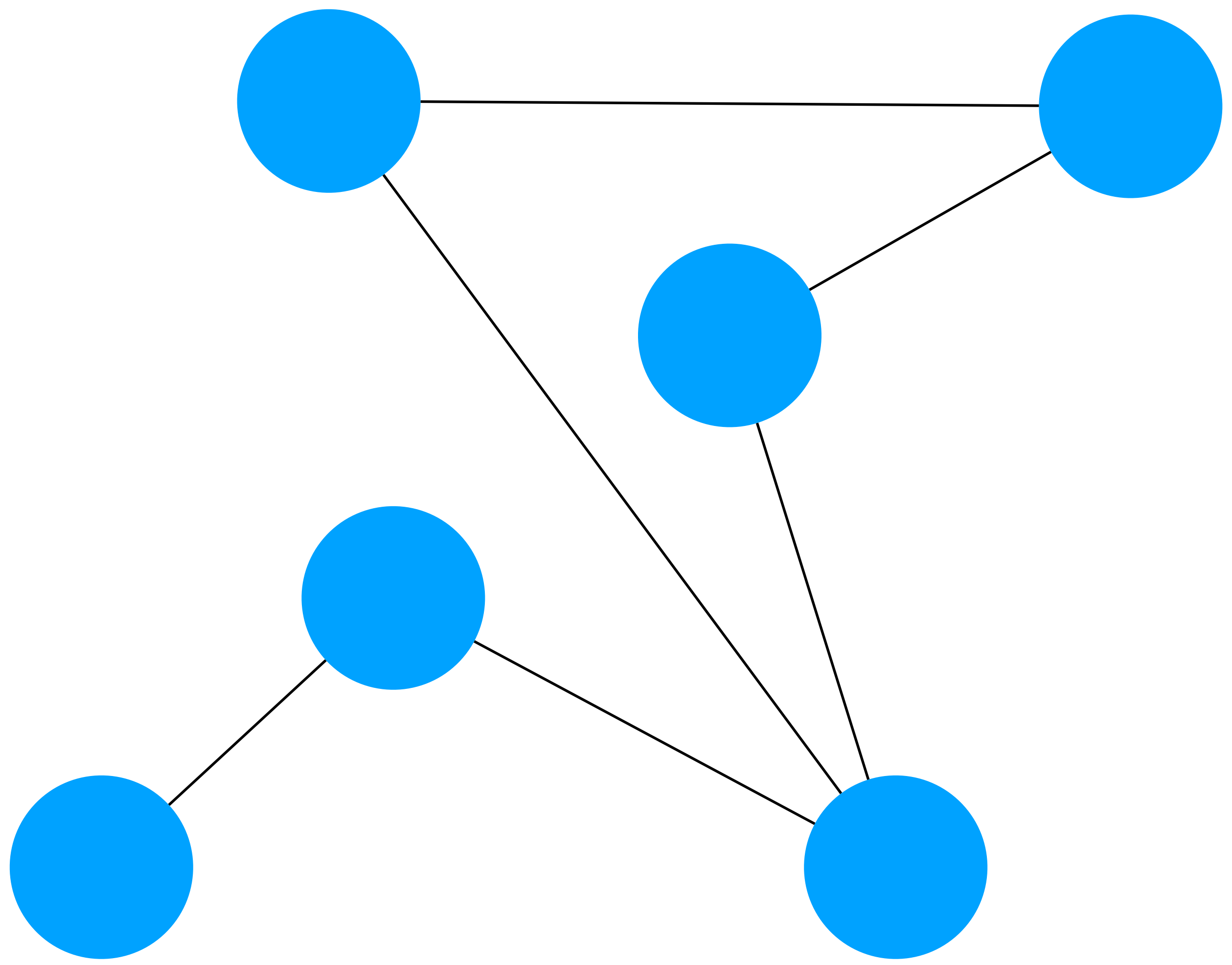
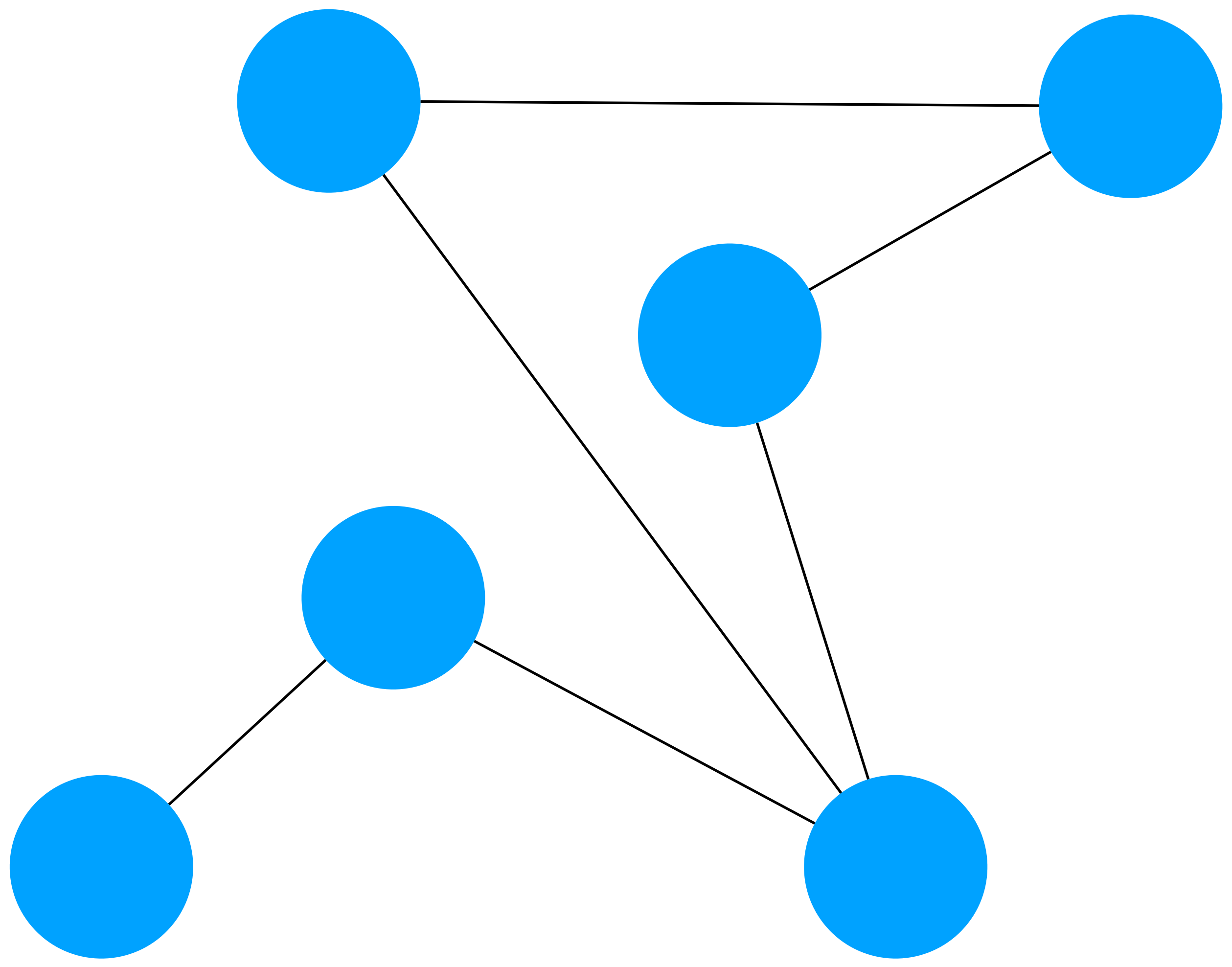
Algorithmics Part 4
Structural Decompositions and Algorithms

Friedrich Slivovsky

Elimination Orderings



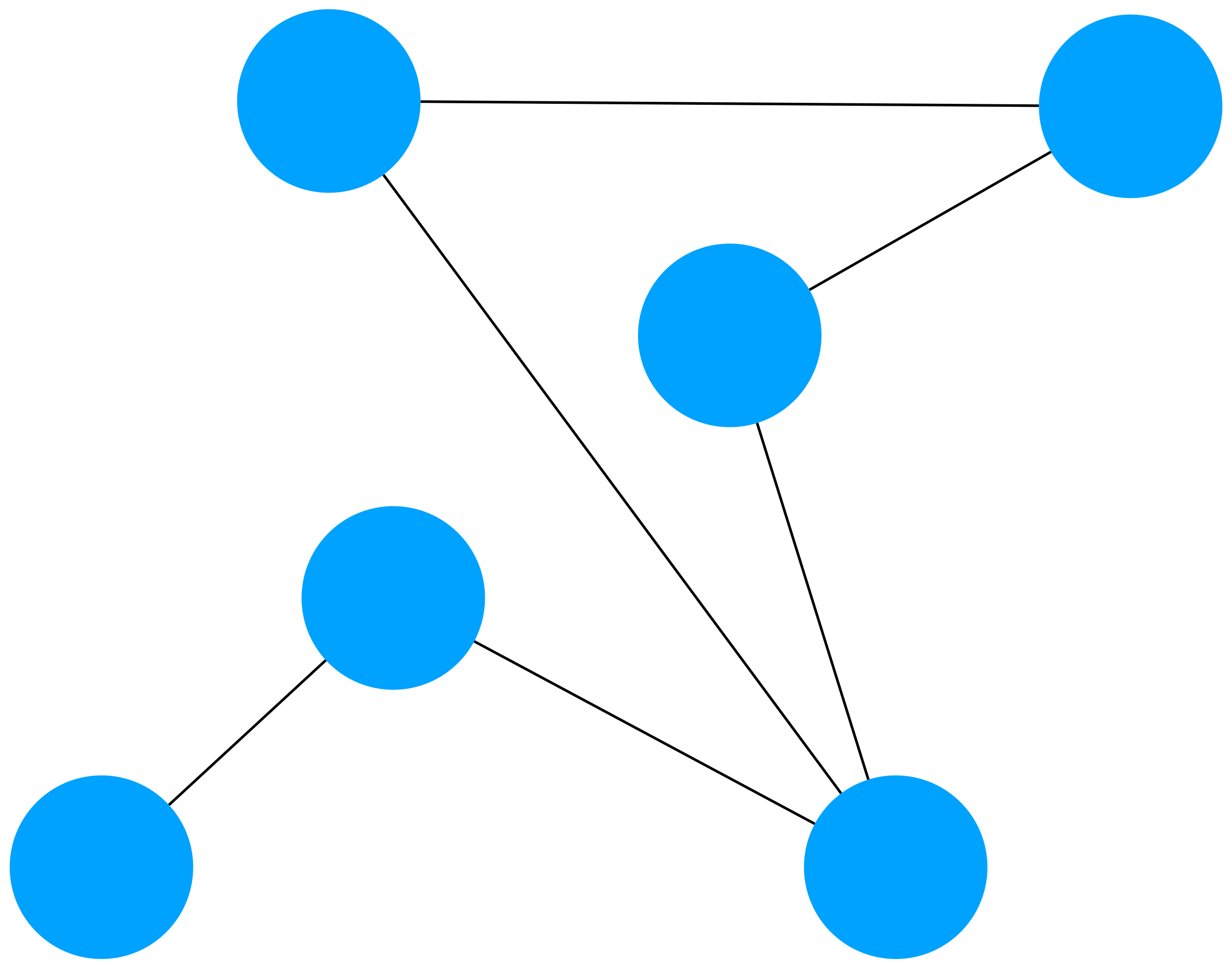
Elimination Orderings



Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Elimination Orderings

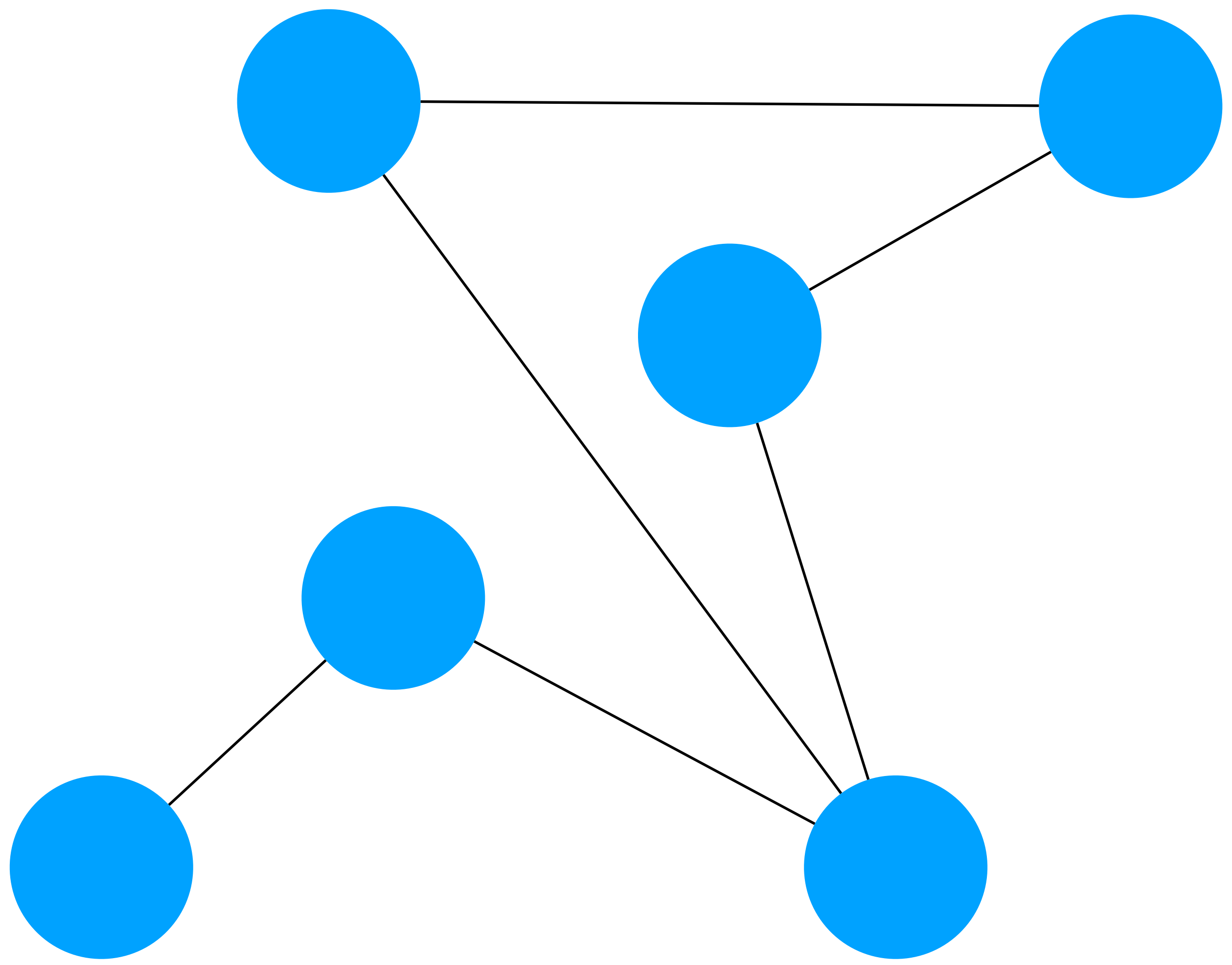


Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.

Elimination Orderings

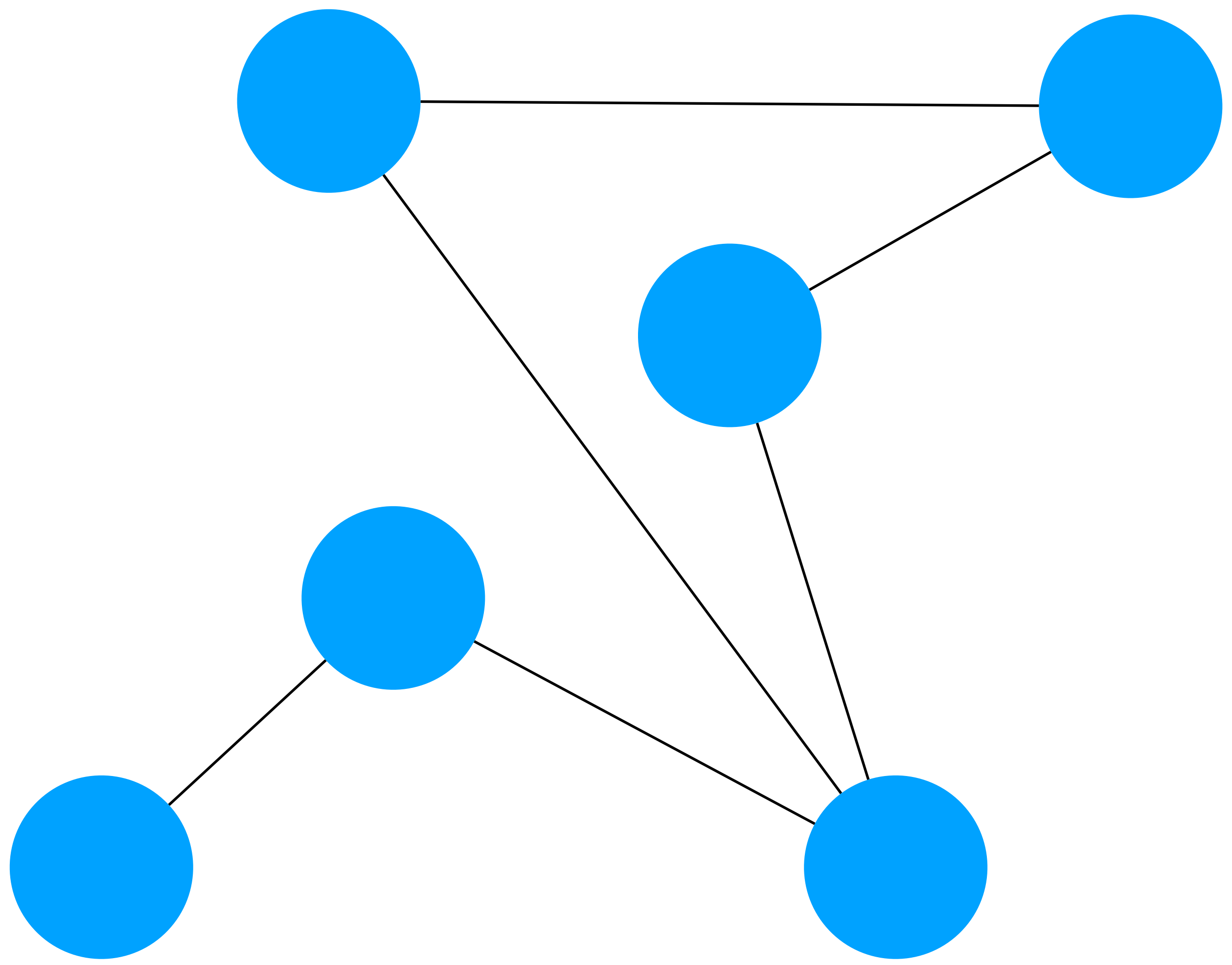


Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.

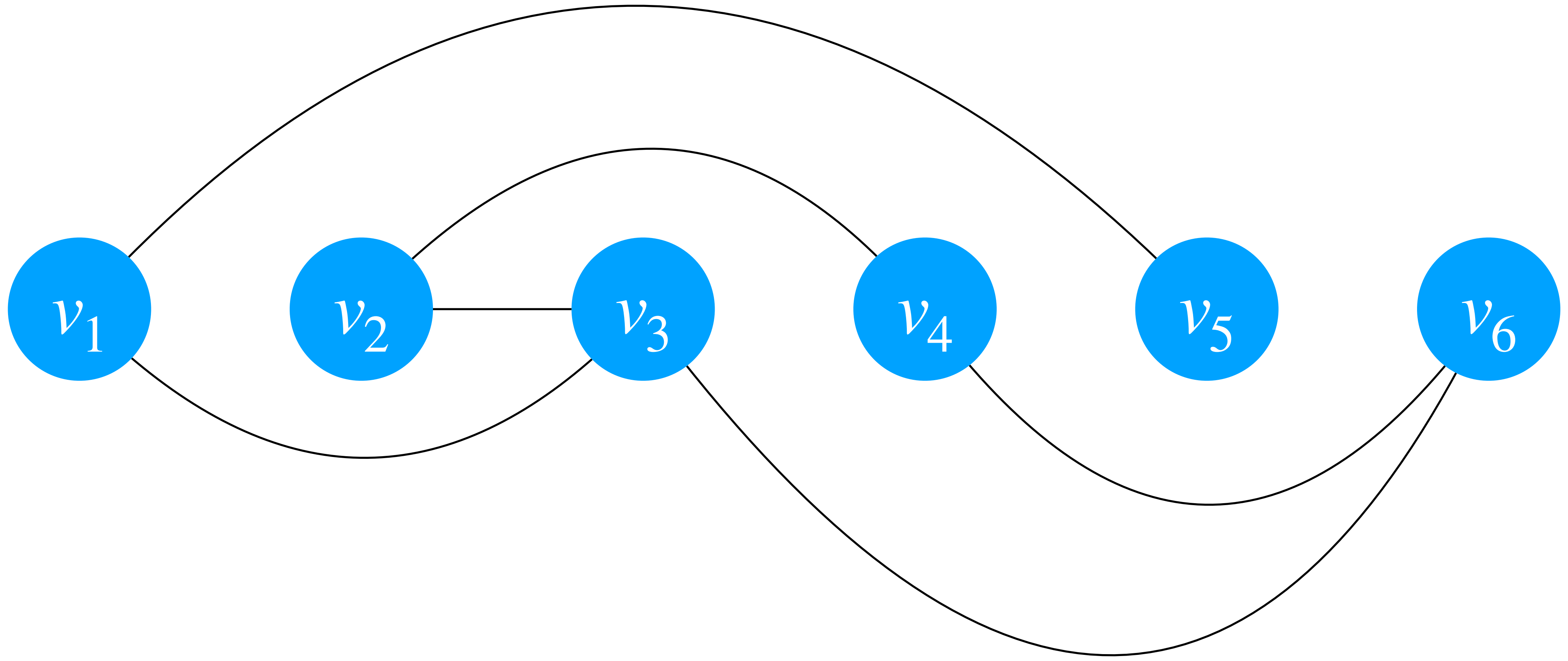
Elimination Orderings



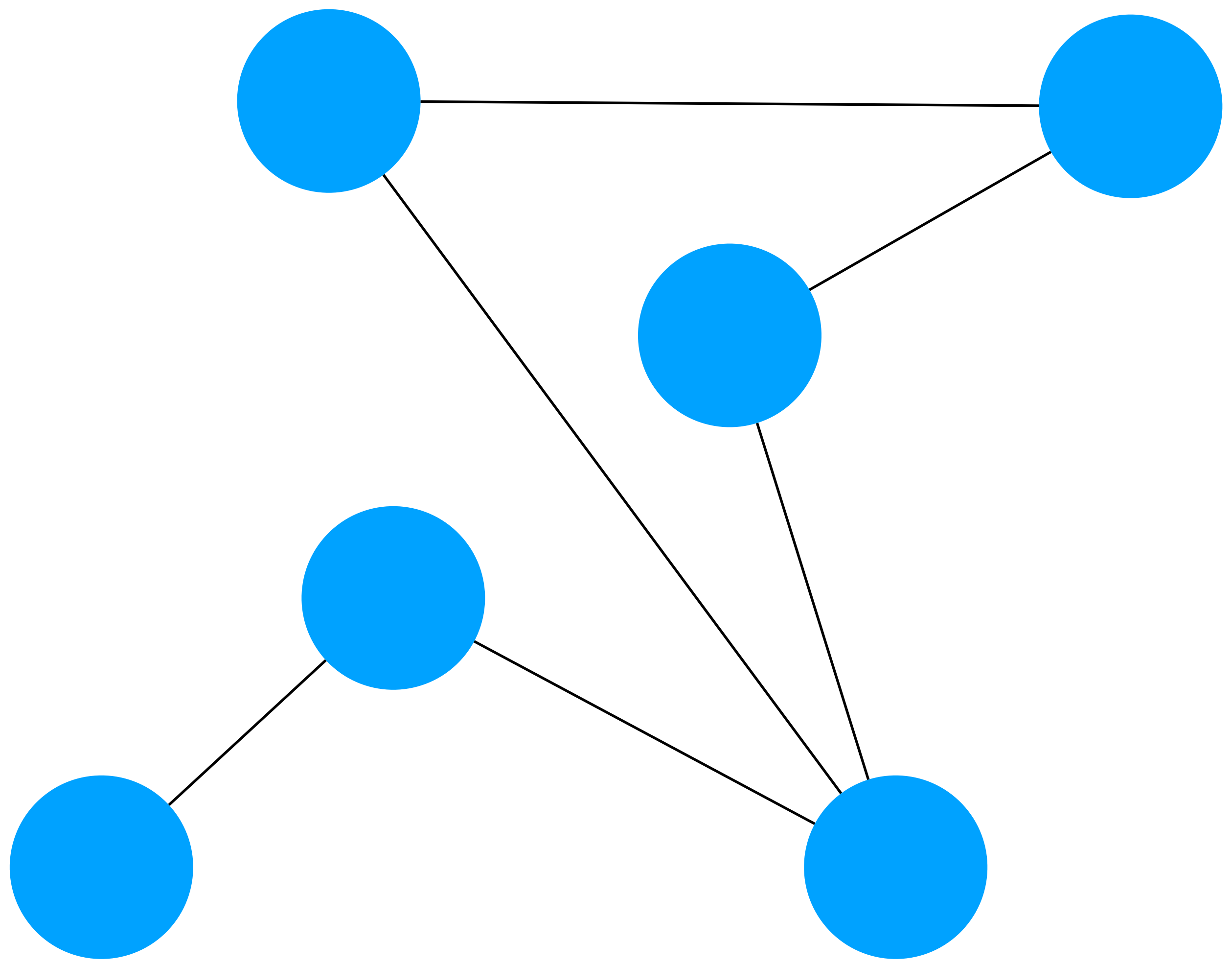
Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.



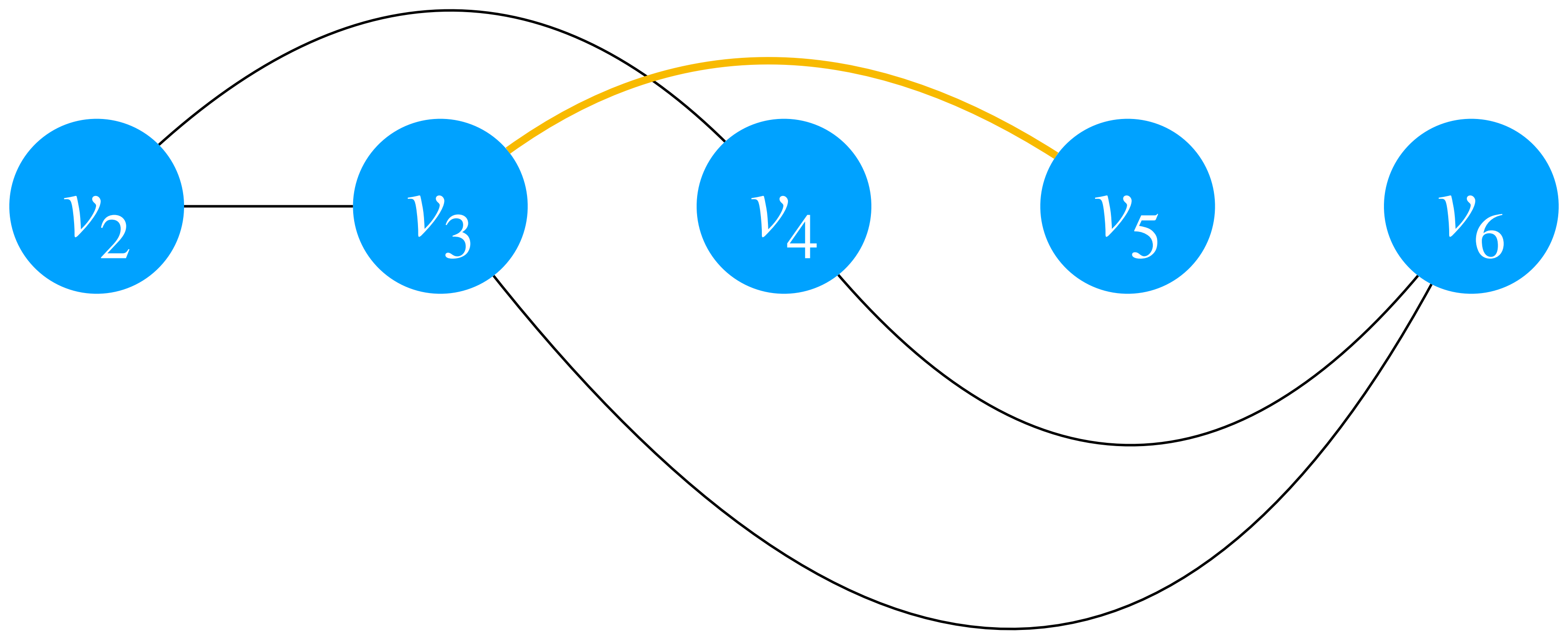
Elimination Orderings



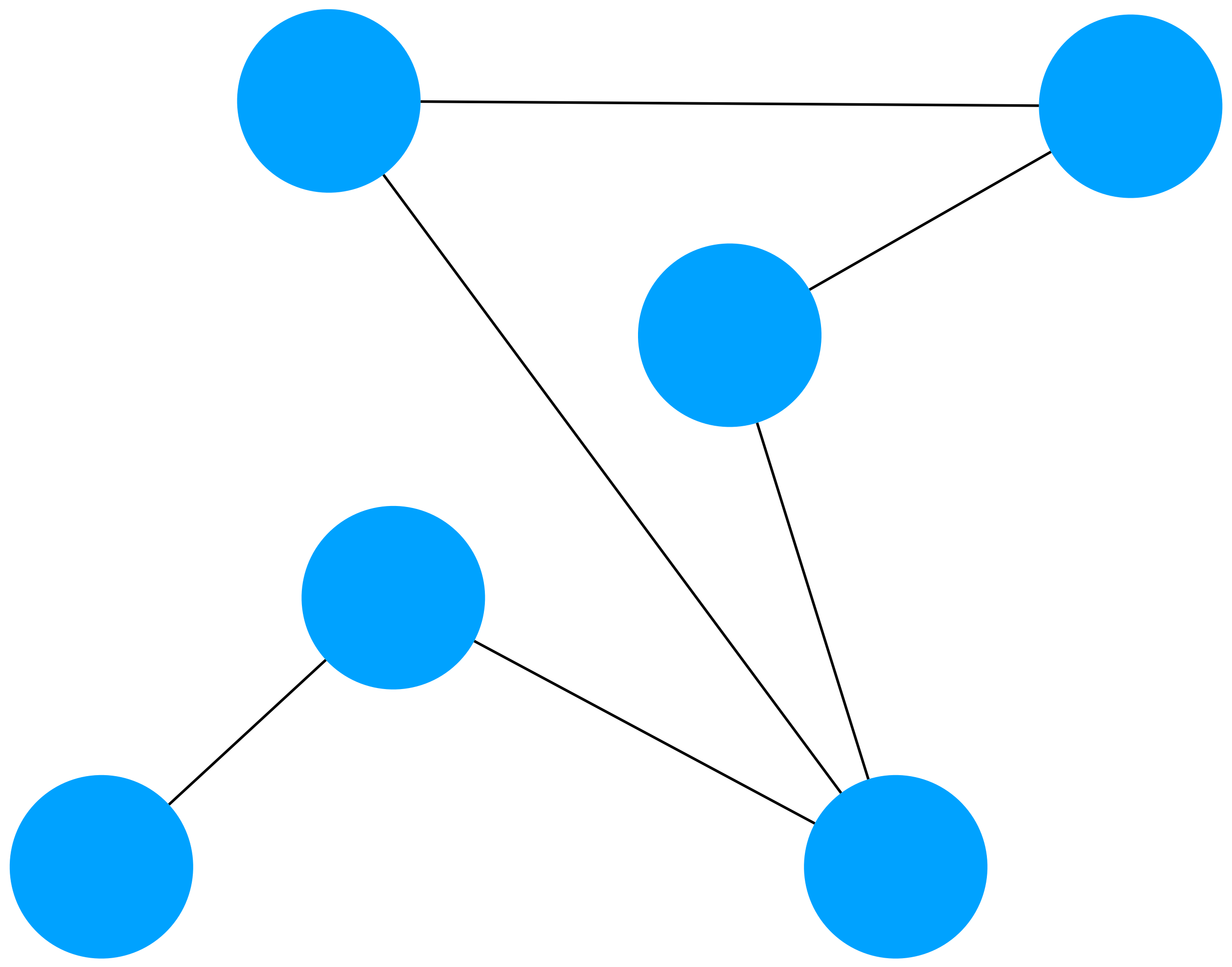
Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.



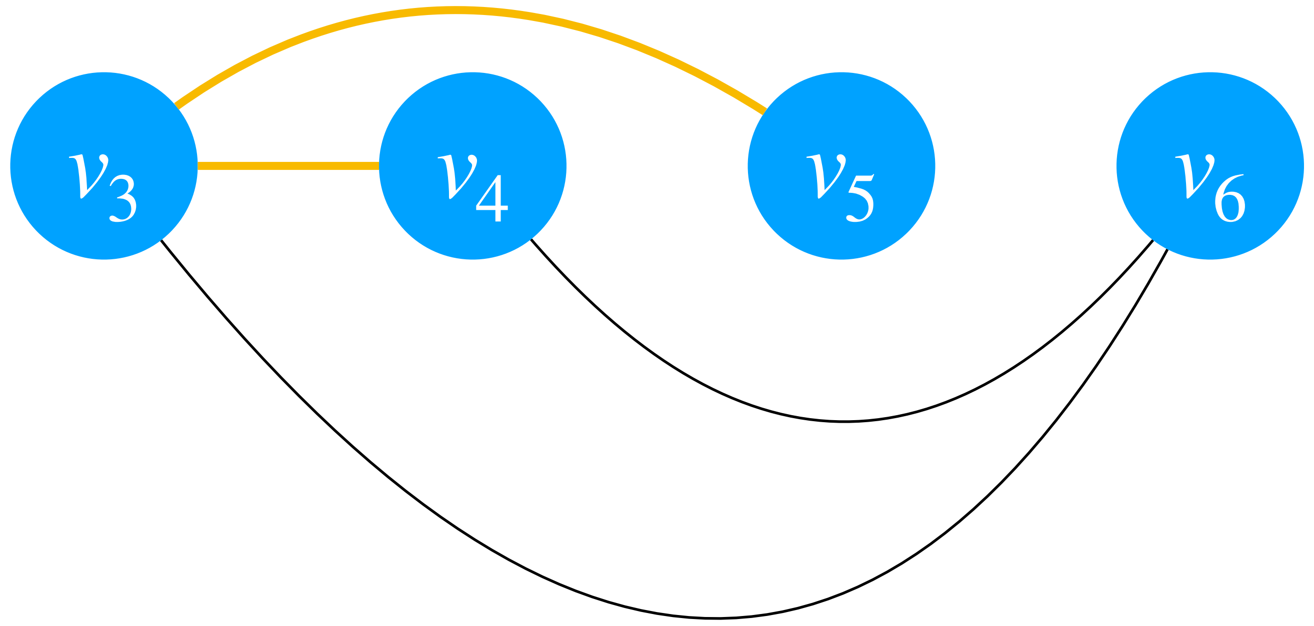
Elimination Orderings



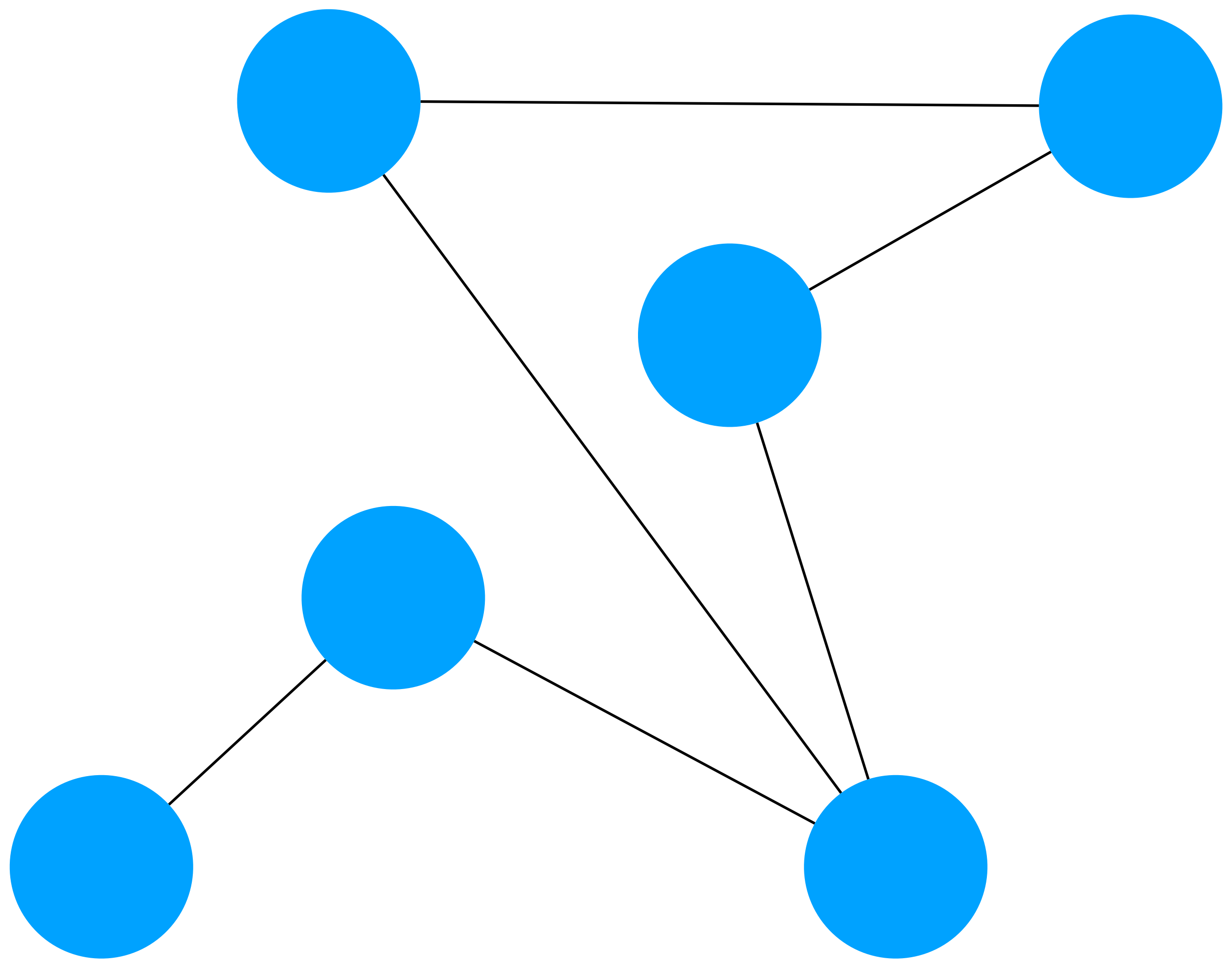
Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.



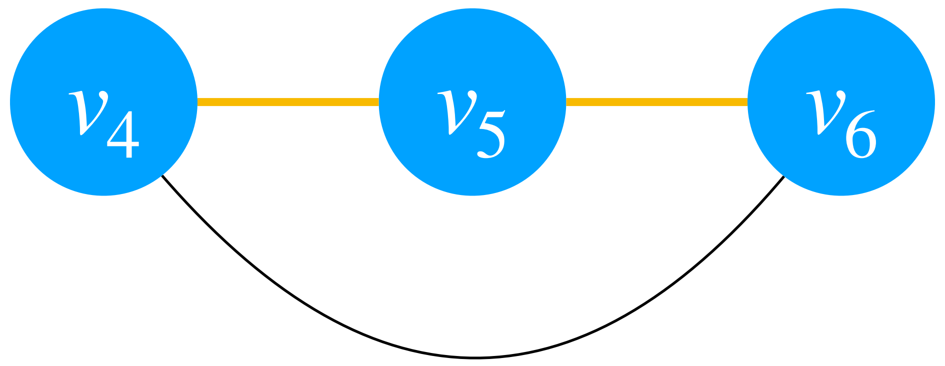
Elimination Orderings



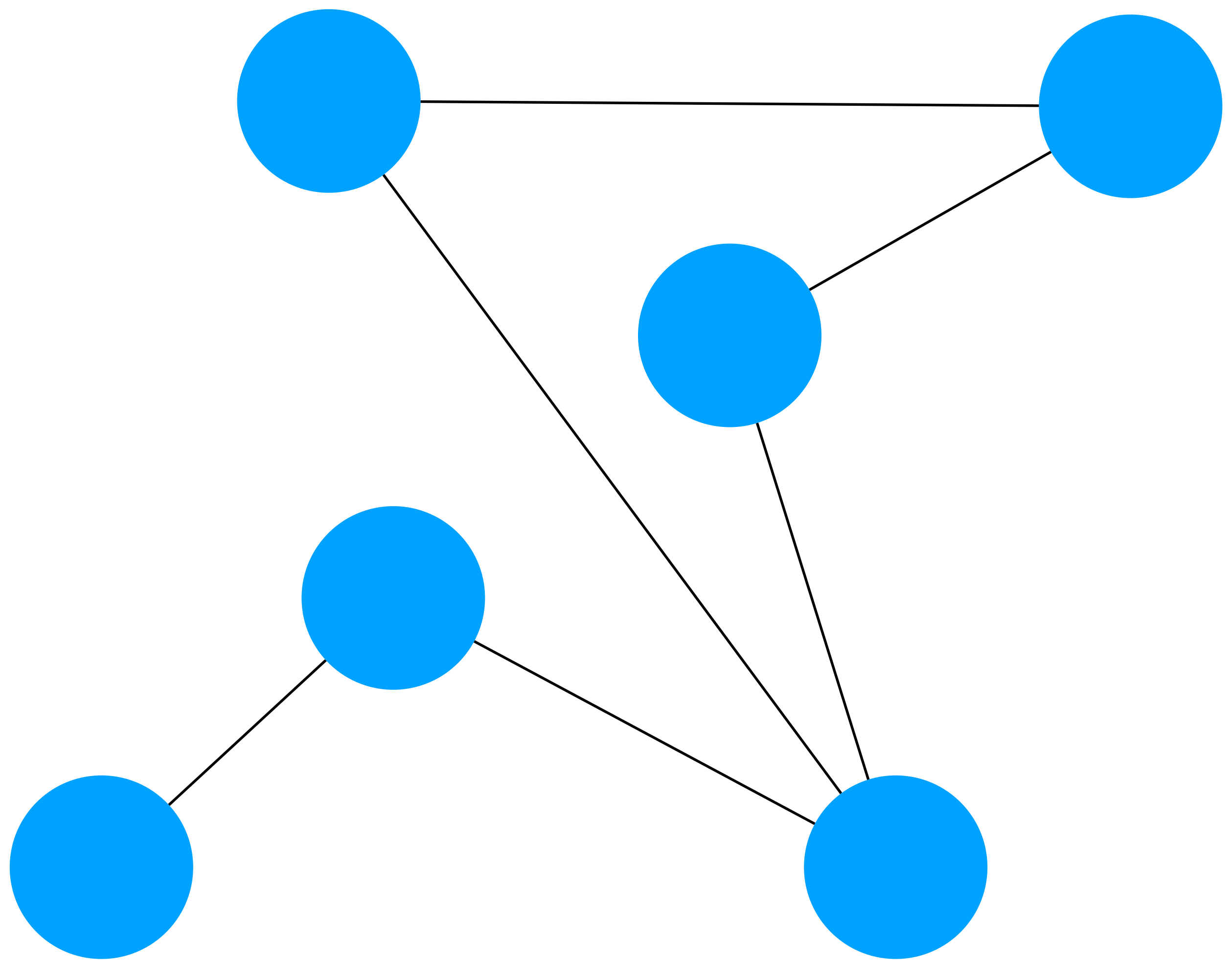
Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.



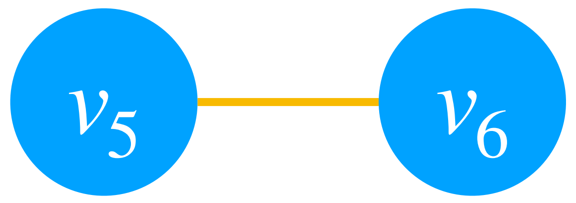
Elimination Orderings



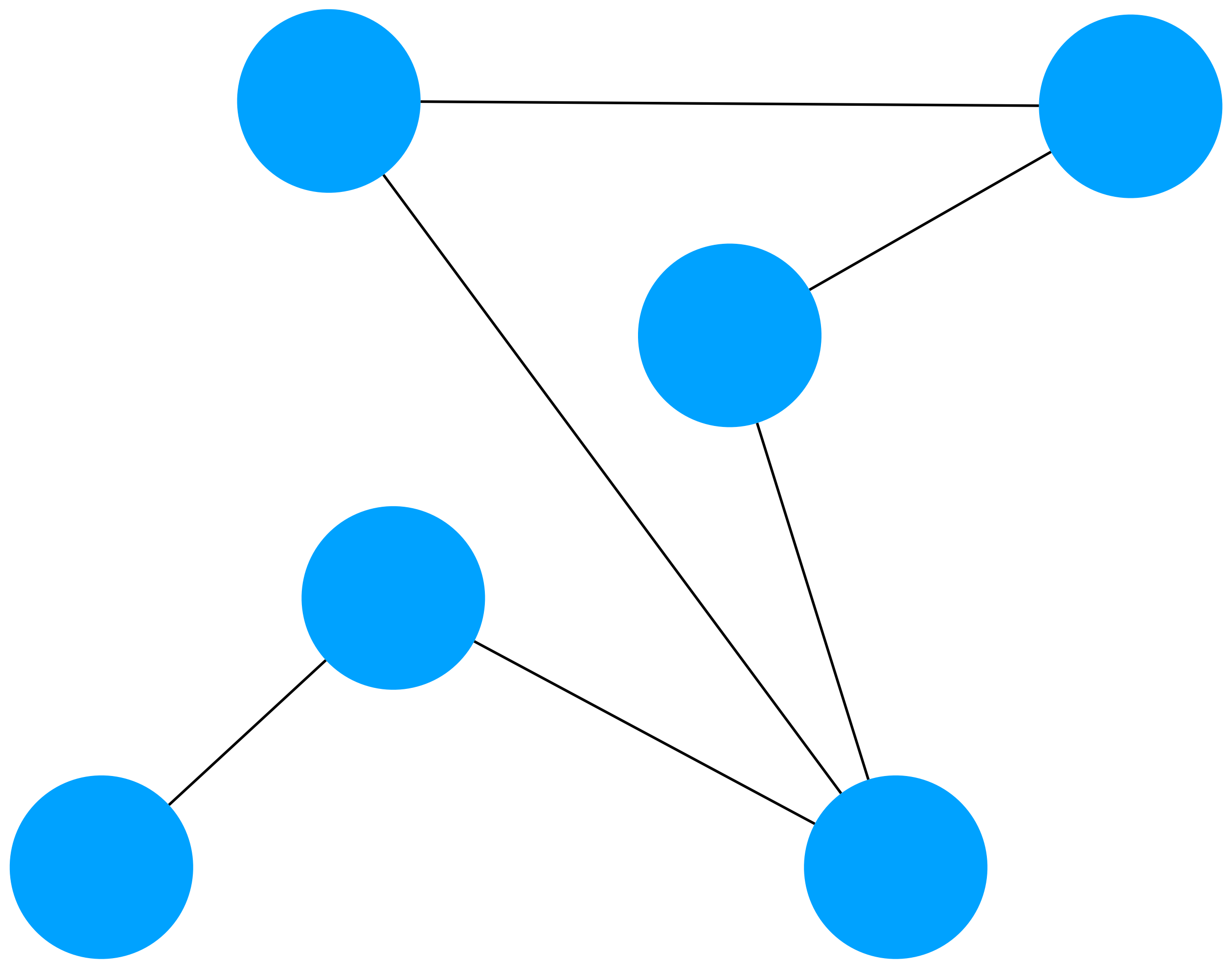
Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.



