CS251

Great Ideas in *Theoretical* Computer Science

P vs NP, Round 2

Quick review

GOAL: Understand the divide between efficiently computable and **not** efficiently computable.

testing primality

matrix multiplication

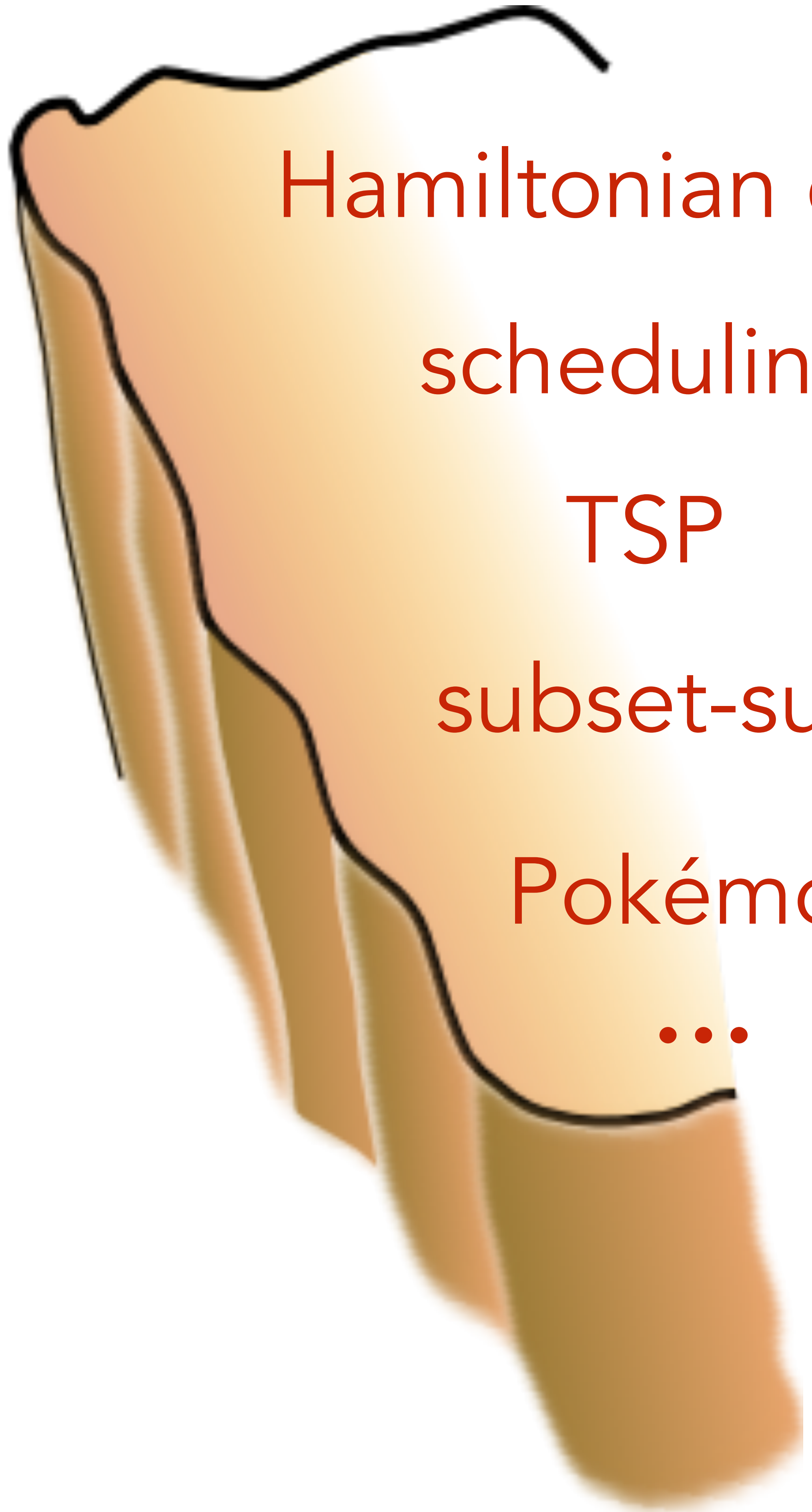
MST

max matching

shortest path

...

poly-time solvable



Hamiltonian cycle

scheduling

TSP

subset-sum

Pokémon

...

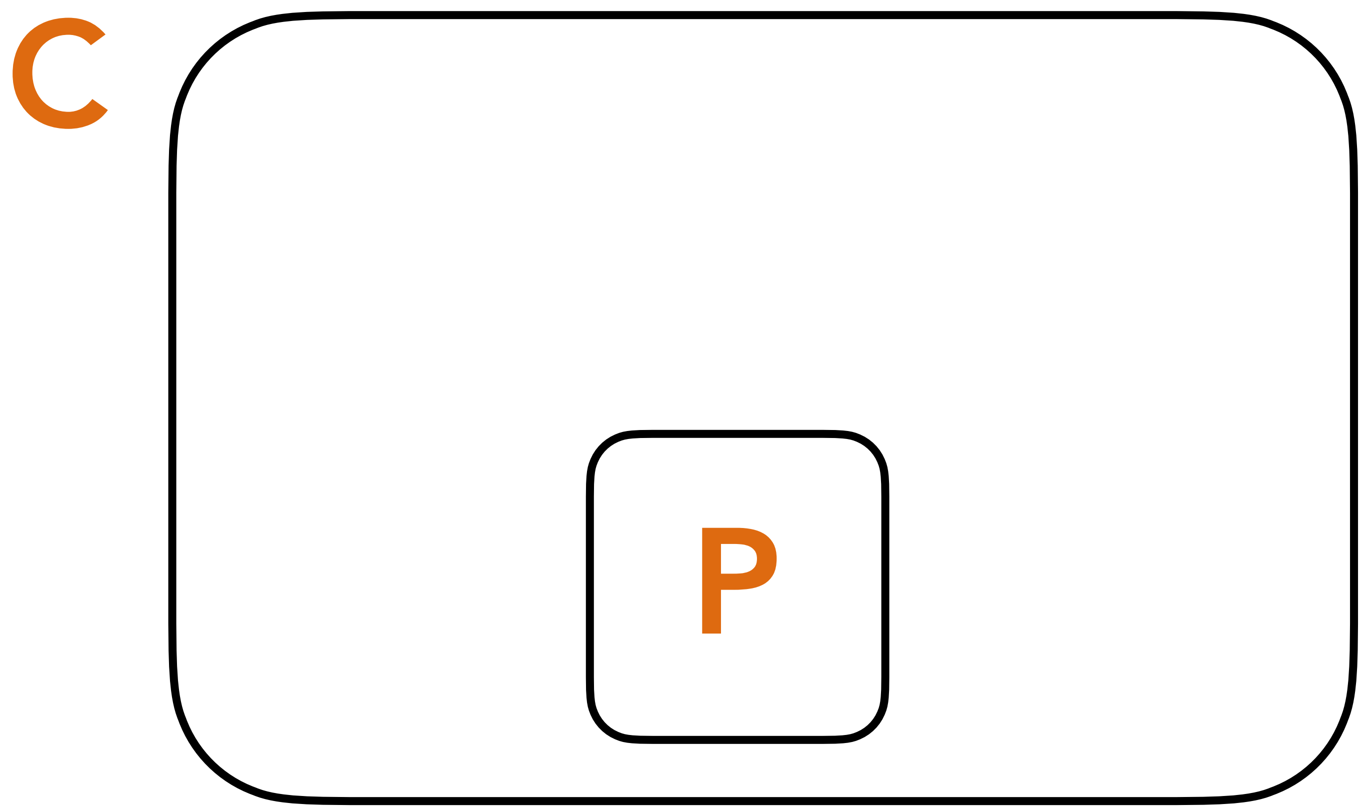
best we can say:

exp-time solvable

A reality we have to deal with:

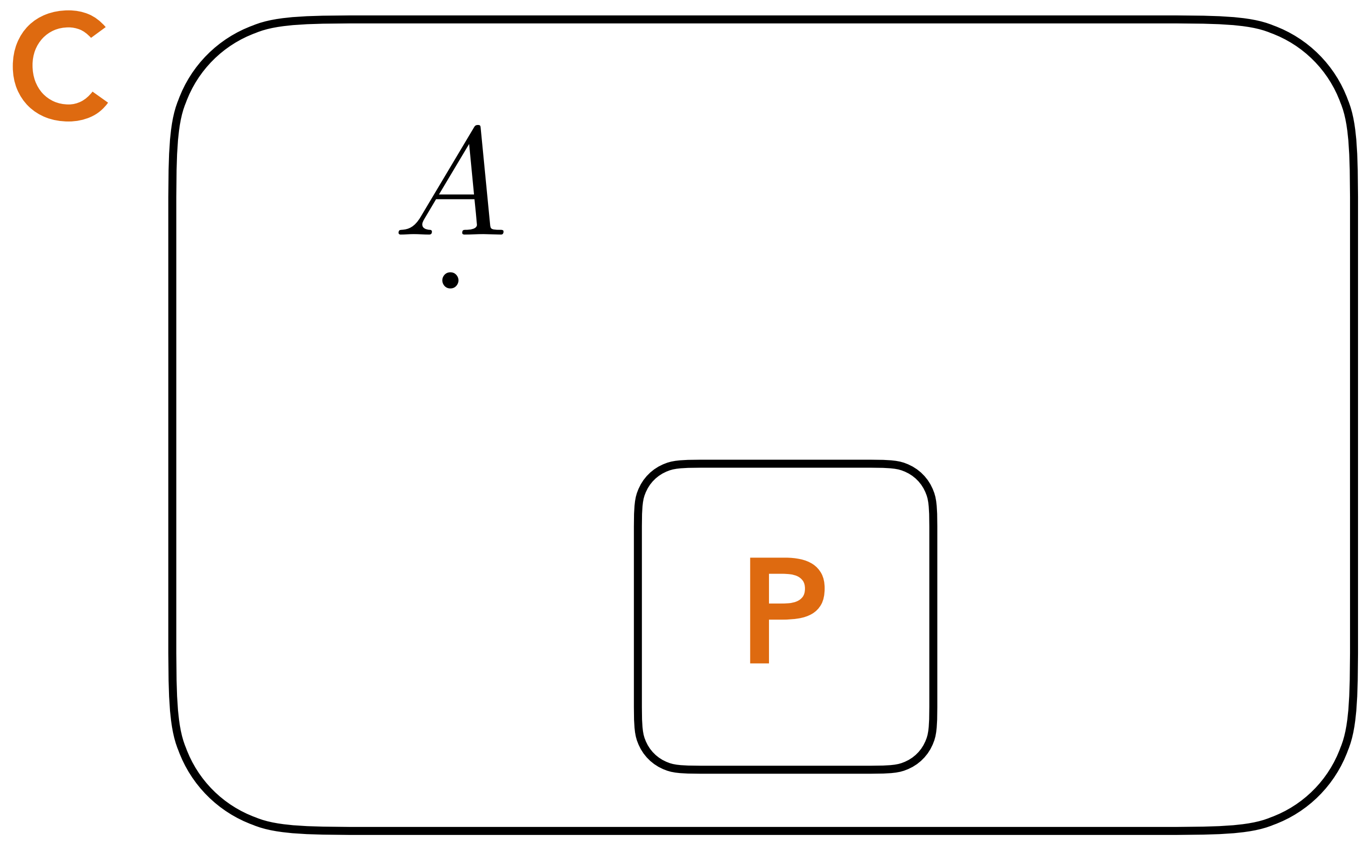
We suck at proving lower bounds…

If A is **C**-hard for a big class **C**,
that's good evidence that $A \notin$ **P**.



