P, NP, NP-hard, and NP-complete

SAT, 3SAT $\in$ NP

- SAT. Given a CNF formula ϕ, does it have a satisfying truth assignment?
- 3SAT. A SAT formula where each clause contains exactly 3 literals (corresponding to different variables)
- $\phi = (\overline{x_1} \vee x_2, \vee x_3) \wedge (x_1, \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$
- Satisfying instance: $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$, where 1 : true, 0 : false
- SAT, 3-SAT $\in$ NP
    - Certificate: truth assignment to variables
    - Poly-time verifier: check if assignment evaluates to true

NP-hard and NP-Complete Problems

Cook-Levin Theorem (Idea)

- If 3SAT can be solved in polynomial time, then *any* problem in NP can be solved in polynomial time

- So: if 3SAT can be solved in polynomial time, then P = NP

NP-hard intuition

- Our goal is to say that a problem X is NP-hard if:
  - If X can be solved in polynomial time, then any problem in NP can be solved in polynomial time
  - Therefore, if X can be solved in polynomial time, then P = NP

What does this mean?

- We think that, probably, $P \neq NP$
- So if a problem is NP-hard, then you probably cannot obtain a polynomial-time algorithm for it

Classifying Problems as Hard

- We are frustratingly unable to prove a lot of problems are **impossible** to solve efficiently
- Instead, we say problem X is likely very hard to solve by saying, *if a polynomial-time algorithm was found for* X*, then something we all believe is impossible will happen*
- Idea: X is NP-hard $\Rightarrow$ if $X \in$ P, then P = NP
- (Erickson) Calling a problem NP hard is like saying, *"If I own a dog, then it can speak fluent English"*
  - You probably don't know whether or not I own a dog, but you are definitely sure I don't own *a talking dog*
  - Corollary: No one should believe that I own a dog
- If a problem is NP hard, no one should believe it can be solved in polynomial time

Use of Reductions: $X \leq_p Y$

Design algorithms:

- If Y can be solved in polynomial time, we know X can also be solved in polynomial time

Establish intractability:

- If we know that X is known to be impossible/hard to solve in polynomial-time, then we can conclude the same about problem Y

Establish Equivalence:

- If $X \leq_p Y$ and $Y \leq_p X$ then X can be solved in poly-time iff Y can be solved in poly time and we use the notation $X \equiv_p Y$

Digging Deeper

- **Graph 2-Color** reduces to **Graph 3-color**
    - Just replace the third color with either of the two
- **Graph 2-Color** can be solved in polynomial time
    - How?
        - We can decide if a graph is bipartite in $O(n + m)$ time using traversal
- **Graph 3-color** (we'll show) is NP hard

Intuitively, if problem X reduces to problem Y, then solving X is *no harder* than solving Y

$$X \leq_p Y$$

Relative Hardness

- Suppose we know problem X is NP hard, how can we use that to show problem Y is also hard to solve?
- How do we compare the relative hardness of problems
- Recurring idea in this class: **reductions!**
- Informally, we say a problem X reduces to a problem Y, if can use an algorithm for Y to solve X
    - Bipartite matching reduces to max flow
    - Edge-disjoint paths reduces to max flow

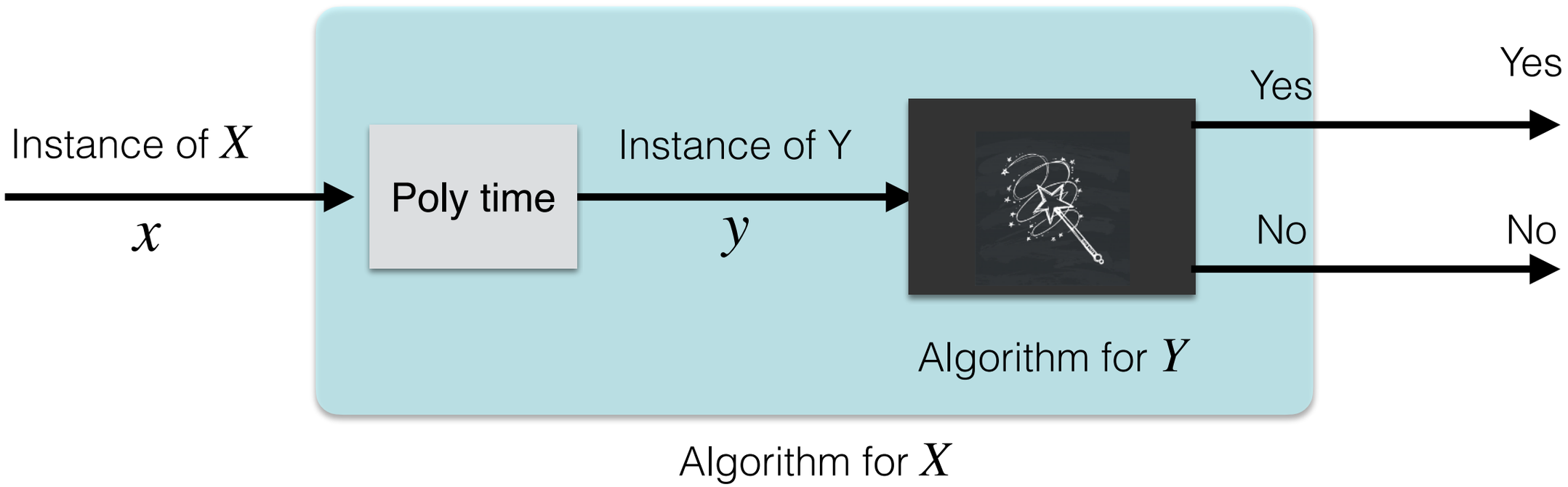
Intuitively, if problem X reduces to problem Y, then solving X is no harder than solving Y

$$X \leq_p Y$$

[Karp] Reductions

Definition. Decision problem X polynomial-time (Karp) reduces to decision problem Y if given any instance x of X, we can construct an instance y of Y in polynomial time s.t $x \in X$ if and only if $y \in Y$.

