Algorithms for Decision Support

NP-Completeness (2/3)

NP-Completeness

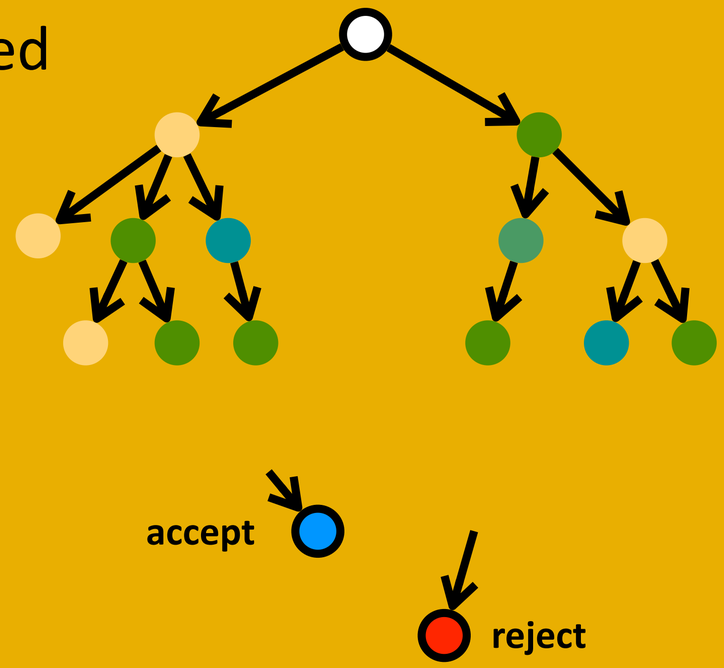
Turing machine

- 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



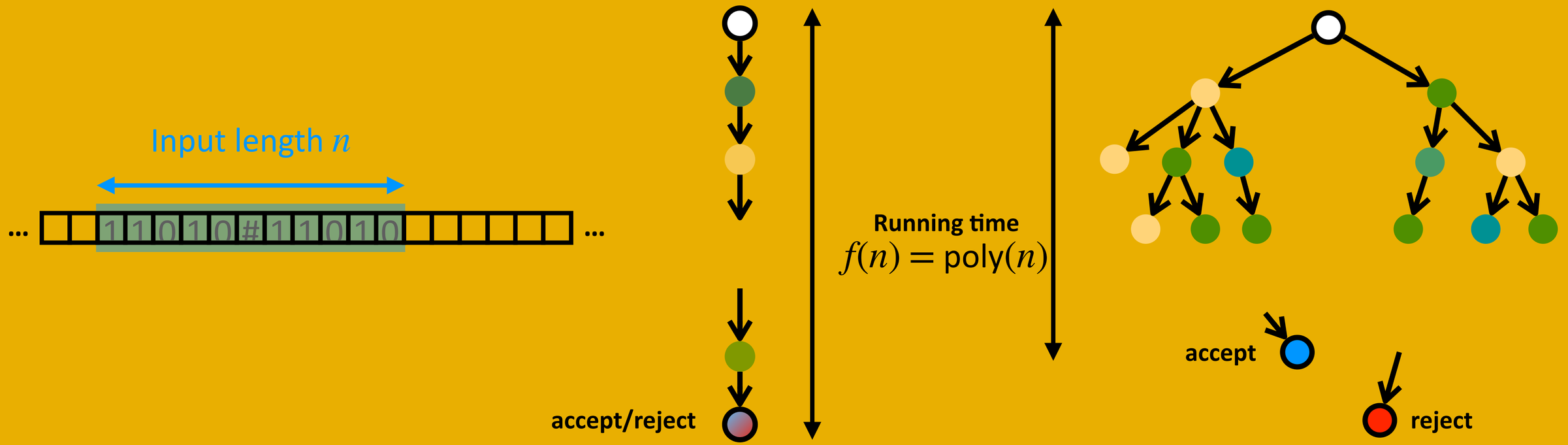
Formal Language Framework

- 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

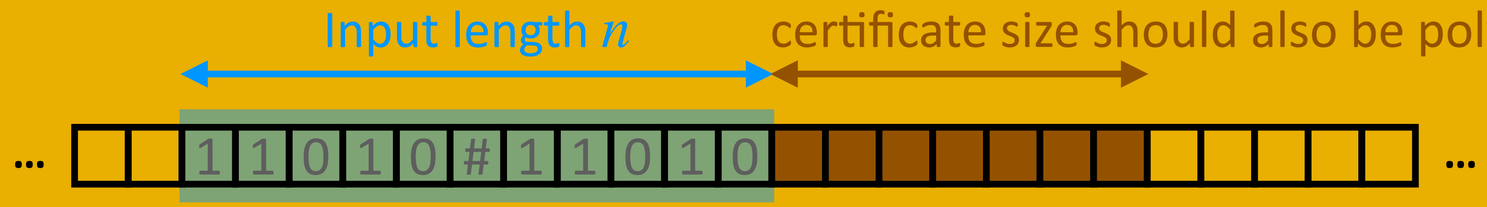
Class **P** and Class **NP**

- 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 in polynomial time by a *non-deterministic* Turing machine.



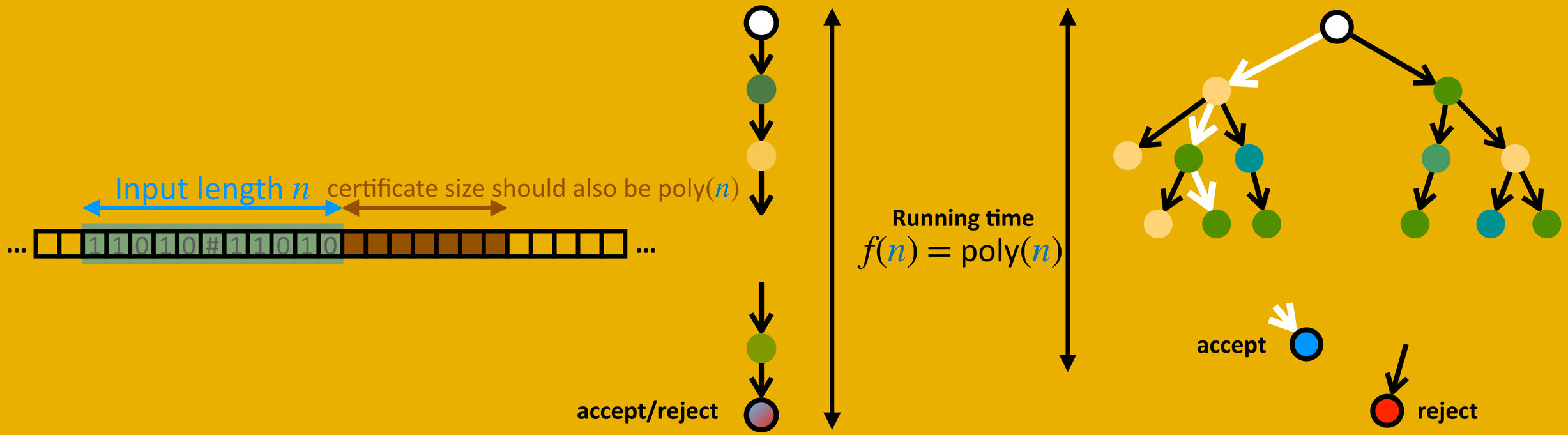
Certificate and (Polynomial-time) Verify

- 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)

Outline

- NP-Completeness
  - NP-hardness: Polynomial time reduction
    - CNF-SAT $\leq_p$ 3SAT
    - 3SAT $\leq_p$ SUBSET-SUM
    - 3SAT $\leq_p$ CLIQUE
    - PARTITION $\leq_p$ BIN-PACKING
- Cook-Leven Theorem: SAT is NP-complete

Cook-Levin Theorem

Boolean Formula

- Boolean formula: an expression involving Boolean variables and operations
  - Example:
    - $\phi = \bar{x} \wedge y \wedge z$ $x =$ FALSE, $y =$ TRUE, $z =$ TRUE  yes-instance
    - $\phi = (\bar{x} \wedge y) \vee (x \wedge \bar{z})$  no-instance
  - (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 = $\{\langle\phi\rangle \mid \phi$ is a satisfiable Boolean formula$\}$

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**

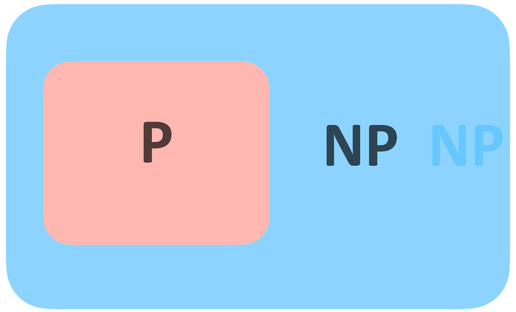
NP-Complete

NP-Complete

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

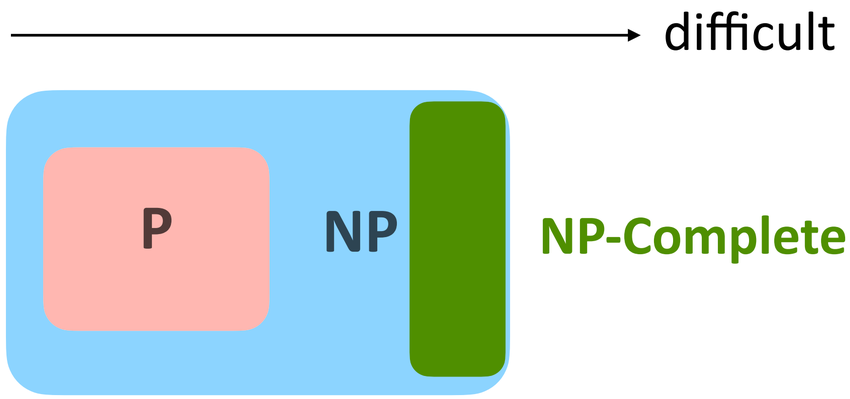
NP-Complete

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



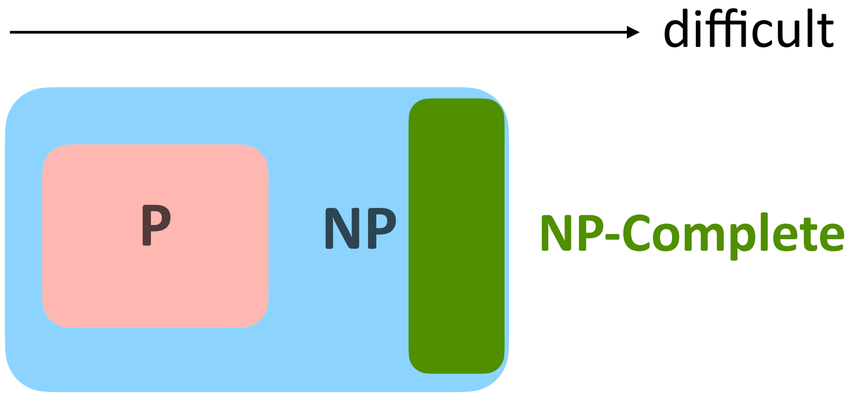
NP-Complete

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



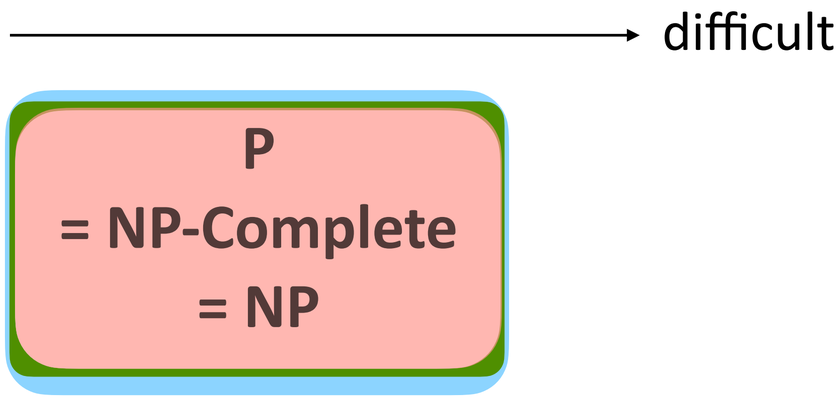
NP-Complete

- 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



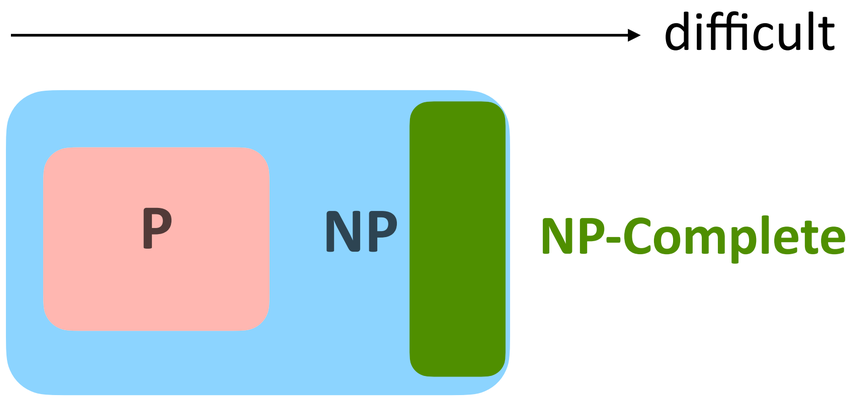
NP-Complete

- 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



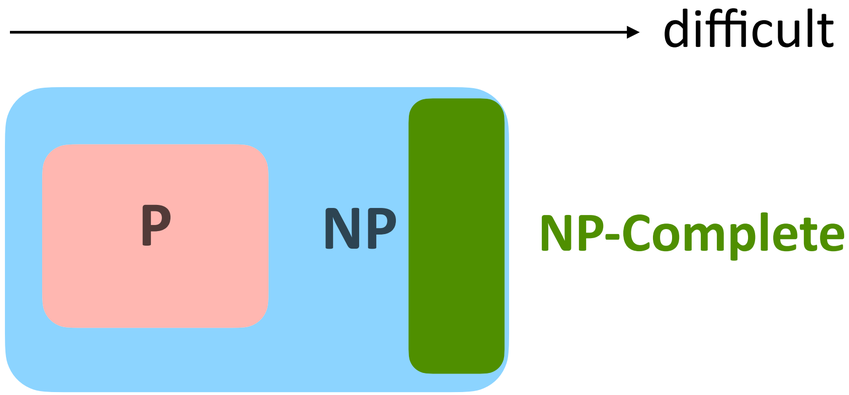
NP-Complete

- 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



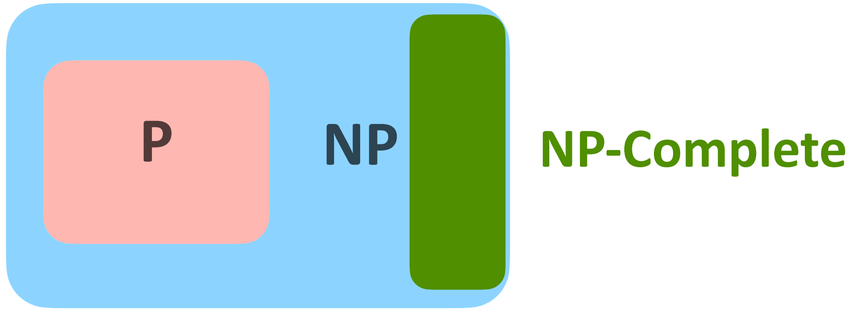
NP-Complete

- 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”?

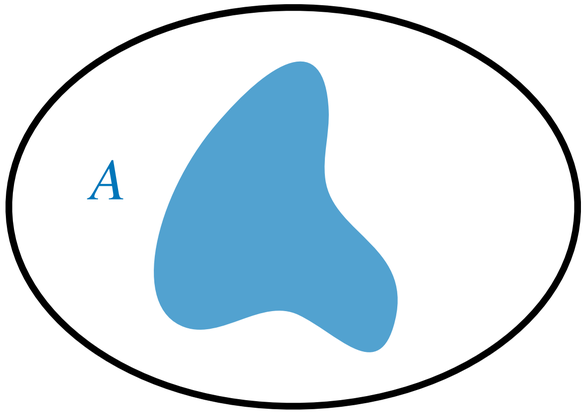
How do we know if a problem is “difficult”?

Reduction

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

Reduction

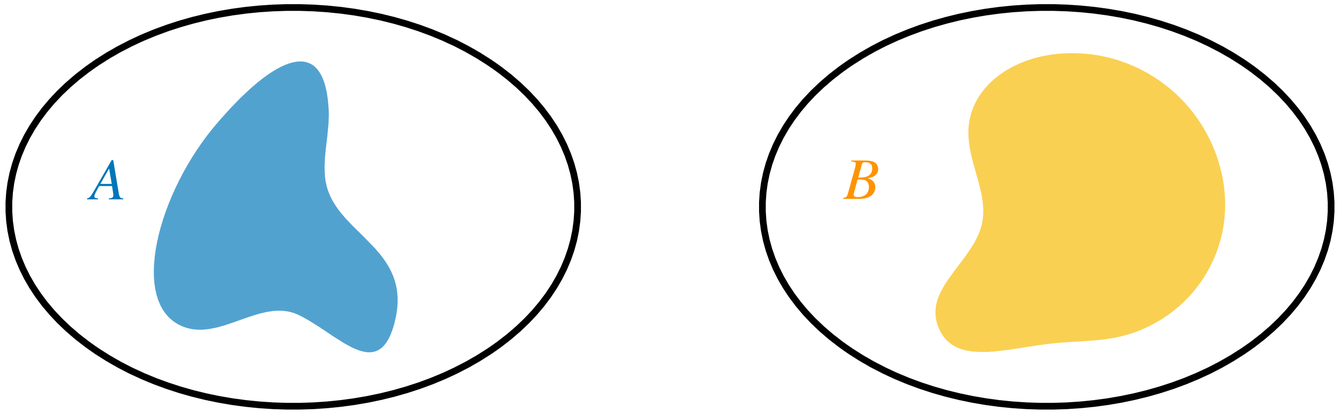
- We want to solve problem A. Instead of solving A directly, we can show that **we are able to solve** A by **using an (existed) algorithm for solving another problem** B.



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

Reduction

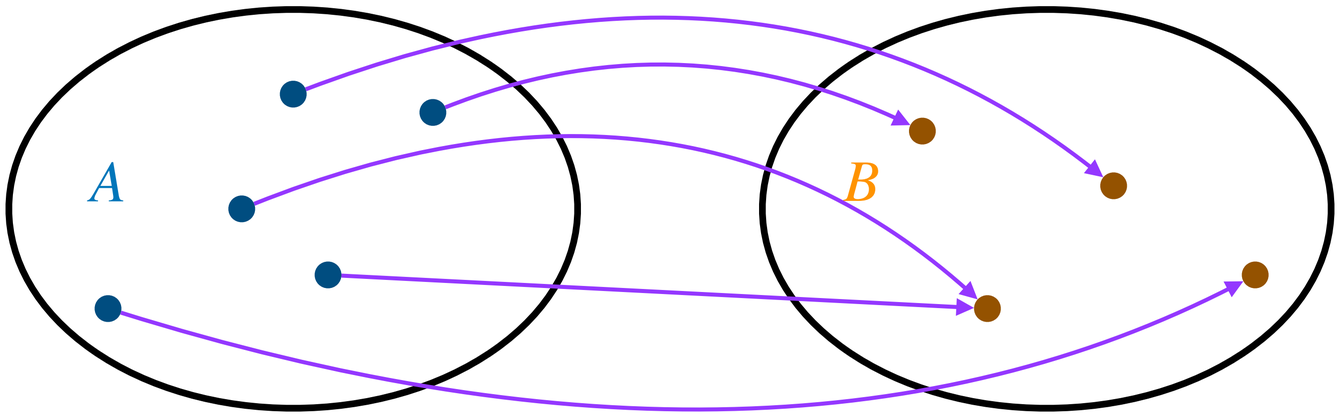
- We want to solve problem A. Instead of solving A directly, we can show that **we are able to solve** A by **using an (existed) algorithm for solving another problem** B. According to the answer to problem B, we know the answer to problem A.



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

Reduction

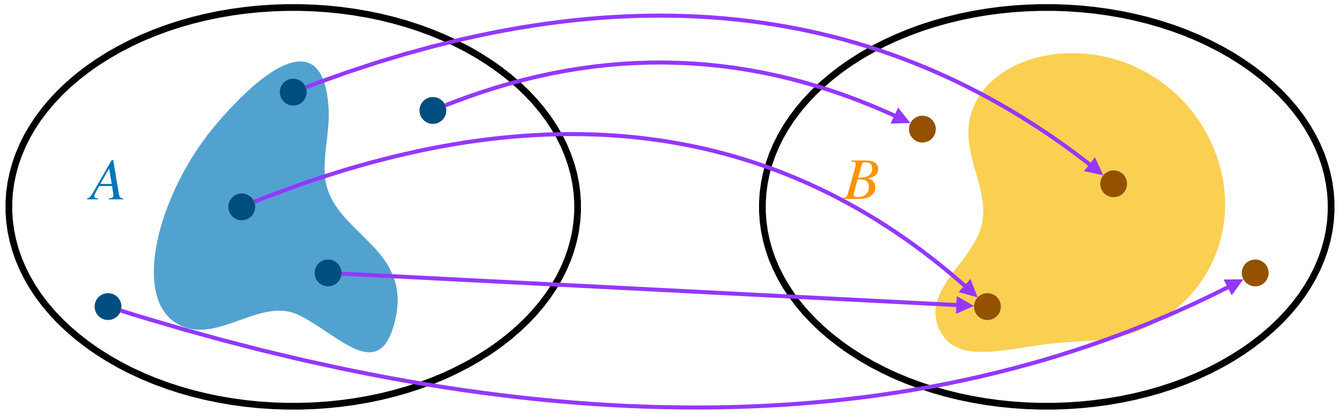
- We want to solve problem A. Instead of solving A directly, we can show that **we are able to solve** A by **using an (existed) algorithm for solving another problem** B. According to the answer to problem B, we know the answer to problem A.



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”?

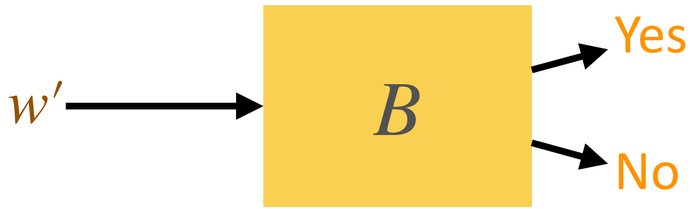
Reduction

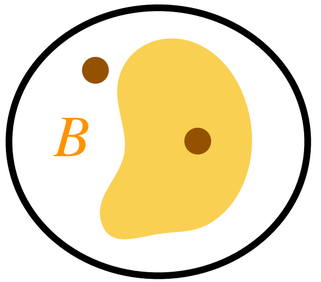
- We want to solve problem A
  - That is, given any instance w, we want to answer yes if $w \in A$ and answer no otherwise
  - Instead of solving A directly, we can show that **we are able to solve** A by **using an (existed) algorithm for solving another problem** B

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 $w' \notin B$

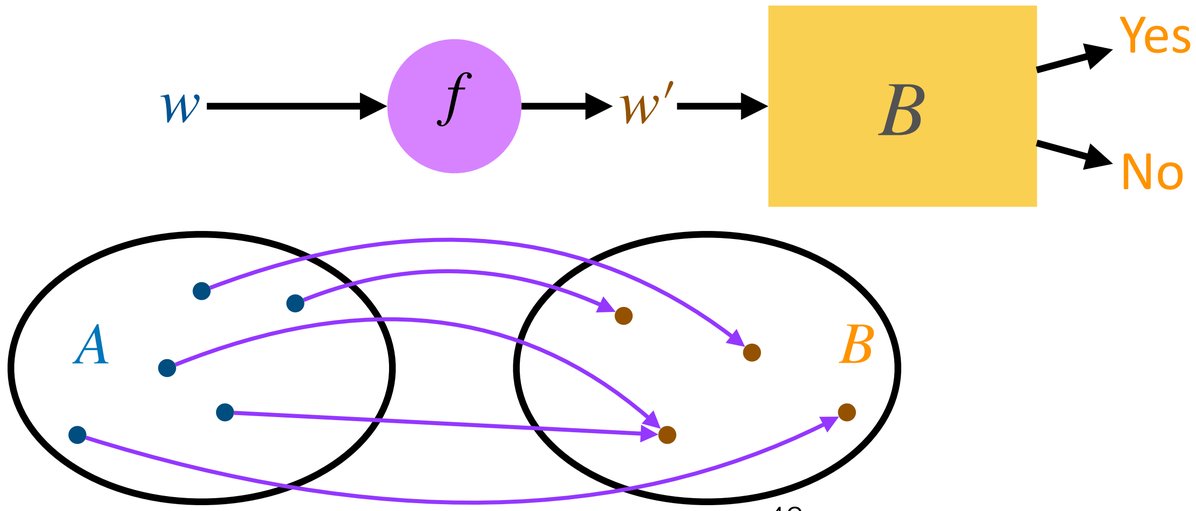
Reduction

- 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

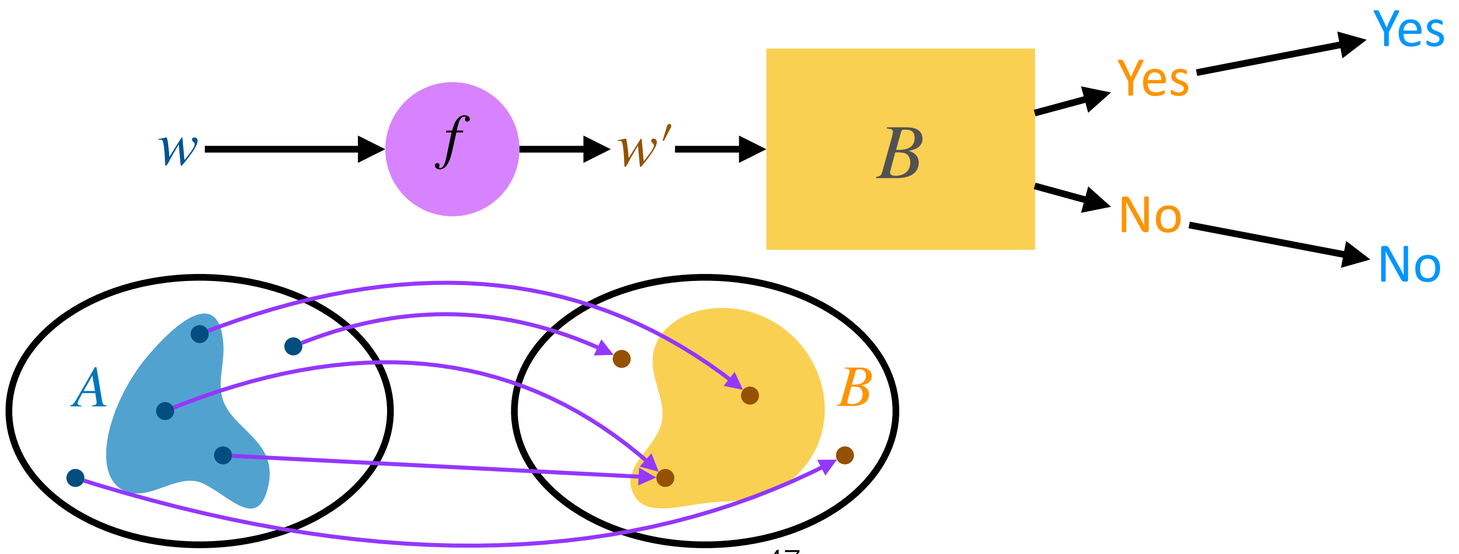
Reduction

- 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 $w' \notin B$



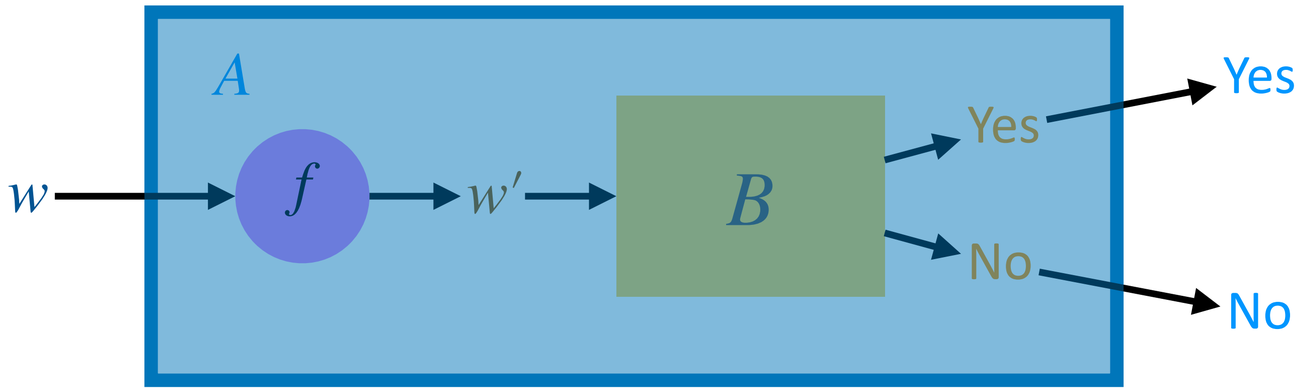
Reduction

- 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 $w' \notin B$



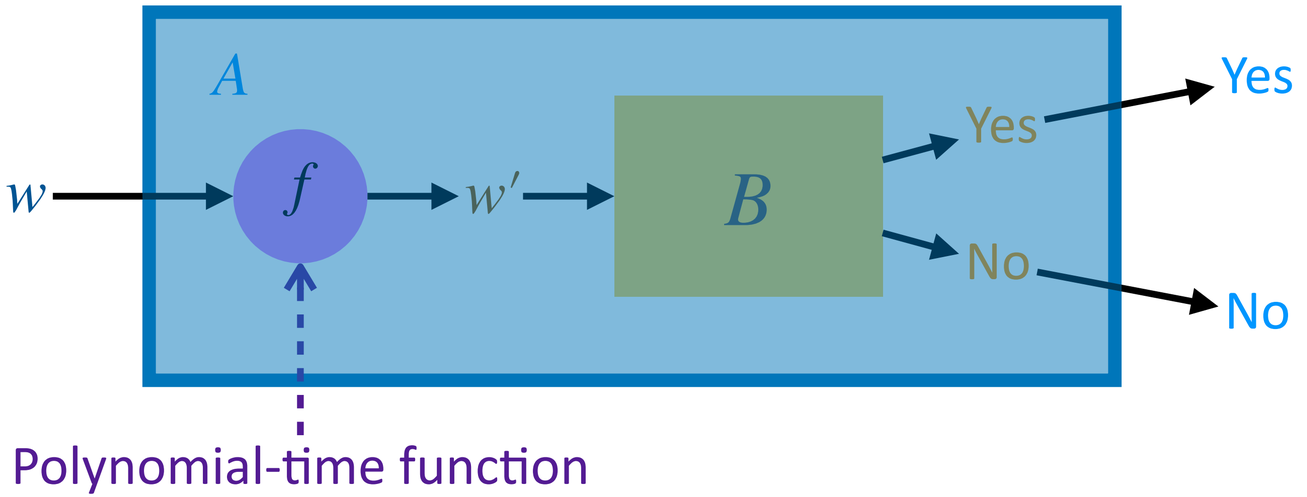
Reduction

- 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 $A \leq_p B$

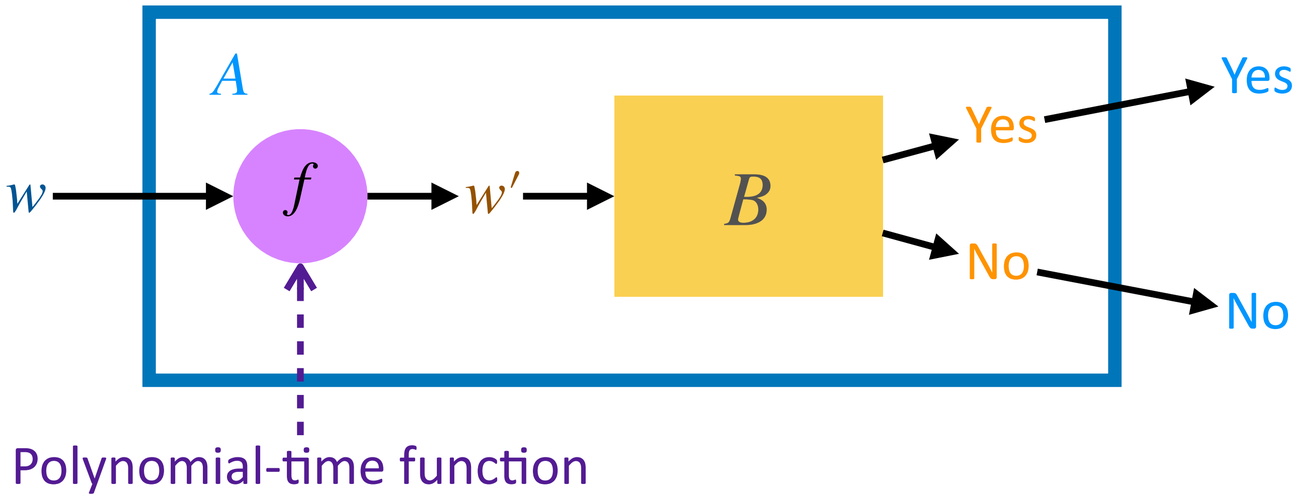
- Problem A with input w
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem B with input w'
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$



Polynomial-Time Reduction $A \leq_p B$

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

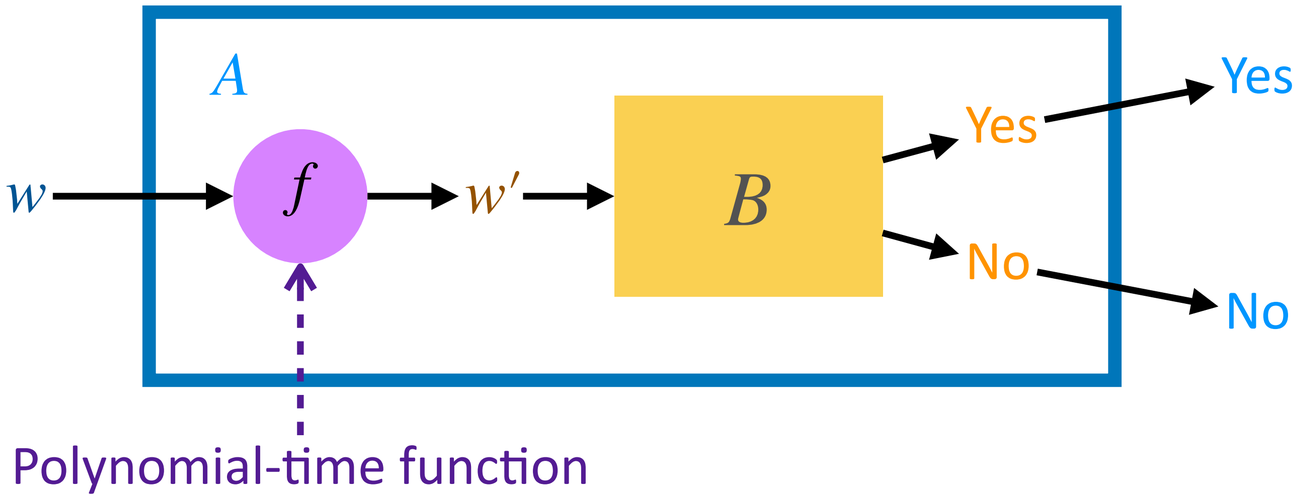
- The sun rises in the east on day w
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$



Polynomial-Time Reduction $A \leq_p B$

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

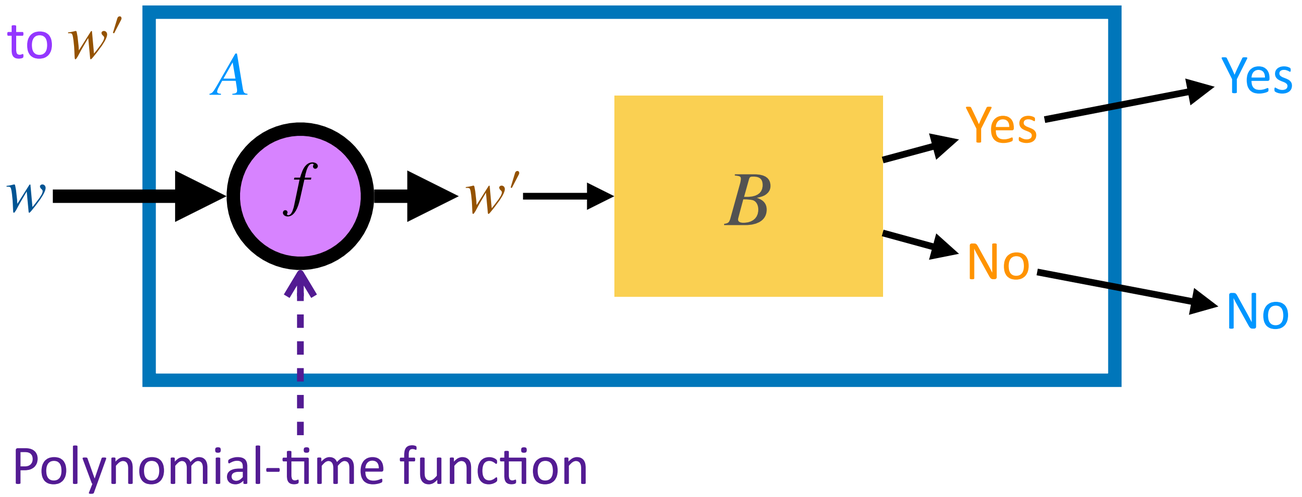
- The sun rises in the east on day w
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$



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 $w' \notin B$

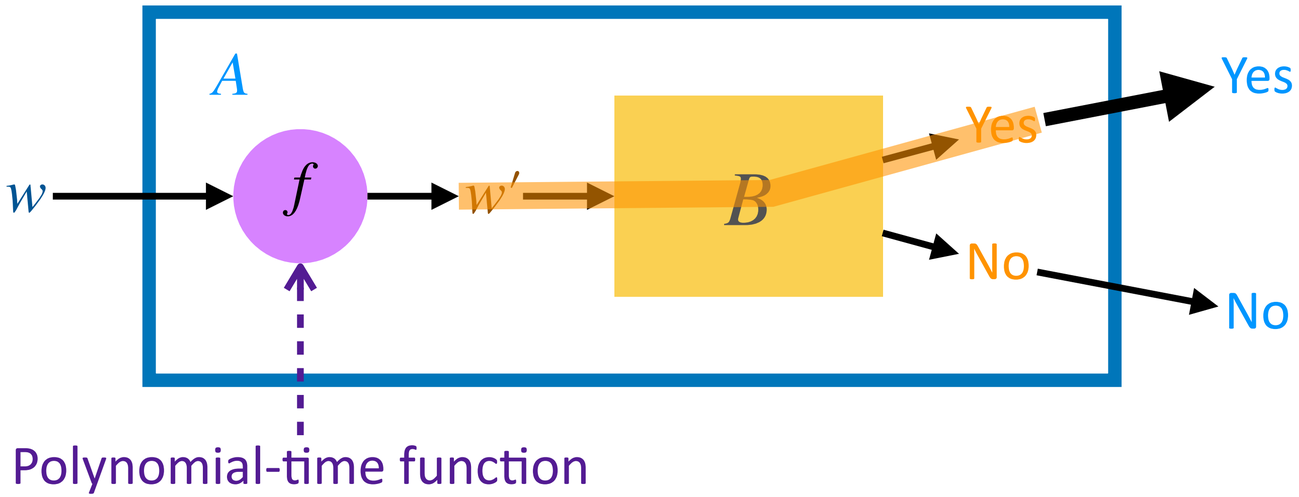
1. Show that there is a function that transforms every w to w' in polynomial time



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 $w' \notin B$

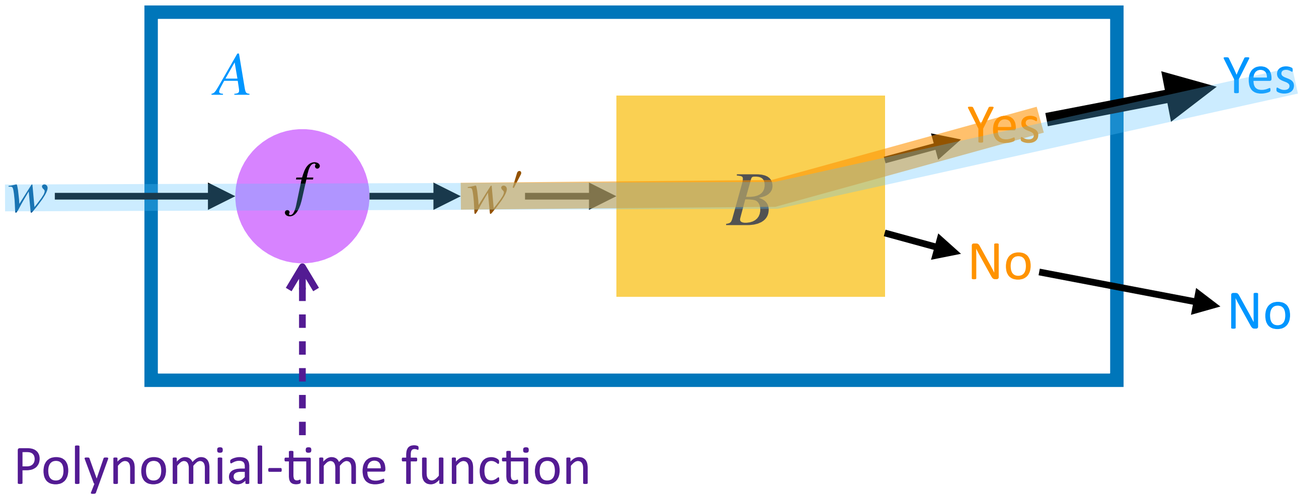
2. Show that for any yes-instance $w' \in B$,



Polynomial-Time Reduction $A \leq_p B$

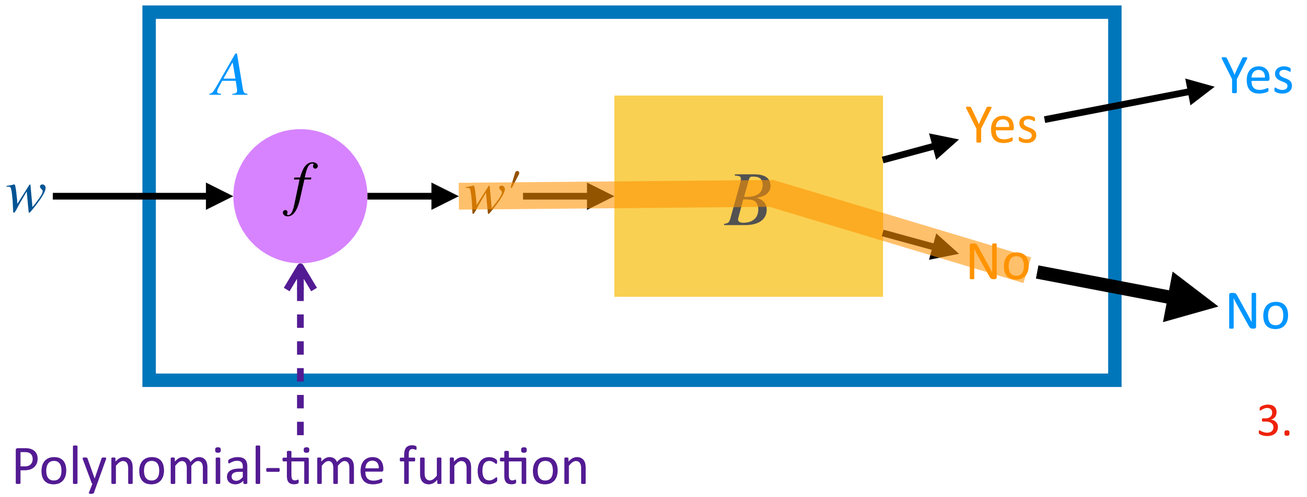
- Problem A with input w
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem B with input w'
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$

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



Polynomial-Time Reduction $A \leq_p B$

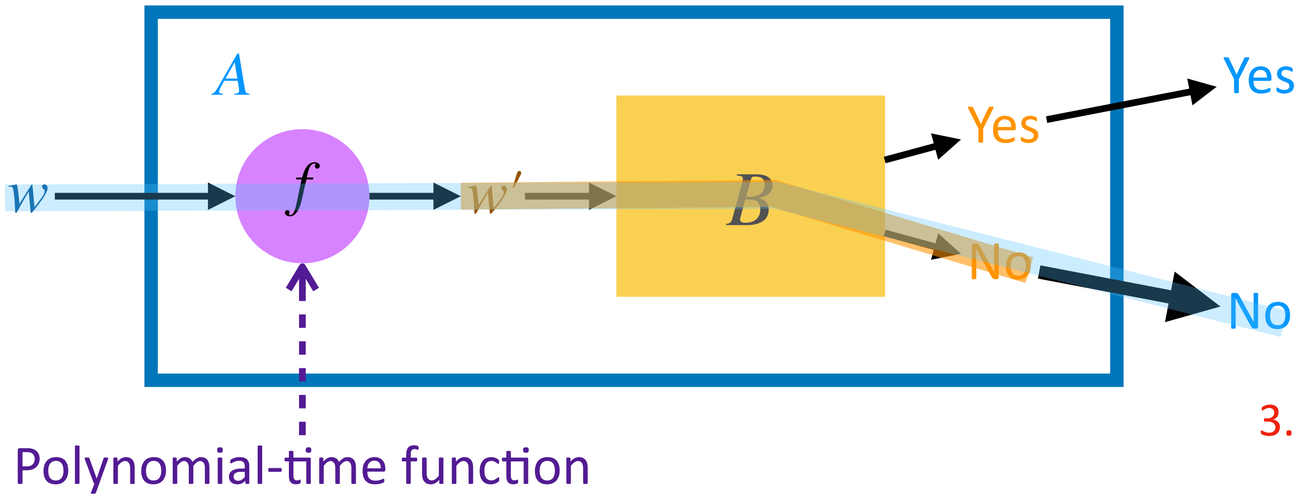
- Problem A with input w
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem B with input w'
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$



