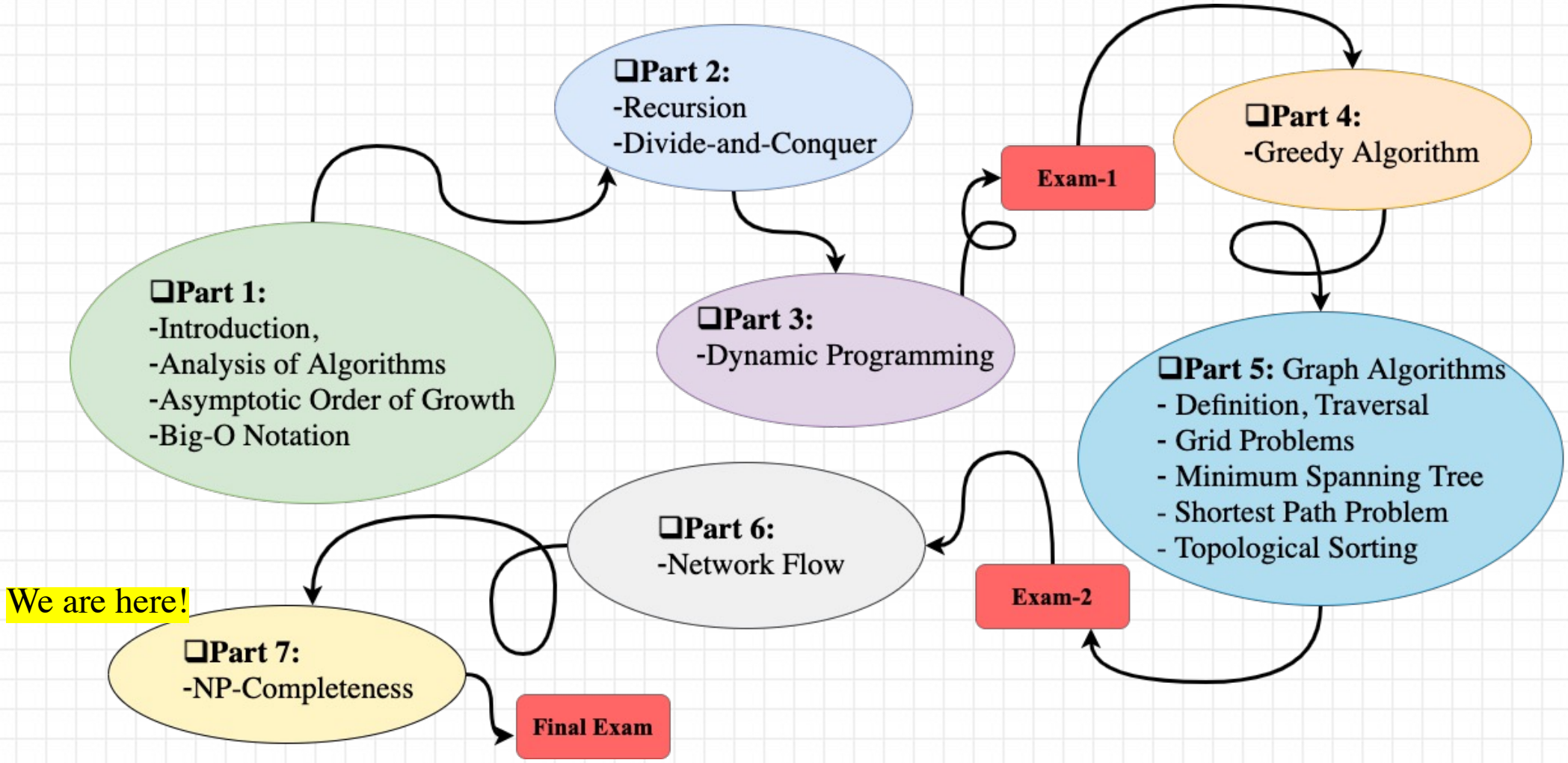
CS-3510:
Design and Analysis of Algorithms

NP Completeness II

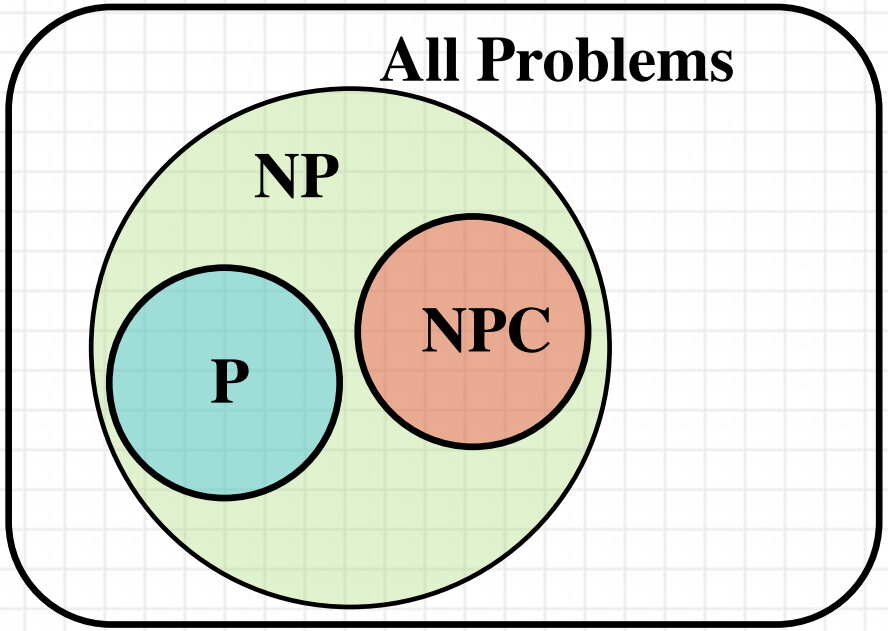
Instructor: Shahrokh Shahi

College of Computing
Georgia Institute of Technology
Summer 2022

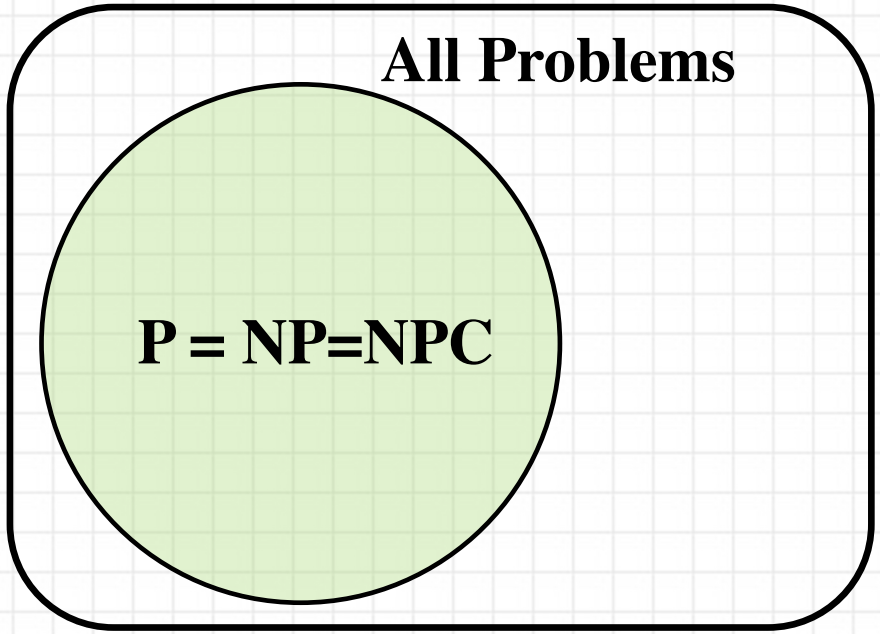
Roadmap


❑Part 1:
-Introduction,
-Analysis of Algorithms
-Asymptotic Order of Growth
-Big-O Notation
❑Part 2:
-Recursion
-Divide-and-Conquer
❑Part 3:
-Dynamic Programming
Exam-1
❑Part 4:
-Greedy Algorithm
❑Part 5: Graph Algorithms
- Definition, Traversal
- Grid Problems
- Minimum Spanning Tree
- Shortest Path Problem
- Topological Sorting
Exam-2
❑Part 6:
-Network Flow
We are here!
❑Part 7:
-NP-Completeness
Final Exam

Possible Worlds



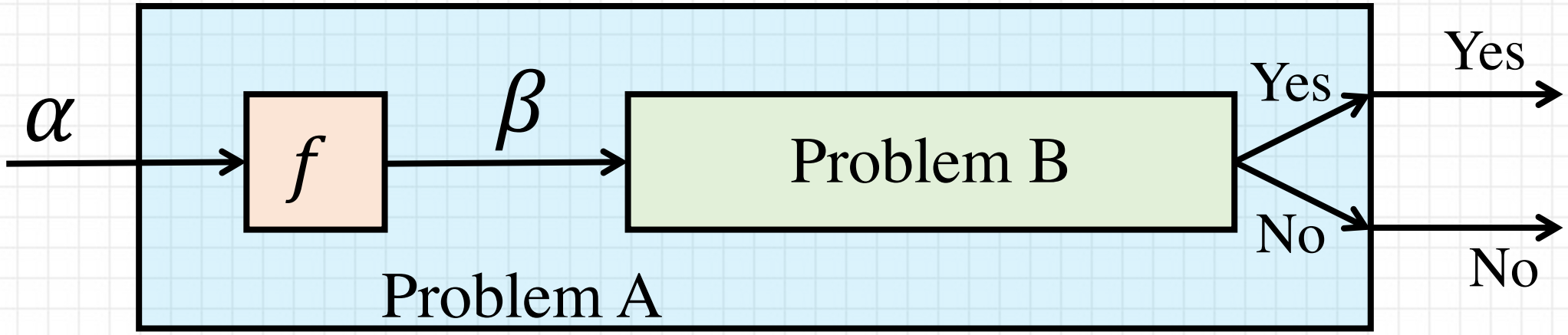
P≠NP



P=NP

Reductions

- $A \leq B$: Reduction from A to B is showing that we can solve A using the algorithm that solves B
- If we have an oracle for solving B, then we can solve A by making polynomial number of computations and polynomial number of calls to the oracle for B
- We can transform the inputs of A to inputs of B



Polynomial Reductions

- Given two problems, A and B, we say that A is polynomially reducible to B, and write it as $A \leq_p B$ if:

1. There exists a function f that converts the input of A **to** inputs of B in polynomial time

2. $A(i) = \text{YES} \iff B(f(i)) = \text{YES}$

Implications of Polynomial-Time Reductions

- **Purpose**. Classify problems according to relative difficulty.
- **Design algorithms**. If $X \leq_p Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time.
- **Establish intractability**. If $X \leq_p Y$ and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial time.
- **Establish equivalence**. If $X \leq_p Y$ and $Y \leq_p X$, we use notation $X \equiv_p Y$.
- **Transitivity**. If $X \leq_p Y$ and $Y \leq_p Z$,then $X \leq_p Z$.

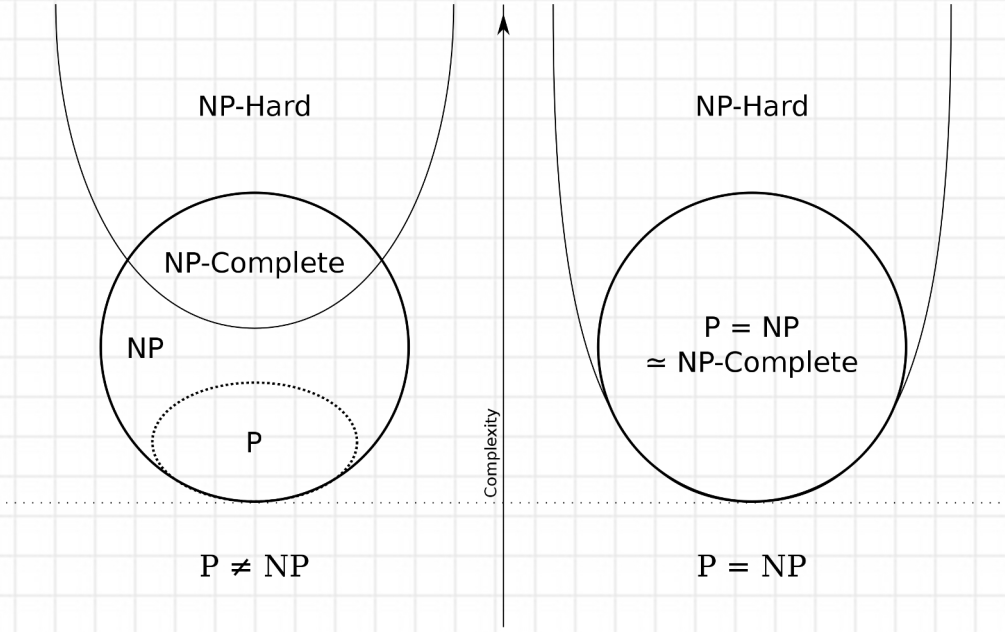
Reductions Strategies

- Given two problems, A and B, we say that A is polynomially reducible to B, and write it as $A \leq_p B$ if:
  1. There exists a function f that converts the input of A **to** inputs of B in polynomial time
  2. $A(i) = \text{YES} \iff B\big(f(i)\big) = \text{YES}$

- Reductions Strategies
  - Reduction by simple equivalence.
  - Reduction from a special case to a general case.
  - Reduction by encoding with gadgets.