Elimination Orderings

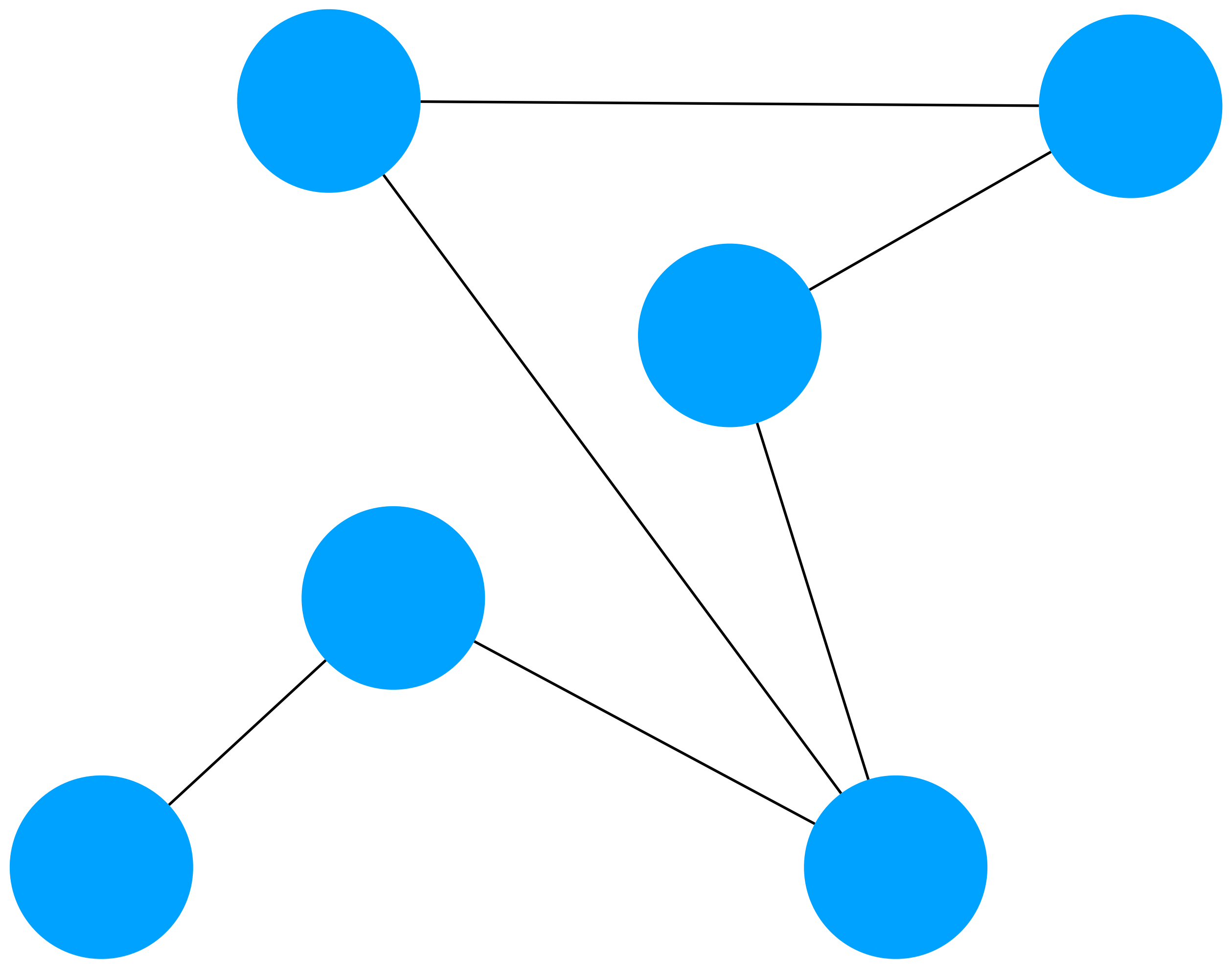


Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.

Elimination Orderings

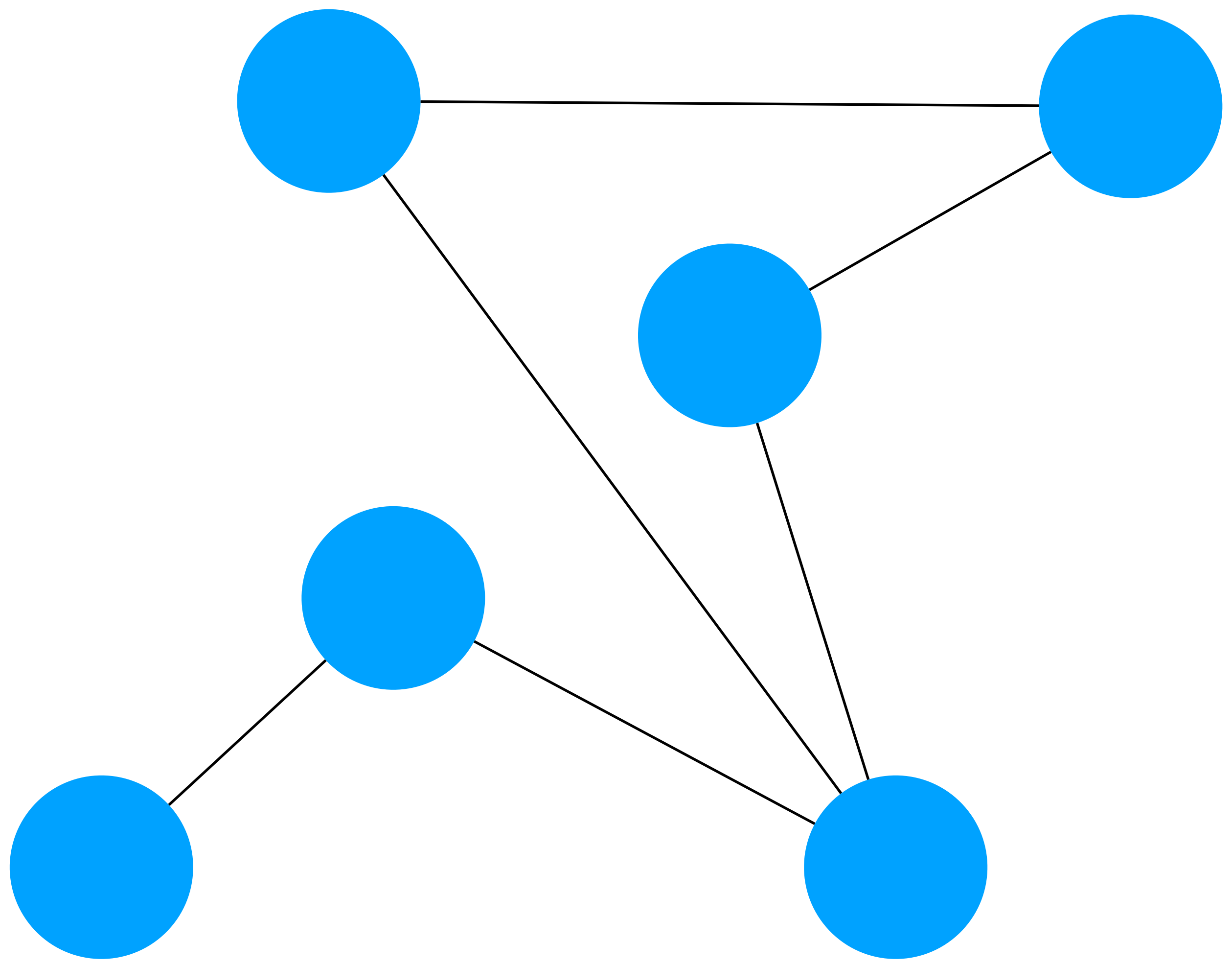


Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.

Elimination Orderings



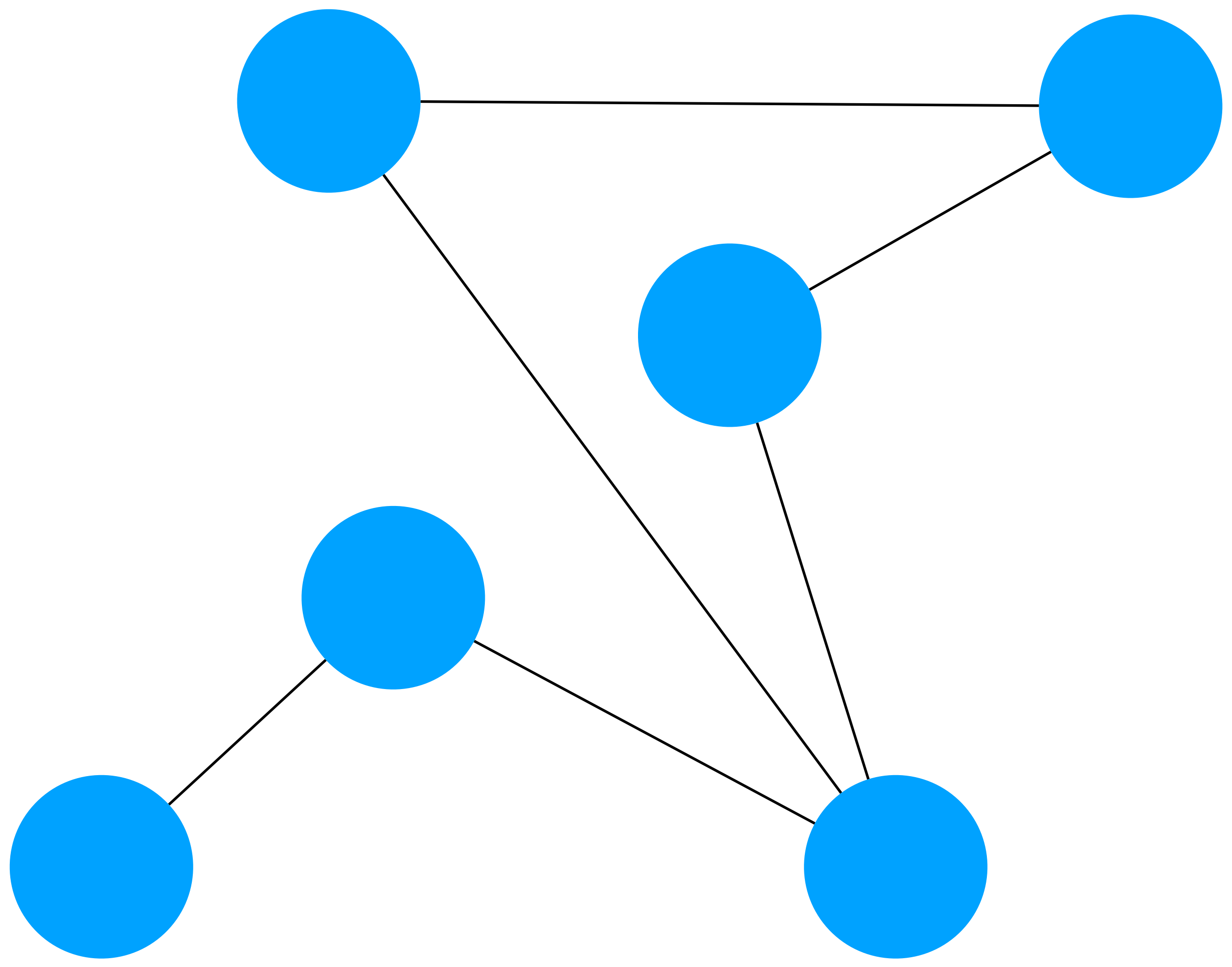
Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.

fill-in edges

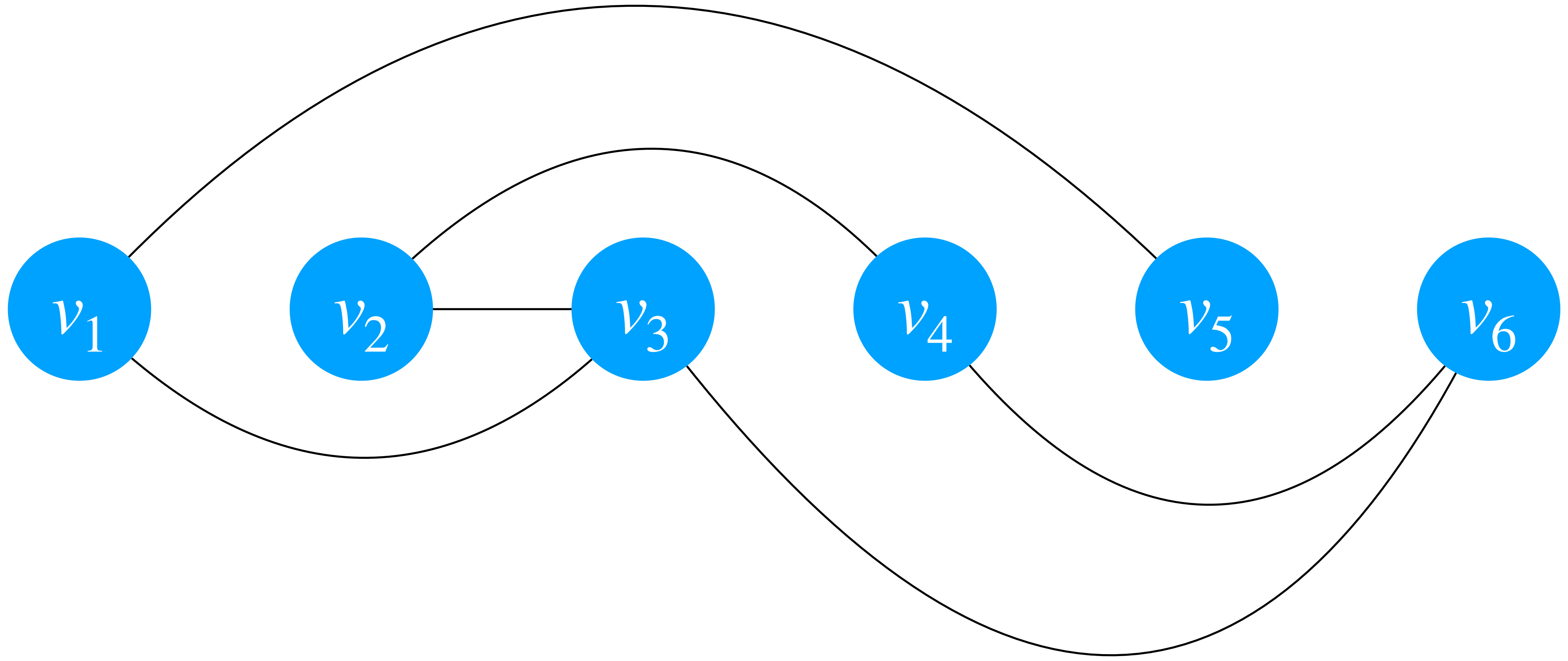
Elimination Orderings



Definition

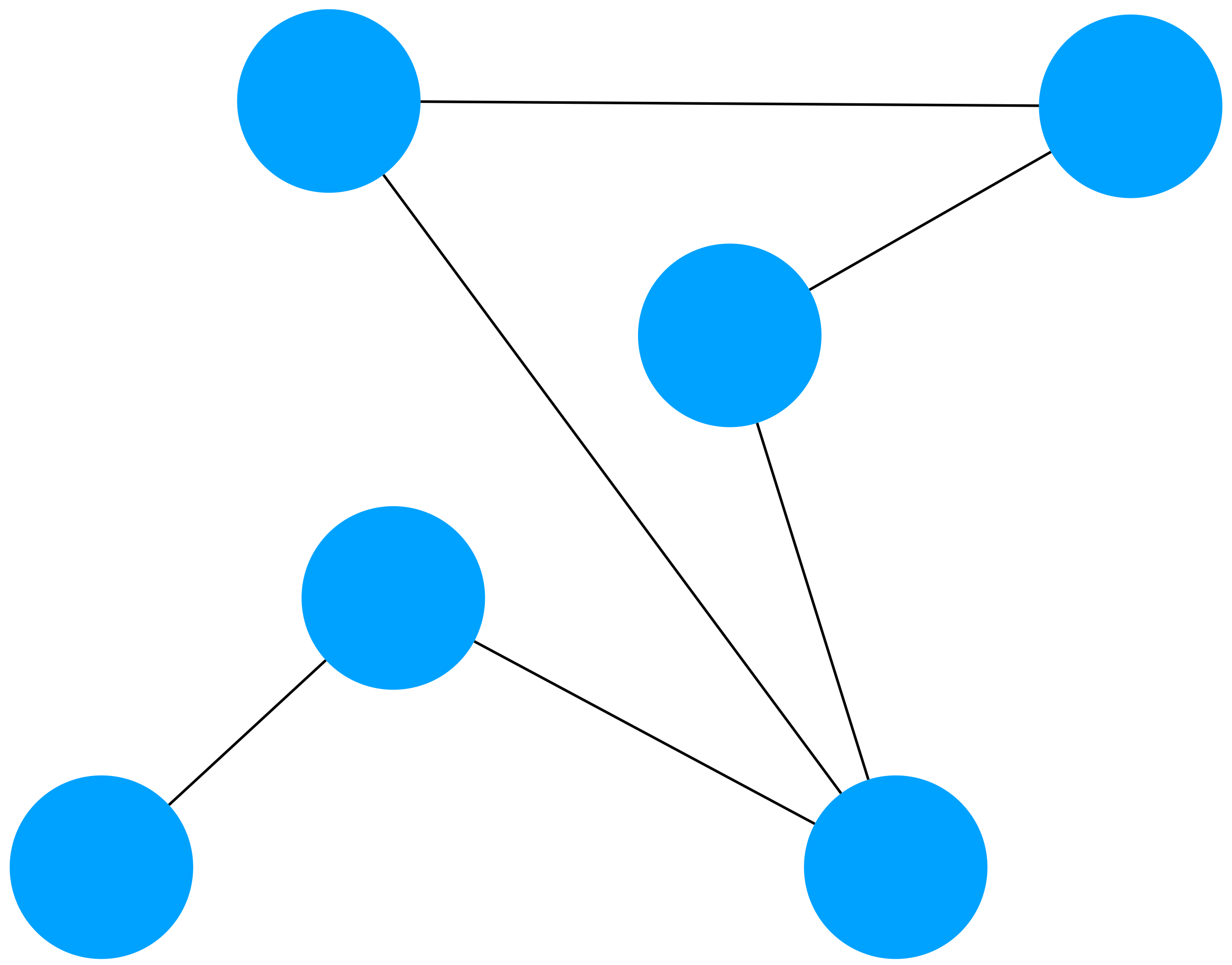
Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.



The **width** of σ is the maximum degree of a vertex upon elimination.

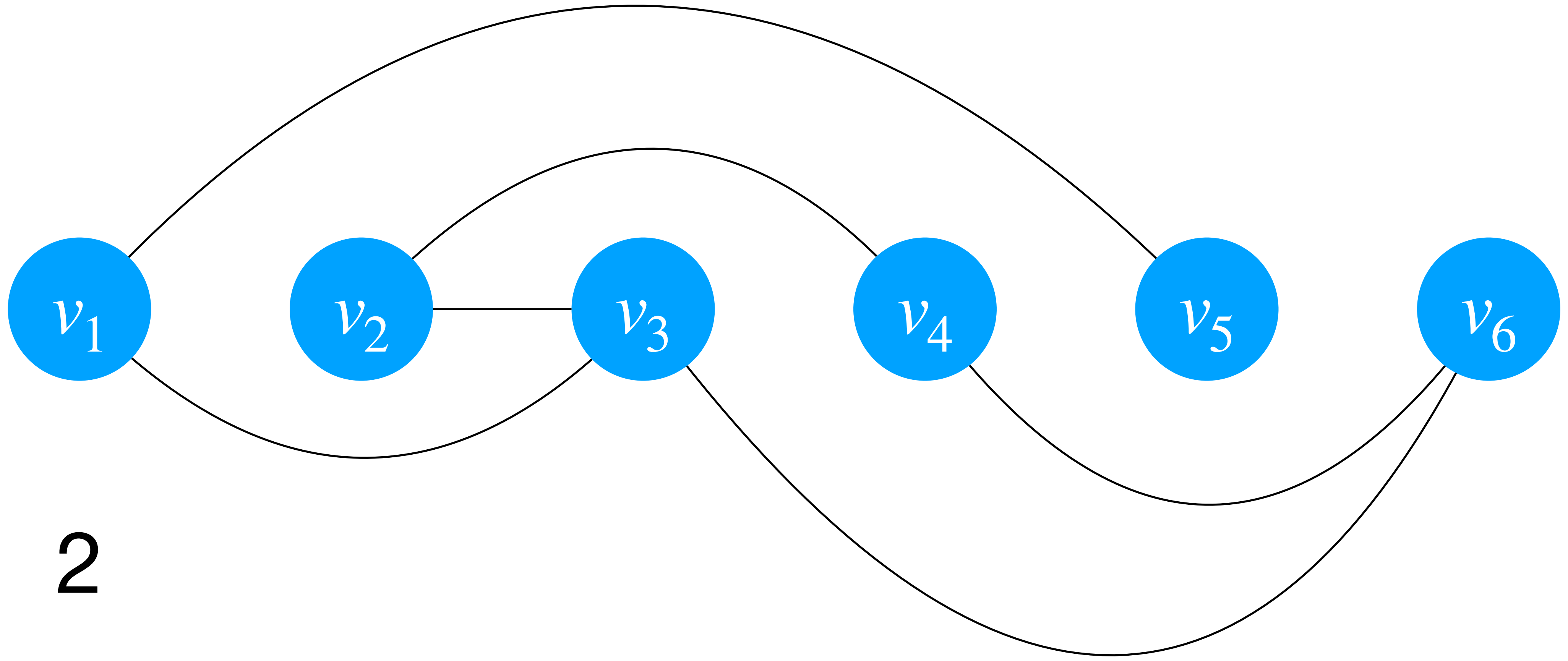
Elimination Orderings



Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

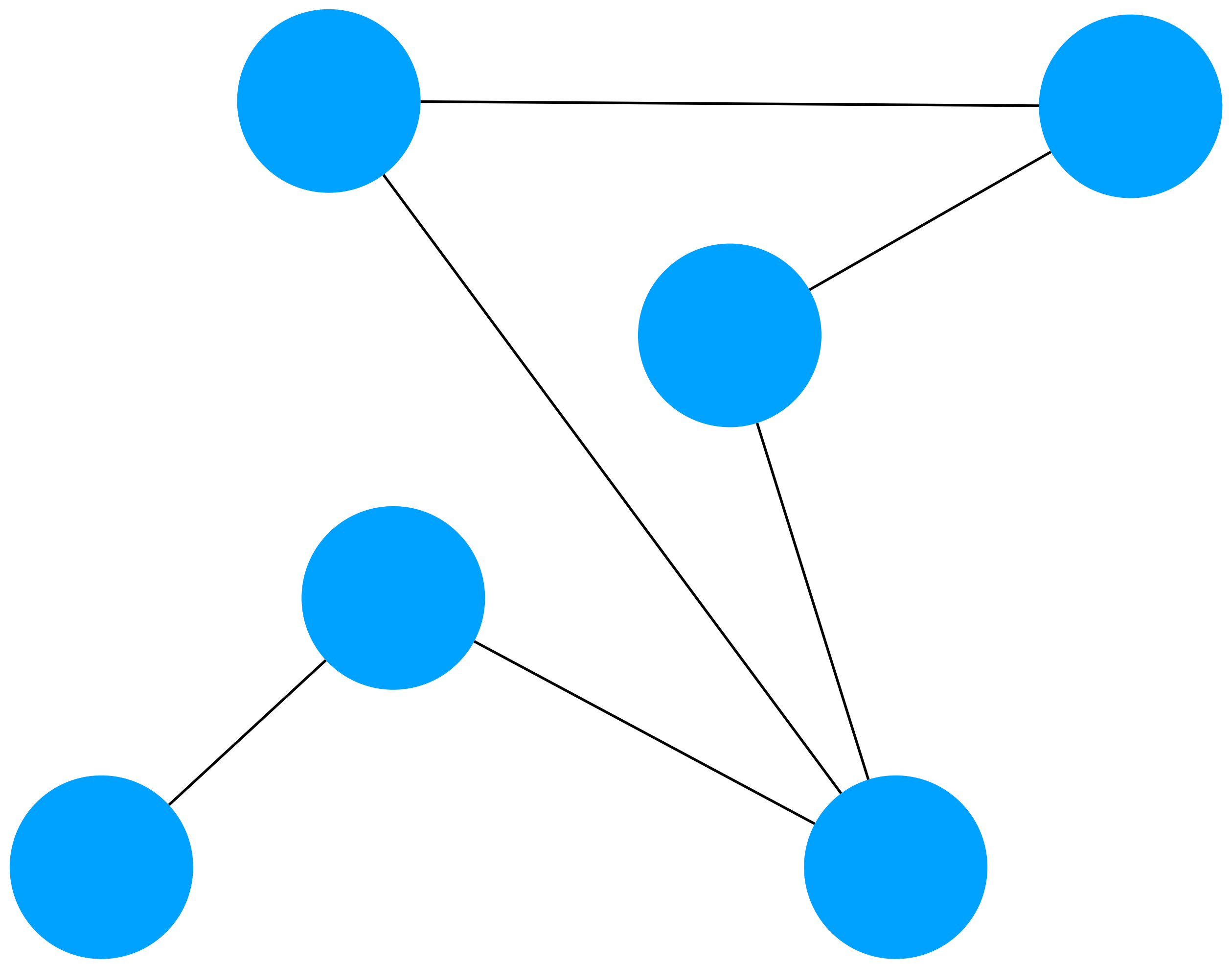
Upon eliminating v_i create edges between its neighbors.



2

The **width** of σ is the maximum degree of a vertex upon elimination.

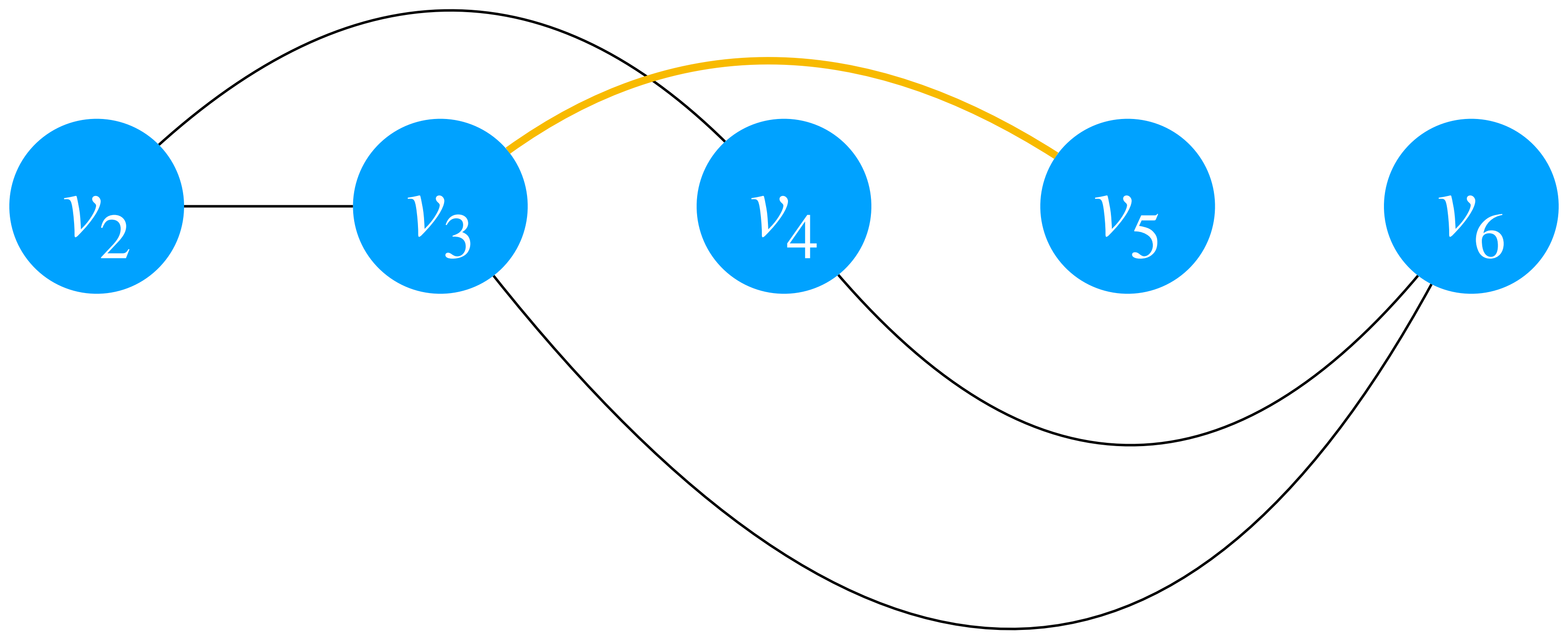
Elimination Orderings



Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

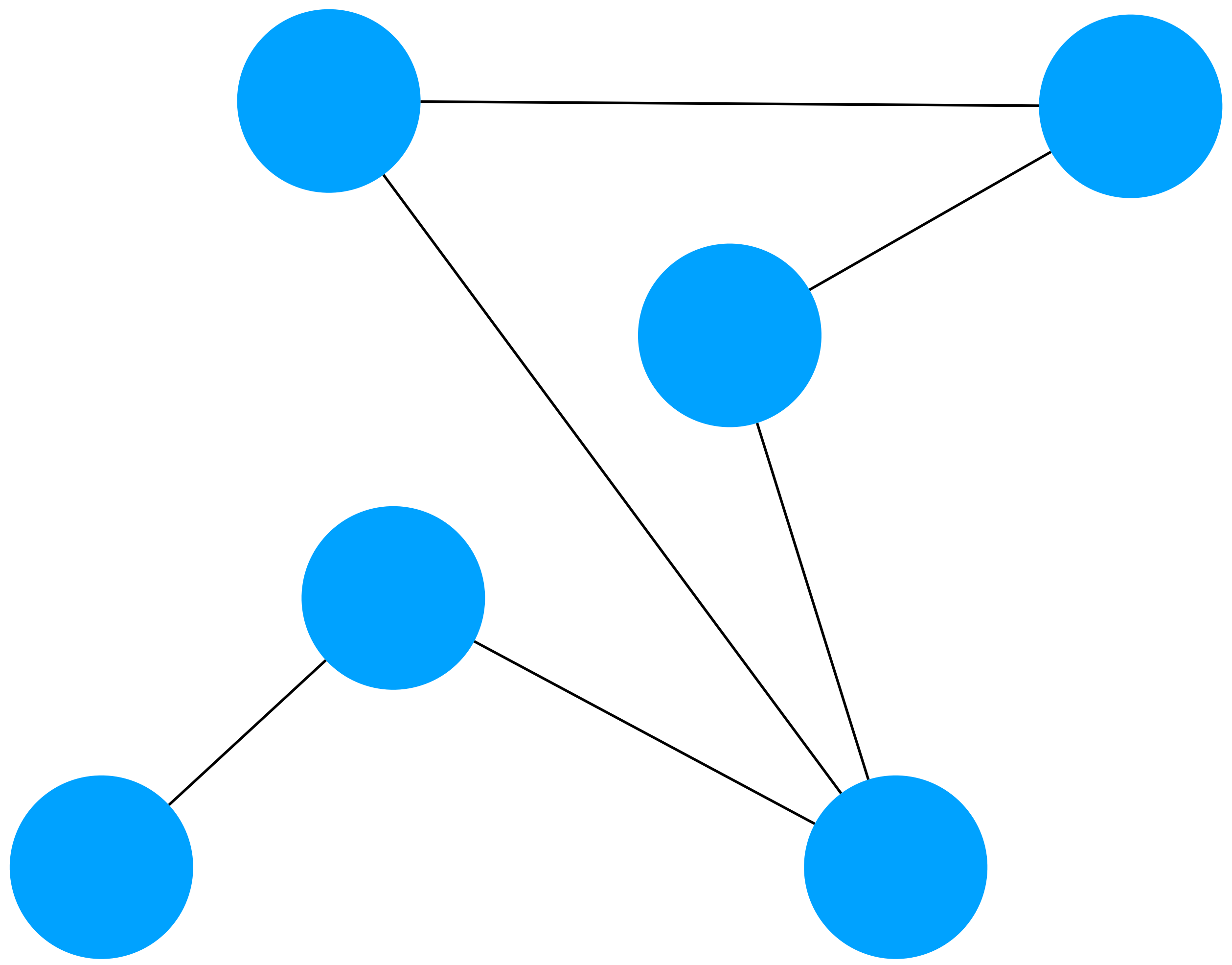
Upon eliminating v_i create edges between its neighbors.



2

The **width** of σ is the maximum degree of a vertex upon elimination.

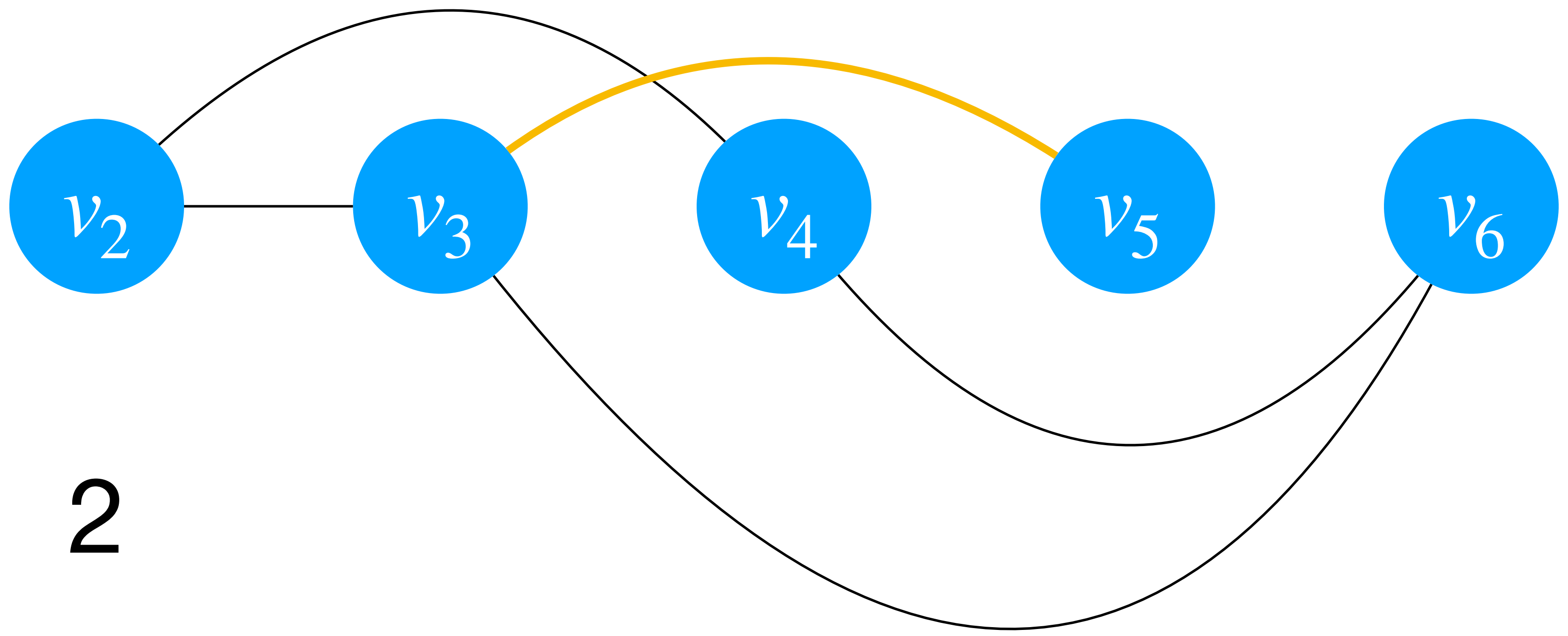
Elimination Orderings



Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

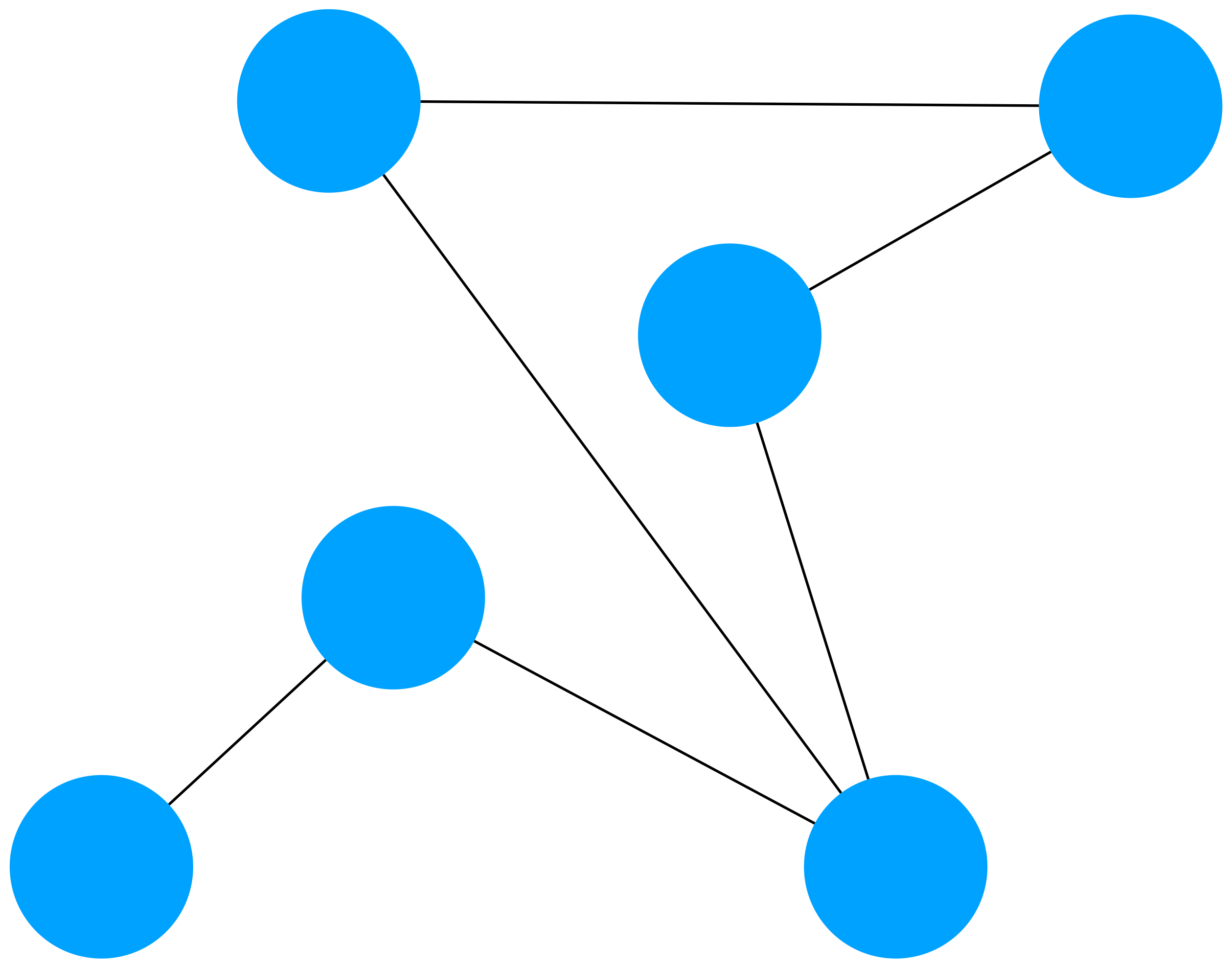
Upon eliminating v_i create edges between its neighbors.



2 2

The **width** of σ is the maximum degree of a vertex upon elimination.

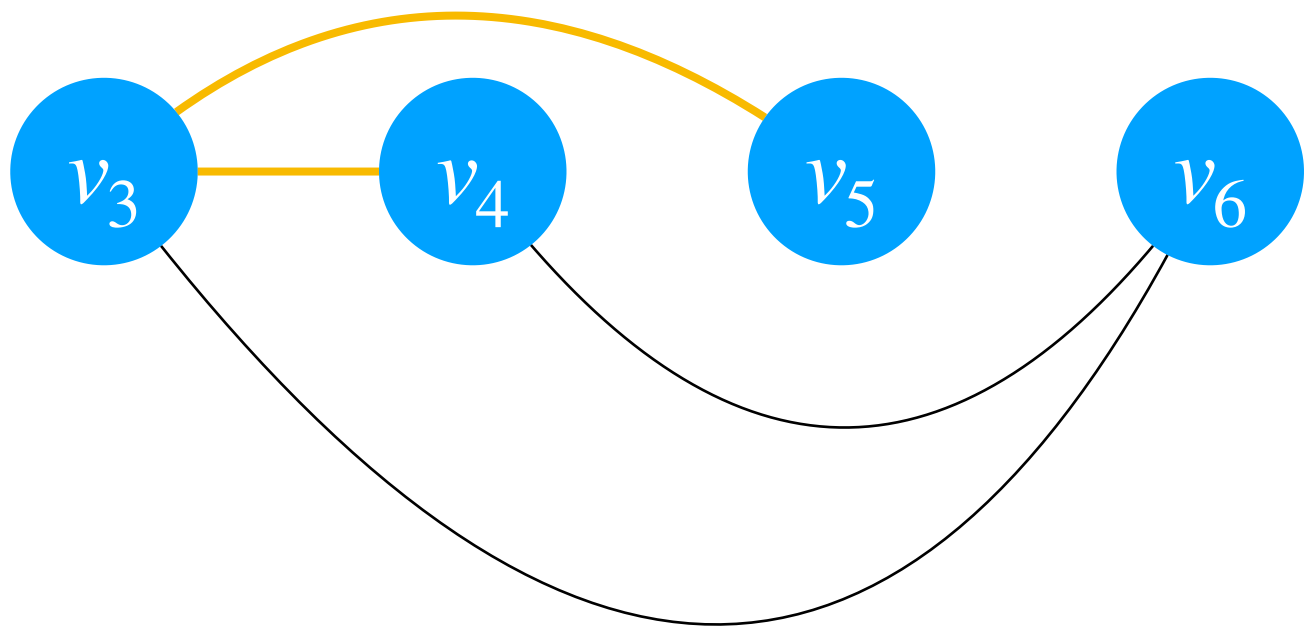
Elimination Orderings



Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

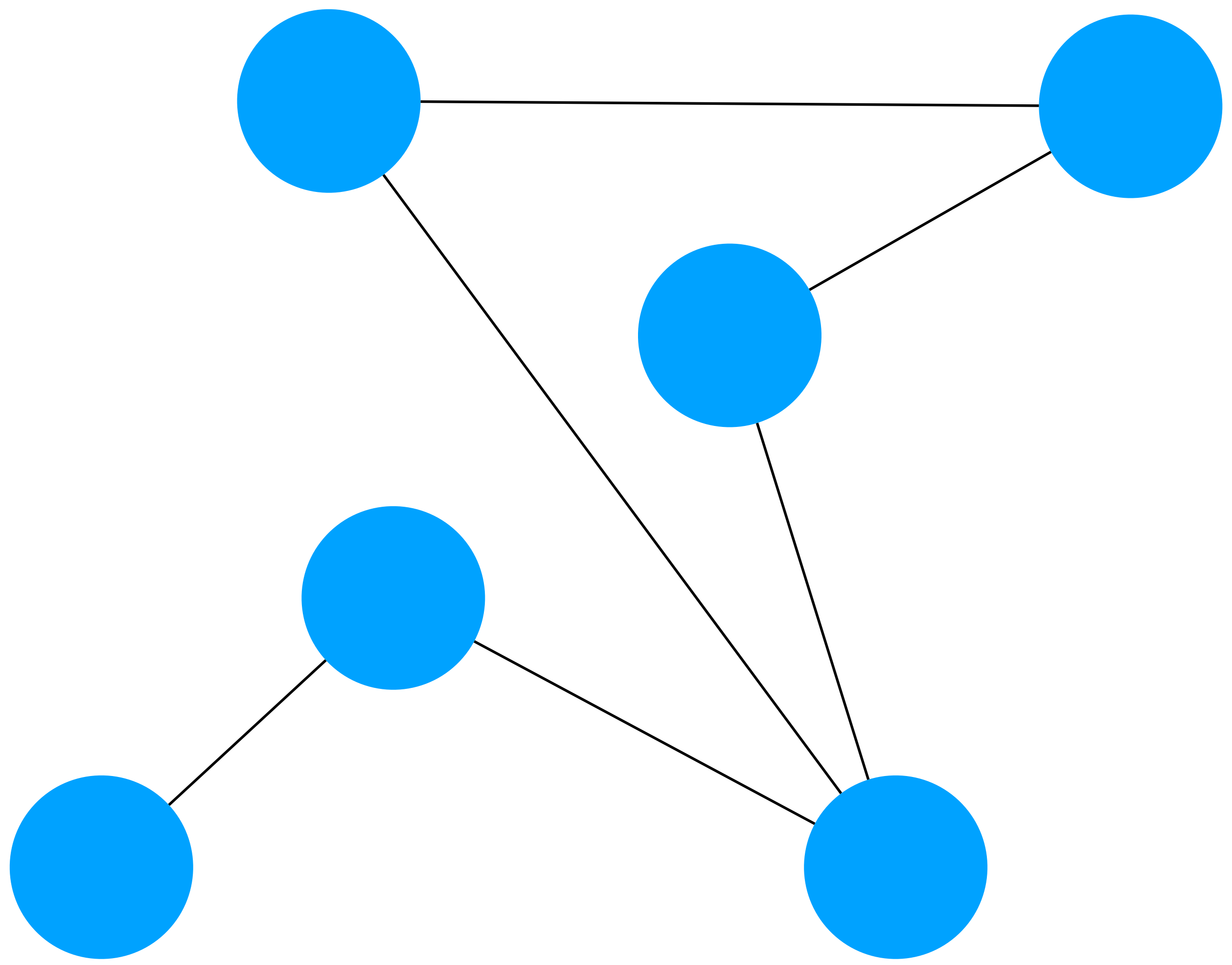
Upon eliminating v_i create edges between its neighbors.



