CSE 421 Winter 2025
Lecture 24:
NP-Complete

Nathan Brunelle

http://www.cs.uw.edu/421

Polynomial Time Reduction

Defn: We write $A \leq_P B$ iff there is an algorithm for A using a 'black box' (subroutine or method) that solves B that

- uses only a polynomial number of steps, and
- makes only a polynomial number of calls to a method for B.

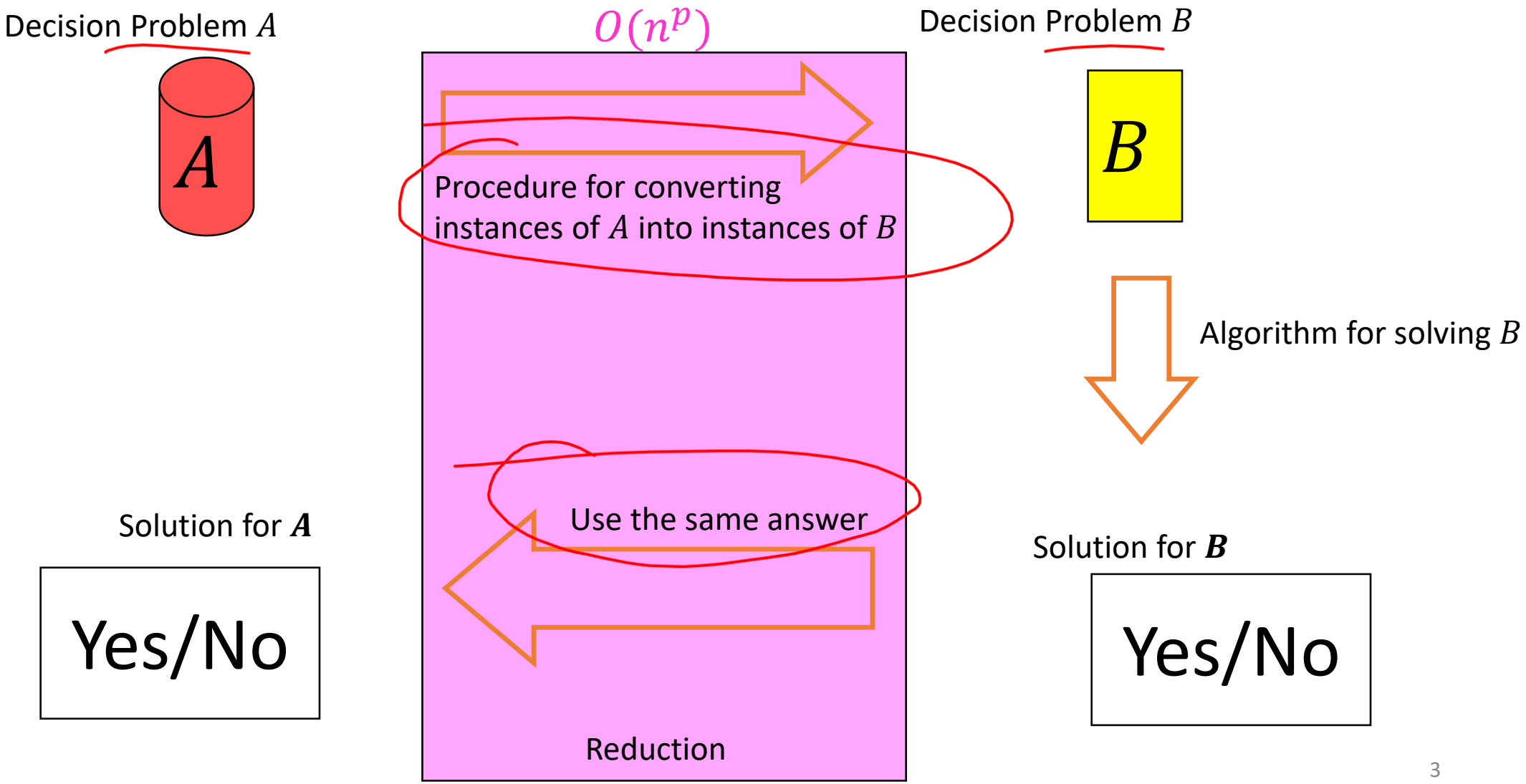
Theorem: If $A \leq_P B$ then a poly time algorithm for $B \Rightarrow$ poly time algorithm for A

Proof: Not only is the number of calls polynomial but the size of the inputs on which the calls are made is polynomial!

Corollary: If you can prove there is no fast algorithm for A, then that proves there is no fast algorithm for B.

Intuition for "$A \leq_P B$": "B is at least as hard* as A" *up to polynomial-time slop.

Polynomial Time Reductions (Decision Problems)



Let's do a reduction

4 steps for reducing (decision problem) A to problem B

1. Describe the reduction itself
   - i.e., the function that converts the input for A to the one for problem B.
   - i.e., describe what the top arrow in the pink box does
2. Make sure the running time would be polynomial
   - In lecture, we'll sometimes skip writing out this step.
3. Argue that if the correct answer (to the instance for A) is **YES**, then the input we produced is a **YES** instance for B.
4. Argue that if the input we produced is a **YES** instance for B then the correct answer (to the instance for A) is **YES**.

Contrapositive

Relationship among the problems so far

Using polynomial time reductions we have found:

- Independent-Set $\leq_P$ Clique
- Clique $\leq_P$ Independent-Set
- Vertex-Cover $\leq_P$ Independent-Set
- Independent-Set $\leq_P$ Vertex-Cover
- Clique $\leq_P$ Vertex-Cover
- Vertex-Cover $\leq_P$ Clique

- All of Independent-Set, Clique, and Vertex-Cover have polynomial-time reductions to each other.
- We do no know of any polynomial time algorithms for them, but we do know:

- If any one problem has a polynomial algorithm, ALL of them do
  - By reducing any problem to that one
- If any one problem is unsolvable in polynomial time, NONE of them are
  - By reducing that problem to the others

Extent and Impact of NP-Completeness

Extent of NP-completeness. [Papadimitriou 1995]

- 6,000 citations per year (title, abstract, keywords).
  - more than "compiler", "operating system", "database"
- Broad applicability and classification power.
- "Captures vast domains of computational, scientific, mathematical endeavors, and seems to roughly delimit what mathematicians and scientists had been aspiring to compute feasibly."

NP-completeness can guide scientific inquiry.

- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager solves 2D case in tour de force.
- 19xx: Feynman and other top minds seek 3D solution.
- 2000: Istrail proves 3D problem NP-complete.

6

Beyond $\mathbf{P}$?

Independent-Set, **Clique**, **Vertex-Cover**, and **3Color** are examples of natural and practically important problems for which we don't know any polynomial-time algorithms.

There are many others such as...

DecisionTSP:

Given a weighted graph G and an integer k,

Is there a simple path that visits all vertices in G having total weight at most k?

and...

Satisfiability

- Boolean variables $x_1, \ldots, x_n$
  - taking values in $\{0, 1\}$. 0=false, 1=true
- Literals
  - x_i or $\neg x_i$ for $i = 1, \ldots, n$. ($\neg x_i$ also written as $\overline{x_i}$.)
- Clause
  - a logical OR of one or more literals
  - e.g. $(x_1 \vee \neg x_3 \vee x_7 \vee x_{12})$
- CNF formula
  - a logical AND of a bunch of clauses
- k-CNF formula
  - All clauses have exactly k variables

Satisfiability

CNF formula example:

$$(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_4 \vee x_3) \wedge (x_2 \vee \neg x_1 \vee x_3)$$

Defn: If there is some assignment of 0's and 1's to the variables that makes it true then we say the formula is satisfiable

- $(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_4 \vee x_3) \wedge (x_2 \vee \neg x_1 \vee x_3)$ is satisfiable: $x_1 = x_3 = 1$
- $x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \neg x_3$ is not satisfiable.

3SAT: Given a CNF formula F with exactly 3 variables per clause, is F satisfiable?

Common property of these problems

- There is a special piece of information, a short certificate or proof, that allows you to **efficiently verify** (in polynomial-time) that the **YES** answer is correct. This certificate might be very hard to find.

  - **3Color**: the assignment of a color to each node.
  - **Independent-Set, Clique**: the set of vertices
  - **Vertex-Cover**: the set of vertices
  - **Decision-TSP**: the path
  - **3SAT**: a truth assignment that makes the CNF formula true.

