CSE 331: Algorithms & Complexity

"NP-Completeness"

Prof. Charlie Anne Carlson (She/Her)

Lecture 35 – 37

November 24th 2025, Dec 1st 2025, and Dec 3rd 2025

University at Buffalo

Schedule

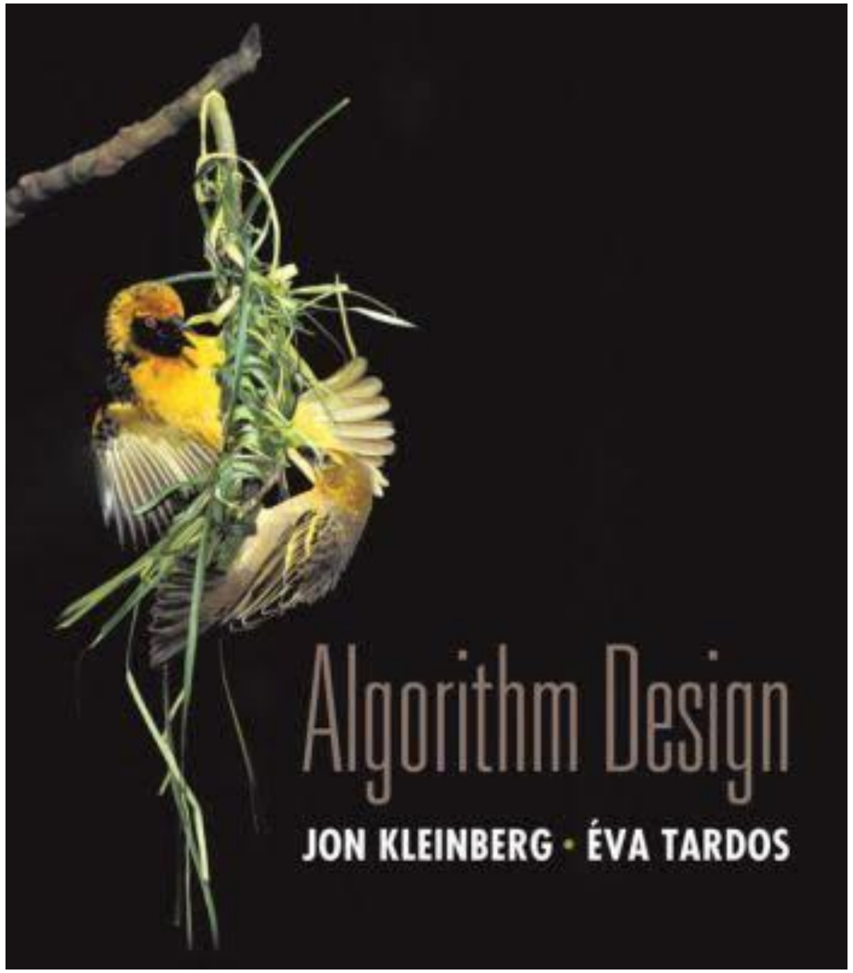
1. Course Updates
2. Recap
3. Hardness
4. Reductions
5. P vs NP
6. Completeness
7. NP-Complete Problems

Course Updates

- HW 8 Out
  - Due December 2nd
- Group Project (Autolab Up)
  - Groups Fixed
  - Reflections 3 Due December 3rd
- Quiz is Today
- Final on December 10th

Reading

- You should have read:
  - Finished 8.1
  - Finished 8.2
  - Finished 8.3
  - Finished 8.4
- Before Next Class:
  - Finish 8.7



Schedule

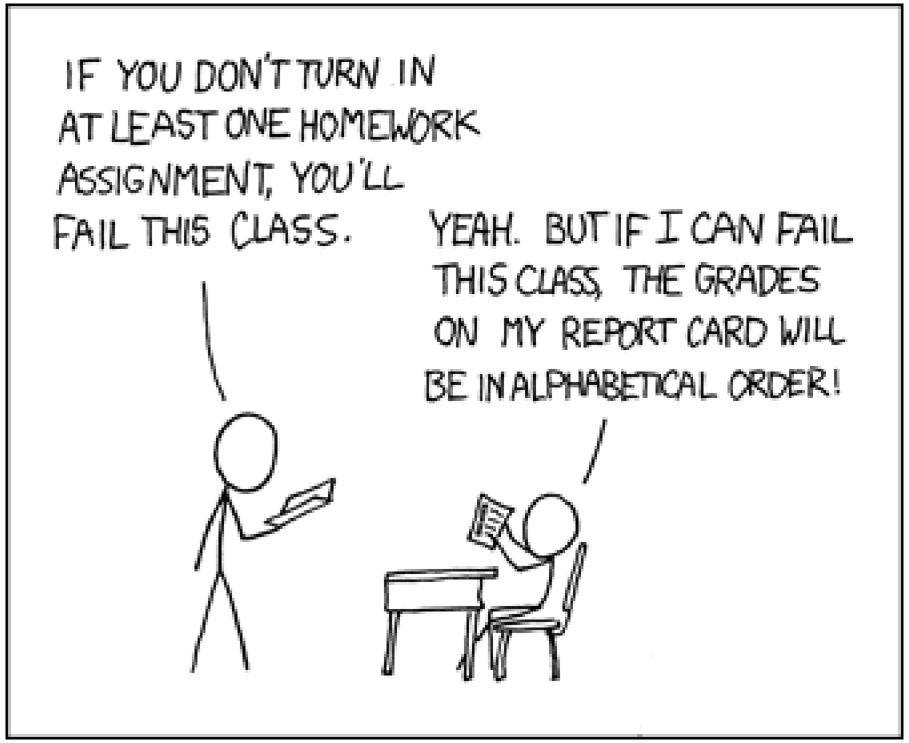
- ~~Monday Nov. 24th : P vs NP~~
- ~~Wednesday Nov. 26th : No Class~~
- ~~Friday Nov. 28th : No Class~~
- **Monday Dec. 1st : NP-Completeness**
- **Wednesday Dec. 3rd : Satisfiability**
- Friday Dec. 5th : Warpup
- Monday Dec. 8th : “Review”

Deadlines

- Monday Dec. 1st :
  - Quiz 2 (in class)
- Tuesday Dec. 2nd :
  - HW 8
    - Problem 3 -> December 4th
- Tuesday Dec. 3rd :
  - Project Reflection Problem 3
- Friday Dec. 7th :
  - Project Code Problem 4 & 5 (updated)
- Tuesday Dec. 9th :
  - Project Reflection Problem 4 & 5
  - Project Survey
- Wednesday Dec. 10th :
  - Final Exam

Goals:

- End of Week:
  - HW 7 Graded
  - HW 8 Solutions
  - Quiz 2 Solutions
- Before Next Week:
  - Project Reflection Problem 1 Graded
- Before End of Next Week:
  - HW 8 Graded
  - Quiz 2 Graded
  - Reflection Graded
- Before End of Following Week
  - Exam Graded
  - Final Grades Posted


IF YOU DON'T TURN IN AT LEAST ONE HOMEWORK ASSIGNMENT, YOU'LL FAIL THIS CLASS.
YEAH. BUT IF I CAN FAIL THIS CLASS, THE GRADES ON MY REPORT CARD WILL BE IN ALPHABETICAL ORDER!

https://xkcd.com/336/

Final

- If you need to miss the final, let me know ASAP because I am scheduling a time for people with valid excuses now.
- It will be 2 hours and 30 minutes
  - One part but both kinds of questions
  - Covers entire class
- You will get two review sheets!
- Bring ID and expected to be assigned a seat

Decision vs Optimization Problem

- We have seen a lot of optimization problems in this class that ask for things like shortest, longest, or maximum weight objects.
    - An optimization problem asks to find the "best" object in a set.
- We haven't talked about decision problems as much which ask if there exist an object with specific property.

Decision vs Optimization Problem

- Shortest Path Optimization Problem:
  - What is the shortest path between s and t?
- Shortest Path Decision Problem:
  - Given a weight W, does there exist a path of length at most W between s and t?
- **Question**: If you can solve one, can you solve the other?
  - **Answer**: Yes, check if shortest path is shortest or do a search.

Witness or Certificate

- We can **witness** or **certify** the answer to a decision problem by demonstrating the object provided one can check that it has the correct property efficiently.
  - Path of length at most W.
  - Path of length at least W.
  - Schedule with at least k jobs.
  - Points that are at most δ away from each other.

P and NP

- **P**: Decision problems for which there exist a polynomial time algorithm that solves it.
- **NP**: Decision problems for which there is always a polynomial time verifiable certificate for the solution.

