15-251
Great Theoretical Ideas in Computer Science

Lecture 18:
NP and NP-completeness 2

March 23rd, 2017

Some important reminders from last time

Summary of last time

- How do you identify *intractable* problems?
  (problems not in **P**) e.g. *SAT, TSP, Subset-Sum*, …

- Poly-time reductions $A \leq_T^P B$ are useful to compare hardness of problems.

- Evidence for intractability of A:
  Show $L \leq_T^P A$, for **all** $L \in$ **C**, for a large class **C**.

- Definitions of **C**-hard, **C**-complete.

- What is a good choice for **C**,
  if we want to show, say, SAT is **C**-hard??

Summary of last time

- The complexity class **NP** (take **C** = **NP**)
- **NP**-hardness, **NP**-completeness
- Cook-Levin Theorem: SAT is **NP**-complete
- Many other languages are **NP**-complete.
- The **P** vs **NP** question

The complexity class **NP**

Informally:

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

$w \in A$ iff

there is a "*simple*" proof (solution) for this fact.

1. The length of the proof is polynomial in $|w|$.

2. The proof can be verified/checked in polynomial time.

The complexity class NP

Formally:

Definition:

A language A is in **NP** if

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

such that for all $x \in \Sigma^*$:

$$x \in A \iff \exists u \text{ with } |u| \leq |x|^k \text{ s.t. } V(x,u) = 1.$$

If $x \in A$, there is some poly-length proof that leads V to accept.

If $x \notin A$, every "proof" leads V to reject.

The Cook-Levin Theorem

Theorem (Cook 1971 - Levin 1973):

SAT is NP-complete.

It easier to show CIRCUIT-SAT is NP-complete.

So we will consider **Cook-Levin Theorem** to be:

"CIRCUIT-SAT is NP-complete."

Showing a language is **NP**-hard

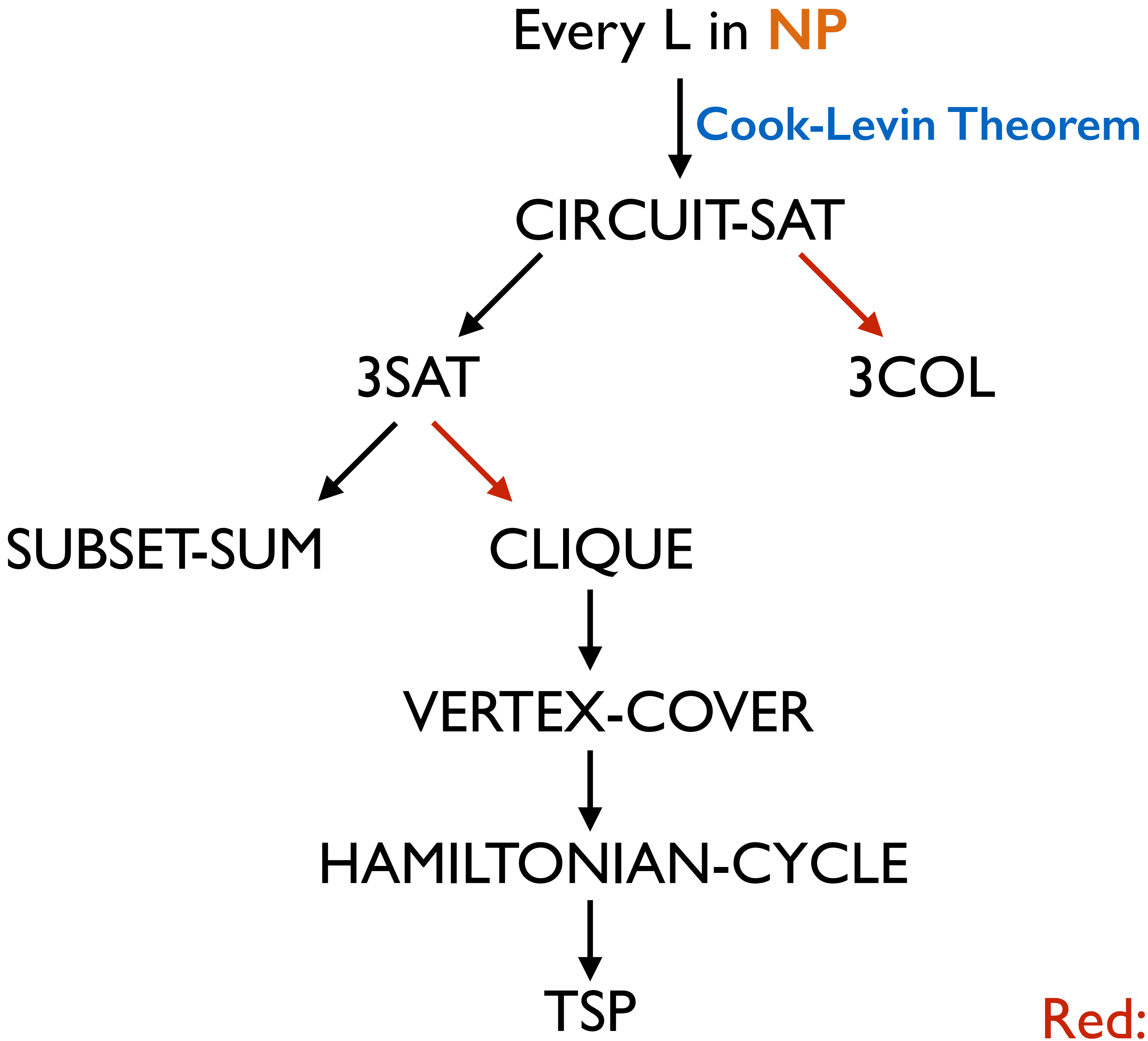
NP

$\leq_T^P$ CIRCUIT-SAT

To show L is **NP**-hard:

Pick your favorite **NP**-hard language K.

Show K $\leq_T^P$ L.



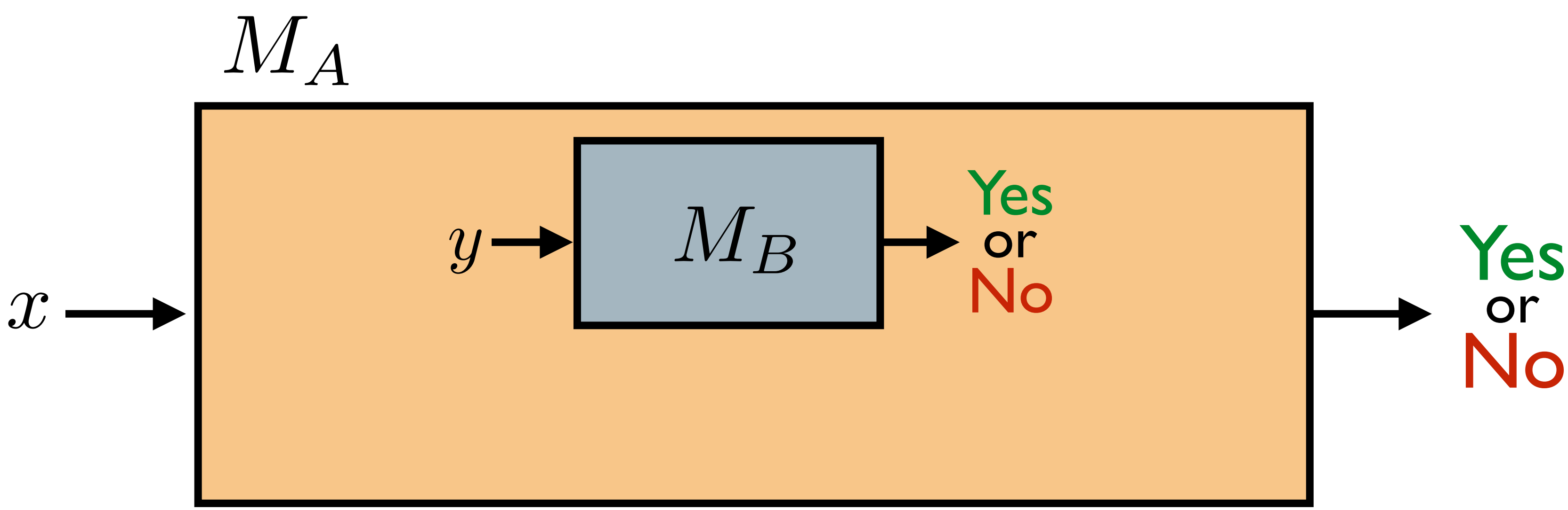
Red: will show

First:
An important note about reductions

Cook reduction

Cook reductions: poly-time Turing reductions

$$\boxed{\text{A} \leq_T^P \text{B}}$$



"You can solve A in poly-time using a blackbox that solves B."