The complexity class **NP**

NP consists of all decision problems where

- You can **verify** the **YES** answers efficiently (in polynomial time) given a short (polynomial-size) ***certificate***

and

- **No fake certificate** can fool your polynomial time verifier into saying **YES** for a **NO** instance

More precise definition of **NP**

A decision problem **A** is in **NP** iff there is

- a polynomial time procedure **VerifyA**(.,.) and
- a polynomial p

s.t.

- for every input x that is a **YES** for **A** there is a string t with $|t| \leq p(|x|)$ with **VerifyA**(x, t) = **YES**

  and

- for every input x that is a **NO** for **A** there does not exist a string t with $|t| \leq p(|x|)$ with **VerifyA**(x, t) = **YES**

- A string t on which **VerifyA**(x, t) = **YES** is called a ***certificate*** for x or a ***proof*** that x is a **YES** input

Verifying the certificate is efficient

3Color: the coloring

- Check that each vertex has one of only 3 colors and check that the endpoints of every edge have different colors

Independent-Set, **Clique**: the set U of vertices

- Check that $|U| \geq k$ and either no (**IS**) or all (**Clique**) edges on present on U

Vertex-Cover: the set W of vertices

- Check that $|W| \leq k$ and W touches every edge.

Decision-TSP: the path

- Check that the path touches each vertex and has total weight $\leq k$.

- **3-SAT**: a truth assignment α that makes the CNF formula F true.
  - Evaluate F on the truth assignment α.

Keys to showing that a problem is in **NP**

1. Must be decision probem (**YES**/**NO**)
2. For every given **YES** input, is there a certificate (i.e., a hint) that would help?
   - OK if some inputs don't need a certificate
3. For any given **NO** input, is there a fake certificate that would trick you?
4. You need a polynomial-time algorithm to be able to tell the difference.

Solving **NP** problems without hints

There is an obvious algorithm for all **NP** problems:

Brute force:

Try all possible certificates and check each one using the verifier to see if it works.

Even though the certificates are short, this is exponential time

- 2^n truth assignments for n variables
- $\binom{n}{k}$ possible k-element subsets of n vertices
- $n!$ possible TSP tours of n vertices
- etc.

What We Know

- Every problem in $\mathbf{NP}$ is in exponential time
- Every problem in $\mathbf{P}$ is in $\mathbf{NP}$
  - You don't need a certificate for problems in $\mathbf{P}$ so just ignore any hint you are given
- Nobody knows if all problems in $\mathbf{NP}$ can be solved in polynomial time; i.e., does $\mathbf{P = NP?}$
  - one of the most important open questions in all of science.
  - huge practical implications
- Most CS researchers believe that $\mathbf{P \neq NP}$
  - $1M prize either way
  - but we don't have good ideas for how to prove this ...

NP-hardness & **NP**-completeness

Notion of hardness we **can** prove that is useful unless $\mathbf{P} = \mathbf{NP}$:

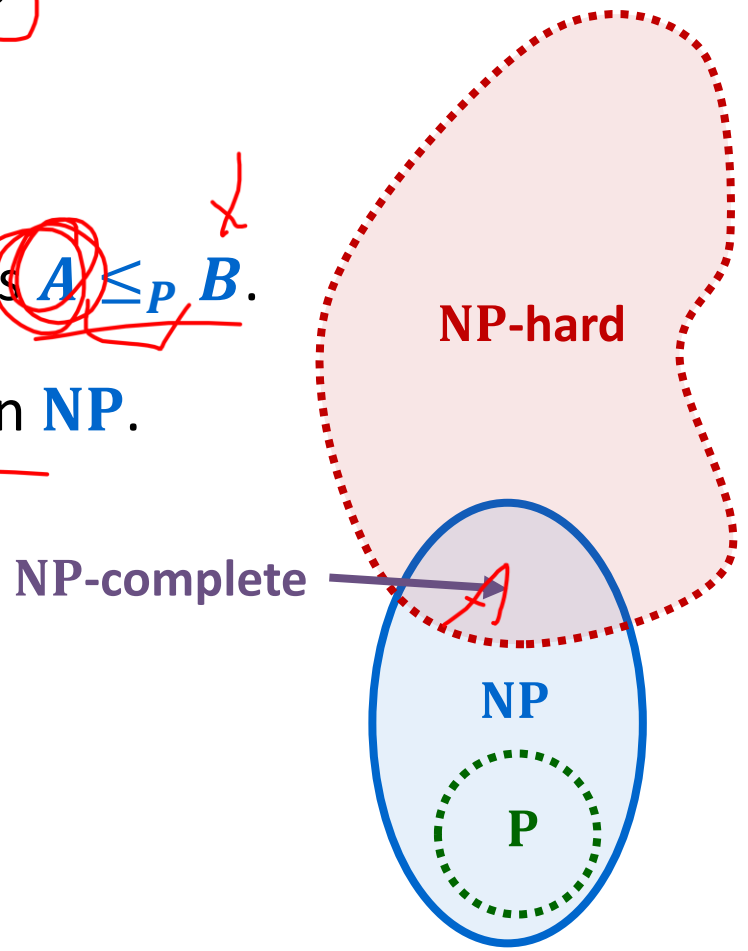
Defn: Problem B is **NP**-hard iff **every** problem $A \in \mathbf{NP}$ satisfies $A \leq_P B$.

This means that B is at least as hard as every problem in **NP**.

Defn: Problem B is **NP**-complete iff

- $B \in \mathbf{NP}$ and
- B is **NP**-hard.

This means that B is a hardest problem in **NP**.



Not at all obvious that any **NP**-complete problems exist!

Cook-Levin Theorem

Theorem [Cook 1971, Levin 1973]: **3SAT** is **NP**-complete

Proof: See CSE 431.

Corollary: If **3SAT** $\leq_P$ **B** then **B** is **NP**-hard.

Proof: Let **A** be an arbitrary problem in **NP**.
Since **3SAT** is **NP**-hard we have **A** $\leq_P$ **3SAT.**

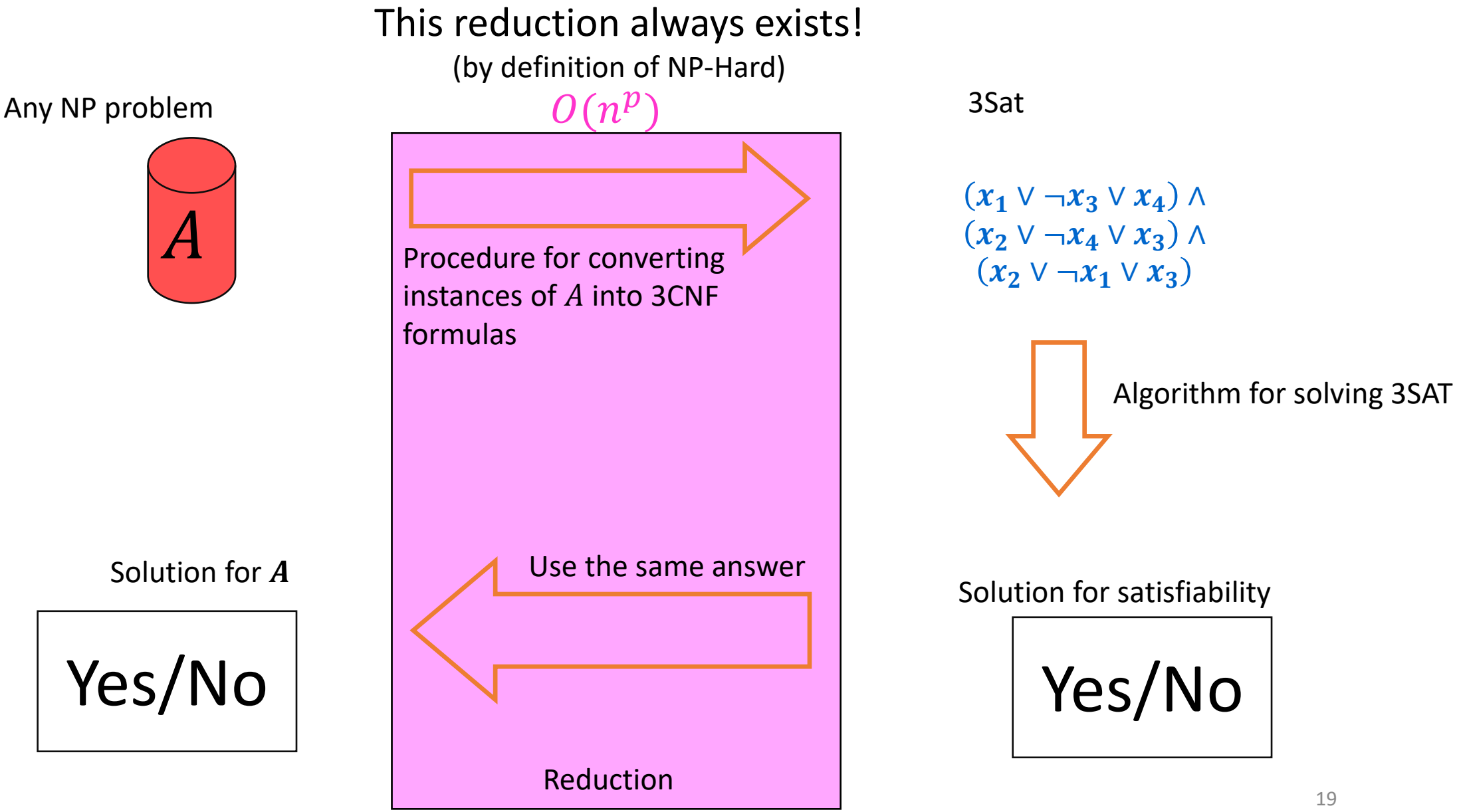
Then **A** $\leq_P$ **3SAT** and **3SAT** $\leq_P$ **B** imply that **A** $\leq_P$ **B**.