The **width** of σ is the maximum degree of a vertex upon elimination.

2 2

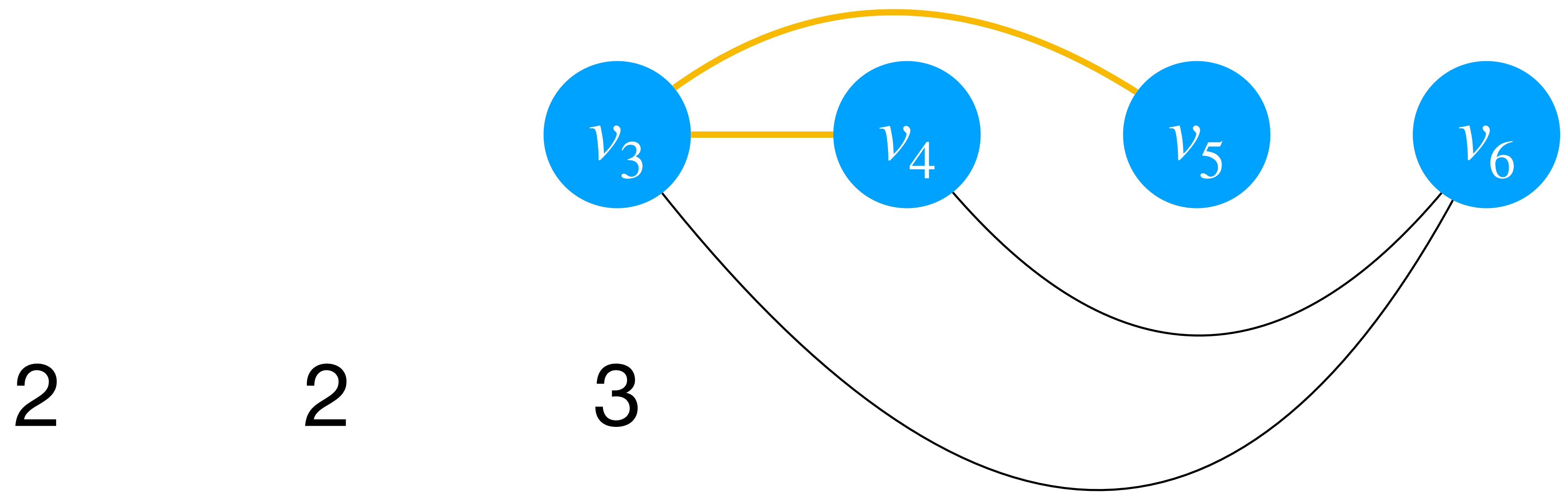
Elimination Orderings



Definition

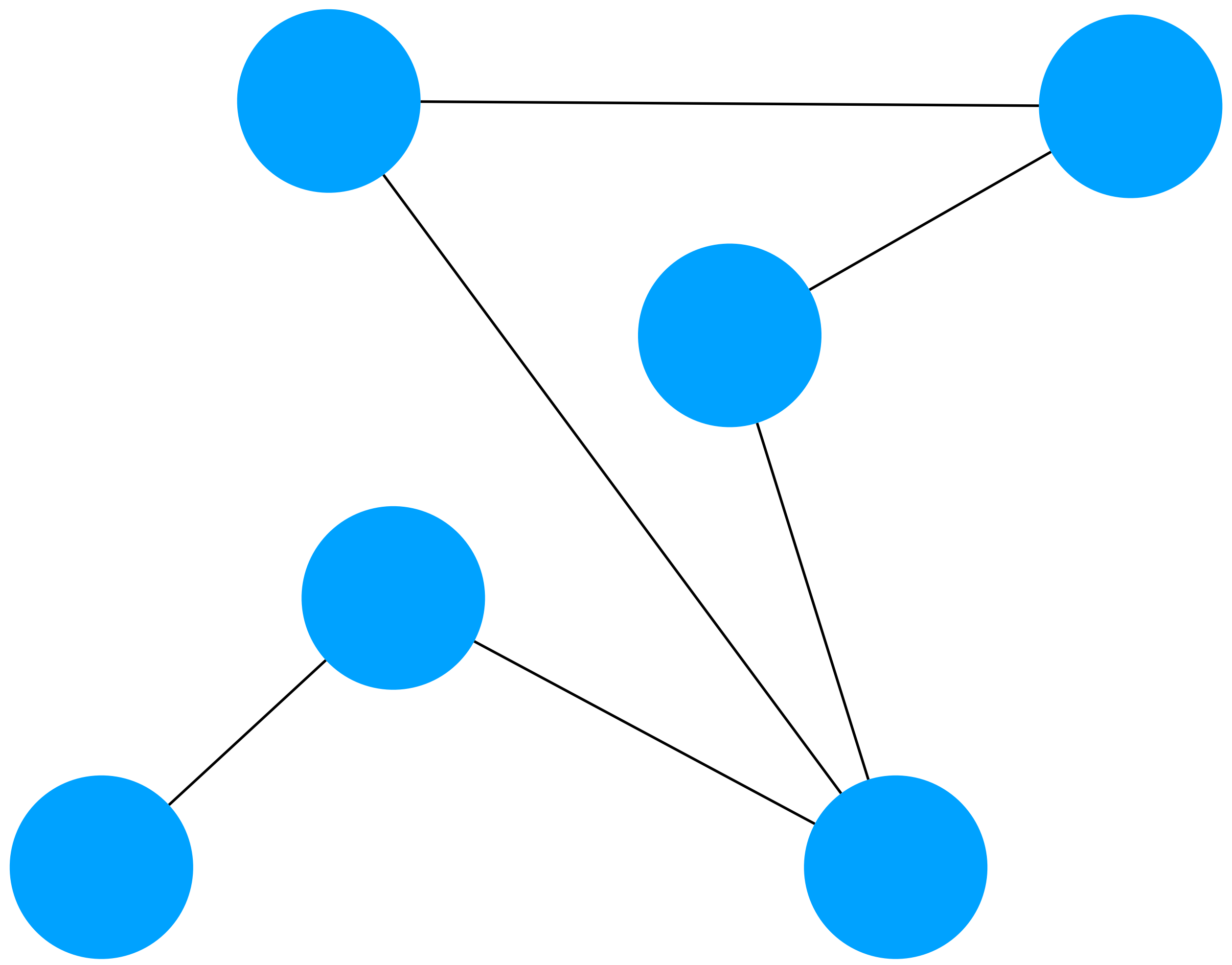
Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.



The **width** of σ is the maximum degree of a vertex upon elimination.

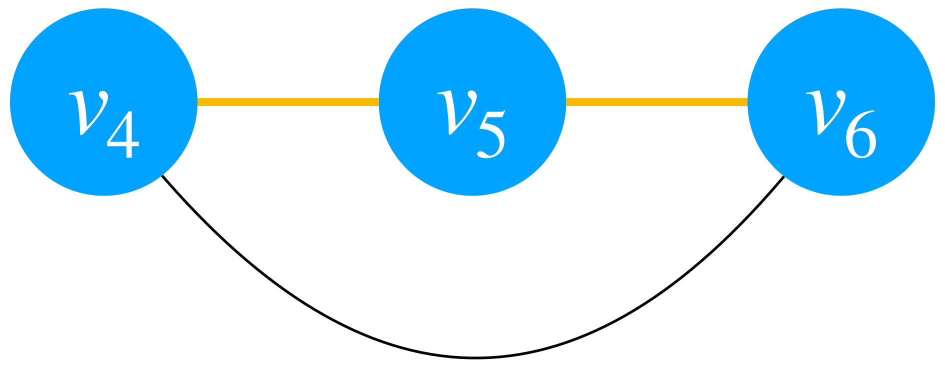
Elimination Orderings



Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

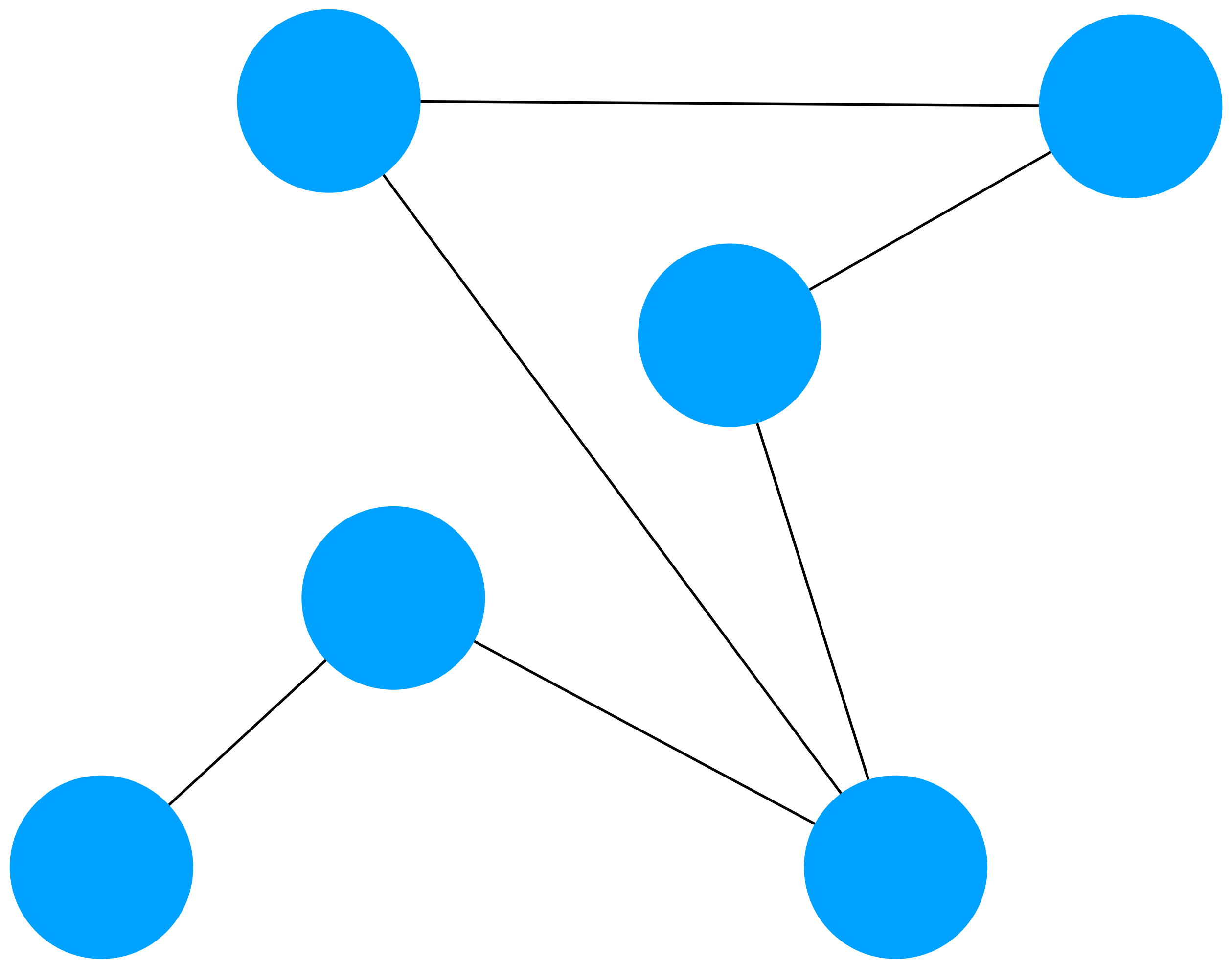
Upon eliminating v_i create edges between its neighbors.



The **width** of σ is the maximum degree of a vertex upon elimination.

2 2 3

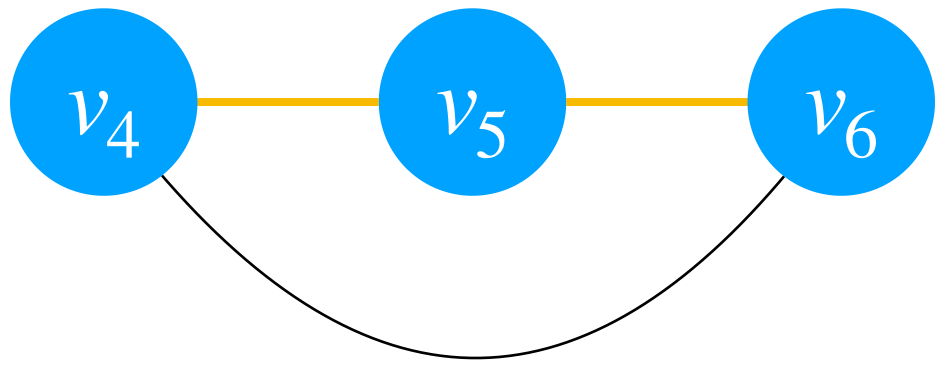
Elimination Orderings



Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

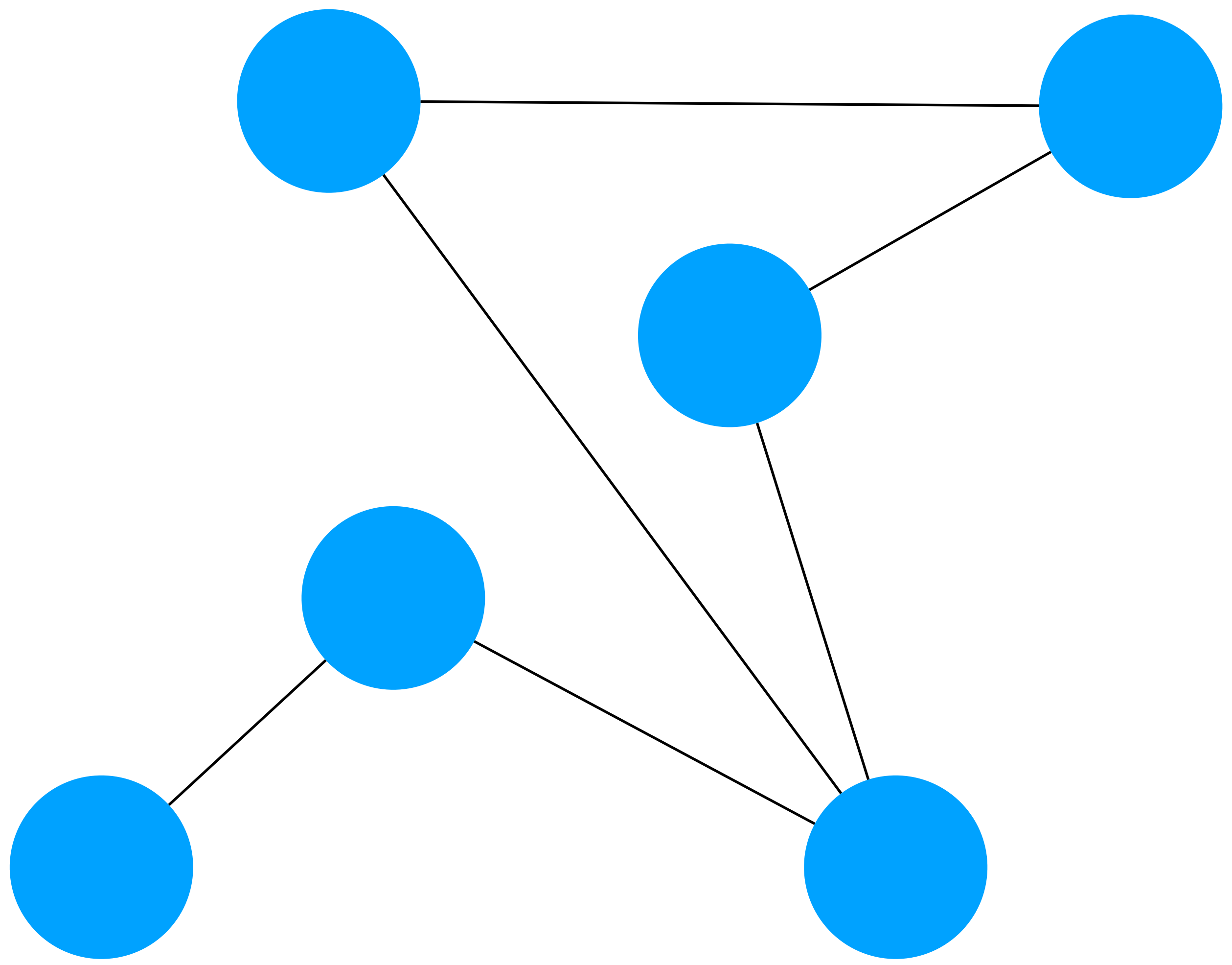
Upon eliminating v_i create edges between its neighbors.



The **width** of σ is the maximum degree of a vertex upon elimination.

2 2 3 2

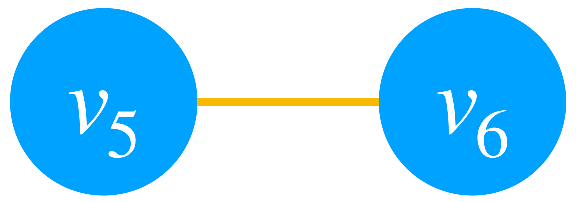
Elimination Orderings



Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

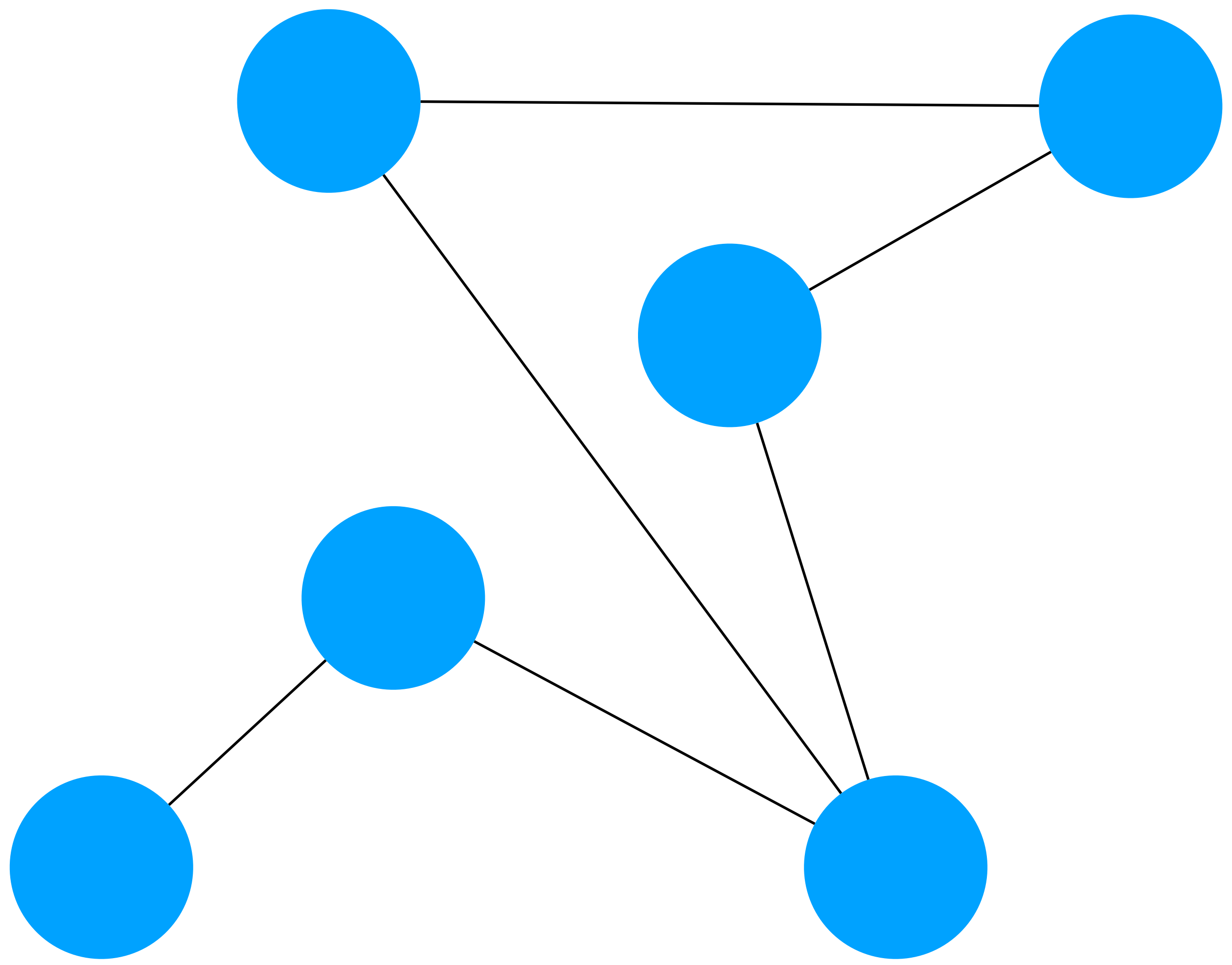
Upon eliminating v_i create edges between its neighbors.



The **width** of σ is the maximum degree of a vertex upon elimination.

2 2 3 2

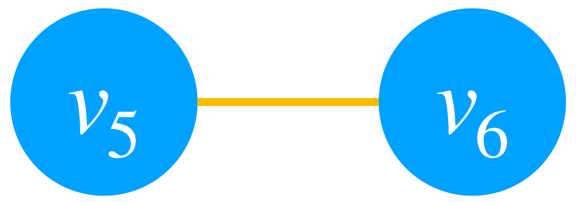
Elimination Orderings



Definition

Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

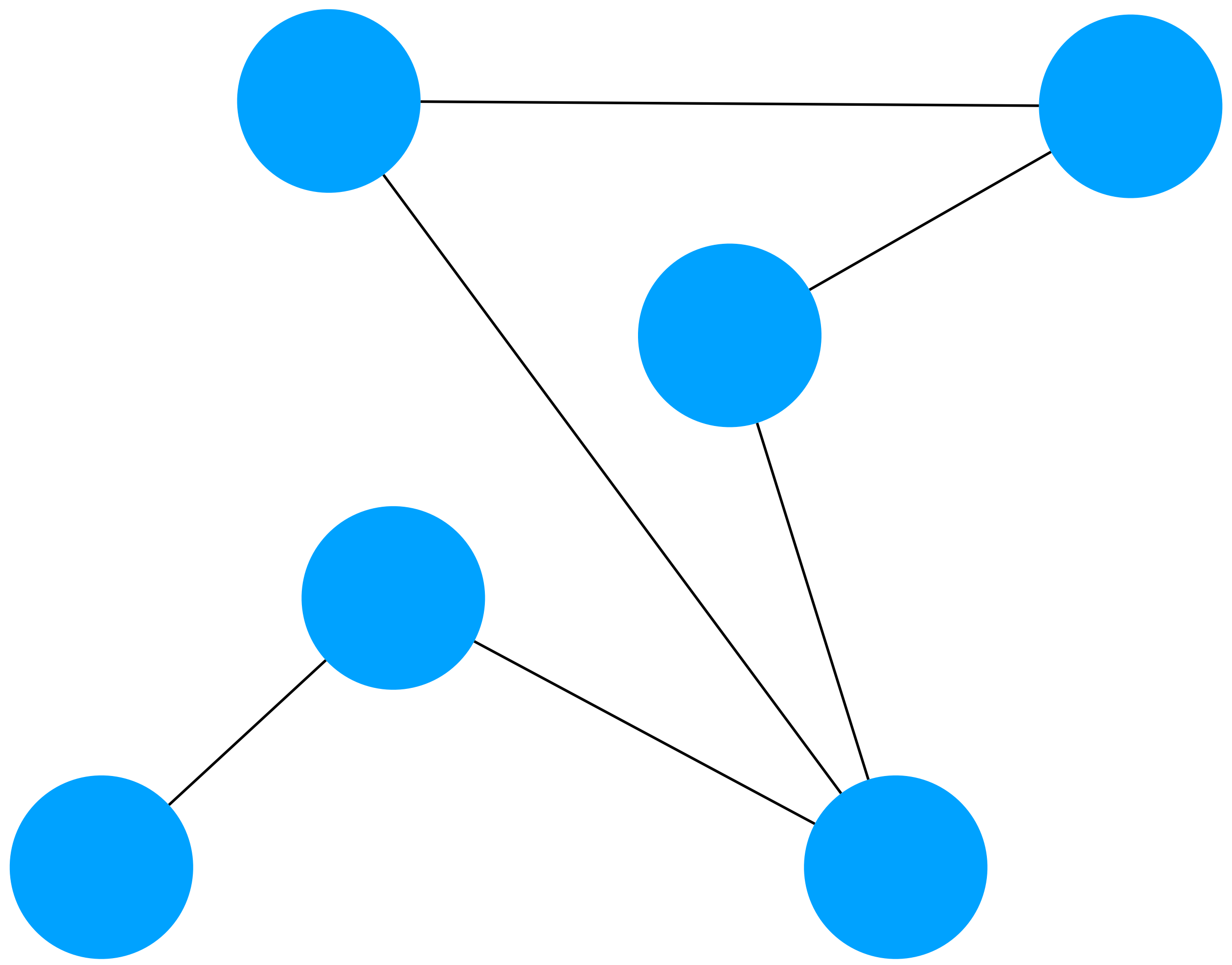
Upon eliminating v_i create edges between its neighbors.



The **width** of σ is the maximum degree of a vertex upon elimination.

2 2 3 2 1

Elimination Orderings



Definition

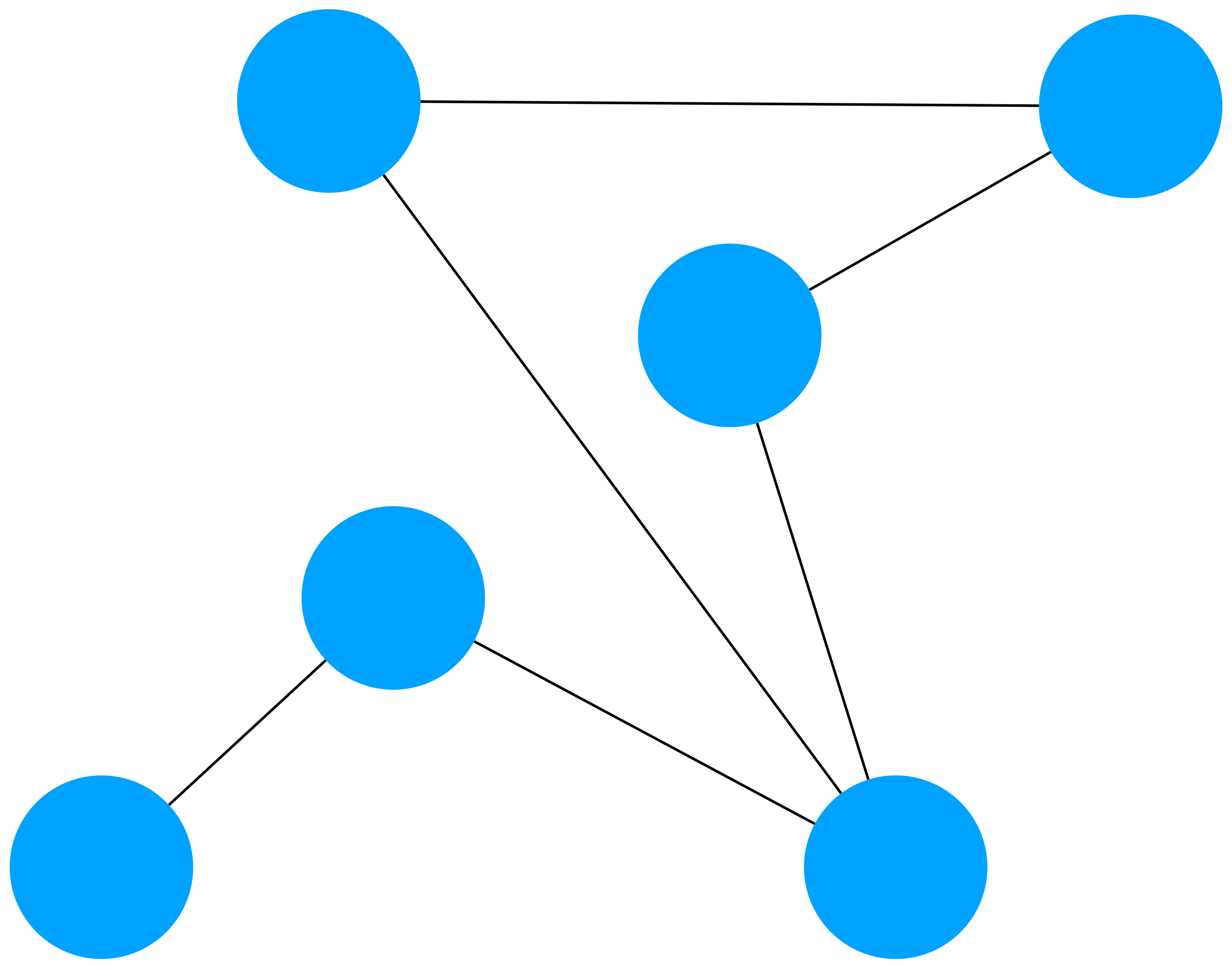
Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.

The **width** of σ is the maximum degree of a vertex upon elimination.

2 2 3 2 1

Elimination Ordorings



Definition

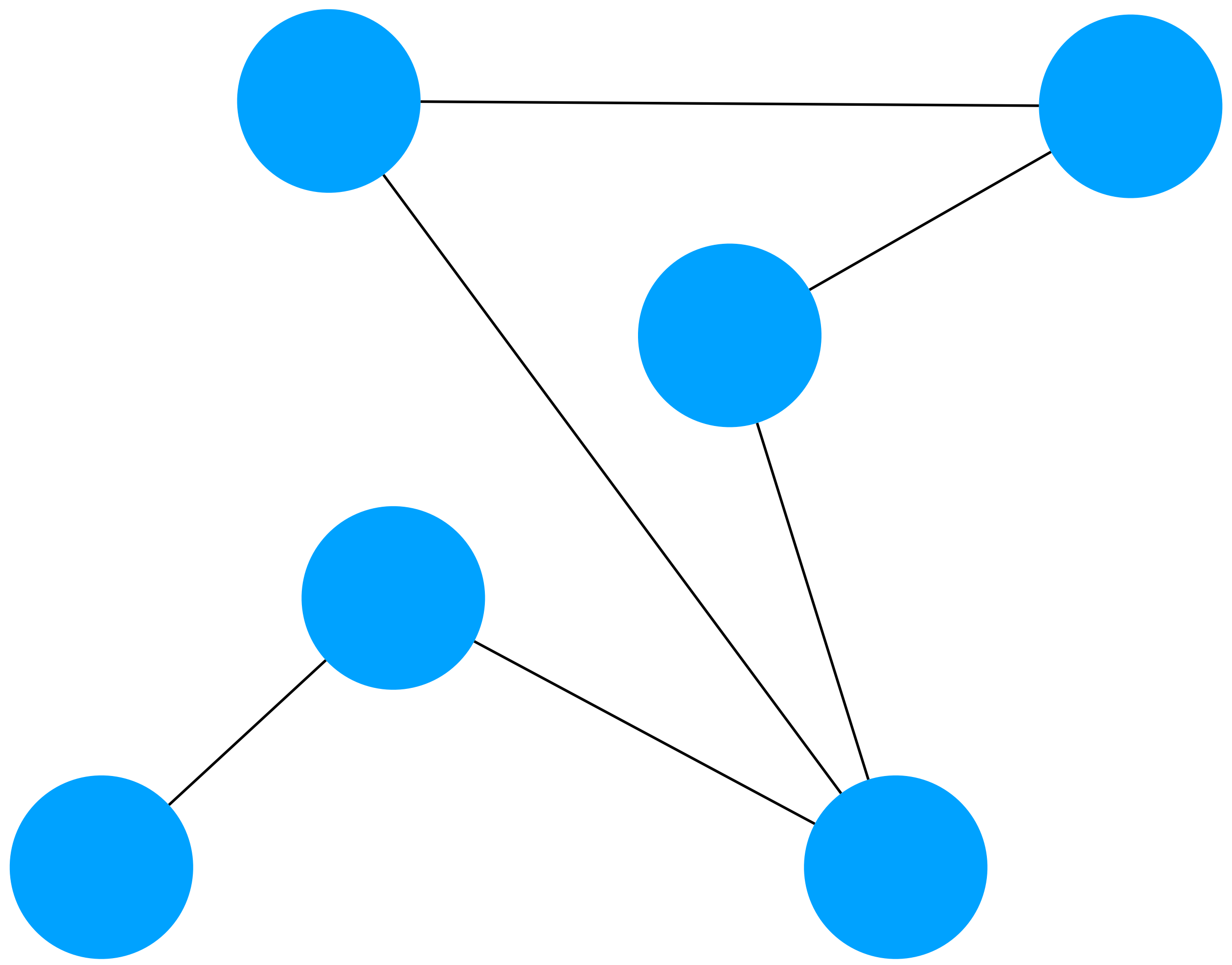
Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.

The **width** of σ is the maximum degree of a vertex upon elimination.

2 2 3 2 1 0

Elimination Orderings



Definition

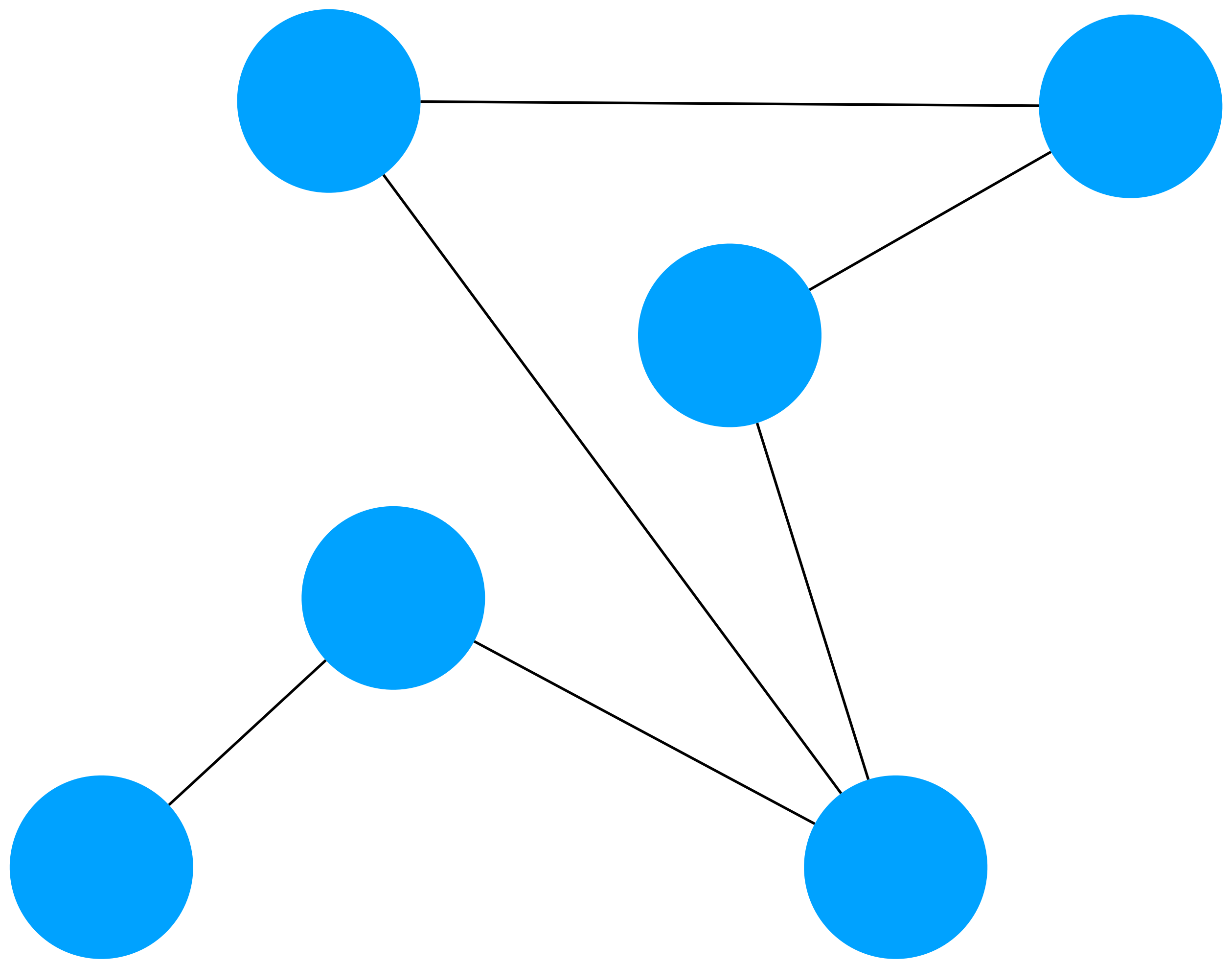
Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

Upon eliminating v_i create edges between its neighbors.

The **width** of σ is the maximum degree of a vertex upon elimination.

2 2 3 2 1 0

Elimination Orderings



Definition

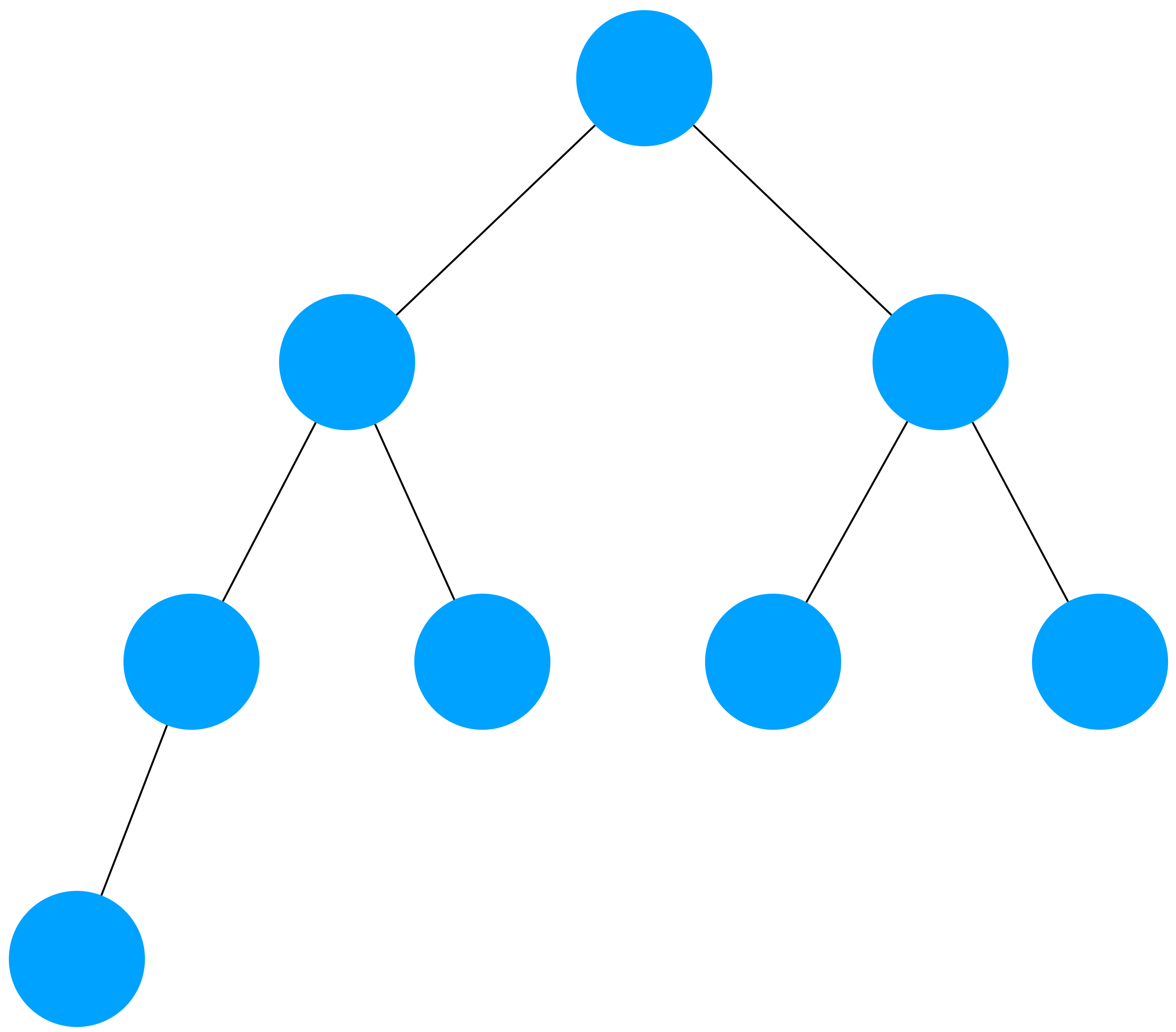
Let $G = (V, E)$ be a graph. An **elimination ordering** of G is simply an ordering $\sigma = (v_1, \ldots, v_n)$ of V.

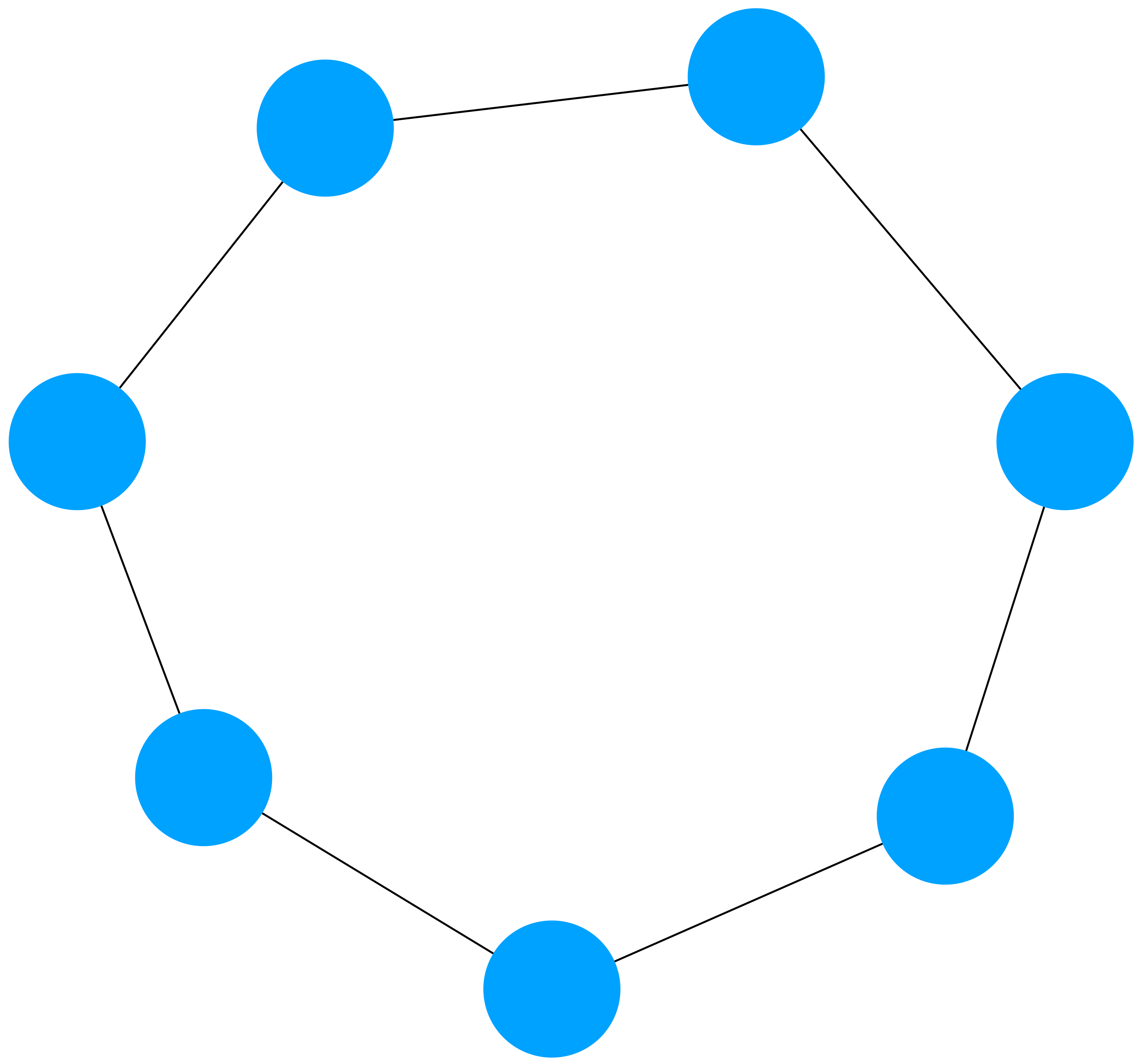
Upon eliminating v_i create edges between its neighbors.

The **width** of σ is the maximum degree of a vertex upon elimination.

