15-251
Great Theoretical Ideas in Computer Science

Lecture 14:
NP and NP-completeness 2

October 13th, 2016

Some important reminders from last time

The complexity class **NP**

What is common about
TSP, Subset-Sum, Theorem Proving Problem,
SAT, CIRCUIT-SAT, Sudoku,
and almost every other interesting problem you can think of?

Seems hard to find a correct **solution**
(solution space is too big!)

BUT, easy to verify a given **solution**.

They are all problems we can solve with Brute-Force Search.

The complexity class **NP**

<u>Informally:</u>

A language is in **NP** if:

whenever we have a Yes instance,
there is a "*<u>simple</u>*" proof (solution) for this fact.

↓

1. The length of the proof is polynomial in the input size.
2. The proof can be verified/checked in polynomial time.

Recall the definition of **NP**

Definition:

A language A is in **NP** if

- there is a polynomial-time TM V
- a polynomial p

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

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

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

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

Examples of languages in **NP**

CIRCUIT-SAT

Input: $\langle C \rangle$ where C is a Boolean circuit.

Output: Yes iff C is satisfiable.

Fact: CIRCUIT-SAT is in **NP**.

Examples of languages in **NP**

The way you need to write the proof:

We need to show a poly-time verifier TM V exists as specified in the definition of **NP**.

def $V(x, u)$:

- if x is not an encoding $\langle C \rangle$ of a valid circuit C, REJECT.
- if u is not an encoding of a valid 0/1 assignment to the input gates of C, REJECT.
- evaluate the output of the circuit with the given u .
- if it evaluates to 0, REJECT.
- else, ACCEPT.

Examples of languages in NP

The way you need to write the proof:

Need to show:

1. if $x \in$ CIRCUIT-SAT, there is some proof u of poly-length that makes V ACCEPT.

2. if $x \notin$ CIRCUIT-SAT, no matter what u is, V REJECTS.

3. V is polynomial-time.

Argue these, point by point.

Poll

Which of the following decision problems are in **NP**?

1. Given numbers $a_1, \ldots, a_n$ and k in $\mathbb{N}$, is there a set $S \subseteq \{1, \ldots, n\}$ s.t. $\sum_{i \in S} a_i = k$?

2. Given a graph G and k in $\mathbb{N}$, is the largest clique in G of size at most k?

3. Both

4. Neither

NP-hard and **NP**-complete

A language L is **NP**-hard if

NP $\leq_T^P$ L

If L is in **P**, then everything in **NP** is in **P**, i.e. **P** = **NP**.

If L is **NP**-hard and in **NP**, then it is **NP**-complete.

Extremely strong property.
How can any language be **NP**-complete?

The Cook-Levin Theorem

Theorem (Cook 1971 - Levin 1973):

SAT is **NP**-complete.

It turns it easier to show CIRCUIT-SAT is **NP**-complete.

So we will consider Cook-Levin Theorem to be:

CIRCUIT-SAT is **NP**-complete.

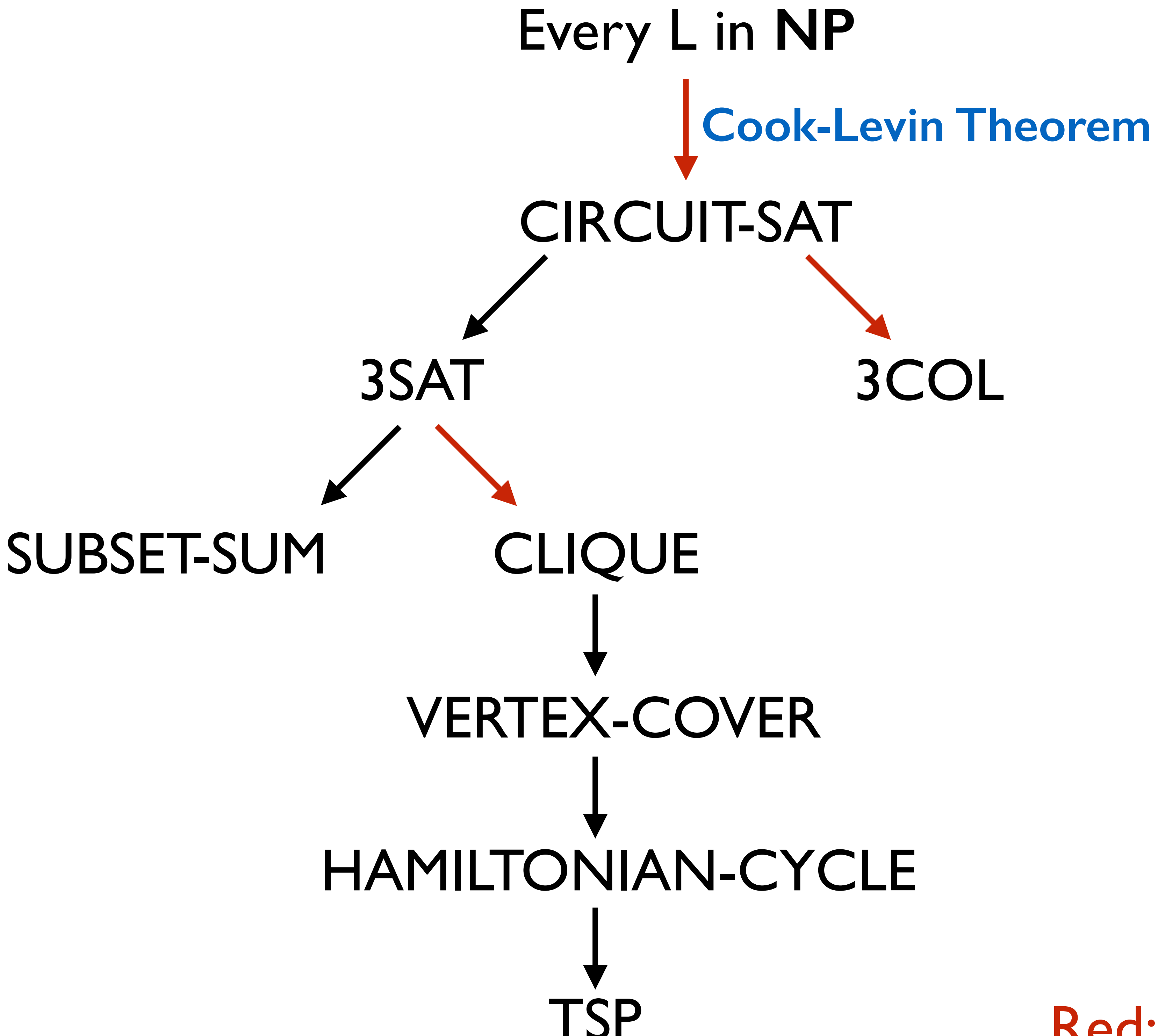
NP-hard and **NP**-complete

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

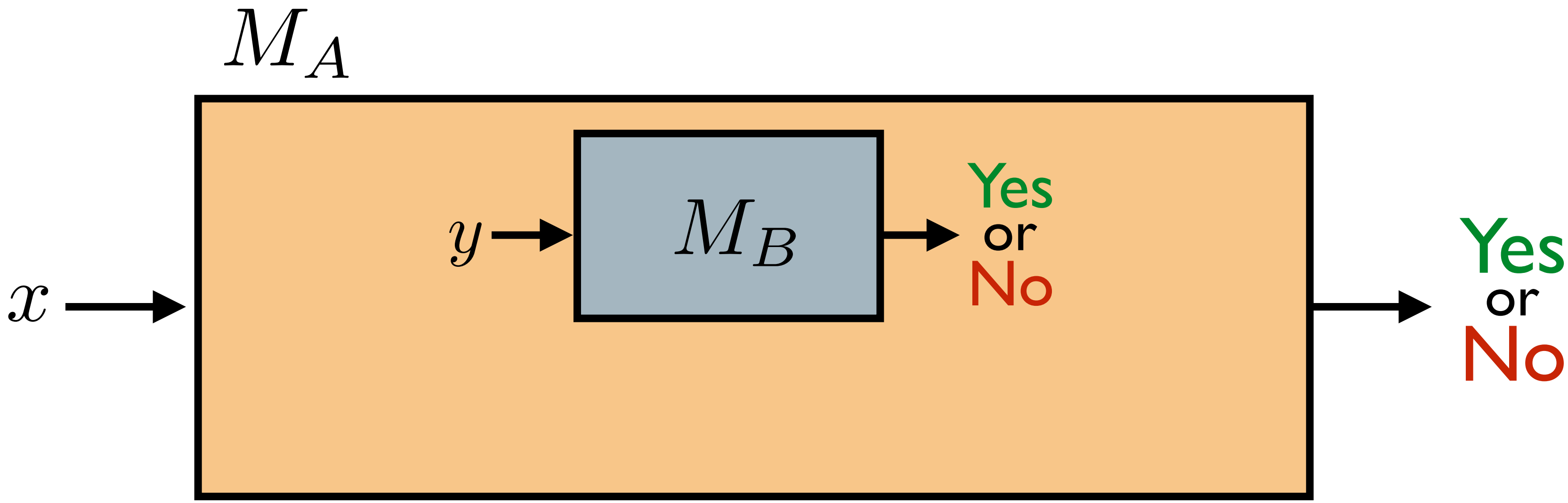
First:
An important note about reductions

Cook reduction

We have defined NP-hardness using polynomial-time Turing reductions.

