NP-Hardness

Textbook Reading

Chapter 34

Overview

- Computational (in)tractability
- Decision problems and optimization problems
- Decision problems and formal languages
- The class P
- Decision and verification
- The class NP
- NP hardness and NP completeness
- Polynomial-time reductions

NP-complete problems:

- Satisfiability
- Vertex cover
- Hamiltonian cycle
- Subset sum

Computational (In)Tractability

A problem is considered computationally tractable if it has a polynomial-time solution.

If no such solution exists, the problem is considered computationally intractable.

Computational (In)Tractability

A problem is considered computationally tractable if it has a polynomial-time solution.

If no such solution exists, the problem is considered computationally intractable.

Tractable problems:

- Sorting
- Shortest paths
- Minimum spanning tree
- Sequence alignment
- …

Computational (In)Tractability

A problem is considered computationally tractable if it has a polynomial-time solution.

If no such solution exists, the problem is considered computationally intractable.

Tractable problems:

- Sorting
- Shortest paths
- Minimum spanning tree
- Sequence alignment
- …

(Probably) intractable problems:

- Satisfiability
- Vertex cover
- Hamiltonian cycle
- Subset sum
- …

Decision Problems & Optimization Problems

A decision problem asks a yes/no question:

- Is this input sequence sorted?
- Does there exist a path from v to w in G?
- Does G contain a cycle?
- Are there two points in S that have distance less than d from each other?
- …

Decision Problems & Optimization Problems

A decision problem asks a yes/no question:

- Is this input sequence sorted?
- Does there exist a path from v to w in G?
- Does G contain a cycle?
- Are there two points in S that have distance less than d from each other?
- …

Every optimization problem has a corresponding decision problem:

- Does G have a spanning tree of weight at most w?
- Is there a path from v to w in G of length at most ℓ?
- Are there k non-overlapping intervals in S?
- …

Decision Problems & Optimization Problems

A decision problem asks a yes/no question:

- Is this input sequence sorted?
- Does there exist a path from v to w in G?
- Does G contain a cycle?
- Are there two points in S that have distance less than d from each other?
- …

Every optimization problem has a corresponding decision problem:

- Does G have a spanning tree of weight at most w?
- Is there a path from v to w in G of length at most ℓ?
- Are there k non-overlapping intervals in S?
- …

To turn an optimization problem into a decision problem, we provide a threshold for the cost/weight/… of the solution.

Decision Is No Harder Than Optimization

Yes/no answers usually aren't that useful in practice.

However, if we can provide evidence that the decision version of an optimization problem is intractable, then so is the optimization problem itself, by the following lemma:

Lemma. If an optimization problem can be solved in polynomial time, then so can its decision version.

Decision Is No Harder Than Optimization

Yes/no answers usually aren't that useful in practice.

However, if we can provide evidence that the decision version of an optimization problem is intractable, then so is the optimization problem itself, by the following lemma:

Lemma. If an optimization problem can be solved in polynomial time, then so can its decision version.

Decision algorithm:

- Solve the optimization problem.
- Compare the value of its solution to the given threshold.

Decision Problems & Formal Languages

A (formal) language over an alphabet Σ is a set of strings formed using letters from Σ: $L \subseteq \Sigma^*$.

Decision Problems & Formal Languages

A (formal) language over an alphabet Σ is a set of strings formed using letters from Σ: $L \subseteq \Sigma^*$.

Formal languages and decision problems are "the same thing".

Decision Problems & Formal Languages

A (formal) language over an alphabet Σ is a set of strings formed using letters from Σ: $L \subseteq \Sigma^*$.

Formal languages and decision problems are "the same thing".

Language $\rightarrow$ decision problem:

- Given a language L, decide whether a given string $x \in \Sigma^*$ belongs to L.

Decision Problems & Formal Languages

A (formal) language over an alphabet Σ is a set of strings formed using letters from Σ: $L \subseteq \Sigma^*$.

Formal languages and decision problems are "the same thing".

Language $\to$ decision problem:

- Given a language L, decide whether a given string $x \in \Sigma^*$ belongs to L.

Decision problem $\to$ language:

- Define a binary encoding of the input instances of the decision problem.
- Every instance is now a string over the alphabet $\Sigma = \{0, 1\}$.
- Let L be the set of all such strings that encode yes-instances of the decision problem.

Decision Problems & Formal Languages

A (formal) language over an alphabet Σ is a set of strings formed using letters from Σ: $L \subseteq \Sigma^*$.

Formal languages and decision problems are "the same thing".

Language $\rightarrow$ decision problem:

- Given a language L, decide whether a given string $x \in \Sigma^*$ belongs to L.

Decision problem $\rightarrow$ language:

- Define a binary encoding of the input instances of the decision problem.
- Every instance is now a string over the alphabet $\Sigma = \{0, 1\}$.
- Let L be the set of all such strings that encode yes-instances of the decision problem.

Consider the transformations

- Problem $P \rightarrow$ language $L \rightarrow$ problem P'
- Language $L \rightarrow$ problem $P \rightarrow$ language L'

Then $P = P'$ and $L = L'$.

The Complexity Class P

Given a string $x \in \Sigma^*$, a decision algorithm D is said to accept x if it answers yes given input x; it rejects x if it answers no given input x.

The Complexity Class P

Given a string $x \in \Sigma^*$, a decision algorithm D is said to accept x if it answers yes given input x; it rejects x if it answers no given input x.

Algorithm D is said to decide a language $L \subseteq \Sigma^*$ if it accepts all strings in L and rejects all other strings.

In other words, the output of D is the answer to the question "Does x belong to L?"

The Complexity Class P

Given a string $x \in \Sigma^*$, a decision algorithm D is said to accept x if it answers yes given input x; it rejects x if it answers no given input x.

Algorithm D is said to decide a language $L \subseteq \Sigma^*$ if it accepts all strings in L and rejects all other strings.

In other words, the output of D is the answer to the question "Does x belong to L?"

The complexity class P is the set of all languages that can be decided in polynomial time.

Formally, a language L belongs to P if and only if there exists an algorithm D that decides L and the running time of D on any input $x \in \Sigma^*$ is in $O(|x|^c)$ for some constant c.

The Complexity Class P

Given a string $x \in \Sigma^*$, a decision algorithm D is said to accept x if it answers yes given input x; it rejects x if it answers no given input x.

Algorithm D is said to decide a language $L \subseteq \Sigma^*$ if it accepts all strings in L and rejects all other strings.

In other words, the output of D is the answer to the question "Does x belong to L?"

The complexity class P is the set of all languages that can be decided in polynomial time.

Formally, a language L belongs to P if and only if there exists an algorithm D that decides L and the running time of D on any input $x \in \Sigma^*$ is in $O(|x|^c)$ for some constant c.

Informally, P is the set of all tractable decision problems, since

- We observed that decision problems and formal languages are the same thing and
- We consider a problem tractable if it can be solved in polynomial time.

Verification

Consider an algorithm V that decides a language $L' \subseteq \Sigma^* \times \Sigma^*$, that is, its input is a pair (x, y) such that $x, y \in \Sigma^*$.

Algorithm V is said to verify a language L if

- for every $x \in L$, there exists a $y \in \Sigma^*$ such that $(x, y) \in L'$ and
- for every $x \notin L$, there is no $y \in \Sigma^*$ such that $(x, y) \in L'$.

Verification

Consider an algorithm V that decides a language $L' \subseteq \Sigma^* \times \Sigma^*$, that is, its input is a pair (x, y) such that $x, y \in \Sigma^*$.

Algorithm V is said to verify a language L if

- for every $x \in L$, there exists a $y \in \Sigma^*$ such that $(x, y) \in L'$ and
- for every $x \notin L$, there is no $y \in \Sigma^*$ such that $(x, y) \in L'$.

Thus, given an input (x, y) consisting of an element $x \in L$ and an appropriate "proof" $y \in \Sigma^*$ that shows that $x \in L$, V answers yes.

For a string $x \notin L$, we can provide whatever "proof" y of its membership in L we want; V will reject every such pair (x, y).

Verification

Consider an algorithm V that decides a language $L' \subseteq \Sigma^* \times \Sigma^*$, that is, its input is a pair (x, y) such that $x, y \in \Sigma^*$.

Algorithm V is said to verify a language L if

- for every $x \in L$, there exists a $y \in \Sigma^*$ such that $(x, y) \in L'$ and
- for every $x \notin L$, there is no $y \in \Sigma^*$ such that $(x, y) \in L'$.

Thus, given an input (x, y) consisting of an element $x \in L$ and an appropriate "proof" $y \in \Sigma^*$ that shows that $x \in L$, V answers yes.

For a string $x \notin L$, we can provide whatever "proof" y of its membership in L we want; V will reject every such pair (x, y).

Thus, we can think of V as a "proof checker" that verifies whether any given proof of x's membership in L is in fact correct.

Verification

Consider an algorithm V that decides a language $L' \subseteq \Sigma^* \times \Sigma^*$, that is, its input is a pair (x, y) such that $x, y \in \Sigma^*$.

Algorithm V is said to verify a language L if

- for every $x \in L$, there exists a $y \in \Sigma^*$ such that $(x, y) \in L'$ and
- for every $x \notin L$, there is no $y \in \Sigma^*$ such that $(x, y) \in L'$.

Thus, given an input (x, y) consisting of an element $x \in L$ and an appropriate "proof" $y \in \Sigma^*$ that shows that $x \in L$, V answers yes.

For a string $x \notin L$, we can provide whatever "proof" y of its membership in L we want; V will reject every such pair (x, y).

Thus, we can think of V as a "proof checker" that verifies whether any given proof of x's membership in L is in fact correct.

V does not decide whether $x \in L$. V may answer no even if $x \in L$ if the provided proof of its membership in L is incorrect.

Verifying is Easier Than Deciding

Verifying a language may be easier than deciding it.

Verifying is Easier Than Deciding

Verifying a language may be easier than deciding it.

Given a sequence $S = \langle x_1, x_2, \ldots, x_n \rangle$ of numbers, the element uniqueness problem asks us to decide whether there exist indices $i \neq j$ such that $x_i = x_j$.

Verifying is Easier Than Deciding

Verifying a language may be easier than deciding it.

Given a sequence $S = \langle x_1, x_2, \ldots, x_n \rangle$ of numbers, the element uniqueness problem asks us to decide whether there exist indices $i \neq j$ such that $x_i = x_j$.

Let L be the language of all sequences where two such indices exist.

Verifying is Easier Than Deciding

Verifying a language may be easier than deciding it.

Given a sequence $S = \langle x_1, x_2, \ldots, x_n \rangle$ of numbers, the element uniqueness problem asks us to decide whether there exist indices $i \neq j$ such that $x_i = x_j$.

Let L be the language of all sequences where two such indices exist.

It can be shown that, using comparisons only, it takes $\Omega(n \lg n)$ time in the worst case to decide whether a given sequence S belongs to L.

Verifying is Easier Than Deciding

Verifying a language may be easier than deciding it.

Given a sequence $S = \langle x_1, x_2, \ldots, x_n \rangle$ of numbers, the element uniqueness problem asks us to decide whether there exist indices $i \neq j$ such that $x_i = x_j$.

Let L be the language of all sequences where two such indices exist.

It can be shown that, using comparisons only, it takes $\Omega(n \lg n)$ time in the worst case to decide whether a given sequence S belongs to L.

Verifying L can be done in constant time!

- Let $L' = \{(S, (i, j)) \mid x_i = x_j, i \neq j\}$
- Given some pair $(S, (i, j))$, we can decide in constant time whether $(S, (i, j)) \in L'$ by comparing x_i and x_j.
- This algorithm verifies L because $x \in L$ if and only if there exists a pair (i, j) such that $(S, (i, j)) \in L'$.

The Complexity Class NP

The complexity class NP is the set of all languages that can be verified in polynomial time.

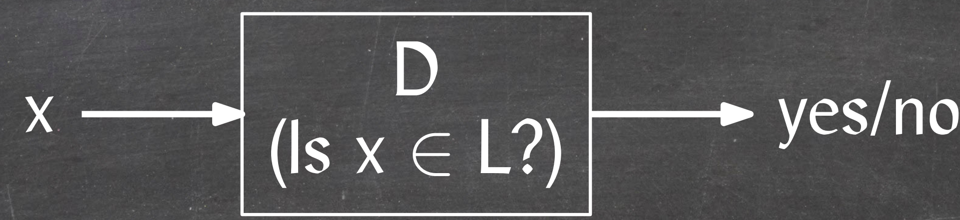
Formally, a language L belongs to NP if and only if there exists a language $L' \in P$ and a constant c such that $x \in L$ if and only if $(x, y) \in L'$ for some $y \in \Sigma^*$, $|y| \leq |x|^c$.

P versus NP

Lemma: $P \subseteq NP$. (Every language that can be decided in polynomial time can be verified in polynomial time.)

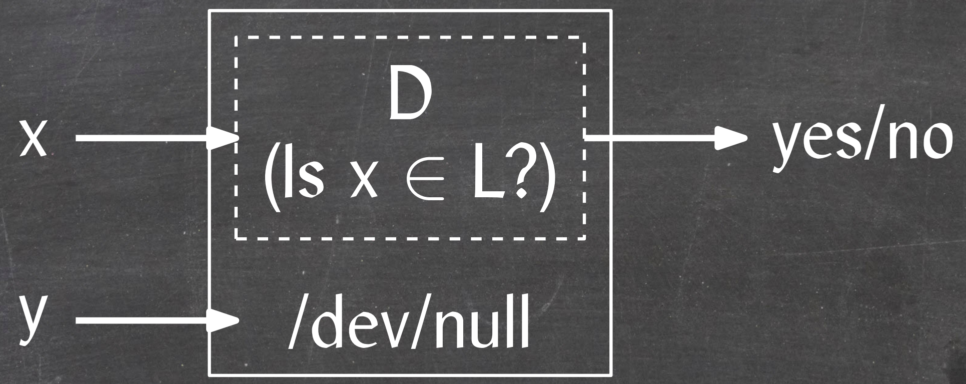
P versus NP

Lemma: P ⊆ NP. (Every language that can be decided in polynomial time can be verified in polynomial time.)



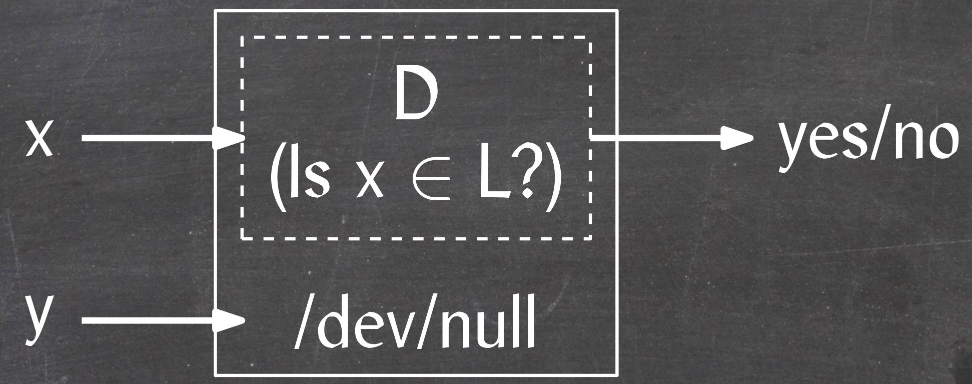
P versus NP

Lemma: P $\subseteq$ NP. (Every language that can be decided in polynomial time can be verified in polynomial time.)



P versus NP

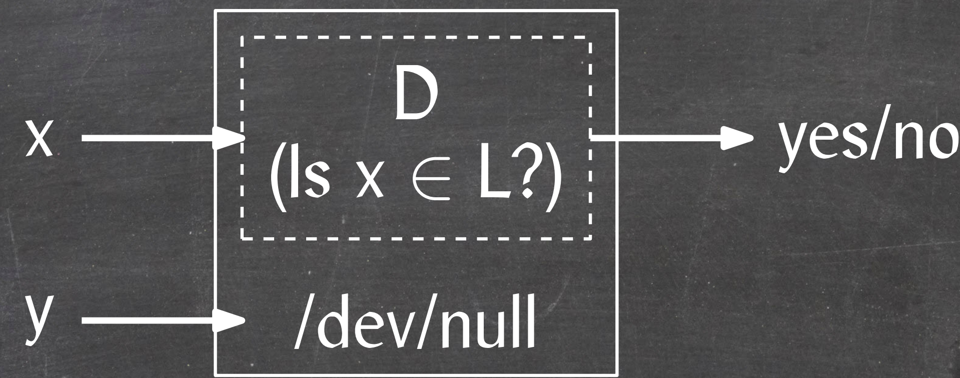
Lemma: $P \subseteq NP$. (Every language that can be decided in polynomial time can be verified in polynomial time.)



Is $P = NP$ or is $P \subset NP$?

P versus NP

Lemma: $P \subseteq NP$. (Every language that can be decided in polynomial time can be verified in polynomial time.)



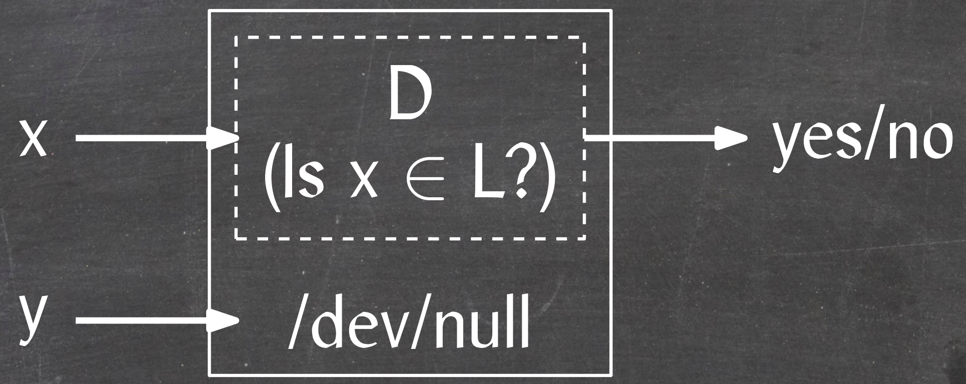
Is $P = NP$ or is $P \subset NP$?

Nobody knows the answer, but …

Given that we know verifying some languages is easier than deciding them, it is likely that $P \subset NP$.

P versus NP

Lemma: $P \subseteq NP$. (Every language that can be decided in polynomial time can be verified in polynomial time.)



Is $P = NP$ or is $P \subset NP$?

Nobody knows the answer, but …

Given that we know verifying some languages is easier than deciding them, it is likely that $P \subset NP$.

We will show that there exist languages that cannot be decided (decision problems that cannot be solved) in polynomial time unless $P = NP$!

NP-Hardness and NP-Completeness

A language L is NP-hard if $L \in P$ implies that $P = NP$.

NP-Hardness and NP-Completeness

A language L is NP-hard if $L \in P$ implies that $P = NP$.

A language L is NP-complete if

- $L \in NP$ and
- L is NP-hard.

Intuitively, NP-complete languages are the hardest languages in NP.

NP-Hardness and NP-Completeness

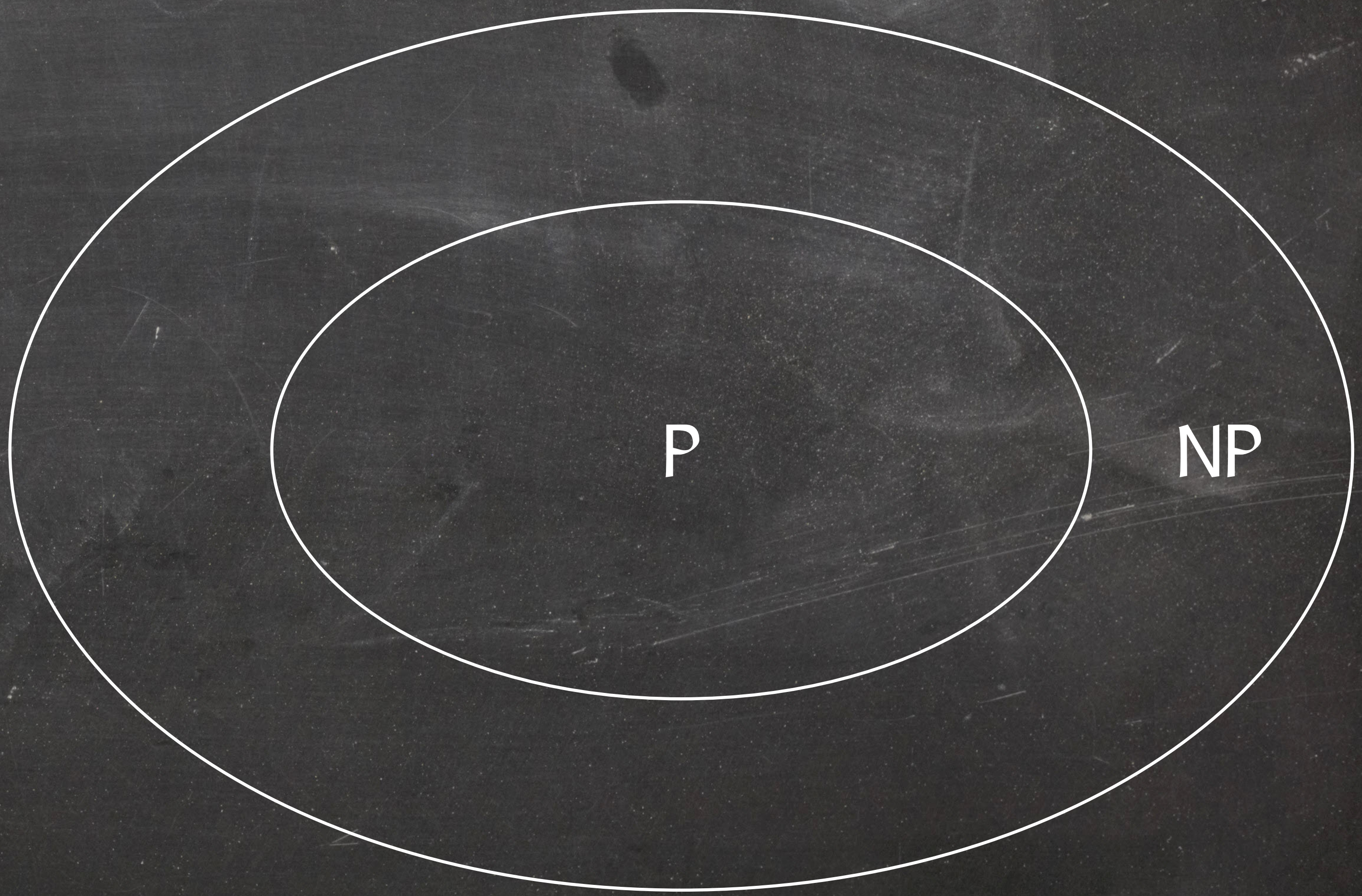
A language L is NP-hard if $L \in P$ implies that $P = NP$.

A language L is NP-complete if

- $L \in NP$ and
- L is NP-hard.

Intuitively, NP-complete languages are the hardest languages in NP.

Assume $P \neq NP$.



NP-Hardness and NP-Completeness

A language L is NP-hard if $L \in P$ implies that $P = NP$.

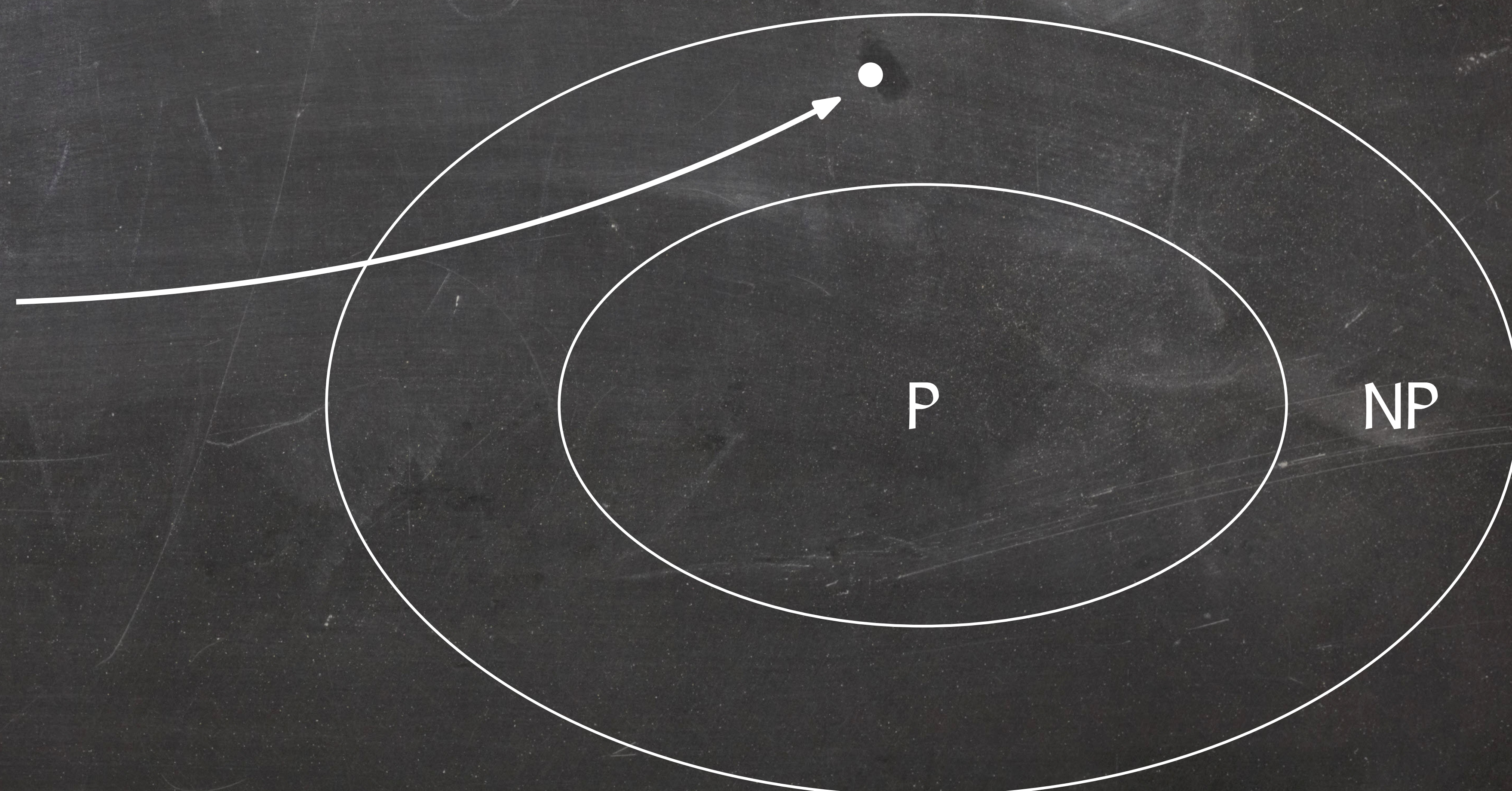
A language L is NP-complete if

- $L \in NP$ and
- L is NP-hard.

Intuitively, NP-complete languages are the hardest languages in NP.

Assume $P \neq NP$.

NP-complete if NP-hard



NP-Hardness and NP-Completeness

A language L is NP-hard if $L \in P$ implies that $P = NP$.

A language L is NP-complete if

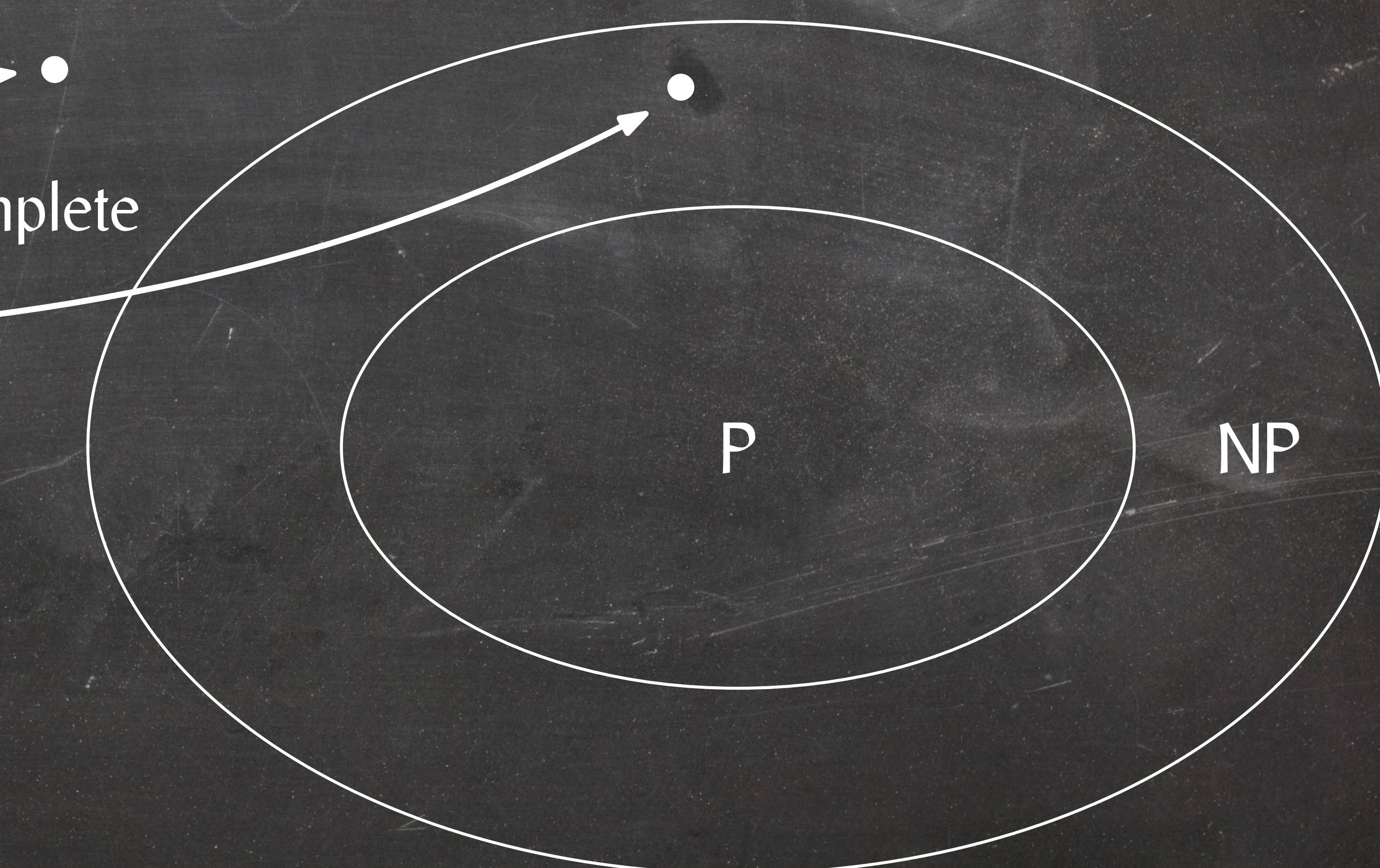
- $L \in NP$ and
- L is NP-hard.

Intuitively, NP-complete languages are the hardest languages in NP.

Assume $P \neq NP$.

Maybe NP-hard but never NP-complete

NP-complete if NP-hard



NP-Hardness and NP-Completeness

A language L is NP-hard if $L \in P$ implies that $P = NP$.

A language L is NP-complete if

- $L \in NP$ and
- L is NP-hard.

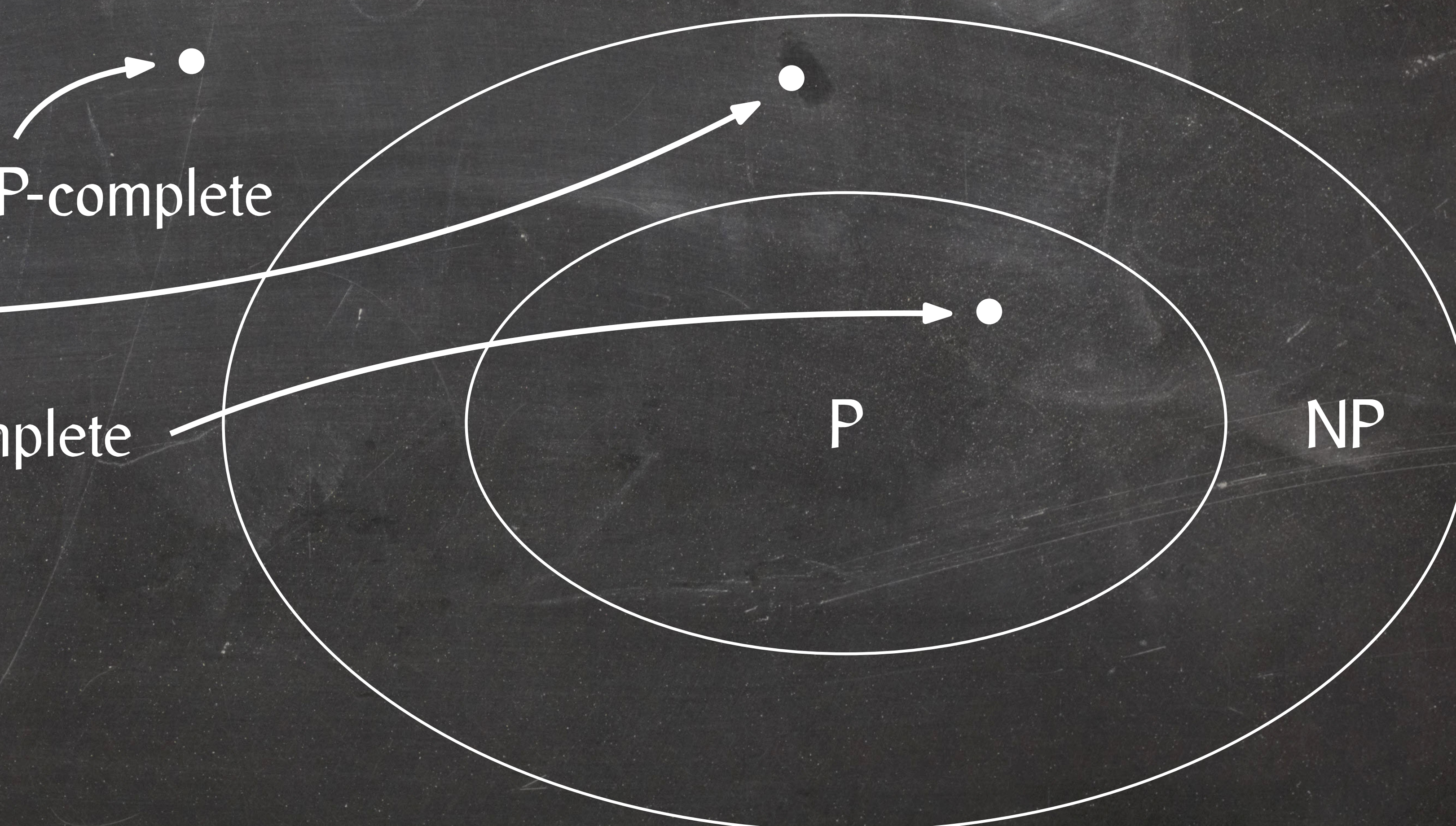
Intuitively, NP-complete languages are the hardest languages in NP.

Assume $P \neq NP$.

Maybe NP-hard but never NP-complete

NP-complete if NP-hard

Neither NP-hard nor NP-complete



Polynomial-Time Reductions

An algorithm R reduces a language $L_1 \subseteq \Sigma^*$ to a language $L_2 \subseteq \Sigma^*$ if, for all $x \in \Sigma^*$,

$$x \in L_1 \Leftrightarrow R(x) \in L_2.$$

$$x \longrightarrow \boxed{\quad R \quad} \longrightarrow R(x)$$

R is a polynomial-time reduction if its running time is polynomial in $|x|$.

Proving NP-Hardness Using Polynomial-Time Reductions

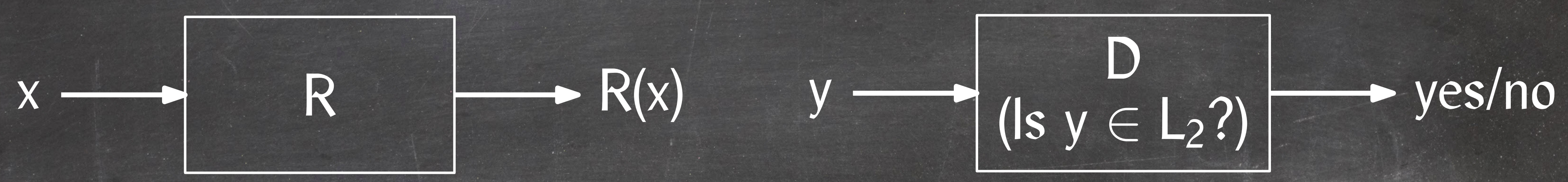
Lemma: If there exists a polynomial-time reduction R from a language L_1 to a language $L_2 \in P$, then $L_1 \in P$.