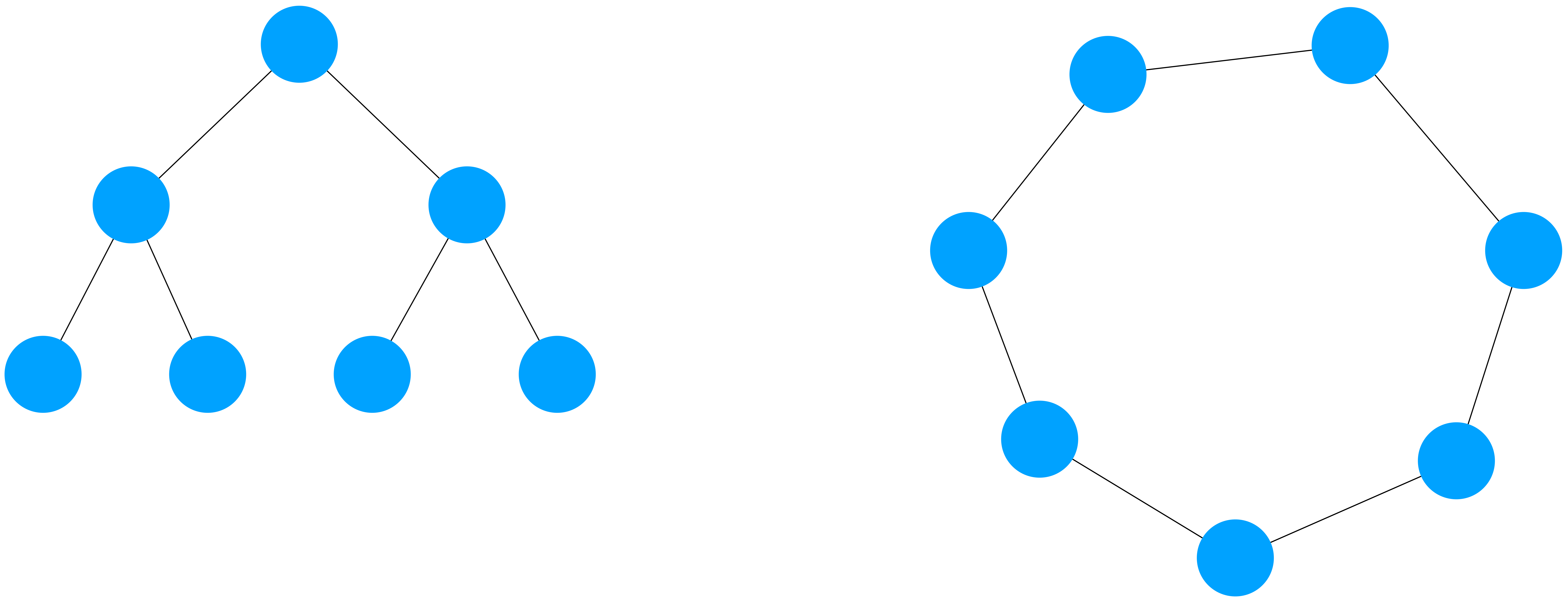
2 2 3 2 1 0

Examples

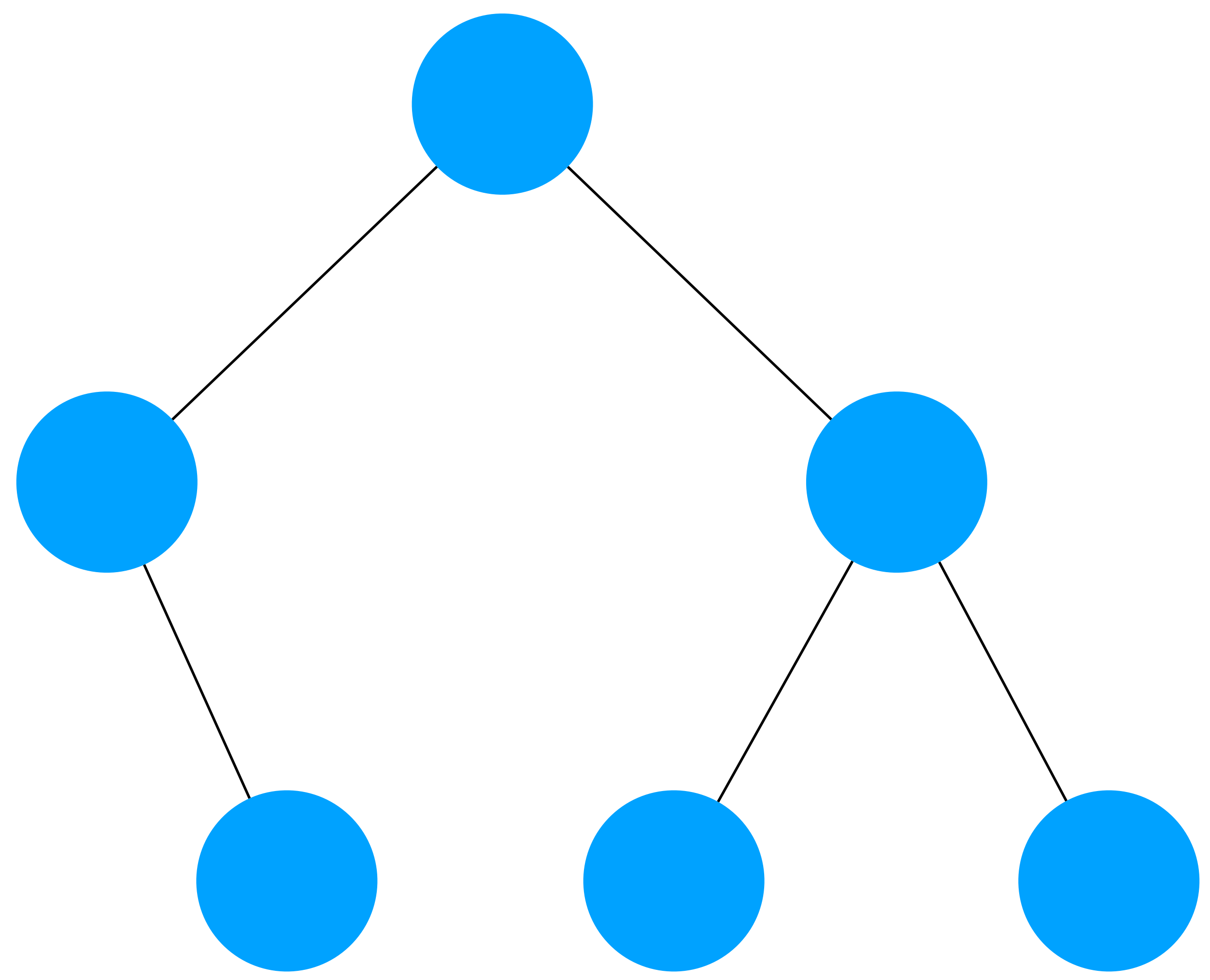


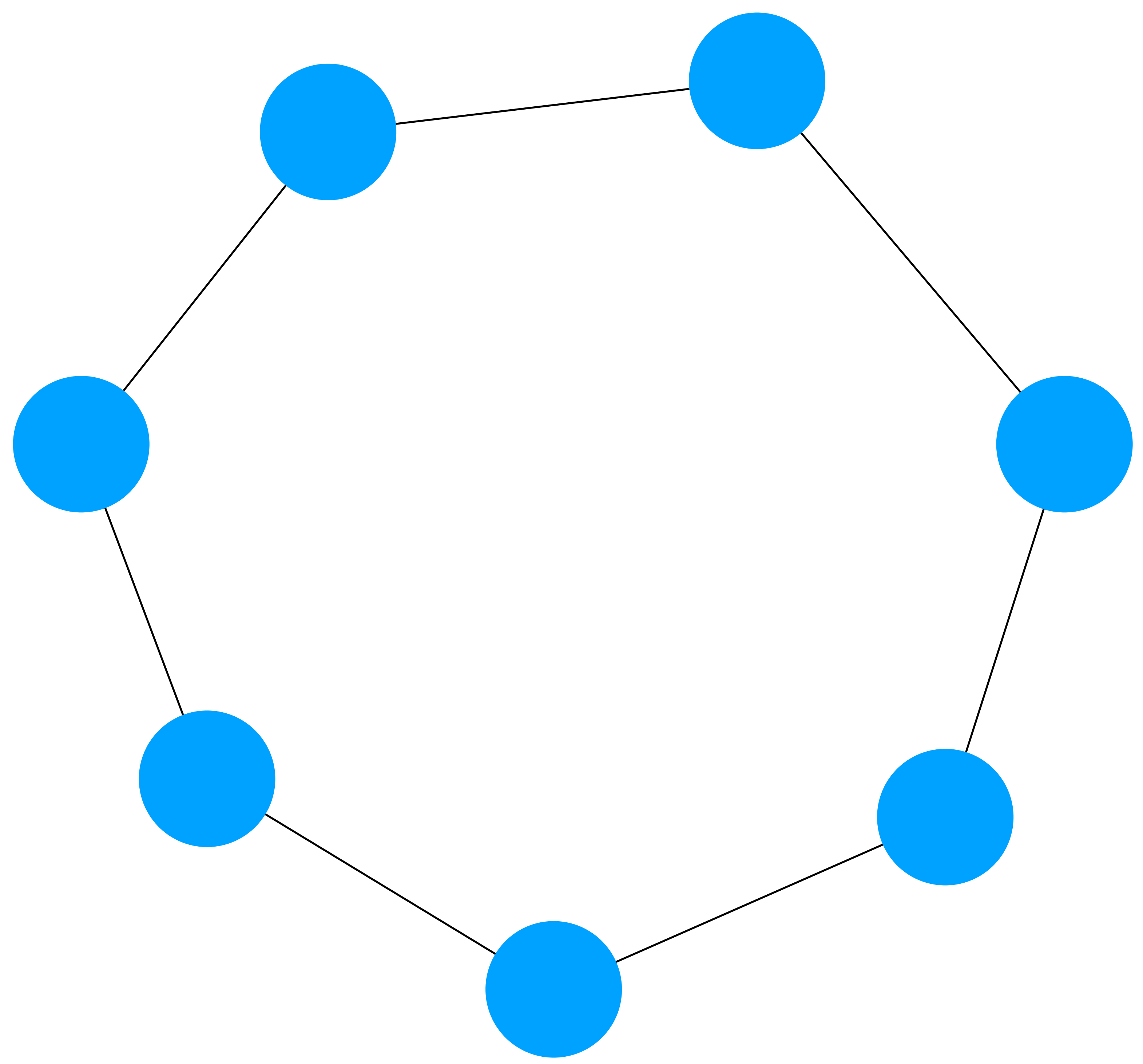


Examples

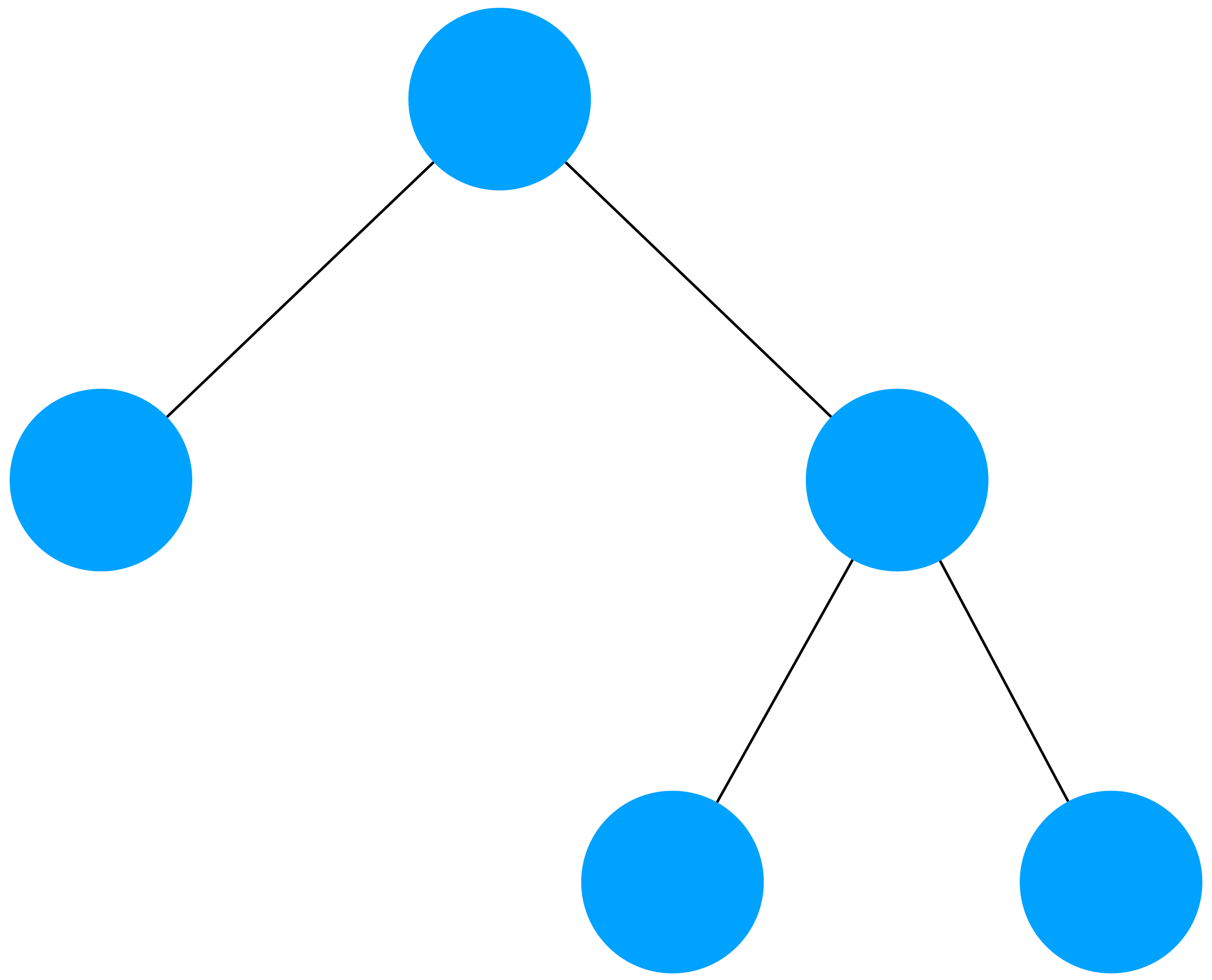


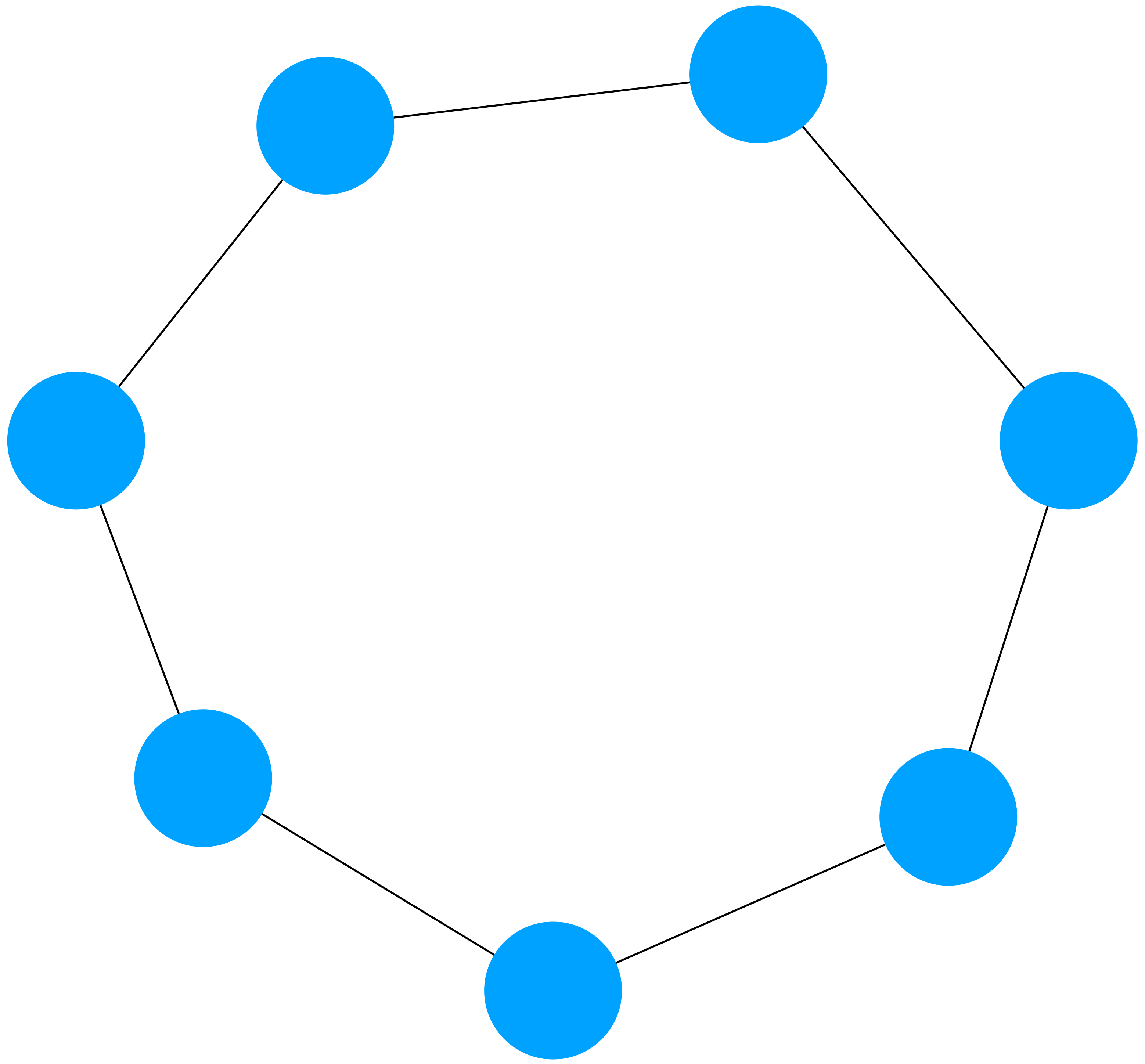
Examples



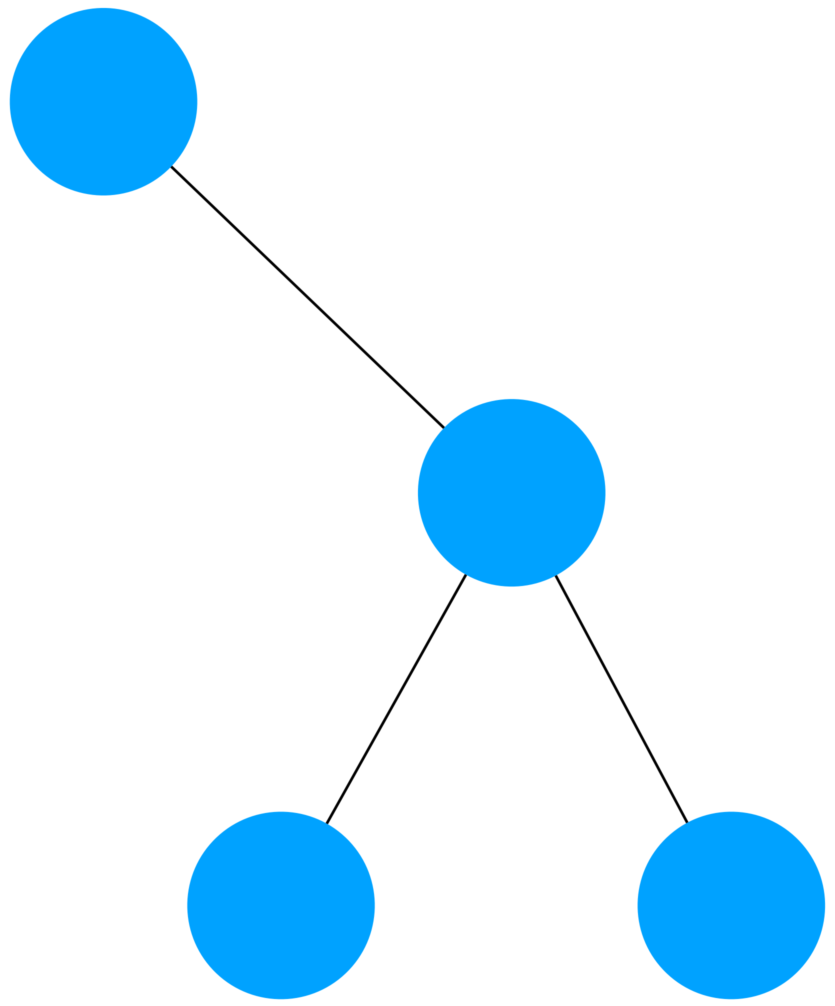


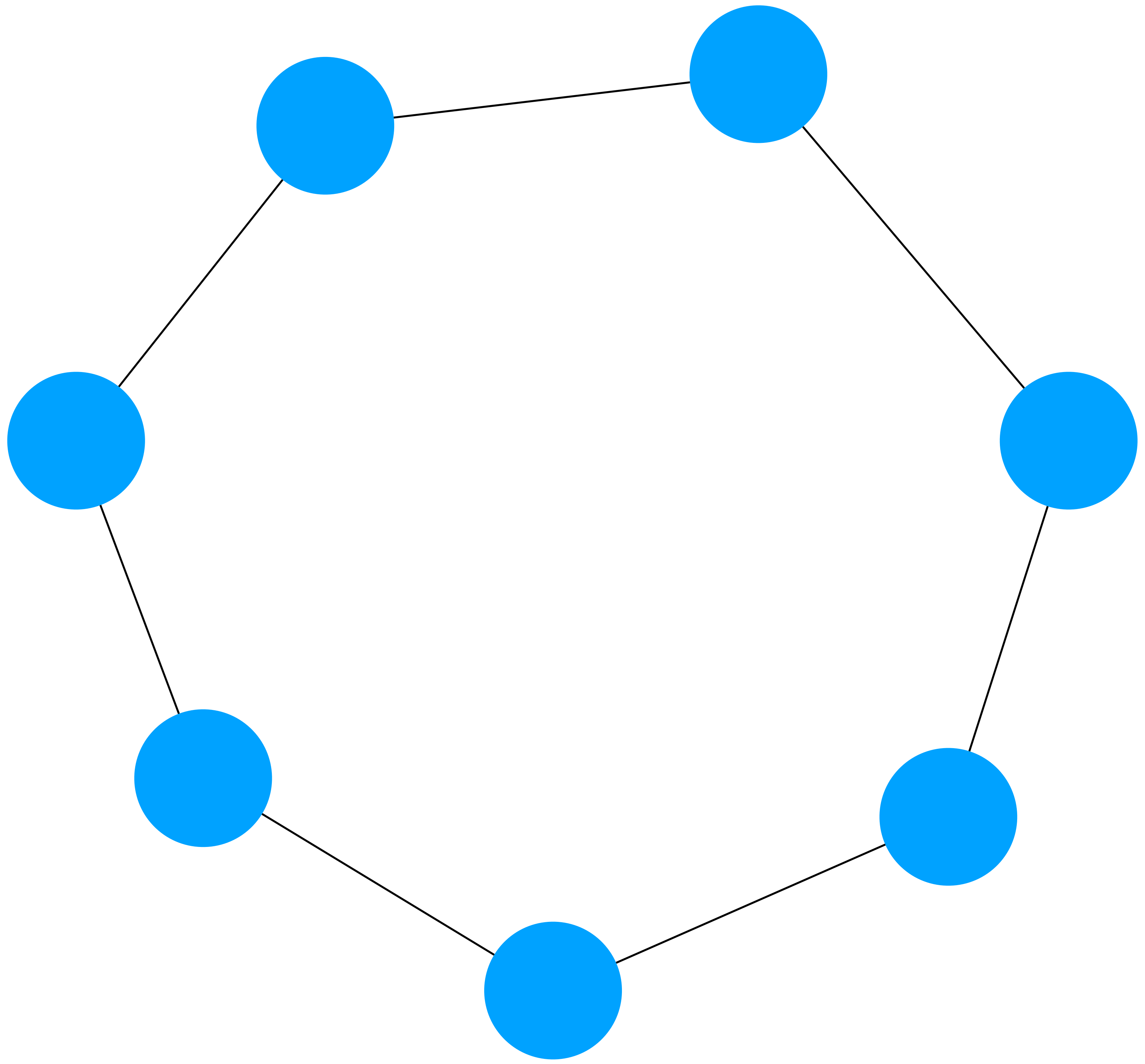
Examples



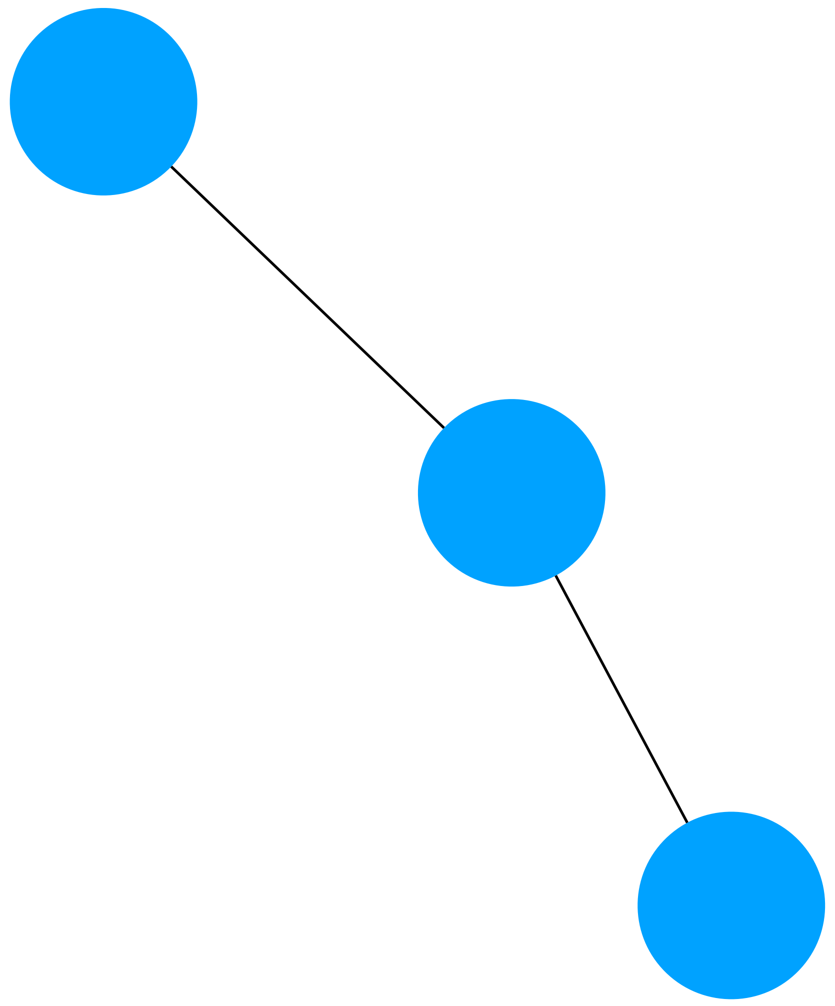


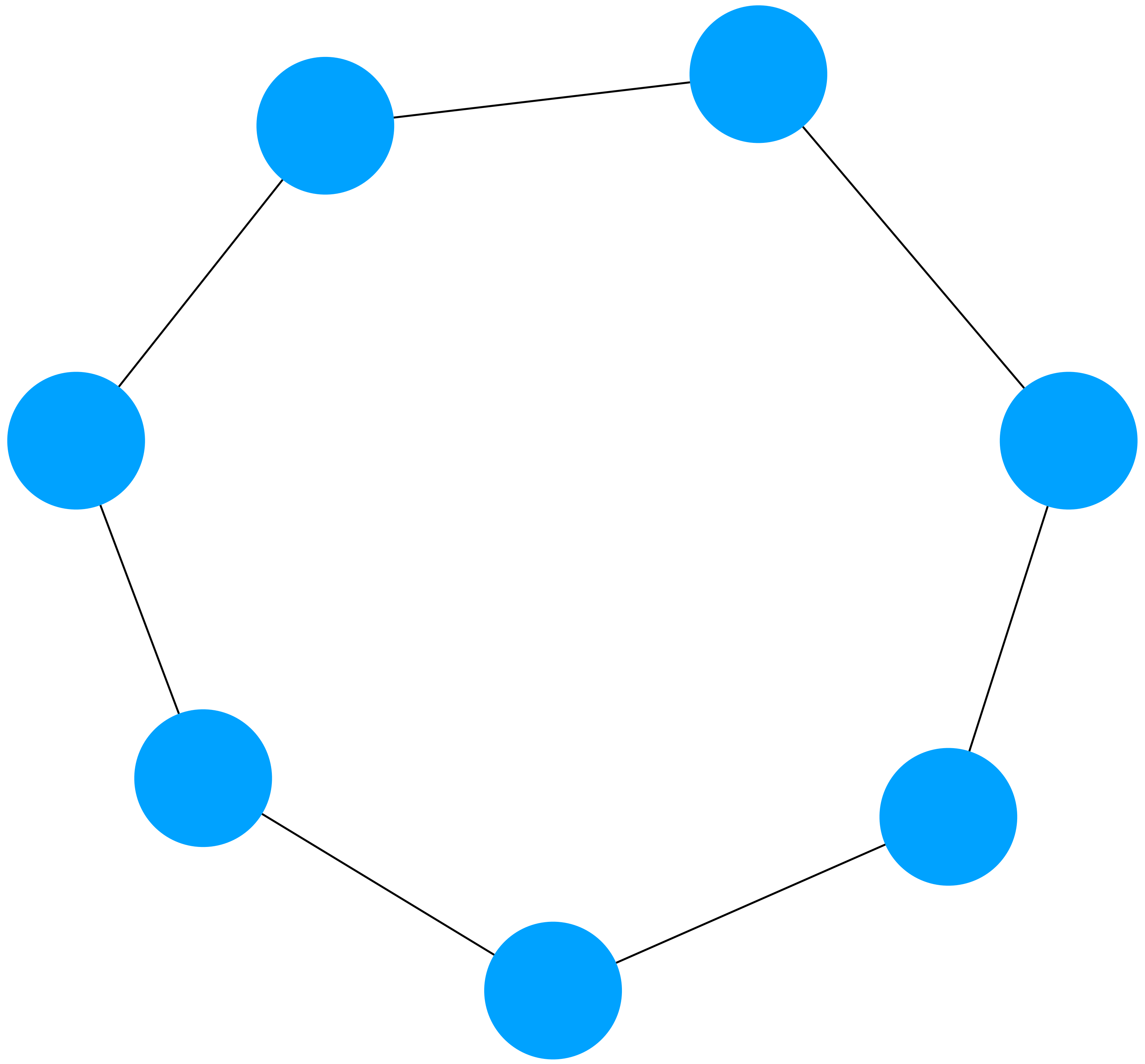
Examples



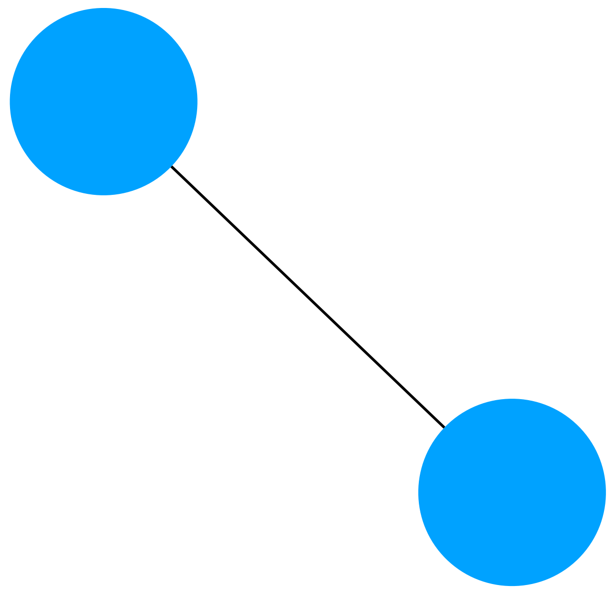


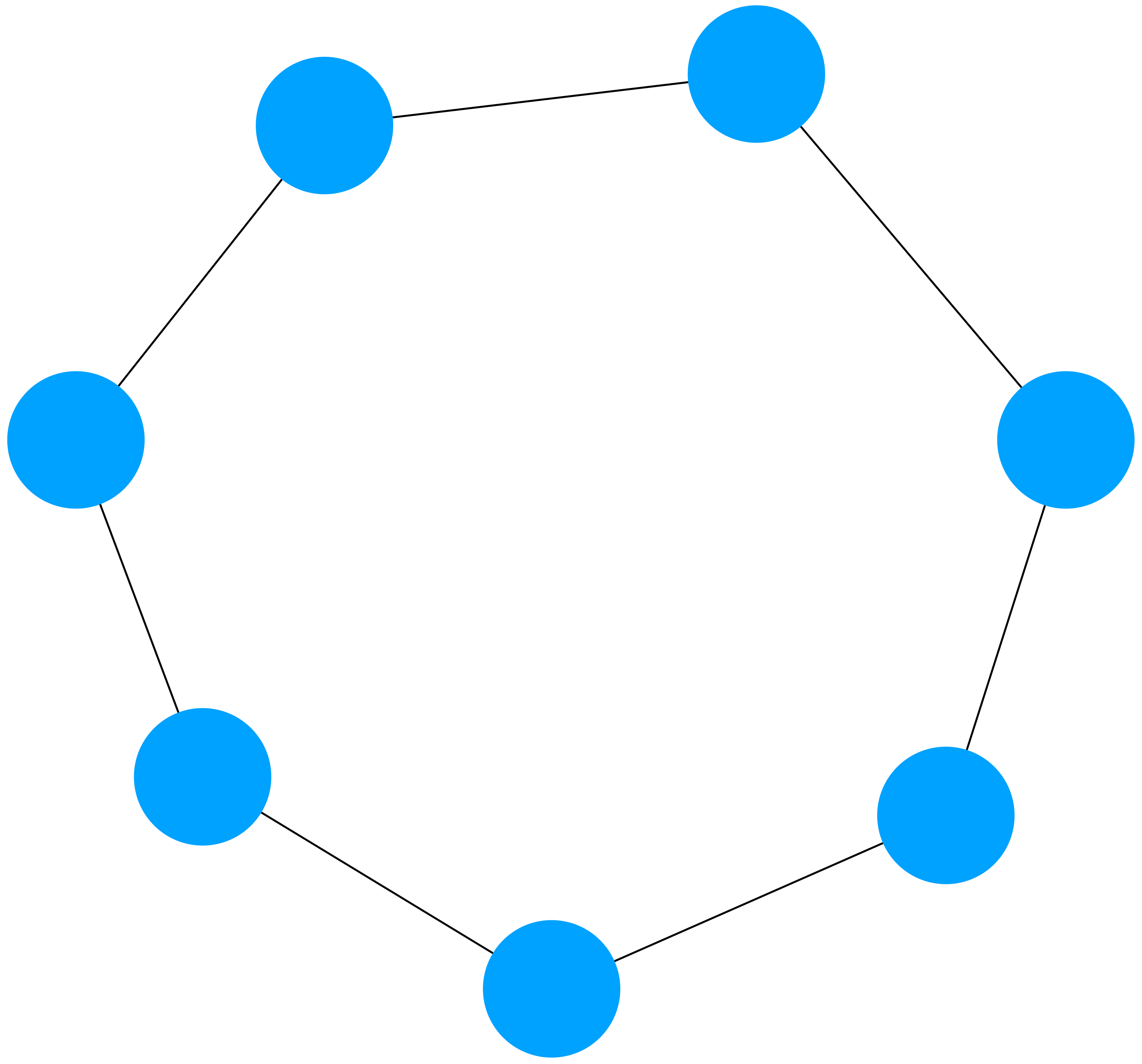
Examples



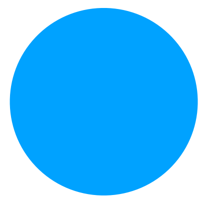


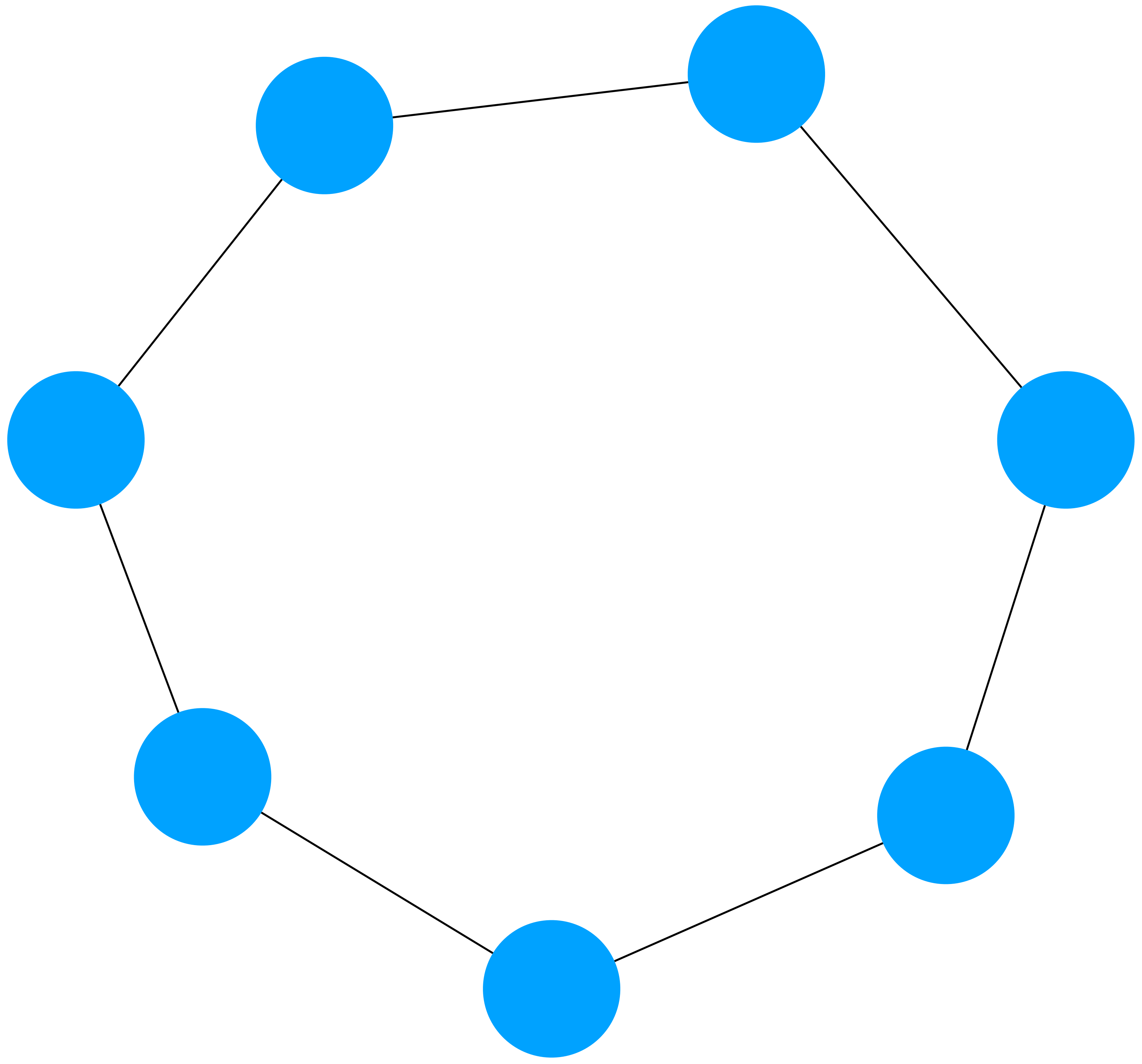
Examples



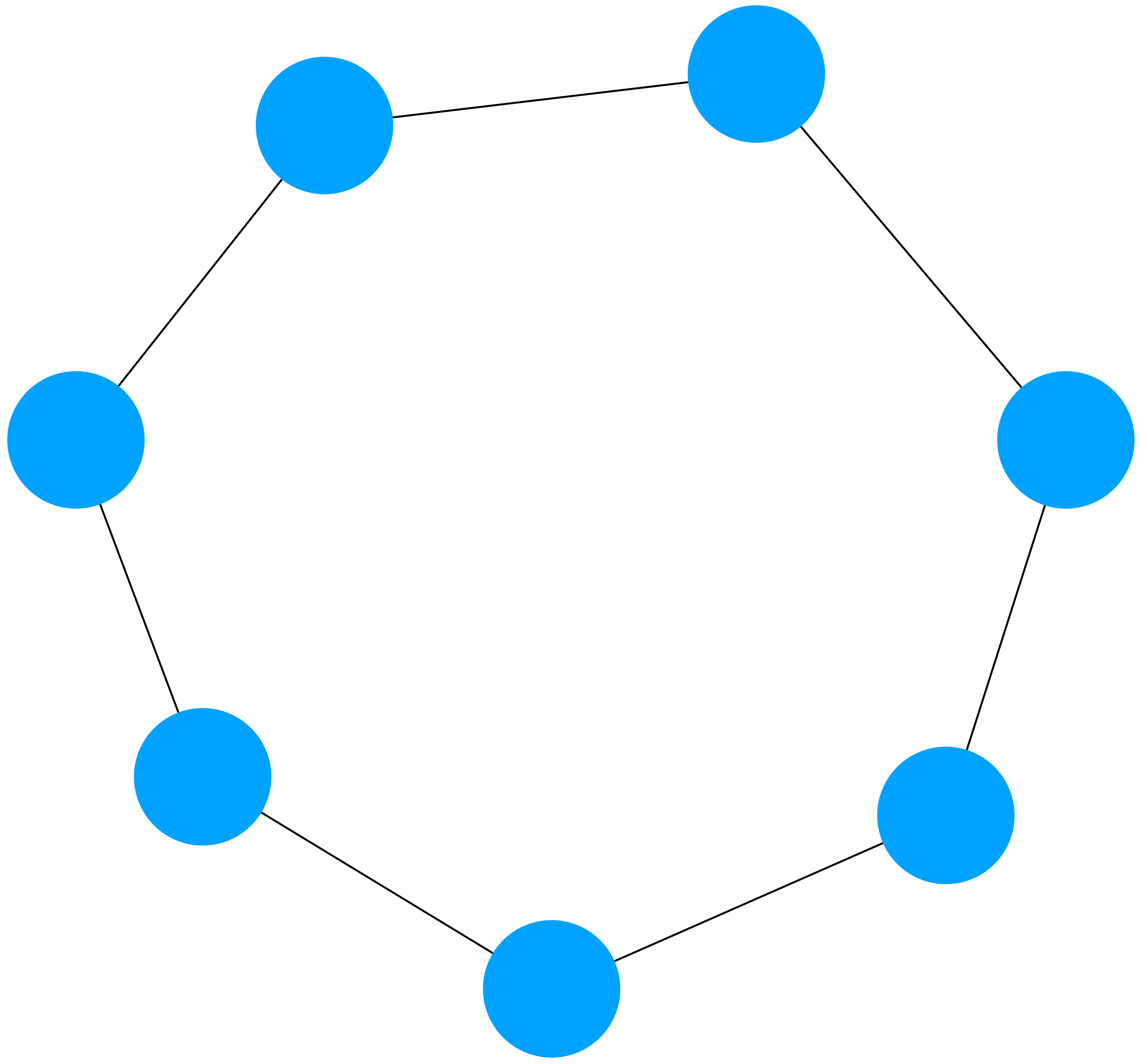


Examples

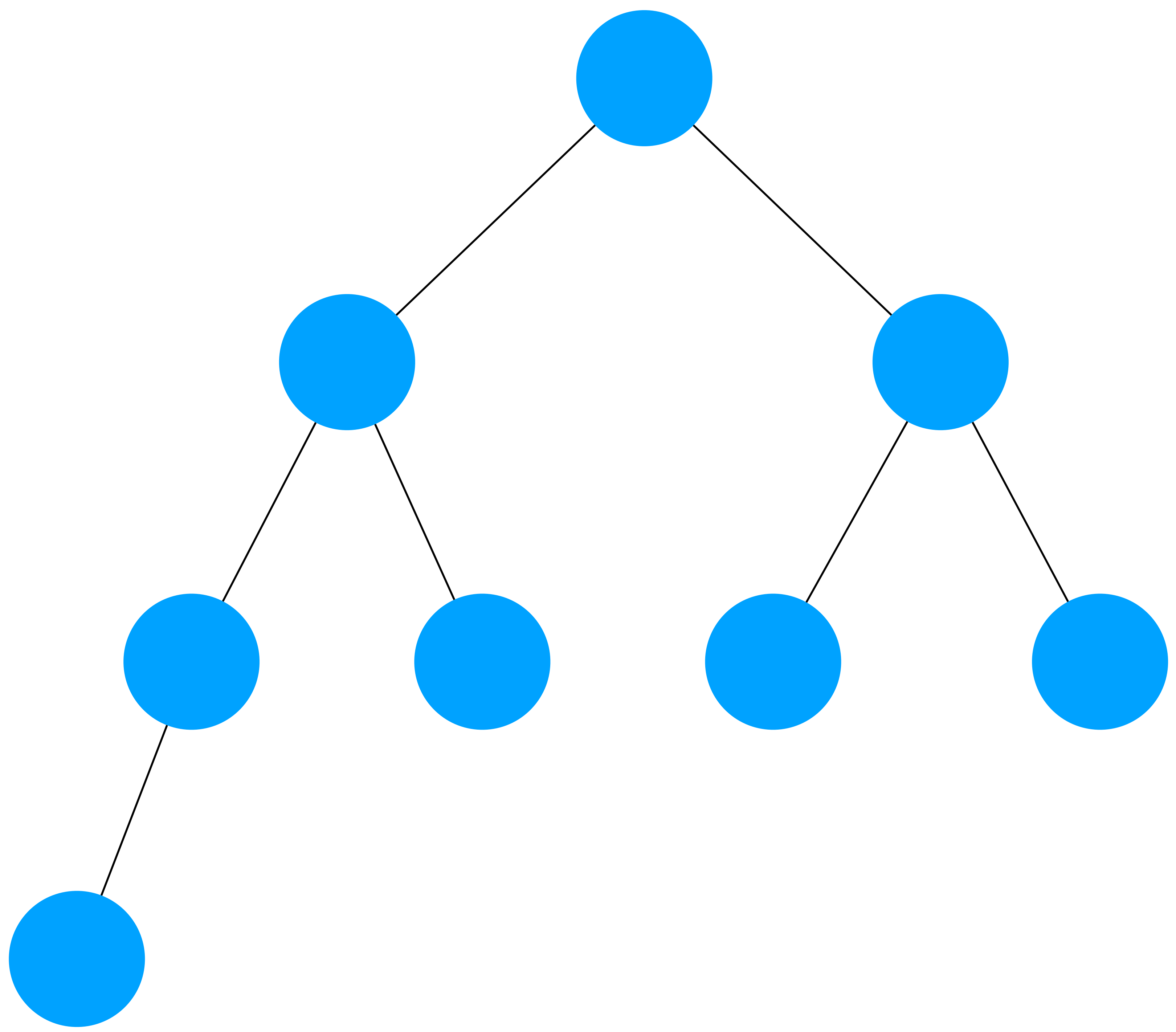




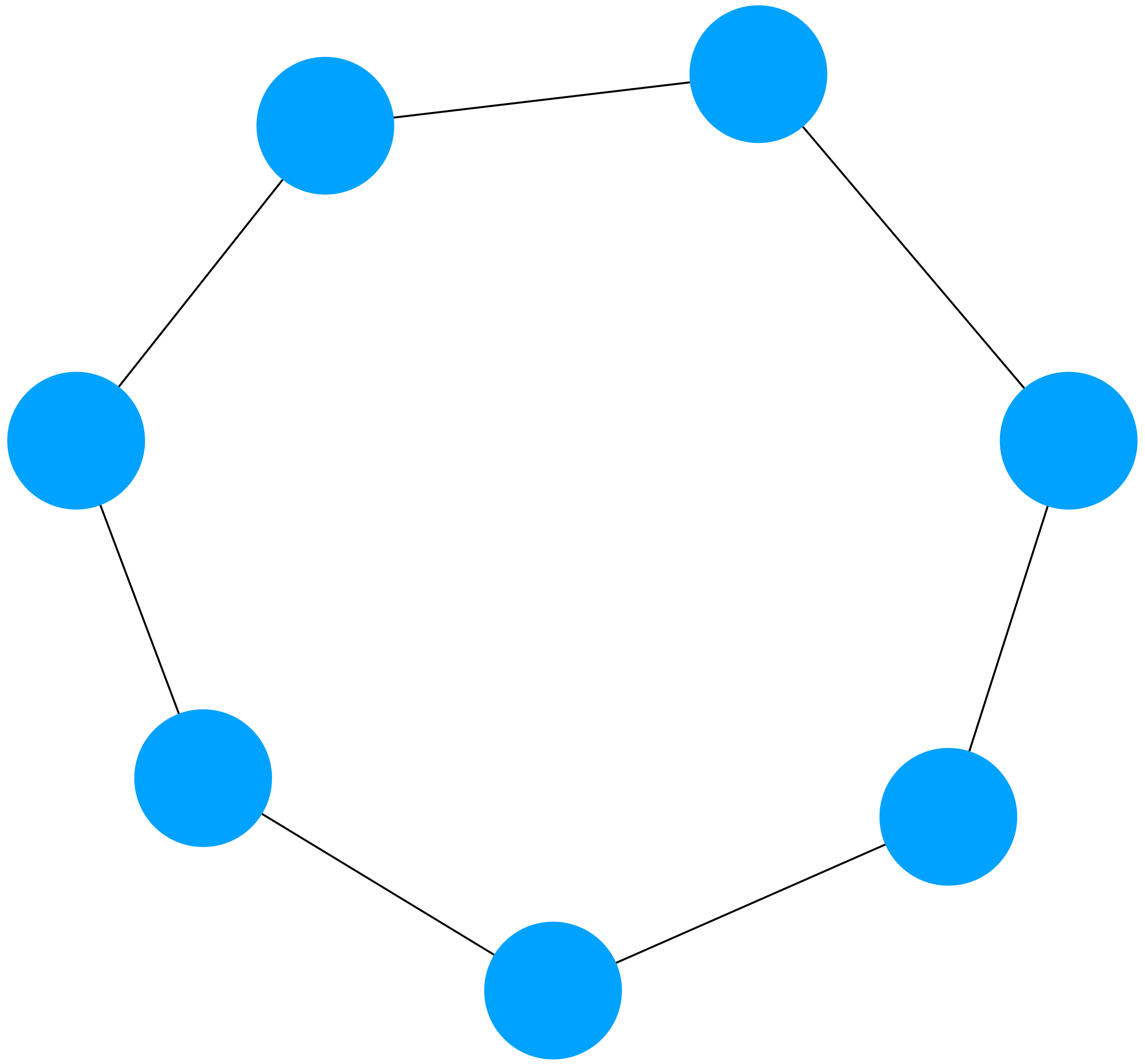
Examples



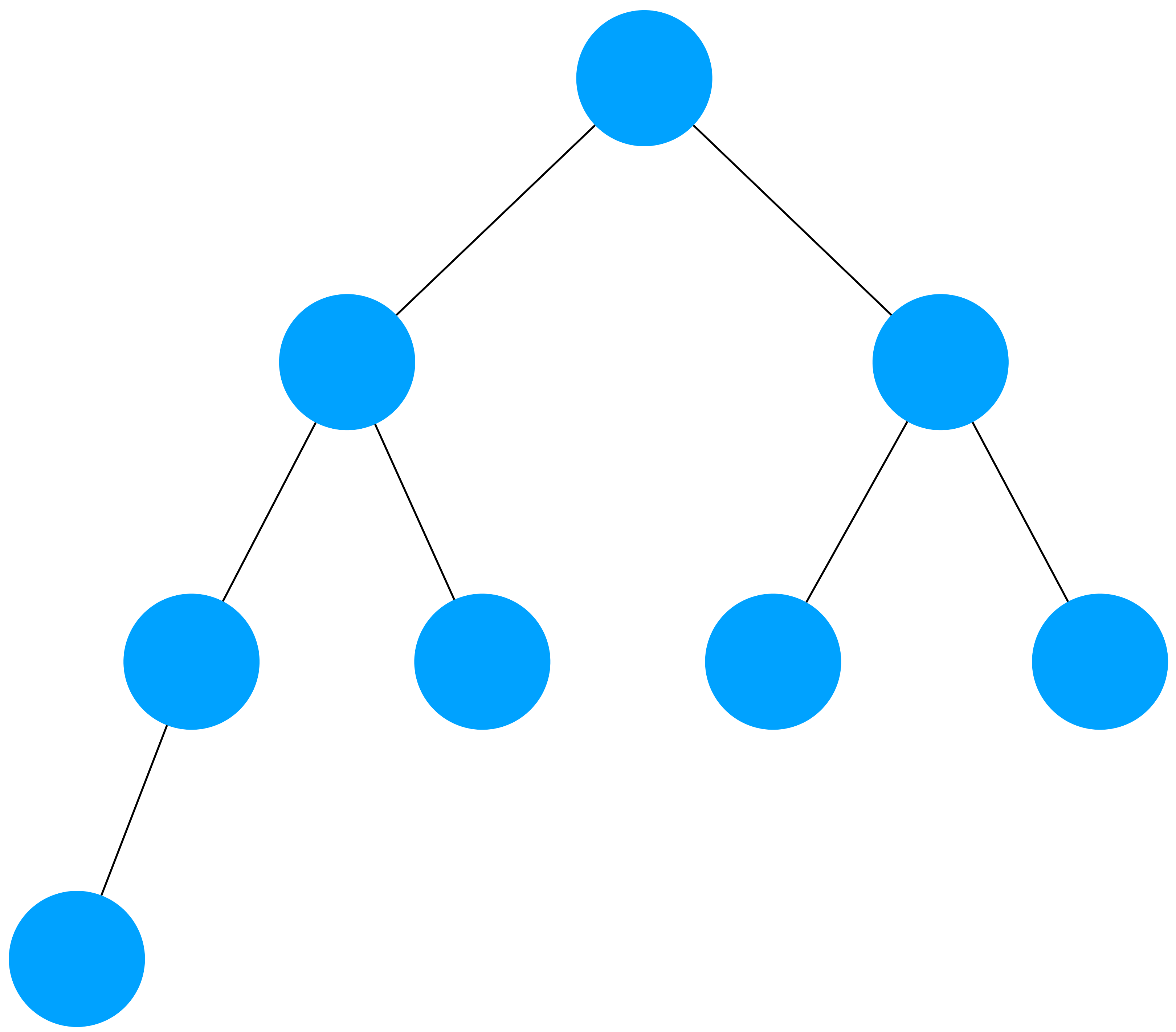
Examples



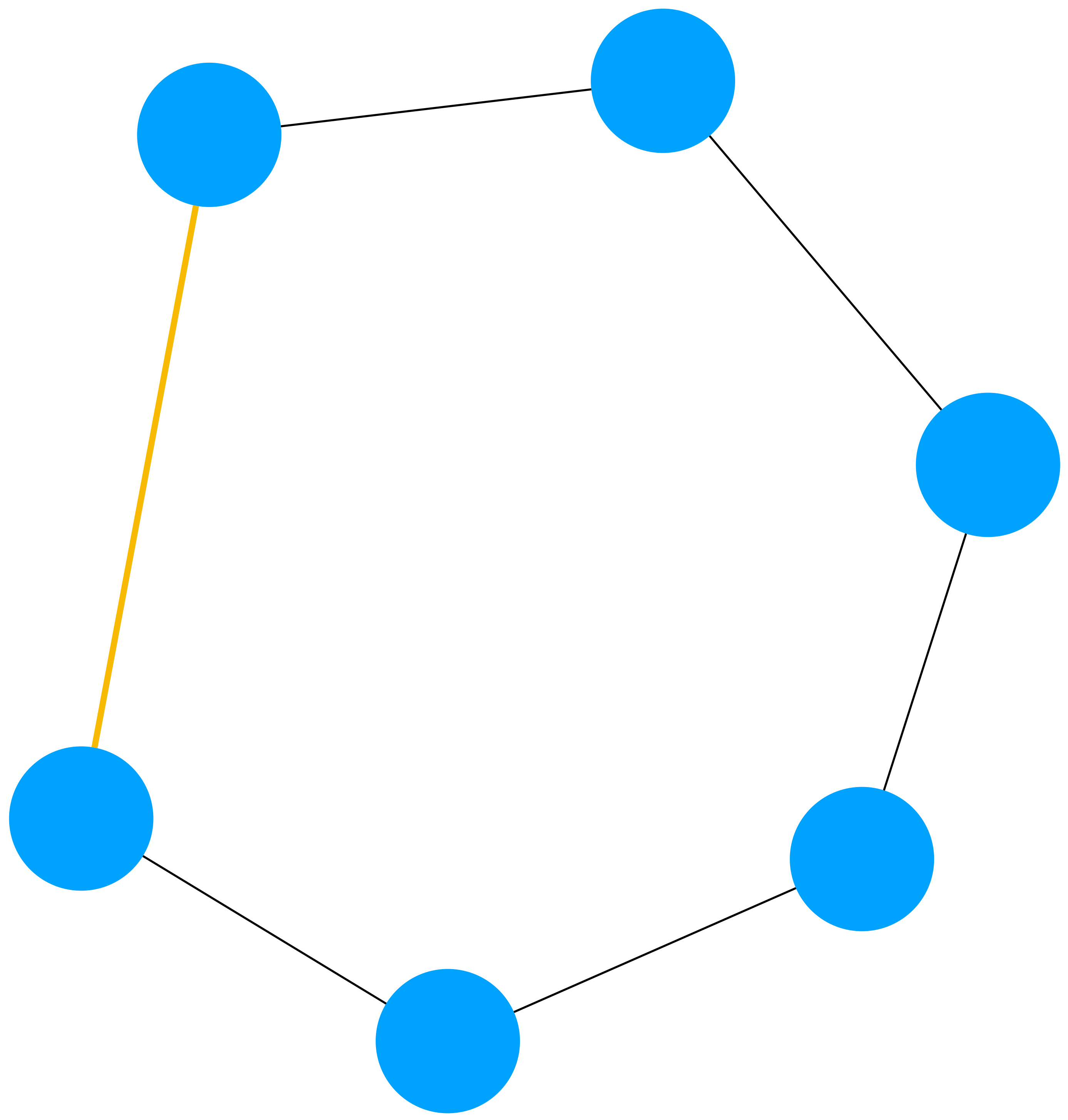
width 1



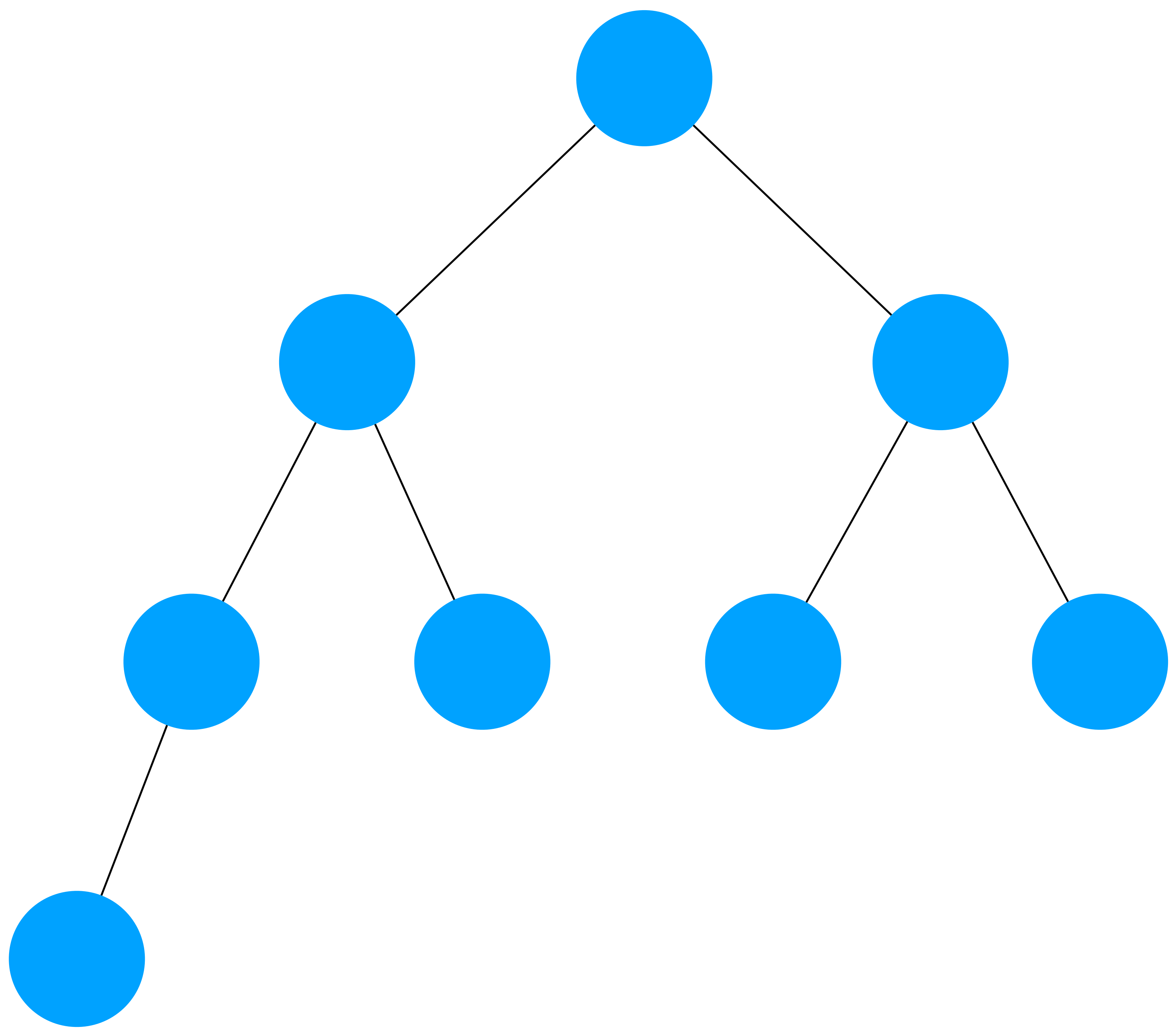
Examples



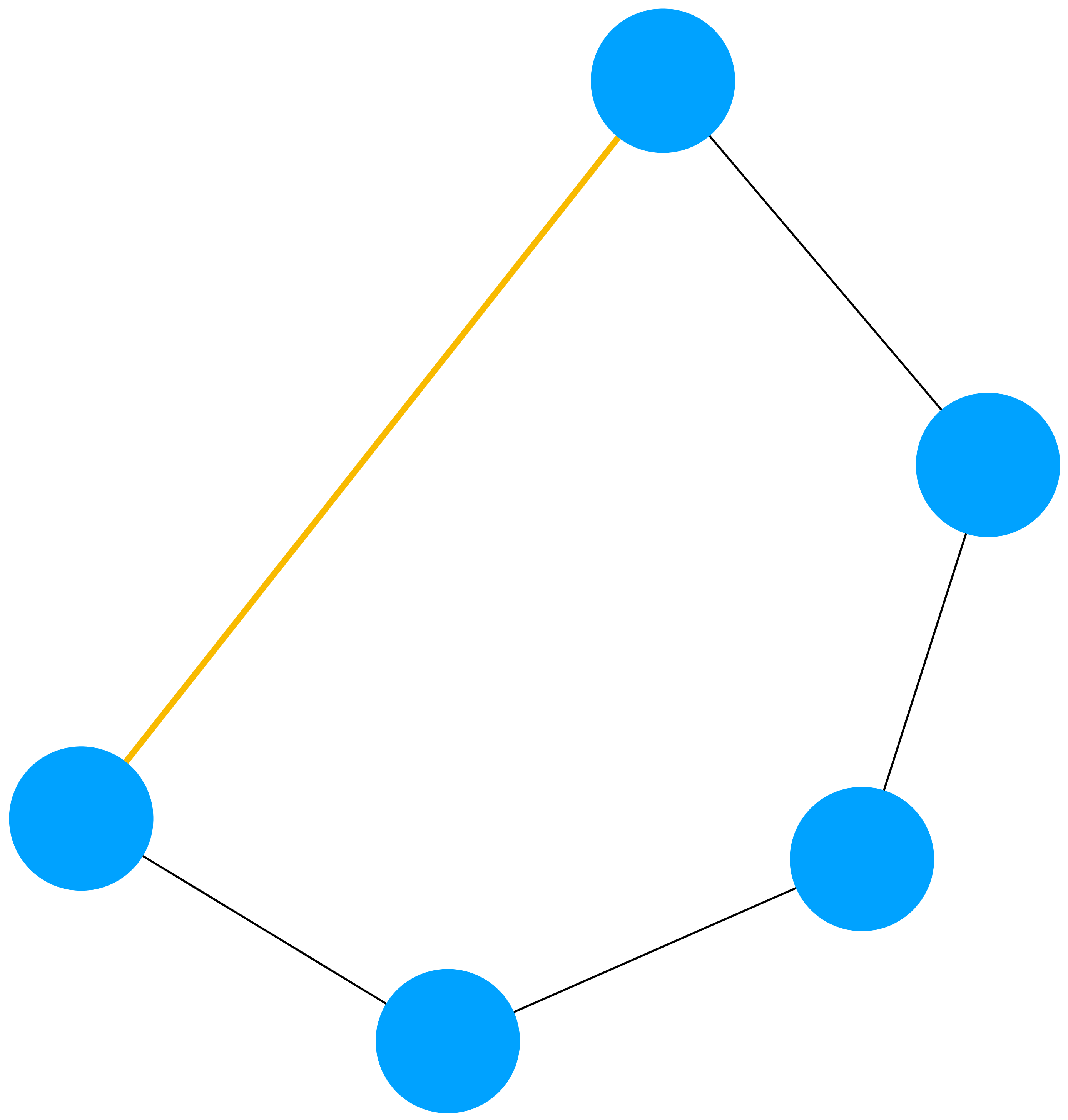
width 1



