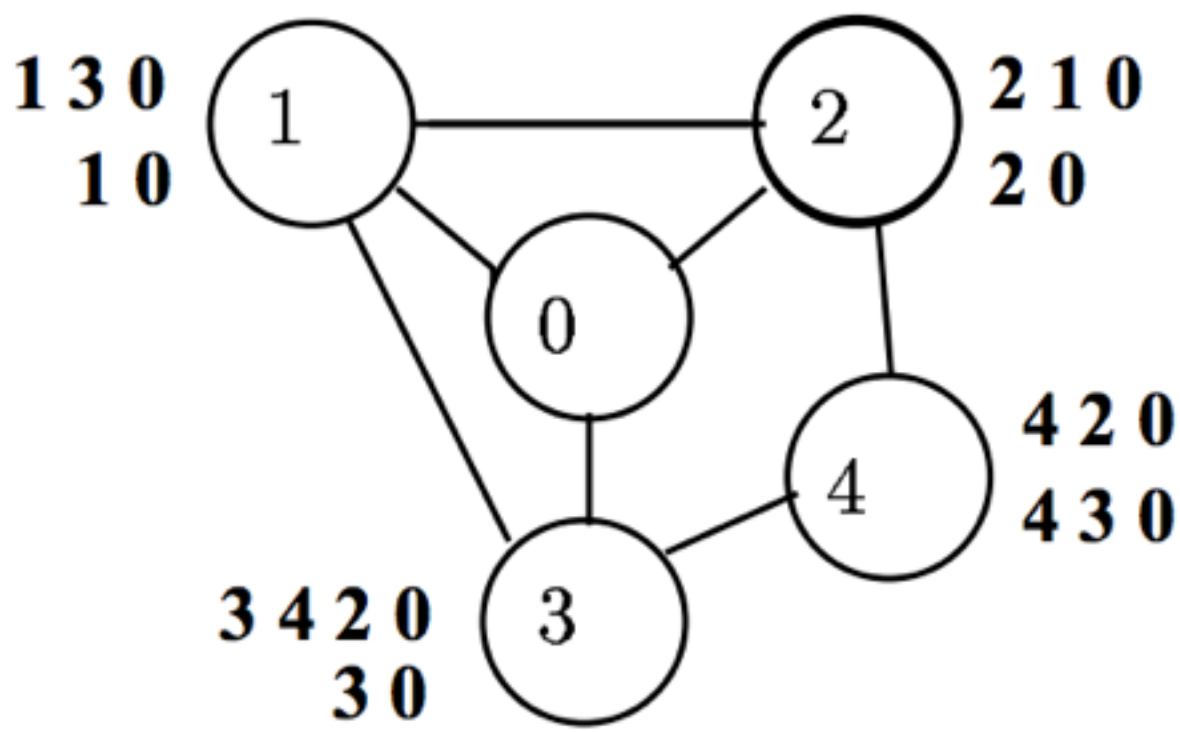
# Quick Summary

- Traditional BGP encoding
- Constructing PG from existing BGP configs (verification, synthesis?)
- Abstract topology safety analysis
  - Reachability under k-failures
  - Aggregation safety
- Proof of compilation correctness

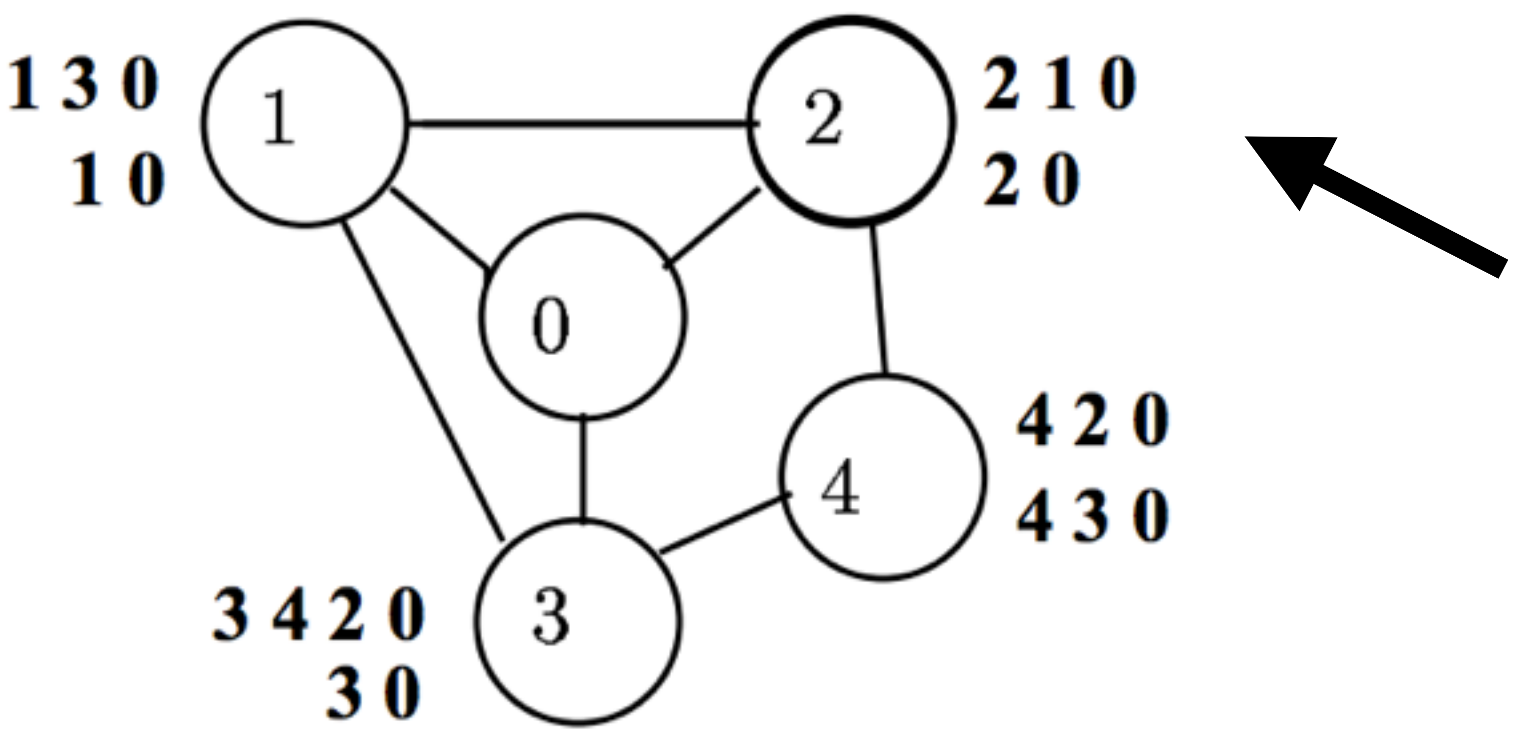
# Traditional BGP Encoding

# BGP ranked paths




BAD GADGET

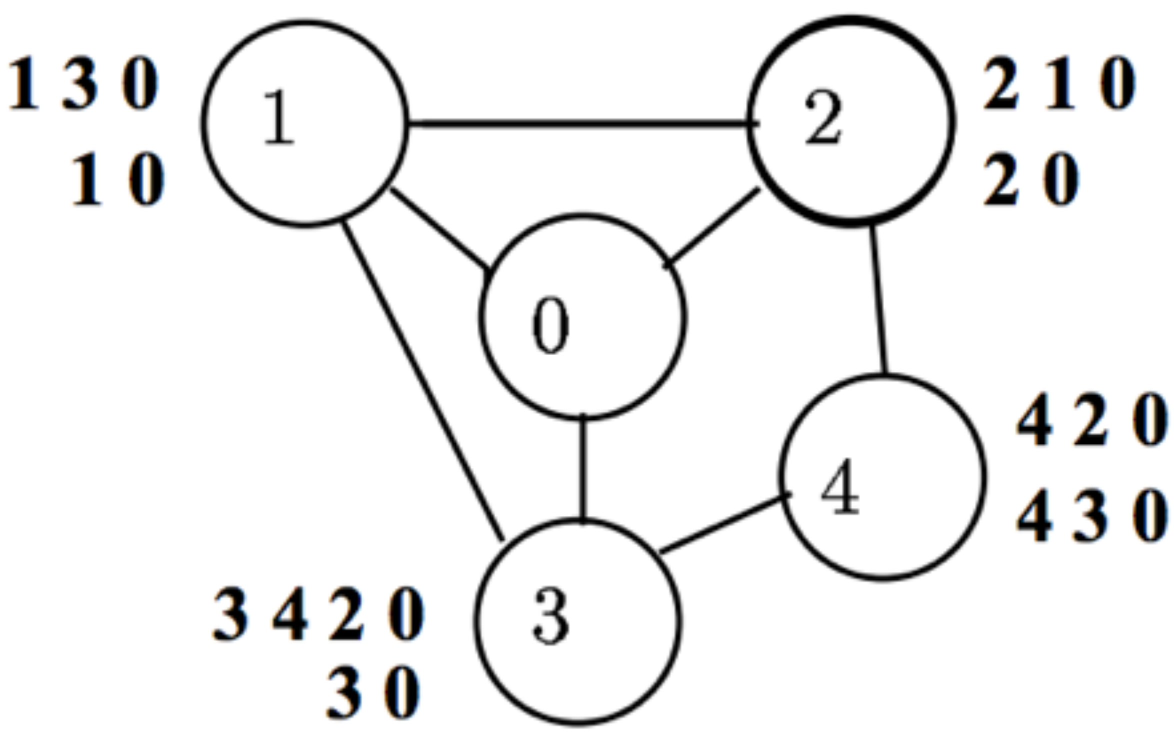
# BGP ranked paths




BAD GADGET

Given as ranked paths

# BGP ranked paths



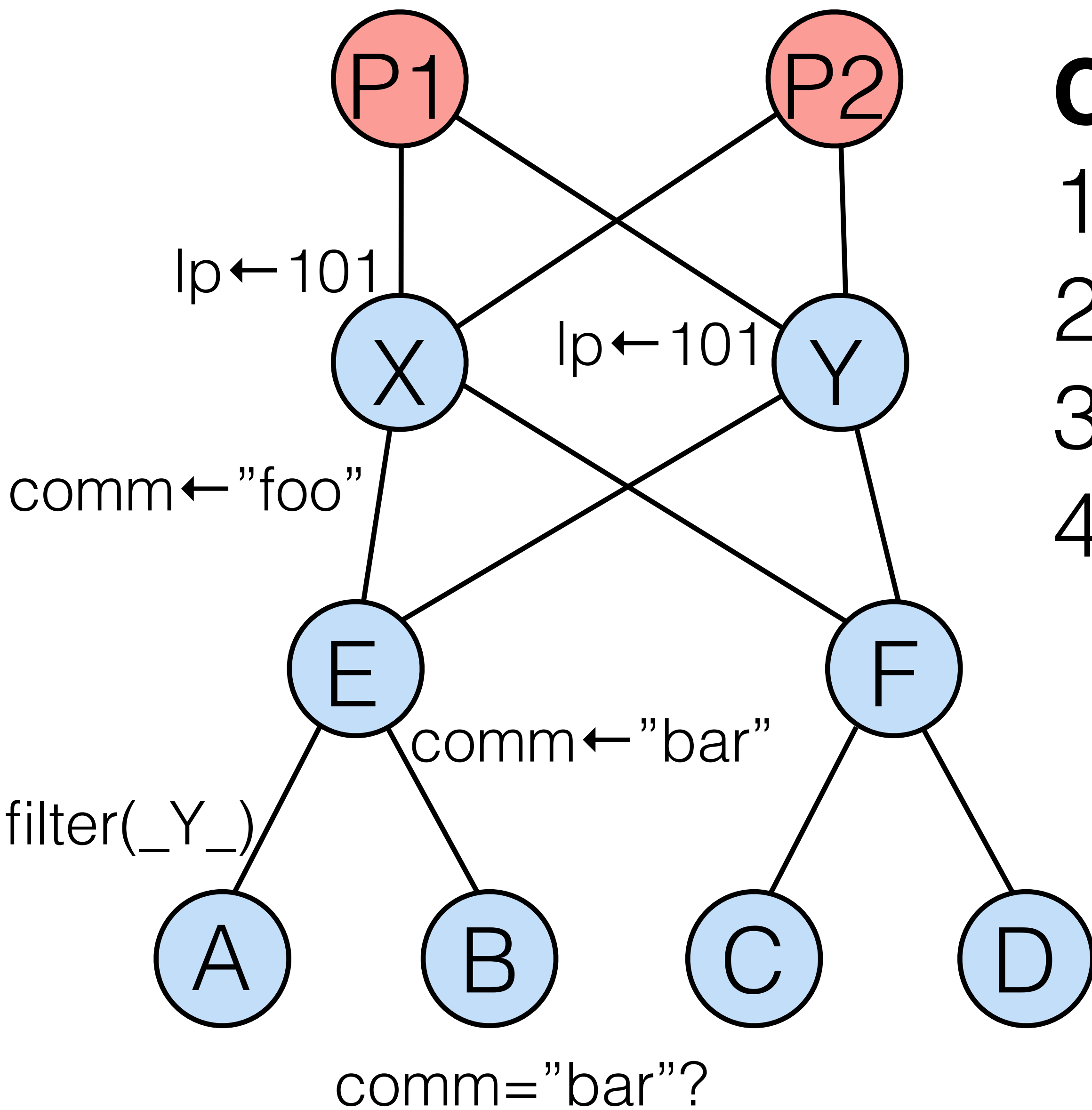

BAD GADGET

(130 ∪ 210 ∪ 420 ∪ 3420) >>

(10 ∪ 20 ∪ 30 ∪ 430) >>

# From BGP to PG

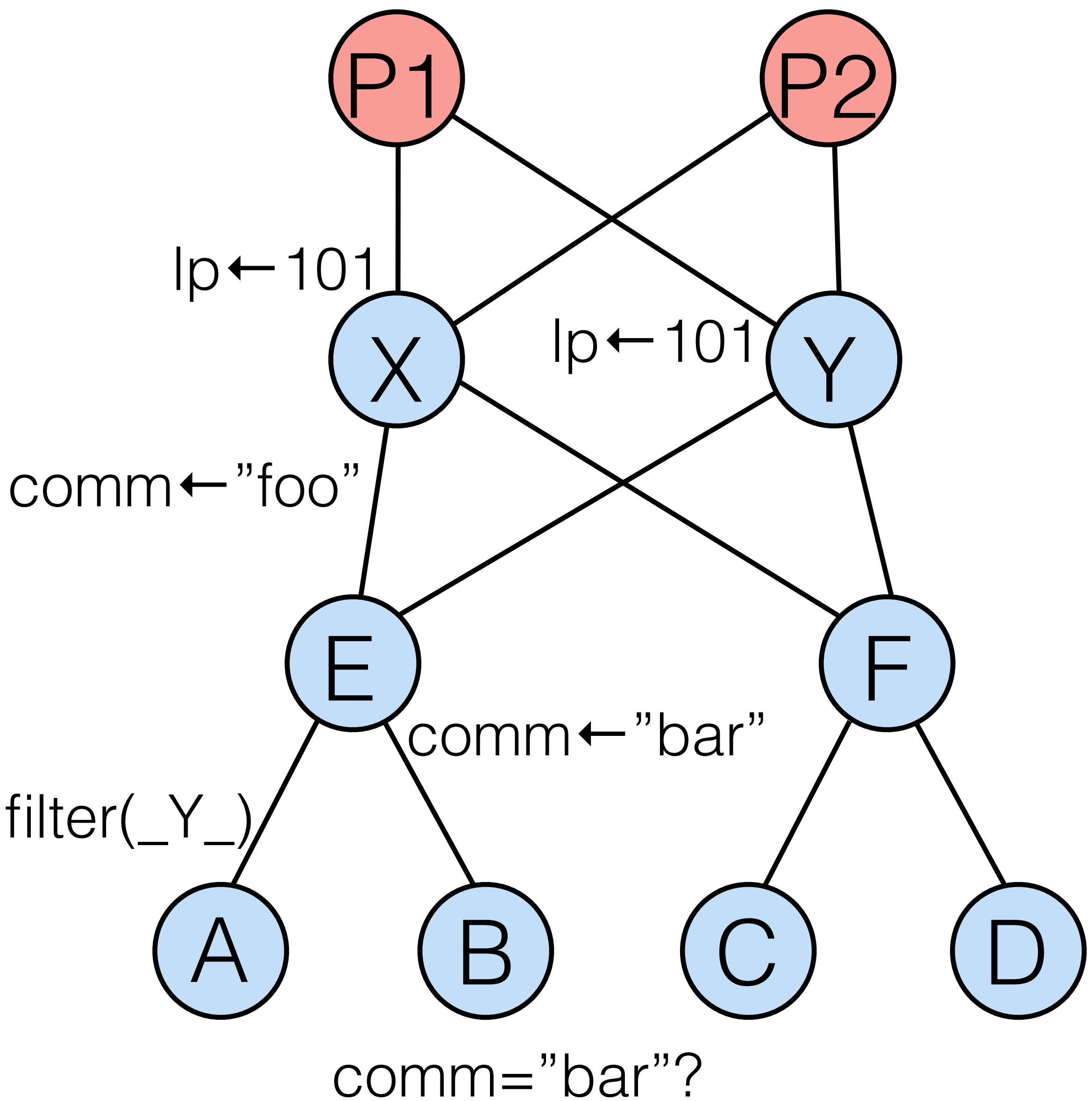
# BGP Configs to PG




**Challenges:**

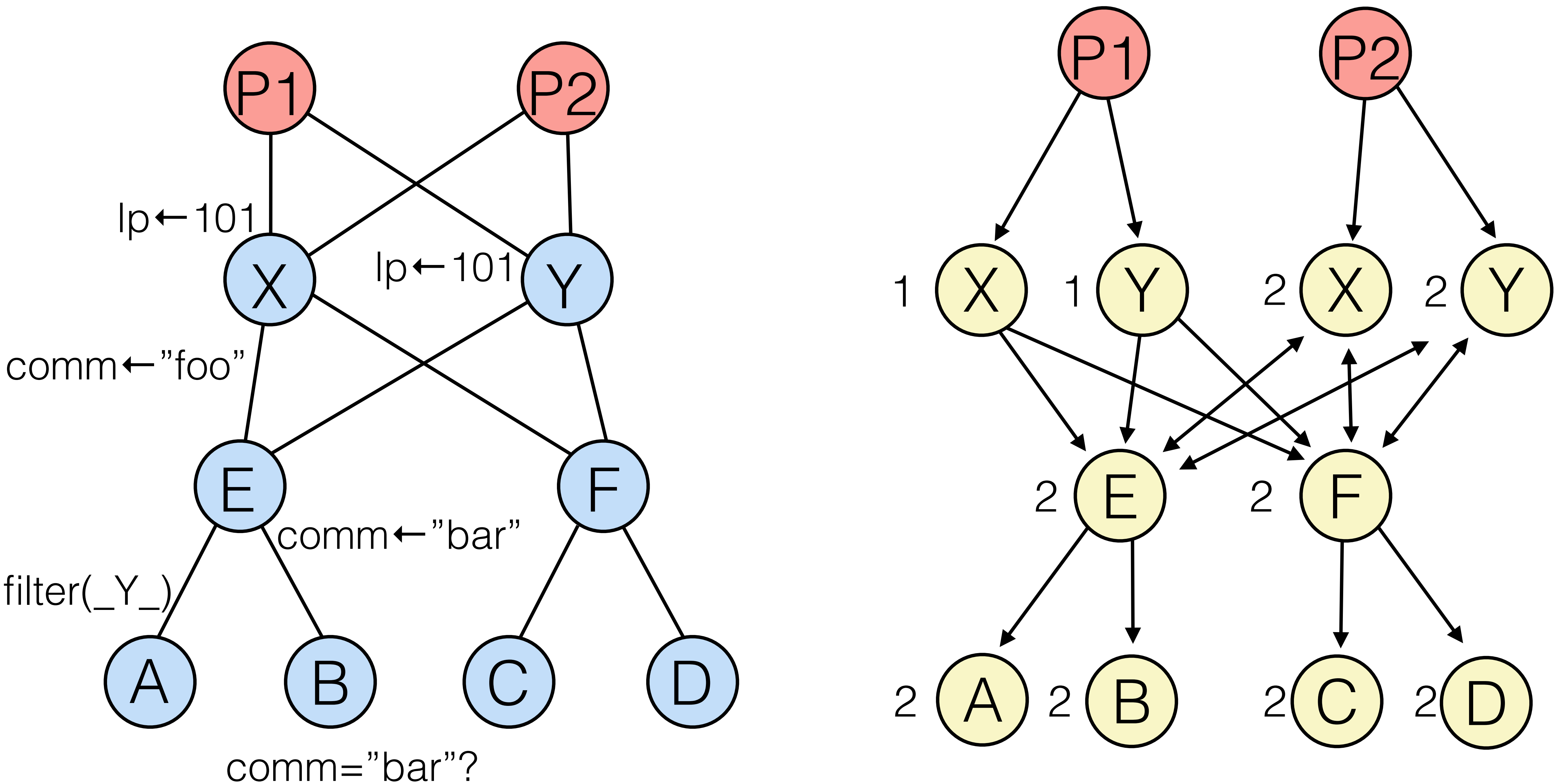
1. Import export filters
2. Arbitrary local preferences
3. Regex filters can occur anywhere
4. Community tags are non-local

# Local Preferences

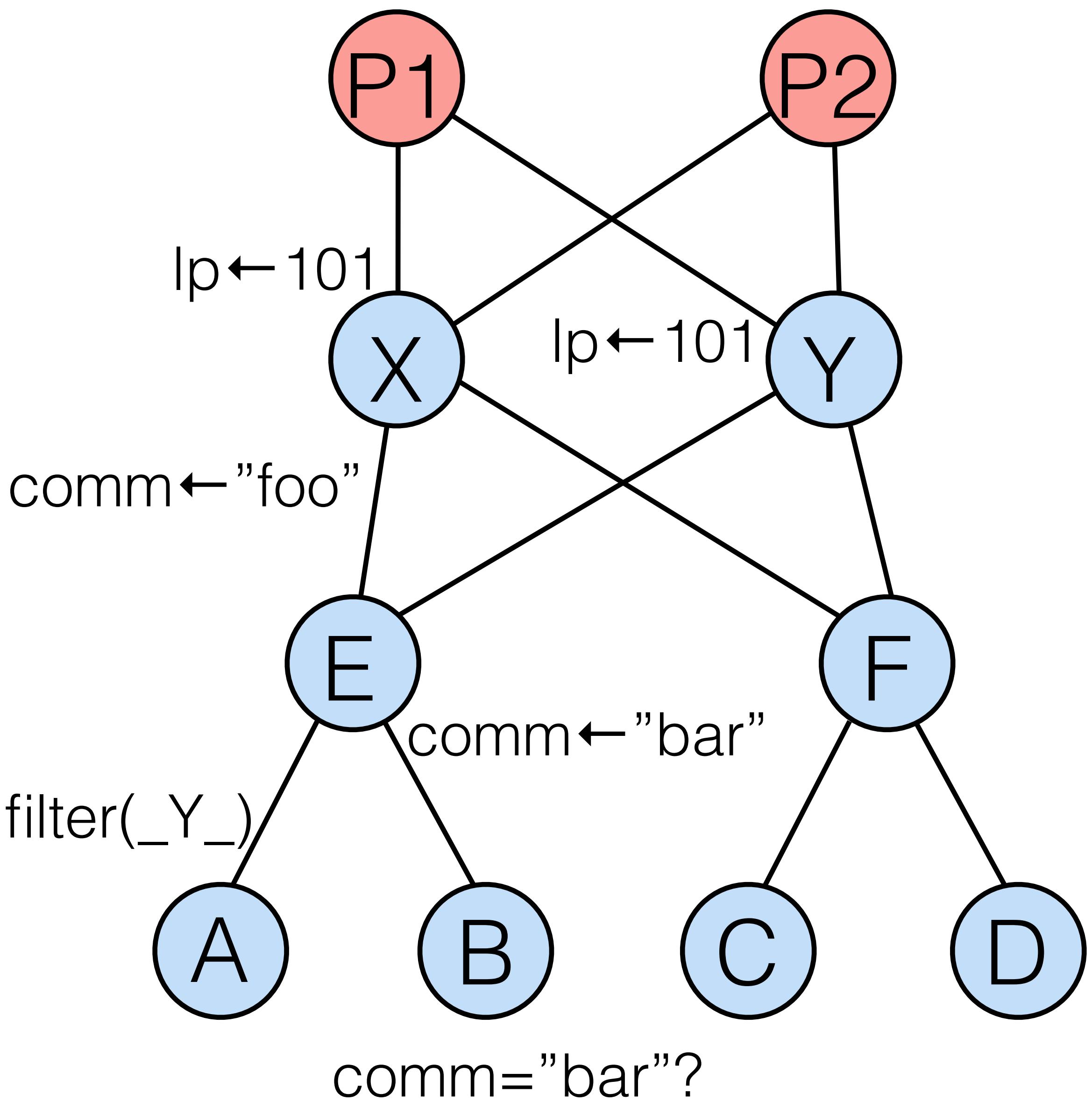



Each unique LP value becomes a unique PG preference!

# Local Preferences



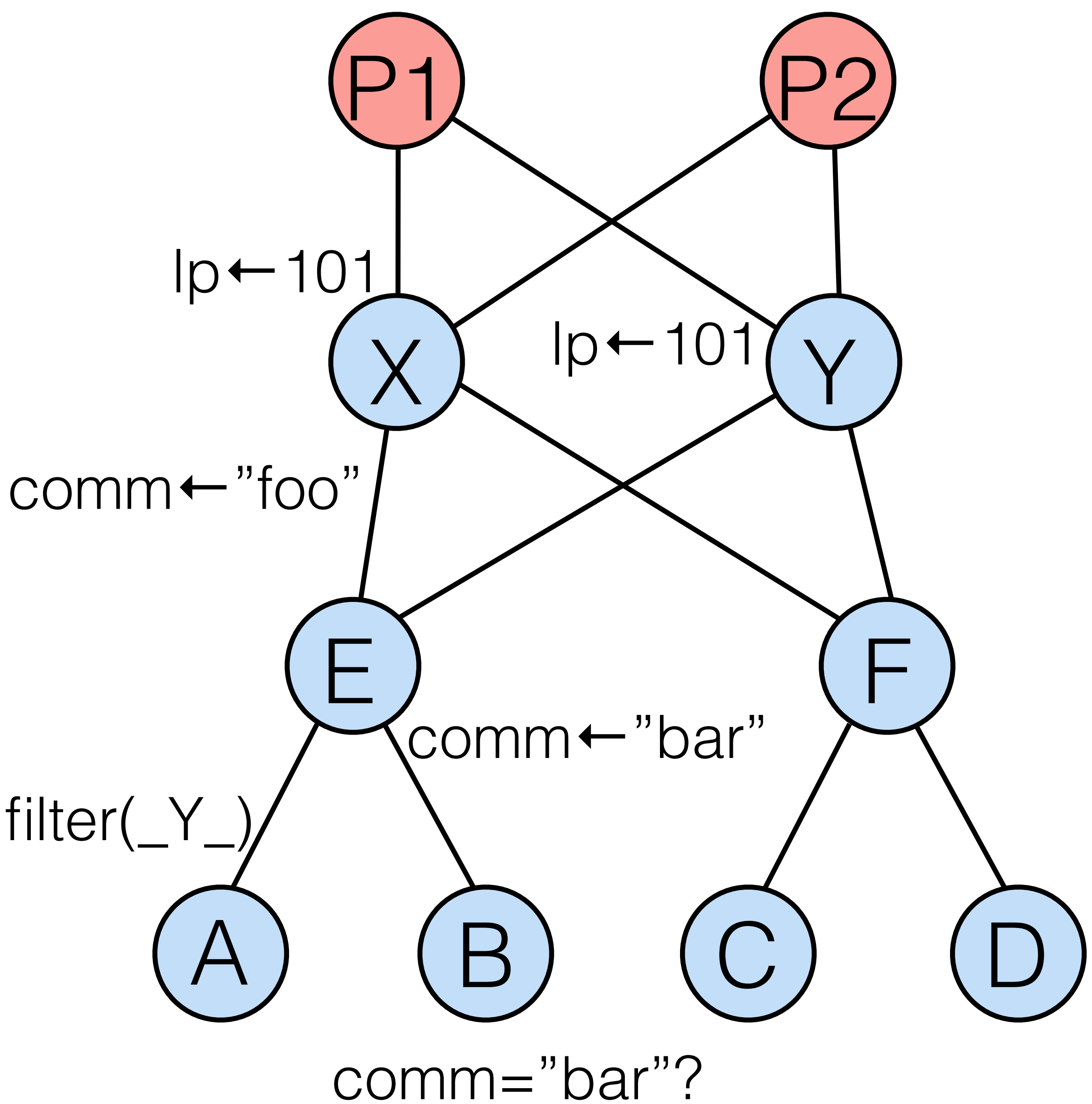
# Regex Filters

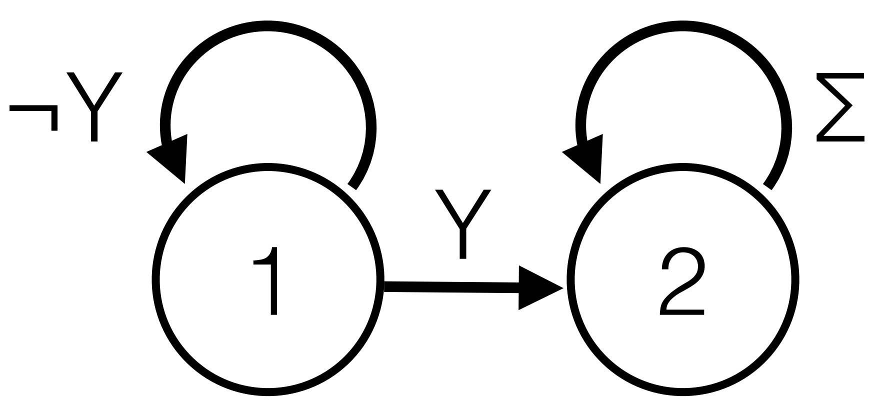


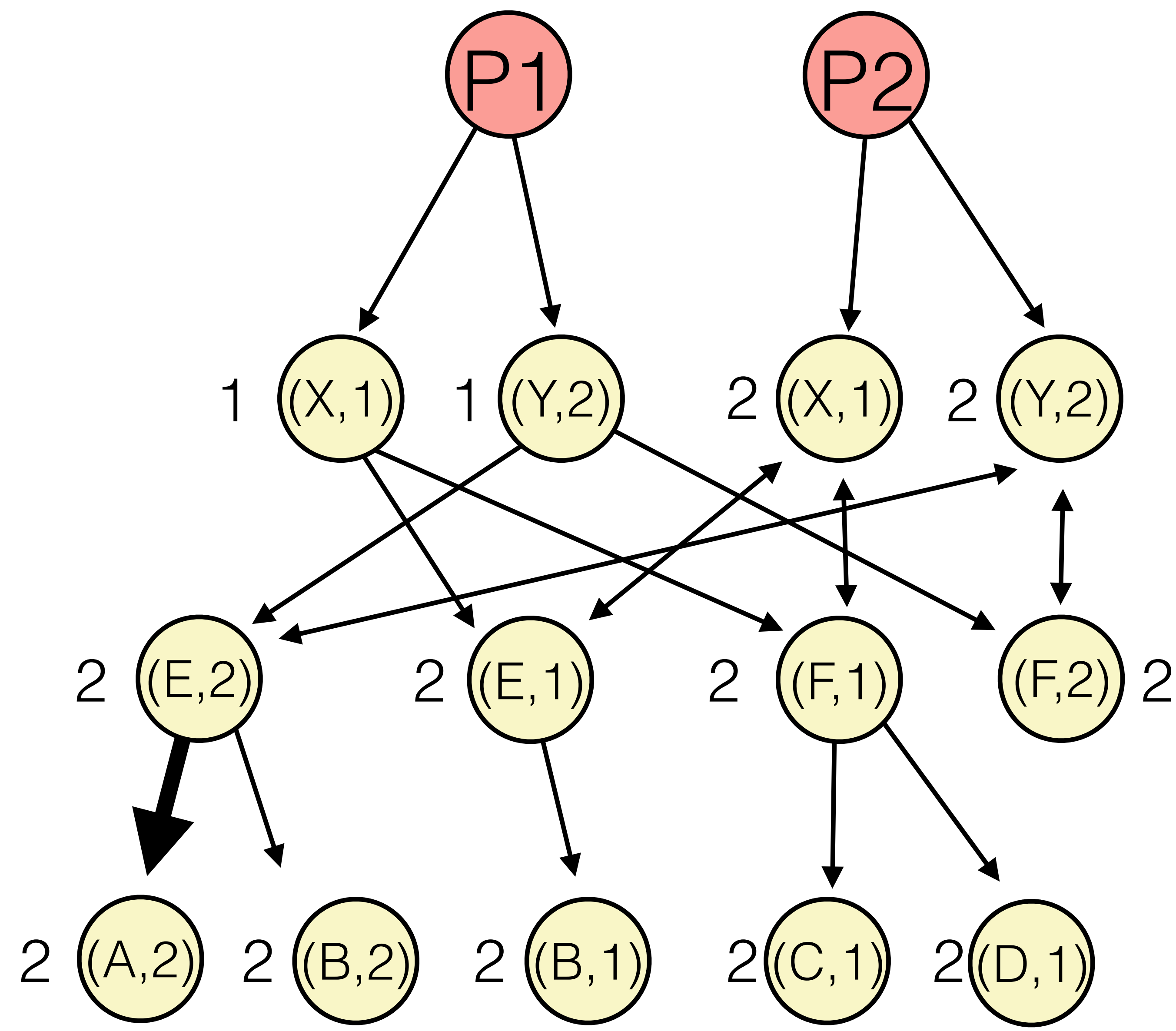

Track the truth of each regex filter while building PG!