NP-Completeness (Formal Definition)

- A problem Y is **NP-hard** if $X \leq_p Y$ for all $X \in$ **NP**
  - A problem is NP-hard if and only if a polynomial-time algorithm for it implies a polynomial-time algorithm for every problem in NP
  - NP-hard problems are at least as hard as any NP problem
- A problem Y is **NP-complete** if:
  1. $Y \in$ **NP**
  2. Y is **NP-hard**



https://en.wikipedia.org/wiki/P_versus_NP_problem

Establishing NP-Completeness

- Recipe to establish NP-completeness of problem Y.
  - Step1. Show that Y is in NP. ($Y \in$ **NP**)
    - Describe how a potential solution will be represented
    - Describe a procedure to check whether the potential solution is a correct solution to the problem instance, and argue that this procedure takes polynomial time
  - Step 2. Choose an NP-complete problem X.
  - Step 3. Prove that $X \leq_p Y$ (X is **poly-time reducible** to Y).
    - Describe a procedure f that converts the inputs i of X to inputs of Y in polynomial time
    - Show that the reduction is correct by showing that $X(i) = \text{YES} \iff Y(f(i)) = \text{YES}$

      Note this is an "if and only if" condition, so proofs are needed for both directions.

The First NPC Problem

- The satisfiability (SAT) problem was the first problem shown to be NP-complete **(Cook–Levin theorem)**

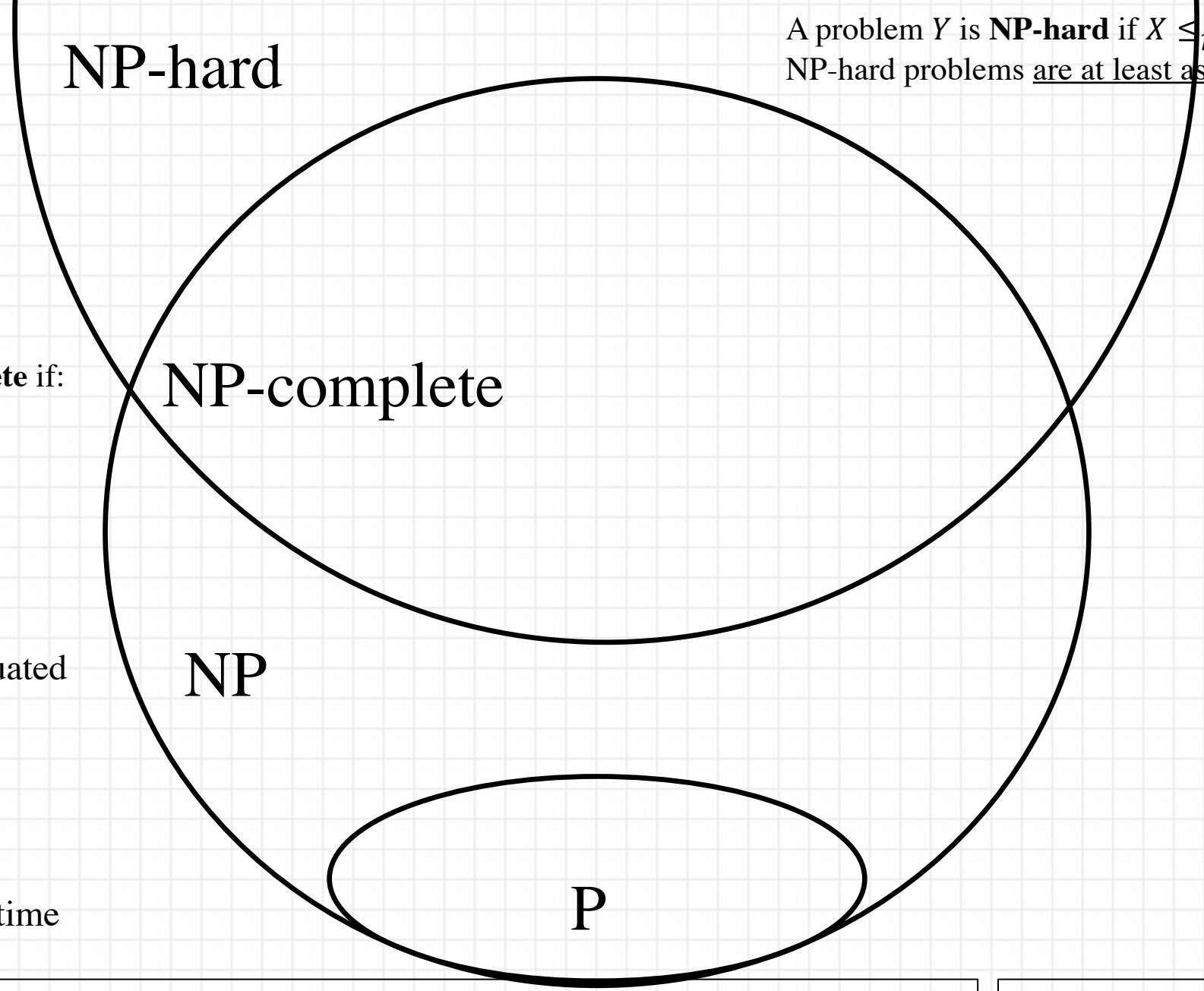
- **Satisfiability problem:** given a logical expression Φ, find an assignment of True/False values to binary variables x_i that causes Φ to evaluate to T.
- Ex. $$\Phi = x_1 \lor \neg x_2 \land x_3 \lor \neg x_4$$

The First NPC Problem

- **Satisfiability problem:** given a logical expression Φ, find an assignment of True/False values to binary variables x_i that causes Φ to evaluate to T.
- Ex. $$\Phi = x_1 \lor \neg x_2 \land x_3 \lor \neg x_4$$
- **SAT is in NP**: given a value assignment, check the Boolean logic of Φ evaluates to True (linear time)

- The satisfiability (SAT) problem was the first problem shown to be NP-complete **(Cook–Levin theorem)**

A problem Y is **NP-hard** if $X \leq_p Y$ for all $X \in$ **NP**
NP-hard problems are at least as hard as any NP problem

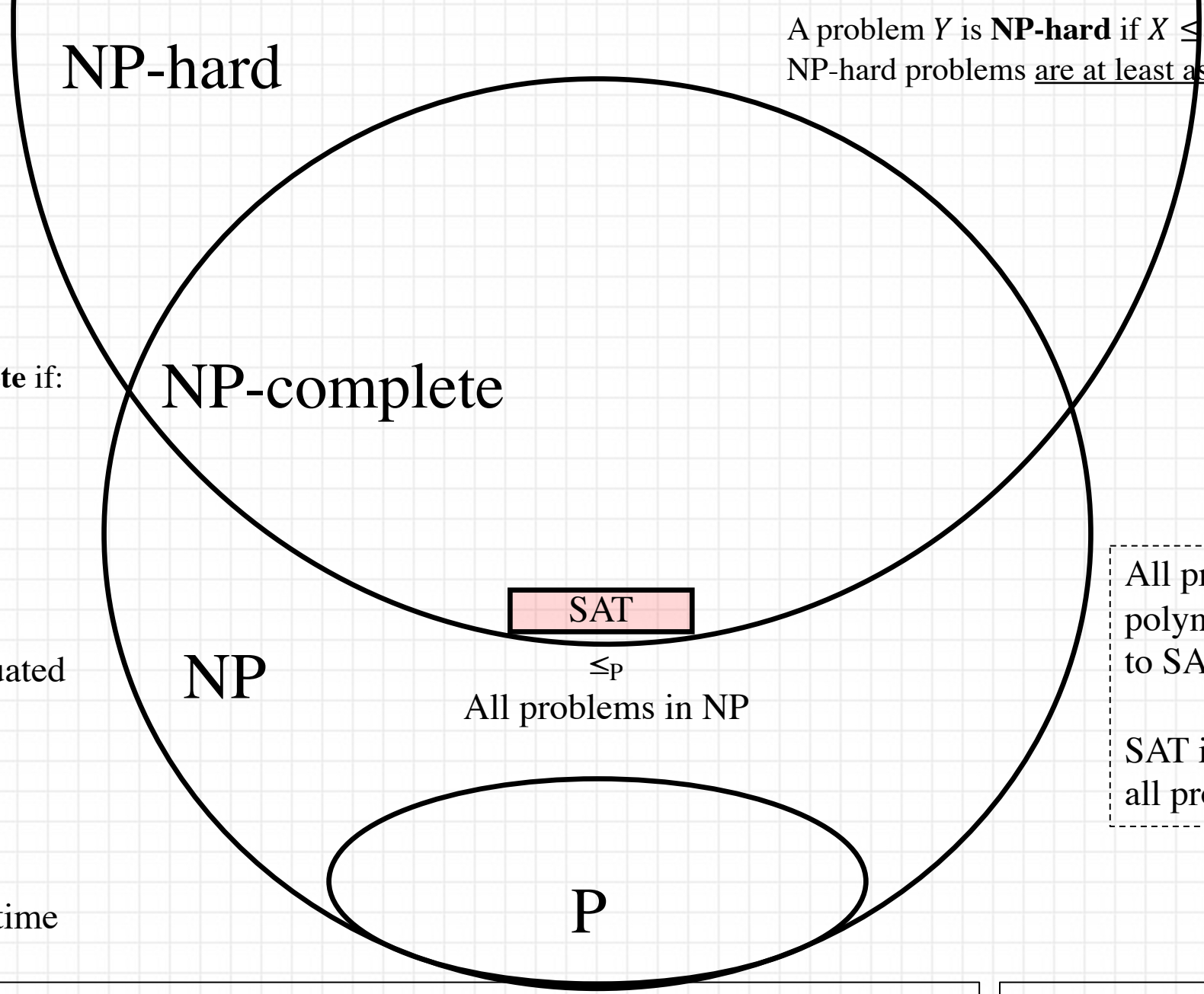


A problem Y is **NP-complete** if:
1. Y ∈ **NP**
2. Y is **NP-hard**

Candidate can be evaluated in poly-time

Solvable in poly-time

A problem Y is **NP-hard** if $X \leq_p Y$ for all $X \in$ **NP**
NP-hard problems are at least as hard as any NP problem



A problem Y is **NP-complete** if:

1. Y ∈ **NP**
2. Y is **NP-hard**

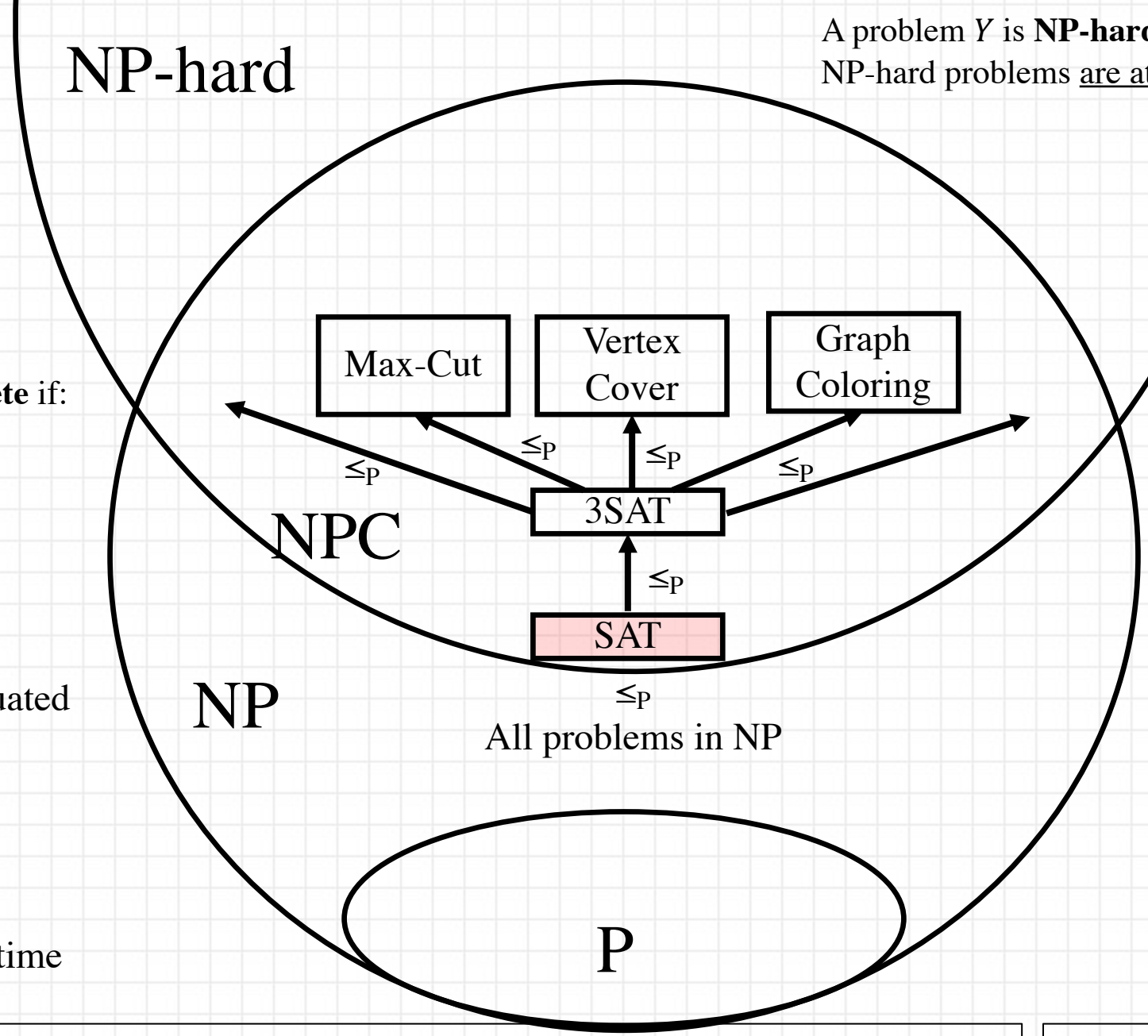
Candidate can be evaluated in poly-time

All problems in NP can polynomially be reduced to SAT

SAT is at least as hard as all problems in NP

Solvable in poly-time

A problem Y is **NP-hard** if $X \leq_p Y$ for all $X \in$ **NP**
NP-hard problems are at least as hard as any NP problem



A problem Y is **NP-complete** if:

1. Y ∈ **NP**
2. Y is **NP-hard**

Candidate can be evaluated in poly-time

Solvable in poly-time

Once we establish the first "natural" NPC problem, others fall like dominoes!

All problems in NP can polynomially be reduced to SAT

SAT is at least as hard as all problems in NP

CNF Satisfiability | 3SAT

- CNF is a special case of SAT
- Φ is an "Conjunctive Normal Form" (CNF) if
  - "AND" of expressions (i.e., clauses)
  - Each clause contains only "OR"s of the variables and their negations
- Ex. $\Phi = (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$

- 3SAT is a subcase of CNF problem, in which each cluse contains three literals.
- Ex. $\Phi = (\neg x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

We want to show 3SAT is an NPC problem

3SAT Problem

- We want to show that 3SAT is an NP-complete problem.

Recipe to establish NP-completeness of problem Y.

Step1. Show that Y is in NP. ($Y \in$ **NP**)

Describe how a potential solution will be represented

Describe a procedure to check whether the potential solution is a correct solution to the problem instance, and argue that this procedure takes polynomial time

Y → 3SAT

Step 2. Choose an NP-complete problem X.

X → SAT

Step 3. Prove that $X \leq_p Y$ (X is **poly-time reducible** to Y).

Describe a procedure f that converts the inputs i of X to inputs of Y in polynomial time

Show that the reduction is correct by showing that $X(i) = \text{YES} \iff Y(f(i)) = \text{YES}$

Note this is an "if and only if" condition, so proofs are needed for both directions.

SAT $\leq_p$ 3SAT

3SAT Problem

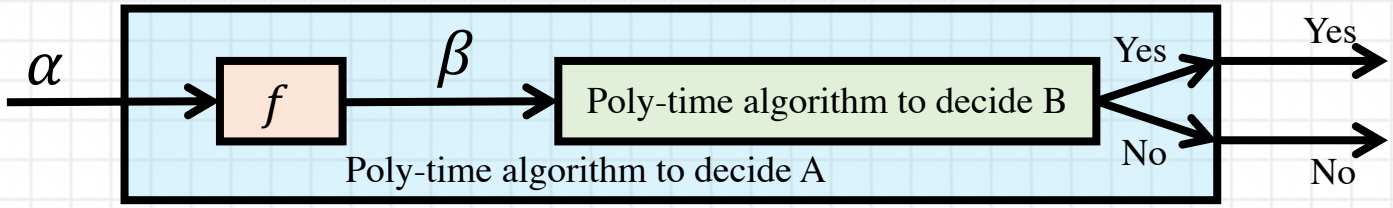
- **(1) To show 3SAT is in NP**
  - A certificate is a truth (0/1) assignment to the variables
  - Certifier: check that each clause has at least one literal set to true according to the certificate
- **(2) Choose SAT as a known NP-complete problem**
- **(3) Describe a reduction from SAT inputs to 3SAT inputs**
  - Computable in polytime
  - SAT input is satisfiable iff constructed 3SAT input is satisfiable
  - **(3a)** Transform I_1 (instance of SAT) into I_2 (instance of 3SAT) in polynomial time
  - **(3b, 3c)** Prove that I_1 has a solution $\Leftrightarrow$ I_2 has a solution

$$X(i) = \text{YES} \Leftrightarrow Y\big(f(i)\big) = \text{YES}$$

(3a) Reduction from SAT to 3SAT: $I_1 \rightarrow I_2$

- We are given an arbitrary CNF formula $C = c_1 \wedge c_2 \wedge \ldots \wedge c_m$ over set of variables U, this is instance I_1
  - each c_i is a clause (disjunction of literals)
- We will replace each clause c_i with a set of clauses C_i', and may use some extra variables U_i' just for this clause
- Each clause in C_i' will have exactly 3 literals
- Transformed input will be conjunction of all the clauses in all the C_i', this is an instance I_2 of 3SAT
- New clauses are carefully chosen…

Note
for **SAT $\leq_p$ 3SAT**
we need to transform the instance of SAT problem into the instance of 3SAT problem



(3a) Reduction from SAT to 3SAT: $I_1 \rightarrow I_2$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$ (the z's are literals)

- Case 1: k = 1.
  - E.g. $c_i = z_1$
  - Use extra variables y_i^1 and y_i^2.
  - Replace clause c_i with 4 clauses:

    $(z_1 \vee \overline{y_i^1} \vee \overline{y_i^2})$

    $(z_1 \vee y_i^1 \vee \overline{y_i^2})$

    $(z_1 \vee \overline{y_i^1} \vee \overline{y_i^2})$

    $(z_1 \vee y_i^1 \vee y_i^2)$
  - Note that no matter what values we give the y variables, in one of the 4 clauses we will be forced to use z_1 to satisfy it

If a clause of the SAT problem has only one literal.

(3a) Reduction from SAT to 3SAT: $I_1 \rightarrow I_2$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 2: k = 2.
  - E.g. $c_i = z_1 \vee z_2$
  - Use extra variable y_i^1.
  - Replace c_i with 2 clauses:

    $(z_1 \vee z_2 \vee \overline{y_i^1})$

    $(z_1 \vee z_2 \vee y_i^1)$

If a clause of the SAT problem has exactly two literals.

(3a) Reduction from SAT to 3SAT: $I_1 \rightarrow I_2$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 3: k = 3.
  - No extra variables are needed.
  - Keep c_i:

    $(z_1 \vee z_2 \vee z_3)$

(3a) Reduction from SAT to 3SAT: $I_1 \rightarrow I_2$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(z_1 \vee z_2 \vee y_i^1)$

$(\overline{y_i^1} \vee z_3 \vee y_i^2)$

$(\overline{y_i^2} \vee z_4 \vee y_i^3)$

$\ldots$

$\ldots$

$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$

$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$

$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

(3a) Reduction from SAT to 3SAT: $I_1 \rightarrow I_2$

- The running time of the reduction (the algorithm to compute the 3SAT formula C', given the SAT formula C) is proportional to the size of C'
- Rules for constructing C' are simple to calculate

So, this is a poly-time reduction.

(3a) Reduction from SAT to 3SAT: $I_1 \rightarrow I_2$

- **Original clause with 1 literal** becomes 4 clauses with 3 literals each (1 to 12 literals conversion)
- **Original clause with 2 literals** becomes 2 clauses with 3 literals each (2 to 6 literals conversion)
- **Original clause with 3 literals** becomes 1 clause with 3 literals
- **Original clause with k > 3 literals** becomes k-2 clauses with 3 literals each (k to 3(k-2) literals conversion)

Size of the new formula after transforming an instance of SAT to an instance of 3SAT.

- So new formula C' is only a constant factor larger than the original formula
  - total *L* literals in formula C to *cL* literals in C', where c is a constant

(3bc) Correctness of Reduction

- Show that CNF formula C is satisfiable iff the 3-CNF formula C' constructed is satisfiable, i.e., $sol(I_1) \Leftrightarrow sol(I_2)$

- Step 3b (=>) Suppose original CNF formula C is satisfiable, i.e., I_1 has a solution. That means it has a truth assignment A to the variables that make the formula C evaluate to true.
- Come up with a satisfying truth assignment for the reduced 3SAT formula C', i.e., a solution to instance I_2.
  - For variables in U, use same truth assignments as for C.
  - How to assign T/F to the new variables in C'?

(3b) $sol(I_1) \Rightarrow sol(I_2)$

(3b) $sol(I_1) \Rightarrow sol(I_2)$: Truth Assignment for New Variables

Let $c_i = z_1$

- Case 1: k = 1.
- Use extra variables y_i^1 and y_i^2.
  - Replace c_i with 4 clauses:

    $(z_1 \vee \overline{y_i^1} \vee y_i^2)$

    $(z_1 \vee y_i^1 \vee \overline{y_i^2})$

    $(z_1 \vee \overline{y_i^1} \vee \overline{y_i^2})$

    $(z_1 \vee y_i^1 \vee y_i^2)$

Since z_1 is true, it does not matter how we assign y_i^1 and y_i^2

(3b) $sol(I_1) \Rightarrow sol(I_2)$: Truth Assignment for New Variables

Let $c_i = (z_1 \vee z_2)$

- Case 2: k = 2.
  - Use extra variable y_i^1.
  - Replace c_i with 2 clauses:

    $(z_1 \vee z_2 \vee y_i^1)$

    $(z_1 \vee z_2 \vee \overline{y_i^1})$

Since either z_1 or z_2 is true, it does not matter how we assign y_i^1

(3b) $sol(I_1) \Rightarrow sol(I_2)$: Truth Assignment for New Variables

Let $c_i = (z_1 \vee z_2 \vee z_3)$

- Case 3: k = 3.
  - No extra variables are needed.
  - Keep c_i:

    $(z_1 \vee z_2 \vee z_3)$

No new variables.

(3b) $sol(I_1) \Rightarrow sol(I_2)$: Truth Assignment for New Variables

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(z_1 \vee z_2 \vee y_i^1)$
$(\overline{y_i^1} \vee z_3 \vee y_i^2)$
$(\overline{y_i^2} \vee z_4 \vee y_i^3)$
. . .

. . .
$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$
$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$
$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

If first true literal is z_1 or z_2, set all y_i's to false: then all later clauses have a true literal

(3b) sol(I_1) ⇒ sol(I_2): Truth Assignment for New Variables

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(z_1 \vee z_2 \vee y_i^1)$
$(\overline{y_i^1} \vee z_3 \vee y_i^2)$
$(\overline{y_i^2} \vee z_4 \vee y_i^3)$
. . .

. . .
$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$
$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$
$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

If first true literal is z_{k-1} or z_k, set all y_i's to true: then all earlier clauses have a true literal

(3b) $sol(I_1) \Rightarrow sol(I_2)$: Truth Assignment for New Variables

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(z_1 \vee z_2 \vee y_i^1)$
$(\overline{y_i^1} \vee z_3 \vee y_i^2)$
$(\overline{y_i^2} \vee z_4 \vee y_i^3)$
. . .

. . .
$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$
$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$
$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

If first true literal is in between,
set all earlier y_i's to true
and
set all later y_i's to false

(3c) sol(I_2) ⇒ sol(I_1): Correctness of Reduction

- (<=) Suppose the newly constructed 3SAT formula C' is satisfiable, i.e., I_2 has a solution. We must show that the original SAT formula C is also satisfiable, i.e., I_1 has a solution.
- Use the same satisfying truth assignment for C as for C' (ignoring new variables).
- Show each original clause has at least one true literal in it.

(3c) $sol(I_2) \Rightarrow sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 1: k = 1.
- Use extra variables y_i^1 and y_i^2.
  - Replace c_i with 4 clauses:

    $(z_1 \vee \overline{y_i^1} \vee \overline{y_i^2})$

    $(z_1 \vee y_i^1 \vee y_i^2)$

    $(z_1 \vee \overline{y_i^1} \vee \overline{y_i^2})$

    $(z_1 \vee y_i^1 \vee y_i^2)$

For every assignment of y_i^1 and y_i^2, in order for all 4 clauses to have a true literal, z_1 must be true.