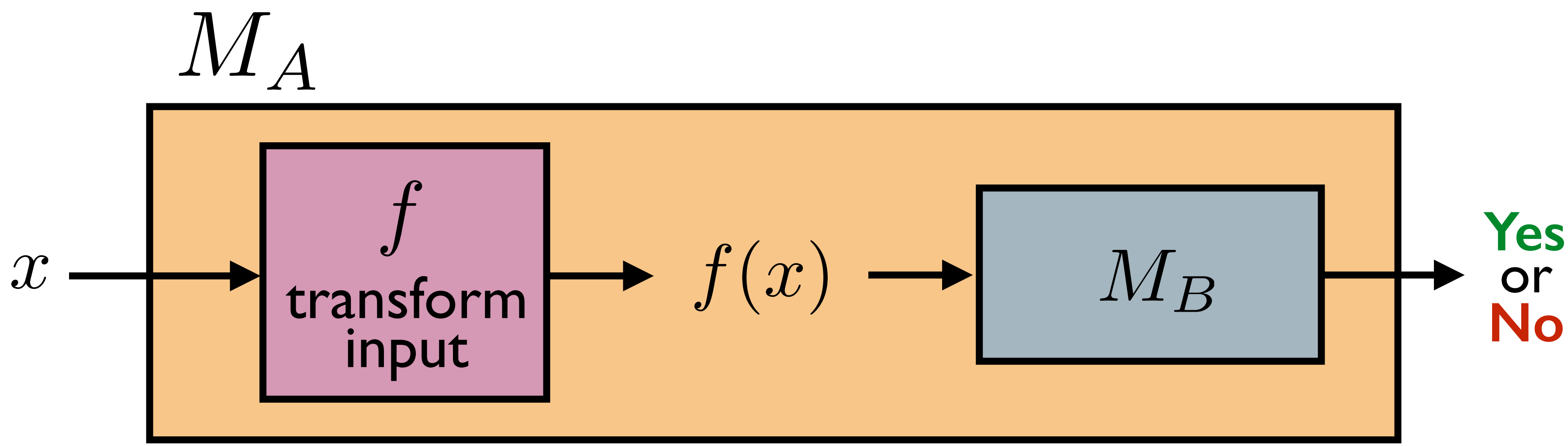
You can call the blackbox poly(|x|) times.

Karp reduction

NP-hardness is usually defined using Karp reductions.

Karp reduction (polynomial-time many-one reduction):

$$\boxed{\mathsf{A} \leq_m^P \mathsf{B}}$$



Make one call to M_B and directly use its answer as output.

<u>We must have:</u>

$$x \in \mathsf{A} \implies f(x) \in \mathsf{B}$$

$$x \notin \mathsf{A} \implies f(x) \notin \mathsf{B}$$

Karp reduction

Definition: Let A and B be two languages.

We say there is a polynomial-time many-one reduction (Karp reduction) from A to B if:

there is a polynomial-time computable function

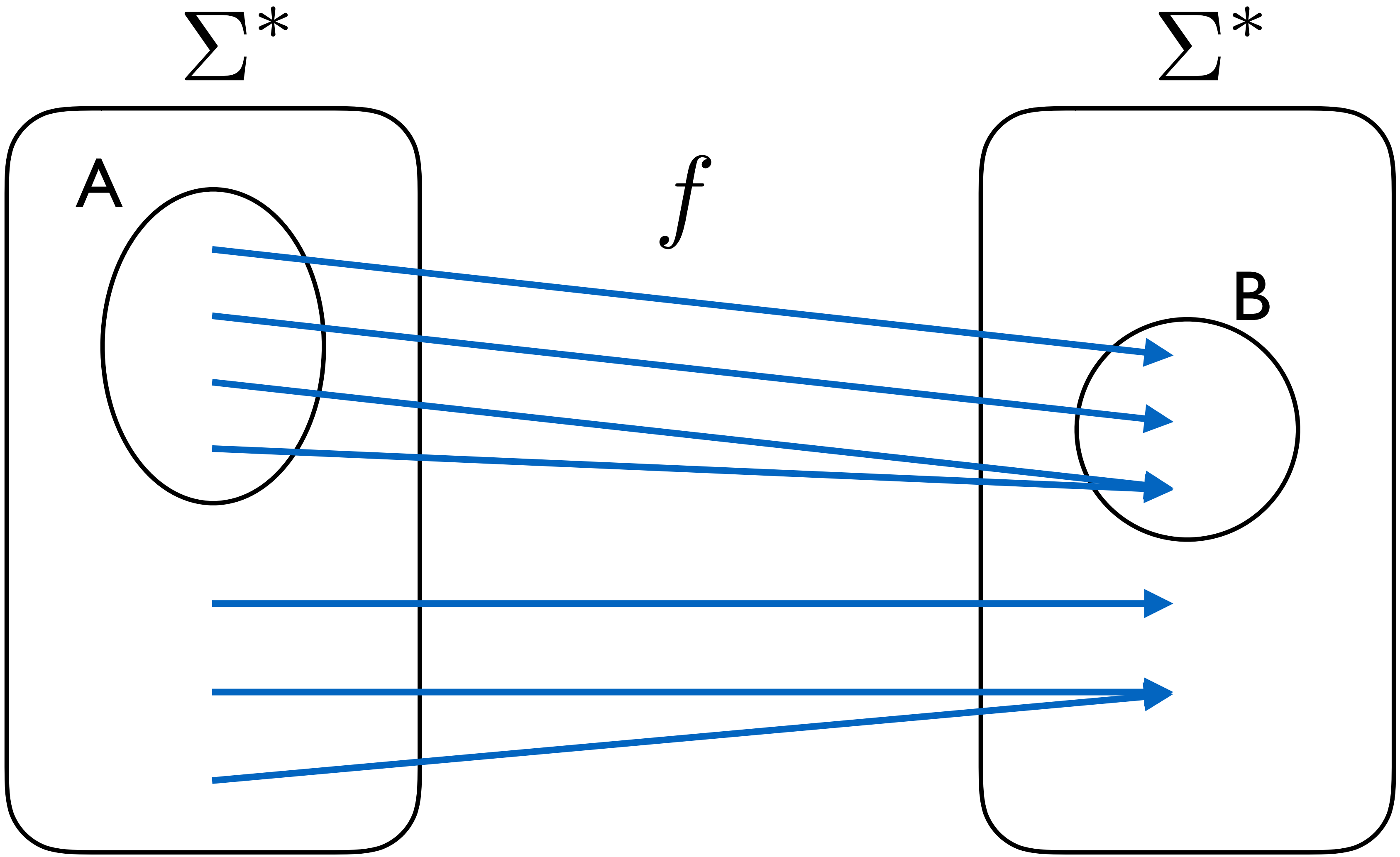
$$f : \Sigma^* \to \Sigma^*$$

such that:

$$x \in \text{A} \iff f(x) \in \text{B}.$$

Notation: $\text{A} \leq_m^P \text{B}$.

Karp reduction



A Karp reduction is a Cook reduction.

But not all Cook reductions are Karp reductions.

Karp Reduction: Example

CLIQUE

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

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

INDEPENDENT-SET (IS)

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

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

Fact: CLIQUE $\leq_m^P$ IS.

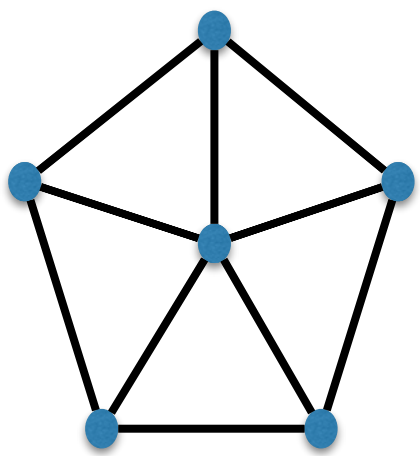
Karp Reduction: Example

Want:

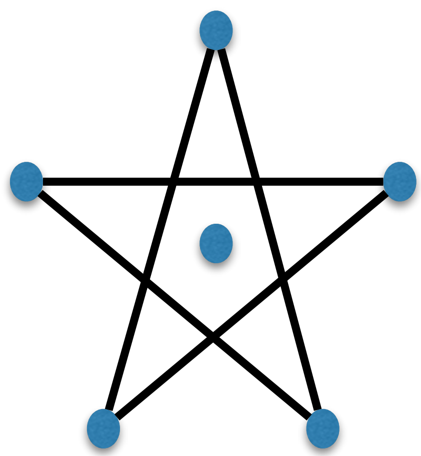
$$\langle G, k \rangle \mapsto \langle G', k' \rangle$$

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

Proof:

We need to:

1. Define a map $f : \Sigma^* \to \Sigma^*$.
2. Show $w \in$ CLIQUE $\implies f(w) \in$ IS
3. Show $w \notin$ CLIQUE $\implies f(w) \notin$ IS
   (often easier to argue the contrapositive)
4. Argue f is computable in polynomial time.

Karp Reduction: Example

Proof (continued):

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

def $f(w)$:

- If *w* is not an encoding $\langle G, k \rangle$ of a graph *G* and int *k*, map it to ϵ.
- Otherwise *w* = $\langle G = (V, E), k \rangle$.
- Let $E^* = \{\{u, v\} : \{u, v\} \notin E\}$
- Return $\langle G^* = (V, E^*), k \rangle$.

$$\text{not valid encoding} \mapsto \epsilon$$

$$\langle G, k \rangle \mapsto \langle G^*, k \rangle$$

Karp Reduction: Example

Proof (continued):

2. Show $w \in$ CLIQUE $\implies f(w) \in$ IS

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

In the complement graph *G**, S is an IS of size *k*.

So $f(w) = \langle G^*, k \rangle \in$ IS

Karp Reduction: Example

Proof (continued):

3. Show $w \notin$ CLIQUE $\implies f(w) \notin$ IS

(Show the contrapositive.)

If $f(w) \in$ IS, then $f(w)$ = $\langle G^* = (V, E^*), k \rangle$ and G^* has an IS $S \subseteq V$ of size *k*.

$w = \langle G, k \rangle$

In the complement of G^*, which is *G*, S is a clique of size *k*.

So $w = \langle G, k \rangle \in$ CLIQUE

Karp Reduction: Example

Proof (continued):

4. Argue f is computable in polynomial time.

- checking if the input is a valid encoding can be done in polynomial time.
  (for any reasonable encoding scheme)

- creating E^*, and therefore G^*, can be done in polynomial time.