Same as when we usually build PG from automata

# Regex Filters

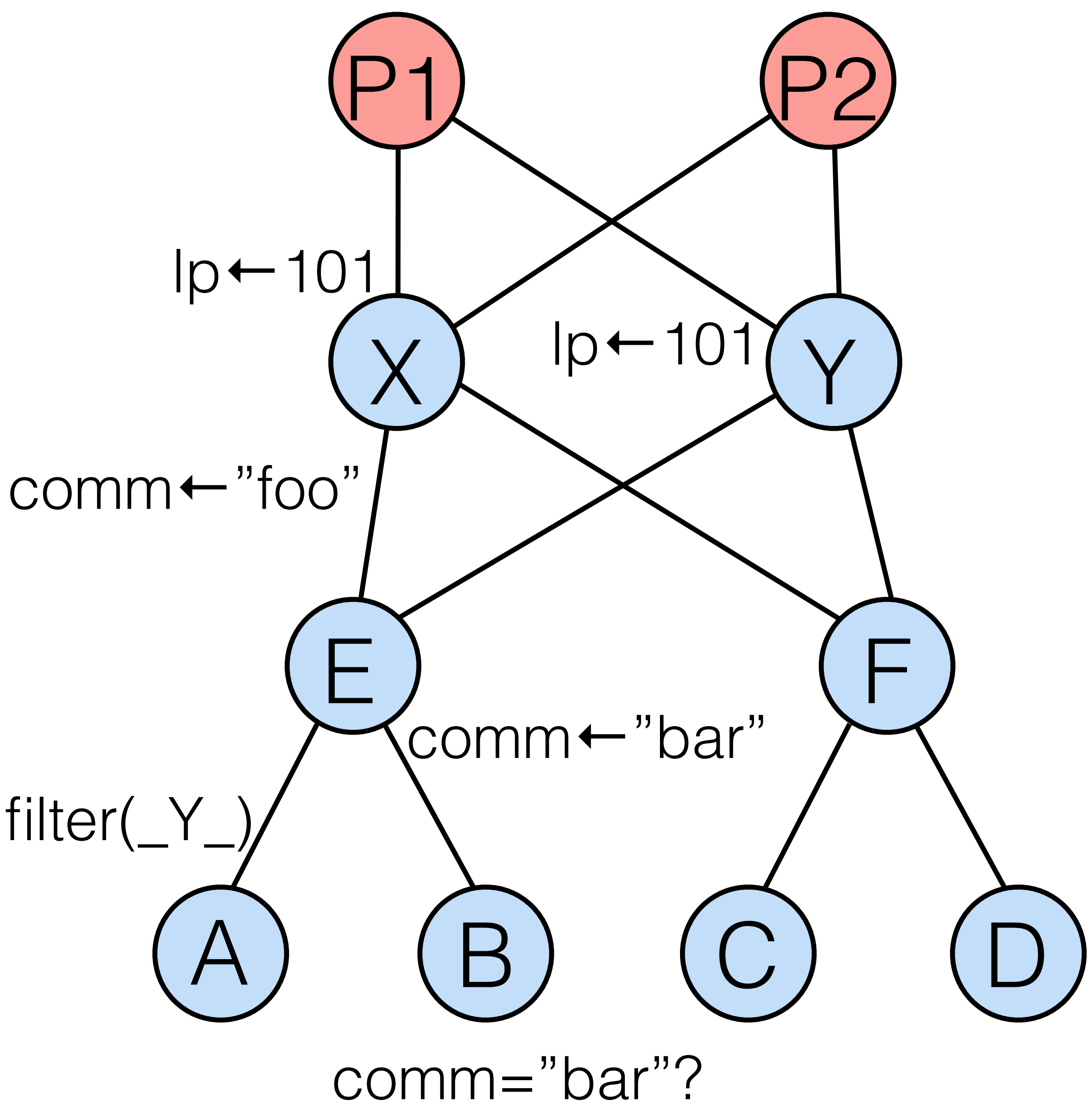


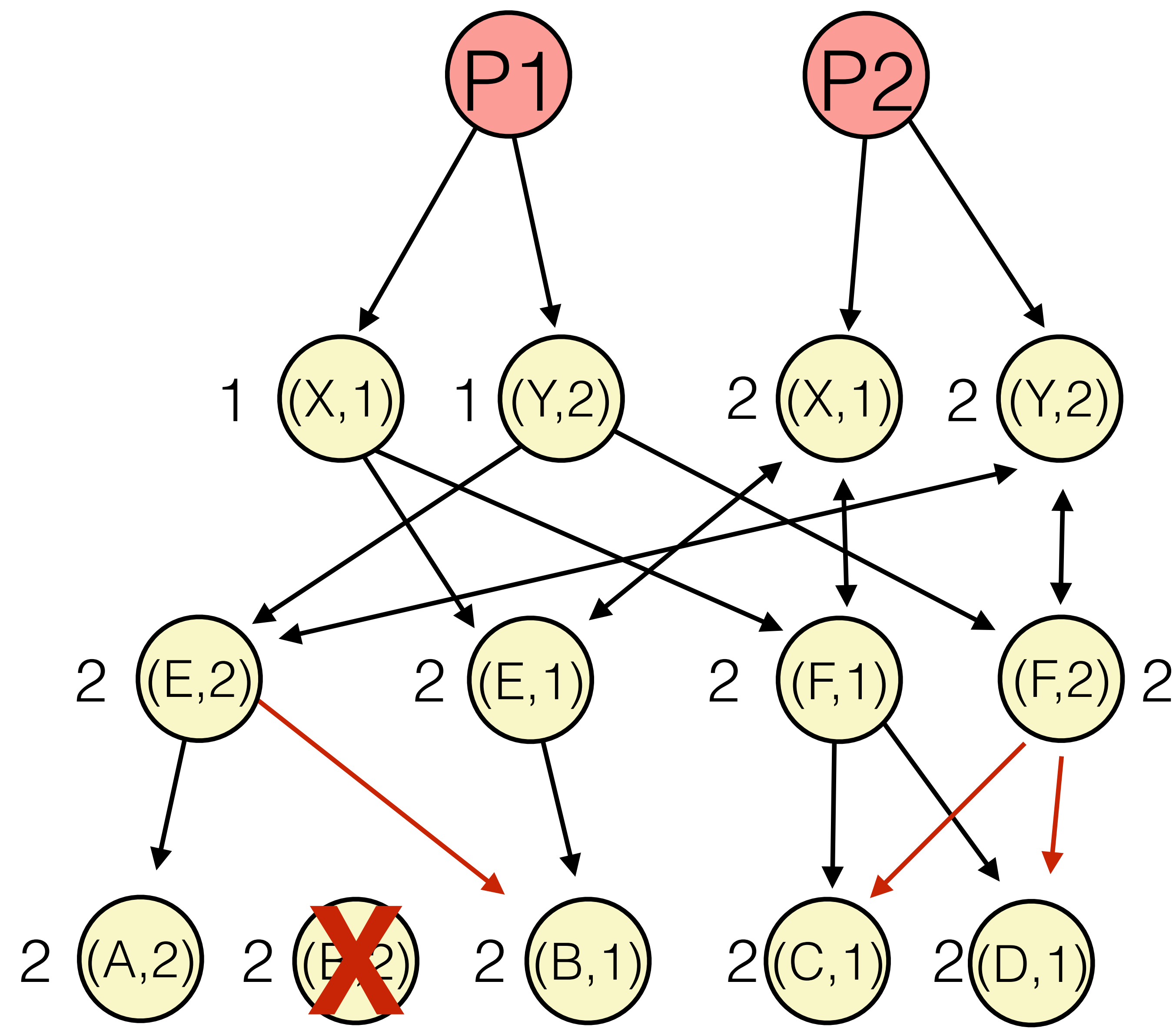



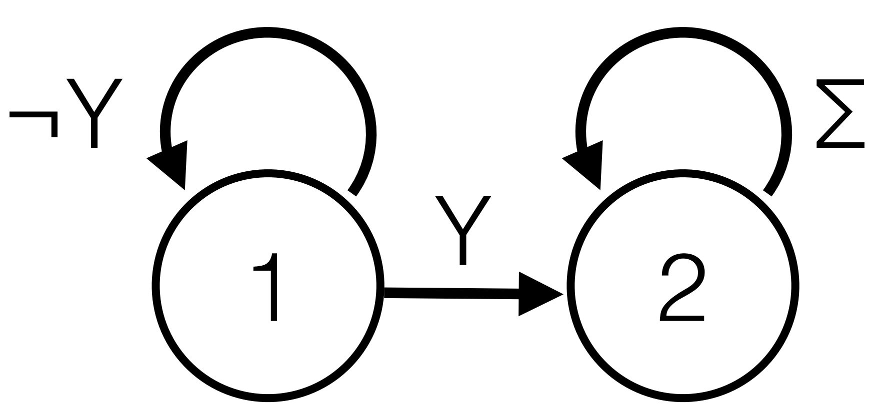




**Filter encoded by (E,2) (A,2) edge, which knows that the regex is satisfied**

# Regex Filters

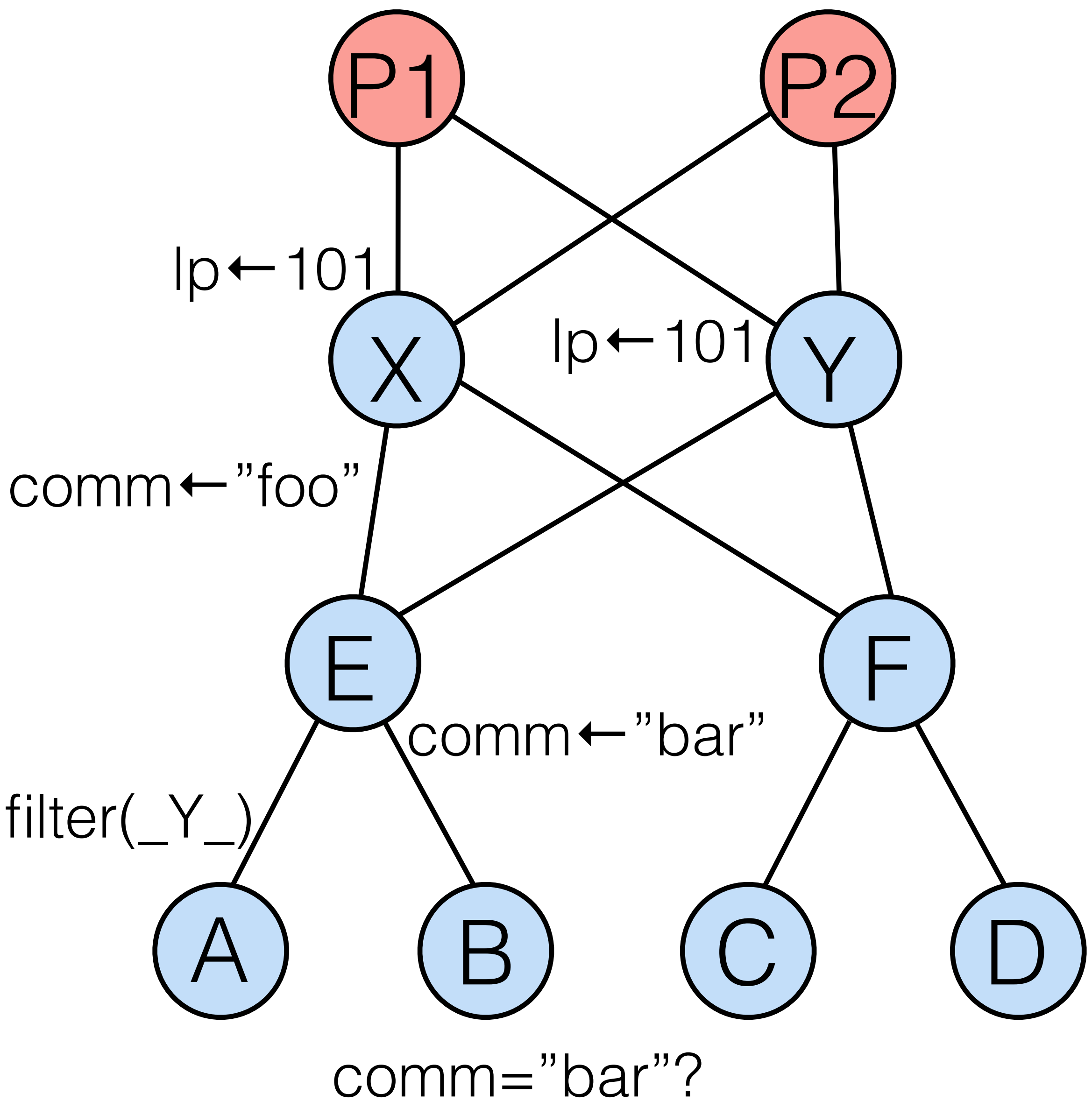



**Possible Optimization:**
merge nodes when tracking information becomes irrelevant?

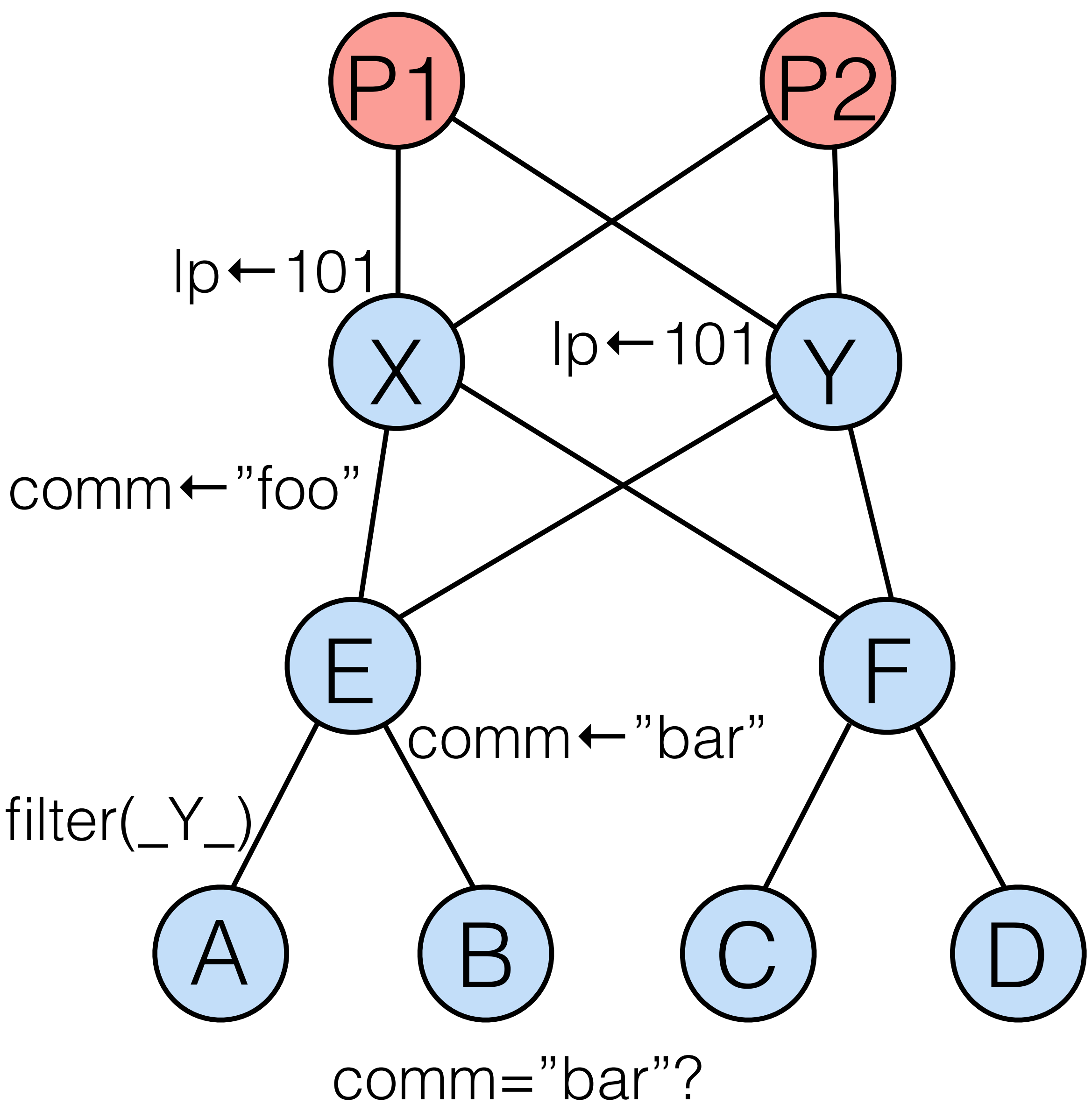
# Community Tags

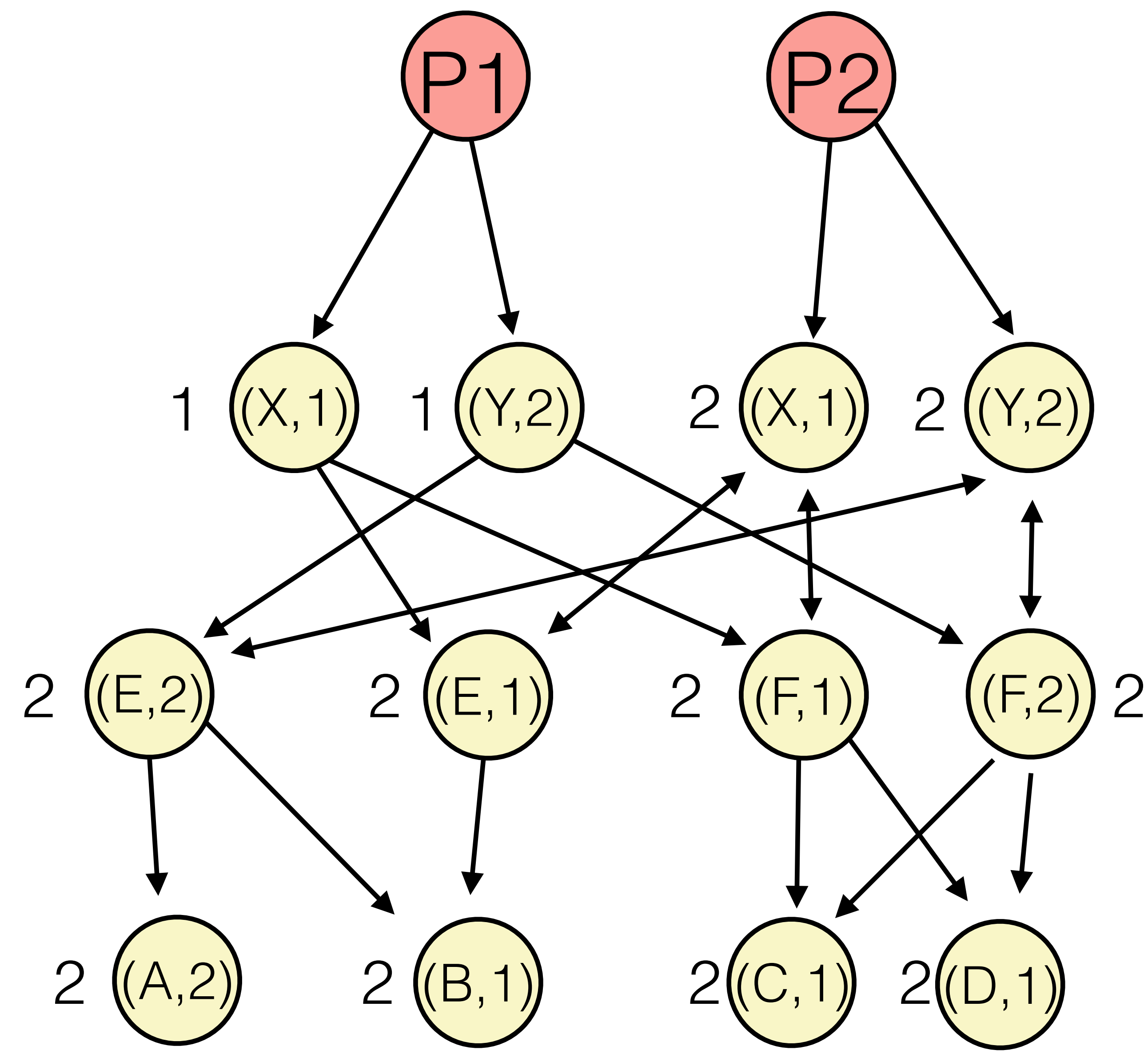



Track which community tags are attached in PG

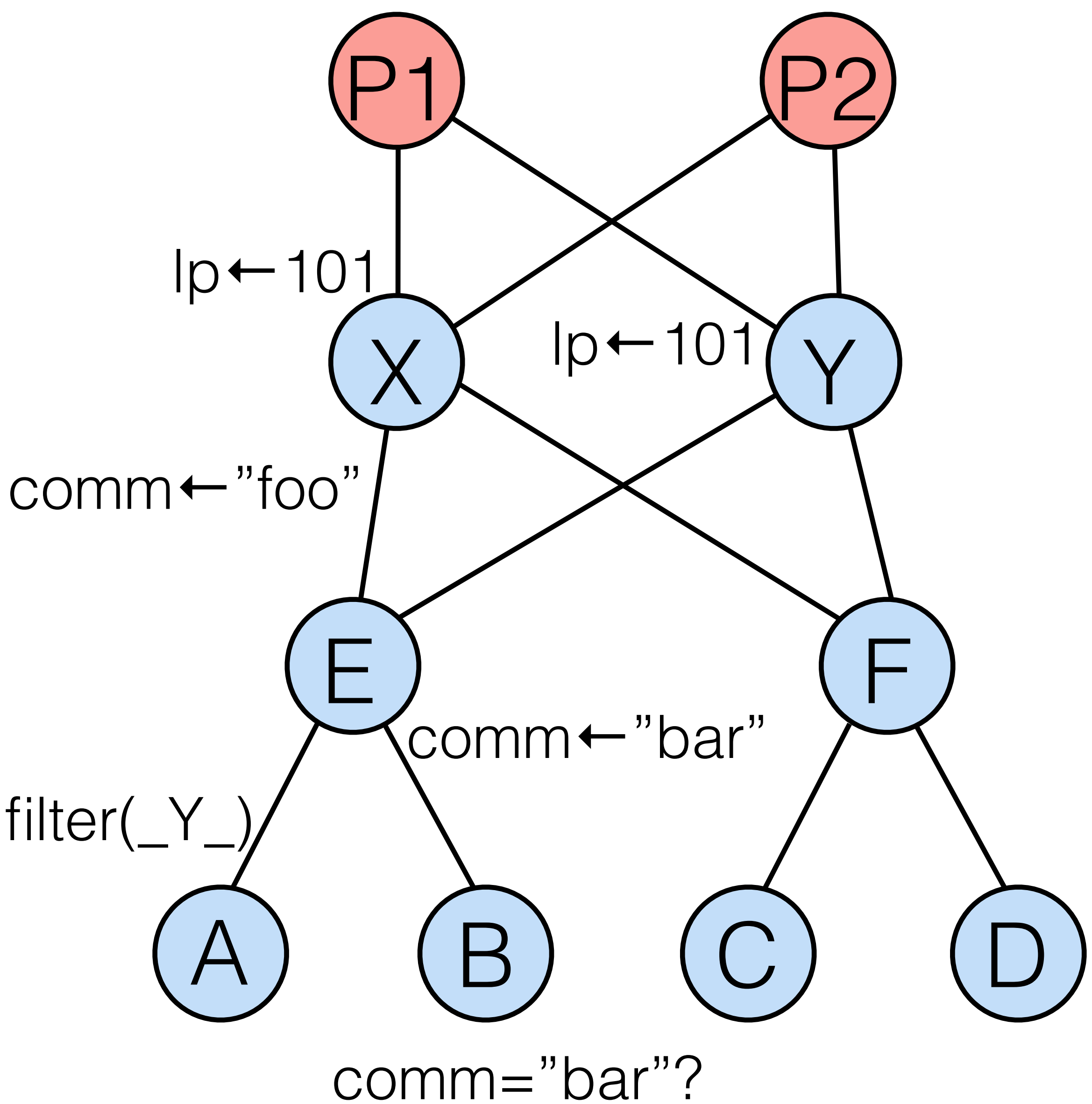
Same idea as before!

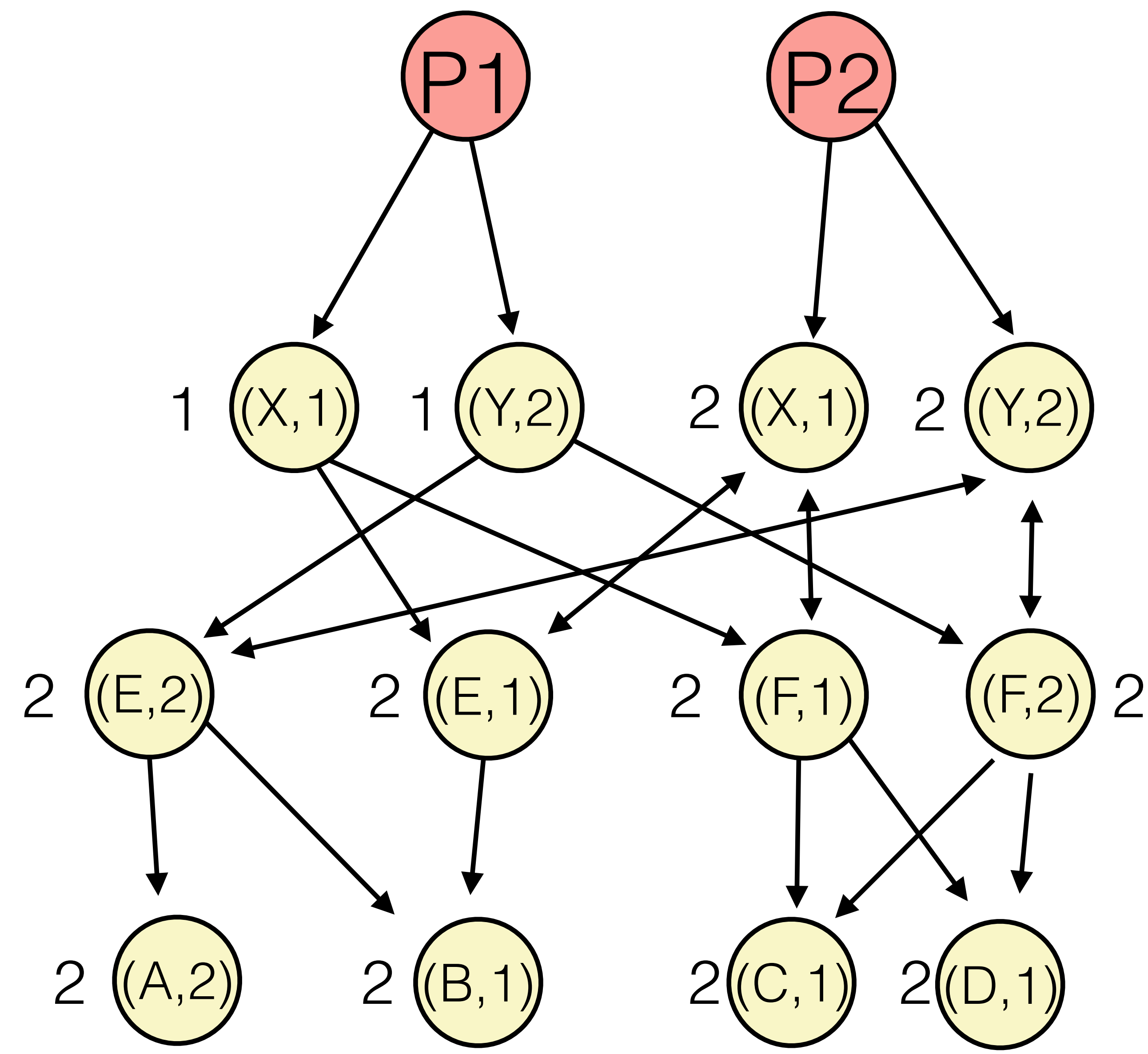
# Community Tags



# Community Tags



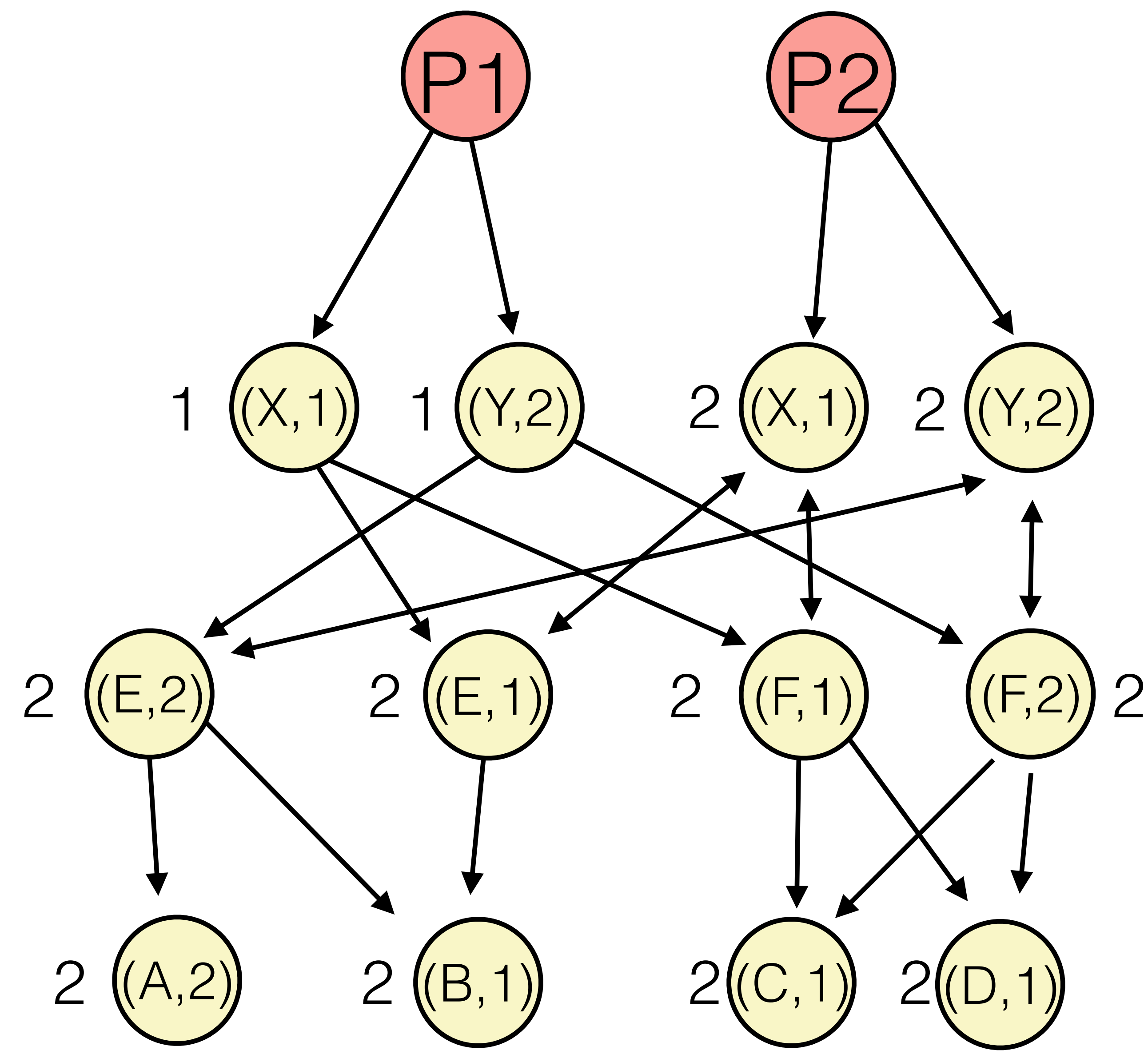




Tag "foo" X —> E

Tag "bar" E —> B when "foo" attached

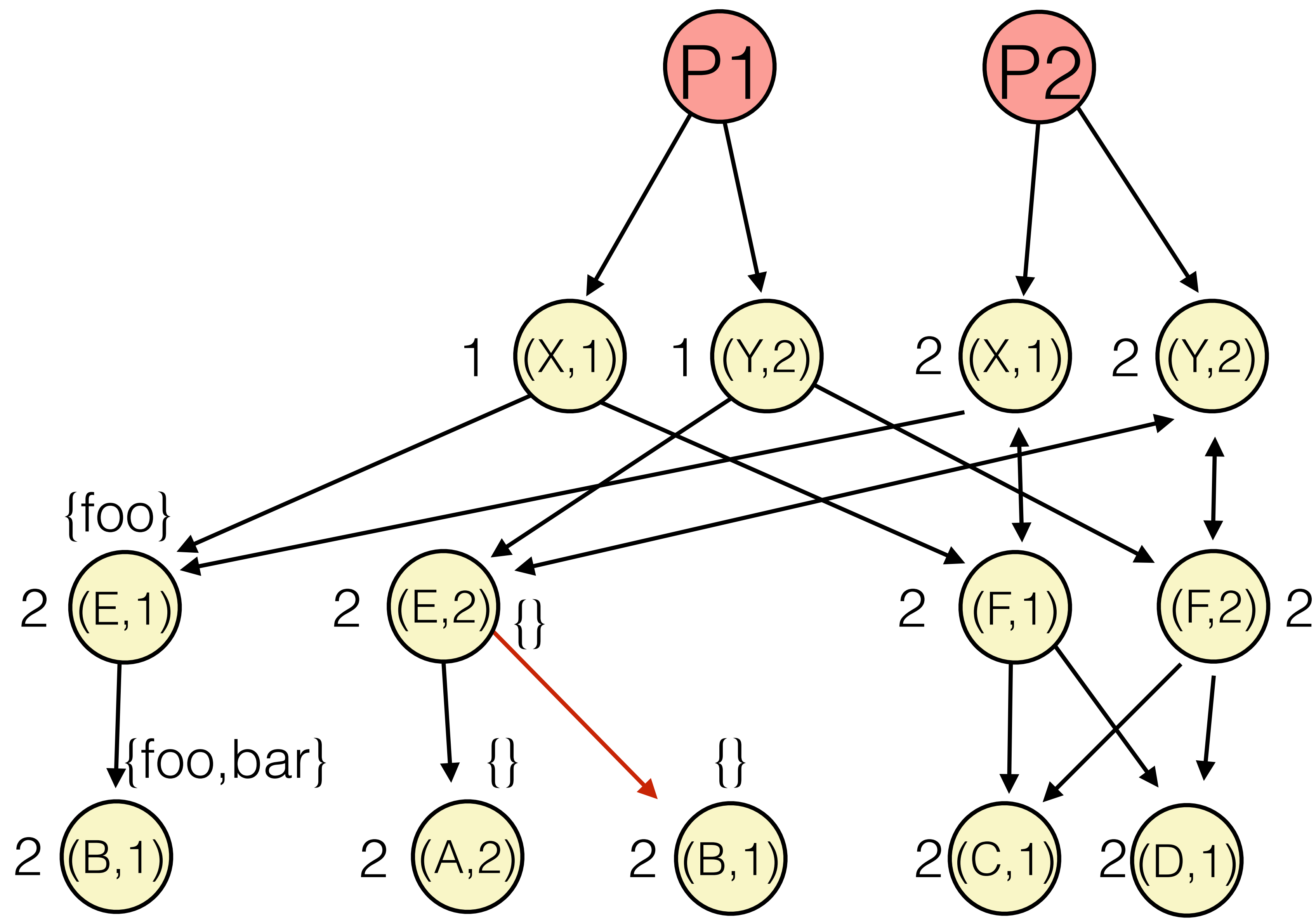
Filter for "bar" from B <— E

# Community Tags




Tag “foo” X —> E
Tag “bar” E —> B when “foo” attached
Filter for “bar” from B <— E