(3c) $sol(I_2) \Rightarrow sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 2: k = 2.
  - Use extra variable y_i^1.
  - Replace c_i with 2 clauses:

    $(z_1 \vee z_2 \vee y_i^1)$

    $(z_1 \vee z_2 \vee \overline{y_i^1})$

For either assignment of y_i^1, in order for both clauses to have a true literal, z_1 or z_2 must be true.

(3c) $sol(I_2) \Rightarrow sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 3: k = 3.
  - No extra variables are needed.
  - Keep c_i:

    $(z_1 \vee z_2 \vee z_3)$

No new variables.

(3c) $sol(I_2) \Rightarrow sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(z_1 \vee z_2 \vee y_i^1)$
$(\overline{y_i^1} \vee z_3 \vee y_i^2)$
$(\overline{y_i^2} \vee z_4 \vee y_i^3)$
. . .

. . .
$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$
$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$
$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

Suppose in contradiction all z_i's are false.

(3c) $sol(I_2) \Rightarrow sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(z_1 \vee z_2 \vee y_i^1)$
$(\overline{y_i^1} \vee z_3 \vee y_i^2)$
$(\overline{y_i^2} \vee z_4 \vee y_i^3)$
. . .

. . .
$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$
$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$
$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

Suppose in contradiction all z_i's are false. Then y_i^1 must be true.

(3c) $sol(I_2) \Rightarrow sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(z_1 \vee z_2 \vee y_i^1)$
$(\overline{y_i^1} \vee z_3 \vee y_i^2)$
$(\overline{y_i^2} \vee z_4 \vee y_i^3)$
. . .
. . .
$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$
$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$
$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

Suppose in contradiction all z_i's are false. Then y_i^1 must be true, so y_i^2 must be true...

(3c) $sol(I_2) \Rightarrow sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(z_1 \vee z_2 \vee y_i^1)$

$(\overline{y_i^1} \vee z_3 \vee y_i^2)$

$(\overline{y_i^2} \vee z_4 \vee y_i^3)$

. . .

. . .

$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$

$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$

$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

Suppose in contradiction all z_i's are false. Then y_i^1 must be true, so y_i^2 must be true...

(3c) $sol(I_2) \Rightarrow sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(z_1 \vee z_2 \vee y_i^1)$

$(\overline{y_i^1} \vee z_3 \vee y_i^2)$

$(\overline{y_i^2} \vee z_4 \vee y_i^3)$

. . .

. . .

$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$

$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$

$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

Suppose in contradiction all z_i's are false. Then y_i^1 must be true, so y_i^2 must be true...

(3c) $sol(I_2) \Rightarrow sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(z_1 \vee z_2 \vee y_i^1)$
$(\overline{y_i^1} \vee z_3 \vee y_i^2)$
$(\overline{y_i^2} \vee z_4 \vee y_i^3)$
. . .
. . .
$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$
$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$
$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

Suppose in contradiction all z_i's are false. Then y_i^1 must be true, ..., y_i^{k-3} must be true, so last clause is false.

Conclusion

- (1) 3SAT is in NP
- (2) We know that SAT is NPC, we want to prove that 3SAT is more difficult than SAT, hence SAT $\leq_P$ 3SAT
- (3a) Take an instance I_1 of SAT, transform it in polynomial time into an instance I_2 of 3SAT
- (3b) Show that if I_1 has a solution, then I_2 has a solution
- (3c) Show that if I_2 has a solution, then I_1 has a solution

- 3SAT is NP-complete! This is your very first NP-completeness proof. Now you can do reductions from 3SAT.

- (All pbs in NP) $\leq_P$ SAT $\leq_P$ 3SAT

Conclusion

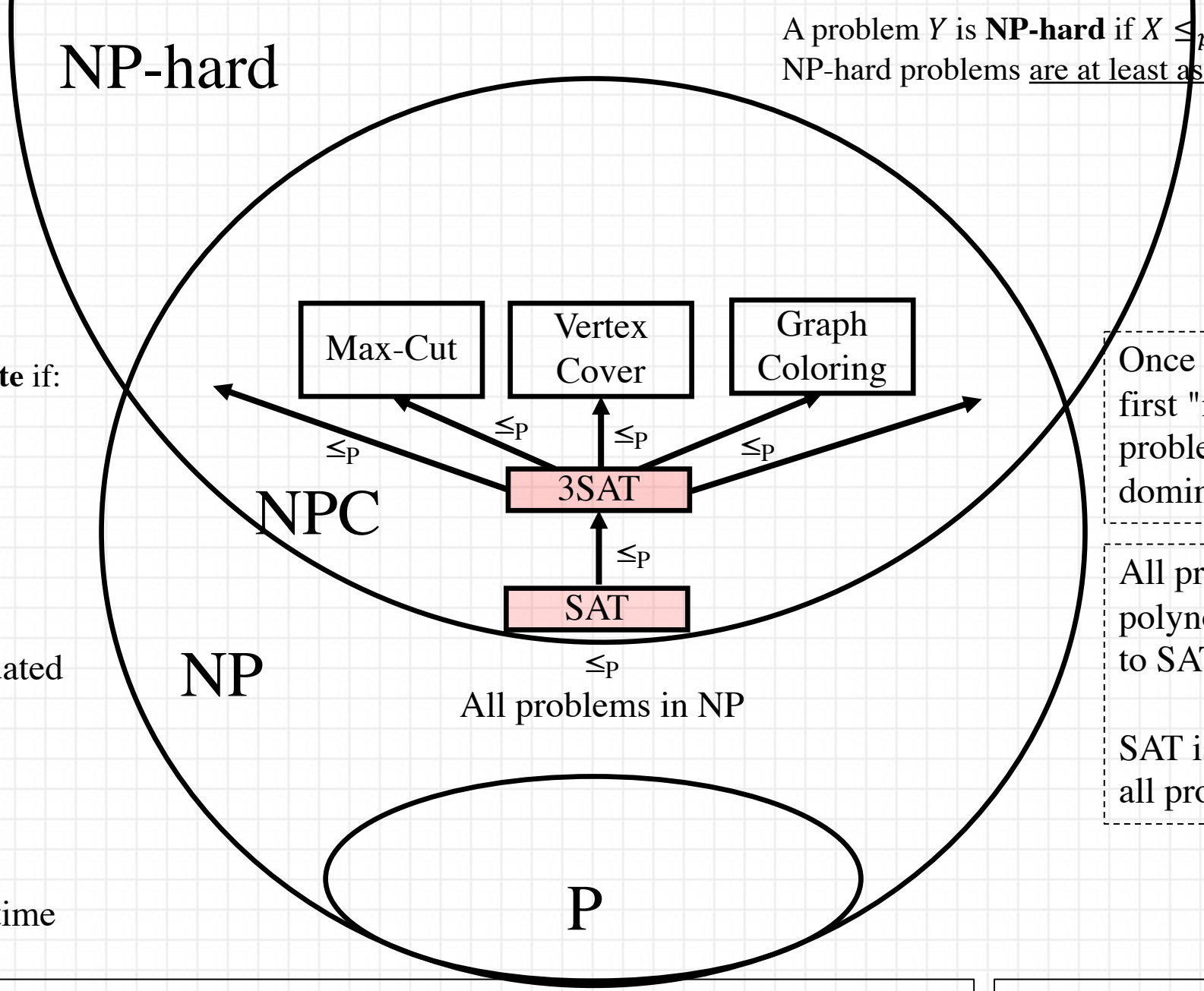
- (1) 3SAT is in NP
- (2) We know that SAT is NPC, we want to prove that 3SAT is more difficult than SAT, hence SAT $\leq_P$ 3SAT
- (3a) Take an instance I_1 of SAT, transform it in polynomial time into an instance I_2 of 3SAT
- (3b) Show that if I_1 has a solution, then I_2 has a solution
- (3c) Show that if I_2 has a solution, then I_1 has a solution

- 3SAT is NP-complete! This is your very first NP-completeness proof. Now you can do reductions from 3SAT.

- (All pbs in NP) $\leq_P$ SAT $\leq_P$ 3SAT

A problem Y is **NP-hard** if $X \leq_p Y$ for all $X \in$ **NP**
NP-hard problems are at least as hard as any NP problem

A problem Y is **NP-complete** if:
1. Y ∈ **NP**
2. Y is **NP-hard**



Candidate can be evaluated in poly-time

Solvable in poly-time

Once we establish the first "natural" NPC problem, others fall like dominoes!

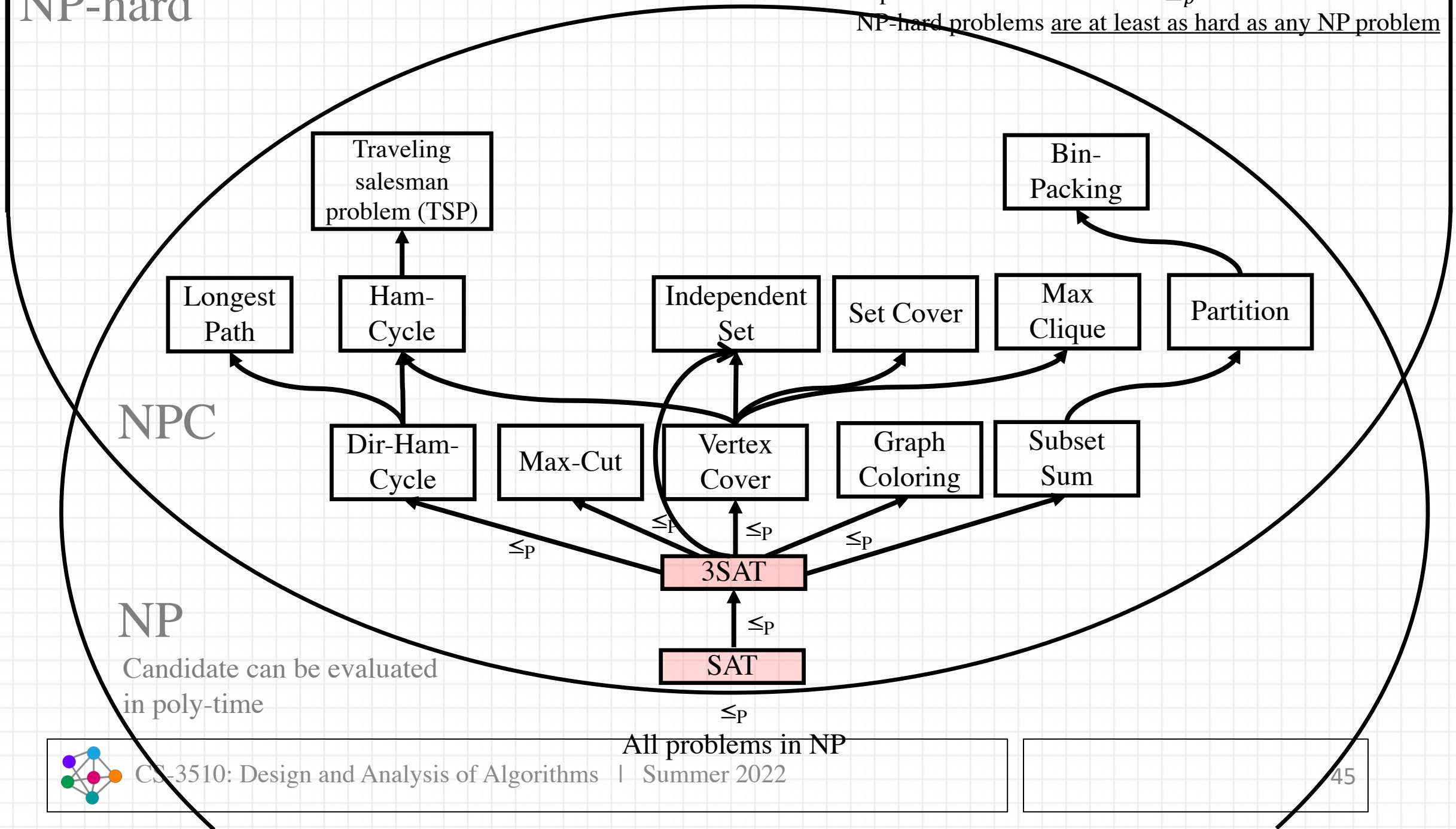
All problems in NP can polynomially be reduced to SAT

SAT is at least as hard as all problems in NP

NP-hard

A problem Y is **NP-hard** if $X \leq_p Y$ for all $X \in$ **NP**

NP-hard problems are at least as hard as any NP problem



Genres of NP-complete problems

- **Six basic genres of NPC problems and paradigmatic examples.**

1. Constraint satisfaction problems: SAT, 3-SAT.
2. Packing problems: SET-PACKING, INDEPENDENT SET.
3. Covering problems: SET-COVER, VERTEX-COVER.
4. Sequencing problems: HAMILTONIAN-CYCLE, TSP.
5. Partitioning problems: 3-COLOR, 3D-MATCHING.
6. Numerical problems: 2-PARTITION, SUBSET-SUM, KNAPSACK.

