Bridging centralized programming and distributed control planes

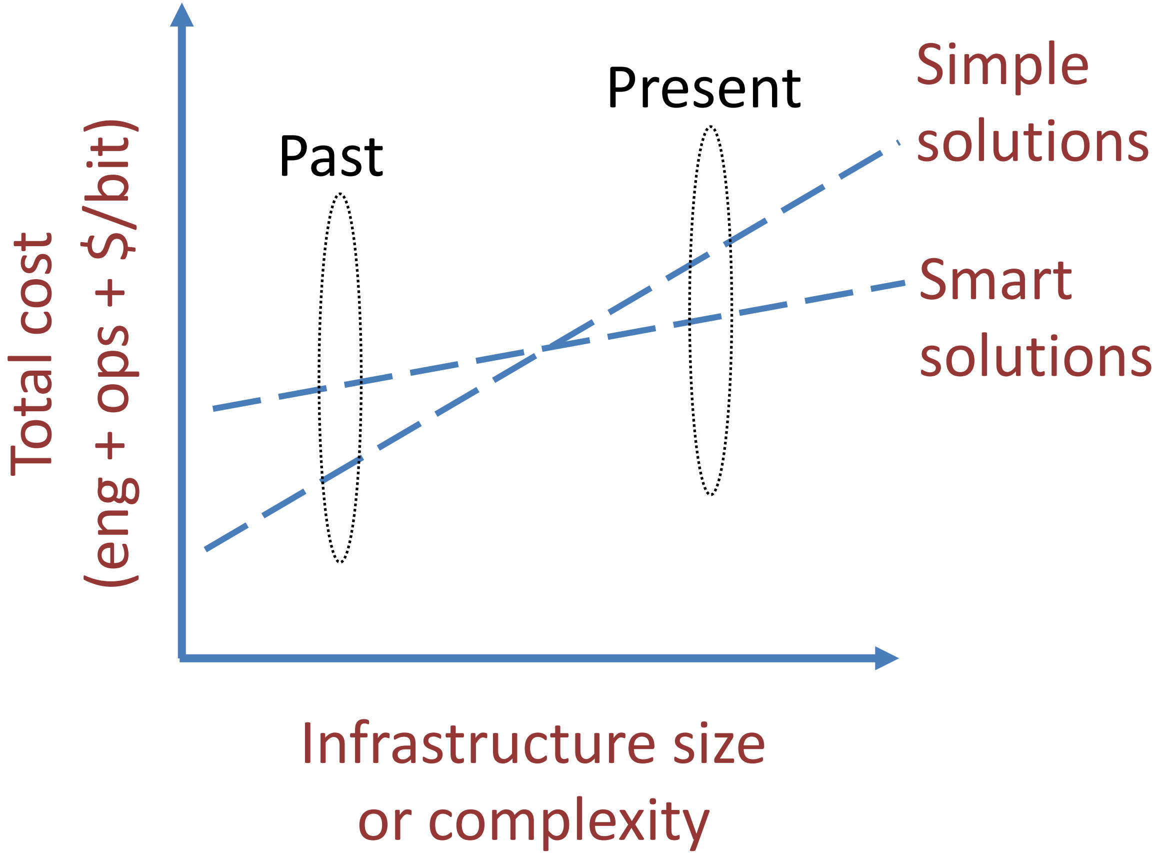




Ryan Beckett Ratul Mahajan Todd Millstein Jitu Padhye David Walker

Microsoft PRINCETON UNIVERSITY UCLA

Hypothesis on smart solutions in networks



Overprovision, best effort,
optimize mean perf, relaxed (distributed) control, ….

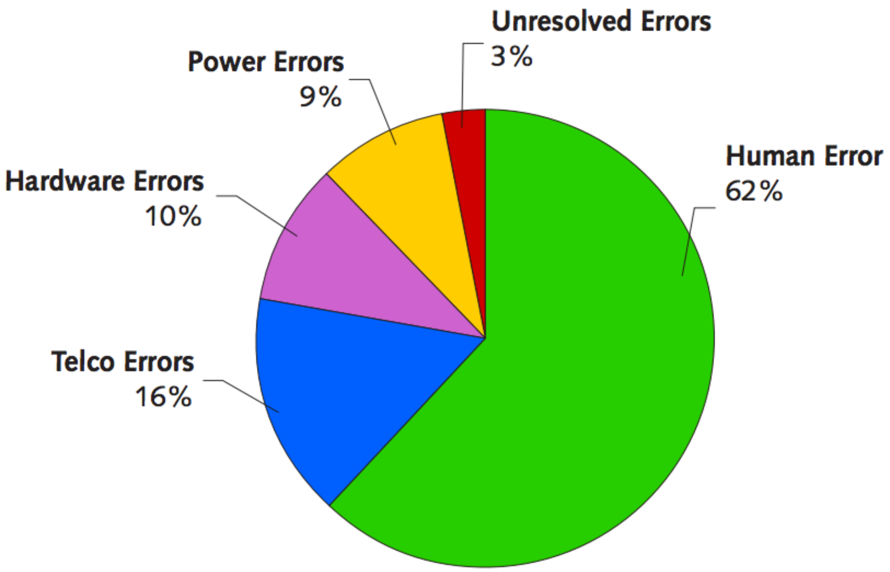
Intelligent resource allocation, multiple priorities,
optimize tail perf, tight (centralized) control, …

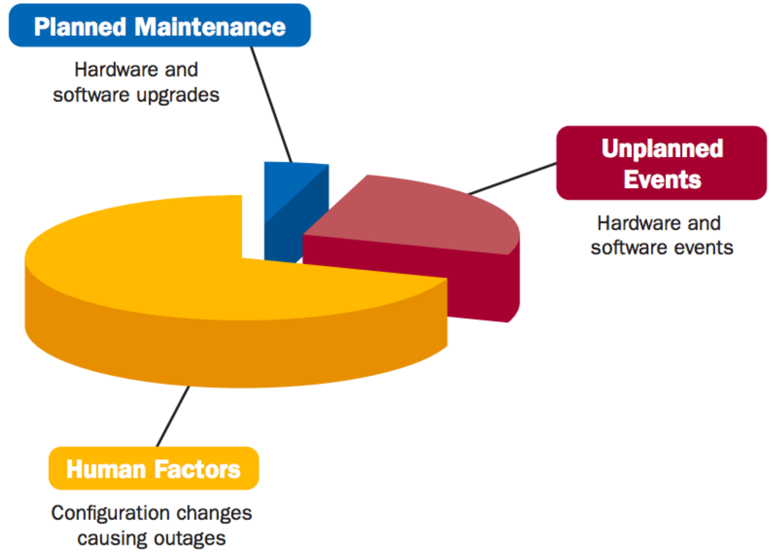
Network programming journey

	Distributed programming	Centralized programming
Distributed control plane	+ Resilience – Programmability	
Centralized Control plane		

Programming (configuring) networks is error-prone

Programming (configuring) networks is error-prone





60% of network downtime is caused by human error

-Yankee group 2002

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

-Juniper 2008

Network programming journey

	Distributed programming	Centralized programming
Distributed control plane	+ Resilience – Programmability	+ Resilience + Programmability
Centralized Control plane		– Resilience + Programmability

Network verification

Programming distributed control planes is hard

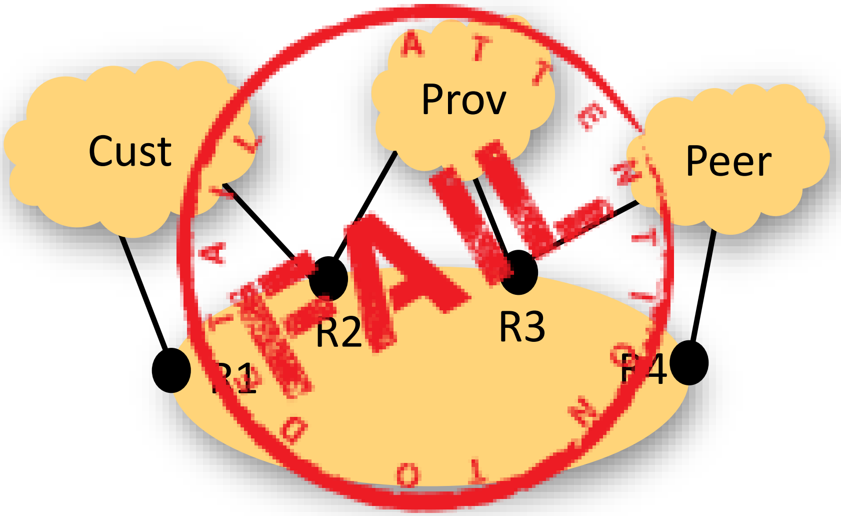
Network-wide policies

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

MIND THE GAP

Router-level mechanisms

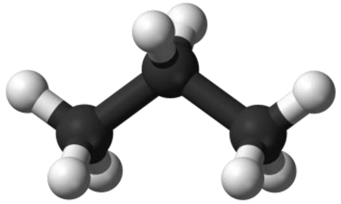
Set consistent, per-link preferences
Tag incoming routing info
Program import and export filters
based on various route attributes



Propane: Centrally programming distributed control planes

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

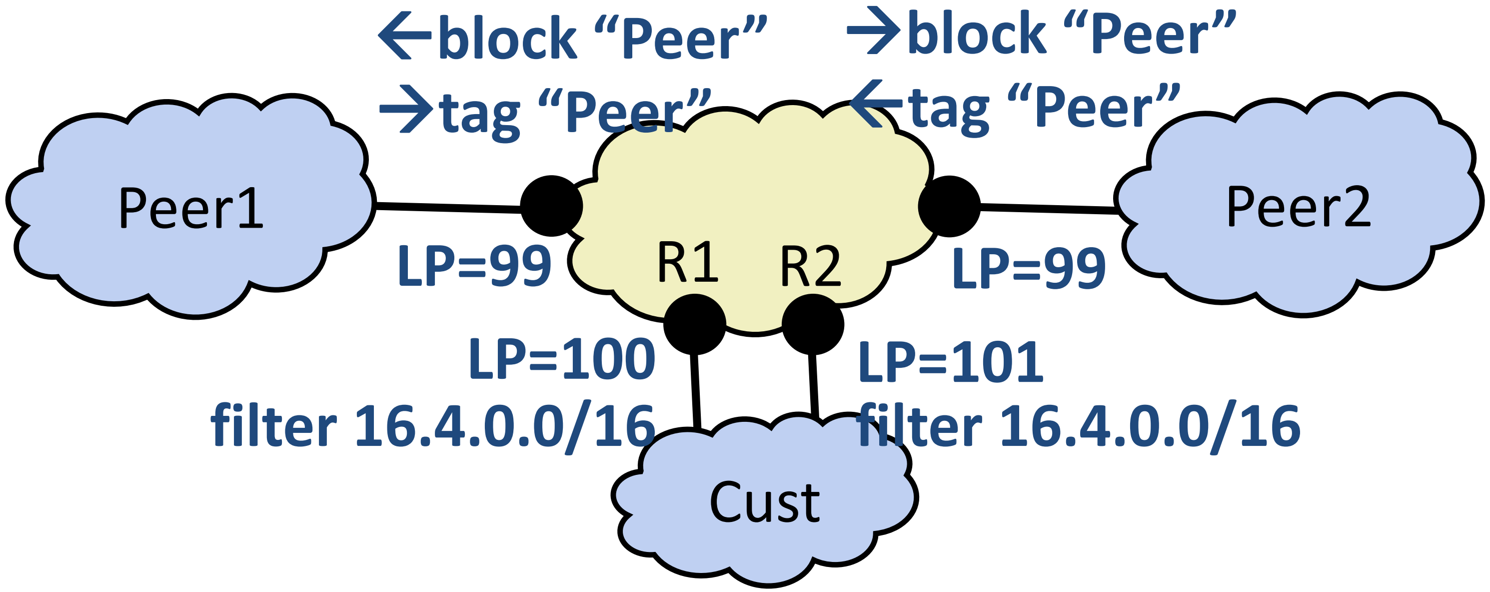
- Path constraints and relative preferences (fallbacks)