# Community Tags



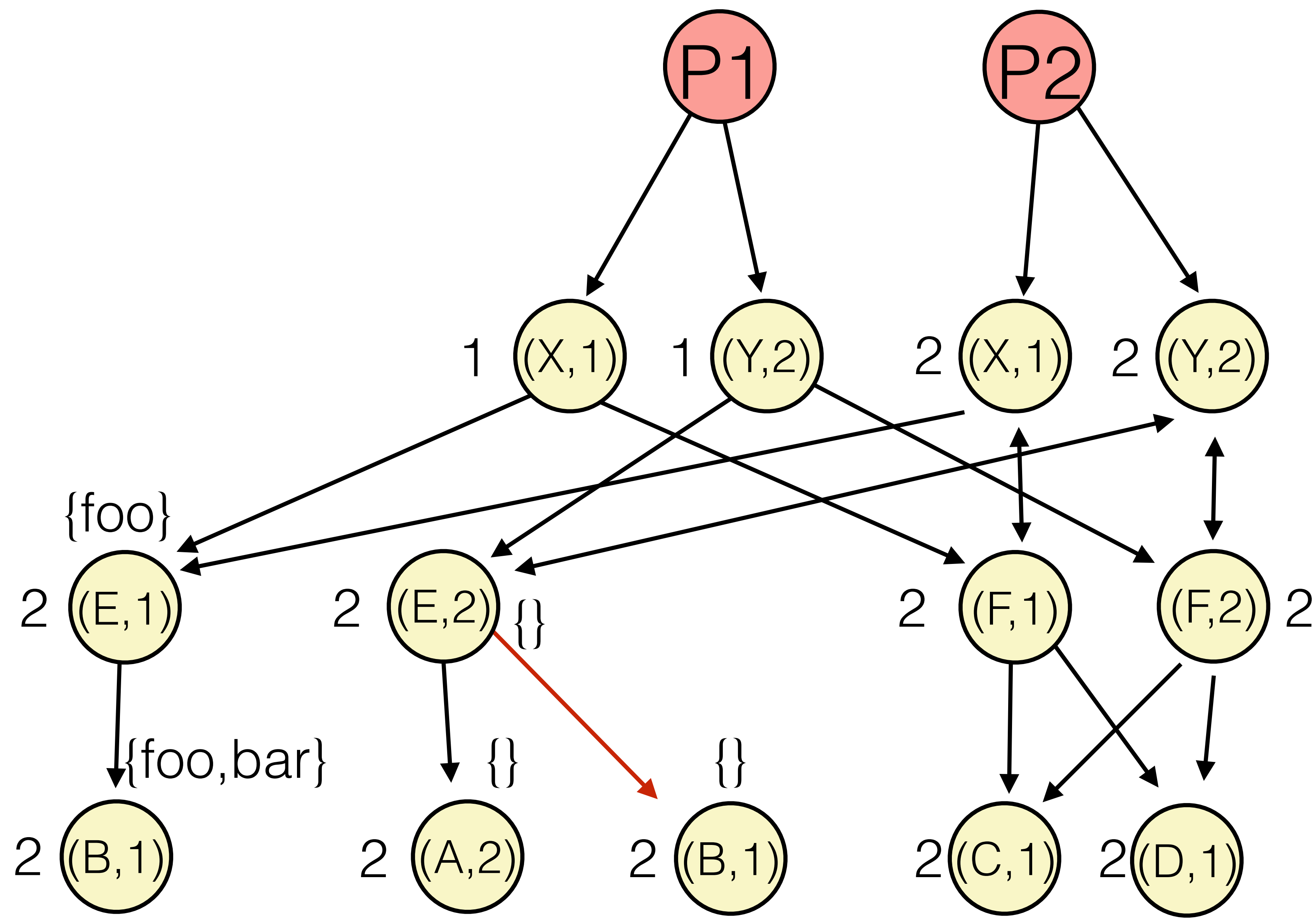

Tag “foo” X —> E

Tag “bar” E —> B when “foo” attached

Filter for “bar” from B <— E

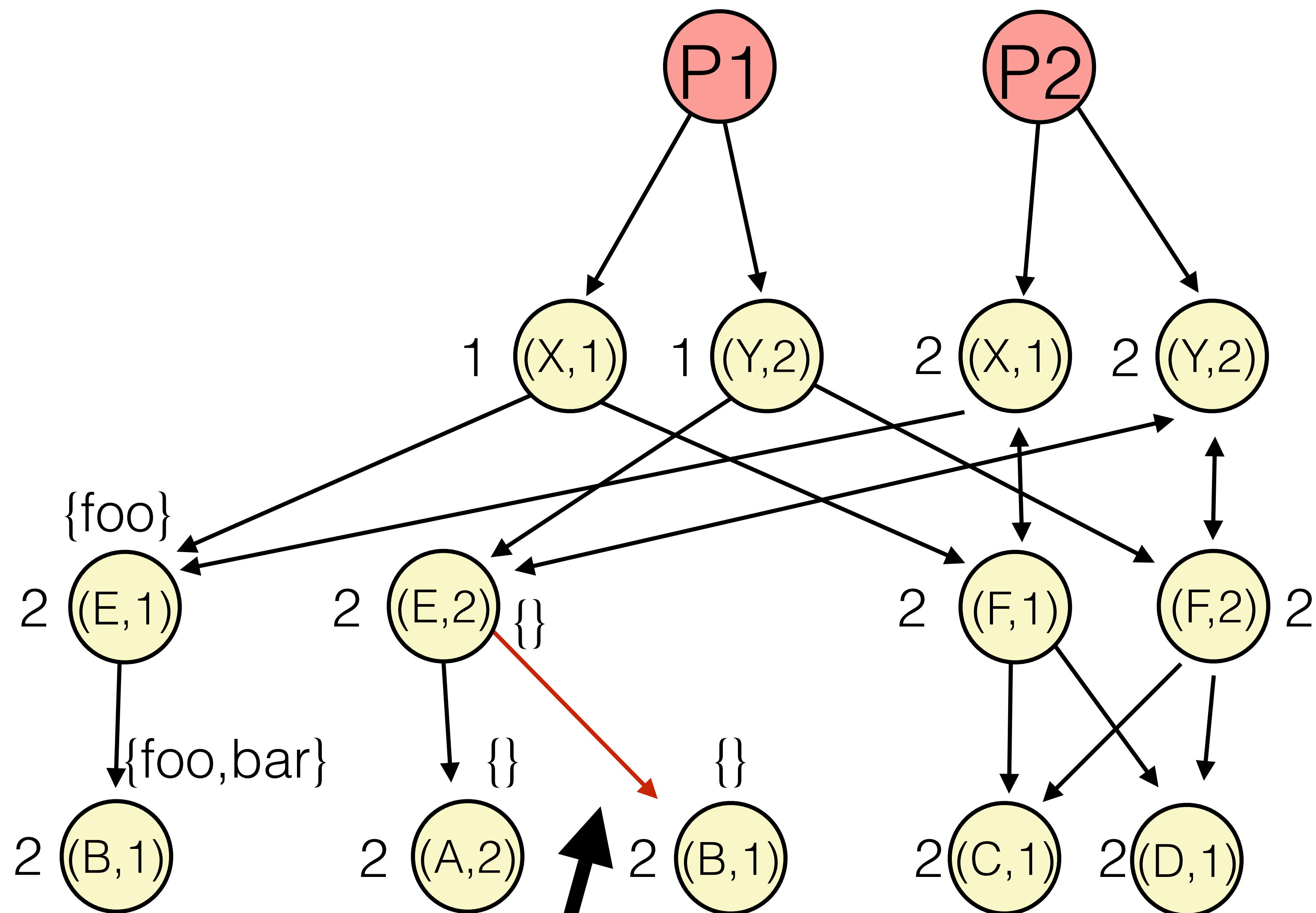
# Community Tags

Edge allowed
since “bar”
known to be attached




Tag “foo” X —> E
Tag “bar” E —> B when “foo” attached
Filter for “bar” from B <— E

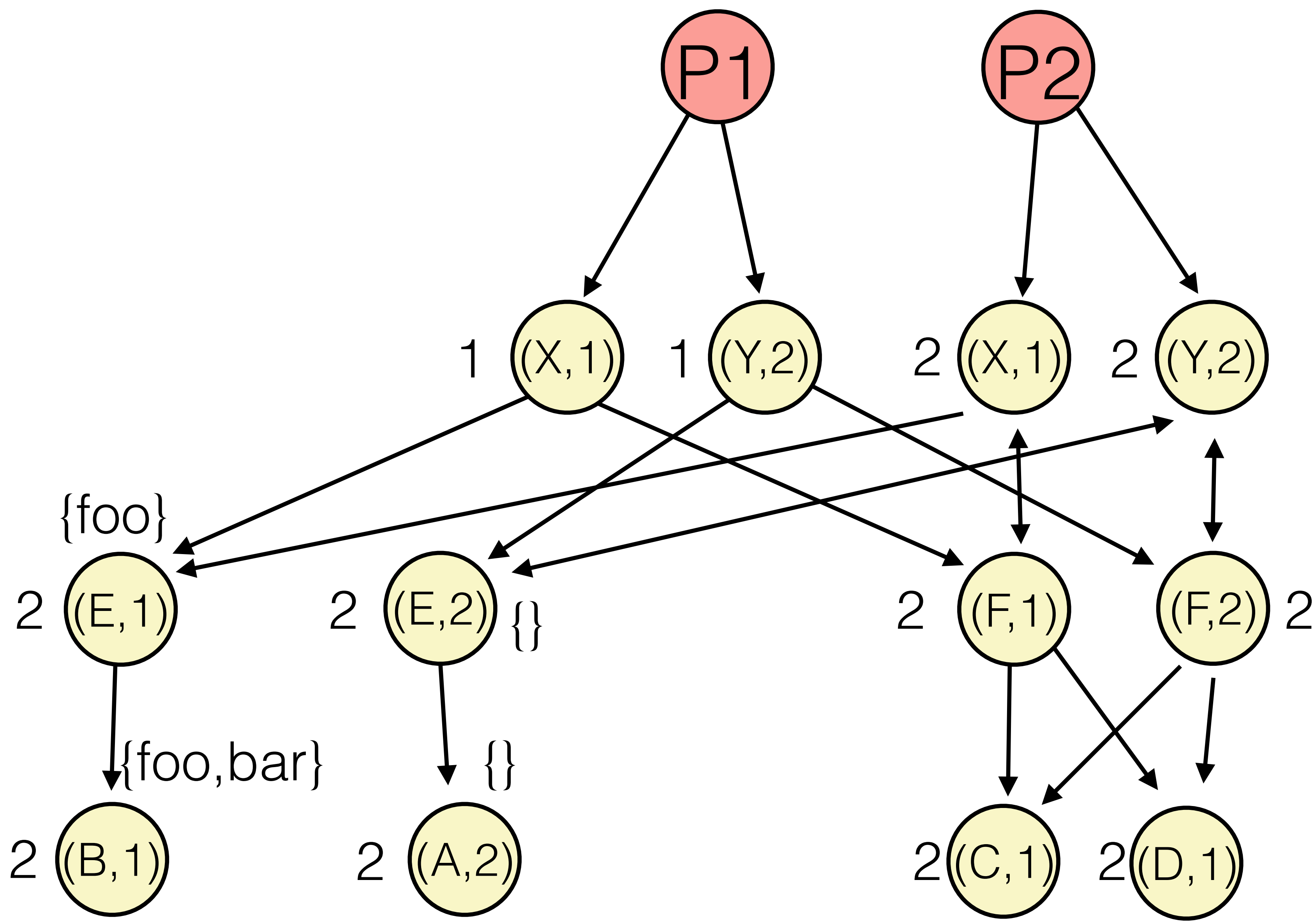
# Community Tags




No edge allowed here

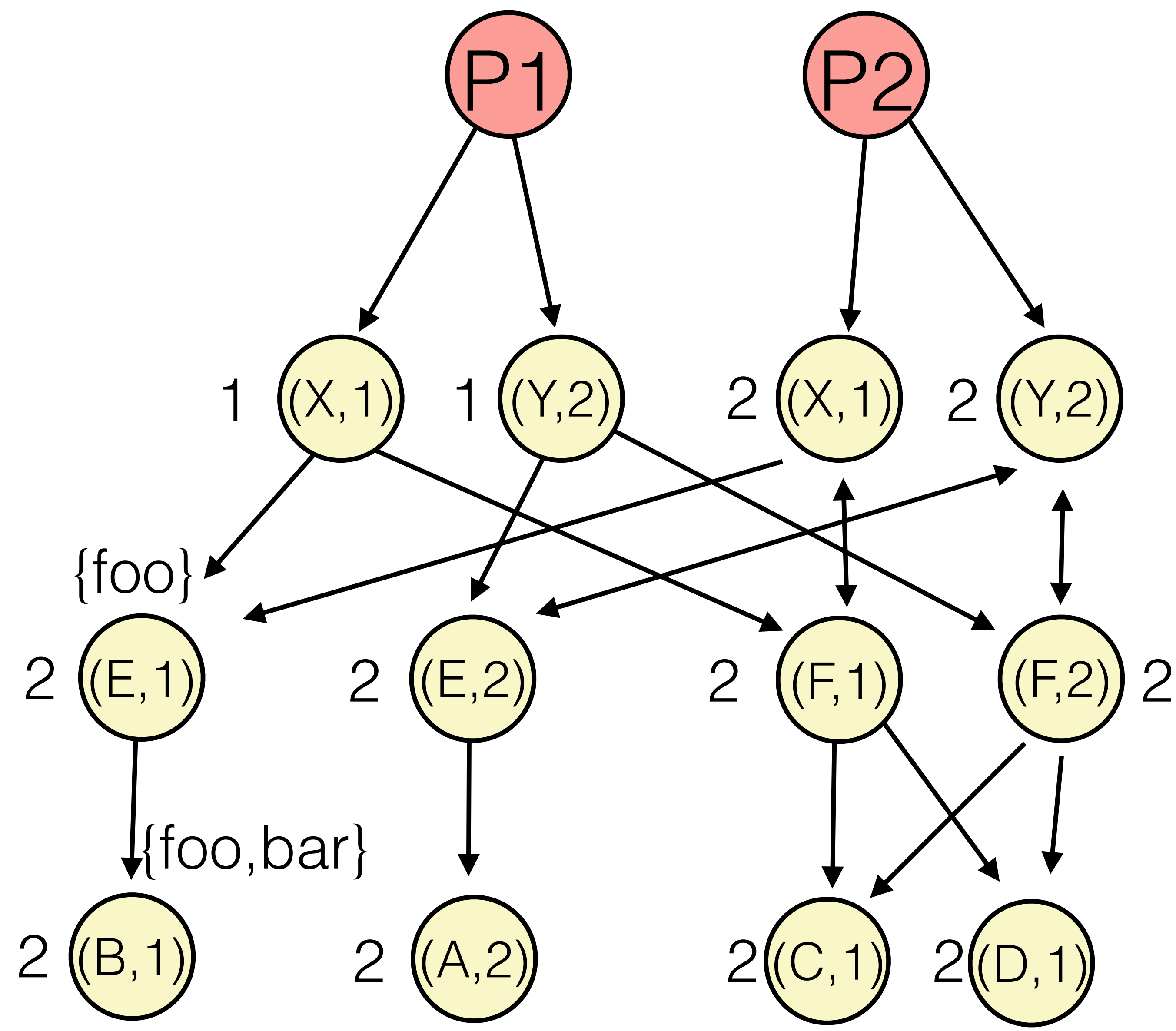
# Community Tags

Can support removing community tags as well



# Community Tags

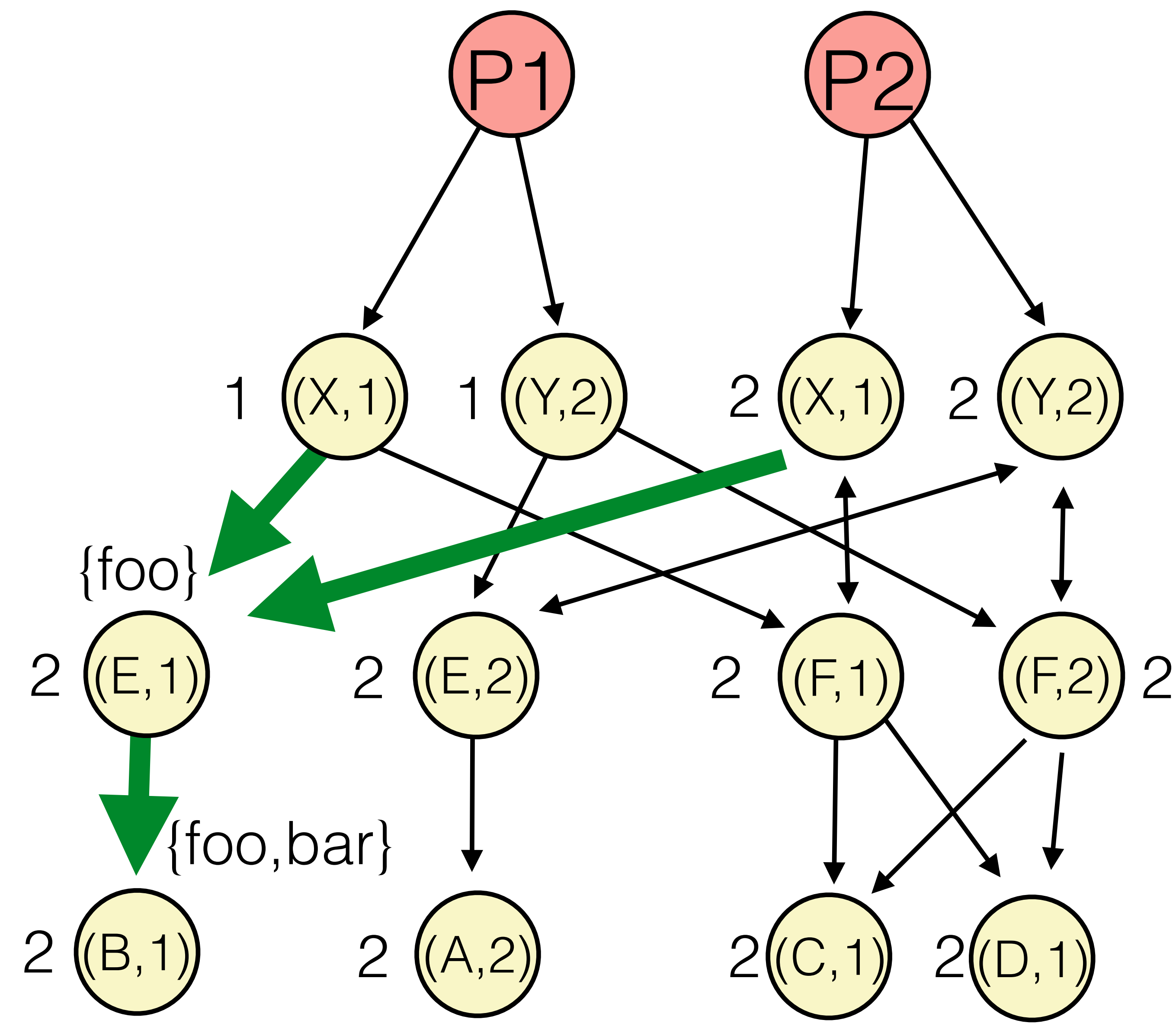
Does traffic sent from B always go through X?



# Community Tags

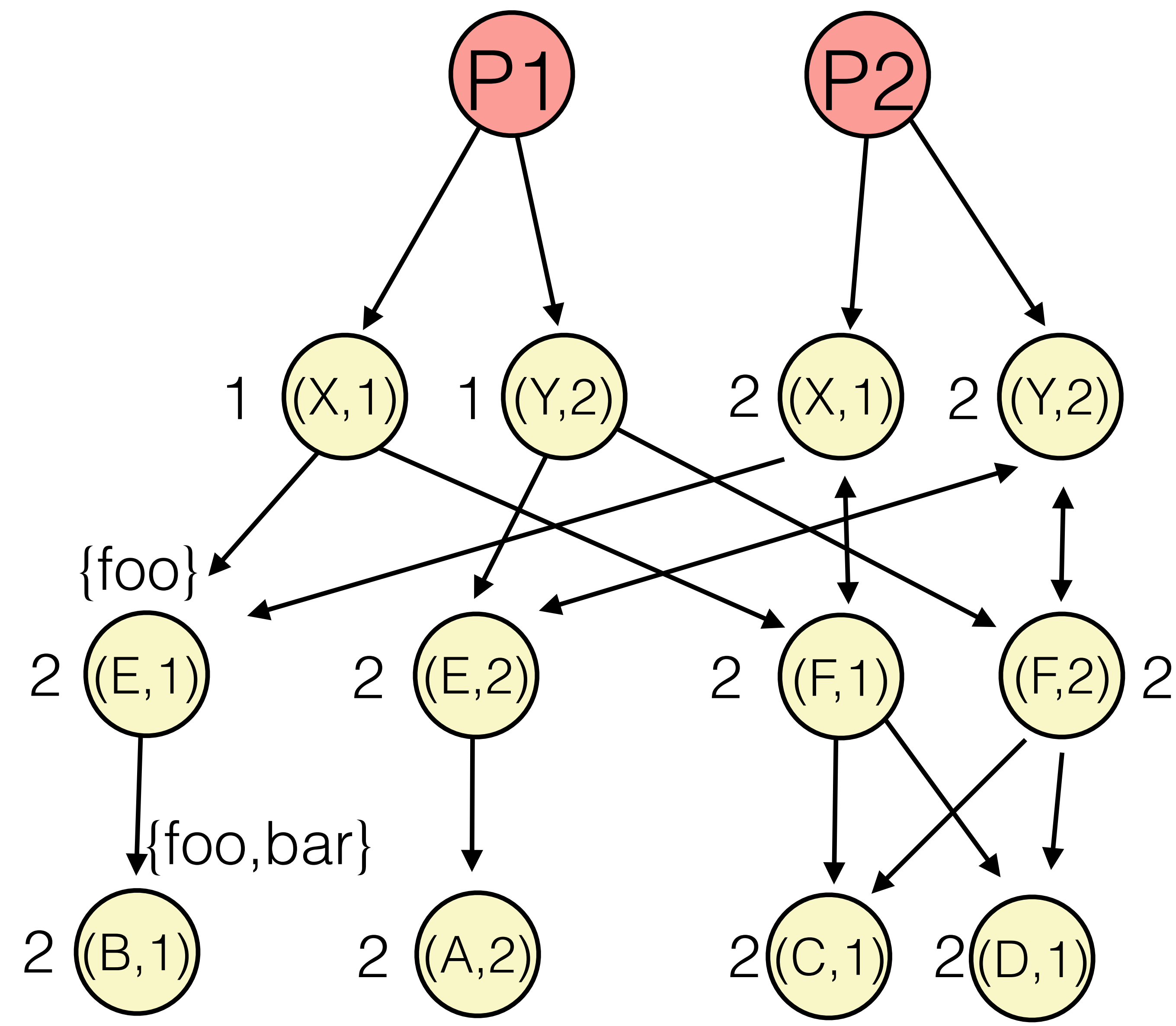
Does traffic sent from B always go through X?

**Yes!**



# Community Tags

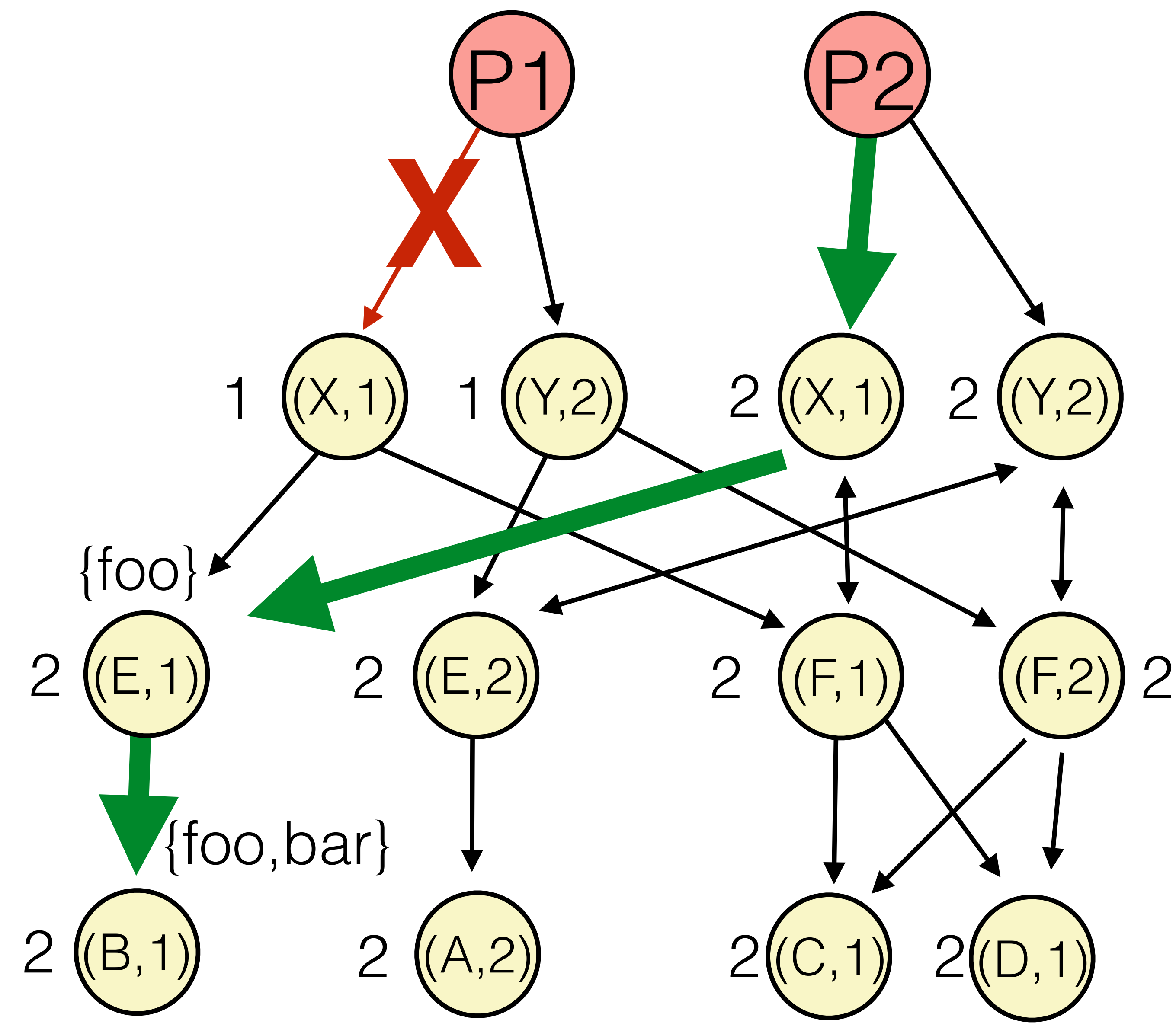
Does traffic always leave the DC through P1 when this is possible?



# Community Tags

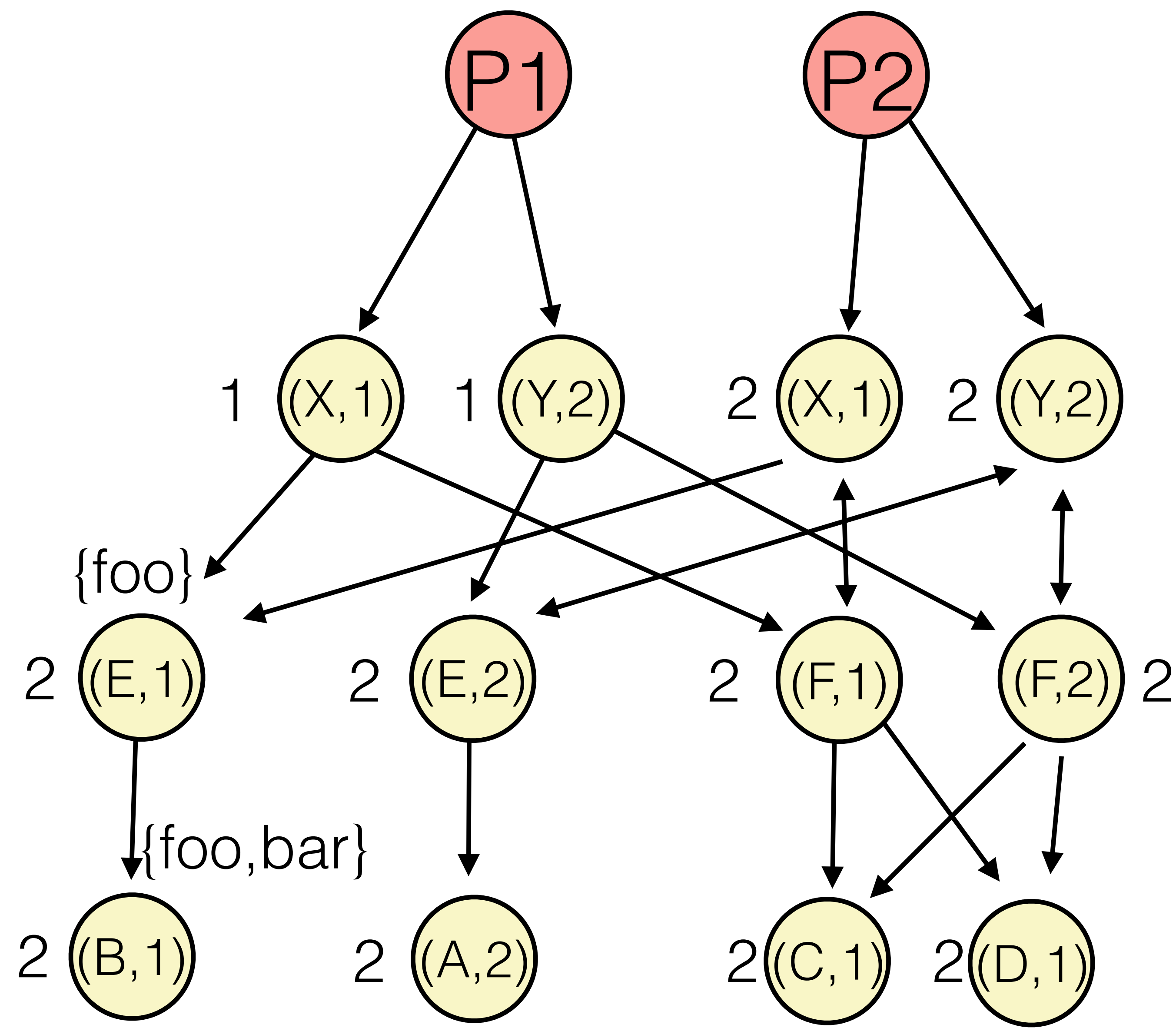
Does traffic always leave the DC through P1 when this is possible?

**Nope!**



# Community Tags

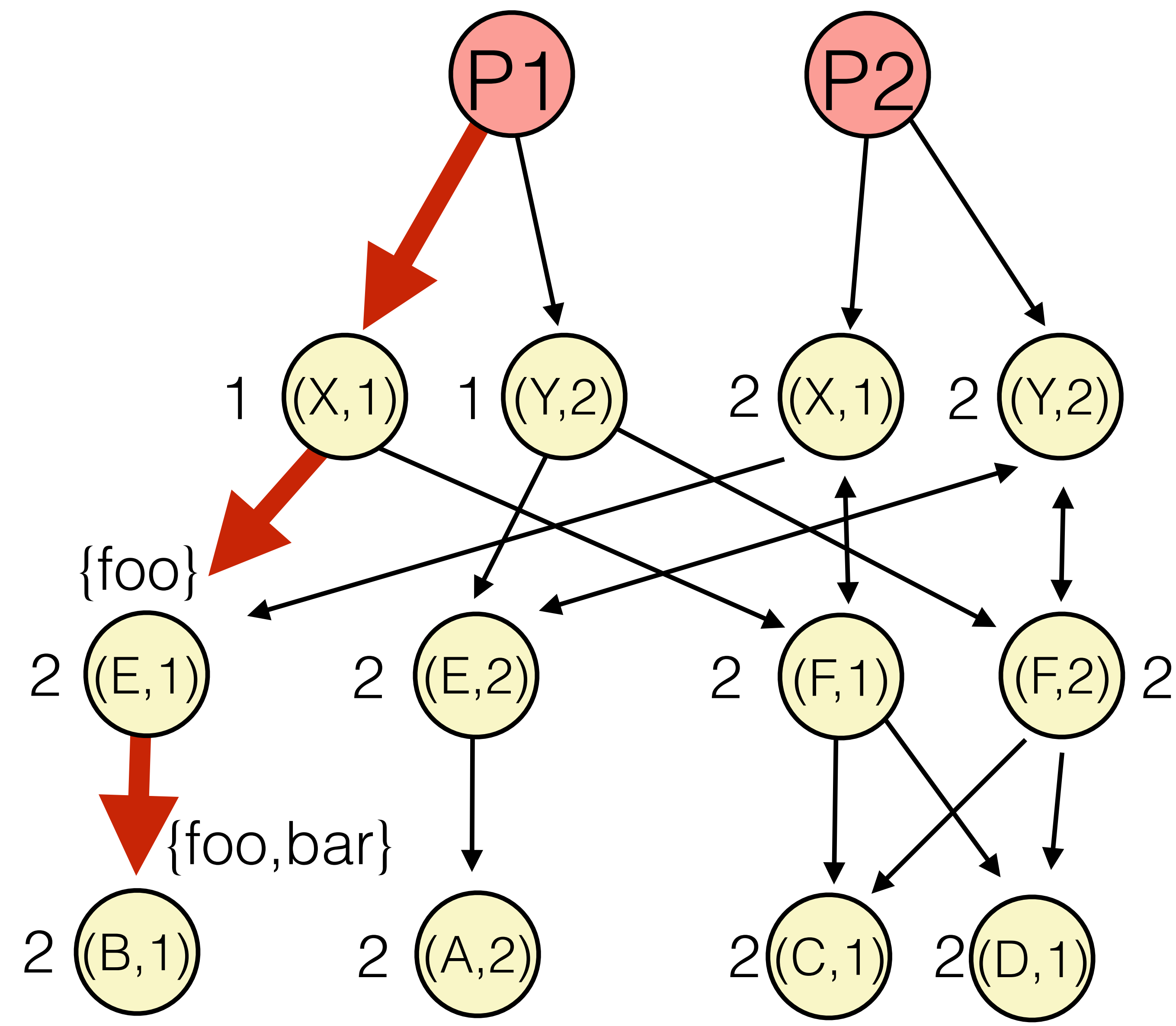
How many failures to disconnect B?



