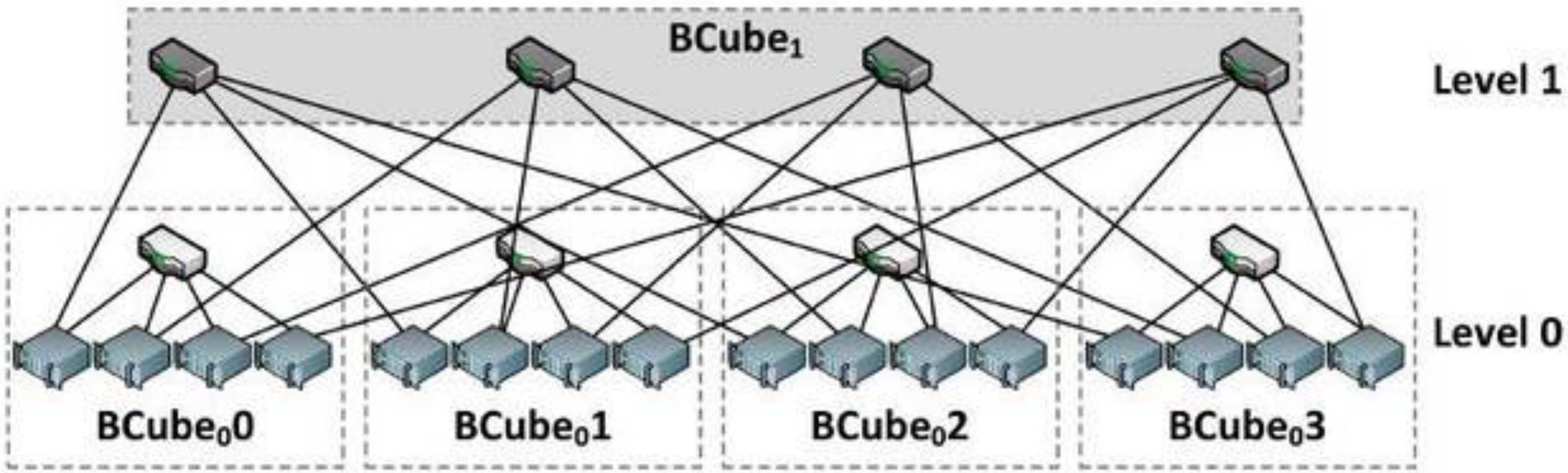
Quick Summary

- Richer Topology Constraints
  - More precise node/edge relations
  - More precise hierarchical information
- Extend to <u>Symbolic</u> Abstract Analysis
- Verification & Synthesis of other protocols

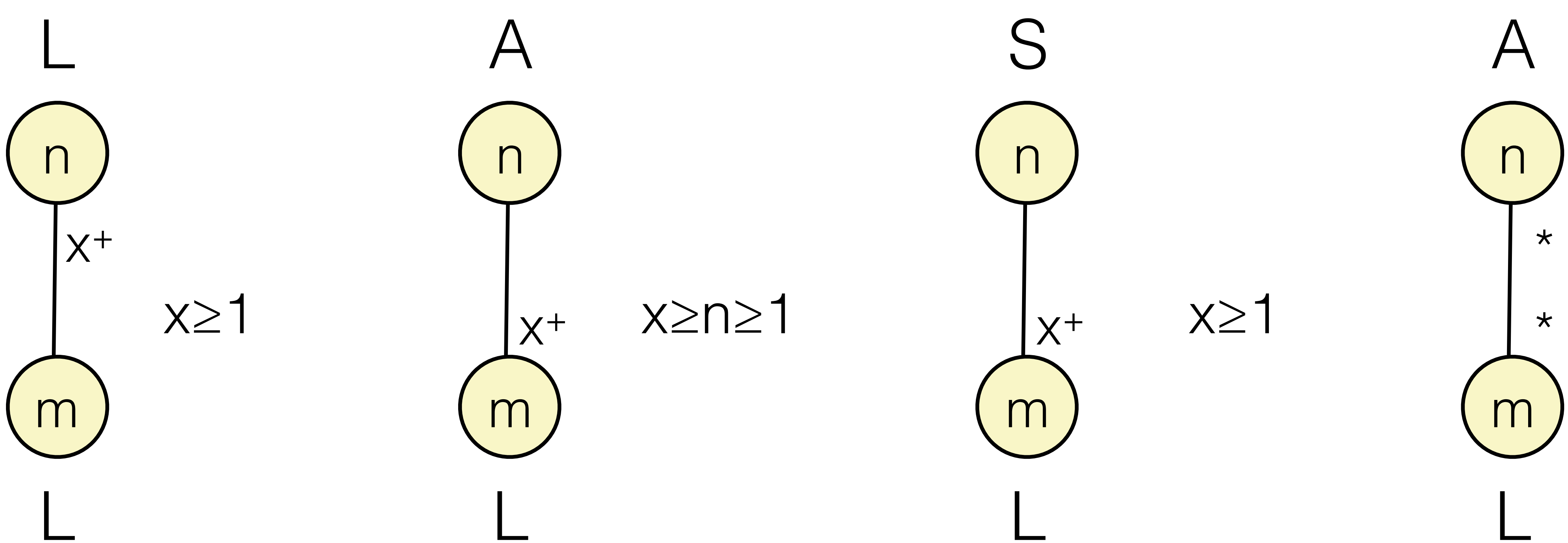
Recap



Problems:

- Node/edge multiplicities related
- Hierarchical invariants convey more info

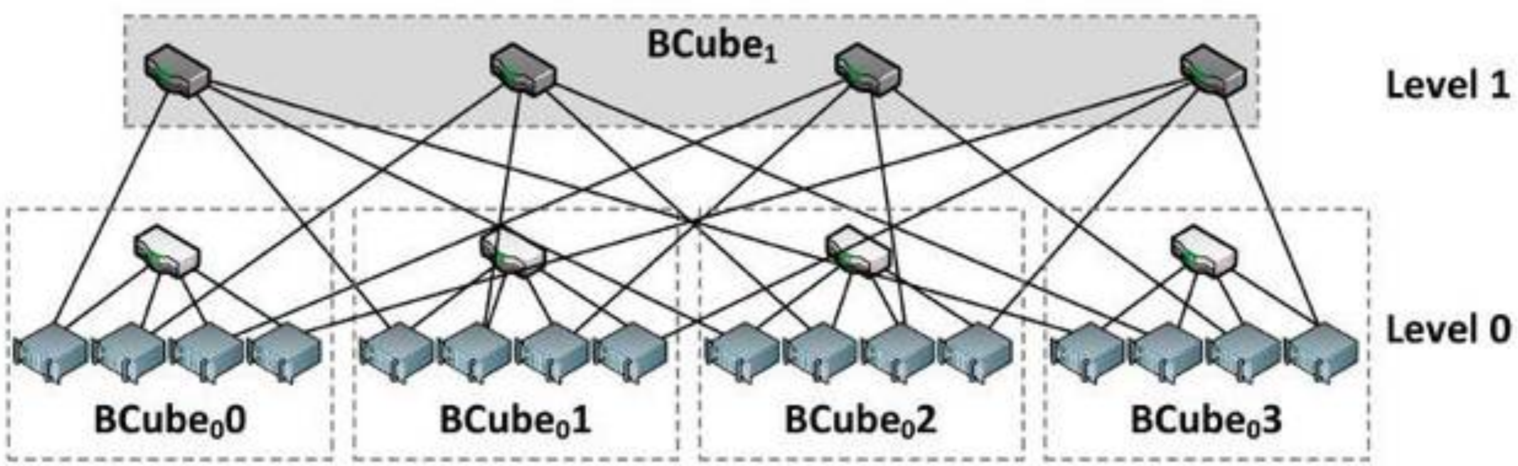
Inference Rules (Recap)

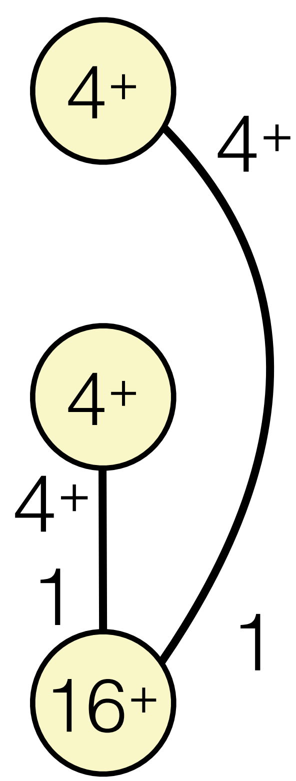


$L \in \{A,S\}$

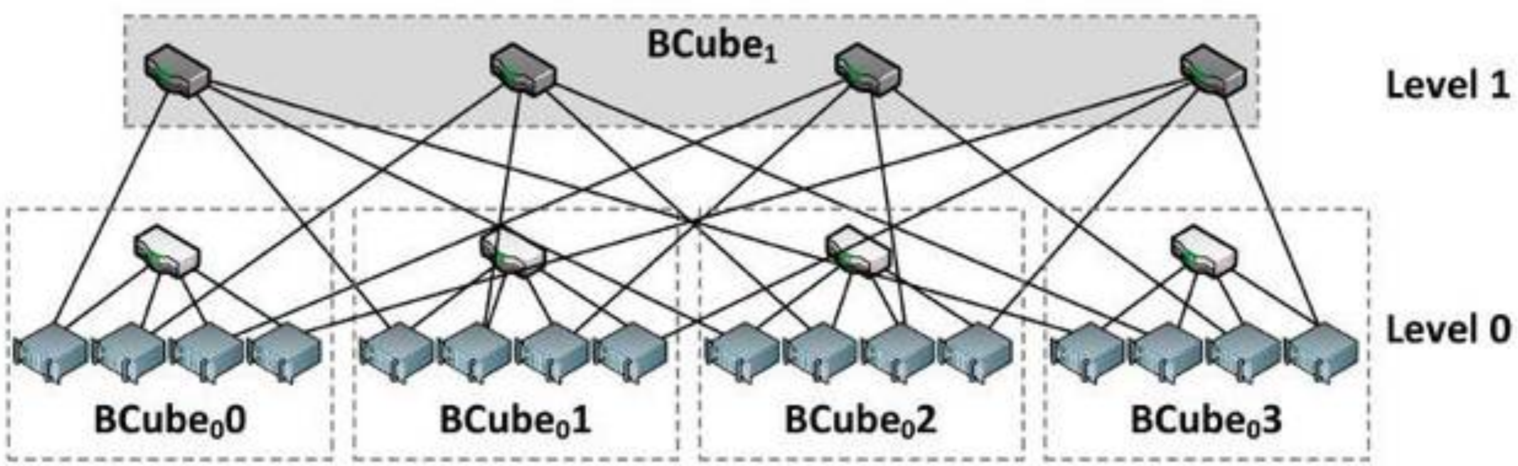
Richer Node/Edge Constraints

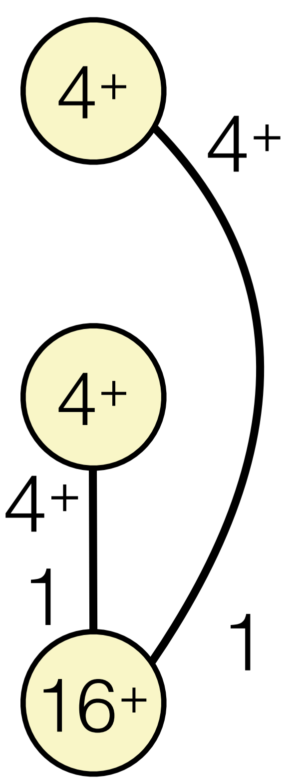
BCube Topology

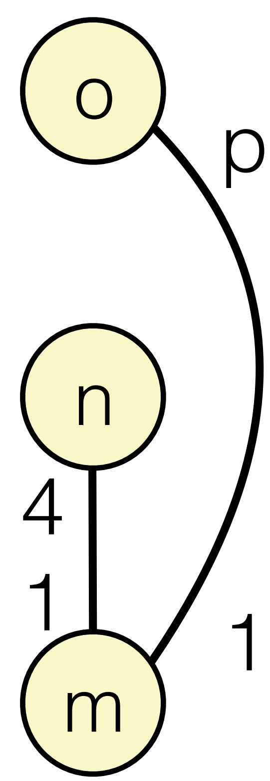




Node/edge dependencies





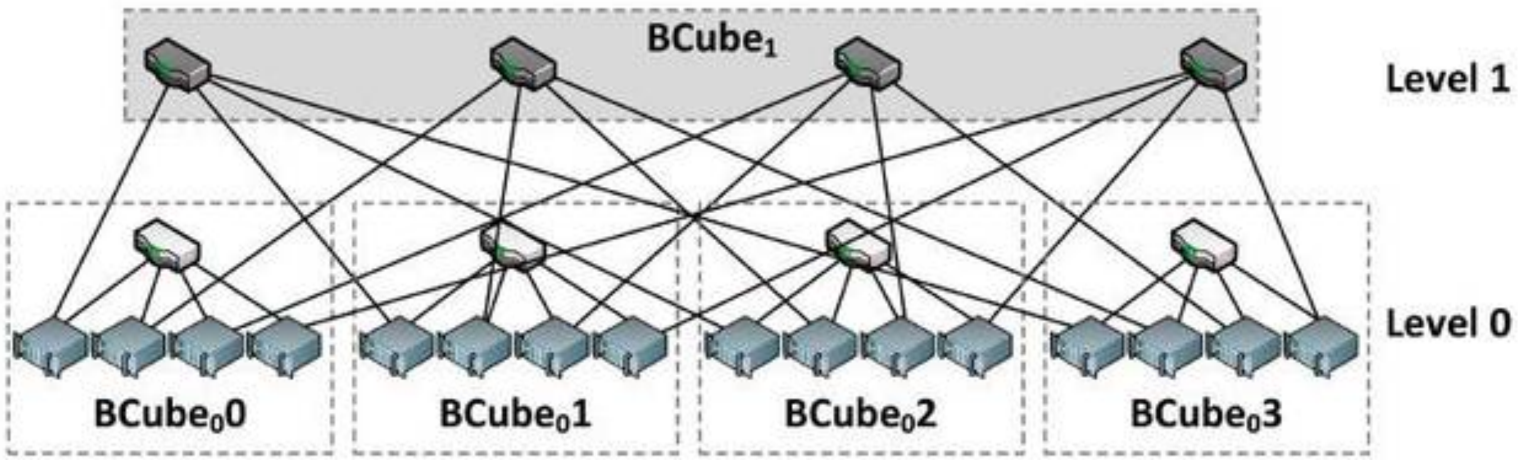


$o = p$

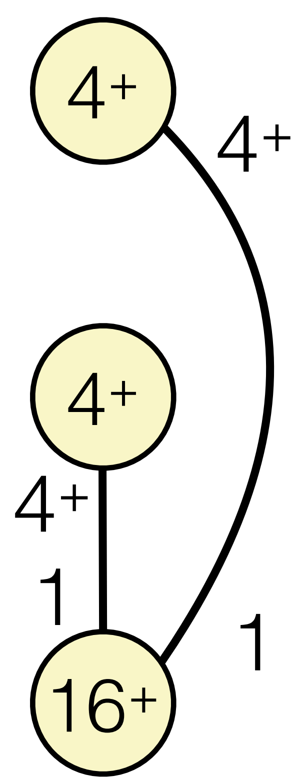
$n = p$

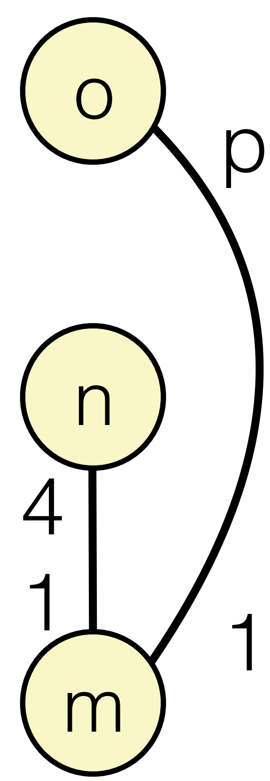
$m \geq 16 \qquad 4 * p = m$

Node/edge dependencies



Strictly more general





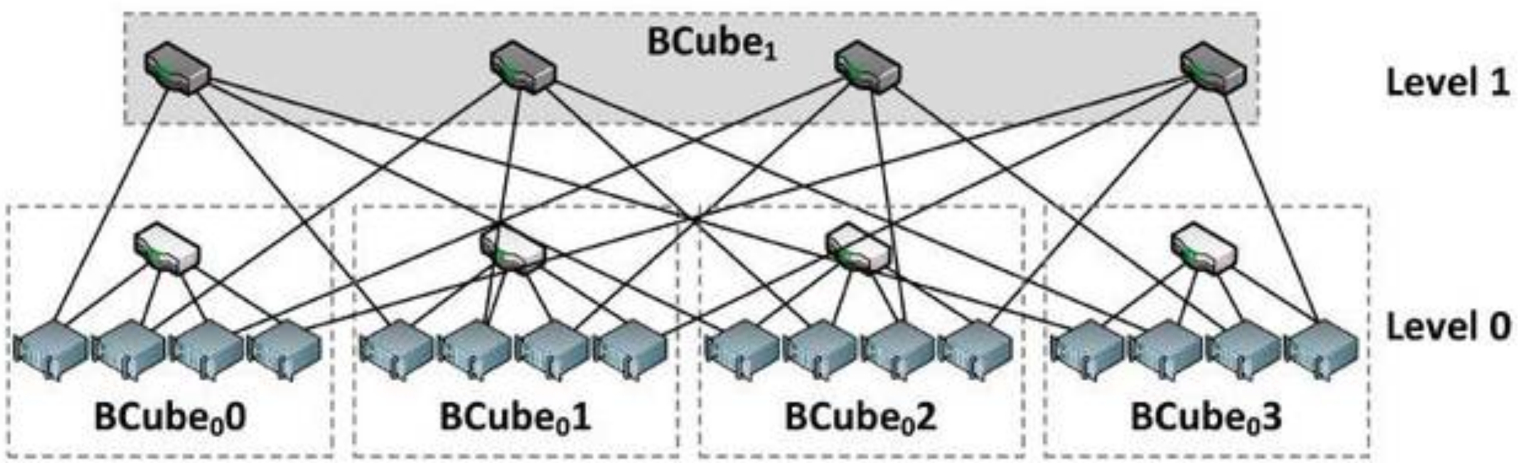
$o = p$

$n = p$

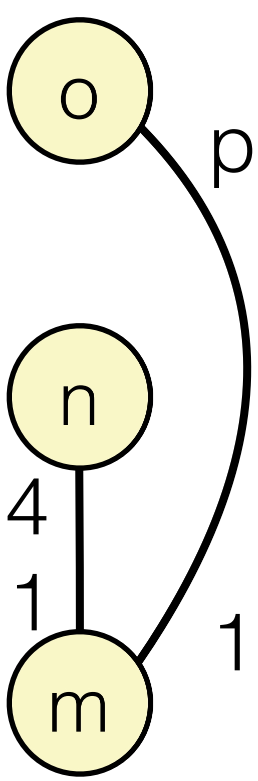
$m \geq 16 \quad 4*p = m$

Symbolic Reachability

Symbolic Reachability



Option 1:
Symbolic backtracking search that evaluates the reachability algorithm

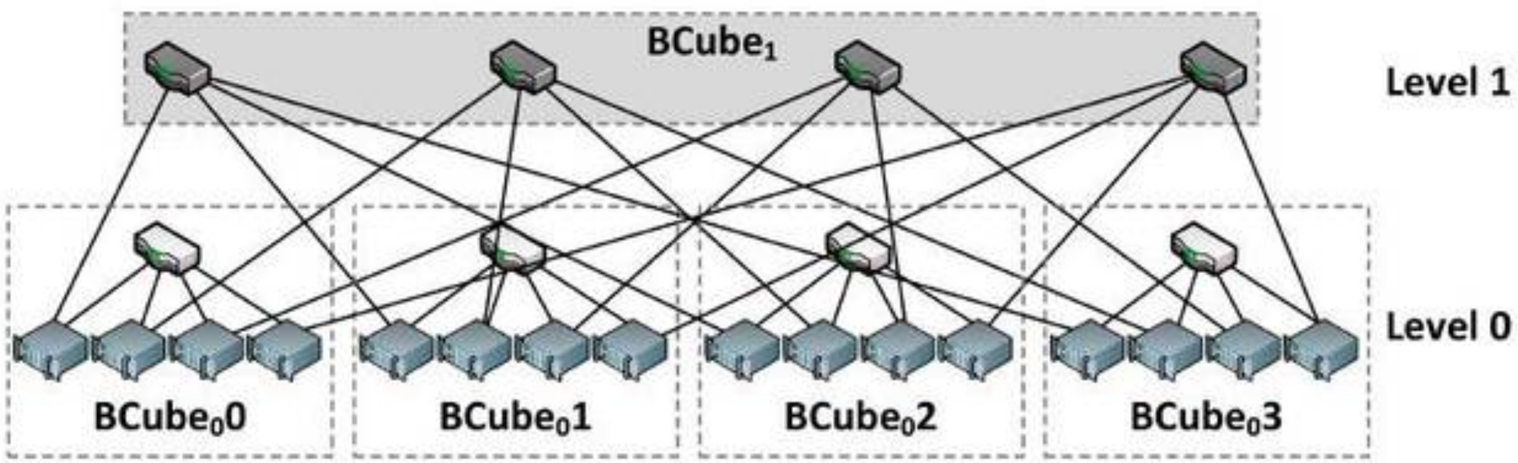


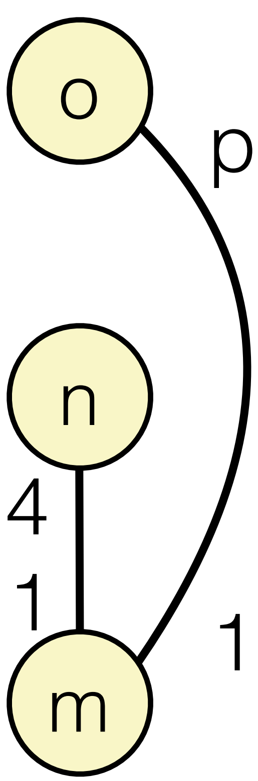
$o = p$

$n = p$

$m \geq 16 \quad 4*p = m$

Symbolic Reachability





$o = p$

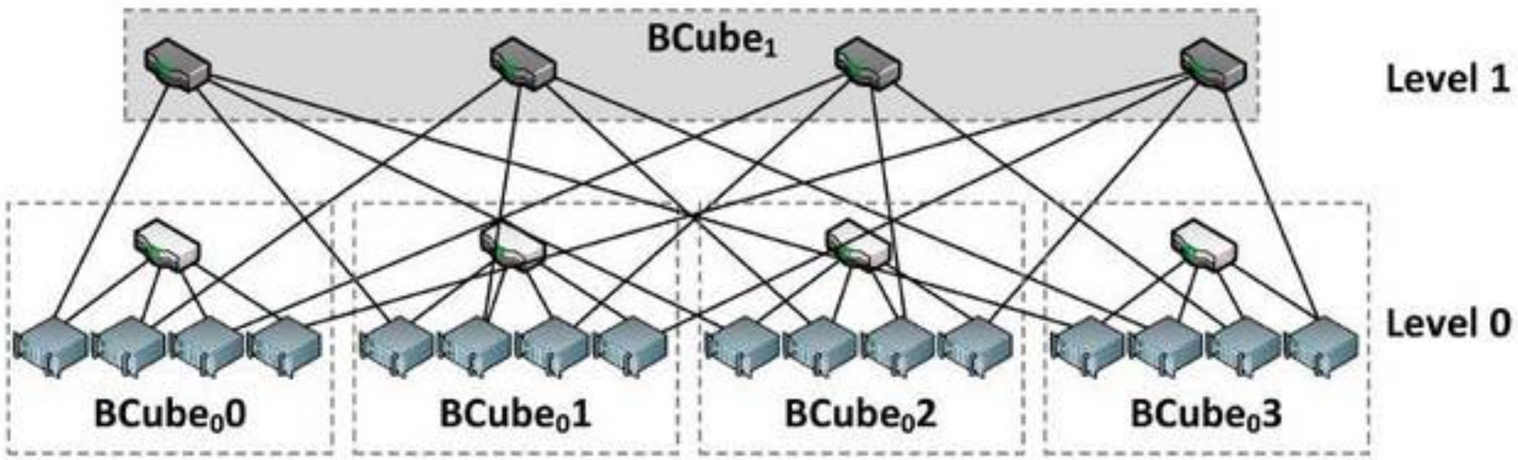
$n = p$

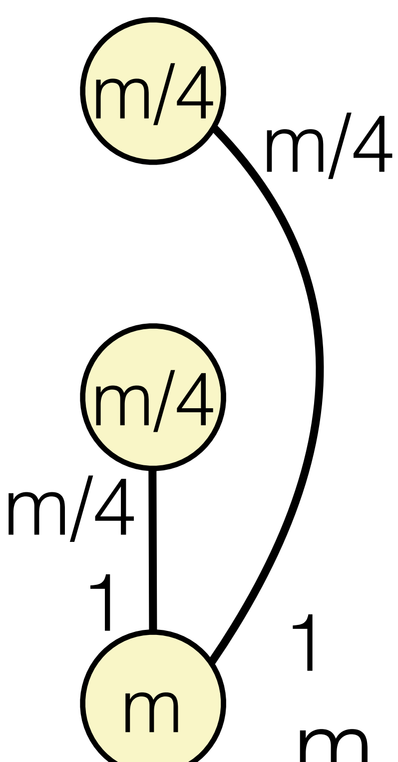
$m \geq 16 \quad 4*p = m$

Option 1:
Symbolic backtracking search that evaluates the reachability algorithm

Look at constraints that change reachability inference rules

Symbolic Reachability





$m \geq 16$

m — o

$1 \geq 1 \longmapsto$ L —> S

$m/4 \geq 1 \longmapsto$ L —> L

$1 \geq m/4 \longmapsto$ L —> A

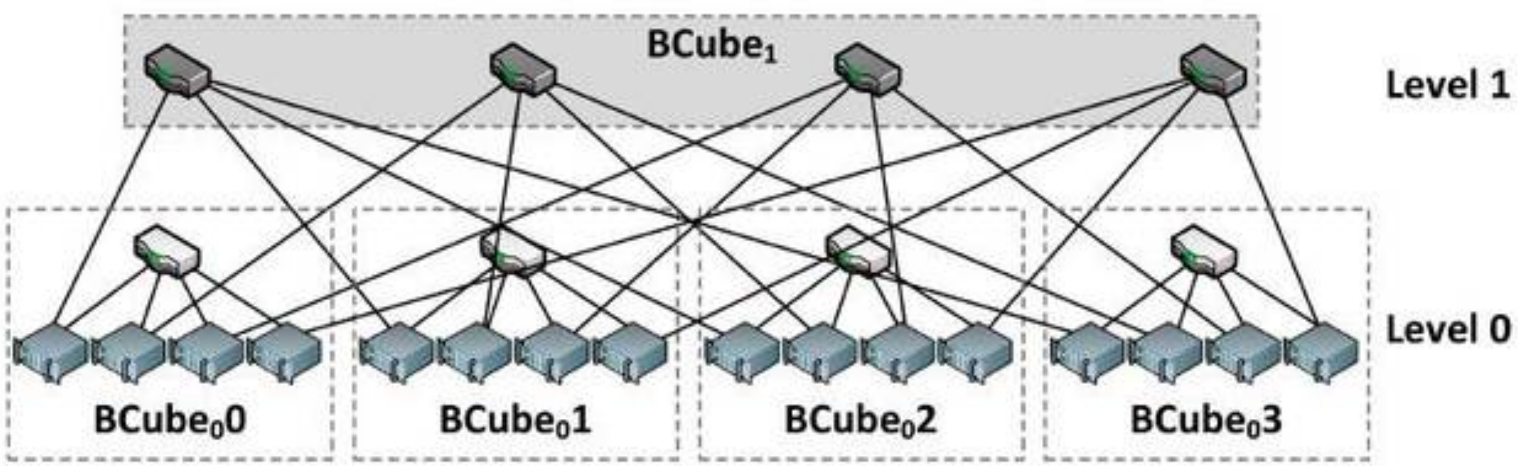
o — m

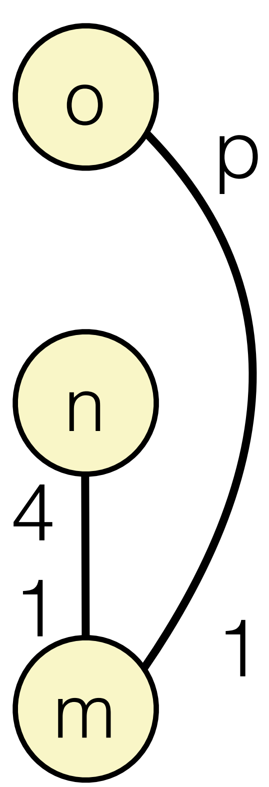
$m/4 \geq 1 \longmapsto$ L —> S

$1 \geq 1 \longmapsto$ L —> L

$m/4 \geq m \longmapsto$ L —> A

Symbolic Reachability





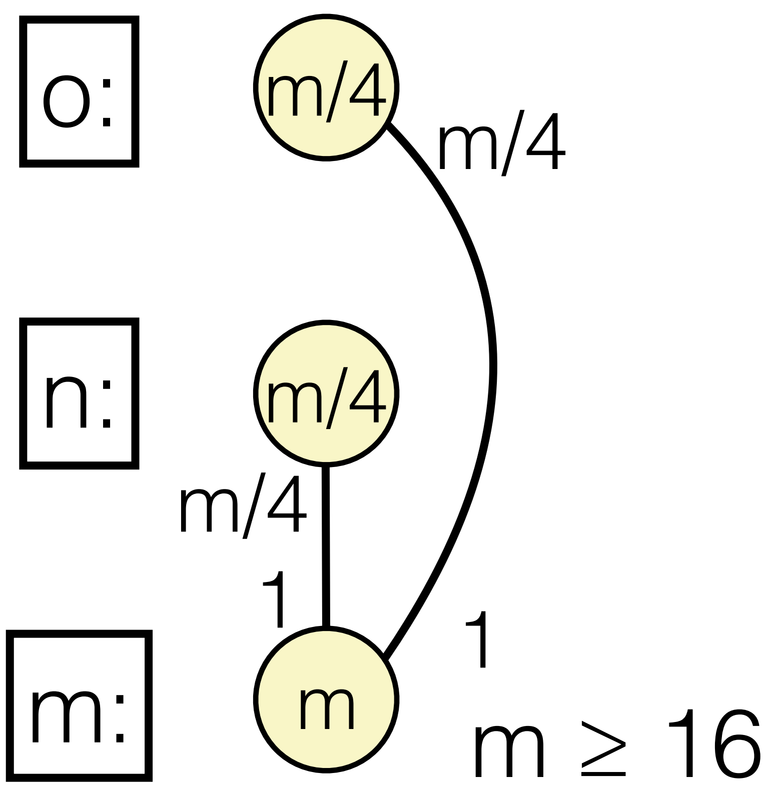
$o = p$

$n = p$

$m \geq 16 \quad 4*p = m$

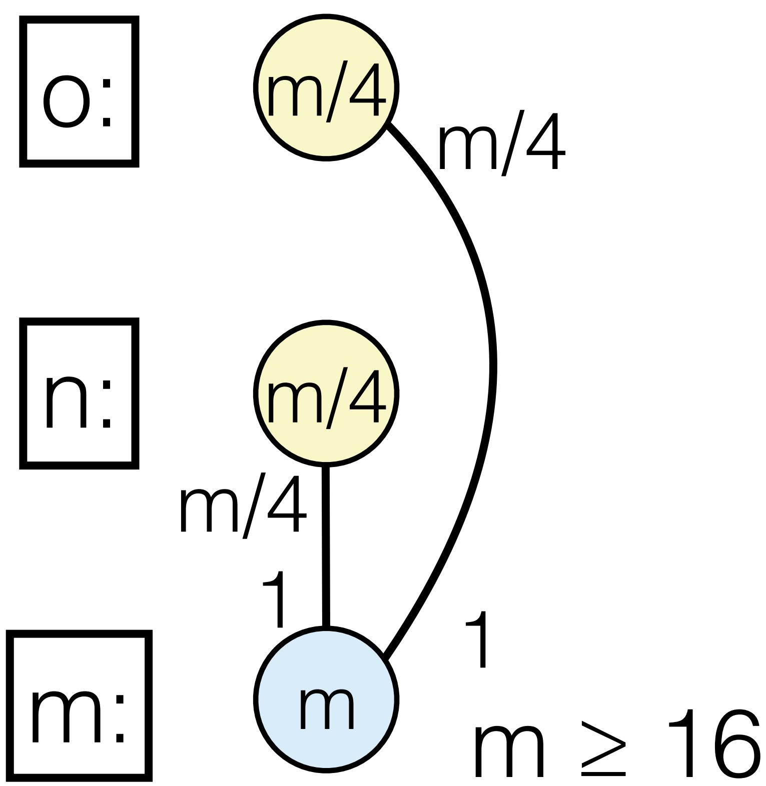
Option 2:
Bake symbolic analysis
into the fixed-point computation

Symbolic Reachability



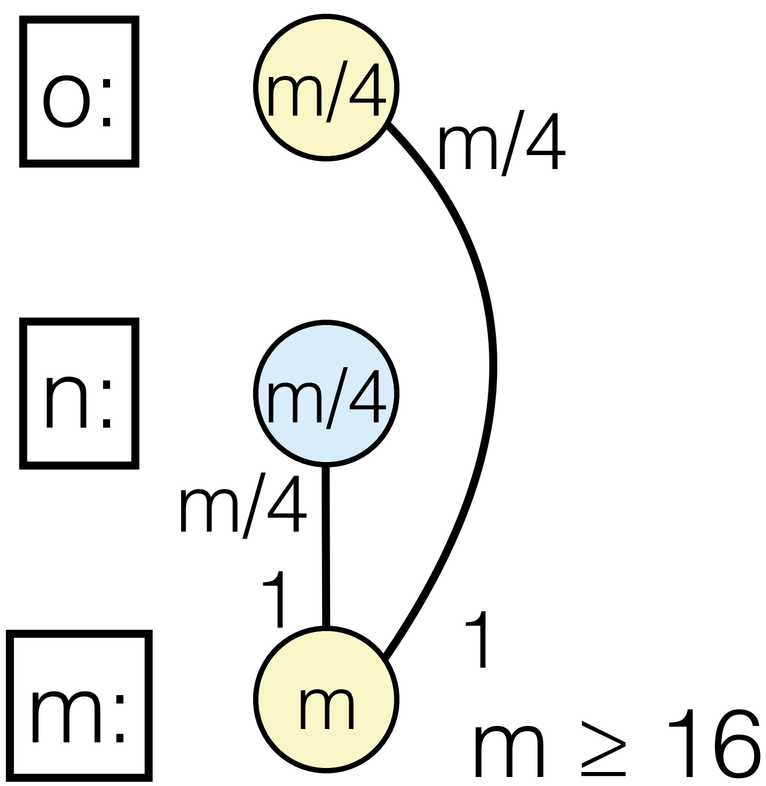
m:	S: A:
n:	S: A:
o:	S: A:

