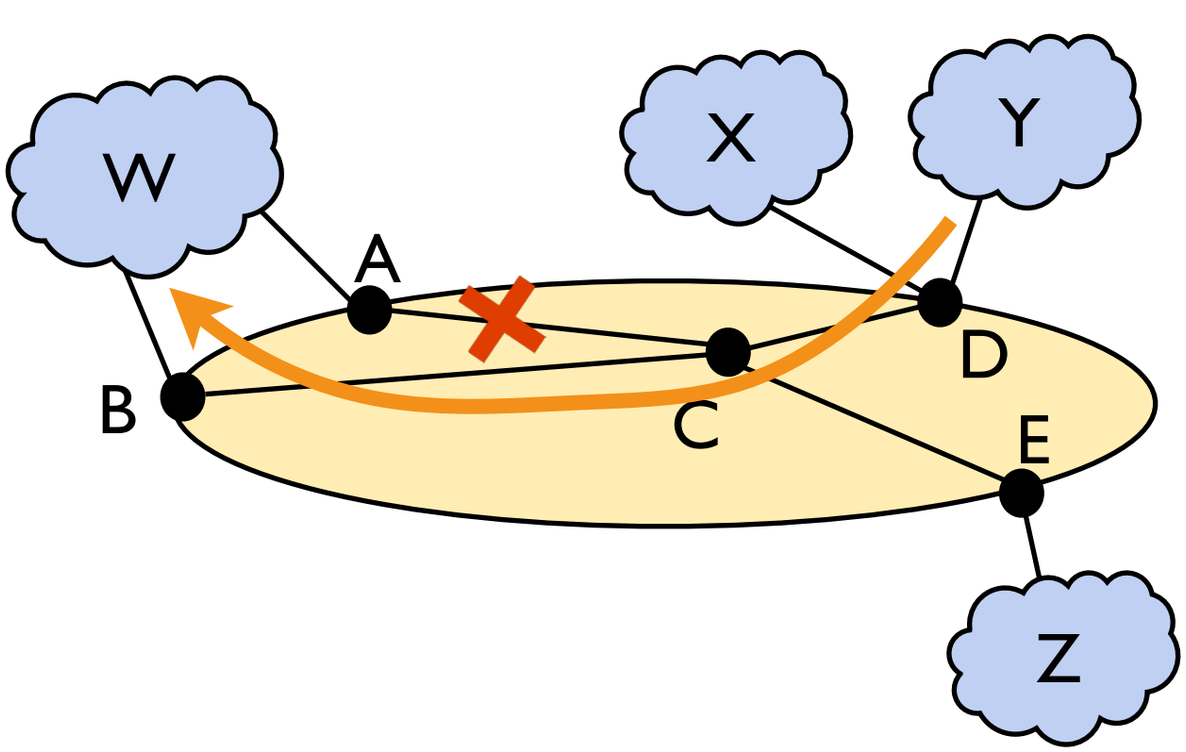
export peer←C, comm←(2,2)

