A Domino Game

- Consider these dominos, can you find a permutation of them, so the text on the upper part is exactly the same as the text on the lower part?
  (You can use one domino more than once, but not put them upside down.)
  - Example:

yx	xy	xy
x	xyyxy	y

→

xy	yx	yx	xy
xyyxy	x	x	y

xyyxyxxy

- Set 1:

ab	b	aba	aa
abab	a	b	a

- Set 2:

0	101	010	0111
01	010	101	1

Algorithms for Decision Support

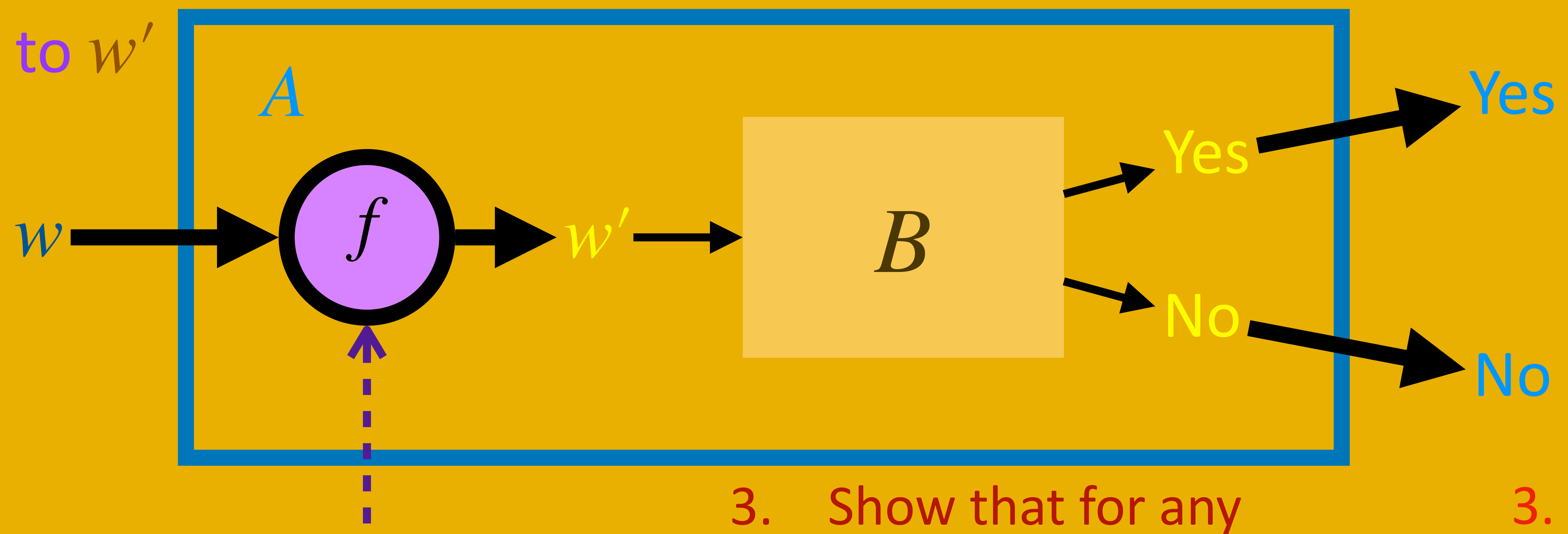
NP-Completeness (3/3)

Optimization problems

Polynomial-Time Reduce A to B

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

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



Polynomial-time function

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

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

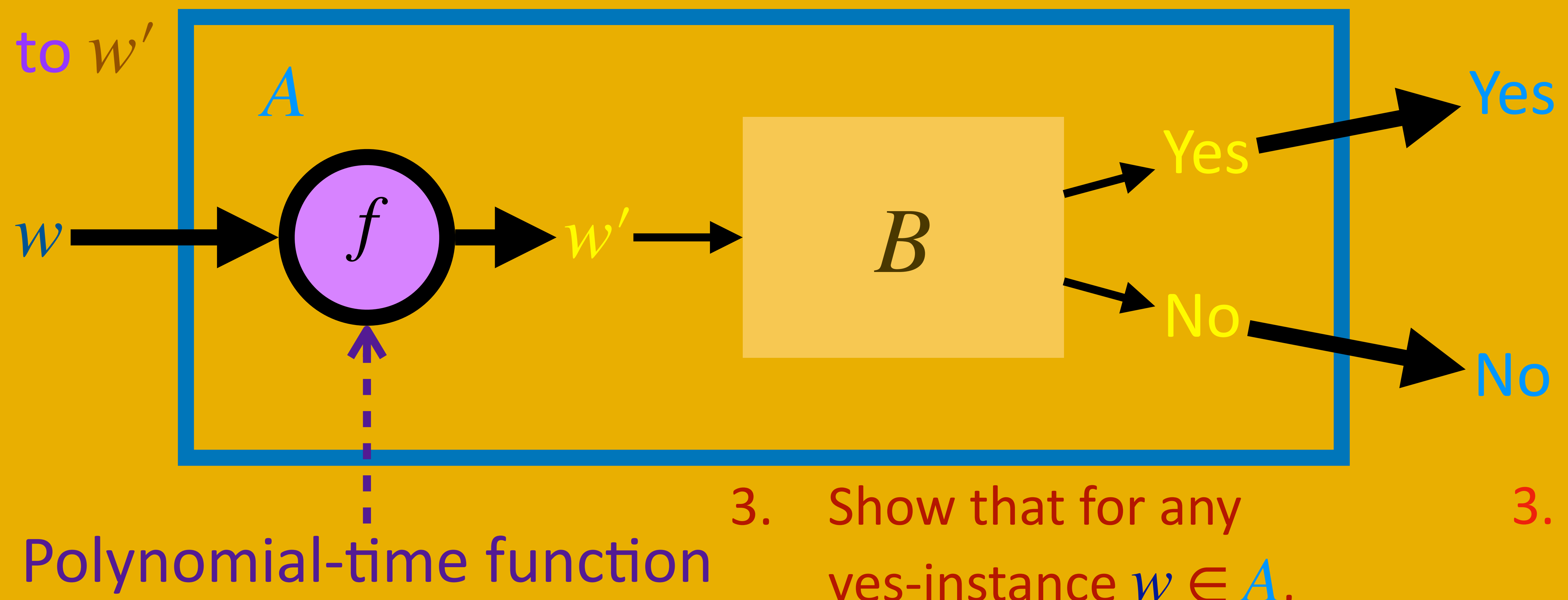
3. Show that for any no-instance $w' \notin B$, the corresponding instance w is also a no-instance of A

Instance Transformation

- Design a method to transform any instance w of A into an instance w' of B
  - The transformation should be done in polynomial time

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

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



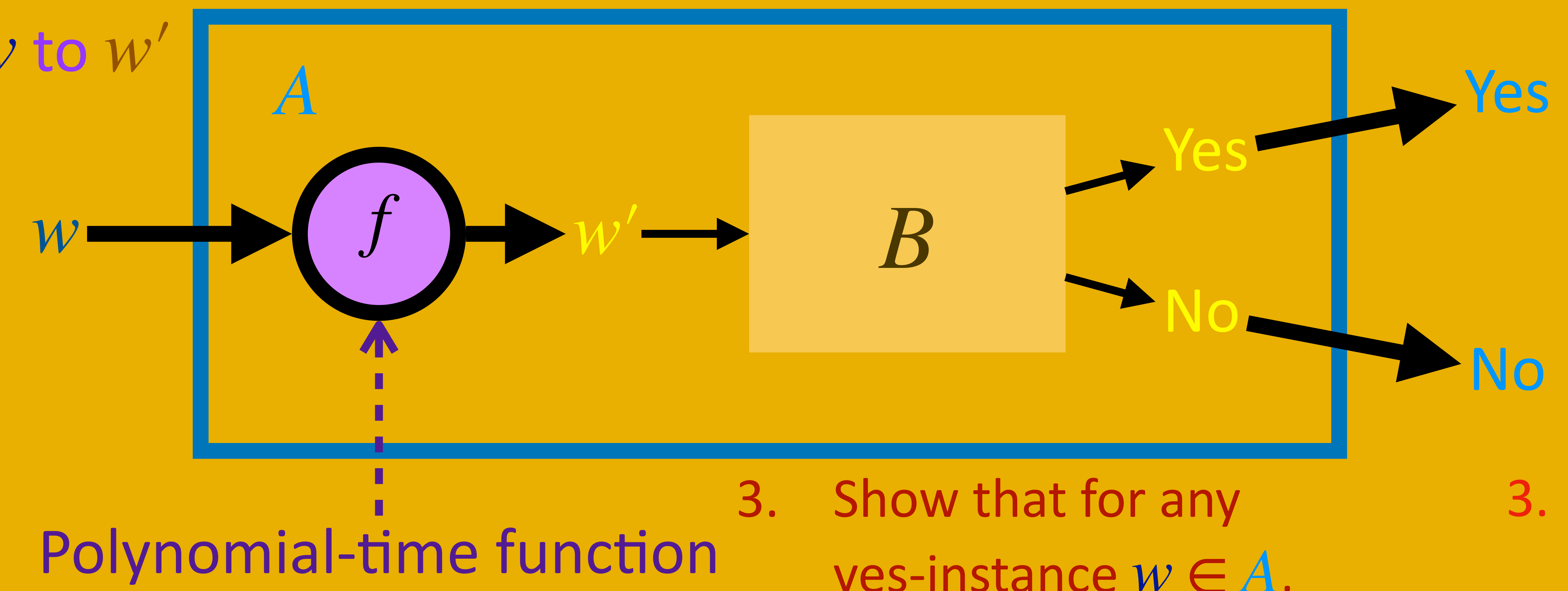
3. Show that for any yes-instance $w \in A$, the corresponding instance w' is also a yes-instance of B

3. Show that for any no-instance $w' \notin B$, the corresponding instance w is also a no-instance of A

Show that the reduction *works*

- That is, w is a yes-instance of A if and only if w' is a yes-instance to B
  - So we can rely on the yes/no answer of $w' \in B$ to decide if $w \in A$

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



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

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

3. Show that for any no-instance $w' \notin B$, the corresponding instance w is also a no-instance of A

Show that the reduction *works*

- That is, w is a yes-instance of A if and only if w' is a yes-instance to B
    - So we can rely on the yes/no answer of $w' \in B$ to decide if $w \in A$

- If w is a yes-instance of A, there is a solution S_w to w
- Using S_w, we can construct a solution $S_{w'}$ to w'
    - Argue by how we construct w'

➡ $w' \in B$

Show that the reduction *works*

- That is, w is a yes-instance of A if and only if w' is a yes-instance to B
  - So we can rely on the yes/no answer of $w' \in B$ to decide if $w \in A$

- If w is a yes-instance of A, there is a solution S_w to w
- Using S_w, we can construct a solution $S_{w'}$ to w'
  - Argue by how we construct w'

➡$w' \in B$

- 3SAT $\leq_p$ SUBSET-SUM
  - If ϕ is a yes-instance, there is a satisfying assignment S_ϕ
  - Consult S_ϕ to choose a subset of numbers in the SUBSET-SUM instance S
  - By how we design the numbers in S and the target number t, the chosen subset has sum t

Show that the reduction *works*

- That is, w is a yes-instance of A if and only if w' is a yes-instance to B
  - So we can rely on the yes/no answer of $w' \in B$ to decide if $w \in A$

- If w' is a yes-instance of B, there is a solution $S_{w'}$ to w'
- Using $S_{w'}$, we can construct a solution S_w to w
  - Argue by how we construct w'

➡$w \in A$

Show that the reduction *works*

- That is, w is a yes-instance of A if and only if w' is a yes-instance to B
  - So we can rely on the yes/no answer of $w' \in B$ to decide if $w \in A$

- 3SAT $\leq_p$ SUBSET-SUM
  - If (S, t) is a yes-instance, there is a subset S' with sum t
  - Consult S' to assign TRUE/FALSE values to the literals in ϕ
  - By how we design the numbers in S and the target number t, the TRUE/FALSE assignment satisfies ϕ

- If w' is a yes-instance of B, there is a solution $S_{w'}$ to w'
- Using $S_{w'}$, we can construct a solution S_w to w
  - Argue by how we construct w'

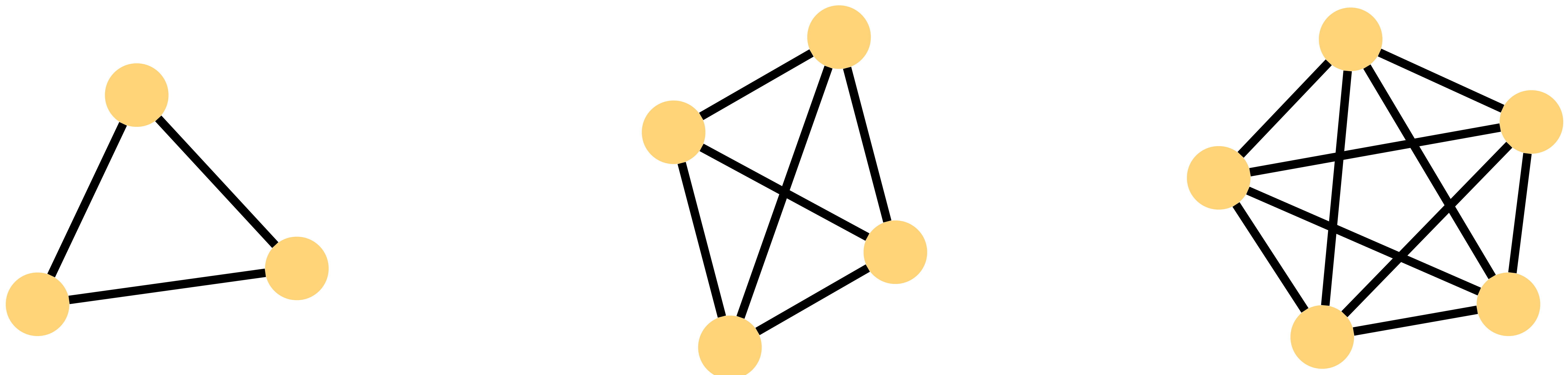
➡$w \in A$

Outline

- More NP-Hardness proofs
    - 3SAT $\leq_p$ CLIQUE
    - VERTEX-COVER $\leq_p$ FEEDBACK-VERTEX-SET
    - PARTITION $\leq_p$ BIN-PACKING
- Pseudo-polynomial time algorithms
- **NP** and **Co-NP**

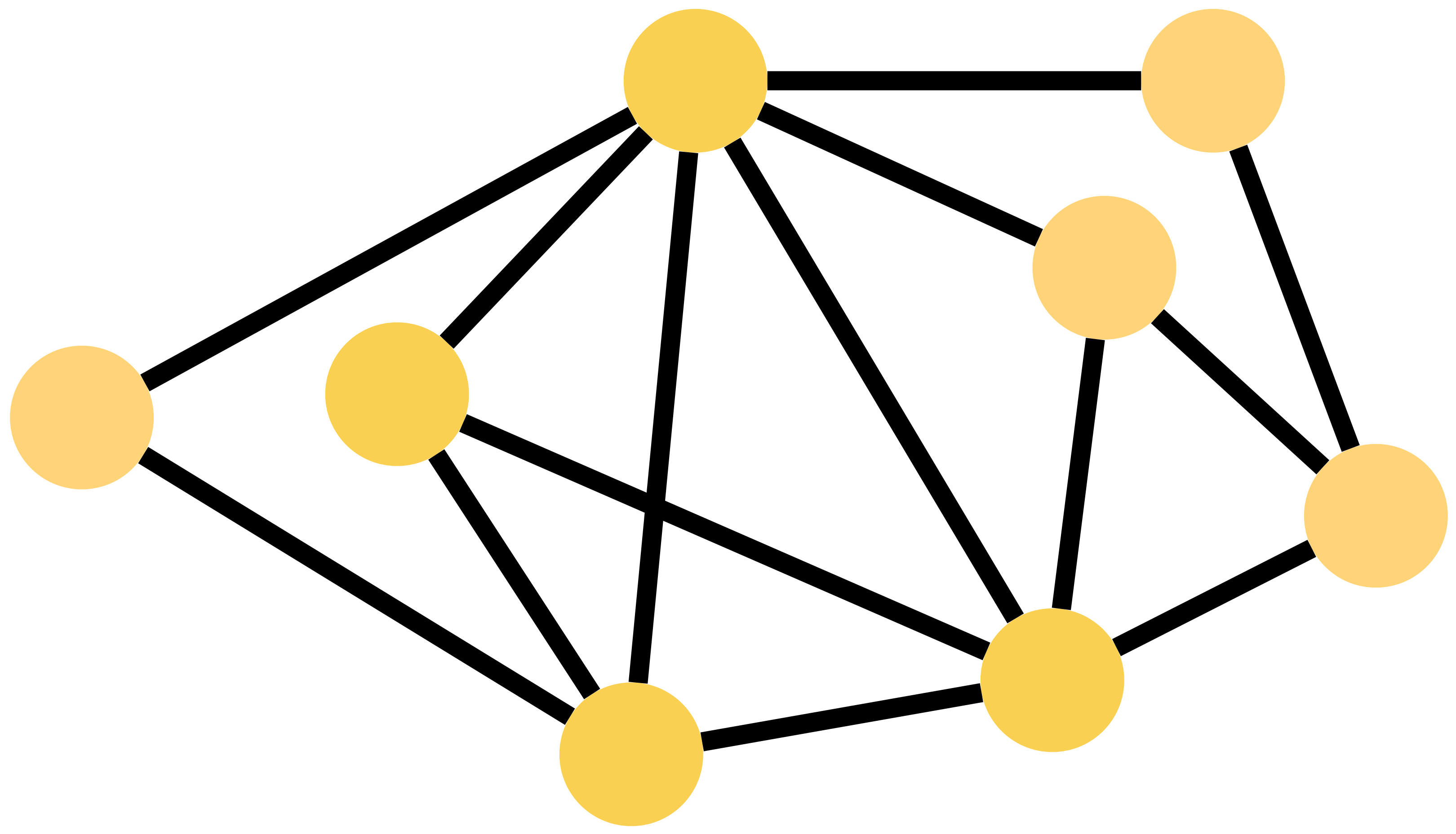
CLIQUE

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



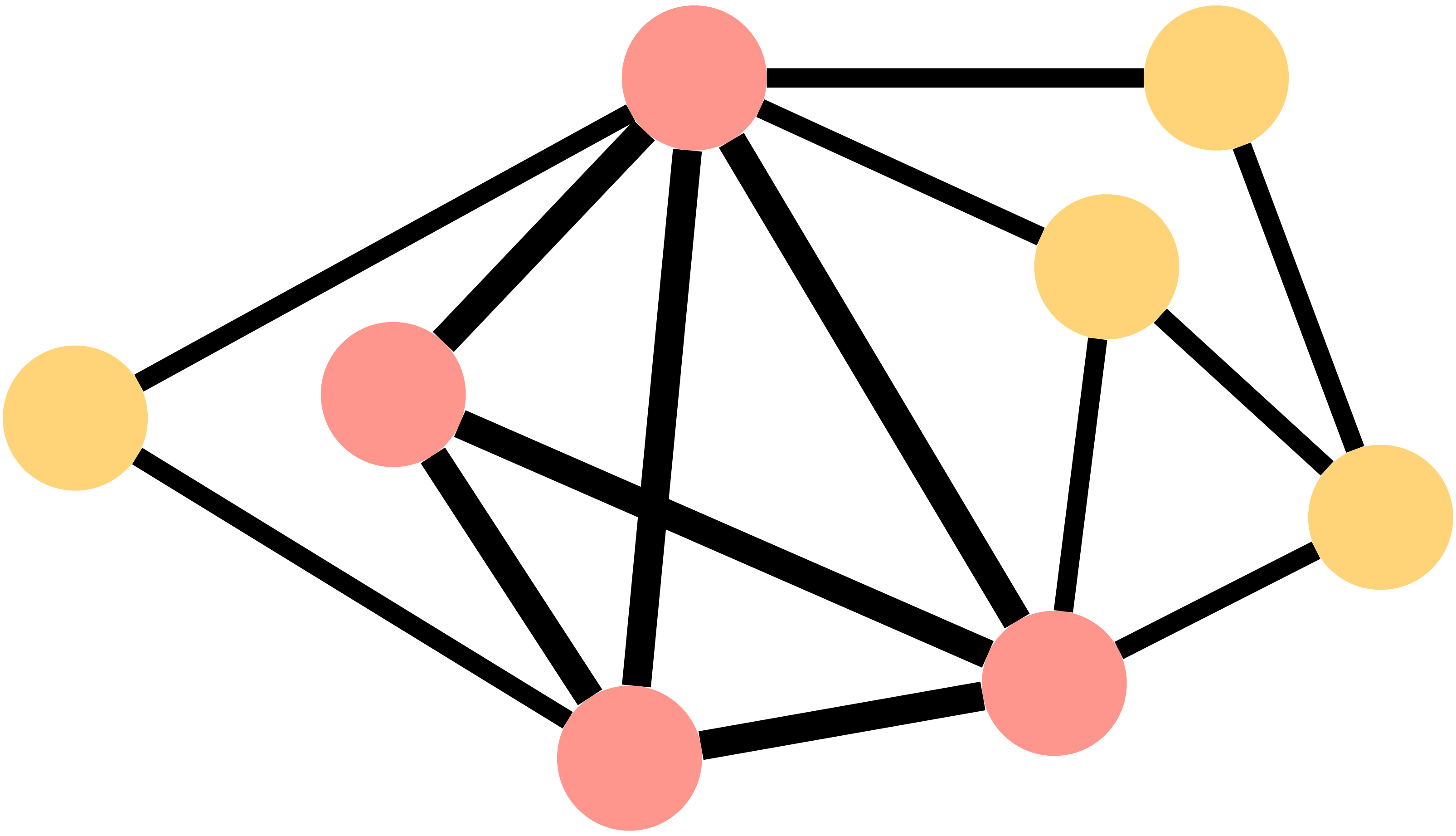
CLIQUE

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



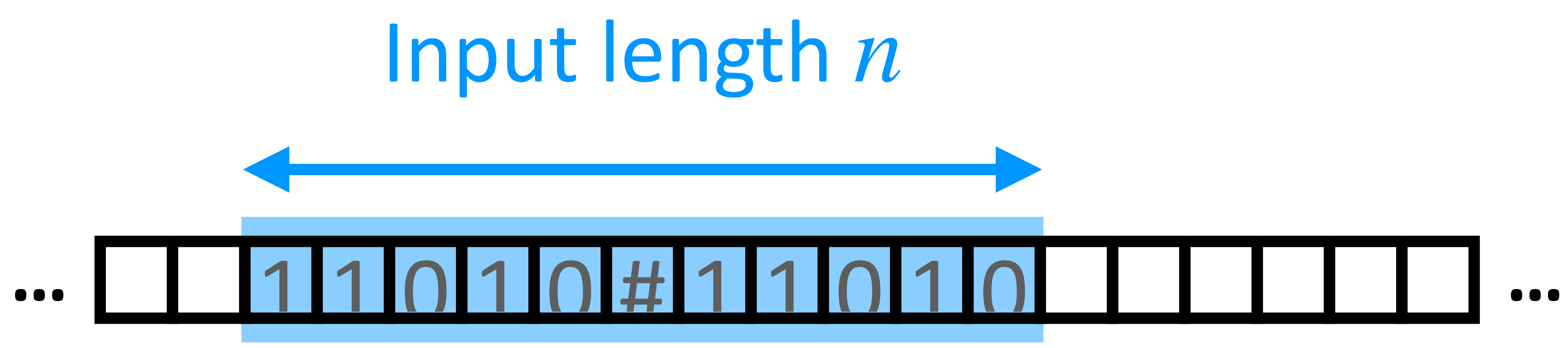
CLIQUE

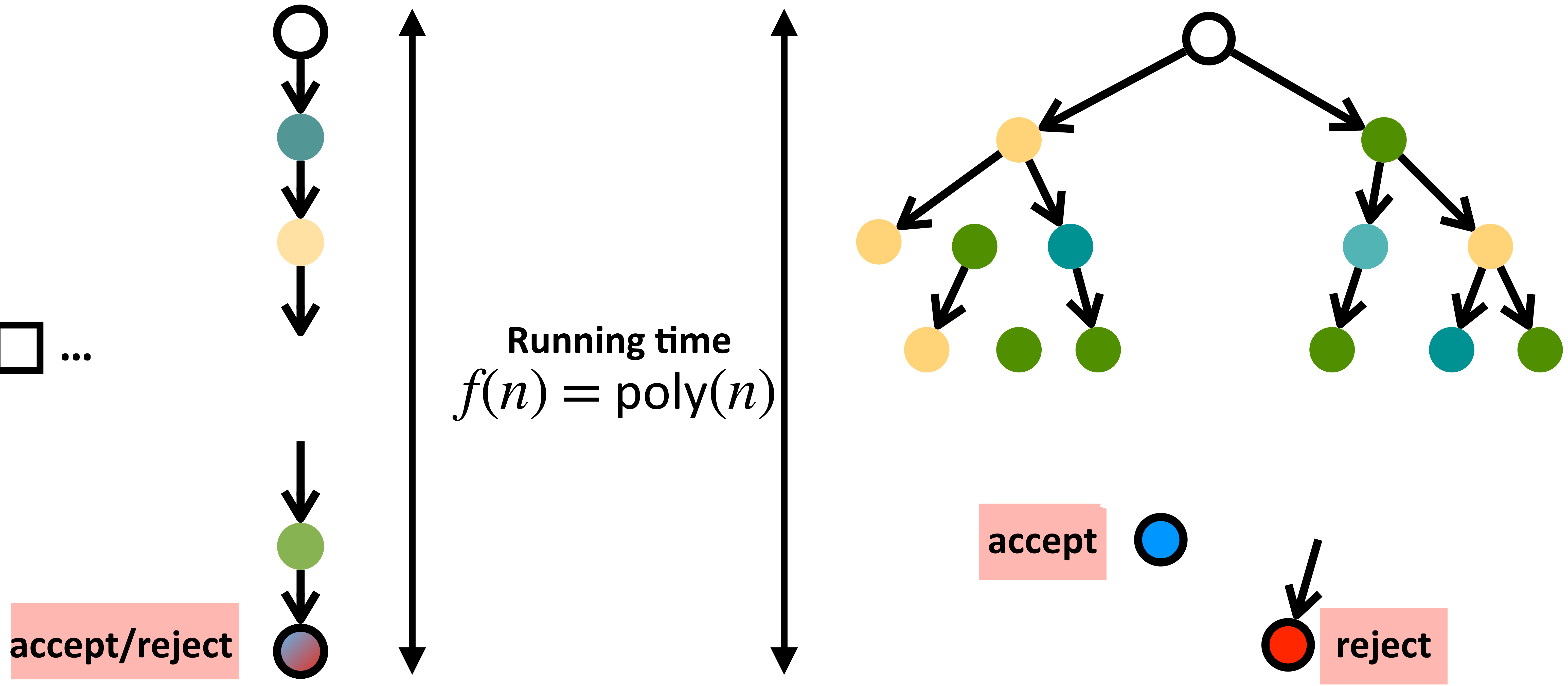
- Maximum clique problem: Given a graph G, what is the size of the maximum clique in G?



Turing machine

- The class **P** is the class of languages that are *accepted* or *rejected* in polynomial time by a deterministic Turing machine
- The class **NP** is the class of languages that can be *verified* in polynomial time by a deterministic Turing machine.





Decision Problems and Optimization Problems

- **Decision** problems:

- **Optimization** problems:

Decision Problems and Optimization Problems

- **Decision** problems: Given a problem and an input of the problem, asking if we feed this input to the problem, the answer is *yes* or *no*

- **Optimization** problems:

Decision Problems and Optimization Problems

- **Decision** problems: Given a problem and an input of the problem, asking if we feed this input to the problem, the answer is *yes* or *no*
    - Ex: Partition problem
- **Optimization** problems:

Decision Problems and Optimization Problems

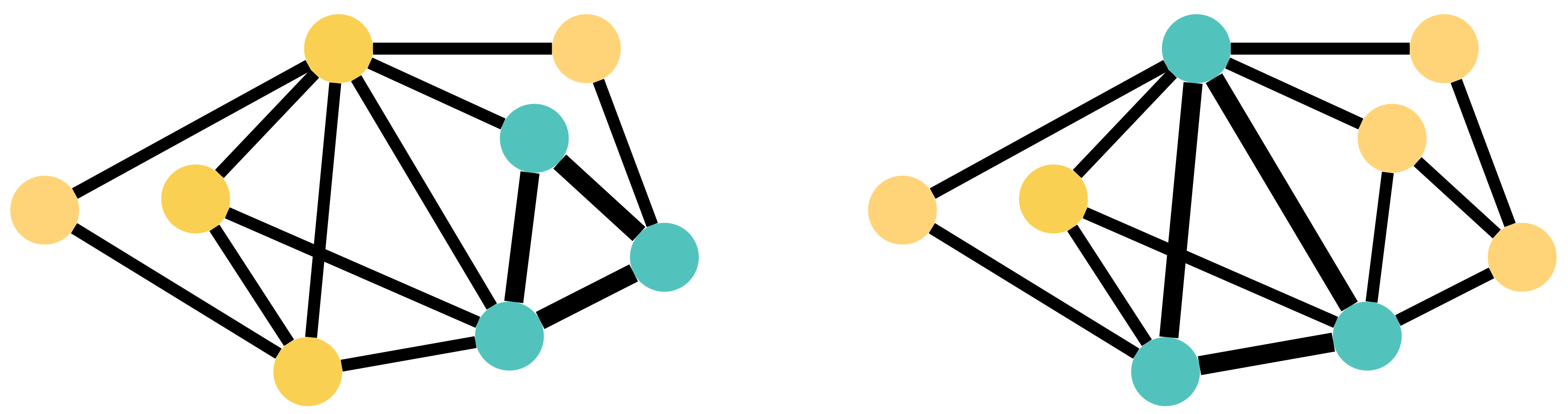
- **Decision** problems: Given a problem and an input of the problem, asking if we feed this input to the problem, the answer is *yes* or *no*
    - Ex: Partition problem
- **Optimization** problems: finding the *best* solution among all feasible solutions

Decision Problems and Optimization Problems

- **Decision** problems: Given a problem and an input of the problem, asking if we feed this input to the problem, the answer is *yes* or *no*
  - Ex: Partition problem
- **Optimization** problems: finding the *best* solution among all feasible solutions
  - Feasible solution: a solution that satisfies the requirement but probably not the best

Decision Problems and Optimization Problems

- **Decision** problems: Given a problem and an input of the problem, asking if we feed this input to the problem, the answer is *yes* or *no*
  - Ex: Partition problem
- **Optimization** problems: finding the *best* solution among all feasible solutions
  - Feasible solution: a solution that satisfies the requirement but probably not the best
    - A subgraph which is a clique is not necessary the one that contains minimum number of vertices



Decision Problems and Optimization Problems

- **Decision** problems: Given a problem and an input of the problem, asking if we feed this input to the problem, the answer is *yes* or *no*
  - Ex: Partition problem
- **Optimization** problems: finding the *best* solution among all feasible solutions
  - Feasible solution: a solution that satisfies the requirement but probably not the best
    - A subgraph which is a clique is not necessary the one that contains minimum number of vertices
    - Minimization or maximization

Decision Problems and Optimization Problems

- **Decision** problems: Given a problem and an input of the problem, asking if we feed this input to the problem, the answer is *yes* or *no*
  - Ex: Partition problem
- **Optimization** problems: finding the *best* solution among all feasible solutions
  - Feasible solution: a solution that satisfies the requirement but probably not the best
    - A subgraph which is a clique is not necessary the one that contains minimum number of vertices
    - Minimization or maximization
      - Ex: Minimum vertex cover or Maximum clique

Optimization? An Equivalent Decision Problem

Optimization? An Equivalent Decision Problem

- The classes **P** and **NP** are both define on decision problems. How do we classify optimization problems?

Optimization? An Equivalent Decision Problem

- The classes **P** and **NP** are both define on decision problems. How do we classify optimization problems?
- We can recast an optimization problem as a decision problem that is no harder!

Optimization? An Equivalent Decision Problem

- The classes **P** and **NP** are both define on decision problems. How do we classify optimization problems?
- We can recast an optimization problem as a decision problem that is no harder!
  - Optimization problem: we want to minimize/maximize…

Optimization? An Equivalent Decision Problem

- The classes **P** and **NP** are both define on decision problems. How do we classify optimization problems?
- We can recast an optimization problem as a decision problem that is no harder!
  - Optimization problem: we want to minimize/maximize…
  - Equivalent decision version problem: we want to find a solution with cost at most/least k

Optimization? An Equivalent Decision Problem

- The classes **P** and **NP** are both define on decision problems. How do we classify optimization problems?
- We can recast an optimization problem as a decision problem that is no harder!
  - Optimization problem: we want to minimize/maximize…
  - Equivalent decision version problem: we want to find a solution with cost at most/least k
    - k is an additional parameter

CLIQUE

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

- Decision version?

CLIQUE

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

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

New parameter!

CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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

CLIQUE

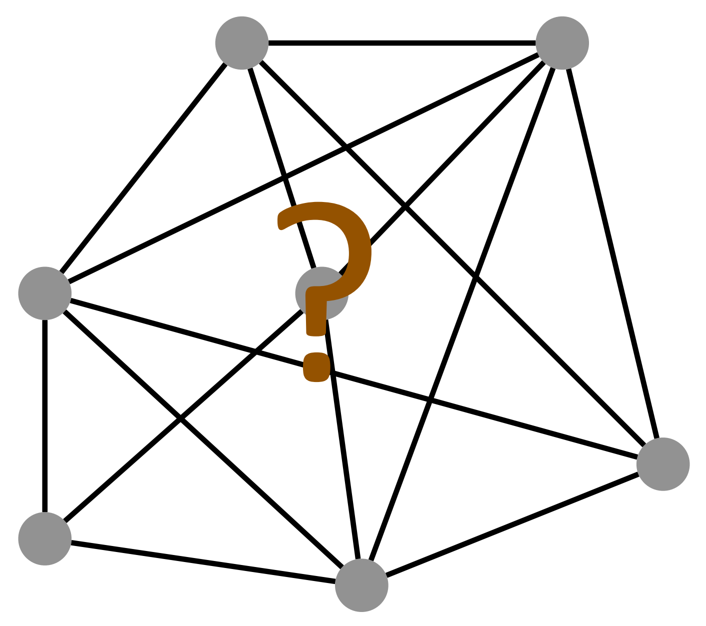
- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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



k = ?

CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

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

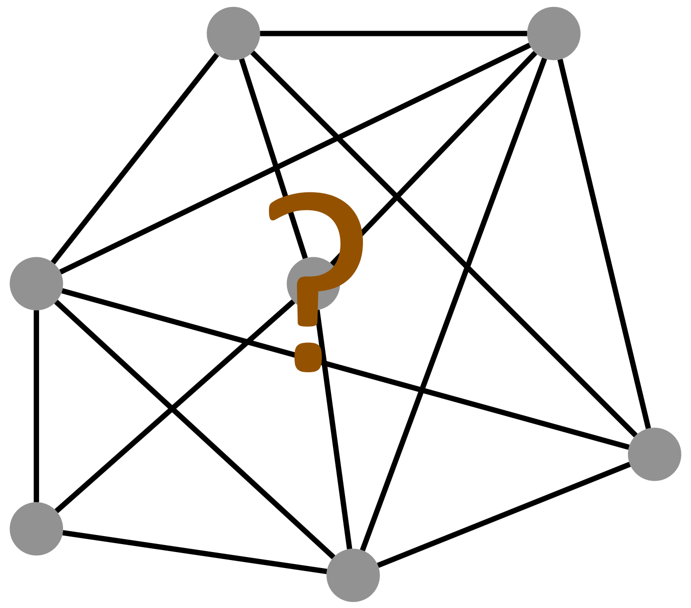
- CLIQUE $= \{\langle G, k\rangle \mid G$ has a clique of size at least $k\}$

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

satisfiable

$\Updownarrow$

There is a k-clique in G



k = ?

CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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

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

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

CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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

x_1 x_1 x_2

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

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

CLIQUE

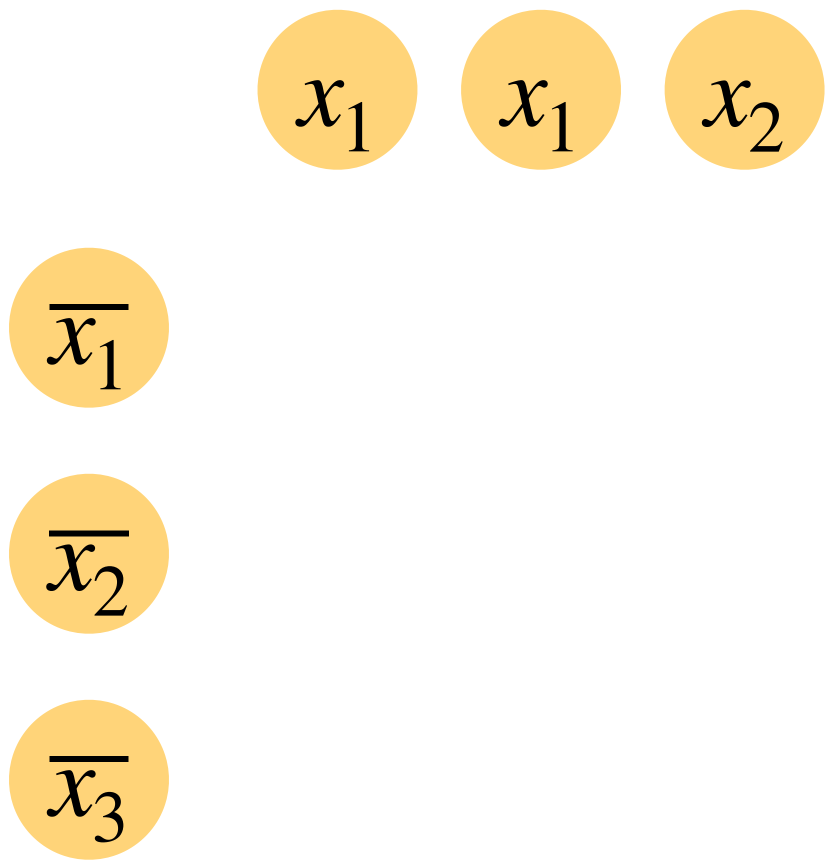
- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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



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

CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

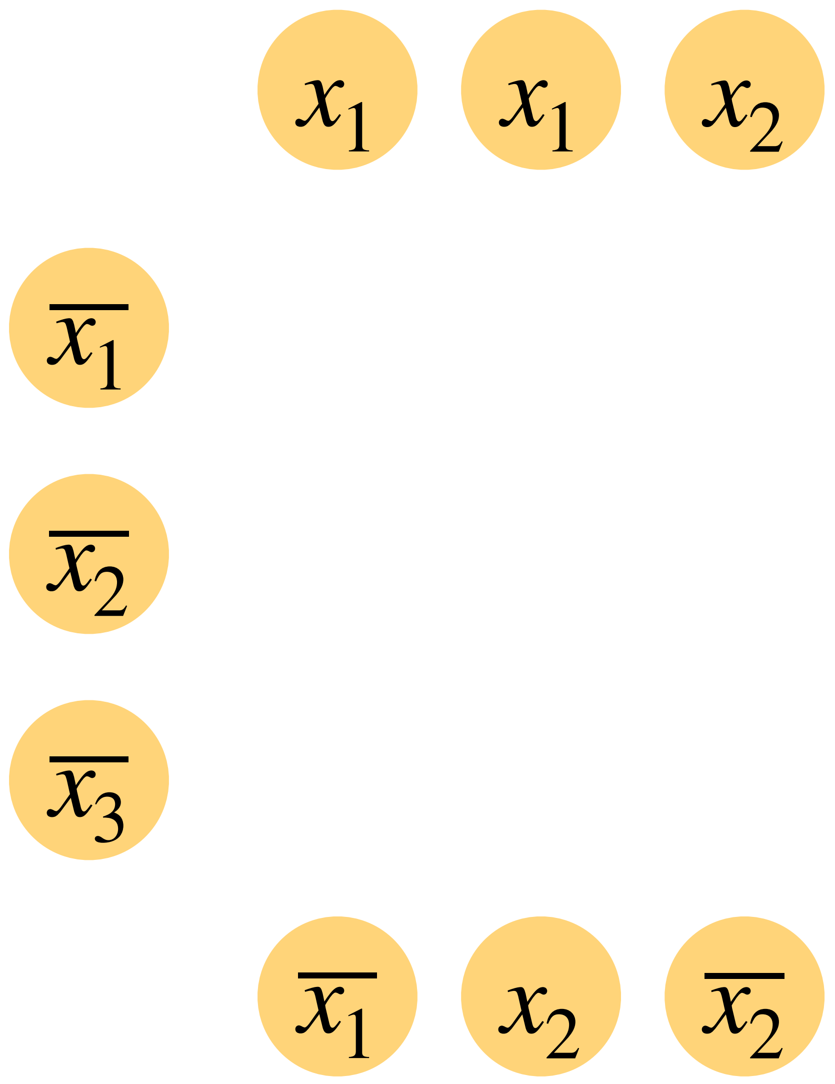
- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$