P ? NP

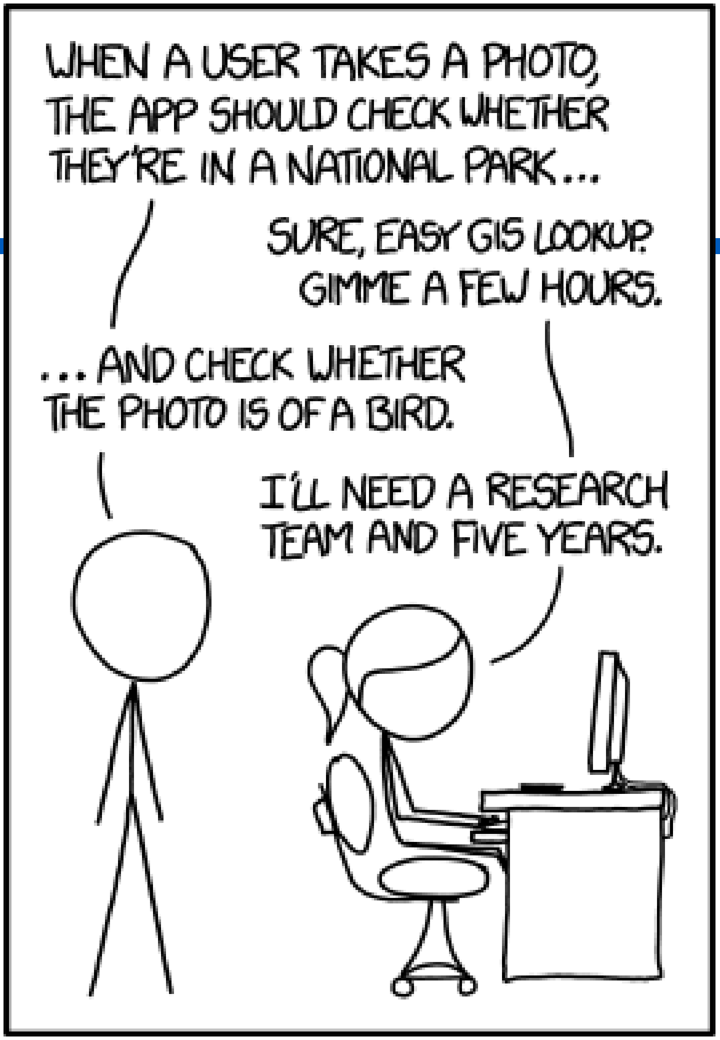
P vs NP

- Both P and NP are sets of problems (languages).
- We know that P is a subset of NP.
- We don’t know if P = NP and if you solve this, you get money, fame, and respect!

P vs NP

P vs NP

- **Question**: What might it help to know a problem is ”hard”?
  - **Answer**: We don’t want to waste time trying to solve problems that we know are very hard to solve.


WHEN A USER TAKES A PHOTO, THE APP SHOULD CHECK WHETHER THEY'RE IN A NATIONAL PARK...
SURE, EASY GIS LOOKUP. GIMME A FEW HOURS.
...AND CHECK WHETHER THE PHOTO IS OF A BIRD.
I'LL NEED A RESEARCH TEAM AND FIVE YEARS.

IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

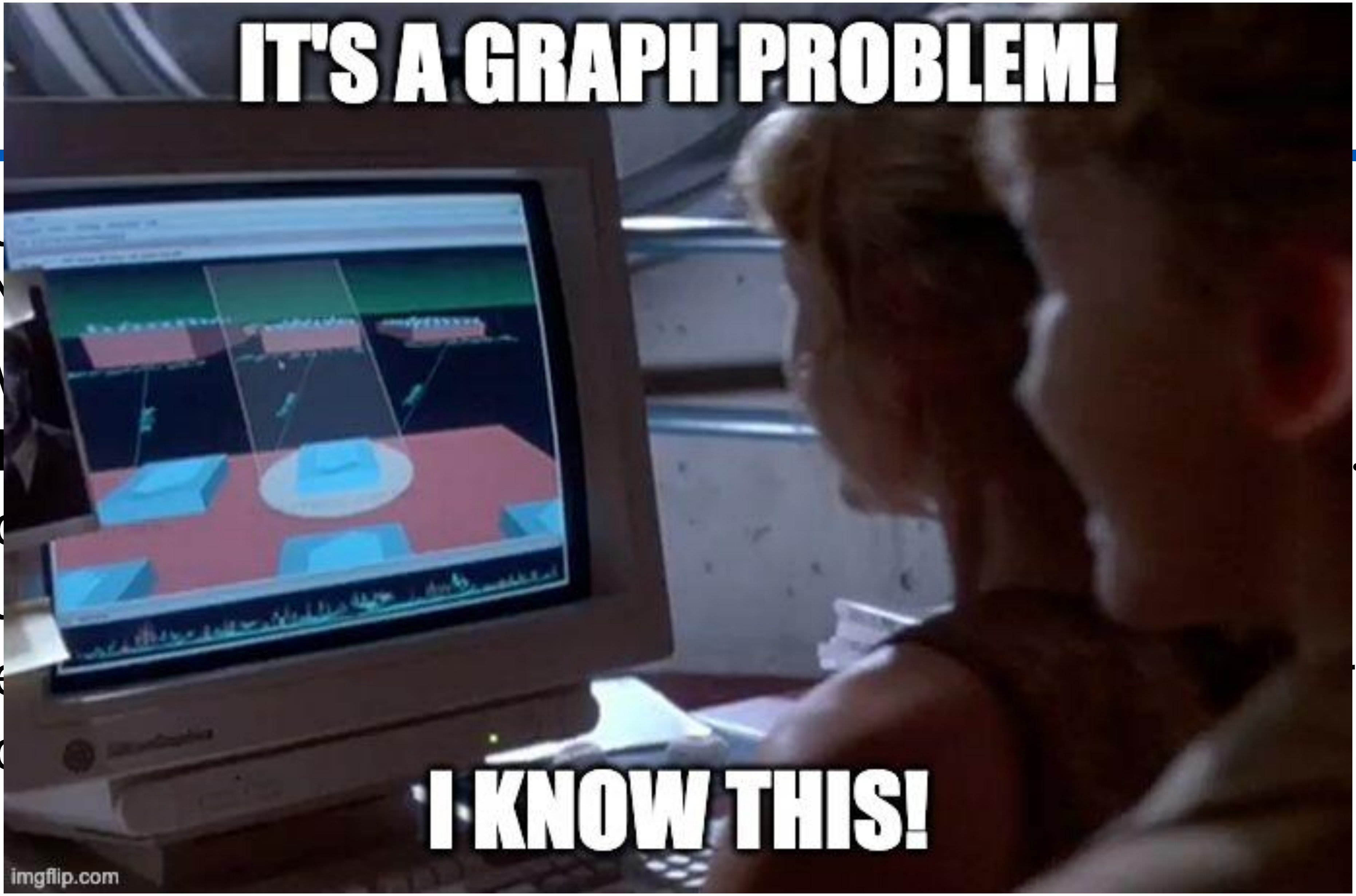
https://xkcd.com/1425/

Reductions Example

- You are given a network of computers to manage.
  - You are given a list of computers along with a list of which computer are directly connected to each.
  - Each connection has a cost for sending data over it.
- Each computer has a unique archive of information.
- Your boss wants you to figure out a way to find the cheapest way to send information from one computer to another upon request.

Redu

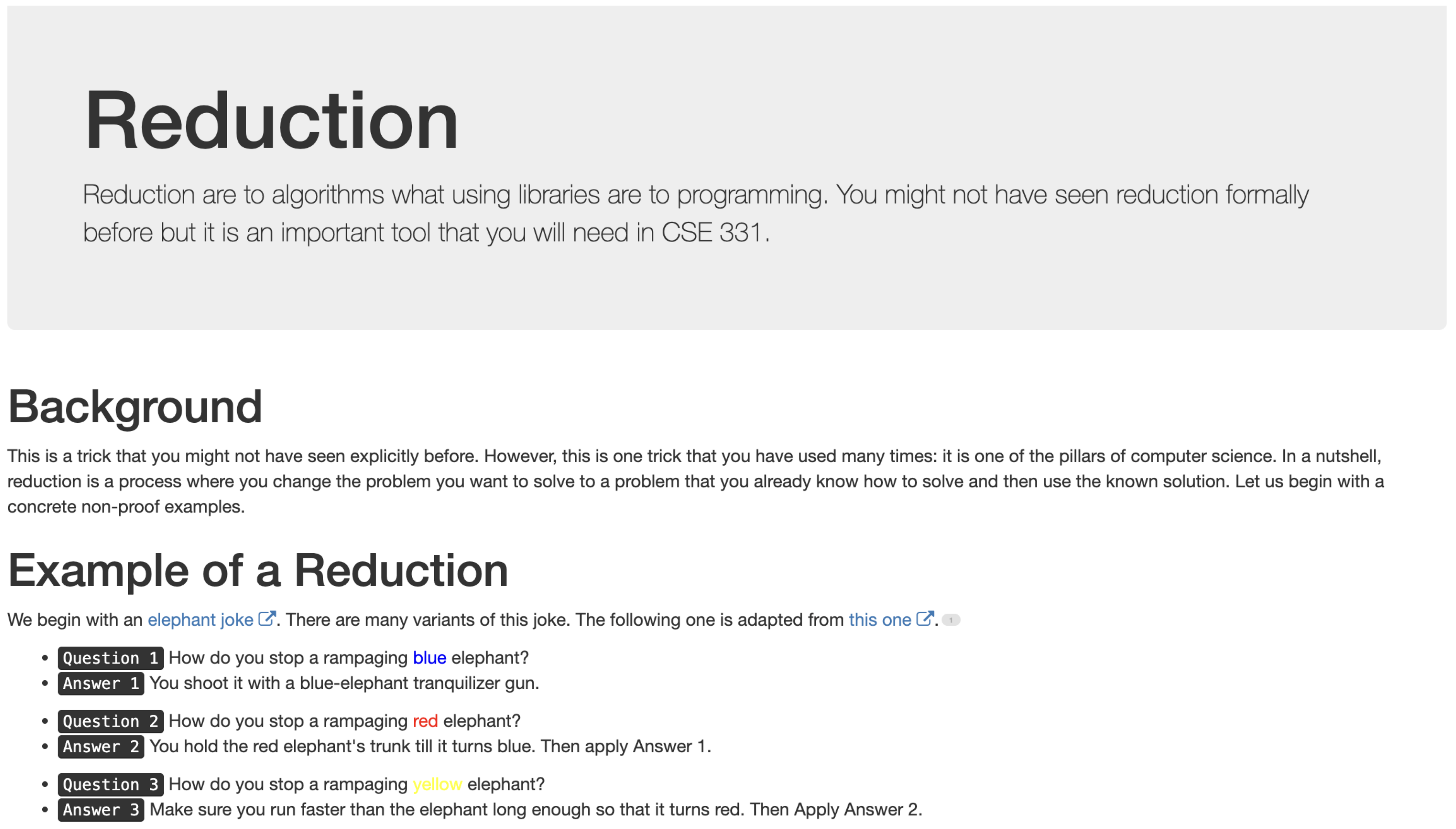
- You
  -
  - .
- Eac
- You
  che to
  anc



Reductions

- You can easily map this to a graph problem.
    - Each computer is a node in a graph where each direct connection is an edge.
    - The cost of sending data across connection is a weight for the edge.
    - Then finding the cheapest way to route data is just the shortest path problem.
        - I know an algorithm for shortest path in this case!