Symbolic Reachability



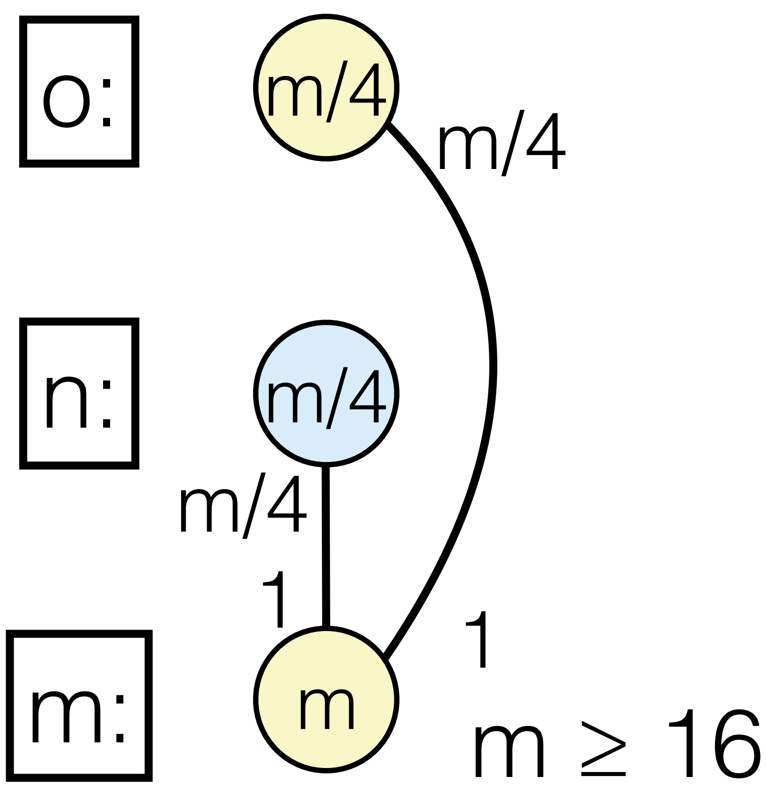
m:	S:	T
	A:	F
n:	S:	F
	A:	F
o:	S:	F
	A:	F

Symbolic Reachability



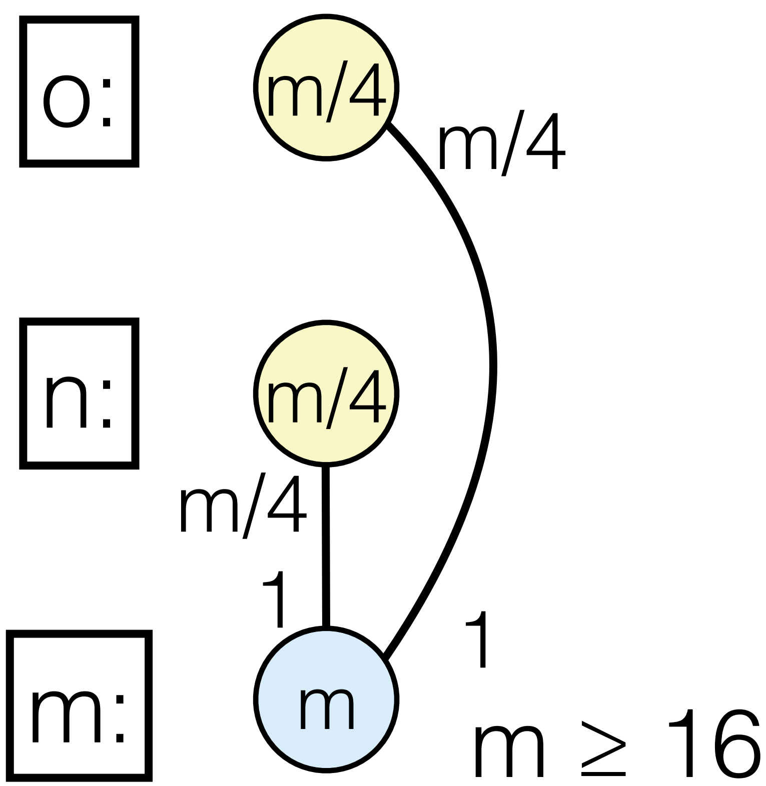
m:	S:	T
	A:	F
n:	S:	$(1 \geq 1) \vee (m/4 \geq m)$
	A:	$(1 \geq m/4) \vee (m/4 \geq m)$
o:	S:	F
	A:	F

Symbolic Reachability



$\boxed{m:}$	S:	T
	A:	F
$\boxed{n:}$	S:	T
	A:	F
$\boxed{o:}$	S:	F
	A:	F

Symbolic Reachability

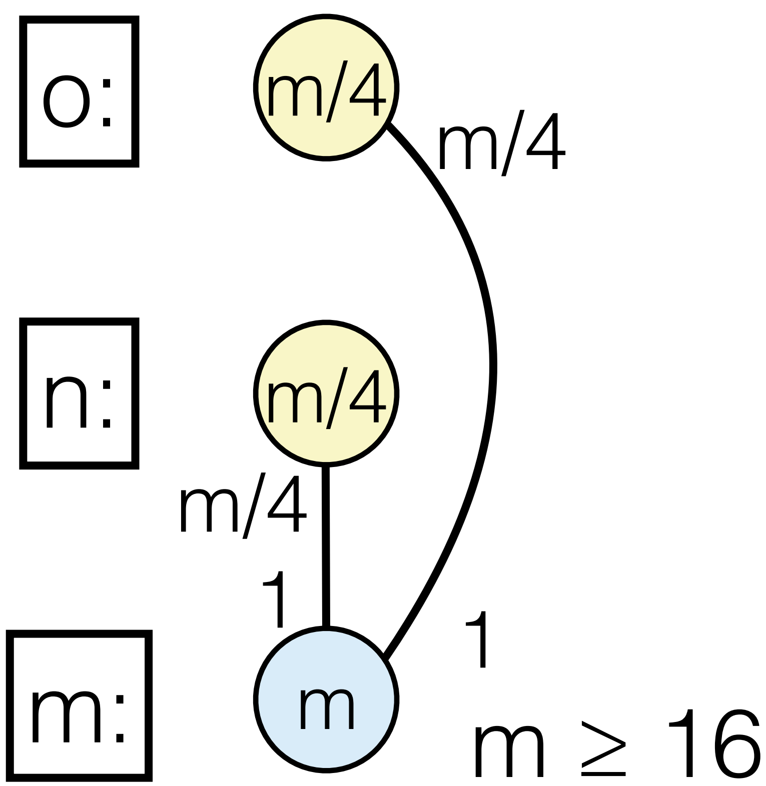


n has S

inference

m:	S:	$T \lor \mathbf{(T \land (m/4 \geq 1 \lor m \geq m/4))}$
	A:	F
n:	S:	T
	A:	F
o:	S:	F
	A:	F

Symbolic Reachability

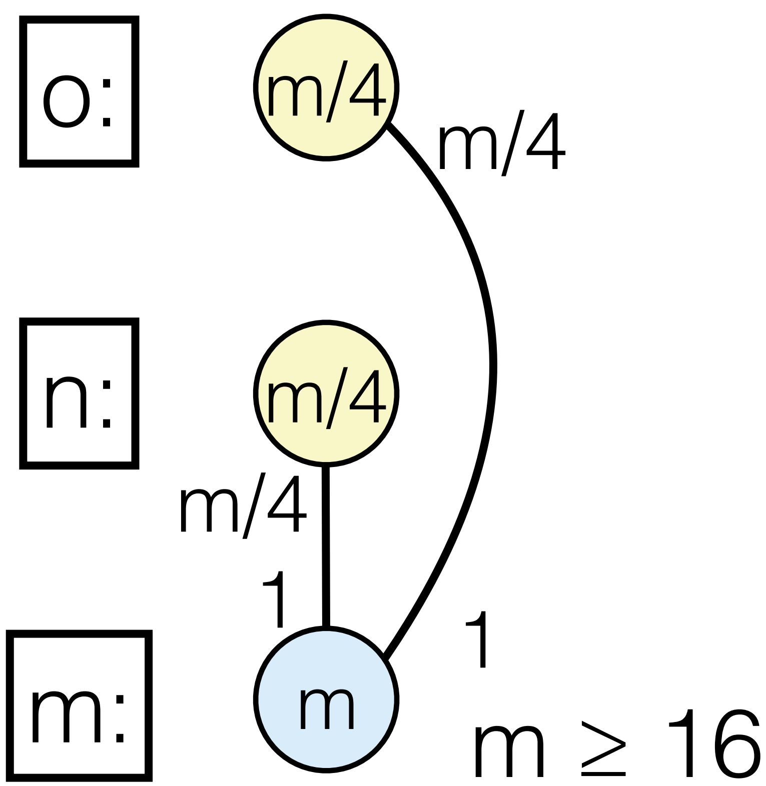


n has A

inference

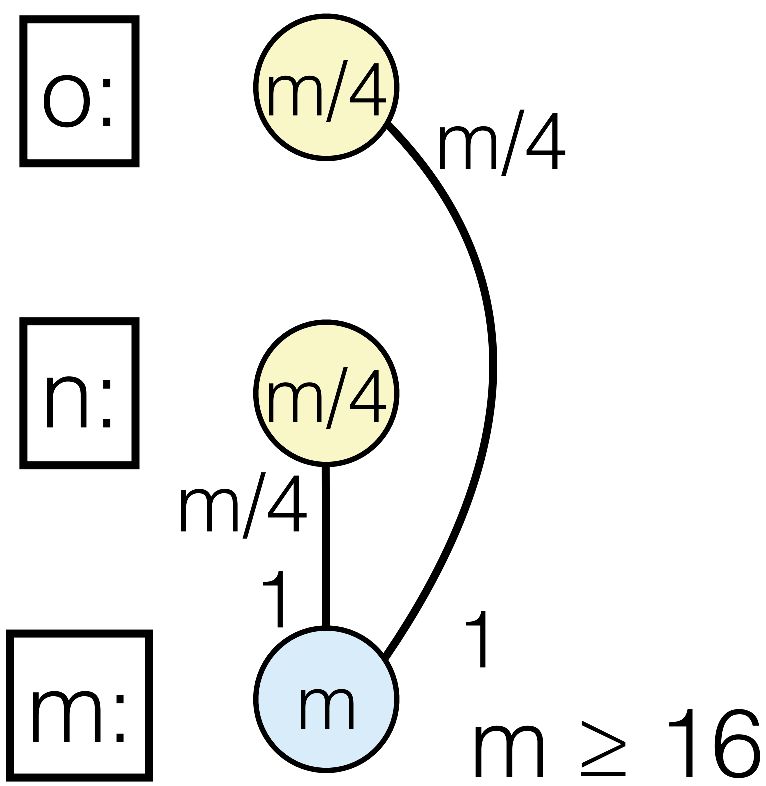
m:	S:	$T \vee \mathbf{(F \wedge (1 \geq 1))}$
	A:	F
n:	S:	T
	A:	F
o:	S:	F
	A:	F

Symbolic Reachability



m:	S:	T
	A:	F
n:	S:	T
	A:	F
o:	S:	F
	A:	F

Symbolic Reachability

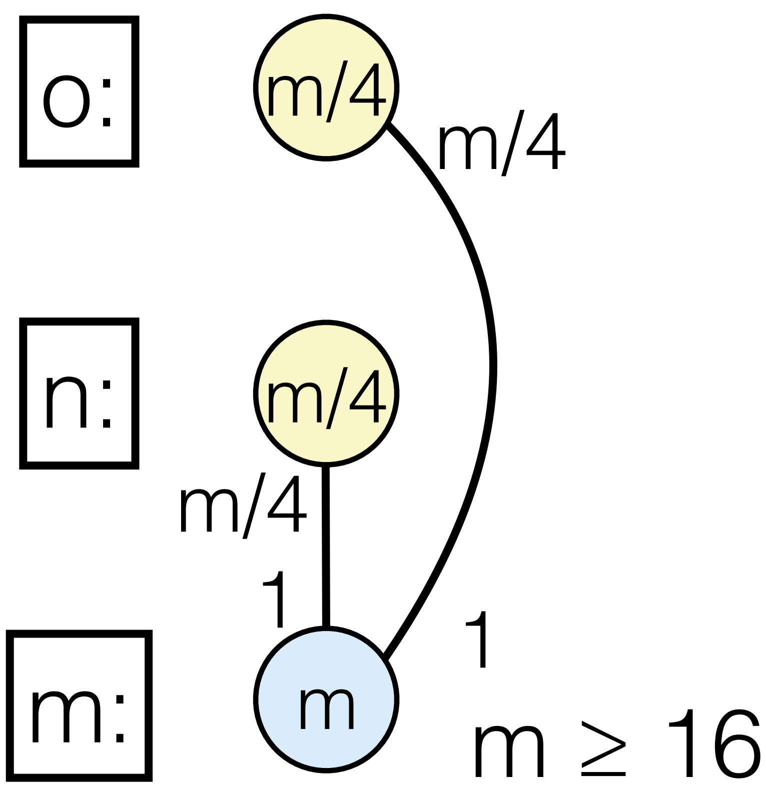


n has S

inference

m:	S:	T
	A:	F∨**(T∧m/4≥m)**
n:	S:	T
	A:	F
o:	S:	F
	A:	F

Symbolic Reachability

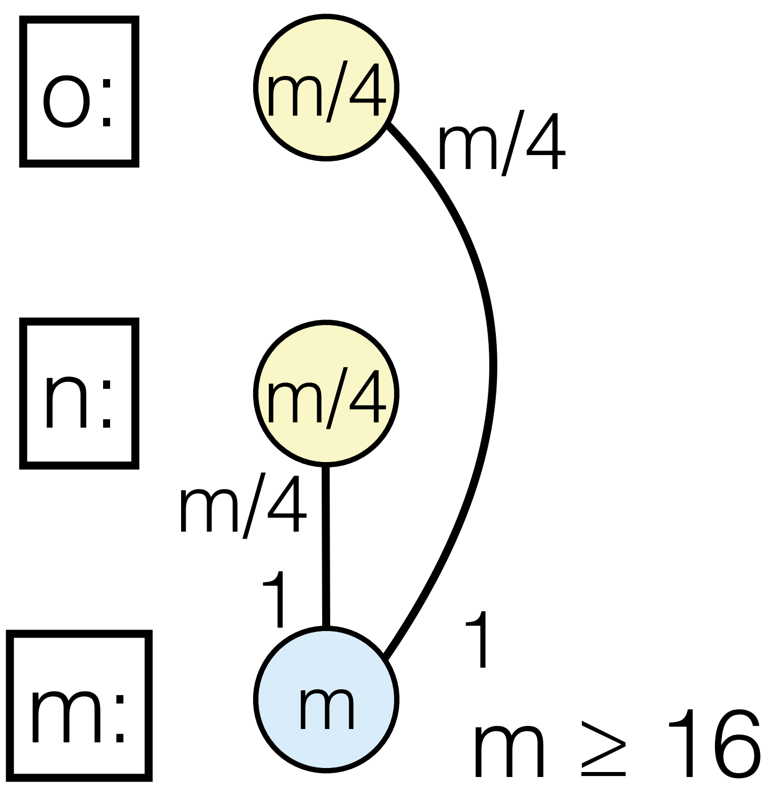


n has A

inference

m:	S:	T
	A:	$F \vee \mathbf{(F \wedge (1 \geq 1))}$
n:	S:	T
	A:	F
o:	S:	F
	A:	F

Symbolic Reachability



m:	S:	T
	A:	F
n:	S:	T
	A:	F
o:	S:	F
	A:	F

Symbolic Reachability

Continue until
Fixed point.

$\boxed{m:}$	S:	T
	A:	F
$\boxed{n:}$	S:	T
	A:	F
$\boxed{o:}$	S:	T
	A:	F

Symbolic Reachability

In general, these give the conditions under which reachability occurs

Validity means any topology is OK.

SAT means some topology is OK

$\boxed{\text{m:}}$	S:	T
	A:	F
$\boxed{\text{n:}}$	S:	T
	A:	F
$\boxed{\text{o:}}$	S:	T
	A:	F

μZ (Microsoft)

Relatively straightforward encoding into the μZ tool.

SMT fixed point engine

- Linear arithmetic
- Propositional Logic
- Relations S(node) A(node)
- Mod, times, division, …

μZ– An Efficient Engine for Fixed points with Constraints*

Kryštof Hoder, Nikolaj Bjørner, and Leonardo de Moura

Manchester University and Microsoft Research

Abstract. The μZ tool is a scalable, efficient engine for fixed points with constraints. It supports high-level declarative fixed point constraints over a combination of built-in and plugin domains. The built-in domains include formulas presented to the SMT solver Z3 and domains known from abstract interpretation. We present the interface to μZ, a number of the domains, and a set of examples illustrating the use of μZ.

1 Introduction

Classical first-order predicate and propositional logic are a useful foundation for many program analysis and verification tools. Efficient SAT and SMT solvers and first-order theorem provers have enabled a broad range of applications based on this premise. However, fixed points are ubiquitous in software analysis. Model-checkers compute a set of reachable states as a least fixed point, or dually a set of states satisfying an inductive invariant as a greatest fixed point. Abstract interpreters compute fixed points over an infinite lattice using approximations. An additional layer is required when using first-order engines in these contexts.

The μZ tool is a scalable, efficient engine for fixed points with constraints. At the core is a bottom-up Datalog engine. Such engines have found several applications for static program analysis. A distinguishing feature of μZ is a pluggable and composable API for adding alternative finite table implementations and abstract relations by supplying implementations of relational algebra operations join, projection, union, selection and renaming. Lattice join and widening can be supplied to use μZ in an abstract interpretation context. The μZ tool is part of Z3 [3] and is available from Microsoft Research since version 2.18[1].

2 Architecture

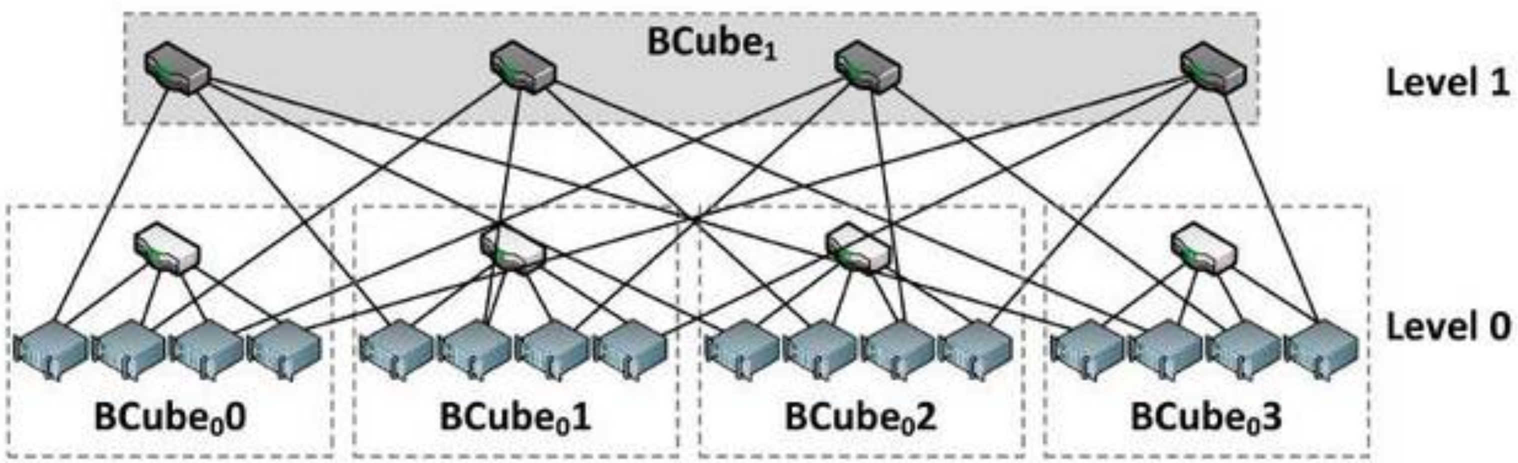
A sample program is in Fig. 1 and the main components of μZ are depicted on Fig. 2. As input μZ receives a set of relations, rules (Horn clauses) and ground facts (unit clauses). The last rule uses the

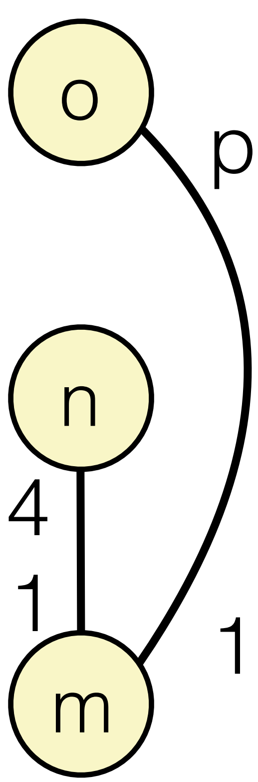
ℓ_0 : [*Int*] using pentagon
ℓ_1 : [*Int*] using pentagon
$\ell_0(0).$
$\ell_0(x) \leftarrow \ell_0(x_0), x = x_0 + 1, x_0 < n.$
$\ell_1(x) \leftarrow \ell_0(x), n \leq x.$

Fig. 1. Sample μZ input

* Appeared in CAV 2011, Copyright Springer Verlag.
[1] http://research.microsoft.com/en-us/um/redmond/projects/z3/

Still Not Enough!





o = p

n = p

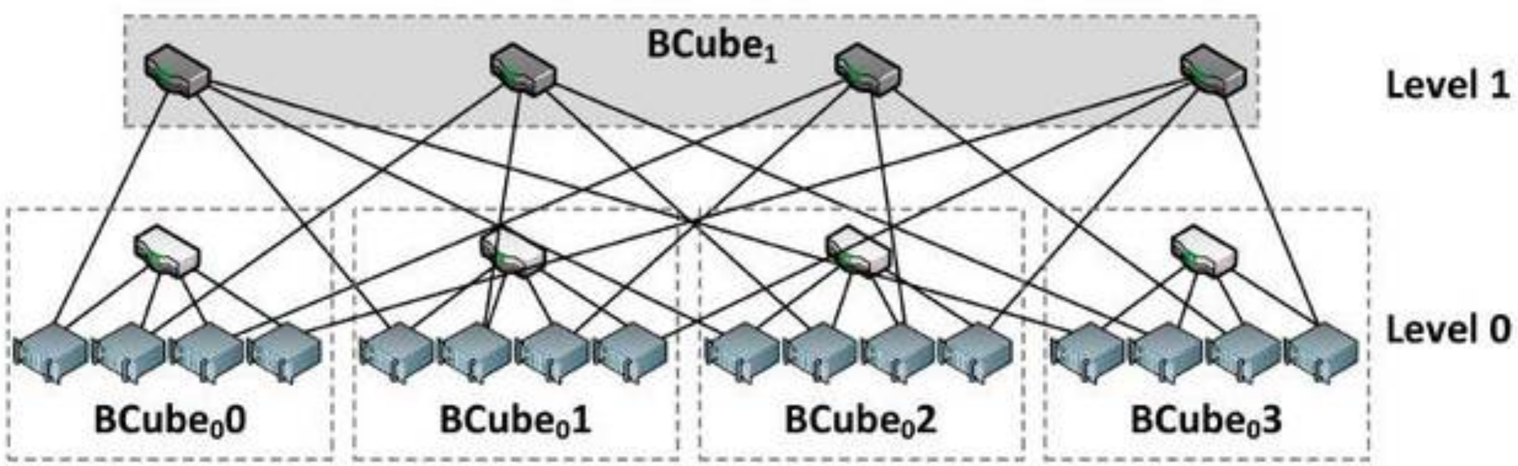
m ≥ 16   4*p = m

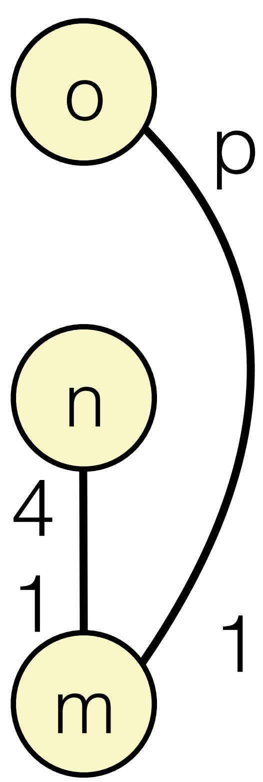
Reachability only infers **some** nodes are reachable