Proving NP-Hardness Using Polynomial-Time Reductions

Lemma: If there exists a polynomial-time reduction R from a language L_1 to a language $L_2 \in P$, then $L_1 \in P$.

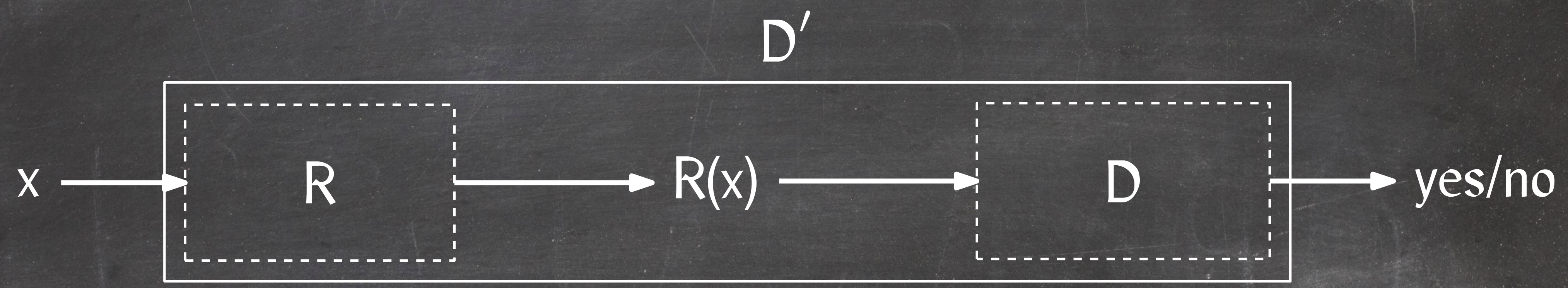
Proving NP-Hardness Using Polynomial-Time Reductions

Lemma: If there exists a polynomial-time reduction R from a language L_1 to a language $L_2 \in P$, then $L_1 \in P$.



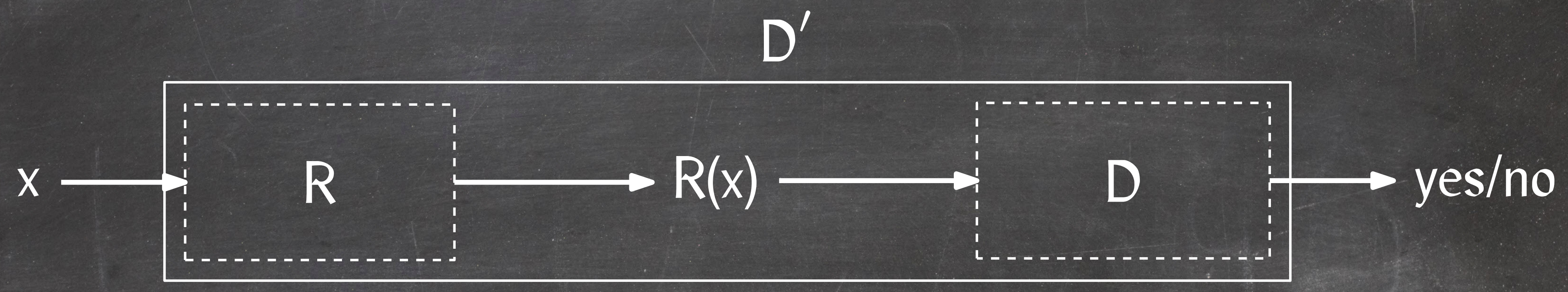
Proving NP-Hardness Using Polynomial-Time Reductions

Lemma: If there exists a polynomial-time reduction R from a language L_1 to a language $L_2 \in P$, then $L_1 \in P$.



Proving NP-Hardness Using Polynomial-Time Reductions

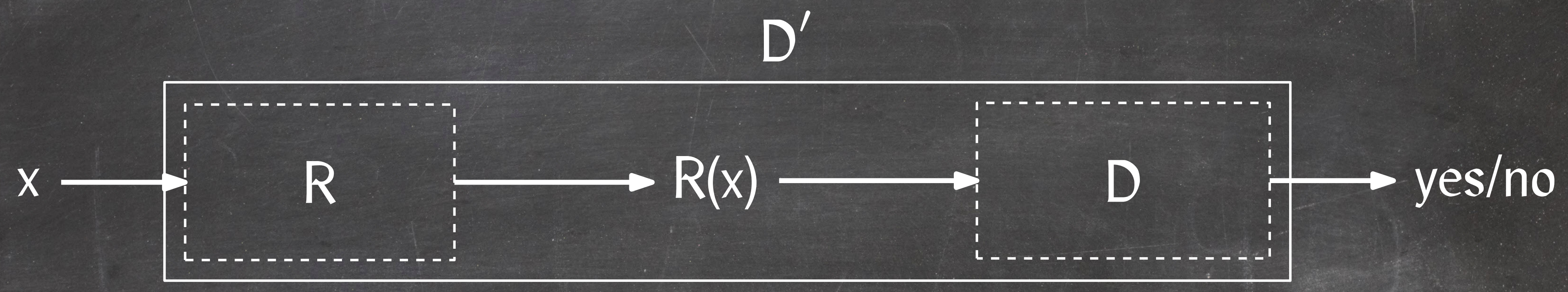
Lemma: If there exists a polynomial-time reduction R from a language L_1 to a language $L_2 \in P$, then $L_1 \in P$.



$x \in L_1 \Leftrightarrow R(x) \in L_2$

Proving NP-Hardness Using Polynomial-Time Reductions

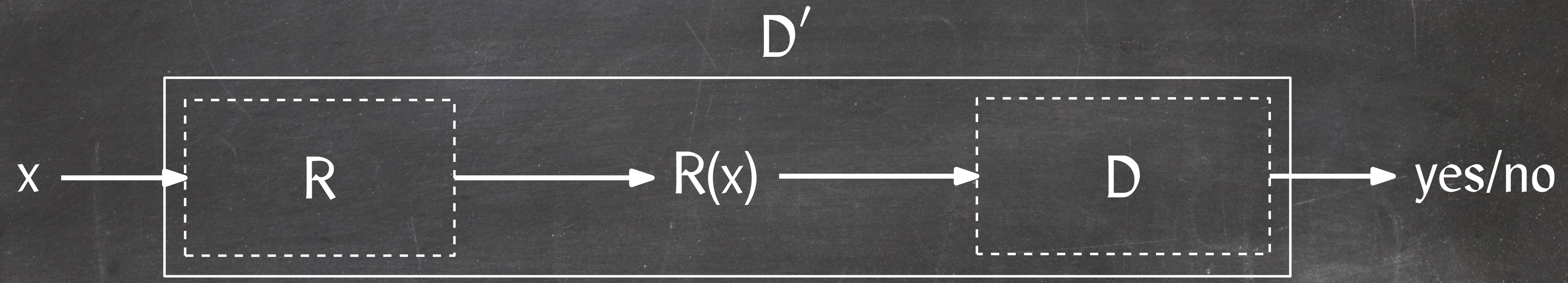
Lemma: If there exists a polynomial-time reduction R from a language L_1 to a language $L_2 \in P$, then $L_1 \in P$.



$x \in L_1 \Leftrightarrow R(x) \in L_2 \qquad R(x) \in L_2 \Leftrightarrow D(R(x)) = \text{yes}$

Proving NP-Hardness Using Polynomial-Time Reductions

Lemma: If there exists a polynomial-time reduction R from a language L_1 to a language $L_2 \in P$, then $L_1 \in P$.

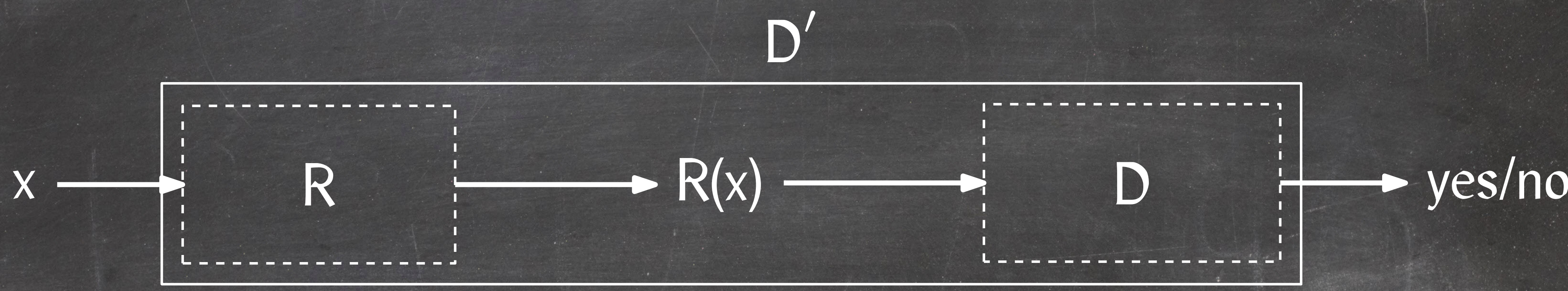


$$x \in L_1 \Leftrightarrow R(x) \in L_2 \qquad R(x) \in L_2 \Leftrightarrow D(R(x)) = \text{yes}$$

$$x \in L_1 \Leftrightarrow D'(x) = \text{yes}$$

Proving NP-Hardness Using Polynomial-Time Reductions

Lemma: If there exists a polynomial-time reduction R from a language L_1 to a language $L_2 \in P$, then $L_1 \in P$.



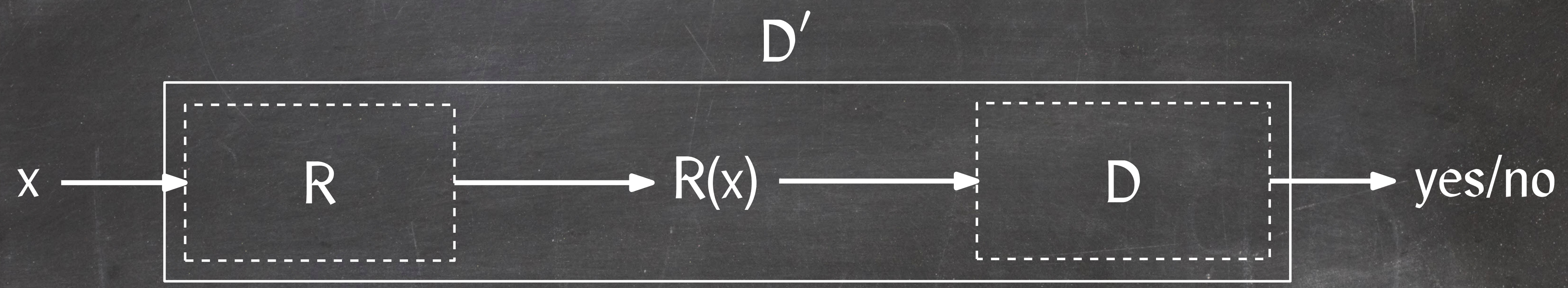
$x \in L_1 \Leftrightarrow R(x) \in L_2$ $\qquad$ $R(x) \in L_2 \Leftrightarrow D(R(x)) = \text{yes}$

$$x \in L_1 \Leftrightarrow D'(x) = \text{yes}$$

$T_R(|x|) \le c|x|^a$, for some a, c.

Proving NP-Hardness Using Polynomial-Time Reductions

Lemma: If there exists a polynomial-time reduction R from a language L_1 to a language $L_2 \in P$, then $L_1 \in P$.



$$x \in L_1 \Leftrightarrow R(x) \in L_2 \qquad R(x) \in L_2 \Leftrightarrow D(R(x)) = \text{yes}$$

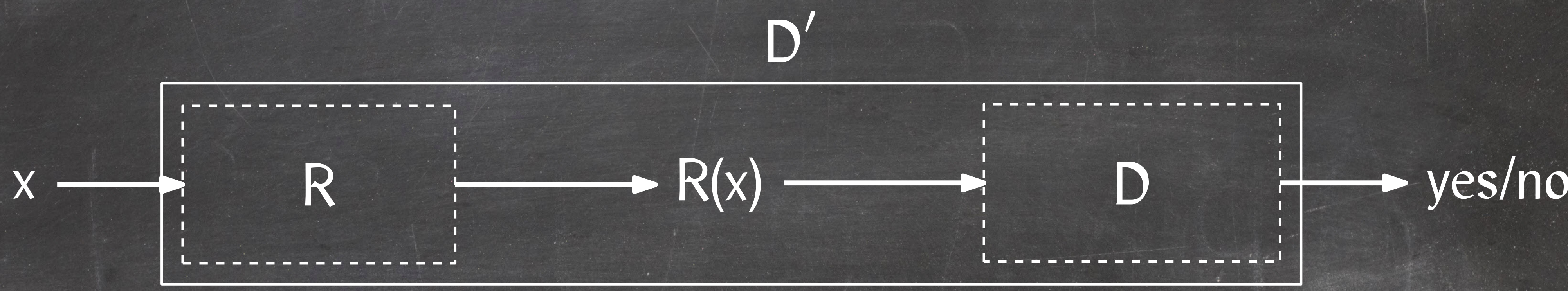
$$x \in L_1 \Leftrightarrow D'(x) = \text{yes}$$

$T_R(|x|) \leq c|x|^a$, for some a, c.

$\Rightarrow \ |R(x)| \leq c|x|^a$, for some a, c.

Proving NP-Hardness Using Polynomial-Time Reductions

Lemma: If there exists a polynomial-time reduction R from a language L_1 to a language $L_2 \in P$, then $L_1 \in P$.



$$x \in L_1 \Leftrightarrow R(x) \in L_2 \qquad R(x) \in L_2 \Leftrightarrow D(R(x)) = \text{yes}$$

$$x \in L_1 \Leftrightarrow D'(x) = \text{yes}$$

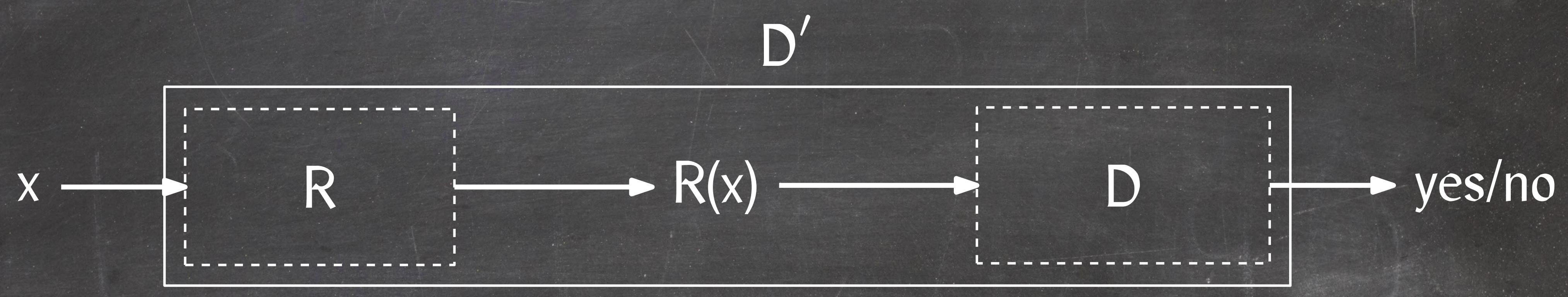
$T_R(|x|) \le c|x|^a$, for some a, c.

$\Rightarrow\ |R(x)| \le c|x|^a$, for some a, c.

$\Rightarrow\ T_D(|R(x)|) \le c'|R(x)|^{a'} \le c'(c|x|^a)^{a'}$, for some a', c'.

Proving NP-Hardness Using Polynomial-Time Reductions

Lemma: If there exists a polynomial-time reduction R from a language L_1 to a language $L_2 \in P$, then $L_1 \in P$.



$$x \in L_1 \Leftrightarrow R(x) \in L_2 \qquad R(x) \in L_2 \Leftrightarrow D(R(x)) = \text{yes}$$

$$x \in L_1 \Leftrightarrow D'(x) = \text{yes}$$

$T_R(|x|) \leq c|x|^a$, for some a, c.

$\Rightarrow \ |R(x)| \leq c|x|^a$, for some a, c.

$\Rightarrow \ T_D(|R(x)|) \leq c'|R(x)|^{a'} \leq c'(c|x|^a)^{a'}$, for some a', c'.

$\Rightarrow \ T_{D'}(|x|) = T_R(|x|) + T_D(|R(x)|) \leq c|x|^a + c'(c|x|^a)^{a'} \in O(|x|^{aa'})$.

Proving NP-Hardness Using Polynomial-Time Reductions

Corollary: If there exists a polynomial-time reduction from an NP-hard language L_1 to a language L_2, then L_2 is also NP-hard.

Proving NP-Hardness Using Polynomial-Time Reductions

Corollary: If there exists a polynomial-time reduction from an NP-hard language L_1 to a language L_2, then L_2 is also NP-hard.

$$L_2 \in P \xrightarrow{\text{Polynomial-time reduction}} L_1 \in P$$

Proving NP-Hardness Using Polynomial-Time Reductions

Corollary: If there exists a polynomial-time reduction from an NP-hard language L_1 to a language L_2, then L_2 is also NP-hard.

$$L_2 \in P \xrightarrow{\text{Polynomial-time reduction}} L_1 \in P \xrightarrow{\text{NP-hardness of } L_1} P = NP$$

Proving NP-Hardness Using Polynomial-Time Reductions

Corollary: If there exists a polynomial-time reduction from an NP-hard language L_1 to a language L_2, then L_2 is also NP-hard.

$$\begin{tikzcd}
L_2 \in P \arrow[r, "\text{Polynomial-time reduction}"] \arrow[rr, bend right, "L_2 \text{ is NP-hard.}"'] & L_1 \in P \arrow[r, "\text{NP-hardness of } L_1"] & P = NP
\end{tikzcd}$$

Where Do We Get Our First NP-Hard Problem From?

To prove that a language L is NP-hard, we need an NP-hard language L' that we can reduce to L in polynomial time.

How do we prove a language L is NP-hard when we haven't shown any other language to be NP-hard yet?

Enter Satisfiability, the mother of all NP-hard problems …

Satisfiability (SAT)

A Boolean formula:

$$F = (x_1 \vee (x_2 \wedge \bar{x}_3)) \wedge (\bar{x}_1 \vee x_4)$$

Satisfiability (SAT)

A Boolean formula:

$$F = (x_1 \vee (x_2 \wedge \bar{x}_3)) \wedge (\bar{x}_1 \vee x_4)$$

- x_1, x_2, x_3, x_4 are Boolean variables, which can be true or false.

Satisfiability (SAT)

A Boolean formula:

$$F = (x_1 \vee (x_2 \wedge \bar{x}_3)) \wedge (\bar{x}_1 \vee x_4)$$

- x_1, x_2, x_3, x_4 are Boolean variables, which can be true or false.
- x_1, $\bar{x}_1$, x_2, $\bar{x}_3$, x_4 are literals (a Boolean variable or its negation).

Satisfiability (SAT)

A Boolean formula:

$$F = (x_1 \vee (x_2 \wedge \bar{x}_3)) \wedge (\bar{x}_1 \vee x_4)$$

- x_1, x_2, x_3, x_4 are Boolean variables, which can be true or false.
- x_1, $\bar{x}_1$, x_2, $\bar{x}_3$, x_4 are literals (a Boolean variable or its negation).
- A truth assignment assigns a value true or false to each variable in F.

Satisfiability (SAT)

A Boolean formula:

$$F = (x_1 \vee (x_2 \wedge \bar{x}_3)) \wedge (\bar{x}_1 \vee x_4)$$

- x_1, x_2, x_3, x_4 are Boolean variables, which can be true or false.
- x_1, $\bar{x}_1$, x_2, $\bar{x}_3$, x_4 are literals (a Boolean variable or its negation).
- A truth assignment assigns a value true or false to each variable in F.
- A truth assigment satisfies F if it makes F true. Example:
  - $x_1 = x_2 = x_3 = x_4 =$ true satisfies F.
  - $x_1 = x_2 = x_3 = x_4 =$ false does not.

Satisfiability (SAT)

A Boolean formula:

$$F = (x_1 \vee (x_2 \wedge \bar{x}_3)) \wedge (\bar{x}_1 \vee x_4)$$

- x_1, x_2, x_3, x_4 are Boolean variables, which can be true or false.
- x_1, $\bar{x}_1$, x_2, $\bar{x}_3$, x_4 are literals (a Boolean variable or its negation).
- A truth assignment assigns a value true or false to each variable in F.
- A truth assigment satisfies F if it makes F true. Example:
  - $x_1 = x_2 = x_3 = x_4 =$ true satisfies F.
  - $x_1 = x_2 = x_3 = x_4 =$ false does not.
- F is satisfiable if it has a satisfying truth assignment.

Satisfiability (SAT)

A Boolean formula:

$$F = (x_1 \vee (x_2 \wedge \bar{x}_3)) \wedge (\bar{x}_1 \vee x_4)$$

- x_1, x_2, x_3, x_4 are Boolean variables, which can be true or false.
- x_1, $\bar{x}_1$, x_2, $\bar{x}_3$, x_4 are literals (a Boolean variable or its negation).
- A truth assignment assigns a value true or false to each variable in F.
- A truth assigment satisfies F if it makes F true. Example:
  - $x_1 = x_2 = x_3 = x_4 =$ true satisfies F.
  - $x_1 = x_2 = x_3 = x_4 =$ false does not.
- F is satisfiable if it has a satisfying truth assignment.

The satisfiability problem (SAT): Given a Boolean formula F, decide whether F is satisfiable.

3-SAT

A formula is in conjunctive normal form (CNF) if it is a conjunction of disjuctions.

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

The disjunctions are also called clauses.

A formula is in 3-CNF if each of its clauses consists of three literals.

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$$

3-SAT: Decide whether a given formula in 3-CNF is satisfiable.

Things We Don't Have Time To Prove

Any polynomial-time verification algorithm for a language L can be turned into a polynomial-time reduction from L to SAT.

Things We Don't Have Time To Prove

Any polynomial-time verification algorithm for a language L can be turned into a polynomial-time reduction from L to SAT.

Thus, SAT $\in$ P $\Rightarrow$ L $\in$ P for all L $\in$ NP, that is, P = NP.

Things We Don't Have Time To Prove

Any polynomial-time verification algorithm for a language L can be turned into a polynomial-time reduction from L to SAT.

Thus, SAT $\in$ P $\Rightarrow$ L $\in$ P for all L $\in$ NP, that is, P = NP.

In other words, SAT is NP-hard.

Things We Don't Have Time To Prove

Any polynomial-time verification algorithm for a language L can be turned into a polynomial-time reduction from L to SAT.

Thus, SAT $\in$ P $\Rightarrow$ L $\in$ P for all L $\in$ NP, that is, P = NP.

In other words, SAT is NP-hard.

Any Boolean formula F can be turned, in polynomial time, into a Boolean formula F' in 3-CNF, of size $|F'| \in O(|F|)$, and such that F is satisfiable if and only if F' is.

Things We Don't Have Time To Prove

Any polynomial-time verification algorithm for a language L can be turned into a polynomial-time reduction from L to SAT.

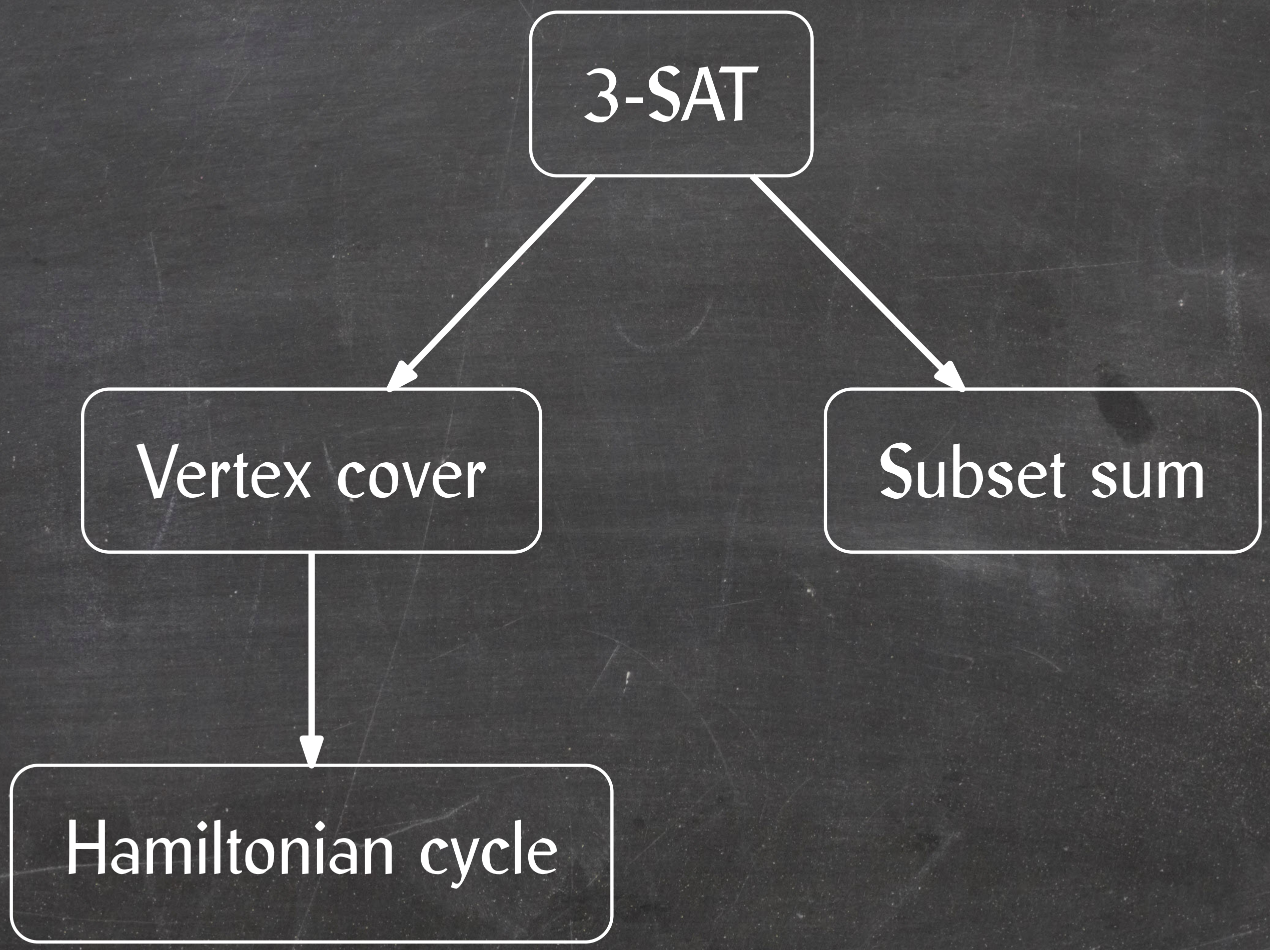
Thus, SAT $\in$ P $\Rightarrow$ L $\in$ P for all L $\in$ NP, that is, P = NP.

In other words, SAT is NP-hard.

Any Boolean formula F can be turned, in polynomial time, into a Boolean formula F' in 3-CNF, of size $|F'| \in O(|F|)$, and such that F is satisfiable if and only if F' is.

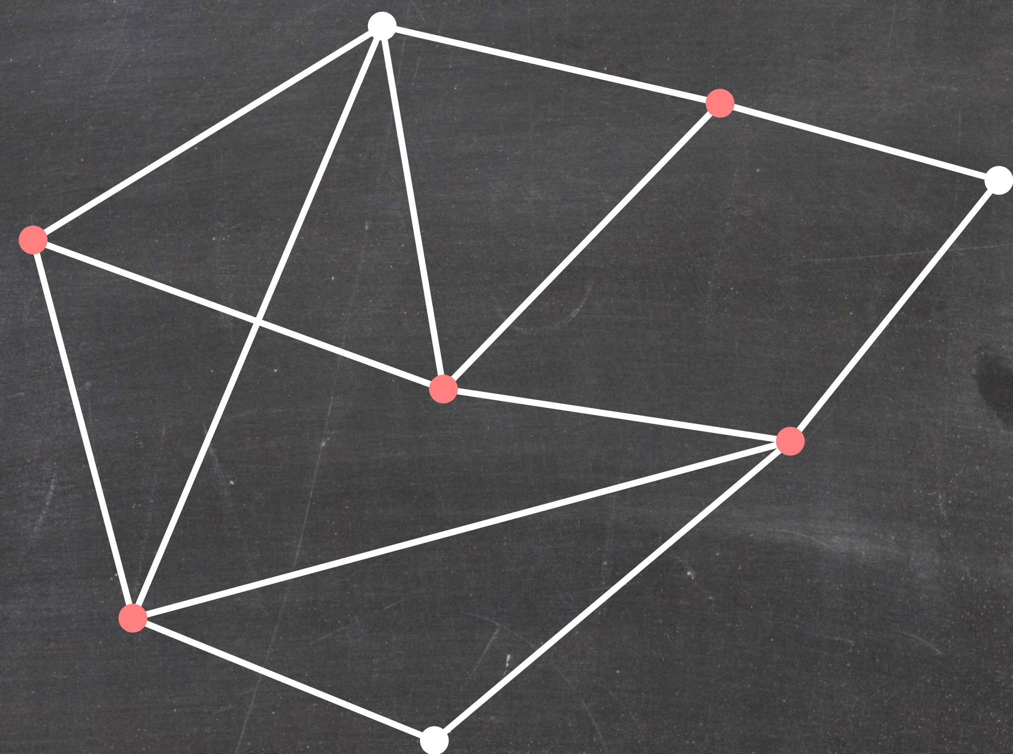
Thus, 3-SAT is NP-hard.

Examples of Polynomial-Time Reductions



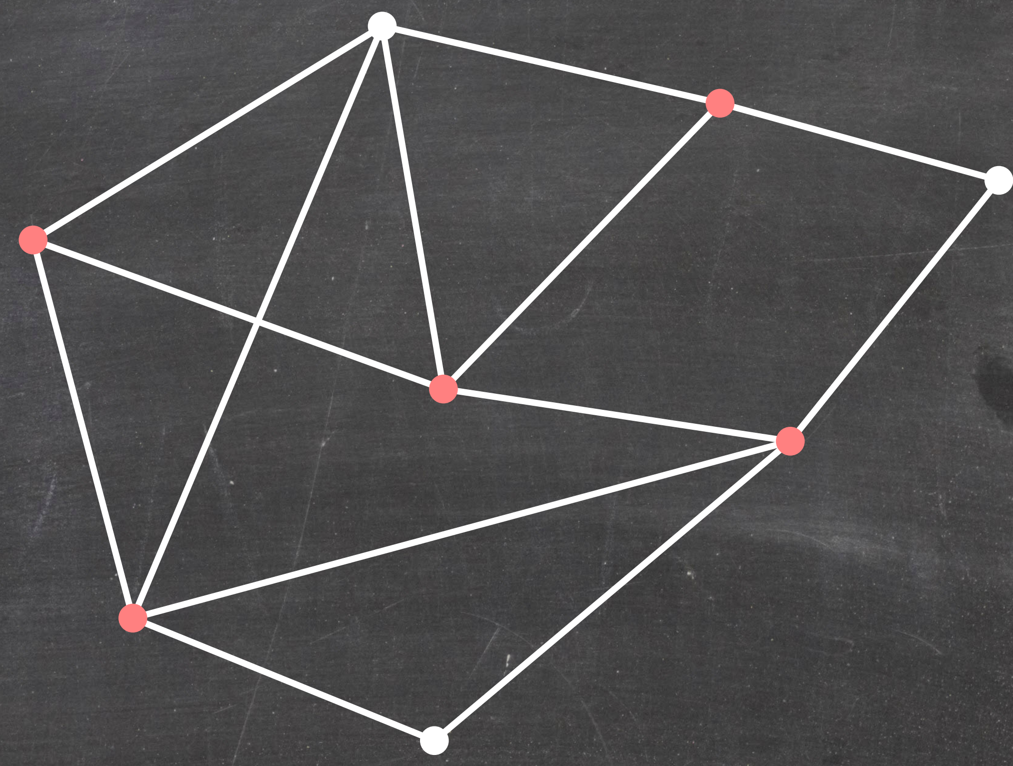
Vertex Cover

A vertex cover of a graph $G = (V, E)$ is a subset $S \subseteq V$ such that every edge in E has at least one endpoint in S.



Vertex Cover

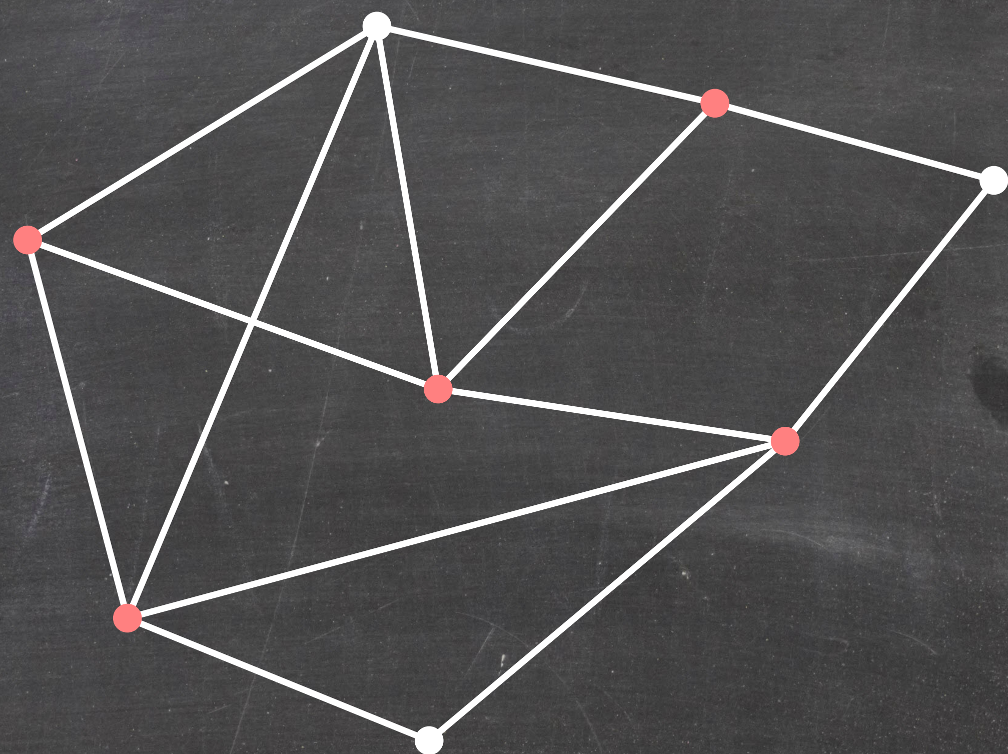
A vertex cover of a graph $G = (V, E)$ is a subset $S \subseteq V$ such that every edge in E has at least one endpoint in S.



Optimization problem: Given a graph G, find the smallest possible vertex cover of G.

Vertex Cover

A vertex cover of a graph $G = (V, E)$ is a subset $S \subseteq V$ such that every edge in E has at least one endpoint in S.



Optimization problem: Given a graph G, find the smallest possible vertex cover of G.

Decision problem: Given a graph G and an integer k, decide whether G has a vertex cover of size k.

Vertex Cover is NP-Hard

Reduction from 3-SAT:

- Given any formula F, we build a graph G_F that has a small vertex cover if and only if F is satisfiable.
- G_F will be built from subgraphs, called widgets, that guarantee certain properties of G_F.
- It will be obvious that this construction can be carried out in polynomial time.

Vertex Cover is NP-Hard

Reduction from 3-SAT:

- Given any formula F, we build a graph G_F that has a small vertex cover if and only if F is satisfiable.
- G_F will be built from subgraphs, called widgets, that guarantee certain properties of G_F.
- It will be obvious that this construction can be carried out in polynomial time.

Variable widget for variable x_i:

- Two vertices x_i and $\bar{x}_i$
- One edge $(x_i, \bar{x}_i)$

Vertex Cover is NP-Hard

Reduction from 3-SAT:

- Given any formula F, we build a graph G_F that has a small vertex cover if and only if F is satisfiable.
- G_F will be built from subgraphs, called widgets, that guarantee certain properties of G_F.
- It will be obvious that this construction can be carried out in polynomial time.