Reductions



Reduction

Reduction are to algorithms what using libraries are to programming. You might not have seen reduction formally before but it is an important tool that you will need in CSE 331.

Background

This is a trick that you might not have seen explicitly before. However, this is one trick that you have used many times: it is one of the pillars of computer science. In a nutshell, reduction is a process where you change the problem you want to solve to a problem that you already know how to solve and then use the known solution. Let us begin with a concrete non-proof examples.

Example of a Reduction

We begin with an elephant joke. There are many variants of this joke. The following one is adapted from this one.[1]

- `Question 1` How do you stop a rampaging blue elephant?
- `Answer 1` You shoot it with a blue-elephant tranquilizer gun.

- `Question 2` How do you stop a rampaging red elephant?
- `Answer 2` You hold the red elephant's trunk till it turns blue. Then apply Answer 1.

- `Question 3` How do you stop a rampaging yellow elephant?
- `Answer 3` Make sure you run faster than the elephant long enough so that it turns red. Then Apply Answer 2.

Reasons to use Reduction I

- You want to show something is easy/possible:
    - You have a real-world problem, and you want to reduce it to a simple graph problem you know how to solve.
    - You have a new graph problem that seems harder but isn’t.
    - You may also want to show how to use already existing algorithms for none graph problems.

Reasons to use Reduction II

- You want to show something is hard/impossible:
  - Reductions give you a formal way to show that a problem may not be possible to solve.
  - We may not have a proof that a problem is actually hard, but we do know that people have been trying for a long time to solve it with no success.
    - If you have a new problem, you don’t want to have to fail to solve it for years to justify not being able to solve it. Instead, you reduce.

Oracles

- Consider a problem X
  - E.g. Shortest path, longest path, stable matching, coloring, sorting, etc.
- Note that X has some input and desired output
  - E.g. Graph + Weights => Shortest Path
- An oracle is a black box that can solve X on any input.

Reduction

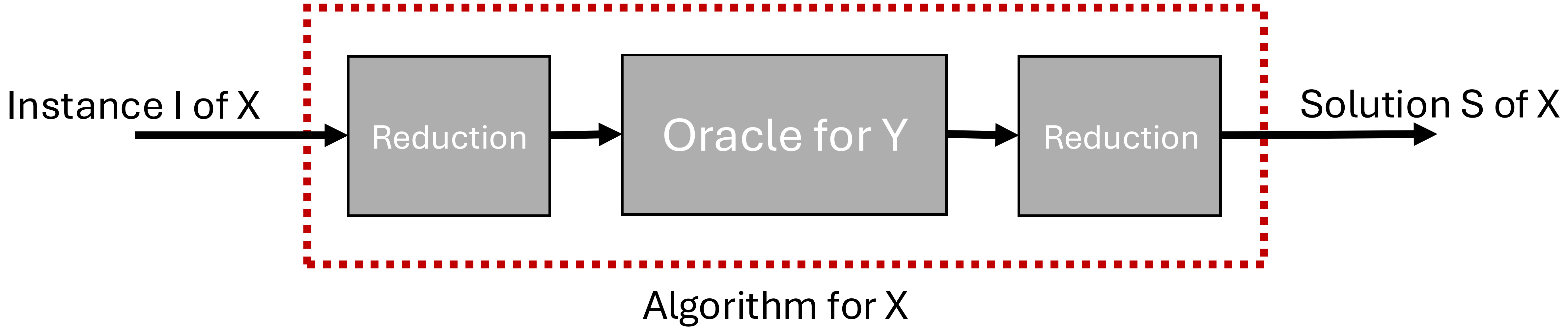
- Consider problem Y that we assume can be solved in poly time.
- **Question**: How can we use this assumption about Y to show that another problem is easy?

Reduction

- Problem X polynomial-time (cook) reduces to problem Y if arbitrary instances of problem X can be solved using:
    - Polynomial number of computational steps, plus
    - Polynomial number of calls to an oracle that solves problem Y.

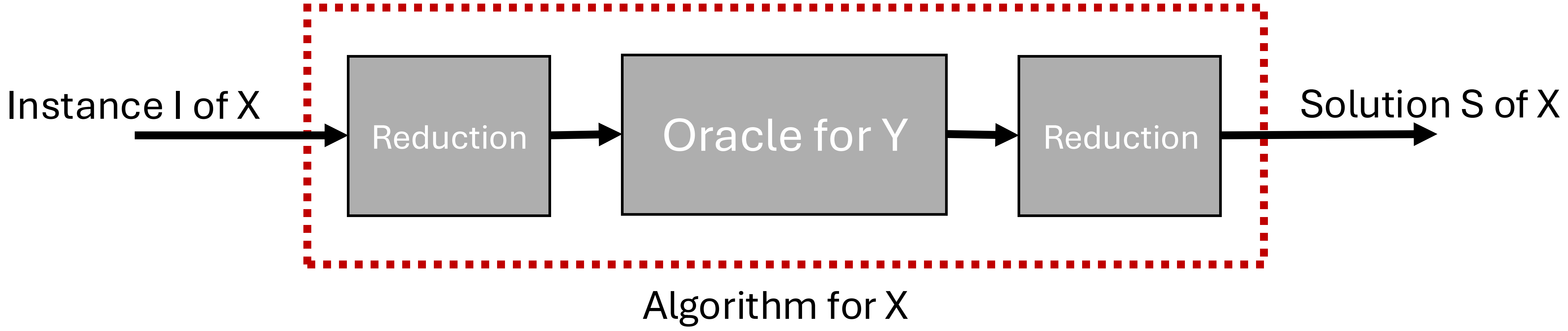
Reduction

- Problem X polynomial-time (cook) reduces to problem Y if arbitrary instances of problem X can be solved using:
  - Polynomial number of computational steps, plus
  - Polynomial number of calls to an oracle that solves problem Y.



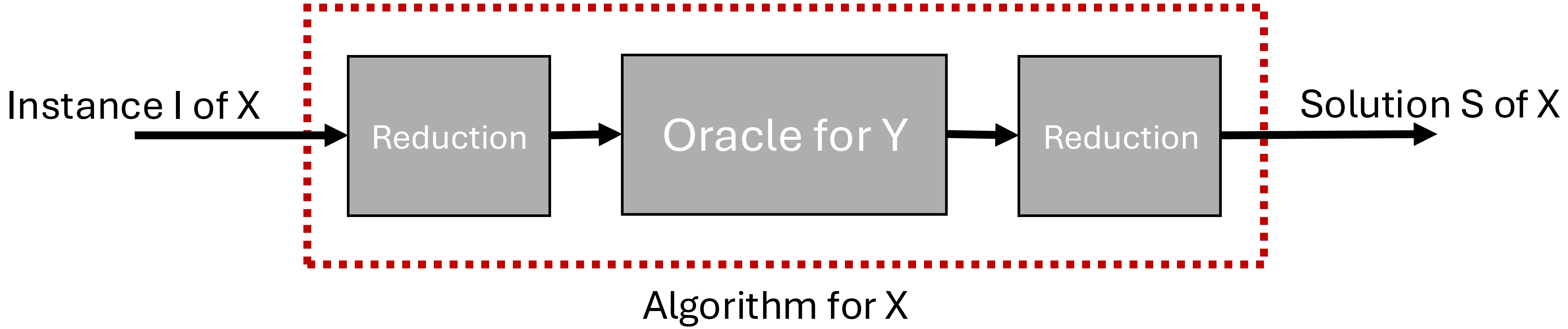
Reduction

- We want both parts of the reduction to run in polynomial time.
    - First part converts instance I of X to an instance J of Y.
    - Second part converts solution T of Y to a solution S of X.



