# Algorithms for Decision Support

## NP-Completeness (2/3)

NP-Completeness

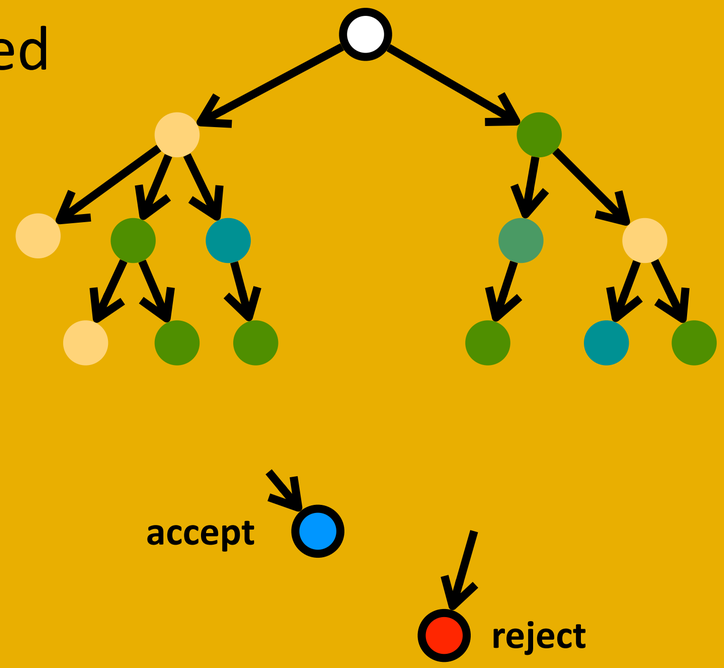
# Turing machine


- An infinitely long tape/memory
  - Initially contains the (finite) input sequence and is blank everywhere else
- A tape head that can read and write symbols and move around on the tape
- Finite-state control

  - The Turing machine may end up with an *accept* state or *reject* state
  - It *accepts* the input or *rejects* the input

# Non-Deterministic Turing machine


- Like the (deterministic) Turing machine, but have non-deterministic behavior
- If there is a path ends at an accept state, the input is accepted



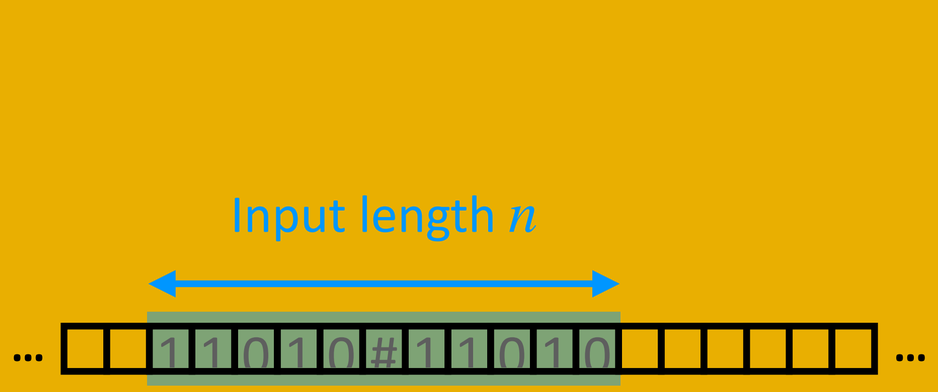
# Formal Language Framework

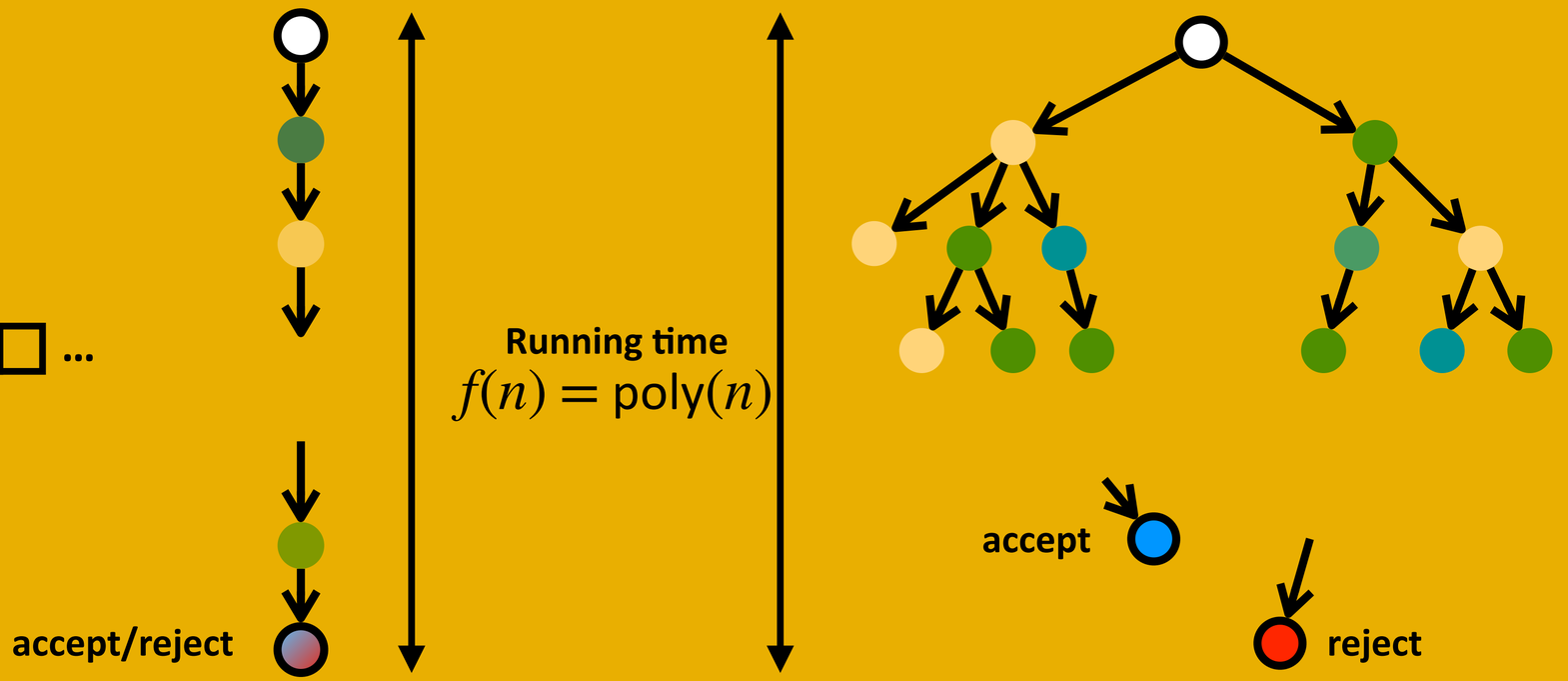
- Following the vein of Turing machine concept,  a **language** is a set of **strings**
  - Language ⇔ **problem**
  - String ⇔ **instance**
  - Asking if a string is in a language
    ⇔ if the instance satisfies the property that the problem asks

- Given a problem/language, a instance/string is a
  - yes instance: an instance that satisfies the property that the problem asks
  - no instance: an instance that does not satisfy the property that the problem asks

# Class **P** and Class **NP**

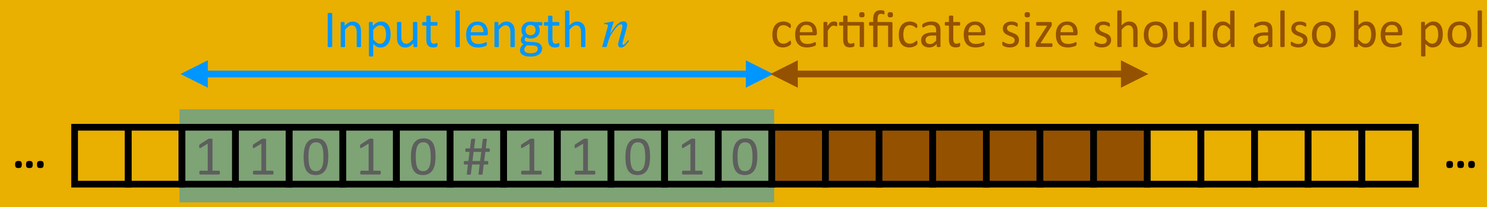
- The class **P** is the class of languages that are accepted or rejected in polynomial time by a *deterministic* Turing machine
- The class **NP** is the class of languages that are accepted or rejected in polynomial time by a *non-deterministic* Turing machine.



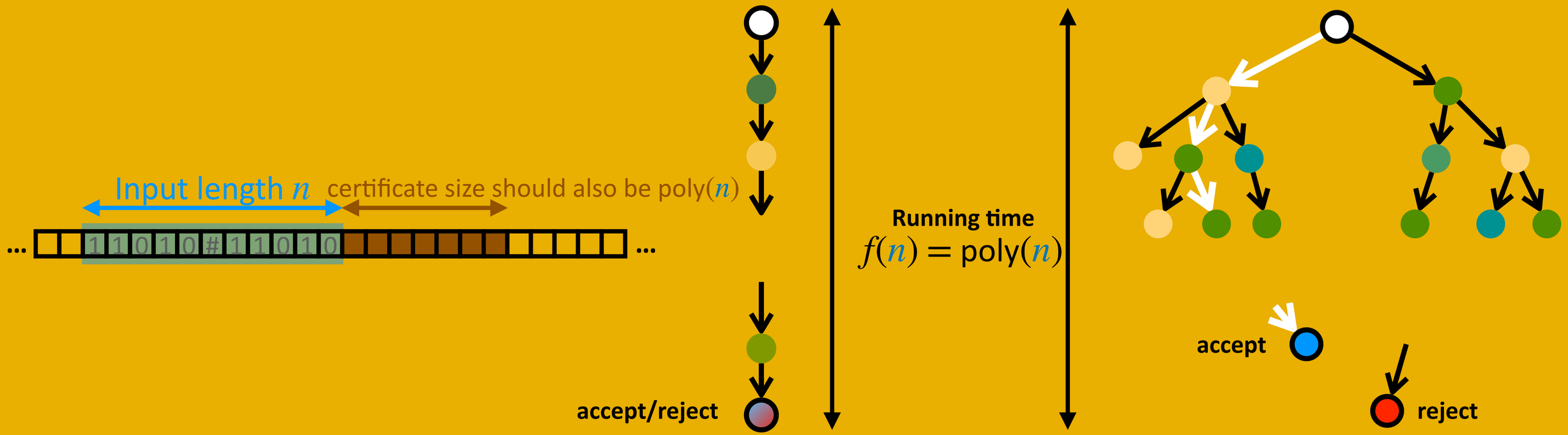
# Certificate and (Polynomial-time) Verify

- A language $A$ is **verifiable** if for any of its yes-instances $w$, there exists a piece of hint (certificate) $c$ such that using this hint $c$, one can be convinced that $w$ is indeed a yes-instance of $A$
  - Only yes-instances have certificates
- **Polynomial-time verifiable**: the verification can be done in time of polynomial in input length
  - The hint size should also be polynomial
  - It does NOT mean that the hint $c$ should be constructed within polynomial time!



# Class **NP** Alternative Definition

- The class **P** is the class of languages that are *accepted* or *rejected* in polynomial time by a deterministic Turing machine
- The class **NP** is the class of languages that can be *verified* in polynomial time by a deterministic Turing machine.

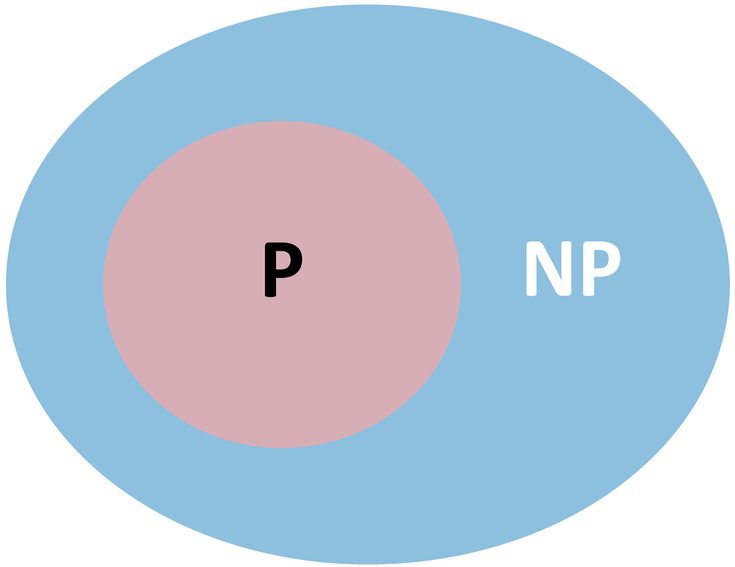


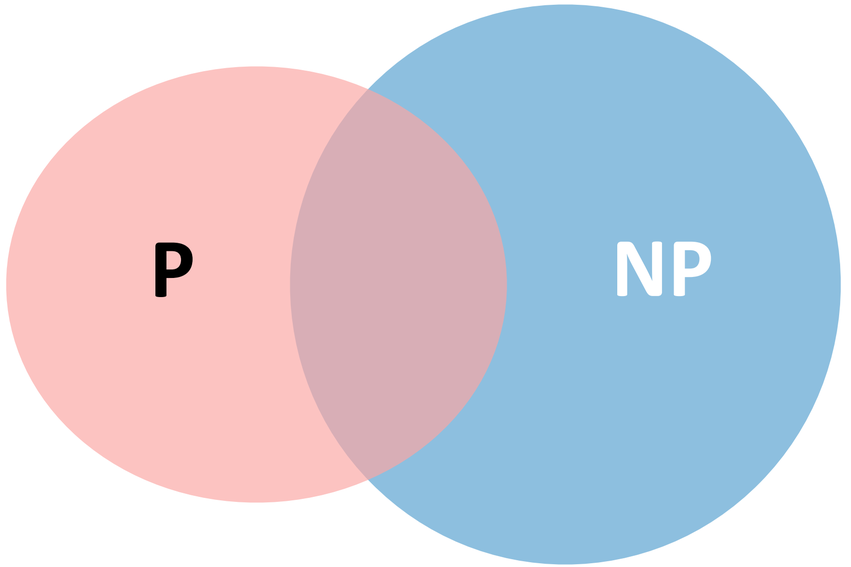
# Prove **NP** Membership

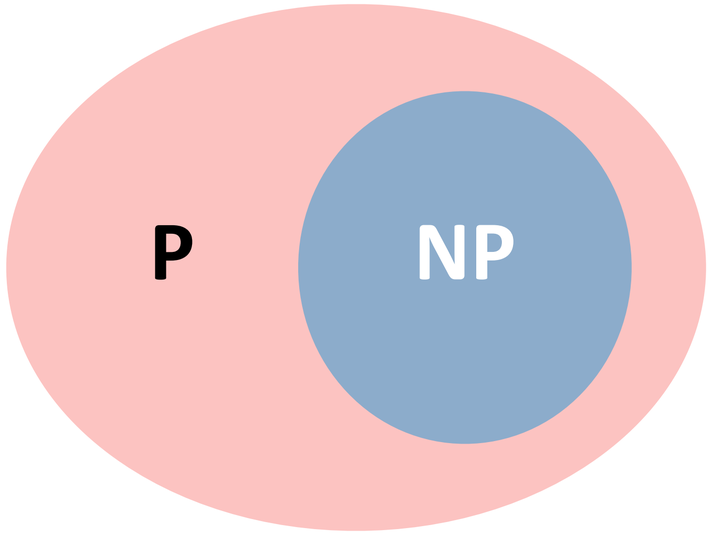
- To show that a problem is in **NP**, we can show that it is polynomial-time verifiable

<Proof Idea>

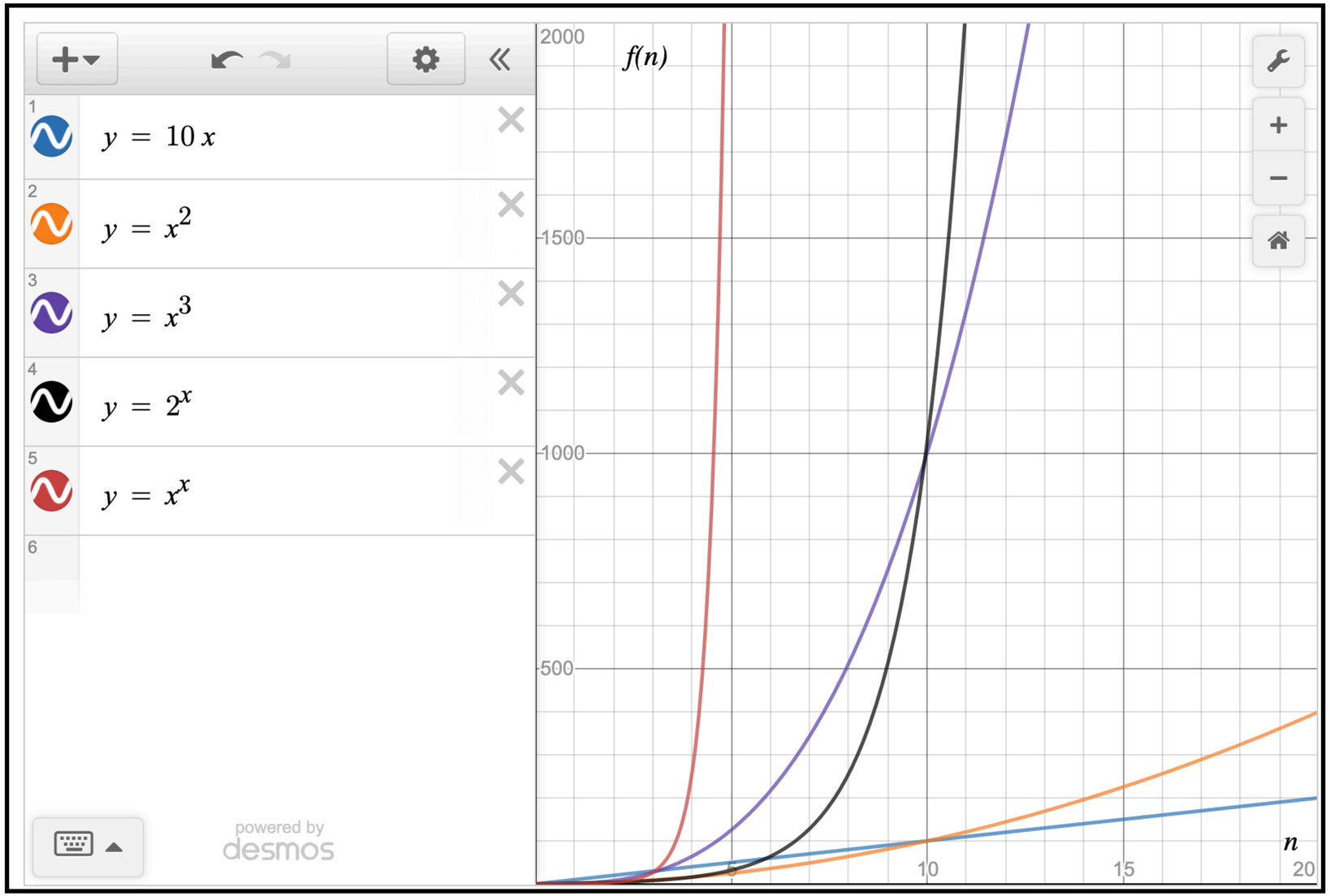
1. Show that for any yes instance $w$, there is a certificate $c$.
2. Design a verifier $V$ on input $\langle w, c \rangle$ that *accepts* all $w \in A$ and *rejects* all $w \notin A$
3. Show that $V$ runs in polynomial time (in the length of $w$)

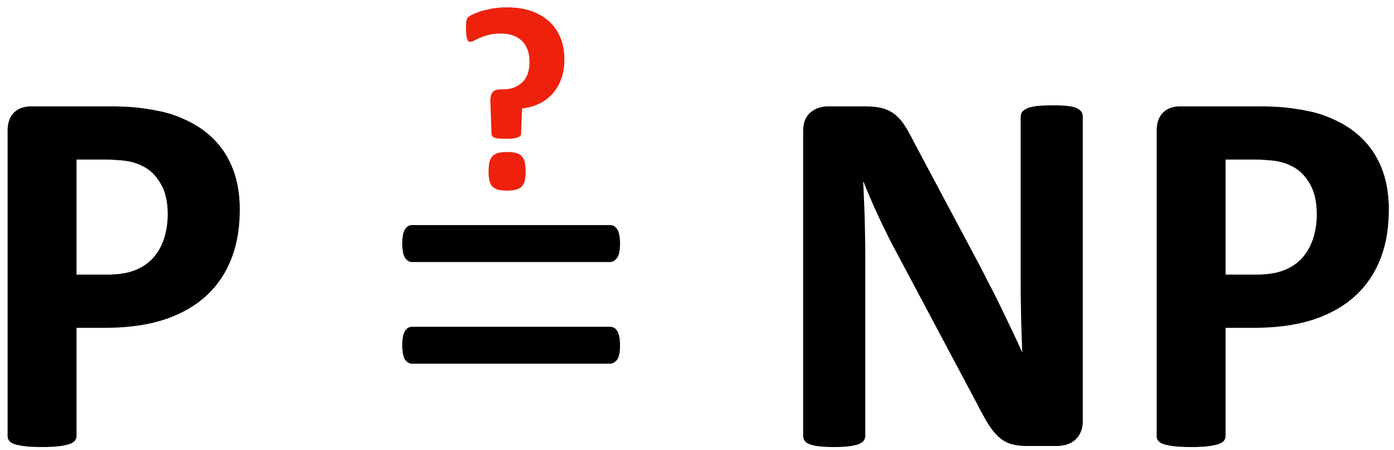

P
NP


P
NP

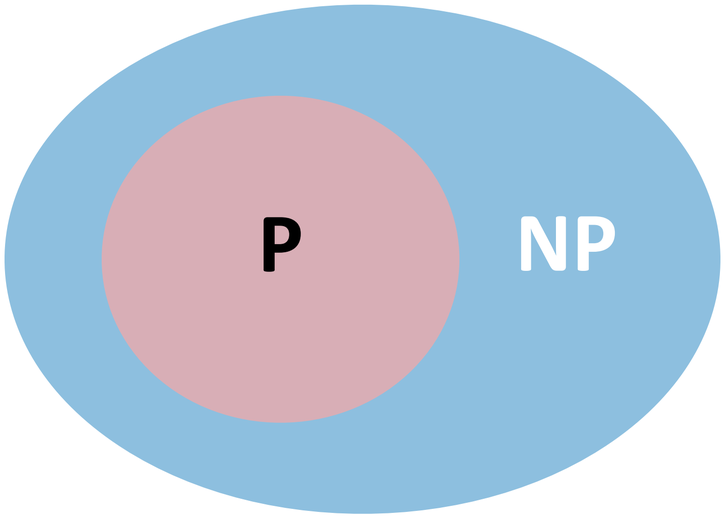

P
NP

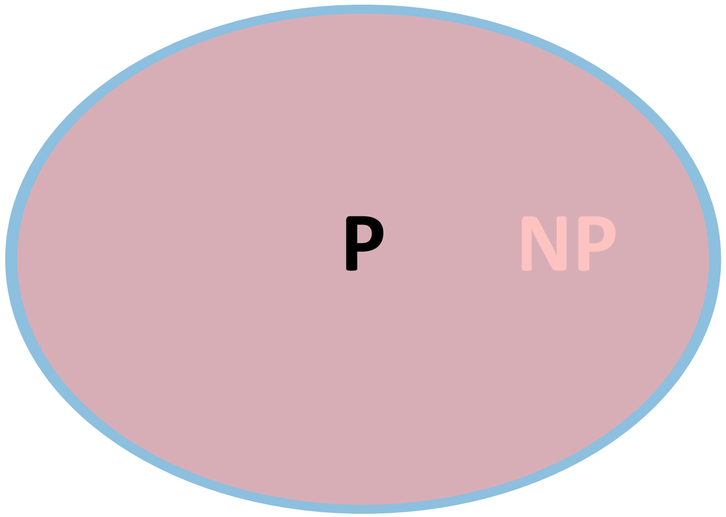
# Exponential vs Polynomial




P ? NP

# P ≟ NP



# Outline

- Cook-Leven Theorem
- NPC
  - Reduction and hardness
    - PARTITION $\leq_p$ 2WAY-PARTITION
  - NP-hard
- 3SAT

# Boolean Formula

- Boolean formula: an expression involving Boolean variables and operations
  - Example:
    - $\phi = \overline{x} \wedge y \wedge z$ $\quad x = \text{FALSE}, y = \text{TRUE}, z = \text{TRUE}$
    - $\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$ $\quad x = \text{FALSE}, y = \text{TRUE}, z = \text{FALSE}$
  - (Boolean) variables: $x$, $y$, $z$
- The Boolean variables can take on the values *TRUE* ($1$) and *FALSE* ($0$)
- A Boolean formula is **satisfiable** if some assignment of *TRUE*s and *FALSE*s to the variables make the formula true
- SAT = $\{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$

# Cook-Levin Theorem

# Cook-Levin Theorem

- In 1971, Stephen Cook published a paper and proposed that there is a problem SAT such that if SAT can be solved (by a deterministic Turing machine) in polynomial time, then all problems in **NP** can be solve in polynomial time.

# Cook-Levin Theorem

- In 1971, Stephen Cook published a paper and proposed that there is a problem SAT such that if SAT can be solved (by a deterministic Turing machine) in polynomial time, then all problems in **NP** can be solve in polynomial time.
  - That is, SAT can be solved in polynomial time only if **P** = **NP**

# Cook-Levin Theorem

- In 1971, Stephen Cook published a paper and proposed that there is a problem SAT such that if SAT can be solved (by a deterministic Turing machine) in polynomial time, then all problems in **NP** can be solve in polynomial time.
  - That is, SAT can be solved in polynomial time only if **P** = **NP**
  - If someone shows that SAT can be solved in polynomial time, then (s)he proves that **P** = **NP**

# Cook-Levin Theorem

- In 1971, Stephen Cook published a paper and proposed that there is a problem SAT such that if SAT can be solved (by a deterministic Turing machine) in polynomial time, then all problems in **NP** can be solve in polynomial time.
  - That is, SAT can be solved in polynomial time only if **P** = **NP**
  - If someone shows that SAT can be solved in polynomial time, then (s)he proves that **P** = **NP**
  - In 1973, Leonid Levin published a paper based on his previous talks and claimed similar theories with the one in Cook's paper

# Cook-Levin Theorem

- In 1971, Stephen Cook published a paper and proposed that there is a problem SAT such that if SAT can be solved (by a deterministic Turing machine) in polynomial time, then all problems in **NP** can be solve in polynomial time.
  - That is, SAT can be solved in polynomial time only if **P** = **NP**
  - If someone shows that SAT can be solved in polynomial time, then (s)he proves that **P** = **NP**
  - In 1973, Leonid Levin published a paper based on his previous talks and claimed similar theories with the one in Cook's paper
- In 1972, Richard Karp published another paper and proved that there are other 21 problems also have the property that if they can be solved in polynomial time, then **P** = **NP**

# Cook-Levin Theorem

- In 1971, Stephen Cook published a paper and proposed that there is a problem SAT such that if SAT can be solved (by a deterministic Turing machine) in polynomial time, then all problems in **NP** can be solve in polynomial time.
  - That is, SAT can be solved in polynomial time only if **P** = **NP**
  - If someone shows that SAT can be solved in polynomial time, then (s)he proves that **P** = **NP**
  - In 1973, Leonid Levin published a paper based on his previous talks and claimed similar theories with the one in Cook's paper
- In 1972, Richard Karp published another paper and proved that there are other 21 problems also have the property that if they can be solved in polynomial time, then **P** = **NP**
  - These problems form a class **NP-Complete**

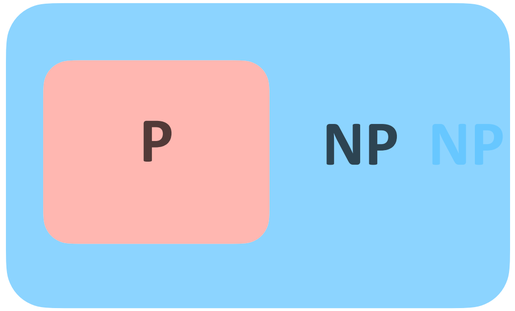
# NP-Complete

# NP-Complete

- The **NP-complete** problems are “the most difficult” ones among all the problems in **NP**

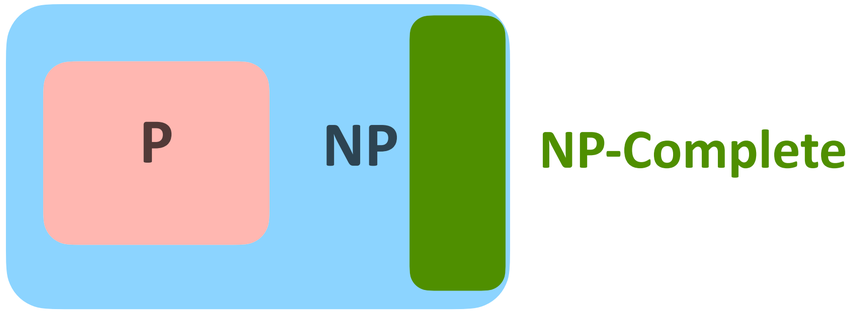
# NP-Complete

- The **NP-complete** problems are “the most difficult” ones among all the problems in **NP**



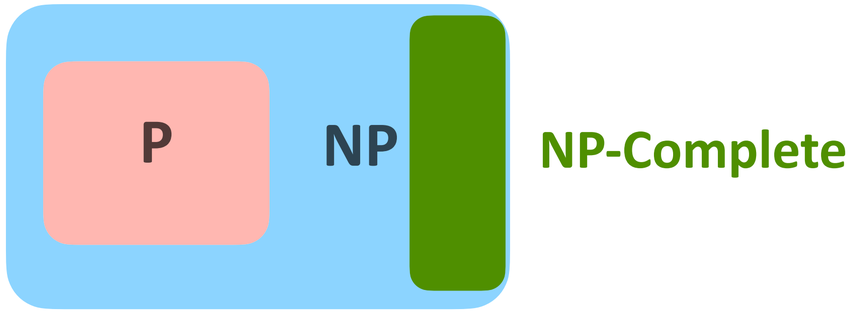
# NP-Complete

- The **NP-complete** problems are “the most difficult” ones among all the problems in **NP**



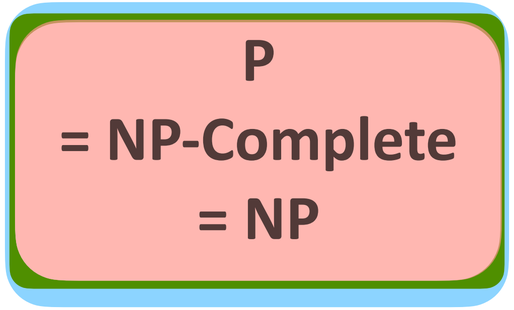
# NP-Complete

- The **NP-complete** problems are “the most difficult” ones among all the problems in **NP**
  - If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



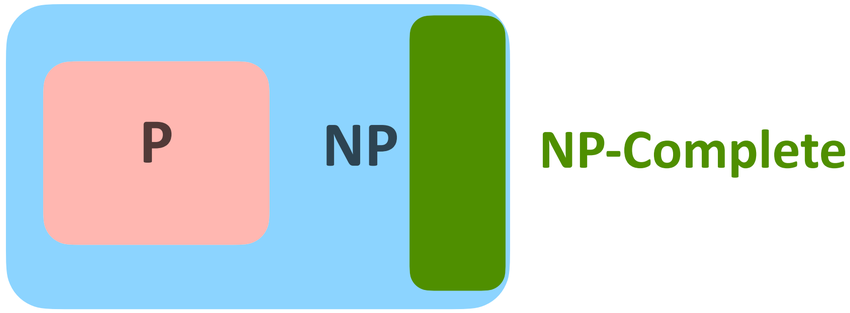
# NP-Complete

- The **NP-complete** problems are “the most difficult” ones among all the problems in **NP**
  - If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



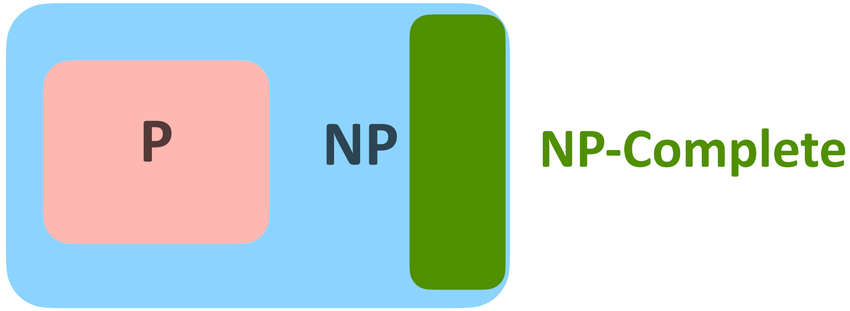
# NP-Complete

- The **NP-complete** problems are “the most difficult” ones among all the problems in **NP**
  - If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time
    - A researcher who attempts to prove that **P** equals **NP** only need to find a polynomial time algorithm for an **NP-complete** problem to achieve this goal



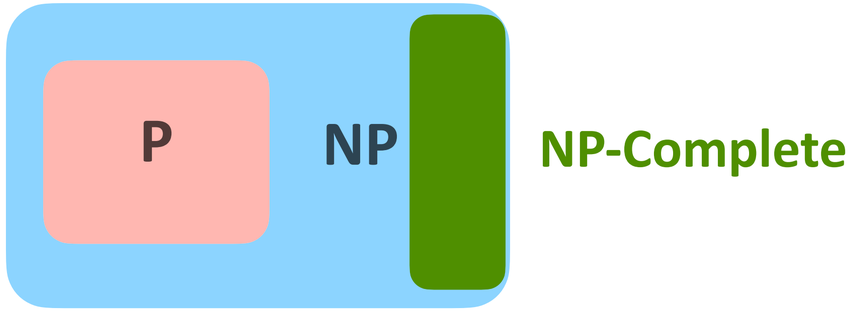
# NP-Complete

- The **NP-complete** problems are “the most difficult” ones among all the problems in **NP**
  - If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time
    - A researcher who attempts to prove that **P** equals **NP** only need to find a polynomial time algorithm for an **NP-complete** problem to achieve this goal
  - If any problem in **NP** requires more than polynomial time, an NP-complete one does



# NP-Complete

- The **NP-complete** problems are “the most difficult” ones among all the problems in **NP**
  - If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time
    - A researcher who attempts to prove that **P** equals **NP** only need to find a polynomial time algorithm for an **NP-complete** problem to achieve this goal
  - If any problem in **NP** requires more than polynomial time, an NP-complete one does
- The phenomenon of NP-completeness may prevent wasting time searching for a nonexistent polynomial time algorithm to solve a particular problem



# NP-Complete

- The **NP-complete** problems are “the most difficult” ones among all the problems in **NP**
  - If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time
    - A researcher who attempts to prove that **P** equals **NP** only need to find a polynomial time algorithm for an **NP-complete** problem to achieve this goal
  - If any problem in **NP** requires more than polynomial time, an NP-complete one does
- The phenomenon of NP-completeness may prevent wasting time searching for a nonexistent polynomial time algorithm to solve a particular problem
- The problems Maximum Clique, Minimum Vertex Cover, Partition, Subset Sum are all NP-complete problems

# How do we know if a problem is “difficult”?

# How do we know if a problem is “difficult”?

# Reduction

# How do we know if a problem is “difficult”?

# Reduction

- We want to solve problem $A$. Instead of solving $A$ directly, we can show that **we are able to solve** $A$ by **using an (existed) algorithm for solving another problem** $B$.

# How do we know if a problem is “difficult”?

# Reduction

- We want to solve problem $A$. Instead of solving $A$ directly, we can show that **we are able to solve** $A$ by **using an (existed) algorithm for solving another problem** $B$. According to the answer to problem $B$, we know the answer to problem $A$.

