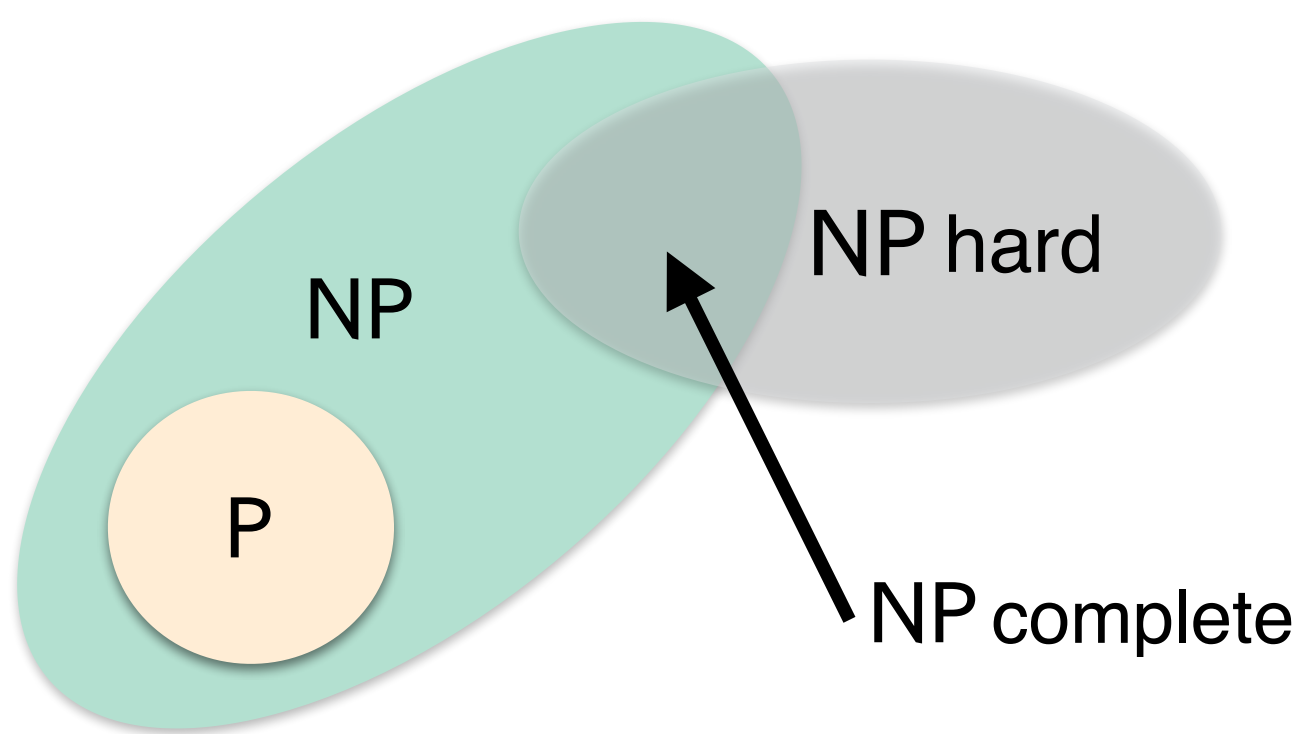
NP Hardness Reductions

Overview

- We have defined classes P and NP
- Notion of NP hardness and NP completeness
- A problem X is NP-hard $\equiv$ if $X \in$ P then P $=$ NP (alternate definition: every problem in NP poly-time reduces to it)
- A problem X is NP-complete if it is NP-hard and in NP

We will define these reductions today



Overview

- We have defined classes P and NP
- Notion of NP hardness and NP completeness
- A problem X is NP-hard $\equiv$ if $X \in$ P then P $=$ NP (alternate definition: every problem in NP poly-time reduces to it)
- A problem X is NP-complete if it is NP-hard and in NP
- (Cook-Levin). 3SAT/SAT is NP hard
- Today: **Problem reductions!**
  - Strategy to prove a problem is NP hard—
    Reduce a known NP hard problem to it
- Will do a bunch of reductions

Relative Hardness

- 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
  - Find max-weight feedback set reduces to finding max spanning trees (which in turn reduces to finding MSTs)
  - Finding opportunity cycles reduce to negative cycles