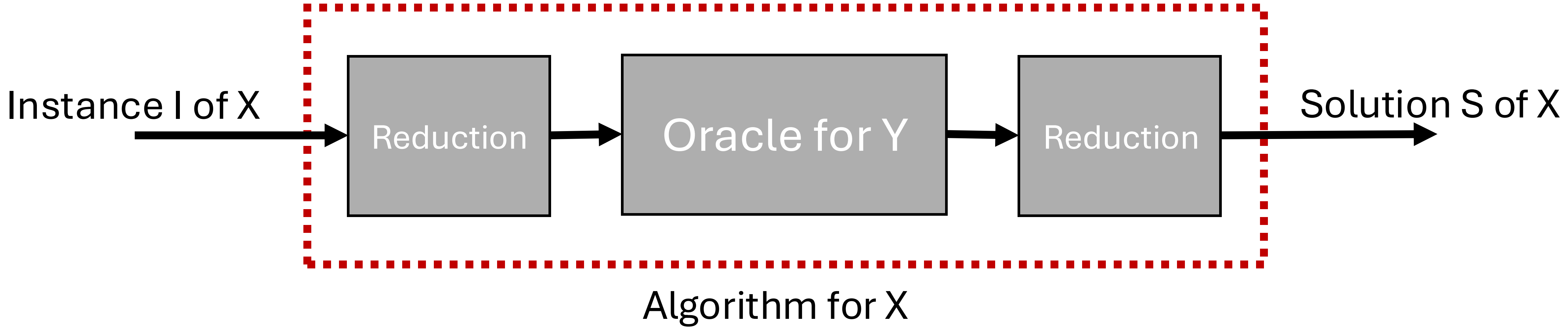
Reduction ($X \leq_p Y$)

- If we can prove that both parts of the reduction run in polynomial time, then if there is a polynomial time algorithm for Y there is a polynomial time algorithm for X.
- "Problem X is at most as hard as Y with respect to poly-time."



Reduction ($X \leq_p Y$)

- If we know there is no efficient algorithm for X, then it follows that there can't be an efficient algorithm for Y.
- "Problem X is at most as hard as Y with respect to poly-time."

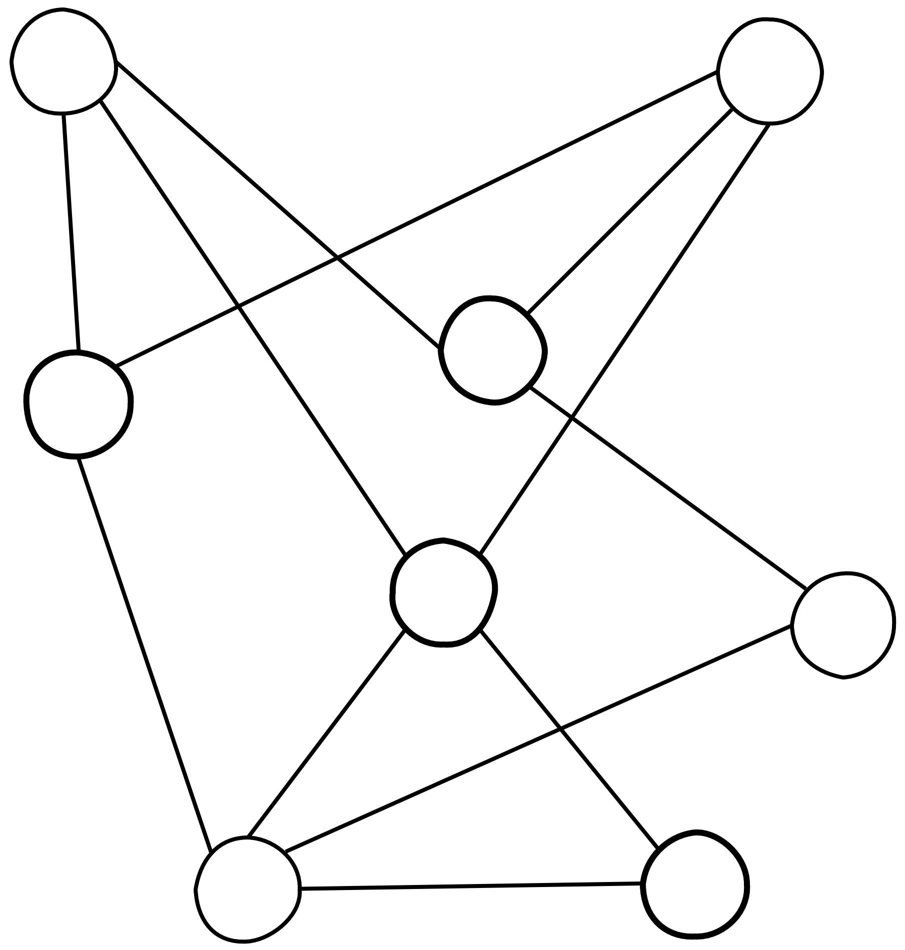


Independent Set

- Fix a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$.
- A set $S \subseteq \mathrm{V}$ is independent if no two nodes in S are joined by an edge in E.

Independent Set Problem:

- **Input**: A graph G and integer k.
- **Output**: If there exist an independent set of size at least k in G.



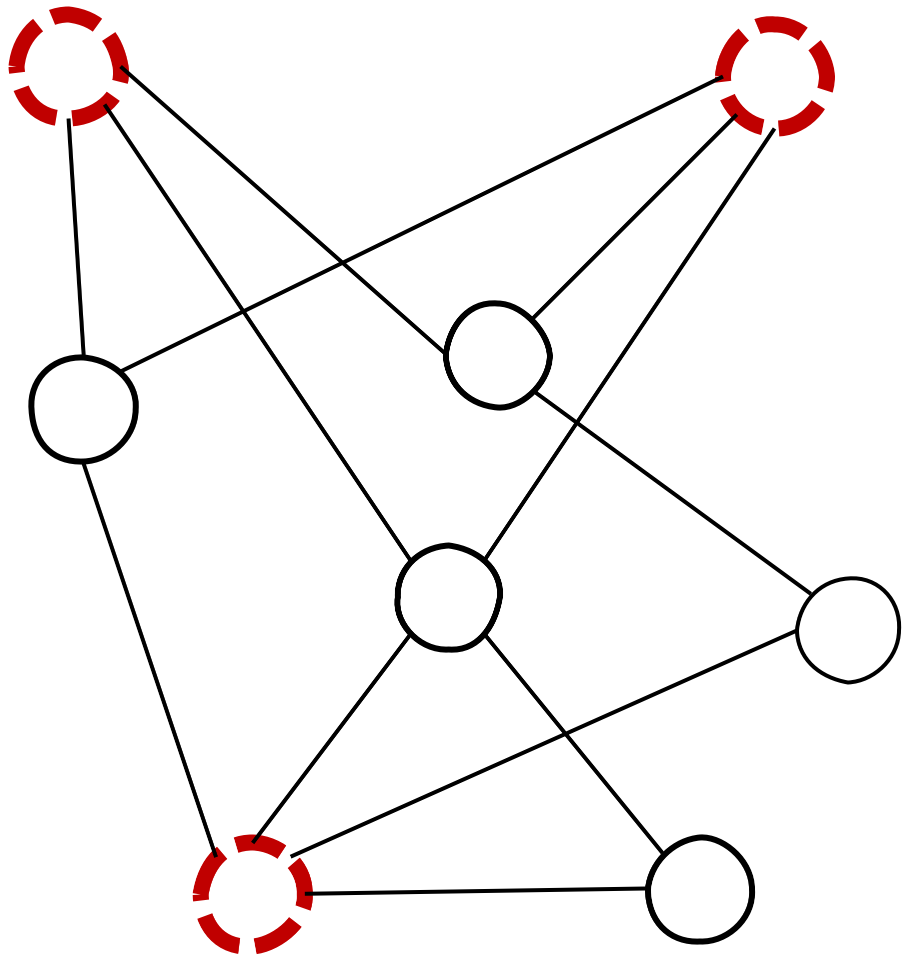
Independent Set

◌ In Independent Set

- Fix a graph $G = (V, E)$.
- A set $S \subseteq V$ is independent if no two nodes in S are joined by an edge in E.

Independent Set Problem:

- **Input**: A graph G and integer k.
- **Output**: If there exist an independent set of size at least k in G.



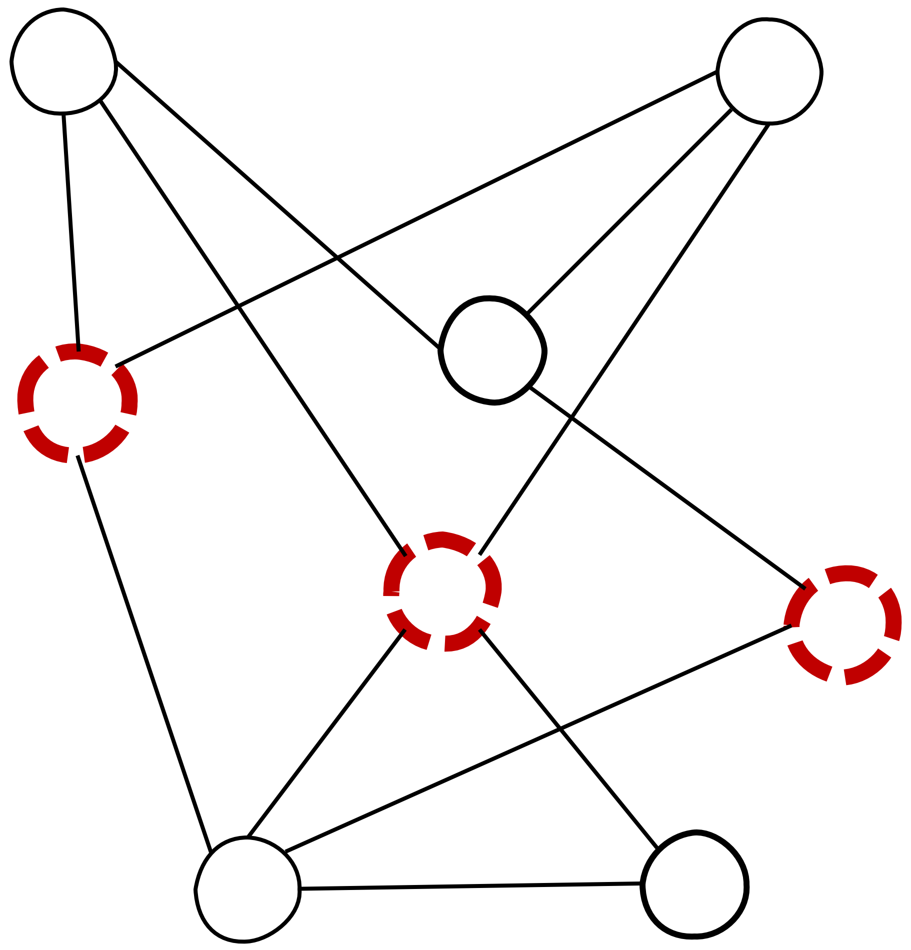
Independent Set

◌ In Independent Set

- Fix a graph $G = (V, E)$.
- A set $S \subseteq V$ is independent if no two nodes in S are joined by an edge in E.