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

- Configurations are policy-compliant under all failures

Example #1: A backbone network



Goals

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

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

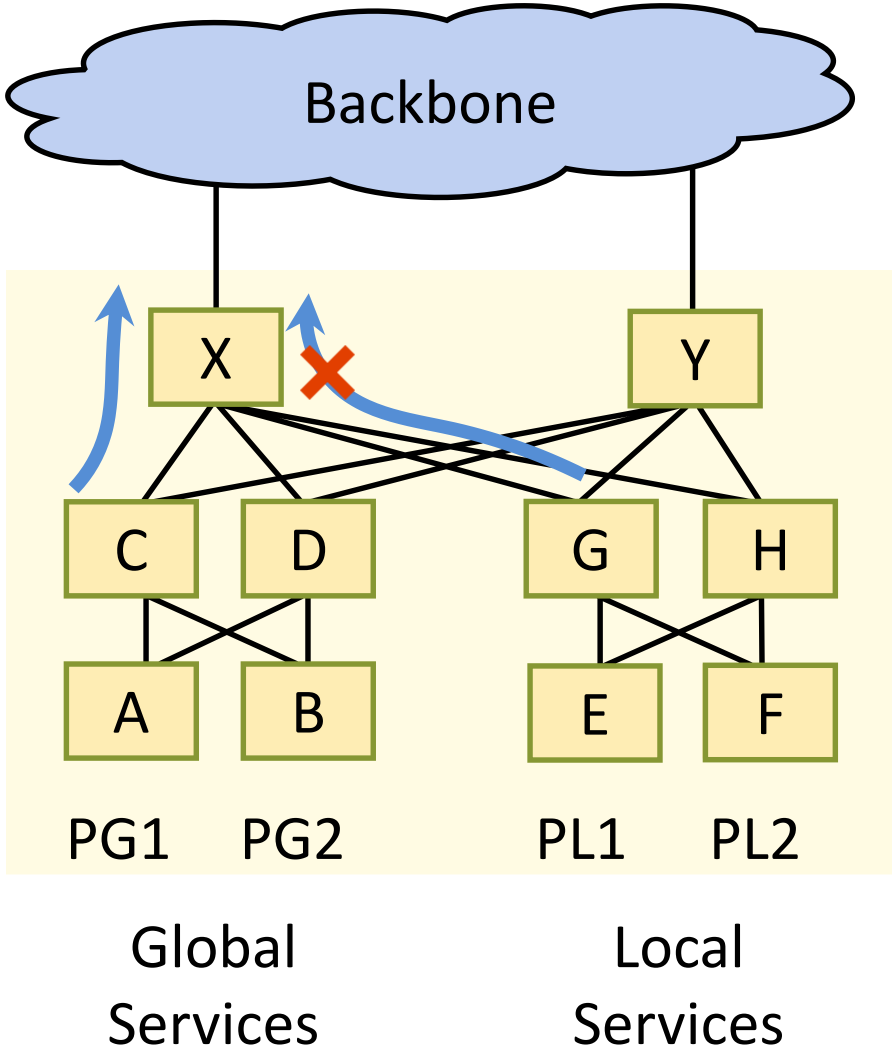
Example #2: A data center network

Goals

- Keep local prefixes internal
- Aggregate global prefixes as PG

Attempt #1

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



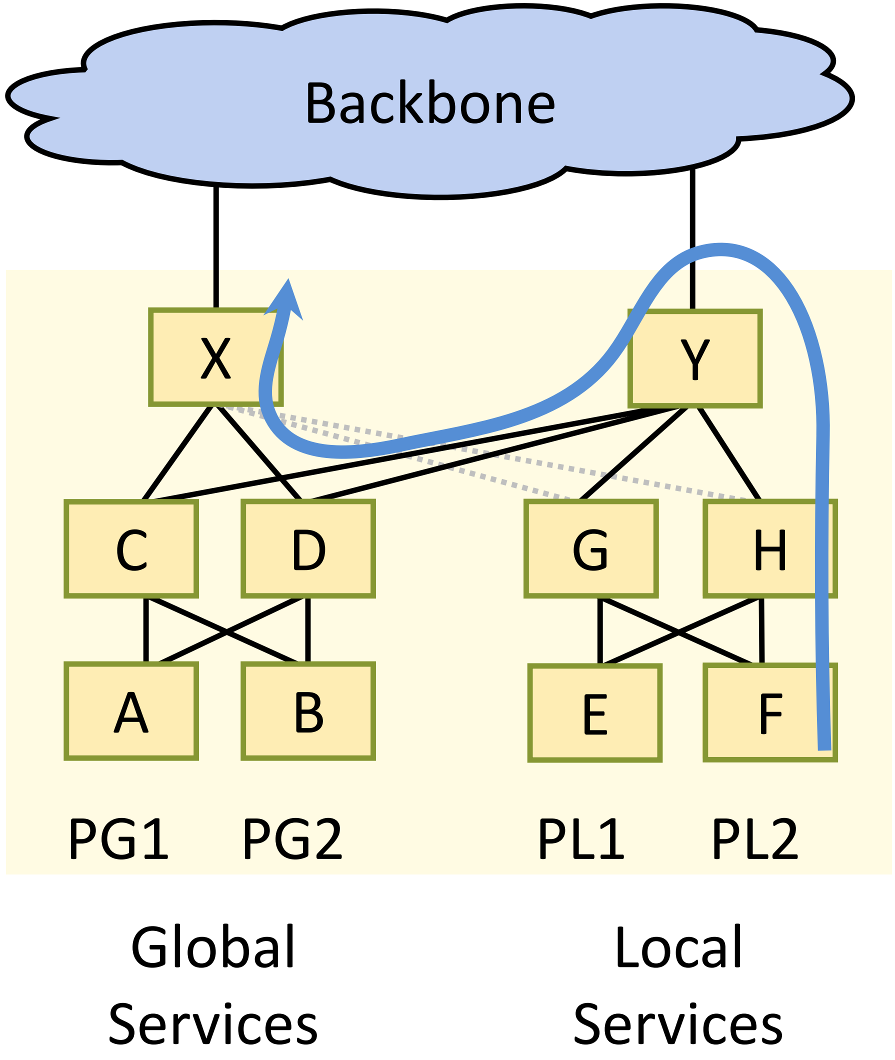
Example #2: A data center network

Goals

- Keep local prefixes internal
- Aggregate global prefixes as PG

Attempt #1

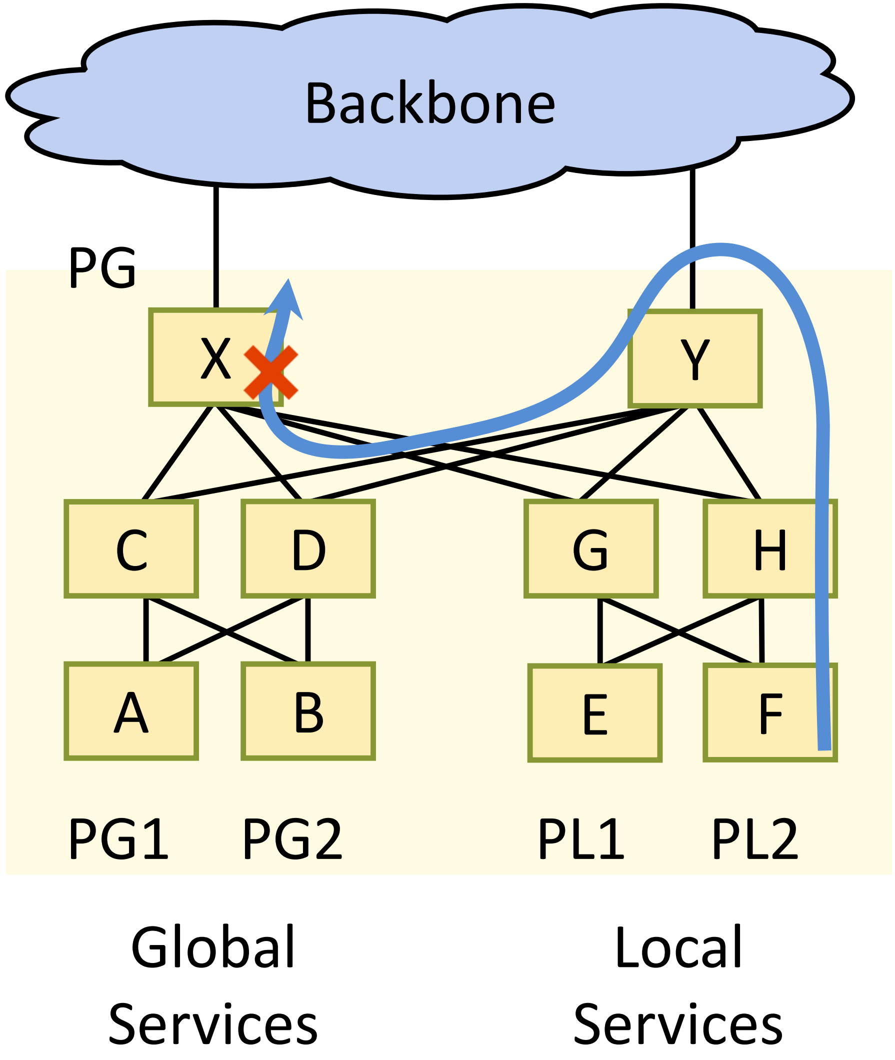
- Don't export from G, H to external
- Aggregate externally as PG



Example #2: A data center network

Goals
- Keep local prefixes internal
- Aggregate global prefixes as PG

Attempt #2
- Don't export from G, H to external
- Aggregate externally as PG
- Valley-free routing