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

Polynomial-Time Reduction $A \leq_p B$

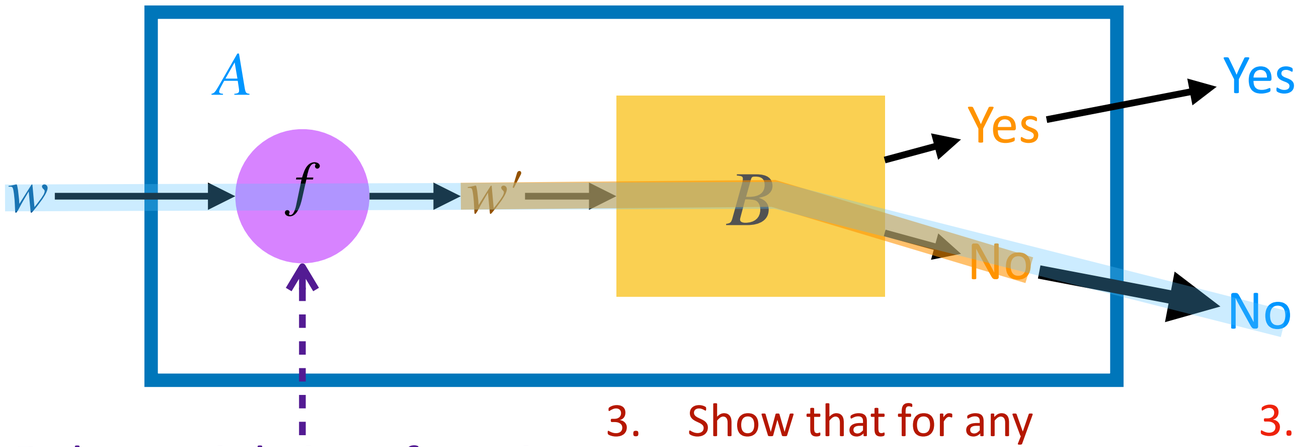
- Problem A with input w
  - Return yes if $w \in A$
  - Return no if $w \notin A$
- Problem B with input w'
  - Return yes if $w' \in B$
  - Return no if $w' \notin B$



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 $w' \notin B$



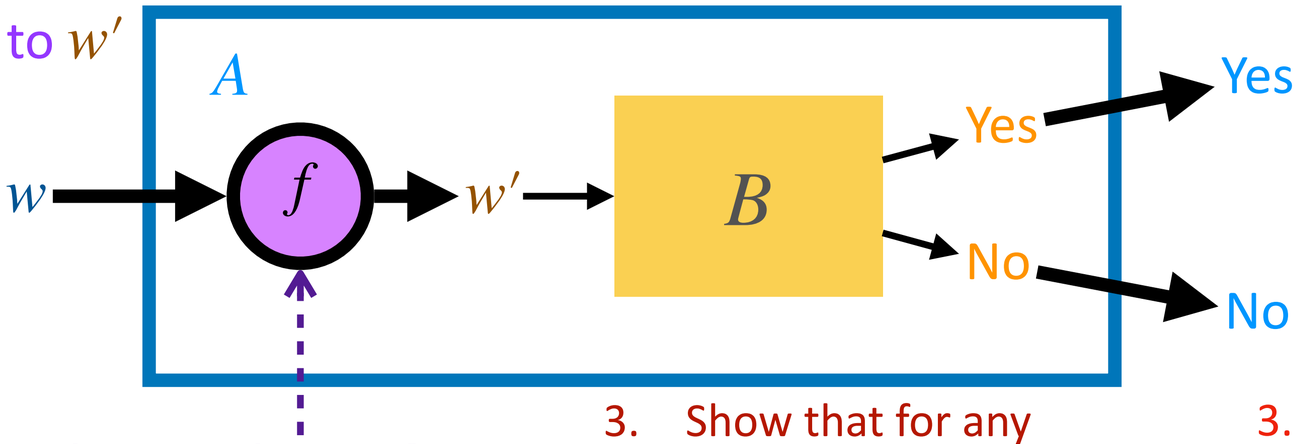
3. Show that for any yes-instance $w \in A$, the corresponding instance w' is also a yes-instance of B

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

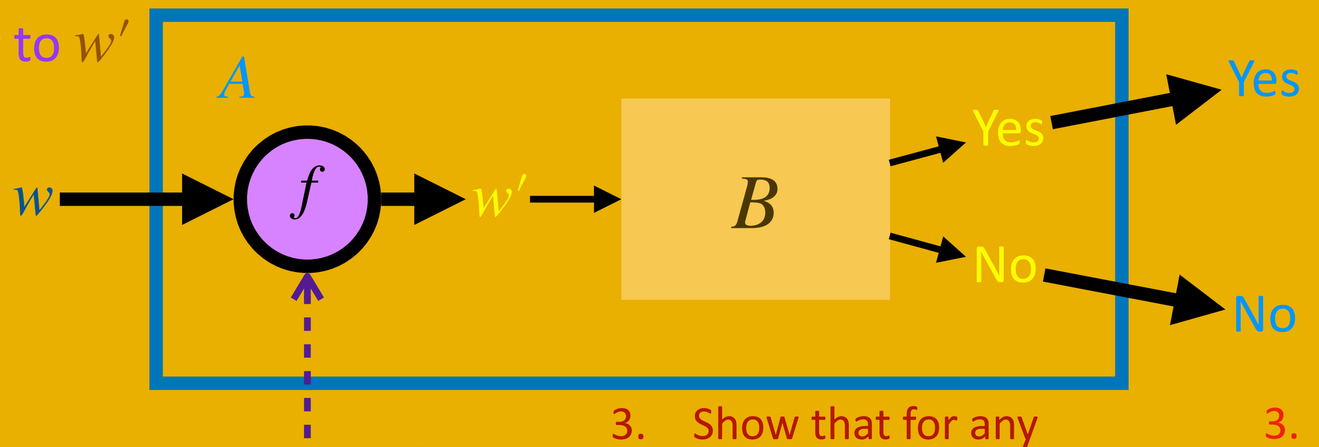
3. Show that for any yes-instance $w \in A$, the corresponding instance w' is also a yes-instance of B

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

Outline

- NP-Completeness
  - NP-hardness: Polynomial time reduction
    - CNF-SAT $\leq_p$ 3SAT
    - 3SAT $\leq_p$ SUBSET-SUM
    - 3SAT $\leq_p$ CLIQUE
    - PARTITION $\leq_p$ BIN-PACKING
- Cook-Leven Theorem: SAT is 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):

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

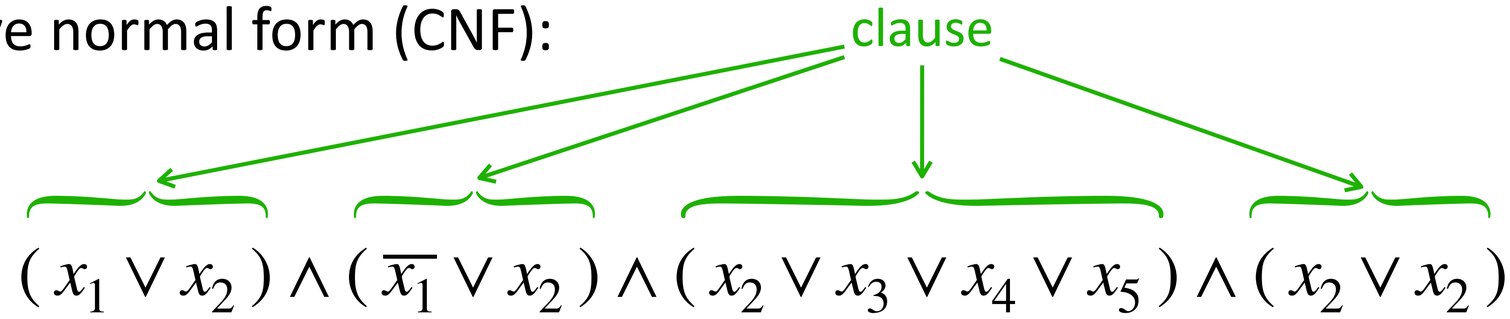
literals

  - 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

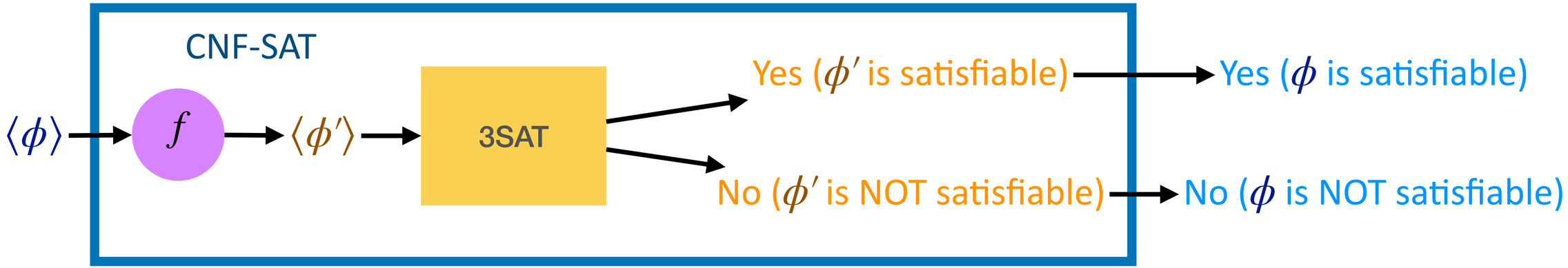
- A Boolean formula is a 3CNF-formula if it is in conjunctive normal form and **all the clauses have exactly three literals**.
  - Example: $(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_2 \vee x_2 \vee \overline{x_4})$

- CNF-SAT $\leq_p$ 3SAT

3SAT

- CNF-SAT $\leq_p$ 3SAT

<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

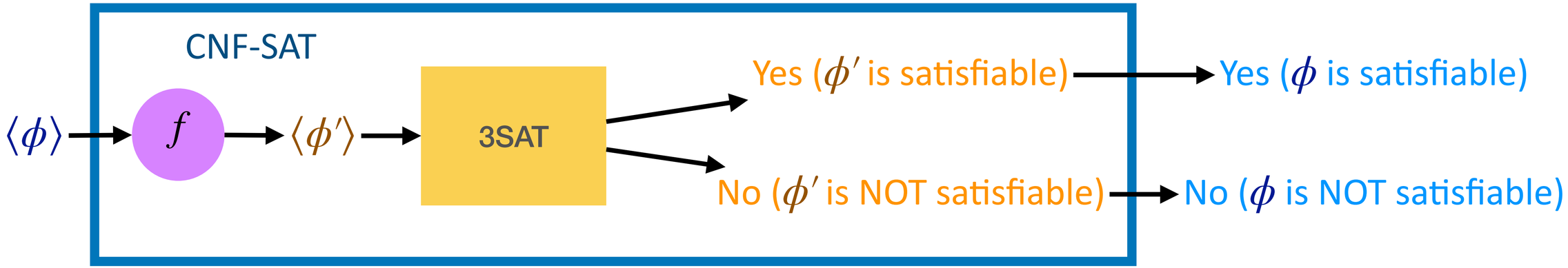
↓ Polynomial time function ⇕

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

3SAT

- CNF-SAT $\leq_p$ 3SAT

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



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

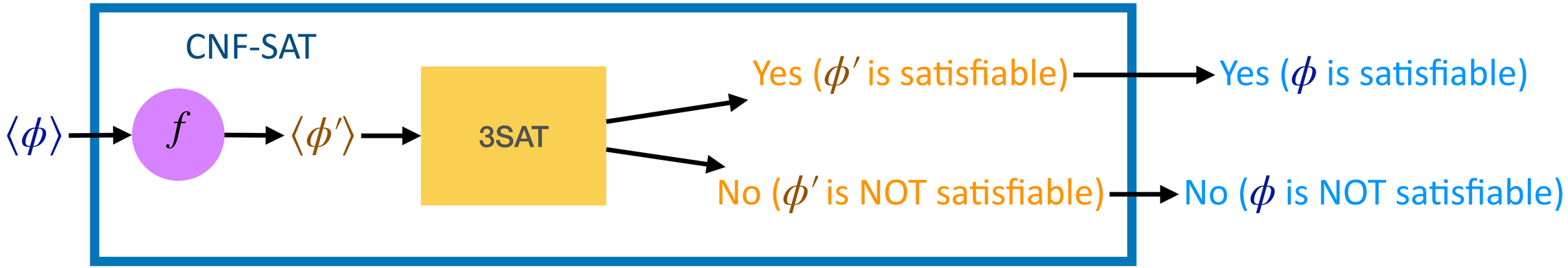
↓

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

3SAT

- CNF-SAT $\leq_p$ 3SAT

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



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

↓

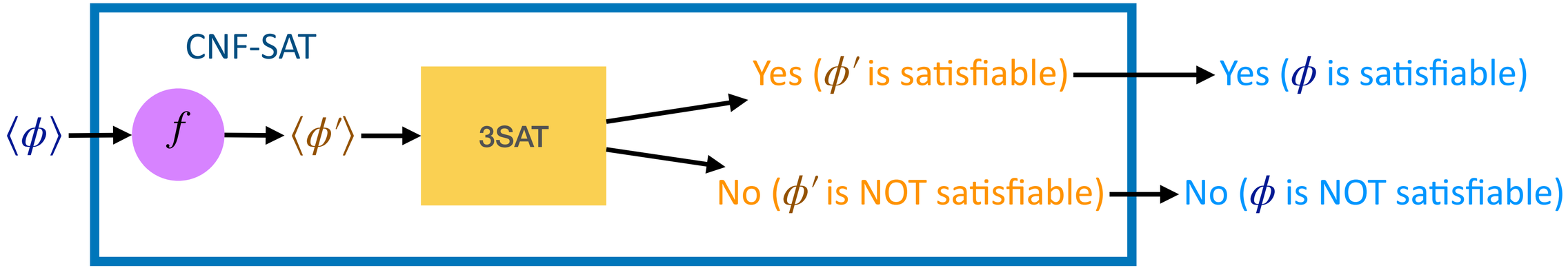
A 3-CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (? \vee ? \vee ?) \wedge (? \vee ? \vee ?) \wedge \cdots$

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

3SAT

- CNF-SAT $\leq_p$ 3SAT

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



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

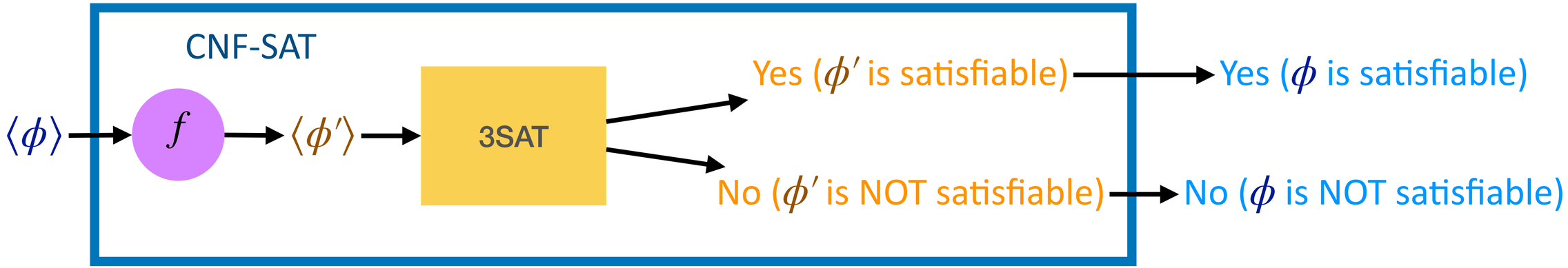
↓

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

3SAT

- CNF-SAT $\leq_p$ 3SAT

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



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

↓

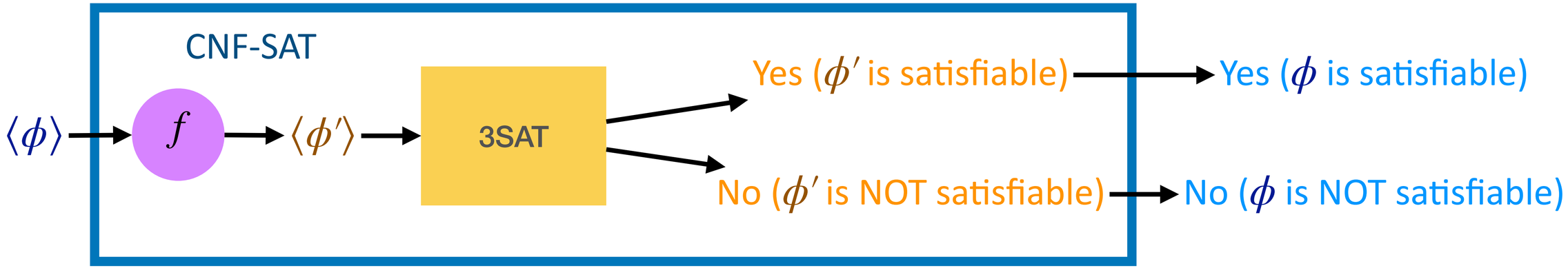
A 3-CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_1} \vee \overline{x_1}) \wedge (? \vee ? \vee ?) \wedge \cdots$

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

3SAT

- CNF-SAT $\leq_p$ 3SAT

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



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

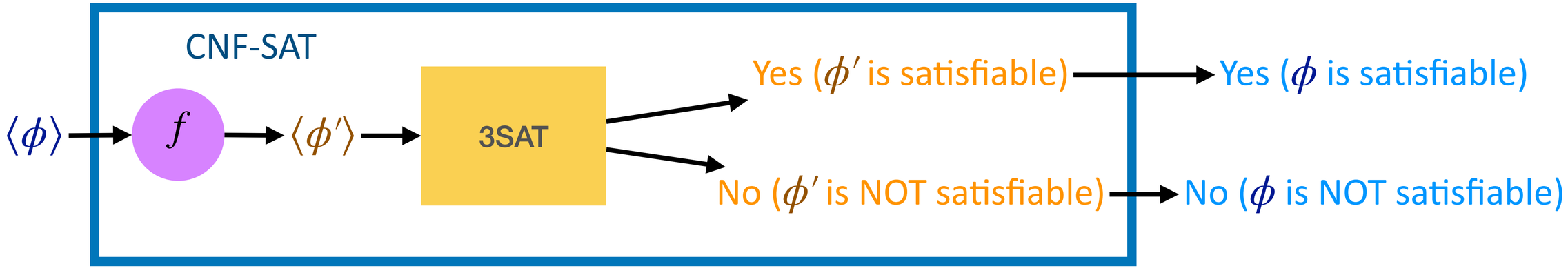
↓

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

- CNF-SAT $\leq_p$ 3SAT

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



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

↓

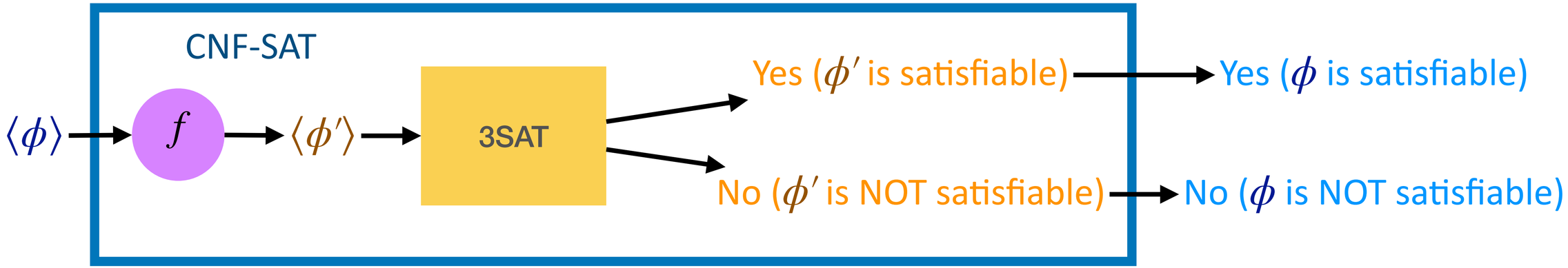
A 3-CNF Boolean formula $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_1} \vee \overline{x_1}) \wedge (? \vee ? \vee ?) \wedge (x_1 \vee x_2 \vee x_2)$

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

3SAT

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

1

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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

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

TRUE

3SAT

- CNF-SAT $\leq_p$ 3SAT

<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

- CNF-SAT $\leq_p$ 3SAT

<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

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

TRUE

3SAT

- CNF-SAT $\leq_p$ 3SAT

<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

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

TRUE

3SAT

- CNF-SAT $\leq_p$ 3SAT

<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

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

$1\ 0\ 1$

TRUE

3SAT

- CNF-SAT $\leq_p$ 3SAT

<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

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

TRUE

3SAT

- CNF-SAT $\leq_p$ 3SAT

<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

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

$1\ 0\ 1\ 0\ 1$

TRUE

3SAT

- CNF-SAT $\leq_p$ 3SAT

<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

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

TRUE

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

3SAT

- CNF-SAT $\leq_p$ 3SAT

<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

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

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

3SAT

- CNF-SAT $\leq_p$ 3SAT

<Proof Idea>

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

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

If this clause is FALSE, every literal is 0

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

At most one TRUE

At most one TRUE

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

3SAT

- CNF-SAT $\leq_p$ 3SAT

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

3SAT

- CNF-SAT $\leq_p$ 3SAT

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

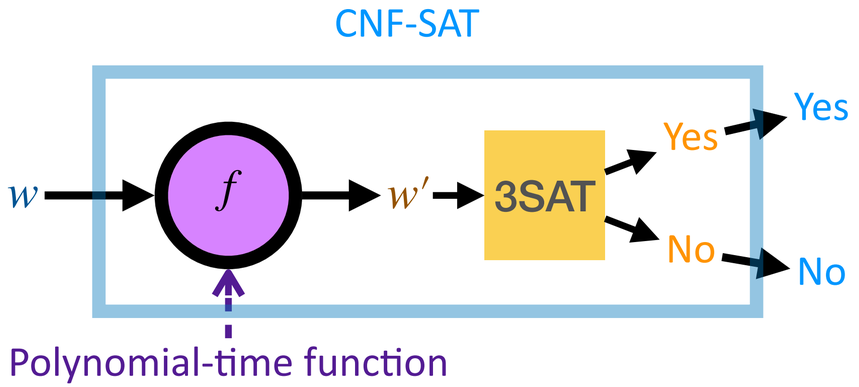
To reduce CNF-SAT to 3SAT, we convert any CNF-formula F into a 3CNF-formula F', with F is satisfiable if and only if F' is satisfiable:

3SAT

- CNF-SAT $\leq_p$ 3SAT

<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:

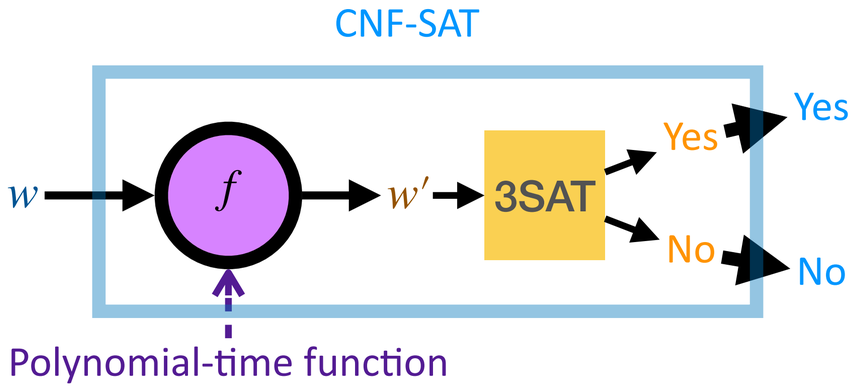


3SAT

- CNF-SAT $\leq_p$ 3SAT

<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**:

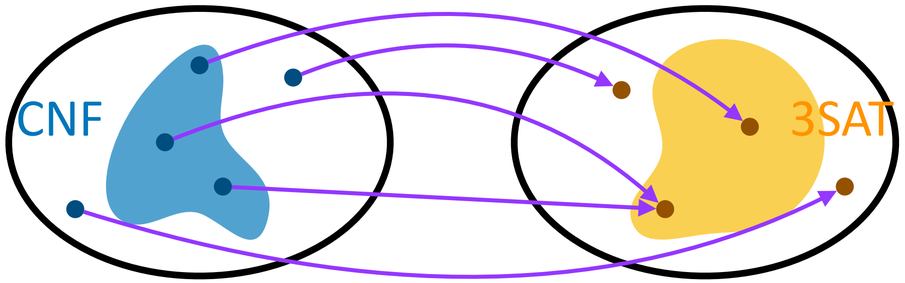


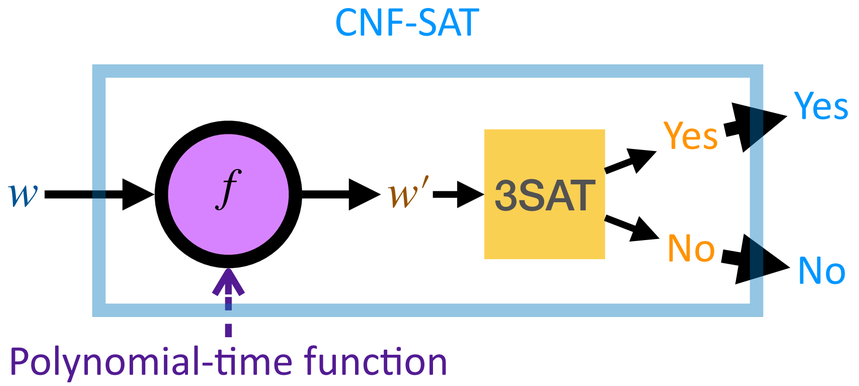
3SAT

- CNF-SAT $\leq_p$ 3SAT

<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**:





3SAT

- CNF-SAT $\leq_p$ 3SAT

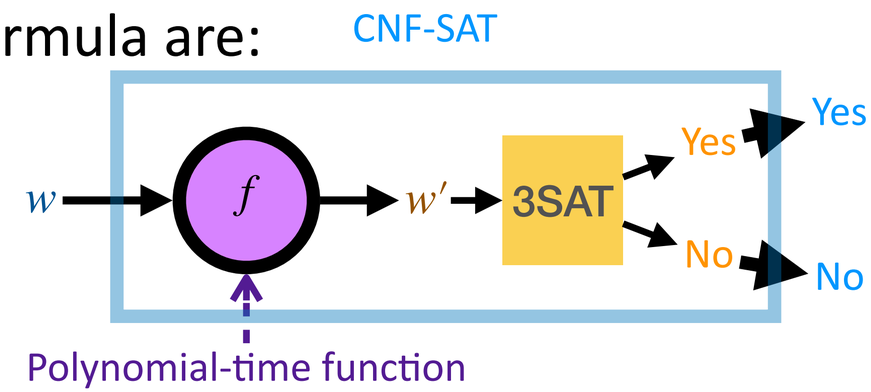
<Proof> Polynomial time reduction from CNF-SAT (since we have known that CNF is NP-Complete).

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

- CNF-SAT $\leq_p$ 3SAT

<Proof (cont.)>

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

3SAT

- CNF-SAT $\leq_p$ 3SAT

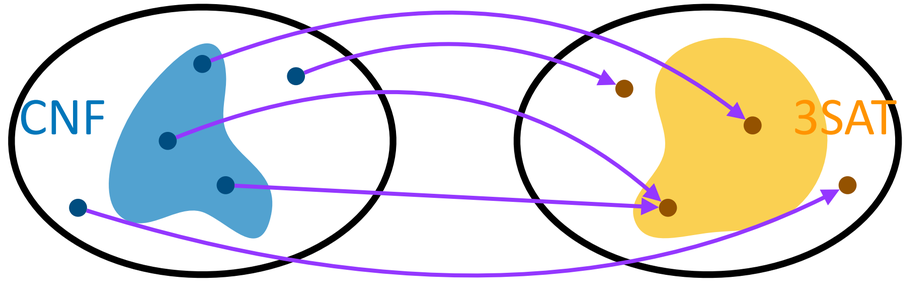
<Proof (cont.)>

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

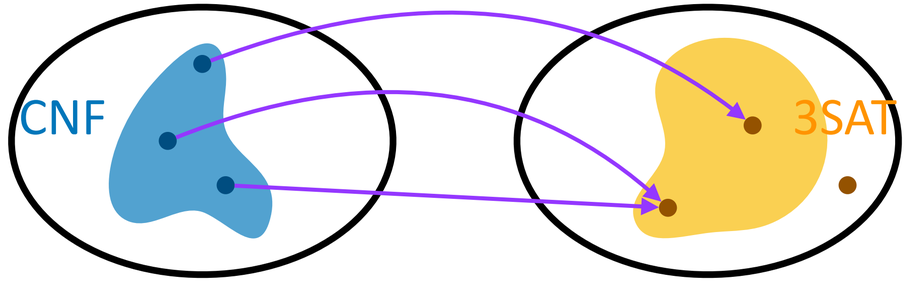


- CNF-SAT $\leq_p$ 3SAT

<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.

3SAT



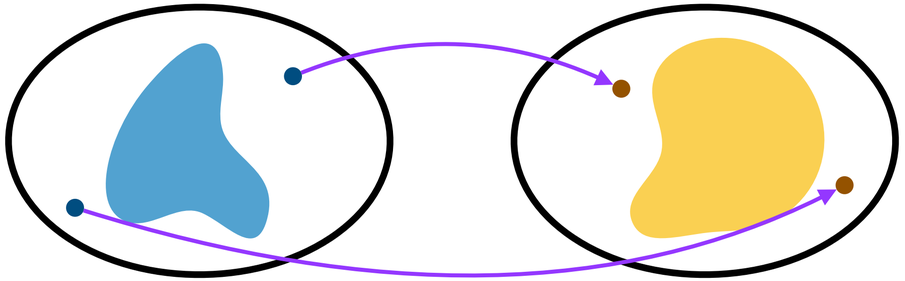
- CNF-SAT $\leq_p$ 3SAT

<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



- CNF-SAT $\leq_p$ 3SAT

<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 ϕ 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 ϕ that has more than 3 literals
    - $\Rightarrow$ make a chain of 3-clauses in ϕ', 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 ϕ is *TRUE* if and only if the corresponding (chain of) clauses in ϕ' are all *TRUE*

Reduction and Hardness

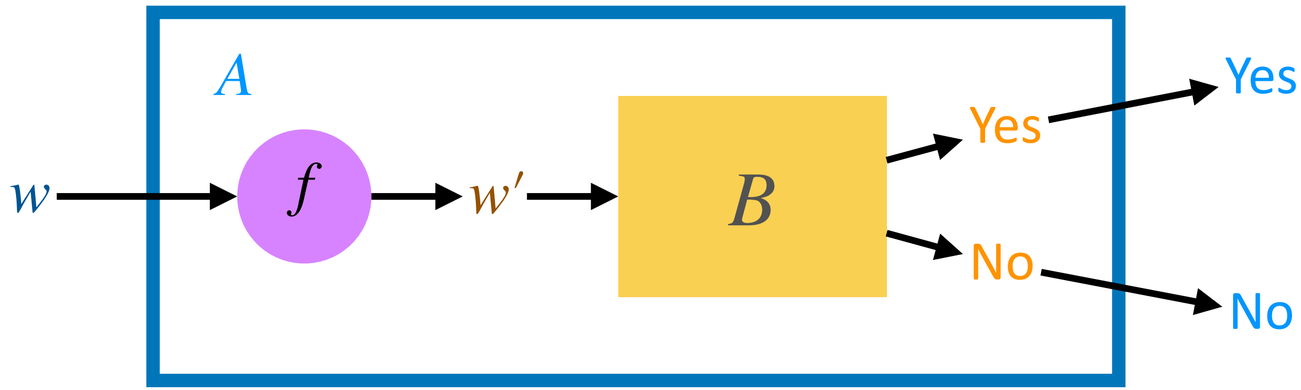
- Reduction from problem A to problem B

Reduction and Hardness

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

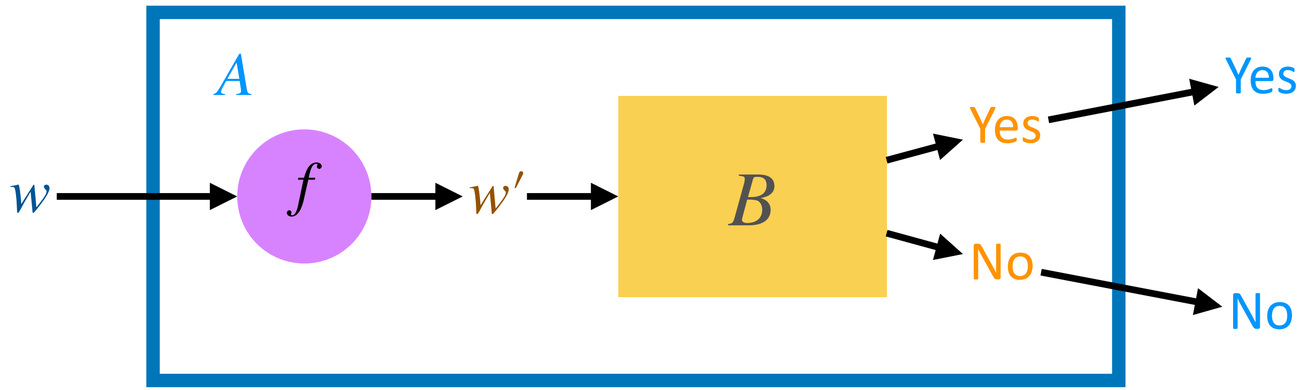
Reduction and Hardness

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



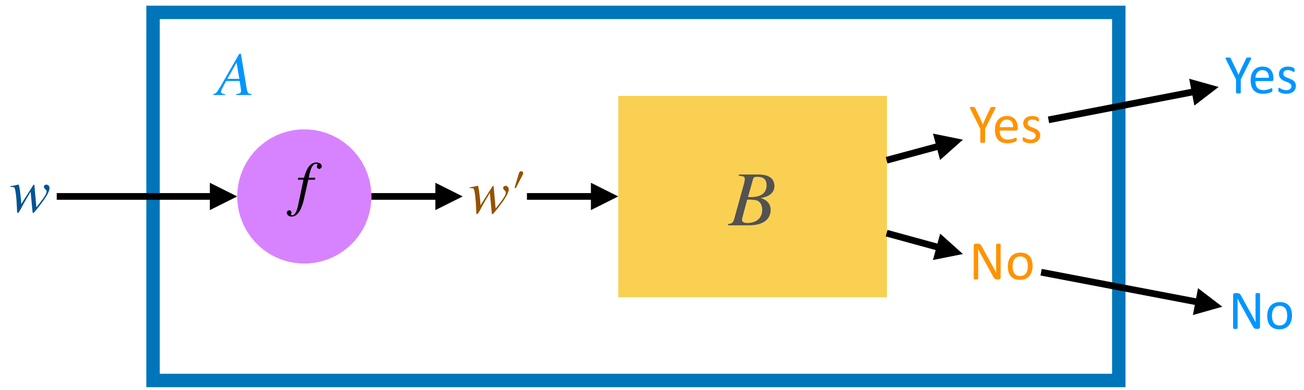
Reduction and Hardness

- 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



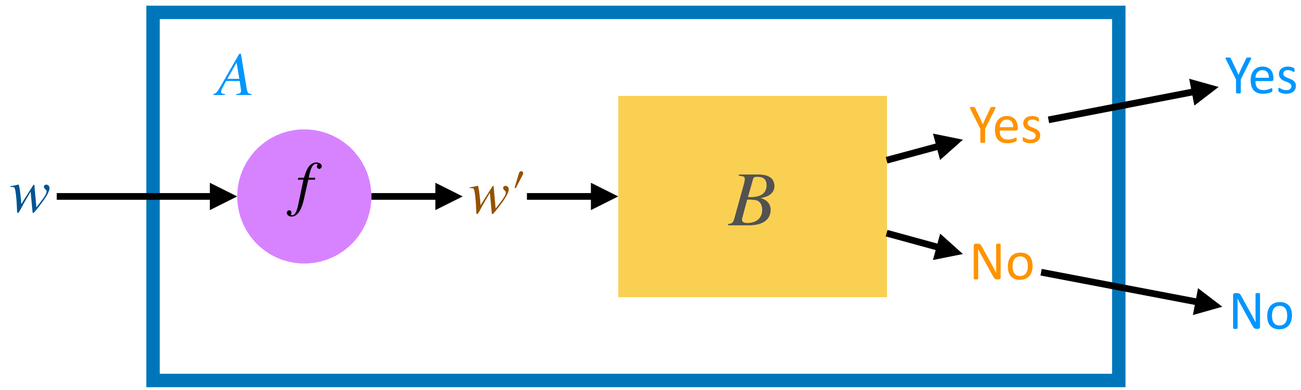
Reduction and Hardness

- 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 not harder 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 not harder than problem B
    - Problem$B$ is not easier than problem A



NP-Hard

NP-Hard

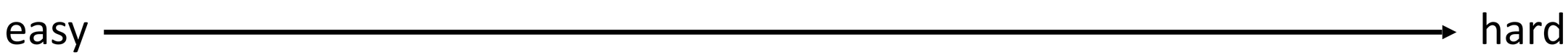
- Definition: A problem B is ***NP-hard*** if all problems in **NP** can be polynomial-time reduced to B