# How do we know if a problem is “difficult”?

# Reduction

- We want to solve problem $A$. Instead of solving $A$ directly, we can show that **we are able to solve** $A$ by **using an (existed) algorithm for solving another problem** $B$. According to the answer to problem $B$, we know the answer to problem $A$.
  - Ex:
    - Problem $A$: Can I travel to New Zealand

# How do we know if a problem is “difficult”?

# Reduction

- We want to solve problem $A$. Instead of solving $A$ directly, we can show that **we are able to solve** $A$ by **using an (existed) algorithm for solving another problem** $B$. According to the answer to problem $B$, we know the answer to problem $A$.
  - Ex:
    - Problem $A$: Can I travel to New Zealand
    - Problem $B$: Do I earn enough money

# How do we know if a problem is “difficult”?

# Reduction

- We want to solve problem $A$. Instead of solving $A$ directly, we can show that **we are able to solve** $A$ by **using an (existed) algorithm for solving another problem** $B$. According to the answer to problem $B$, we know the answer to problem $A$.
  - Ex:
    - Problem $A$: Can I travel to New Zealand
    - Problem $B$: Do I earn enough money
    - If I earn enough money, I can travel to New Zealand;

# How do we know if a problem is “difficult”?

# Reduction

- We want to solve problem $A$. Instead of solving $A$ directly, we can show that **we are able to solve** $A$ by **using an (existed) algorithm for solving another problem** $B$. According to the answer to problem $B$, we know the answer to problem $A$.
  - Ex:
    - Problem $A$: Can I travel to New Zealand
    - Problem $B$: Do I earn enough money
    - If I earn enough money, I can travel to New Zealand;
      If I don’t have enough money, I cannot travel to New Zealand.

# How do we know if a problem is “difficult”?

# Reduction

- We want to solve problem $A$. Instead of solving $A$ directly, we can show that **we are able to solve** $A$ by **using an (existed) algorithm for solving another problem** $B$. According to the answer to problem $B$, we know the answer to problem $A$.
  - Ex:
    - Problem $A$: Can I travel to New Zealand
    - Problem $B$: Do I earn enough money
    - If I earn enough money, I can travel to New Zealand;
      If I don’t have enough money, I cannot travel to New Zealand.
      ($\leftrightarrow$ If I travel to New Zealand, I must have enough money.)


# How do we know if a problem is “difficult”?

# Reduction

- We want to solve problem $A$

  - Instead of solving $A$ directly, we can show that **we are able to solve** $A$ by **using an (existed) algorithm for solving another problem** $B$

# How do we know if a problem is “difficult”?

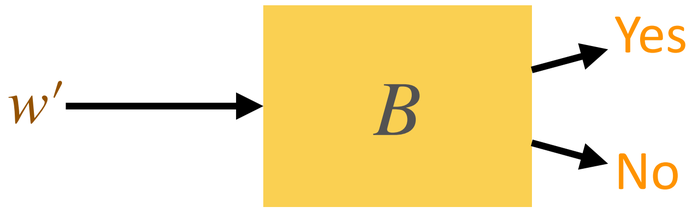
# Reduction

- We want to solve problem $A$
  - That is, given any instance $w$, we want to answer yes if $w \in A$ and answer no otherwise
  - Instead of solving $A$ directly, we can show that **we are able to solve** $A$ by **using an (existed) algorithm for solving another problem** $B$

# How do we know if a problem is “difficult”?

# Reduction

- We want to solve problem $A$
  - That is, given any instance $w$, we want to answer yes if $w \in A$ and answer no otherwise
  - Instead of solving $A$ directly, we can show that **we are able to solve $A$** by **using an (existed) algorithm for solving another problem $B$**
    - The $B$-solver (algorithm for solving $B$) returns yes if the input $w' \in B$ and returns no if $w' \notin B$



# How do we know if a problem is “difficult”?

# Reduction

- We want to solve problem $A$
  - That is, given any instance $w$, we want to answer yes if $w \in A$ and answer no otherwise
  - Instead of solving $A$ directly, we can show that **we are able to solve** $A$ by **using an (existed) algorithm for solving another problem** $B$
    - The $B$-solver (algorithm for solving $B$) returns yes if the input $w' \in B$ and returns no if $w' \notin B$
    - This $B$-solver might be hypothetical

# Reduction

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$

# Reduction

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$

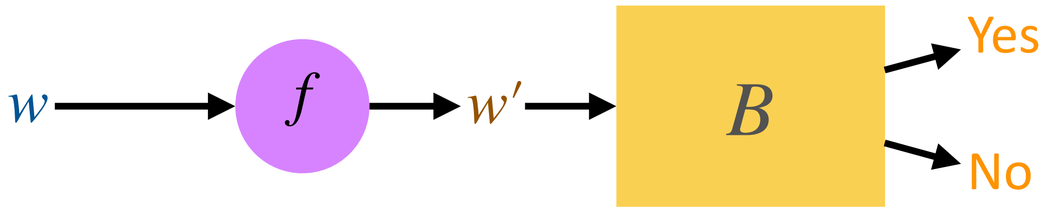
# Reduction

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$

$w$

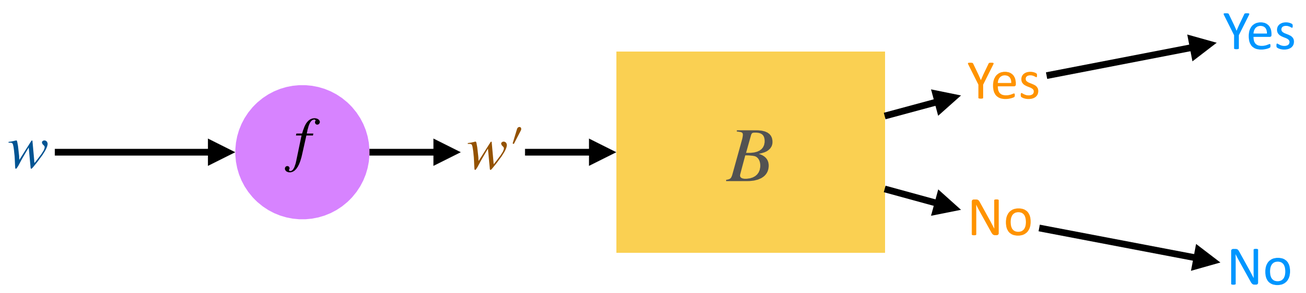
# Reduction

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$



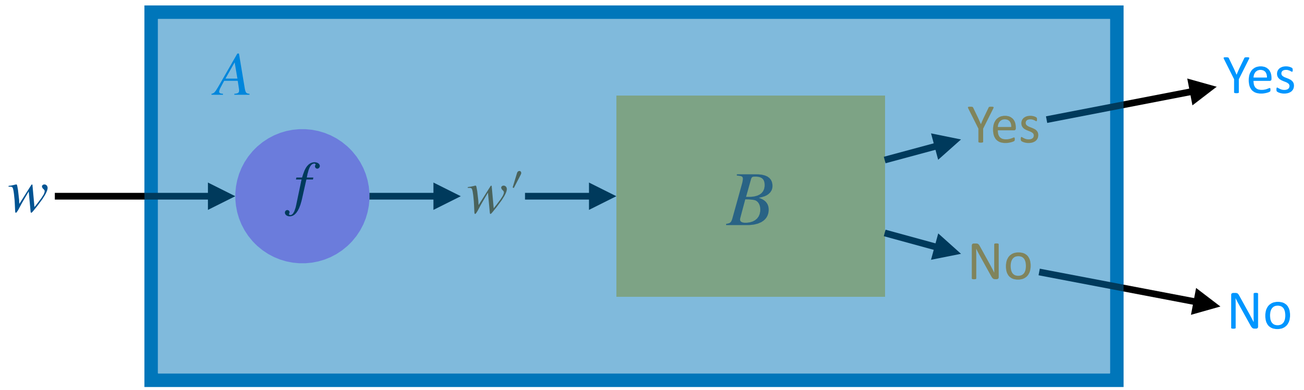
# Reduction

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$



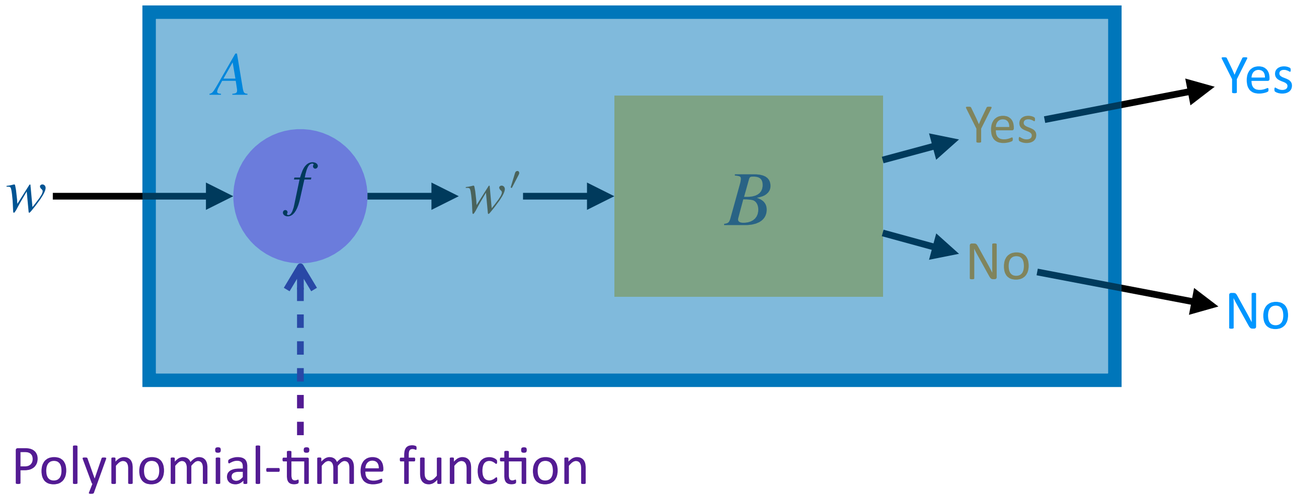
# Reduction

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$



# Polynomial-Time Reduction $A \leq_p B$

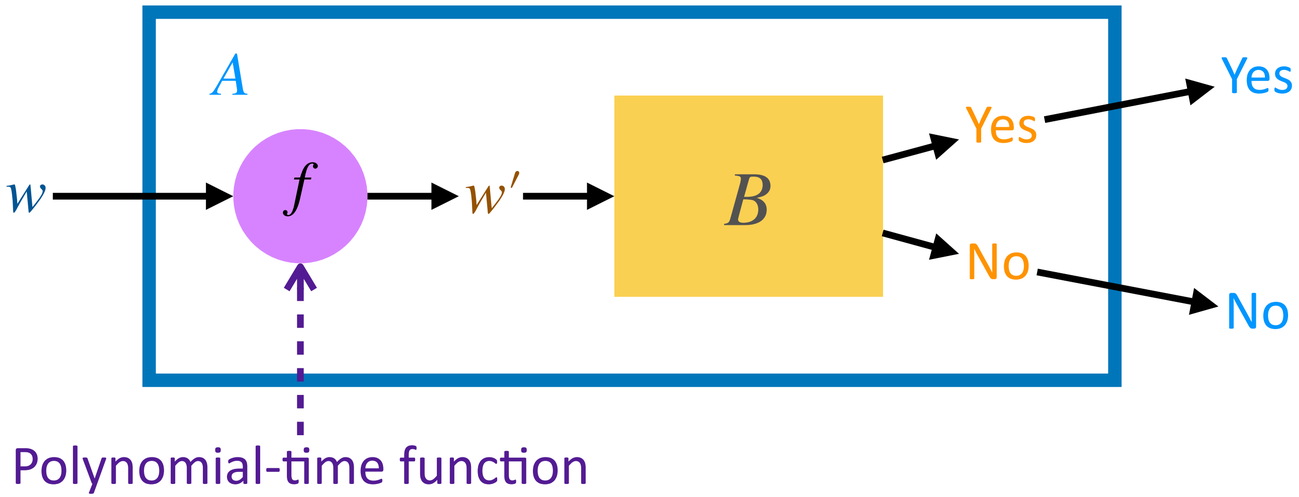
- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$



# Polynomial-Time Reduction $A \leq_p B$

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$

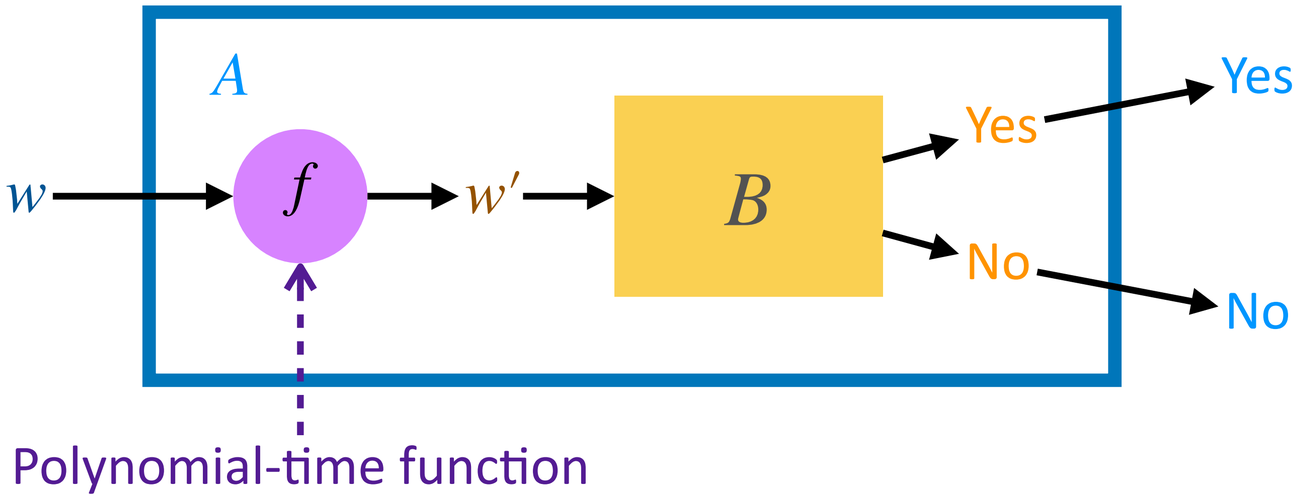
- The sun rises in the east on day $w$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$



# Polynomial-Time Reduction $A \leq_p B$

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$

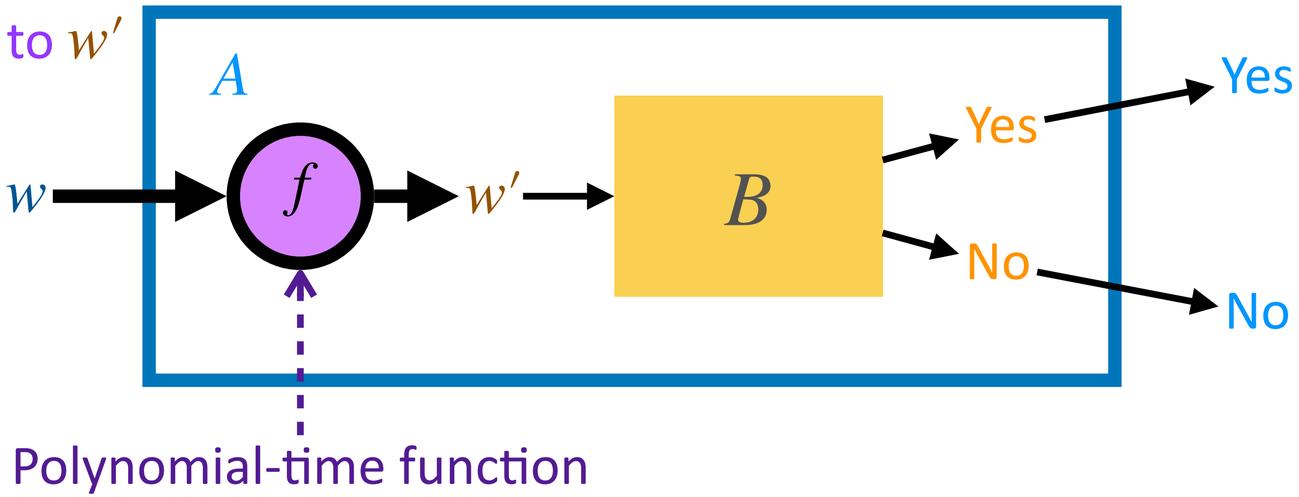
- The sun rises in the east on day $w$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$



# Polynomial-Time Reduction $A \leq_p B$

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$

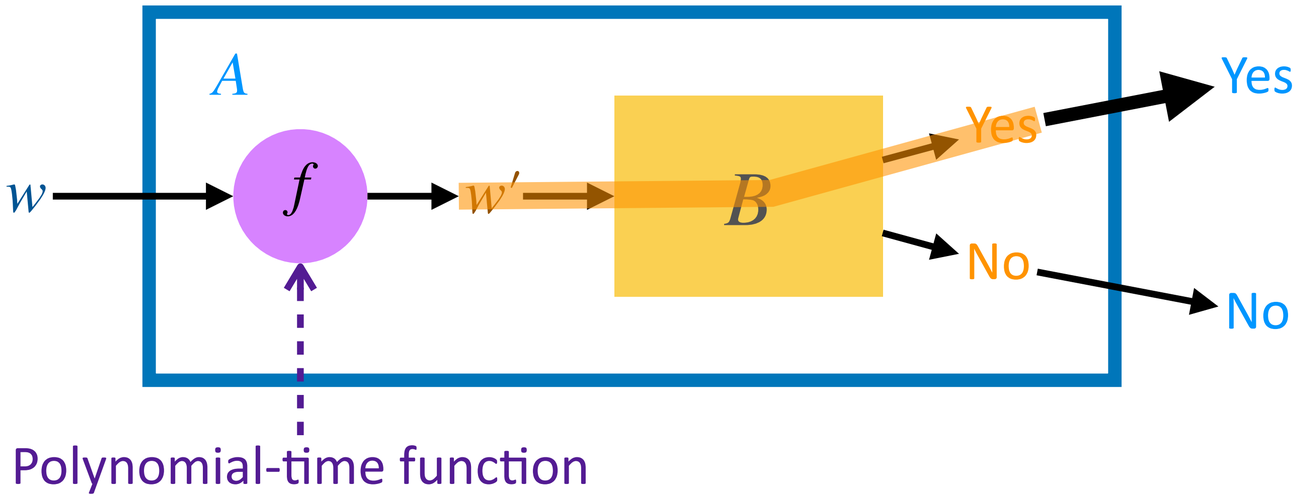
1. Show that there is a function that transforms every $w$ to $w'$ in polynomial time



# Polynomial-Time Reduction $A \leq_p B$

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$

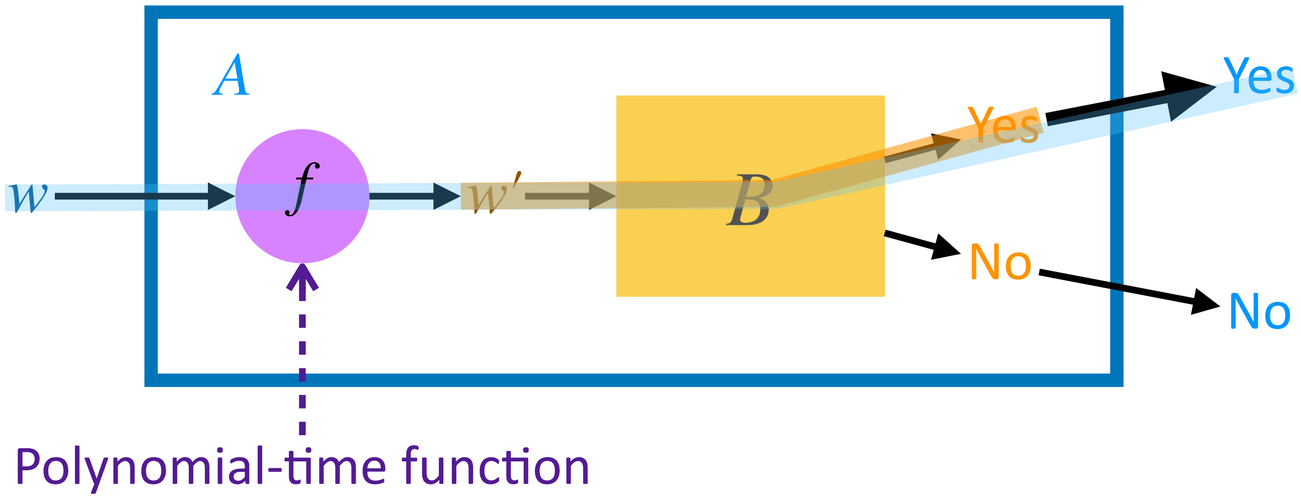
2. Show that for any yes-instance $w' \in B$,



# Polynomial-Time Reduction $A \leq_p B$

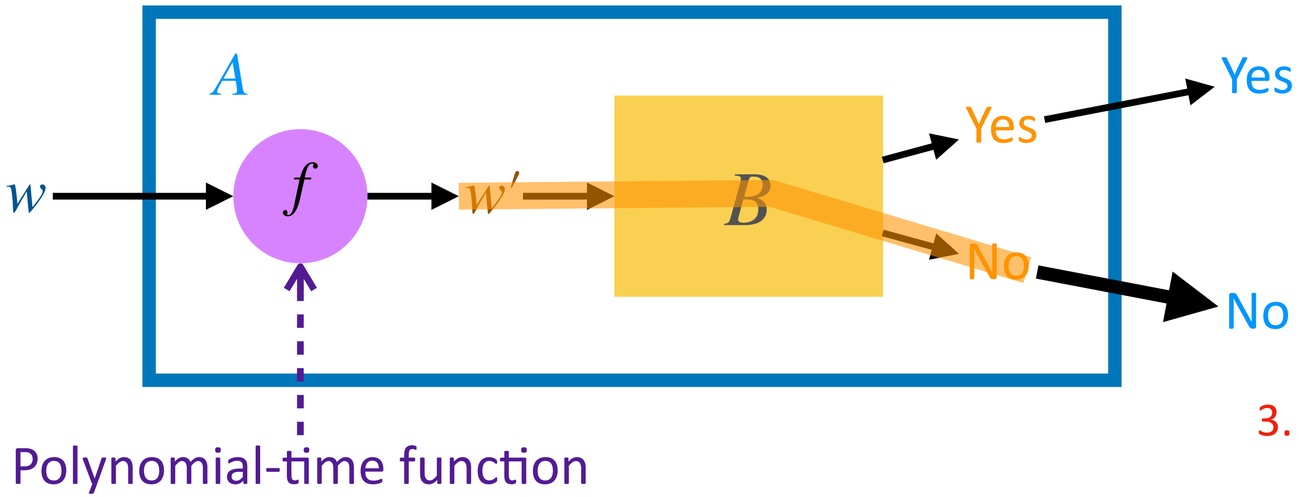
- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$

2. Show that for any yes-instance $w' \in B$, the corresponding instance $w$ is also a yes-instance of $A$



# Polynomial-Time Reduction $A \leq_p B$

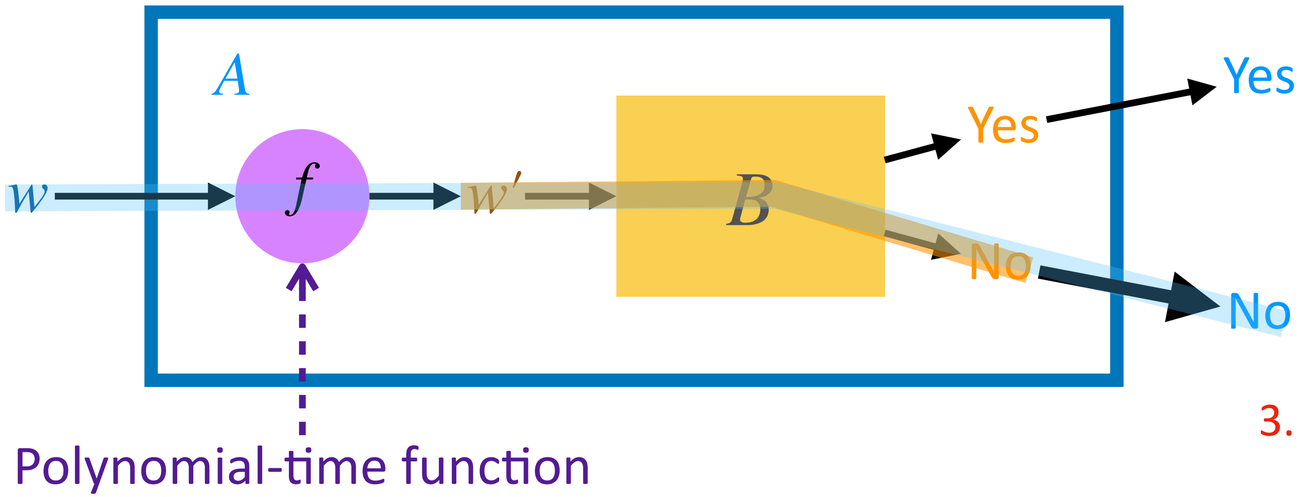
- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$




3. Show that for any no-instance $w' \notin B$,

# Polynomial-Time Reduction $A \leq_p B$

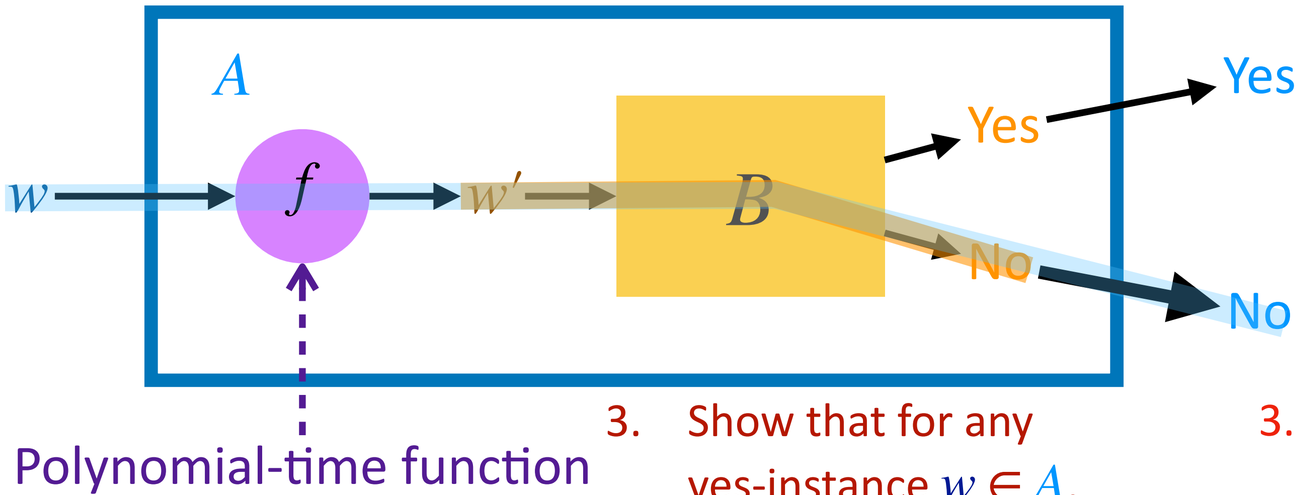
- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$




3. Show that for any no-instance $w' \notin B$, the corresponding instance $w$ is also a no-instance of $A$

# Polynomial-Time Reduction $A \leq_p B$

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$



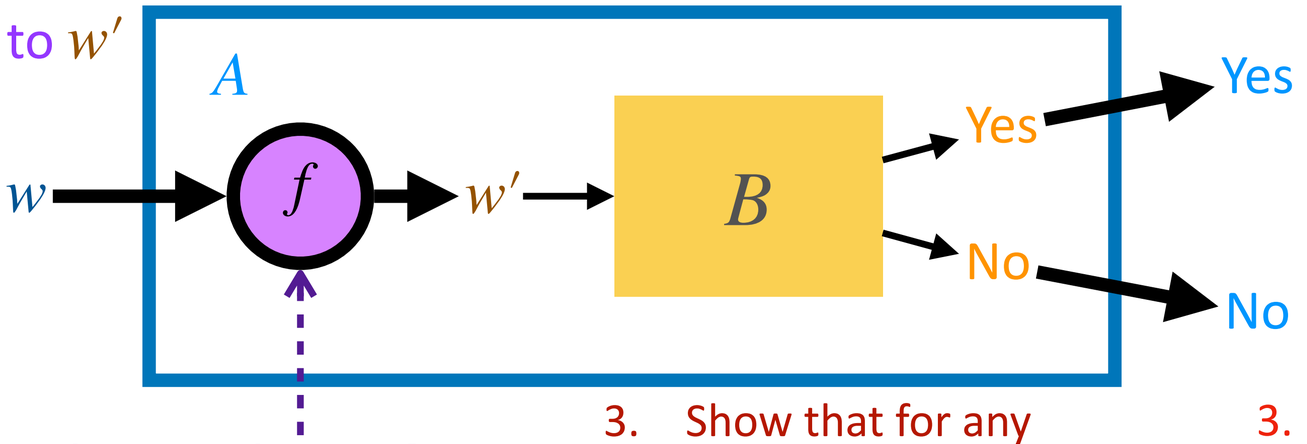

3. Show that for any yes-instance $w \in A$, the corresponding instance $w'$ is also a yes-instance of $B$

3. Show that for any no-instance $w' \notin B$, the corresponding instance $w$ is also a no-instance of $A$

# Polynomial-Time Reduction $A \leq_p B$

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$

1. Show that there is a function that transforms every $w$ to $w'$ in polynomial time




2. Show that for any yes-instance $w' \in B$, the corresponding instance $w$ is also a yes-instance of $A$

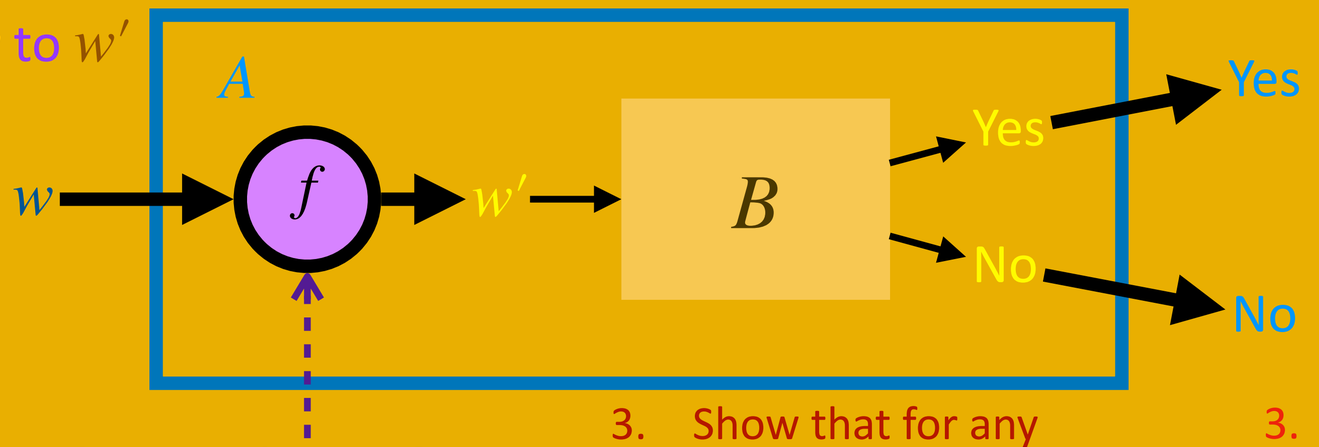
3. Show that for any yes-instance $w \in A$, the corresponding instance $w'$ is also a yes-instance of $B$

3. Show that for any no-instance $w' \notin B$, the corresponding instance $w$ is also a no-instance of $A$

# Polynomial-Time Reduction $A \leq_p B$

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$

1. Show that there is a function that transforms every $w$ to $w'$ in polynomial time




2. Show that for any yes-instance $w' \in B$, the corresponding instance $w$ is also a yes-instance of $A$

3. Show that for any yes-instance $w \in A$, the corresponding instance $w'$ is also a yes-instance of $B$

3. Show that for any no-instance $w' \notin B$, the corresponding instance $w$ is also a no-instance of $A$

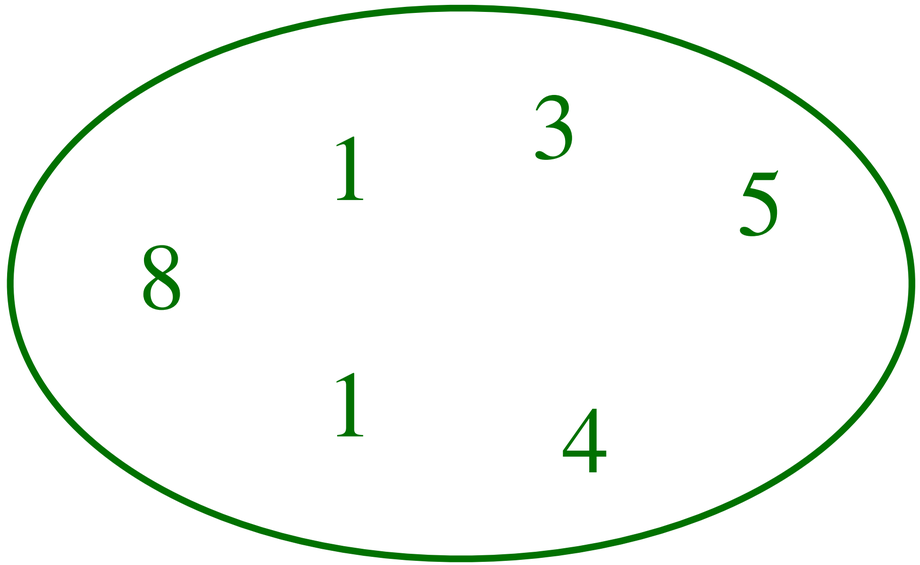
# Ex: Reduce PARTITION to 2WAY-PARTITION

# PARTITION

- PARTITION $= \{\langle S \rangle \mid S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$

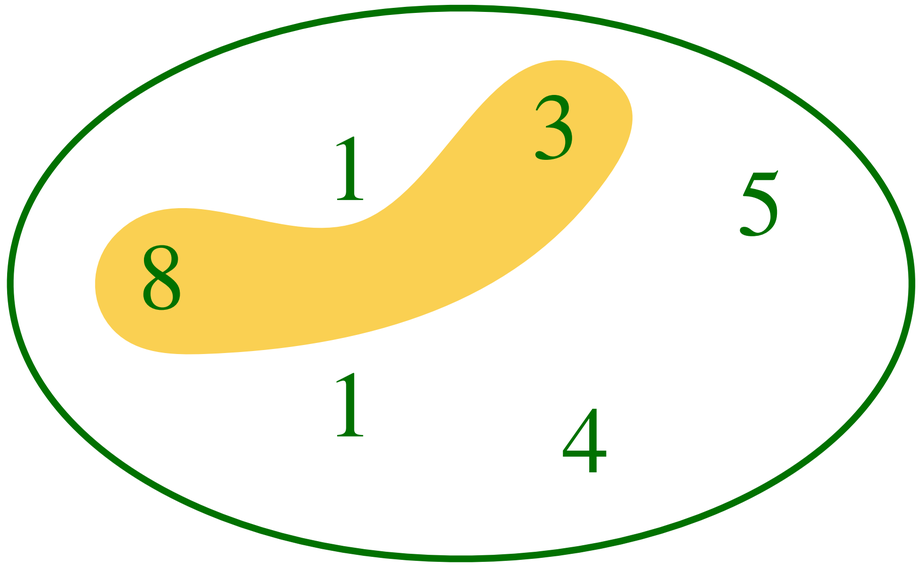
# PARTITION

- PARTITION $= \{\langle S \rangle \mid S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
  - Ex: $S$ = {1, 1, 3, 4, 5, 8}