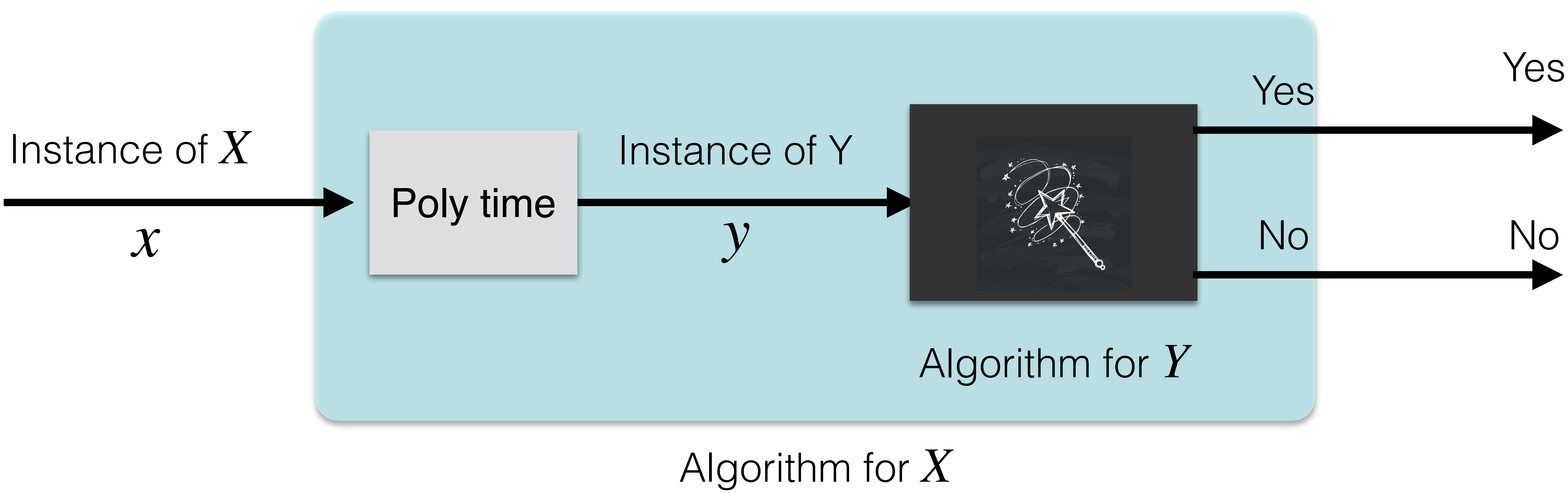
Intuitively, if problem X reduces to problem Y, then solving X is no harder than solving Y

[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. $X \leq_p Y$

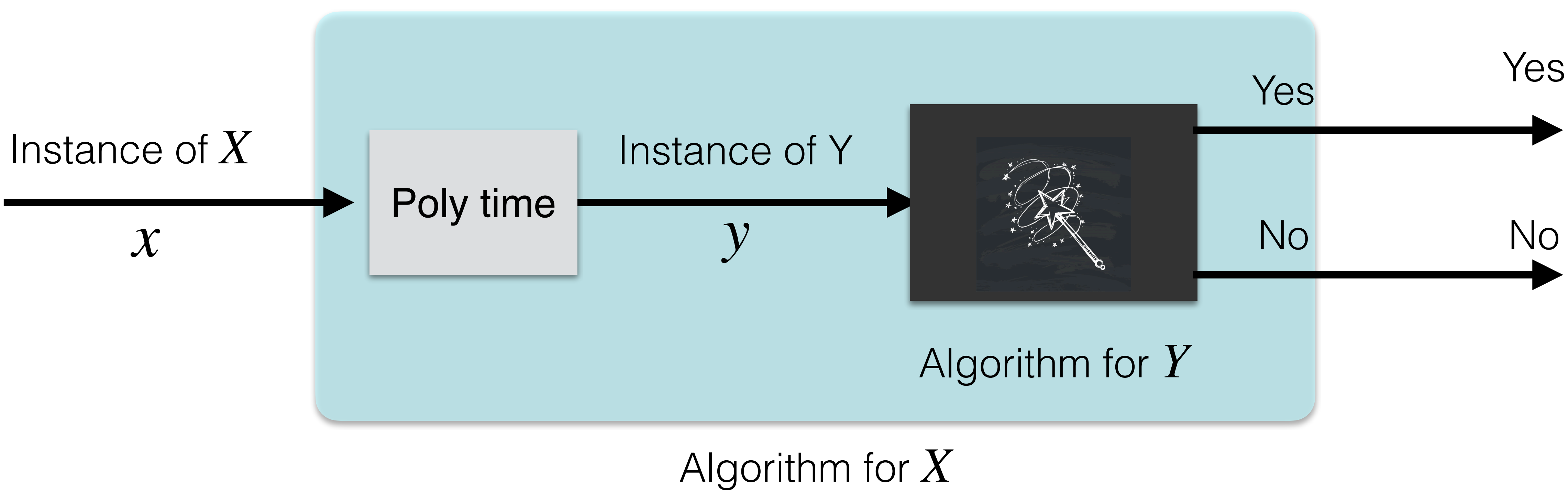
- Solving X is no harder than solving Y: if we have an algorithm for Y, we can use it + poly time reduction to solve X



Reductions Quiz

Say $X \leq_p Y$. Which of the following can we infer?

- If X can be solved in polynomial time, then so can Y.
- X can be solved in poly time iff Y can be solved in poly time.
- If X cannot be solved in polynomial time, then neither can Y.
- If Y cannot be solved in polynomial time, then neither can X.