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**



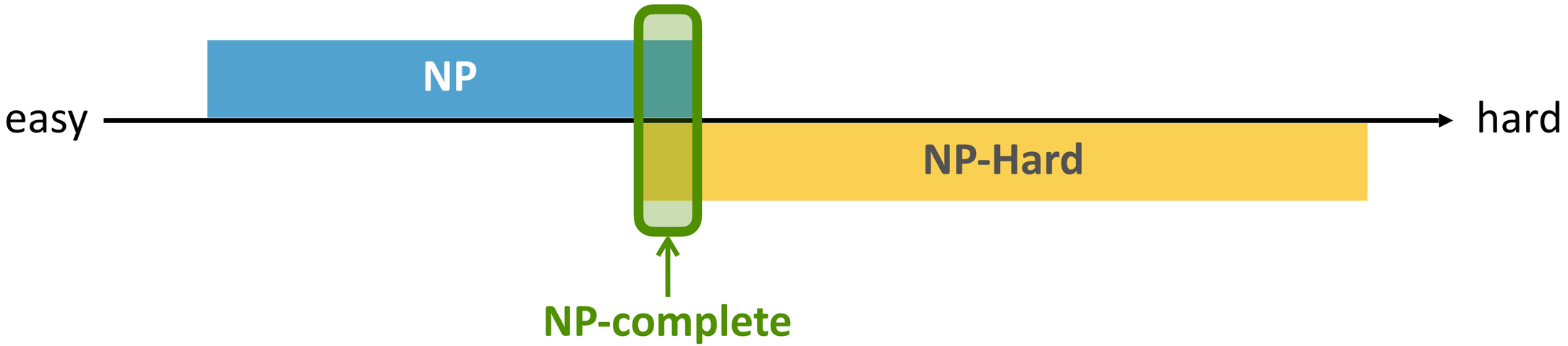
NP-Hard

- 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**

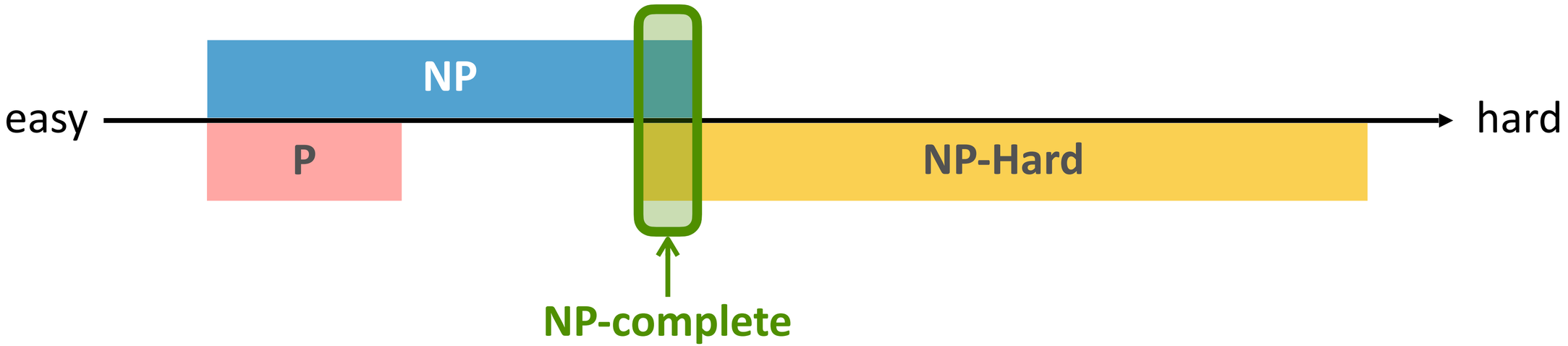
NP-Hard

- 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

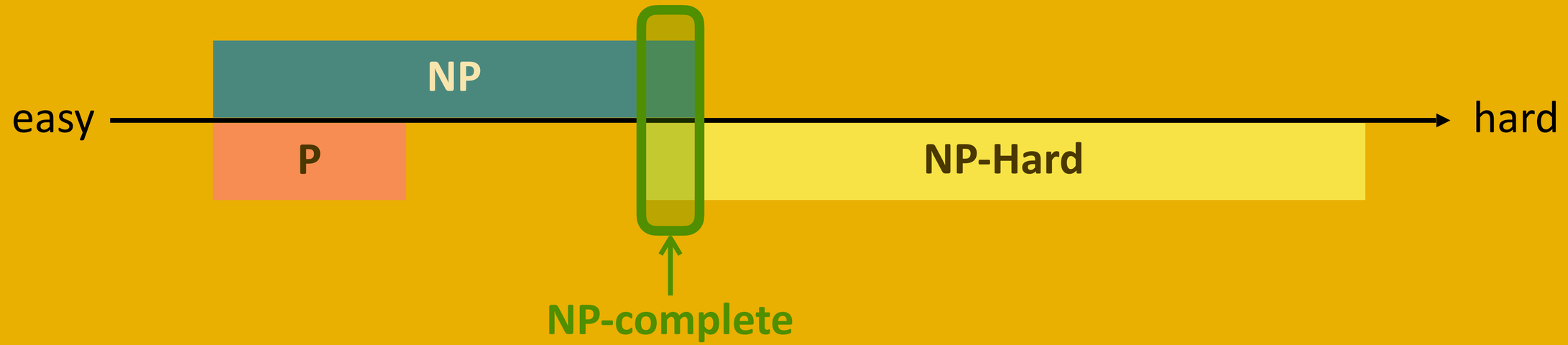


NP-Complete



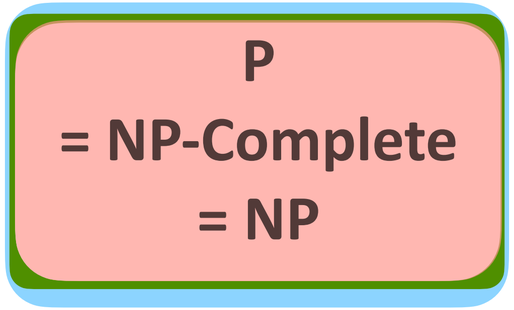
What Happened

- 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**



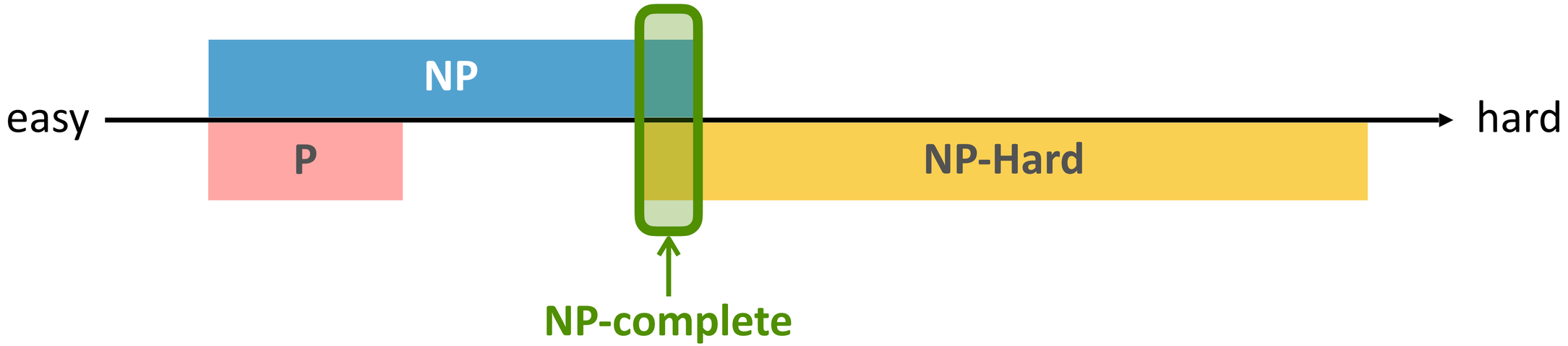
NP-Completeness Revisit

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



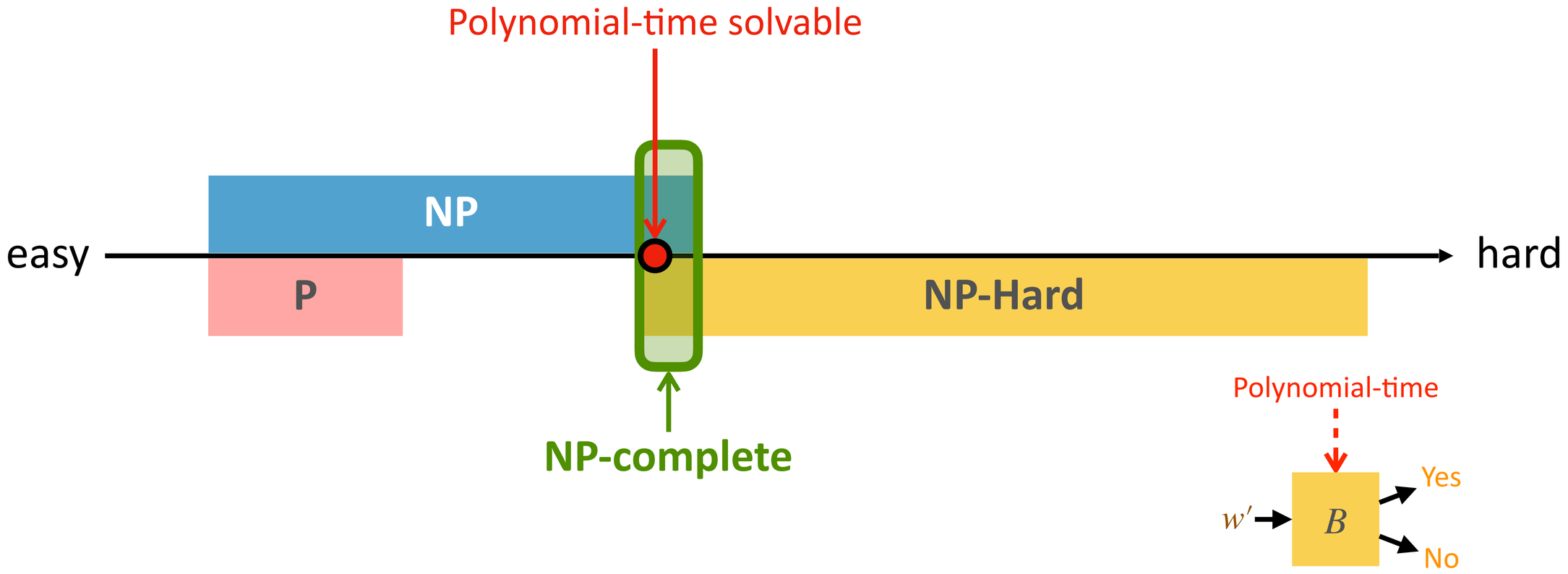
NP-Completeness Revisit

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



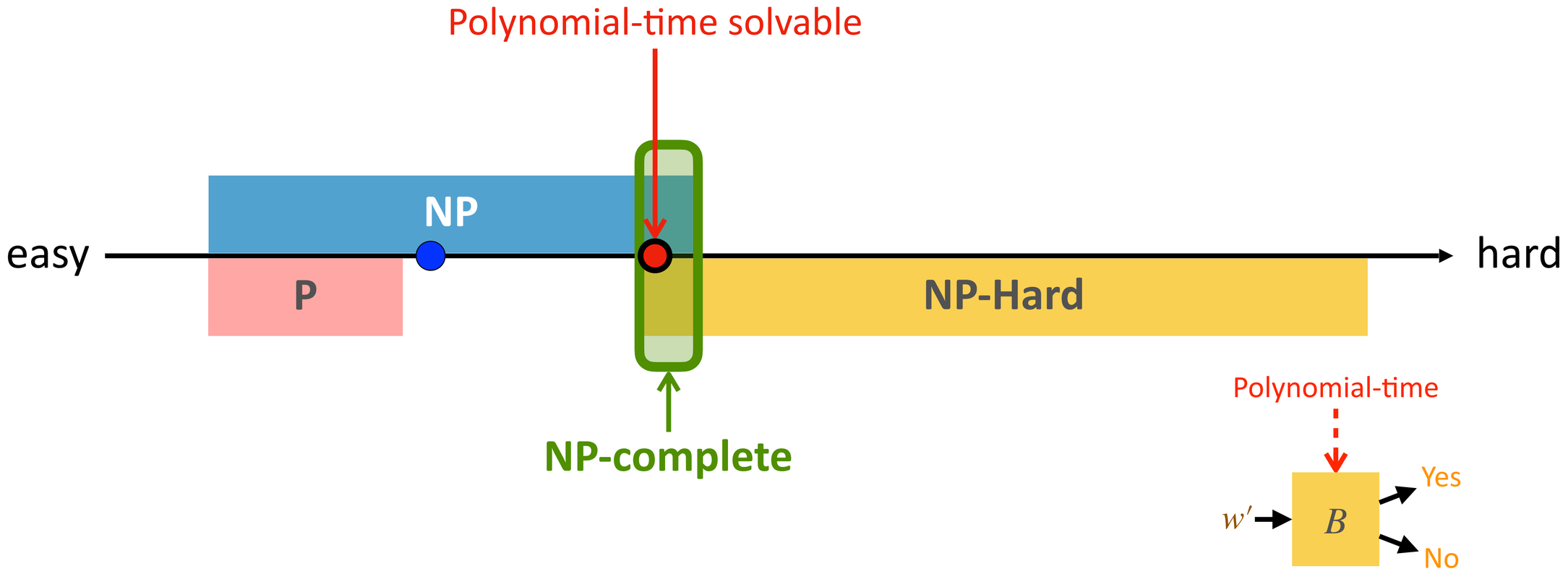
NP-Completeness Revisit

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



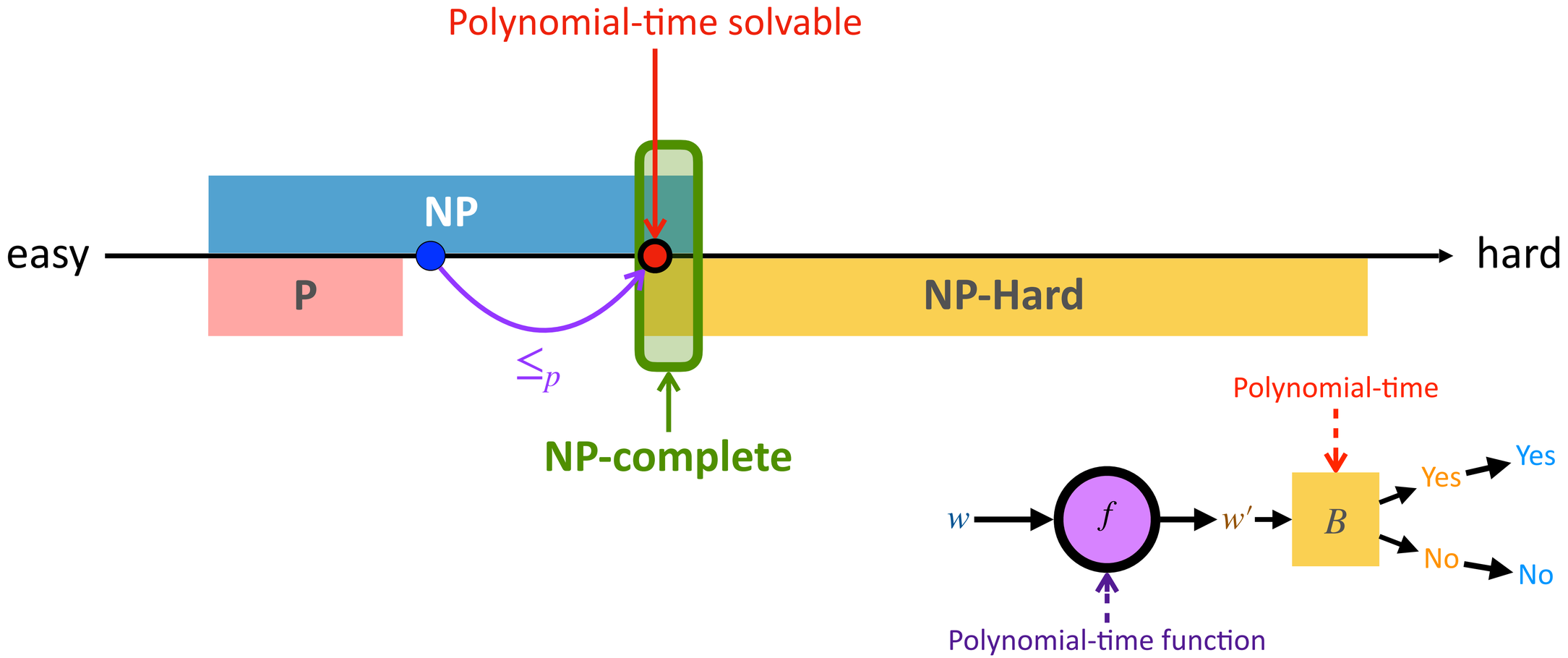
NP-Completeness Revisit

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



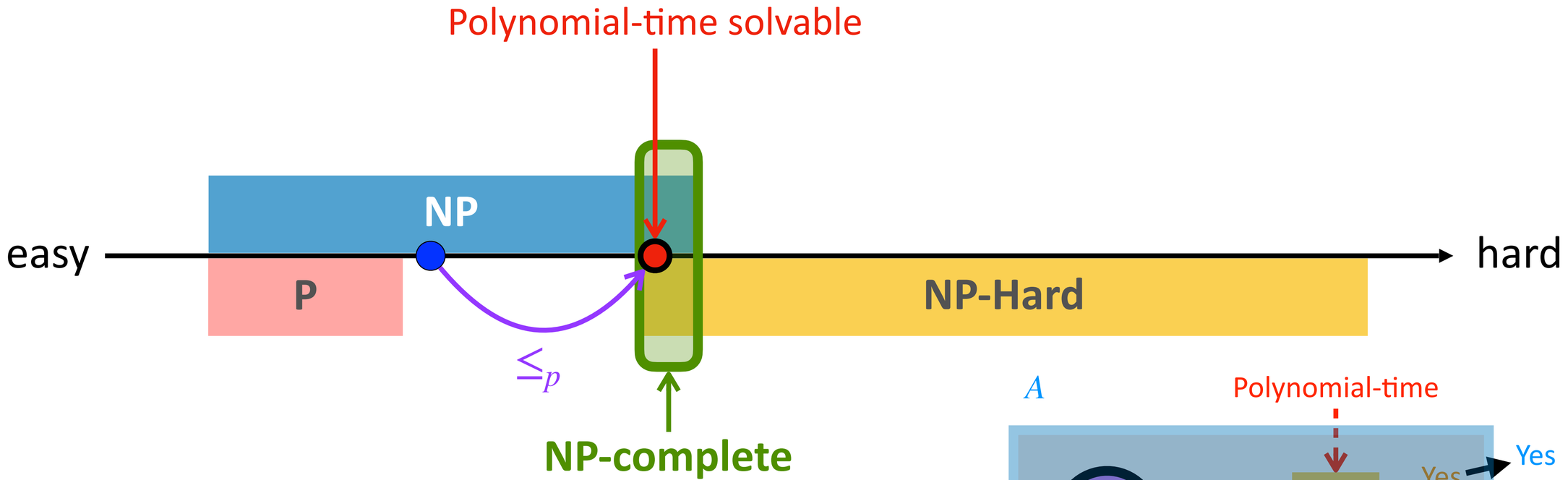
NP-Completeness Revisit

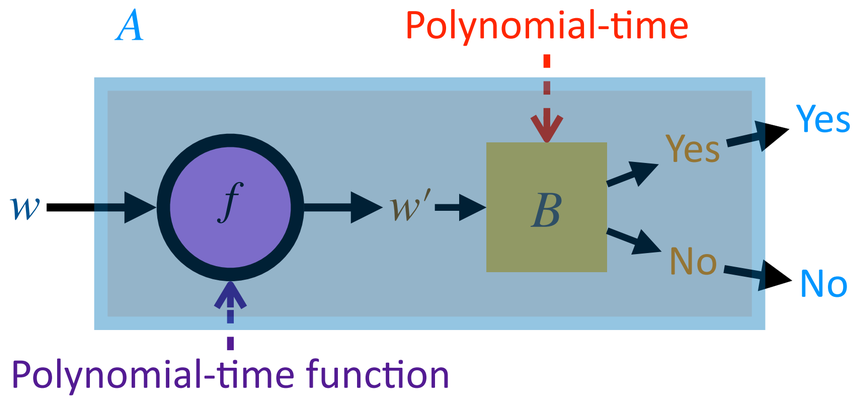
- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



NP-Completeness Revisit

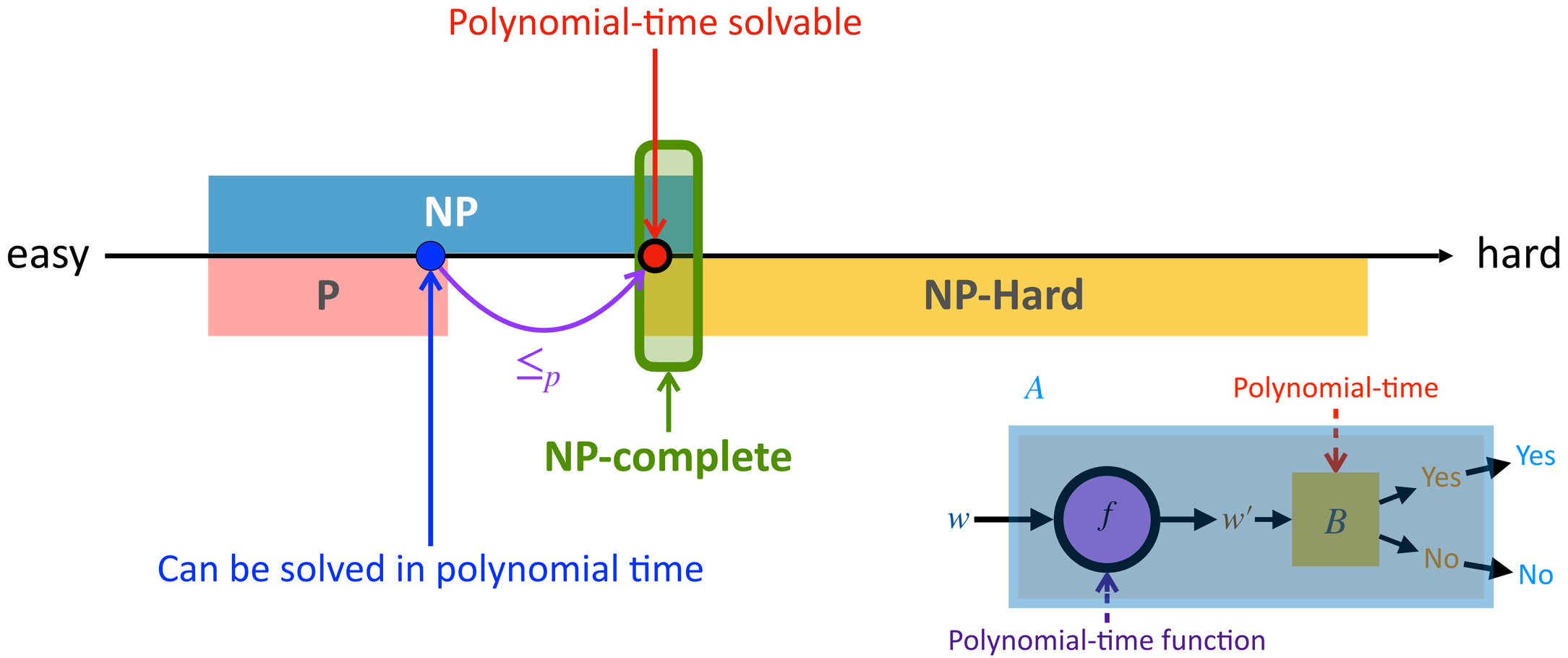
- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time





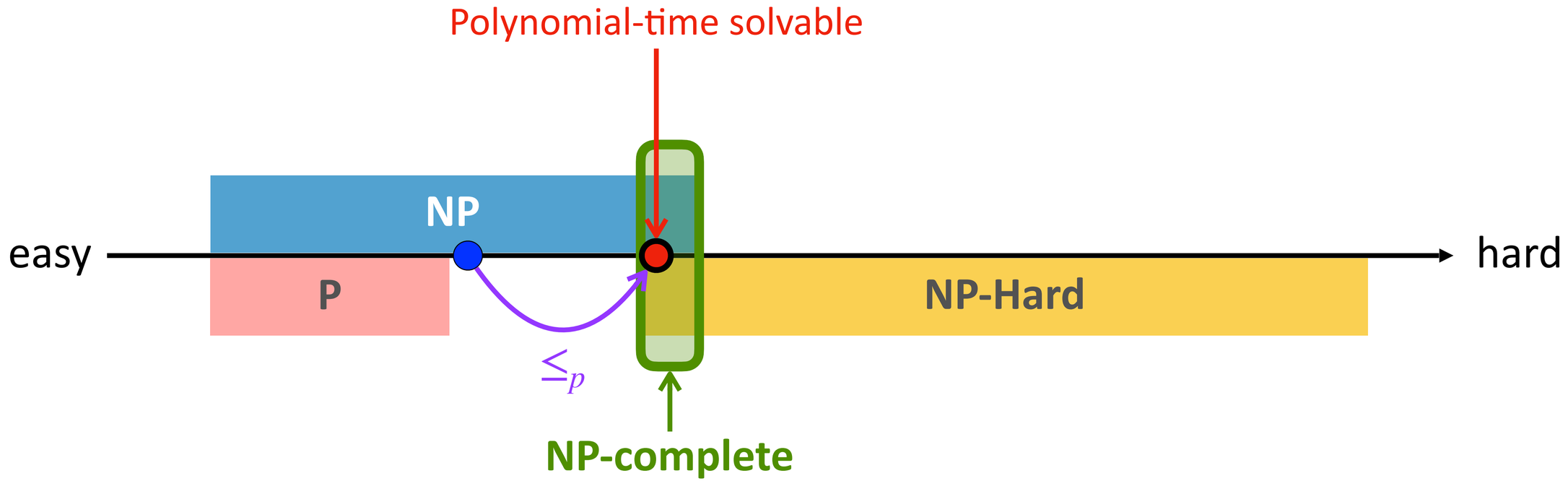
NP-Completeness Revisit

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



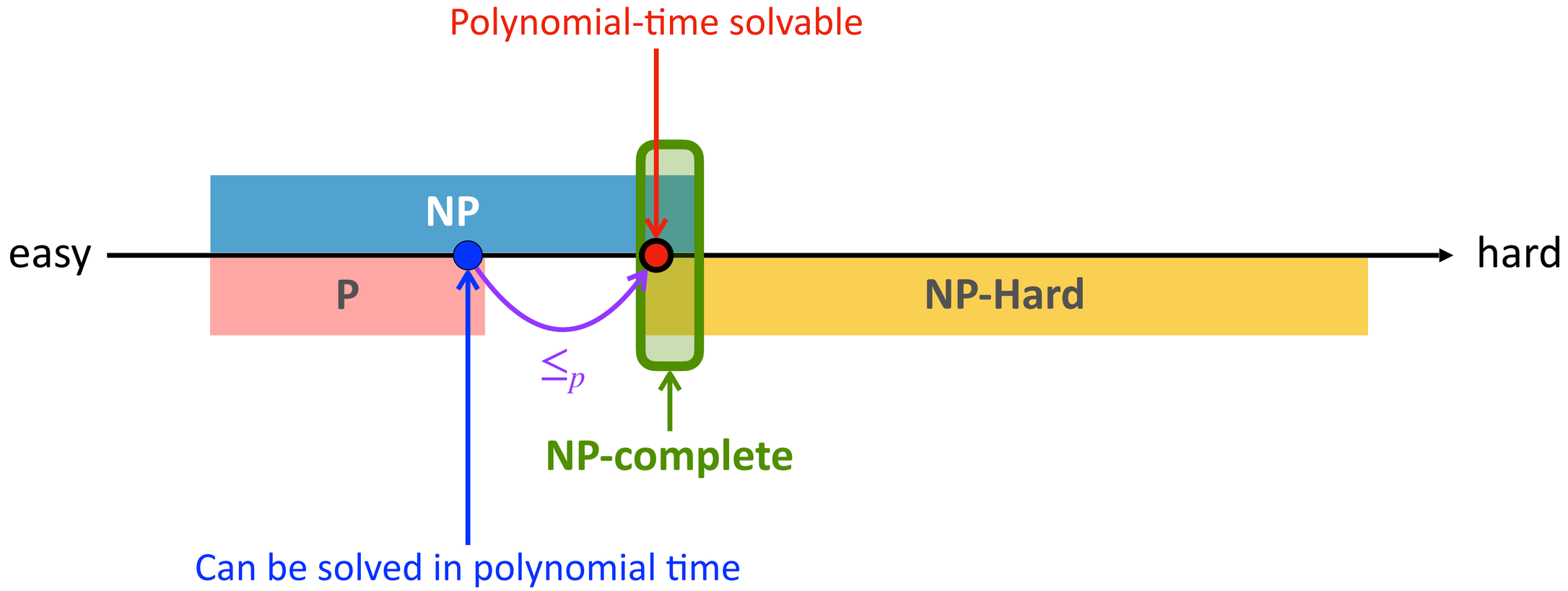
NP-Completeness Revisit

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



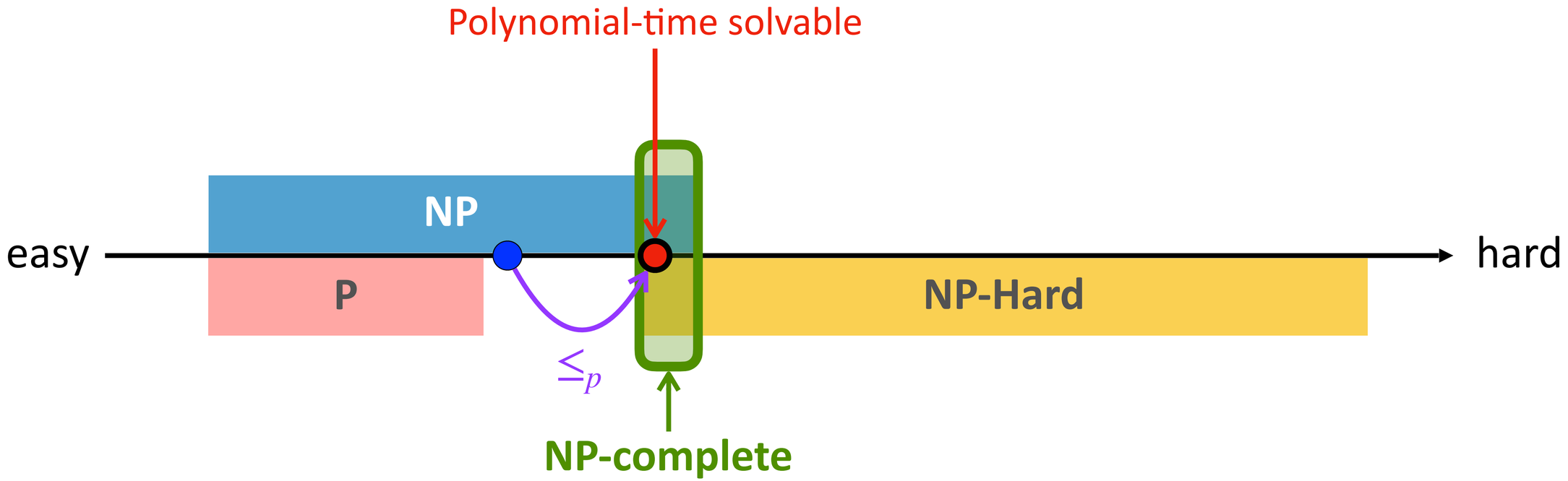
NP-Completeness Revisit

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



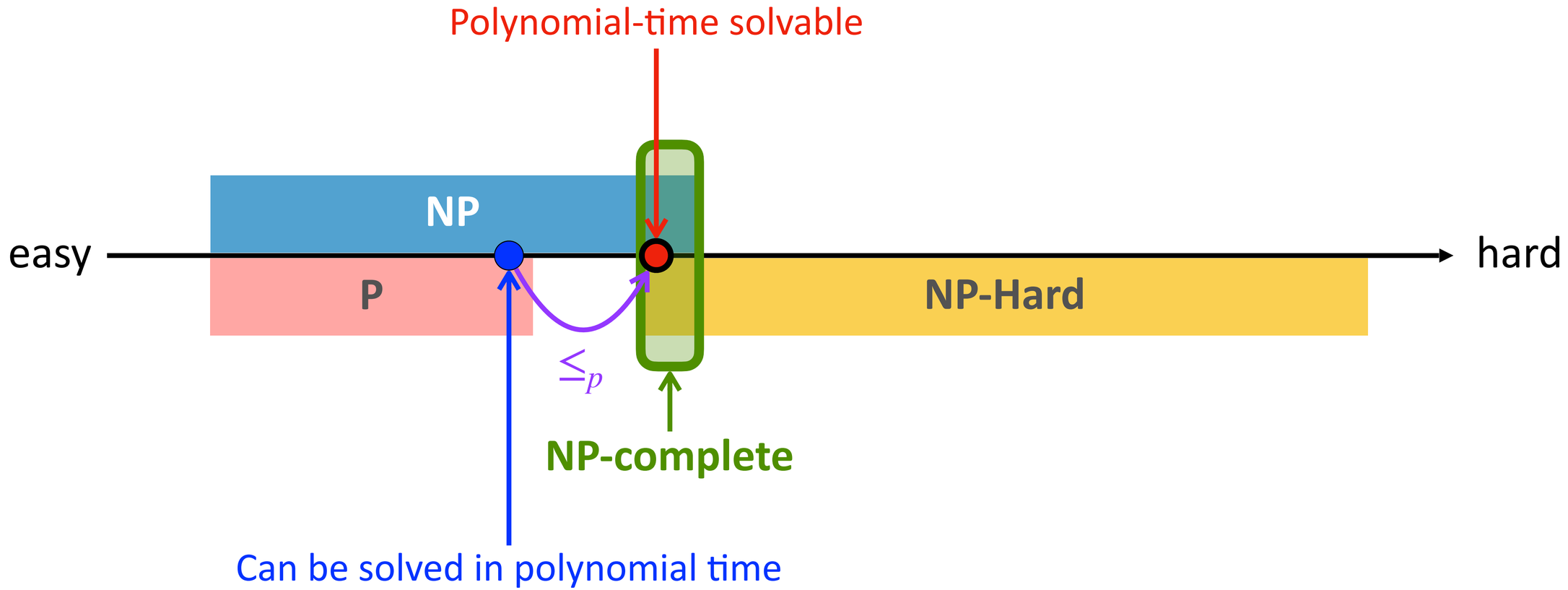
NP-Completeness Revisit

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



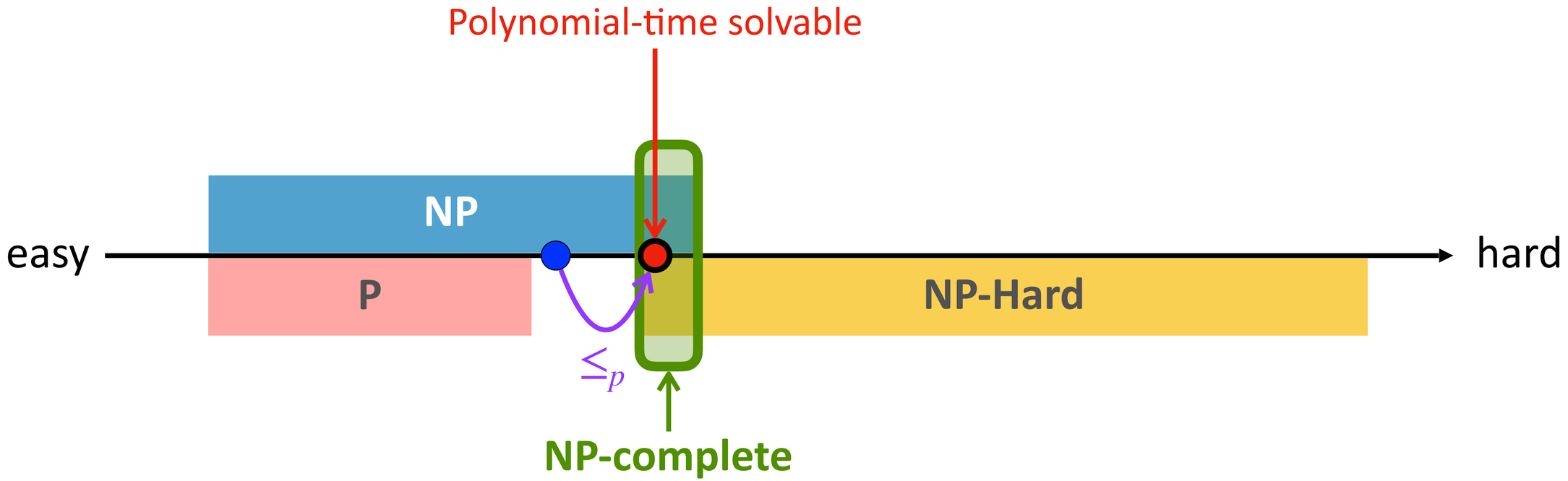
NP-Completeness Revisit

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



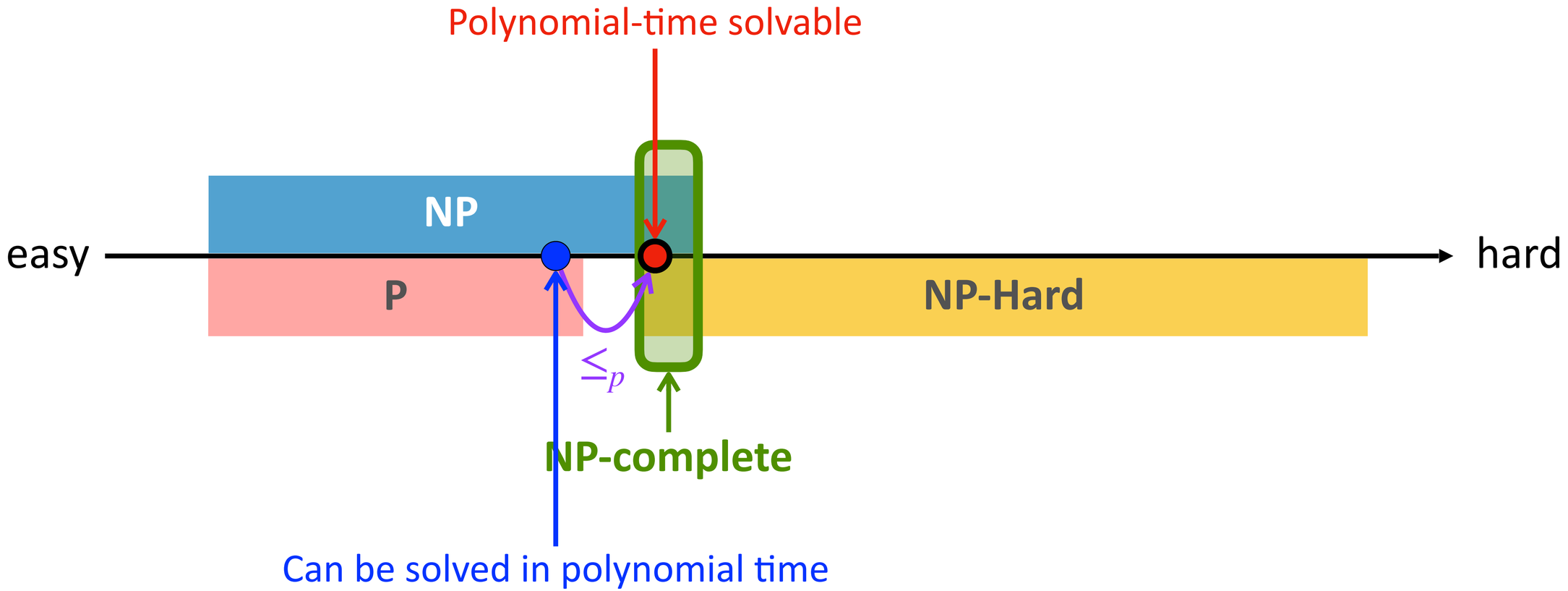
NP-Completeness Revisit

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



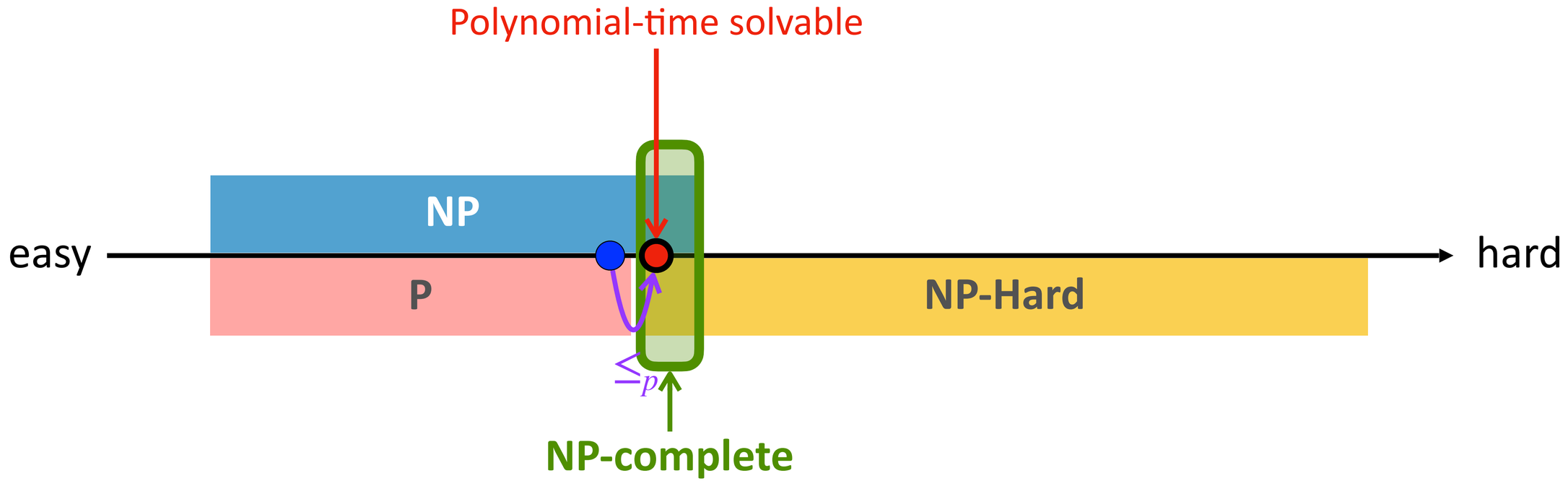
NP-Completeness Revisit

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



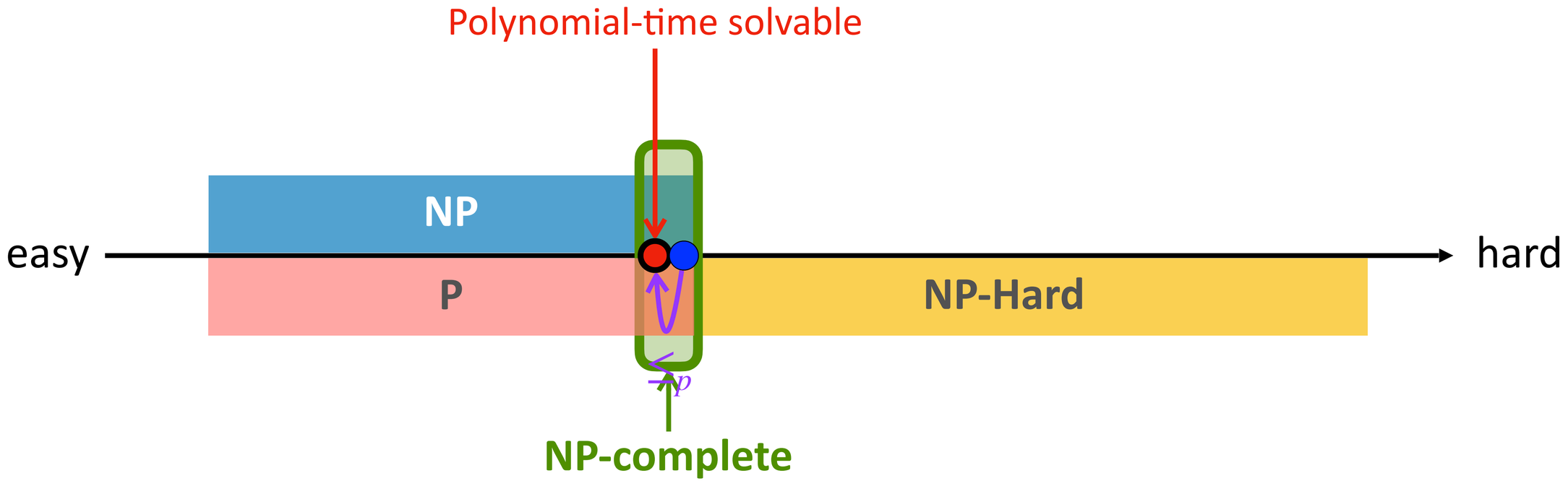
NP-Completeness Revisit

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



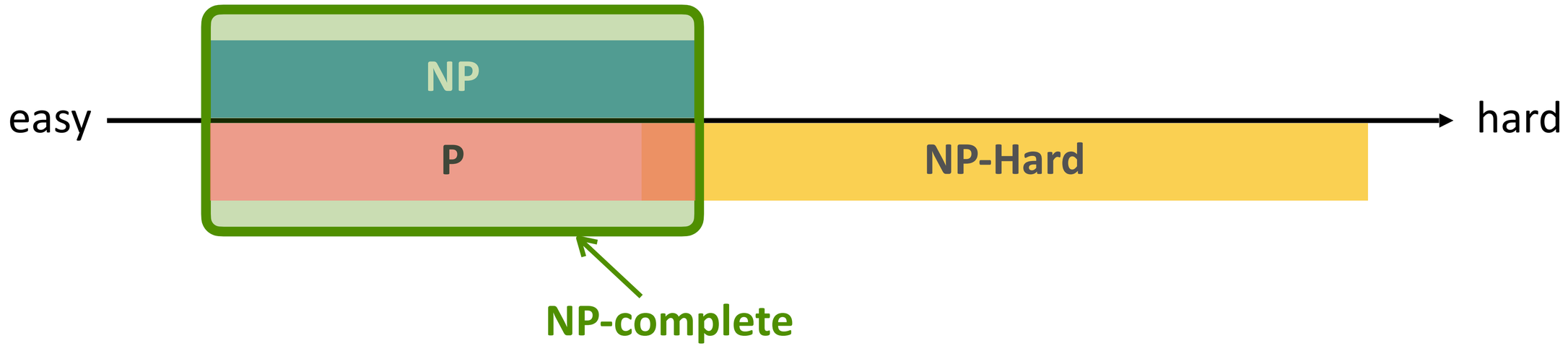
NP-Completeness Revisit

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

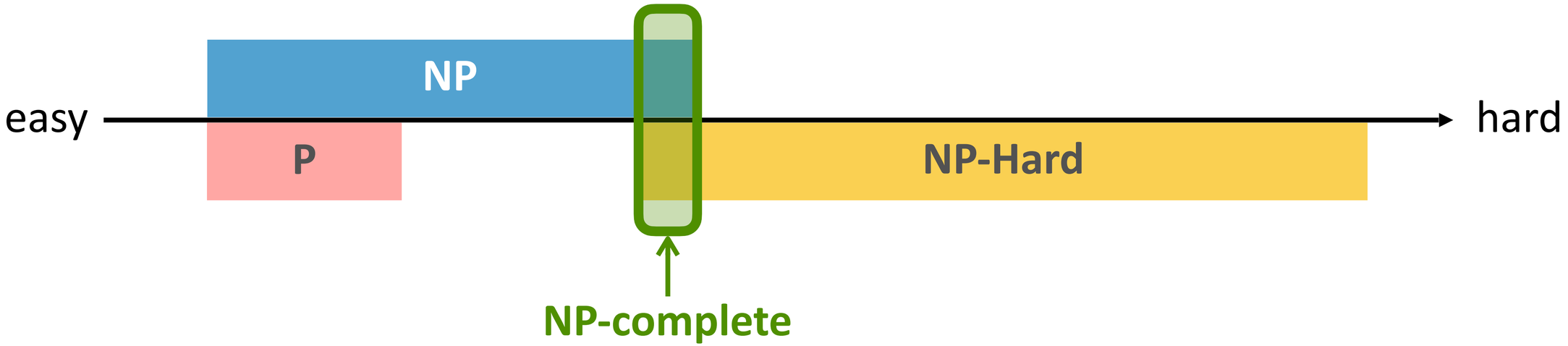


NP-Completeness Revisit

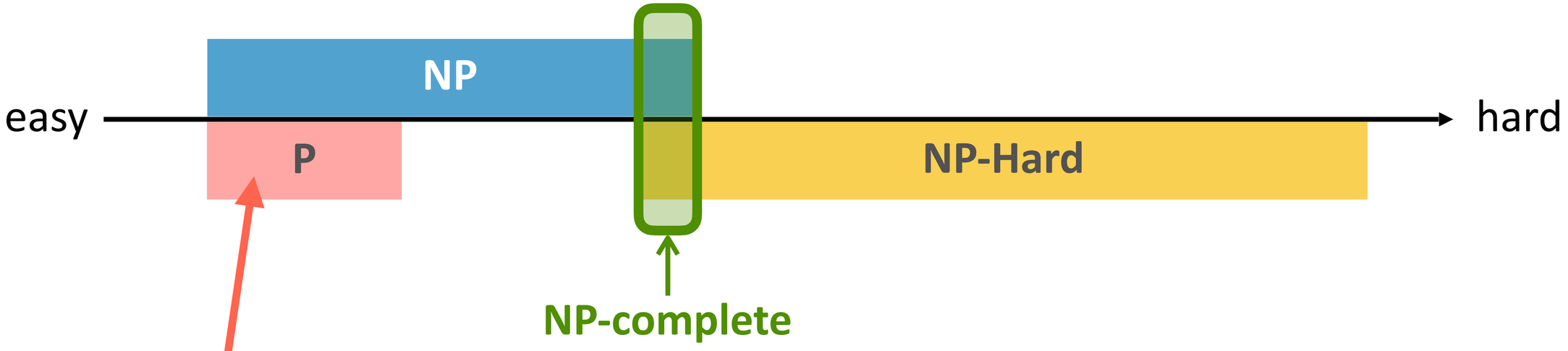
- If an **NP-complete** problem is shown to be polynomial-time solvable, *every* problem in NP can be solved in polynomial time



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

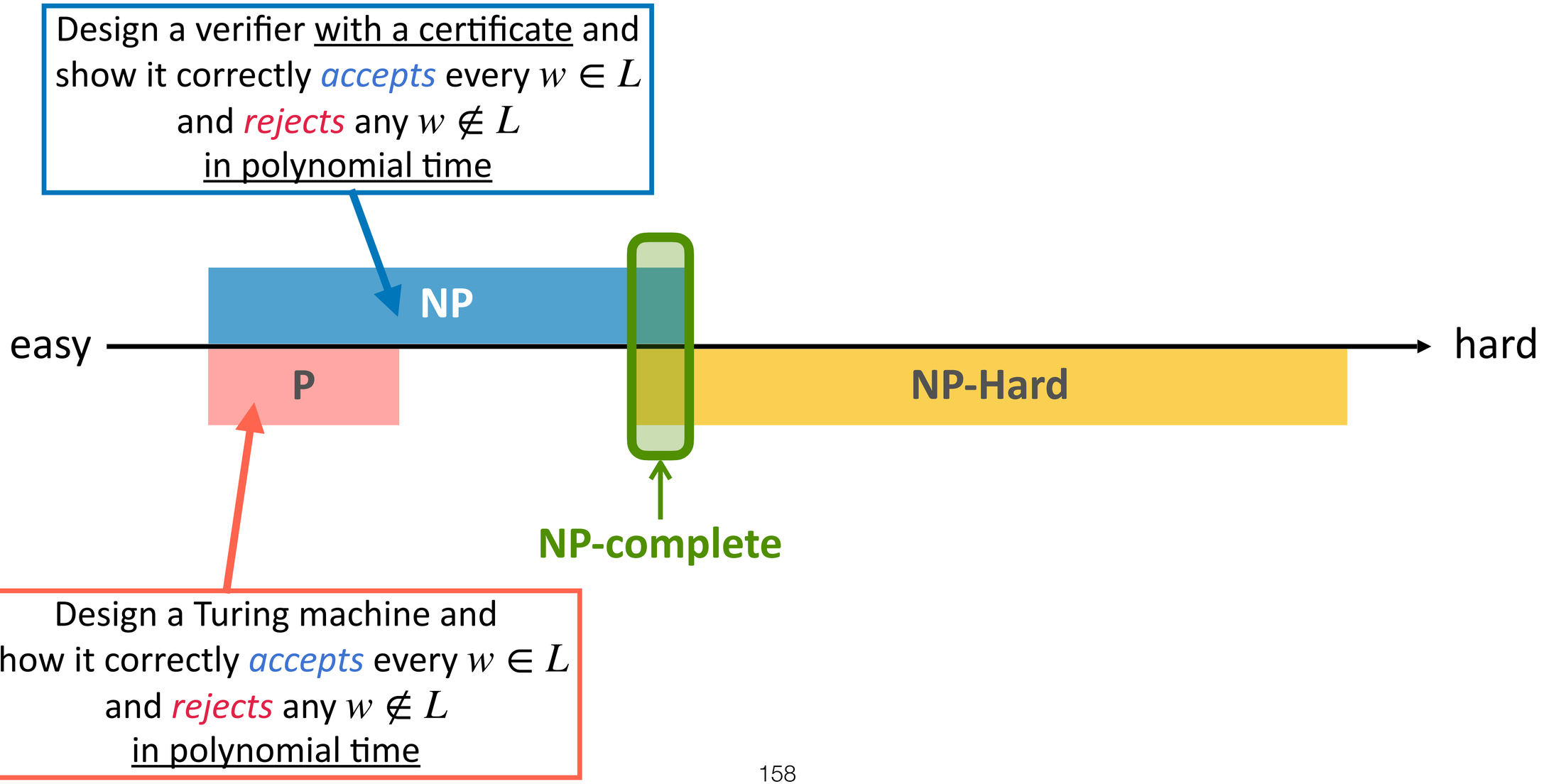


P, NP, NP-Hard, and NP-Complete

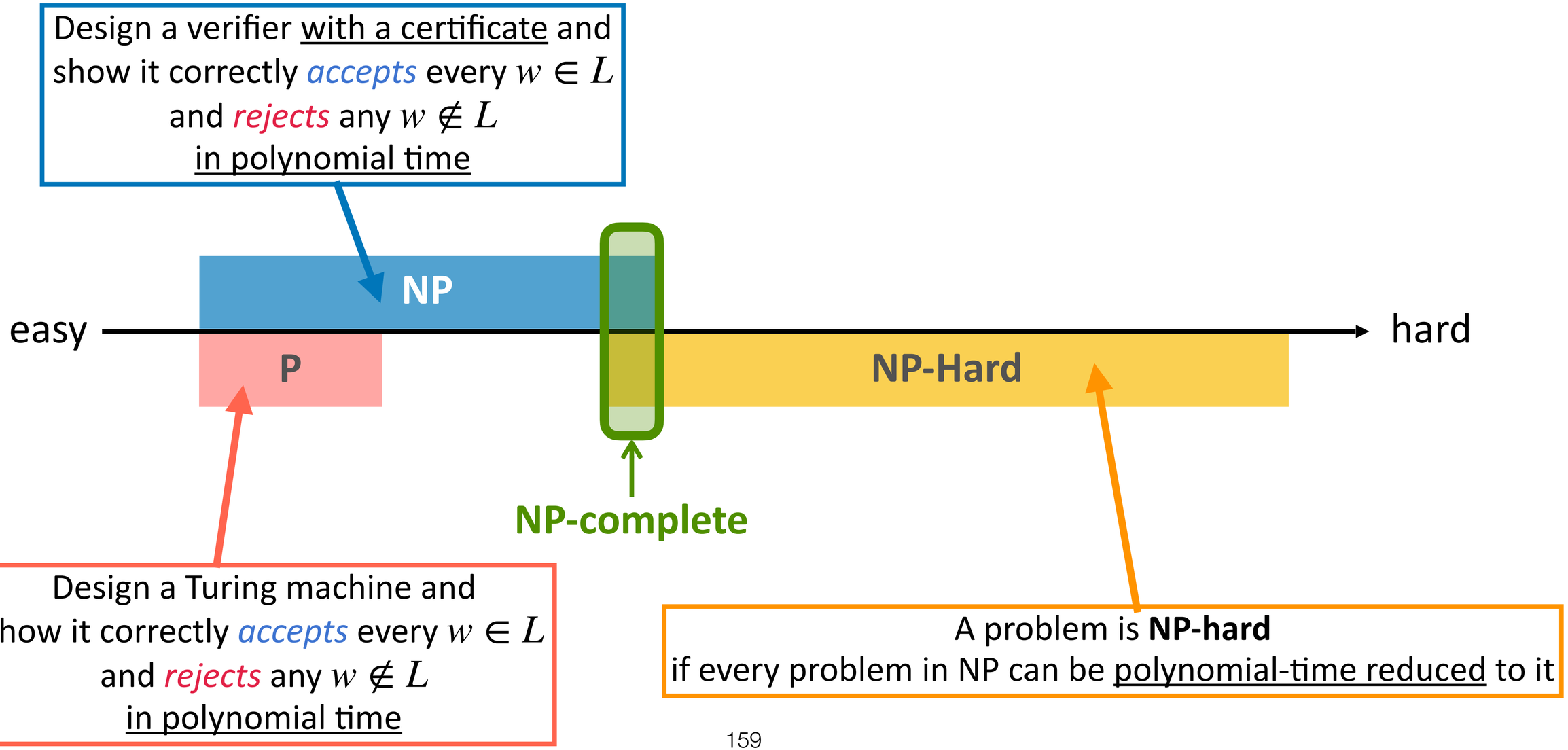


Design a Turing machine and
show it correctly *accepts* every $w \in L$
and *rejects* any $w \notin L$
in polynomial time

