

NP-Completeness

Part II

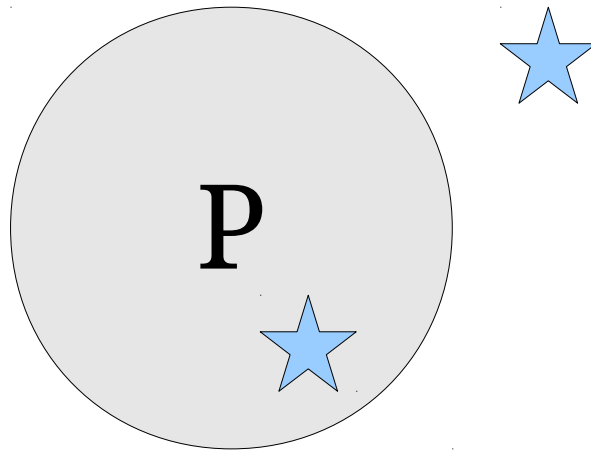
Outline for Today

- **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.

Recap from Last Time

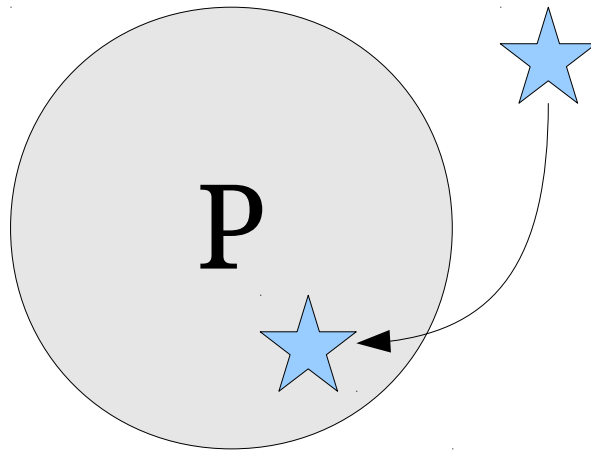
Polynomial-Time Reductions

- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.



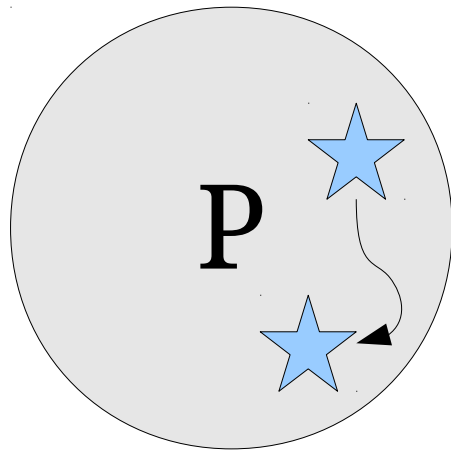
Polynomial-Time Reductions

- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.



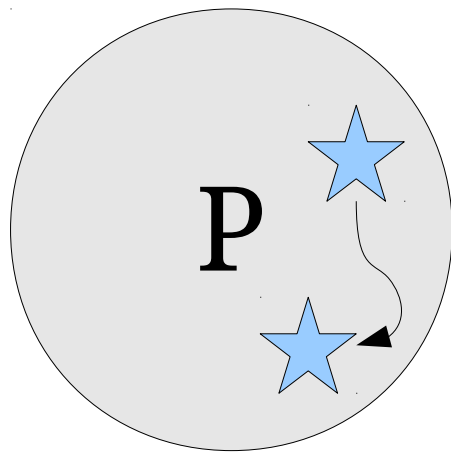
Polynomial-Time Reductions

- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.



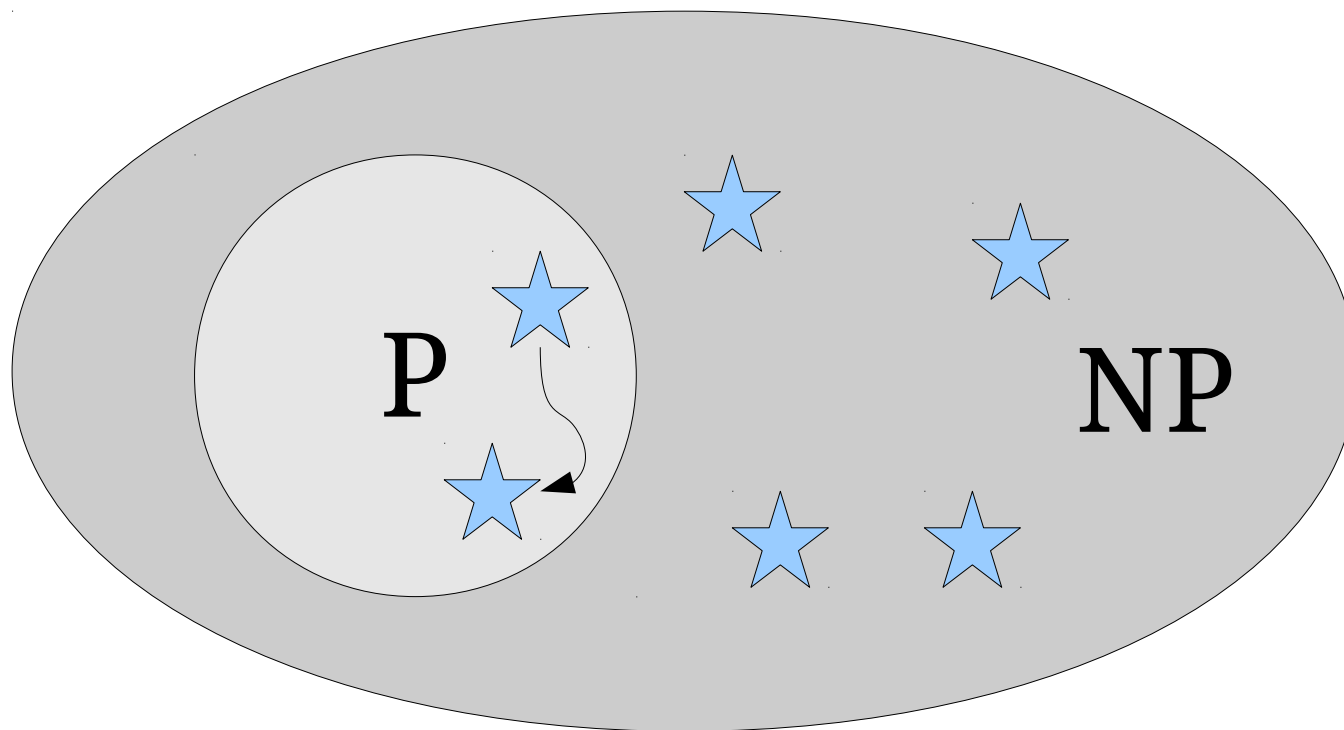
Polynomial-Time Reductions

- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.
- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{NP}$, then $L_1 \in \mathbf{NP}$.



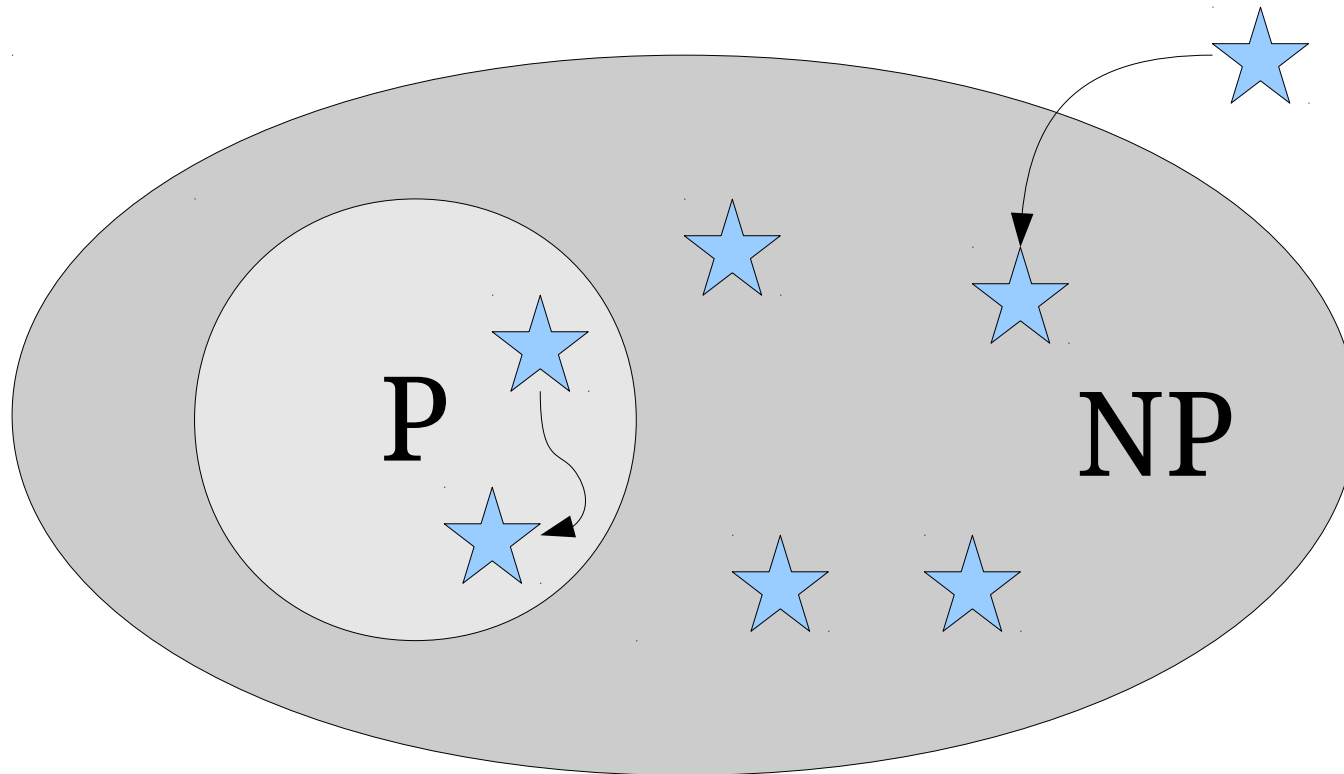
Polynomial-Time Reductions

- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.
- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{NP}$, then $L_1 \in \mathbf{NP}$.



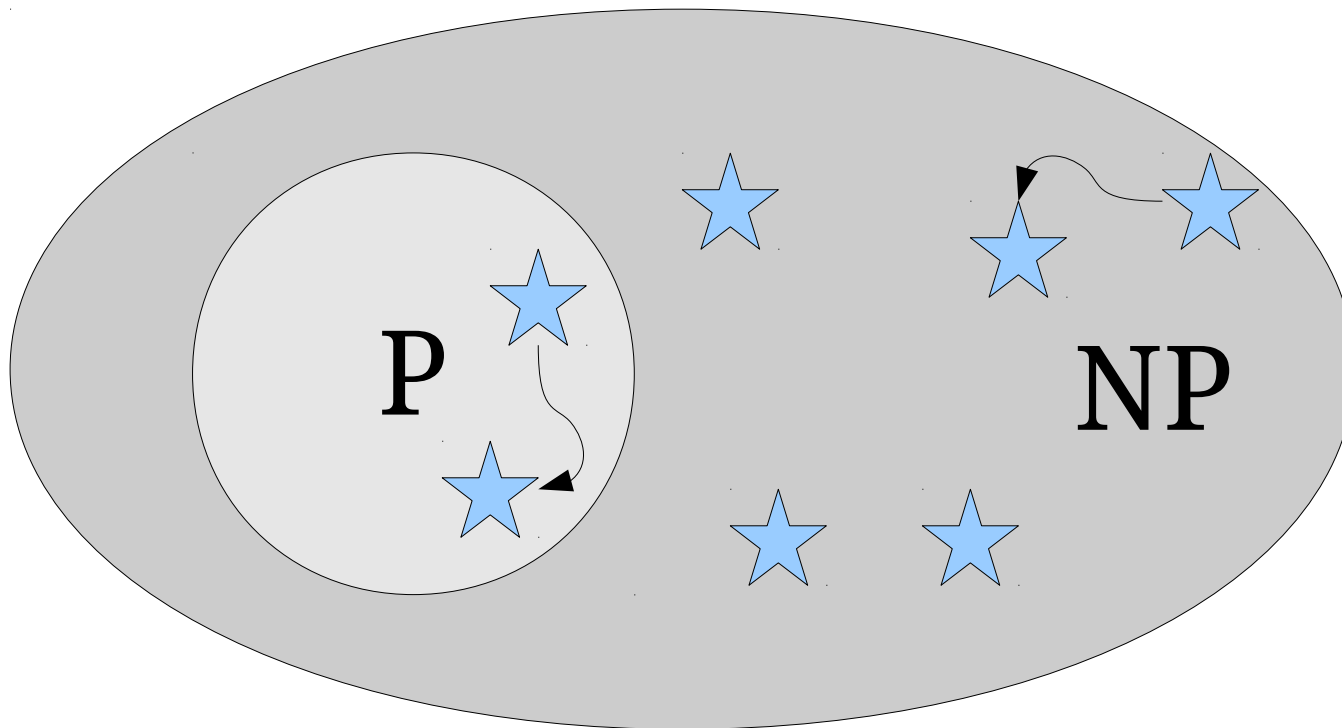
Polynomial-Time Reductions

- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.
- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{NP}$, then $L_1 \in \mathbf{NP}$.



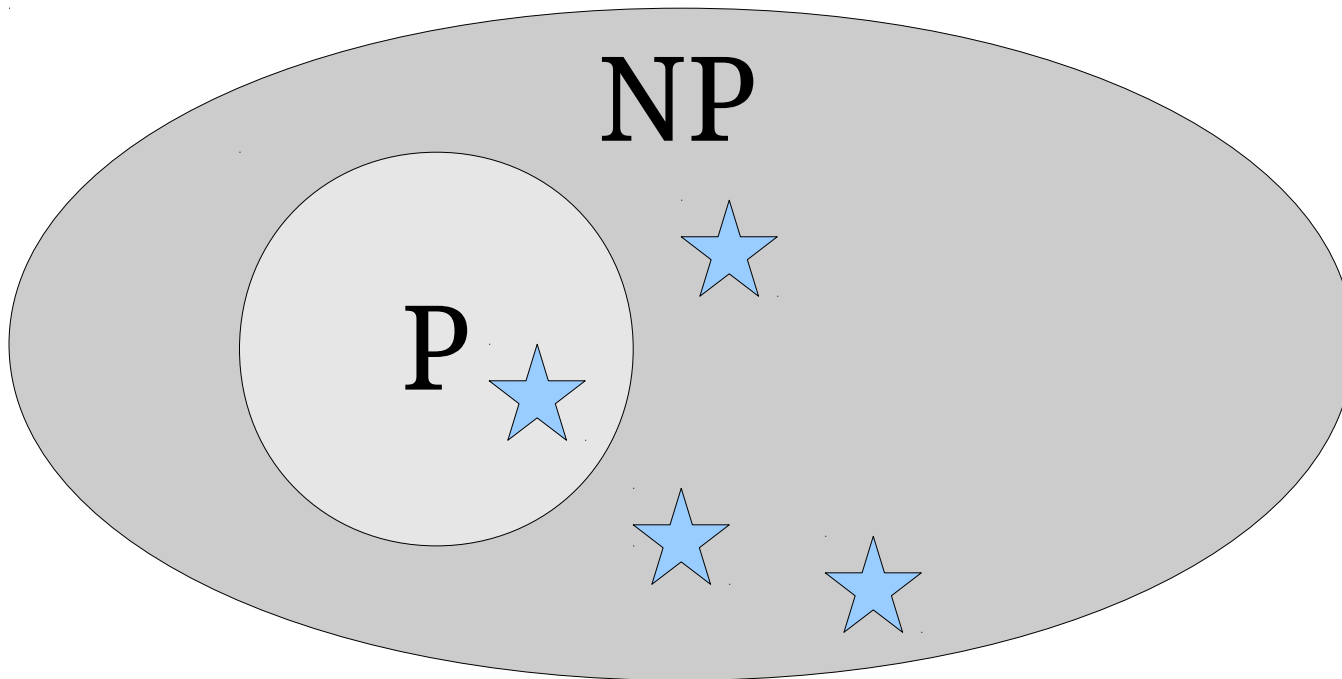
Polynomial-Time Reductions

- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$.
- If $L_1 \leq_P L_2$ and $L_2 \in \mathbf{NP}$, then $L_1 \in \mathbf{NP}$.



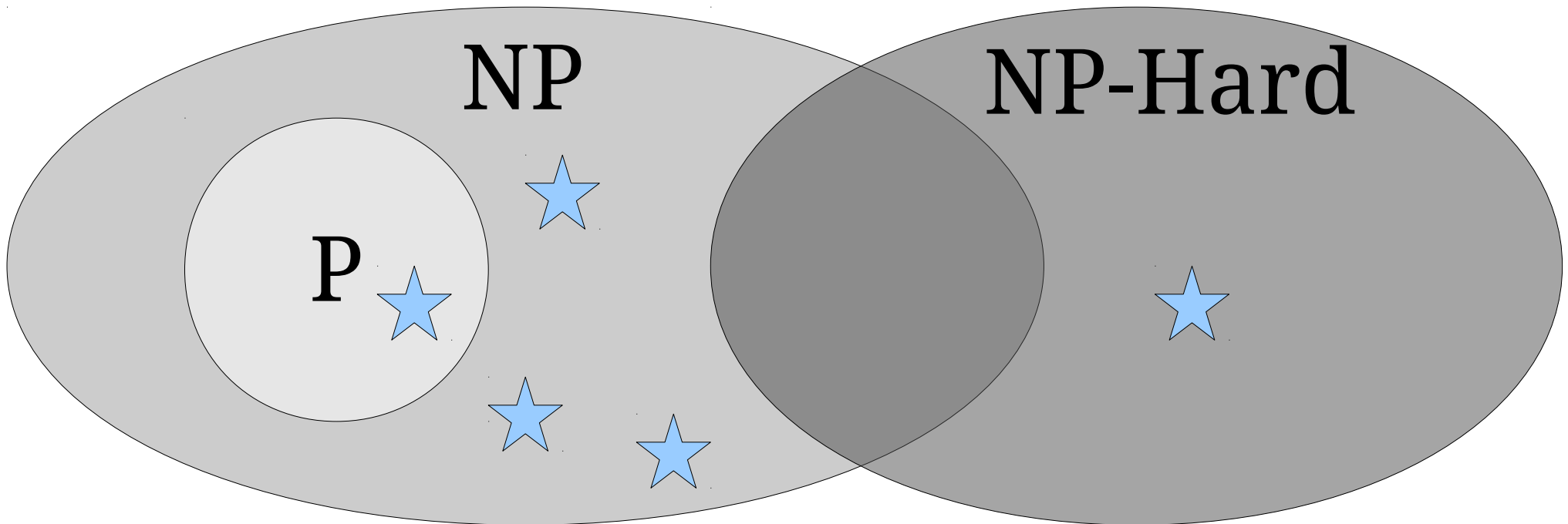
NP-Hardness

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



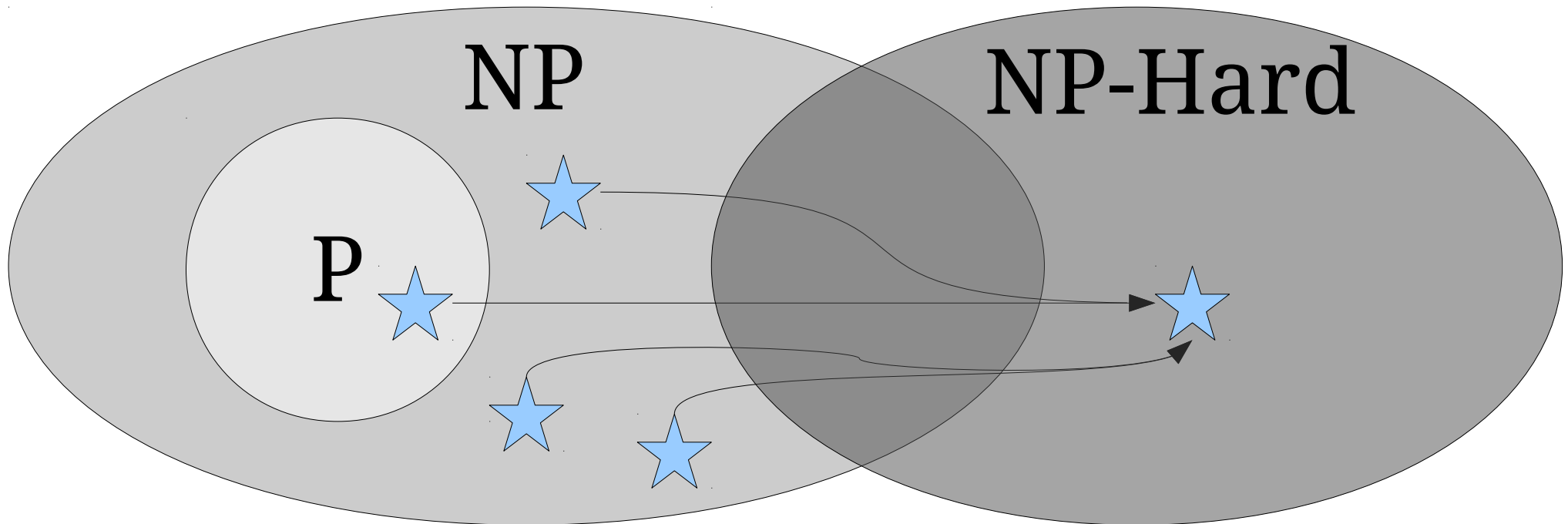
NP-Hardness

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



NP-Hardness

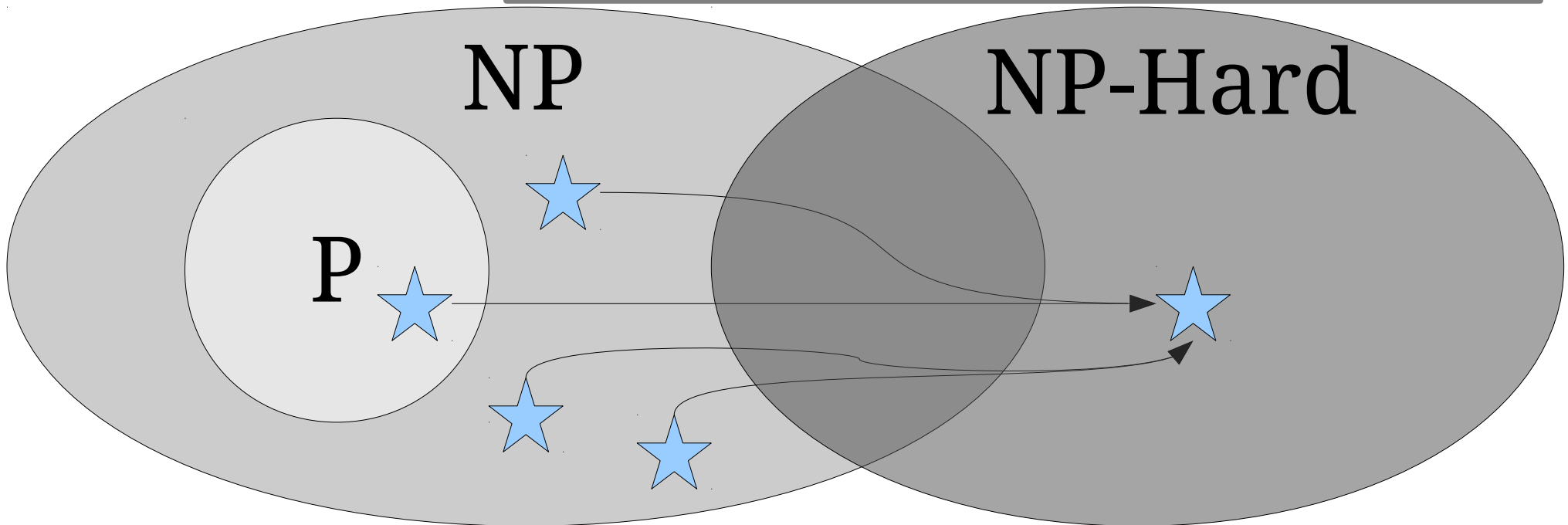
- A language L is called **NP-hard** if for *every* $L' \in \mathbf{NP}$, we have $L' \leq_p L$.



NP-Hardness

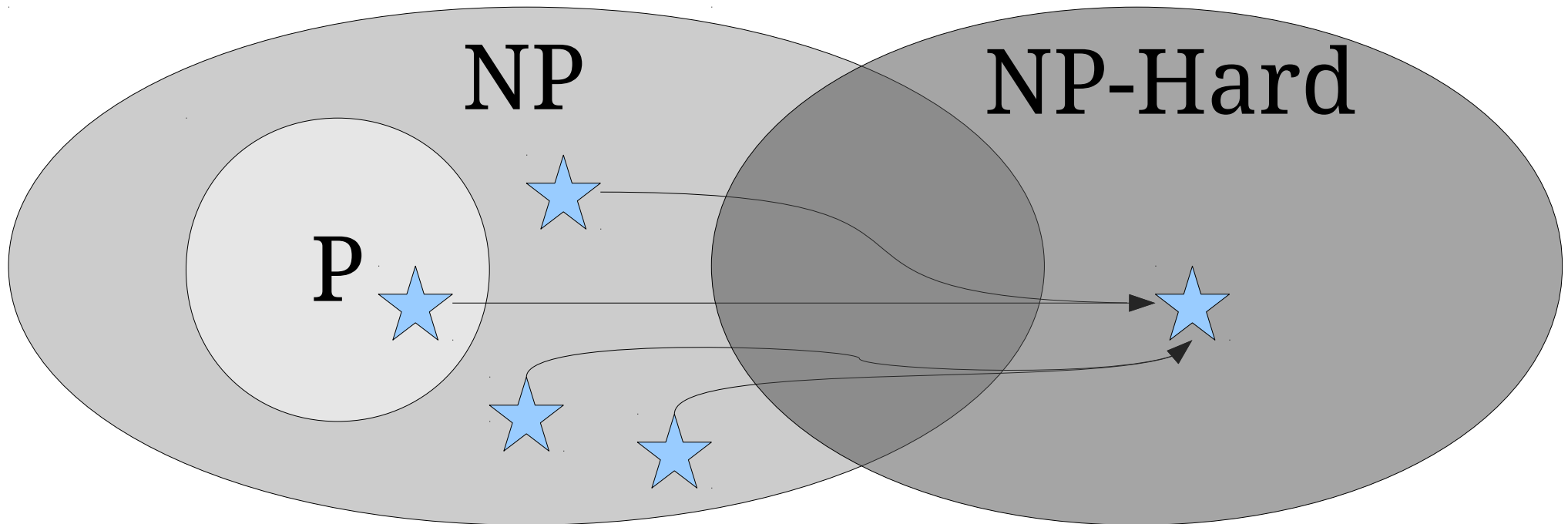
- A language L is called **NP-hard** if for *every* $L' \in \mathbf{NP}$, we have $L' \leq_p L$.

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**.