These reductions are also known as Cook reductions.

$$A \leq_T^P B$$



"You can solve A in poly-time by using an oracle that solves B."

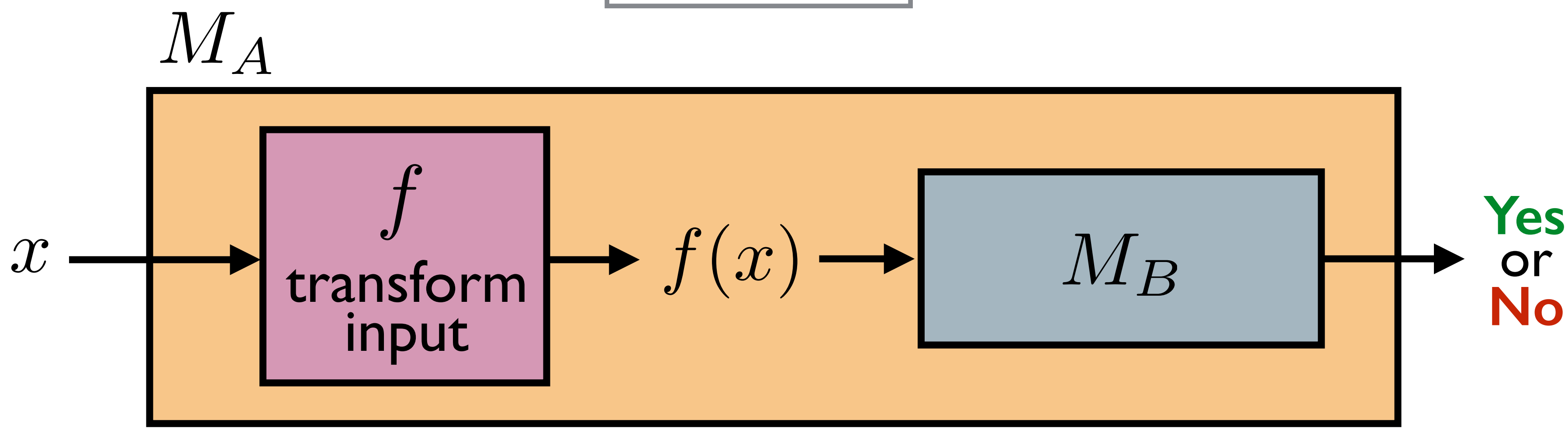
You can call the oracle poly(|x|) times.

Karp reduction

For technical reasons (which you might explore in HW) **NP**-hardness is not usually defined using Cook reductions.

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

$$\text{A} \le_m^P \text{B}$$



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

We must have:

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

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

Karp reduction

Definition:

Let A and B be two languages.

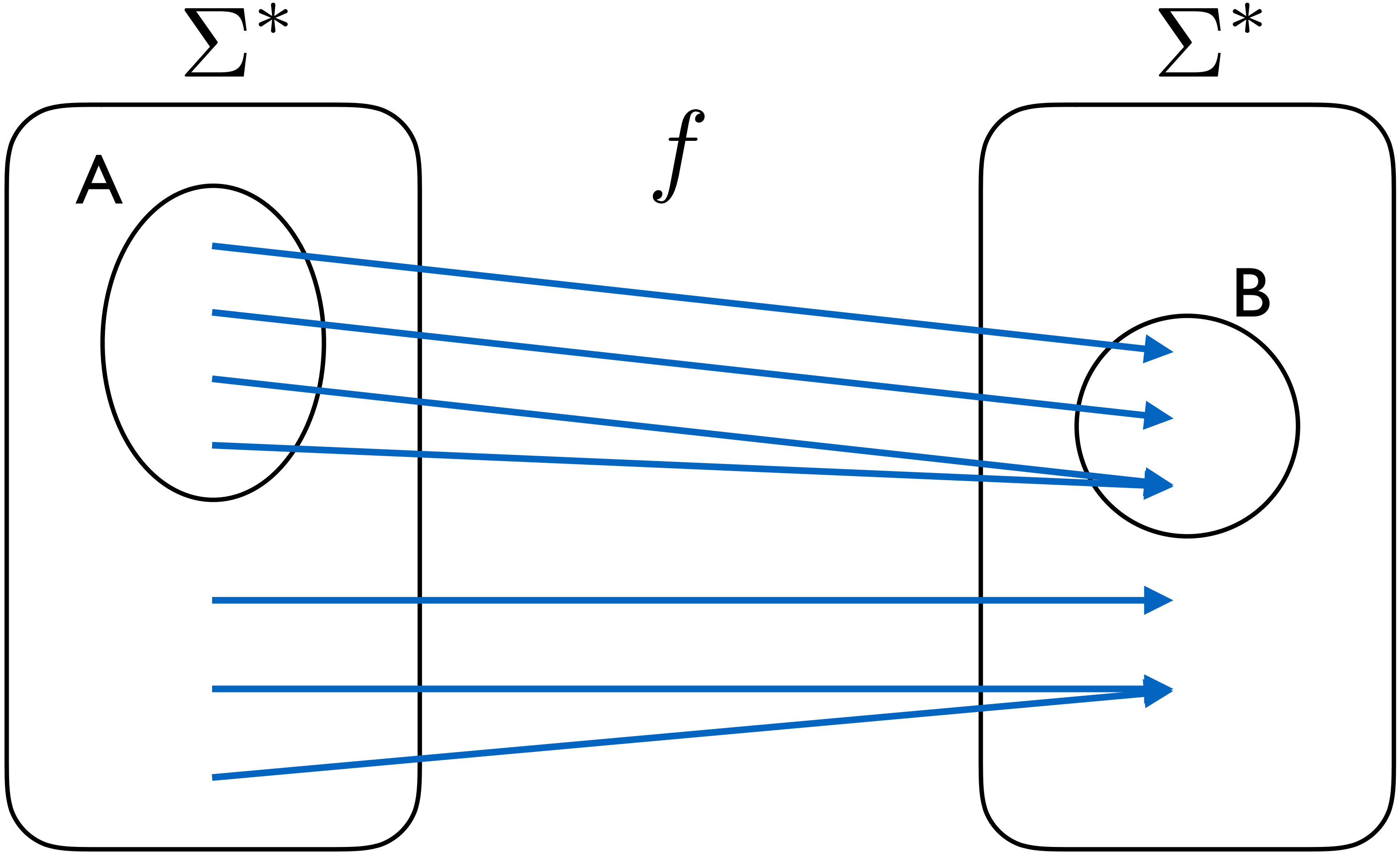
We say there is a polynomial-time many-one reduction from A to B (or a Karp reduction from A to B) if there is a polynomial-time computable function

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

such that: $x \in$ A if and only if $f(x) \in$ B.

In this case, we write A $\leq_m^P$ 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.

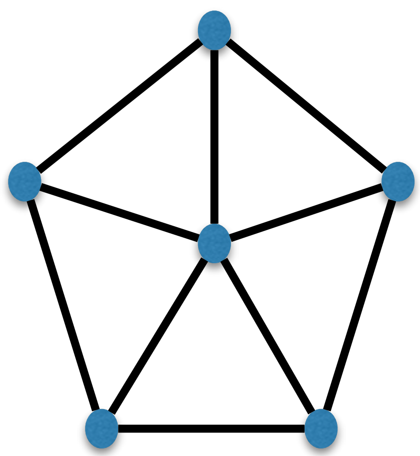
Karp Reduction: Example

Want:

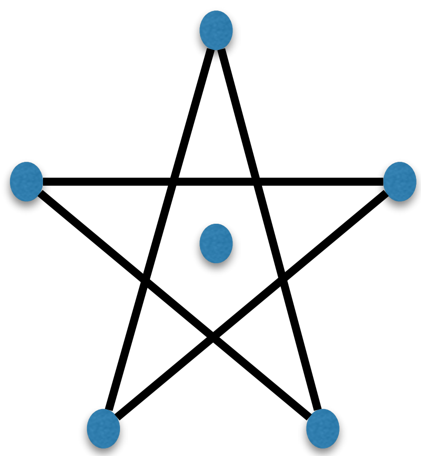
$$\langle G, k \rangle \to \langle G', k' \rangle$$

G has a clique of size k **iff**
G' has an independent 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 $f : \Sigma^* \to \Sigma^*$.

Definition of the function:

- If w is not a valid 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 $\langle G^* = (V, E^*), k \rangle$.

Karp Reduction: Example

Proof (continued):

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

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

   This implies in the complement graph G*,
   S is an IS of size k.

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.

This means in the complement of G*, which is G, S is a clique of size k.

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.