# Community Tags

How many failures to disconnect B?

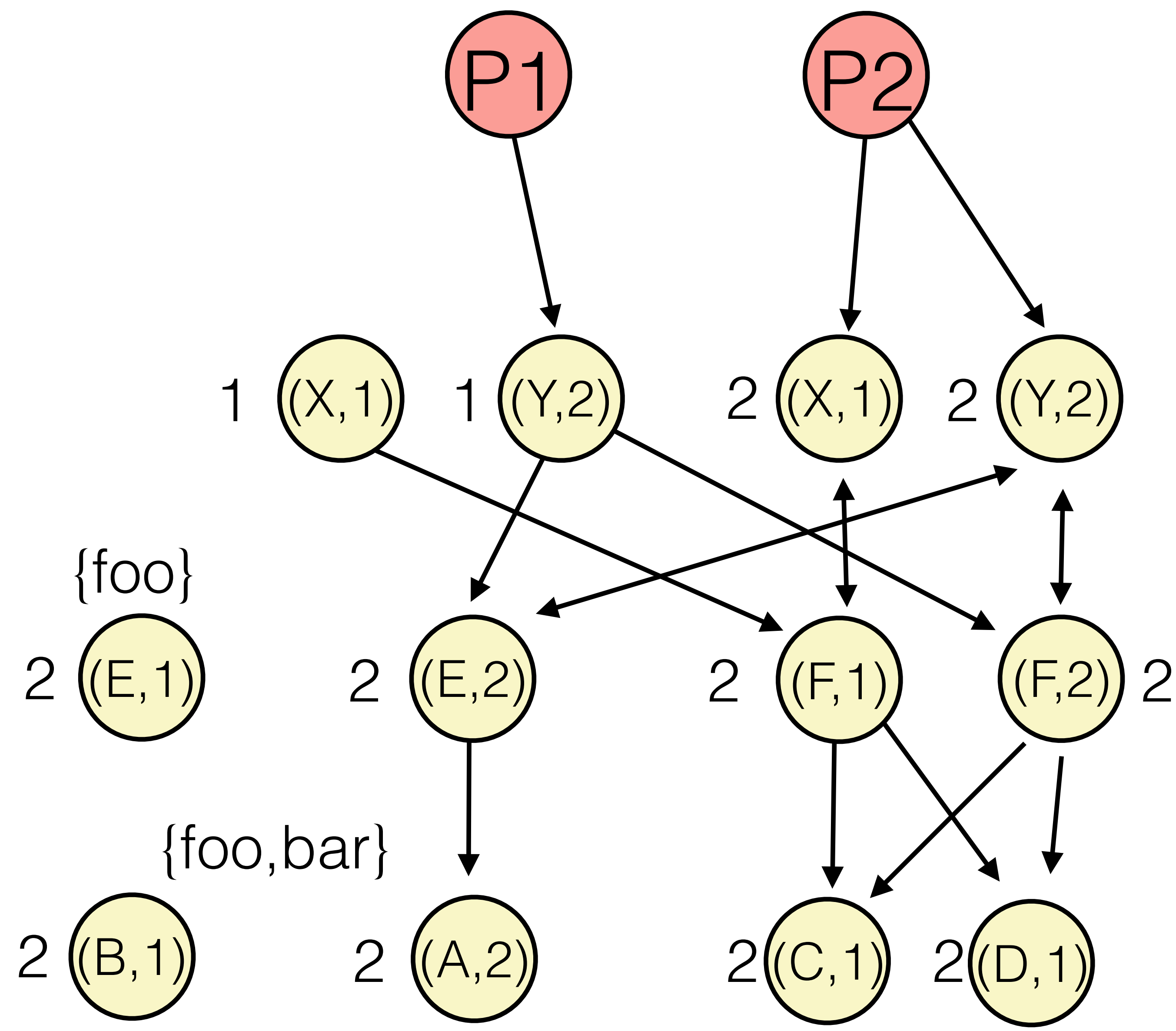
**1!**



# Community Tags

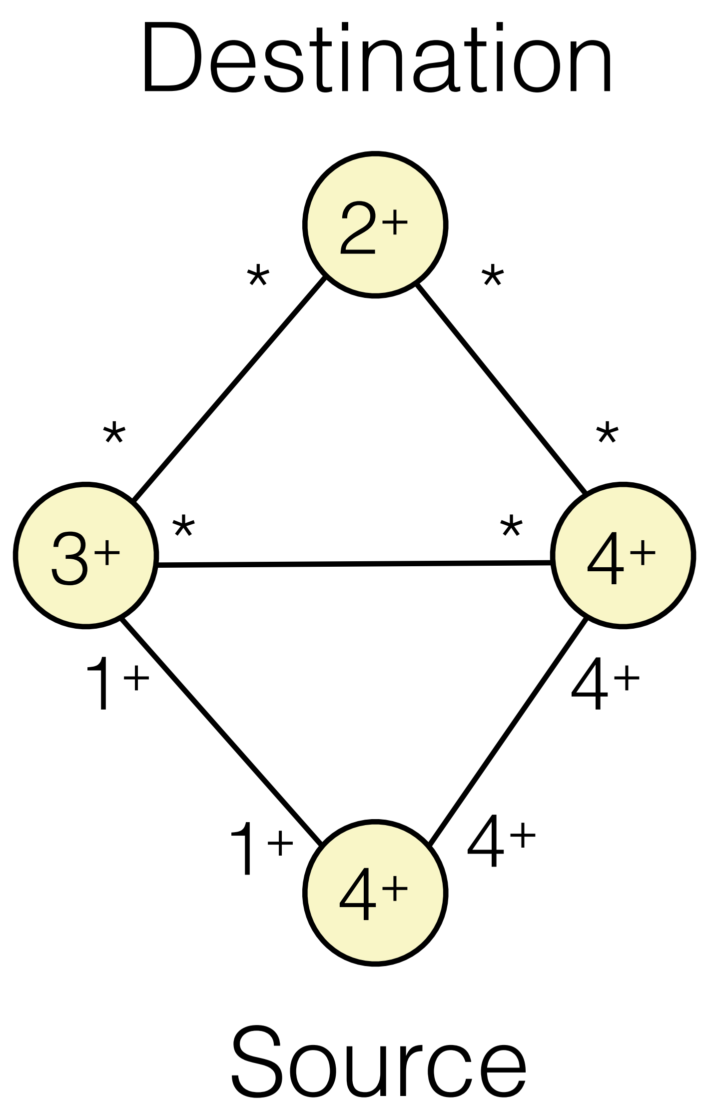
How many failures
to disconnect B?