$\leq^P A$

A is **C**-hard



$\leq^P A$

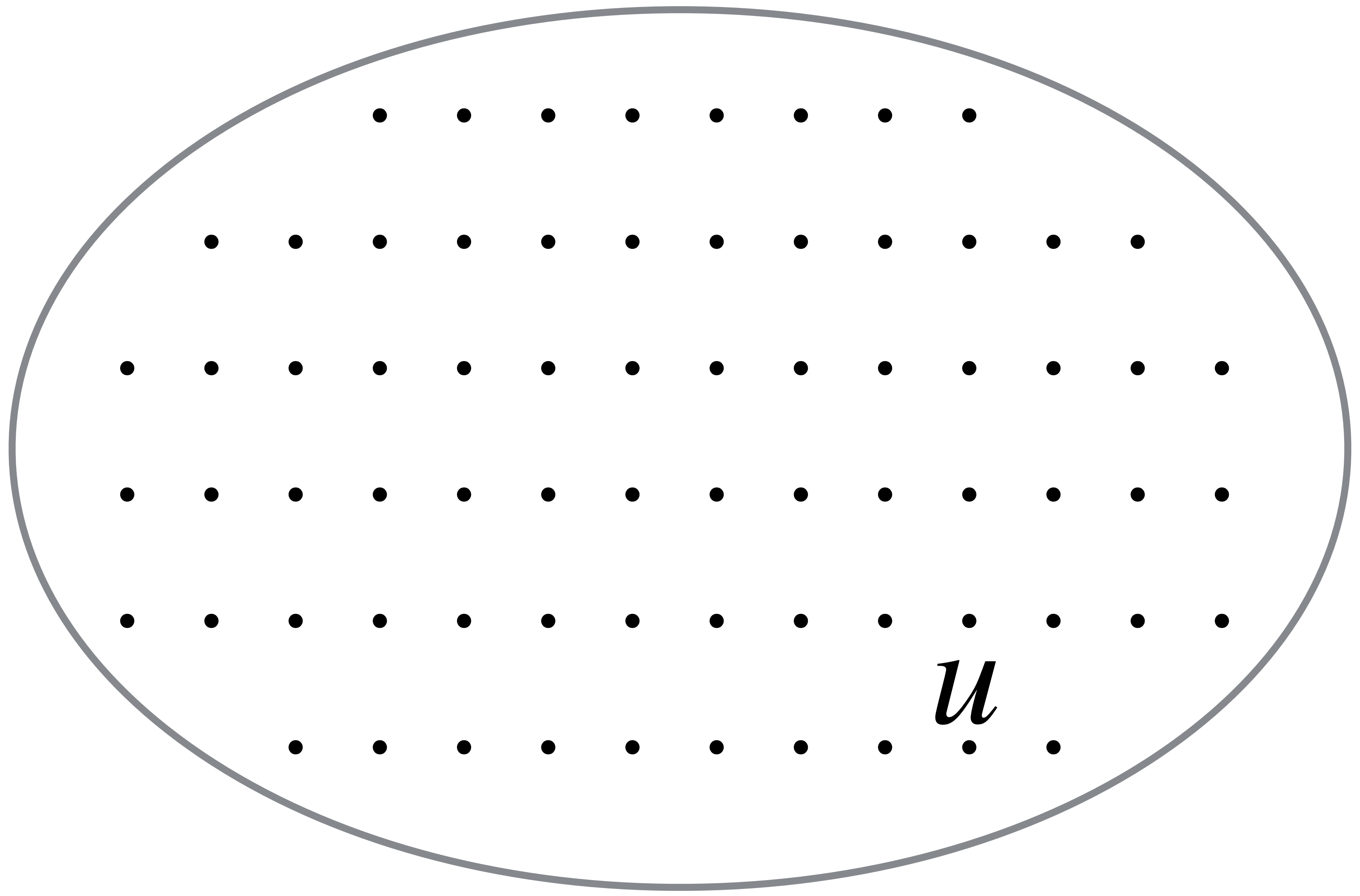
A is **C**-complete

$A \in \mathbf{P} \iff \mathbf{C} = \mathbf{P}$

But what is a good choice for **C**?

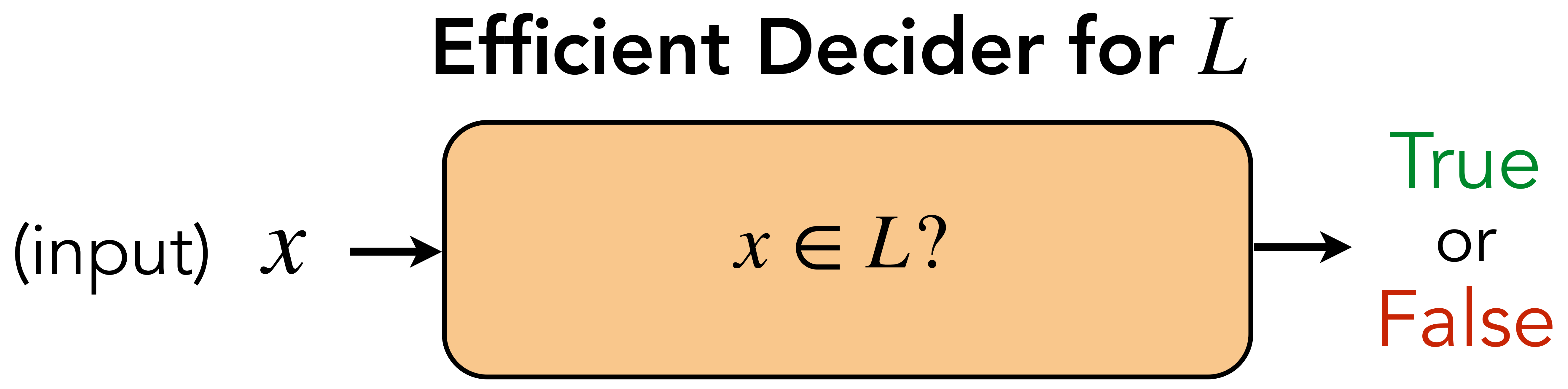
Which languages L are in **NP**?

- Every input x induces (at most) exponentially large "*possible solutions space*".

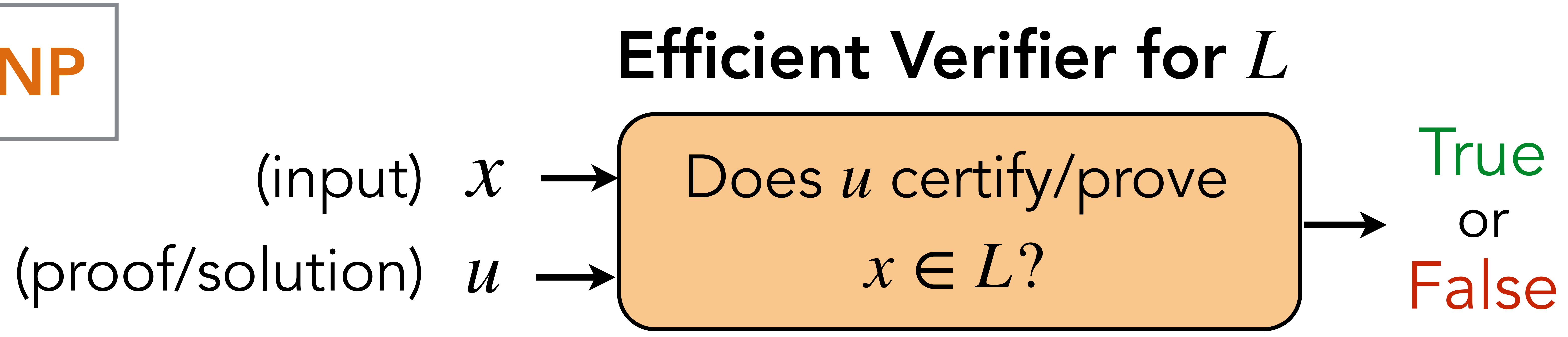


- If $x \in L$, there exists a solution u (certifying $x \in L$).
- If $x \notin L$, there is no solution.
- Easy (poly-time) to verify whether a possible solution is a solution.

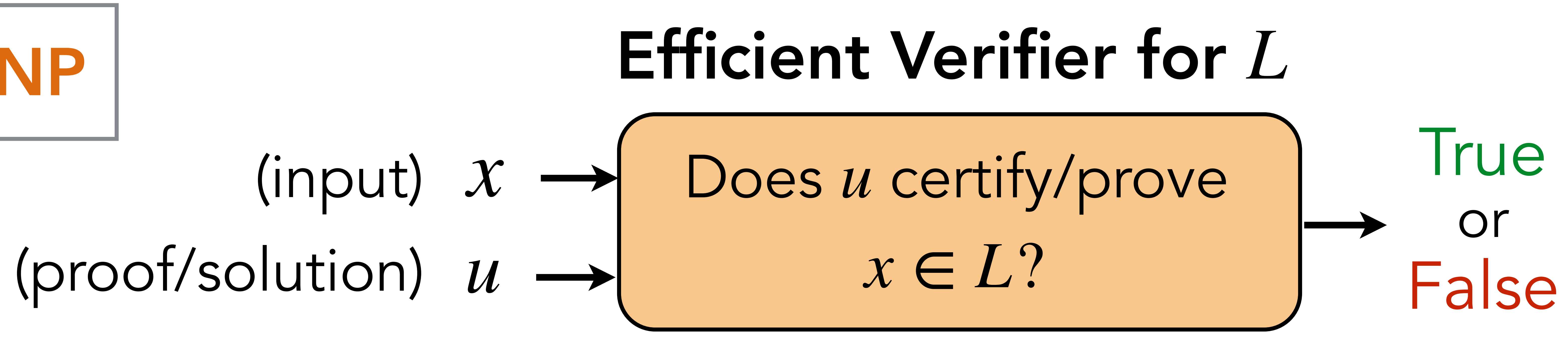
$L \in$ **P**


Efficient Decider for L
(input) x
x ∈ L?
True
or
False

$L \in$ **NP**


Efficient Verifier for L
(input) x
(proof/solution) u
Does u certify/prove
x ∈ L?
True
or
False

$L \in$ **NP**



Definition: A language L is in **NP** if

- there is a polynomial-time TM V,
- a constant k,

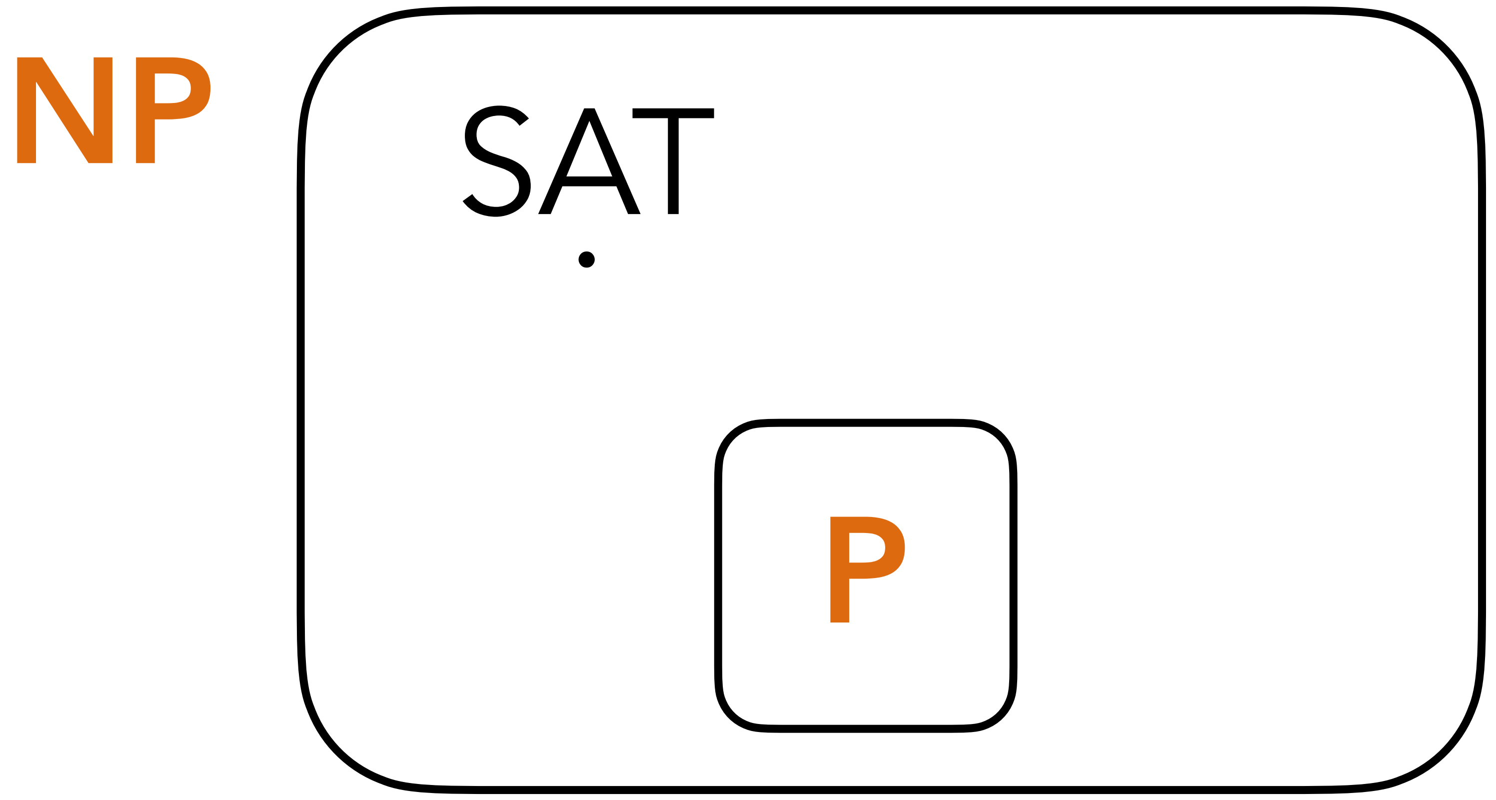
such that:

$$x \in L \implies \exists u \text{ with } |u| \leq |x|^k \text{ s.t. } V(x, u) \text{ accepts},$$

