NP-Completeness

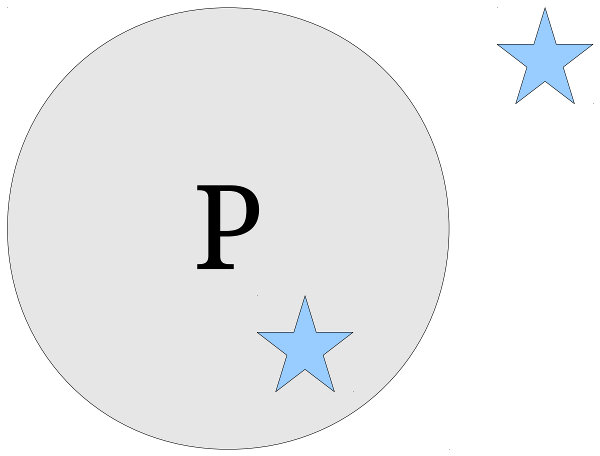
Part II

Outline for Today

- **Recap from Last Time**
  - What is **NP**-completeness again, anyway?
- **3SAT**
  - A simple, canonical **NP**-complete problem.
- **Independent Sets**
  - Discovering a new **NP**-complete problem.
- **Gadget-Based Reductions**
  - A common technique in **NP** reductions.
- **3-Colorability**
  - A more elaborate **NP**-completeness reduction.
- **Recent News**
  - Things happened! What were they?

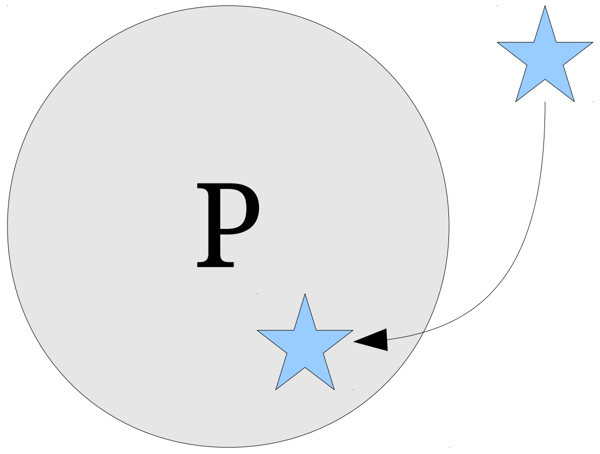
Polynomial-Time Reductions

- If $A \leq_{\mathrm{P}} B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.



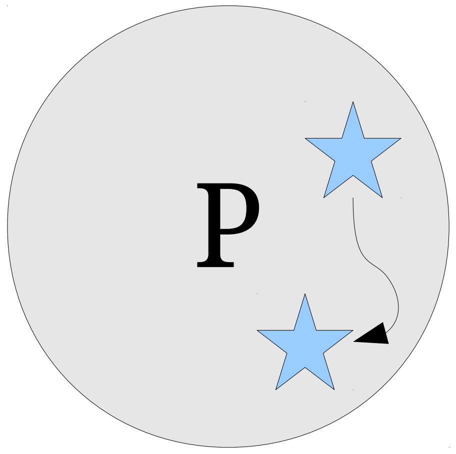
Polynomial-Time Reductions

- If $A \leq_P B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.



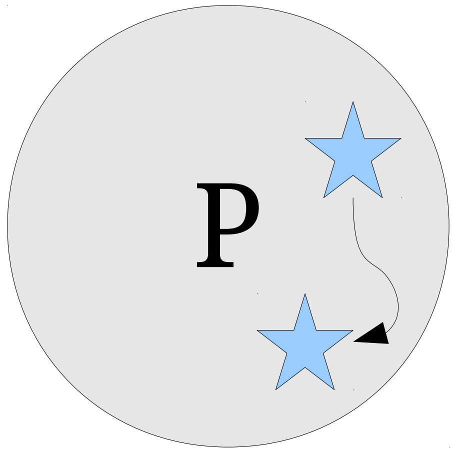
Polynomial-Time Reductions

- If $A \leq_P B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.



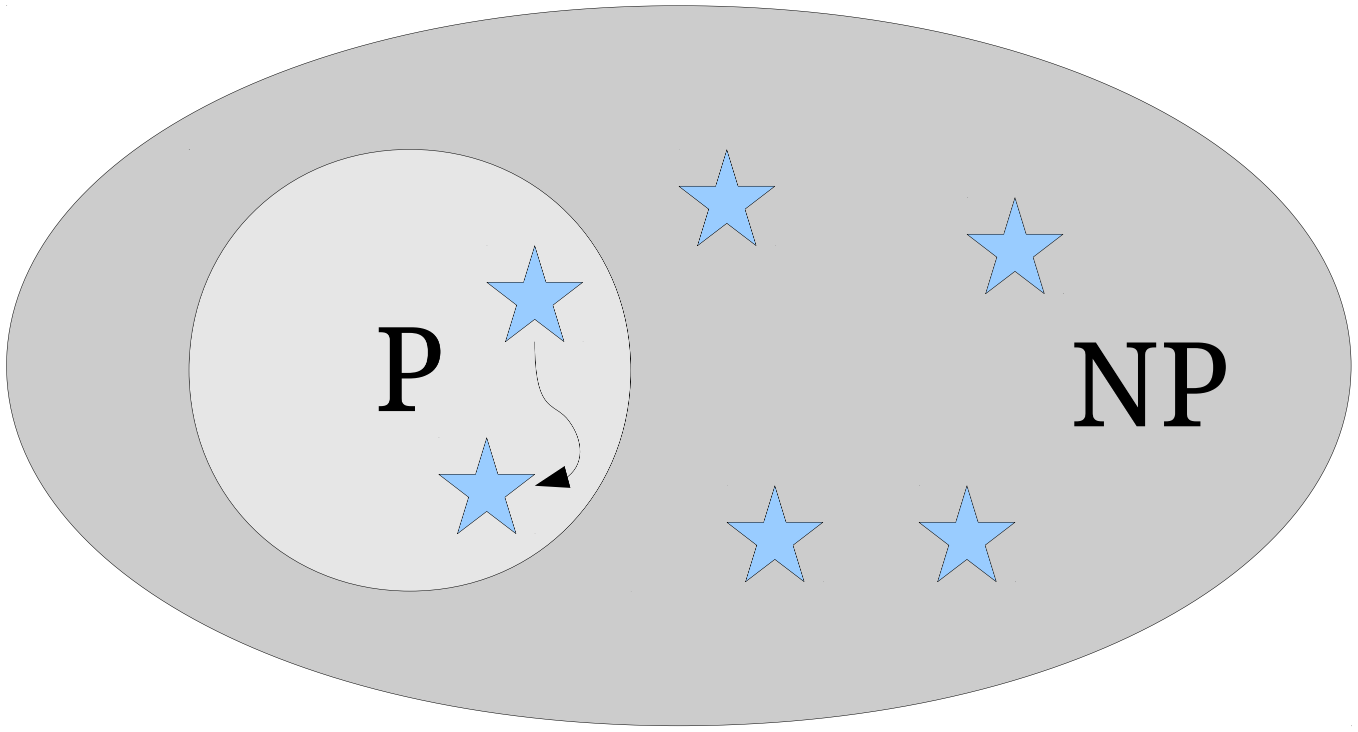
Polynomial-Time Reductions

- If $A \leq_P B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.
- If $A \leq_P B$ and $B \in \mathbf{NP}$, then $A \in \mathbf{NP}$.



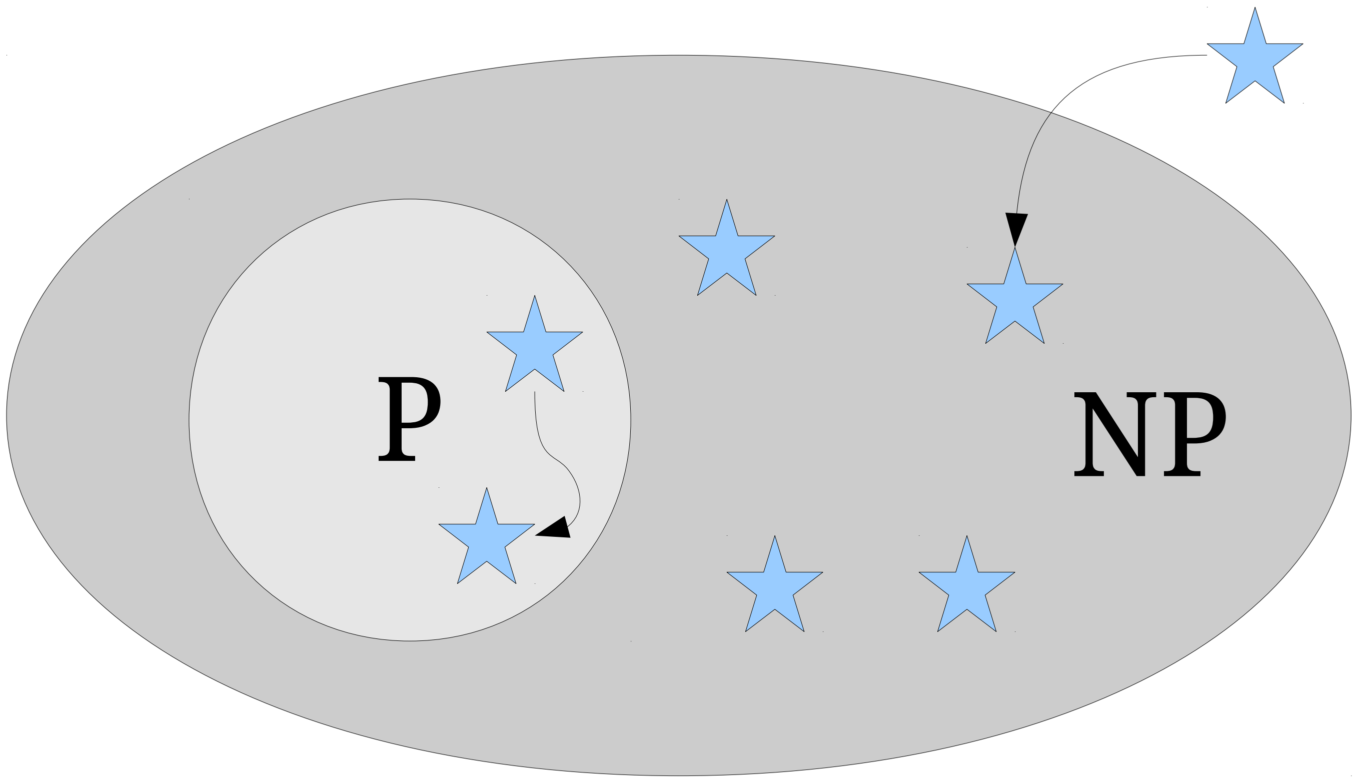
Polynomial-Time Reductions

- If $A \leq_P B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.
- If $A \leq_P B$ and $B \in \mathbf{NP}$, then $A \in \mathbf{NP}$.



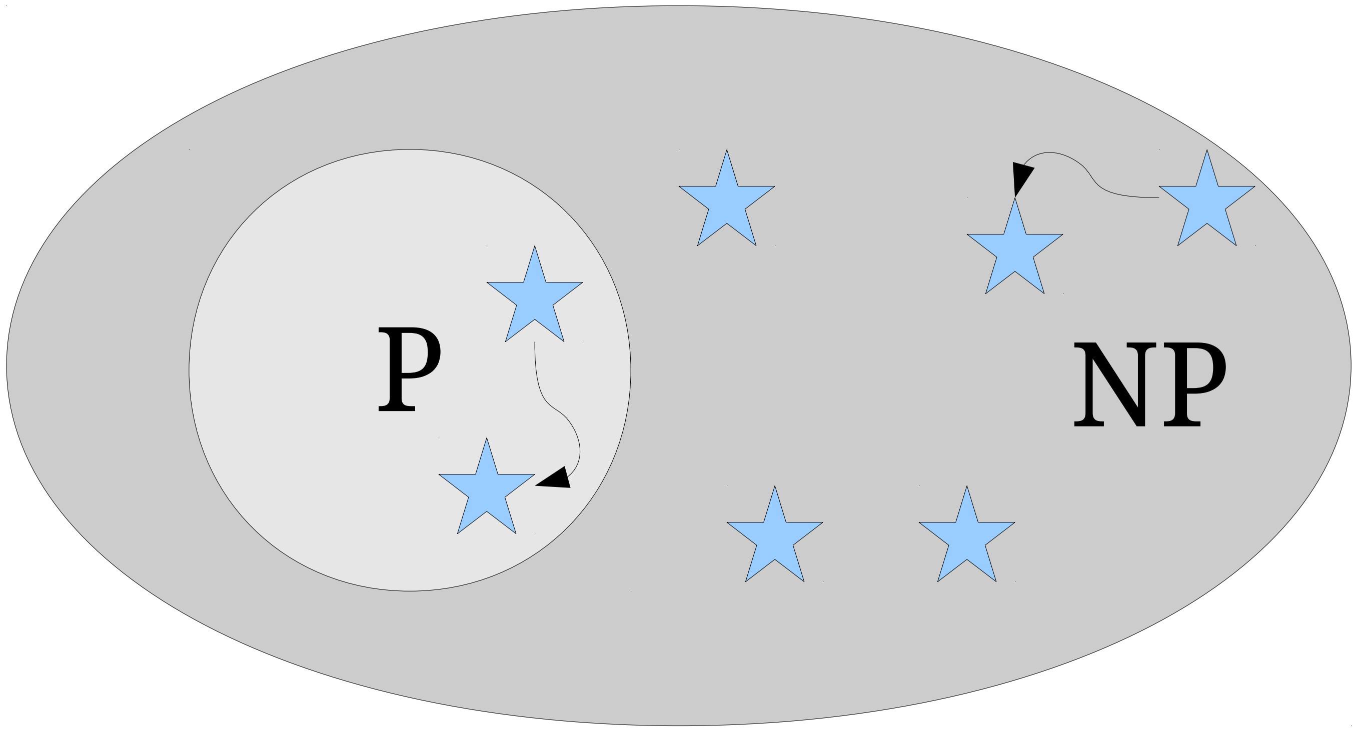
Polynomial-Time Reductions

- If $A \leq_P B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.
- If $A \leq_P B$ and $B \in \mathbf{NP}$, then $A \in \mathbf{NP}$.



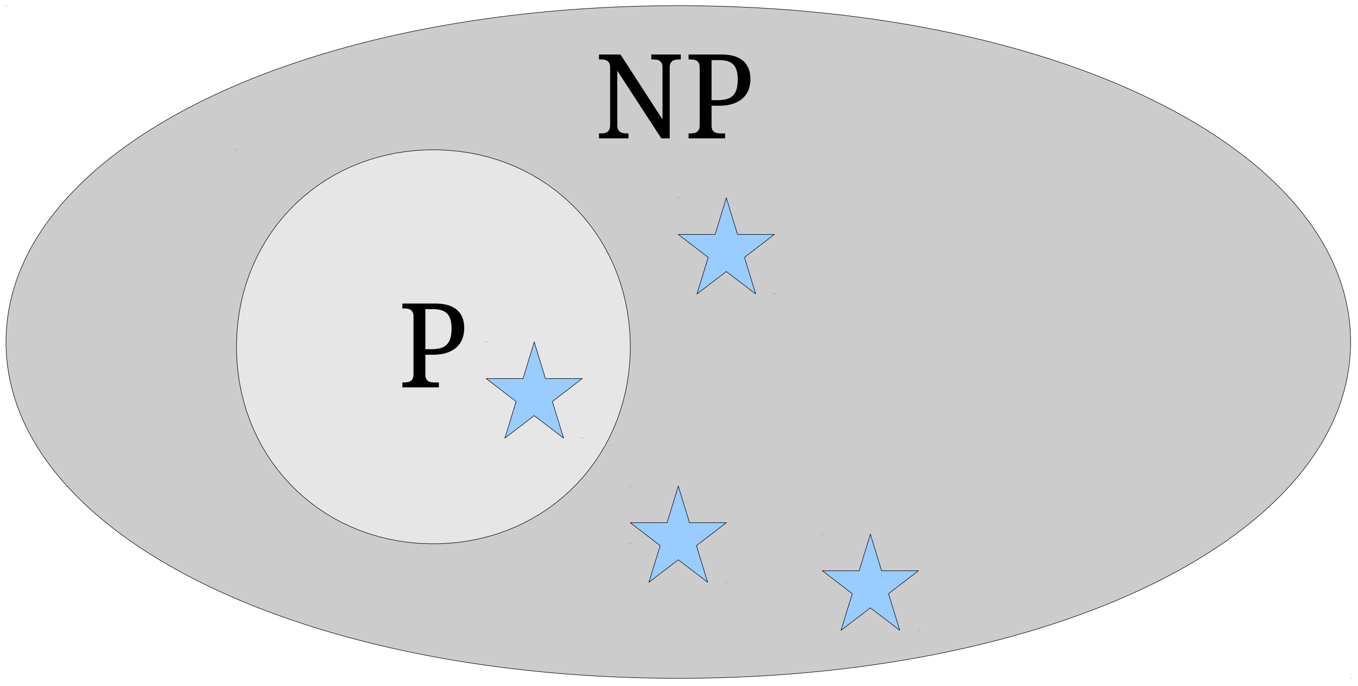
Polynomial-Time Reductions

- If $A \leq_P B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.
- If $A \leq_P B$ and $B \in \mathbf{NP}$, then $A \in \mathbf{NP}$.



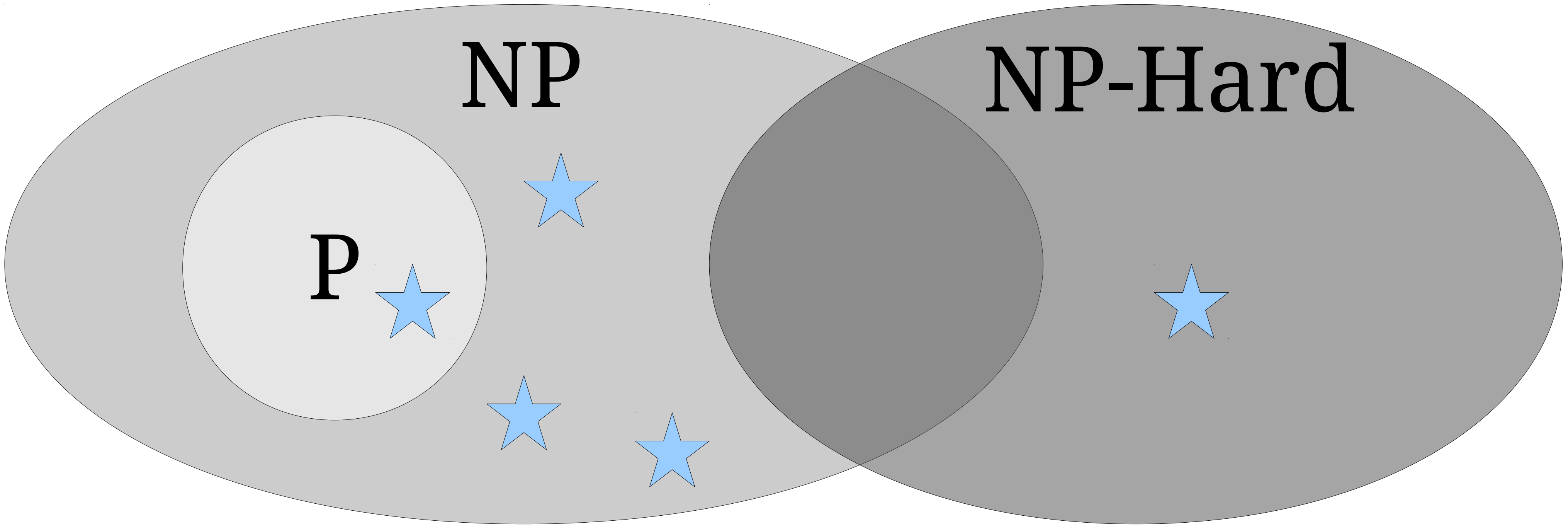
NP-Hardness

- A language L is called ***NP-hard*** if for *every* $A \in \mathbf{NP}$, we have $A \leq_{\mathrm{P}} L$.



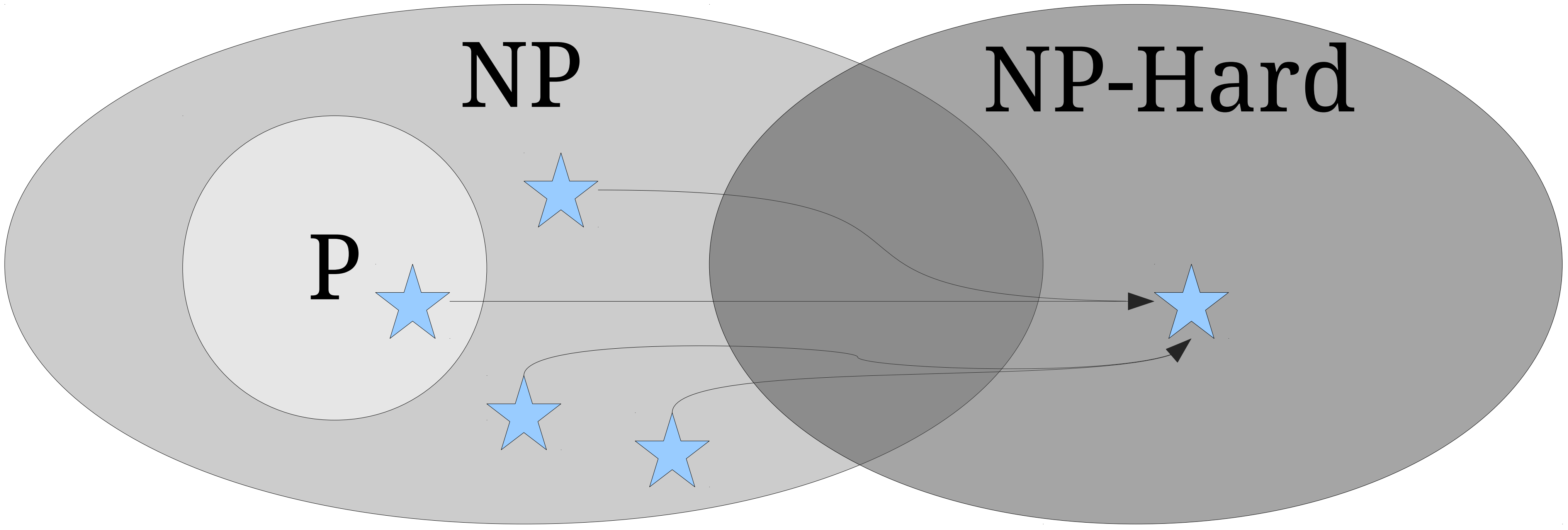
NP-Hardness

- A language L is called ***NP-hard*** if for *every* $A \in \mathbf{NP}$, we have $A \leq_{\mathrm{P}} L$.



NP-Hardness

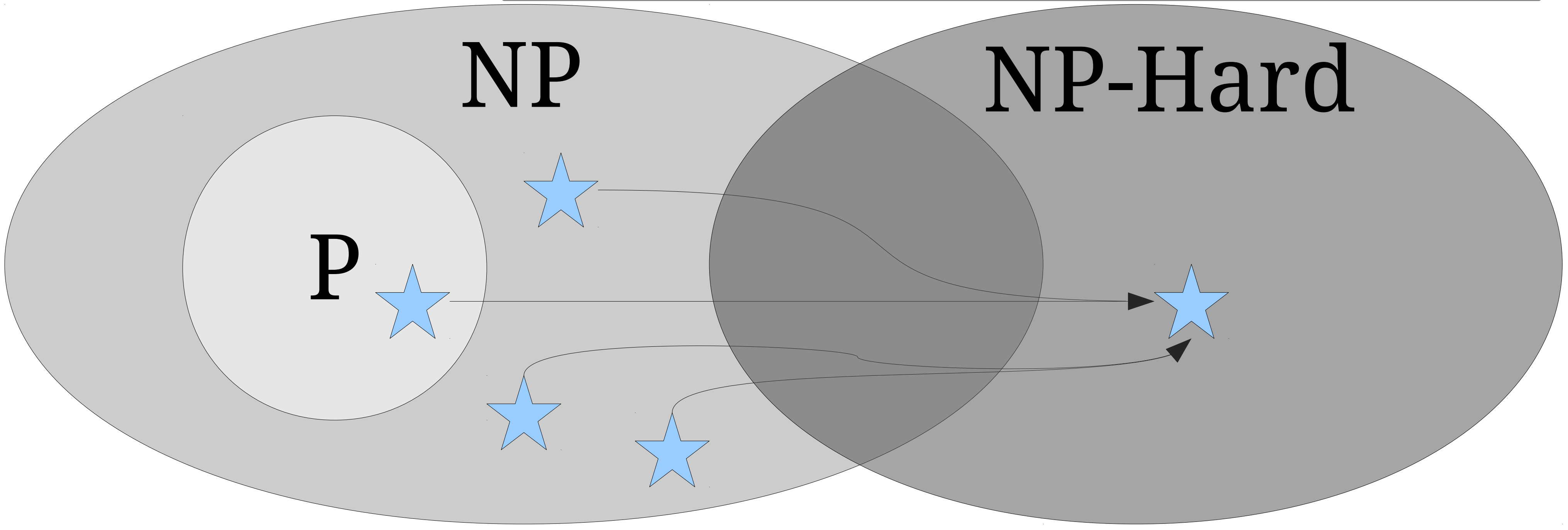
- A language L is called ***NP-hard*** if for *every* $A \in \mathbf{NP}$, we have $A \leq_P L$.



NP-Hardness

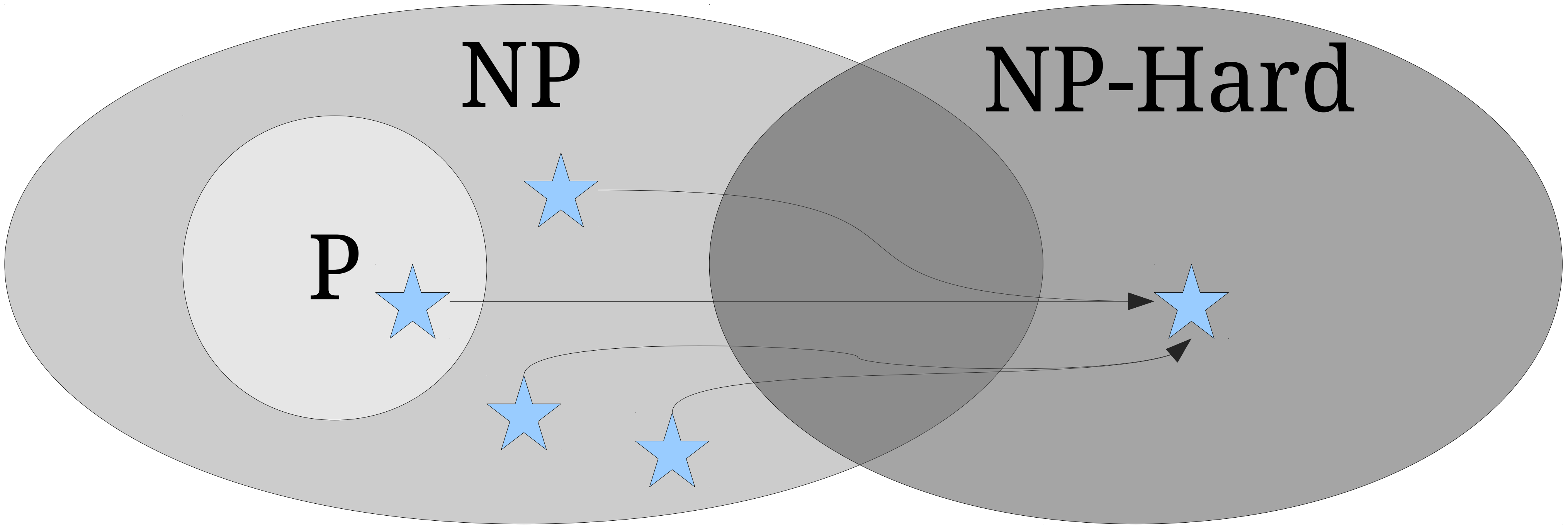
- A language L is called ***NP-hard*** if for *every* $A \in$ **NP**, we have $A \leq_P L$.

Intuitively: L has to be at least as hard as every problem in **NP**, since an algorithm for L can be used to decide all problems in **NP**.



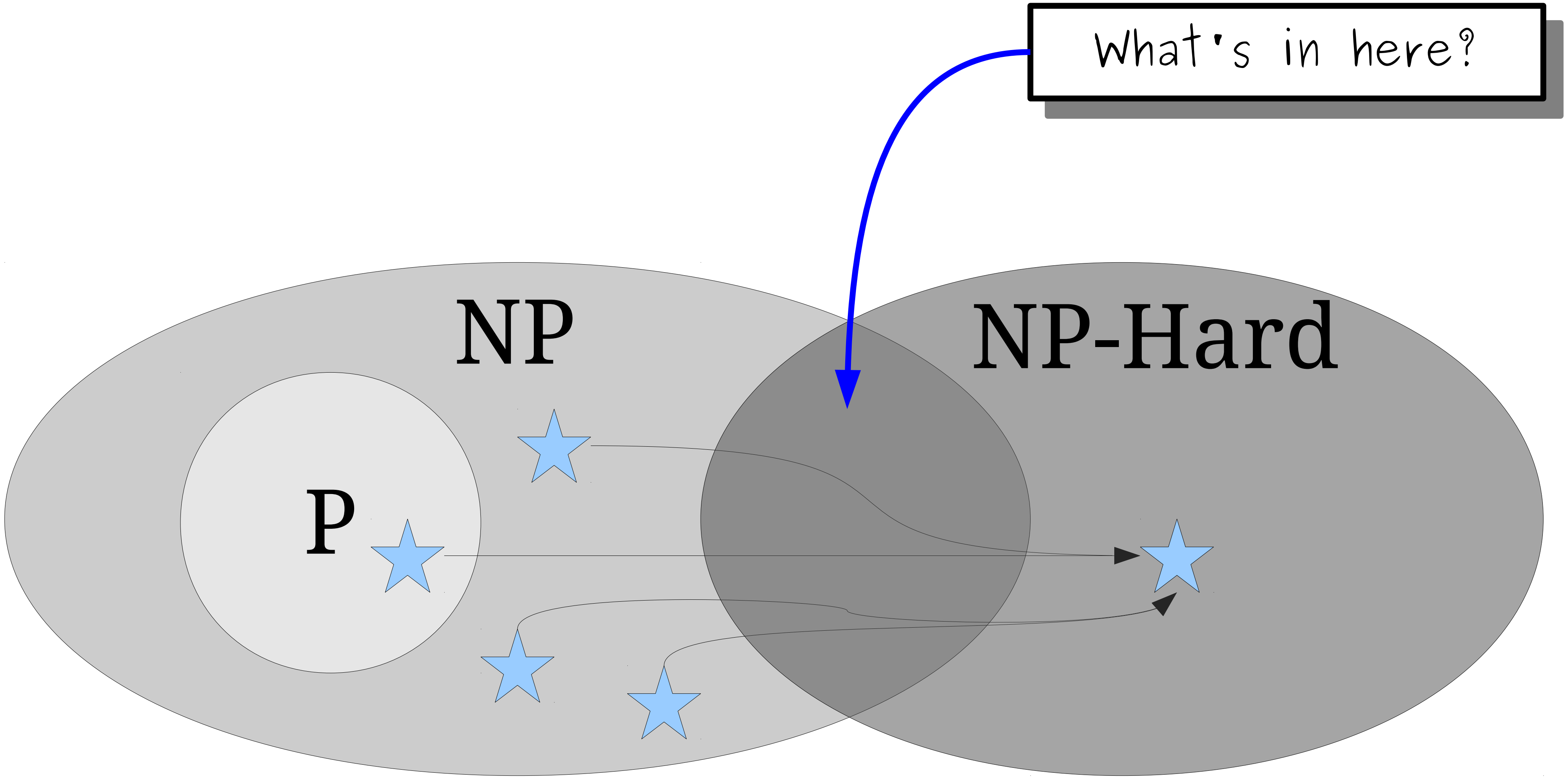
NP-Hardness

- A language L is called ***NP-hard*** if for *every* $A \in$ **NP**, we have $A \leq_P L$.



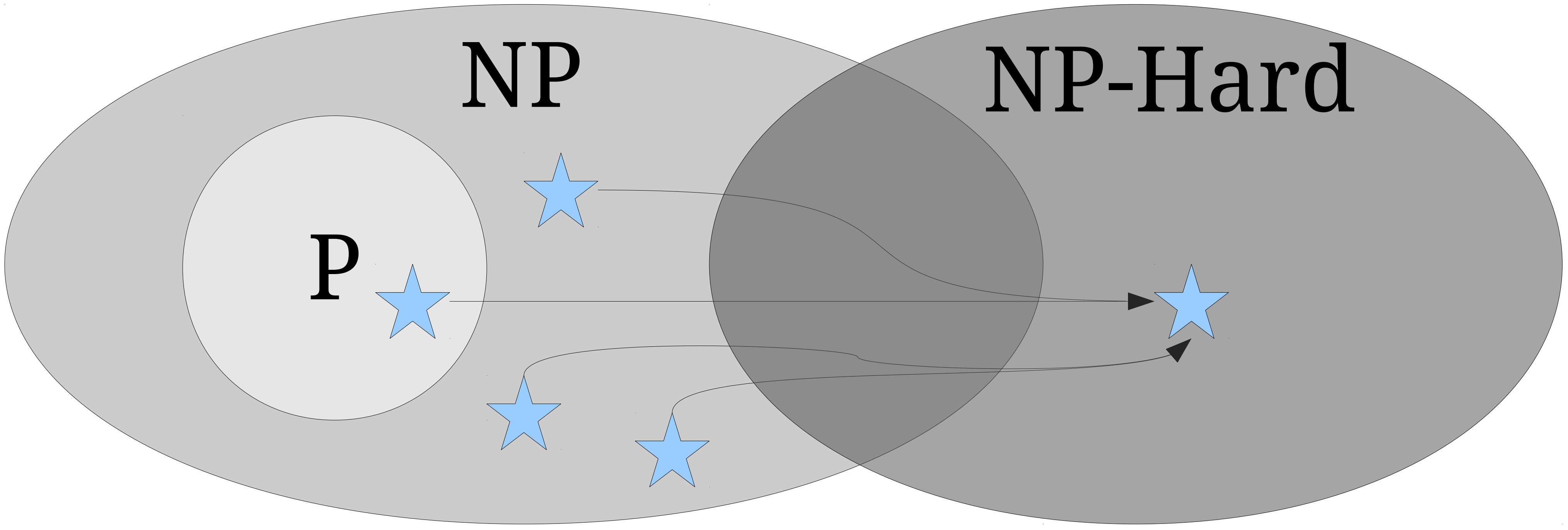
NP-Hardness

- A language L is called ***NP-hard*** if for *every* $A \in$ **NP**, we have $A \leq_P L$.



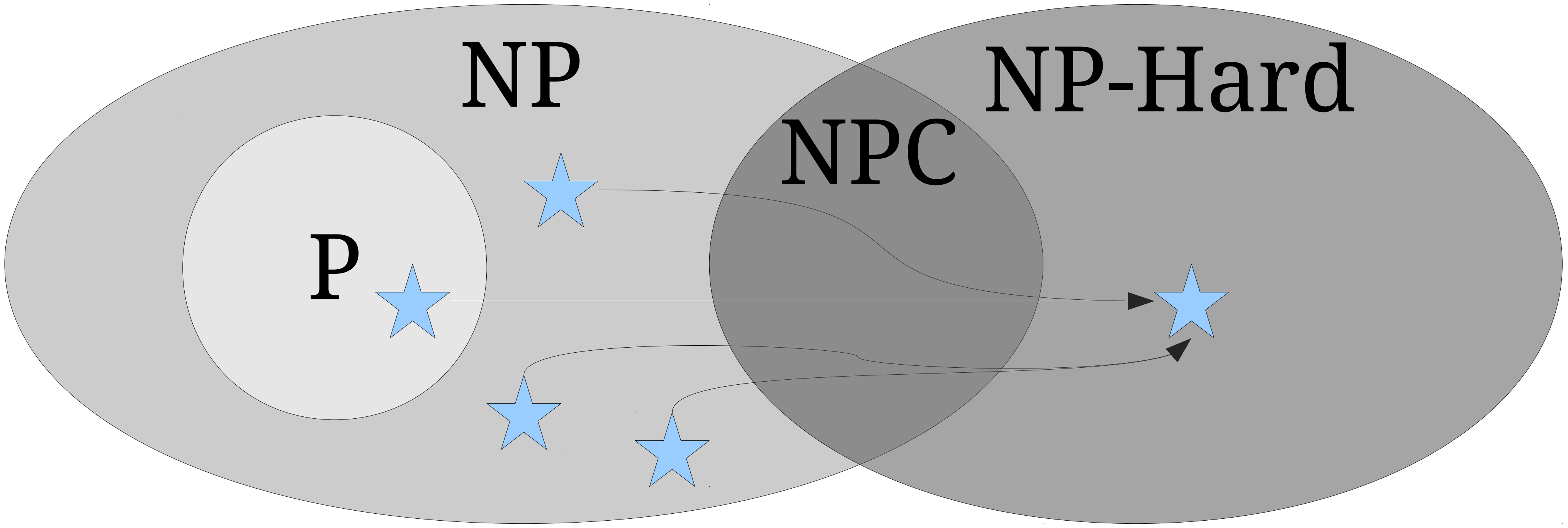
NP-Hardness

- A language L is called ***NP-hard*** if for *every* $A \in \mathbf{NP}$, we have $A \leq_P L$.
- A language in L is called ***NP-complete*** if L is **NP**-hard and $L \in \mathbf{NP}$.
- The class ***NPC*** is the set of **NP**-complete problems.



NP-Hardness

- A language L is called ***NP-hard*** if for *every* $A \in \mathbf{NP}$, we have $A \leq_{\mathrm{P}} L$.
- A language in L is called ***NP-complete*** if L is **NP**-hard and $L \in \mathbf{NP}$.
- The class ***NPC*** is the set of **NP**-complete problems.

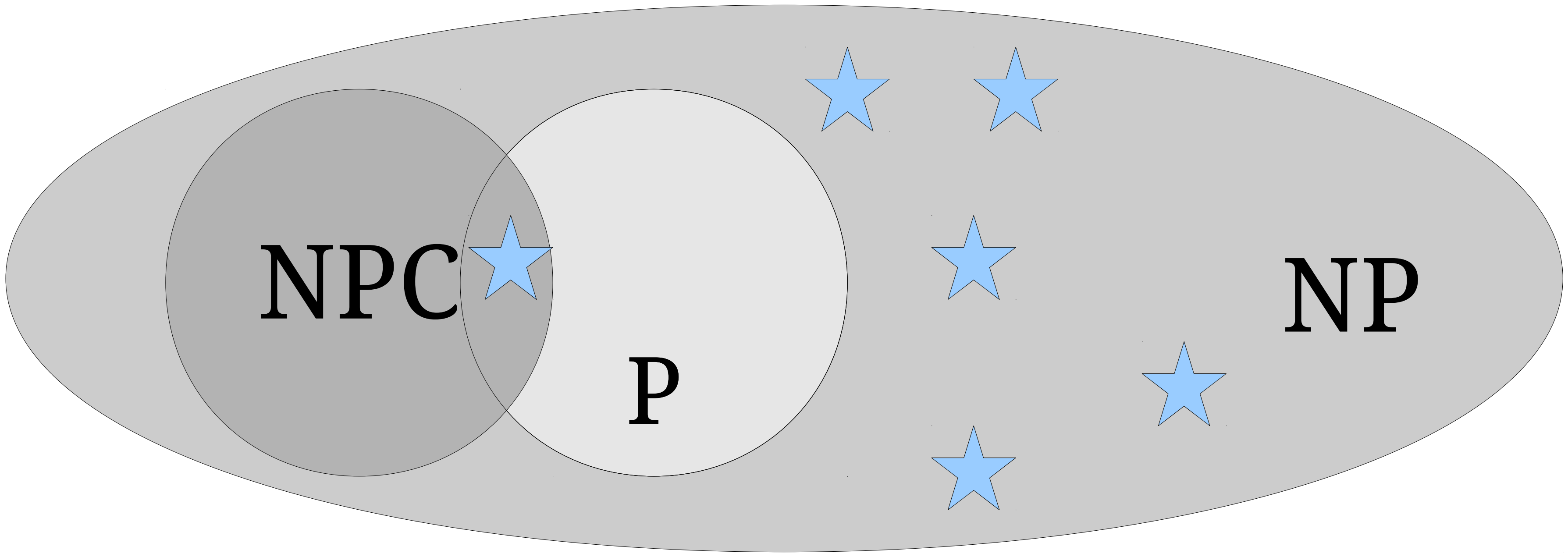


The Tantalizing Truth

Theorem: If *any* **NP**-complete language is in **P**, then **P** = **NP**.