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

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

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

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



CLIQUE

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

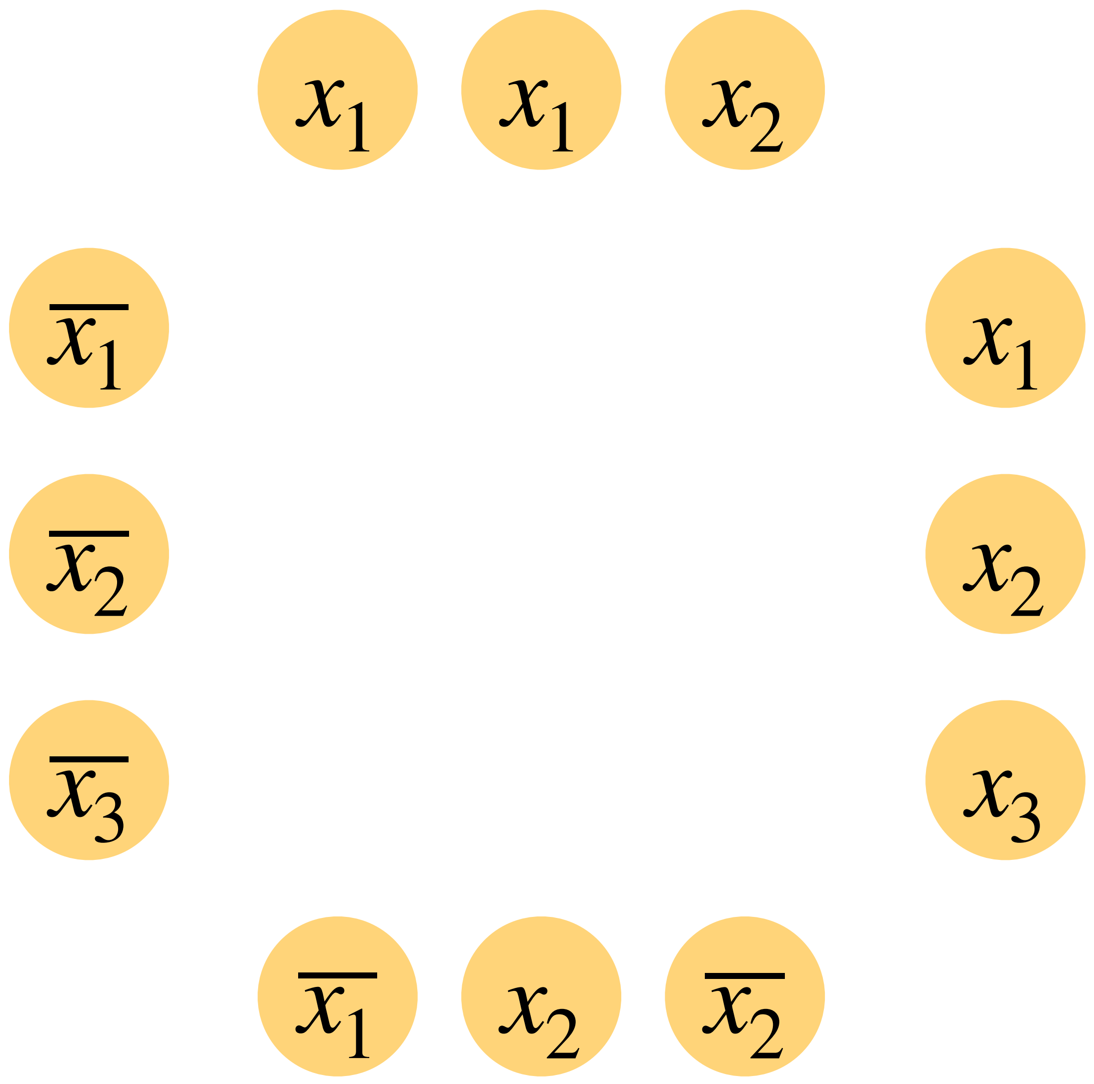
<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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

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



CLIQUE

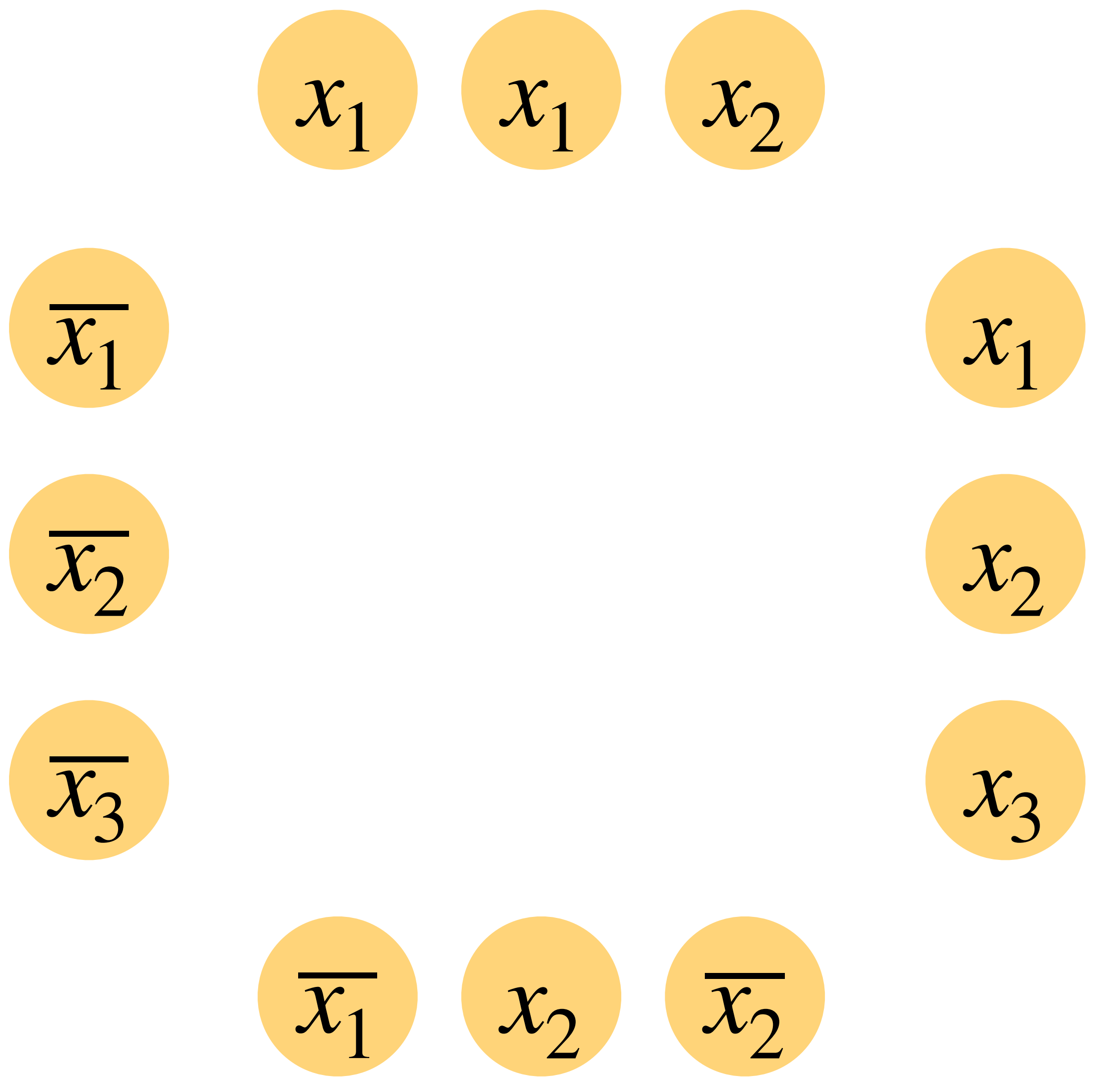
- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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



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

CLIQUE

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

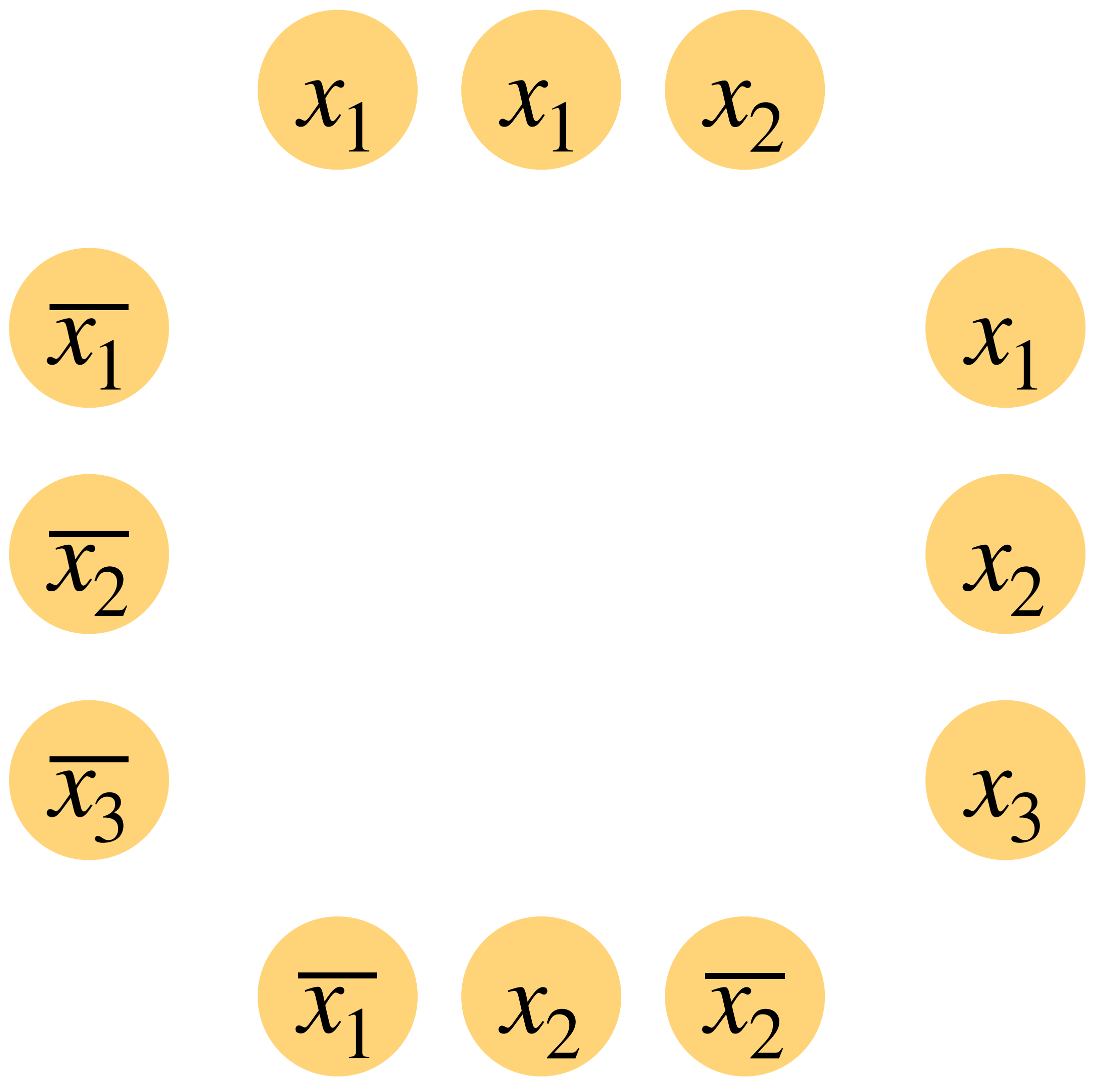
<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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

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



CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

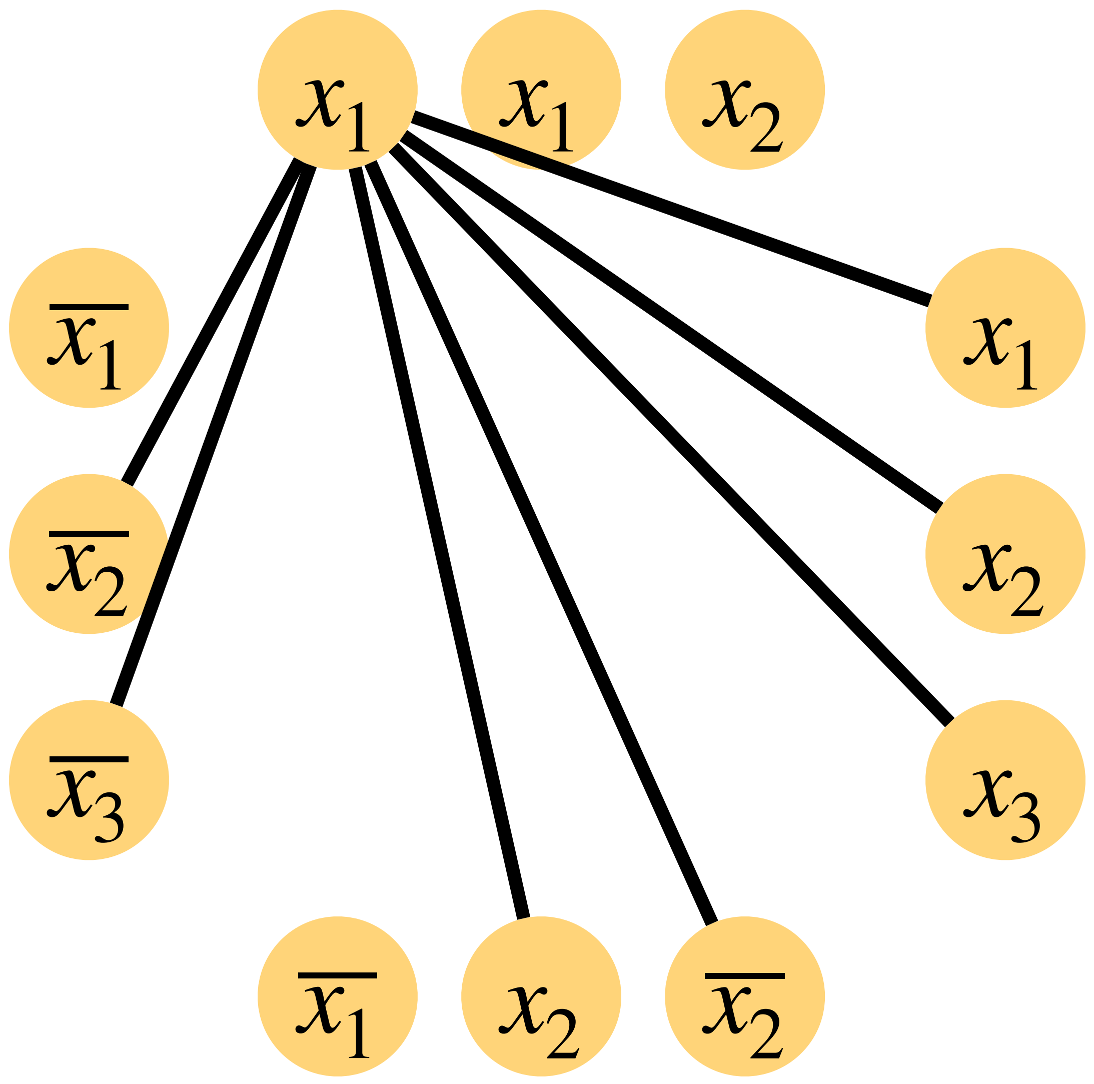
- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$

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

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

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

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



CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

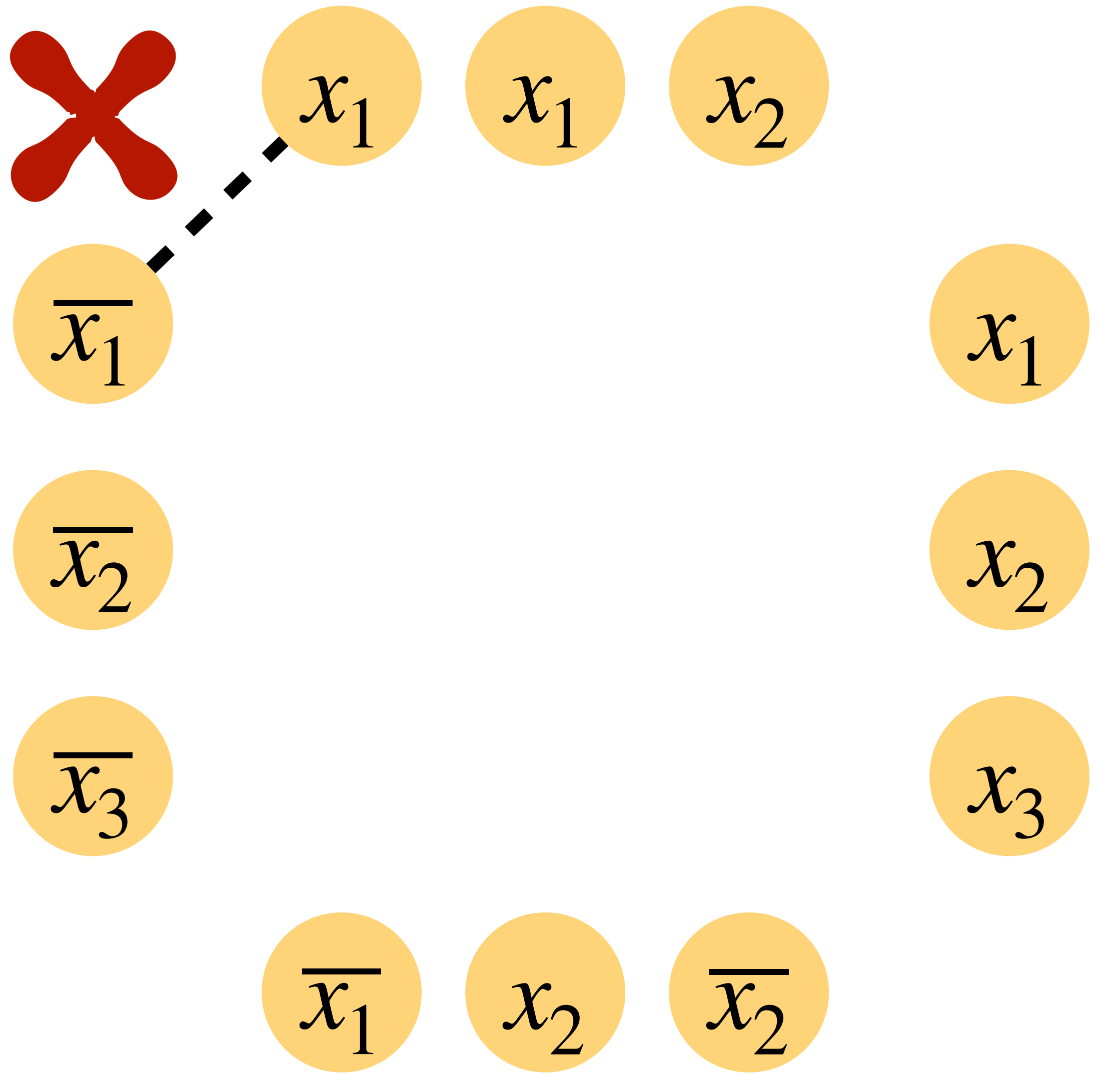
- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$

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

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

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

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



CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

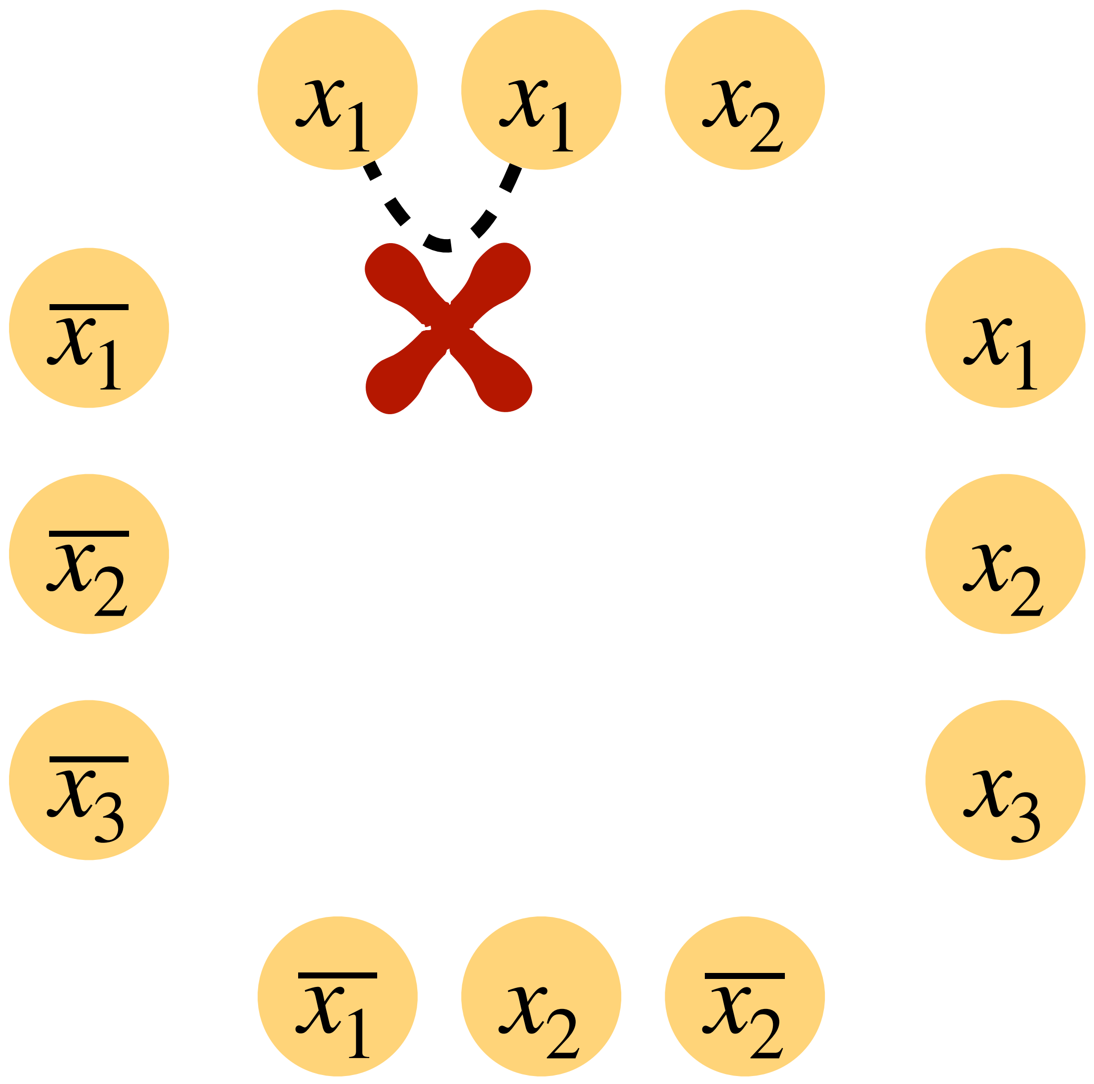
- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$

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

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

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

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



CLIQUE

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

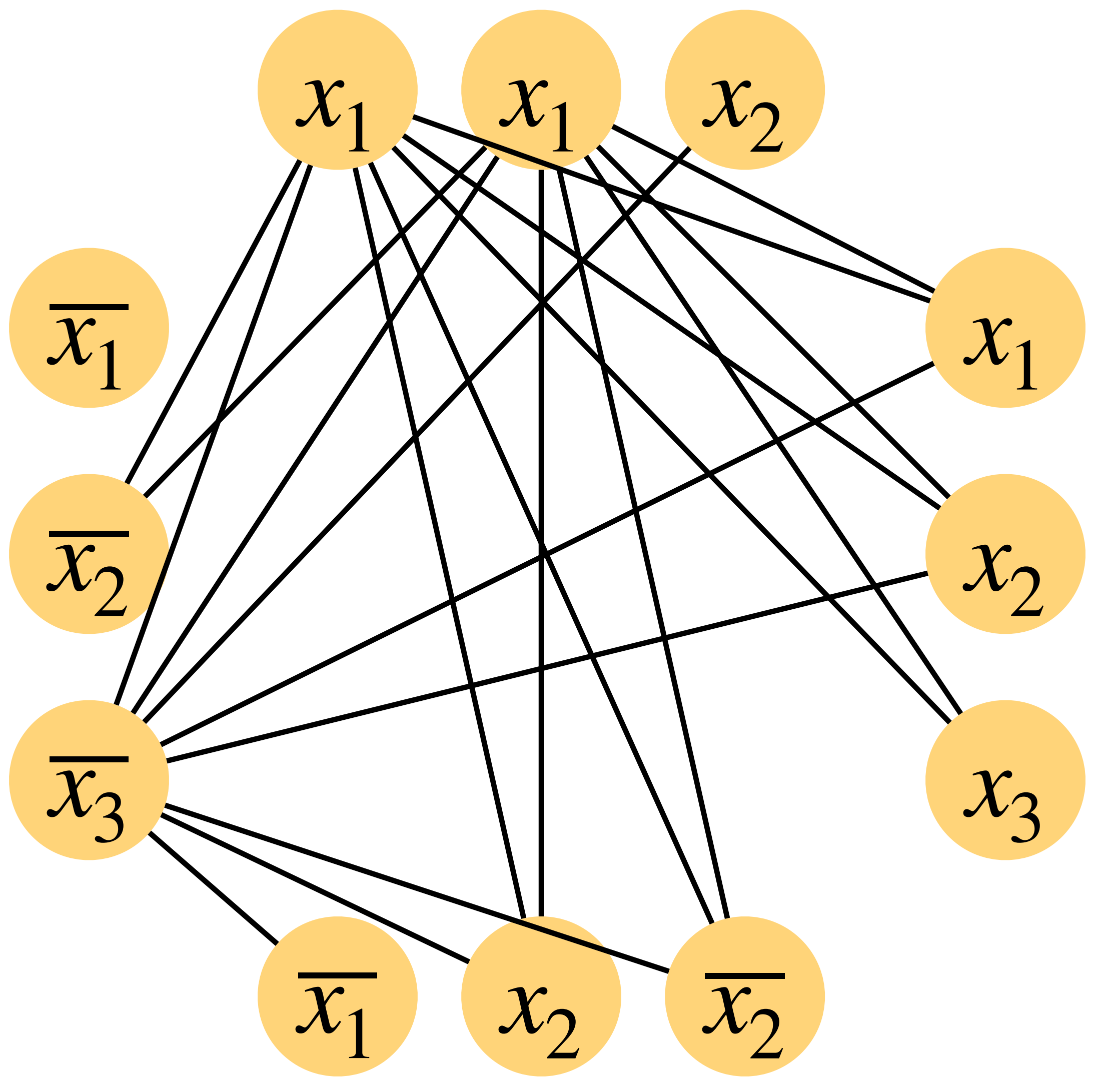
<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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

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



CLIQUE

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

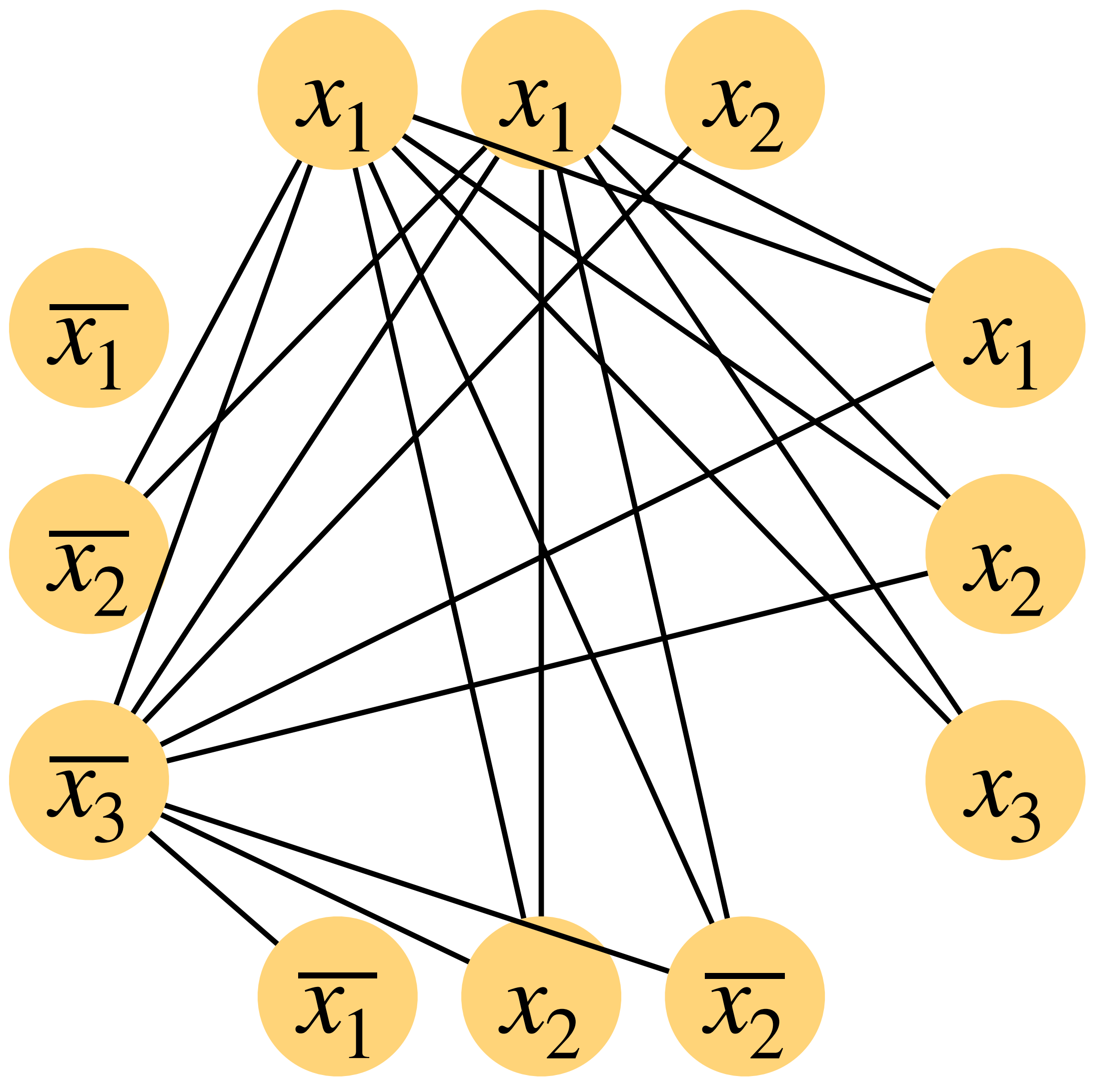
<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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

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



CLIQUE

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

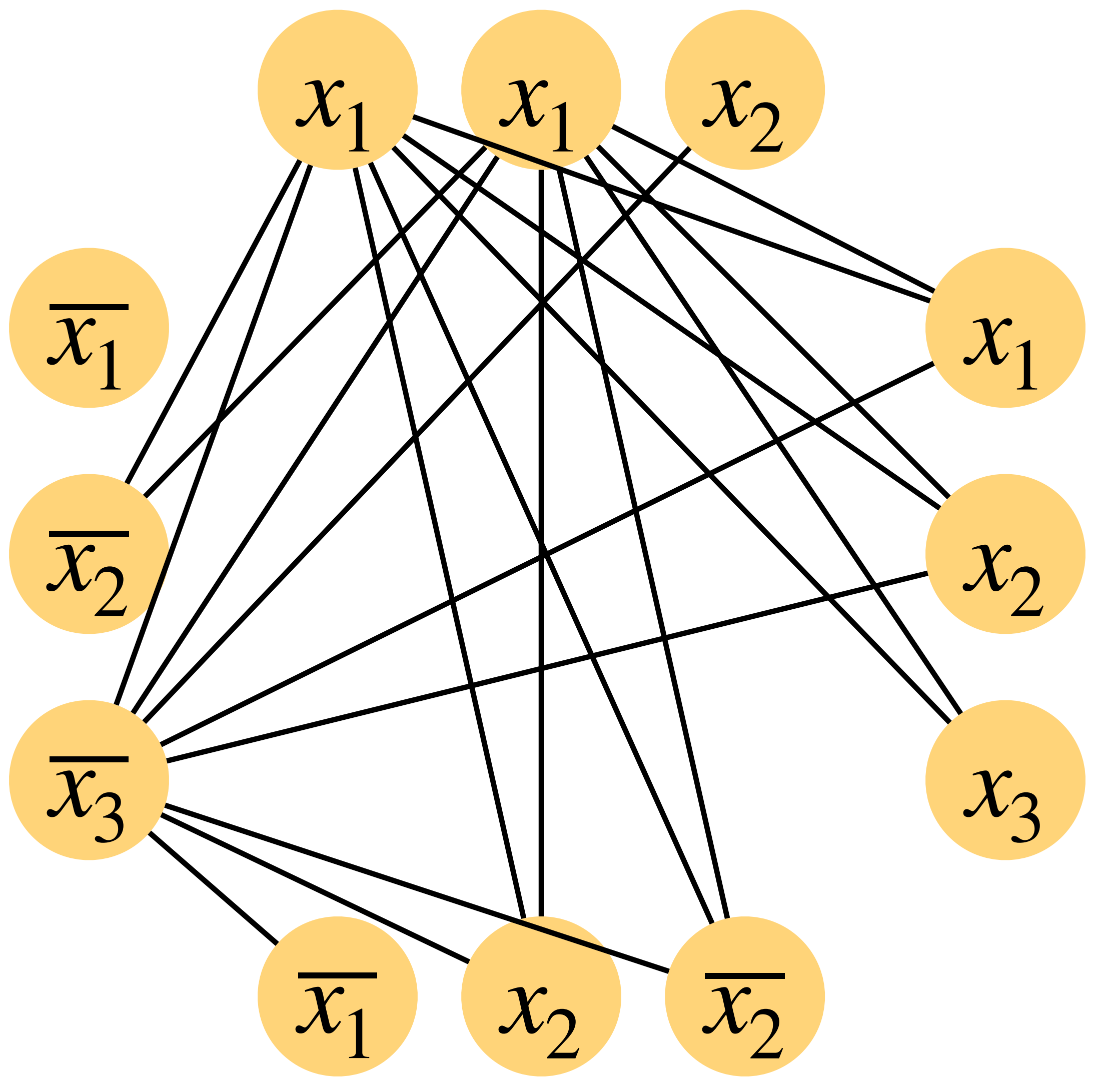
<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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

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



CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

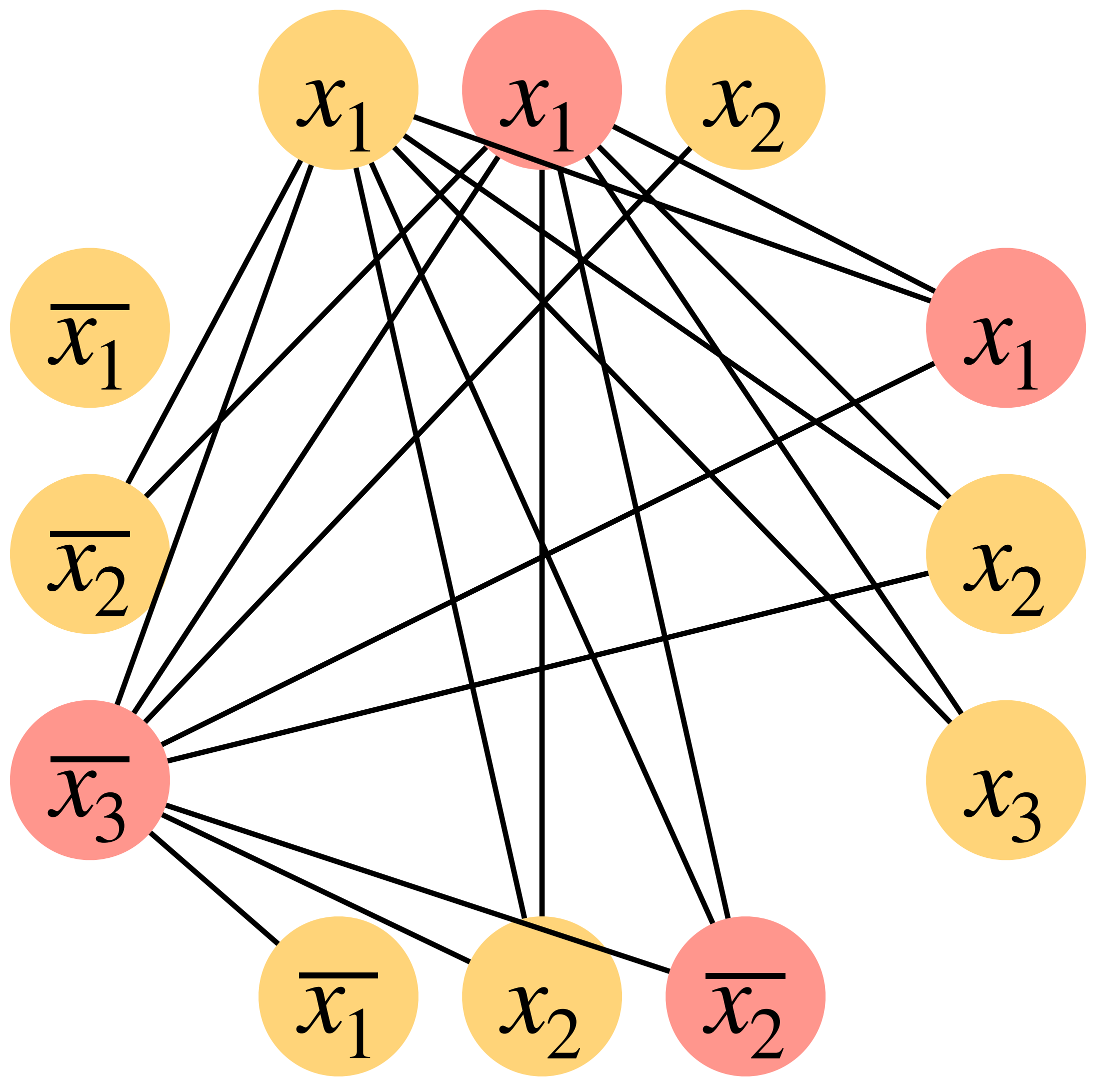
- 3SAT $= \{\langle \phi \rangle \mid \phi$ is a satisfiable 3-CNF Boolean formula$\}$

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

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

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

Consult the satisfying assignment to construct a solution to CLIQUE



CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

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

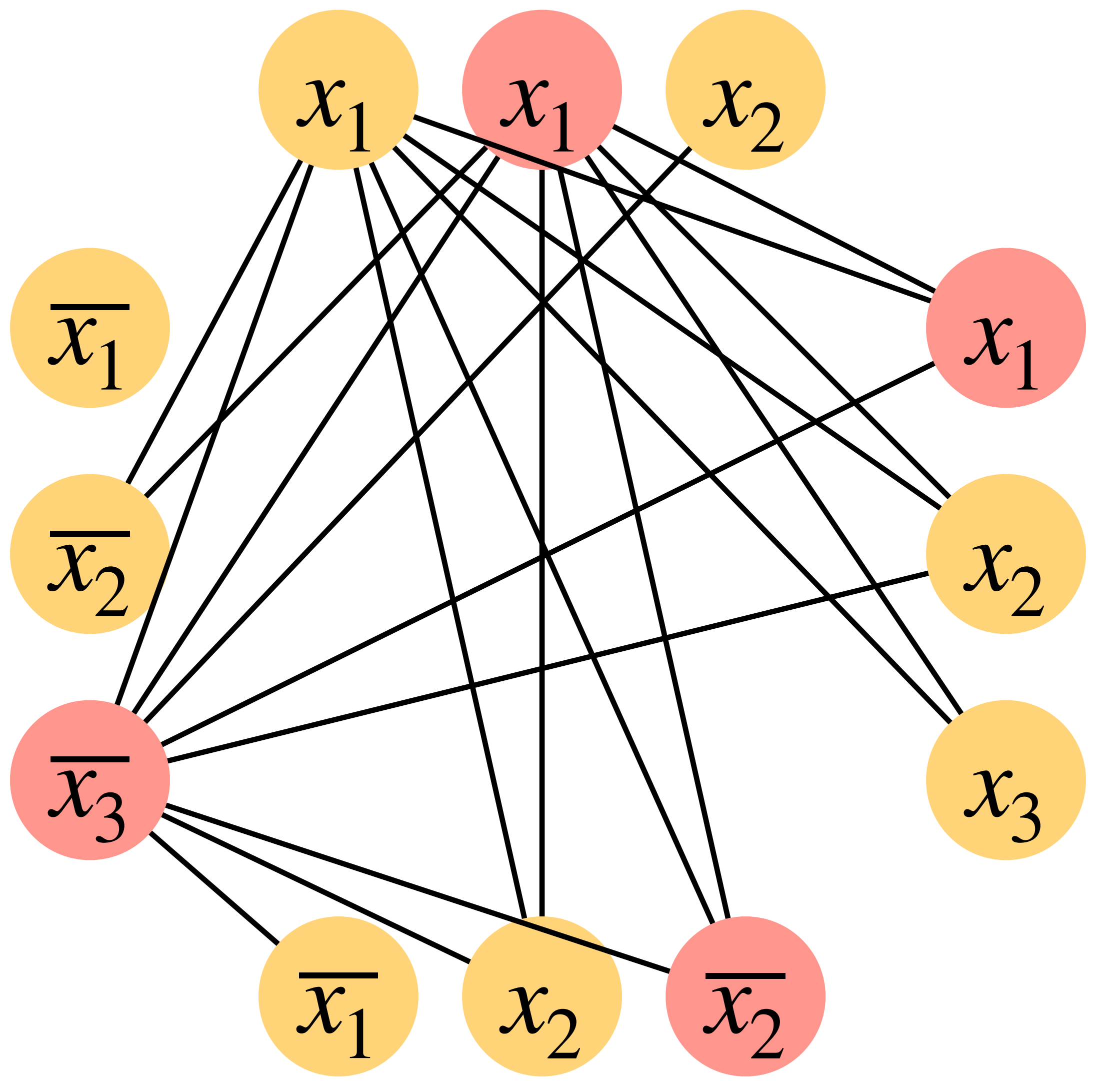
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

$$(x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_3}) \land (\overline{x_1} \lor x_2 \lor \overline{x_2}) \land (x_1 \lor x_2 \lor x_3)$$