□

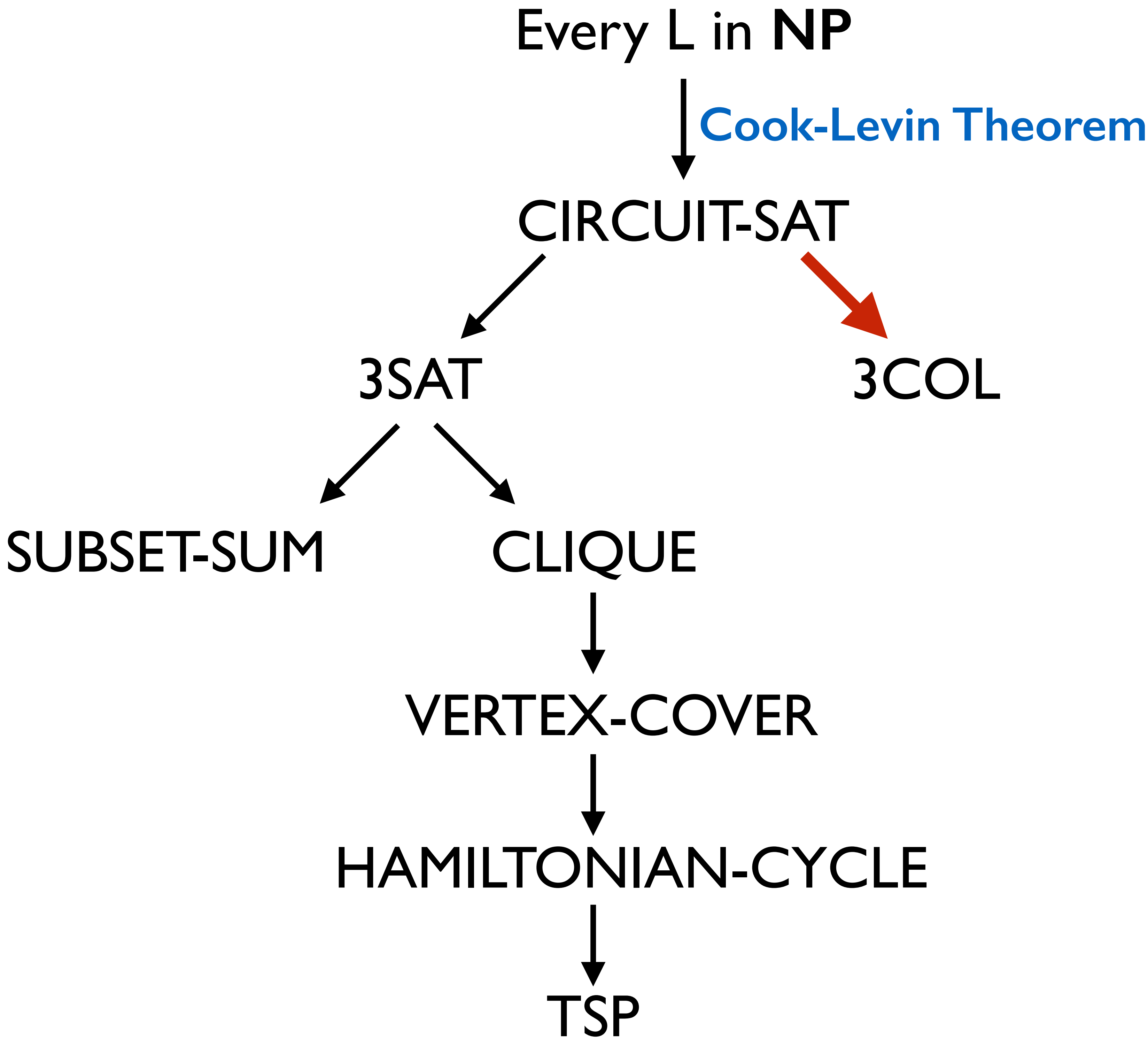
Can define **NP**-hardness with respect to $\leq_T^P$.

(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.


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

3COL is NP-complete

CIRCUIT-SAT $\leq$ 3COL: High level steps

We have already seen 3COL is in **NP** (sort of).

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

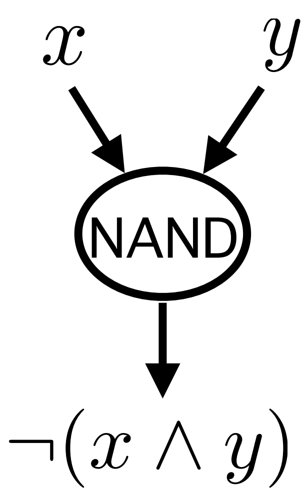
If x is not an encoding $\langle C \rangle$ of a valid circuit C, map it to ϵ.

So assume x is a valid encoding of a circuit.

Transform the circuit into an equivalent one that consists of only NAND gates.
(in addition to input gates and constant gates)

CIRCUIT-SAT ≤ 3COL: The main gadget

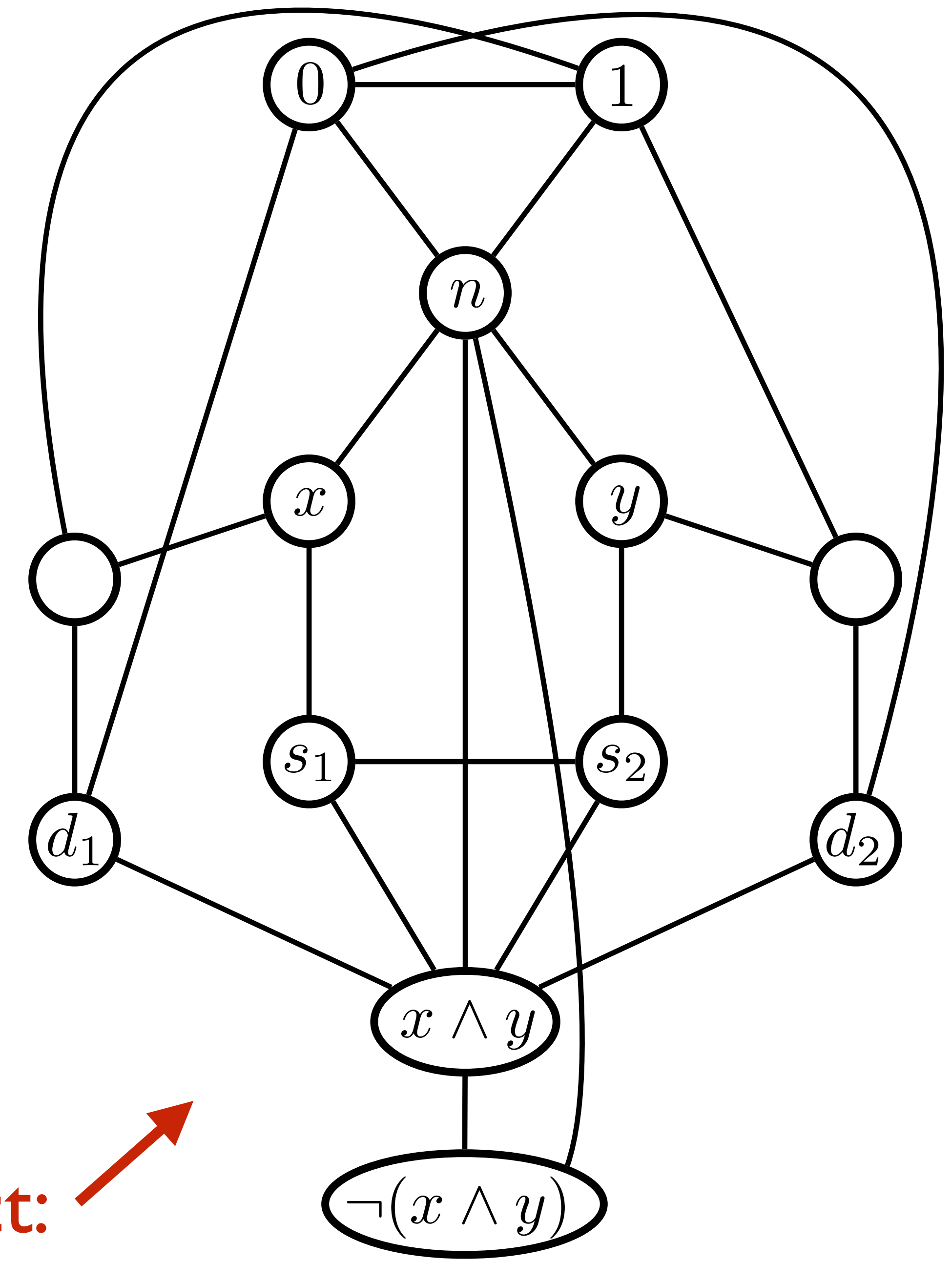
Consider a NAND gate.



x and y represent some other gates.