Missing invariant that each Level1 goes to each pod

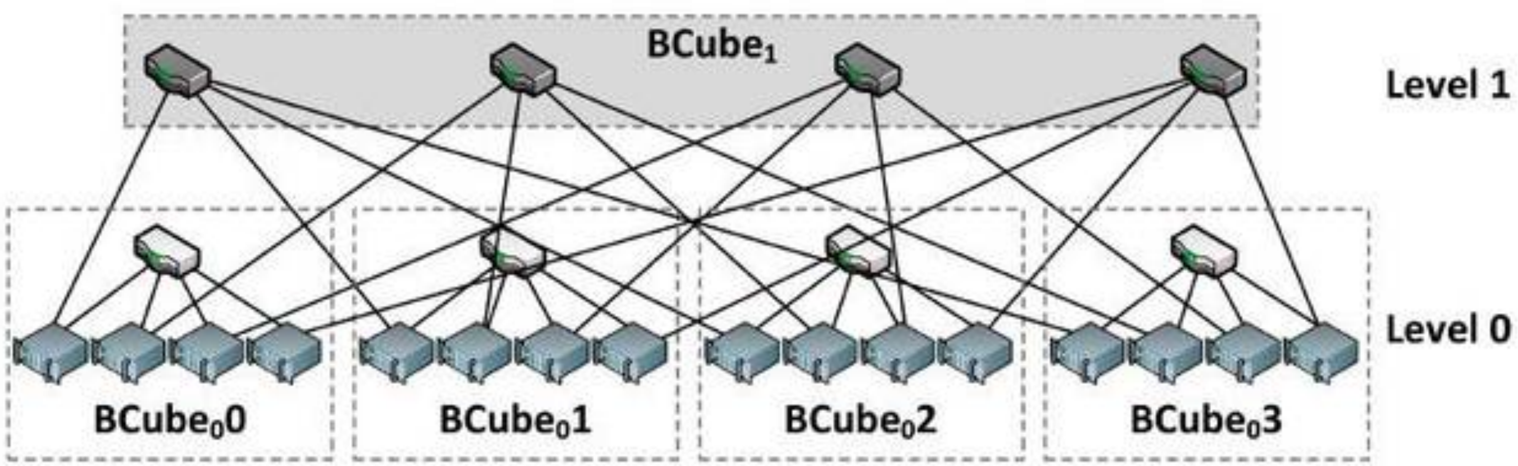
Richer Hierarchy Information

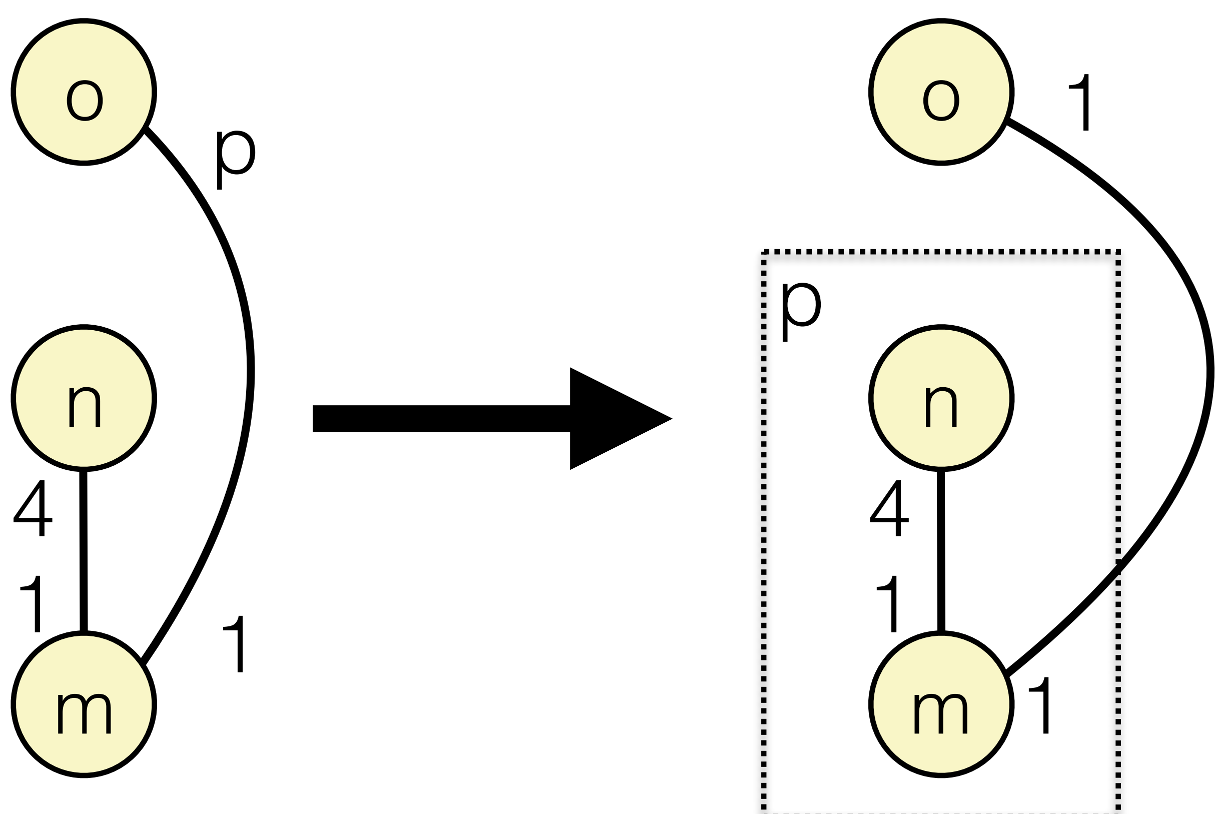
Hierarchical Information





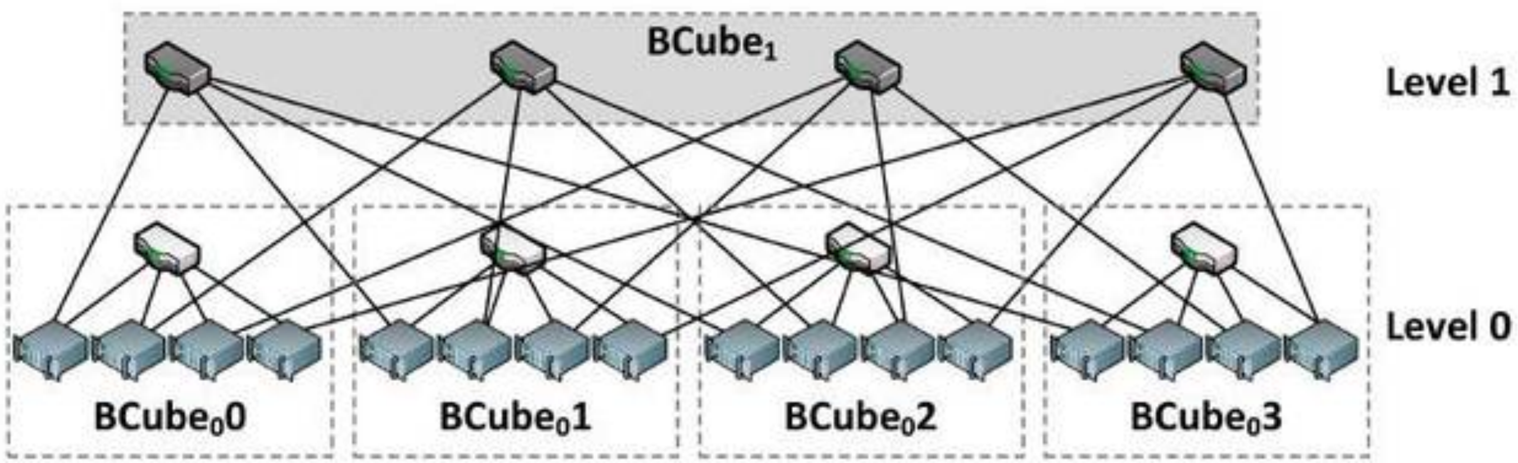
Hierarchical Information

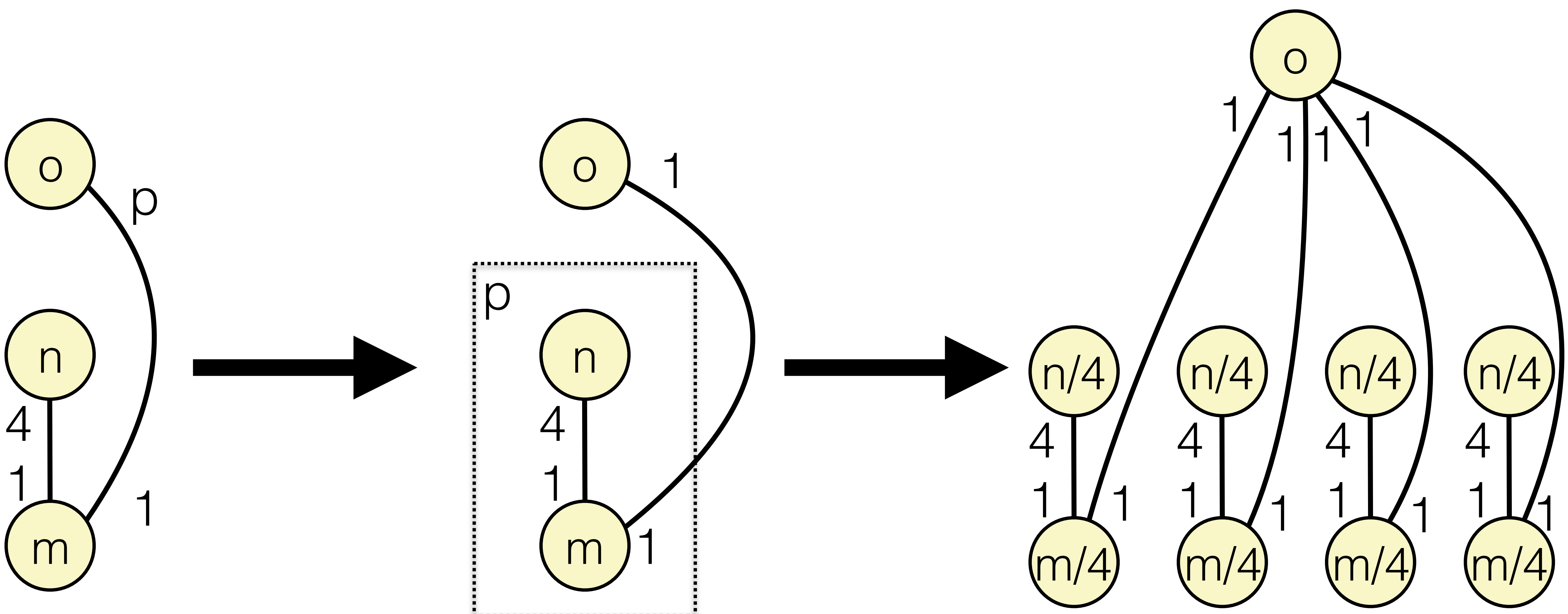




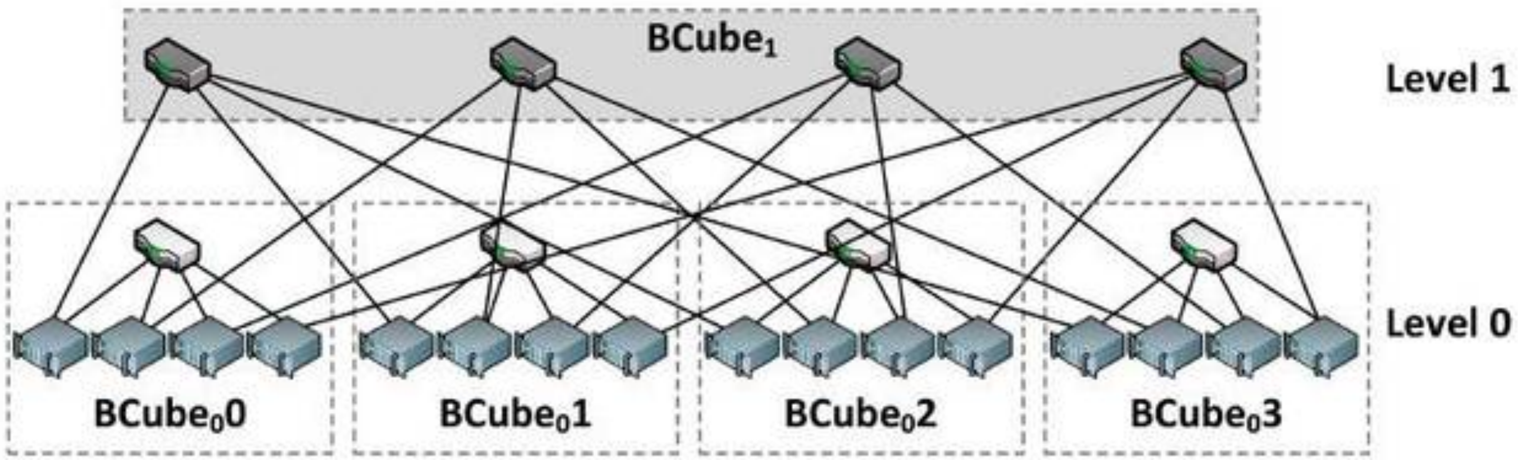
1 in/out edge
from each pod

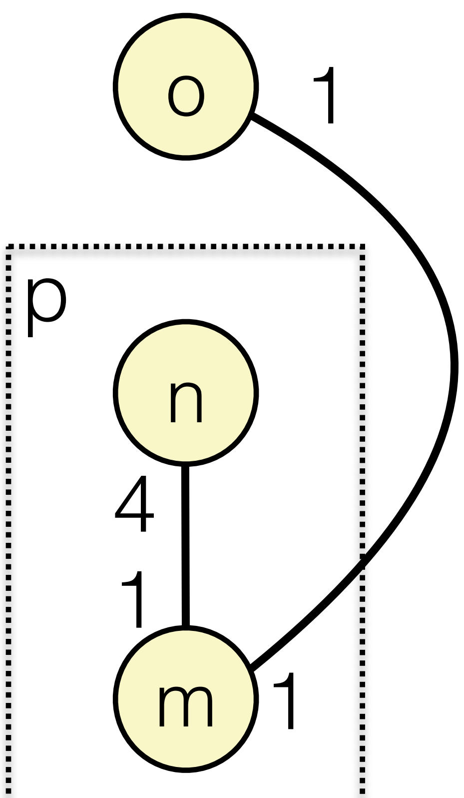
Hierarchical Information

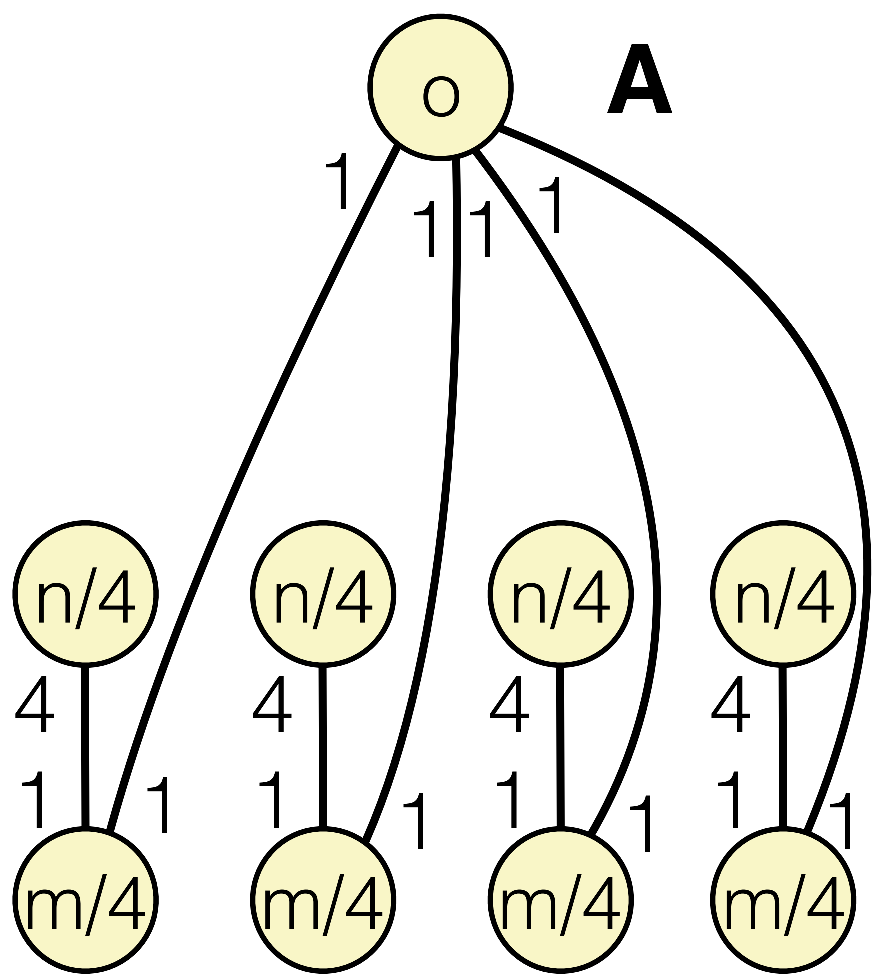




Hierarchical Information

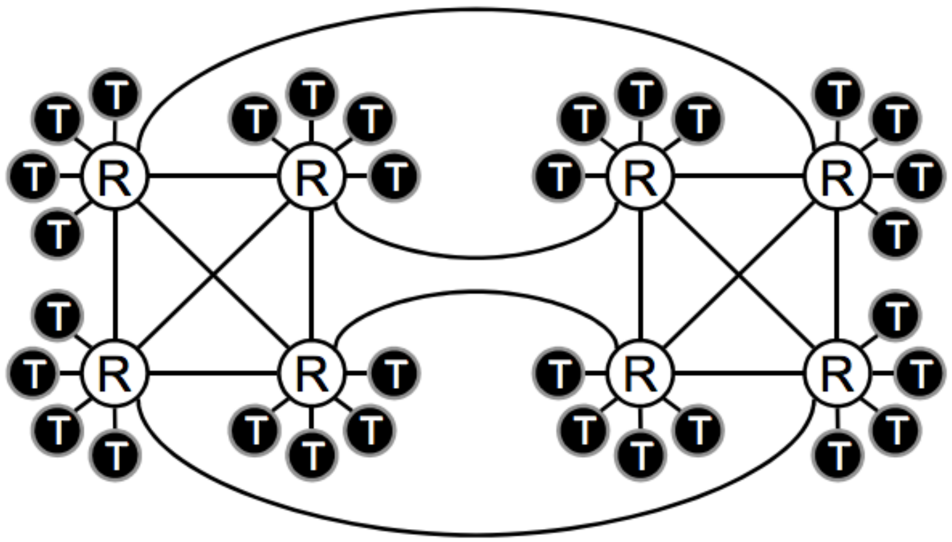




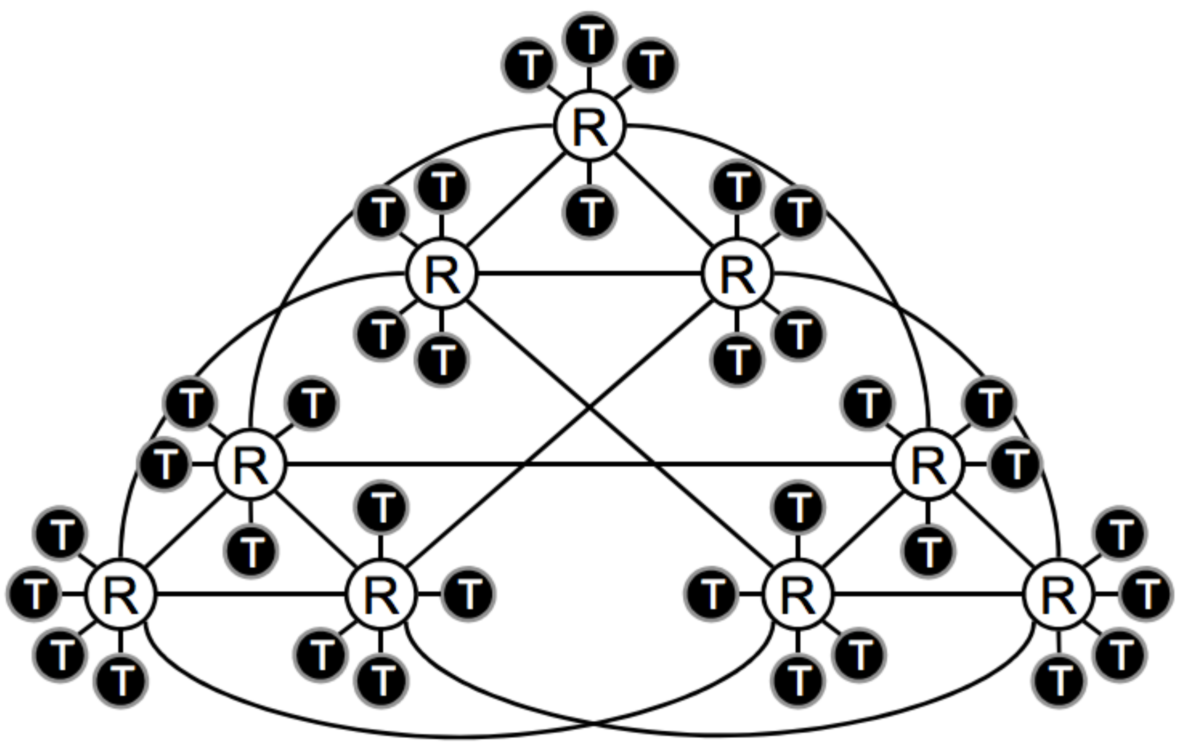


Reachability analysis now passes if we add one inference rule

HyperX Topology

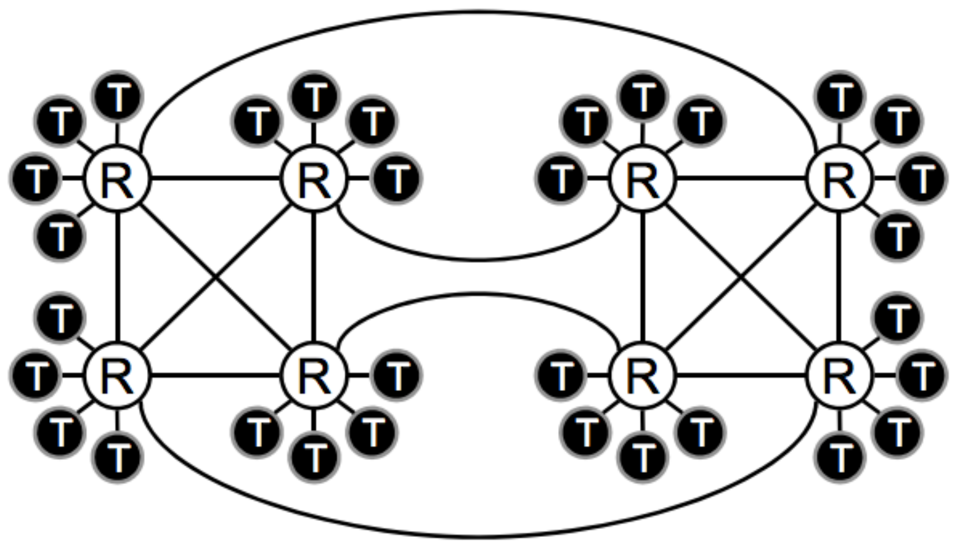


(a) $L = 2, S_1 = 2, S_2 = 4, K = 1, T = 4$

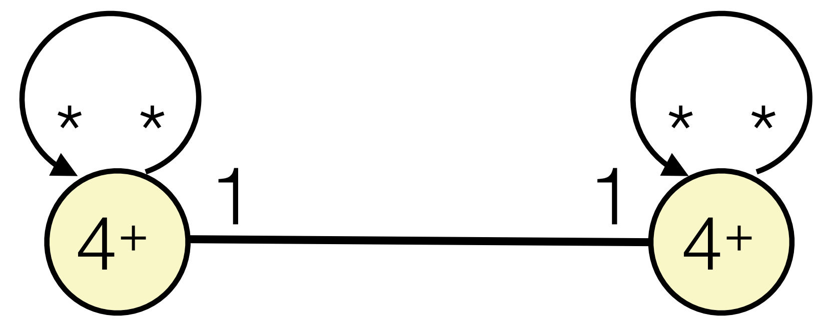


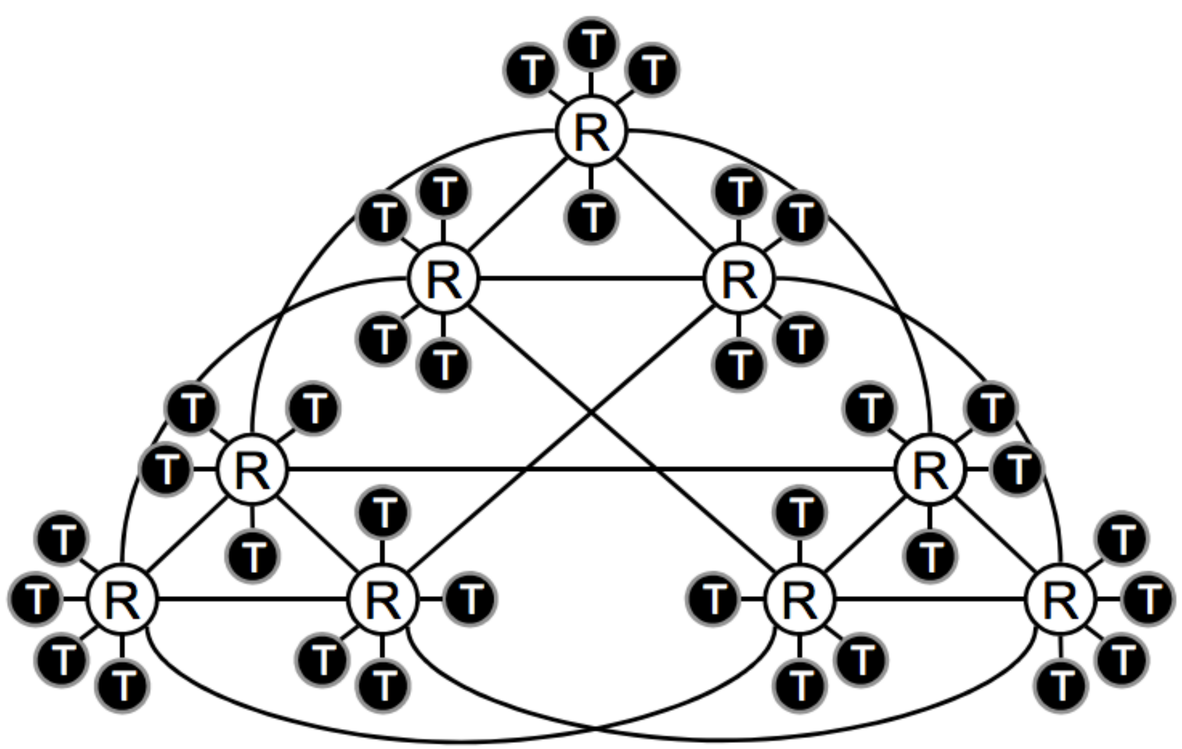
(b) $L = 2, S_1 = 3, S_2 = 3, K = 1, T = 4$

HyperX Topology

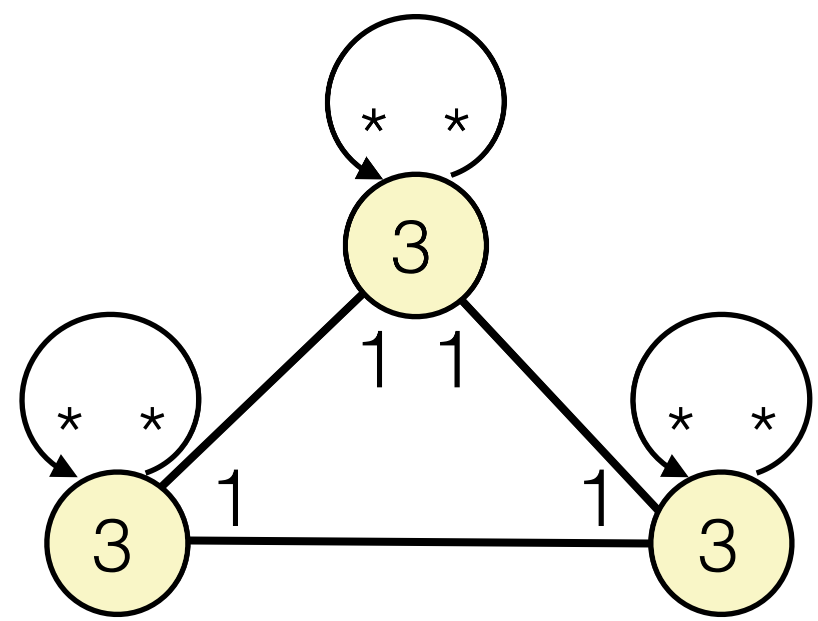


(a) $L = 2, S_1 = 2, S_2 = 4, K = 1, T = 4$

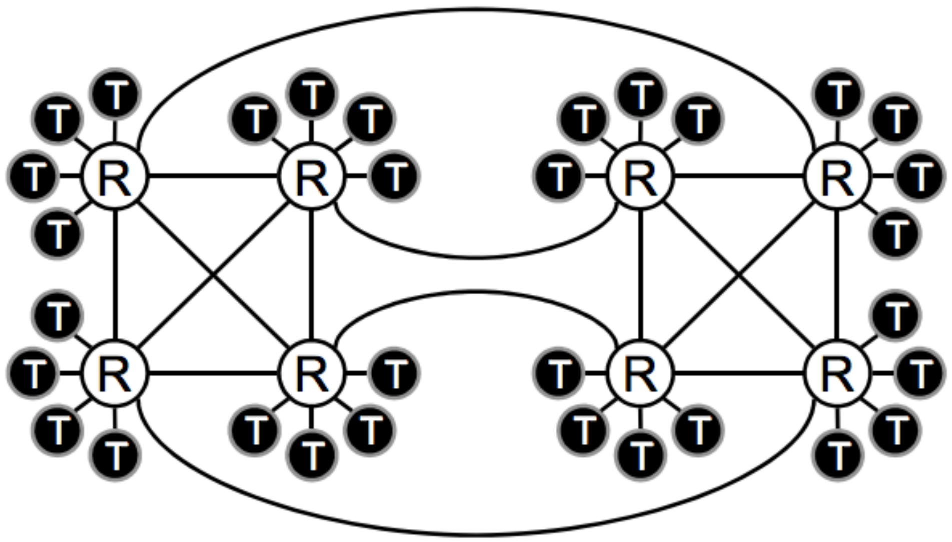




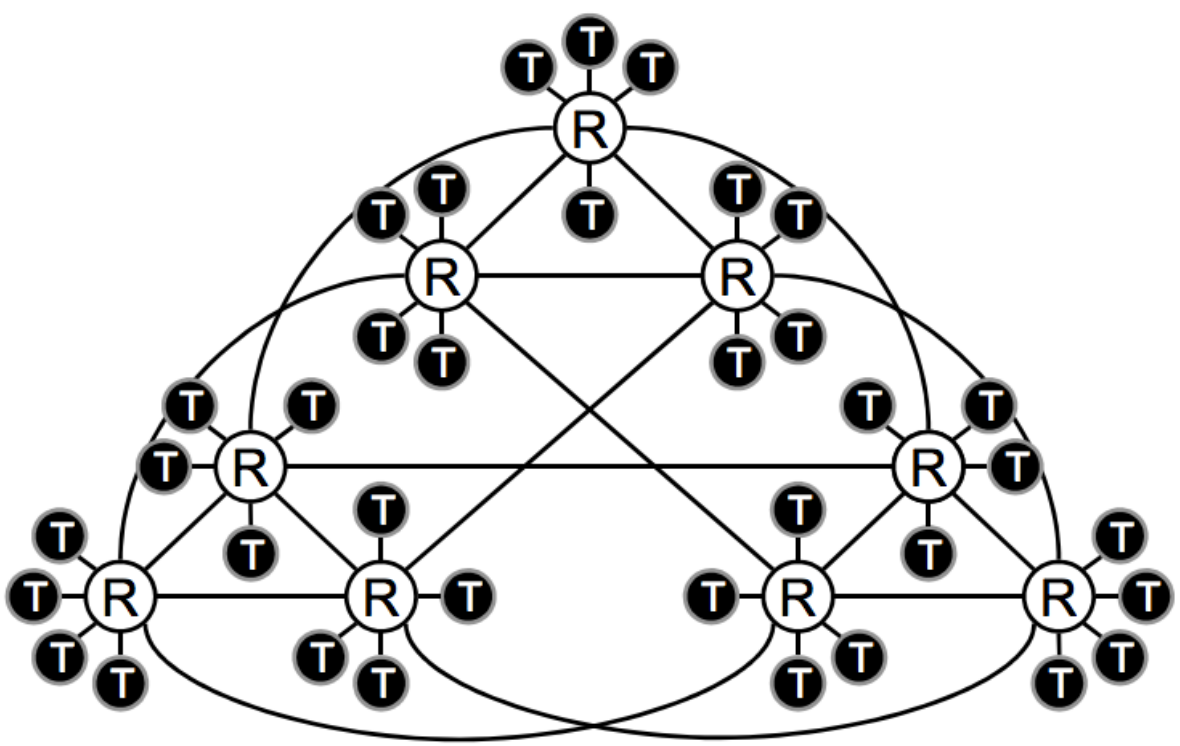
(b) $L = 2, S_1 = 3, S_2 = 3, K = 1, T = 4$



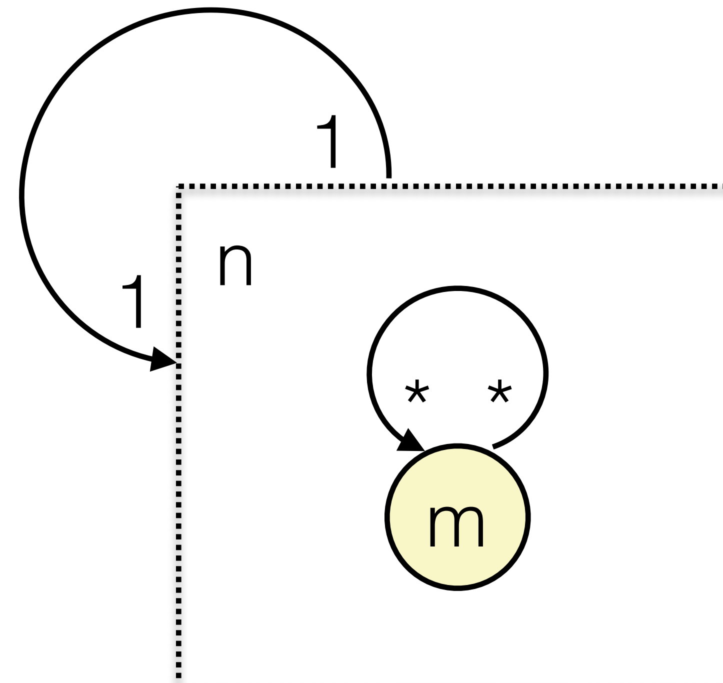
HyperX Topology



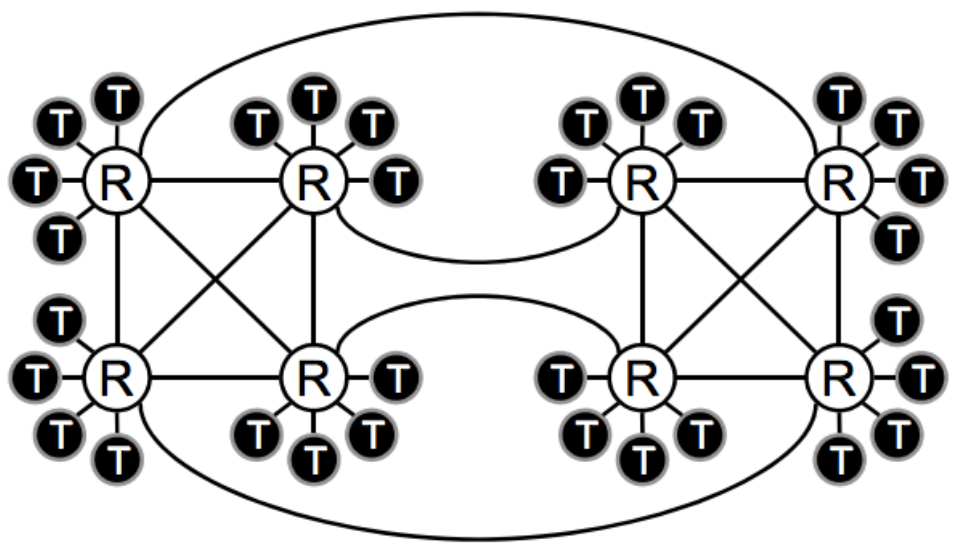
(a) $L = 2, S_1 = 2, S_2 = 4, K = 1, T = 4$



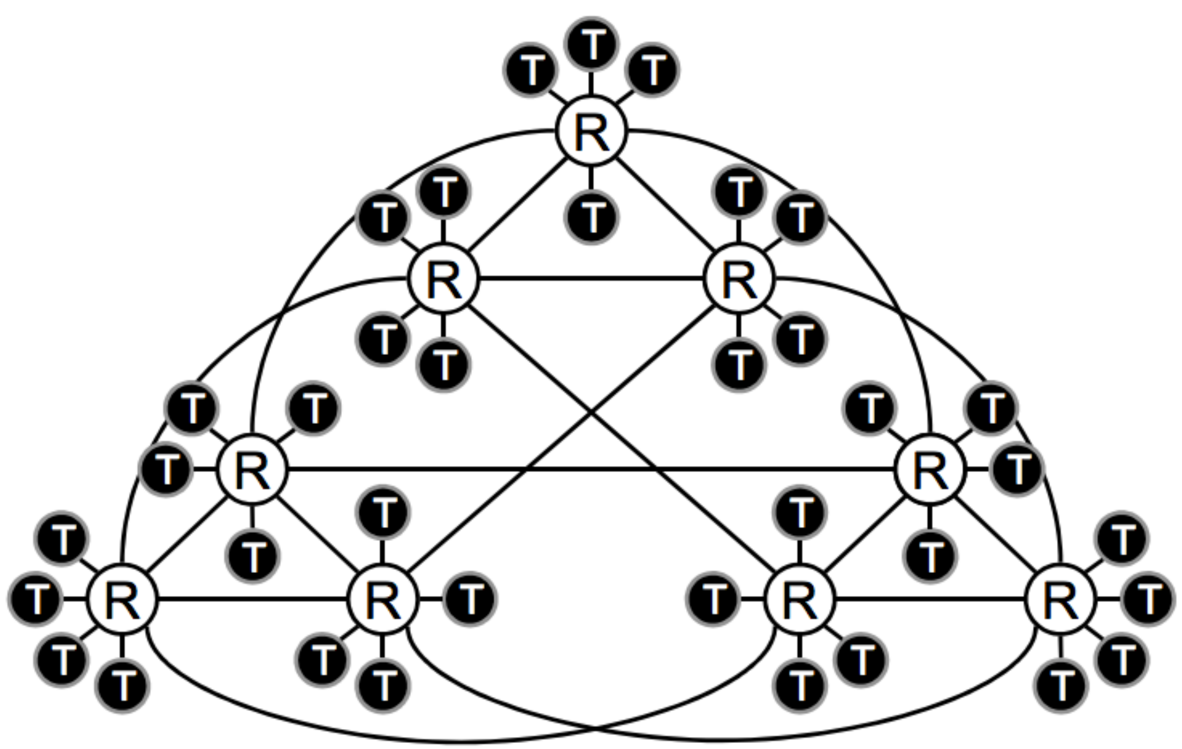
(b) $L = 2, S_1 = 3, S_2 = 3, K = 1, T = 4$



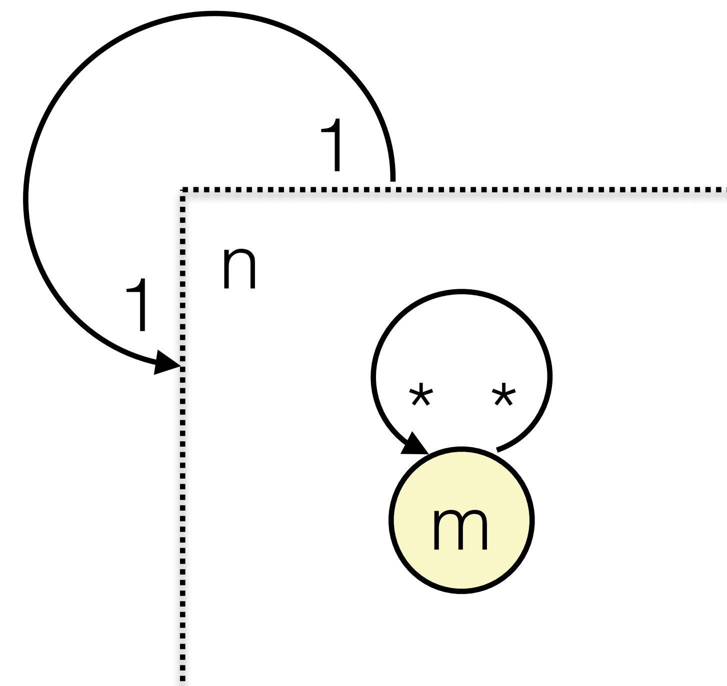
HyperX Topology



(a) $L = 2, S_1 = 2, S_2 = 4, K = 1, T = 4$

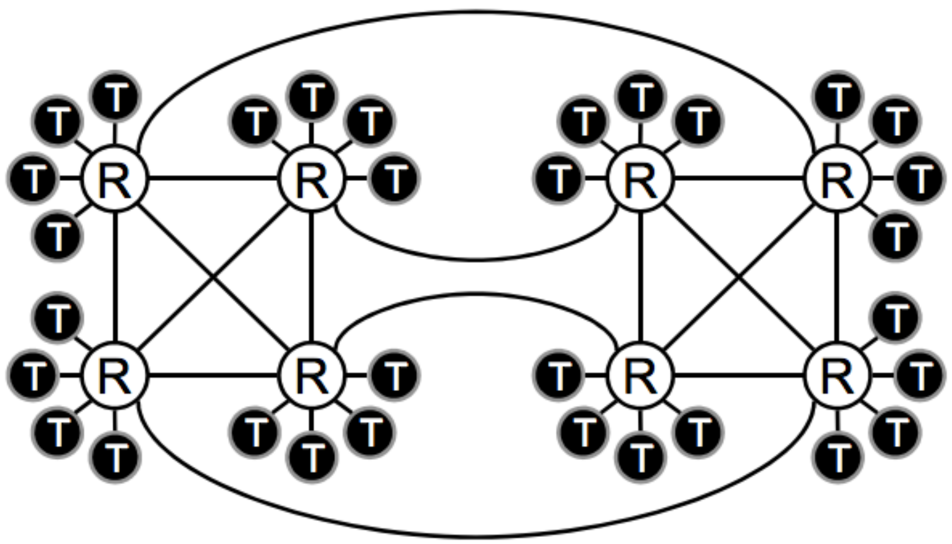


(b) $L = 2, S_1 = 3, S_2 = 3, K = 1, T = 4$

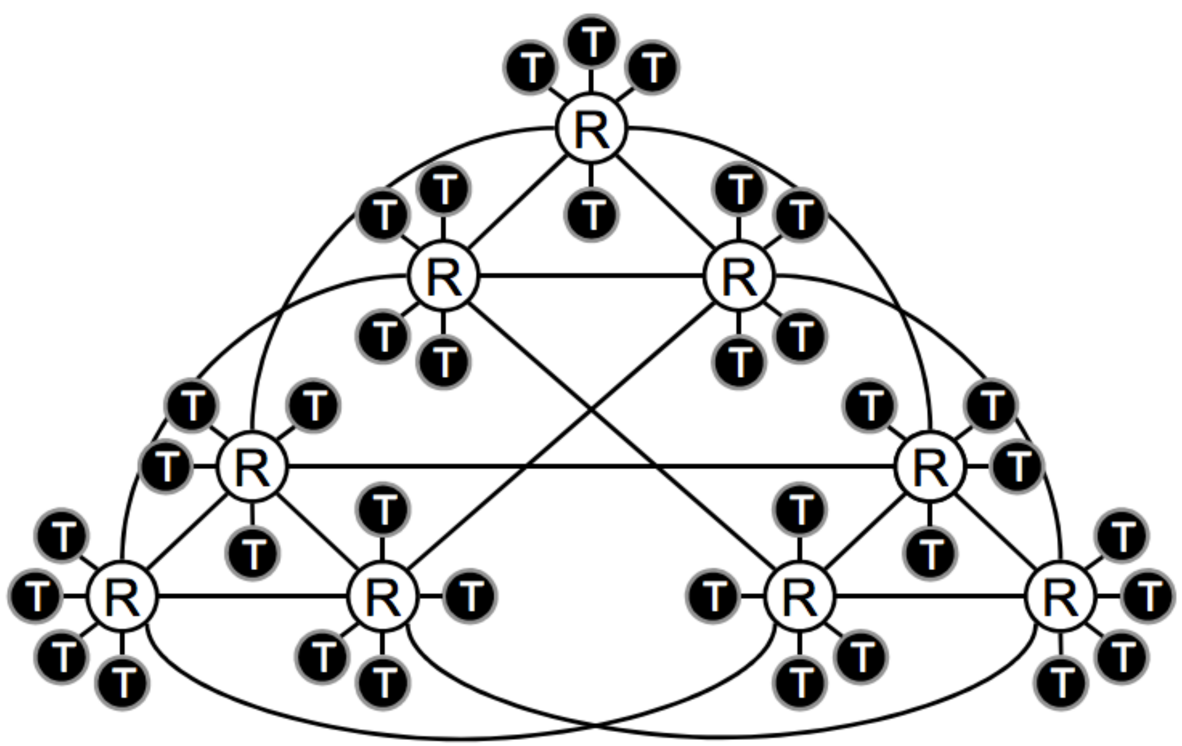


Edge between boxes means the constraint holds for **each pair** of “pods”

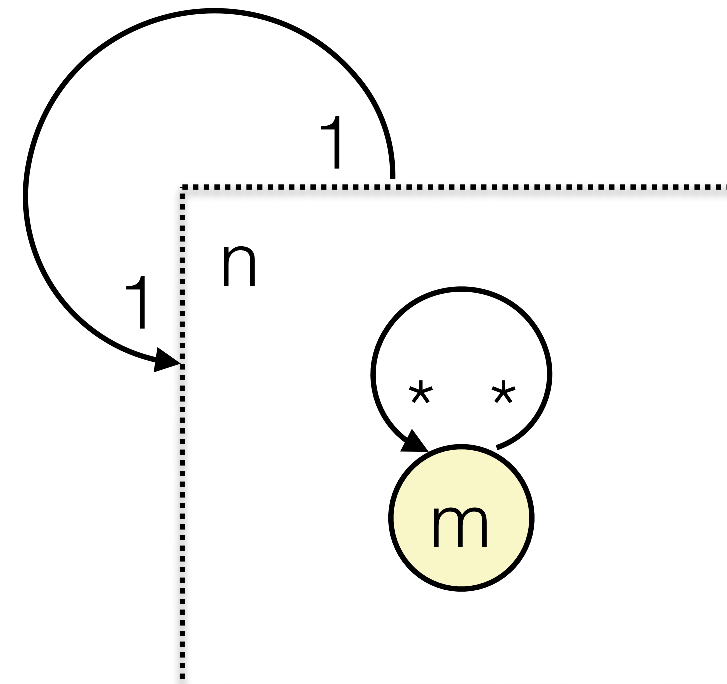
HyperX Topology



(a) $L = 2, S_1 = 2, S_2 = 4, K = 1, T = 4$



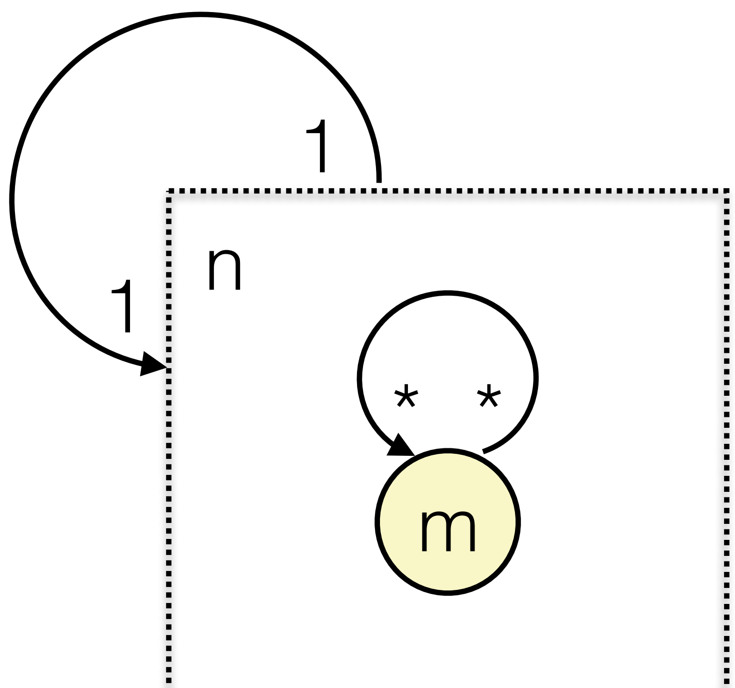
(b) $L = 2, S_1 = 3, S_2 = 3, K = 1, T = 4$



Basically just a macro here, the value **n must be fixed**

Can we make n a variable?