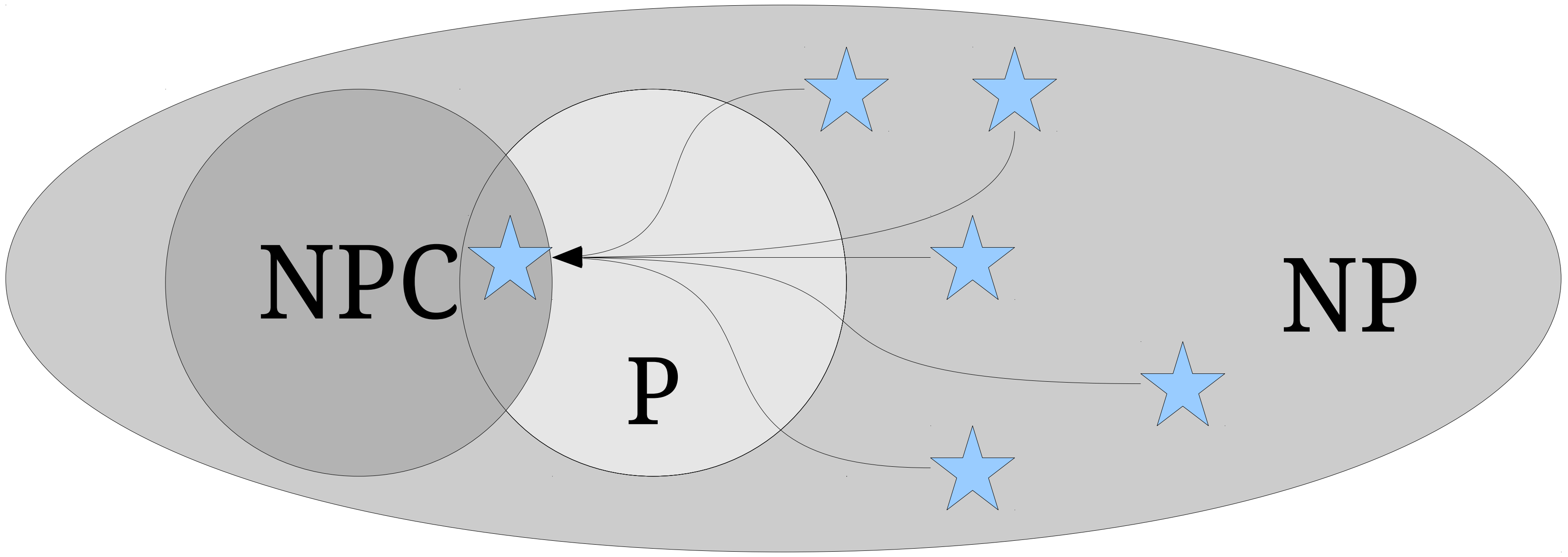
The Tantalizing Truth

Theorem: If *any* **NP**-complete language is in **P**, then **P** = **NP**.



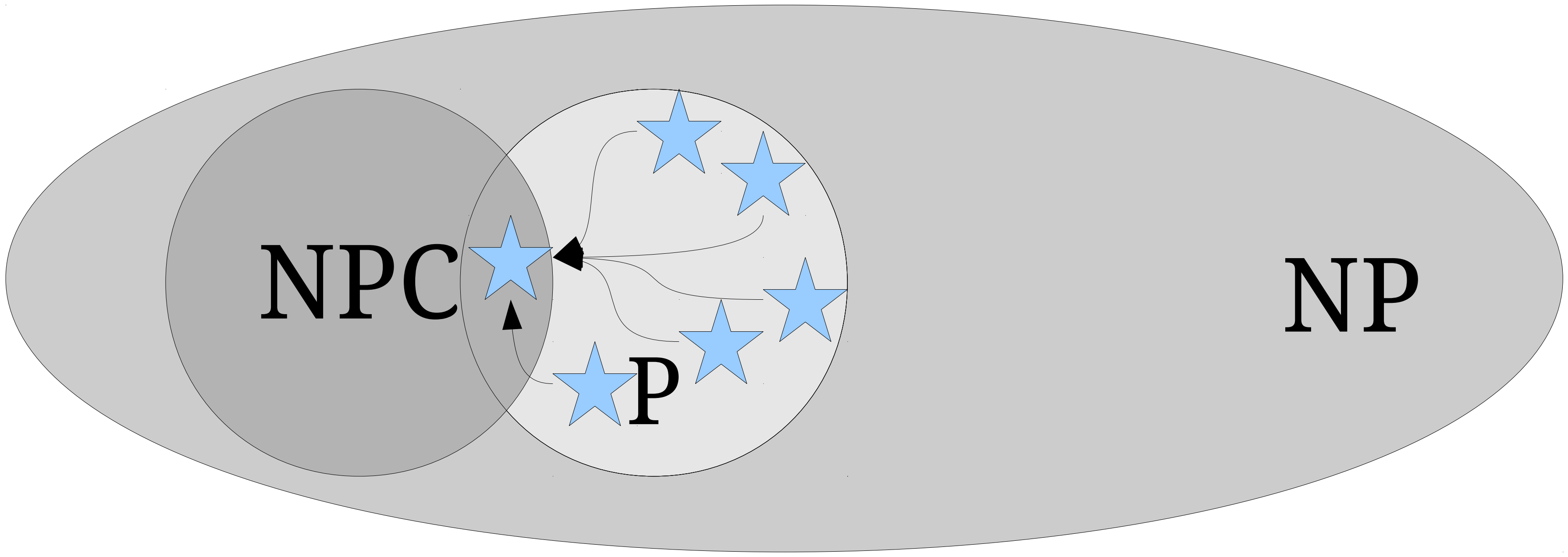
The Tantalizing Truth

Theorem: If *any* **NP**-complete language is in **P**, then **P** = **NP**.



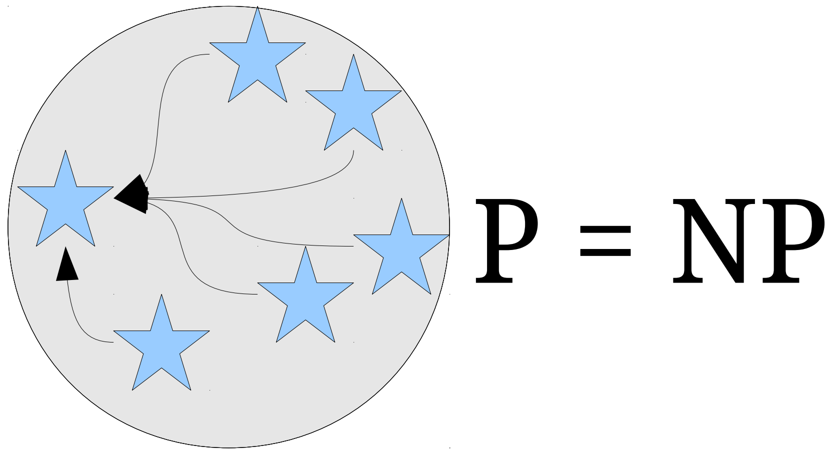
The Tantalizing Truth

Theorem: If *any* **NP**-complete language is in **P**, then **P** = **NP**.



The Tantalizing Truth

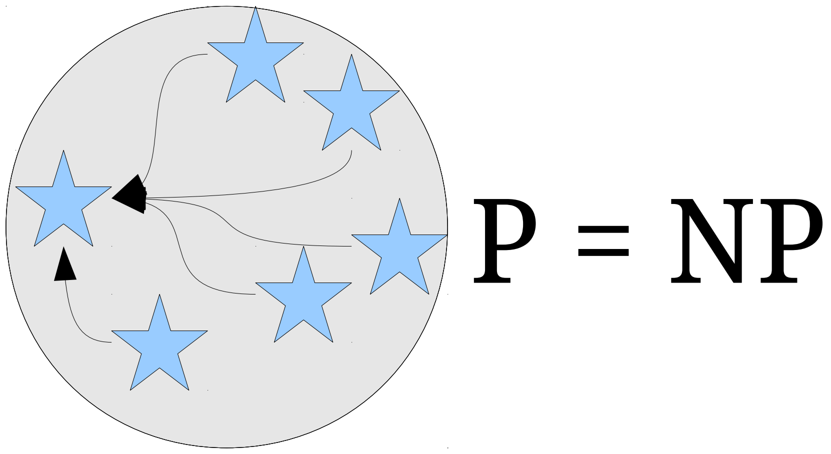
Theorem: If *any* **NP**-complete language is in **P**, then **P** = **NP**.



The Tantalizing Truth

Theorem: If *any* **NP**-complete language is in **P**, then **P** = **NP**.

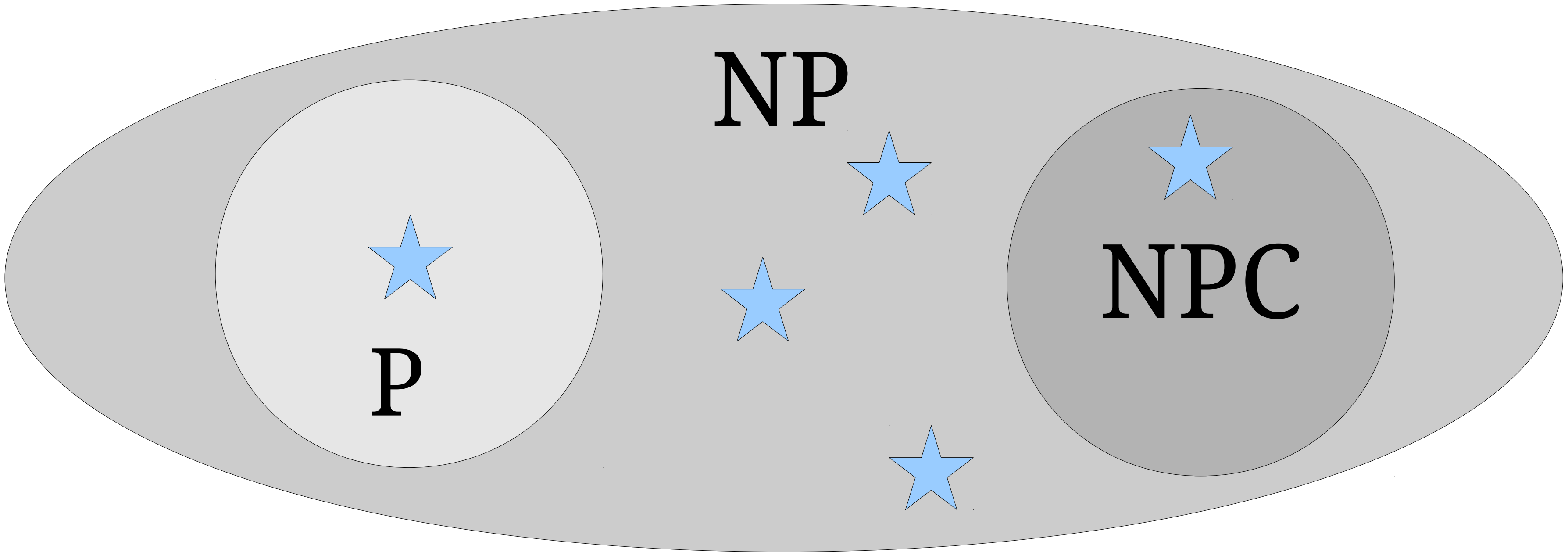
Proof: Suppose that L is **NP**-complete and $L \in$ **P**. Now consider any arbitrary **NP** problem A. Since L is **NP**-complete, we know that $A \leq_p L$. Since $L \in$ **P** and $A \leq_p L$, we see that $A \in$ **P**. Since our choice of A was arbitrary, this means that **NP** ⊆ **P**, so **P** = **NP**. ■



The Tantalizing Truth

Theorem: If *any* **NP**-complete language is not in **P**, then **P** ≠ **NP**.

Proof: Suppose that L is an **NP**-complete language not in **P**. Since L is **NP**-complete, we know that $L \in$ **NP**. Therefore, we know that $L \in$ **NP** and $L \notin$ **P**, so **P** ≠ **NP**. ■



How do we even know NP-complete problems exist in the first place?

Satisfiability

- A propositional logic formula φ is called ***satisfiable*** if there is some assignment to its variables that makes it evaluate to true.
  - $p \land q$ is satisfiable.
  - $p \land \neg p$ is unsatisfiable.
  - $p \to (q \land \neg q)$ is satisfiable.
- An assignment of true and false to the variables of φ that makes it evaluate to true is called a ***satisfying assignment***.

SAT

- The ***boolean satisfiability problem*** (***SAT***) is the following:

Given a propositional logic formula φ, is φ satisfiable?

- Formally:

***SAT* = { ⟨φ⟩ | φ is a satisfiable PL formula }**

Theorem (Cook-Levin): SAT is **NP**-complete.

Proof: Read Sipser or take CS154!

New Stuff!

A Simpler **NP**-Complete Problem

Literals and Clauses

- A ***literal*** in propositional logic is a variable or its negation:
  - x
  - $\neg y$
  - But not $x \land y$.
- A ***clause*** is a many-way OR (*disjunction*) of literals.
  - $(\neg x \lor y \lor \neg z)$
  - (x)
  - But not $x \lor \neg(y \lor z)$

Conjunctive Normal Form

- A propositional logic formula φ is in ***conjunctive normal form*** (***CNF***) if it is the many-way AND (*conjunction*) of clauses.
    - $(x \lor y \lor z) \land (\neg x \lor \neg y) \land (x \lor y \lor z \lor \neg w)$
    - $(x \lor z)$
    - But not $(x \lor (y \land z)) \lor (x \lor y)$
- Only legal operators are ¬, ∨, ∧.
- No nesting allowed.

The Structure of CNF

$$(\underbrace{x \lor y \lor \neg z}) \land (\underbrace{\neg x \lor \neg y \lor z}) \land (\underbrace{\neg x \lor y \lor \neg z})$$

For this formula to be satisfiable, each clause must have <u>at least</u> one true literal in it.

The Structure of CNF

$$\underbrace{(\,x \lor y \lor \boxed{\neg z}\,)}\land\underbrace{(\,\neg x \lor \boxed{\neg y} \lor z\,)}\land\underbrace{(\,\neg x \lor y \lor \boxed{\neg z}\,)}$$

We should pick at least one true literal from each clause…

The Structure of CNF

$$\underbrace{(\ x \lor y \lor \neg z\ )}_{} \land \underbrace{(\ \neg x \lor \neg y \lor z\ )}_{} \land \underbrace{(\ \neg x \lor y \lor \neg z\ )}_{}$$

… but never choose a literal and its negation

3-CNF

- A propositional formula is in ***3-CNF*** if
  - it is in CNF, and
  - every clause has *exactly* three literals.
- For example:
  - $(x \lor y \lor z) \land (\neg x \lor \neg y \lor z)$
  - $(x \lor x \lor x) \land (y \lor \neg y \lor \neg x) \land (x \lor y \lor \neg y)$
  - but not $(x \lor y \lor z \lor w) \land (x \lor y)$
- The language ***3SAT*** is defined as follows:

3SAT = { ⟨φ⟩ | φ is a satisfiable 3-CNF formula }

Theorem: 3SAT is **NP**-Complete

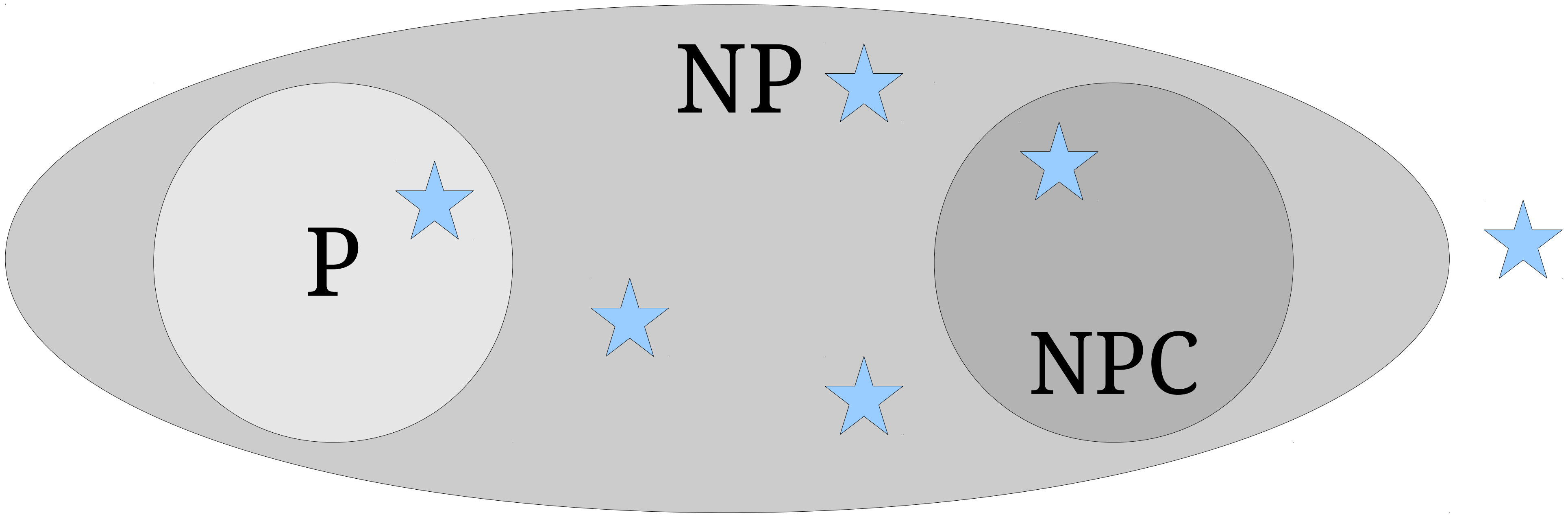
Finding Additional **NP**-Complete Problems

NP-Completeness

Theorem: Let A and B be languages. If $A \leq_{\mathrm{P}} B$ and A is **NP**-hard, then B is **NP**-hard.

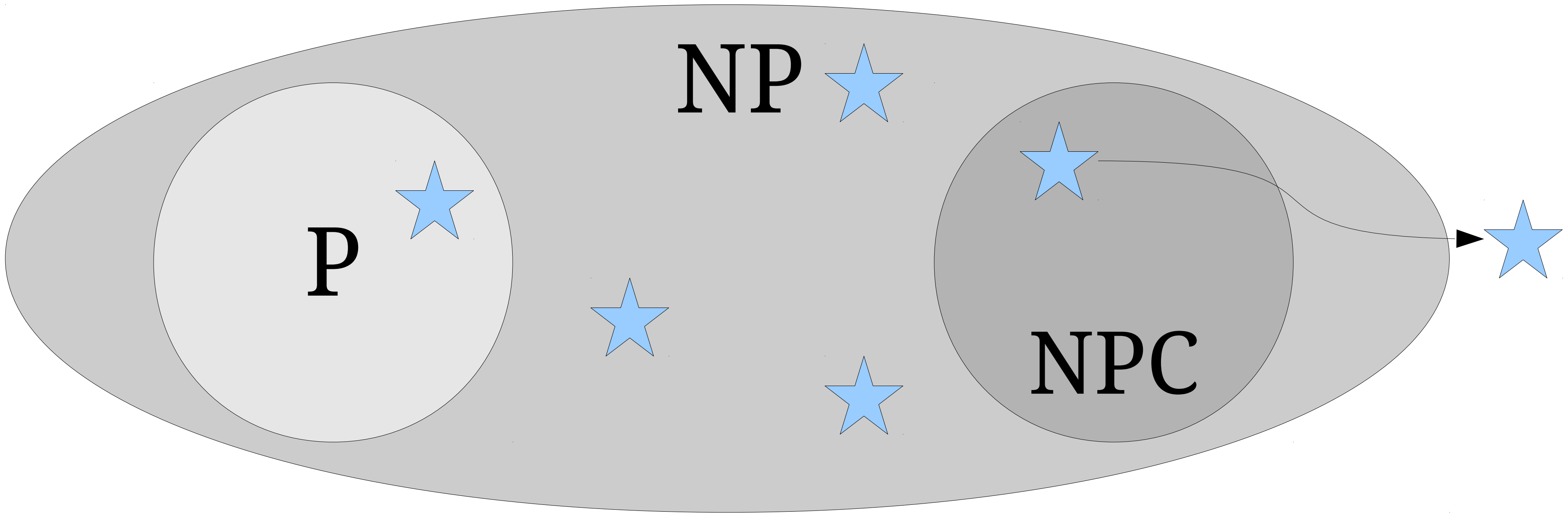
NP-Completeness

Theorem: Let A and B be languages. If $A \leq_P B$ and A is **NP**-hard, then B is **NP**-hard.



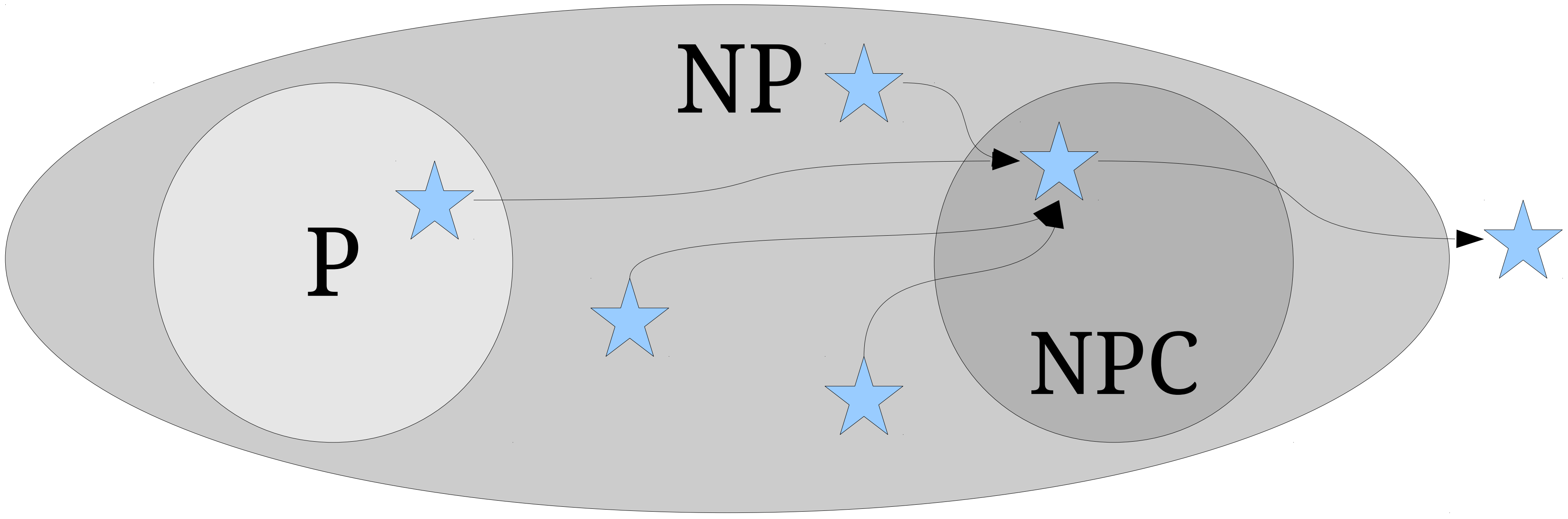
NP-Completeness

Theorem: Let A and B be languages. If $A \leq_{\text{P}} B$ and A is **NP**-hard, then B is **NP**-hard.



NP-Completeness

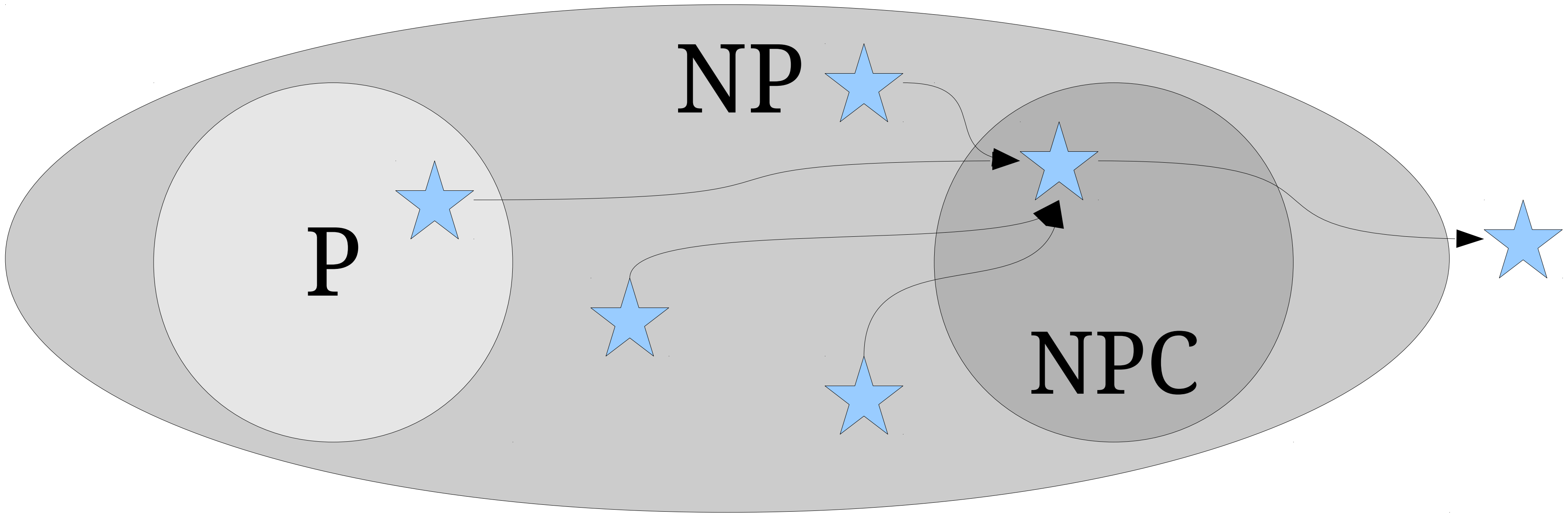
Theorem: Let A and B be languages. If $A \leq_P B$ and A is **NP**-hard, then B is **NP**-hard.



NP-Completeness

Theorem: Let A and B be languages. If $A \leq_P B$ and A is **NP**-hard, then B is **NP**-hard.

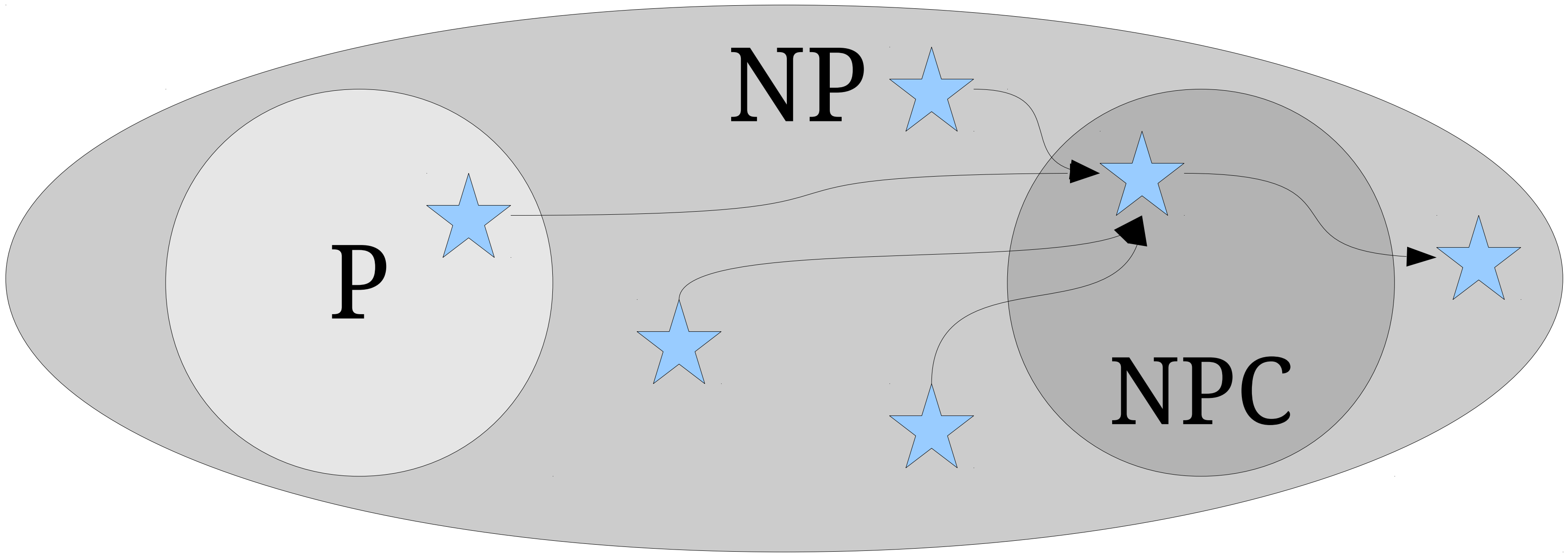
Theorem: Let A and B be languages where $A \in$ **NPC** and $B \in$ **NP**. If $A \leq_P B$, then $B \in$ **NPC**.



NP-Completeness

Theorem: Let A and B be languages. If $A \leq_P B$ and A is **NP**-hard, then B is **NP**-hard.

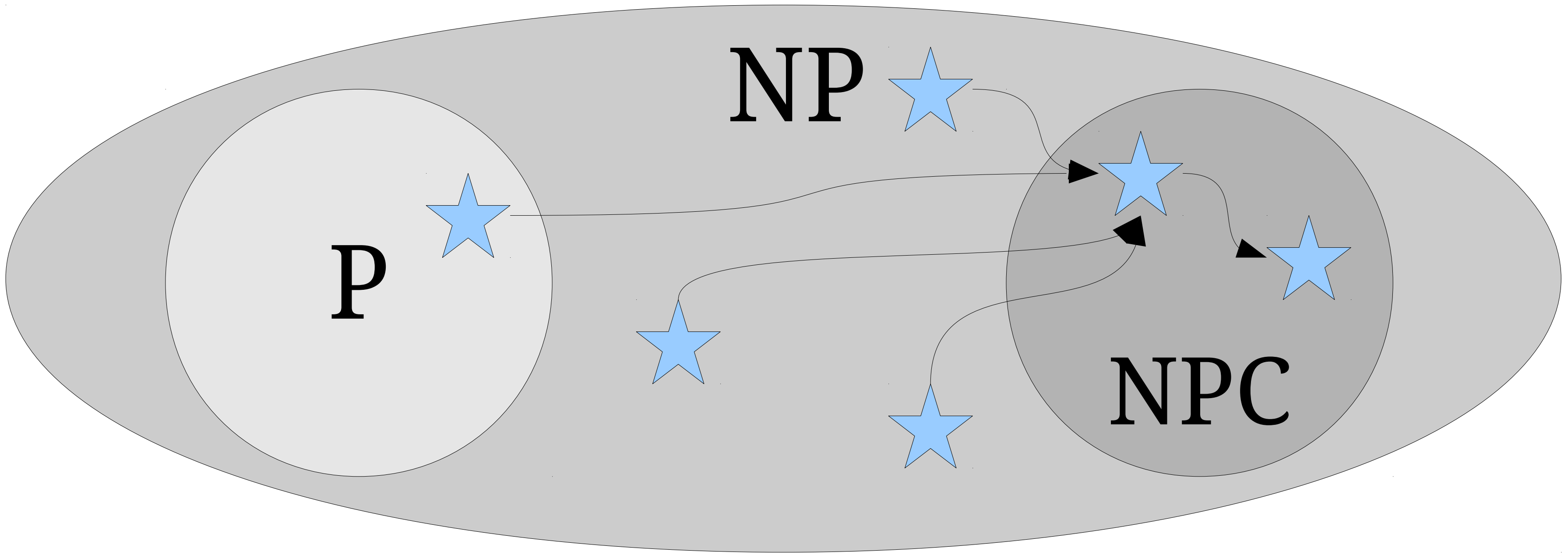
Theorem: Let A and B be languages where $A \in$ **NPC** and $B \in$ **NP**. If $A \leq_P B$, then $B \in$ **NPC**.



NP-Completeness

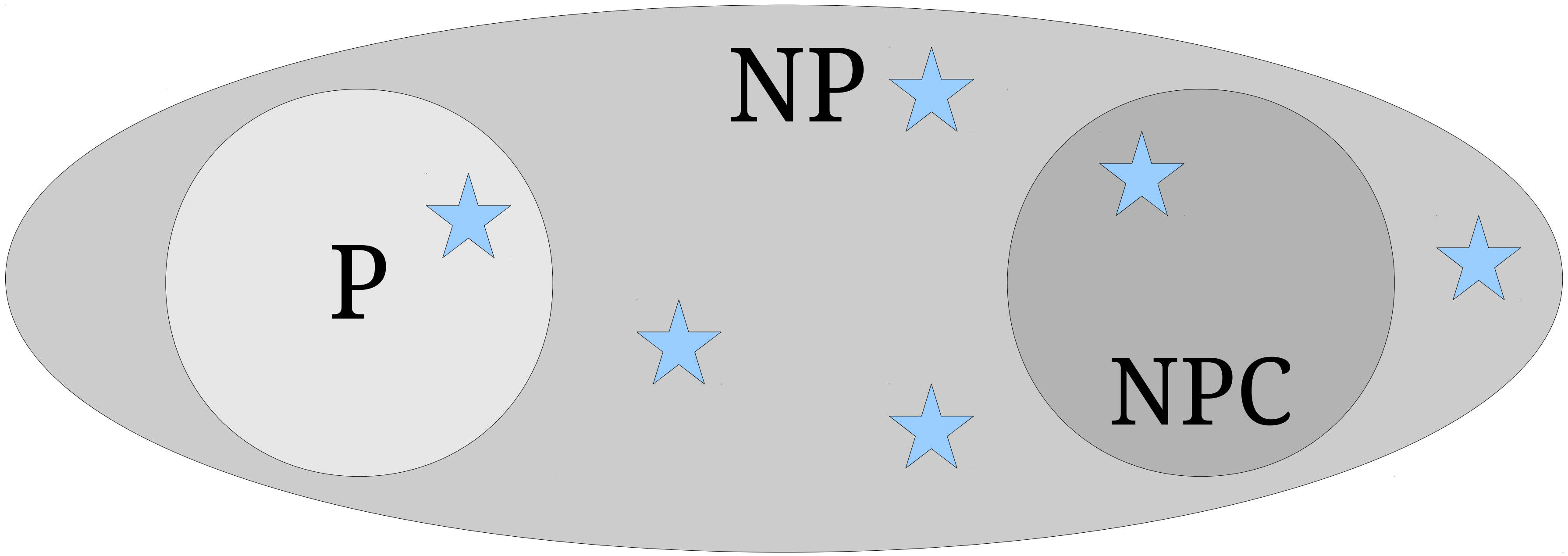
Theorem: Let A and B be languages. If $A \leq_P B$ and A is **NP**-hard, then B is **NP**-hard.

Theorem: Let A and B be languages where $A \in$ **NPC** and $B \in$ **NP**. If $A \leq_P B$, then $B \in$ **NPC**.



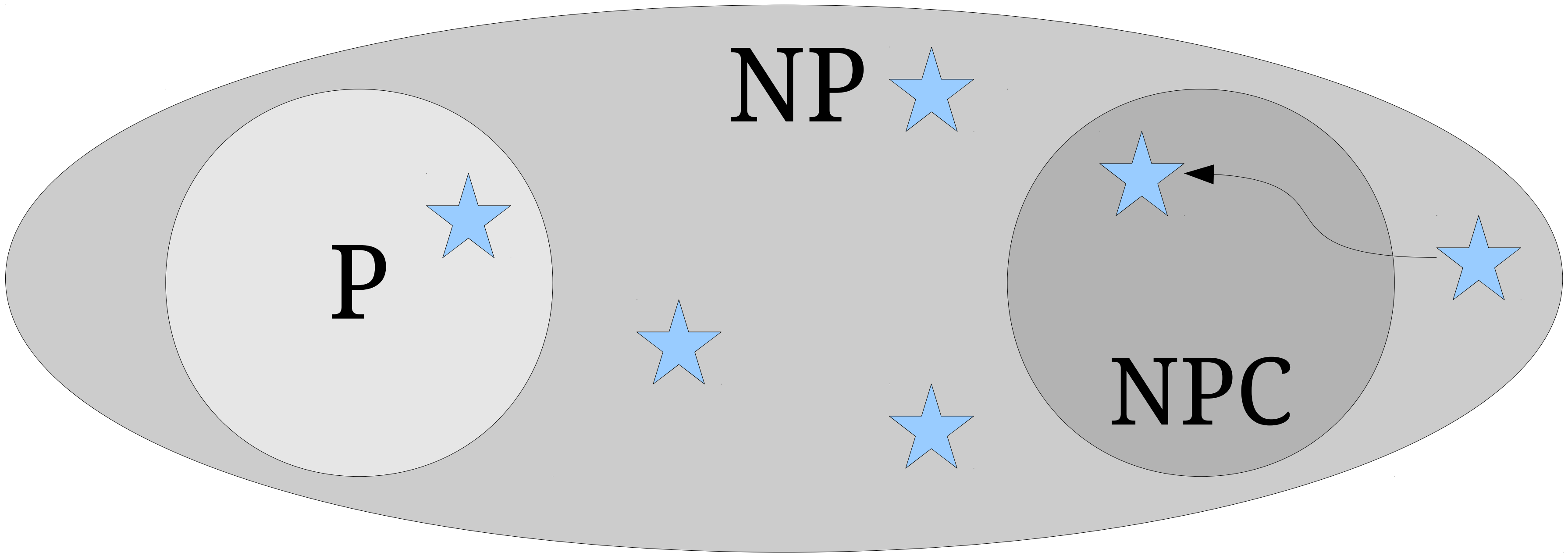
Be Careful!

- To prove that some language L is **NP**-complete, show that $L \in$ **NP**, then reduce some known **NP**-complete problem to L.
- **Do not** reduce L to a known **NP**-complete problem.
  - We already knew you could do this; *every* **NP** problem is reducible to any **NP**-complete problem!



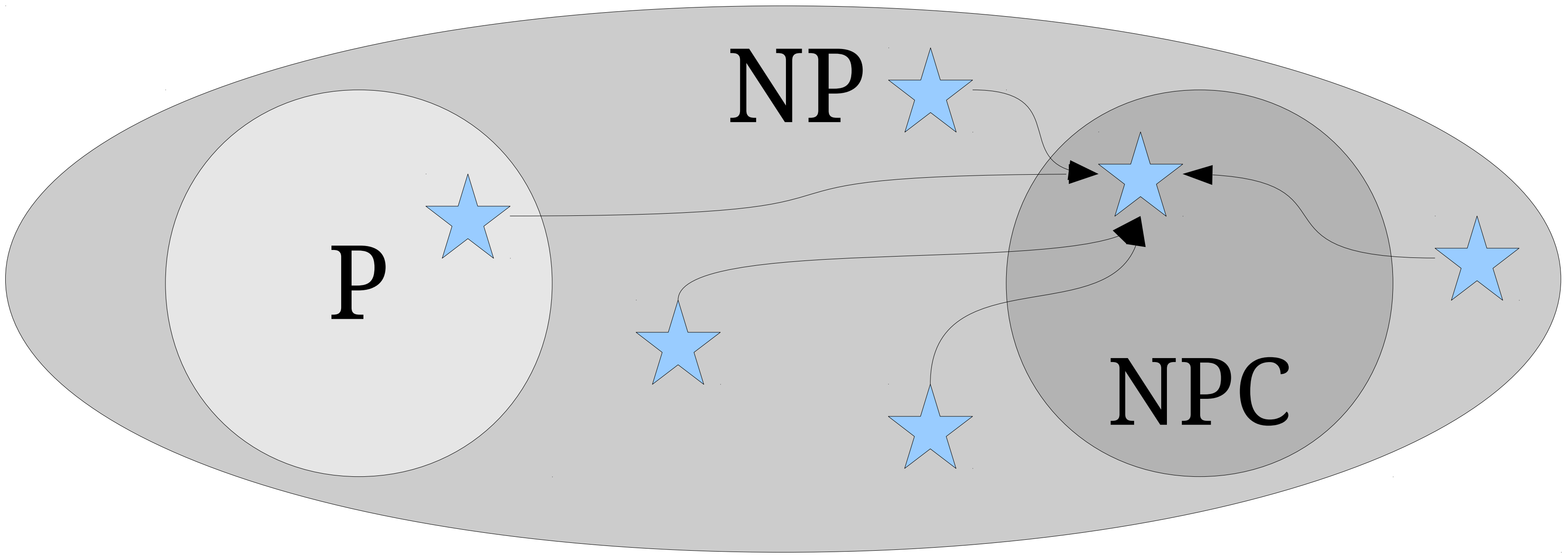
Be Careful!