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

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

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



CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

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

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

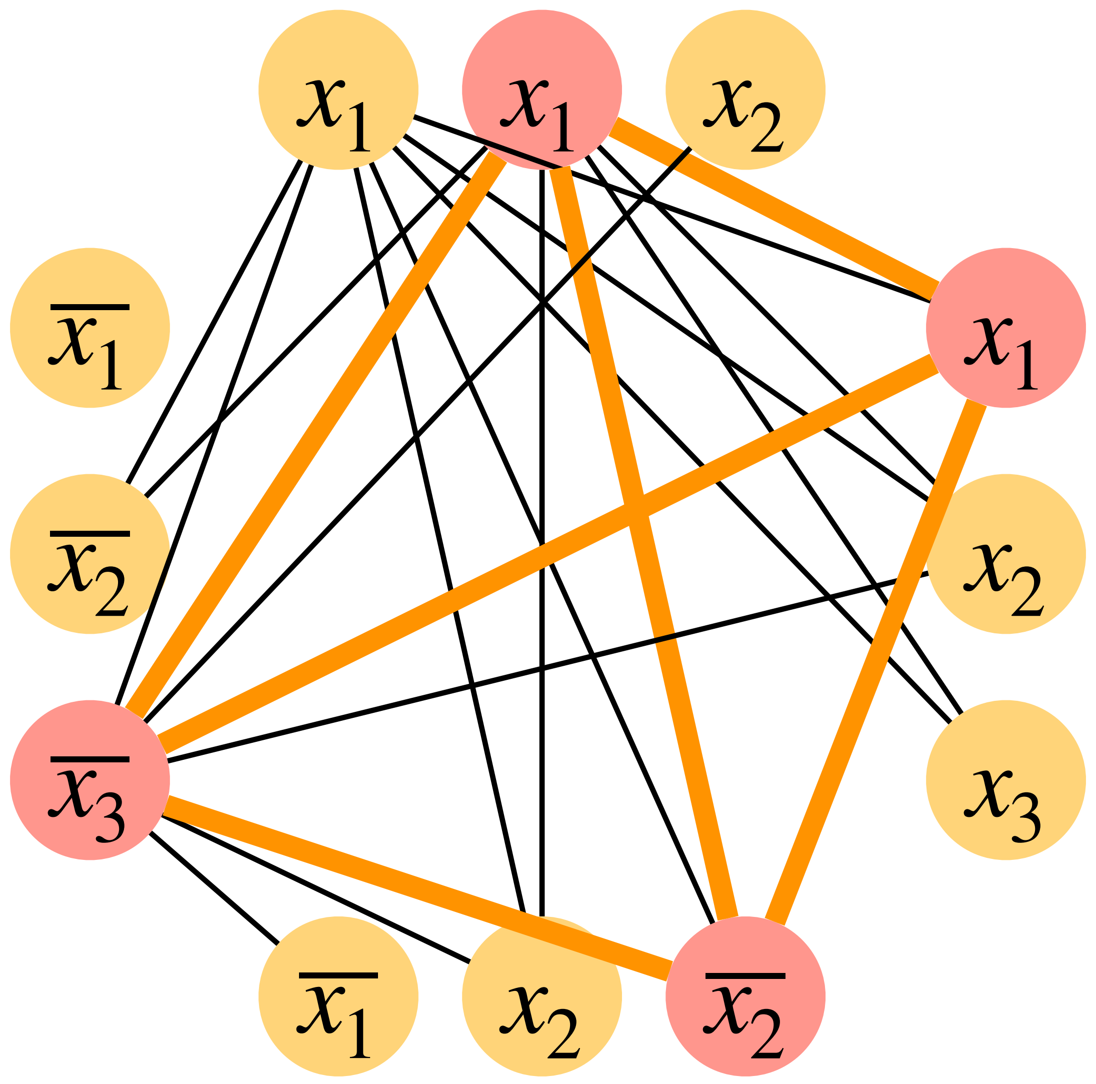
$$(x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_2}) \wedge (x_1 \vee x_2 \vee x_3)$$

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

$\Downarrow$

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

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



CLIQUE

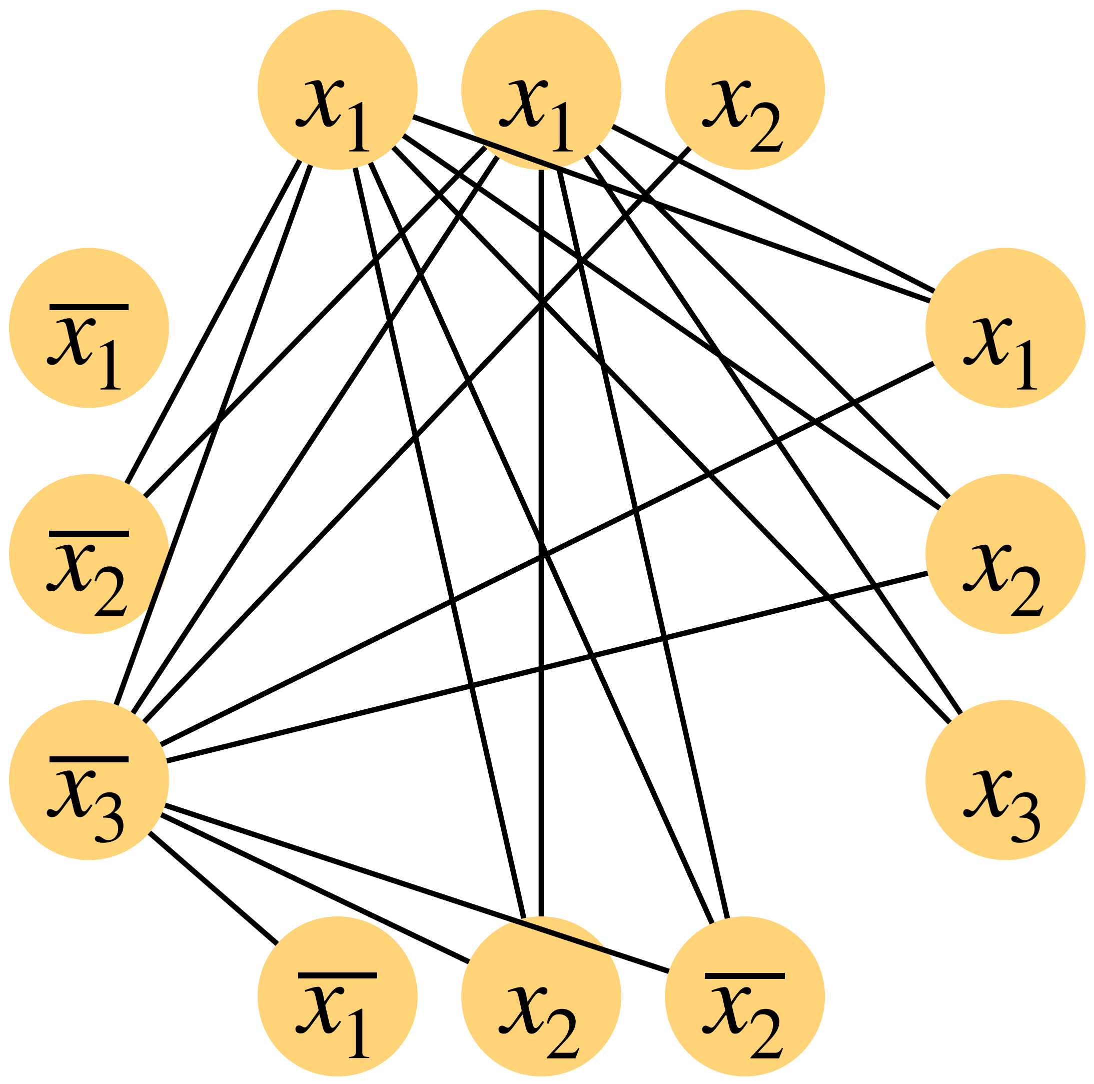
- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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



CLIQUE

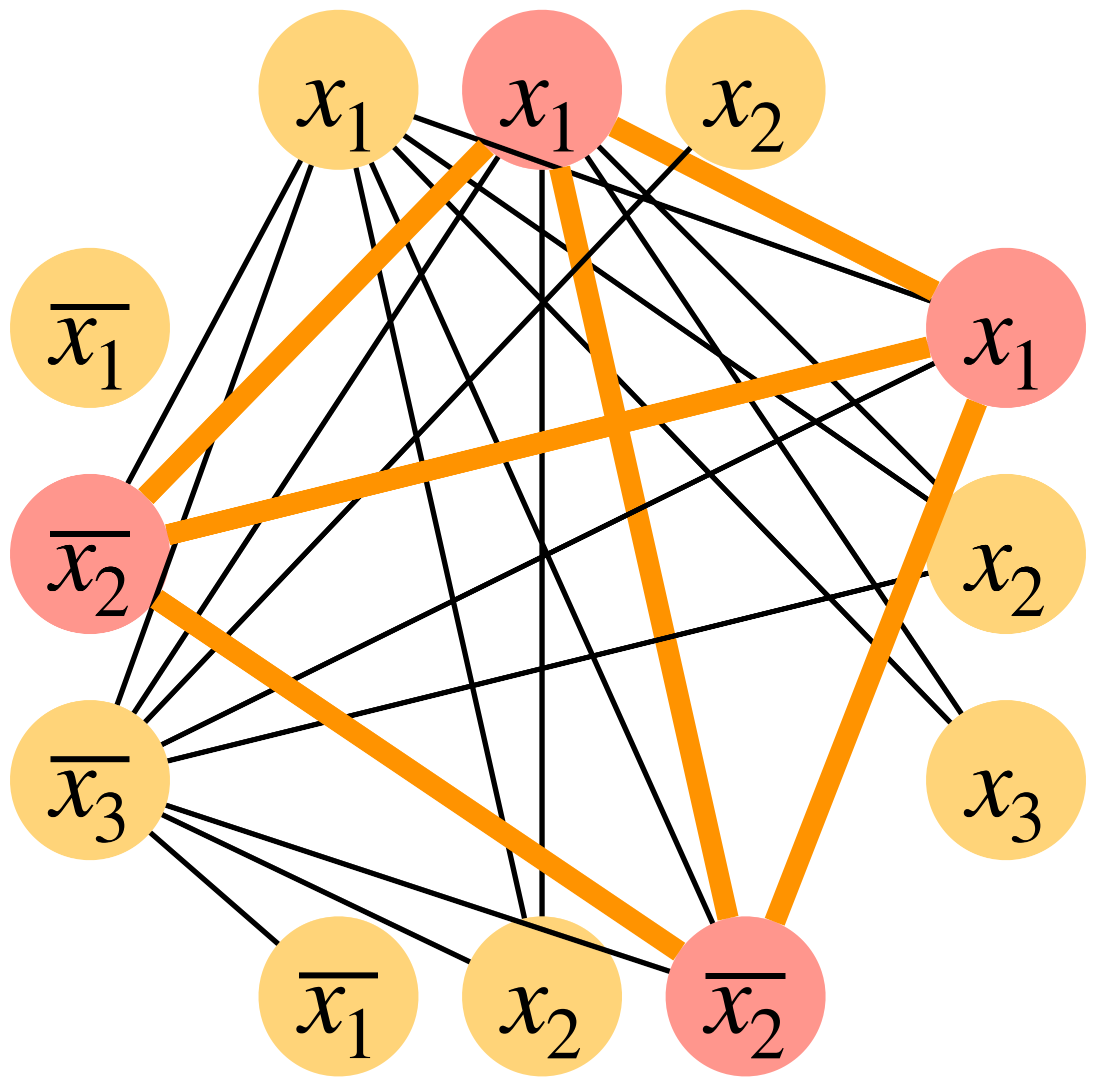
- Theorem: CLIQUE = $\{\langle G, k\rangle$ | There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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



If there is a m-clique in G

CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

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

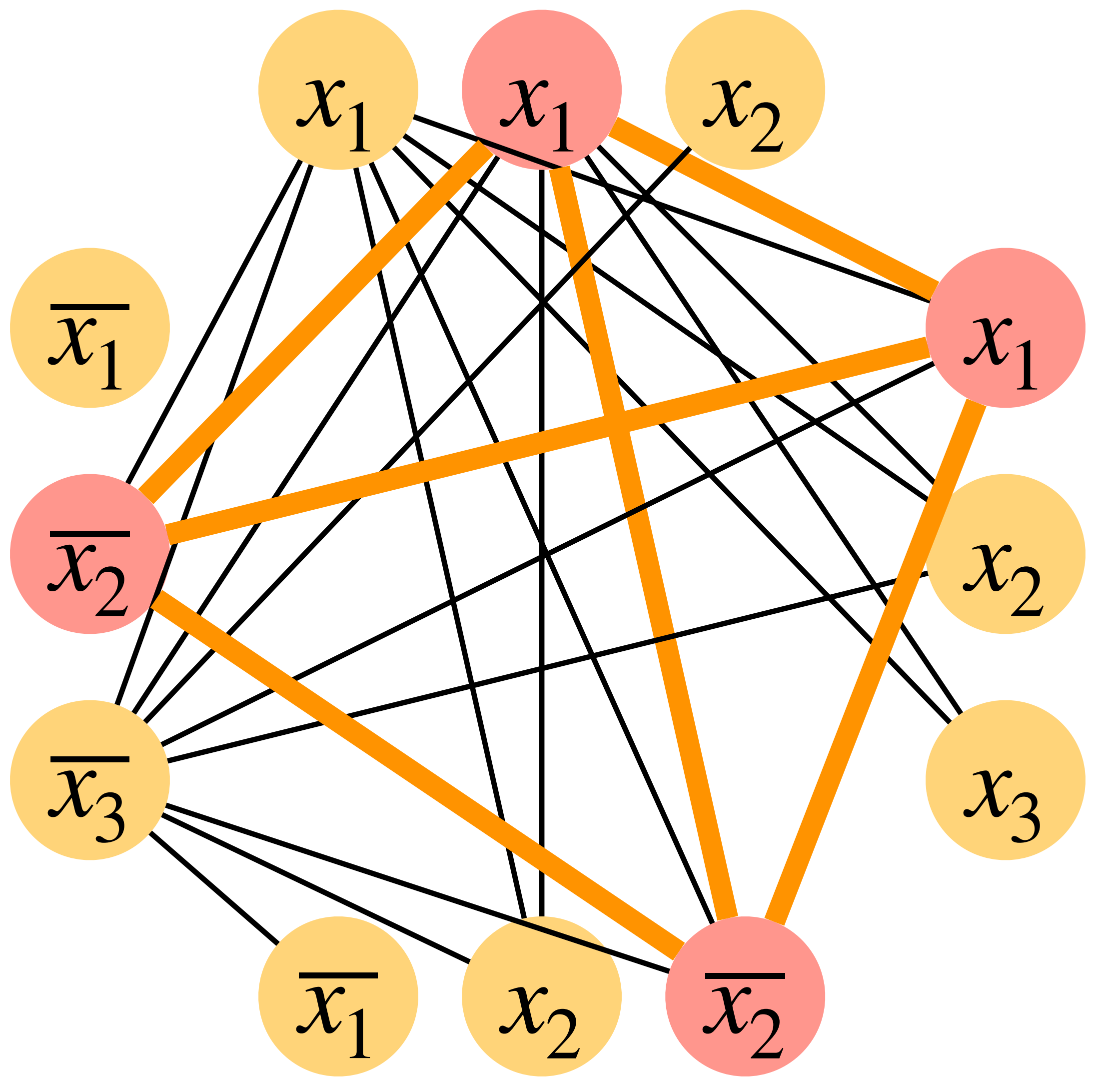
- CLIQUE $= \{\langle G, m\rangle \mid G$ has a clique of size at least $m\}$

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

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

$\Uparrow$

If there is a m-clique in G



CLIQUE

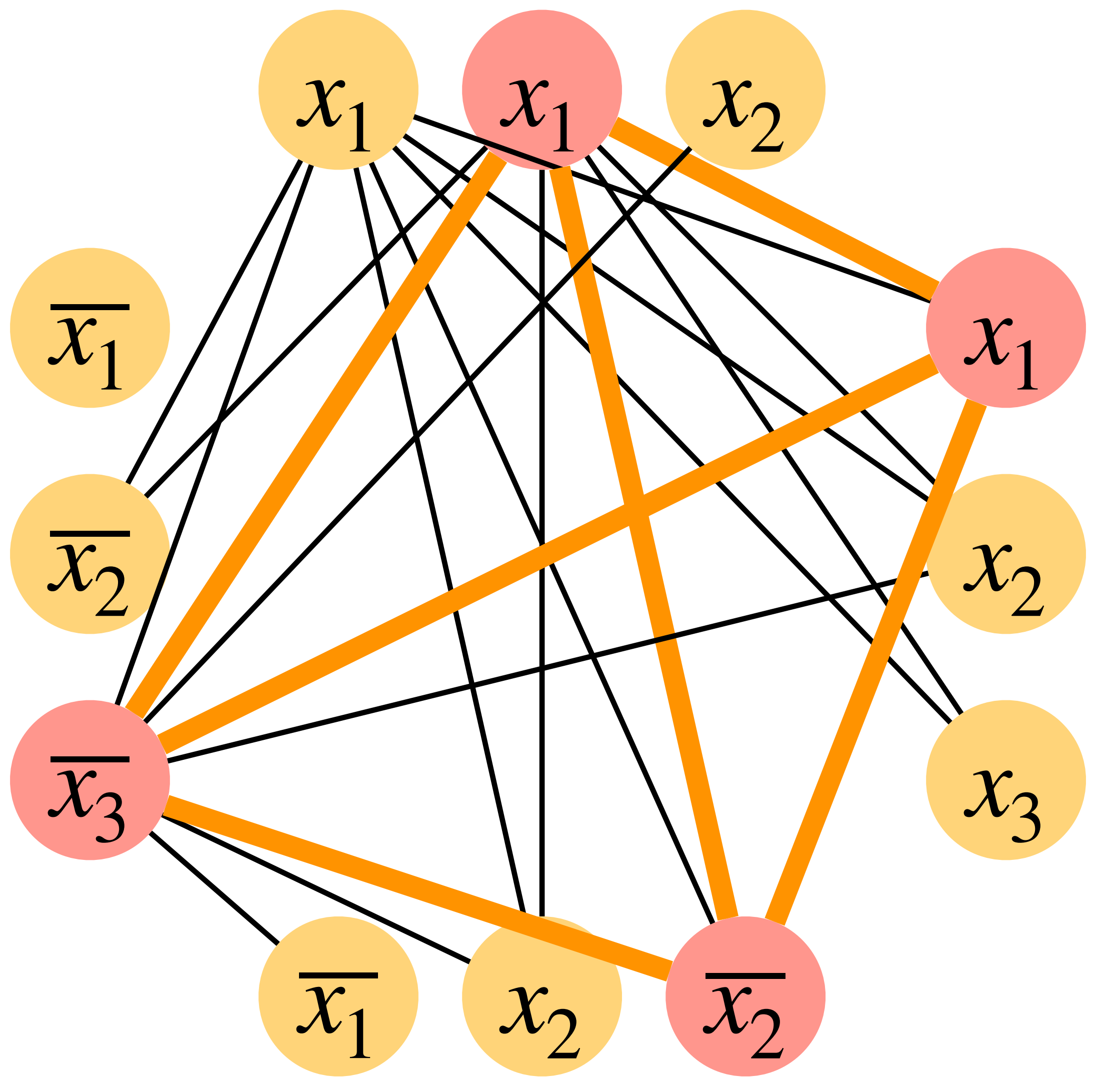
- Theorem: CLIQUE = $\{\langle G, k\rangle \mid$ There is a clique in G with size at least $k\}$ is NP-Hard

<Proof Idea> Polynomial-time reduction from 3SAT

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

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

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



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

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

CLIQUE

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

<Proof Idea> Polynomial-time reduction from 3SAT

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

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

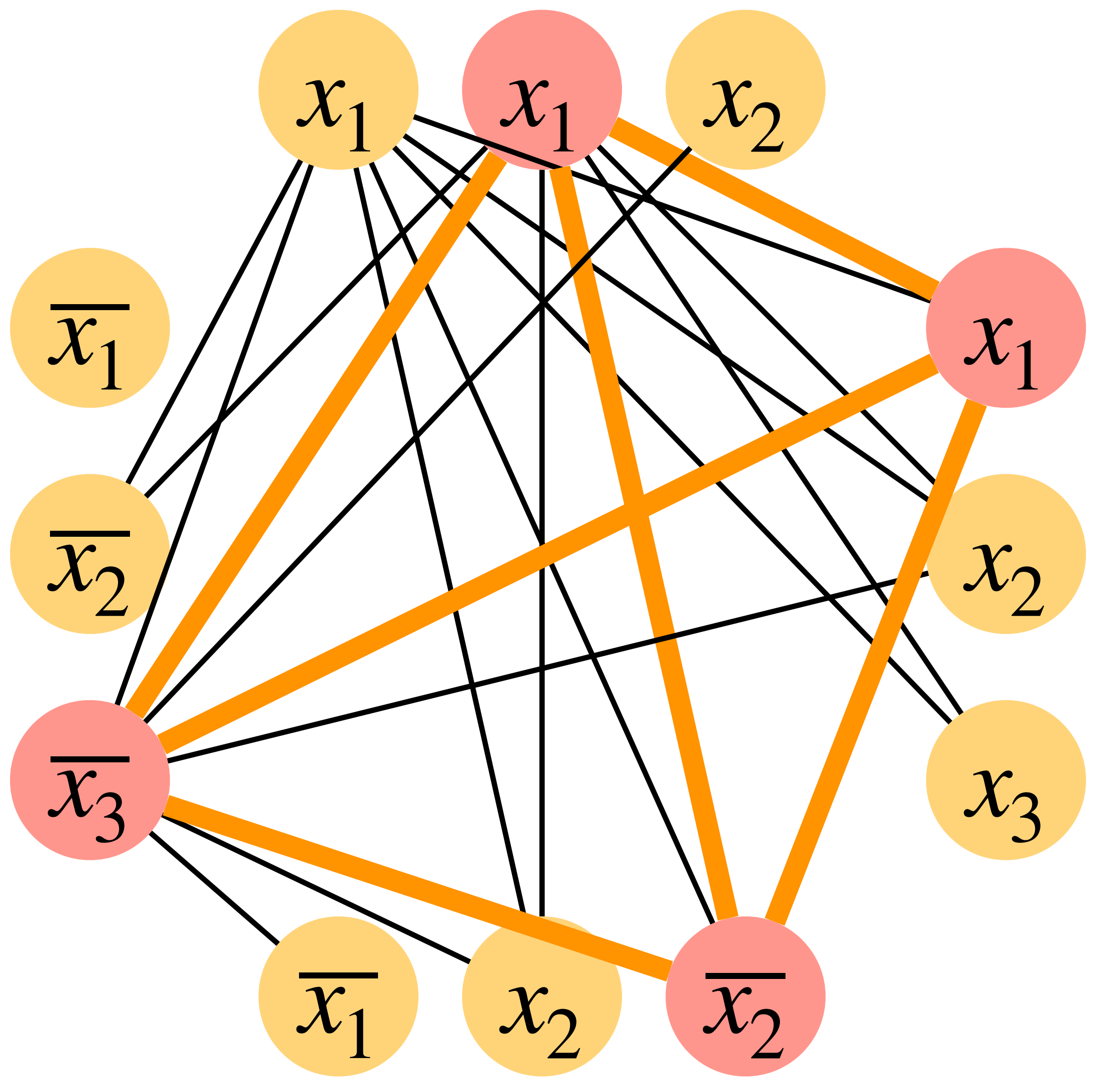
$$(x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_2}) \wedge (x_1 \vee x_2 \vee x_3)$$

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

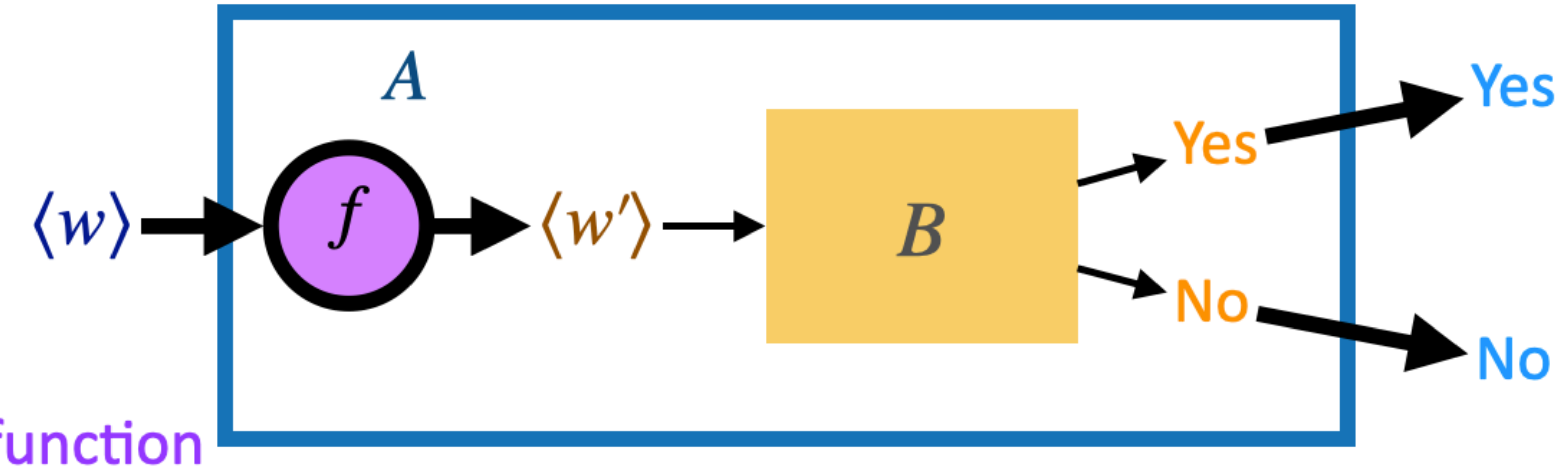
$\Uparrow$

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

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



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



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

3. Show that for any no-instance $w' \notin B$, the corresponding instance w is also a no-instance of A

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

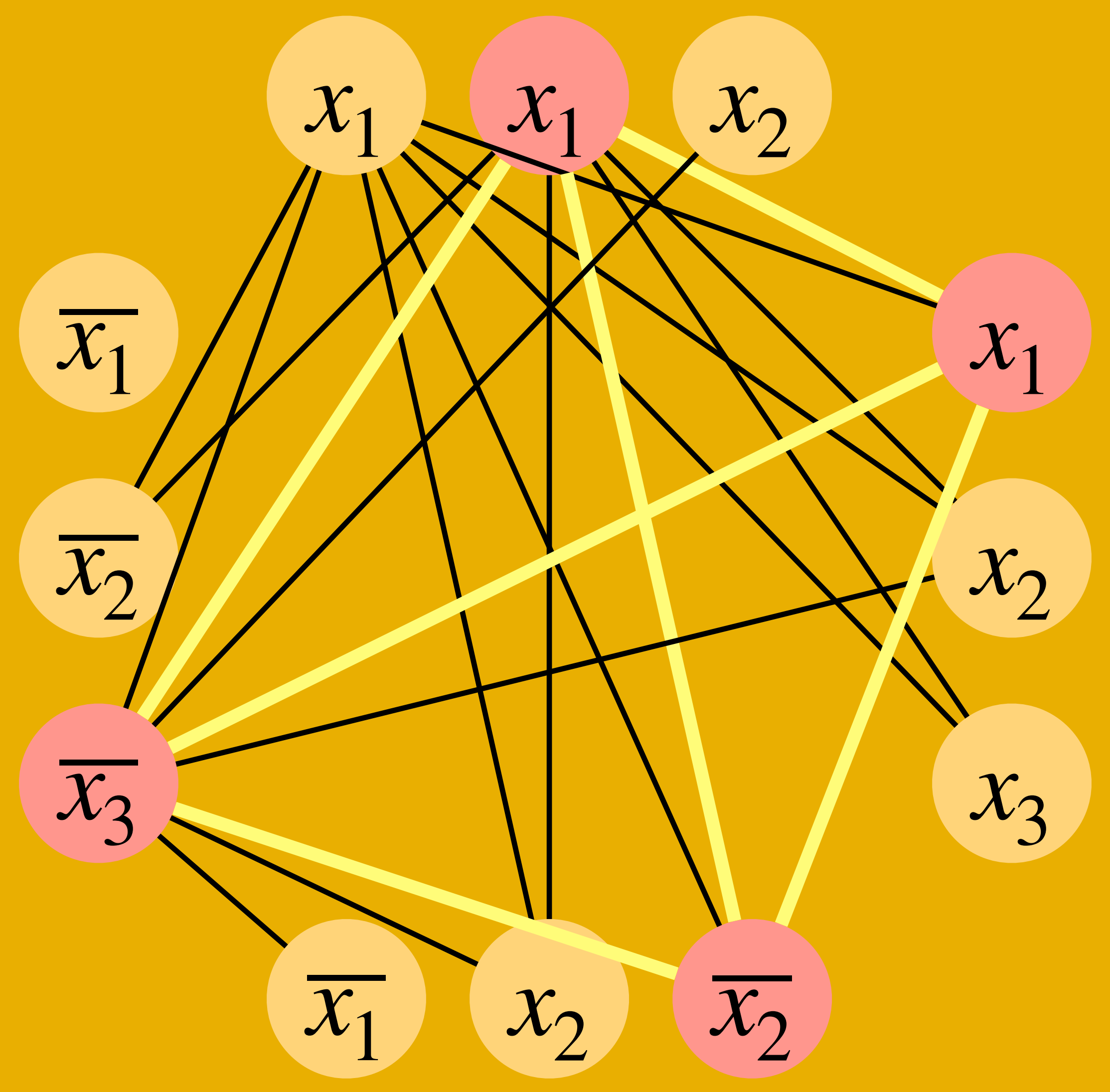
satisfiable $\Rightarrow$ There is a truth assignment such that there is at least one TRUE in each clause

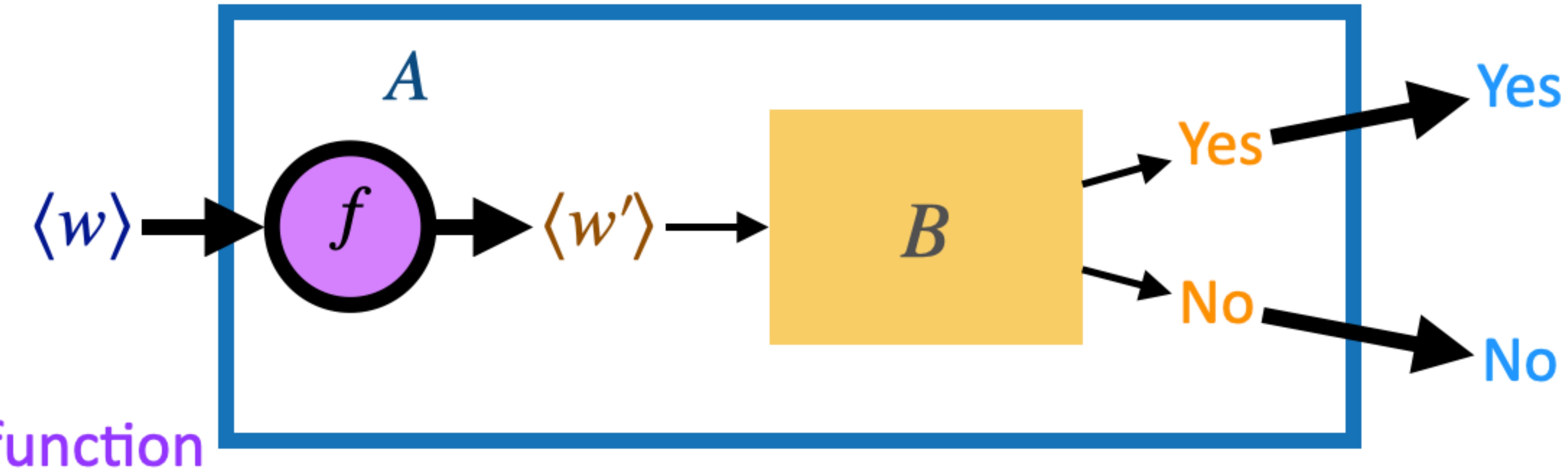
$\Downarrow$

Put the corresponding vertices in the clique

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

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





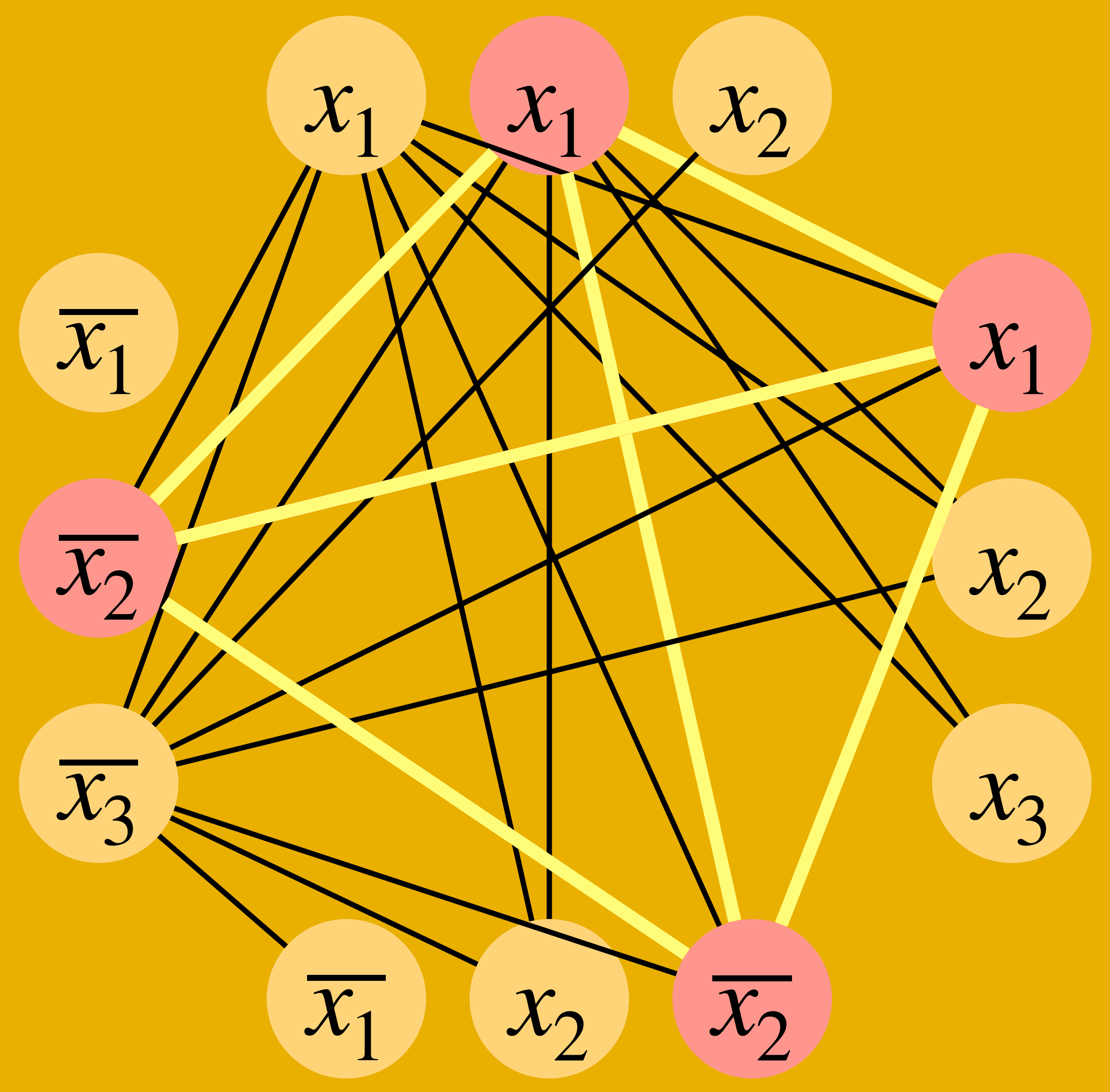
1. Show that there is a function that transforms every w to w' in polynomial time
2. Show that for any yes-instance $w' \in B$, the corresponding instance w is also a yes-instance of A
3. Show that for any no-instance $w' \notin B$, the corresponding instance w is also a no-instance of A

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

Set the corresponding variables as TRUE
There is at least one TRUE in each clause

$\Uparrow$

If there is a m-clique in G



CLIQUE

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

<Proof> Polynomial-time reduction from 3SAT

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

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

CLIQUE

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

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

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

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

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

CLIQUE

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

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

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

CLIQUE

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

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

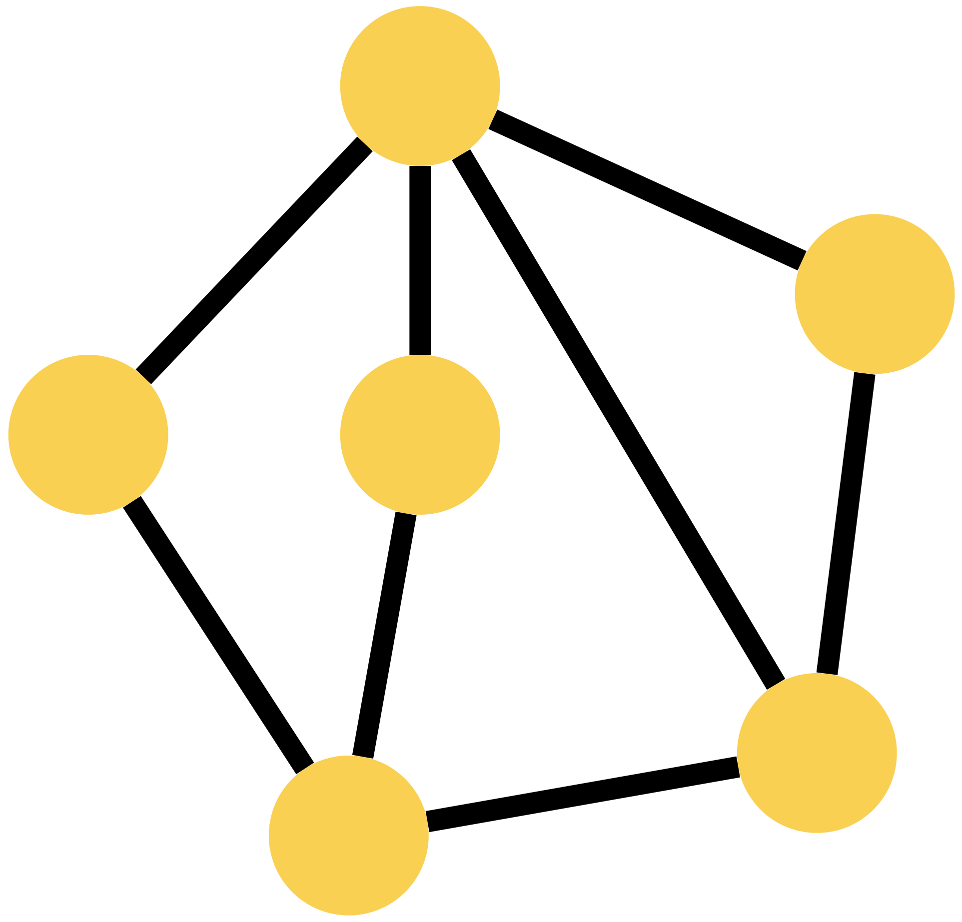
$\blacksquare$

Outline

- More NP-Hardness proofs
  - 3SAT $\leq_p$ CLIQUE
  - VERTEX-COVER $\leq_p$ FEEDBACK-VERTEX-SET
  - PARTITION $\leq_p$ BIN-PACKING
- Pseudo-polynomial time algorithms
- **NP** and **Co-NP**

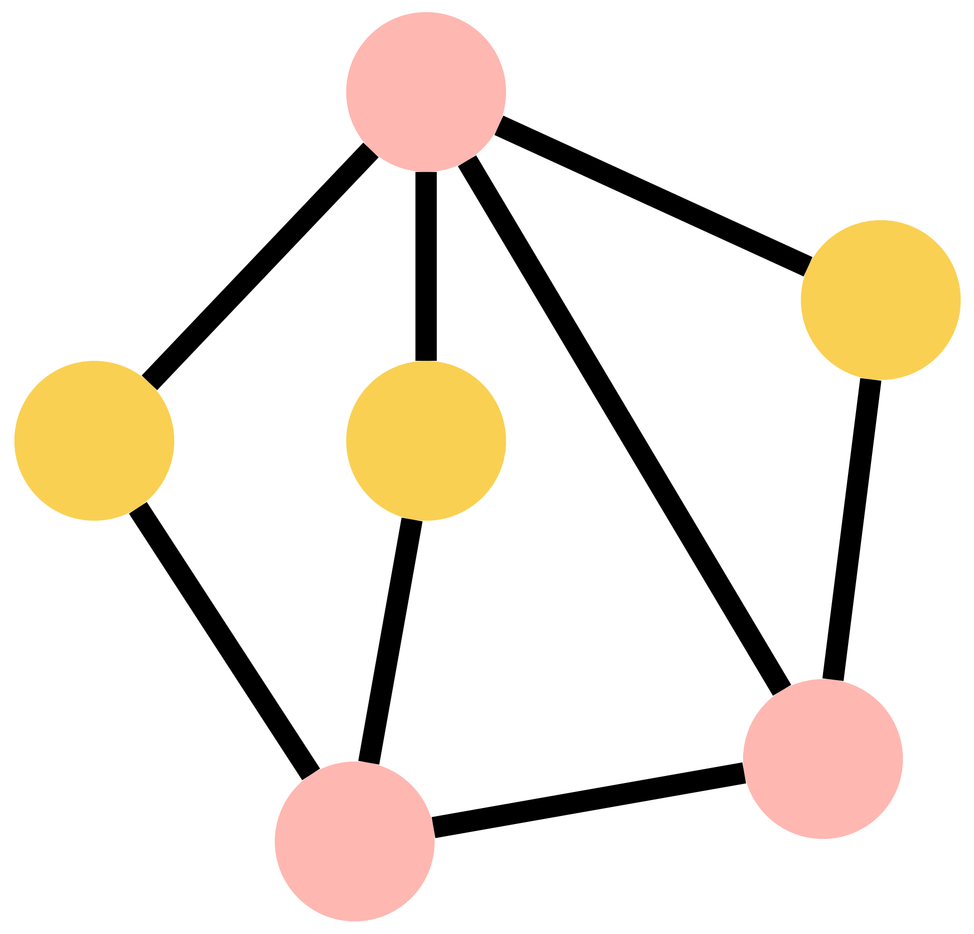
Vertex Cover

- Given a graph $G = (V, E)$, a *vertex cover* is a subset U of vertices such that for every edge (u, v), $|\{u, v\} \cap U| \geq 1$
  - That is, every edge is *covered* by at least one of its endpoints
  - Removing all vertices in U leaves no edge
    - When a vertex is removed, all the edges incident to it are also removed



Vertex Cover

- Given a graph $G = (V, E)$, a *vertex cover* is a subset U of vertices such that for every edge (u, v), $|\{u, v\} \cap U| \geq 1$
  - That is, every edge is *covered* by at least one of its endpoints
  - Removing all vertices in U leaves no edge
    - When a vertex is removed, all the edges incident to it are also removed



Vertex Cover

- Given a graph $G = (V, E)$, a *vertex cover* is a subset U of vertices such that for every edge (u, v), $|\{u, v\} \cap U| \geq 1$
  - That is, every edge is *covered* by at least one of its endpoints
  - Removing all vertices in U leaves no edge
    - When a vertex is removed, all the edges incident to it are also removed

Vertex Cover

- Minimum vertex cover problem: Given a graph G, what is the size of the minimum vertex cover in G?

- Decision version: Given a graph G, is there a vertex cover of size at most k in G?
  - An instance of VERTEX-COVER is $\langle\langle G\rangle, k\rangle$

New parameter!