$\square$

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

(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.

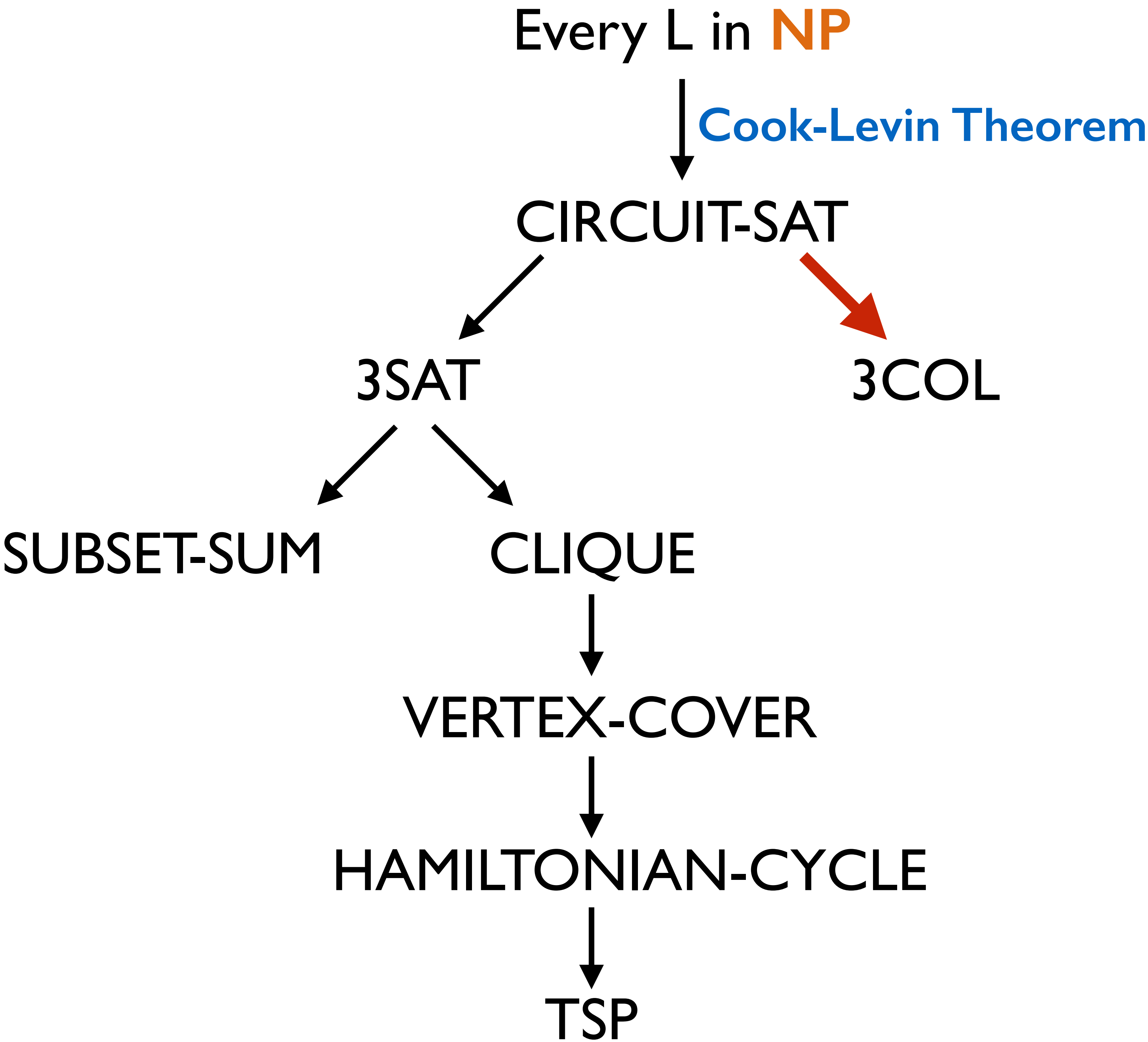
Poll 1

Which of the following are true?

- if $A \leq_m^P B$ and $B \leq_m^P C$, then $A \leq_m^P C$.

- $A \leq_m^P B$ if and only if $B \leq_m^P A$.

- if $A \leq_m^P B$ and $B \in$ **NP**, then $A \in$ **NP**.


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

3COL is NP-complete

3COL is **NP**-complete: High level steps

3COL is in **NP** (exercise).

We know CIRCUIT-SAT is **NP**-hard.
So it suffices to show CIRCUIT-SAT $\leq_m^P$ 3COL.

<u>We need to:</u>

1. Define a map $f : \Sigma^* \to \Sigma^*$.
2. Show $w \in$ CIRCUIT-SAT $\implies f(w) \in$ 3COL
3. Show $w \notin$ CIRCUIT-SAT $\implies f(w) \notin$ 3COL
4. Argue f is computable in polynomial time.

CIRCUIT-SAT $\leq$ 3COL: The construction

1. Define a map $f : \Sigma^* \rightarrow \Sigma^*$.

If x is not $\langle C \rangle$ for a circuit C, map it to ϵ.

So assume x is a valid encoding of a circuit.

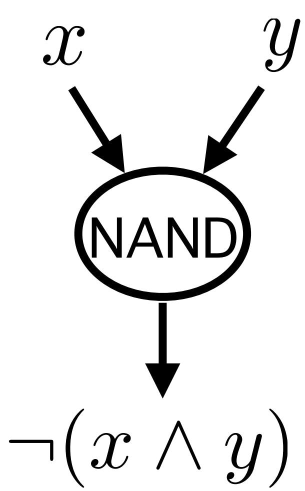
Circuit with AND, OR, NOT gates

Circuit with only NAND gates

(in addition to input gates and constant gates)

CIRCUIT-SAT $\leq$ 3COL: The main gadget

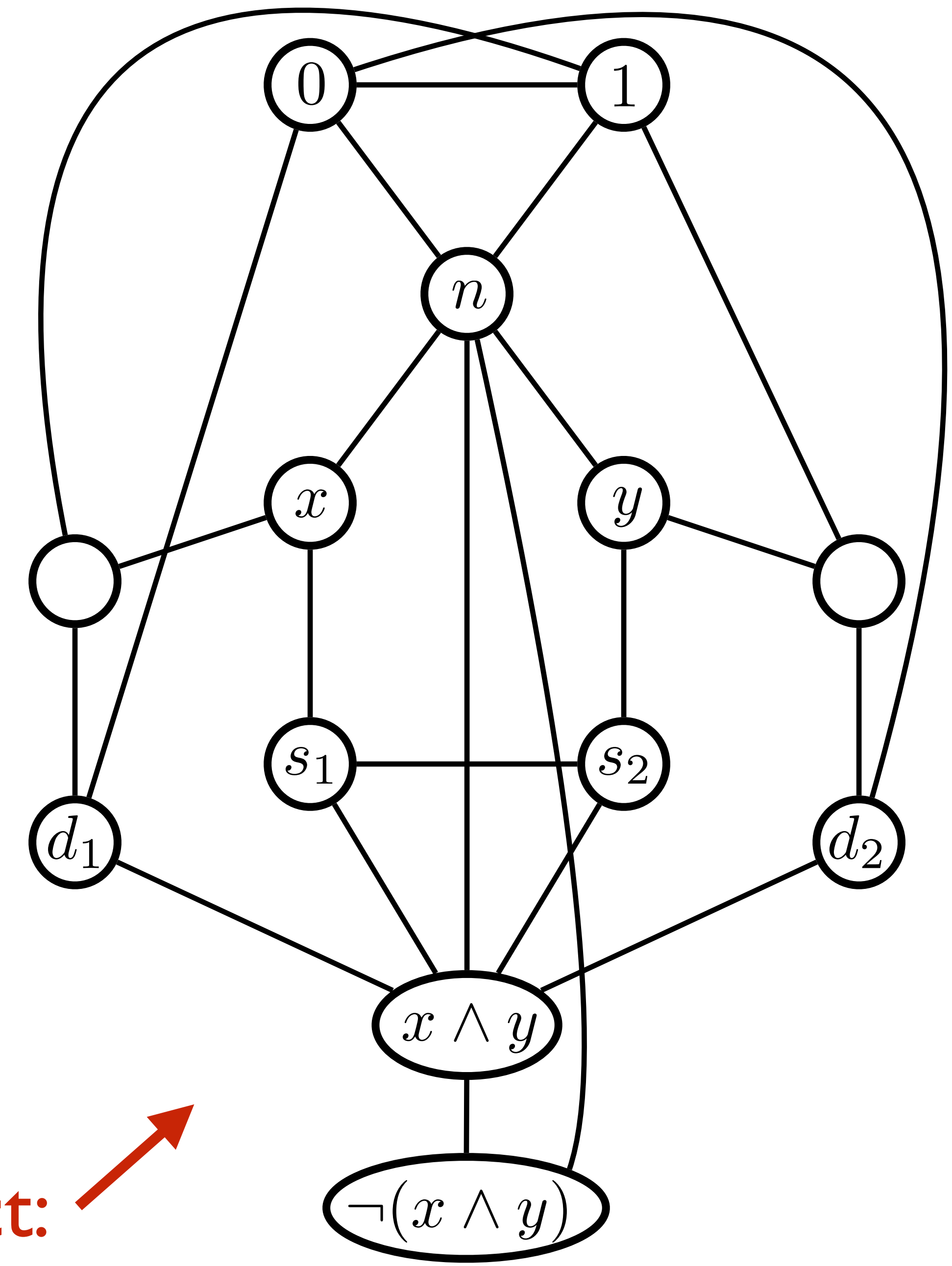
Consider a NAND gate.



x and y represent some other gates.