HyperX Topology



Can all nodes in m reach all other nodes in m?

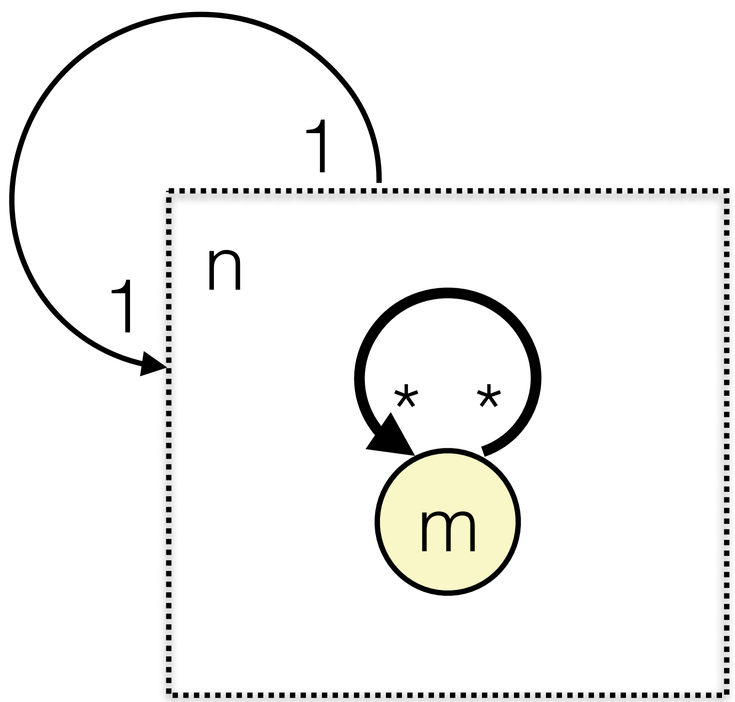
Create 2 variables for m

Initial pod / other pods

m_{init} m_{other}

S

HyperX Topology



Can all nodes in m reach all other nodes in m?

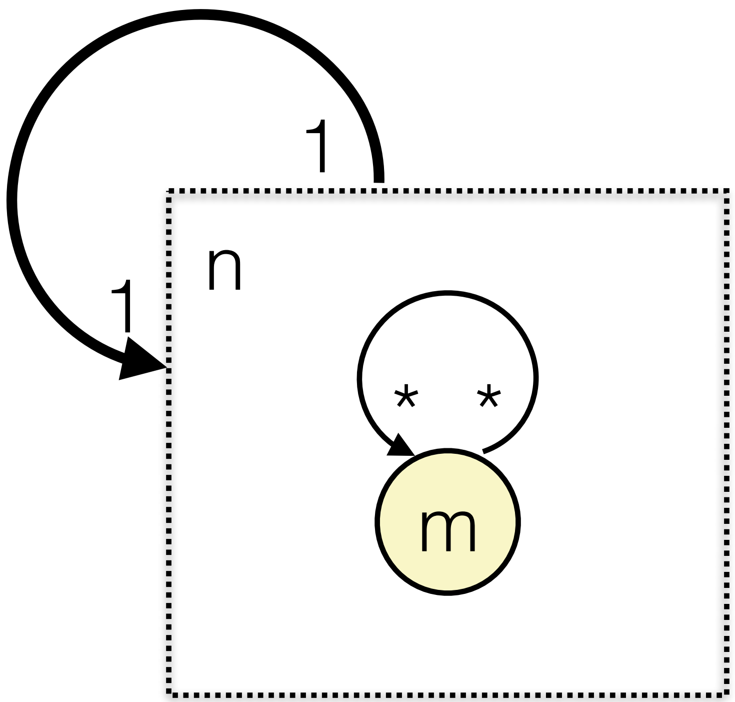
Create 2 variables for m

Initial pod / other pods

m_{init} m_{other}

$$S \downarrow A$$

HyperX Topology



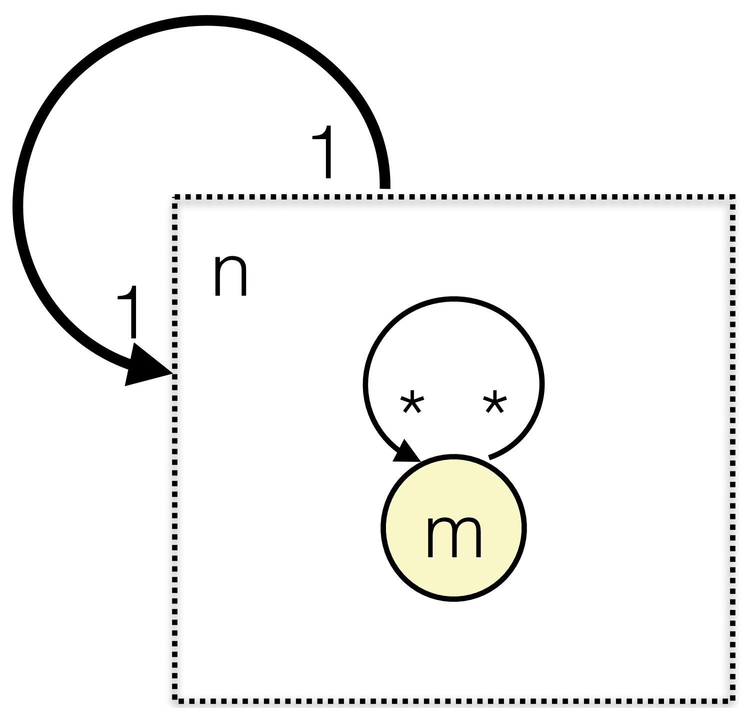
Can all nodes in m reach all other nodes in m?

Create 2 variables for m

Initial pod / other pods

$$\begin{array}{ccc} m_{init} & & m_{other} \\ S & & \\ \downarrow & & \\ A & \longrightarrow & A \end{array}$$

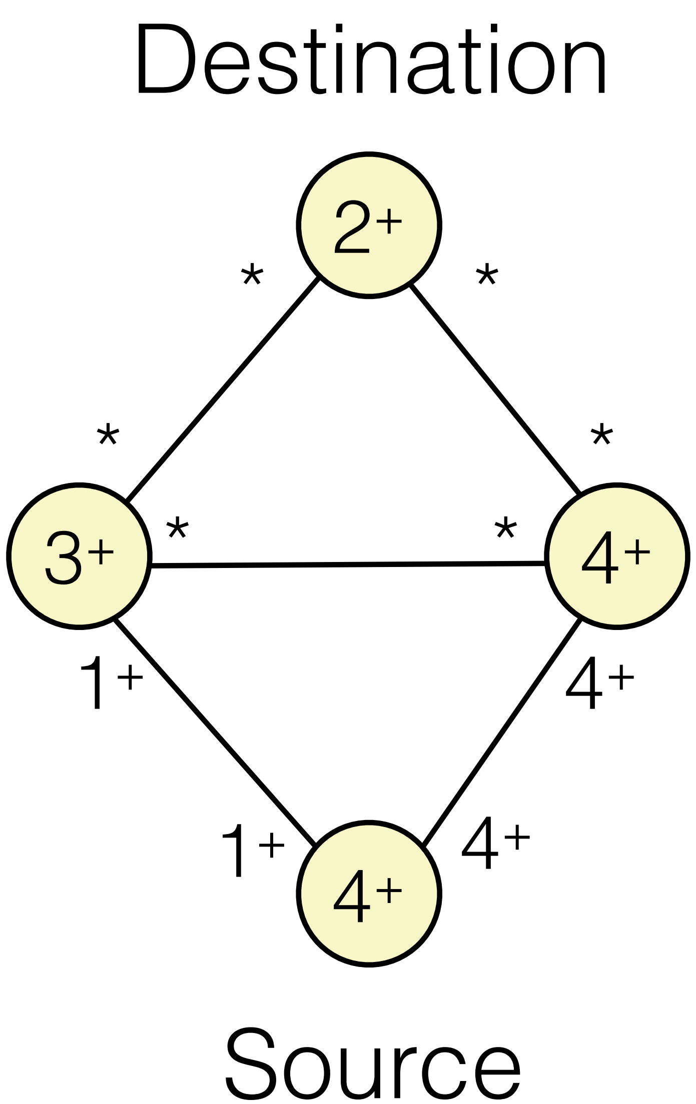
HyperX Topology



But not clear how this abstract topology lifts to the PG with hierarchy

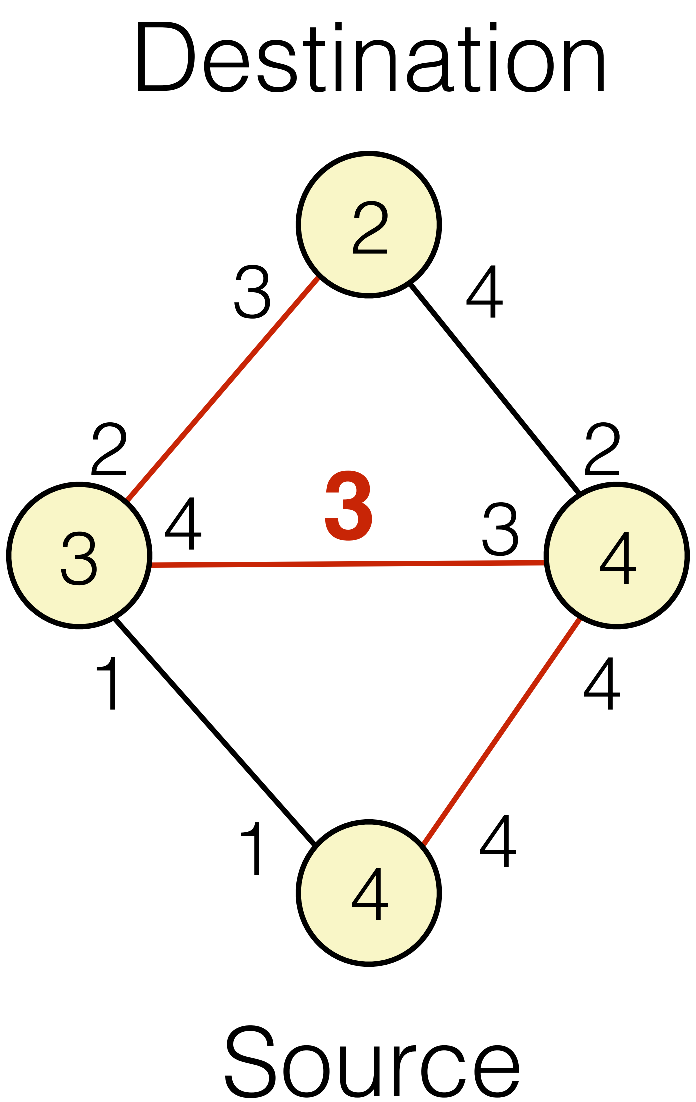
Symbolic Failure Analysis

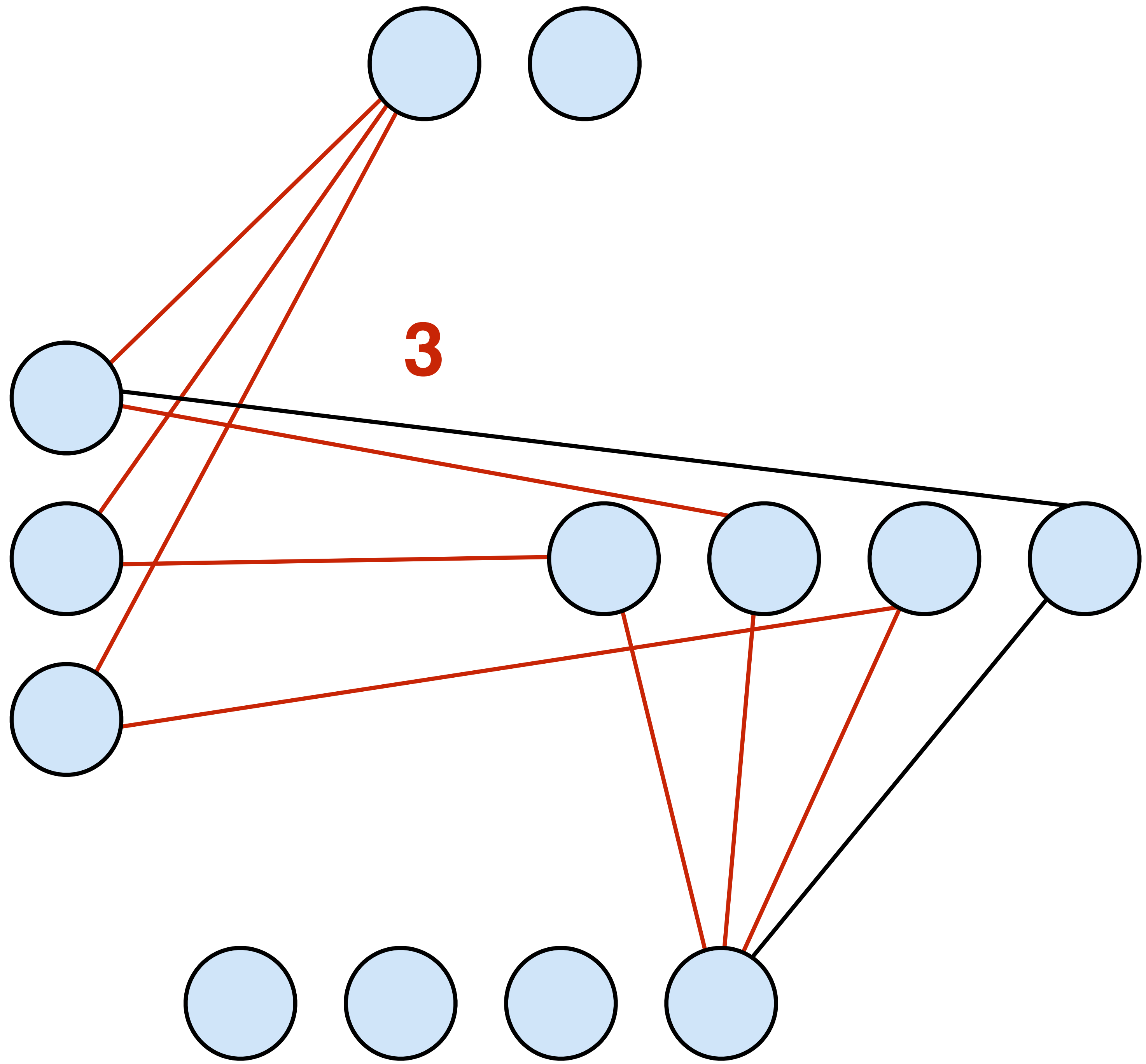
Reachability under k-failures



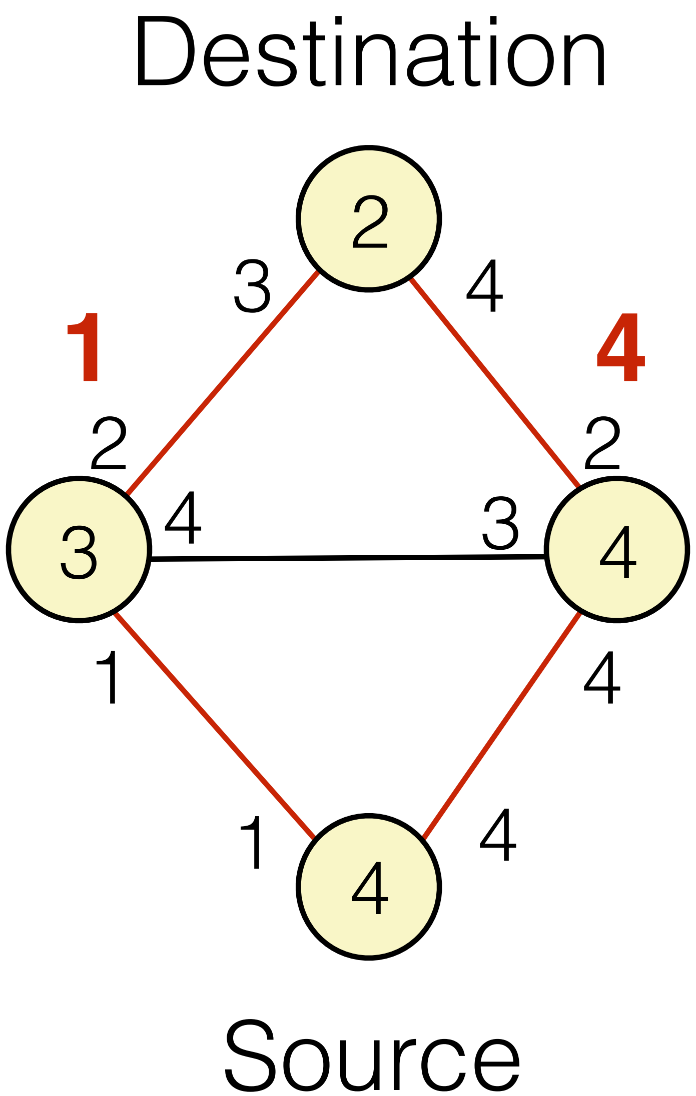
Idea from before:
generate a “worst” case concrete topology, and find a lower bound on the min-cut of this topology

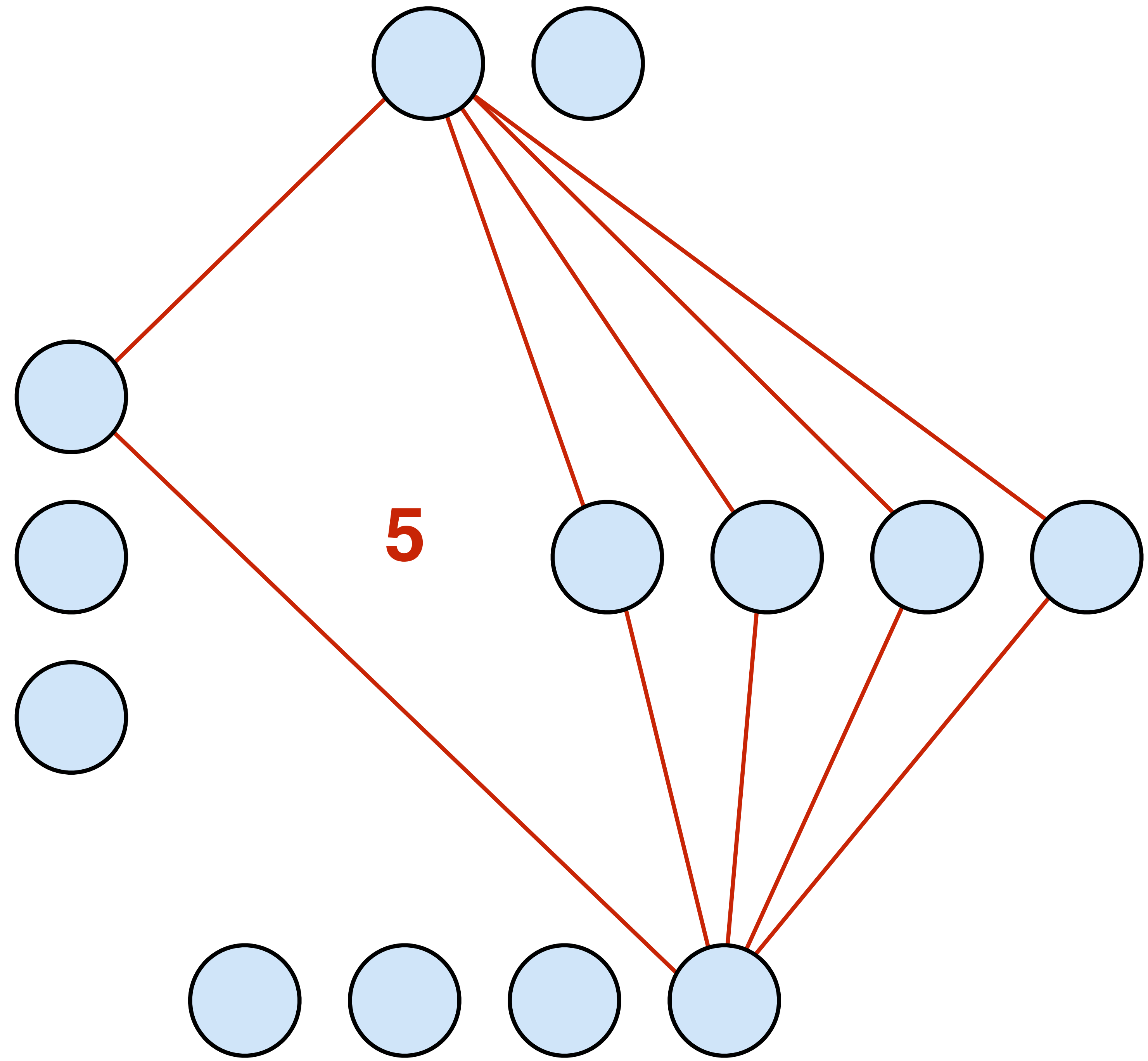
Reachability under k-failures





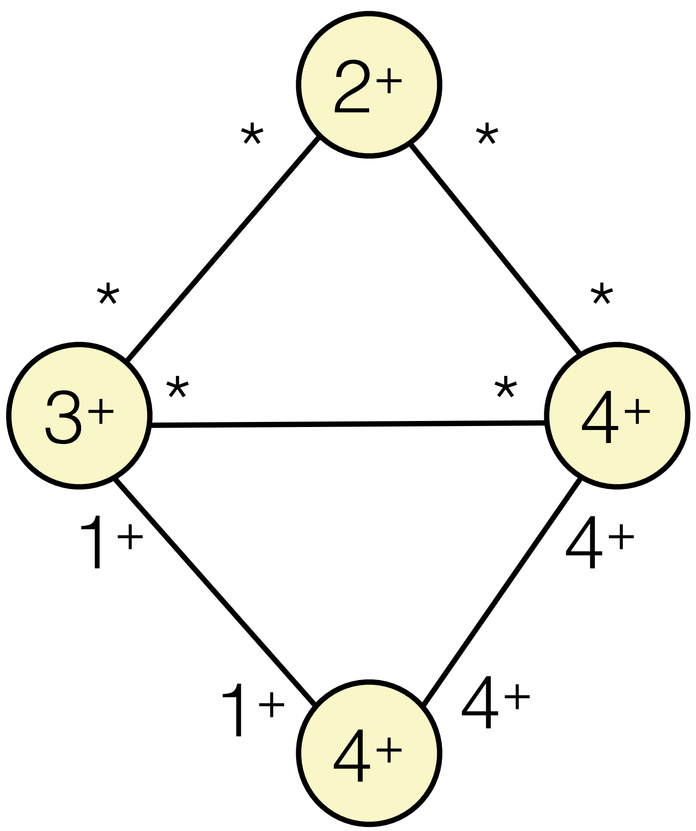
Reachability under k-failures





Reachability under k-failures

Destination



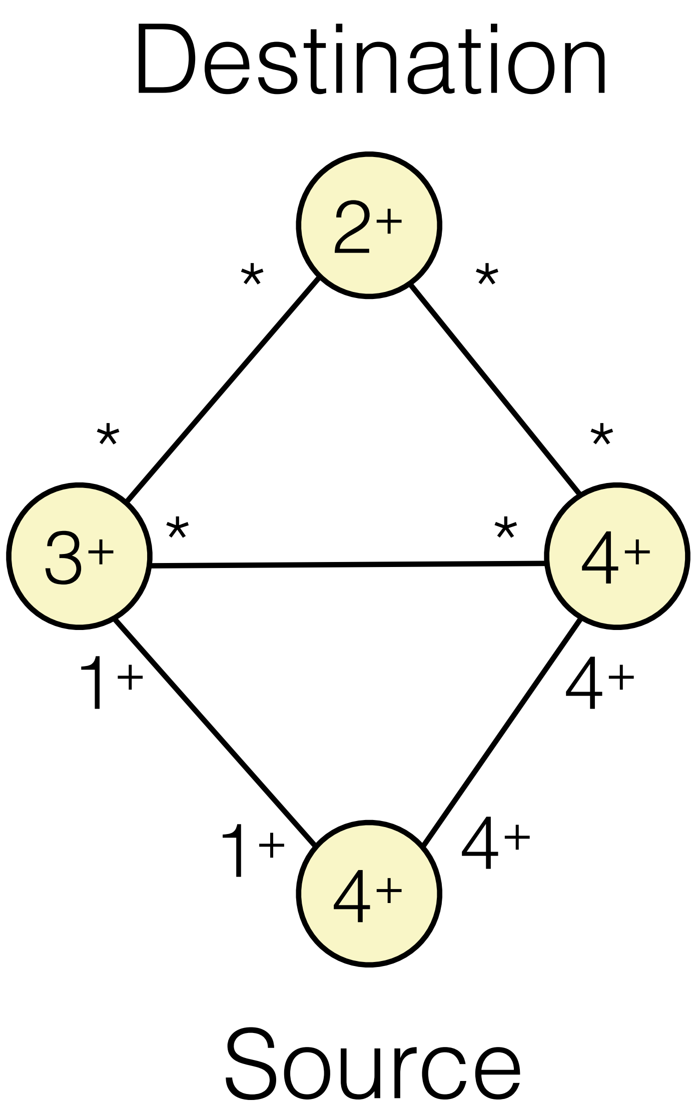
Source

Idea from before:
generate a “worst” case concrete topology, and find a lower bound on the min-cut of this topology