- To prove that some language L is **NP**-complete, show that $L \in$ **NP**, then reduce some known **NP**-complete problem to L.
- **Do not** reduce L to a known **NP**-complete problem.
    - We already knew you could do this; *every* **NP** problem is reducible to any **NP**-complete problem!



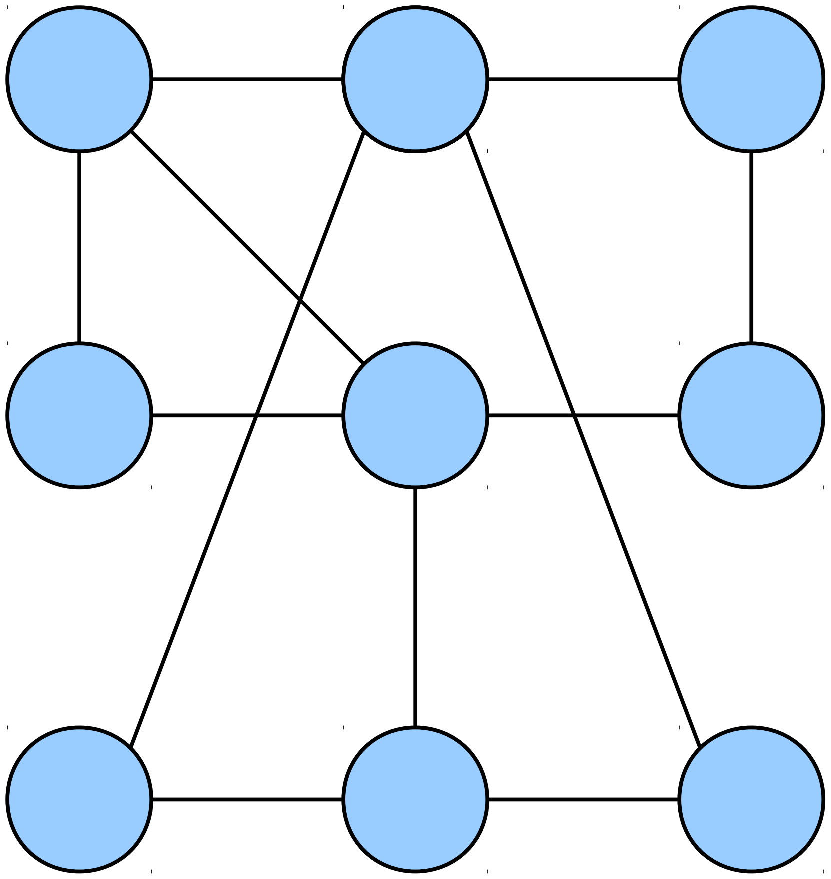
Be Careful!

- To prove that some language L is **NP**-complete, show that $L \in$ **NP**, then reduce some known **NP**-complete problem to L.
- **Do not** reduce L to a known **NP**-complete problem.
    - We already knew you could do this; *every* **NP** problem is reducible to any **NP**-complete problem!

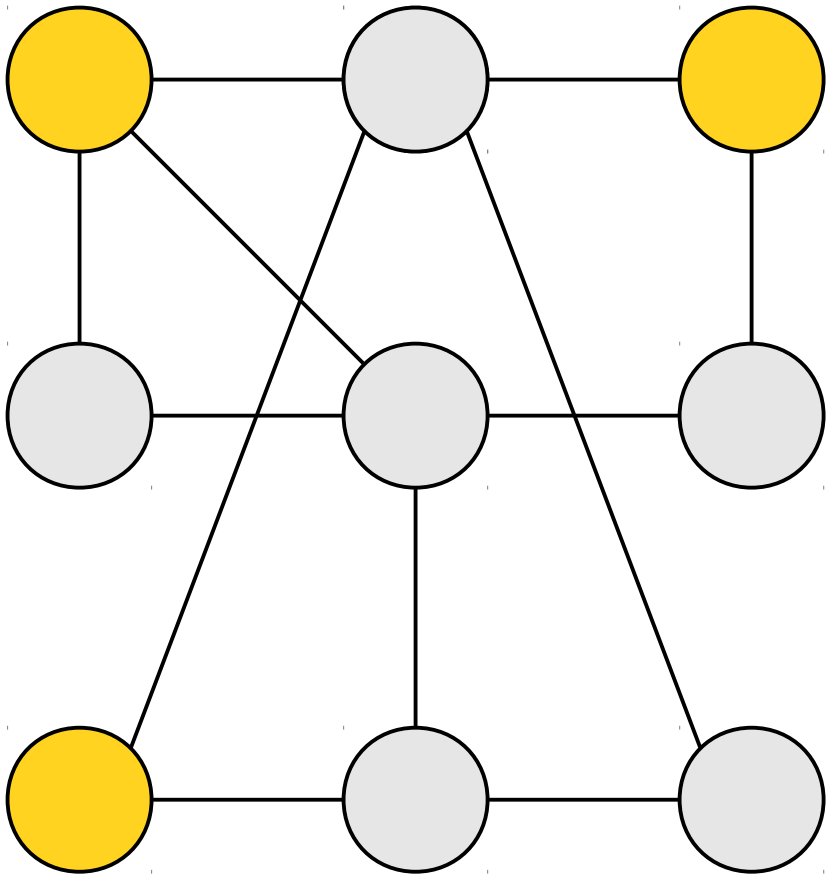


So what other problems are **NP**-complete?

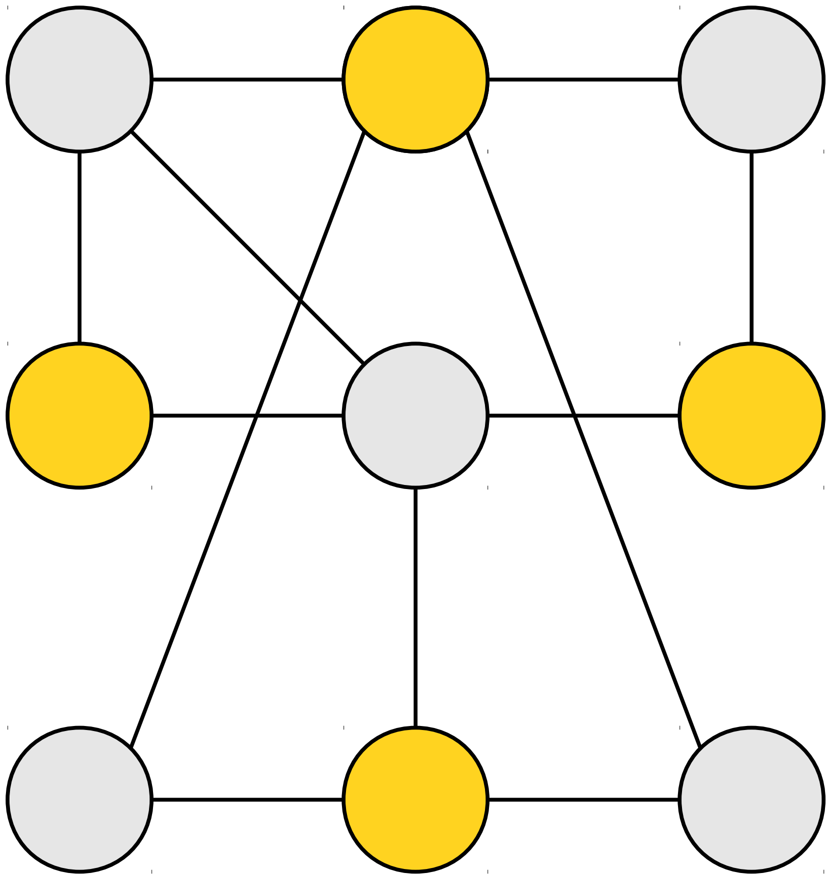
An ***independent set*** in an undirected graph
is a set of nodes that have no edges between them.



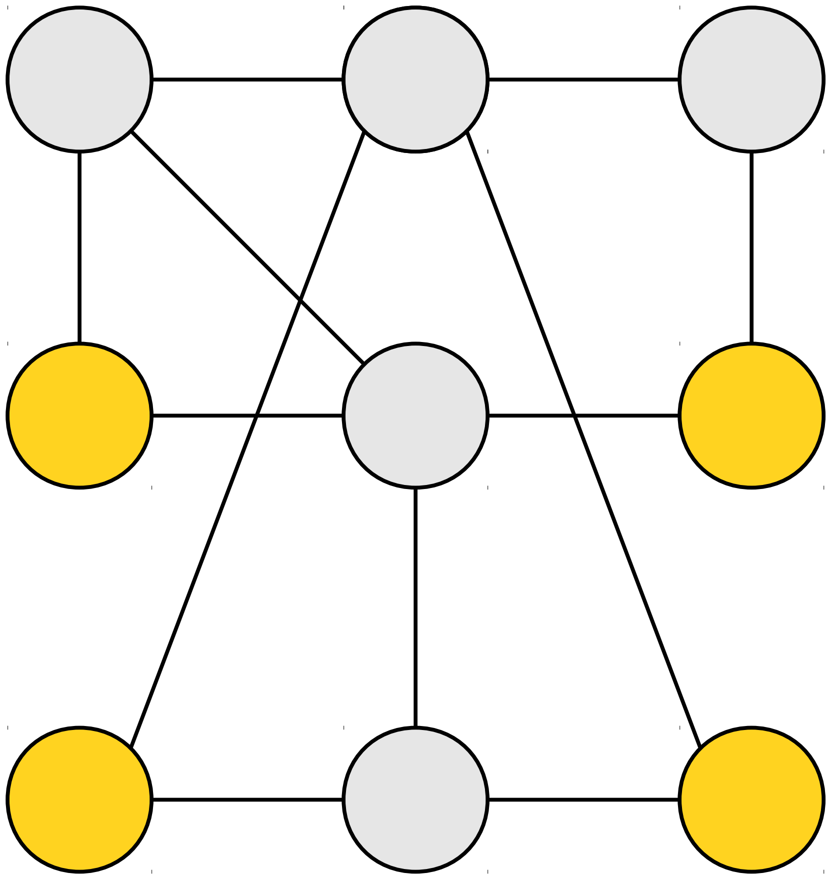
An ***independent set*** in an undirected graph is a set of nodes that have no edges between them.



An ***independent set*** in an undirected graph is a set of nodes that have no edges between them.



An ***independent set*** in an undirected graph
is a set of nodes that have no edges between them.



An ***independent set*** in an undirected graph is a set of nodes that have no edges between them.

The Independent Set Problem

- Given an undirected graph *G* and a natural number *n*, the ***independent set problem*** is

Does *G* contain an independent set of size at least *n*?

- As a formal language:

INDSET = { ⟨*G*, *n*⟩ | *G* is an undirected graph with an independent set of size at least *n* }

$INDSET \in \mathbf{NP}$

- The independent set problem is in **NP**.
- Here is a polynomial-time verifier that checks whether S is an n-element independent set:

```
V = "On input ⟨G, n, S⟩, where G is a graph,
          n ∈ ℕ, and S is a set of nodes in G:
               If |S| < n, reject.
               For each edge in G, if both
               endpoints are in S, reject.
               Otherwise, accept."
```

INDSET ∈ **NPC**

- The *INDSET* problem is **NP**-complete.
- To prove this, we will find a polynomial-time reduction from 3SAT to *INDSET*.
- ***Goal:*** Given a 3CNF formula φ, build a graph *G* and number *n* such that φ is satisfiable iff *G* has an independent set of size *n*.
- How can we accomplish this?

The Structure of 3CNF

$$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$$

The Structure of 3CNF

$$(\underbrace{x \lor y \lor \neg z}) \land (\underbrace{\neg x \lor \neg y \lor z}) \land (\underbrace{\neg x \lor y \lor \neg z})$$

The Structure of 3CNF

$$\underbrace{(\, x \lor y \lor \neg z \,)} \land \underbrace{(\, \neg x \lor \neg y \lor z \,)} \land \underbrace{(\, \neg x \lor y \lor \neg z \,)}$$

$$\boxed{\text{Each clause must have } \underline{\text{at least}} \text{ one true literal in it.}}$$

The Structure of 3CNF

$$(\underbrace{x \lor y \lor \neg z}) \land (\underbrace{\neg x \lor \neg y \lor z}) \land (\underbrace{\neg x \lor y \lor \neg z})$$

The Structure of 3CNF

$$\underbrace{(\ x \lor y \lor \neg z\ )} \land \underbrace{(\ \neg x \lor \neg y \lor z\ )} \land \underbrace{(\ \neg x \lor y \lor \neg z\ )}$$

We should pick at least one true literal from each clause

The Structure of 3CNF

$$\underbrace{(\; x \lor \boxed{y} \lor \neg z\;)}_{} \land \underbrace{(\;\boxed{\neg x} \lor \neg y \lor z\;)}_{} \land \underbrace{(\;\boxed{\neg x} \lor y \lor \neg z\;)}_{}$$

The Structure of 3CNF

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$

The Structure of 3CNF

$$\underbrace{(\ x \lor y \lor \neg z\ )} \land \underbrace{(\ \neg x \lor \neg y \lor z\ )} \land \underbrace{(\ \neg x \lor y \lor \neg z\ )}$$

The Structure of 3CNF

$$(\underbrace{x \lor y \lor \neg z}) \land (\underbrace{\neg x \lor \neg y \lor z}) \land (\underbrace{\neg x \lor y \lor \neg z})$$

The Structure of 3CNF

$$\underbrace{(\ x \lor y \lor \neg z\ )}\ \land\ \underbrace{(\ \neg x \lor \neg y \lor z\ )}\ \land\ \underbrace{(\ \neg x \lor y \lor \neg z\ )}$$

… subject to the constraint that we never choose a literal and its negation

From 3SAT to INDSET

- To convert a 3SAT instance φ to an *INDSET* instance, we need to create a graph *G* and number *n* such that an independent set of size at least *n* in *G*
    - gives us a way to choose which literal in each clause of φ should be true,
    - doesn't simultaneously choose a literal and its negation, and
    - has size polynomially large in the length of the formula φ.

From 3SAT to INDSET

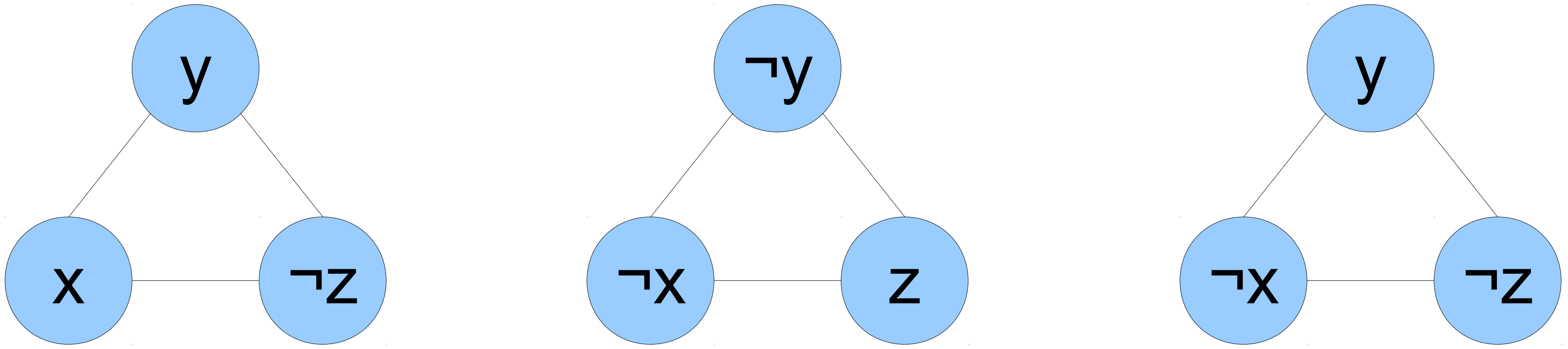
$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$

From 3SAT to INDSET

$$(\underbrace{x \lor y \lor \neg z}_{}) \land (\underbrace{\neg x \lor \neg y \lor z}_{}) \land (\underbrace{\neg x \lor y \lor \neg z}_{})$$

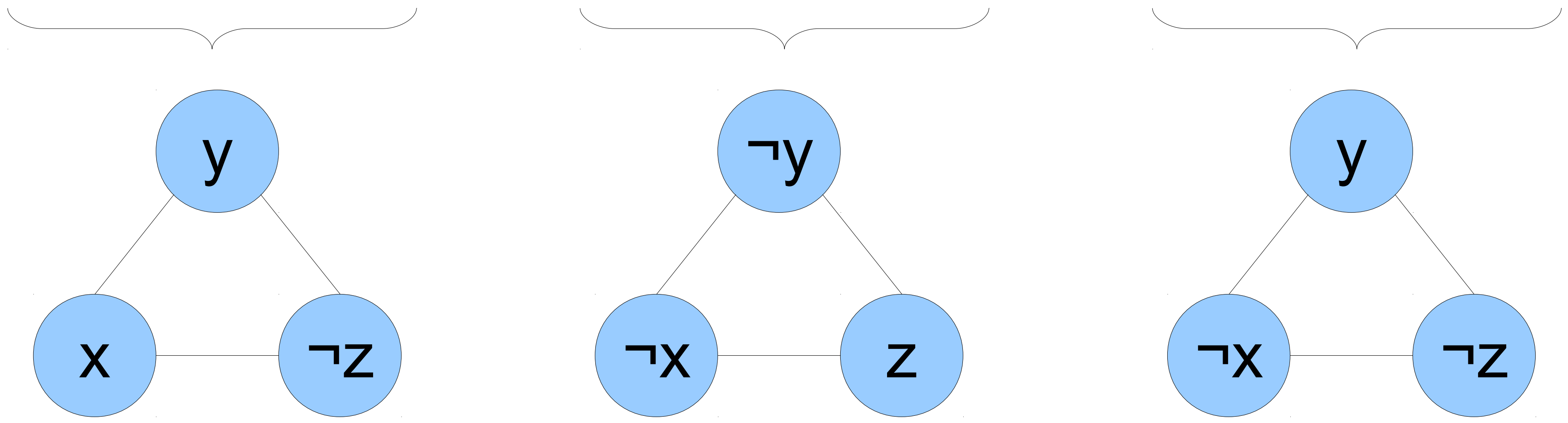
From 3SAT to INDSET

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



From 3SAT to INDSET

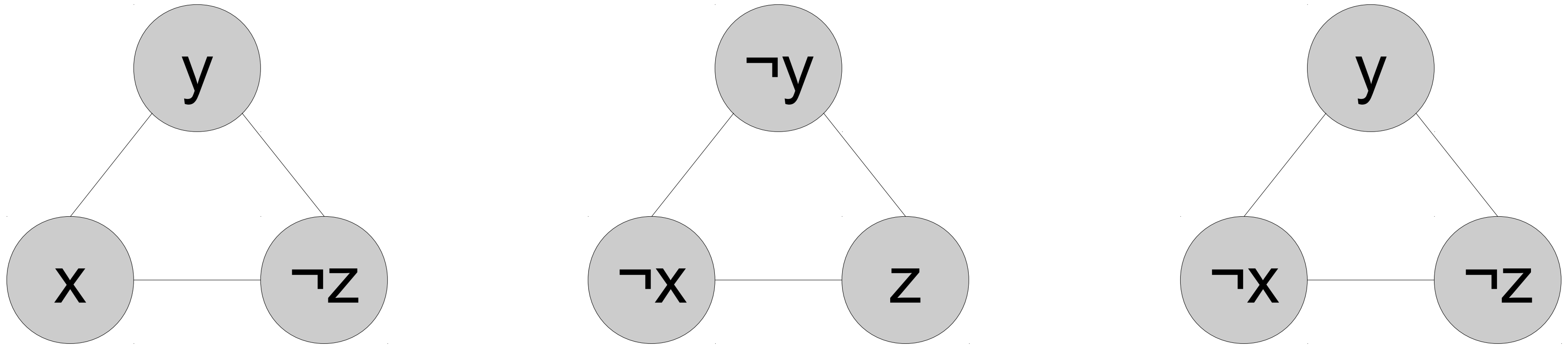
$$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$$



Any independent set in this graph chooses **exactly one** literal from each clause to be true.

From 3SAT to INDSET

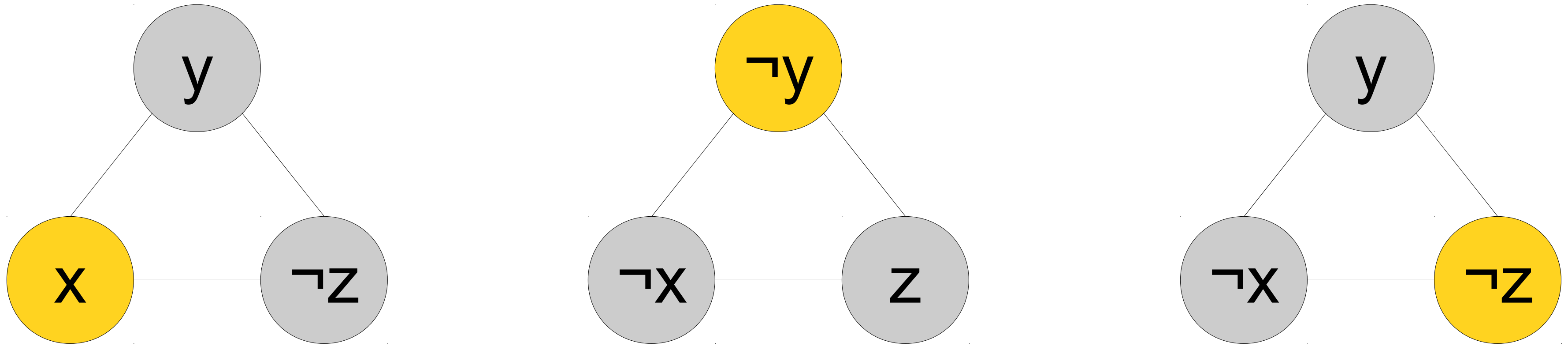
$$\underbrace{(x \lor y \lor \neg z)}\land\underbrace{(\neg x \lor \neg y \lor z)}\land\underbrace{(\neg x \lor y \lor \neg z)}$$



Any independent set in this graph chooses **exactly one** literal from each clause to be true.

From 3SAT to INDSET

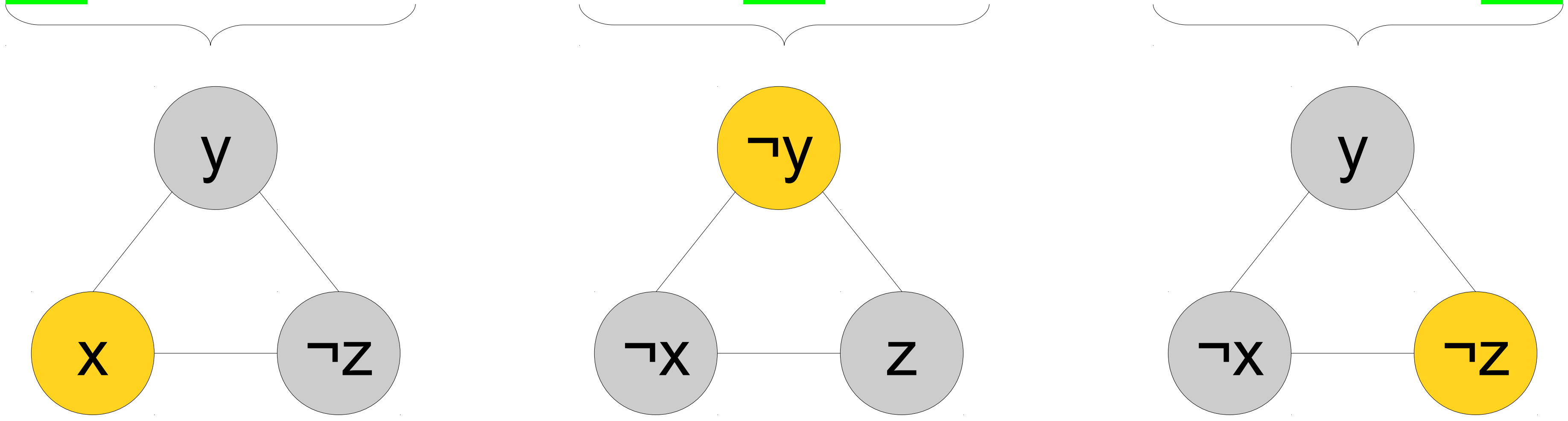
$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



Any independent set in this graph chooses exactly one literal from each clause to be true.

From 3SAT to INDSET

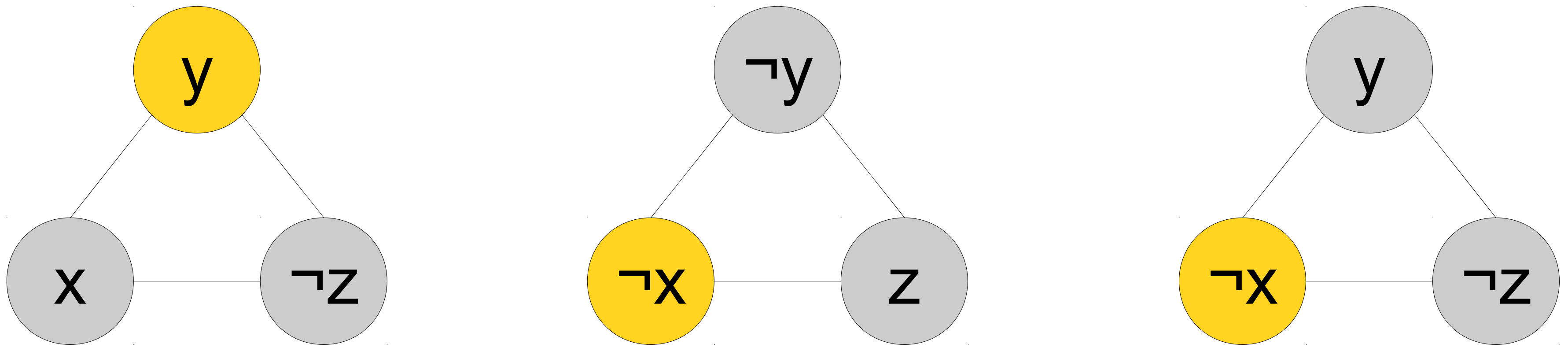
$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$



Any independent set in this graph chooses **exactly one** literal from each clause to be true.

From 3SAT to INDSET

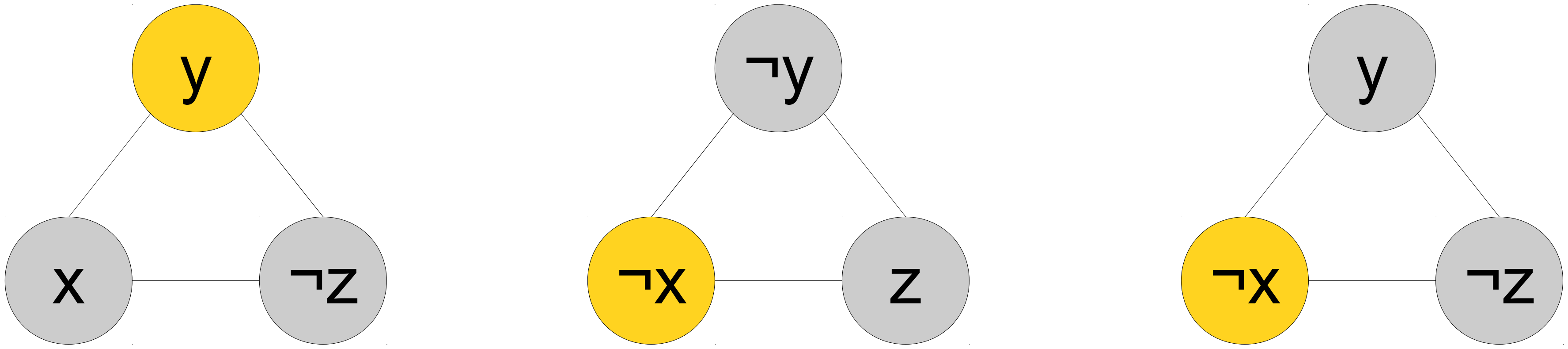
$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



Any independent set in this graph chooses exactly one literal from each clause to be true.

From 3SAT to INDSET

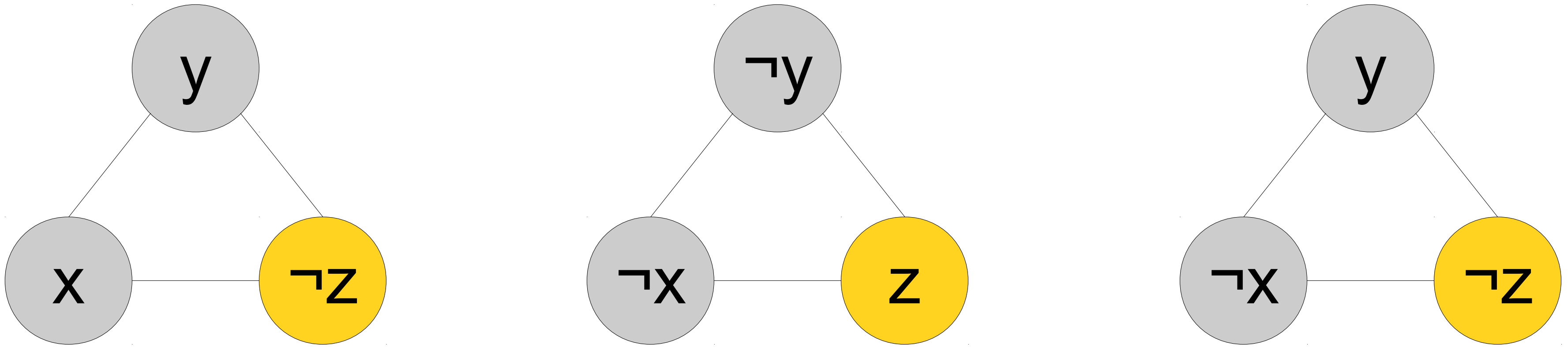
$$\underbrace{(\, x \lor y \lor \neg z \,)} \land \underbrace{(\, \neg x \lor \neg y \lor z \,)} \land \underbrace{(\, \neg x \lor y \lor \neg z \,)}$$



Any independent set in this graph chooses **exactly one** literal from each clause to be true.

From 3SAT to INDSET

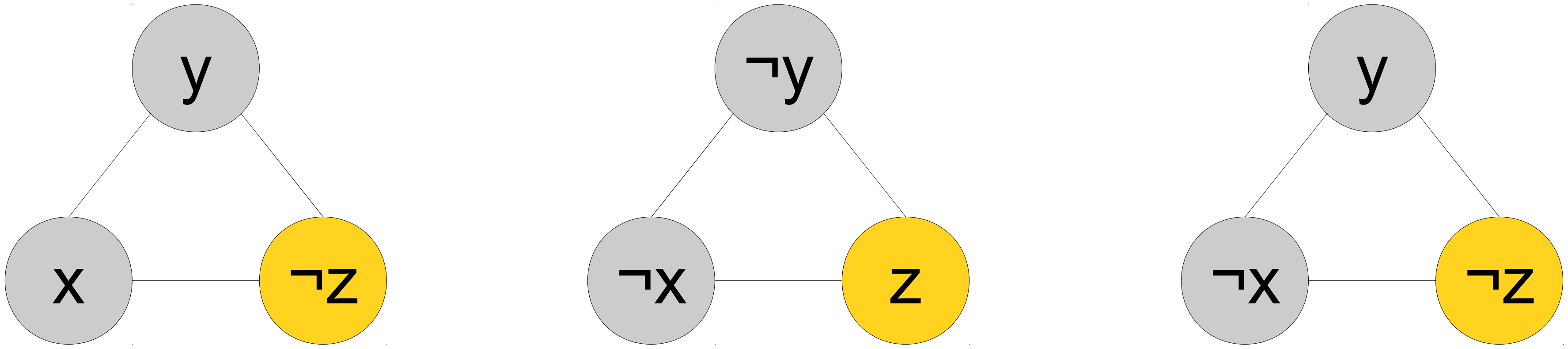
$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



Any independent set in this graph chooses **exactly one** literal from each clause to be true.

From 3SAT to INDSET

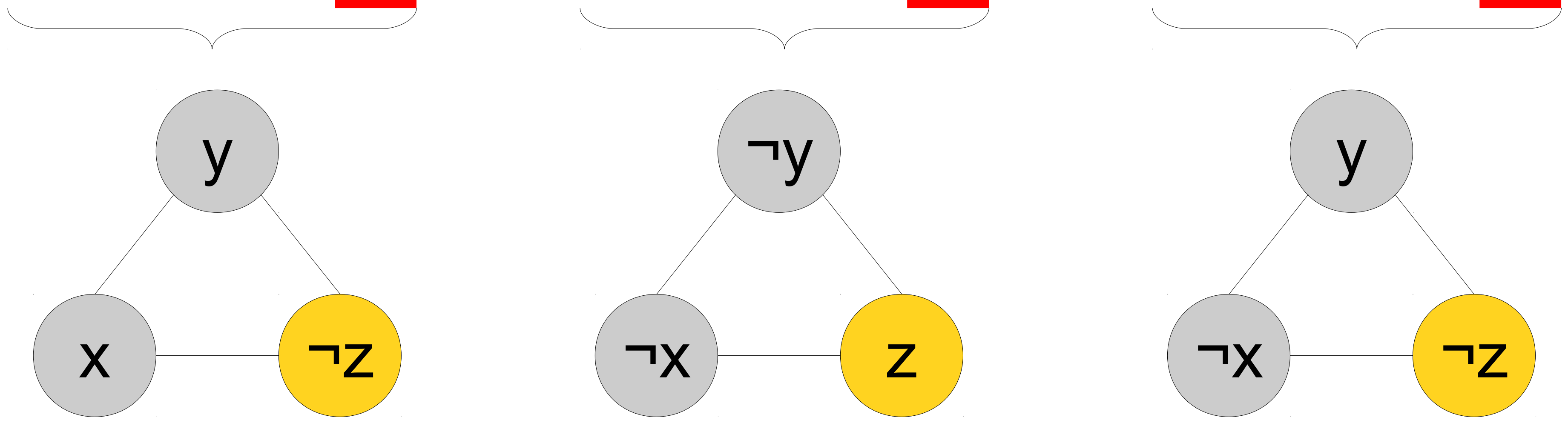
$$\underbrace{(\,x \lor y \lor \neg z\,)}\land\underbrace{(\,\neg x \lor \neg y \lor z\,)}\land\underbrace{(\,\neg x \lor y \lor \neg z\,)}$$



Any independent set in this graph chooses **exactly one** literal from each clause to be true.

From 3SAT to INDSET

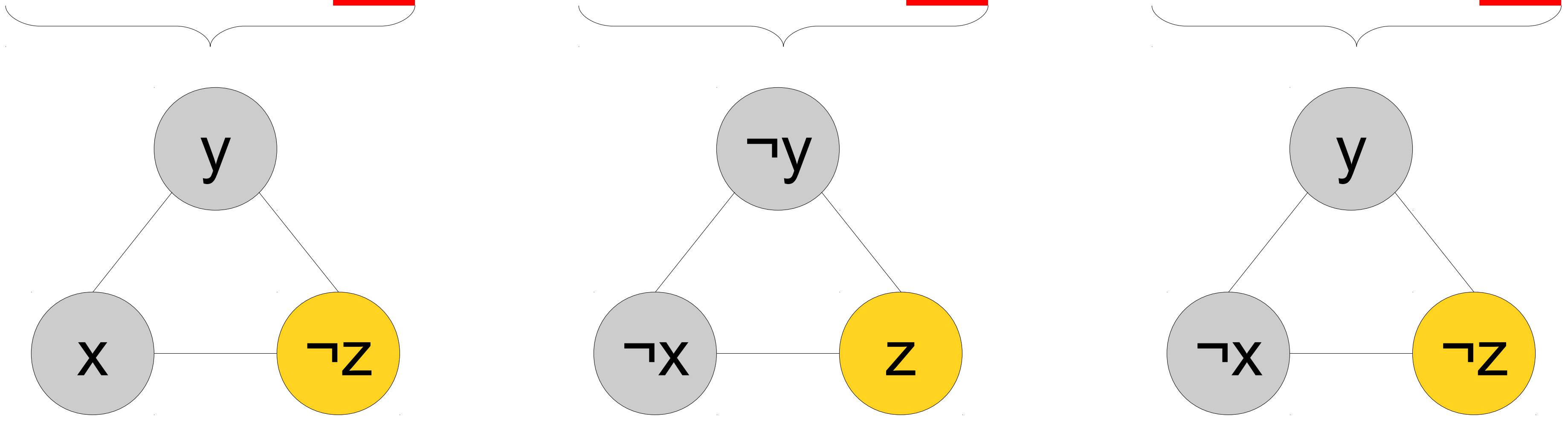
$$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$$



Any independent set in this graph chooses **exactly one** literal from each clause to be true.

From 3SAT to INDSET

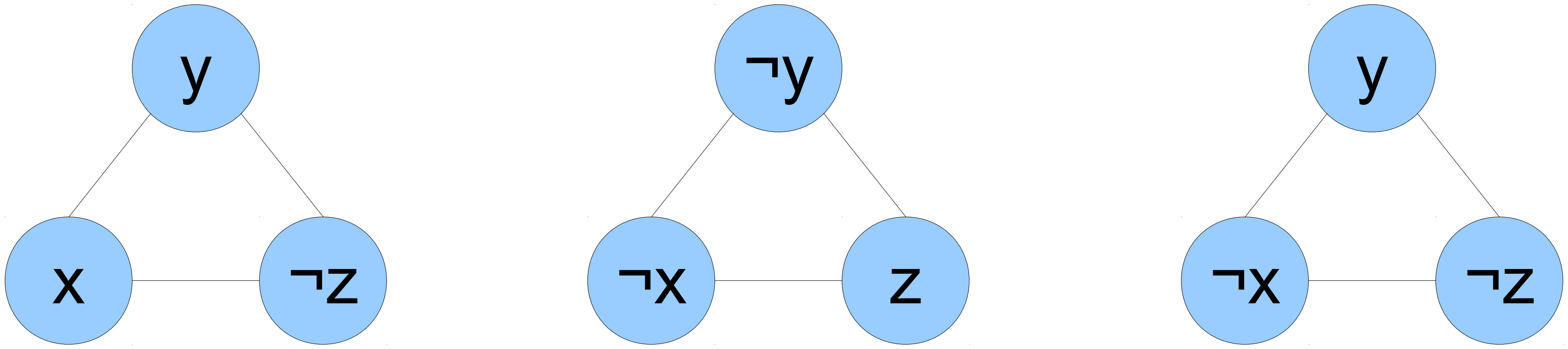
$$(x \lor y \lor \lnot z) \land (\lnot x \lor \lnot y \lor z) \land (\lnot x \lor y \lor \lnot z)$$



We need a way to ensure we never pick a literal and its negation.