Genres of NP-complete problems

- **Six basic genres of NPC problems and paradigmatic examples.**

1. Constraint satisfaction problems: SAT, 3-SAT.
2. Packing problems: SET-PACKING, INDEPENDENT SET.
3. Covering problems: SET-COVER, VERTEX-COVER.
4. Sequencing problems: HAMILTONIAN-CYCLE, TSP.
5. Partitioning problems: 3-COLOR, 3D-MATCHING.
6. Numerical problems: 2-PARTITION, SUBSET-SUM, KNAPSACK.

Independent Set

- Independent set (IS)
  - Given a graph G=(V,E), find the largest independent set: a set of vertices in the graph with no edges between them.
- **Decision version**?
  - Is there an independent set of at least K vertices?

Minimum / least / shortest /… → at most k
Maximum/ greatest / longest / … → at least k

Independent Set

- We want to show Independent Set (IS) problem is an NPC problem

Recipe to establish <u>NP-completeness of problem Y.</u>

Step1. Show that Y is in NP. ($Y \in$ **NP**)

Describe how a potential solution will be represented

Describe a procedure to check whether the potential solution is a correct solution to the problem instance, and argue that this procedure takes polynomial time

Y → IS

Step 2. Choose <u>an</u> NP-complete problem X.

X → 3SAT

Step 3. Prove that $X \leq_p Y$ (X is **poly-time reducible** to Y).

Describe a procedure f that converts the inputs i of X to inputs of Y in polynomial time

Show that the reduction is correct by showing that $\boxed{X(i) = \text{YES} \iff Y(f(i)) = \text{YES}}$

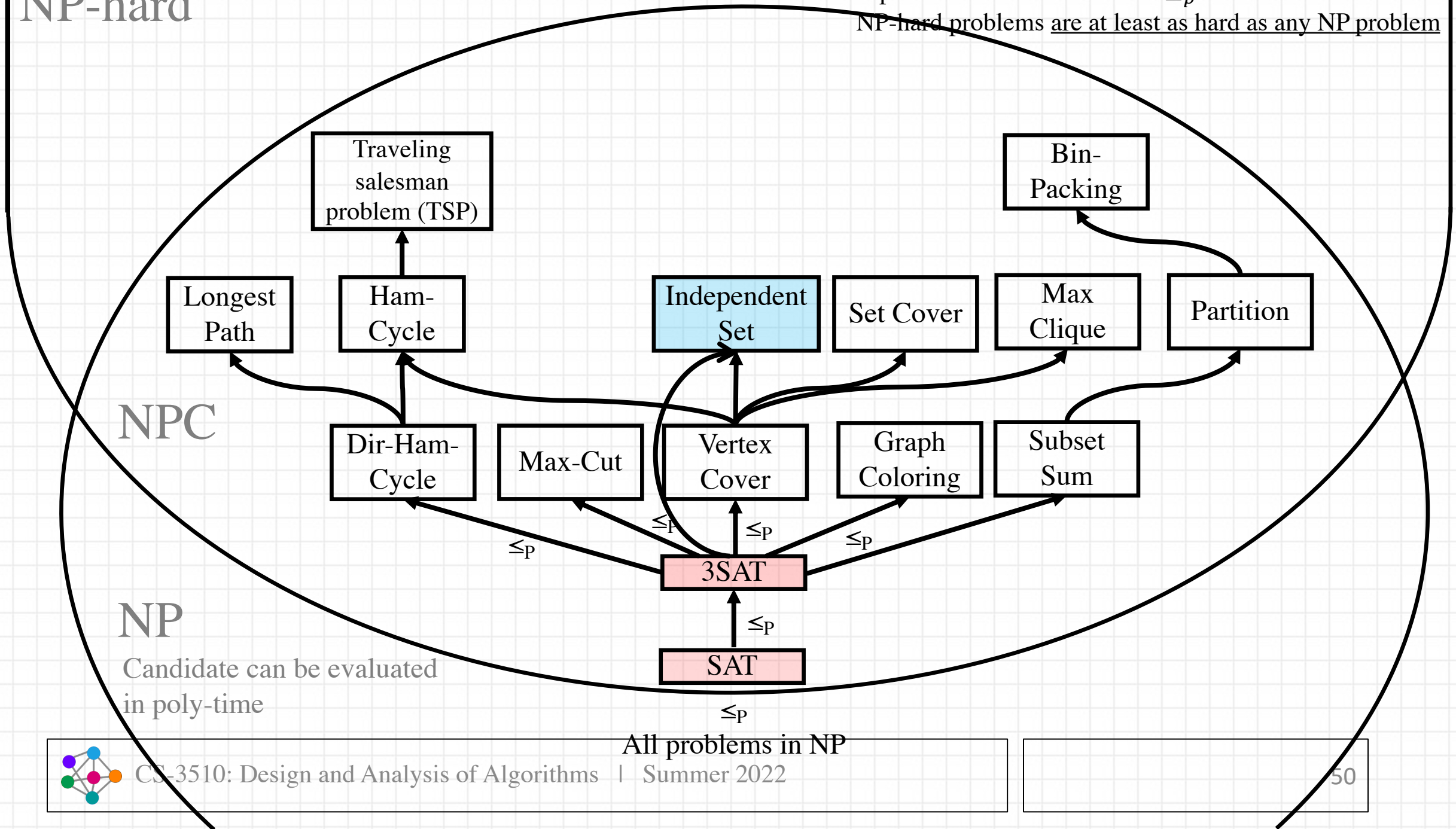
Note this is an "if and only if" condition, so proofs are needed for both directions.

3SAT $\leq_p$ IS

NP-hard

A problem Y is **NP-hard** if $X \leq_p Y$ for all $X \in$ **NP**

NP-hard problems are at least as hard as any NP problem



Independent Set

- Step1. Show that IS is in NP. ($Y \in$ **NP**)
  - Certificate: A set of vertices S
  - Certifier: Check size of S ≥ K, and no pair of vertices in S is connected by an edge, O(n+m)
- Step 2. Choose an NP-complete problem X. → 3SAT
- Step 3. Prove that $3SAT \leq_p IS$ (3SAT is **poly-time reducible** to IS).
  - Reduction by gadget

Reductions Strategies

Reduction by simple equivalence.

Reduction from a special case to a general case.

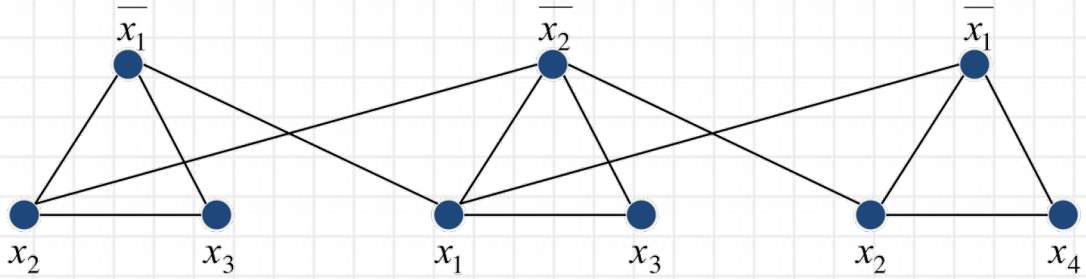
Reduction by encoding with gadgets.

3SAT $\leq_p$ IS

- **Claim**. 3-SAT $\leq_P$ INDEPENDENT-SET.
- **Pf**. Given an instance Φ of 3-SAT (I_1), we construct an instance (G, k) of INDEPENDENT-SET (I_2) that has an independent set of size k iff Φ is satisfiable.

- **Construction** (Step 3a)
  - G contains 3 vertices for each clause, one for each literal.
  - Connect 3 literals in a clause in a triangle.
  - Connect literal to each of its negations.
  - *The size of I_2 is polynomial in the size of I_1*

G



k = 3

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

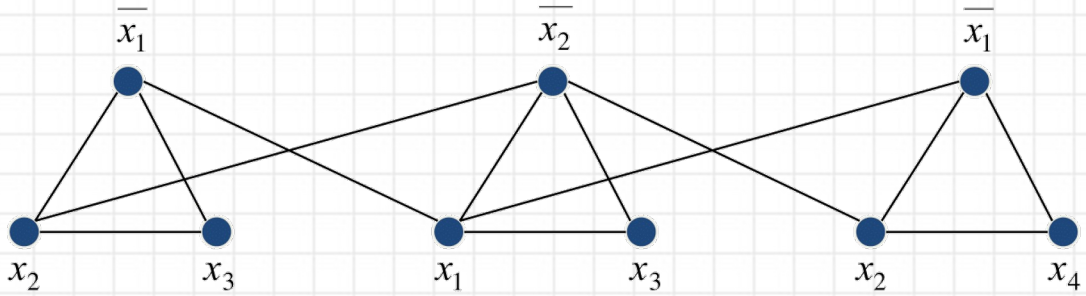
3SAT $\leq_p$ IS

- Claim. Φ is satisfiable iff G contains independent set of size k = $|\Phi|$
- (I_1 has a solution $\Leftrightarrow$ I_2 has a solution)

- => Given satisfying assignment (sol to I_1), select one true literal from each triangle. This is an independent set of size k, hence a sol to I_2.

- <= Let S be independent set of size k (sol. to I_2)
  - S must contain exactly one vertex in each triangle.
  - Set these literals to true. ← and any other variables in a consistent way
  - Truth assignment is consistent and all clauses are satisfied.

G



k = 3

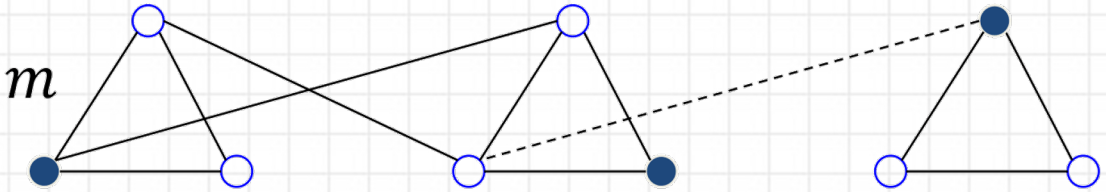
$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4\right)$$

3SAT $\leq_p$ IS

- Claim. $\Phi = c_1 \wedge c_2 \wedge \cdots c_m$ is satisfiable iff G contains independent set of size $k = m$
- (I_1 has a solution $\Leftrightarrow$ I_2 has a solution)

I_1: $\Phi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$

I_2: $G, k = m$

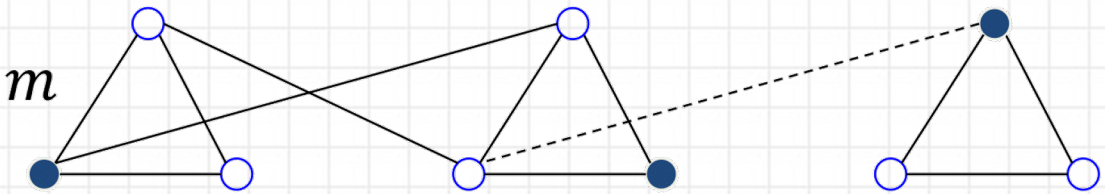


- (3b) sol (I_1) => sol (I_2)
  - Given satisfying assignment (sol to I_1),
  - select exactly one true literal from each triangle and add to S (sol to I_2)
  - Size S ? $k = m$
  - Independent set?
    - One node per triangle
    - If there is an edge between two nodes x_1 and x_2, with $x_1 \in S$ and $x_2 \in S$
    - (by construction) It means that $x_1 = x$ and $x_1 = \bar{x}$ (or reverse)

3SAT $\leq_p$ IS

I_1: $\Phi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$

I_2: $G, k = m$



- (3c) sol (I_2) => sol (I_1)
  - Let S be solution to IS
  - We have m triangles in G, we can pick at most 1 vertex in S and $|S| \geq k = m$