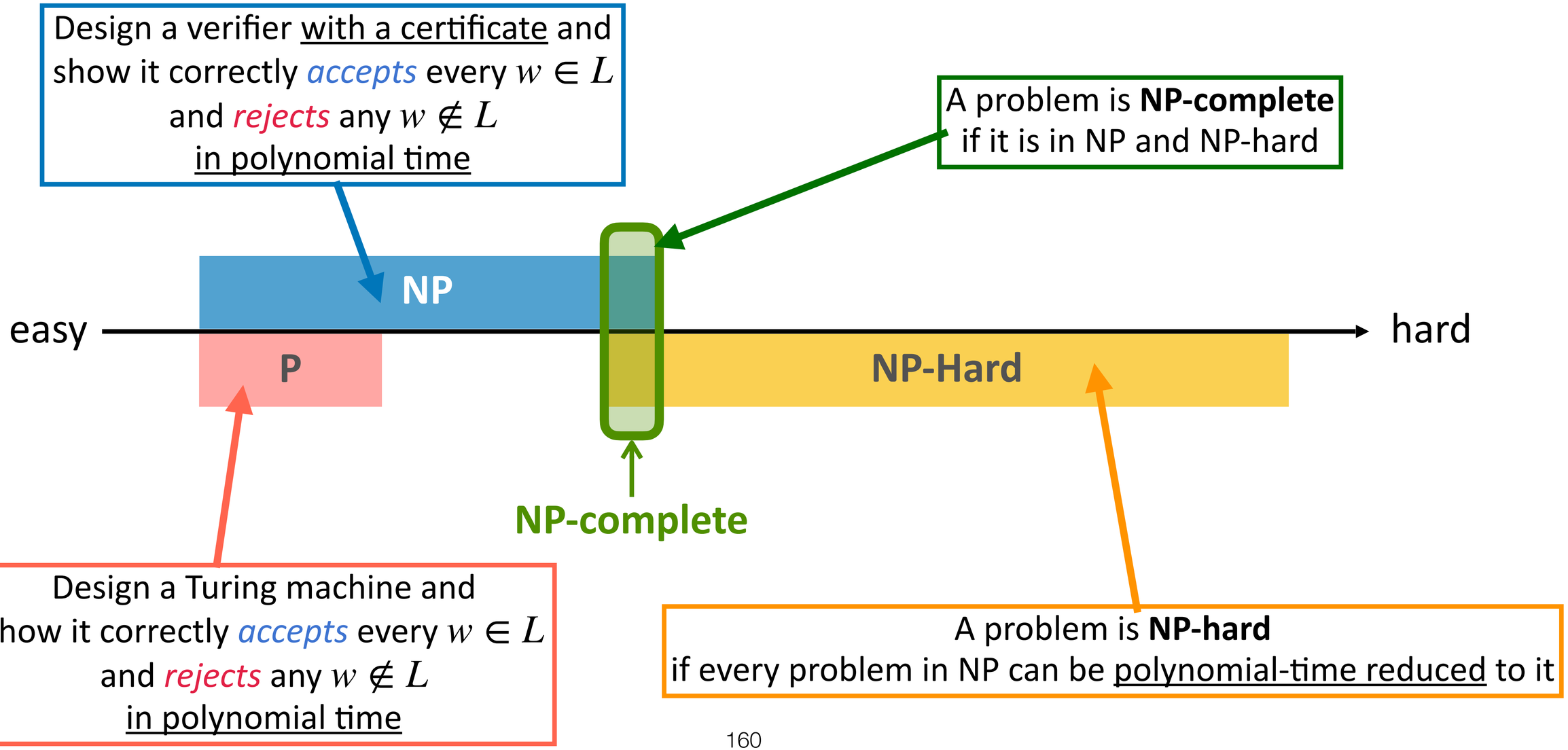
P, NP, NP-Hard, and NP-Complete



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



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



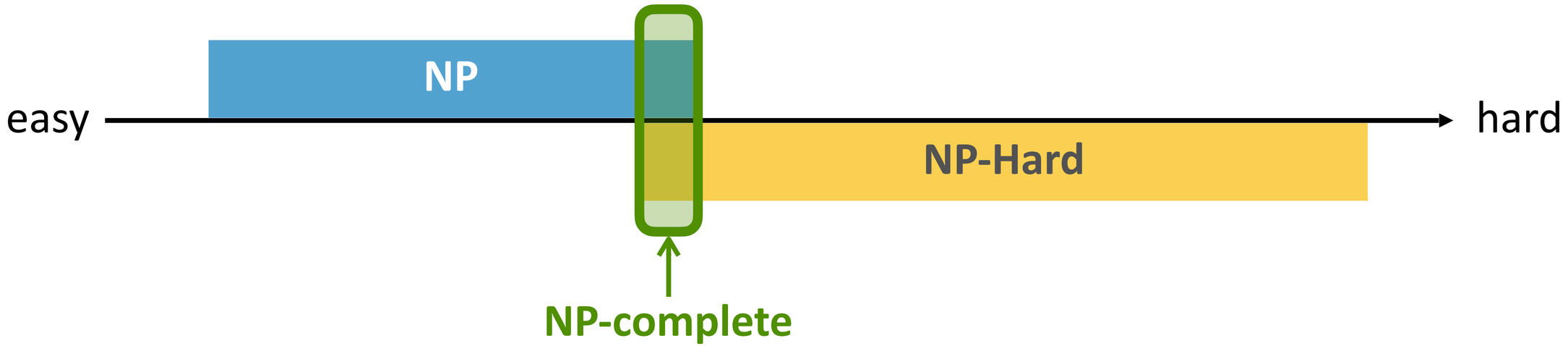
How to prove a problem is NP-Hard

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

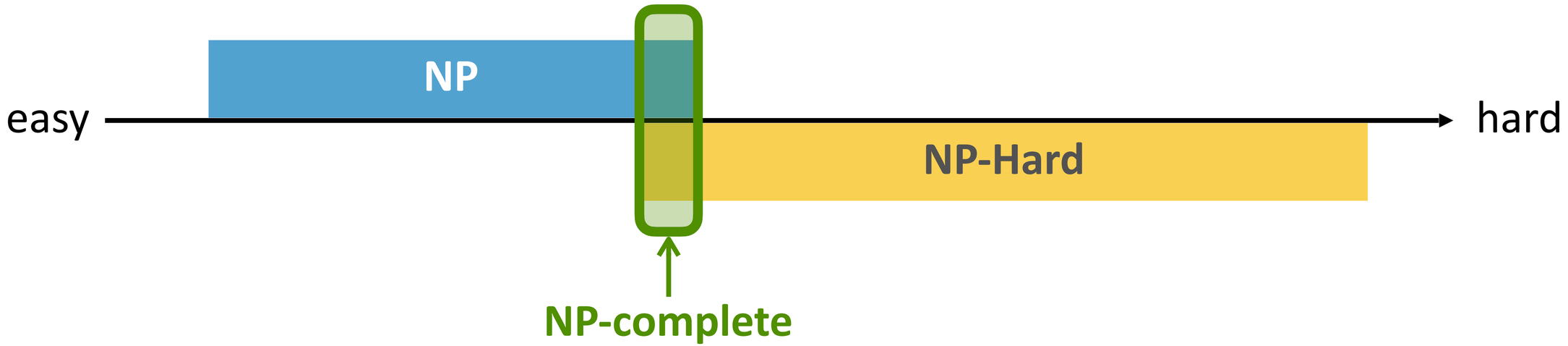
How to prove a problem is NP-Hard

- 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

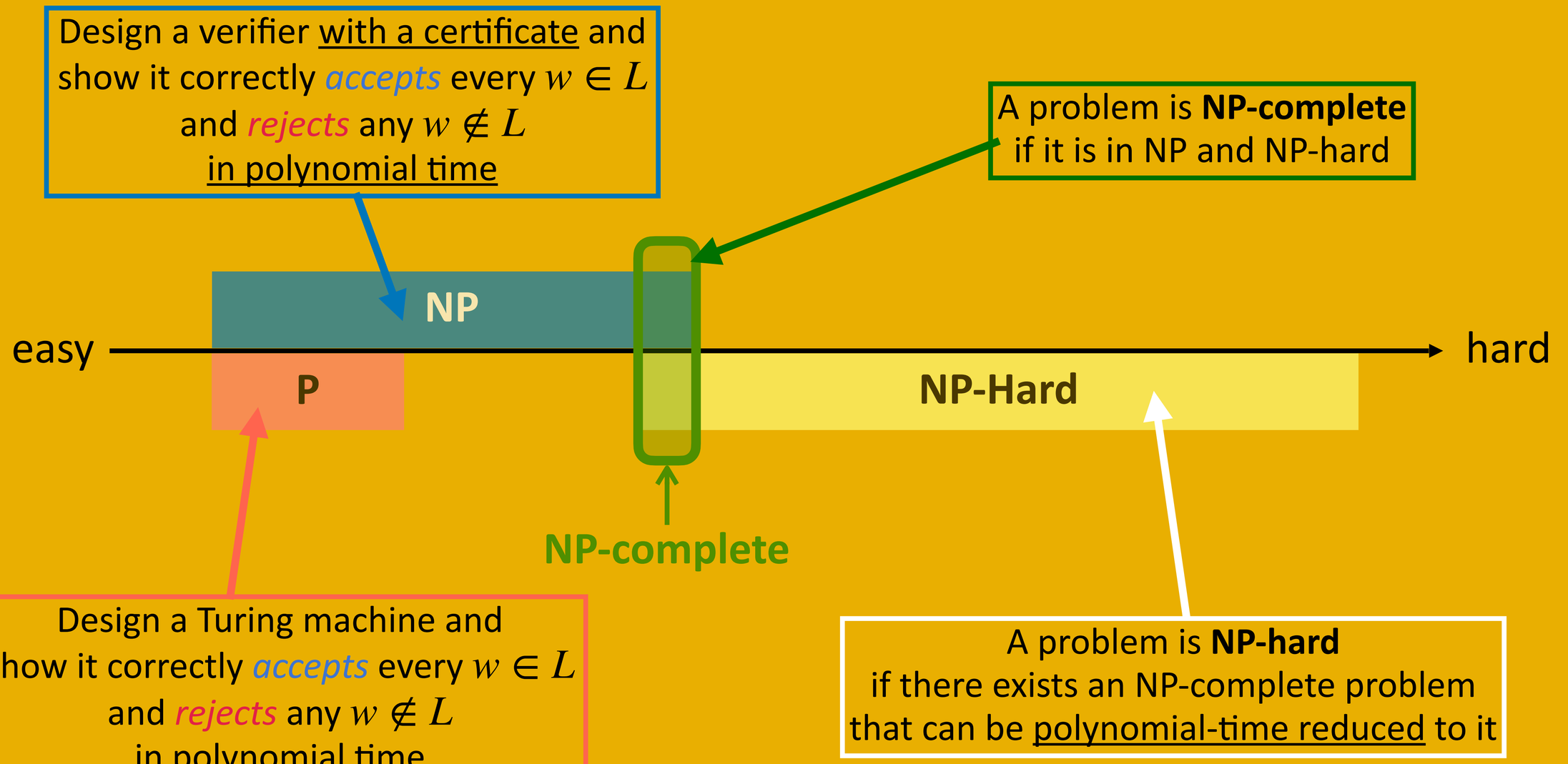


How to prove a problem is NP-Hard

- 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



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



~~Tattoo this on your arm~~

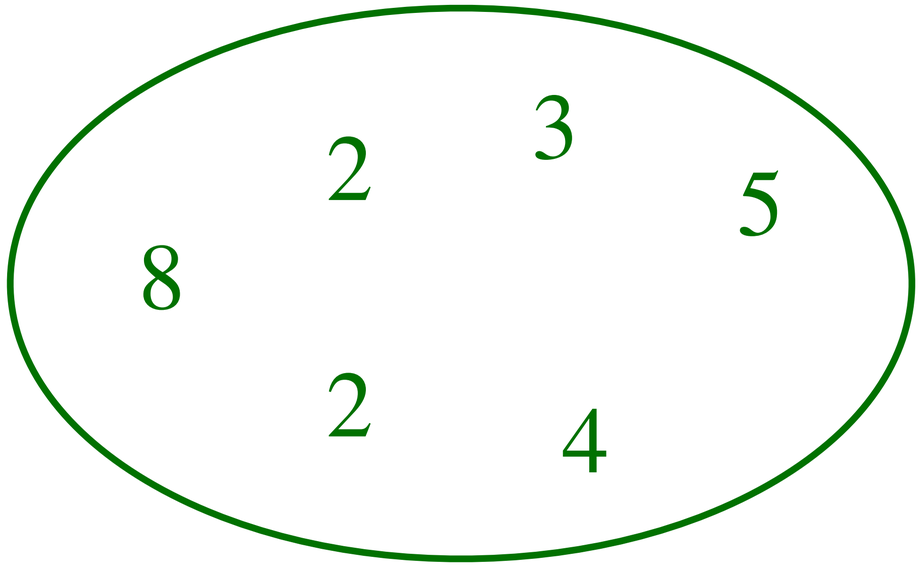
- If you want to prove some problem Q is NP-hard, reduce some NP-complete problem Q' to Q.

Outline

- NP-Completeness
  - NP-hardness: Polynomial time reduction
    - CNF-SAT $\leq_p$ 3SAT
    - 3SAT $\leq_p$ SUBSET-SUM
    - 3SAT $\leq_p$ CLIQUE
    - PARTITION $\leq_p$ BIN-PACKING
- Cook-Leven Theorem: SAT is NP-complete

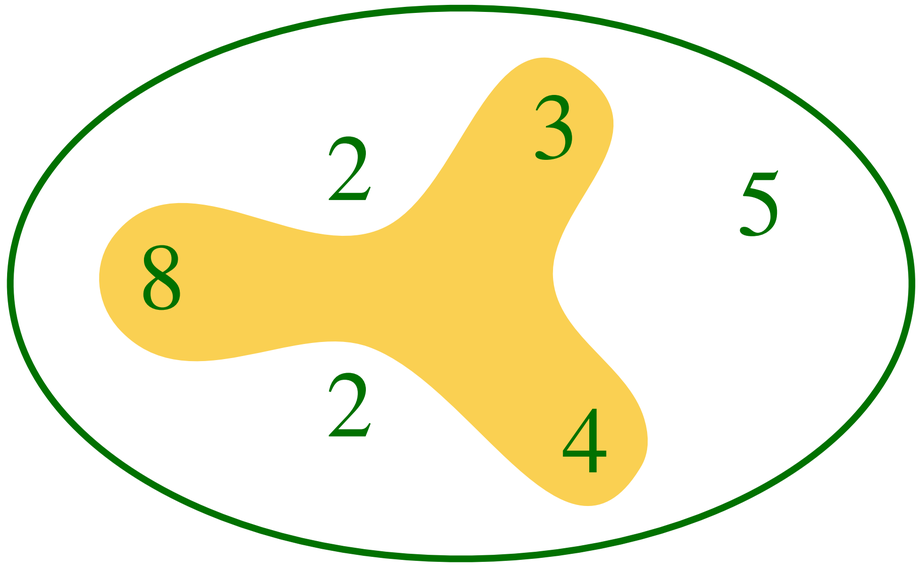
SUBSET-SUM

- 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



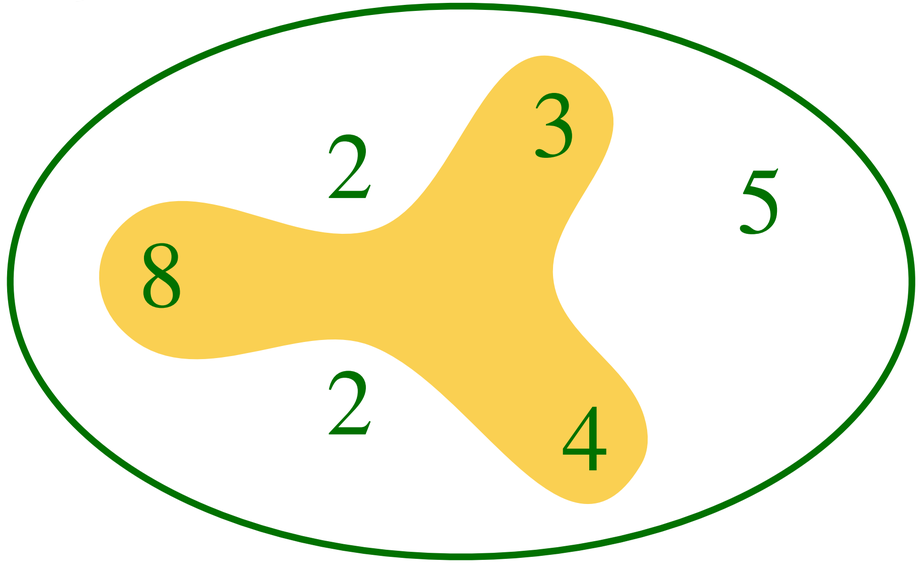
SUBSET-SUM

- 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$\}$

  ℓ 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$\}$

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

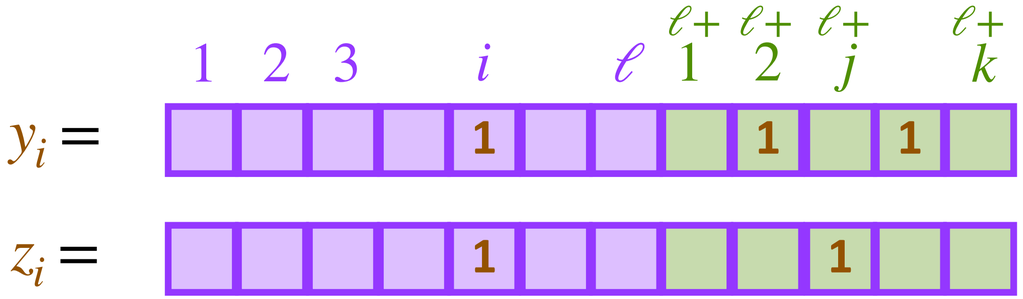
  k clauses $c_1, c_2, \cdots, c_k$

- 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^{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^{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

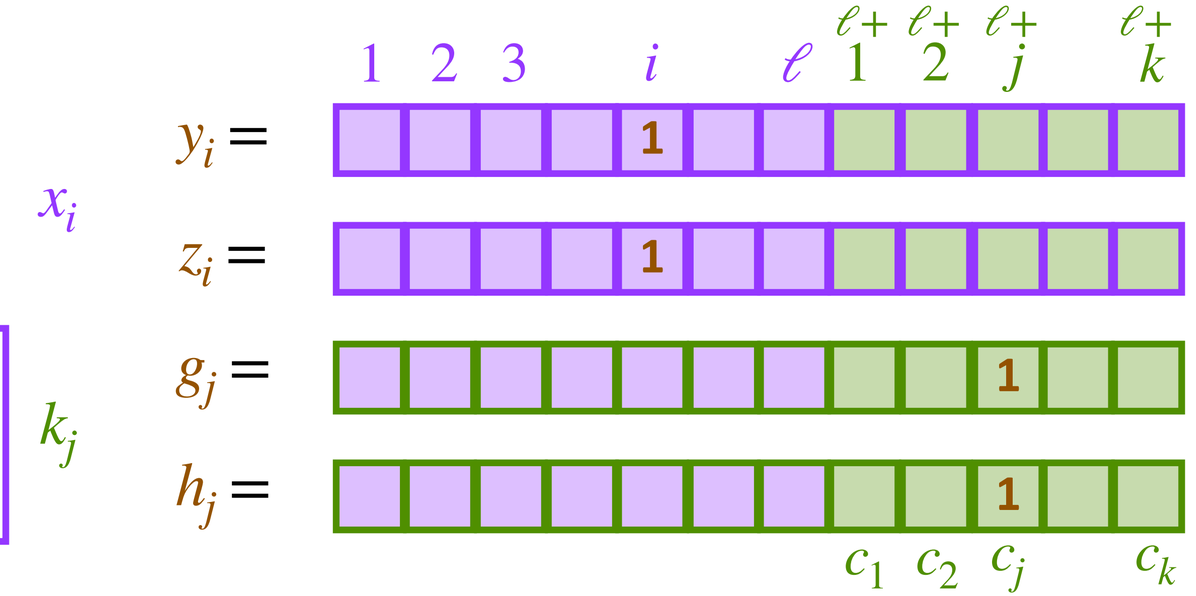
- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$

  ℓ 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 consist of single 1 at the $(\ell + j)^{\text{th}}$ decimal and all other decimals are 0's.



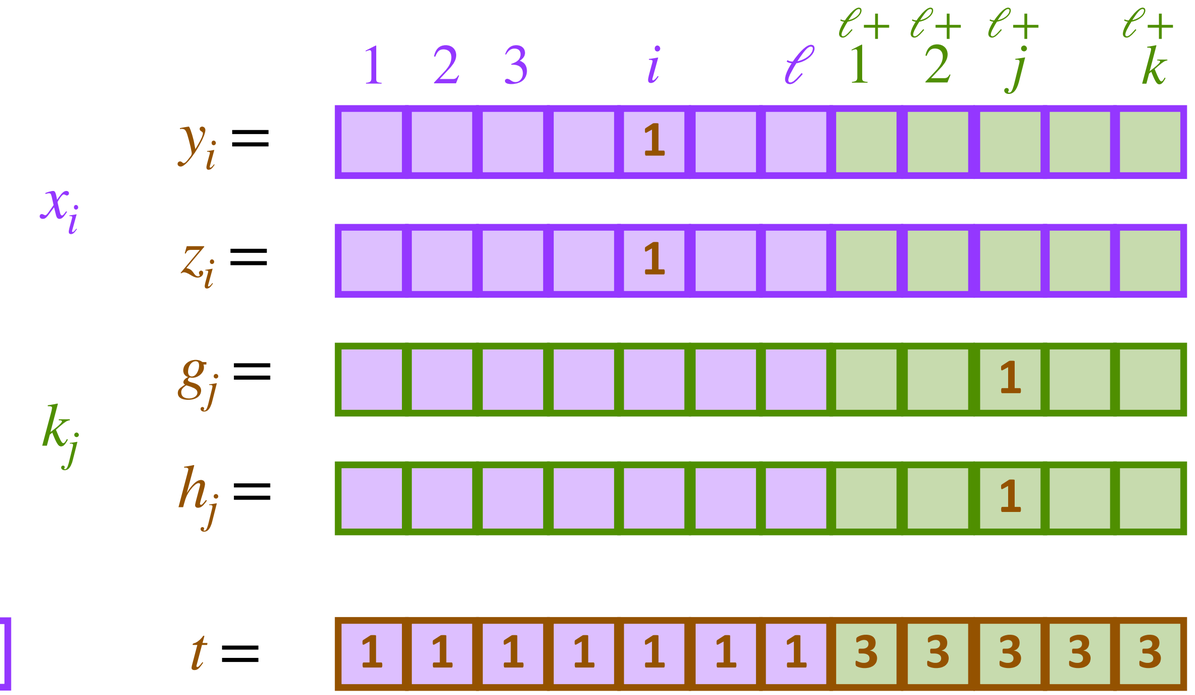
SUBSET-SUM

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

  ℓ 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

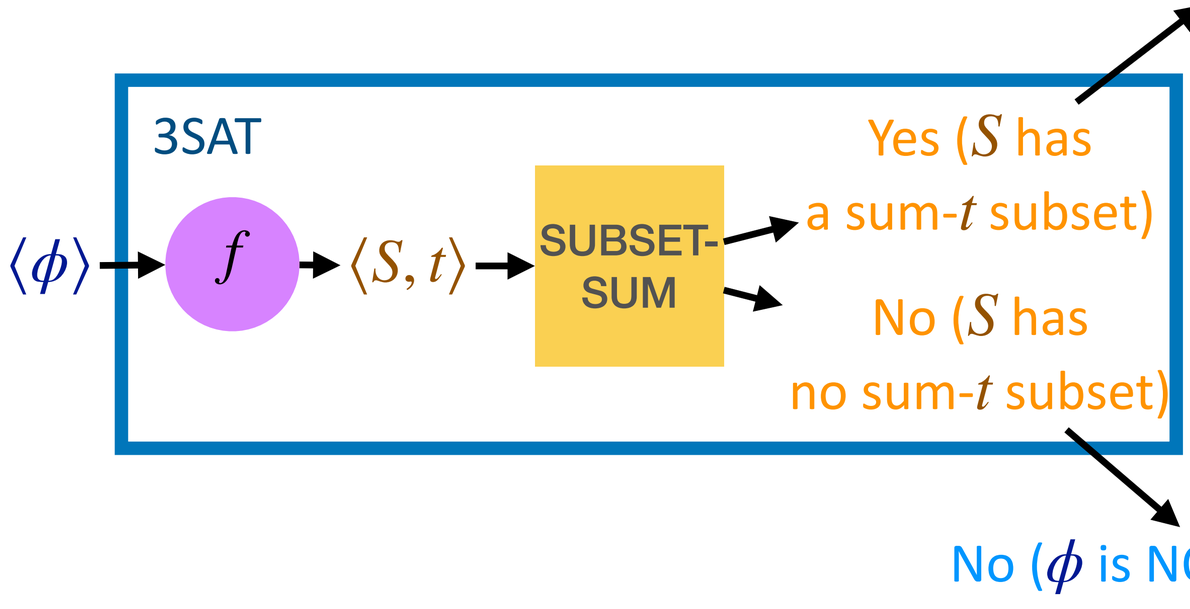
clause 1 clause 2 clause 3 clause 4

$$\overbrace{(x_1 \vee x_2 \vee x_3)}^{\text{clause 1}} \wedge \overbrace{(x_1 \vee \bar{x_1} \vee x_3)}^{\text{clause 2}} \wedge \overbrace{(x_2 \vee \bar{x_2} \vee \bar{x_3})}^{\text{clause 3}} \wedge \overbrace{(x_1 \vee x_1 \vee \bar{x_2})}^{\text{clause 4}}$$

	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	0	
$z_3 =$	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	0	
$z_3 =$	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

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

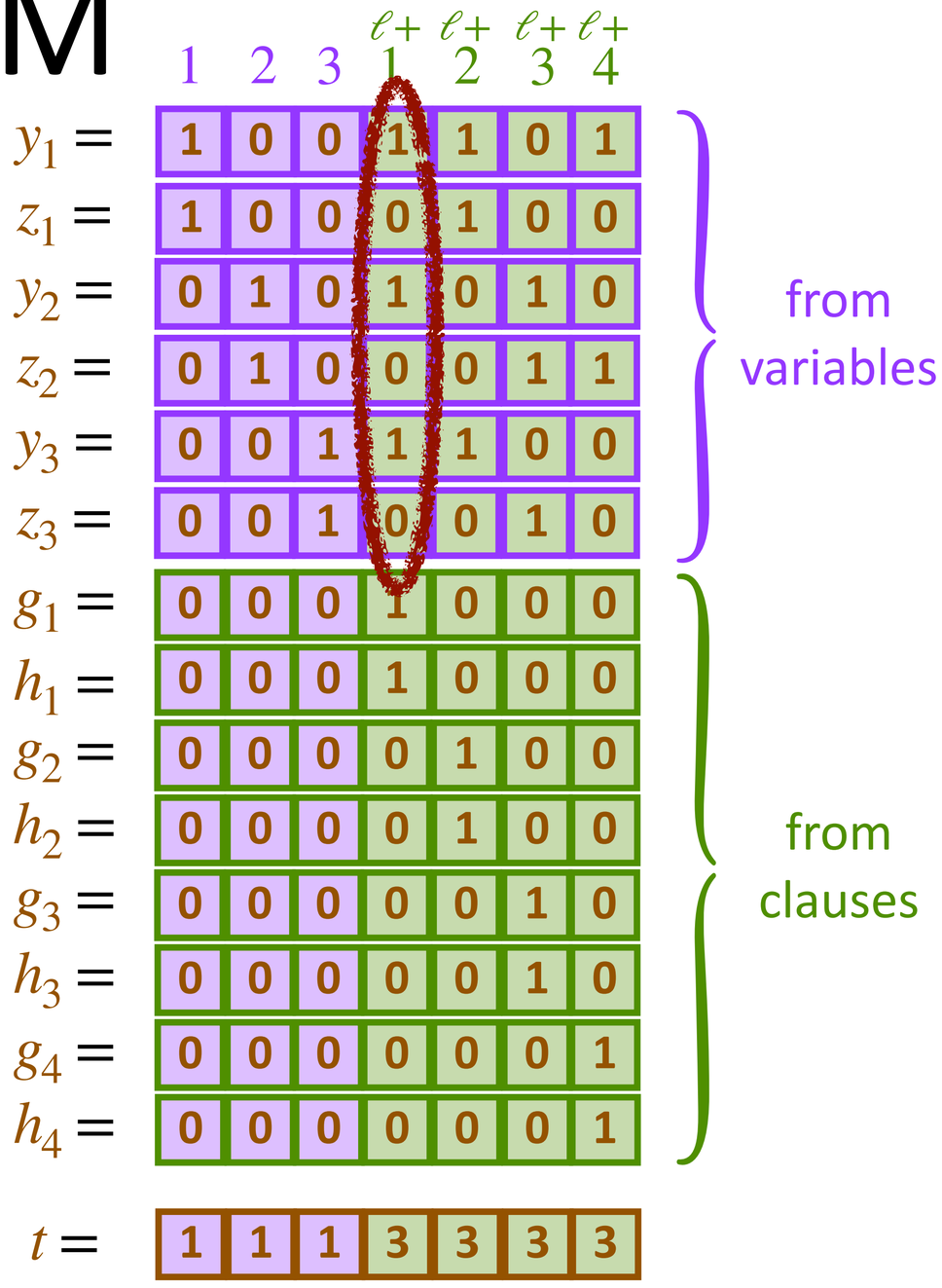
If there is a subset with sum t, a 1's of the first ℓ decimals must come from a y_i or z_i for some x_i.

	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	0	
$z_3 =$	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

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

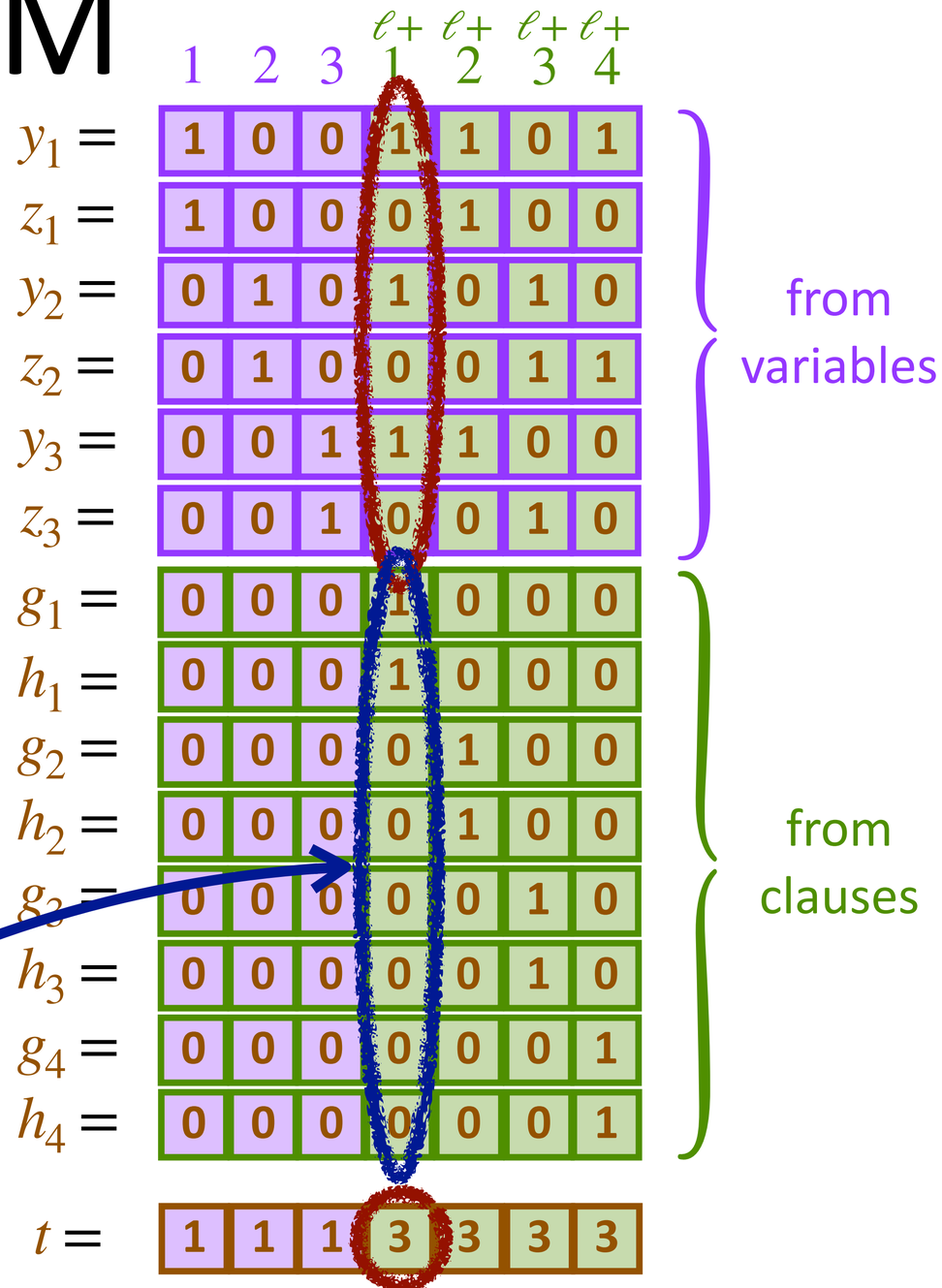
There are at most three 1's in each column representing the j^{th} clause.



SUBSET-SUM

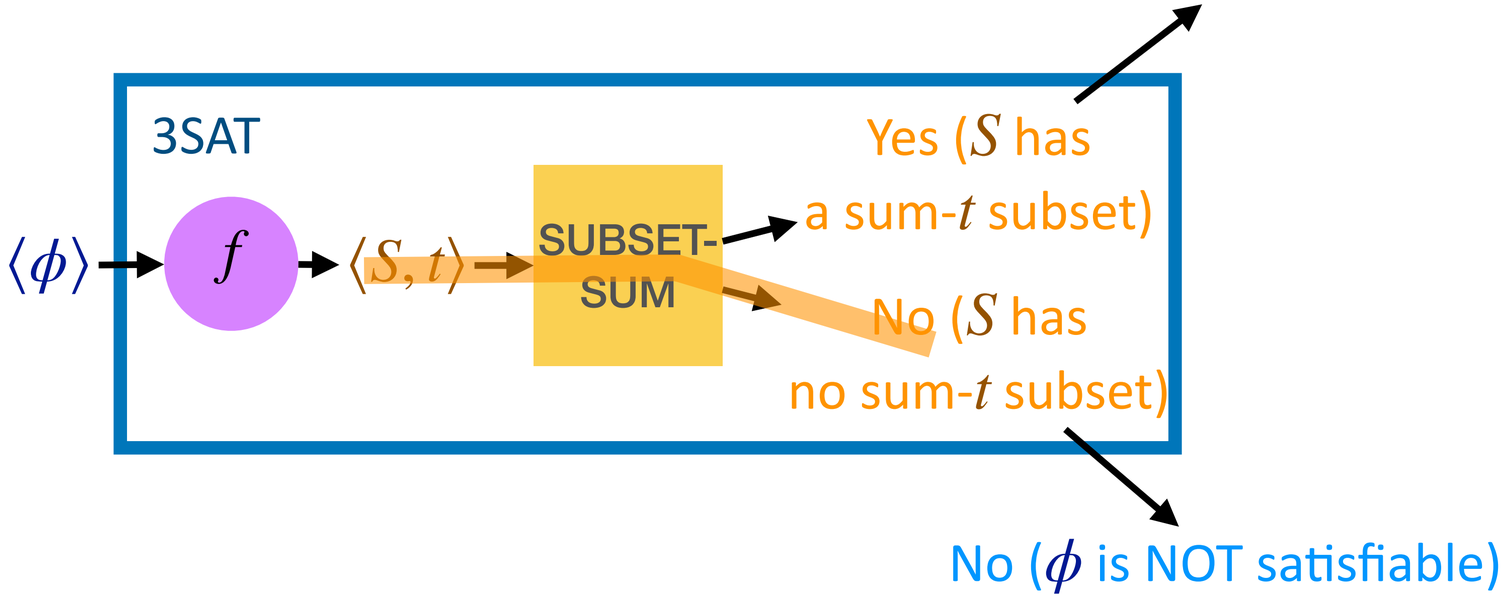
$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x_1} \vee x_3) \wedge (x_2 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_1 \vee x_1 \vee \bar{x_2})$

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



SUBSET-SUM

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



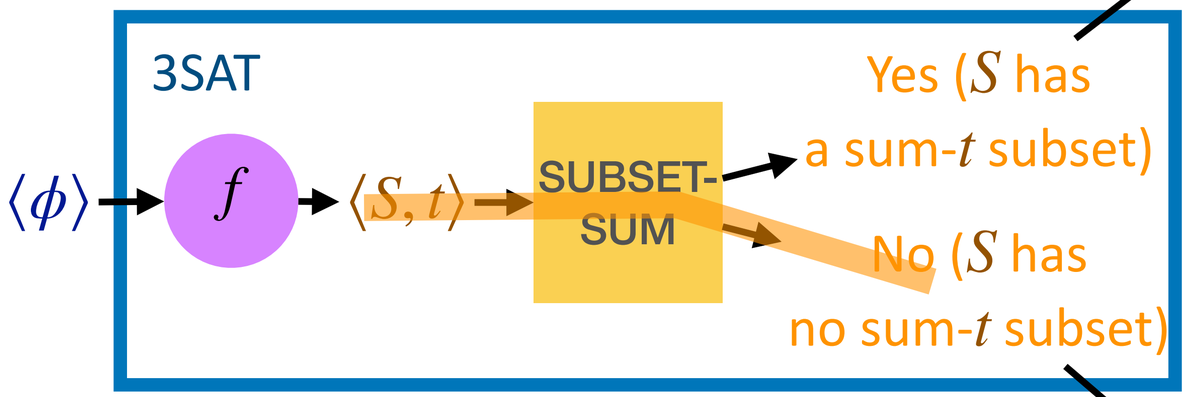
If ϕ is satisfiable, there exists an assignment ($x_1 = T$, $x_2 = F$, $\cdots$) such that every clause has at least one T.
If $x_i = T$, choose y_i as part of the subset. Otherwise, choose z_i.

	1	2	3	ℓ+ 1	ℓ+ 2	ℓ+ 3	ℓ+ 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	0	
$z_3 =$	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

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

Yes (ϕ is satisfiable)



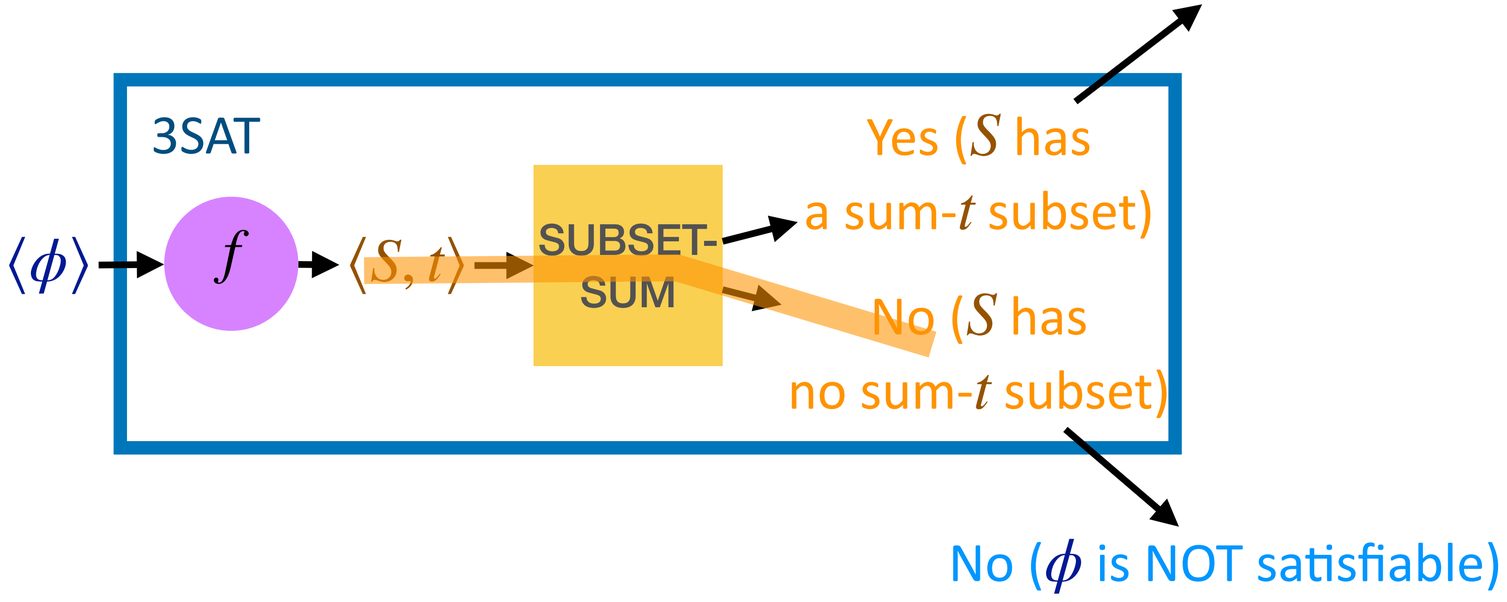
No (ϕ is NOT satisfiable)

If ϕ 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	ℓ+ 1	ℓ+ 2	ℓ+ 3	ℓ+ 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	0	
$z_3 =$	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

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

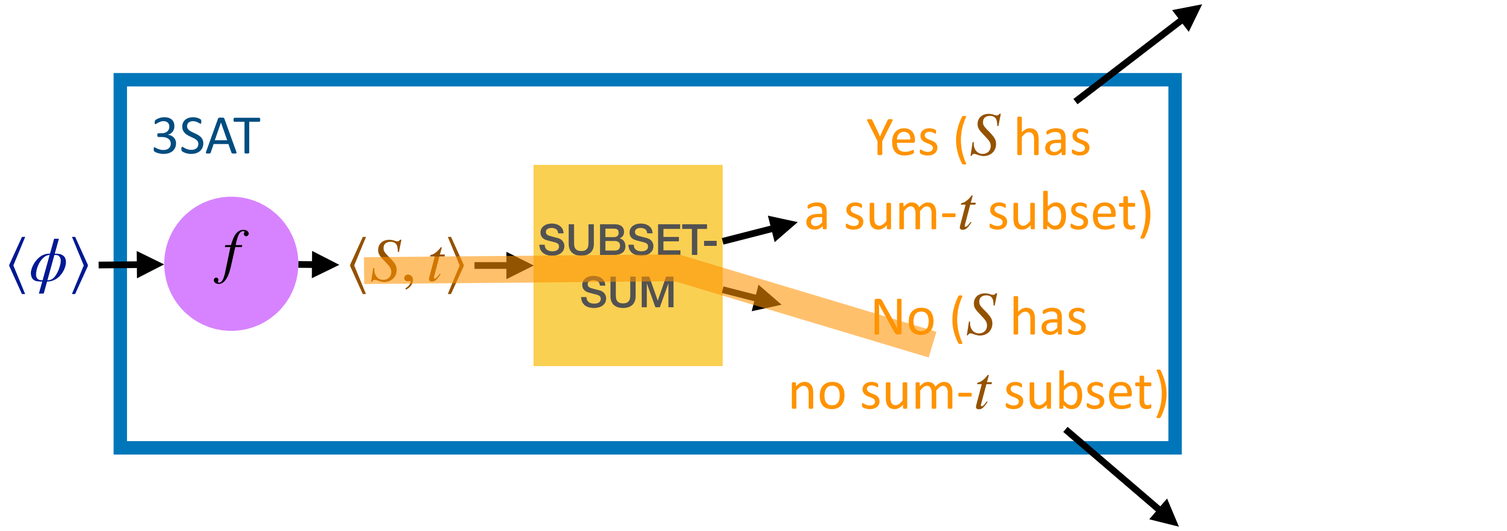


		1	2	3	$\ell+1$	$\ell+2$	$\ell+3$	$\ell+4$	
y_1	=	1	0	0	1	1	0	1	from variables
z_1	=	1	0	0	0	1	0	0	
y_2	=	0	1	0	1	0	1	0	
z_2	=	0	1	0	0	0	1	1	
y_3	=	0	0	1	1	1	0	0	
z_3	=	0	0	1	0	0	1	0	
g_1	=	0	0	0	1	0	0	0	from clauses
h_1	=	0	0	0	1	0	0	0	
g_2	=	0	0	0	0	1	0	0	
h_2	=	0	0	0	0	1	0	0	
g_3	=	0	0	0	0	0	1	0	
h_3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