Notation. $X \leq_p Y$

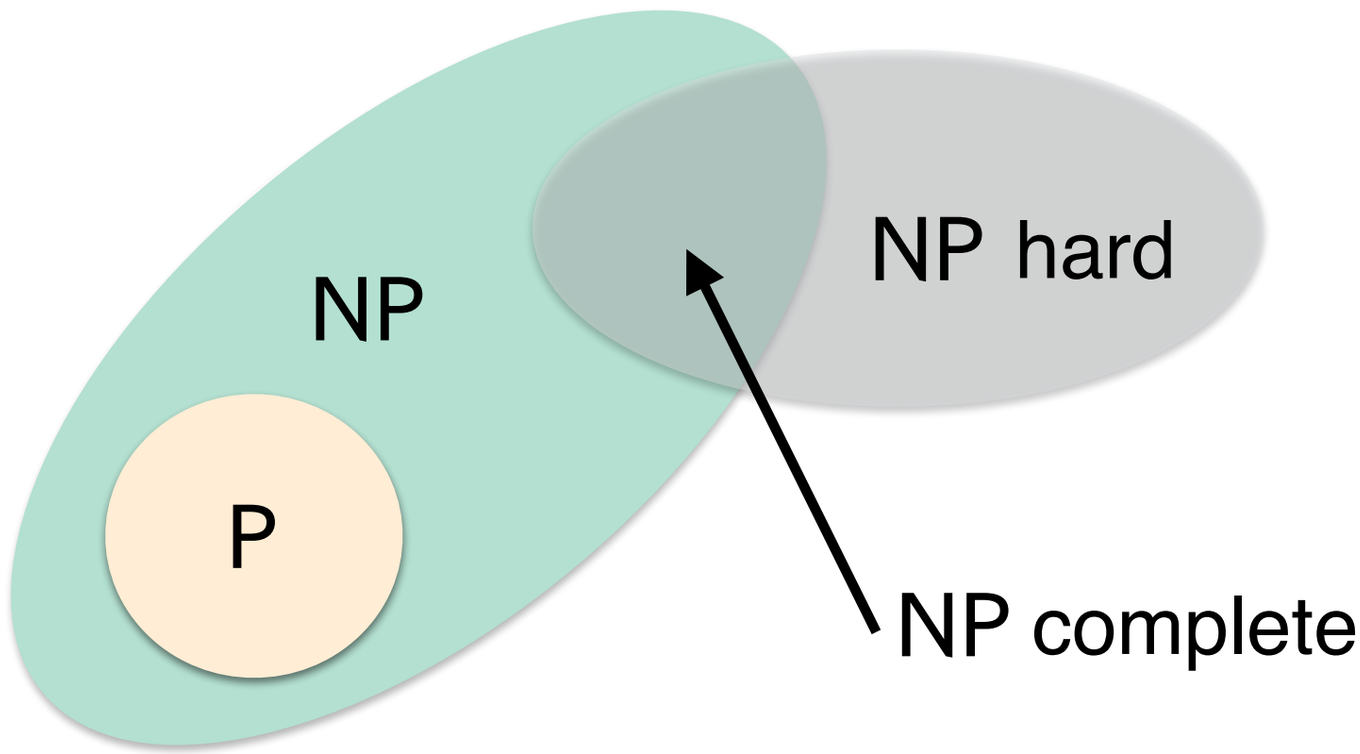


NP hard: Definition

- **We will show problems are NP hard using reductions.**
  - A problem Y is **NP hard**, if, for any problem $X \in \mathsf{NP}$, $X \leq_p Y$
- This means that if $Y \in \mathsf{P}$, then $\mathsf{P} = \mathsf{NP}$
- Cook-Levin theorem [1973]: 3SAT is NP hard

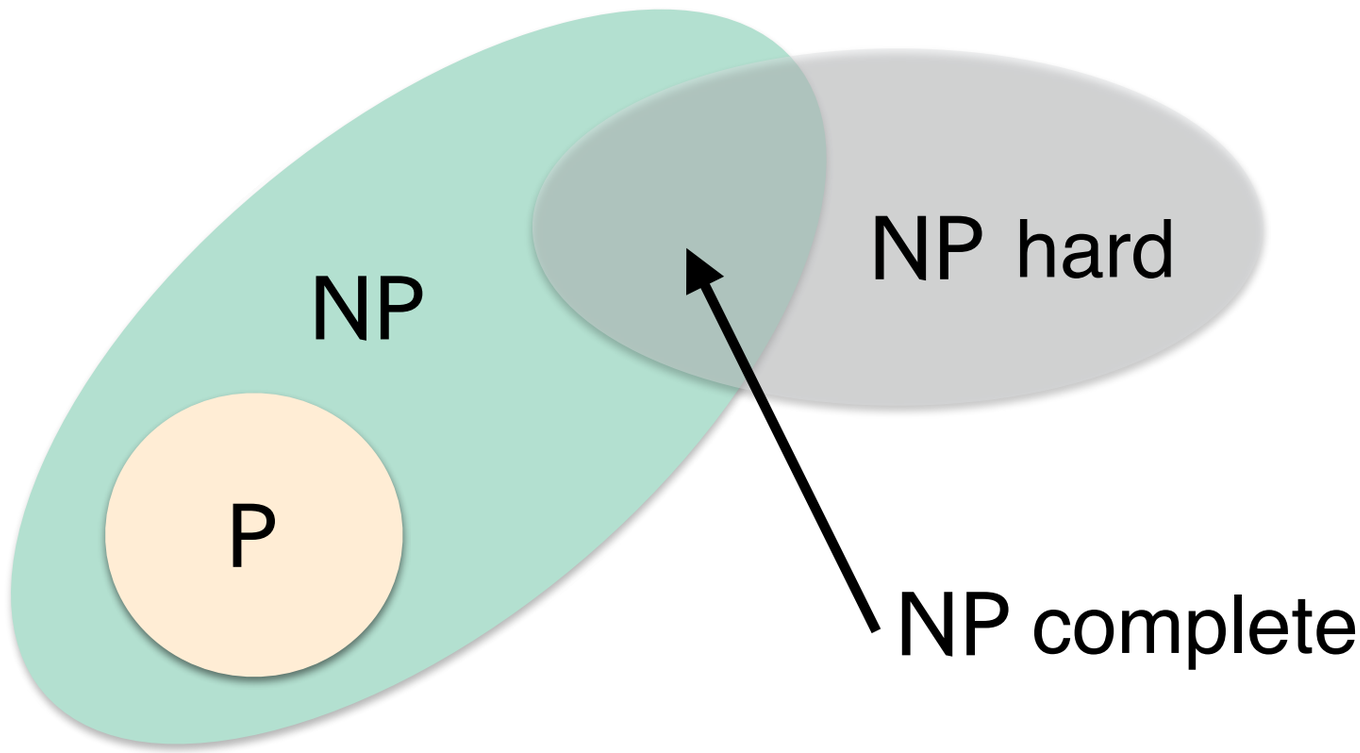
NP Completeness

- **Definition.** A problem X is NP complete if X is NP hard and $X \in$ **NP**
- 3SAT is **NP** complete
    - 3SAT $\in$ **NP**: given an assignment to input gates (certificate), can verify whether output is one or zero in poly-time
    - 3SAT is **NP** hard (Cook-Levin Theorem)



Summary

- "X is NP-hard" $\Leftrightarrow$ "$X \in$ P if and only if P = NP"
- A problem X is NP complete if X is NP hard and $X \in$ NP
- Thus, NP-complete problems are the hardest problems in NP