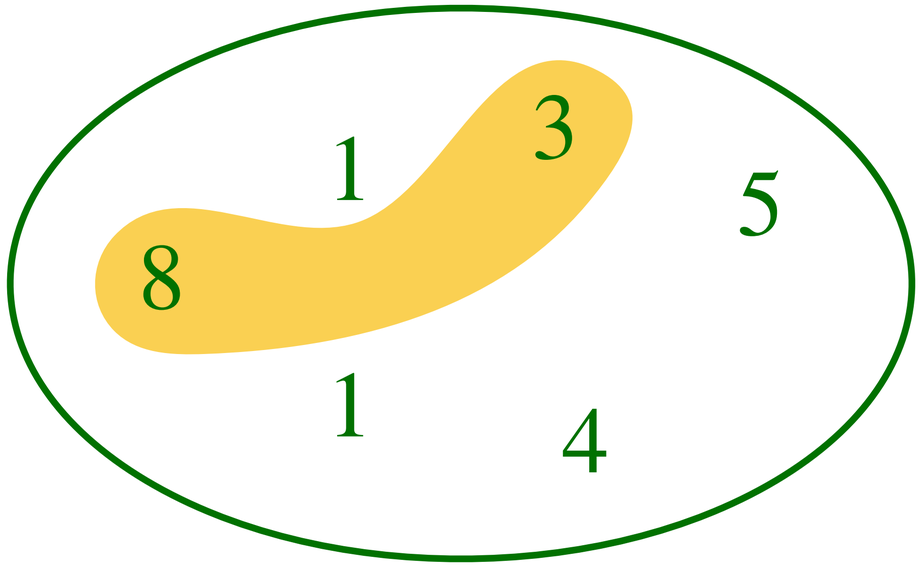
# PARTITION

- PARTITION $= \{\langle S \rangle \mid S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
  - Ex: $S$ = {1, 1, **3**, 4, 5, **8**} $\Rightarrow$ $T$ = {**3**, **8**}



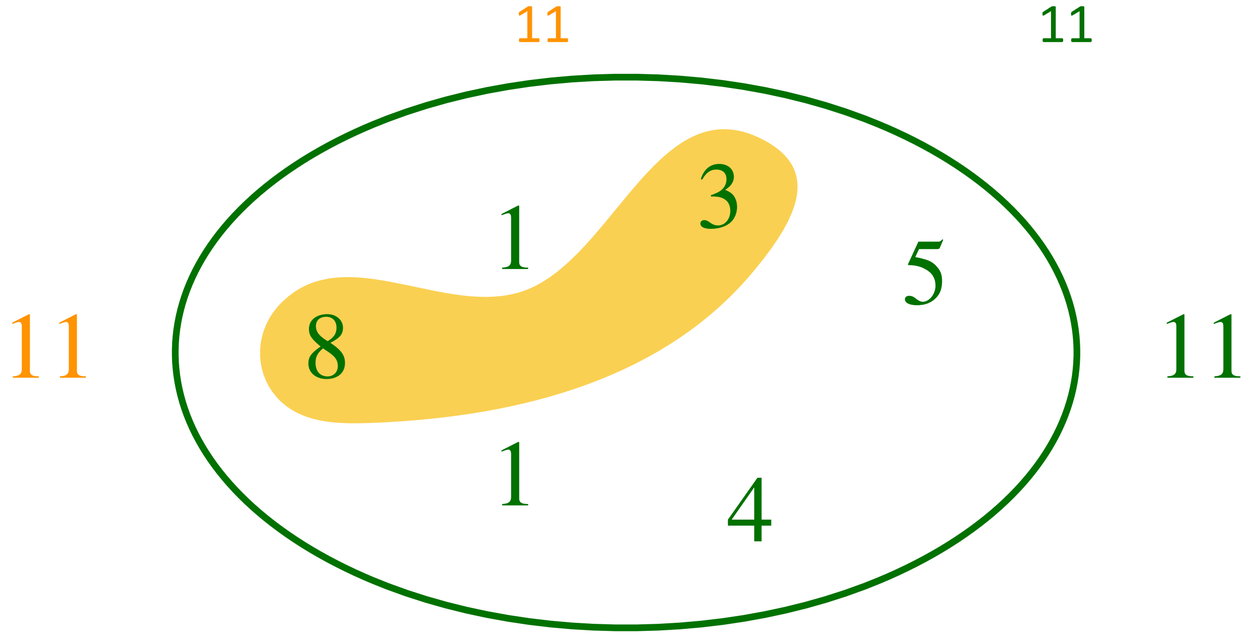
# PARTITION

- PARTITION $= \{\langle S \rangle \mid S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
  - Ex: $S$ = {1, 1, **3**, 4, 5, **8**} $\Rightarrow$ $T$ = {**3**, **8**} and $S \setminus T$ = {1, 1, 4, 5}



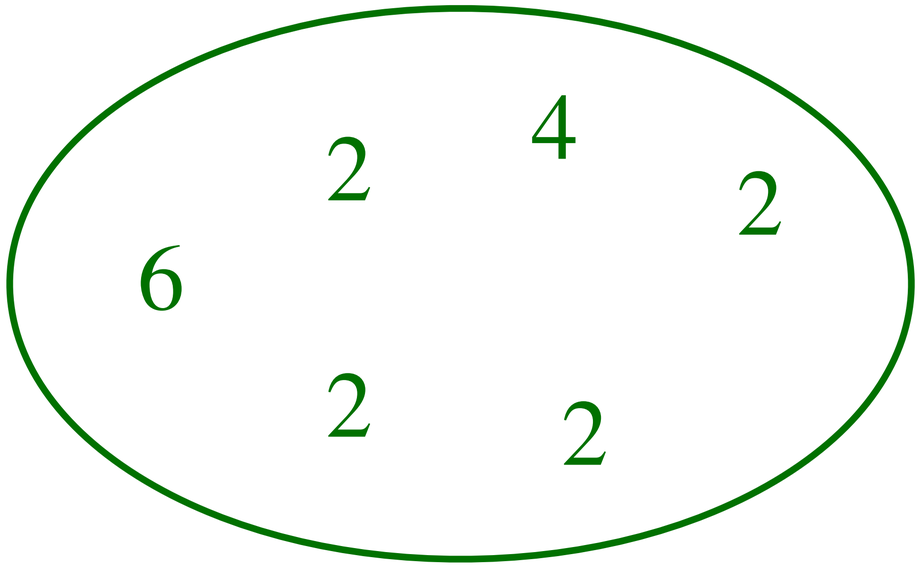
# PARTITION

- PARTITION $= \{\langle S \rangle \mid S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
  - Ex: $S$ = {1, 1, **3**, 4, 5, **8**} $\Rightarrow$ $T$ = {**3**, **8**} and $S \setminus T$ = {1, 1, 4, 5} ✅ Yes-instance



# PARTITION

- PARTITION $= \{\langle S \rangle \mid S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
  - Ex: $S$ = {1, 1, 3, 4, 5, 8} $\Rightarrow$ $T$ = $\underbrace{\{3, 8\}}_{11}$ and $S \setminus T$ = $\underbrace{\{1, 1, 4, 5\}}_{11}$ Yes-instance
  - Ex: $S$ = {2, 2, 2, 2, 4, 6} $\Rightarrow$ No answer



# PARTITION

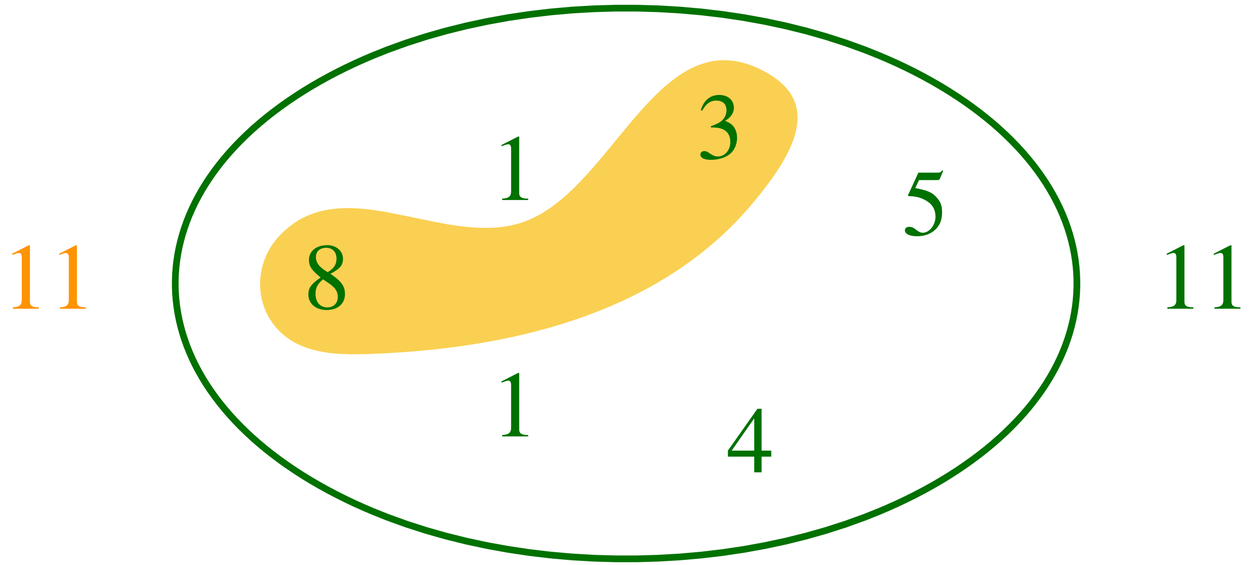
- PARTITION $= \{\langle S \rangle \mid S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
  - Ex: $S$ = {1, 1, 3, 4, 5, 8} $\Rightarrow$ $T$ = $\underbrace{\{3, 8\}}_{11}$ and $S \setminus T$ = $\underbrace{\{1, 1, 4, 5\}}_{11}$ Yes-instance
  - Ex: $S$ = {2, 2, 2, 2, 4, 6} $\Rightarrow$ No answer No-instance

# 2WAY-PARTITION

- 2WAY-PARTITION $= \{\langle S, t\rangle \mid S = \{x_1, \cdots, x_k\}$ where exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$ and $|T| \leq t$ and $|S \setminus T| \leq t\}$

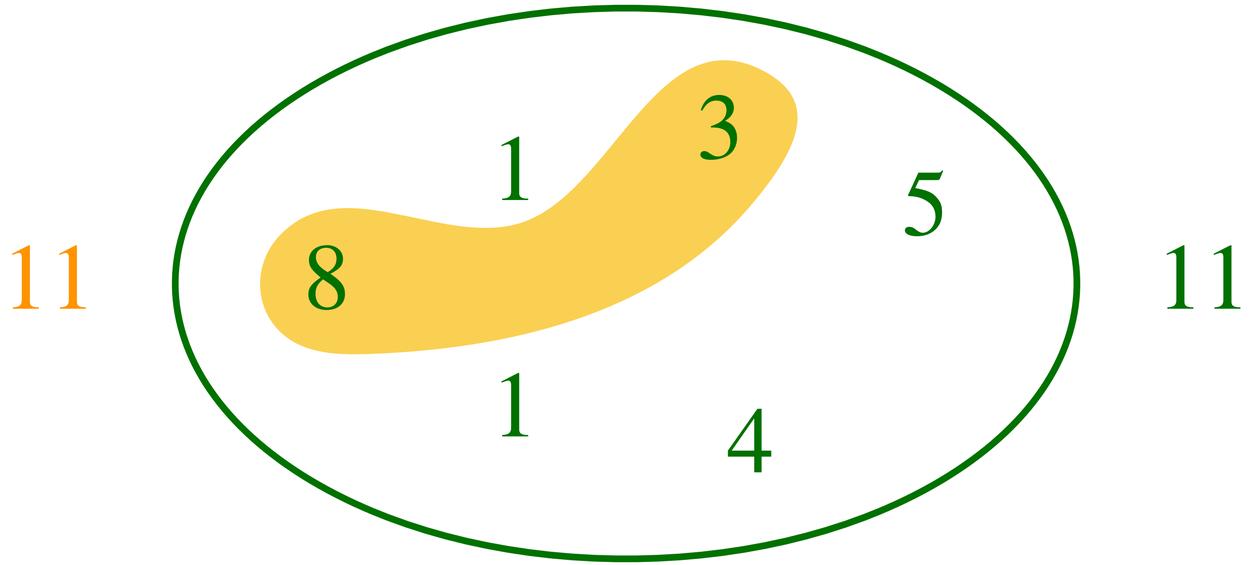
# 2WAY-PARTITION

- 2WAY-PARTITION $= \{\langle S, t\rangle \mid S = \{x_1, \cdots, x_k\}$ where exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$ and $|T| \leq t$ and $|S \setminus T| \leq t\}$
  - Ex: $S$ = {1, 1, **3**, 4, 5, **8**}, $t$ = 4 ✅ Yes-instance
    $\Rightarrow T$ = {**3**, **8**}, $S \setminus T$ = {1, 1, 4, 5}, $|T| \leq t$, $|S \setminus T| \leq t$



# 2WAY-PARTITION

- 2WAY-PARTITION $= \{\langle S, t\rangle \mid S = \{x_1, \cdots, x_k\}$ where exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$ and $|T| \leq t$ and $|S \setminus T| \leq t\}$
  - Ex: $S$ = {1, 1, **3**, 4, 5, **8**}, $t$ = 3 ✗ No-instance
    $\Rightarrow T$ = {**3**, **8**}, $S \setminus T$ = {1, 1, 4, 5}, $|T| \leq t$, $|S \setminus T| > t$

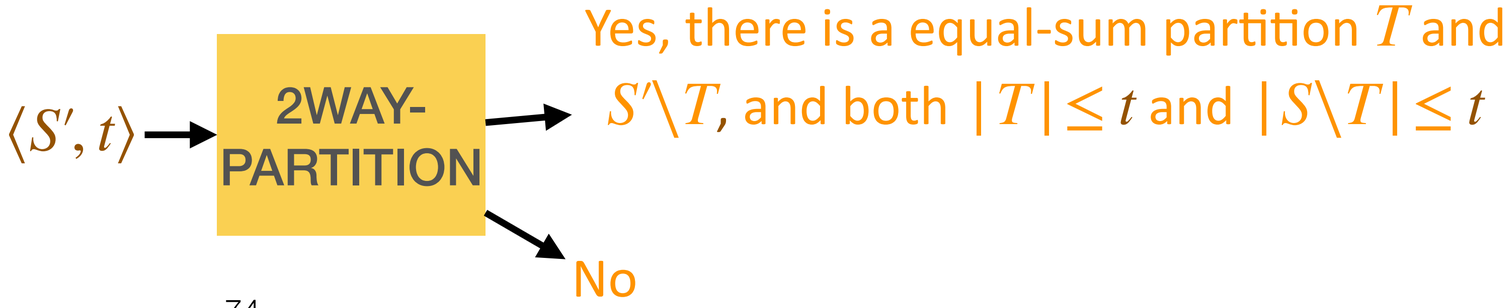


# Ex: Reduce PARTITION to 2WAY-PARTITION

- PARTITION $= \{\langle S \rangle \mid$ $S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
- 2WAY-PARTITION $= \{\langle S, t \rangle \mid$ $S = \{x_1, \cdots, x_k\}$ where exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$ and $|T| \leq t$ and $|S \setminus T| \leq t\}$

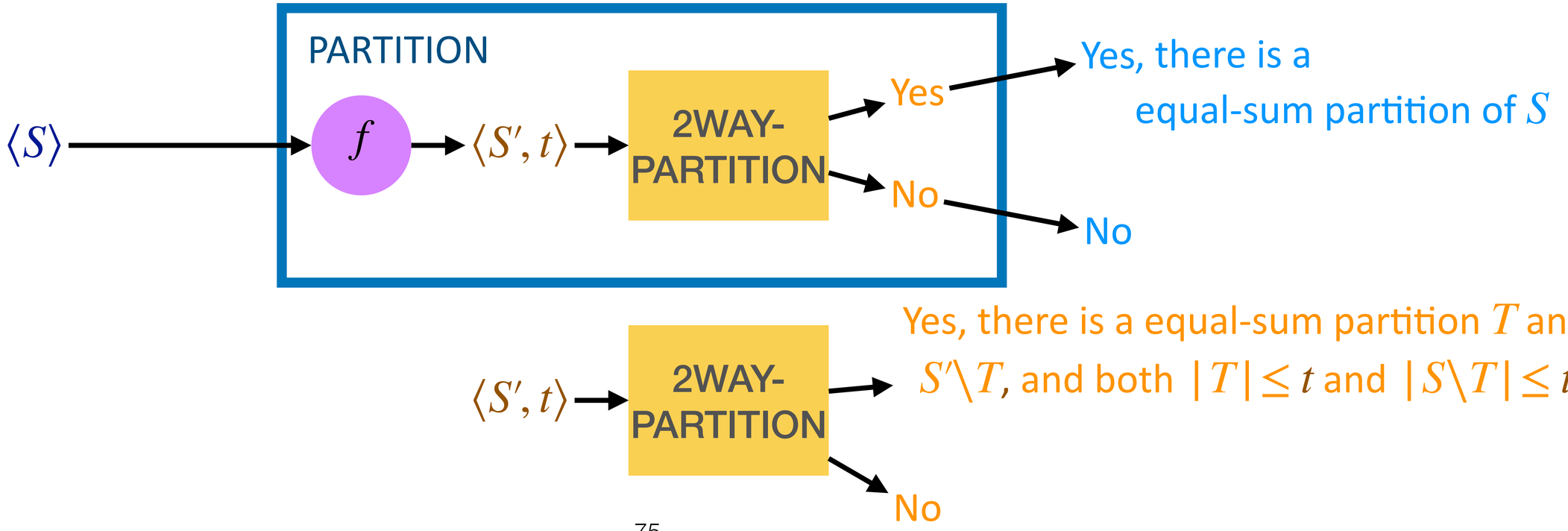
# Ex: Reduce PARTITION to 2WAY-PARTITION

- PARTITION $= \{\langle S \rangle \mid$ $S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
- 2WAY-PARTITION $= \{\langle S, t \rangle \mid$ $S = \{x_1, \cdots, x_k\}$ where exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$ and $|T| \leq t$ and $|S \setminus T| \leq t\}$



# Ex: Reduce PARTITION to 2WAY-PARTITION

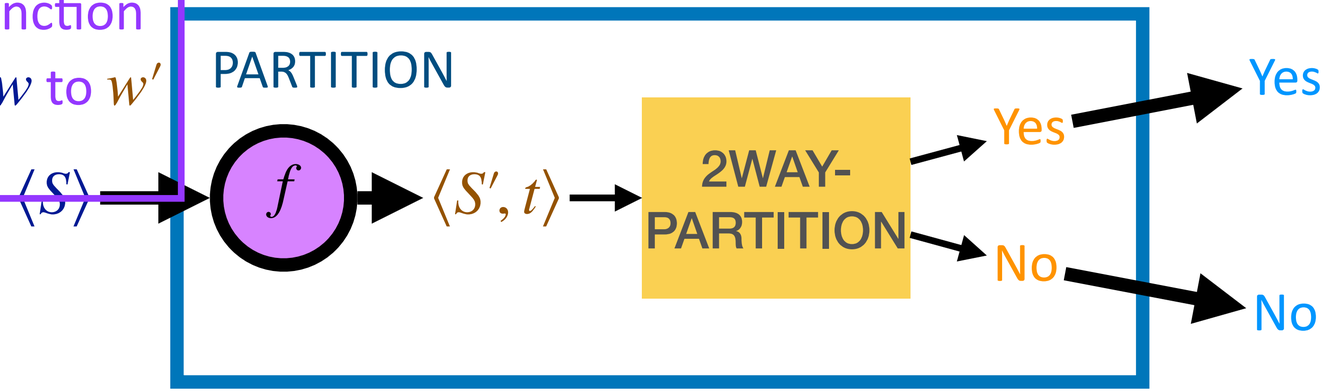
- PARTITION $= \{\langle S \rangle \mid$ $S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
- 2WAY-PARTITION $= \{\langle S, t \rangle \mid$ $S = \{x_1, \cdots, x_k\}$ where exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$ and $|T| \leq t$ and $|S \setminus T| \leq t\}$



# Ex: Reduce PARTITION to 2WAY-PARTITION

- PARTITION $= \{\langle S \rangle \mid S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
- 2WAY-PARTITION $= \{\langle S, t \rangle \mid S = \{x_1, \cdots, x_k\}$ where exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$ and $|T| \leq t$ and $|S \setminus T| \leq t\}$

1. Show that there is a function that transforms every $w$ to $w'$ in polynomial time



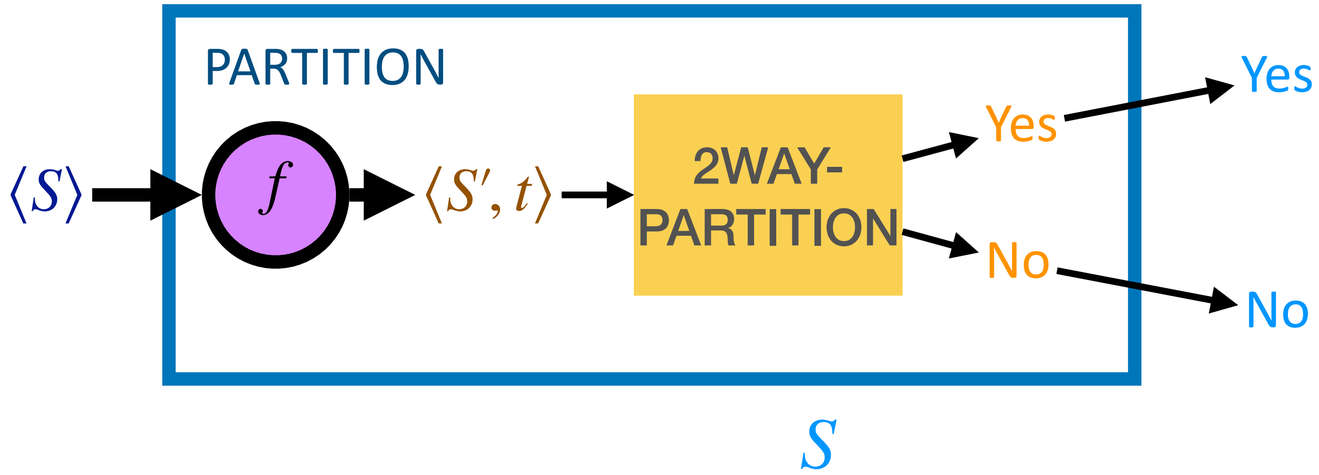

2. Show that for any yes-instance $w' \in B$, the corresponding instance $w$ is also a yes-instance of $A$

3. Show that for any yes-instance $w \in A$, the corresponding instance $w'$ is also a yes-instance of $B$

3. Show that for any no-instance $w' \notin B$, the corresponding instance $w$ is also a no-instance of $A$

# Ex: Reduce PARTITION to 2WAY-PARTITION

- PARTITION $= \{\langle S \rangle \mid$ $S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
- 2WAY-PARTITION $= \{\langle S, t \rangle \mid$ $S = \{x_1, \cdots, x_k\}$ where exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$ and $|T| \leq t$ and $|S \setminus T| \leq t\}$

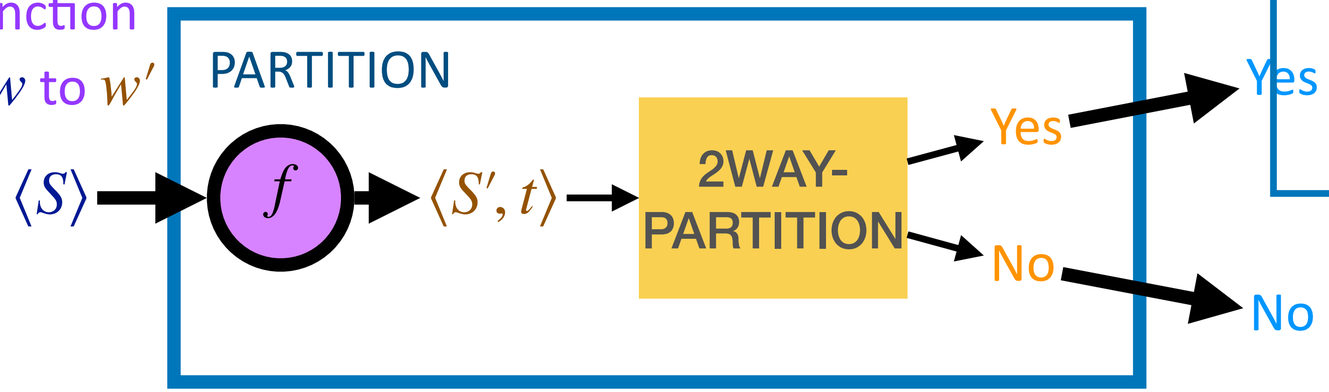



- $f$: Let $S'$ be $S$ and let $t$ be $|S'|$.
  The function $f$ can be computed in polynomial time.

# Ex: Reduce PARTITION to 2WAY-PARTITION

- PARTITION $= \{\langle S \rangle \mid S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
- 2WAY-PARTITION $= \{\langle S, t \rangle \mid S = \{x_1, \cdots, x_k\}$ where exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$ and $|T| \leq t$ and $|S \setminus T| \leq t\}$

1. Show that there is a function that transforms every $w$ to $w'$ in polynomial time

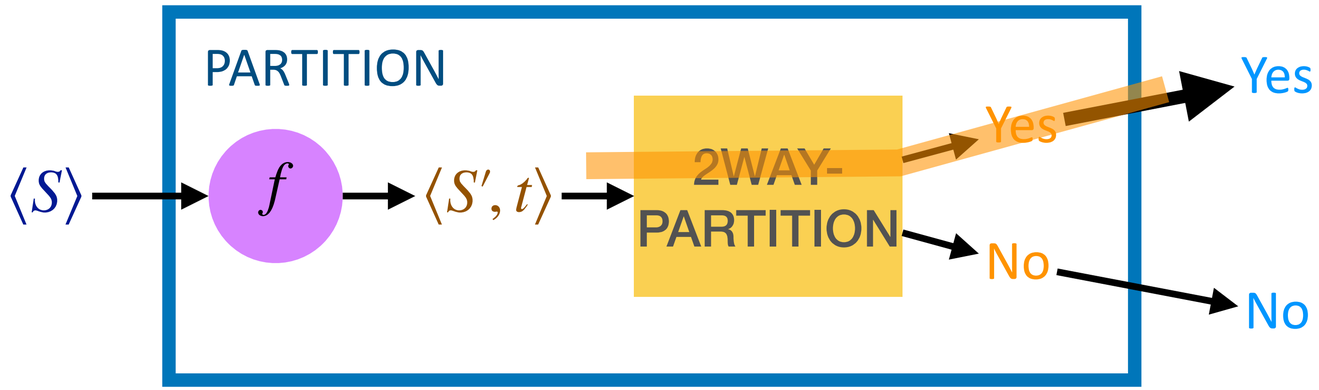


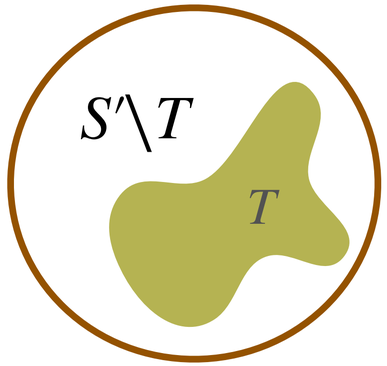

2. Show that for any yes-instance $w' \in B$, the corresponding instance $w$ is also a yes-instance of $A$

3. Show that for any yes-instance $w \in A$, the corresponding instance $w'$ is also a yes-instance of $B$

3. Show that for any no-instance $w' \notin B$, the corresponding instance $w$ is also a no-instance of $A$

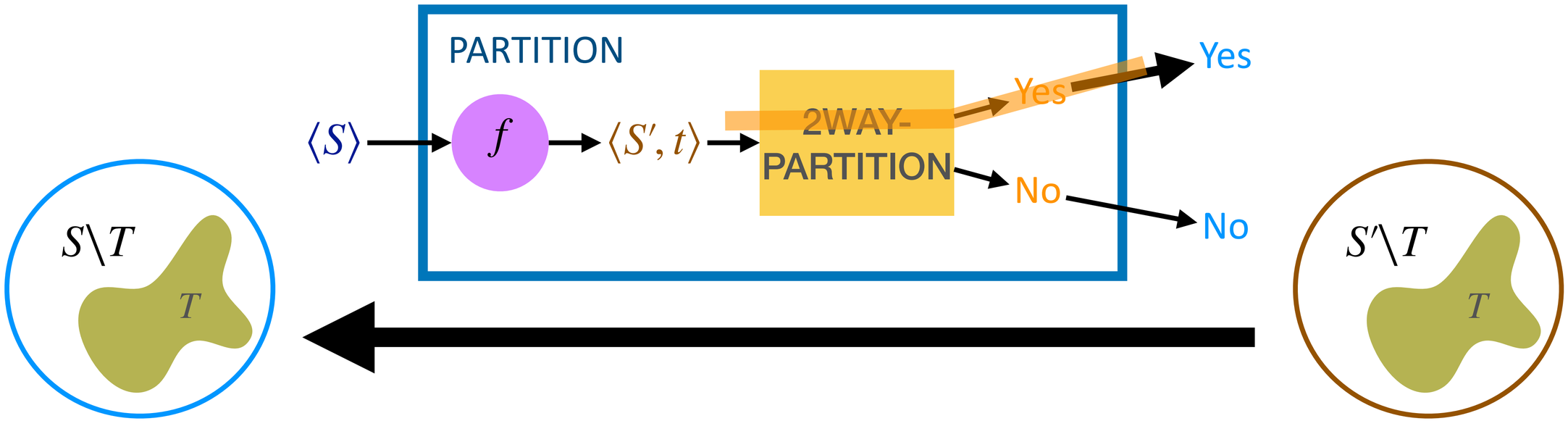
# Ex: Reduce PARTITION to 2WAY-PARTITION




- For any yes-instance $\langle S', t \rangle$ of 2WAY-PARTITION, there exists a subset $T$ of $S'$ such that the sum of elements in $T$ is equal to the sum of elements in $S' \backslash T$, $|T| \leq t$, and $|S' \backslash T| \leq t$.

# Ex: Reduce PARTITION to 2WAY-PARTITION

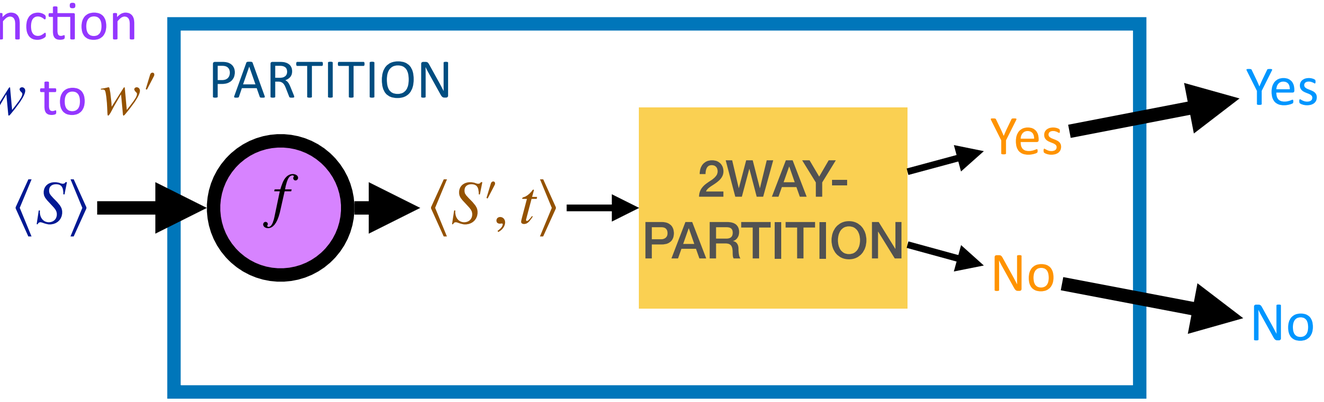



- For any yes-instance $\langle S', t \rangle$ of 2WAY-PARTITION, there exists a subset $T$ of $S'$ such that the sum of elements in $T$ is equal to the sum of elements in $S' \backslash T$, $|T| \leq t$, and $|S' \backslash T| \leq t$. The partition $T$ is also a partition in $S$. Therefore, the corresponding instance $\langle S \rangle$ is a yes-instance of PARTITION.

# Ex: Reduce PARTITION to 2WAY-PARTITION

- PARTITION $= \{\langle S \rangle \mid S = \{x_1, \cdots, x_k\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$
- 2WAY-PARTITION $= \{\langle S, t \rangle \mid S = \{x_1, \cdots, x_k\}$ where exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$ and $|T| \leq t$ and $|S \setminus T| \leq t\}$

1. Show that there is a function that transforms every $w$ to $w'$ in polynomial time

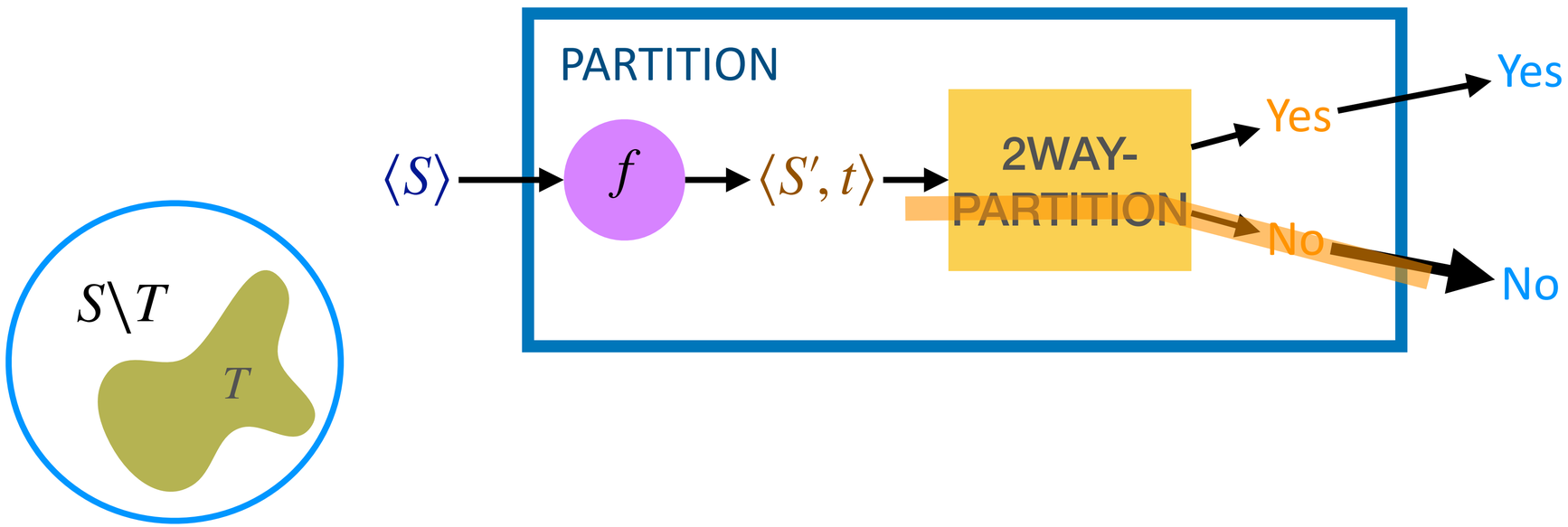



2. Show that for any yes-instance $w' \in B$, the corresponding instance $w$ is also a yes-instance of $A$

3. Show that for any yes-instance $w \in A$, the corresponding instance $w'$ is also a yes-instance of $B$

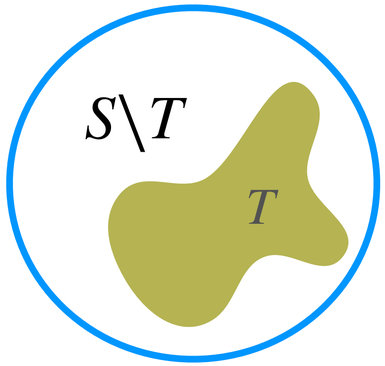
3. Show that for any no-instance $w' \notin B$, the corresponding instance $w$ is also a no-instance of $A$

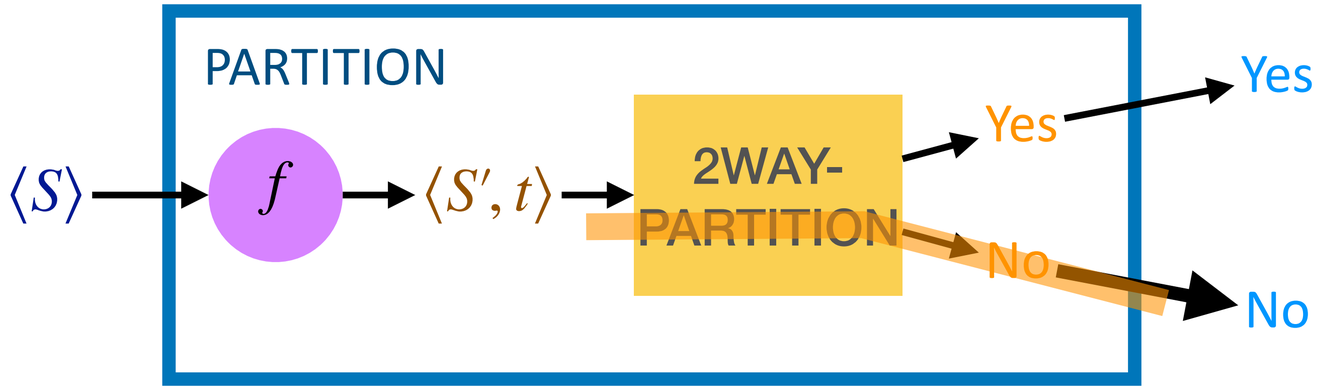
# Ex: Reduce PARTITION to 2WAY-PARTITION




- For any yes-instance $\langle S \rangle$ of PARTITION, there exists an equal-sum partition of $S$.