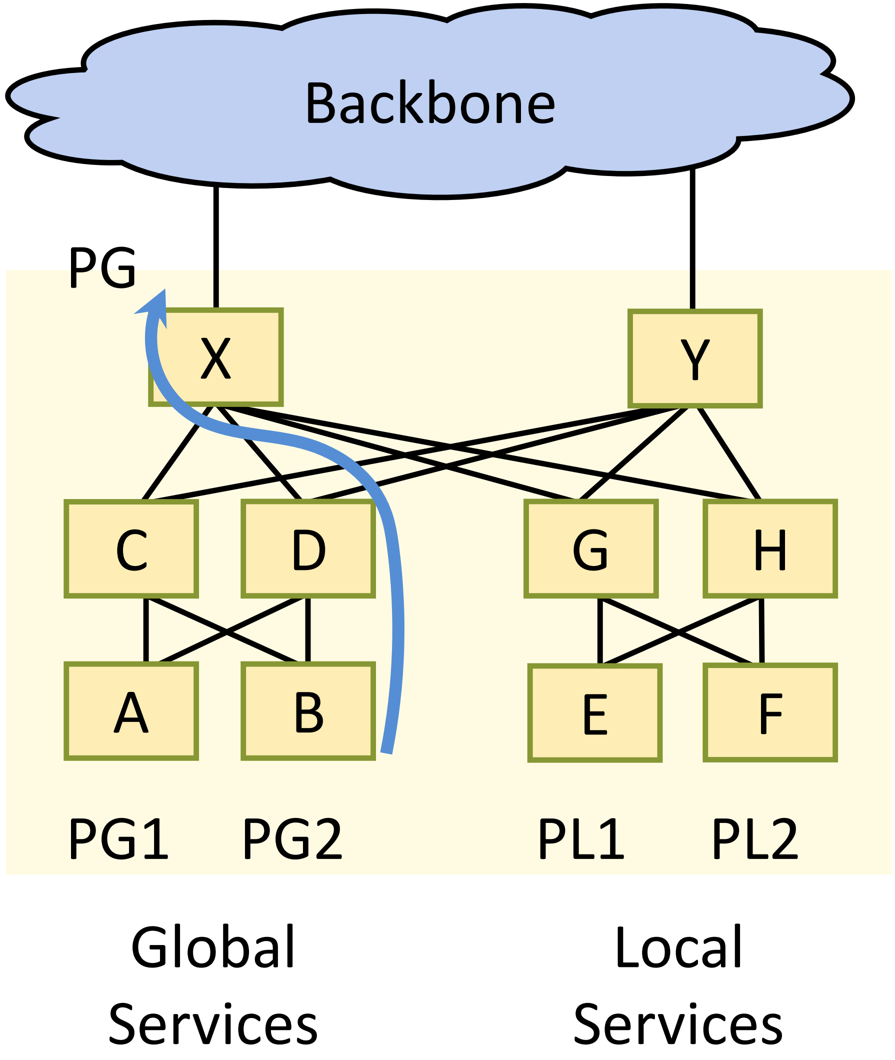
Example #2: A data center network

Goals

- Keep local prefixes internal
- Aggregate global prefixes as PG

Attempt #2

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



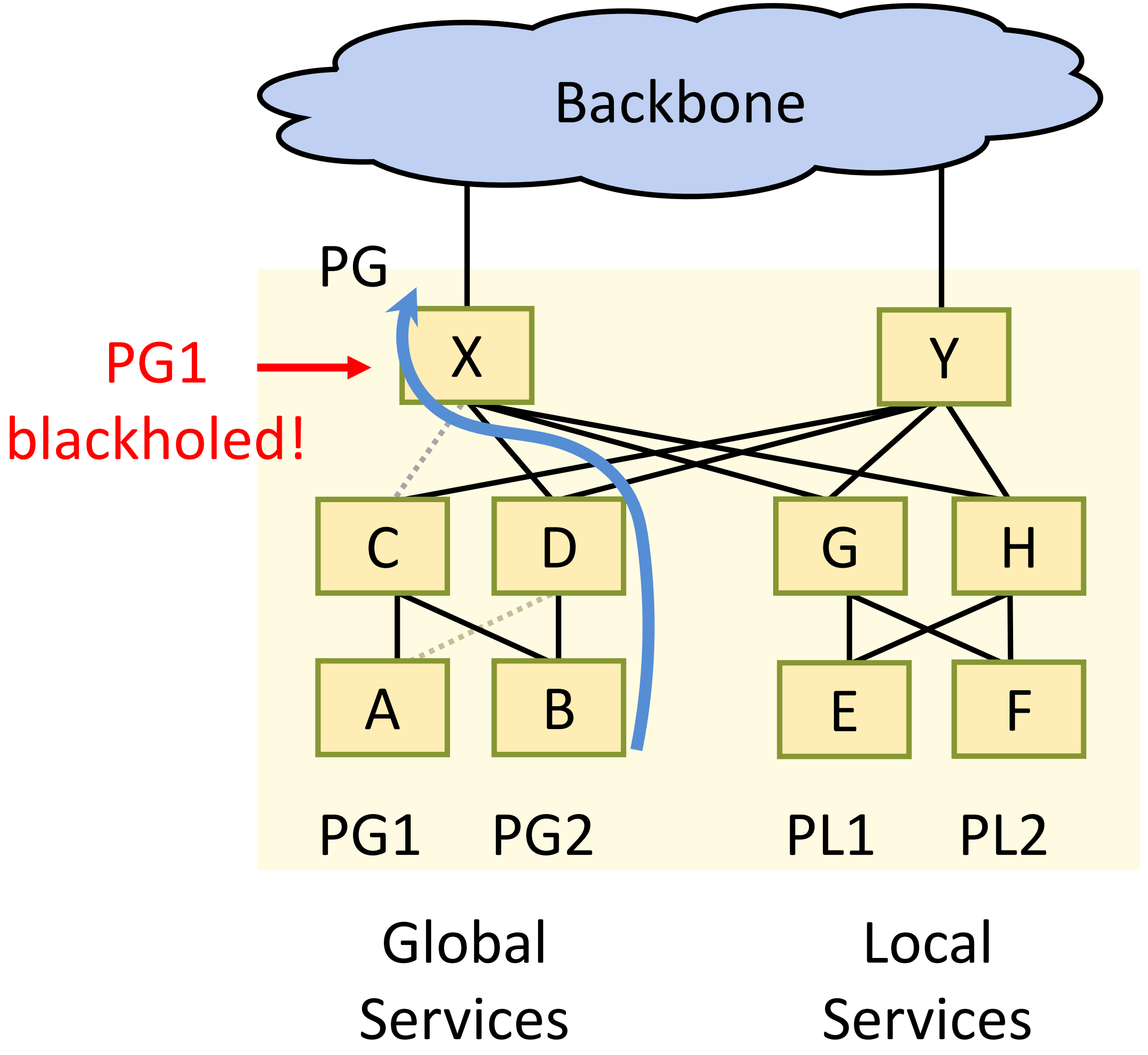
Example #2: A data center network

Goals

- Keep local prefixes internal
- Aggregate global prefixes as PG

Attempt #2

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

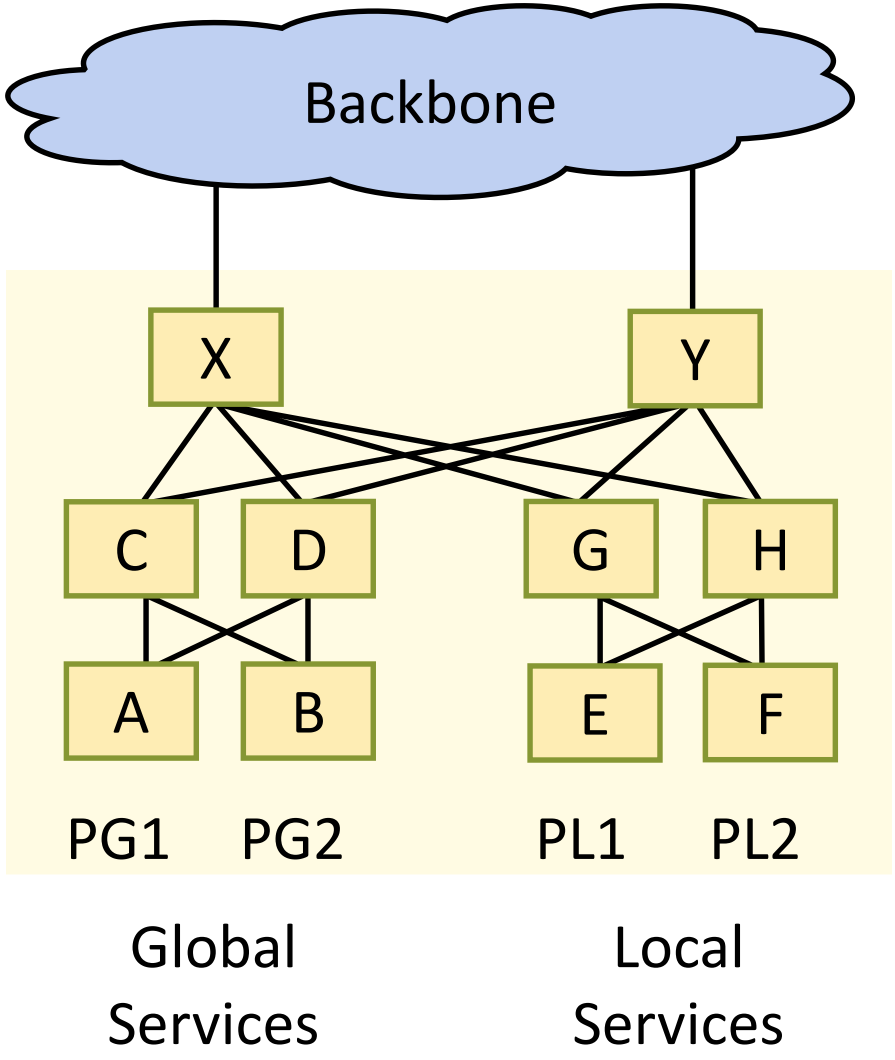


Example #2: A data center network

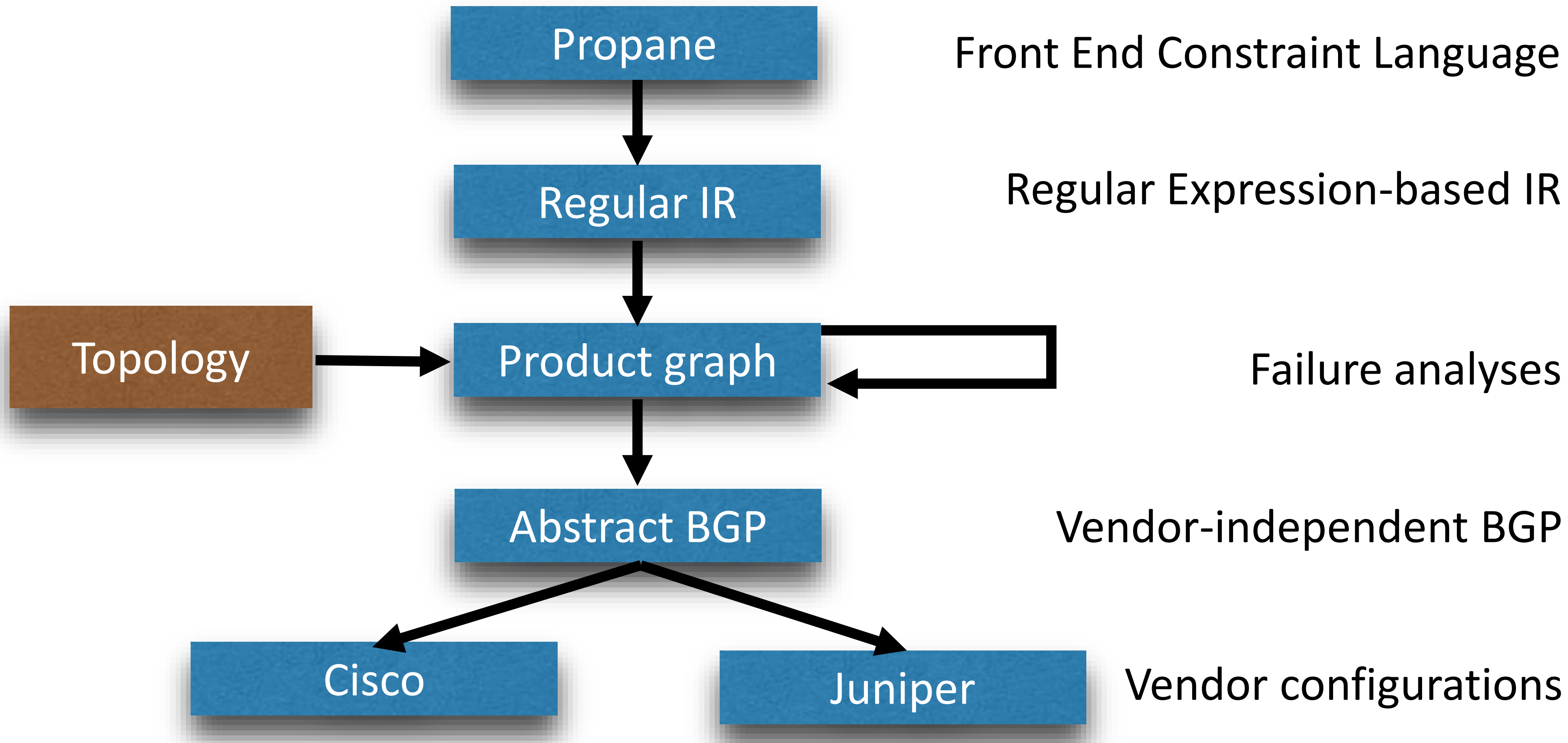
Goals

- Keep local prefixes internal
- Aggregate global prefixes as PG

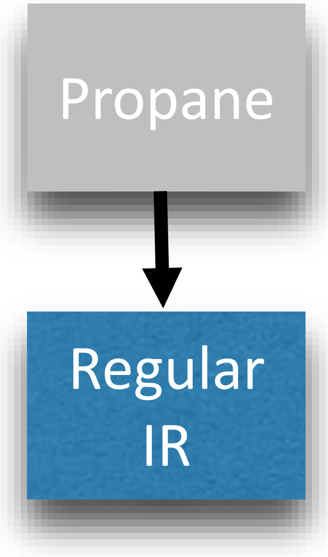
```
define ownership = {PG1 => end(A),
                    PG2 => end(B),
                    PL1 => end(E),
                    PL2 => end(F)}
define locality = { {PL1, PL2} => always(in)}
control {aggregate(PG, in -> out)}
define main = routing and locality
```



Propane compiler



Propane Regular IR

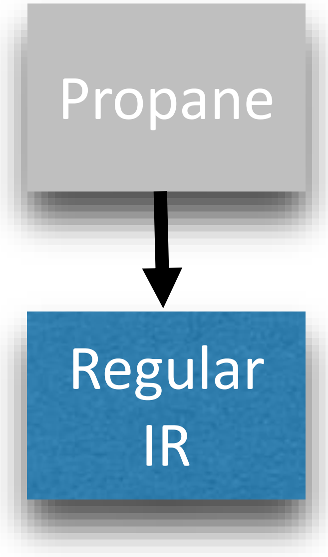


Step 1: Combine modular constraints
Prefix-by-prefix intersection of constraints

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

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

Propane Regular IR



Step 2: Expand constraints in to regular expressions

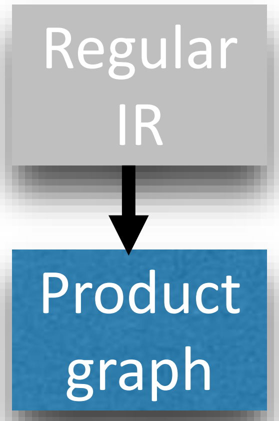
```
          any = out*.in+.out*
       end(X) = (Σ*.X)
    always(X) = (X)*
      exit(X) = (out*.in*.(X ∩ in).out+)|
                (out*.in+.(X ∩ out).out*)
     start(X) = (X.Σ*)
     avoid(X) = (!X)*
  waypoint(X) = (Σ*.X.Σ*)
```