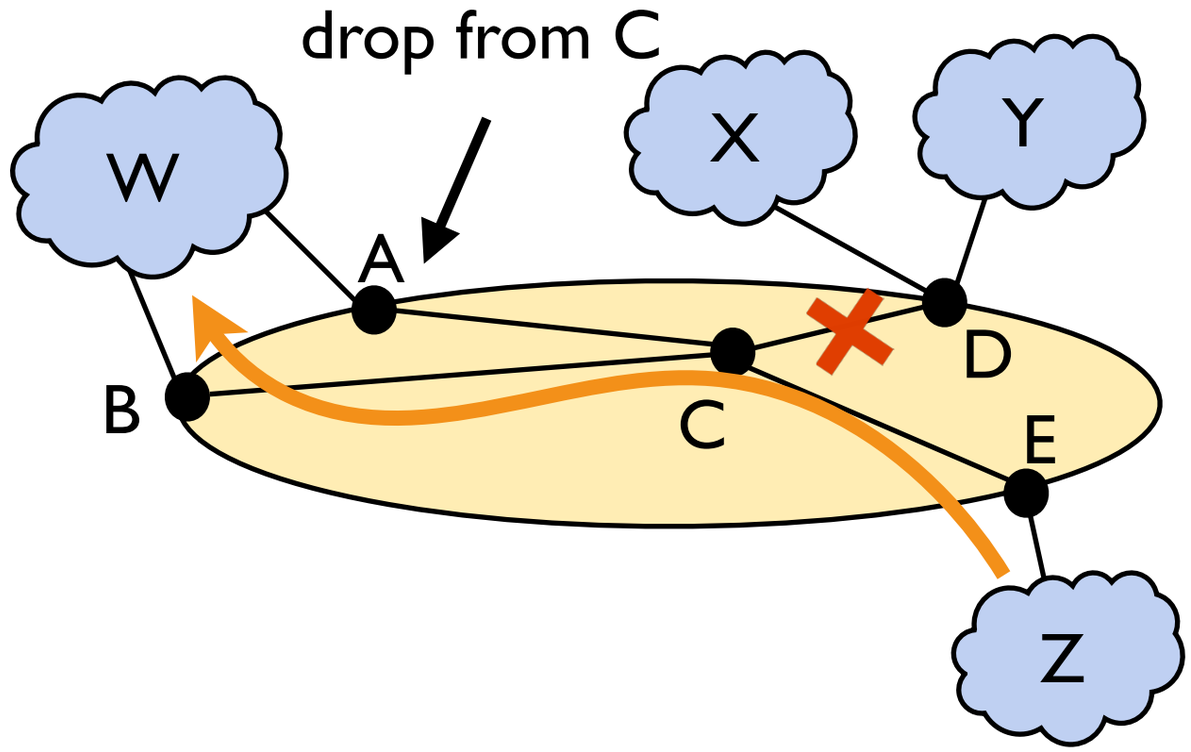
…

Prefer D's announce

Compilation to BGP:

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





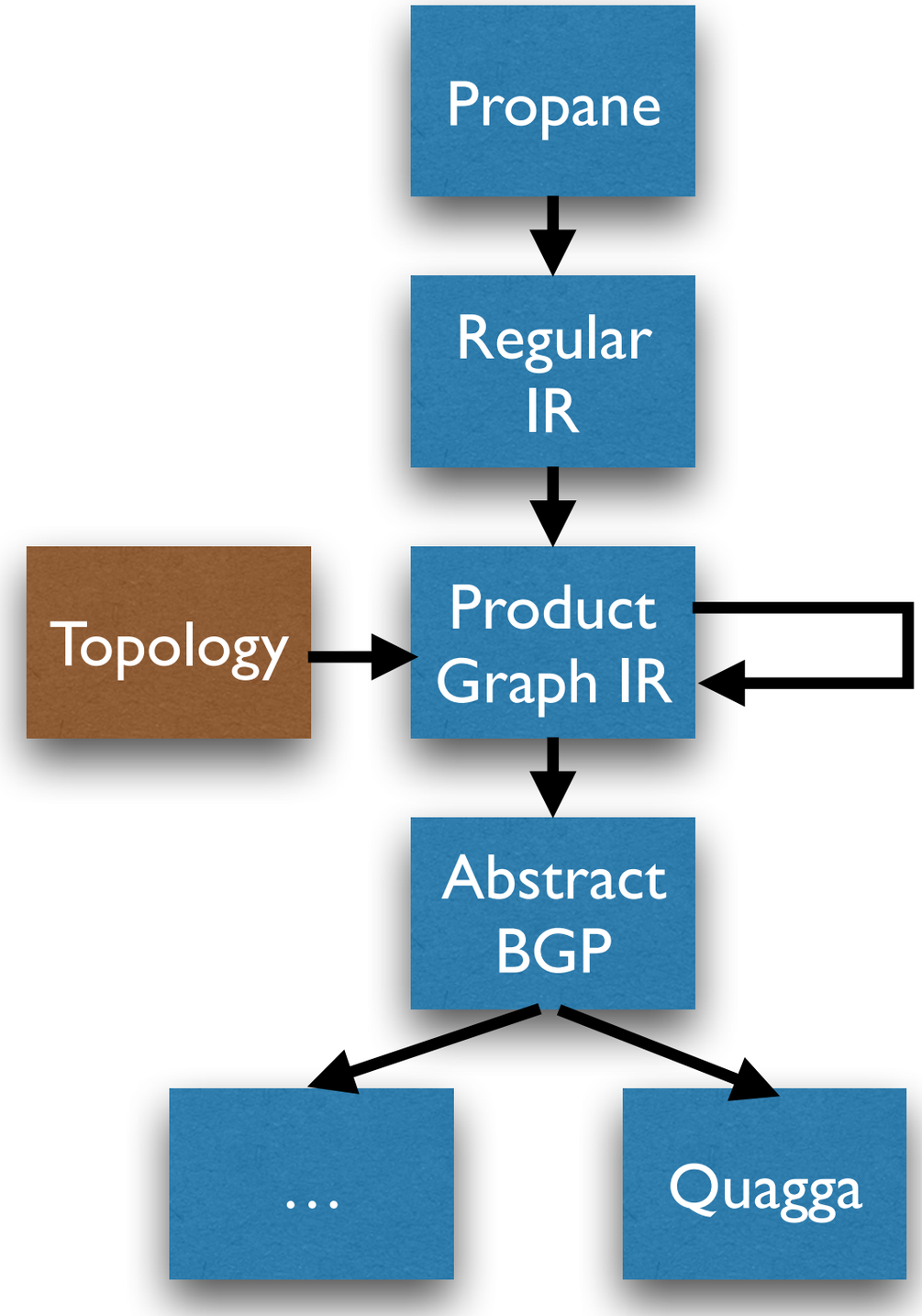
Implementation: C prefers D to E and tags accordingly

The implementation always uses the best available path

Implementation (5,500 lines of F#)

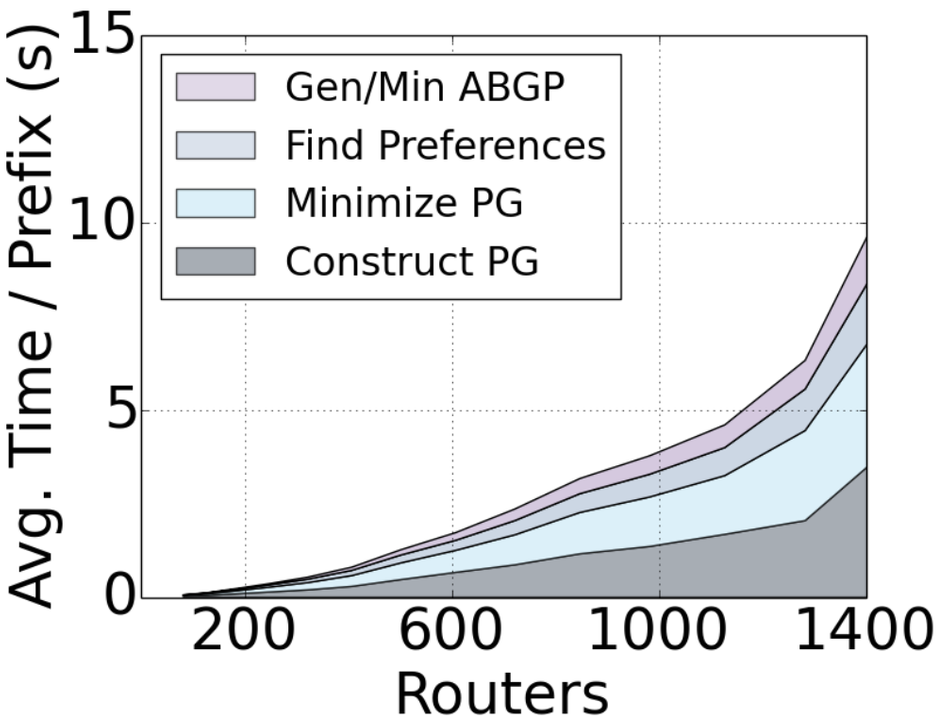
Benchmarks:

- Network configurations from Microsoft's networks (Policy from English docs)
- Data center policies (~1400 routers)
- Backbone policies (~200 routers, many peers/router)
- Ignoring prefix, customer group and ownership definitions:
  - 31 lines for data center
  - 43 lines for backbone

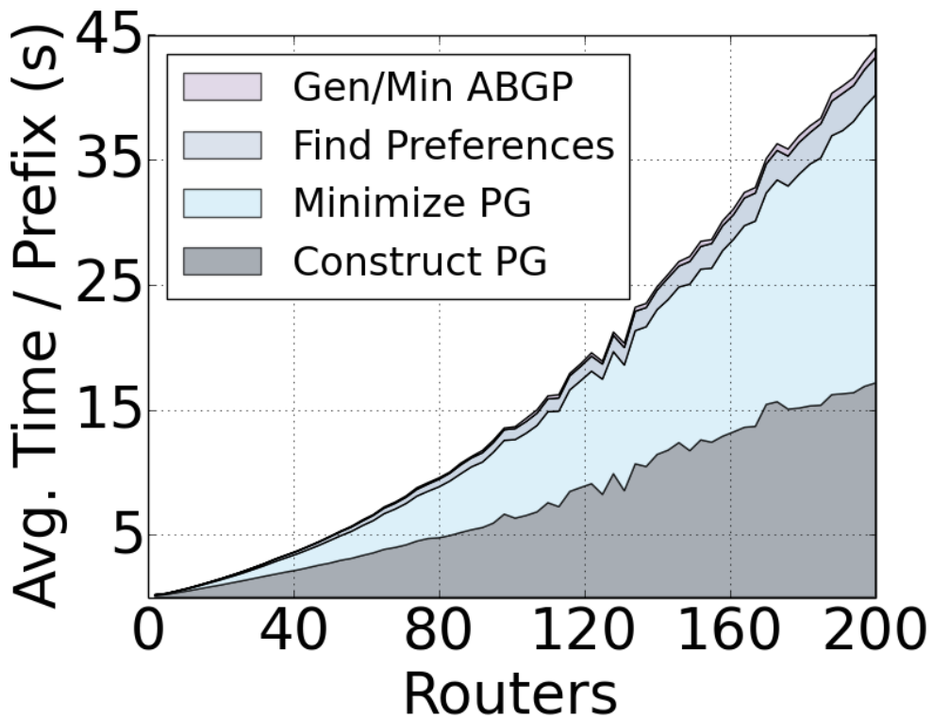


Compilation Time

- Parallelize compilation by prefix
- Use artificial topology to explore scalability