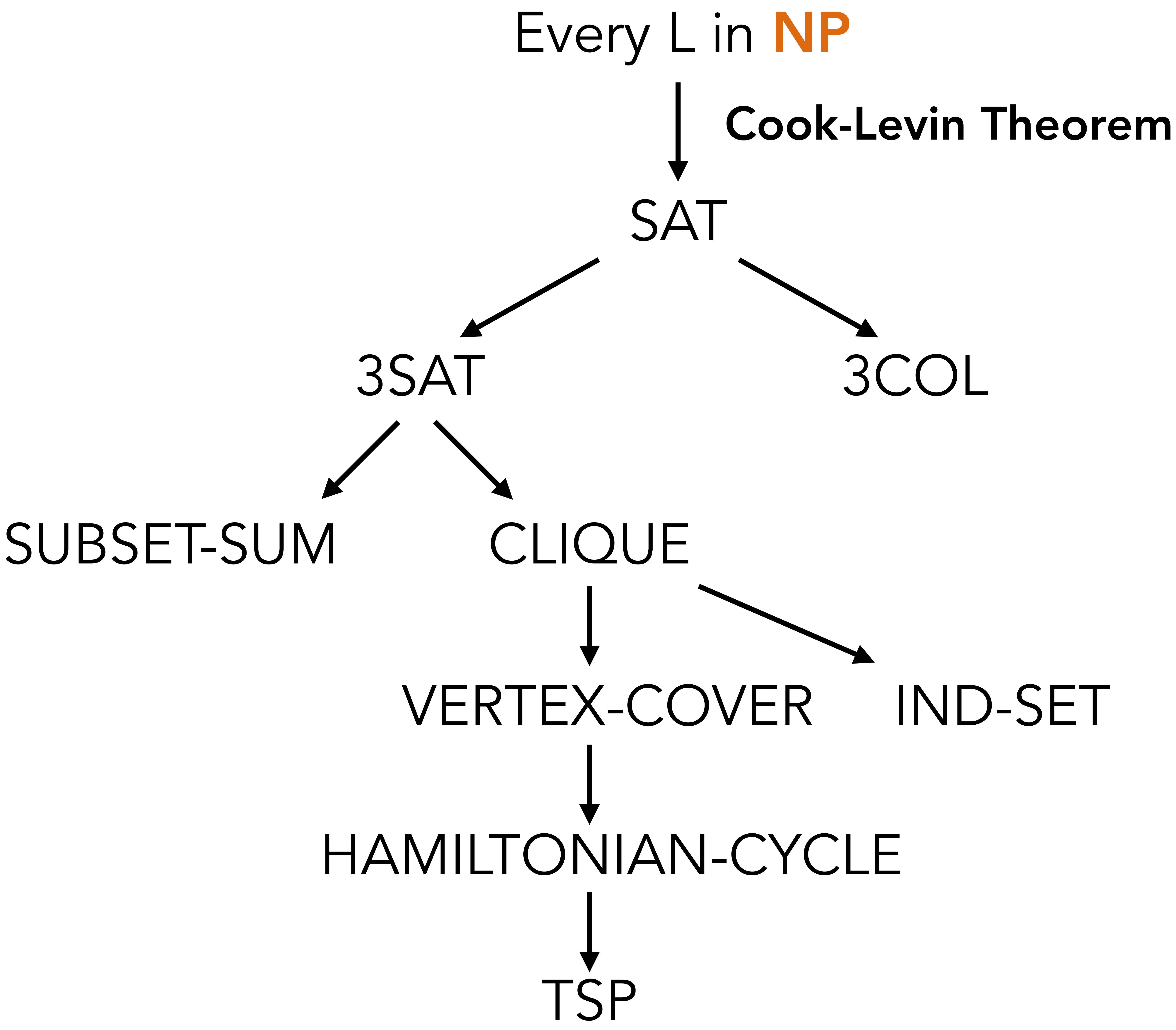
$$x \notin L \implies \forall u, V(x, u) \text{ rejects}.$$

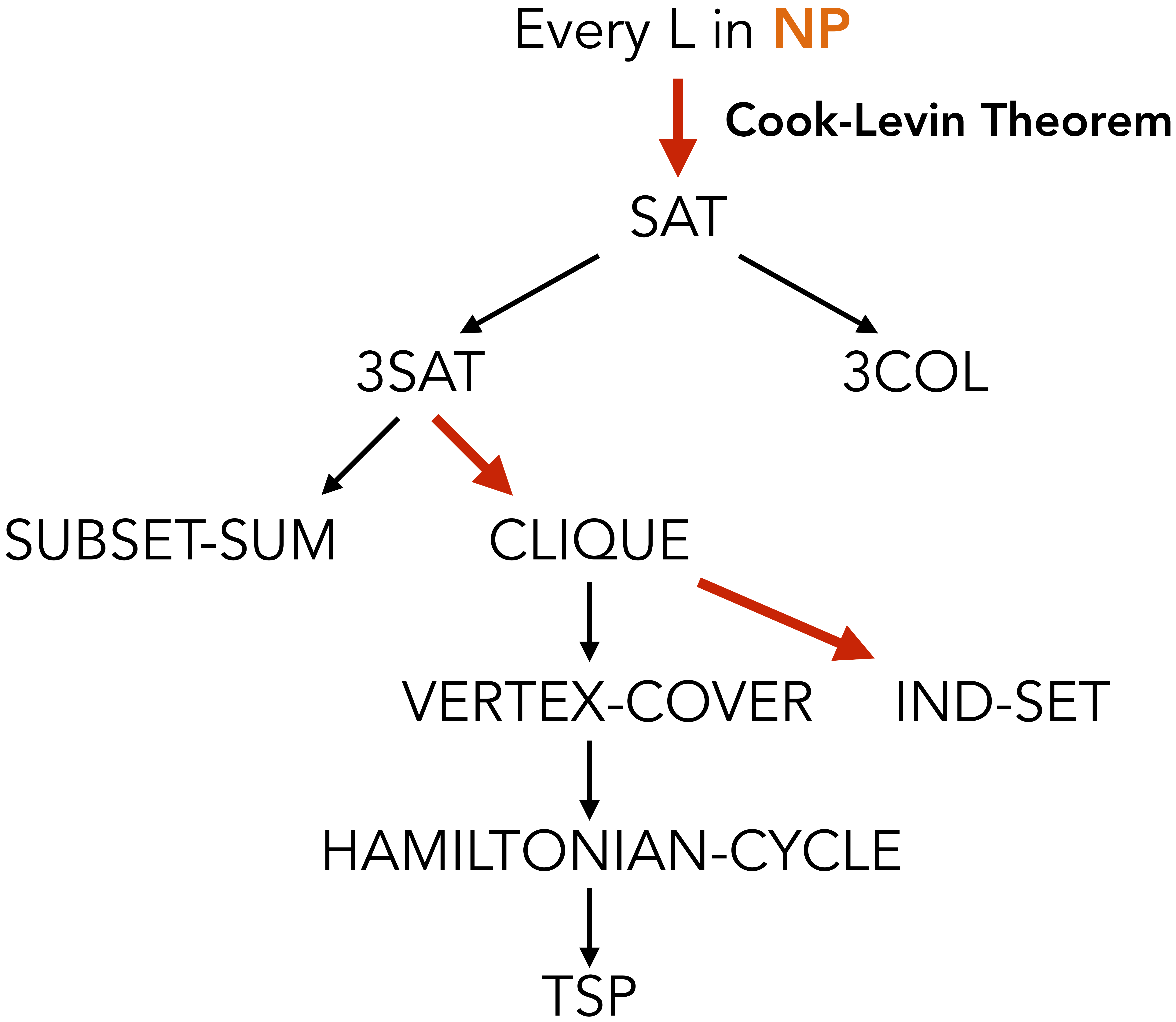
NP-hardness, **NP**-completeness

Cook-Levin Theorem:



$\leq^P$ SAT


Every L in NP
Cook-Levin Theorem
SAT
3SAT
3COL
SUBSET-SUM
CLIQUE
VERTEX-COVER
IND-SET
HAMILTONIAN-CYCLE
TSP

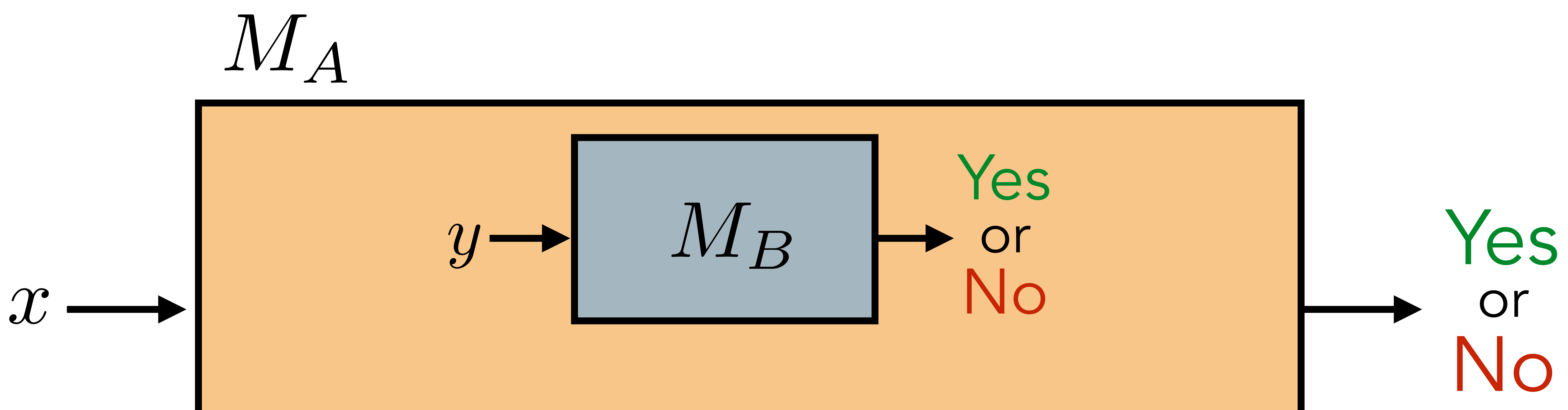

Every L in NP
Cook-Levin Theorem
SAT
3SAT
3COL
SUBSET-SUM
CLIQUE
VERTEX-COVER
IND-SET
HAMILTONIAN-CYCLE
TSP

A note about reductions

Cook reductions: Poly-time Turing reductions

$A \leq^P B$

Solve A in poly-time using a blackbox that solves B.