Therefore every problem **A** in **NP** has **A** $\leq_P$ **B**
so **B** is **NP**-hard.

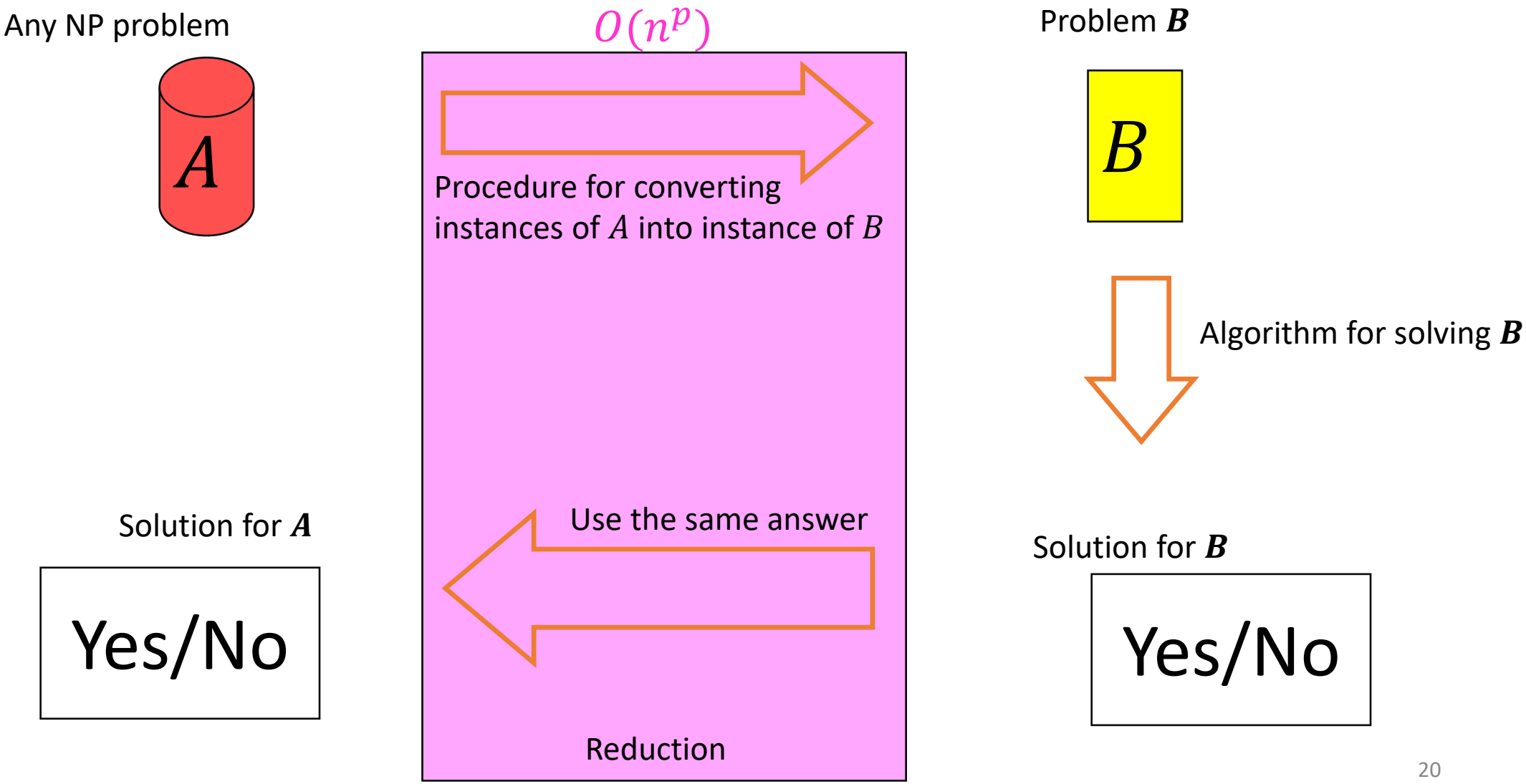
Cook & Levin did the hard work.

We only need to give one reduction to show that a problem is NP-hard!