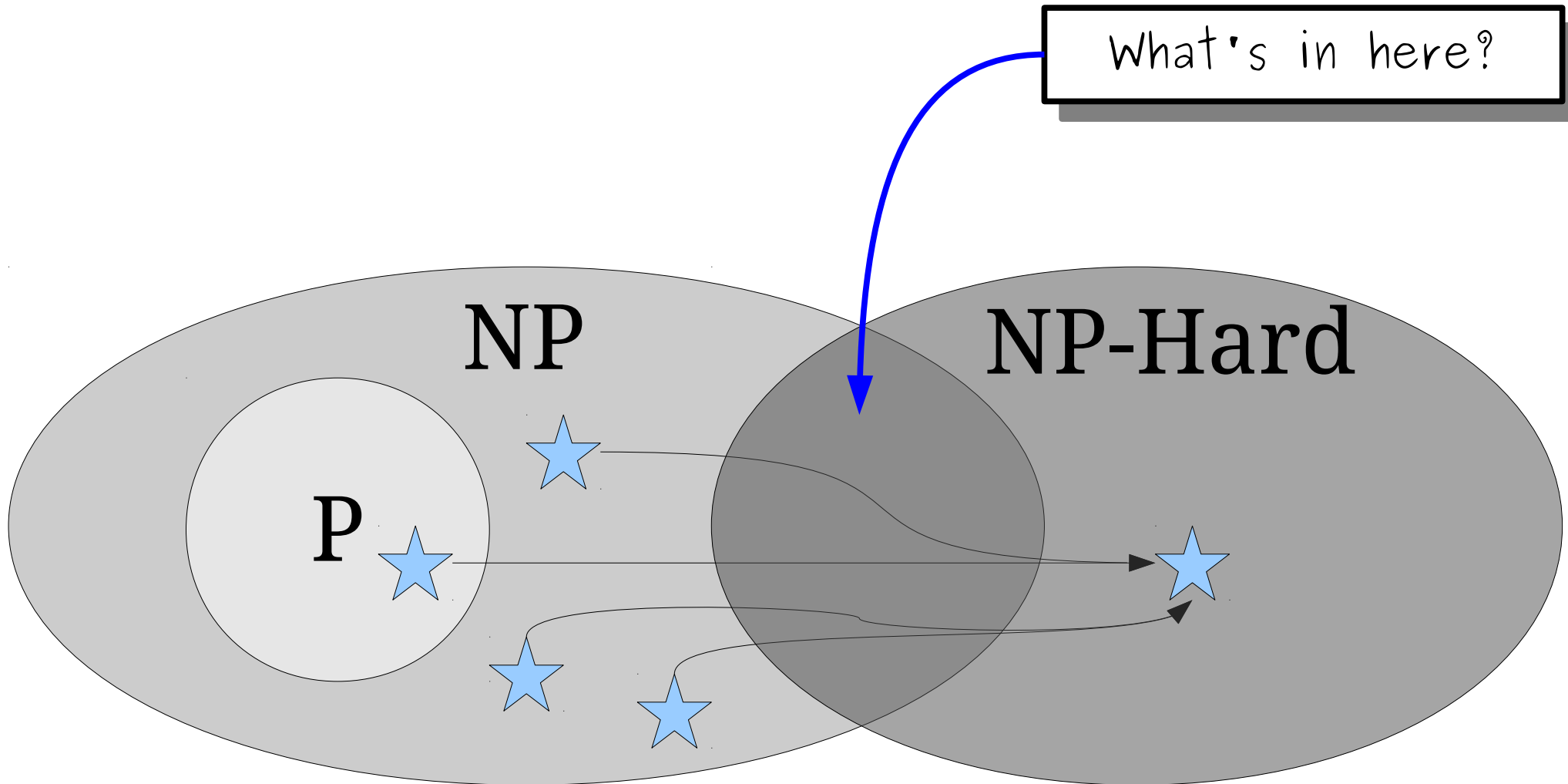
NP-Hardness

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



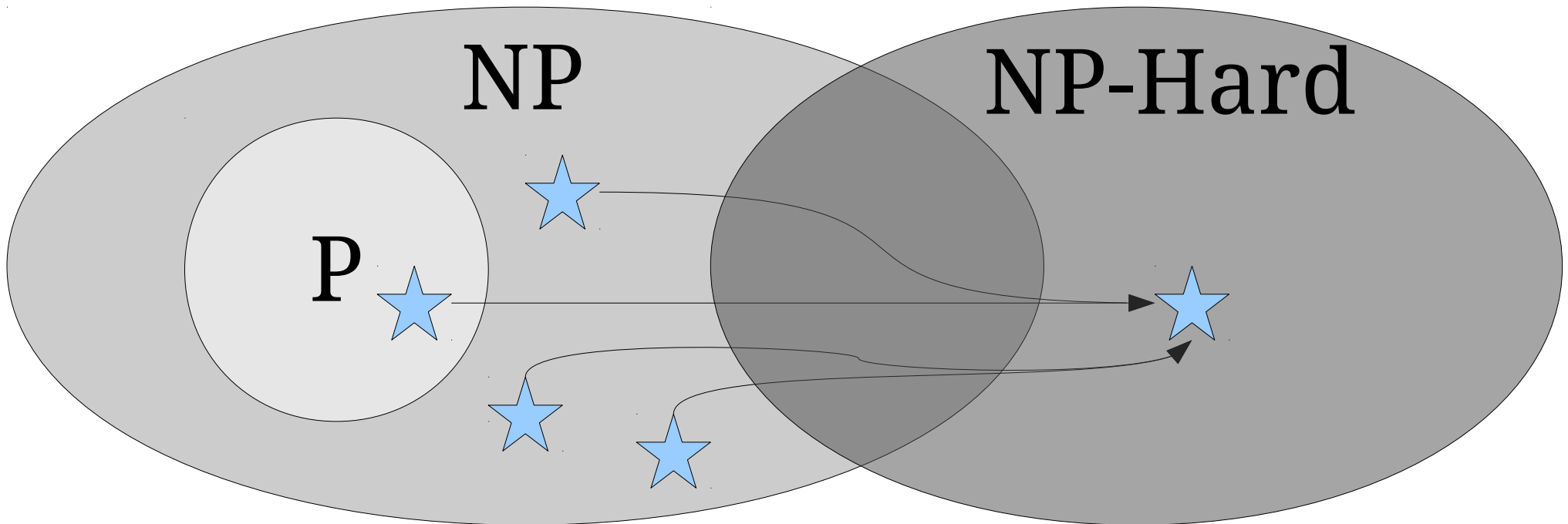
NP-Hardness

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



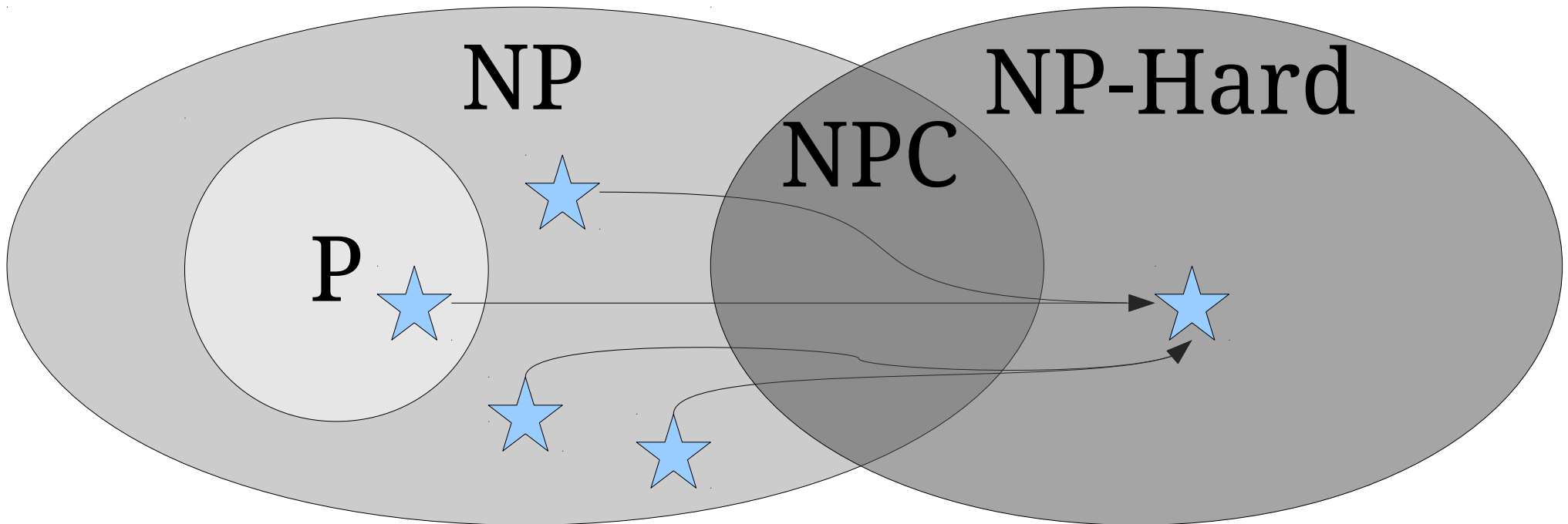
NP-Hardness

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

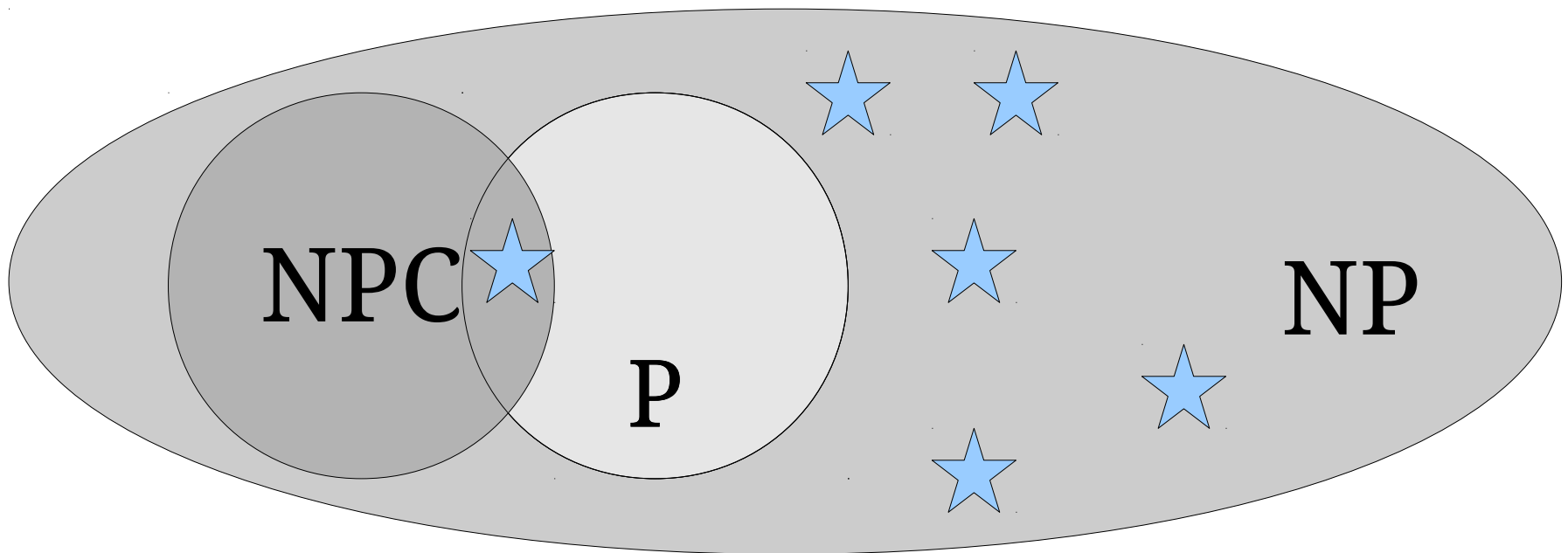


The Tantalizing Truth

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

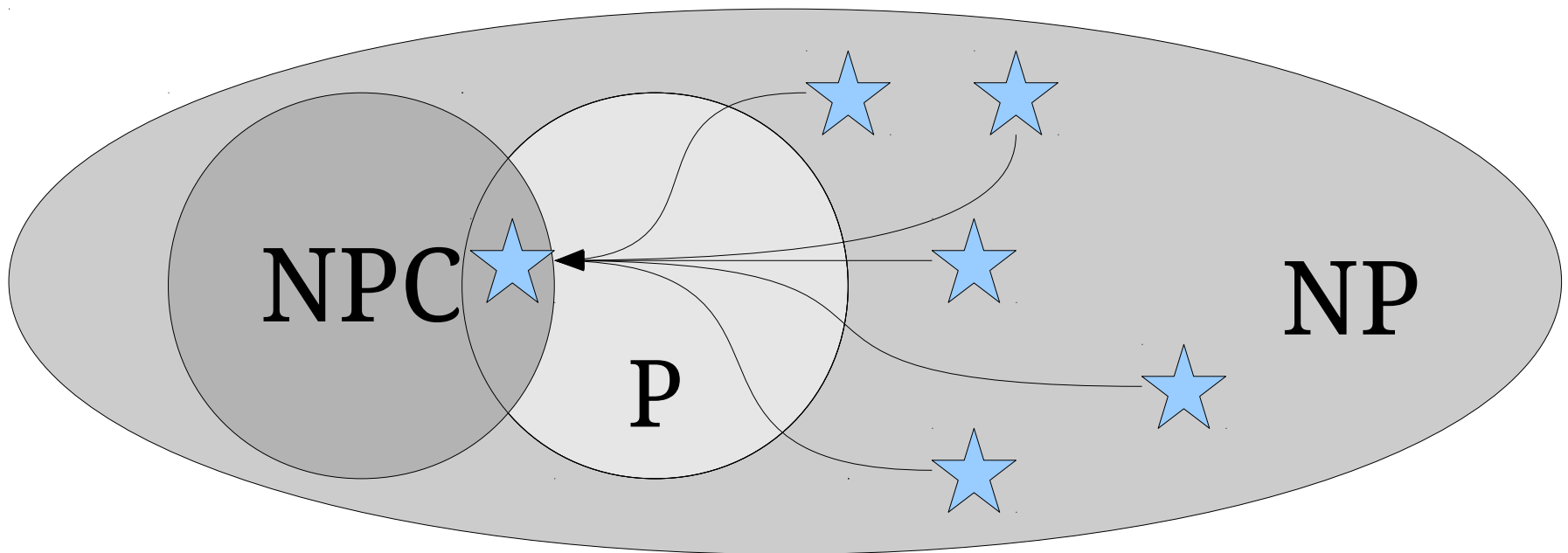
The Tantalizing Truth

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



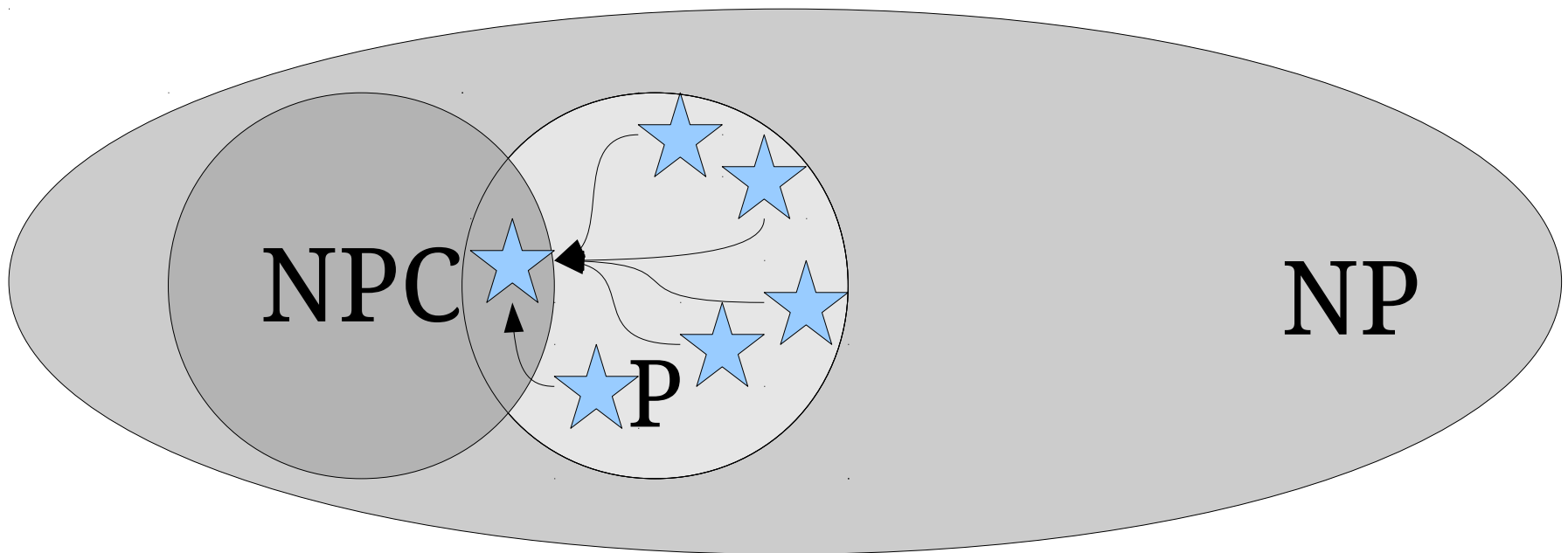
The Tantalizing Truth

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



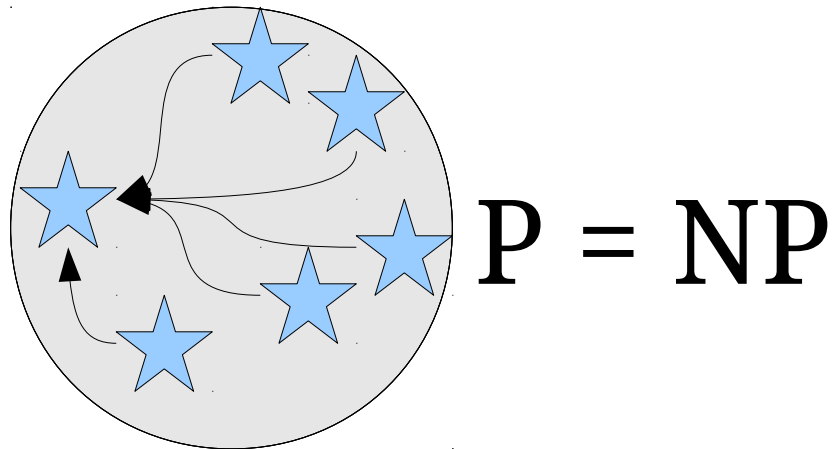
The Tantalizing Truth

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



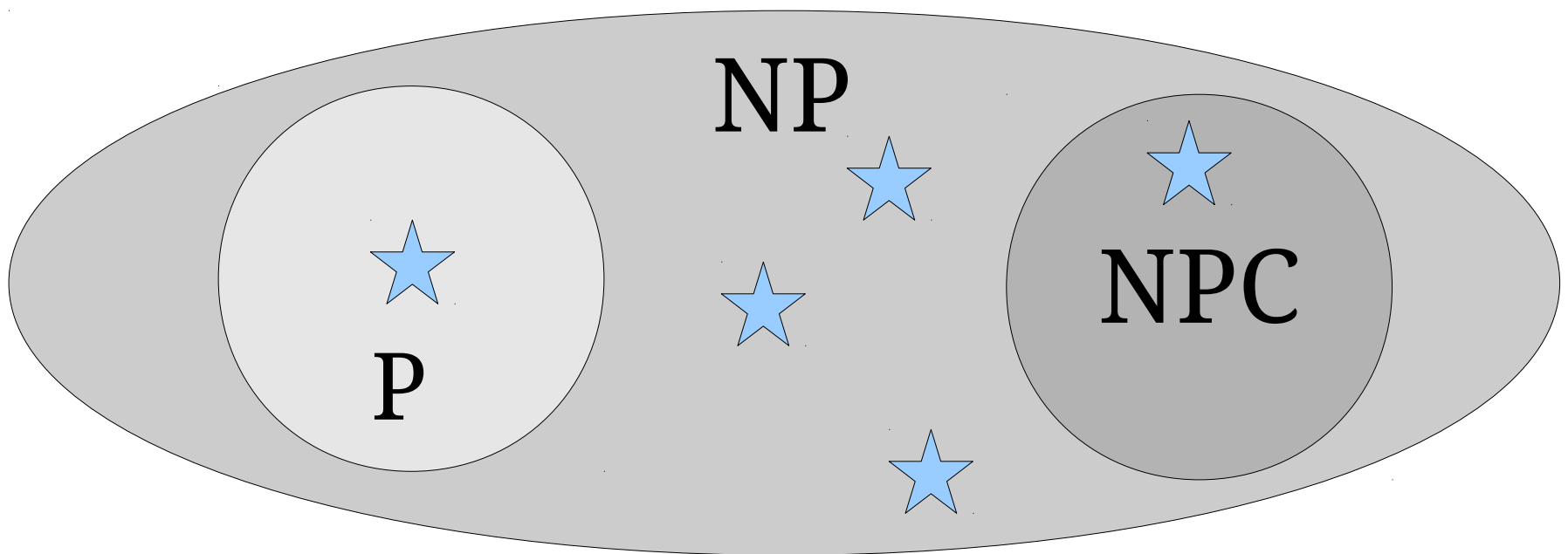
The Tantalizing Truth

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



The Tantalizing Truth

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



A Feel for **NP**-Completeness

- If a problem is **NP**-complete, then under the assumption that $\mathbf{P} \neq \mathbf{NP}$, there cannot be an efficient algorithm for it.
- In a sense, **NP**-complete problems are the hardest problems in **NP**.
- All known **NP**-complete problems are enormously hard to solve:
 - All known algorithms for **NP**-complete problems run in worst-case exponential time.
 - Most algorithms for **NP**-complete problems are infeasible for reasonably-sized inputs.

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.
 - $p \wedge q$ is satisfiable.
 - $p \wedge \neg p$ is unsatisfiable.
 - $p \rightarrow (q \wedge \neg q)$ 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:

$SAT = \{ \langle \varphi \rangle \mid \varphi \text{ is a satisfiable PL formula} \}$

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

Proof: Take CS154!

A Simpler **NP**-Complete Problem

Literals and Clauses

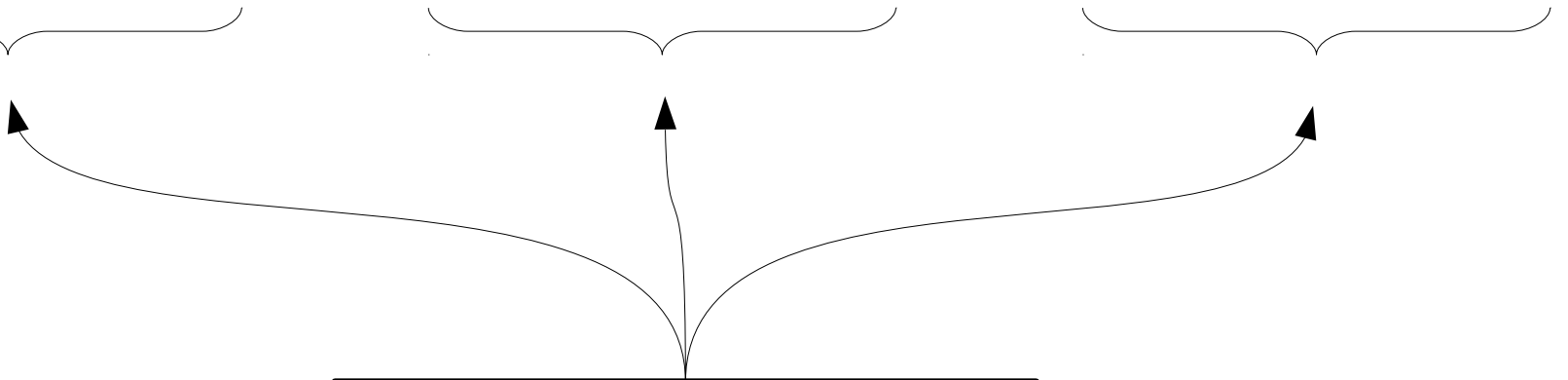
- A ***literal*** in propositional logic is a variable or its negation:
 - x
 - $\neg y$
 - But not $x \wedge y$.
- A ***clause*** is a many-way OR (*disjunction*) of literals.
 - $\neg x \vee y \vee \neg z$
 - x
 - But not $x \vee \neg(y \vee z)$

Conjunctive Normal Form

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

The Structure of CNF

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



Each clause must have
at least one
true literal in it.

The Structure of CNF

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

We should pick at least
one true literal from
each clause

The Structure of CNF

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

... subject to the constraint
that we 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:
 - $(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z)$
 - $(x \vee x \vee x) \wedge (y \vee \neg y \vee \neg x) \wedge (x \vee y \vee \neg y)$
 - But not $(x \vee y \vee z \vee w) \wedge (x \vee y)$
- The language **3SAT** is defined as follows:

$$\mathbf{3SAT = \{ \langle \varphi \rangle \mid \varphi \text{ is a satisfiable 3-CNF formula} \}}$$

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

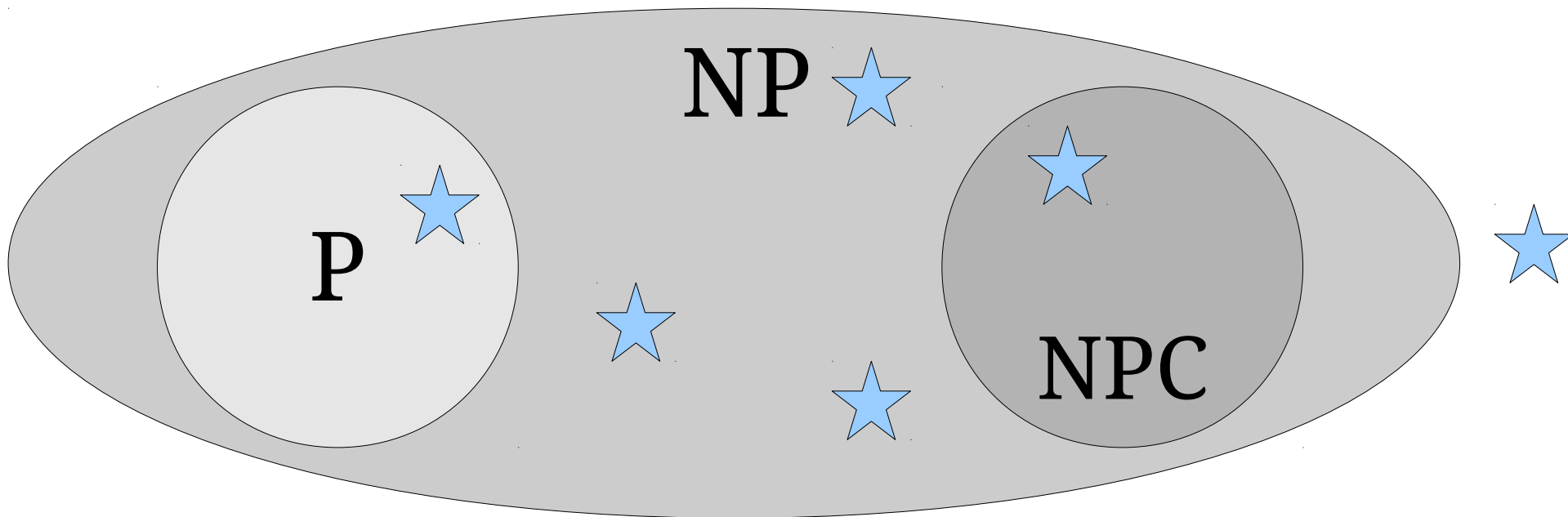
Finding Additional **NP**-Complete Problems

NP-Completeness

Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_p L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.

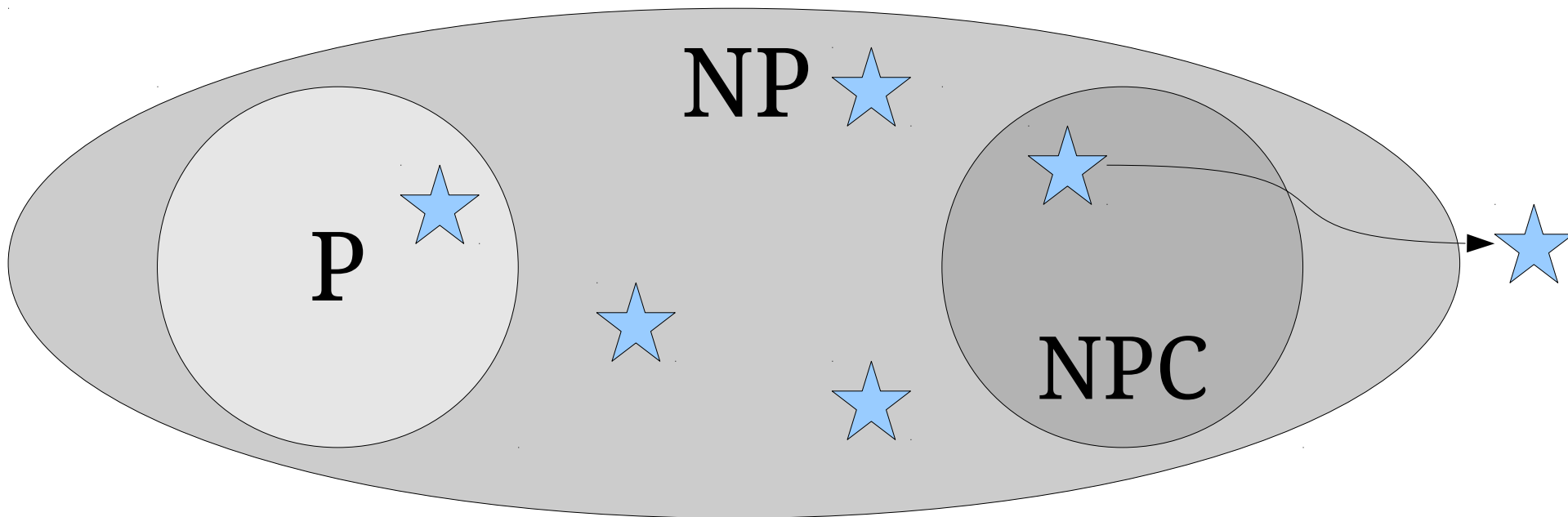
NP-Completeness

Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_p L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.



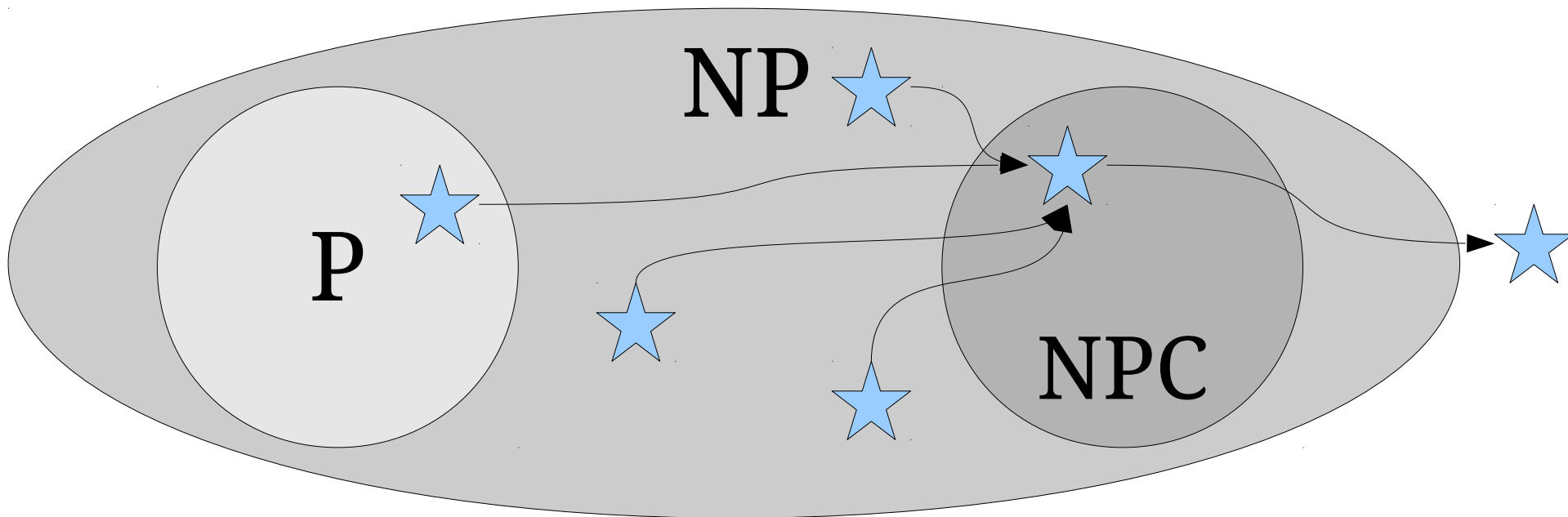
NP-Completeness

Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_p L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.



NP-Completeness

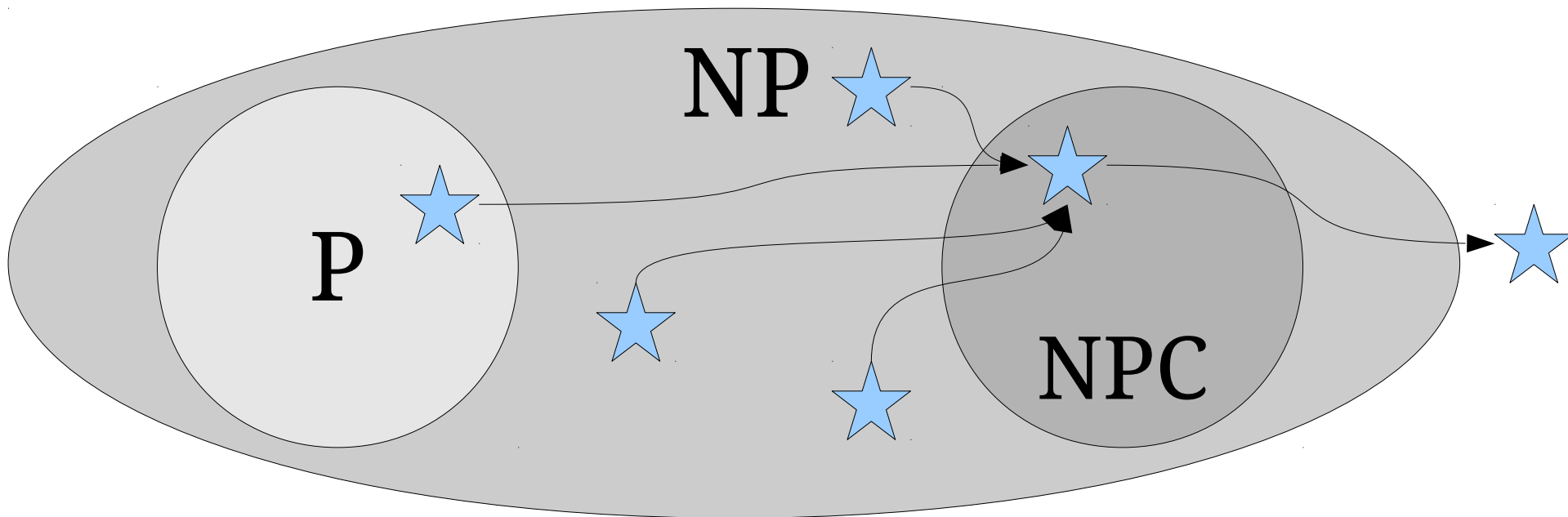
Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_p L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.



NP-Completeness

Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_p L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.

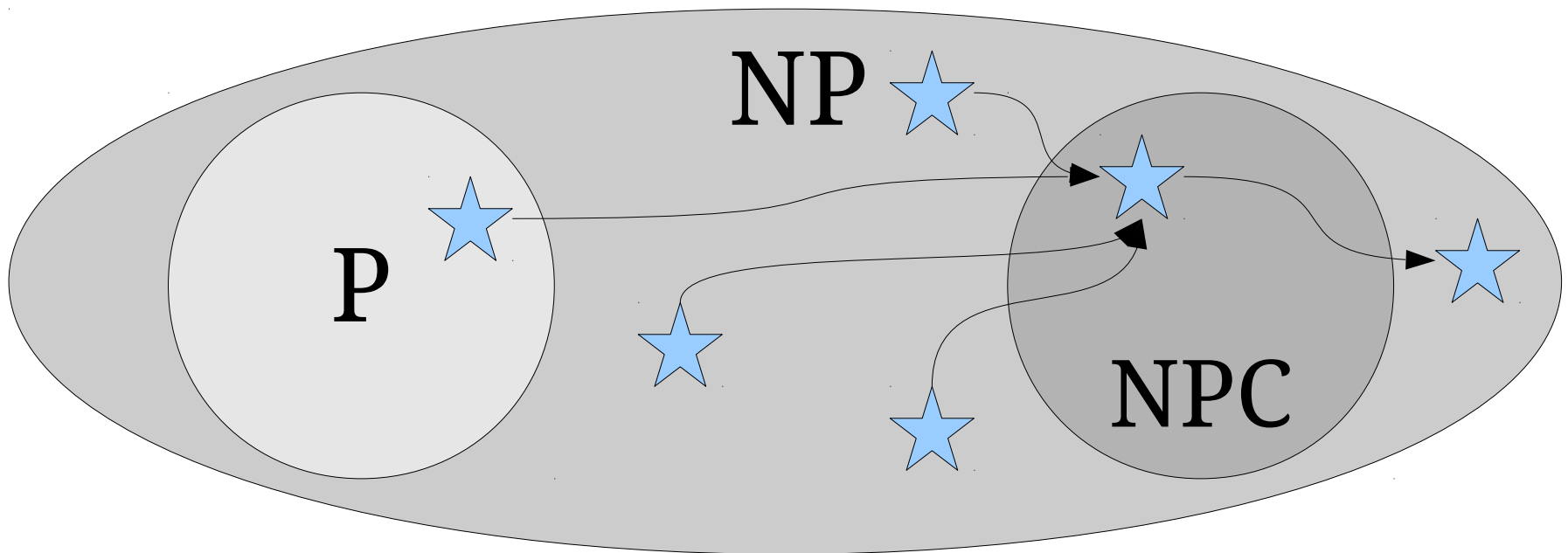
Theorem: Let L_1 and L_2 be languages where $L_1 \in \mathbf{NPC}$ and $L_2 \in \mathbf{NP}$. If $L_1 \leq_p L_2$, then $L_2 \in \mathbf{NPC}$.



NP-Completeness

Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_p L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.

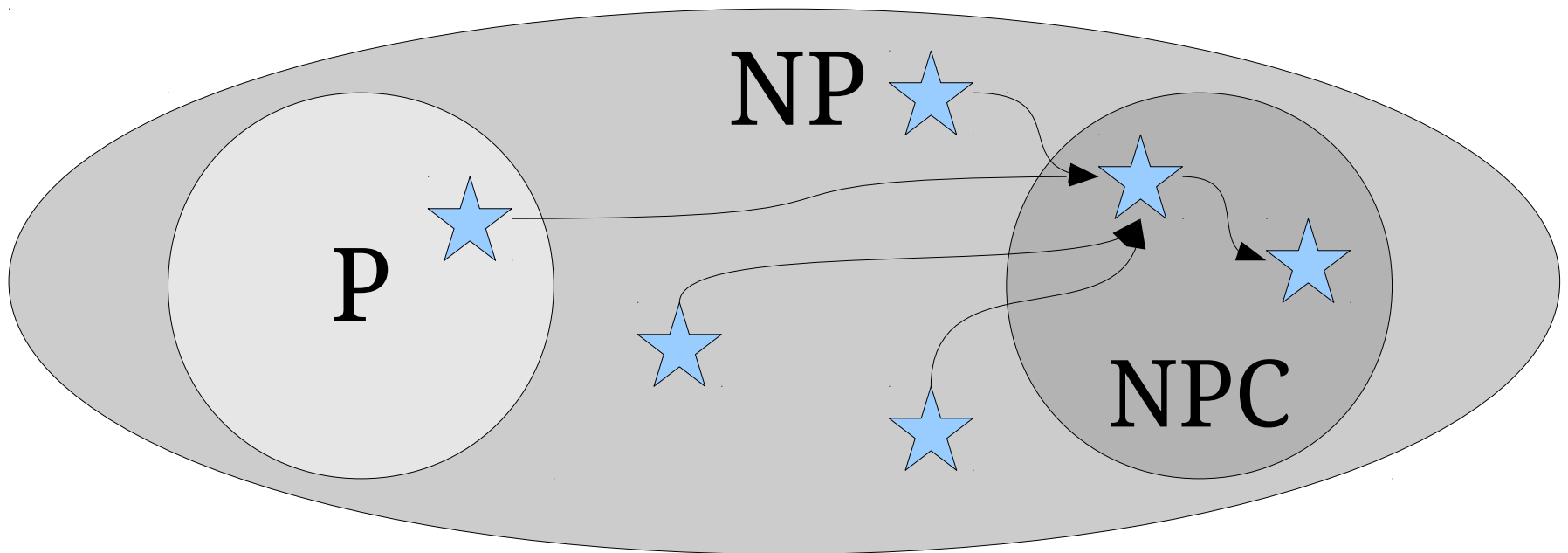
Theorem: Let L_1 and L_2 be languages where $L_1 \in \mathbf{NPC}$ and $L_2 \in \mathbf{NP}$. If $L_1 \leq_p L_2$, then $L_2 \in \mathbf{NPC}$.



NP-Completeness

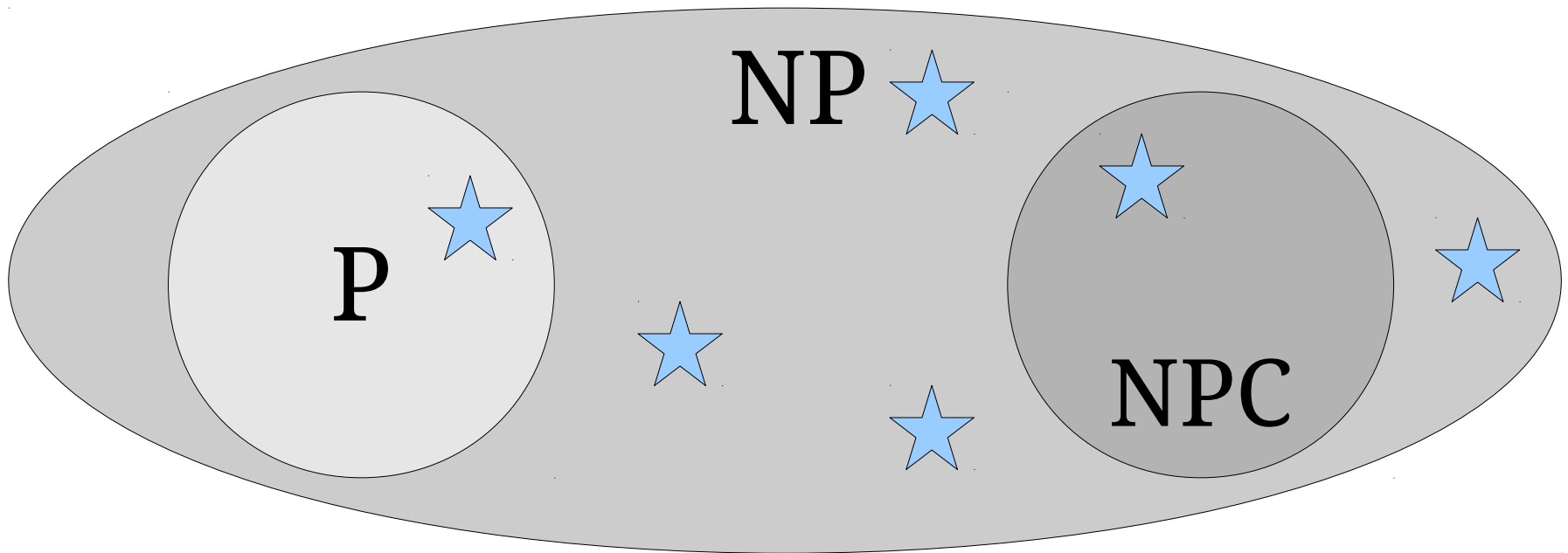
Theorem: Let L_1 and L_2 be languages. If $L_1 \leq_p L_2$ and L_1 is **NP**-hard, then L_2 is **NP**-hard.