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

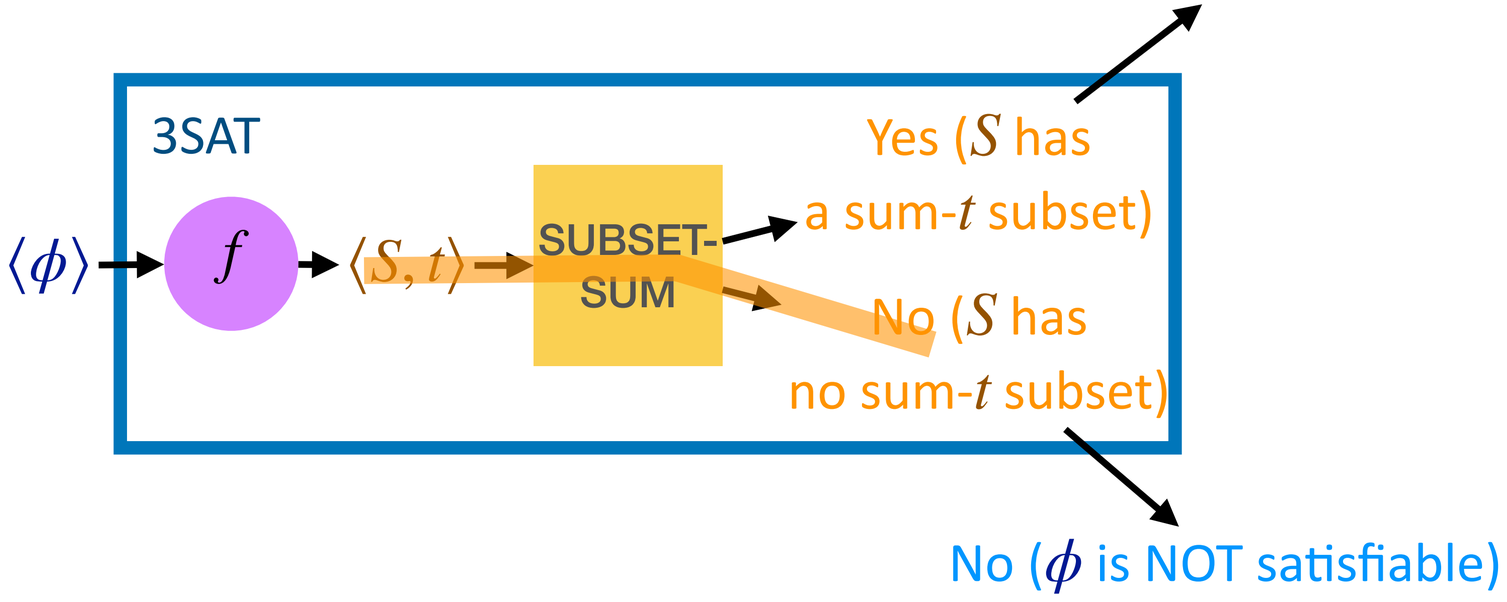


	1	2	3	$\ell+1$	$\ell+2$	$\ell+3$	$\ell+4$	
$y_1 =$	1	0	0	1	1	0	1	from variables
$z_1 =$	1	0	0	0	1	0	0	
$y_2 =$	0	1	0	1	0	1	0	
$z_2 =$	0	1	0	0	0	1	1	
$y_3 =$	0	0	1	1	1	0	0	
$z_3 =$	0	0	1	0	0	1	0	
$g_1 =$	0	0	0	1	0	0	0	from clauses
$h_1 =$	0	0	0	1	0	0	0	
$g_2 =$	0	0	0	0	1	0	0	
$h_2 =$	0	0	0	0	1	0	0	
$g_3 =$	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 ϕ 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	ℓ+ 1	ℓ+ 2	ℓ+ 3	ℓ+ 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	0	
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	

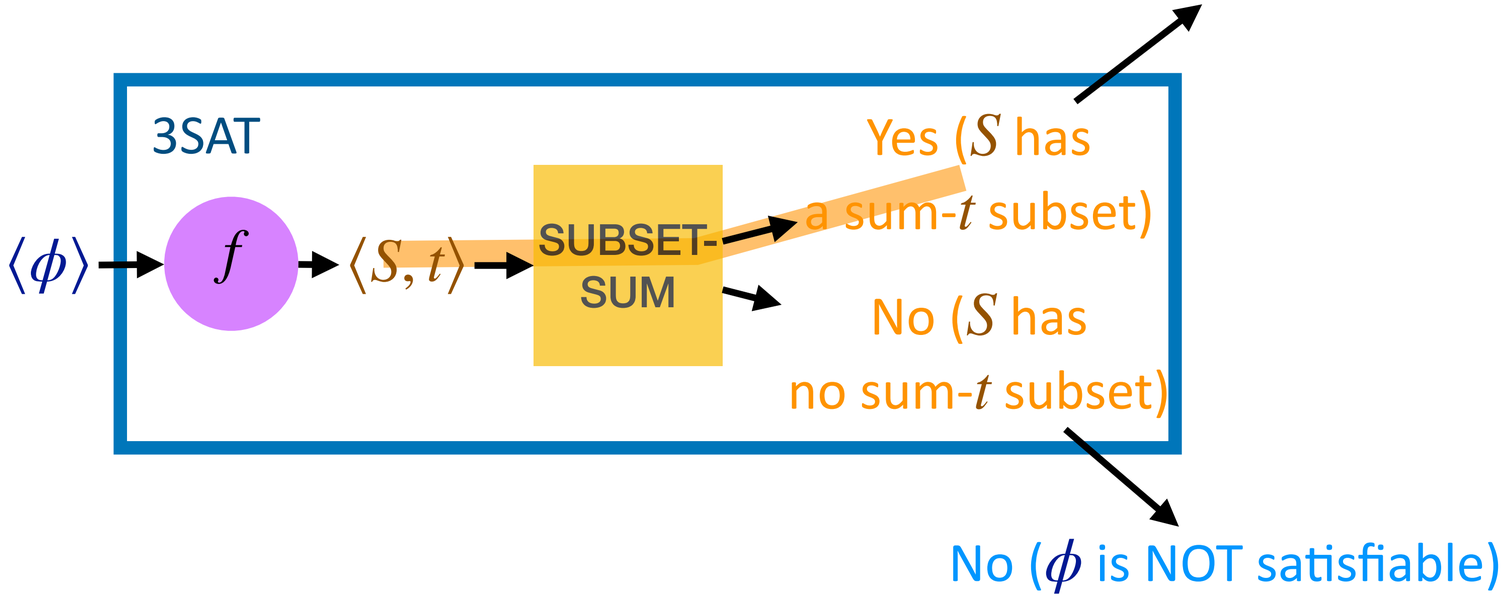
Since the assignment is feasible, each x_i is either T or F

⇒ for any i, either y_i or z_i is chosen

⇒ for each of the first ℓ decimals, the sum is 1

SUBSET-SUM

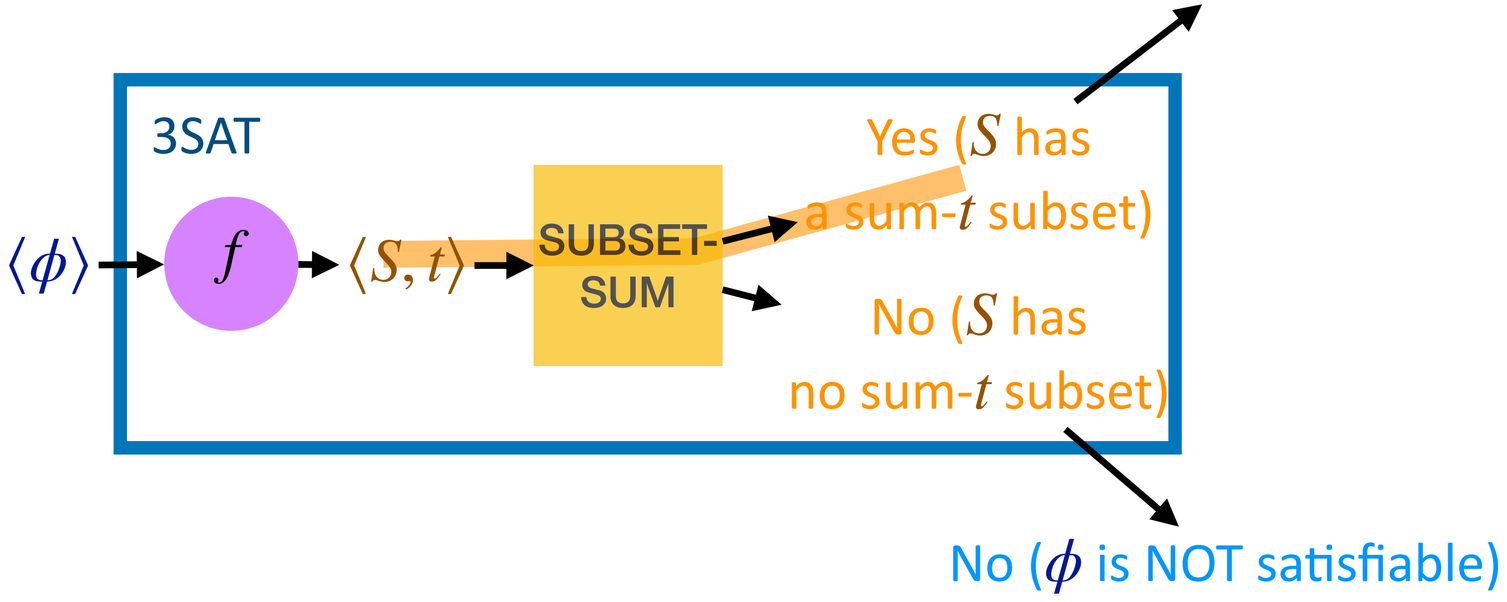
$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_1 \vee x_3) \wedge (x_2 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_1 \vee \bar{x}_2)$



	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	0	
$z_3 =$	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

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

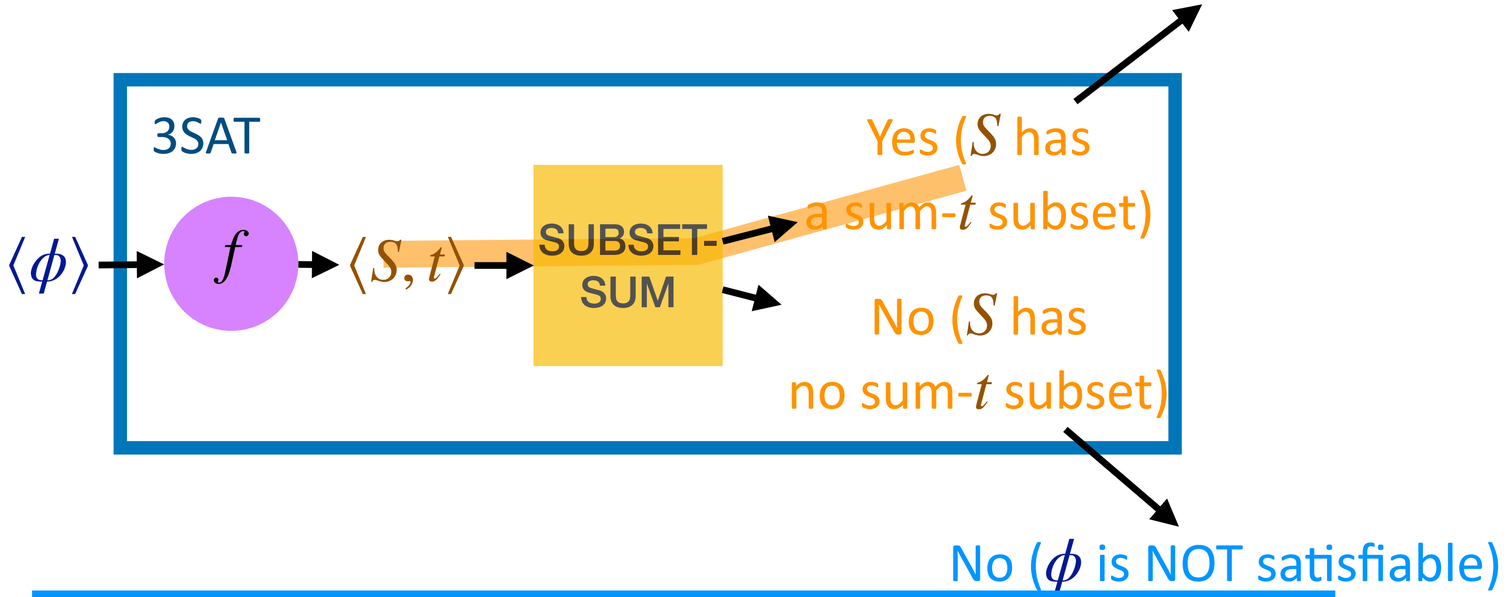


	1	2	3	$\ell+1$	$\ell+2$	$\ell+3$	$\ell+4$	
$y_1 =$	1	0	0	1	1	0	1	from variables
$z_1 =$	1	0	0	0	1	0	0	
$y_2 =$	0	1	0	1	0	1	0	
$z_2 =$	0	1	0	0	0	1	1	
$y_3 =$	0	0	1	1	1	0	0	
$z_3 =$	0	0	1	0	0	1	0	
$g_1 =$	0	0	0	1	0	0	0	from clauses
$h_1 =$	0	0	0	1	0	0	0	
$g_2 =$	0	0	0	0	1	0	0	
$h_2 =$	0	0	0	0	1	0	0	
$g_3 =$	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	ℓ+ 1	ℓ+ 2	ℓ+ 3	ℓ+ 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	0	
$z_3 =$	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 ϕ: 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 ϕ we construct an instance of the SUBSET-SUM problem that contains a subcollection summing to the target k if and only if ϕ 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 ϕ 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 ϕ, 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 ϕ is satisfiable if and only if some subset of S sum to t.

Suppose ϕ 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 ϕ. First we observe that no carry into the next decimal is needed since all the decimals 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 coming from g_i or h_i, there is at least one 1 coming from some y_i or z_i. Hence this assignment satisfies ϕ.

What Happened

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

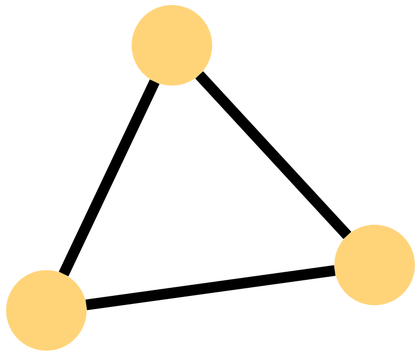
- Each clause in ϕ is *TRUE* if and only if the corresponding (chain of) clauses in ϕ' are all *TRUE*

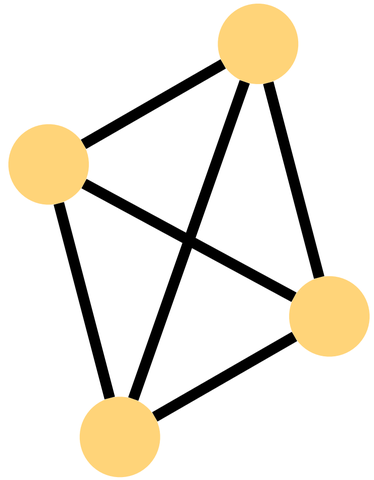
Outline

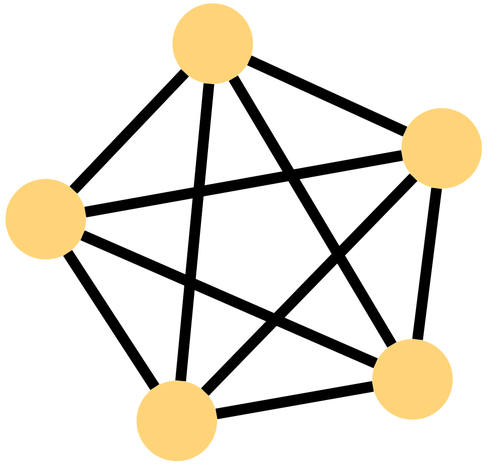
- NP-Completeness
  - NP-hardness: Polynomial time reduction
    - CNF-SAT $\leq_p$ 3SAT
    - 3SAT $\leq_p$ SUBSET-SUM
    - 3SAT $\leq_p$ CLIQUE
    - PARTITION $\leq_p$ BIN-PACKING
- Cook-Leven Theorem: SAT is NP-complete

CLIQUE

- Clique: a graph in which every pair of vertices are adjacent

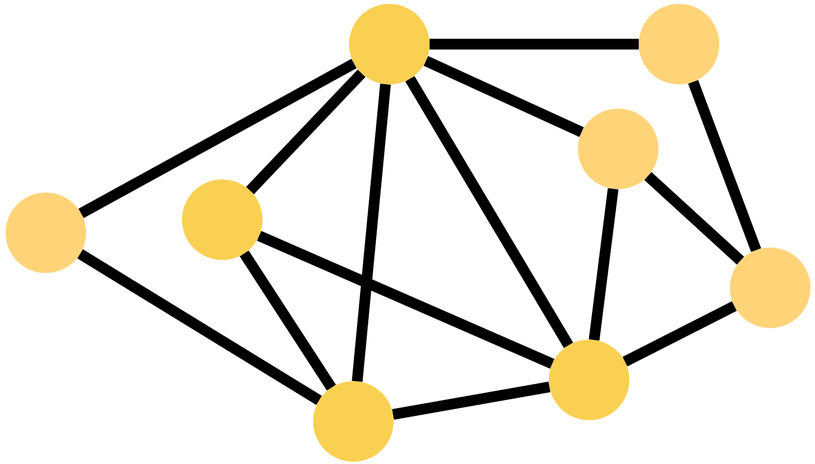






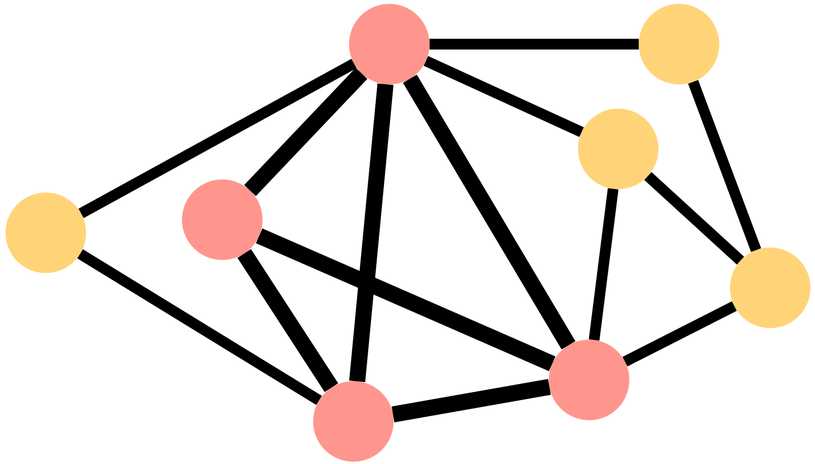
CLIQUE

- Maximum clique problem: Given a graph G, what is the size of the maximum clique in G?



CLIQUE

- Maximum clique problem: Given a graph G, what is the size of the maximum clique in G?



CLIQUE

- Maximum clique problem: Given a graph G, what is the size of the maximum clique in G?

- Decision version?

CLIQUE

- Maximum clique problem: Given a graph G, what is the size of the maximum clique in G?

- Decision version: Given a graph G, is there a clique of size at least k in G?
  - An instance of CLIQUE is $\langle G, k \rangle$

New parameter!

CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

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

- CLIQUE $= \{\langle G, k\rangle \mid G$ has a clique of size at least $k\}$

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

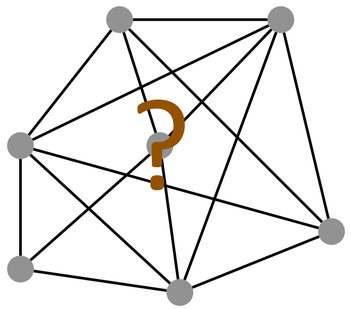
CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, k\rangle \mid G$ has a clique of size at least $k\}$

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



k = ?

CLIQUE

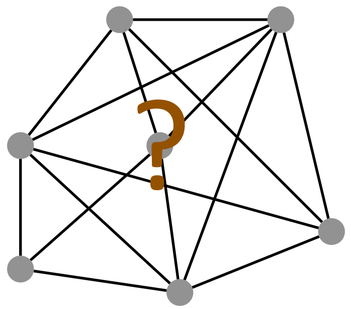
- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, k\rangle \mid G$ has a clique of size at least $k\}$

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

satisfiable



k = ?

⇕

There is a k-clique in G

CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, k\rangle \mid G$ has a clique of size at least $k\}$

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

For each clause C_i containing three literals $l_{i_1}, l_{i_2}, l_{i_3}$,

there are three vertices $v_{\ell_{i_1}}$, $v_{\ell_{i_2}}$, and $v_{\ell_{i_3}}$ in V.

CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, k\rangle \mid G$ has a clique of size at least $k\}$

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

x_1 x_1 x_2

For each clause C_i containing three literals $l_{i_1}, l_{i_2}, l_{i_3}$,

there are three vertices $v_{\ell_{i_1}}$, $v_{\ell_{i_2}}$, and $v_{\ell_{i_3}}$ in V.

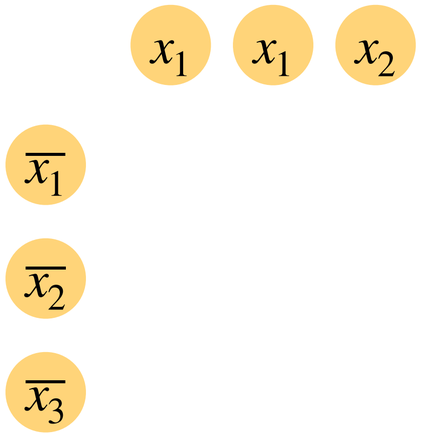
CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, k\rangle \mid G$ has a clique of size at least $k\}$

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



For each clause C_i containing three literals $l_{i_1}, l_{i_2}, l_{i_3}$,

there are three vertices $v_{\ell_{i_1}}$, $v_{\ell_{i_2}}$, and $v_{\ell_{i_3}}$ in V.

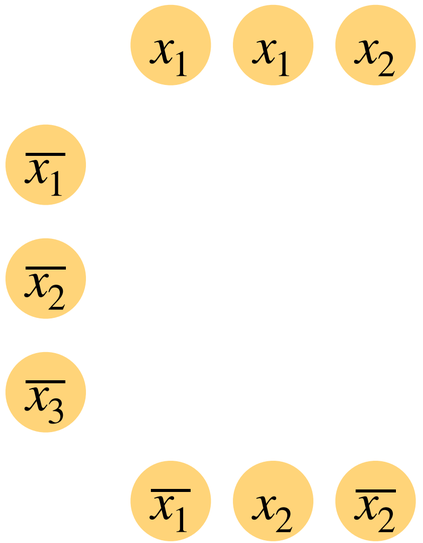
CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, k\rangle \mid G$ has a clique of size at least $k\}$

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



For each clause C_i containing three literals $l_{i_1}, l_{i_2}, l_{i_3}$,

there are three vertices $v_{\ell_{i_1}}$, $v_{\ell_{i_2}}$, and $v_{\ell_{i_3}}$ in V.

CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

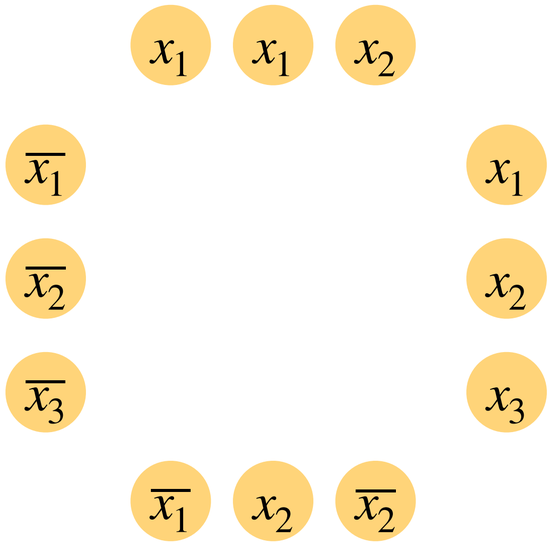
<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, k\rangle \mid G$ has a clique of size at least $k\}$

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

For each clause C_i containing three literals $l_{i_1}, l_{i_2}, l_{i_3}$,

there are three vertices $v_{\ell_{i_1}}$, $v_{\ell_{i_2}}$, and $v_{\ell_{i_3}}$ in V.



CLIQUE

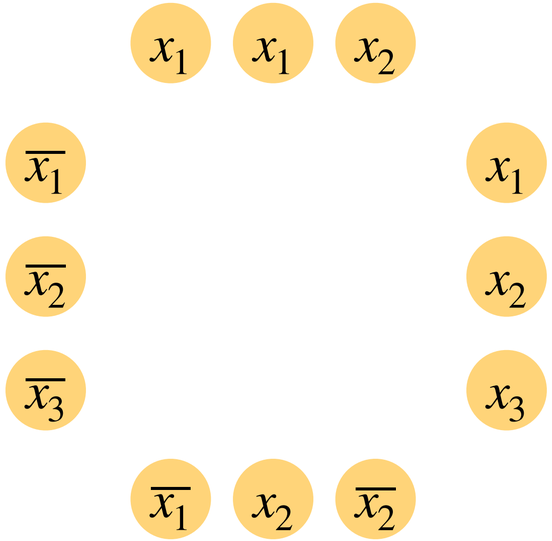
- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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

If there are m clauses in ϕ, let k be m



CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

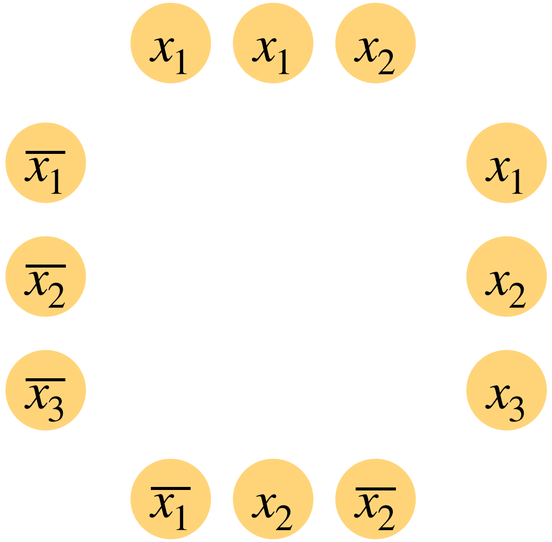
<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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

There is an edge (l_x, l_y) in E if and only if

- The two vertices l_x and l_y come from different clauses, and
- The corresponding literals of l_x and l_y are not the negation to each other.



CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

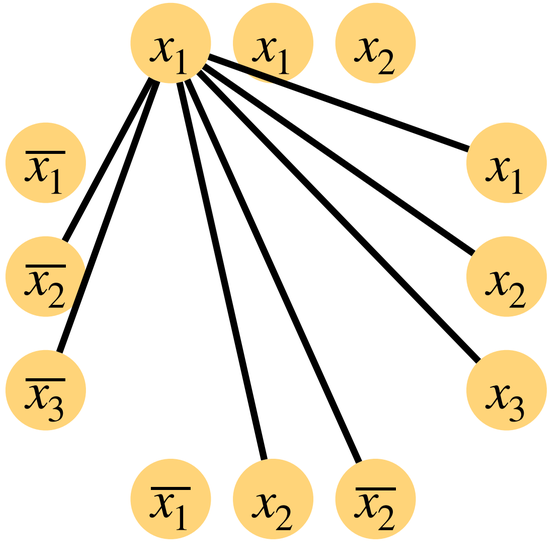
<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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

There is an edge (l_x, l_y) in E if and only if

- The two vertices l_x and l_y come from different clauses, and
- The corresponding literals of l_x and l_y are not the negation to each other.



CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

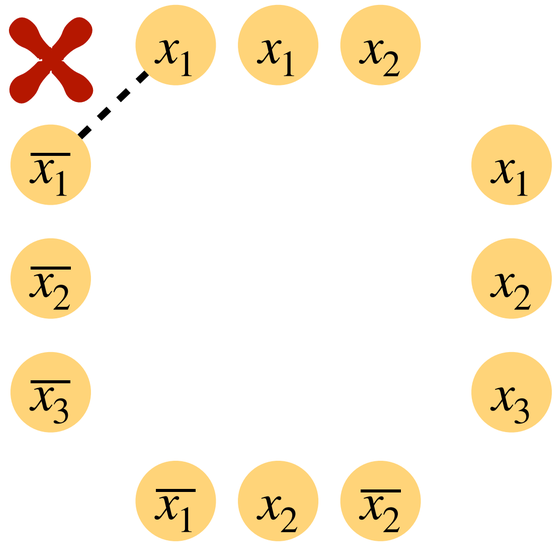
<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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

There is an edge (l_x, l_y) in E if and only if

- The two vertices l_x and l_y come from different clauses, and
- **The corresponding literals of l_x and l_y are not the negation to each other.**



CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

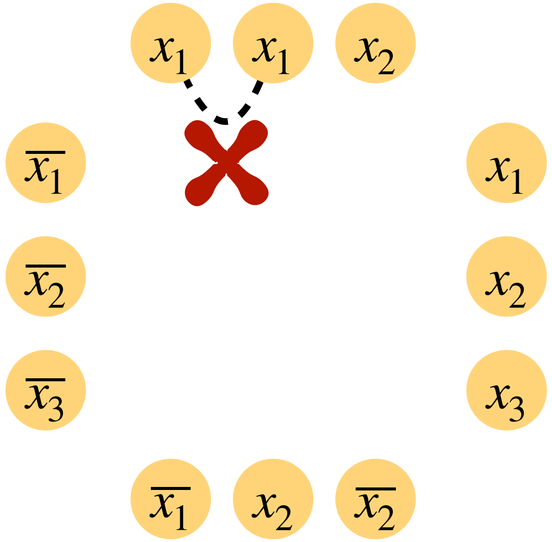
<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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

There is an edge (l_x, l_y) in E if and only if

- **The two vertices l_x and l_y come from different clauses, and**
- The corresponding literals of l_x and l_y are not the negation to each other.



CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

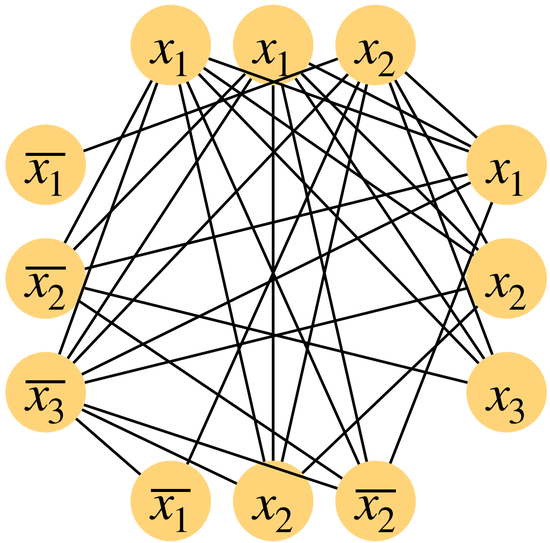
<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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

There is an edge (l_x, l_y) in E if and only if

- The two vertices l_x and l_y come from different clauses, and
- The corresponding literals of l_x and l_y are not the negation to each other.



CLIQUE

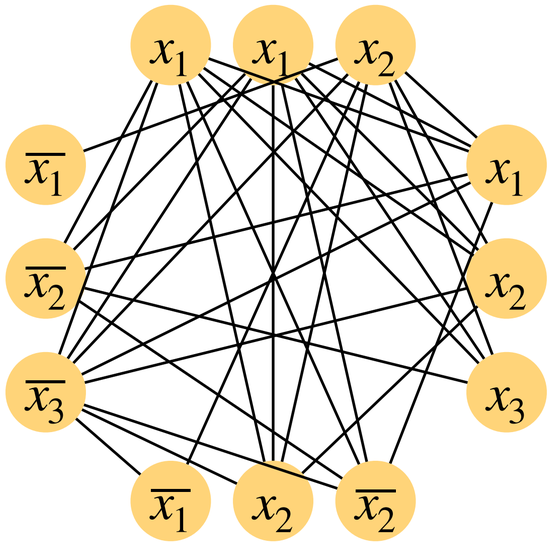
- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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

satisfiable ⇒ There is a truth assignment such that there is at least one TRUE in each clause



CLIQUE

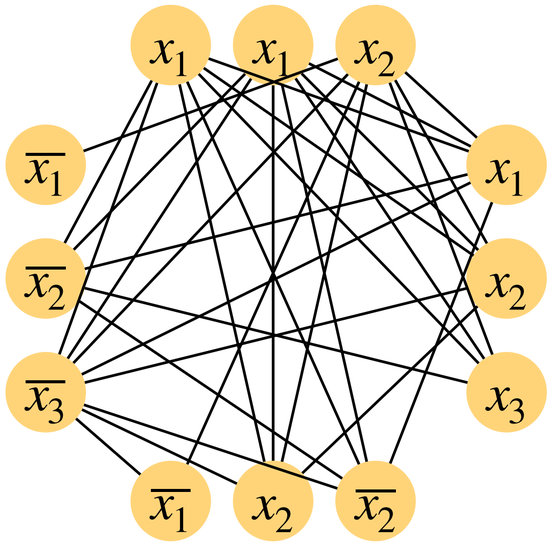
- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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

satisfiable ⇒ There is a truth assignment such that there is at least one TRUE in each clause



CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

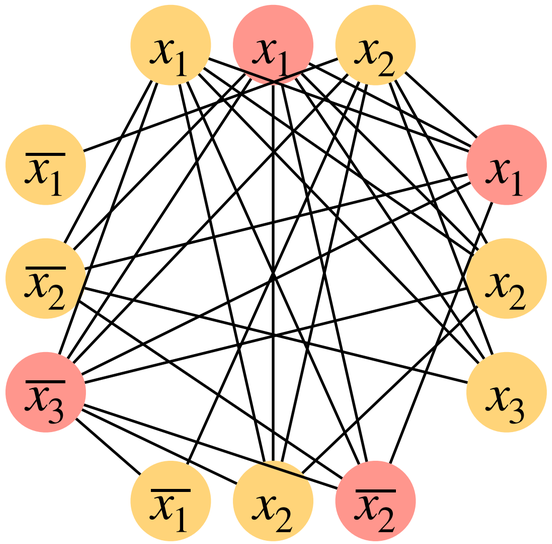
<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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

satisfiable ⇒ There is a truth assignment such that there is at least one TRUE in each clause

Consult the satisfying assignment to construct a solution to CLIQUE



CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

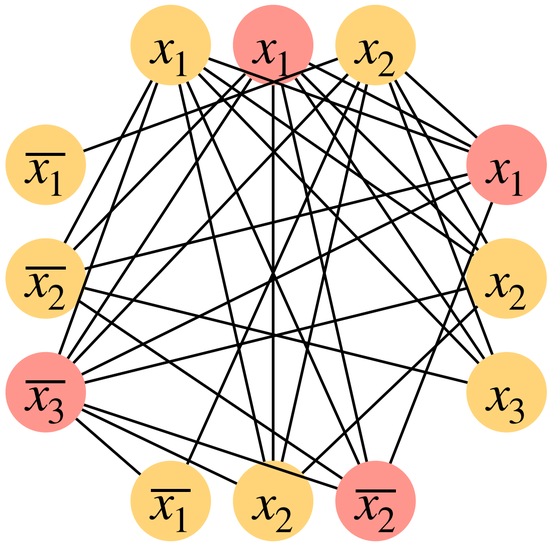
- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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

satisfiable ⇒ There is a truth assignment such that there is at least one TRUE in each clause

There is an edge between each pair of m corresponding vertices in G

(They are **not in the same clause**, and **can be TRUE at the same time**)



CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

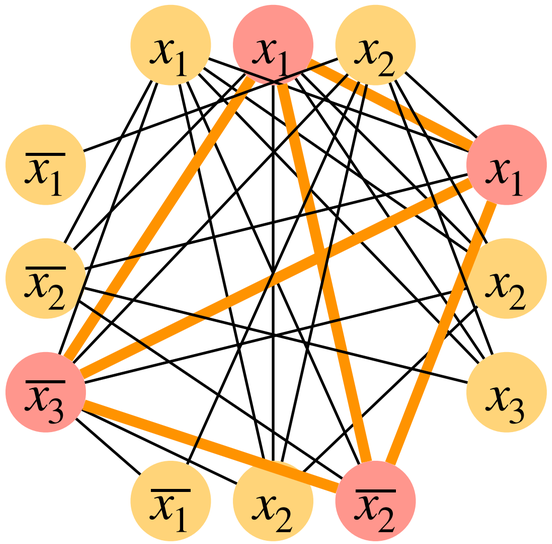
$$(x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_2}) \wedge (x_1 \vee x_2 \vee x_3)$$

satisfiable ⇒ There is a truth assignment such that there is at least one TRUE in each clause

⇓

There is an edge between each pair of m corresponding vertices in G

(They are **not in the same clause**, and **can be TRUE at the same time**)



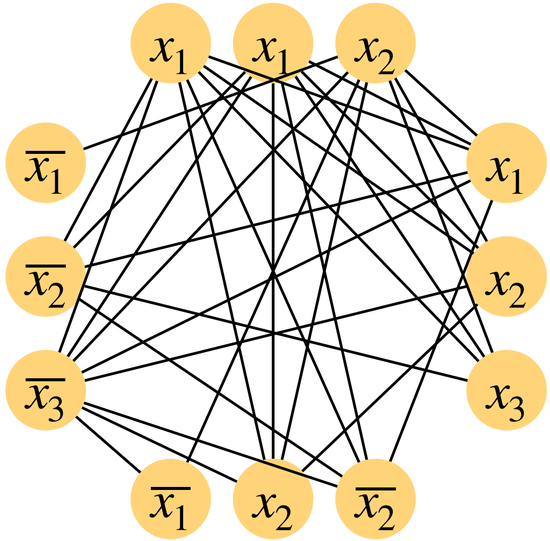
CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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



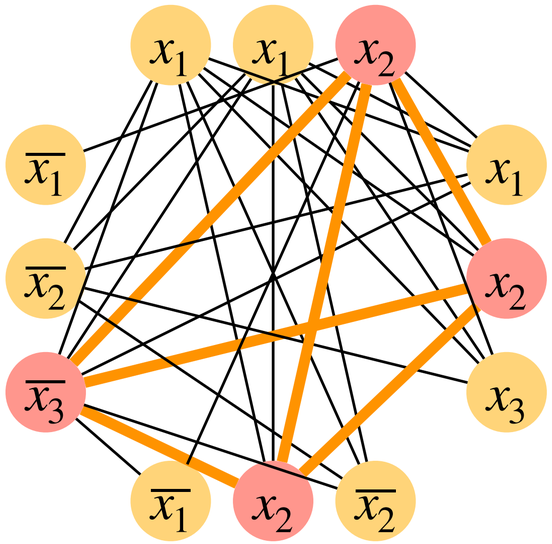
CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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



$\langle G, m\rangle$ is a yes-instance ⇒ there is a m-clique in G

CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

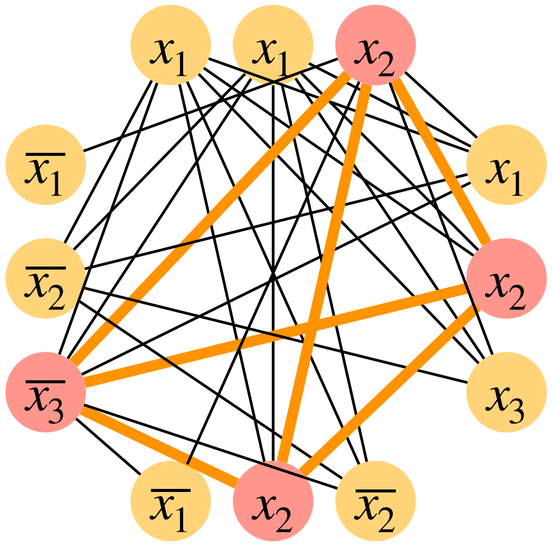
- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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

Consult the m-clique to construct a truth-assignment to 3SAT:
Set the corresponding variables as TRUE

$\Uparrow$

$\langle G, m\rangle$ is a yes-instance $\Rightarrow$ there is a m-clique in G



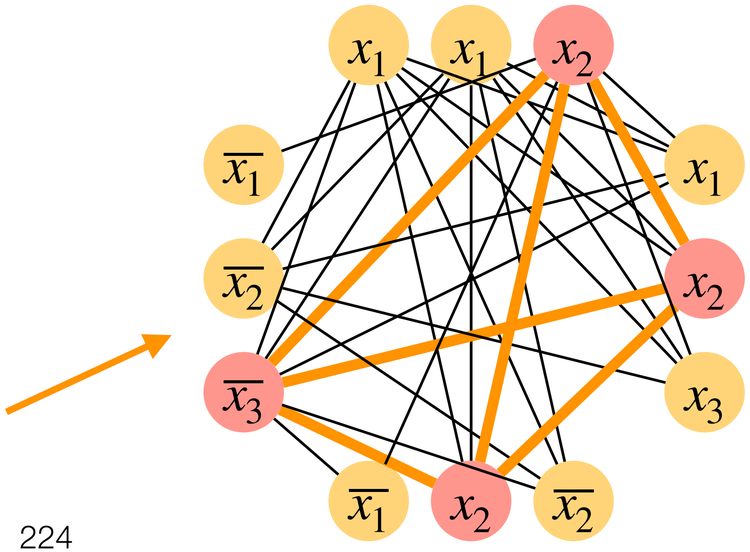
CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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



There is an edge between each pair of the m corresponding vertices in G

⇒ By our construction, they are from different clauses and can be TRUE at the same time

CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

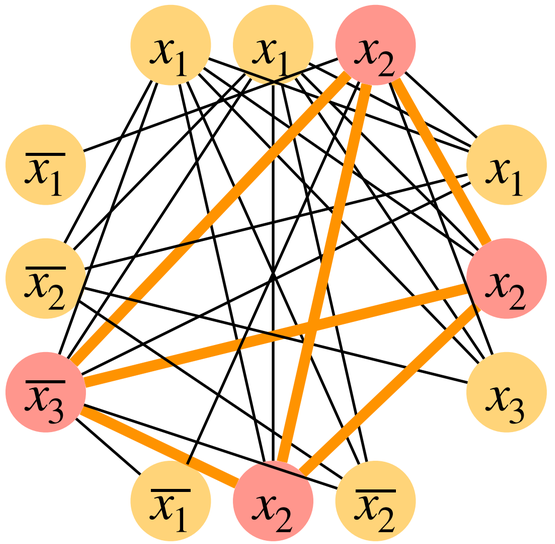
$$(x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_2}) \wedge (x_1 \vee x_2 \vee x_3)$$

The constructed assignment is satisfying since there is at least one TRUE in each clause

$\Uparrow$

There is an edge between each pair of the m corresponding vertices in G

$\Rightarrow$ By our construction, they are from different clauses and can be TRUE at the same time



CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof> Polynomial-time reduction from 3SAT

For any instance of 3SAT, $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, we generate an instance of CLIQUE, $G = (V, E)$ and k, as follows:

For each clause C_i containing three literals $l_{i_1}, l_{i_2}, l_{i_3}$, there are three vertices in V.

CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof (cont.)> For any pair of vertices l_x, l_y in V, there is an edge (l_x, l_y) in E if and only if

- The two vertices l_x and l_y come from different clauses, and
- The corresponding literals of l_x and l_y are not the negation to each other.

Finally, we let k equals to m, the number of clauses in ϕ.

The construction can be done in polynomial time since $|V| = 3m$ and there are $O(m^2)$ edges, where each of the edges needs constant time to check.

CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

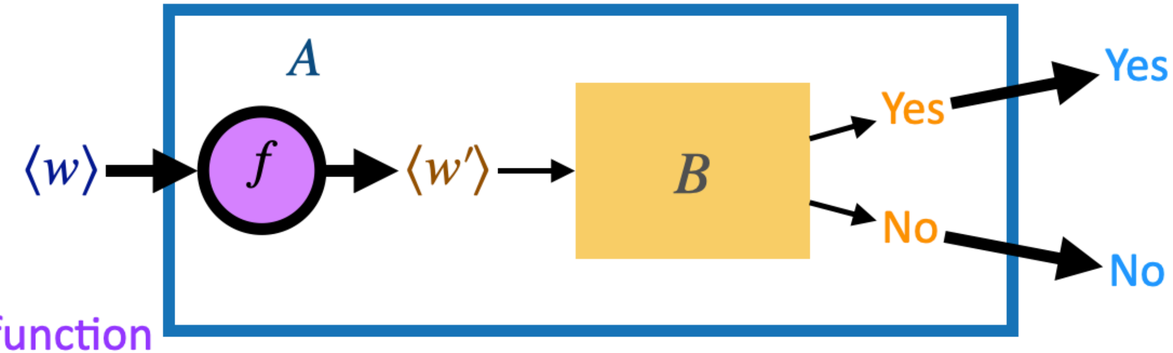
<Proof (cont.)> Now we show that the reduction works by showing that there is a satisfying assignment to ϕ if and only there is a k-clique in G.

Suppose that ϕ has a satisfying assignment, we construct a k-clique by selecting one of the vertices which are corresponding to a literal with "TRUE" value from each of the clauses. Since ϕ is satisfiable, there must be one of such a literal in every clause. As the satisfying assignment is feasible, every variable is assigned to either TRUE or FALSE but not both. Hence, there must be an edge between two vertices picked from different clauses. Therefore, the picked vertices form a k-clique.

CLIQUE

- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof (cont.)> Suppose that G has a clique V' of size k. No edges in G connect vertices in the same clause, so V' contains exactly one vertex form each of the k clauses. We assign value TRUE to the corresponding literal. It is a feasible assignment since there is no edges between literals corresponding to x and $\bar{x}$ for each variable x. Hence, each clause has one literal which is assigned TRUE and the formula ϕ is satisfied.