Proving NP Hardness

- To prove problem Y is **NP**-hard
  - Difficult to prove every problem in **NP** reduces to Y
  - Instead, we use a known-NP-hard problem Z
  - We know every problem X in **NP**, $X \leq_p Z$
  - Notice that $\leq_p$ is transitive
  - Thus, enough to prove $Z \leq_p Y$

To prove that a problem Y is NP hard, reduce a known NP hard problem Z to Y

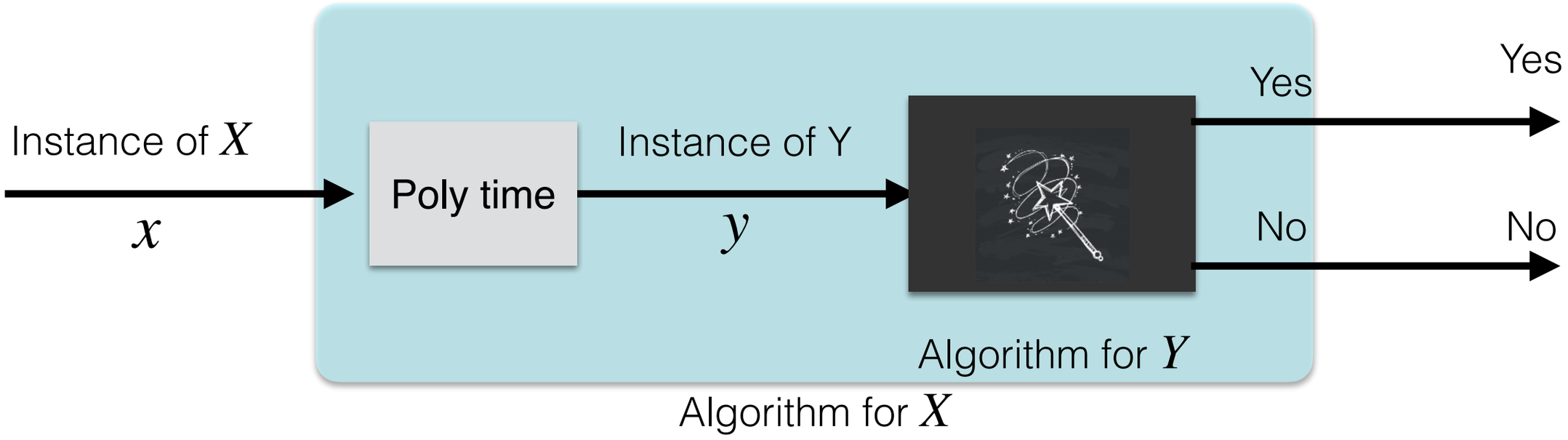
Known NP Hard Problems?

- For now: SAT (Cook-Levin Theorem)
- We will prove a whole repertoire of NP hard and NP complete problems by using reductions
- Before reducing SAT to other problems to prove them NP hard, let us practice some easier reductions first

To prove that a problem Y is NP hard, reduce a known NP hard problem Z to Y

Reductions: General Pattern

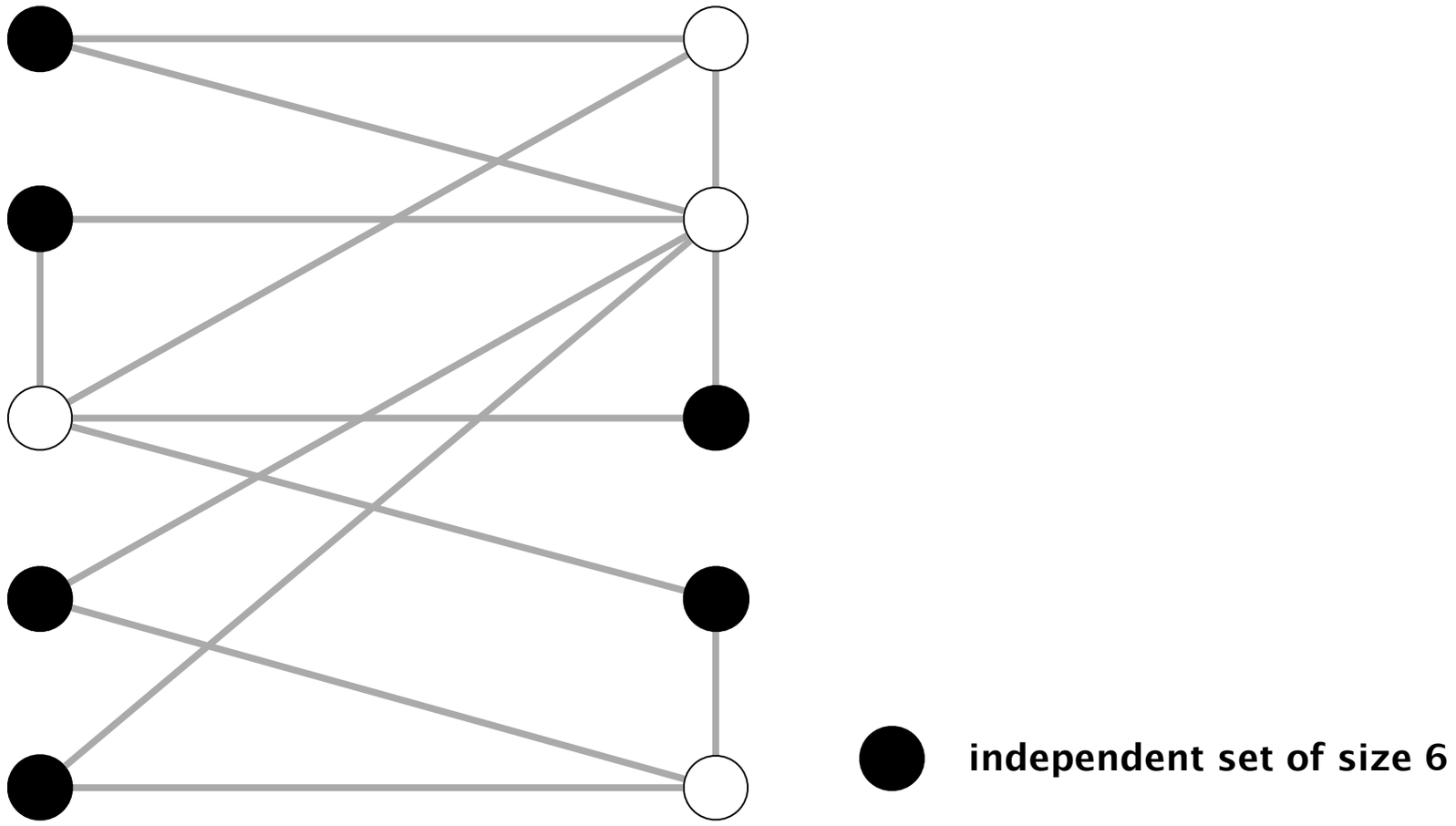
- Describe a polynomial-time algorithm to transform an arbitrary instance x of Problem X into a special instance y of Problem Y
- Prove that if x is a "yes" instance of X, then y is a "yes" instance of Y
- Prove that if y is a "yes" instance of Y, then x is a "yes" instance of X
- Notice that correctness of reductions are not symmetric:
  - the "if" proof needs to handle arbitrary instances of X
  - the "only if" needs to handle the special instance of Y