**1!**

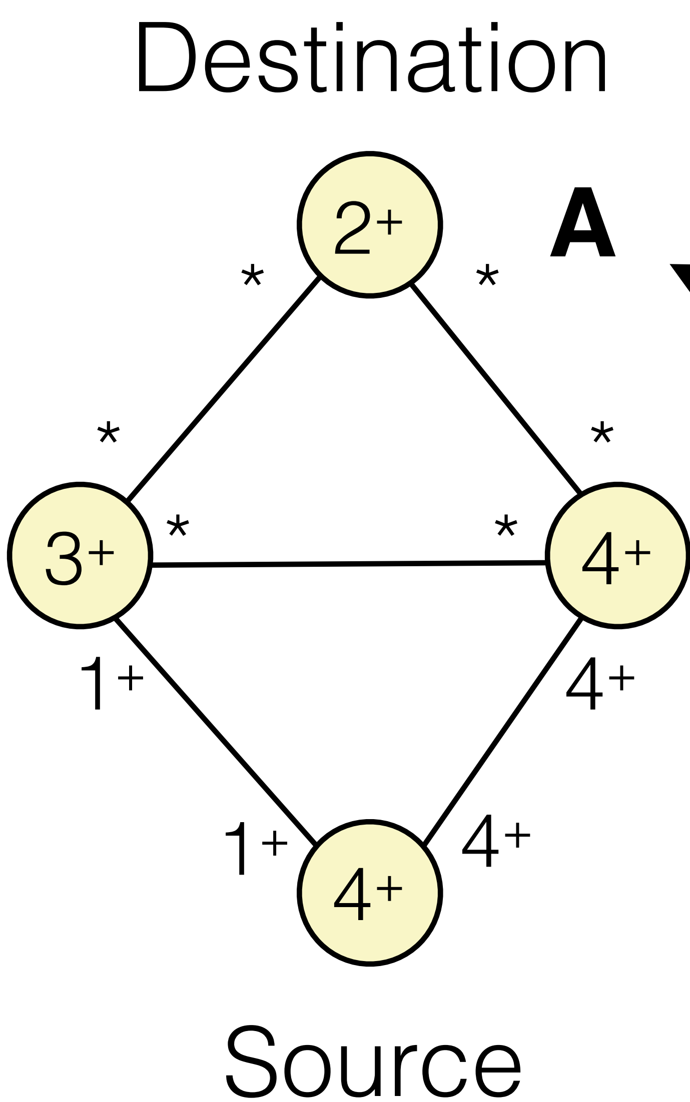


# Abstract Safety Analysis

# Reachability under k-failures

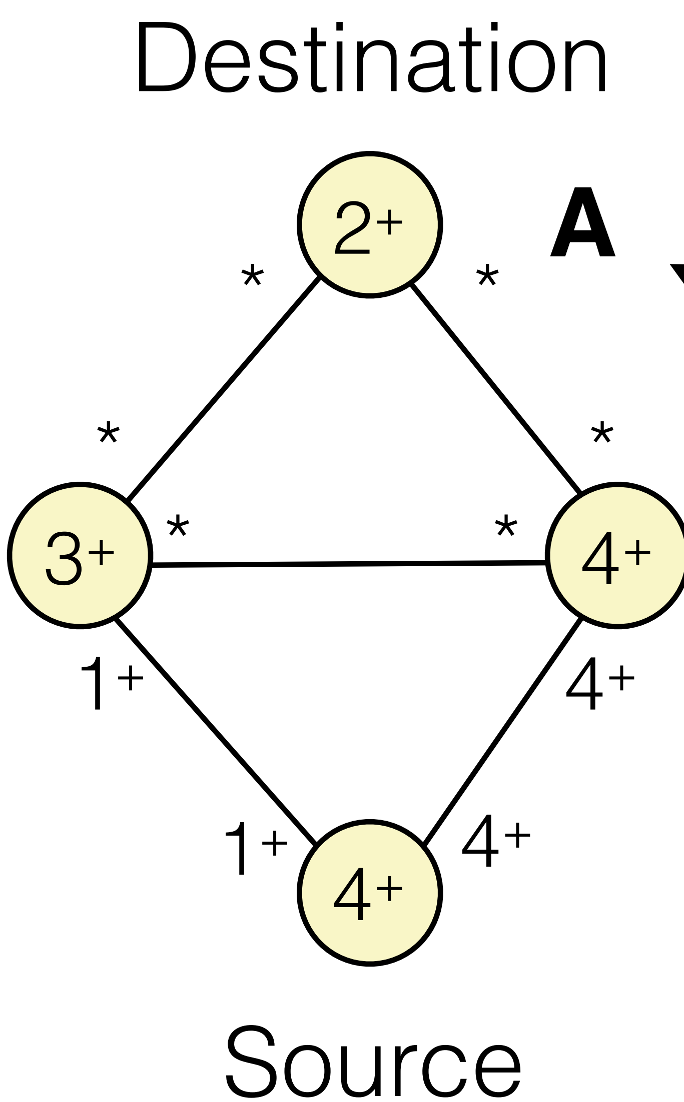


# Reachability under k-failures



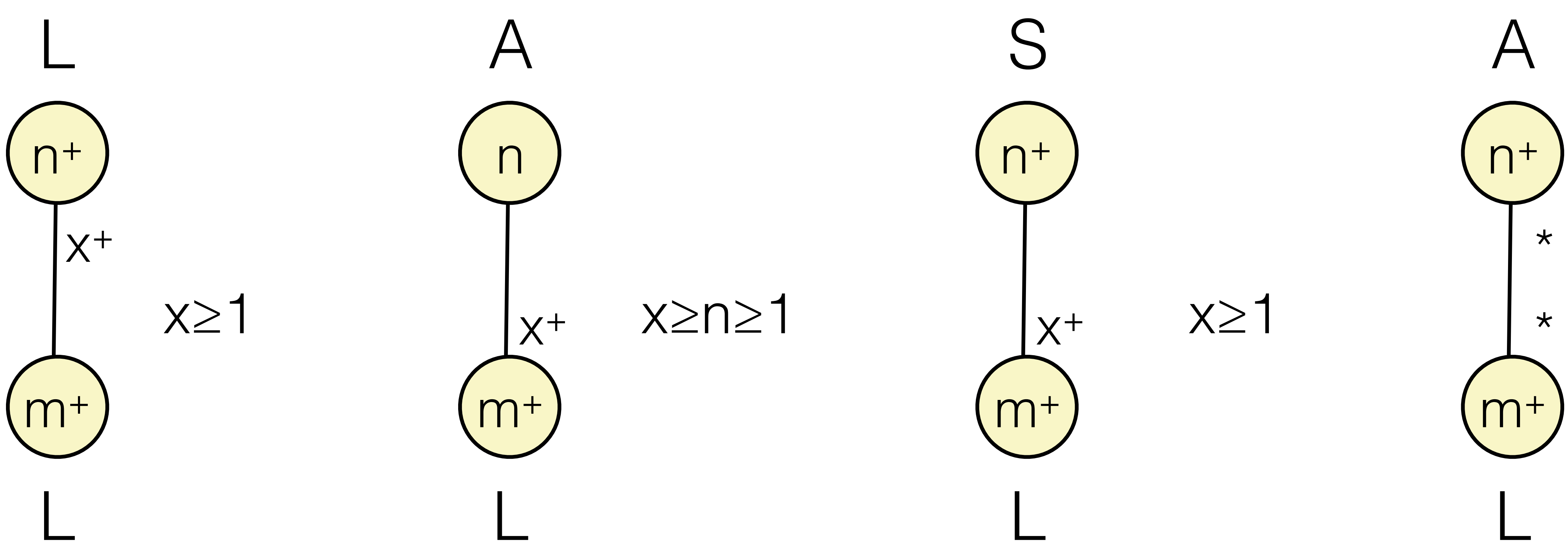

Abstract reachability tells us that all nodes are reachable

# Reachability under k-failures



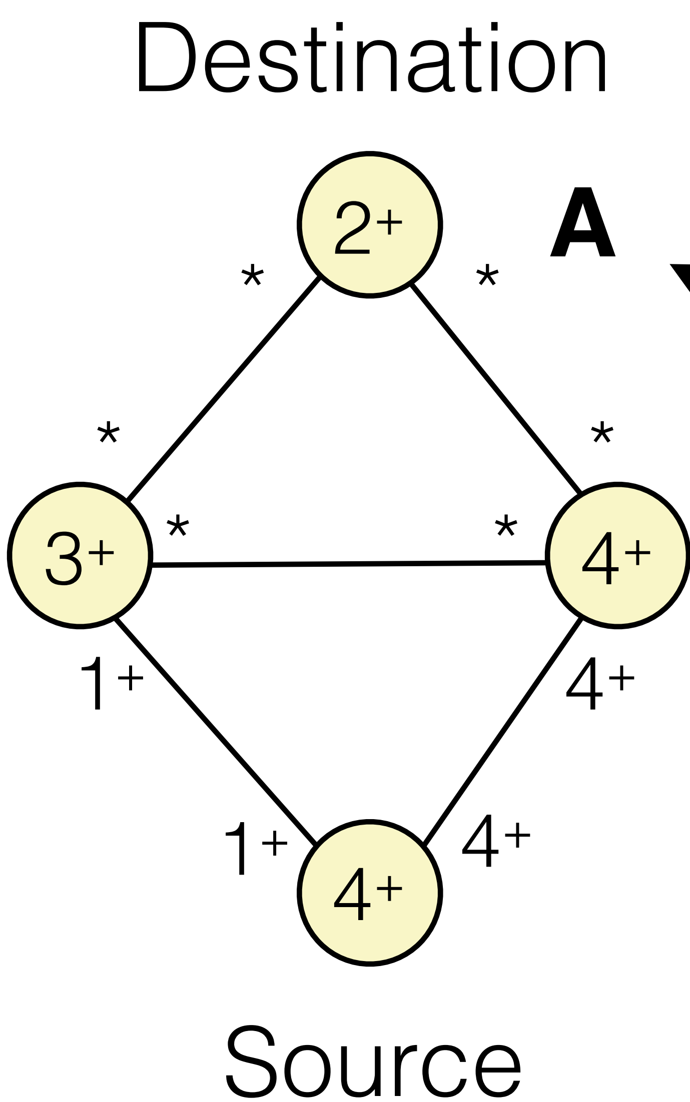

How many failures required to turn this **A** into an **N**

# Inference Rules:



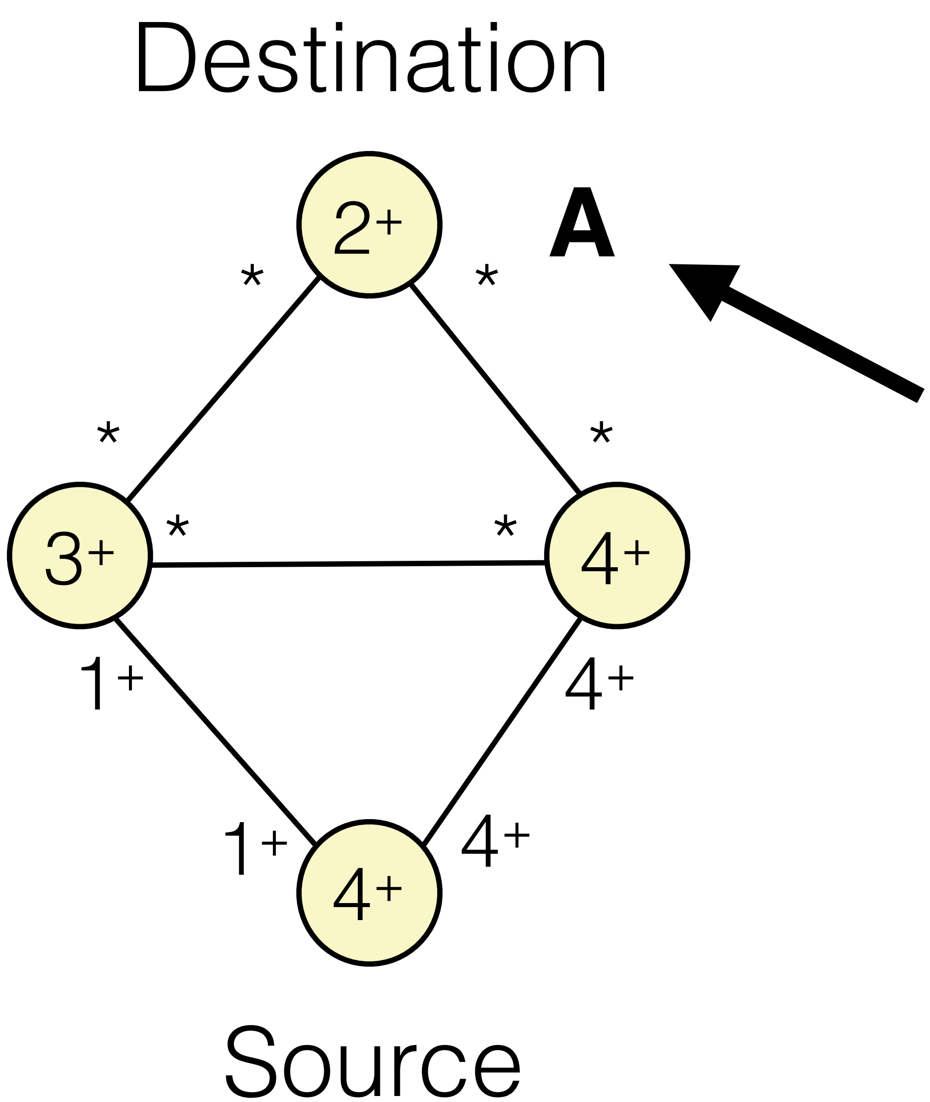

$L \in \{A, S\}$

# Reachability under k-failures




How many failures required to turn this **A** into an **N**

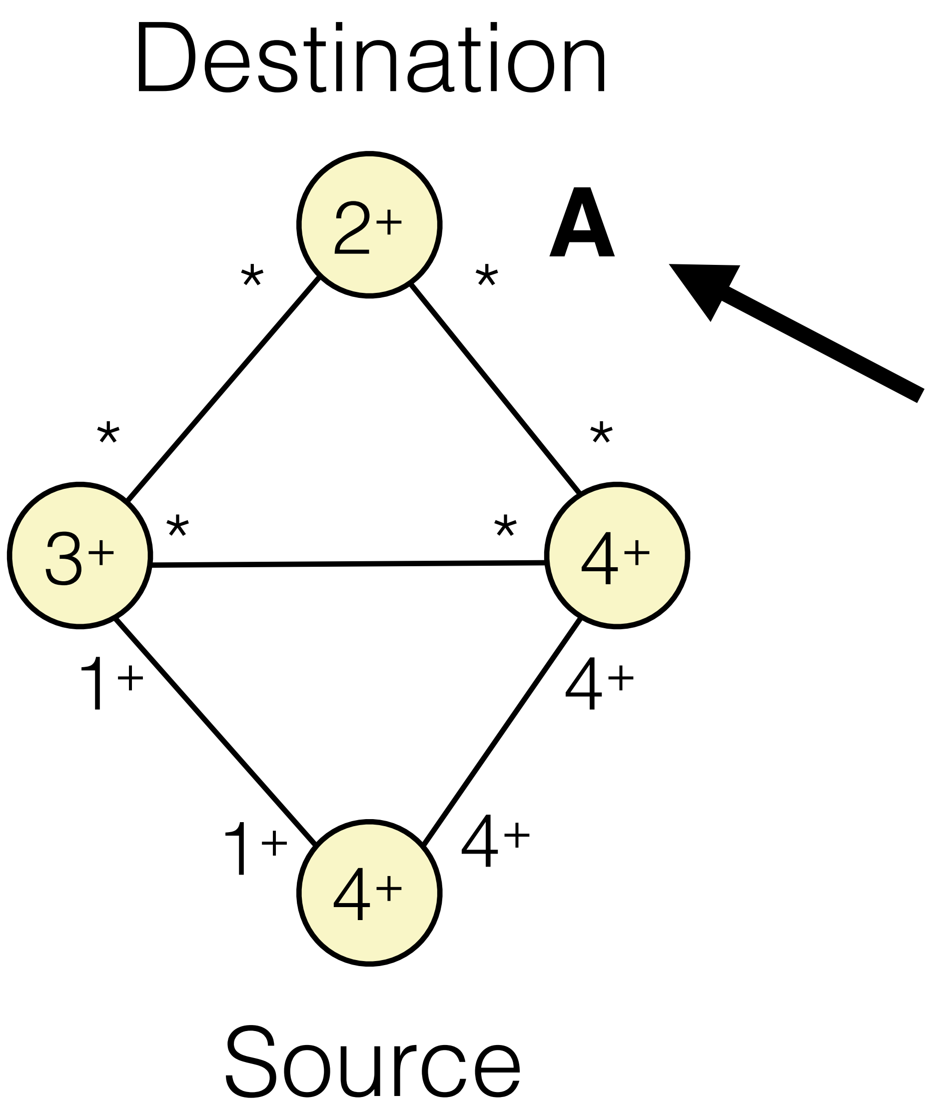
# Reachability under k-failures




How many failures required to turn this **A** into an **N**

In order to infer **N**, a single edge must result in **N**

# Reachability under k-failures

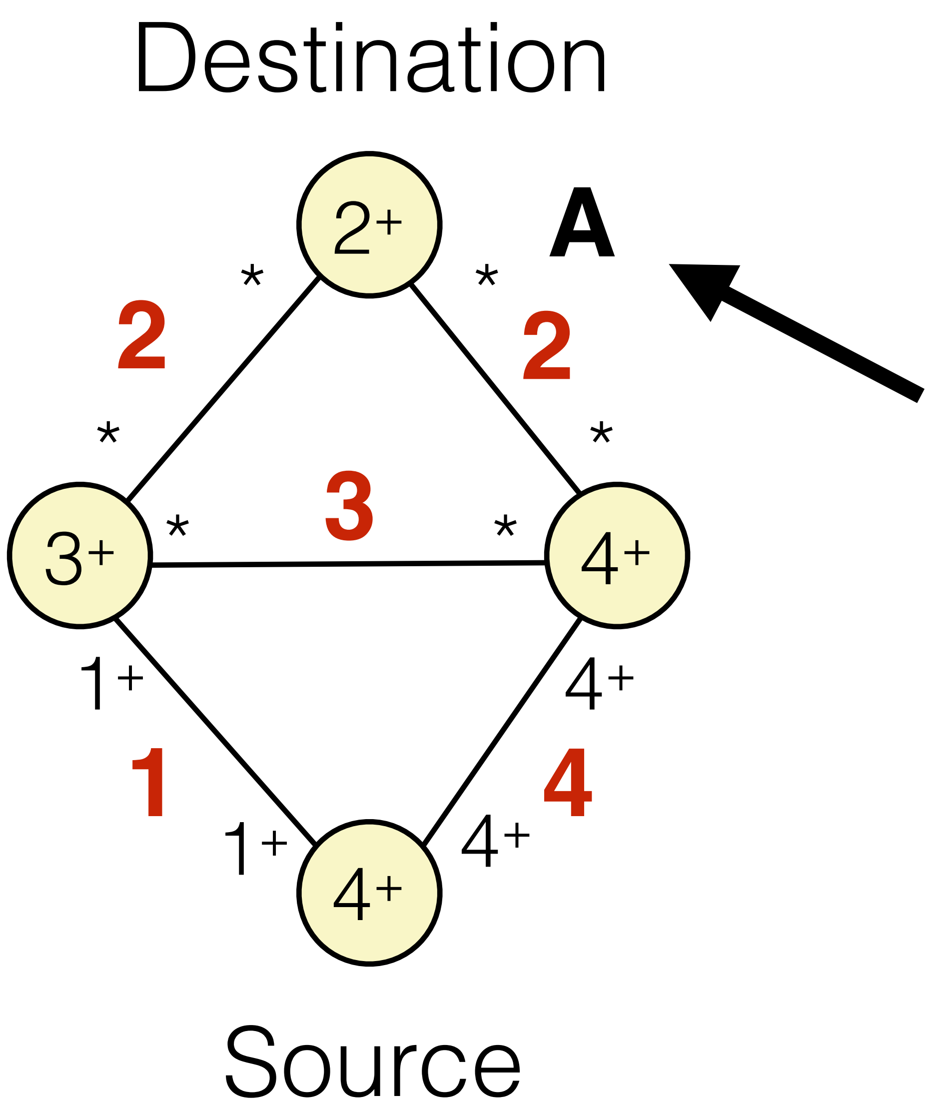



How many failures required to turn this **A** into an **N**

In order to infer **N**, a single edge must result in **N**

Edge-by-edge, how many failures change the inference

# Reachability under k-failures



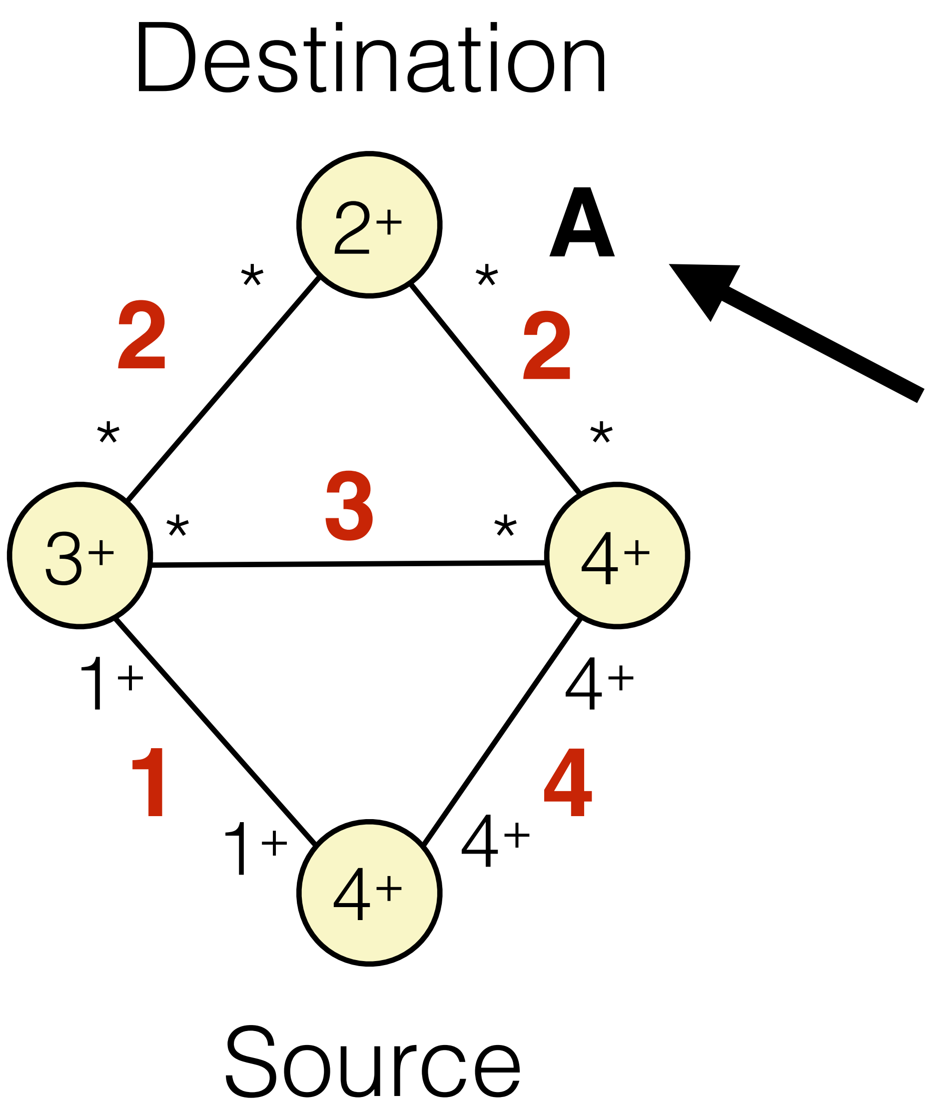

Conservatively assume only S is reachable for each node

How many failures required to turn this **A** into an **N**

In order to infer **N**, a single edge must result in **N**

Edge-by-edge, how many failures change the inference

# Reachability under k-failures

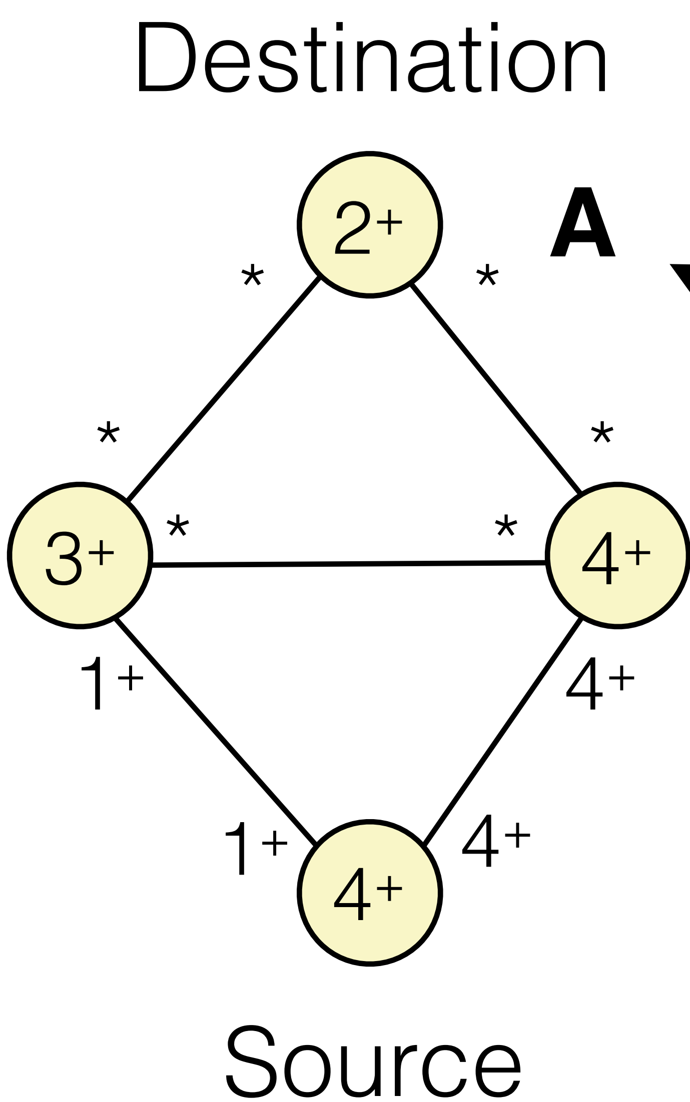



Conservatively assume only S is reachable for each node

How many failures required to turn this **A** into an **N**

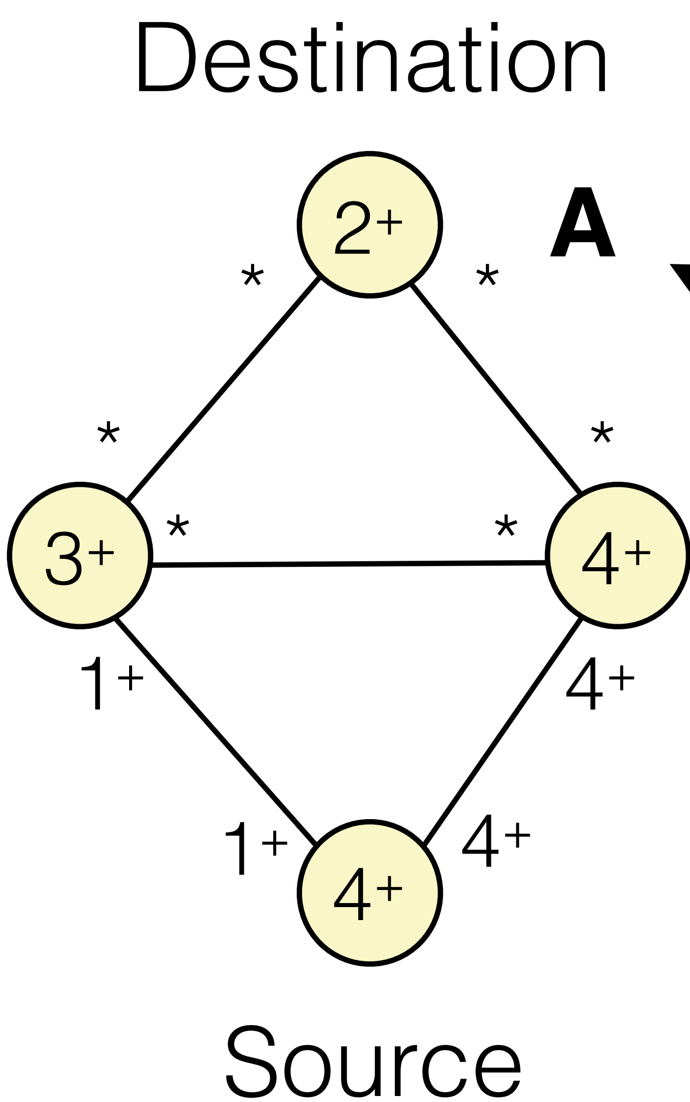
Min-cut = **4**

# Reachability under k-failures




How many failures required to turn this **A** into an **S**

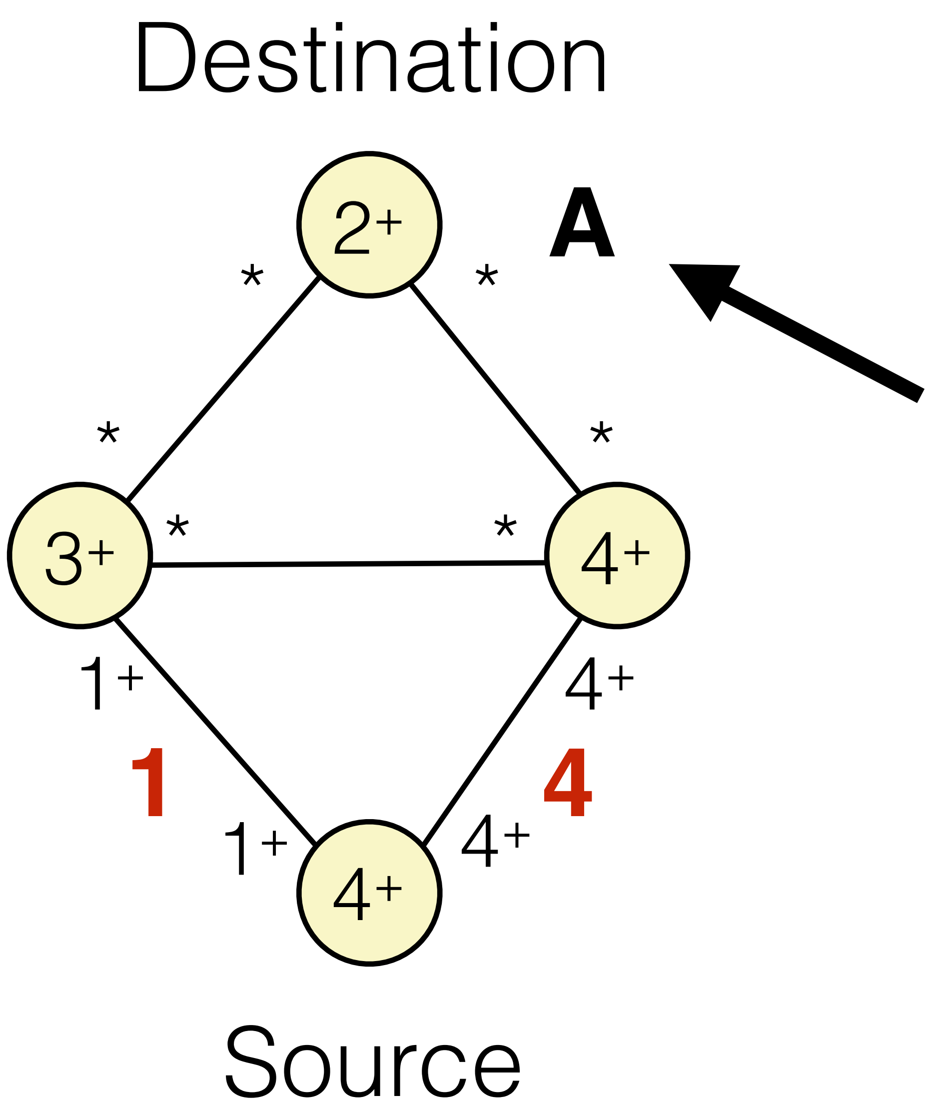
# Reachability under k-failures




How many failures required to turn this **A** into an **S**

More challenging problem

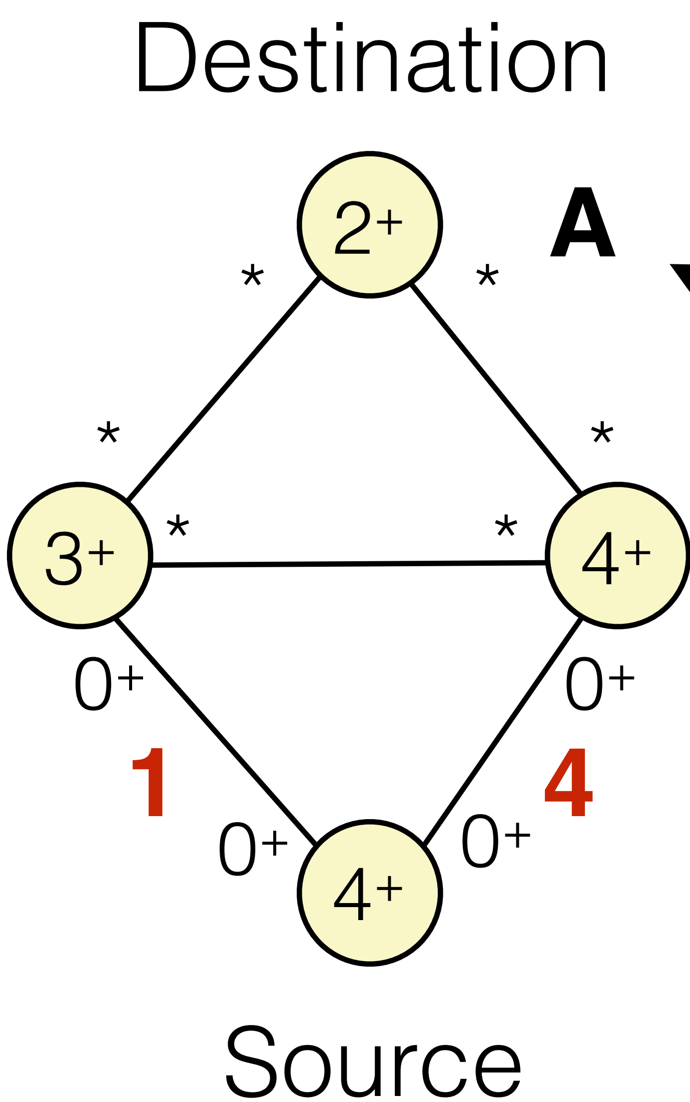
# Reachability under k-failures




How many failures required to turn this **A** into an **S**

More challenging problem

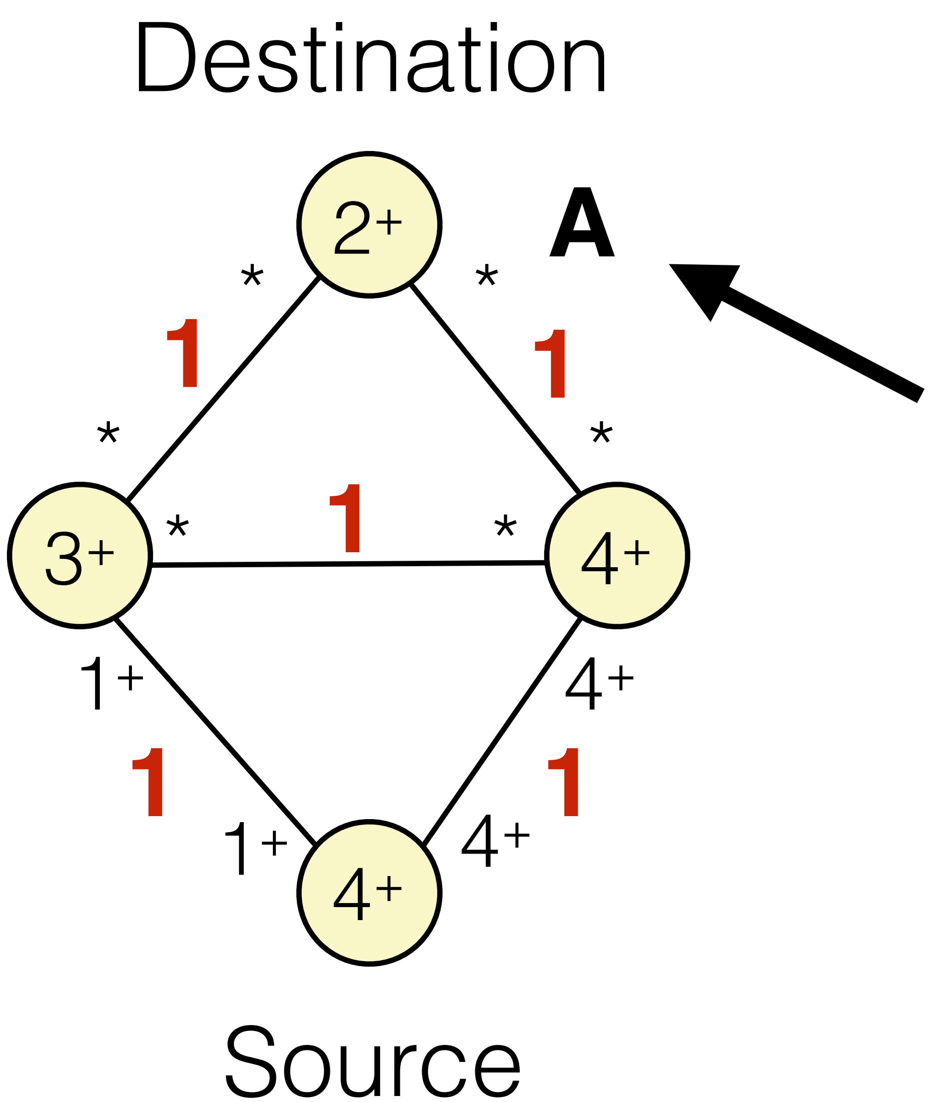
# Reachability under k-failures




How many failures required to turn this **A** into an **S**

More challenging problem

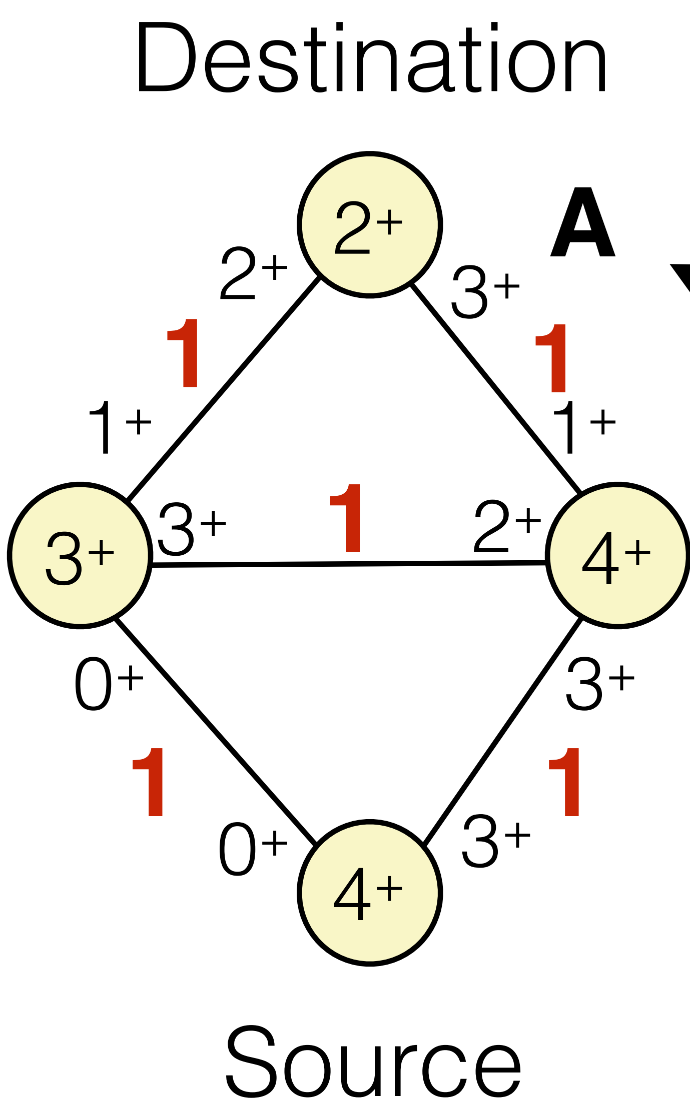
# Reachability under k-failures




How many failures required to turn this **A** into an **S**

More challenging problem

# Reachability under k-failures

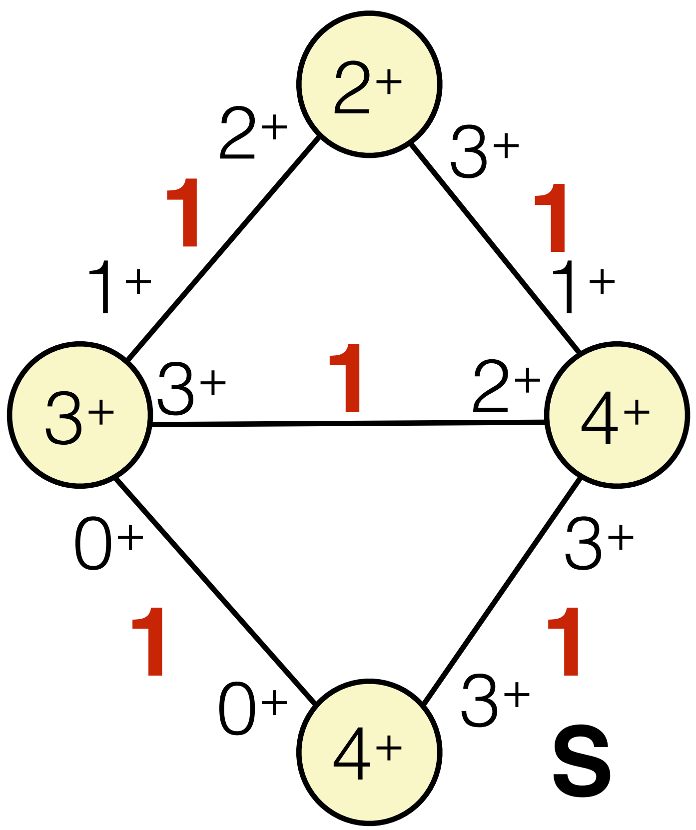



How many failures required to turn this **A** into an **S**

More challenging problem

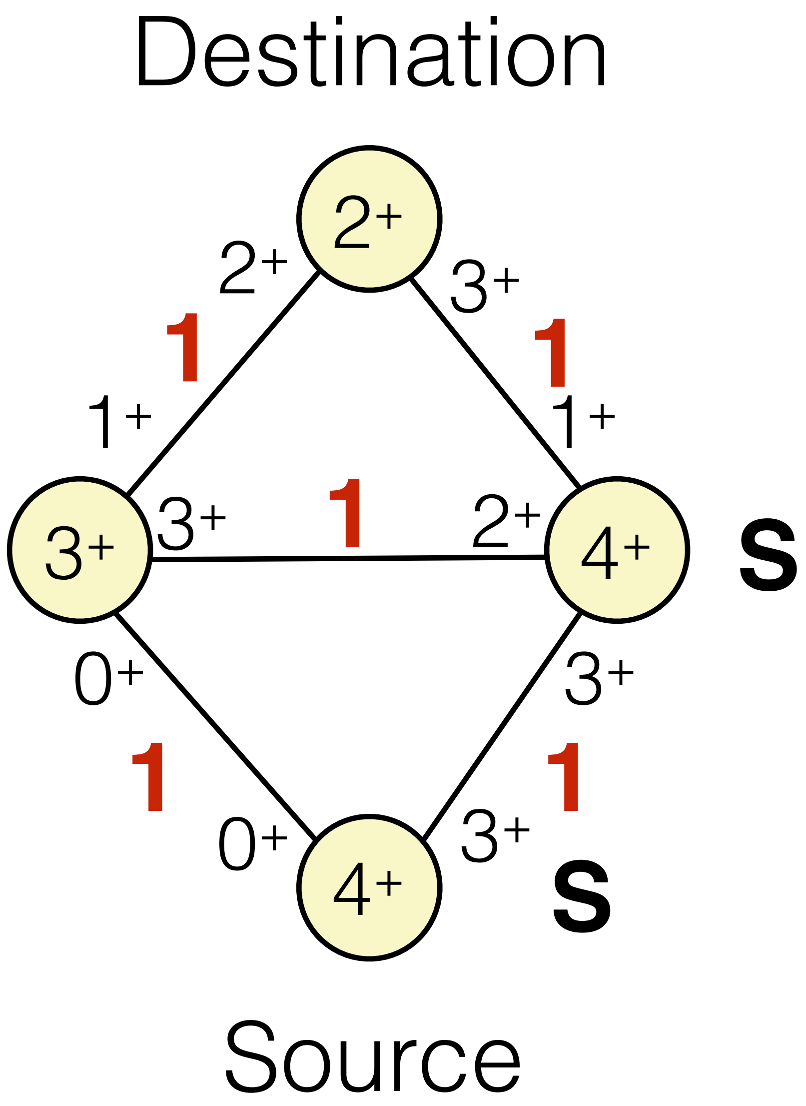
# Reachability under k-failures

Destination

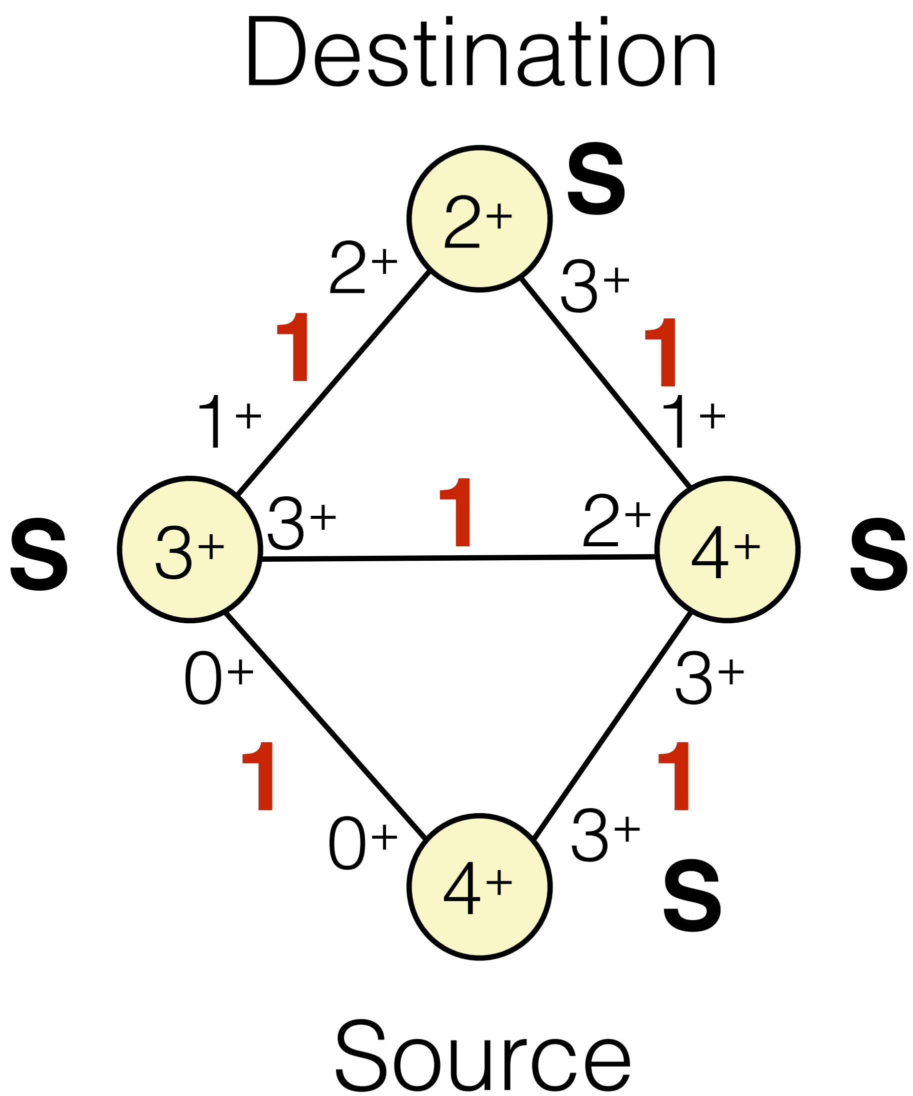



Source

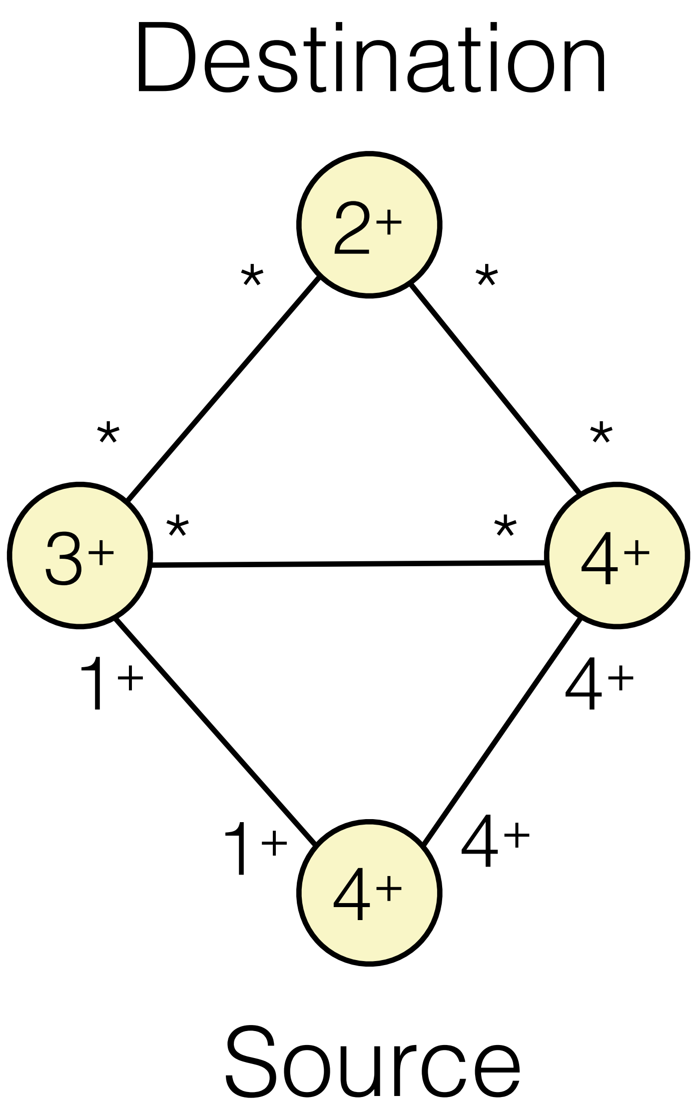
# Reachability under k-failures



# Reachability under k-failures

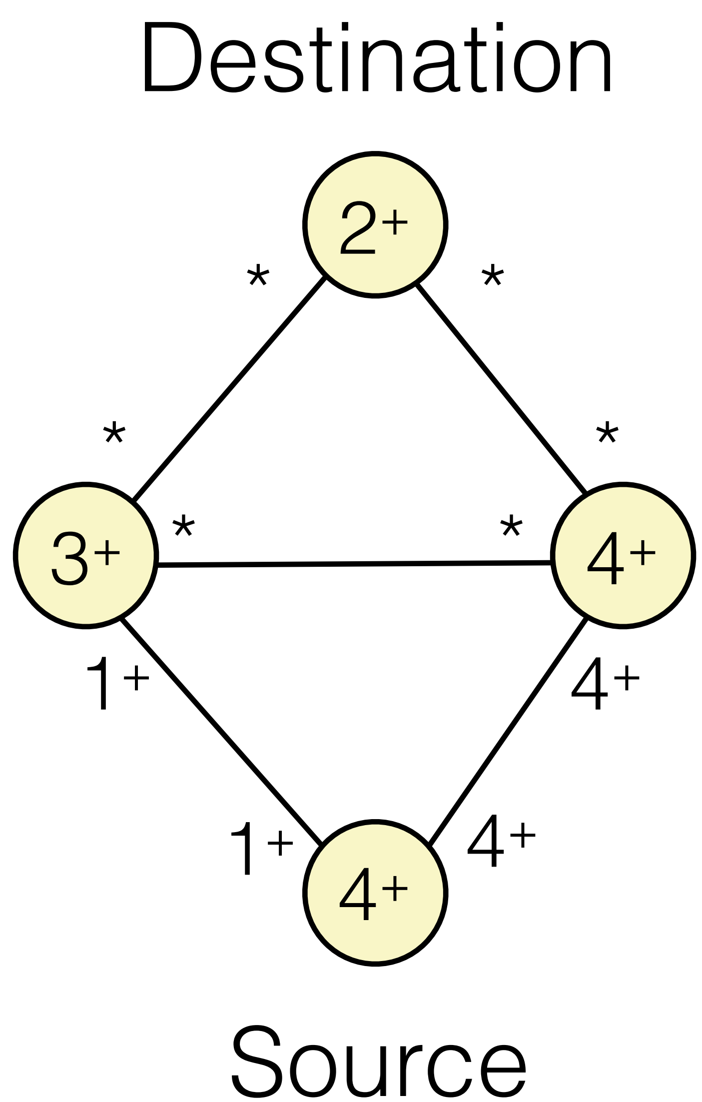


# Reachability under k-failures




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

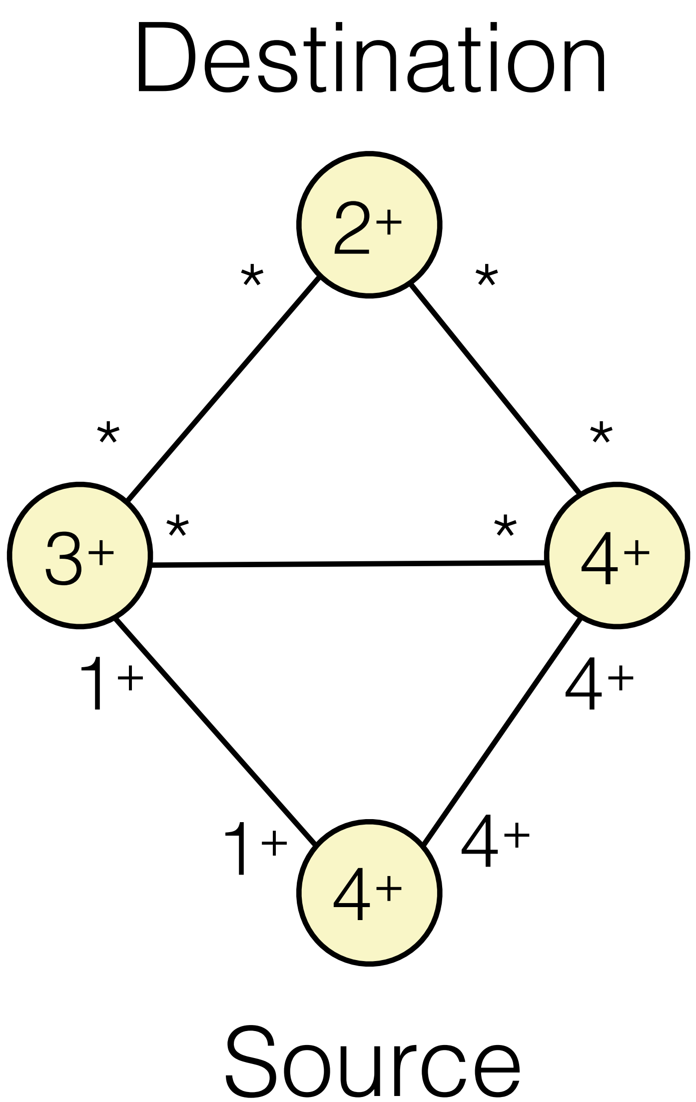
# Reachability under k-failures




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

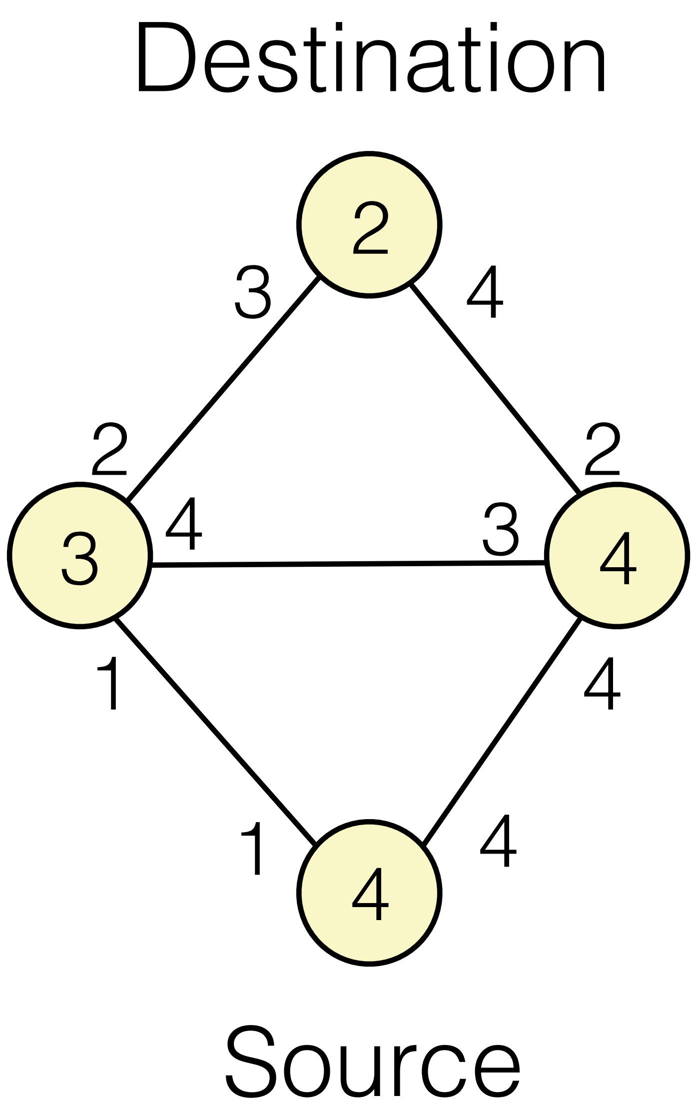
Keep the topology abstract (small)