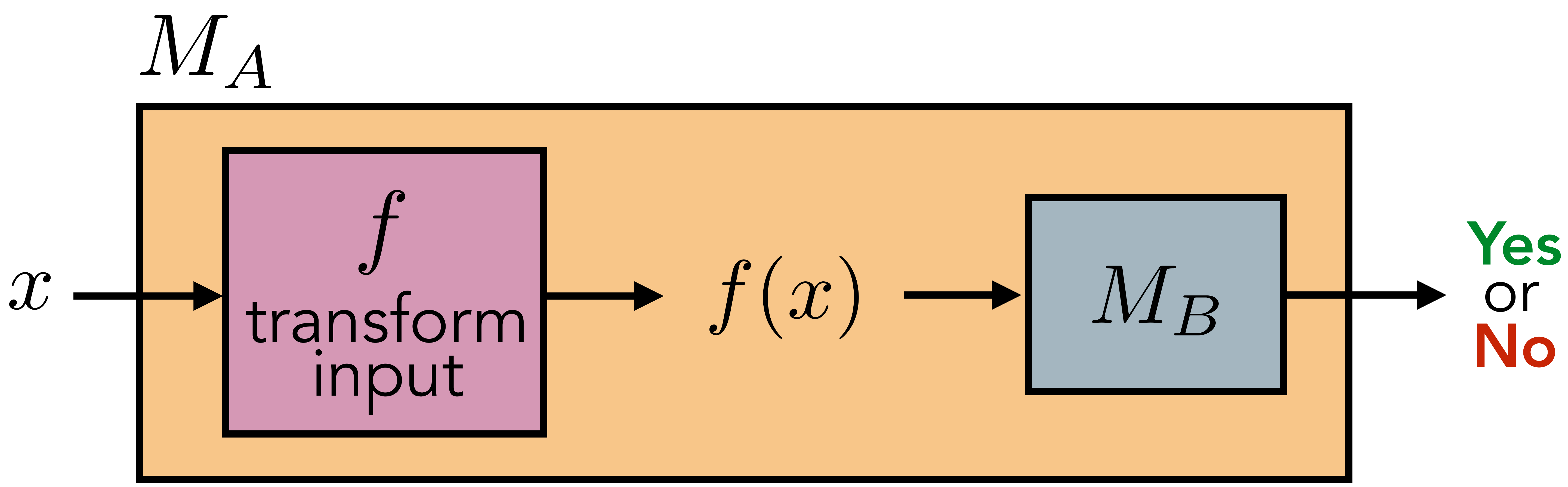
Can call M_B poly($|x|$) times.

B poly-time decidable $\implies$ A poly-time decidable

Karp reductions: Poly-time mapping reductions

$A \leq_m^P B$

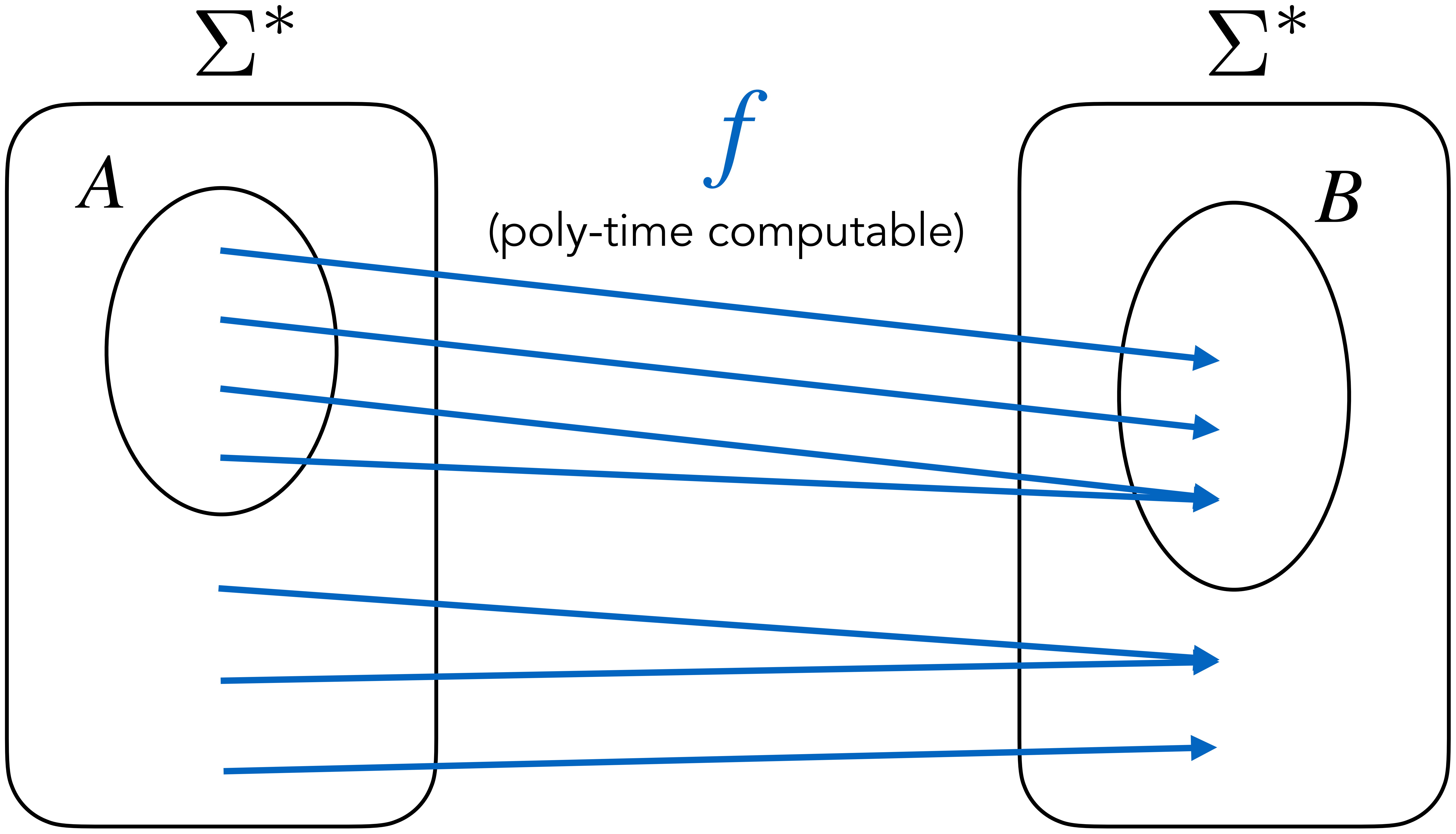
Make **one** call to M_B. Directly use its answer as output.



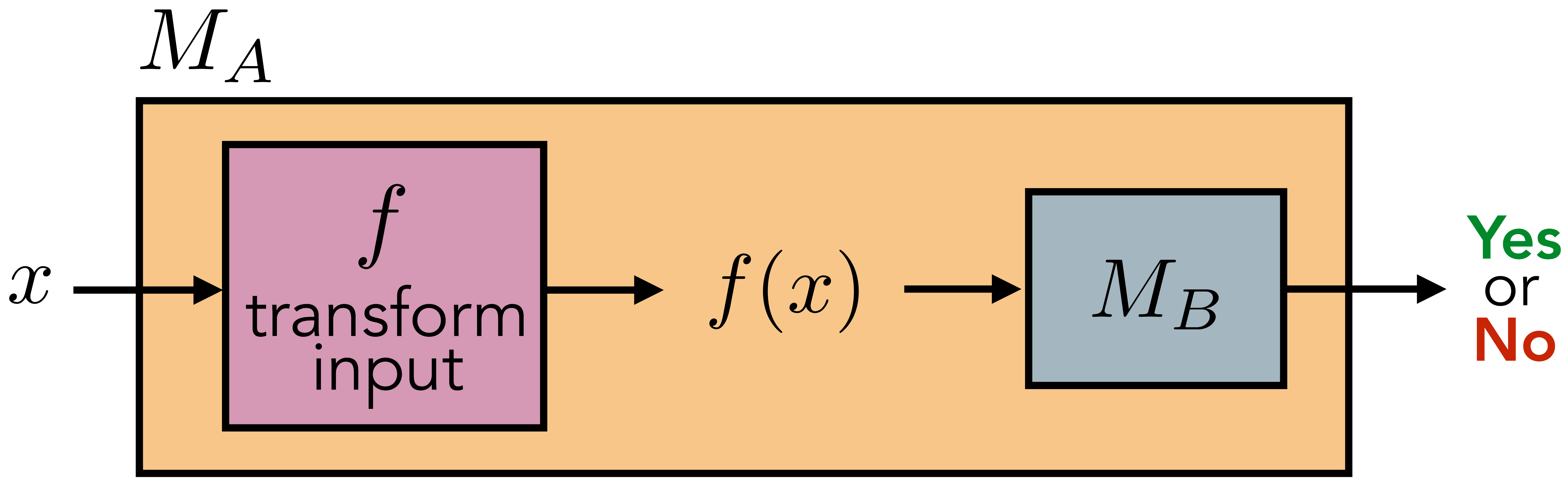
<u>We must have:</u> $x \in A \iff f(x) \in B$

Karp reductions: Poly-time mapping reductions

<u>We must have:</u> $x \in A \iff f(x) \in B$



Karp reductions: Poly-time mapping reductions



To show $A \leq_m^P B$:

1. **Define**: $f : \Sigma^* \to \Sigma^*$.
2. **Show**: $x \in A \iff f(x) \in B$.
3. **Show**: f is computable in poly-time.

Cook vs Karp

Can define **NP**-hardness with respect to $\leq^P$.

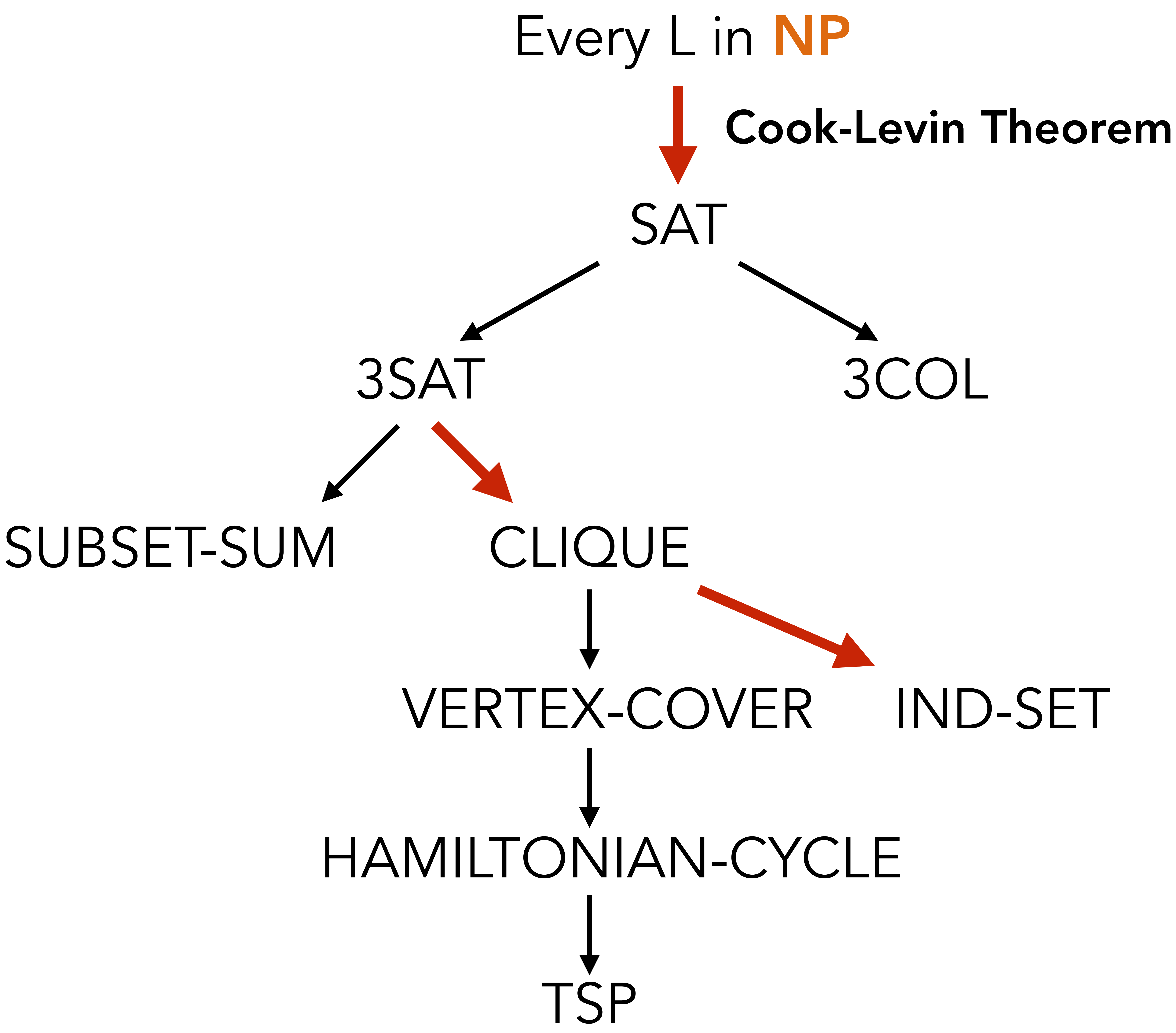
(what some courses use for simplicity)

Can define **NP**-hardness with respect to $\leq_m^P$.