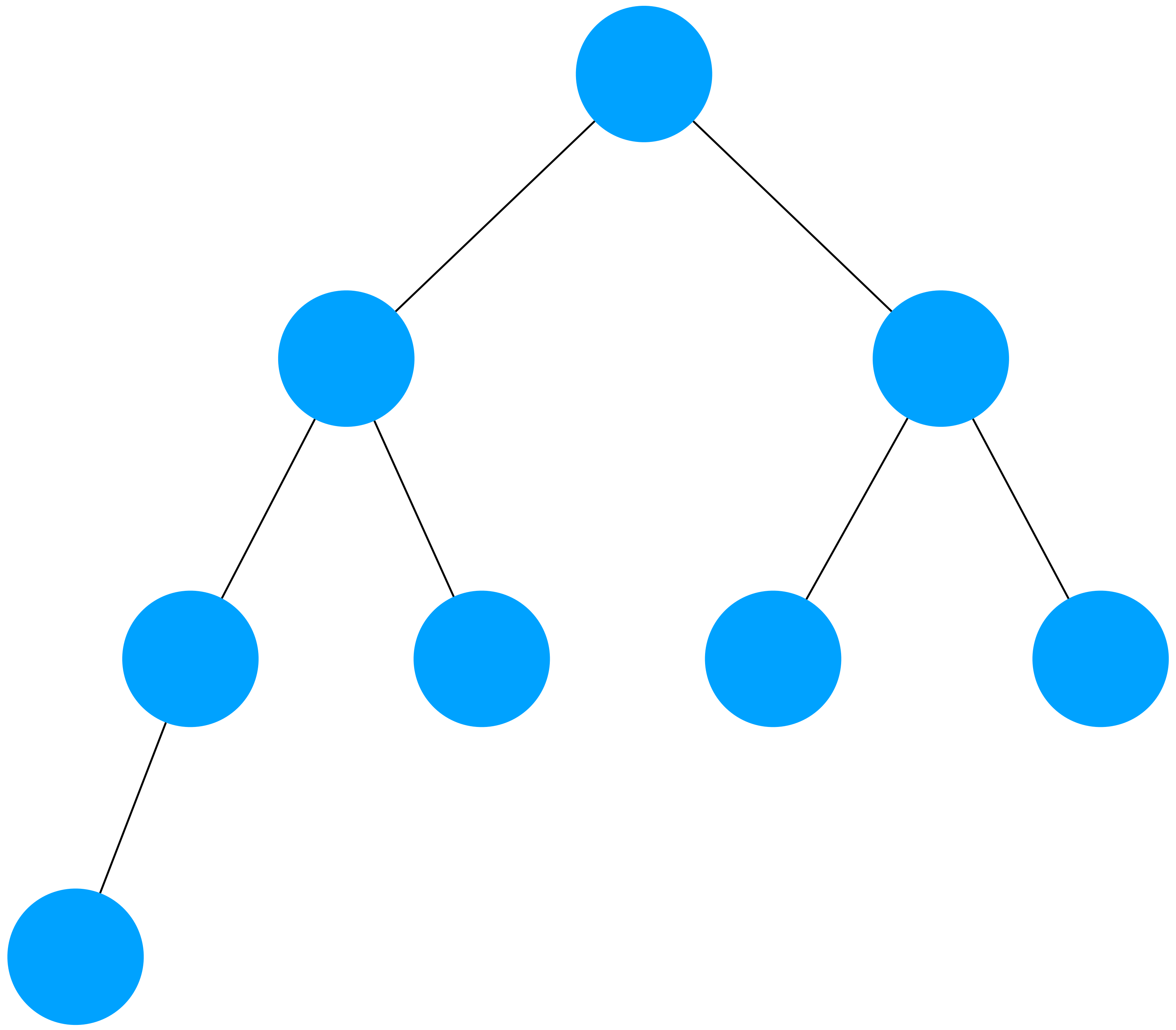
Examples



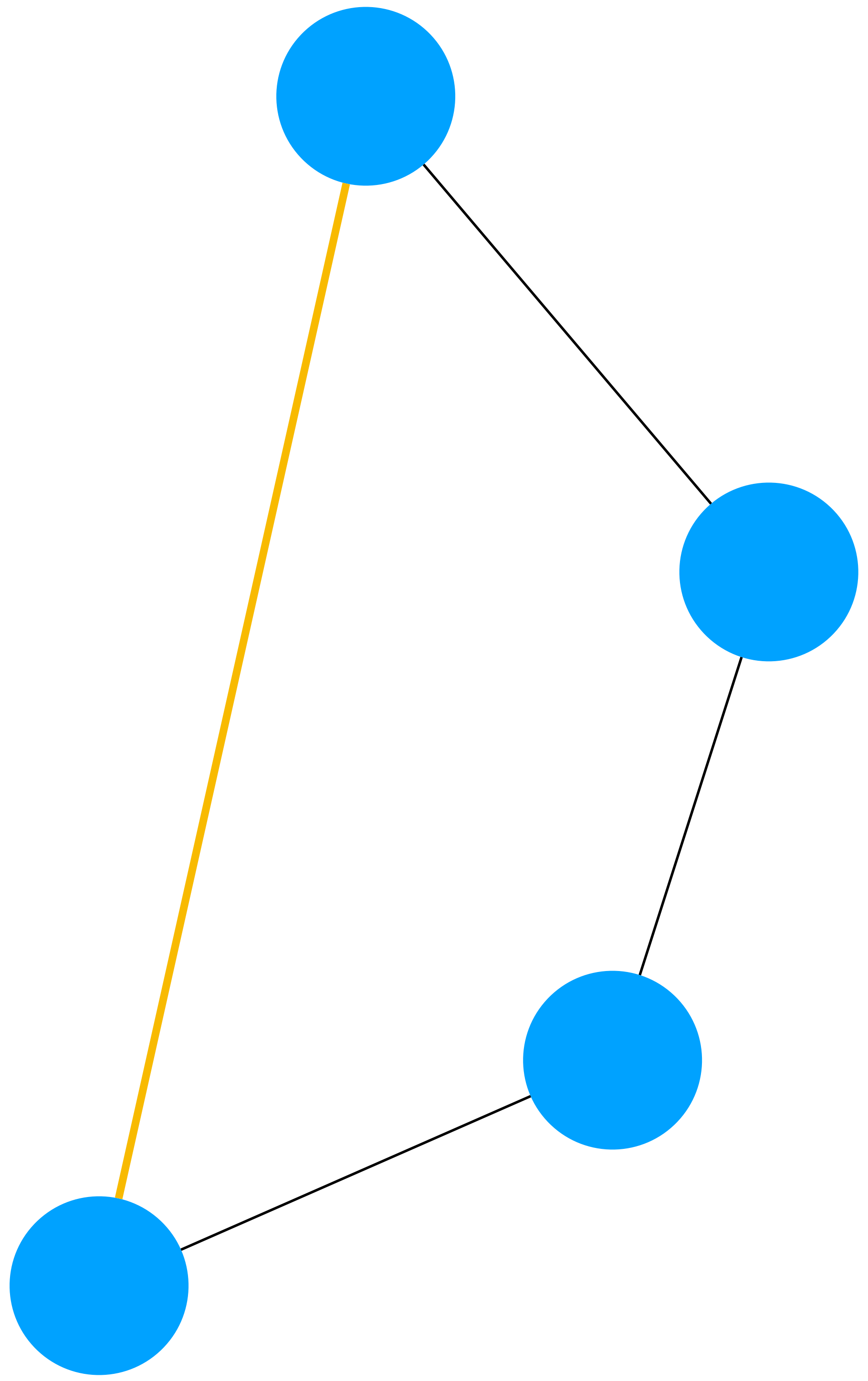
width 1



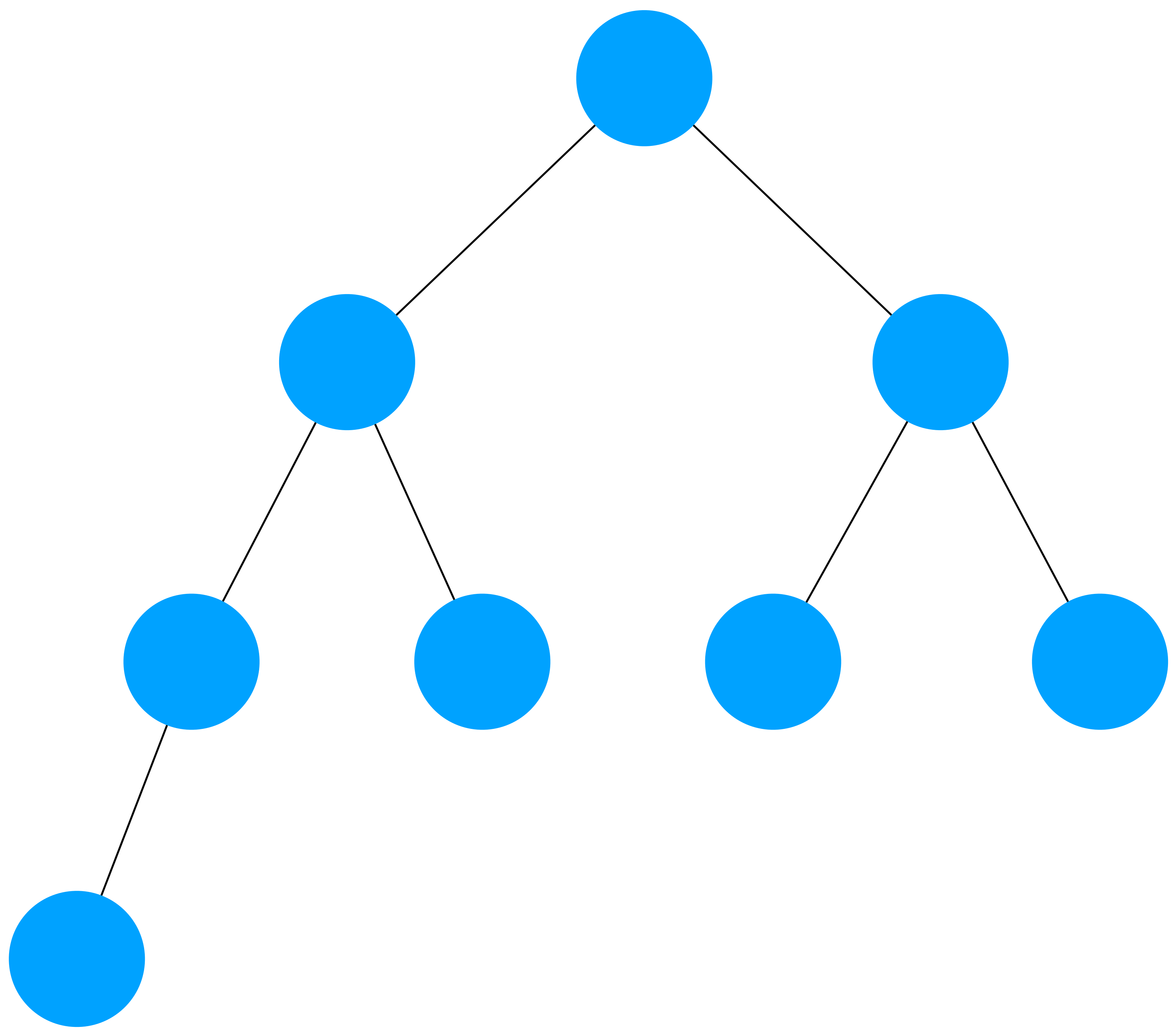
Examples



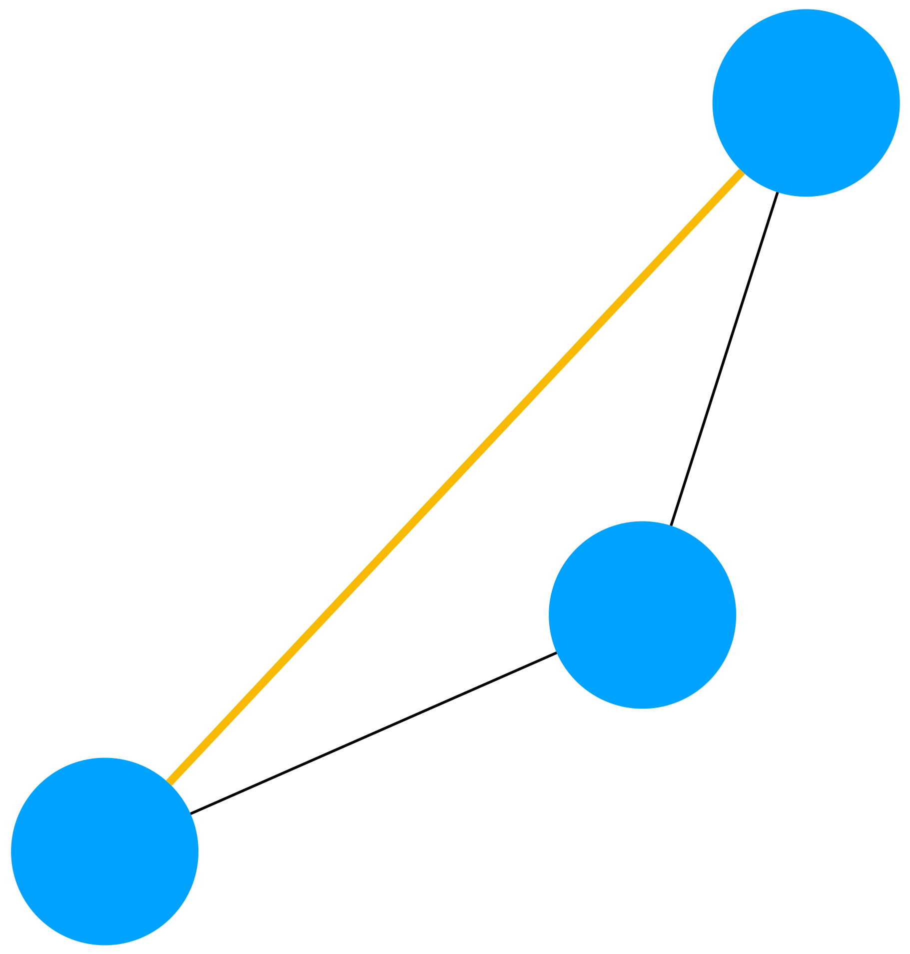
width 1



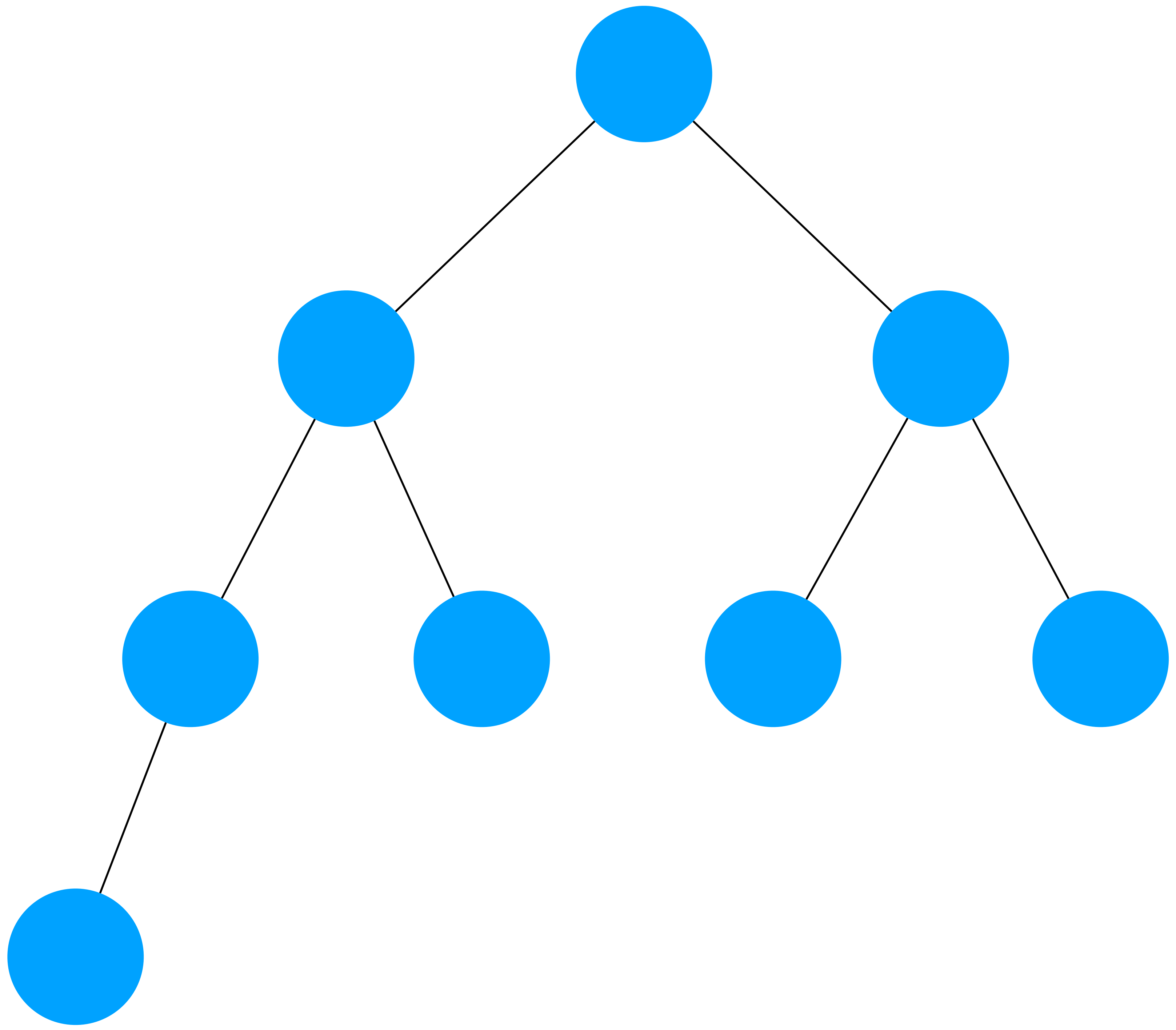
Examples



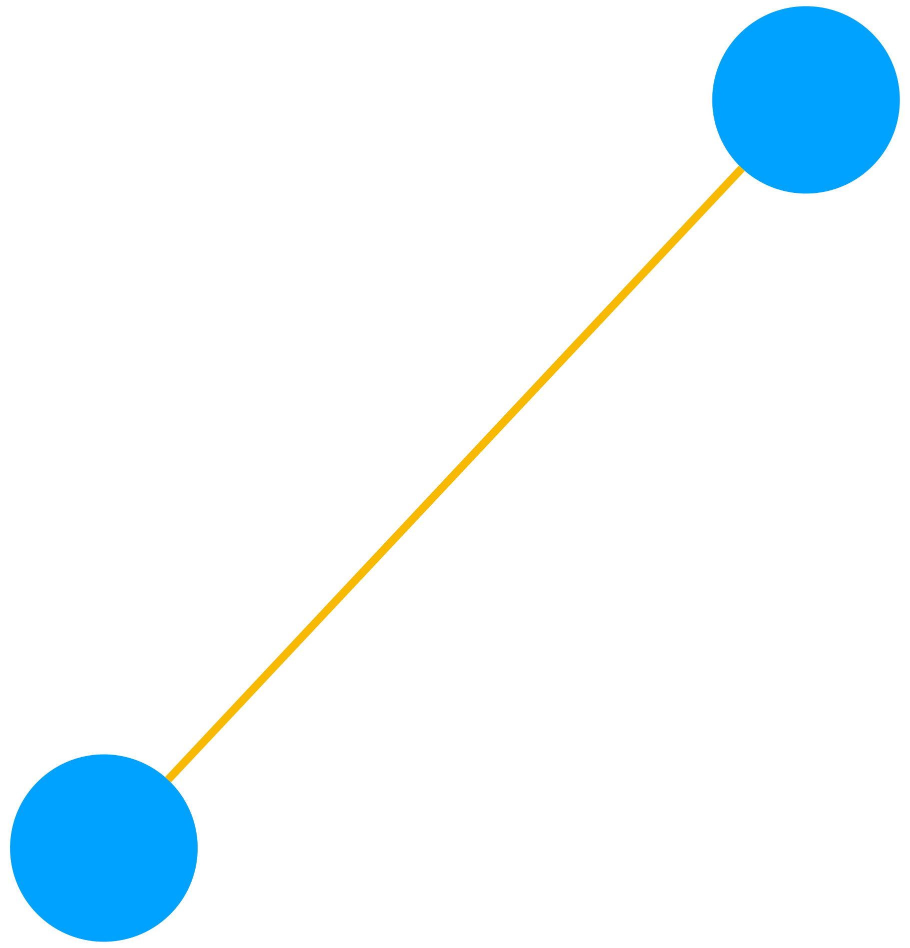
width 1



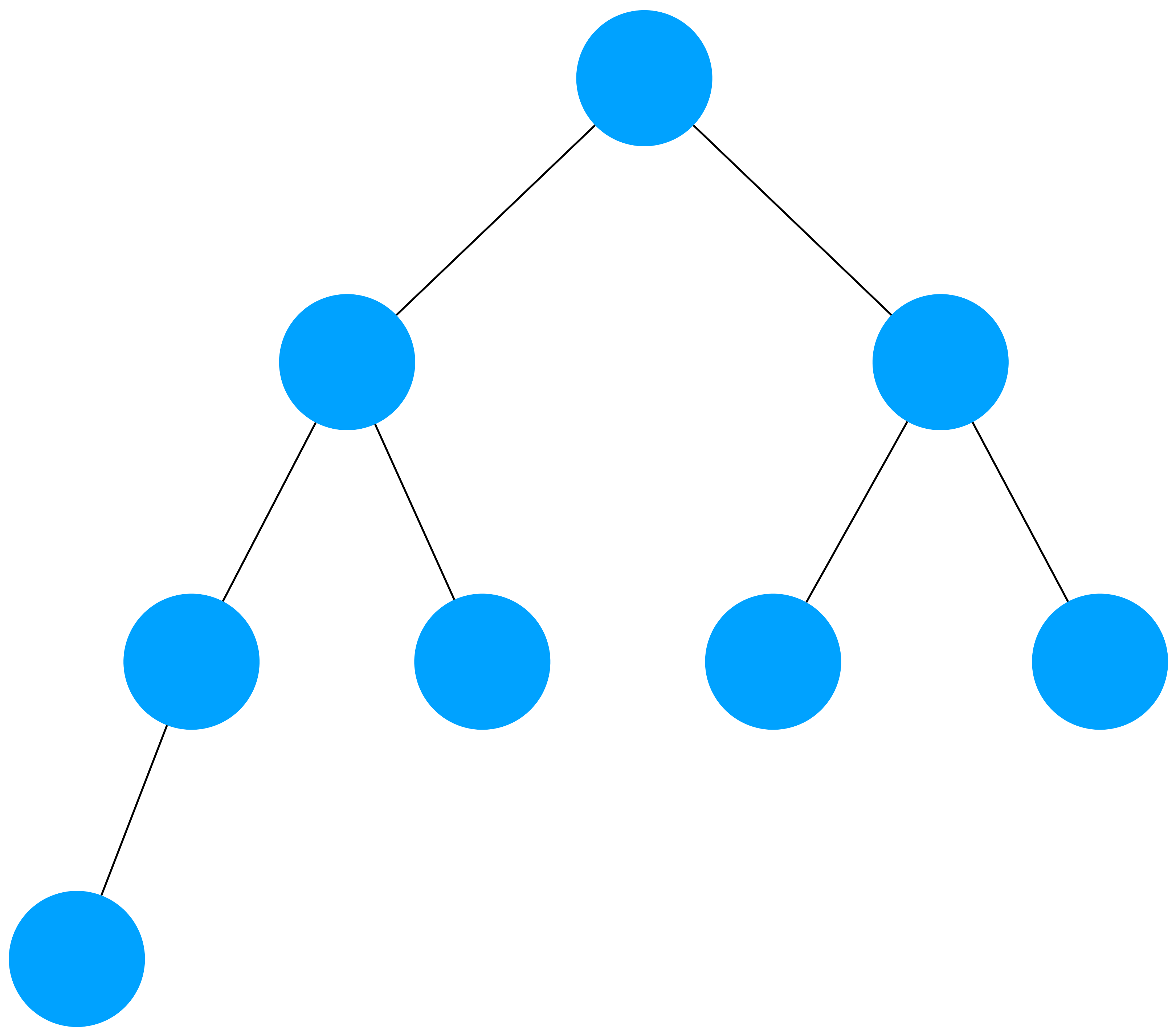
Examples



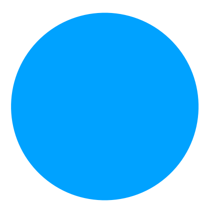
width 1



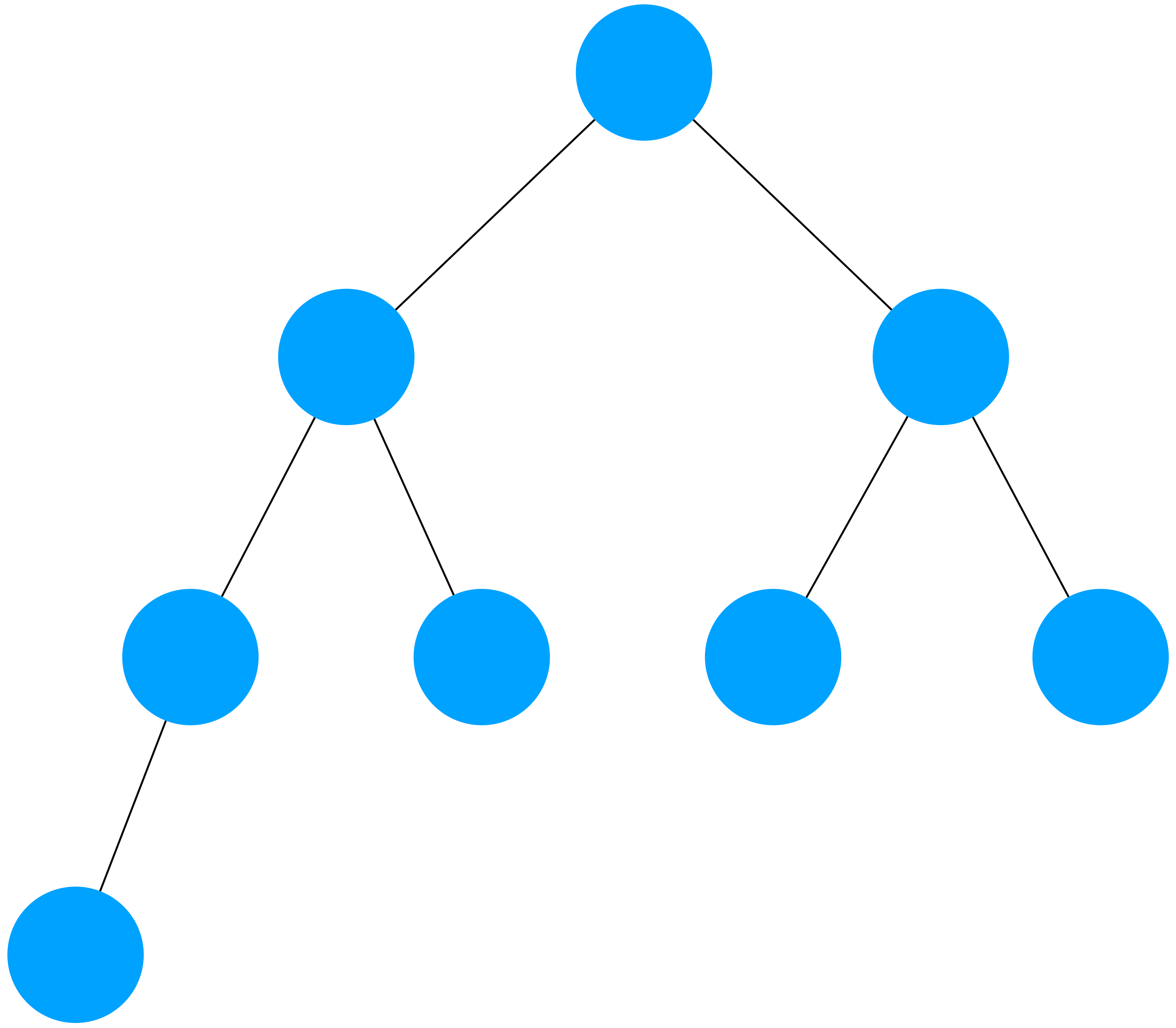
Examples



width 1

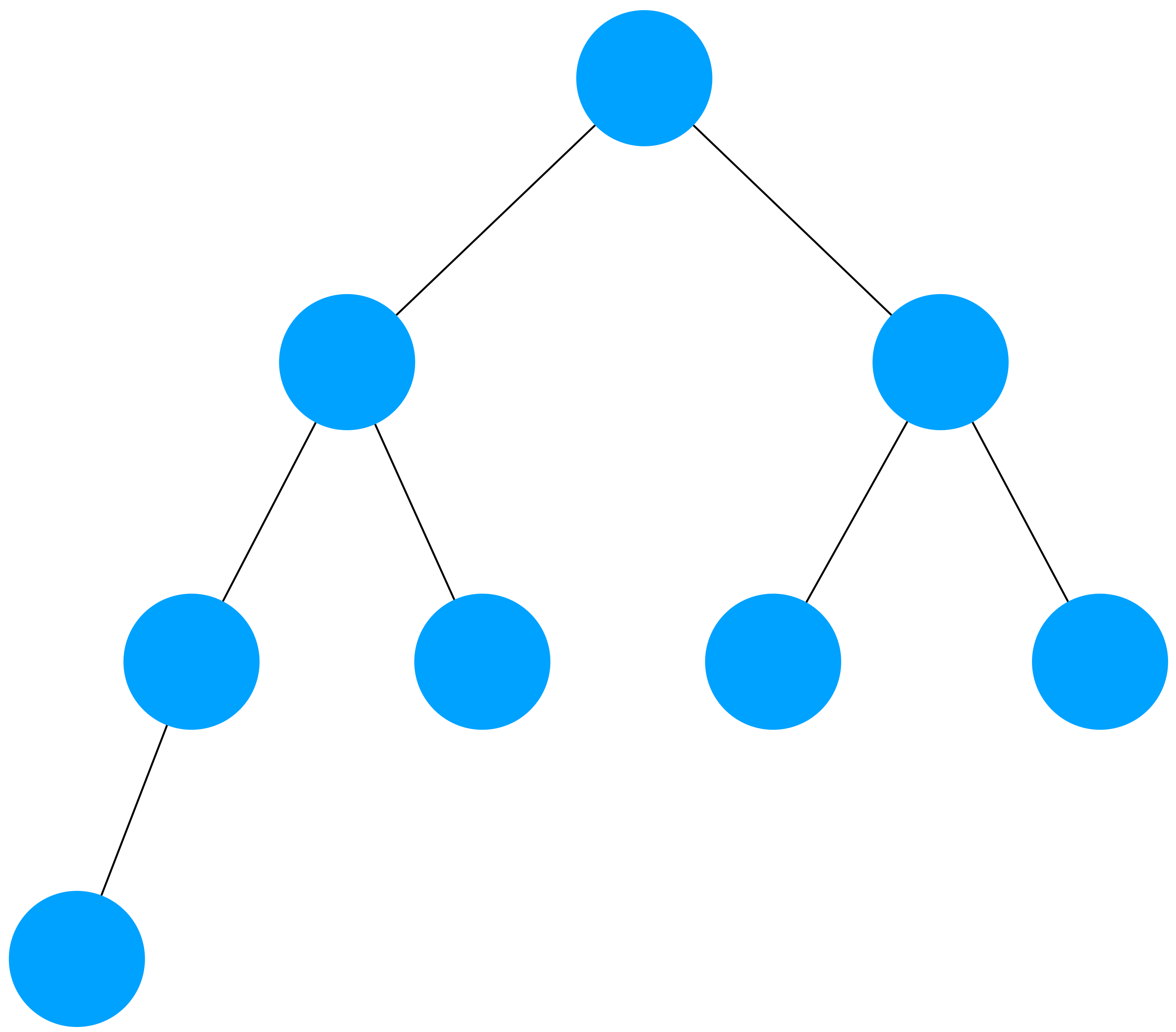


Examples

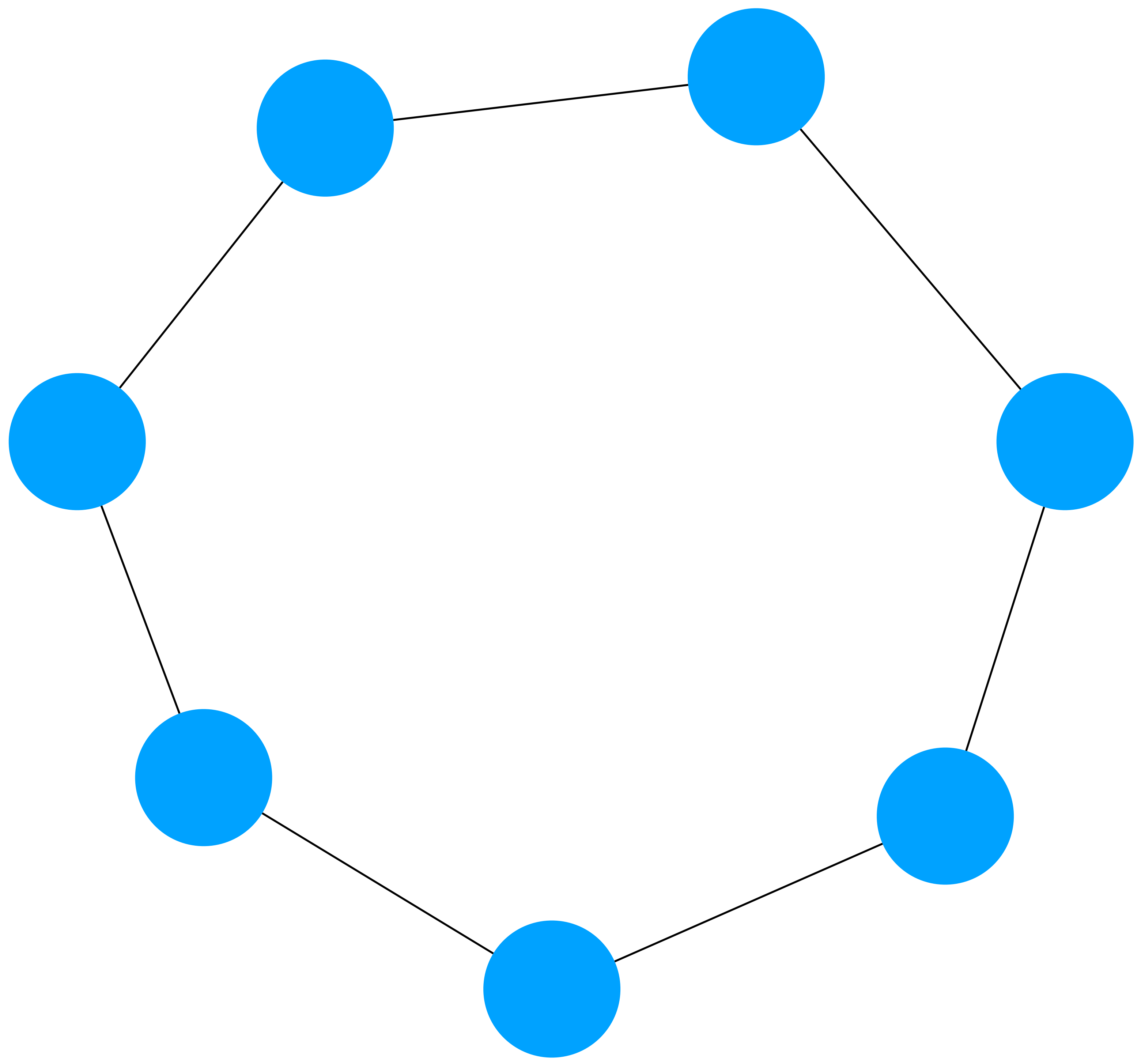


width 1

Examples



width 1



width 2

Treewidth

Treewidth

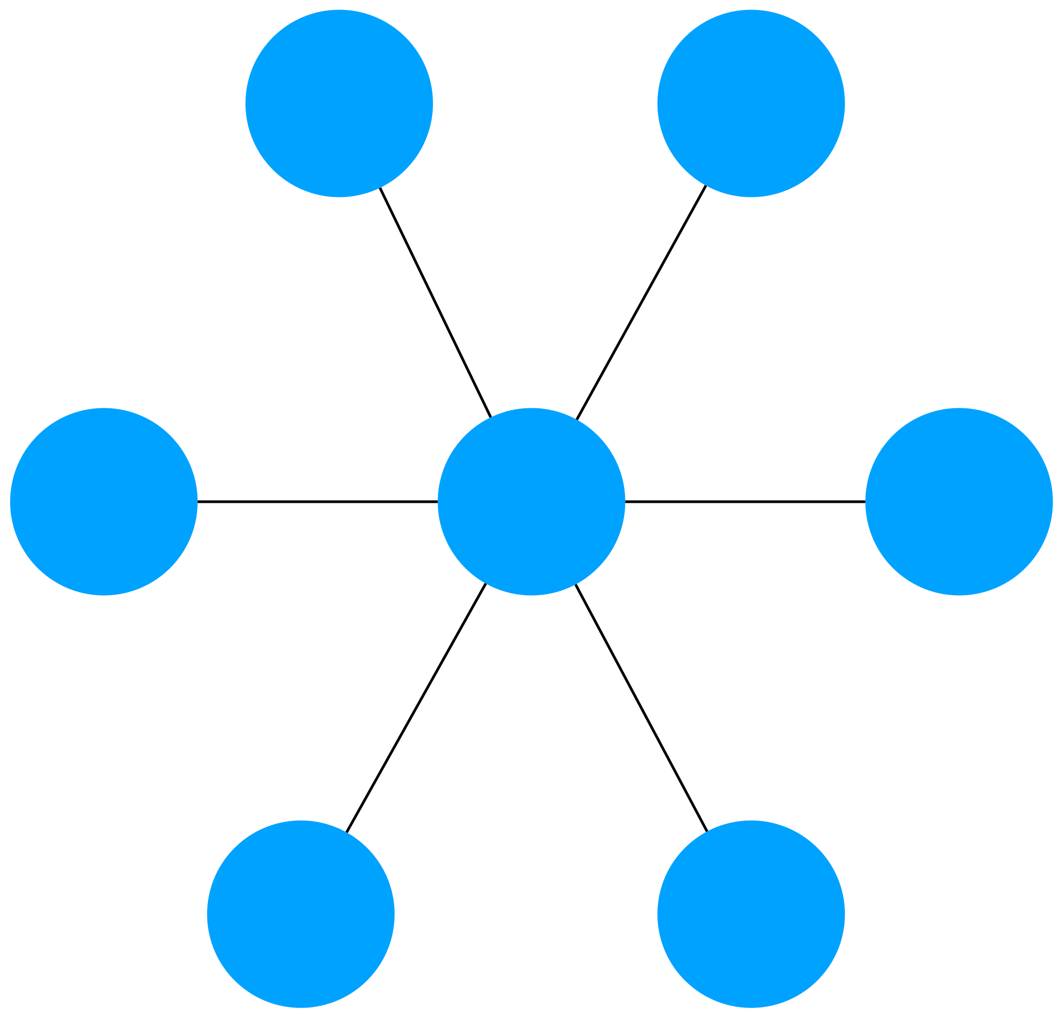
Definition

The **treewidth** of a graph G is the minimum width of an elimination ordering of G.

Treewidth

Definition

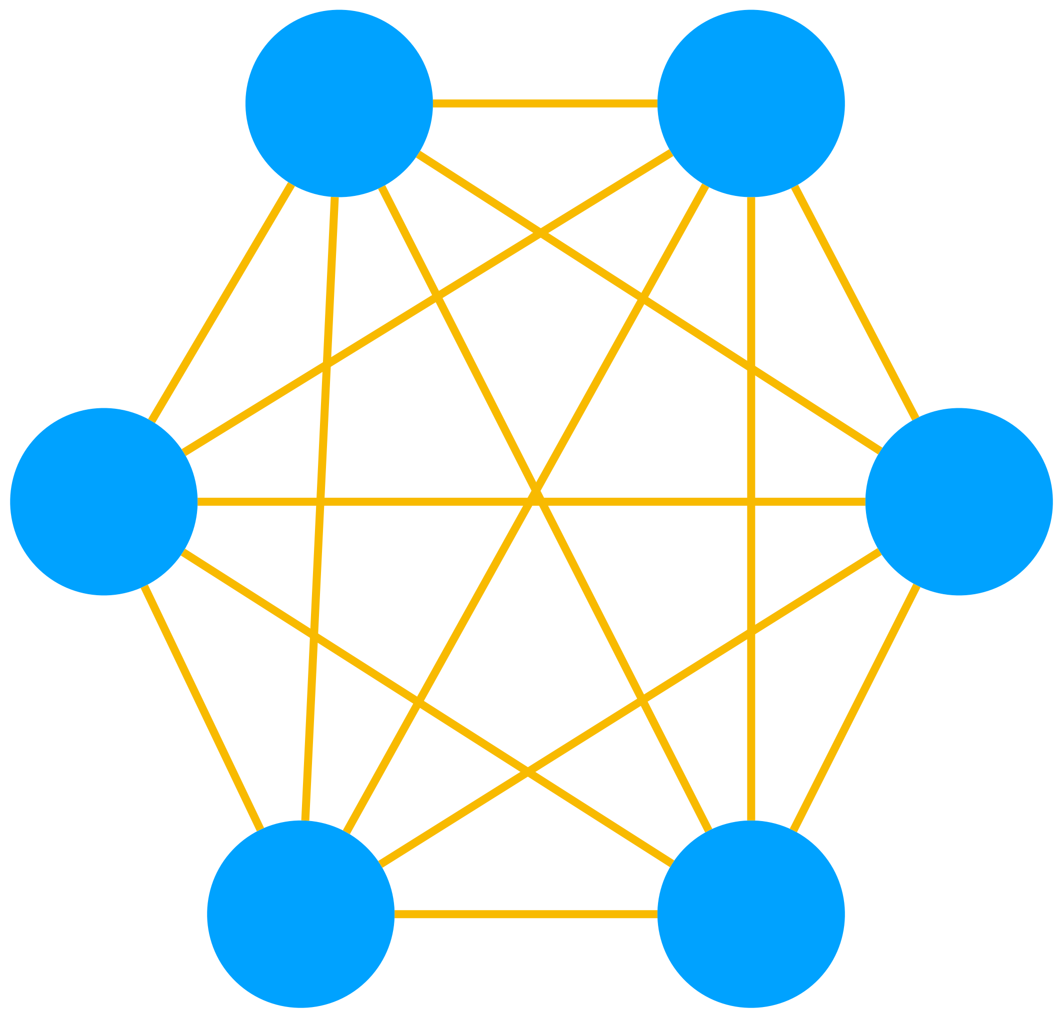
The **treewidth** of a graph G is the minimum width of an elimination ordering of G.



Treewidth

Definition

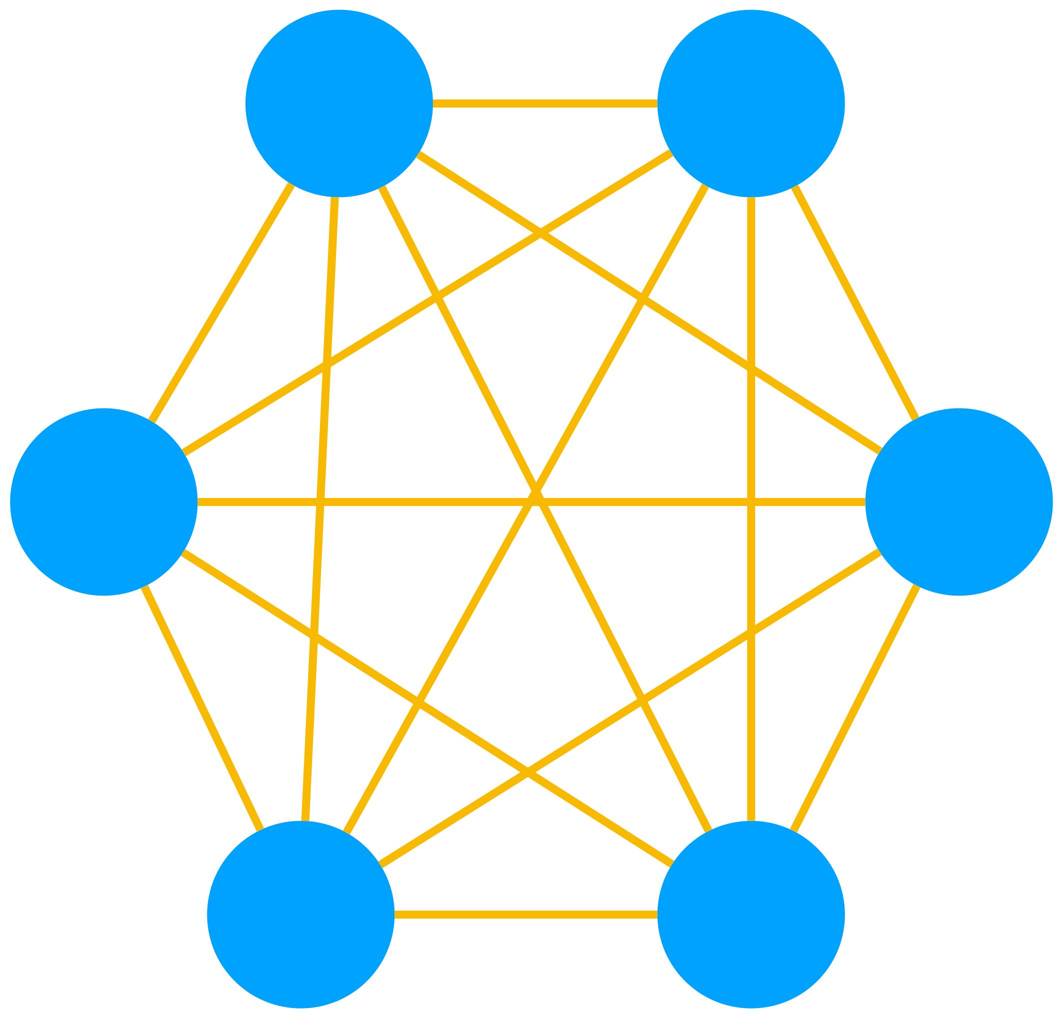