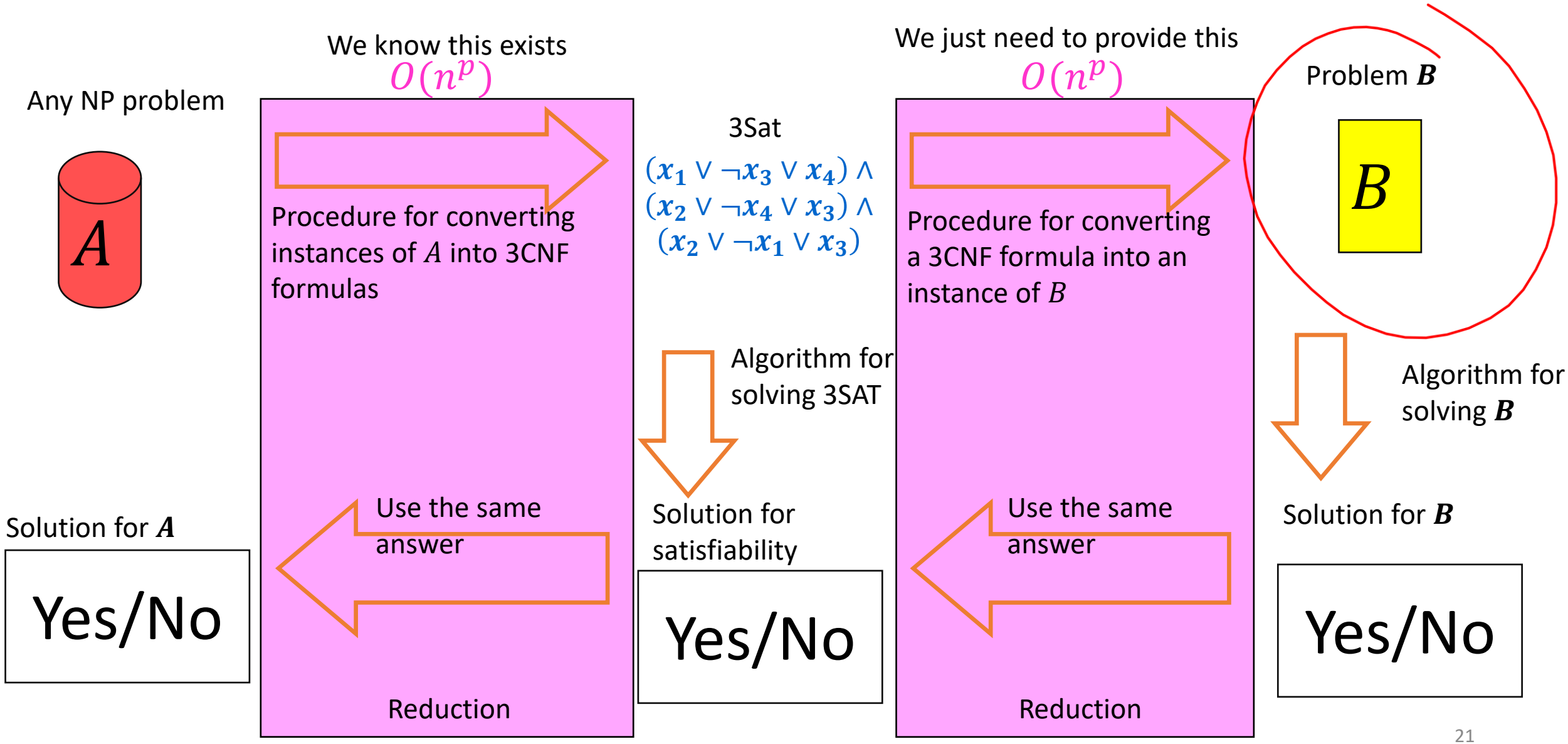
What we know: 3Sat is NP-Hard



Goal: B is NP-Hard



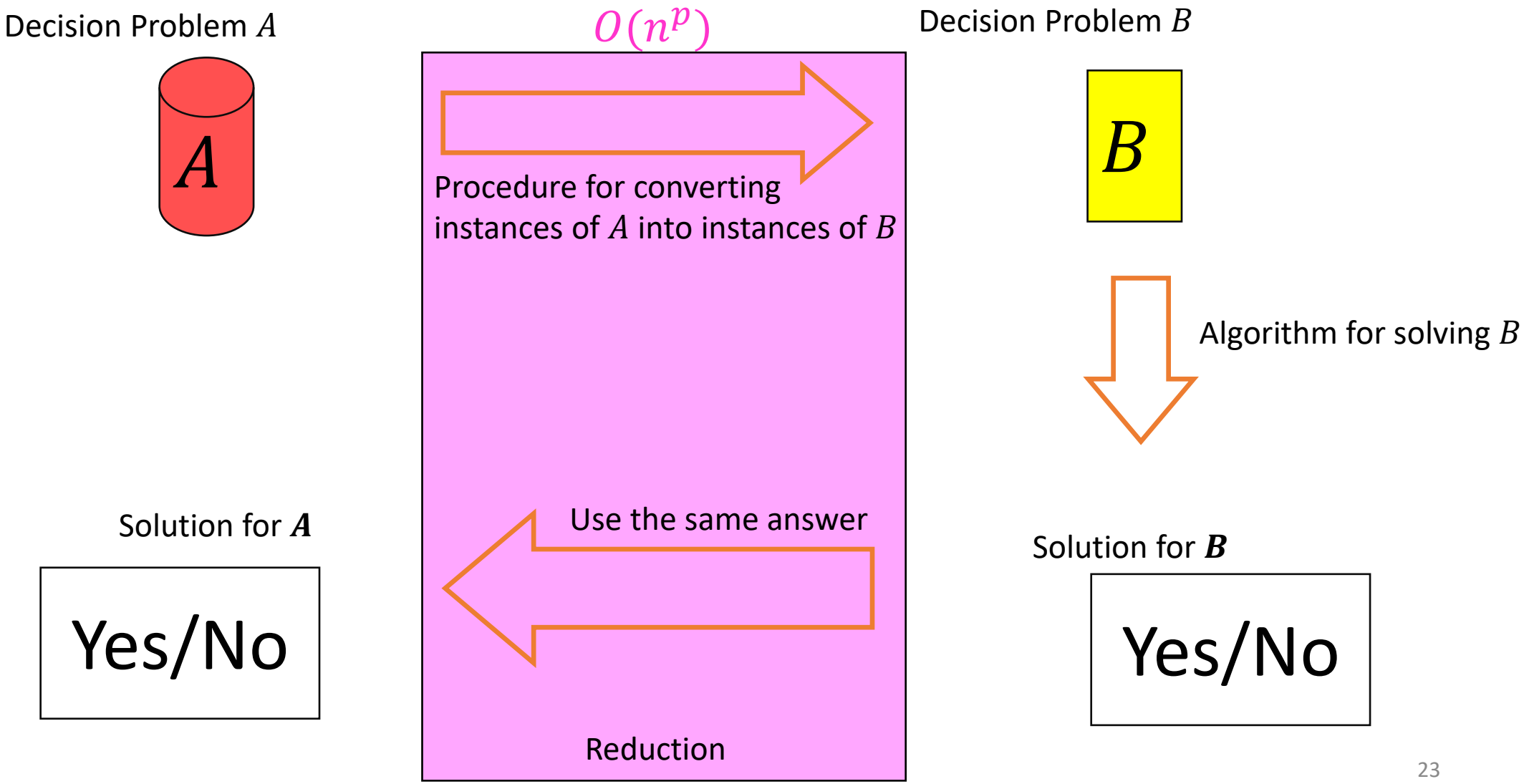
Showing B is NP-Hard



Steps to Proving Problem B is NP-complete

- Show B is in NP
  - State what the hint/certificate is.
  - Argue that it is polynomial-time to check and you won't get fooled.
- Show B is NP-hard:
  - State: "Reduction is from NP-hard Problem A"
  - Show what the reduction function f is.
  - Argue that f is polynomial time.
  - Argue correctness in two directions:
    - x a **YES** for A implies $f(x)$ is a **YES** for B
      - Do this by showing how to convert a certificate for x being **YES** for A to a certificate for $f(x)$ being a **YES** for B.
    - $f(x)$ a **YES** for B implies x is a **YES** for A
      - … by converting certificates for $f(x)$ to certificates for x

Next up: Let's show Independent Set is NP-Hard



Showing Independent Set is NP-Hard

3Sat

$(x_1 \vee \neg x_3 \vee x_4) \wedge$
$(x_2 \vee \neg x_4 \vee x_3) \wedge$
$(x_2 \vee \neg x_1 \vee x_3)$

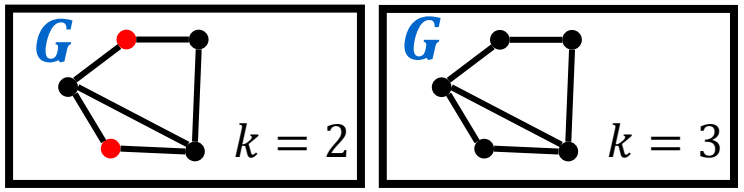
$O(n^p)$

Covert a 3CNF formula F into a graph G and a number k such that G has an independent set of size k if and only if F has a satisfying assignment

Use the same answer

Reduction

Independent Set



Algorithm for solving Independent Set

Solution for the instance of Independent Set

Yes/No

Solution for the instance of 3Sat

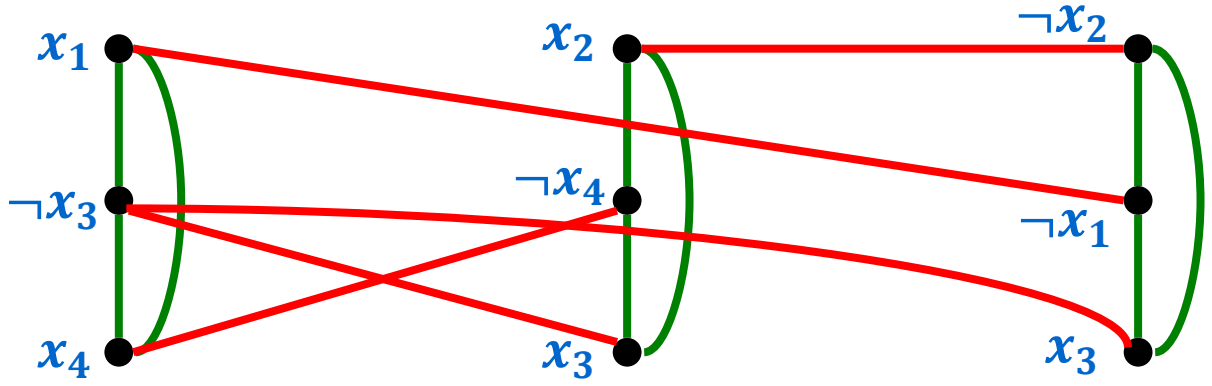
Yes/No

Another **NP**-complete problem: 3SAT $\leq_P$ Independent-Set

1. The reduction:
   - Map CNF formula F to a graph G and integer k
   - Let m = # of clauses of F
   - Create a vertex in G for each literal occurrence in F
     - $3m$ total vertices
   - Join two vertices u, v in G by an edge iff
     - u and v correspond to literals in the same clause of F or
     - u and v correspond to literals x and $\neg x$ (or vice versa) for some variable x (i.e. they contradict).
   - Set $k = m$
2. Clearly polynomial-time computable

Another **NP**-complete problem: 3SAT $\leq_P$ Independent-Set

$\boldsymbol{F = (x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_4 \vee x_3) \wedge (\neg x_2 \vee \neg x_1 \vee x_3)}$



$\boldsymbol{m = 3}$

$\boldsymbol{G}$ has both kinds of edges.
The color is just to show why the edges were included.