Digging Deeper

- **Graph 2-Color** reduces to **Graph 3-color**
    - We'll see this soon
- **Graph 2-Color** can be solved in polynomial time
    - How?
    - Can decide if a graph is bipartite in $O(n + m)$ time using BFS
- **Graph 3-color** (we'll show) is NP hard and unlikely to have a polynomial-time solution

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

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 $X \equiv_p Y$

NP hard: Operational Definition

- **New definition of NP hard using reductions.**
  - A problem Y is NP hard, if for any problem $X \in$ NP, $X \leq_p Y$
- Recall we said Y is NP hard if $Y \in$ P, then P $=$ NP.
- Lets show that both definitions are equivalent
  - ($\Rightarrow$) every problem in **NP** reduces to Y, and if $Y \in$ P, then P $=$ NP
  - ($\Leftarrow$) Suppose $Y \in$ P, then P $=$ NP: which means every problem in NP($=$ P) reduces to Y

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?

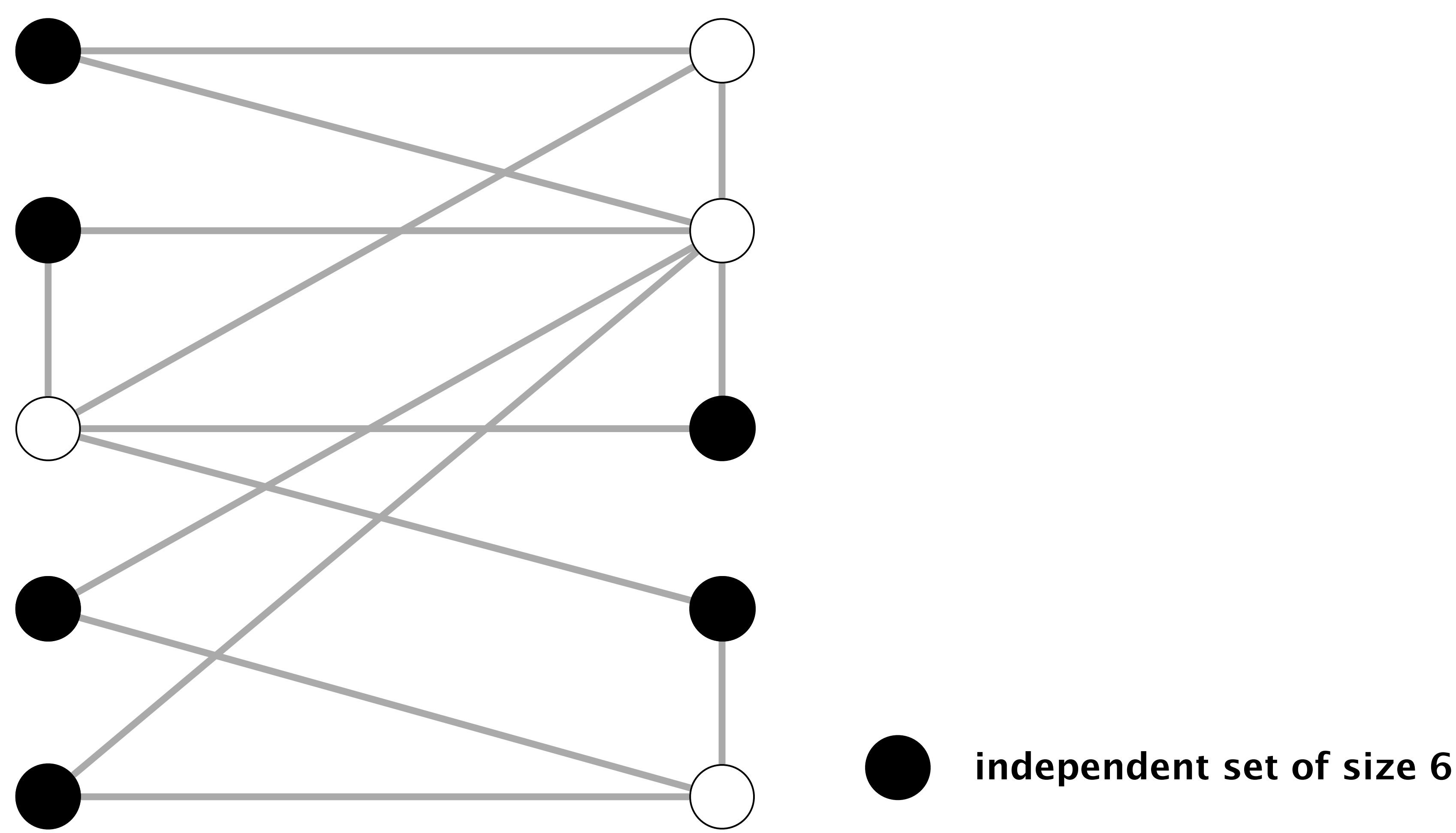
- For now: **3SAT** and **SAT** (Cook-Levin Theorem)
- We will prove a whole repertoire of NP hard and NP complete problems by using reductions
- Before reducing **3SAT** to other problems to prove them NP hard, let us practice some easier reductions first