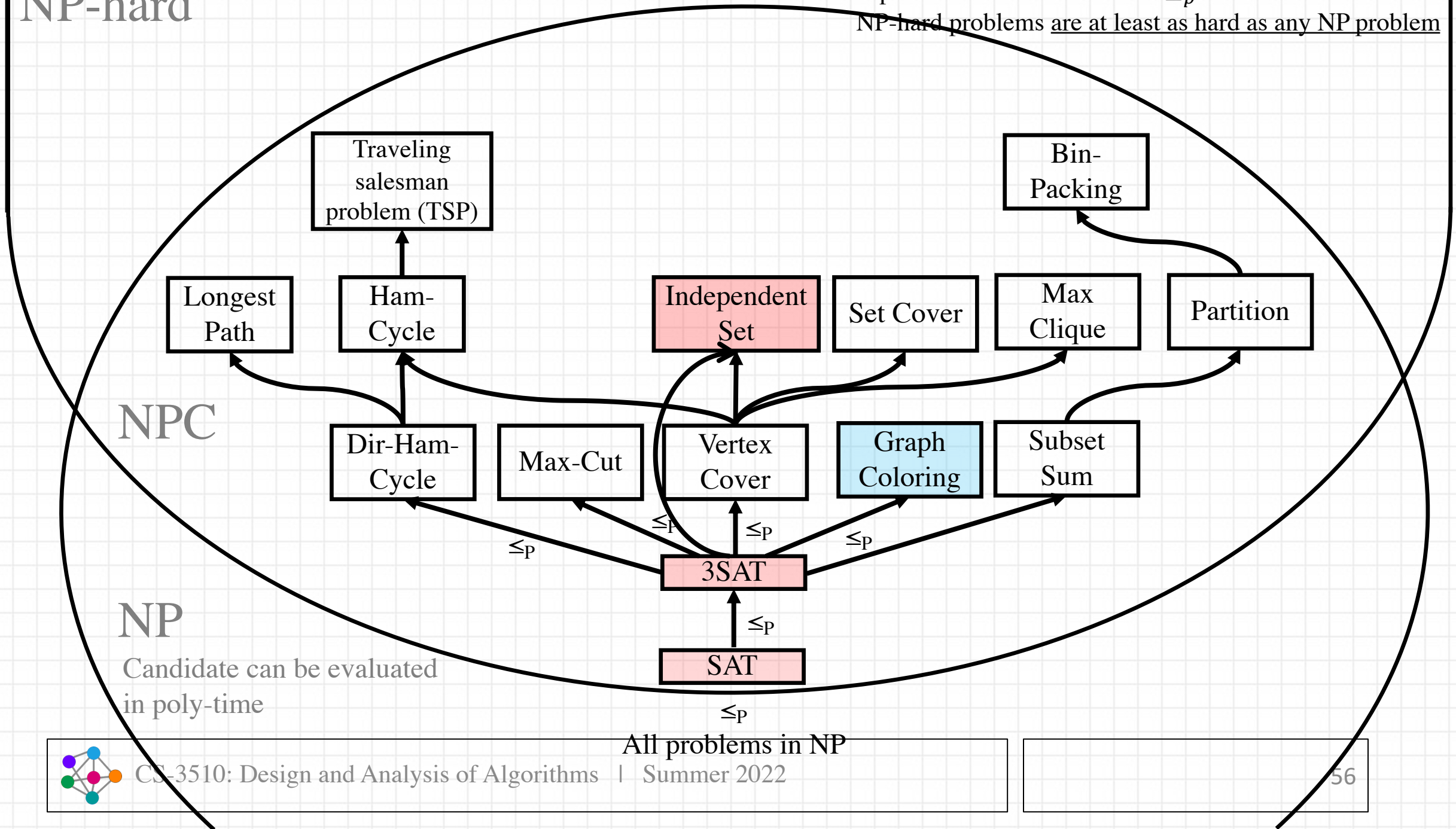
  $\Rightarrow$ S contains exactly one vertex per triangle
  - Set the literal corresponding to each vertex in S to true

  $\Rightarrow$ Every clause has one true literal.
  - *Argue a variable is not assigned both true and false?*
  - Let there be such variable x, then there must have been a vertex labeled x_i and $\bar{x_i}$ in S
  - By construction vertices with opposing labels x and $\bar{x}$ share and edge
  - $\rightarrow\leftarrow$ contradicts S being independent set ■

NP-hard

A problem Y is **NP-hard** if $X \leq_p Y$ for all $X \in$ **NP**

NP-hard problems are at least as hard as any NP problem



Genres of NP-complete problems

- **Six basic genres of NPC problems and paradigmatic examples.**

1. Constraint satisfaction problems: SAT, 3-SAT.
2. Packing problems: SET-PACKING, INDEPENDENT SET.
3. Covering problems: SET-COVER, VERTEX-COVER.
4. Sequencing problems: HAMILTONIAN-CYCLE, TSP.
5. Partitioning problems: 3-COLOR, 3D-MATCHING.
6. Numerical problems: 2-PARTITION, SUBSET-SUM, KNAPSACK.

Graph Coloring: 3-Colorability

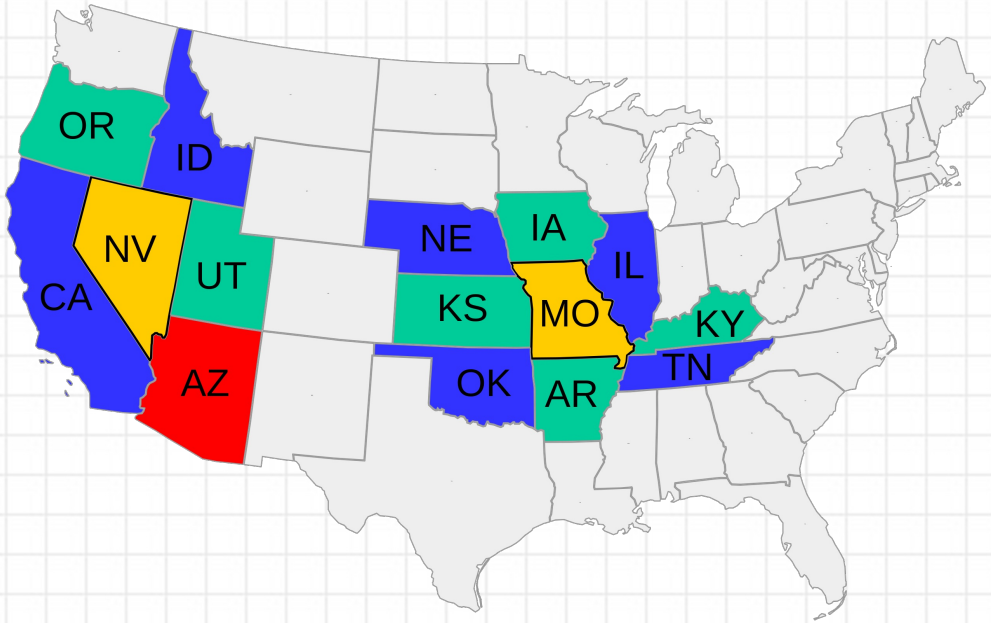
- Map Coloring
  - Given a map, can it be colored using 3 colors so that no adjacent states (regions) have the same color?
  - **Four color theorem**, or the **four-color map theorem**: We need no more than four colors to color the regions of any map so that no two adjacent regions have the same color. (Every planar map can be colored with four colors)
  - However, deciding whether an arbitrary planar map can be colored with just three colors is an NPC problem.

Graph Coloring: 3-Colorability

- Map Coloring
  - Given a map, can it be colored using 3 colors so that no adjacent states (regions) have the same color?
  - **Four color theorem**, or the **four-color map theorem**: We need no more than four colors to color the regions of any map so that no two adjacent regions have the same color. (Every planar map can be colored with four colors)
  - However, deciding whether an arbitrary planar map can be colored with just three colors is an NPC problem.

Odd number of adjacent states

Even number of adjacent states

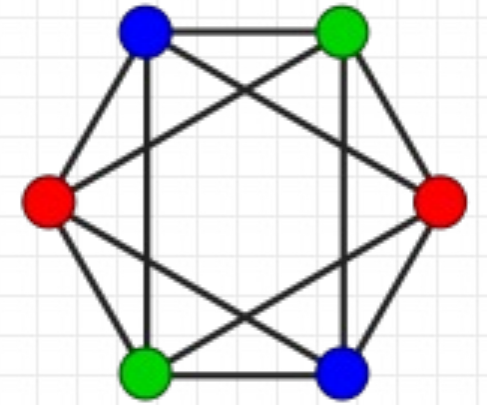


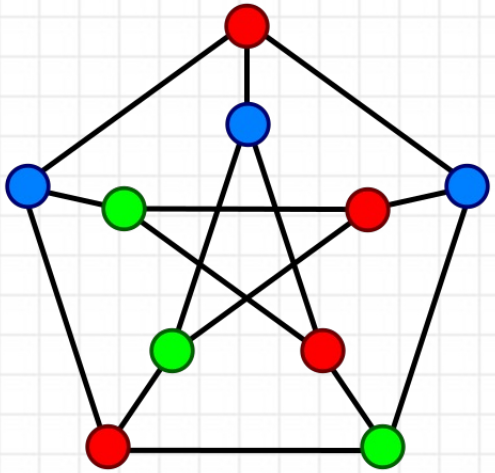
US states map needs at least four colors

https://en.wikipedia.org/wiki/Four_color_theorem

Graph Coloring: 3-Colorability

- Graph Coloring
  - Each region is represented by a node in a graph; if 2 regions have a common boundary represent this by an edge between them. So, we wish to assign colors to the nodes so that no two nodes have the same color if there is an edge between them





Petersen graph

https://en.wikipedia.org/wiki/Graph_coloring

Graph Coloring: 3-Colorability

- Graph Coloring
  - We seek to assign a color to each node of a graph G so that if (u,v) is an edge, then u and v are assigned different colors; and the goal is to do this while using the smallest set of colors
  - A k-coloring of G is a function f: V → {1, 2, ..., k} so that for every edge (u,v), we have $f(u) \neq f(v)$.
  - If G has a k-coloring, we say that it is a k-colorable graph

- **Decision version**: Given a graph G and a bound k, does G have a k-coloring?

Graph Coloring: 3-Colorability

- Graph Coloring
  - We seek to assign a color to each node of a graph G so that if (u,v) is an edge, then u and v are assigned different colors; and the goal is to do this while using the smallest set of colors
  - A k-coloring of G is a function f: V → {1, 2, ..., k} so that for every edge (u,v), we have $f(u) \neq f(v)$.
  - If G has a k-coloring, we say that it is a k-colorable graph
- **Decision version**: Given a graph G and a bound k, does G have a k-coloring?

Side note regarding 4-coloring problem (k=4)

- Planar graphs/maps
- Problem over a century
- Resolved in 1976 by Appel and Haken
  - Induction on the number of the regions
  - But the induction step involved nearly 2000 complicated

3-Colorability

- `3-COLOR`: Given an undirected graph G does there exists a way to color the nodes using at most three colors (e.g., red, green, and blue) so that no adjacent nodes have the same color?
- We want to show `3-COLOR` problem is an NPC problem.

Recipe to establish <u>NP-completeness of problem Y.</u>

Step1. Show that Y is in NP. ($Y \in$ **NP**)

Step 2. Choose <u>an</u> NP-complete problem X.

Step 3. Prove that $X \leq_p Y$ (X is **poly-time reducible** to Y).

Describe a procedure f that converts the inputs i of X to inputs of Y in polynomial time

Show that the reduction is correct by showing that $\boxed{X(i) = \text{YES} \iff Y(f(i)) = \text{YES}}$

Note this is an "if and only if" condition, so proofs are needed for both directions.