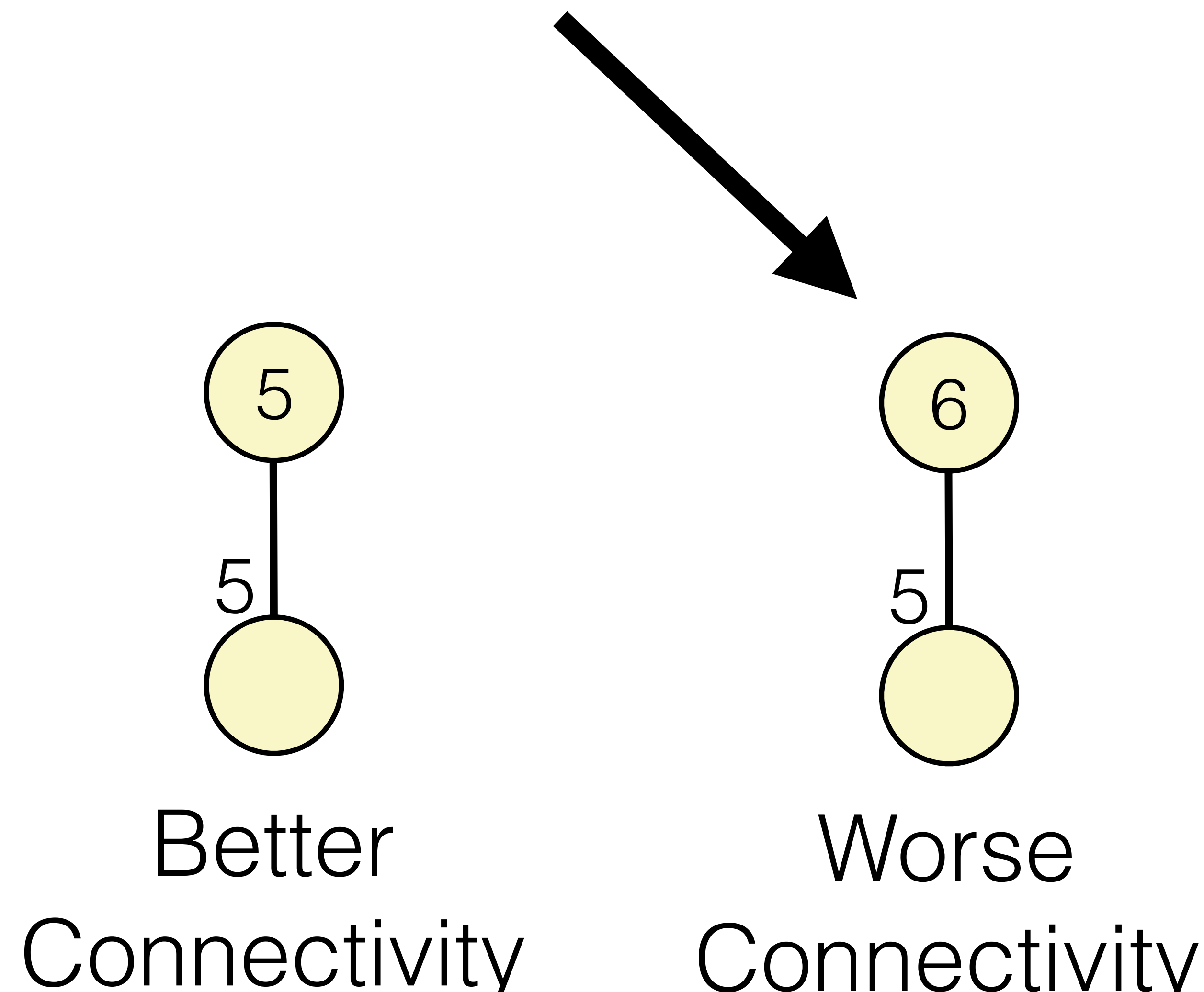
Better Connectivity

Worse Connectivity

Reachability under k-failures

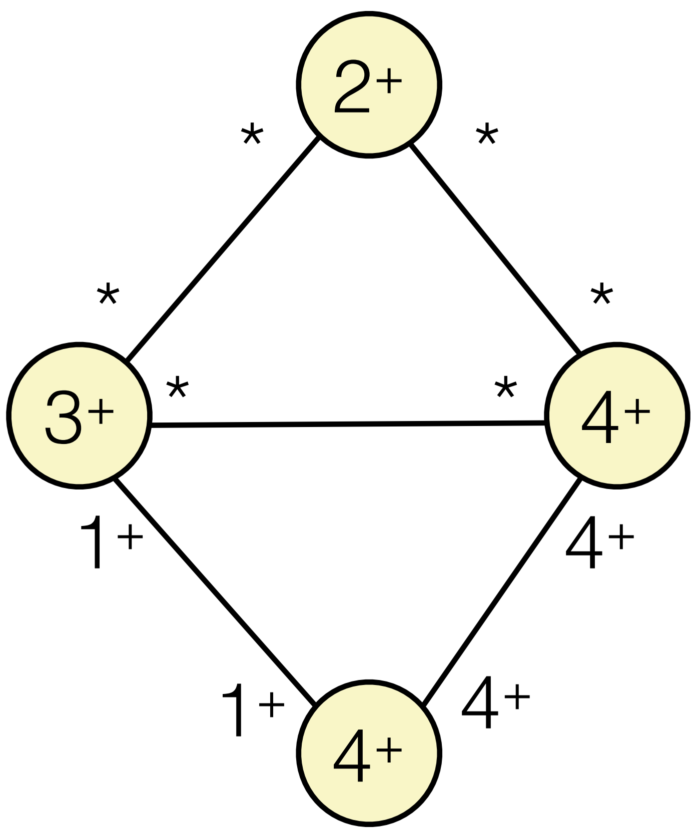


A topology with more nodes can be less connected



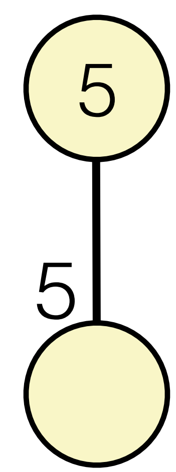
Reachability under k-failures

Destination

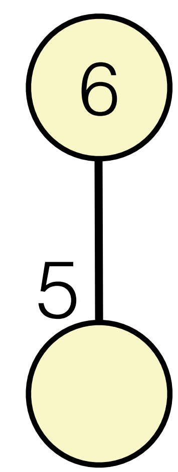


Source

Need to lower bound failures symbolically

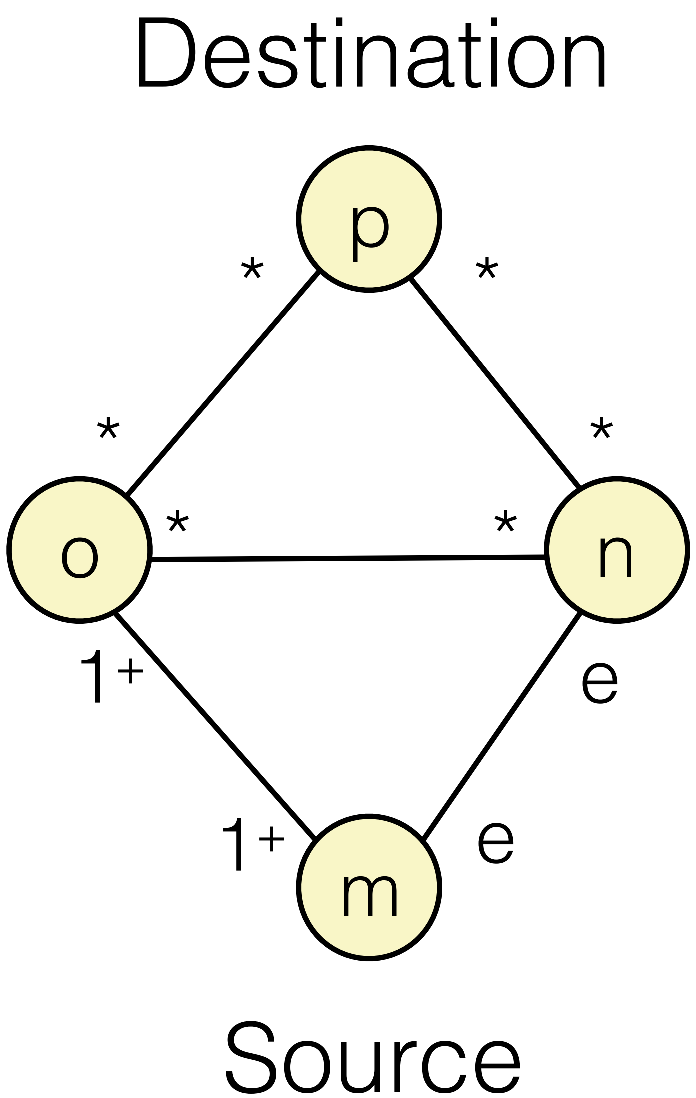


Better Connectivity



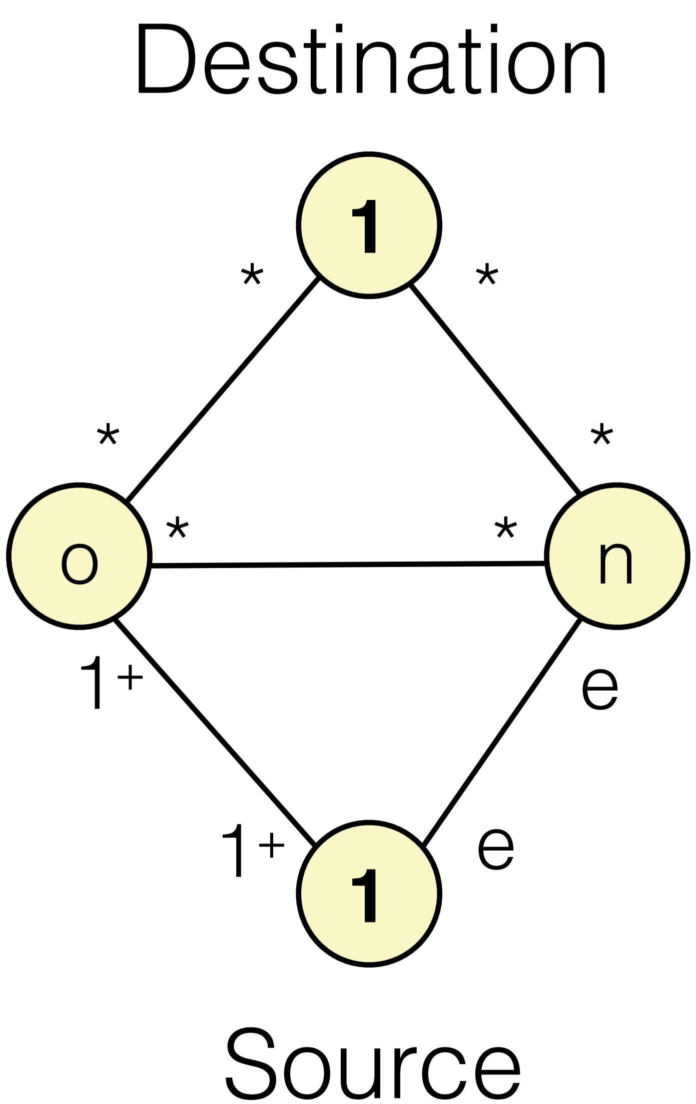
Worse Connectivity

Reachability under k-failures



$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

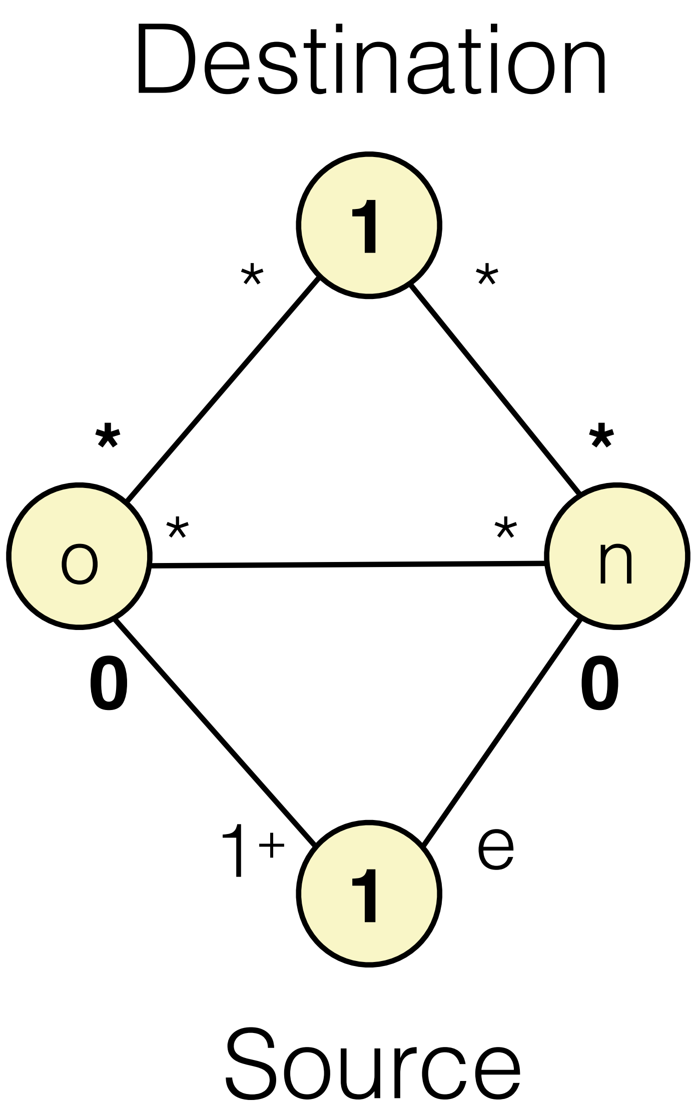
Reachability under k-failures



$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

First we rewrite the topology to consider only a single source and destination node

Reachability under k-failures



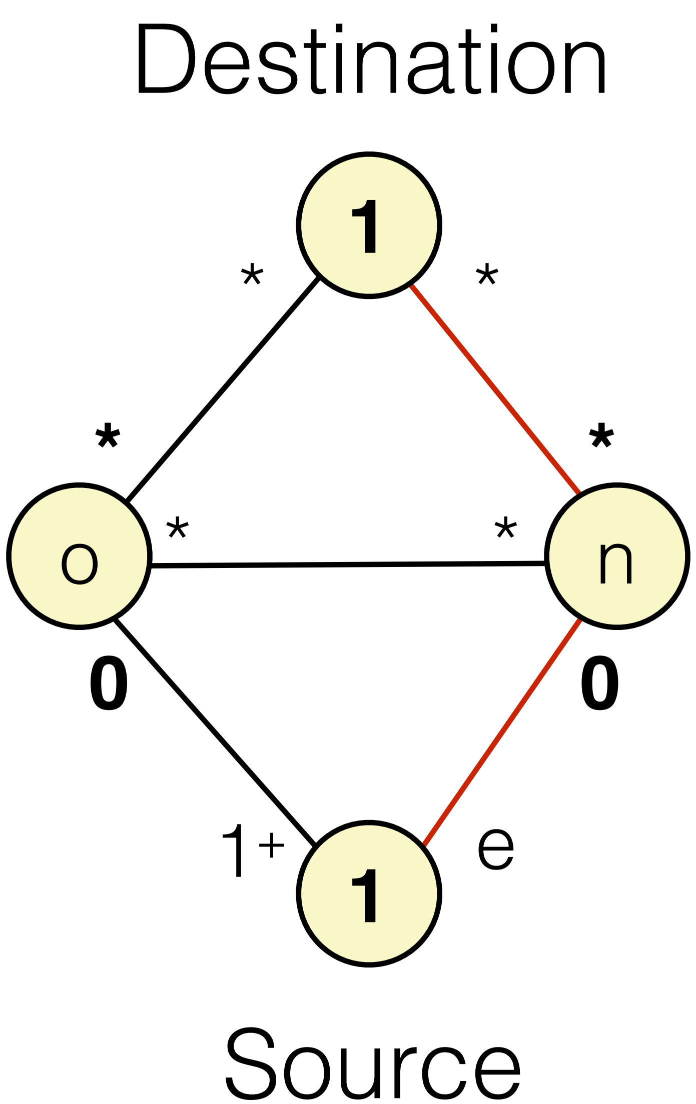
$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

First we rewrite the topology to consider only a single source and destination node

Conservatively reduce edge count — more on how to do this better later

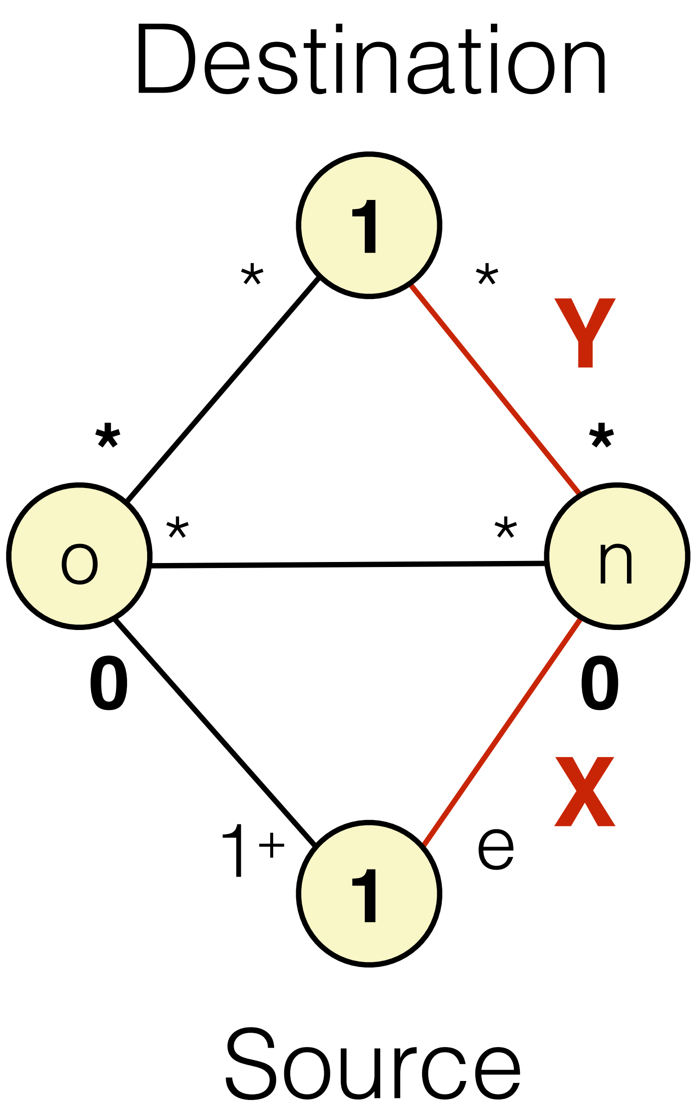
Reachability under k-failures

Step 1:
Pick a path from Source to Destination



$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

Reachability under k-failures

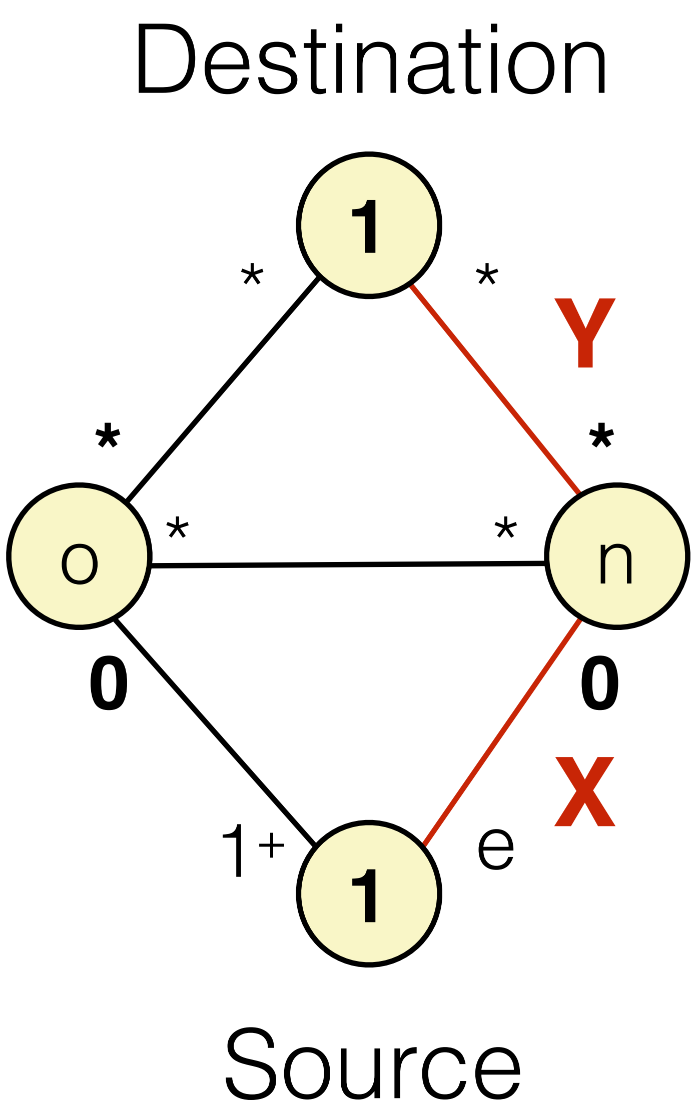


$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

Step 1:
Pick a path from Source to Destination

Step 2:
Label each edge with a unique variable

Reachability under k-failures



$p \geq 2$

$e \geq 4$

$o \geq 3$

$n \geq 4$

$m \geq 4$

Step 1:
Pick a path from Source to Destination

Step 2:
Label each edge with a unique variable

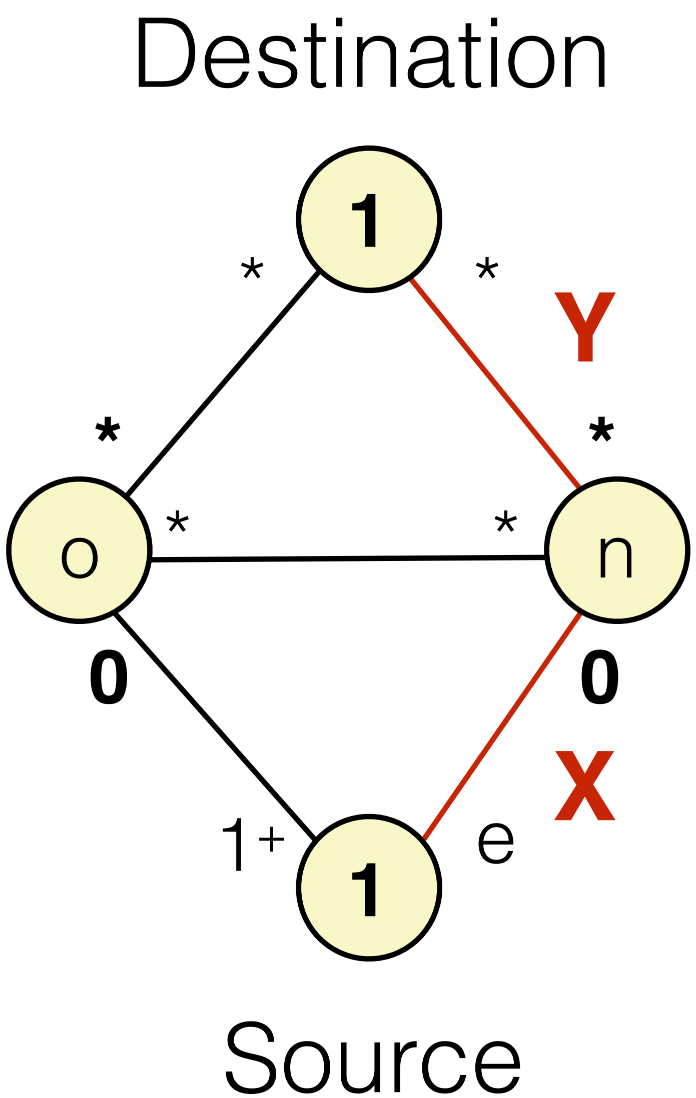
Step 3:
Compute number of disjoint paths symbolically

Step 4:
Minimize the resulting ILP

Reachability under k-failures

How disjoint paths to n?

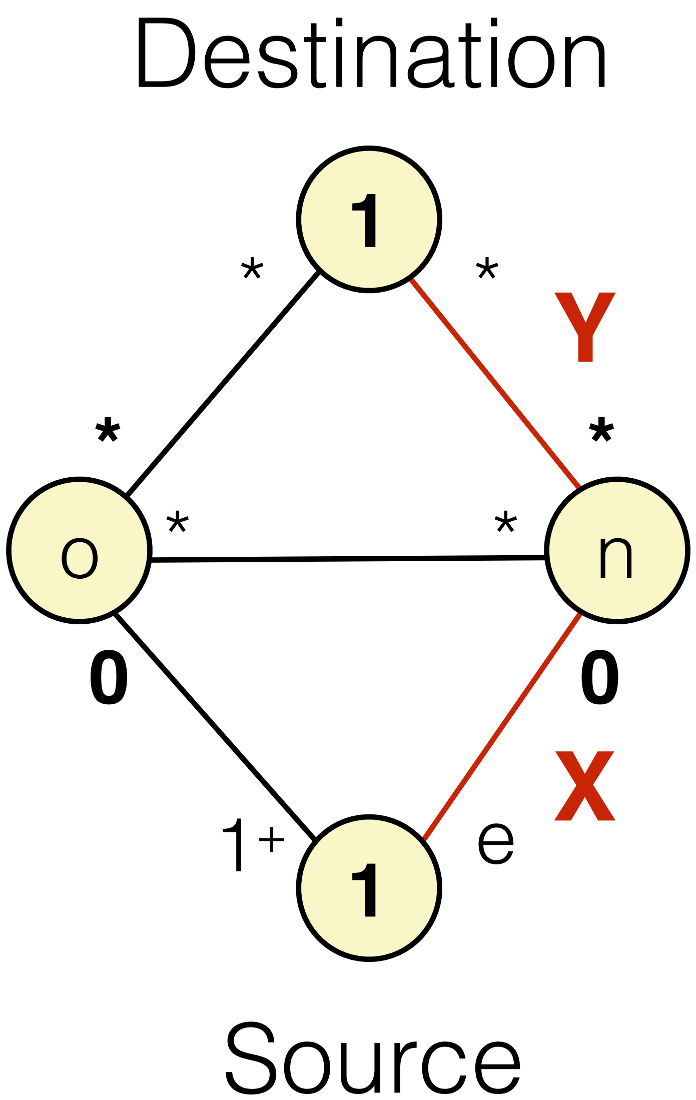
$X = \min(e,n)$



$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

Reachability under k-failures

How disjoint paths to Destination?



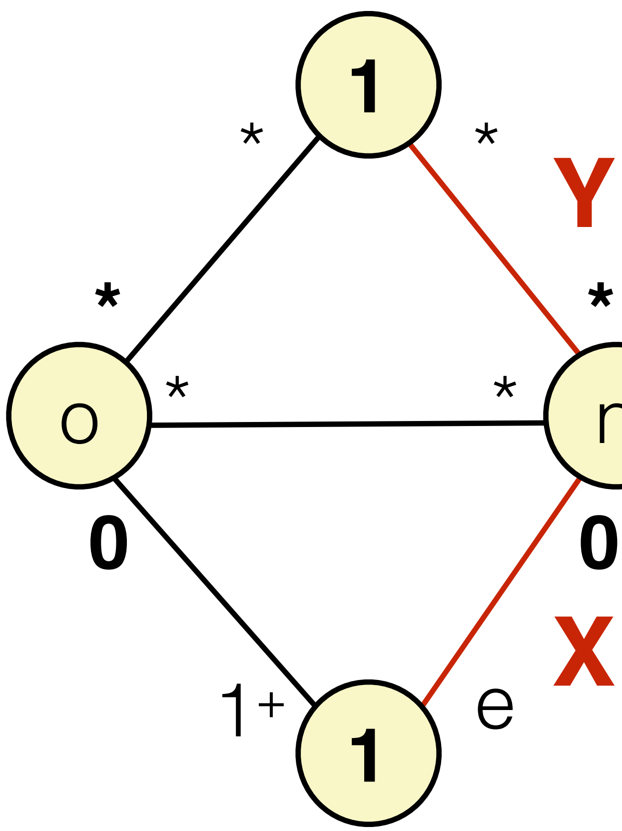
$$X = \min(e,n)$$

$$Y = \min(X,\infty)$$

$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

Reachability under k-failures

Destination



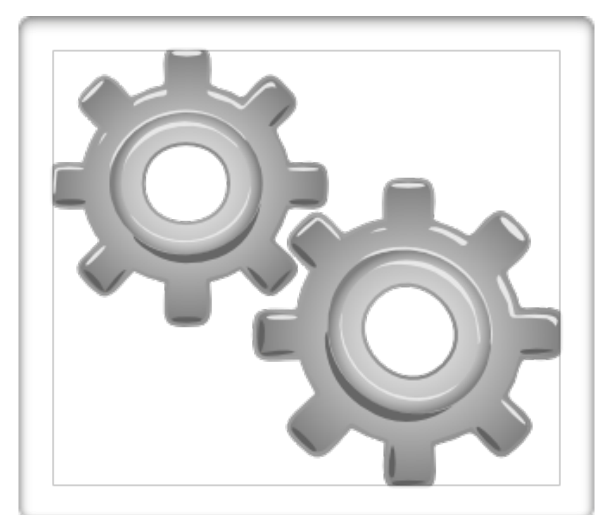
Source

$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

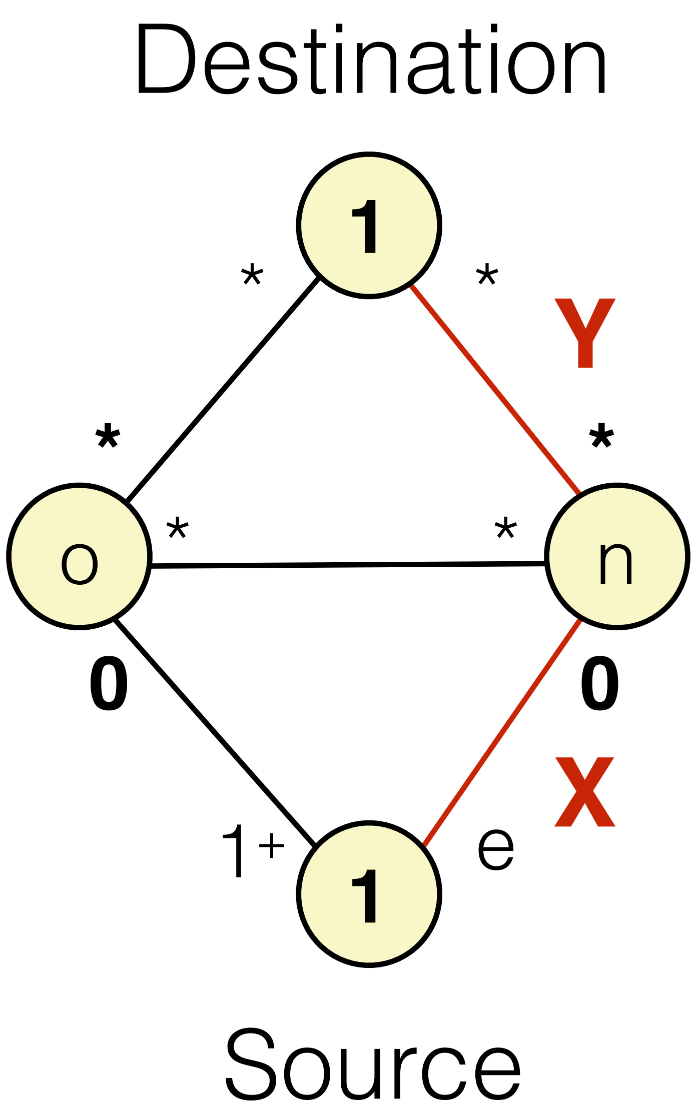
$X = \min(e,n)$

$Y = \min(X,\infty)$

ILP Solver



Reachability under k-failures



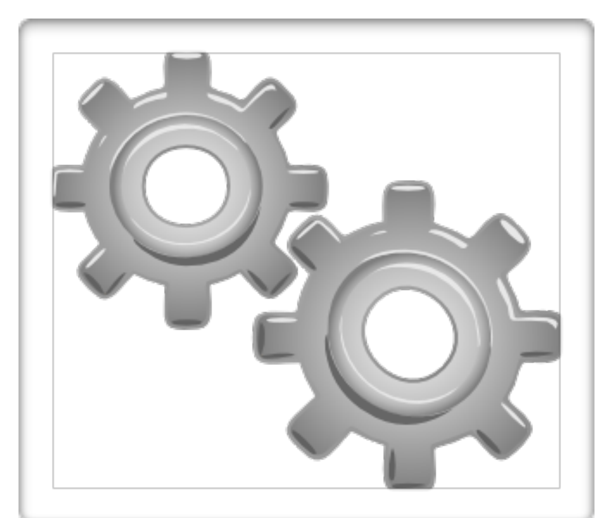
Objective:
minimize Y

$X = \min(e,n)$

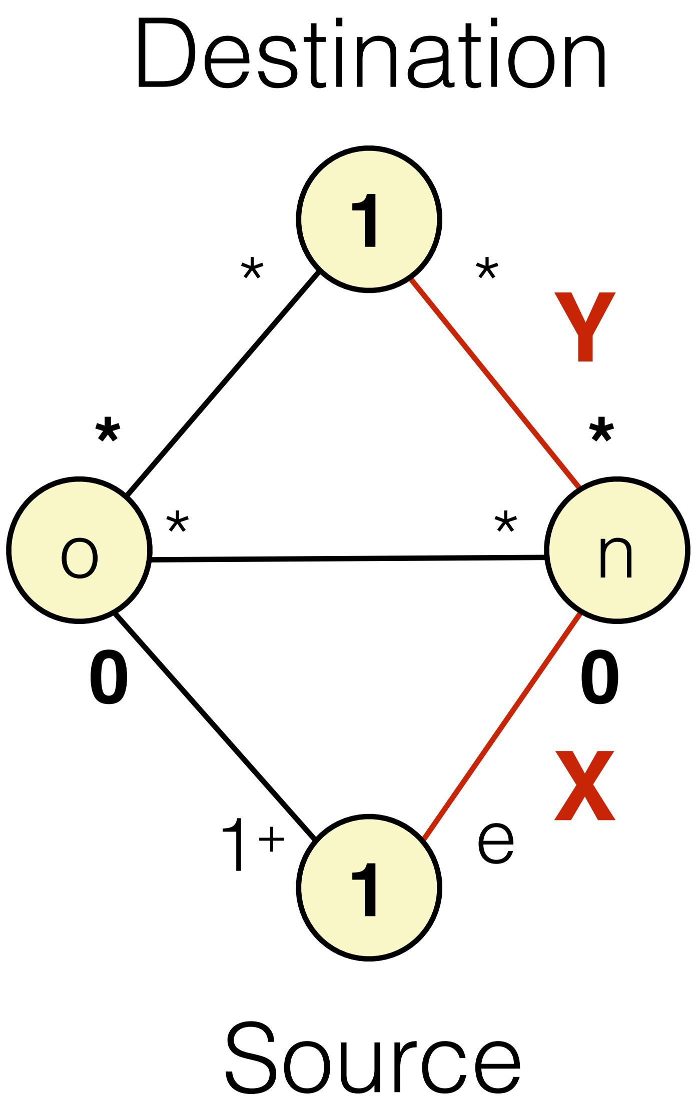
$Y = \min(X,\infty)$

$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

ILP Solver



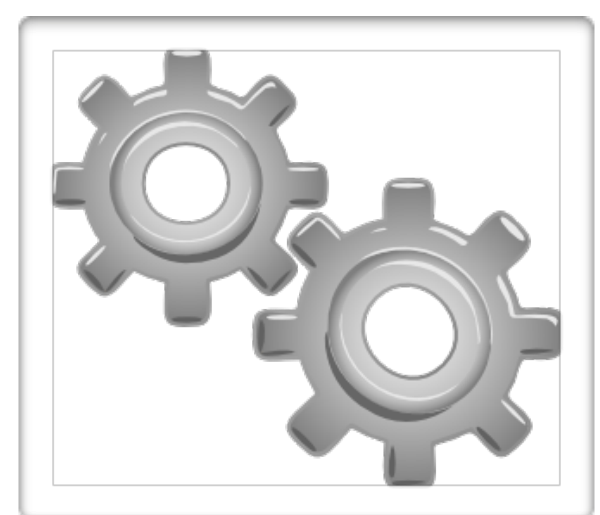
Reachability under k-failures



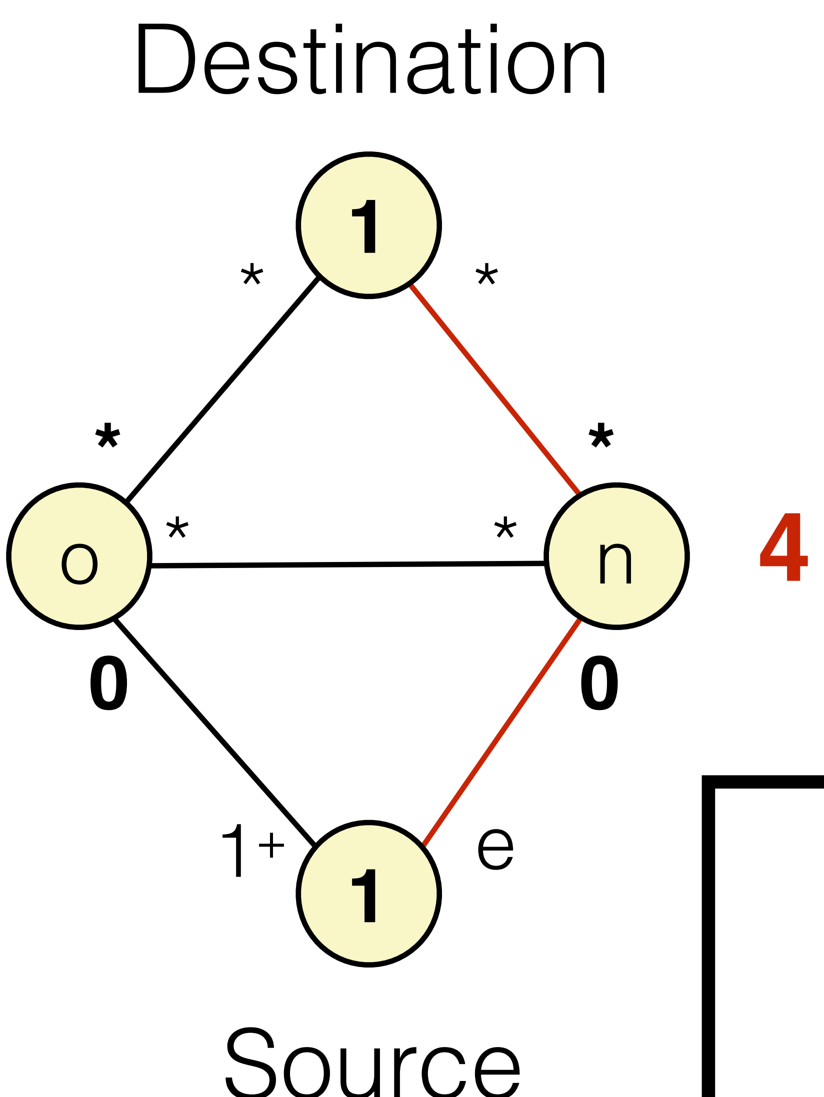
$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