# Reachability under k-failures




**Claim:** Take the min of every edge/node, and this will be the worst topology

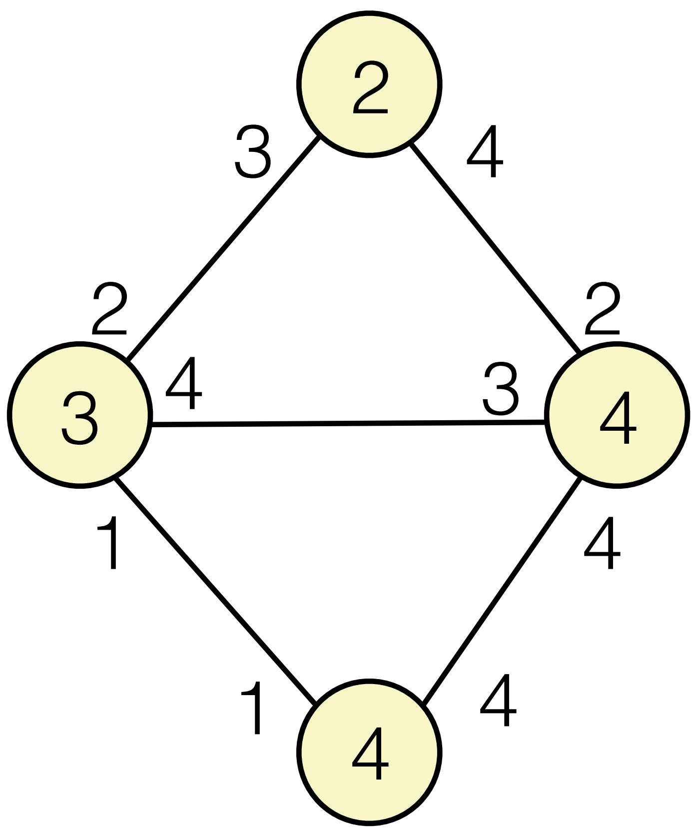
# Reachability under k-failures




**Claim:** Take the min of every edge/node, and this will be the worst topology

# Reachability under k-failures

Destination

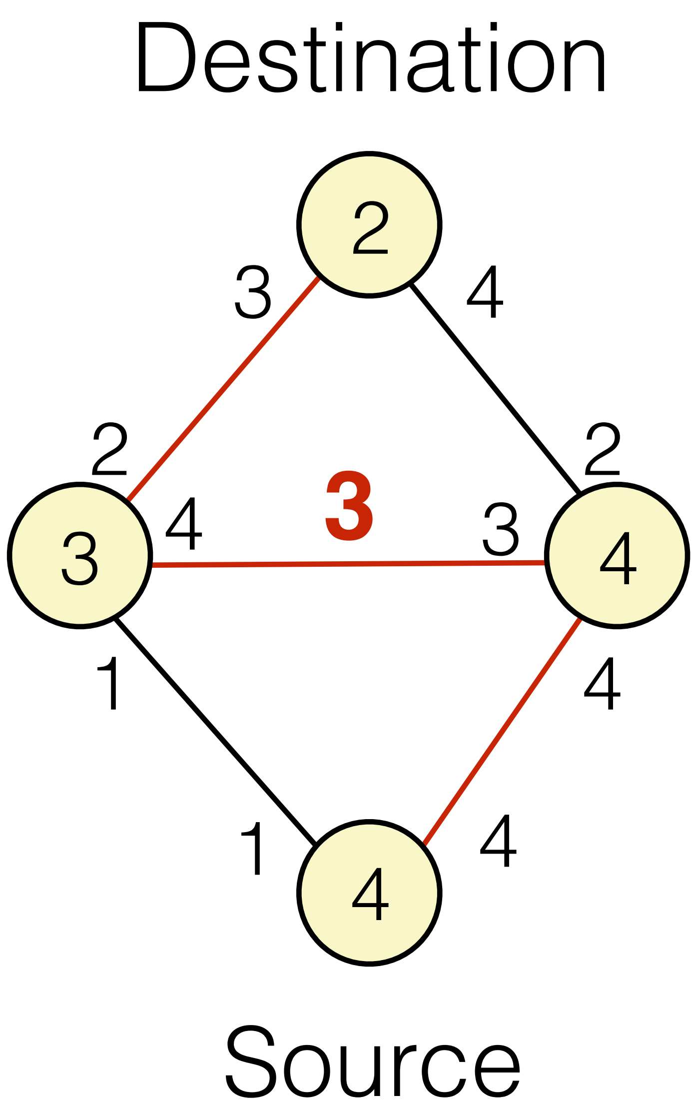



Source

**Claim:** Take the min of every edge/node, and this will be the worst topology

**Why:** Can always fail more nodes/edges to get this topology

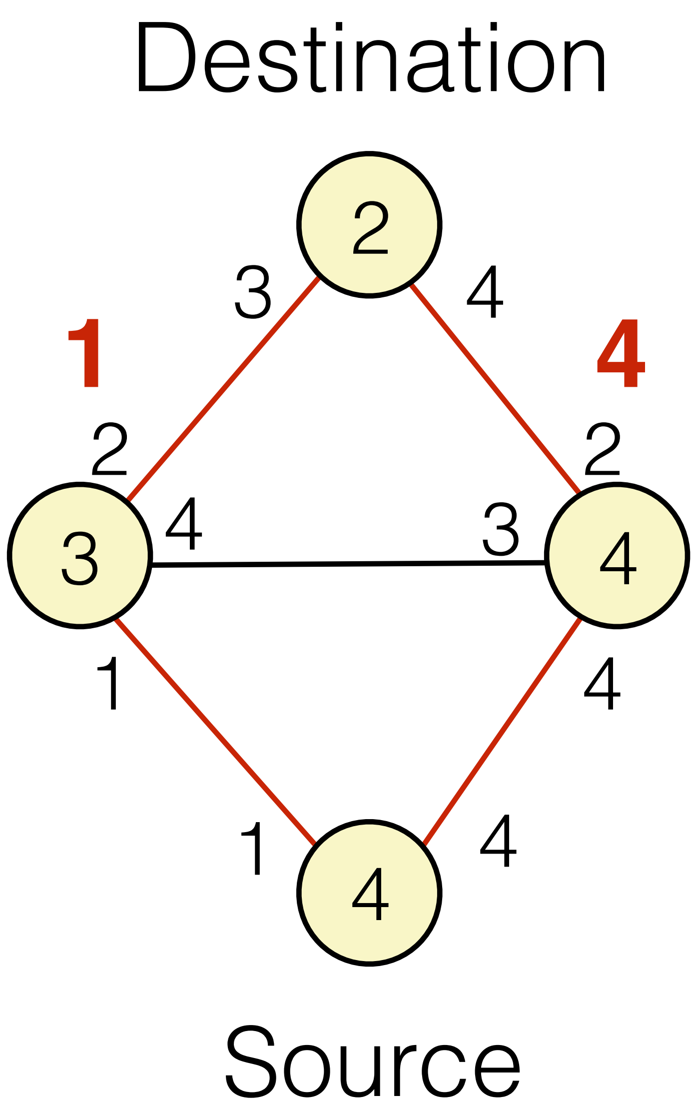
# Reachability under k-failures




**Claim:** Take the min of every edge/node, and this will be the worst topology

**Idea:** Find a lower bound on the number of disjoint paths in the concrete topology.

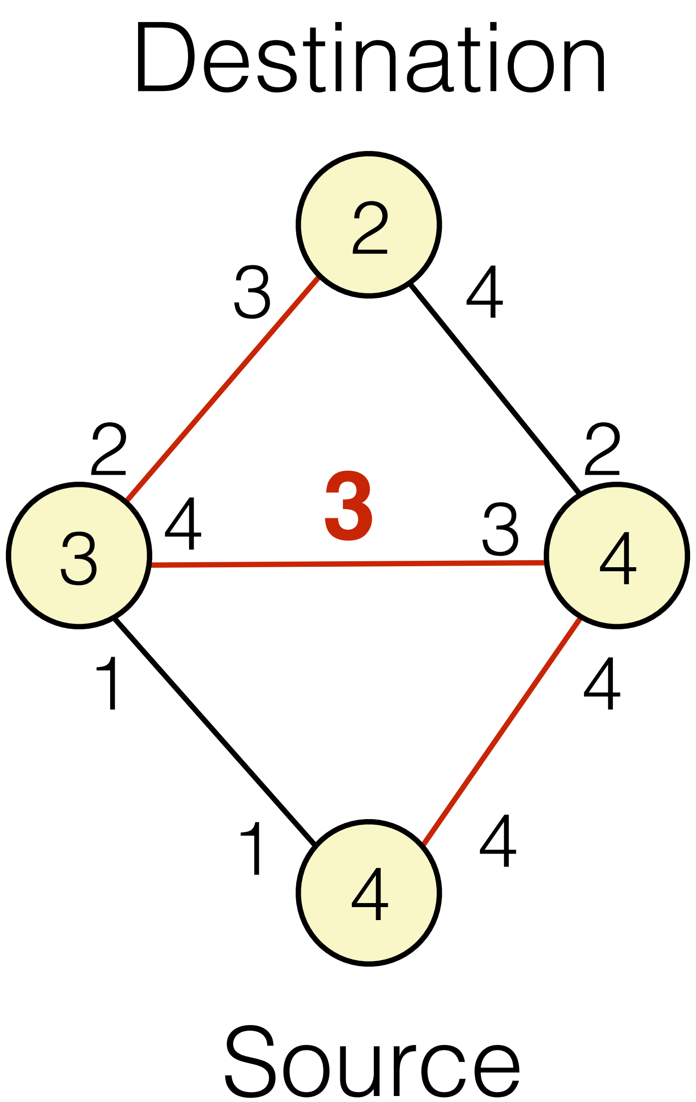
# Reachability under k-failures

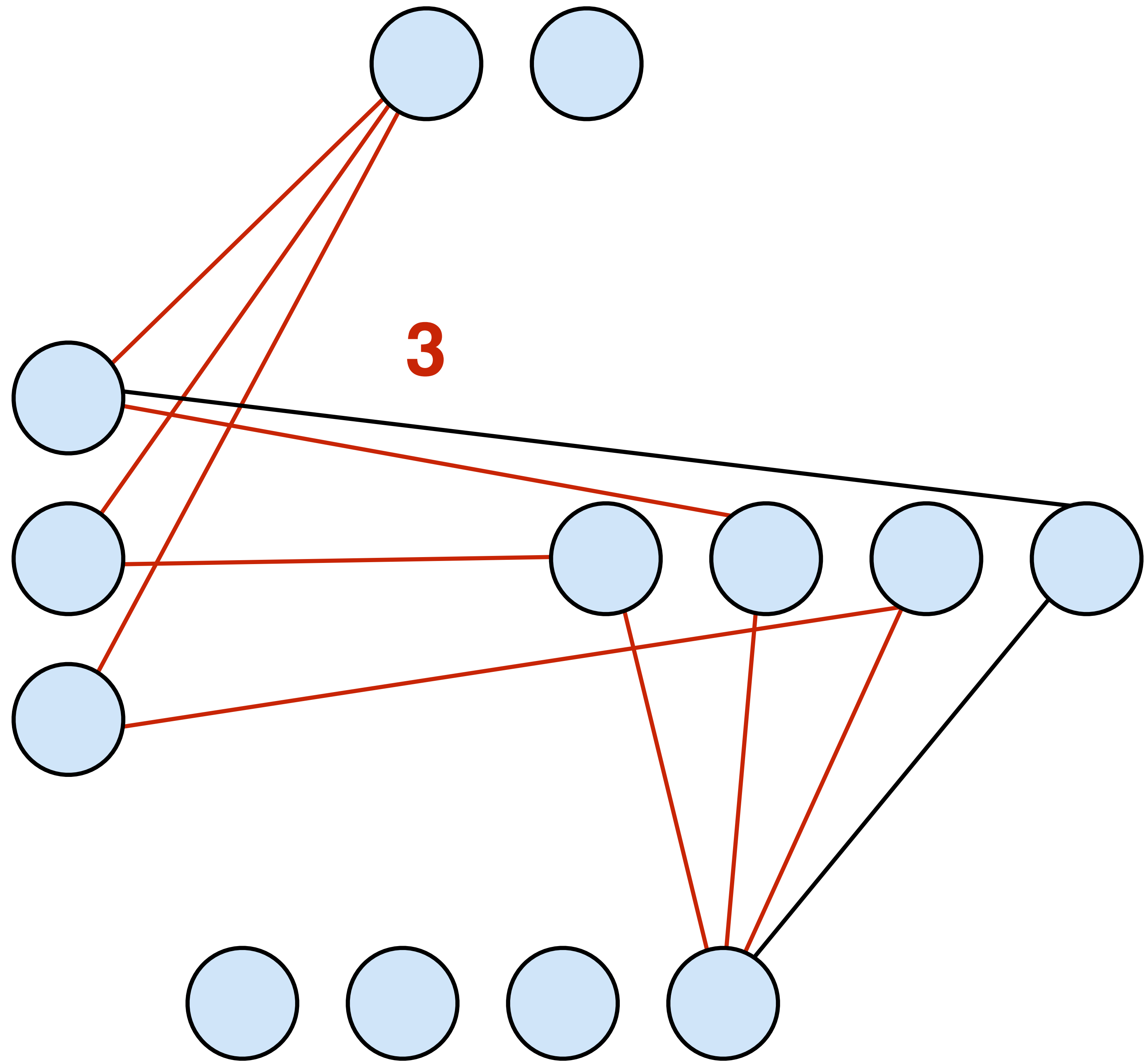



**Claim:** Take the min of every edge/node, and this will be the worst topology

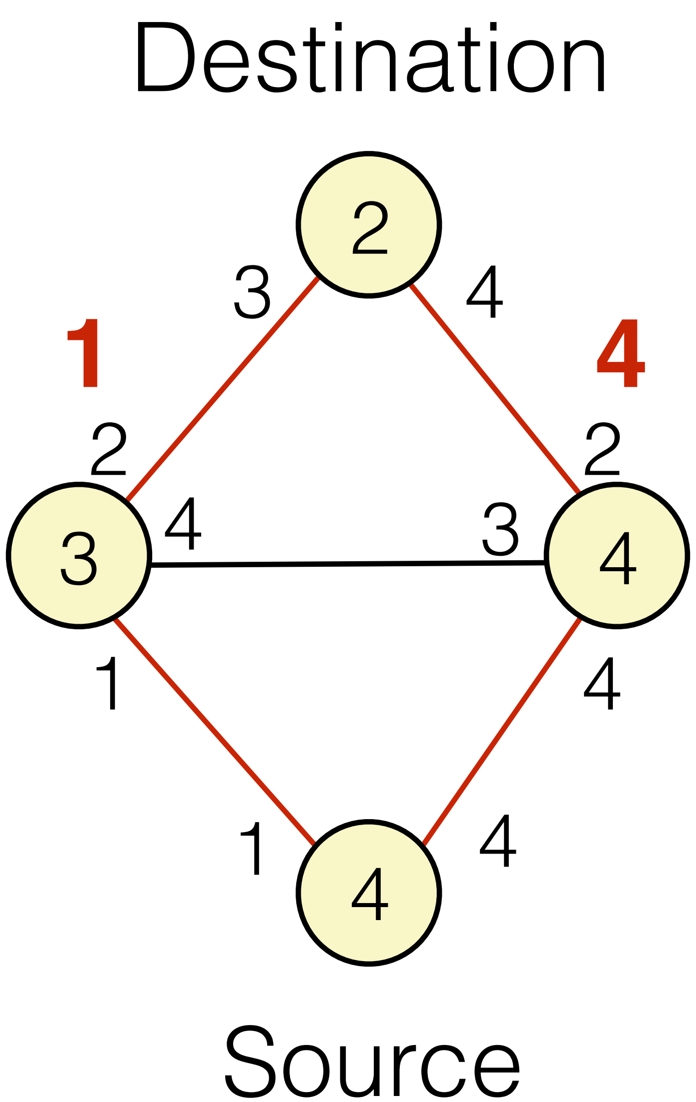
**Idea:** Find a lower bound on the number of disjoint paths in the concrete topology.

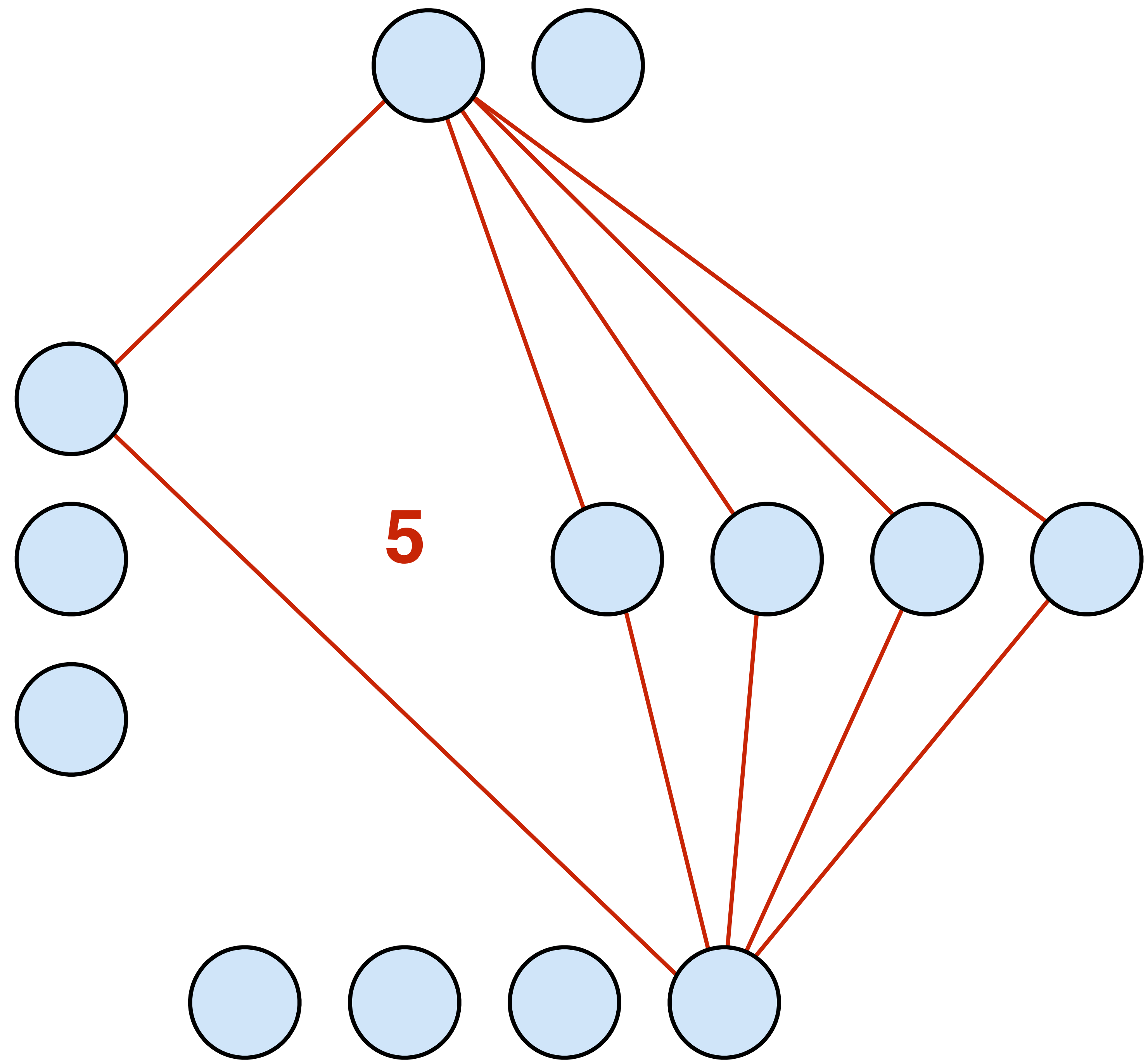
# Reachability under k-failures



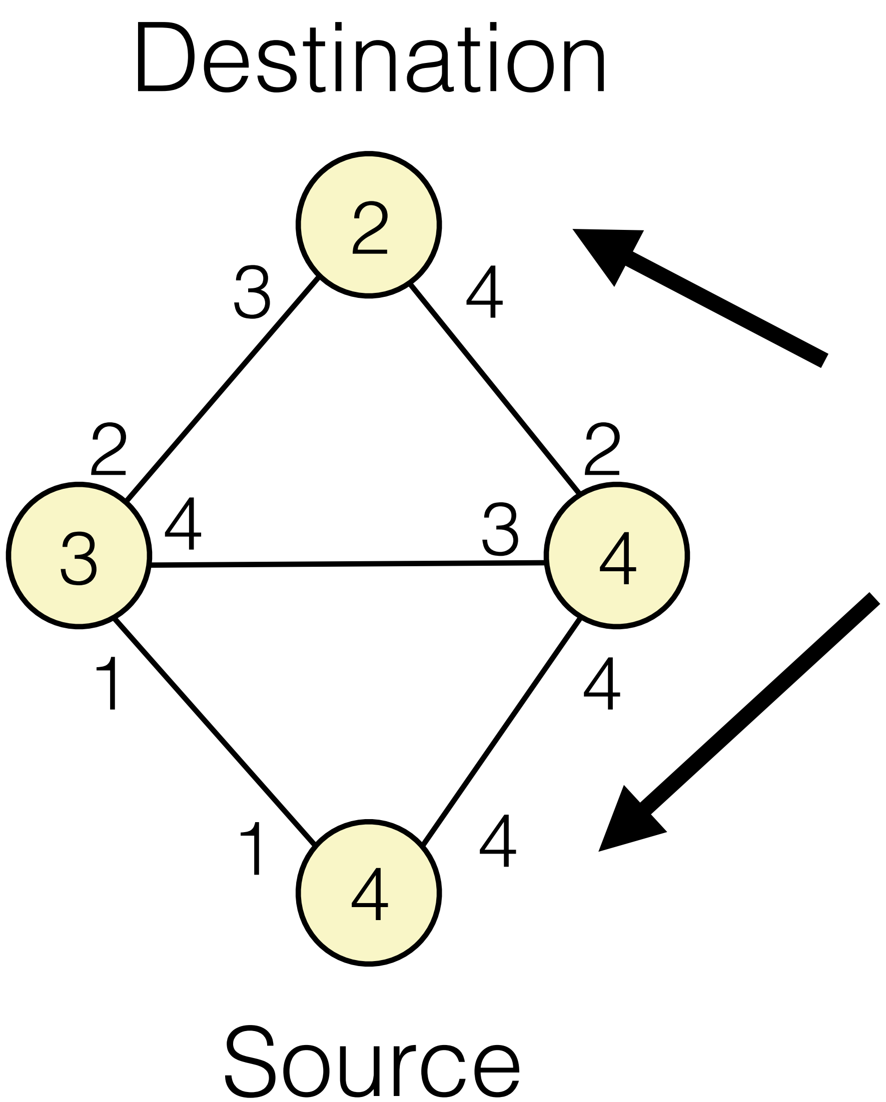



# Reachability under k-failures



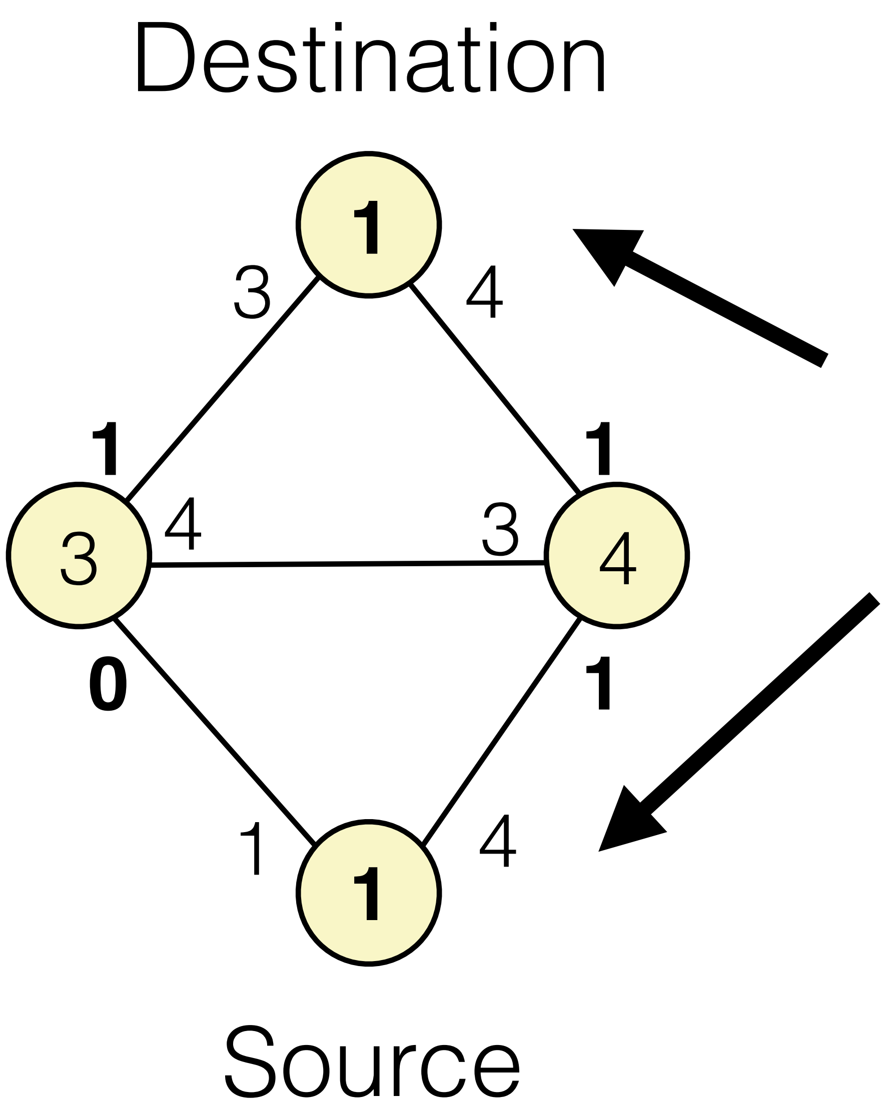



# Reachability under k-failures



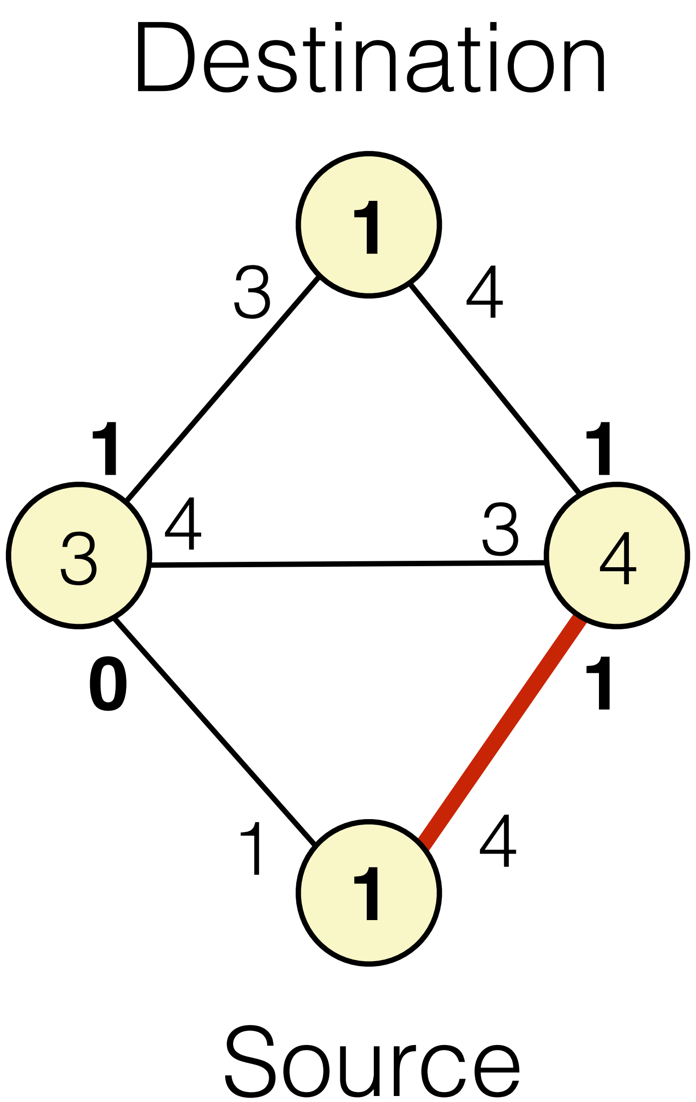

We care about disconnecting some arbitrary node at the top from some arbitrary node at the bottom

# Reachability under k-failures



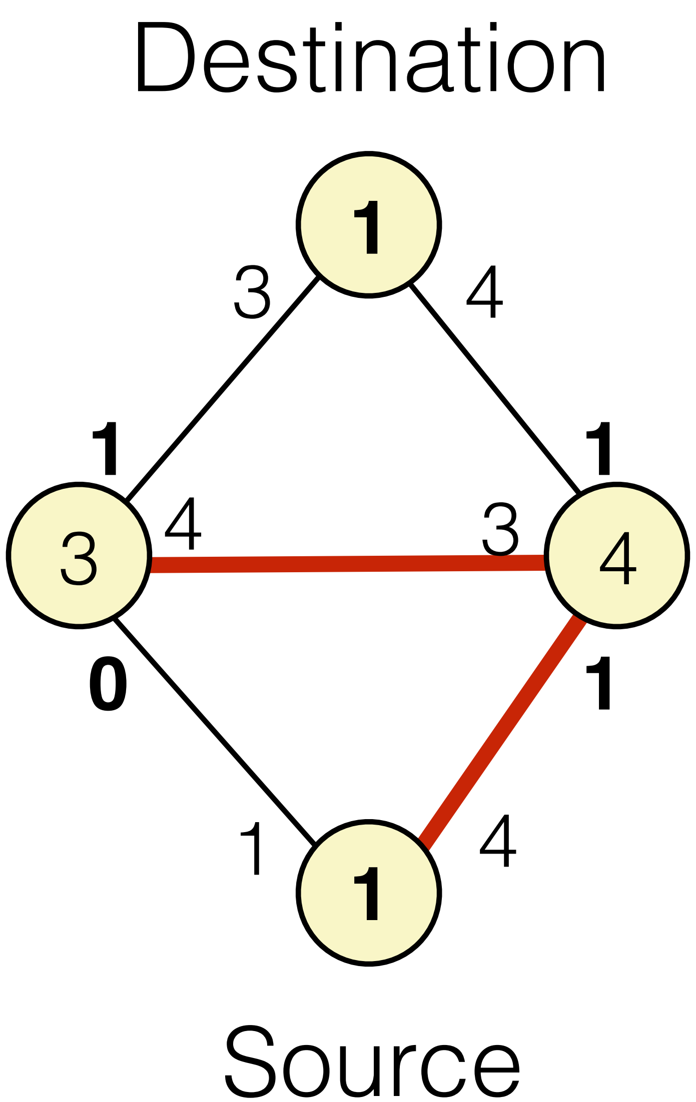

We care about disconnecting some arbitrary node at the top from some arbitrary node at the bottom