The **treewidth** of a graph G is the minimum width of an elimination ordering of G.



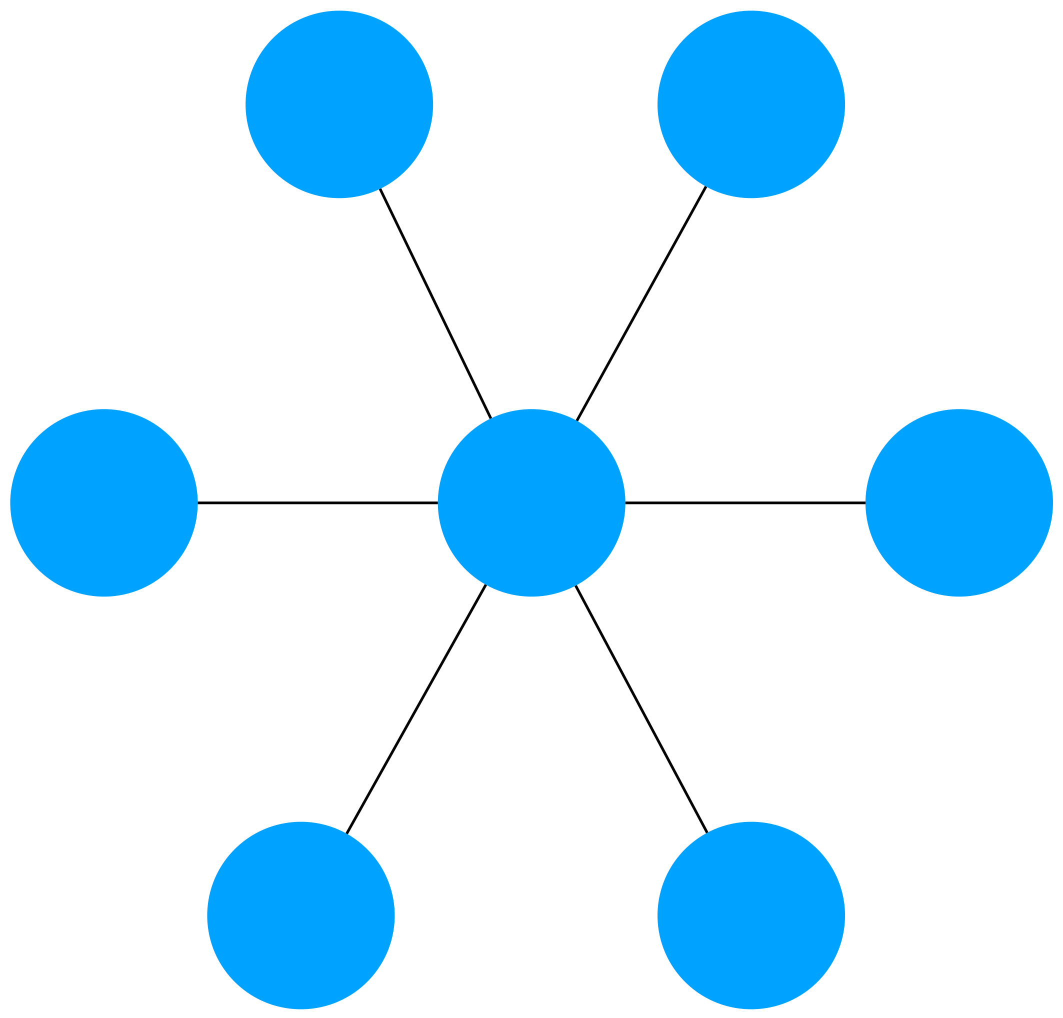
Treewidth

Definition

The **treewidth** of a graph G is the minimum width of an elimination ordering of G.



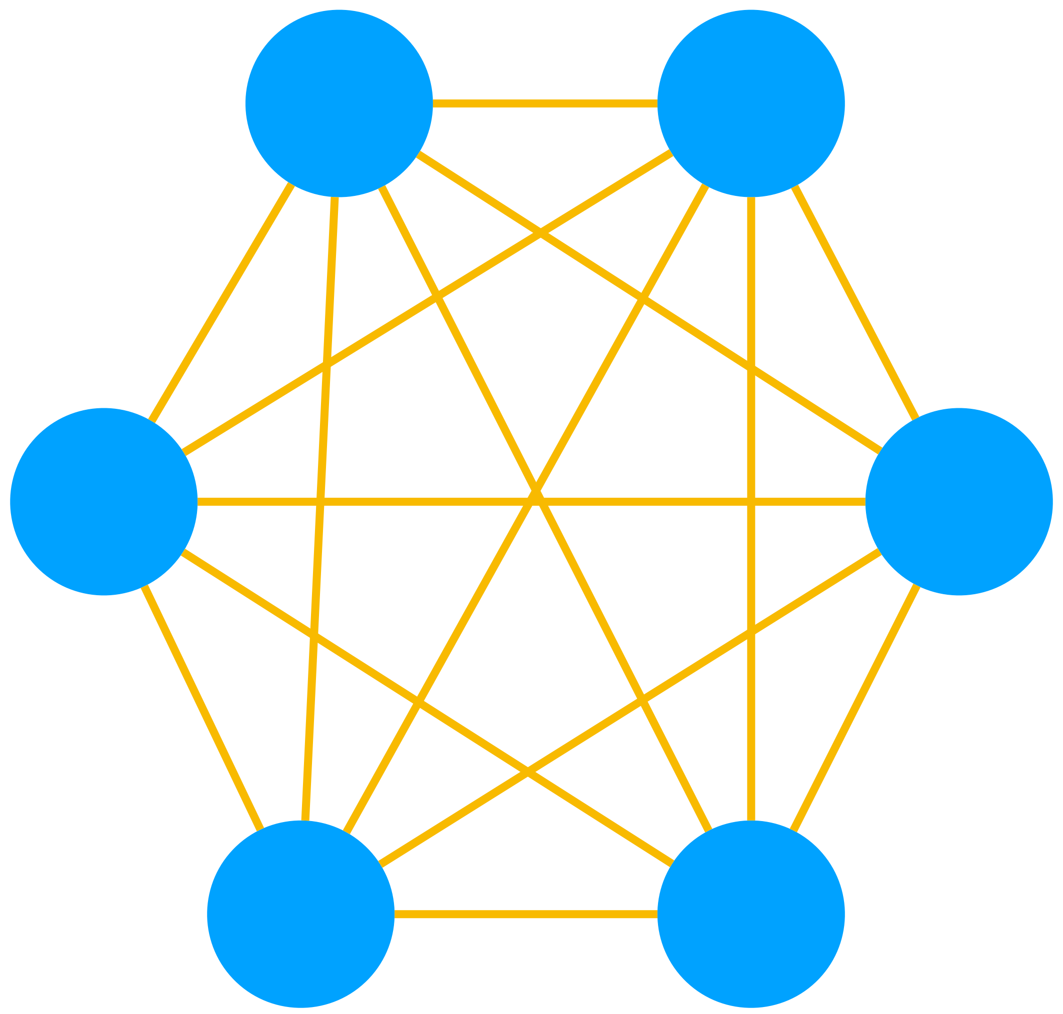
width 6



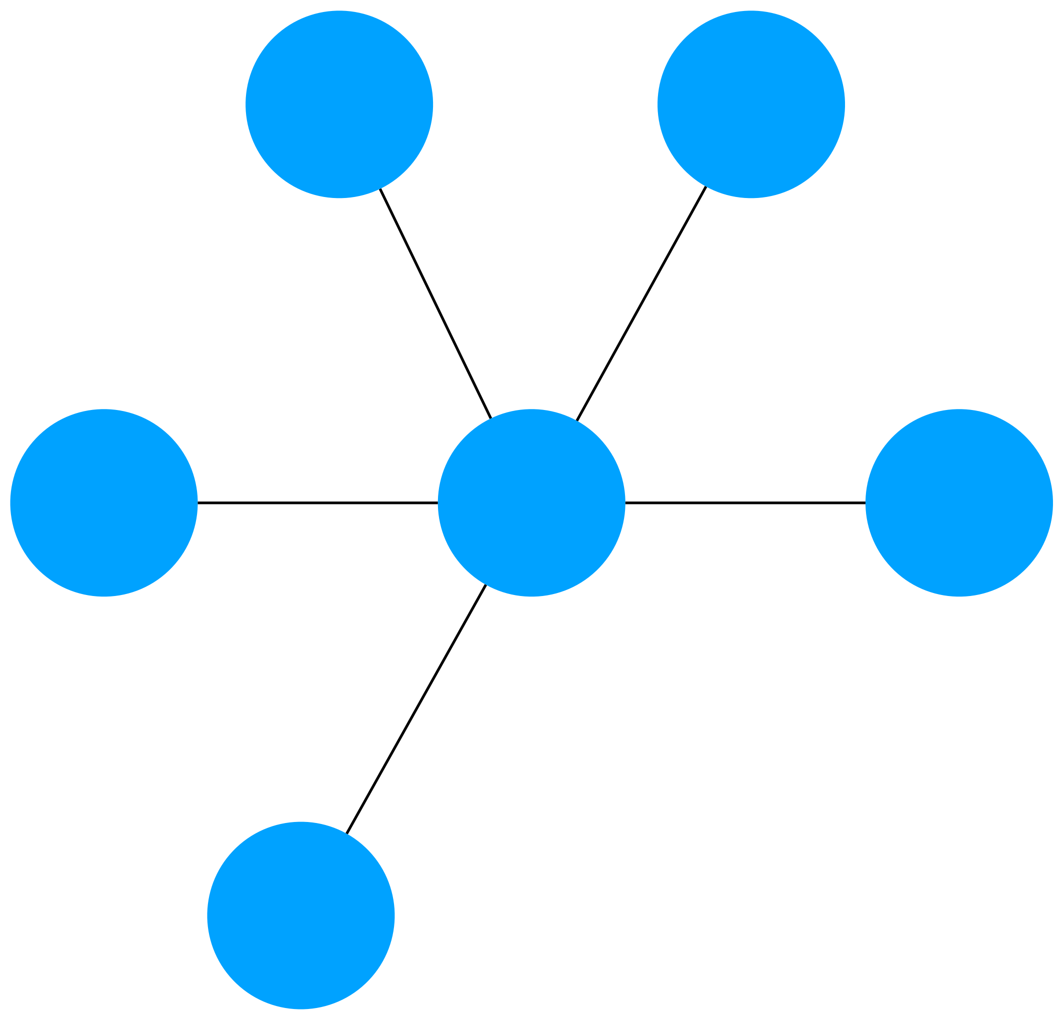
Treewidth

Definition

The **treewidth** of a graph G is the minimum width of an elimination ordering of G.



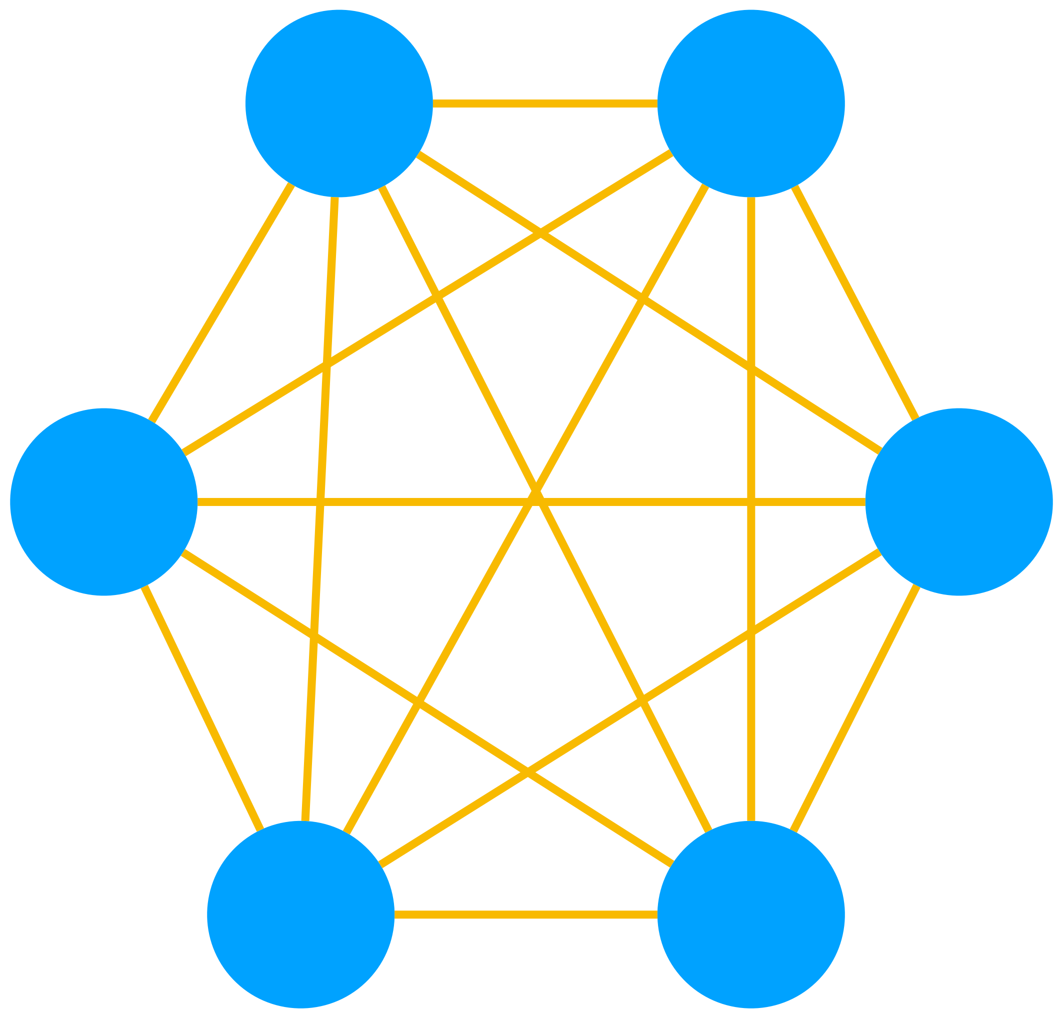
width 6



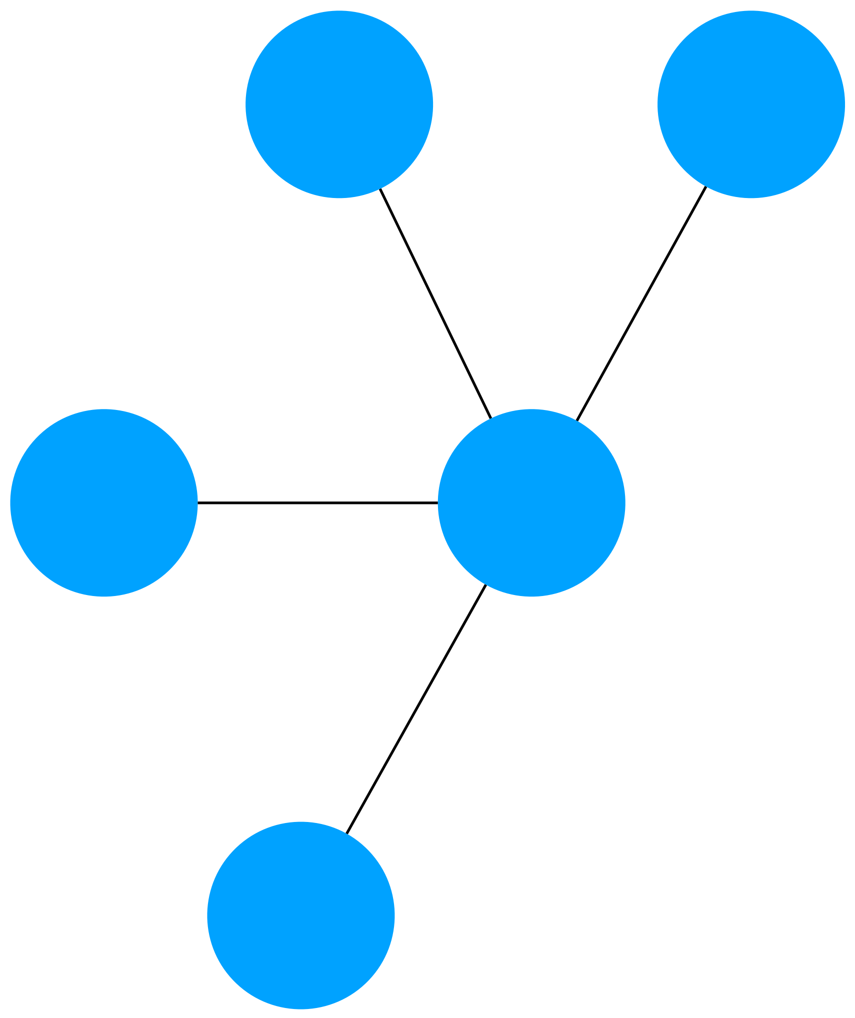
Treewidth

Definition

The **treewidth** of a graph G is the minimum width of an elimination ordering of G.



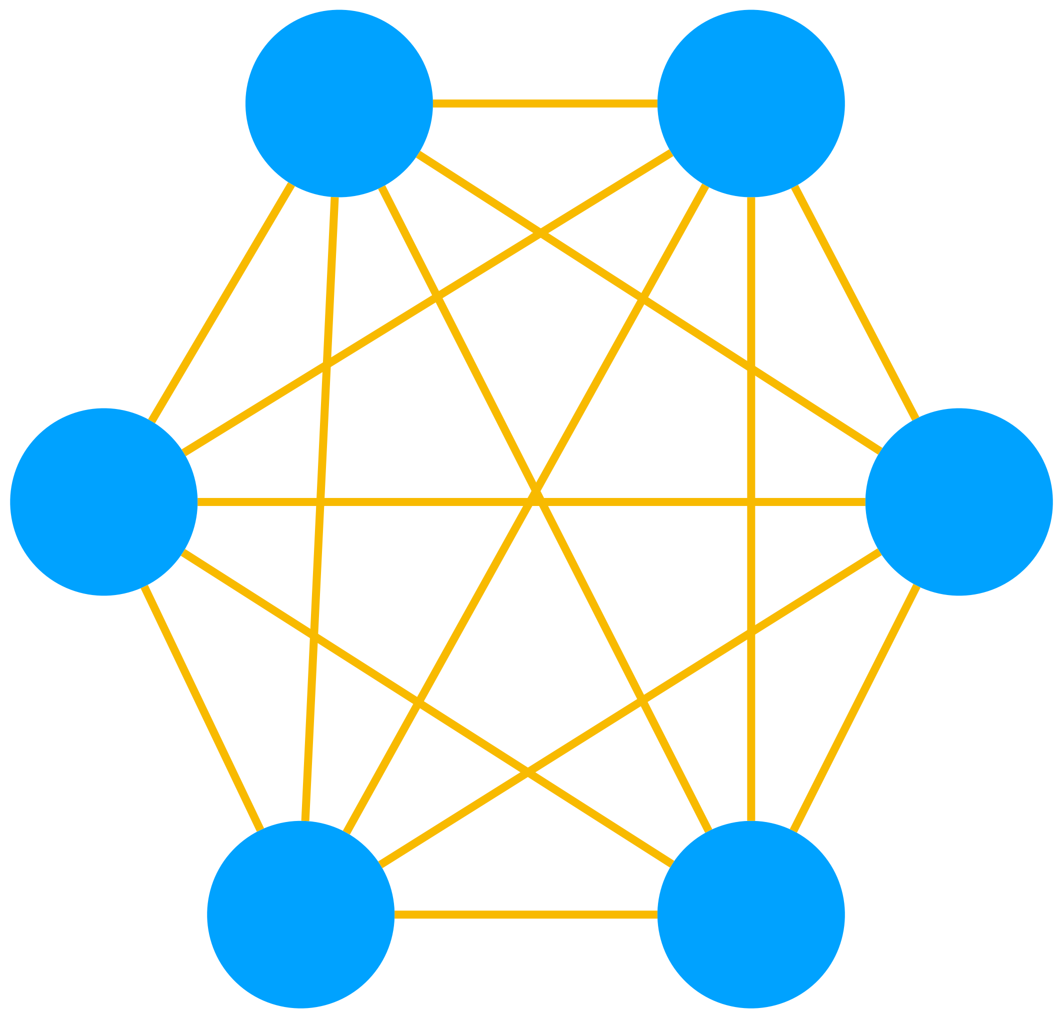
width 6



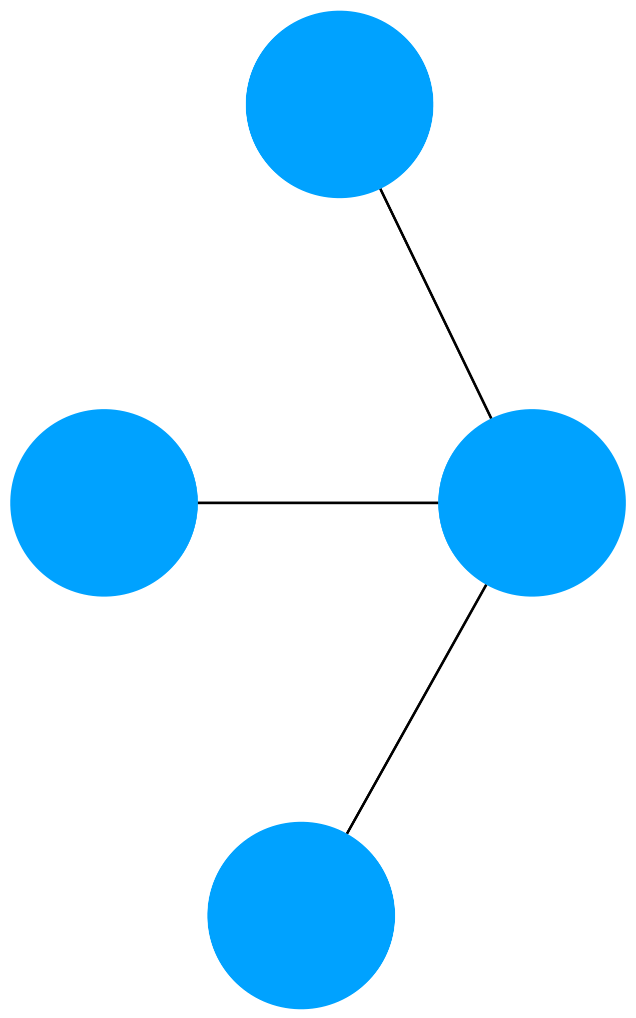
Treewidth

Definition

The **treewidth** of a graph G is the minimum width of an elimination ordering of G.



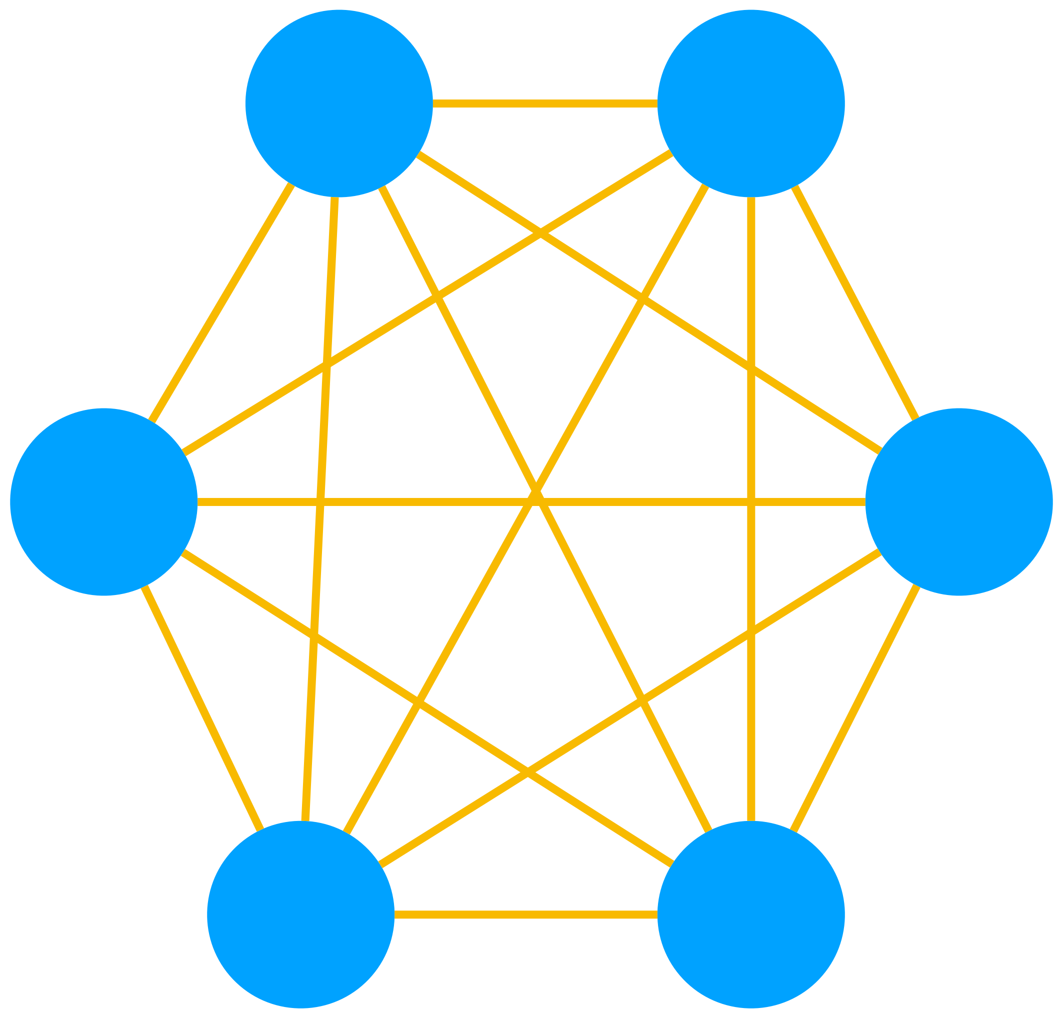
width 6



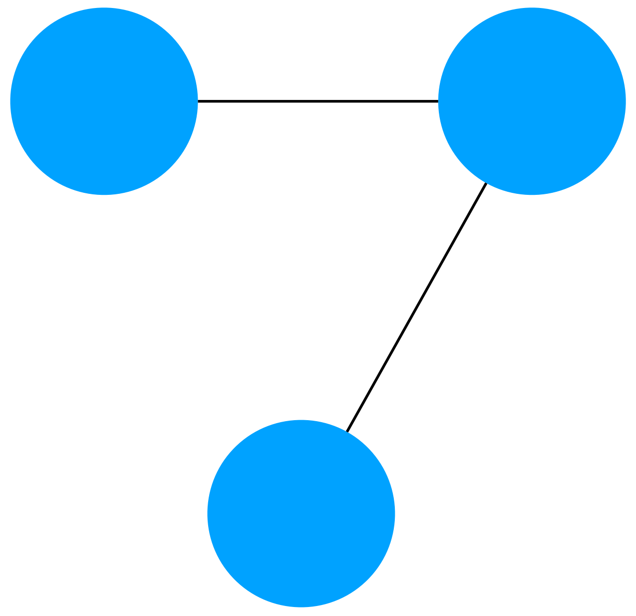
Treewidth

Definition

The **treewidth** of a graph G is the minimum width of an elimination ordering of G.