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 no-instance $w' \notin B$, the corresponding instance w is also a no-instance of A

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

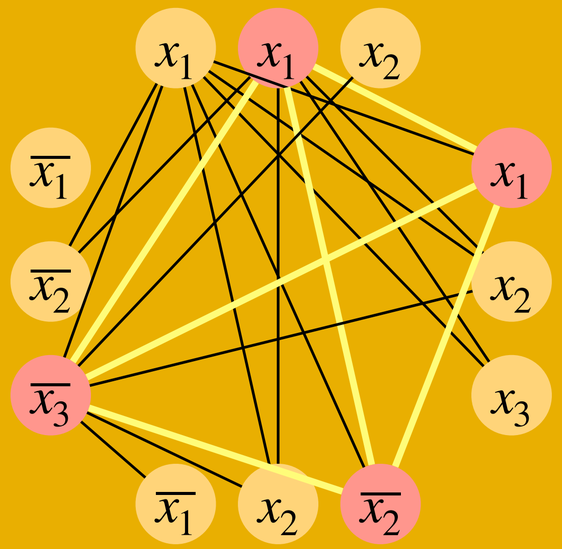
satisfiable ⇒ There is a truth assignment such that there is at least one TRUE in each clause

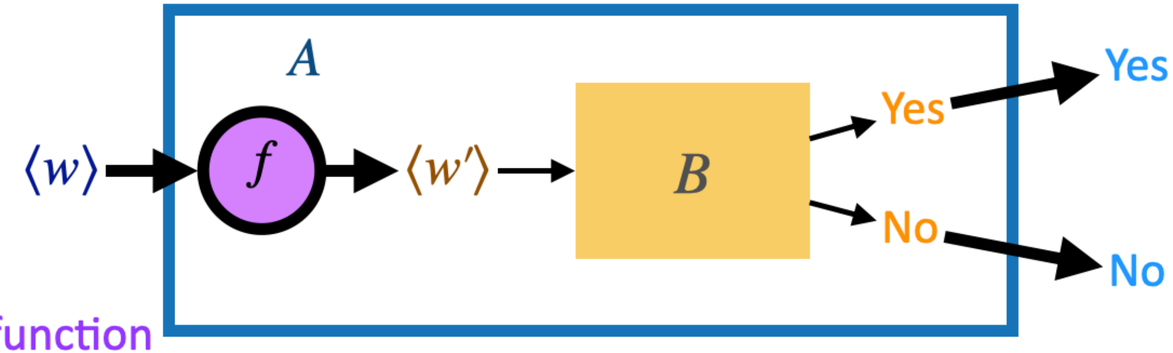
⇓

Put the corresponding vertices in the clique

There is an edge between each pair of m corresponding vertices in G

(Because they are not in the same clause, and can be TRUE at the same time)





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 no-instance $w' \notin B$, the corresponding instance w is also a no-instance of A

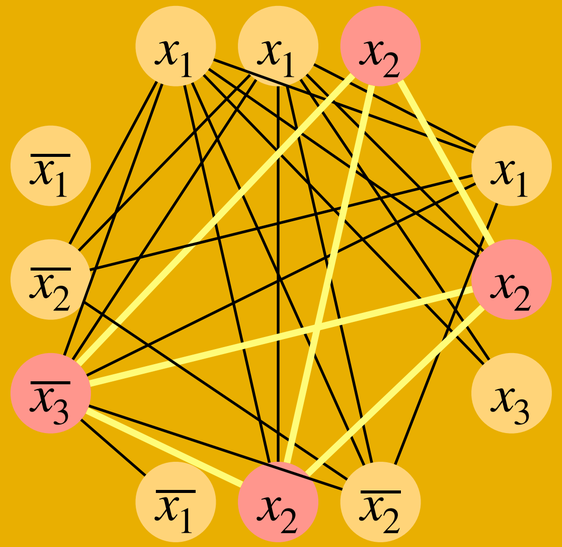
$$(x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_2}) \wedge (x_1 \vee x_2 \vee x_3)$$

Set the corresponding variables as TRUE

There is at least one TRUE in each clause

$\Uparrow$

If there is a m-clique in G



Outline

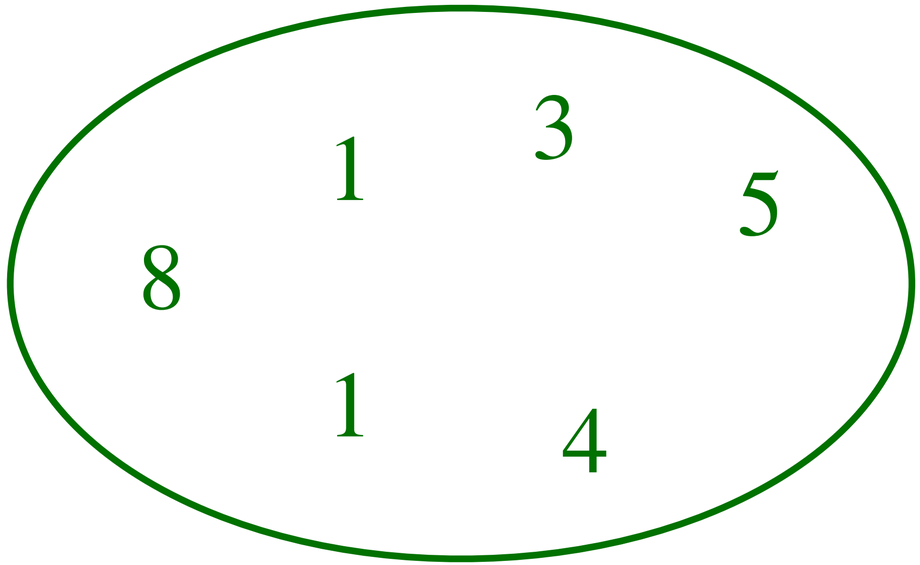
- NP-Completeness
  - NP-hardness: Polynomial time reduction
    - CNF-SAT $\leq_p$ 3SAT
    - 3SAT $\leq_p$ SUBSET-SUM
    - 3SAT $\leq_p$ CLIQUE
    - PARTITION $\leq_p$ BIN-PACKING
- Cook-Leven Theorem: SAT is NP-complete

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 $\Sigma_{y_i \in T} y_i = \Sigma_{z_i \in S \setminus T} z_i\}$

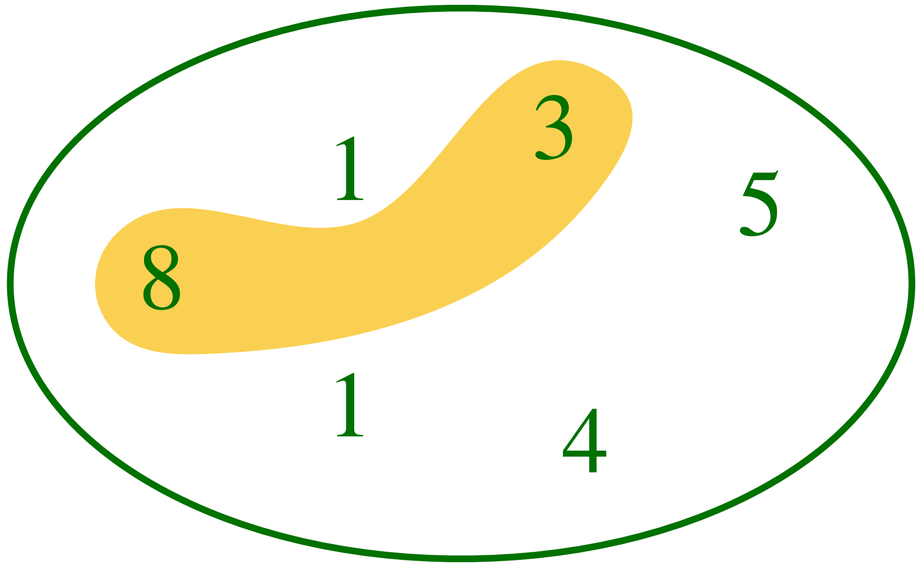
PARTITION

- 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}



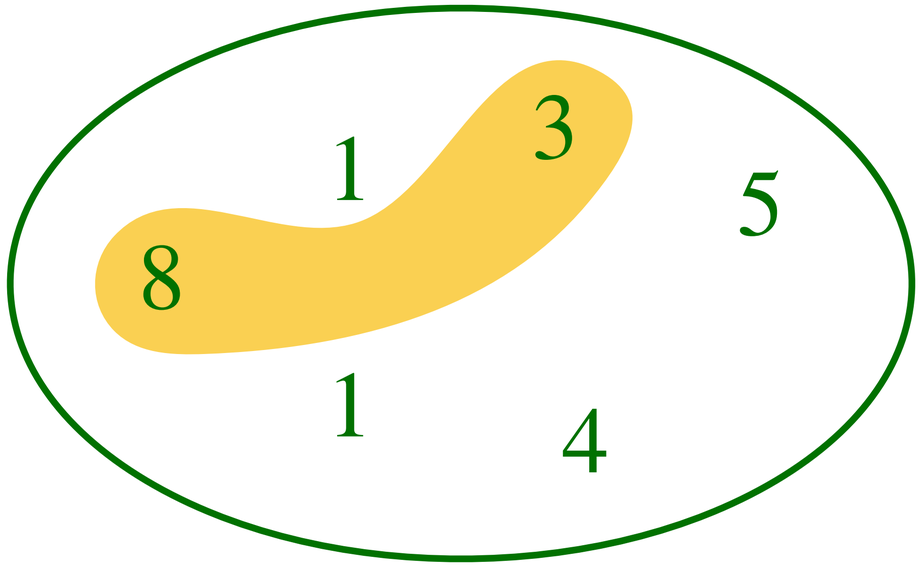
PARTITION

- 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**}



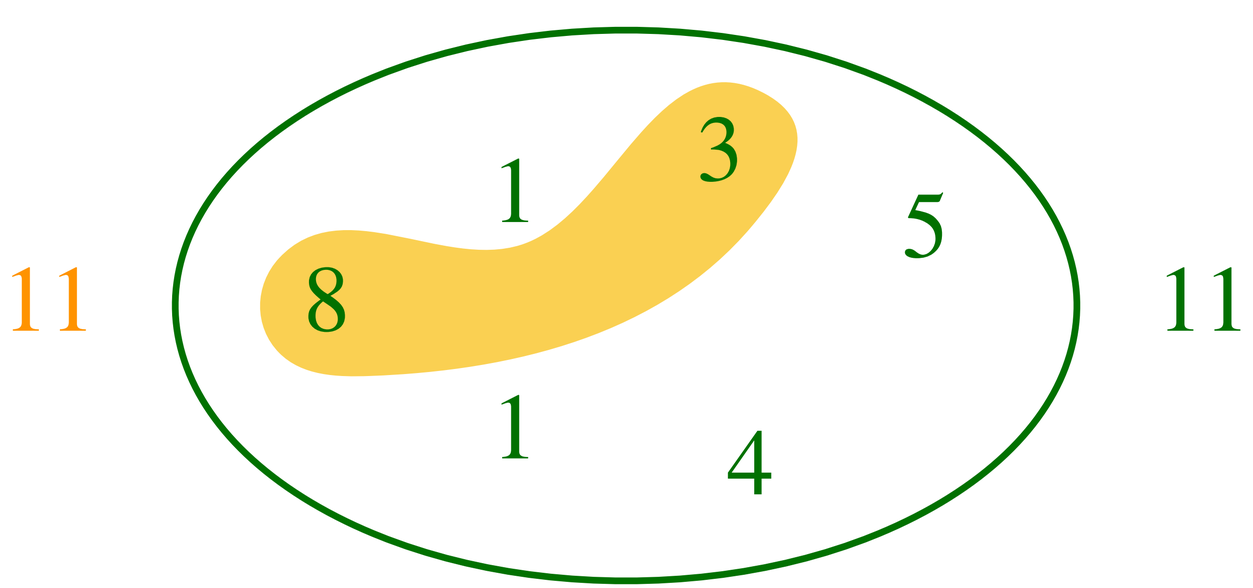
PARTITION

- 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}



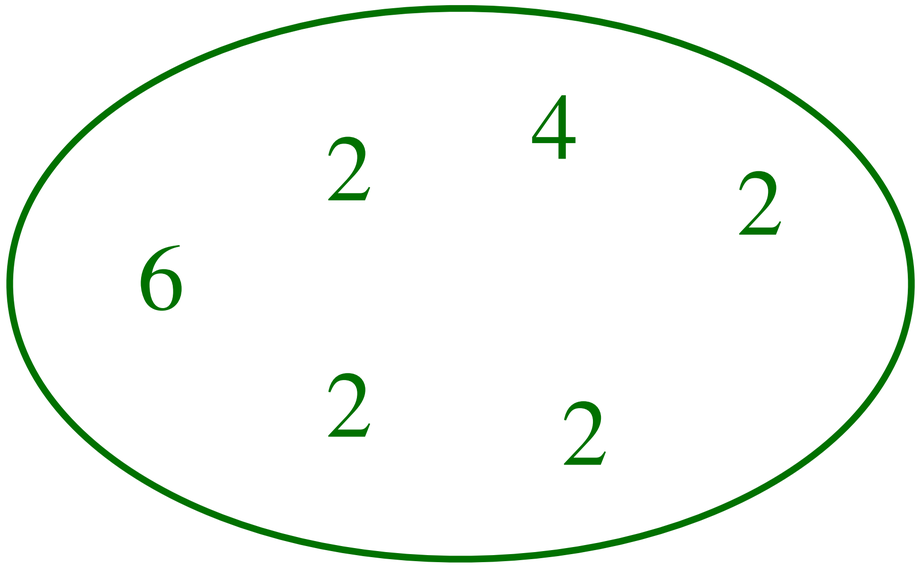
PARTITION

- 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

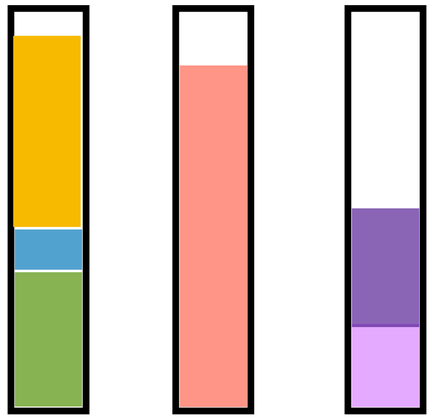


PARTITION

- 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

BIN-PACKING

- Given a finite set $U = \{u_1, u_2, \cdots, u_n\}$ of items and a rational size $s(u_i) \in [0,1]$ for each item $u_i \in U$, find a partition of U into disjoint subsets $U_1, U_2, \cdots, U_k$ such that the sum of the sizes of the items in each U_i is no more than 1 and such that k is as small as possible.



BIN-PACKING

- Given a finite set $U = \{u_1, u_2, \cdots, u_n\}$ of items and a rational size $s(u_i) \in [0,1]$ for each item $u_i \in U$, find a partition of U into disjoint subsets $U_1, U_2, \cdots, U_k$ such that the sum of the sizes of the items in each U_i is no more than 1 and such that k is as small as possible.

- What is the decision version of the bin-packing problem?

BIN-PACKING

- Given a finite set $U = \{u_1, u_2, \cdots, u_n\}$ of items and a rational size $s(u_i) \in [0,1]$ for each item $u_i \in U$, find a partition of U into disjoint subsets $U_1, U_2, \cdots, U_k$ such that the sum of the sizes of the items in each U_i is no more than 1 and such that k is as small as possible.

- What is the decision version of the bin-packing problem?

  - Given a finite set U of items, can they be packed into at most k bins?

BIN-PACKING

- Given a finite set $U = \{u_1, u_2, \cdots, u_n\}$ of items and a rational size $s(u_i) \in [0,1]$ for each item $u_i \in U$, find a partition of U into disjoint subsets $U_1, U_2, \cdots, U_k$ such that the sum of the sizes of the items in each U_i is no more than 1 and such that k is as small as possible.

- What is the decision version of the bin-packing problem?

  - Given a finite set U of items, can they be packed into at most k bins?

- Theorem: BIN-PACKING is NP-complete

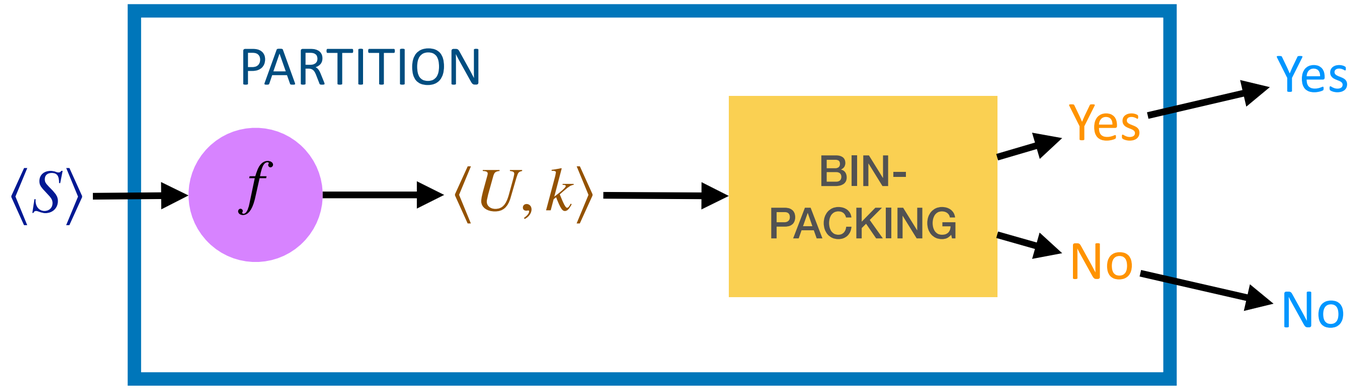
BIN-PACKING

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

- BIN-PACKING $= \{\langle U, k \rangle \mid U$ can be partitioned into at most k disjoint subsets such that the total size of the items in each subset is no more than $1\}$

Reduce PARTITION to BIN-PACKING

- PARTITION $= \{\langle S \rangle \mid S = \{x_1, \cdots, x_n\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\sum_{y_i \in T} y_i = \sum_{z_i \in S \setminus T} z_i\}$
- BIN-PACKING $= \{\langle U, k \rangle \mid U$ can be partitioned into at most k disjoint subsets such that the total size of the items in each subset is no more than $1\}$

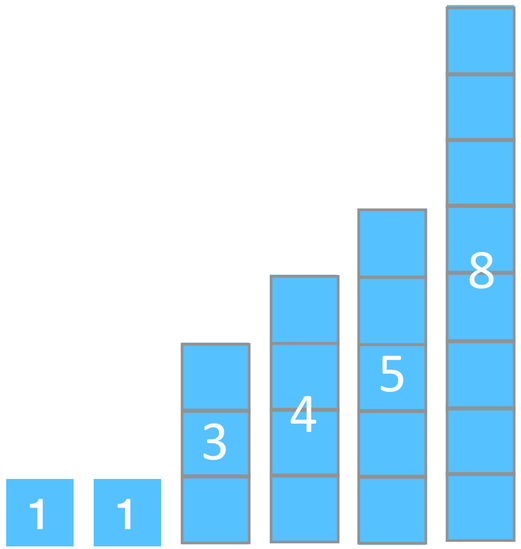


Reduce PARTITION to BIN-PACKING

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

- BIN-PACKING $= \{\langle U, k \rangle \mid U$ can be partitioned into at most k disjoint subsets such that the total size of the items in each subset is no more than $1\}$

$S = \{1,1,3,4,5,8\}$



Reduce PARTITION to BIN-PACKING

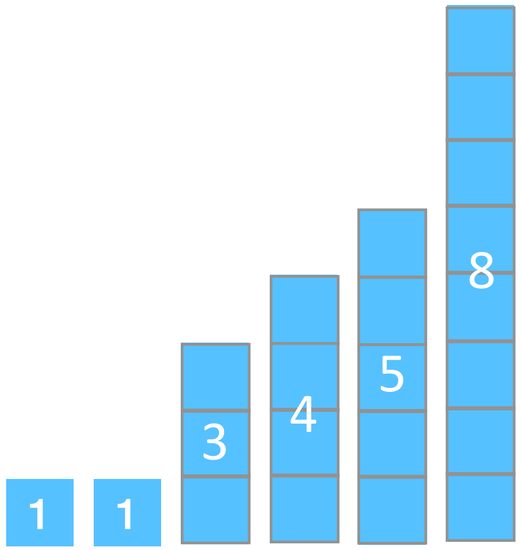
- PARTITION $= \{\langle S \rangle \mid$ $S = \{1,1,3,4,5,8\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\sum_{y_i \in T} y_i = \sum_{z_i \in S \setminus T} z_i\}$

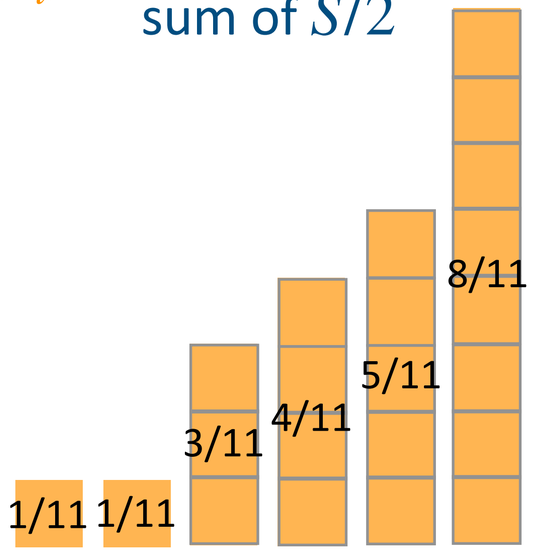
- BIN-PACKING $= \{\langle U, k \rangle \mid U$ can be partitioned into at most k disjoint subsets such that the total size of the items in each subset is no more than $1\}$

Items in U with size $s(u_i) = \dfrac{y_i}{\text{sum of } S/2}$

$S = \{1,1,3,4,5,8\}$

$$\frac{\text{sum of } S}{2} = \frac{22}{2} = 11$$



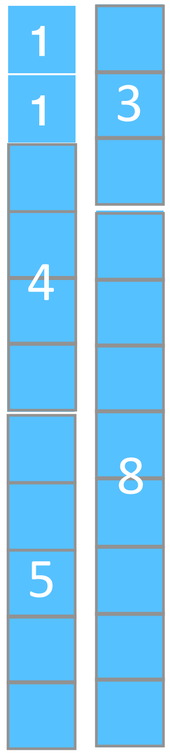


Reduce PARTITION to BIN-PACKING

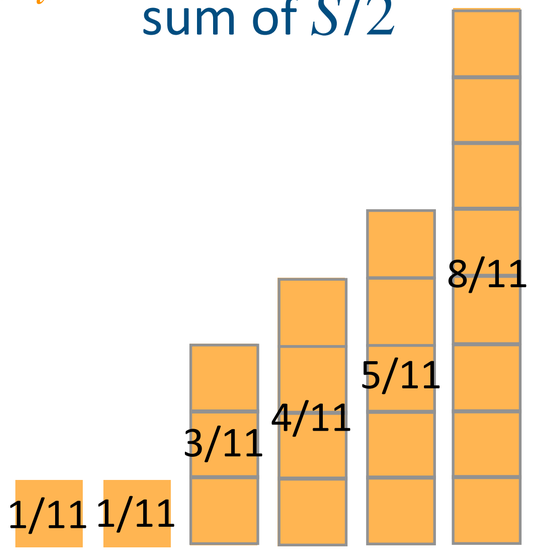
- PARTITION $= \{\langle S \rangle \mid$ $S = \{1,1,3,4,5,8\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\sum_{y_i \in T} y_i = \sum_{z_i \in S \setminus T} z_i\}$

- BIN-PACKING $= \{\langle U, k \rangle \mid U$ can be partitioned into at most k disjoint subsets such that the total size of the items in each subset is no more than $1\}$

$S = \{1,1,3,4,5,8\}$

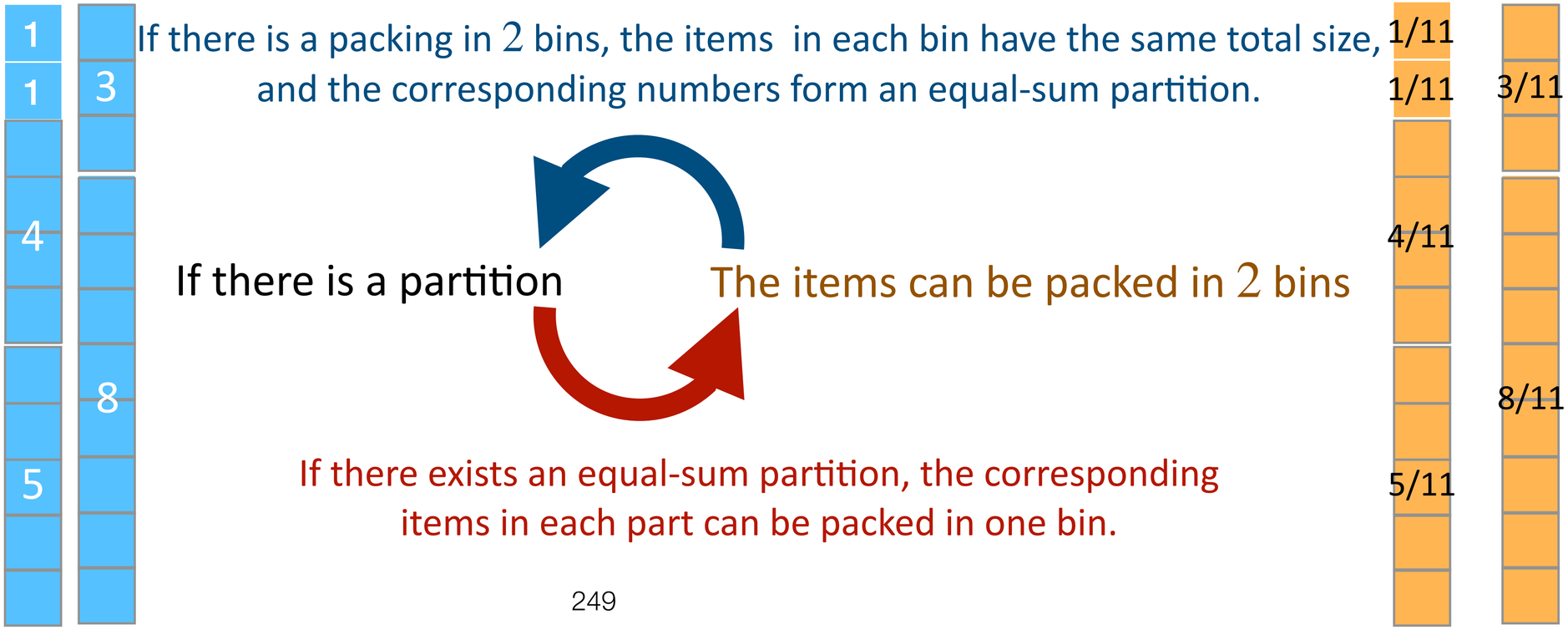


Items in U with size $s(u_i) = \dfrac{y_i}{\text{sum of } S/2}$



Reduce PARTITION to BIN-PACKING

- PARTITION $= \{\langle S \rangle \mid$ $S = \{1,1,3,4,5,8\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\sum_{y_i \in T} y_i = \sum_{z_i \in S \setminus T} z_i\}$
- BIN-PACKING $= \{\langle U, k \rangle \mid U$ can be partitioned into at most k disjoint subsets such that the total size of the items in each subset is no more than $1\}$



BIN-PACKING

- **BIN-PACKING** $= \{\langle U, k\rangle \mid U$ can be partitioned into at most k disjoint subsets such that the total size of the items in each subset is no more than $1\}$

- Theorem: BIN-PACKING is NP-complete

<proof> To prove that BIN-PACKING is in NP, we use a k-partition of U as the certificate. The verifier should check if this partition is a proper partition of U, and if each subset has sum no more than 1. The checking time is in polynomial of the number of elements in U.

BIN-PACKING

To prove the NP-hardness, we show that PARTITION $\leq_p$ BIN-PACKING. For any instance of PARTITION, S, we construct an instance of BIN-PACKING, S' and k as follows. For each element $a_i \in S$, there is a corresponding element u_i in S' and $s(u_i) = \frac{2 \cdot a_i}{X}$, where X is half of the sum of all elements in S. We set $k = 2$. The construction can be done in polynomial time.

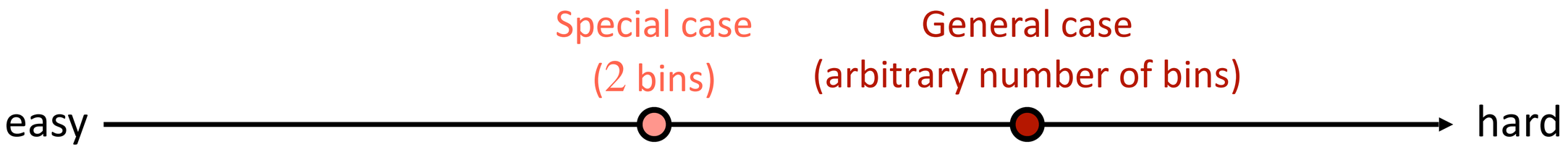
BIN-PACKING

Now we prove that the reduction works. Suppose that there is a partition of S, S_1 and S_2. For all elements $a_i \in S_1$, the sum is X. The sum of corresponding u_i's is 1, so the corresponding items can be placed in one bin. It also holds for S_2. Hence, the items can be packed into 2 bins.

For the other direction, suppose that the items in S' can be packed in two bins. Each of the bin has total size 1 since the total size of all items in S' is $\Sigma_i \ s(u_i) = \frac{\Sigma_i \ a_i}{X} = 2$. The corresponding two subsets of S has equal size and form a partition.

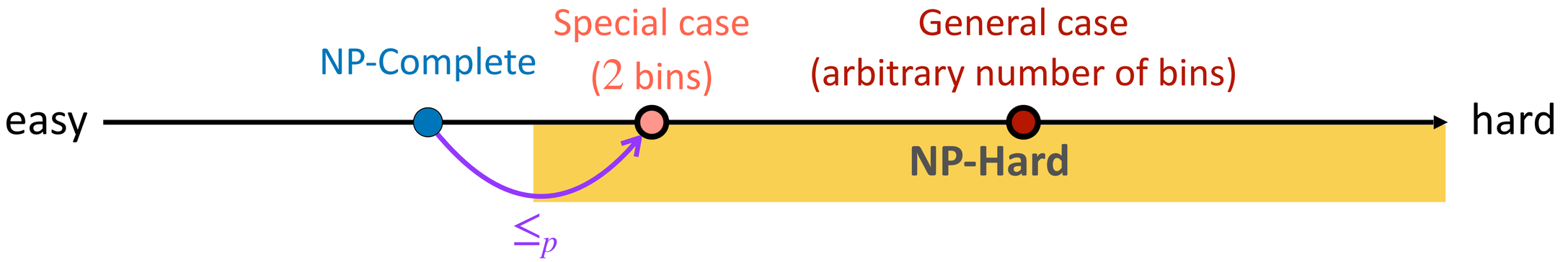
■

Special Case and Hardness

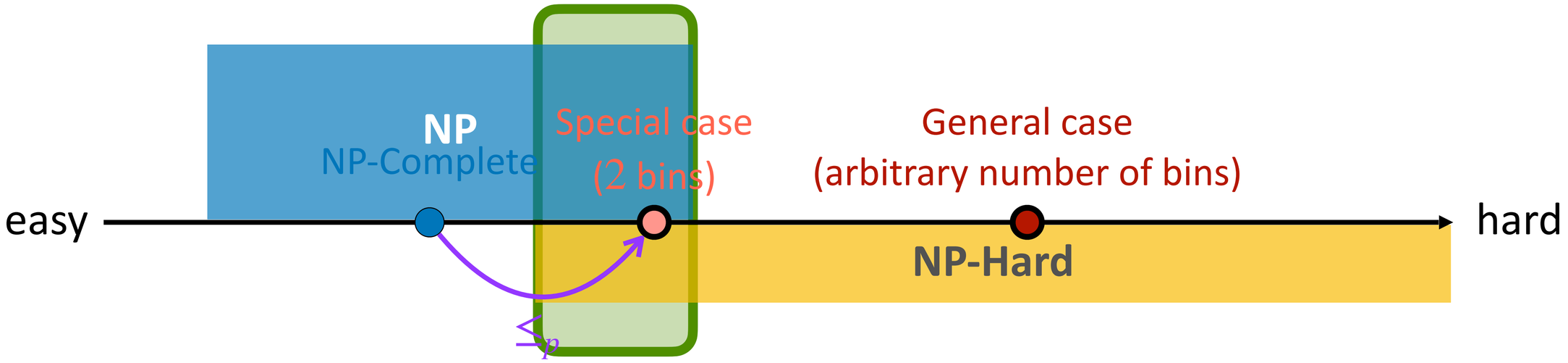


The special case is not harder than the general case

Special Case and Hardness

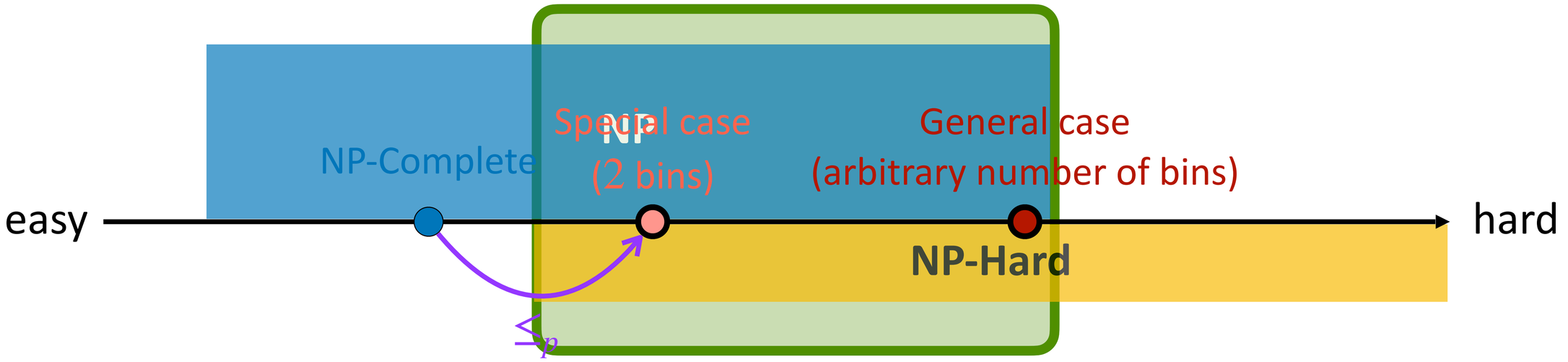


Special Case and Hardness



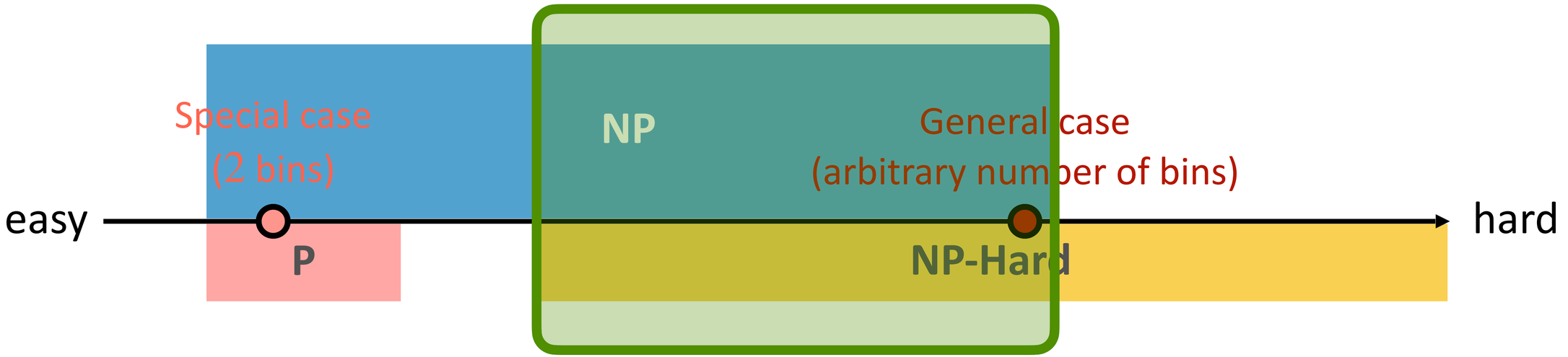
If the special case is NP-complete,
it does not imply that the general case is also NP-complete

Special Case and Hardness



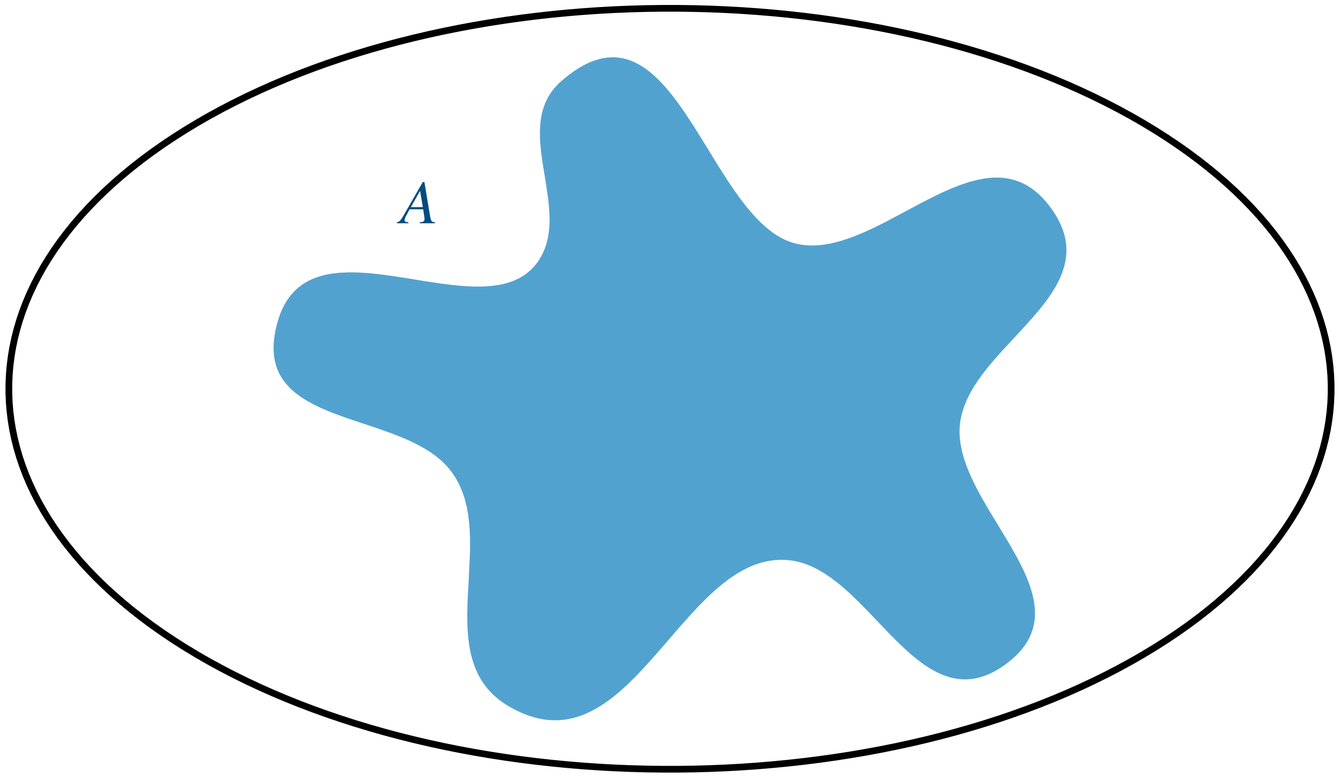
If the general case is NP-complete,
it implies that the special case is also NP-complete

Special Case and Hardness



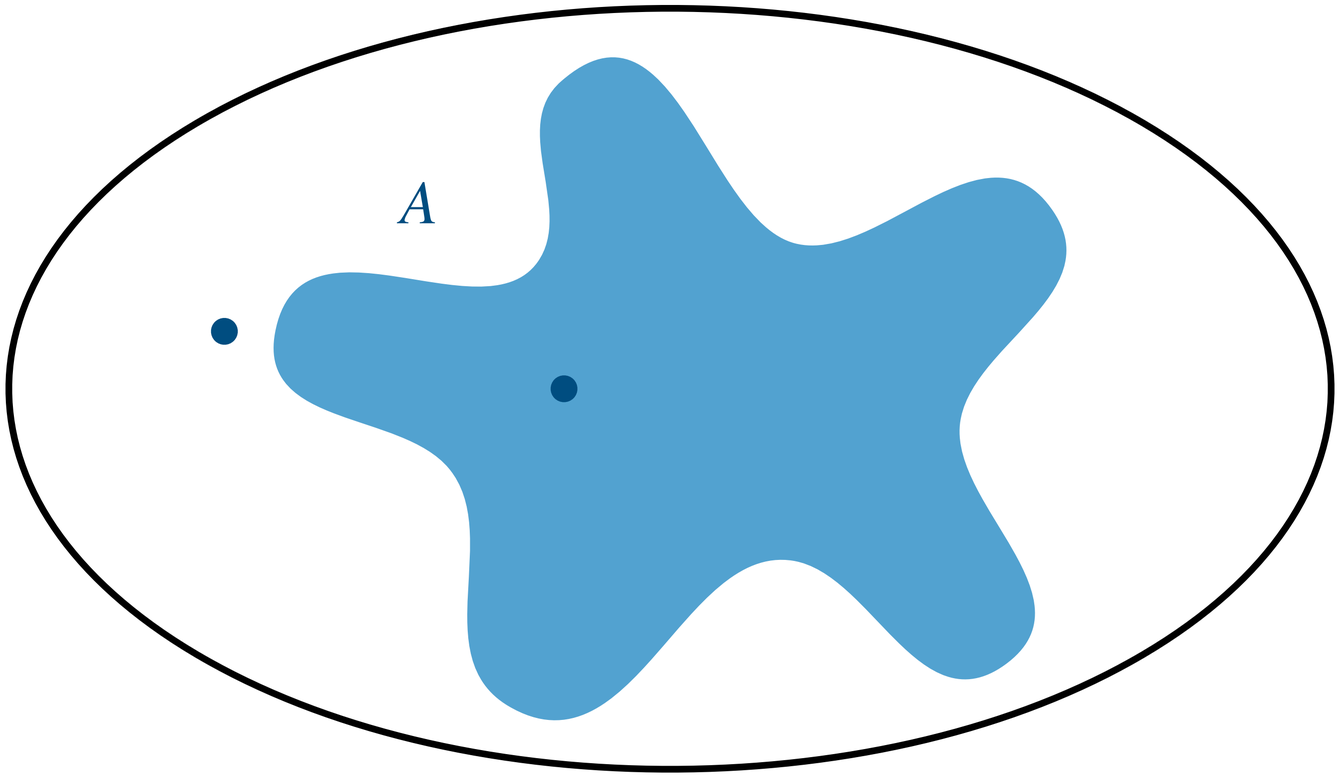
It’s also possible!