Variable widget for variable x_i:

- Two vertices x_i and $\bar{x}_i$
- One edge $(x_i, \bar{x}_i)$

Clause widget for clause C_j:

- Three literal vertices $\lambda_{j,1}$, $\lambda_{j,2}$, and $\lambda_{j,3}$
- Three edges $(\lambda_{j,1}, \lambda_{j,2})$, $(\lambda_{j,2}, \lambda_{j,3})$, and $(\lambda_{j,3}, \lambda_{j,1})$

Vertex Cover is NP-Hard

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

Vertex Cover is NP-Hard

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

G_F:

Vertex Cover is NP-Hard

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

G_F:

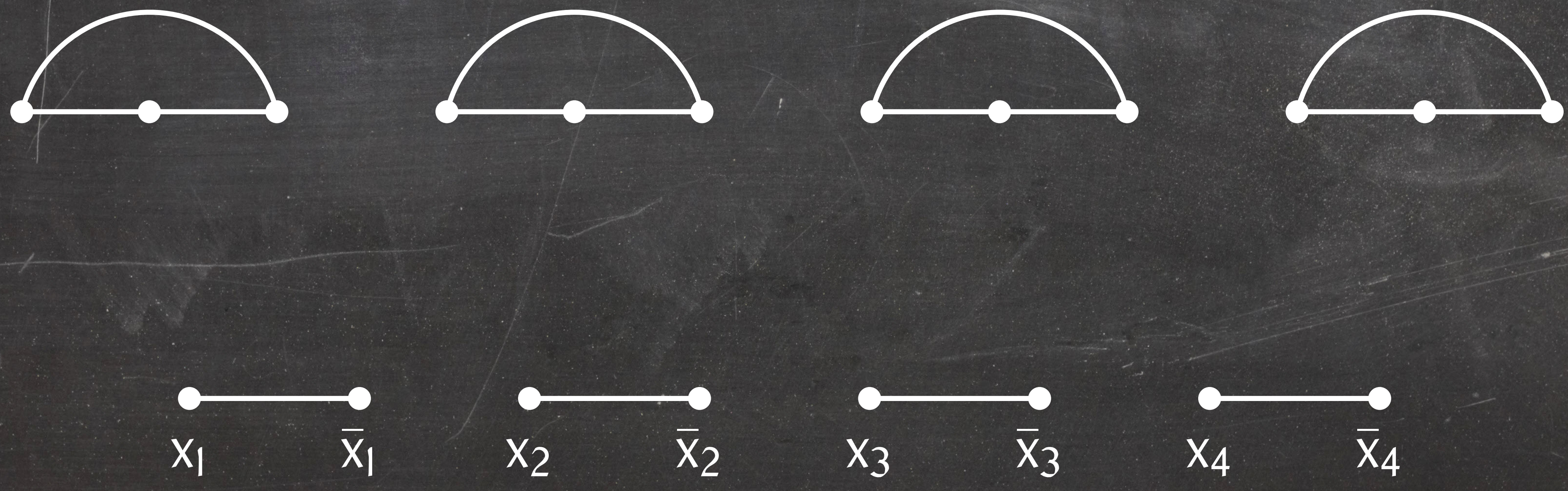
- One variable widget per variable

Vertex Cover is NP-Hard

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

G_F:

- One variable widget per variable
- One clause widget per clause

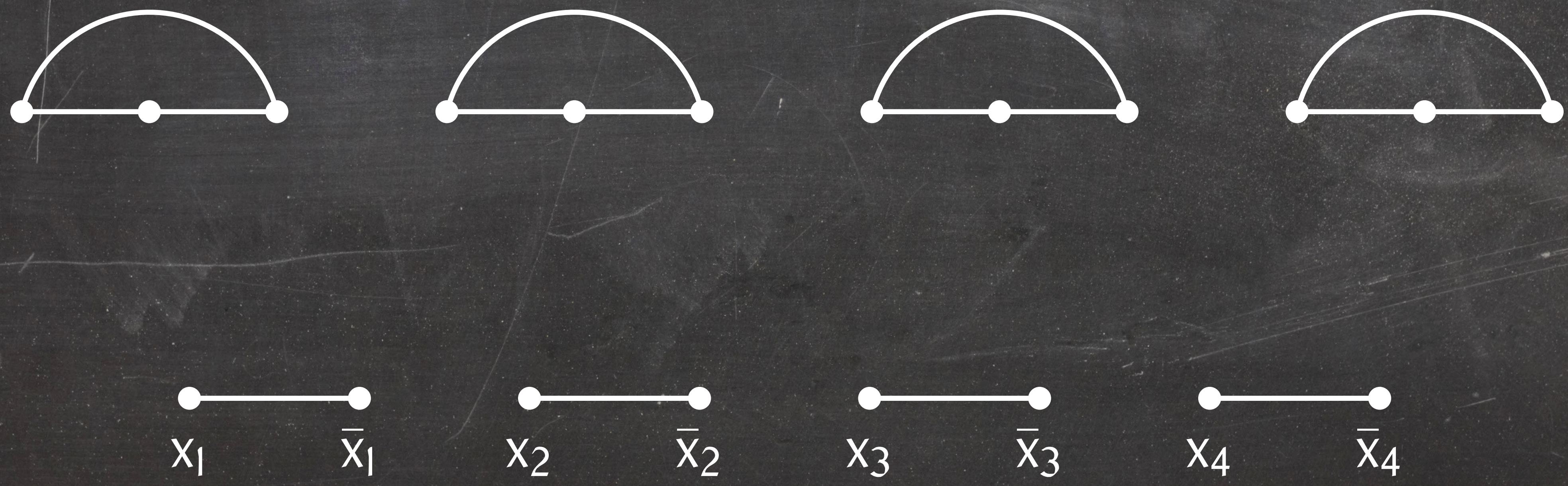


Vertex Cover is NP-Hard

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

G_F:

- One variable widget per variable
- One clause widget per clause
- Connect every literal node $\lambda_{i,j}$ to its corresponding node x_k or $\bar{x}_k$

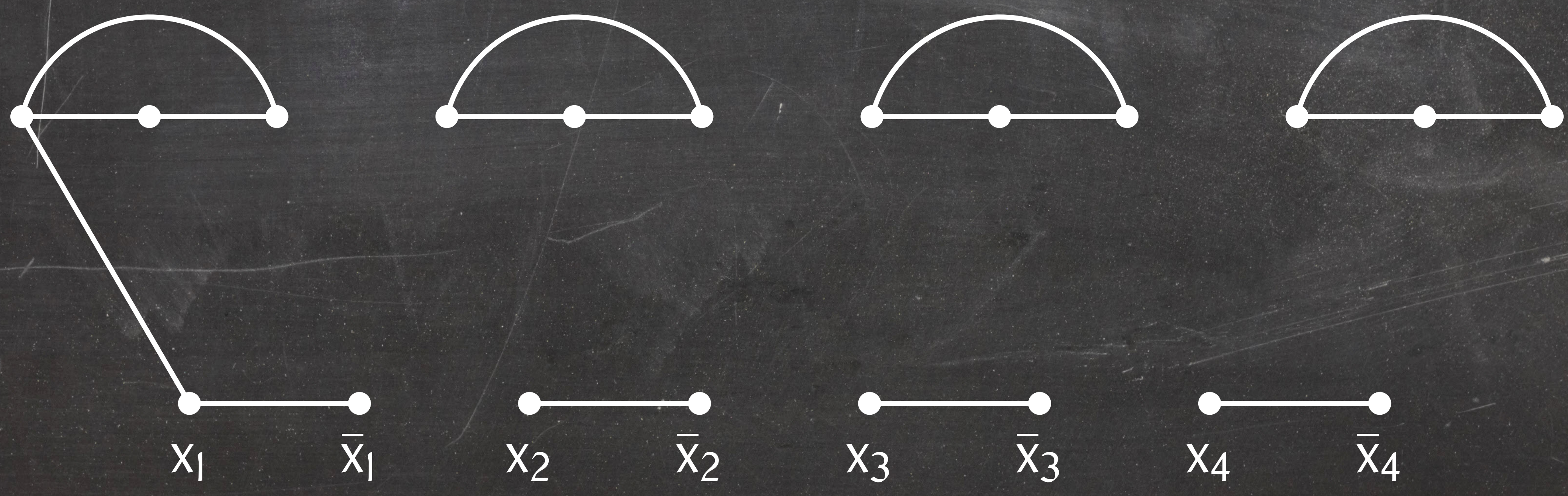


Vertex Cover is NP-Hard

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

G_F:

- One variable widget per variable
- One clause widget per clause
- Connect every literal node $\lambda_{i,j}$ to its corresponding node x_k or $\bar{x}_k$

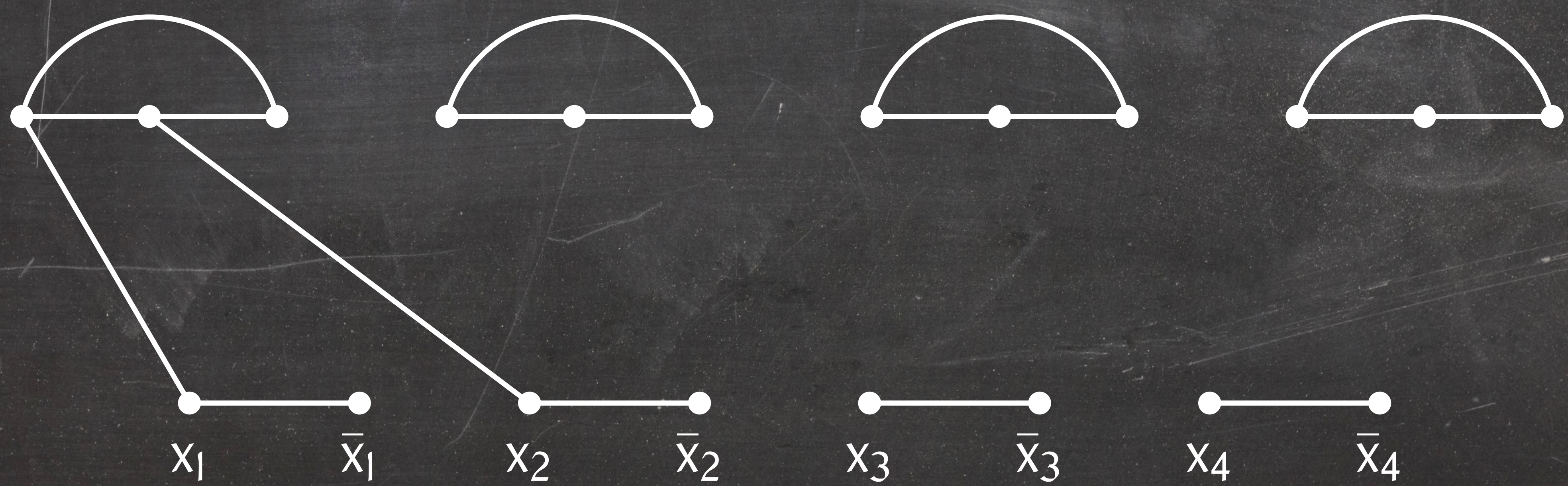


Vertex Cover is NP-Hard

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

G_F:

- One variable widget per variable
- One clause widget per clause
- Connect every literal node $\lambda_{i,j}$ to its corresponding node x_k or $\bar{x}_k$

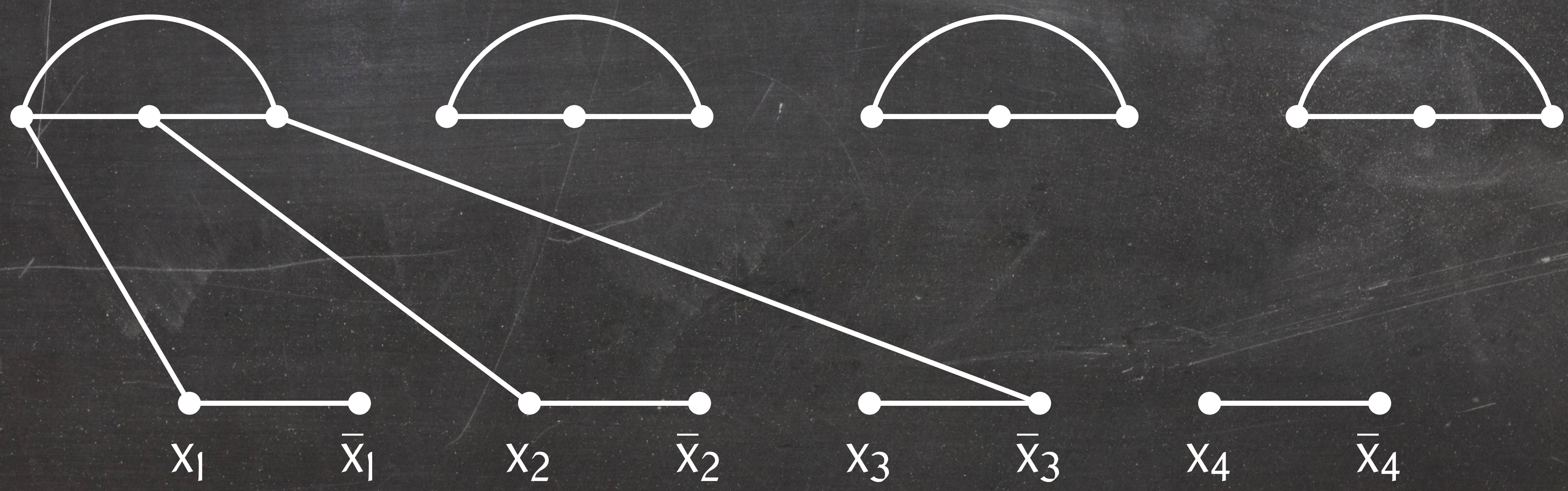


Vertex Cover is NP-Hard

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

G_F:

- One variable widget per variable
- One clause widget per clause
- Connect every literal node $\lambda_{i,j}$ to its corresponding node x_k or $\bar{x}_k$

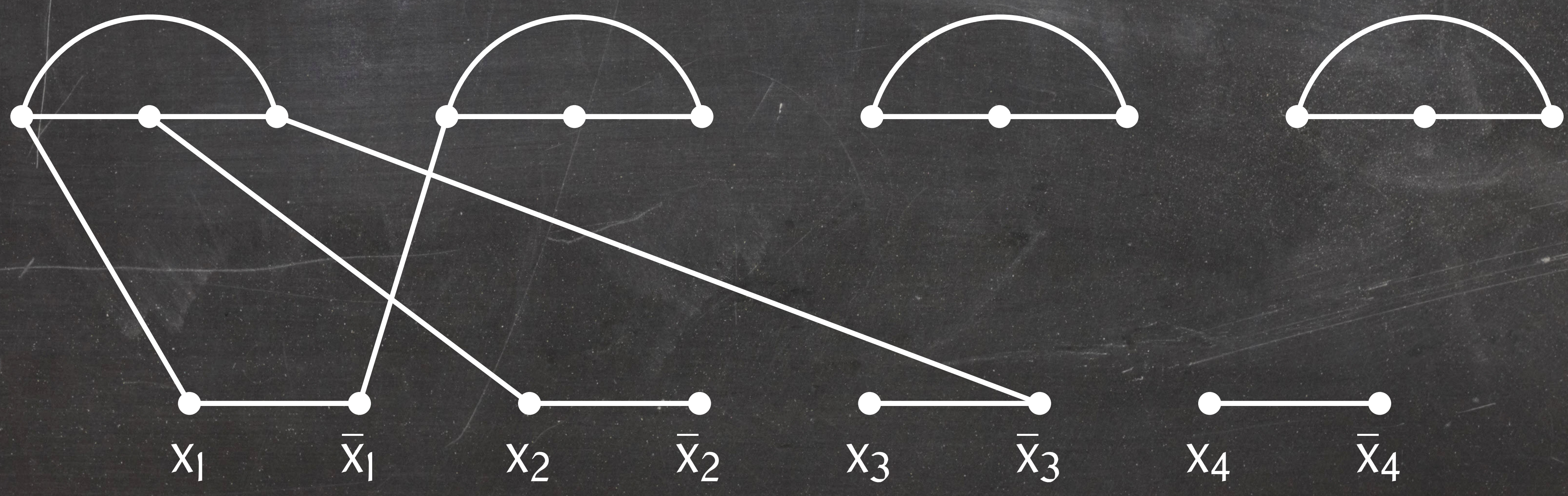


Vertex Cover is NP-Hard

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

G_F:

- One variable widget per variable
- One clause widget per clause
- Connect every literal node $\lambda_{i,j}$ to its corresponding node x_k or $\bar{x}_k$

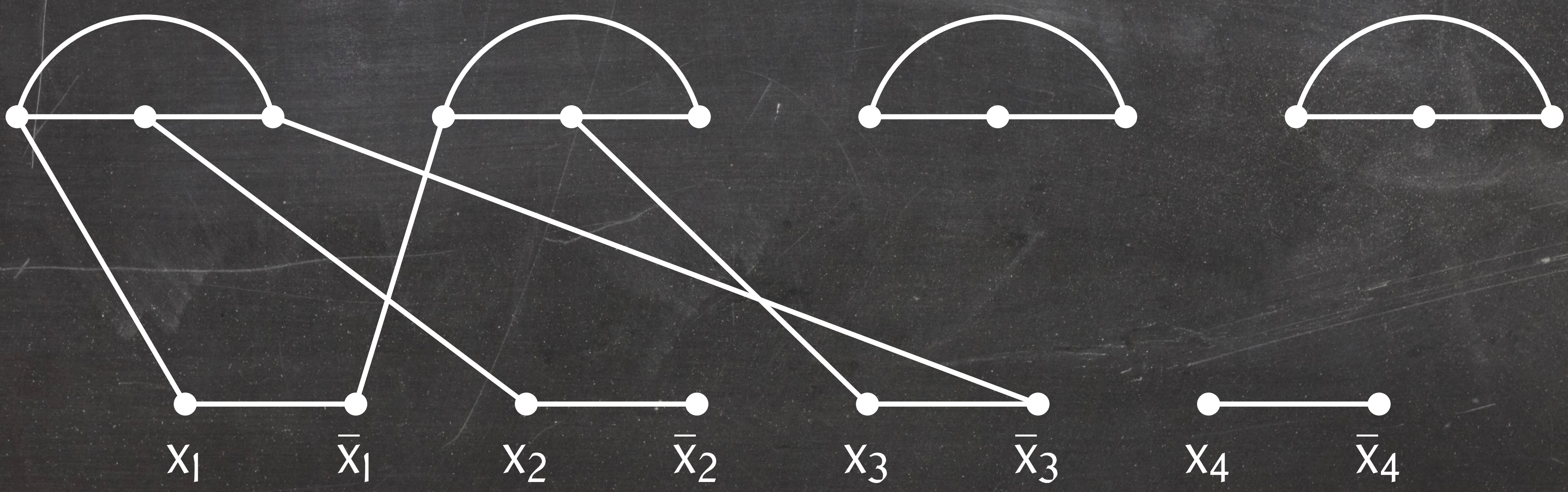


Vertex Cover is NP-Hard

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

G_F:

- One variable widget per variable
- One clause widget per clause
- Connect every literal node $\lambda_{i,j}$ to its corresponding node x_k or $\bar{x}_k$

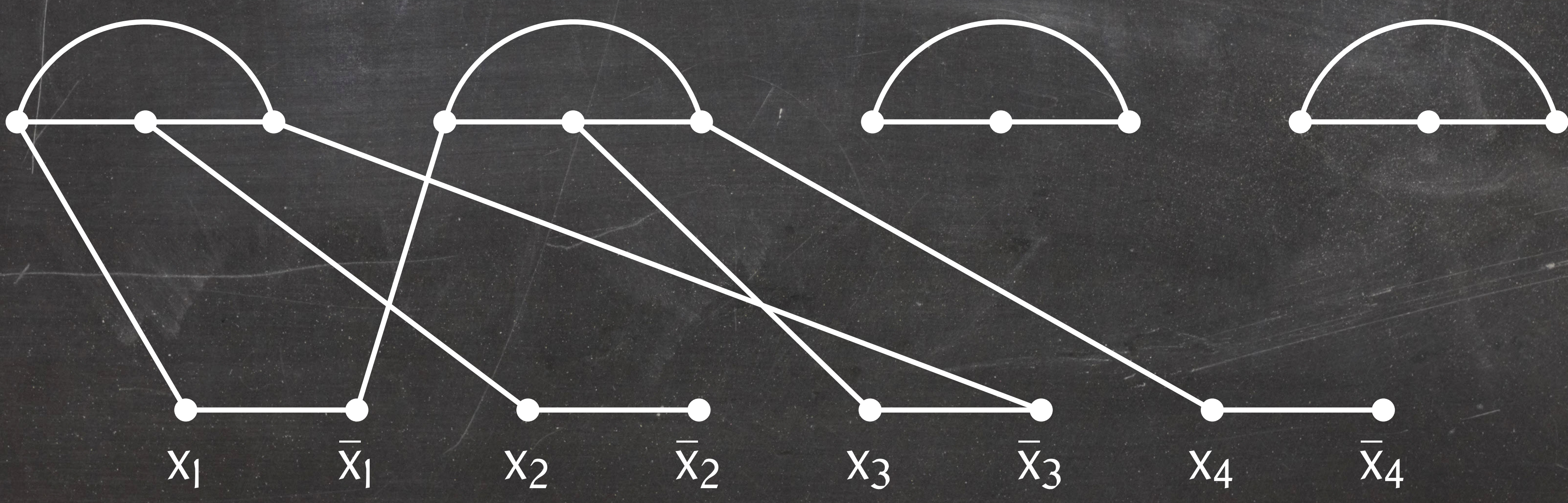


Vertex Cover is NP-Hard

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

G_F:

- One variable widget per variable
- One clause widget per clause
- Connect every literal node $\lambda_{i,j}$ to its corresponding node x_k or $\bar{x}_k$

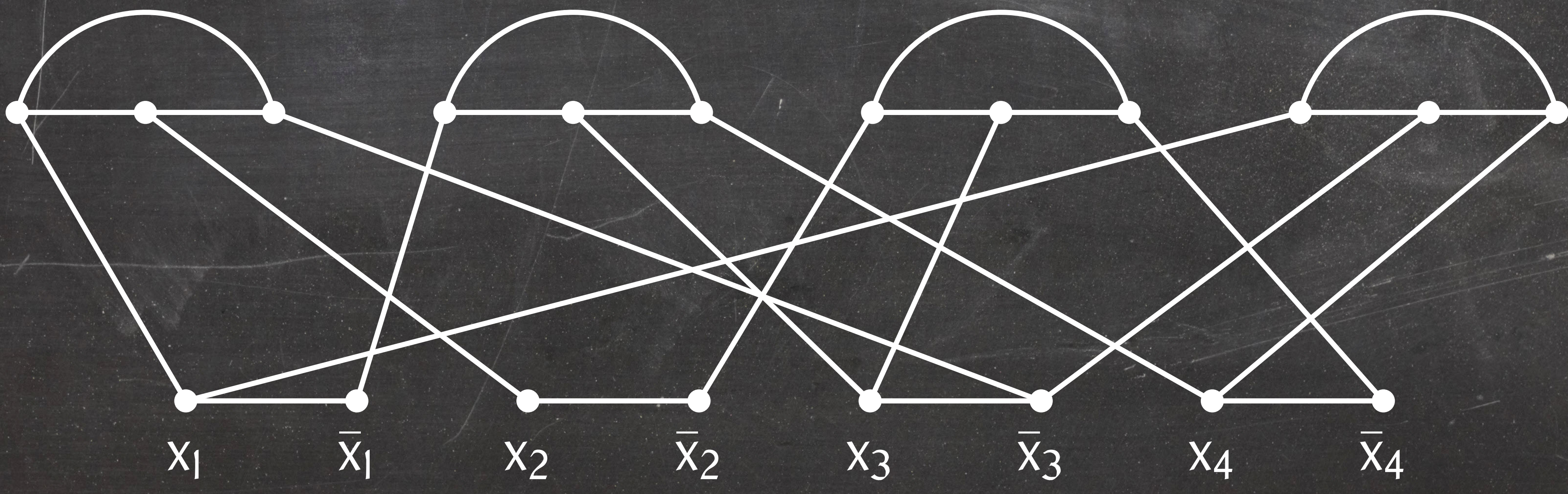


Vertex Cover is NP-Hard

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

G_F:

- One variable widget per variable
- One clause widget per clause
- Connect every literal node $\lambda_{i,j}$ to its corresponding node x_k or $\bar{x}_k$



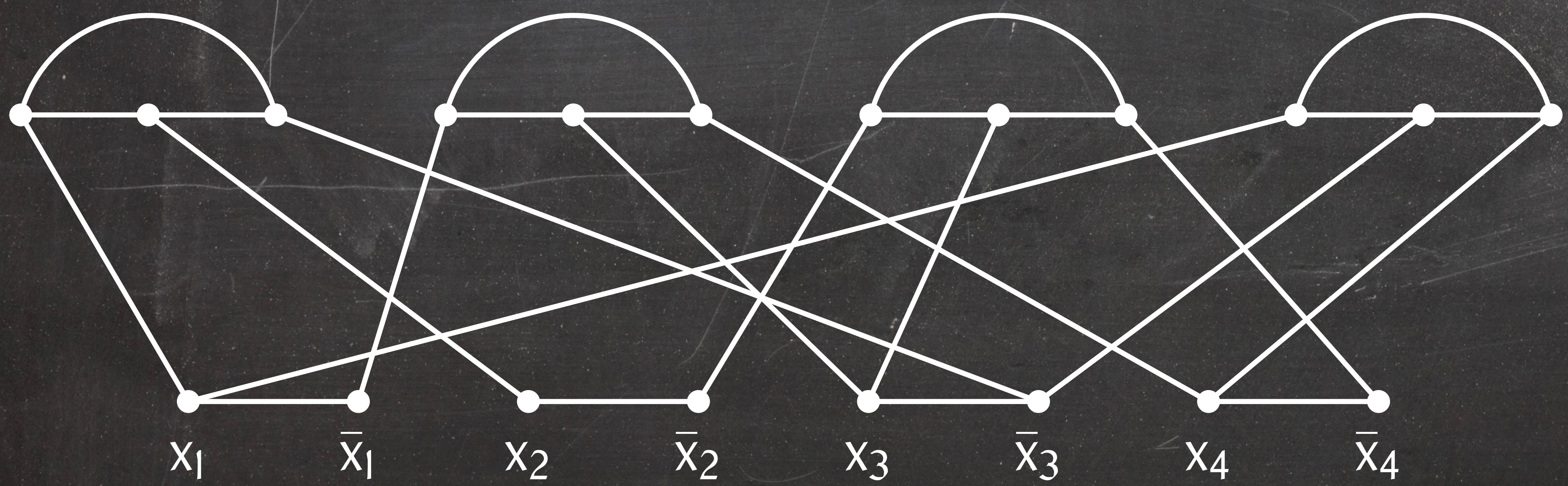
Vertex Cover is NP-Hard

n = number of variables   m = number of clauses

Vertex Cover is NP-Hard

n = number of variables   m = number of clauses

Observation: Any vertex cover of G_F of size $n + 2m$ contains one vertex per variable widget and two vertices per clause widget.



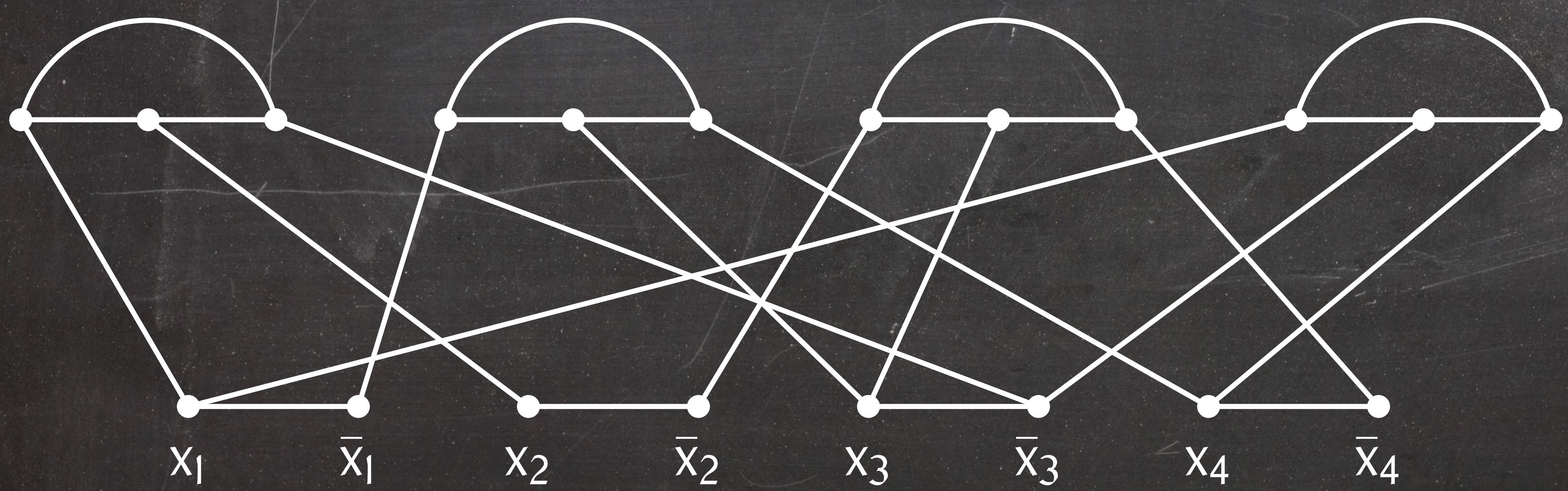
Vertex Cover is NP-Hard

n = number of variables $\qquad$ m = number of clauses

Observation: Any vertex cover of G_F of size $n + 2m$ contains one vertex per variable widget and two vertices per clause widget.

Lemma: F is satisfiable if and only if G_F has a vertex cover of size $n + 2m$.

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$



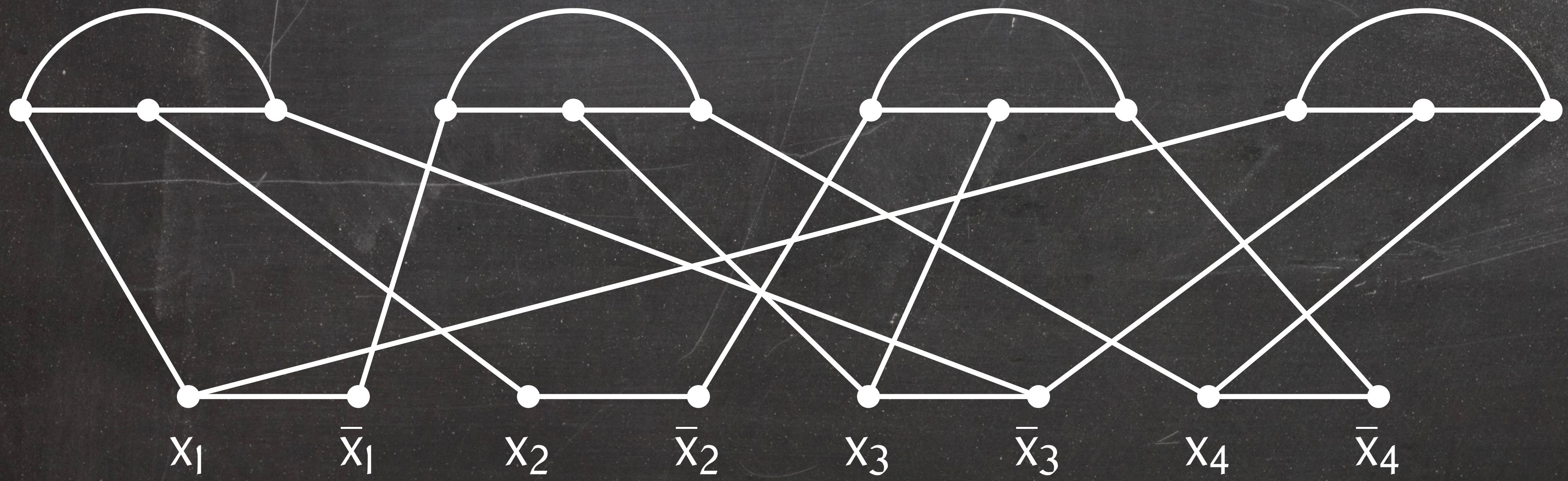
Vertex Cover is NP-Hard

n = number of variables m = number of clauses

Observation: Any vertex cover of G_F of size $n + 2m$ contains one vertex per variable widget and two vertices per clause widget.

Lemma: F is satisfiable if and only if G_F has a vertex cover of size $n + 2m$.

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$



Truth assignment

Vertex cover

Vertex Cover is NP-Hard

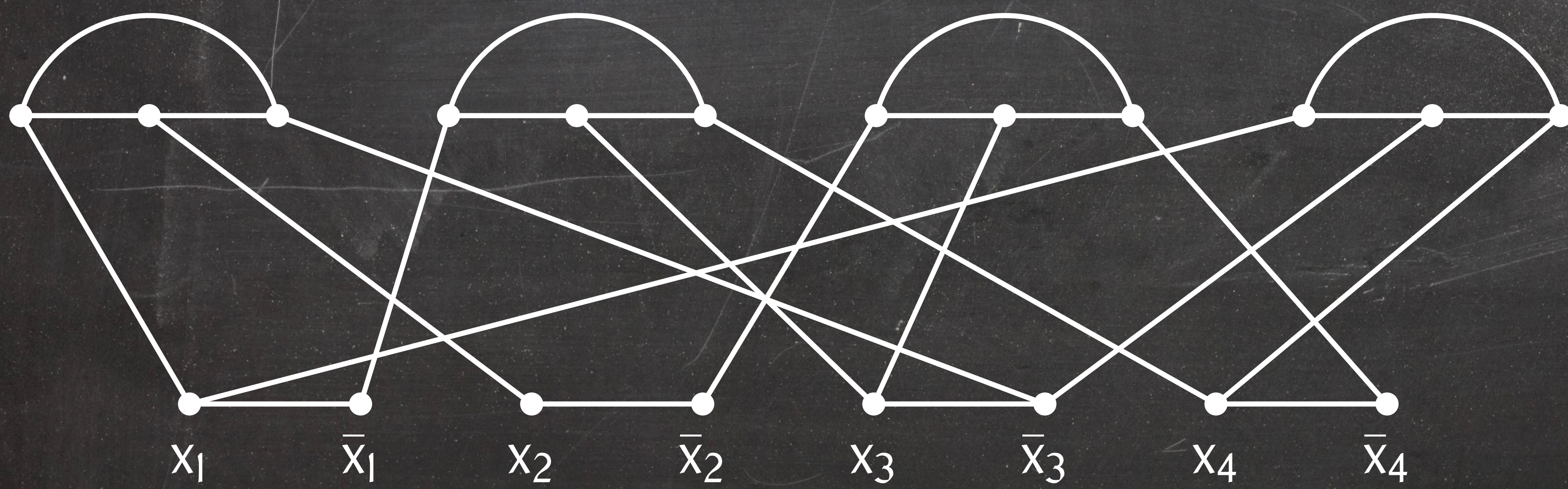
n = number of variables m = number of clauses

Observation: Any vertex cover of G_F of size $n + 2m$ contains one vertex per variable widget and two vertices per clause widget.

Lemma: F is satisfiable if and only if G_F has a vertex cover of size $n + 2m$.

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$$x_1 = x_2 = x_3 = x_4 = \text{true}$$



Truth assignment

Vertex cover

Vertex Cover is NP-Hard

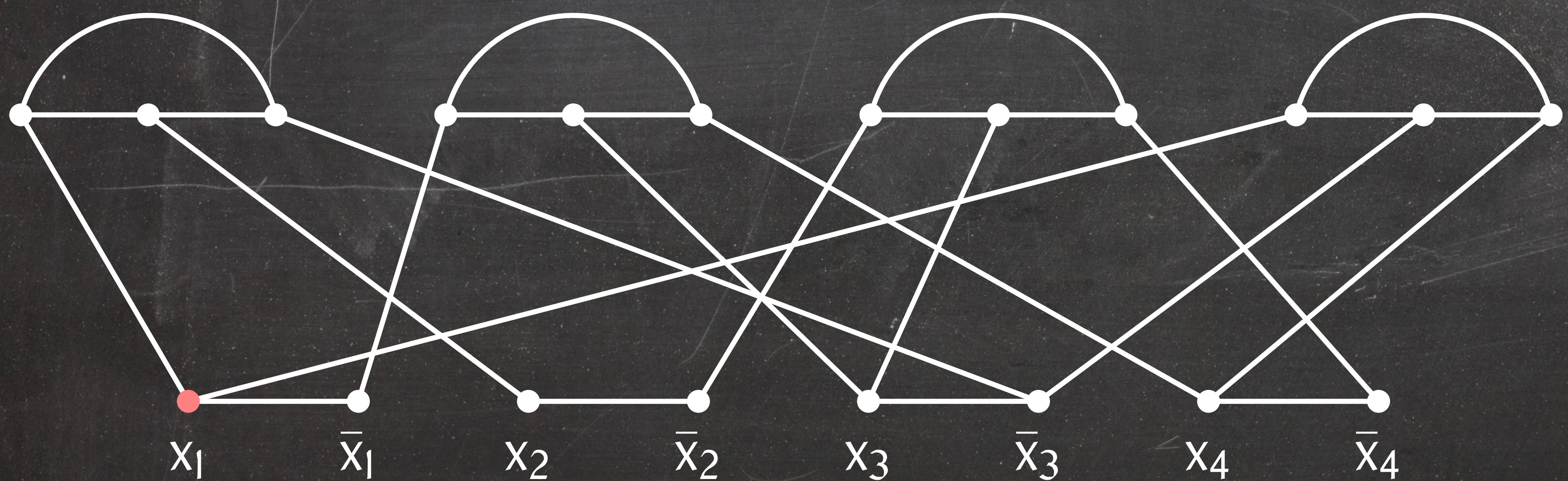
n = number of variables $\qquad$ m = number of clauses

Observation: Any vertex cover of G_F of size $n + 2m$ contains one vertex per variable widget and two vertices per clause widget.