$\neg(x \wedge y)$ becomes the input of another gate.

For each NAND gate, construct:



CIRCUIT-SAT ≤ 3COL: The main gadget

Claim:

A valid coloring of this "gadget" mimics the behaviour of the NAND gate.

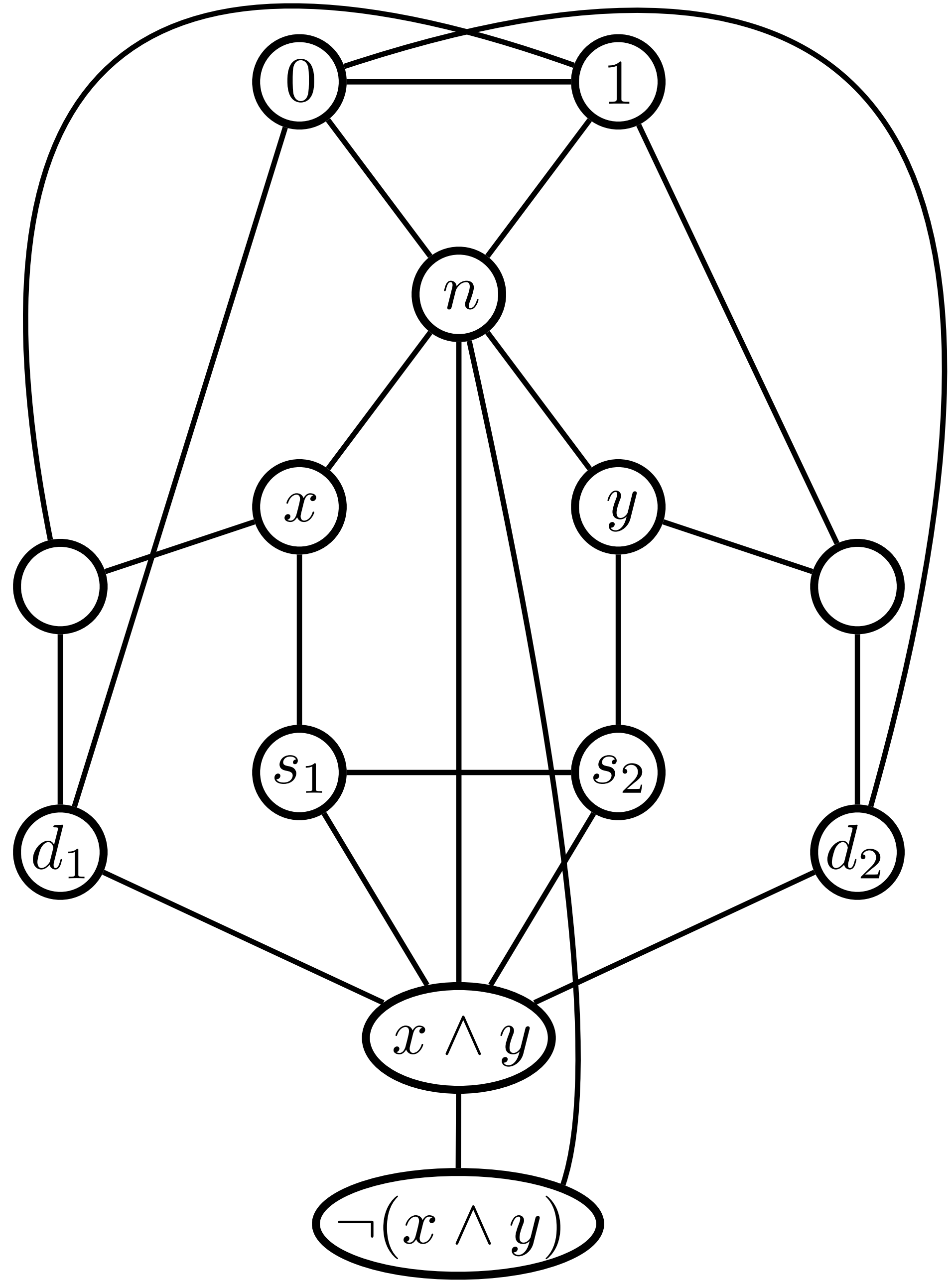
Colors = {0, 1, n}

WLOG:

vertex **0** gets color 0

vertex **1** gets color 1

vertex **n** gets color n



CIRCUIT-SAT $\leq$ 3COL: The main gadget

A couple of observations:

<u>Observation1:</u>

vertices x, y

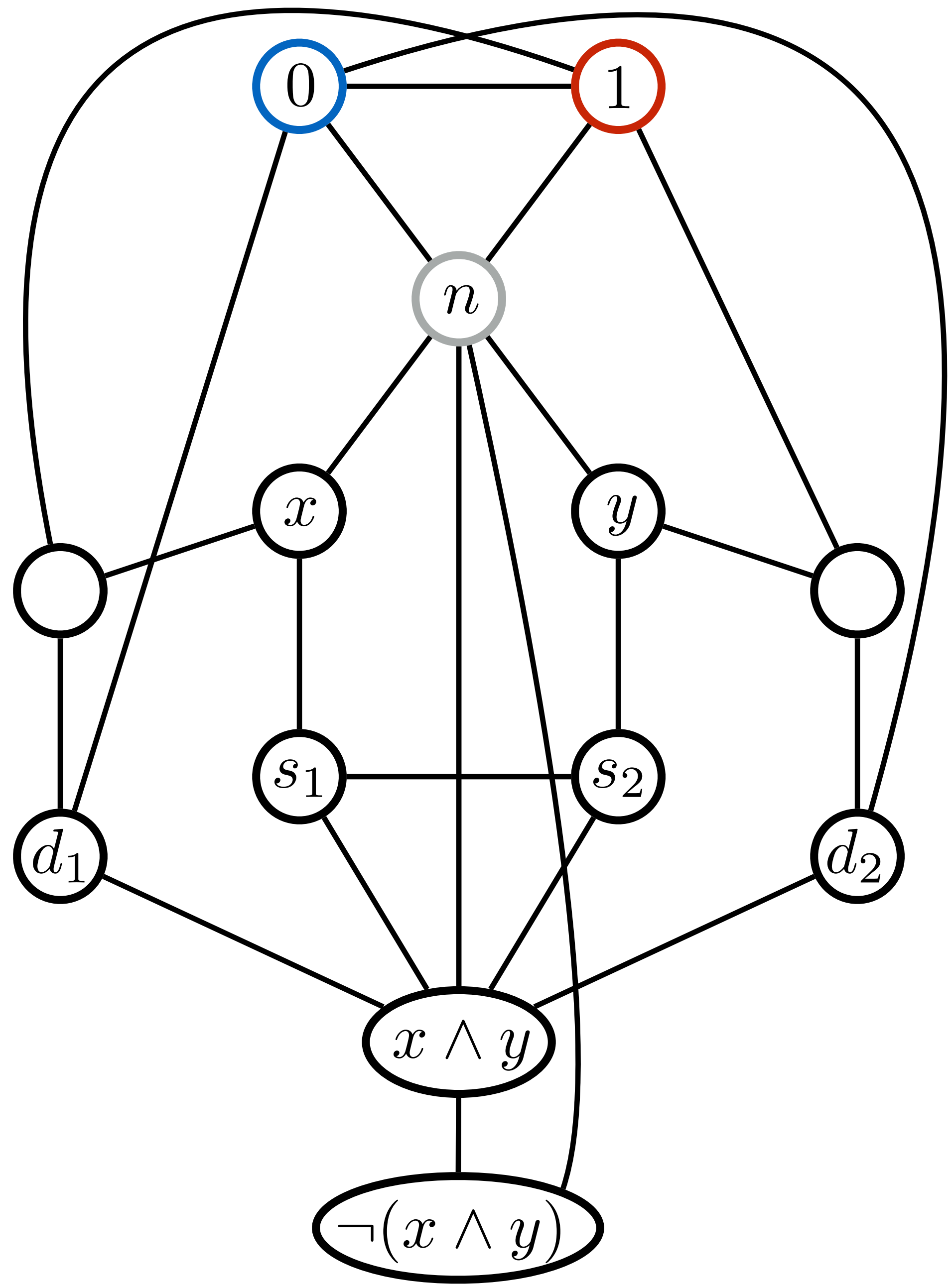
$x \wedge y$ and $\neg(x \wedge y)$

will not be assigned the color n.