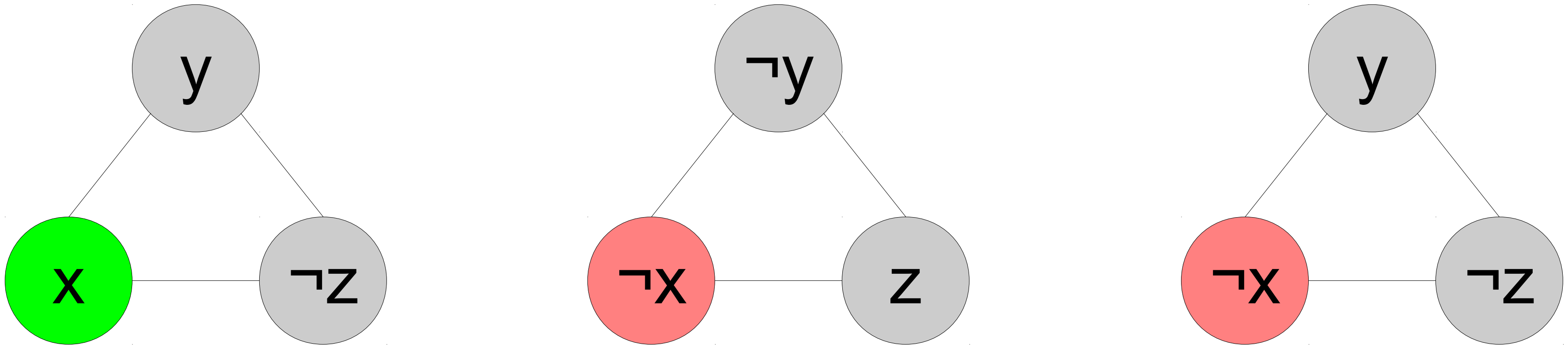
From 3SAT to INDSET

$$\underbrace{(\ x \lor y \lor \neg z\ )} \land \underbrace{(\ \neg x \lor \neg y \lor z\ )} \land \underbrace{(\ \neg x \lor y \lor \neg z\ )}$$

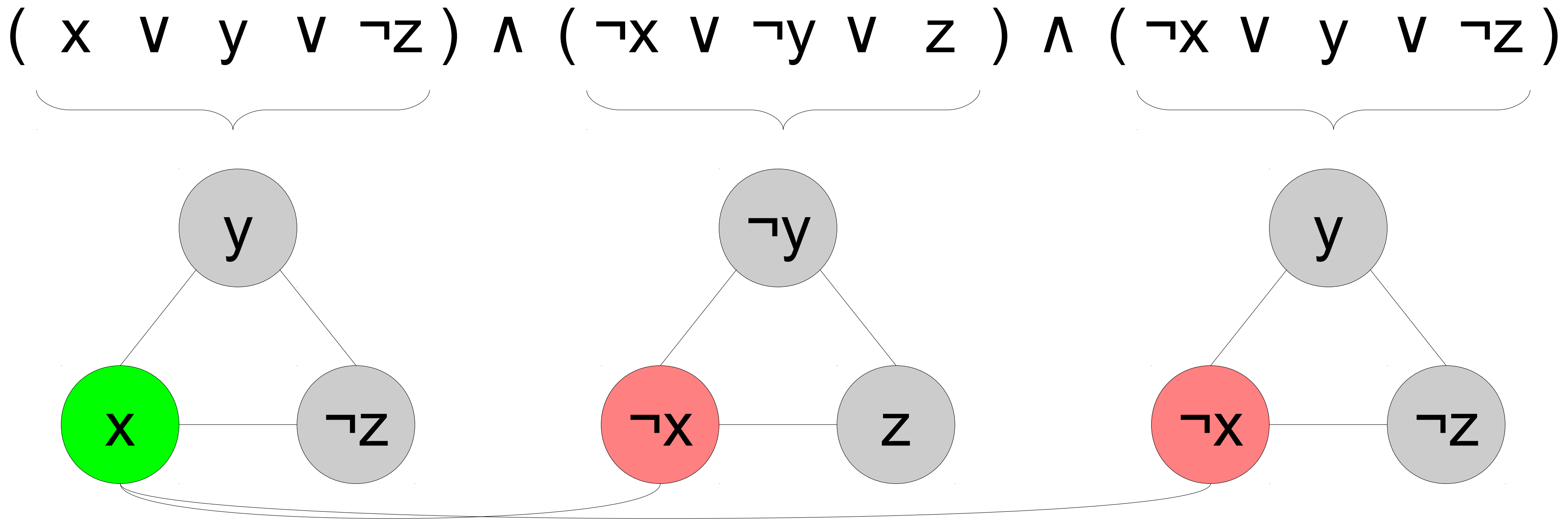


From 3SAT to INDSET

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$

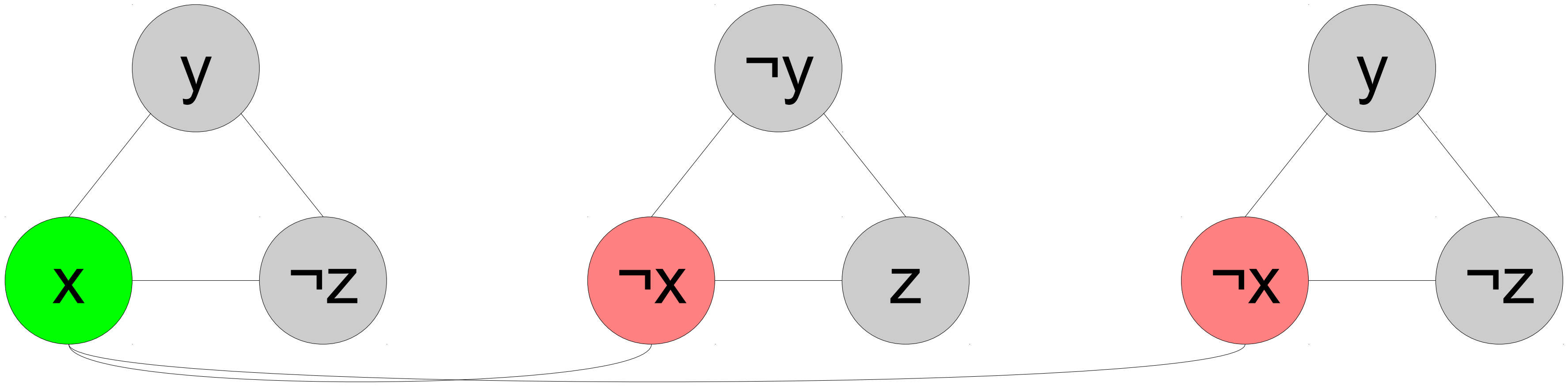


From 3SAT to INDSET



From 3SAT to INDSET

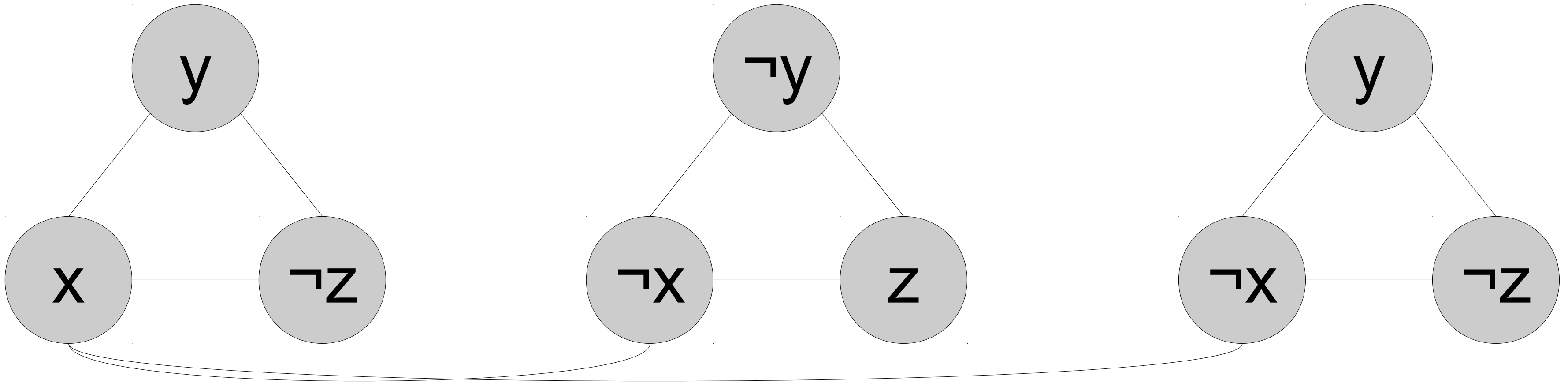
$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



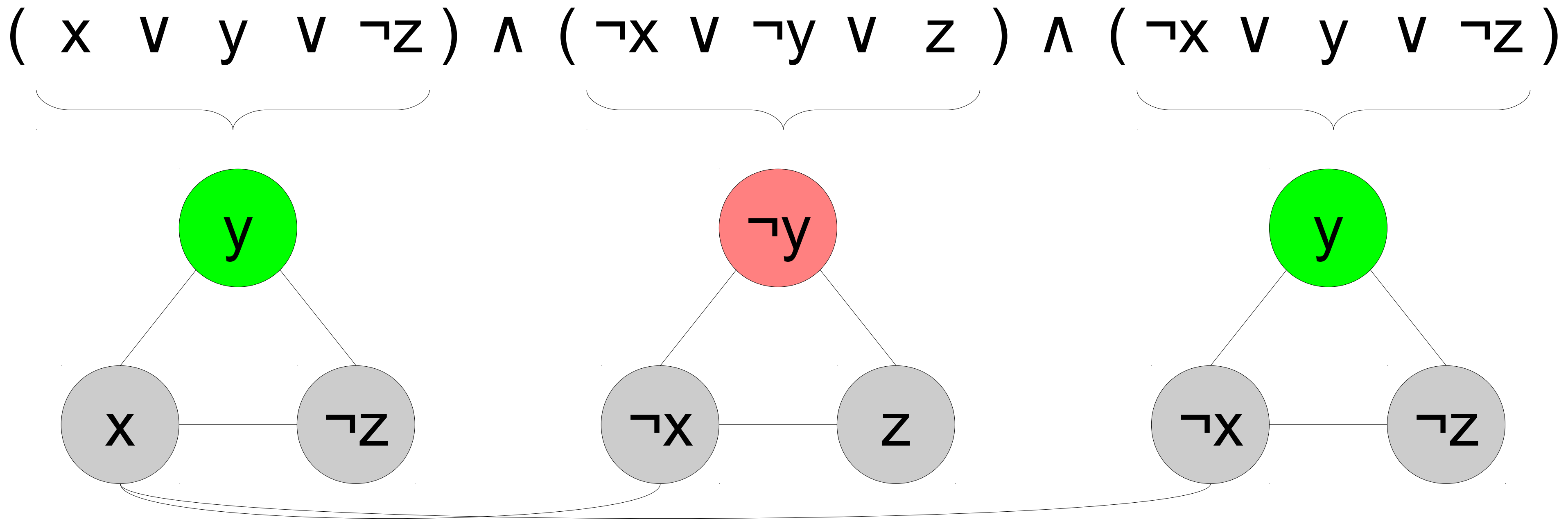
No independent set in this graph can choose two nodes labeled ***x*** and **¬*x***.

From 3SAT to INDSET

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$

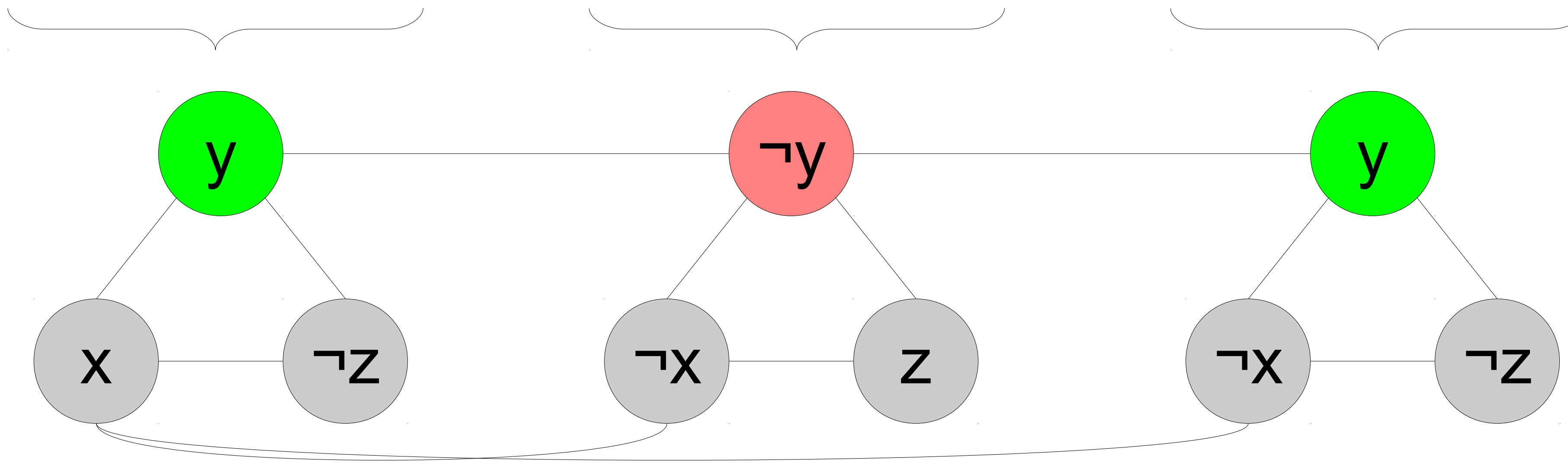


From 3SAT to INDSET



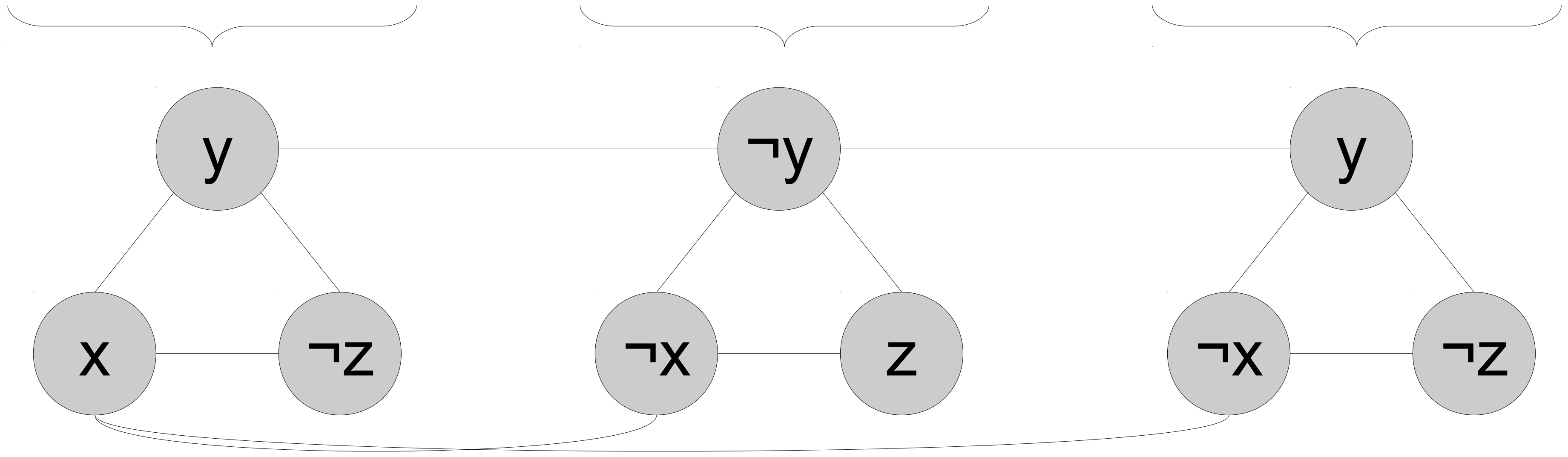
From 3SAT to INDSET

(x ∨ y ∨ ¬z) ∧ (¬x ∨ ¬y ∨ z) ∧ (¬x ∨ y ∨ ¬z)



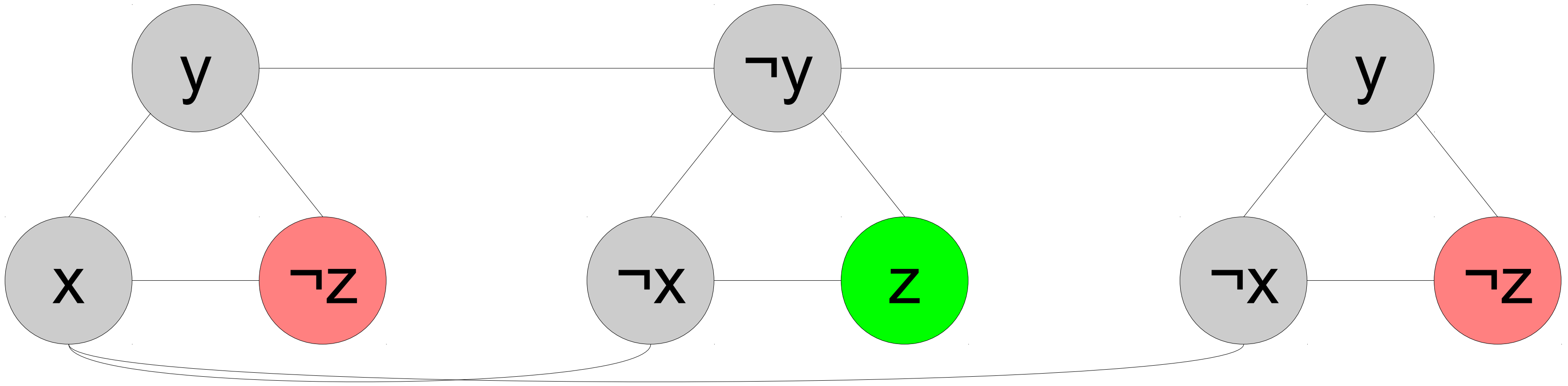
From 3SAT to INDSET

(x ∨ y ∨ ¬z) ∧ (¬x ∨ ¬y ∨ z) ∧ (¬x ∨ y ∨ ¬z)



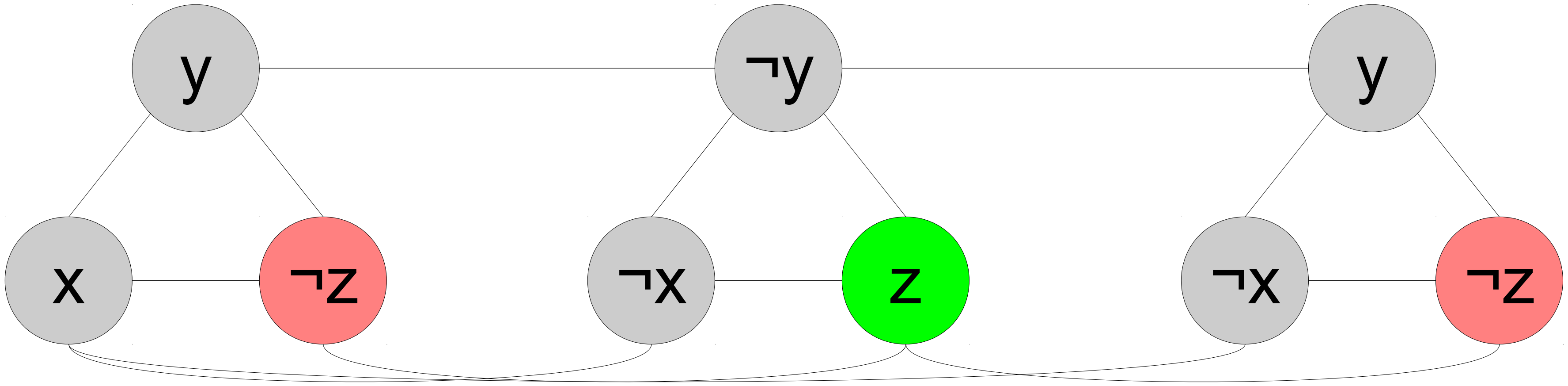
From 3SAT to INDSET

$$\underbrace{(\,x \lor y \lor \neg z\,)}\land\underbrace{(\,\neg x \lor \neg y \lor z\,)}\land\underbrace{(\,\neg x \lor y \lor \neg z\,)}$$



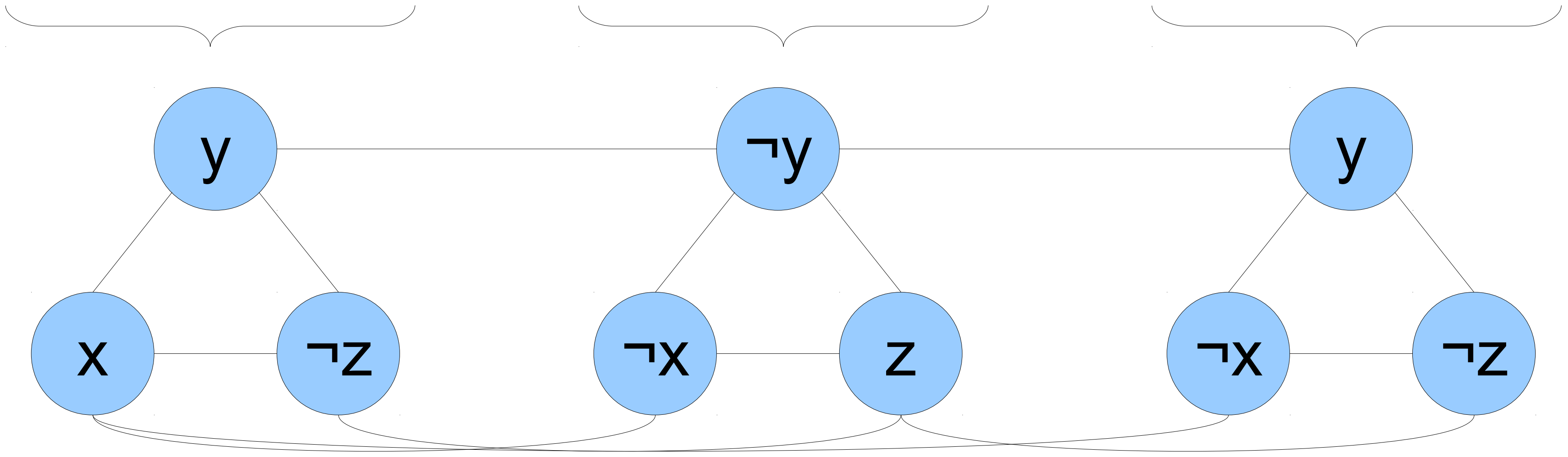
From 3SAT to INDSET

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



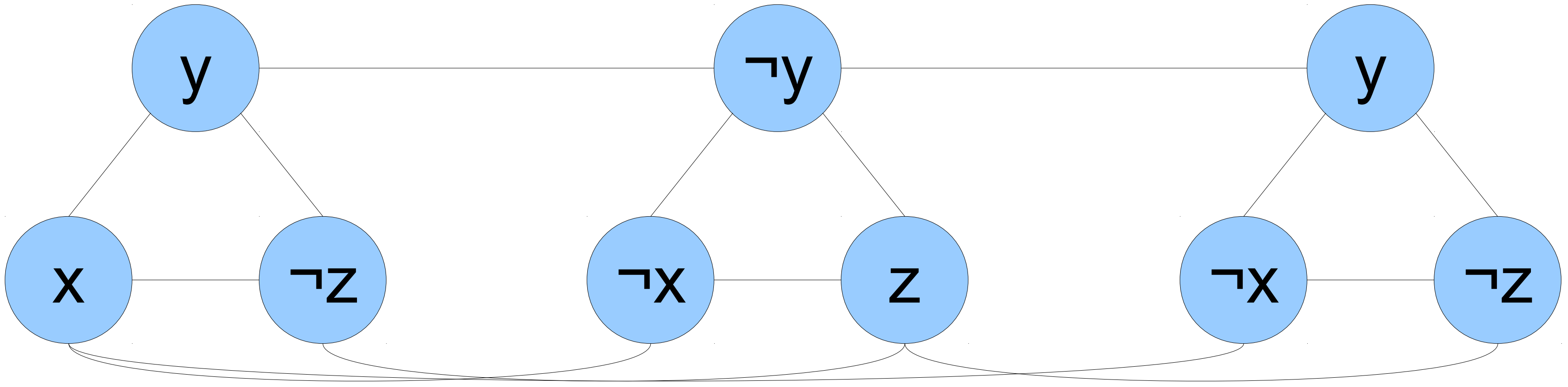
From 3SAT to INDSET

$$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$$



From 3SAT to INDSET

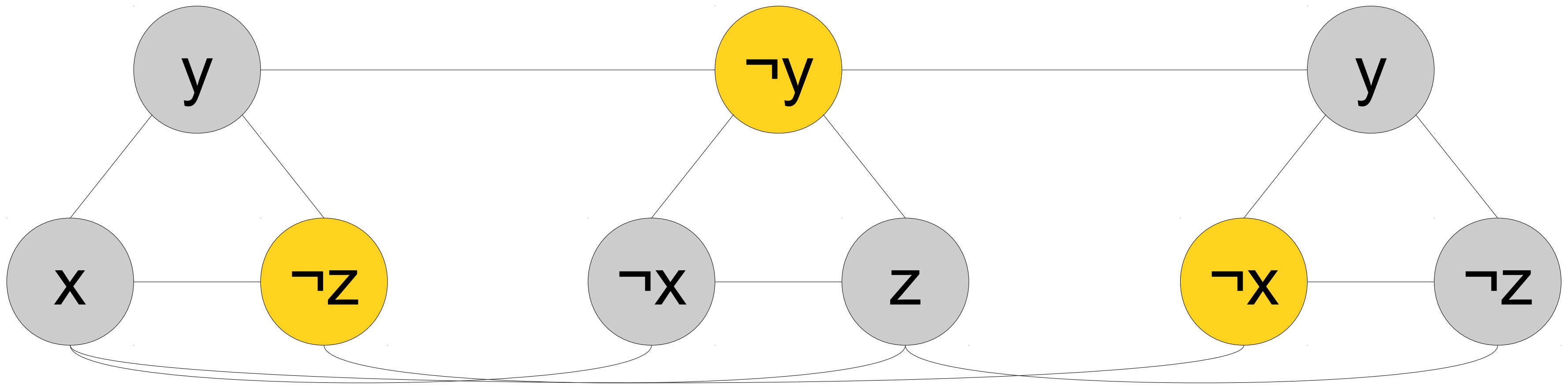
$$\underbrace{(x \lor y \lor \neg z)}_{} \land \underbrace{(\neg x \lor \neg y \lor z)}_{} \land \underbrace{(\neg x \lor y \lor \neg z)}_{}$$



If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET

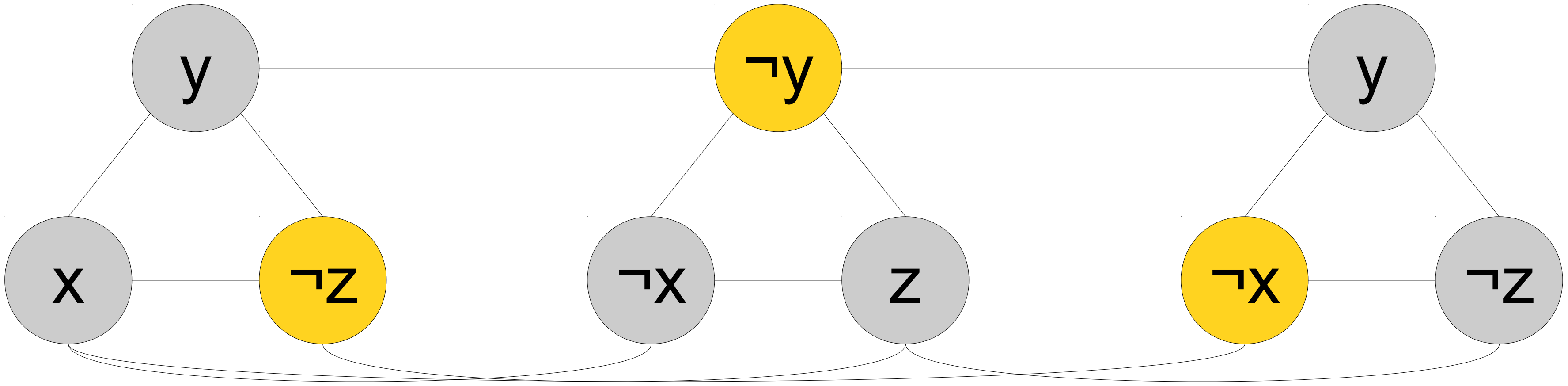
$$\underbrace{(x \lor y \lor \lnot z)} \land \underbrace{(\lnot x \lor \lnot y \lor z)} \land \underbrace{(\lnot x \lor y \lor \lnot z)}$$



If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET

$$(\underbrace{x \lor y \lor \neg z}) \land (\underbrace{\neg x \lor \neg y \lor z}) \land (\underbrace{\neg x \lor y \lor \neg z})$$

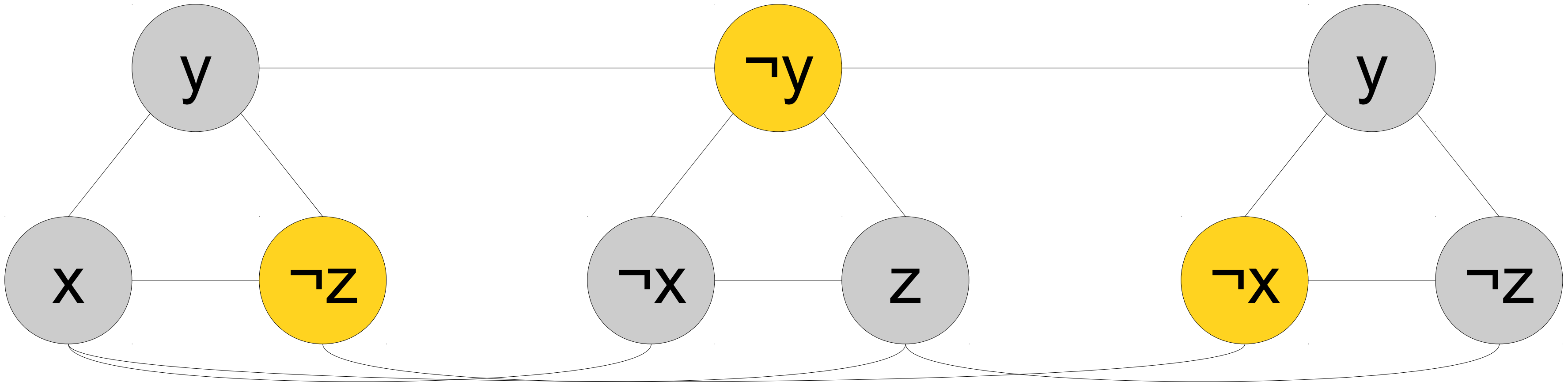


If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET

x = false, y = false, z = false.

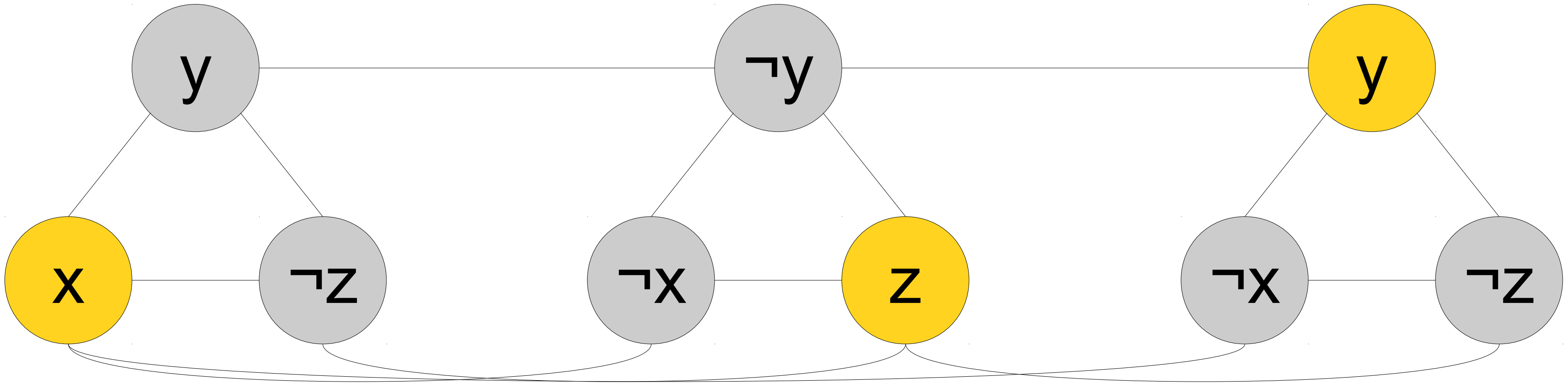
$$(\underbrace{x \lor y \lor \neg z}) \land (\underbrace{\neg x \lor \neg y \lor z}) \land (\underbrace{\neg x \lor y \lor \neg z})$$



If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET

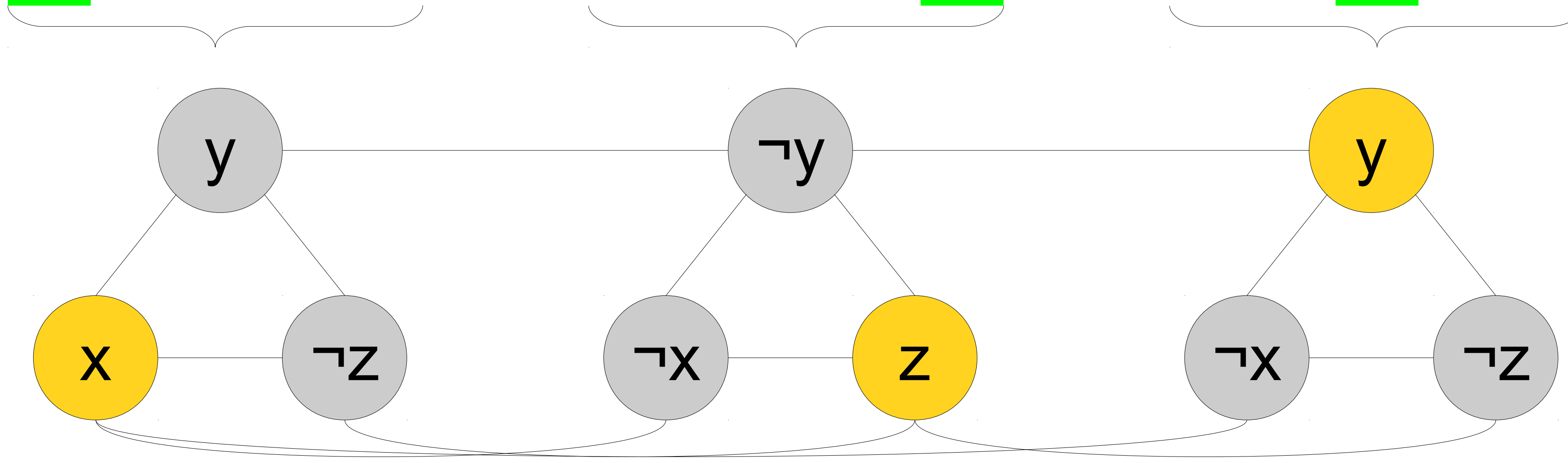
$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET

$$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$$

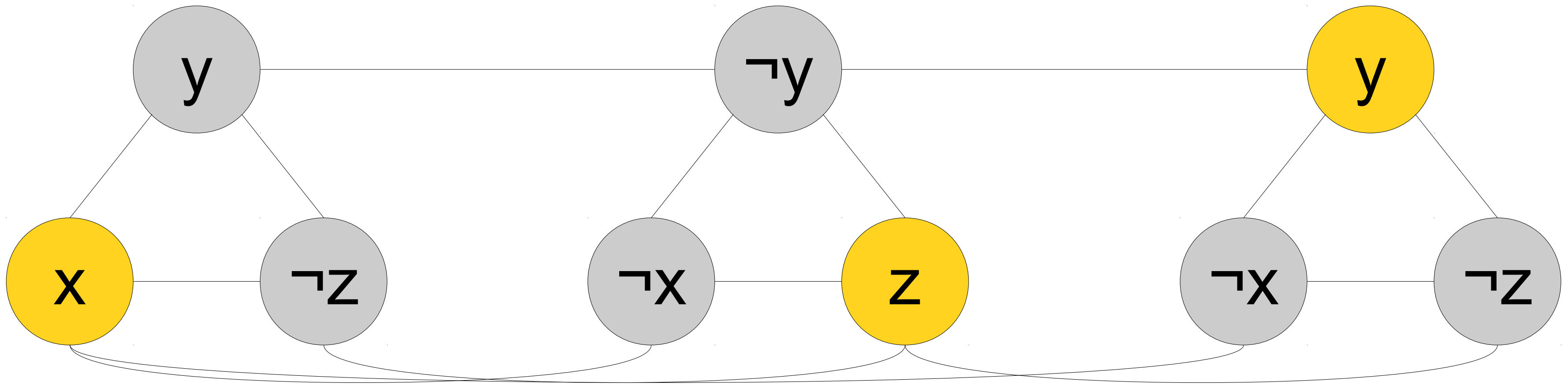


If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET

x = true, y = true, z = true.