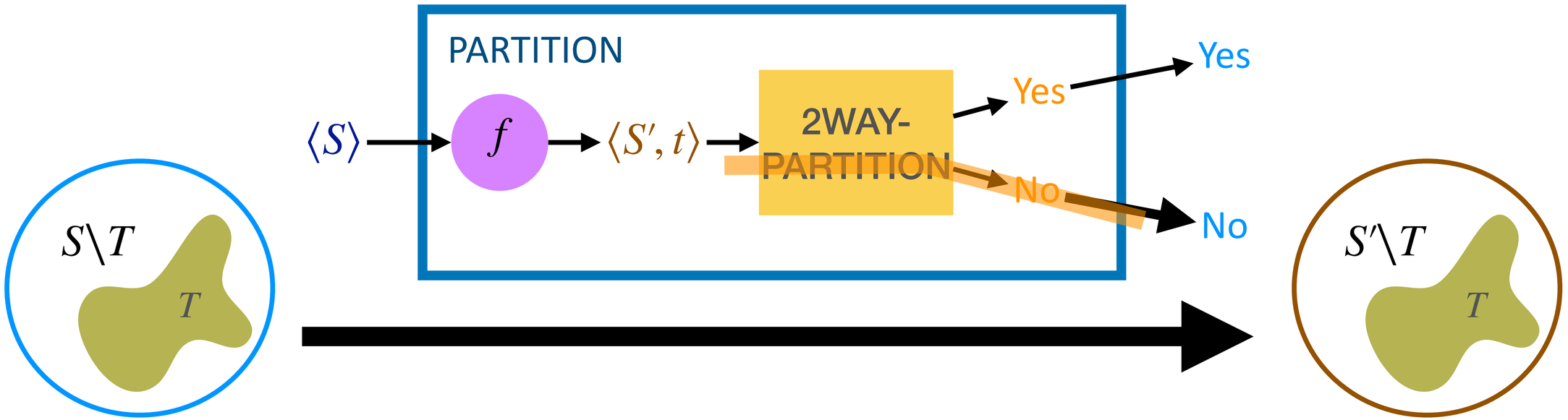
# Ex: Reduce PARTITION to 2WAY-PARTITION




- For any yes-instance $\langle S \rangle$ of PARTITION, there exists an equal-sum partition of $S$. Moreover, each of the two parts $T$ and $S \backslash T$ has cardinality at most $|S| = |S'| = t$.

# Ex: Reduce PARTITION to 2WAY-PARTITION

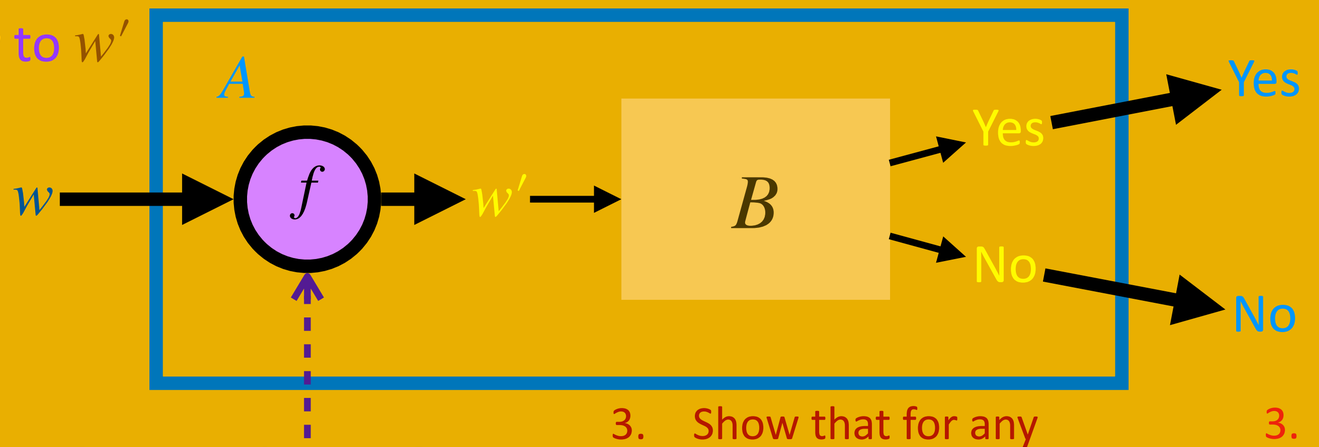



- For any yes-instance $\langle S \rangle$ of PARTITION, there exists an equal-sum partition of $S$. Moreover, each of the two parts $T$ and $S \backslash T$ has cardinality at most $|S| = |S'| = t$. Therefore, the corresponding instance $\langle S', t \rangle$ is a yes-instance of 2WAY-PARTITION.

# Polynomial-Time Reduce $A$ to $B$

- Problem $A$ with input $w$
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem $B$ with input $w'$
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$

1. Show that there is a function that transforms every $w$ to $w'$ in polynomial time




2. Show that for any yes-instance $w' \in B$, the corresponding instance $w$ is also a yes-instance of $A$

3. Show that for any yes-instance $w \in A$, the corresponding instance $w'$ is also a yes-instance of $B$

3. Show that for any no-instance $w' \notin B$, the corresponding instance $w$ is also a no-instance of $A$

# Reduction and Hardness

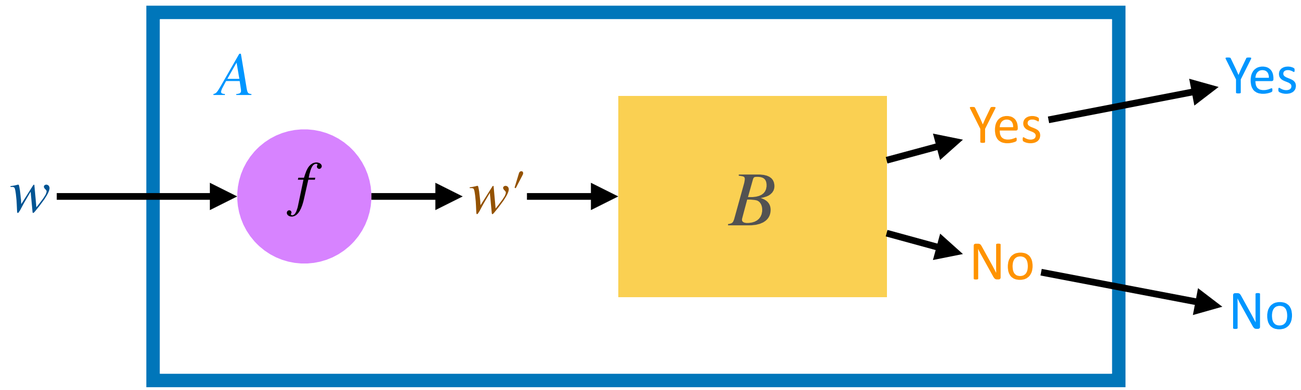
- Reduction from problem $A$ to problem $B$

# Reduction and Hardness

- Reduction from problem $A$ to problem $B$
  - If we can solve $B$,

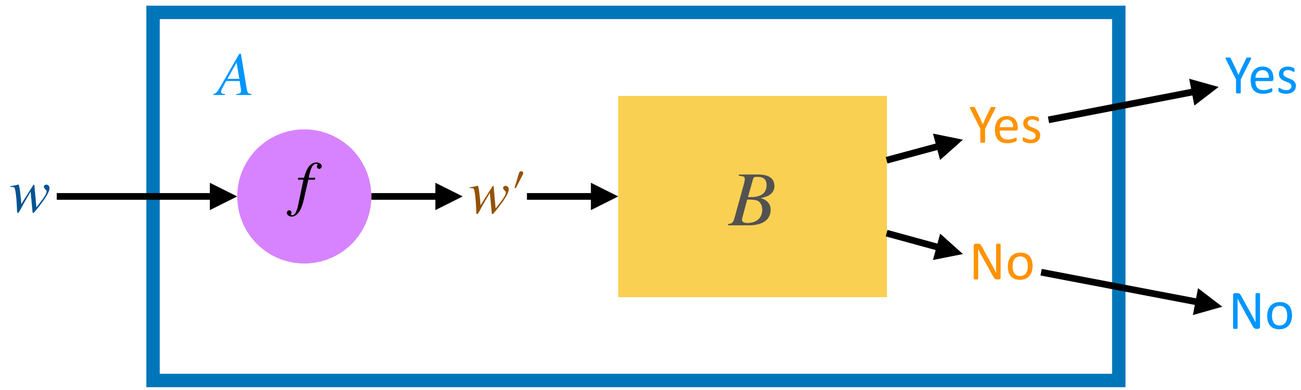
# Reduction and Hardness

- Reduction from problem $A$ to problem $B$
  - If we can solve $B$, we can solve $A$



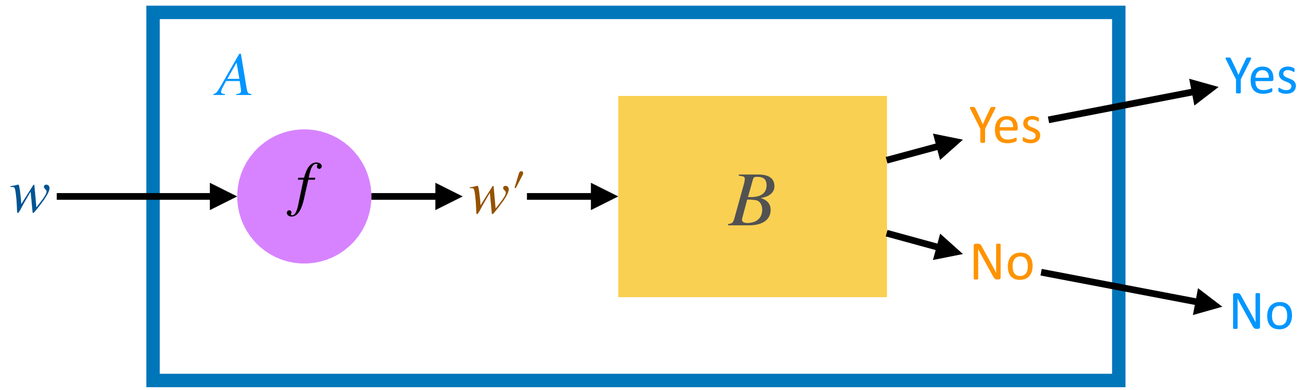
# Reduction and Hardness

- Reduction from problem $A$ to problem $B$
  - If we can solve $B$, we can solve $A$
  - It implies that solving $B$ is at least as hard as solving $A$



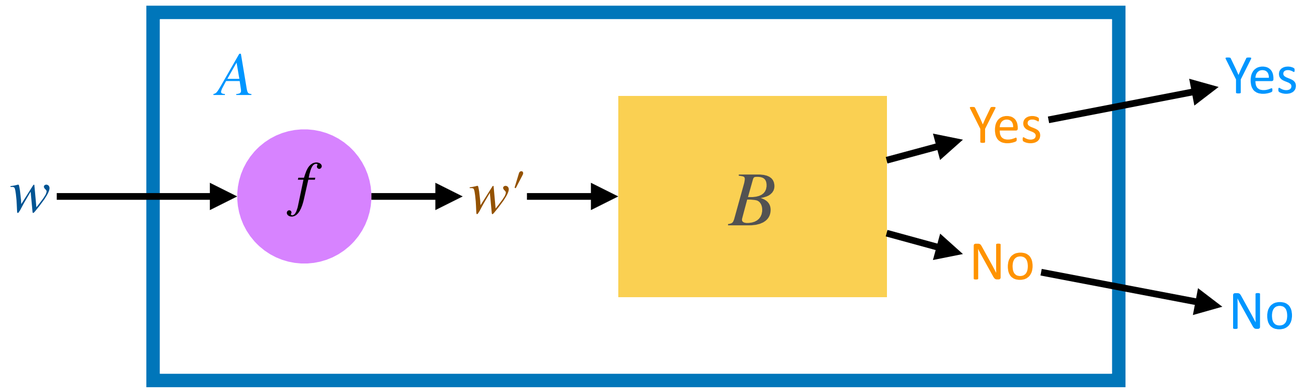
# Reduction and Hardness

- Reduction from problem $A$ to problem $B$
  - If we can solve $B$, we can solve $A$
  - It implies that solving $B$ is at least as hard as solving $A$
    - Problem $A$ is easier than problem $B$



# Reduction and Hardness

- Reduction from problem $A$ to problem $B$
  - If we can solve $B$, we can solve $A$
  - It implies that solving $B$ is at least as hard as solving $A$
    - Problem $A$ is easier than problem $B$
    - Problem $A$ and problem $B$ are equally difficult



# NP-Hard

# NP-Hard

- Definition: A problem $B$ is ***NP-hard*** if all problems in **NP** can be polynomial-time reduced to $B$

# NP-Hard

- Definition: A problem $B$ is ***NP-hard*** if all problems in **NP** can be polynomial-time reduced to $B$
  - That is, an NP-hard problem is at least as hard as any problem in **NP**

# NP-Hard

- Definition: A problem $B$ is ***NP-hard*** if all problems in **NP** can be polynomial-time reduced to $B$
  - That is, an NP-hard problem is at least as hard as any problem in **NP**

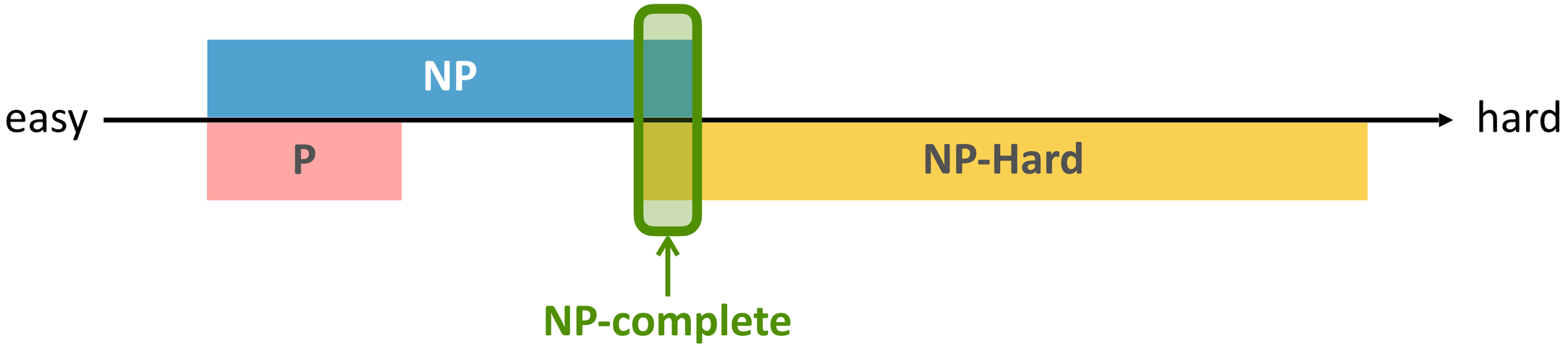
# NP-Hard

- Definition: A problem $B$ is ***NP-hard*** if all problems in **NP** can be polynomial-time reduced to $B$
  - That is, an NP-hard problem is at least as hard as any problem in **NP**

# NP-Hard

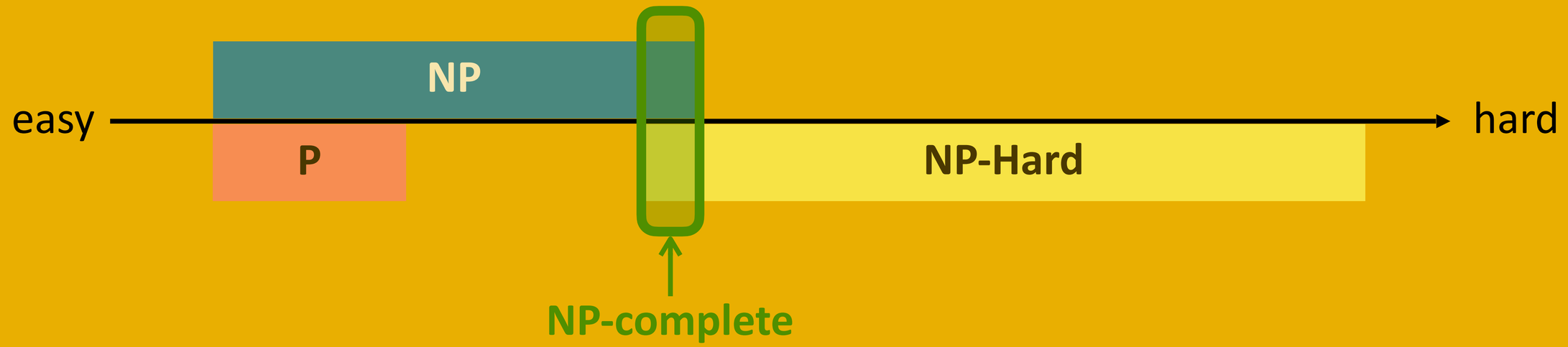
- Definition: A problem $B$ is ***NP-hard*** if all problems in **NP** can be polynomial-time reduced to $B$
  - That is, an NP-hard problem is at least as hard as any problem in **NP**

# NP-Complete



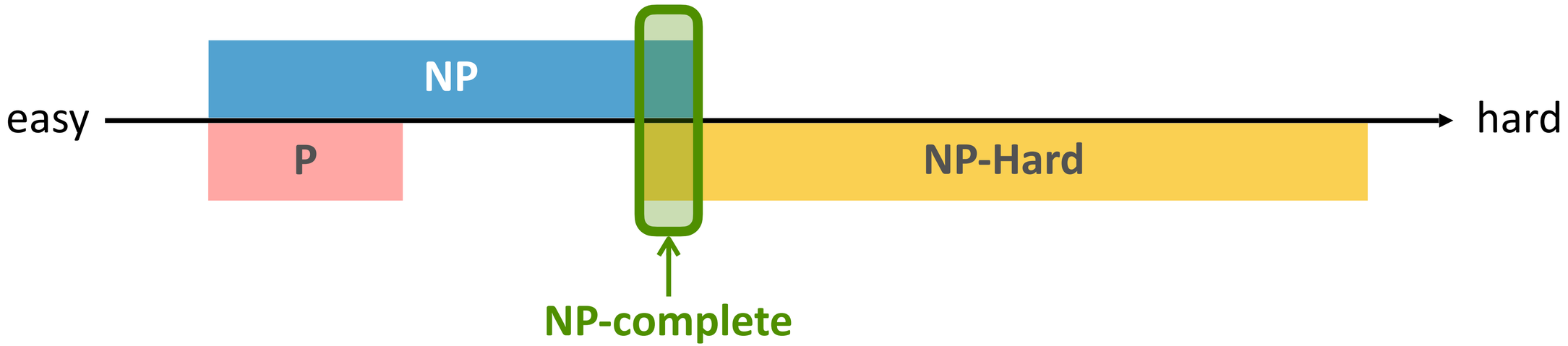
# What Happened

- If we can reduce problem $A$ to problem $B$, problem $A$ is not harder than $B$
- **NP-hard** problems are those at least as hard as any problem in **NP**
- **NP-complete** problems are those "hardest" in **NP**
  - The intersection of **NP** and **NP-hard**



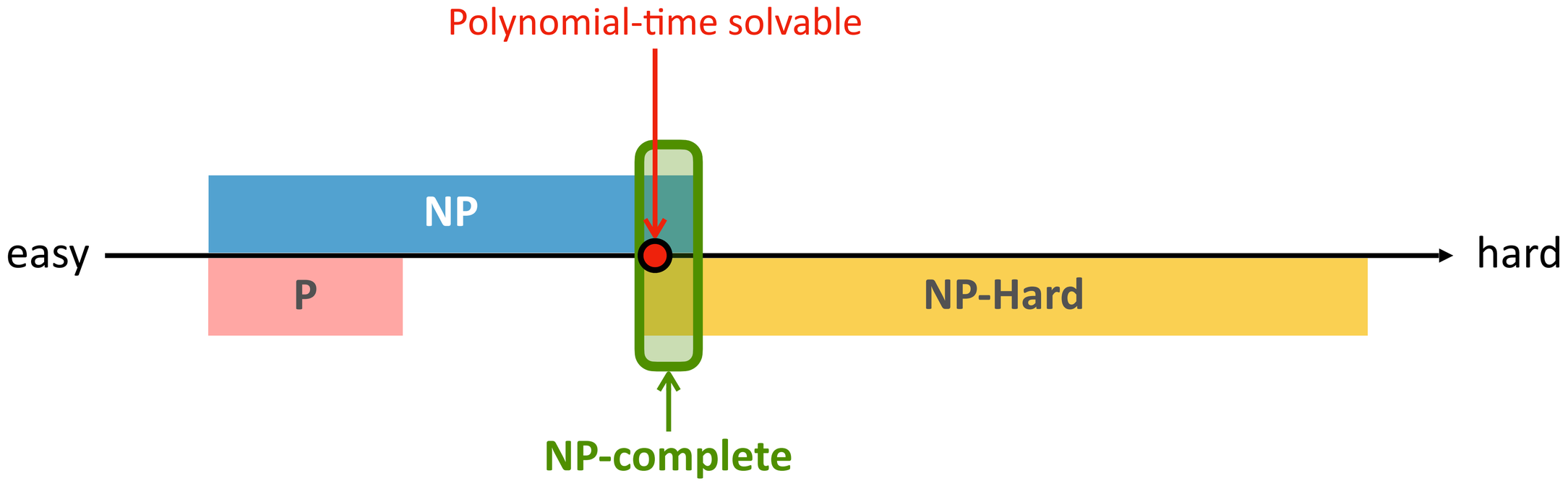
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



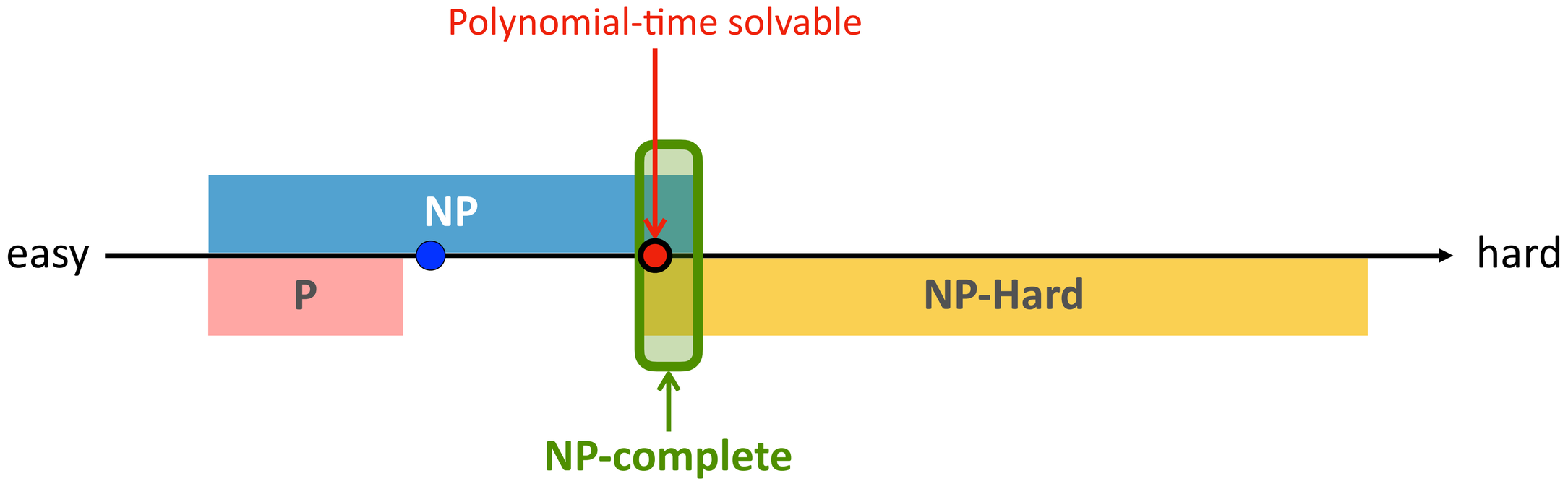
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



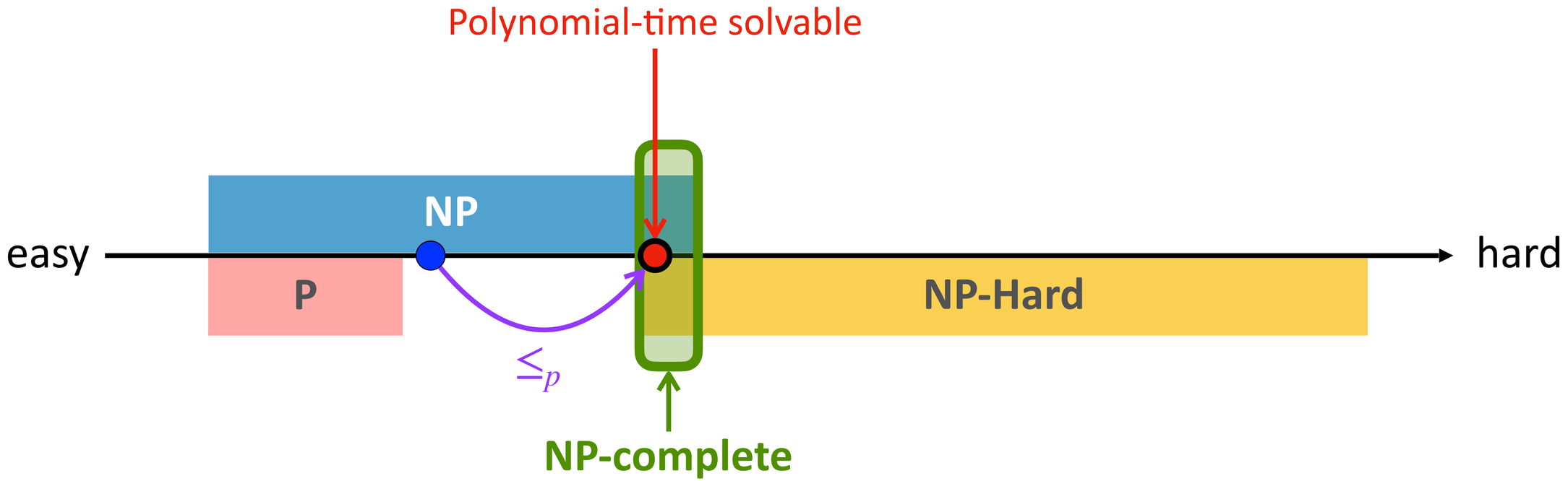
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



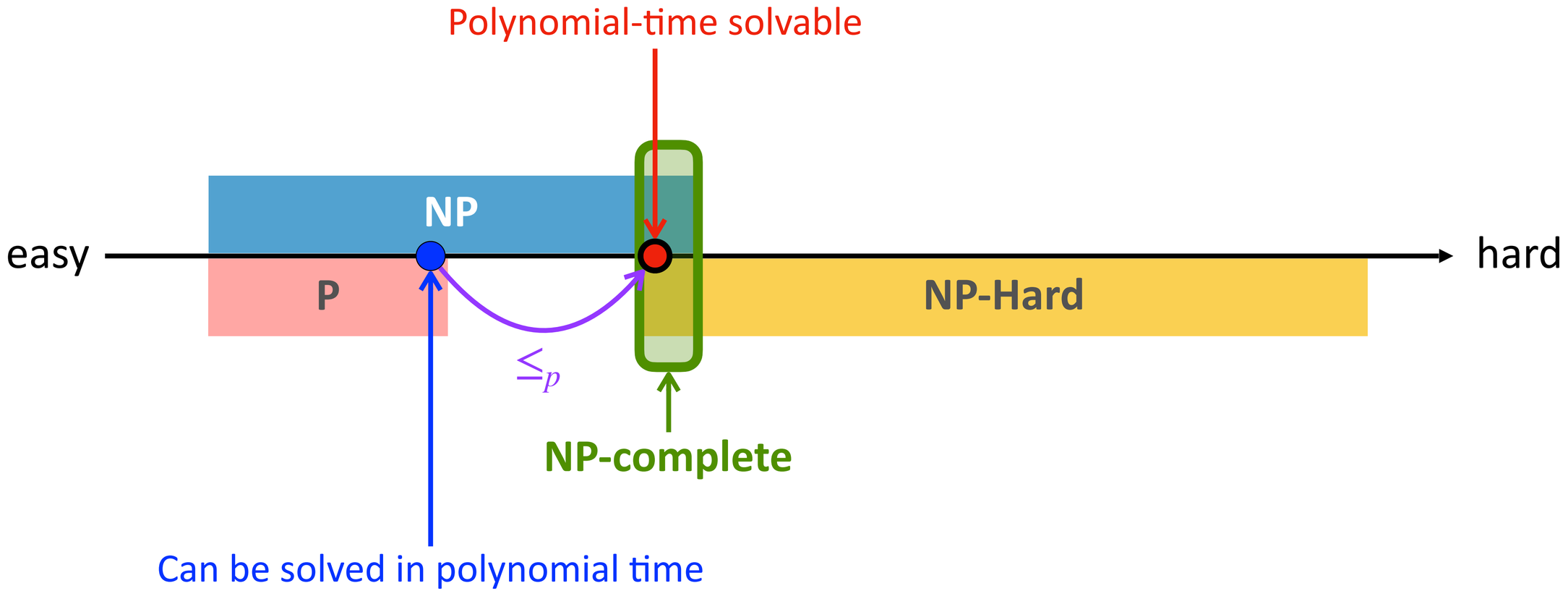
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



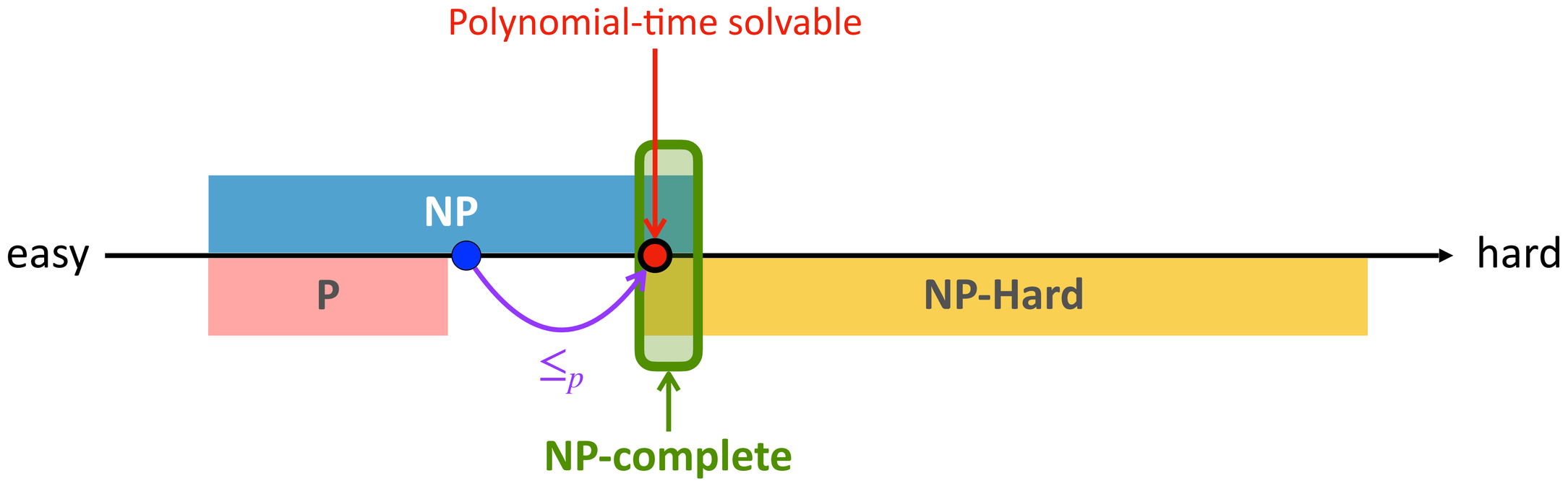
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



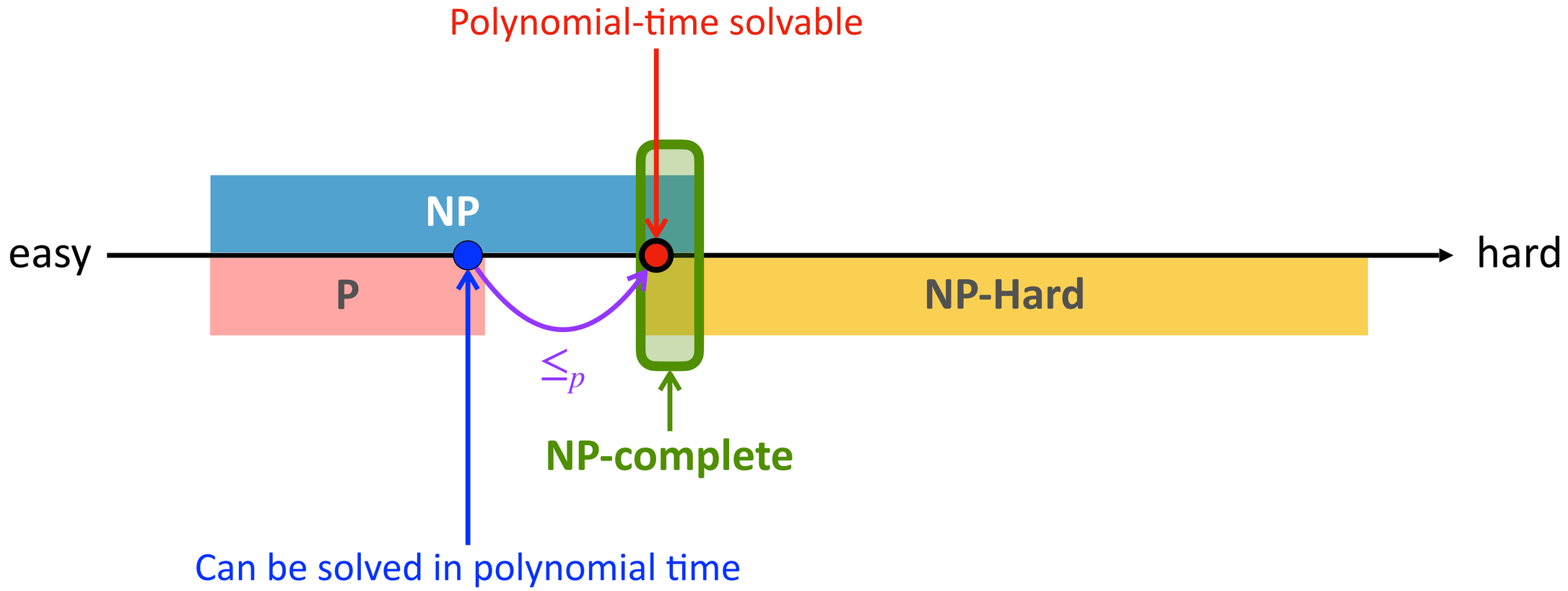
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