Lemma: F is satisfiable if and only if G_F has a vertex cover of size $n + 2m$.

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$$x_1 = x_2 = x_3 = x_4 = \text{true}$$



Truth assignment

Vertex cover

Vertex Cover is NP-Hard

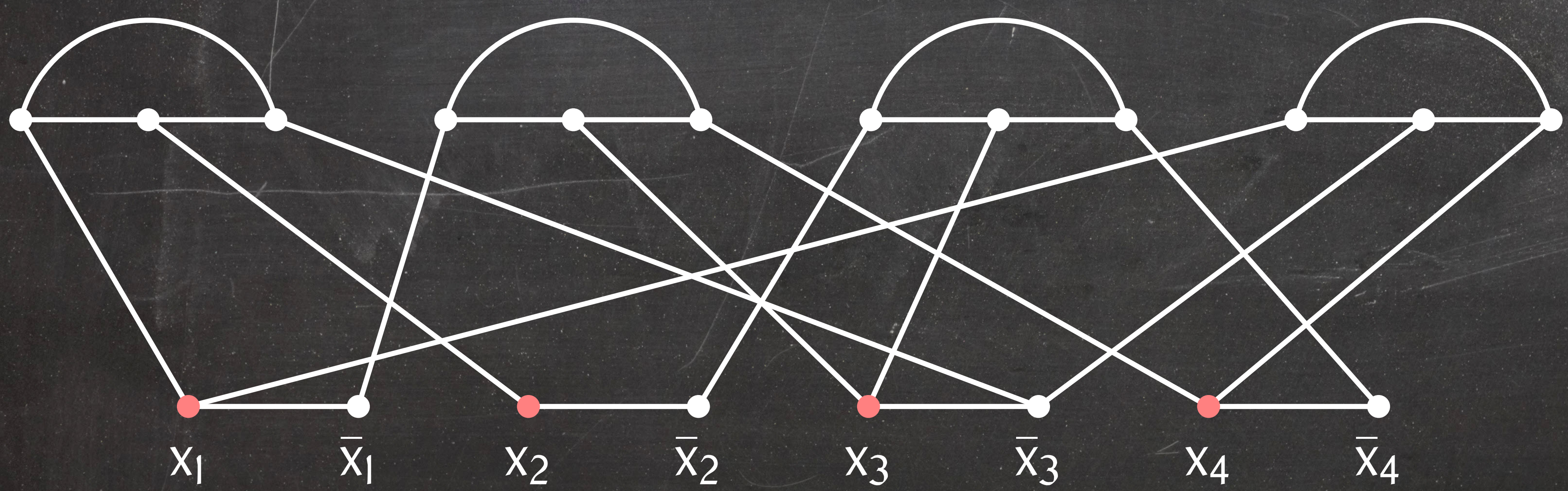
n = number of variables m = number of clauses

Observation: Any vertex cover of G_F of size $n + 2m$ contains one vertex per variable widget and two vertices per clause widget.

Lemma: F is satisfiable if and only if G_F has a vertex cover of size $n + 2m$.

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$$x_1 = x_2 = x_3 = x_4 = \text{true}$$



Truth assignment

↓

Vertex cover

Vertex Cover is NP-Hard

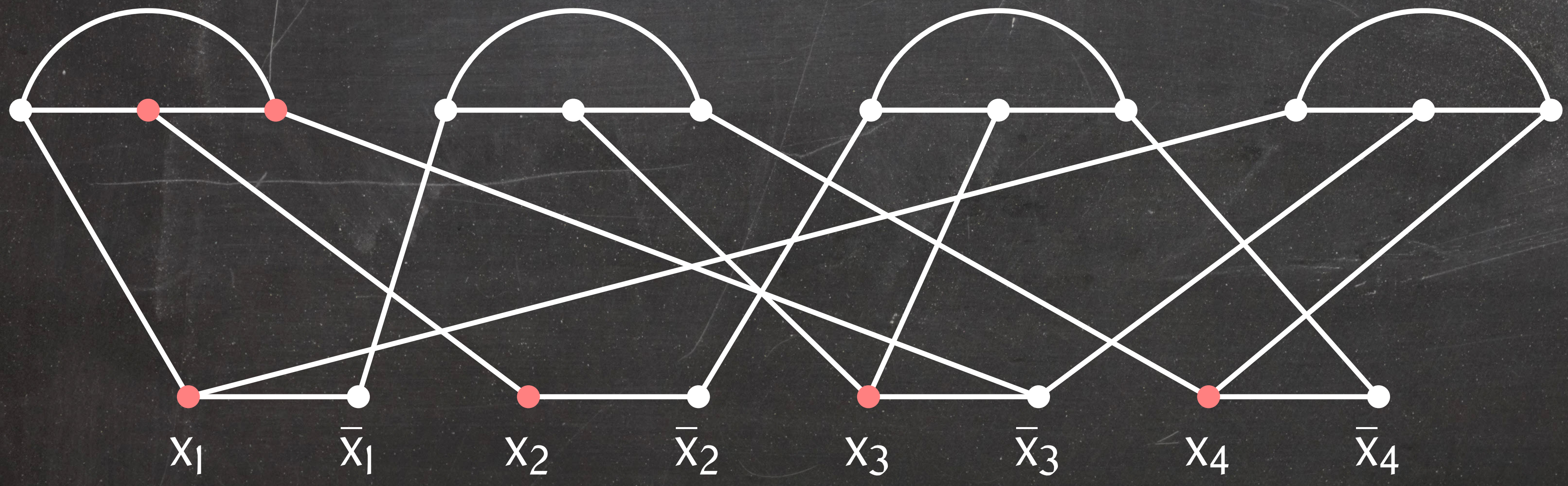
n = number of variables $\qquad$ m = number of clauses

Observation: Any vertex cover of G_F of size $n + 2m$ contains one vertex per variable widget and two vertices per clause widget.

Lemma: F is satisfiable if and only if G_F has a vertex cover of size $n + 2m$.

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$$x_1 = x_2 = x_3 = x_4 = \text{true}$$



Truth assignment

↓

Vertex cover

Vertex Cover is NP-Hard

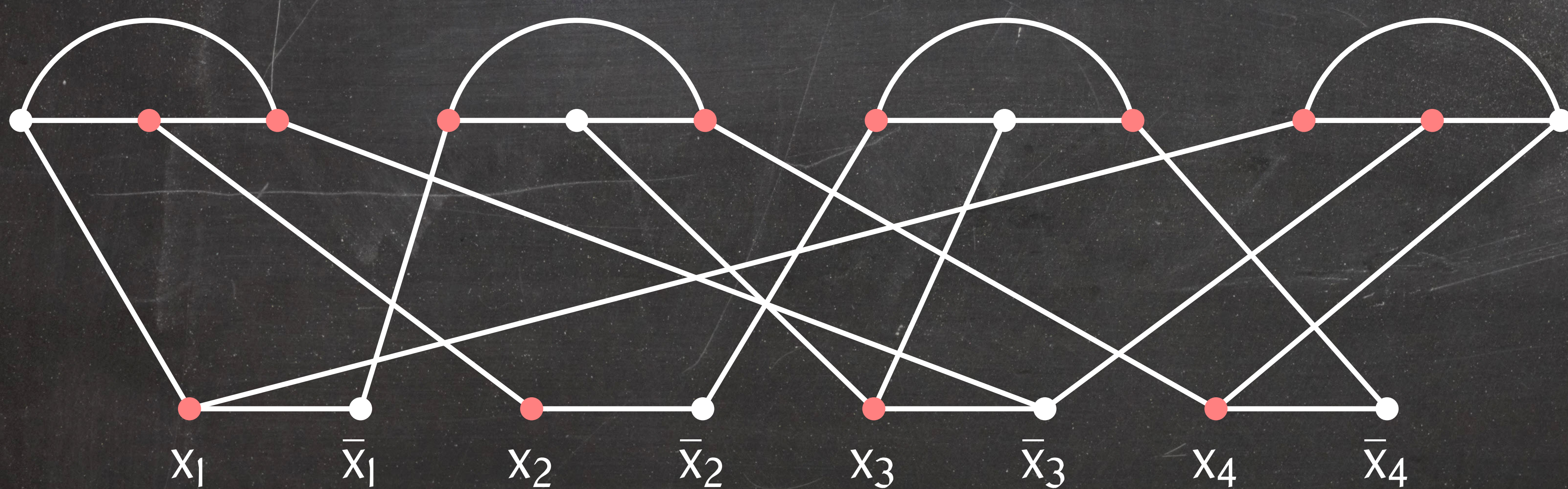
n = number of variables   m = number of clauses

Observation: Any vertex cover of G_F of size $n + 2m$ contains one vertex per variable widget and two vertices per clause widget.

Lemma: F is satisfiable if and only if G_F has a vertex cover of size $n + 2m$.

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$$x_1 = x_2 = x_3 = x_4 = \text{true}$$



Truth assignment

$\downarrow$

Vertex cover

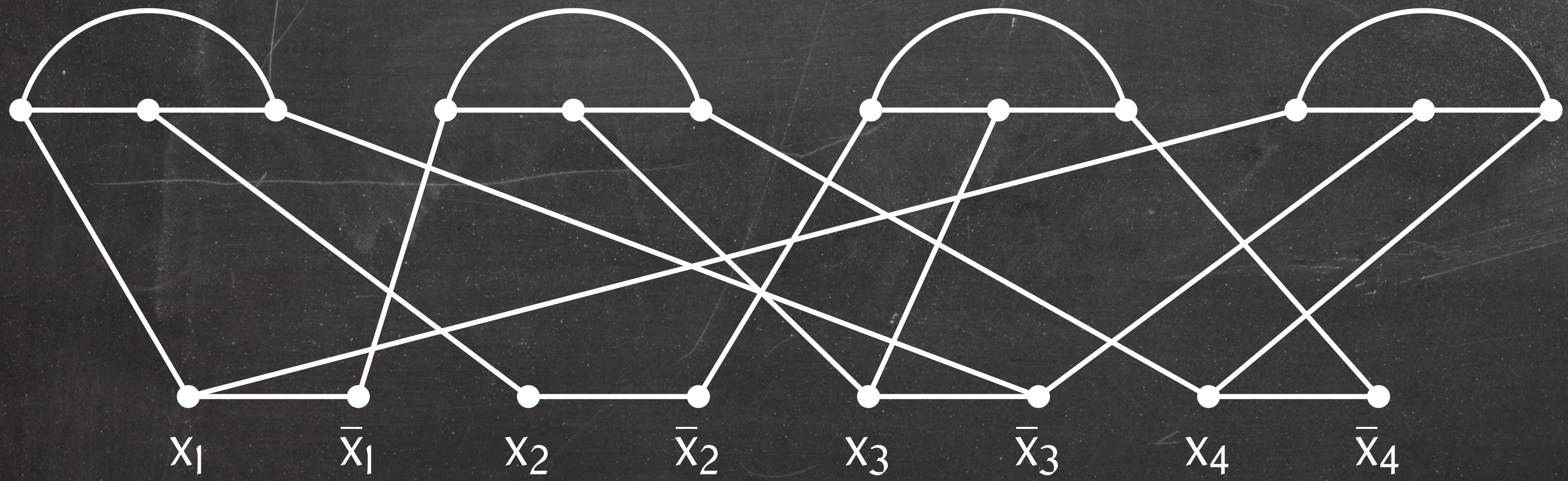
Vertex Cover is NP-Hard

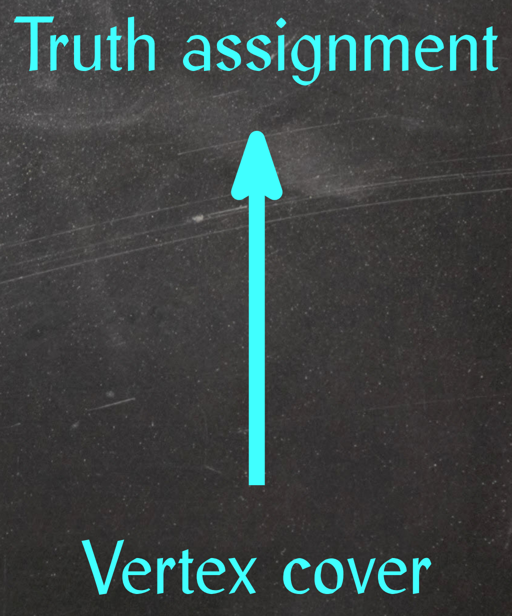
n = number of variables   m = number of clauses

Observation: Any vertex cover of G_F of size $n + 2m$ contains one vertex per variable widget and two vertices per clause widget.

Lemma: F is satisfiable if and only if G_F has a vertex cover of size $n + 2m$.

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$





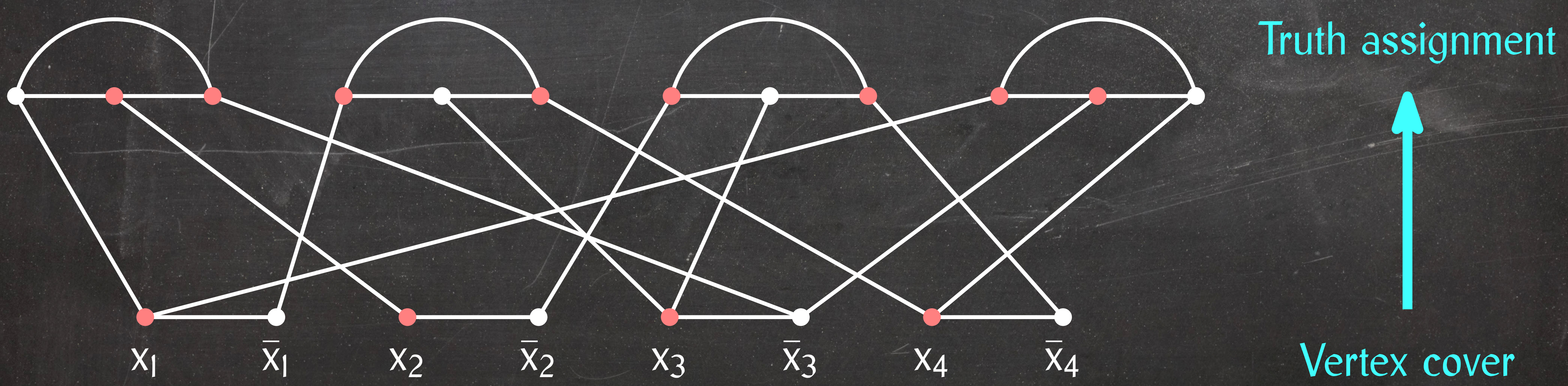
Vertex Cover is NP-Hard

n = number of variables $\quad$ m = number of clauses

Observation: Any vertex cover of G_F of size $n + 2m$ contains one vertex per variable widget and two vertices per clause widget.

Lemma: F is satisfiable if and only if G_F has a vertex cover of size $n + 2m$.

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$



Vertex Cover is NP-Hard

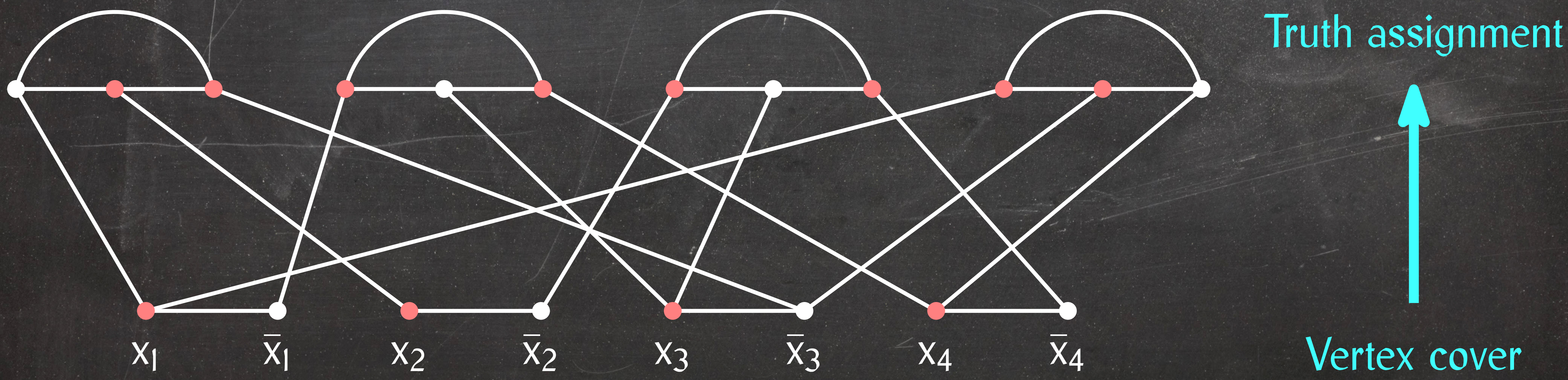
n = number of variables $\qquad m$ = number of clauses

Observation: Any vertex cover of G_F of size $n + 2m$ contains one vertex per variable widget and two vertices per clause widget.

Lemma: F is satisfiable if and only if G_F has a vertex cover of size $n + 2m$.

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$$x_1 = x_2 = x_3 = x_4 = \text{true}$$



Vertex Cover is NP-Hard

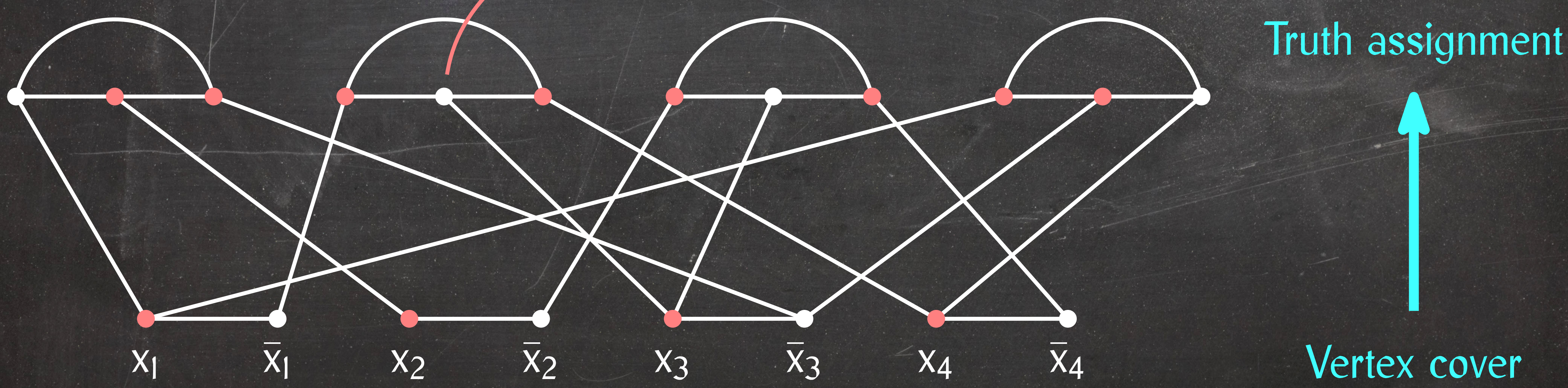
n = number of variables $\qquad$ m = number of clauses

Observation: Any vertex cover of G_F of size $n + 2m$ contains one vertex per variable widget and two vertices per clause widget.

Lemma: F is satisfiable if and only if G_F has a vertex cover of size $n + 2m$.

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$$x_1 = x_2 = x_3 = x_4 = \text{true}$$



Vertex Cover is NP-Complete

Since Vertex Cover is NP-hard, we only have to verify that it is in NP:

Vertex Cover is NP-Complete

Since Vertex Cover is NP-hard, we only have to verify that it is in NP:

Let $VC = \{(G, k) \mid G \text{ has a vertex cover of size } k\}$.

To prove that $VC \in NP$, we have to prove that there exists a language $VC' \in P$ such that $(G, k) \in VC$ if and only if $(G, k, y) \in VC'$ for some y with $|y| \in O(|(G, k)|^c)$.

Vertex Cover is NP-Complete

Since Vertex Cover is NP-hard, we only have to verify that it is in NP:

Let $VC = \{(G, k) \mid G \text{ has a vertex cover of size } k\}$.

To prove that $VC \in NP$, we have to prove that there exists a language $VC' \in P$ such that $(G, k) \in VC$ if and only if $(G, k, y) \in VC'$ for some y with $|y| \in O(|(G, k)|^c)$.

Let $VC' = \{(G, k, C) \mid C \text{ is a vertex cover of } G \text{ of size } k\}$.

Vertex Cover is NP-Complete

Since Vertex Cover is NP-hard, we only have to verify that it is in NP:

Let $VC = \{(G, k) \mid G \text{ has a vertex cover of size } k\}$.

To prove that $VC \in NP$, we have to prove that there exists a language $VC' \in P$ such that $(G, k) \in VC$ if and only if $(G, k, y) \in VC'$ for some y with $|y| \in O(|(G, k)|^c)$.

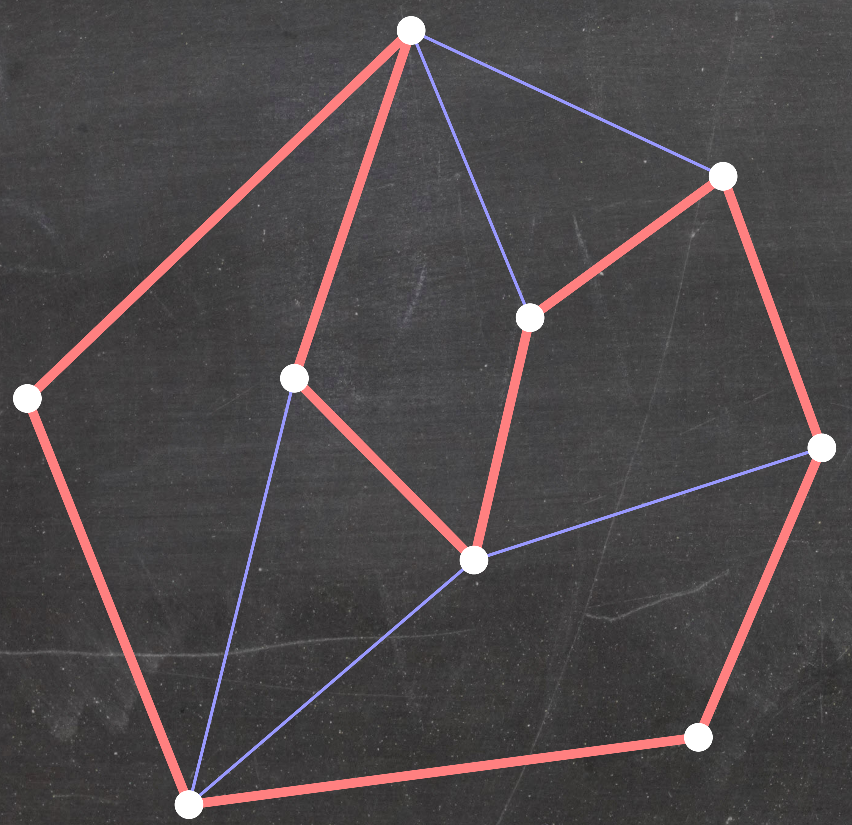
Let $VC' = \{(G, k, C) \mid C \text{ is a vertex cover of } G \text{ of size } k\}$.

$VC' \in P$:

- We can test in polynomial time whether every vertex in C belongs to G.
- We can test in polynomial time whether $|C| = k$.
- We can test in polynomial time whether every edge of G has at least one endpoint in C.

Hamiltonian Cycle

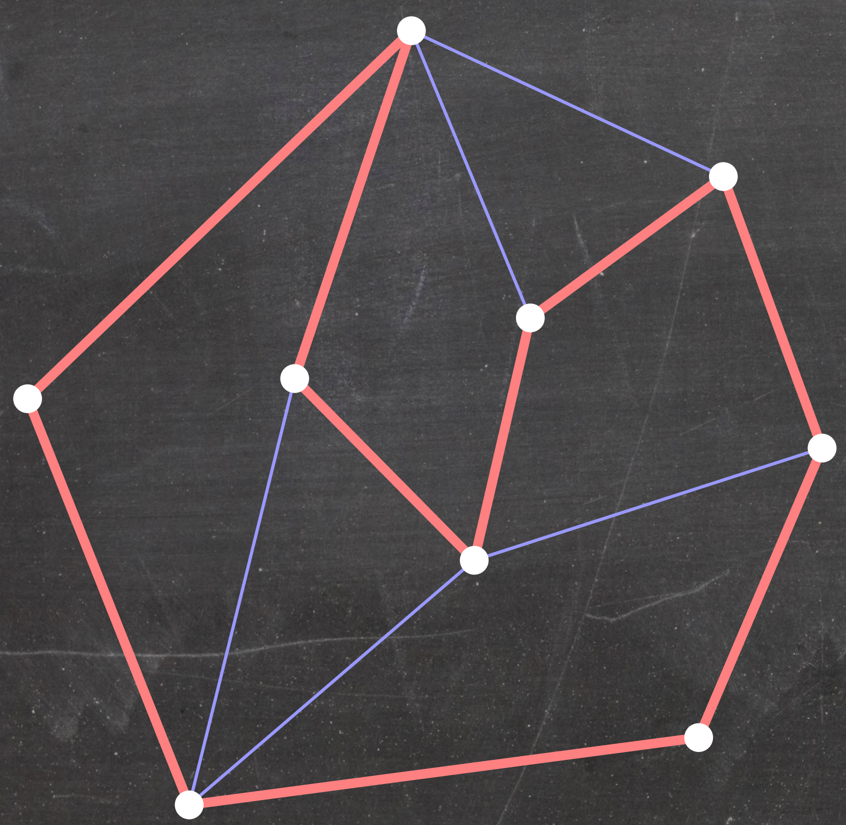
A Hamiltonian cycle of a graph G is a simple cycle that contains all vertices of G and whose edges are edges of G.



Hamiltonian Cycle

A Hamiltonian cycle of a graph G is a simple cycle that contains all vertices of G and whose edges are edges of G.

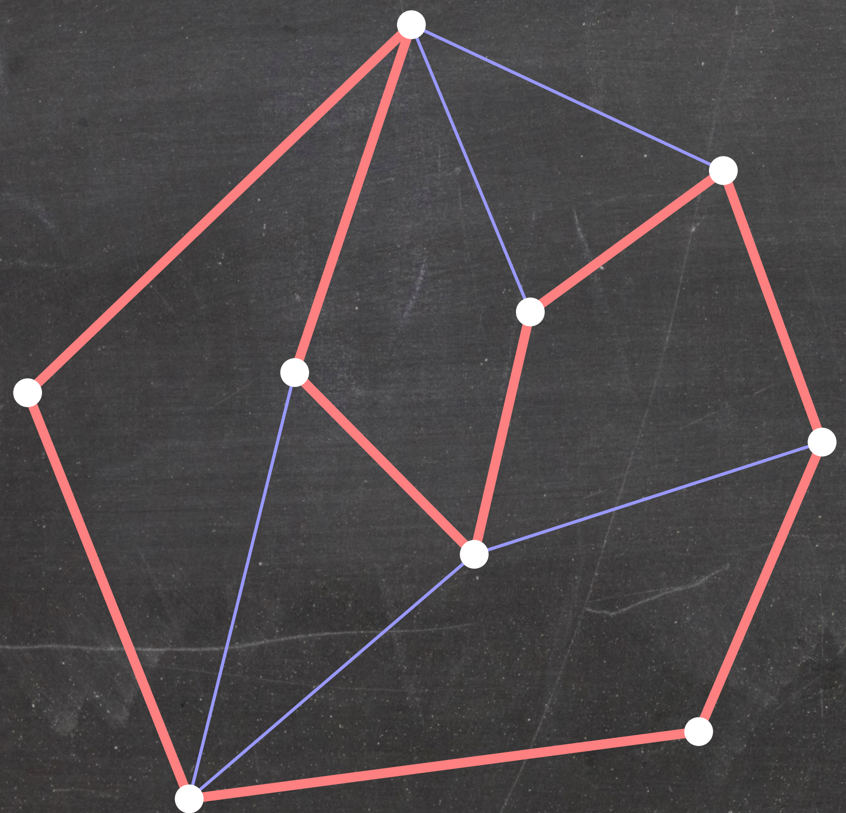
A graph G is Hamiltonian if it has a Hamiltonian cycle.



Hamiltonian Cycle

A Hamiltonian cycle of a graph G is a simple cycle that contains all vertices of G and whose edges are edges of G.

A graph G is Hamiltonian if it has a Hamiltonian cycle.

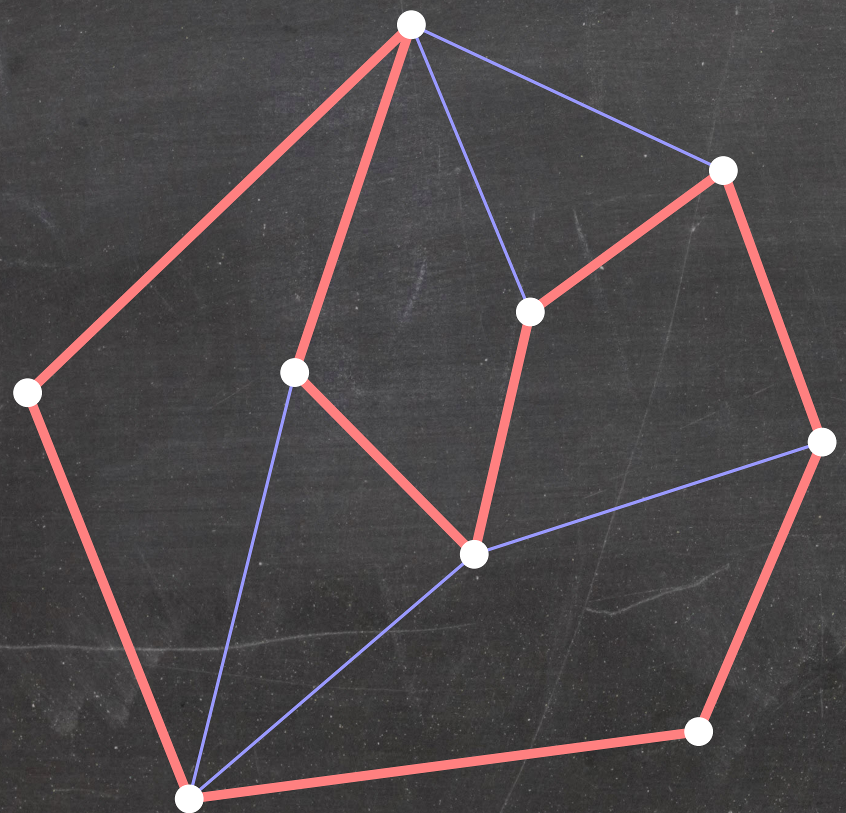


Hamiltonian

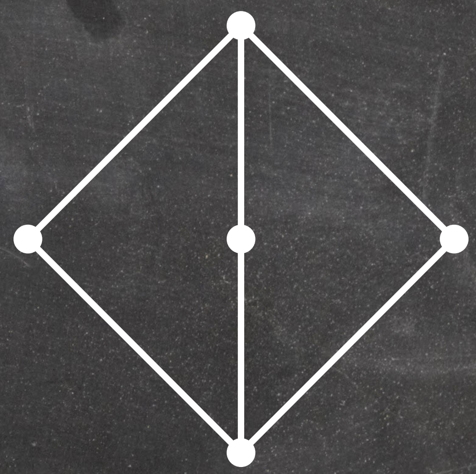
Hamiltonian Cycle

A Hamiltonian cycle of a graph G is a simple cycle that contains all vertices of G and whose edges are edges of G.

A graph G is Hamiltonian if it has a Hamiltonian cycle.



Hamiltonian



not Hamiltonian

Hamiltonian Cycle is NP-Complete

Hamiltonian Cycle Problem: Decide whether a given graph G is Hamiltonian.

Hamiltonian Cycle is NP-Complete

Hamiltonian Cycle Problem: Decide whether a given graph G is Hamiltonian.

Exercise: Verify that Hamiltonian Cycle is in NP.

Hamiltonian Cycle is NP-Complete

Hamiltonian Cycle Problem: Decide whether a given graph G is Hamiltonian.

Exercise: Verify that Hamiltonian Cycle is in NP.

To prove: Hamiltonian Cycle is NP-hard.

Hamiltonian Cycle is NP-Complete

Hamiltonian Cycle Problem: Decide whether a given graph G is Hamiltonian.

Exercise: Verify that Hamiltonian Cycle is in NP.

To prove: Hamiltonian Cycle is NP-hard.

Reduction from Vertex Cover: Given a vertex cover instance (G, k), we build a graph G' that has a Hamiltonian cycle if and only if G has a vertex cover of size k.

Hamiltonian Cycle is NP-Complete

Hamiltonian Cycle Problem: Decide whether a given graph G is Hamiltonian.

Exercise: Verify that Hamiltonian Cycle is in NP.

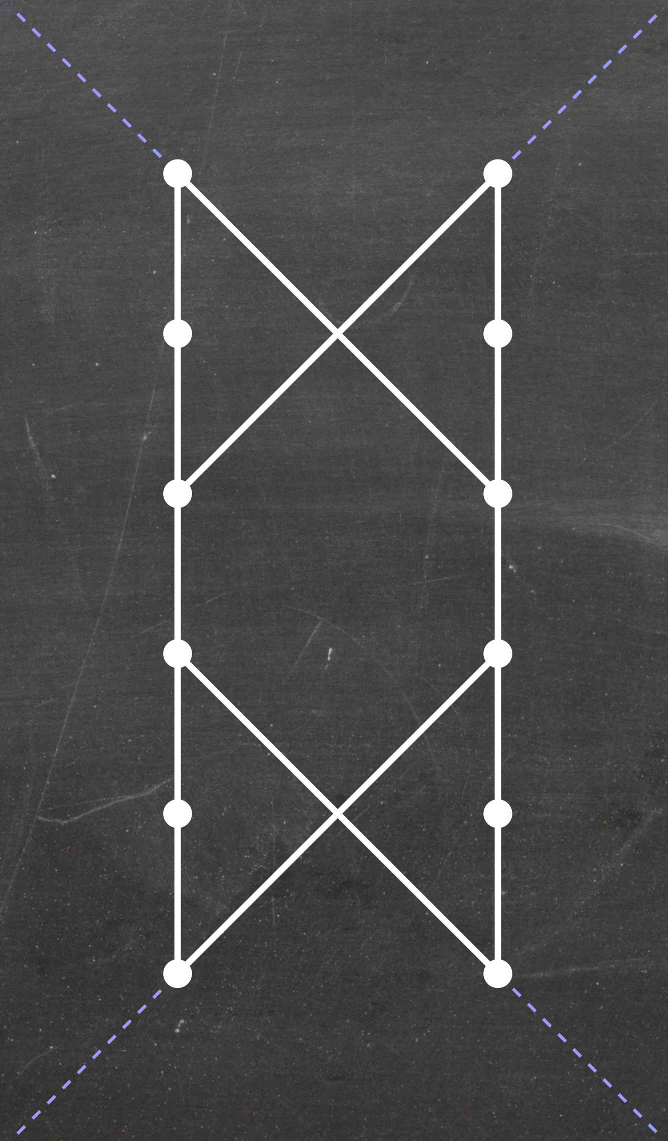
To prove: Hamiltonian Cycle is NP-hard.

Reduction from Vertex Cover: Given a vertex cover instance (G, k), we build a graph G' that has a Hamiltonian cycle if and only if G has a vertex cover of size k.

Again, it is trivial to verify that the construction takes polynomial time.