$$\text{VERTEX-COVER} \equiv_p \text{IND-SET}$$

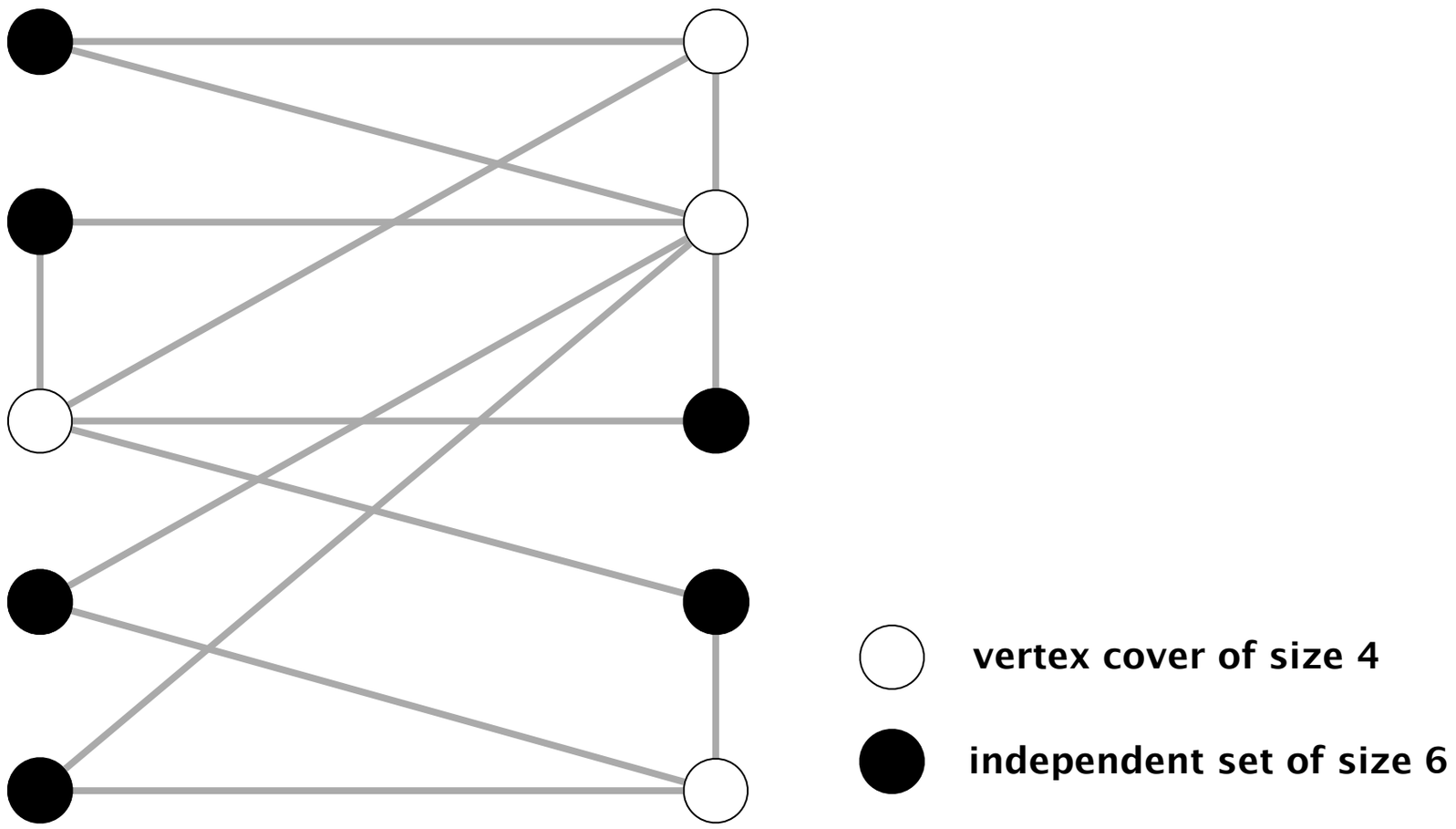
IND-SET

- Given a graph $G = (V, E)$, an independent set is a subset of vertices $S \subseteq V$ such that no two of them are adjacent, that is, for any $x, y \in S$, $(x, y) \notin E$
- **IND-SET Problem.** Given a graph $G = (V, E)$ and an integer k, does G have an independent set of size at least k?



Vertex-Cover

- Given a graph $G = (V, E)$, a vertex cover is a subset of vertices $T \subseteq V$ such that for every edge $e = (u, v) \in E$, either $u \in T$ or $v \in T$.
- **VERTEX-COVER Problem.** Given a graph $G = (V, E)$ and an integer k, does G have a vertex cover of size at most k?



Our First Reduction

- **VERTEX-COVER** $\leq_p$ **IND-SET**
    - Suppose we know how to solve independent set, can we use it to solve vertex cover?

- **Claim.** S is an independent set of size k iff $V \setminus S$ is a vertex cover of size $n - k$.

- **Proof.** ($\Rightarrow$) Consider an edge $e = (u, v) \in E$
    - S is independent: u, v both cannot be in S
    - At least one of $u, v \in V \setminus S$
    - $V \setminus S$ covers e ∎

Our First Reduction

- **VERTEX-COVER** $\leq_p$ **IND-SET**
    - Suppose we know how to solve independent set, can we use it to solve vertex cover?

- **Claim.** S is an independent set of size k iff $V \setminus S$ is a vertex cover of size $n - k$.