Independent Set Problem:

- **Input**: A graph G and integer k.
- **Output**: If there exist an independent set of size at least k in G.

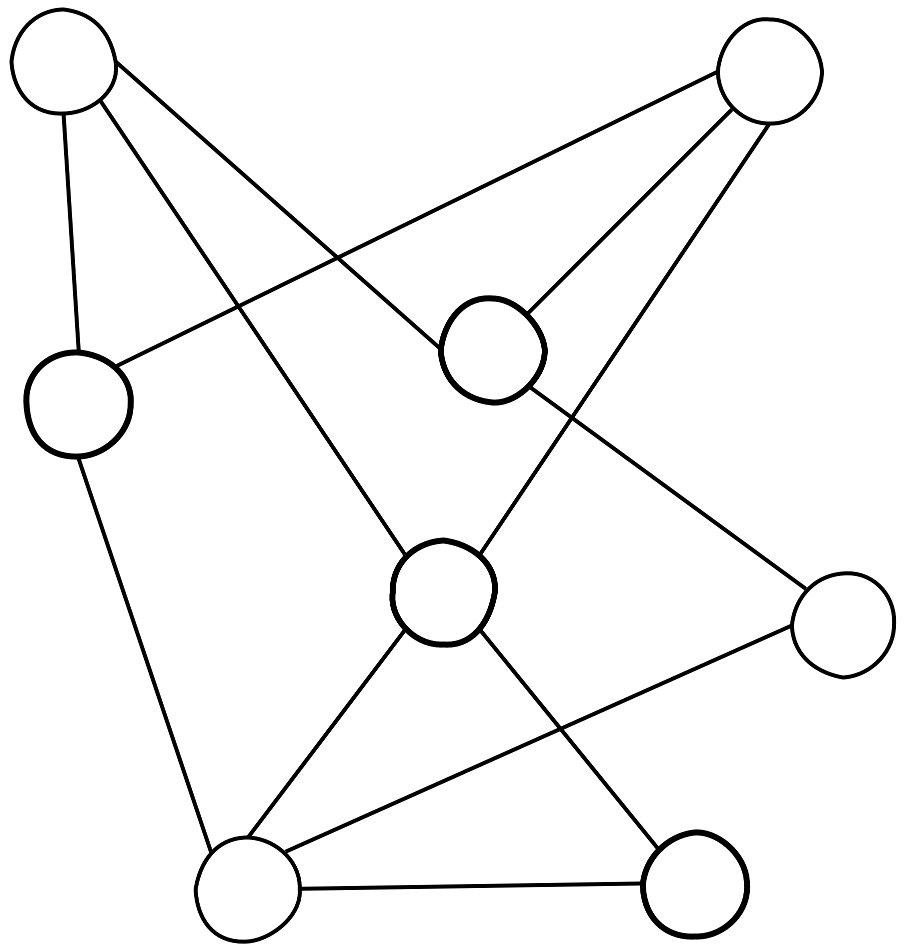


Vertex Cover

- Fix a graph $G = (V, E)$.
- A set $S \subseteq V$ is a vertex cover if for every edge $e \in E$, at least one endpoint is in S.

Vertex Cover Problem:

- **Input**: A graph G and integer k.
- **Output**: If there exist a vertex cover size at most k in G.



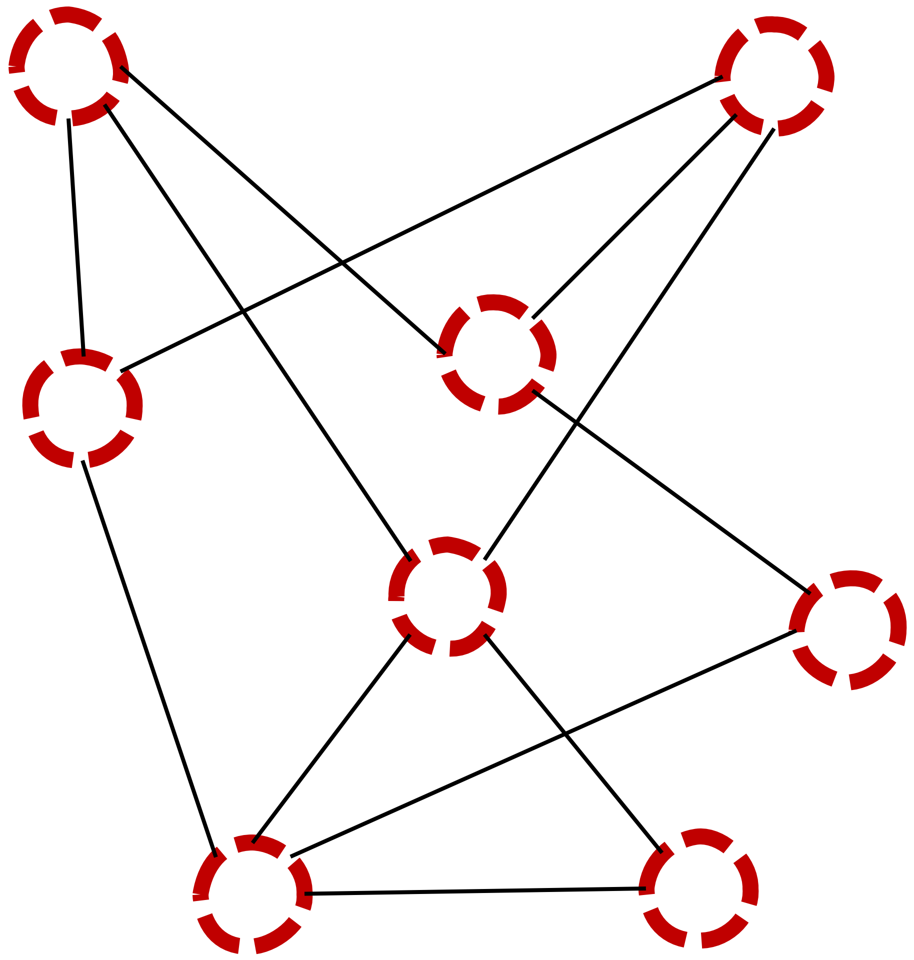
Vertex Cover

◌ In Vertex Cover

- Fix a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$.
- A set $S \subseteq \mathrm{V}$ is independent if no two nodes in S are joined by an edge in E.

Independent Set Problem:

- **Input**: A graph G and integer k.
- **Output**: If there exist an independent set of size at least k in G.