$\boldsymbol{k = m}$

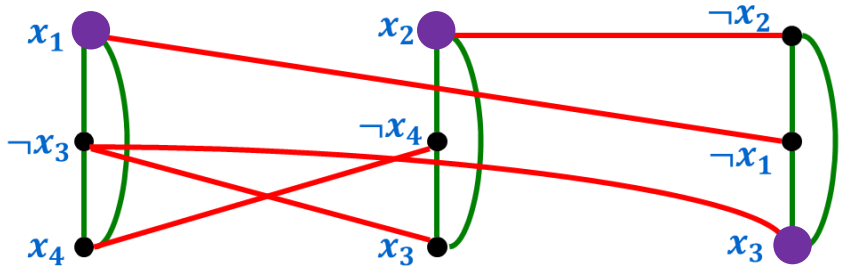
Correctness (⇒)

Suppose that F is satisfiable (**YES** for **3SAT**)

- Let α be a satisfying assignment; it satisfies at least one literal in each clause.
- Choose the set U in G to correspond to the **first satisfied literal in each clause**.
  - $|U| = m$
  - Since U has 1 vertex per clause, no green edges inside U.
  - A truth assignment never satisfies both x and $\neg x$, so no red edges inside U.
  - Therefore U is an independent set of size m

Therefore (G, m) is a **YES** for **Independent-Set**.

$$F = (x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_4 \vee x_3) \wedge (\neg x_2 \vee \neg x_1 \vee x_3)$$



Satisfying assignment α:

$$\alpha(x_1) = \alpha(x_2) = \alpha(x_3) = \alpha(x_4) = 1$$

Set U marked in purple is independent.

Correctness (⇐)

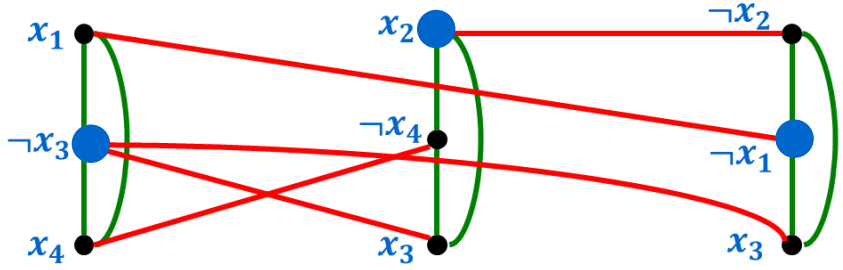
Suppose that G has an independent set of size m

((G, m) is a **YES** for **Independent-Set**)

- Let U be the independent set of size m;
- U must have one vertex per column (green edges)
- Because of red edges, U doesn't have vertex labels with conflicting literals.
- Set all literals labelling vertices in U to true
- This may not be a total assignment but just extend arbitrarily to a total assignment α.
  - This assignment satisfies F since it makes at least one literal per clause true.

Therefore F is satisfiable and a **YES** for **3SAT**.

$$F = (x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_4 \vee x_3) \wedge (\neg x_2 \vee \neg x_1 \vee x_3)$$



Given independent set U of size m

Satisfying assignment α: Part defined by U:

$\alpha(x_1) = 0, \alpha(x_2) = 1, \alpha(x_3) = 0$

Set $\alpha(x_4) = 0$.

Showing Independent Set is NP-Hard

3Sat

$(x_1 \vee \neg x_3 \vee x_4) \wedge$
$(x_2 \vee \neg x_4 \vee x_3) \wedge$
$(x_2 \vee \neg x_1 \vee x_3)$

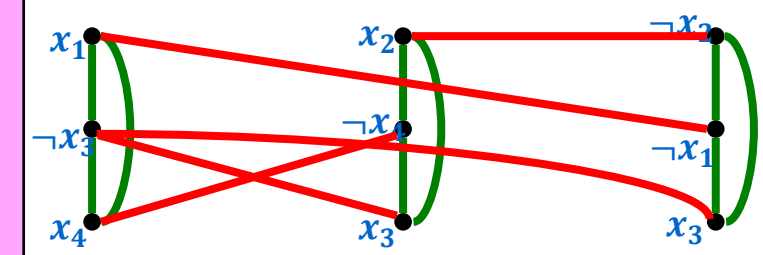
$O(n^p)$

Make one node per literal, connect each to other nodes in the same clause, connect literals with their negations, set k to be the number of clauses

Use the same answer

Reduction

Independent Set



$k = 3$

Algorithm for solving Independent Set

Solution for the instance of 3Sat

Yes/No

Solution for the instance of Independent Set

Yes/No

Many **NP**-complete problems

Since **3SAT** $\leq_P$ **Independent-Set**, **Independent-Set** is **NP**-hard.

We already showed that **Independent-Set** is in **NP**.

⇒ **Independent-Set** is **NP**-complete

Corollary: **Clique** and **Vertex-Cover** are also **NP**-complete.

Proof: We already showed that all are in **NP**.

We also showed that **Independent-Set** polytime reduces to all of them.

Combining this with **3SAT** $\leq_P$ **Independent-Set** we get that all are **NP**-hard.

NP-complete problems so far

So far:

3SAT → **Independent-Set** → **Clique**

↓

Vertex-Cover

Recall: Graph Colorability

Defn: A undirected graph $G = (V, E)$ is k-colorable iff
we can assign one of k colors to each vertex of V s.t.
for every edge (u, v) has different colored endpoints, $\chi(u) \neq \chi(v)$.
"edges are not monochromatic"

Theorem: 3Color is **NP**-complete

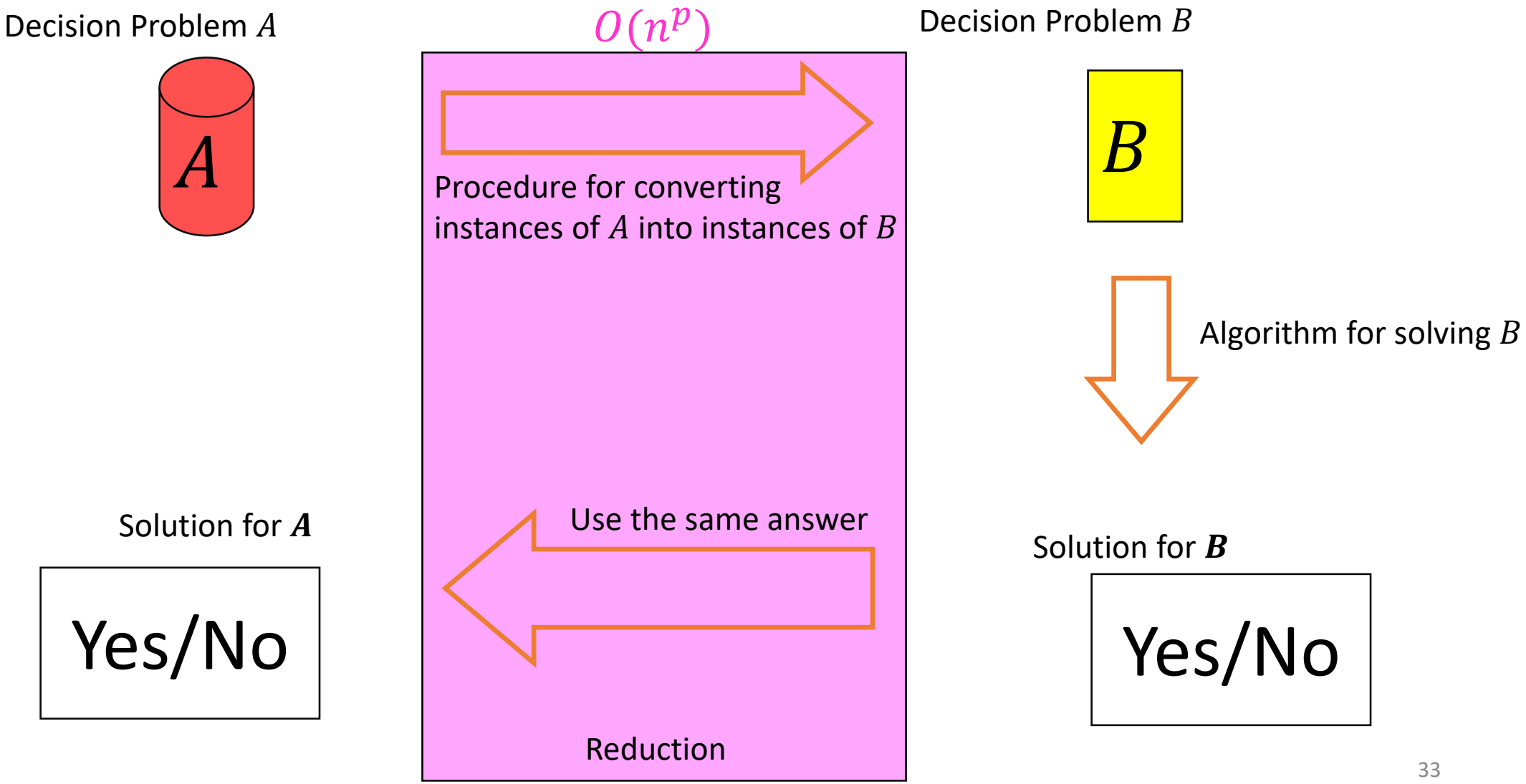
Proof:

1. **3Color** is in **NP**:
   - We already showed this; the certificate was the coloring.
2. **3Color** is **NP**-hard:

   Claim: 3SAT$\leq_P$3Color

   We need to find a function f that maps a 3CNF formula F to a graph G s.t.
   F is satisfiable $\Leftrightarrow$ G is 3-colorable.

Next up: Let's show 3Color is NP-Hard



Showing 3Color is NP-Hard

3Sat

$(x_1 \vee \neg x_3 \vee x_4) \wedge$
$(x_2 \vee \neg x_4 \vee x_3) \wedge$
$(x_2 \vee \neg x_1 \vee x_3)$

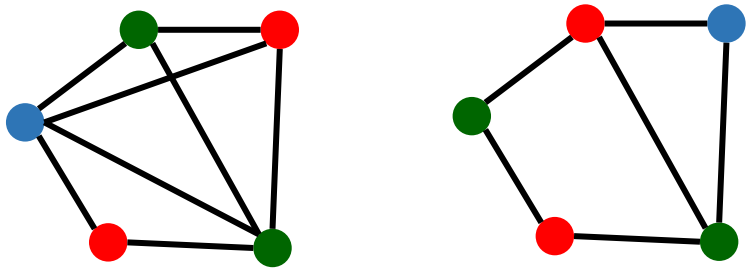
$O(n^p)$

Covert a 3CNF formula F into a graph G such that G is 3 colorable if and only if F has a satisfying assignment

Use the same answer

Reduction

3Color



Algorithm for solving Independent Set

Solution for the instance of Independent Set

Yes/No

Solution for the instance of 3Sat

Yes/No

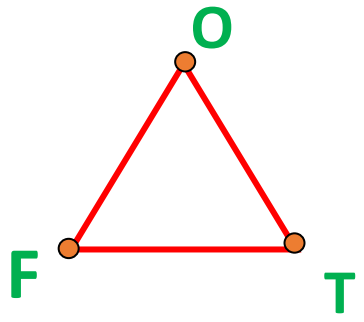
3SAT $\leq_P$ 3Color

Start with a base triangle with vertices **T**, **F**, and **O**.

We can assume that **T**, **F**, and **O** are the three colors used.

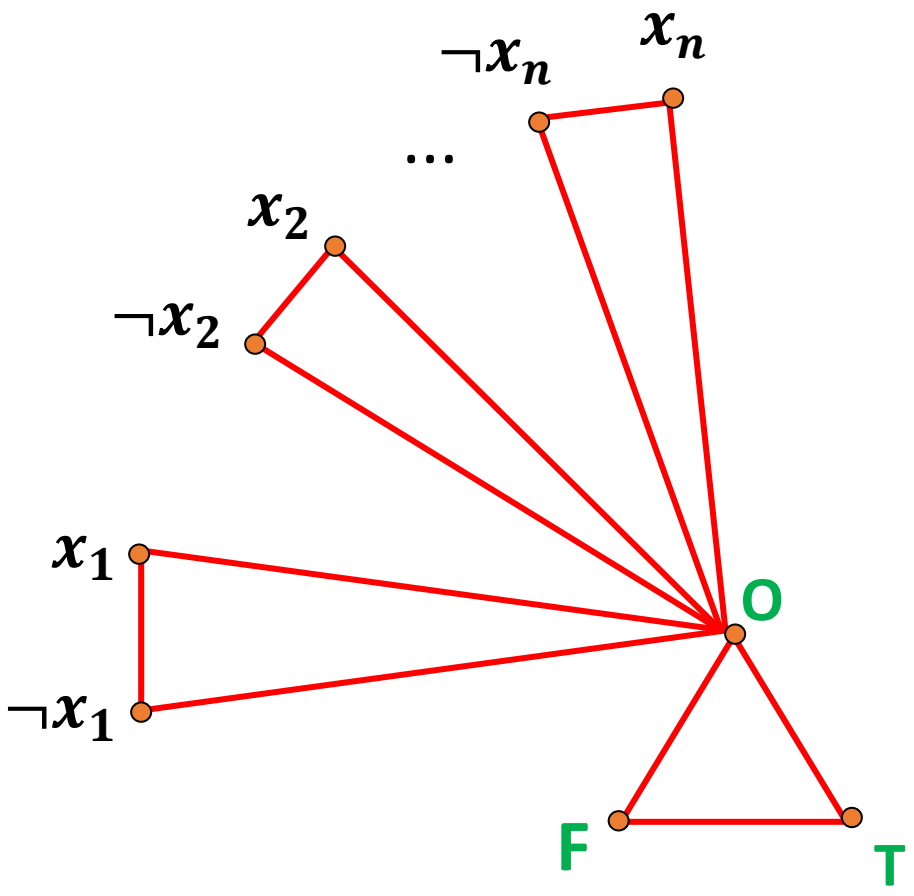
- Intuition: **T** and **F** will stand for *true* and *false*; **O** will stand for *other*.

To represent the properties of the 3CNF formula F we will need both a Boolean variable part and a clause part.



Base Triangle

3SAT $\leq_P$ 3Color



Base Triangle

Boolean variable part:

- For each Boolean variable add a triangle with two nodes labelled by literals as shown.
- Since both nodes are joined to node **O** and to each other, they must have opposite colors **T** and **F** in any 3-coloring.
- So, any 3-coloring corresponds to a unique truth assignment.

3SAT $\leq_P$ 3Color

Idea:

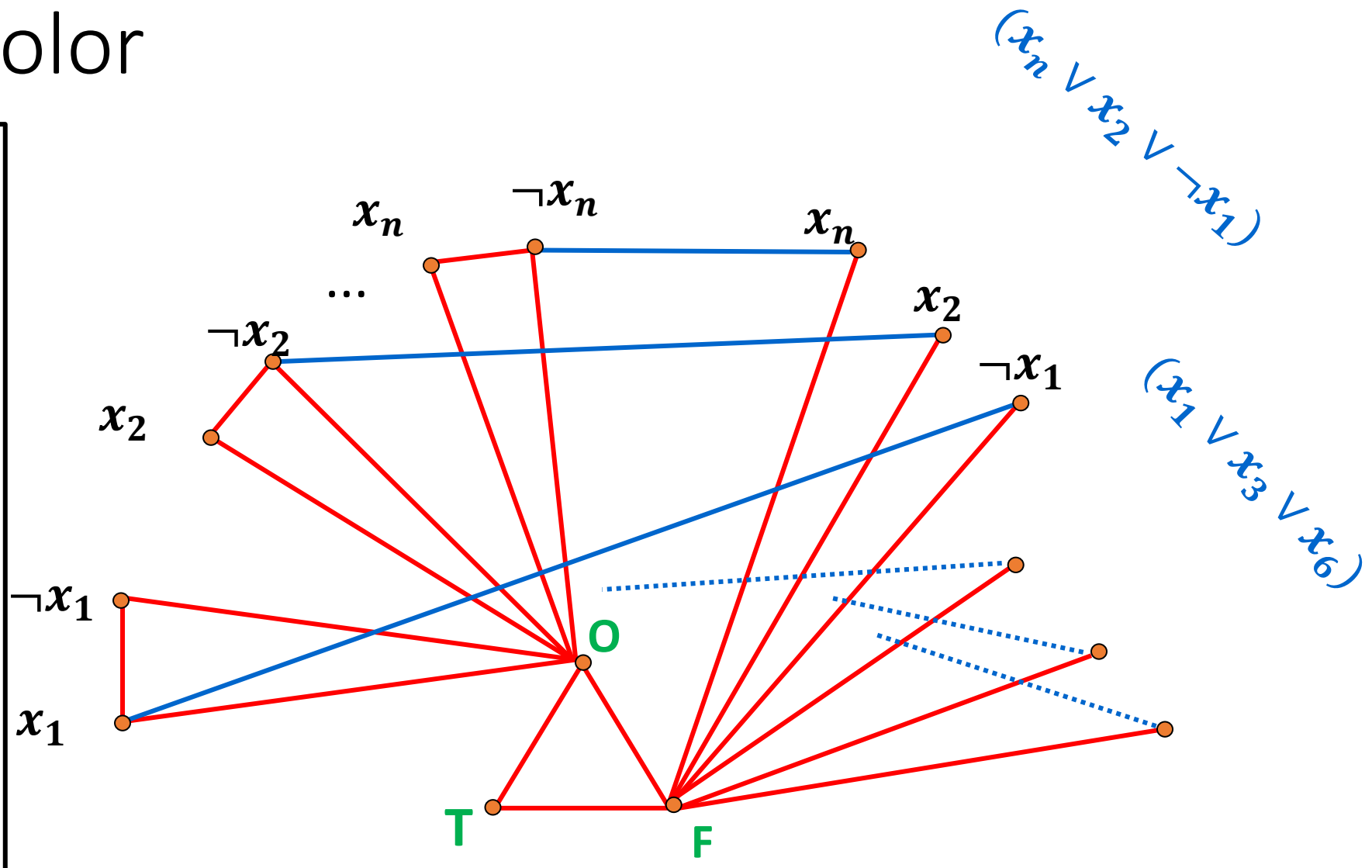
Create a "middle" node per literal for each clause, we will consider a T-colored middle node to satisfy a clause.

In the graph:

For each clause of F add 3 "middle" nodes. Then:

- Join each middle node to it opposite literal node
- Join each middle node to F

Now each middle node must be either T or O, and any connect to something T-colored must be O-colored



3SAT $\leq_P$ 3Color

Idea:

Force at least one middle node per clause to be T-colored.

In the graph:

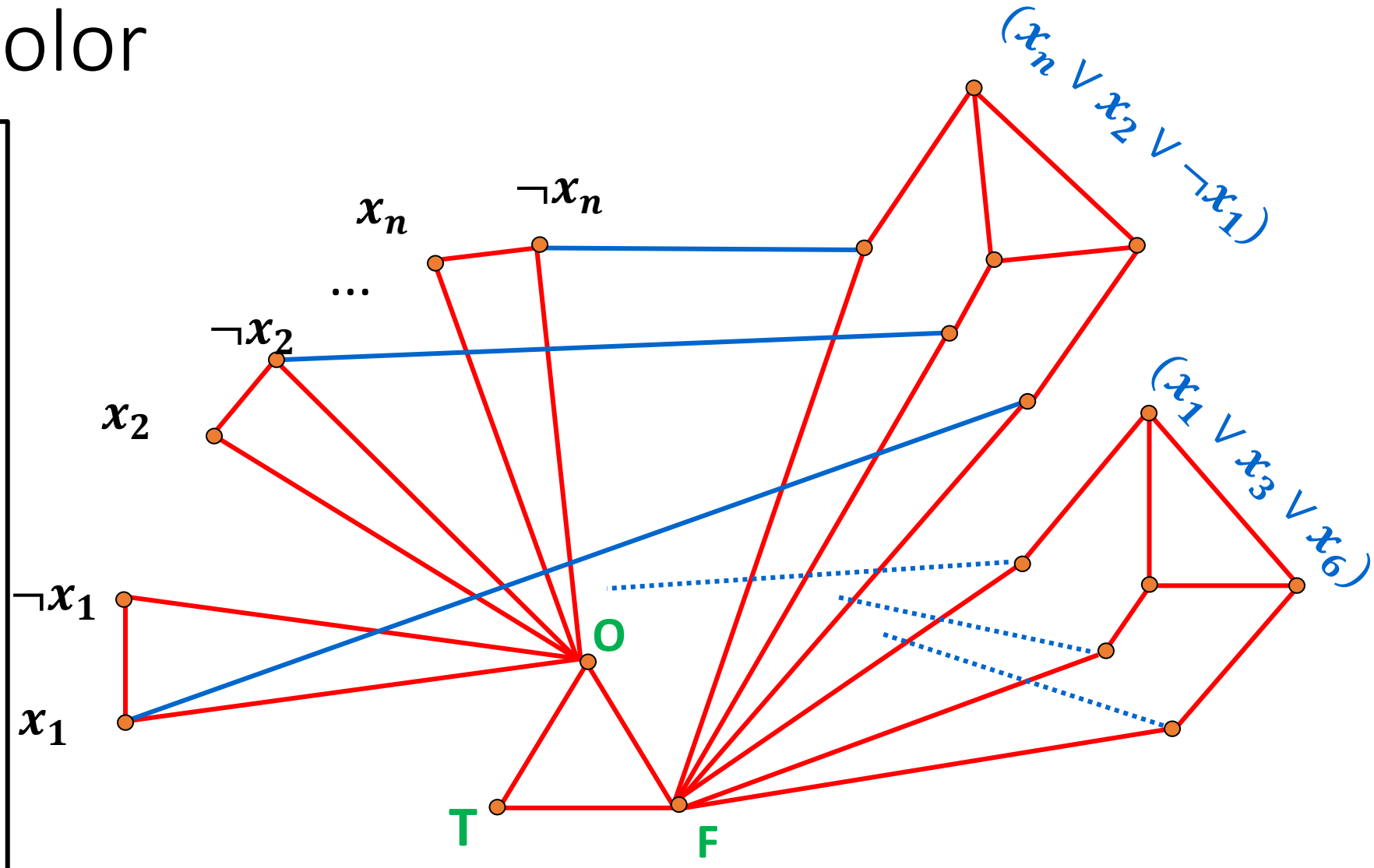
For each clause of F add an outer triangle.

- Join each middle node a vertex in the triangle

No middle node can be F-colored (all connect to F)

Not all middle nodes are O-colored (because something in the outer triangle must be)