- **Proof.** ($\Leftarrow$) Consider an edge $e = (u, v) \in E$
    - $V \setminus S$ is a vertex cover: at least one of u, v or both must be in $V \setminus S$
    - Both u, v cannot be in S
    - Thus, S is an independent set. ∎

Vertex Cover $\equiv_p$ IND Set

- **VERTEX-COVER** $\leq_p$ **IND-SET**
    - Suppose we know how to solve independent set, can we use it to solve vertex cover?
- **Reduction.** Let $G' = G,\ k' = n - k$.
    - ($\Rightarrow$) If G has a vertex cover of size at most k then G' has an independent set of size at least k'
    - ($\Leftarrow$) If G' has an independent set of size at least k' then G has a vertex cover of size at most k
- **IND-SET** $\leq_p$ **VERTEX-COVER**
    - Same reduction works: $G' = G$, $k' = n - k$
- **VERTEX-COVER** $\equiv_p$ **IND-SET**

IND-SET is NP Complete:

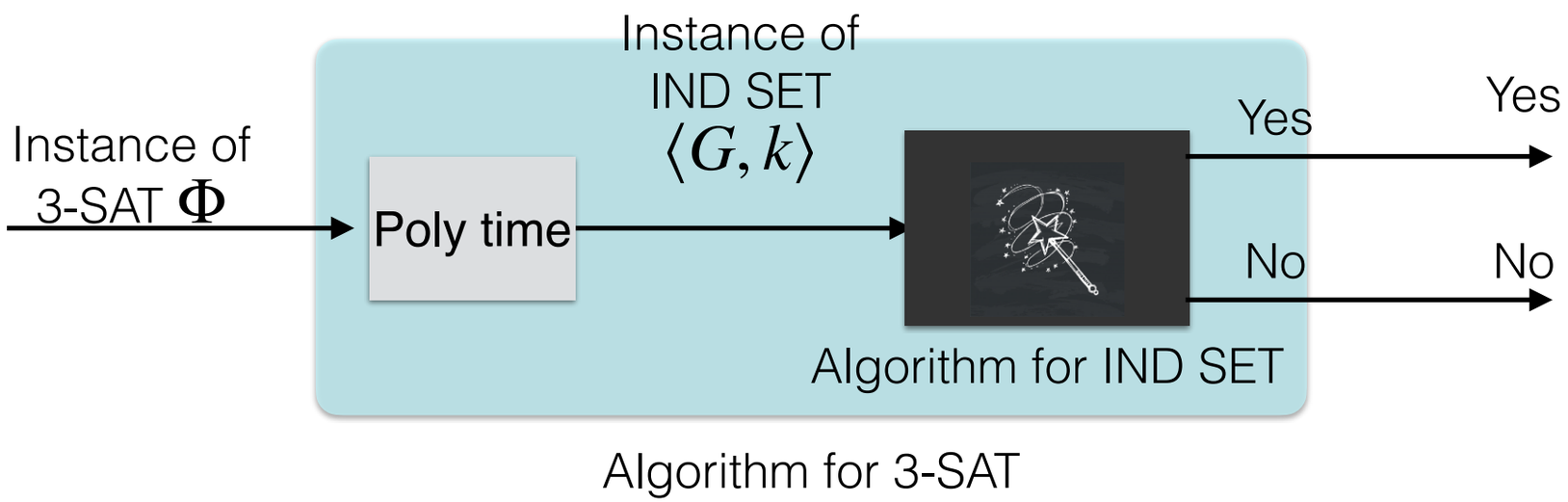
3SAT $\leq_p$ IND-SET

IND-SET: NP Complete

- To show Independent set is NP complete
    - Show it is in NP (already did in previous lectures)
    - Reduce a known NP complete problem to it
        - We will use 3-SAT
- Looking ahead: once we have shown 3-SAT $\leq_p$ IND-SET
    - Since **IND-SET $\leq_p$ Vertex Cover**
    - And **Vertex Cover $\leq_p$ Set Cover**
    - We can conclude they are also NP hard
    - As they are both in NP, they are also NP complete!

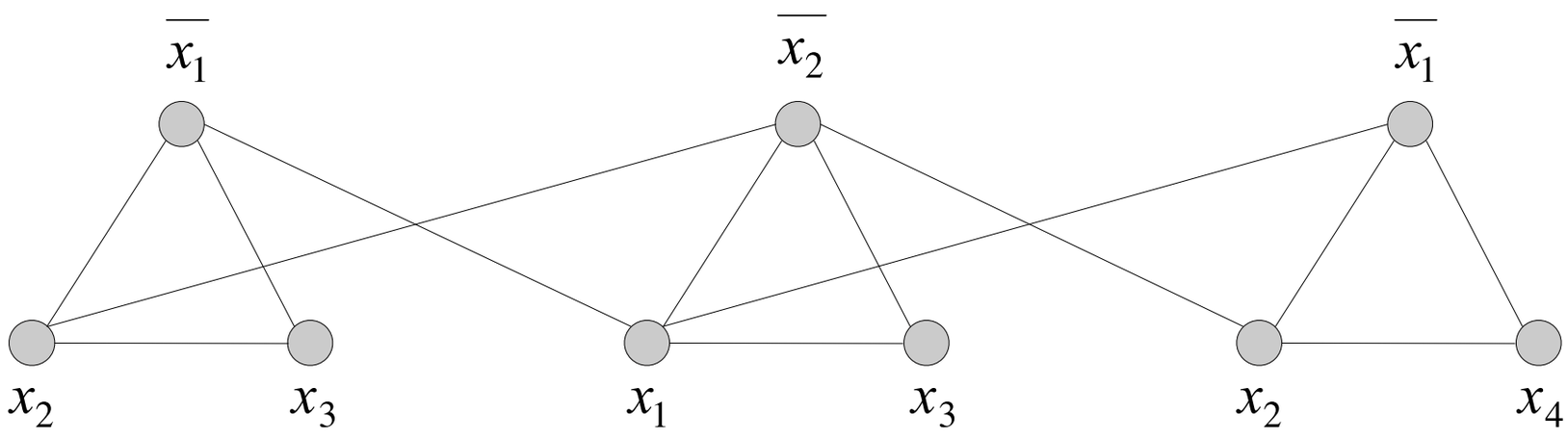
IND-SET: NP hard

- **Theorem.** **3-SAT $\leq_p$ IND-SET**
- Given an instance Φ of 3-SAT, we construct an instance $\langle G, k \rangle$ of IND-SET s.t. G has an independent set of size k iff ϕ is satisfiable.