Theorem: Let L_1 and L_2 be languages where $L_1 \in \mathbf{NPC}$ and $L_2 \in \mathbf{NP}$. If $L_1 \leq_p L_2$, then $L_2 \in \mathbf{NPC}$.



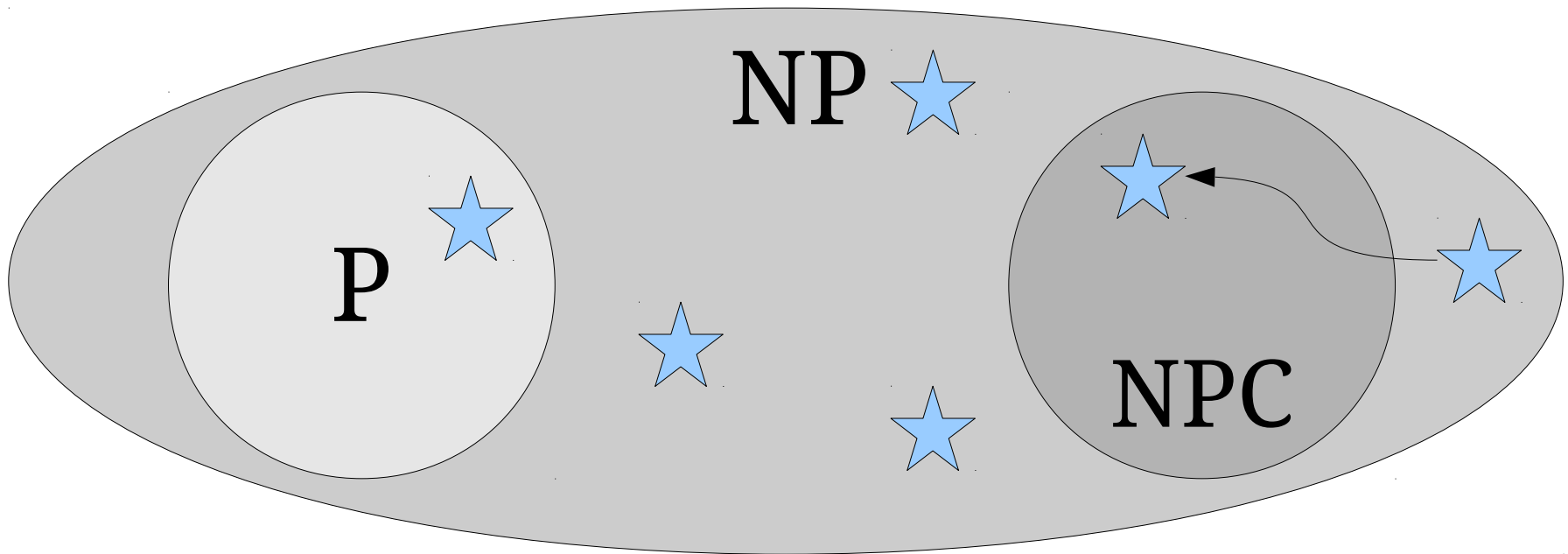
Be Careful!

- To prove that some language L is **NP**-complete, show that $L \in \mathbf{NP}$, 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** problems is reducible to any **NP**-complete problem!



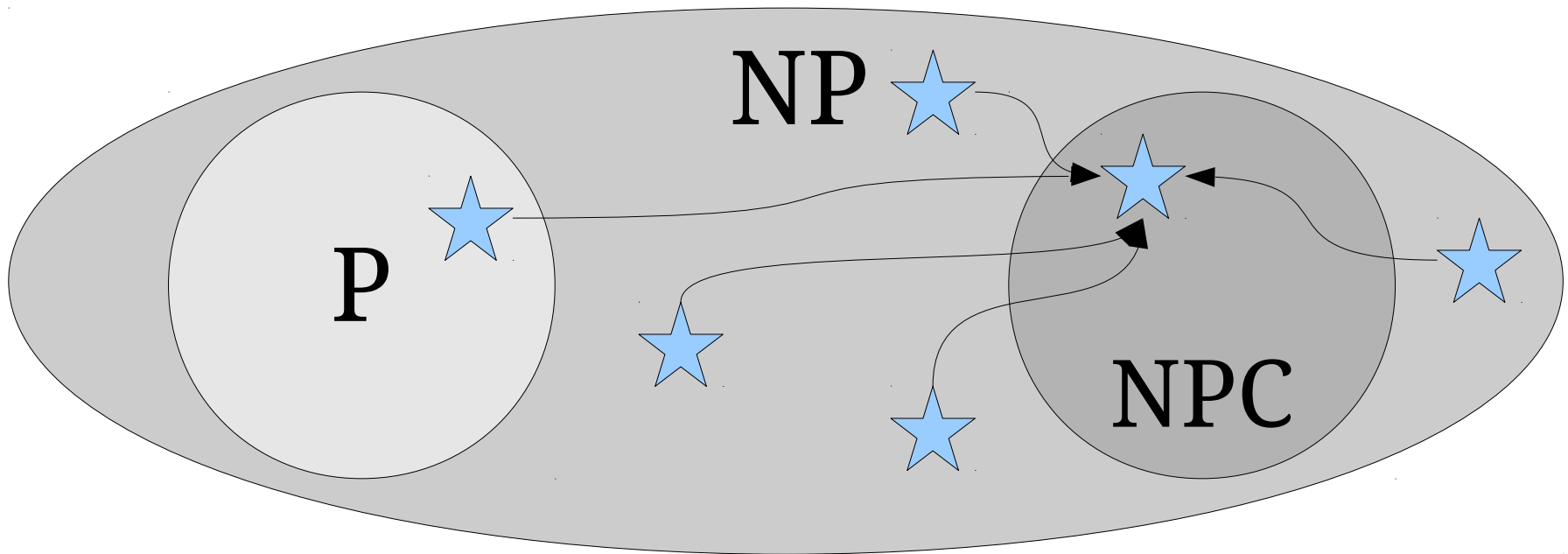
Be Careful!

- To prove that some language L is **NP**-complete, show that $L \in \mathbf{NP}$, 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** problems is reducible to any **NP**-complete problem!



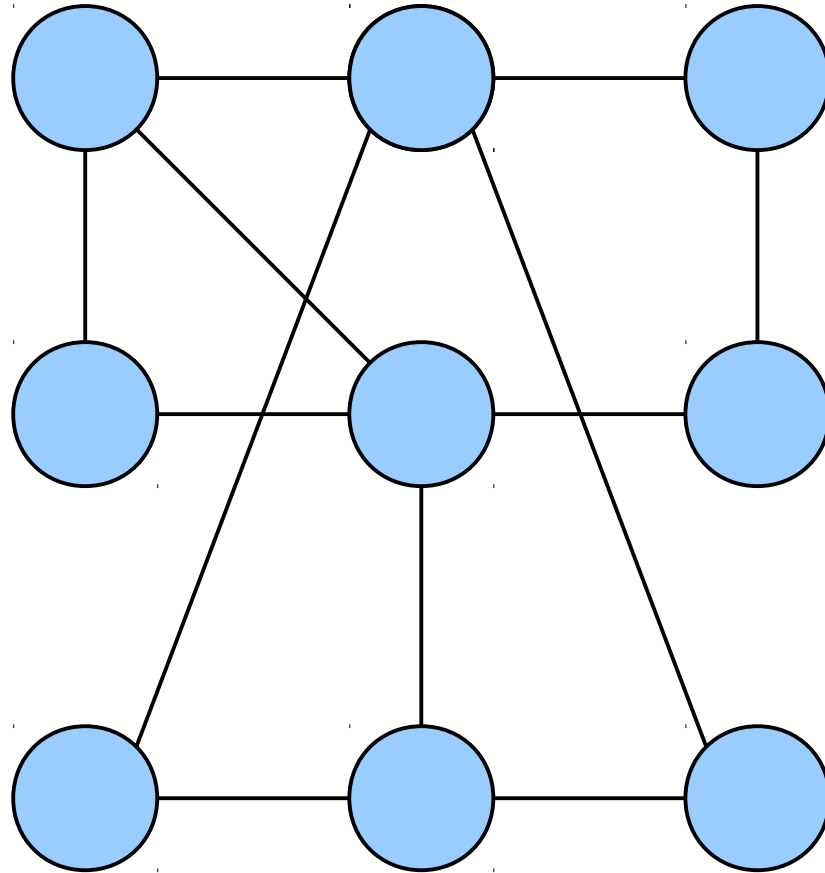
Be Careful!

- To prove that some language L is **NP**-complete, show that $L \in \mathbf{NP}$, 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** problems is reducible to any **NP**-complete problem!

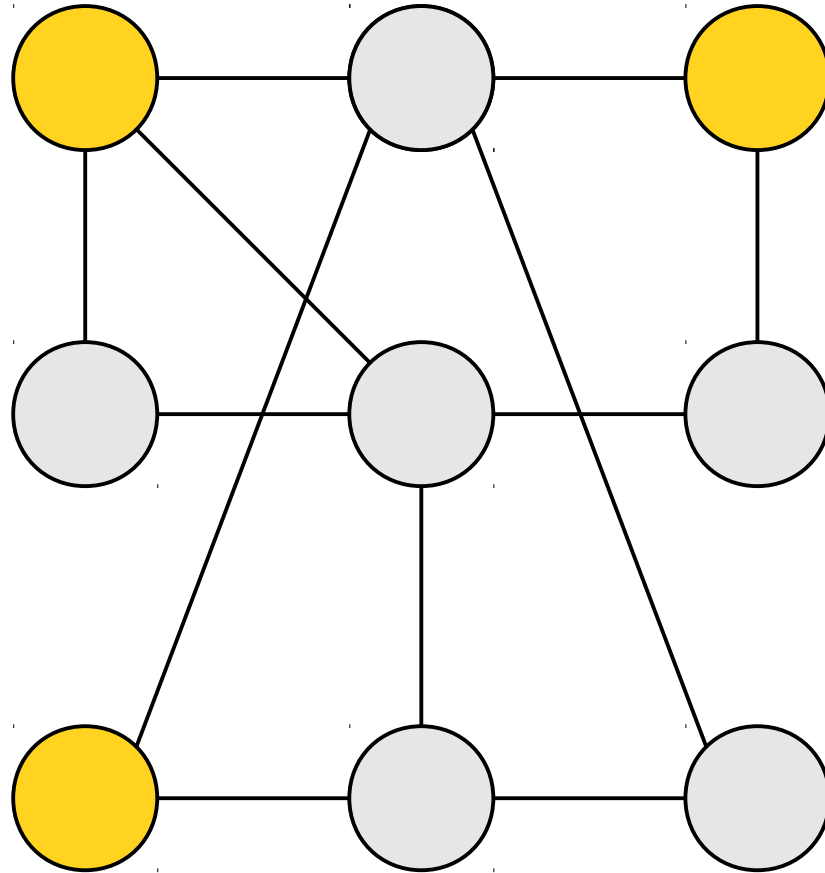


So what other problems are **NP**-complete?

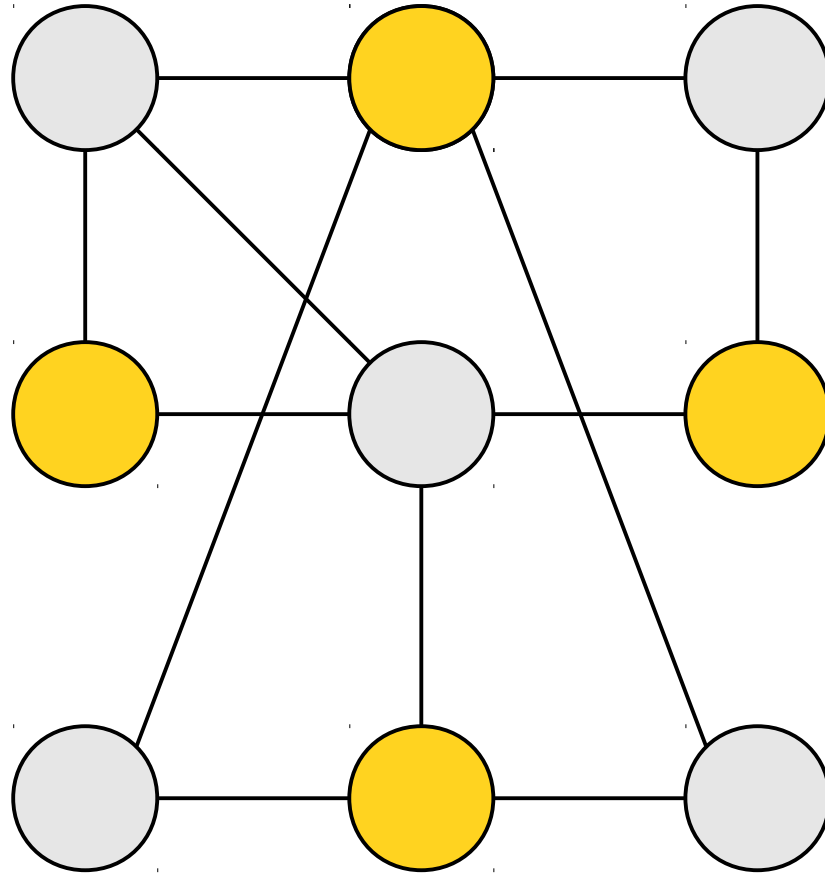
An ***independent set*** in an undirected graph is a set of nodes that have no edges between them.



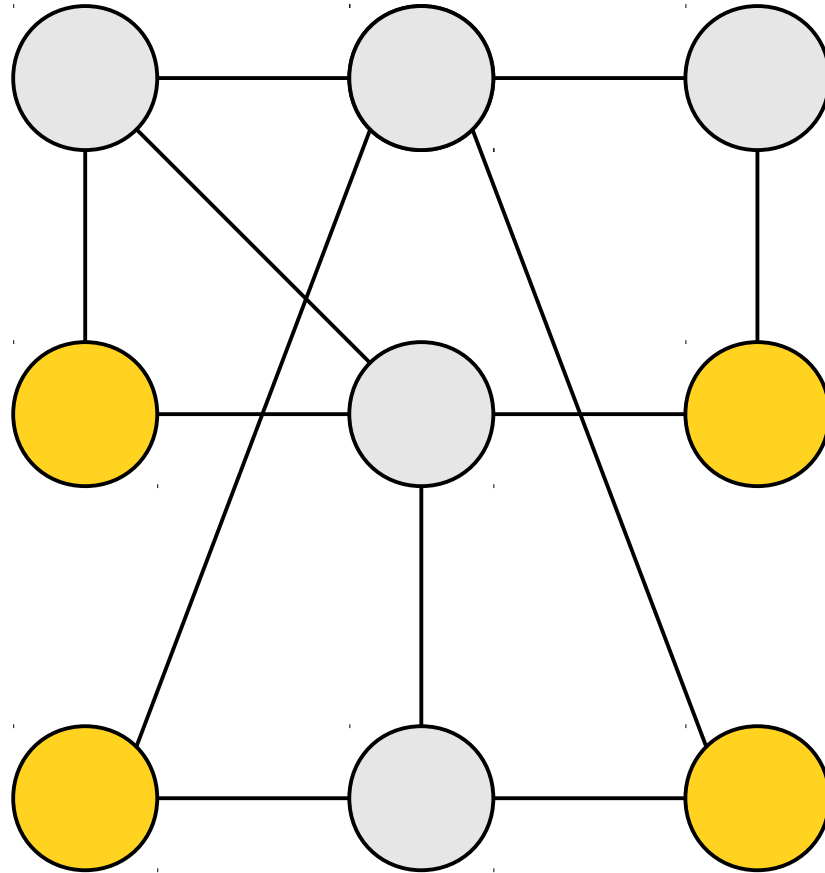
An ***independent set*** in an undirected graph is a set of nodes that have no edges between them.



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 = { $\langle G, n \rangle$ | G is an undirected graph with an independent set of size at least n }

INDSET \in **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 $\langle G, n, S \rangle$, where G is a graph, $n \in \mathbb{N}$, 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.”

$INDSET \in \mathbf{NPC}$

- The $INDSET$ problem is \mathbf{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

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

The Structure of 3CNF

$$\underbrace{(x \vee y \vee \neg z)} \wedge \underbrace{(\neg x \vee \neg y \vee z)} \wedge \underbrace{(\neg x \vee y \vee \neg z)}$$

The Structure of 3CNF

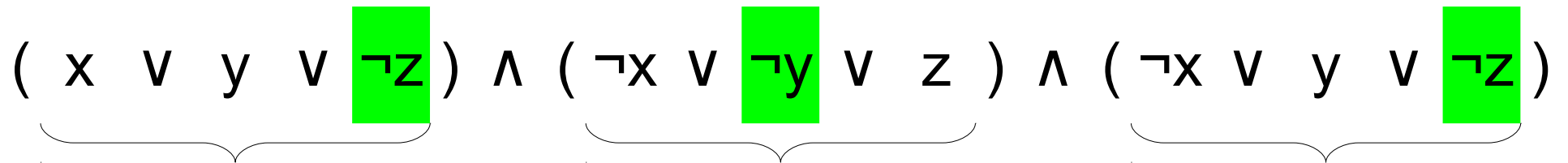
$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

Each clause must have
at least one
true literal in it.

The Structure of 3CNF

$$\underbrace{(x \vee y \vee \neg z)} \wedge \underbrace{(\neg x \vee \neg y \vee z)} \wedge \underbrace{(\neg x \vee y \vee \neg z)}$$

The Structure of 3CNF

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$


We should pick at least one true literal from each clause

The Structure of 3CNF

$$\underbrace{(x \vee y \vee \neg z)} \wedge \underbrace{(\neg x \vee \neg y \vee z)} \wedge \underbrace{(\neg x \vee y \vee \neg z)}$$

The Structure of 3CNF

$$\underbrace{(x \vee y \vee \neg z)} \wedge \underbrace{(\neg x \vee \neg y \vee z)} \wedge \underbrace{(\neg x \vee y \vee \neg z)}$$

The Structure of 3CNF

$$\underbrace{(x \vee y \vee \neg z)} \wedge \underbrace{(\neg x \vee \neg y \vee z)} \wedge \underbrace{(\neg x \vee y \vee \neg z)}$$

The Structure of 3CNF

$$\underbrace{(x \vee y \vee \neg z)} \wedge \underbrace{(\neg x \vee \neg y \vee z)} \wedge \underbrace{(\neg x \vee y \vee \neg z)}$$

The Structure of 3CNF

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

... 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

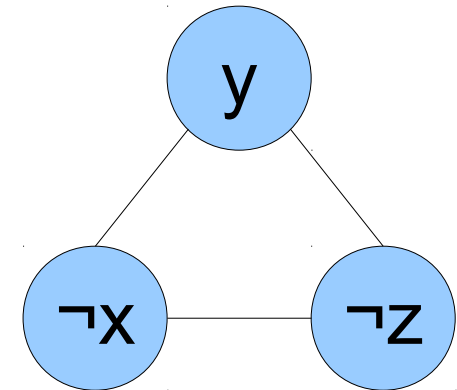
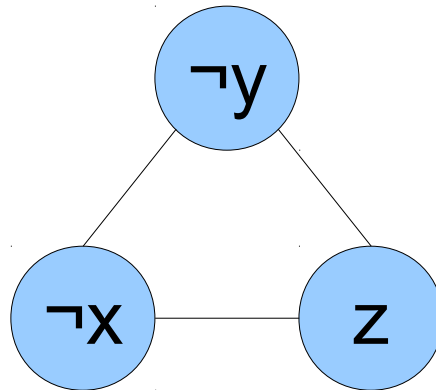
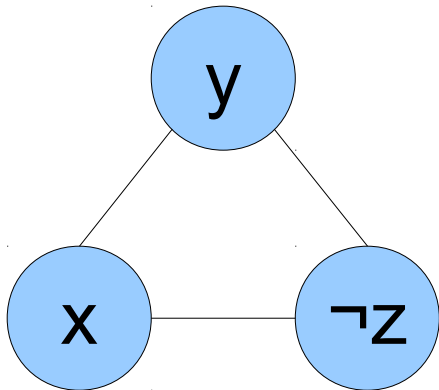
$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

From 3SAT to INDSET

$$\underbrace{(x \vee y \vee \neg z)} \wedge \underbrace{(\neg x \vee \neg y \vee z)} \wedge \underbrace{(\neg x \vee y \vee \neg z)}$$

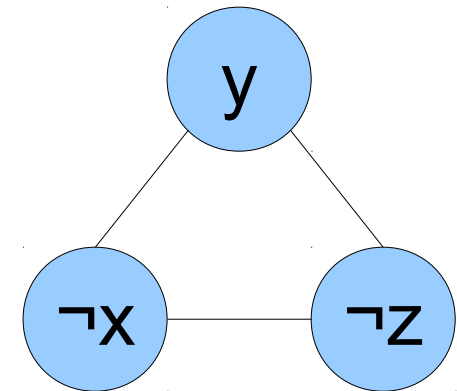
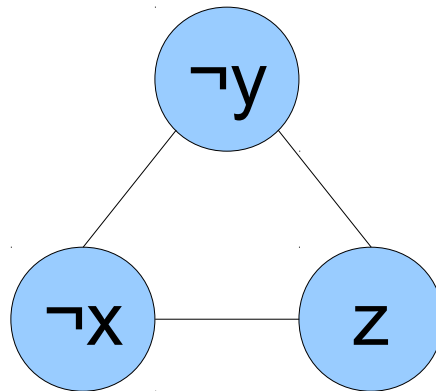
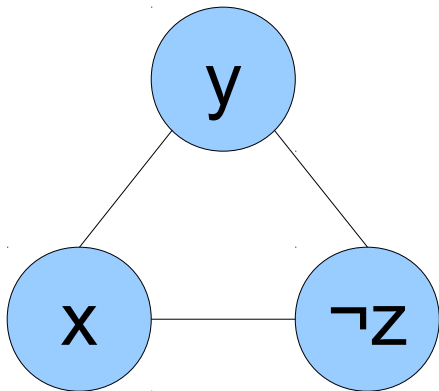
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



From 3SAT to INDSET

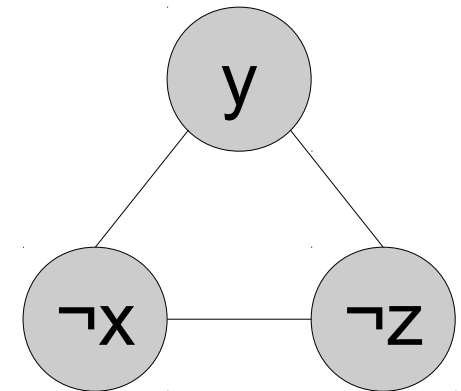
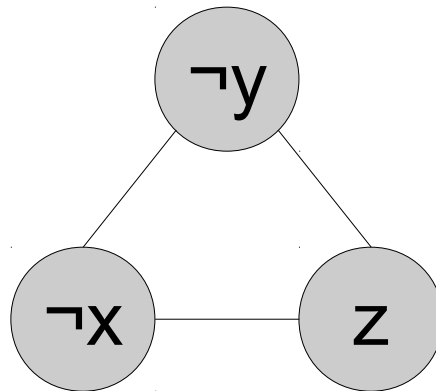
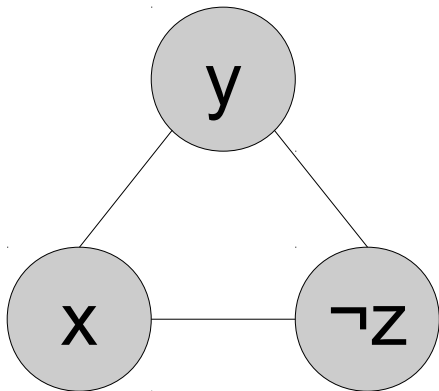
$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



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

From 3SAT to INDSET

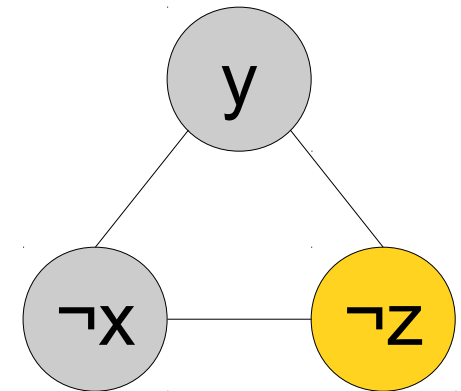
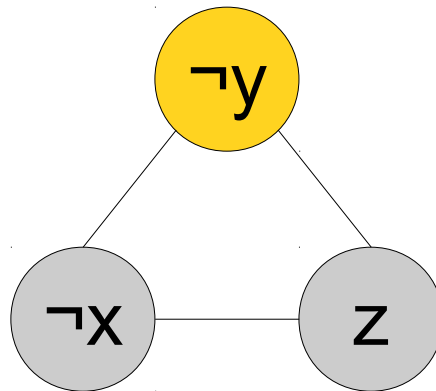
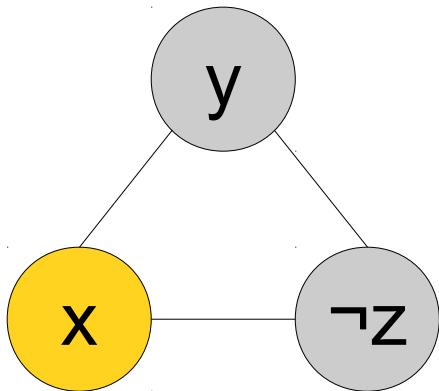
$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



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

From 3SAT to INDSET

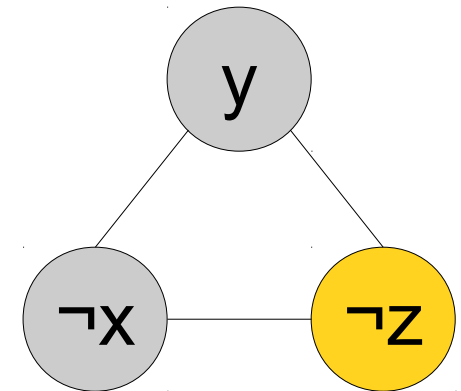
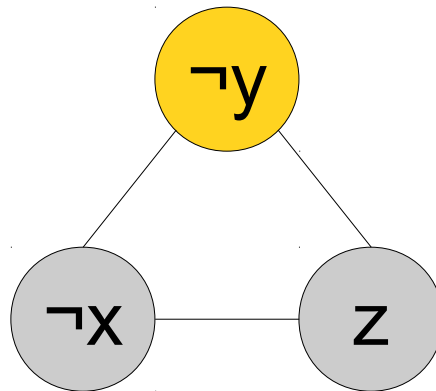
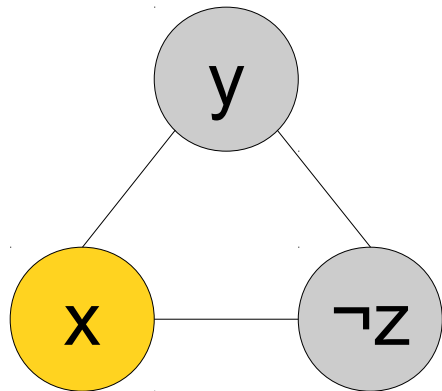
$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



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

From 3SAT to INDSET

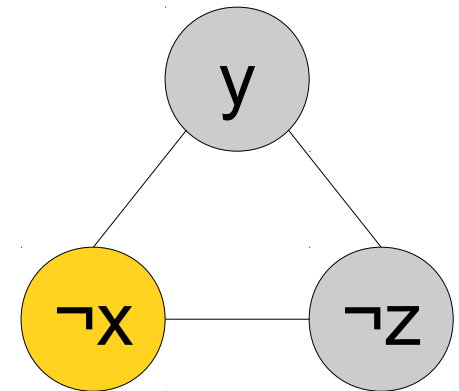
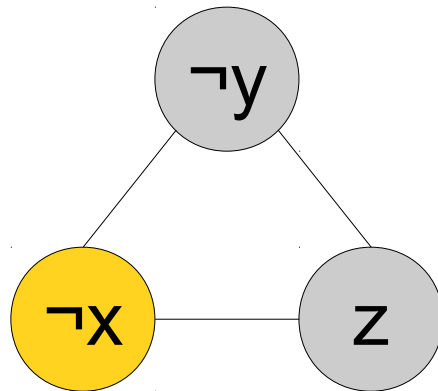
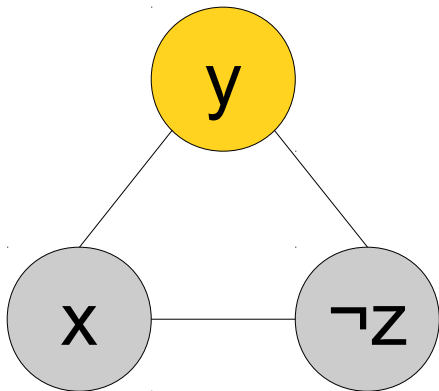
$$(\boxed{x} \vee y \vee \neg z) \wedge (\neg x \vee \boxed{\neg y} \vee z) \wedge (\neg x \vee y \vee \boxed{\neg z})$$



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

From 3SAT to INDSET

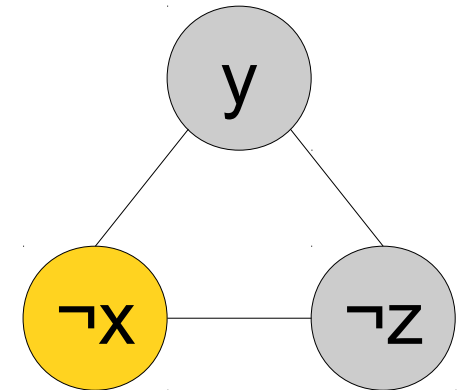
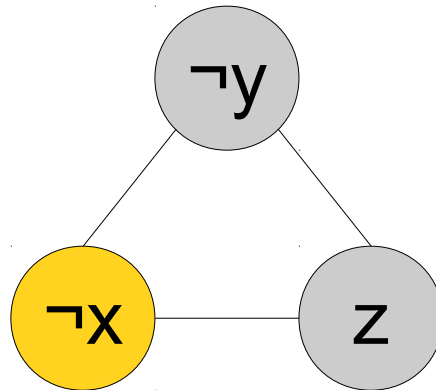
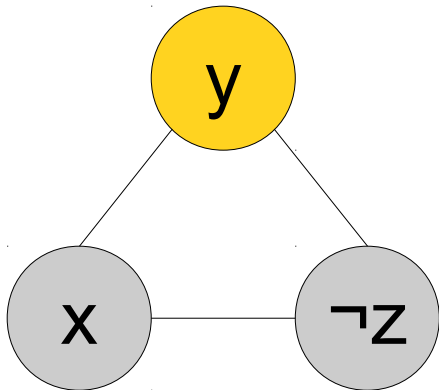
$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



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

From 3SAT to INDSET

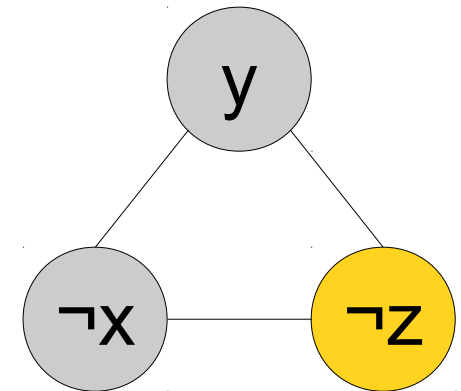
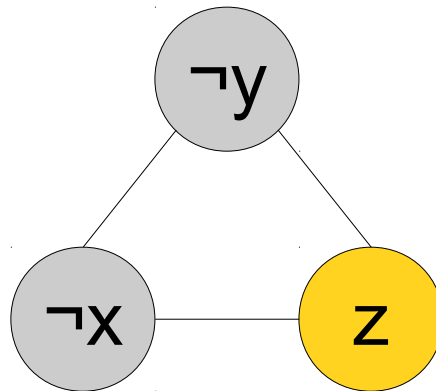
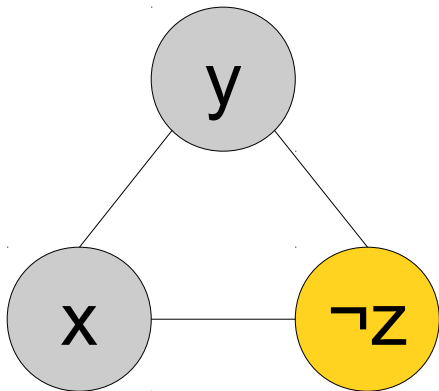
$$(x \vee \boxed{y} \vee \neg z) \wedge (\boxed{\neg x} \vee \neg y \vee z) \wedge (\boxed{\neg x} \vee y \vee \neg z)$$



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

From 3SAT to INDSET

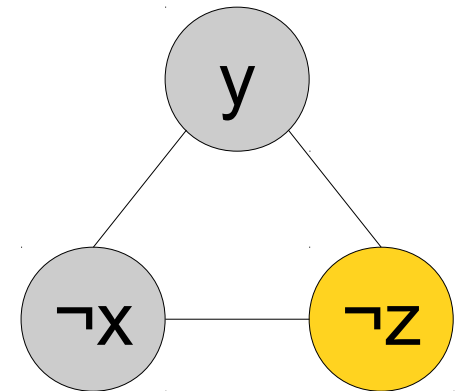
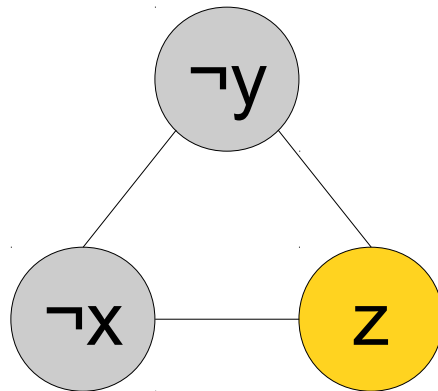
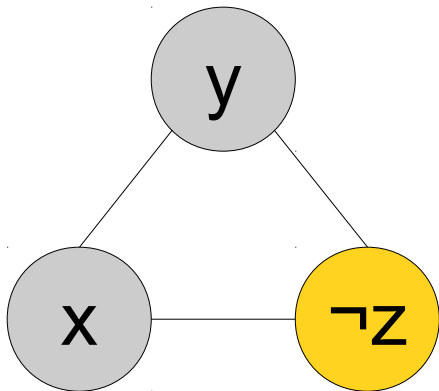
$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