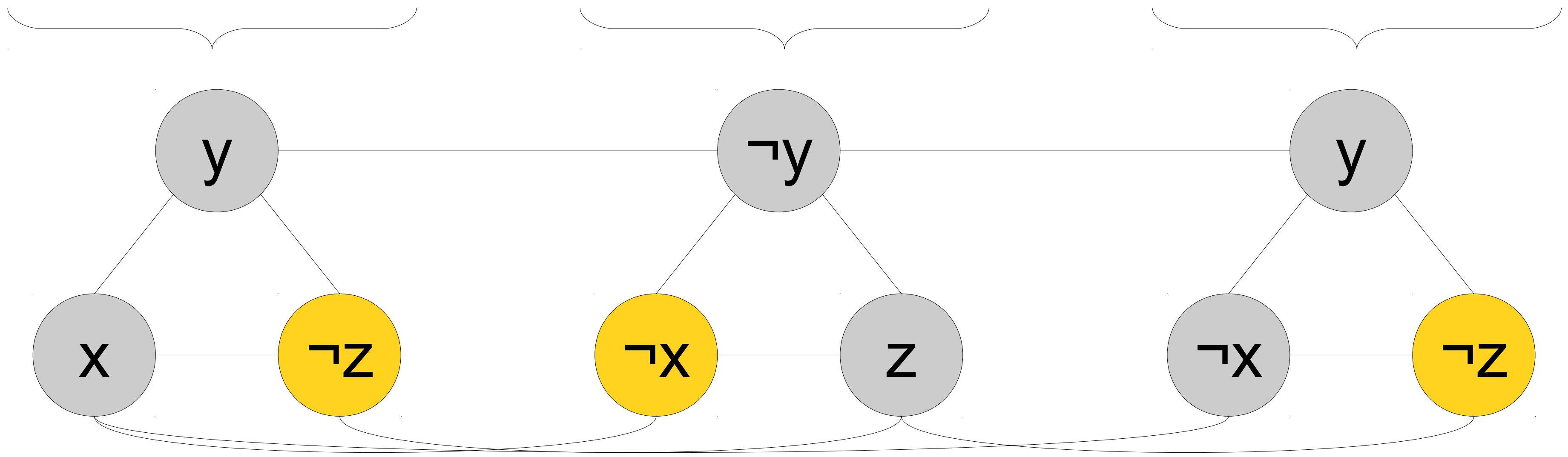
$$\underbrace{(\, x \lor y \lor \neg z \,)} \land \underbrace{(\, \neg x \lor \neg y \lor z \,)} \land \underbrace{(\, \neg x \lor y \lor \neg z \,)}$$



If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET

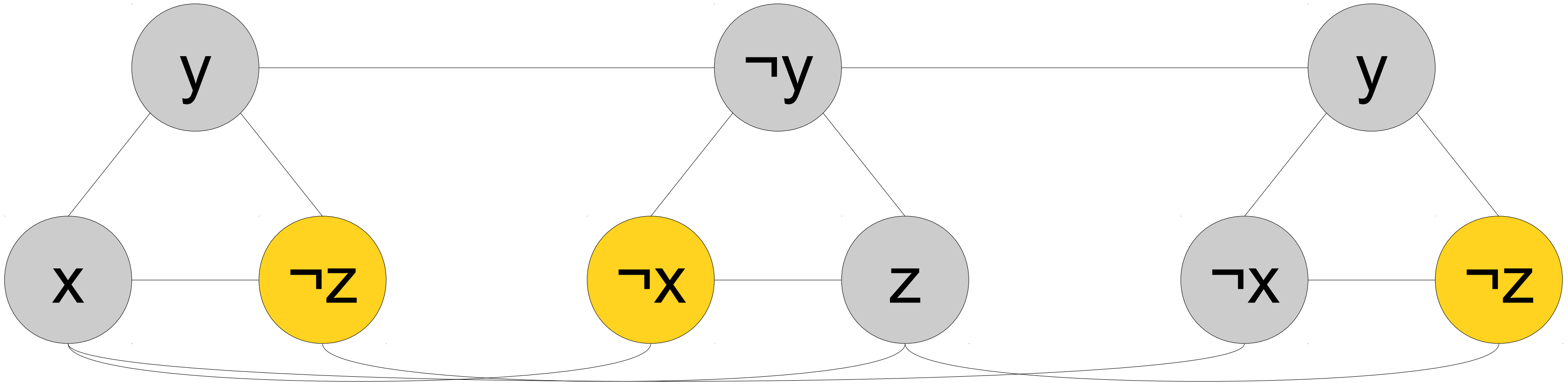
$$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$$



If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET

$$\underbrace{(\ x \lor y \lor \neg z\ )}\ \land\ \underbrace{(\ \neg x \lor \neg y \lor z\ )}\ \land\ \underbrace{(\ \neg x \lor y \lor \neg z\ )}$$

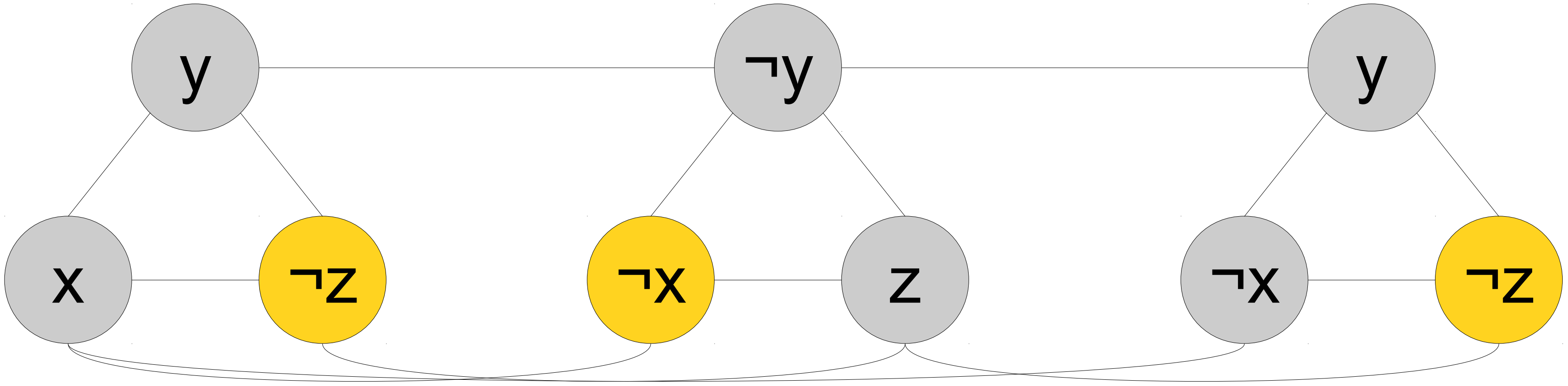


If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET

x = false, y = ??, z = false.

$$\underbrace{(\ x \lor y \lor \neg z\ )} \land \underbrace{(\ \neg x \lor \neg y \lor z\ )} \land \underbrace{(\ \neg x \lor y \lor \neg z\ )}$$

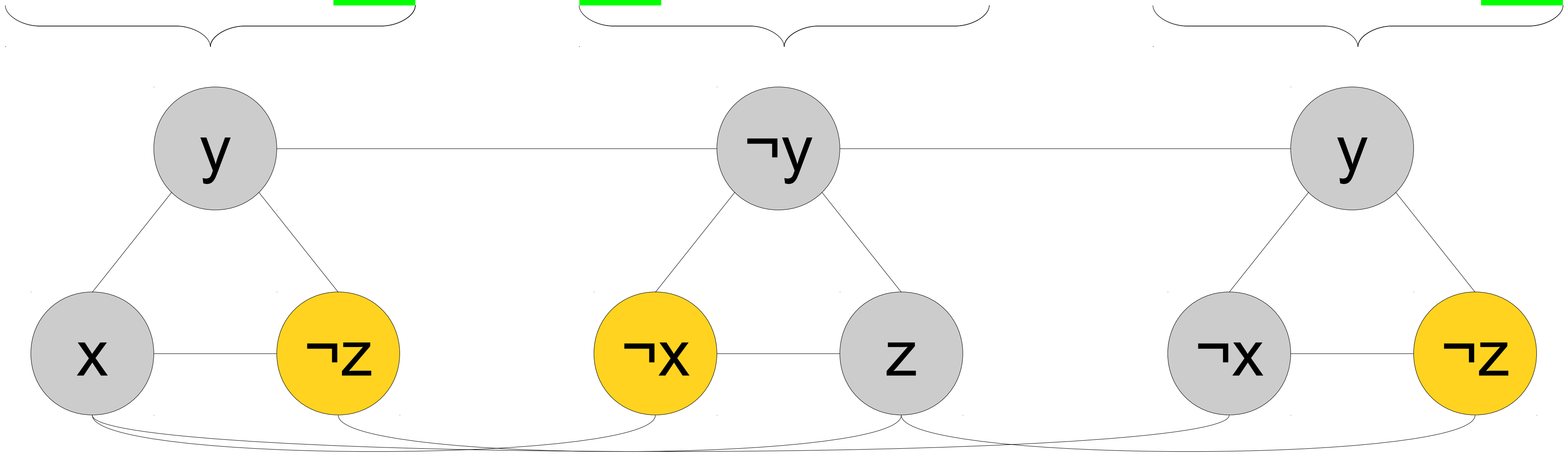


If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET

x = false, y = true, z = false.

(x ∨ y ∨ ¬z) ∧ (¬x ∨ ¬y ∨ z) ∧ (¬x ∨ y ∨ ¬z)

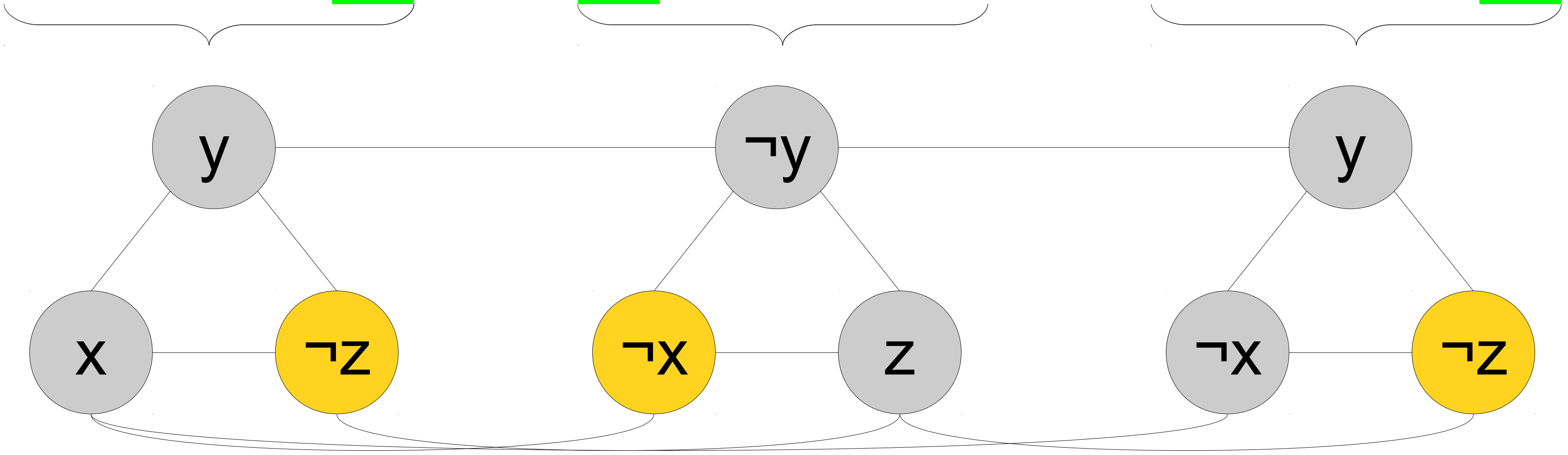


If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET

x = false, y = false, z = false.

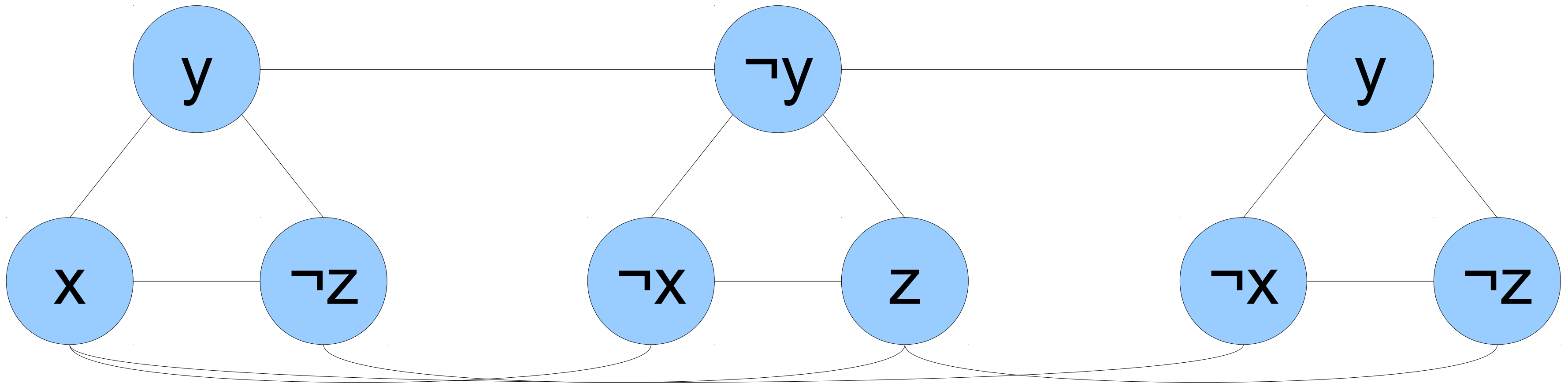
$$\underbrace{(\ x\ \lor\ y\ \lor\ \neg z)}\ \land\ \underbrace{(\neg x\ \lor\ \neg y\ \lor\ z\ )}\ \land\ \underbrace{(\neg x\ \lor\ y\ \lor\ \neg z)}$$



If this graph has an independent set of size three, the original formula is satisfiable.

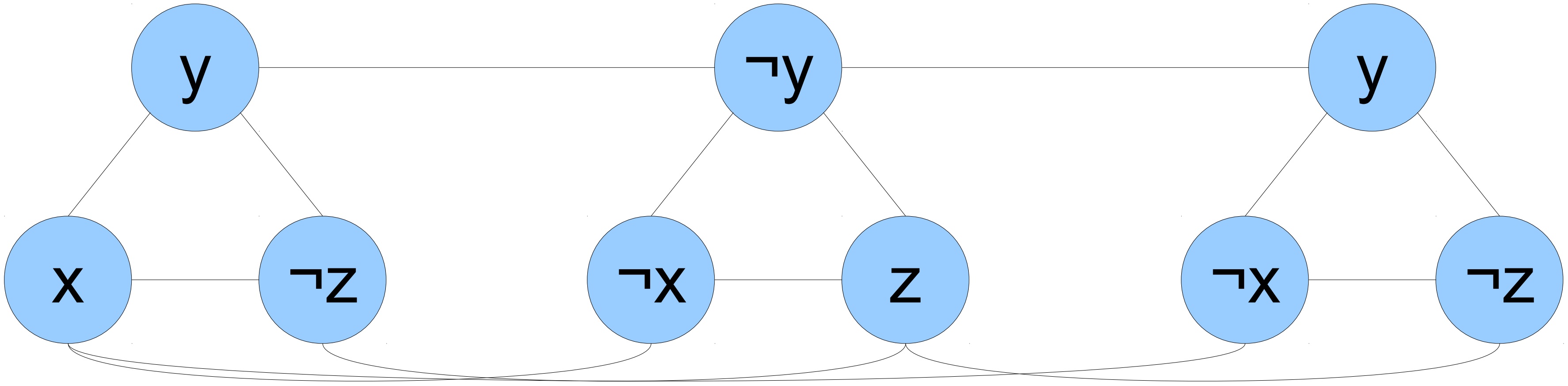
From 3SAT to INDSET

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



From 3SAT to INDSET

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$

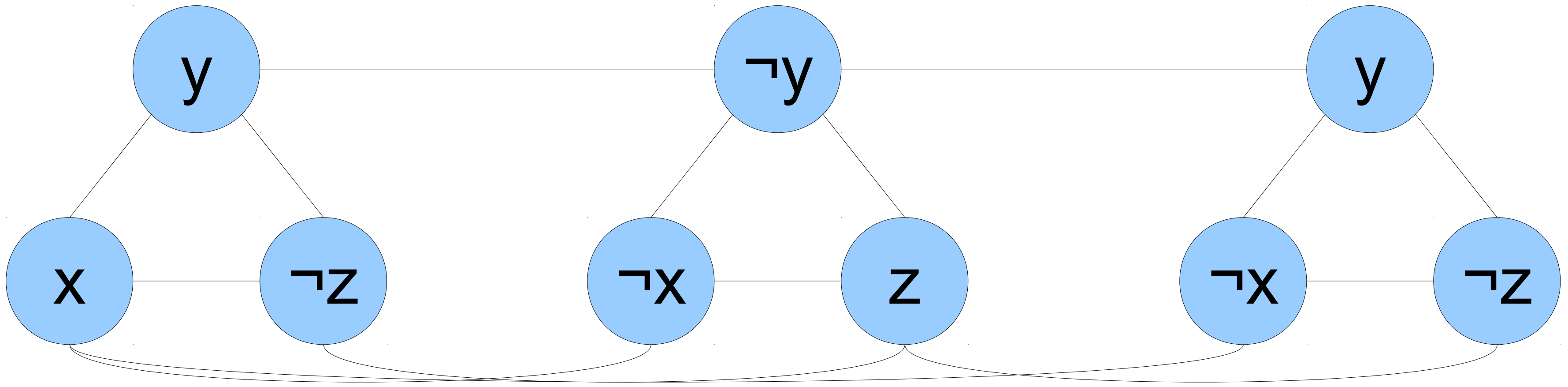


If the original formula is satisfiable,
this graph has an independent set of size three.

From 3SAT to INDSET

x = false, y = true, z = false.

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$

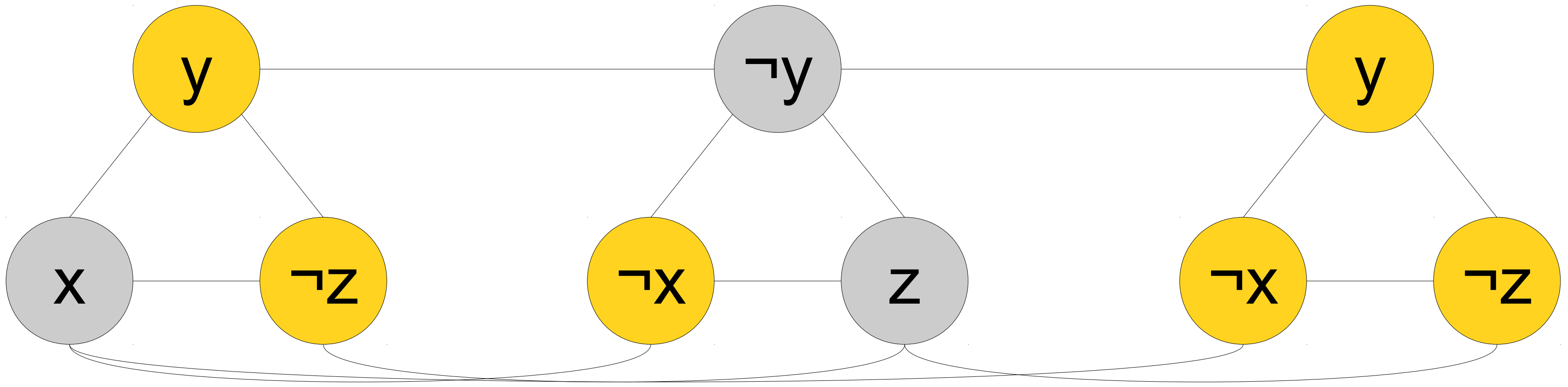


If the original formula is satisfiable,
this graph has an independent set of size three.

From 3SAT to INDSET

x = false, y = true, z = false.

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$

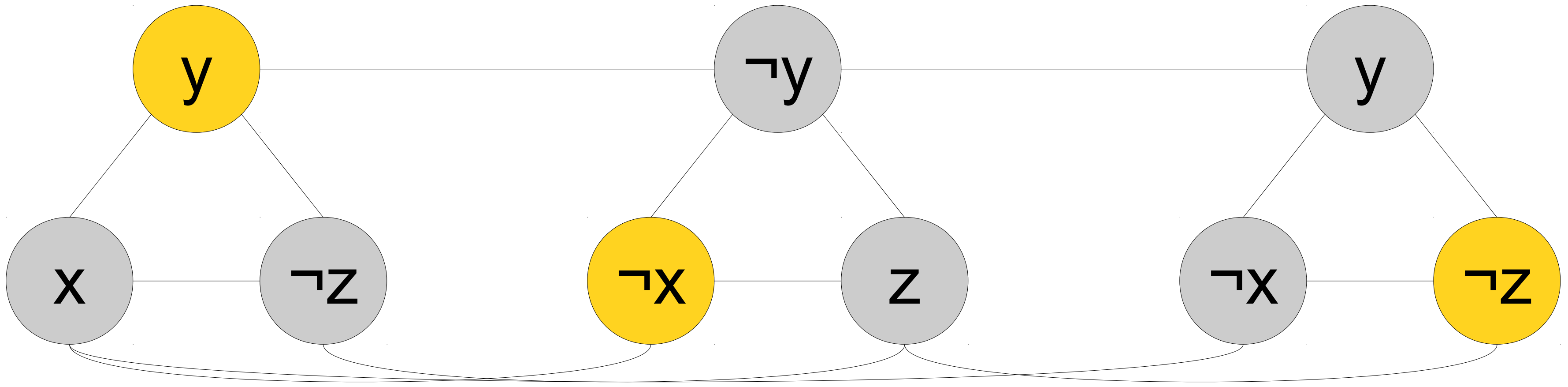


If the original formula is satisfiable, this graph has an independent set of size three.

From 3SAT to INDSET

x = false, y = true, z = false.

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



If the original formula is satisfiable,
this graph has an independent set of size three.

From 3SAT to INDSET

- Let $\varphi = C_1 \land C_2 \land \ldots \land C_n$ be a 3-CNF formula.
- Construct the graph G as follows:
  - For each clause $C_\mathrm{i} = x_1 \lor x_2 \lor x_3$, where x_1, x_2, and x_3 are literals, add three new nodes into G with edges connecting them.
  - For each pair of nodes v_i and $\neg v_i$, where v_i is some variable, add an edge connecting v_i and $\neg v_i$. (Note that there are multiple copies of these nodes)
- ***Claim One:*** This reduction can be computed in polynomial time.
- ***Claim Two:*** G has an independent set of size n iff φ is satisfiable.

INDSET ∈ **NPC**

- ***Theorem:*** INDSET is **NP**-complete.
- ***Proof sketch:*** We just showed that INDSET ∈ **NP** and that 3SAT $\le_p$ INDSET. Therefore, INDSET is **NP**-complete. ■

Time-Out For Announcements!

Please evaluate this course in Axess.

Your feedback really makes a difference.

Final Exam Logistics

- The final exam will be one week from today: **Wednesday, March 16** from **3:30PM – 6:30PM**, location TBA.
- As before, the exam is closed-book, closed-computer, and limited-note. You can have one 8.5” × 11”, double-sided sheet of notes with you when you take the exam.
- Topic coverage is cumulative with a slight focus on PS7 – PS9.

Extra Practice

- We have released
  - three sets of extra practice problems (EPP8 – EPP10), with solutions;
  - a set of challenge problems, without solutions; and
  - two practice finals, with solutions.
- You are ***encouraged*** to stop by office hours with questions or to ask questions on Piazza. We want you to do well!

My Favorite Handout

- We just released, online, my favorite handout of the whole quarter: the Timeline of CS103 Results.
- This handout gives the chronology of the results we've covered in CS103. It's amazing in that it's almost entirely in the wrong order!
- If you have a few free minutes, take a look over it. It's fascinating to see the history of ideas!

Your Questions

“You've mentioned how we can rely on human verification when computer verification is not enough. But aren't humans just computers made of cells not chips? If so, does this lead to paradoxes about what we can say about that amorphous enormity called truth?”

I think it's a good idea to rely on human verification not because it works in all cases, but because people can make judgment calls about what's best for society based on years of learned experience. Ultimately, the goal of verifying a voting machine is to use it in an election, and people are well-equipped to make calls like “this program is too weird – we shouldn't use it” or “something about this program seems suspicious.”

So... yeah! I don't think there's any paradoxes here.

"Can we pleaaaase have the mock practice final? Those were so helpful…. Please?"

Quick show of hands – how many of you would go to one?

“What myth/false belief do you think is most widespread among Stanford students? Care to dispel it?”

I've got two. ☺

Back to CS103!

Structuring **NP**-Completeness Reductions

The Shape of a Reduction

- Polynomial-time reductions work by solving one problem with a solver for a different problem.
- Most problems in **NP** have different pieces that must be solved simultaneously.
- For example, in 3SAT:
  - Each clause must be made true,
  - but no literal and its complement may be picked.
- In INDSET:
  - You can choose any nodes you want to put into the set,
  - but no two adjacent nodes can be added.

Reductions and Gadgets

- Many reductions used to show **NP**-completeness work by using ***gadgets***.
- Each piece of the original problem is translated into a “gadget” that handles some particular detail of the problem.
- These gadgets are then connected together to solve the overall problem.

Gadgets in INDSET

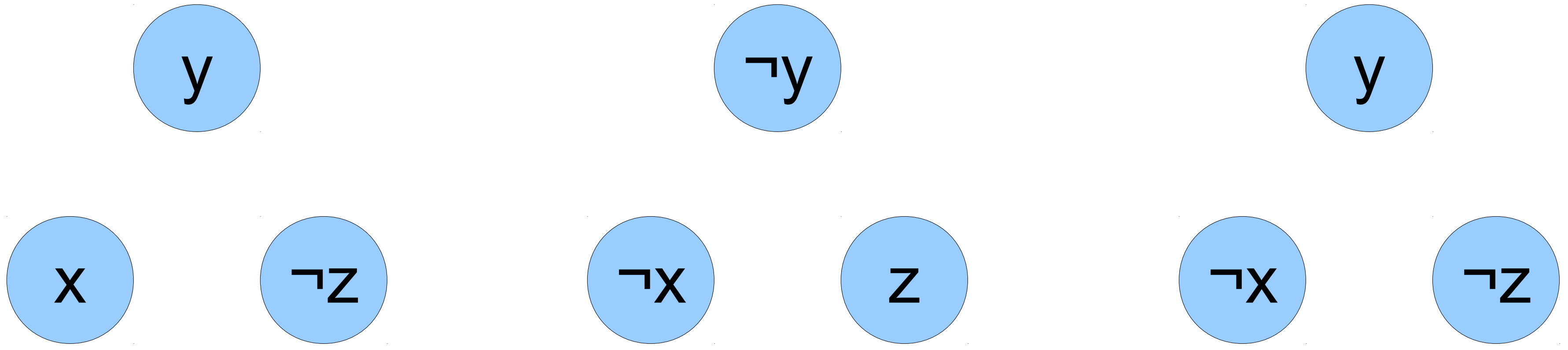
$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$

Gadgets in INDSET

$$(\underbrace{x \lor y \lor \neg z}_{}) \land (\underbrace{\neg x \lor \neg y \lor z}_{}) \land (\underbrace{\neg x \lor y \lor \neg z}_{})$$

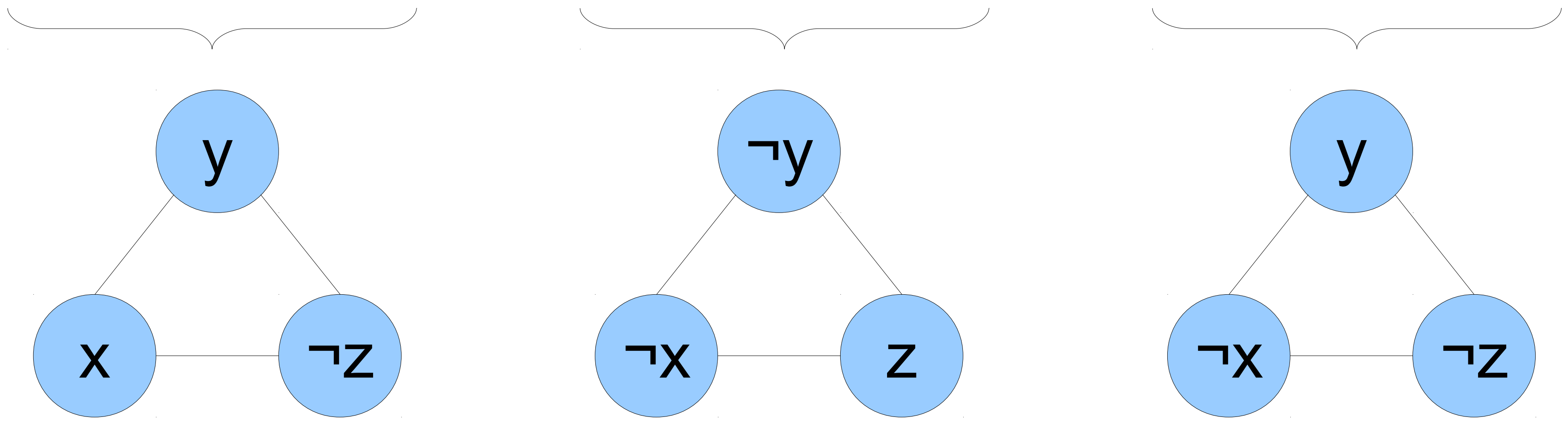
Gadgets in INDSET

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



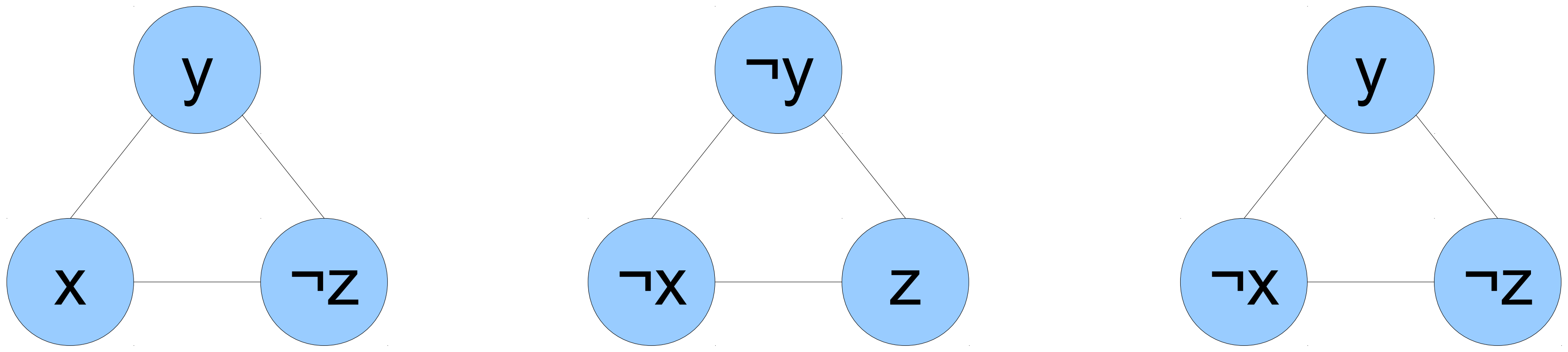
Gadgets in INDSET

$$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$$



Gadgets in INDSET

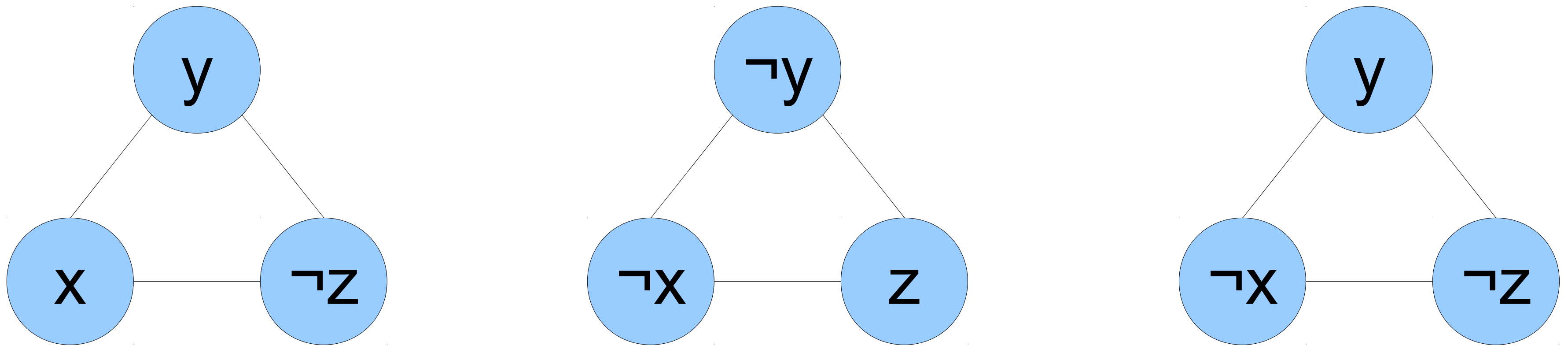
$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



Each of these gadgets is designed to solve one part of the problem: ensuring each clause is satisfied.

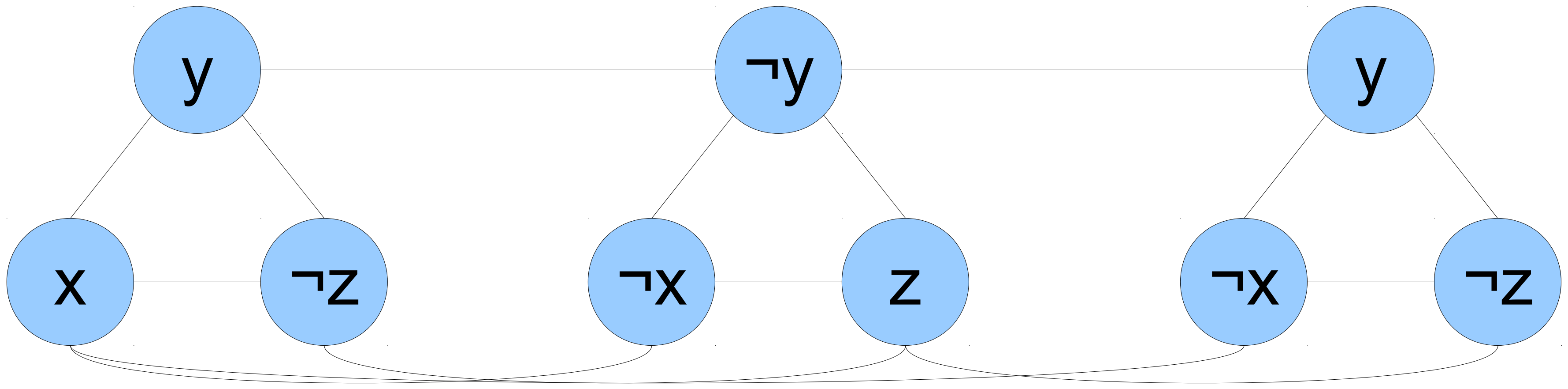
Gadgets in INDSET

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



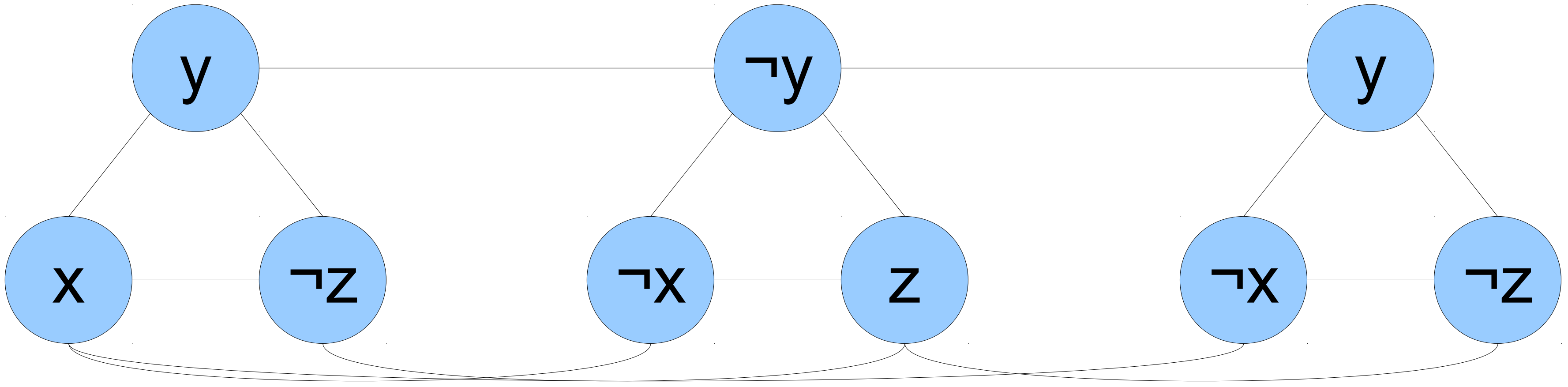
Gadgets in INDSET

$$(\underbrace{x \lor y \lor \neg z}) \land (\underbrace{\neg x \lor \neg y \lor z}) \land (\underbrace{\neg x \lor y \lor \neg z})$$



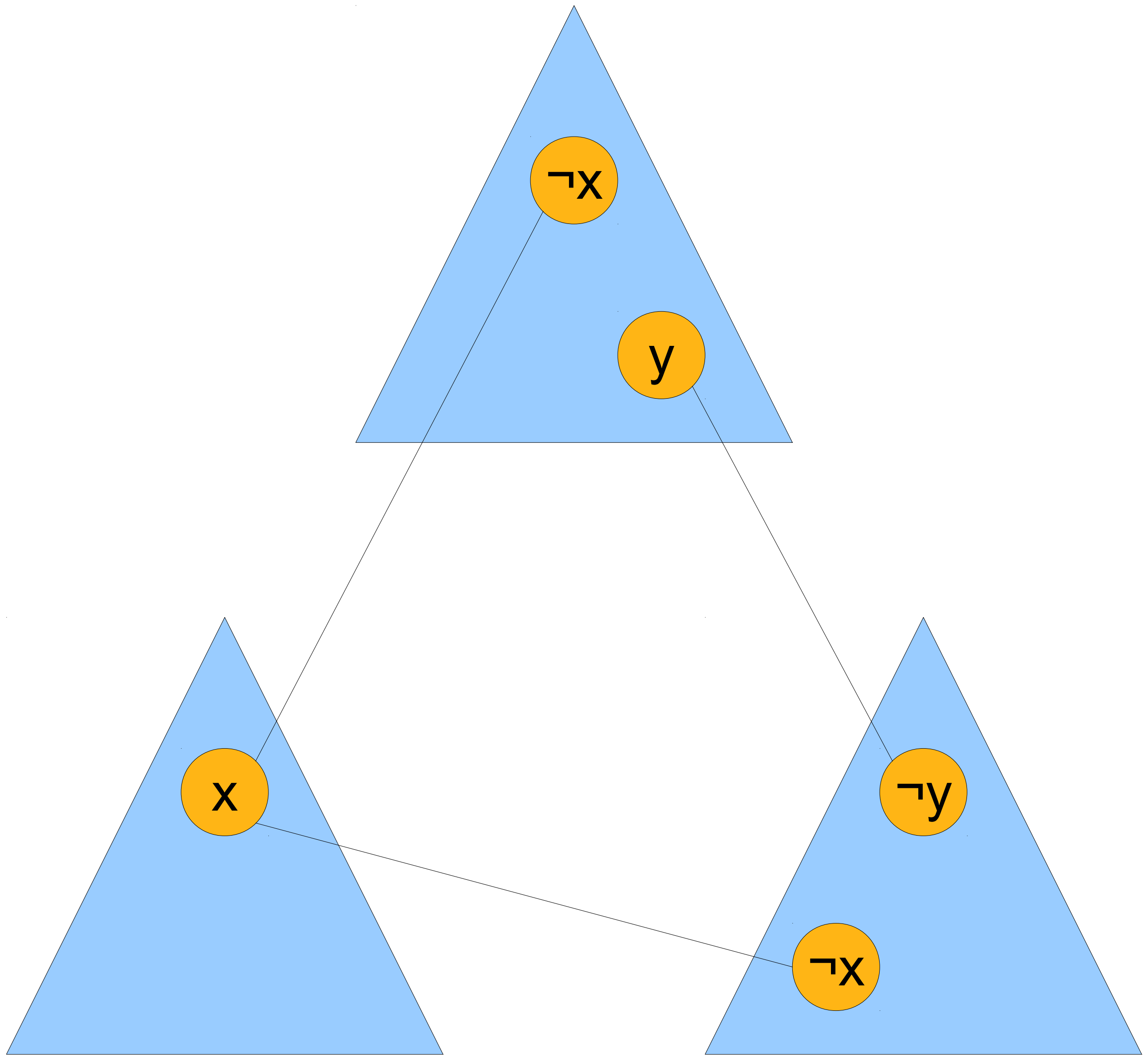
Gadgets in INDSET

$$\underbrace{(x \lor y \lor \neg z)} \land \underbrace{(\neg x \lor \neg y \lor z)} \land \underbrace{(\neg x \lor y \lor \neg z)}$$



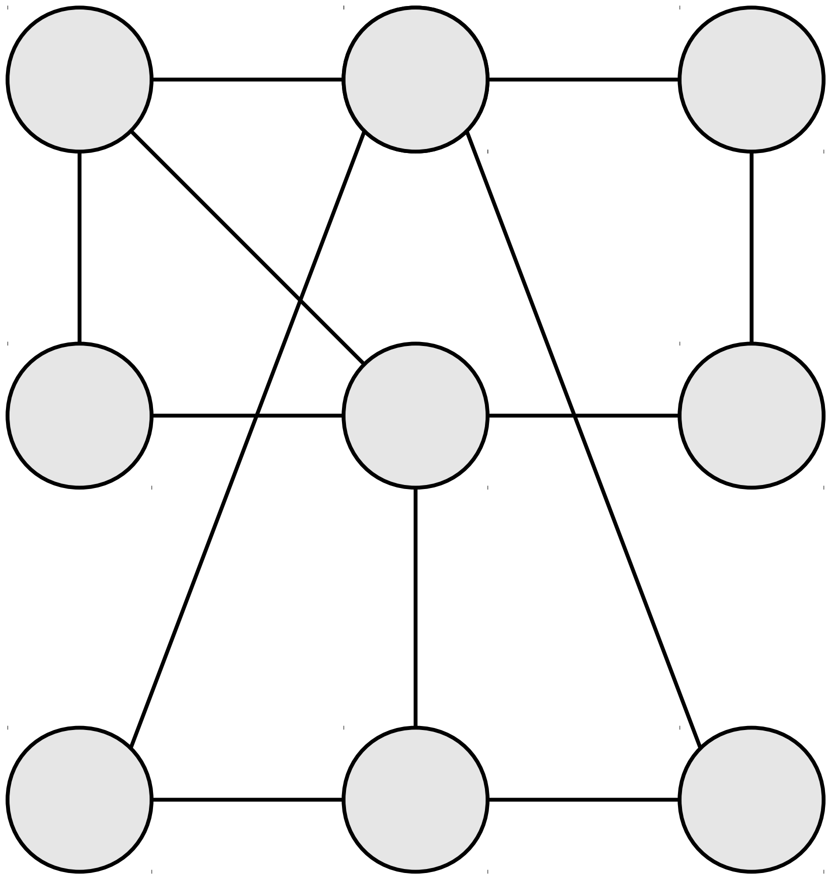
These connections ensure that the solutions to each gadget are linked to one another.