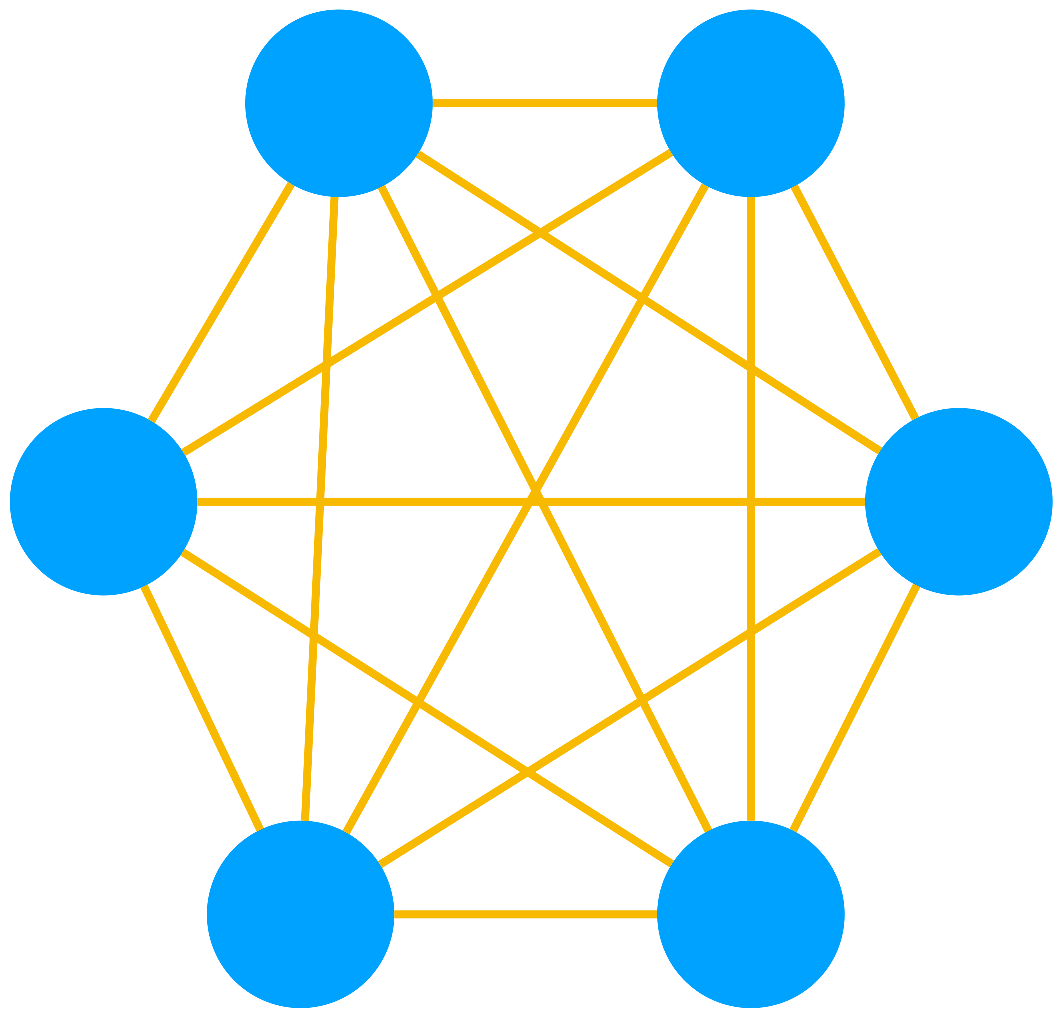
width 6



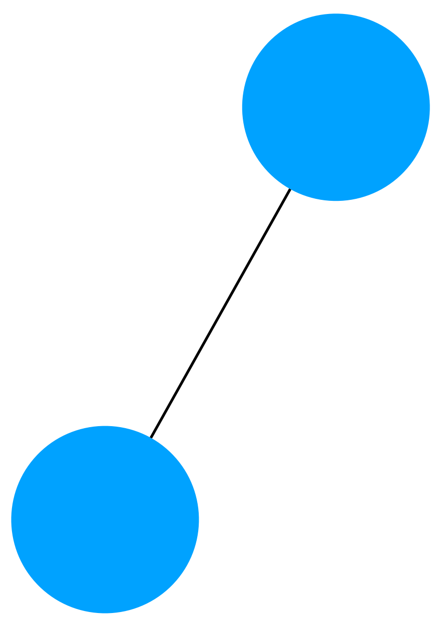
Treewidth

Definition

The **treewidth** of a graph G is the minimum width of an elimination ordering of G.



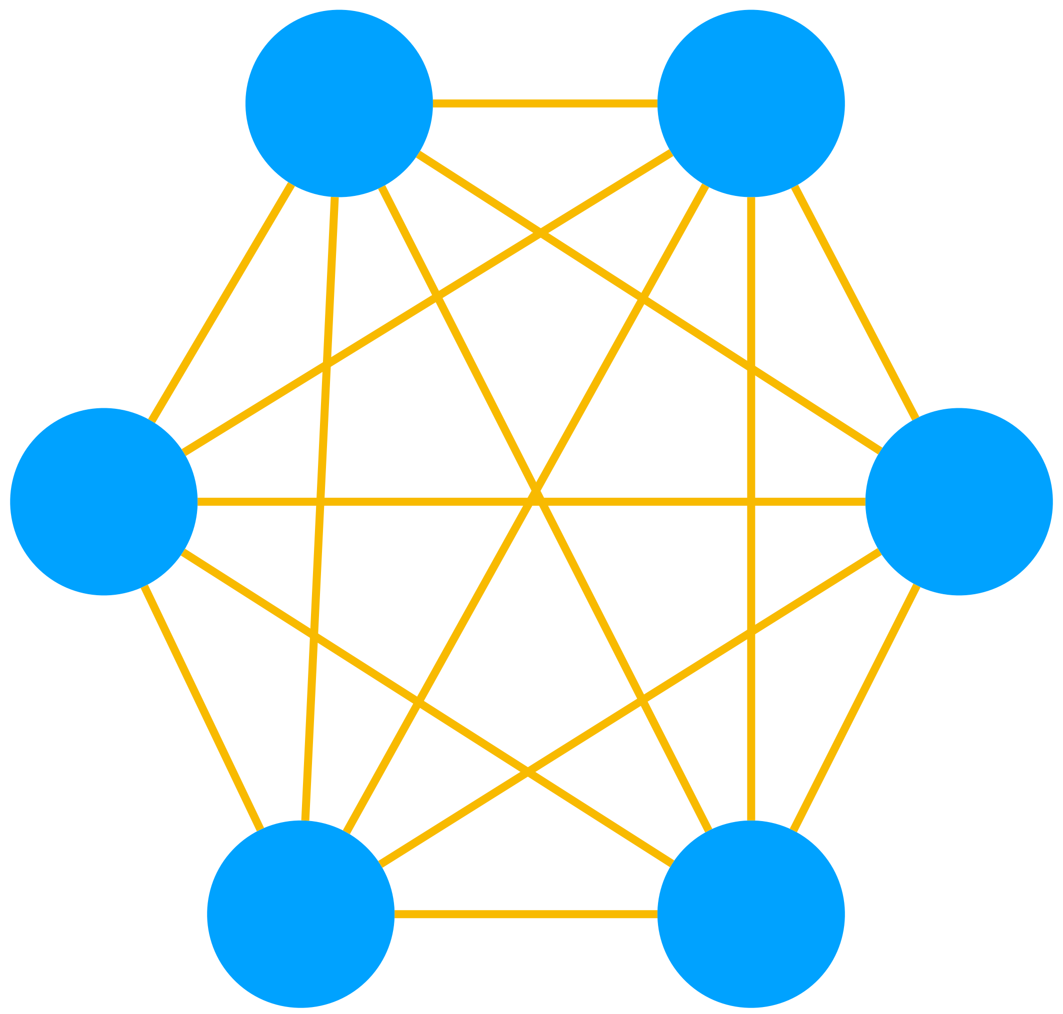
width 6



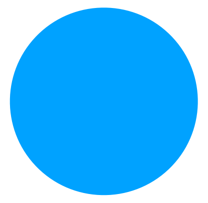
Treewidth

Definition

The **treewidth** of a graph G is the minimum width of an elimination ordering of G.



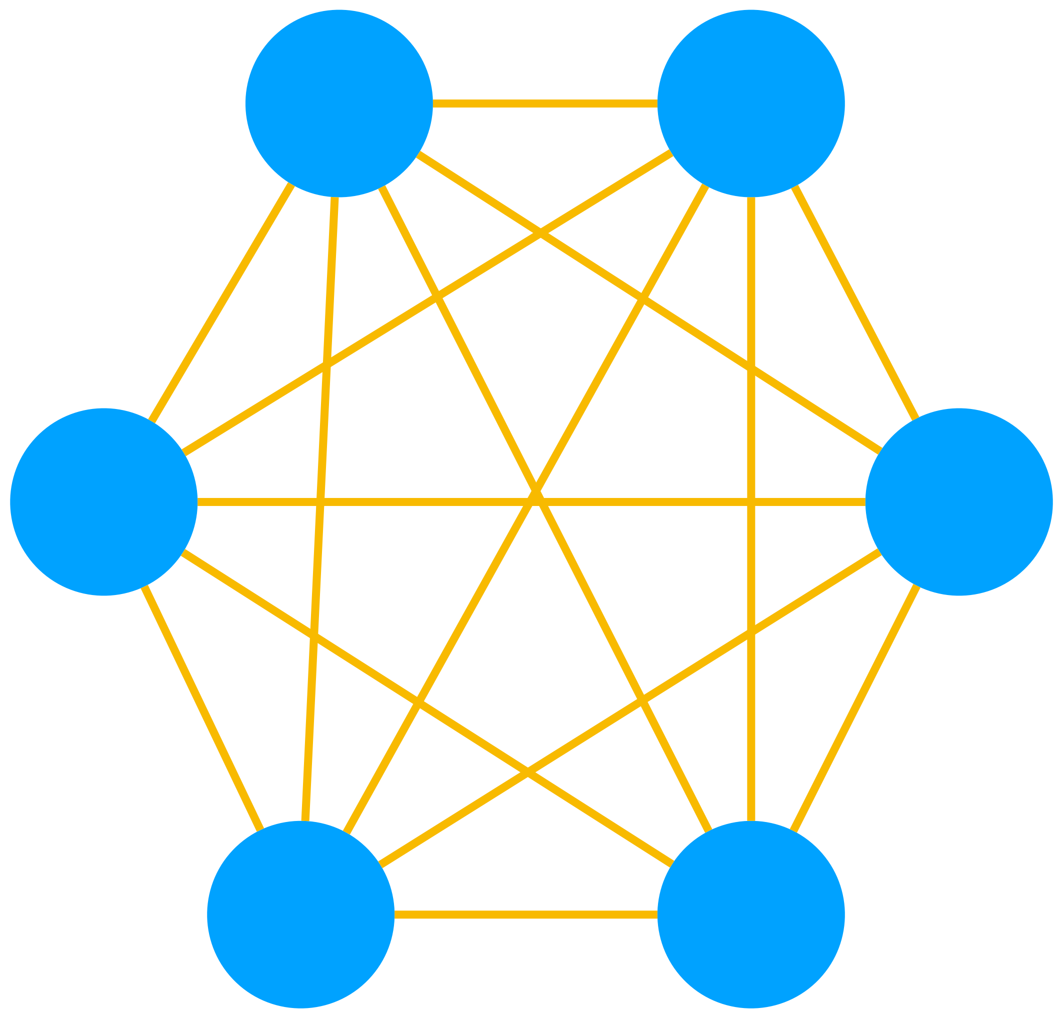
width 6



Treewidth

Definition

The **treewidth** of a graph G is the minimum width of an elimination ordering of G.

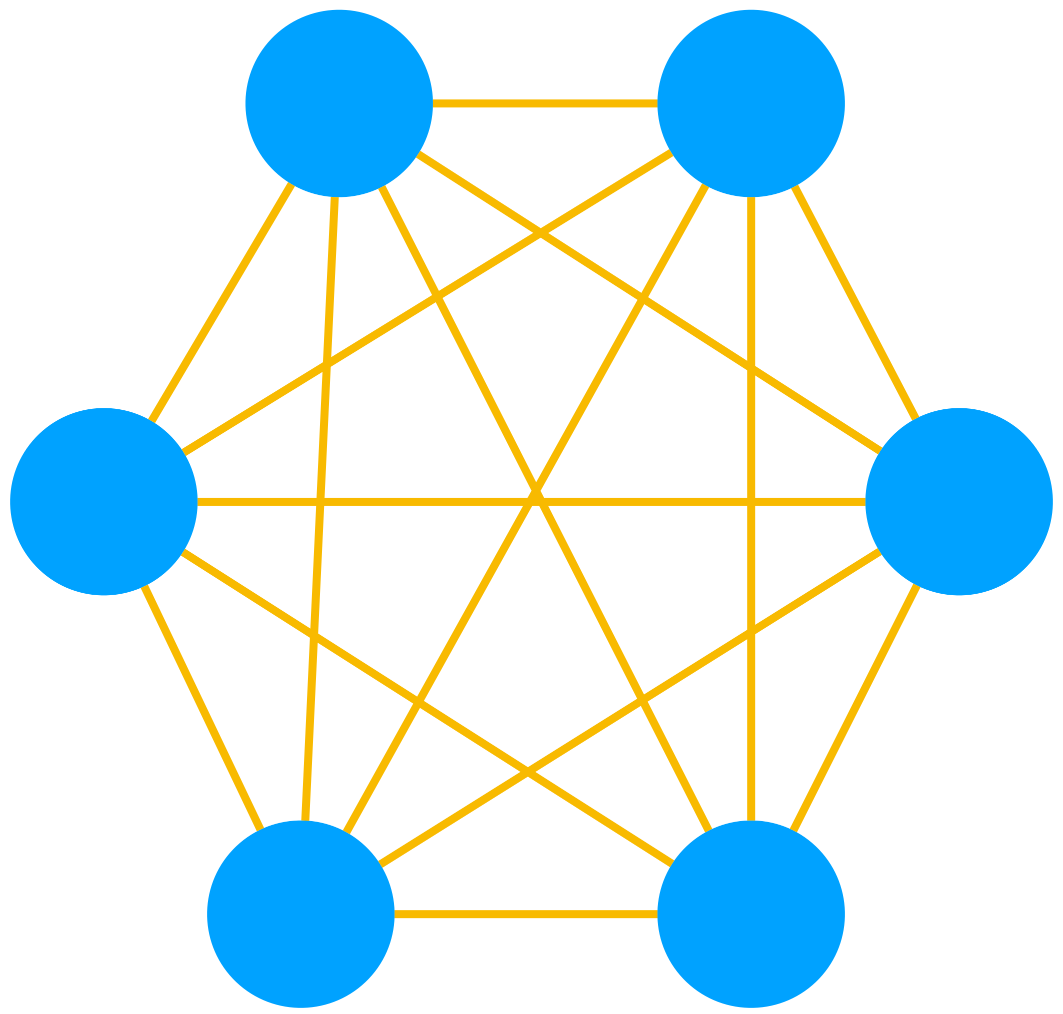


width 6

Treewidth

Definition

The **treewidth** of a graph G is the minimum width of an elimination ordering of G.



width 6

width 1

Degree and Treewidth

Proposition

If G has minimum degree d then $d \leq \mathbf{tw}(G)$.

Degree and Treewidth

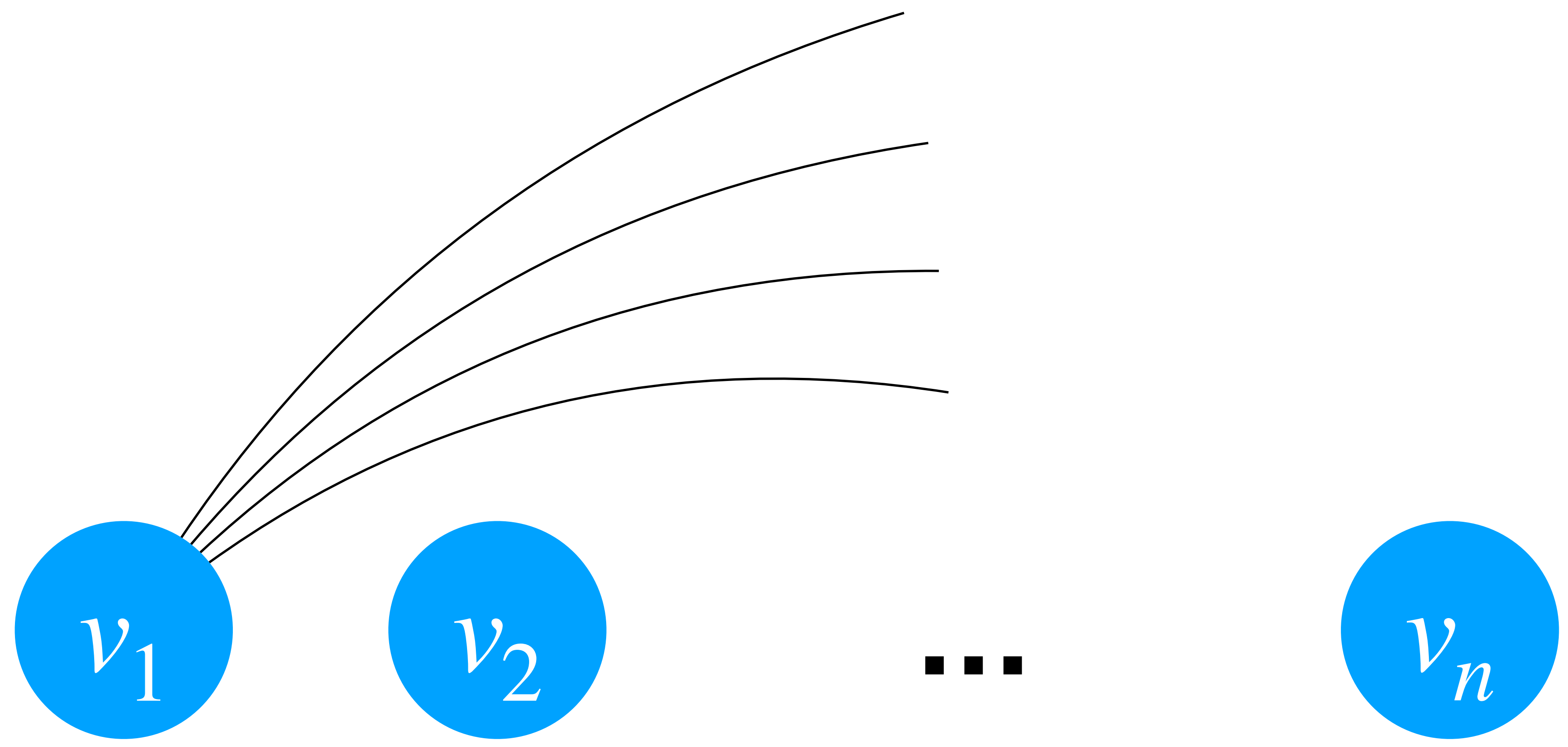
Proposition

If G has minimum degree d then $d \leq \mathbf{tw}(G)$.

Degree and Treewidth

Proposition

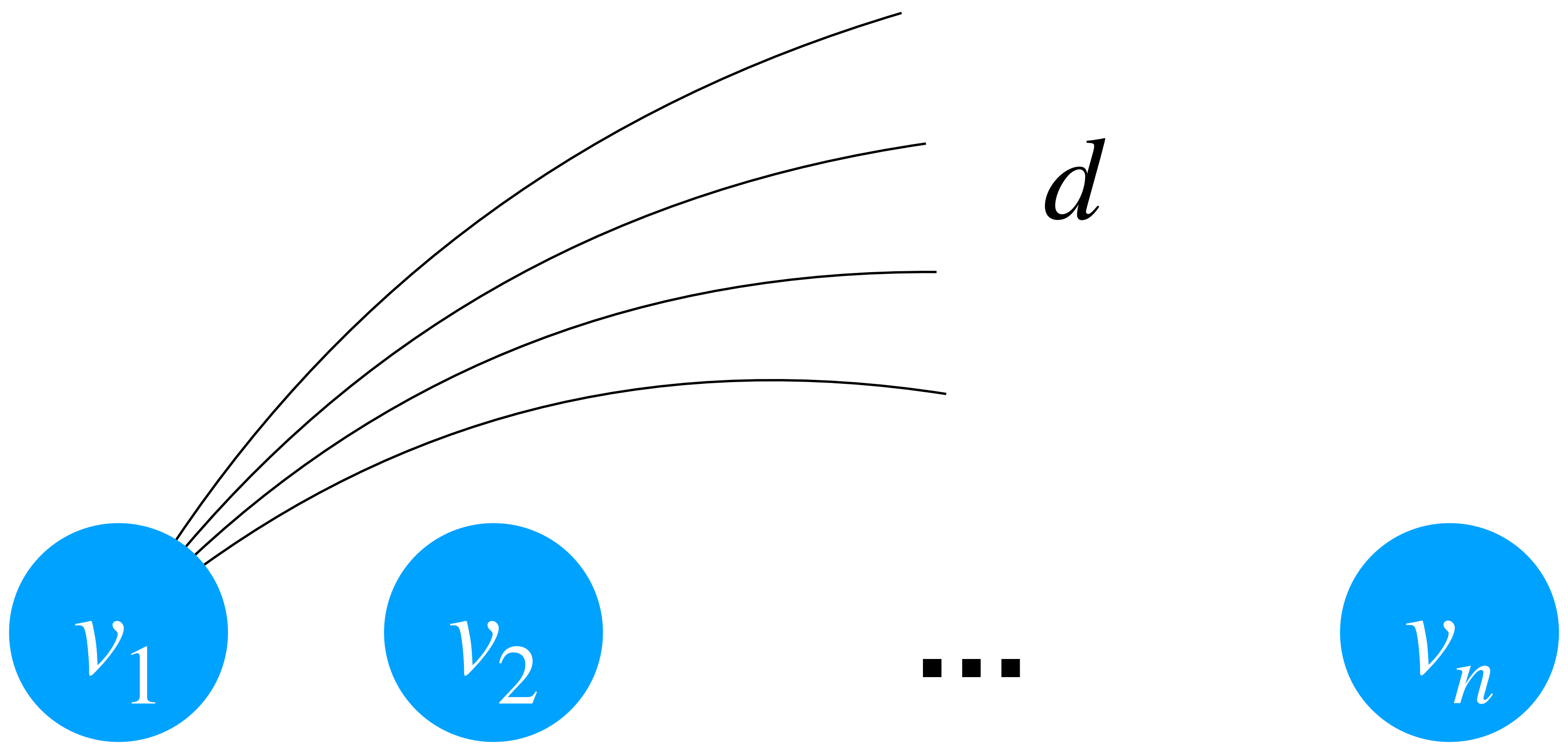
If G has minimum degree d then $d \leq \mathbf{tw}(G)$.



Degree and Treewidth

Proposition

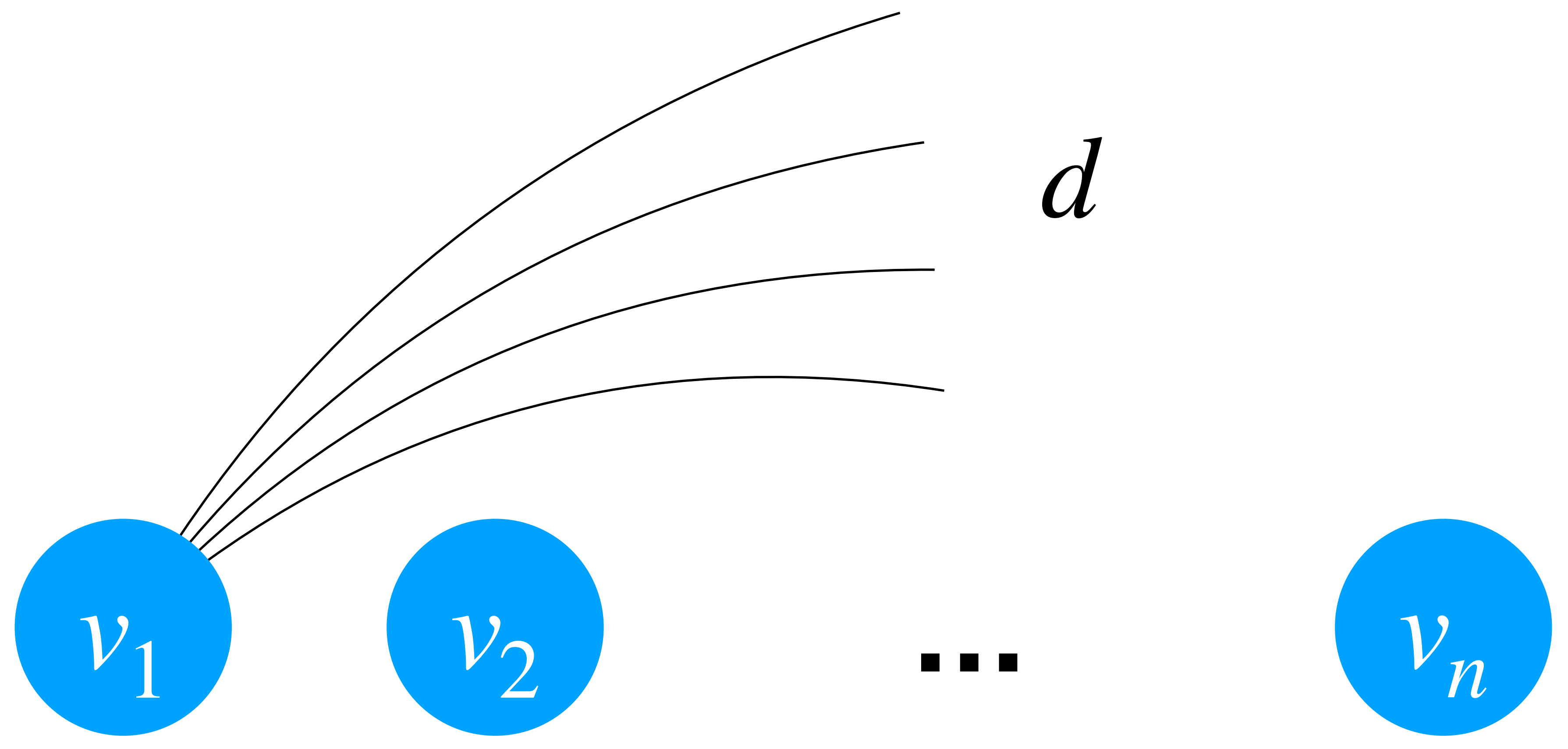
If G has minimum degree d then $d \leq \mathbf{tw}(G)$.

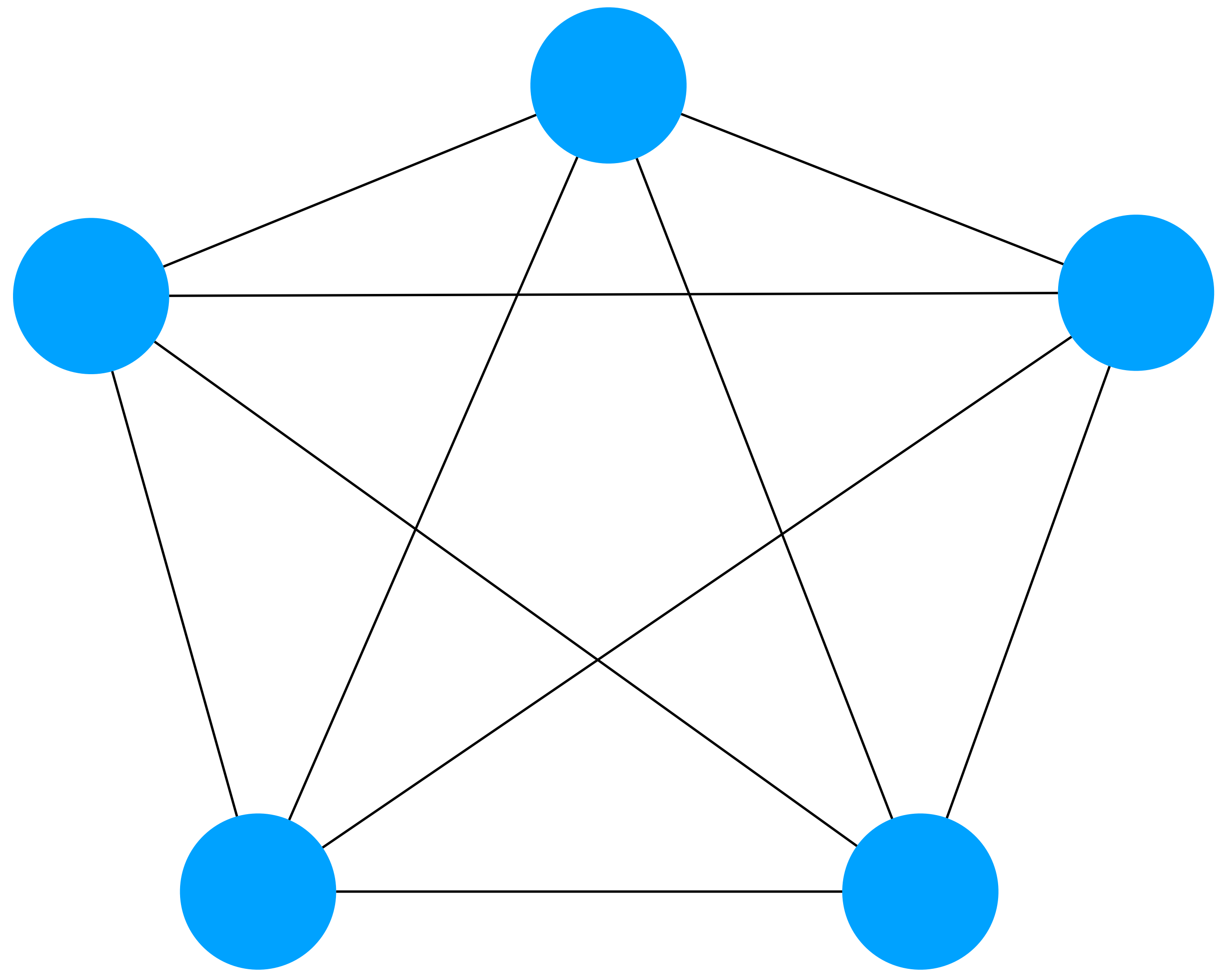


Degree and Treewidth

Proposition

If G has minimum degree d then $d \leq \mathbf{tw}(G)$.



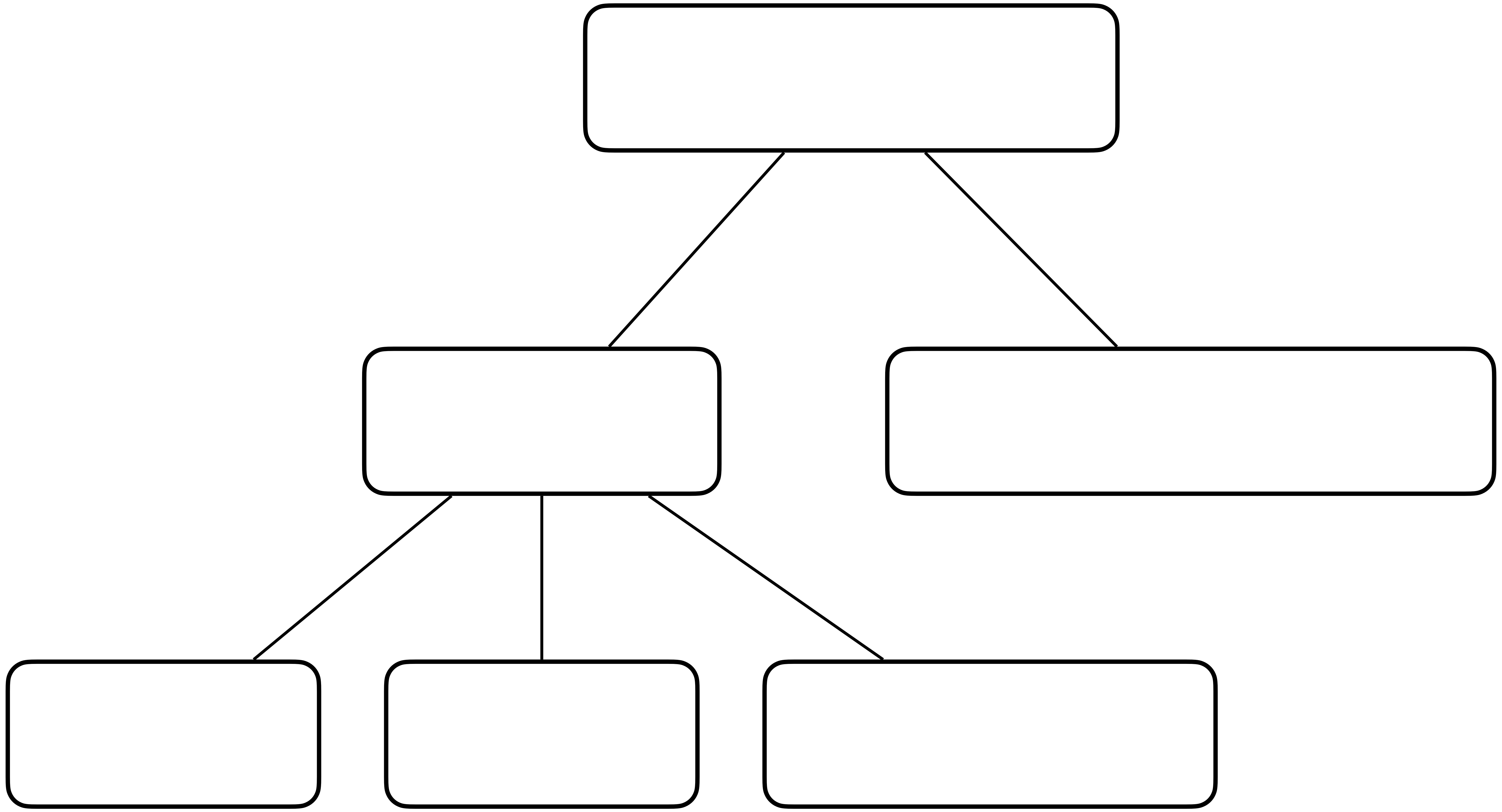


K_n treewidth $n - 1$

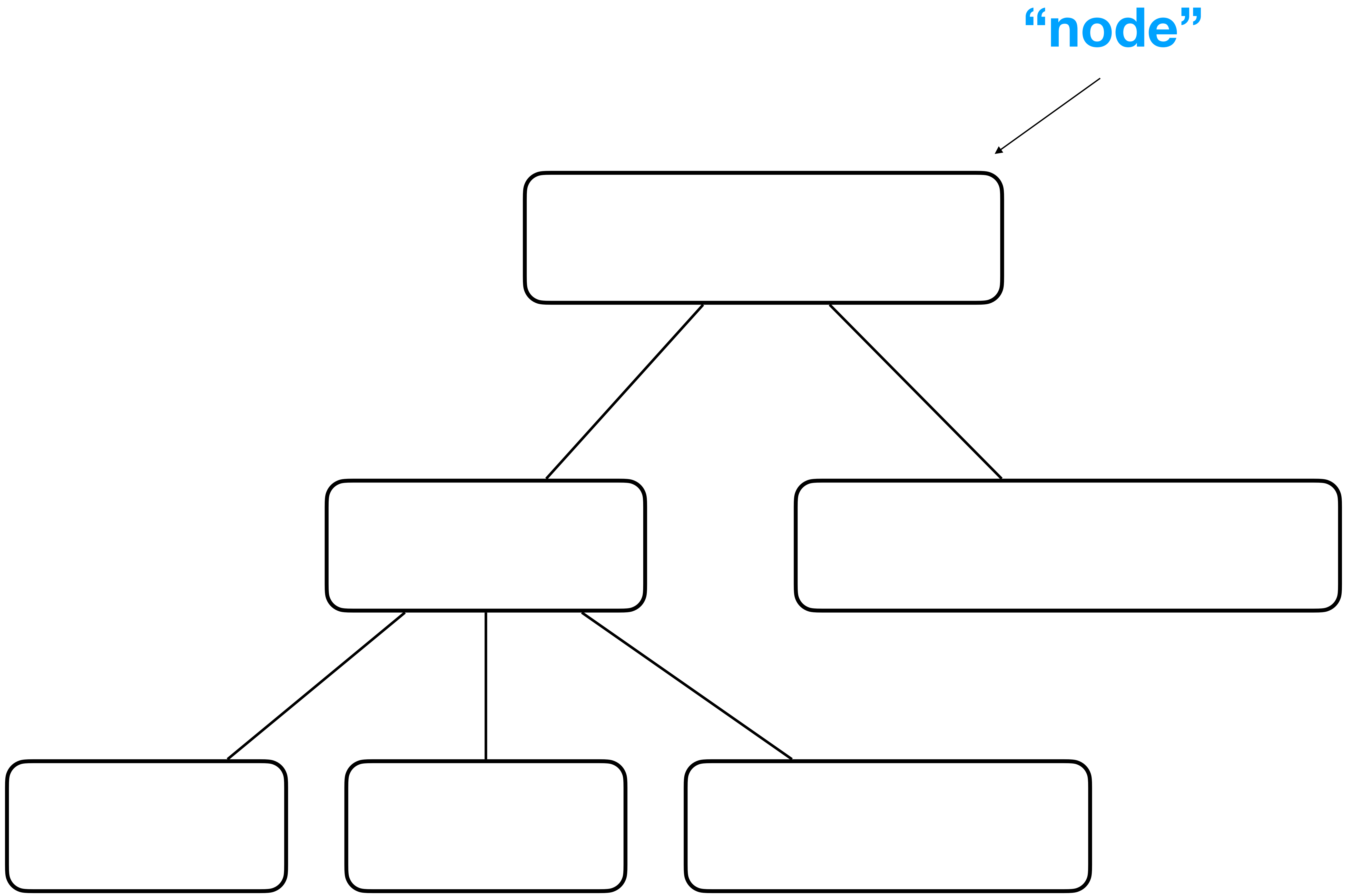
Tree Decompositions

Tree Decomposition

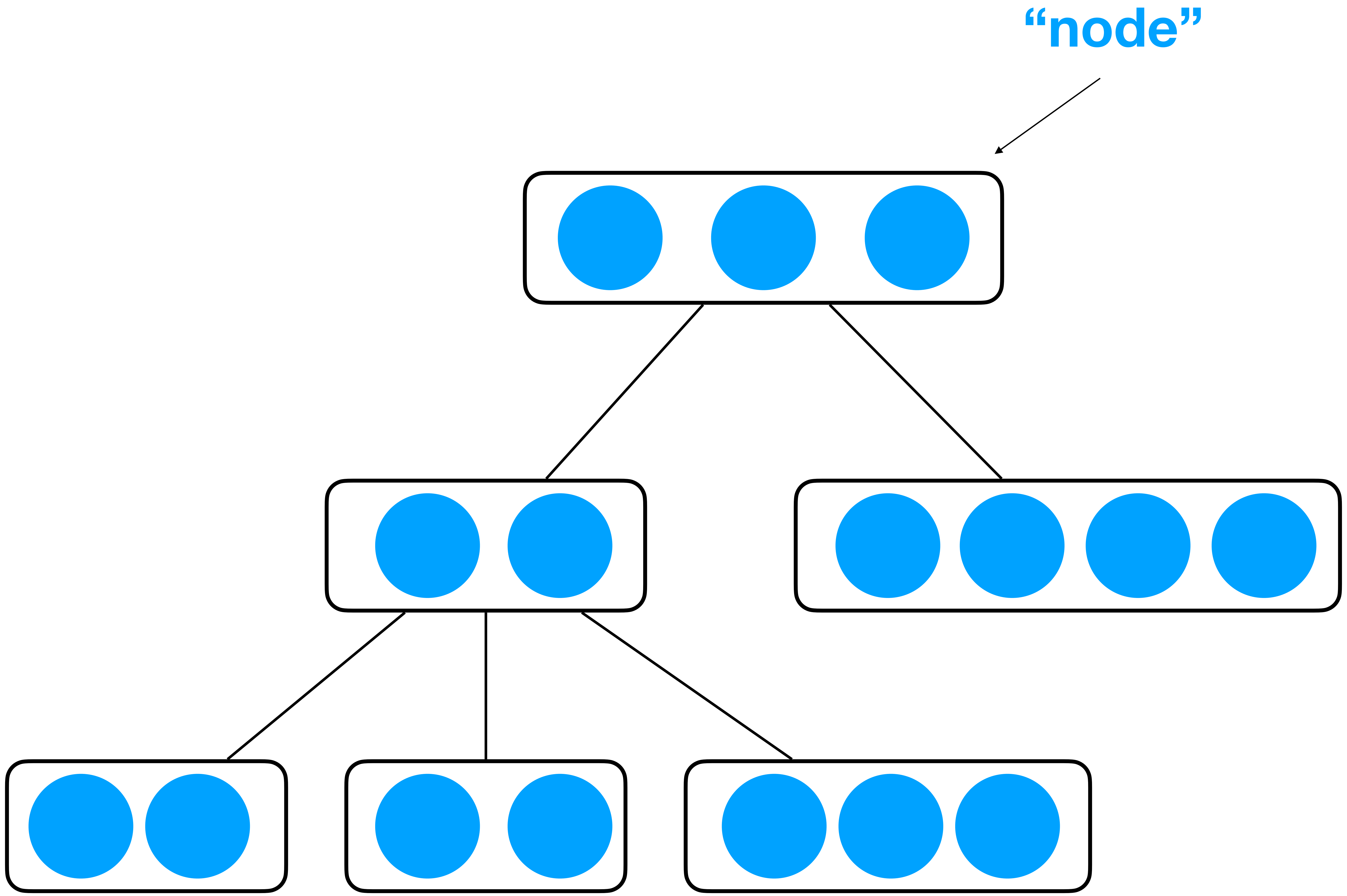
Tree Decomposition



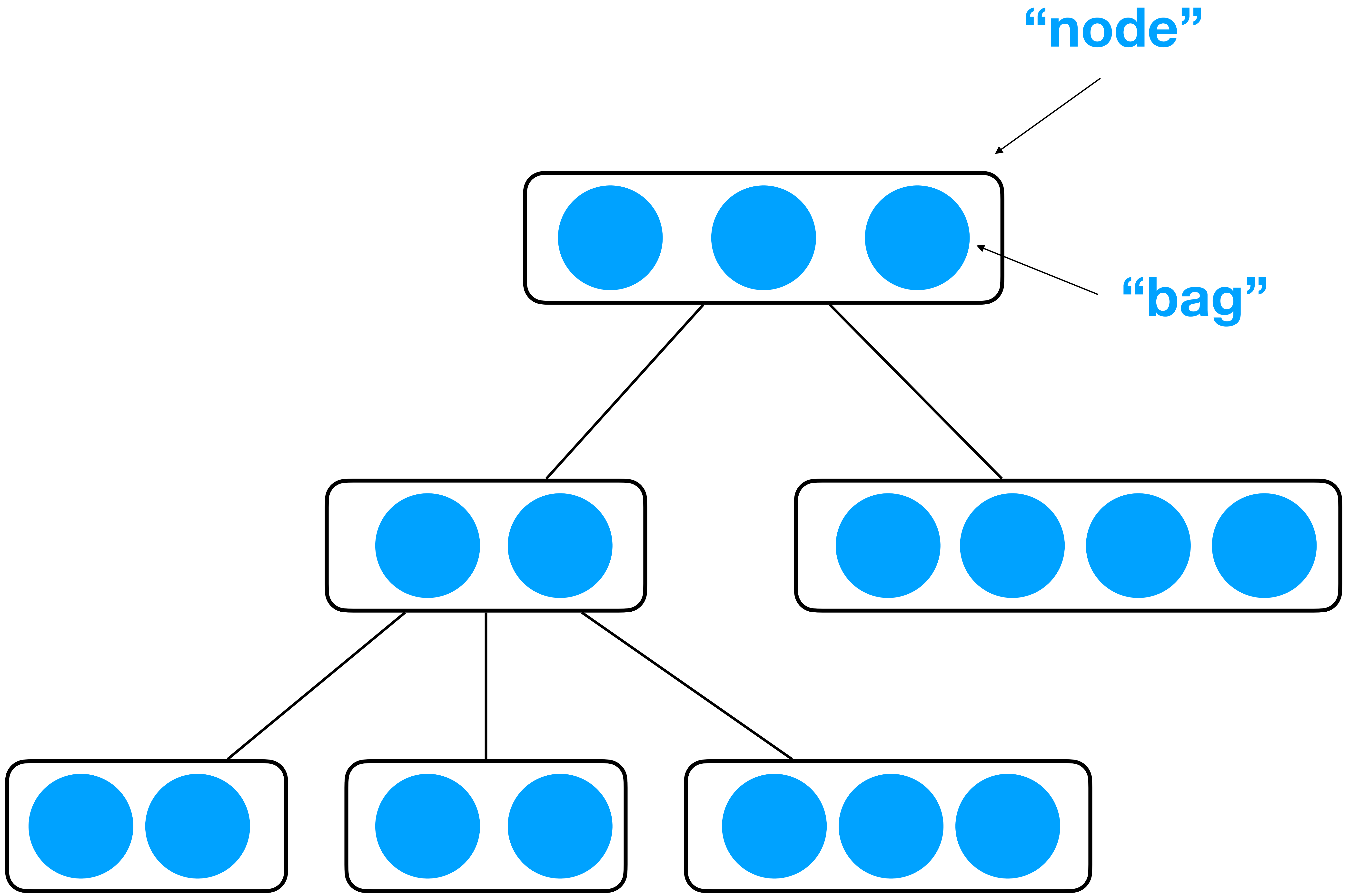
Tree Decomposition



Tree Decomposition

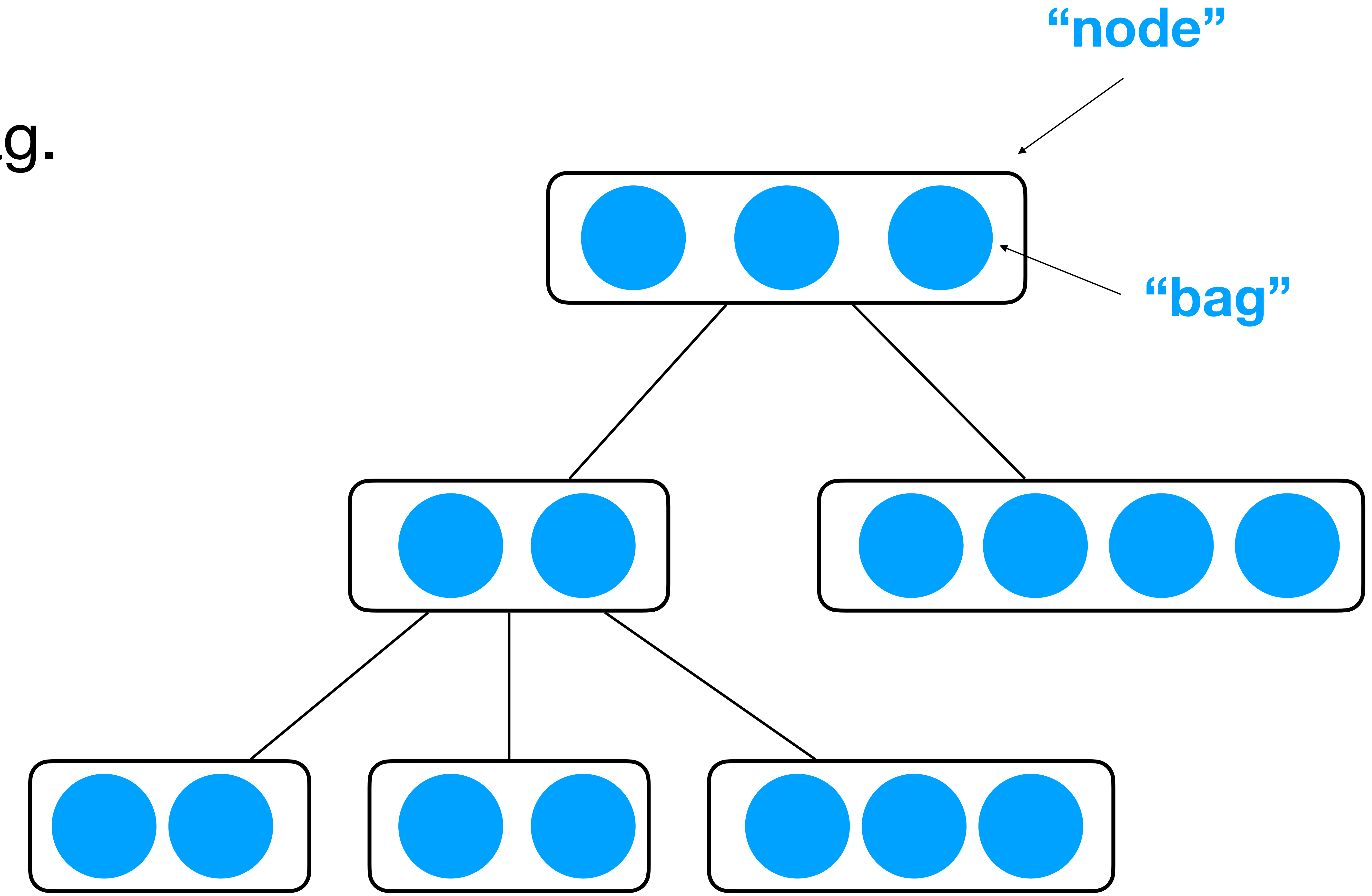


Tree Decomposition



Tree Decomposition

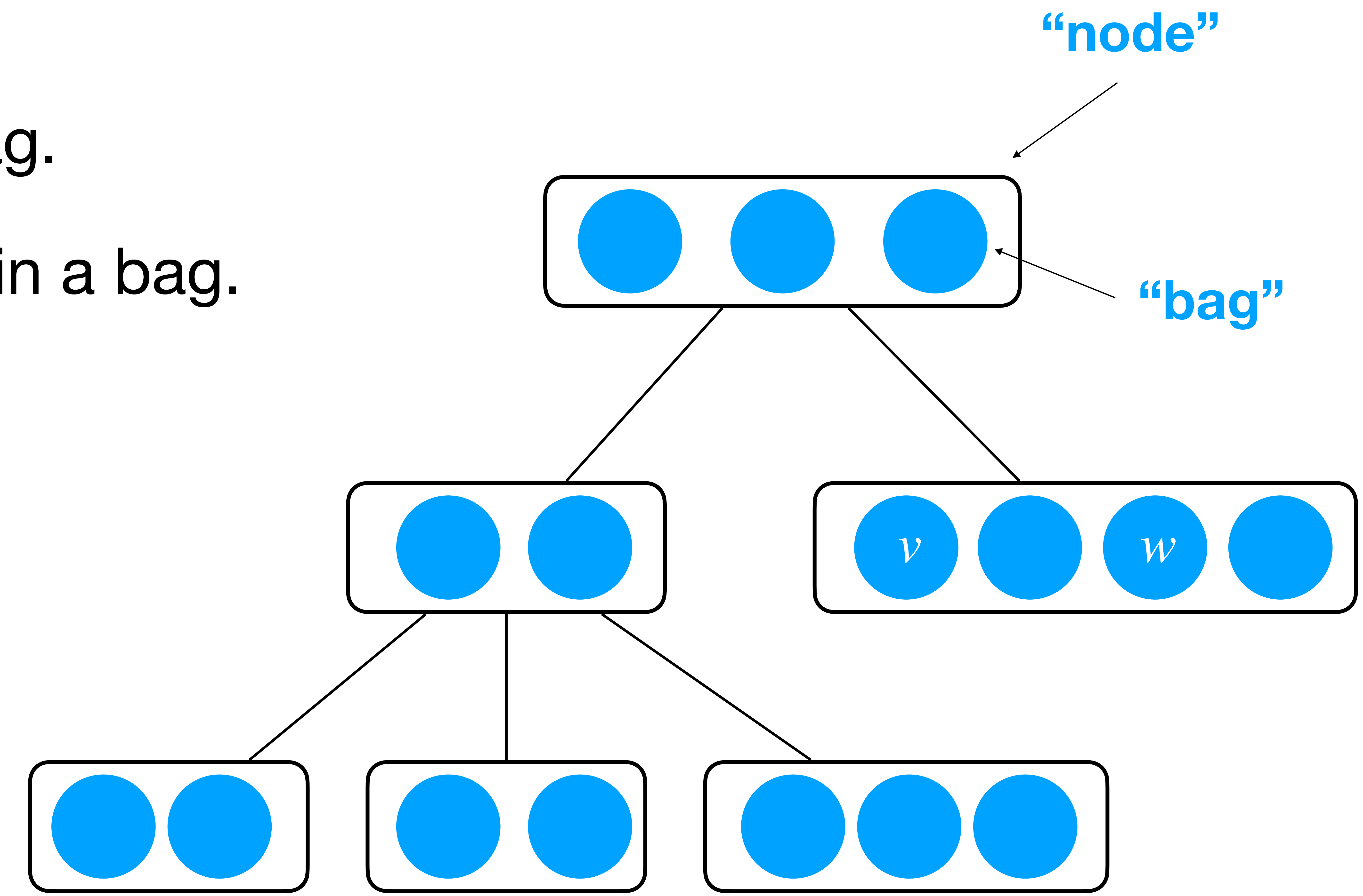
1. Each **vertex** appears in a bag.