Answer: 4

ILP Solver



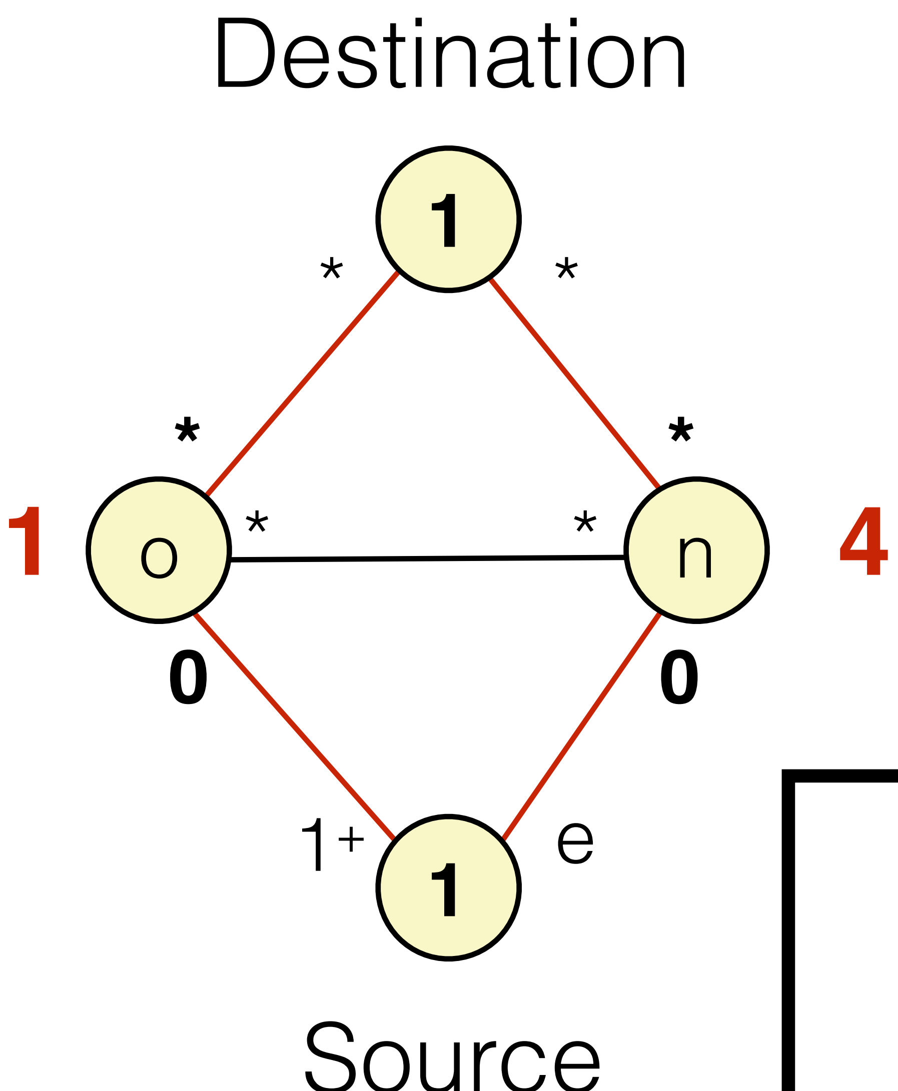
Reachability under k-failures



$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

Repeat until no path remains to the destination

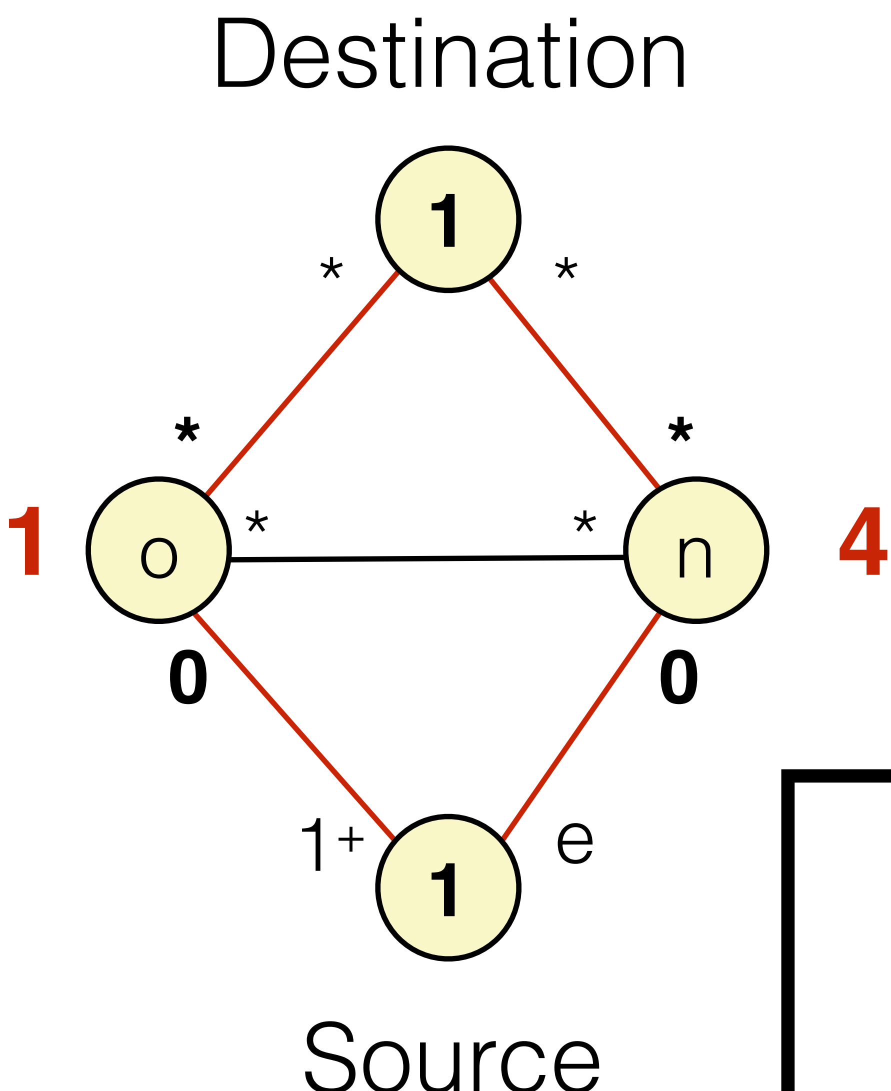
Reachability under k-failures



Repeat until no path remains to the destination

$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

Reachability under k-failures

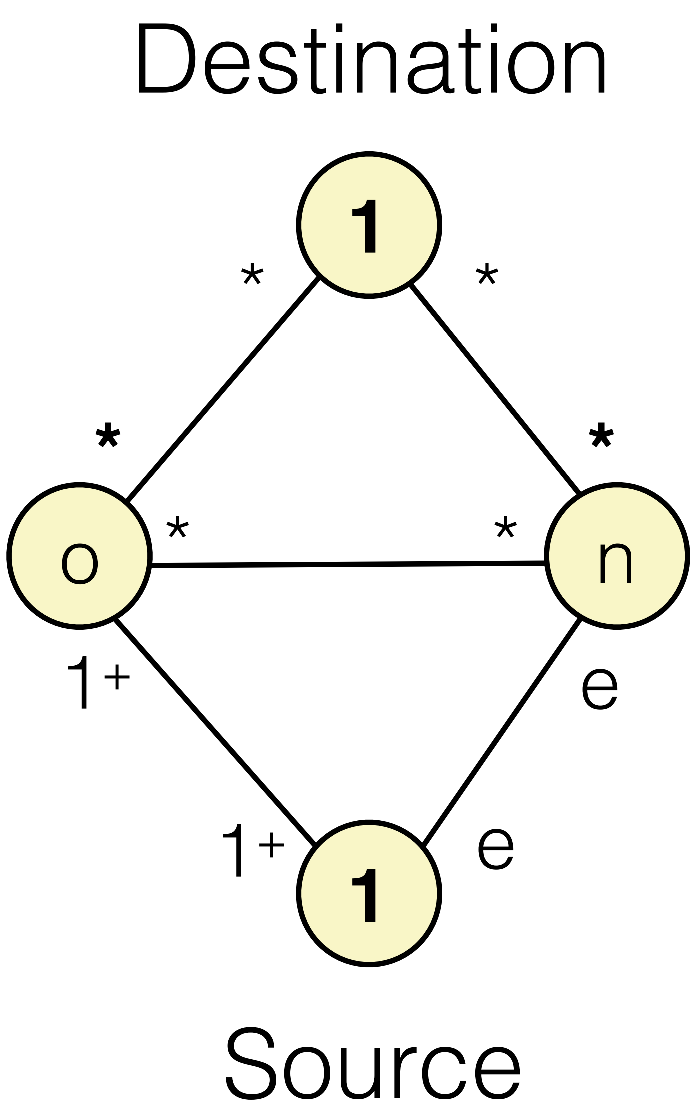


$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

Repeat until no path remains to the destination

Note: failures along the paths are considered independently

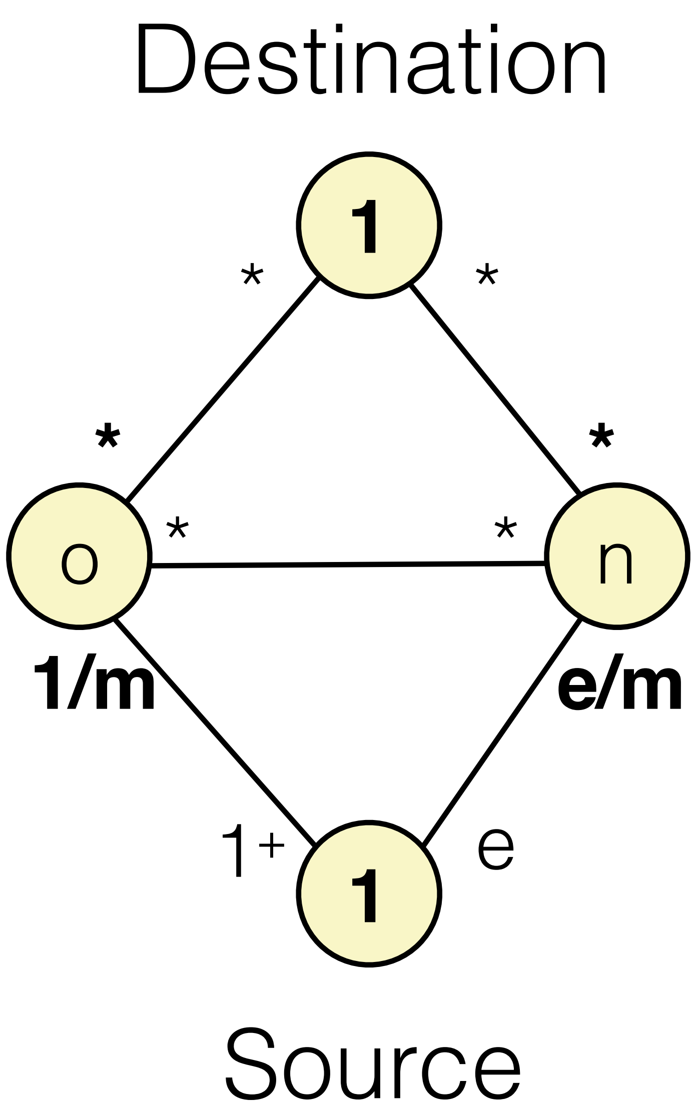
Reachability under k-failures



$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

A Better Approximation: Reduce the edge counts in a more reasonable way than just setting to 0.

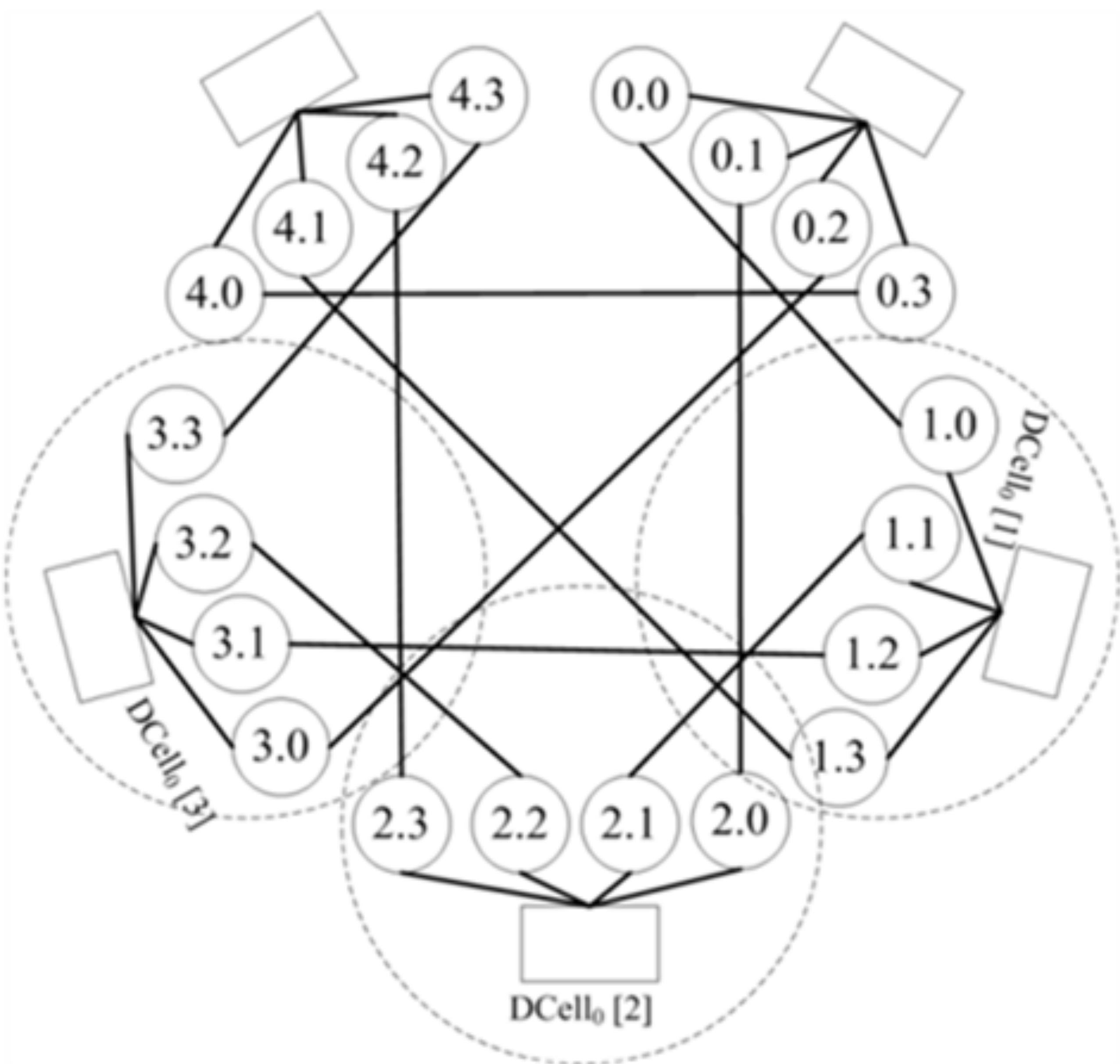
Reachability under k-failures



$p \geq 2$
$e \geq 4$
$o \geq 3$
$n \geq 4$
$m \geq 4$

A Better Approximation: Reduce the edge counts in a more reasonable way than just setting to 0.

DCell Topology



I still don't know how to abstract this topology :(

Invariant is an existential but we can only use foralls

Propane LP Extension

Propane - Local Preferences

Encoding Local Preference:
Sometimes you want a local preference rather than a global preference in Propane:

- Easier to specify LP, MEDs
- Easier for operators new to lang.
- Can compose nicer
- Load balancing can be better

Propane - Local Preferences

lp(X, P1 >> P2)
lp(Y, P1 >> P2)

Propane - Local Preferences

lp(X, P1 >> P2)
lp(Y, P1 >> P2)

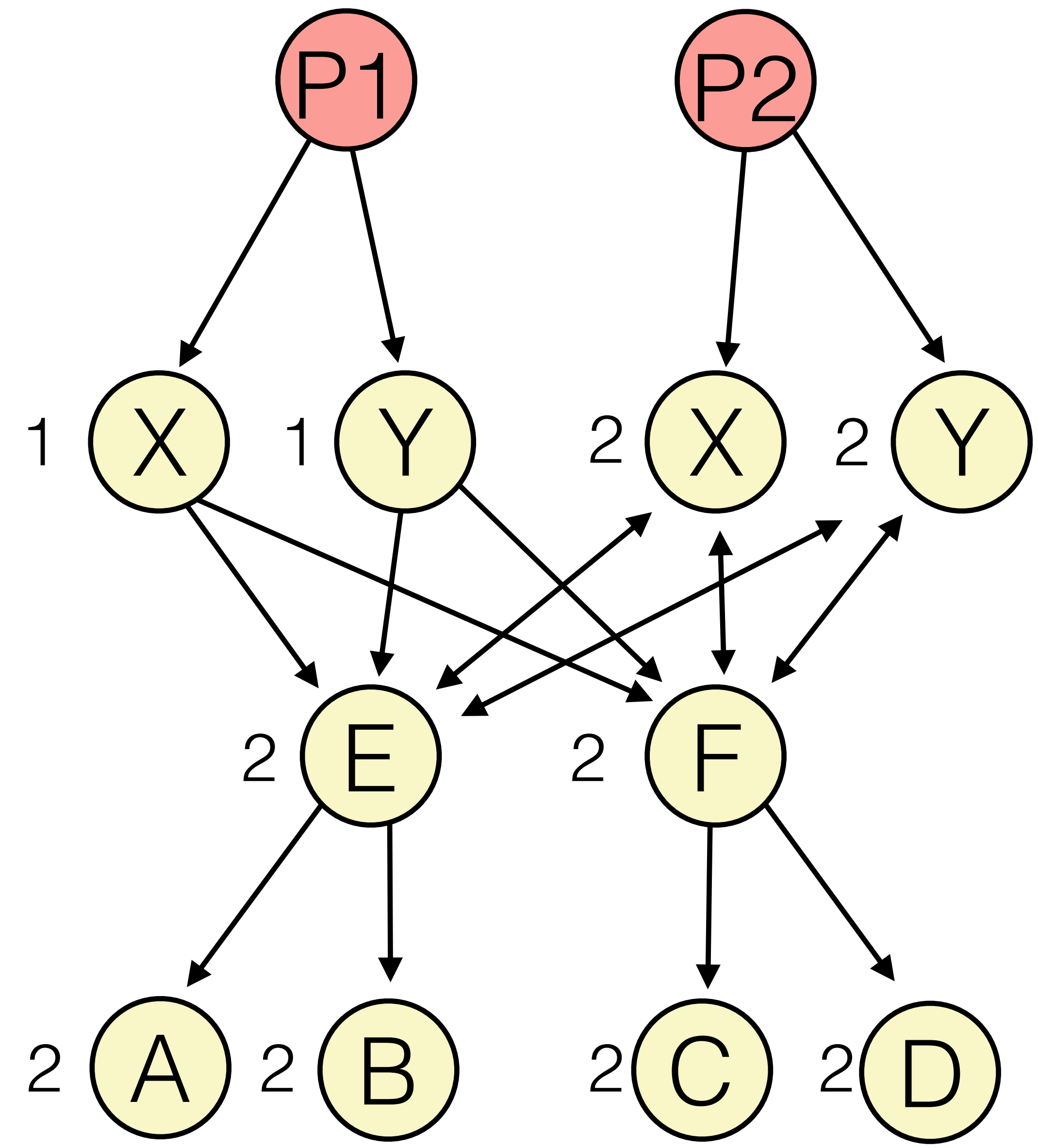
$\downarrow$

(.*; P1; X) >> (.*; P2; X)
∪ (.*; P1; Y) >> (.*; P2; Y)

Propane - Local Preferences

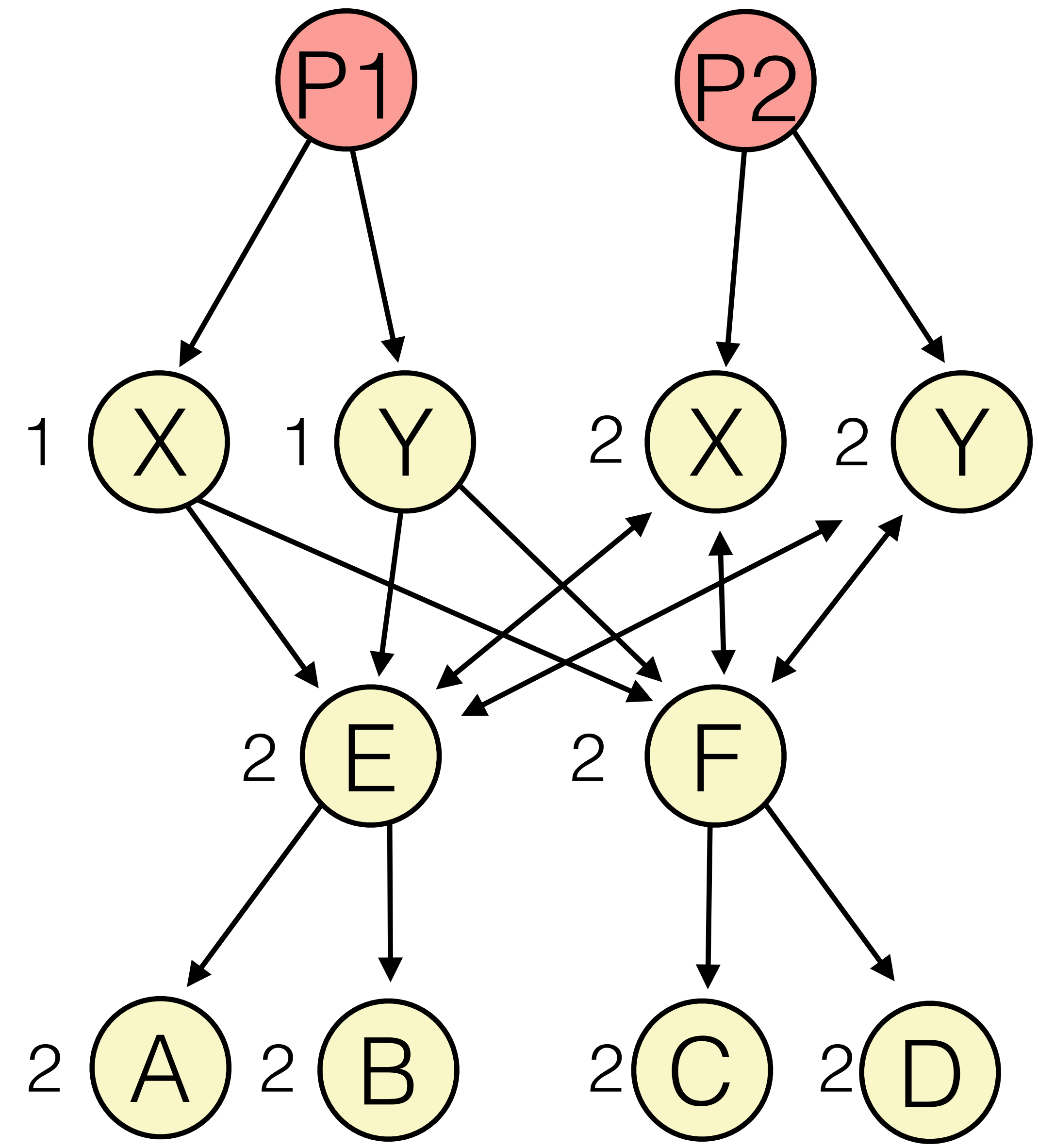
lp(X, P1 >> P2)
lp(Y, P1 >> P2)

(.*; P1; X) >> (.*; P2; X)
U (.*; P1; Y) >> (.*; P2; Y)



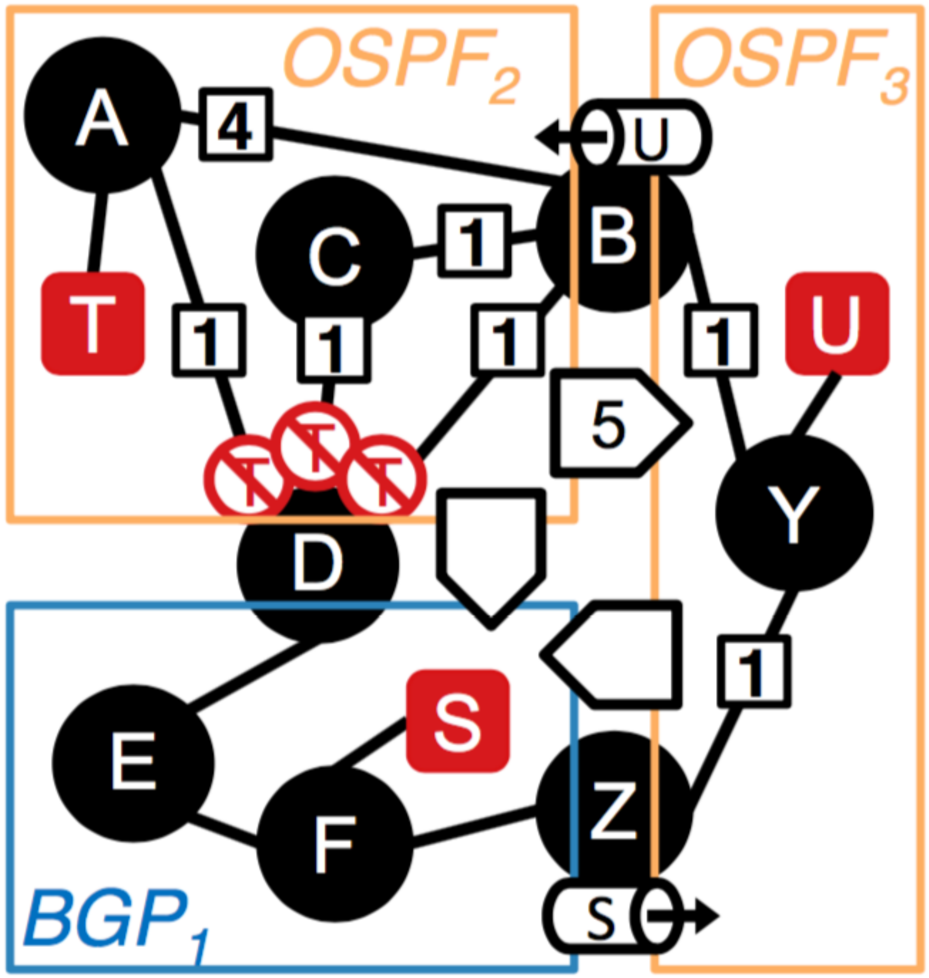
Propane - Local Preferences

Note:
Global safety analysis still passes, so the policy is still safe under all failures

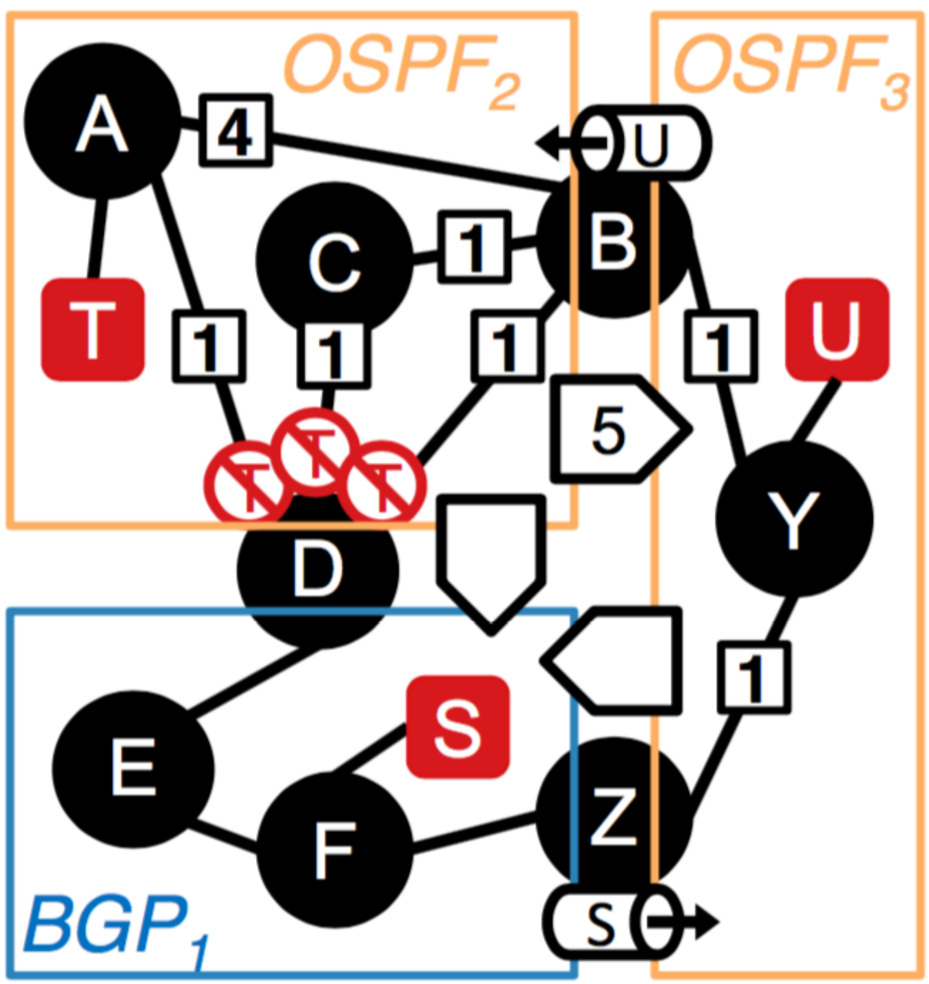


Synthesis of Other Protocols

Verification Recap



Verification Recap



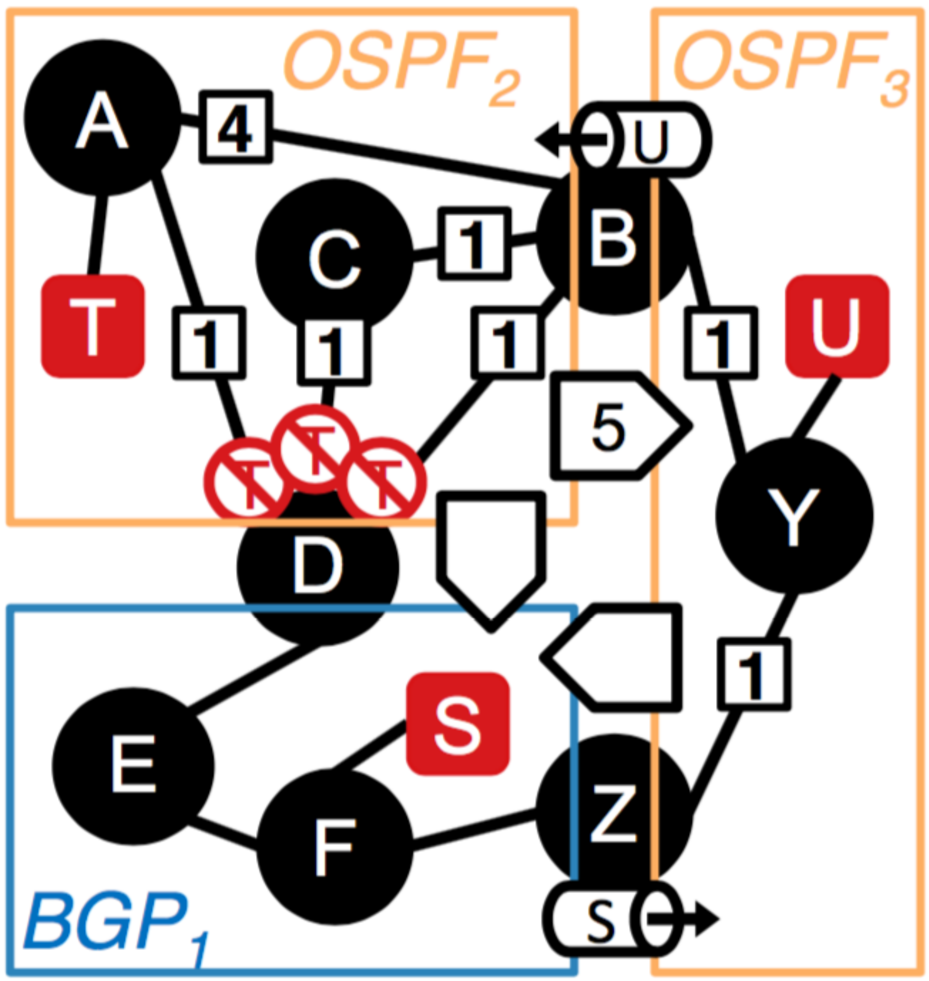
EC: {T}, {S}, {U}

Equivalence Classes:
Traffic classes that will experience the exact same forwarding behavior after the control plane stabilizes

High-level Idea
PG lets us represent a local preference among neighbors. We wish to prefer based on:

(1) Protocol (AD)
(2) Protocol-specific preference

Verification Recap



EC: {T}, {S}, {U}

AD: {1↦1, 5↦2, 20↦3, 110↦4}

Static **User** **eBGP** **OSPF**

BGP (lp): {100 ↦ 1}

OSPF: {_ ↦ 1}

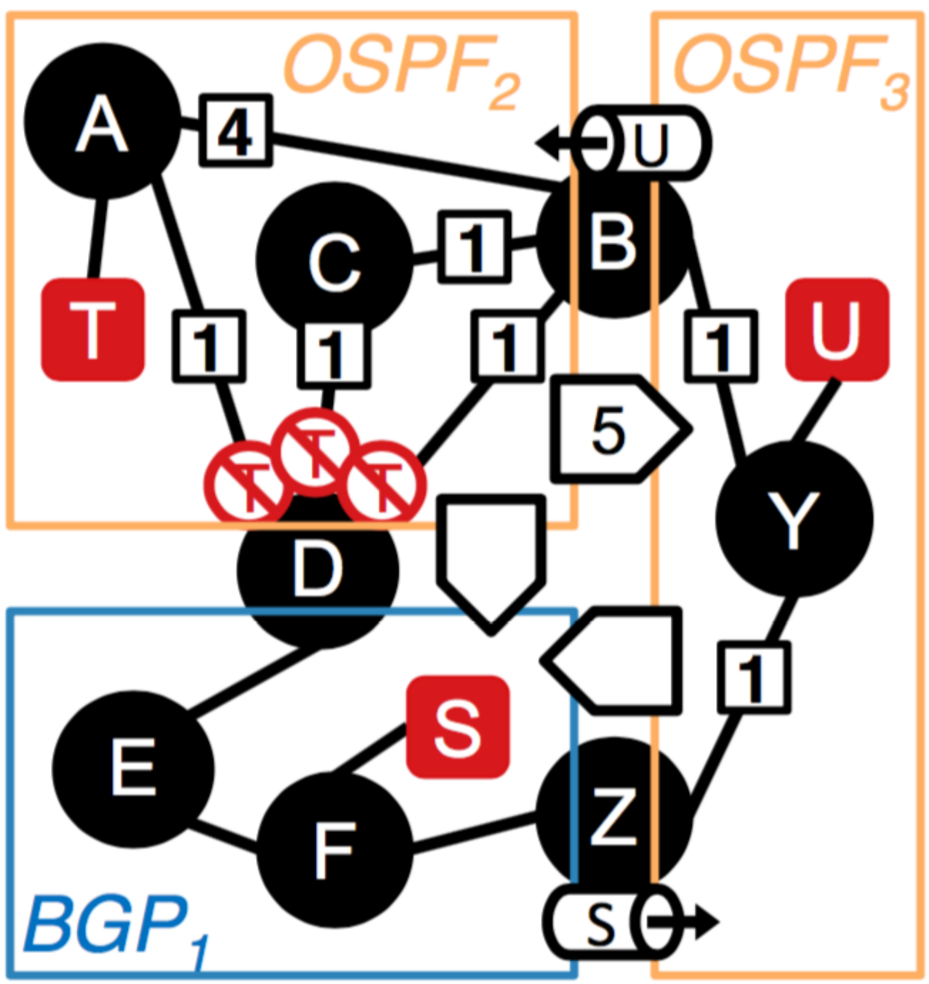
Preference: (AD x _)