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

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

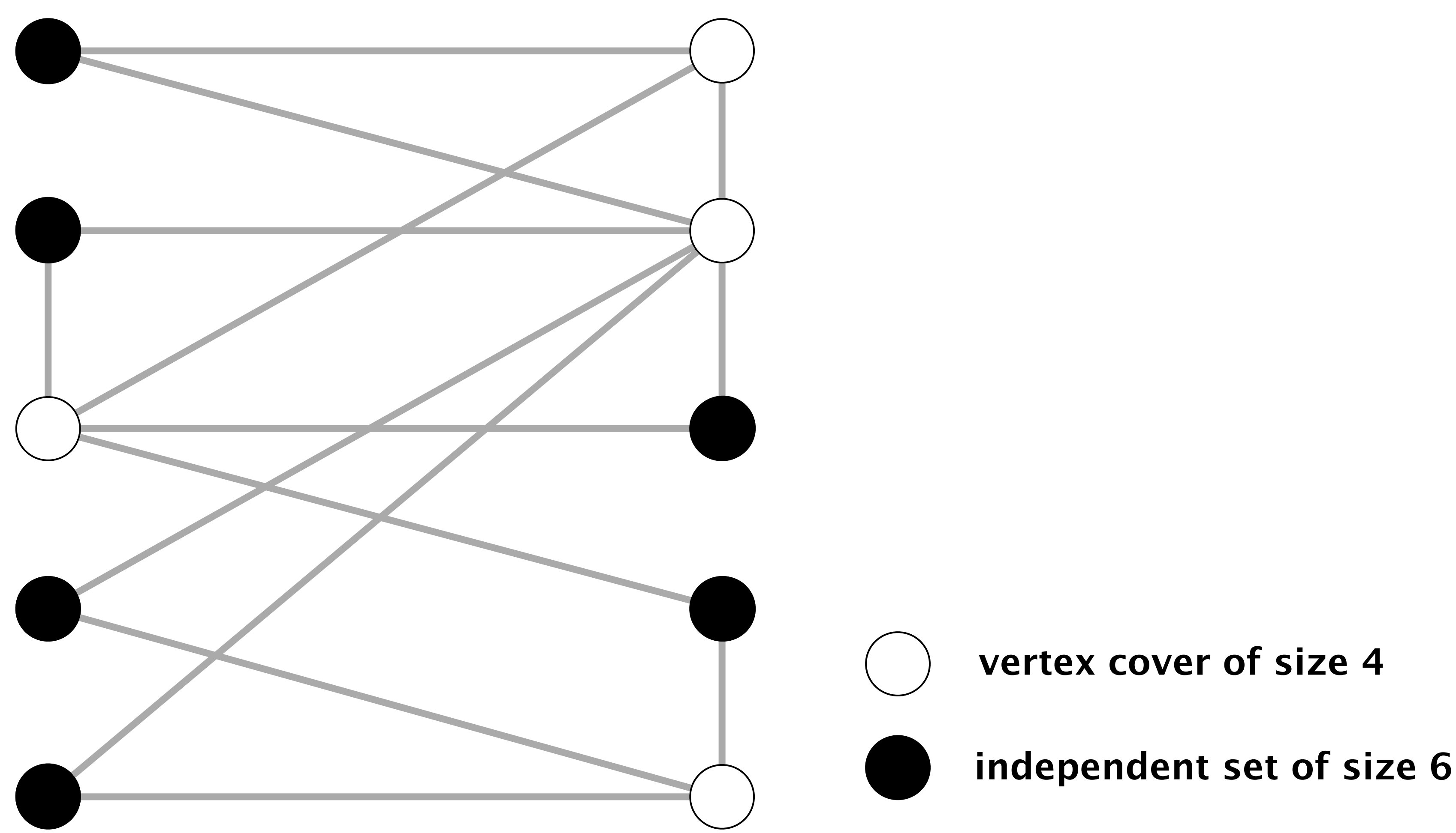
IND-SET

- 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 $V - S$ 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 $u, v \in V - S$
  - $V - S$ covers e
  - $\blacksquare$

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

Vertex Cover $\equiv_p$ IND Set

- **VERTEX-COVER** $\leq_p$ **IND-SET**
- **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**

VERTEX-COVER $\leq_p$ SET-COVER

Set Cover

- **Set-Cover.** Given a set U of elements, a collection $\mathcal{S}$ of subsets of U and an integer k, are there **at most** k subsets $S_1, \ldots, S_k$ whose union covers U, that is, $U \subseteq \cup_{i=1}^{k} S_i$

$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$S_a = \{ 3, 7 \}$ $\quad$ $S_b = \{ 2, 4 \}$