3SAT $\leq_p$ IND-SET

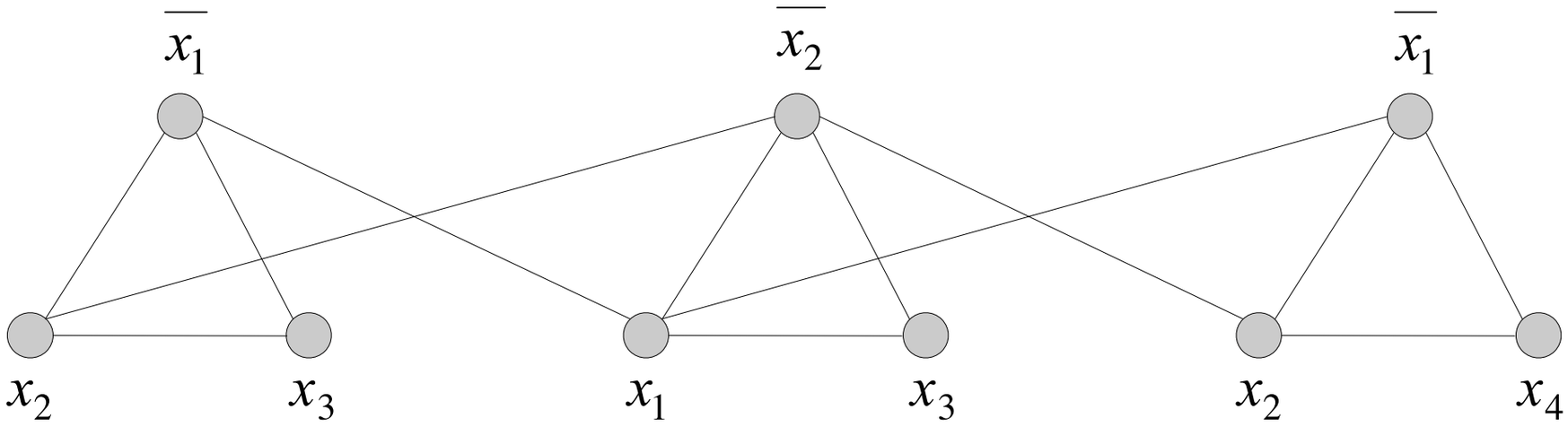
- **Reduction.** Let k be the number of clauses in Φ.
  - G has $3k$ vertices, one for each literal in Φ
  - (Clause gadget) For each clause, connect the three literals in a triangle
  - (Variable gadget) Each variable is connected to its negation



$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3 \right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3 \right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4 \right)$$

3SAT $\leq_p$ IND-SET

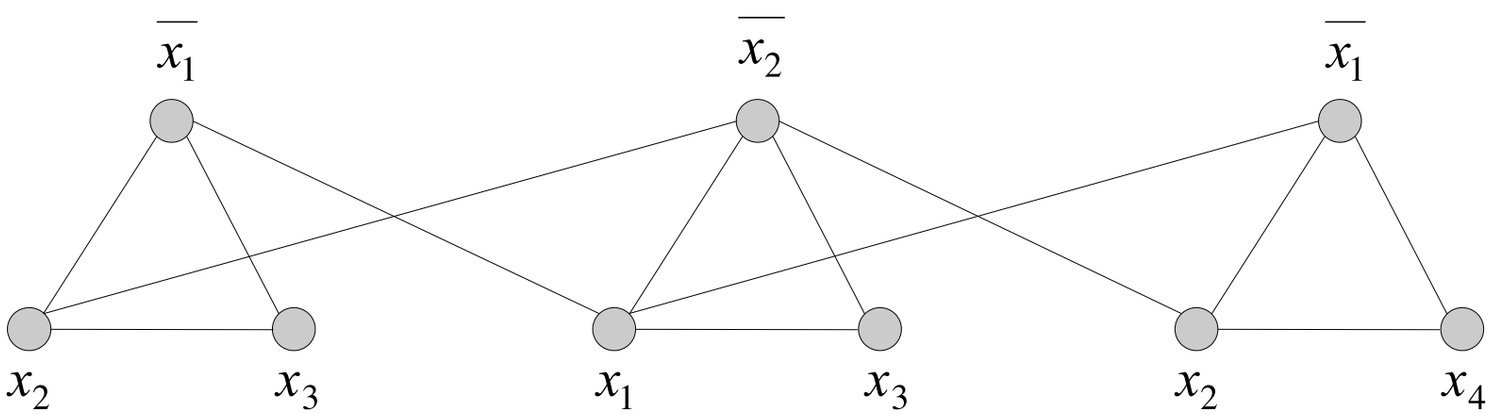
- **Observations.**
  - Any independent set is G can contain at most 1 vertex from each clause triangle
  - Only one of x_i or $\overline{x_i}$ can be in an independent set (*consistency*)



$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3 \right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3 \right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4 \right)$$

3SAT$\leq_p$ IND-SET

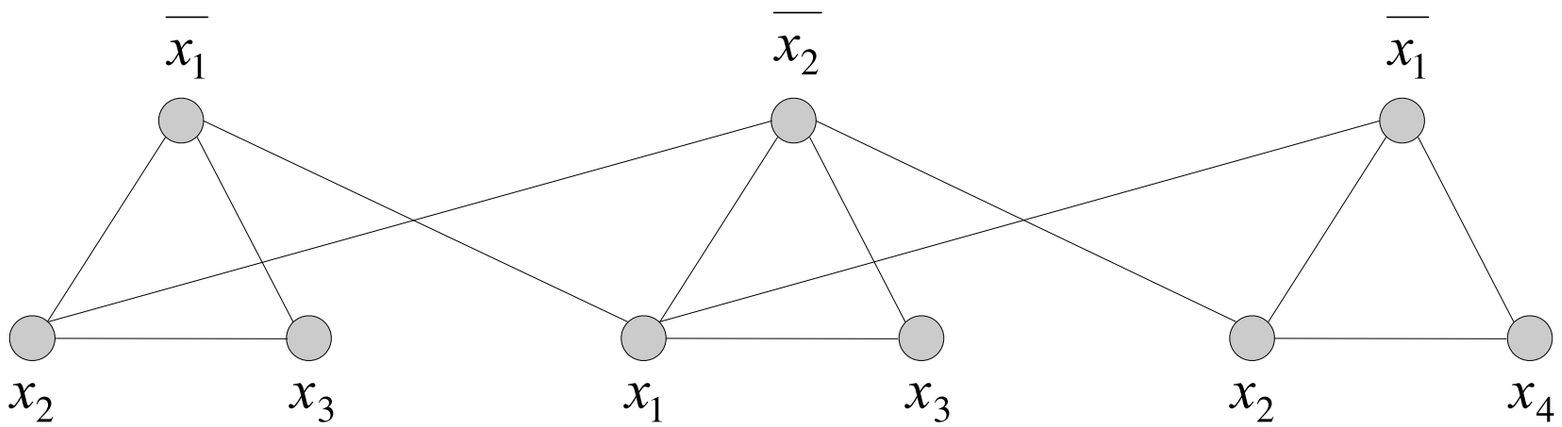
- **Claim.** Φ is satisfiable iff G has an independent set of size k
- ($\Rightarrow$) Suppose Φ is satisfiable, consider a satisfying assignment
    - There is at least one true literal in each clause
    - Select one true literal from each clause/triangle
    - This is an independent set of size k



$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3 \right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3 \right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4 \right)$$

3SAT$\leq_p$ IND-SET

- **Claim.** Φ is satisfiable iff G has an independent set of size $k = |\phi|$
- ($\Leftarrow$) Let S be in an independent set in G of size k
    - S must contain exactly one node in each triangle
    - Set the corresponding literals to *true*
    - Set remaining literals consistently
    - All clauses are satisfied — Φ is satisfiable ∎