(what experts use)

These lead to different notions of **NP**-hardness.

! In CS251, we'll use Karp reductions $\leq_m^P$.


Every L in NP
Cook-Levin Theorem
SAT
3SAT
3COL
SUBSET-SUM
CLIQUE
VERTEX-COVER
IND-SET
HAMILTONIAN-CYCLE
TSP

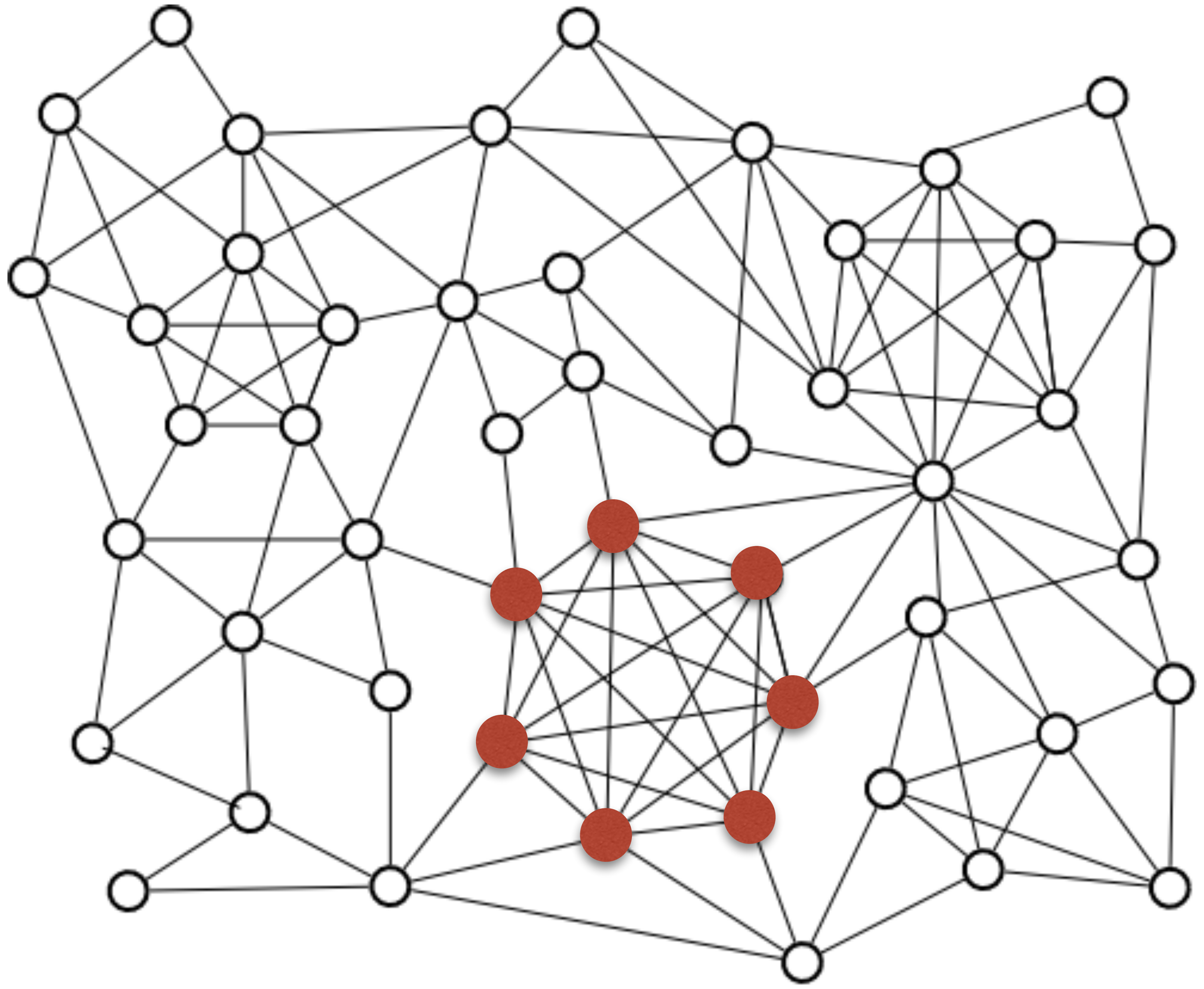
CLIQUE reduces to IND-SET

Karp reduction example: CLIQUE ≤ IND-SET

CLIQUE

Input: $\langle G, k \rangle$ where G is a graph and k is a positive int.

Output: True iff G contains a **clique** of size k.

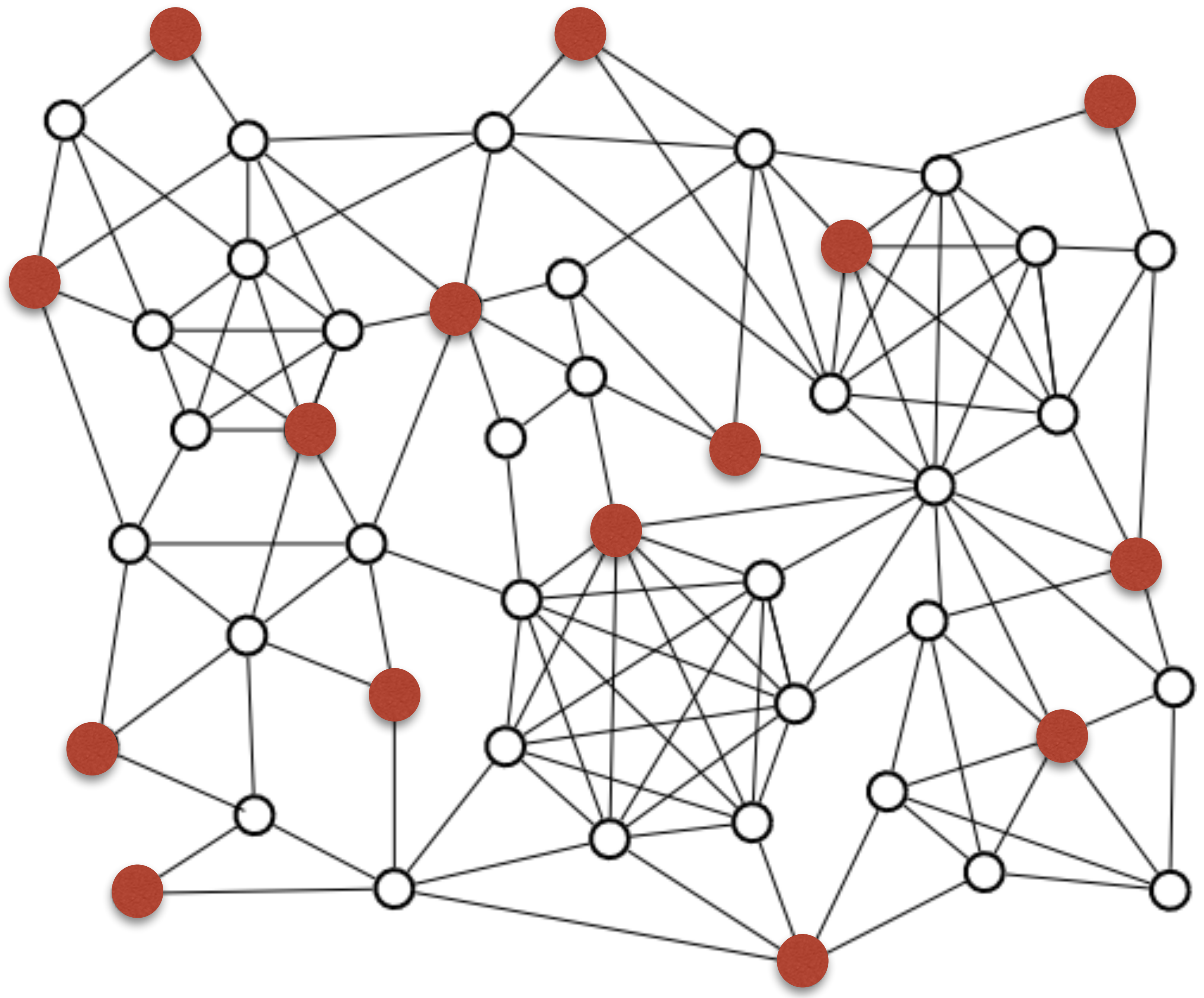


Karp reduction example: CLIQUE ≤ IND-SET

IND-SET

Input: $\langle G, k \rangle$ where G is a graph and k is a positive int.

Output: True iff G contains an **independent set** of size k.



Karp reduction example: CLIQUE ≤ IND-SET

Fact: CLIQUE $\leq_m^P$ IND-SET.

Karp reduction example: CLIQUE ≤ IND-SET

We need to:

1. **Define**: $f : \Sigma^* \to \Sigma^*$.
2. **Show**: $w \in$ CLIQUE $\iff f(w) \in$ IND-SET.
3. **Show**: f is computable in poly-time.

$$\langle G, k \rangle \stackrel{f}{\mapsto} \langle G', k' \rangle$$

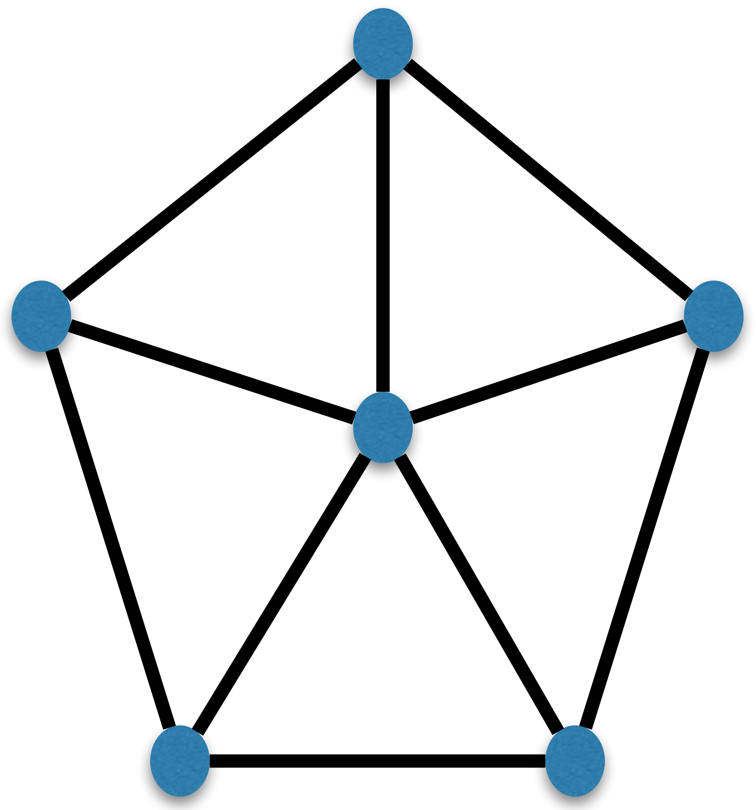
G has a clique of size k **iff** G' has an ind. set of size k'

Karp reduction example: CLIQUE ≤ IND-SET

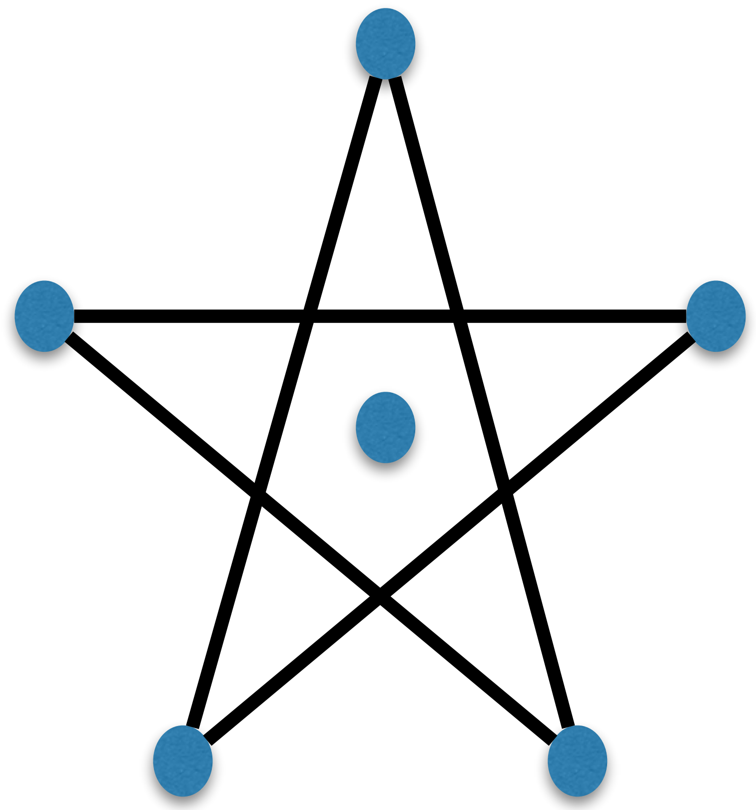
$$\langle G, k \rangle \overset{f}{\mapsto} \langle G', k' \rangle$$

G has a clique of size k **iff** G' has an ind. set of size k'

G



G'



This is called the complement of G.