<u>Observation2:</u>

$x \wedge y$ and $\neg(x \wedge y)$

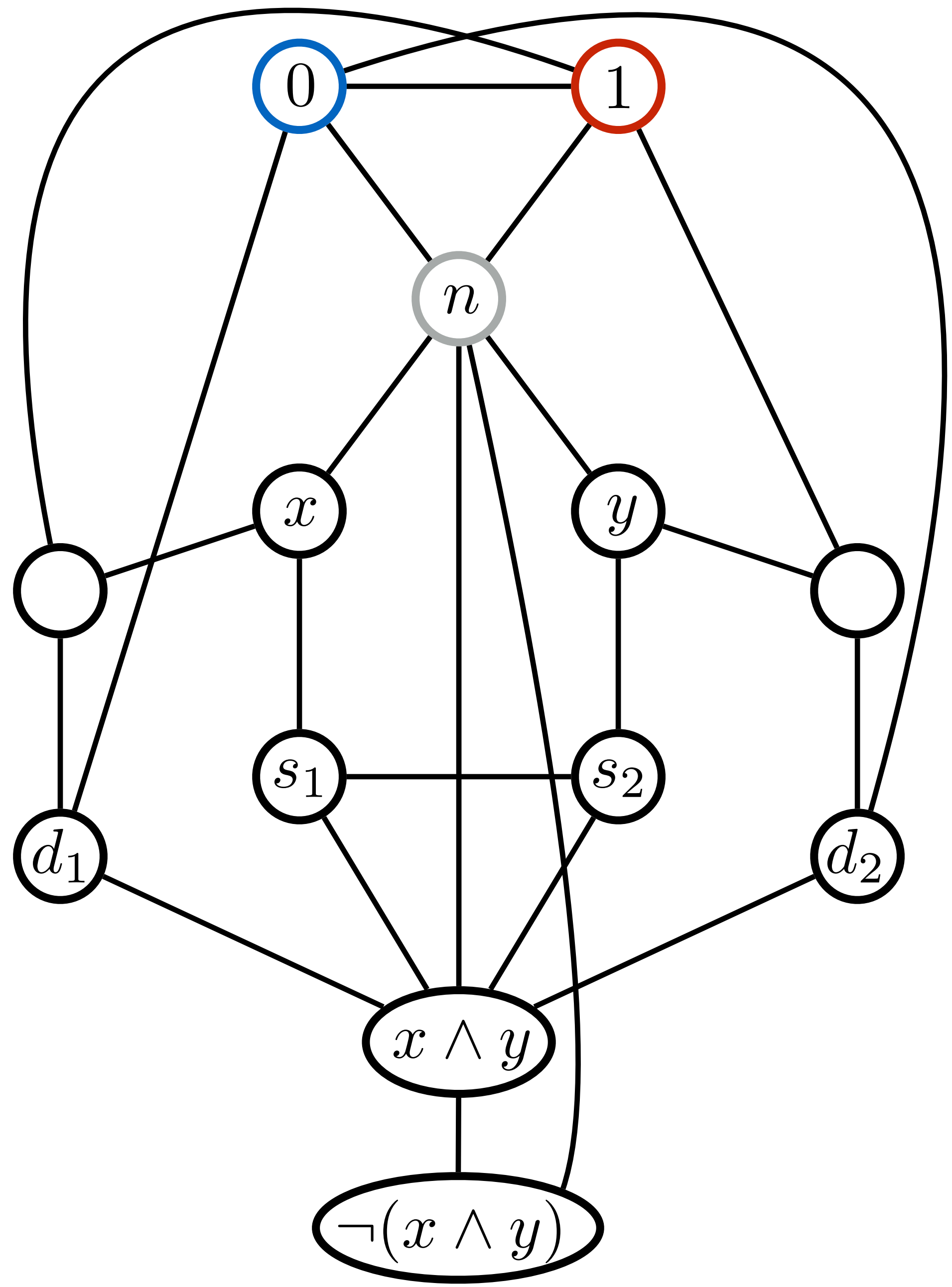
will be assigned different colors.



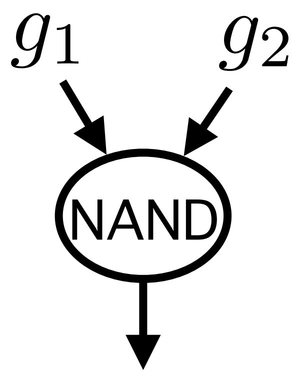
CIRCUIT-SAT $\leq$ 3COL: The main gadget

Possible colorings of the vertices x, y and $\neg(x \wedge y)$:

x	y	$\neg(x \wedge y)$
0	0	1
1	1	0
0	1	1
1	0	1

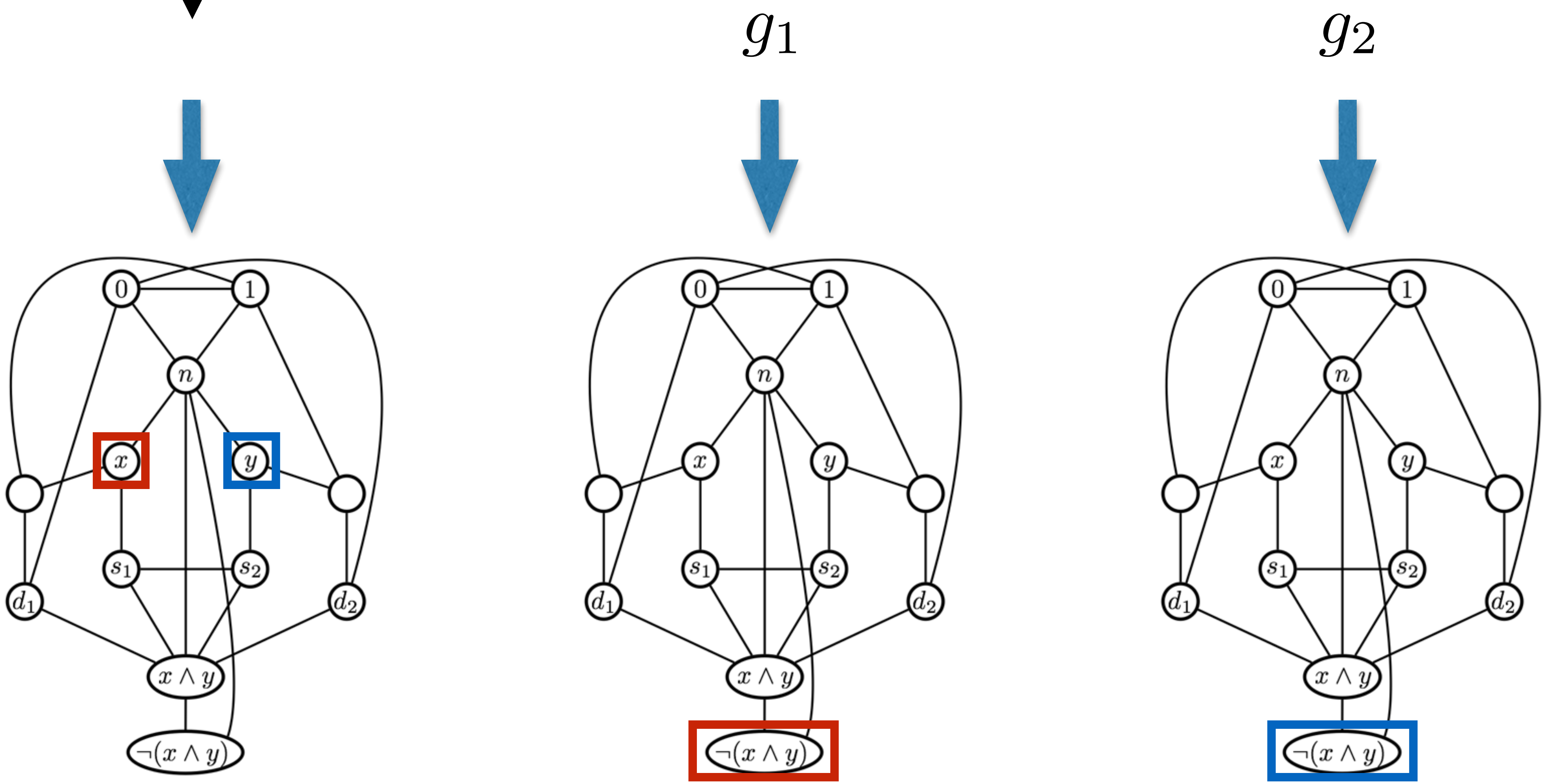


CIRCUIT-SAT ≤ 3COL: Rest of construction

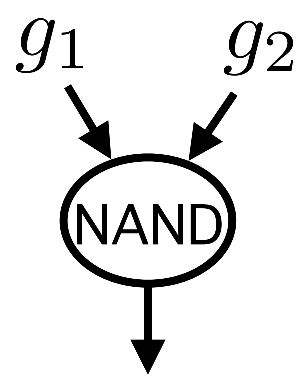


blue vertices are the same vertex.