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

From 3SAT to INDSET

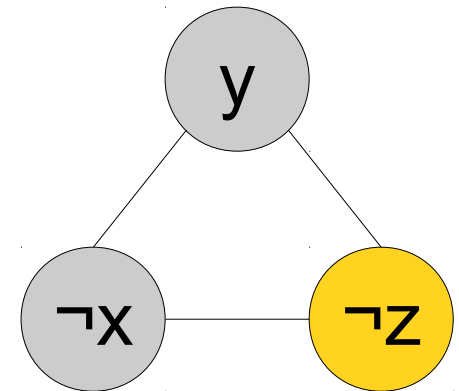
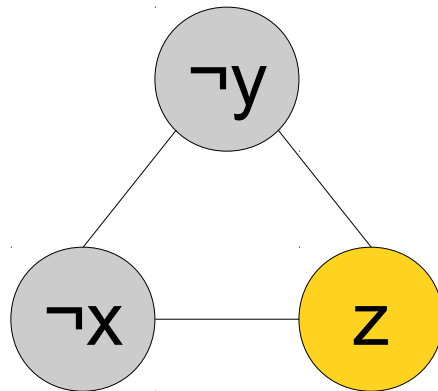
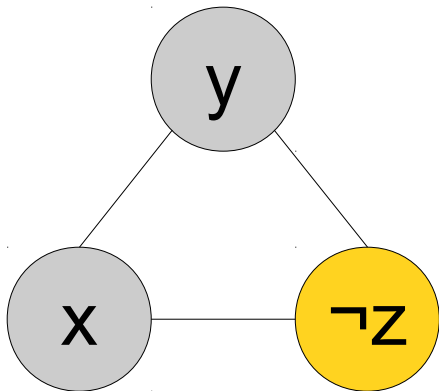
$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



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

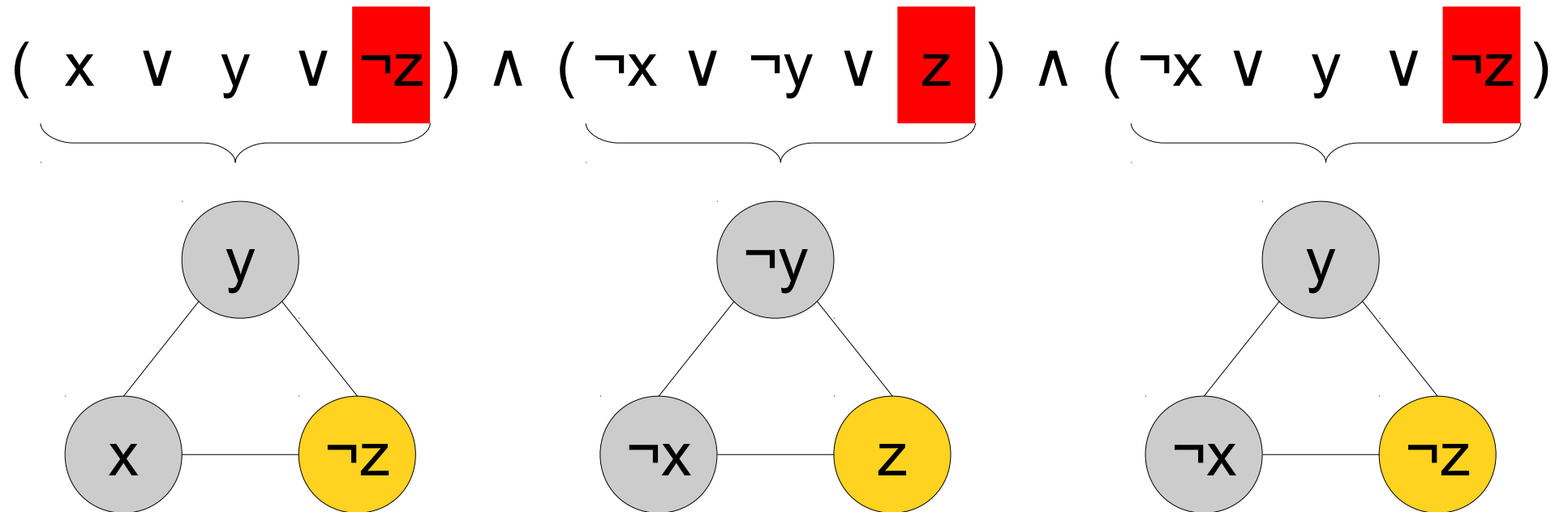
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



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

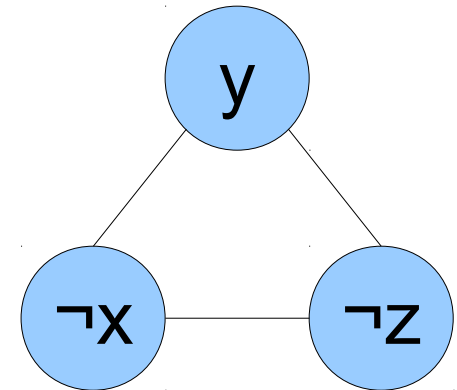
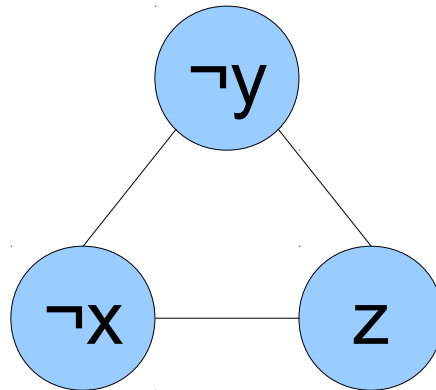
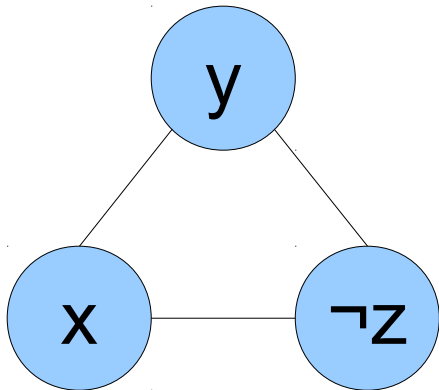
From 3SAT to INDSET



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

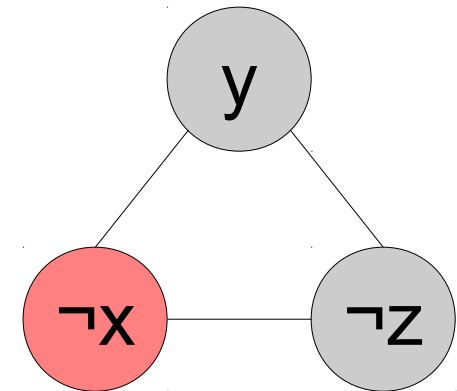
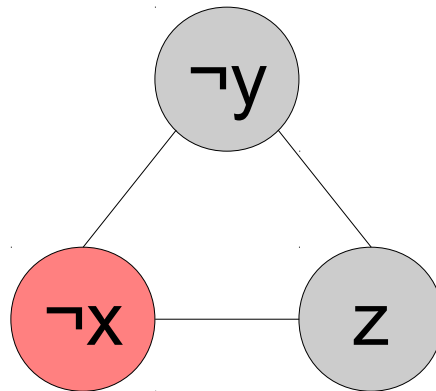
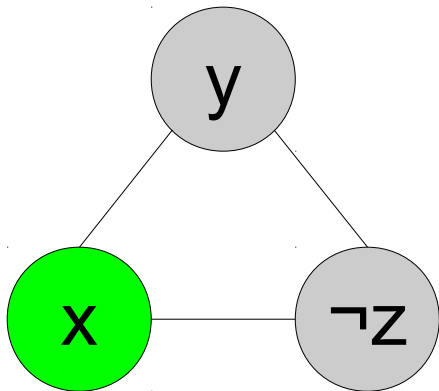
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



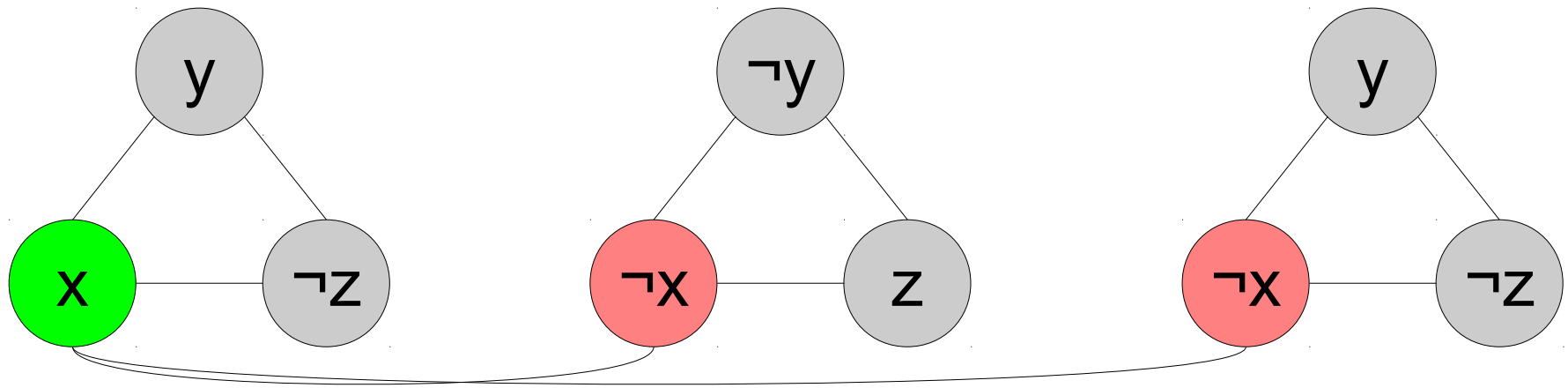
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

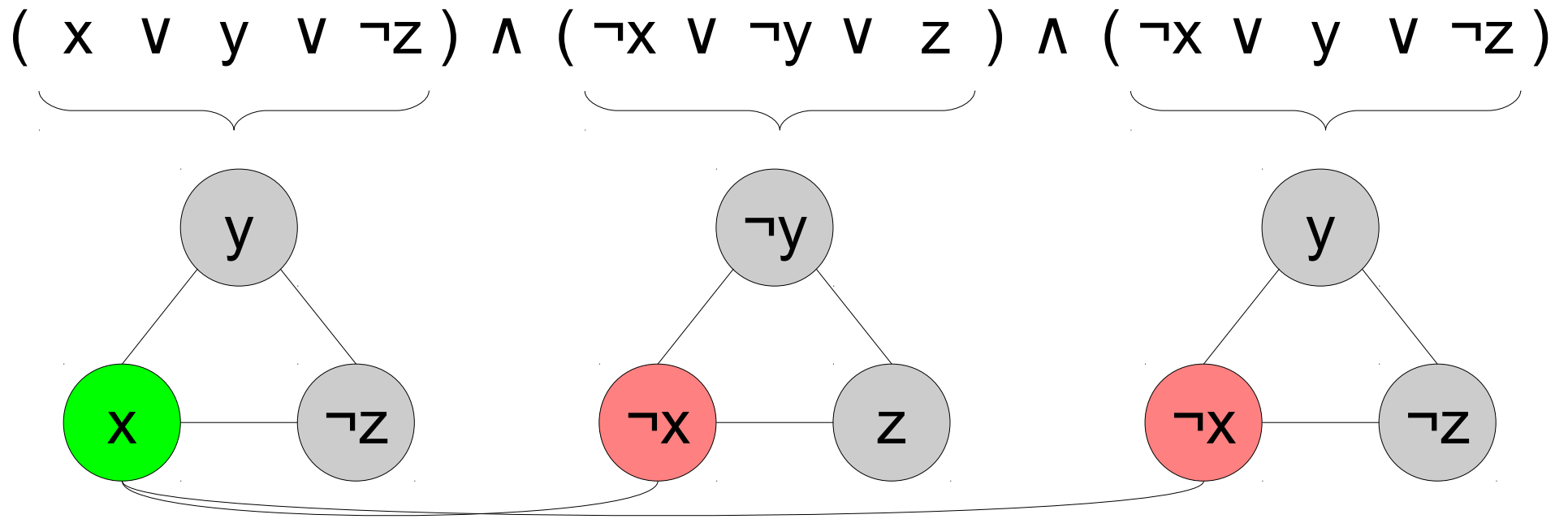


From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



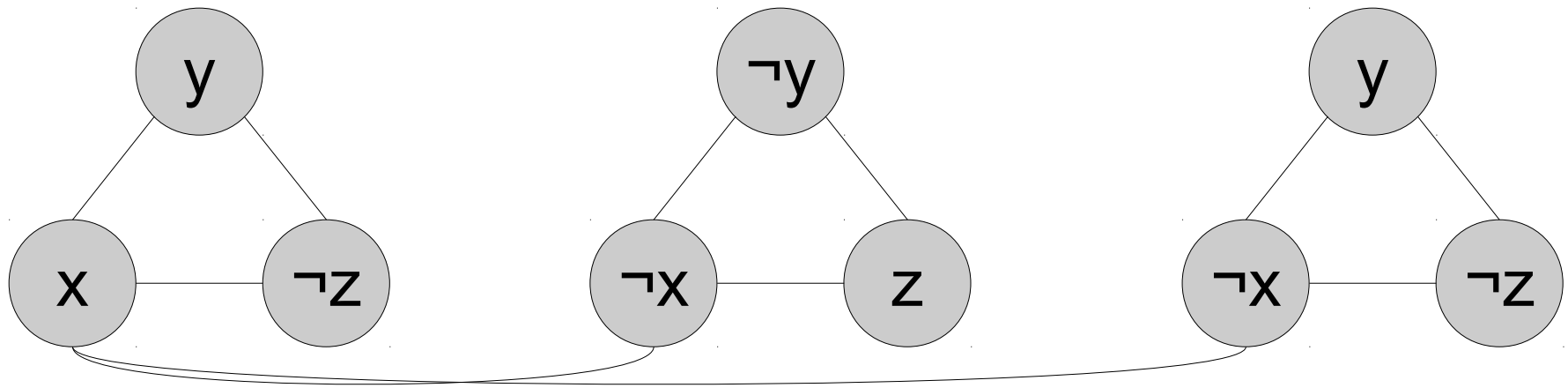
From 3SAT to INDSET



No independent set in this graph can choose two nodes labeled x and $\neg x$.

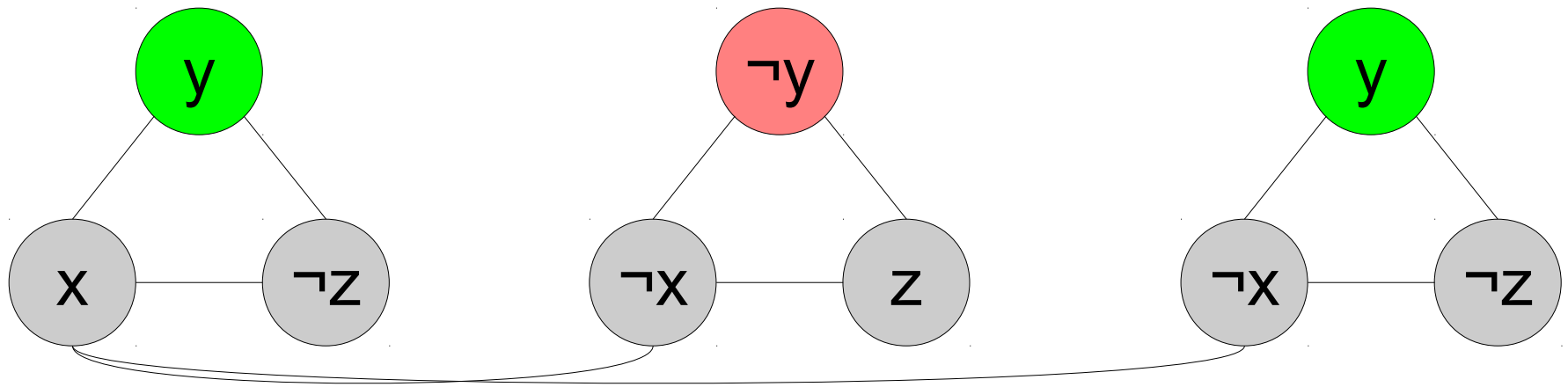
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



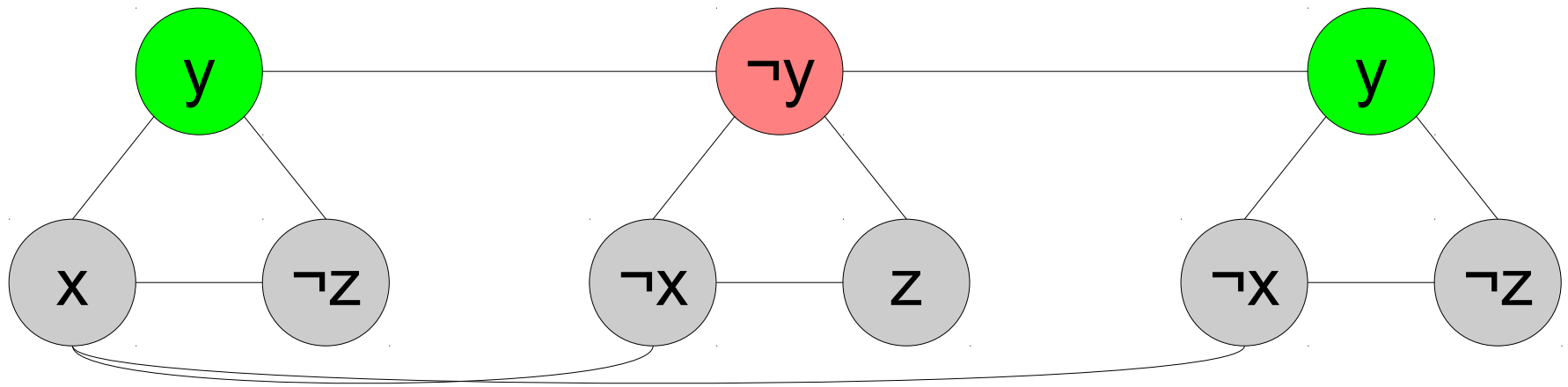
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



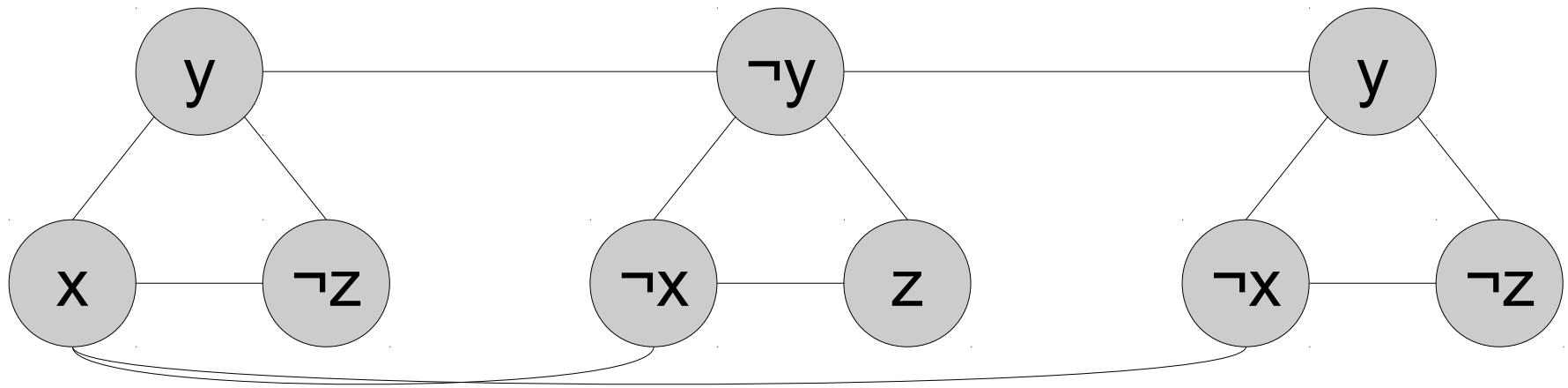
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



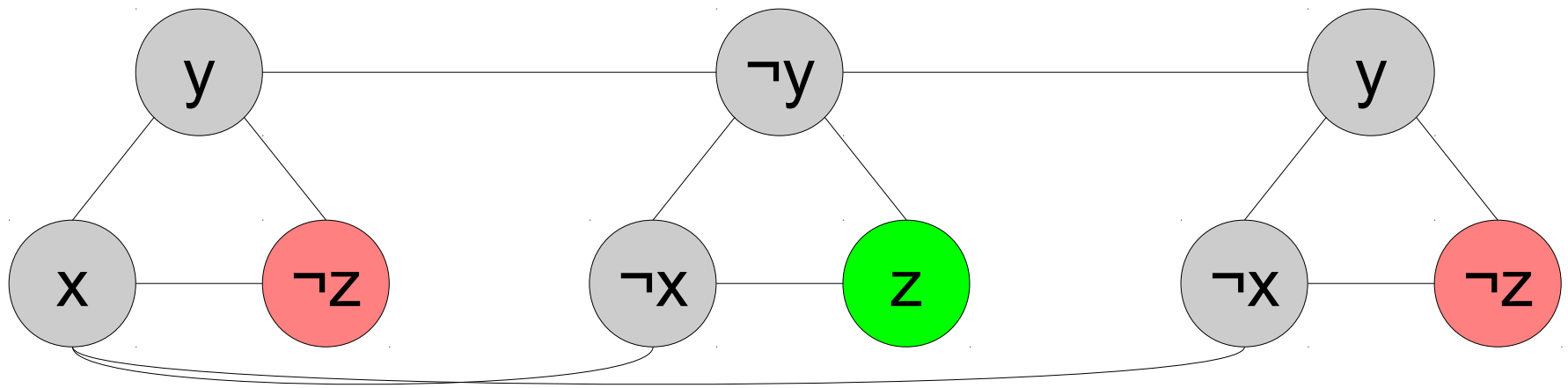
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



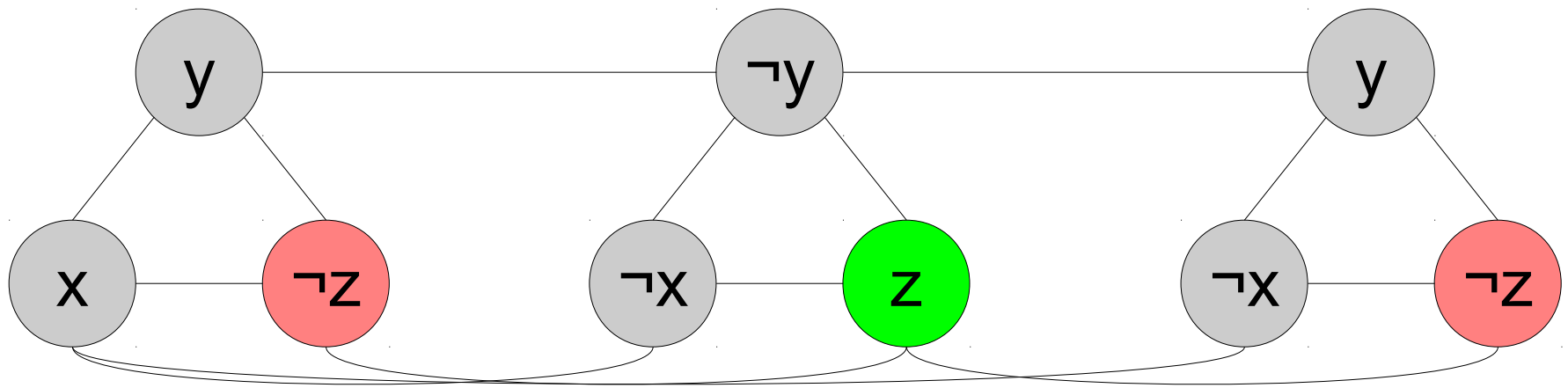
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



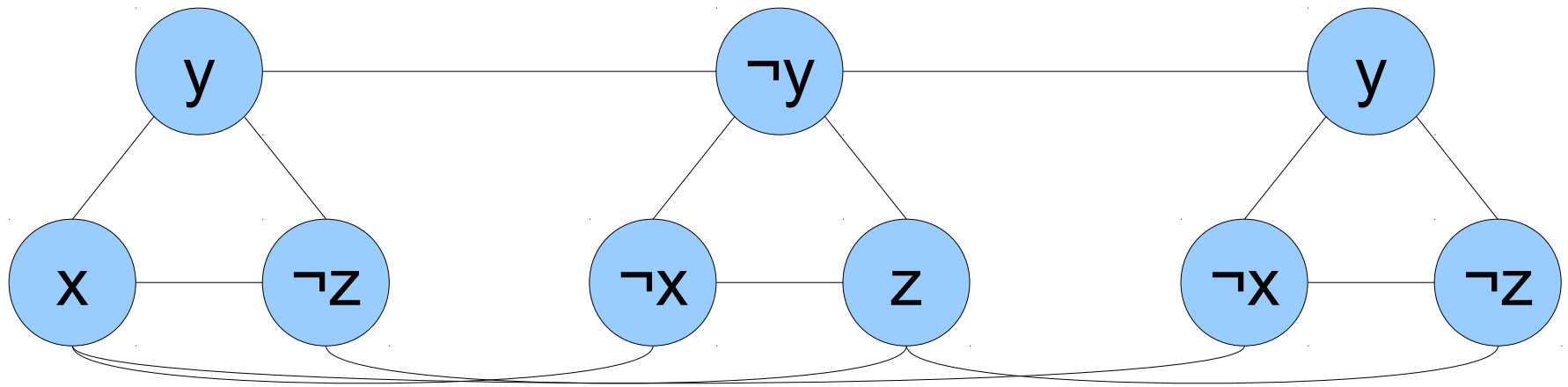
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



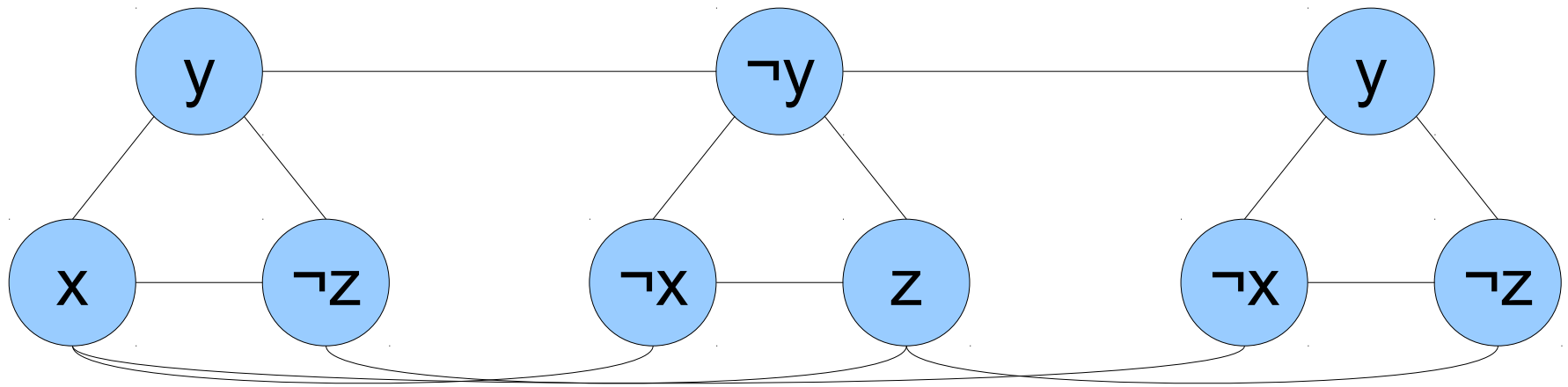
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



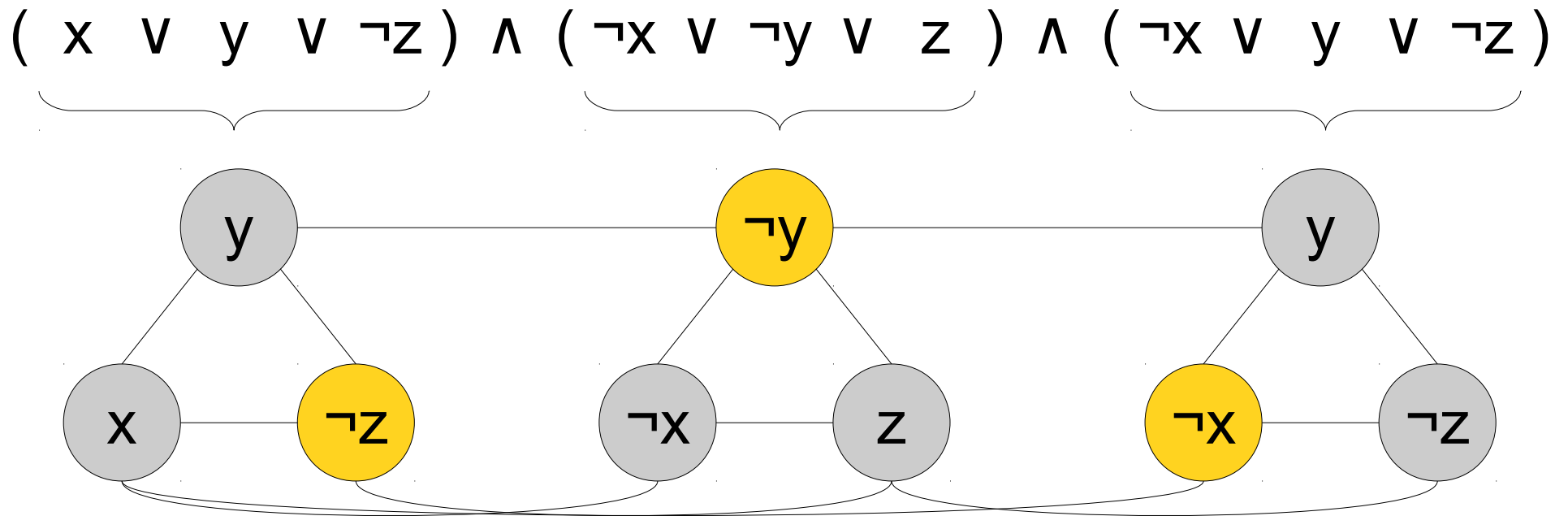
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



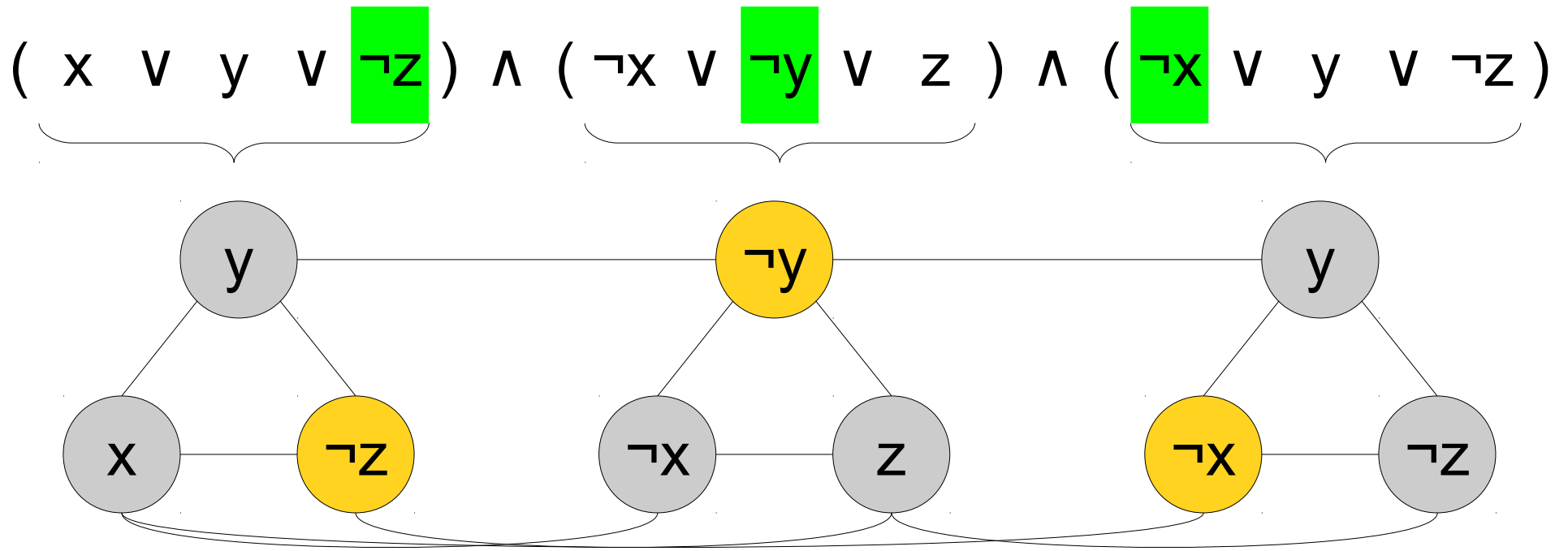
If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET



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

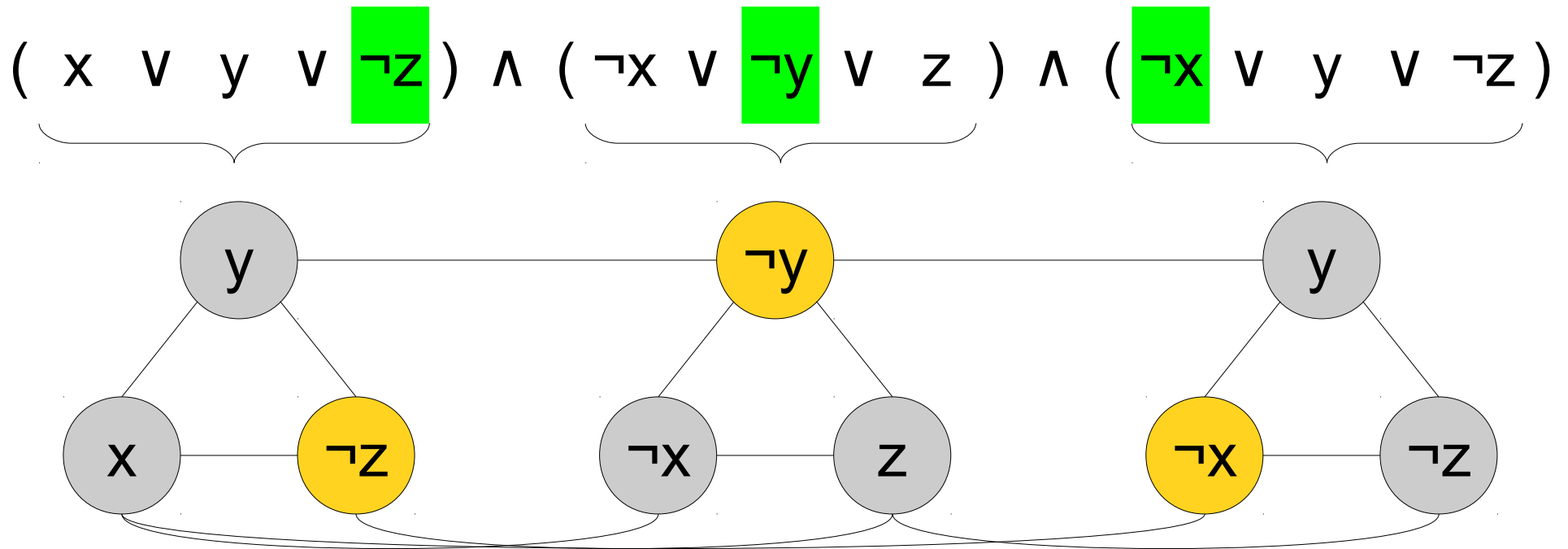
From 3SAT to INDSET



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

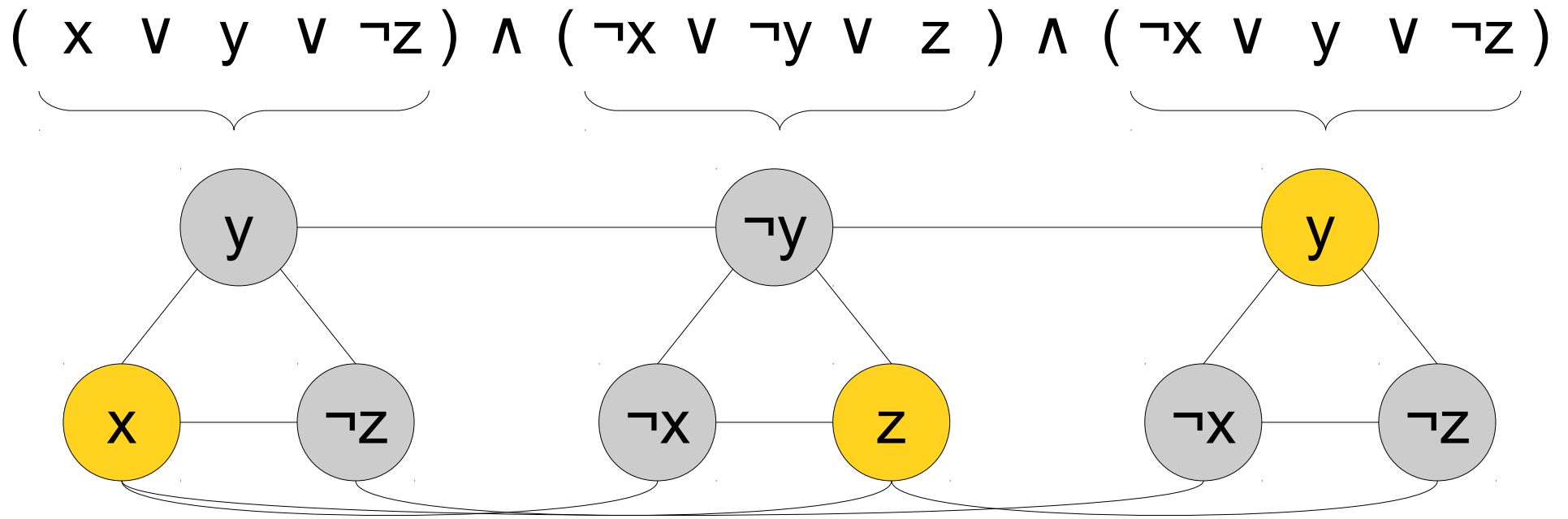
From 3SAT to INDSET

$x = \text{false}, y = \text{false}, z = \text{false}.$



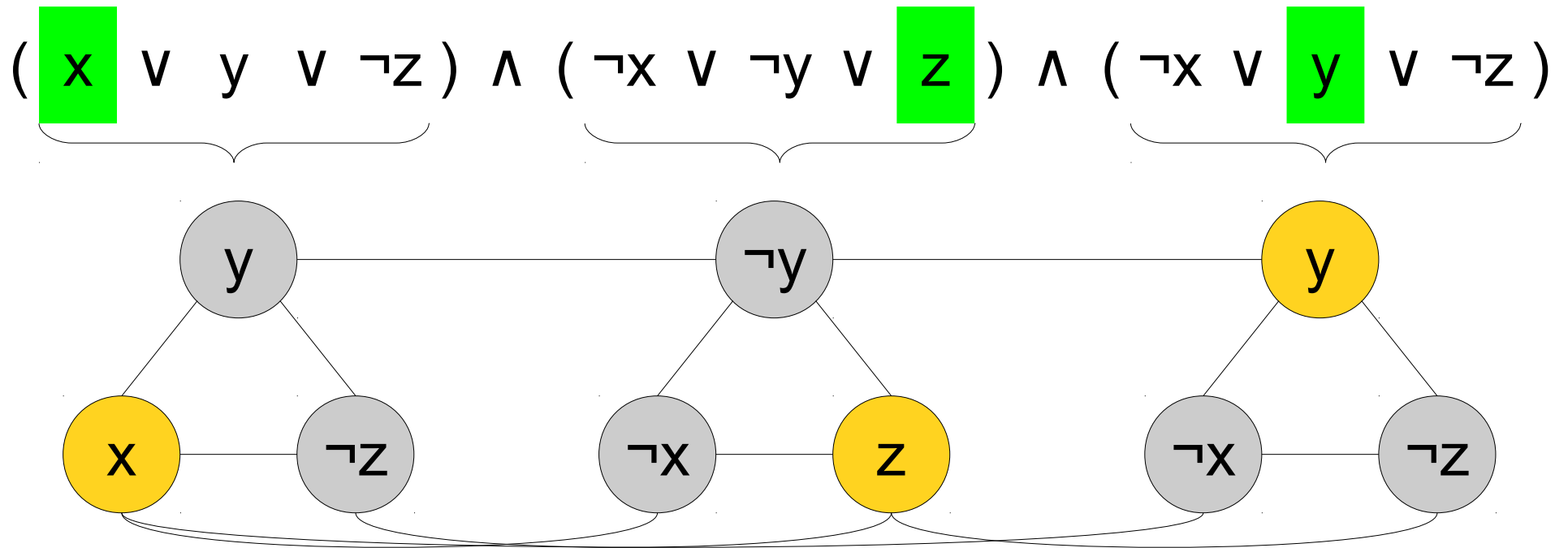
If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET



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

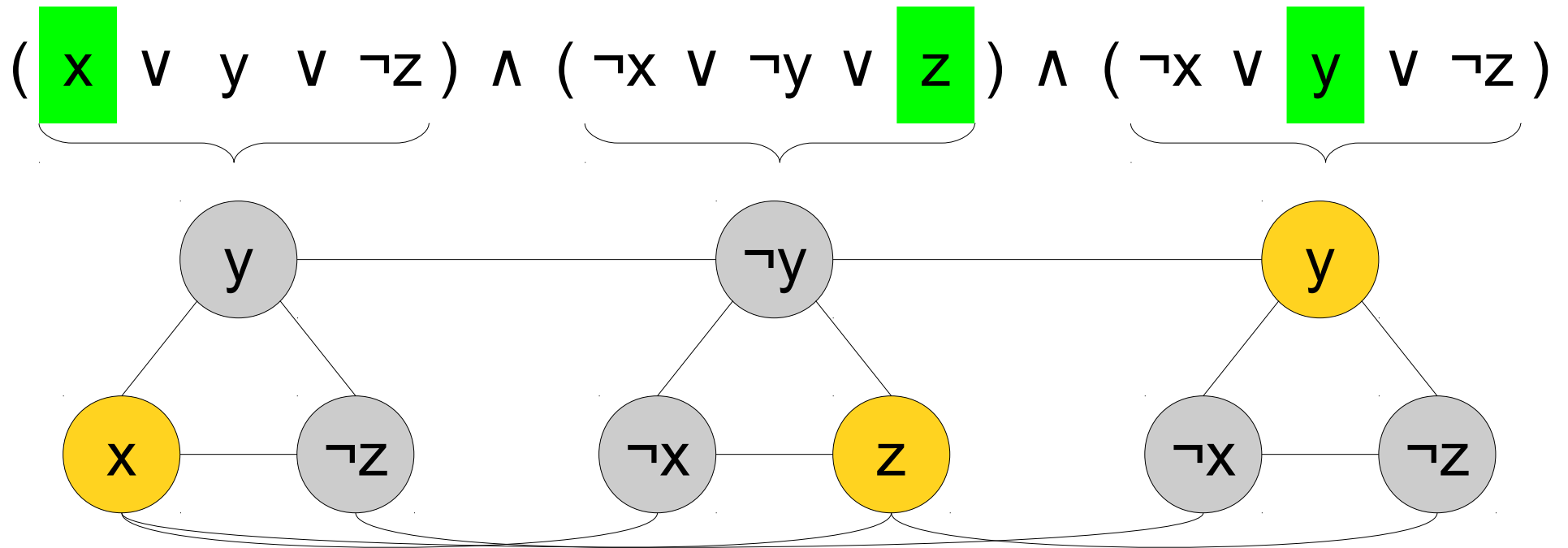
From 3SAT to INDSET



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

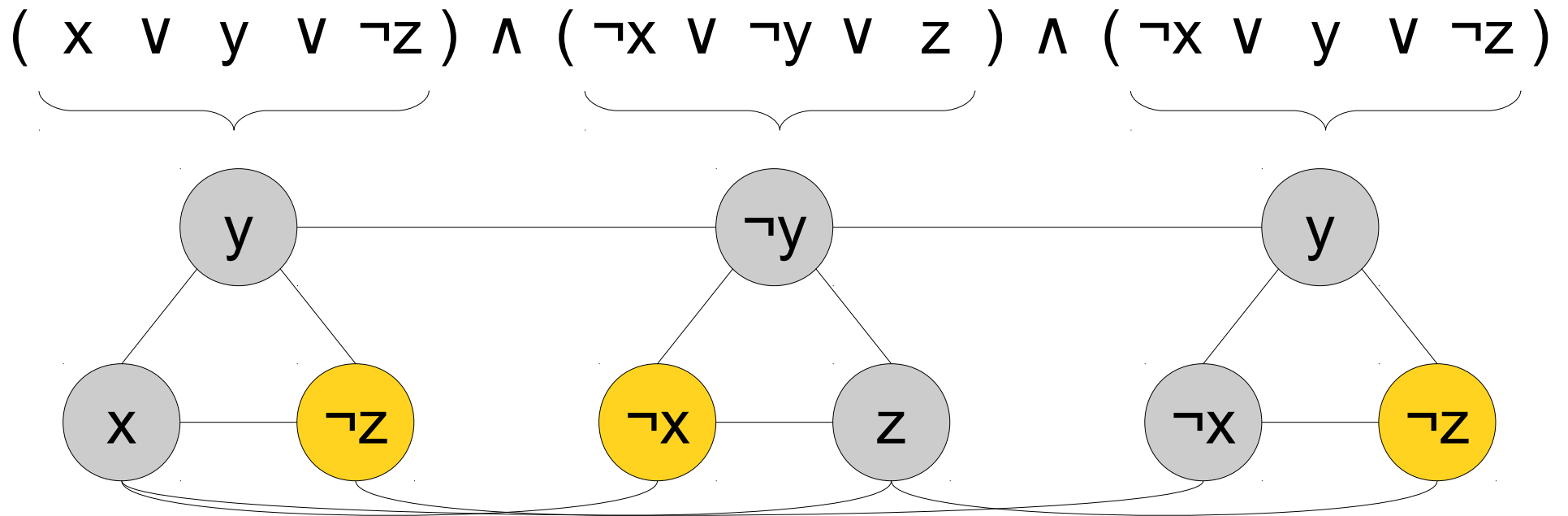
From 3SAT to INDSET

$x = \text{true}, y = \text{true}, z = \text{true}.$



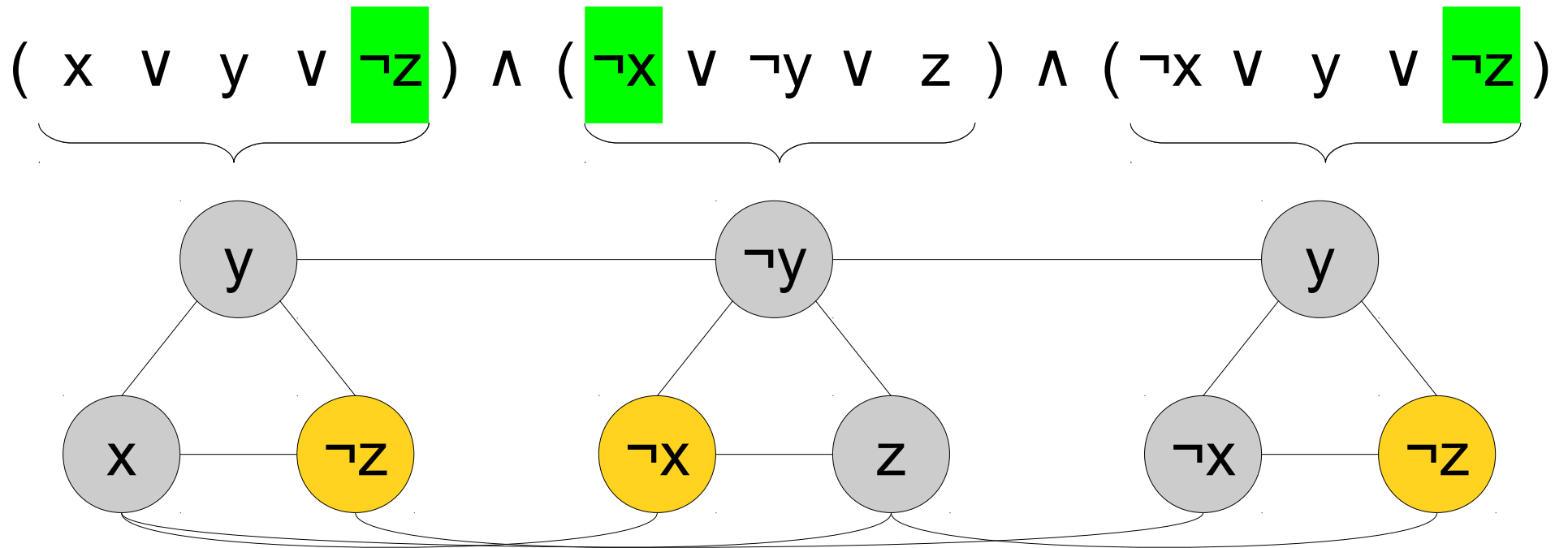
If this graph has an independent set of size three, the original formula is satisfiable.

From 3SAT to INDSET



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

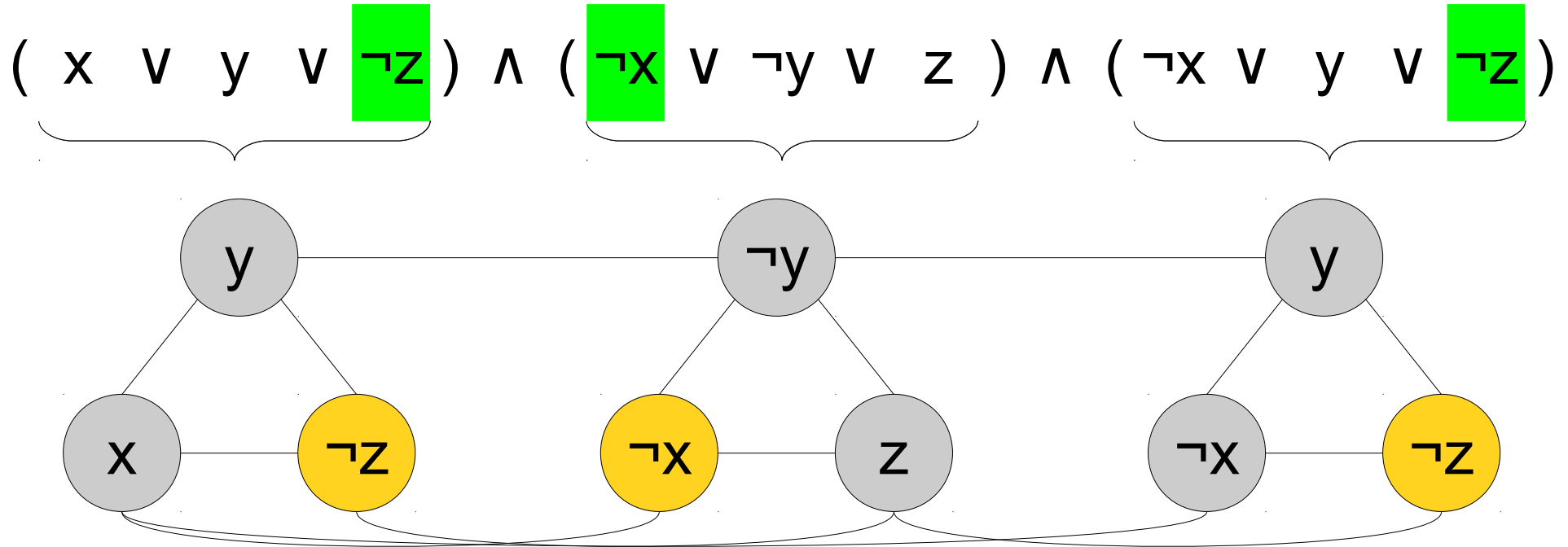
From 3SAT to INDSET



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

From 3SAT to INDSET

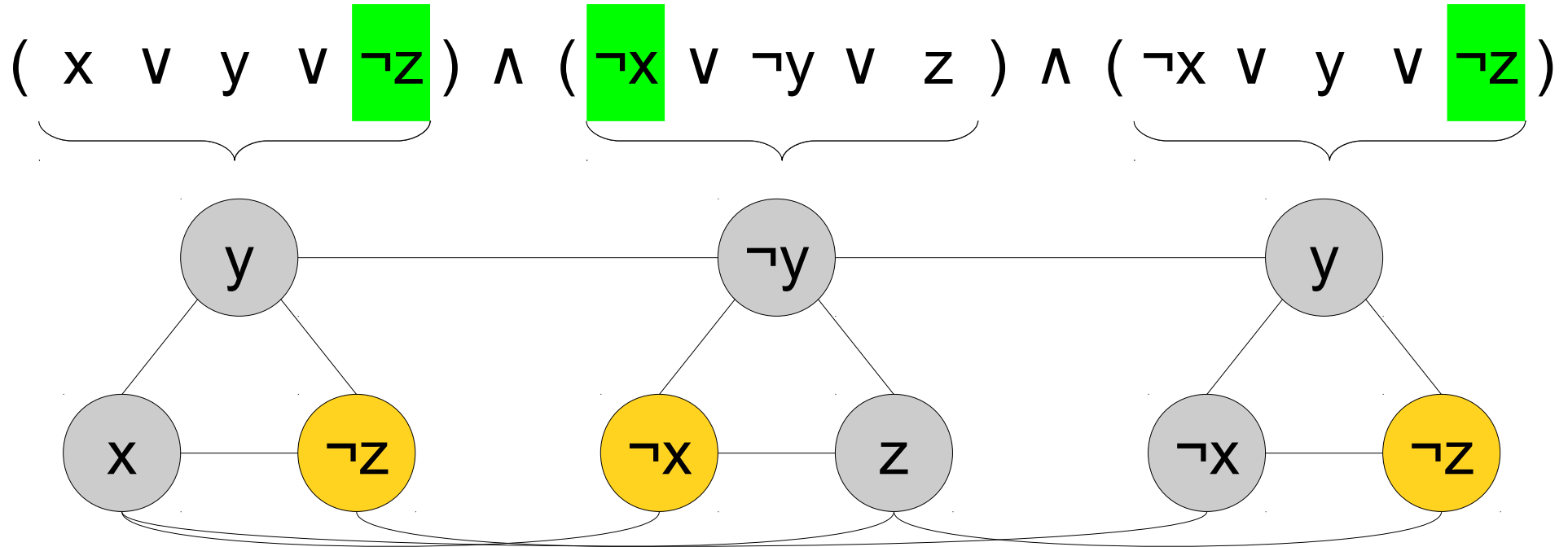
$x = \text{false}, y = ??, z = \text{false}.$



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

From 3SAT to INDSET

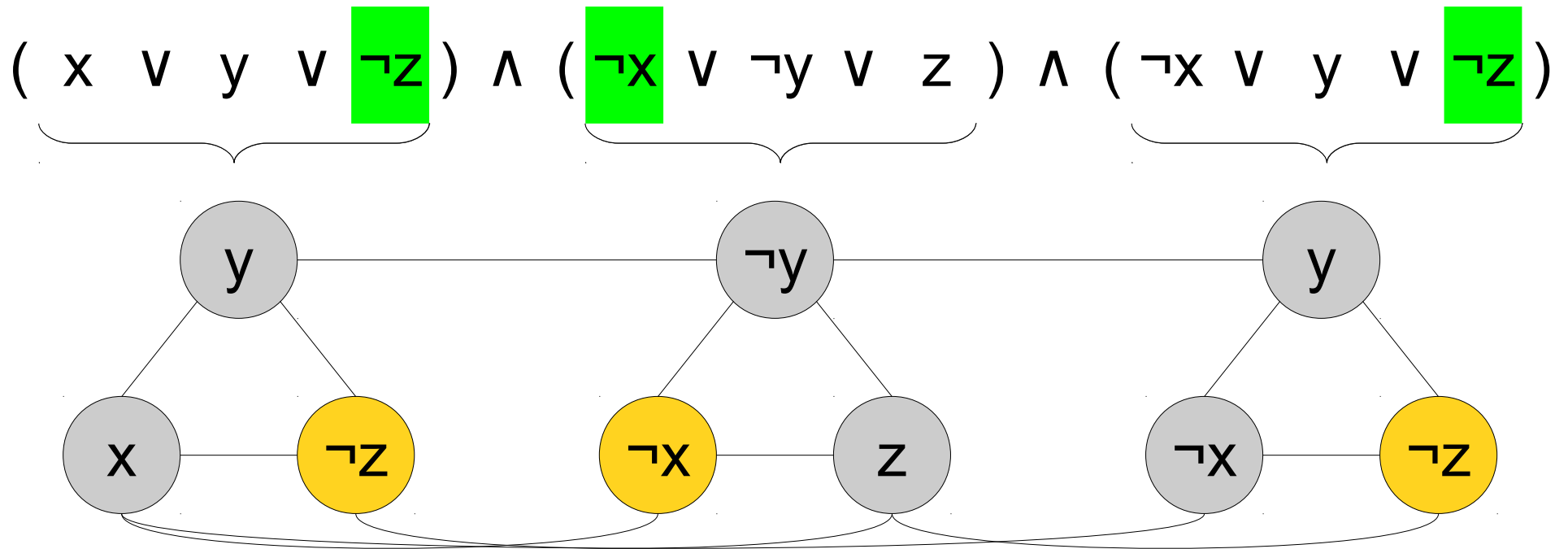
$x = \text{false}, y = \text{true}, z = \text{false}.$



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

From 3SAT to INDSET

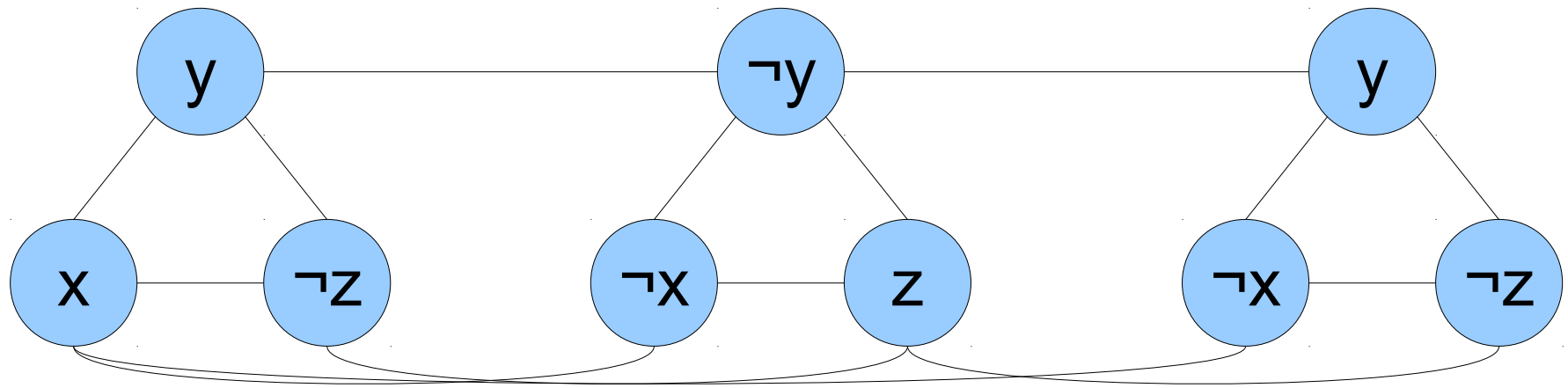
$x = \text{false}, y = \text{false}, z = \text{false}.$



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

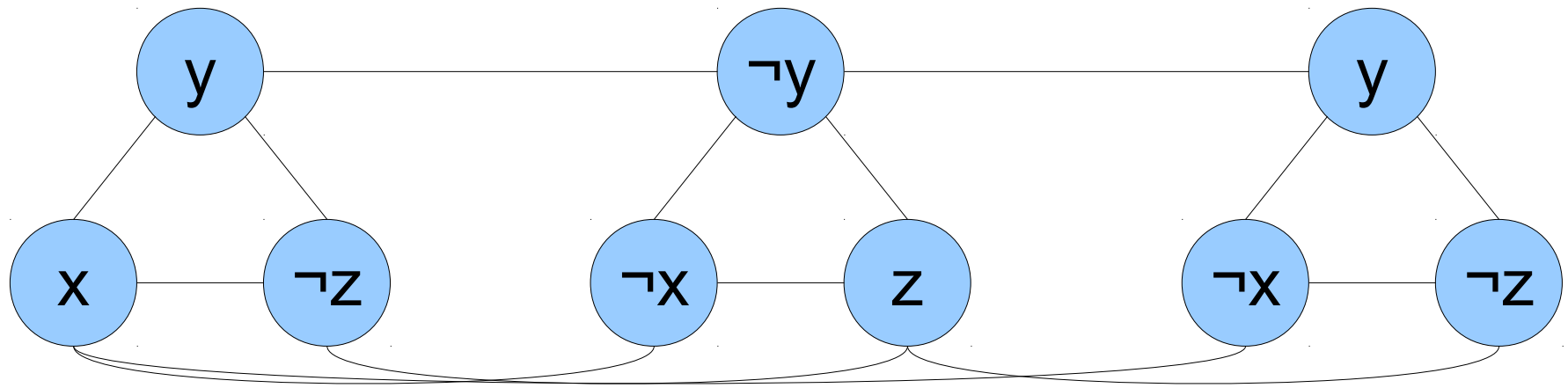
From 3SAT to INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



From 3SAT to INDSET

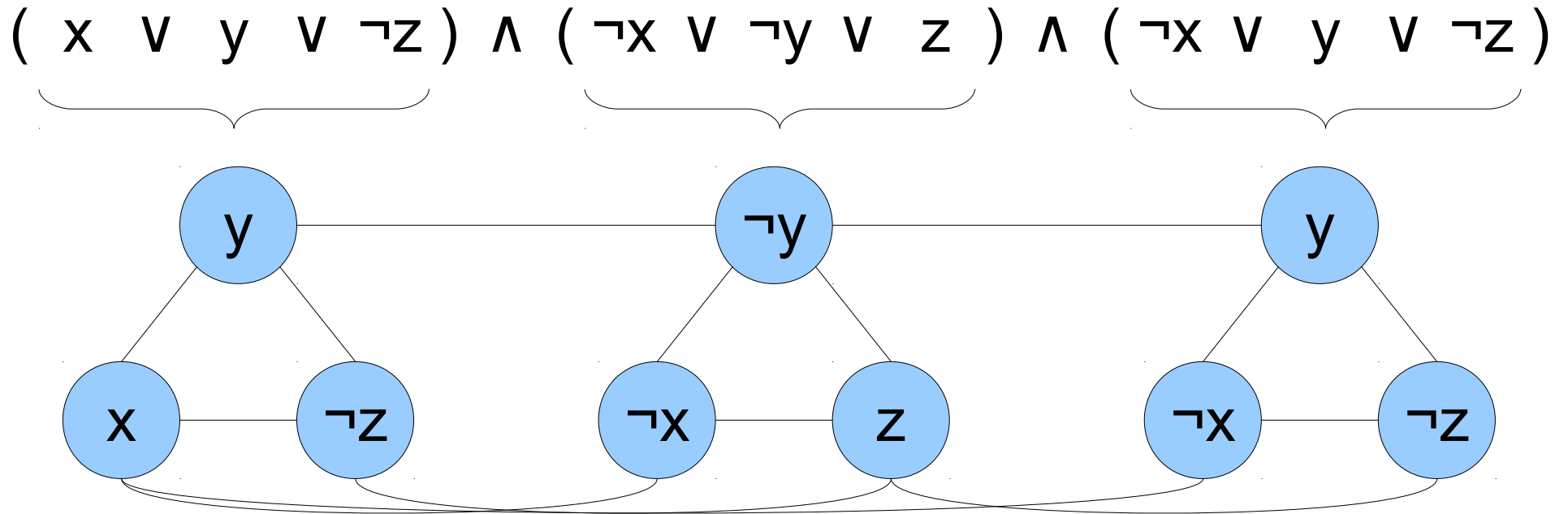
$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



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

From 3SAT to INDSET

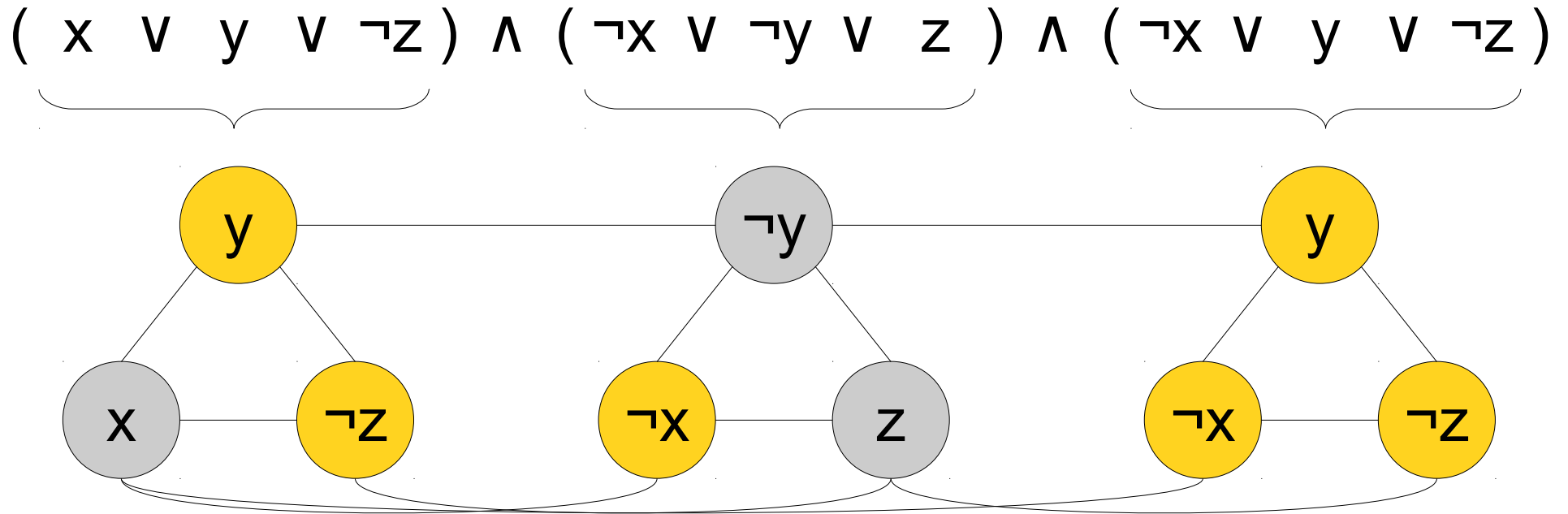
$x = \text{false}, y = \text{true}, z = \text{false}.$



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

From 3SAT to INDSET

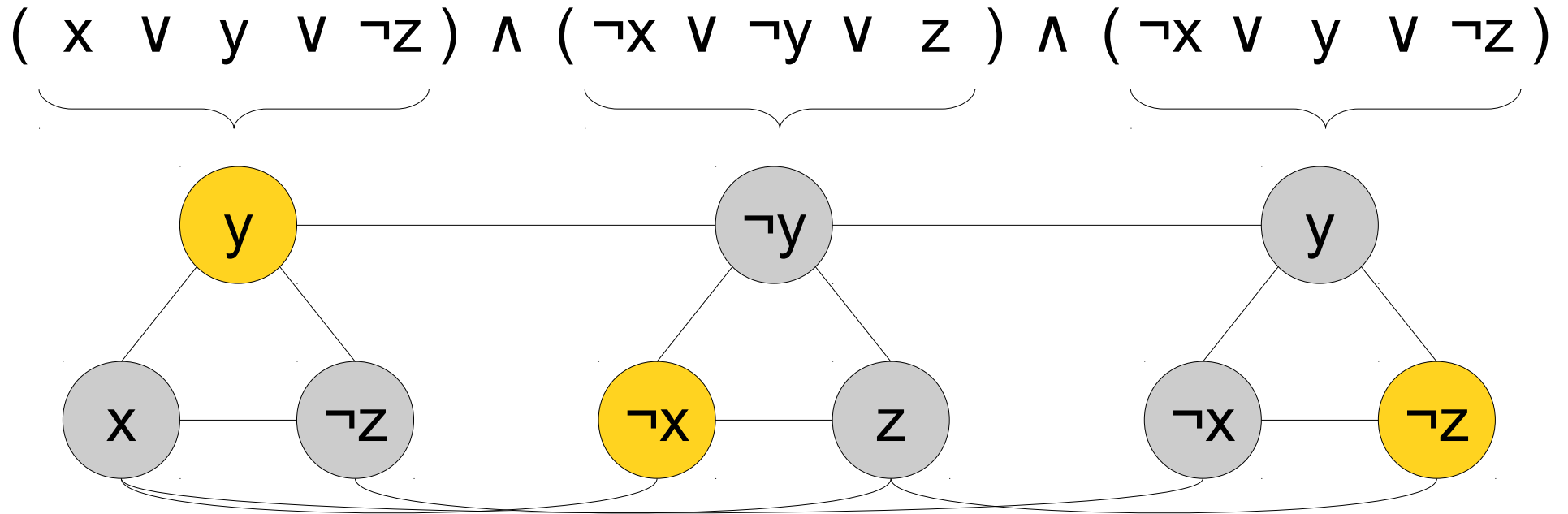
$x = \text{false}, y = \text{true}, z = \text{false}.$



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

From 3SAT to INDSET

$x = \text{false}, y = \text{true}, z = \text{false}.$



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

From 3SAT to INDSET

- Let $\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_n$ be a 3-CNF formula.
- Construct the graph G as follows:
 - For each clause $C_i = x_1 \vee x_2 \vee x_3$, 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 \in NPC

- ***Theorem:*** INDSET is **NP**-complete.
- ***Proof sketch:*** We just showed that INDSET \in **NP** and that $3\text{SAT} \leq_p \text{INDSET}$. 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 is next Friday, December 12 from 12:15PM – 3:15PM.
- Room locations divvied up by last name:
 - **Abr** - **Sad**: Go to Cemex Auditorium
 - **Sal** - **Zie**: Go to Cubberly Auditorium
- Same format as midterms: three hours, closed-book, closed-computer, open one page of notes.
- Cumulative exam, but slightly weighted towards material from PS7 – PS8.
- **P** and **NP** will be tested, but not in-depth.

Practice Final Exam

- We will be holding a practice final exam on Monday, December 8 from 3:00PM – 6:00PM in Cemex Auditorium.
- Same format as the practice midterms: show up, give it your best shot, and we'll answer questions afterwards.
- Practice exam will be posted online the night of the exam; solutions will be released in hardcopy and sent out to the SCPD distribution list.

Cumulative Review Problems

- I've just posted a set of 15 cumulative review problems to the course website under the “handouts” section.
- Feel free to work through these for extra practice. We'll release solutions later this week.

Your Questions

“How do you tell the difference between liking something and thinking it's interesting vs. being passionate about something and building a career out of it?”

“I'm still stuck on the intuition behind the proof that a language is not in **R**. Why does that logic not apply to any language at all? Can you give an example of trying to use that proof on a language that *is* in **R**, and show where it breaks down?”

Let $L = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts } w \}$

Assume $L \in \mathbf{R}$.