Protocol specific

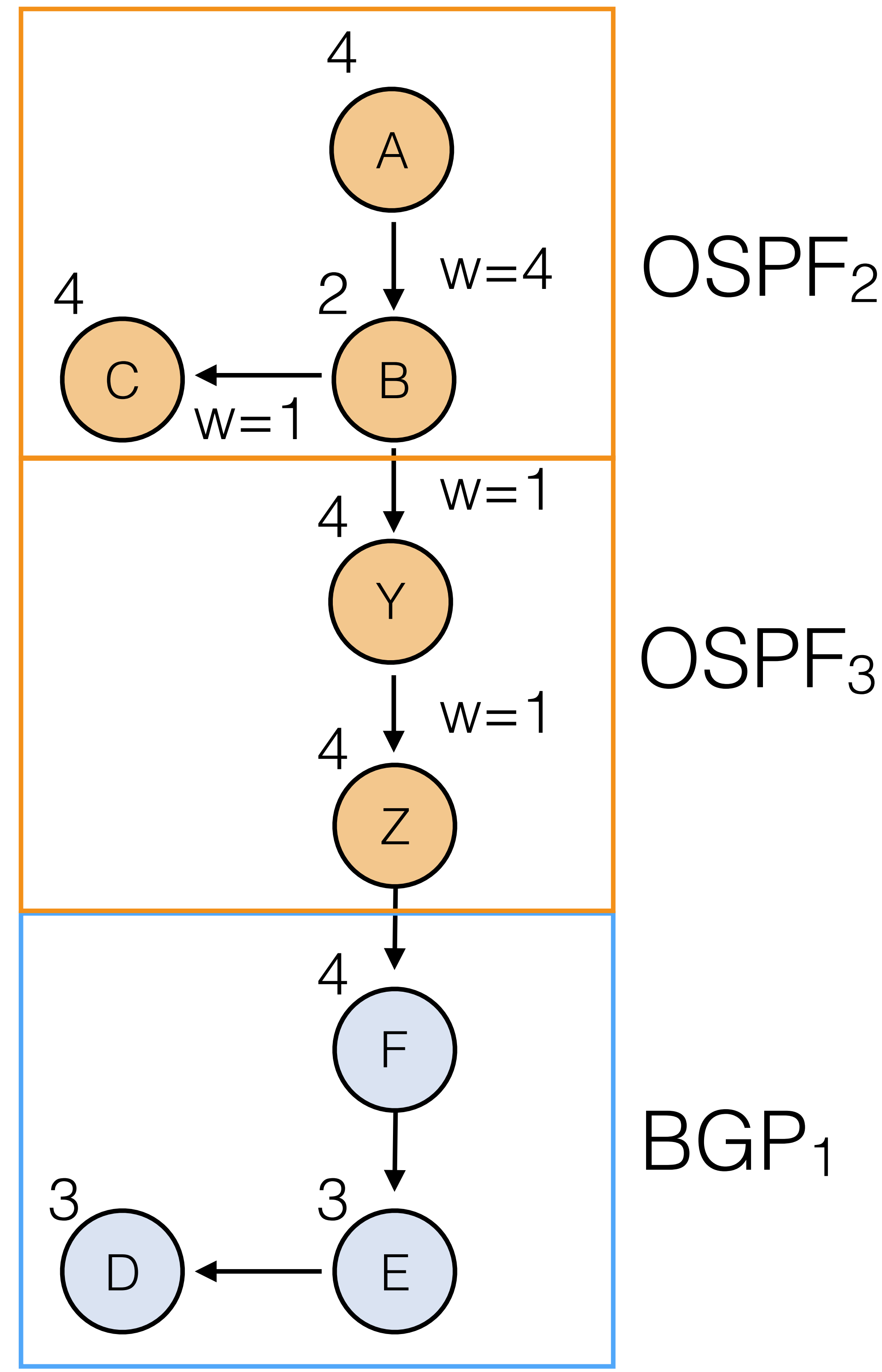
BGP: {1,2,3,4}

OSPF: {1,2,3,4}

Verification Recap



EC: {T}, {S}, {U}

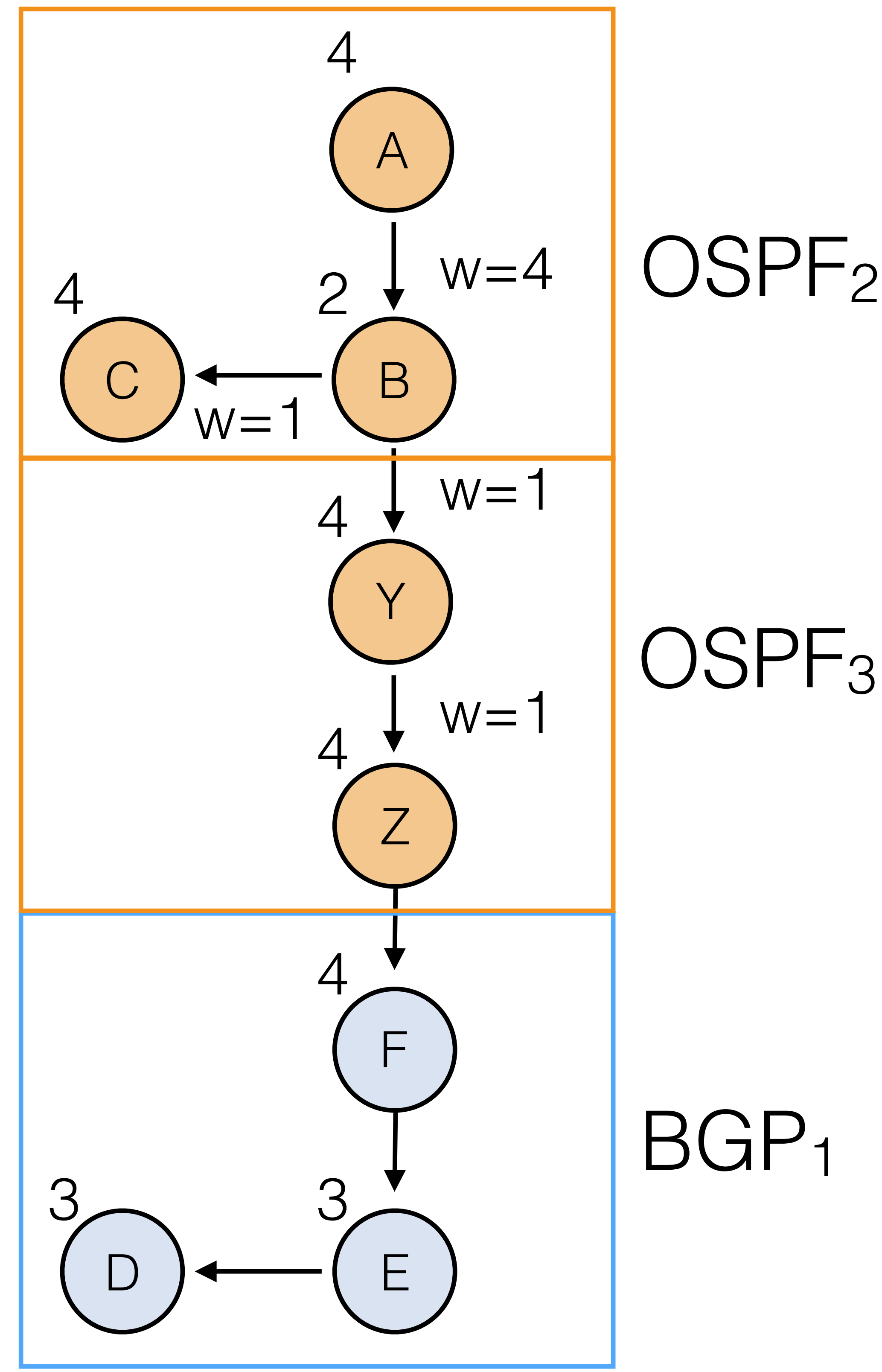


Synthesis of Other Protocols:

Idea From Last Time:

PG can encode preferences:

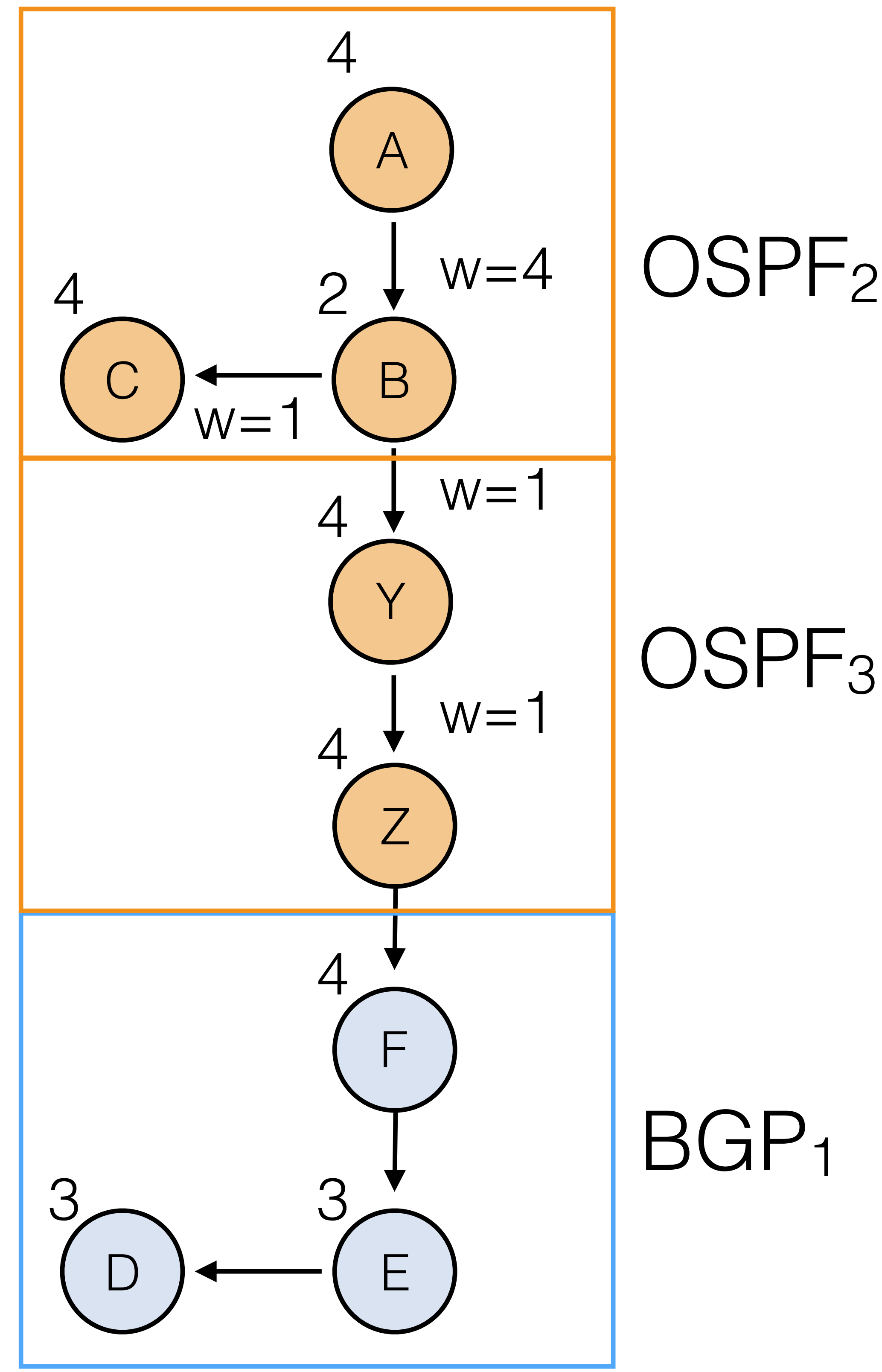
1. BGP local-pref
2. Route Redistribution AD
3. Edge weight etc.



Synthesis of Other Protocols:

Idea From Last Time:
PG can encode preferences:
1. BGP local-pref
2. Route Redistribution AD
3. Edge weight etc.

Key Insight:
In the PG (Propane), how we interpret the preferences determines the implementation.



Synthesis of Other Protocols:

Idea From Last Time:

PG can encode preferences:

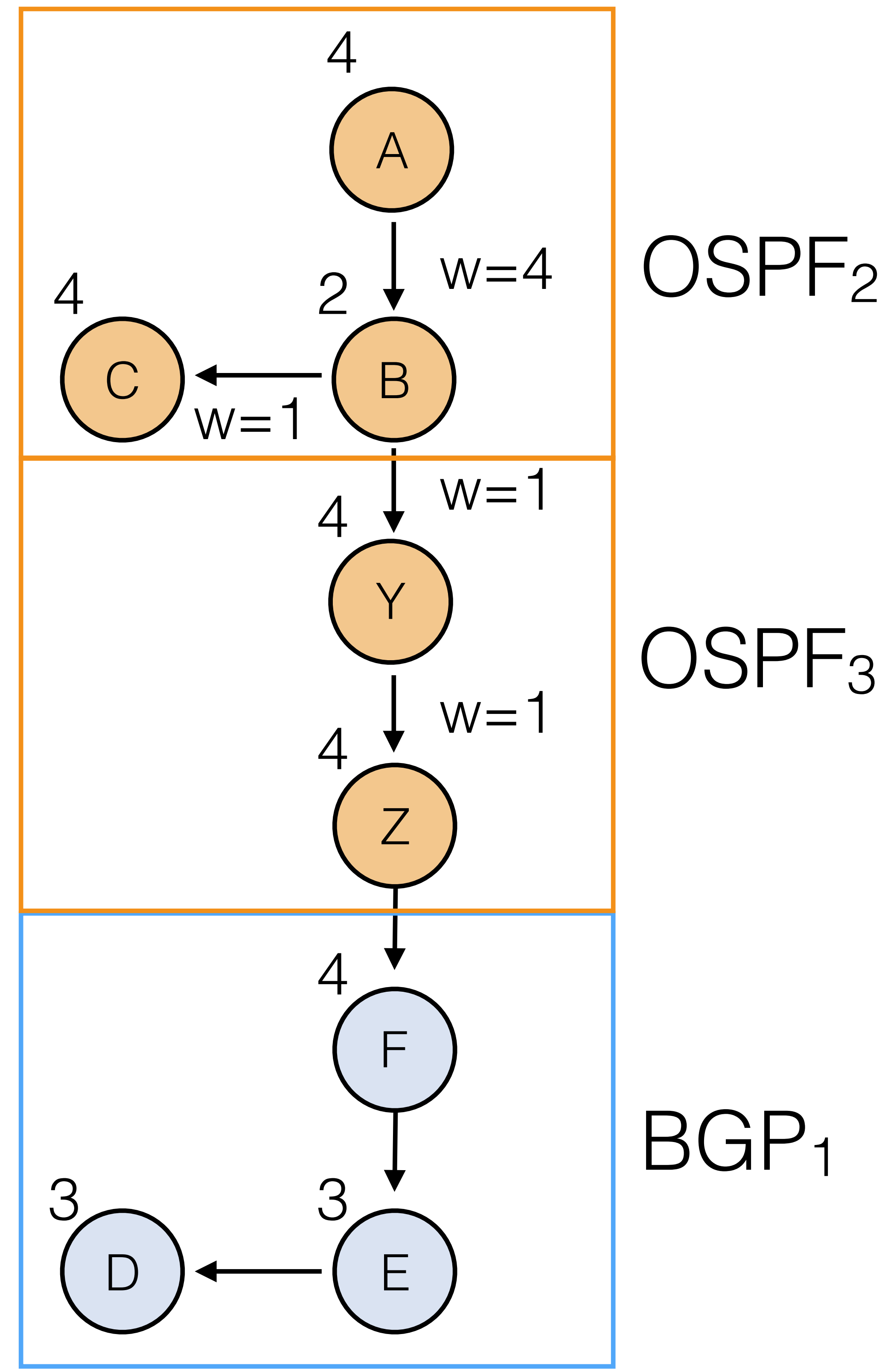
1. BGP local-pref
2. Route Redistribution AD
3. Edge weight etc.

Key Insight:

In the PG (Propane), how we interpret the preferences determines the implementation.

LP ⟼ BGP

AD ⟼ Route Redist.



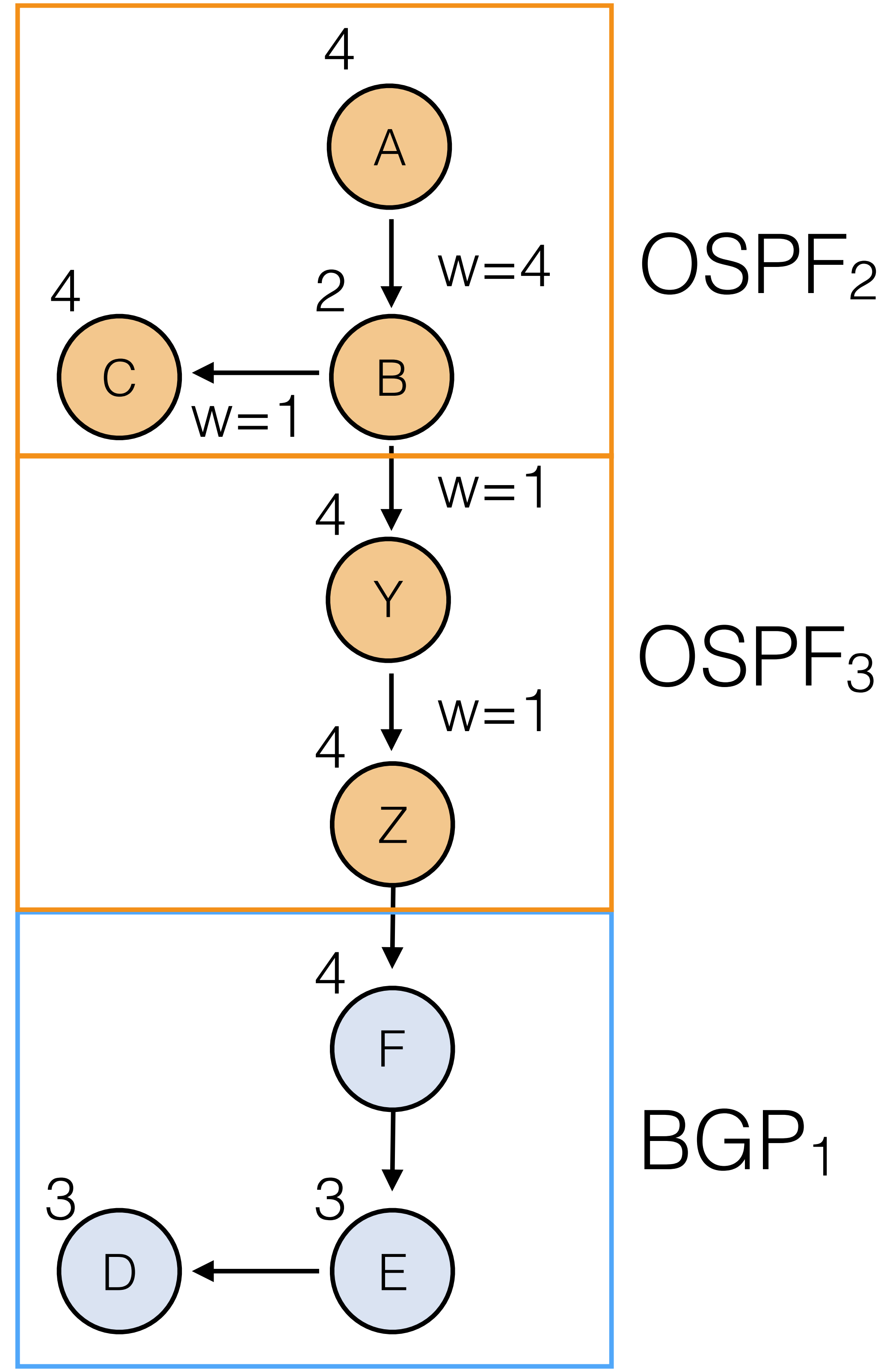
Synthesis of Other Protocols:

Key Insight:
In the PG (Propane), how we interpret the preferences determines the implementation.

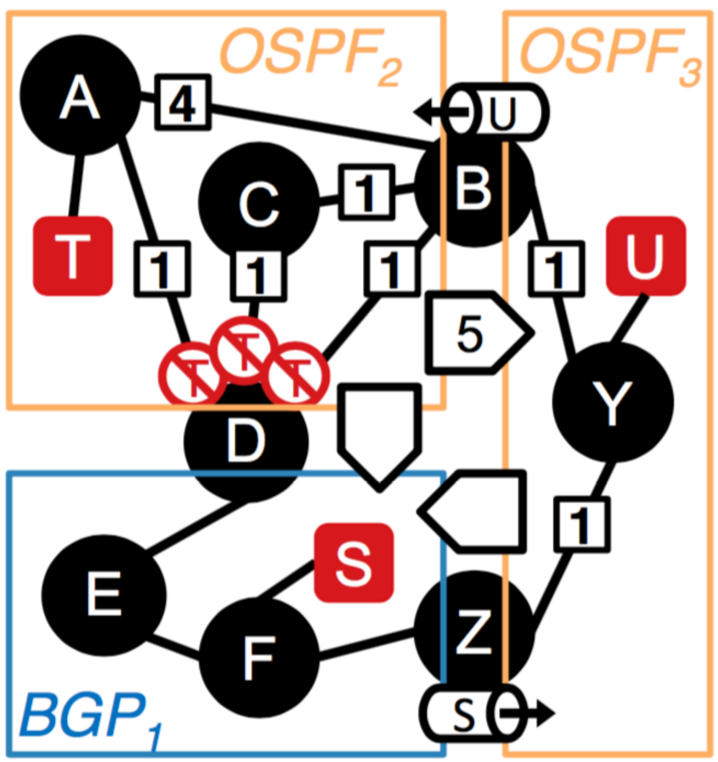
LP $\longmapsto$ BGP

AD $\longmapsto$ Route Redist.

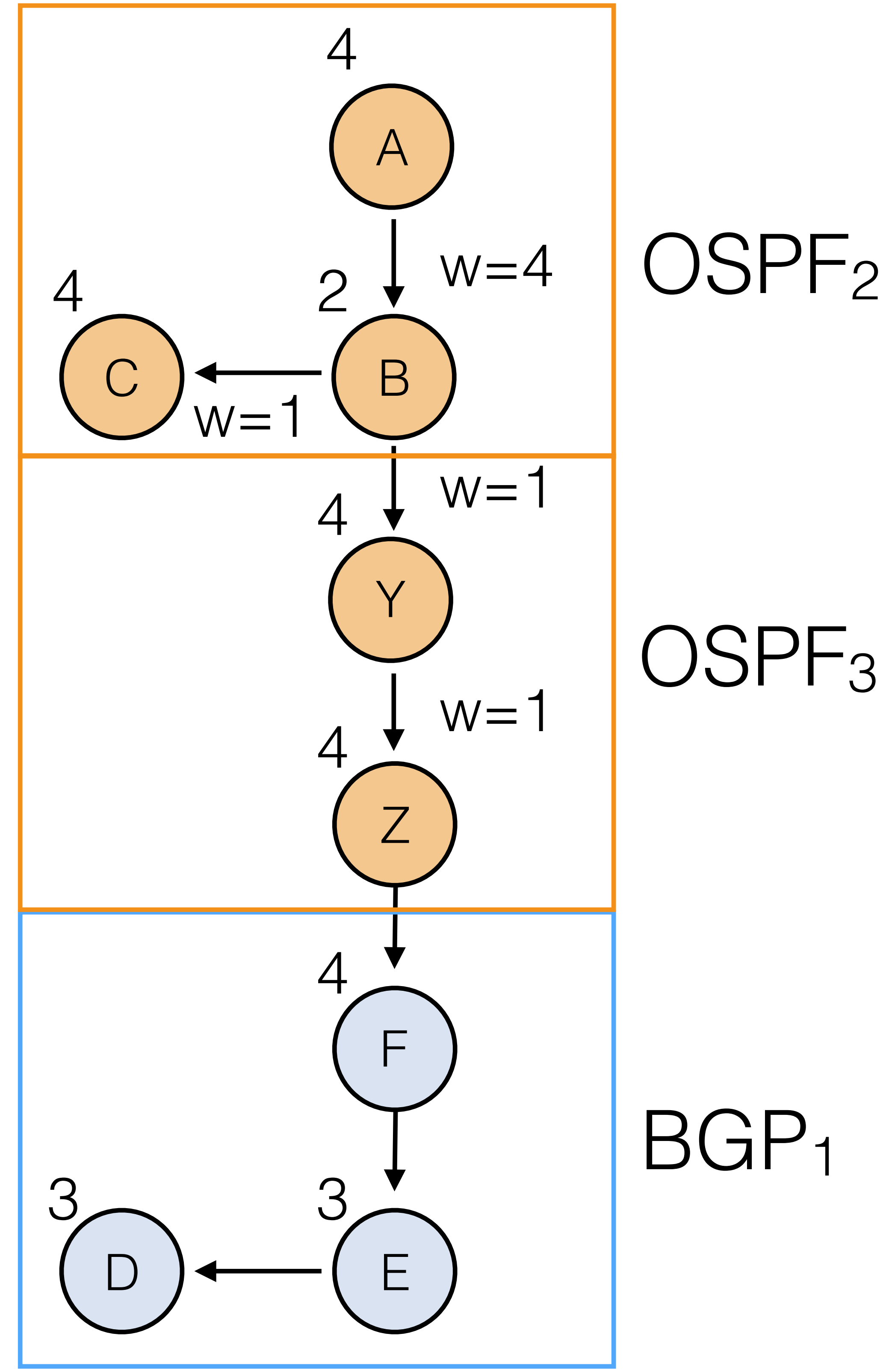
Propane:
Compiler infers all node preferences and can choose how to partition the topology according to RR preferences.



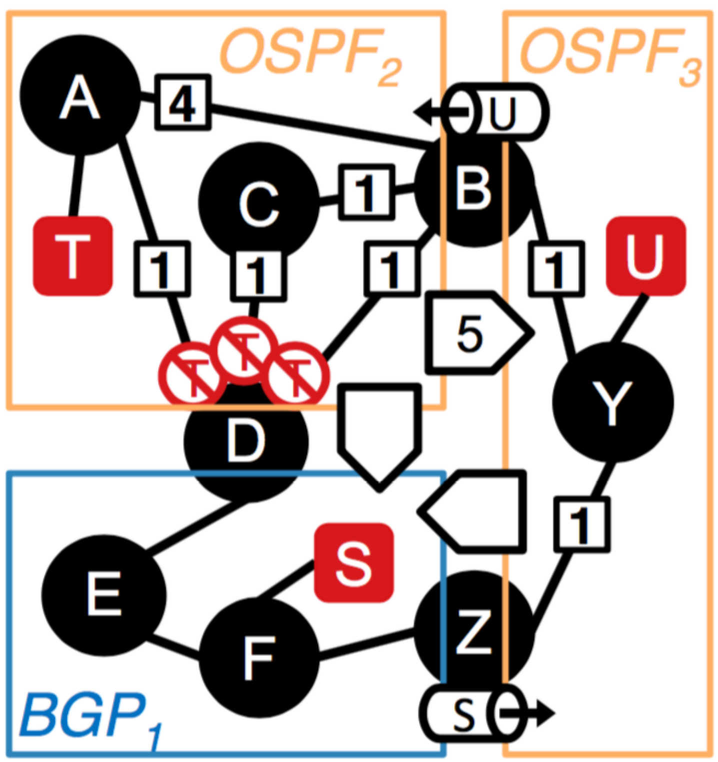
Synthesis of Other Protocols:



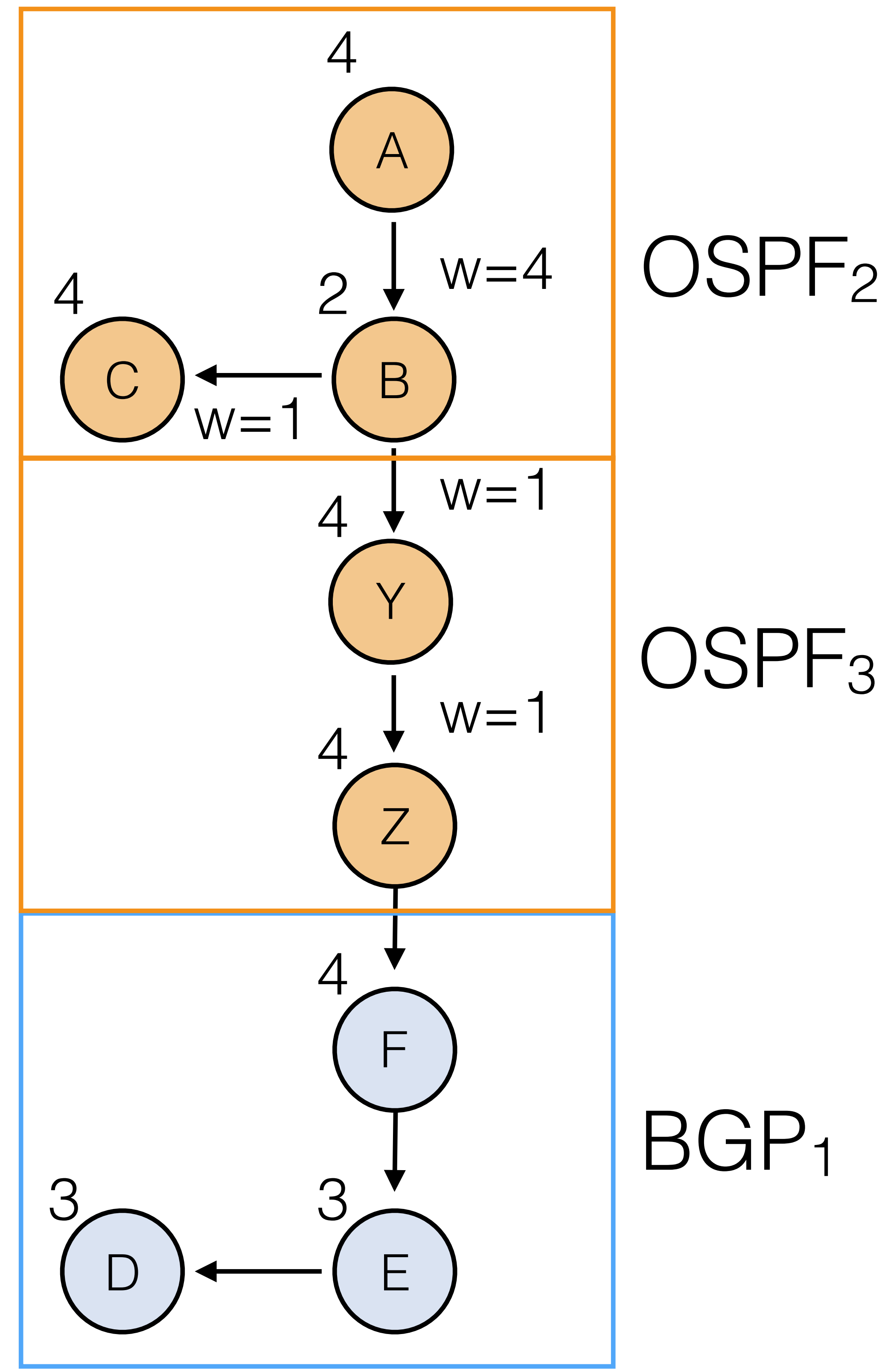
Constraints: *RR*:
Can't rely on context (e.g, community tag), preference must be unique from neighbors.



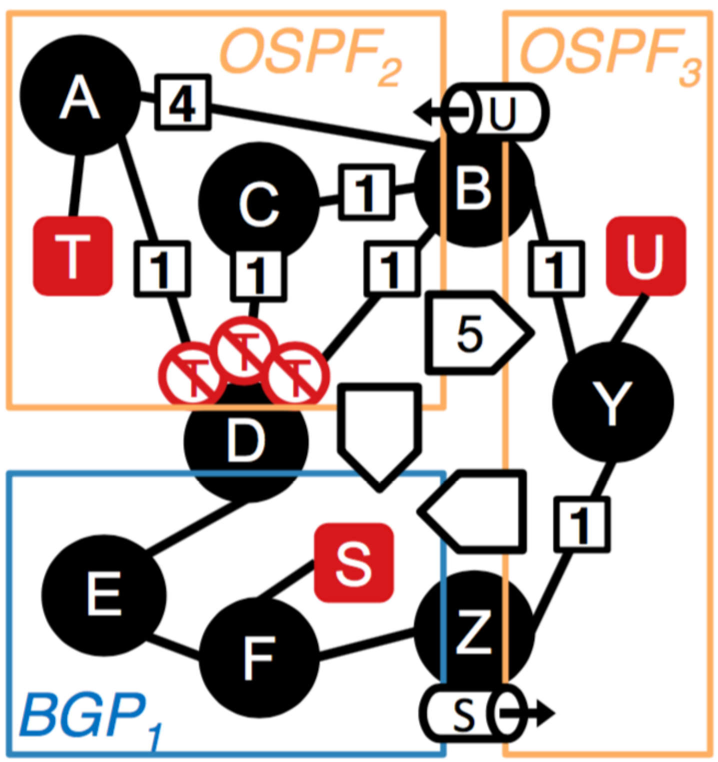
Synthesis of Other Protocols:



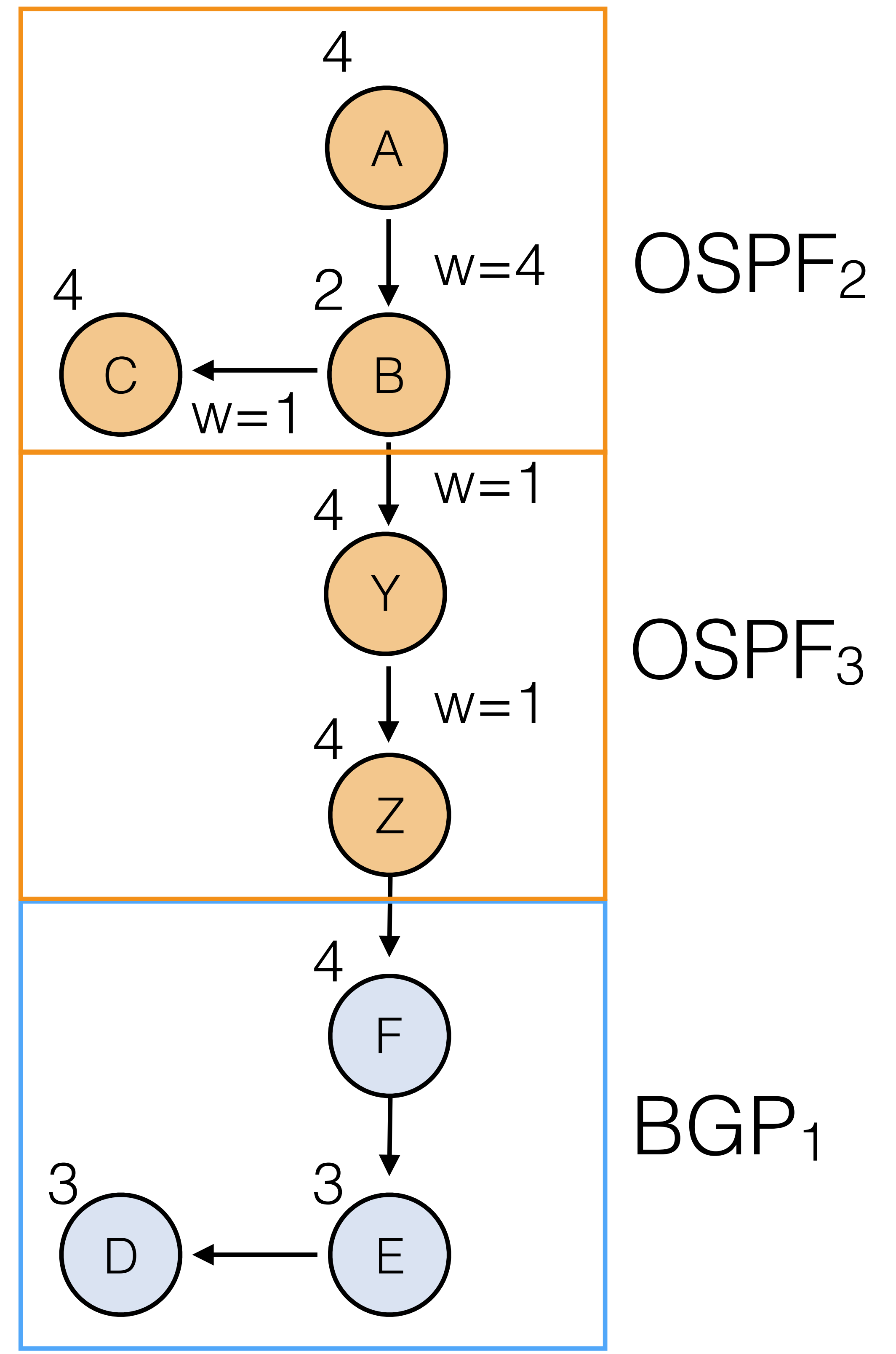
Constraints: *OSPF*:
No preferences allowed within an OSPF region.