$S_c = \{ 3, 4, 5, 6 \}$ $\quad$ $S_d = \{ 5 \}$

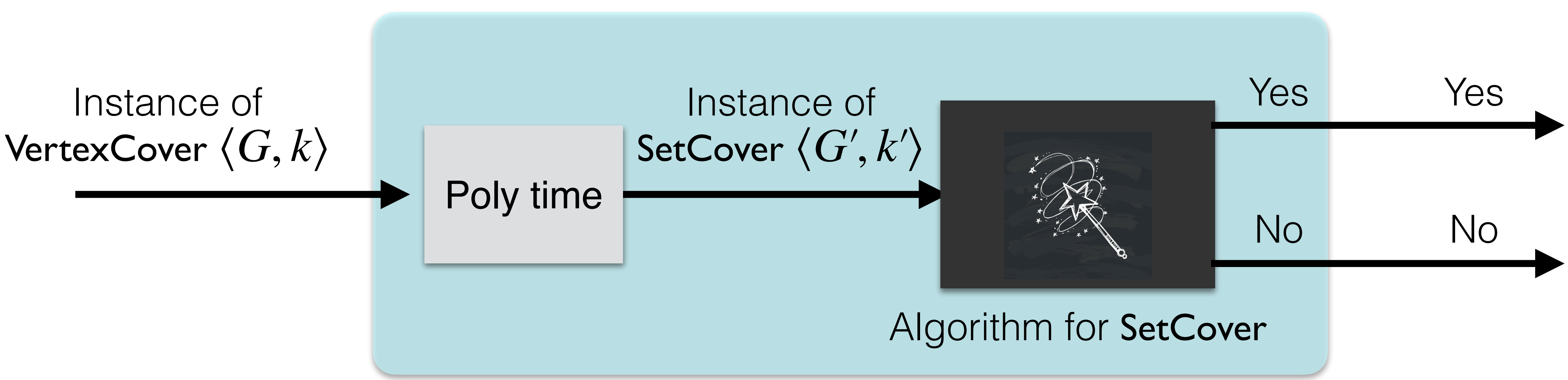
$S_e = \{ 1 \}$ $\quad$ $S_f = \{ 1, 2, 6, 7 \}$

$k = 2$

a set cover instance

Vertex Cover $\leq_p$ Set Cover

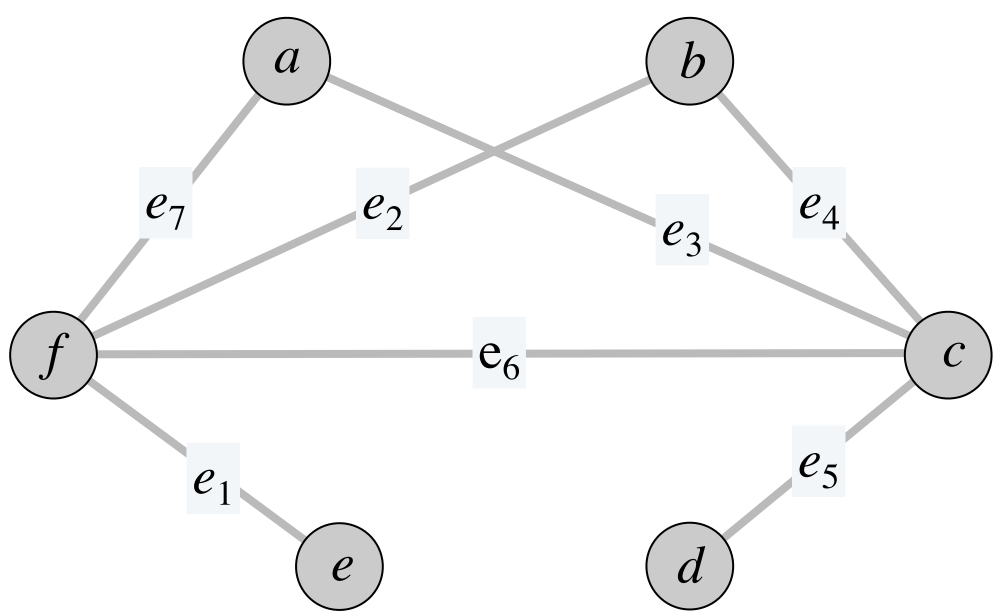
- **Theorem.** VERTEX-COVER $\leq_p$ SET-COVER
- **Proof.** Given instance $\langle G, k \rangle$ of vertex cover, construct an instance $\langle U, \mathcal{S}, k' \rangle$ of set cover problem such that
- G has a vertex cover of size at most k if and only if $\langle U, \mathcal{S}, k' \rangle$ has a set cover of size at most k.



Algorithm for VertexCover

Vertex Cover $\leq_p$ Set Cover

- **Theorem.** VERTEX-COVER $\leq_p$ SET-COVER
- **Proof.** Given instance $\langle G, k \rangle$ of vertex cover, construct an instance $\langle U, \mathcal{S}, k \rangle$ of set cover problem that has a set cover of size k iff G has a vertex cover of size k.
- **Reduction.** $U = E$, for each node $v \in V$, let $S_v = \{e \in E \mid e \text{ incident to } v\}$



vertex cover instance
(k = 2)