Gadgets in INDSET

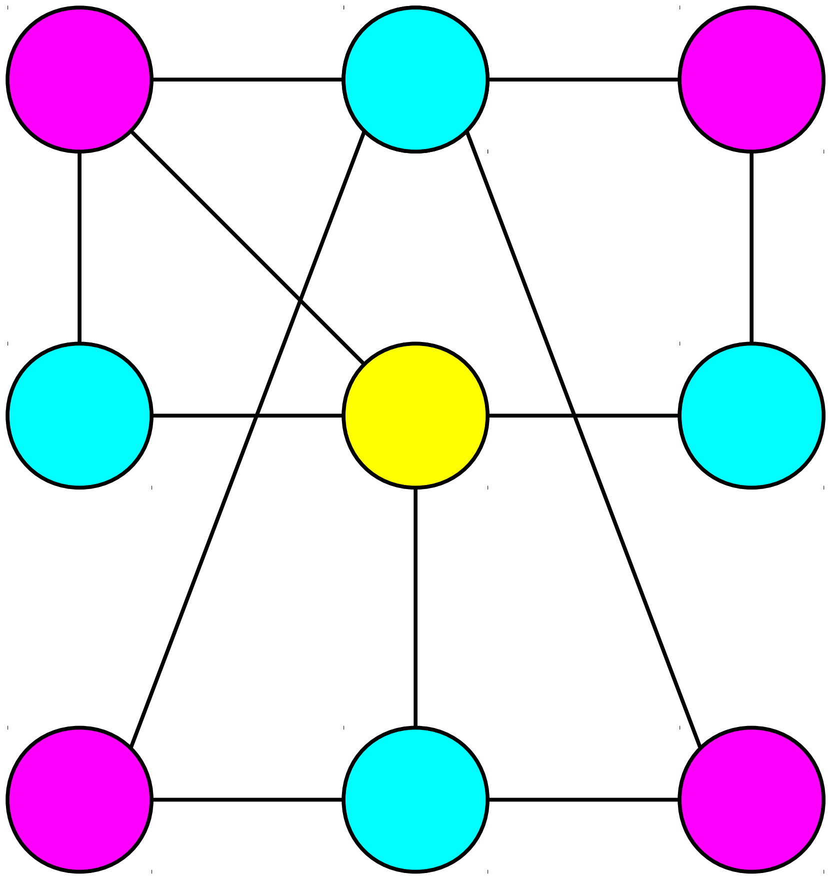


A More Complex Reduction

A ***3-coloring*** of a graph is a way of coloring its nodes one of three colors such that no two connected nodes have the same color.



A ***3-coloring*** of a graph is a way of coloring its nodes one of three colors such that no two connected nodes have the same color.



A ***3-coloring*** of a graph is a way of coloring its nodes one of three colors such that no two connected nodes have the same color.

The 3-Coloring Problem

- The ***3-coloring problem*** is

**Given an undirected graph *G*,
is there a legal 3-coloring of its nodes?**

- As a formal language:

3COLOR = { ⟨*G*⟩ | *G* is an undirected graph with a legal 3-coloring. }

- This problem is known to be **NP**-complete by a reduction from 3SAT.

3COLOR ∈ **NP**

- We can prove that 3COLOR ∈ **NP** by designing a polynomial-time verifier for 3COLOR.
- V = "On input $\langle G, C\rangle$, where G is a graph and C is a way of assigning colors to the nodes of G:
  - Check whether each edge's endpoints are different colors.
  - If so, accept; otherwise reject."

A Note on Terminology

- Although 3COLOR and 3SAT both have “3” in their names, the two are very different problems.
  - 3SAT means “there are three literals in every clause.” However, each literal can take on only one of two different values.
  - 3COLOR means “every node can take on one of three different colors.”
- **Key difference**:
  - In 3SAT variables have two choices of value.
  - In 3COLOR nodes have three choices of value.

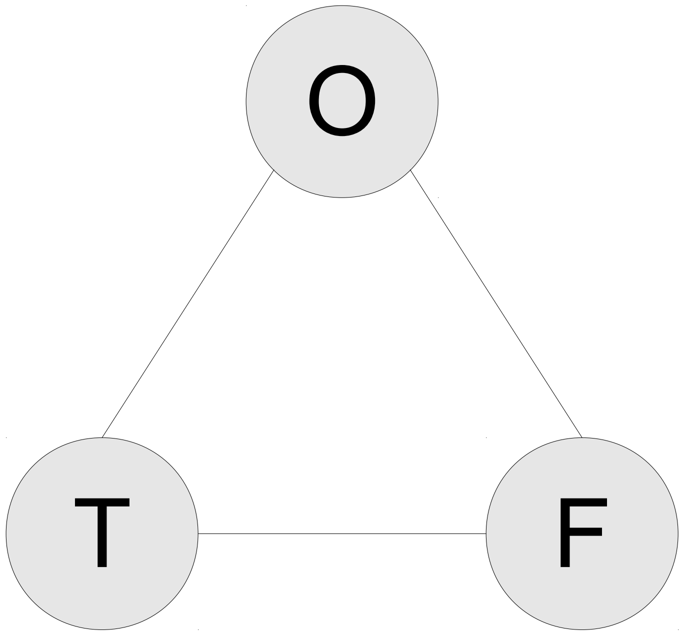
Why Not Two Colors?

- It would seem that 2COLOR (whether a graph has a 2-coloring) would be a better fit.
  - Every variable has one of two values.
  - Every node has one of two values.
- Interestingly, 2COLOR is known to be in **P** and is conjectured not to be **NP**-complete.
  - Though, if you can prove that it is, you've just won $1,000,000!

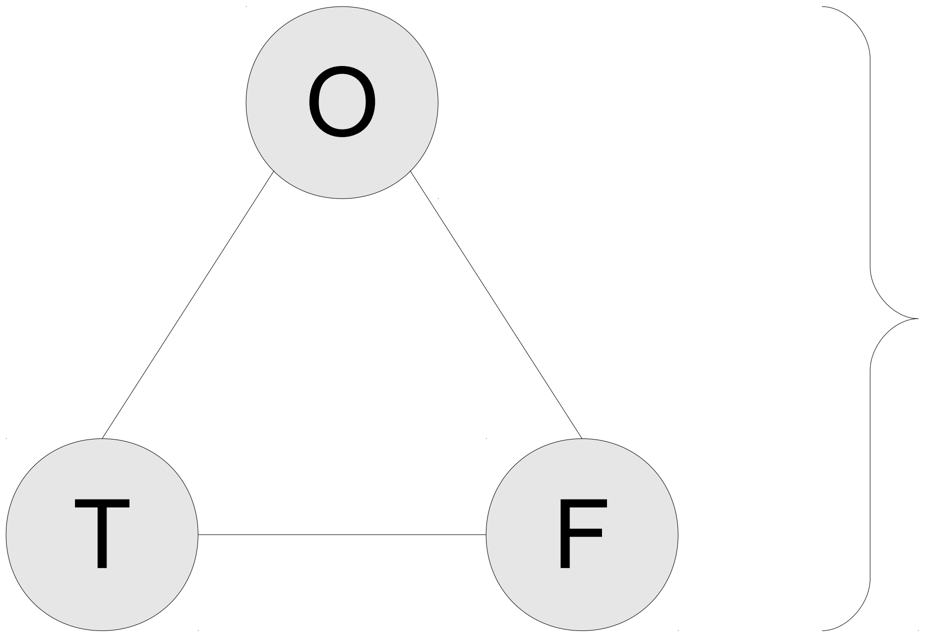
From 3SAT to 3COLOR

- In order to reduce 3SAT to 3COLOR, we need to somehow make a graph that is 3-colorable iff some 3-CNF formula φ is satisfiable.
- ***Idea:*** Use a collection of gadgets to solve the problem.
  - Build a gadget to assign two of the colors the labels “true” and “false.”
  - Build a gadget to force each variable to be either true or false.
  - Build a series of gadgets to force those variable assignments to satisfy each clause.

Gadget One: Assigning Meanings

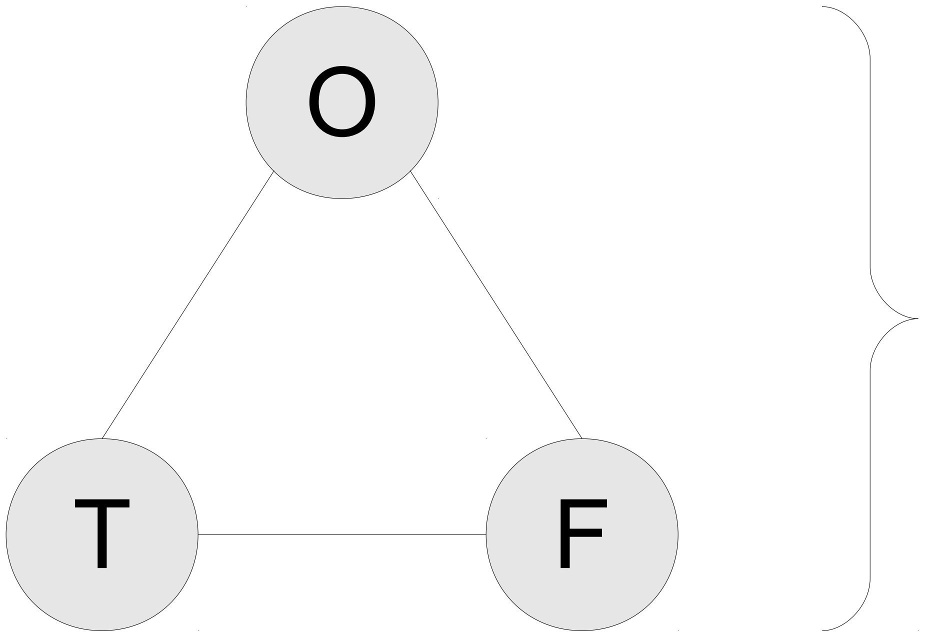


Gadget One: Assigning Meanings



These nodes must all have different colors.

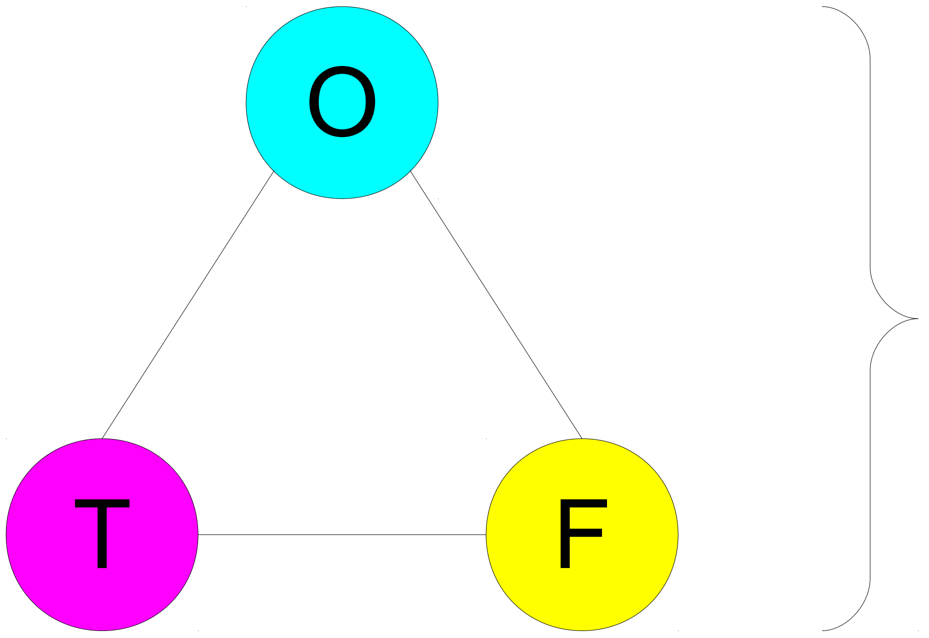
Gadget One: Assigning Meanings



These nodes must all have different colors.

The color assigned to T will be interpreted as "true."
The color assigned to F will be interpreted as "false."
We do not associate any special meaning with O.

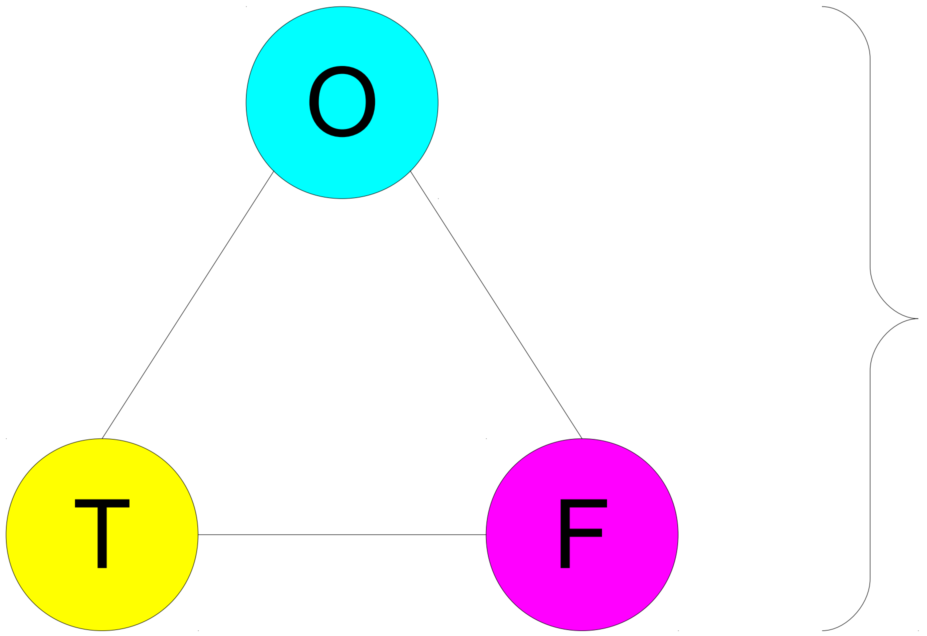
Gadget One: Assigning Meanings



These nodes must all have different colors.

The color assigned to T will be interpreted as "true."
The color assigned to F will be interpreted as "false."
We do not associate any special meaning with O.

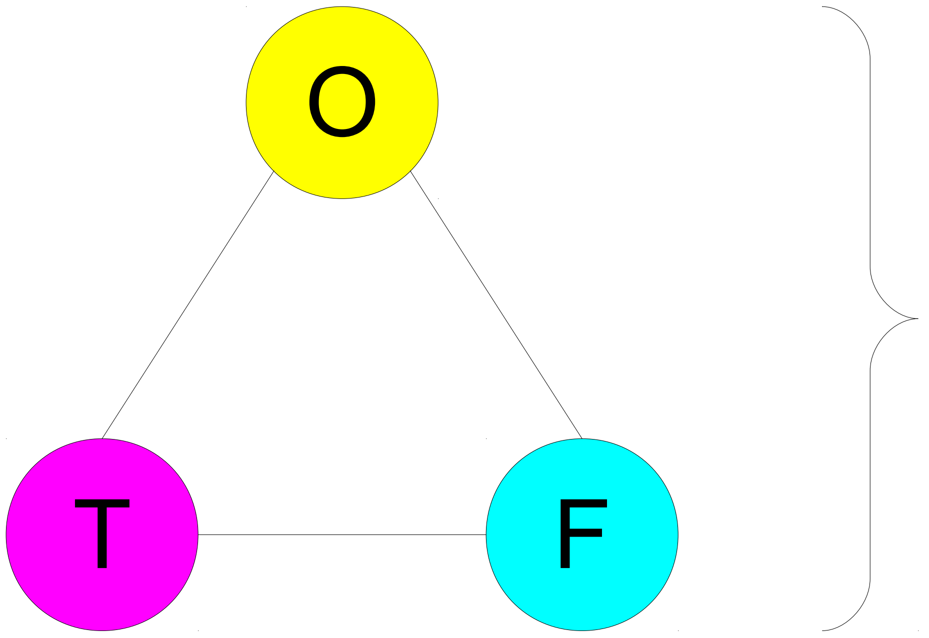
Gadget One: Assigning Meanings



These nodes must all have different colors.

The color assigned to T will be interpreted as "true."
The color assigned to F will be interpreted as "false."
We do not associate any special meaning with O.

Gadget One: Assigning Meanings



These nodes must all have different colors.

The color assigned to T will be interpreted as "true."
The color assigned to F will be interpreted as "false."
We do not associate any special meaning with O.

Gadget Two: Forcing a Choice

$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$

Gadget Two: Forcing a Choice

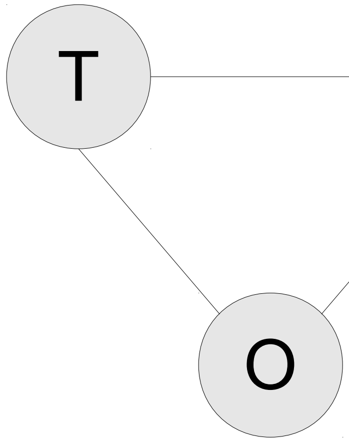
$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$