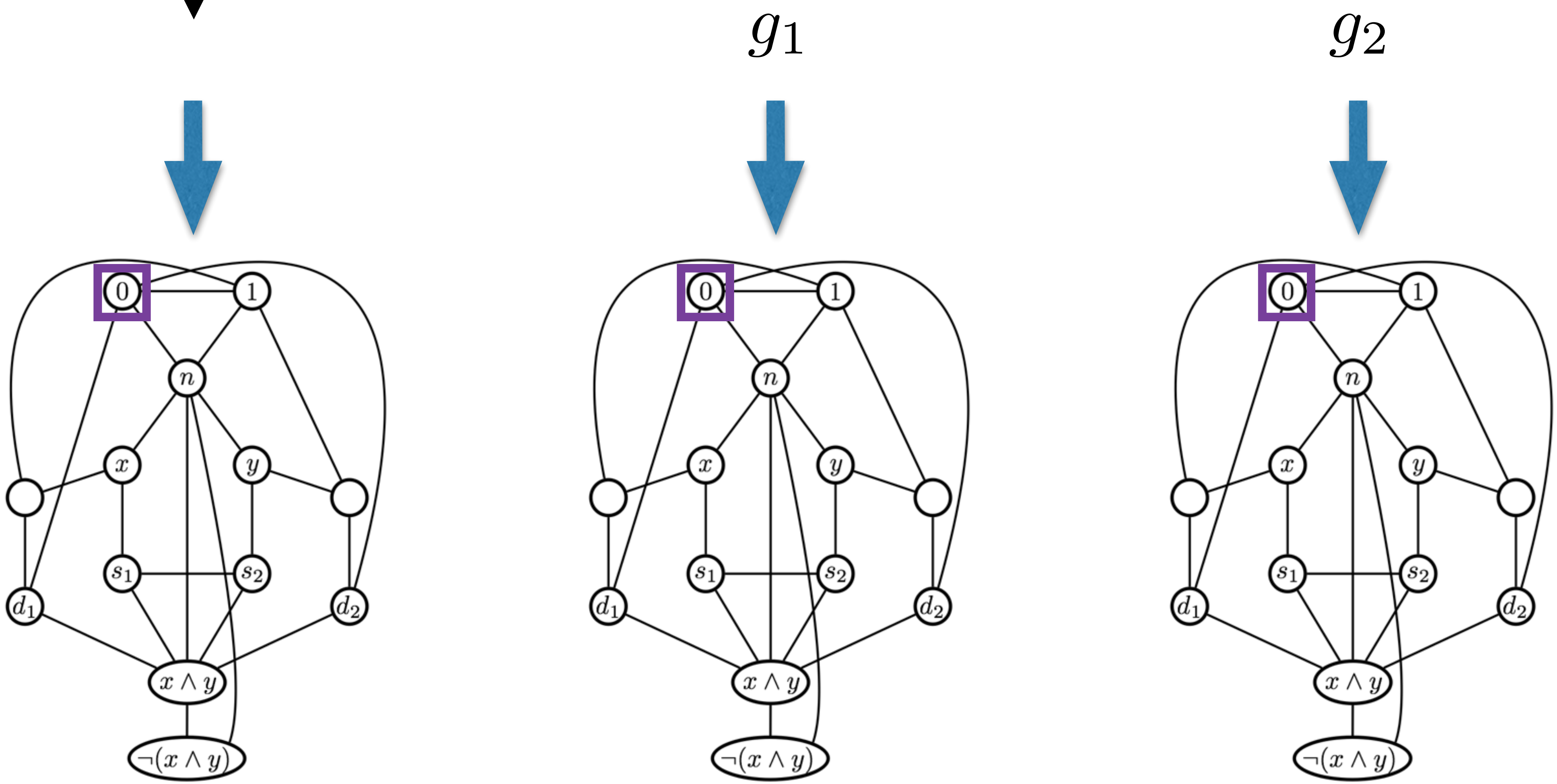
red vertices are the same vertex.



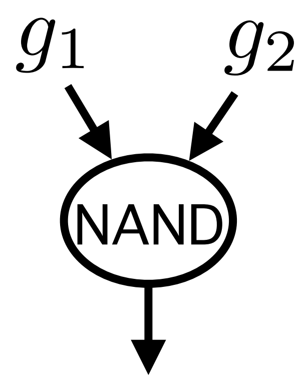
CIRCUIT-SAT ≤ 3COL: Rest of construction



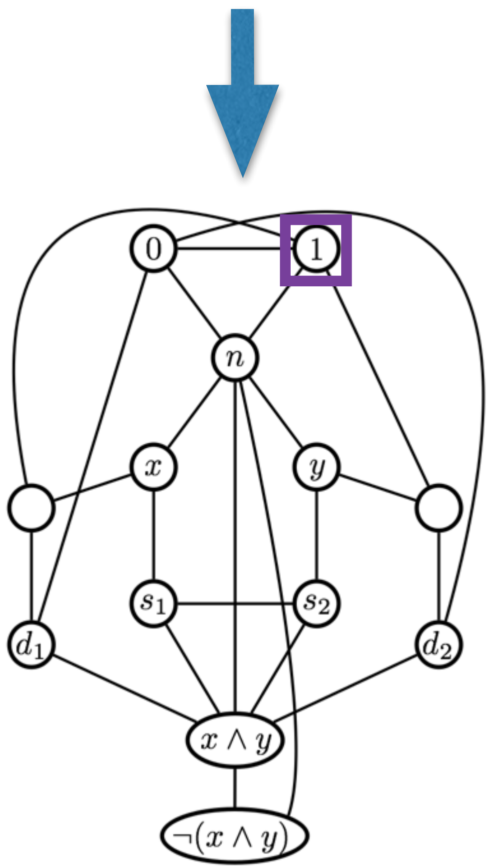
vertices labeled **0** are all the same.

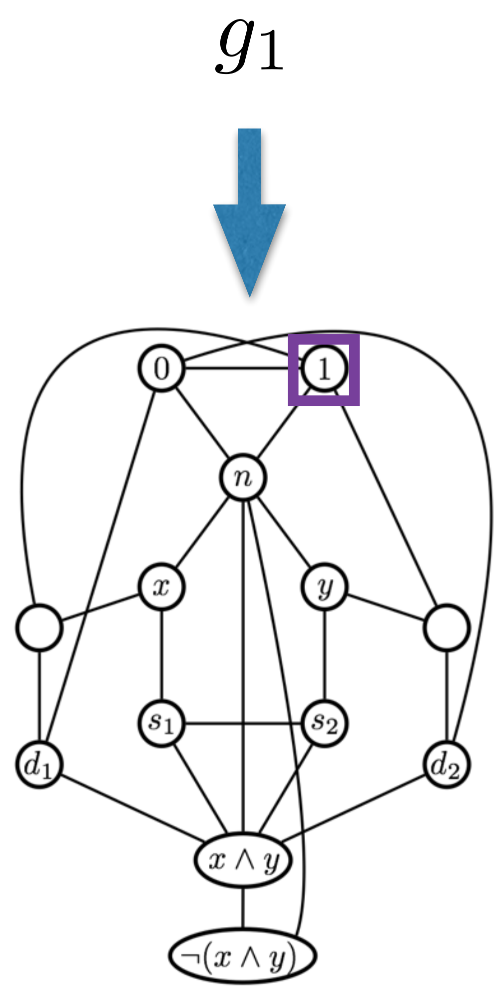


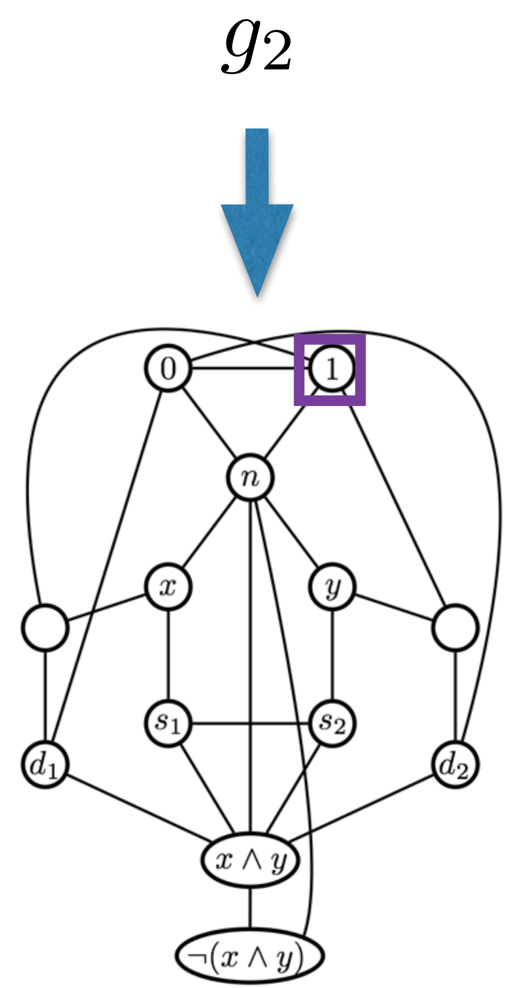
CIRCUIT-SAT ≤ 3COL: Rest of construction



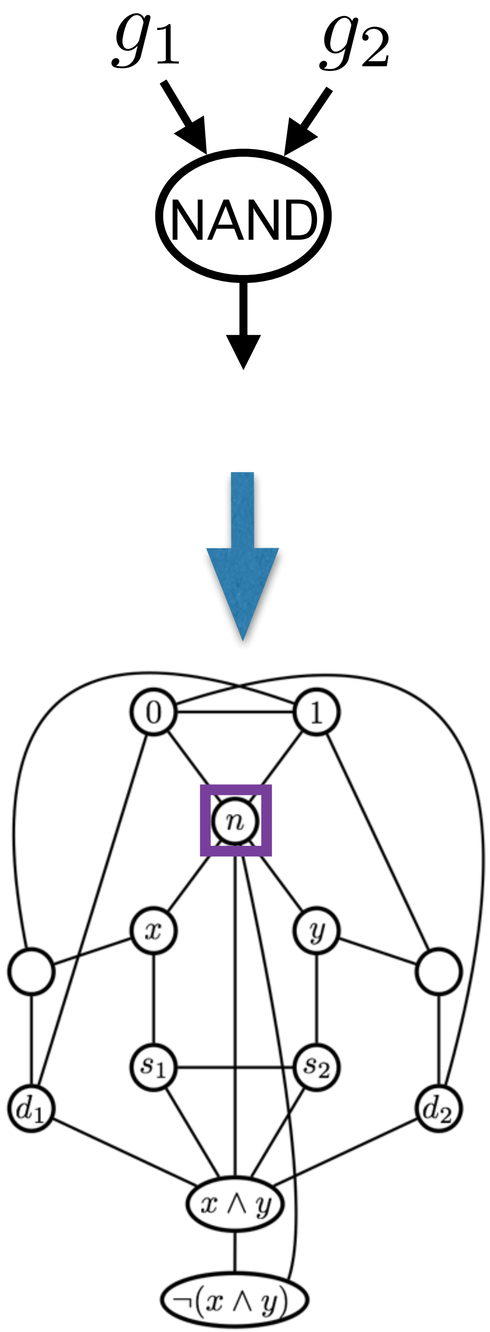
vertices labeled 1 are all the same.