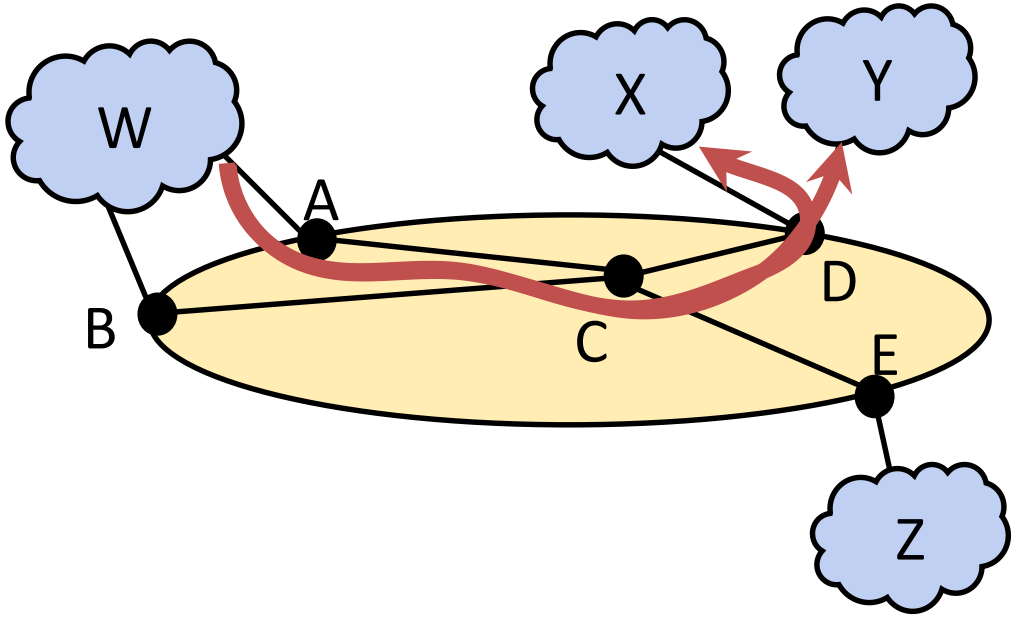
Step 3: Reduced syntax

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

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

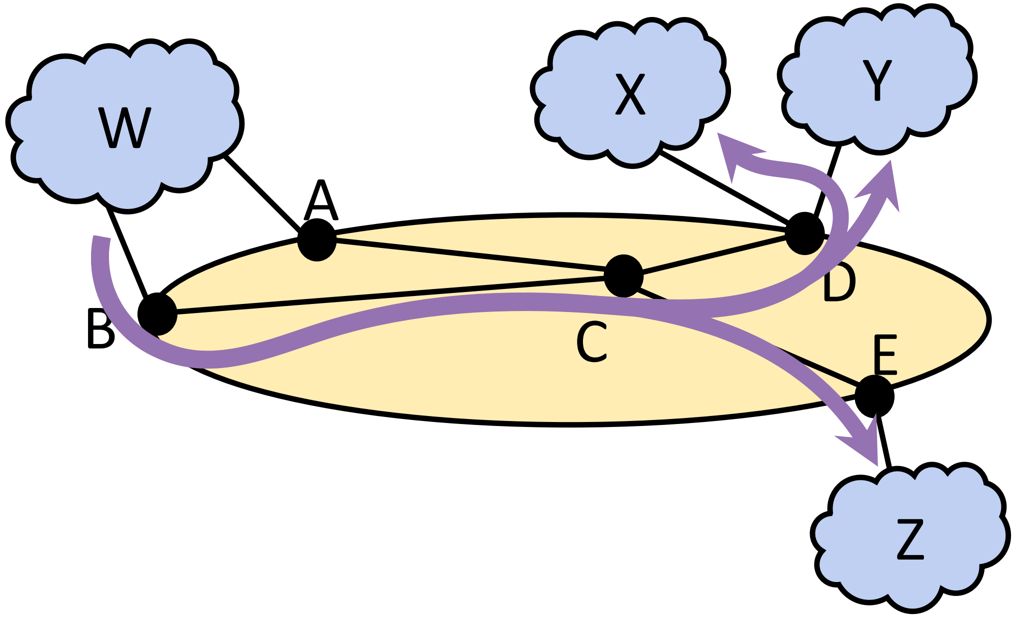
PG construction: An Example

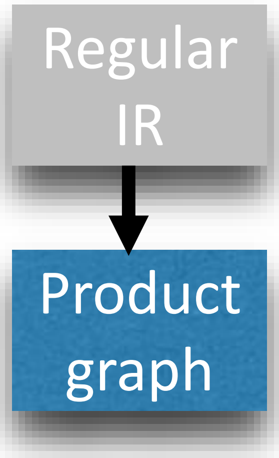




Policy: `(W.A.C.D.out) >> (W.B.in+.out)`

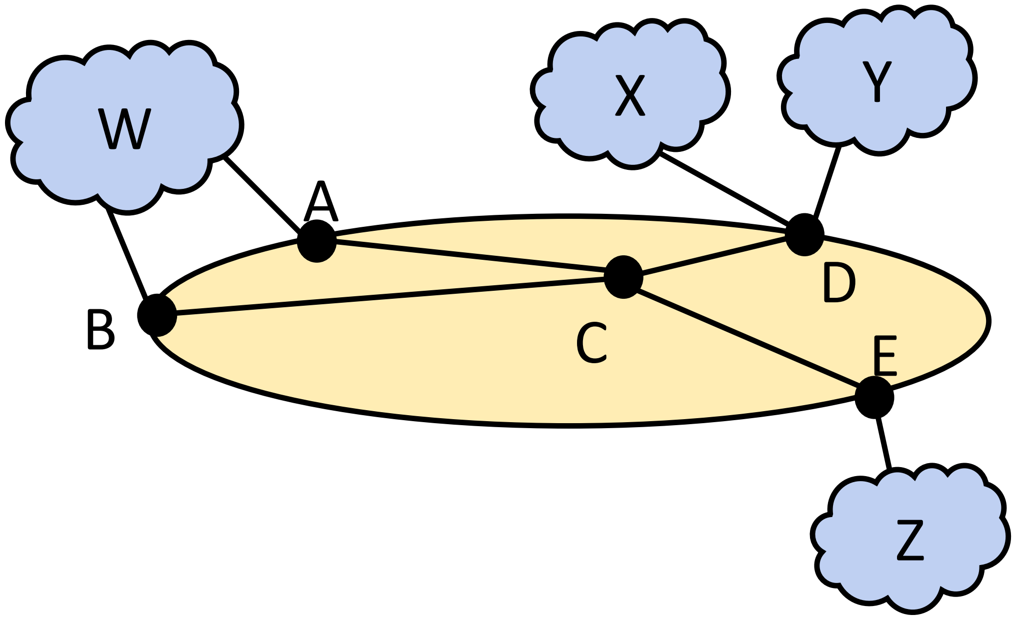
PG construction: An Example



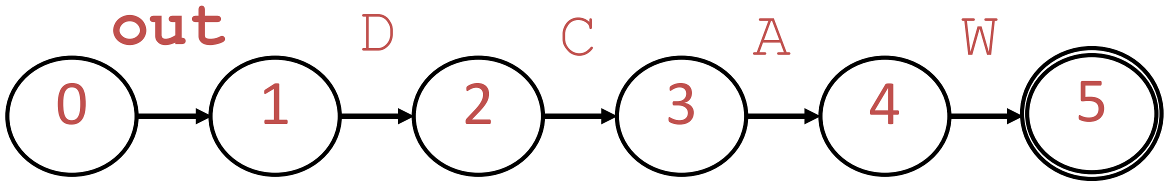


Policy: `(W.A.C.D.out) >> (W.B.in+.out)`

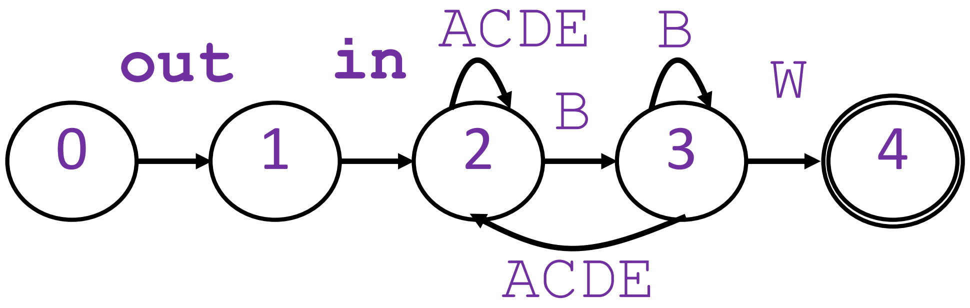
PG construction: Reversed policy automata



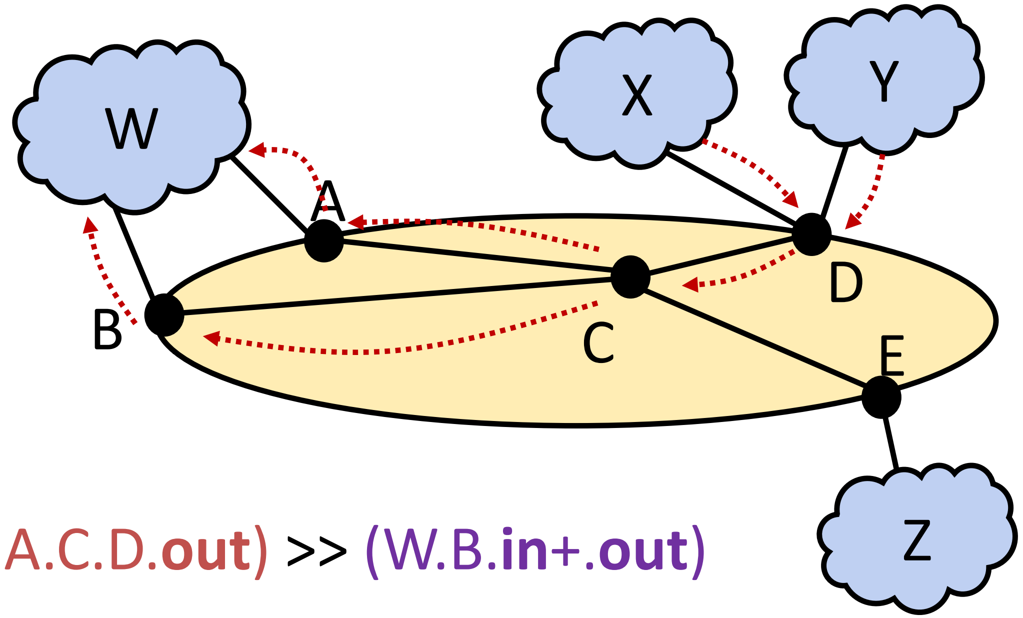
(W.A.C.D.**out**)



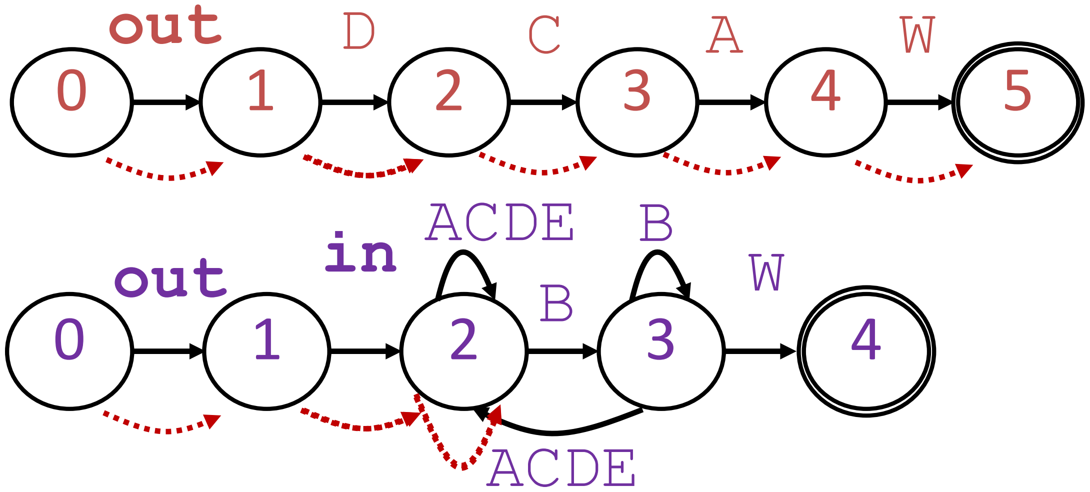
(W.B.**in**+.**out**)