Karp reduction example: CLIQUE ≤ IND-SET

1. **Define**: $f : \Sigma^* \to \Sigma^*$.

def $f(\langle G = (V, E), k \rangle)$:

- Let $E' = \{\{u, v\} : u, v \in V, \{u, v\} \notin E\}$.

- Return $\langle G' = (V, E'), k \rangle$.

$$\langle G, k \rangle \mapsto \langle G', k \rangle$$

Implicit type-checker:

not valid encoding $\mapsto$ a string not in IND-SET (e.g. ϵ)

Karp reduction example: CLIQUE ≤ IND-SET

2. **Show:** $w \in$ CLIQUE $\iff$ $f(w) \in$ IND-SET.

$w \in$ CLIQUE

$\iff$

$w = \langle G = (V, E), k \rangle$ and G has a clique $S \subseteq V$ of size k.

$\iff$

In $G' = (V, E')$, $S \subseteq V$ is an ind. set of size k.

$\iff$

$f(w) = \langle G' = (V, E'), k \rangle \in$ IND-SET.

Karp reduction example: CLIQUE ≤ IND-SET

3. **Show**: f is computable in poly-time.

Creating E', and therefore G', can be done in poly-time.

$\blacksquare$

Poll Question

kCOL Problem

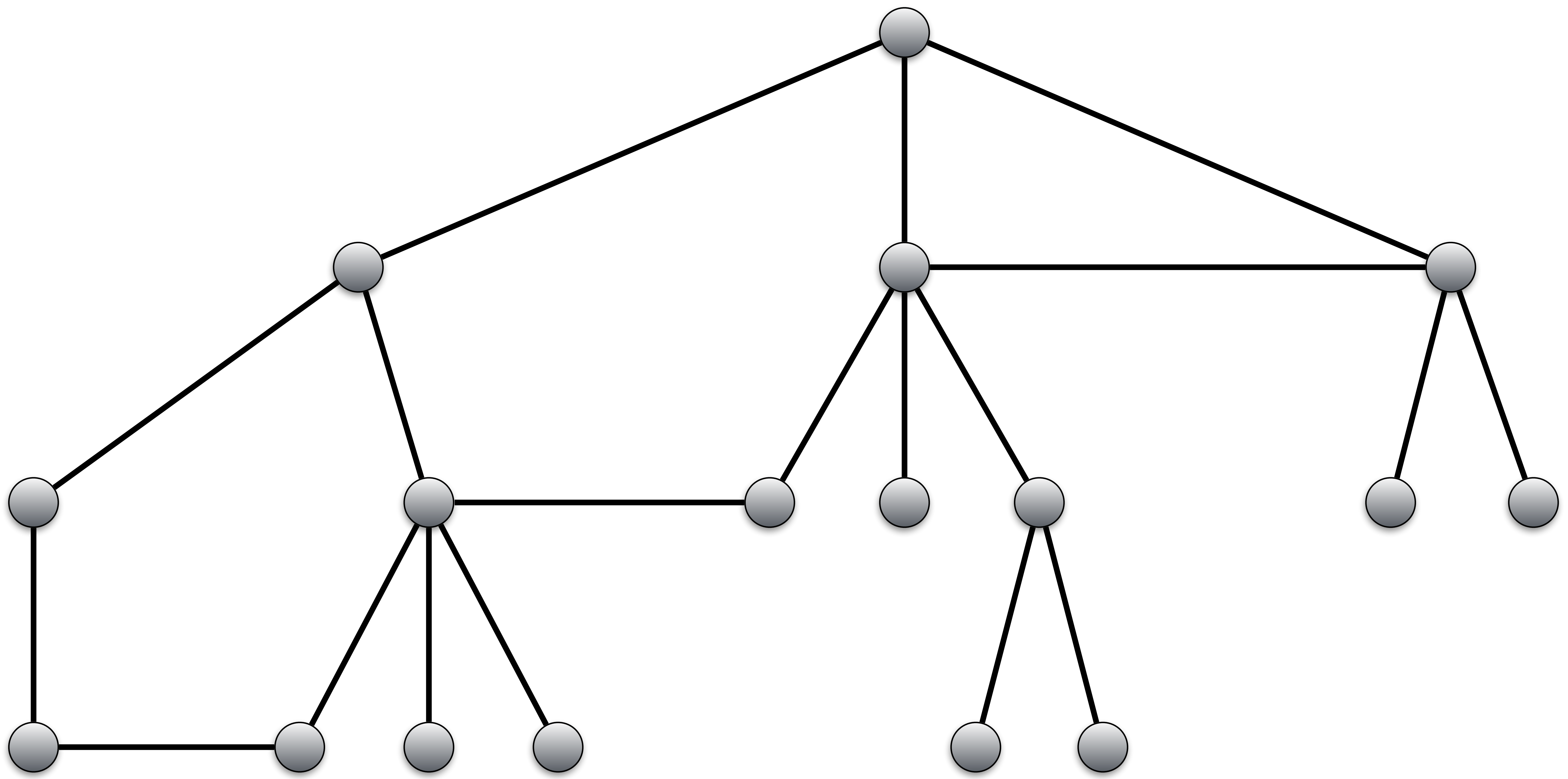
Input: A graph G.

Output: Yes/True if it is possible to color the vertices with k colors such that every edge is bichromatic (the endpoints have different colors).

3COL Problem

Input: A graph G.

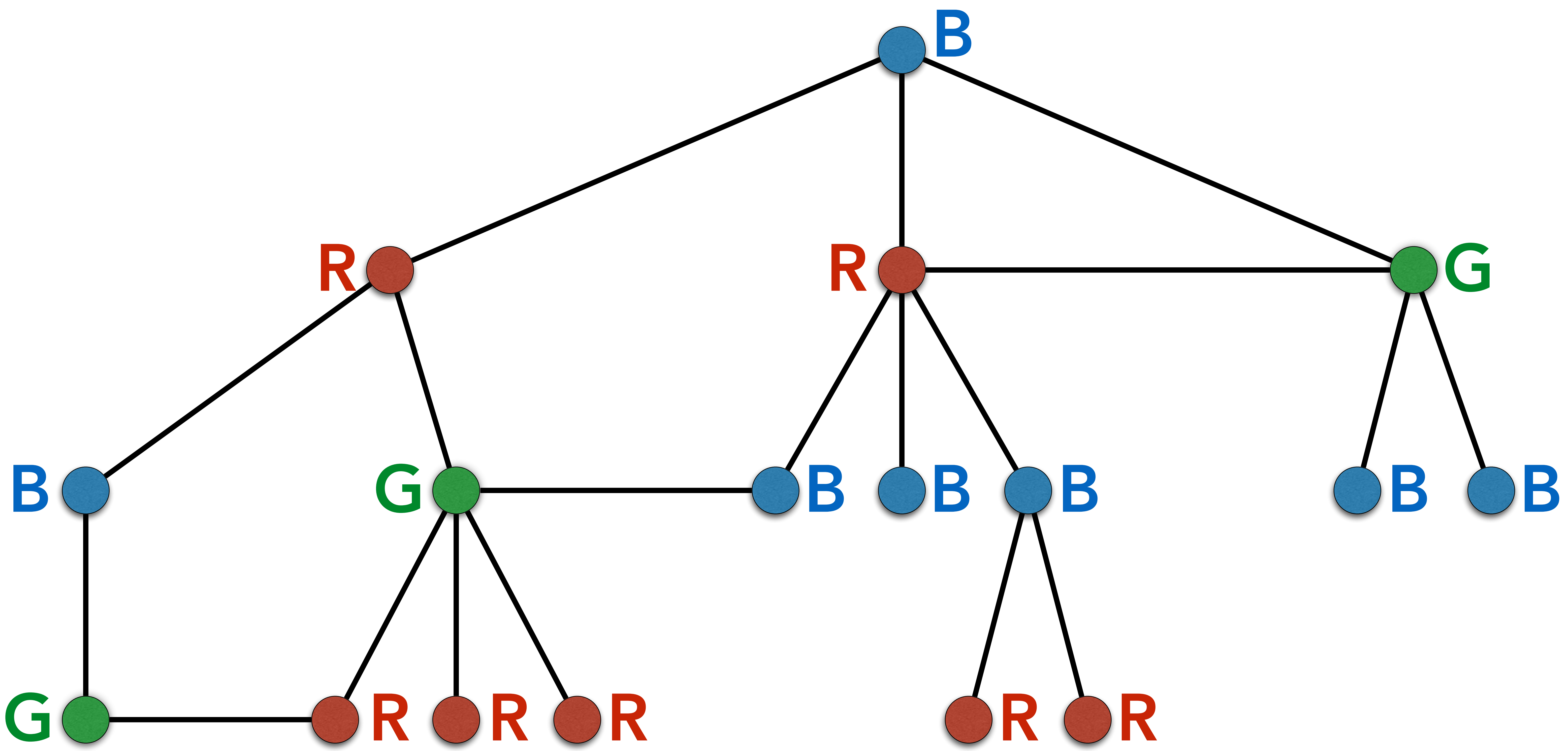
Output: Yes/True if it is possible to color the vertices with 3 colors such that every edge is bichromatic (the endpoints have different colors).



3COL Problem

Input: A graph G.

Output: Yes/True if it is possible to color the vertices with 3 colors such that every edge is bichromatic (the endpoints have different colors).

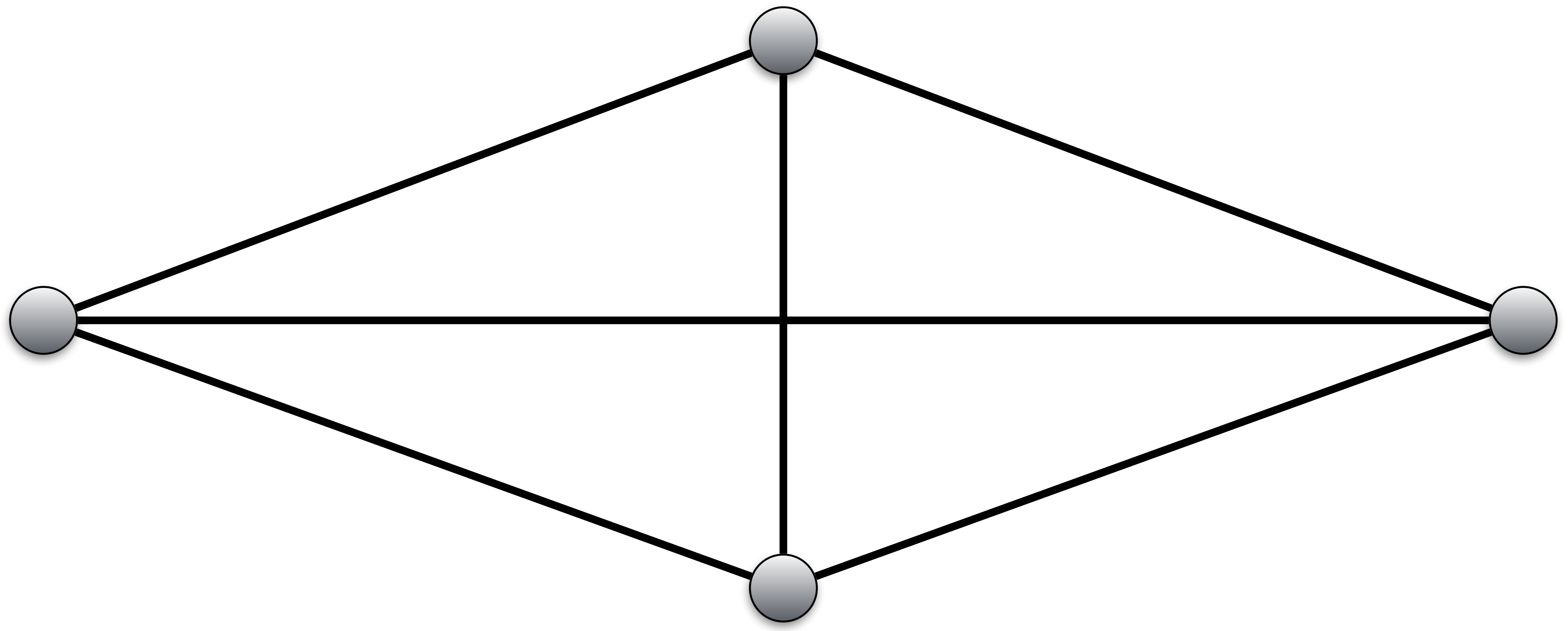


3COL Problem

Input: A graph G.