That means there's a TM that given any TM M and string w , *decides* whether M accepts w .

Now, build this machine M :

$M =$ “On input w :

Have M get its encoding $\langle M \rangle$.

Decide whether M accepts w .

If so, M rejects w .

If not, M accepts w .”

Let $L = a^*b^*$.

Assume $L \in \mathbf{R}$.

That means there's a TM that given any string w ,
decides whether w matches a^*b^* .

Now, build this machine M :

$M =$ “On input w :

Have M get its encoding $\langle M \rangle$.

Decide whether w has the form a^*b^* .

If so, M rejects w .

If not, M accepts w .”

“I had some trouble with the intuition behind question 7 on the last problem set, especially with the last five or six parts. Do you have any advice on how to think through those types of questions?”

“I love programming but I am interested in applying the skill to social causes in a meaningful way. What advice do you have for me/what fields do you recommend I explore?”

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 connected 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.

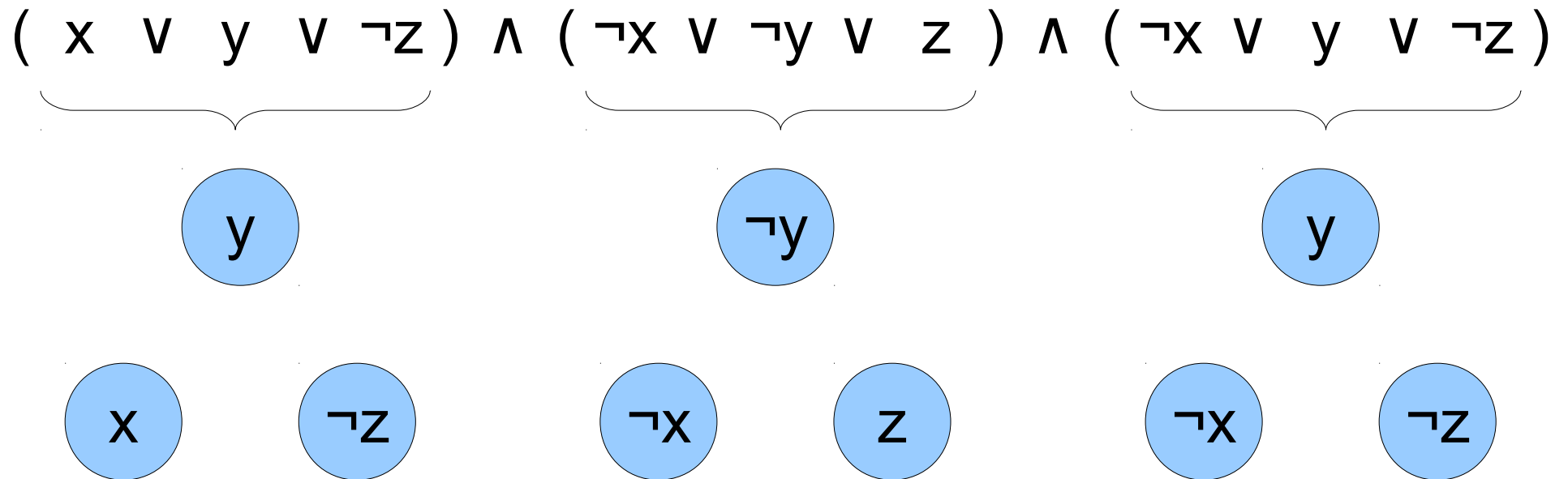
Gadgets in INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

Gadgets in INDSET

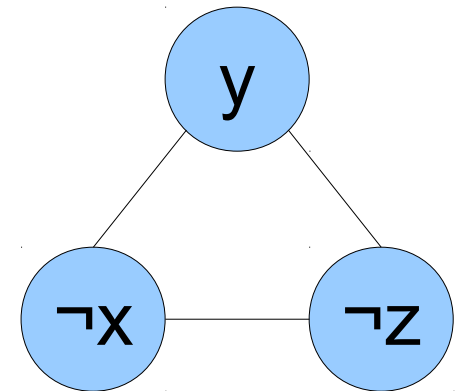
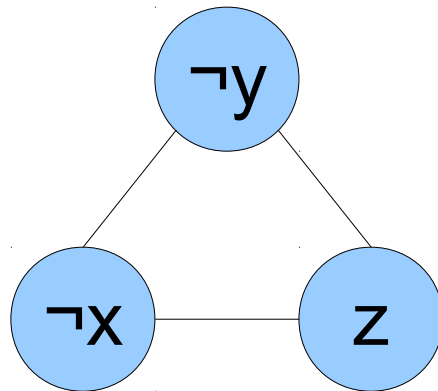
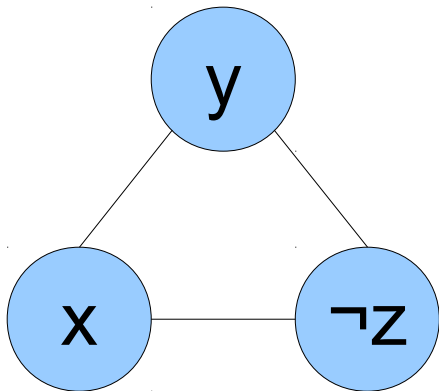
$$\underbrace{(x \vee y \vee \neg z)} \wedge \underbrace{(\neg x \vee \neg y \vee z)} \wedge \underbrace{(\neg x \vee y \vee \neg z)}$$

Gadgets in INDSET



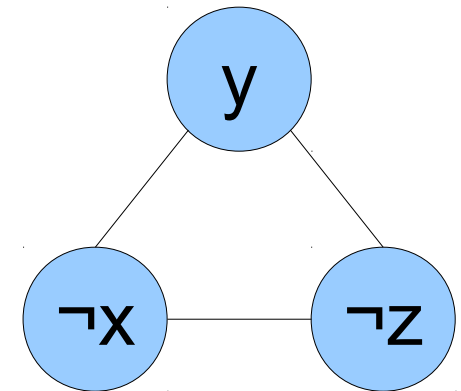
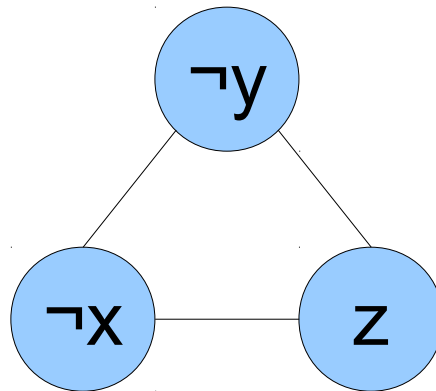
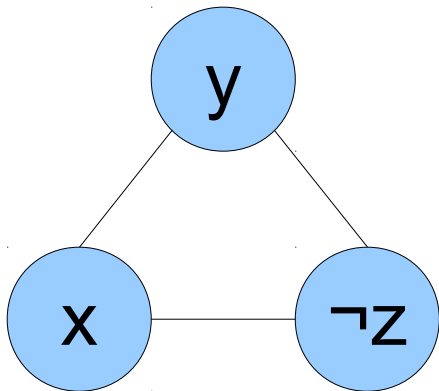
Gadgets in INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



Gadgets in INDSET

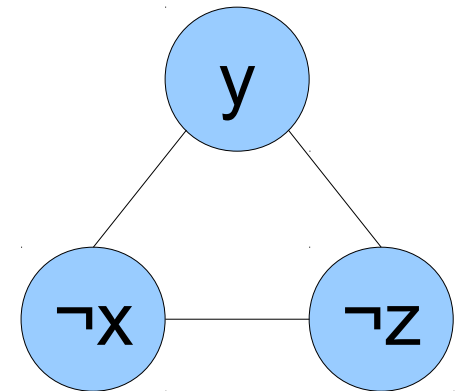
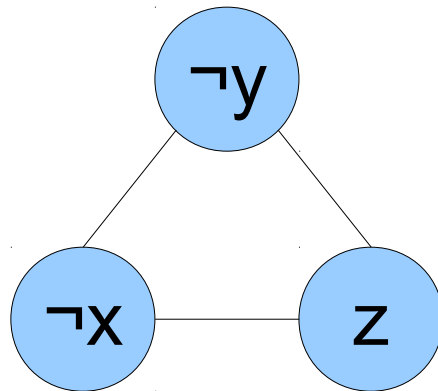
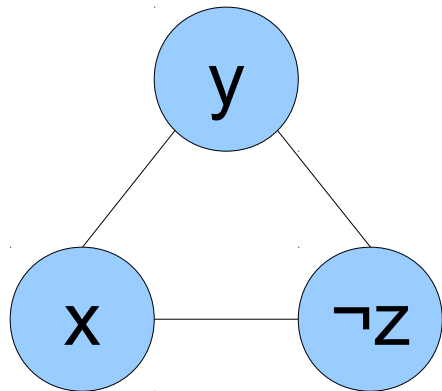
$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



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

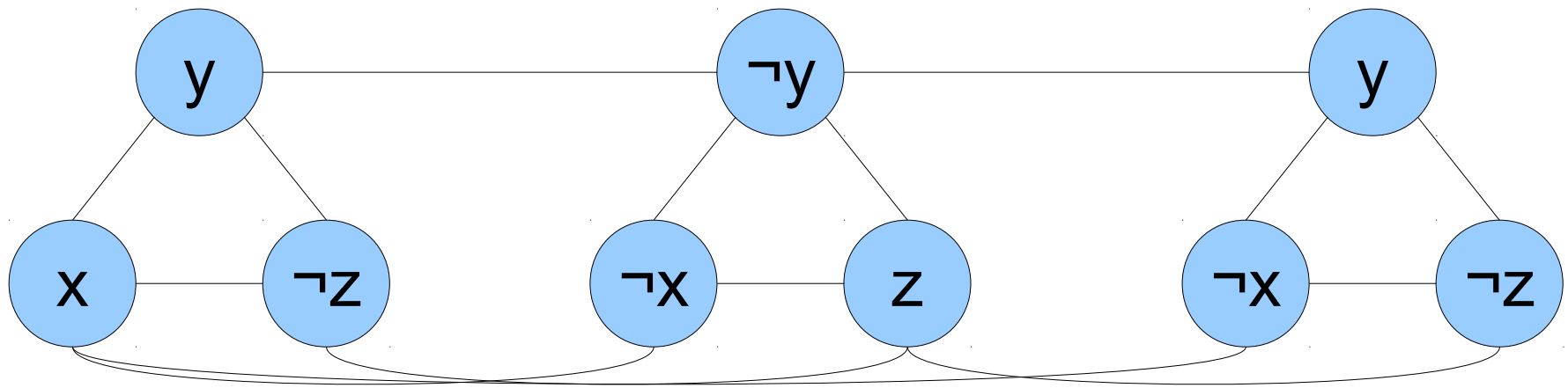
Gadgets in INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



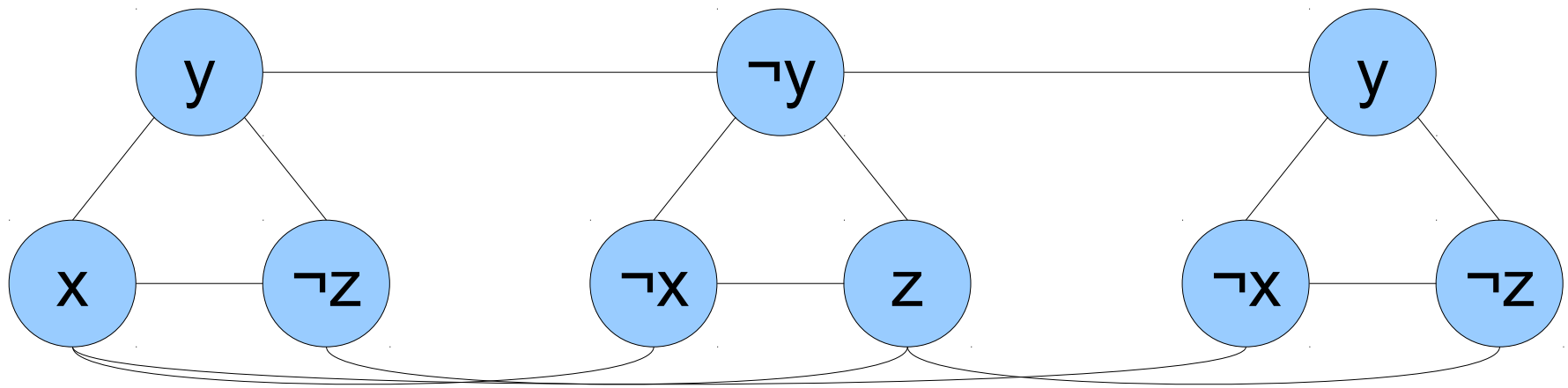
Gadgets in INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



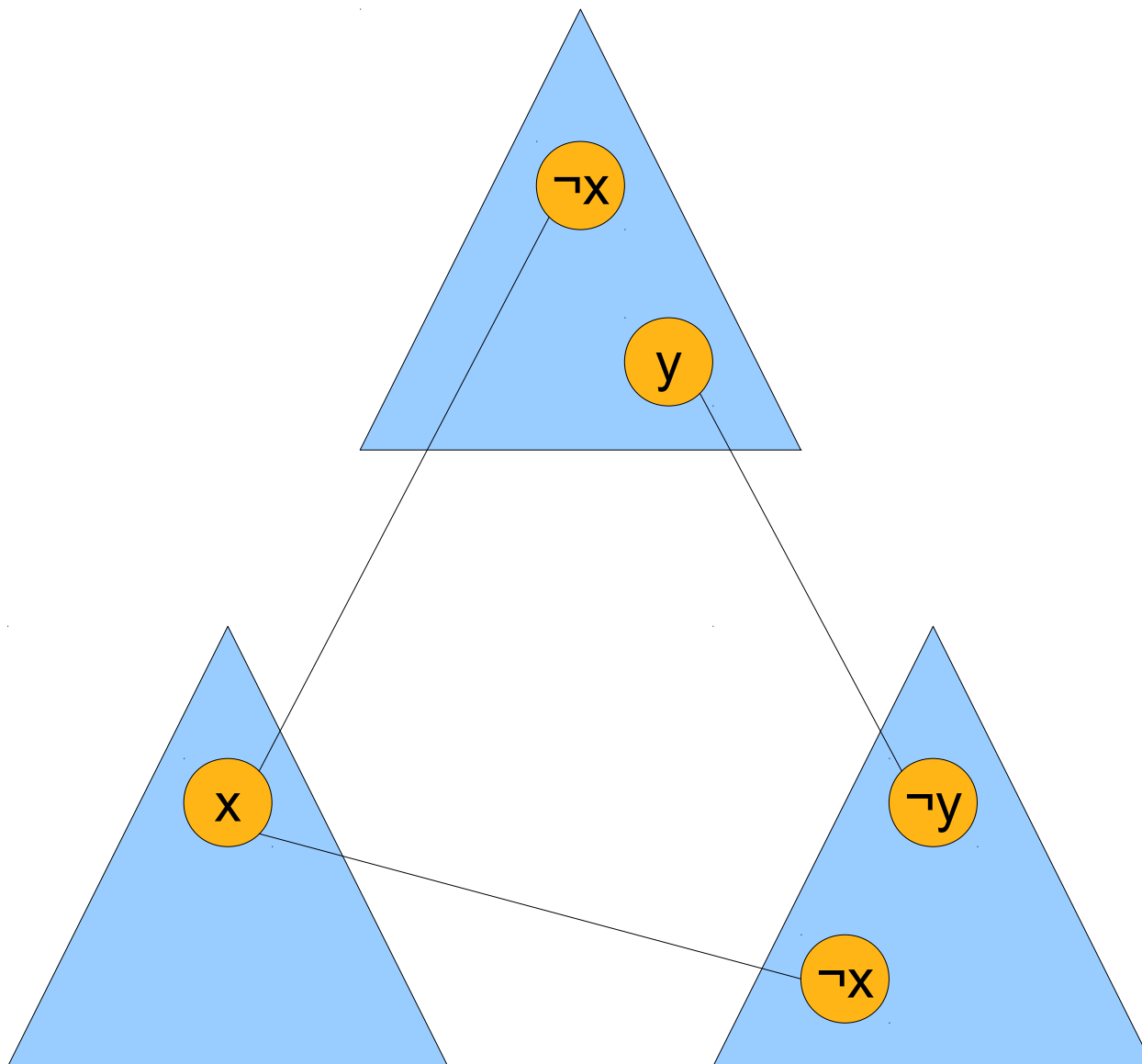
Gadgets in INDSET

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



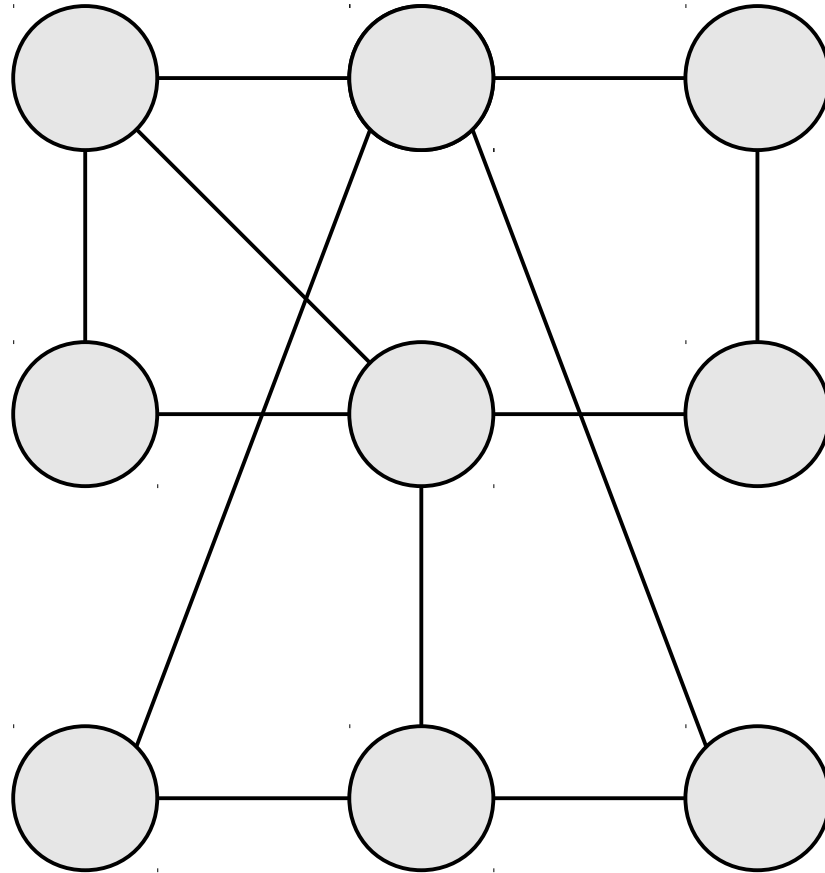
These connections ensure that the solutions to each gadget are linked to one another.

Gadgets in INDSET

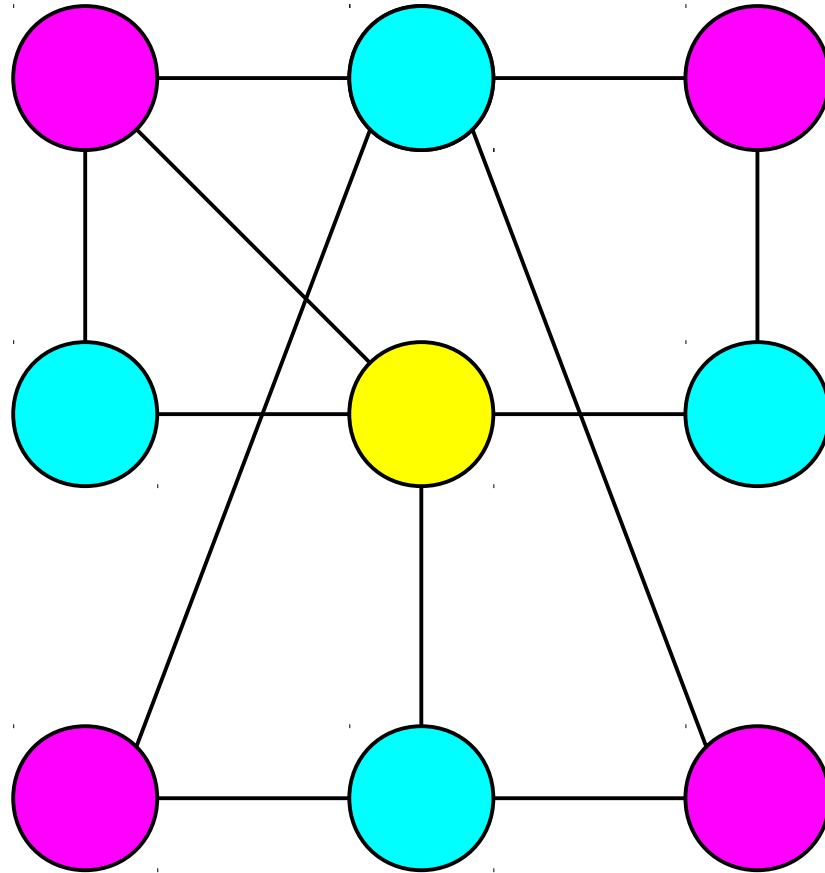


A More Complex Reduction

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 = { $\langle G \rangle$ | G is an undirected
graph with a legal 3-coloring. }**

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

3COLOR \in NP

- We can prove that 3COLOR \in NP by designing a polynomial-time nondeterministic TM for 3COLOR.
- M = “On input $\langle G \rangle$:
 - *Nondeterministically* guess an assignment of colors to the nodes.
 - *Deterministically* check whether it is a 3-coloring.
 - 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.

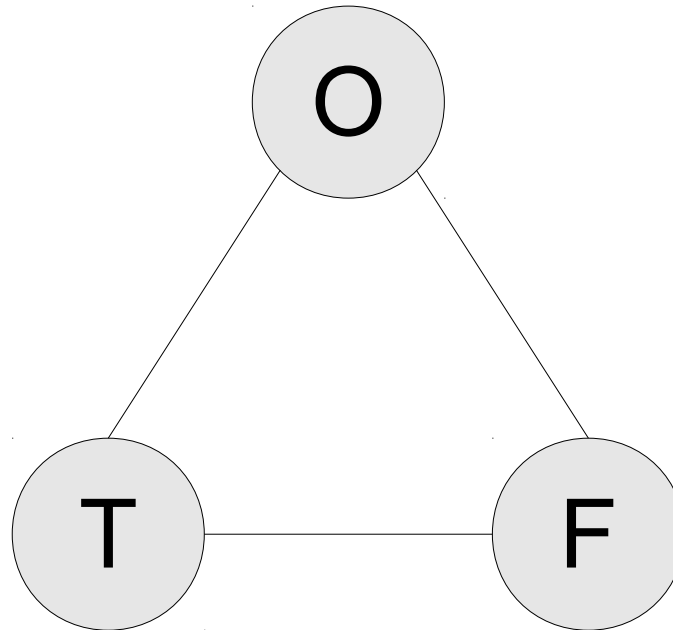
Why Not Two Colors?

- 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!

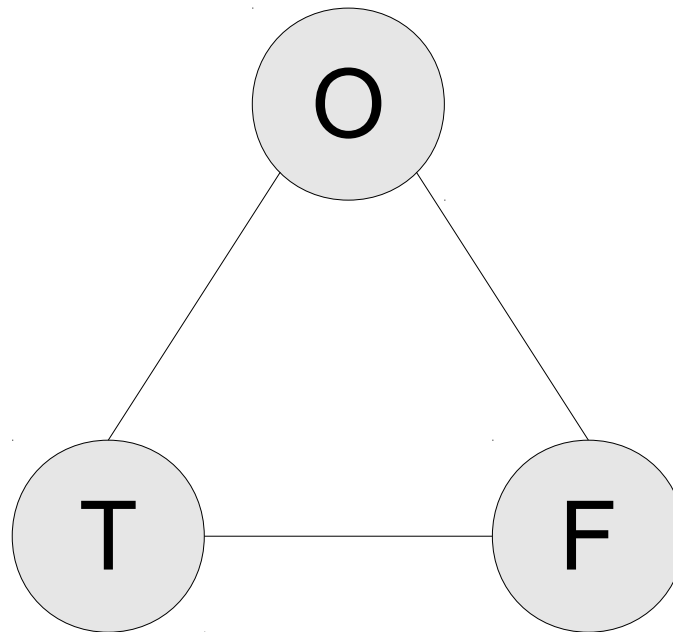
From 3SAT to 3COLOR

- 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*

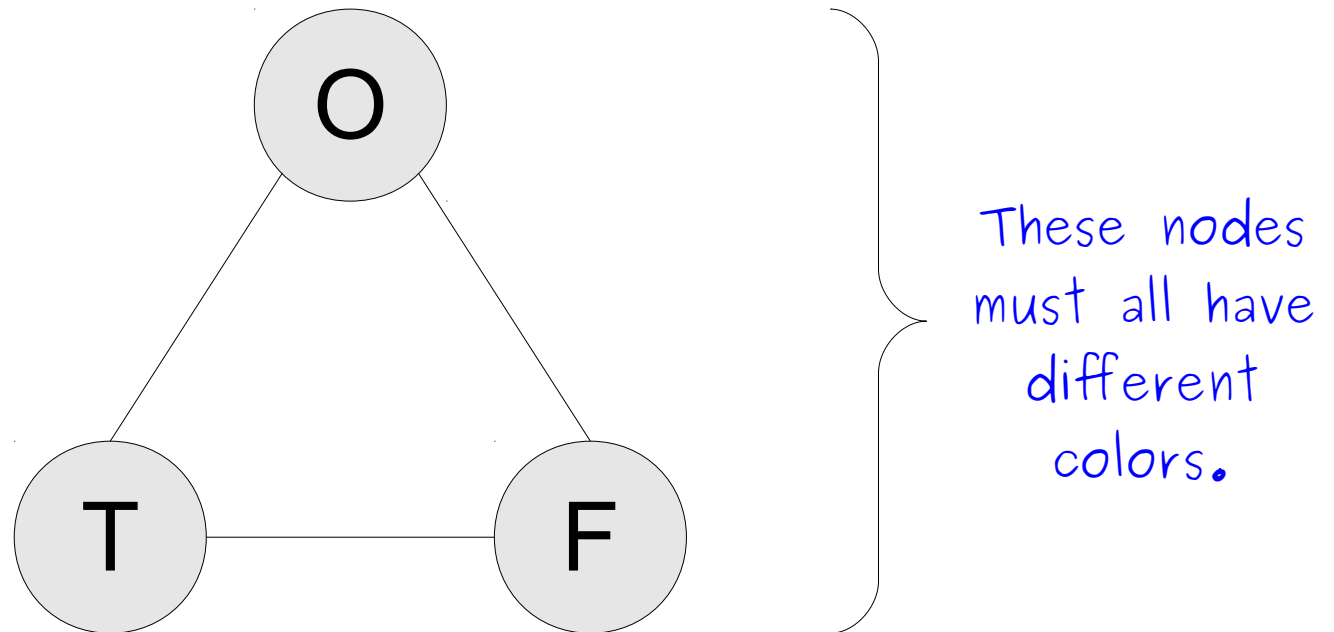


Gadget One: Assigning Meanings



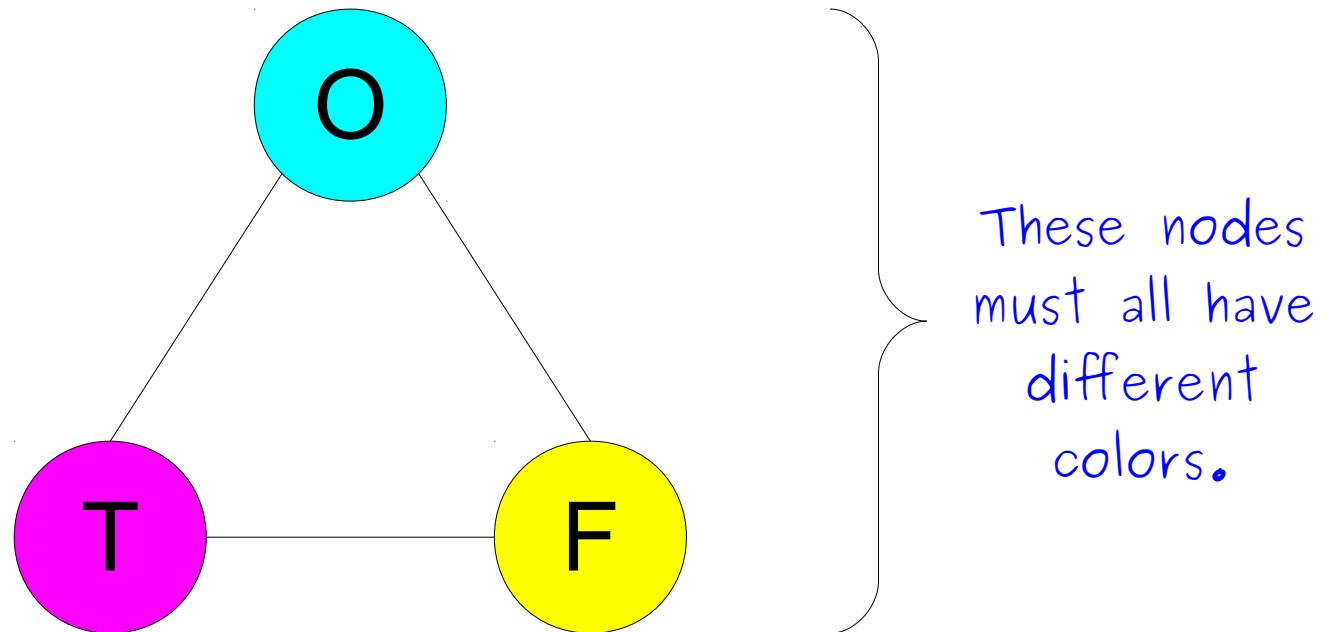
These nodes
must all have
different
colors.

Gadget One: Assigning Meanings



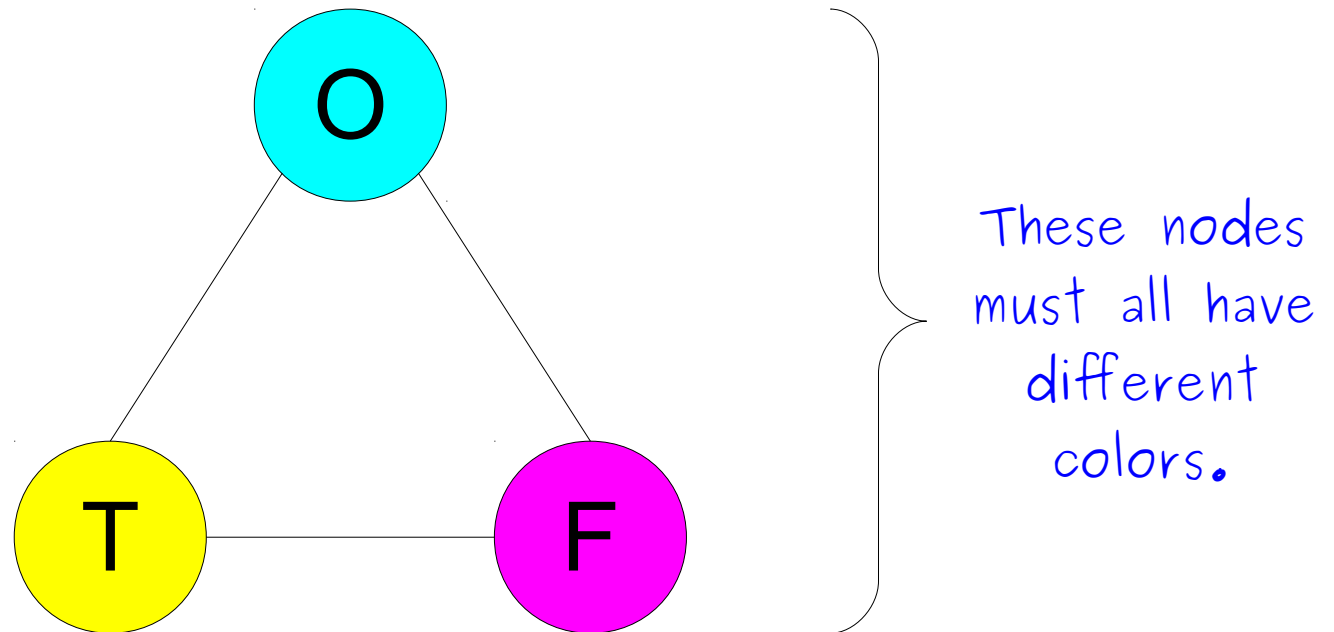
The color assigned to T will be interpreted as "true."
The color assigned to F will be interpreted as "false."
We do not associate any special meaning with O.

Gadget One: Assigning Meanings



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

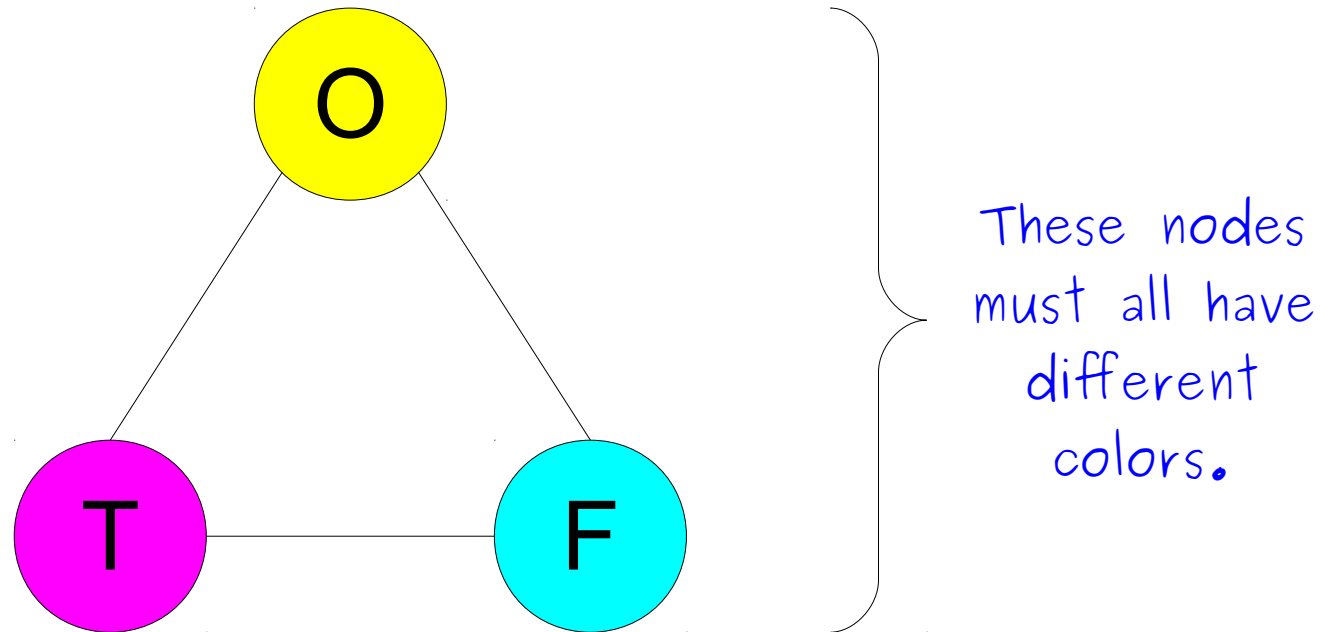


The color assigned to T will be interpreted as "true."