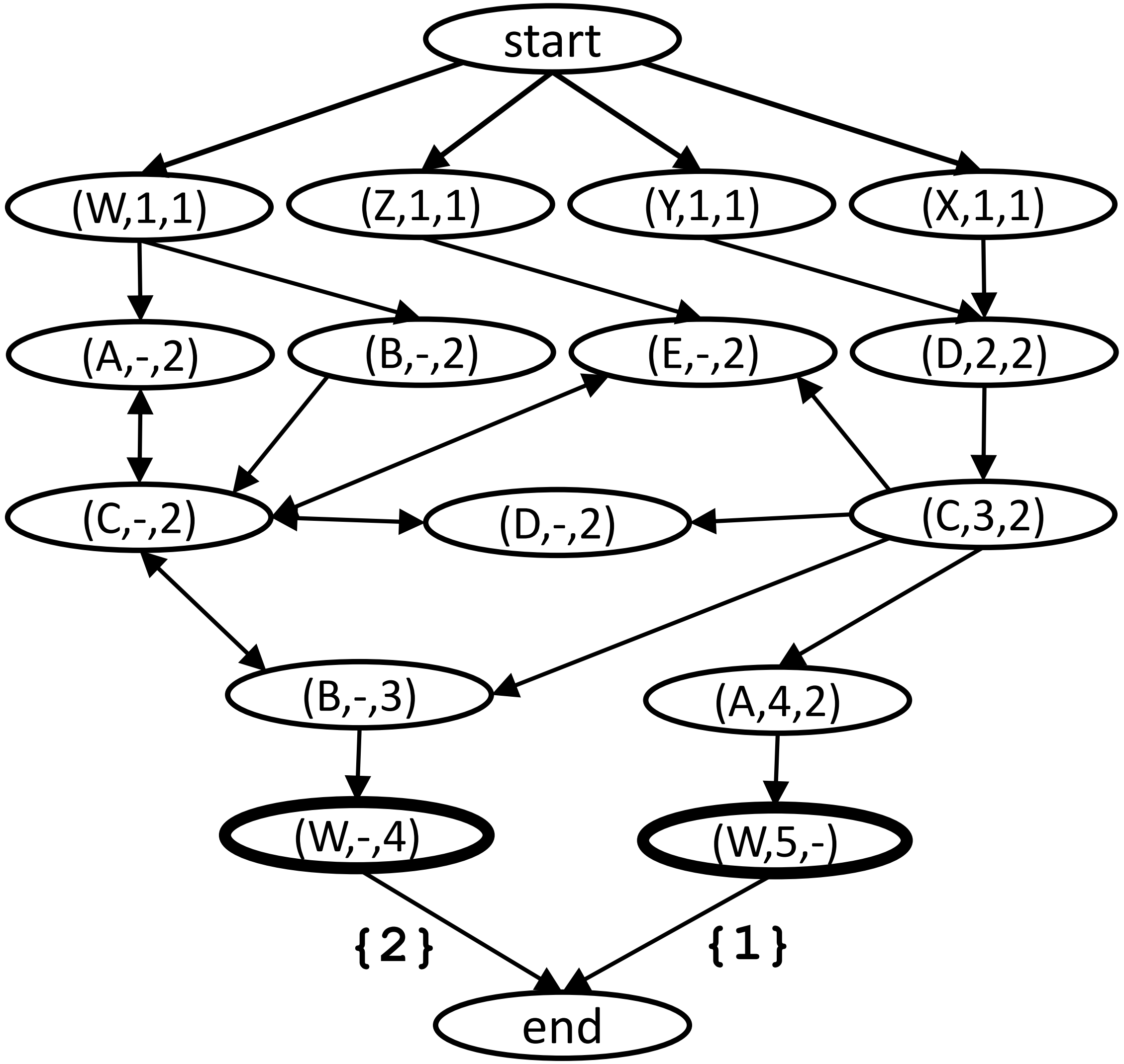


PG construction: Graph generation

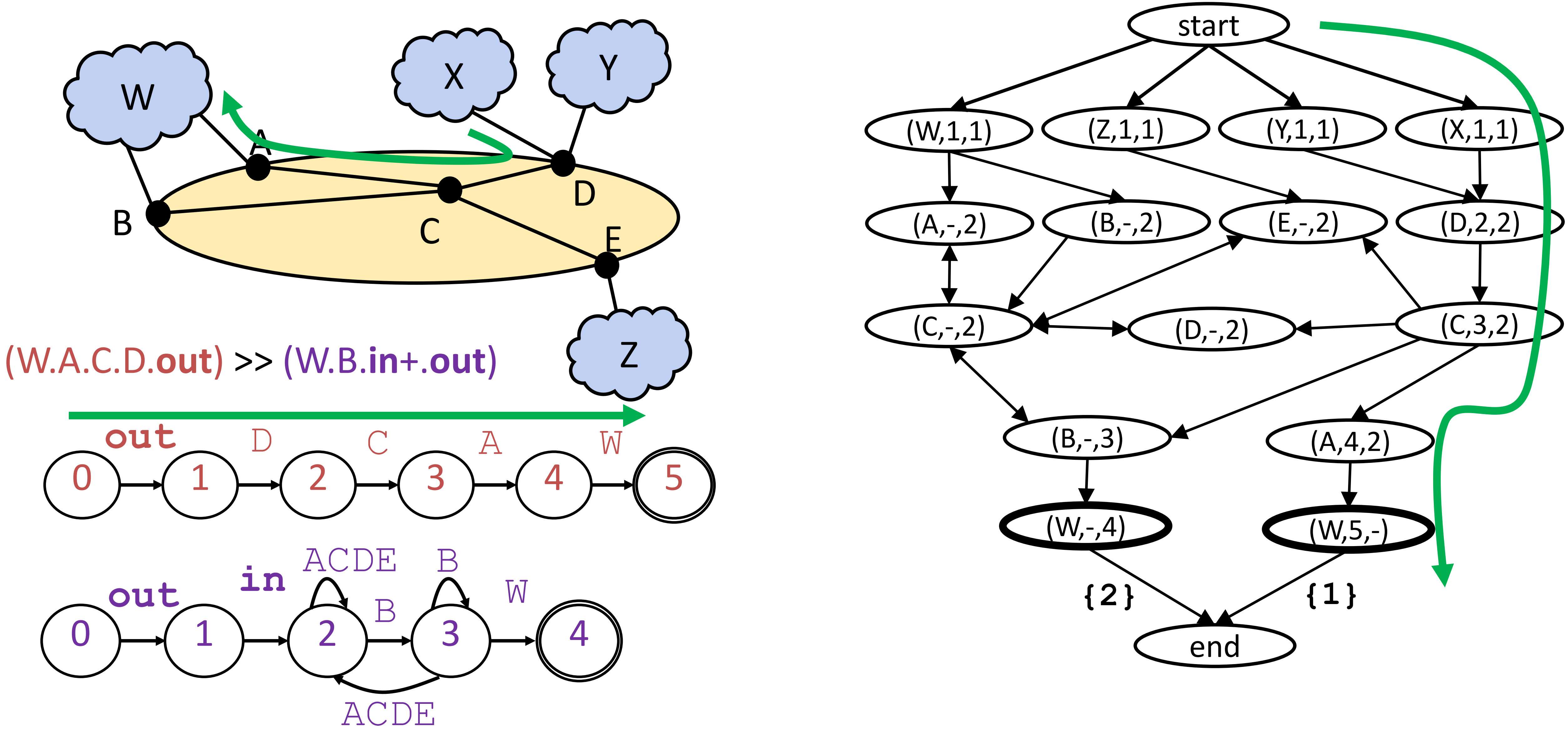


(W.A.C.D.**out**) >> (W.B.**in**+.**out**)

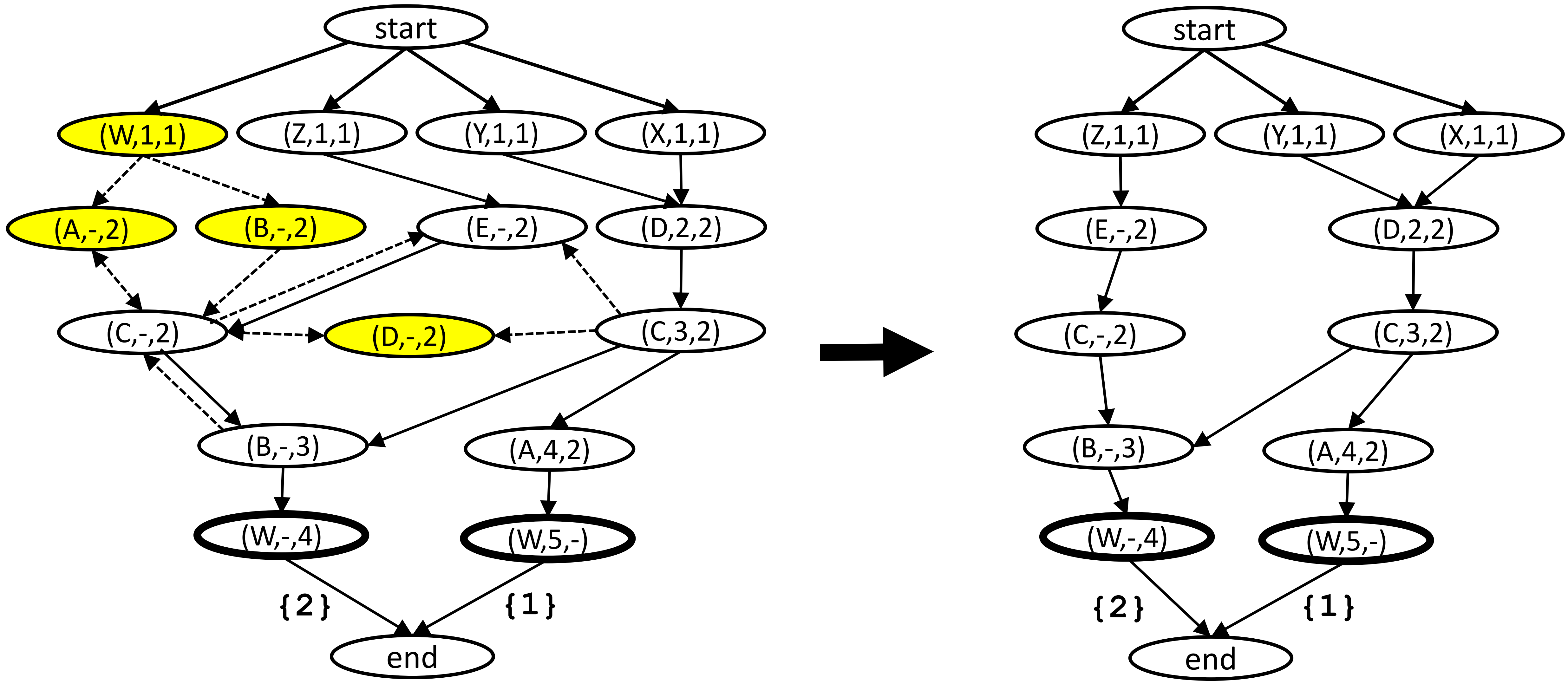




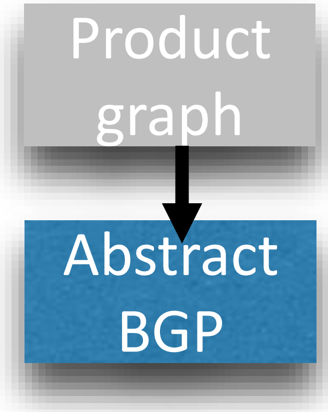
PG construction: Graph generation

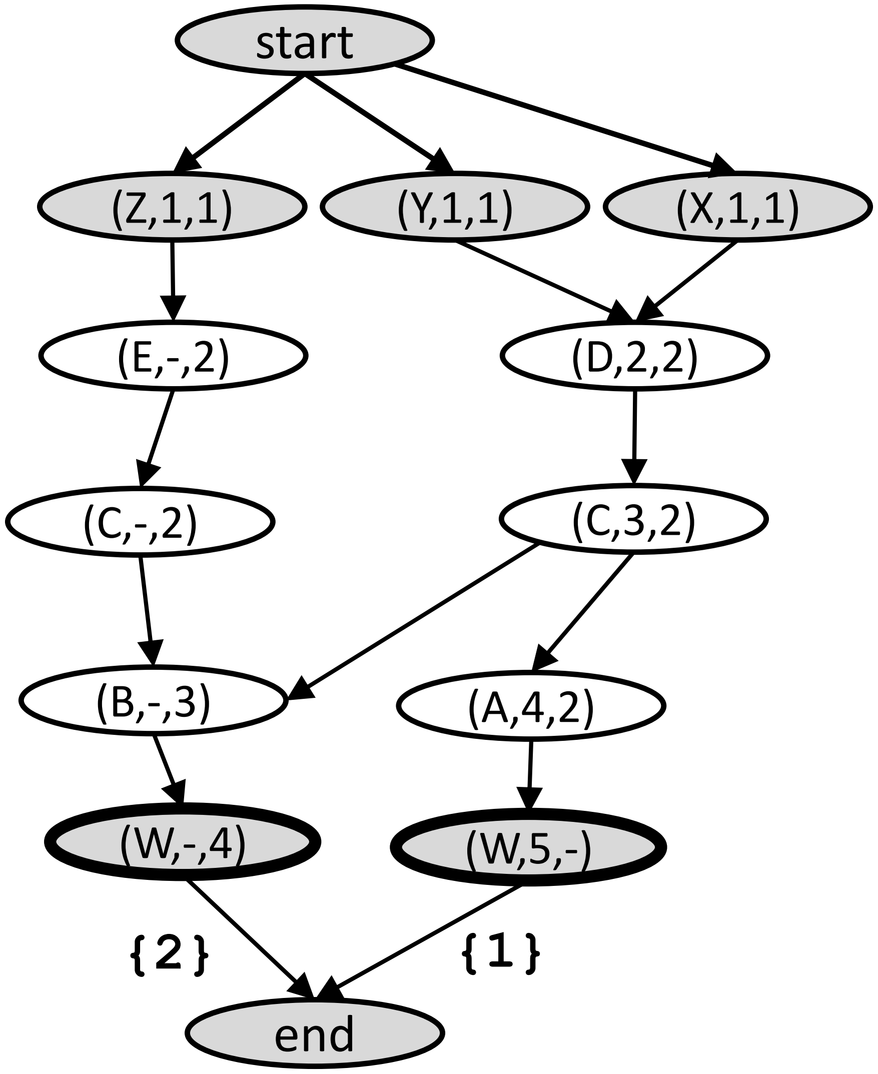


PG construction: minimization (loop analysis)



Compilation to ABGP



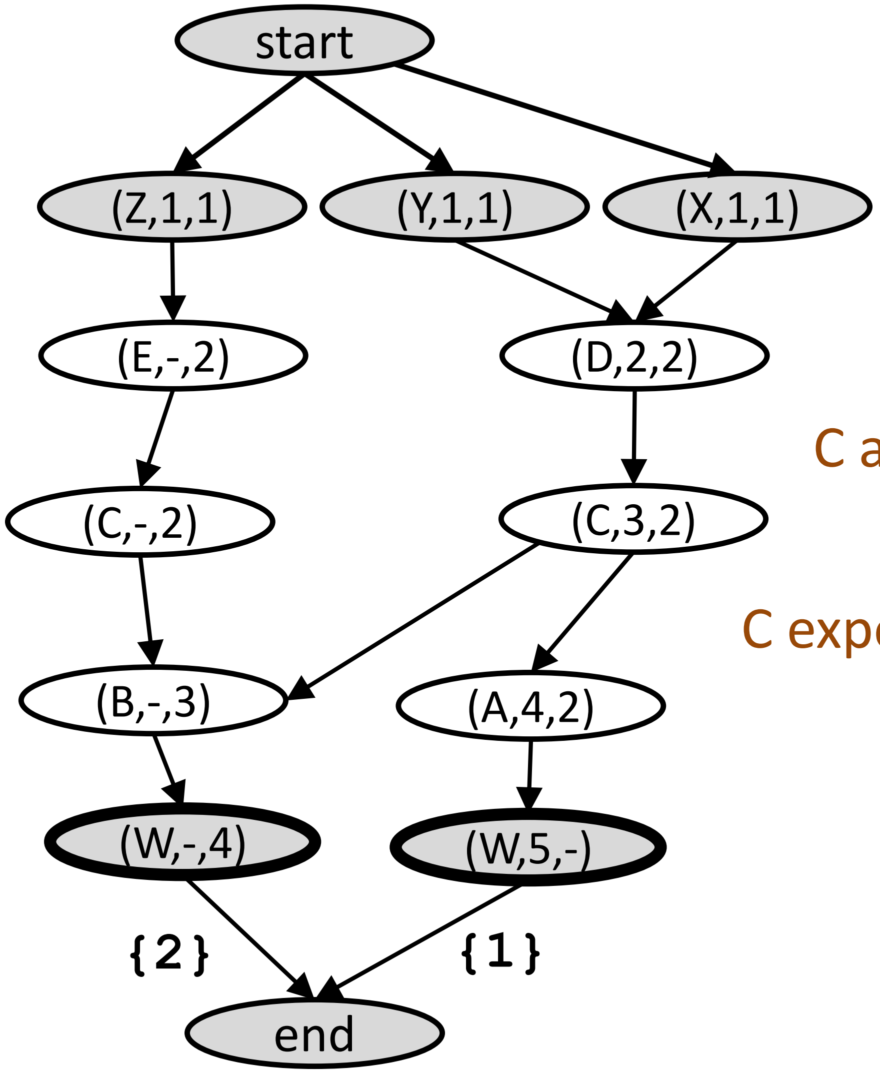


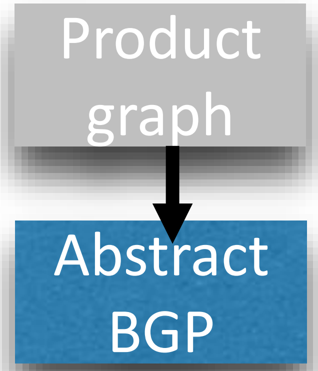
Idea 1: Restrict advertisements to PG edges