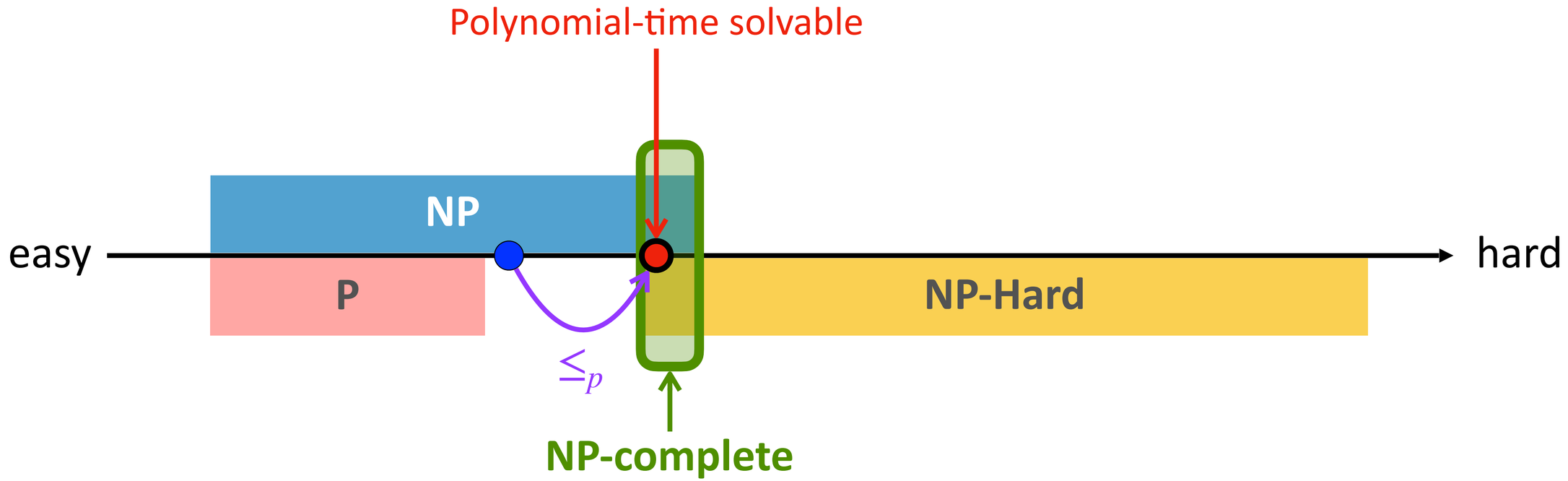
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



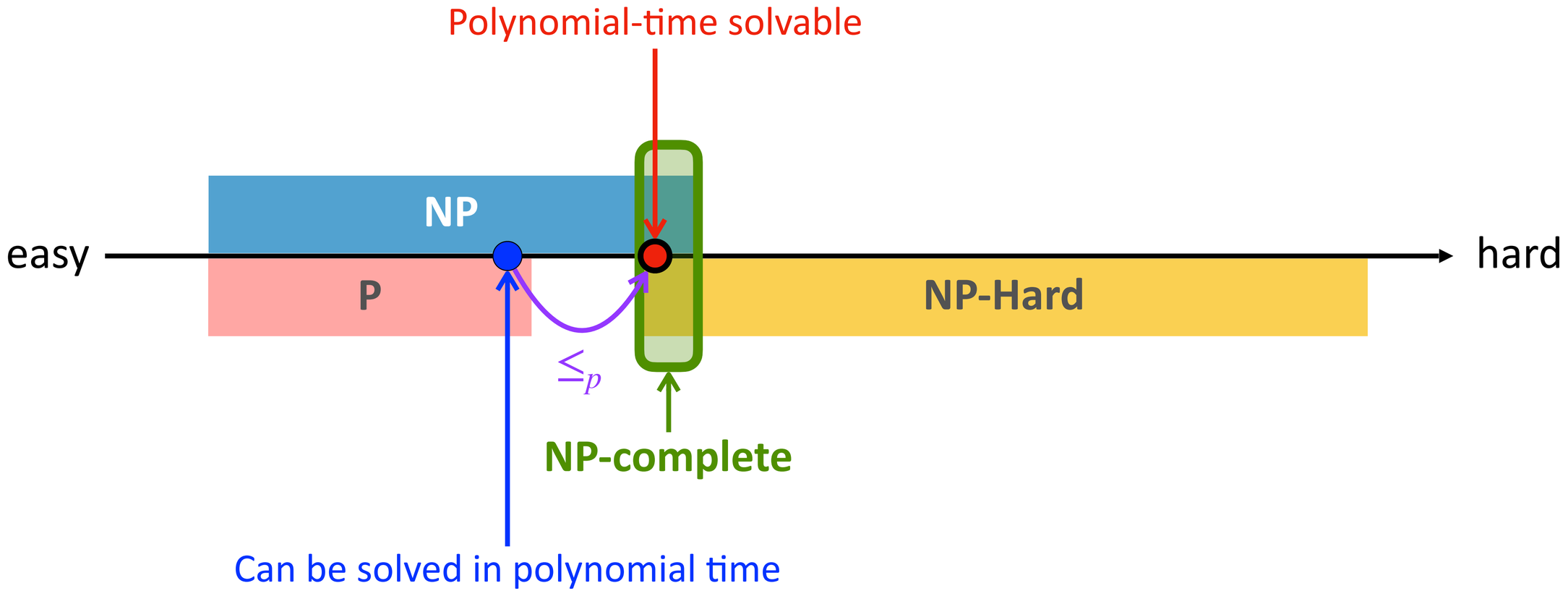
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



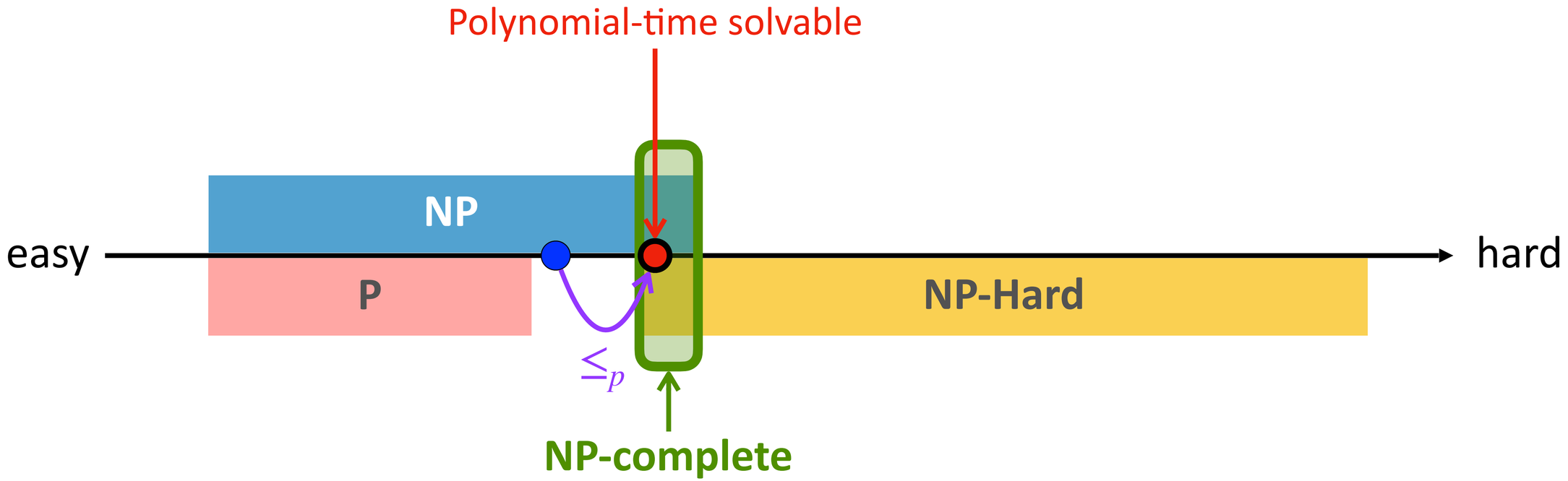
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



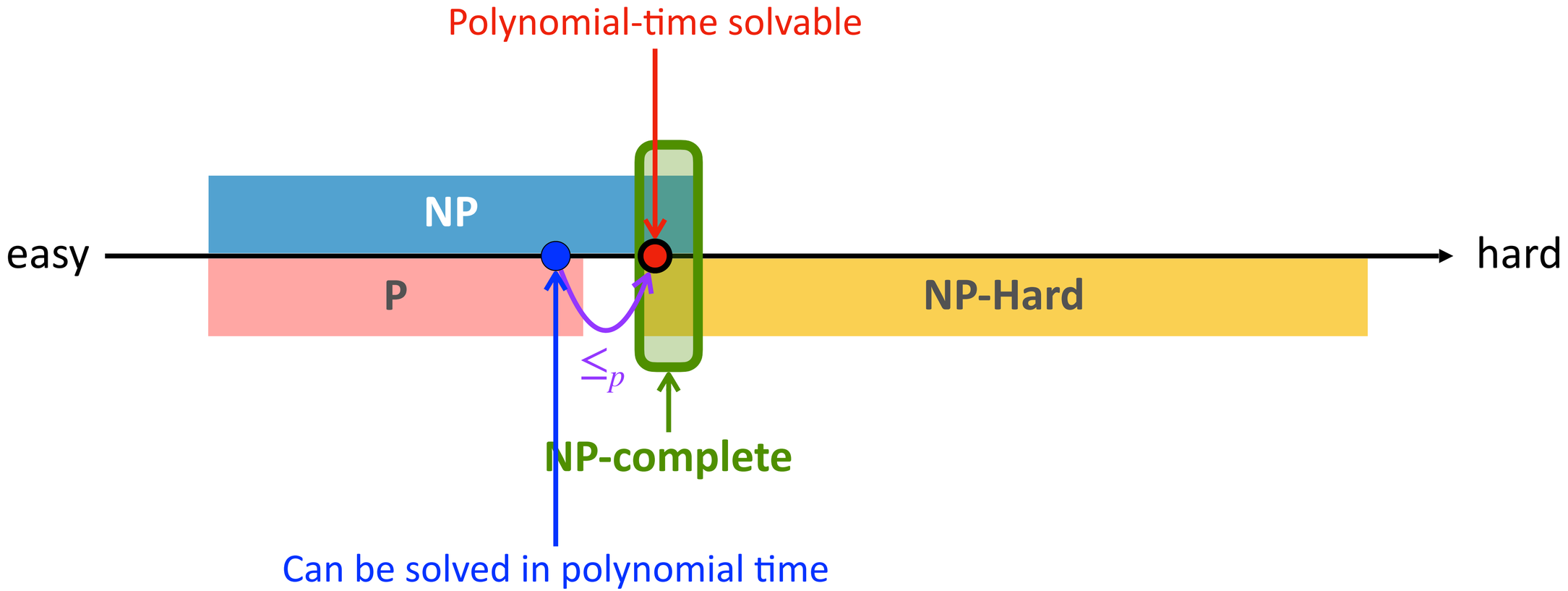
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



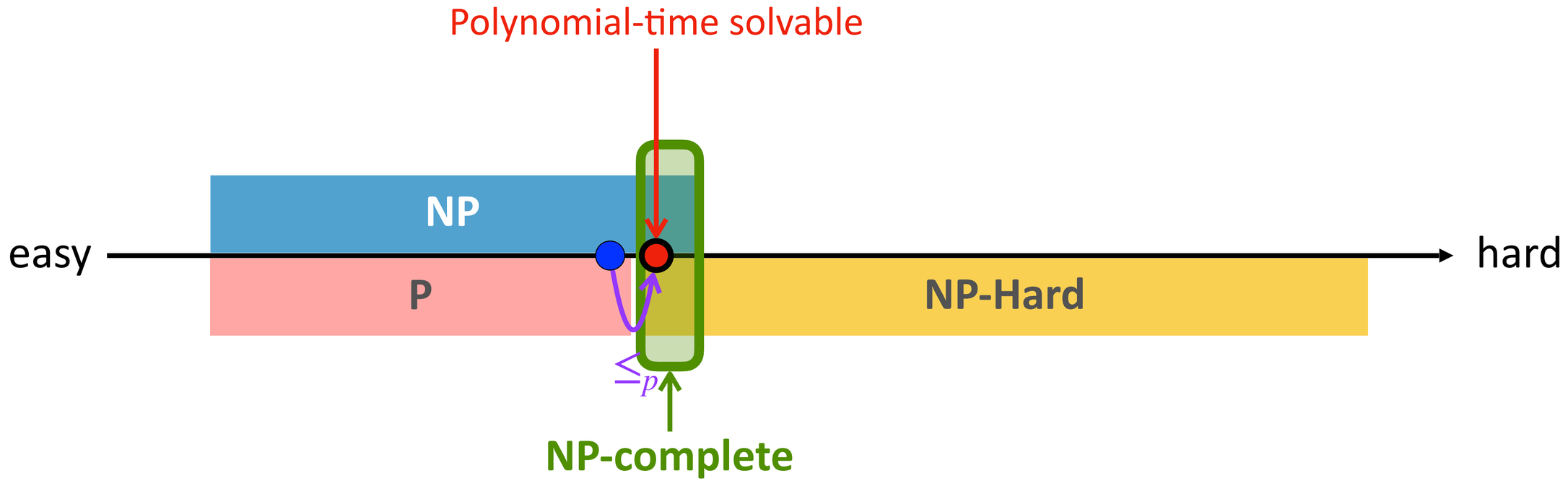
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



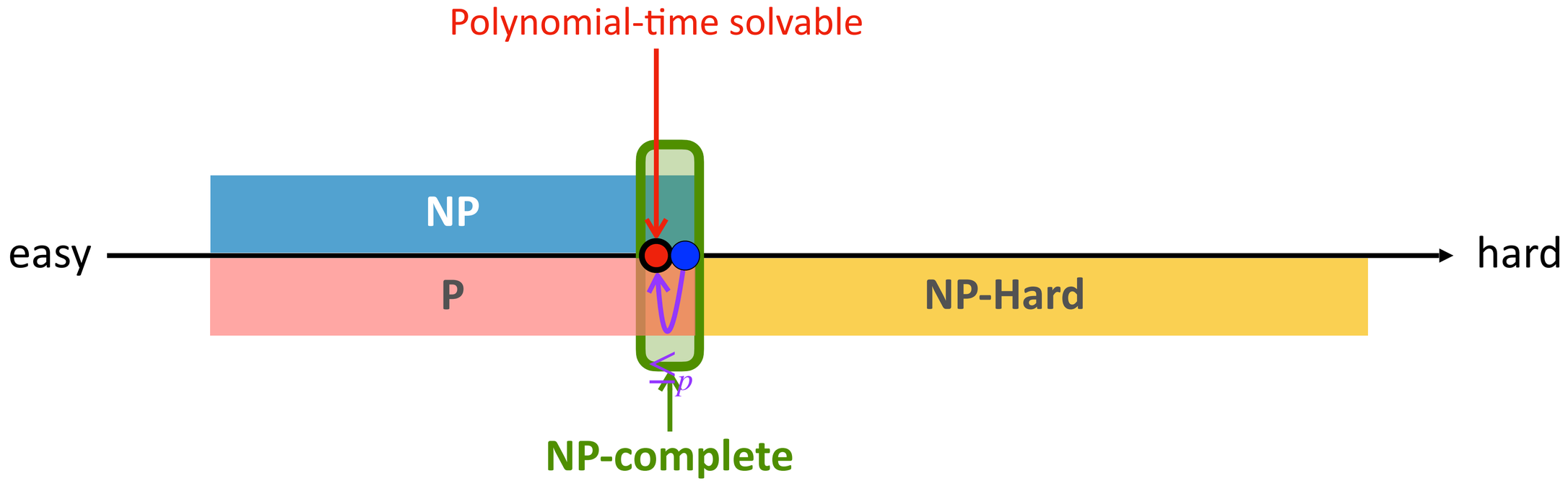
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



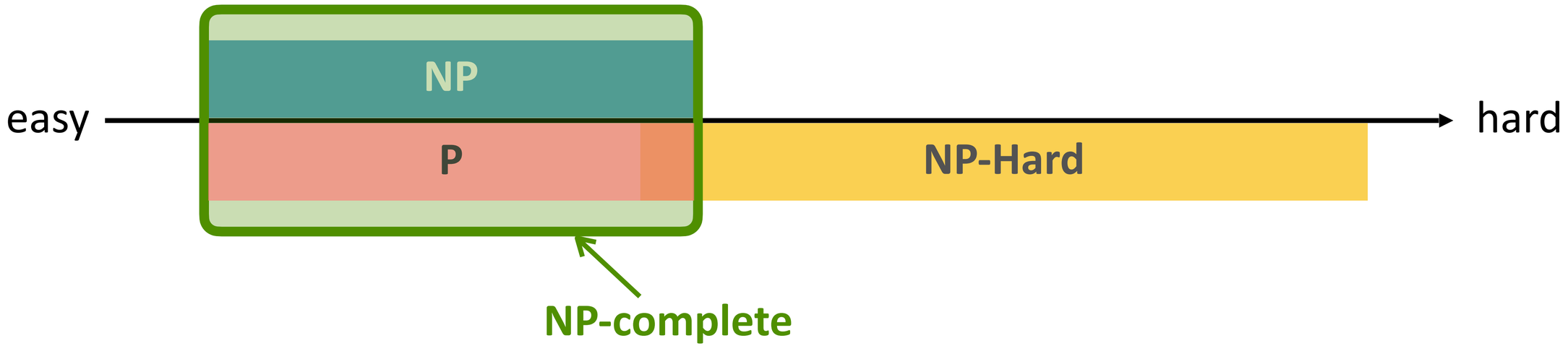
# NP-Completeness Revisit

- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time

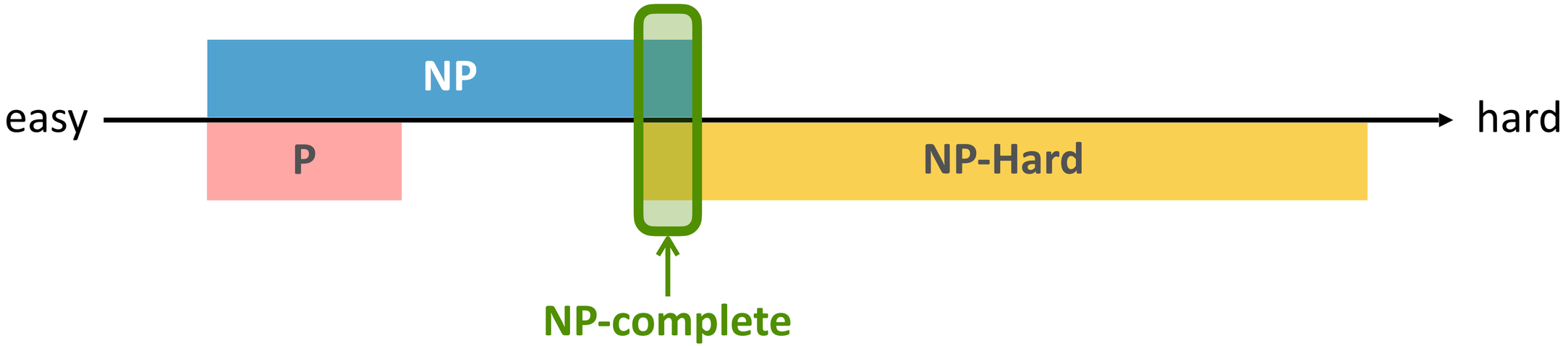


# NP-Completeness Revisit

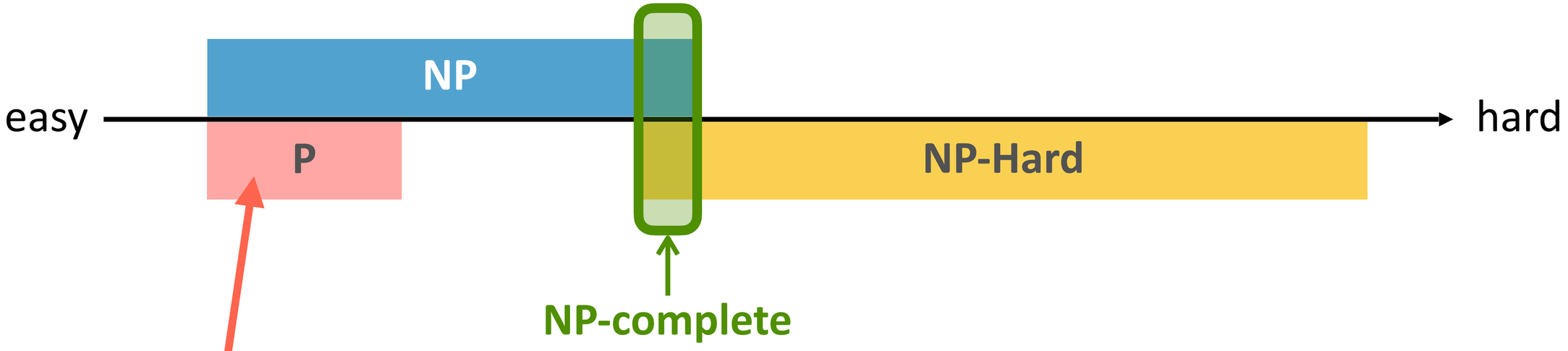
- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



# P, NP, NP-Hard, and NP-Complete

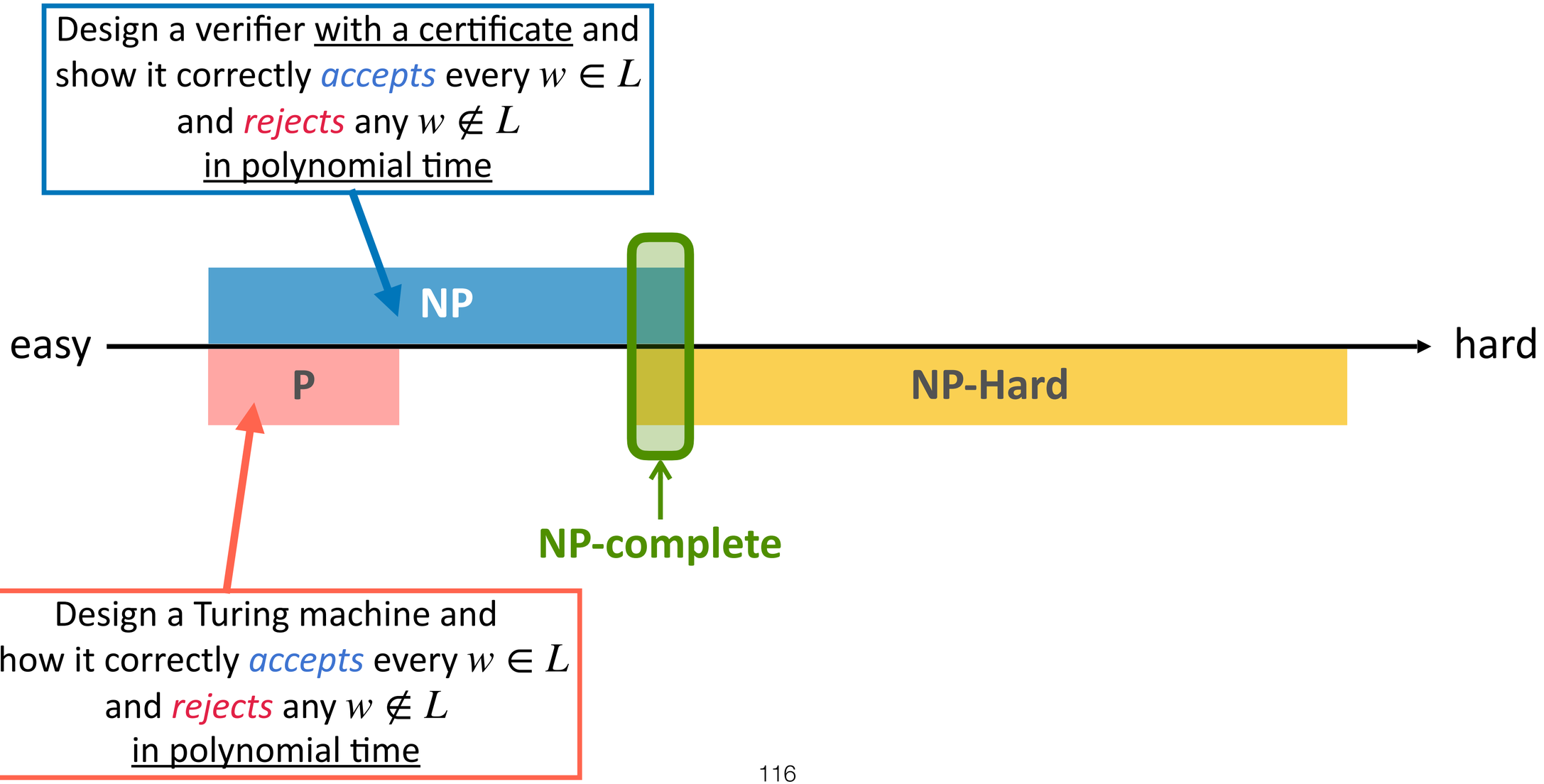


# P, NP, NP-Hard, and NP-Complete

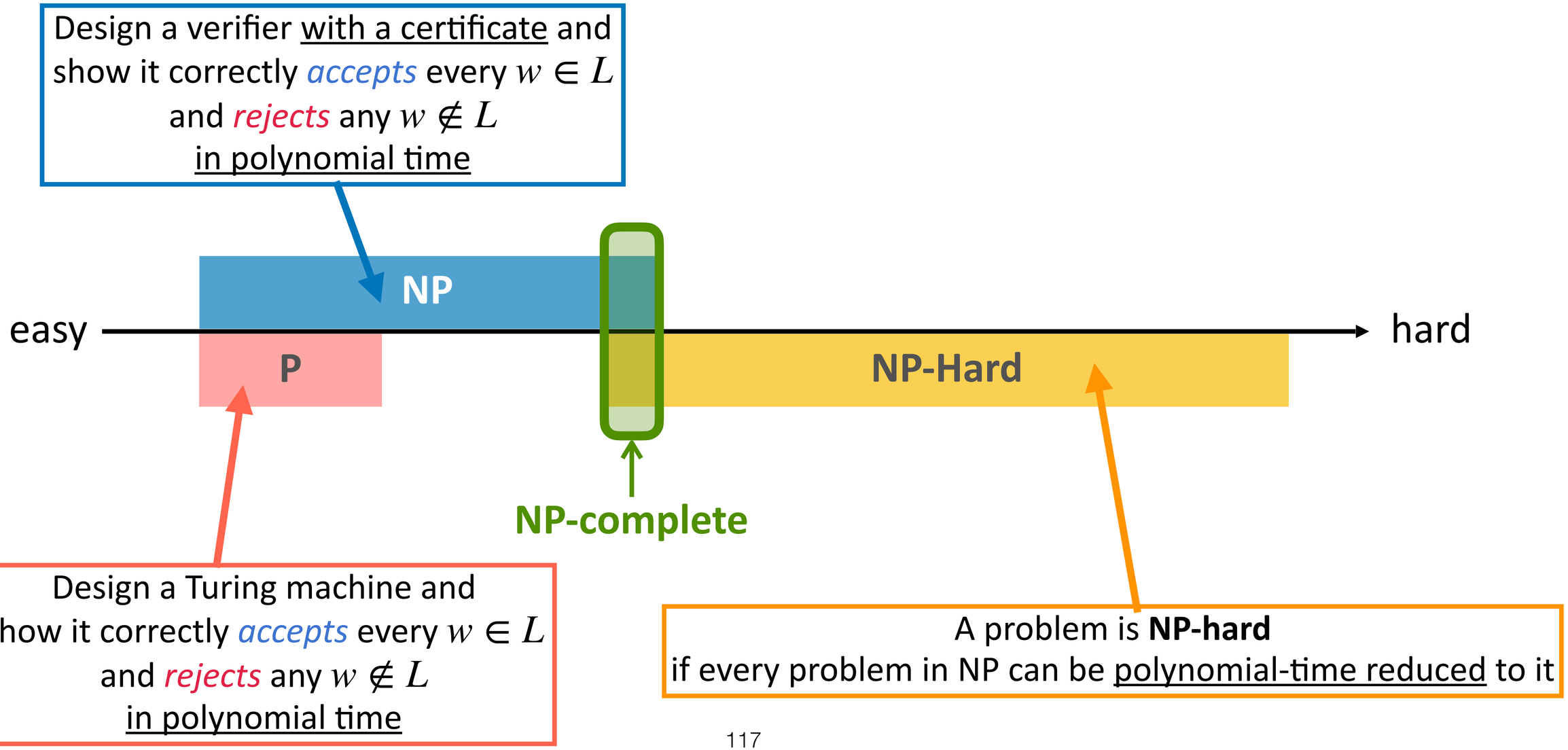



Design a Turing machine and
show it correctly *accepts* every $w \in L$
and *rejects* any $w \notin L$
in polynomial time

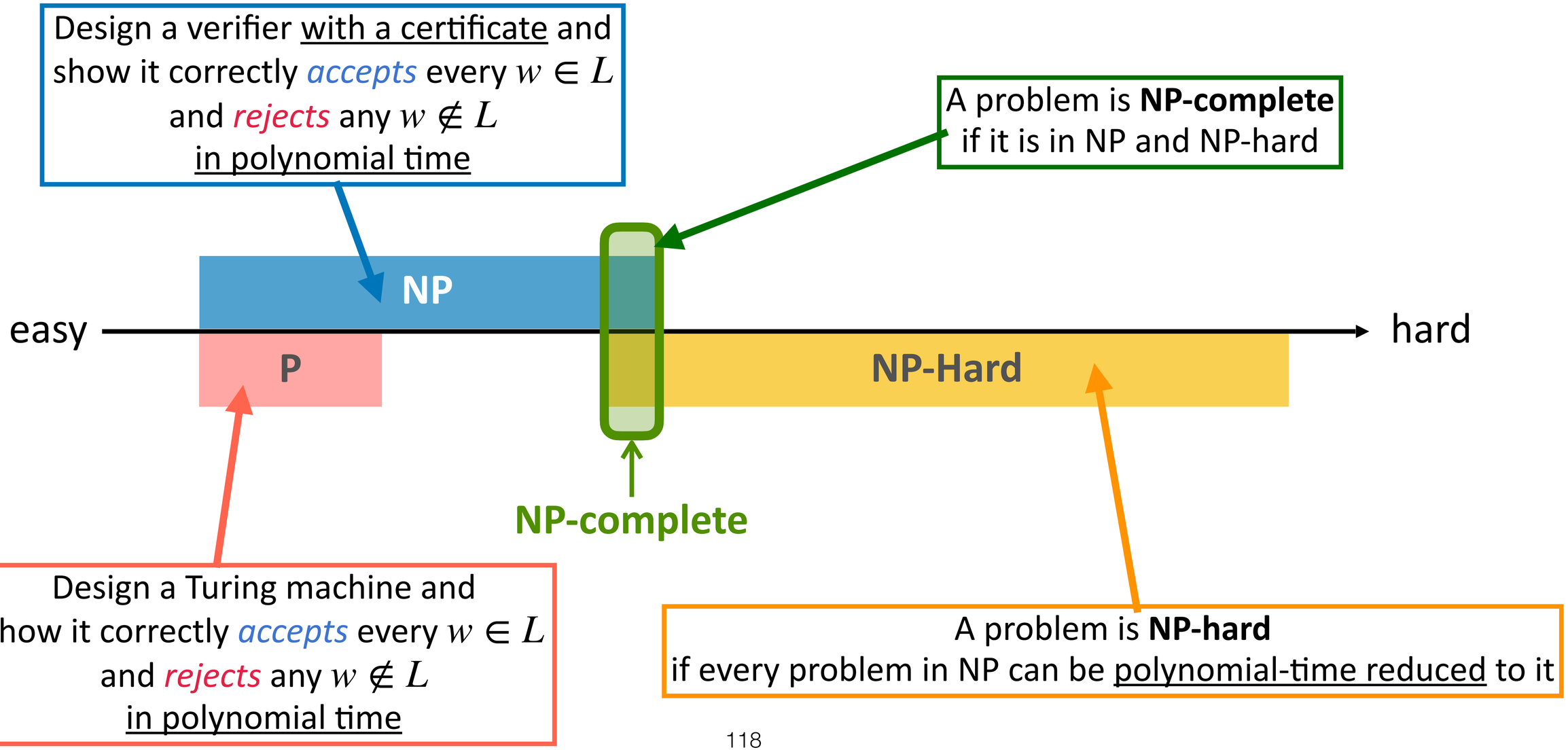
# P, NP, NP-Hard, and NP-Complete



# P, NP, NP-Hard, and NP-Complete



# P, NP, NP-Hard, and NP-Complete



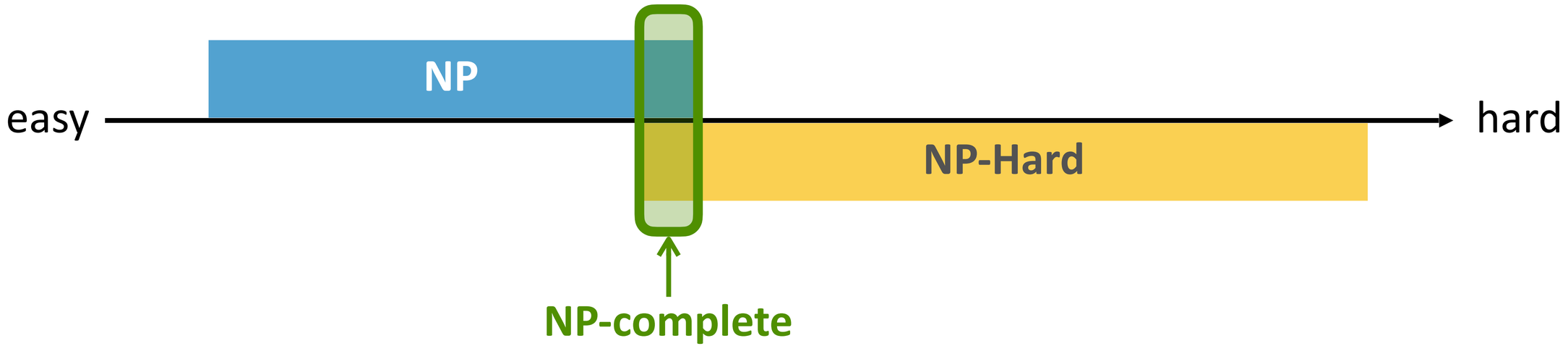
# How to prove a problem is NP-Hard

# How to prove a problem is NP-Hard

- Definition: A problem $B$ is ***NP-hard*** if all problems in **NP** can be polynomial-time reduced to $B$