The color assigned to F will be interpreted as "false."

We do not associate any special meaning with O.

Gadget One: Assigning Meanings



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

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

Gadget Two: Forcing a Choice

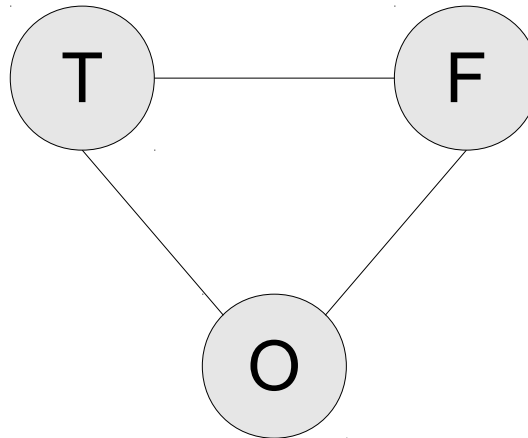
$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

Gadget Two: Forcing a Choice

$$(x \vee y \vee \boxed{\neg z}) \wedge (\neg x \vee \neg y \vee \boxed{z}) \wedge (\neg x \vee y \vee \boxed{\neg z})$$

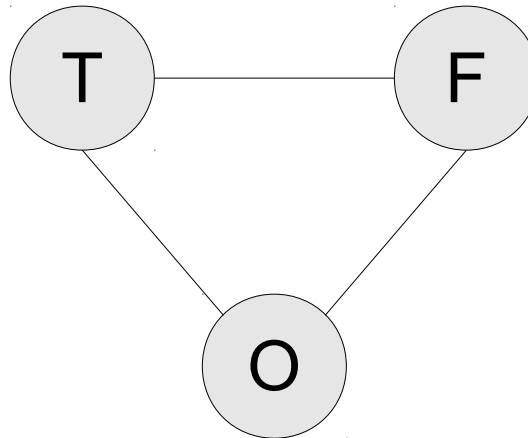
Gadget Two: Forcing a Choice

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



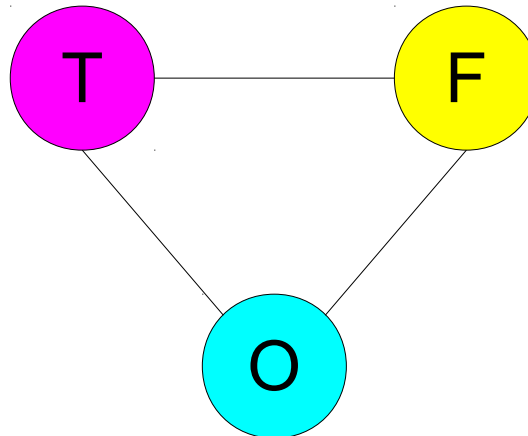
Gadget Two: Forcing a Choice

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



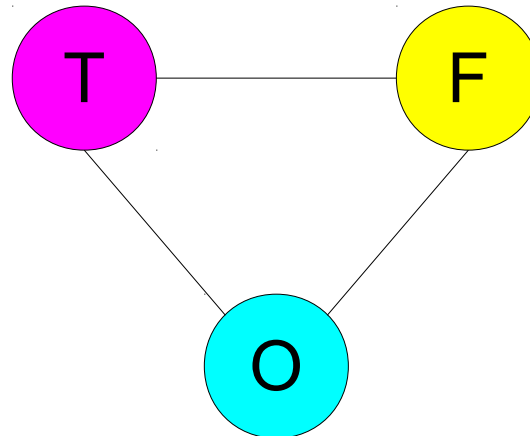
Gadget Two: Forcing a Choice

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



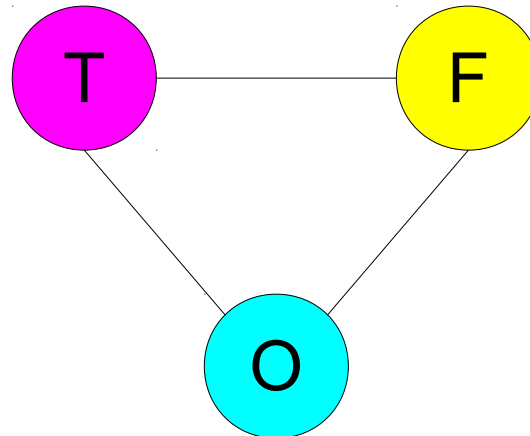
Gadget Two: Forcing a Choice

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



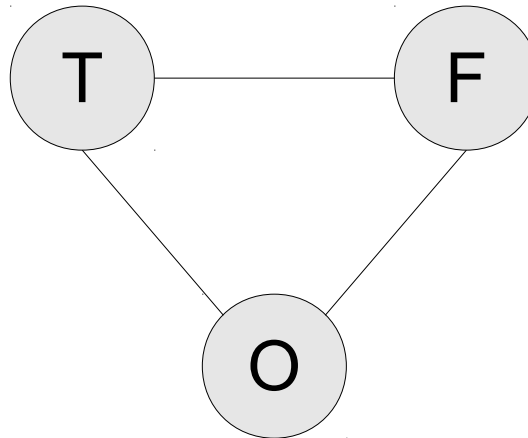
Gadget Two: Forcing a Choice

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



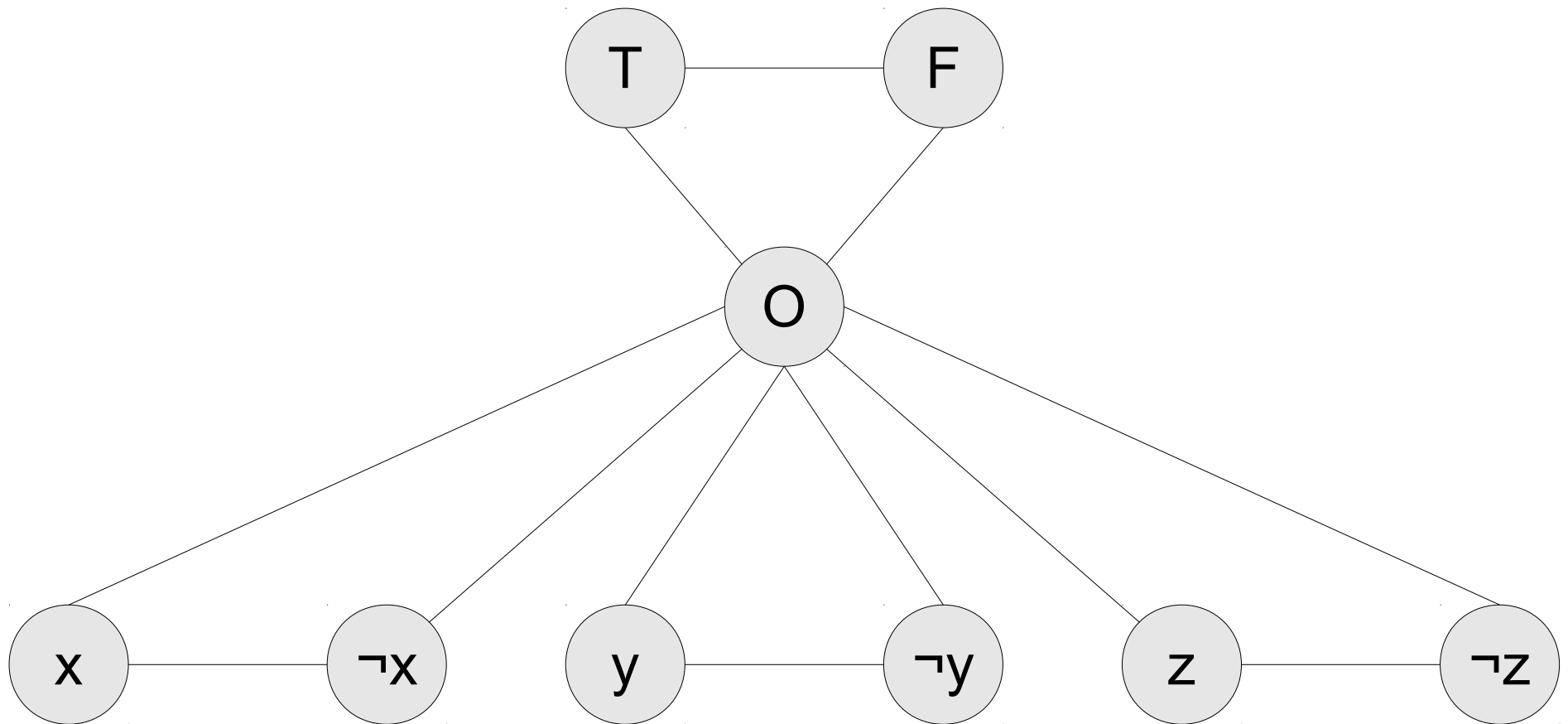
Gadget Two: Forcing a Choice

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



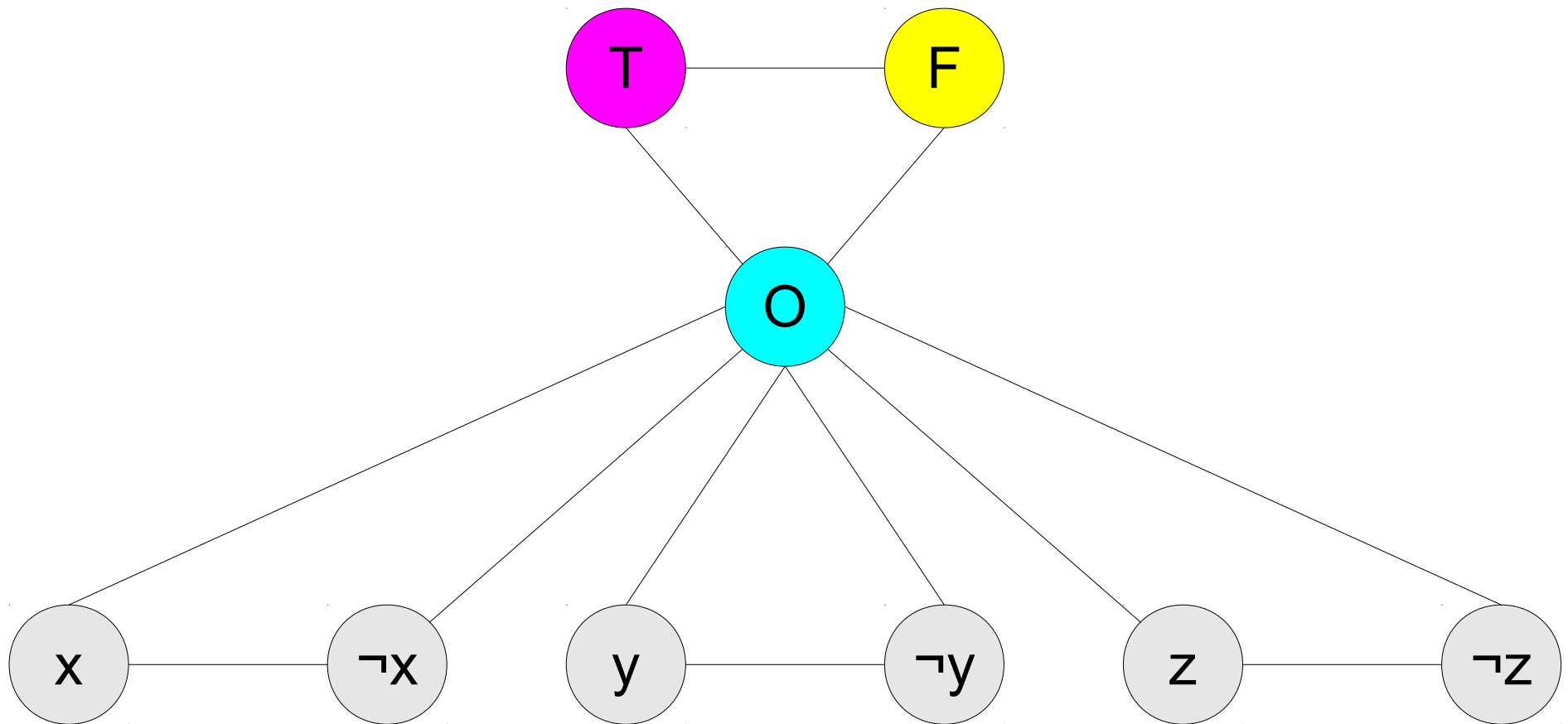
Gadget Two: Forcing a Choice

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



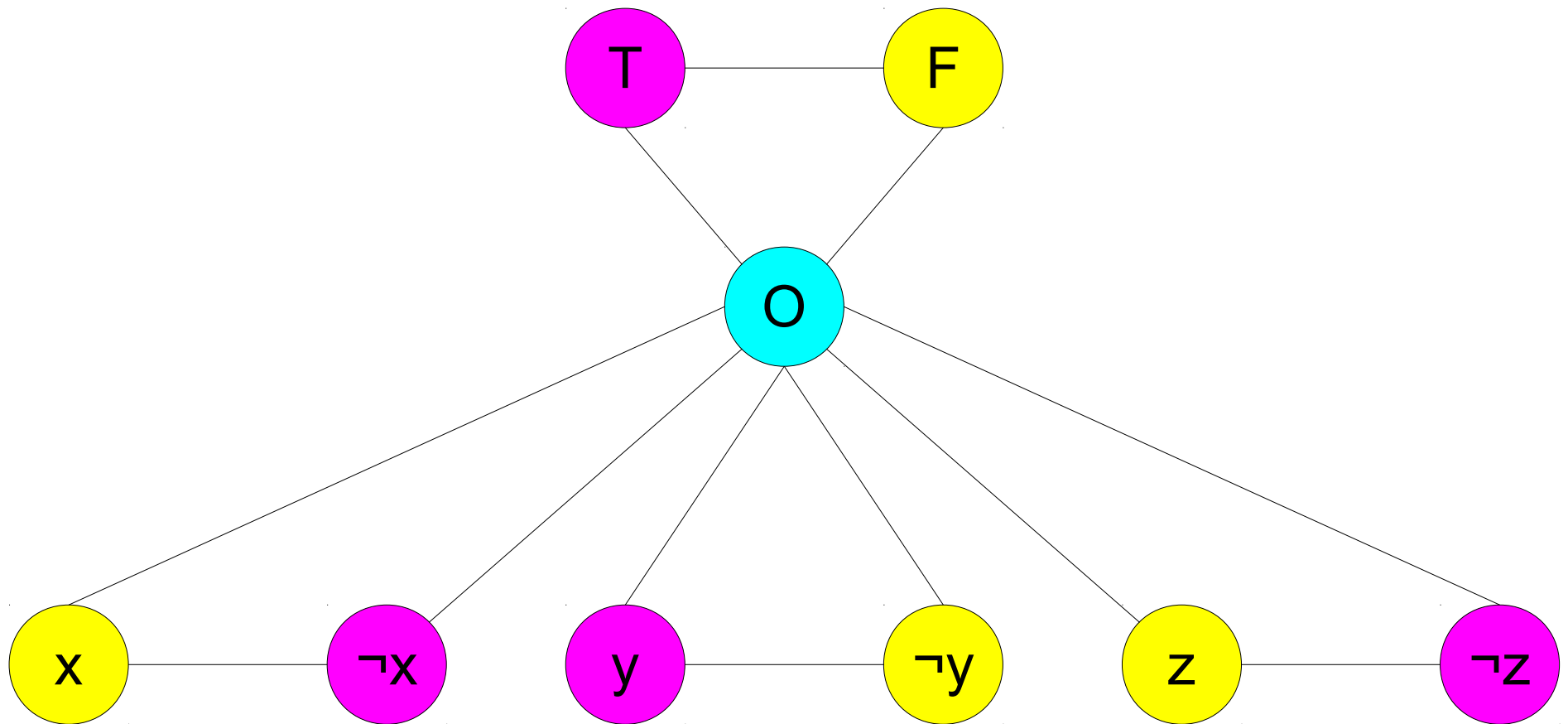
Gadget Two: Forcing a Choice

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



Gadget Two: Forcing a Choice

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

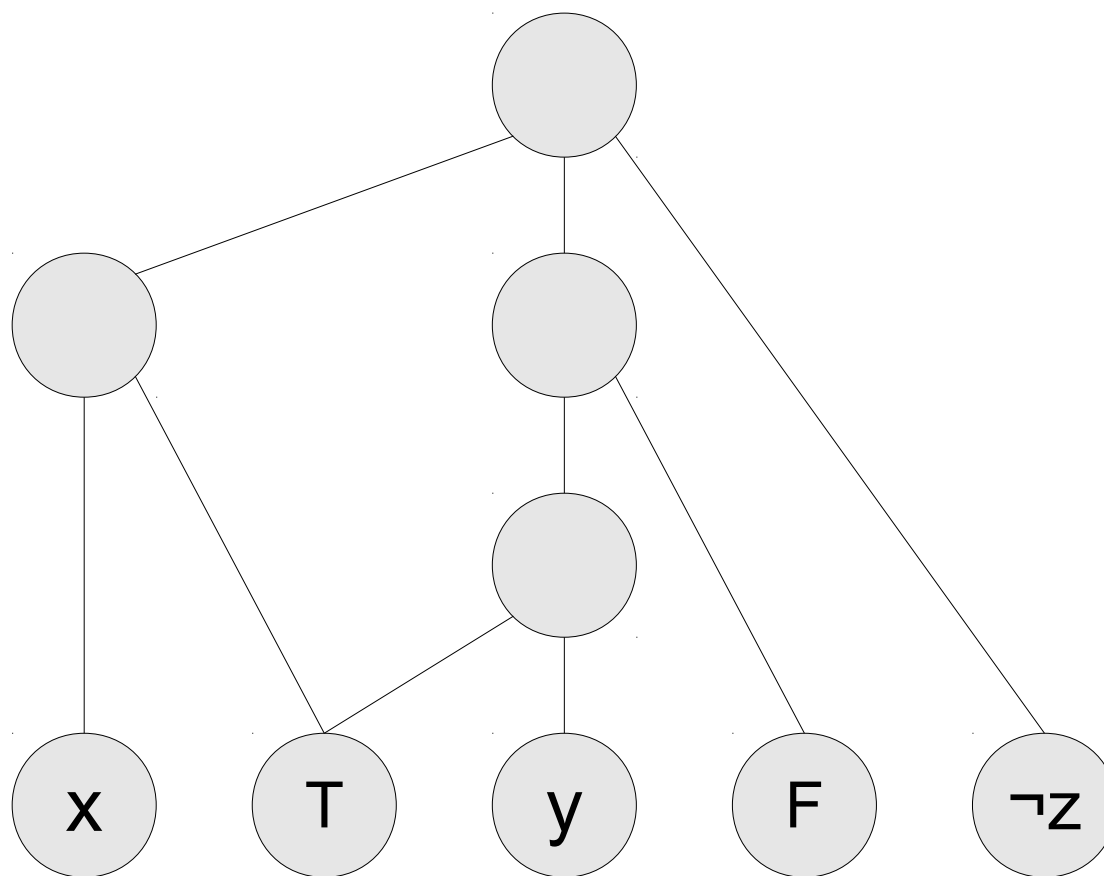


Gadget Three: Clause Satisfiability

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

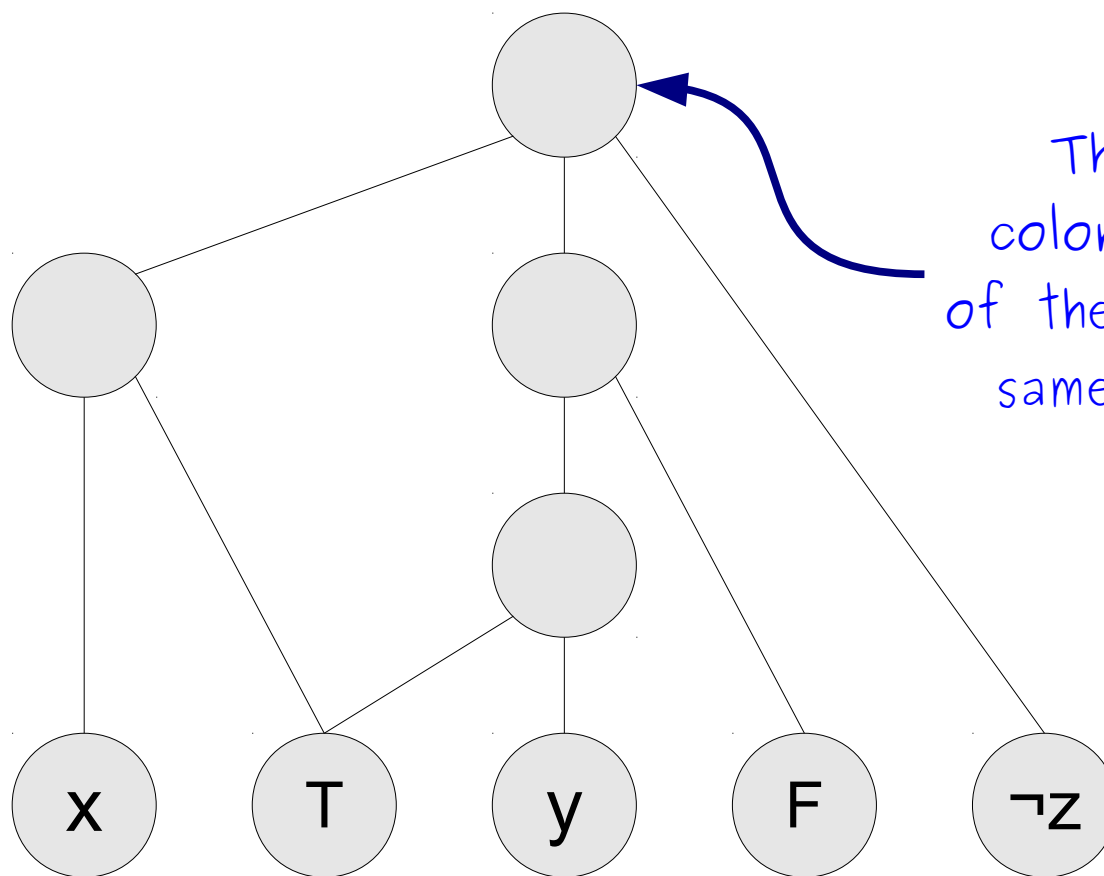
Gadget Three: Clause Satisfiability

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



Gadget Three: Clause Satisfiability

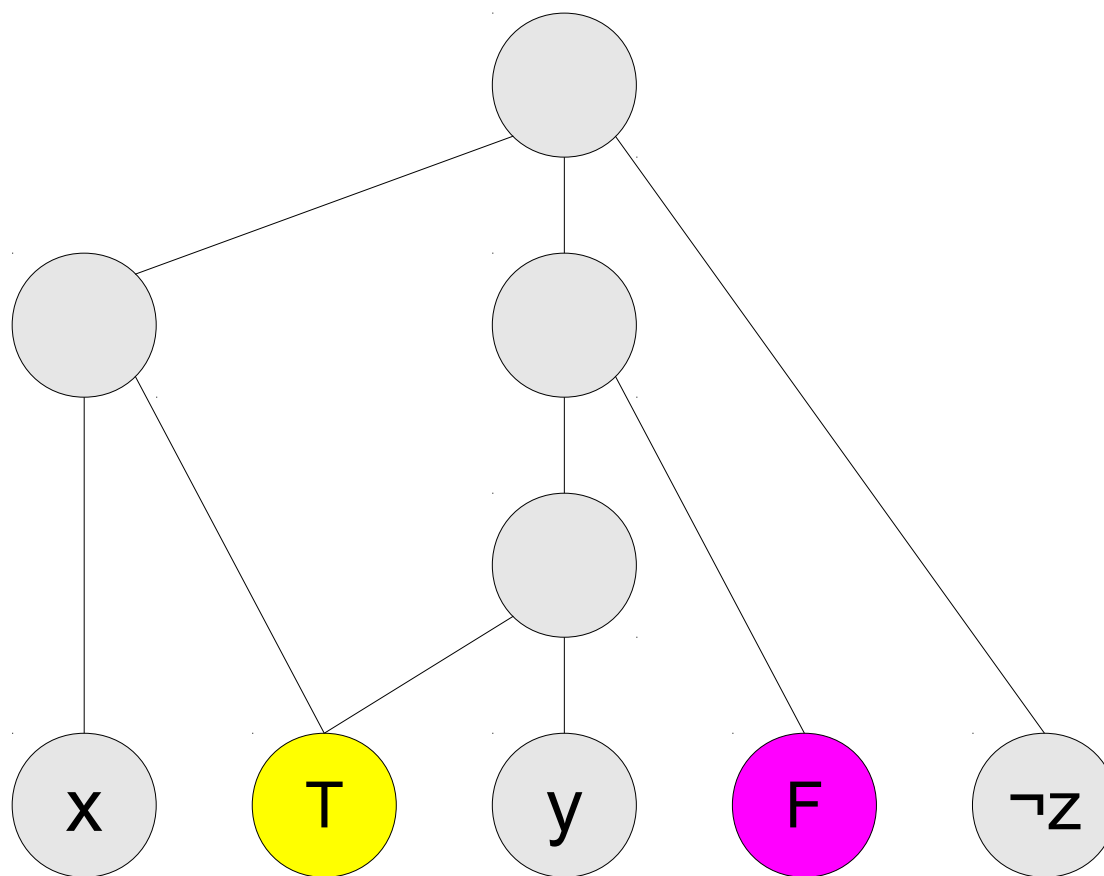
$(x \vee y \vee \neg z)$