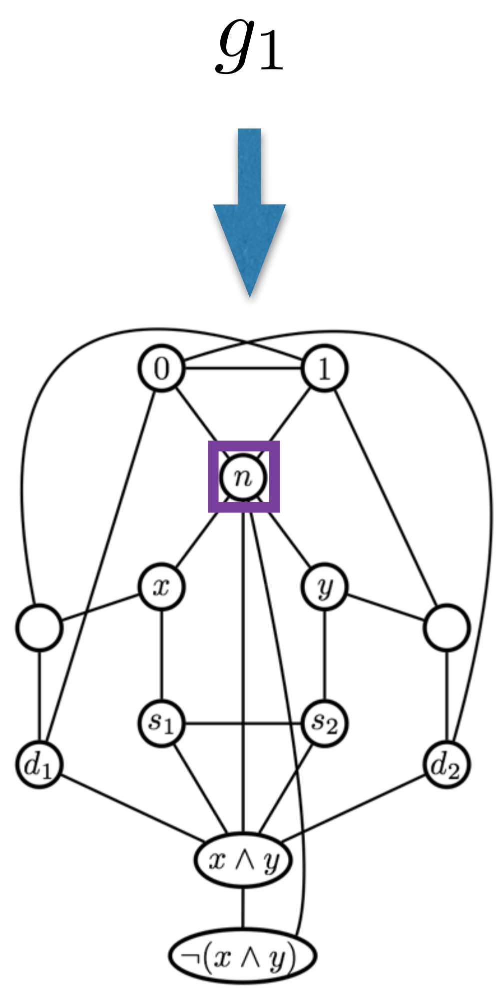


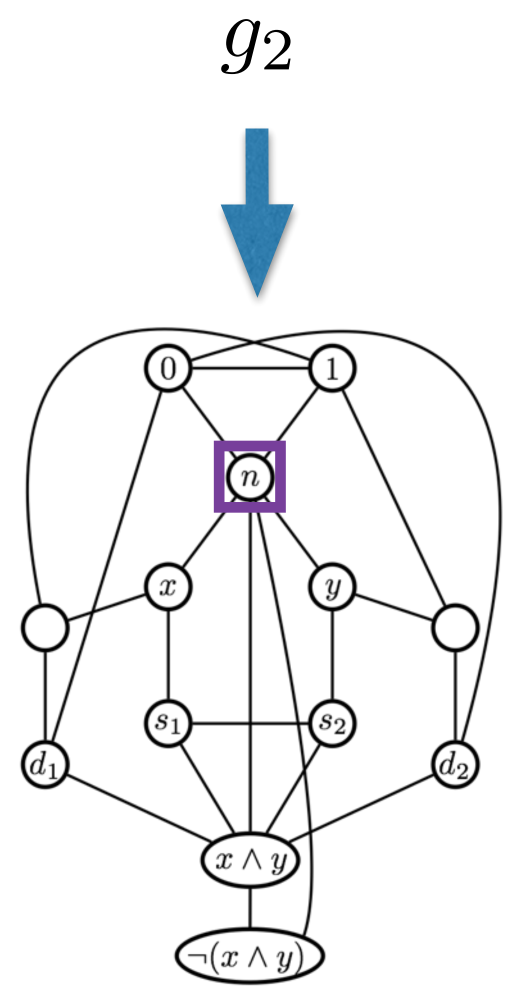


CIRCUIT-SAT ≤ 3COL: Rest of construction



vertices labeled **n** are all the same.

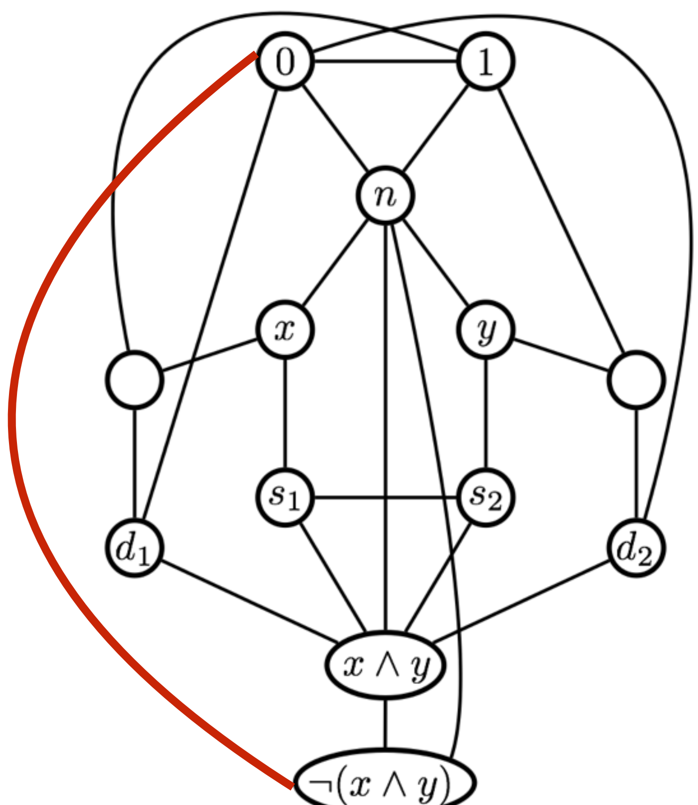




CIRCUIT-SAT ≤ 3COL: Rest of construction

Input gates just map to a single vertex.

Gadget for the **output gate** has one extra edge:



CIRCUIT-SAT $\leq$ 3COL: Why does it work?

Convince yourself that:

$$w \in \text{CIRCUIT-SAT} \implies f(w) \in \text{3COL}$$

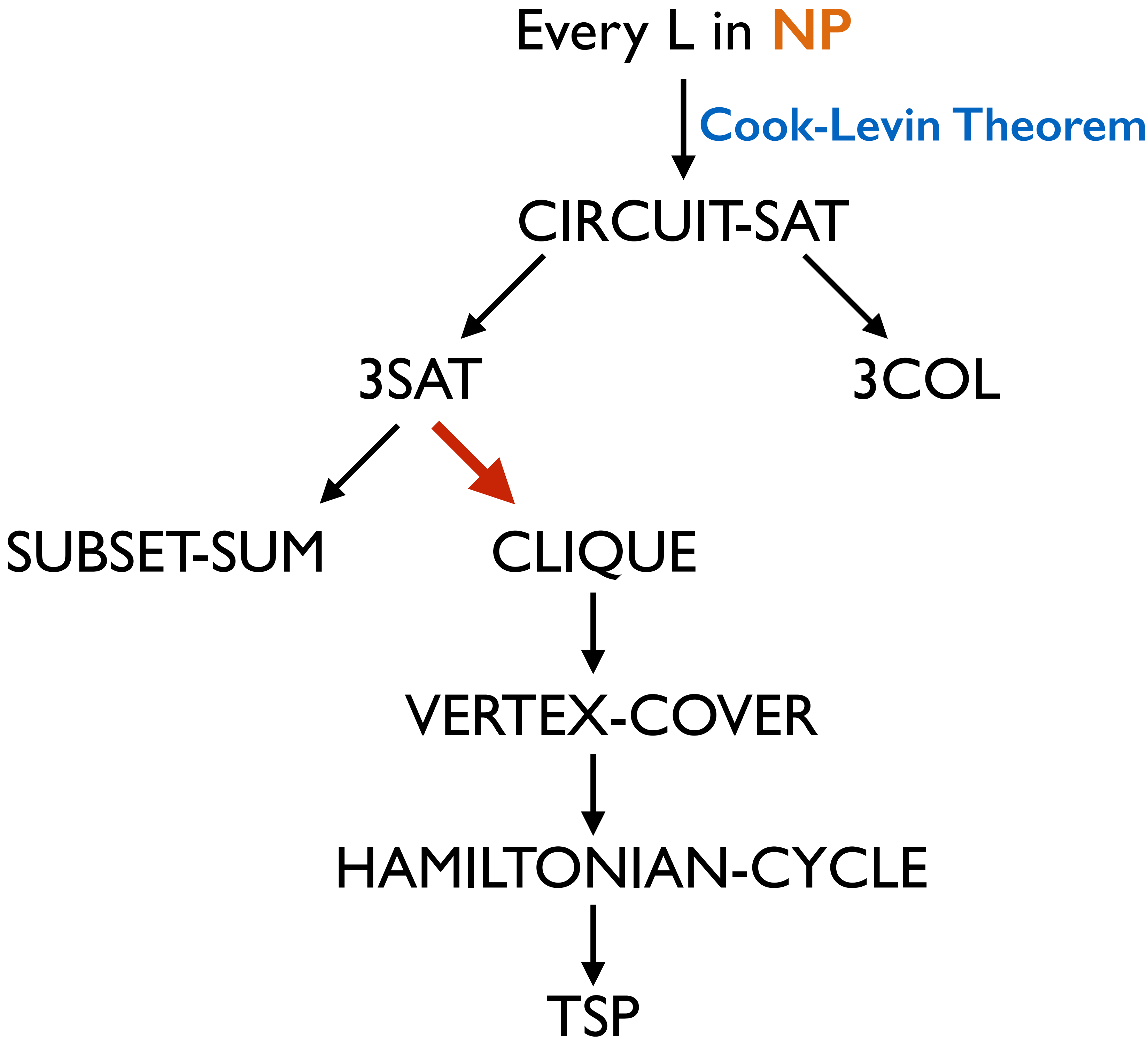
$$w \notin \text{CIRCUIT-SAT} \implies f(w) \notin \text{3COL}$$

f is computable in polynomial time.