Special case and general case



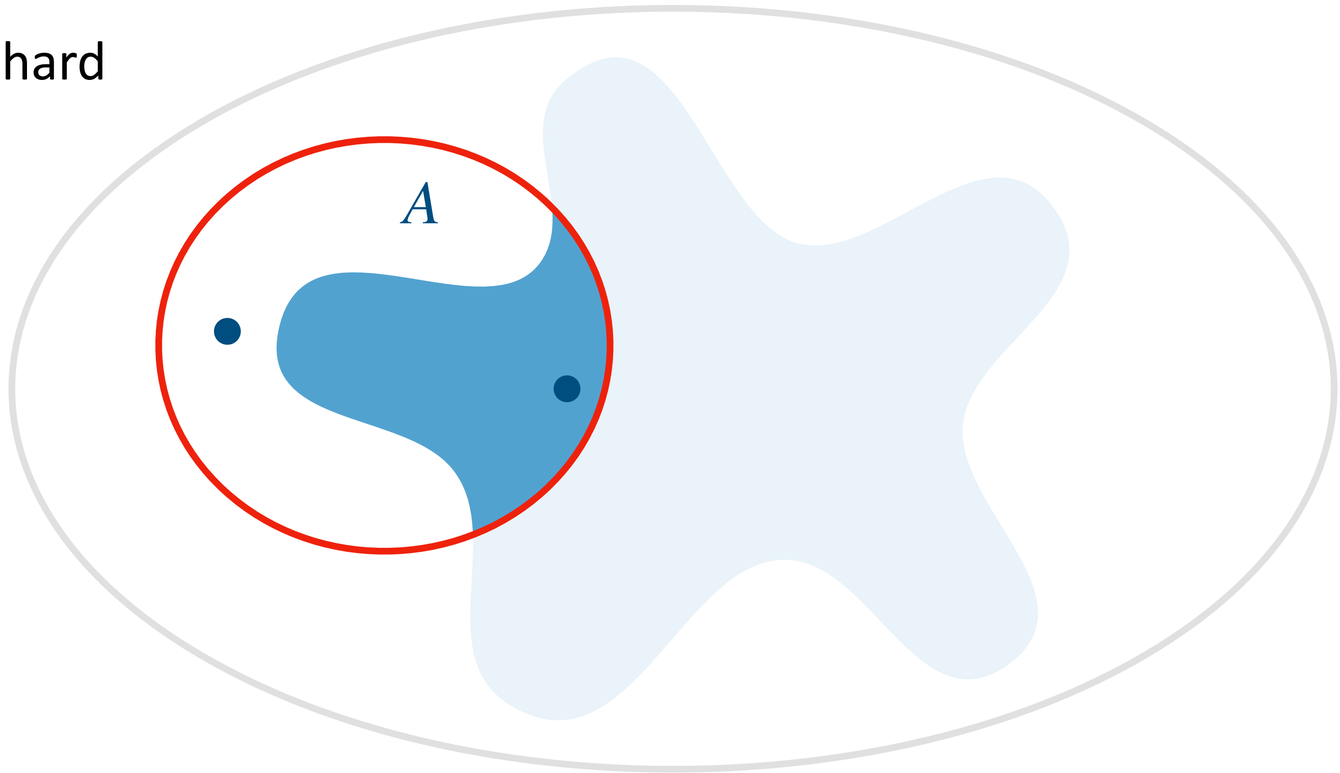
Special case and general case

- A is in **P**: for any instance w, it can be decided if $w \in A$ or $w \notin A$ in polynomial time



Special case and general case

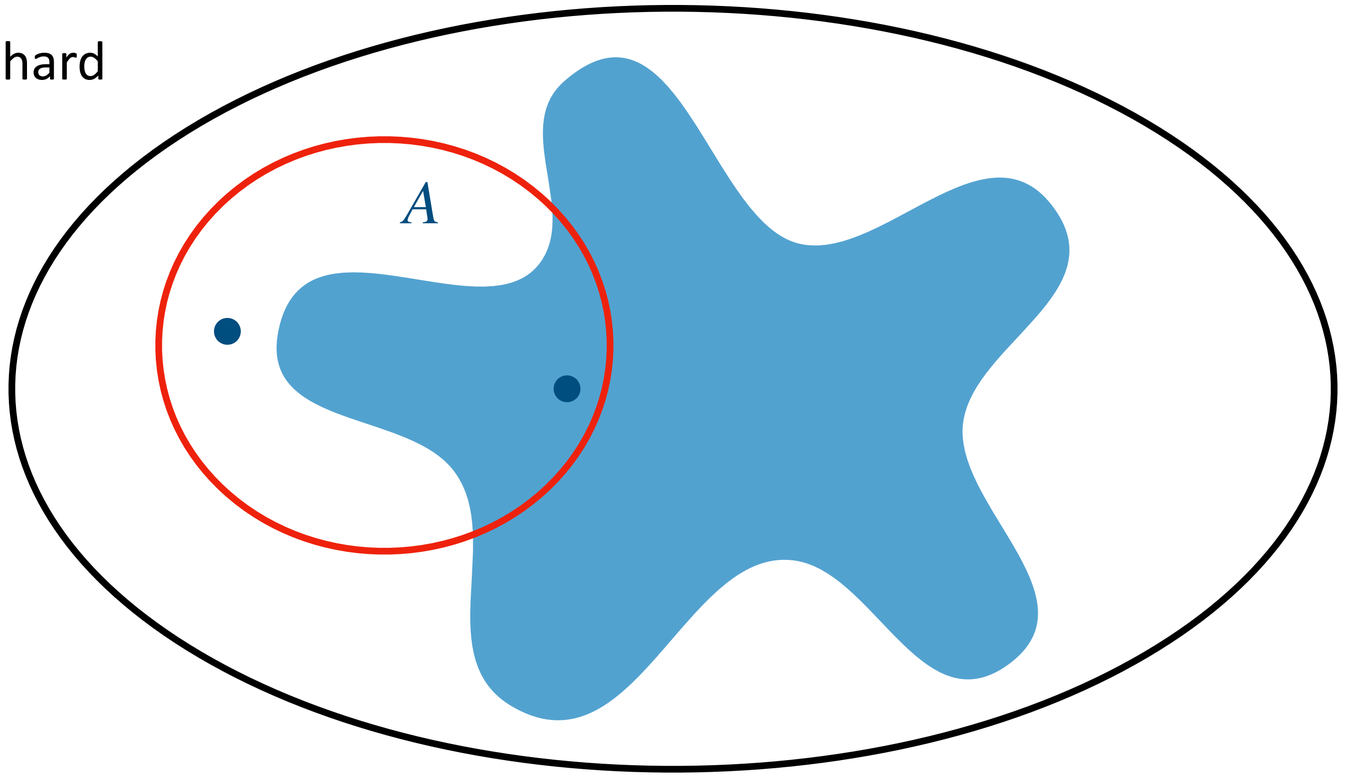
A special case of A is NP-hard



Special case and general case

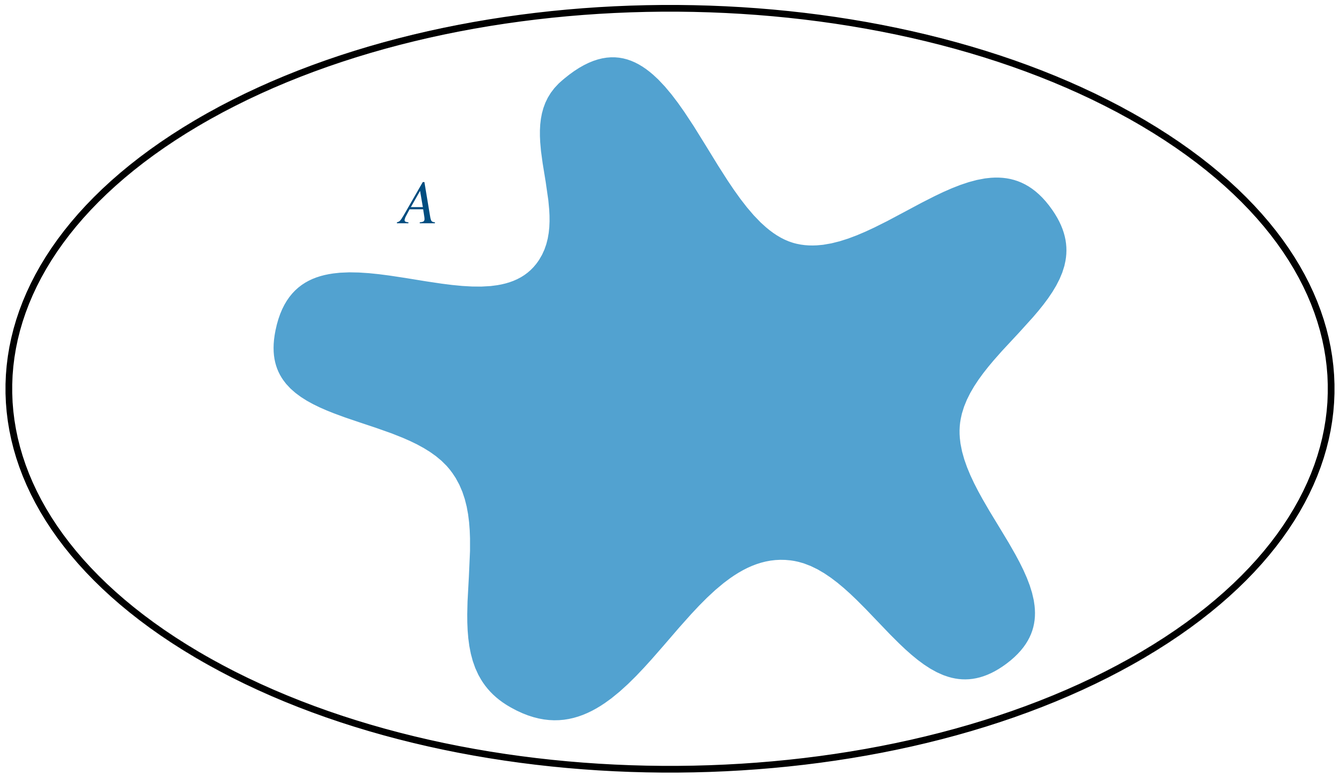
A special case of A is NP-hard

A is NP-hard



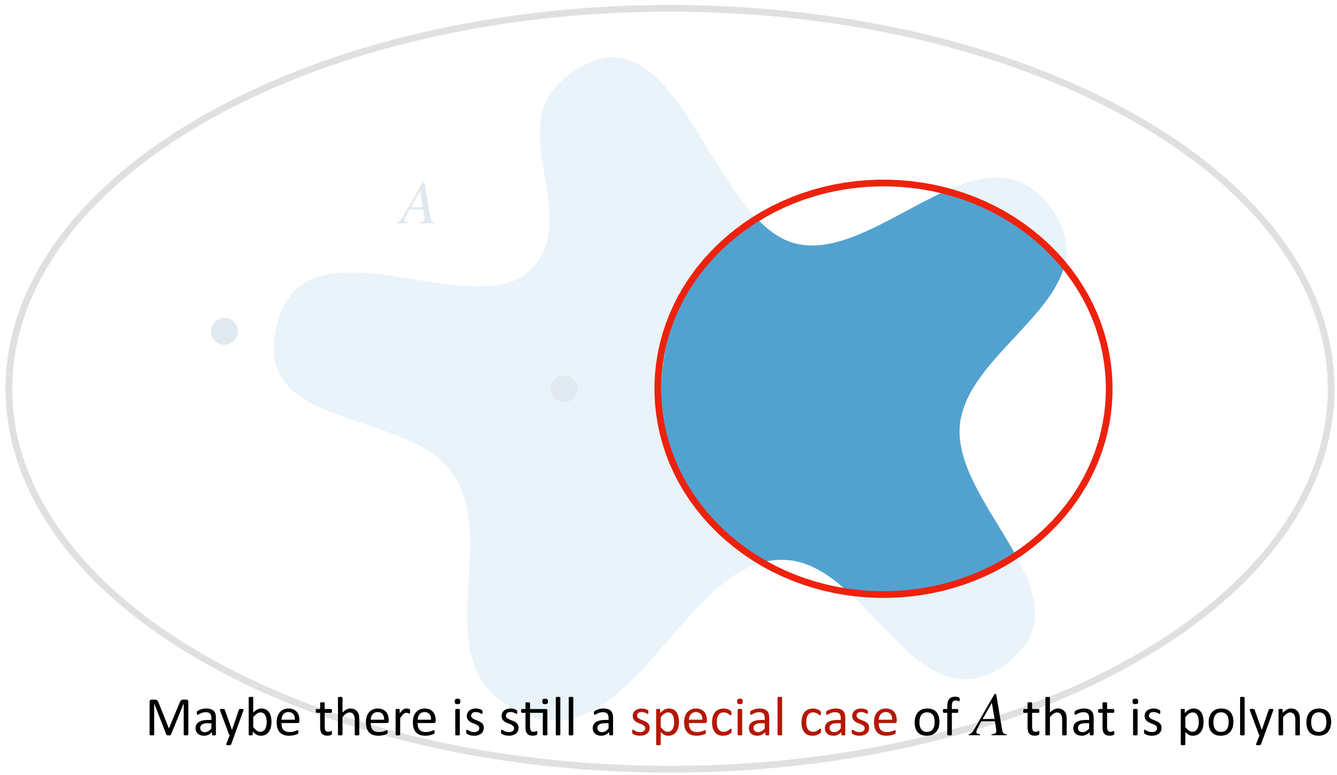
Special case and general case

A is NP-hard



Special case and general case

A is NP-hard



Maybe there is still a special case of A that is polynomial time solvable

Outline

- NP-Completeness
  - NP-hardness: Polynomial time reduction
    - CNF-SAT $\leq_p$ 3SAT
    - 3SAT $\leq_p$ SUBSET-SUM
    - 3SAT $\leq_p$ CLIQUE
    - PARTITION $\leq_p$ BIN-PACKING
- Cook-Leven Theorem: SAT is NP-complete

Cook-Levin Theorem

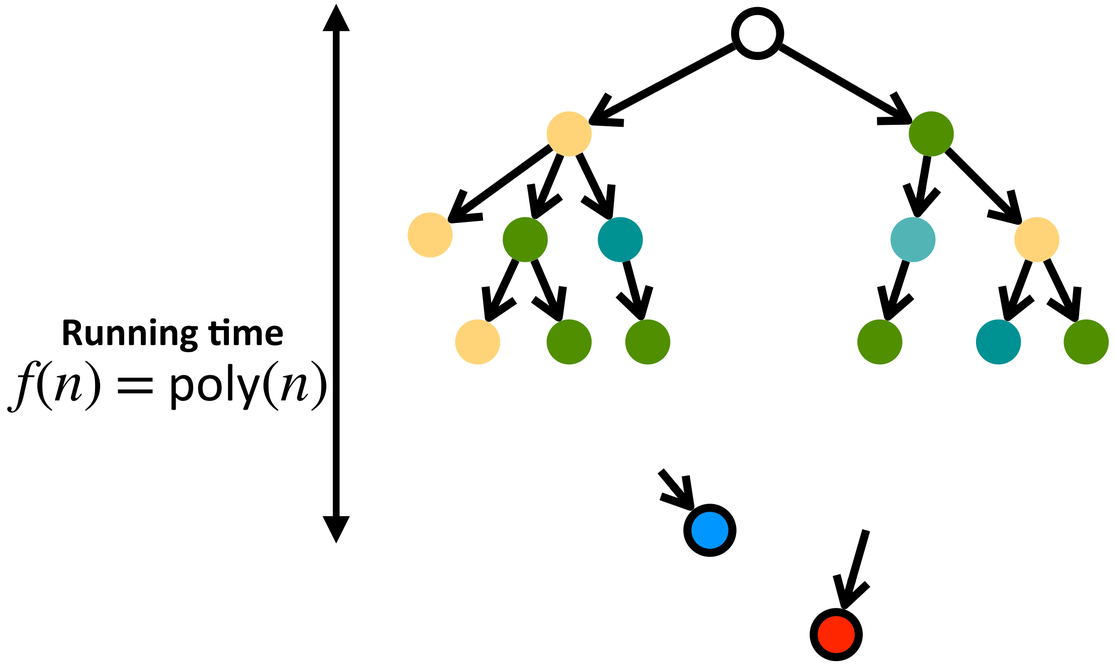
- The first NP-complete problem: **satisfiability problem**
  SAT $= \{\langle\phi\rangle \mid \phi$ is a satisfiable Boolean formaula$\}$

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

  $\leftrightarrow$ SAT is NP-complete

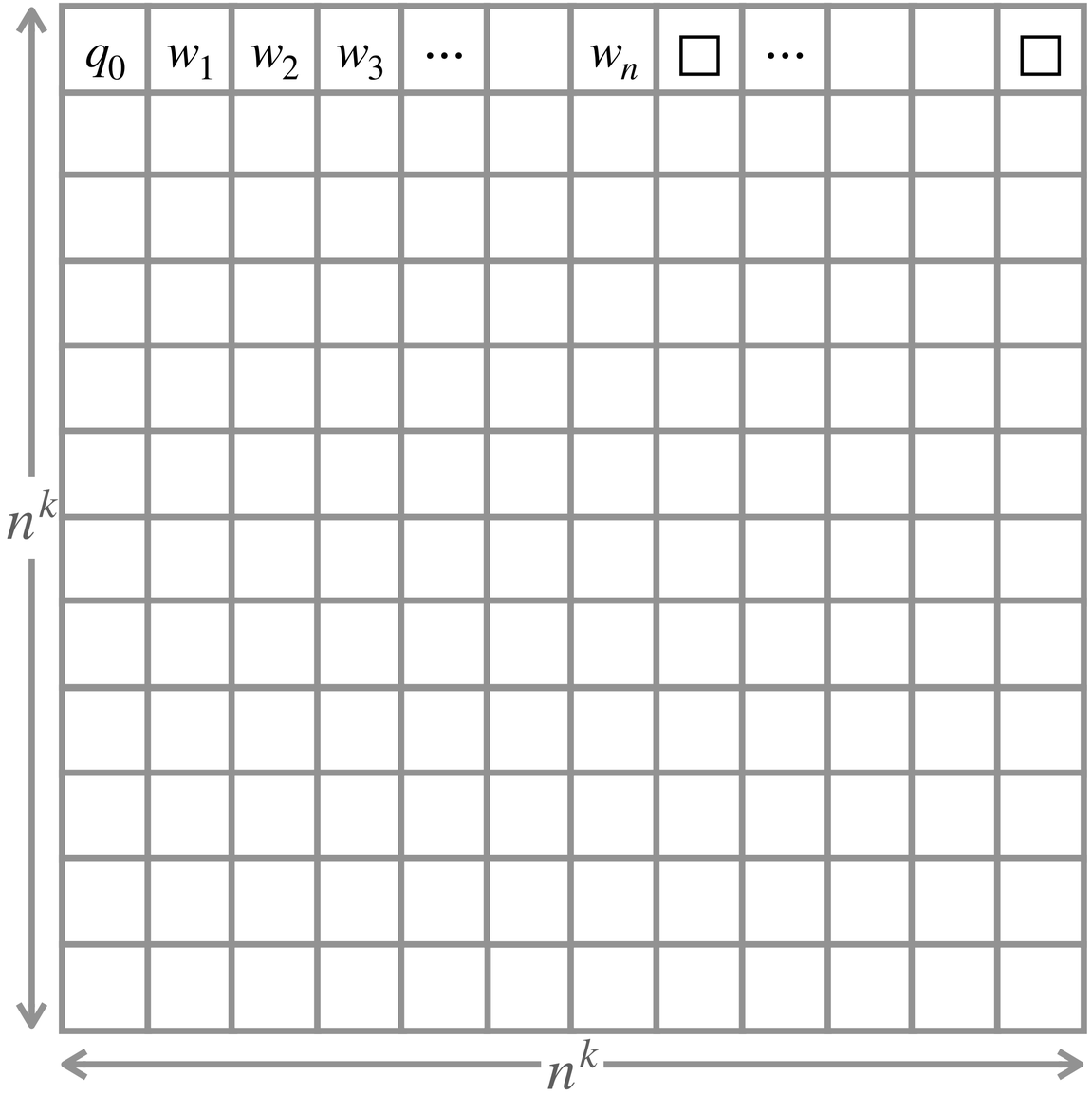
SAT is NP-complete

SAT is NP-complete



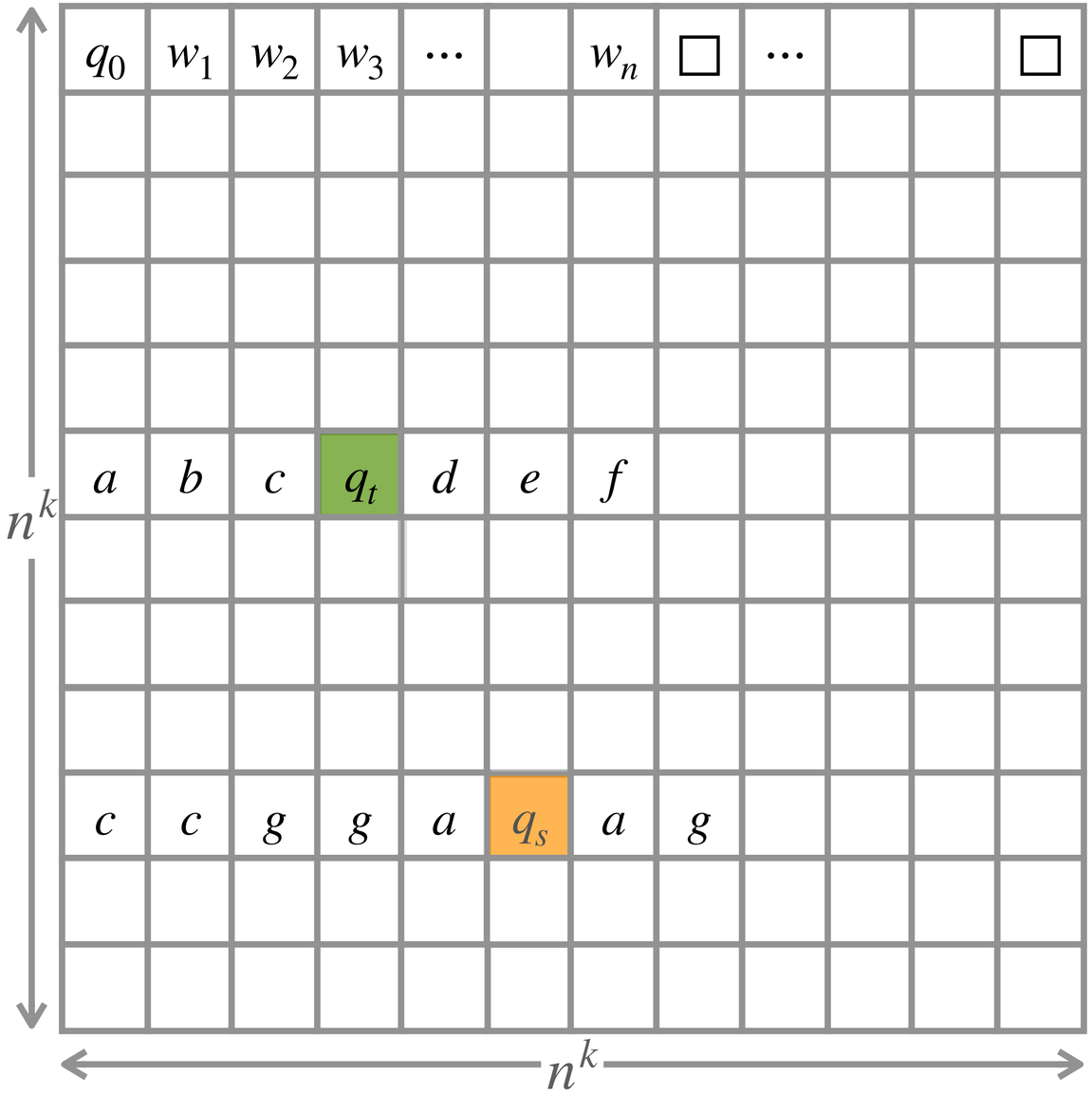
- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps

SAT is NP-complete

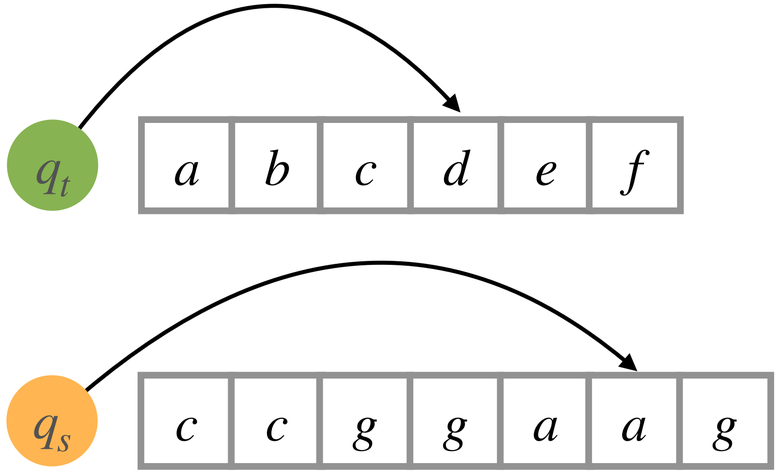


- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. Each row in the table is a configuration of the NTM

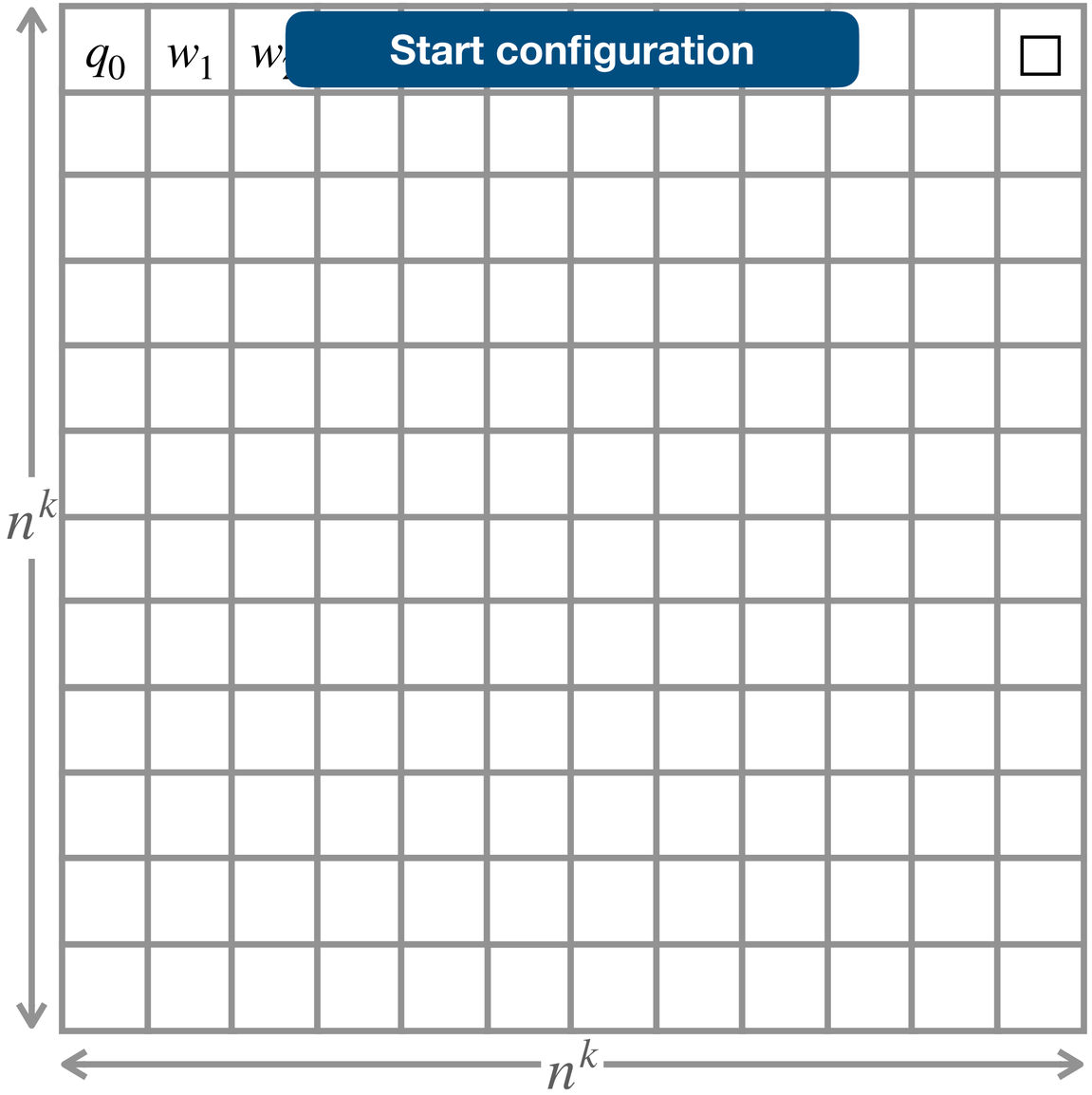
SAT is NP-complete



- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. Each row in the table is a configuration of the NTM

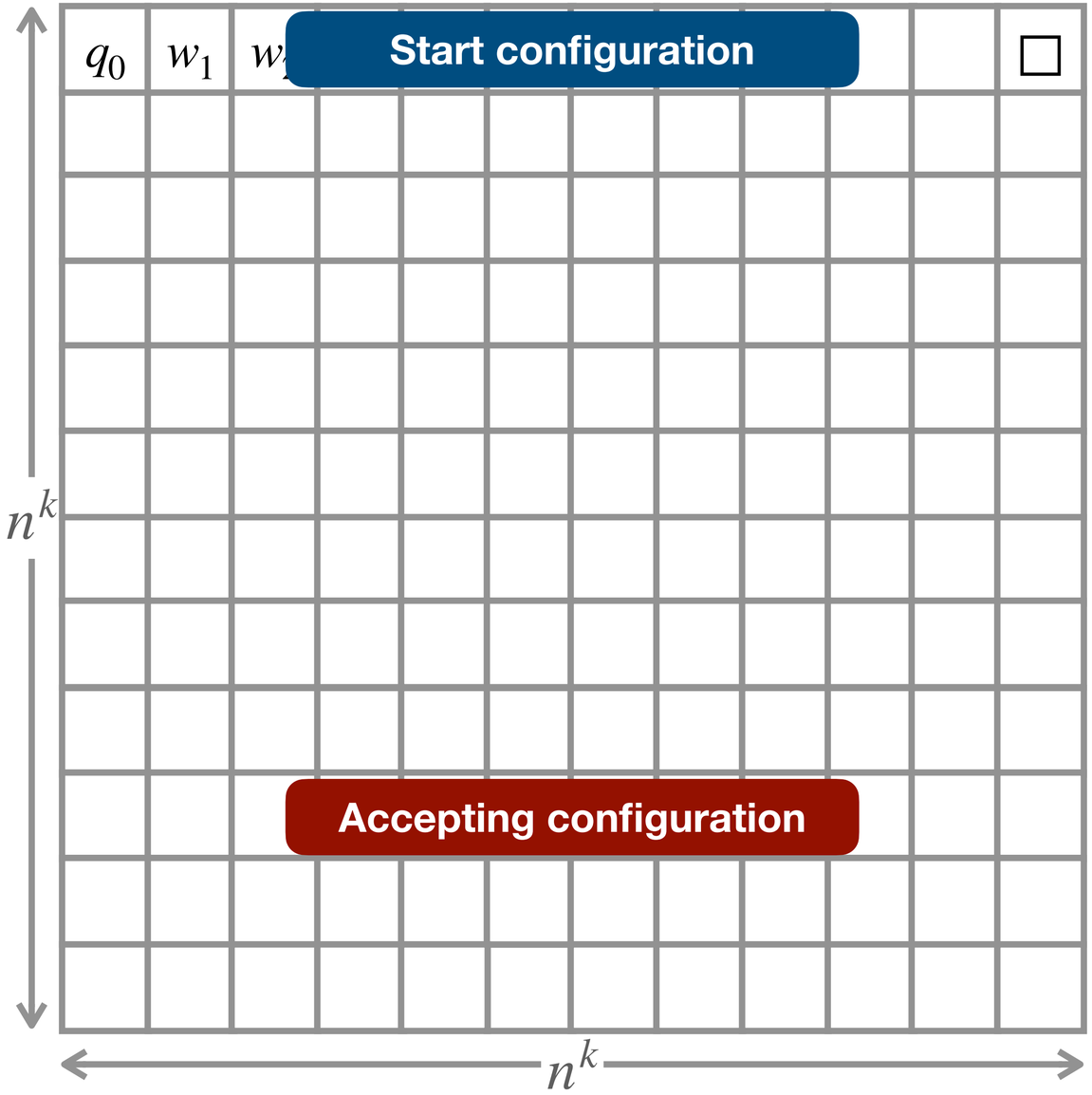


SAT is NP-complete



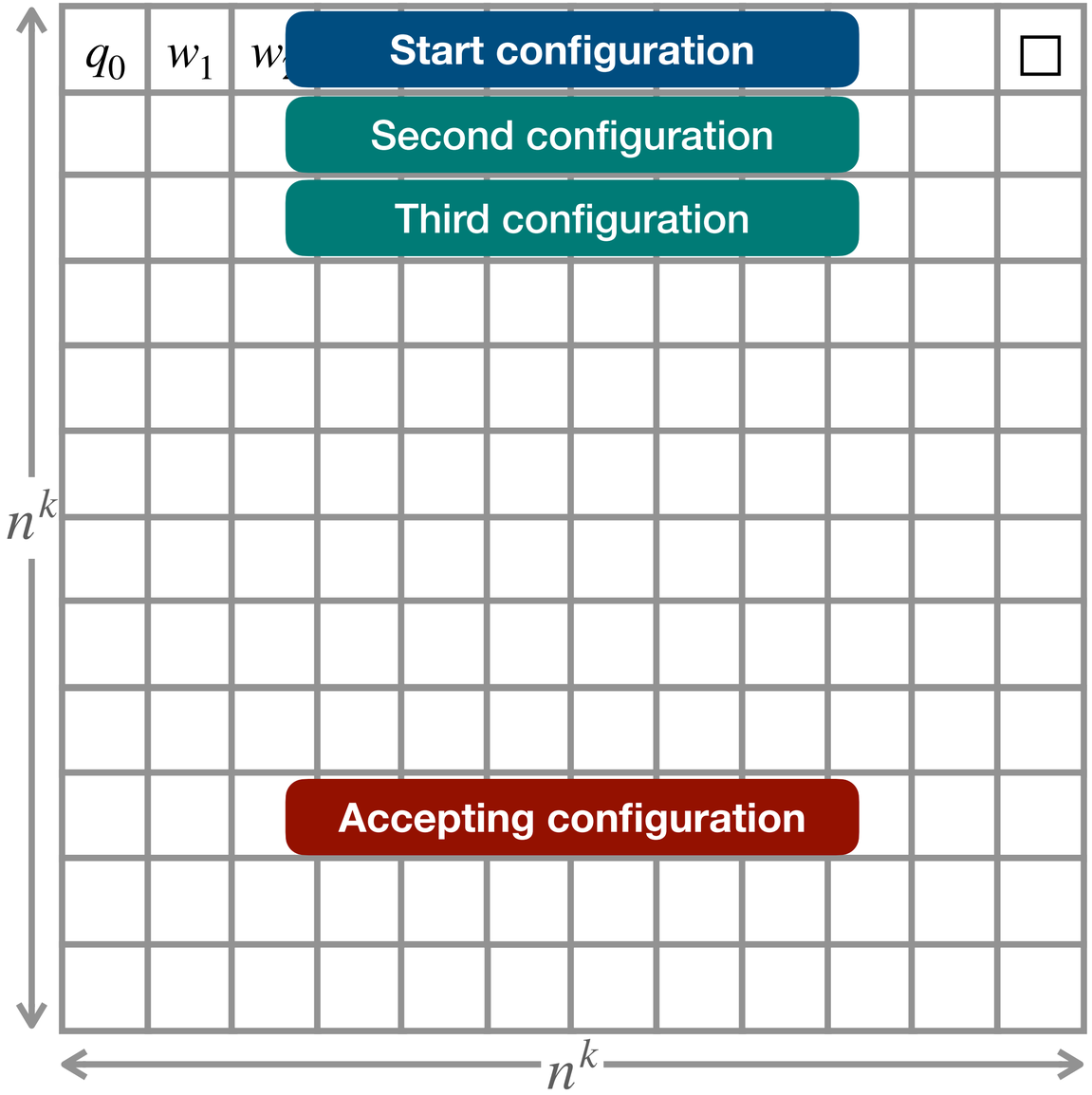
- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. Each row in the table is a configuration of the NTM
  2. The first row is the starting configuration

SAT is NP-complete



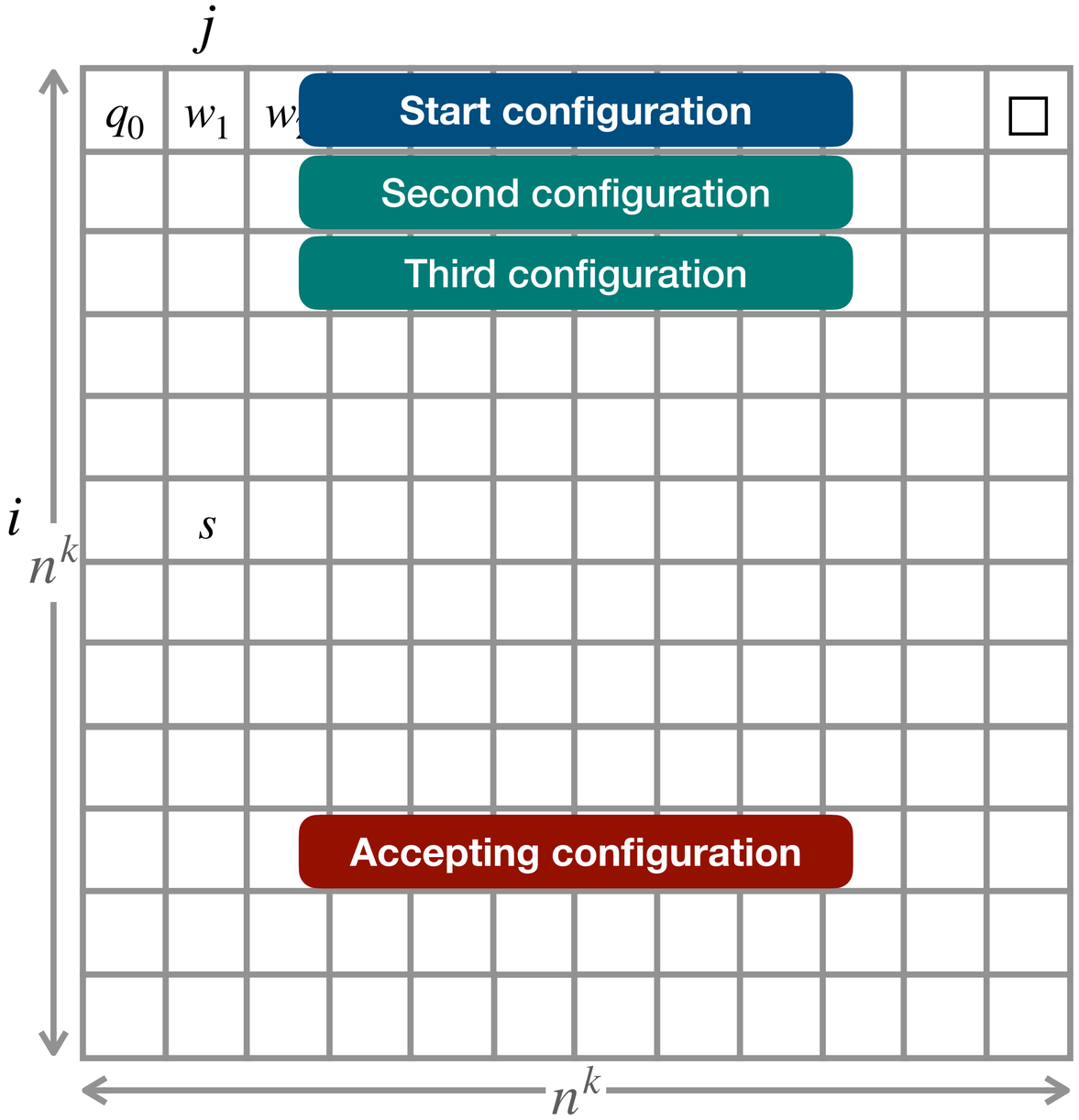
- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. Each row in the table is a configuration of the NTM
  2. The first row is the starting configuration
  3. There is a accepting configuration

SAT is NP-complete



- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. Each row in the table is a configuration of the NTM
  2. The first row is the starting configuration
  3. There is a accepting configuration
  4. The configurations corresponding to consecutive rows follow the NTM's rules

SAT is NP-complete

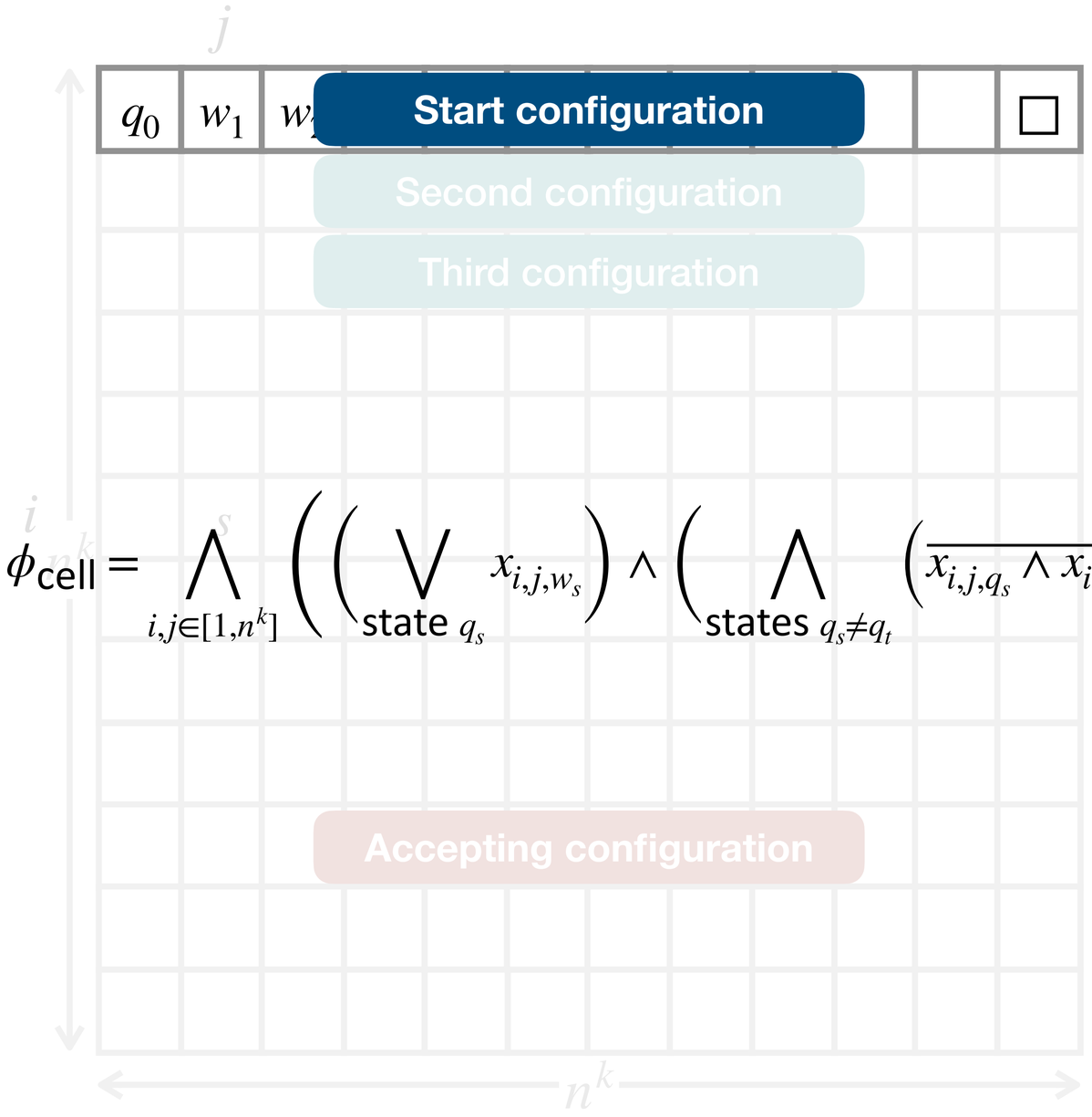


- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. Each row in the table is a configuration of the NTM
  2. The first row is the starting configuration
  3. There is a accepting configuration
  4. The configurations corresponding to consecutive rows follow the NTM's rules

$x_{row,\ column,\ symbol} = 1$ if the cell$[i, j]$ is the symbol

$$x_{1,2,w_1} = 1, x_{i,j,s} = 1, x_{1,1,q_1} = 0$$

SAT is NP-complete



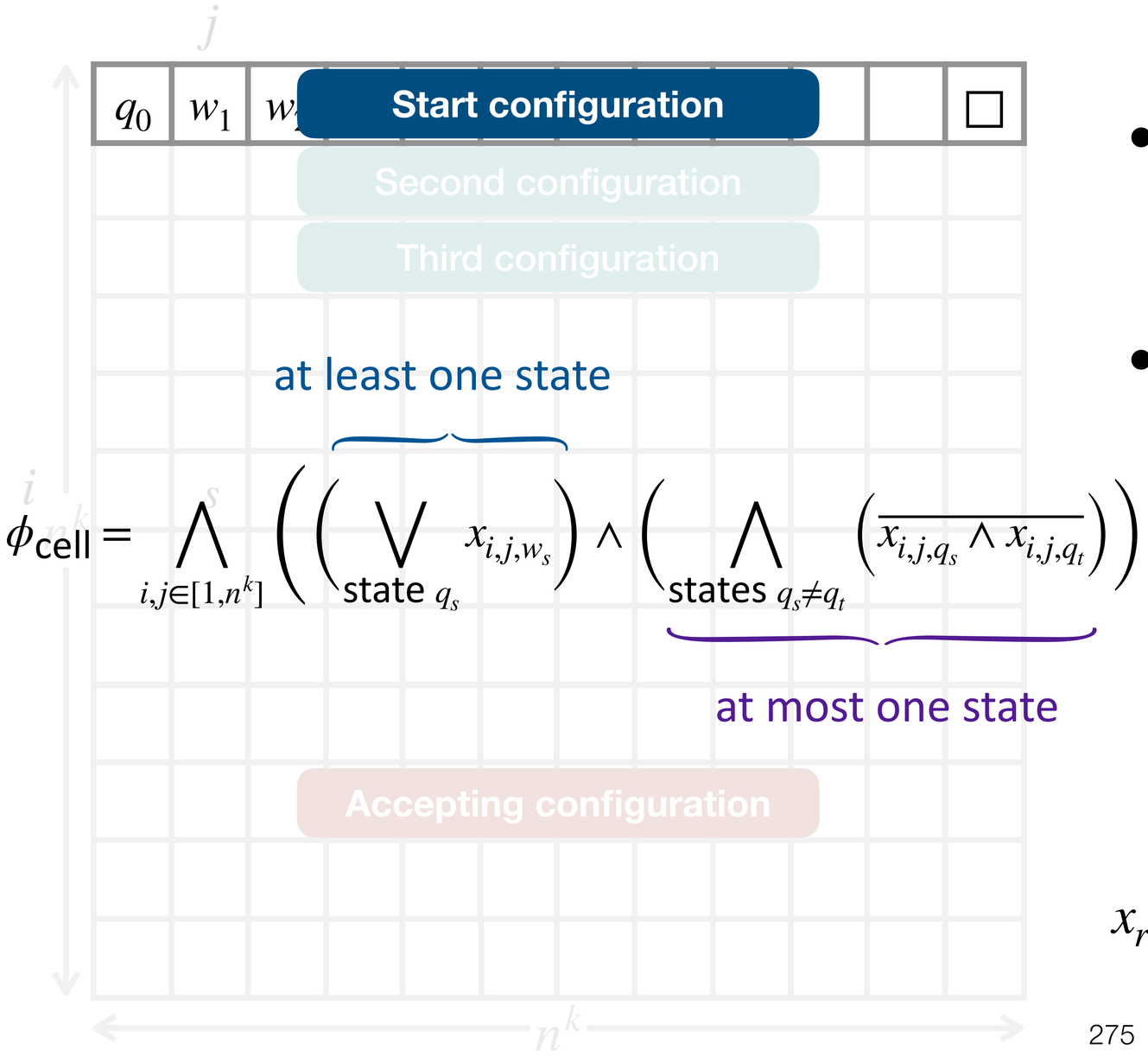
$$\phi_{\text{cell}} = \bigwedge_{i,j \in [1,n^k]} \left(\left(\bigvee_{\text{state } q_s} x_{i,j,w_s} \right) \wedge \left(\bigwedge_{\text{states } q_s \neq q_t} \left(\overline{x_{i,j,q_s} \wedge x_{i,j,q_t}} \right) \right) \right)$$

- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. **Each row in the table is a configuration of the NTM**
  2. The first row is the starting configuration
  3. There is a accepting configuration
  4. The configurations corresponding to consecutive rows follow the NTM's rules

$x_{row,\ column,\ symbol} = 1$ if the cell$[i, j]$ is the symbol

$$x_{1,2,w_1} = 1, x_{i,j,s} = 1, x_{1,1,q_1} = 0$$

SAT is NP-complete

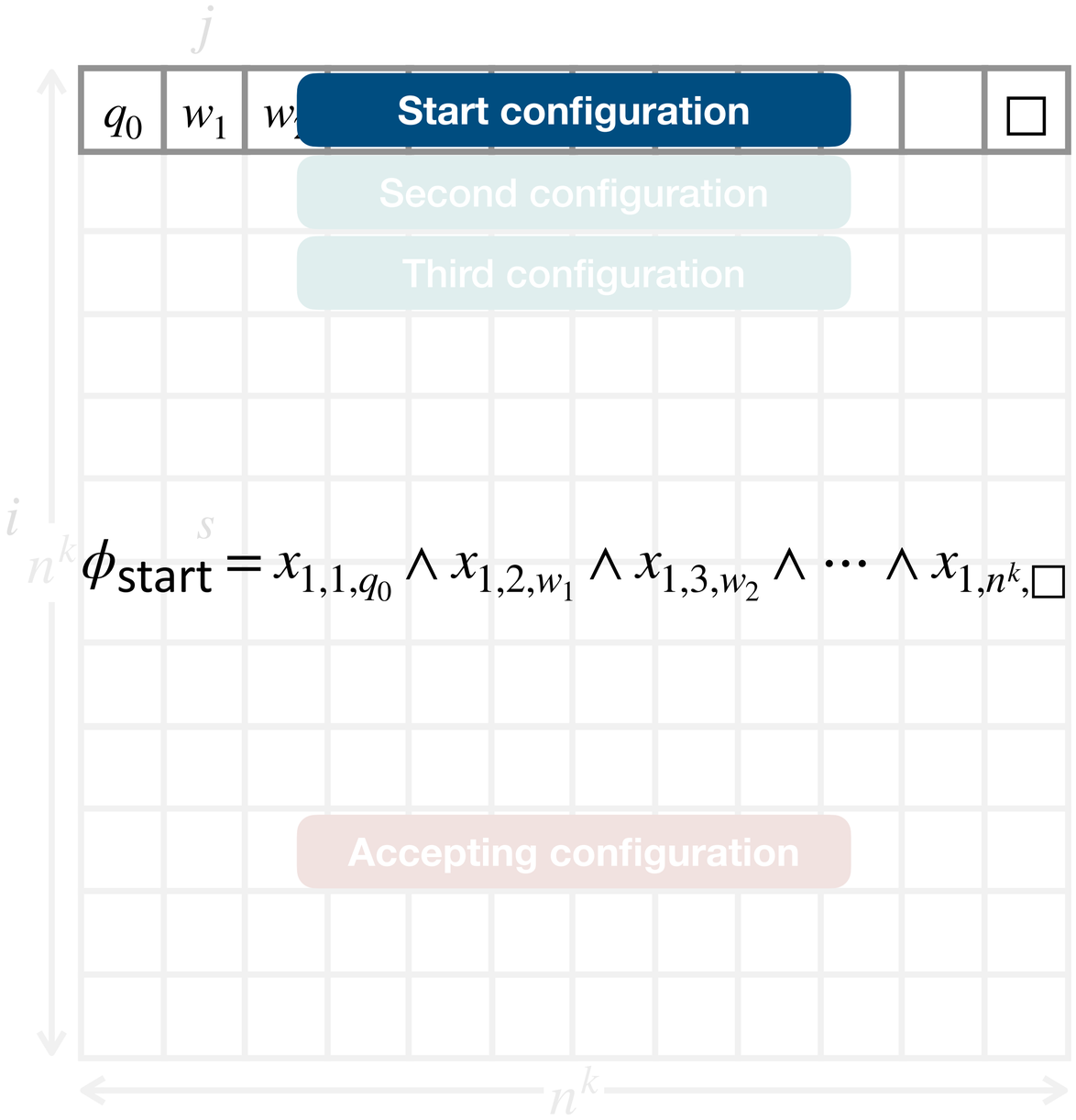


- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. **Each row in the table is a configuration of the NTM**
  2. The first row is the starting configuration
  3. There is a accepting configuration
  4. The configurations corresponding to consecutive rows follow the NTM's rules