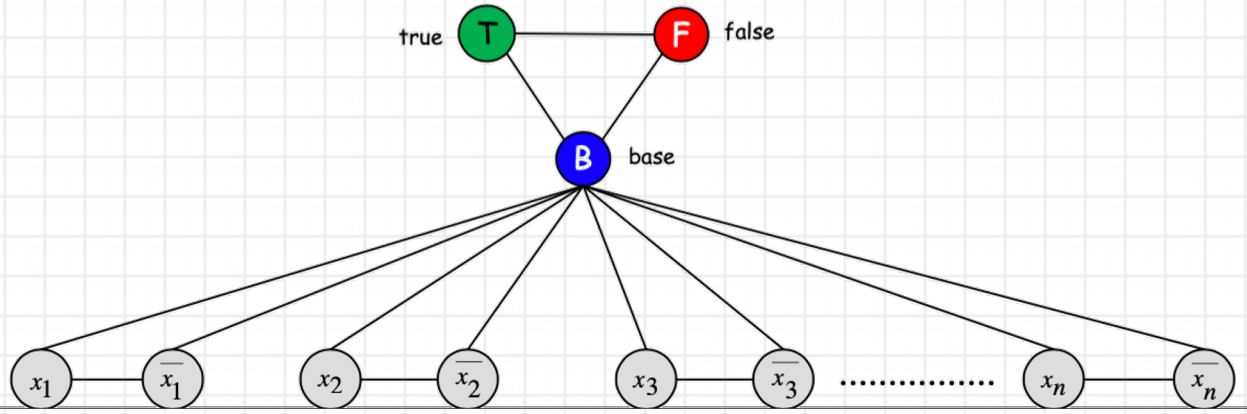
Y → 3-COLOR

X → 3SAT

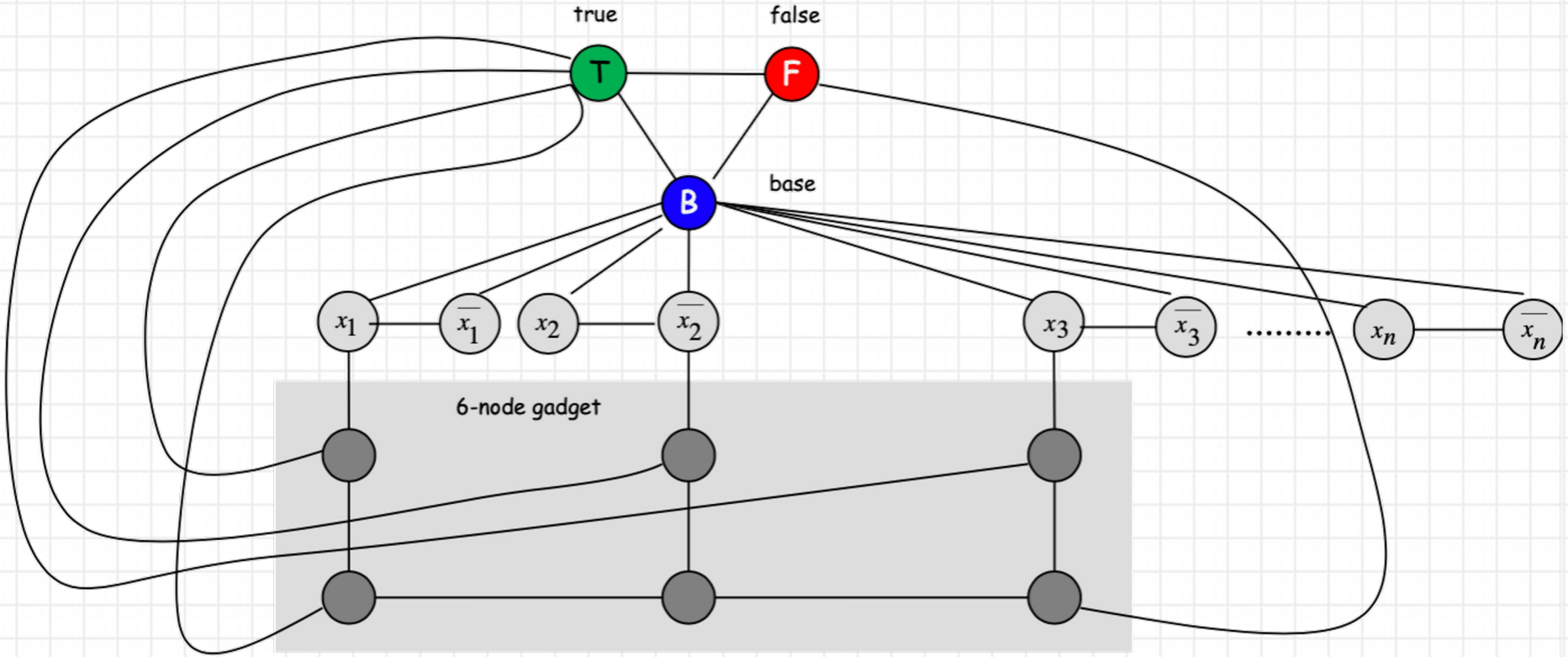
3SAT $\leq_p$ 3-COLOR

$3SAT \leq_p 3\text{–}COLOR$

- Claim: 3-SAT $\leq_P$ 3-COLOR
- Pf: Given 3-SAT instance Φ (I_1), we construct an instance I_2 of 3-COLOR such that Φ is satisfiable iff 3-colorable
- Construction
  - Create 3 new nodes T, F, B; connect them in a triangle (True, False, and Base)
  - For each literal, create a node
  - Connect each literal to B (i.e., every literal will have to be colored same as T or F)
  - Connect each literal to its negation (literals of same var cannot be same color)
  - For each clause, add gadget of 6 nodes and 13 edges as in next slide



$3SAT \leq_p 3\text{–}COLOR$

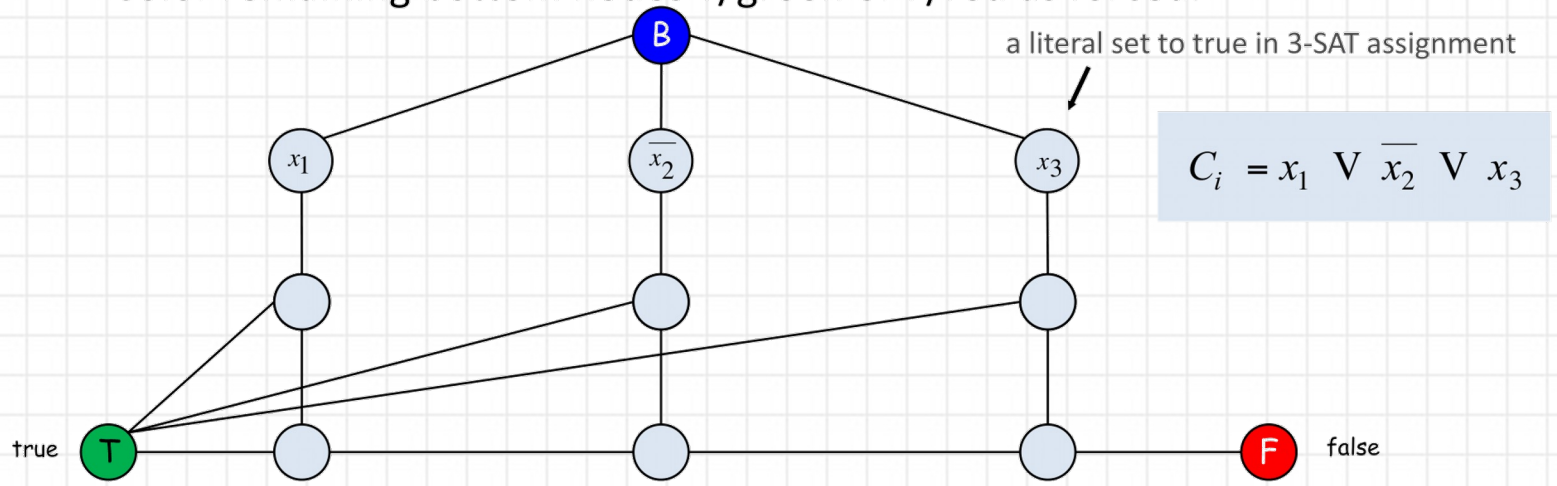


$$C_i = x_1 \text{ V } \overline{x_2} \text{ V } x_3$$

$3SAT \leq_p 3\text{–}COLOR$

- Claim. Φ is satisfiable (sol(I_1)) iff graph is 3-colorable (sol(I_2))

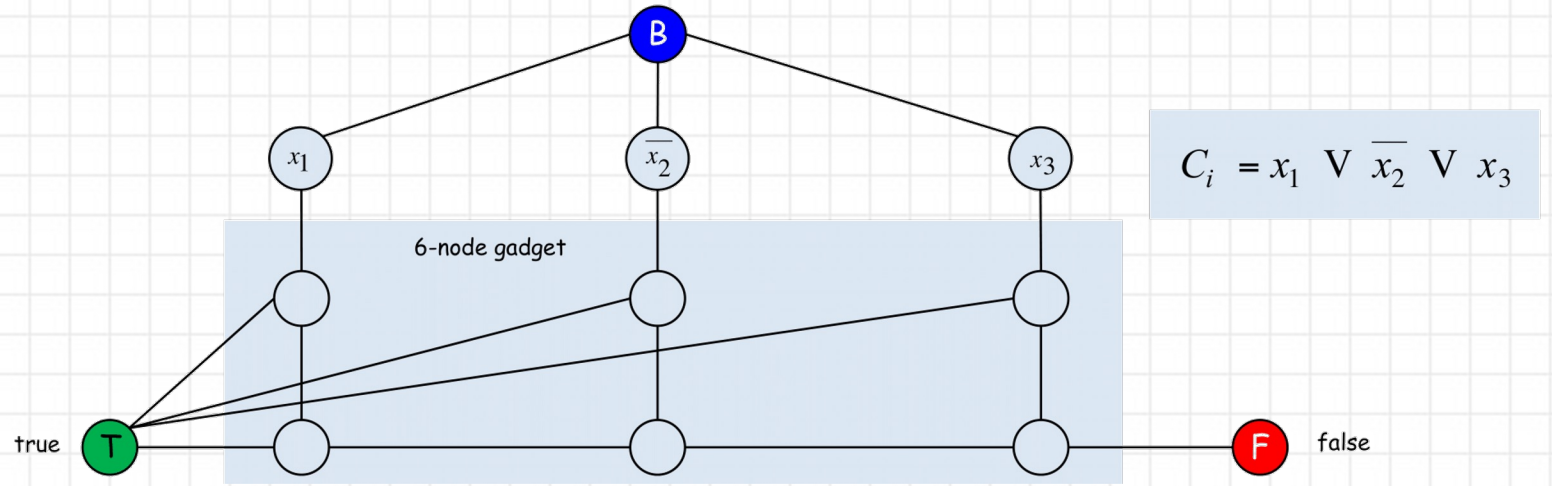
- Pf. $\Rightarrow$ Suppose 3-SAT formula Φ is satisfiable.
  - Color all nodes of true literals in the satisfying assignment with color T.
  - Then at least one literal in each clause is true, hence colored T/green
  - Color node below T/green node F/red, and node below that B/blue.
  - Color remaining middle row nodes B/blue.
  - Color remaining bottom nodes T/green or F/red as forced. ▪



3SAT $\leq_p$ 3–COLOR

- Claim. Φ is satisfiable (sol(I_1)) iff graph is 3-colorable (sol(I_2))

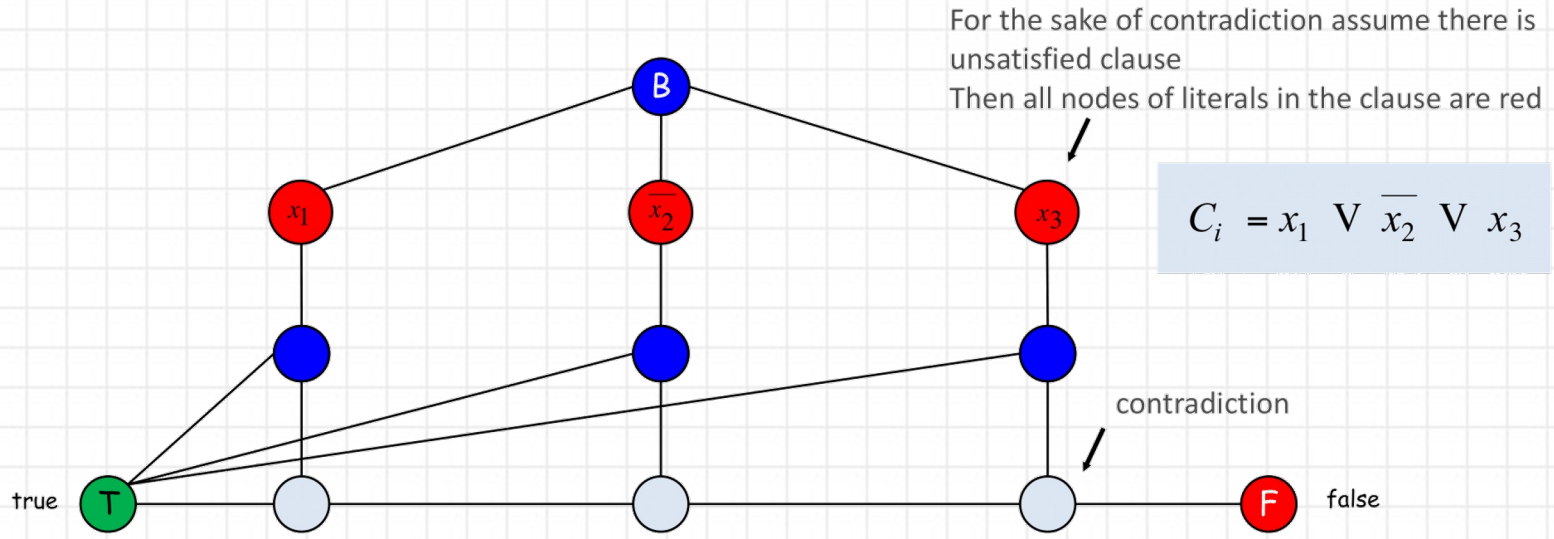
- Pf. $\Leftarrow$ Suppose graph is 3-colorable.
  - Consider assignment that sets all T-colored literals to true.
  - (i) ensures each literal is True or False.
  - (ii) ensures a literal and its negation are opposites.
  - (iii) ensures at least one literal in each clause is T (let's see why).