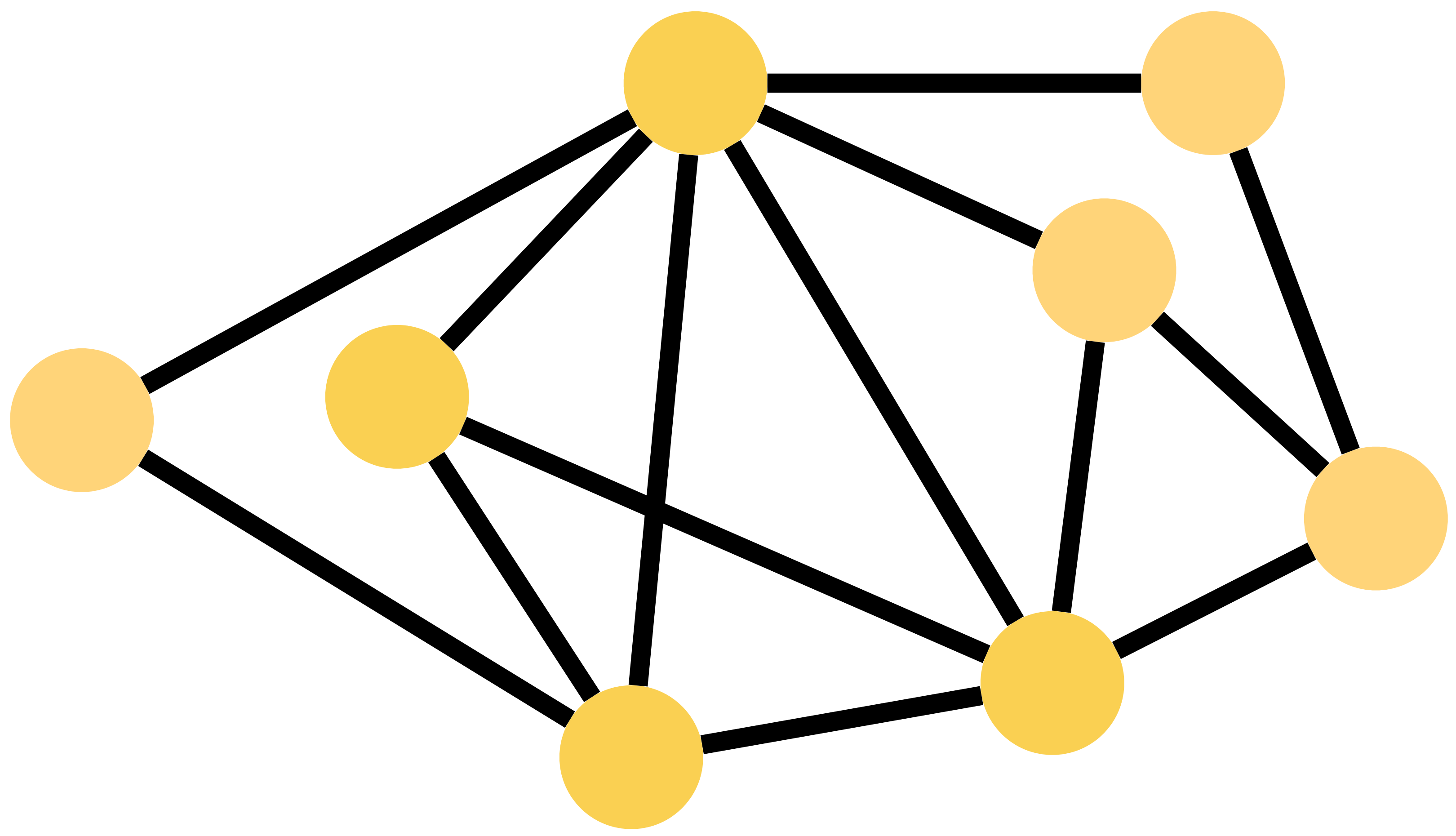
- VERTEX-COVER is NP-complete

Feedback Vertex Set (FVS)

- Given a graph $G = (V, E)$, a *feedback vertex set* is a subset U of vertices such that removing the vertices in U leaves a graph without cycles

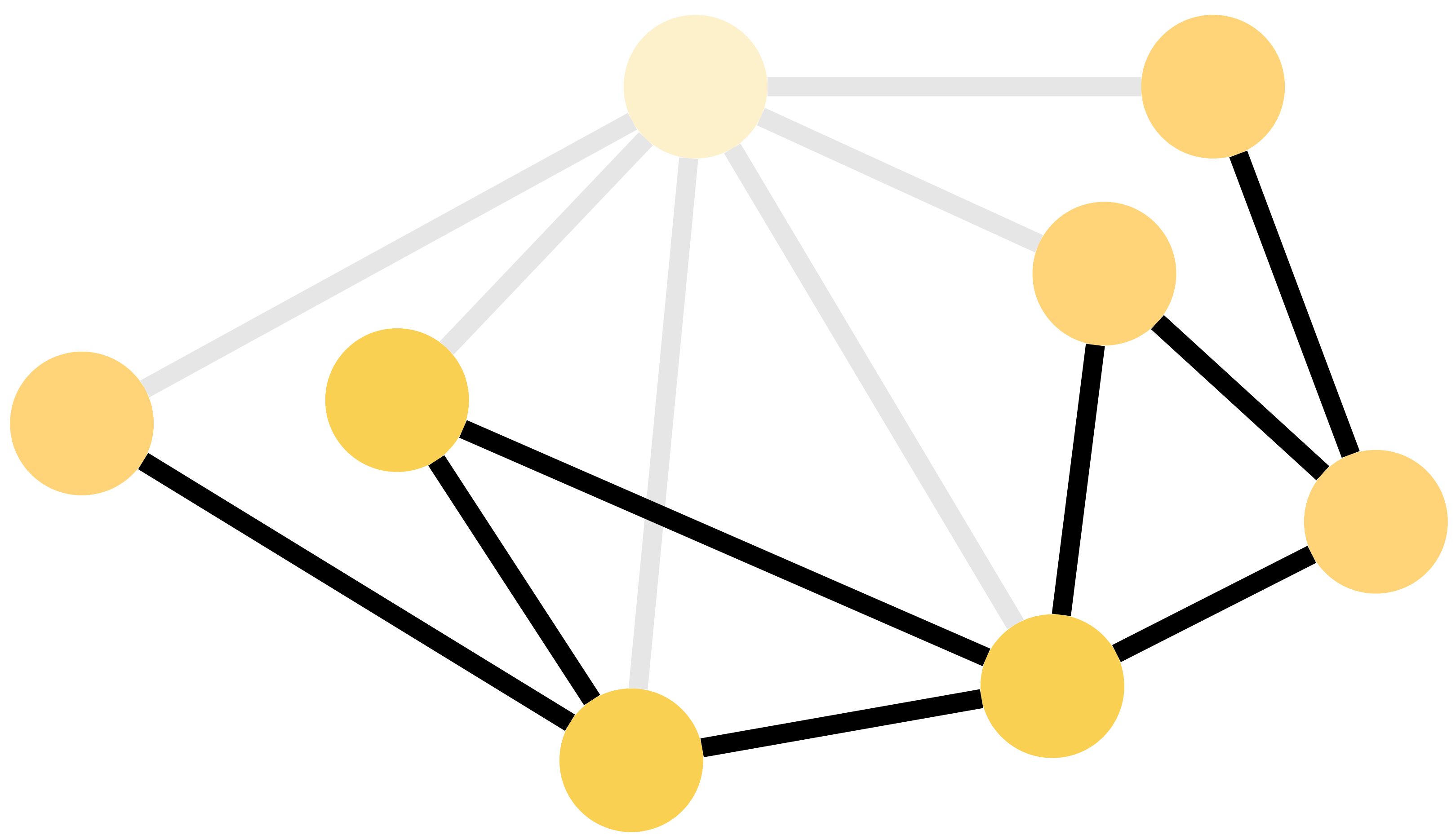
Feedback Vertex Set (FVS)

- Given a graph $G = (V, E)$, a *feedback vertex set* is a subset U of vertices such that removing the vertices in U leaves a graph without cycles
  - When a vertex is removed, all the edges incident to it are also removed



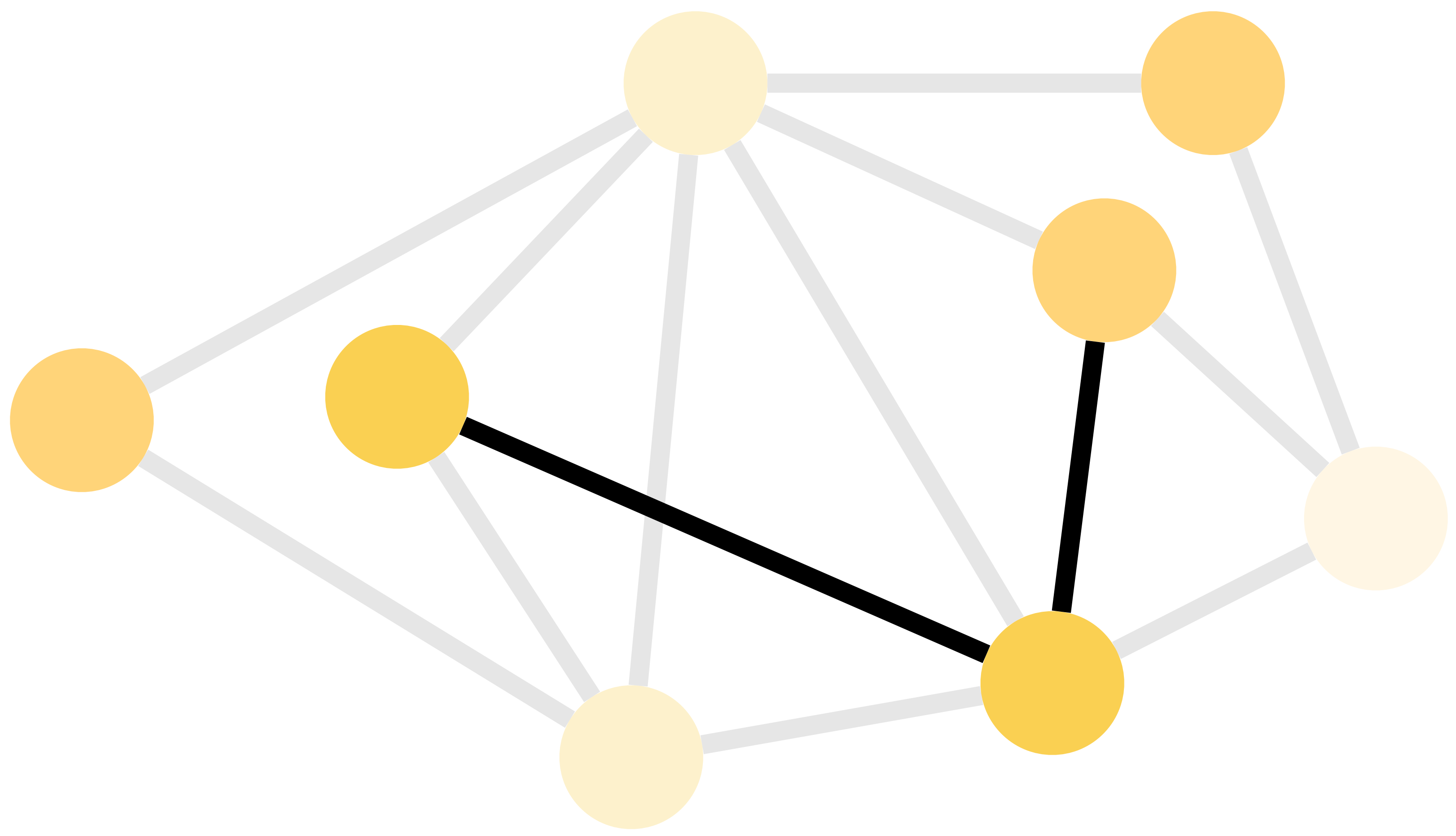
Feedback Vertex Set (FVS)

- Given a graph $G = (V, E)$, a *feedback vertex set* is a subset U of vertices such that removing the vertices in U leaves a graph without cycles
  - When a vertex is removed, all the edges incident to it are also removed



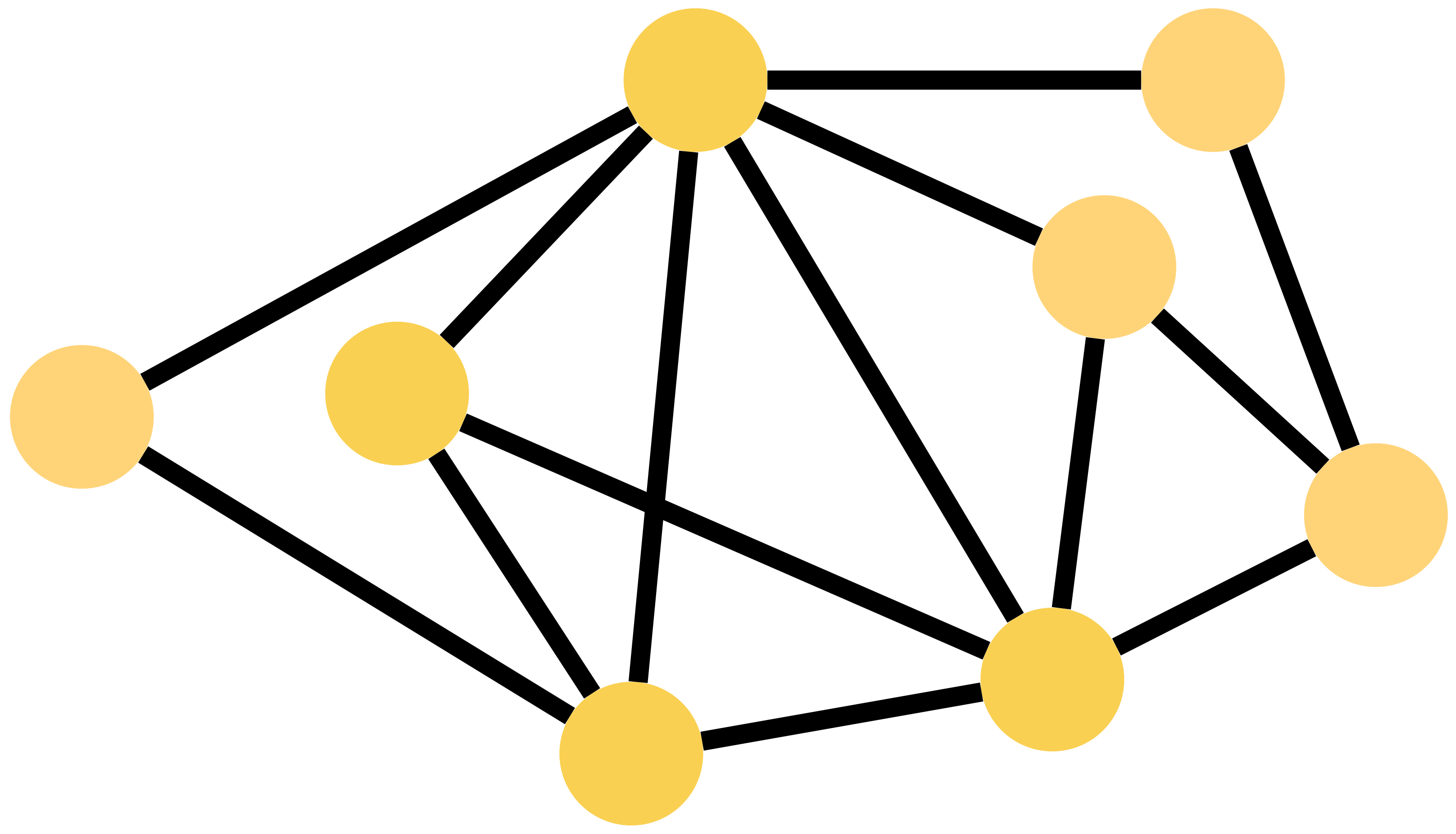
Feedback Vertex Set (FVS)

- Given a graph $G = (V, E)$, a *feedback vertex set* is a subset U of vertices such that removing the vertices in U leaves a graph without cycles
  - When a vertex is removed, all the edges incident to it are also removed



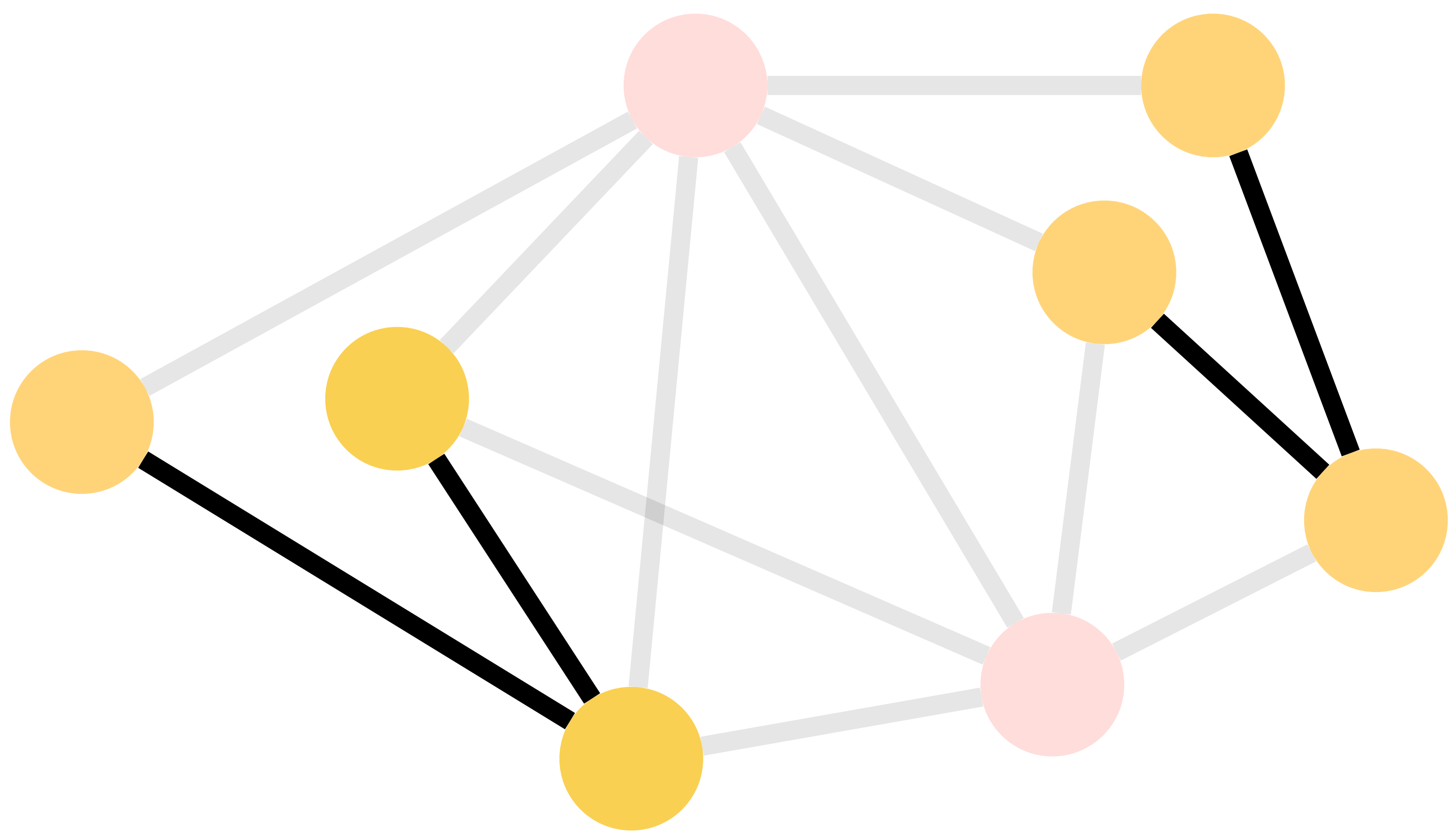
Feedback Vertex Set (FVS)

- Minimum-FVS: Given a graph $G = (V, E)$, what is the size of its minimum feedback vertex set?



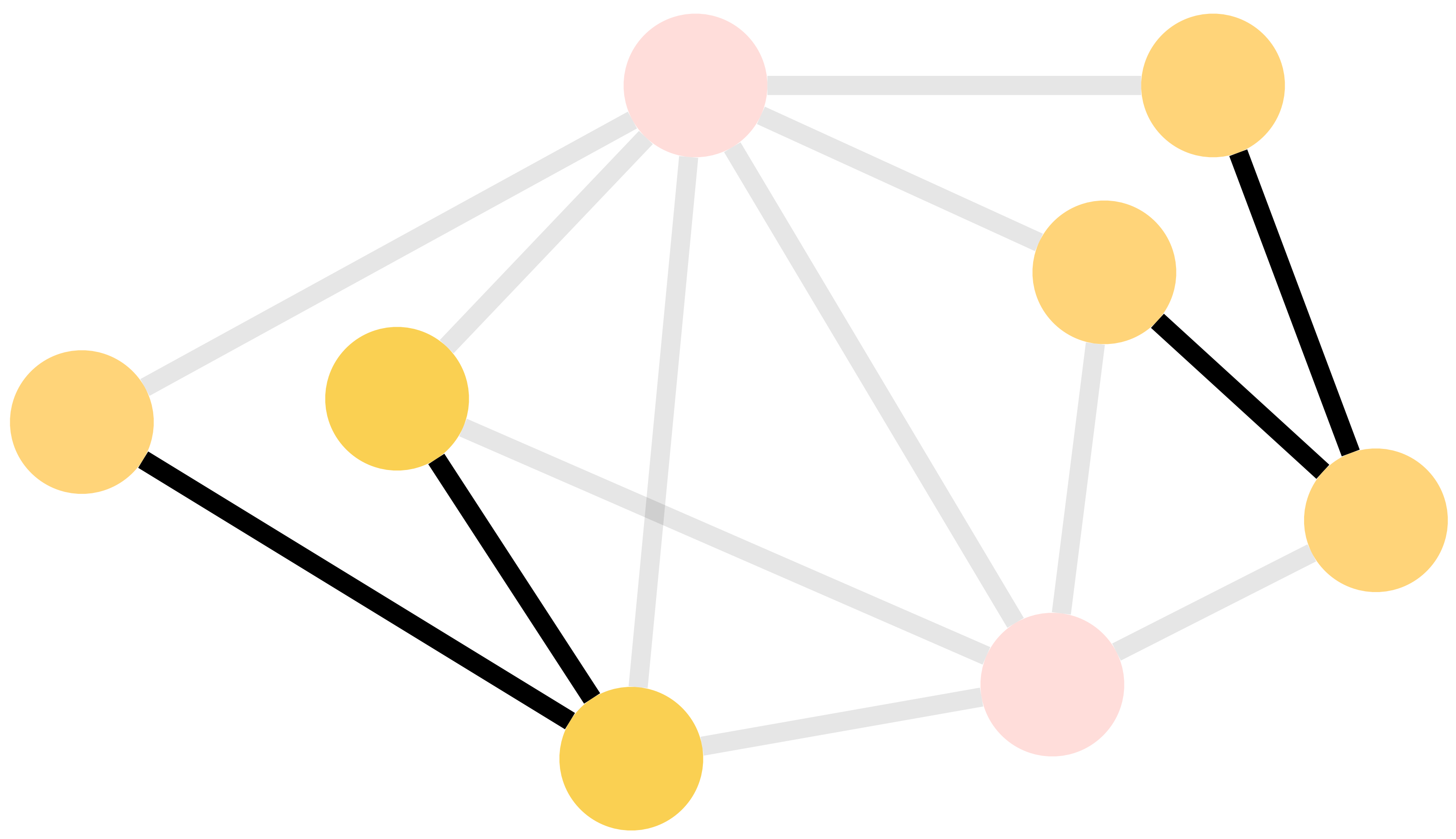
Feedback Vertex Set (FVS)

- Minimum-FVS: Given a graph $G = (V, E)$, what is the size of its minimum feedback vertex set?



Feedback Vertex Set (FVS)

- Minimum-FVS: Given a graph $G = (V, E)$, what is the size of its minimum feedback vertex set?
- Decision version of Minimum-FVS problem:
    - Given a graph $G = (V, E)$, is there a feedback vertex set with size at most k?



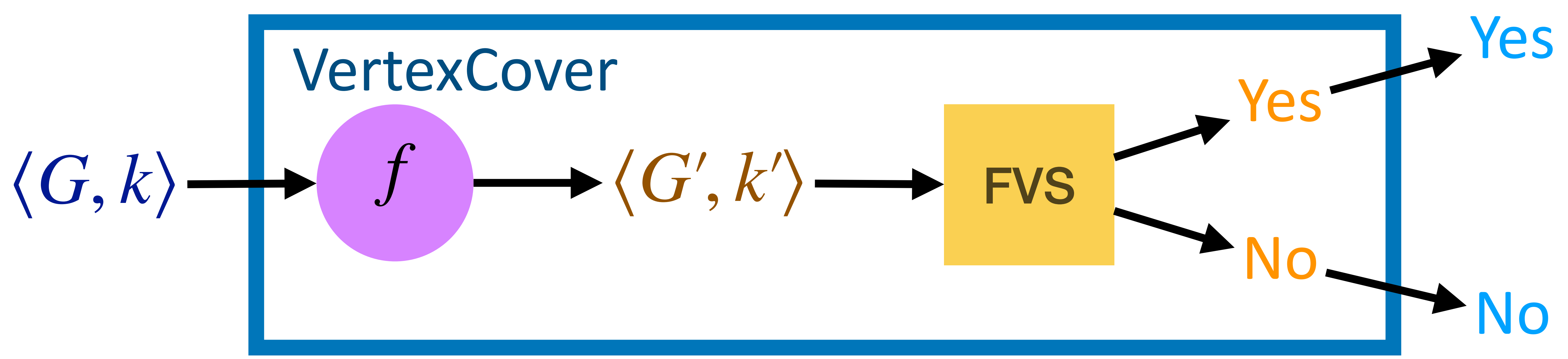
Feedback Vertex Set (FVS)

- Minimum-FVS: Given a graph $G = (V, E)$, what is the size of its minimum feedback vertex set?
- Decision version of Minimum-FVS problem:
  - Given a graph $G = (V, E)$, is there a feedback vertex set with size at most k?

- Theorem: FVS is NP-complete

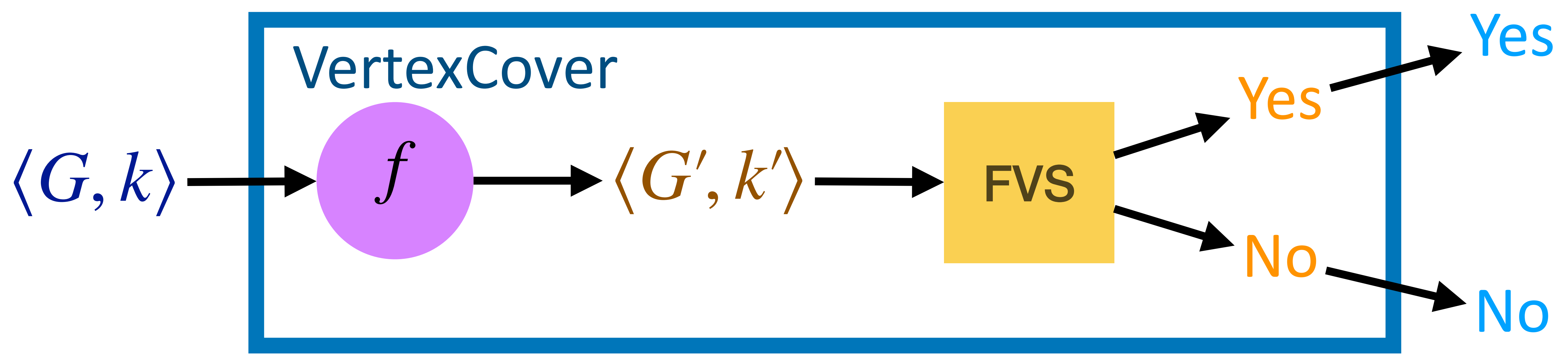
Feedback Vertex Set (FVS)

- VertexCover $= \{\langle G, k\rangle \mid$ There is a vertex cover in G with size at most $k\}$
- FVS $= \{\langle G', k'\rangle \mid$ There is a feedback vertex set in G' with size at most $k'\}$



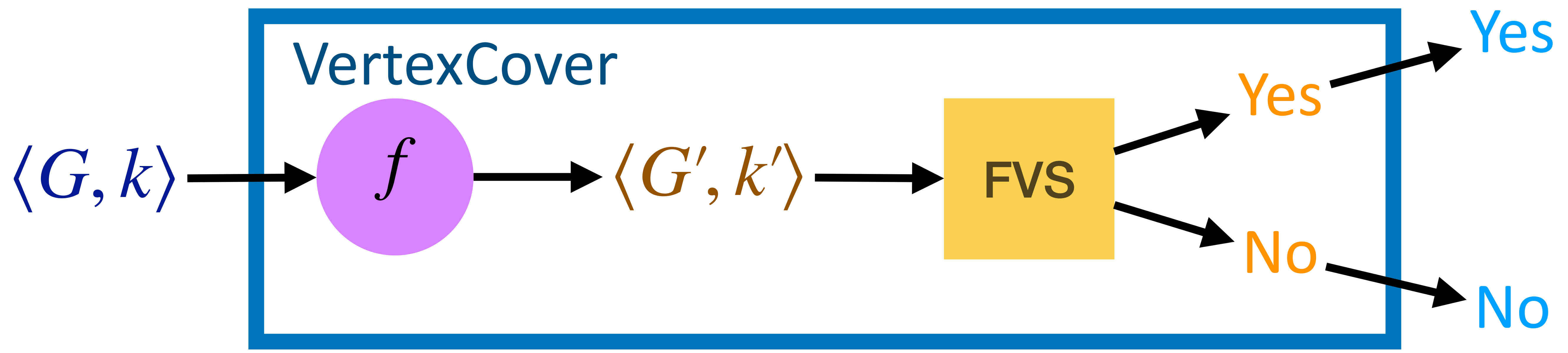
Feedback Vertex Set (FVS)

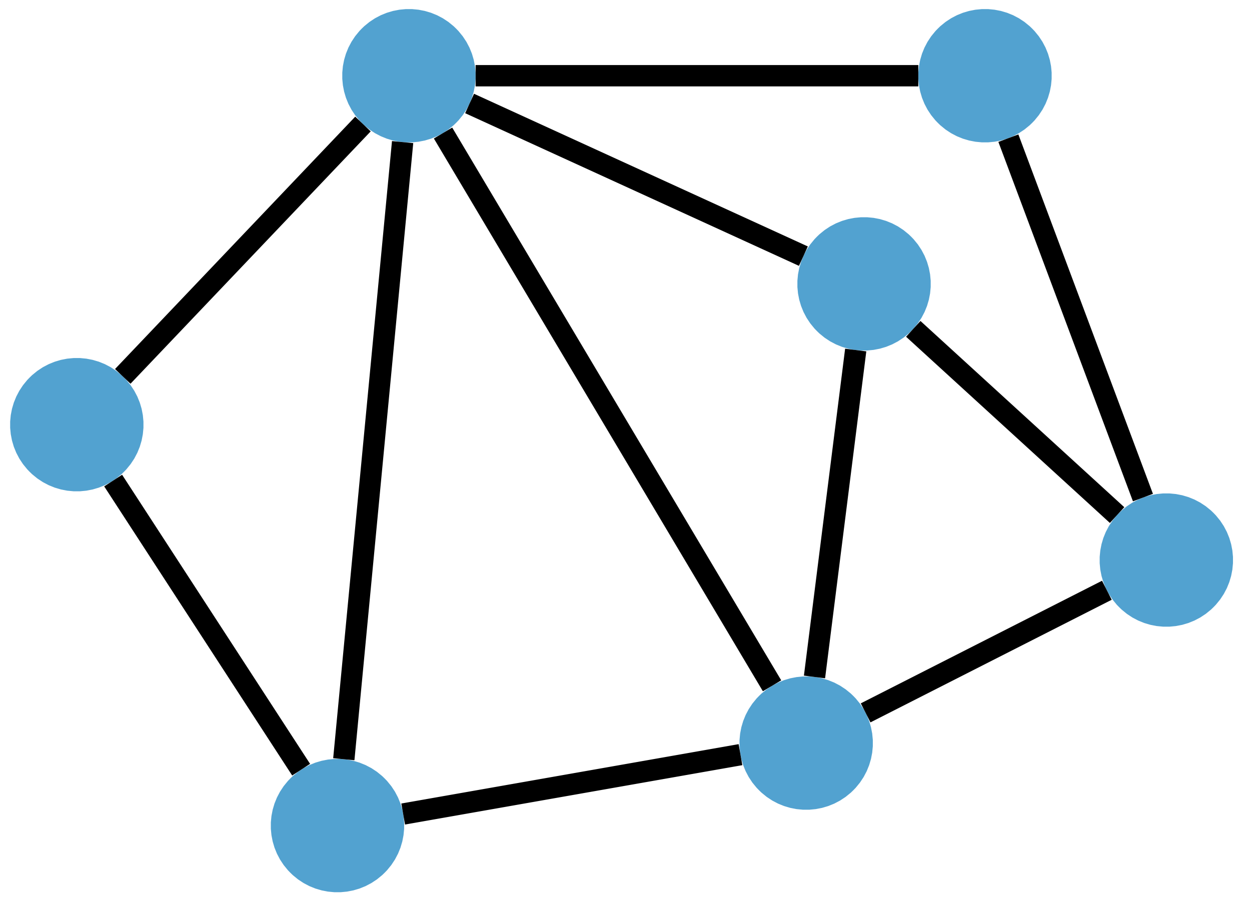
- VertexCover $= \{\langle G, k\rangle \mid$ There is a set of at most k vertices in G such that removing them leaves no edges$\}$
- FVS $= \{\langle G', k'\rangle \mid$ There is a set of at most k' vertices in G' such that removing them leaves no cycles$\}$

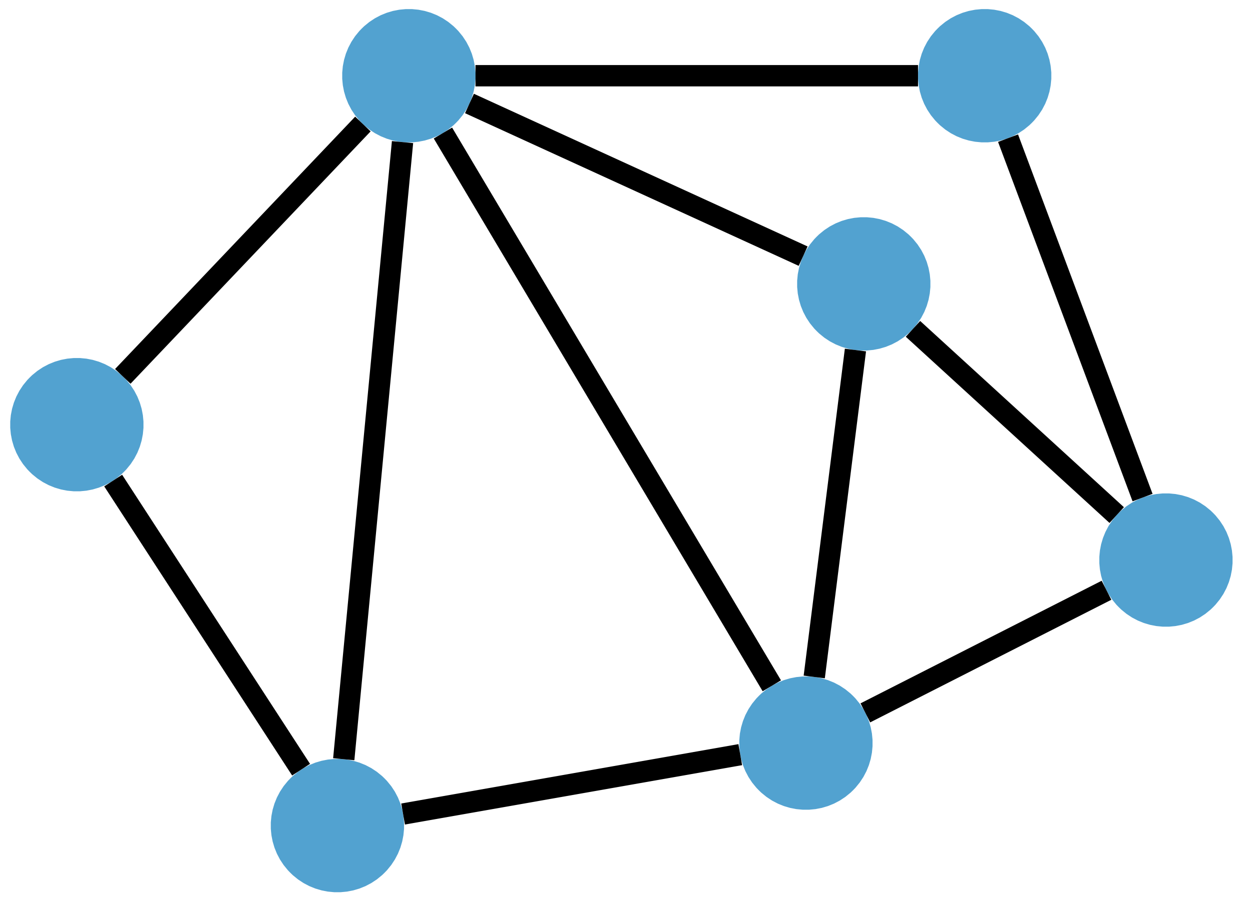


Feedback Vertex Set (FVS)

- VertexCover $= \{\langle G, k\rangle \mid$ There is a set of at most k vertices in G such that removing them leaves no edges$\}$
- FVS $= \{\langle G', k'\rangle \mid$ There is a set of at most k' vertices in G' such that removing them leaves no cycles$\}$

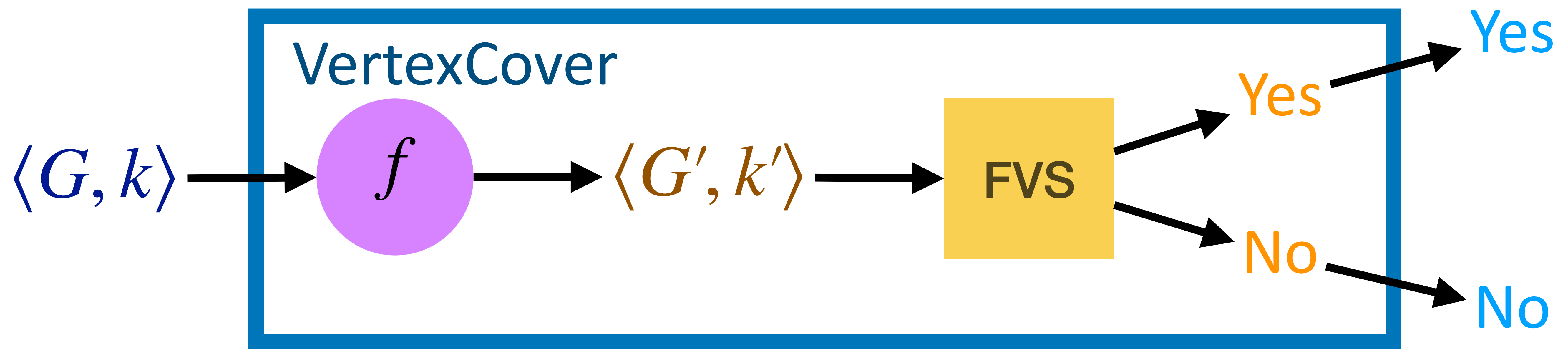


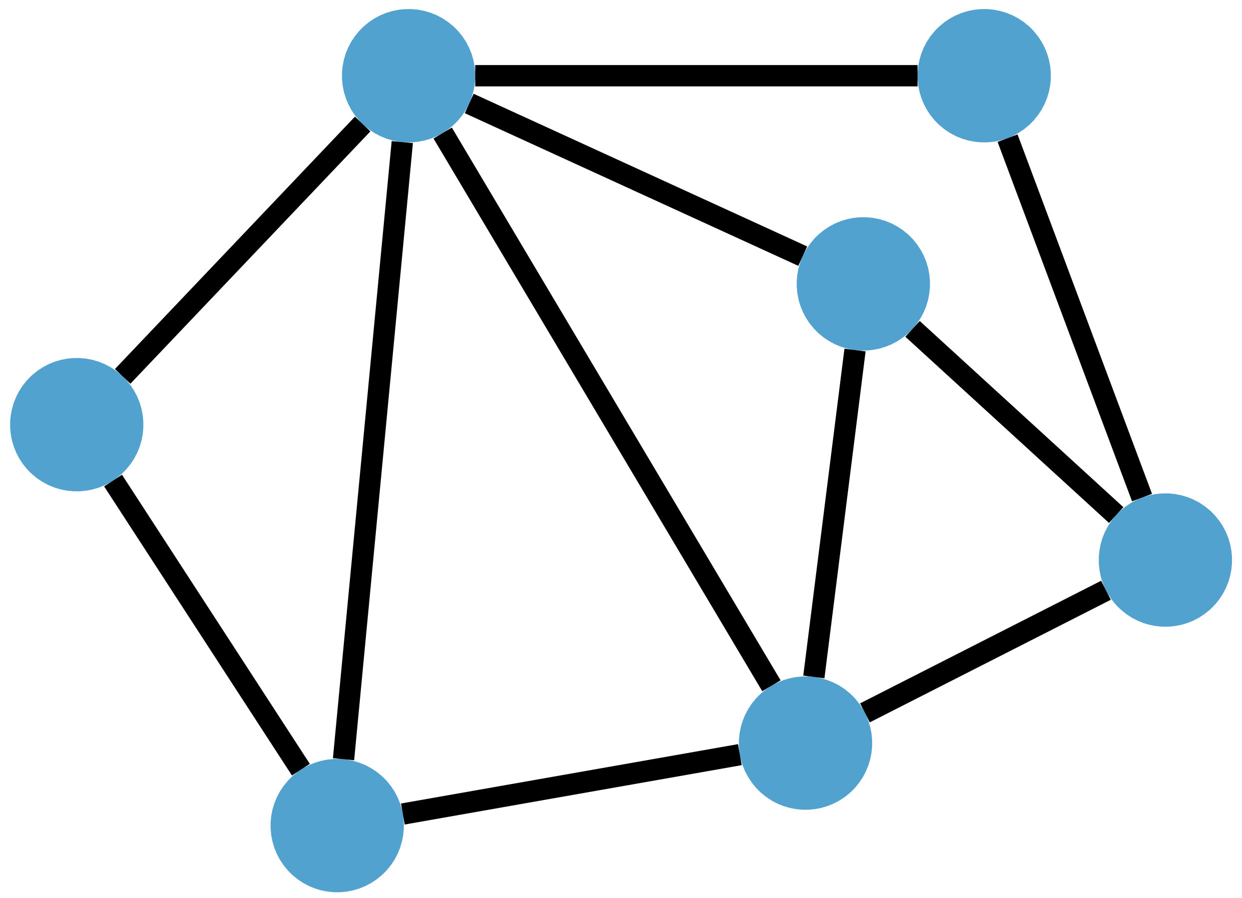


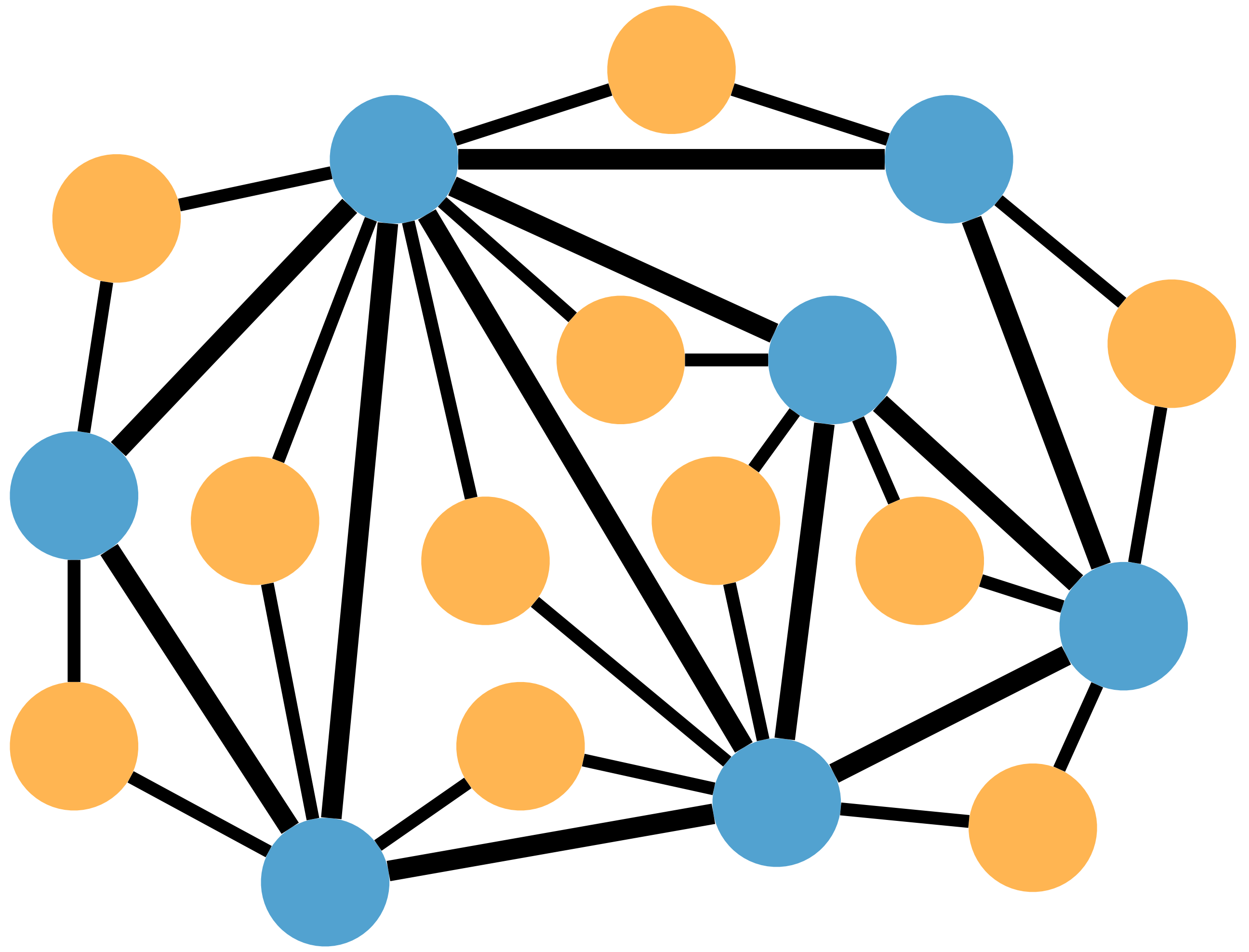


Feedback Vertex Set (FVS)