$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4\right)$$

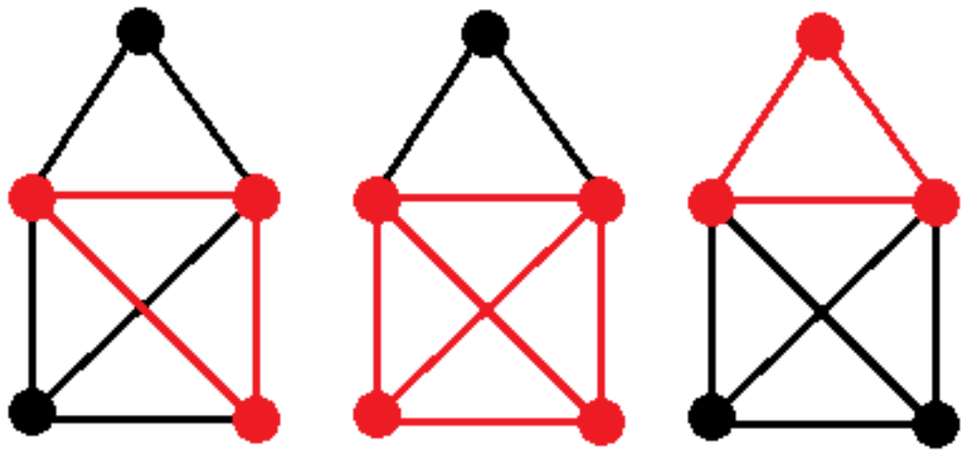
Reduction Strategies

- Equivalence
  - **VERTEX-COVER** $\equiv_p$ **IND-SET**
- Special case to general case
  - **VERTEX-COVER** $\leq_p$ **SET-COVER**
- Encoding with gadgets
  - **3-SAT** $\leq_p$ **IND-SET**
- Transitivity
  - **3-SAT** $\leq_p$ **IND-SET** $\leq_p$ **VERTEX-COVER** $\leq_p$ **SET-COVER**
  - Thus, **IND-SET**, **VERTEX-COVER** and **SET-COVER** are NP hard
  - Since they are all in NP, also NP - complete

IND-SET $\leq_p$ Clique

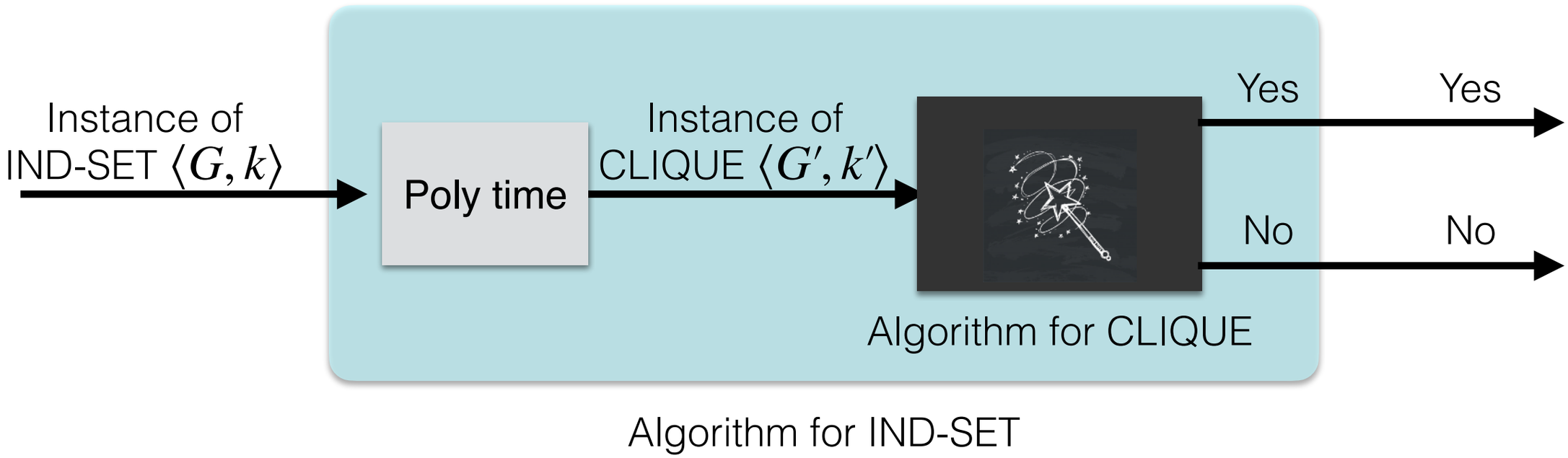
Clique

- A **clique** in an undirected graph is a subset of nodes such that every two nodes are connected by an edge. A k-clique is a clique that contains k nodes.
- **CLIQUE.** Given a graph G and a number k, does G contain a k-clique?



IND-SET to CLIQUE

- **Theorem.** IND-SET $\leq_p$ CLIQUE.
- **We want to:** Reduce IND-SET to Clique. Given instance $\langle G, k \rangle$ of independent set, construct an instance $\langle G', k' \rangle$ of clique such that
    - G has independent set of size k iff G' has clique of size k'.



IND-SET to CLIQUE

- **Theorem.** IND-SET $\leq_p$ CLIQUE.
- Proof. Given instance $\langle G, k \rangle$ of independent set, we construct an instance $\langle G', k' \rangle$ of clique such that G has independent set of size k iff G' has clique of size k'
- **Reduction.**
  - Let $G' = (V, \overline{E})$, where $e = (u, v) \in \overline{E}$ iff $e \notin E$
  - Let $k' = k$
  - ($\Rightarrow$) G has an independent set S of size k, then S is a clique in G'
  - ($\Leftarrow$) G' has a clique Q of size k, then Q is an independent set in G

List of NPC Problems So Far

- SAT
- 3-SAT
- INDEPENDENT SET
- VERTEX COVER
- SET COVER
- CLIQUE
- More to come:
    - Subset Sum/Knapsack
    - 3-COLOR
    - Hamiltonian cycle / path

SUBSET-SUM is NP Complete:

Vertex-Cover $\leq_p$ SUBSET-SUM

Subset Sum Problem

- **SUBSET-SUM**.
  Given n positive integers $a_1, \ldots, a_n$ and a target integer T, is there a subset of numbers that adds up to exactly T

- **SUBSET-SUM** $\in$ NP

    - Certificate: a subset of numbers
    - Poly-time verifier: checks if subset is from the given set and sums exactly to T