Output: Yes/True if it is possible to color the vertices with 3 colors such that every edge is bichromatic (the endpoints have different colors).

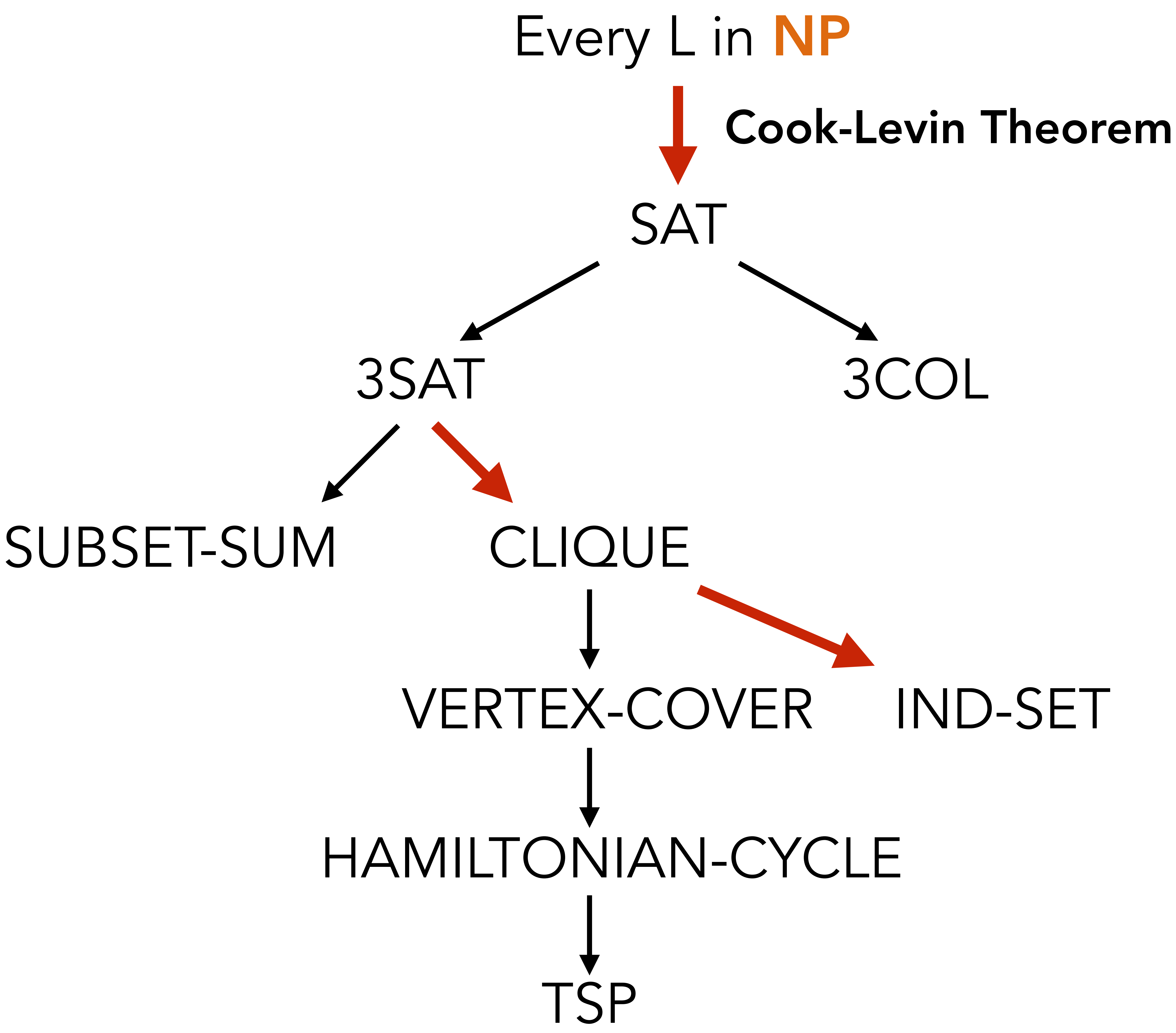
Not 3-colorable



Poll

2COL $\leq_m^P$ 3COL is true, false or open?

3COL $\leq_m^P$ 2COL is true, false or open?


Every L in NP
Cook-Levin Theorem
SAT
3SAT
3COL
SUBSET-SUM
CLIQUE
VERTEX-COVER
IND-SET
HAMILTONIAN-CYCLE
TSP

3SAT reduces to CLIQUE

Definition of 3SAT

Input: A Boolean formula in "conjunctive normal form" in which every clause has exactly 3 literals.

$$\underbrace{(x_1 \vee \neg x_2 \vee x_3)} \wedge (\neg x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee \neg x_5 \vee x_6)$$

a **clause**
(an OR of literals)

literal: a variable or its negation

conjunctive normal form: AND of clauses.

To satisfy a formula: Satisfy every single clause.

To satisfy a clause: Satisfy at least one literal in the clause.

Output: Yes iff the formula is satisfiable.

3SAT ≤ CLIQUE: High level steps

We need to:

1. **Define**: $f : \Sigma^* \to \Sigma^*$.
2. **Show:** $w \in \text{3SAT} \iff f(w) \in \text{CLIQUE}$.
3. **Show**: f is computable in poly-time.

Strategy:

w	$\mapsto$	$f(w)$
$\in$ 3SAT		$\in$ CLIQUE
proof (solution)	$\leftrightarrow$	**proof** (solution)

3SAT: What is a "good" proof?

$$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee \neg x_5 \vee x_6)$$

φ satisfiable

$\iff$

can pick one literal from each clause and set them to True

$\iff$

the sequence of literals picked does not contain both a variable and its negation.

What is a "good" proof that $\langle \varphi \rangle \in$ 3SAT ?

- a truth assignment to the variables that satisfies the formula.
- a sequence of literals, one from each clause, not containing both a variable and its negation.

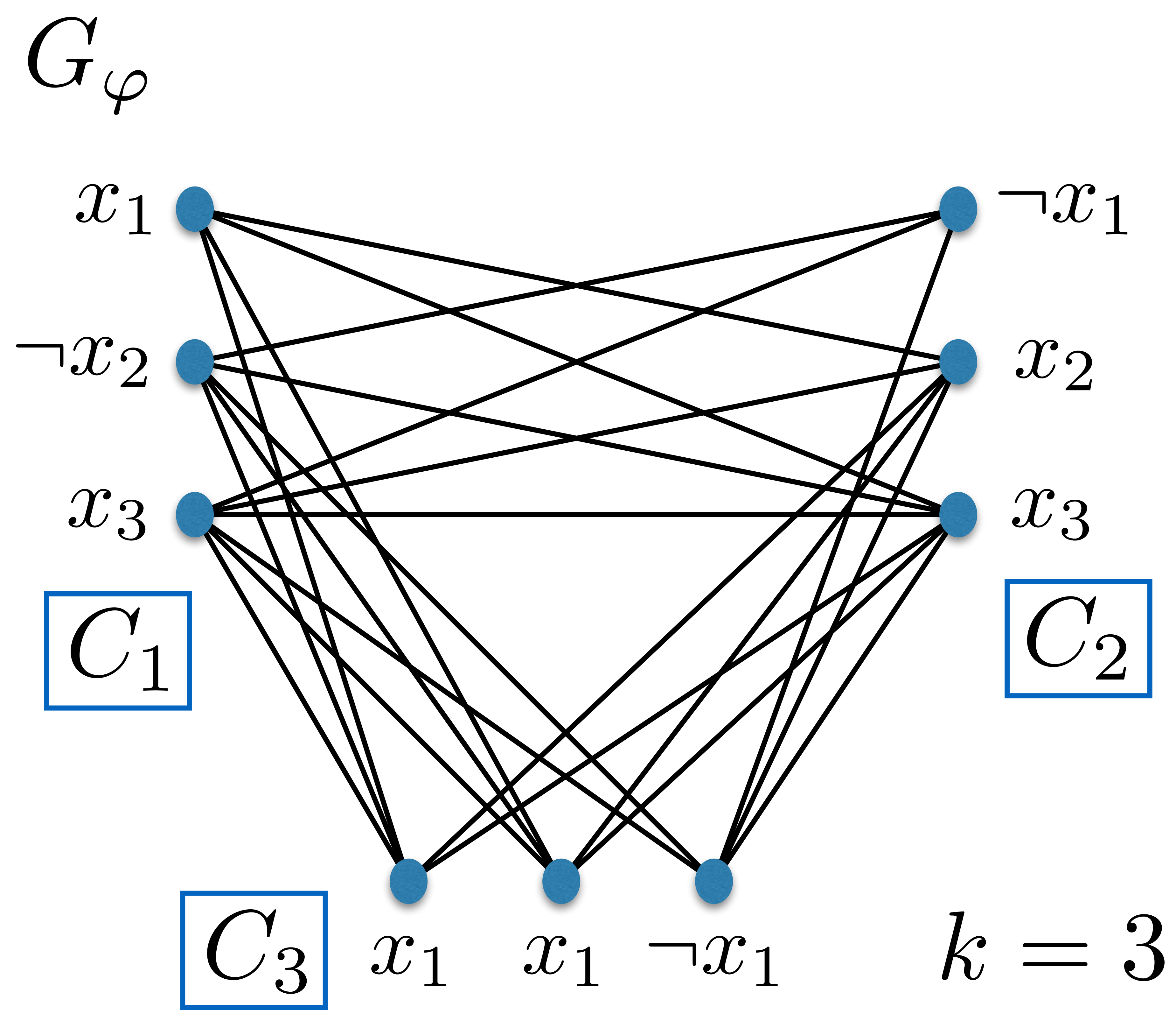
3SAT ≤ CLIQUE: Defining the map

1. **Define**: $f : \Sigma^* \to \Sigma^*$.

$\langle \varphi \rangle$ m clauses	$\mapsto$	$\langle G_\varphi, m \rangle$
proof	$\leftrightarrow$	**proof**
sequence of m literals, one from each clause, not containing a variable and its negation.	$\leftrightarrow$	clique of size m.

3SAT ≤ CLIQUE: Defining the map

$$\varphi = \underbrace{(x_1 \vee \neg x_2 \vee x_3)}_{C_1} \wedge \underbrace{(\neg x_1 \vee x_2 \vee x_3)}_{C_2} \wedge \underbrace{(x_1 \vee x_1 \vee \neg x_1)}_{C_3}$$



The construction:

- A vertex for each literal in each clause.
- No edges between two literals in same clause.
- No edges between x_i and $\neg x_i$ for any i.
- All other possible edges present.
- Set k to be # clauses in φ .

3SAT ≤ CLIQUE: Why it works

2. **Show:** $w \in \text{3SAT} \iff f(w) \in \text{CLIQUE}$.

$$w = \langle \varphi \rangle \qquad \mapsto \qquad f(w) = \langle G_\varphi, m \rangle$$

m clauses

φ satisfiable $\iff$ G_φ contains an m-clique

This is true because by construction:

proof $\leftrightarrow$ **proof**

3SAT ≤ CLIQUE: Why it works

2. **Show:** φ satisfiable $\iff$ G_φ contains an m-clique

φ is satisfiable

$\iff$

can pick m literals, one from each clause,
such that we don't pick a variable and its negation.

$\iff$

can pick m vertices in G_φ which are all connected
(by an edge).

$\iff$

G_φ contains an m-clique.