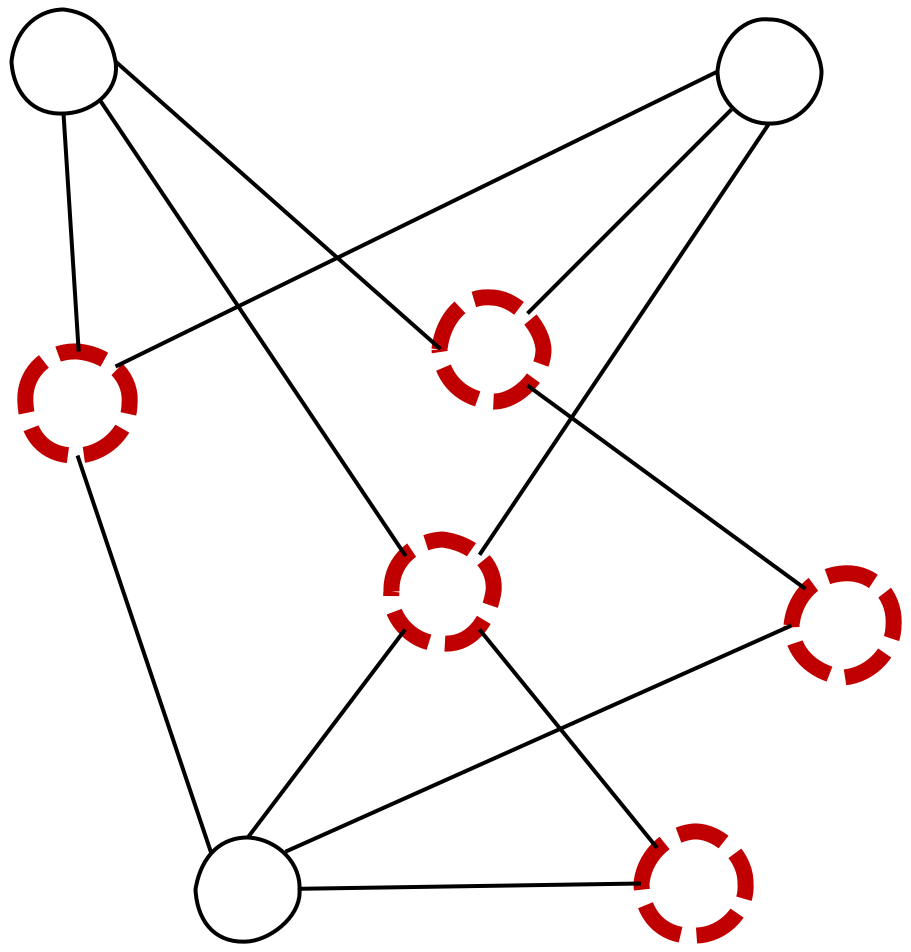
Vertex Cover

◌ In Vertex Cover

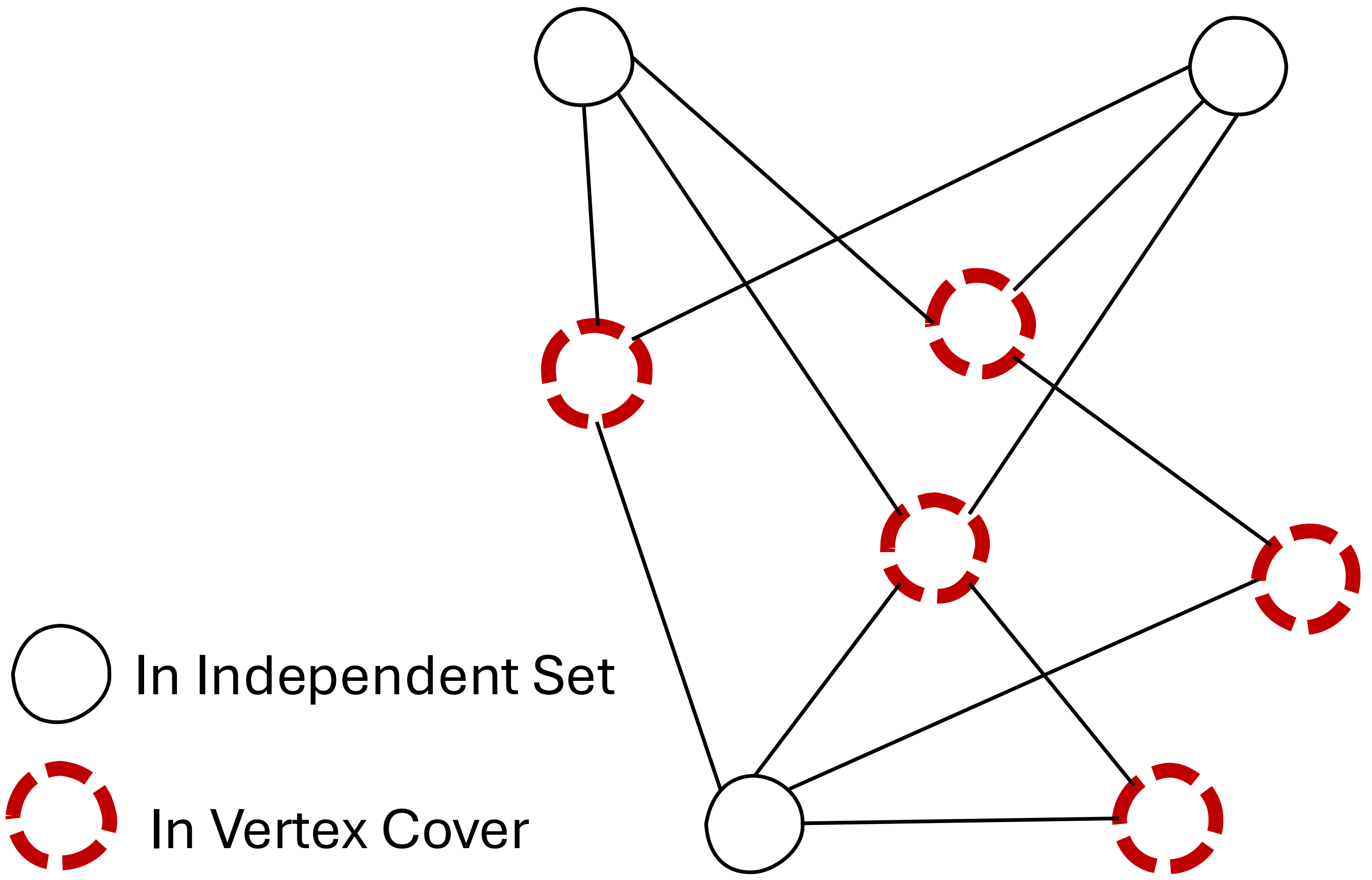
- Fix a graph $G = (V, E)$.
- A set $S \subseteq V$ is independent if no two nodes in S are joined by an edge in E.

Independent Set Problem:

- **Input**: A graph G and integer k.
- **Output**: If there exist an independent set of size at least k in G.

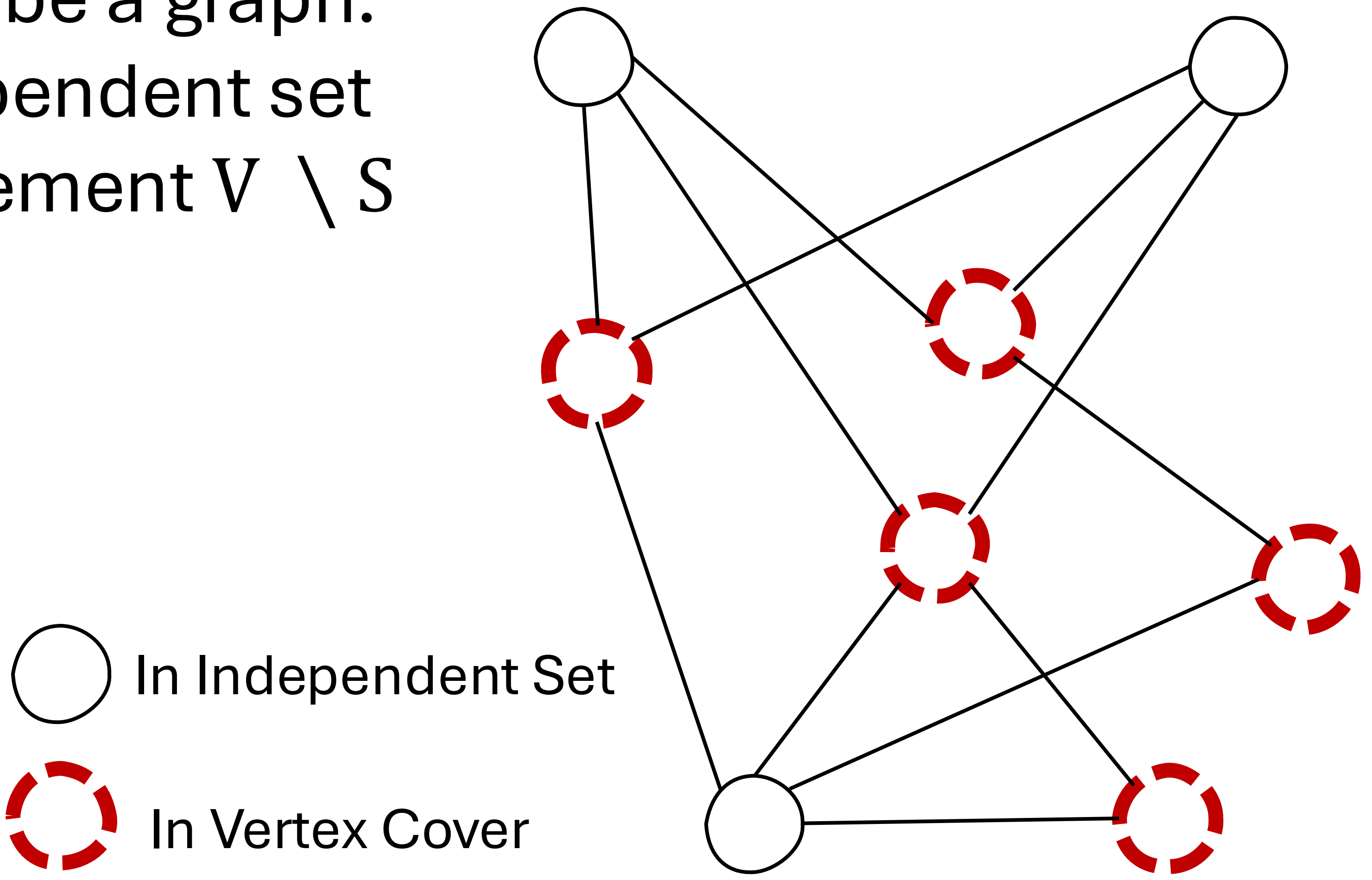


Independent Set vs Vertex Cover



Independent Set vs Vertex Cover

Claim: Let $G = (V, E)$ be a graph. Then $S \subseteq V$ is an independent set if and only if its complement $V \setminus S$ is a vertex cover.



Proof Part (=>)

Claim: Let $G = (V, E)$ be a graph. Then $S \subseteq V$ is an independent set if and only if its complement $V \setminus S$ is a vertex cover.

Proof (= >):

- Consider an independent set S and an edge $e \in E$. Since S is an independent set, it must be the case that one endpoint of e is not in it and thus at least one endpoint of e is in $V \setminus S$.