Poll 2

Which of the following are true?

- 3COL $\leq_m^P$ 2COL is known to be true.
- 3COL $\leq_m^P$ 2COL is known to be false.
- 3COL $\leq_m^P$ 2COL is open.
- 2COL $\leq_m^P$ 3COL is known to be true.
- 2COL $\leq_m^P$ 3COL is known to be false.
- 2COL $\leq_m^P$ 3COL is open.


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

CLIQUE is NP-complete

Definition of 3SAT Problem

3SAT

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

e.g.:

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

(an OR of literals)

literal: a variable or its negation

conjunctive normal form: AND of clauses.

(Note: To satisfy the formula, you need to satisfy each clause.)

Output: Yes iff the formula is satisfiable.

Aside: 3SAT is in NP

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

φ satisfiable

$$\Longleftrightarrow$$

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

$$\Longleftrightarrow$$

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

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

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

CLIQUE is **NP**-complete: High level steps

CLIQUE is in **NP**. ✓

We know 3SAT is **NP**-hard.
So suffices to show 3SAT $\leq_m^P$ CLIQUE.

We need to:

1. Define a map $f : \Sigma^* \to \Sigma^*$.
2. Show $w \in$ 3SAT $\implies f(w) \in$ CLIQUE
3. Show $w \notin$ 3SAT $\implies f(w) \notin$ CLIQUE
4. Argue f is computable in polynomial time.

3SAT $\leq$ CLIQUE: Defining the map

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

not valid encoding of a 3SAT formula $\quad \mapsto \quad \epsilon$

otherwise we have valid 3SAT formula φ
(with *m* clauses).

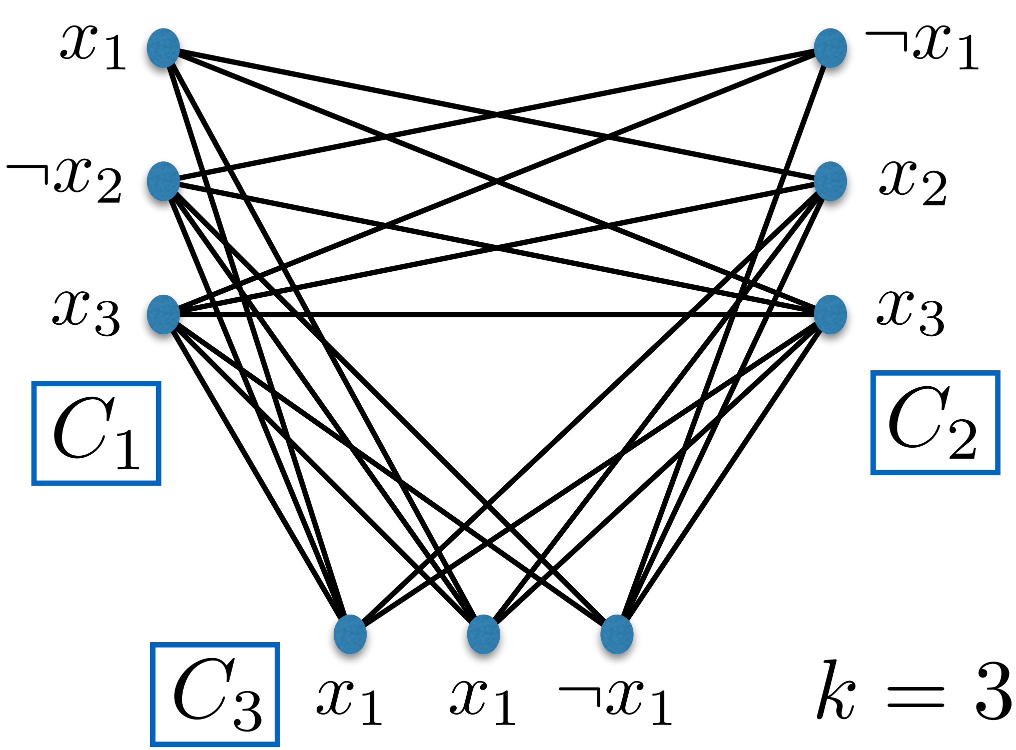
$\varphi \mapsto \langle G, k \rangle$ (we set $k = m$)

Construction demonstrated with an example.

3SAT $\leq$ CLIQUE: Defining the map

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

G_φ



$k = 3$

The construction:

- A vertex for each literal in each clause.
- No edges between two literals in the 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 $\leq$ CLIQUE: Why it works

If φ is satisfiable, then G_φ contains an m-clique:

φ is satisfiable

$\Longrightarrow$

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

$\Longrightarrow$

by construction of G_φ, vertices corresponding to
those literals are all connected (by an edge).

$\Longrightarrow$

G_φ contains an m-clique.

3SAT $\leq$ CLIQUE: Why it works

If G_φ contains an m-clique, then φ is satisfiable:

G_φ has a clique K of size m

$\Longrightarrow$

by construction of G_φ:

- K must contain exactly one literal from each clause.
- K cannot contain a variable and its negation.

$\Longrightarrow$

φ is satisfiable.

3SAT $\leq$ CLIQUE: Poly-time reduction?

Creation of G_φ is 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.

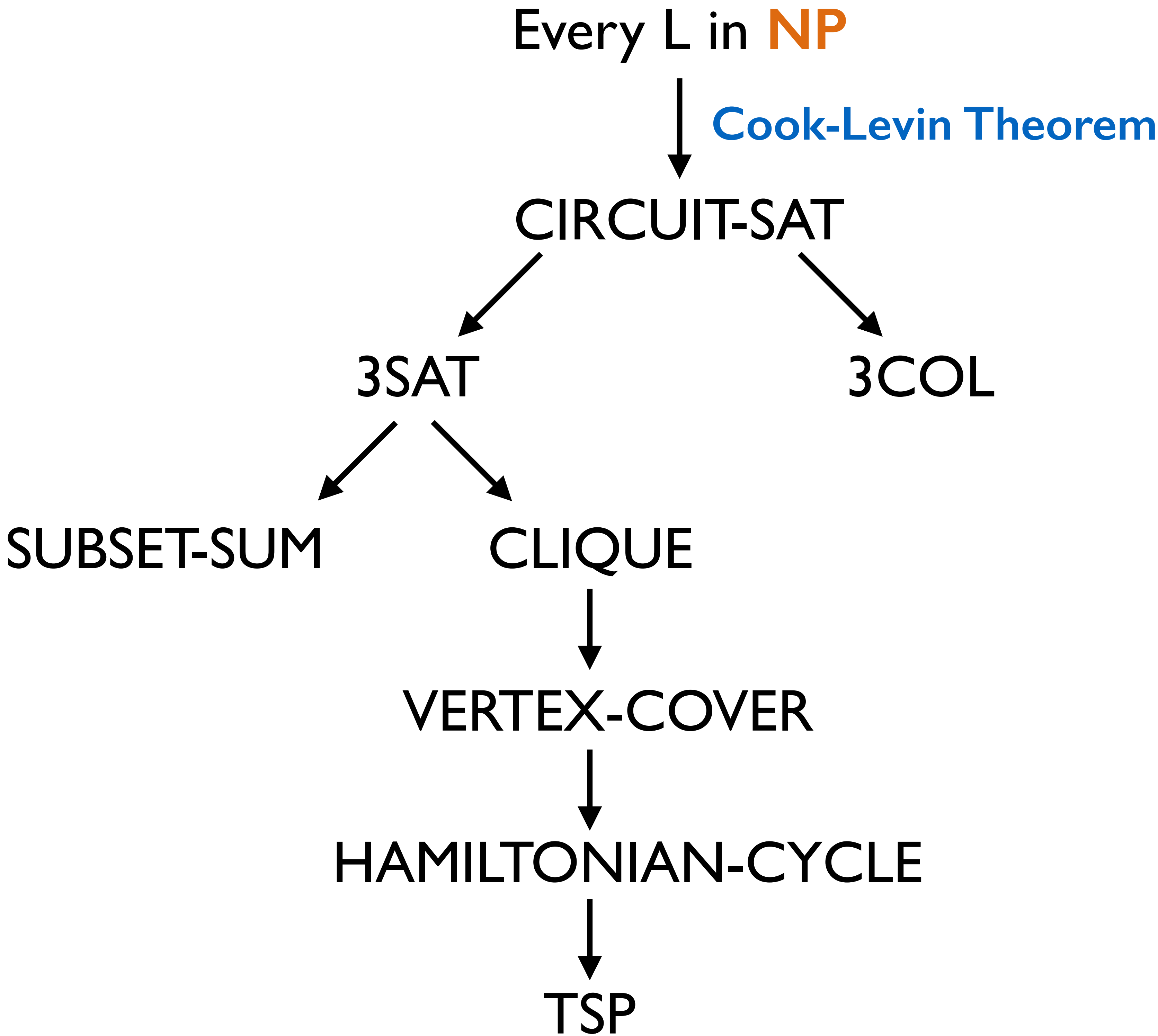
Creating the edge set:

- there are at most $O(m^2)$ possible edges.
- scan input formula to determine if an edge should be present.

$\square$

Independent Set is NP-complete

Corollary: IS is **NP**-hard.



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

NEXT TIME:

Cook-Levin Theorem: CIRCUIT-SAT is **NP**-complete