- VertexCover $= \{\langle G, k\rangle$ | There is a set of at most k vertices in G such that removing them leaves no edges$\}$
- FVS $= \{\langle G', k'\rangle$ | There is a set of at most k' vertices in G' such that removing them leaves no cycles$\}$

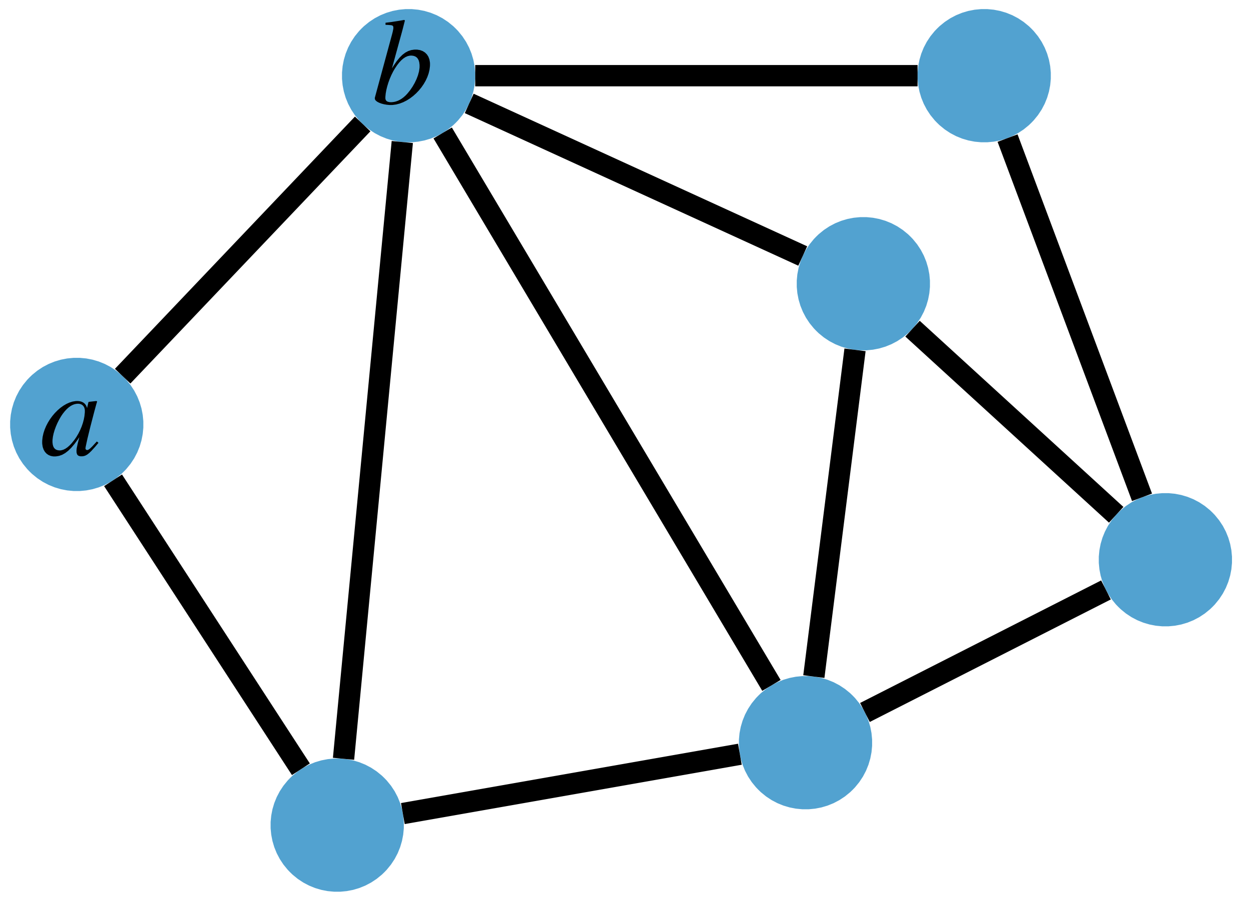




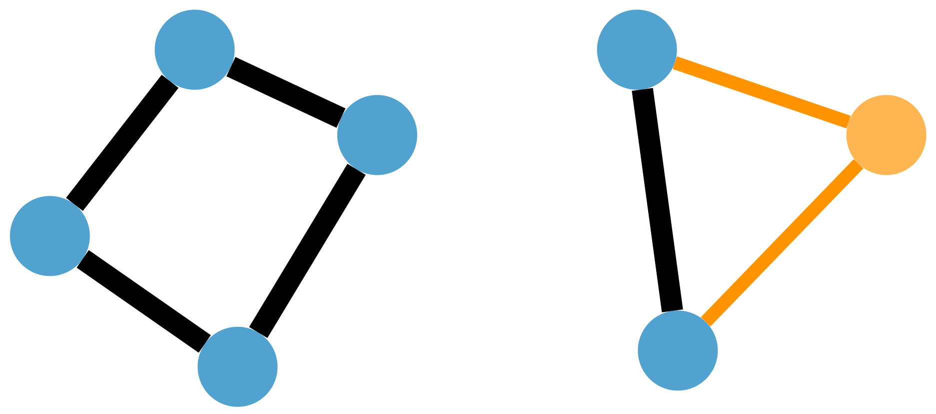


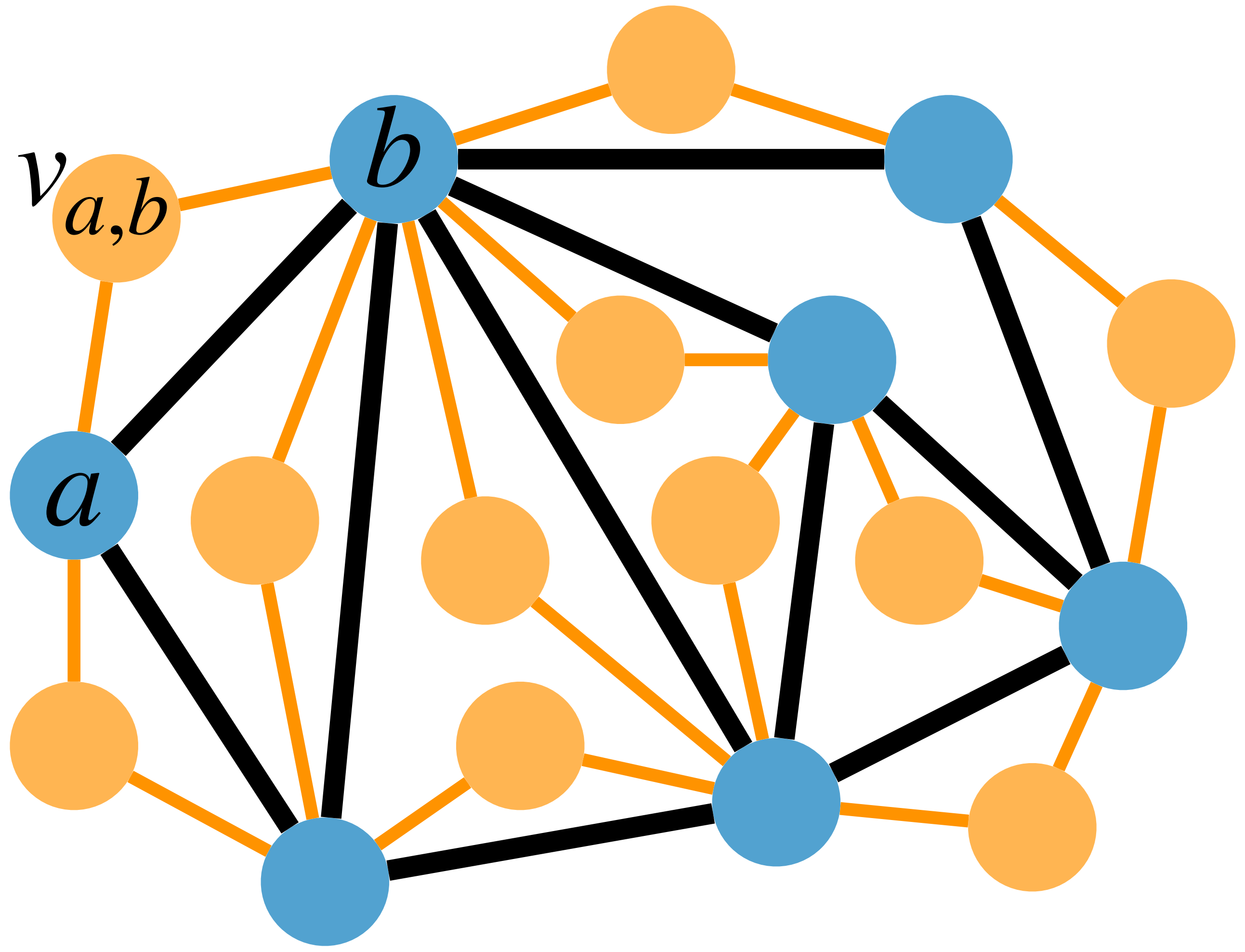
Feedback Vertex Set (FVS)

- VertexCover $= \{\langle G, k \rangle \mid$ There is a set of at most k vertices in G such that removing them leaves no edges$\}$



- FVS $= \{\langle G', k' \rangle \mid$ There is a set of at most k' vertices in G' such that removing them leaves no cycles$\}$





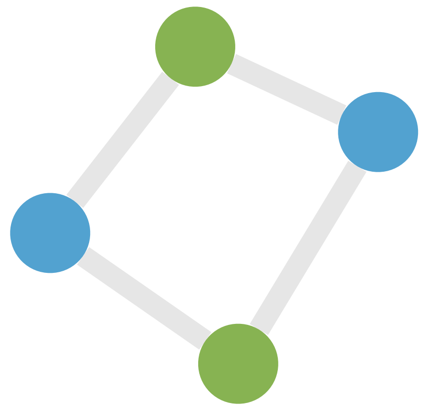
$G' = (V', E')$

$V' = V \cup V_E$

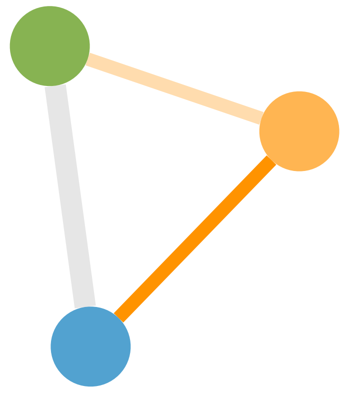
$E' = E \cup E_{V,E}$

Feedback Vertex Set (FVS)

After removing the size-k vertex cover from G':



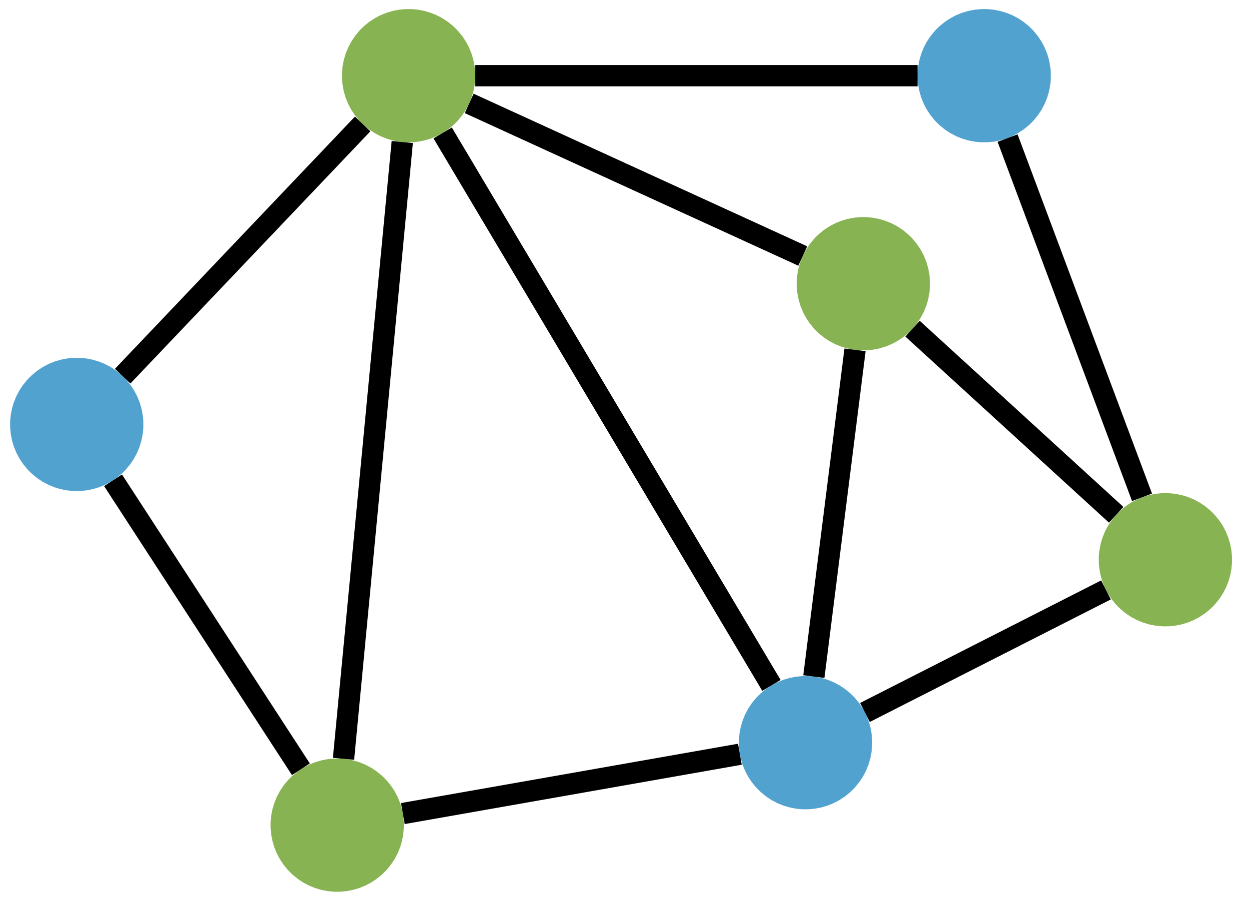
There is no edge between the remaining V vertices

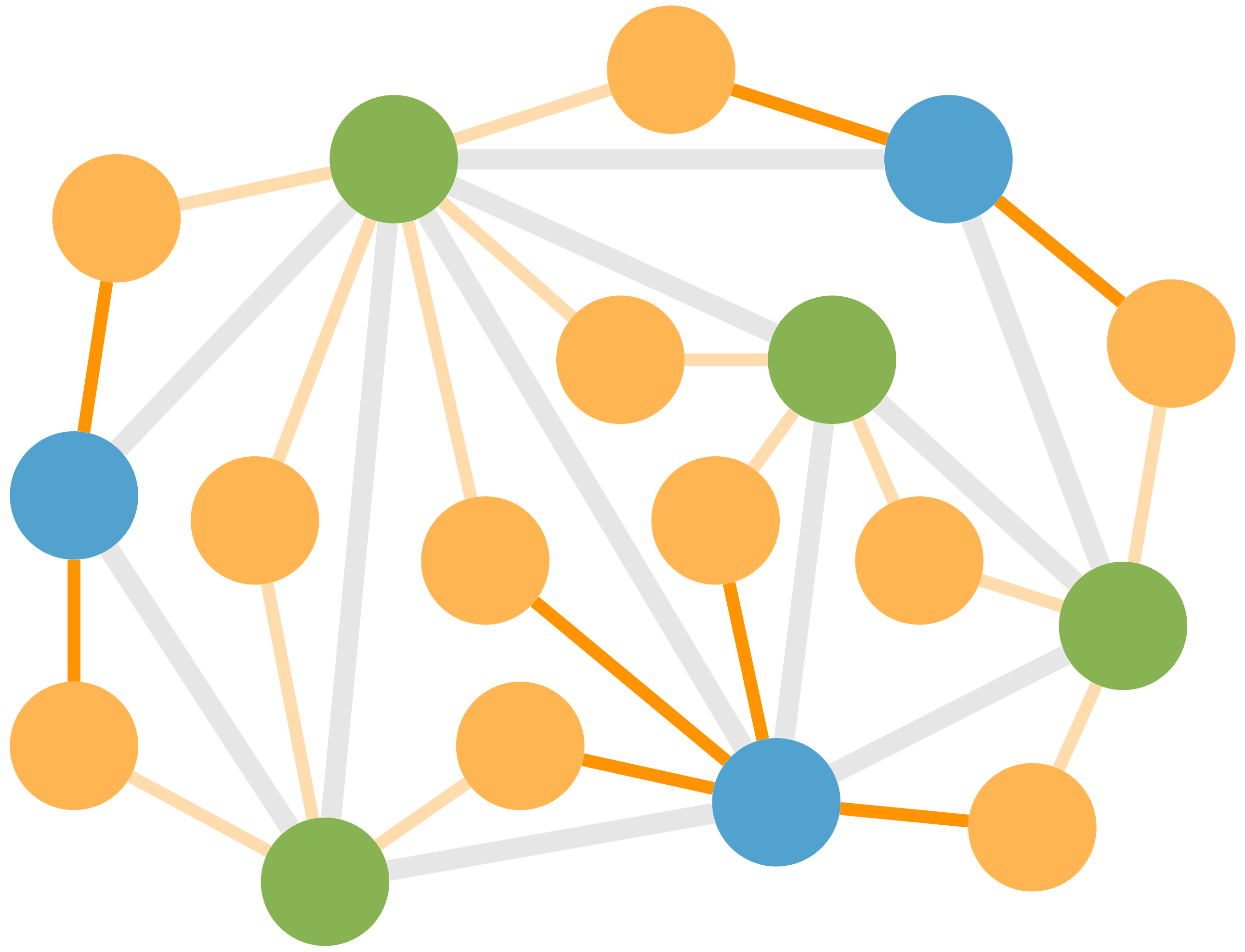


The $E_{V,E}$ vertices have degree at most 1

The size-k vertex cover

The size-k vertex cover is a FVS of G'





$G' = (V', E')$

$V' = V \cup V_E$

$E' = E \cup E_{V,E}$

Feedback Vertex Set (FVS)

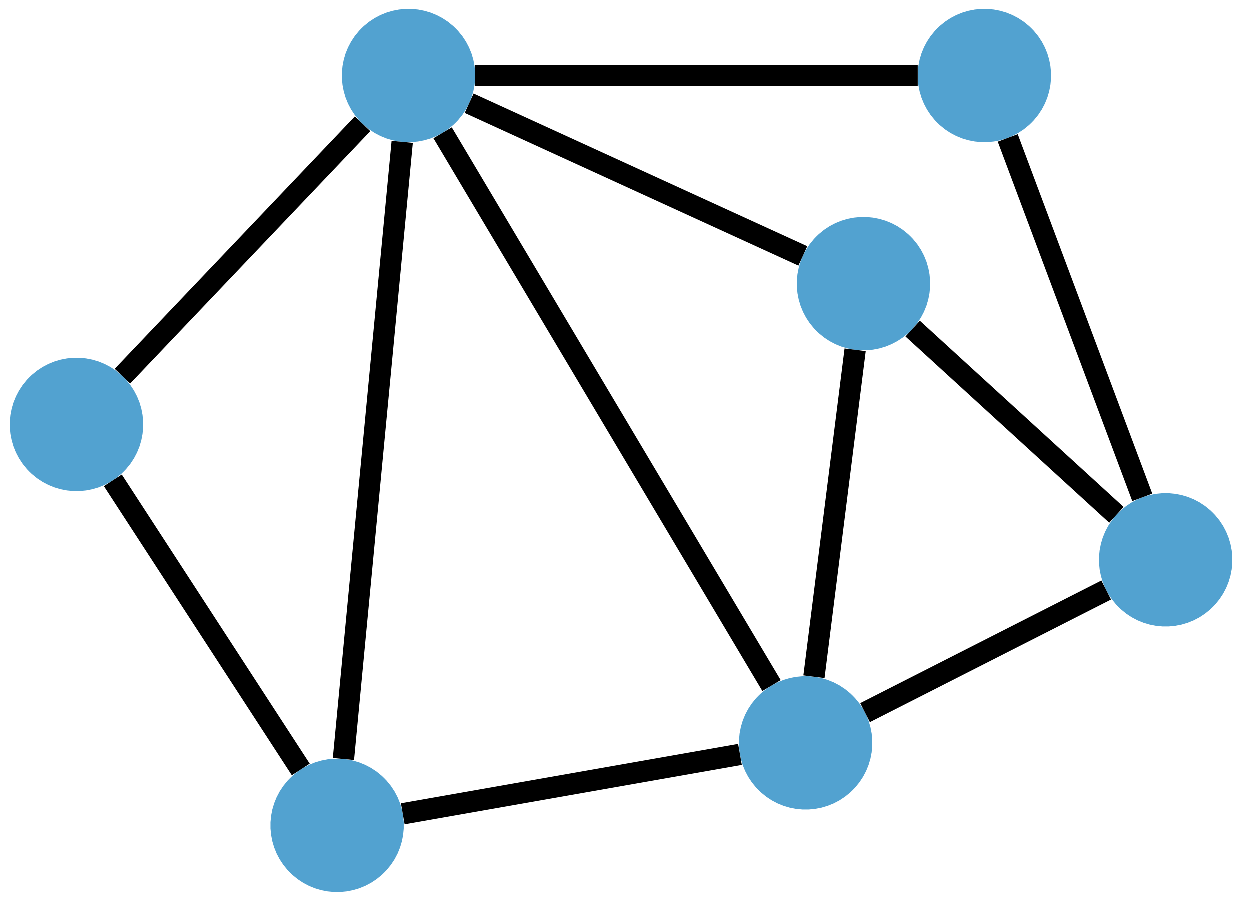
The size-k FVS is a vertex cover of G ⇐ The size-k FVS of G'

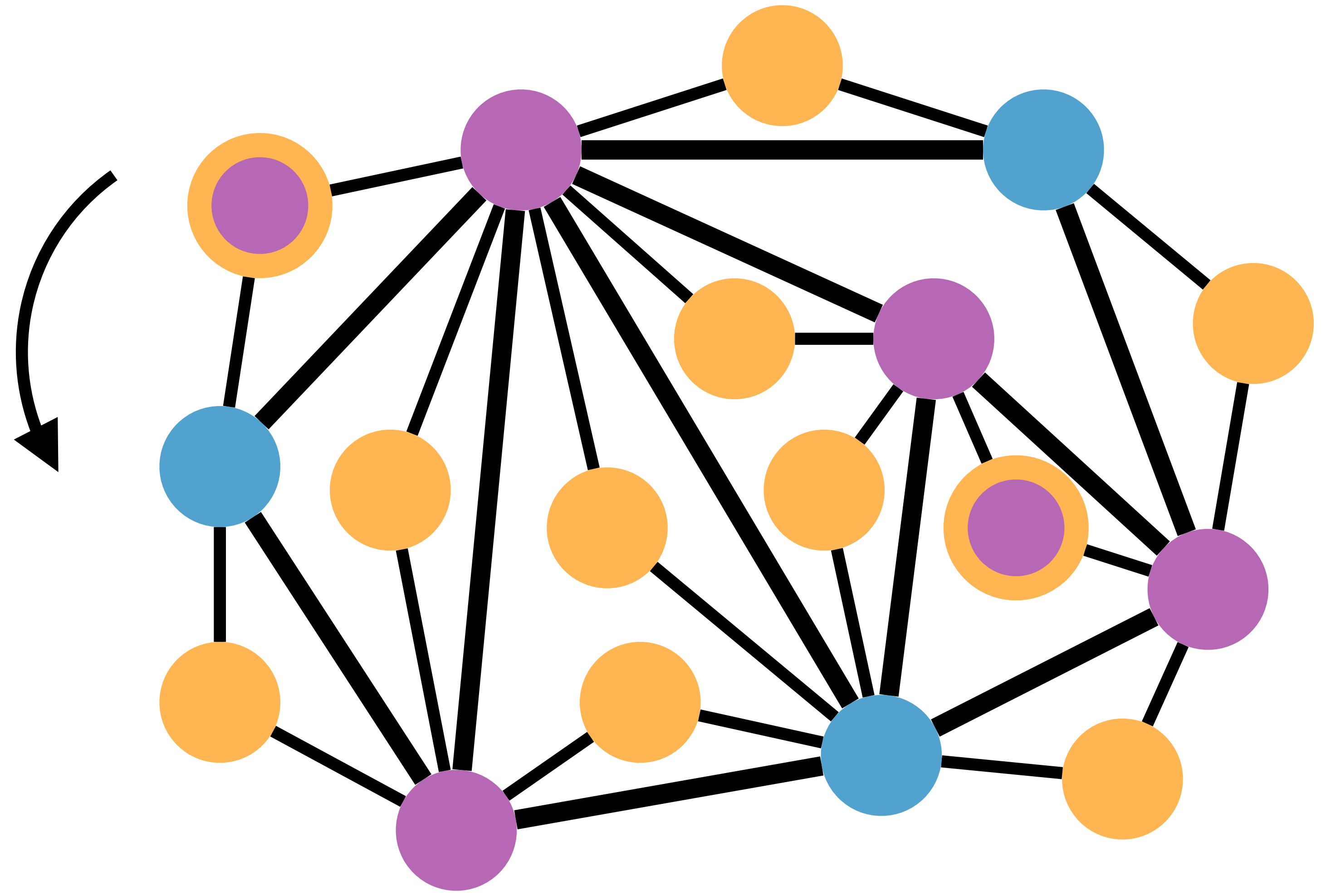
Construct a set F':

Keep all u_i in the FVS

If some $u_{i,j}$ is in the FVS, replace it by u_i or u_j

If both u_i are in u_j the FVS, remove $u_{i,j}$





Feedback Vertex Set (FVS)

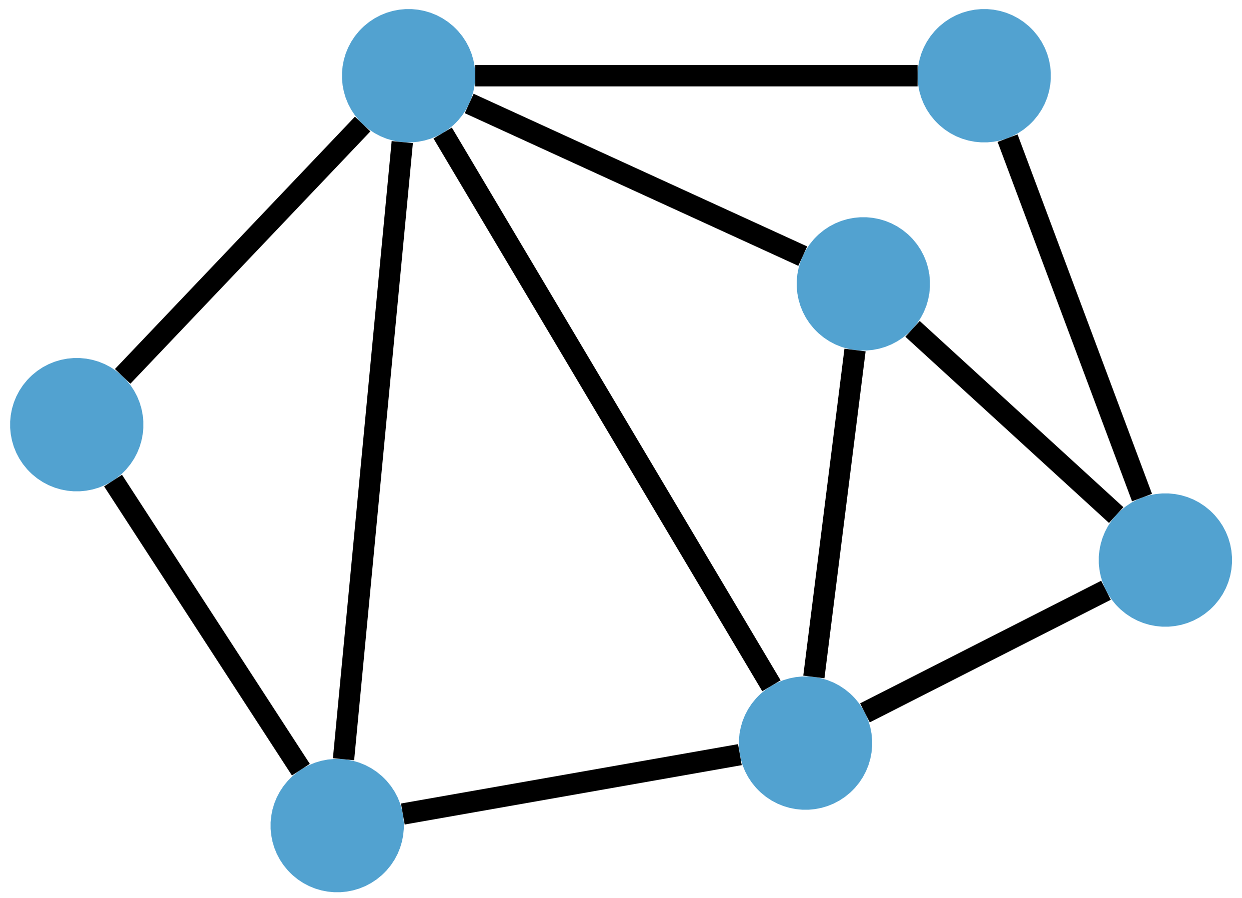
The size-k FVS is a vertex cover of G ⬅ The size-k FVS of G'

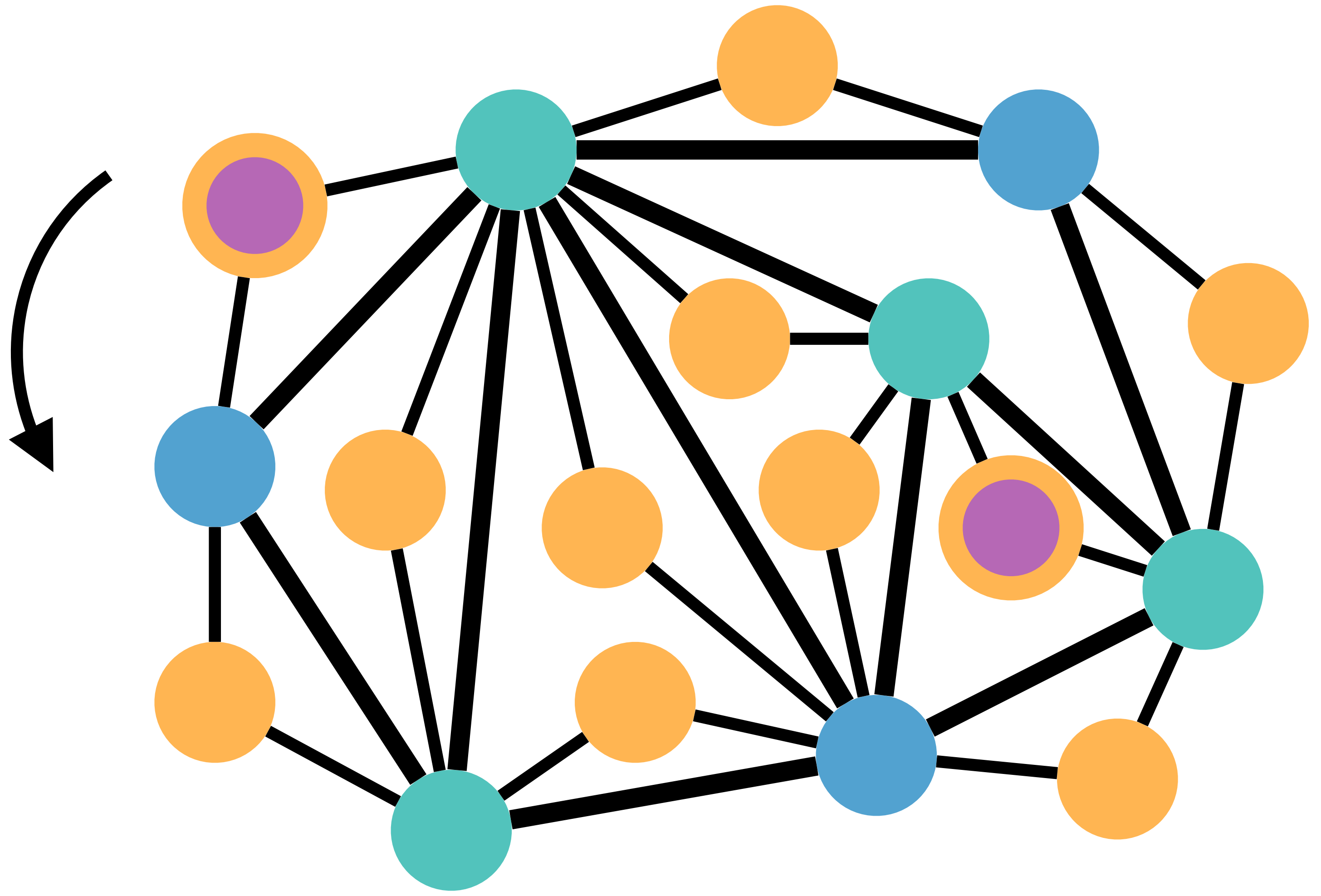
Construct a set F':

Keep all u_i in the FVS

If some $u_{i,j}$ is in the FVS, replace it by u_i or u_j

If both u_i are in u_j the FVS, remove $u_{i,j}$





Feedback Vertex Set (FVS)

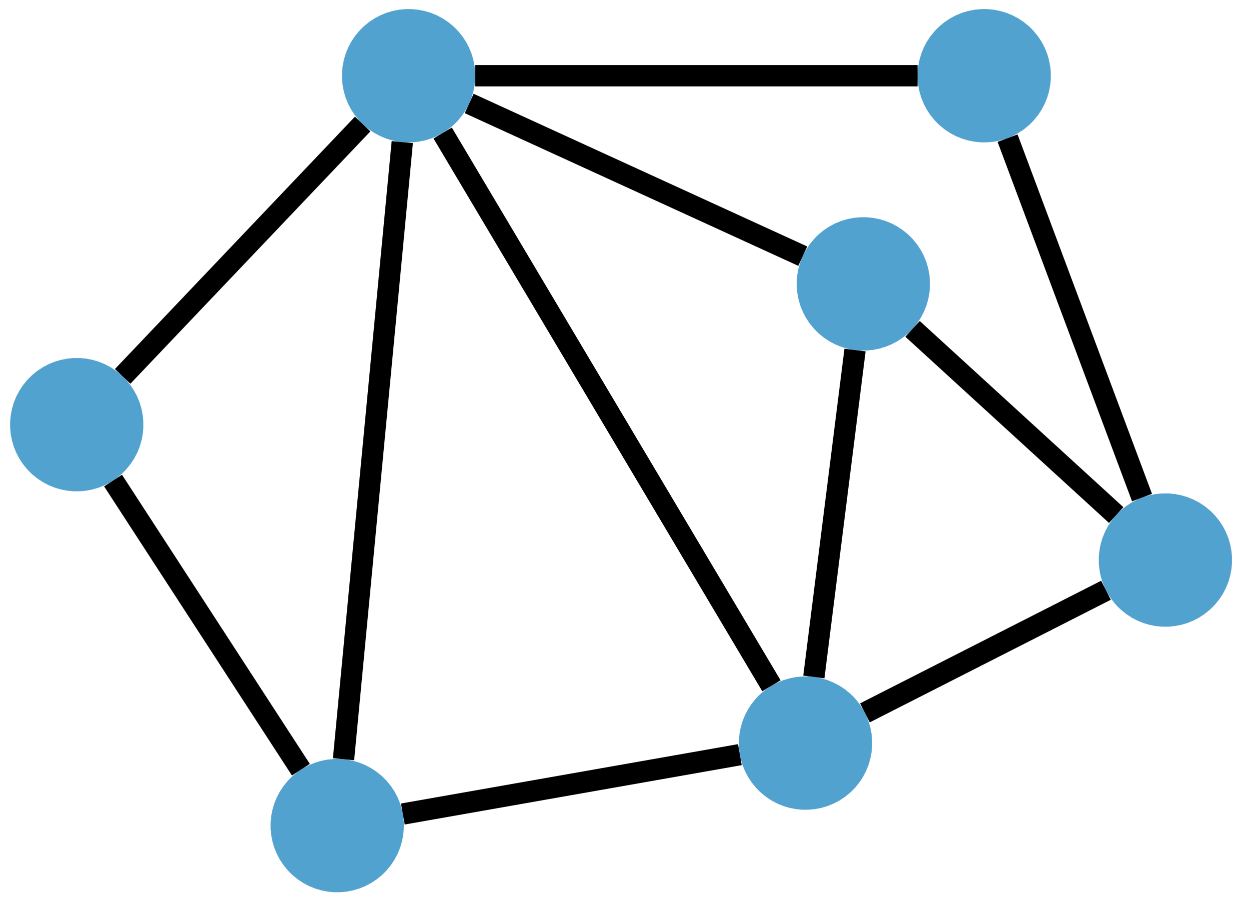
The size-k FVS is a vertex cover of G ⟸ The size-k FVS of G'