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

Let BGP find some allowed path dynamically

Compilation to BGP

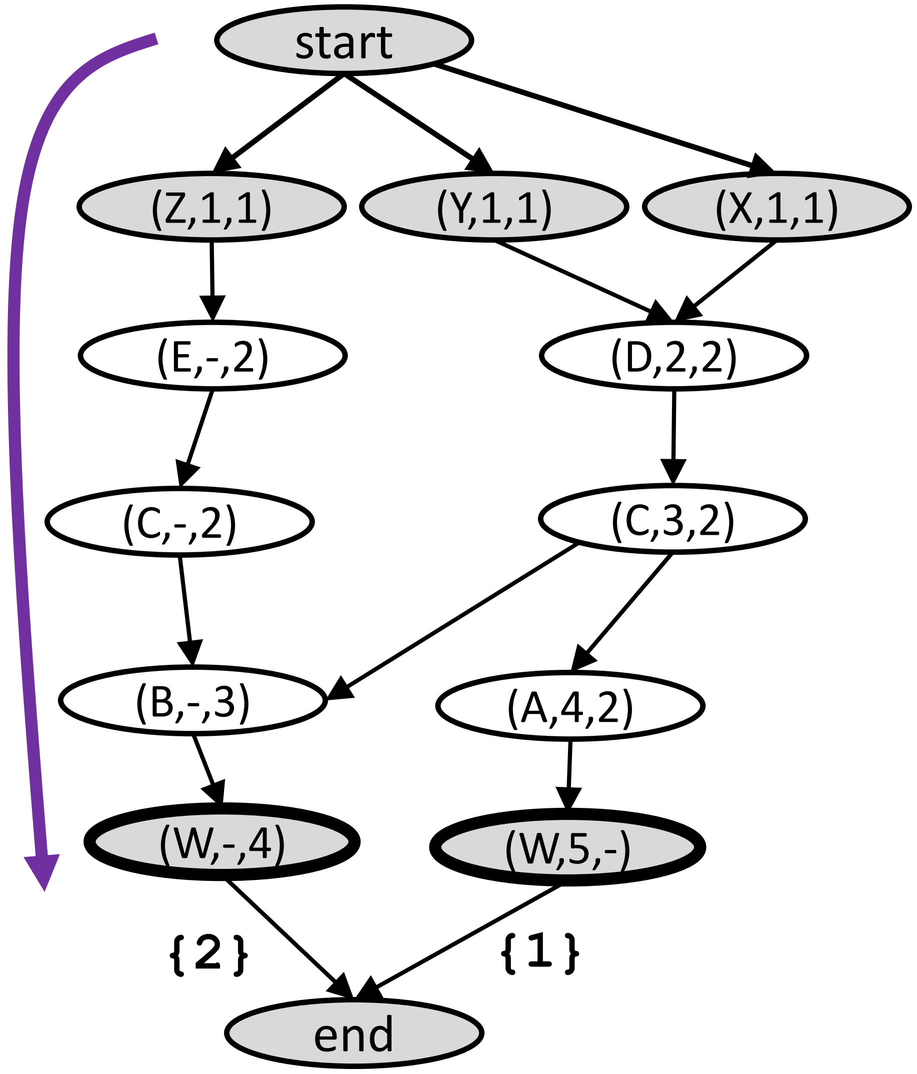




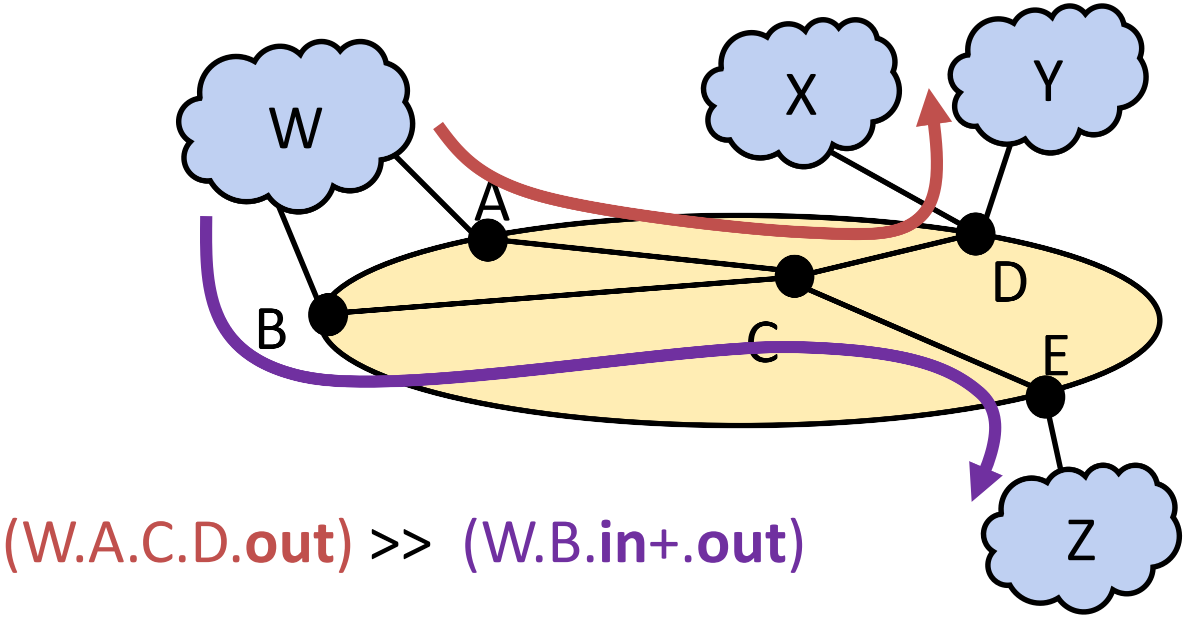
C allows import from D with tag (2,2)

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

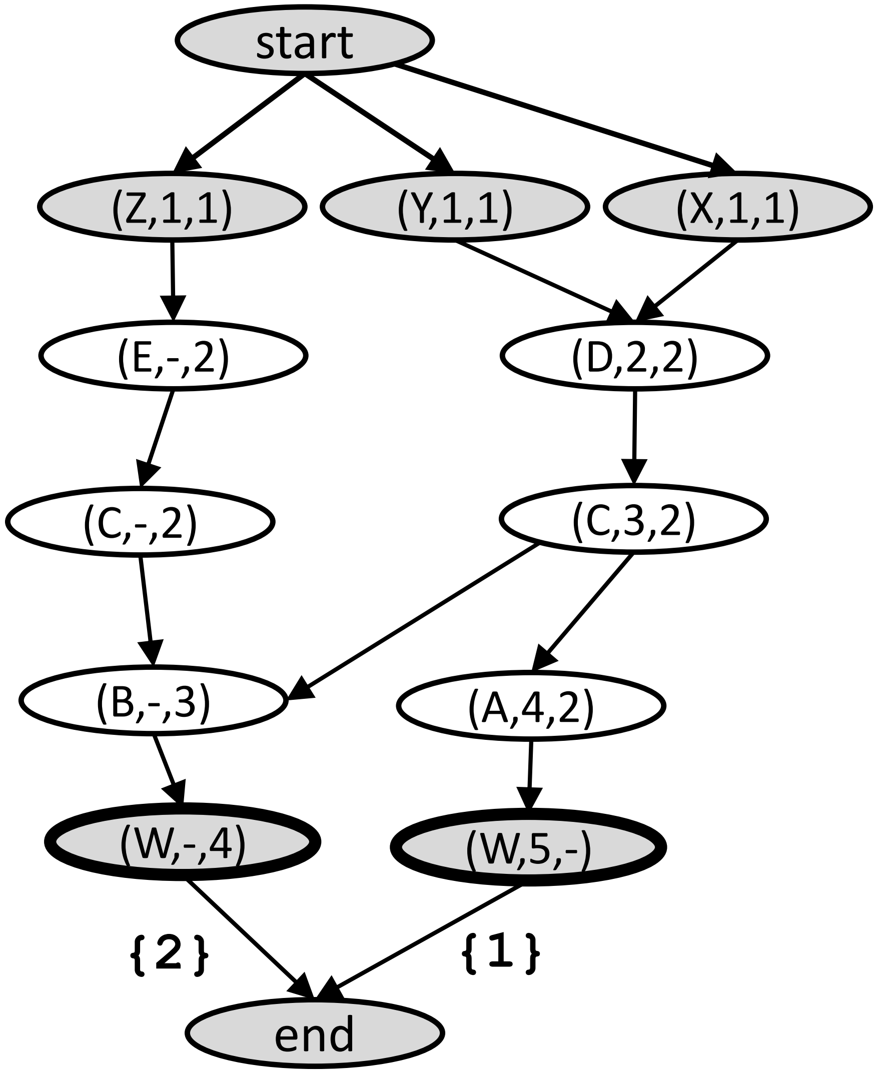
Compilation to BGP



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



Compilation to BGP

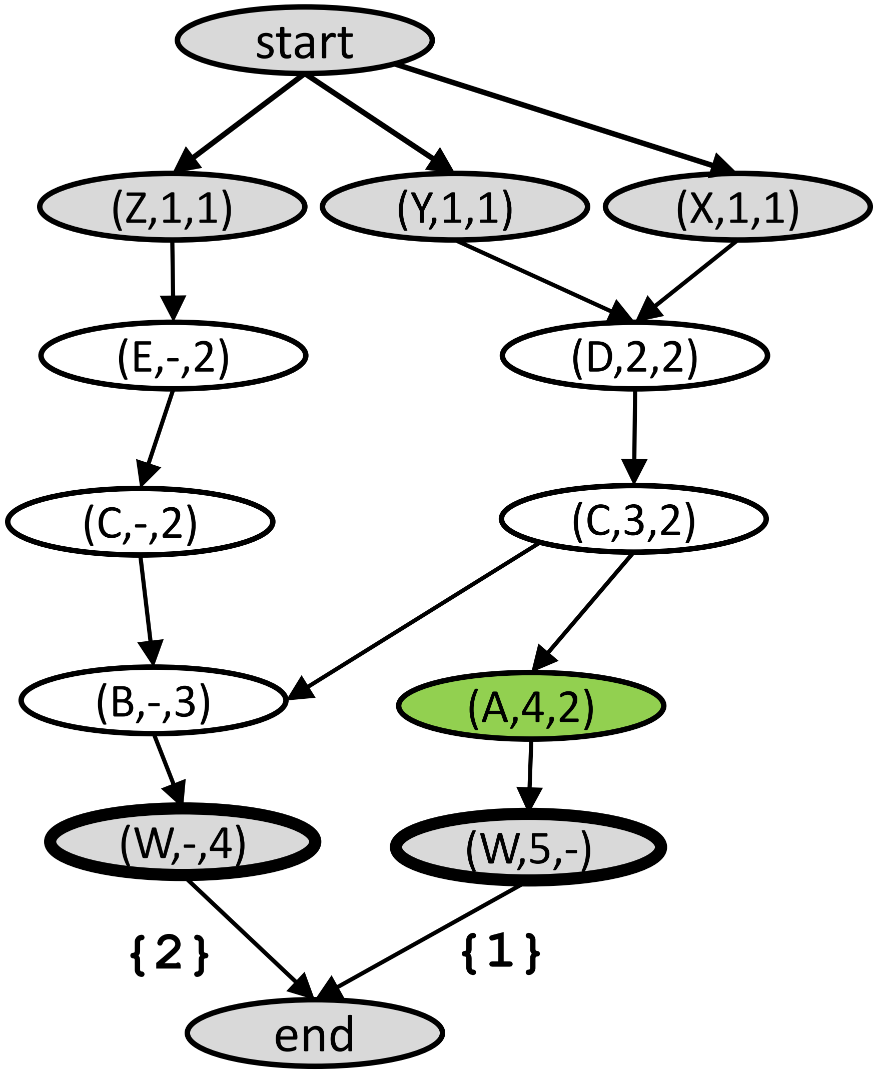


Idea 2: Synthesize local preferences

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

Let BGP find the best allowed path dynamically

Compilation to BGP



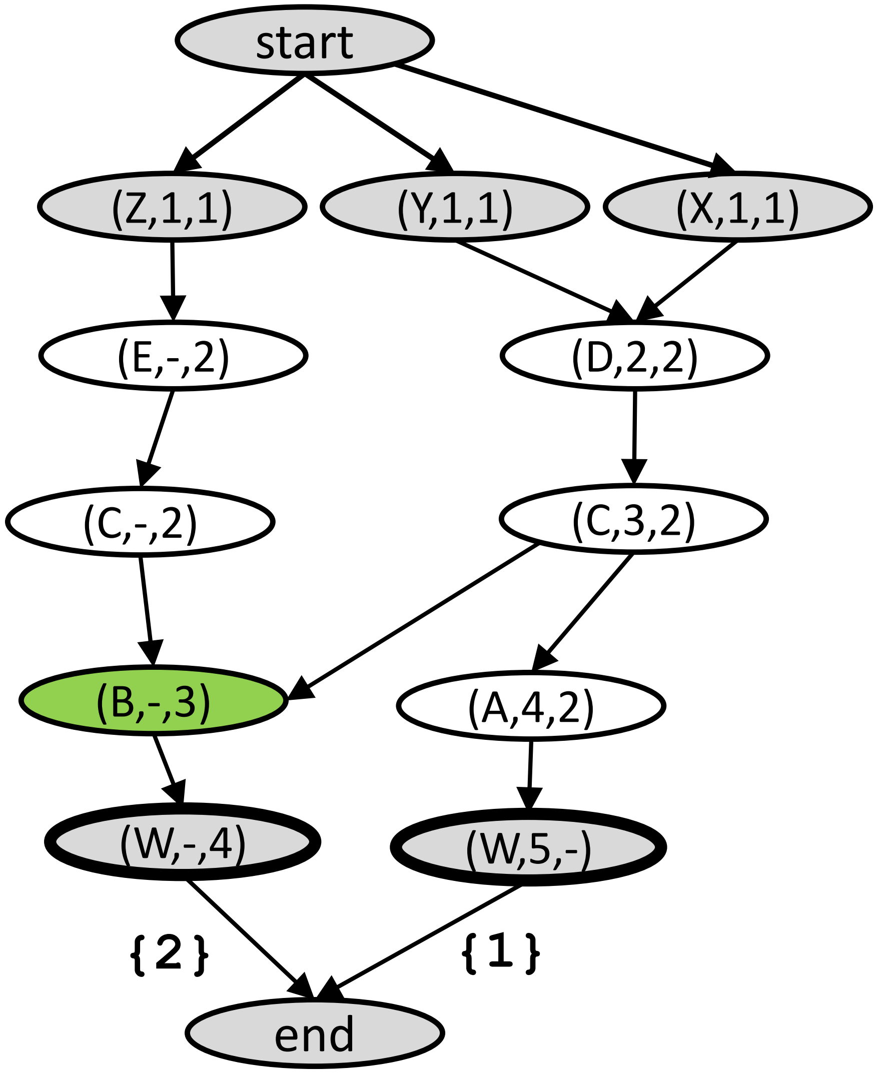
Router A
match peer=C comm=(3,2)
export peer←W, comm←(4,2),
comm← noexport, MED←80