3SAT $\leq_p$ 3–COLOR

- Claim. Φ is satisfiable (sol(I_1)) iff graph is 3-colorable (sol(I_2))

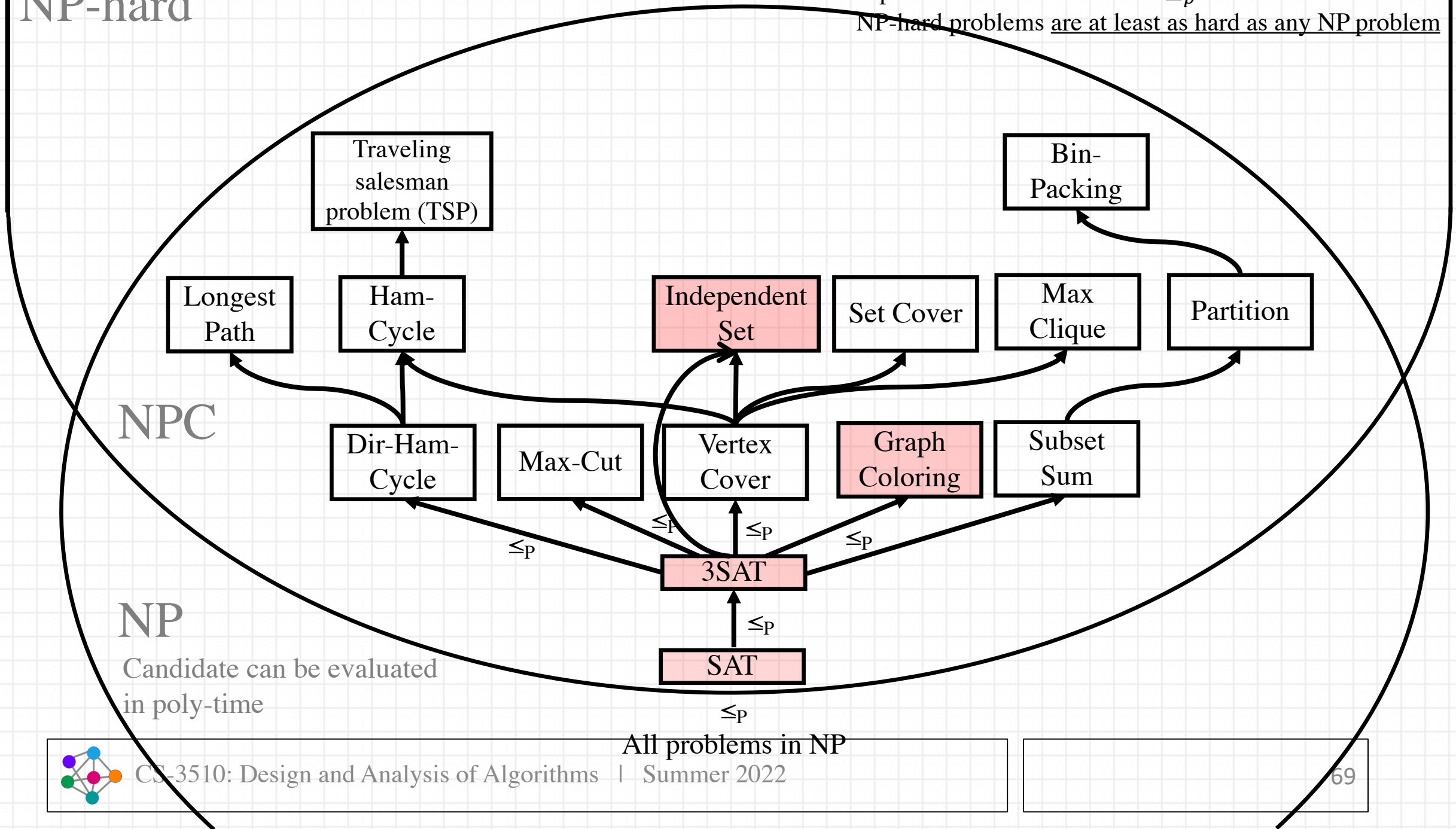
- Pf. $\Leftarrow$ Suppose graph is 3-colorable.
  - Consider assignment that sets all T-colored literals to true.
  - (i) ensures each literal is True or False.
  - (ii) ensures a literal and its negation are opposites.
  - (iii) ensures at least one literal in each clause is T (let's see why).



NP-hard

A problem Y is **NP-hard** if $X \leq_p Y$ for all $X \in$ **NP**

NP-hard problems are at least as hard as any NP problem



Coping with NP-completeness

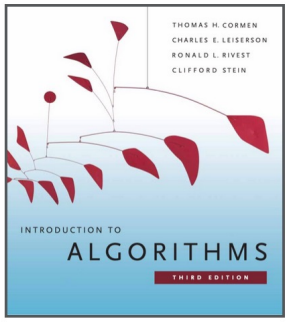
- **Exact solution**
  - Brute force → It will always explore all search space
  - Branch and bound → Create an algorithm with running time exponential in the input size (but which might do well on the inputs you use)
- **Parameterized algorithms**
  - Allow the running time to have an exponential factor, but ensure that the exponential dependence is not on the entire input size but just on some parameter that is hopefully small
- **Approximation**
  - Quickly find a solution that is *provably not very bad*
- **Local search**
  - Quickly find a solution for which you cannot give any quality guarantee (but which might often be good in practice on real problem instances)
- **Restriction**
  - By restricting the structure of the input (e.g., to planar graphs, 2SAT), faster algorithms are usually possible.
- **Randomization**
  - Use randomness to get a faster average running time and allow the algorithm to fail to find optimum with some small probability.

Coping with NP-completeness

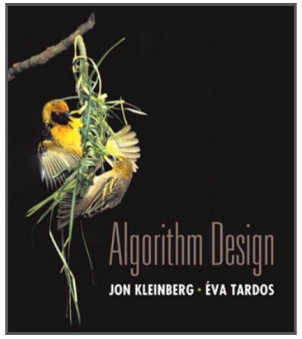
- **Exact solution** (Sacrifice running time)
  - Brute force → It will always explore all search space
  - Branch and bound → Create an algorithm with running time exponential in the input size (but which might do well on the inputs you use)
- **Parameterized algorithms** (Sacrifice running time)
  - Allow the running time to have an exponential factor, but ensure that the exponential dependence is not on the entire input size but just on some parameter that is hopefully small
- **Approximation** (Sacrifice quality)
  - Quickly find a solution that is *provably not very bad*
- **Local search**
  - Quickly find a solution for which you cannot give any quality guarantee (but which might often be good in practice on real problem instances)
- **Restriction**
  - By restricting the structure of the input (e.g., to planar graphs, 2SAT), faster algorithms are usually possible.
- **Randomization**
  - Use randomness to get a faster average running time and allow the algorithm to fail to find optimum with some small probability.

Interested to Know More?

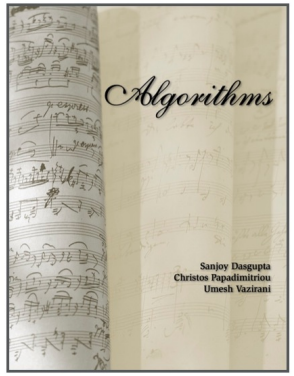
- Here, we have just scratched the surface!



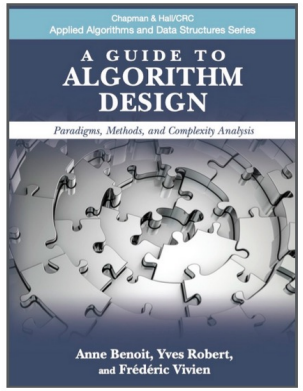
Chapter 34:
NPC
Chapter 35:
Approximation



Chapter 8: NPC
Chapter 9-13: Coping with NPC
(Approximation, local search, randomized algorithms, etc.)



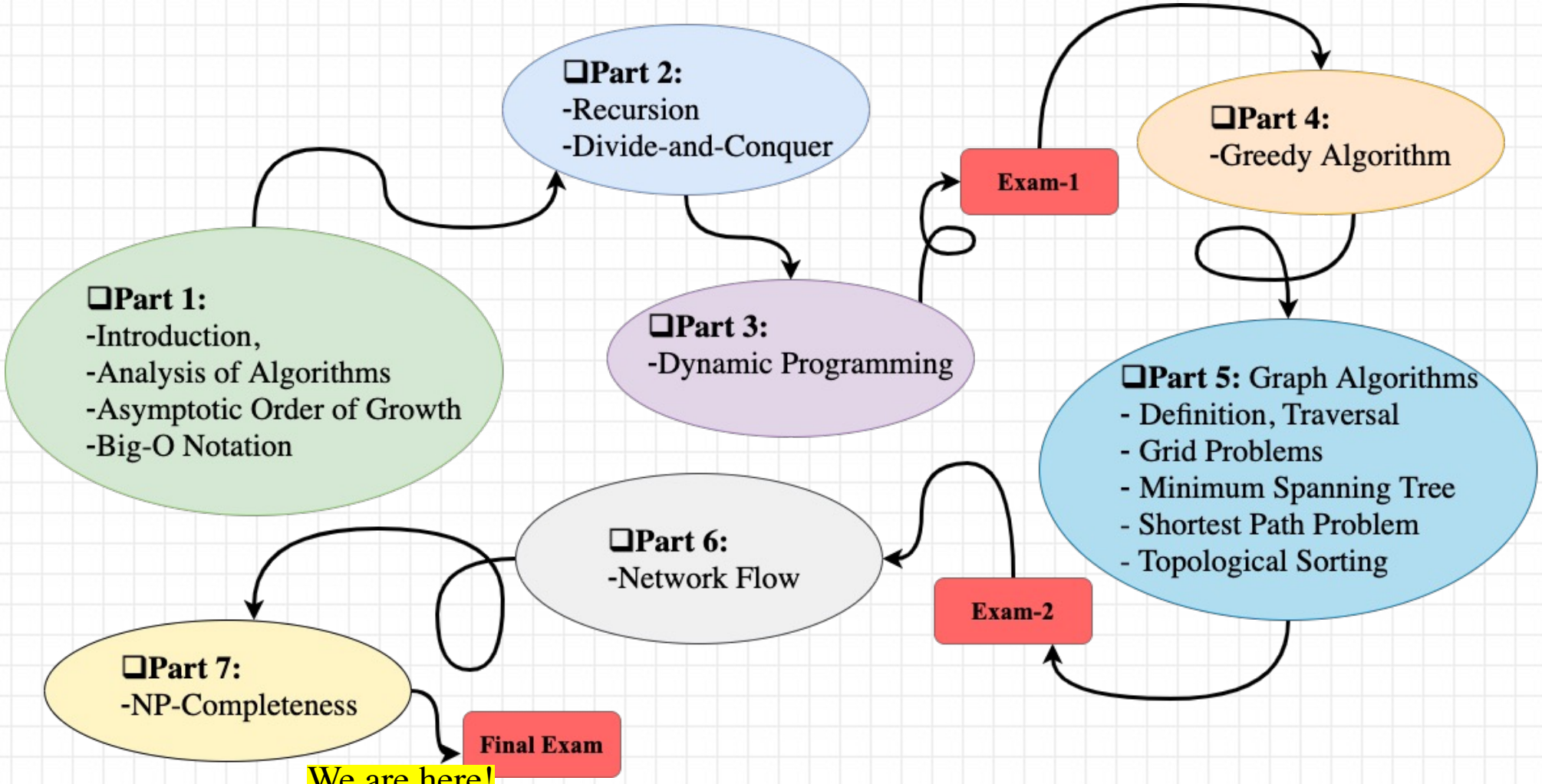
Chapter 8: NPC
Chapter 9: Coping with NPC



Part II: NPC and beyond

Chapter 6: NPC
Chapter 7: NPC Exercises
Chapter 8: Beyond NPC
Chapter 9: Beyond NPC Exercises

Roadmap


❑Part 1:
-Introduction,
-Analysis of Algorithms
-Asymptotic Order of Growth
-Big-O Notation
❑Part 2:
-Recursion
-Divide-and-Conquer
❑Part 3:
-Dynamic Programming
Exam-1
❑Part 4:
-Greedy Algorithm
❑Part 5: Graph Algorithms
- Definition, Traversal
- Grid Problems
- Minimum Spanning Tree
- Shortest Path Problem
- Topological Sorting
Exam-2
❑Part 6:
-Network Flow
❑Part 7:
-NP-Completeness
Final Exam
We are here!

References

- The lecture slides are heavily based on the suggested textbooks and the corresponding published lecture notes:
  - Slides by Umit Catalyurek, Georgia Institute of Technology.
    (Based on slides by Bistra Dilkina, Anne Benoit, Jennifer Welch, George Bebis, and Kevin Wayne)
  - CLRS: Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. Introduction to Algorithms, Third Edition, MIT Press, 2009.
  - KT: Kleinberg, J., & Tardos, E. Algorithm design. Pearson/Addison-Wesley, 2006.