Synthesis of Other Protocols:



Constraints: *RIP*:
No preferences allowed within a RIP region.



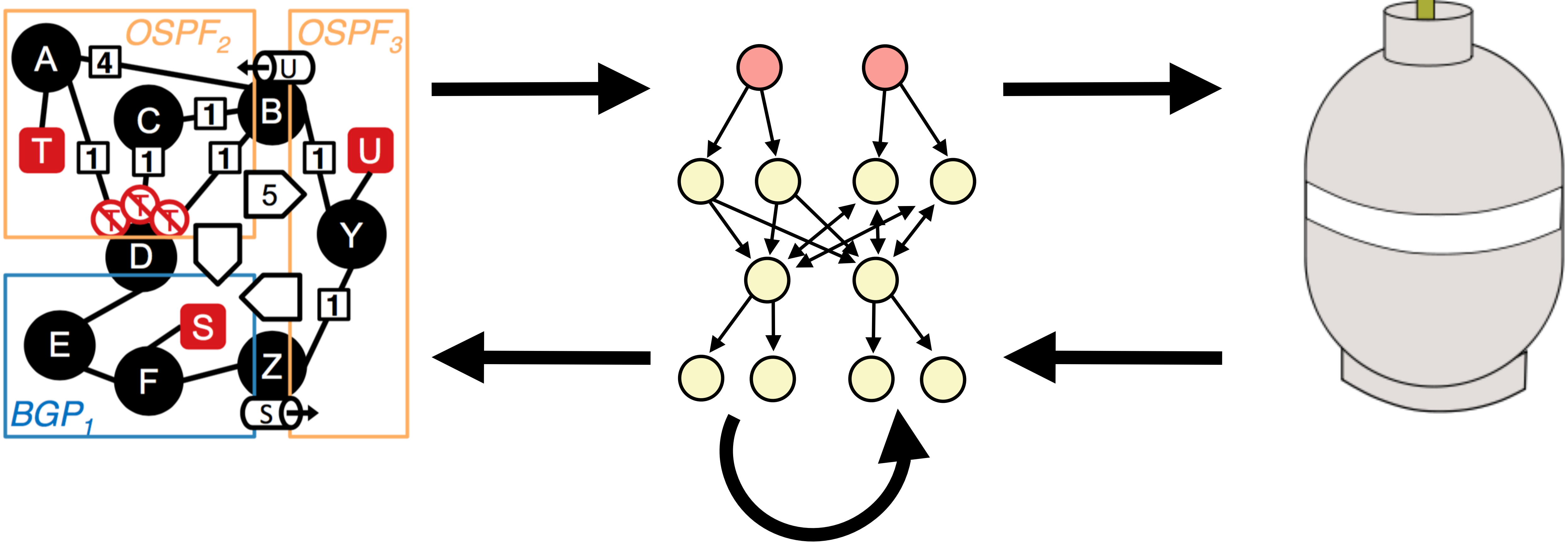
The Right Abstraction?

Configs

BGP, OPSF, RIP

PG

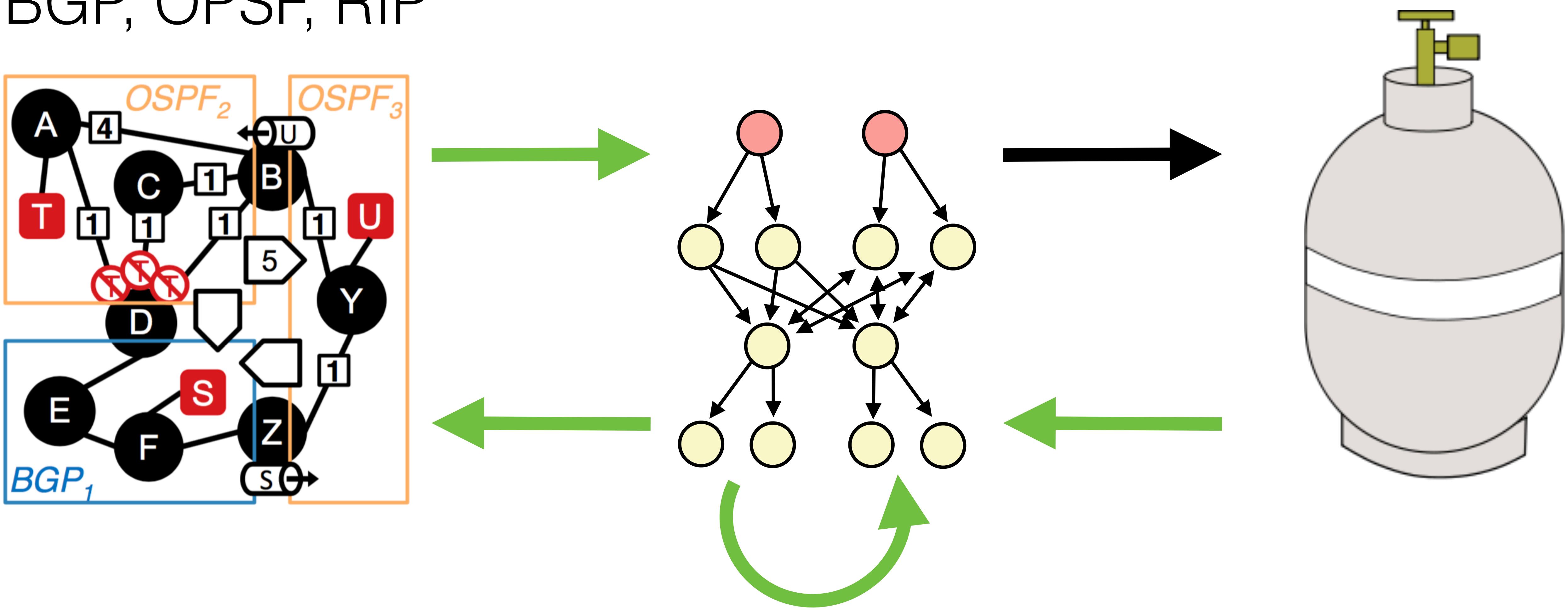
Propane



The Right Abstraction?

Configs **PG** **Propane**

BGP, OPSF, RIP

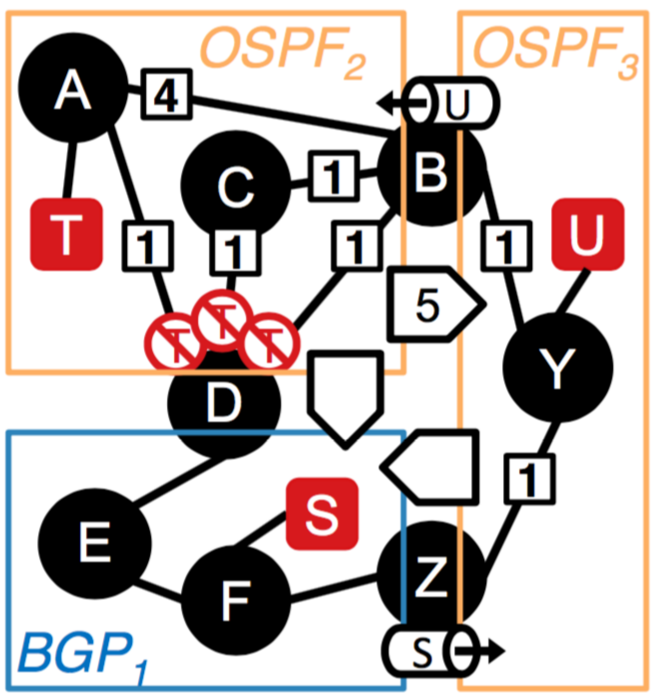


We could possible support the following arrows so far

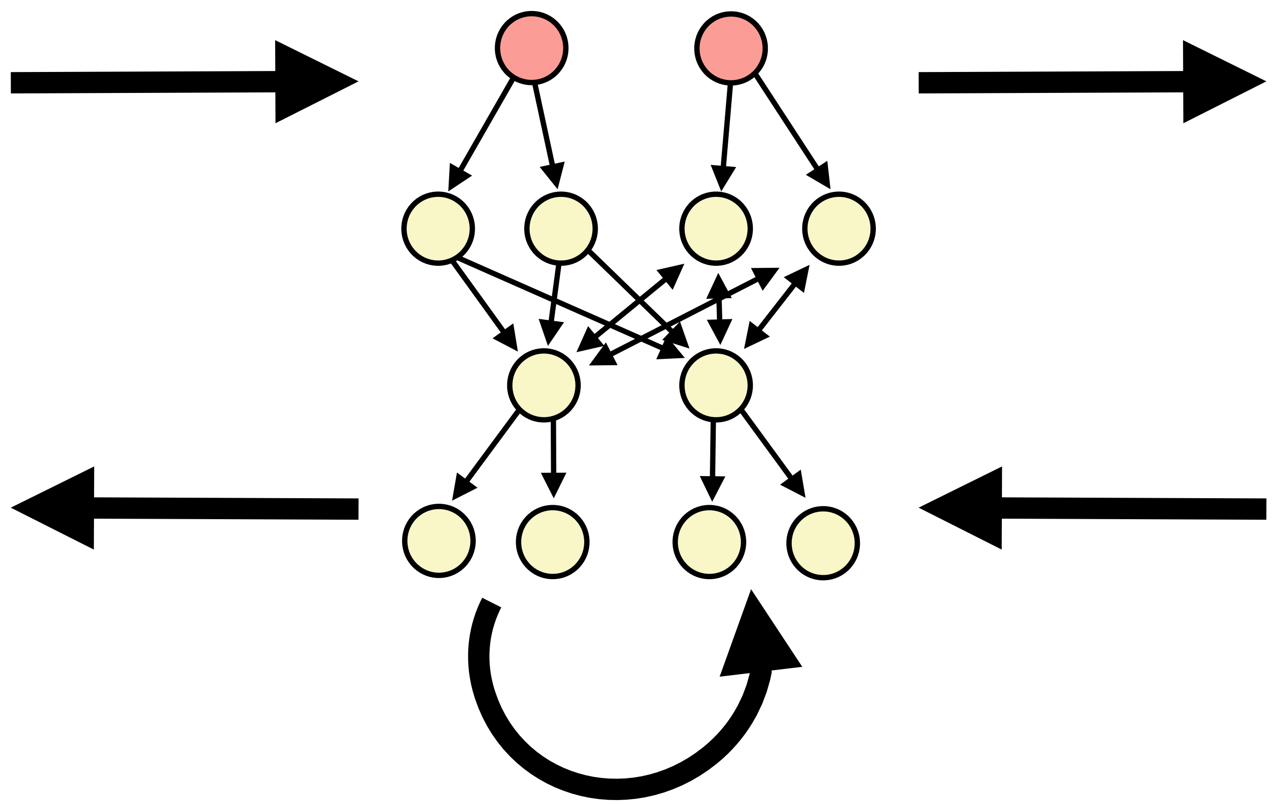
The Right Abstraction?

Configs

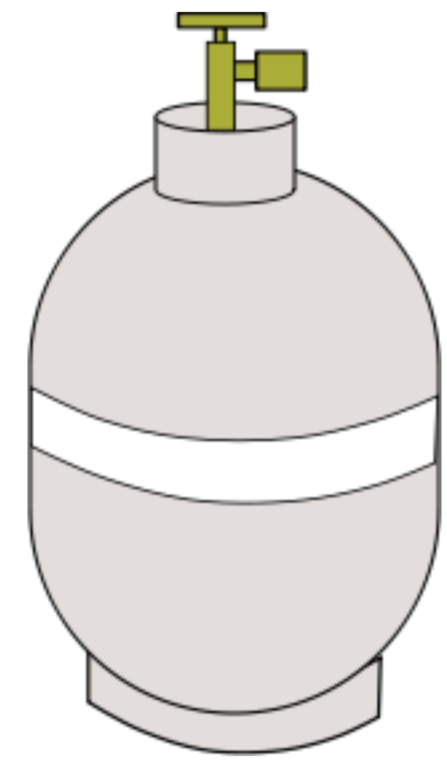
BGP, OPSF, RIP



PG



Propane

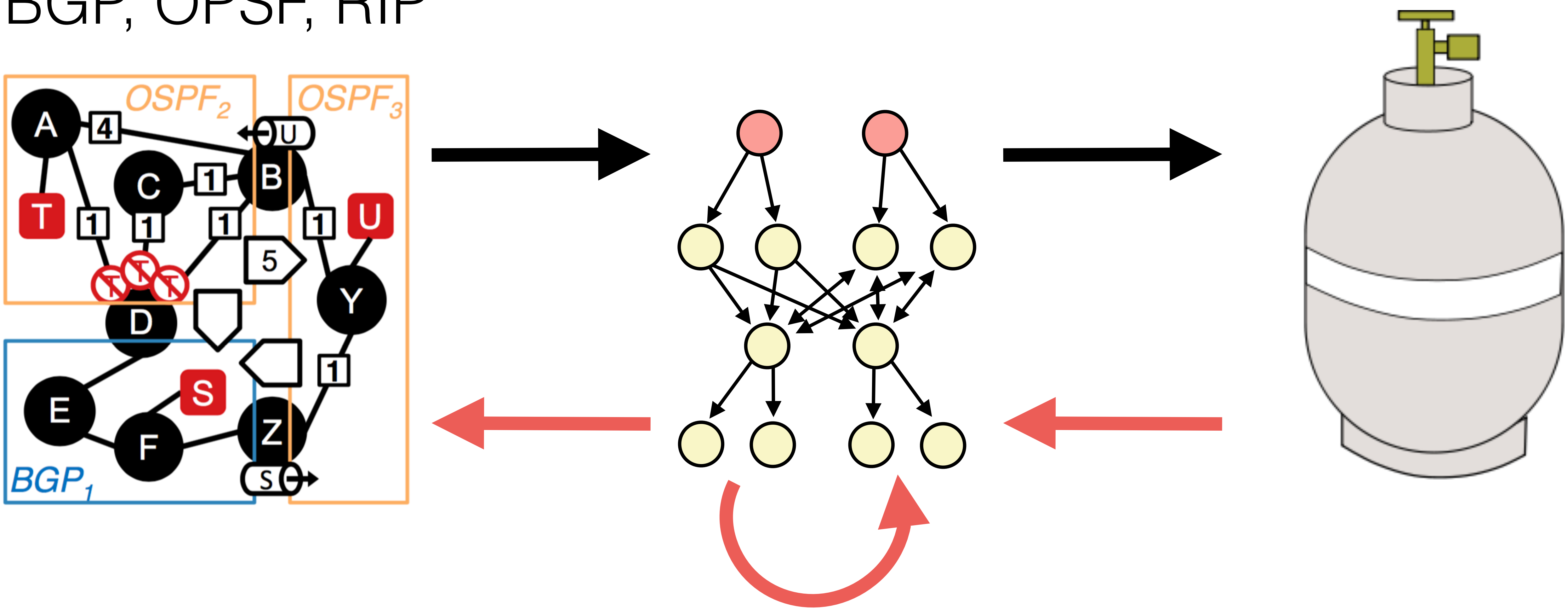


Pick any combination of arrows to generate a tool & paper

The Right Abstraction?

Configs **PG** **Propane**

BGP, OPSF, RIP

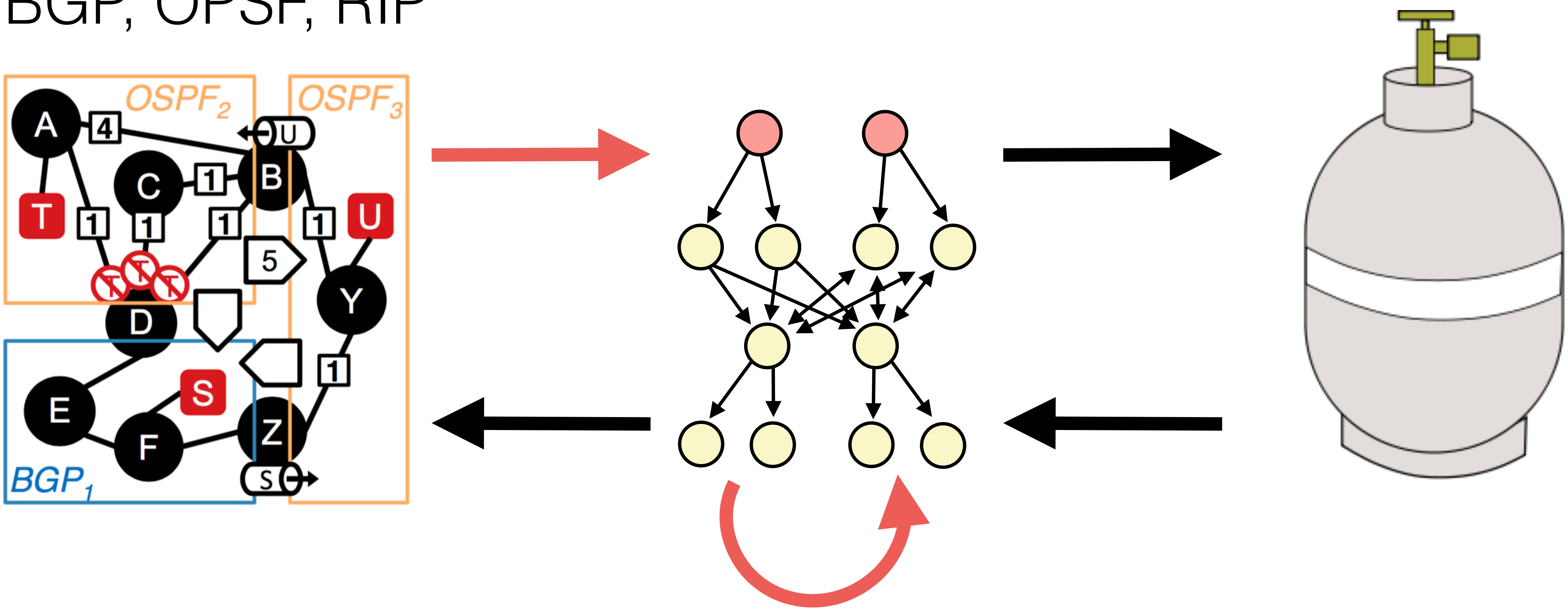


Compilation / Safety

The Right Abstraction?

Configs **PG** **Propane**

BGP, OPSF, RIP



Verification

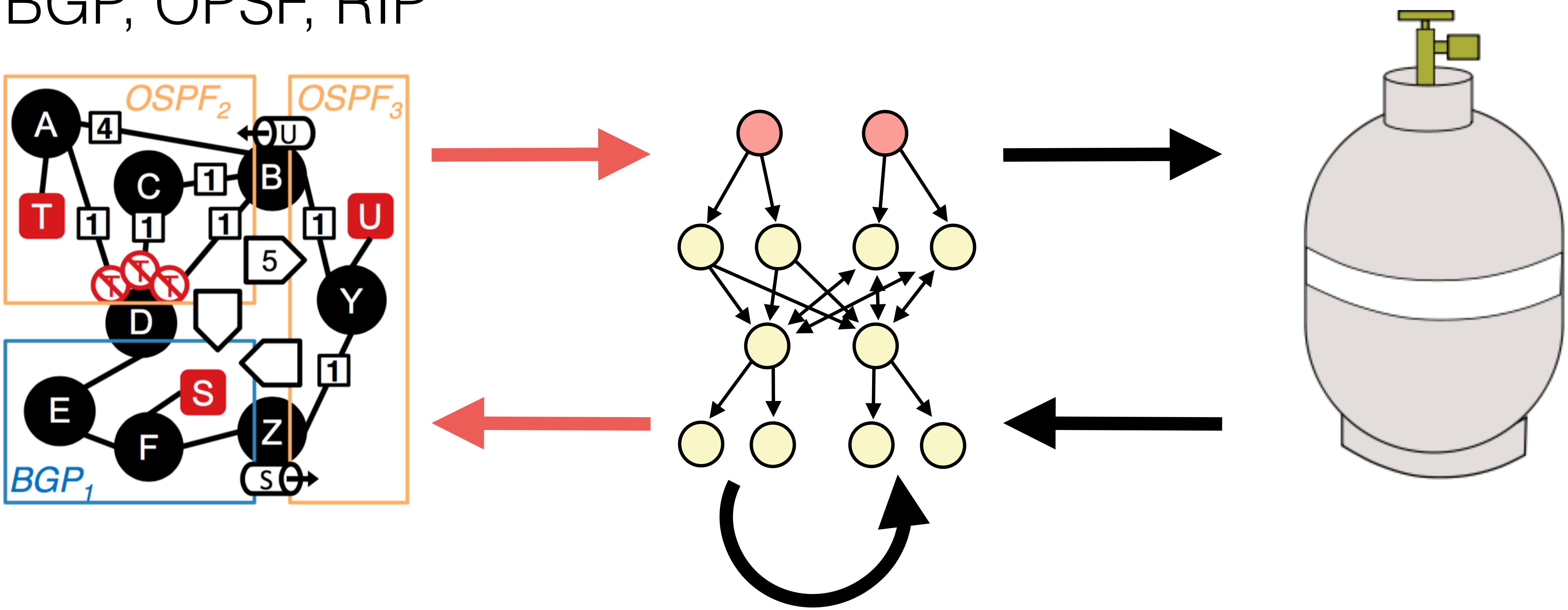
The Right Abstraction?

Configs

BGP, OPSF, RIP

PG

Propane

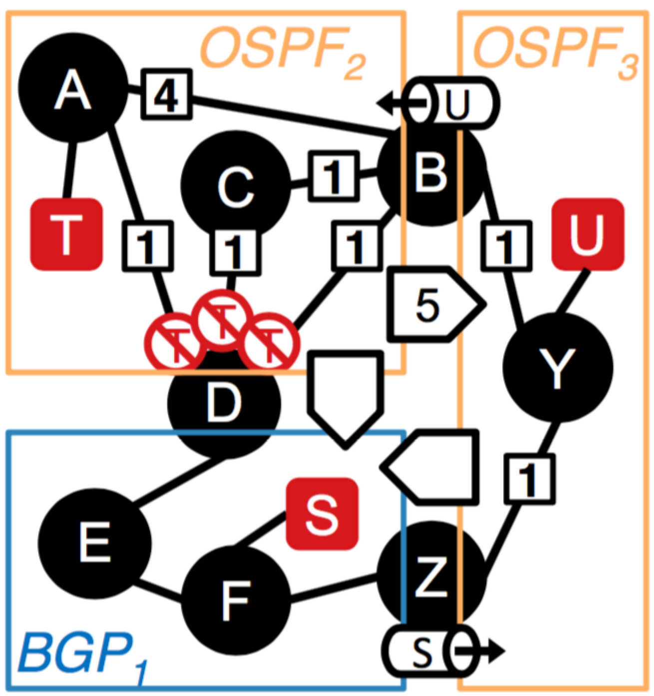


Config Minimization / Migration

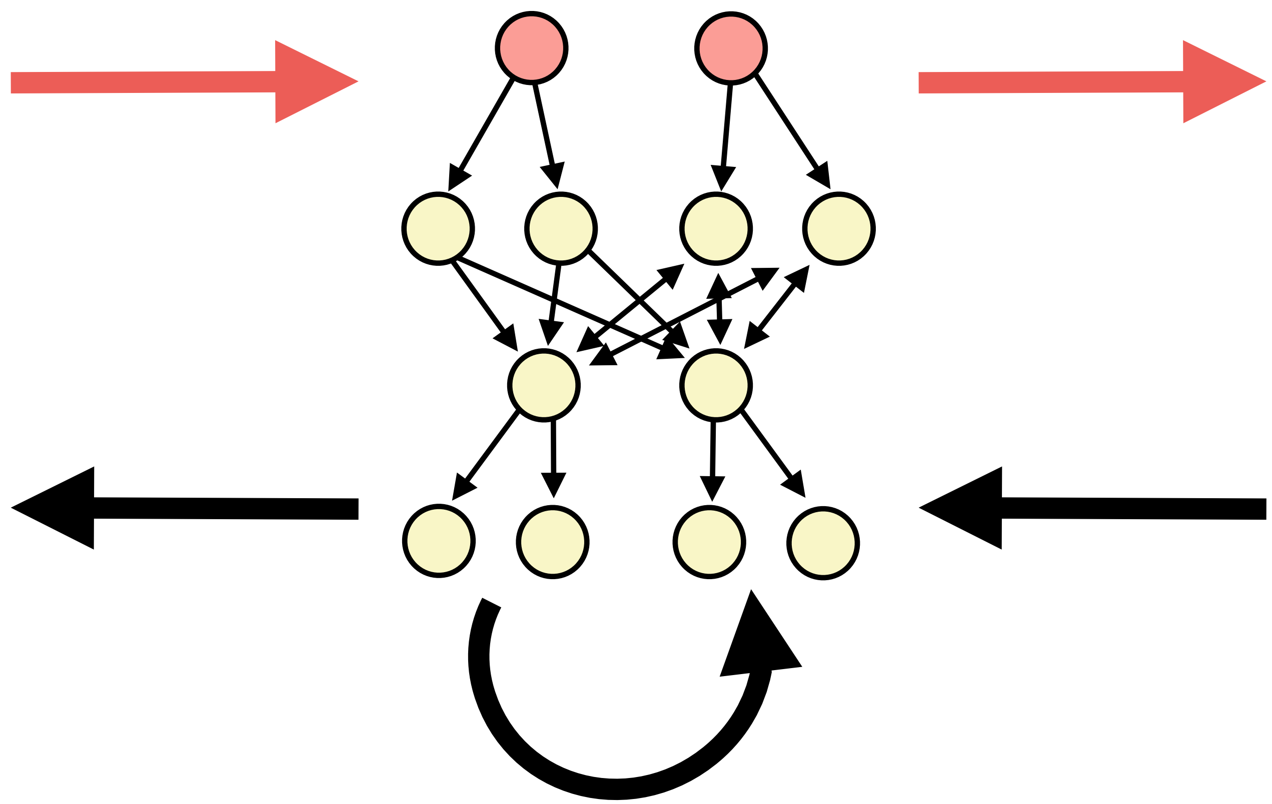
The Right Abstraction?

Configs

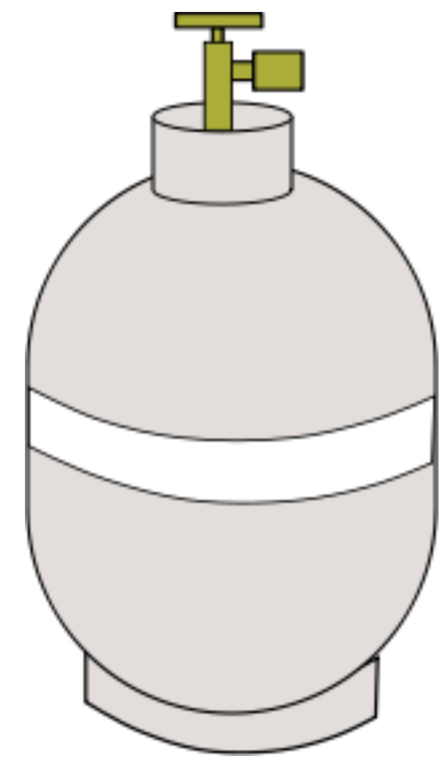
BGP, OPSF, RIP



PG



Propane

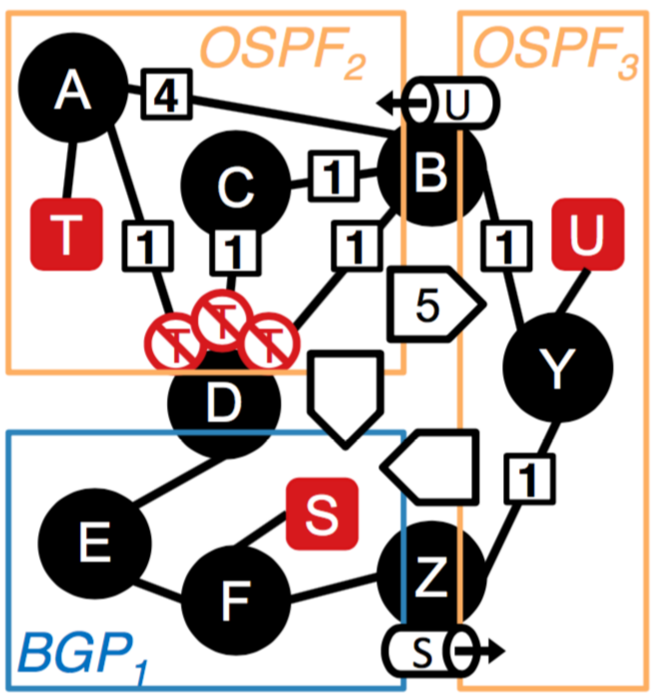


Policy Synthesis

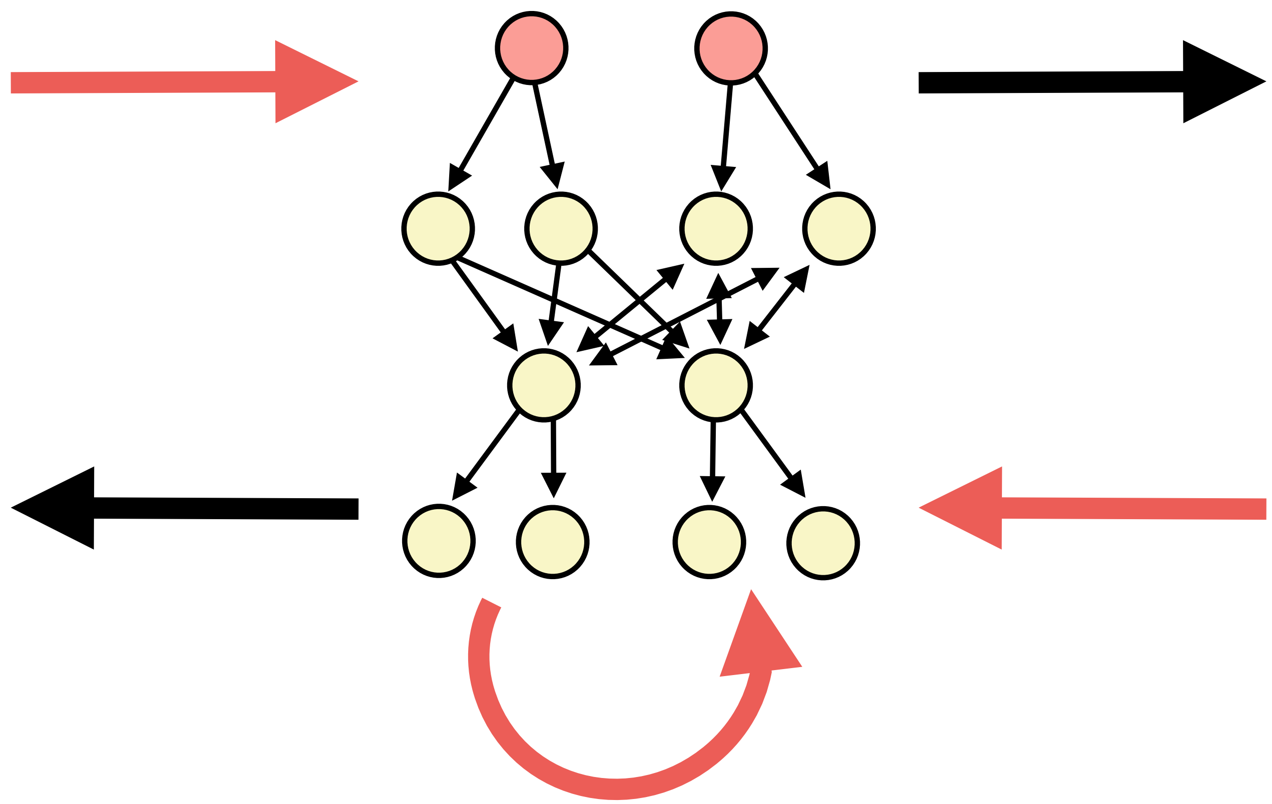
The Right Abstraction?

Configs

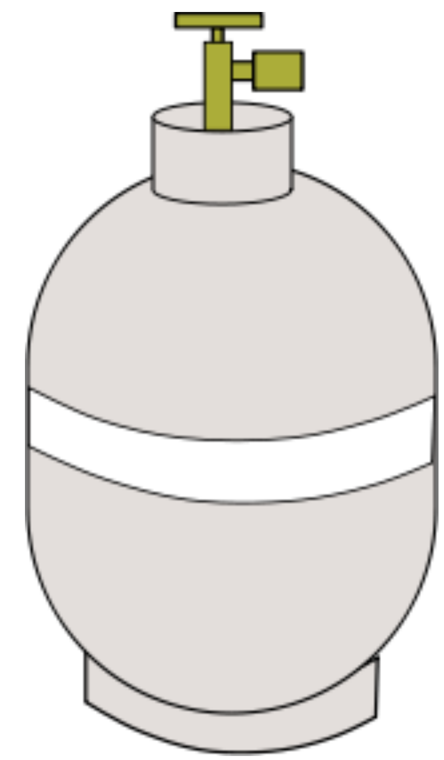
BGP, OPSF, RIP



PG



Propane



Verification / Equivalence