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

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

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

Compilation to BGP



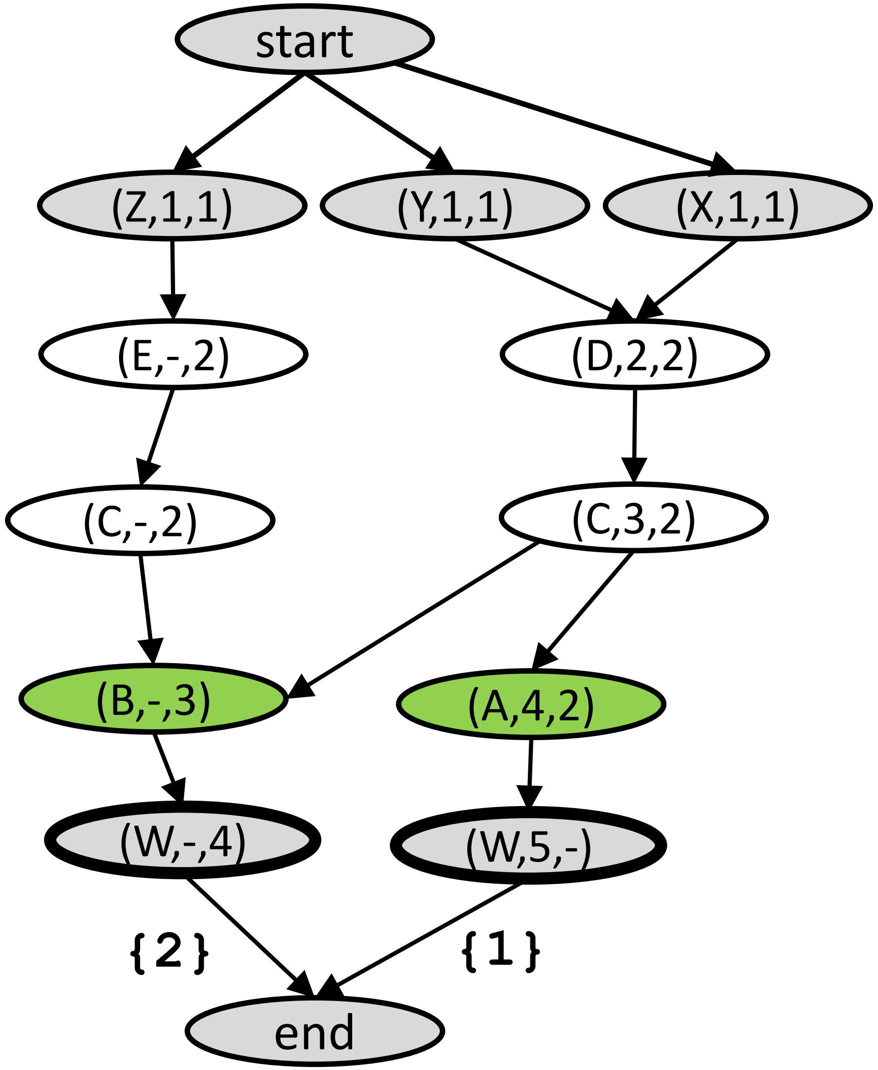
Router A
  match peer=C comm=(3,2)
    export peer←W, comm←(4,2),
      comm← noexport, MED←80

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

Router C
  match[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



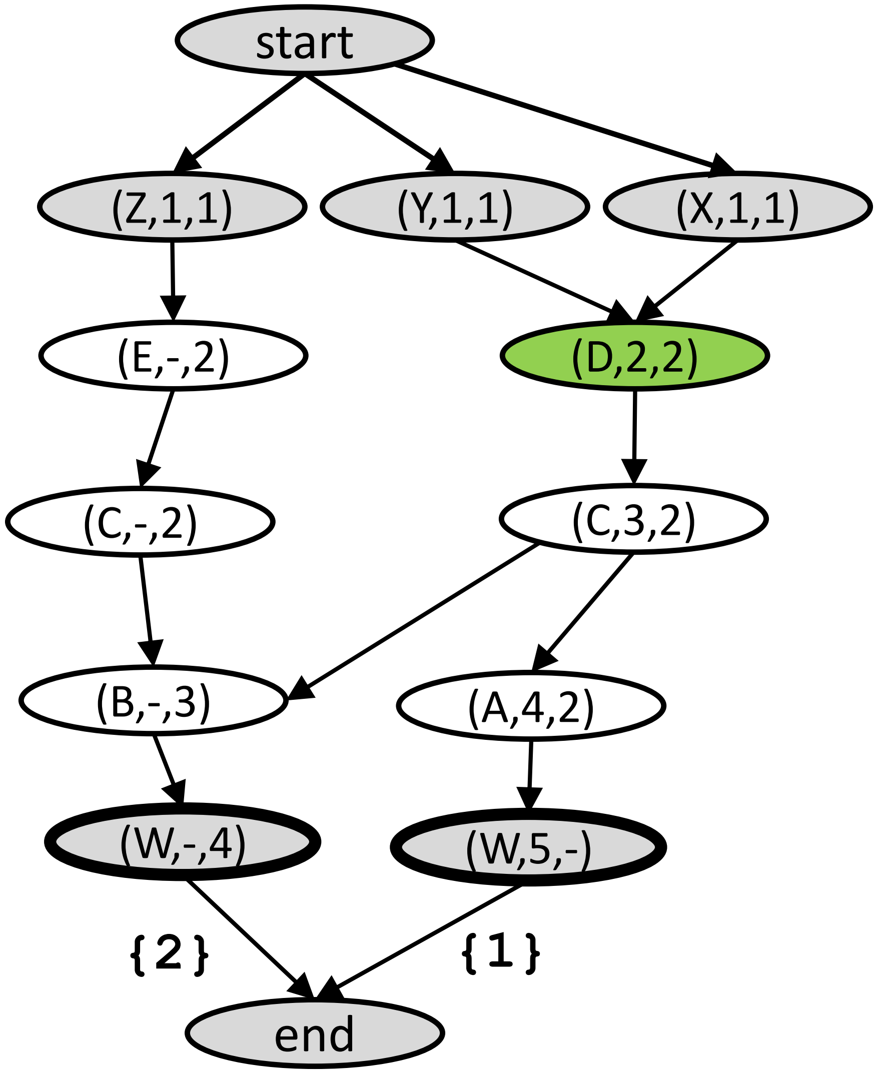
Router A
match peer=C comm=(3,2)
export peer←W, comm←(4,2),
comm← noexport, MED←80

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



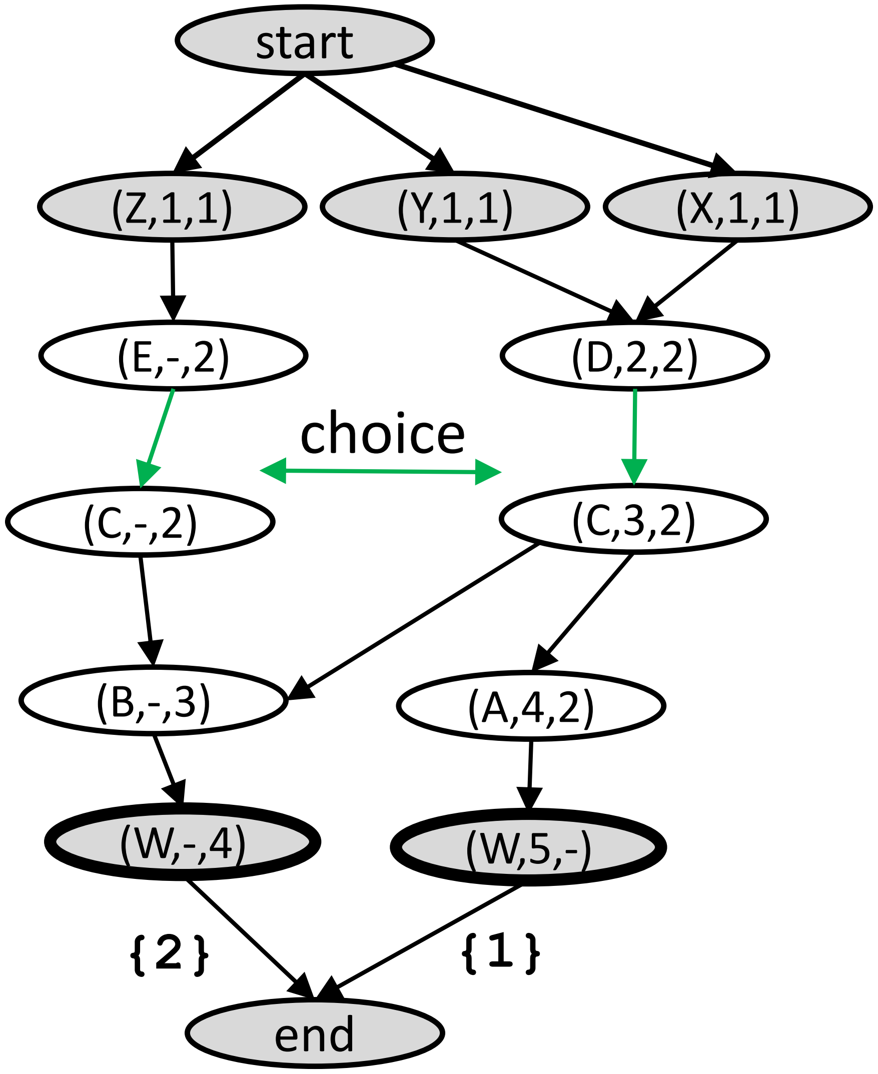
Router A
  match peer=C comm=(3,2)
    export peer←W, comm←(4,2),
        comm← noexport, MED←80

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



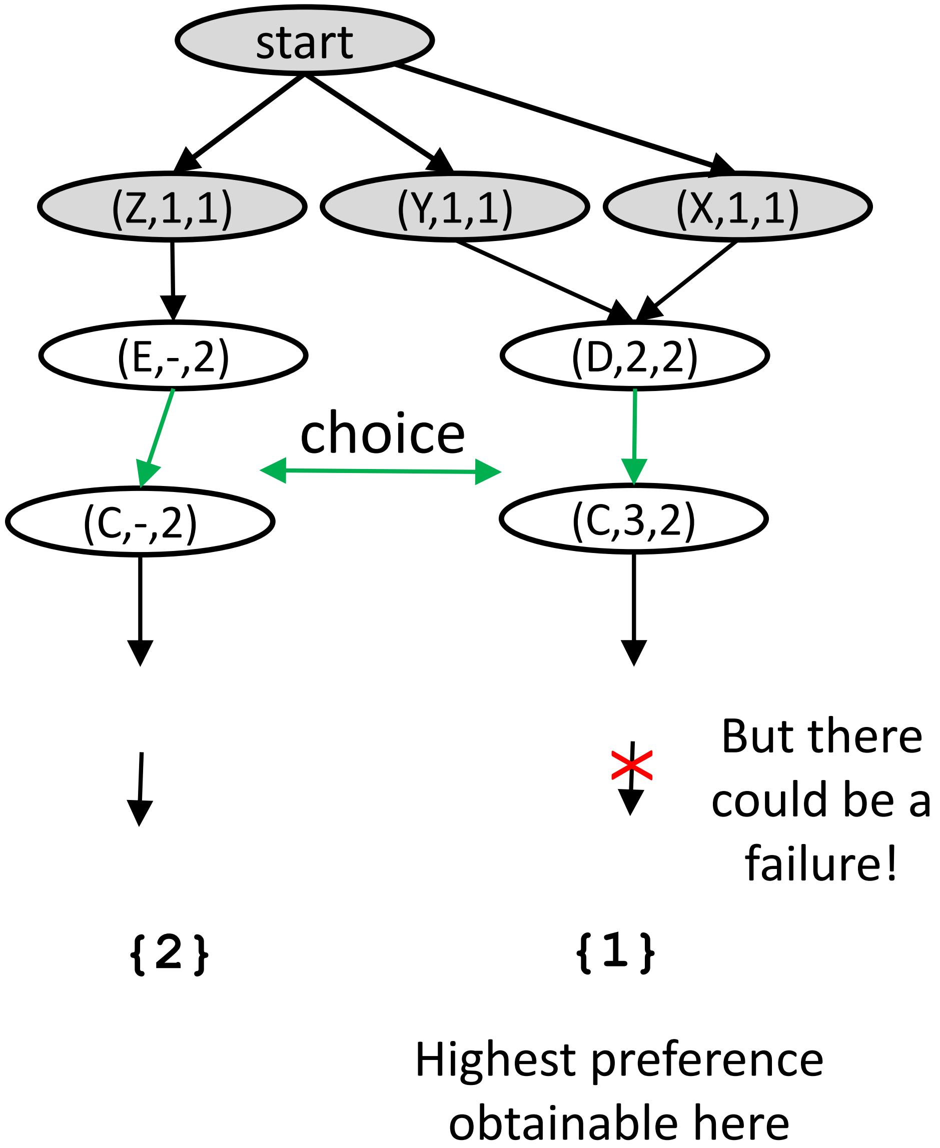
Router A
match peer=C comm=(3,2)
export peer←W, comm←(4,2),
comm← noexport, MED←80