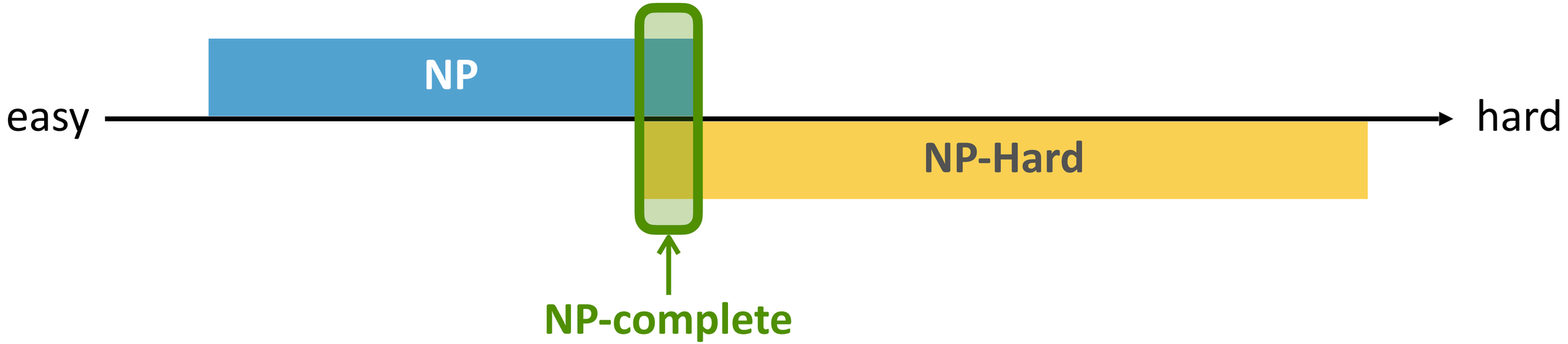
# How to prove a problem is NP-Hard

- Definition: A problem $B$ is ***NP-hard*** if all problems in **NP** can be polynomial-time reduced to $B$
- To prove that a problem $B$ is NP-hard, we find a NP-complete problem $A$ and reduce $A$ to $B$



# How to prove a problem is NP-Hard

- Definition: A problem $B$ is ***NP-hard*** if all problems in **NP** can be polynomial-time reduced to $B$
- To prove that a problem $B$ is NP-hard, we find a NP-complete problem $A$ and reduce $A$ to $B$
  - An NP-complete problem is NP-hard and all problems in NP can be reduced to it



# Satisfiability Problem SAT

- The first NP-Complete problem: Satisfiability problem SAT
- Boolean formula: an expression involving Boolean variables and operations
  - Example:
    - $\phi = \bar{x} \wedge y \wedge z$ $\quad x = \text{FALSE}, y = \text{TRUE}, z = \text{TRUE}$
    - $\phi = (\bar{x} \wedge y) \vee (x \wedge \bar{z})$ $\quad x = \text{FALSE}, y = \text{TRUE}, z = \text{FALSE}$
  - The Boolean variables can take on the values TRUE (1) and FALSE (0)
  - A Boolean formula is satisfiable if some assignment of TRUEs and FALSEs to the variables make the formula true
- SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable Boolean formula$\}$

# Cook-Levin Theorem

- The first NP-complete problem: **satisfiability problem**
  SAT $= \{\langle\phi\rangle \mid \phi$ is a satisfiable Boolean formaula$\}$

- Cook-Levin theorem: SAT $\in$ **P** iff **P** = **NP**

  $\leftrightarrow$ SAT is NP-complete

# Cook-Levin Theorem

- The first NP-complete problem: **satisfiability problem**
  SAT $= \{\langle\phi\rangle \mid \phi$ is a satisfiable Boolean formaula$\}$

- Cook-Levin theorem: SAT $\in$ **P** iff **P** = **NP**

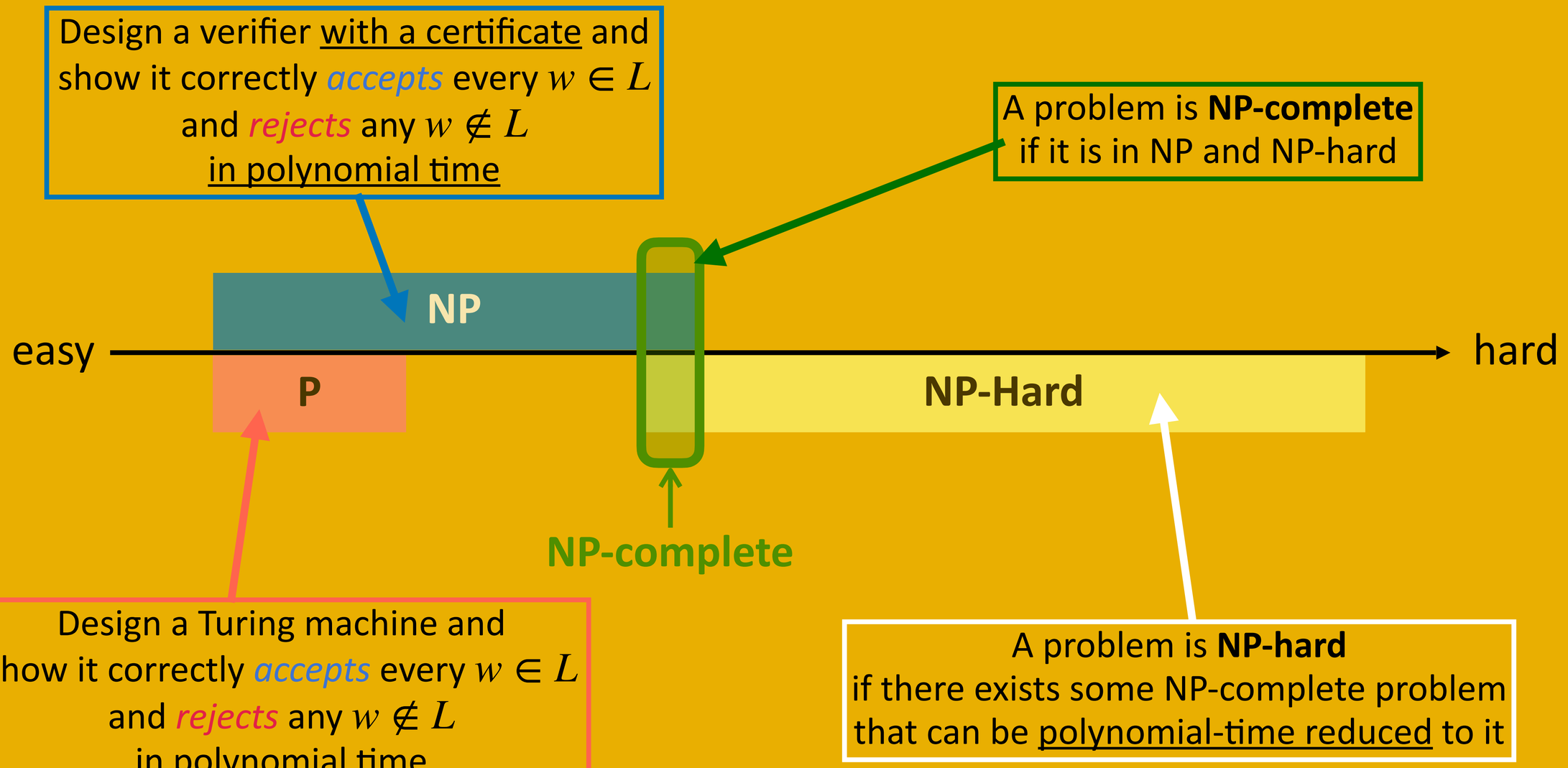
  $\leftrightarrow$ SAT is NP-complete

  <Proof Idea> For any problem in NP, there exists a Turing machine that verifies any yes-instance in polynomial time

  $\Rightarrow$ You can use first-order logic to formulate the Turing machine's behavior

  $\Rightarrow$ The formula is satisfiable if and only if the Turing machine accepts on $(w, c)$

# How to prove P, NP, NP-Hard, or NP-Complete



# CNF-SAT

- Conjunctive normal form (CNF):

$$( x_1 \vee \overline{x_2} ) \wedge ( \overline{x_1} \vee x_2 ) \wedge ( x_2 \vee x_3 \vee x_4 \vee x_5 ) \wedge ( x_2 \vee x_2 )$$

  - Variables: $x_1, x_2, \cdots, x_n$

- CNF-SAT = $\{ \langle \phi \rangle \mid \phi$ is a satisfiable conjunctive normal form Boolean formula$\}$

# CNF-SAT

- Conjunctive normal form (CNF):

$$( x_1 \vee \overline{x_2} ) \wedge ( \overline{x_1} \vee x_2 ) \wedge ( x_2 \vee x_3 \vee x_4 \vee x_5 ) \wedge ( x_2 \vee x_2 )$$

  - Variables: $x_1, x_2, \cdots, x_n$

- CNF-SAT = $\{\langle \phi \rangle \mid \phi$ is a satisfiable conjunctive normal form Boolean formula$\}$

# CNF-SAT

- Conjunctive normal form (CNF):

$$( x_1 \vee \overline{x_2} ) \wedge ( \overline{x_1} \vee x_2 ) \wedge ( x_2 \vee x_3 \vee x_4 \vee x_5 ) \wedge ( x_2 \vee x_2 )$$

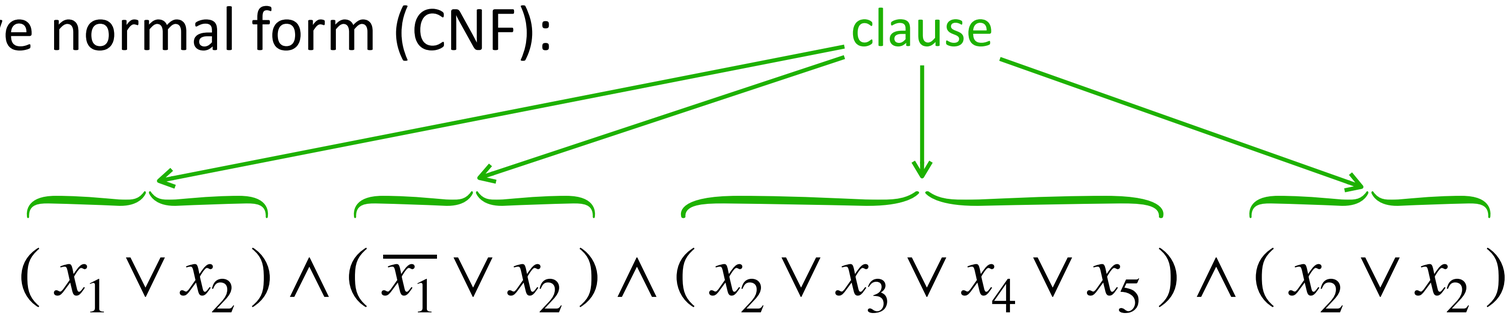
literals

  - Variables: $x_1, x_2, \cdots, x_n$

- CNF-SAT = $\{\langle \phi \rangle \mid \phi$ is a satisfiable conjunctive normal form Boolean formula$\}$

# CNF-SAT

- Conjunctive normal form (CNF):




  - Variables: $x_1, x_2, \cdots, x_n$

- CNF-SAT = $\{\langle \phi \rangle \mid \phi$ is a satisfiable conjunctive normal form Boolean formula$\}$

# CNF-SAT

- Conjunctive normal form (CNF):

$$( x_1 \vee x_2 ) \wedge ( \overline{x_1} \vee x_2 ) \wedge ( x_2 \vee x_3 \vee x_4 \vee x_5 ) \wedge ( x_2 \vee x_2 )$$

Only “or”s in each clause

  - Variables: $x_1, x_2, \cdots, x_n$

- CNF-SAT = $\{ \langle \phi \rangle \mid \phi$ is a satisfiable conjunctive normal form Boolean formula$\}$

# CNF-SAT

- Conjunctive normal form (CNF):

$$( x_1 \vee x_2 ) \wedge ( \overline{x_1} \vee x_2 ) \wedge ( x_2 \vee x_3 \vee x_4 \vee x_5 ) \wedge ( x_2 \vee x_2 )$$

"and"s between clauses

  - Variables: $x_1, x_2, \cdots, x_n$

- CNF-SAT = $\{ \langle \phi \rangle \mid \phi$ is a satisfiable conjunctive normal form Boolean formula$\}$

# 3SAT

# 3SAT

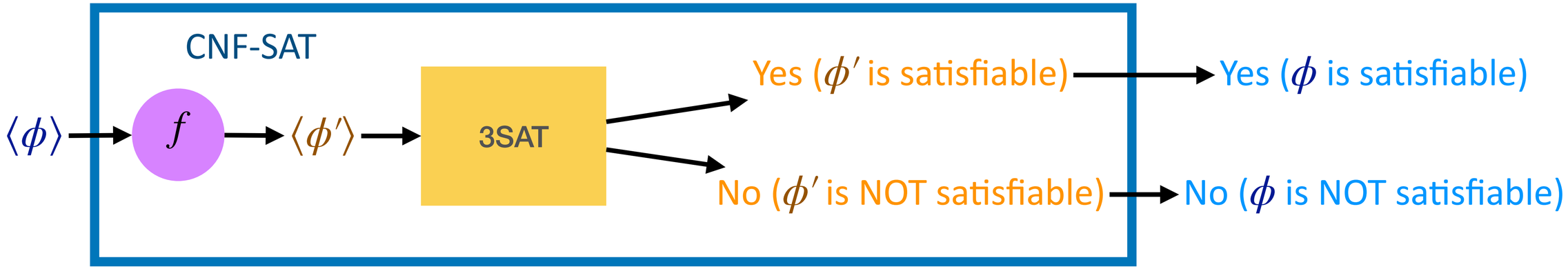
- A Boolean formula is a 3CNF-formula if it is in conjunctive normal form and **all the clauses have exactly three literals**.
  - Example: $( x_1 \vee x_2 \vee x_3 ) \wedge ( \overline{x_1} \vee x_2 \vee x_4 ) \wedge ( x_2 \vee x_3 \vee x_5 ) \wedge ( x_2 \vee x_2 \vee \overline{x_4} )$

- Theorem: 3SAT = $\{ \langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea> Polynomial time reduction from CNF-SAT (since we have known that CNF is NP-Complete)




Any CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1}) \wedge (x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2)$ Satisfiable

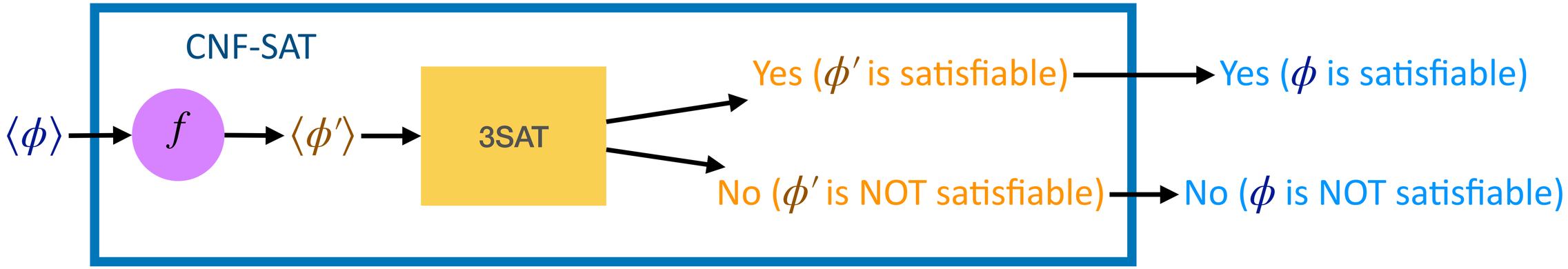
↓ Polynomial time function ⇕

A 3-CNF Boolean formula $(? \vee ? \vee ?) \wedge (? \vee ? \vee ?) \wedge (? \vee ? \vee ?) \wedge \cdots$ Satisfiable

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea> Polynomial time reduction from CNF-SAT (since we have known that CNF is NP-Complete)




Any CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1}) \wedge (x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2)$

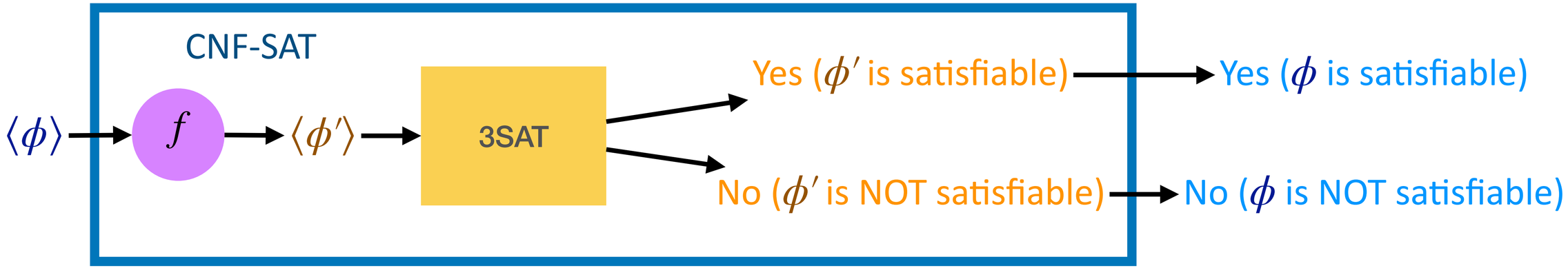
↓

A 3-CNF Boolean formula $(? \vee ? \vee ?) \wedge (? \vee ? \vee ?) \wedge (? \vee ? \vee ?) \wedge \cdots$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea> Polynomial time reduction from CNF-SAT (since we have known that CNF is NP-Complete)




Any CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1}) \wedge (x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2)$

↓

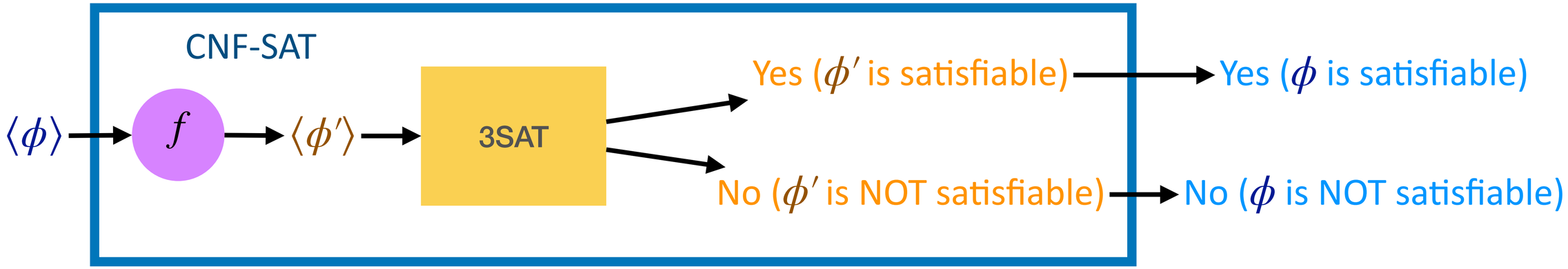
A 3-CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (? \vee ? \vee ?) \wedge (? \vee ? \vee ?) \wedge \cdots$

The **clause** is true iff the **original clause** is true

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea> Polynomial time reduction from CNF-SAT (since we have known that CNF is NP-Complete)




Any CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1}) \wedge (x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2)$

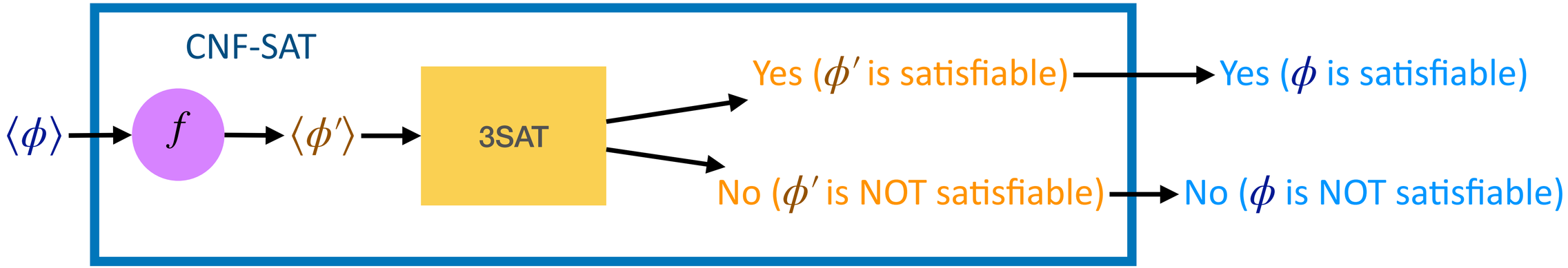
↓

A 3-CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (? \vee ? \vee ?) \wedge (? \vee ? \vee ?) \wedge \cdots$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea> Polynomial time reduction from CNF-SAT (since we have known that CNF is NP-Complete)




Any CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1}) \wedge (x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2)$

↓

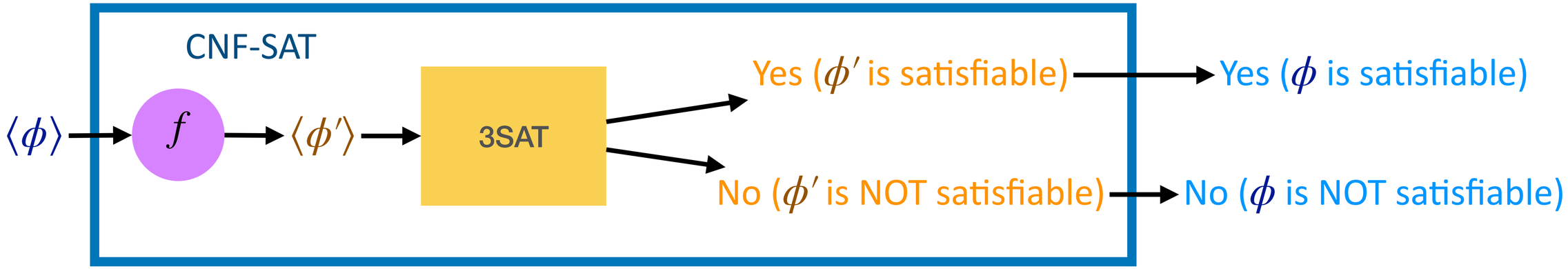
A 3-CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_1} \vee \overline{x_1}) \wedge (? \vee ? \vee ?) \wedge \cdots$

The **clause** is true iff the **original clause** is true

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea> Polynomial time reduction from CNF-SAT (since we have known that CNF is NP-Complete)




Any CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1}) \wedge (x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2)$

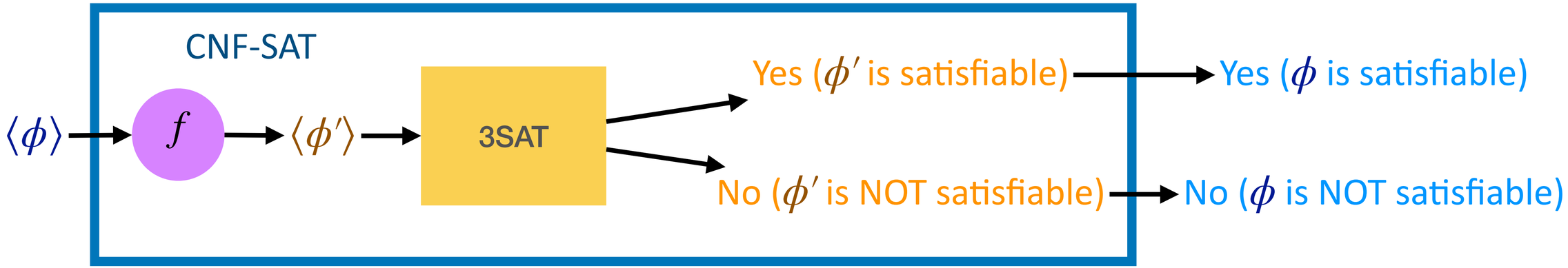
↓

A 3-CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_1} \vee \overline{x_1}) \wedge (? \vee ? \vee ?) \wedge (? \vee ? \vee ?)$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea> Polynomial time reduction from CNF-SAT (since we have known that CNF is NP-Complete)




Any CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1}) \wedge (x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2)$

↓

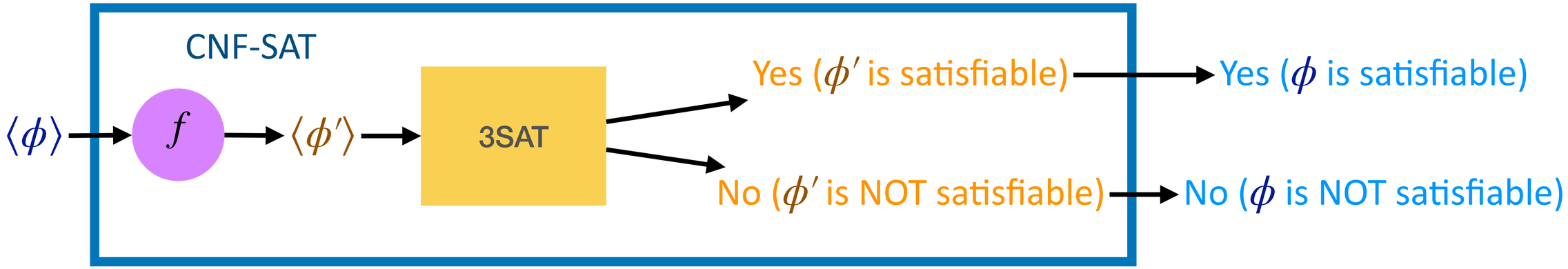
A 3-CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_1} \vee \overline{x_1}) \wedge (? \vee ? \vee ?) \wedge (x_1 \vee x_2 \vee x_2)$

The **clause** is true iff the **original clause** is true

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea> Polynomial time reduction from CNF-SAT (since we have known that CNF is NP-Complete)




Any CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1}) \wedge (x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2)$

↓

A 3-CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_1} \vee \overline{x_1}) \wedge (? \vee ? \vee ?) \wedge (x_1 \vee x_2 \vee x_2)$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\ x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k\ )$ can be replaced with the $k-2$ clauses

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1)$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1)$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1)$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1)$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2)$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2)$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3)$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3)$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee x_{k-1} \vee x_k).$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee x_{k-1} \vee x_k).$$

Dummy variable

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee x_{k-1} \vee x_k).$$

Dummy variable

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee x_{k-1} \vee x_k).$$

Dummy variable

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee x_{k-1} \vee x_k).$$

Dummy variable

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee x_{k-1} \vee x_k).$$

Dummy variable

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee x_{k-1} \vee x_k).$$

Dummy variable

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee x_{k-1} \vee x_k).$$

Dummy variable: no single dummy variable can make more than one clause TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee x_{k-1} \vee x_k).$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k)$ can be replaced with the $k-2$ clauses


If this clause is TRUE, at least one literal is 1

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee x_{k-1} \vee x_k).$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\overset{0}{x_1} \vee \overset{0}{x_2} \vee \overset{0}{x_3} \vee \overset{1}{x_4} \vee \cdots \vee \overset{0}{x_{k-1}} \vee \overset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee x_{k-1} \vee x_k).$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{0}{x_3} \vee \underset{1}{x_4} \vee \cdots \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(\underset{0}{x_1} \vee \underset{0}{x_2} \vee d_1) \wedge (\overline{d_1} \vee \underset{0}{x_3} \vee d_2) \wedge (\overline{d_2} \vee \underset{1}{x_4} \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k}).$$

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{0}{x_3} \vee \underset{1}{x_4} \vee \cdots \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(\underset{0}{x_1} \vee \underset{0}{x_2} \vee d_1) \wedge (\overline{d_1} \vee \underset{0}{x_3} \vee d_2) \wedge \boxed{(\overline{d_2} \vee \underset{1}{x_4} \vee d_3)} \wedge \cdots \wedge (\overline{d_{k-3}} \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k}).$$

TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{0}{x_3} \vee \underset{1}{x_4} \vee \cdots \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(\underset{0}{x_1} \vee \underset{0}{x_2} \vee d_1) \wedge (\overline{d_1} \vee \underset{0}{x_3} \vee d_2) \wedge \boxed{(\underset{0}{\overline{d_2}} \vee \underset{1}{x_4} \vee \underset{0}{d_3})} \wedge \cdots \wedge (\overline{d_{k-3}} \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k}).$$

TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{0}{x_3} \vee \underset{1}{x_4} \vee \cdots \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(\underset{0}{x_1} \vee \underset{0}{x_2} \vee d_1) \wedge (\overline{d_1} \vee \underset{0}{x_3} \vee \underset{1}{d_2}) \wedge \boxed{(\underset{0}{\overline{d_2}} \vee \underset{1}{x_4} \vee \underset{0}{d_3})} \wedge \cdots \wedge (\overline{d_{k-3}} \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k}).$$

TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{0}{x_3} \vee \underset{1}{x_4} \vee \cdots \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(\underset{0}{x_1} \vee \underset{0}{x_2} \vee d_1) \wedge (\overline{\underset{0}{d_1}} \vee \underset{0}{x_3} \vee \underset{1}{d_2}) \wedge \boxed{(\overline{\underset{0}{d_2}} \vee \underset{1}{x_4} \vee \underset{0}{d_3})} \wedge \cdots \wedge (\overline{d_{k-3}} \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k}).$$

TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{0}{x_3} \vee \underset{1}{x_4} \vee \cdots \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{1}{d_1}) \wedge (\underset{0}{\overline{d_1}} \vee \underset{0}{x_3} \vee \underset{1}{d_2}) \wedge \boxed{(\underset{0}{\overline{d_2}} \vee \underset{1}{x_4} \vee \underset{0}{d_3})} \wedge \cdots \wedge (\underset{0}{\overline{d_{k-3}}} \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k}).$$

TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{0}{x_3} \vee \underset{1}{x_4} \vee \cdots \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{1}{d_1}) \wedge (\underset{0}{\overline{d_1}} \vee \underset{0}{x_3} \vee \underset{1}{d_2}) \wedge (\underset{0}{\overline{d_2}} \vee \underset{1}{x_4} \vee \underset{0}{d_3}) \wedge \cdots \wedge (\underset{1}{\overline{d_{k-3}}} \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k}).$$

TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{0}{x_3} \vee \underset{1}{x_4} \vee \cdots \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{1}{d_1}) \wedge (\underset{0}{\overline{d_1}} \vee \underset{0}{x_3} \vee \underset{1}{d_2}) \wedge (\underset{0}{\overline{d_2}} \vee \underset{1}{x_4} \vee \underset{0}{d_3}) \wedge \cdots \wedge (\underset{1}{\overline{d_{k-3}}} \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k}).$$

TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{0}{x_3} \vee \underset{1}{x_4} \vee \cdots \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{1}{d_1}) \wedge (\underset{0}{\overline{d_1}} \vee \underset{0}{x_3} \vee \underset{1}{d_2}) \wedge (\underset{0}{\overline{d_2}} \vee \underset{1}{x_4} \vee \underset{0}{d_3}) \wedge \cdots \wedge (\underset{0}{\overline{d_{k-3}}} \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k}).$$

1 0 1

TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{0}{x_3} \vee \underset{1}{x_4} \vee \cdots \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{1}{d_1}) \wedge (\underset{0}{\overline{d_1}} \vee \underset{0}{x_3} \vee \underset{1}{d_2}) \wedge (\underset{0}{\overline{d_2}} \vee \underset{1}{x_4} \vee \underset{0}{d_3}) \wedge \cdots \wedge (\underset{0}{\overline{d_{k-3}}} \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k}).$$

1 0 1 0

TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{0}{x_3} \vee \underset{1}{x_4} \vee \cdots \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{1}{d_1}) \wedge (\underset{0}{\overline{d_1}} \vee \underset{0}{x_3} \vee \underset{1}{d_2}) \wedge (\underset{0}{\overline{d_2}} \vee \underset{1}{x_4} \vee \underset{0}{d_3}) \wedge \cdots \wedge (\underset{1\ 0\ 1\ 0\ 1}{\overline{d_{k-3}}} \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k}).$$

TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\underset{0}{x_1} \vee \underset{0}{x_2} \vee \underset{0}{x_3} \vee \underset{1}{x_4} \vee \cdots \vee \underset{0}{x_{k-1}} \vee \underset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is TRUE, at least one literal is 1

$$\underset{0\ 0\ 1}{(x_1 \vee x_2 \vee d_1)} \wedge \underset{0\ 0\ 1}{(\overline{d_1} \vee x_3 \vee d_2)} \wedge \underset{0\ 1\ 0}{(\overline{d_2} \vee x_4 \vee d_3)} \wedge \cdots \wedge \underset{1\ 0\ 1\ 0}{(\overline{d_{k-3}} \vee x_{k-1} \vee x_k)}.$$

TRUE

If the (big) clause is TRUE, there exists an assignment to $\phi'$ such that the sequence of 3-clauses are all TRUE.

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\overset{0}{x_1} \vee \overset{0}{x_2} \vee \overset{0}{x_3} \vee \overset{0}{x_4} \vee \cdots \vee \overset{0}{x_{k-1}} \vee \overset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is FALSE, every literal is 0

$$(\overset{0}{x_1} \vee \overset{0}{x_2} \vee d_1) \wedge (\overline{d_1} \vee \overset{0}{x_3} \vee d_2) \wedge (\overline{d_2} \vee \overset{0}{x_4} \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee \overset{0}{x_{k-1}} \vee \overset{0}{x_k}).$$

If the (big) clause is FALSE, there exists NO assignment to $\phi'$ such that the sequence of 3-clauses are all TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof Idea>

For each clause that has more than 3 literals, we split it into several clauses and add additional (dummy) variables

For example: $(\overset{0}{x_1} \vee \overset{0}{x_2} \vee \overset{0}{x_3} \vee \overset{0}{x_4} \vee \cdots \vee \overset{0}{x_{k-1}} \vee \overset{0}{x_k})$ can be replaced with the $k-2$ clauses

If this clause is FALSE, every literal is 0

$$(\overset{0}{x_1} \vee \overset{0}{x_2} \vee d_1) \wedge (\overline{d_1} \vee \overset{0}{x_3} \vee d_2) \wedge (\overline{d_2} \vee \overset{0}{x_4} \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee \overset{0}{x_{k-1}} \vee \overset{0}{x_k}).$$

At most one TRUE

At most one TRUE

If the (big) clause is FALSE, there exists NO assignment to $\phi'$ such that the sequence of 3-clauses are all TRUE since no single dummy variable can make more than one clause TRUE

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof> Polynomial time reduction from CNF-SAT (since we have known that CNF is NP-Complete).

To reduce CNF-SAT to 3SAT, we convert any CNF-formula $F$ into a 3CNF-formula $F'$, with $F$ is satisfiable if and only if $F'$ is satisfiable:

First, let $C_1$, $C_2$, $\cdots$, $C_m$ be the clauses in $F$. If $F$ is a 3CNF-formula, we just set $F' = F$.

Otherwise, the only reasons why $F$ is not a 3CNF-formula are:

1. Some clauses $C_i$ has less than 3 literals, or
2. Some clauses $C_i$ has more than 3 literals.

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof (cont.)>

For each clause that has less than 3 literals, we duplicate one of the literals until the total number is three.

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof (cont.)>

For each clause that has more than 3 literals, we split it into several clauses and add additional variables to preserve the satisfiability or non-satisfiability of the original clause: each of the clauses $( x_1 \vee x_2 \vee x_3 \vee x_4 \vee \cdots \vee x_{k-1} \vee x_k )$ can be replaced with the $k - 2$ clauses

$$( x_1 \vee x_2 \vee d_1 ) \wedge ( \overline{d_1} \vee x_3 \vee d_2 ) \wedge ( \overline{d_2} \vee x_4 \vee d_3 ) \wedge \cdots \wedge ( \overline{d_{k-3}} \vee x_{k-1} \vee x_k ).$$

The conversion can be done in $O(n \cdot m \cdot k)$ time, where $n$ is the number of variables, $m$ is the number of clauses, and $k$ is the number of literals in the largest clause.

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof (cont.)>

Now we prove that the satisfiability or non-satisfiability of the 3SAT problem is preserved. That is, $F$ is satisfiable if and only if $F'$ is satisfiable.

If $F$ is satisfiable, there exists a corresponding truth assignment in $F'$ such that $F'$ = 1 (TRUE). For each of the clauses with less than or equal to 3 literals in $F$ which is true, the corresponding clause in $F'$ is also true since we only duplicate the literals from the same clause. For each clause with more than 3 literals in $F$, since $F$ is satisfiable, there must be at least one literal $x_t$ which has value 1. There exists a corresponding true assignment in $F'$: $d_i$ = 1 for all $i \leq t-2$, and $d_i$ = 0 for all $i \geq t-1$.