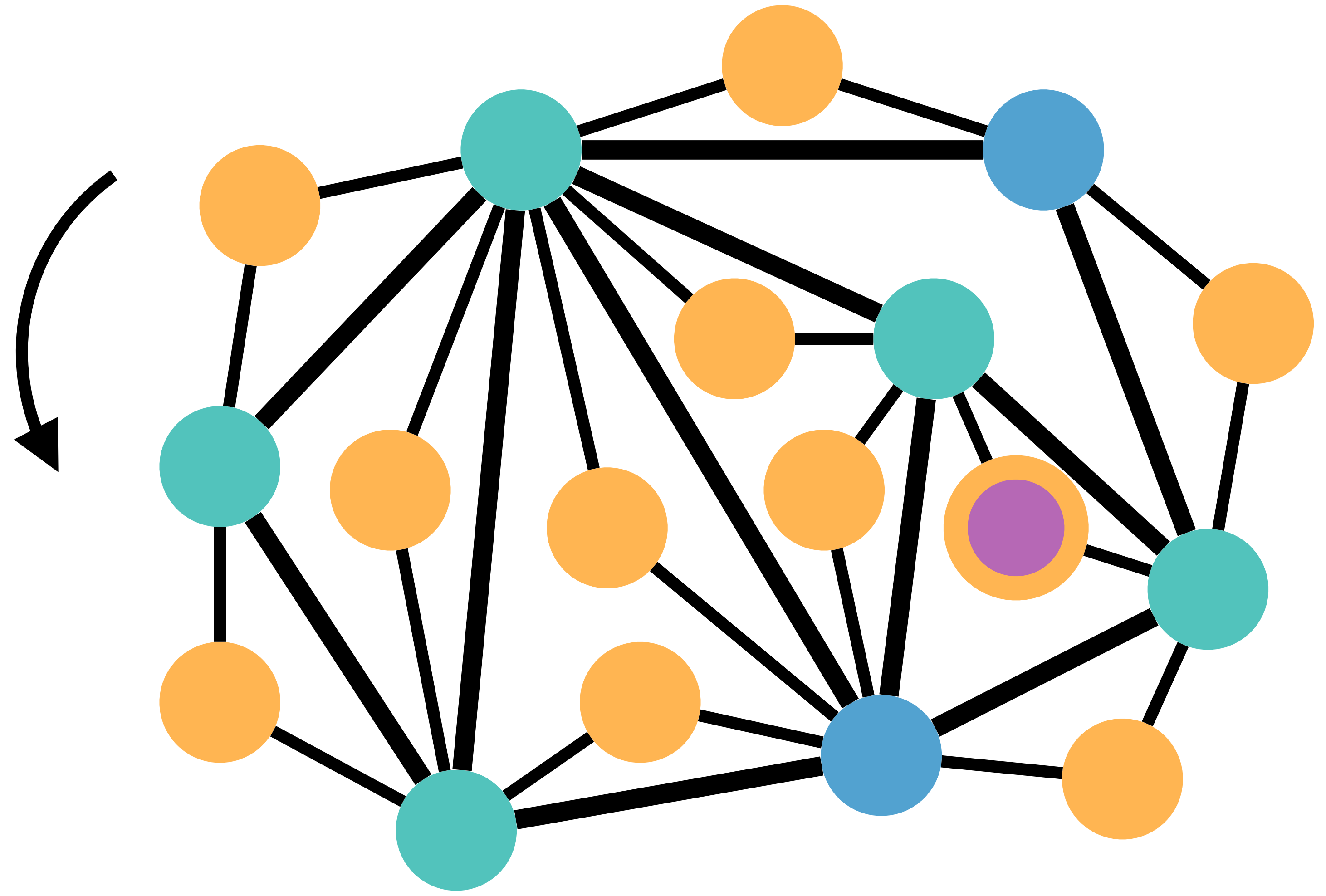
Construct a set F':

Keep all u_i in the FVS

If some $u_{i,j}$ is in the FVS, replace it by u_i or u_j

If both u_i are in u_j the FVS, remove $u_{i,j}$





Feedback Vertex Set (FVS)

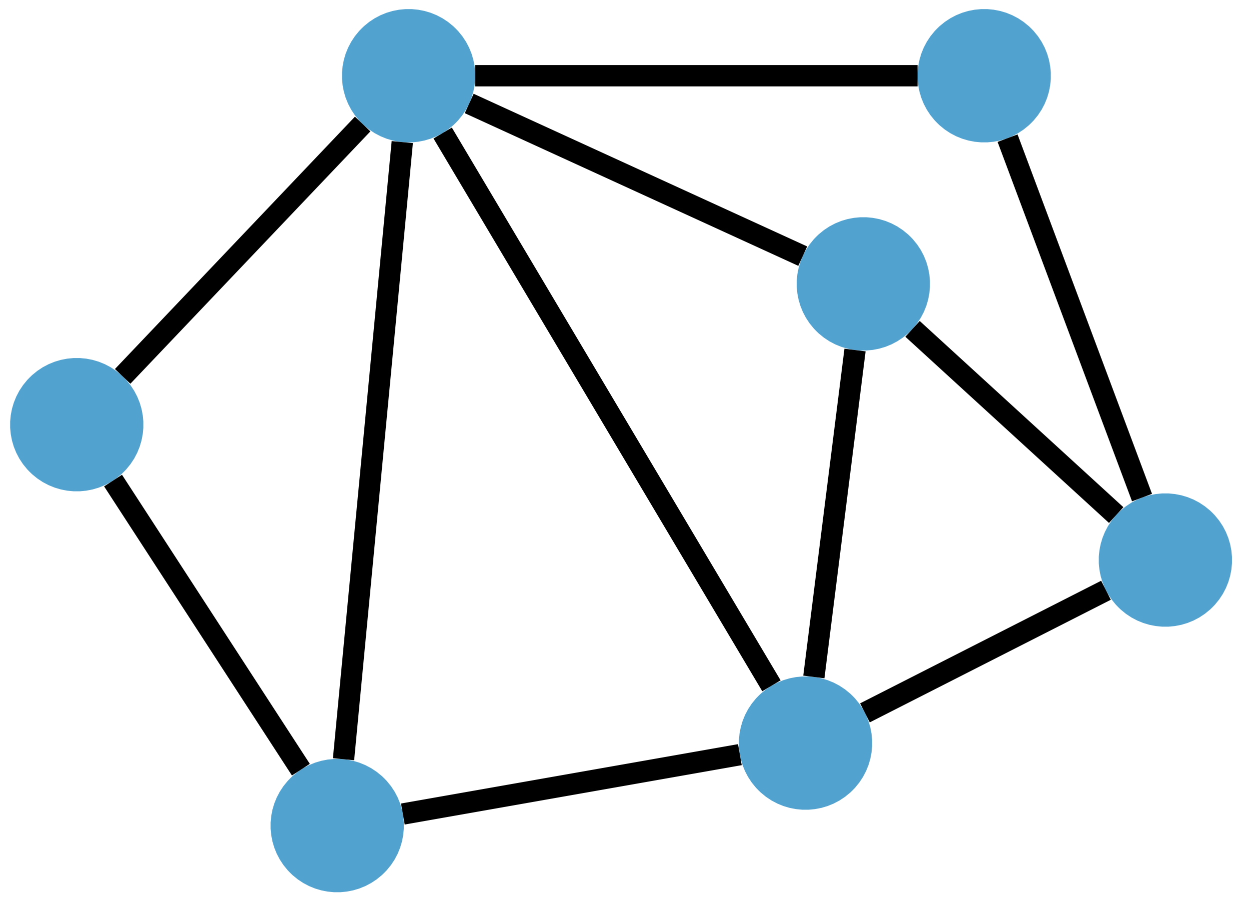
The size-k FVS is a vertex cover of G ⬅ The size-k FVS of G'

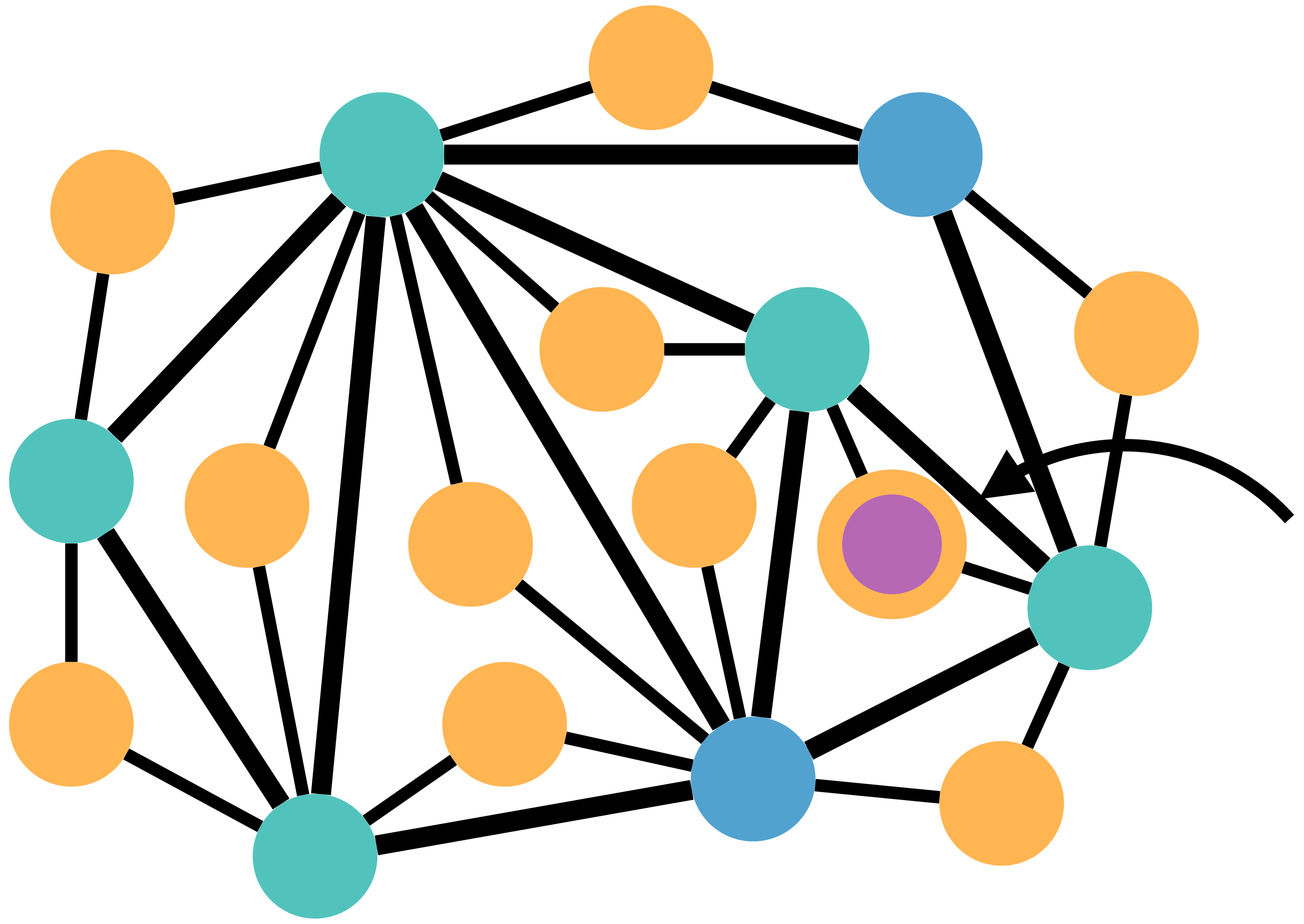
Construct a set F':

Keep all u_i in the FVS

If some $u_{i,j}$ is in the FVS, replace it by u_i or u_j

If both u_i are in u_j the FVS, remove $u_{i,j}$





Feedback Vertex Set (FVS)

F' is a vertex cover of G

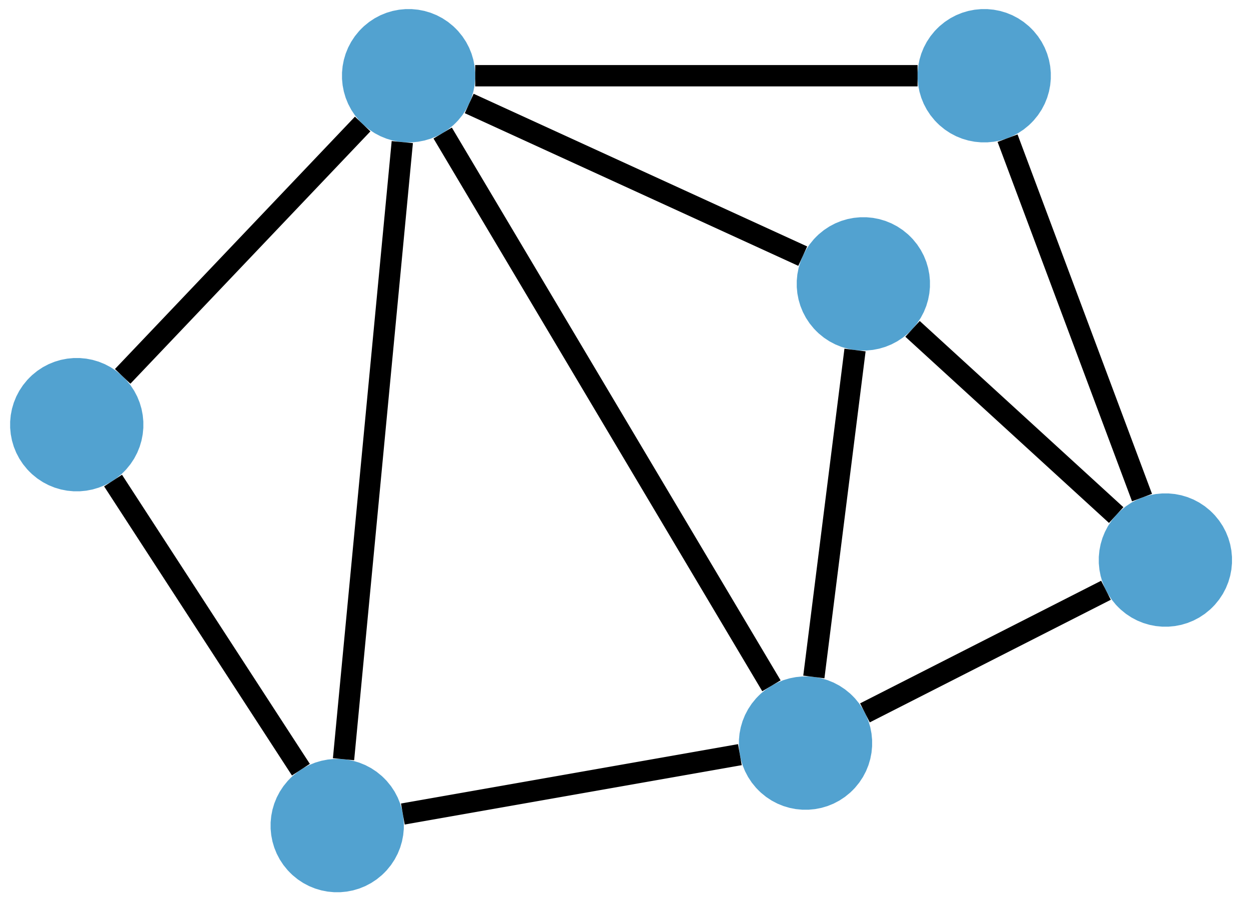
The size-k FVS of G'

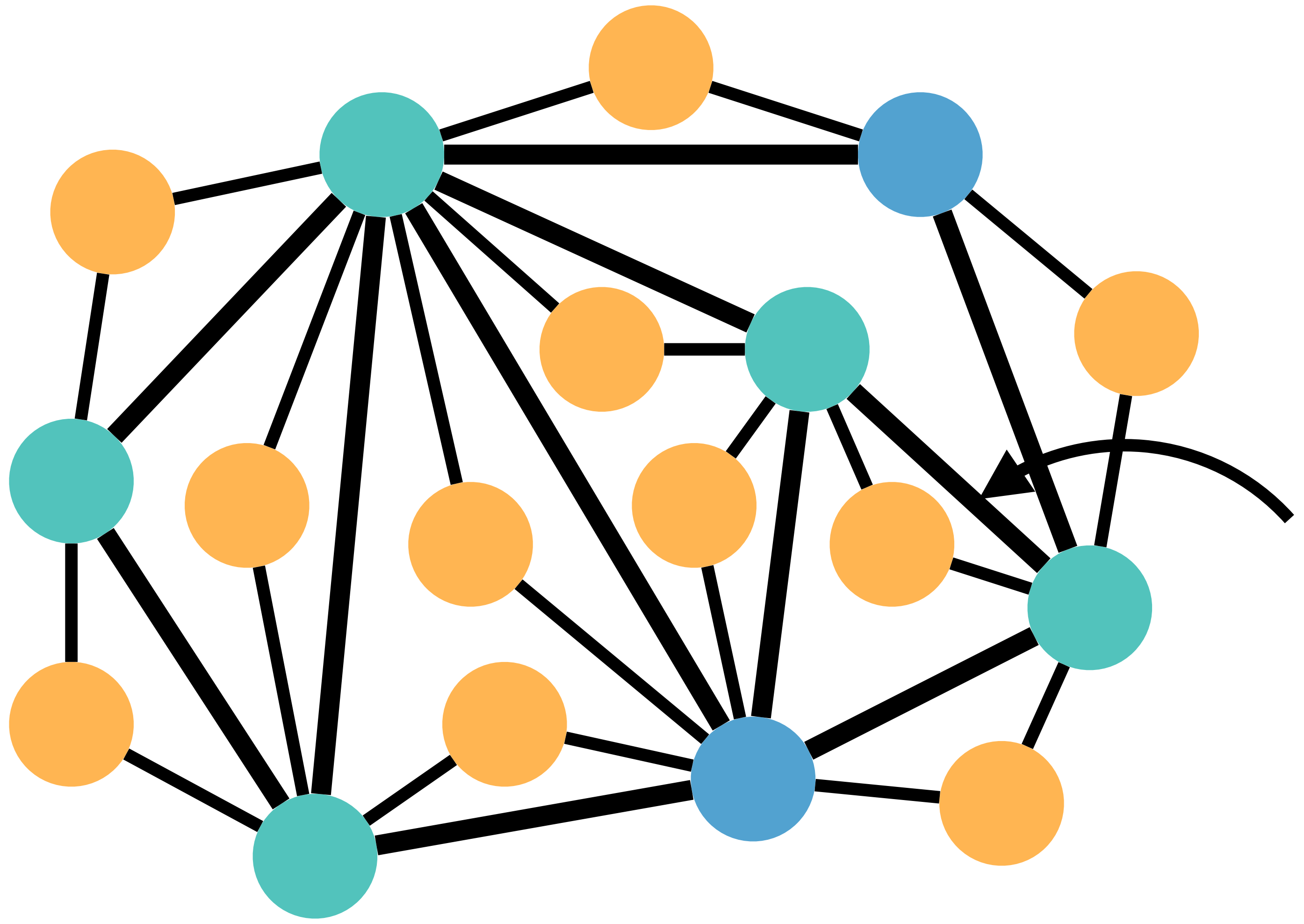
Construct a set F':

Keep all u_i in the FVS

If some $u_{i,j}$ is in the FVS, replace it by u_i or u_j

If both u_i are in u_j the FVS, remove $u_{i,j}$



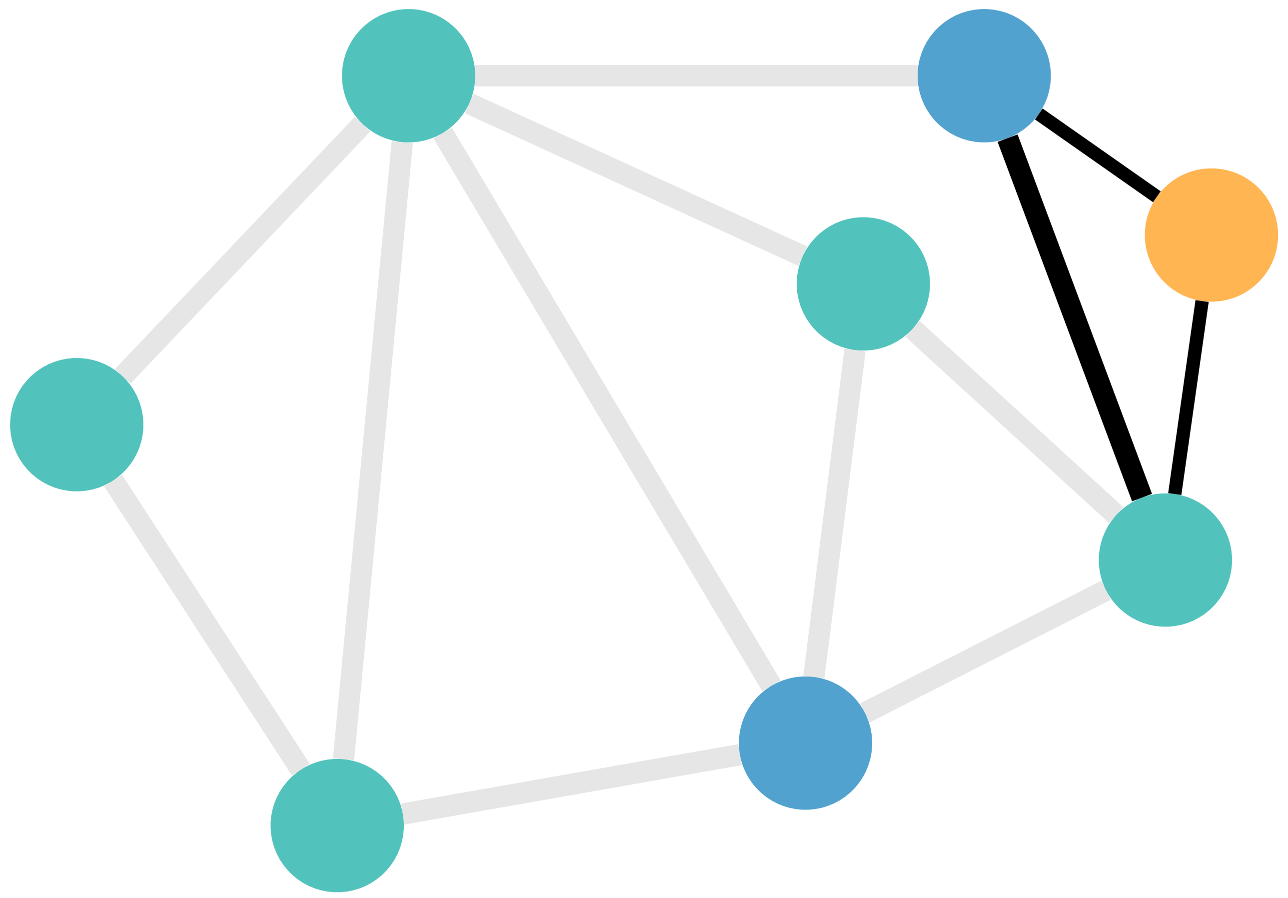


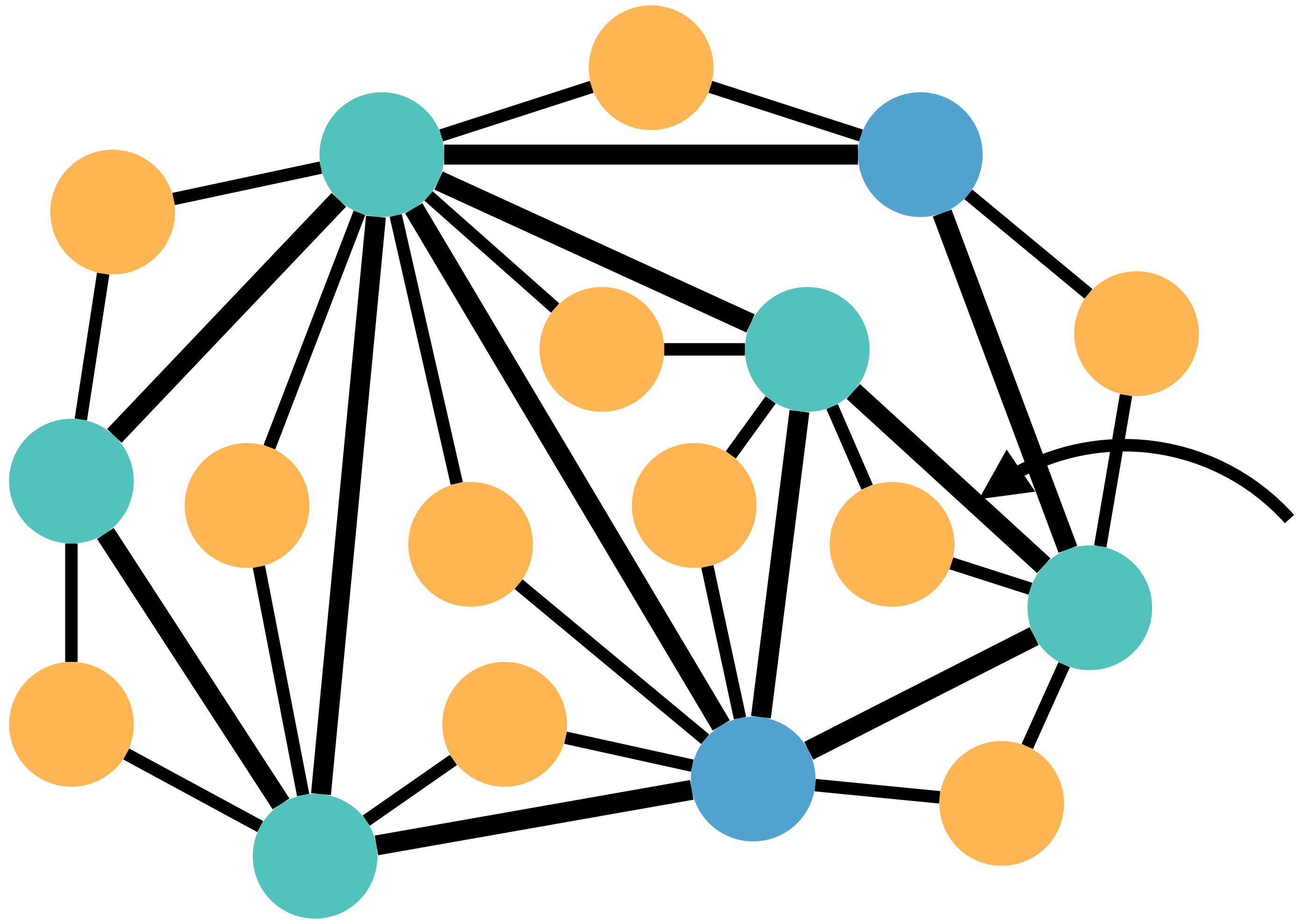
Feedback Vertex Set (FVS)

F' is a vertex cover of G ⇐ The size-k FVS of G'

After removing F' from G', there is no cycle
⇒ After removing F' from G, there is no edge
(Otherwise, there is a cycle in G' since no $u_{i,j}$ is in F')
⇒ F' is a vertex cover with size at most k in G

Construct a set F':
Keep all u_i in the FVS
If some $u_{i,j}$ is in the FVS, replace it by u_i or u_j
If both u_i are in u_j the FVS, remove $u_{i,j}$
⇒ F' is a FVS with size at most k'





Feedback Vertex Set (FVS)

- **FVS** $= \{\langle G, k\rangle \mid$ There is a set of at most k vertices in G such that removing them leaves no cycles$\}$

- Theorem: FVS is NP-complete

<proof> To prove that FVS is in NP, we use a size-k feedback vertex set U as the certificate. The verifier should check U it is a proper subset of the vertices in G, and if G is cycle-free after removing all edges incident to the vertices in U. The later can be done by running a breadth-first-search on the resulting graph. The checking time is in polynomial of the size of G.

Feedback Vertex Set (FVS)

To prove the NP-hardness, we show that VERTEX-COVER $\leq_p$ FVS. For any instance of VERTEX-COVER, $G = (V, E)$ and k, we construct an instance of FVS, $G' = (V', E')$ and k' as follows. For each vertex $v_i \in V$, there is a corresponding vertex $u_i \in V'$. More over, for each edge $(v_i, v_j) \in E$, there is a corresponding vertex $u_{i,j} \in V'$.

For each edge $(v_i, v_j) \in E$, we construct three edges in E': (u_i, u_j), $(u_i, u_{i,j})$, and $(u_j, u_{i,j})$.

We set $k' = k$.

The construction takes constant time to each element in V or E and can be done in polynomial-time.

Feedback Vertex Set (FVS)

Now we prove that the reduction works. Suppose that there is a size-k vertex cover C of G. Consider removing all vertices in C from V', there is no edge between any u_i and u_j. Furthermore, because every vertex $u_{i,j}$ only adjacent to u_i and u_j, the degree of $u_{i,j}$ is at most 1 after removing. Thus, there is no cycle left, and C is a size-k' feedback vertex set of G'. That is, G' is a yes-instance of FVS.

Feedback Vertex Set (FVS)

For the other direction, suppose that there is a size-k' feedback vertex set F of G'. We make a feedback vertex set F' of G' with size at most k' as follows. If there is a vertex $u_{i,j}$ in F, we replace it by u_i or u_j, which was not in F. If both u_i and u_j are already in F, we simply remove $u_{i,j}$. Because $u_{i,j}$ has degree 2, it can only breaks the cycle $(u_i, u_j, u_{i,j})$, and this cycle can be broken by u_i or u_j. Therefore, F' is a feasible feedback vertex set with size at most k'.

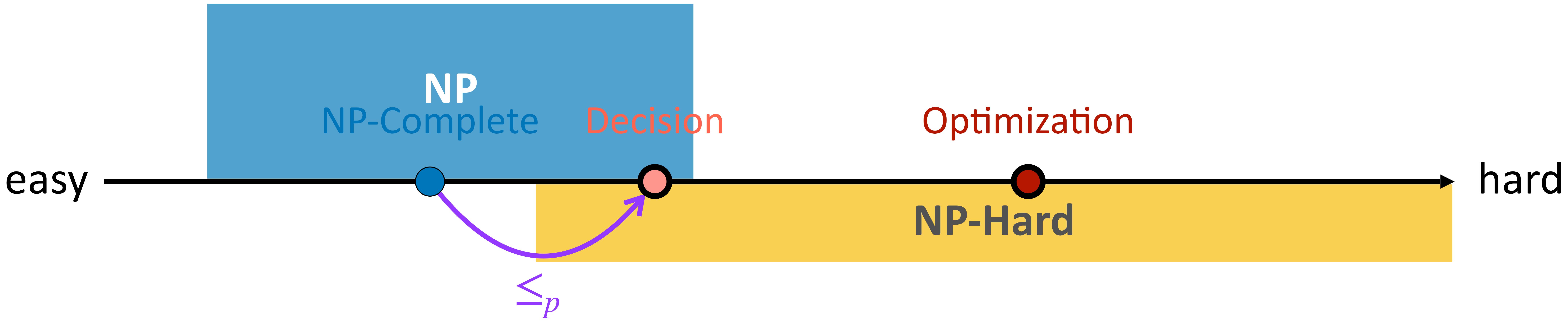
Now, we argue that the vertices in F' form a vertex cover in G. Since there is no vertex $u_{i,j}$ in F', removing all vertices in F' leaves no edge between any pair of u_i and u_j. Otherwise, there is a cycle $(u_i, u_j, u_{i,j})$, and it contradicts to the fact that F' is a feedback vertex set. Thus, F' is a vertex cover in G. That is, G is a yes-instance of the VERTEX-COVER problem. ∎

Decision Version Problem and Hardness

The decision version problem is not harder than the optimization problem

Decision Version Problem and Hardness

Decision Version Problem and Hardness



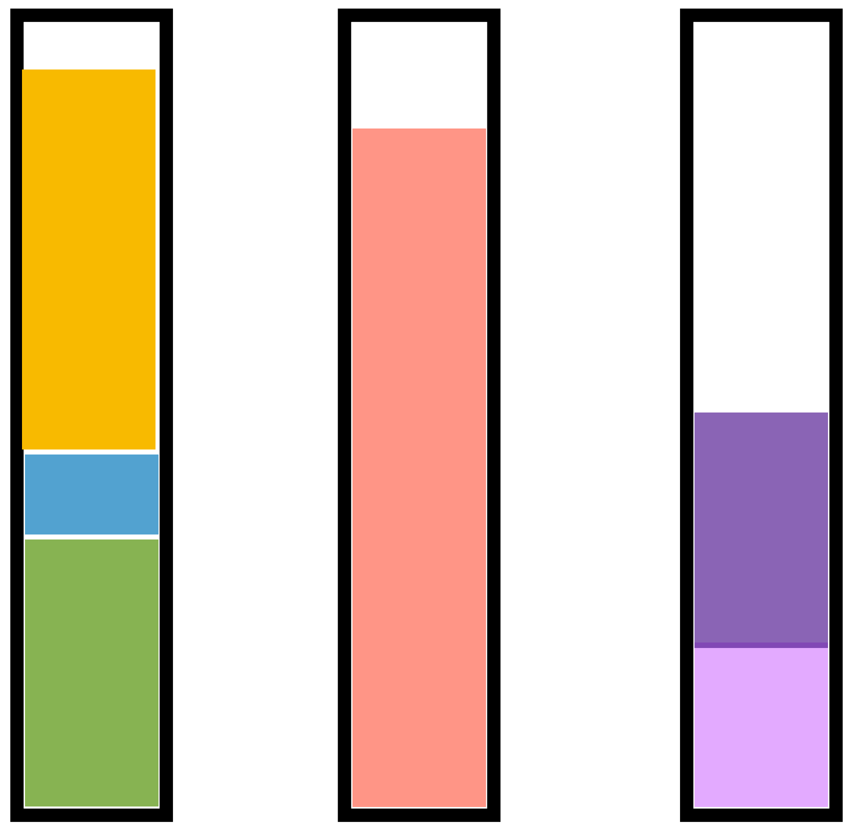
If the decision version problem is NP-complete,
it does not imply that the optimization problem is also NP-complete

Outline

- More NP-Hardness proofs
  - 3SAT $\leq_p$ CLIQUE
  - VERTEX-COVER $\leq_p$ FEEDBACK-VERTEX-SET
  - PARTITION $\leq_p$ BIN-PACKING
- Pseudo-polynomial time algorithms
- **NP** and **Co-NP**

BIN-PACKING

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



BIN-PACKING

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

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

BIN-PACKING

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

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

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

BIN-PACKING

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

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

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

- Theorem: BIN-PACKING is NP-complete

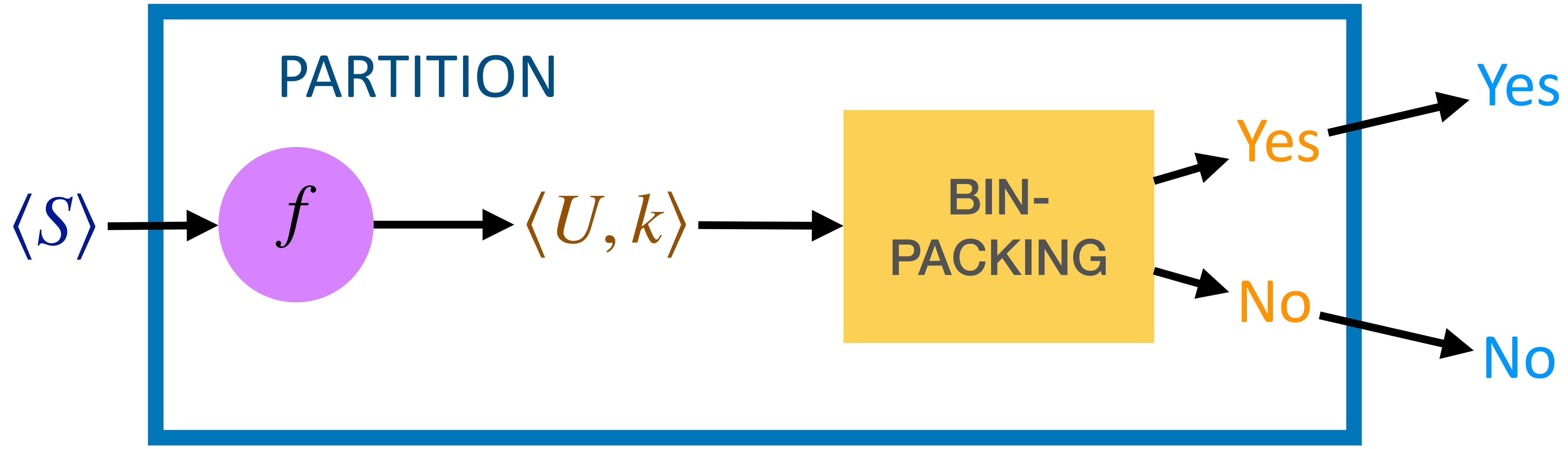
BIN-PACKING

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

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

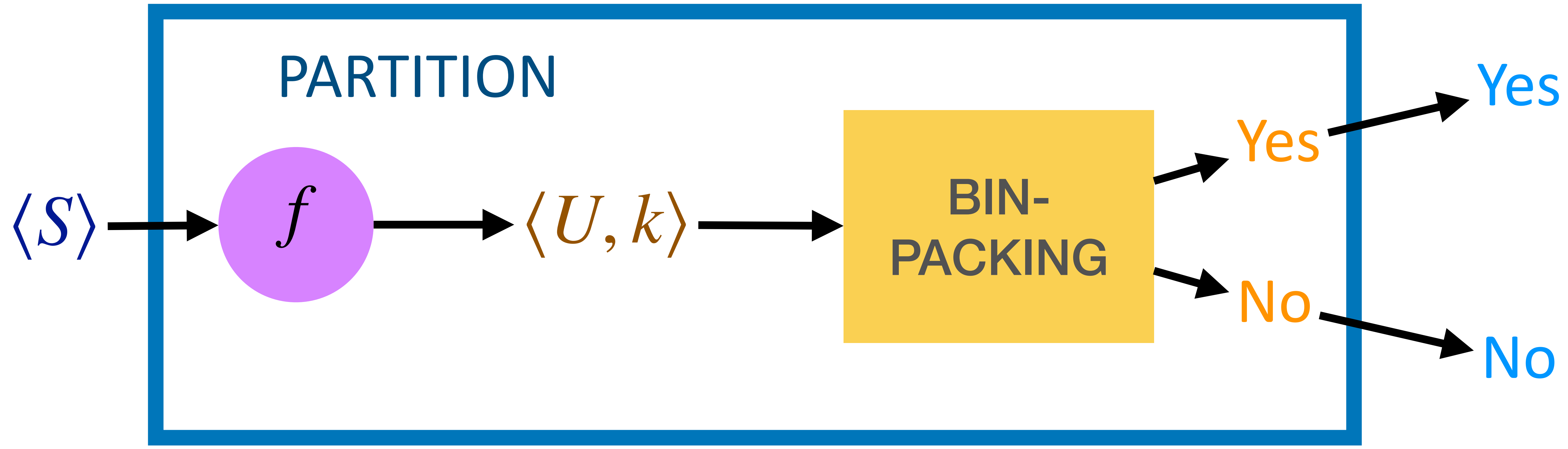
Reduce PARTITION to BIN-PACKING

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



Reduce PARTITION to BIN-PACKING

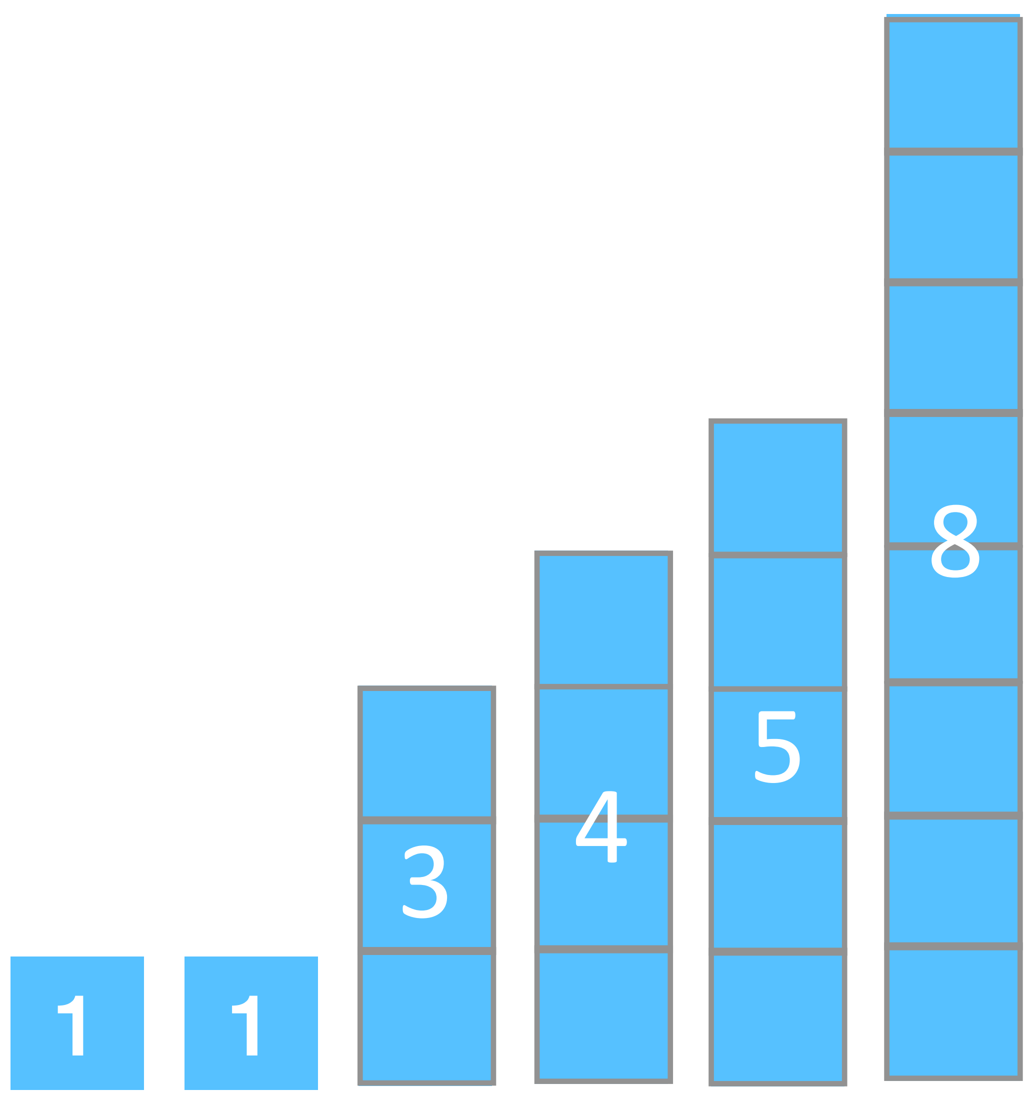
- PARTITION $= \{\langle S \rangle \mid S = \{x_1, \cdots, x_n\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\sum_{y_i \in T} y_i = \sum_{z_i \in S \setminus T} z_i\}$
- BIN-PACKING $= \{\langle U, k \rangle \mid U$ can be partitioned into at most k disjoint subsets such that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