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

For each NAND gate, construct:



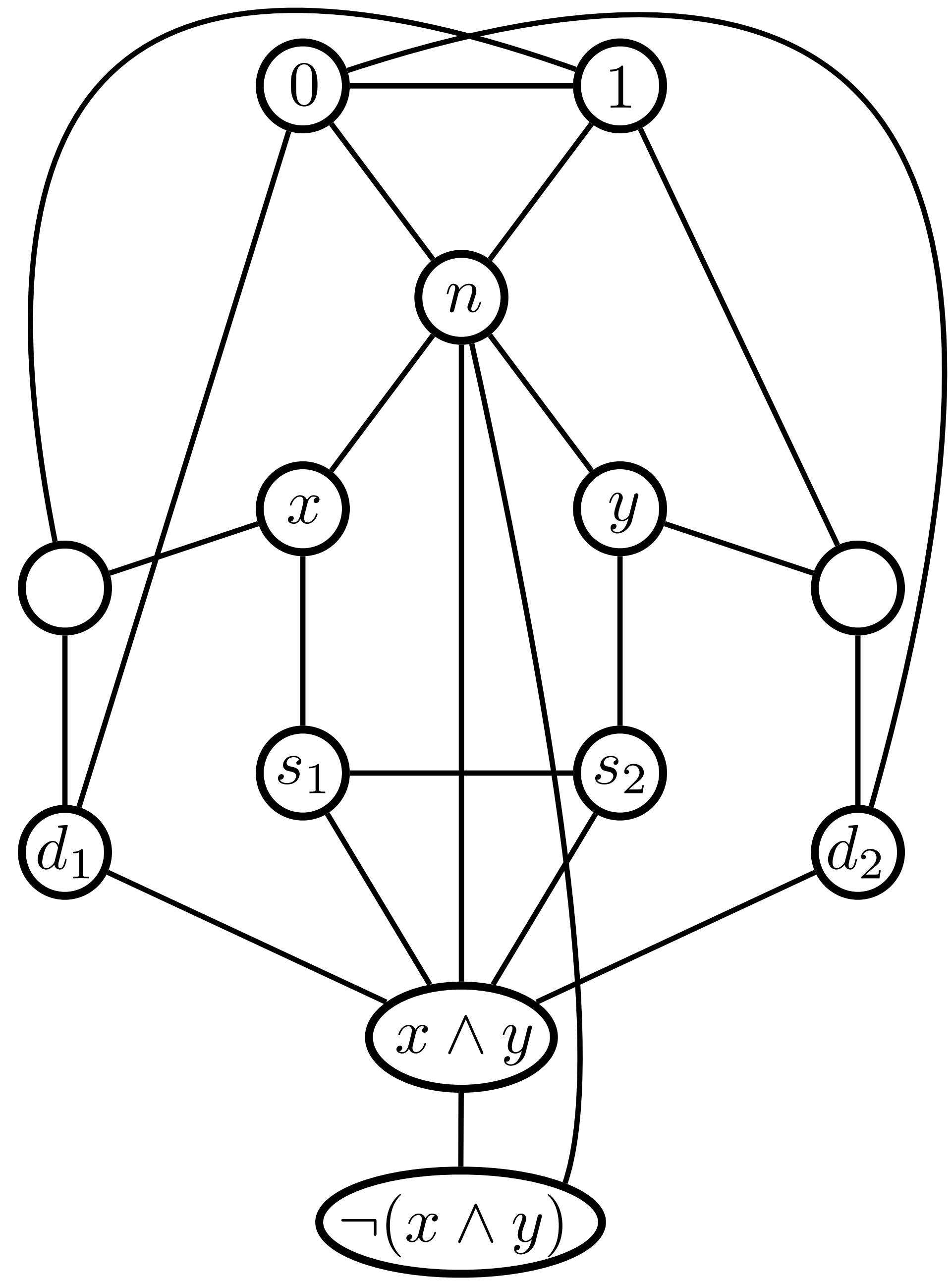
CIRCUIT-SAT ≤ 3COL: The main gadget

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:

Observation1:

vertices $x, \; y$

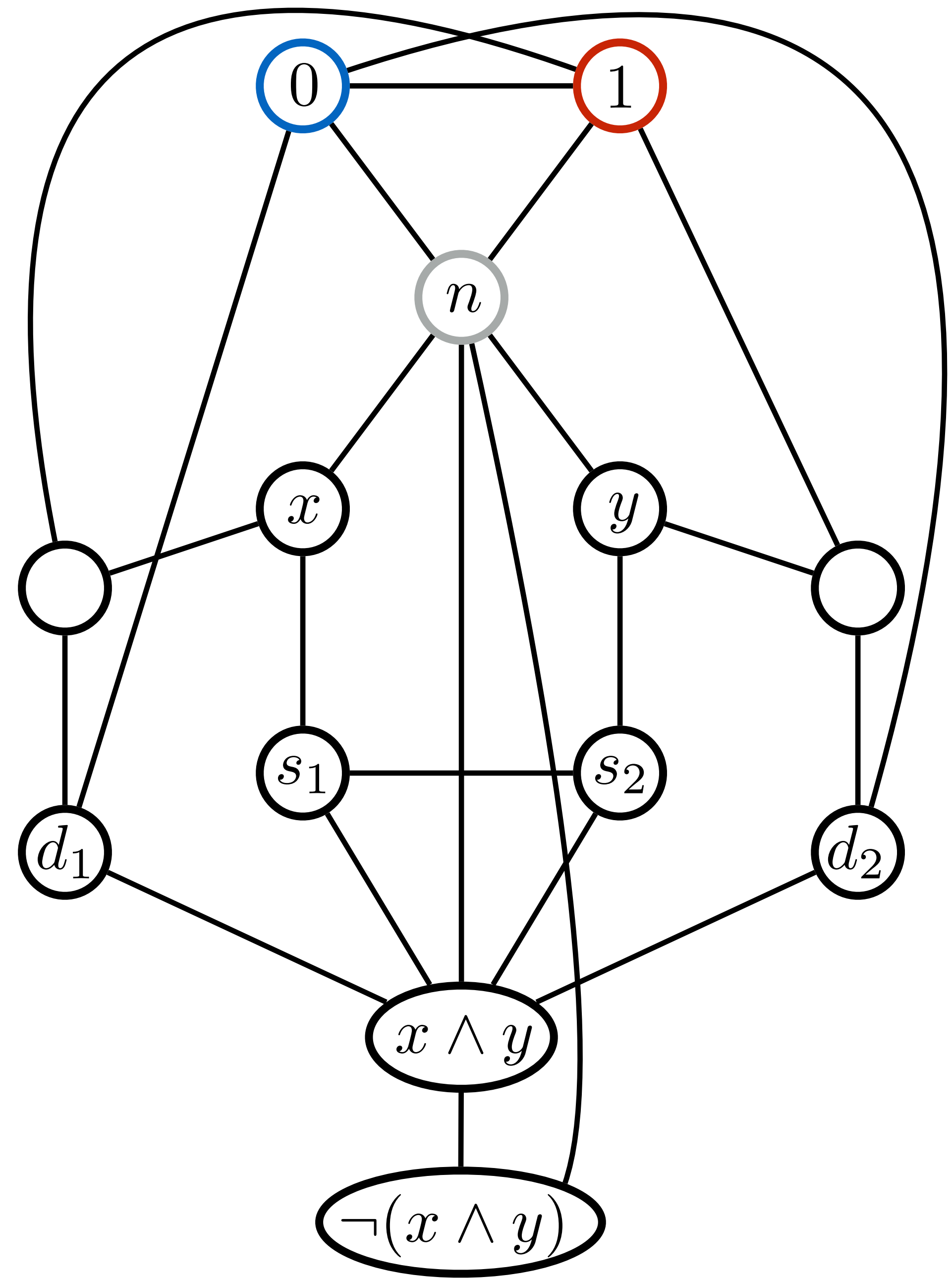
$x \wedge y$ and $\neg(x \wedge y)$

will not be assigned the color n.

Observation2:

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

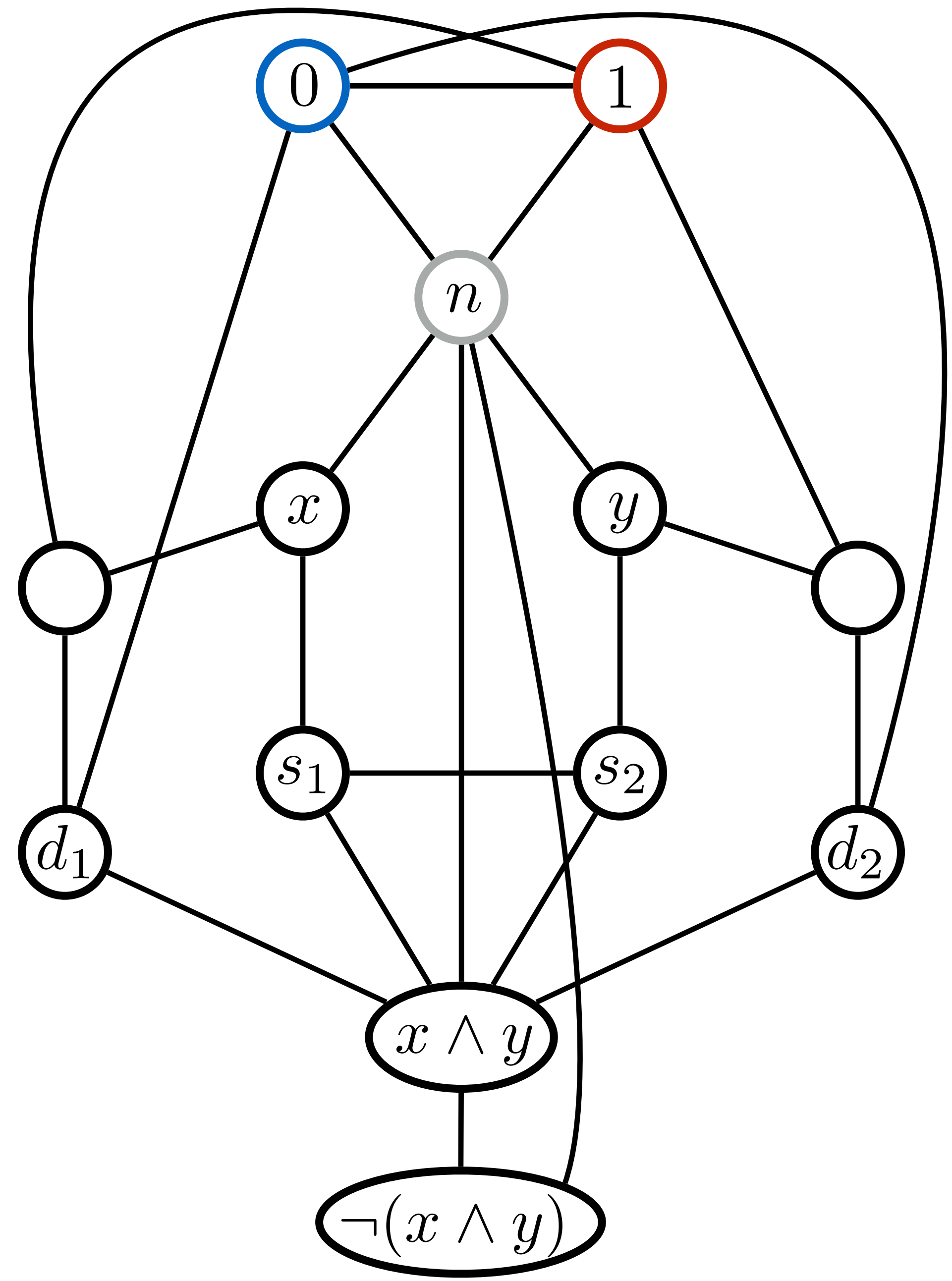
will be assigned different colors.



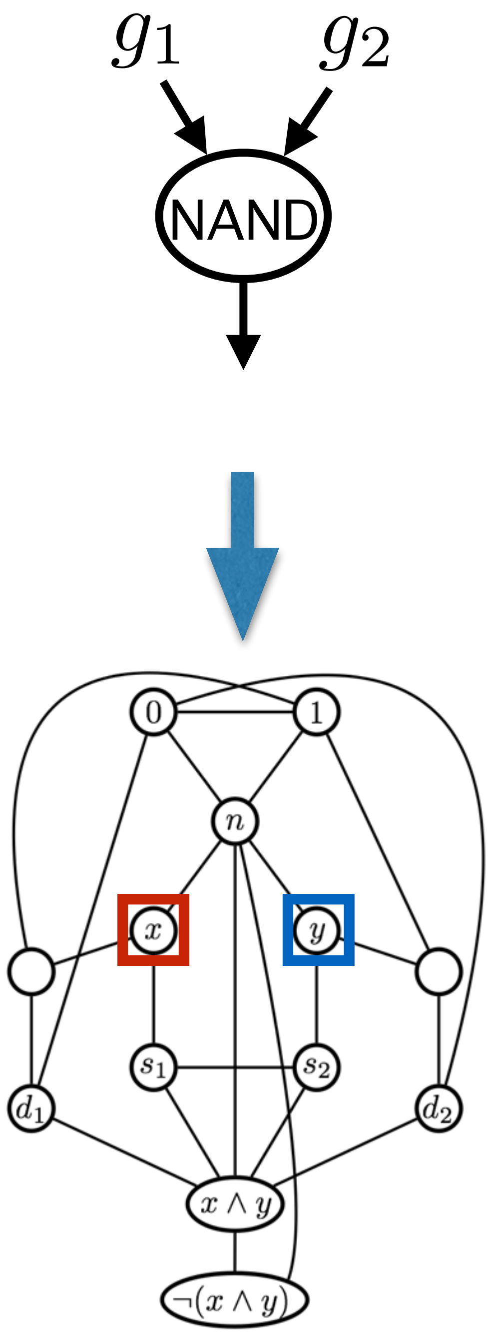
CIRCUIT-SAT ≤ 3COL: The main gadget

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

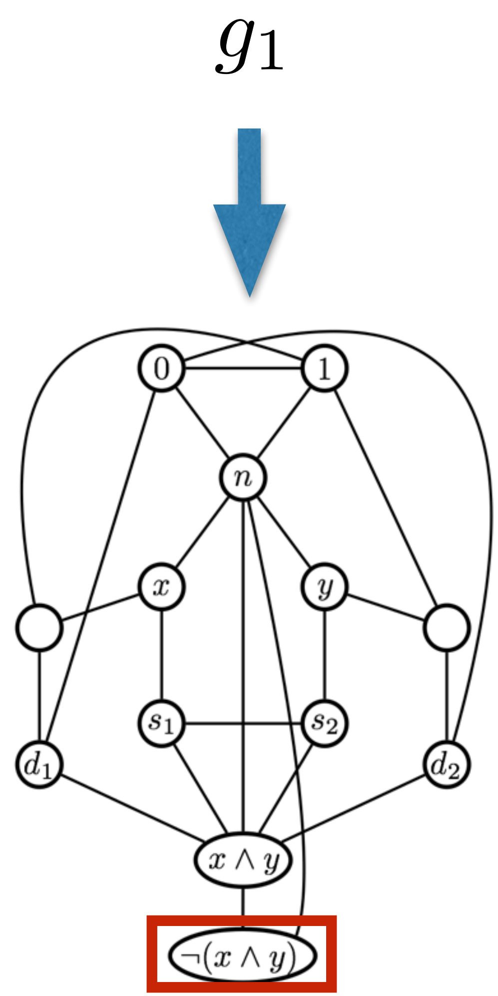


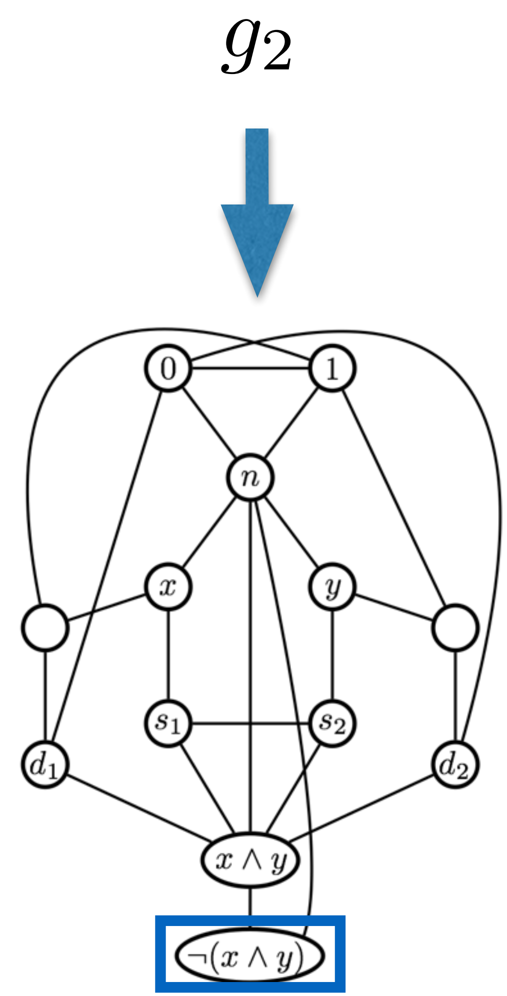
CIRCUIT-SAT ≤ 3COL: Rest of construction



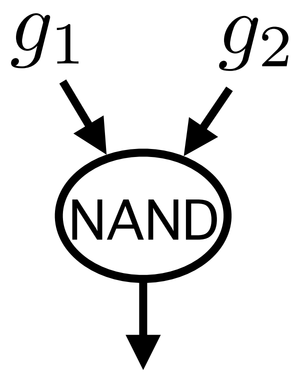
blue vertices are the same vertex.

red vertices are the same vertex.