# 3SAT

- Theorem: 3SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$ is NP-Hard

<Proof (cont.)>

If $F'$ is satisfiable, the corresponding truth assignment for variables $x_1, x_2, \cdots, x_n$ makes $F$ = 1 (TRUE). For each of the clauses with less than or equal to 3 literals in $F$, all these clauses are TRUE since duplicating the literals from the same clause does not change the TRUE/FALSE of a clause. For each clause with more than 3 literals in $F$, since $F'$ is satisfiable, the corresponding clauses in $F'$ must be all TRUE as no truth value of a dummy literal can solely make more than two clauses TRUE.

# What Happened

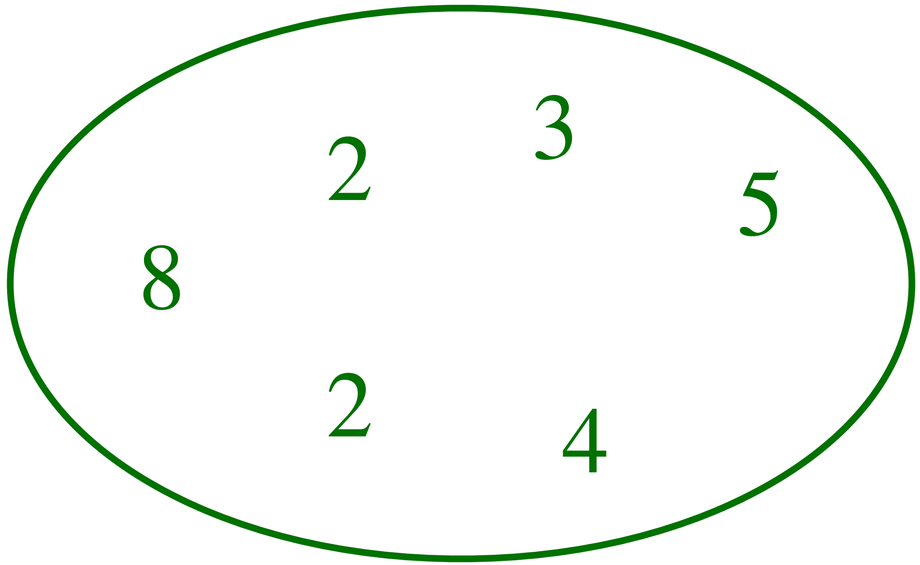
- CNF-SAT $\leq_p$ 3SAT
  - There may be some clause in the CNF-SAT instance $\phi$ that has fewer than 3 literals
    - $\Rightarrow$ Duplicate existed literal $(x_1 \vee x_2) \rightarrow (x_1 \vee x_2 \vee x_2)$
  - There may be some clause in $\phi$ that has more than 3 literals
    - $\Rightarrow$ make a chain of 3-clauses in $\phi'$, using dummy variables

$$(x_1 \vee x_2 \vee d_1) \wedge (\overline{d_1} \vee x_3 \vee d_2) \wedge (\overline{d_2} \vee x_4 \vee d_3) \wedge \cdots \wedge (\overline{d_{k-3}} \vee x_{k-1} \vee x_k)$$

- Each clause in $\phi$ is *TRUE* if and only if the corresponding (chain of) clauses in $\phi'$ are all *TRUE*

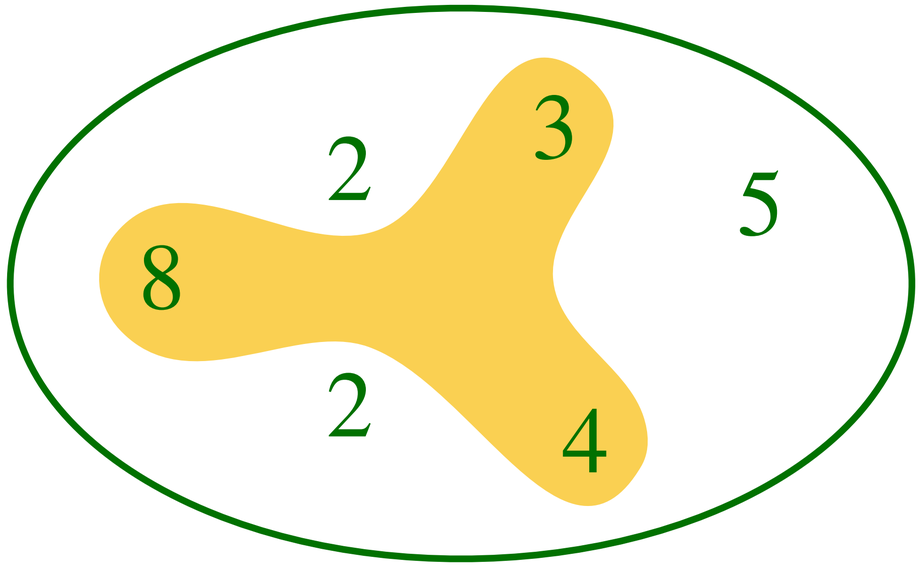
# SUBSET-SUM

- SUBSET-SUM $= \{\langle S, t\rangle \mid S = \{x_1, \cdots, x_k\}$ and there exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = t\}$
  - Ex: $S$ = {2, 2, 3, 4, 5, 8}, $t$ = 12



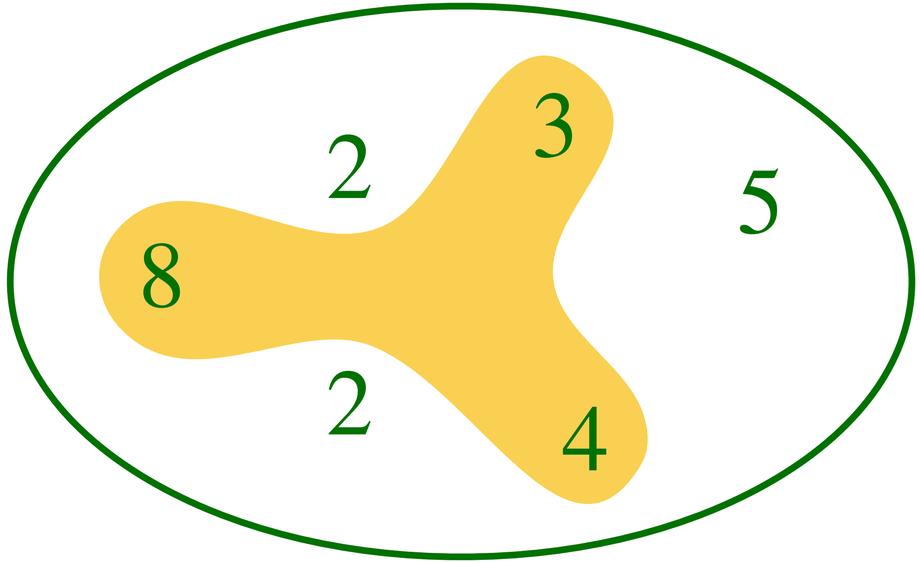
# SUBSET-SUM

- SUBSET-SUM $= \{\langle S, t\rangle \mid S = \{x_1, \cdots, x_k\}$ and there exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = t\}$
  - Ex: $S$ = {2, 2, 3, 4, 5, 8}, $t$ = 15 ✅ Yes-instance



# SUBSET-SUM

- SUBSET-SUM $= \{\langle S, t\rangle \mid S = \{x_1, \cdots, x_k\}$ and there exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = t\}$
  - Ex: $S$ = {2, 2, 3, 4, 5, 8}, $t$ = 15 Yes-instance
  - Ex: $S$ = {2, 2, 3, 4, 5, 8}, $t$ = 23 No-instance



# SUBSET-SUM

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- SUBSET-SUM $= \{\langle S, t \rangle \mid S = \{x_1, \cdots, x_k\}$ and there exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = t\}$

# SUBSET-SUM

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$

  $\ell$ variables $x_1, x_2, \cdots, x_\ell$

  $k$ clauses $c_1, c_2, \cdots, c_k$

- SUBSET-SUM $= \{\langle S, t \rangle \mid S = \{x_1, \cdots, x_k\}$ and there exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = t\}$

# SUBSET-SUM

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$

  $\ell$ variables $x_1, x_2, \cdots, x_\ell$

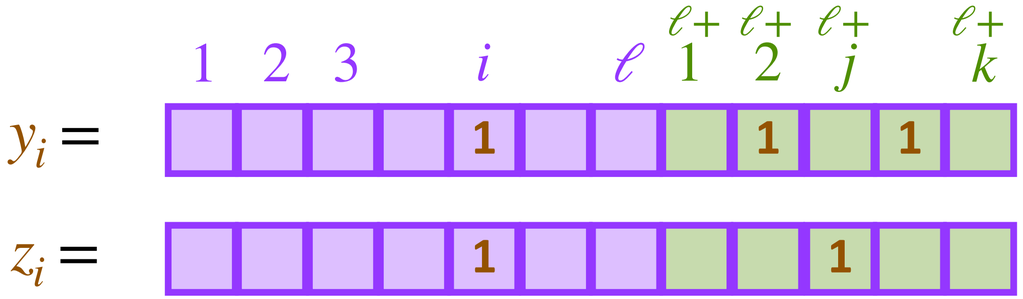
  $k$ clauses $c_1, c_2, \cdots, c_k$

- SUBSET-SUM $= \{\langle S, t \rangle \mid S = \{x_1, \cdots, x_k\}$ and there exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = t\}$

For every variable $x_i$ in 3SAT, create two numbers $y_i$ and $z_i$ in $S$:

- The $i^{\text{th}}$ decimal of $y_i$ is $1$, and all the decimals in the first $l$ decimals of $y_i$ are $0$. The $(\ell + j)^{\text{th}}$ decimal of $y_i$ is $1$ if and only if the clause $c_j$ contains literal $x_i$.
- The $i^{\text{th}}$ decimal of $z_i$ is $1$, and all the decimals in the first $l$ decimals of $z_i$ are $0$. The $(\ell + j)^{\text{th}}$ decimal of $z_i$ is $1$ if and only if the clause $c_j$ contains literal $\overline{x_i}$.

$x_i$



# SUBSET-SUM

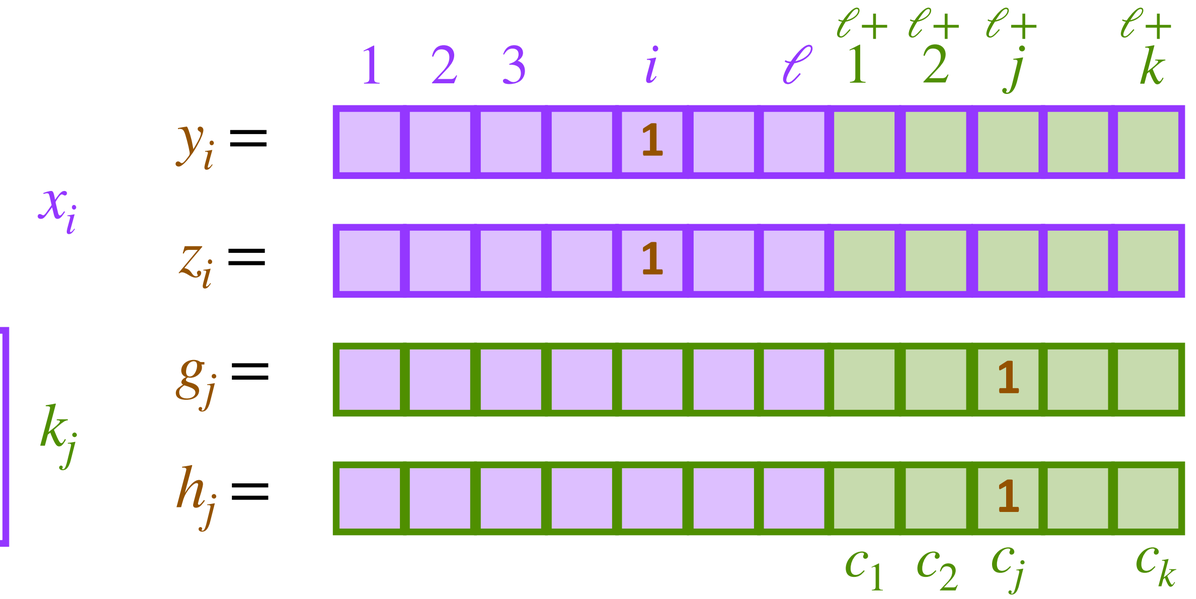
- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$

  $\ell$ variables $x_1, x_2, \cdots, x_\ell$

  $k$ clauses $c_1, c_2, \cdots, c_k$

- SUBSET-SUM $= \{\langle S, t \rangle \mid S = \{x_1, \cdots, x_k\}$ and there exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = t\}$

For every clause $c_j$ in 3SAT, create two numbers $g_j$ and $h_j$ in $S$. These two numbers are equal and consists of single $1$ at the $(\ell + j)^{\text{th}}$ decimal and all other decimals are $0$'s.



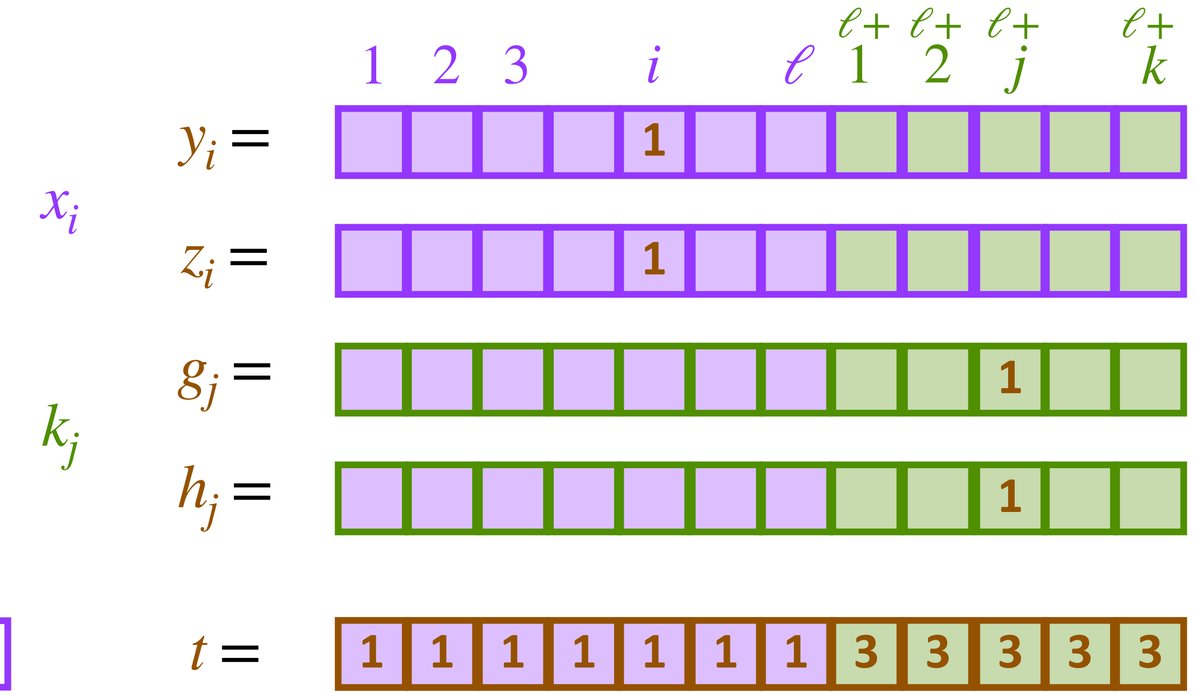
# SUBSET-SUM

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$

  $\ell$ variables $x_1, x_2, \cdots, x_\ell$

  $k$ clauses $c_1, c_2, \cdots, c_k$

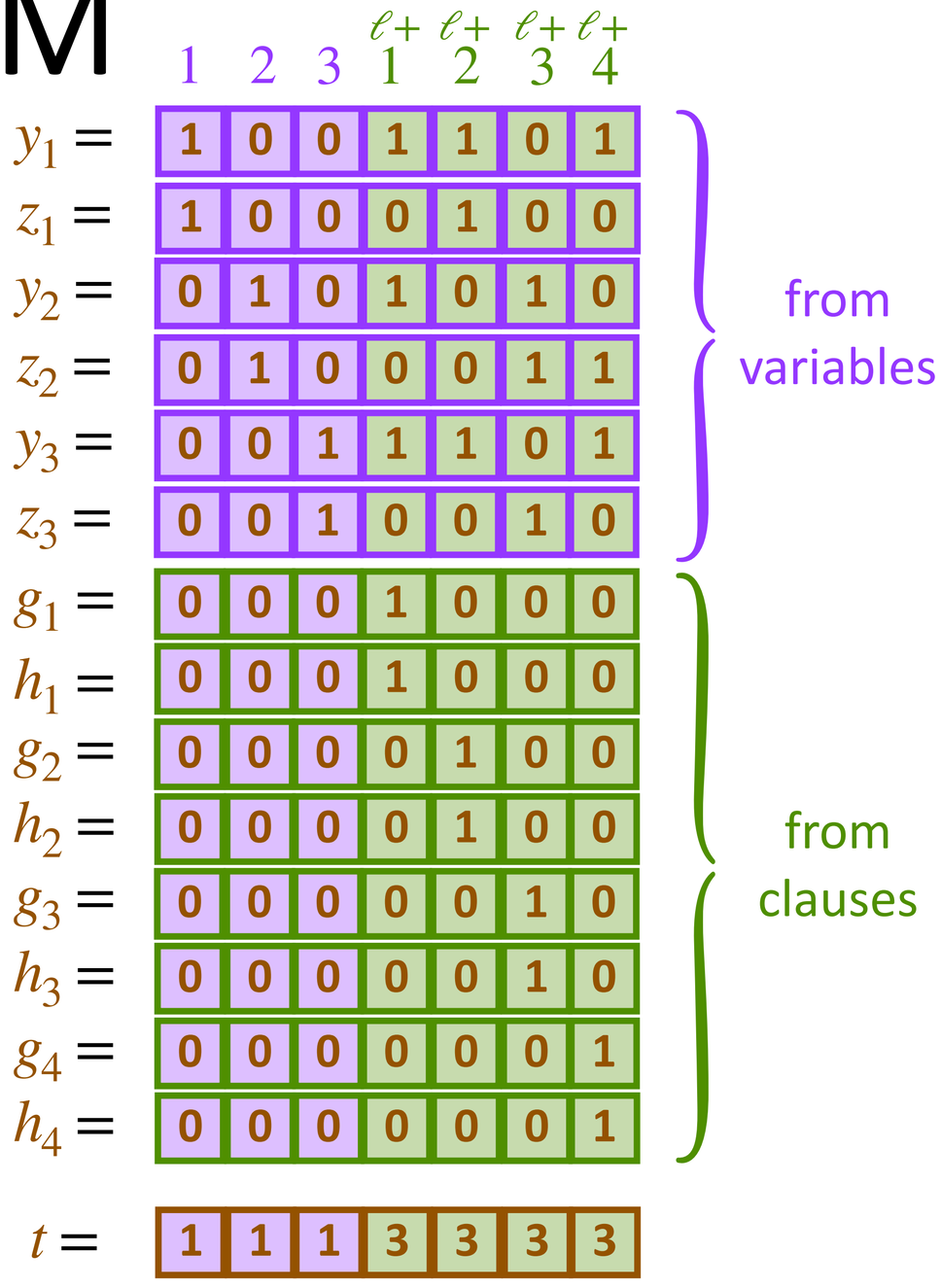
- SUBSET-SUM $= \{\langle S, t \rangle \mid S = \{x_1, \cdots, x_k\}$ and there exists a subset $T = \{y_1, \cdots, y_m\} \subset S$ such that $\Sigma_{y_i \in T} y_i = t\}$




Finally, set the target $t$ with $l$ 1's followed by $k$ 3's.

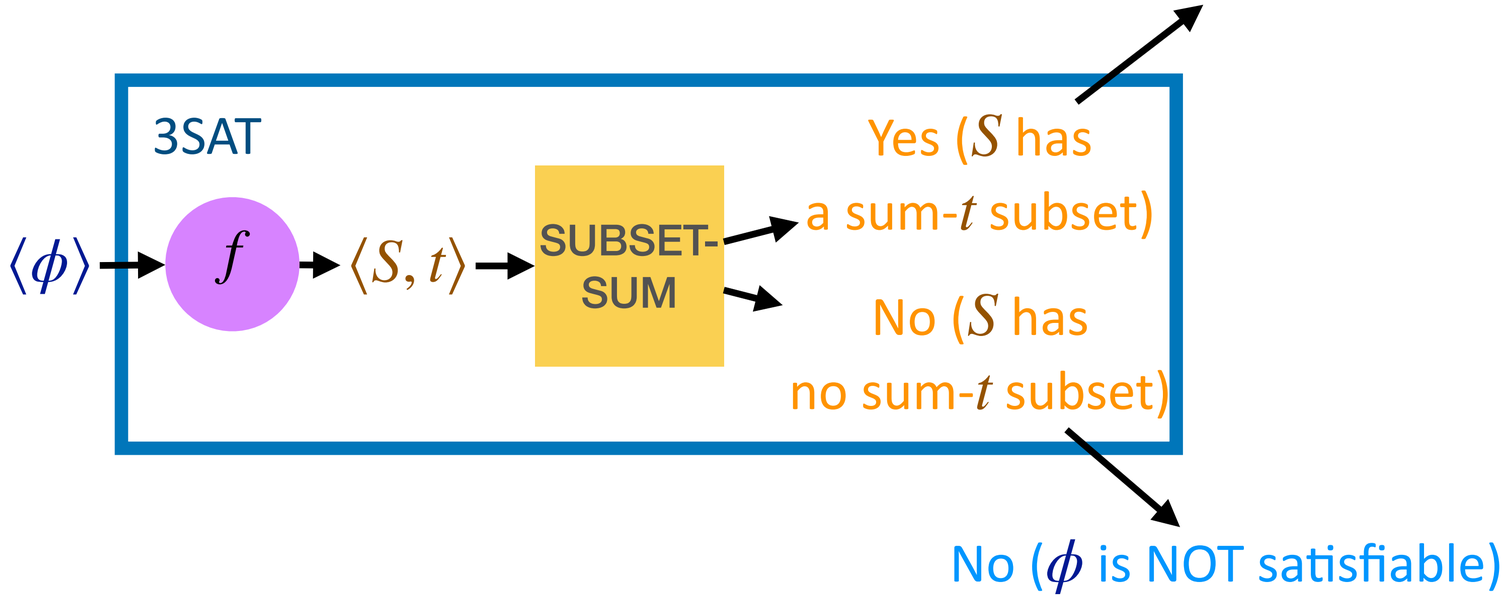
# SUBSET-SUM

$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x_1} \vee x_3) \wedge (x_2 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_1 \vee x_1 \vee \bar{x_2})$



# SUBSET-SUM

$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x_1} \vee x_3) \wedge (x_2 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_1 \vee x_1 \vee \bar{x_2})$

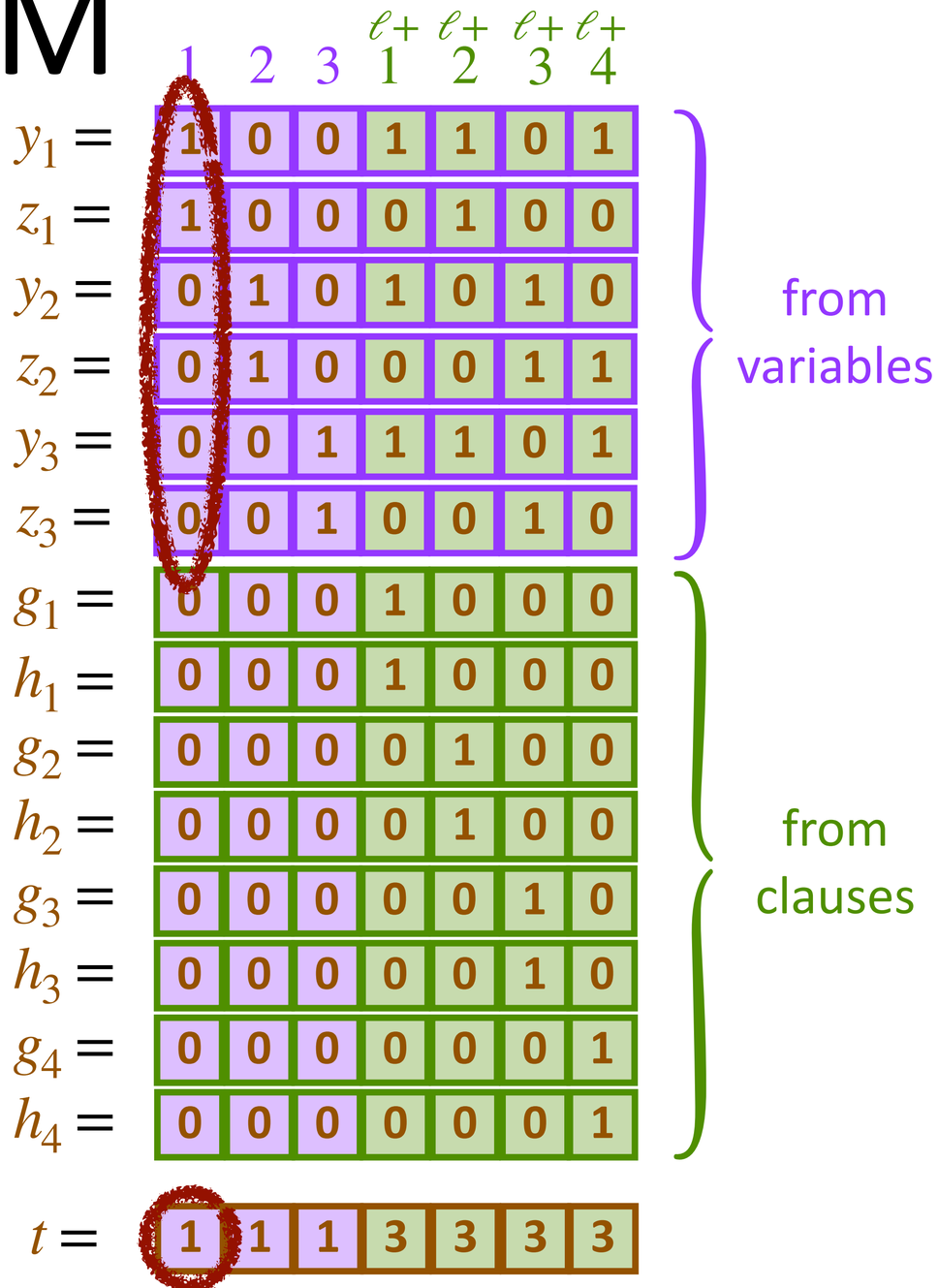



| | 1 | 2 | 3 | $\ell+1$ | $\ell+2$ | $\ell+3$ | $\ell+4$ | |
|---|---|---|---|---|---|---|---|---|
| $y_1 =$ | 1 | 0 | 0 | 1 | 1 | 0 | 1 | from variables |
| $z_1 =$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $y_2 =$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| $z_2 =$ | 0 | 1 | 0 | 0 | 0 | 1 | 1 | |
| $y_3 =$ | 0 | 0 | 1 | 1 | 1 | 0 | 1 | |
| $z_3 =$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| $g_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | from clauses |
| $h_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |
| $g_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $h_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $g_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $h_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $g_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $h_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $t =$ | 1 | 1 | 1 | 3 | 3 | 3 | 3 | |

# SUBSET-SUM

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x_1} \vee x_3) \wedge (x_2 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_1 \vee x_1 \vee \bar{x_2})$$

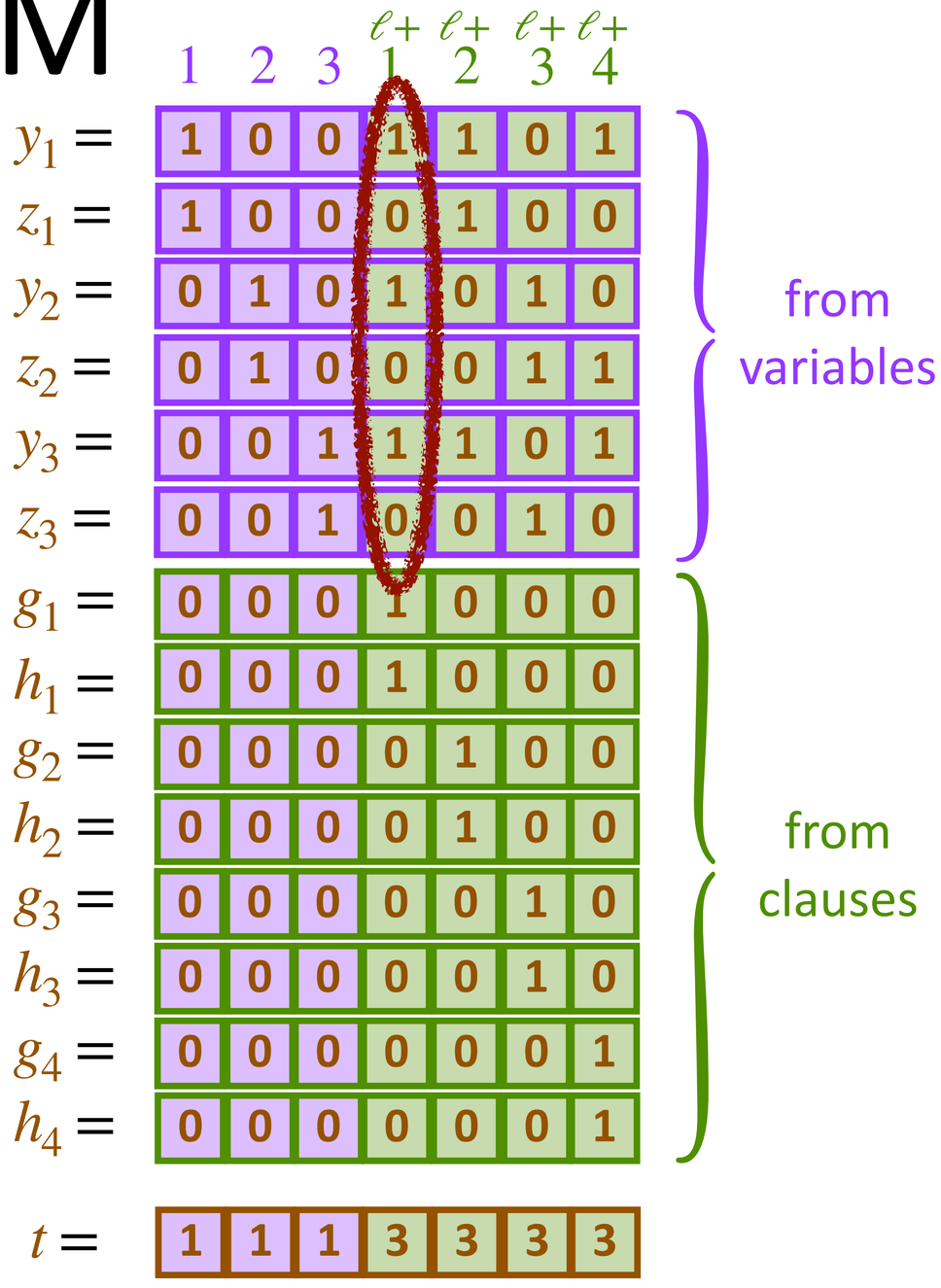
If there is a subset with sum $t$, a $1$'s of the first $\ell$ decimals must come from a $y_i$ or $z_i$ for some $x_i$.



# SUBSET-SUM

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x_1} \vee x_3) \wedge (x_2 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_1 \vee x_1 \vee \bar{x_2})$$

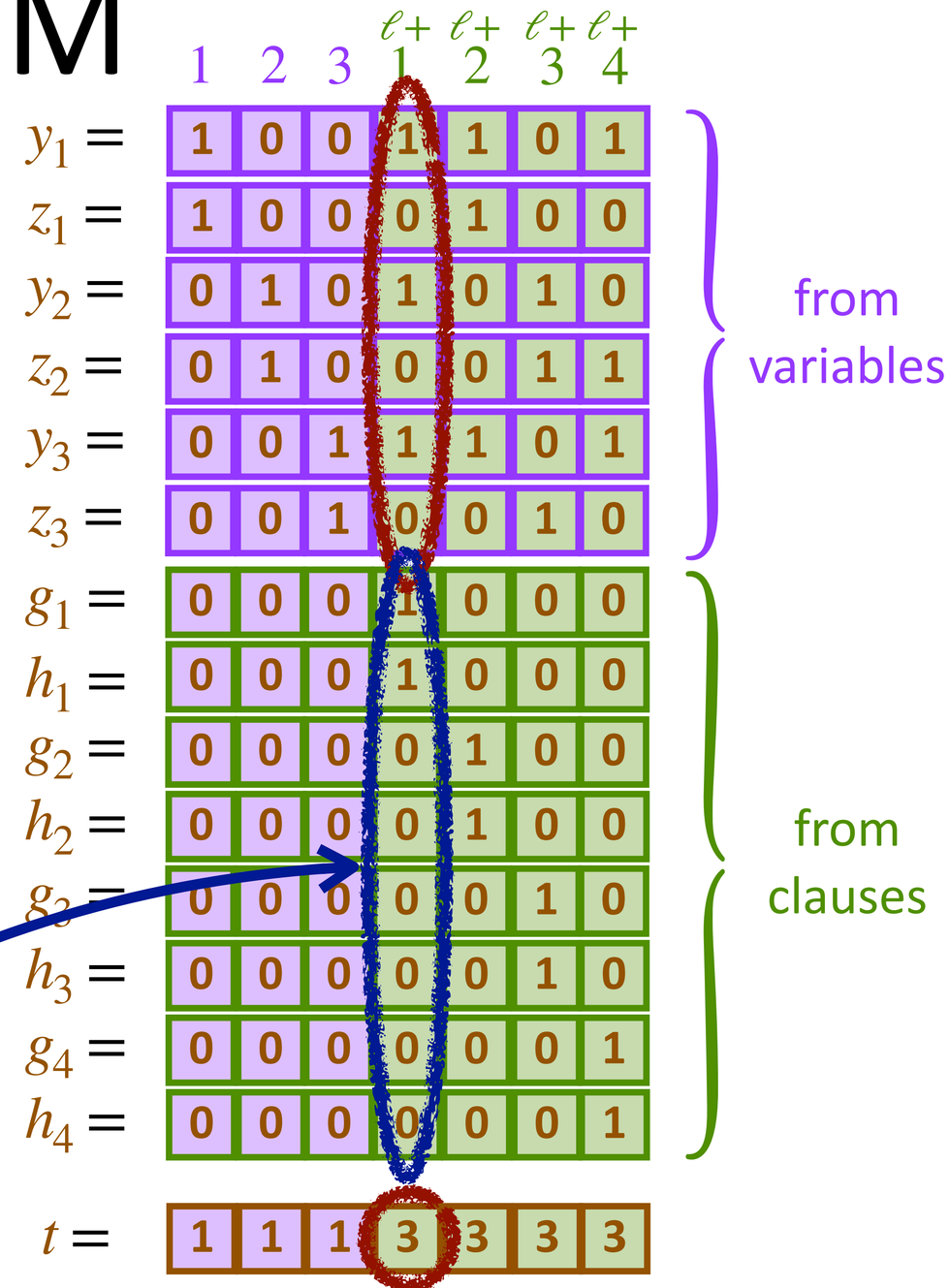
There are exact three $1$'s in each column representing the $j^{\text{th}}$ clause.