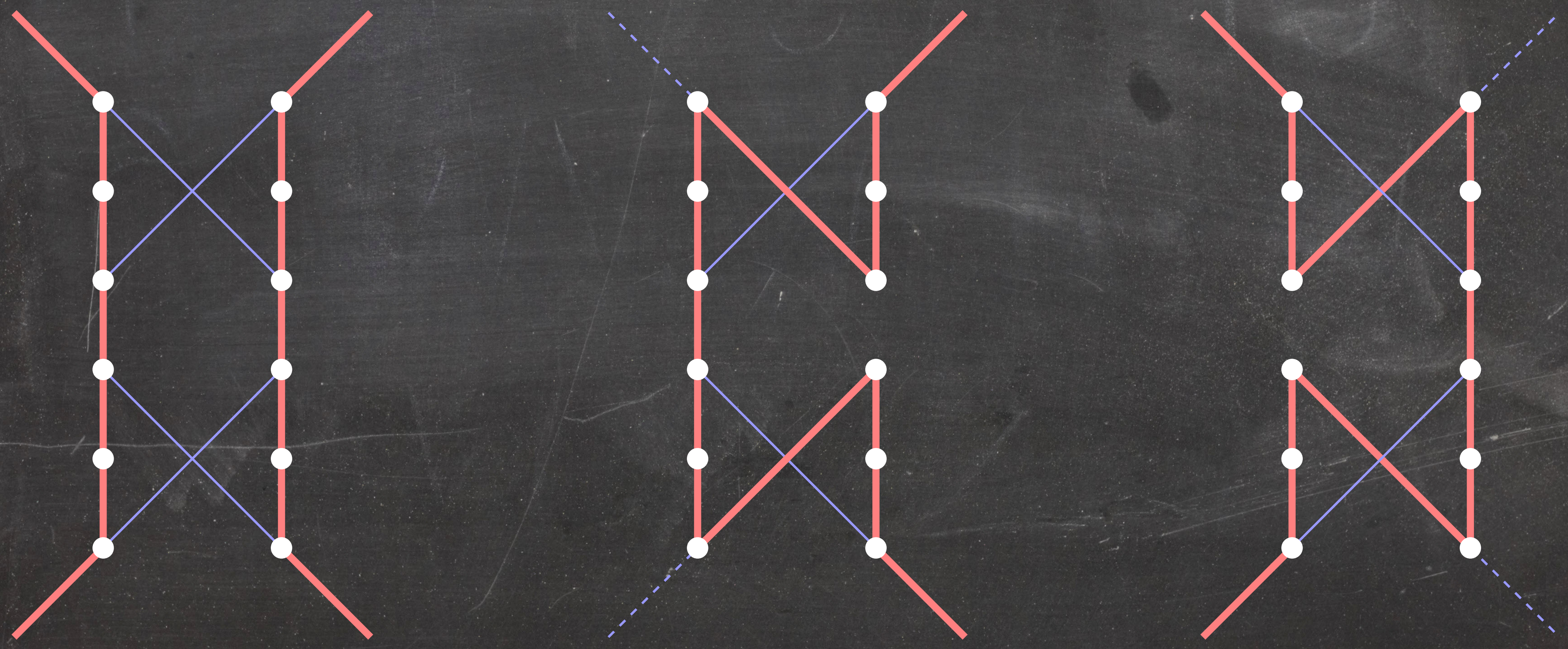
Edge Widgets

We build G' from edge widgets.



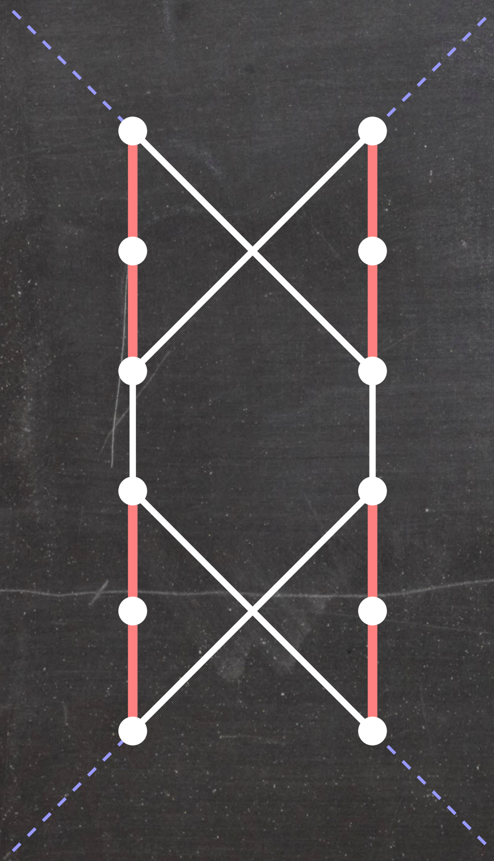
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



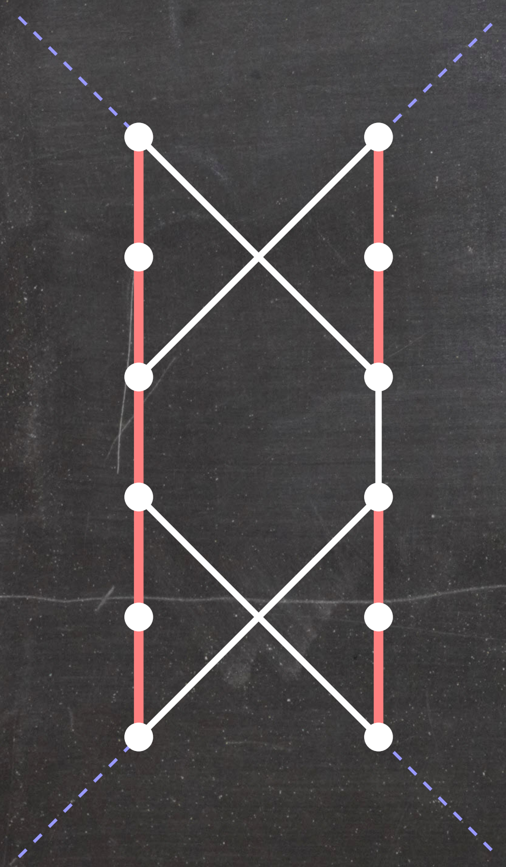
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



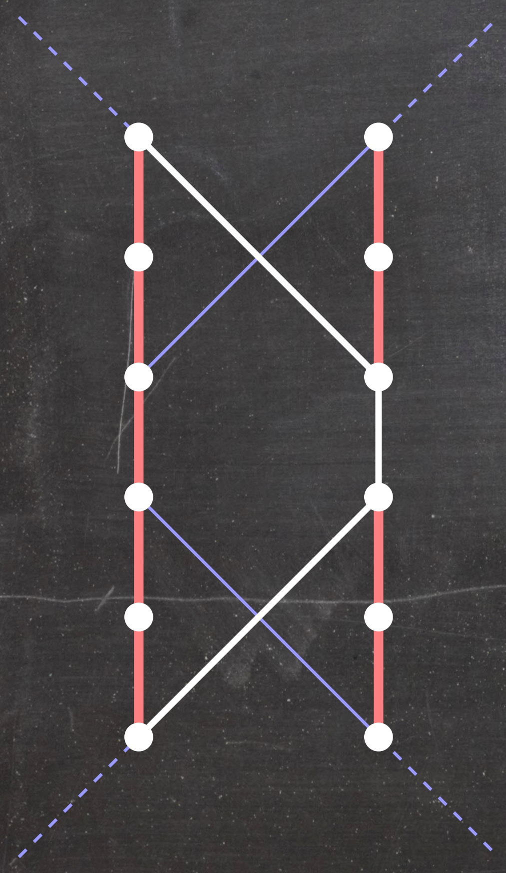
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



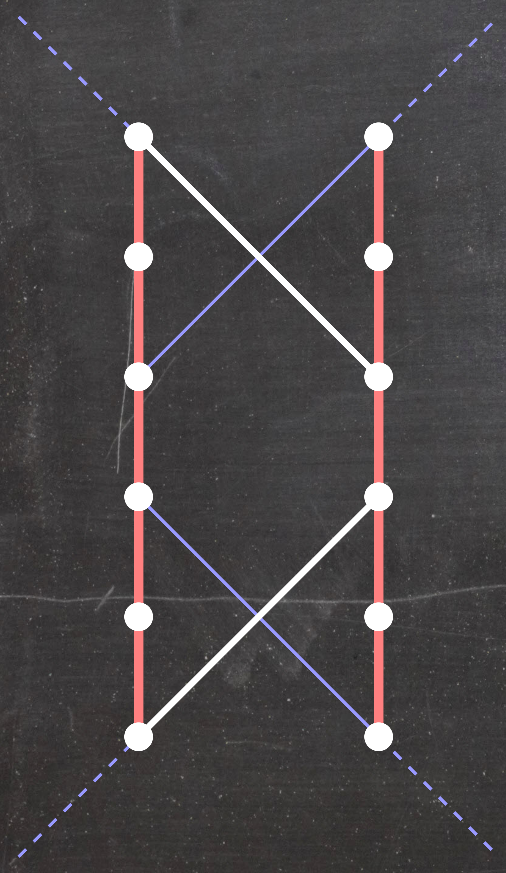
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



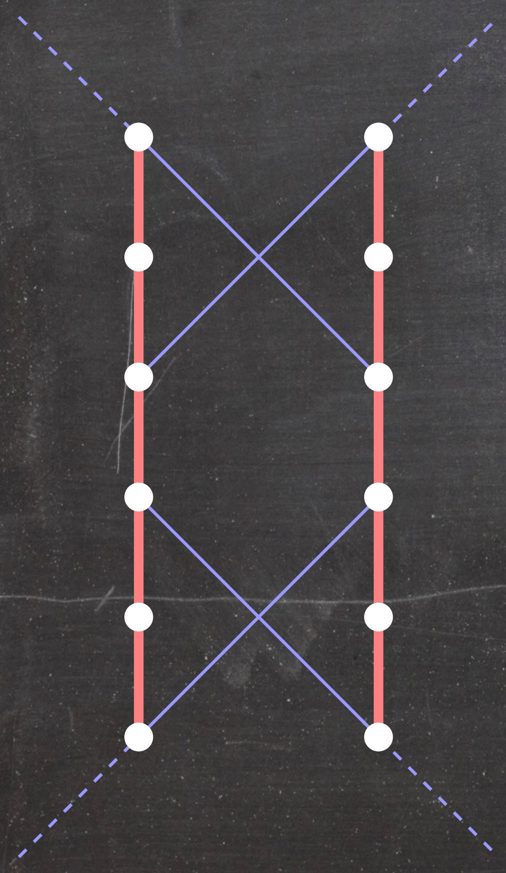
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



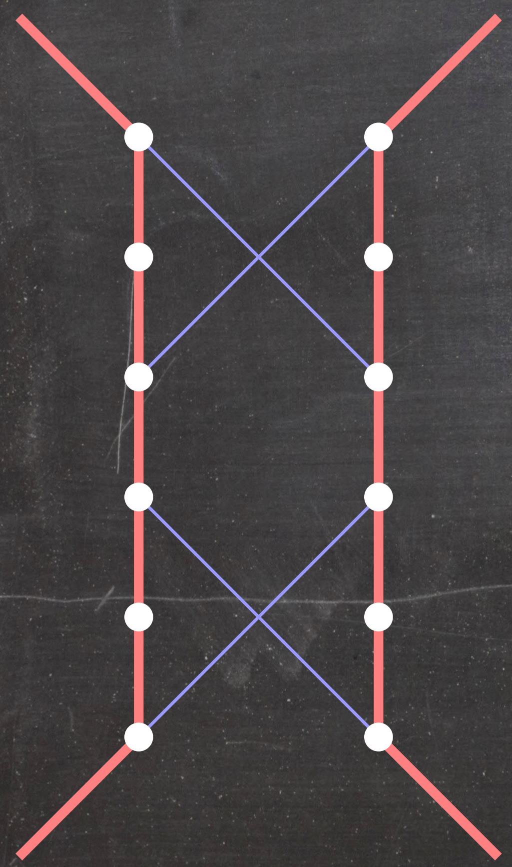
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



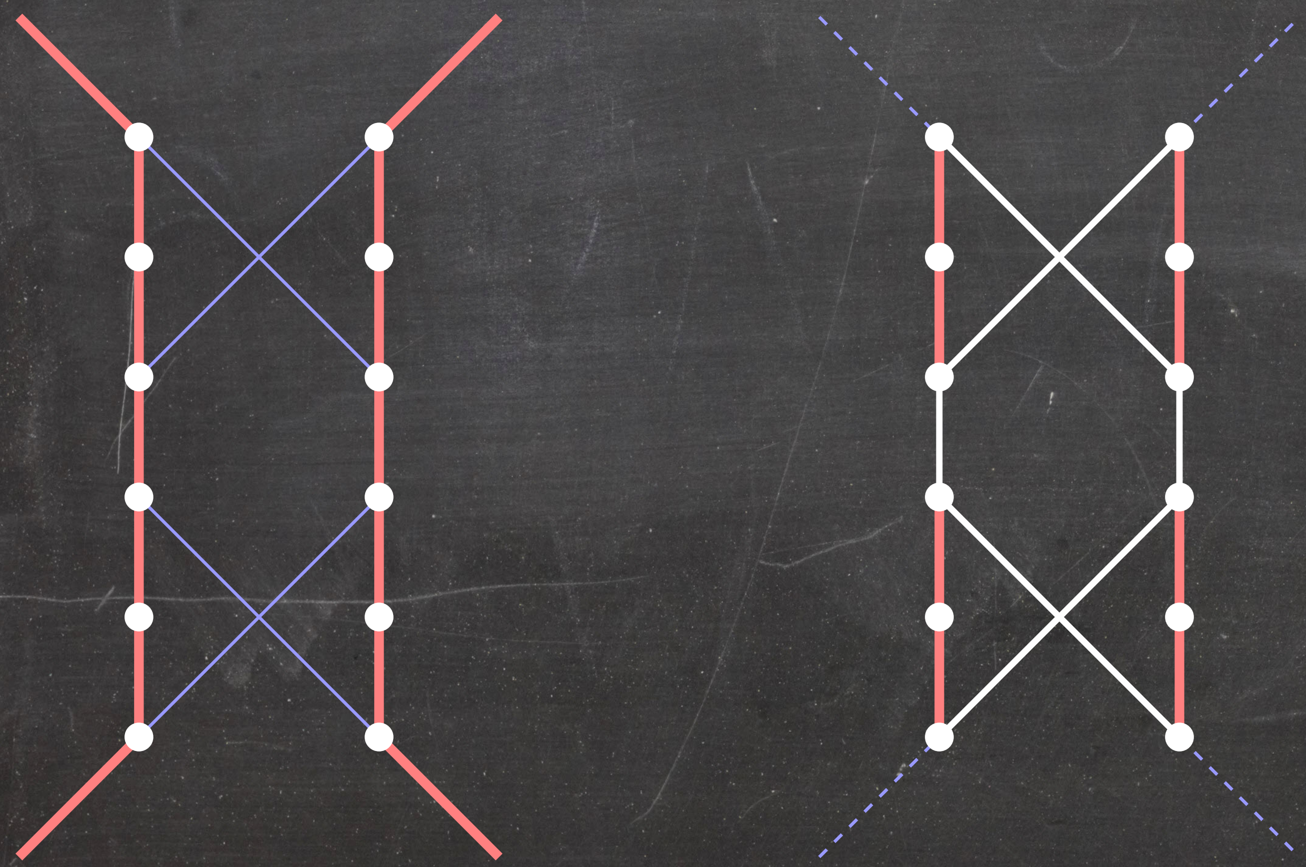
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



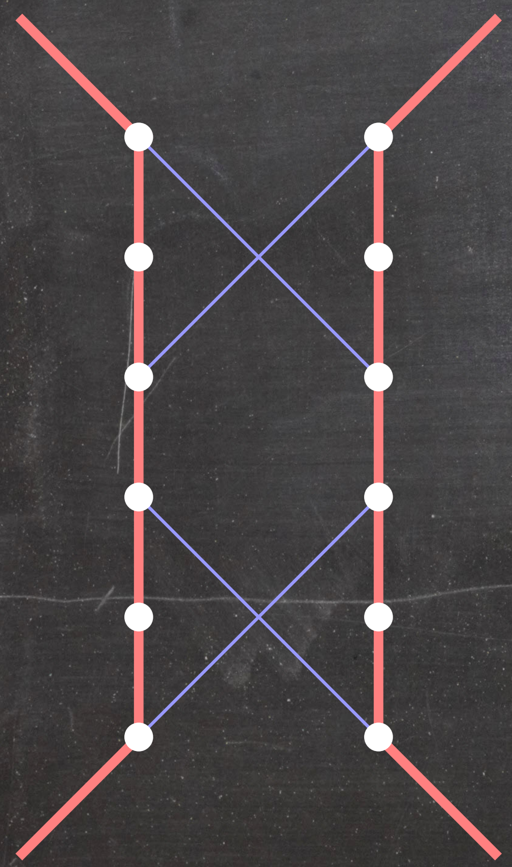
Edge Widgets

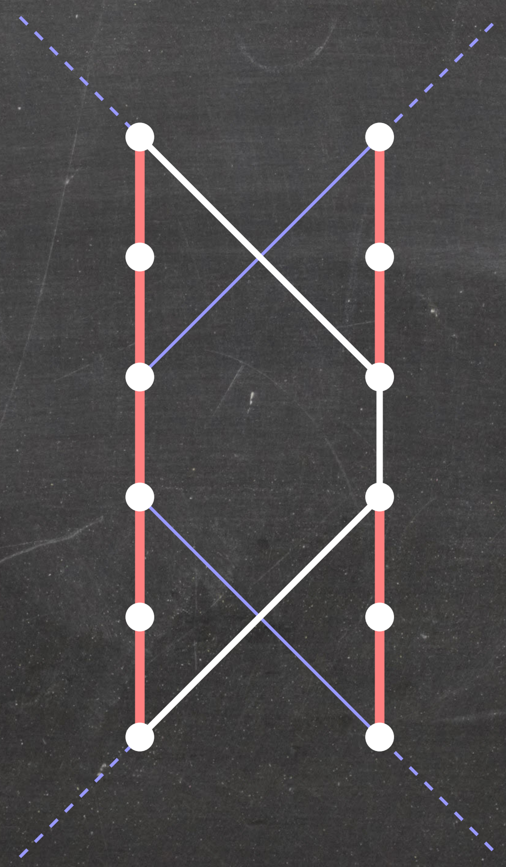
Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



Edge Widgets

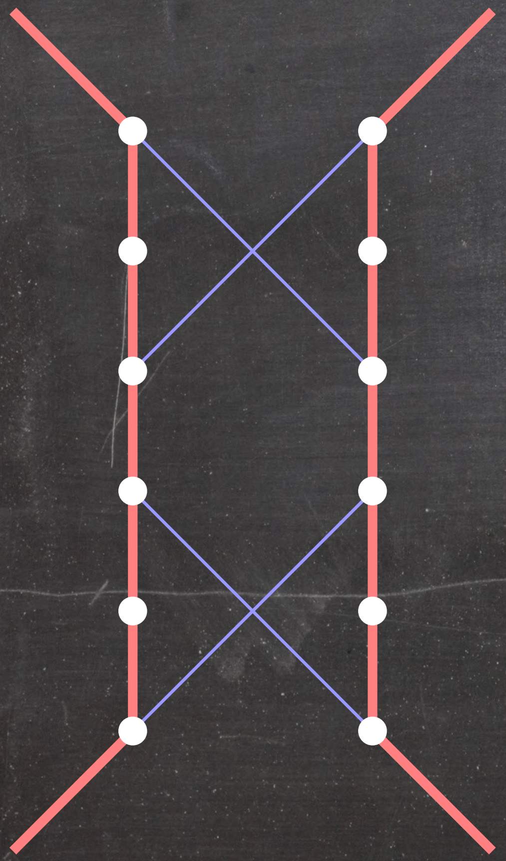
Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.

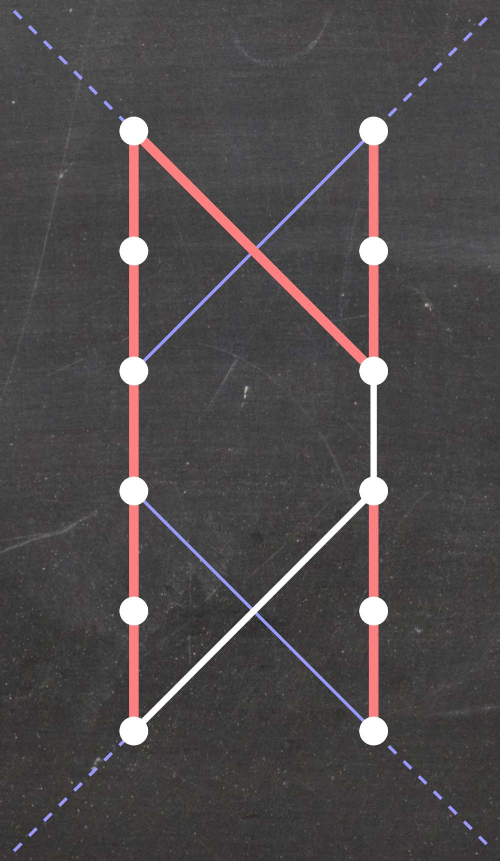




Edge Widgets

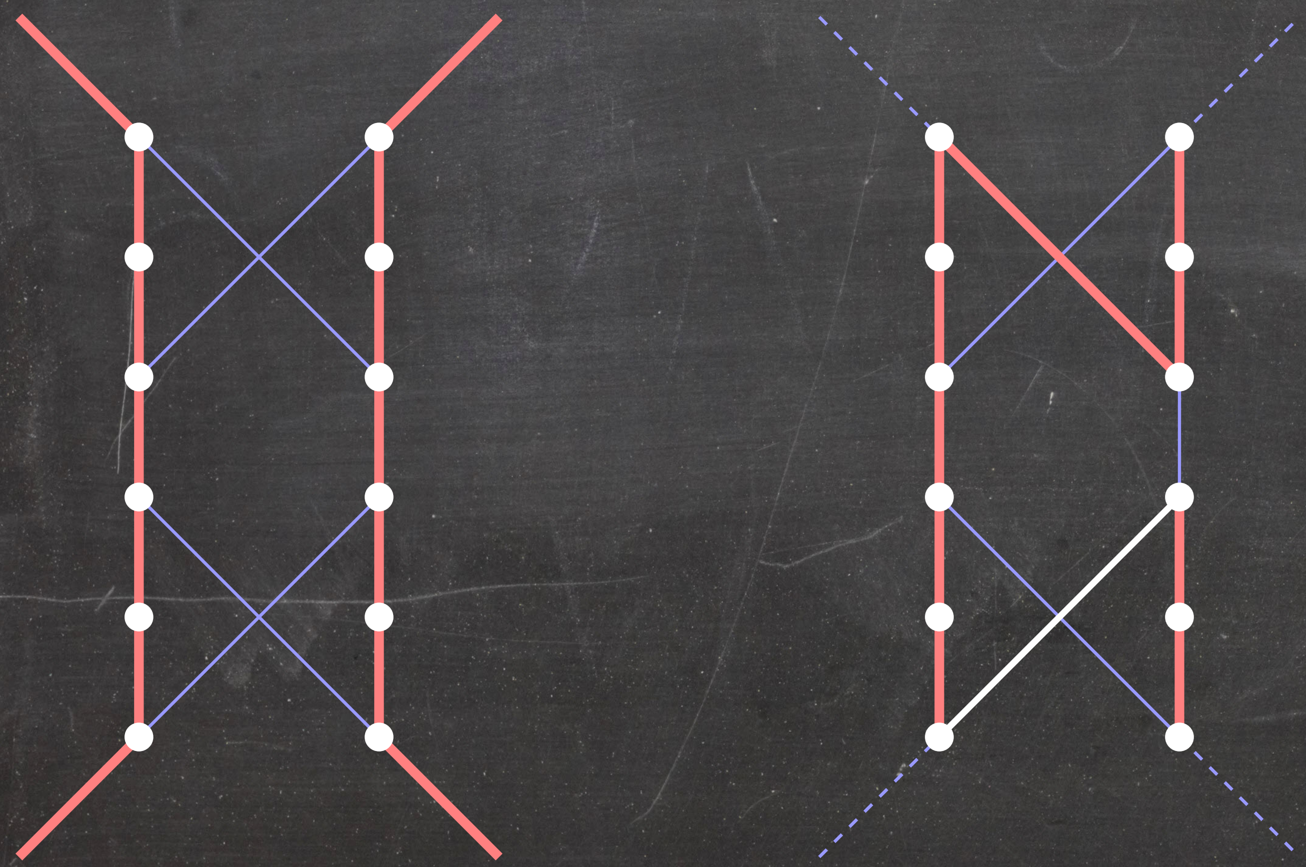
Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.





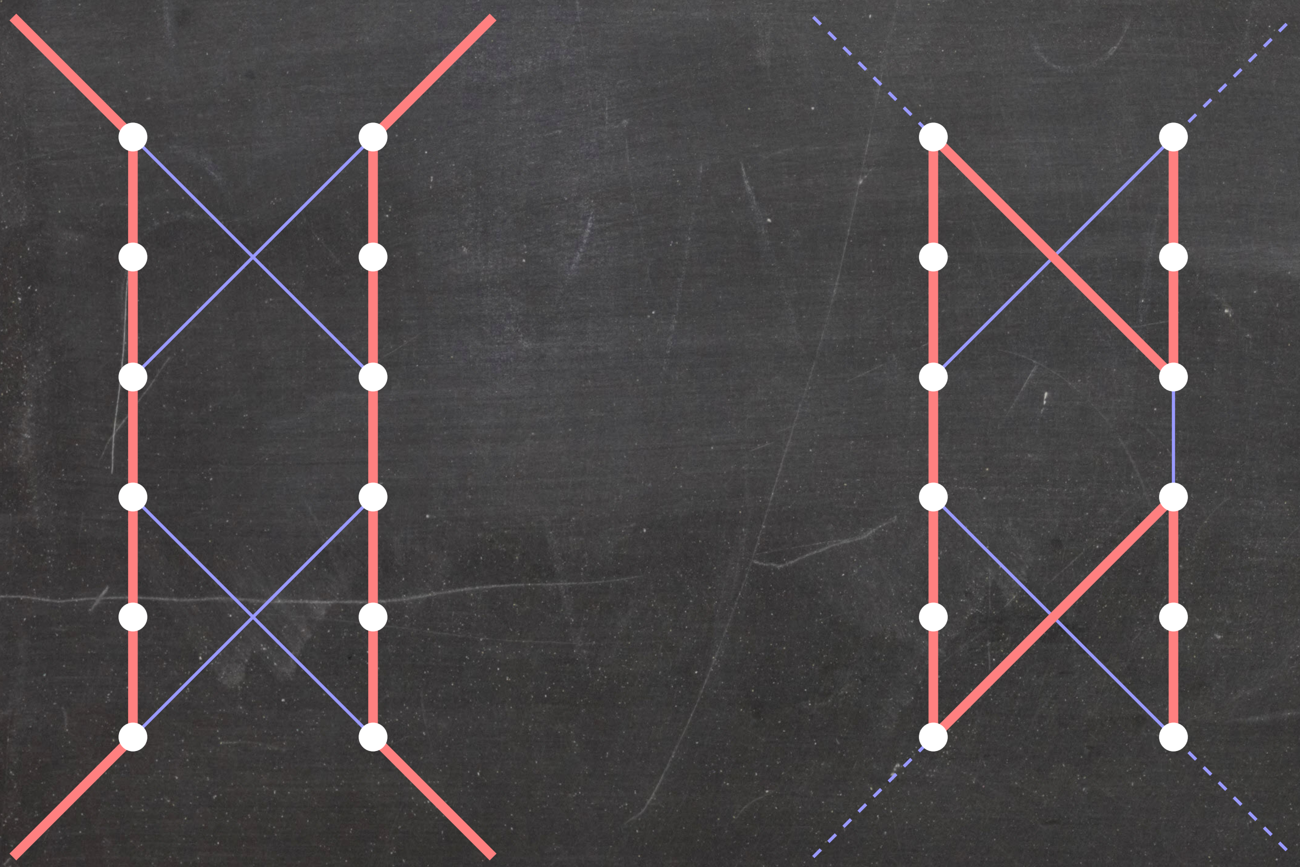
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



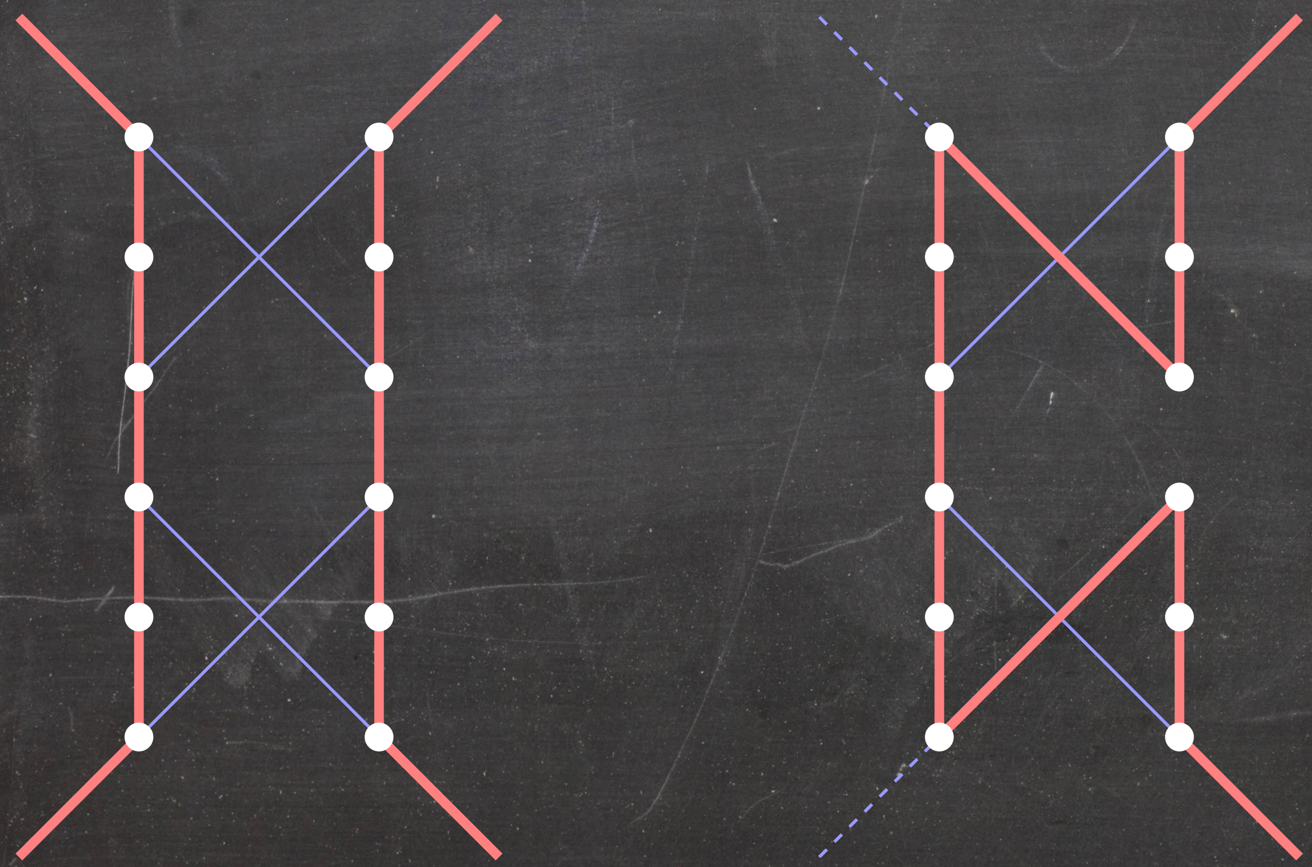
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



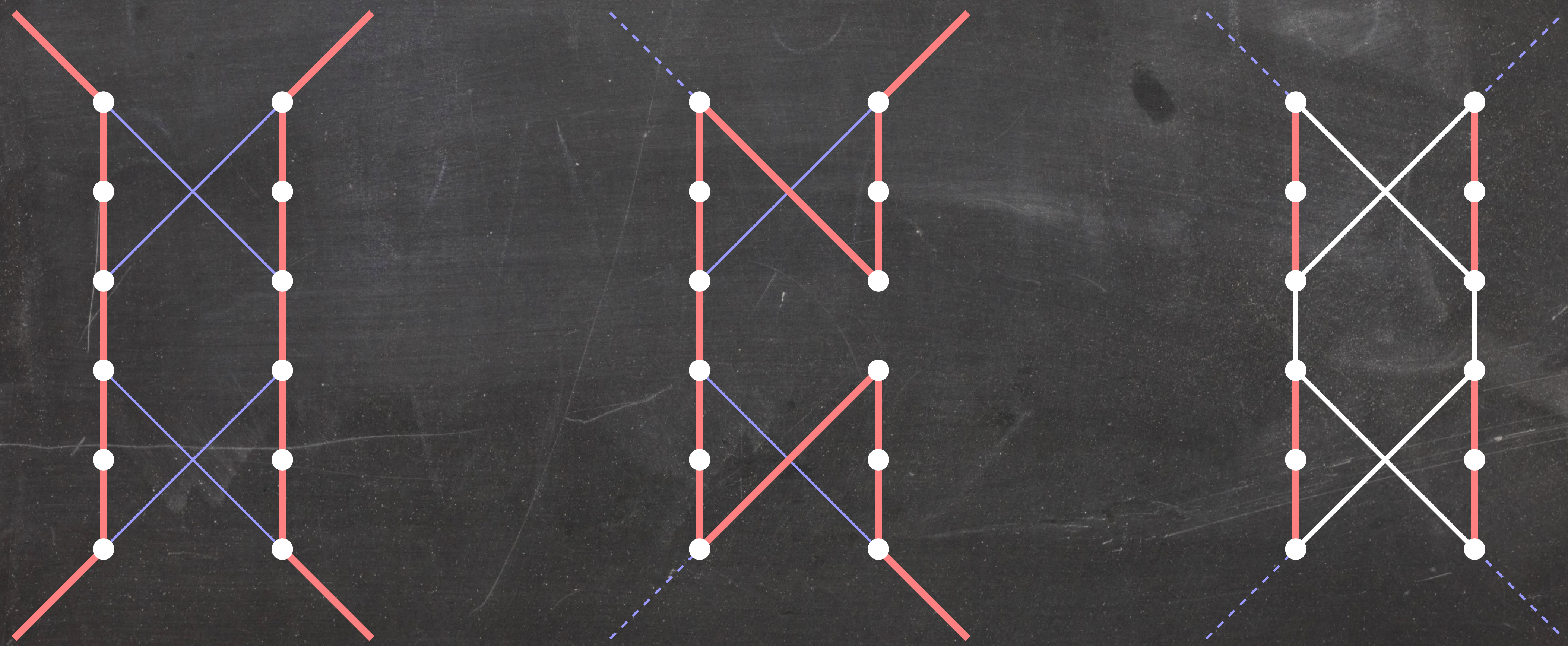
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



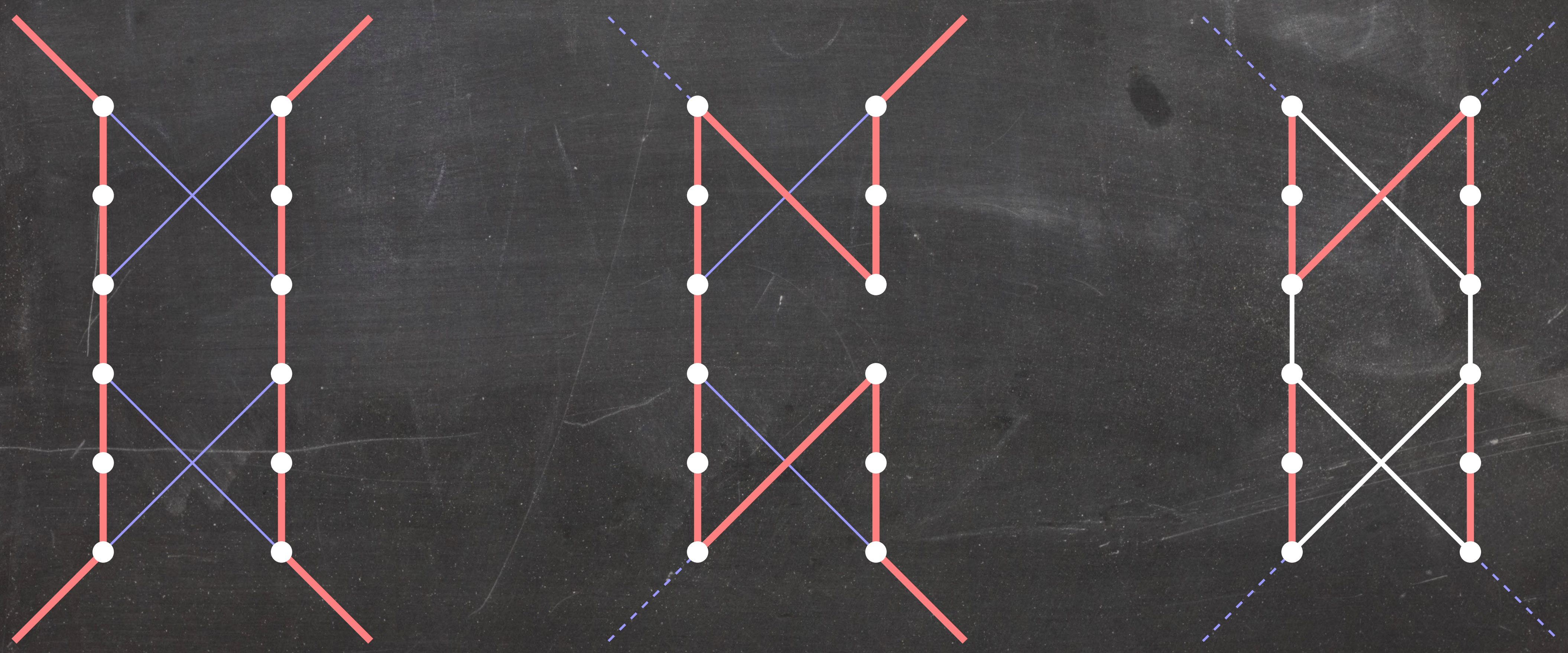
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