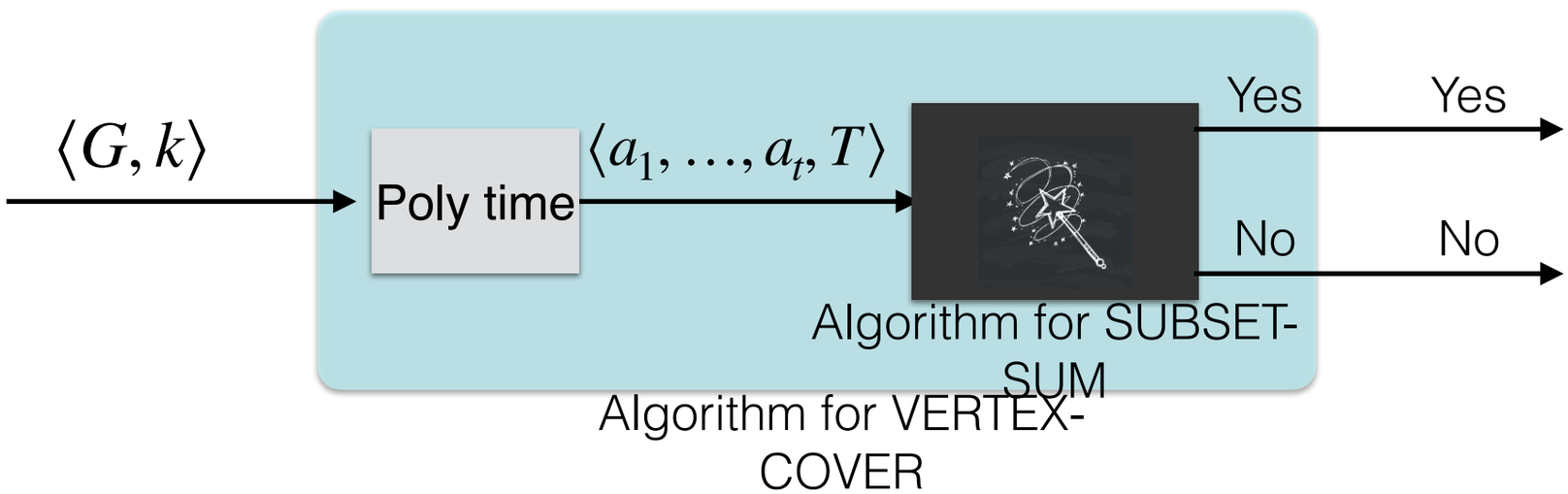
- Problem has a pseudo-polynomial $O(nT)$-time dynamic programming algorithm similar to Knapsack

- Will prove **SUBSET-SUM** is **NP hard:** reduction from vertex cover

- NP hard problems that have pseudo-polynomial algorithms are called *weakly NP hard*

Vertex Cover to Subset Sum

- **Theorem.** VERTEX-COVER $\leq_p$ SUBSET-SUM
- Proof. Given a graph G with n vertices and m edges and a number k, we construct a set of numbers $a_1, \ldots, a_t$ and a target sum T such that G has a vertex cover of size k iff there is a subset of numbers that sum to T

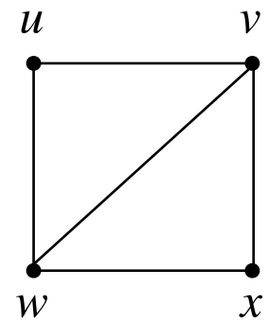


Vertex Cover to Subset Sum

- **Theorem.** VERTEX-COVER $\leq_p$ SUBSET-SUM
- **Proof.** Label the edges of G as $0, 1, \ldots, m-1$.
- **Reduction**. Create $n + m$ integers and a target value T as follows
- Each integer is a $m + 1$-bit number in base four
- Integers representing vertices and edges:
  - Vertex integer a_v : mth (most significant) bit is 1 and for $i < m$, the ith bit is 1 if ith edge is incident to vertex v
  - Edge integer b_{uv} : mth digit is 0 and for $i < m$, the ith bit is 1 if this integer represents an edge $i = (u, v)$
  - Target value $T = k \cdot 4^m + \sum_{i=0}^{m-1} 2 \cdot 4^i$

Vertex Cover to Subset Sum

- Example: consider the graph $G = (V, E)$ where $V = \{u, v, w, x\}$ and $E = \{(u, v),\ (u, w),\ (v, w),\ (v, x),\ (w, x)\}$



	5th	4th : (wx)	3rd : (vx)	2nd : (vw)	1st : (uw)	0th : (uv)
a_u	1	0	0	0	1	1
a_v	1	0	1	1	0	1
a_w	1	1	0	1	1	0
a_x	1	1	1	0	0	0
b_{uv}	0	0	0	0	0	1
b_{uw}	0	0	0	0	1	0
b_{vw}	0	0	0	1	0	0
b_{vx}	0	0	1	0	0	0
b_{wx}	0	1	0	0	0	0

$$a_u := 111000_4 = 1344$$
$$a_v := 110110_4 = 1300$$
$$a_w := 101101_4 = 1105$$
$$a_x := 100011_4 = 1029$$

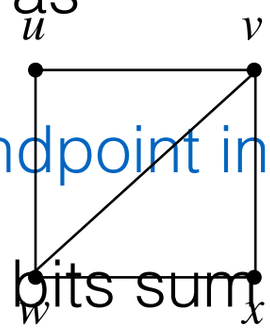
$$b_{uv} := 010000_4 = 256$$
$$b_{uw} := 001000_4 = 64$$
$$b_{vw} := 000100_4 = 16$$
$$b_{vx} := 000010_4 = 4$$
$$b_{wx} := 000001_4 = 1$$

- If $k = 2$ then $T = 222222_4 = 2730$

Correctness

- **Claim.** G has a vertex cover of size k if and only there is a subset X of corresponding integers that sums to value T
- ($\Rightarrow$) Let C be a vertex cover of size k in G, define X as

  $X := \{a_v \mid v \in C\} \cup \{b_i \mid$ edge i has exactly one endpoint in $C\}$



- Sum of the most significant bits of X is k and all other bits sum to 2
- Thus the elements of X sum to exactly T

$$C = \{v, w\}$$

$$T = k \cdot 4^m + \sum_{i=0}^{m-1} 2 \cdot 4^i$$

$$T = 222222_4 = 2730$$

Vertex Cover to Subset Sum

- **Claim.** G has a vertex cover of size k if and only there is a subset X of corresponding integers that sums to value T
- $(\Leftarrow)$ Let X be the subset of numbers that sum to T
- That is, there is $V' \subseteq V, E' \subseteq E$ s.t.

$$X := \sum_{v \in V'} a_v + \sum_{i \in E'} b_i = T = k \cdot 4^m + \sum_{i=0}^{m-1} 2 \cdot 4^i$$

- These numbers are base 4 and there are no carries
- Each b_i only contributes 1 to the ith digit, which is 2
- Thus, for each edge i, at least one of its endpoints must be in V'
    - V' is a vertex cover
- Size of V' is k: only vertex-numbers have a 1 in the mth position

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)