Proof Part (=>)

Claim: Let $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ be a graph. Then $S \subseteq \mathrm{V}$ is an independent set if and only if its complement $\mathrm{V} \setminus S$ is a vertex cover.

Proof (< =):

- Consider a set S such that $\mathrm{V} \setminus S$ is a vertex cover and an edge $\mathrm{e} \in \mathrm{E}$. If both endpoints of e where in S, then $\mathrm{V} \setminus S$ would not be a vertex cover (contradiction). Hence, S must be an independent set.

Independent Set to Vertex Cover

Claim: Independent Set $\leq_p$ Vertex Cover.

Independent Set to Vertex Cover

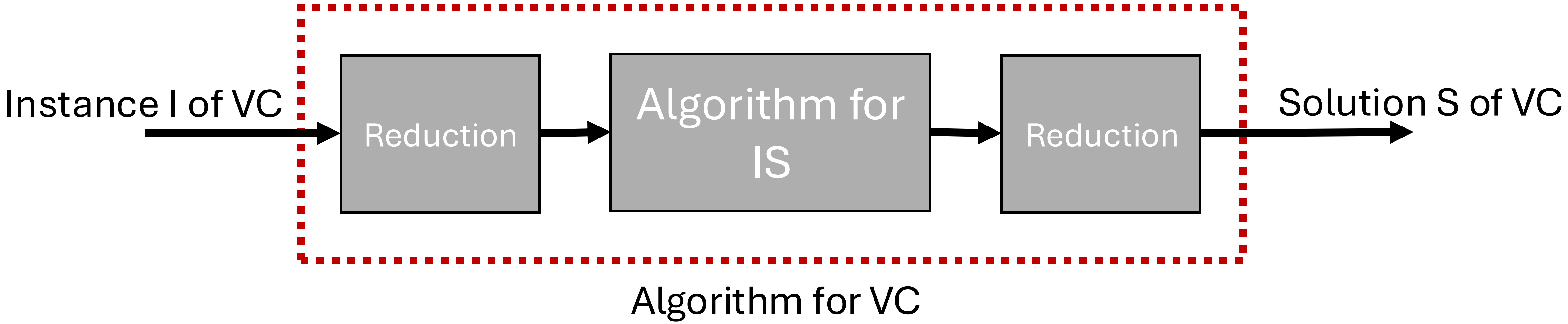
Claim: Independent Set $\leq_p$ Vertex Cover.

Proof: Suppose we have a black box that solves vertex cover problem efficiently. Then we can decide whether G has an independent set of size at least k by asking the black box if G has a vertex cover of size at most n-k.

Note that this reduction is trivial as we are using the same G and we can compute n-k in O(1) time.

Reduction (Ind Set $\leq_p$ Vertex Cover)

- If we know there is no efficient algorithm for Independent Set, then it follows that there can't be an efficient algorithm for Vertex Cover.



Independent Set from Vertex Cover

Claim: Vertex Cover $\leq_p$ Independent Set.

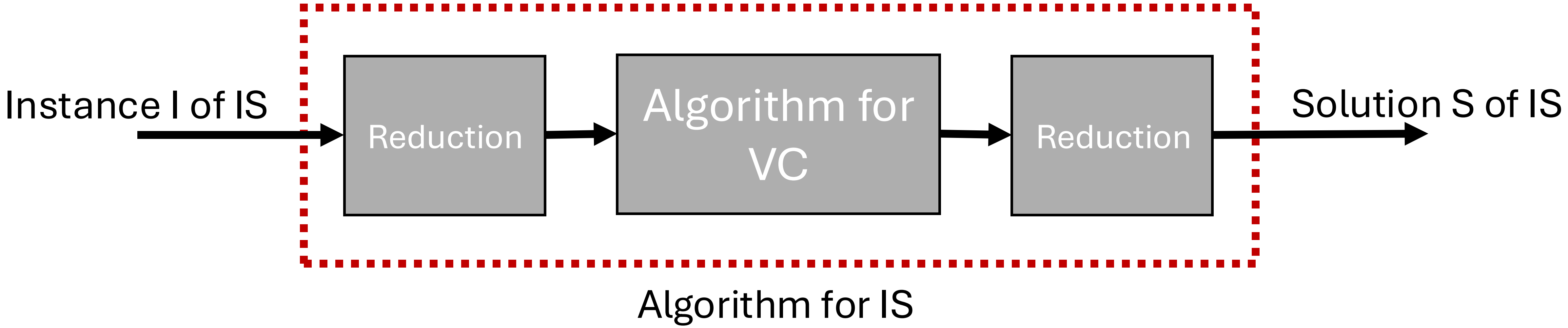
Independent Set from Vertex Cover

Claim: Vertex Cover $\leq_p$ Independent Set.

Proof: Suppose we have a black box that solves independent set problem efficiently. Then we can decide whether G has a vertex cover of size at most k by asking the black box if G has an independent set of size at least n-k.

Reduction (Vertex Cover $\leq_p$ Ind. Set)

- If we know there is no efficient algorithm for Vertex Cover, then it follows that there can't be an efficient algorithm for Independent Set.



Independent Set & Vertex Cover

Claim: Independent Set $\leq_p$ Vertex Cover.

Claim: Vertex Cover $\leq_p$ Independent Set.

Observation: The above claims tell us that up to our notion of efficiency, these two problems are equally “hard”. That is, if you can solve one, you can solve the other.

NP-hard & NP-complete

Definition: We say that a problem X is **NP-hard** if for every problem Y in NP, there is a polynomial-time reduction from Y to X ($\mathrm{Y} \leq_p \mathrm{X}$).

Definition: We say that a problem X is **NP-complete** if it is in NP and if it is NP-hard.

Observation: NP-Complete problems represent the ”hardest” problems in NP.

Boolean Formula

- Suppose you have a set X of n Boolean variables $x_1, x_2, \dots, x_n$ that can each take value 0 or 1.
- A “term” over X is the variable x_i or its negation $\overline{x}_i$.
- A “clause” is a simple disjunction of distinct terms

$$t_1 \vee t_2 \vee t_3 \vee \cdots \vee t_\ell$$

where each term $t_i \in \{x_1, x_2, \dots, x_n, \overline{x}_1, \overline{x}_2, \dots, \overline{x}_n\}$.

E.g.

$$(x_1 \vee x_4 \vee \overline{x}_5)$$

Boolean Formula

- We say that a an "assignment" of values to the variables in X satisfy a clause if it cause it to evaluate to TRUE.
- An assignment satisfies a collection of clauses $C_1, C_2, \dots, C_m$ if and only if it causes the conjunction
$$C_1 \wedge C_2 \wedge \cdots \wedge C_m$$
to evaluate to TRUE.

E.g.
$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4 \vee \overline{x}_5)$$