Reduce PARTITION to BIN-PACKING

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

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



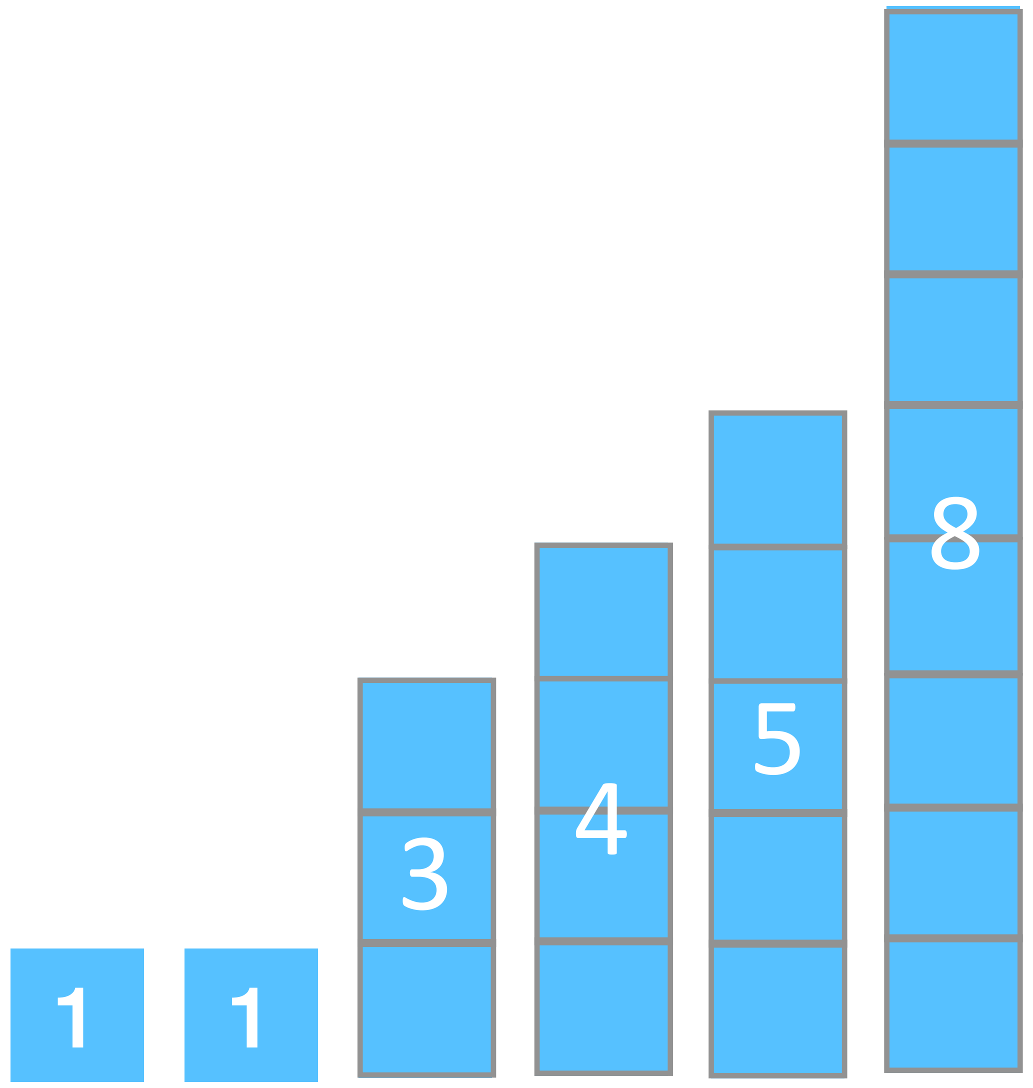
Reduce PARTITION to BIN-PACKING

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

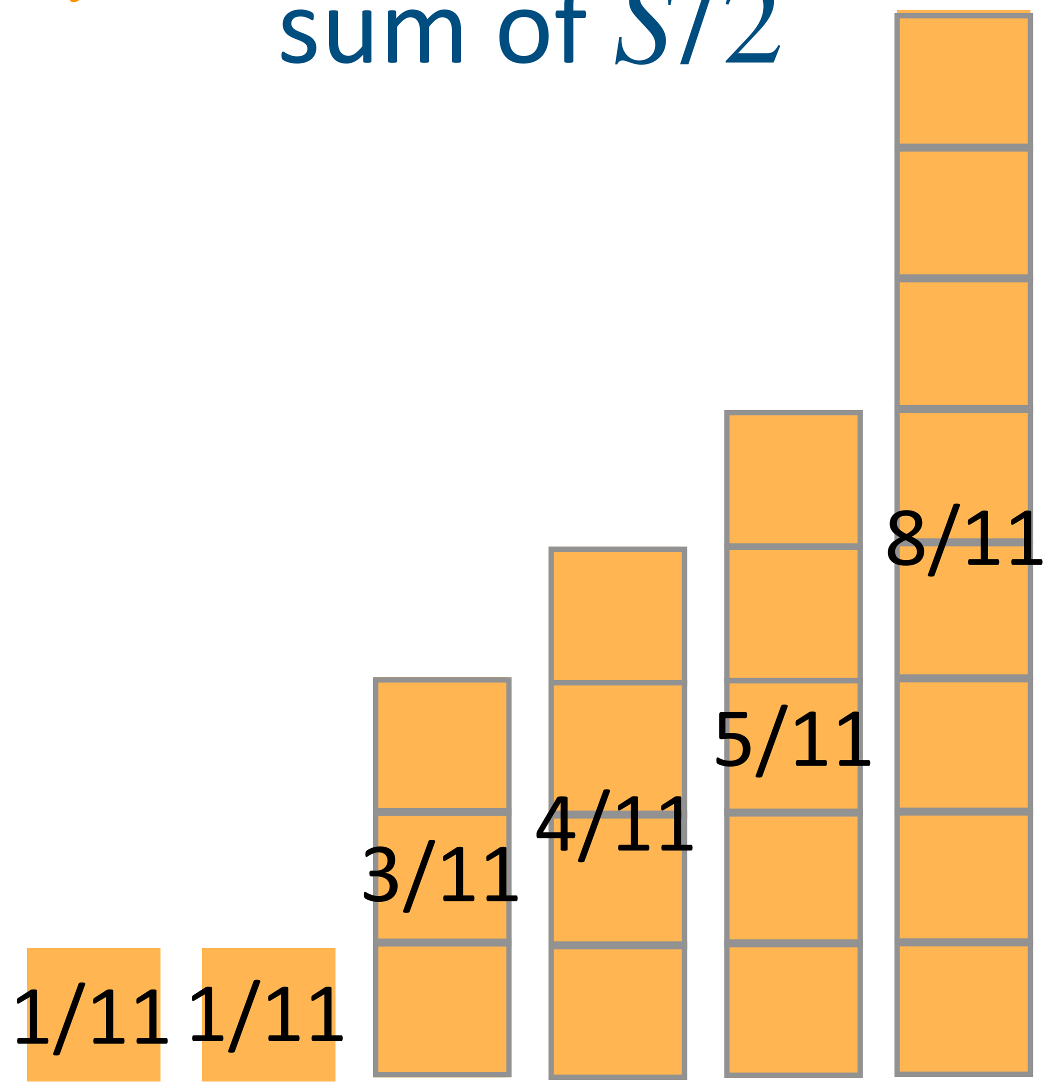
- BIN-PACKING $= \{\langle U, k\rangle \mid U$ can be partitioned into at most k disjoint subsets such that the total size of the items in each subset is no more than $1\}$

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

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



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

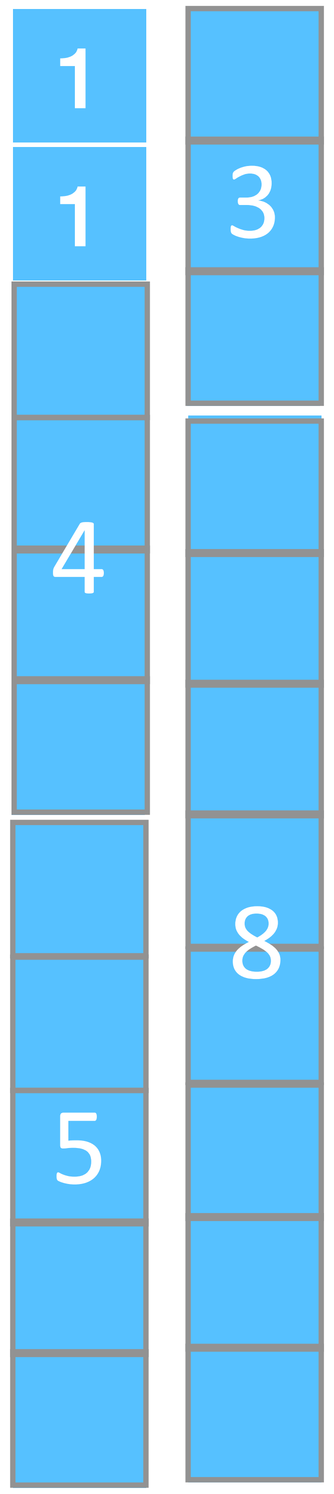


Reduce PARTITION to BIN-PACKING

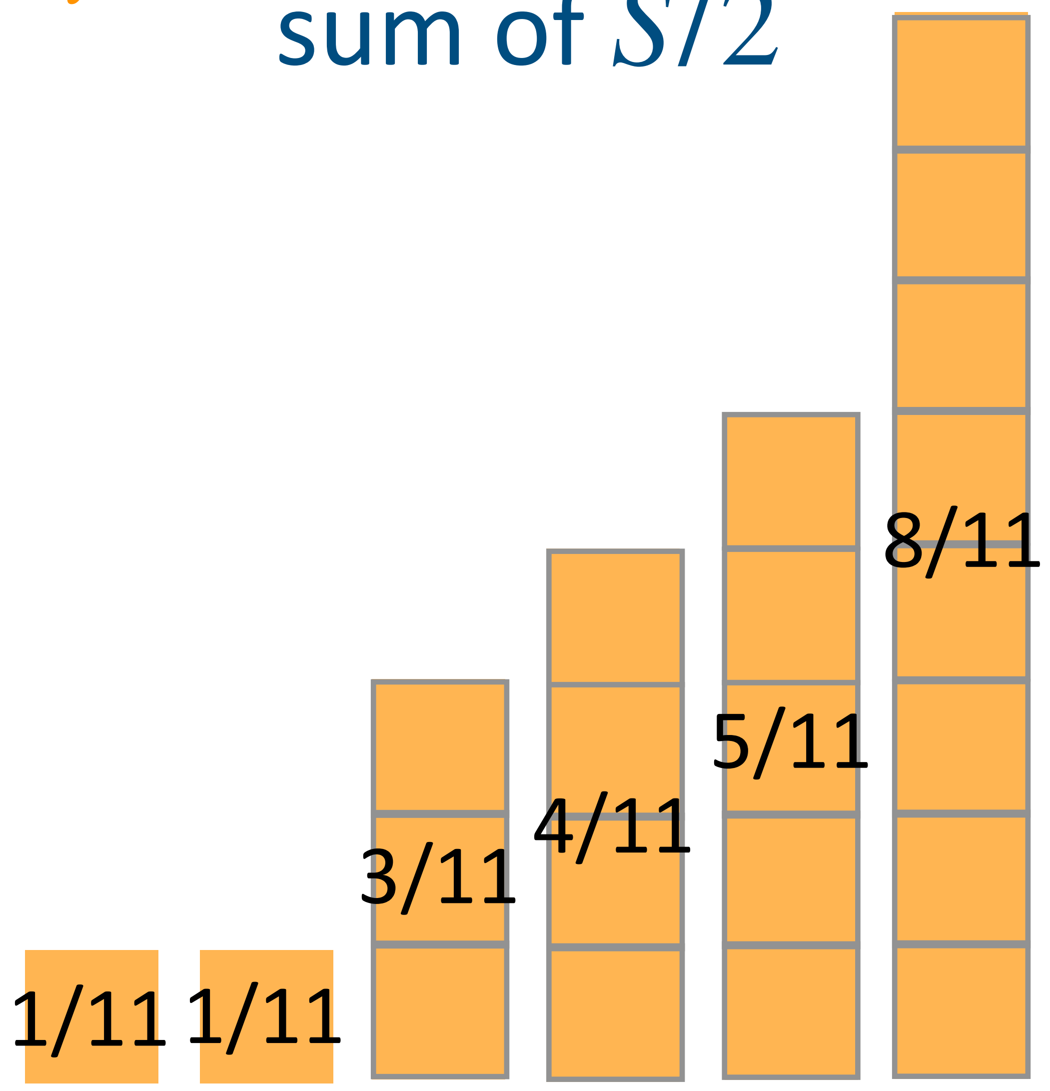
- PARTITION $= \{\langle S\rangle \mid S = \{1,1,3,4,5,8\}$ and for some subset $T = \{y_1, \cdots, y_m\} \subset S$, we have $\sum_{y_i \in T} y_i = \sum_{z_i \in S \setminus T} z_i\}$

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

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

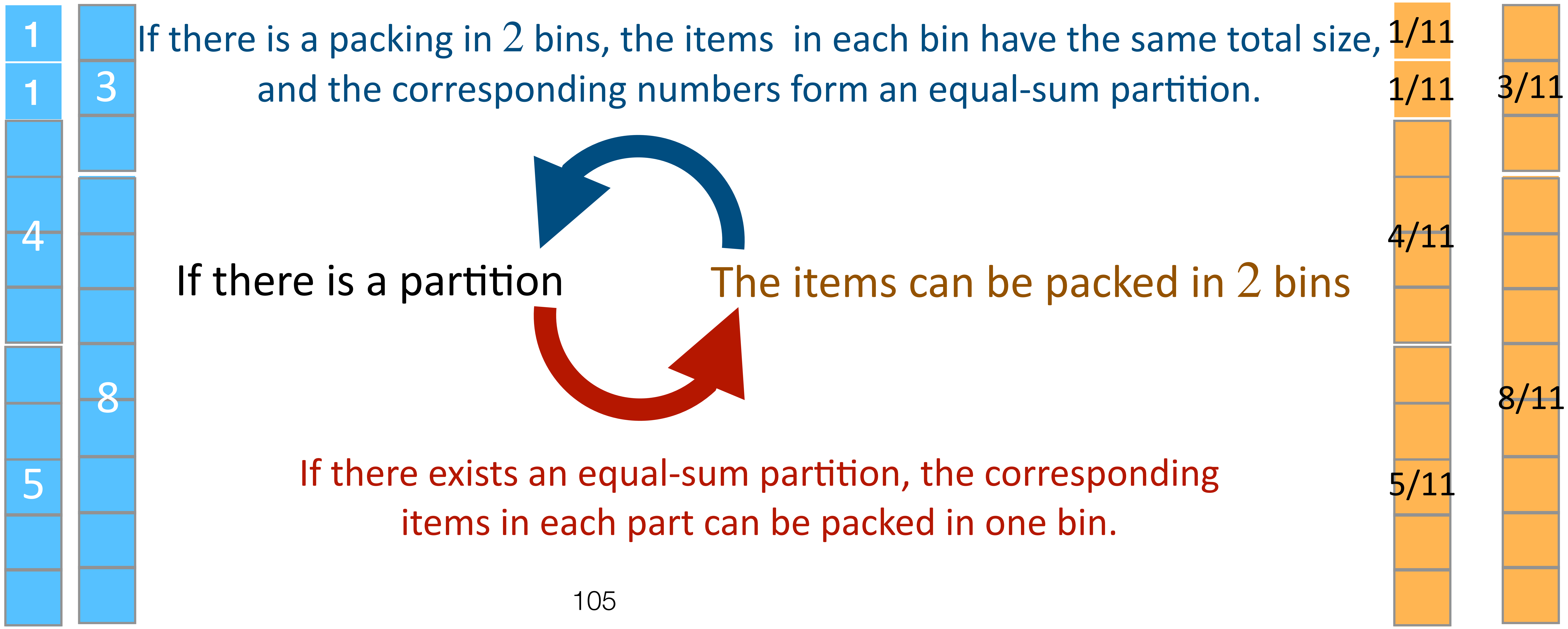


Items in U with size $s(u_i) = \dfrac{y_i}{\text{sum of } S/2}$



Reduce PARTITION to BIN-PACKING

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



BIN-PACKING

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

- Theorem: BIN-PACKING is NP-complete

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

BIN-PACKING

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

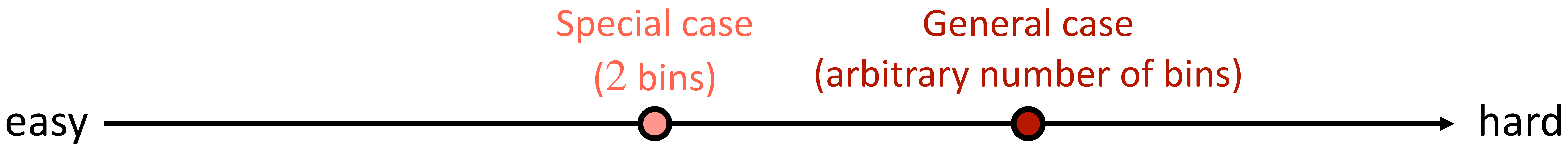
BIN-PACKING

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

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

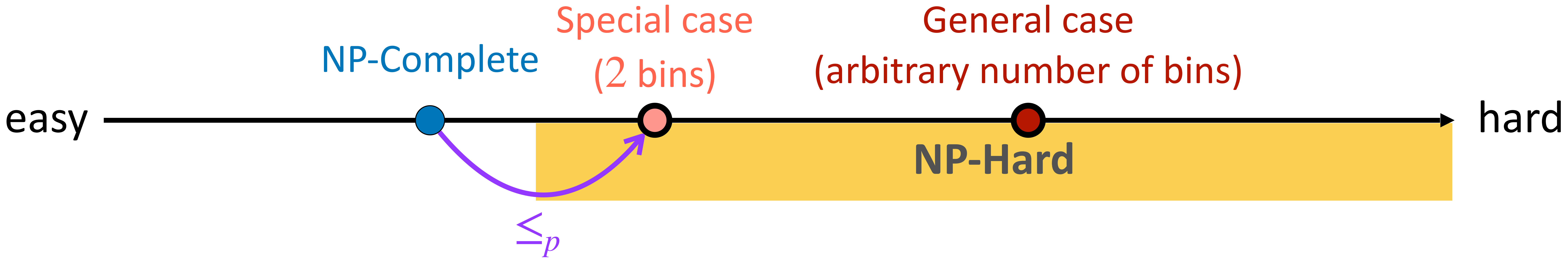
$\blacksquare$

Special Case and Hardness

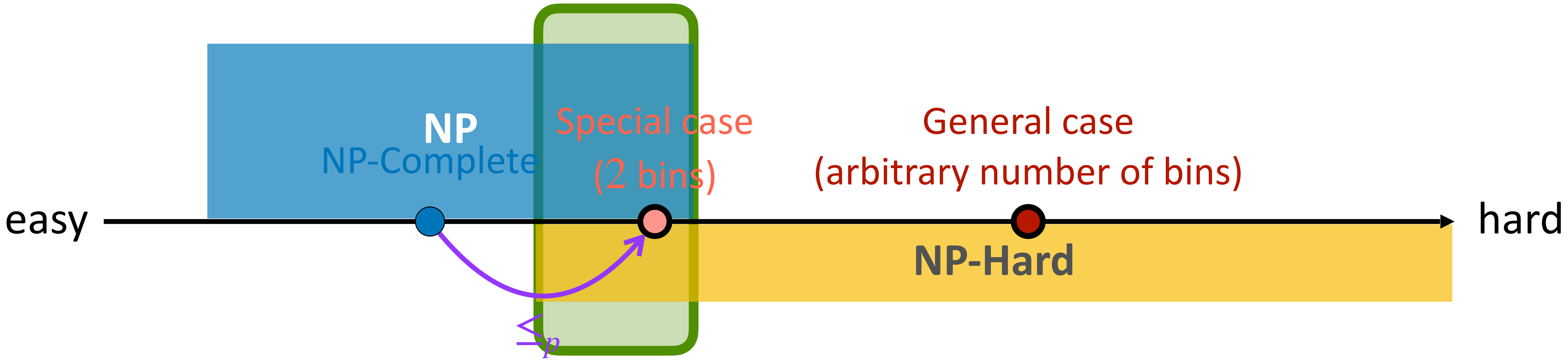


The special case is not harder than the general case

Special Case and Hardness

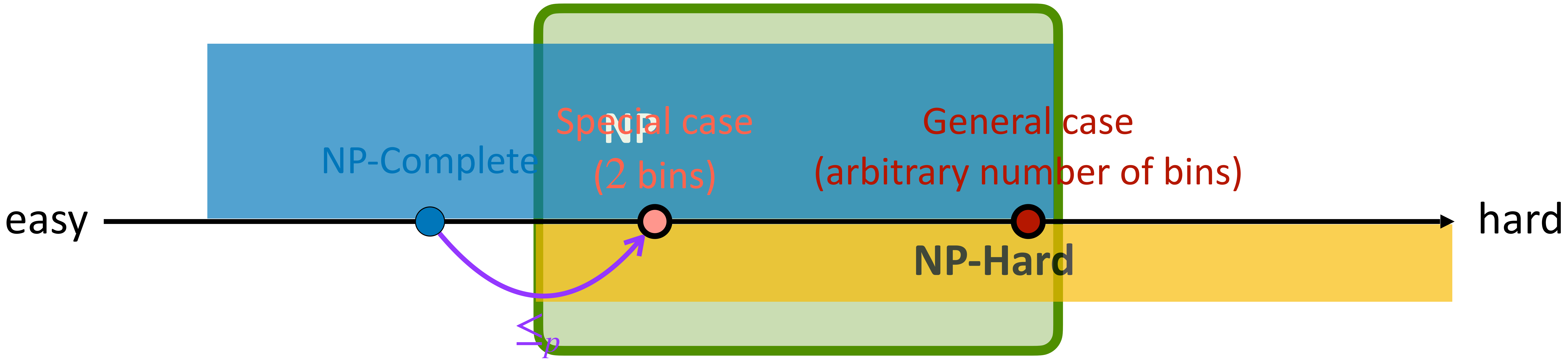


Special Case and Hardness



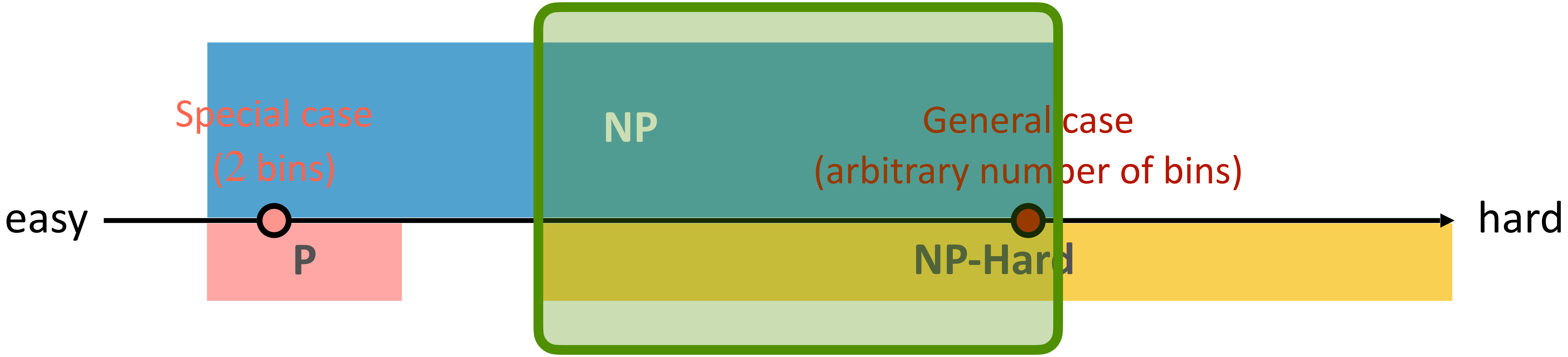
If the special case is NP-complete,
it does not imply that the general case is also NP-complete

Special Case and Hardness



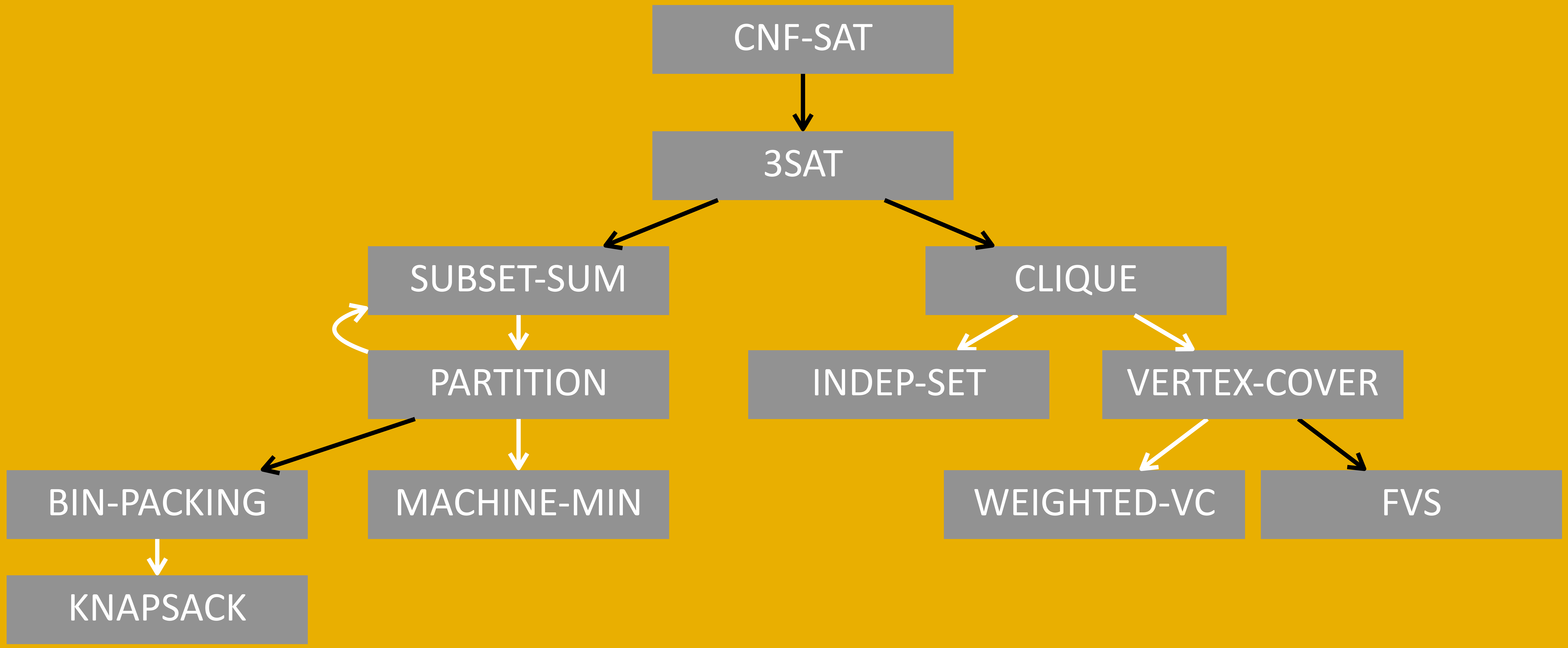
If the general case is NP-complete,
it implies that the special case is also NP-complete

Special Case and Hardness



It’s also possible!

NP-complete Problems Map



Outline

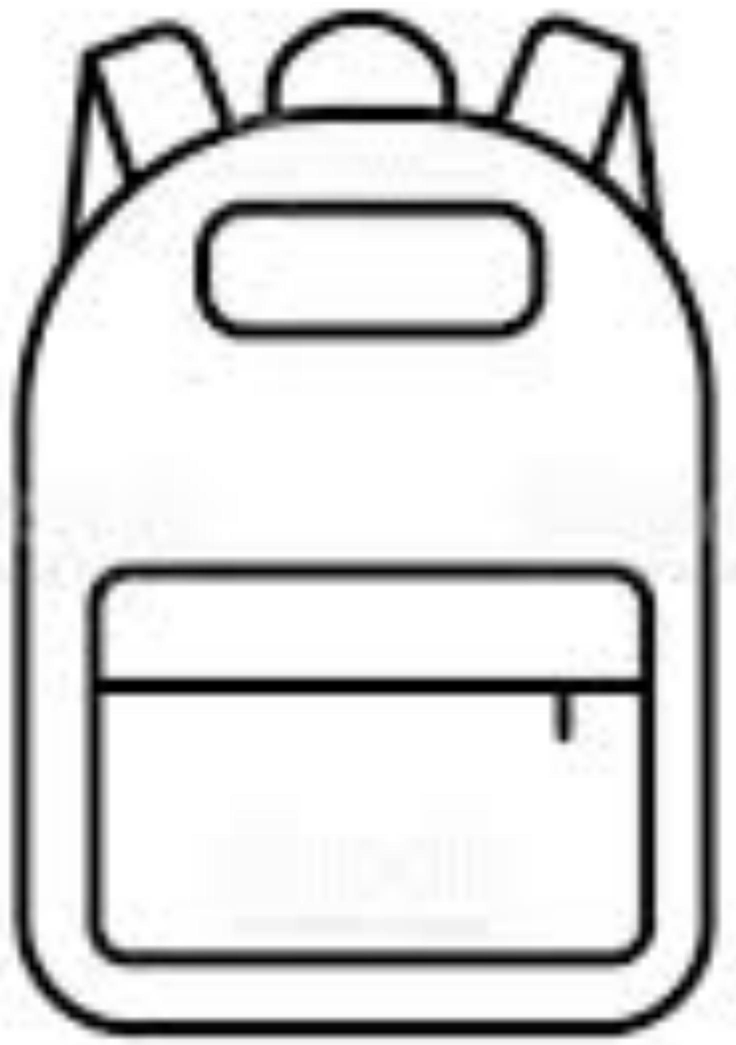
- More NP-Hardness proofs
  - 3SAT $\leq_p$ CLIQUE
  - VERTEX-COVER $\leq_p$ FEEDBACK-VERTEX-SET
  - PARTITION $\leq_p$ BIN-PACKING
- Pseudo-polynomial time algorithms
- **NP** and **Co-NP**

Strong and Weak NP-complete

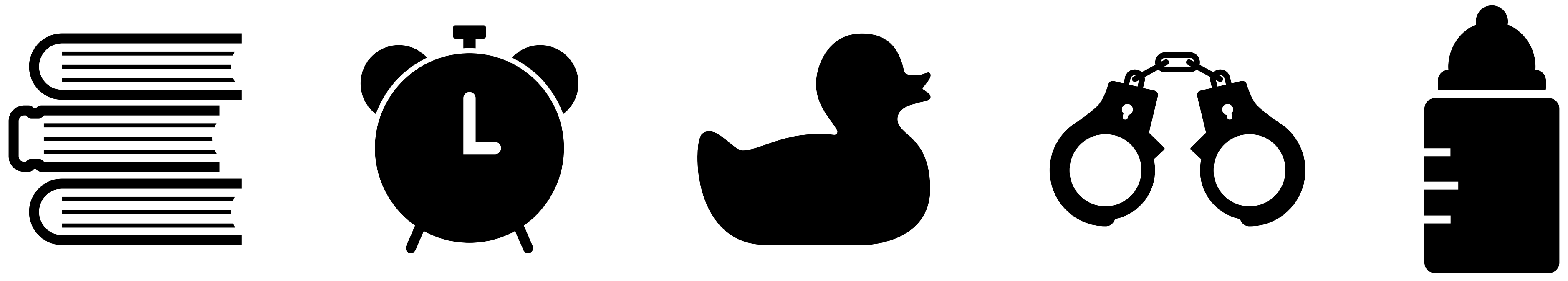
- KANPSACK problem: Give a set S of items, each with an integer value v_i and integer weight w_i. Also give integers B and V. Is there a subset of S of weight no more than B with total value at least V?

Strong and Weak NP-complete

- KANPSACK problem: Give a set S of items, each with an integer value v_i and integer weight w_i. Also give integers B and V. Is there a subset of S of weight no more than B with total value at least V?



Weight capacity: 100



value	100	80	68	42	25
weight	50	45	10	15	20

Strong and Weak NP-complete

- KANPSACK problem: Give a set S of items, each with an integer value v_i and integer weight w_i. Also give integers B and V. Is there a subset of S of weight no more than B with total value at least V?

- KNAPSACK is NP-complete

Strong and Weak NP-complete

- KANPSACK problem: Give a set S of items, each with an integer value v_i and integer weight w_i. Also give integers B and V. Is there a subset of S of weight no more than B with total value at least V?

- KNAPSACK is NP-complete

- Using dynamic programming, it can be solved in $O(nB)$ time.
  - $W(j, w) := \max\{\Sigma_{i\in S} v_i \mid S \subseteq \{1,2,\cdots,j\}, \Sigma_{i\in S} w_i \leq w\}$
  - $W(j+1, w) = \max\{W(j, w), W(j, w - w_{j+1}) + v_{j+1}\}$

Strong and Weak NP-complete

- KANPSACK problem: Give a set S of items, each with an integer value v_i and integer weight w_i. Also give integers B and V. Is there a subset of S of weight no more than B with total value at least V?

- KNAPSACK is NP-complete

- Using dynamic programming, it can be solved in $O(nB)$ time.
  - $W(j, w) := \max\{\Sigma_{i \in S} v_i \,|\, S \subseteq \{1,2,\cdots,j\}, \Sigma_{i \in S} w_i \leq w\}$
  - $W(j+1, w) = \max\{W(j, w), W(j, w - w_{j+1}) + v_{j+1}\}$

- Have we just shown that P = NP?

Strong and Weak NP-complete

- **Weak NP-complete** (**NP-complete in the ordinary sense**):

Strong and Weak NP-complete

- **Weak NP-complete** (**NP-complete in the ordinary sense**):
    - Problem is NP-complete if numbers are given in binary, but polynomial time solvable when numbers are given in unary encoding.

Strong and Weak NP-complete

- **Weak NP-complete** (**NP-complete in the ordinary sense**):
    - Problem is NP-complete if numbers are given in binary, but polynomial time solvable when numbers are given in unary encoding.
    - Algorithms are known which solve them in time bounded by a polynomial in the numeric value of the input instead of in the length of the input length.

Strong and Weak NP-complete

- **Weak NP-complete** (**NP-complete in the ordinary sense**):
  - Problem is NP-complete if numbers are given in binary, but polynomial time solvable when numbers are given in unary encoding.
  - Algorithms are known which solve them in time bounded by a polynomial in the numeric value of the input instead of in the length of the input length.
  - The problem is solvable in **pseudo-polynomial** time

Strong and Weak NP-complete

- **Weak NP-complete** (**NP-complete in the ordinary sense**):
  - Problem is NP-complete if numbers are given in binary, but polynomial time solvable when numbers are given in unary encoding.
  - Algorithms are known which solve them in time bounded by a polynomial in the numeric value of the input instead of in the length of the input length.
  - The problem is solvable in **pseudo-polynomial** time

- **Strong NP-complete** (**NP-complete in the strong sense**):
  - Problem is NP-complete if numbers are given in unary encoding
  - Problem is NP-complete even when the numerical parameters are bounded by a polynomial in the input size
  - Ex: BinPacking

Outline

- More NP-Hardness proofs
  - 3SAT $\leq_p$ CLIQUE
  - VERTEX-COVER $\leq_p$ FEEDBACK-VERTEX-SET
  - PARTITION $\leq_p$ BIN-PACKING
- Pseudo-polynomial time algorithms
- **NP** and **Co-NP**

The Class **NP**

- Definition: **NP** is the class of languages that are decidable in polynomial time on a nondeterministic Turing machine.

- Definition: **NP** is the class of languages that are polynomial time *verifiable*.

The Class **co-NP**

- Definition: **co-NP** is the class of languages that any no-instance are polynomial time *verifiable*.

The Class **co-NP**

- Definition: **co-NP** is the class of languages that any no-instance are polynomial time *verifiable*.

- Definition: A language L is in **co-NP** if $\overline{L} \in$ NP.
  - $\overline{L}$: complement language of L
    - NOT-HAMILTONIAN $= \{\langle G \rangle \,|\, G$ has no Hamiltonian cycle$\}$
    - UNSATISFIABLE $= \{\langle \phi \rangle \,|\,$ All truth assignments make ϕ false$\}$
- $P \subseteq$ NP $\cap$ co-NP

The Class **co-NP**

- Definition: **co-NP** is the class of languages that any no-instance are polynomial time *verifiable*.

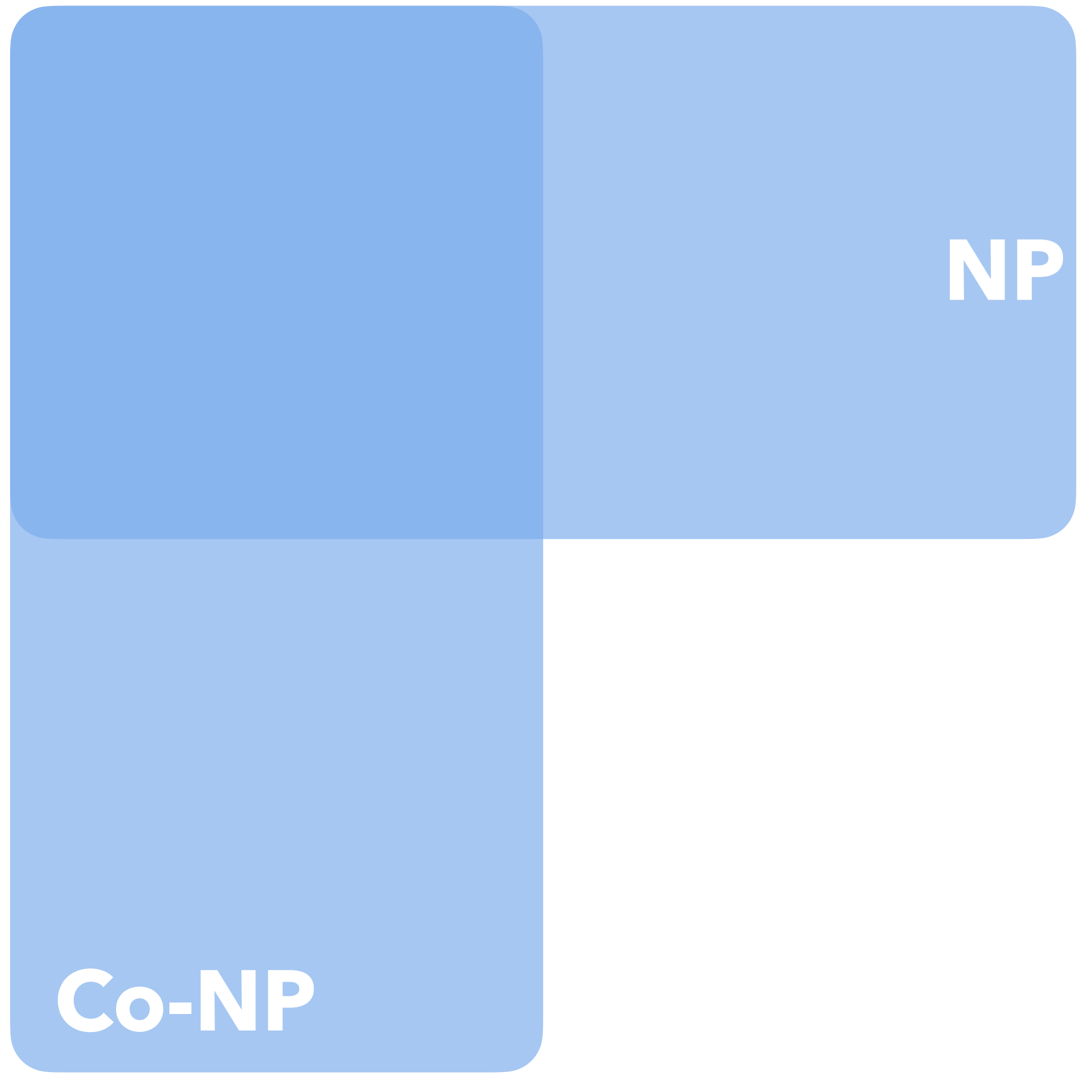
- Definition: A language L is in **co-NP** if $\overline{L} \in$ NP.

  - $\overline{L}$: complement language of L

    - NOT-HAMILTONIAN $= \{\langle G \rangle \mid G$ has no Hamiltonian cycle$\}$

    - UNSATISFIABLE $= \{\langle \phi \rangle \mid$ All truth assignments make ϕ false$\}$

- $P \subseteq$ NP $\cap$ co-NP

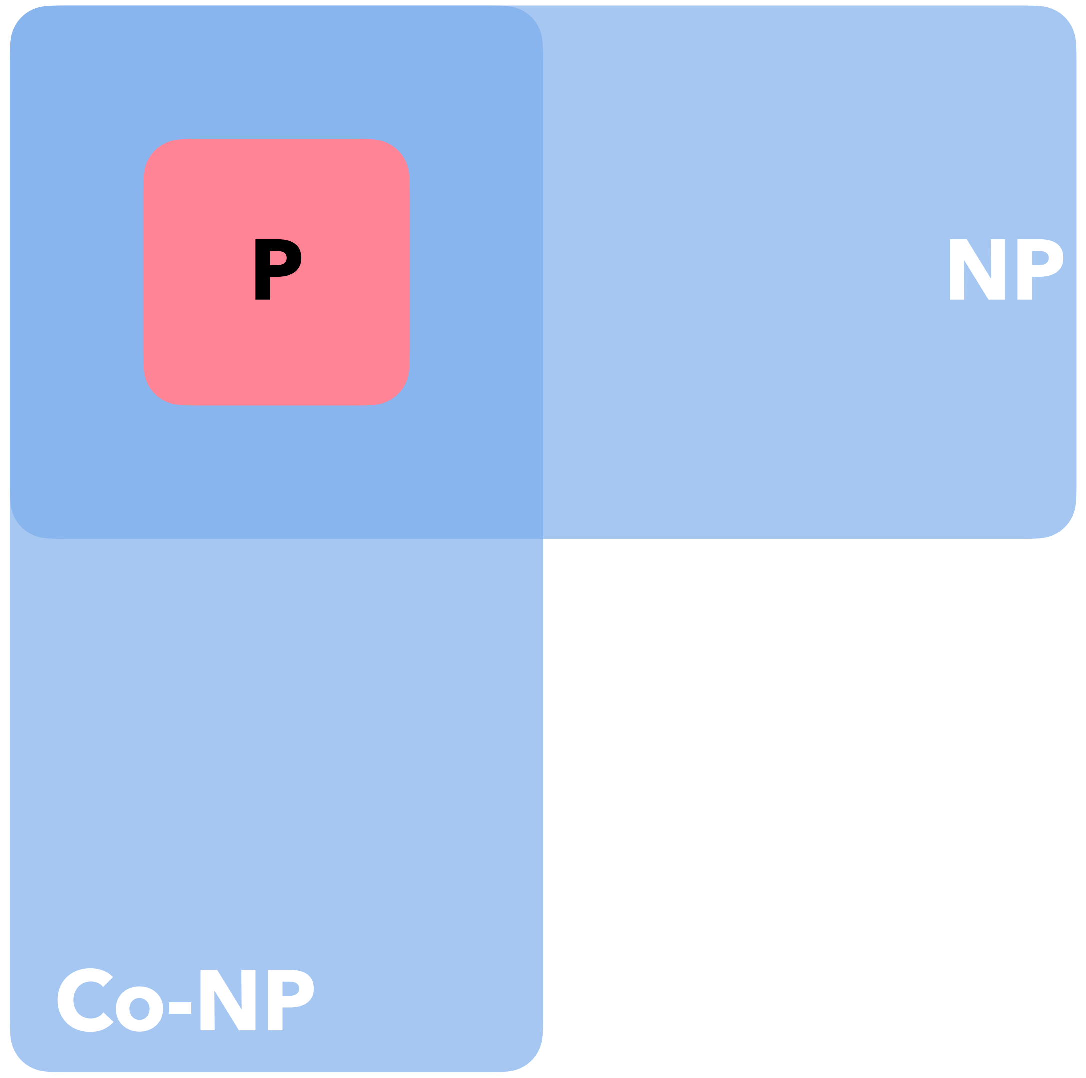


The Class **co-NP**

- Definition: **co-NP** is the class of languages that any no-instance are polynomial time *verifiable*.