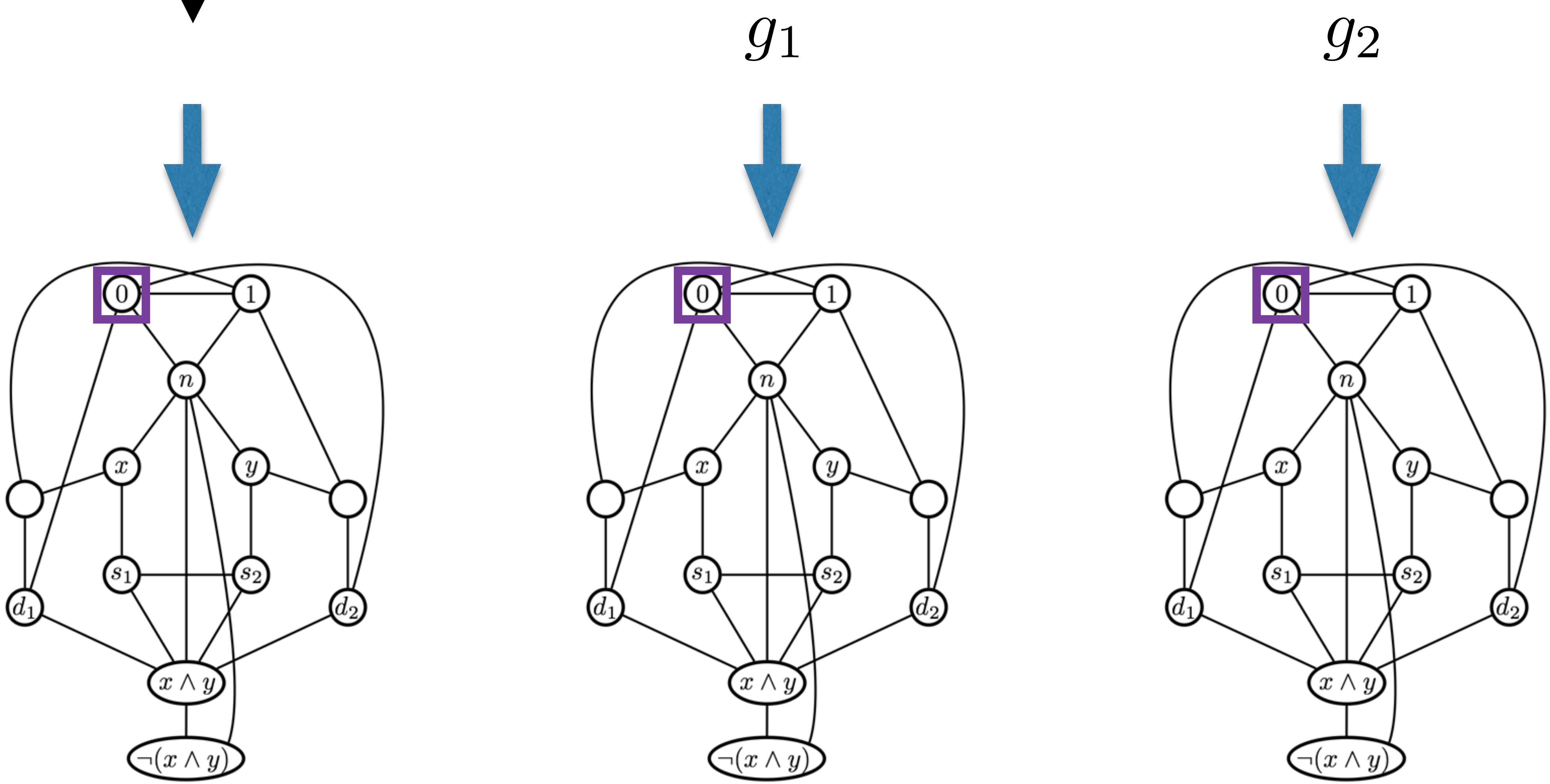




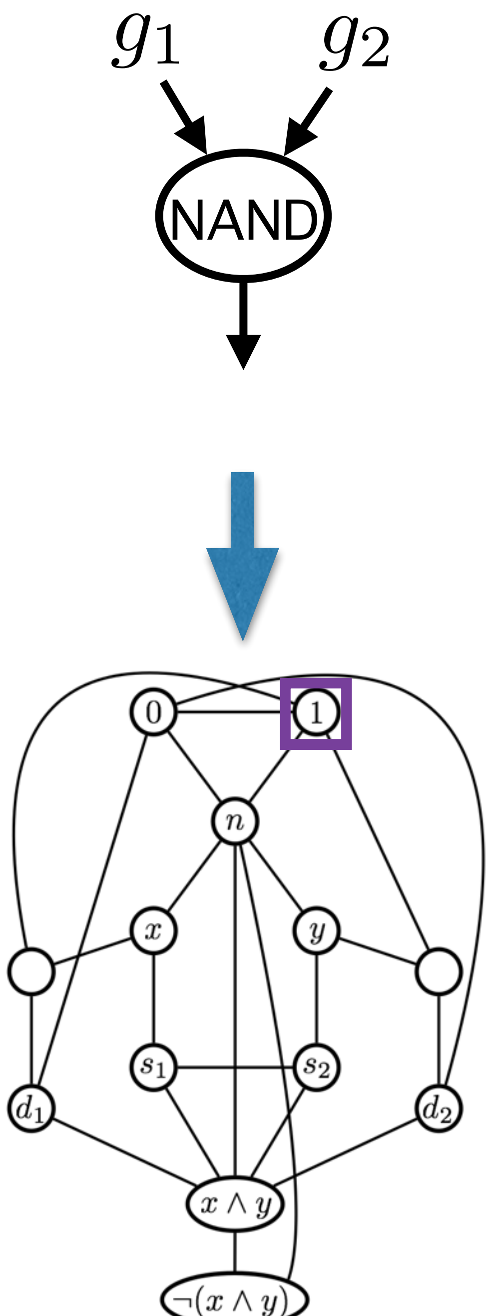
CIRCUIT-SAT ≤ 3COL: Rest of construction



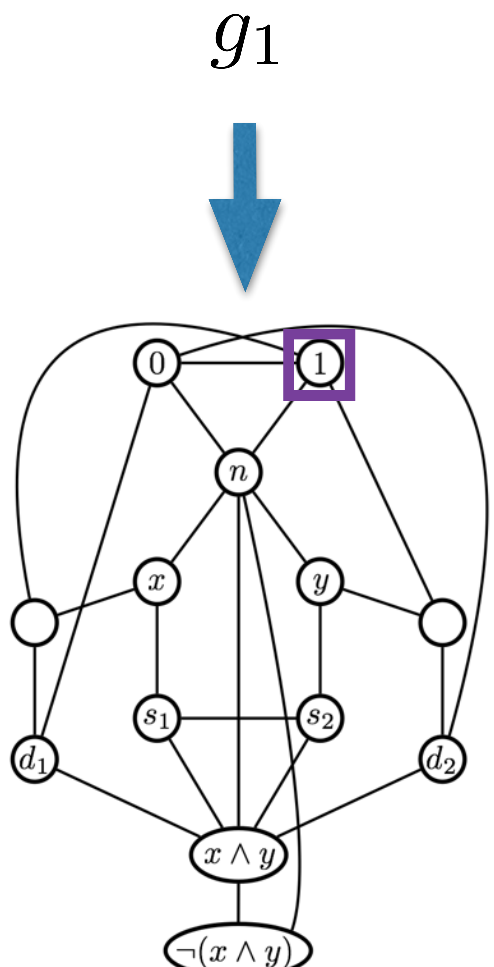
vertices labeled **0** are all the same.

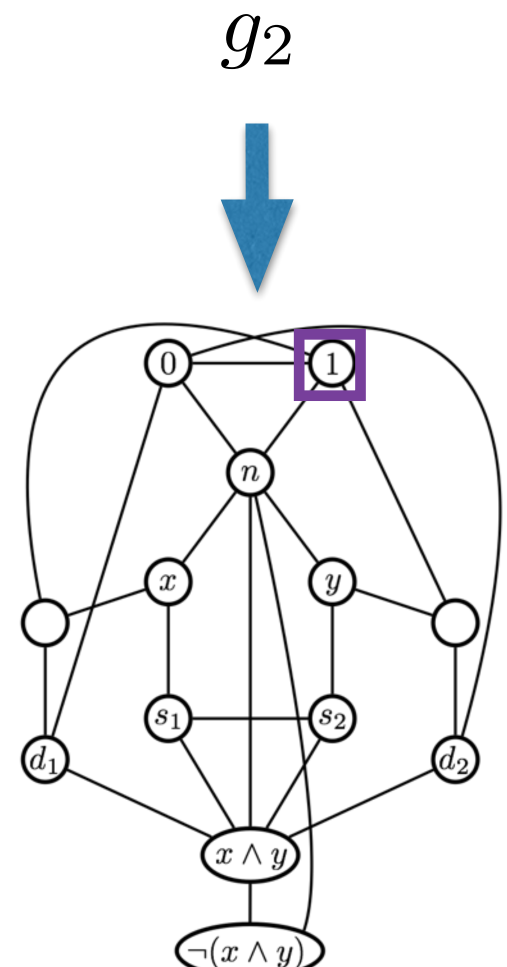


CIRCUIT-SAT ≤ 3COL: Rest of construction

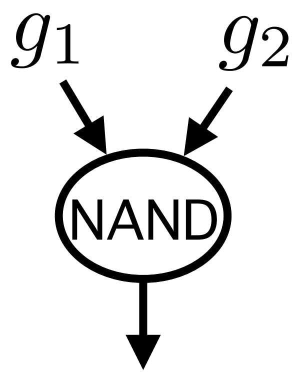


vertices labeled 1 are all the same.