$x_{row,\ column,\ symbol} = 1$ if the cell$[i, j]$ is the symbol

$$x_{1,2,w_1} = 1, x_{i,j,s} = 1, x_{1,1,q_1} = 0$$

SAT is NP-complete

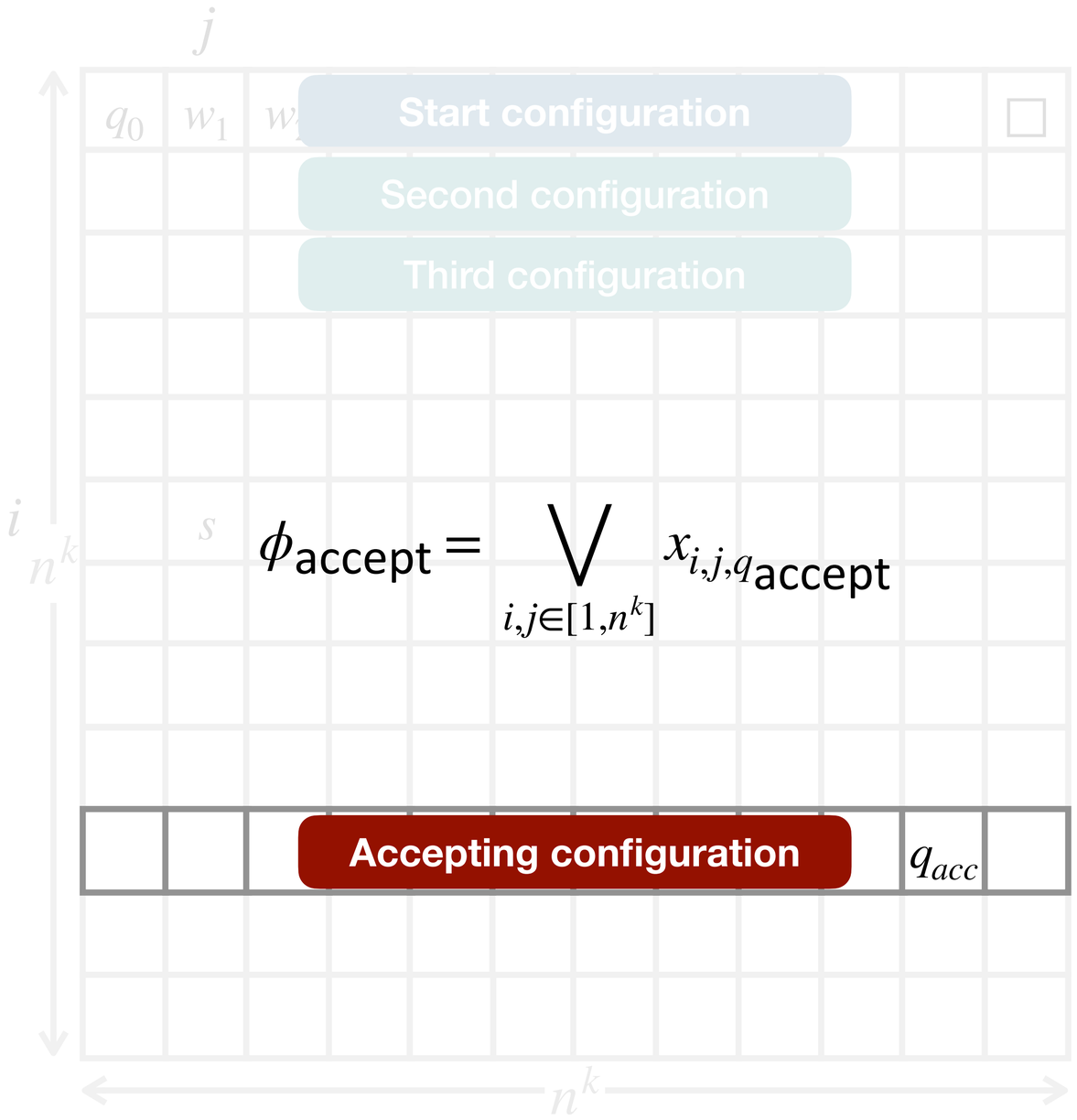


- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. Each row in the table is a configuration of the NTM
  2. **The first row is the starting configuration**
  3. There is a accepting configuration
  4. The configurations corresponding to consecutive rows follow the NTM's rules

$x_{row,\ column,\ symbol} = 1$ if the cell$[i, j]$ is the symbol

$$x_{1,2,w_1} = 1, x_{i,j,s} = 1, x_{1,1,q_1} = 0$$

SAT is NP-complete

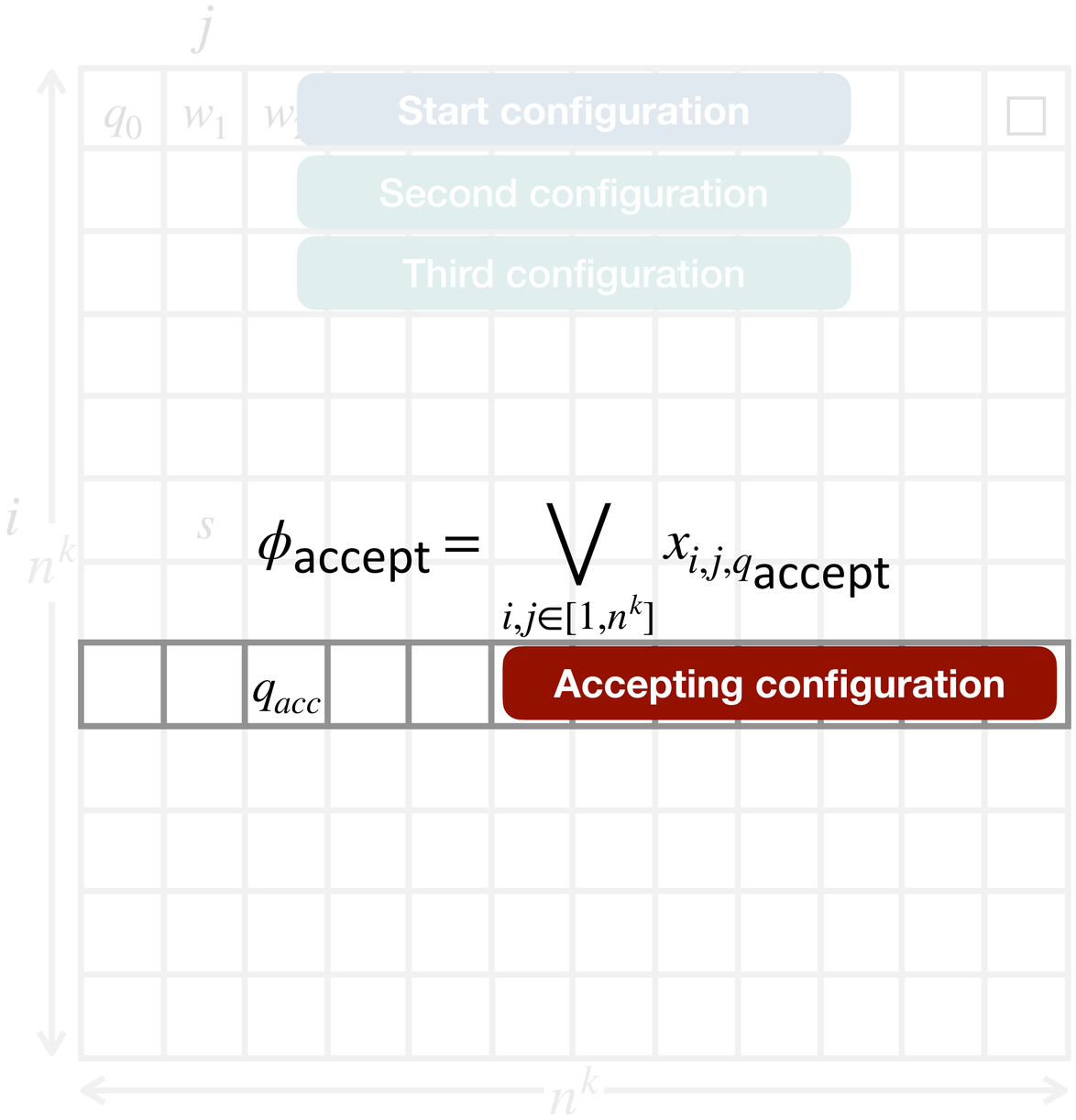


- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. Each row in the table is a configuration of the NTM
  2. The first row is the starting configuration
  3. **There is a accepting configuration**
  4. The configurations corresponding to consecutive rows follow the NTM's rules

$x_{row,\ column,\ symbol} = 1$ if the cell$[i, j]$ is the symbol

$$x_{1,2,w_1} = 1, x_{i,j,s} = 1, x_{1,1,q_1} = 0$$

SAT is NP-complete

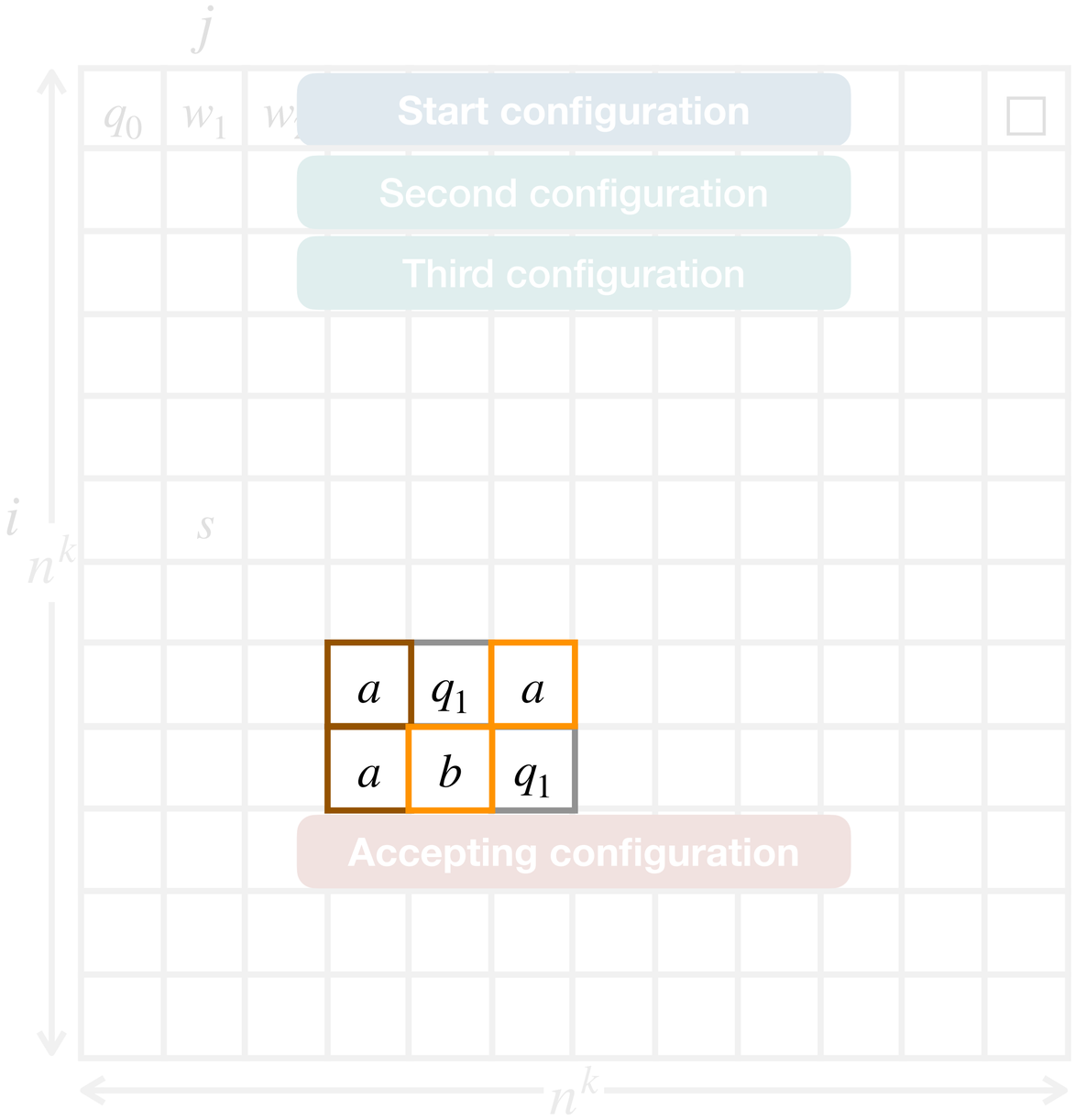


- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. Each row in the table is a configuration of the NTM
  2. The first row is the starting configuration
  3. **There is a accepting configuration**
  4. The configurations corresponding to consecutive rows follow the NTM's rules

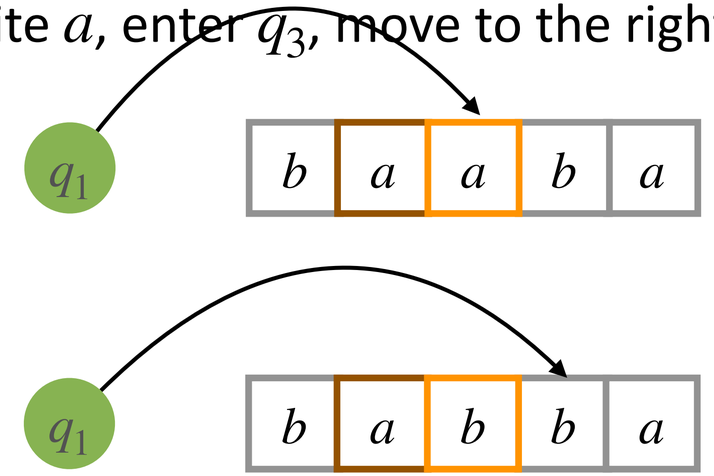
$x_{row,\ column,\ symbol} = 1$ if the cell$[i, j]$ is the symbol

$$x_{1,2,w_1} = 1, x_{i,j,s} = 1, x_{1,1,q_1} = 0$$

SAT is NP-complete



- State q_1 and read a: write b, move to the right
- State q_1 and read b:
  - write c, enter q_2, move to the left, or
  - write a, enter q_3, move to the right

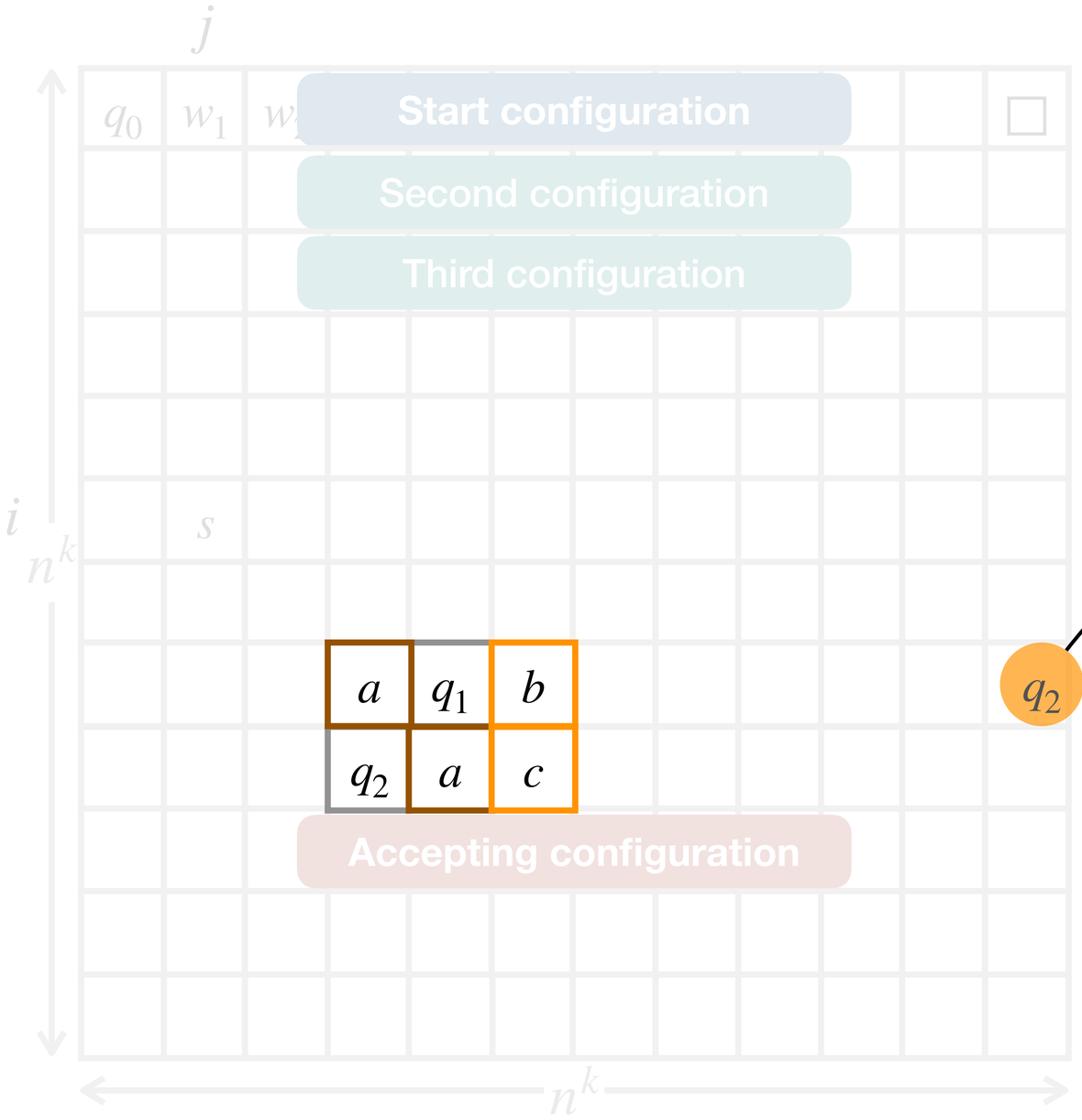


4. **The configurations corresponding to consecutive rows follow the NTM's rules**

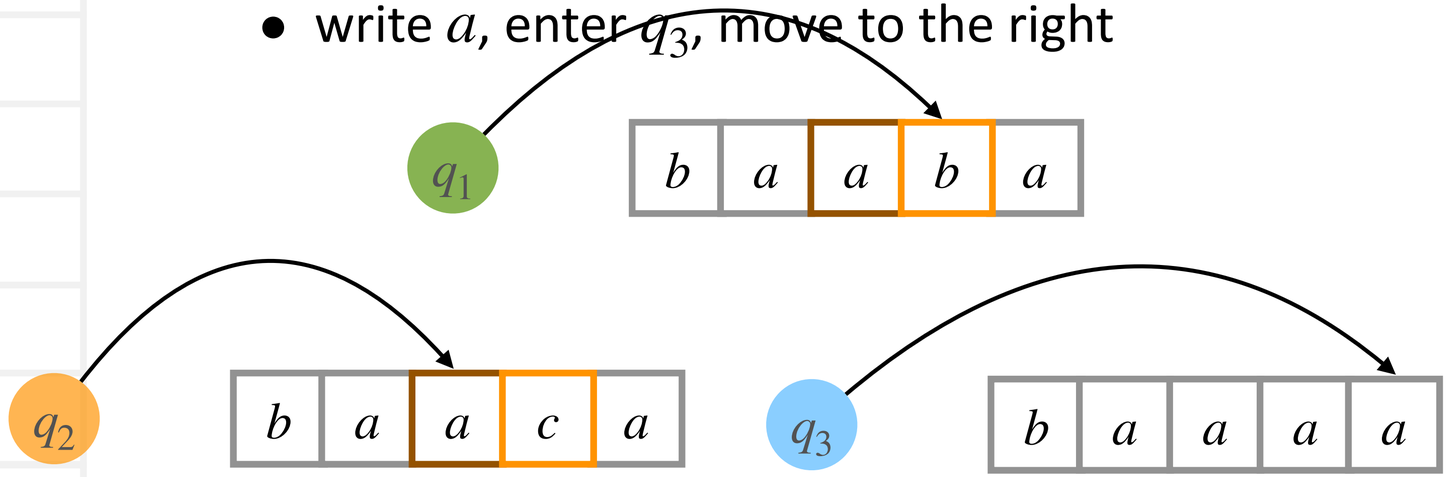
$x_{row,\ column,\ symbol} = 1$ if the cell$[i, j]$ is the symbol

$$x_{1,2,w_1} = 1, x_{i,j,s} = 1, x_{1,1,q_1} = 0$$

SAT is NP-complete



- State q_1 and read a: write b, move to the right
- State q_1 and read b:
  - write c, enter q_2, move to the left, or
  - write a, enter q_3, move to the right

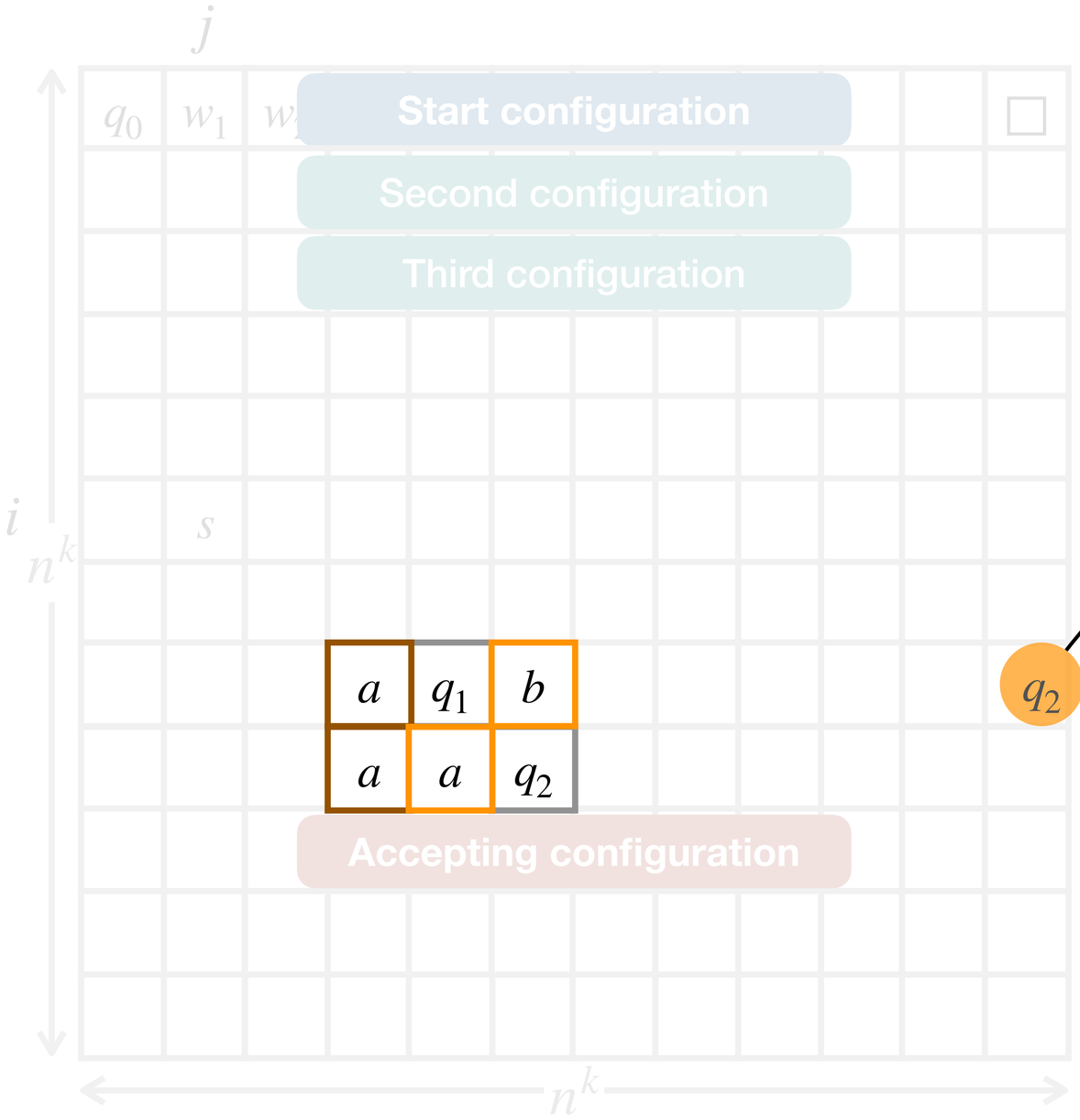


4. **The configurations corresponding to consecutive rows follow the NTM's rules**

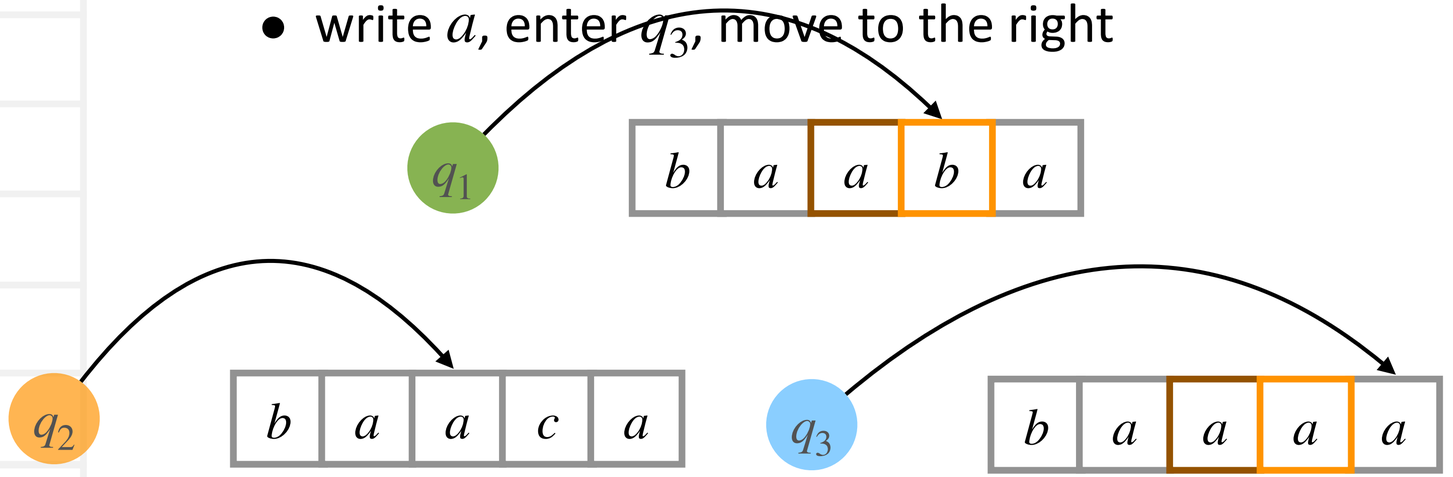
$x_{row,\ column,\ symbol} = 1$ if the cell$[i, j]$ is the symbol

$$x_{1,2,w_1} = 1, x_{i,j,s} = 1, x_{1,1,q_1} = 0$$

SAT is NP-complete



- State q_1 and read a: write b, move to the right
- State q_1 and read b:
  - write c, enter q_2, move to the left, or
  - write a, enter q_3, move to the right

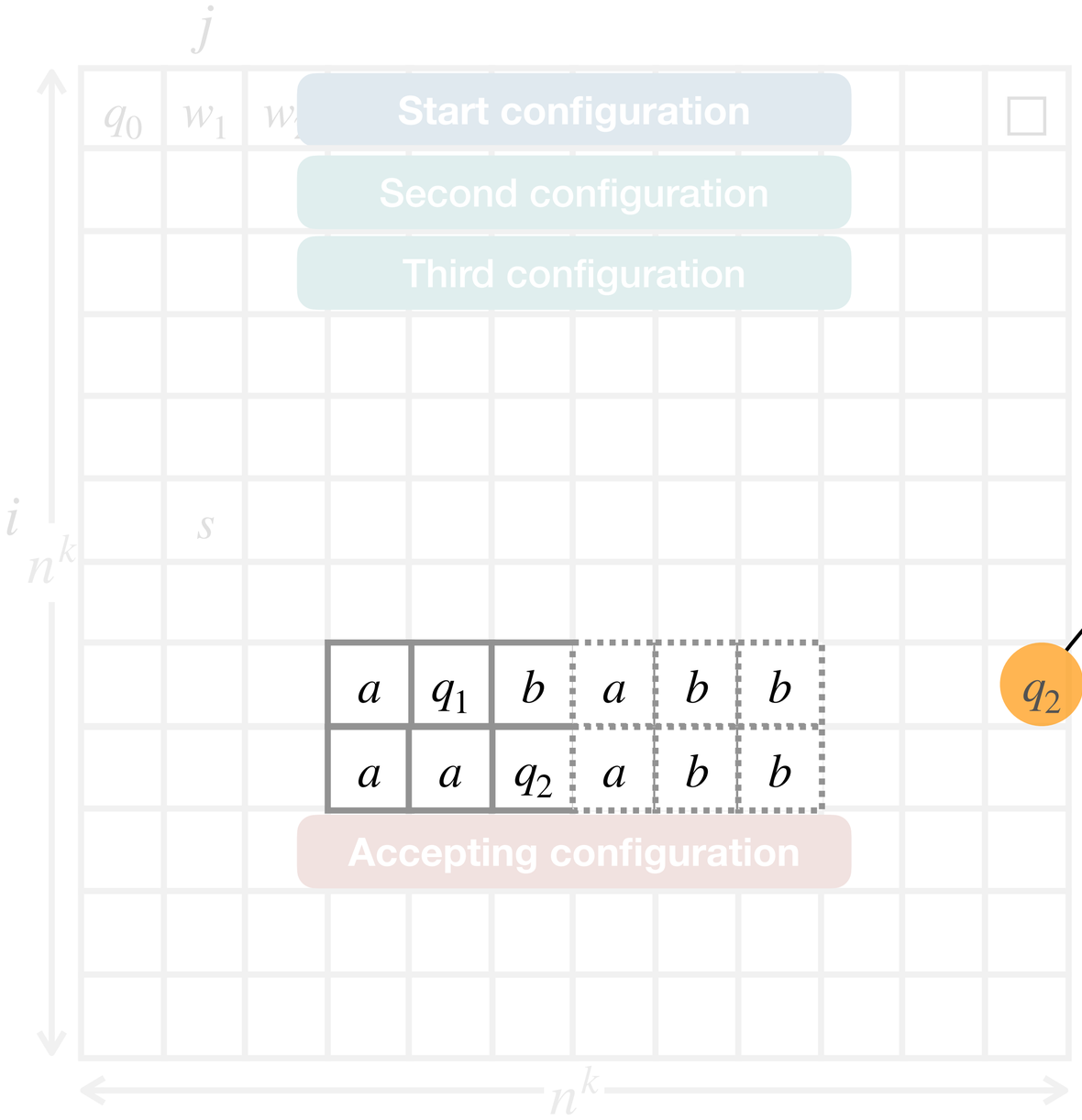


4. **The configurations corresponding to consecutive rows follow the NTM's rules**

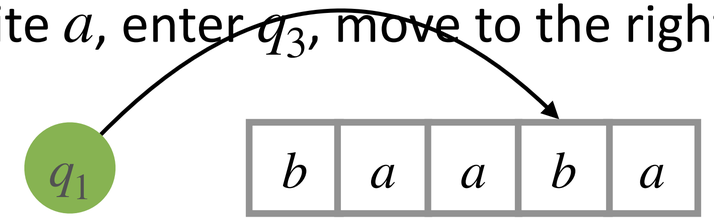
$x_{row,\ column,\ symbol} = 1$ if the cell$[i, j]$ is the symbol

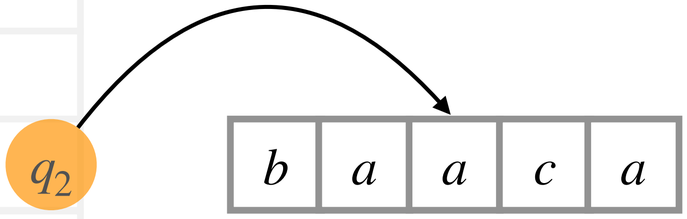
$$x_{1,2,w_1} = 1, x_{i,j,s} = 1, x_{1,1,q_1} = 0$$

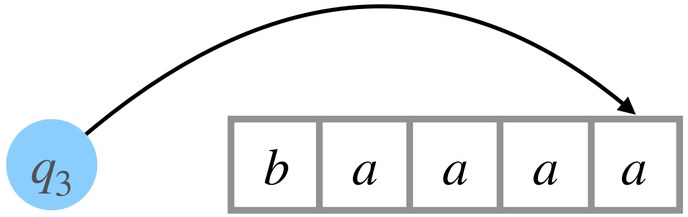
SAT is NP-complete



- State q_1 and read a: write b, move to the right
- State q_1 and read b:
  - write c, enter q_2, move to the left, or
  - write a, enter q_3, move to the right





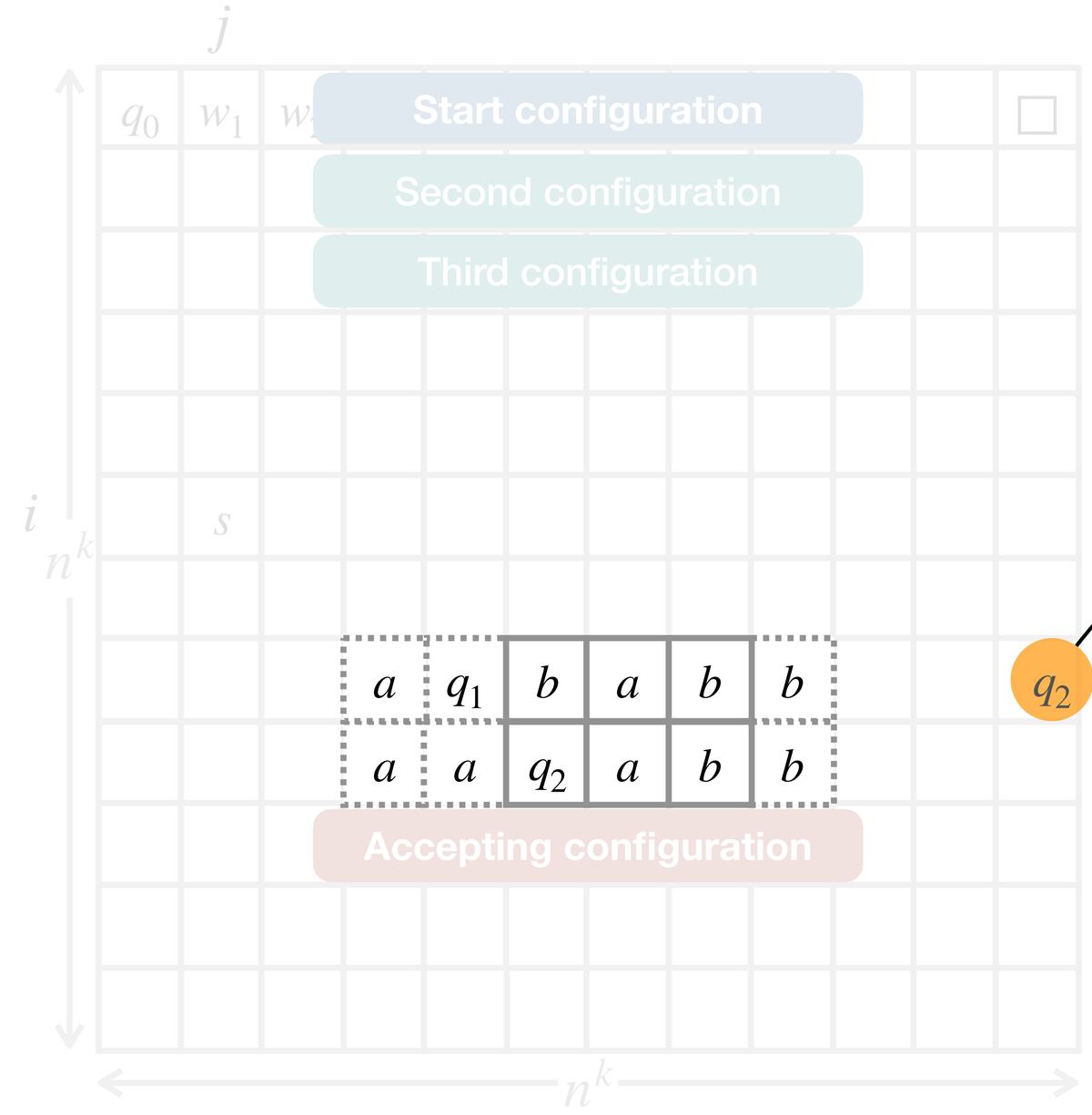


4. **The configurations corresponding to consecutive rows follow the NTM's rules**

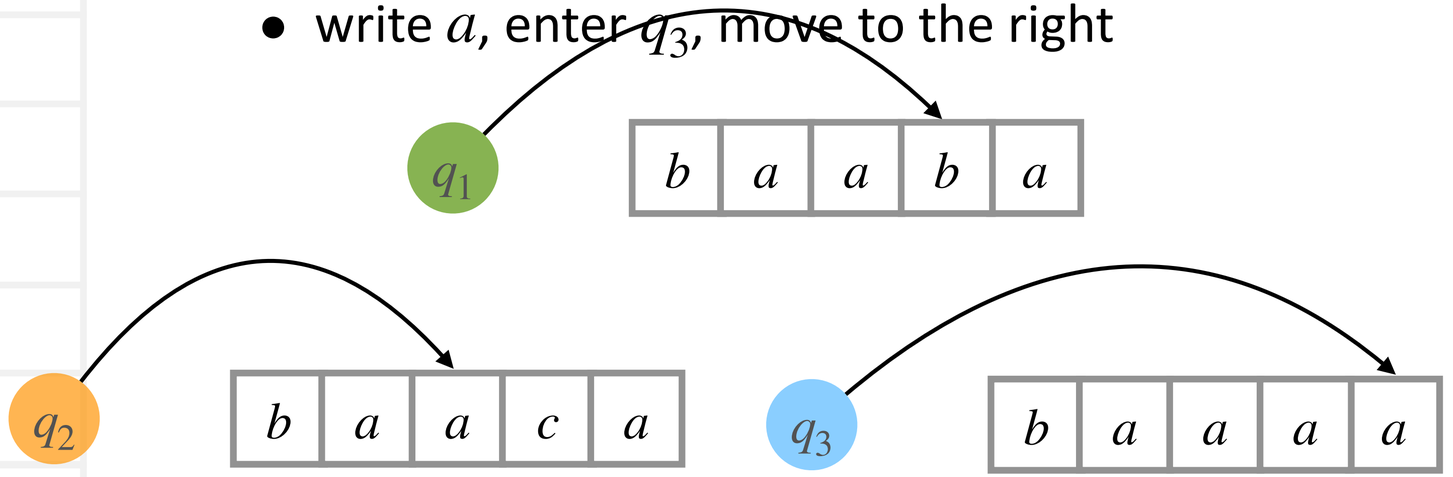
$x_{row,\ column,\ symbol} = 1$ if the cell$[i, j]$ is the symbol

$$x_{1,2,w_1} = 1, x_{i,j,s} = 1, x_{1,1,q_1} = 0$$

SAT is NP-complete



- State q_1 and read a: write b, move to the right
- State q_1 and read b:
  - write c, enter q_2, move to the left, or
  - write a, enter q_3, move to the right

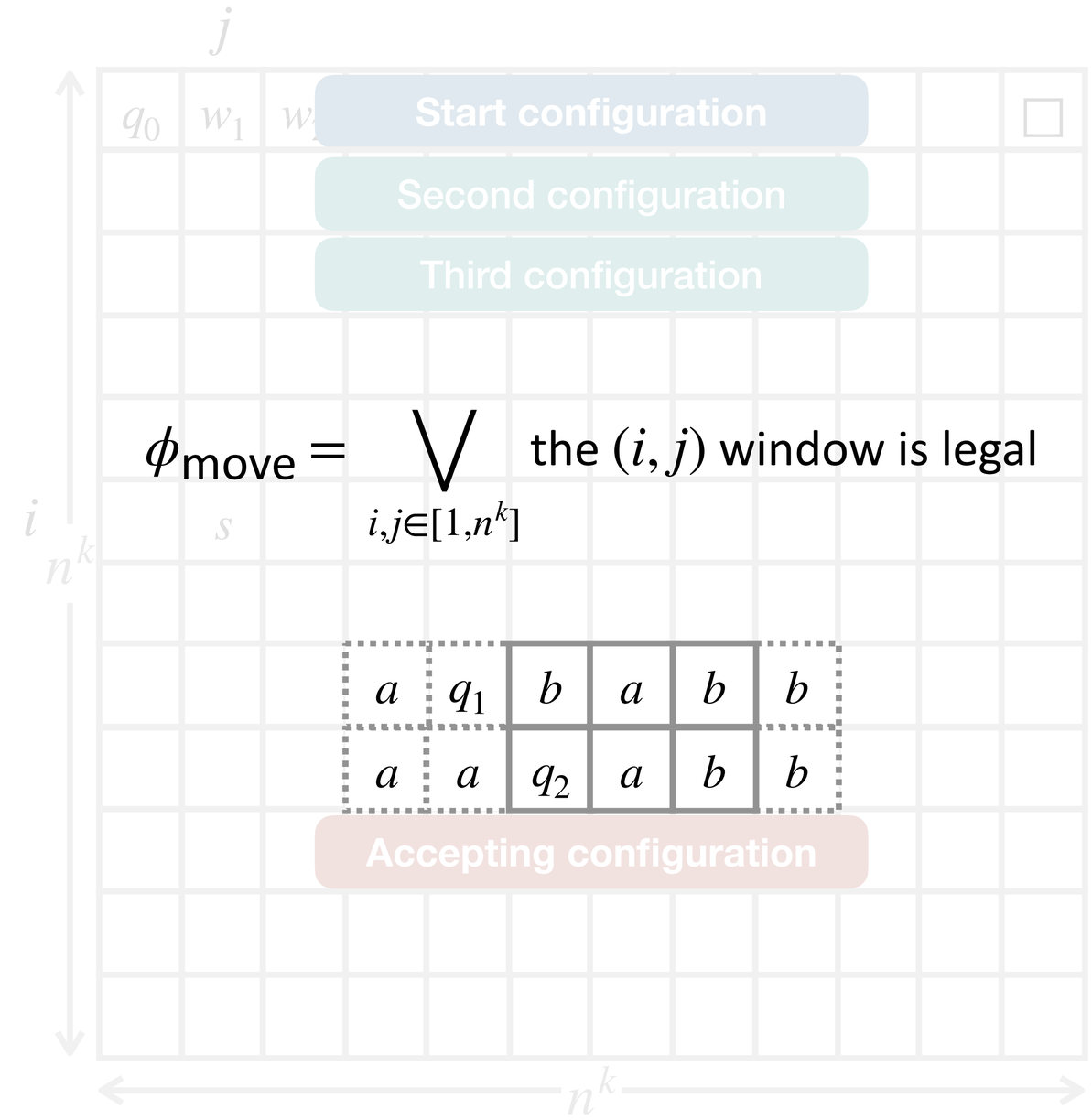


4. **The configurations corresponding to consecutive rows follow the NTM's rules**

$x_{row,\ column,\ symbol} = 1$ if the cell$[i, j]$ is the symbol

$$x_{1,2,w_1} = 1, x_{i,j,s} = 1, x_{1,1,q_1} = 0$$

SAT is NP-complete



- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. Each row in the table is a configuration of the NTM
  2. The first row is the starting configuration
  3. There is a accepting configuration
  4. **The configurations corresponding to consecutive rows follow the NTM's rules**

$x_{row,\ column,\ symbol} = 1$ if the cell$[i,j]$ is the symbol

$$x_{1,2,w_1} = 1, x_{i,j,s} = 1, x_{1,1,q_1} = 0$$

SAT is NP-complete

$$\phi_{\text{cell}} = \bigwedge_{i,j \in [1,n^k]} \left(\left(\bigvee_{\text{state } q_s} x_{i,j,w_s} \right) \wedge \left(\bigwedge_{\text{states } q_s \neq q_t} \left(\overline{x_{i,j,q_s} \wedge x_{i,j,q_t}} \right) \right) \right)$$

$$\phi_{\text{start}} = x_{1,1,q_0} \wedge x_{1,2,w_1} \wedge x_{1,3,w_2} \wedge \cdots \wedge x_{1,n^k,\square}$$

$$\phi_{\text{accept}} = \bigvee_{i,j \in [1,n^k]} x_{i,j,q_{\text{accept}}}$$

$$\phi_{\text{move}} = \bigvee_{i,j \in [1,n^k]} \text{the } (i,j) \text{ window is legal}$$

Time for construction:

ϕ_{start}: $O(n^k)$ ϕ_{cell}, ϕ_{accept}, and ϕ_{move}: $O(n^{2k})$

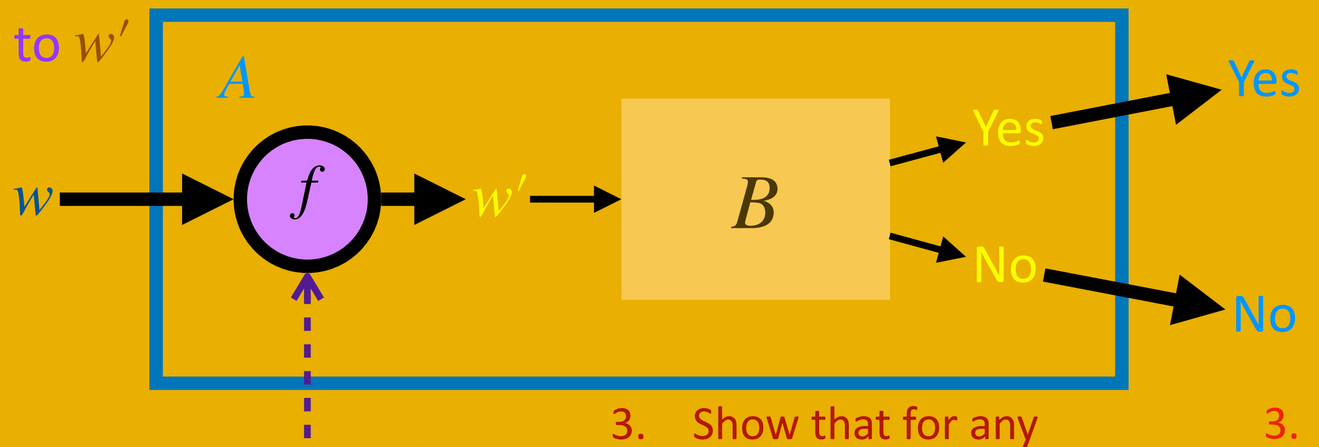
- If language A is in NP, there is a non-deterministic Turing machine (NTM) that accepts $w \in A$ in $O(n^k)$ steps
- There is a table with size $n^k \times n^k$ such that
  1. Each row in the table is a configuration of the NTM
  2. The first row is the starting configuration
  3. There is a accepting configuration
  4. The configurations corresponding to consecutive rows follow the NTM's rules

$w \in A$ iff $\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$ is satisfiable

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