$U = \{ e_1, e_2, \ldots, e_7 \}$

$S_a = \{ e_3, e_7 \}$ $\quad$ $S_b = \{ e_2, e_4 \}$

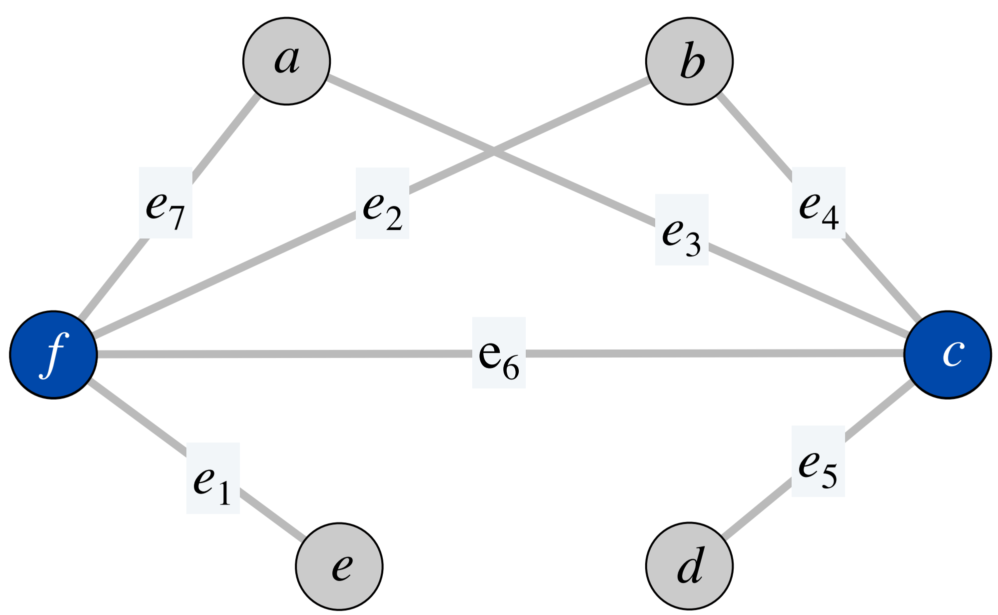
$S_c = \{ e_3, e_4, e_5, e_6 \}$ $\quad$ $S_d = \{ e_5 \}$

$S_e = \{ e_1 \}$ $\quad$ $S_f = \{ e_1, e_2, e_6, e_7 \}$

set cover instance
(k = 2)

Correctness

- **Claim.** ($\Rightarrow$) If G has a vertex cover of size at most k, then U can be covered using at most k subsets.
- **Proof.** Let $X \subseteq V$ be a vertex cover in G
  - Then, $Y = \{S_v \mid v \in X\}$ is a set cover of U of the same size



vertex cover instance
(k = 2)

$U = \{ e_1, e_2, \ldots, e_7\}$

$S_a = \{ e_3, e_7 \}$ $\quad$ $S_b = \{ e_2, e_4\}$

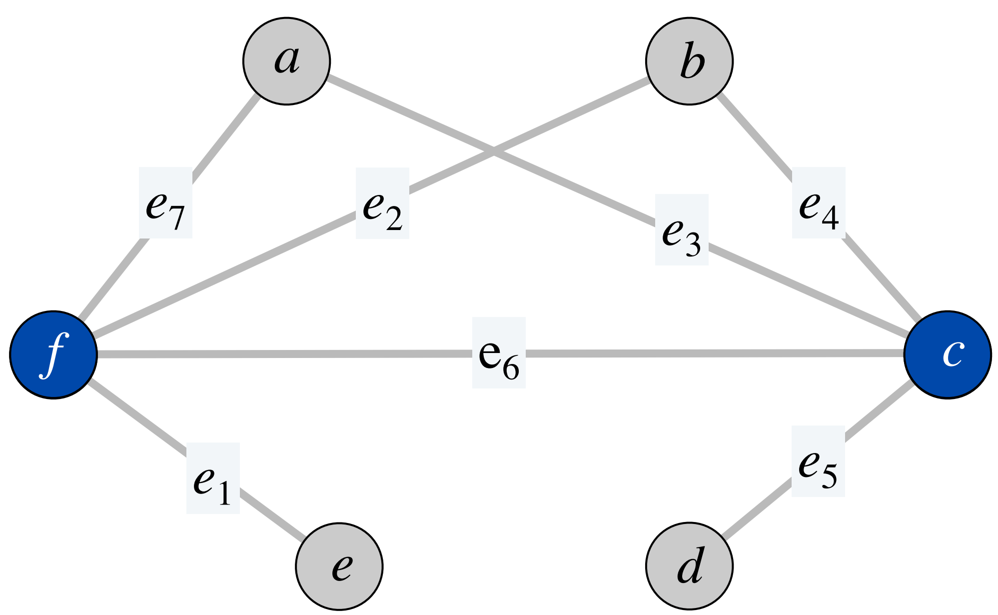
$S_c = \{ e_3, e_4, e_5, e_6\}$ $\quad$ $S_d = \{ e_5 \}$

$S_e = \{ e_1 \}$ $\quad$ $S_f = \{ e_1, e_2, e_6, e_7 \}$

set cover instance
(k = 2)

Correctness

- **Claim.** $(\Leftarrow)$ If U can be covered using at most k subsets then G has a vertex cover of size at most k.
- **Proof.** Let $Y \subseteq \mathcal{S}$ be a set cover of size k
  - Then, $X = \{v \mid S_v \in Y\}$ is a vertex cover of size k



vertex cover instance
(k = 2)