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

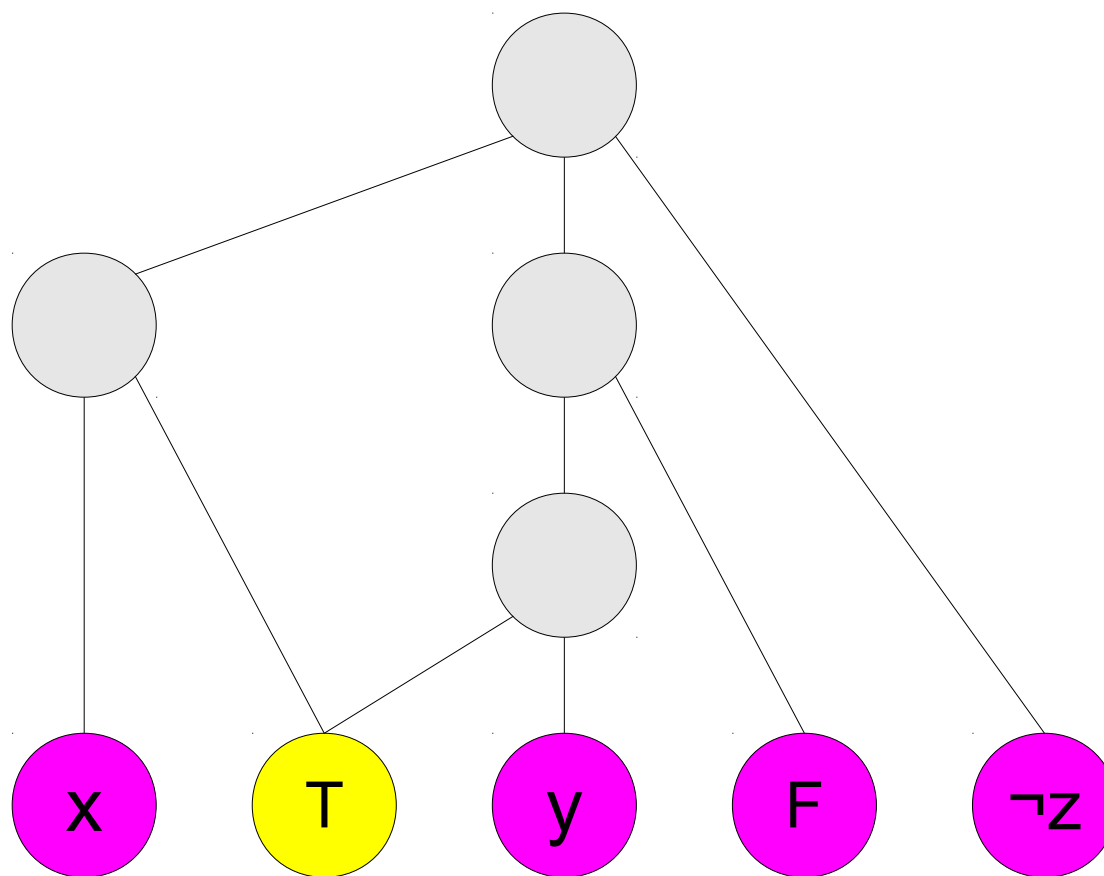
Gadget Three: Clause Satisfiability

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



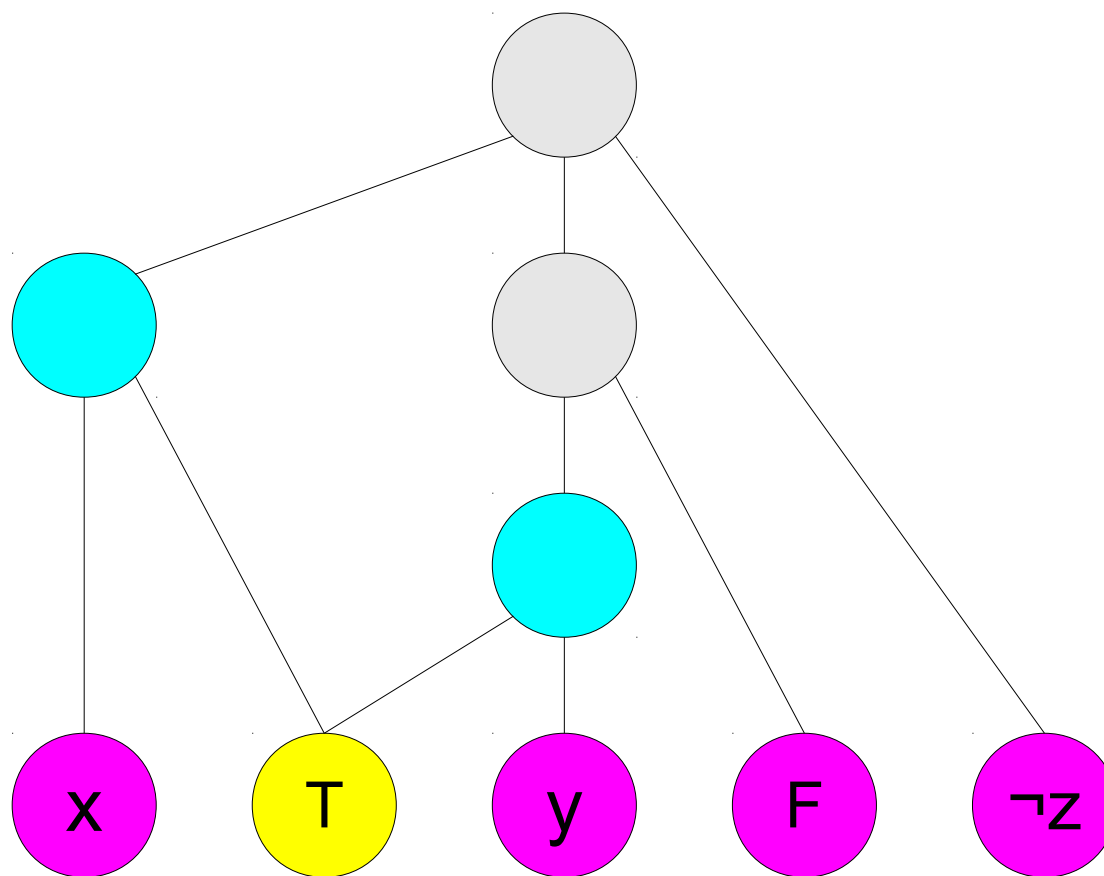
Gadget Three: Clause Satisfiability

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



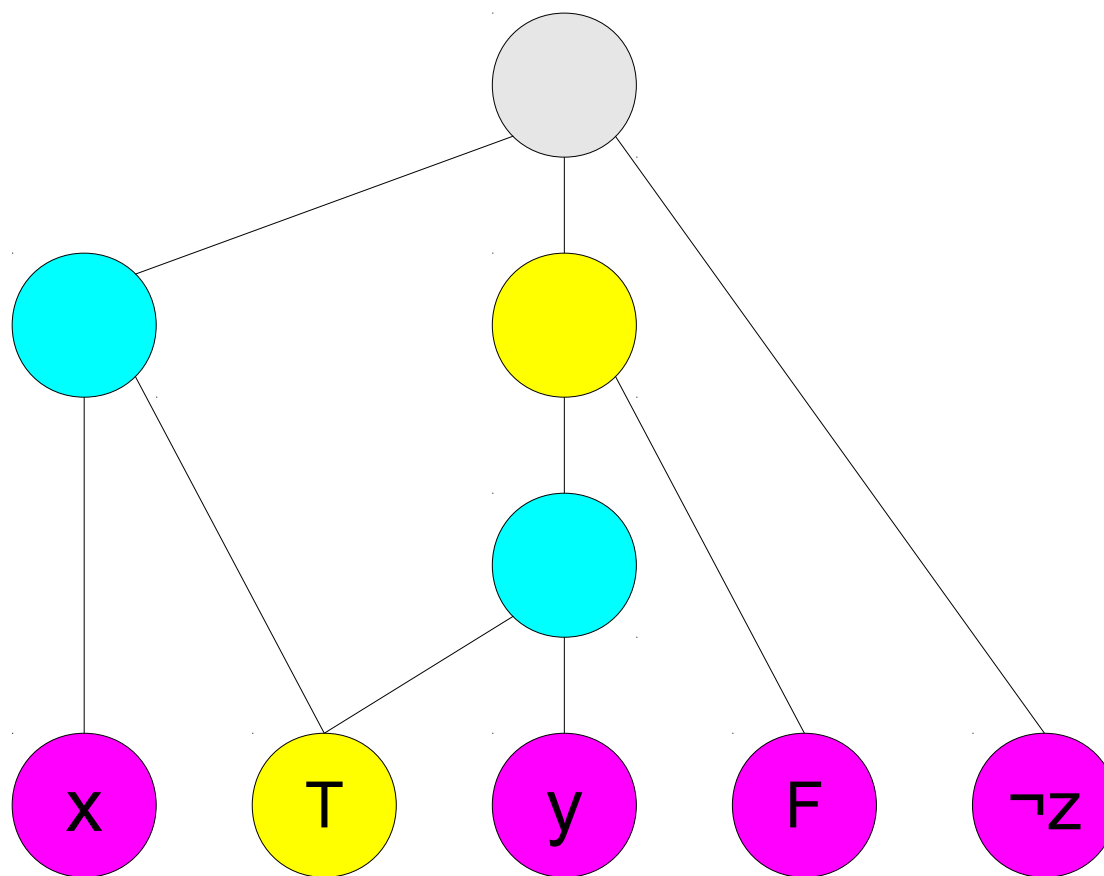
Gadget Three: Clause Satisfiability

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

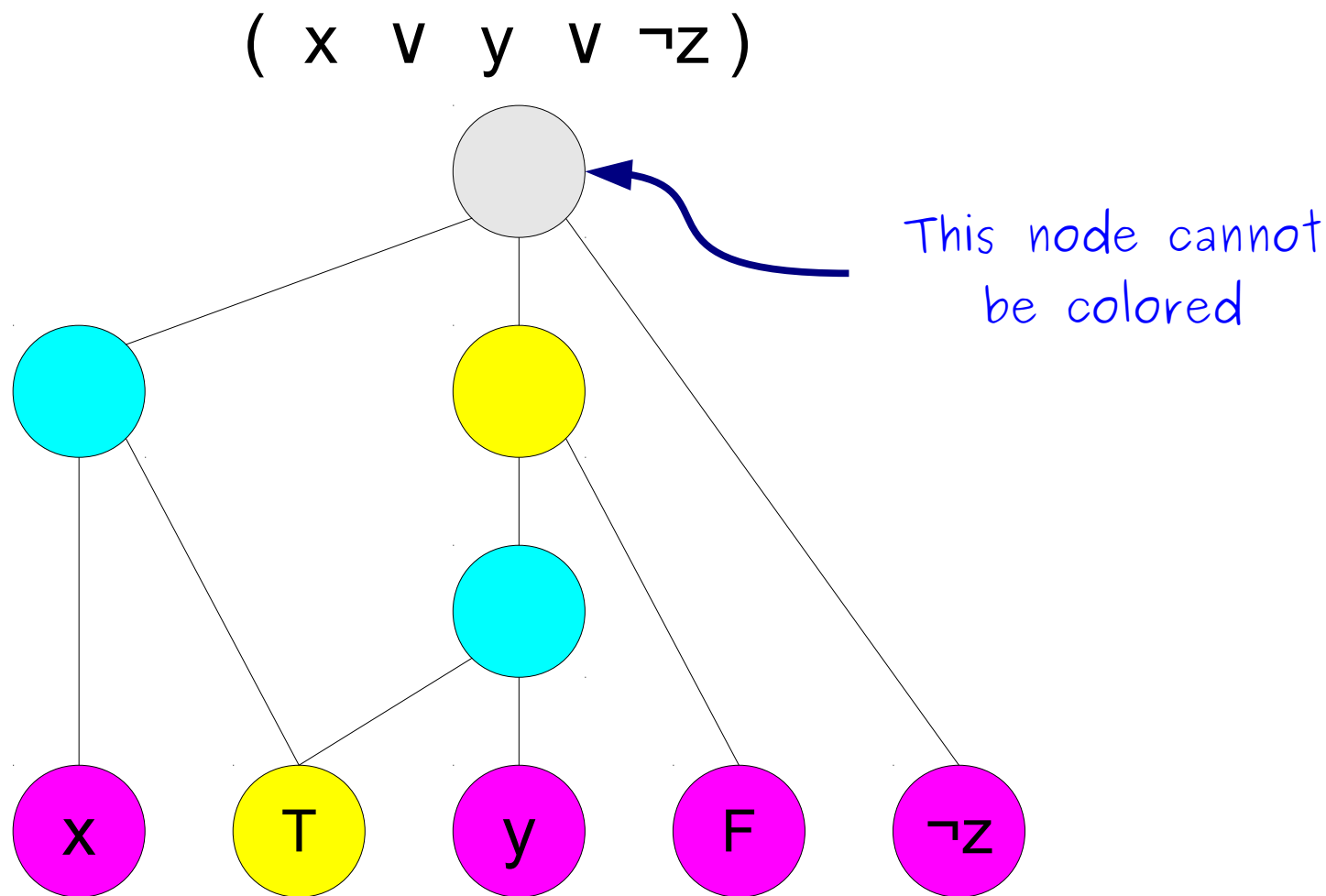


Gadget Three: Clause Satisfiability

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

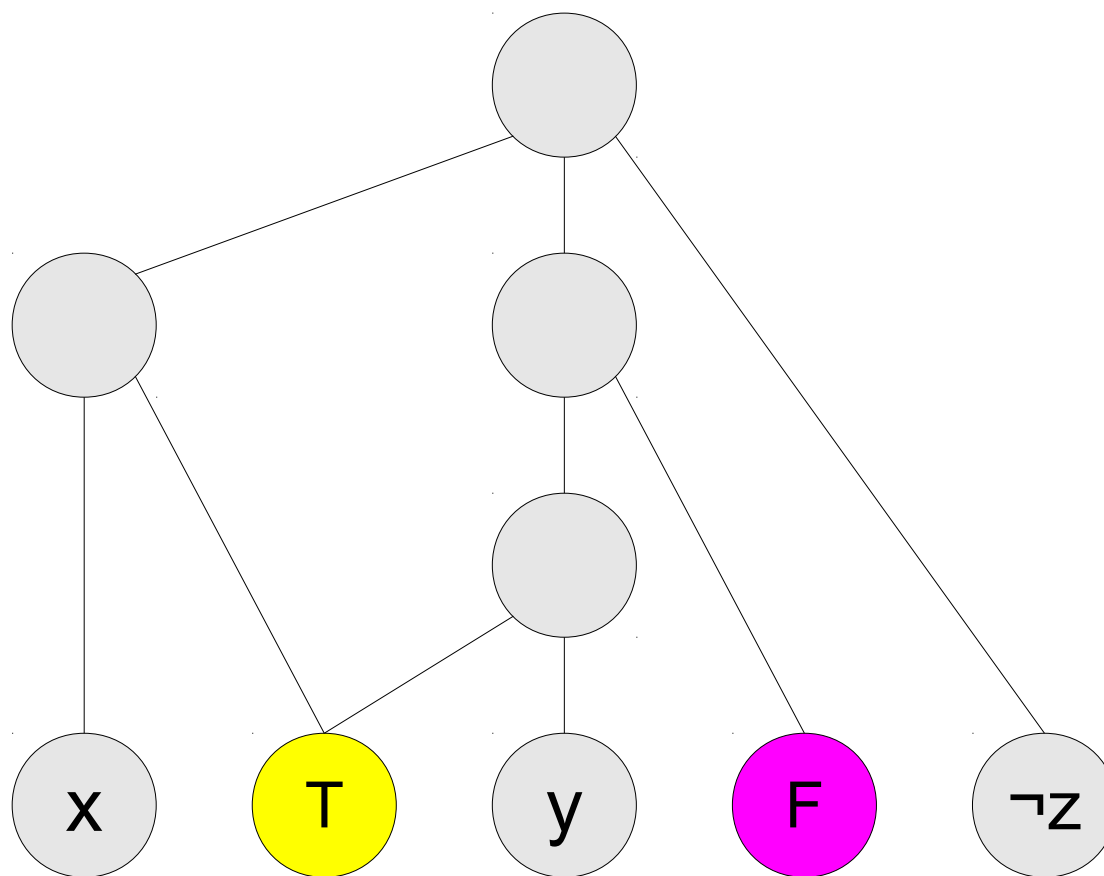


Gadget Three: Clause Satisfiability



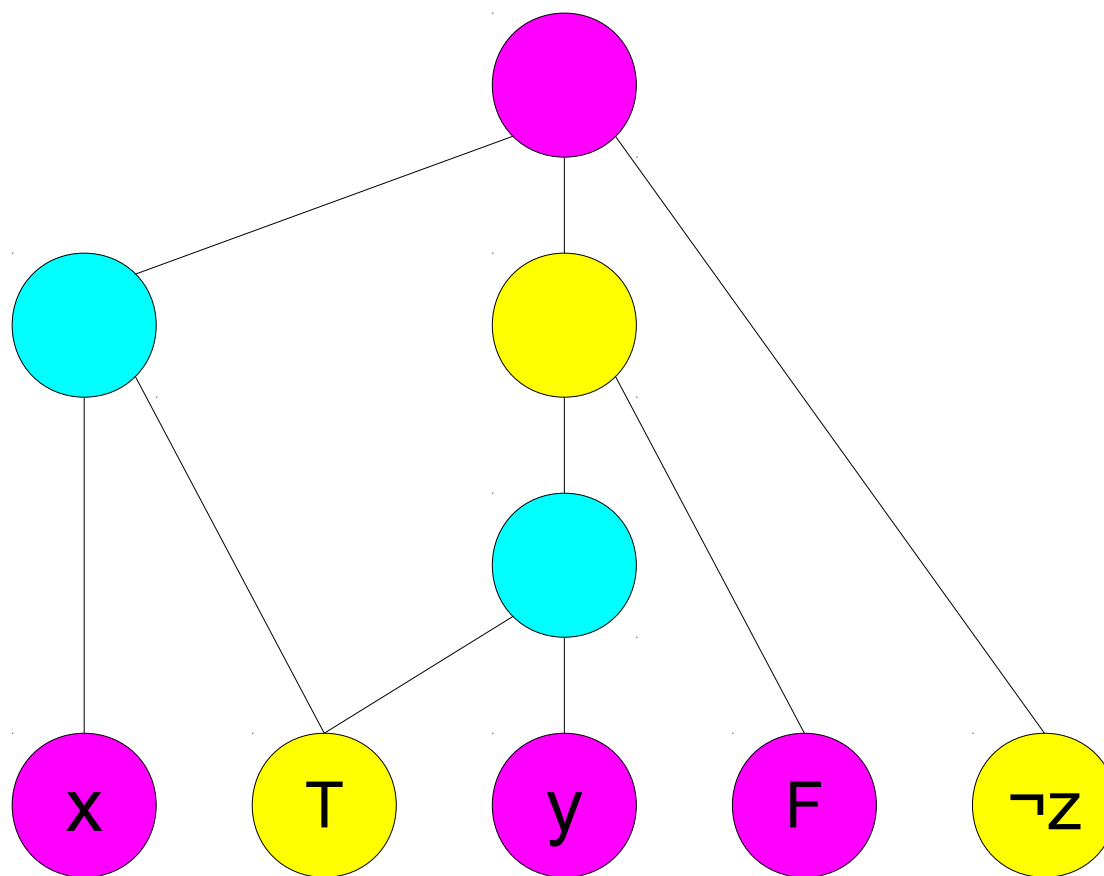
Gadget Three: Clause Satisfiability

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



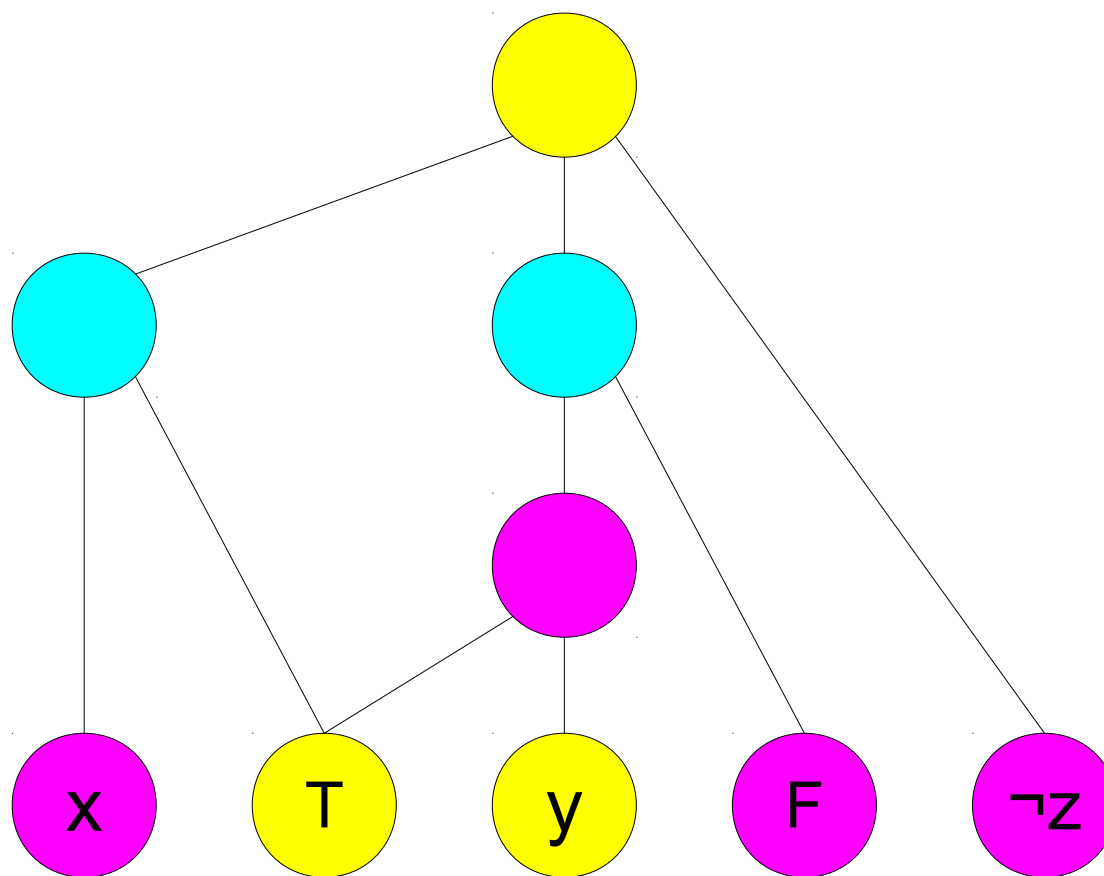
Gadget Three: Clause Satisfiability

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



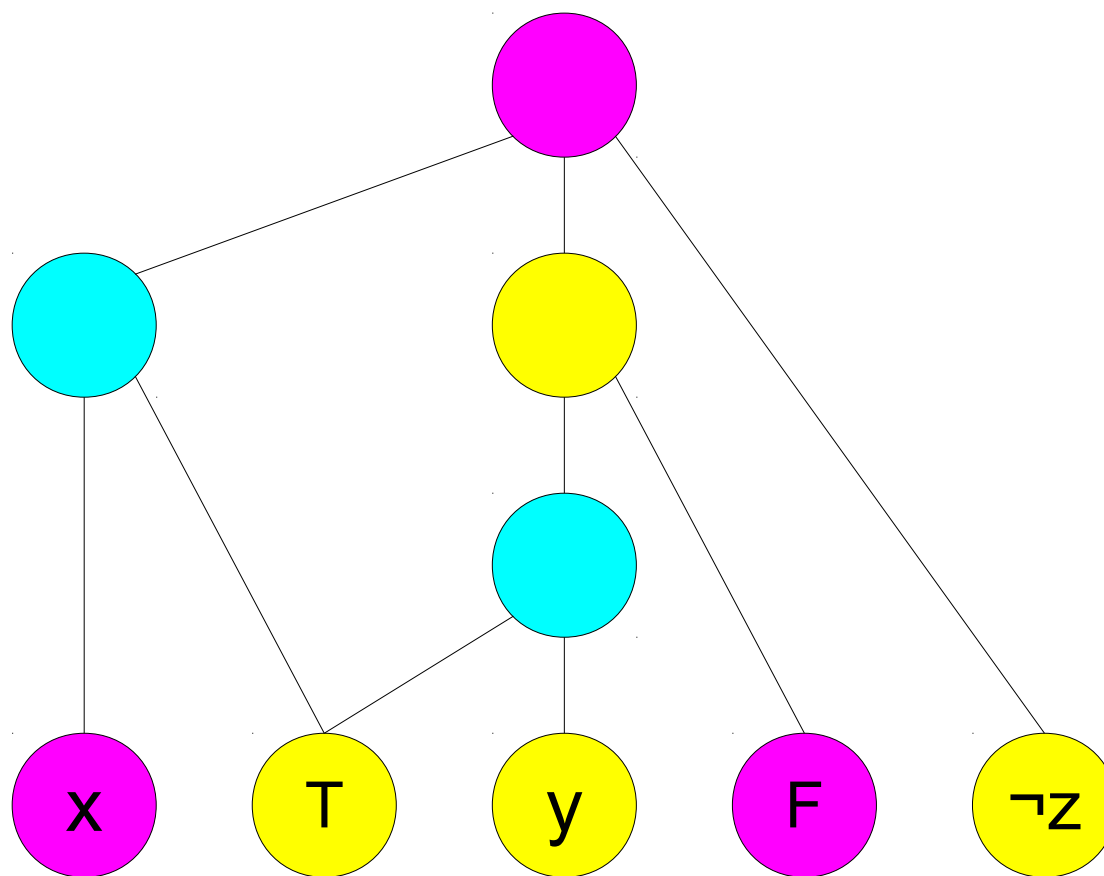
Gadget Three: Clause Satisfiability

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



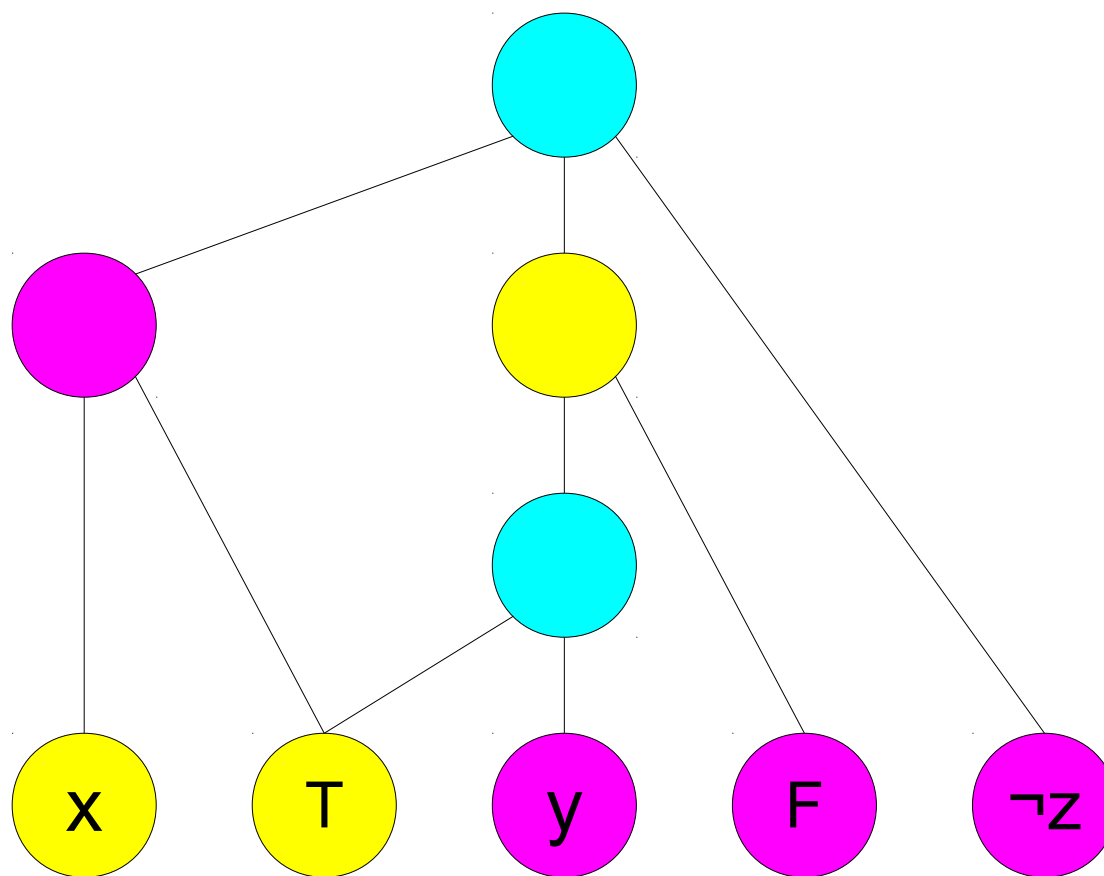
Gadget Three: Clause Satisfiability

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



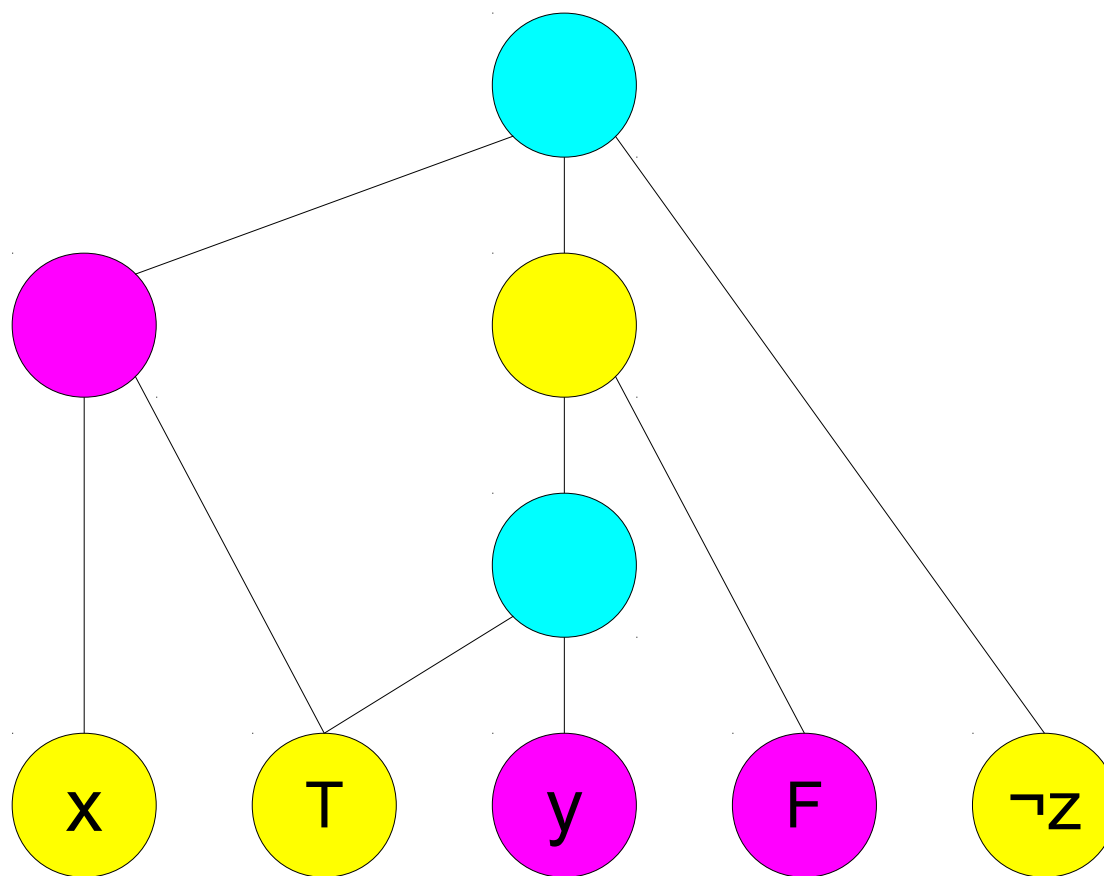
Gadget Three: Clause Satisfiability

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



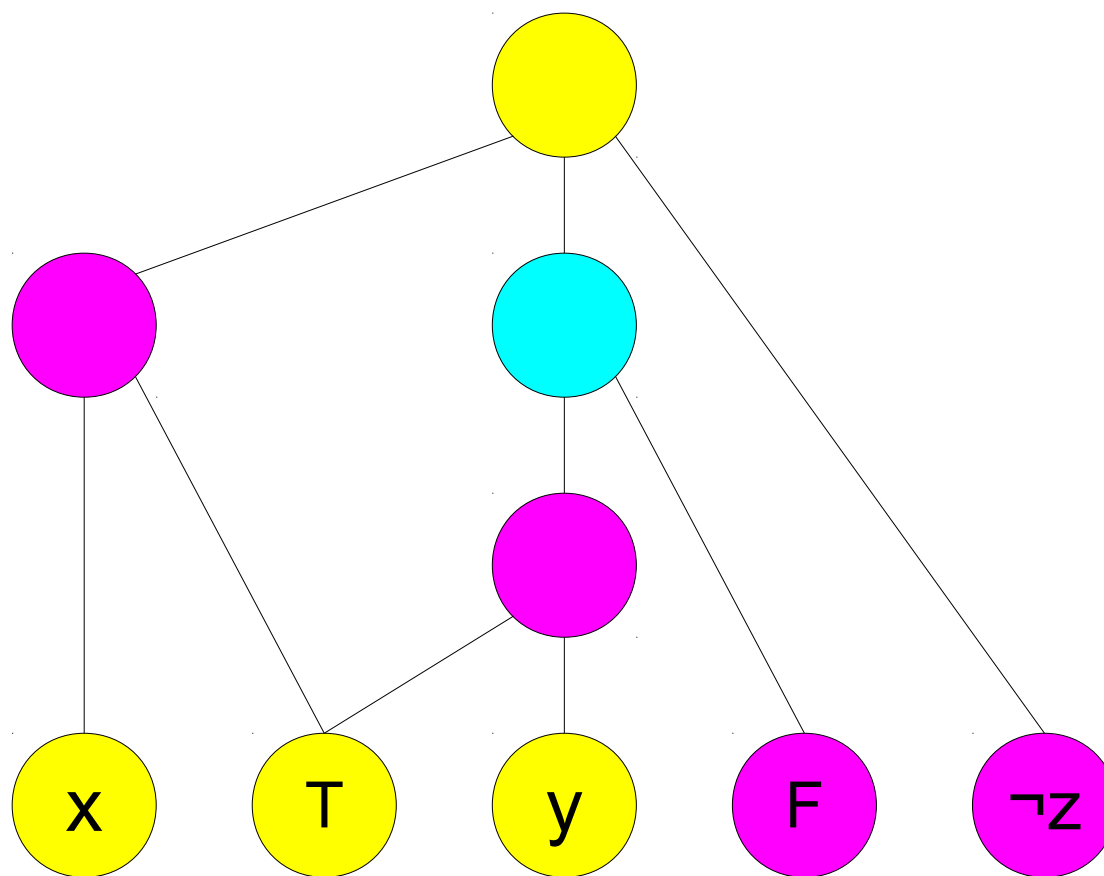
Gadget Three: Clause Satisfiability

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



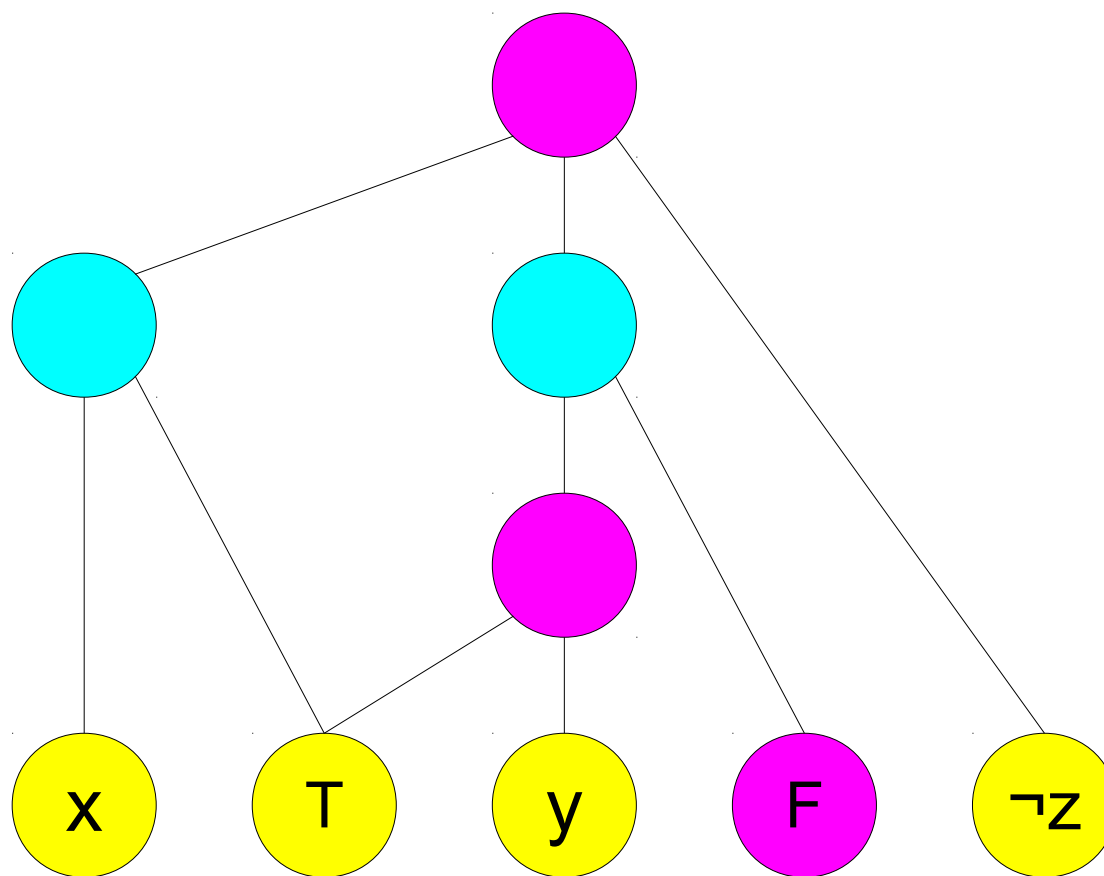
Gadget Three: Clause Satisfiability

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



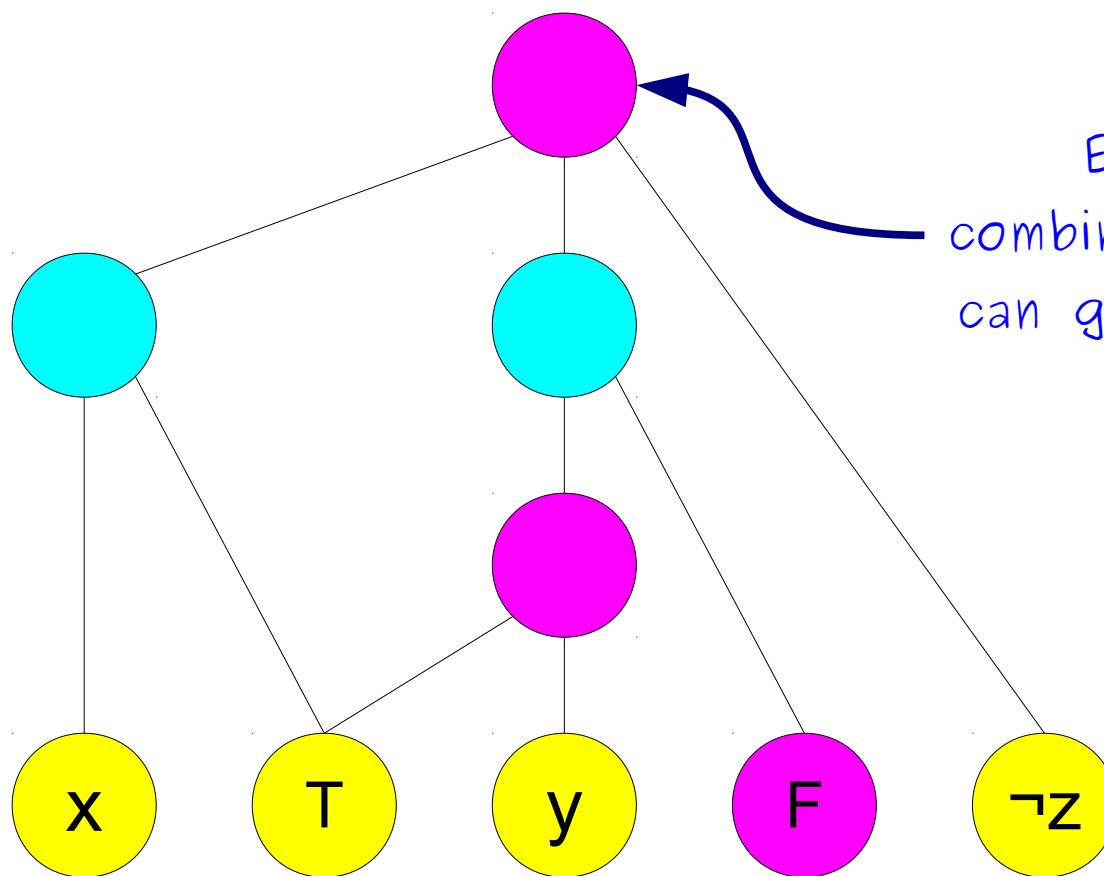
Gadget Three: Clause Satisfiability

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



Gadget Three: Clause Satisfiability

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

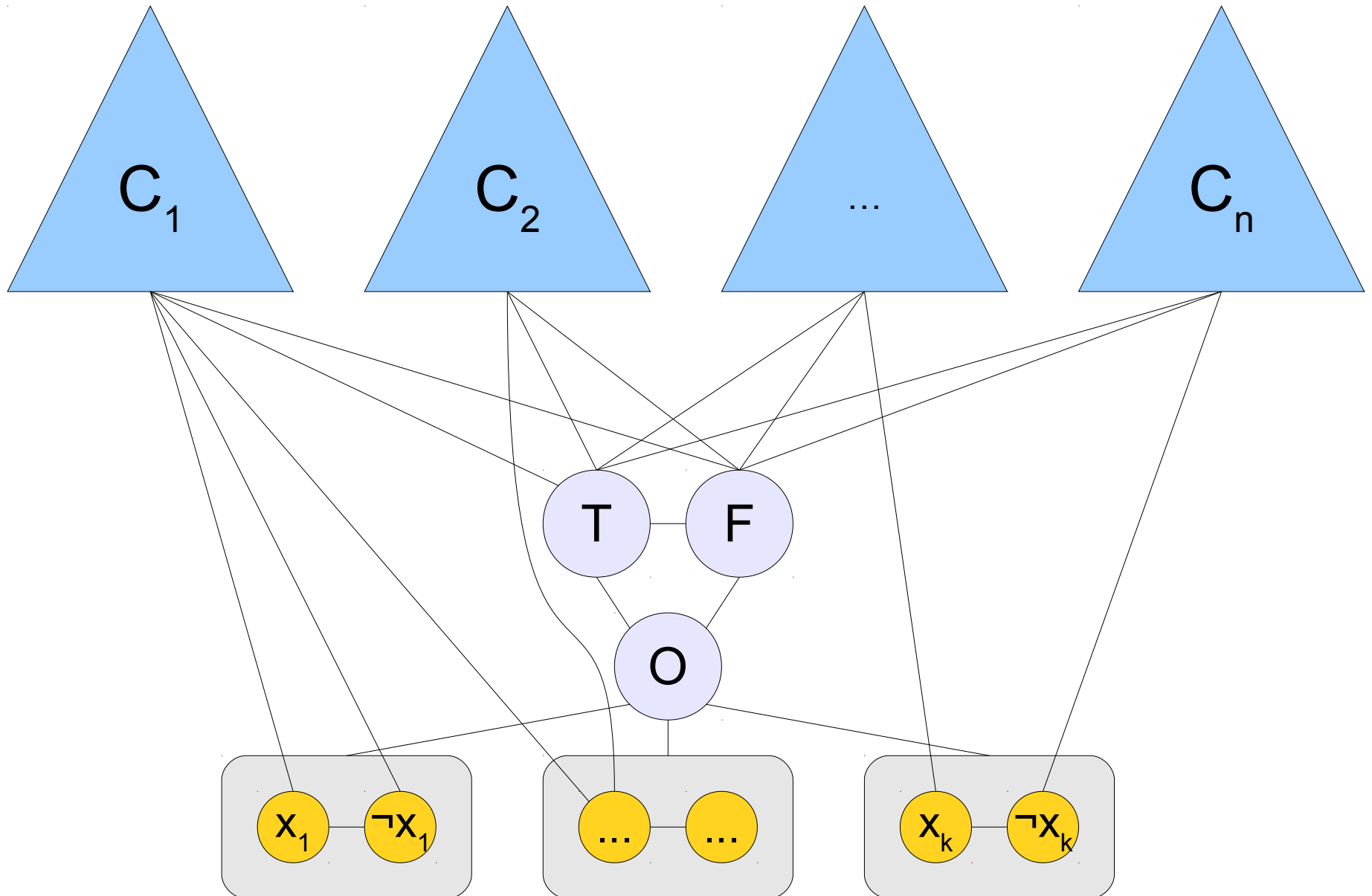


Every other combination of inputs can give this a color

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



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?