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

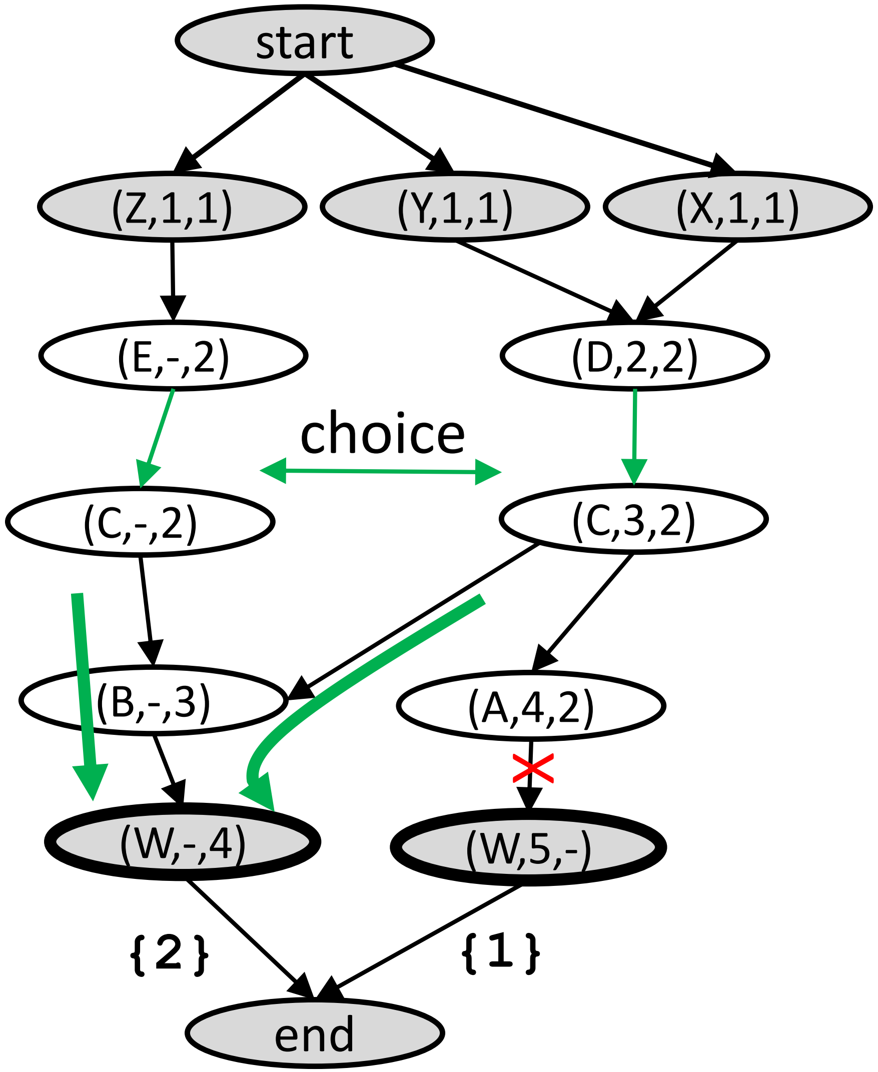
Router B
  match peer=C
    export peer←W, comm←(-,3),
          comm←noexport, MED←81

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



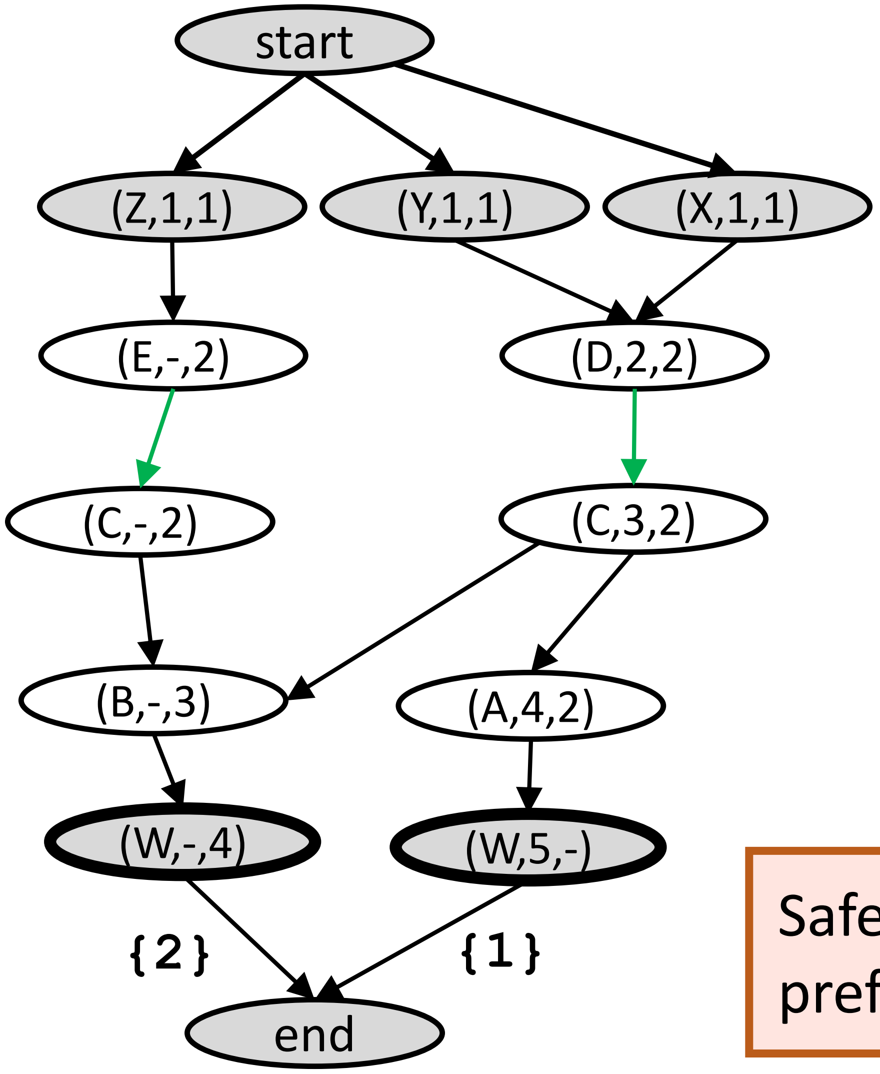
Router A
  match peer=C comm=(3,2)
    export peer←W, comm←(4,2),
      comm← noexport, MED←80

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



Safe to prefer D

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

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

Router C
match[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)
…

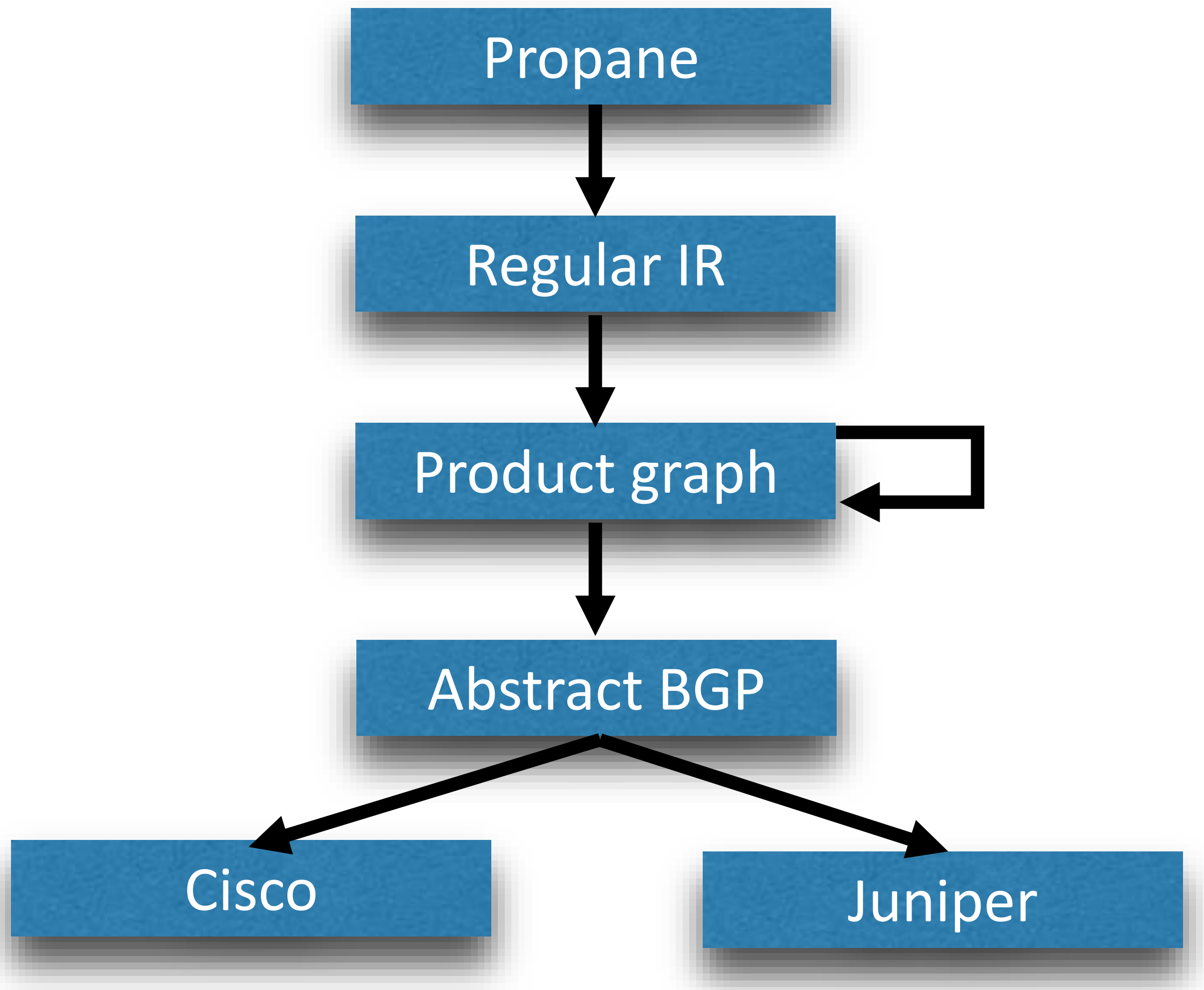
Propane compiler implementation

Efficient graph algorithms

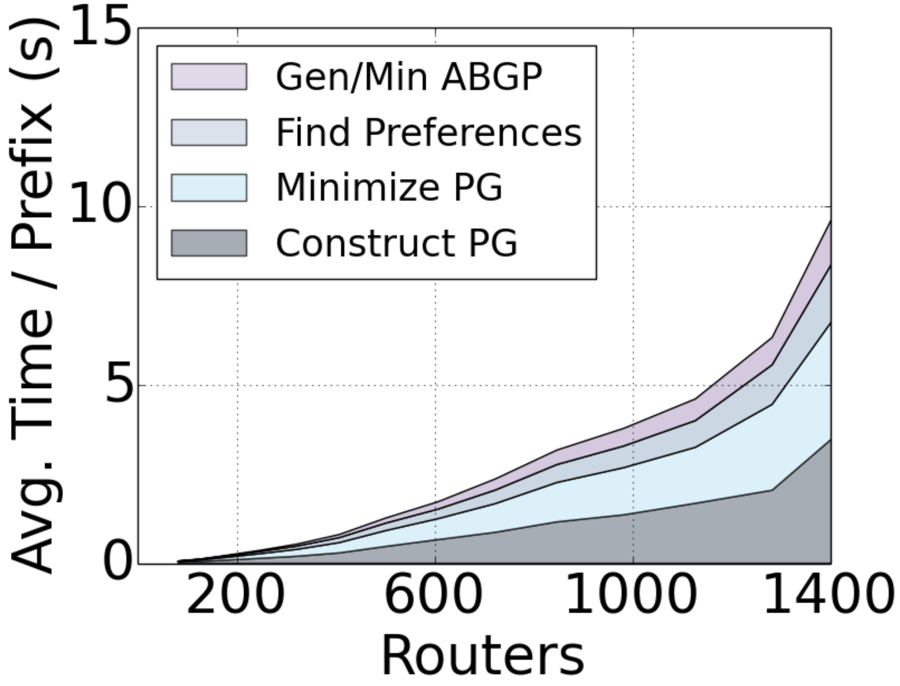
- Minimization
- Failure safety
- Aggregation blackholes

Config minimization

5500 LoC (F#)

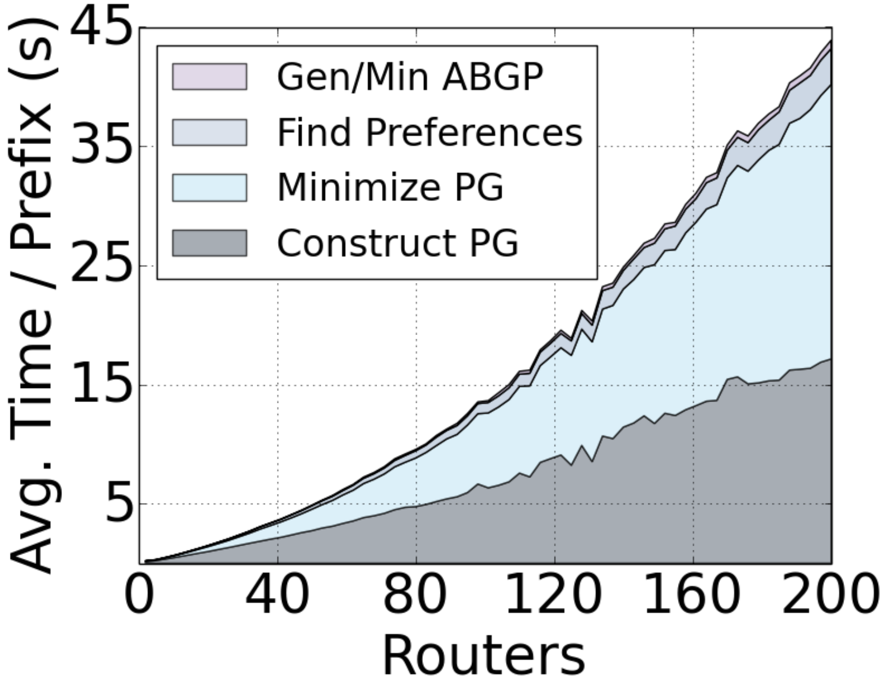


Evaluation on Microsoft network policies



Data center networks

- 31 lines of Propane
- 9 mins for 1400 routers

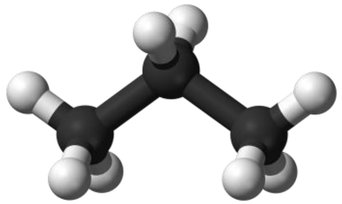


Backbone networks

- 43 lines of Propane
- 3 mins for 200 routers

Summary

Centralized programming of distributed control planes → Resilient **and** programmable networks



Propane

github.com/rabeckett/propane

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