# Reachability under k-failures




4 disjoint edges to 4 disjoint nodes

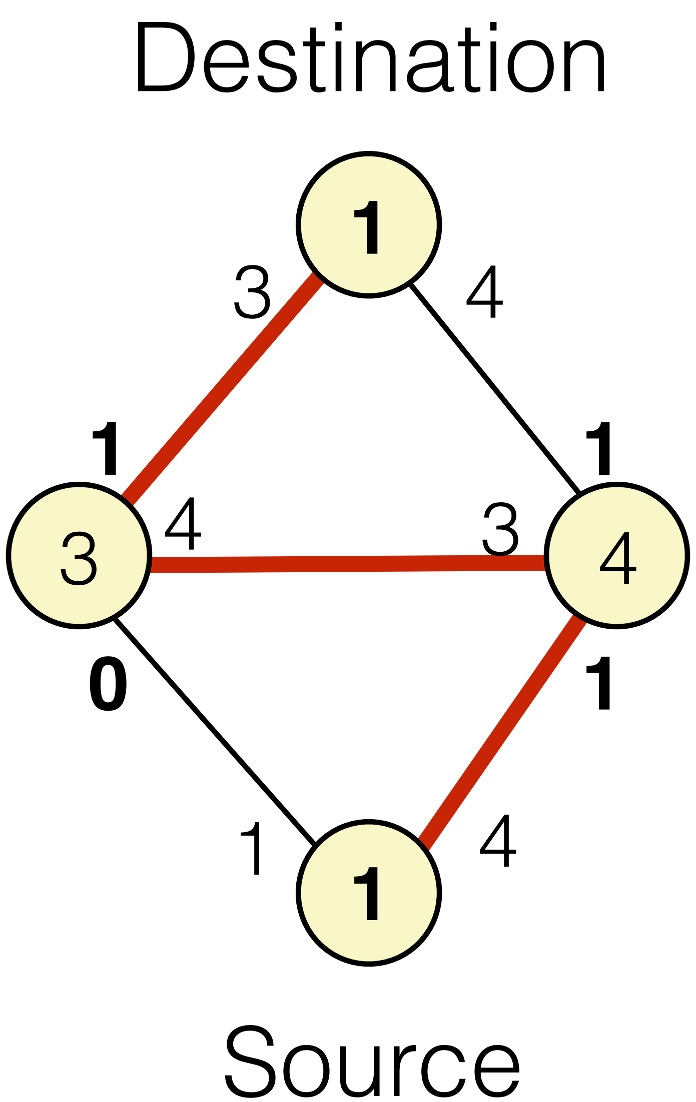
# Reachability under k-failures




4 disjoint edges to 3 disjoint nodes

4 disjoint edges to 4 disjoint nodes

# Reachability under k-failures

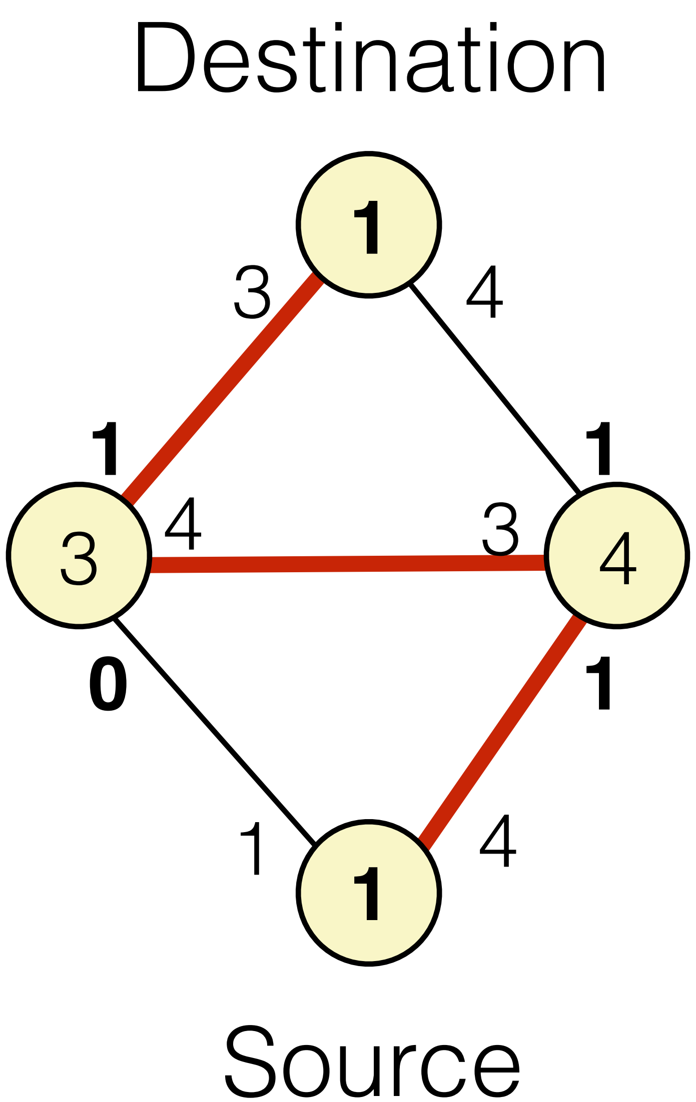



3 disjoint edges to 1 node

4 disjoint edges to 3 disjoint nodes

4 disjoint edges to 4 disjoint nodes

# Reachability under k-failures




3 disjoint edges to 1 node

4 disjoint edges to 3 disjoint nodes

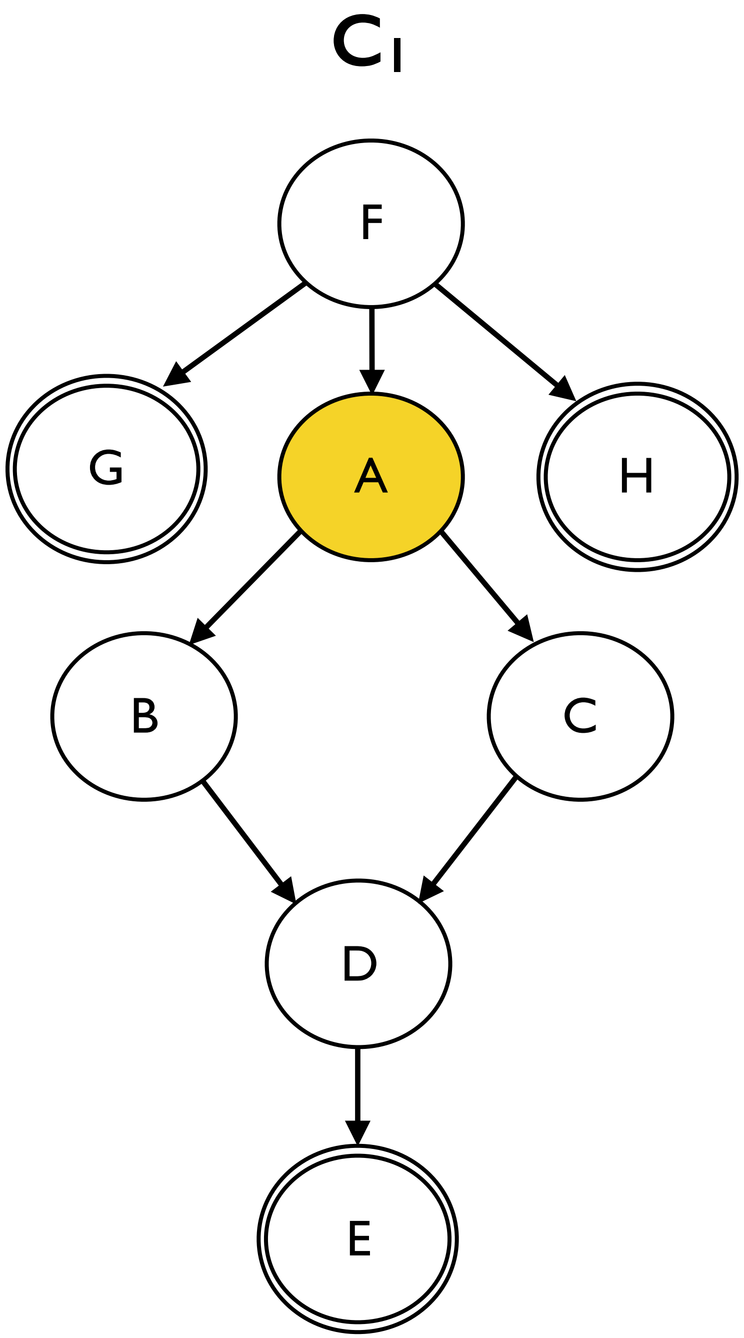
4 disjoint edges to 4 disjoint nodes

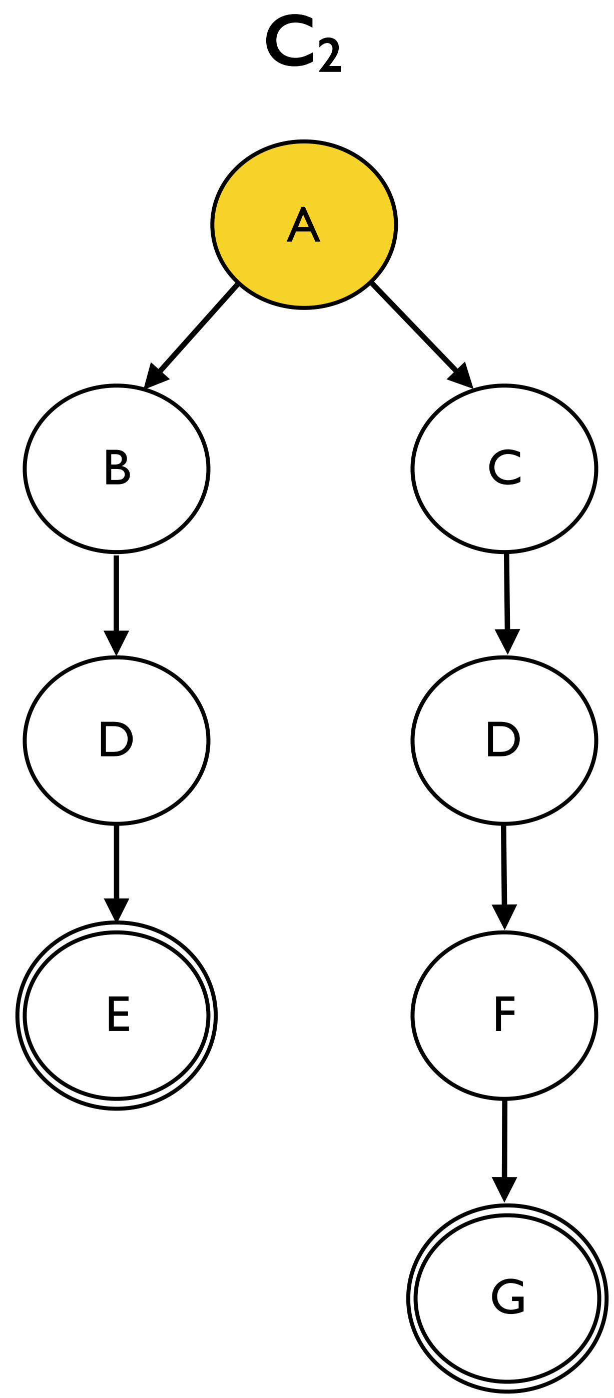
---

min{3,4,4} = **3**

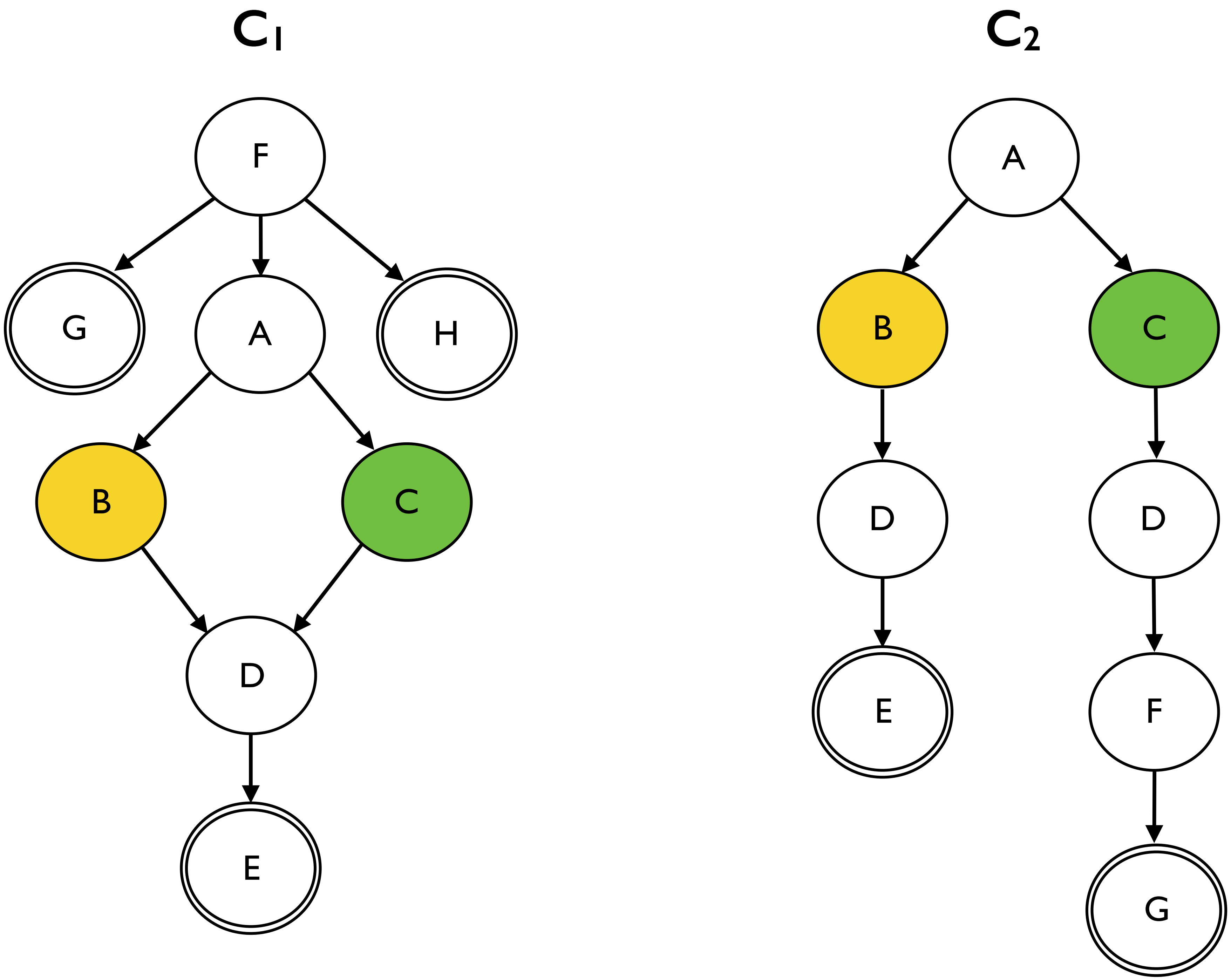
# Compilation Correctness

# Failure Safety (Recap)



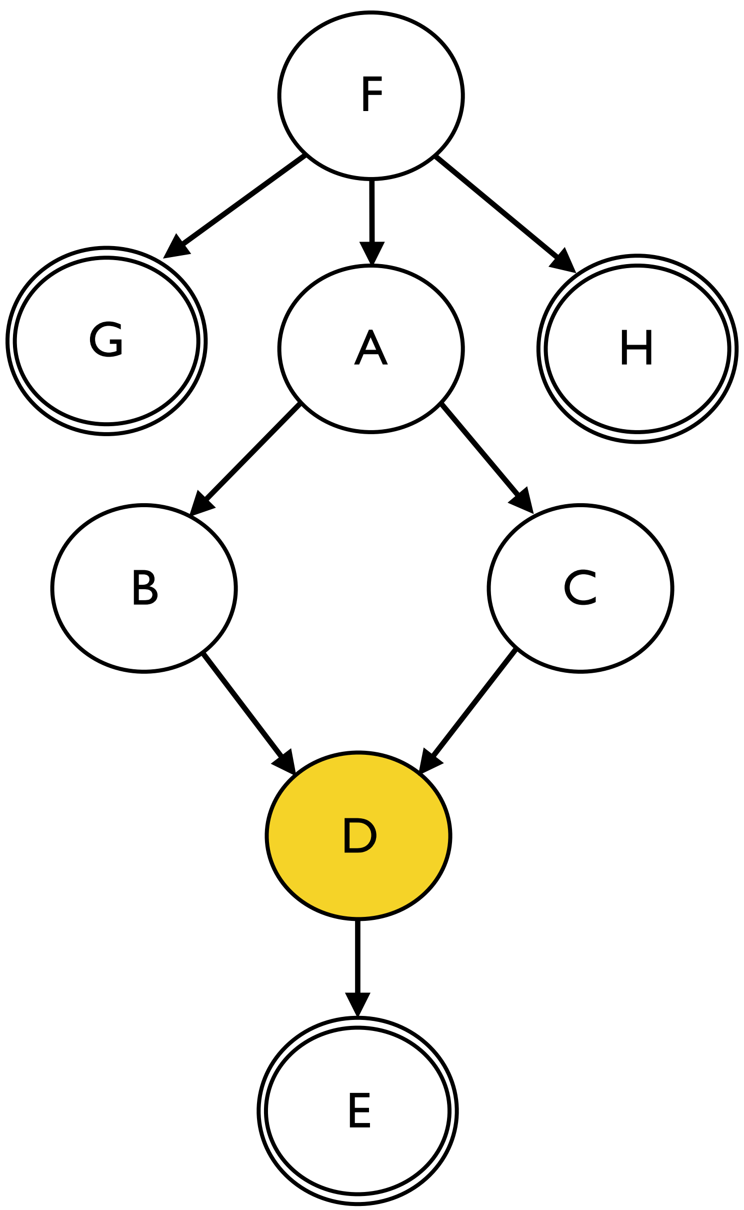



# Failure Safety (Recap)

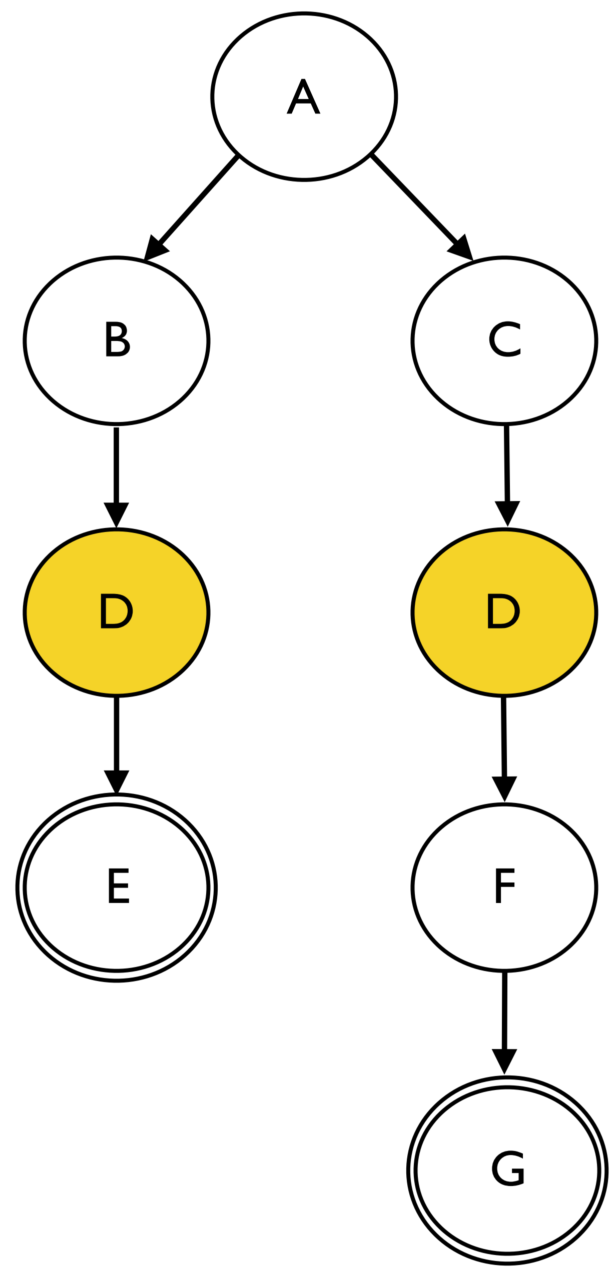


# Failure Safety (Recap)

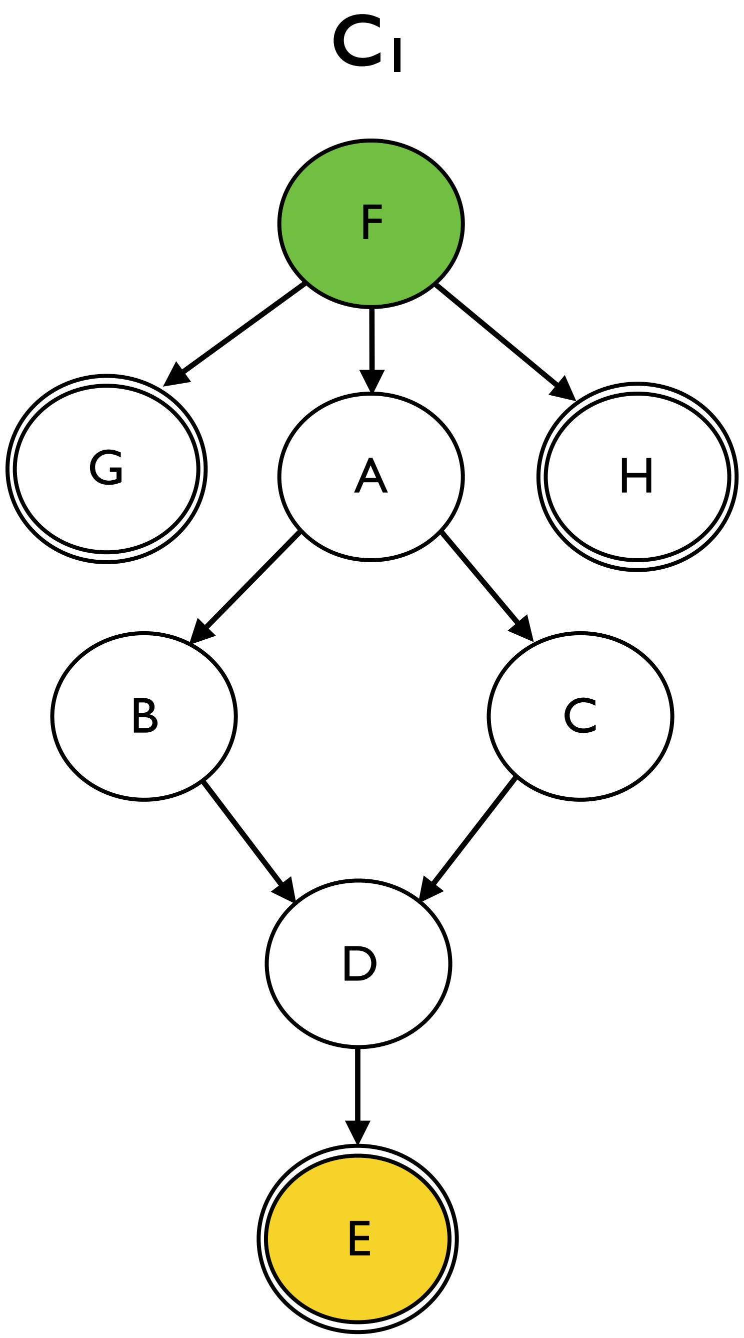
$C_1$

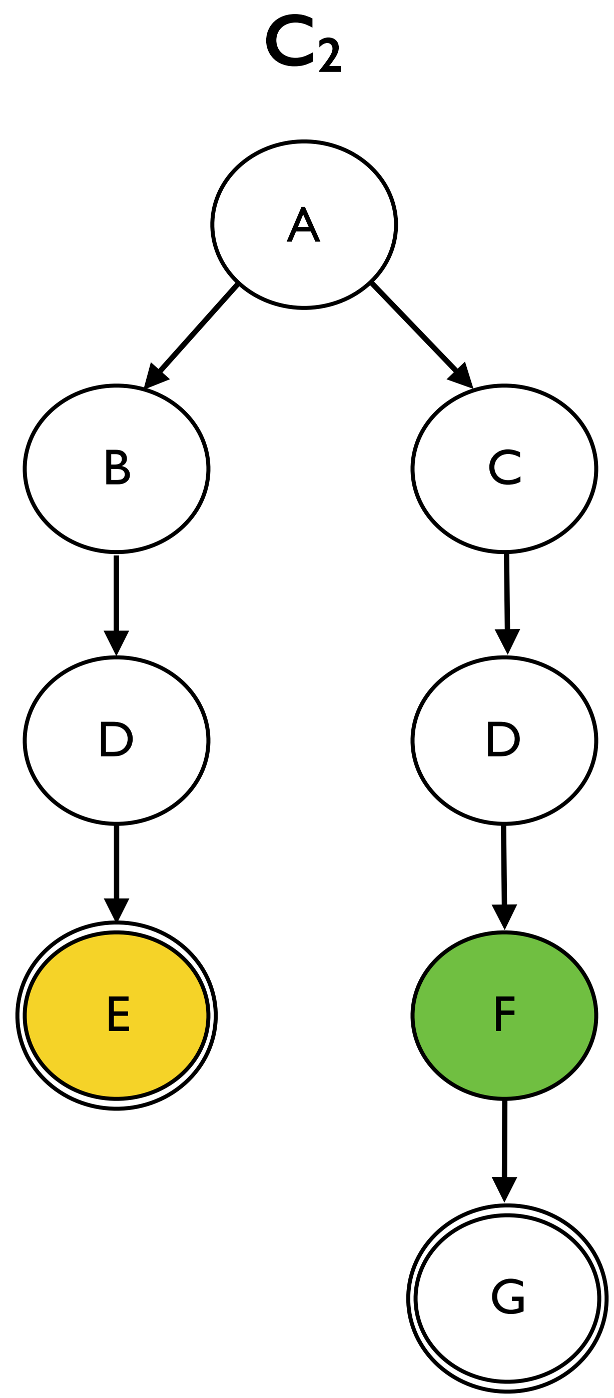



$C_2$

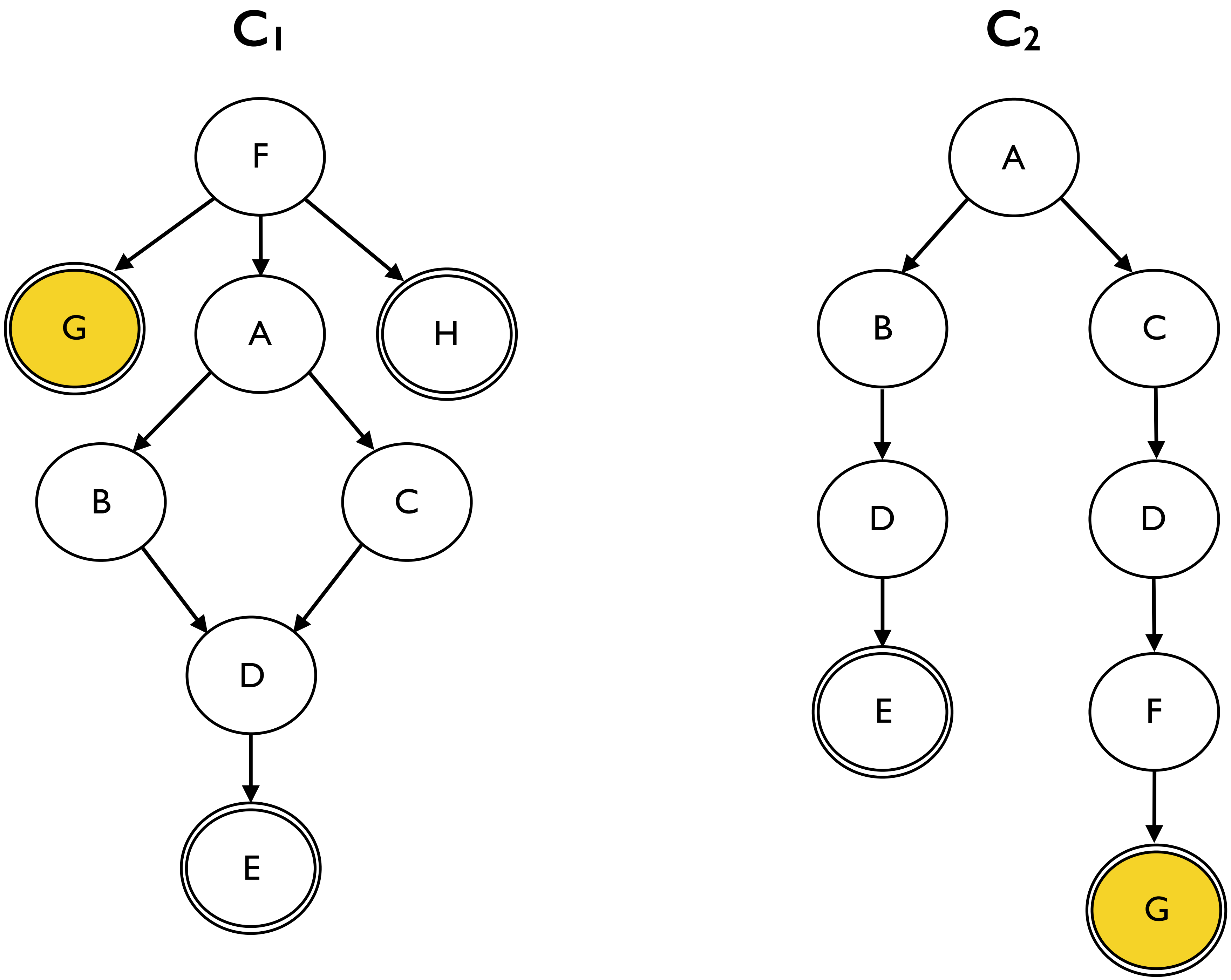


# Failure Safety (Recap)

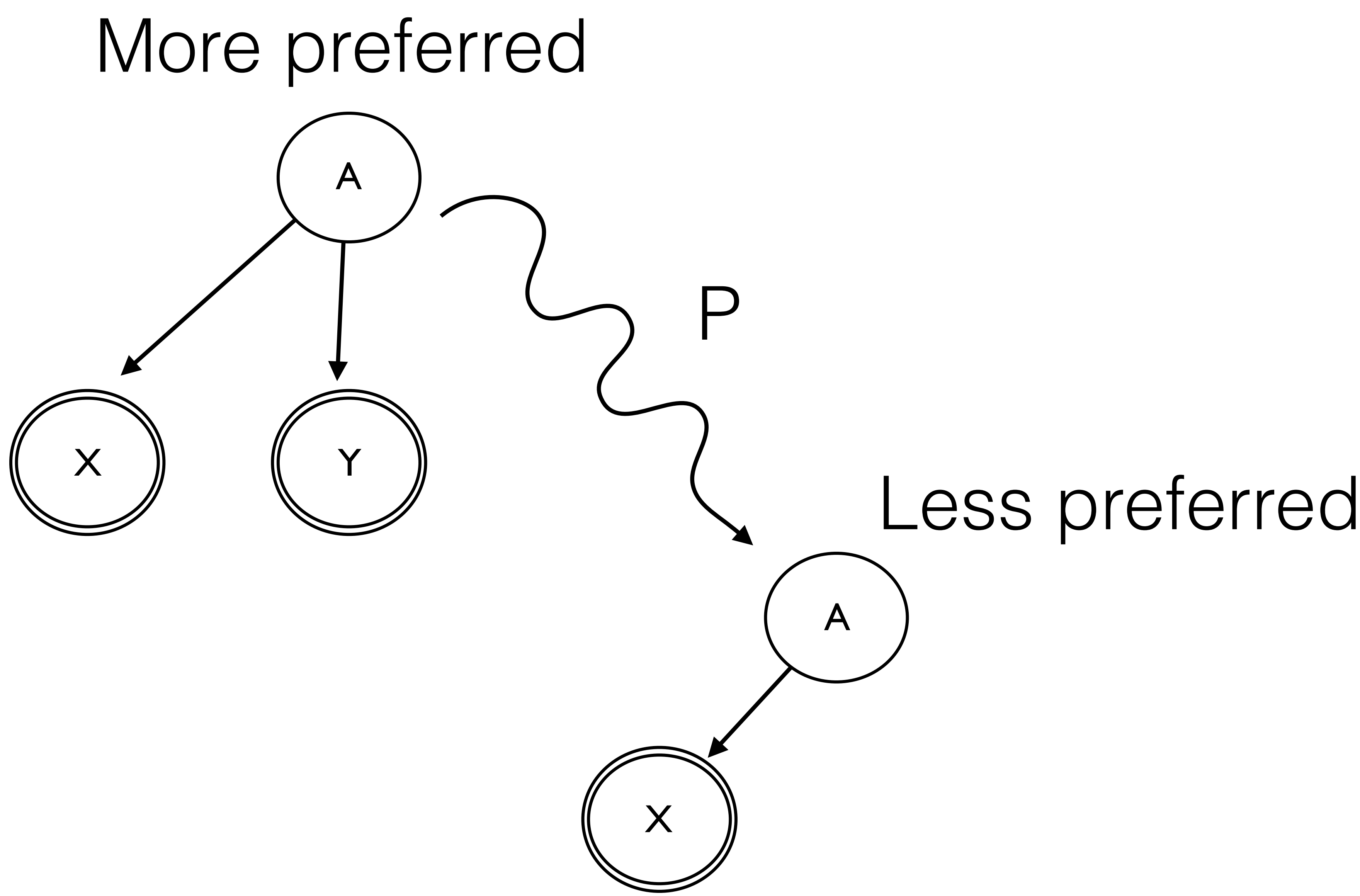


# Failure Safety (Recap)



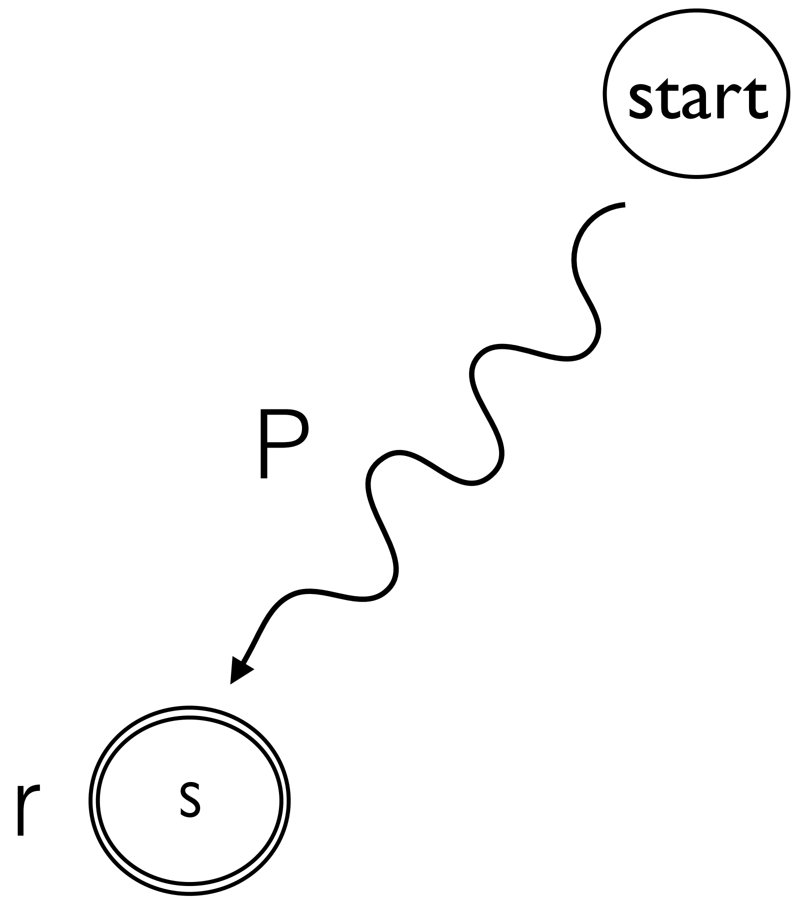
# Failure Safety (Recap)



# Proof of Correctness (High level)

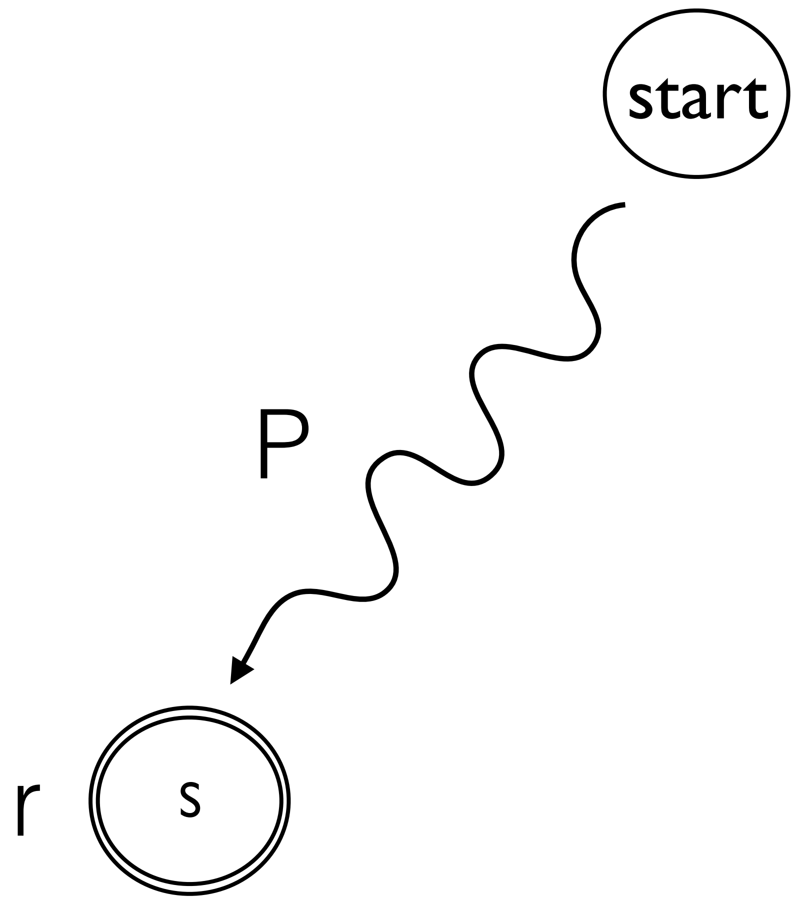
**Statement:**

Traffic always flows along ***some best simple*** path to source *s* when such a path exists in the network (given failures)