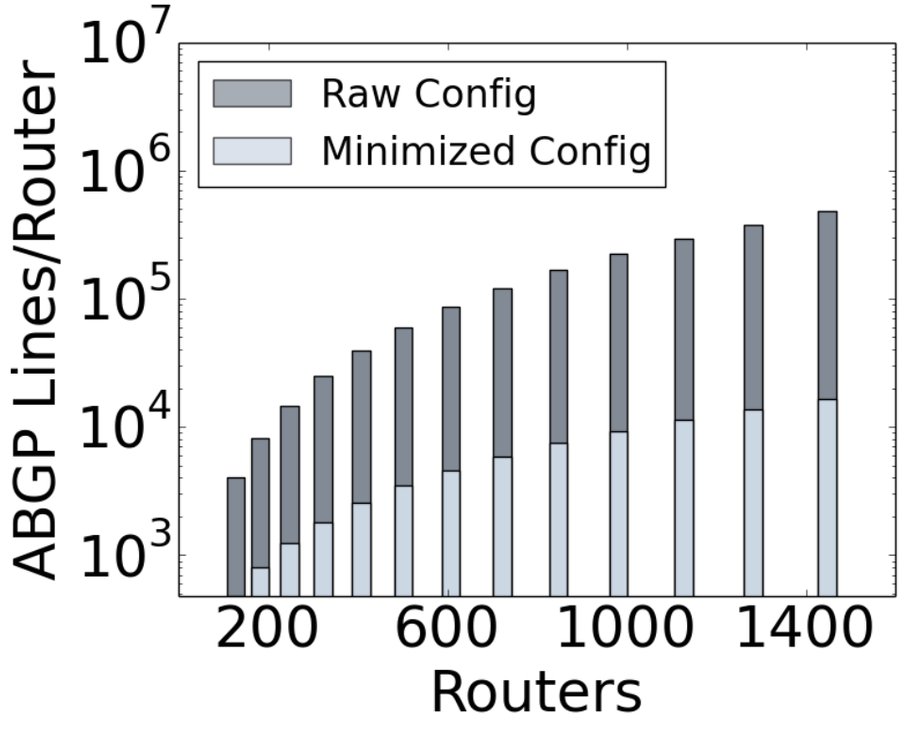
Data center (< 9 min)



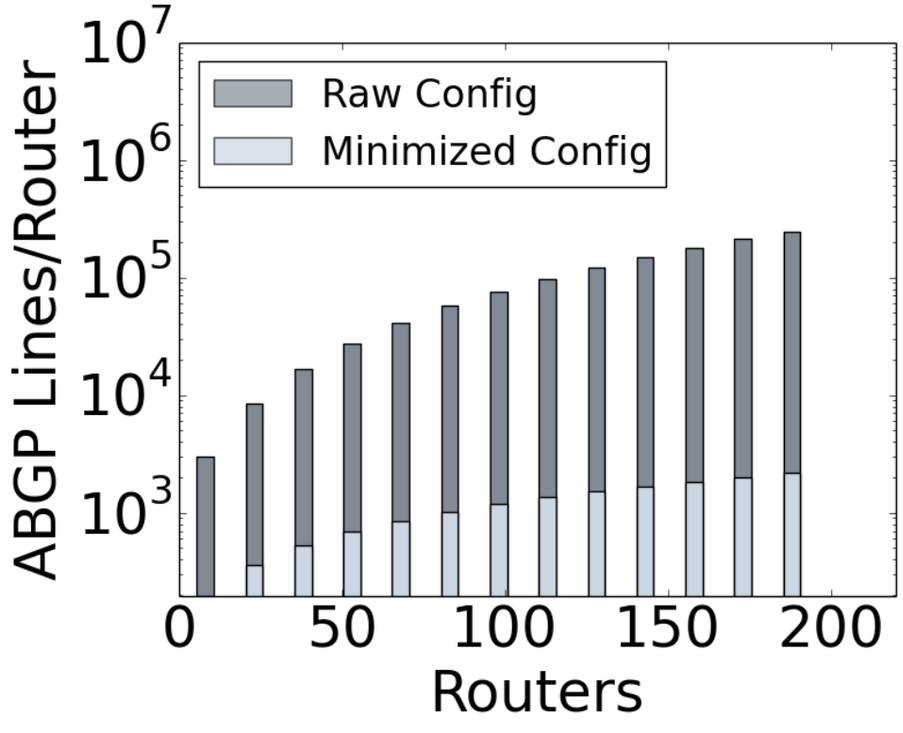
Backbone (< 3 min)

Configuration Size

- Perform configuration minimization during generation
- Avoid using community tags when choices are unambiguous
- Fall-through elimination of route maps



Data center



Backbone