# SUBSET-SUM

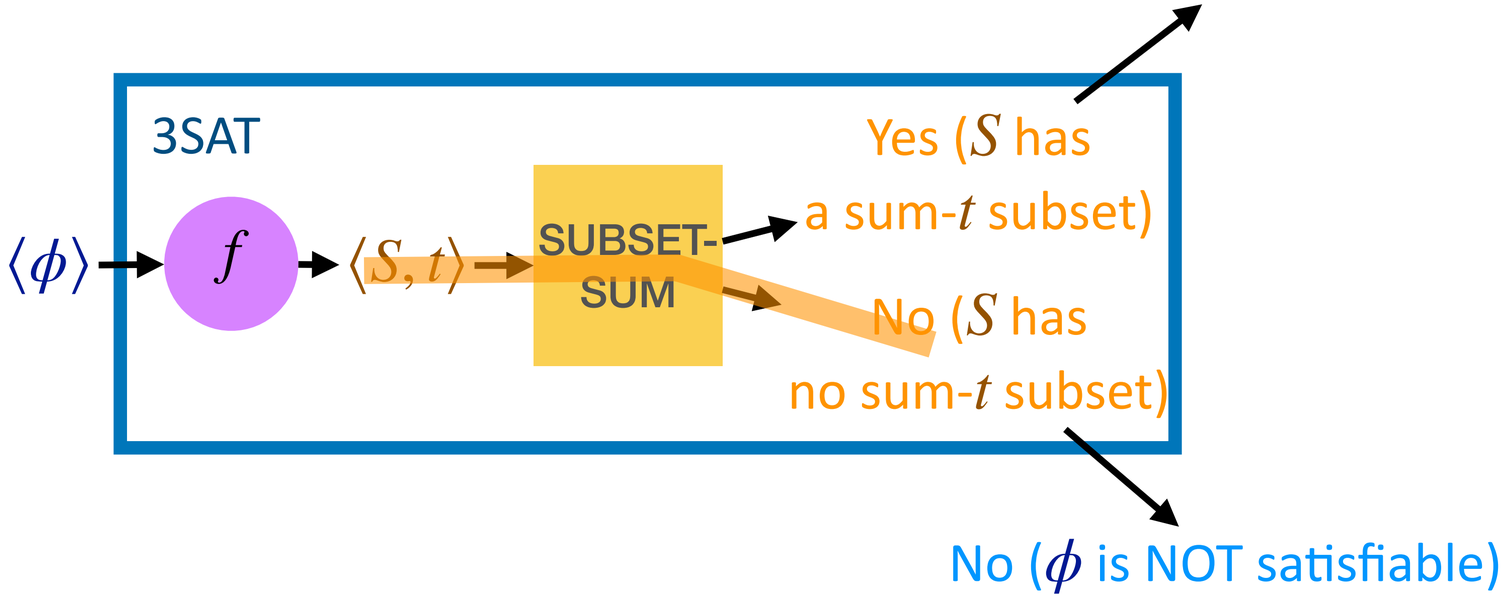
$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x_1} \vee x_3) \wedge (x_2 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_1 \vee x_1 \vee \bar{x_2})$

If there is a subset with sum $t$, there must be at least $1$ contributed by the numbers from the variables.



# SUBSET-SUM

$$\overset{T}{(x_1 \vee x_2 \vee x_3)} \wedge \overset{F}{(x_1 \vee \bar{x}_1 \vee x_3)} \wedge \overset{F}{(x_2 \vee \bar{x}_2 \vee \bar{x}_3)} \wedge (x_1 \vee x_1 \vee \bar{x}_2)$$

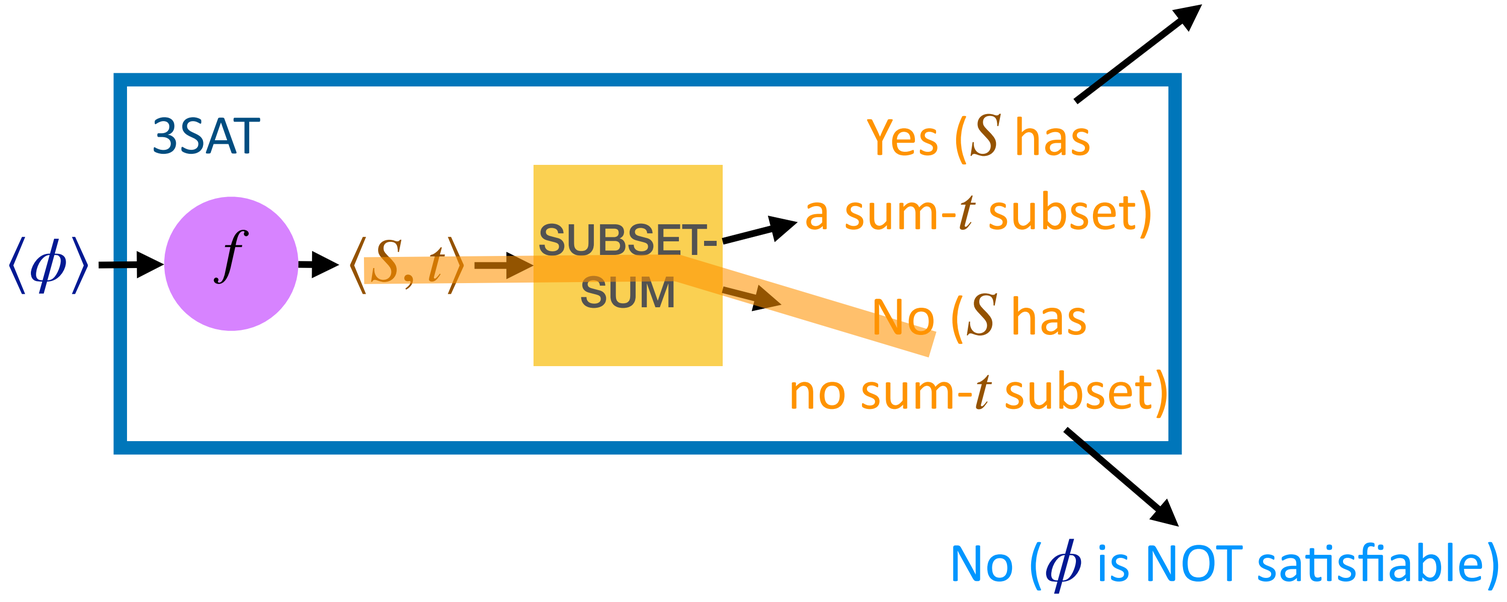



If $\phi$ is satisfiable, there exists an assignment ($x_1 = T$, $x_2 = F$, $\cdots$) such that every clause has at least one $T$.
If $x_i = T$, choose $y_i$ as part of the subset. Otherwise, choose $z_i$.

| | 1 | 2 | 3 | $\ell$+ 1 | $\ell$+ 2 | $\ell$+ 3 | $\ell$+ 4 | |
|---|---|---|---|---|---|---|---|---|
| $y_1 =$ | 1 | 0 | 0 | 1 | 1 | 0 | 1 | from variables |
| $z_1 =$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $y_2 =$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| $z_2 =$ | 0 | 1 | 0 | 0 | 0 | 1 | 1 | |
| $y_3 =$ | 0 | 0 | 1 | 1 | 1 | 0 | 1 | |
| $z_3 =$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| $g_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | from clauses |
| $h_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |
| $g_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $h_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $g_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $h_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $g_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $h_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $t =$ | 1 | 1 | 1 | 3 | 3 | 3 | 3 | |

# SUBSET-SUM

$$\overset{T}{(x_1 \vee x_2 \vee x_3)} \wedge \overset{F}{(x_1 \vee \bar{x}_1 \vee x_3)} \wedge \overset{F}{(x_2 \vee \bar{x}_2 \vee \bar{x}_3)} \wedge (x_1 \vee x_1 \vee \bar{x}_2)$$



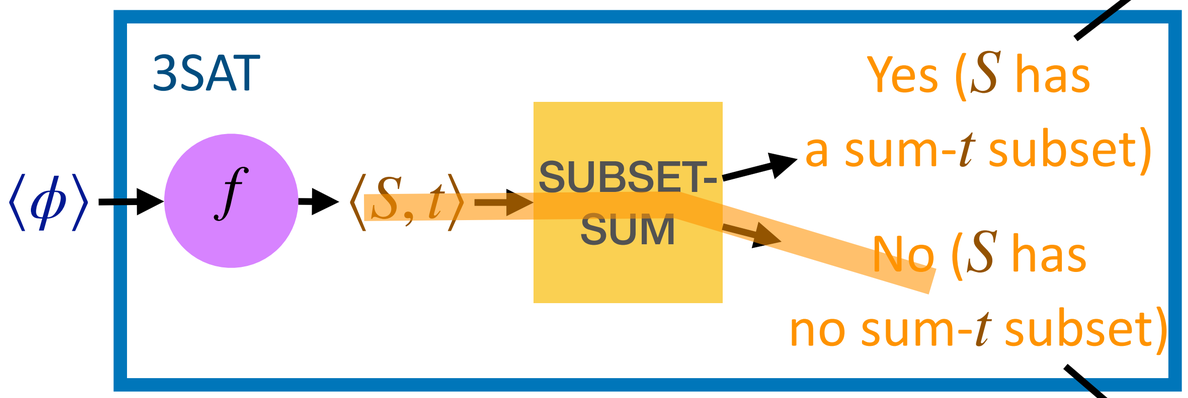

| | | 1 | 2 | 3 | $\ell+1$ | $\ell+2$ | $\ell+3$ | $\ell+4$ | |
|---|---|---|---|---|---|---|---|---|---|
| $y_1$ | = | 1 | 0 | 0 | 1 | 1 | 0 | 1 | from variables |
| $z_1$ | = | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $y_2$ | = | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| $z_2$ | = | 0 | 1 | 0 | 0 | 0 | 1 | 1 | |
| $y_3$ | = | 0 | 0 | 1 | 1 | 1 | 0 | 1 | |
| $z_3$ | = | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| $g_1$ | = | 0 | 0 | 0 | 1 | 0 | 0 | 0 | from clauses |
| $h_1$ | = | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |
| $g_2$ | = | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $h_2$ | = | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $g_3$ | = | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $h_3$ | = | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $g_4$ | = | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $h_4$ | = | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $t$ | = | 1 | 1 | 1 | 3 | 3 | 3 | 3 | |

If $\phi$ is satisfiable, there exists an assignment ($x_1 = T$, $x_2 = F$, $\cdots$) such that every clause has at least one $T$.
If $x_i = T$, choose $y_i$ as part of the subset. Otherwise, choose $z_i$. Further select enough of the $g_j$ and $h_j$ numbers to bring each of the last $k$ decimals up to 3.

# SUBSET-SUM

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_1 \vee x_3) \wedge (x_2 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_1 \vee \bar{x}_2)$$
($T$ for $x_1$, $F$ for $x_3$, $F$ for $x_2$)

Yes ($\phi$ is satisfiable)



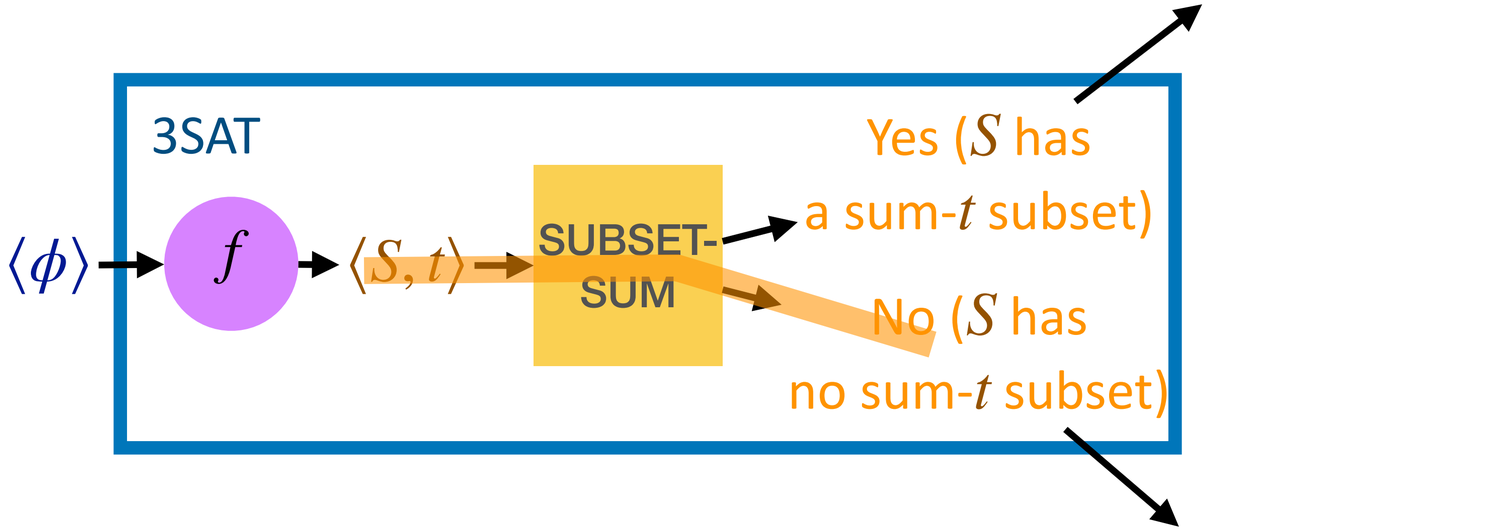

No ($\phi$ is NOT satisfiable)

If $\phi$ is satisfiable, there exists an assignment ($x_1 = T$, $x_2 = F$, $\cdots$) such that every clause has at least one $T$.
If $x_i = T$, choose $y_i$ as part of the subset. Otherwise, choose $z_i$. Further select enough of the $g_j$ and $h_j$ numbers to bring each of the last $k$ decimals up to 3.

| | 1 | 2 | 3 | $\ell$+ 1 | $\ell$+ 2 | $\ell$+ 3 | $\ell$+ 4 | |
|---|---|---|---|---|---|---|---|---|
| $y_1 =$ | 1 | 0 | 0 | 1 | 1 | 0 | 1 | from variables |
| $z_1 =$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $y_2 =$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| $z_2 =$ | 0 | 1 | 0 | 0 | 0 | 1 | 1 | |
| $y_3 =$ | 0 | 0 | 1 | 1 | 1 | 0 | 1 | |
| $z_3 =$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| $g_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | from clauses |
| $h_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |
| $g_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $h_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $g_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $h_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $g_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $h_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $t =$ | 1 | 1 | 1 | 3 | 3 | 3 | 3 | |

# SUBSET-SUM

$$\overset{T}{(x_1 \vee x_2 \vee x_3)} \wedge \overset{F}{(x_1 \vee \bar{x}_1 \vee x_3)} \wedge \overset{F}{(x_2 \vee \bar{x}_2 \vee \bar{x}_3)} \wedge (x_1 \vee x_1 \vee \bar{x}_2)$$

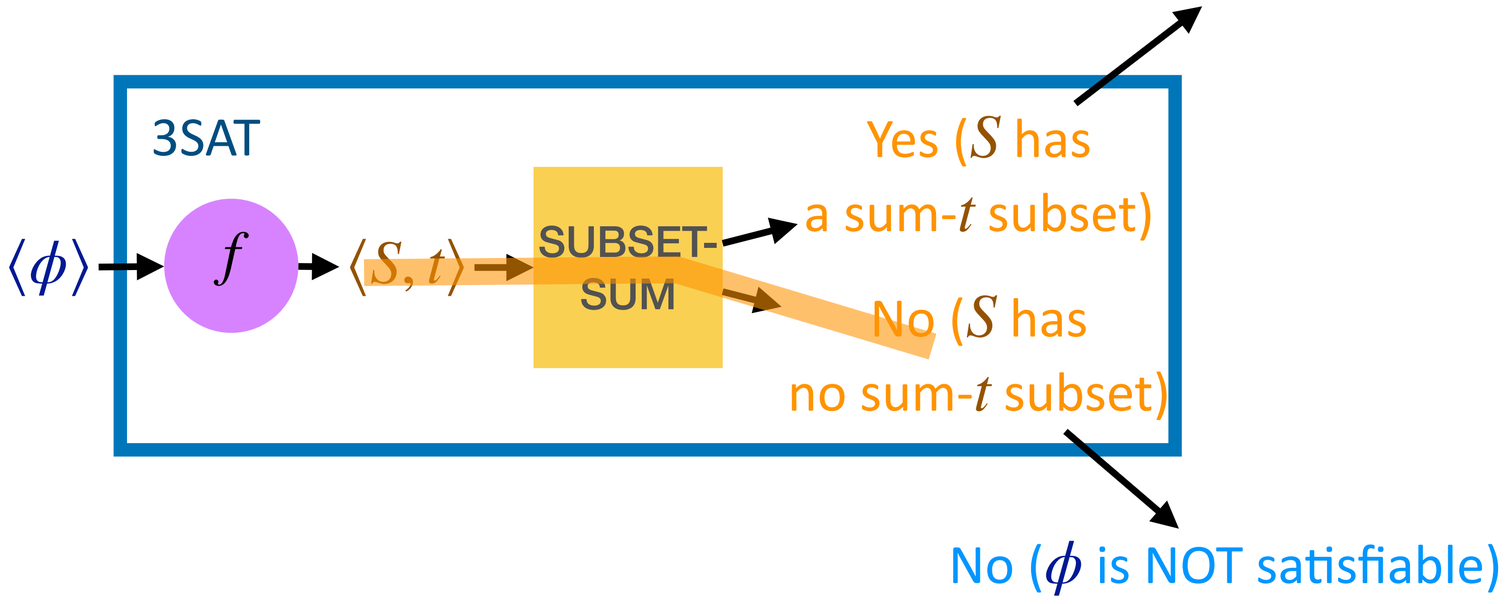



| | 1 | 2 | 3 | $\ell$+ 1 | $\ell$+ 2 | $\ell$+ 3 | $\ell$+ 4 | |
|---|---|---|---|---|---|---|---|---|
| $y_1 =$ | 1 | 0 | 0 | 1 | 1 | 0 | 1 | from variables |
| $z_1 =$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $y_2 =$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| $z_2 =$ | 0 | 1 | 0 | 0 | 0 | 1 | 1 | |
| $y_3 =$ | 0 | 0 | 1 | 1 | 1 | 0 | 1 | |
| $z_3 =$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| $g_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | from clauses |
| $h_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |
| $g_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $h_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $g_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $h_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $g_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $h_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $t =$ | 1 | 1 | 1 | 3 | 3 | 3 | 3 | |

If $\phi$ is satisfiable, there exists an assignment ($x_1 = T$, $x_2 = F$, $\cdots$) such that every clause has at least one $T$.
If $x_i = T$, choose $y_i$ as part of the subset. Otherwise, choose $z_i$. Further select enough of the $g_j$ and $h_j$ numbers to bring each of the last $k$ decimals up to 3.

# SUBSET-SUM

$$\overset{T}{(x_1 \vee x_2 \vee x_3)} \wedge \overset{F}{(x_1 \vee \bar{x}_1 \vee x_3)} \wedge \overset{F}{(x_2 \vee \bar{x}_2 \vee \bar{x}_3)} \wedge (x_1 \vee x_1 \vee \bar{x}_2)$$




| | | 1 | 2 | 3 | $\ell+1$ | $\ell+2$ | $\ell+3$ | $\ell+4$ | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 1 | 2 | 3 | 4 | |
| $y_1$ | = | 1 | 0 | 0 | 1 | 1 | 0 | 1 | from variables |
| $z_1$ | = | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $y_2$ | = | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| $z_2$ | = | 0 | 1 | 0 | 0 | 0 | 1 | 1 | |
| $y_3$ | = | 0 | 0 | 1 | 1 | 1 | 0 | 1 | |
| $z_3$ | = | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| $g_1$ | = | 0 | 0 | 0 | 1 | 0 | 0 | 0 | from clauses |
| $h_1$ | = | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |
| $g_2$ | = | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $h_2$ | = | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $g_3$ | = | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $h_3$ | = | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $g_4$ | = | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $h_4$ | = | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $t$ | = | 1 | 1 | 1 | 3 | 3 | 3 | 3 | |

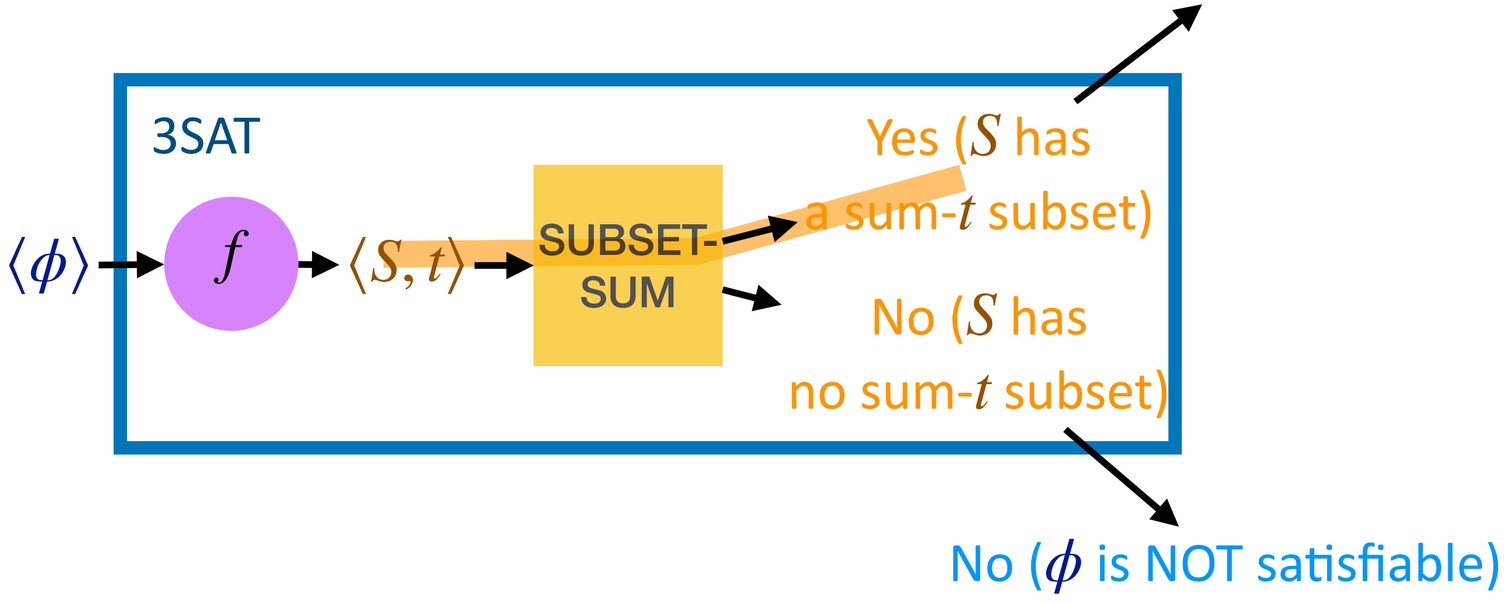
Since the assignment is feasible, each $x_i$ is either $T$ or $F$

$\Rightarrow$ for any $i$, either $y_i$ or $z_i$ is chosen

$\Rightarrow$ for each of the first $\ell$ decimals, the sum is $1$

# SUBSET-SUM

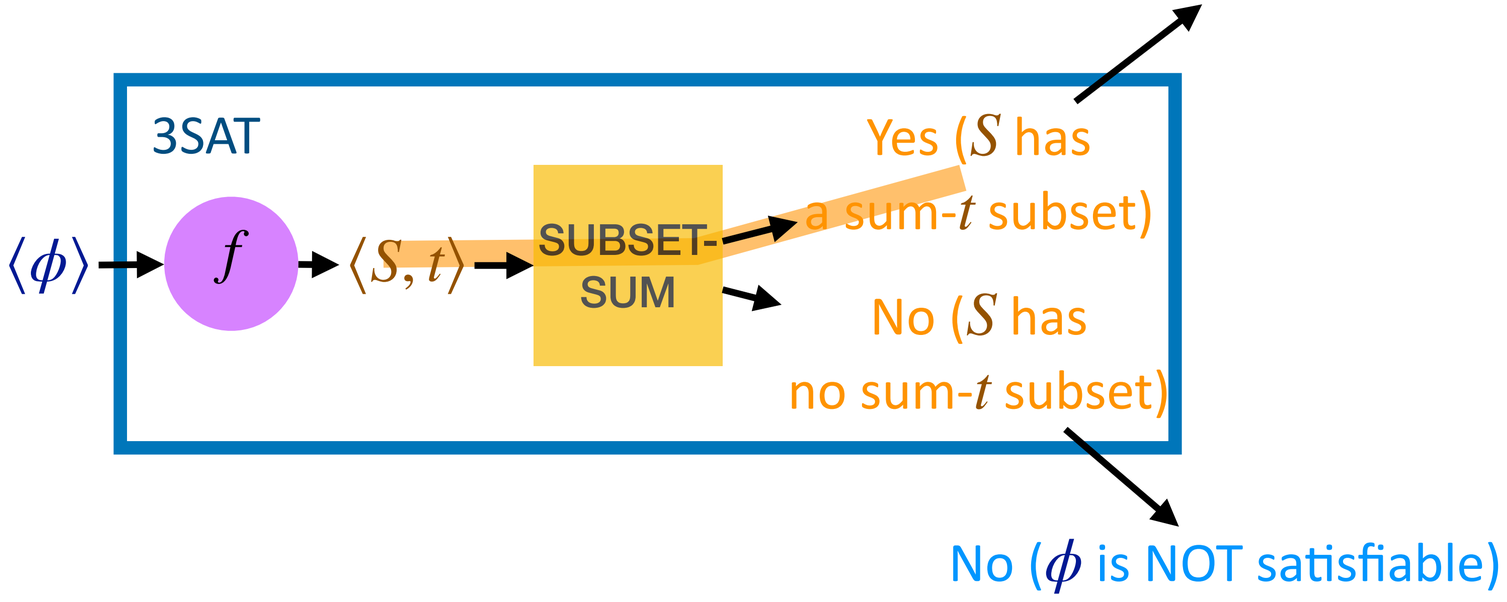
$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_1 \vee x_3) \wedge (x_2 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_1 \vee \bar{x}_2)$




| | 1 | 2 | 3 | $\ell+1$ | $\ell+2$ | $\ell+3$ | $\ell+4$ | |
|---|---|---|---|---|---|---|---|---|
| $y_1 =$ | 1 | 0 | 0 | 1 | 1 | 0 | 1 | from variables |
| $z_1 =$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $y_2 =$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| $z_2 =$ | 0 | 1 | 0 | 0 | 0 | 1 | 1 | |
| $y_3 =$ | 0 | 0 | 1 | 1 | 1 | 0 | 1 | |
| $z_3 =$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| $g_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | from clauses |
| $h_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |
| $g_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $h_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $g_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $h_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $g_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $h_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $t =$ | 1 | 1 | 1 | 3 | 3 | 3 | 3 | |

# SUBSET-SUM

$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_1 \vee x_3) \wedge (x_2 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_1 \vee \bar{x}_2)$

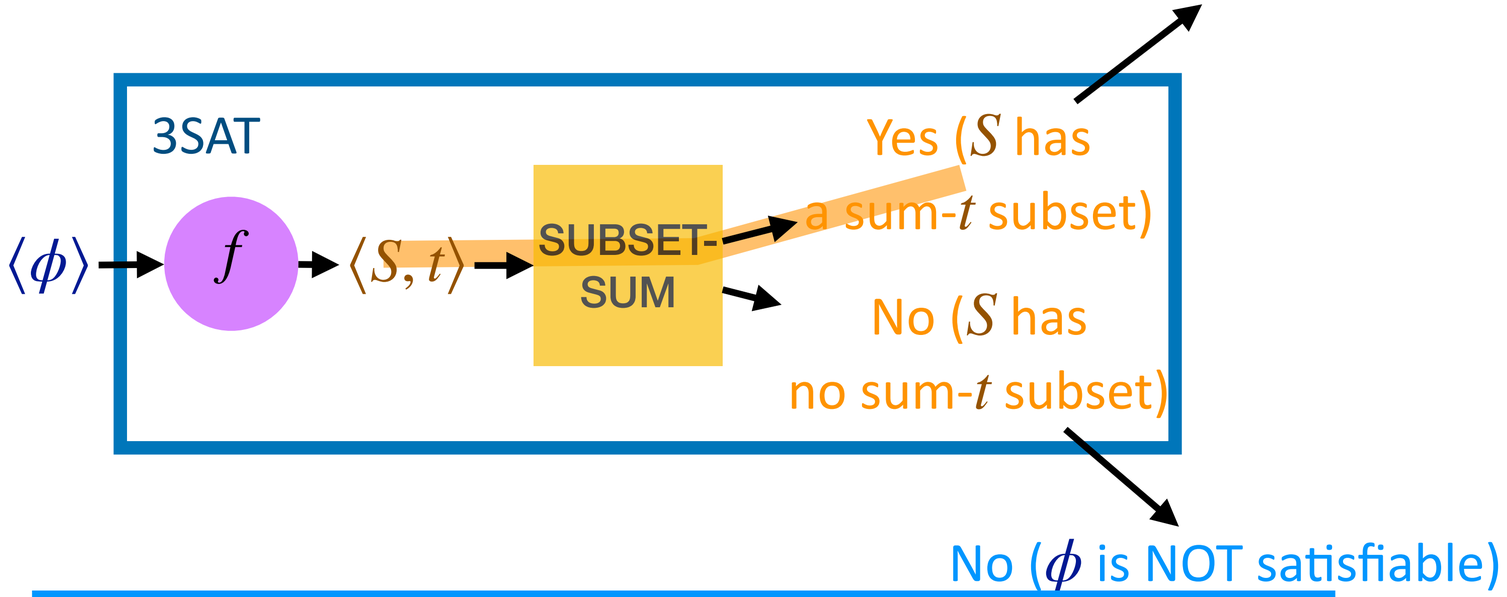



| | 1 | 2 | 3 | $\ell$+ 1 | $\ell$+ 2 | $\ell$+ 3 | $\ell$+ 4 | |
|---|---|---|---|---|---|---|---|---|
| $y_1 =$ | 1 | 0 | 0 | 1 | 1 | 0 | 1 | from variables |
| $z_1 =$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $y_2 =$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| $z_2 =$ | 0 | 1 | 0 | 0 | 0 | 1 | 1 | |
| $y_3 =$ | 0 | 0 | 1 | 1 | 1 | 0 | 1 | |
| $z_3 =$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| $g_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | from clauses |
| $h_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |
| $g_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $h_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $g_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $h_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $g_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $h_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $t =$ | 1 | 1 | 1 | 3 | 3 | 3 | 3 | |

For any $j$, $g_j$ and $h_j$ can contribute at most two, so at least one $1$ come from some $y_i$ or $z_i$. Therefore, every clause has at least one *true* literal (that is, the $y_i$ or $z_i$).

# SUBSET-SUM

$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x_1} \vee x_3) \wedge (x_2 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_1 \vee x_1 \vee \bar{x_2})$




| | 1 | 2 | 3 | $\ell+1$ | $\ell+2$ | $\ell+3$ | $\ell+4$ | |
|---|---|---|---|---|---|---|---|---|
| $y_1 =$ | 1 | 0 | 0 | 1 | 1 | 0 | 1 | from variables |
| $z_1 =$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $y_2 =$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| $z_2 =$ | 0 | 1 | 0 | 0 | 0 | 1 | 1 | |
| $y_3 =$ | 0 | 0 | 1 | 1 | 1 | 0 | 1 | |
| $z_3 =$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| $g_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | from clauses |
| $h_1 =$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |
| $g_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $h_2 =$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $g_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $h_3 =$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| $g_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $h_4 =$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| $t =$ | 1 | 1 | 1 | 3 | 3 | 3 | 3 | |

Suppose that there is a subset of $S$ sums to $t$. We construct a satisfying assignment to $\phi$: if the subset contains $y_i$, we assign $x_i = T$; otherwise, we assign it $F$. Since exactly one among $y_i$ and $z_i$ can be chosen, the assignment is feasible. For any $j$, $g_j$ and $h_j$ can contribute at most two, so at least one $1$ come from some $y_i$ or $z_i$. Therefore, every clause has at least one *true* literal

# SUBSET-SUM

- Theorem: SUBSET-SUM is NP-Hard

SUBSET-SUM $= \{\langle S, t\rangle \mid S = \{x_1, x_2, \cdots, x_n\}$ and for some $\{y_1, \cdots, y_m\} \subseteq S$, we have $\Sigma y_i = t\}$

<proof idea> We prove that all languages in NP are polynomial time reducible to SUBSET-SUM by reducing the NP-complete language 3SAT to it. Given a 3cnf-formula $\phi$ we construct an instance of the SUBSET-SUM problem that contains a subcollection summing to the target $k$ if and only if $\phi$ is satisfiable.

# SUBSET-SUM

- Theorem: SUBSET-SUM is NP-Hard

<proof> To prove the NP-hardness of SUBSET-SUM, it is sufficient to reduce the NP-complete problem 3SAT to it. Given a 3cnf-formula $\phi$ with variables $x_1, \cdots, x_l$ and clauses $c_1, \cdots, c_k$, we construct an instance of the SUBSET-SUM problem, $\langle S, t \rangle$, contains large numbers with $l + k$ decimals. For each variable $x_i$ in $\phi$, there are two numbers $y_i, z_i$ in $S$.

- The i-th decimal of $y_i$ is 1, and all the decimals in the first $l$ decimals of $y_i$ are 0. The $(l + j)$-th decimal of $y_i$ is 1 if and only if the clause $c_j$ contains literal $x_i$.
- The i-th decimal of $z_i$ is 1, and all the decimals in the first $l$ decimals of $y_i$ are 0. The $(l + j)$-th decimal of $z_i$ is 1 if and only if the clause $c_j$ contains literal $\overline{x_i}$.

# SUBSET-SUM

Additionally, $S$ contains one pair of numbers $g_j, h_j$ for each clause $c_j$, These two numbers are equal and consists of single 1 at the $(l + j)$-th decimal and all other decimals are 0s.

Finally, the target number $t$ consists of $l$ 1s and followed by $k$ 3s.

The construction for each number in $S$ takes $O(k(l + k))$ time since every decimal needs at most $3k$ time to check. There are $2l + 2k$ numbers, so the total construction time is $O((l + k)^3)$ times which is polynomial in the size of $\langle \phi \rangle$.

# SUBSET-SUM

Now we show why this constructions works by demonstrating that $\phi$ is satisfiable if and only if some subset of $S$ sum to $t$.

Suppose $\phi$ is satisfiable. We construct a subset of $S$ as follows. We select $y_i$ if $x_i$ is assigned *true* in the satisfying assignment and $z_i$ if $x_i$ is assigned *false*. For each of the first $l$ decimals, the sum is exactly 1 since the assignment is legal. Furthermore, each of the last $k$ decimals is between 1 to 3 because each of the 3-literal clauses has at least one *true* literal. By selecting enough of the $g$ and $h$ numbers to bring each of the last $k$ decimals up to 3, the large target is hit.

# SUBSET-SUM

Suppose that a subset of $S$ sums to $t$. We construct a satisfying assignment to $\phi$. First we observe that no carry into the next decimal is needed since all the decimal in members of $S$ are either 0 or 1 and each decimal altogether contains at most five 1s. Hence, to get a 1 in each of the first $l$ decimals, the subset must have either $y_i$ or $z_i$ for each $i$, but not both.

Now we make the satisfying assignment. If the subset contains $y_i$, we assign $x_i$ *true*; otherwise, we assign it *false*. Since in each of the final $k$ decimals the sum is always 3 and there are at most two 1s come from $g_i$ or $h_i$, there is at least one 1 come from some $y_i$ or $z_i$. Hence this assignment satisfies $\phi$.

# What Happened

- 3SAT $\leq_p$ SUBSET-SUM
  - There may be some clause in the CNF-SAT instance $\phi$ that has fewer than 3 literals
    - $\Rightarrow$ Duplicate existed literal
  - There may be some clause in $\phi$ that has more than 3 literals
    - $\Rightarrow$ make a chain of 3-clauses in $\phi'$, using dummy variables

- Each clause in $\phi$ is *TRUE* if and only if the corresponding (chain of) clauses in $\phi'$ are all *TRUE*

# ~~Tattoo this on your arm~~

- If you want to prove some problem $Q$ is NP-hard, reduce some NP-complete problem $Q'$ to $Q$.