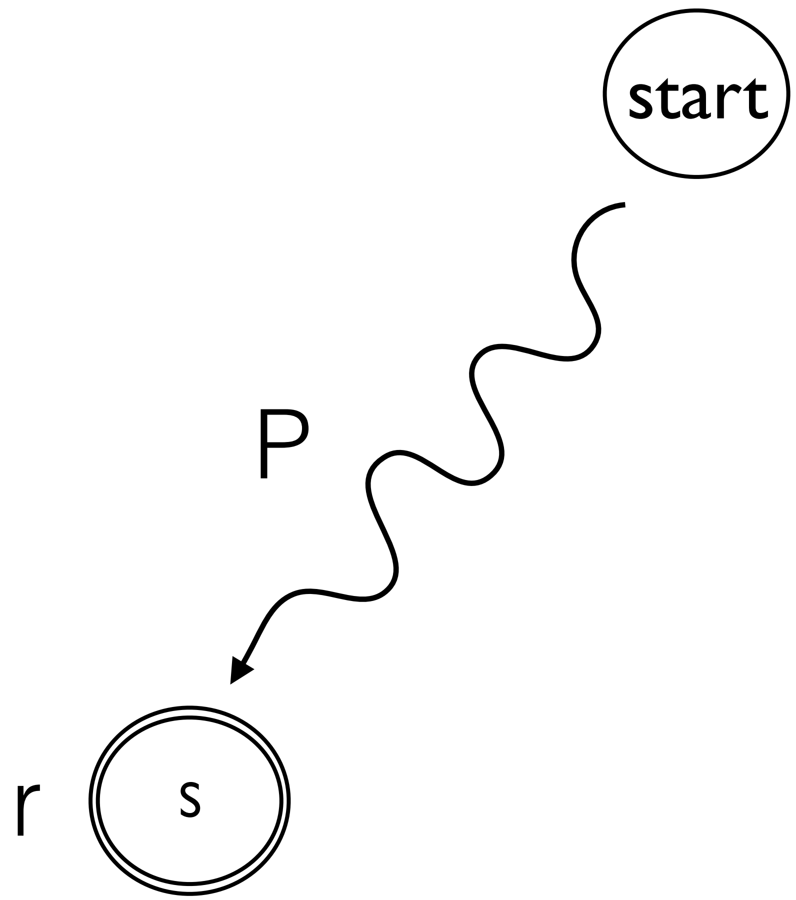

Assume path P is one
of the highest rank
simple paths in the network
given the failures

Then path P exists in the PG
with (best) rank of r for source s

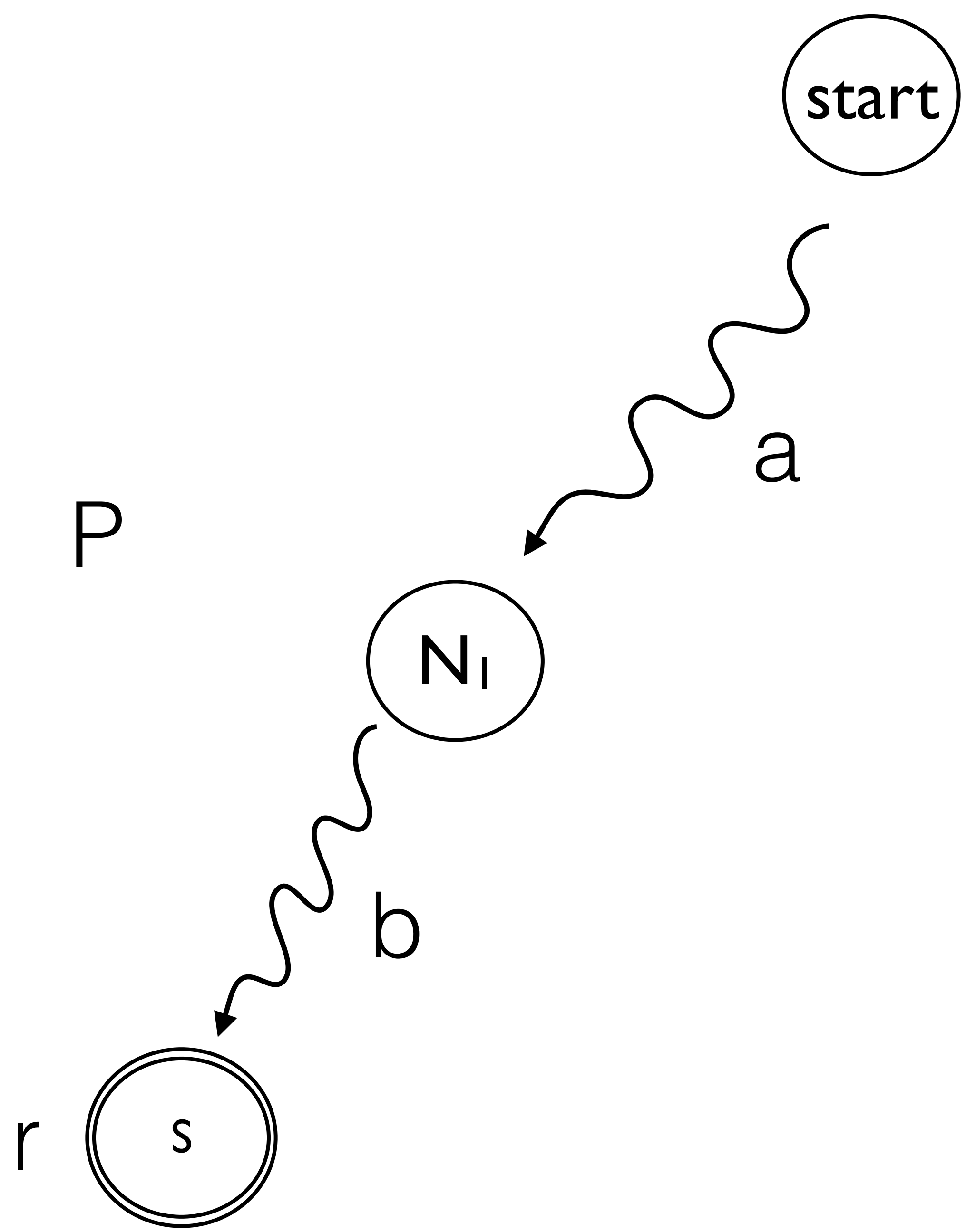



Assume path P is one
of the highest rank
simple paths in the network
given the failures

Then path P exists in the PG
with (best) rank of r for source s



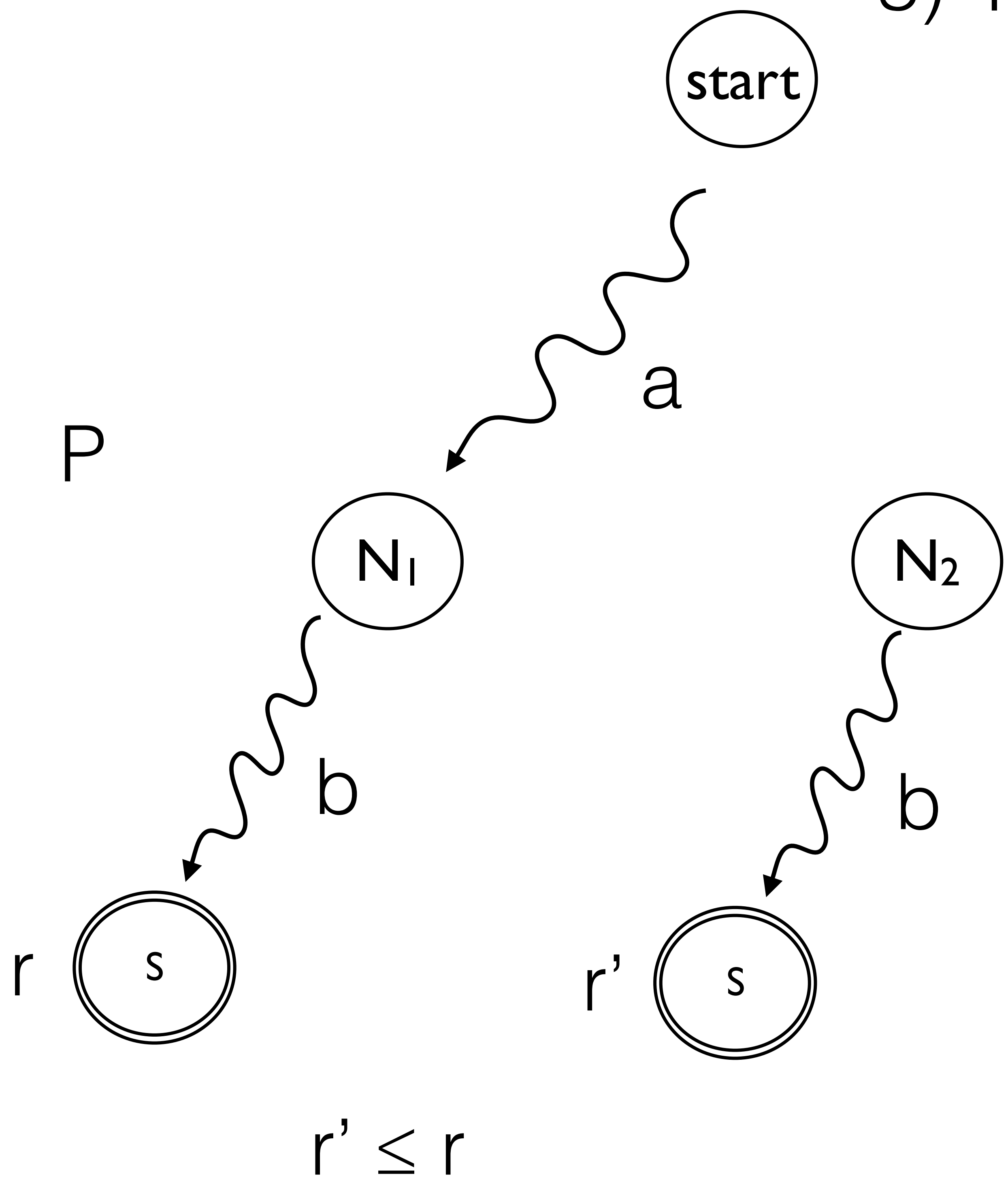

The only way traffic does not flow along path P is if some node on P prefers a different advertisement




The only way traffic does not flow along path P is if some node on P prefers a different advertisement

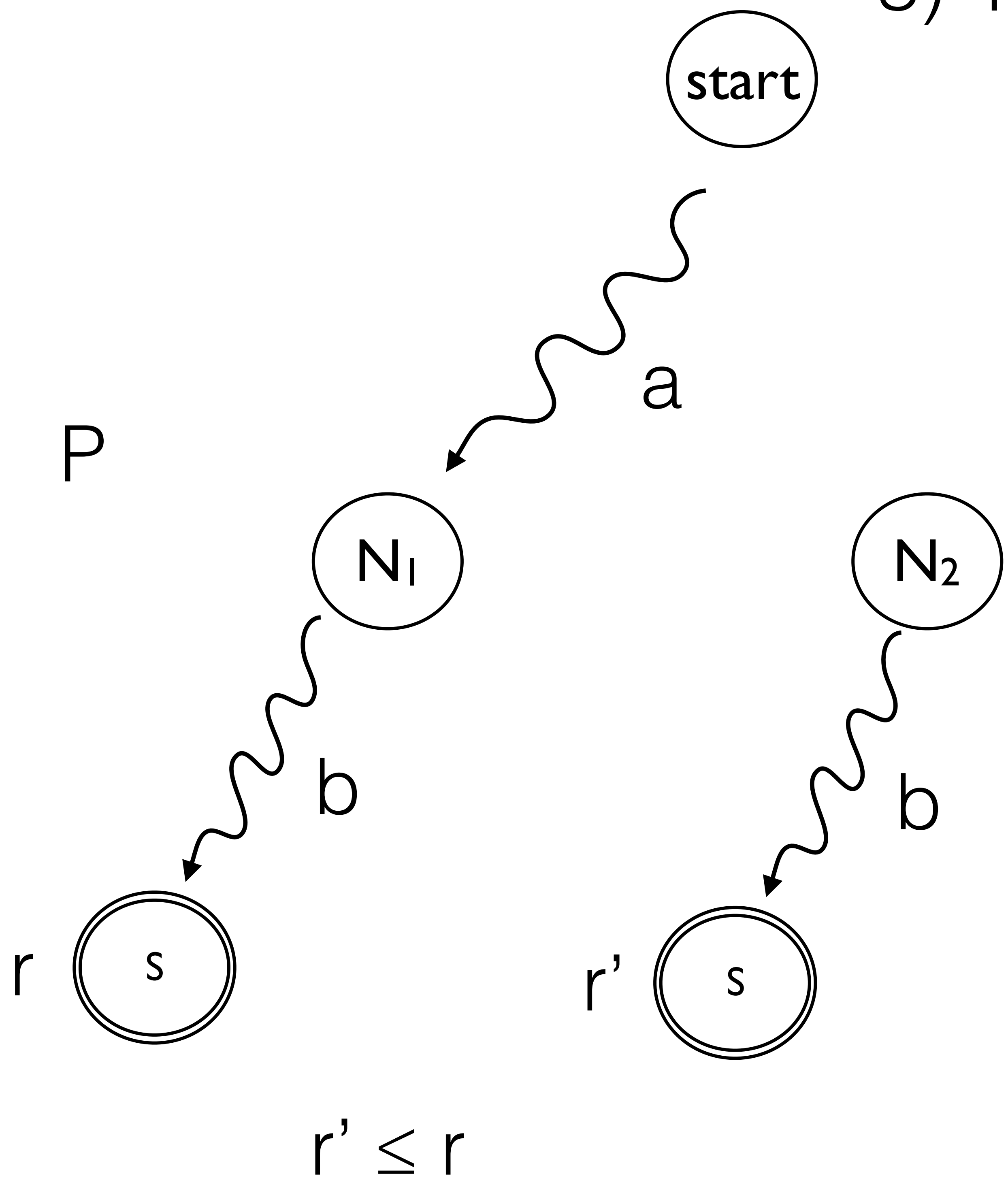
**What we know:**

1) advertisement reaches $N_2$
2) Failure analysis prefers $N_2$
3) $N_2$ has the same path b to $r' \leq r$



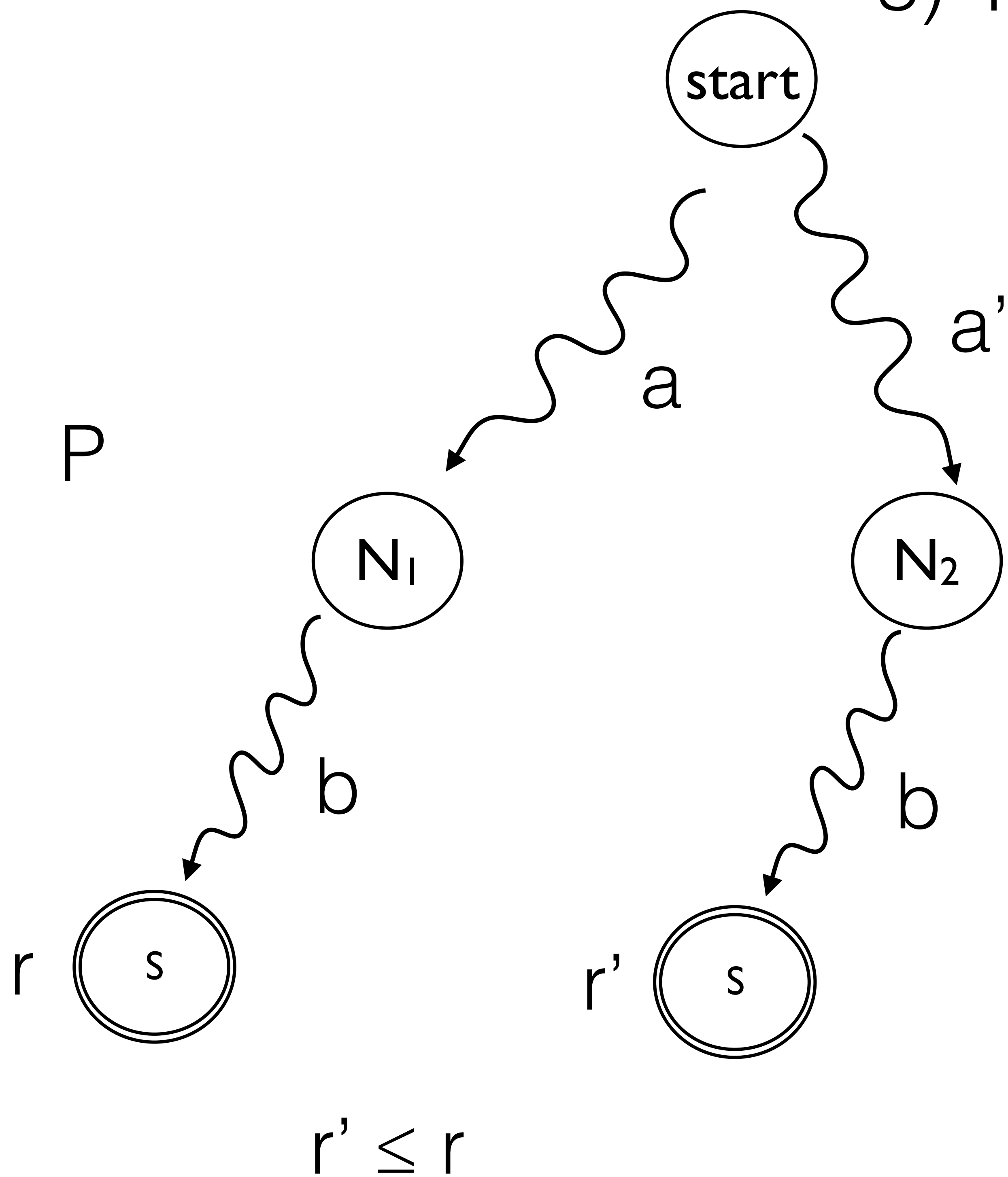
**What we know:**

1) advertisement reaches $N_2$
2) Failure analysis prefers $N_2 \geq N_1$
3) $N_2$ has the same path b to $r' \leq r$



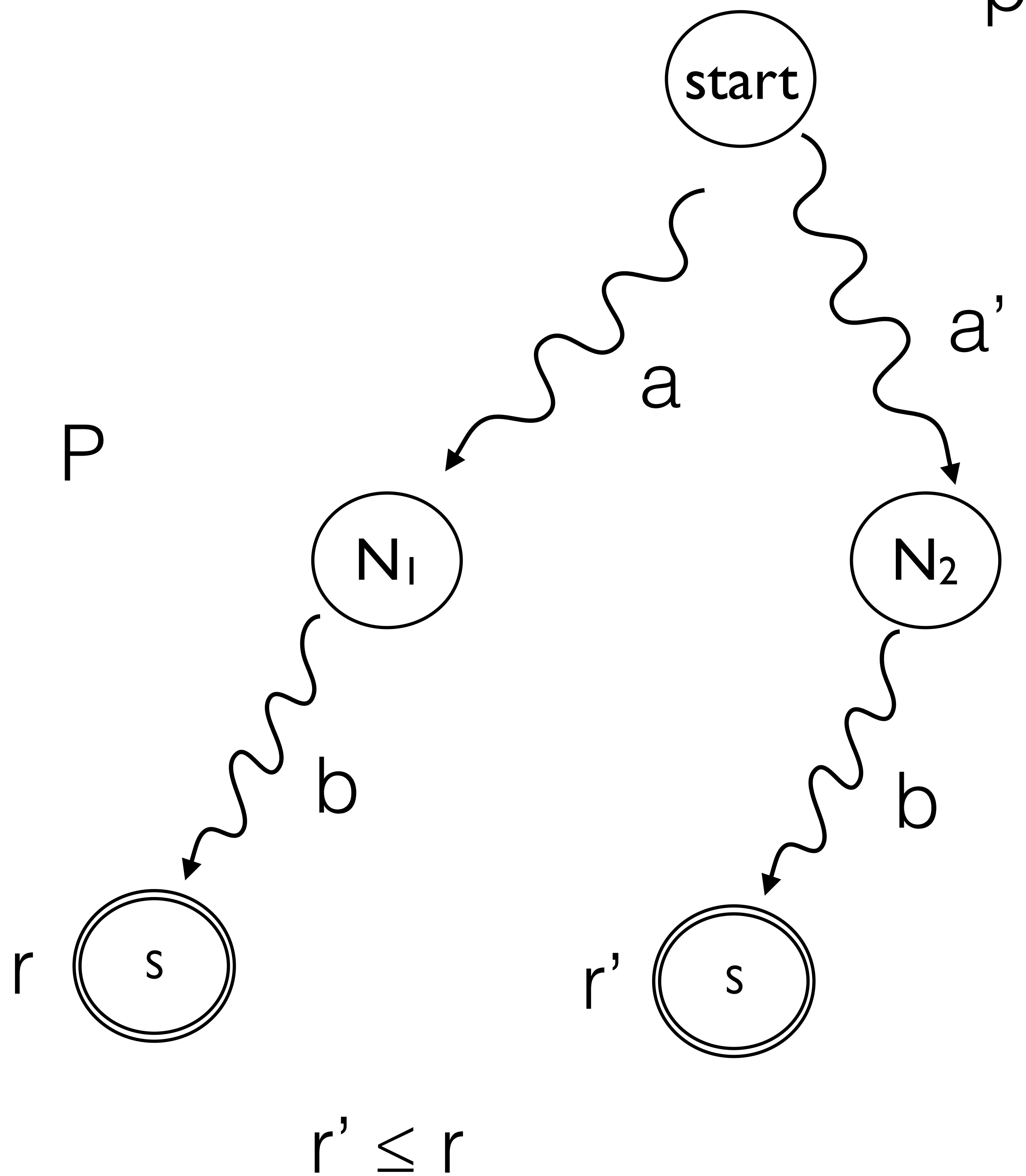
**What we know:**

1) advertisement reaches $N_2$
2) Failure analysis prefers $N_2 \geq N_1$
3) $N_2$ has the same path b to $r' \leq r$

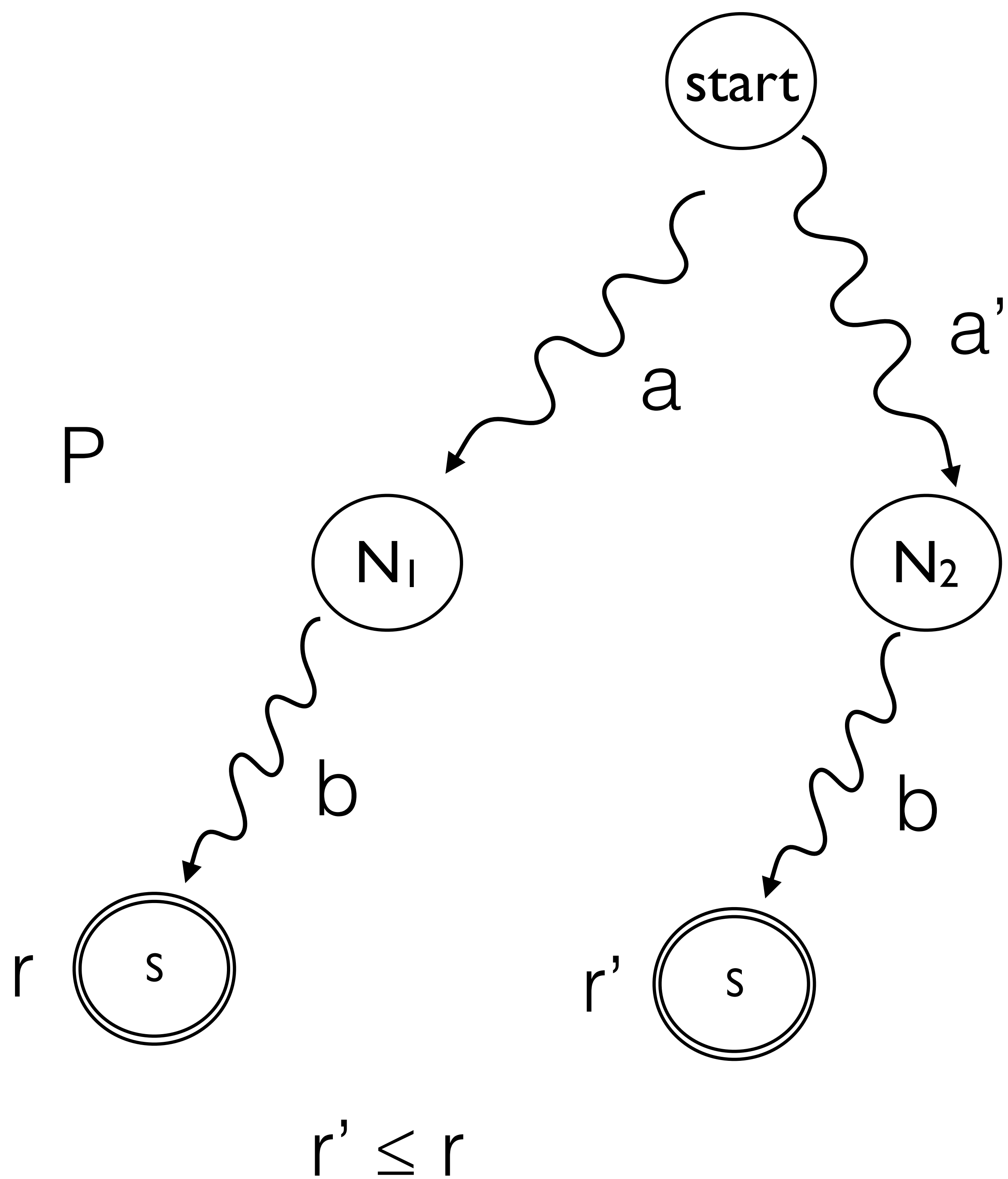


**Case 1** (a’.b is a simple path)

Proceed by induction on path length of b



**Case 2** (a'.b is **not** simple)

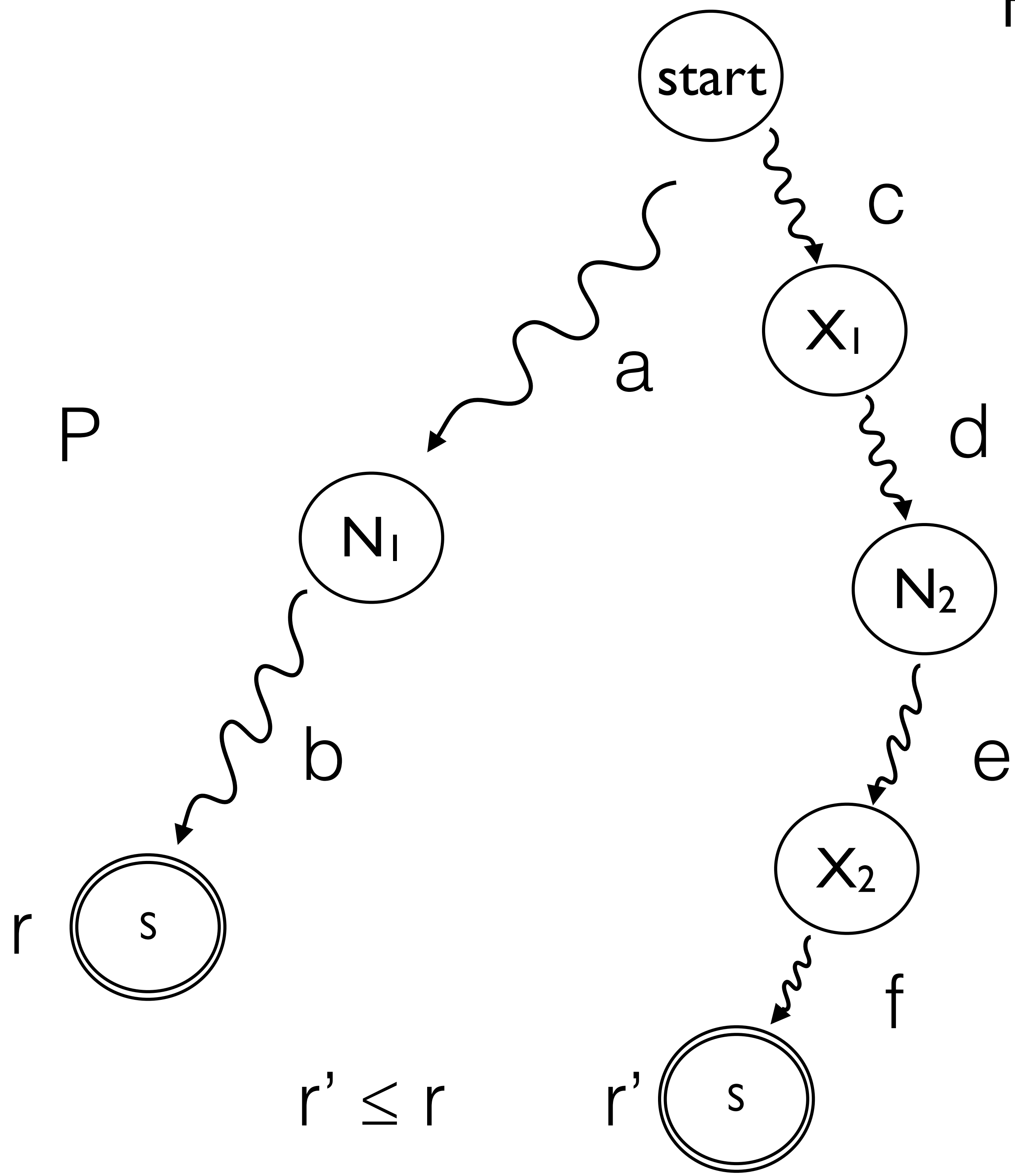



a' must be simple (advertisement)

b must be simple (since P is simple)

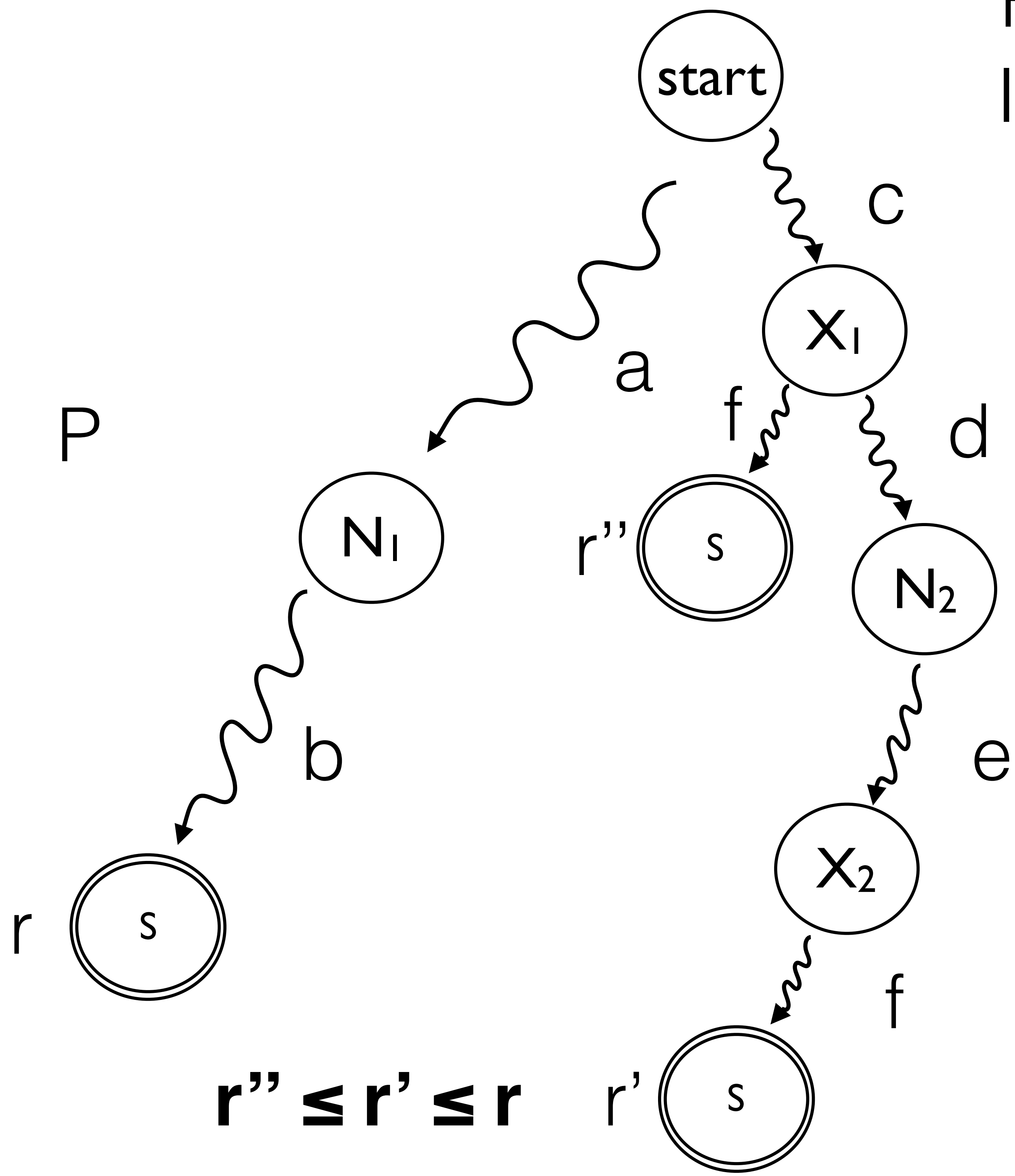
**Case 2** (a’.b is **not** simple)

From failure analysis, $X_1 \geq X_2$



**Case 2** (a'.b is **not** simple)

From failure analysis, $X_1 \geq X_2$
Induction on smaller paths c,f




**r'' ≤ r' ≤ r**