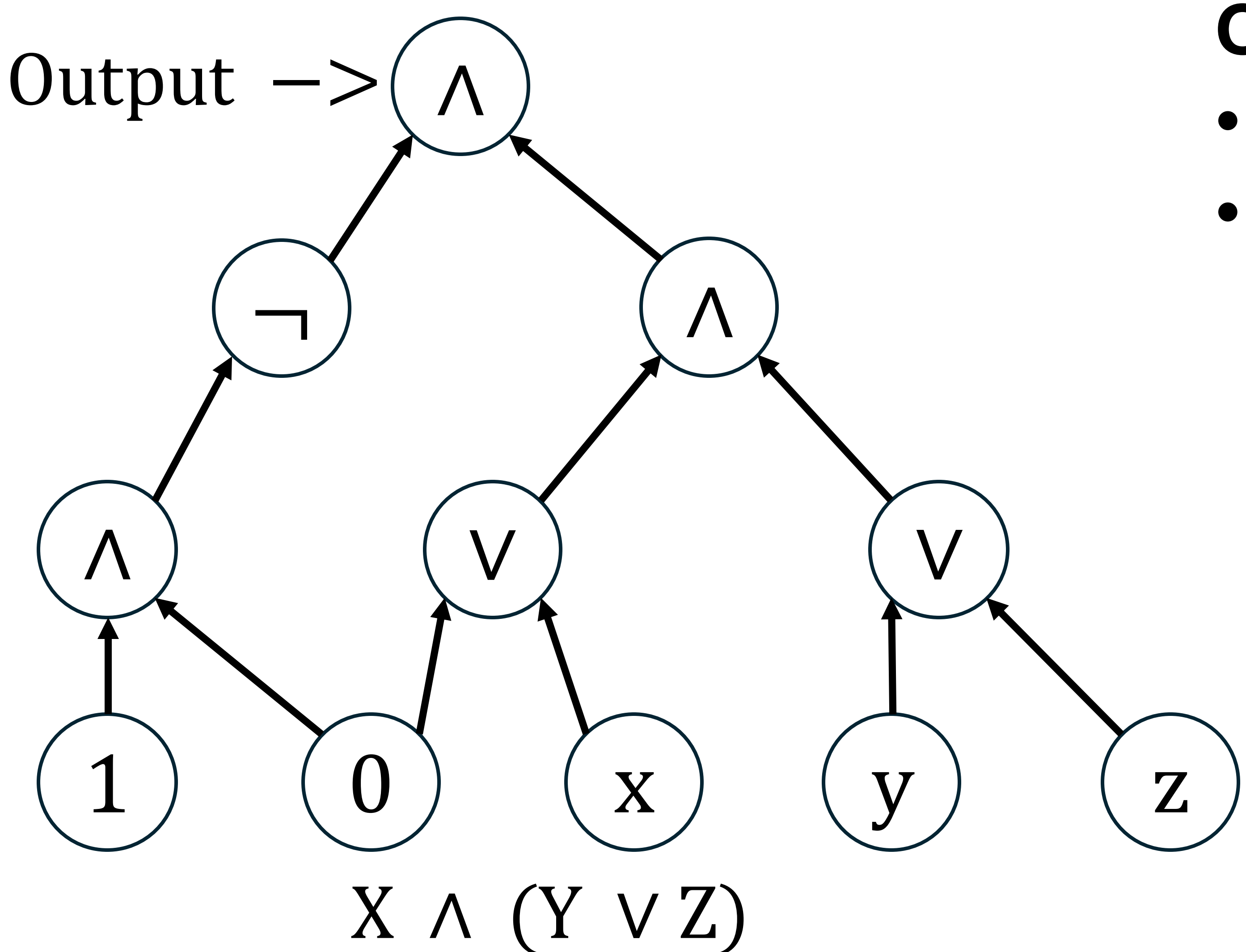
SAT Problems

- **SAT:**
  - **Input**: A set of clauses $C_1, C_2, \dots, C_m$ over a set of variables $x_1, x_2, \dots, x_n$.
  - **Output**: If there exist an assignment to the variables such that each clause is satisfied.
- **3-SAT:**
  - **Input**: A set of clauses $C_1, C_2, \dots, C_m$ each of size 3 over a set of variables $x_1, x_2, \dots, x_n$.
  - **Output**: If there exist an assignment to the variables such that each clause is satisfied.

Circuit:

- A circuit is a labeled directed acyclic graph K such that
  - All sources (no incoming edges) are labeled with 0, 1, or a variable. **<-inputs**
  - All other nodes are labeled with AND, OR, or NOT.
    - Each AND and OR node has two incoming edges
    - Each NOT node has one incoming edge
  - There is a single sink (no outgoing edges). **<- output**

Circuit SAT



Circuit Sat Problem (SAT):

- **Input**: A circuit K
- **Output**: If there exist an assignment to the variables that makes that satisfies the circuit.

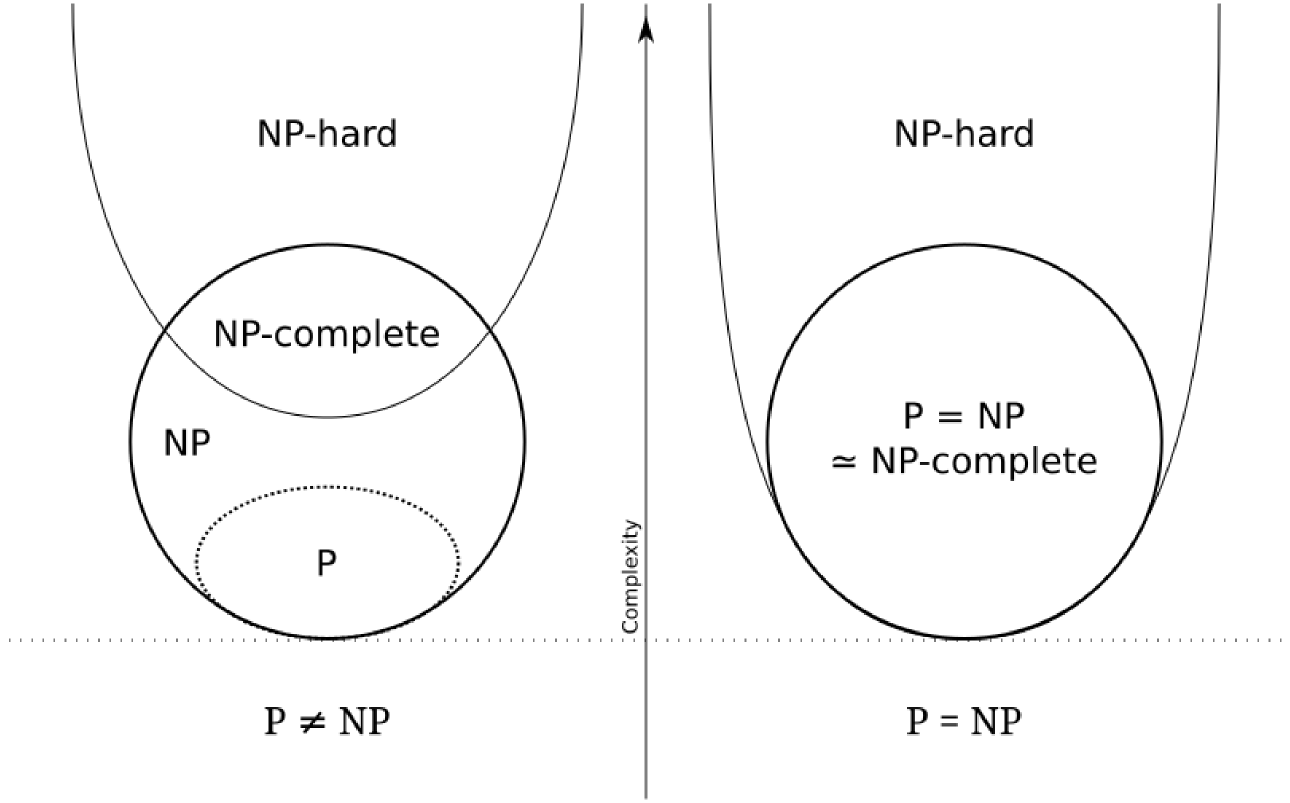
Circuit SAT

Claim: Circuit Satisfiability is NP-complete.

- Not hard to show NP-hard.
- To show NP-Complete, you have to reduce from an arbitrary NP-hard problem.
  - We wont talk about proof in this class, but the idea is that you can represent any verification algorithm as a circuit!

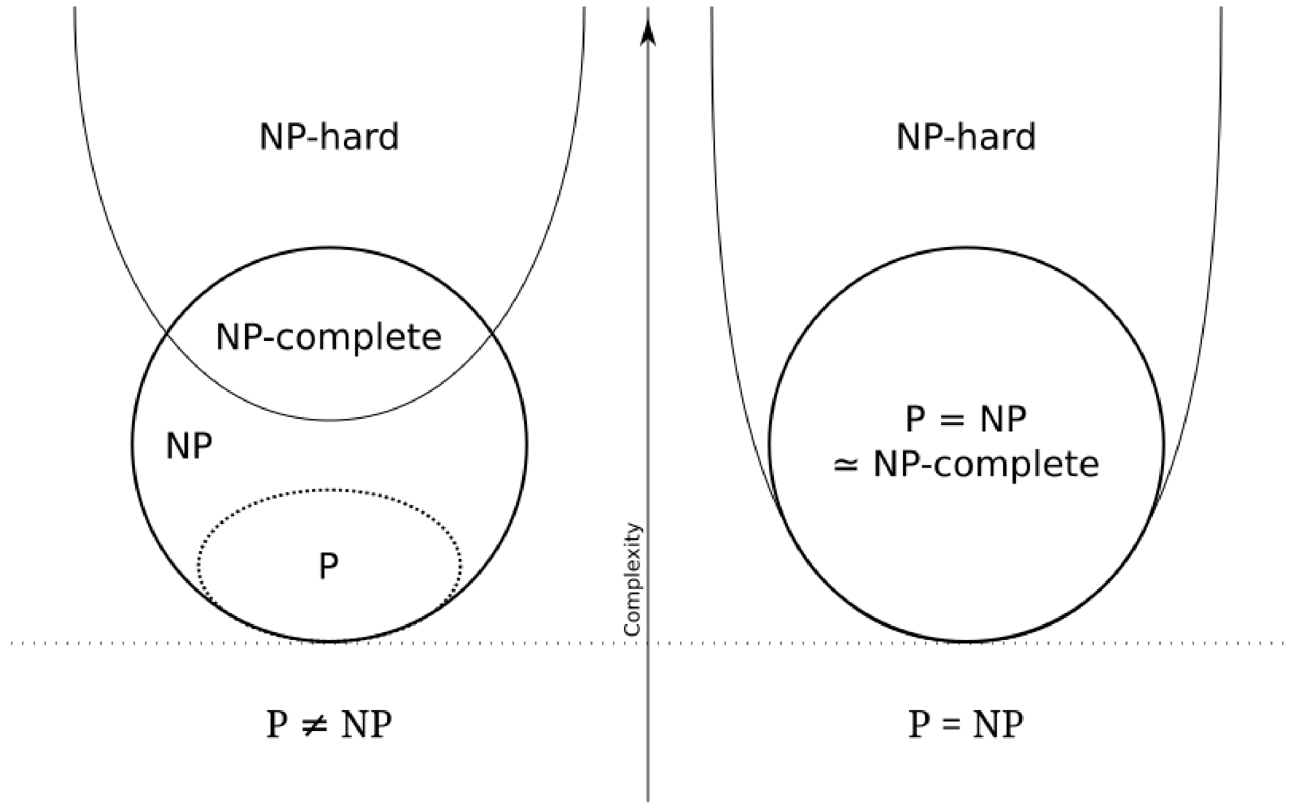
NP-complete

Observation: If Y is an NP-complete problem, and X is a problem in NP with the property that $\mathrm{Y} \leq_p \mathrm{X}$, then X is NP-complete.



NP-complete Problems

- (Circuit) SAT
- 3-SAT
- Independent Set
- Vertex Cover
- Set Cover


NP-hard
NP-complete
NP
P
Complexity
P ≠ NP
NP-hard
P = NP
≃ NP-complete
P = NP