$U = \{ e_1, e_2, \ldots, e_7 \}$

$S_a = \{ e_3, e_7 \}$ $\quad$ $S_b = \{ e_2, e_4 \}$

$S_c = \{ e_3, e_4, e_5, e_6 \}$ $\quad$ $S_d = \{ e_5 \}$

$S_e = \{ e_1 \}$ $\quad$ $S_f = \{ e_1, e_2, e_6, e_7 \}$

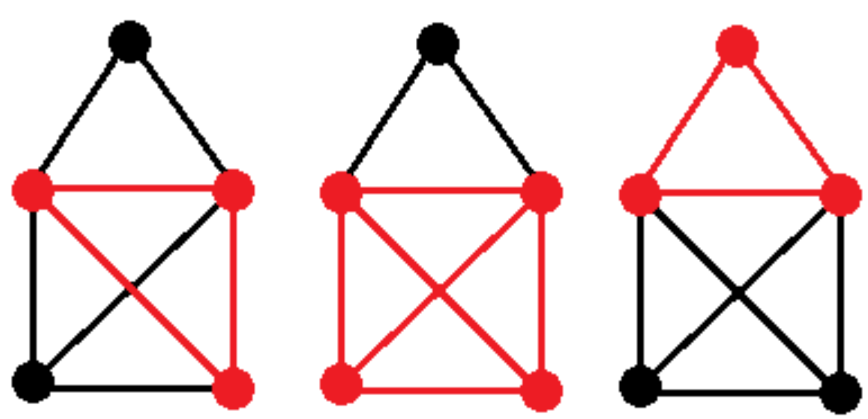
set cover instance
(k = 2)

Class Exercise

IND-SET $\leq_p$ Clique

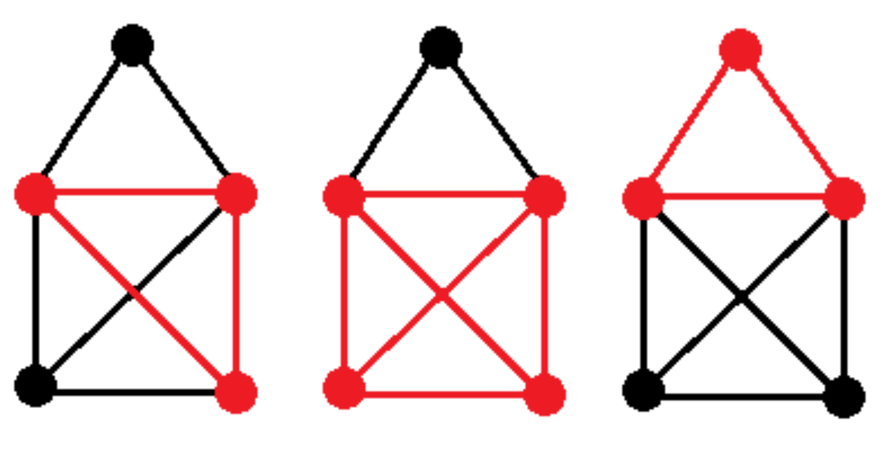
Clique

- 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?



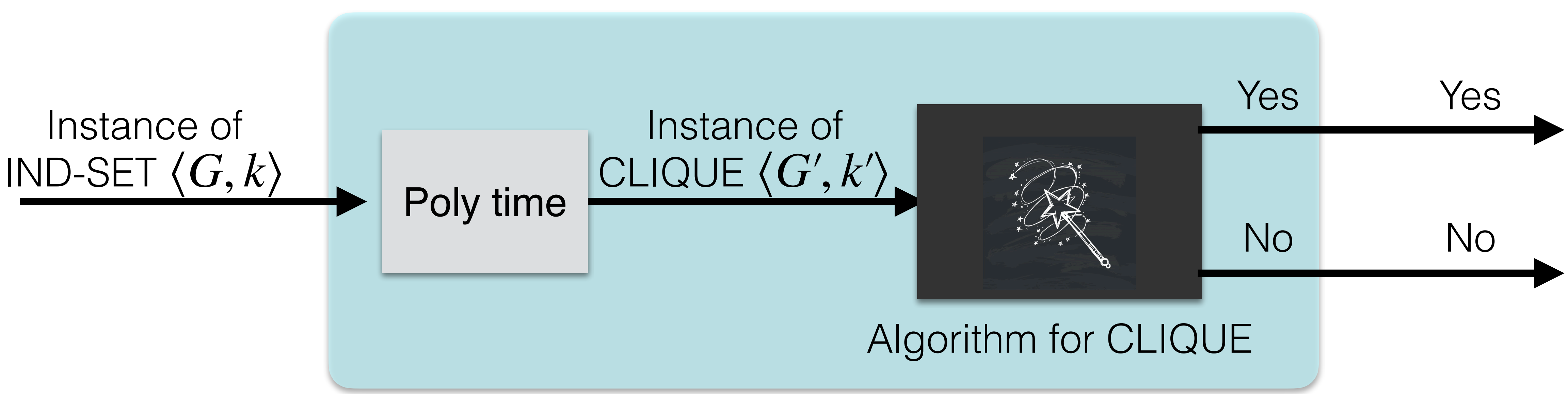
Clique

- 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?
- **CLIQUE** $\in$ NP
    - Certificate: a subset of vertices
    - Poly-time verifier: check is each pair of vertices have an edge between them and if size of subset is k



IND-SET to CLIQUE

- **Theorem.** **IND-SET $\leq_p$ CLIQUE.**
- **In class exercise.** 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'.