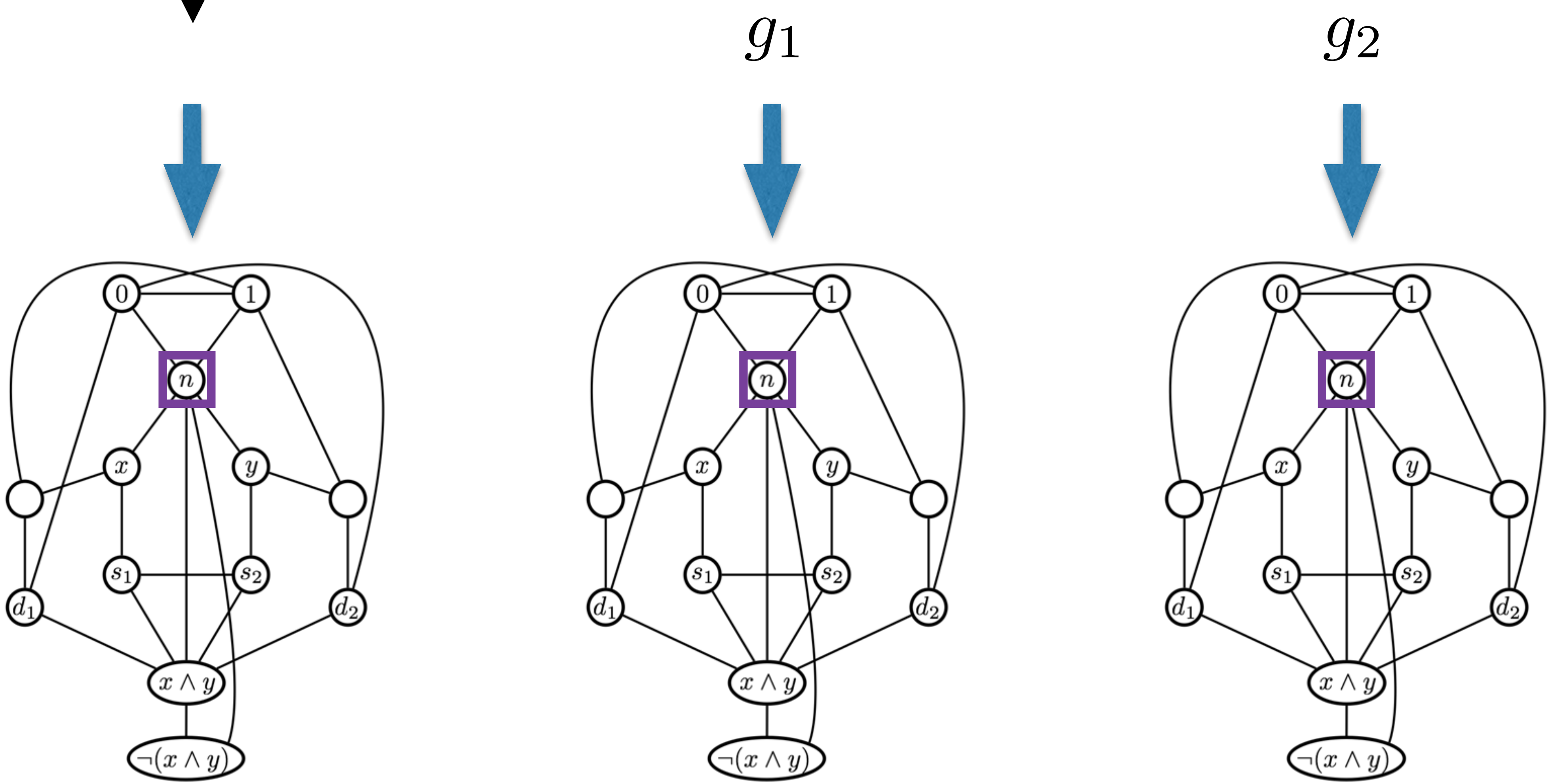




CIRCUIT-SAT ≤ 3COL: Rest of construction



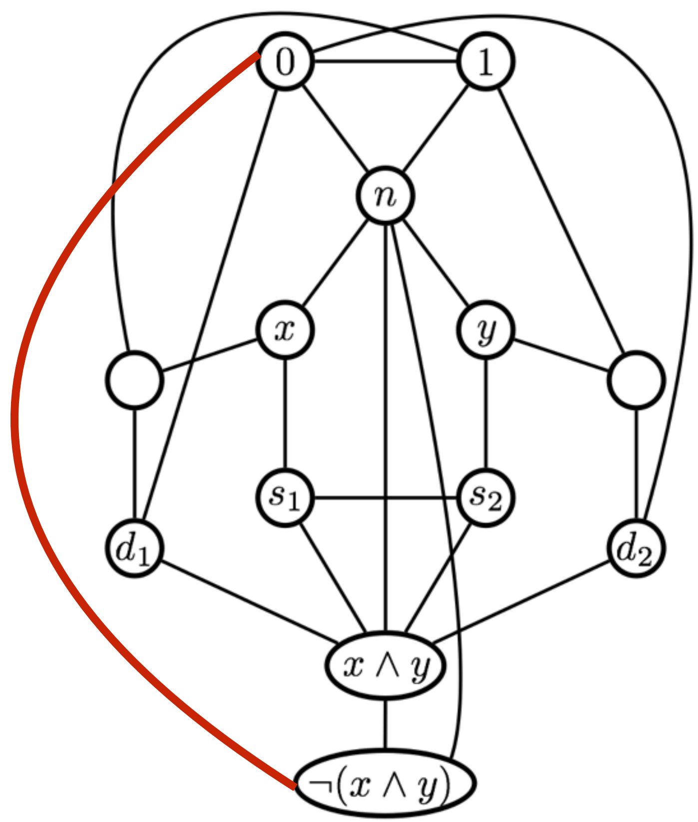
vertices labeled **n** are all the same.



CIRCUIT-SAT ≤ 3COL: Rest of construction

Input gates just map to a single vertex.

For the gadget corresponding to the **output gate**, we have one extra edge:



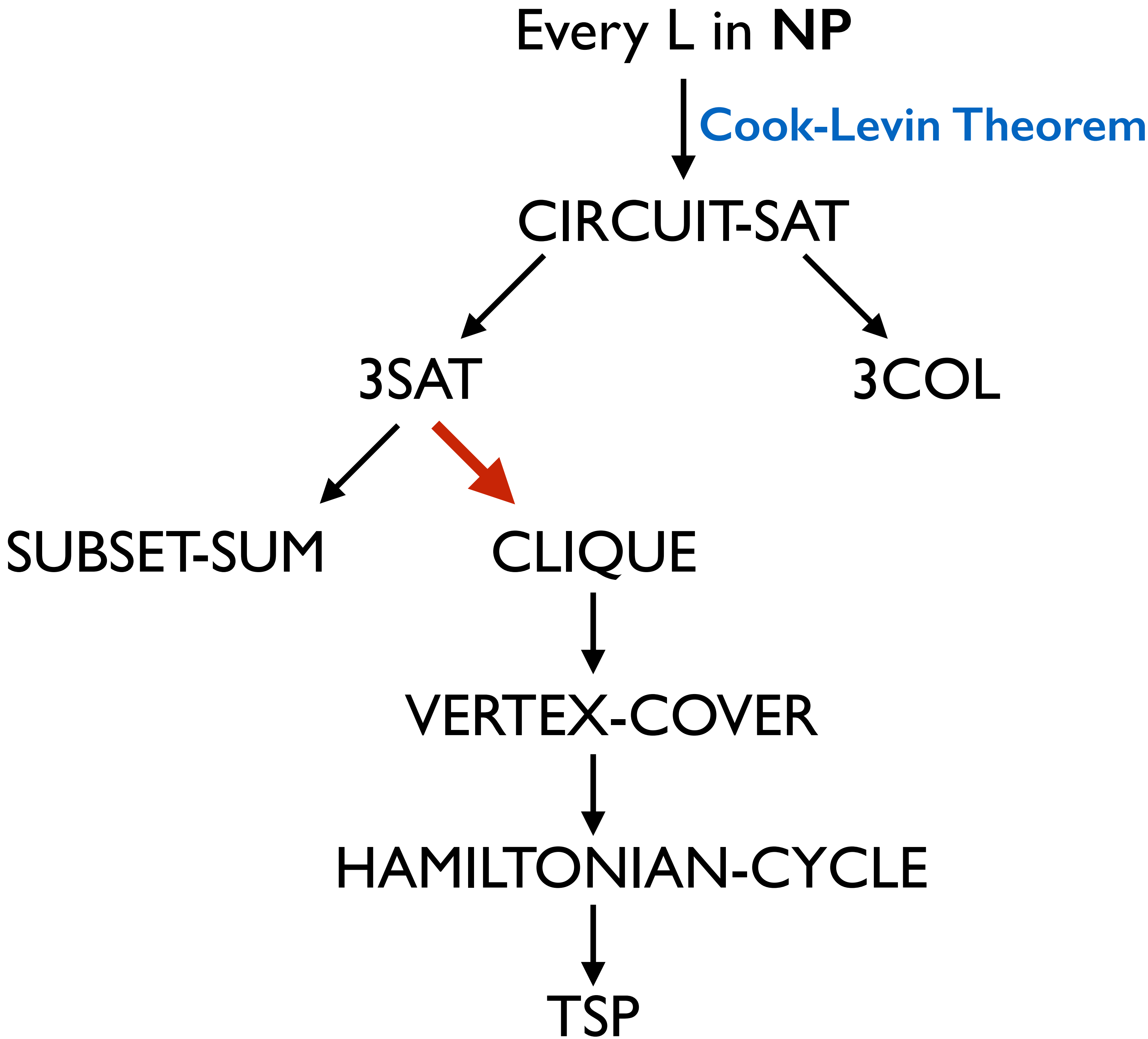
CIRCUIT-SAT $\leq$ 3COL: Why does it work?