Gadget Two: Forcing a Choice

$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$

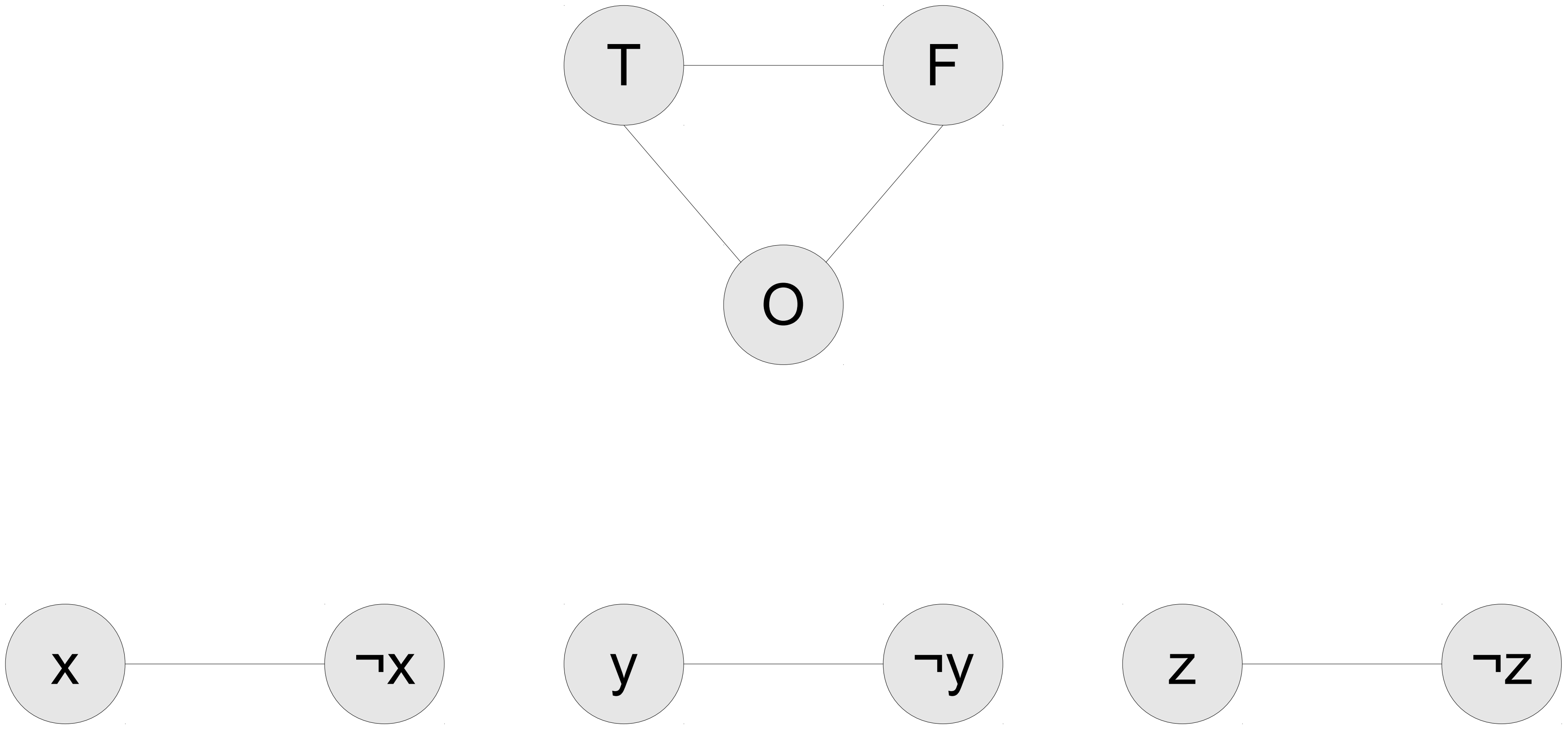
Gadget Two: Forcing a Choice

$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$



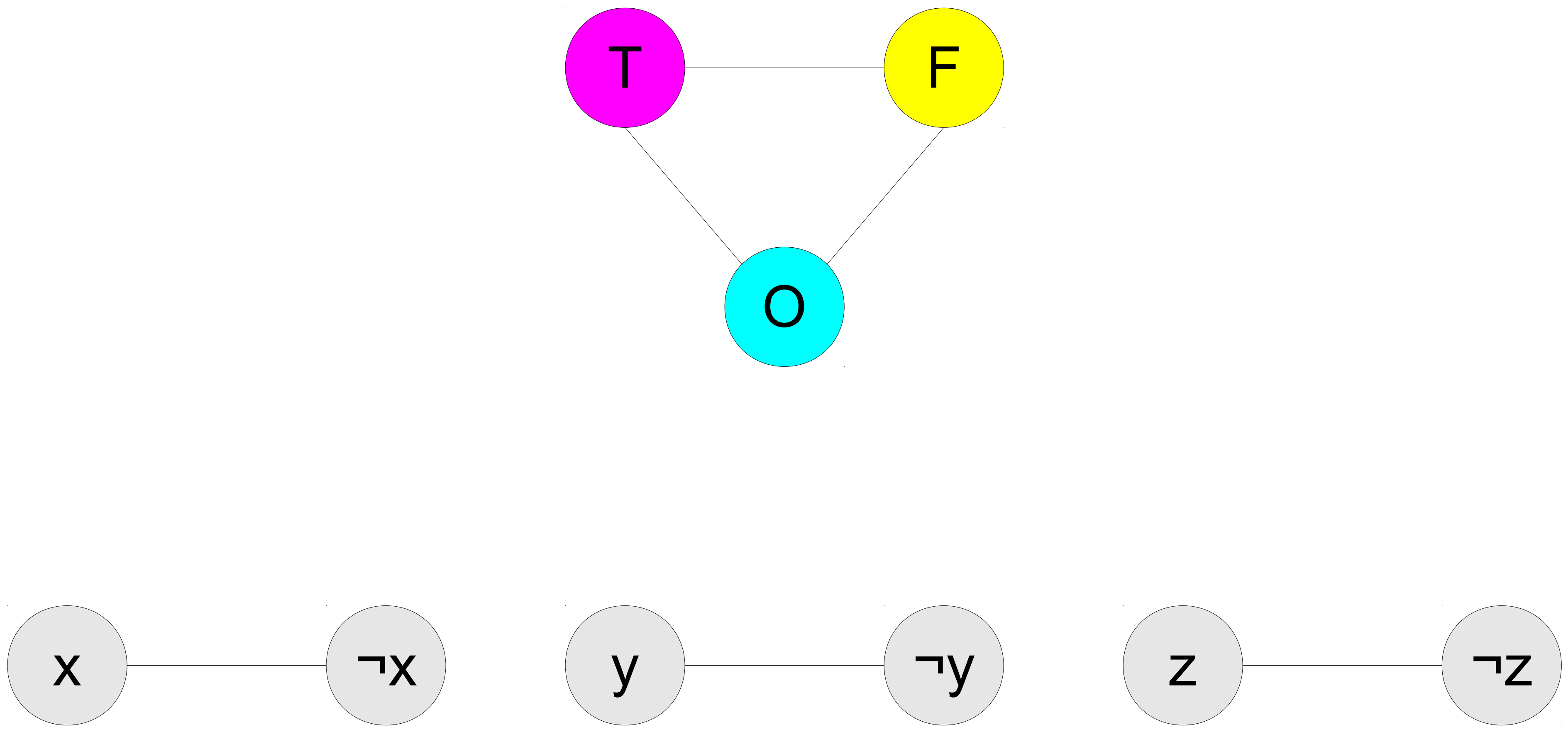
Gadget Two: Forcing a Choice

$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$



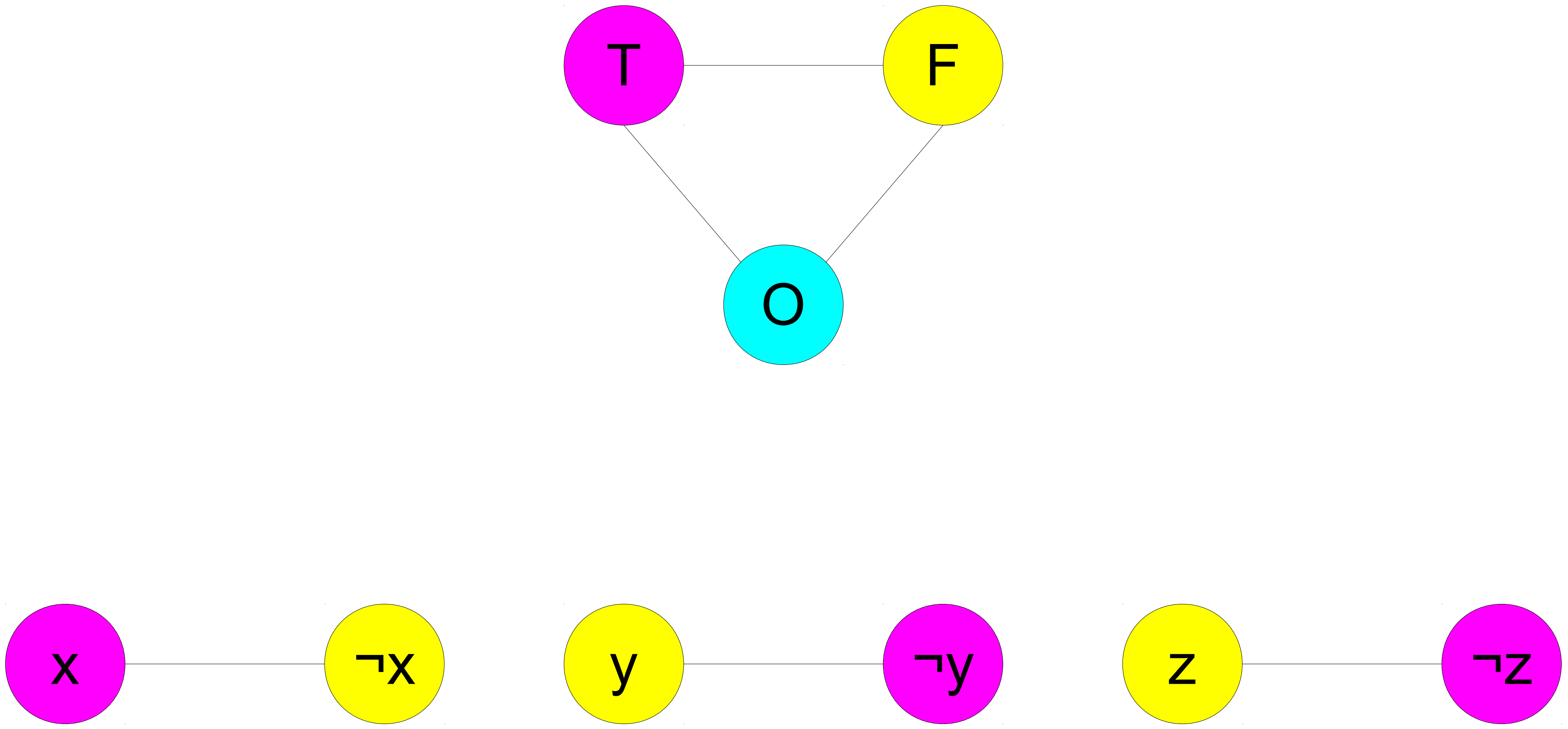
Gadget Two: Forcing a Choice

$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$



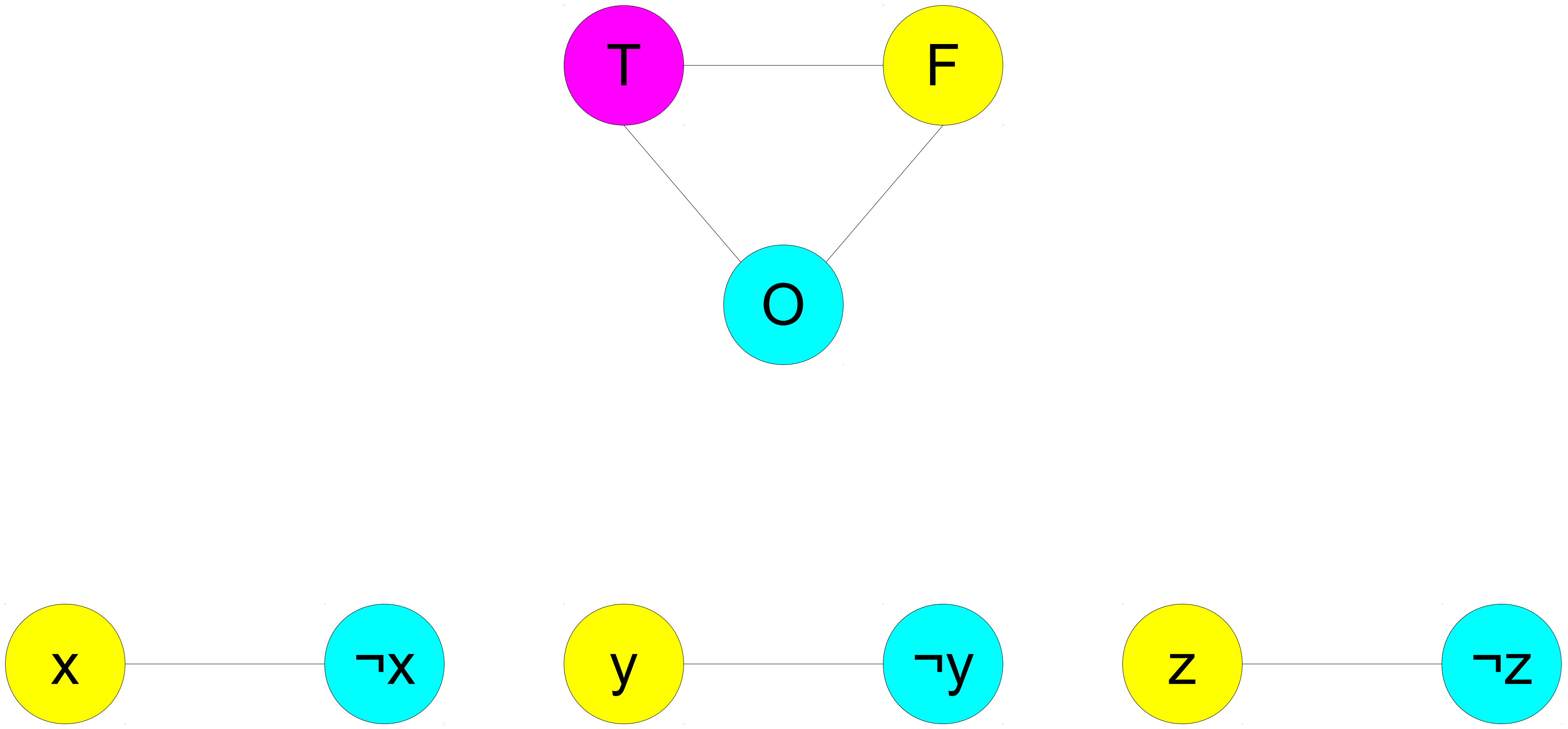
Gadget Two: Forcing a Choice

$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$



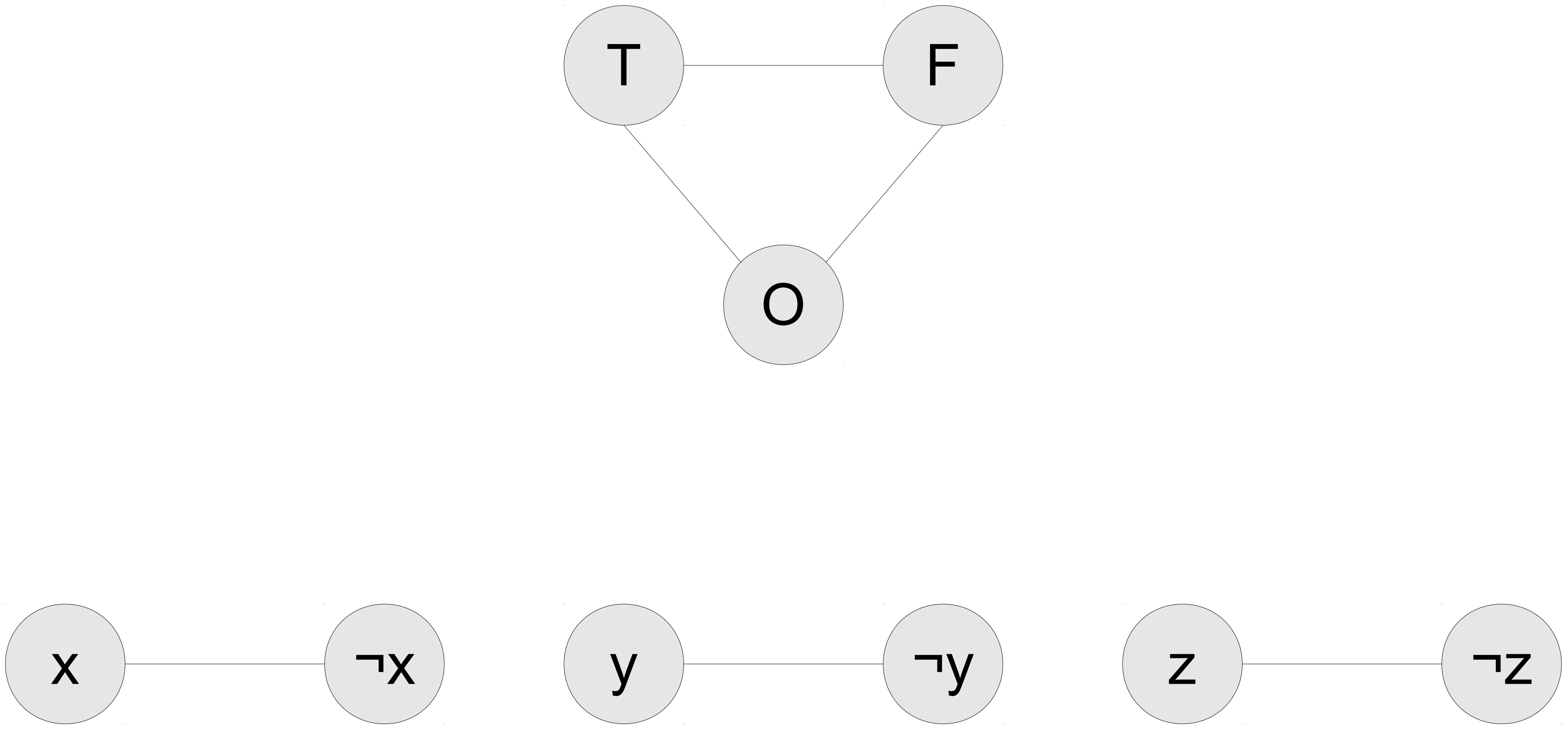
Gadget Two: Forcing a Choice

$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$



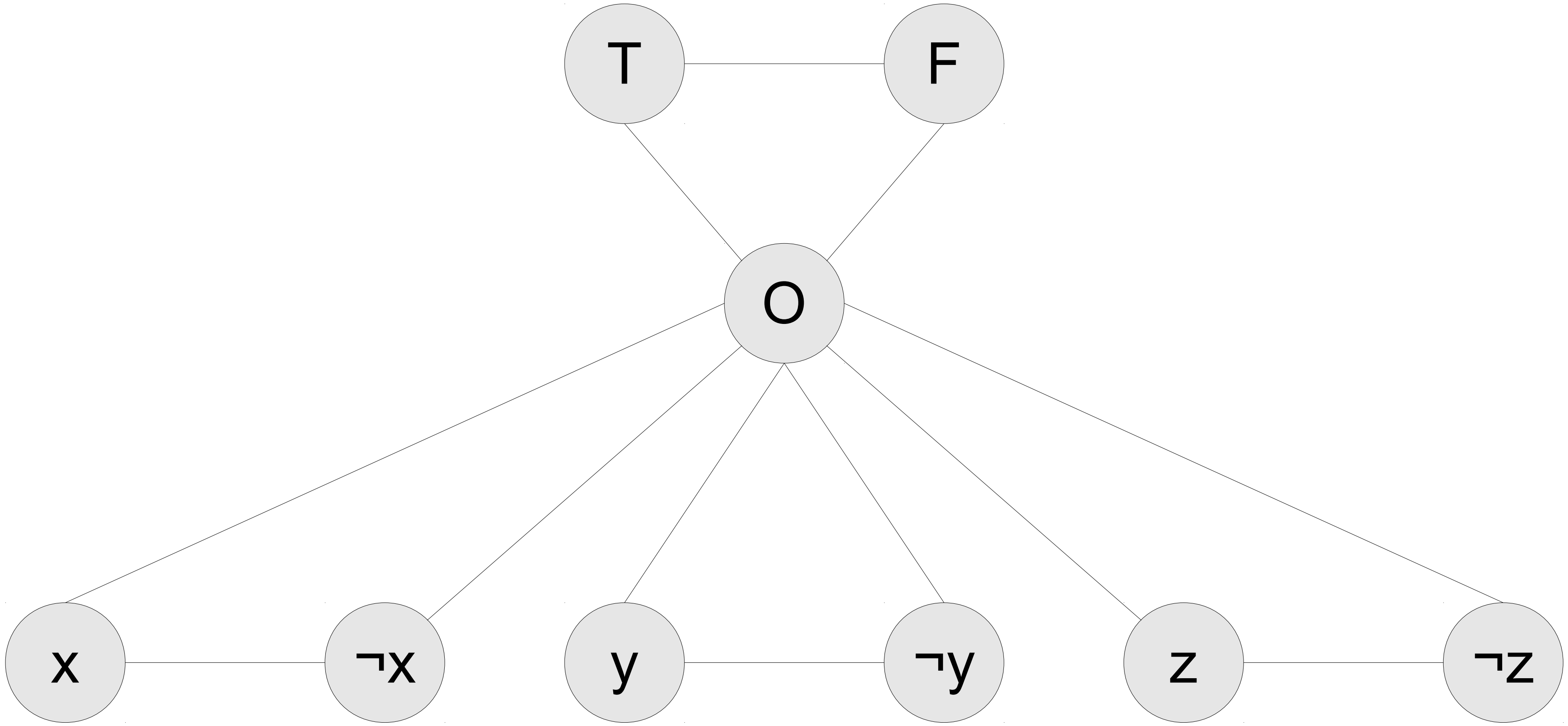
Gadget Two: Forcing a Choice

$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$



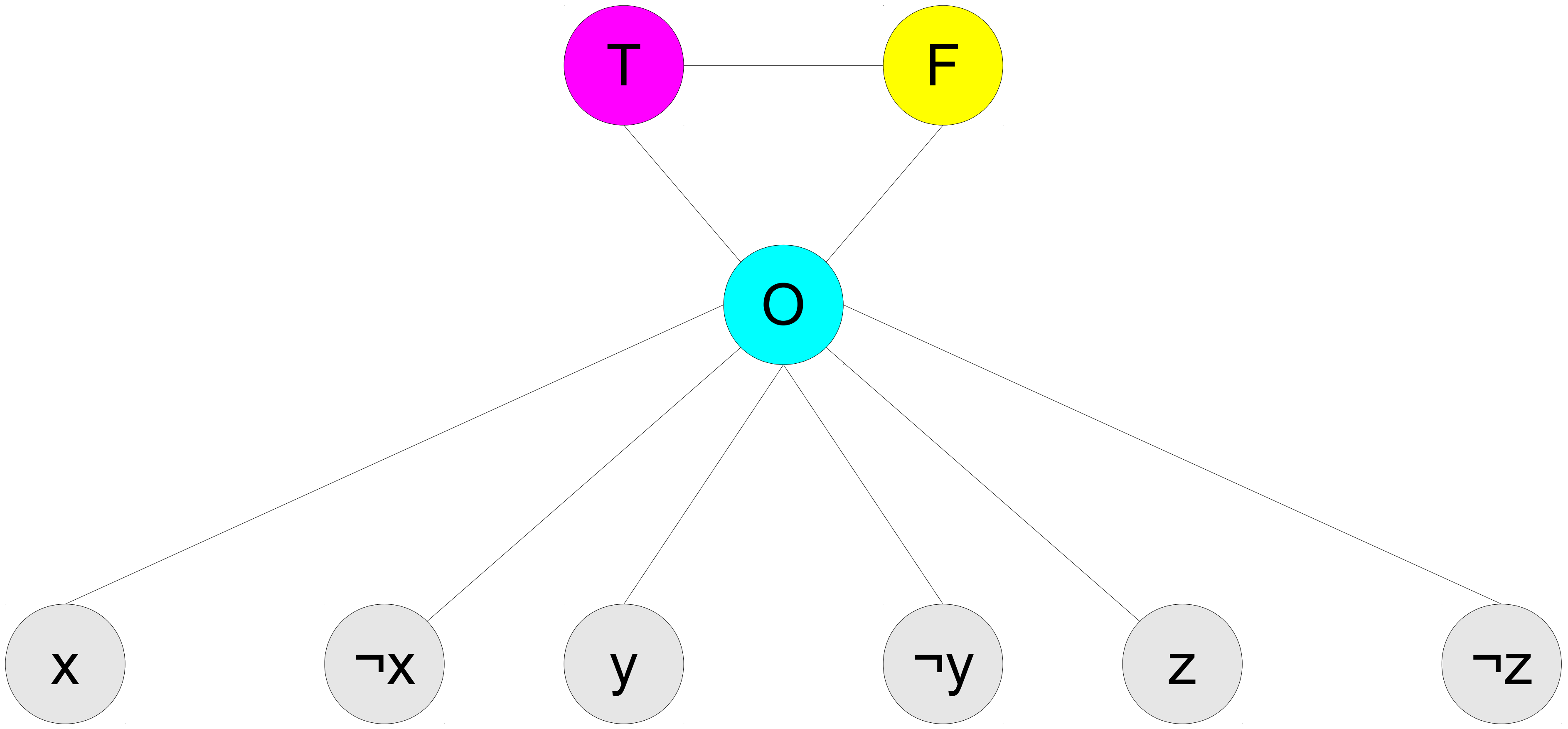
Gadget Two: Forcing a Choice

$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$



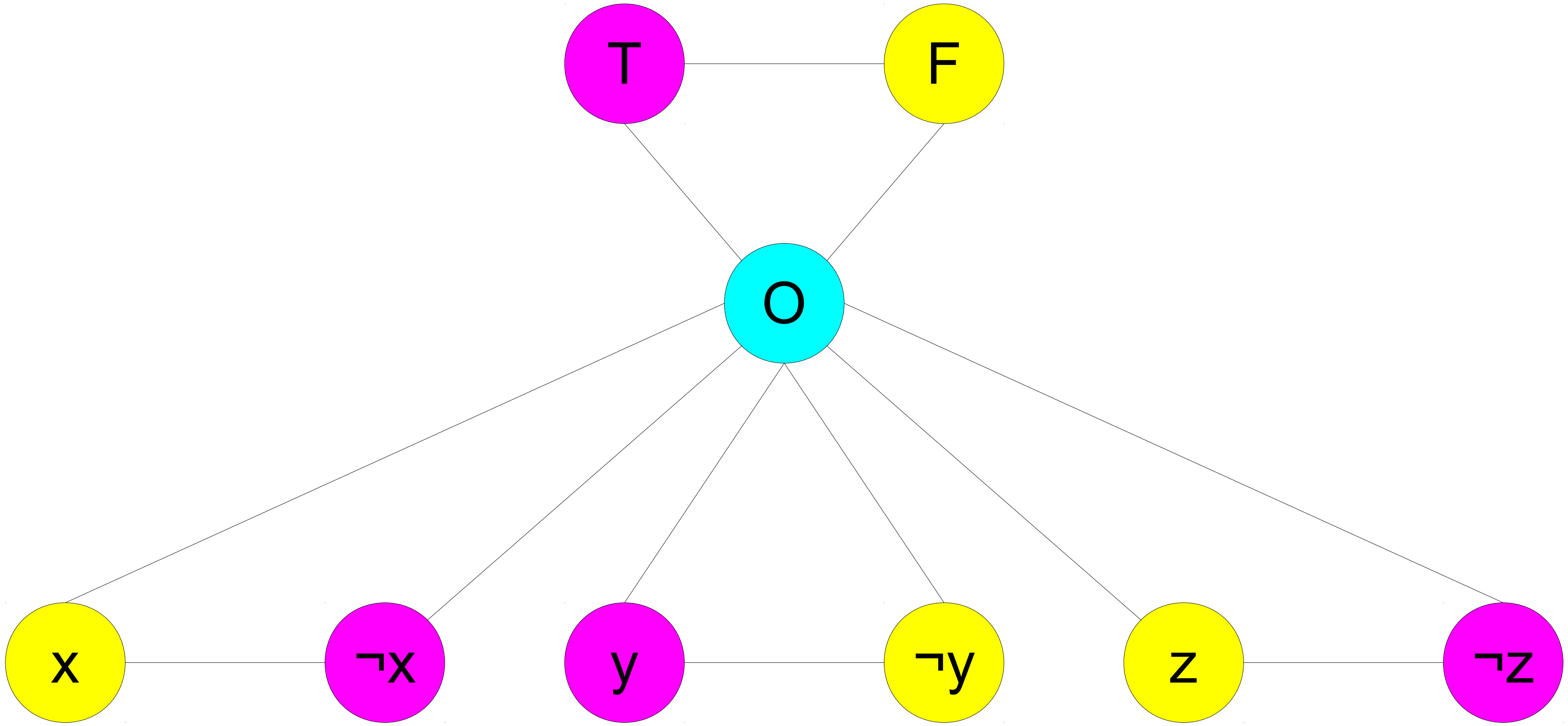
Gadget Two: Forcing a Choice

$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$



Gadget Two: Forcing a Choice

$(x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z) \land (\neg x \lor y \lor \neg z)$

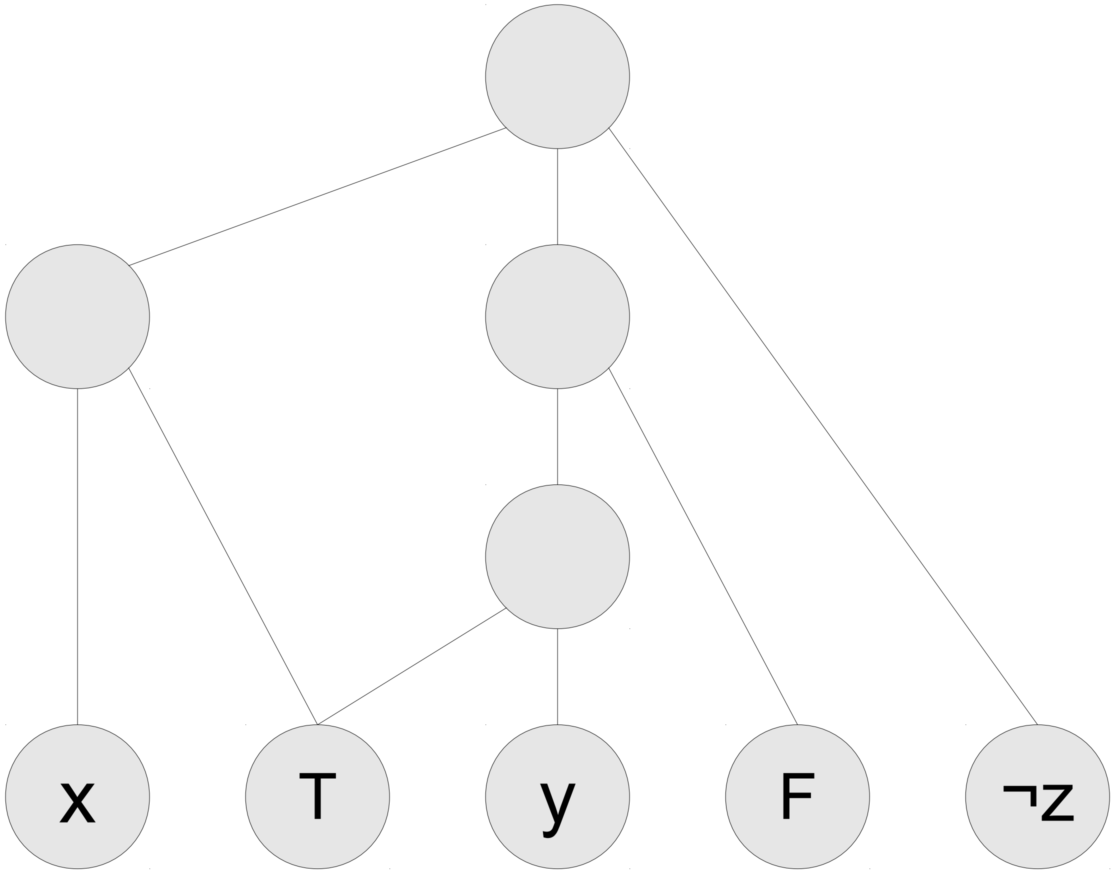


Gadget Three: Clause Satisfiability

$(x \lor y \lor \neg z)$

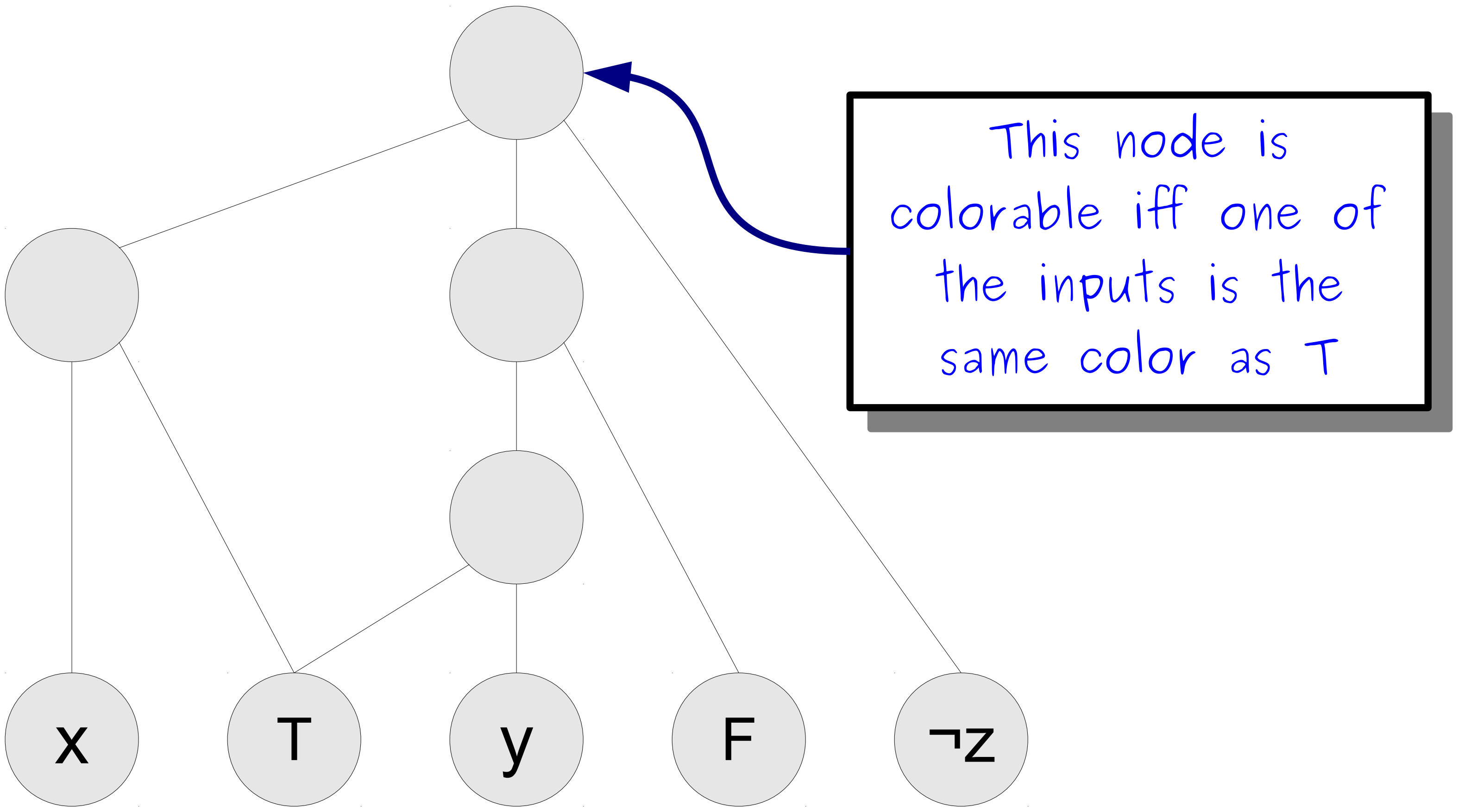
Gadget Three: Clause Satisfiability

(x ∨ y ∨ ¬z)



Gadget Three: Clause Satisfiability

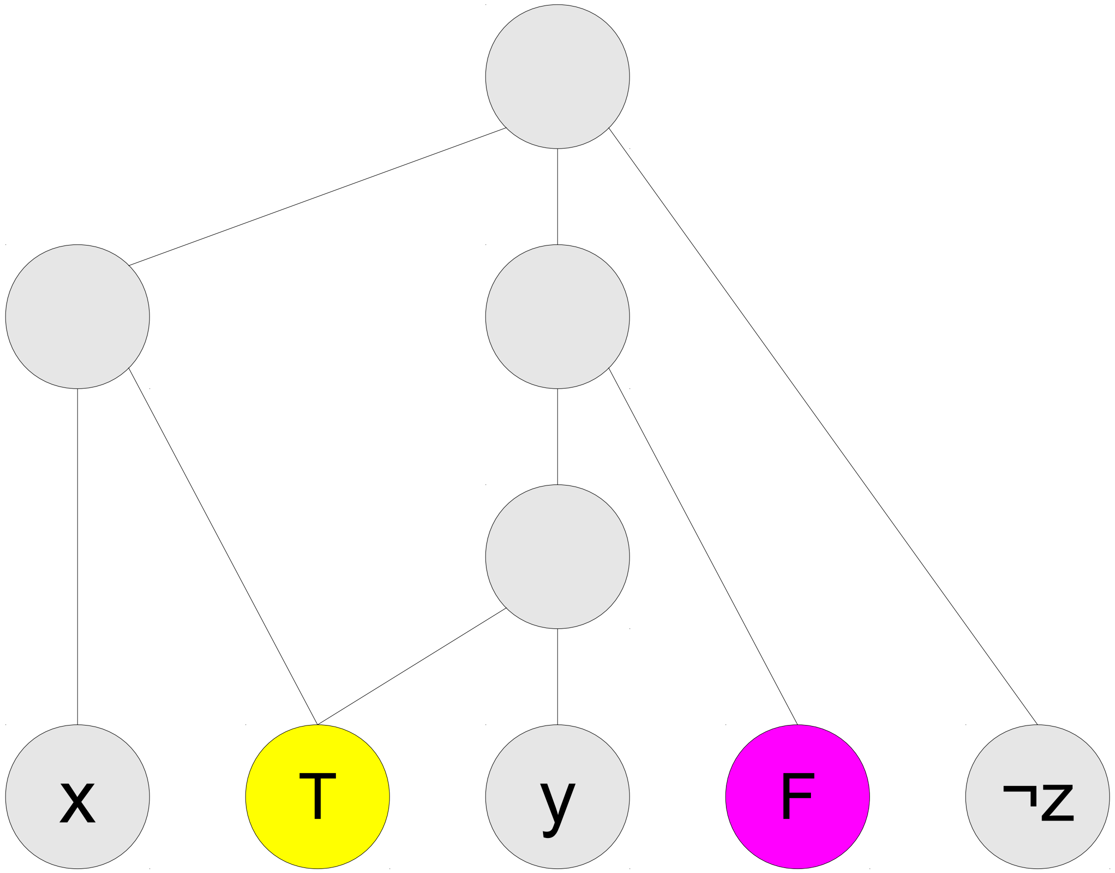
(x ∨ y ∨ ¬z)



This node is colorable iff one of the inputs is the same color as T

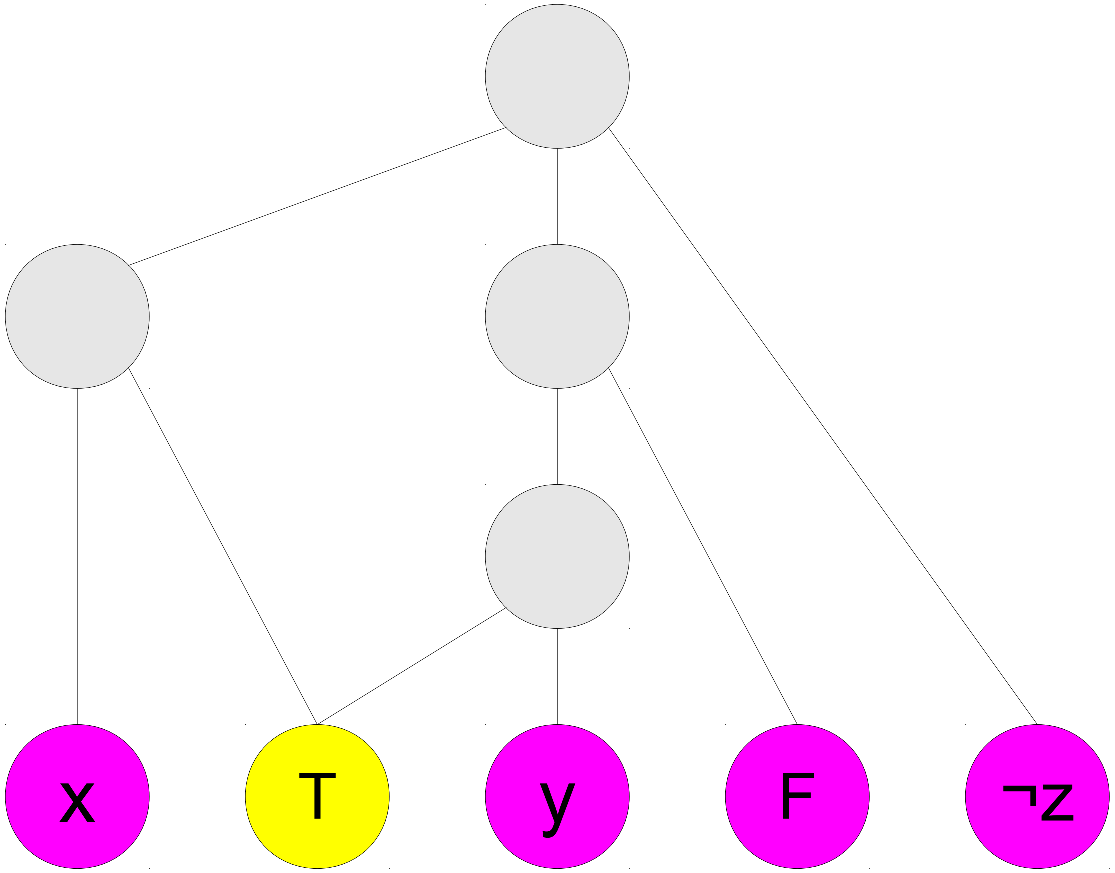
Gadget Three: Clause Satisfiability

(x ∨ y ∨ ¬z)



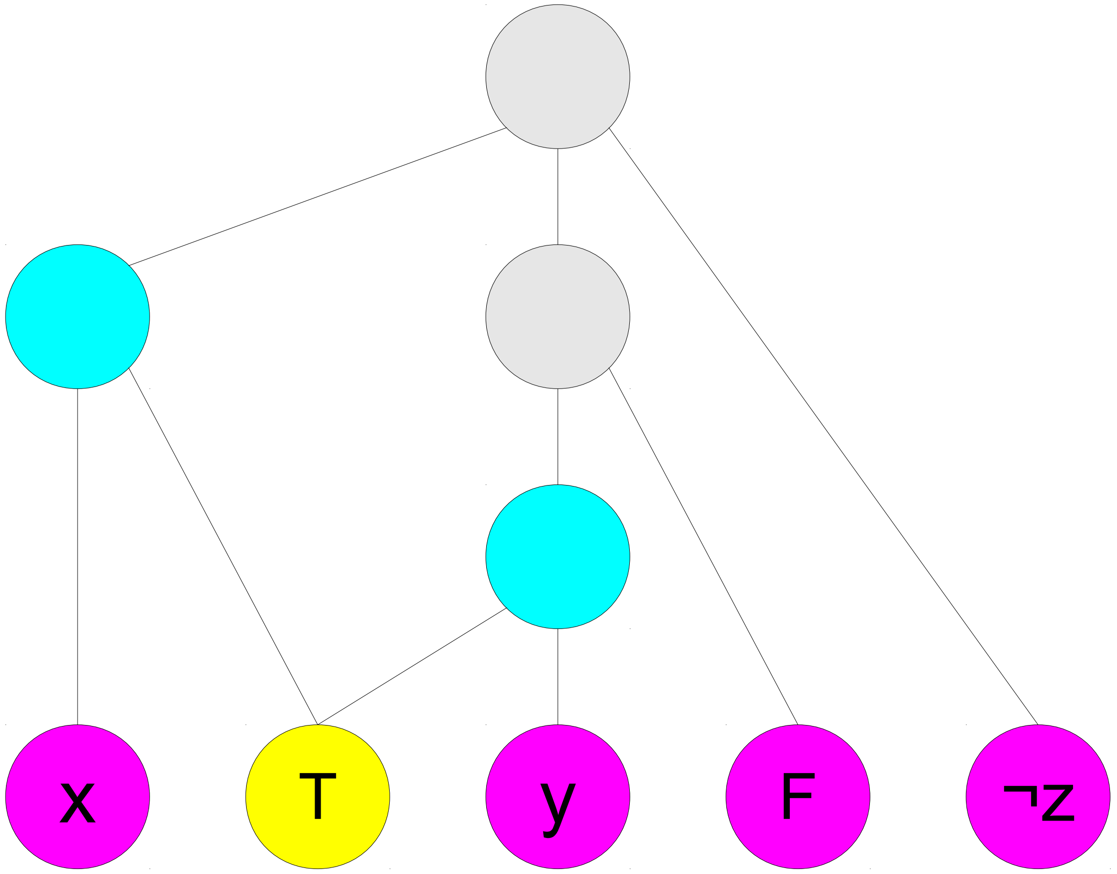
Gadget Three: Clause Satisfiability

(x ∨ y ∨ ¬z)



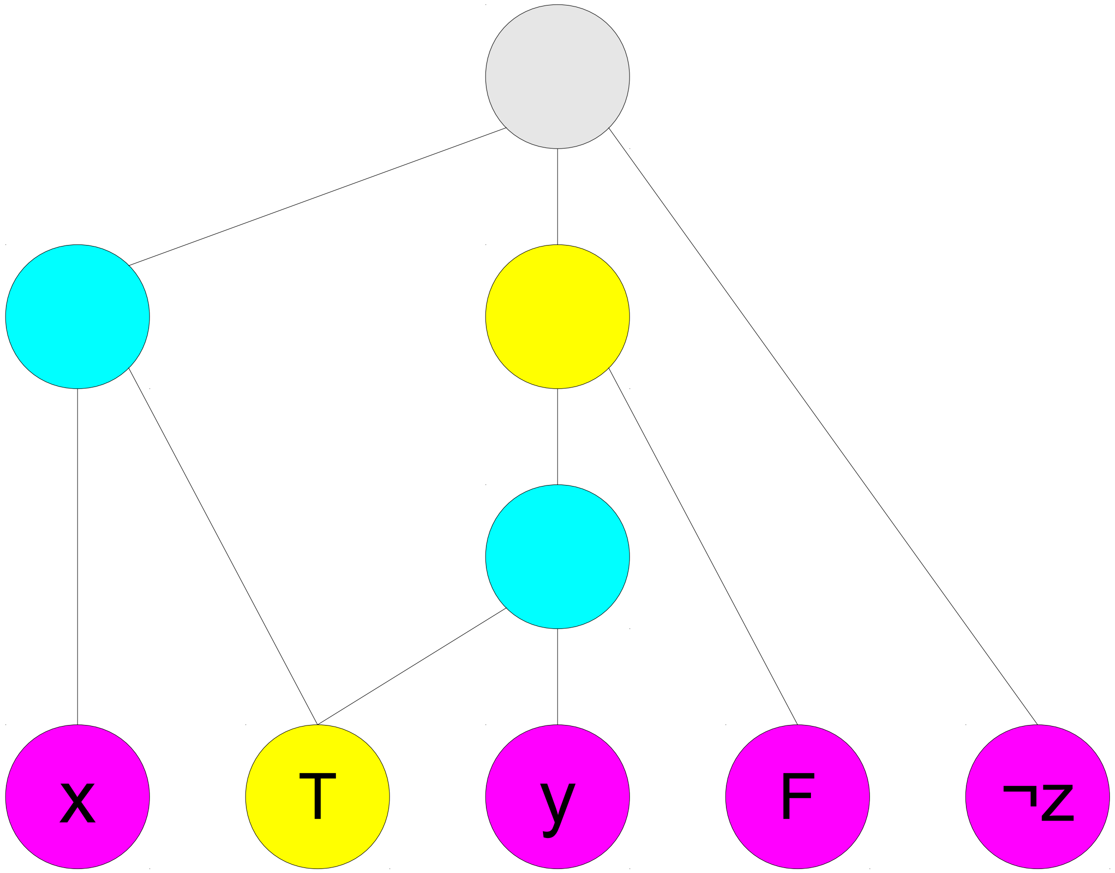
Gadget Three: Clause Satisfiability

(x ∨ y ∨ ¬z)



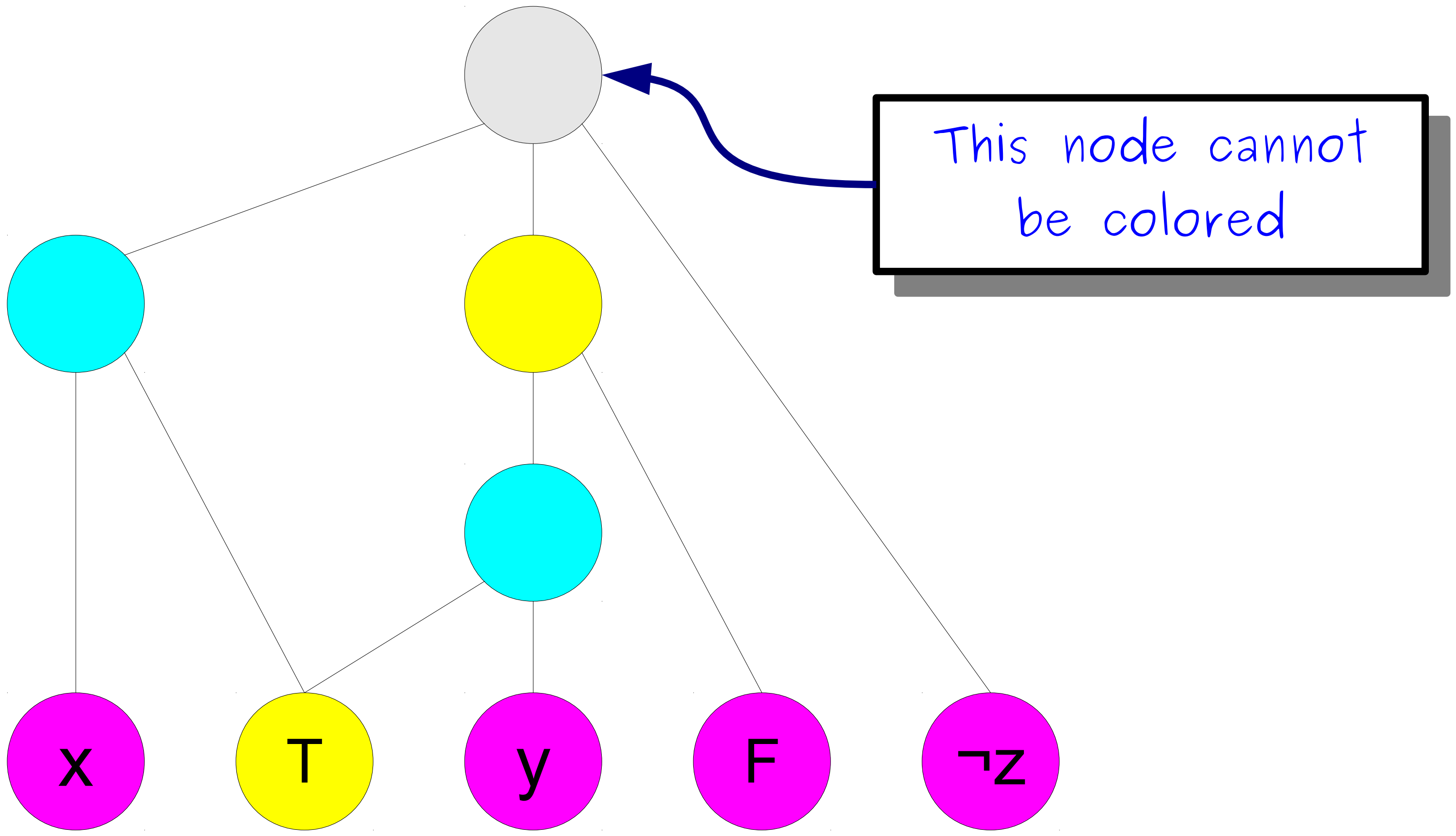
Gadget Three: Clause Satisfiability

(x ∨ y ∨ ¬z)



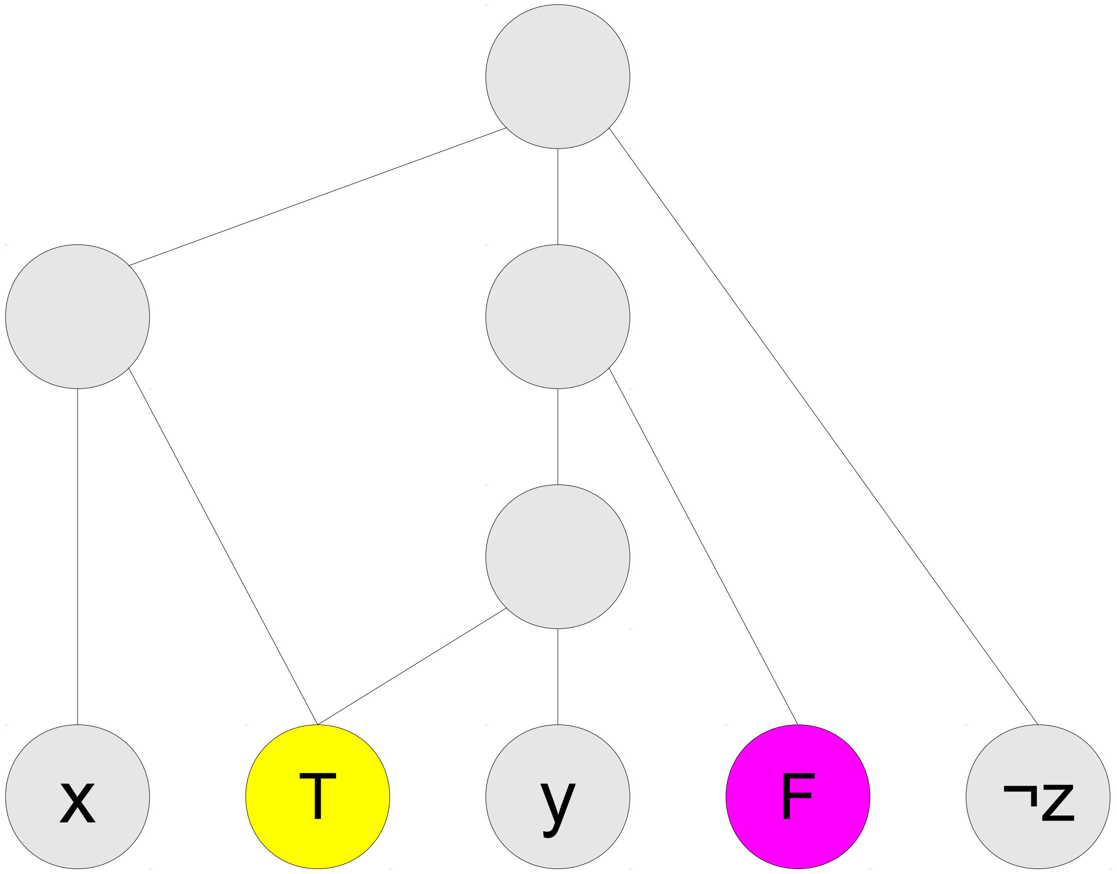
Gadget Three: Clause Satisfiability

(x ∨ y ∨ ¬z)



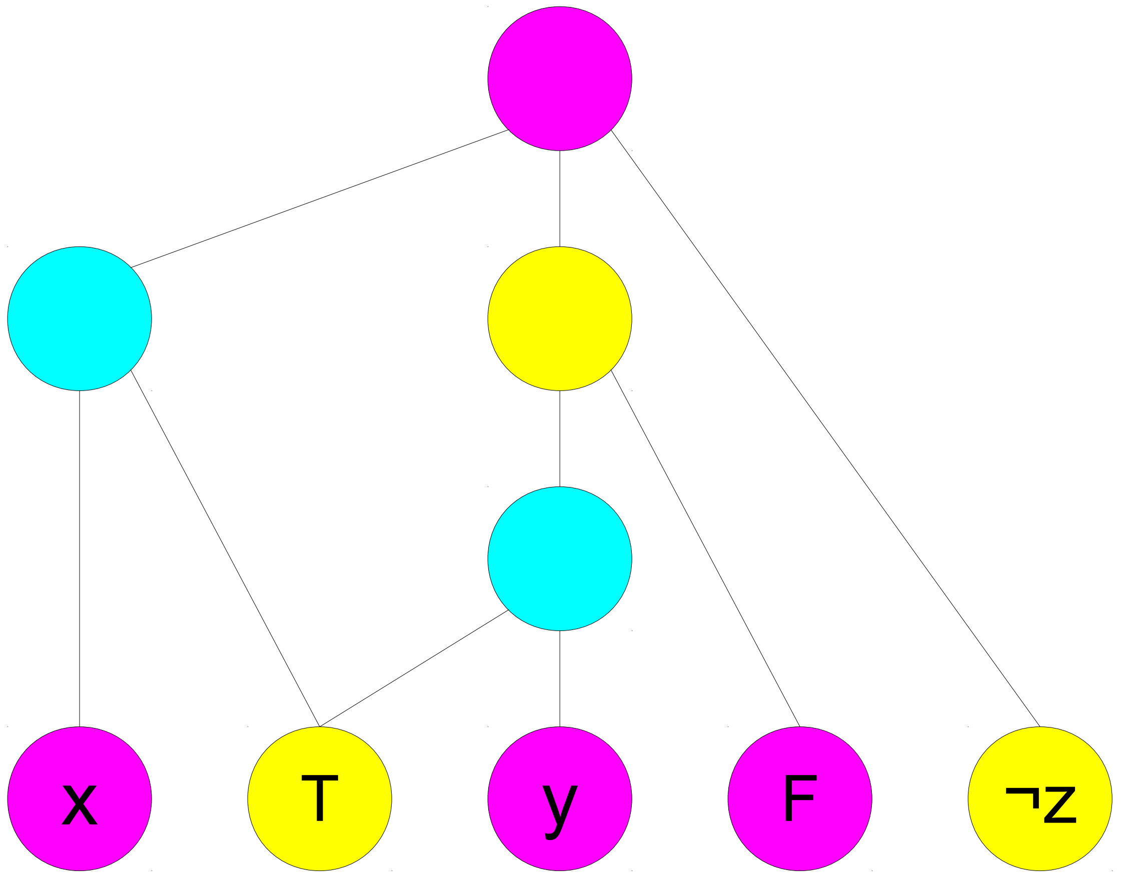
Gadget Three: Clause Satisfiability

(x ∨ y ∨ ¬z)



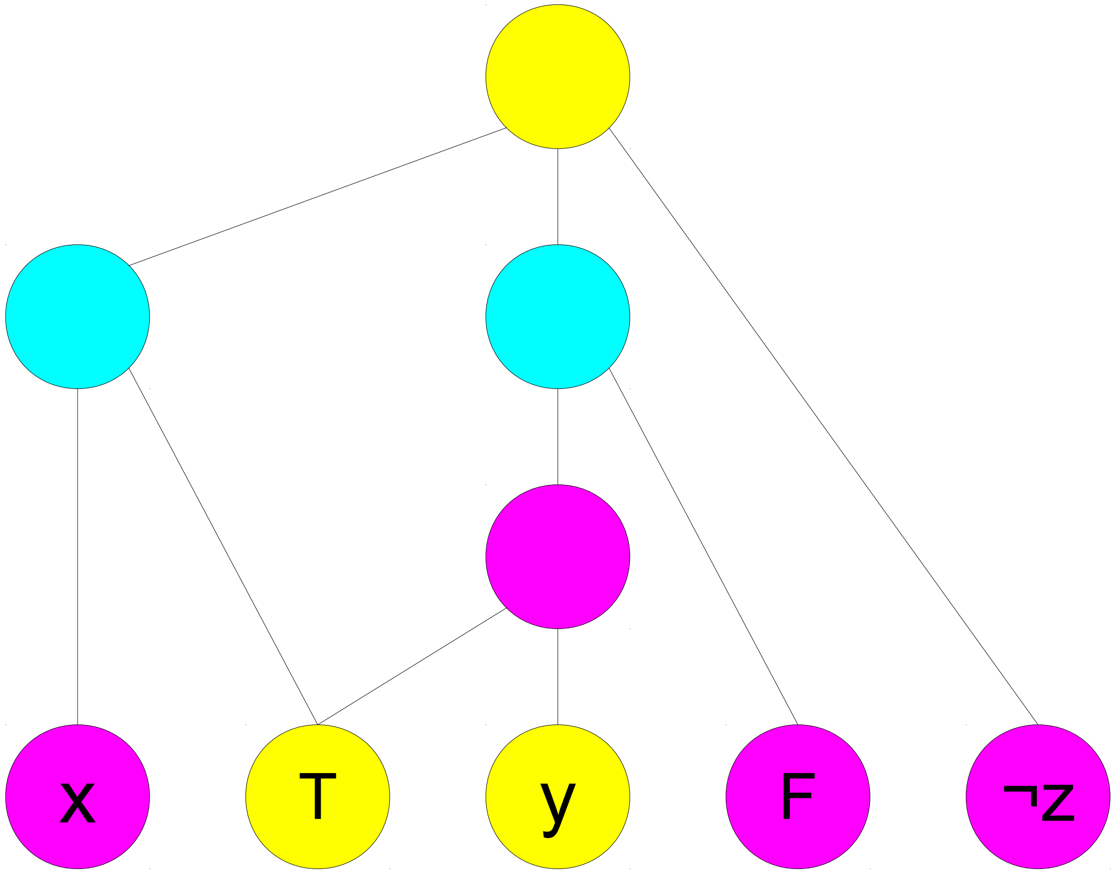
Gadget Three: Clause Satisfiability

(x ∨ y ∨ ¬z)