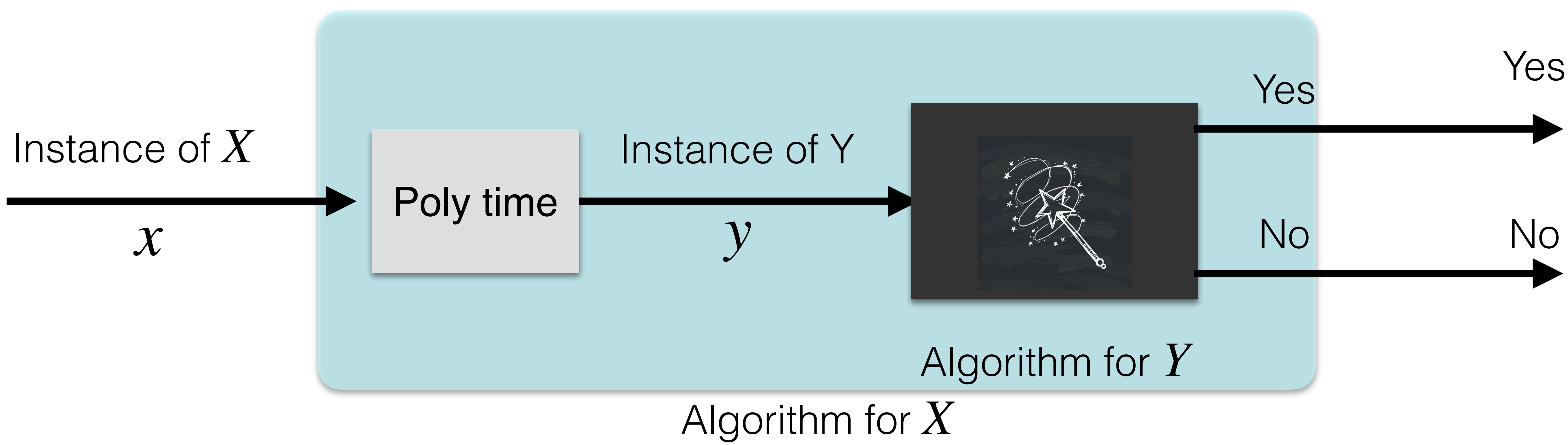
Algorithm for IND-SET

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 $e \notin E$ and $k' = k$
  - $(\Rightarrow)$ 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

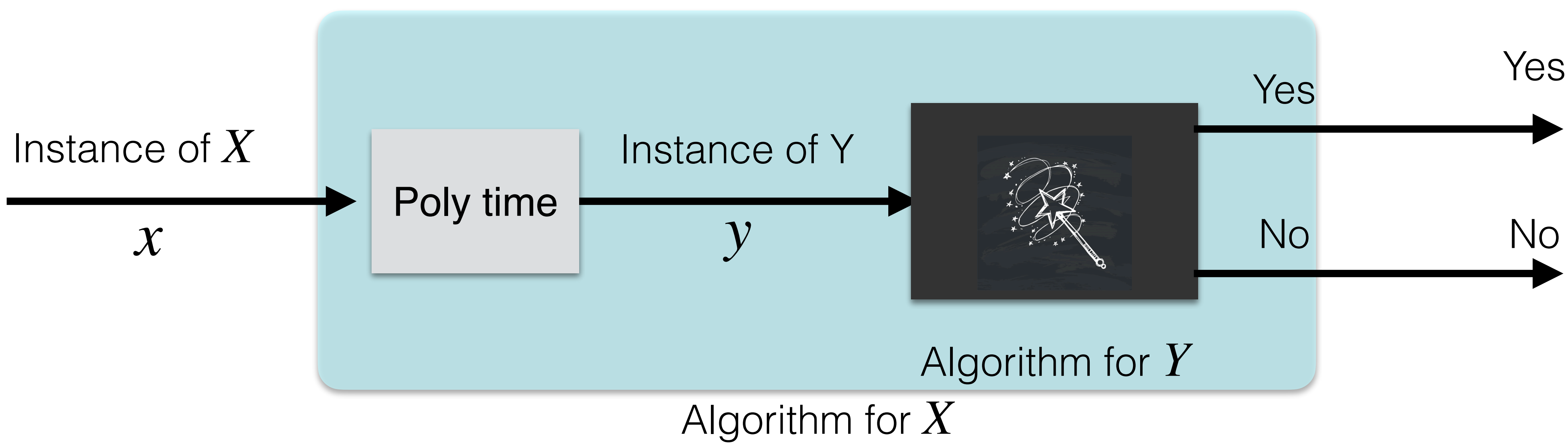
Reductions: General Pattern

- 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
  - If y is a “yes” instance of Y, then x is a “yes” instance of X
    $\Longleftrightarrow$ if x is a "no" instance of X, then y is a "no" instance of 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
- 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



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)