So at least one is T-colored



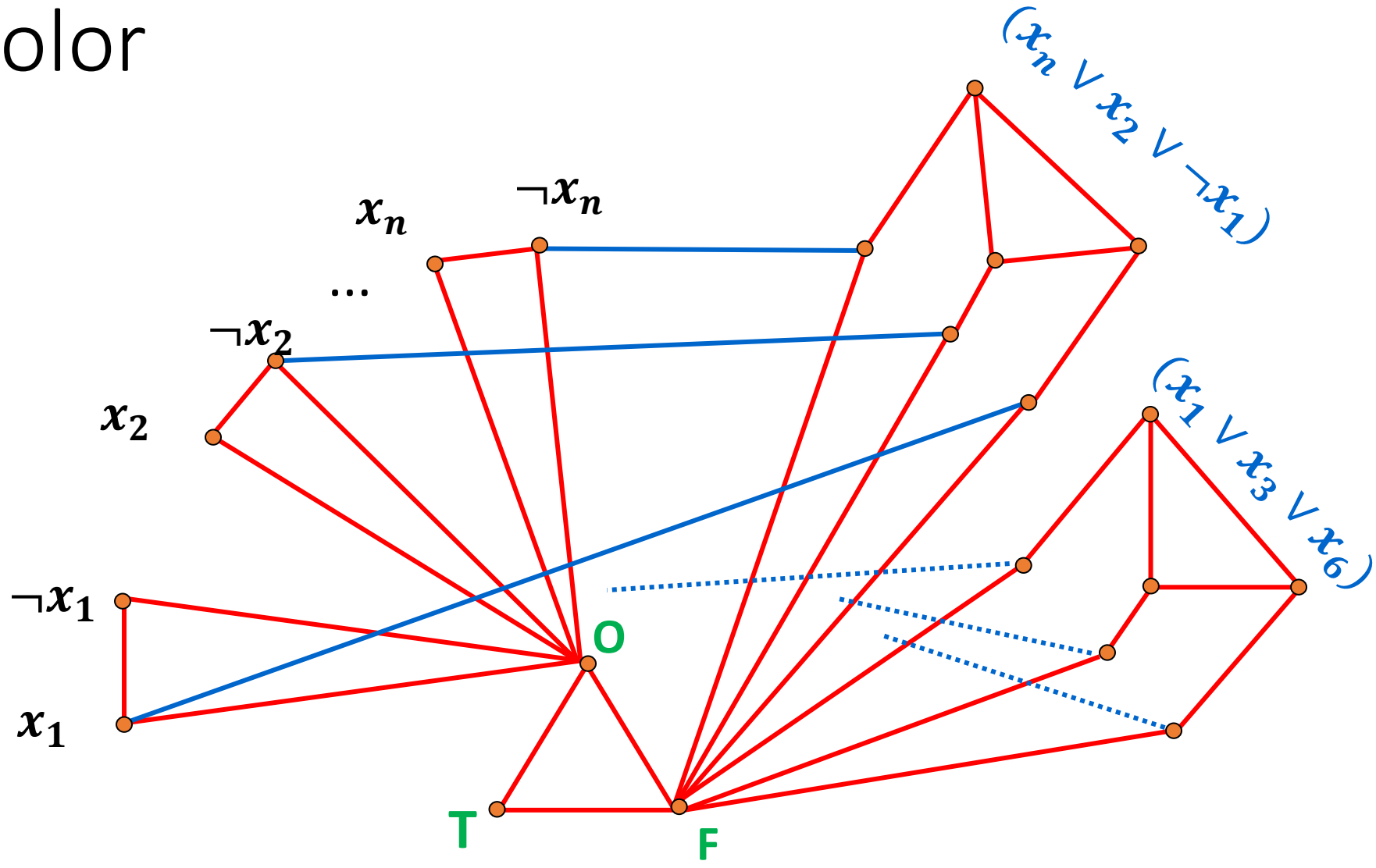
3SAT $\leq_P$ 3Color

Key property:

In any 3-coloring:

outer nodes either T or O

inner triangle must use O



Showing 3Color is NP-Hard

3Sat

$(x_1 \vee \neg x_3 \vee x_4) \wedge$
$(x_2 \vee \neg x_4 \vee x_3) \wedge$
$(x_2 \vee \neg x_1 \vee x_3)$

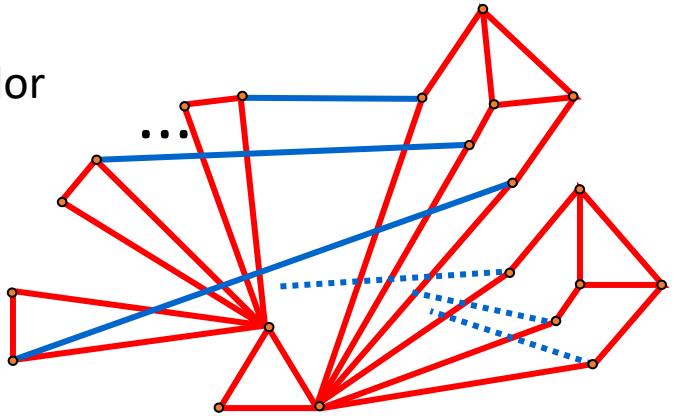
$O(n^p)$

Create “base triangle” and one node per variable and negation. Connect each variable node to the “false color” node. Per clause, create a triangle and one middle node per literal. Connect each middle to the triangle, false, and the opposite variable

Use the same answer

Reduction

3Color



Algorithm for solving 3Color

Solution for the instance of 3Sat

Yes/No

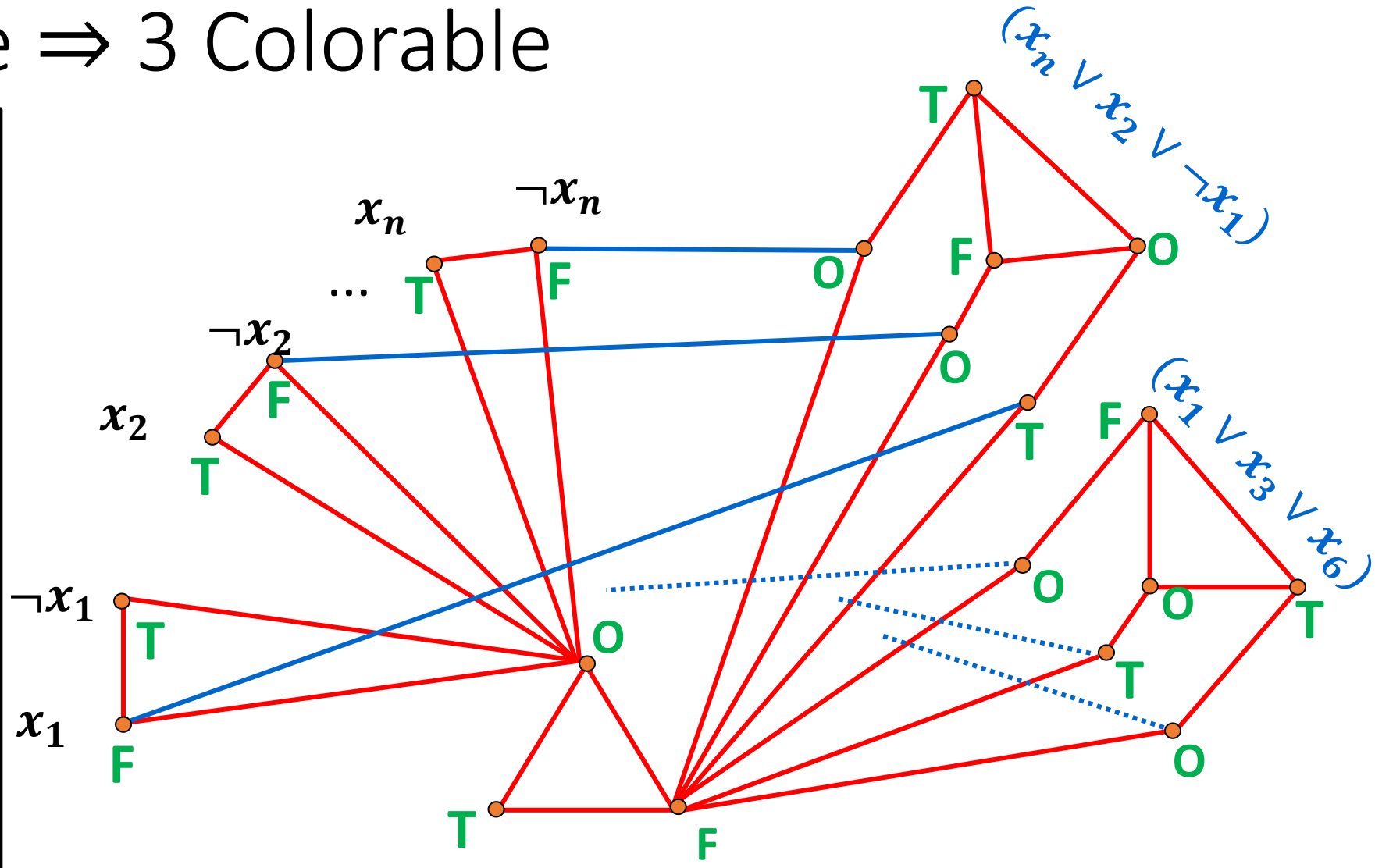
Solution for the instance of 3Color

Yes/No

F satisfiable ⇒ 3 Colorable

Suppose F is satisfiable. We can then 3-Color the graph by:

- Make each True literal node T-colored
- Make each False literal node F-colored
- Make one True middle node per clause T-colored
- Make the remaining middle nodes O-colored
- Color each outer triangle (node connect to the T-colored middle node will be O-colored, the others can be either T-colored or F-colored)



3 Colorable $\Rightarrow F$ satisfiable

Suppose the graph is 3-colorable. We can satisfy F by:

- Making each T-colored literal node True and each F-colored literal node False
  - No nodes are O-colored, so this will work out
- We know this satisfies F because:
  - Each clause will have one T-colored middle node (connected to the O-colored outer triangle node) which matches the color of its equivalent literal