- Definition: A language L is in **co-NP** if $\overline{L} \in$ NP.
  - $\overline{L}$: complement language of L
    - NOT-HAMILTONIAN $= \{\langle G \rangle \mid G$ has no Hamiltonian cycle$\}$
    - UNSATISFIABLE $= \{\langle \phi \rangle \mid$ All truth assignments make ϕ false$\}$

- $P \subseteq$ NP $\cap$ co-NP



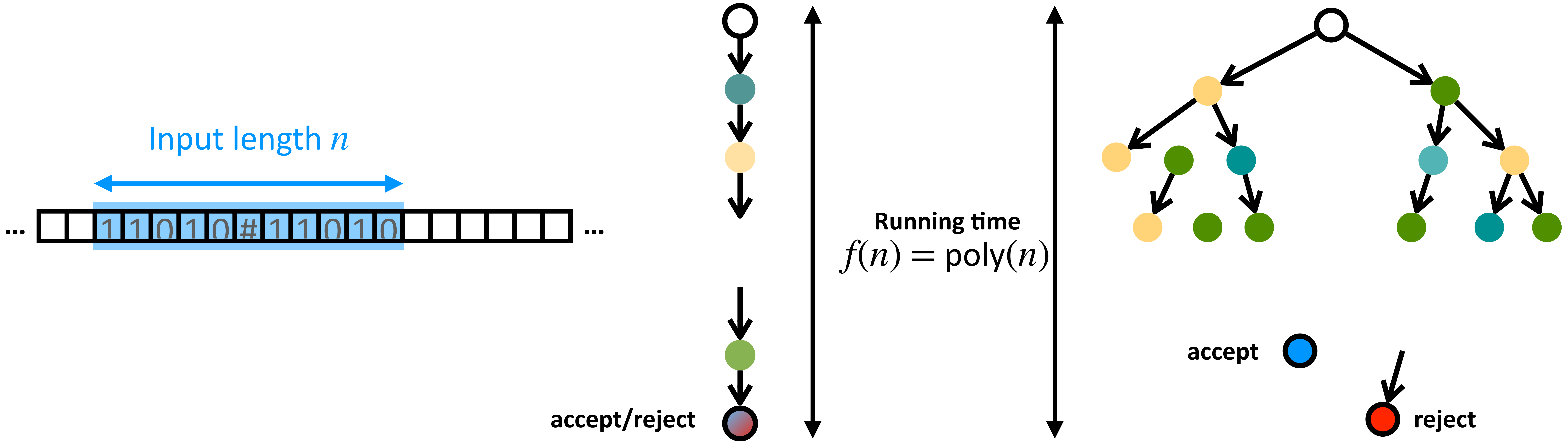
A more natural example for NP and coNP

- INTEGER_FACTORISATION $= \{\langle n, k \rangle \mid n$ has a prime factor less than $k\}$ is in NP and co-NP:
  - In NP: A certificate is two numbers c and $p < k$ where p is a prime* such that $cp = n$
  - In co-NP: A certificate is the prime factorization of n
  - Is INTETER_FACTORISATION in P? For cryptography sake we hope not!

* Prime-testing is in **P** [M Agrawal, N Kayal, N Saxena, 2004]

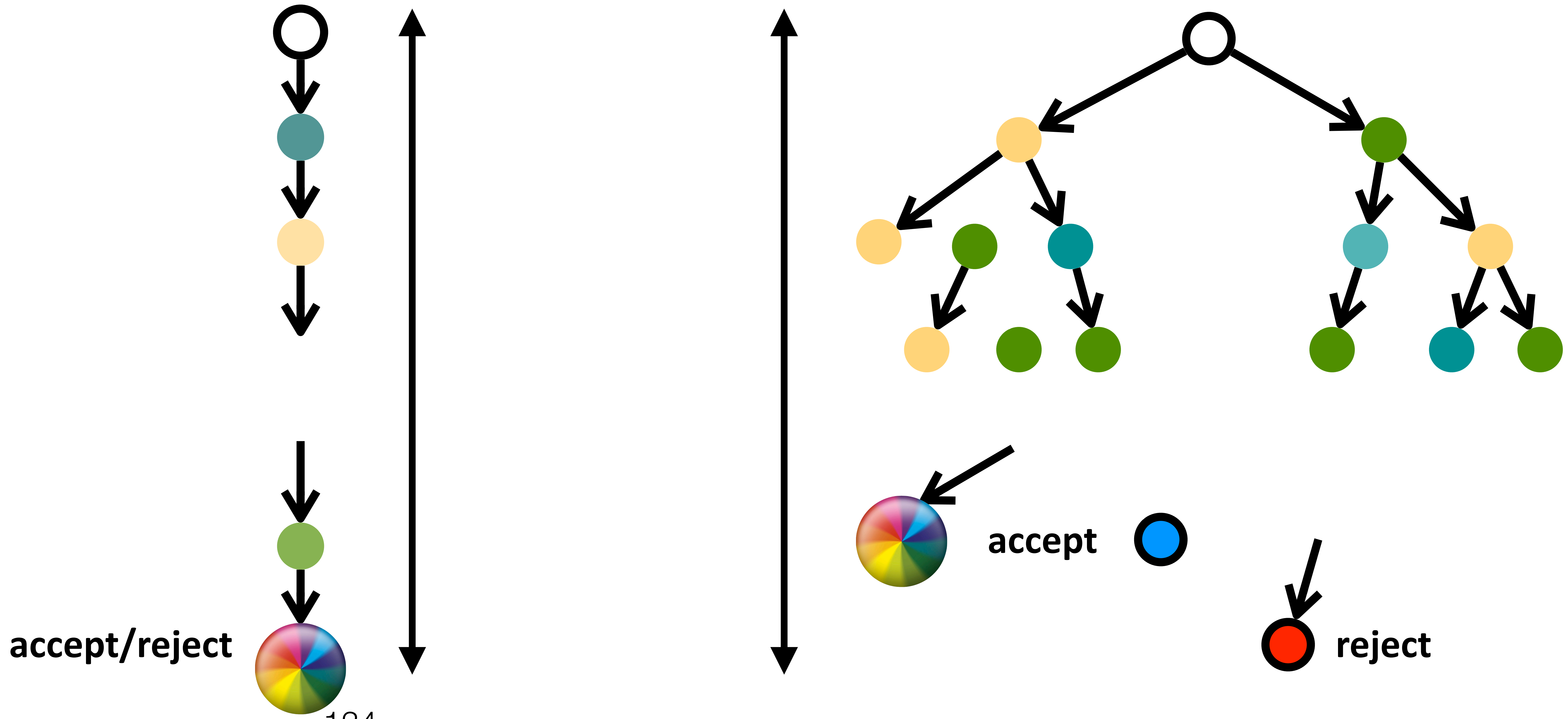
Turing machine and Decidability

- The class **P** is the class of languages that are *accepted* or *rejected* in polynomial time by a deterministic Turing machine
- The class **NP** is the class of languages that can be *verified* in polynomial time by a deterministic Turing machine.



Turing-Decidable Language

- Turing machine may not halt and enter a loop

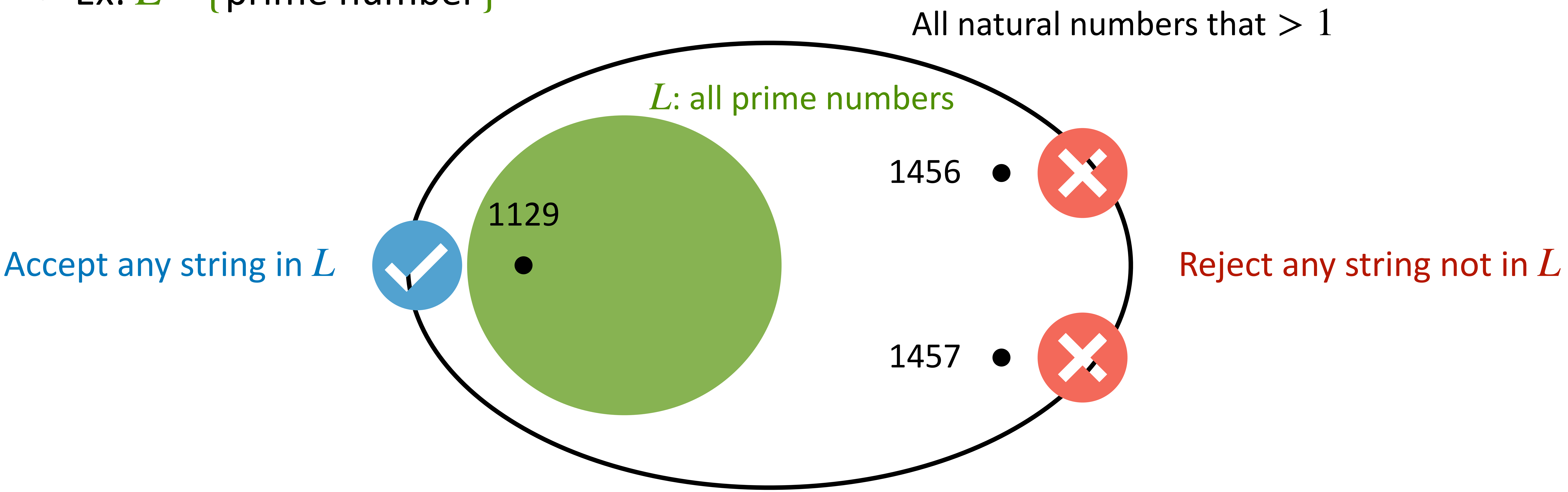


Turing-Decidable Language

- A language L is (Turing-)**decidable** if **some Turing machine *decides* it**
  - The Turing machine accepts all strings in L and rejects all strings not in L

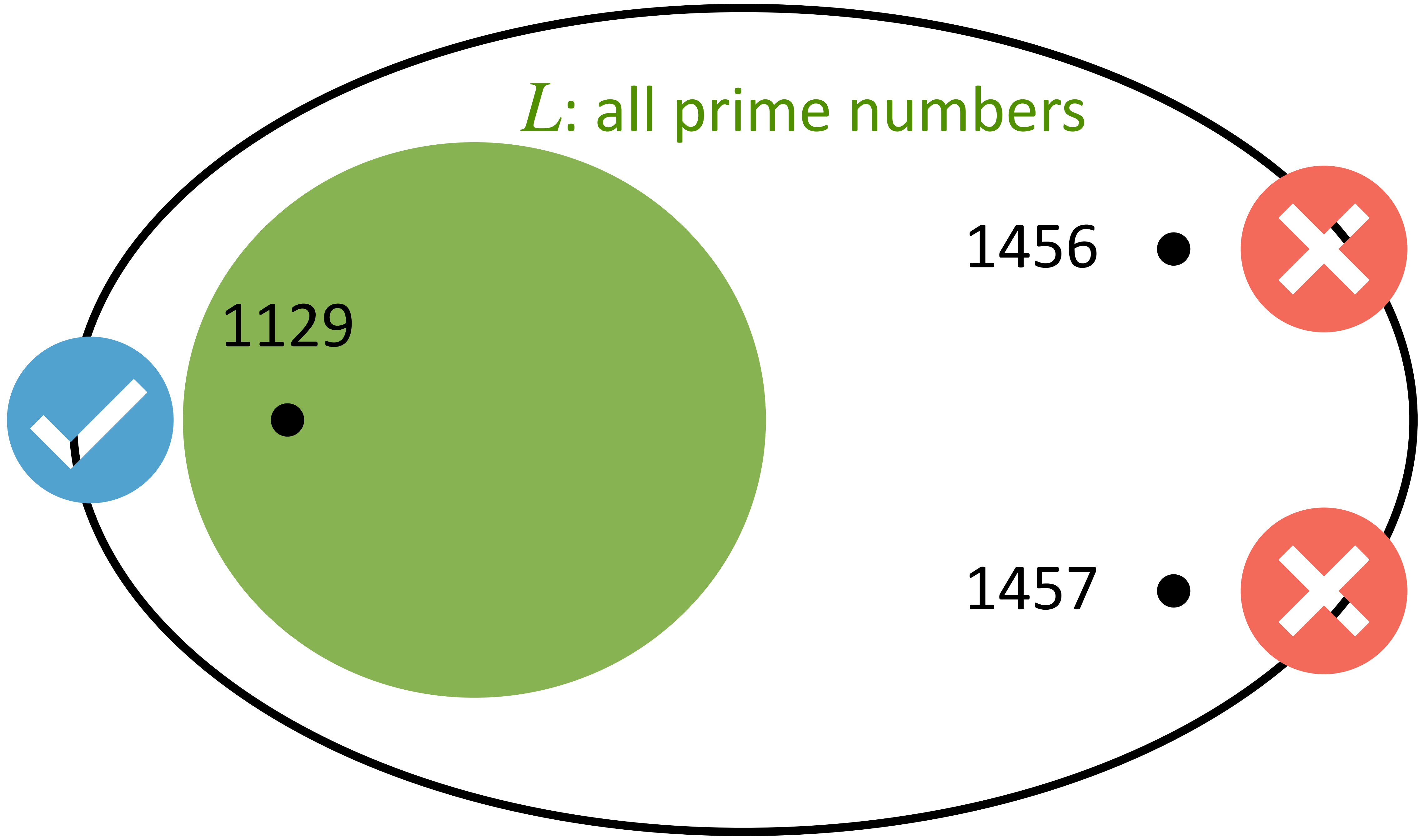
Turing-Decidable Language

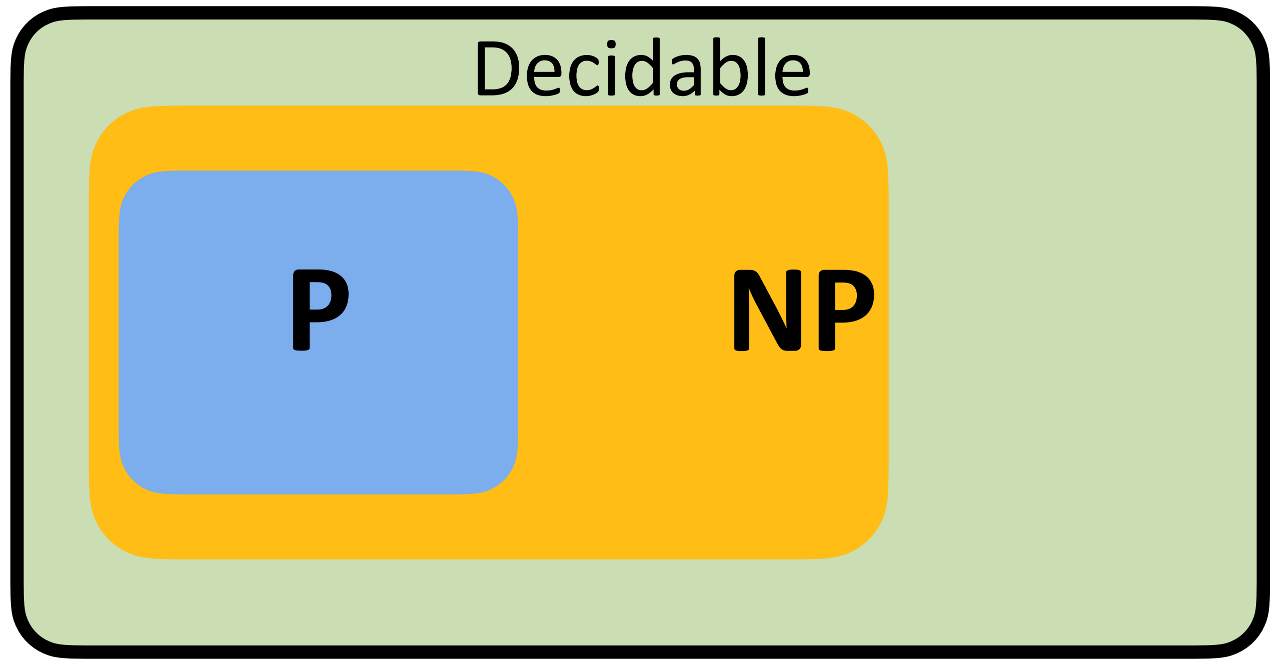
- A language L is (Turing-)**decidable** if **some Turing machine *decides* it**
  - The Turing machine accepts all strings in L and rejects all strings not in L
- Ex: $L = \{\text{prime number}\}$



Turing-Decidable Language

- A language L is (Turing-)**decidable** if **some Turing machine *decides* it**
  - The Turing machine accepts all strings in L and rejects all strings not in L
- All the problems in NP are decidable

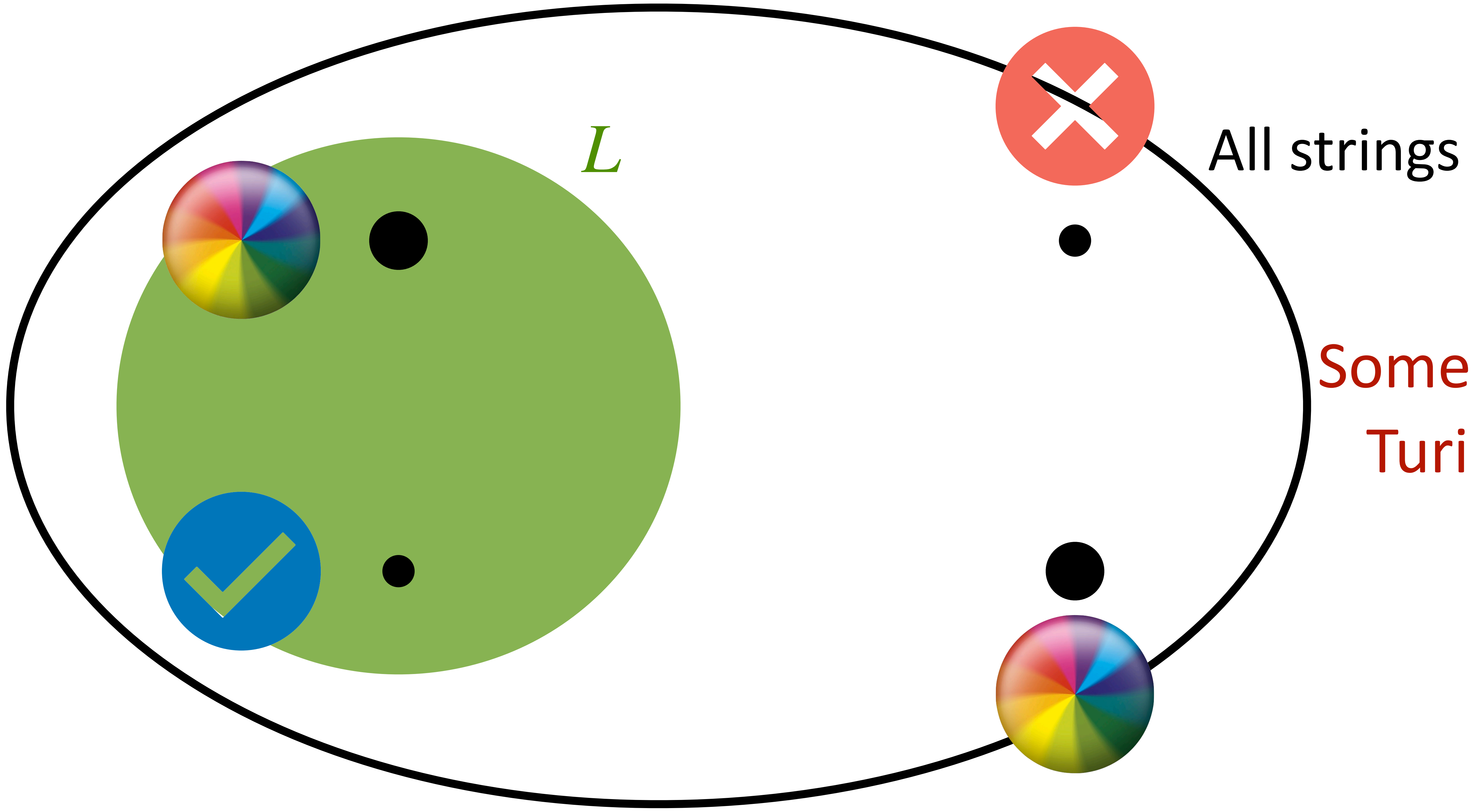




Undecidable Language

- A language L is **undecidable** if
  for all Turing machine M,
  there exists $w \in L$ such that M does not accept w or there exists $w \notin L$ such that M does not reject w

Some string in L make the Turing machine into a loop



Some string not in L make the Turing machine into a loop

Undecidable Languages

- $A_{TM} = \{\langle M, w\rangle \mid M$ is a Turing machine and M accepts input string $w\}$
- Halting problem:
  $HALT_{TM} = \{\langle M, w\rangle \mid M$ is a Turing machine and M accepts or rejects input string $w\}$
- Hilbert's 10th problem:
  $H = \{\langle p\rangle \mid p$ is a polynomial with an integral root$\}$
- Post correspondence problem (PCP):
  Given a collection D of dominos, each containing two strings, one on each side. A match is a list of these dominos (repetition permitted) such that the string on the top is the same as the string on the bottom.

Optimization? An Equivalent Decision Problem

- Machine minimization problem:
  - Given a set of jobs with processing time and feasible interval. Find a feasible schedule using minimum number of machines.

Optimization? An Equivalent Decision Problem

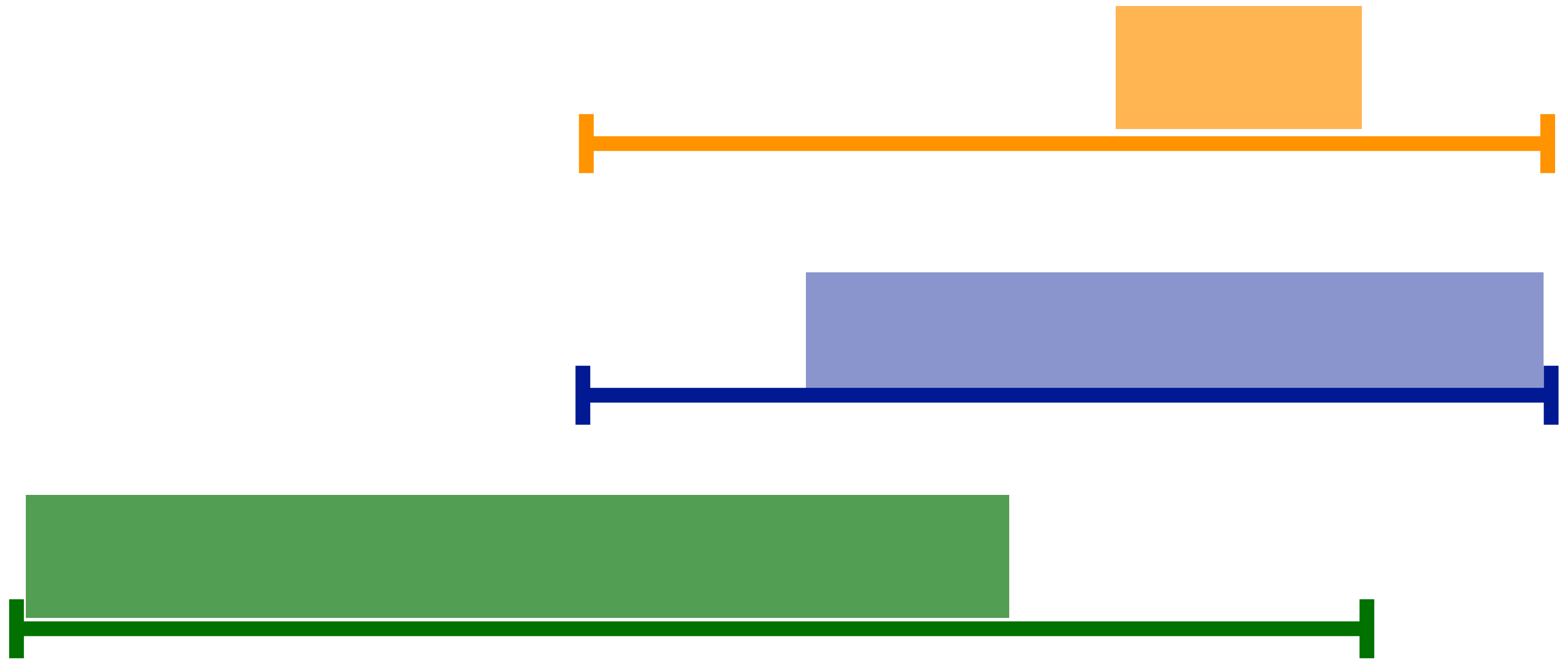
- Machine minimization problem:
  - Given a set of jobs with processing time and feasible interval. Find a feasible schedule using minimum number of machines.
- Decision version of machine minimization problem:

Optimization? An Equivalent Decision Problem

- Machine minimization problem:
  - Given a set of jobs with processing time and feasible interval. Find a feasible schedule using minimum number of machines.
- Decision version of machine minimization problem:
  - Given a set of jobs with processing time and feasible interval. Is there a feasible schedule that uses at most k machines.

MACHINE-MINIMIZATION

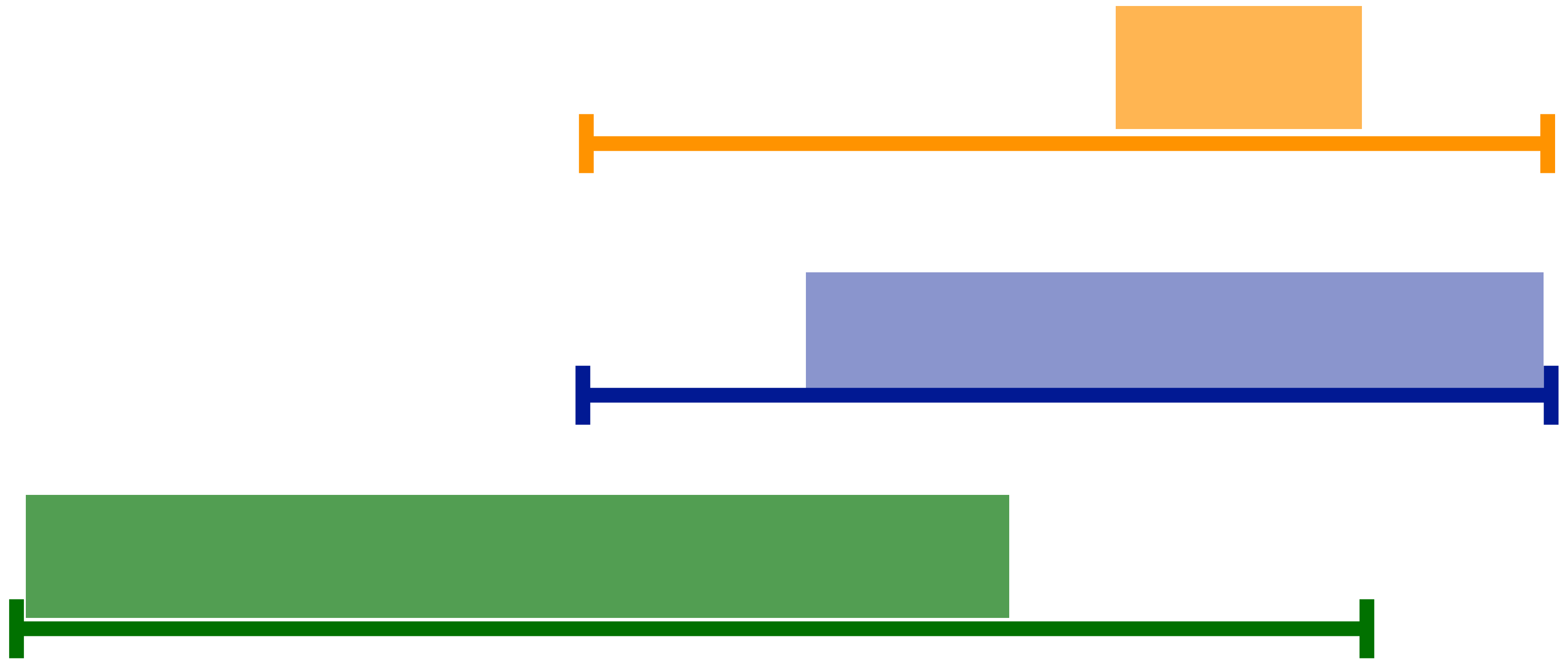
- MACHINE-MINIMIZATION = $\{\langle S, P, L, k\rangle \mid S = \{J_1, \cdots, J_n\},\ P = \{p_1, \cdots, p_n\}$, and $L = \{I_1, \cdots, I_n\}$. S is a set of jobs where each job J_i has processing time p_i and feasible interval I_i. The jobs in S can be feasibly scheduled on at most k machines.$\}$

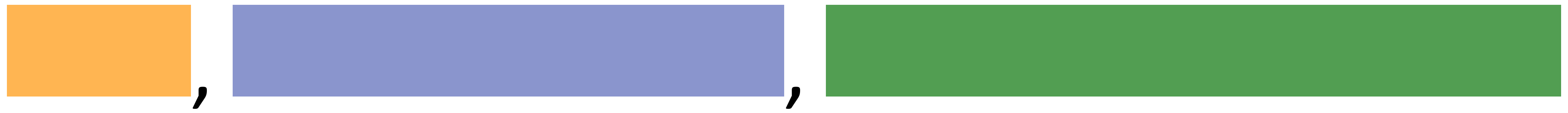
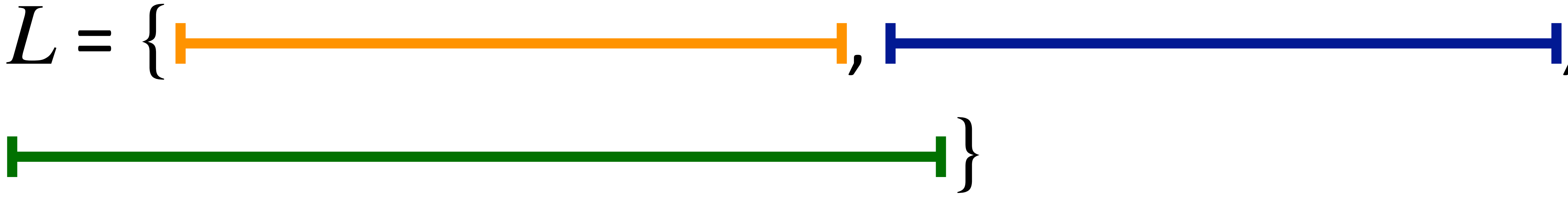


- S = {1, 2, 3}
- P = { , , }
- L = { , , }
- $k = 2$

MACHINE-MINIMIZATION

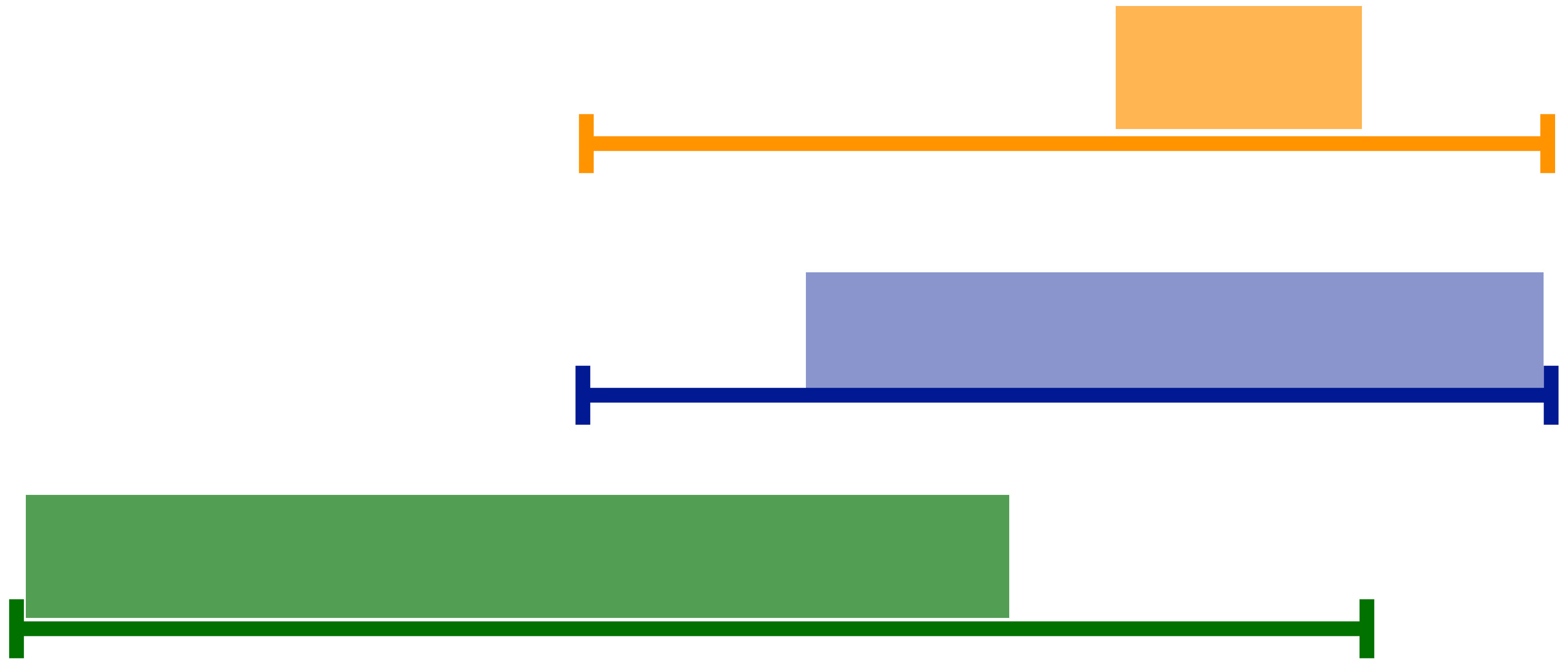
- MACHINE-MINIMIZATION = $\{\langle S, P, L, k\rangle \mid S = \{J_1, \cdots, J_n\},\ P = \{p_1, \cdots, p_n\}$, and $L = \{I_1, \cdots, I_n\}$. S is a set of jobs where each job J_i has processing time p_i and feasible interval I_i. The jobs in S can be feasibly scheduled on at most k machines.$\}$

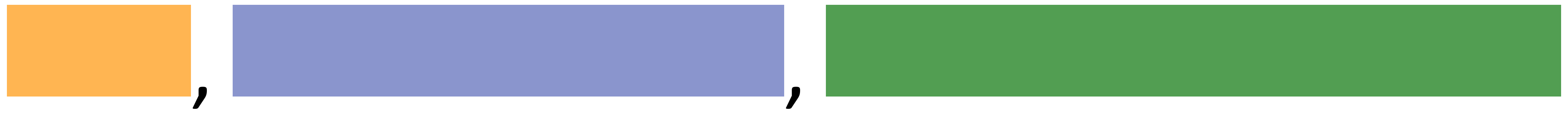
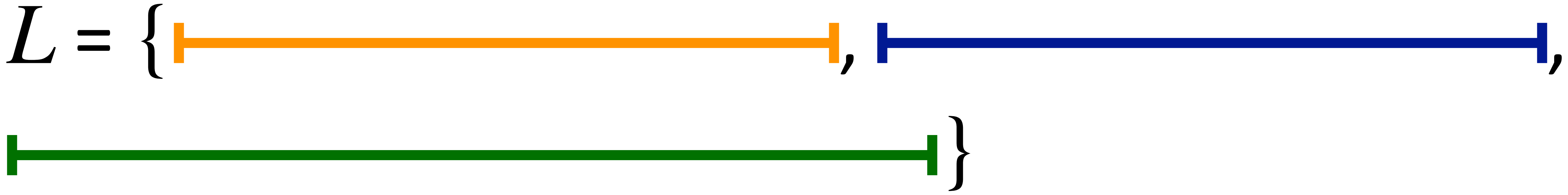


- S = {1, 2, 3}
- P = {  }
- L = {  }
- $k = 2$ Yes-instance

MACHINE-MINIMIZATION

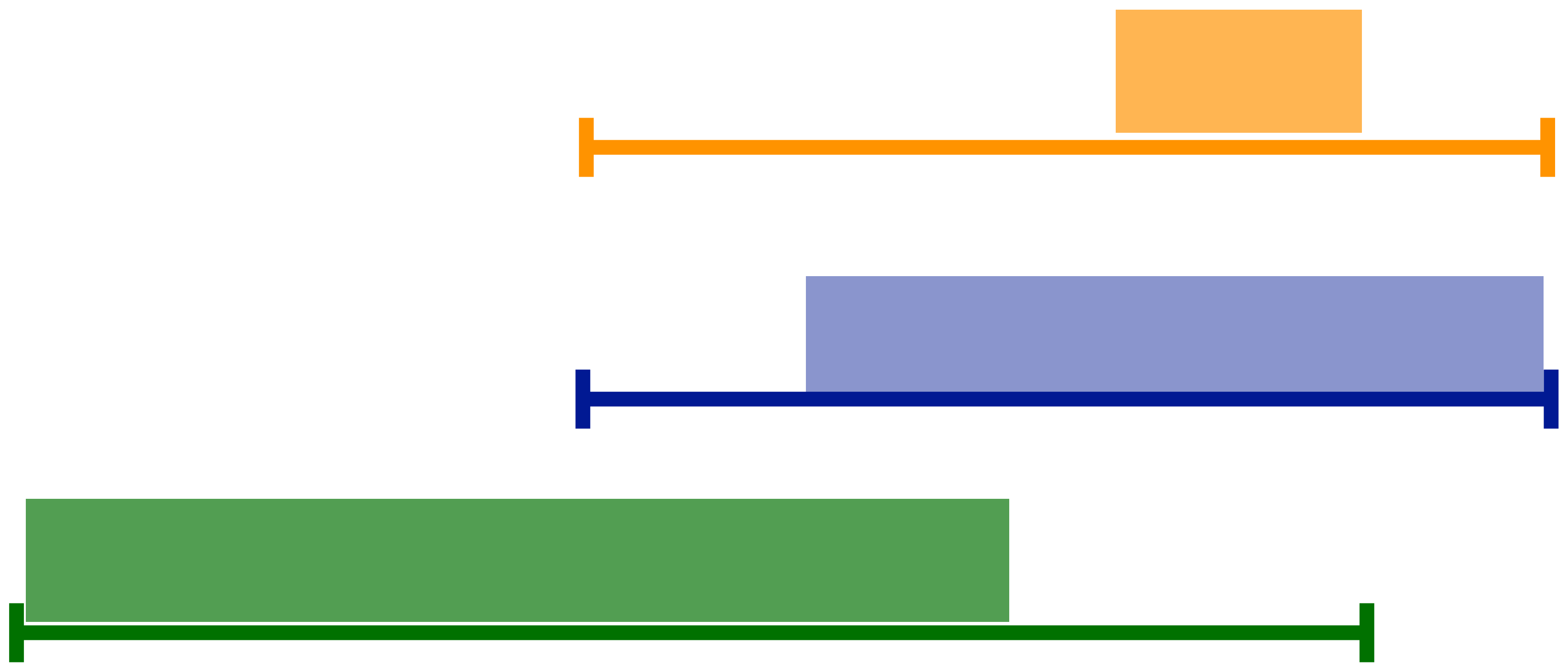
- MACHINE-MINIMIZATION = $\{\langle S, P, L, k\rangle \mid S = \{J_1, \cdots, J_n\},\ P = \{p_1, \cdots, p_n\}$, and $L = \{I_1, \cdots, I_n\}$. S is a set of jobs where each job J_i has processing time p_i and feasible interval I_i. The jobs in S can be feasibly scheduled on at most k machines.$\}$

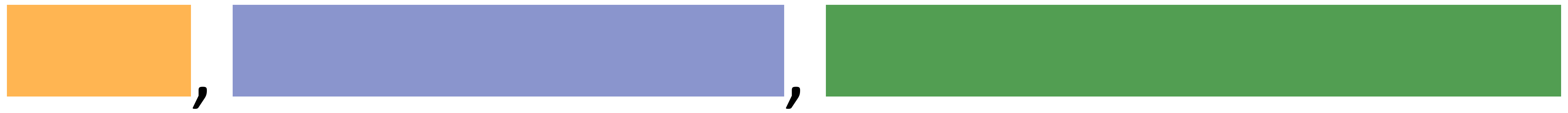
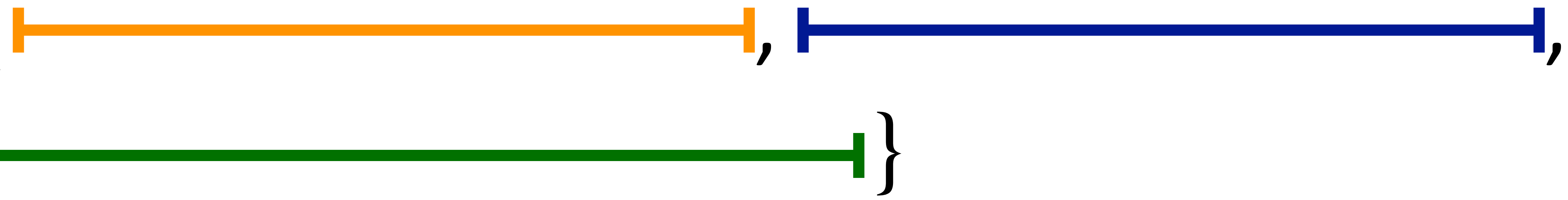


- S = {1, 2, 3}
- P = {  }
- L = {  }
- k = 1

MACHINE-MINIMIZATION

- MACHINE-MINIMIZATION = $\{\langle S, P, L, k\rangle \mid S = \{J_1, \cdots, J_n\},\ P = \{p_1, \cdots, p_n\}$, and $L = \{I_1, \cdots, I_n\}$. S is a set of jobs where each job J_i has processing time p_i and feasible interval I_i. The jobs in S can be feasibly scheduled on at most k machines.$\}$



- S = {1, 2, 3}
- P = {}
- L = {}
- k = 1 No-instance