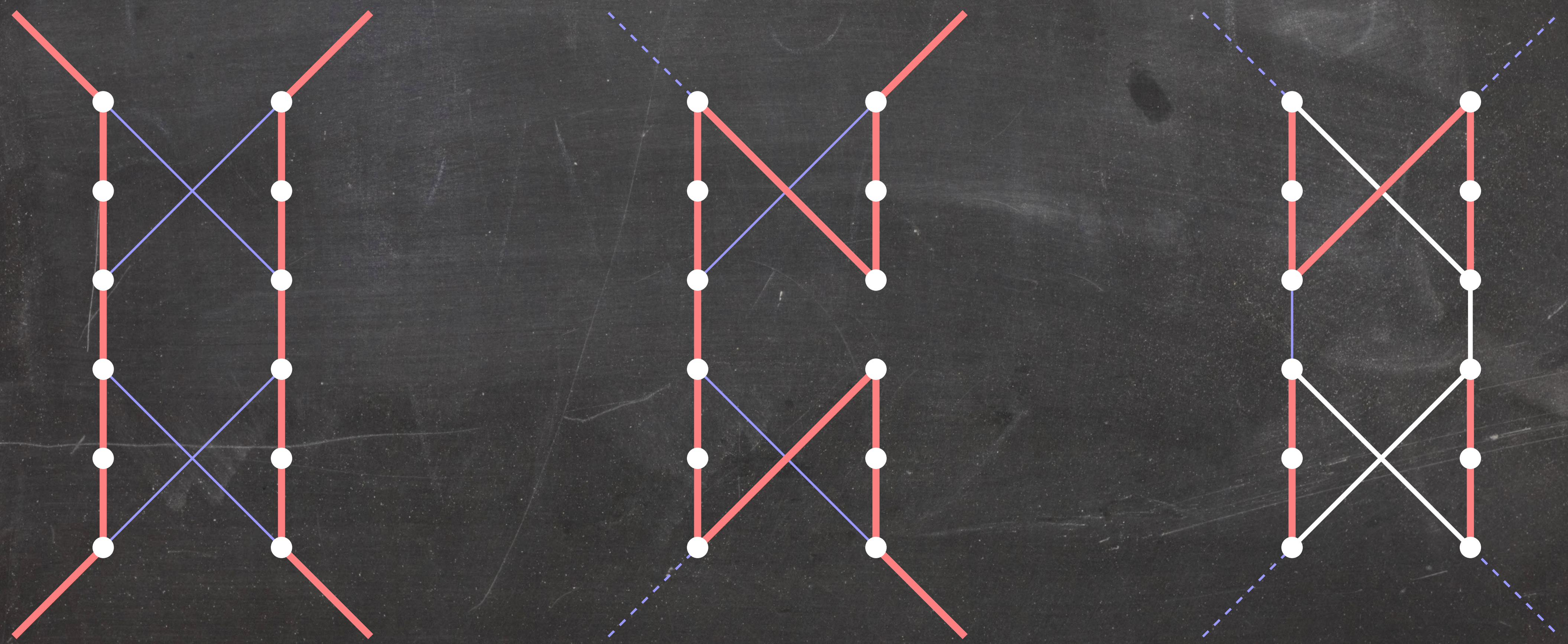
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



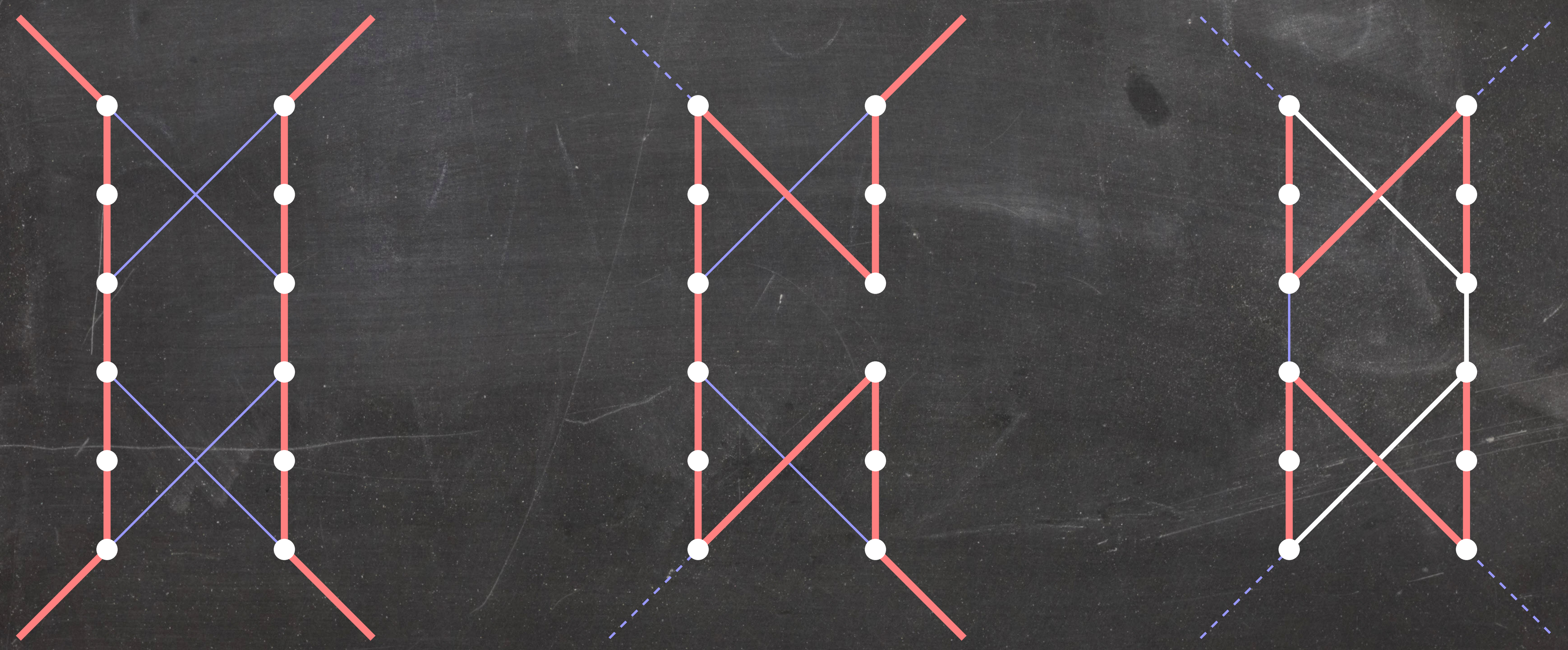
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



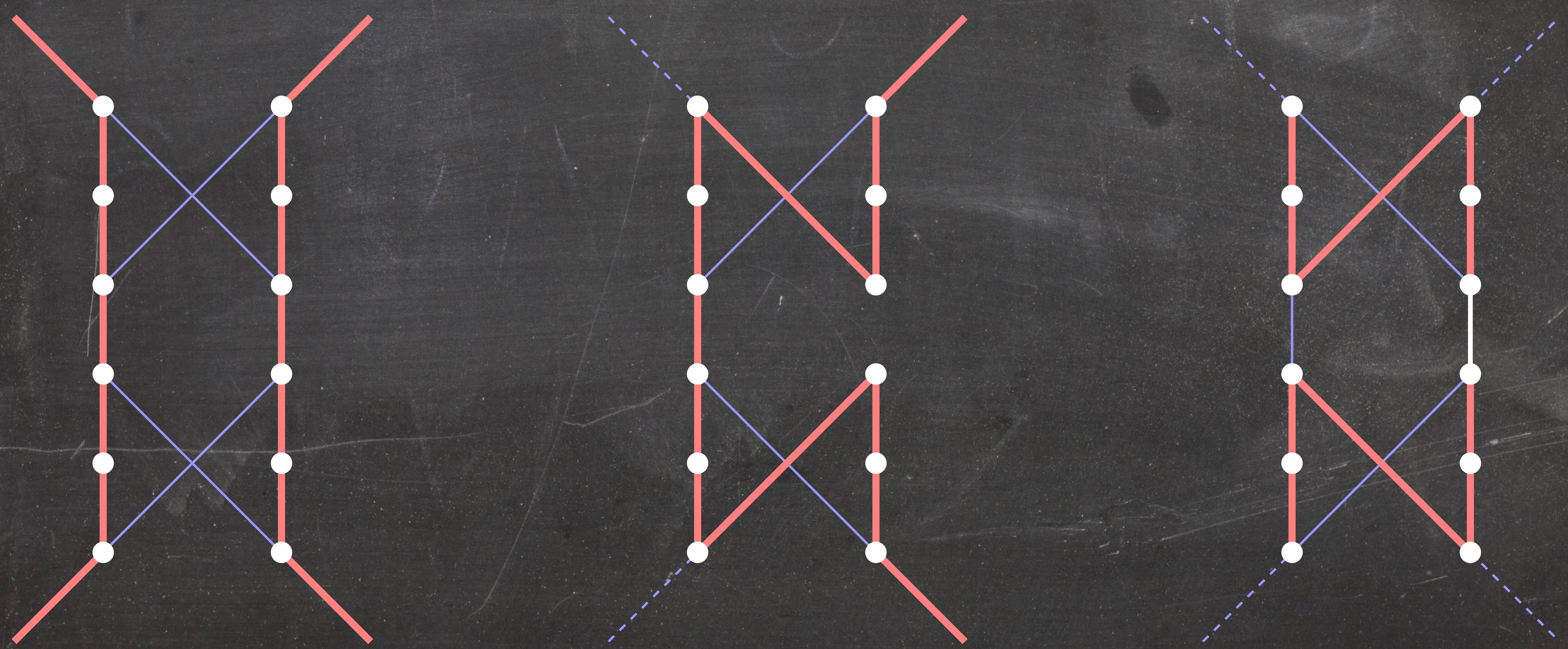
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



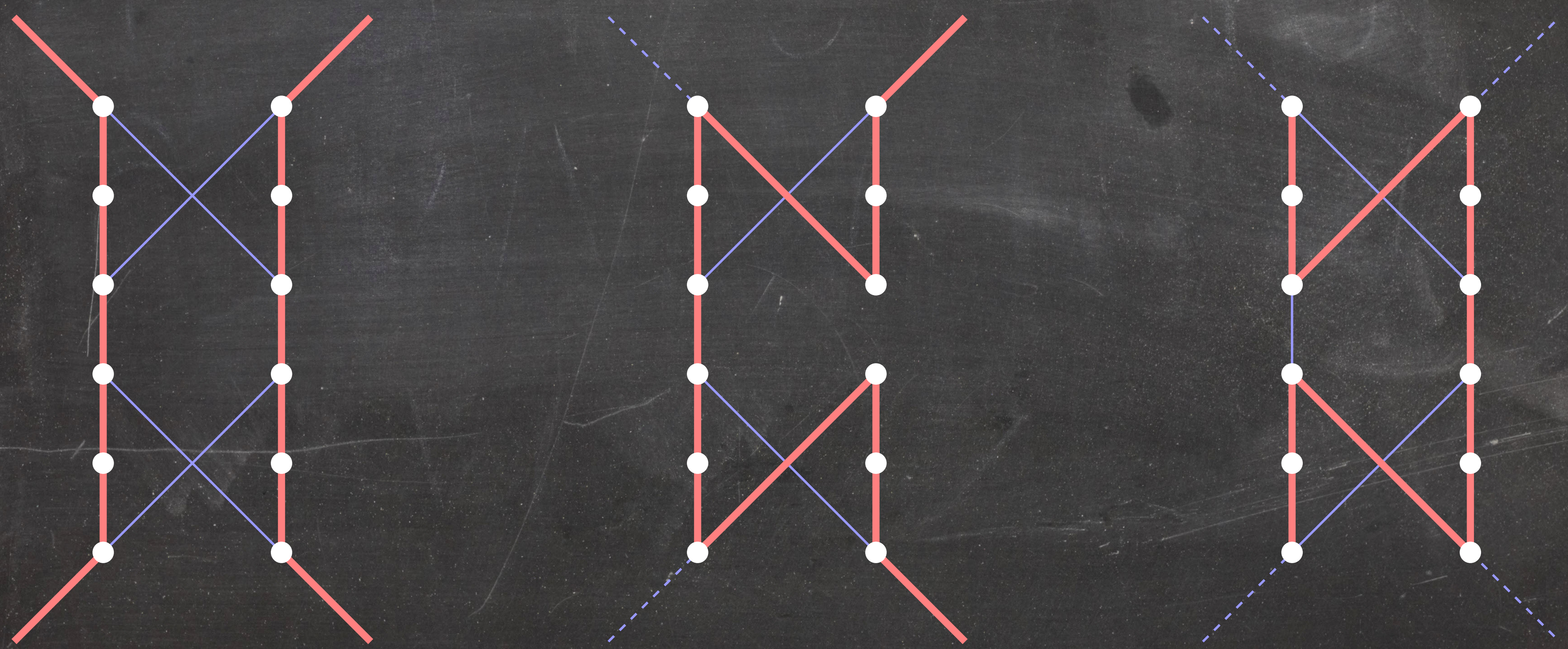
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



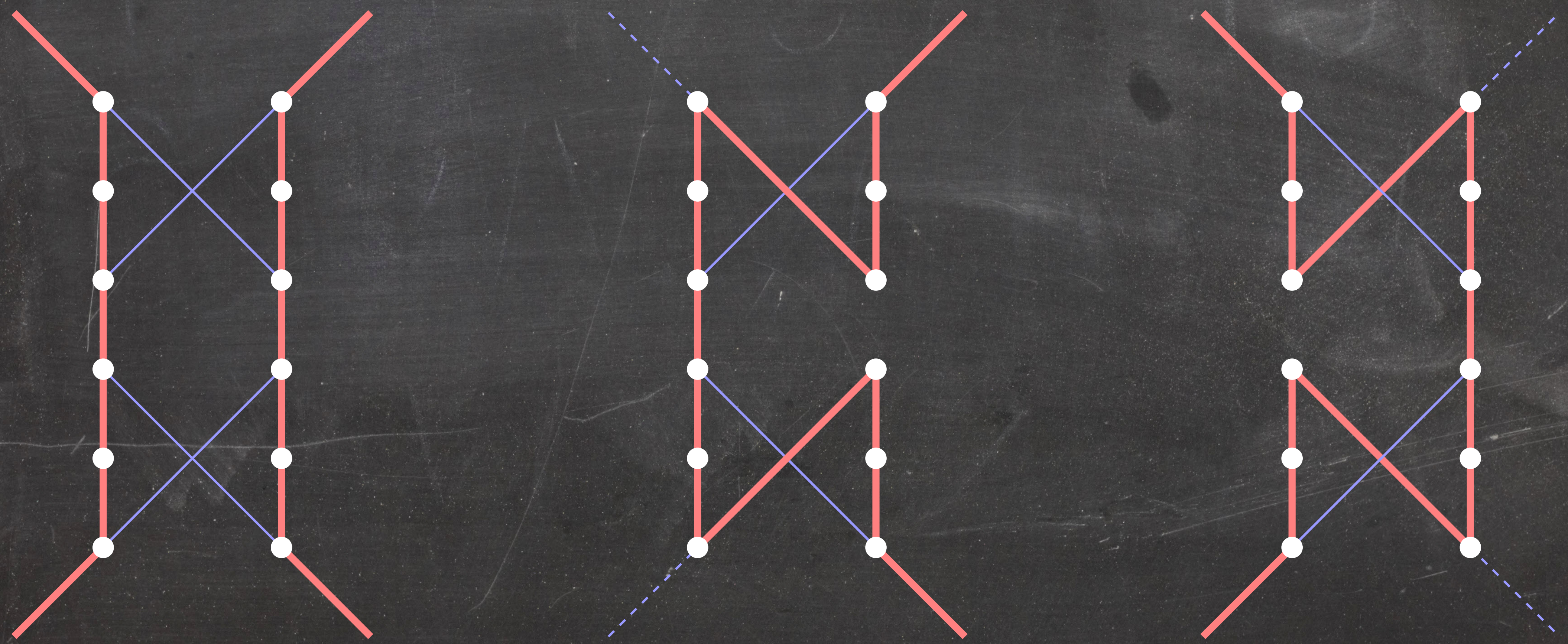
Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.

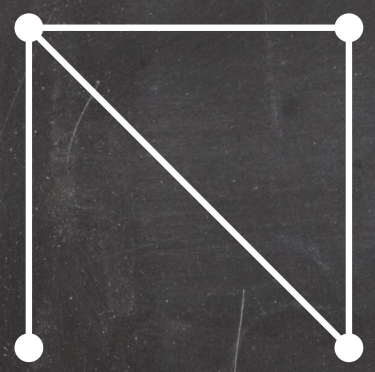


Edge Widgets

Observation: Any Hamiltonian cycle of a graph built from edge widgets traverses every edge widget in one of three ways.



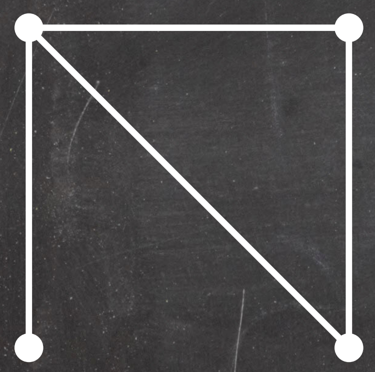
Hamiltonian Cycle is NP-Complete



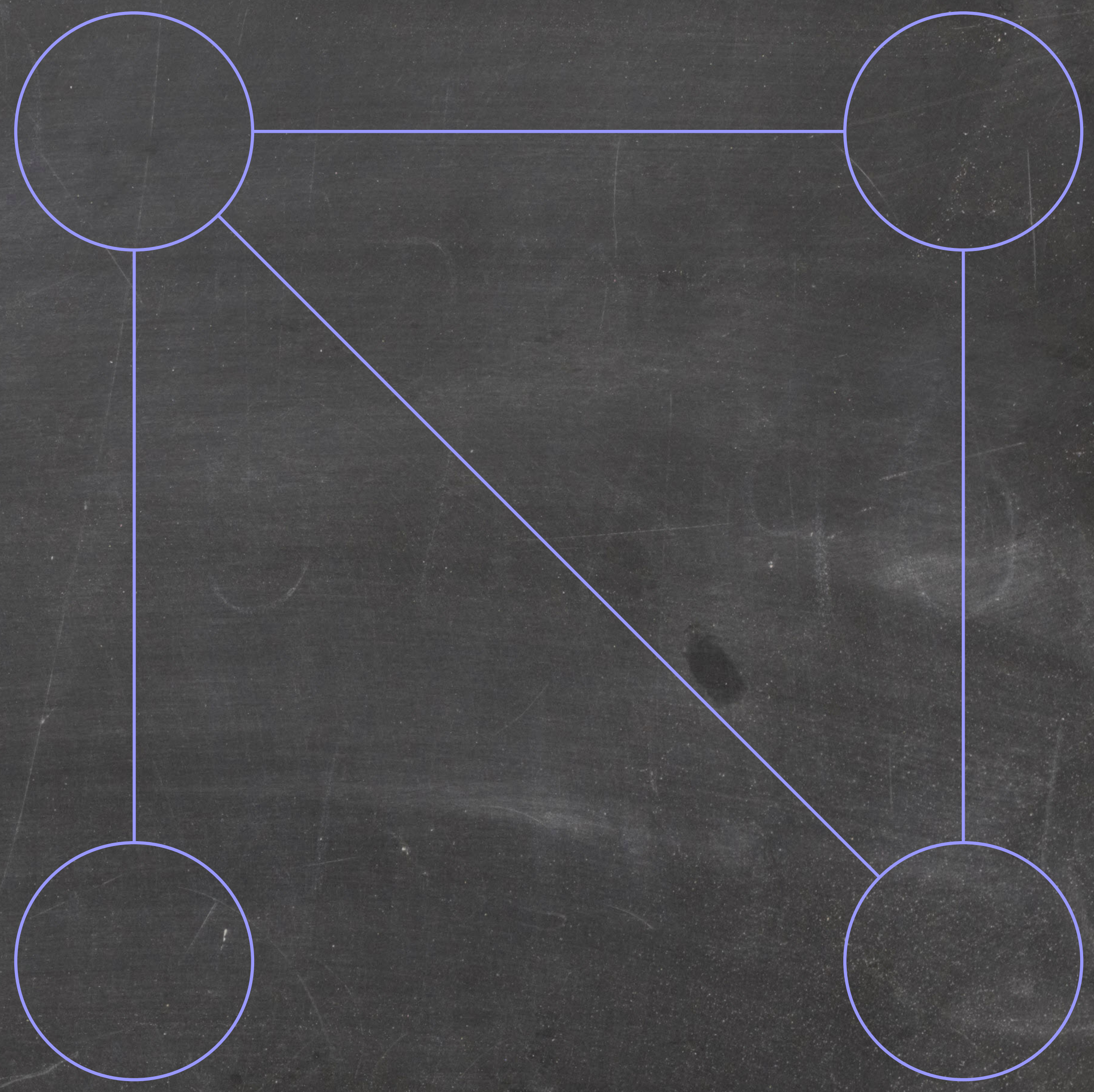
G

k = 2

Hamiltonian Cycle is NP-Complete

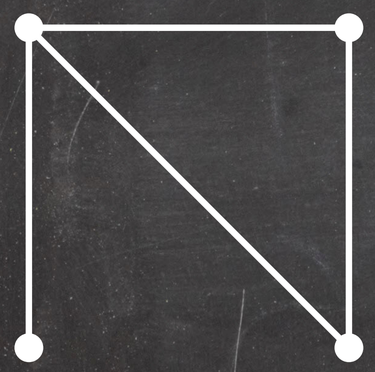


G
$k = 2$

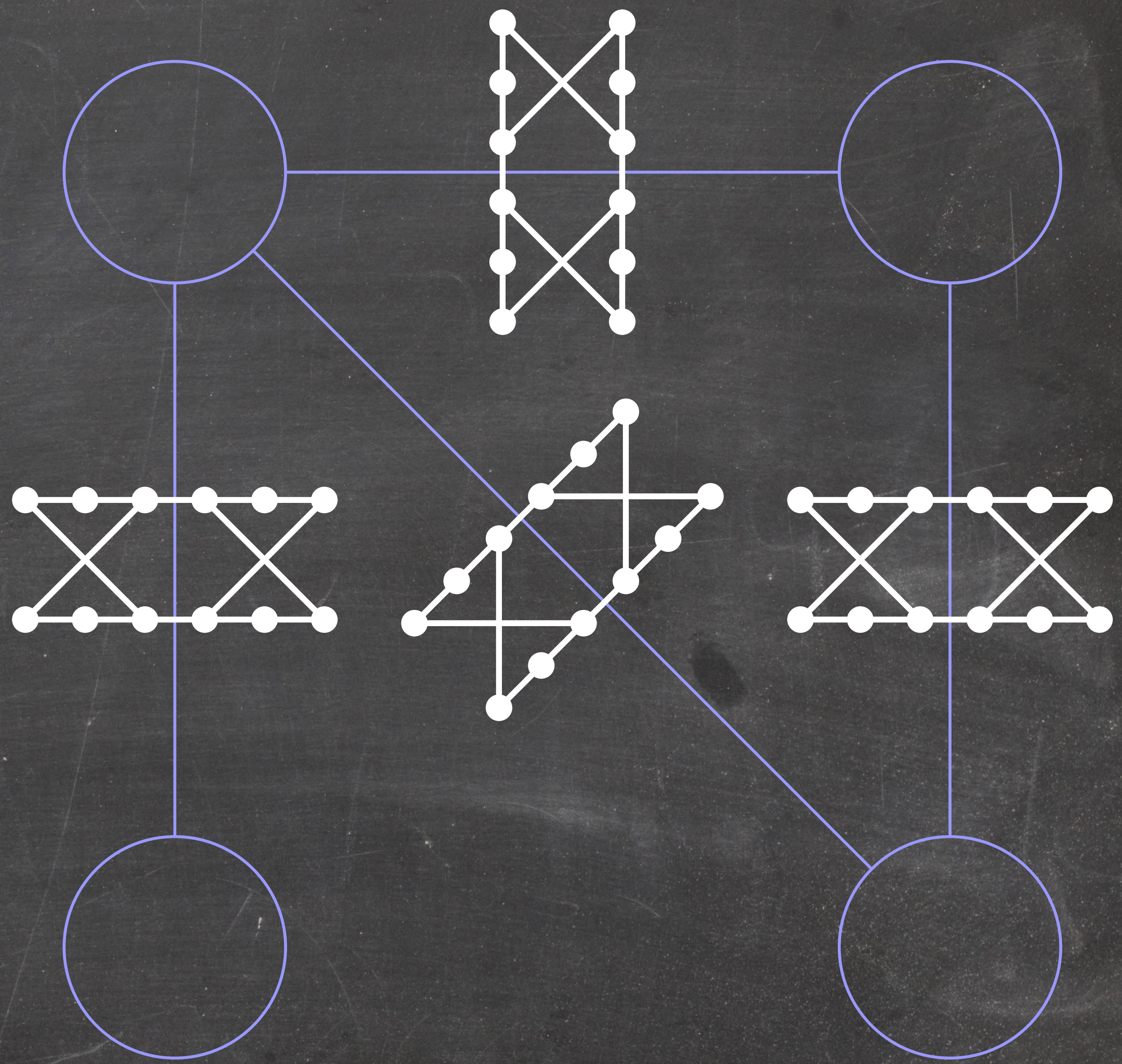


G'

Hamiltonian Cycle is NP-Complete

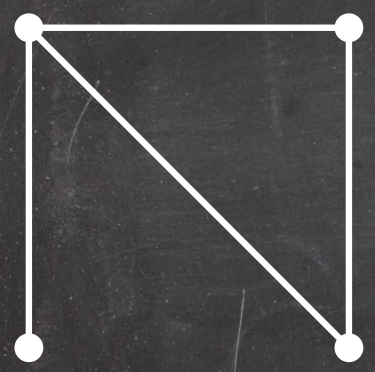


G
k = 2

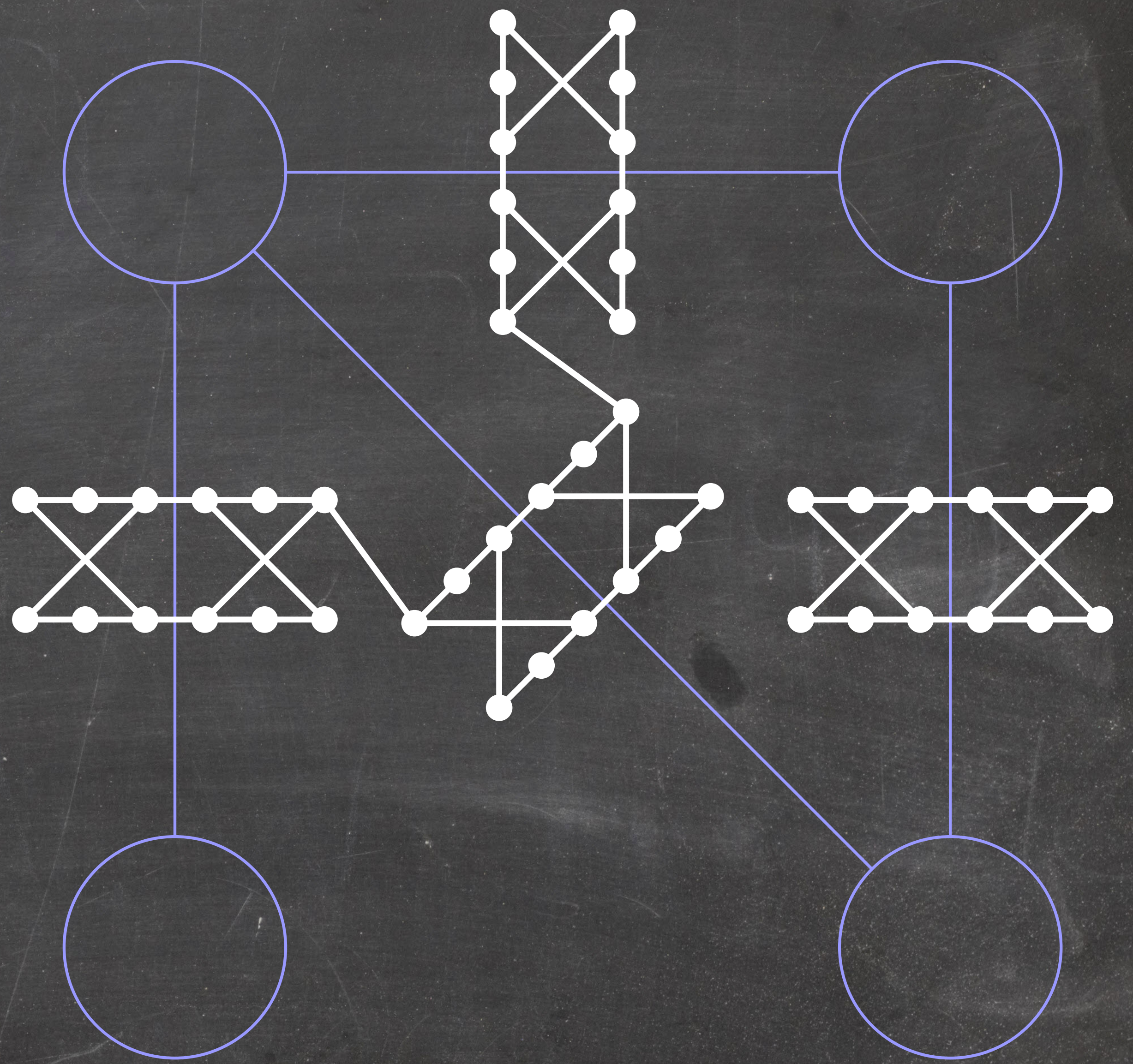


G'

Hamiltonian Cycle is NP-Complete

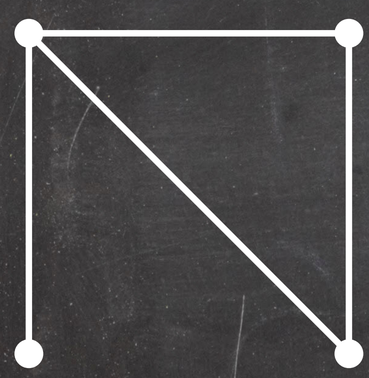


G
$k = 2$

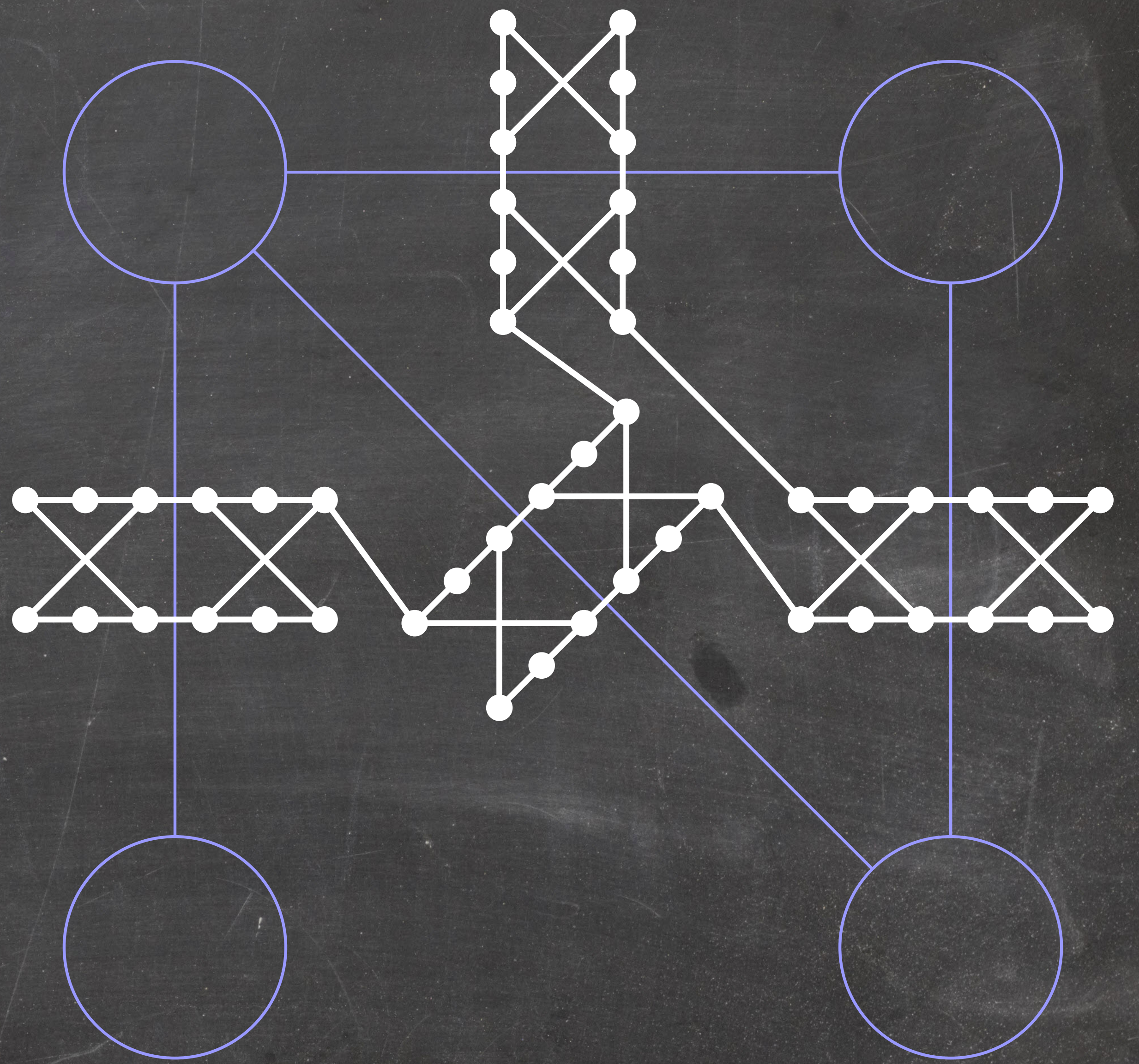


G'