3SAT ≤ CLIQUE: Poly-time reduction

3. **Show**: f is computable in poly-time.

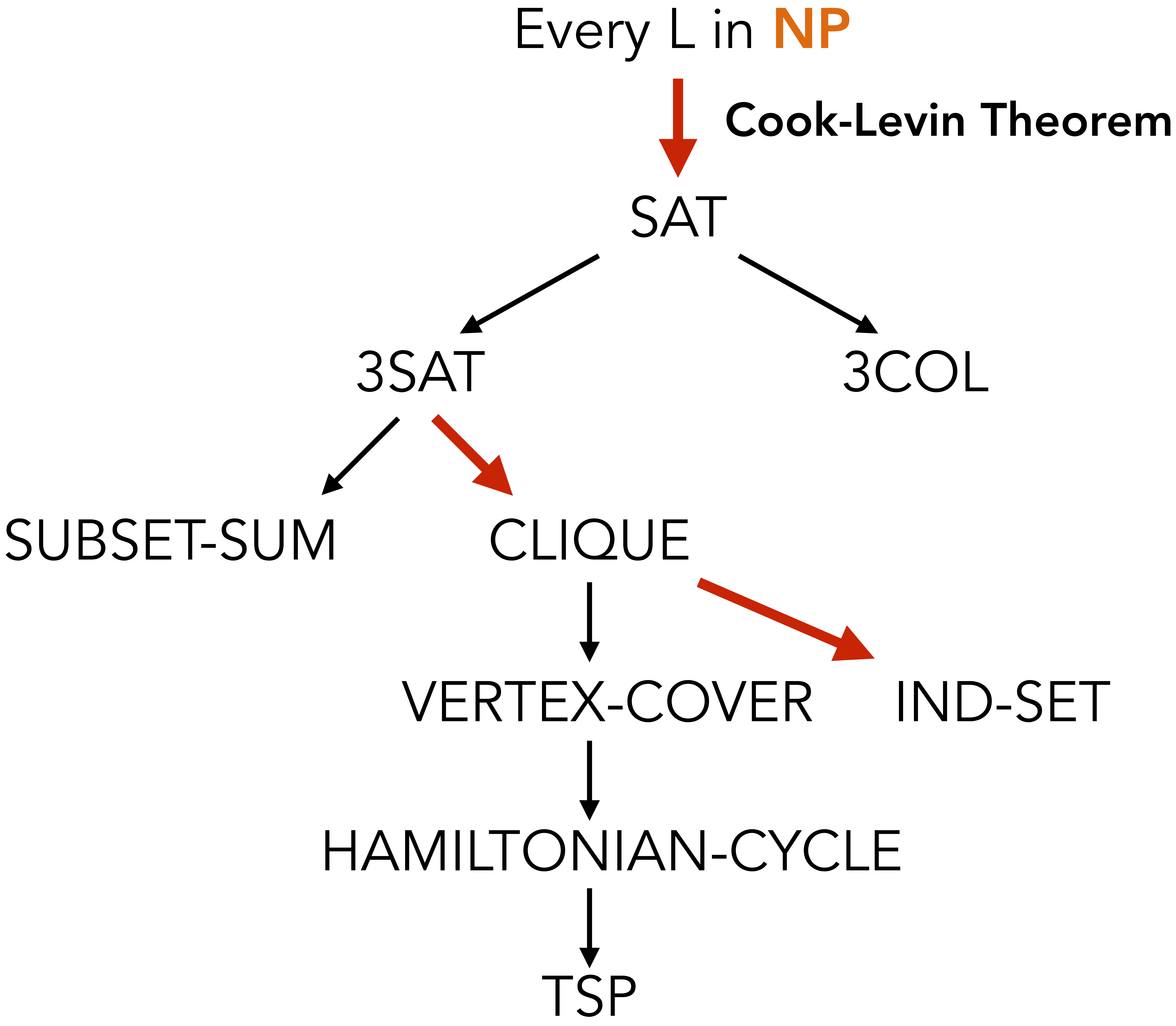
Creating the vertex set:

- there is just one vertex for each literal in each clause.
- scan input formula and create the vertex set.

Creating the edge set:

- there are at most $O(m^2)$ possible edges.
- determining if an edge should be present is polynomial time.

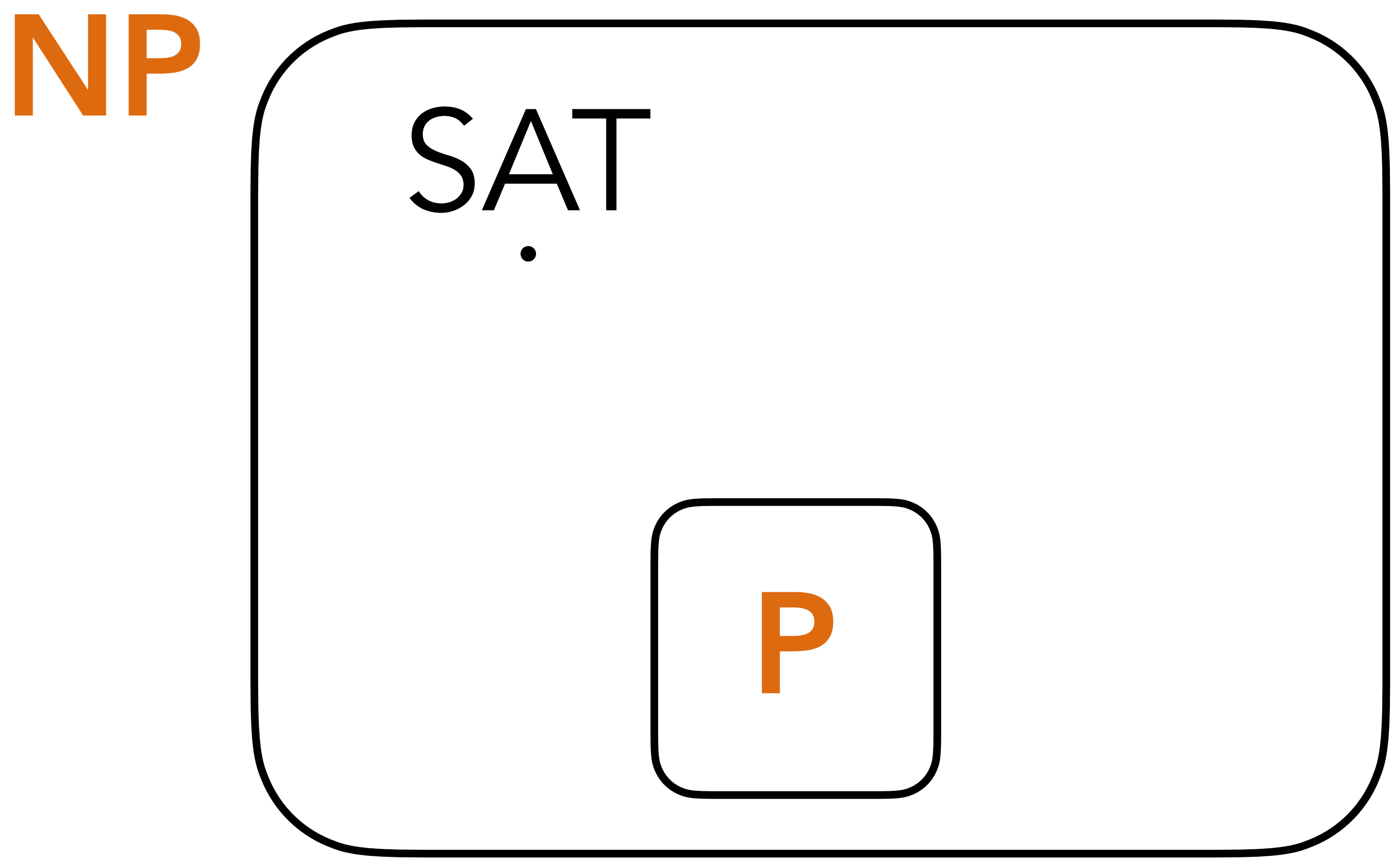
$\blacksquare$


Every L in NP
Cook-Levin Theorem
SAT
3SAT
3COL
SUBSET-SUM
CLIQUE
VERTEX-COVER
IND-SET
HAMILTONIAN-CYCLE
TSP

Cook-Levin Theorem

2 potentially surprising things about Cook-Levin

Theorem (Cook 1971, Levin 1973):
SAT is **NP**-complete.



$\le_m^P$ SAT

1. There exists an **NP**-complete language.
2. SAT is one of them.

TM-SAT is NP-hard

A TM V is **satisfiable** if $\exists u \in \Sigma^*$ such that $V(u)$ accepts.

Theorem: TM-SAT $= \{\langle V \rangle : V$ is a satisfiable TM$\}$ is **NP**-hard.

Want to show: for an arbitrary L in **NP**, $L \leq_m^P$ TM-SAT.

$$w \mapsto \langle V_w \rangle$$

$$w \in L \Leftrightarrow \langle V_w \rangle \in \text{TM-SAT}$$

(V is the verifier for L)

Definition: A language A is in **NP** if

- there is a polynomial-time TM V,
- a constant k,

such that:

$$x \in L \implies \exists u \text{ with } |u| \leq |x|^k \text{ s.t. } V(x, u) \text{ accepts},$$

$$x \notin L \implies \forall u, V(x, u) \text{ rejects}.$$

$x \in L \iff V(x, \cdot\,)$ is "**satisfiable**" (with a short string/proof)

$V_x(\,\cdot\,)$ is "**satisfiable**"

TM-SAT is NP-hard

A TM V is **satisfiable** if $\exists u \in \Sigma^*$ such that $V(u)$ accepts.

Theorem: TM-SAT $= \{\langle V \rangle : V$ is a satisfiable TM$\}$ is **NP**-hard.

Want to show: For an arbitrary L in **NP**, $L \leq_m^P$ TM-SAT.

$$w \quad \mapsto \quad \langle V_w \rangle$$

$$w \in L \quad \Leftrightarrow \quad V_w \text{ is satisfiable}$$

(V is the verifier for L)

Theorem: BOUNDED-TM-SAT is **NP**-complete.

SAT is NP-hard

Want to show: For an arbitrary L in **NP**, $L \leq_m^P$ SAT.

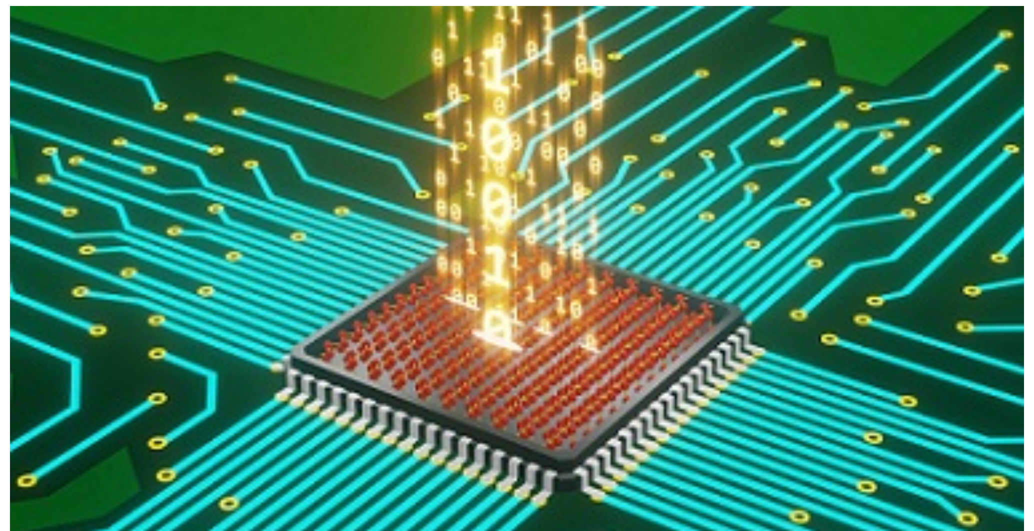
$$w \quad \mapsto \quad \langle \varphi_w \rangle$$

$$w \in L \quad \Leftrightarrow \quad \varphi_w \text{ is satisfiable}$$

We have: $w \in L \quad \Leftrightarrow \quad V_w$ is satisfiable

Main technical work: From V_w construct φ_w such that

$$V_w \text{ is satisfiable} \quad \Leftrightarrow \quad \varphi_w \text{ is satisfiable}$$



Is **NP**-completeness a death sentence?

Should we just give up?