Convince yourself that:

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

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

f is computable in polynomial time.


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.

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

a clause
(an OR of literals)

literal: a variable or its negation

conjunctive normal form: AND of clauses.

To satisfy the formula, you need to satisfy each clause.

Output: Yes iff the formula is satisfiable.

3SAT $\leq$ CLIQUE: High level steps

We have already seen CLIQUE is in **NP**.

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

<u>We need to:</u>

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 $f : \Sigma^* \rightarrow \Sigma^*$.

Words that don't correspond to a valid encoding of a 3SAT formula get mapped to ϵ.

So assume we are given a valid 3SAT formula φ (with m clauses).

We construct $\langle G, k \rangle$ from φ. (we set k = m)

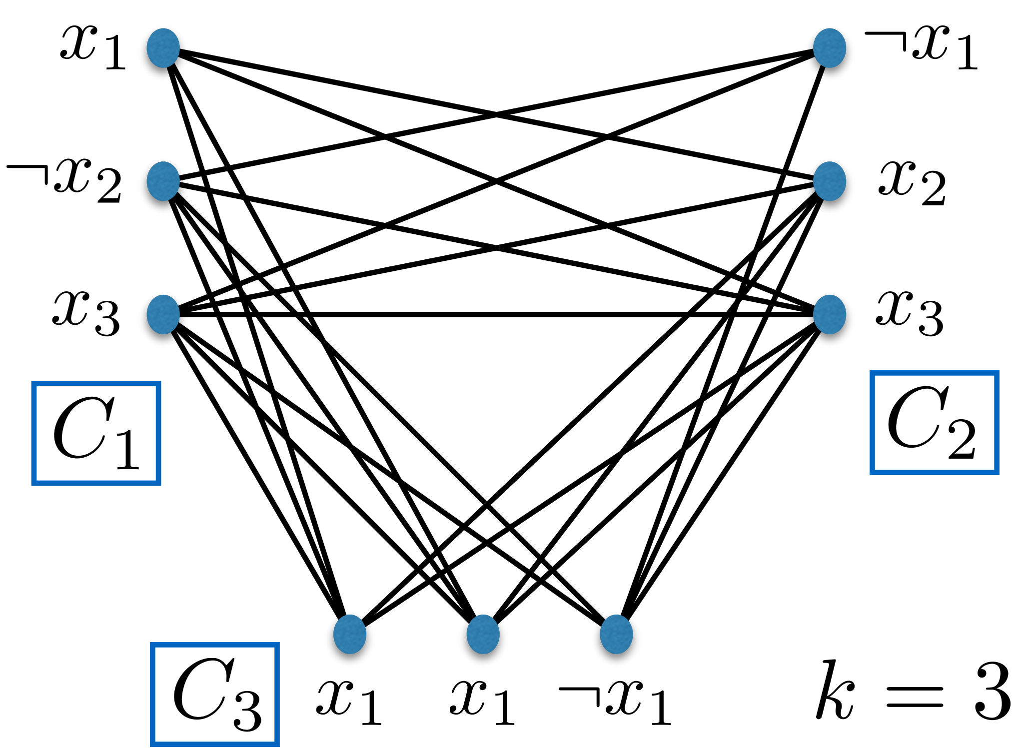
Construction demonstrated with an example.

3SAT ≤ CLIQUE: Defining the map

$C_1 \quad \wedge \quad C_2 \quad \wedge \quad C_3$

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

G_φ



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 has a clique of size m:

φ is satisfiable $\Longrightarrow$

$\exists$ a truth assignment to variables such that all the clauses are satisfied.

i.e., in each clause, there is a literal set to True.

The vertices corresponding to these literals form a clique of size m.

- two such literals/vertices are not connected only if one is the negation of the other.

3SAT $\leq$ CLIQUE: Why it works

If G_φ has a clique of size m, then φ is satisfiable:

G_φ has a clique K of size m $\implies$

there is exactly one vertex from each clause in K.

Claim: The literals corresponding to these vertices can be set to True. (i.e., φ is satisfiable)

Proof: Only way we could not do this is if
K contains a literal and its negation.
But a literal and its negation cannot be both in K
(since there is no edge between them).

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.

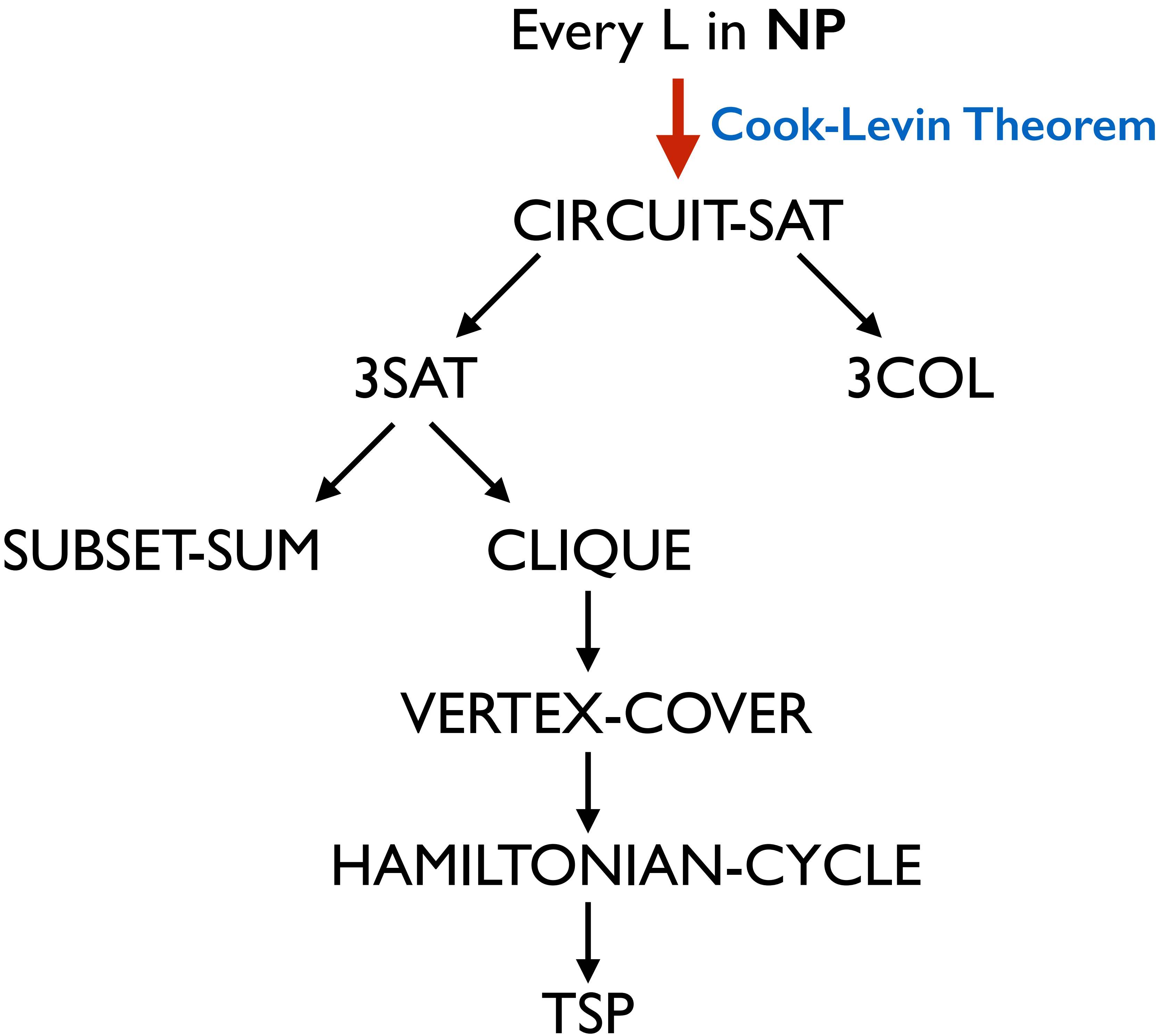
Creating the edge set:

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

□

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

CIRCUIT-SAT is NP-complete

Recall

Theorem: Let $f : \{0,1\}^* \to \{0,1\}$ be a decision problem which can be decided in time $O(T(n))$.
Then it can be computed by a circuit family of size $O(T(n)^2)$.

With this Theorem, it is actually easy to prove that

CIRCUIT-SAT is **NP**-hard.

Proof Sketch

WTS: for an arbitrary L in **NP**, L $\leq_m^P$ CIRCUIT-SAT.

i.e., we need to map $x \in \Sigma^*$ to a circuit C_x such that:

$$x \in L \iff C_x \text{ is satisfiable.}$$

Since L is in **NP**, there is a poly-time verifier TM V s.t.:

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

Let C be a poly-size circuit that simulates V.

For $x \in \Sigma^*$, let C_x be C with x-variables set to x.
(u-variables are the input)

$$x \in L \iff \exists u \text{ s.t. } V(x,u) = 1$$
$$\iff C_x \text{ is satisfiable.}$$

□