Hamiltonian Cycle is NP-Complete

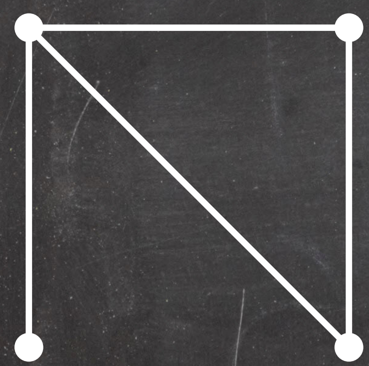


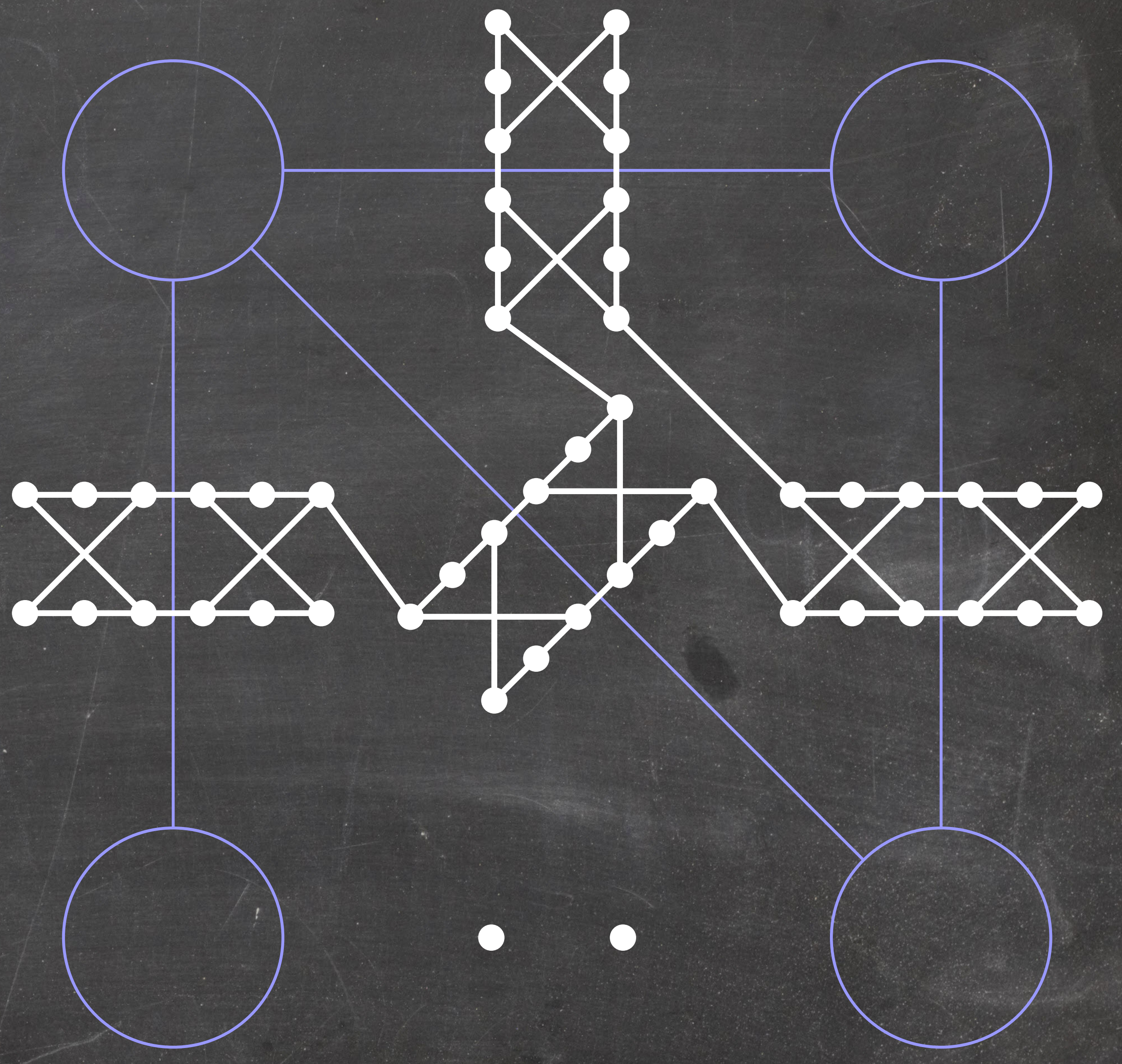
G
k = 2



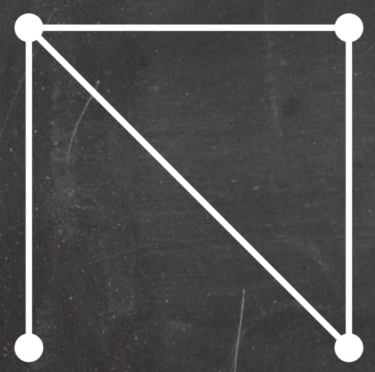
G′

Hamiltonian Cycle is NP-Complete



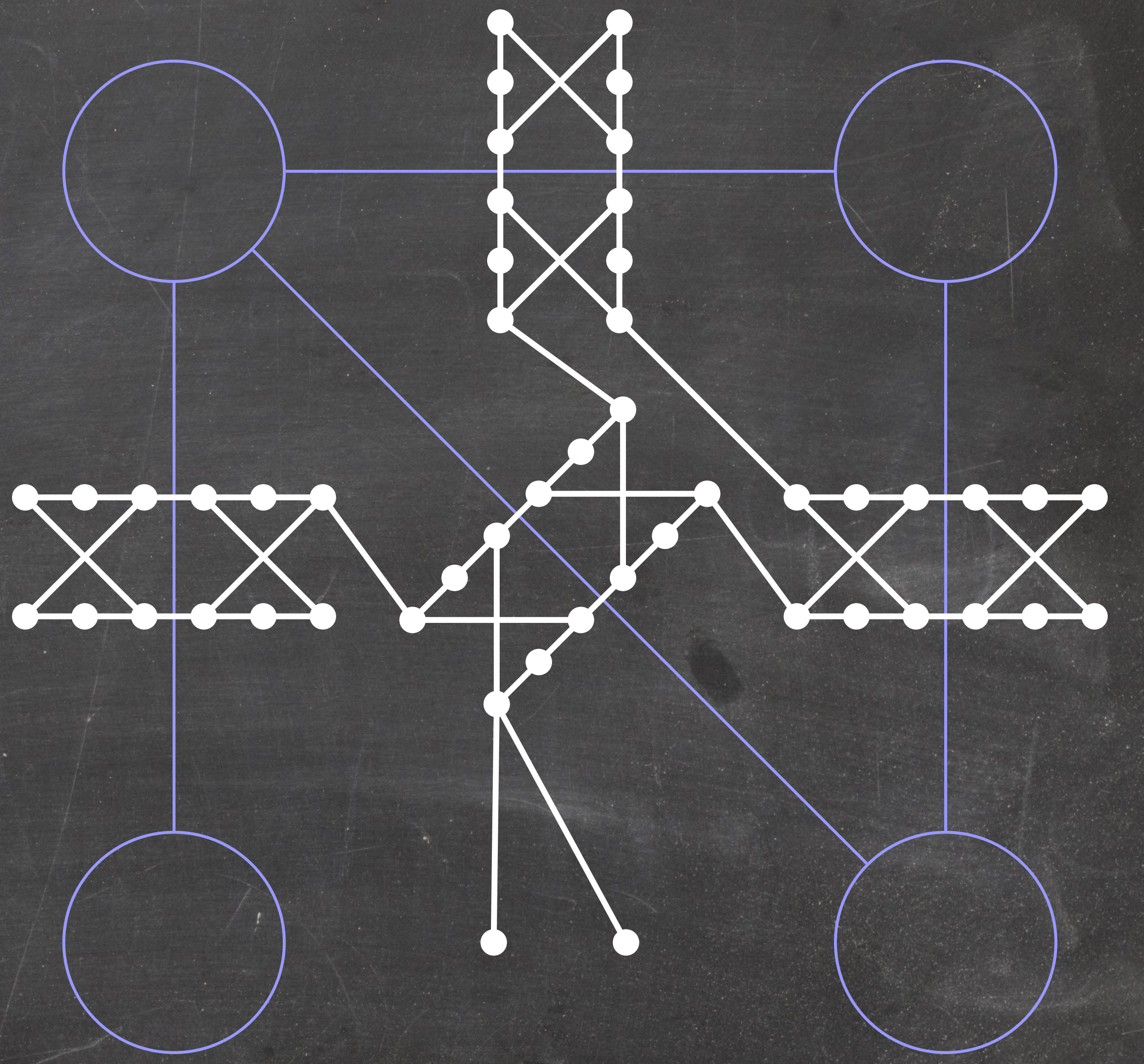


Hamiltonian Cycle is NP-Complete



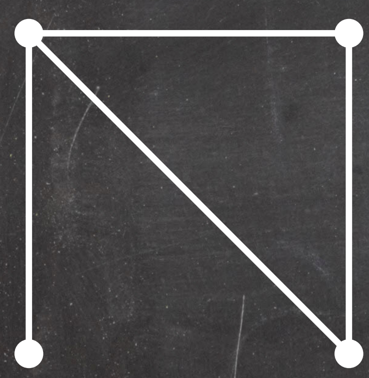
G

k = 2

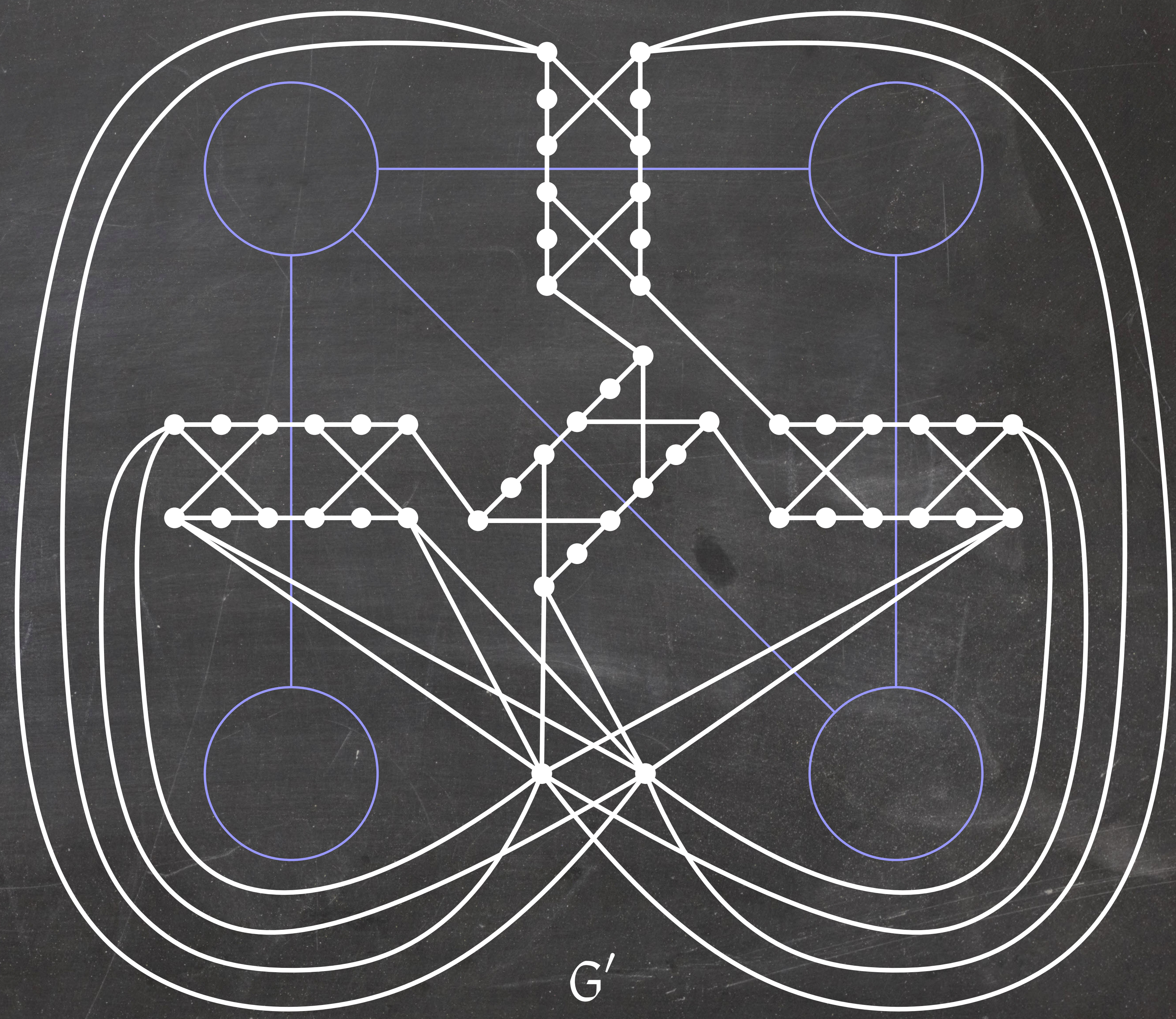


G'

Hamiltonian Cycle is NP-Complete

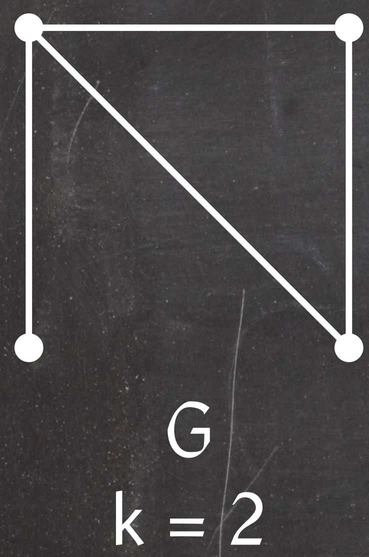


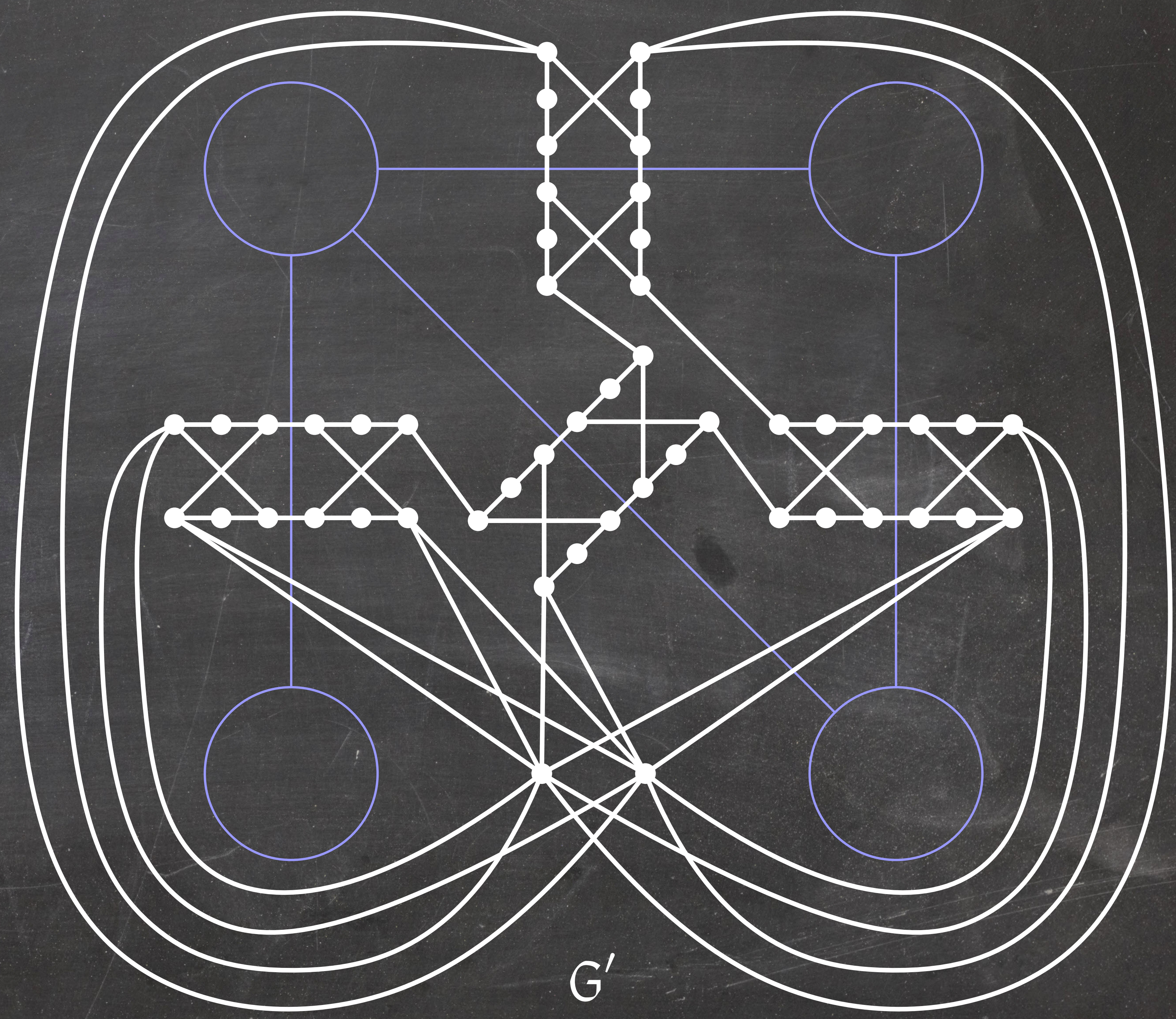
G
k = 2



G'

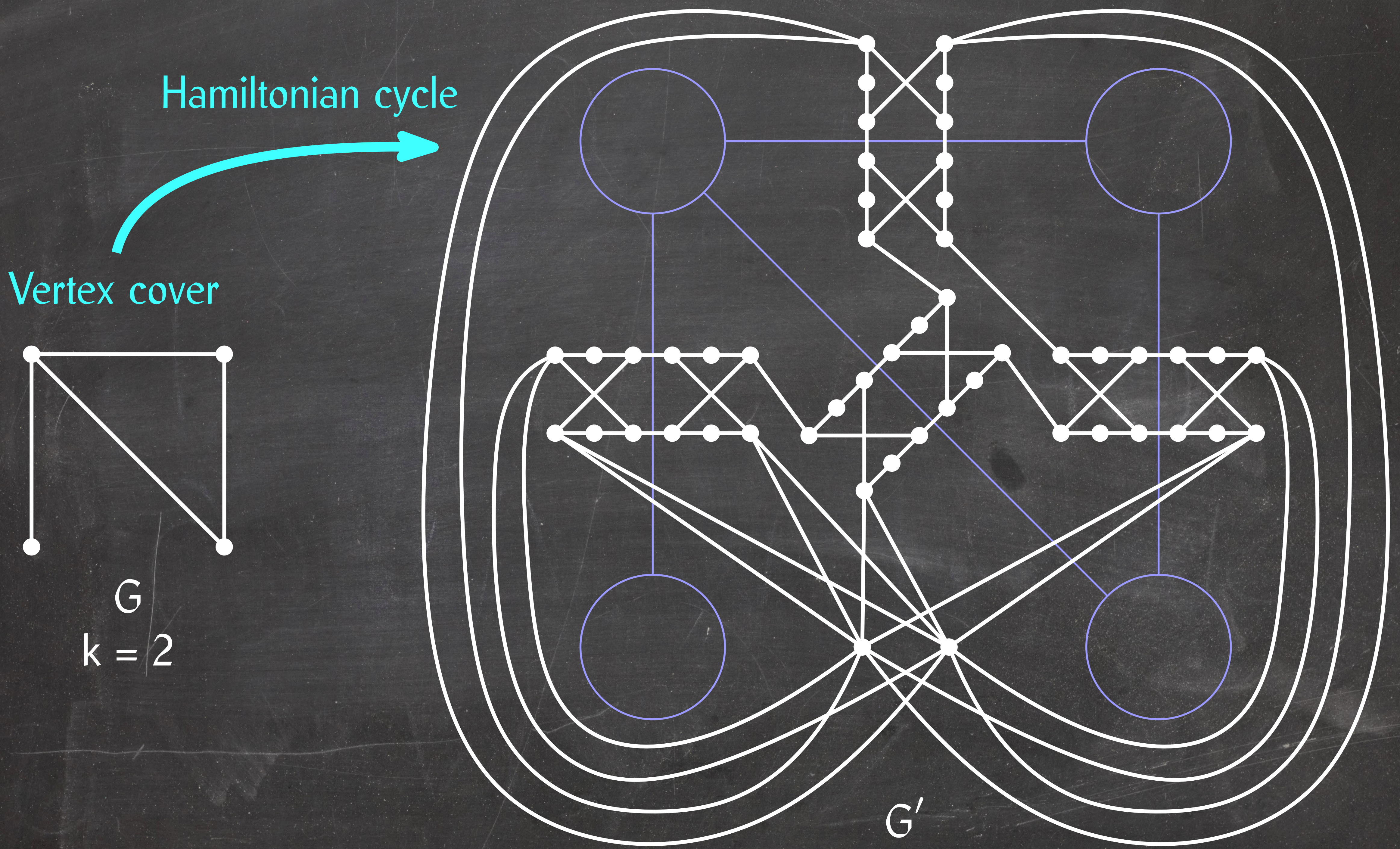
Hamiltonian Cycle is NP-Complete





Lemma: The graph G' has a Hamiltonian cycle if and only if G has a vertex cover of size k.

Hamiltonian Cycle is NP-Complete

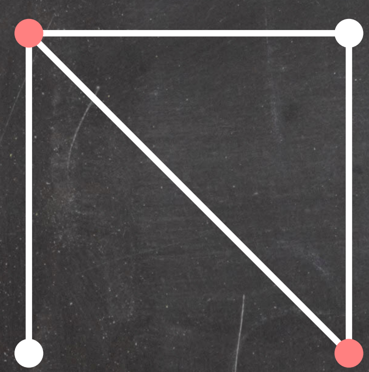


Lemma: The graph G' has a Hamiltonian cycle if and only if G has a vertex cover of size k.

Hamiltonian Cycle is NP-Complete

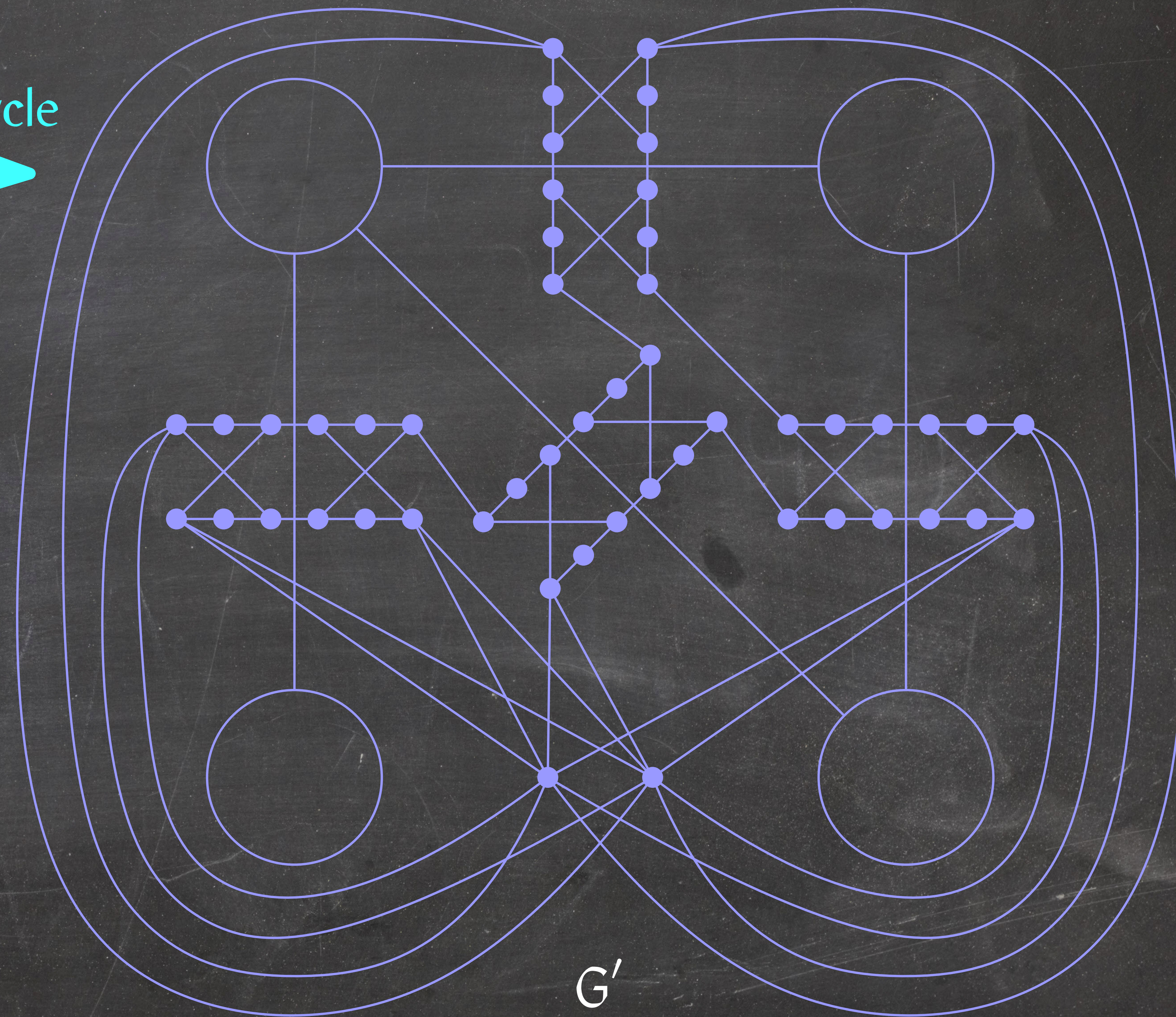
Hamiltonian cycle

Vertex cover



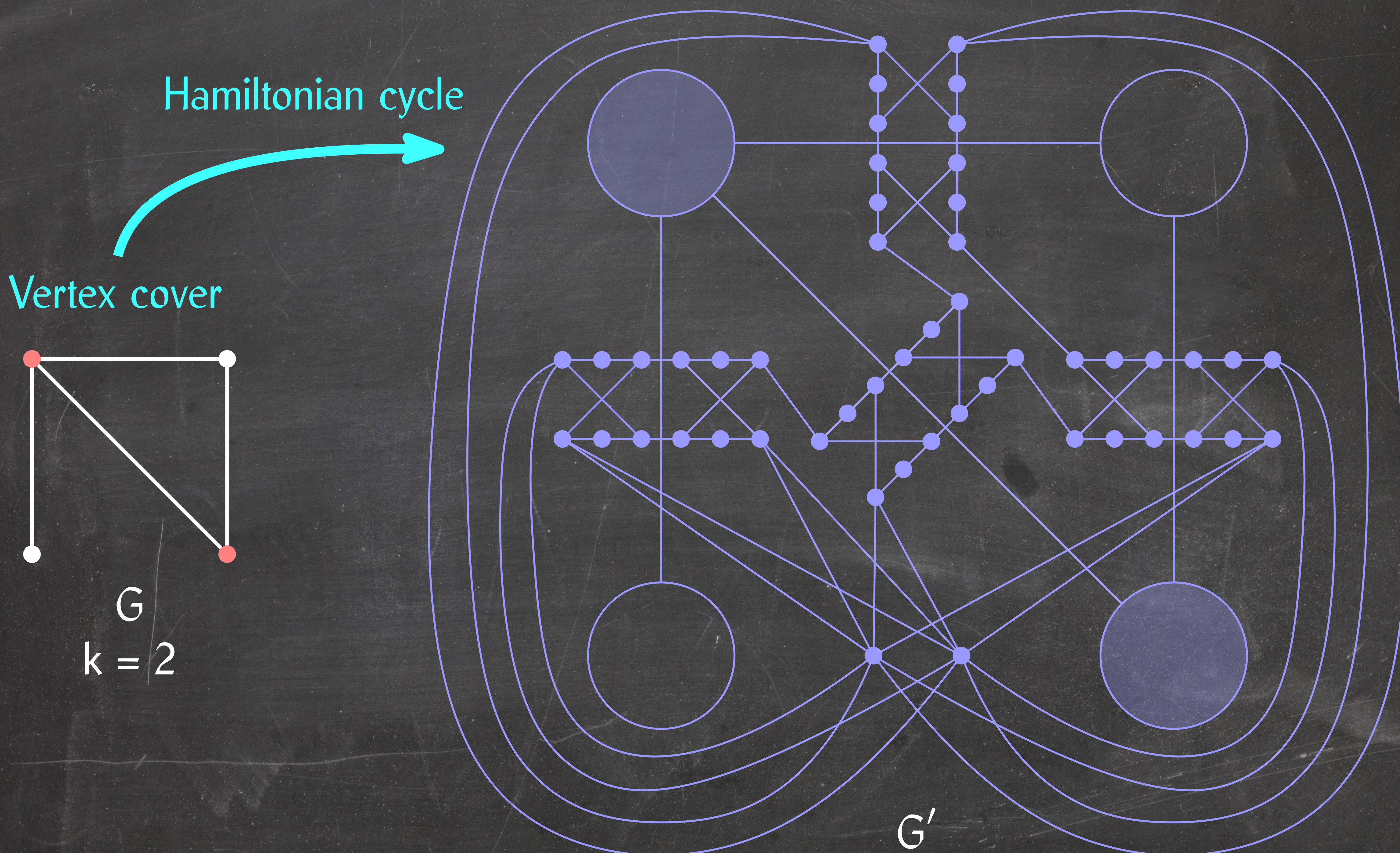
G

k = 2



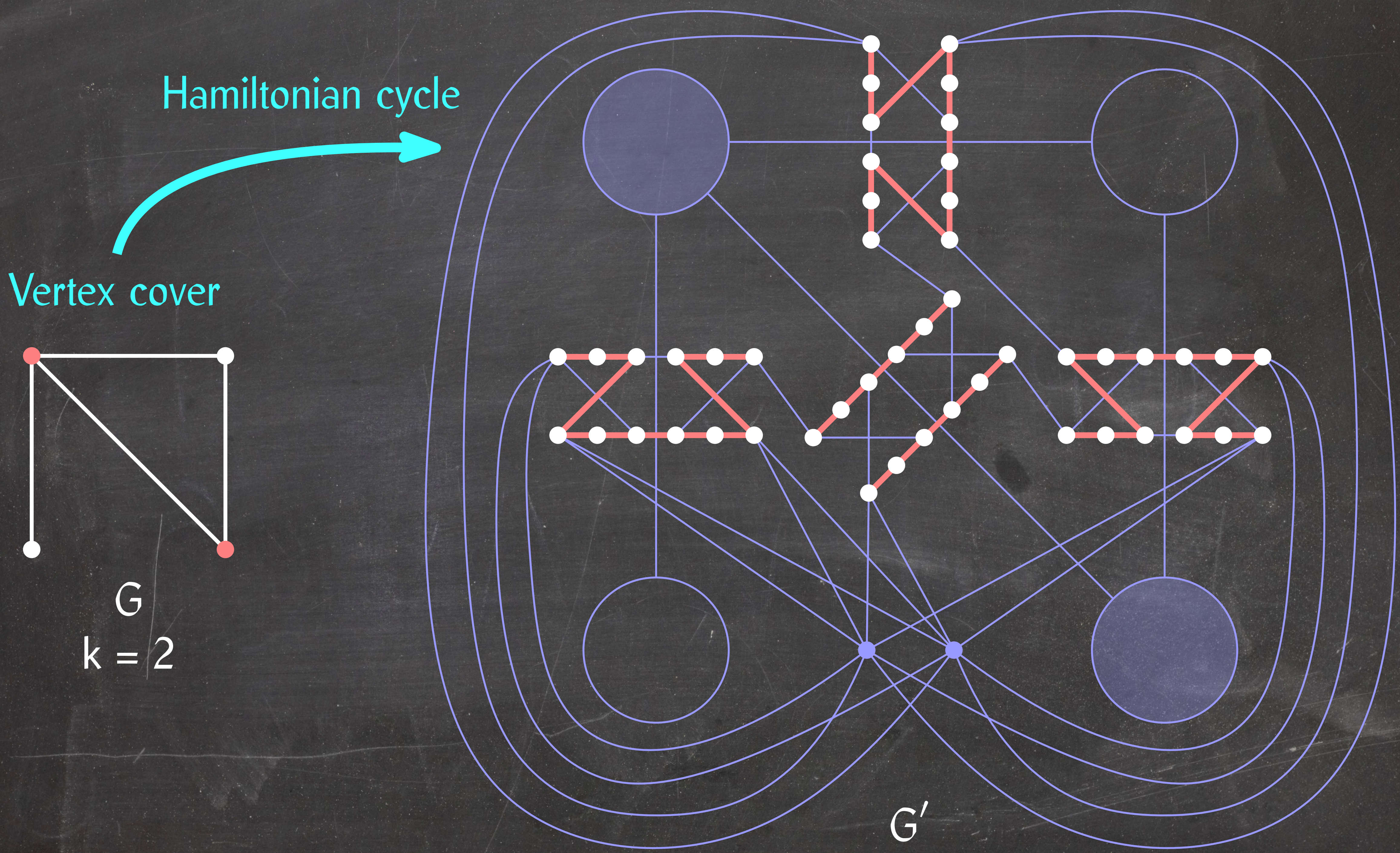
Lemma: The graph G' has a Hamiltonian cycle if and only if G has a vertex cover of size k.

Hamiltonian Cycle is NP-Complete



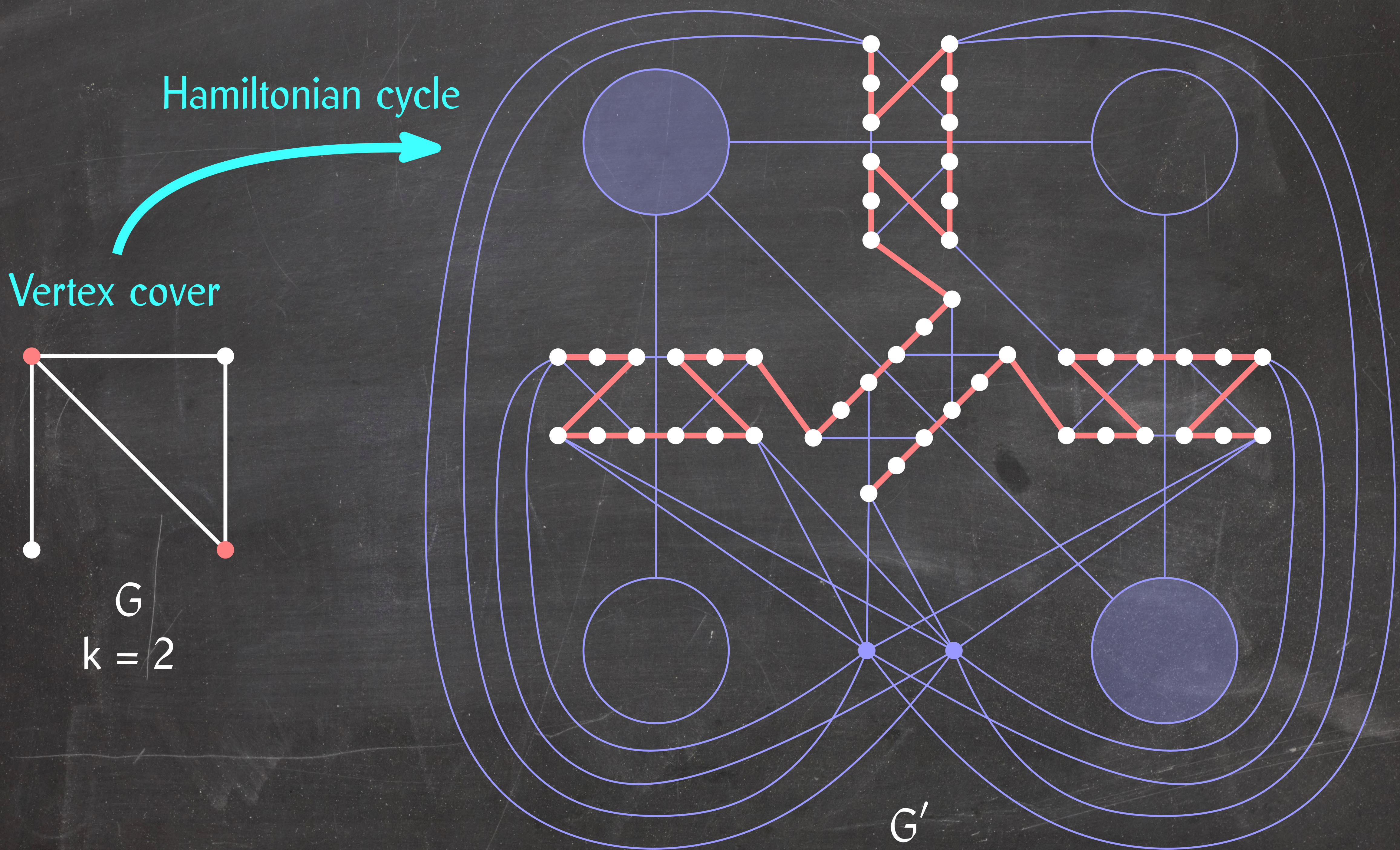
Lemma: The graph G' has a Hamiltonian cycle if and only if G has a vertex cover of size k.