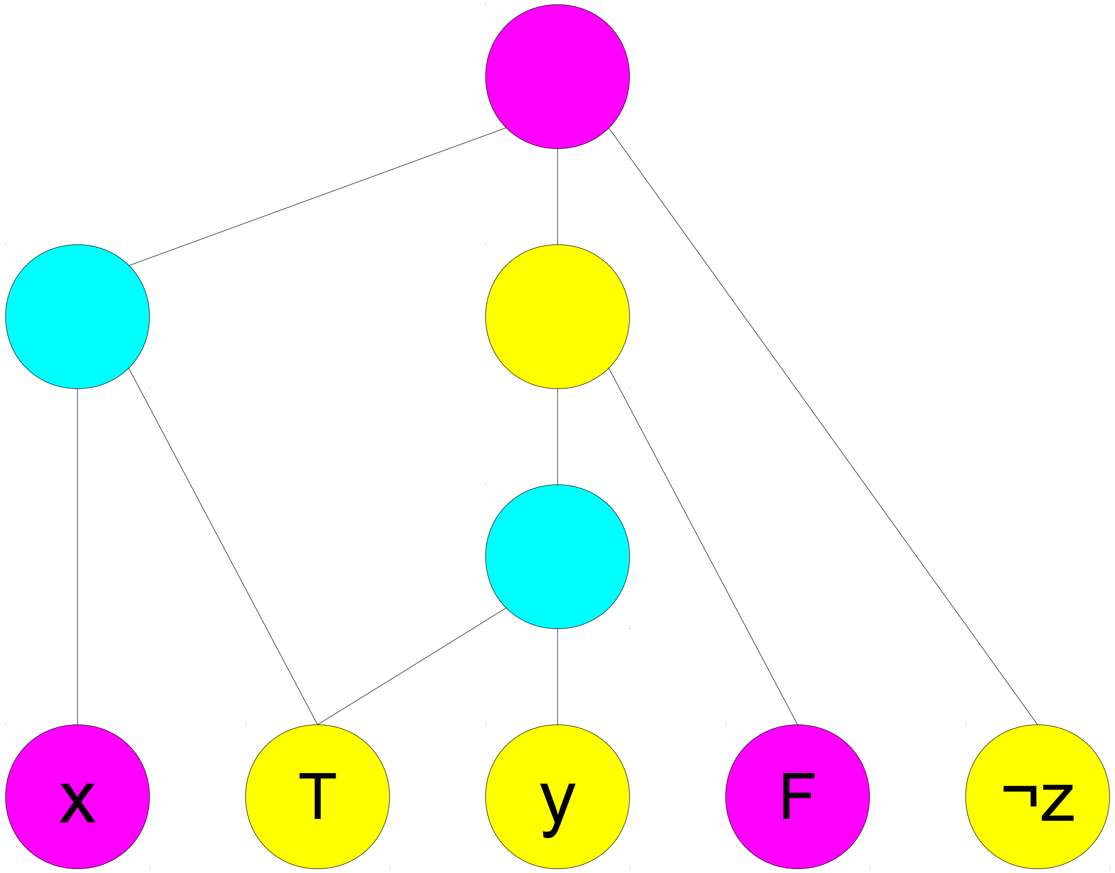
Gadget Three: Clause Satisfiability

$(x \lor y \lor \neg z)$

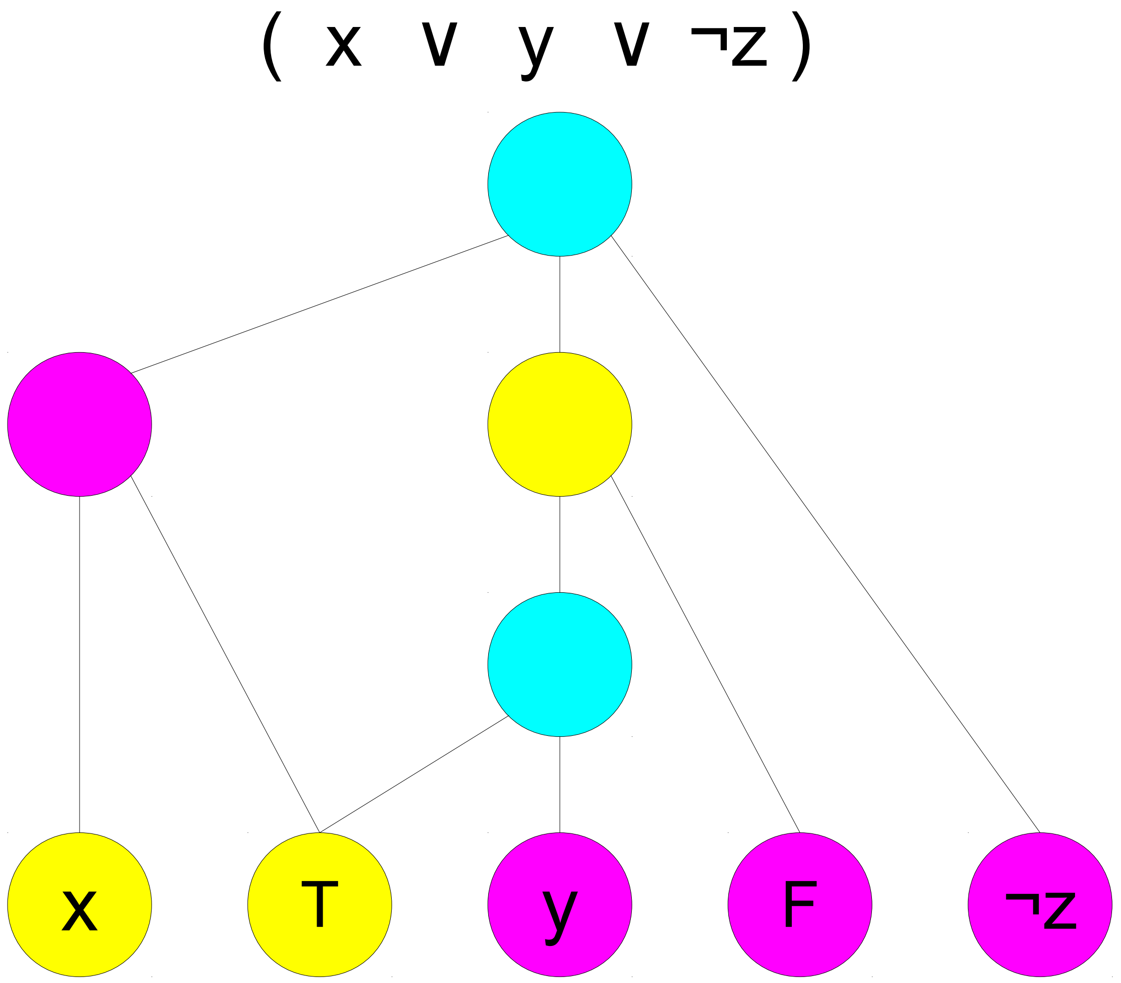


Gadget Three: Clause Satisfiability

(x ∨ y ∨ ¬z)

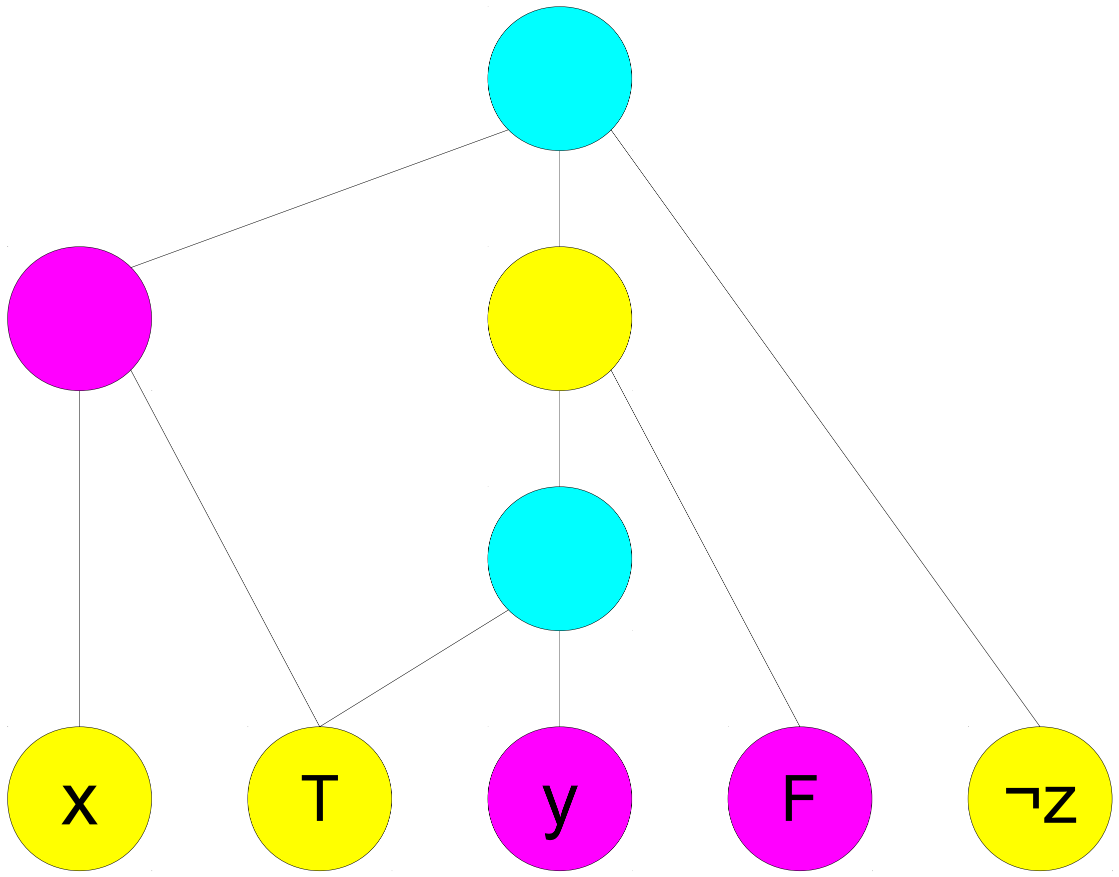


Gadget Three: Clause Satisfiability



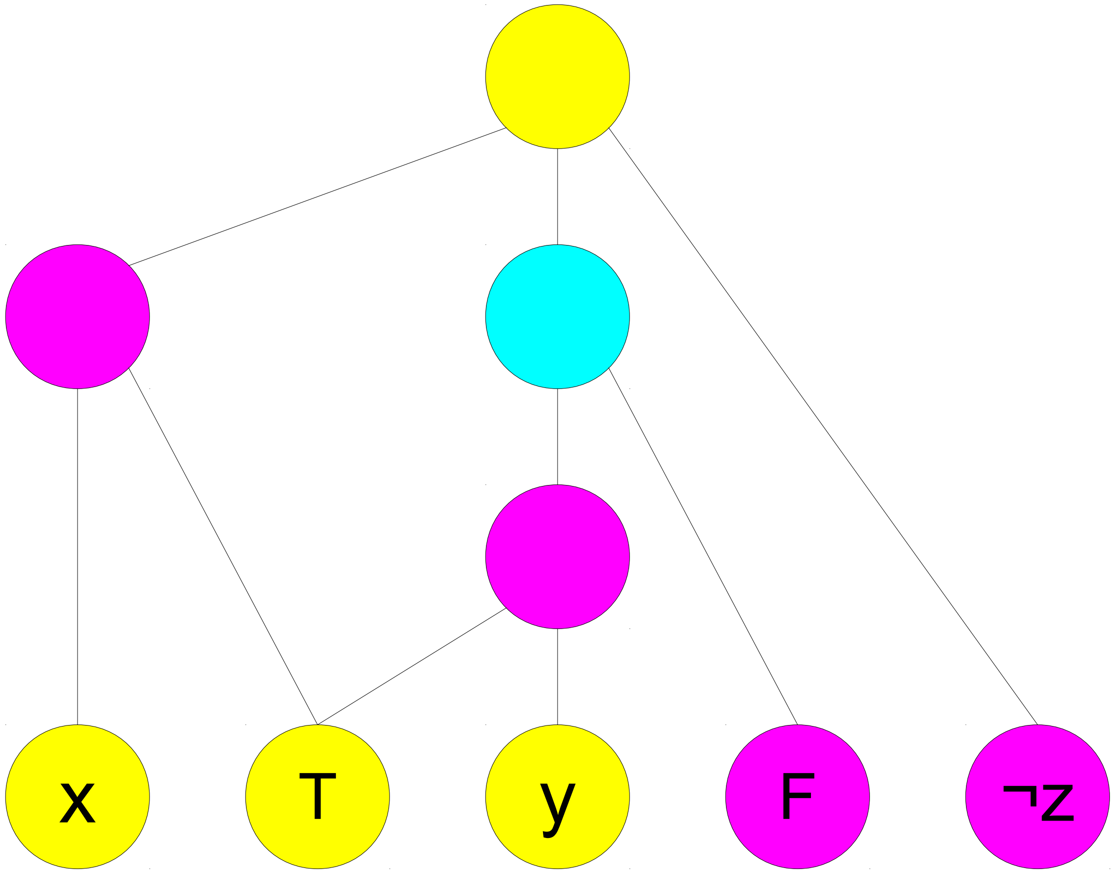
Gadget Three: Clause Satisfiability

(x ∨ y ∨ ¬z)



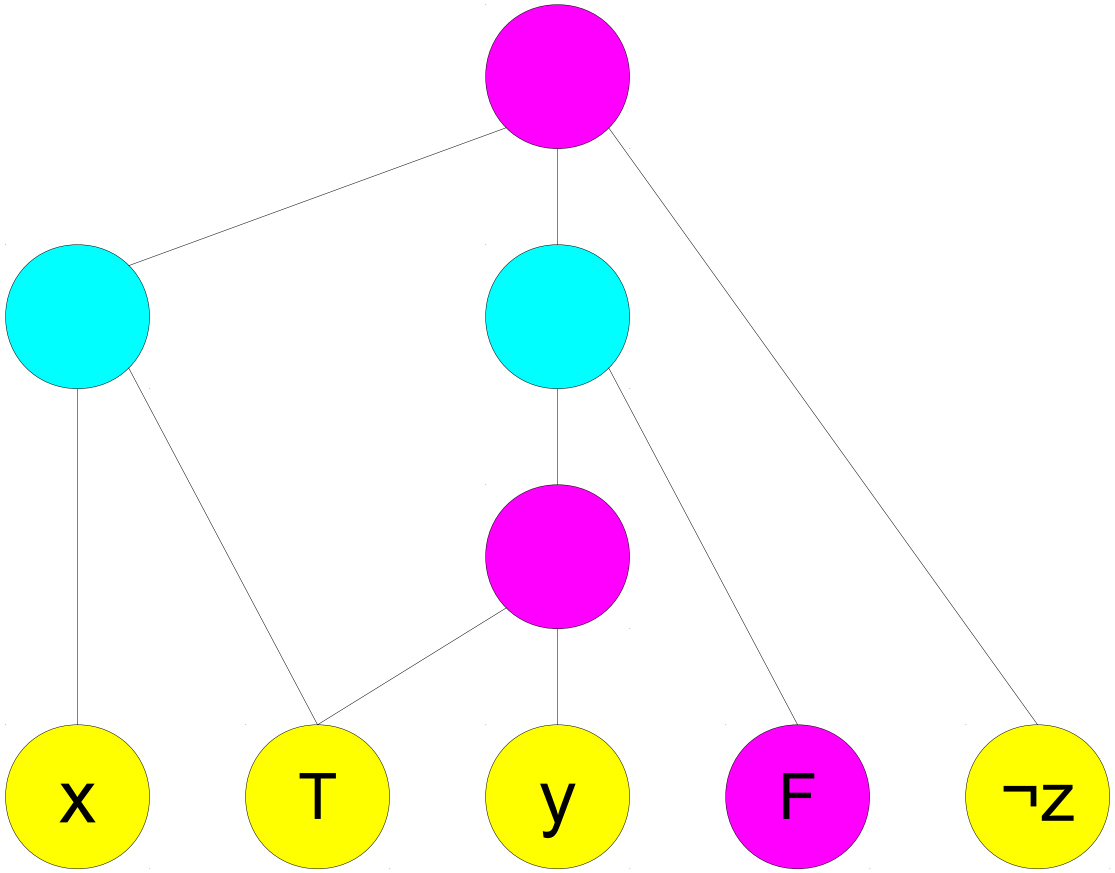
Gadget Three: Clause Satisfiability

(x ∨ y ∨ ¬z)



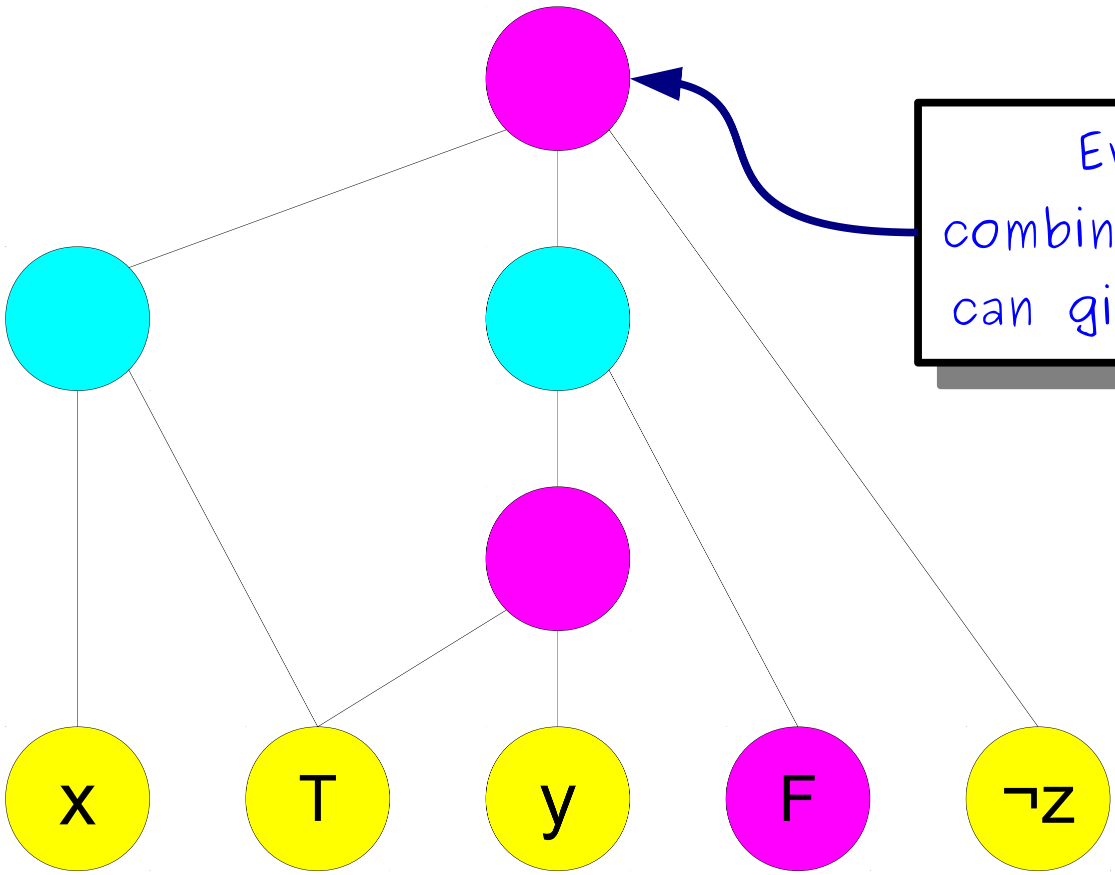
Gadget Three: Clause Satisfiability

(x ∨ y ∨ ¬z)



Gadget Three: Clause Satisfiability

$(x \lor y \lor \neg z)$

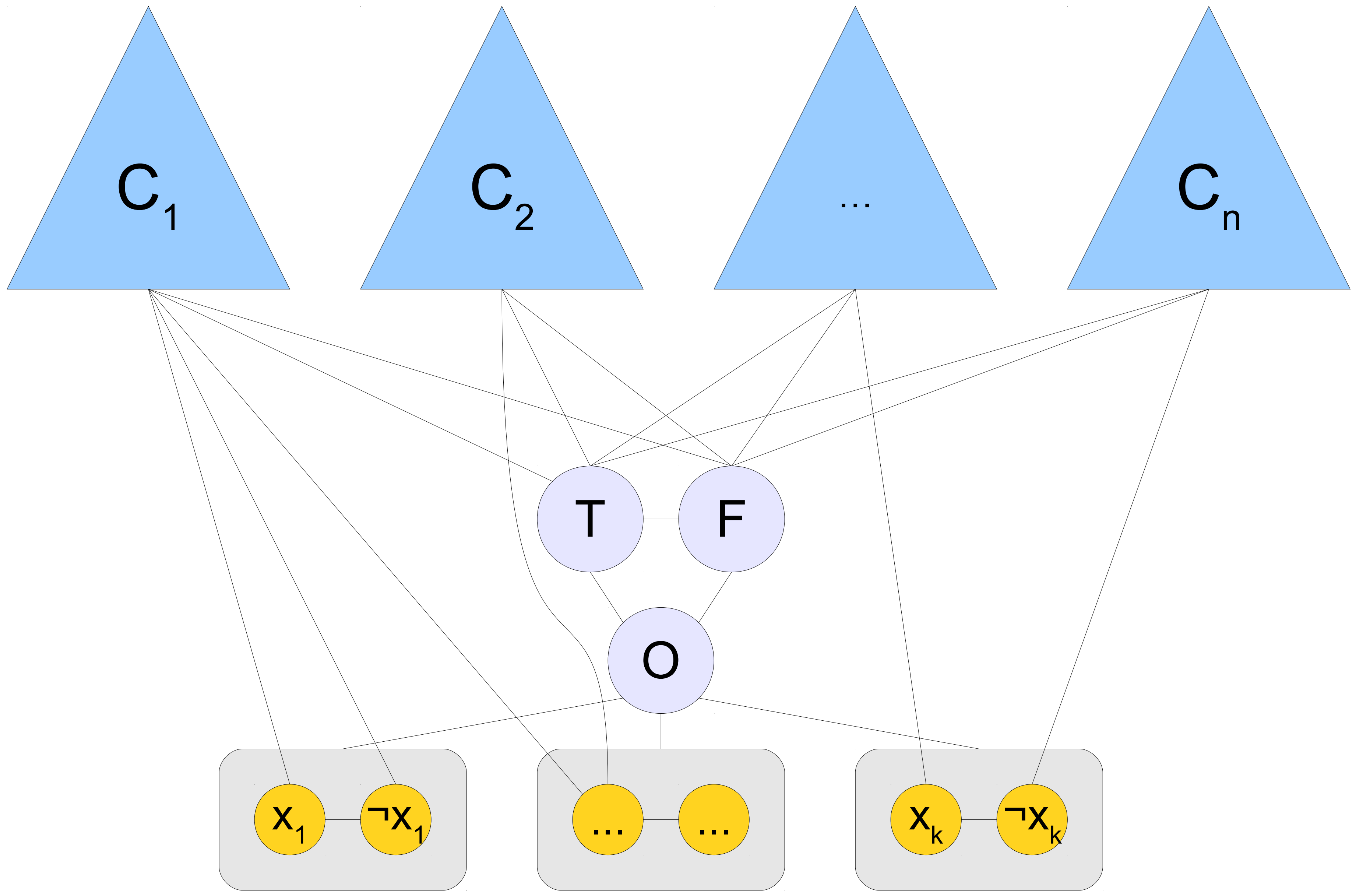


Every other combination of inputs can give this a color

Putting It All Together

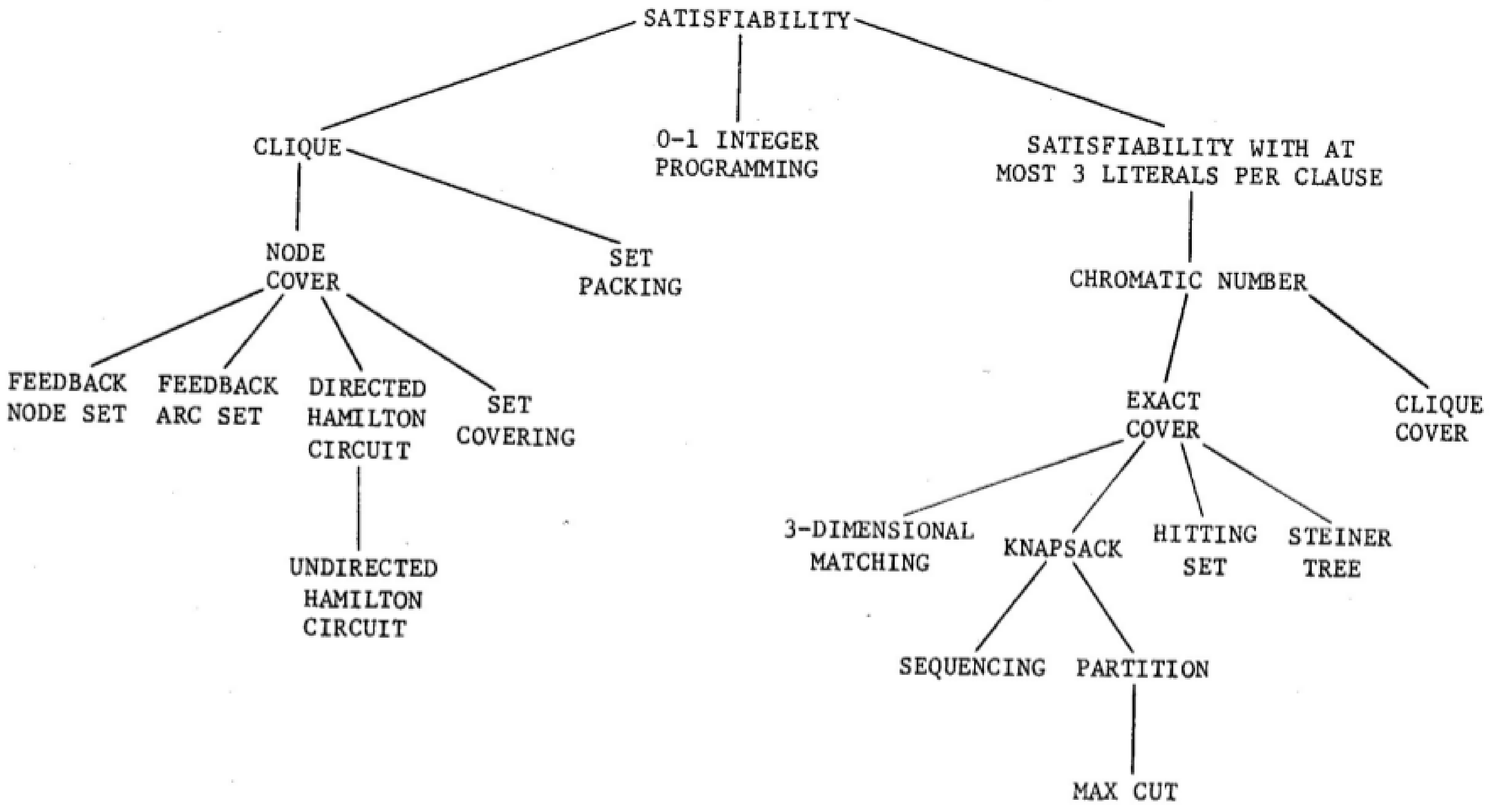
- Construct the first gadget so we have a consistent definition of true and false.
- For each variable v:
  - Construct nodes v and $\neg v$.
  - Add an edge between v and $\neg v$.
  - Add an edge between v and O and between $\neg v$ and O.
- For each clause C:
  - Construct the earlier gadget from C by adding in the extra nodes and edges.

Putting It All Together



NP-Completeness Recap

Karp's 21 **NP**-Complete Problems



Sample **NP**-Complete Problems

- Given a graph, is that graph 3-colorable?
- Given a graph, does that graph contain a Hamiltonian path?
- Given a set of cities and a set of substation locations, can you provide power to all the cities using at most k substations?
- Given a set of jobs and workers who can perform those tasks in parallel, can you complete all the jobs in at most T units of time?
- Given a set of numbers S, can you split S into two disjoint sets with the same sum?
- *And many, many more!*

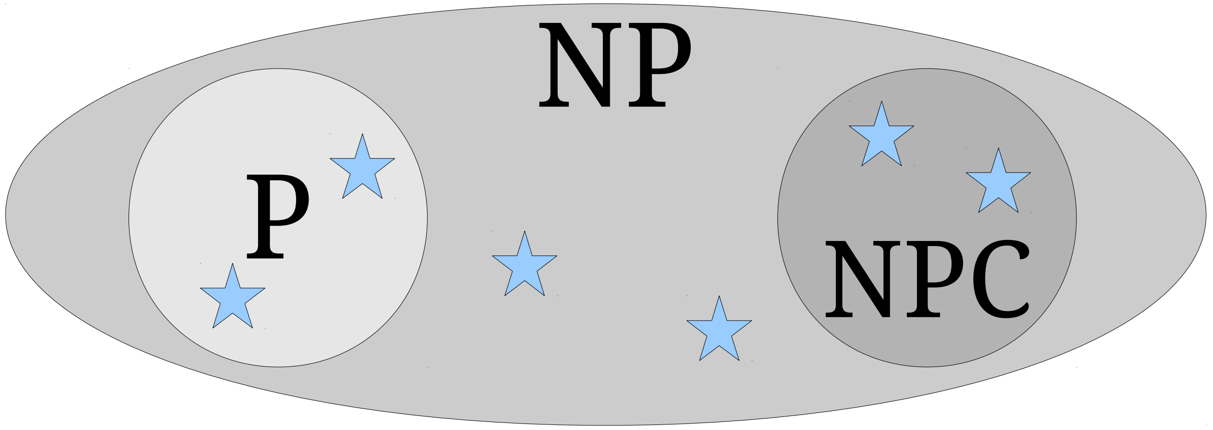
A Feel for **NP**-Completeness

- There are **NP**-complete problems in
  - formal logic (SAT),
  - graph theory (3-colorability),
  - operations research (job scheduling),
  - number theory (partition problem),
  - *and basically everywhere*.
- *You will undoubtedly encounter **NP**-complete problems in the real world.*

Some Recent News

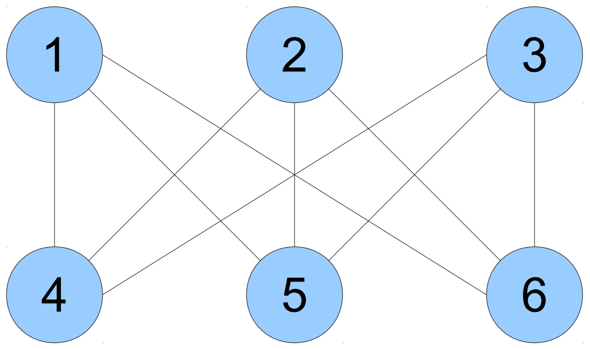
Intermediate Problems

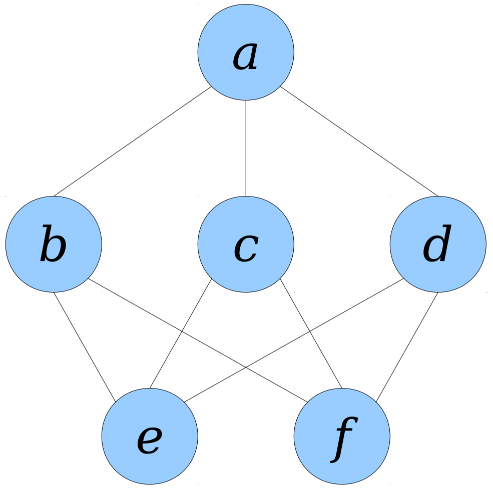
- With few exceptions, every problem we've discovered in **NP** has either
  - definitely been proven to be in **P**, or
  - definitely been proven to be **NP**-complete.
- A problem that's **NP**, not in **P**, but not **NP**-complete is called ***NP-intermediate***.
- ***Theorem (Ladner):*** There are **NP**-intermediate problems if and only if **P ≠ NP**.



Graph Isomorphism

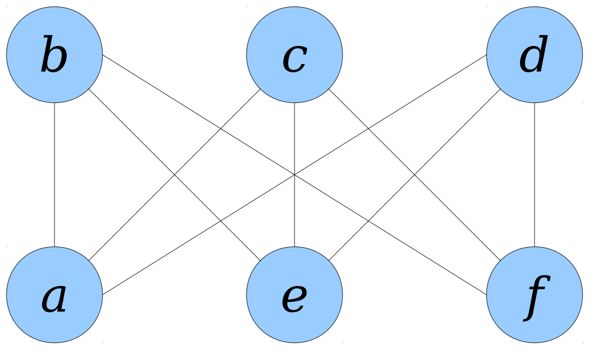
- The ***graph isomorphism problem*** is the following: given two graphs G_1 and G_2, is there a way to relabel the nodes in G_1 so that the resulting graph is G_2?
- This problem is in **NP**, but no one knows whether it's in **P**, whether it's **NP**-complete, or whether it's **NP**-intermediate.

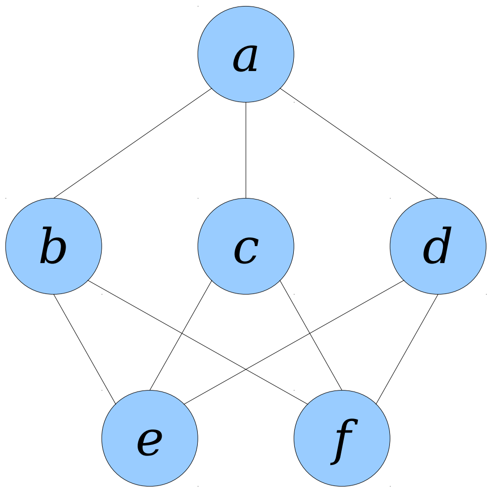




Graph Isomorphism

- The ***graph isomorphism problem*** is the following: given two graphs G_1 and G_2, is there a way to relabel the nodes in G_1 so that the resulting graph is G_2?
- This problem is in **NP**, but no one knows whether it's in **P**, whether it's **NP**-complete, or whether it's **NP**-intermediate.





Recent News

- Last December, a mathematician named Laszlo Babai is presented a proof that graph isomorphism can be solved in “almost” polynomial time.
- This is a huge deal for a number of reasons:
  - This particular problem has resisted any improvements for a long time. Remember – we don't have many good tools for reasoning about **P** and **NP**!
  - The particular technique he's using is somewhat novel and interesting. Remember – we don't have many good tools for reasoning about **P** and **NP**!
  - This probably means that the problem will eventually be proven to be in **P**, meaning that we'll probably develop some really clever new algorithmic technique while solving it. Remember – we don't have many good tools for reasoning about **P** and **NP**!

More Recent News

Beyond **P** and **NP**

- The classes **P** and **NP** are important from both a theoretical and practical perspective, but they're not the only complexity classes out there.
- Both **P** and **NP** are subsets of a class called ***PSPACE*** (**p**olynomial **space**) consisting of all problems that can be solved with a polynomial amount of memory.

What is **PSPACE**?

- The most well-known examples of problems in **PSPACE** are questions of the form

  Given a two-player game, does the first player have a winning strategy?

- Intuitively, you can represent a game as a graph, where each node represents one state of the game and each edge represents a move made by a player.
- You can determine whether a player has a winning strategy by trying all possible strategies, where a "strategy" corresponds to choosing an action for each state in the game.

The Theory

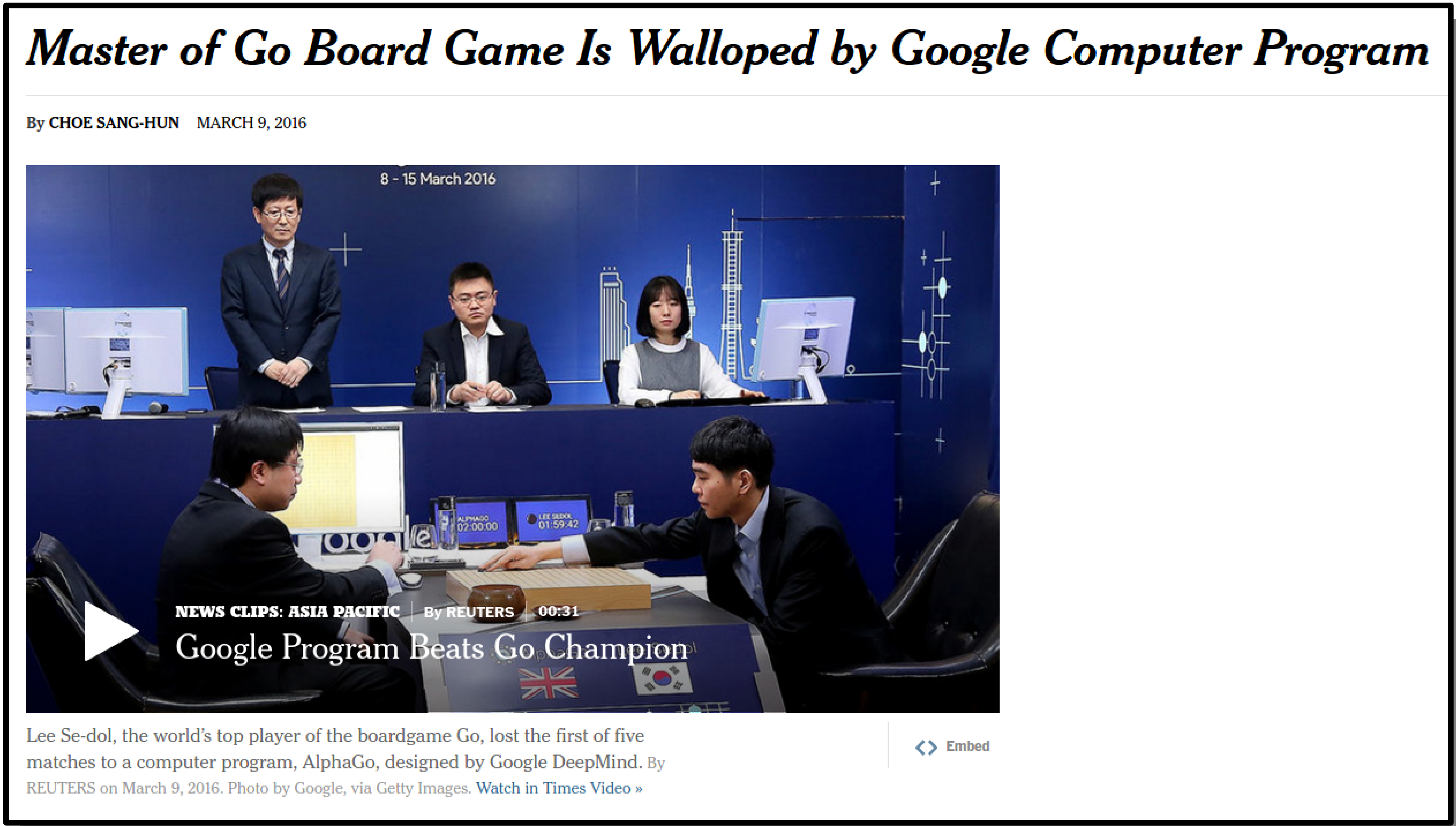
- We know that

  $$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$$

  but are not sure whether these are strict or non-strict subsets.
- It's *conjectured* that $\mathbf{P} \subsetneq \mathbf{NP}$ and that $\mathbf{NP} \subsetneq \mathbf{PSPACE}$, but no one knows the answer to either question!
- The suspicion is that **NP** is smaller than **PSPACE**, since it's unclear how you'd easily be able to prove that a player in a game has a winning strategy.

Games are Hard

- Generally speaking, it's really hard to determine whether a player in a game has a winning strategy because the space to explore is so huge.
- As an example, the game Go has about 10^{172} possible different game states.
- When I was an undergrad here, I was told that it was basically impossible for computers to play Go and that we'd never see good computer Go players.
- Well, then this happened...



Master of Go Board Game Is Walloped by Google Computer Program

By CHOE SANG-HUN MARCH 9, 2016

NEWS CLIPS: ASIA PACIFIC | By REUTERS | 00:31

Google Program Beats Go Champion

Lee Se-dol, the world's top player of the boardgame Go, lost the first of five matches to a computer program, AlphaGo, designed by Google DeepMind. By REUTERS on March 9, 2016. Photo by Google, via Getty Images. Watch in Times Video »

Embed

http://www.nytimes.com/2016/03/10/world/asia/google-alphago-lee-se-dol.html?_r=0

Just because it's (PSPACE-)hard doesn't mean that you shouldn't try.

Next Time

- **The Big Picture**
  - How do all of our results relate to one another?
- **Where to Go from Here**
  - What's next in CS theory?