Hamiltonian Cycle is NP-Complete



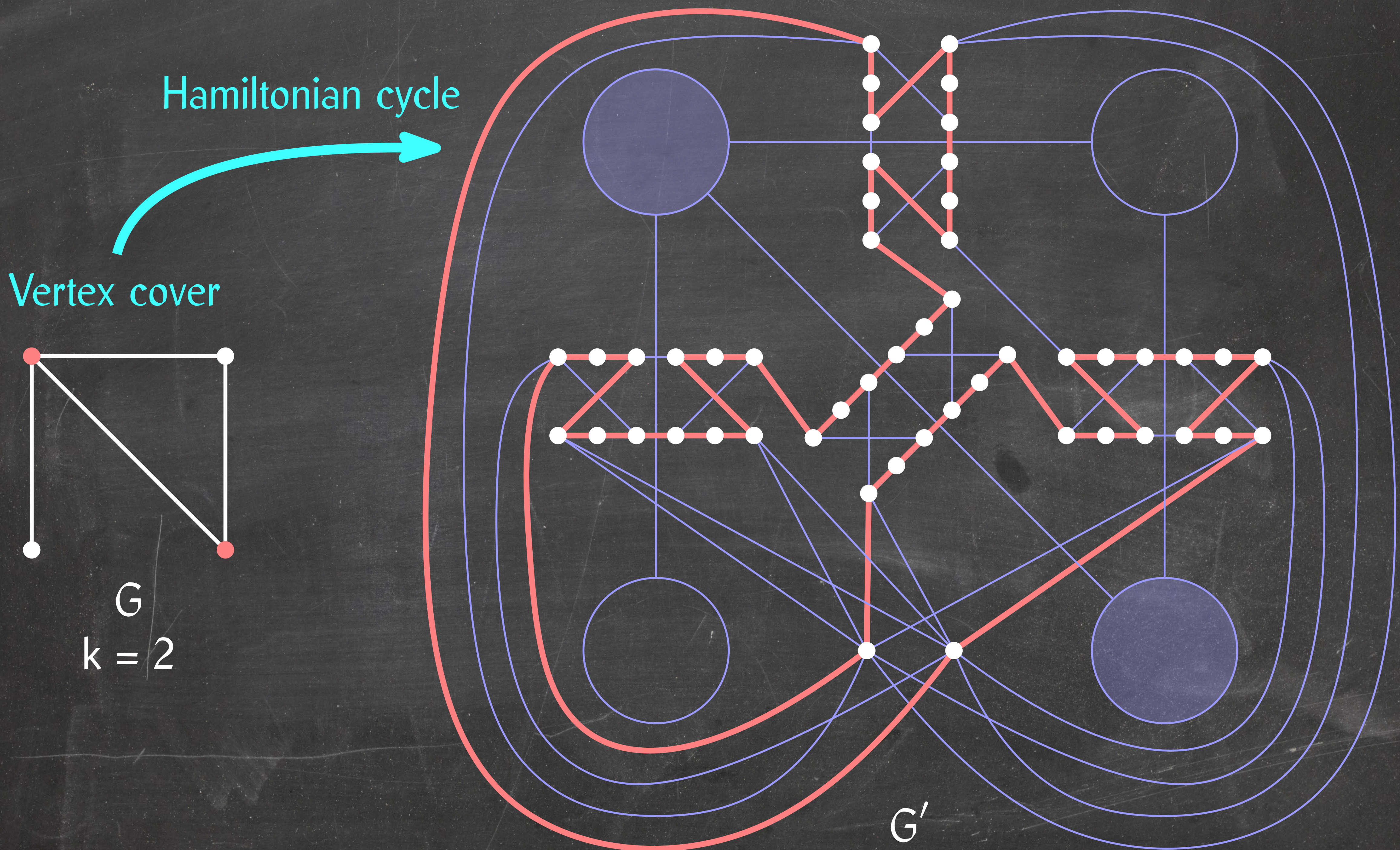
Lemma: The graph G' has a Hamiltonian cycle if and only if G has a vertex cover of size k.

Hamiltonian Cycle is NP-Complete



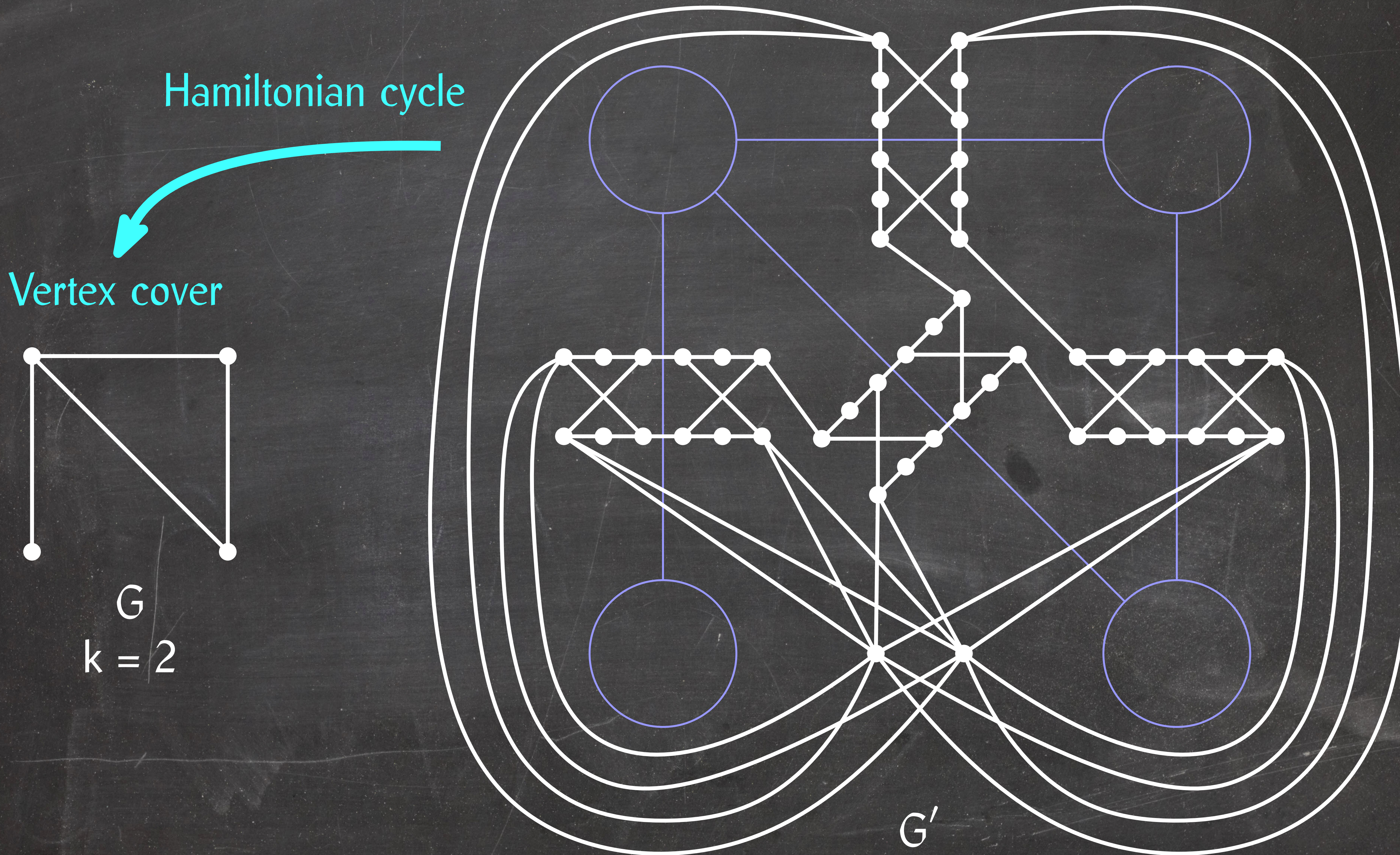
Lemma: The graph G' has a Hamiltonian cycle if and only if G has a vertex cover of size k.

Hamiltonian Cycle is NP-Complete



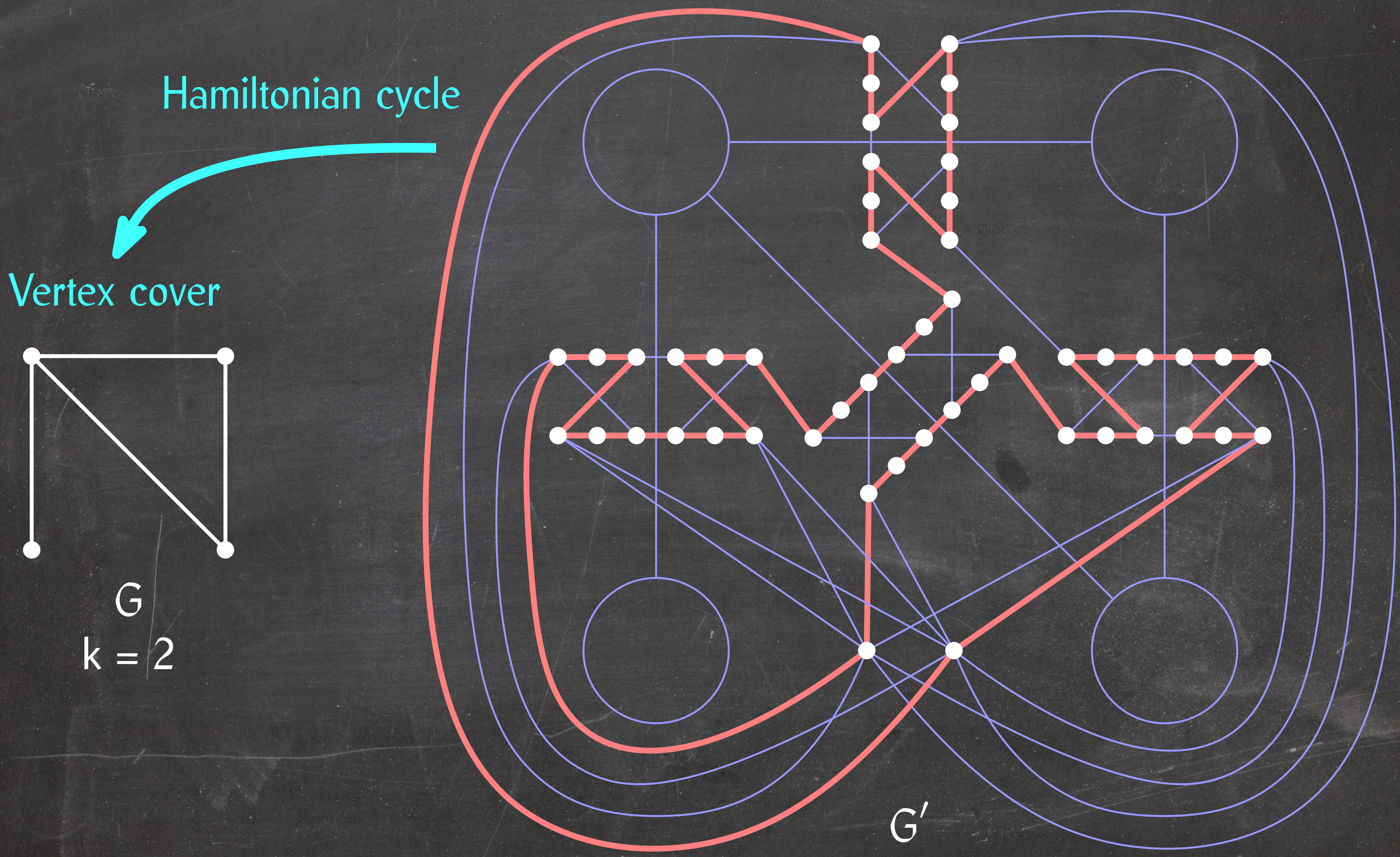
Lemma: The graph G' has a Hamiltonian cycle if and only if G has a vertex cover of size k.

Hamiltonian Cycle is NP-Complete



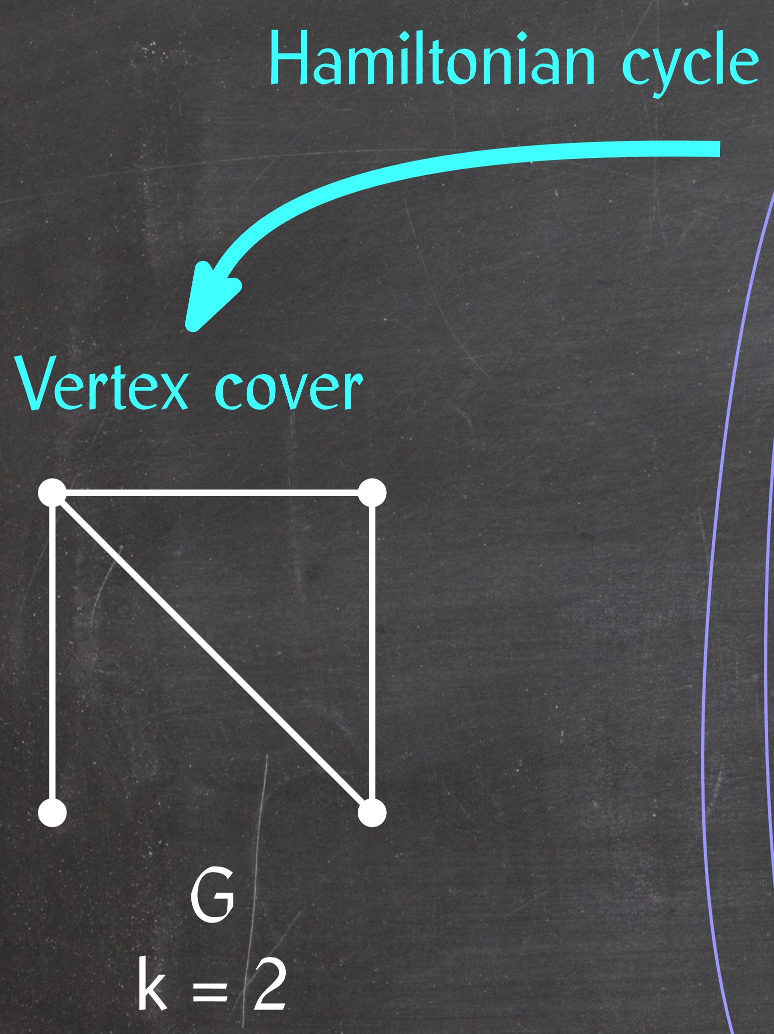
Lemma: The graph G' has a Hamiltonian cycle if and only if G has a vertex cover of size k.

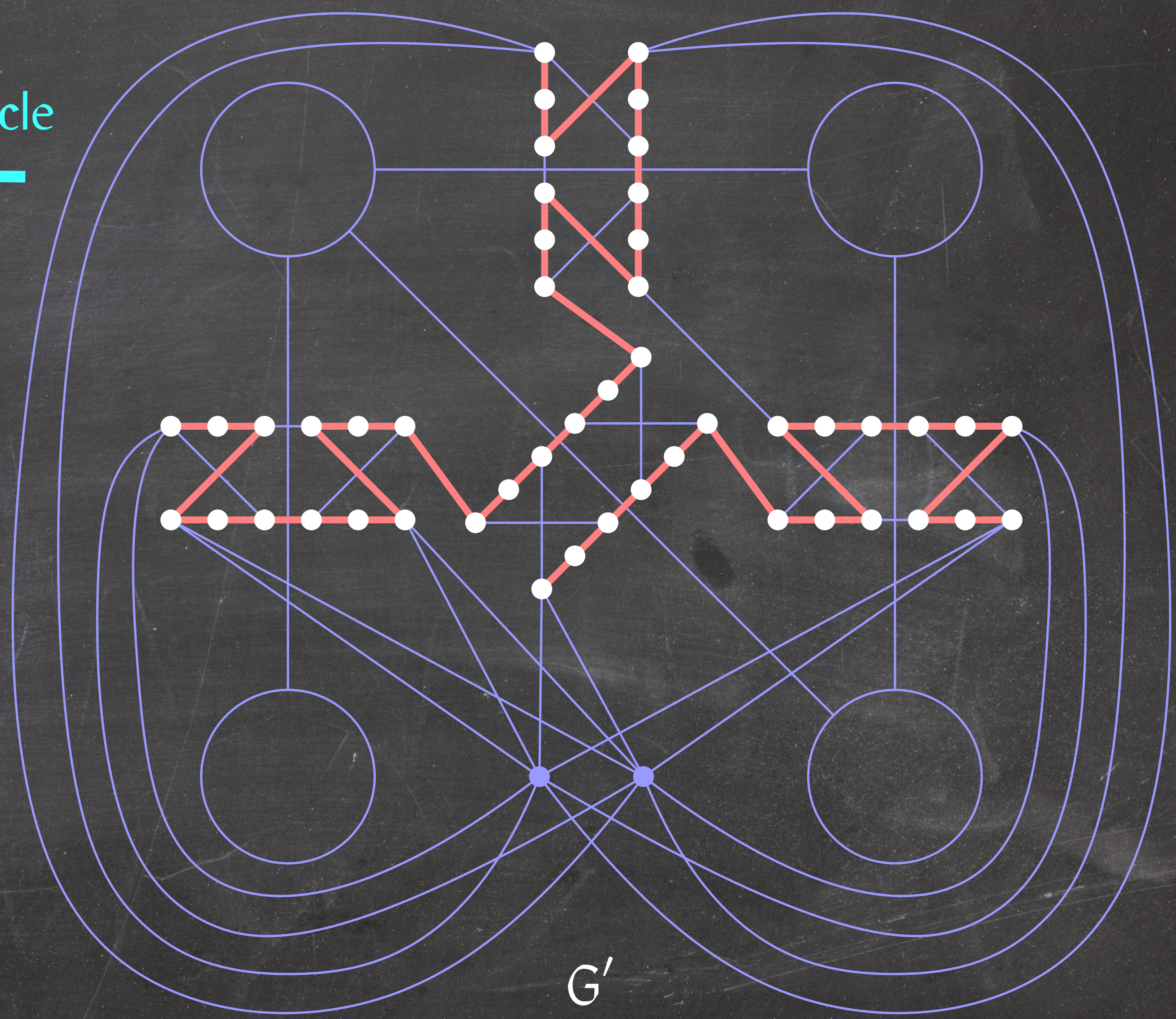
Hamiltonian Cycle is NP-Complete



Lemma: The graph G' has a Hamiltonian cycle if and only if G has a vertex cover of size k.

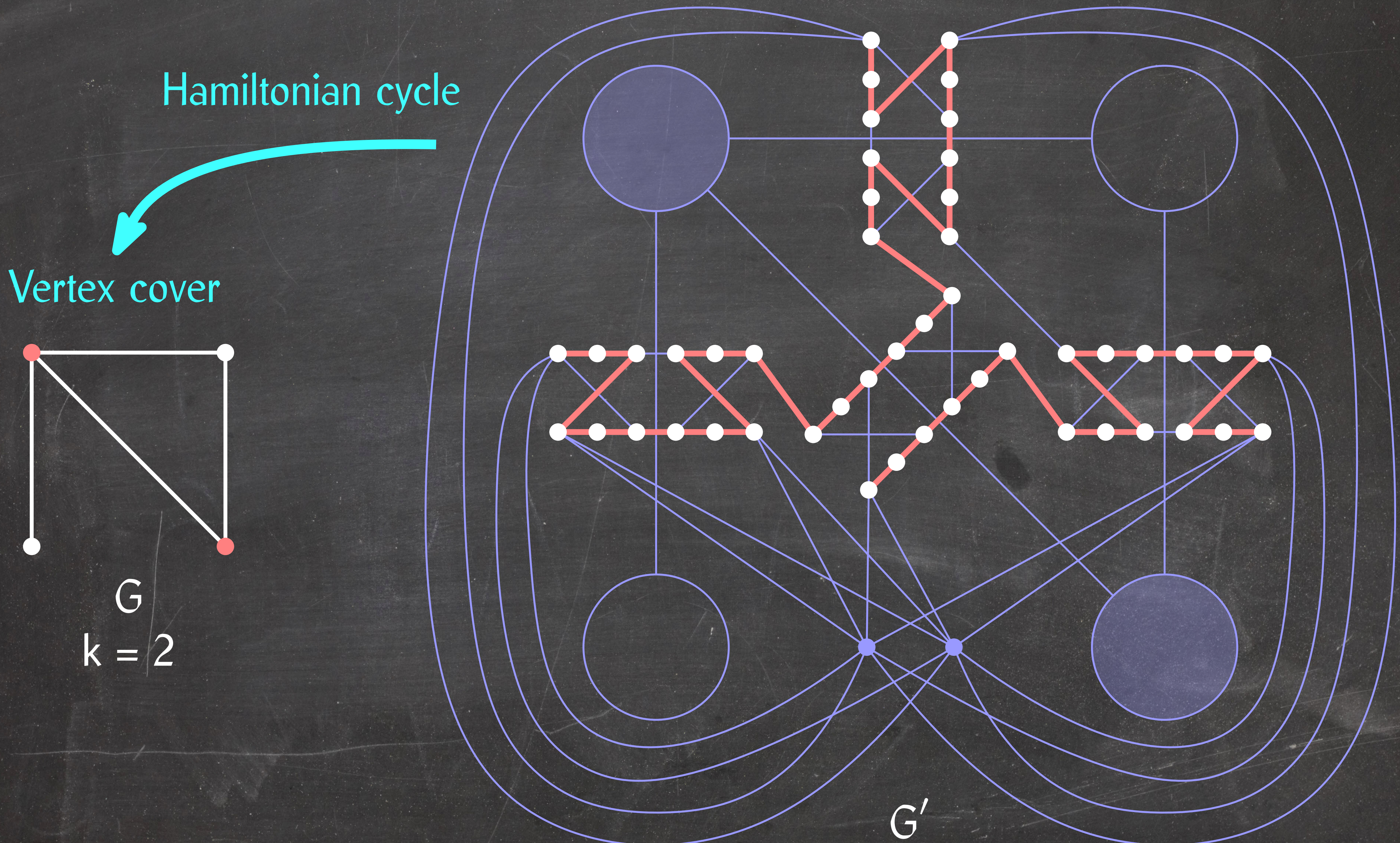
Hamiltonian Cycle is NP-Complete





Lemma: The graph G' has a Hamiltonian cycle if and only if G has a vertex cover of size k.

Hamiltonian Cycle is NP-Complete



Lemma: The graph G' has a Hamiltonian cycle if and only if G has a vertex cover of size k.

Subset Sum

Given:

- A set $S = \{x_1, x_2, \ldots, x_n\}$ of distinct numbers
- A parameter t

Question:

Is there a subset $S' \subseteq S$ such that $\sum_{x \in S'} x = t$?

Subset Sum

Given:

- A set $S = \{x_1, x_2, \ldots, x_n\}$ of distinct numbers
- A parameter t

Question:

Is there a subset $S' \subseteq S$ such that $\sum_{x \in S'} x = t$?

Example:

$$S = \{1, 2, 8, 13\}$$

S has a subset S' whose elements sum to 22, namely $S' = \{1, 8, 13\}$, but there is no subset whose elements sum to 12.

Subset Sum is NP-Complete

Exercise: Verify that Subset Sum is in NP.

Subset Sum is NP-Complete

Exercise: Verify that Subset Sum is in NP.

To prove: Subset Sum is NP-hard.

Subset Sum is NP-Complete

Exercise: Verify that Subset Sum is in NP.

To prove: Subset Sum is NP-hard.

Reduction from 3-SAT:

Given a formula F in 3-CNF, we construct a set S_F of $2n + 2m$ numbers with $n + m$ digits in base-10 notation and a number $t = \sum_{i=0}^{n-1} 10^{i+m} + \sum_{i=0}^{m-1} 4 \cdot 10^i$:

$$\underbrace{11 \cdots 1}_{\text{n variable digits}} \quad \underbrace{44 \cdots 4}_{\text{m clause digits}}$$

Subset Sum is NP-Complete

Exercise: Verify that Subset Sum is in NP.

To prove: Subset Sum is NP-hard.

Reduction from 3-SAT:

Given a formula F in 3-CNF, we construct a set S_F of $2n + 2m$ numbers with $n + m$ digits in base-10 notation and a number $t = \sum_{i=0}^{n-1} 10^{i+m} + \sum_{i=0}^{m-1} 4 \cdot 10^i$:

$$\underbrace{11 \cdots 1}_{n \text{ variable digits}} \quad \underbrace{44 \cdots 4}_{m \text{ clause digits}}$$

There will be a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$ if and only if F is satisfiable.

Subset Sum is NP-Complete

$$F = (x_1 \lor x_2 \lor \bar{x}_3) \land (\bar{x}_1 \lor x_3 \lor x_4) \land (\bar{x}_2 \lor x_3 \lor \bar{x}_4) \land (x_1 \lor \bar{x}_3 \lor x_4)$$

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$$\text{Literal numbers} \left\{ \begin{array}{l} x_1 \\ \bar{x}_1 \\ x_2 \\ \bar{x}_2 \\ x_3 \\ \bar{x}_3 \\ x_4 \\ \bar{x}_4 \end{array} \right.$$

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$$\text{Literal numbers} \left\{ \begin{array}{l} x_1 \\ \bar{x}_1 \\ x_2 \\ \bar{x}_2 \\ x_3 \\ \bar{x}_3 \\ x_4 \\ \bar{x}_4 \end{array} \right.$$

$$\text{Slack numbers} \left\{ \begin{array}{l} s_1 \\ s_1' \\ s_2 \\ s_2' \\ s_3 \\ s_3' \\ s_4 \\ s_4' \end{array} \right.$$

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$$\text{Literal numbers} \left\{ \begin{array}{l} x_1 \\ \bar{x}_1 \\ x_2 \\ \bar{x}_2 \\ x_3 \\ \bar{x}_3 \\ x_4 \\ \bar{x}_4 \end{array} \right.$$

$$\text{Slack numbers} \left\{ \begin{array}{l} s_1 \\ s_1' \\ s_2 \\ s_2' \\ s_3 \\ s_3' \\ s_4 \\ s_4' \end{array} \right.$$

t	1	1	1	1	4	4	4	4

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

Literal numbers	x_1	1							
	$\bar{x}_1$	1							
	x_2								
	$\bar{x}_2$								
	x_3								
	$\bar{x}_3$								
	x_4								
	$\bar{x}_4$								
Slack numbers	s_1								
	s_1'								
	s_2								
	s_2'								
	s_3								
	s_3'								
	s_4								
	s_4'								
	t	1	1	1	1	4	4	4	4

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

Literal numbers	x_1	1							
	$\bar{x}_1$	1							
	x_2		1						
	$\bar{x}_2$		1						
	x_3			1					
	$\bar{x}_3$			1					
	x_4				1				
	$\bar{x}_4$				1				
Slack numbers	s_1								
	s_1'								
	s_2								
	s_2'								
	s_3								
	s_3'								
	s_4								
	s_4'								
	t	1	1	1	1	4	4	4	4

Subset Sum is NP-Complete

$$F = (x_1 \lor x_2 \lor \bar{x}_3) \land (\bar{x}_1 \lor x_3 \lor x_4) \land (\bar{x}_2 \lor x_3 \lor \bar{x}_4) \land (x_1 \lor \bar{x}_3 \lor x_4)$$

Literal numbers	x_1	1				1			1
	$\bar{x}_1$	1							
	x_2		1						
	$\bar{x}_2$		1						
	x_3			1					
	$\bar{x}_3$			1					
	x_4				1				
	$\bar{x}_4$				1				
Slack numbers	s_1								
	s_1'								
	s_2								
	s_2'								
	s_3								
	s_3'								
	s_4								
	s_4'								
	t	1	1	1	1	4	4	4	4

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

Literal numbers	x_1	1				1			1
	$\bar{x}_1$	1					1		
	x_2		1						
	$\bar{x}_2$		1						
	x_3			1					
	$\bar{x}_3$			1					
	x_4				1				
	$\bar{x}_4$				1				
Slack numbers	s_1								
	s_1'								
	s_2								
	s_2'								
	s_3								
	s_3'								
	s_4								
	s_4'								
	t	1	1	1	1	4	4	4	4

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

Literal numbers	x_1	1				1			1
	$\bar{x}_1$	1					1		
	x_2		1			1			
	$\bar{x}_2$		1					1	
	x_3			1			1	1	
	$\bar{x}_3$			1		1			1
	x_4				1		1		1
	$\bar{x}_4$				1			1	
Slack numbers	s_1								
	s_1'								
	s_2								
	s_2'								
	s_3								
	s_3'								
	s_4								
	s_4'								
	t	1	1	1	1	4	4	4	4

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

Literal numbers	x_1	1				1			1
	$\bar{x}_1$	1					1		
	x_2		1			1			
	$\bar{x}_2$		1					1	
	x_3			1			1	1	
	$\bar{x}_3$			1		1			1
	x_4				1		1		1
	$\bar{x}_4$				1			1	
Slack numbers	s_1					1			
	s_1'					2			
	s_2								
	s_2'								
	s_3								
	s_3'								
	s_4								
	s_4'								
	t	1	1	1	1	4	4	4	4

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

Literal numbers	x_1	1				1			1
	$\bar{x}_1$	1					1		
	x_2		1			1			
	$\bar{x}_2$		1					1	
	x_3			1			1	1	
	$\bar{x}_3$			1		1			1
	x_4				1		1		1
	$\bar{x}_4$				1			1	
Slack numbers	s_1					1			
	s_1'					2			
	s_2						1		
	s_2'						2		
	s_3							1	
	s_3'							2	
	s_4								1
	s_4'								2
	t	1	1	1	1	4	4	4	4

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$$\begin{array}{lll}
\text{Literal numbers} & \left\{ \begin{array}{l} x_1 \\ \bar{x}_1 \\ x_2 \\ \bar{x}_2 \\ x_3 \\ \bar{x}_3 \\ x_4 \\ \bar{x}_4 \end{array} \right. & \begin{array}{cccc cccc} 1&0&0&0&1&0&0&1 \\ 1&0&0&0&0&1&0&0 \\ 0&1&0&0&1&0&0&0 \\ 0&1&0&0&0&0&1&0 \\ 0&0&1&0&0&1&1&0 \\ 0&0&1&0&1&0&0&1 \\ 0&0&0&1&0&1&0&1 \\ 0&0&0&1&0&0&1&0 \end{array} \\
\text{Slack numbers} & \left\{ \begin{array}{l} s_1 \\ s'_1 \\ s_2 \\ s'_2 \\ s_3 \\ s'_3 \\ s_4 \\ s'_4 \end{array} \right. & \begin{array}{cccc cccc} 0&0&0&0&1&0&0&0 \\ 0&0&0&0&2&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&2&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&2&0 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&2 \end{array} \\
\hline
 & t & \begin{array}{cccc cccc} 1&1&1&1&4&4&4&4 \end{array}
\end{array}$$

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

Literal numbers	x_1	1	0	0	0	1	0	0	1
	$\bar{x}_1$	1	0	0	0	0	1	0	0
	x_2	0	1	0	0	1	0	0	0
	$\bar{x}_2$	0	1	0	0	0	0	1	0
	x_3	0	0	1	0	0	1	1	0
	$\bar{x}_3$	0	0	1	0	1	0	0	1
	x_4	0	0	0	1	0	1	0	1
	$\bar{x}_4$	0	0	0	1	0	0	1	0
Slack numbers	s_1	0	0	0	0	1	0	0	0
	s_1'	0	0	0	0	2	0	0	0
	s_2	0	0	0	0	0	1	0	0
	s_2'	0	0	0	0	0	2	0	0
	s_3	0	0	0	0	0	0	1	0
	s_3'	0	0	0	0	0	0	2	0
	s_4	0	0	0	0	0	0	0	1
	s_4'	0	0	0	0	0	0	0	2
	t	1	1	1	1	4	4	4	4

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

Truth assignment

$\downarrow$

Subset S'

Literal numbers	x_1	1 0 0 0	1 0 0 1
	$\bar{x}_1$	1 0 0 0	0 1 0 0
	x_2	0 1 0 0	1 0 0 0
	$\bar{x}_2$	0 1 0 0	0 0 1 0
	x_3	0 0 1 0	0 1 1 0
	$\bar{x}_3$	0 0 1 0	1 0 0 1
	x_4	0 0 0 1	0 1 0 1
	$\bar{x}_4$	0 0 0 1	0 0 1 0
Slack numbers	s_1	0 0 0 0	1 0 0 0
	s_1'	0 0 0 0	2 0 0 0
	s_2	0 0 0 0	0 1 0 0
	s_2'	0 0 0 0	0 2 0 0
	s_3	0 0 0 0	0 0 1 0
	s_3'	0 0 0 0	0 0 2 0
	s_4	0 0 0 0	0 0 0 1
	s_4'	0 0 0 0	0 0 0 2
	t	1 1 1 1	4 4 4 4

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$x_1 = x_2 = x_3 = x_4 = \text{true}$

Truth assignment

Subset S'

Literal numbers	x_1	1				1			1
	$\bar{x}_1$	1					1		
	x_2		1			1			
	$\bar{x}_2$		1					1	
	x_3			1			1	1	
	$\bar{x}_3$			1		1			1
	x_4				1		1		1
	$\bar{x}_4$				1			1	
Slack numbers									
	t	1	1	1	1	4	4	4	4

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$x_1 = x_2 = x_3 = x_4 = \text{true}$

Truth assignment

$\downarrow$

Subset S'

Literal numbers	x_1	1				1			1
	x_2		1			1			
	$\bar{x}_2$		1					1	
	x_3			1			1	1	
	$\bar{x}_3$			1		1			1
	x_4				1		1		1
	$\bar{x}_4$				1			1	
Slack numbers									
	t	1	1	1	1	4	4	4	4

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$x_1 = x_2 = x_3 = x_4 = \text{true}$

Truth assignment

↓

Subset S'

Literal numbers	x_1	1				1			1
	x_2		1			1			
	x_3			1			1	1	
	x_4				1		1		1
Slack numbers									
	t	1	1	1	1	4	4	4	4

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$x_1 = x_2 = x_3 = x_4 =$ true

Truth assignment

↓

Subset S'

Literal numbers	x_1	1				1			1
	x_2		1			1			
	x_3			1			1	1	
	x_4				1		1		1
Slack numbers									
	t	1	1	1	1	2	2	1	2

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$x_1 = x_2 = x_3 = x_4 = \text{true}$

Truth assignment

$\downarrow$

Subset S'

Literal numbers	x_1	1				1			1
	x_2		1			1			
	x_3			1			1	1	
	x_4				1		1		1
Slack numbers	s'_1					2			
	t	1	1	1	1	4	2	1	2

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$x_1 = x_2 = x_3 = x_4 = \text{true}$

Truth assignment

Subset S'

Literal numbers	x_1	1				1			1
	x_2		1			1			
	x_3			1			1	1	
	x_4				1		1		1
Slack numbers	s_1'					2			
	s_2'						2		
	s_3							1	
	s_3'							2	
	s_4'								2
	t	1	1	1	1	4	4	4	4

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

Truth assignment

$\uparrow$

Subset S'

Literal numbers	x_1	1				1			1
	$\bar{x}_1$	1					1		
	x_2		1			1			
	$\bar{x}_2$		1					1	
	x_3			1			1	1	
	$\bar{x}_3$			1		1			1
	x_4				1		1		1
	$\bar{x}_4$				1			1	
Slack numbers	s_1					1			
	s_1'					2			
	s_2						1		
	s_2'						2		
	s_3							1	
	s_3'							2	
	s_4								1
	s_4'								2
	t	1	1	1	1	4	4	4	4

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

Truth assignment

Subset S'

Literal numbers	x_1	1				1			1
	x_2		1			1			
	x_3			1			1	1	
	x_4				1		1		1
Slack numbers	s_1'					2			
	s_2'						2		
	s_3							1	
	s_3'							2	
	s_4'								2
	t	1	1	1	1	4	4	4	4

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$x_1 = x_2 = x_3 = x_4 = \text{true}$

Truth assignment

Subset S'

Literal numbers	x_1	1				1			1
	x_2		1			1			
	x_3			1			1	1	
	x_4				1		1		1
Slack numbers	s_1'					2			
	s_2'						2		
	s_3							1	
	s_3'							2	
	s_4'								2
	t	1	1	1	1	4	4	4	4

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$x_1 = x_2 = x_3 = x_4 = \text{true}$

Truth assignment

↑

Subset S'

Literal numbers	x_1	1				1			1
	x_2		1			1			
	x_3			1			1	1	
	x_4				1		1		1
Slack numbers									
	t	1	1	1	1	2	2	1	2

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Subset Sum is NP-Complete

$$F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4)$$

$x_1 = x_2 = x_3 = x_4 = \text{true}$

Truth assignment

Subset S'

Literal numbers	x_1	1				1			1
	x_2		1			1			
	x_3			1			1	1	
	x_4				1		1		1
Slack numbers									
	t	1	1	1	1	2	2	1	2

Lemma: F is satisfiable if and only if there is a subset $S' \subseteq S_F$ such that $\sum_{x \in S'} x = t$.

Summary

Many important problems are NP-hard or NP-complete.

Examples:

- Satisfiability
- Vertex cover
- Subset sum
- Hamiltonian cycle
- Clique
- Independent set
- …

These problems are unlikely to be solvable in polynomial time.

Techniques to cope with NP-hardness:

- Parameterized algorithms
- Approximation algorithms
- Heuristics