Tree Decomposition

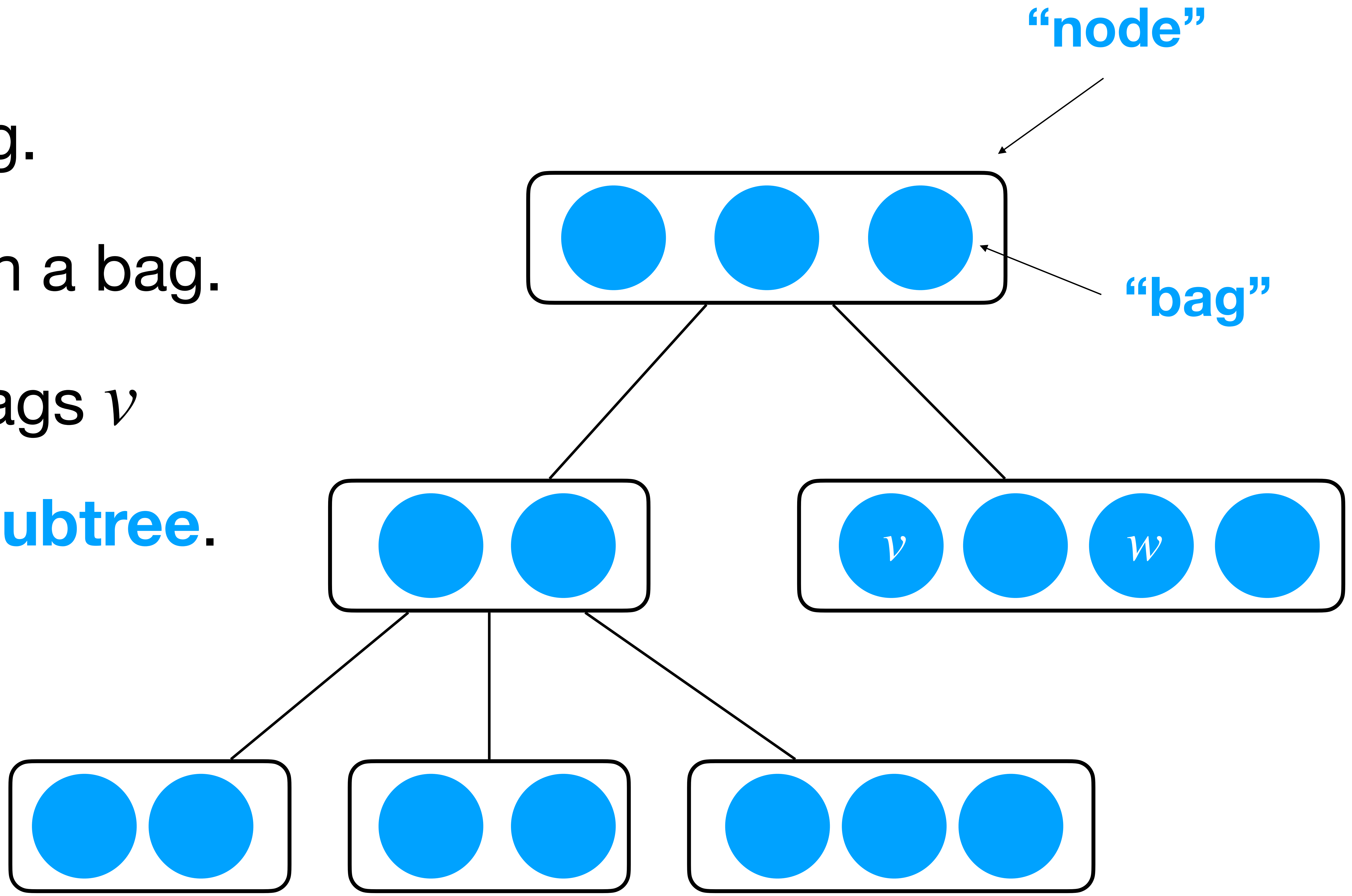
1. Each **vertex** appears in a bag.

2. Each **edge** vw is contained in a bag.



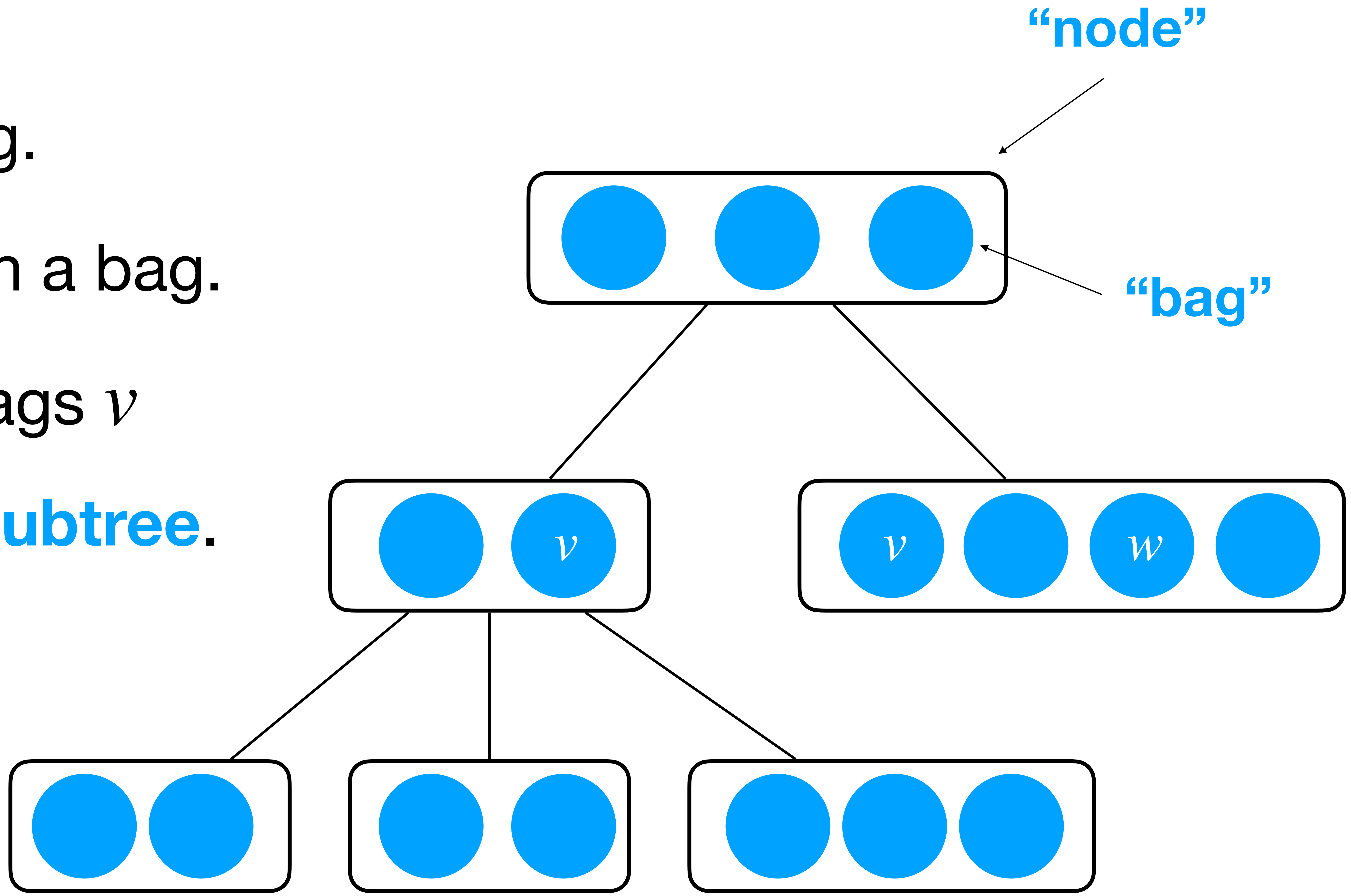
Tree Decomposition

1. Each **vertex** appears in a bag.
2. Each **edge** vw is contained in a bag.
3. The set of nodes in whose bags v appears form a **connected subtree**.



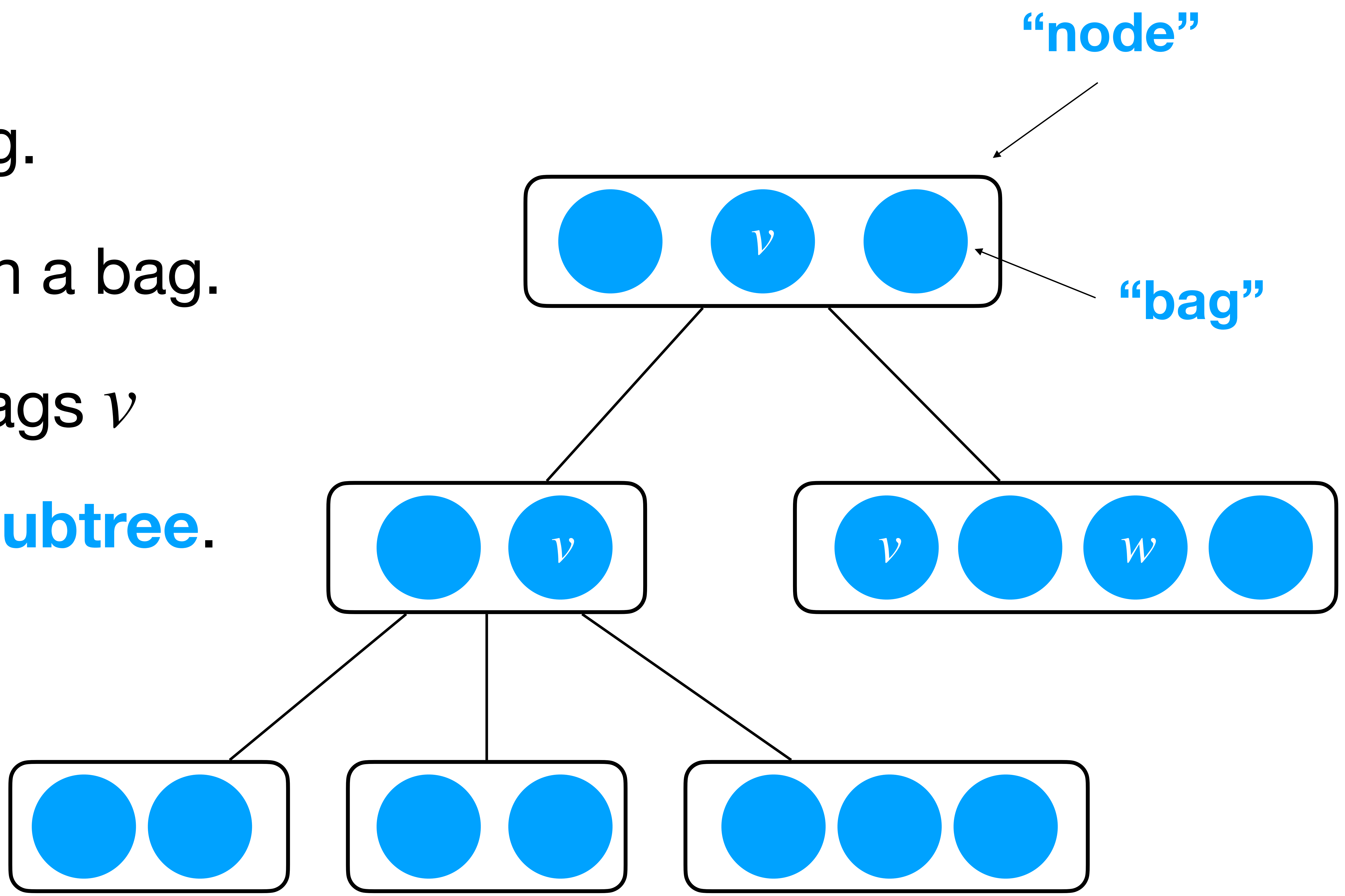
Tree Decomposition

1. Each **vertex** appears in a bag.
2. Each **edge** vw is contained in a bag.
3. The set of nodes in whose bags v appears form a **connected subtree**.



Tree Decomposition

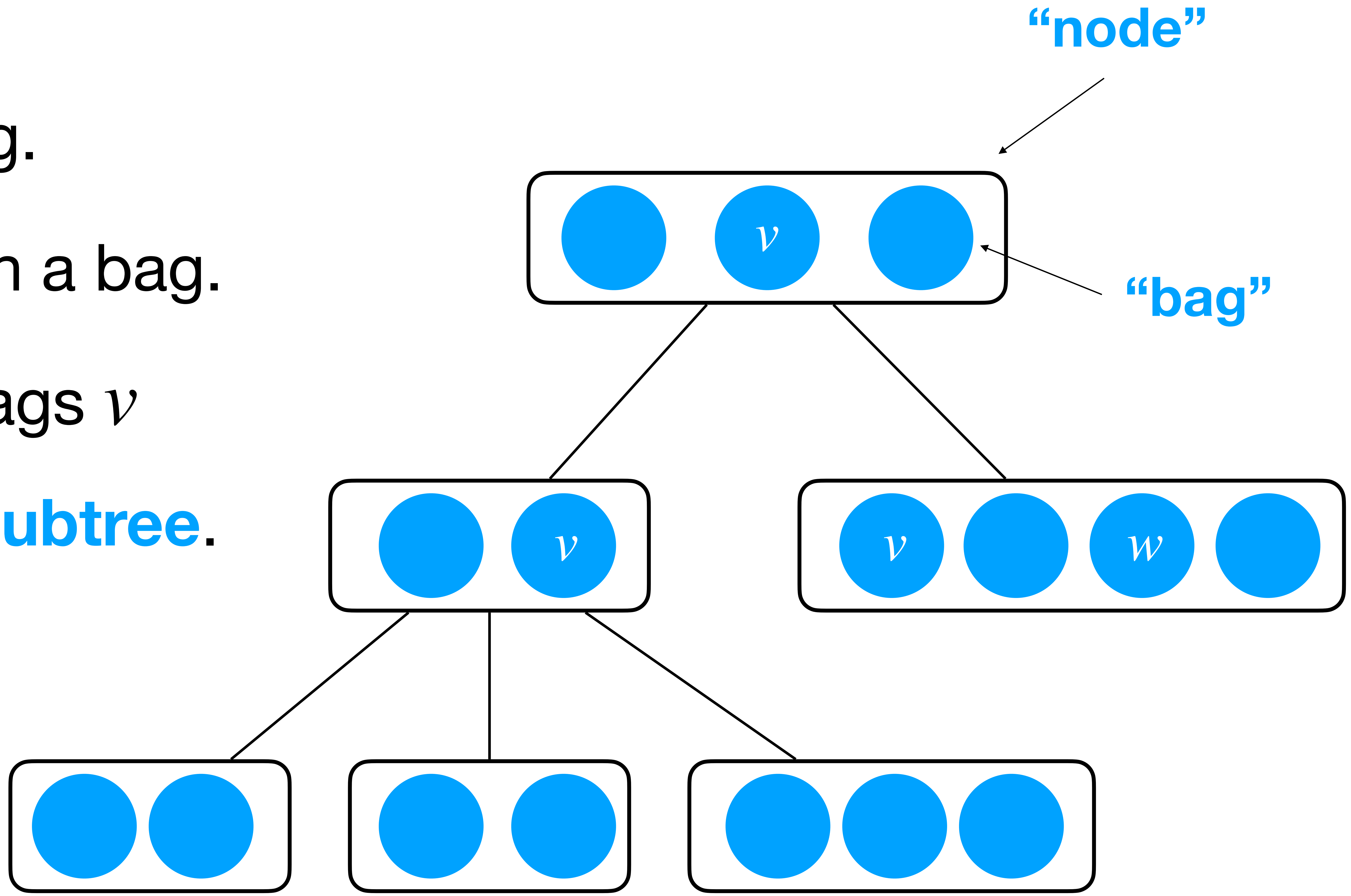
1. Each **vertex** appears in a bag.
2. Each **edge** vw is contained in a bag.
3. The set of nodes in whose bags v appears form a **connected subtree**.



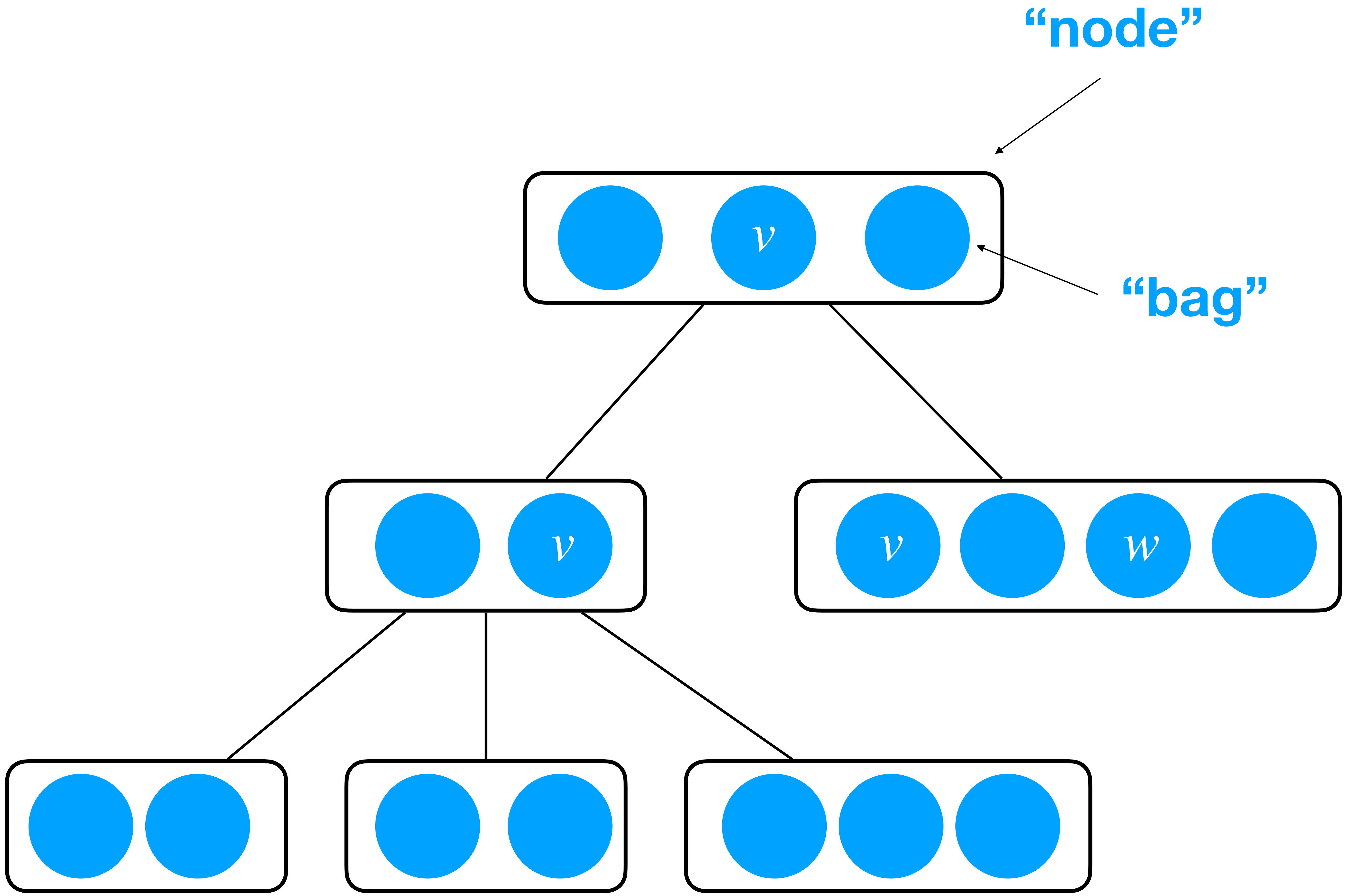
Tree Decomposition

1. Each **vertex** appears in a bag.
2. Each **edge** vw is contained in a bag.
3. The set of nodes in whose bags v appears form a **connected subtree**.

The **width** of a tree decomposition is the size of its largest bag - 1.



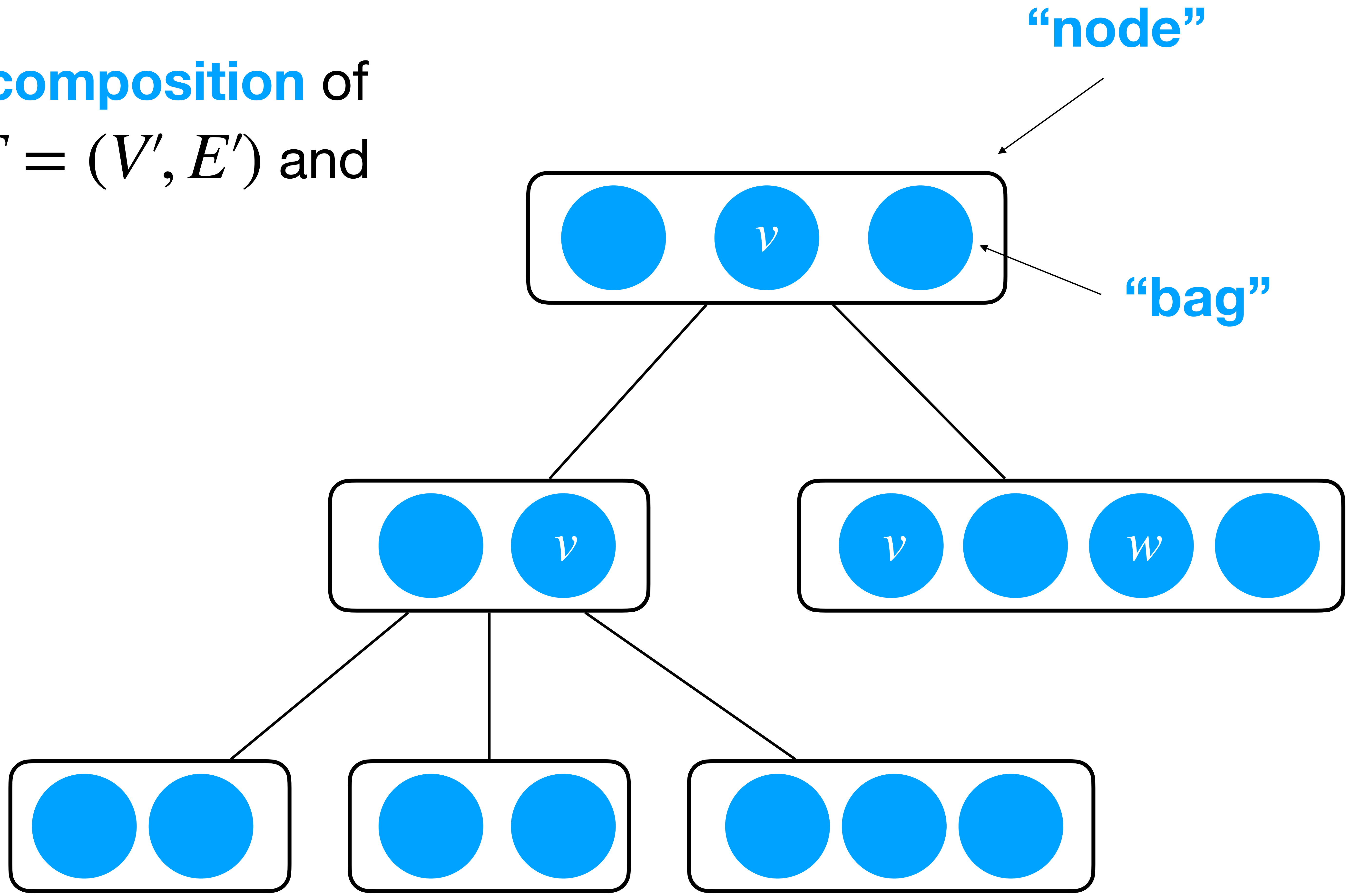
Tree Decomposition



Tree Decomposition

Definition

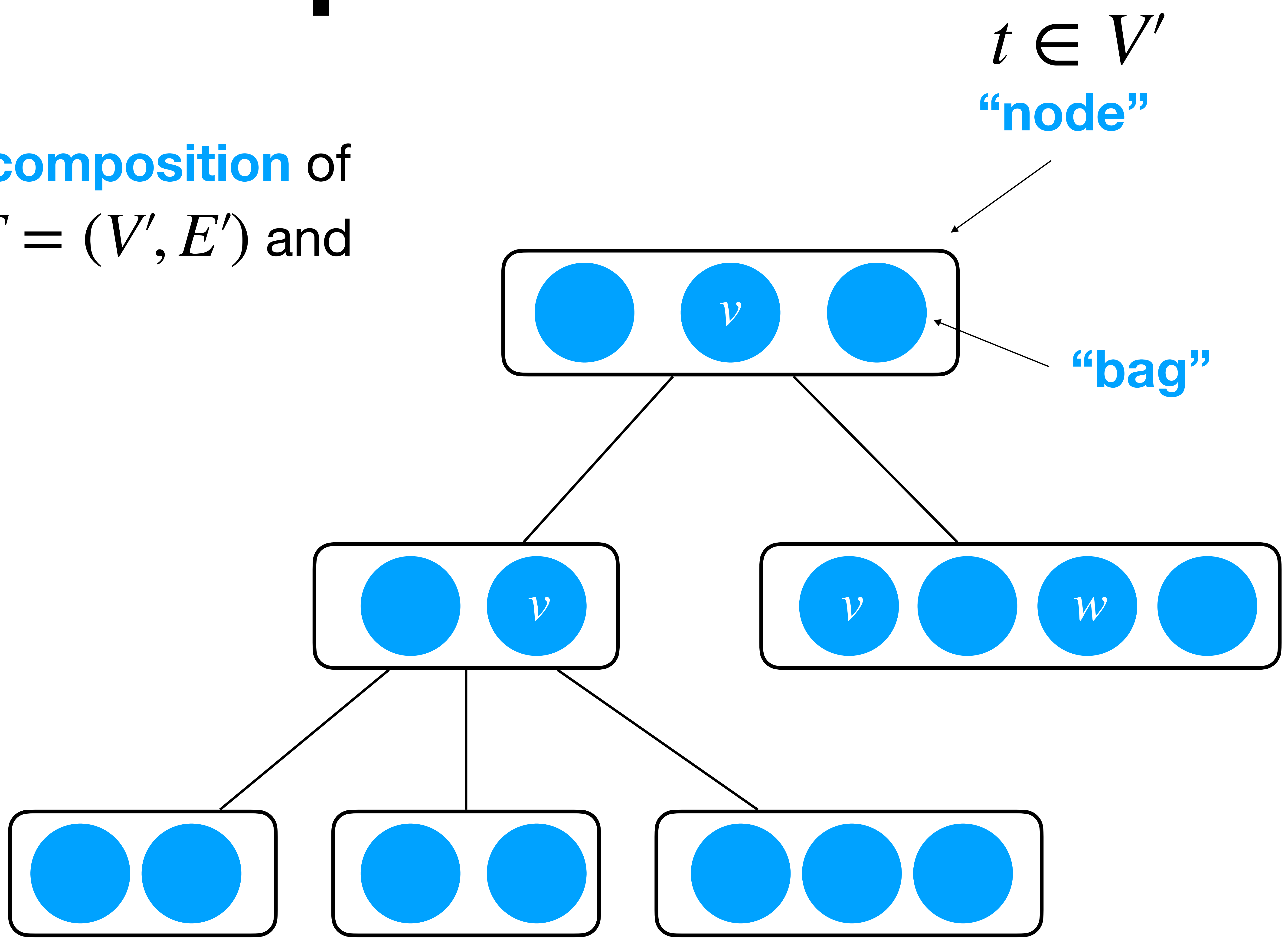
Let $G = (V, E)$ be a graph. A **tree decomposition** of G is a pair (T, χ) consisting of a tree $T = (V', E')$ and a mapping $\chi : V' \to 2^V$ such that



Tree Decomposition

Definition

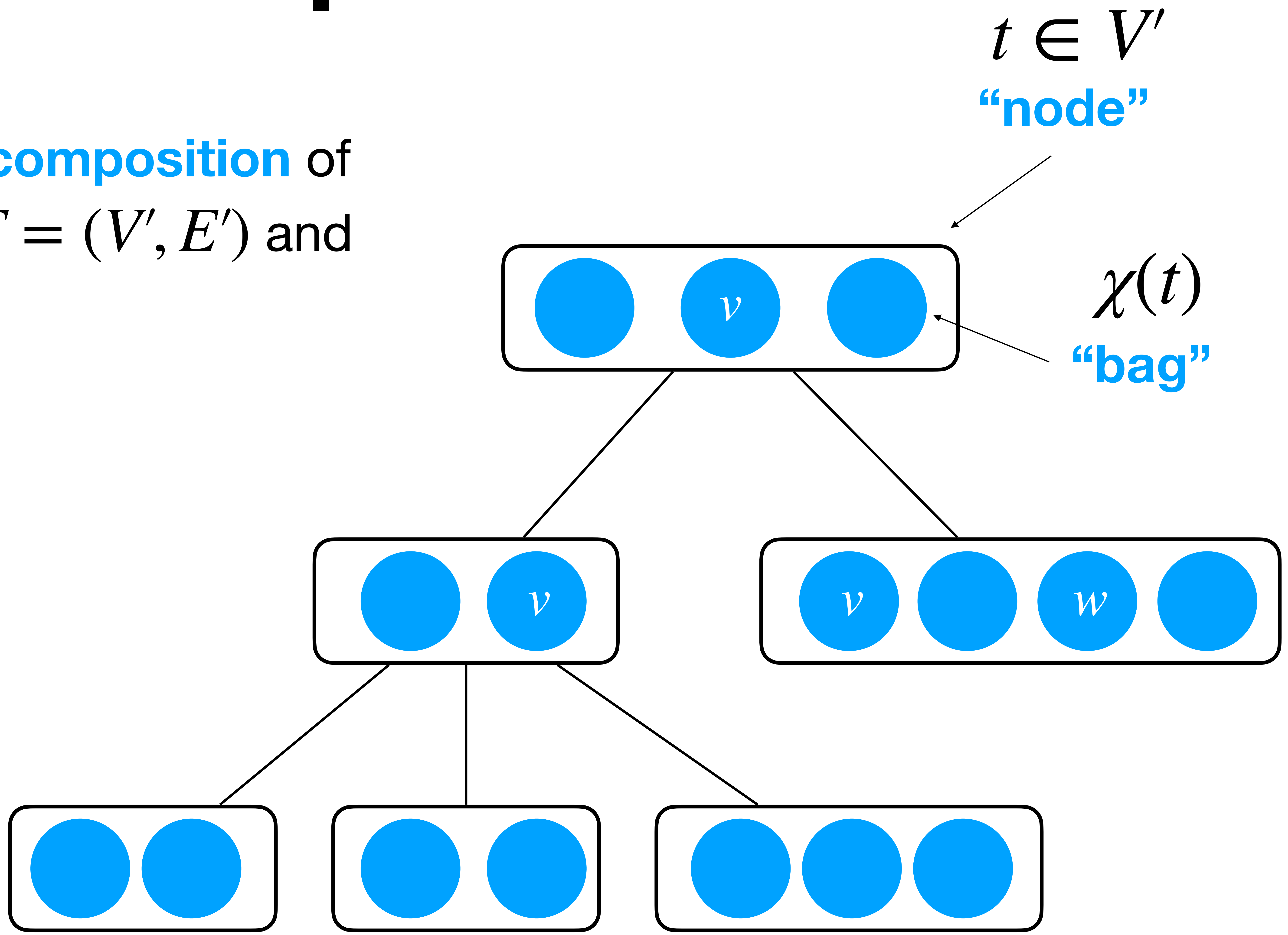
Let $G = (V, E)$ be a graph. A **tree decomposition** of G is a pair (T, χ) consisting of a tree $T = (V', E')$ and a mapping $\chi : V' \to 2^V$ such that



Tree Decomposition

Definition

Let $G = (V, E)$ be a graph. A **tree decomposition** of G is a pair (T, χ) consisting of a tree $T = (V', E')$ and a mapping $\chi : V' \to 2^V$ such that

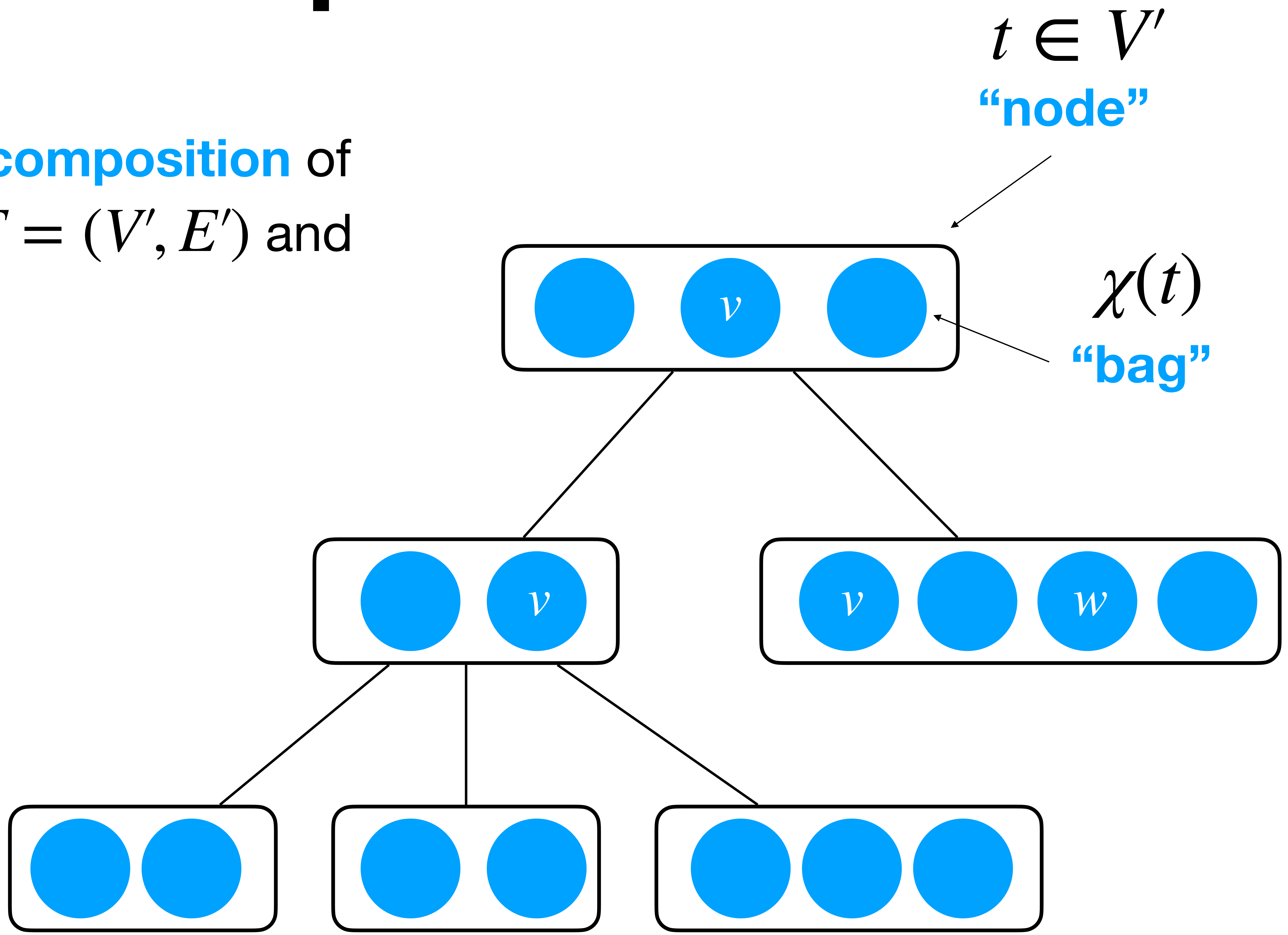


Tree Decomposition

Definition

Let $G = (V, E)$ be a graph. A **tree decomposition** of G is a pair (T, χ) consisting of a tree $T = (V', E')$ and a mapping $\chi : V' \to 2^V$ such that

1. $\bigcup_{t \in V'} \chi(t) = V$



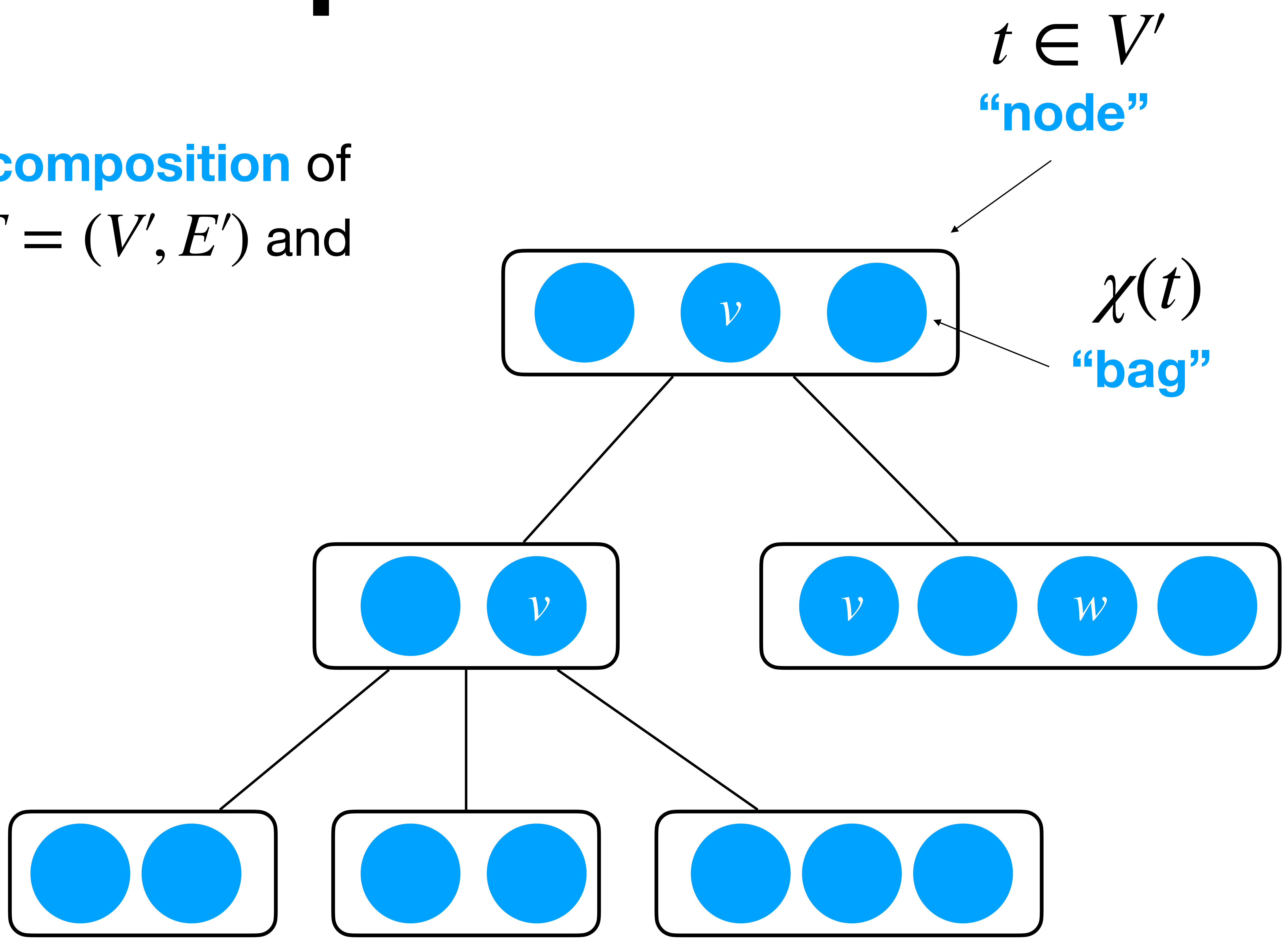
Tree Decomposition

Definition

Let $G = (V, E)$ be a graph. A **tree decomposition** of G is a pair (T, χ) consisting of a tree $T = (V', E')$ and a mapping $\chi : V' \to 2^V$ such that

1. $\bigcup_{t \in V'} \chi(t) = V$

2. For each $vw \in E$ there is a $t \in V'$ with $v, w \in \chi(t)$.

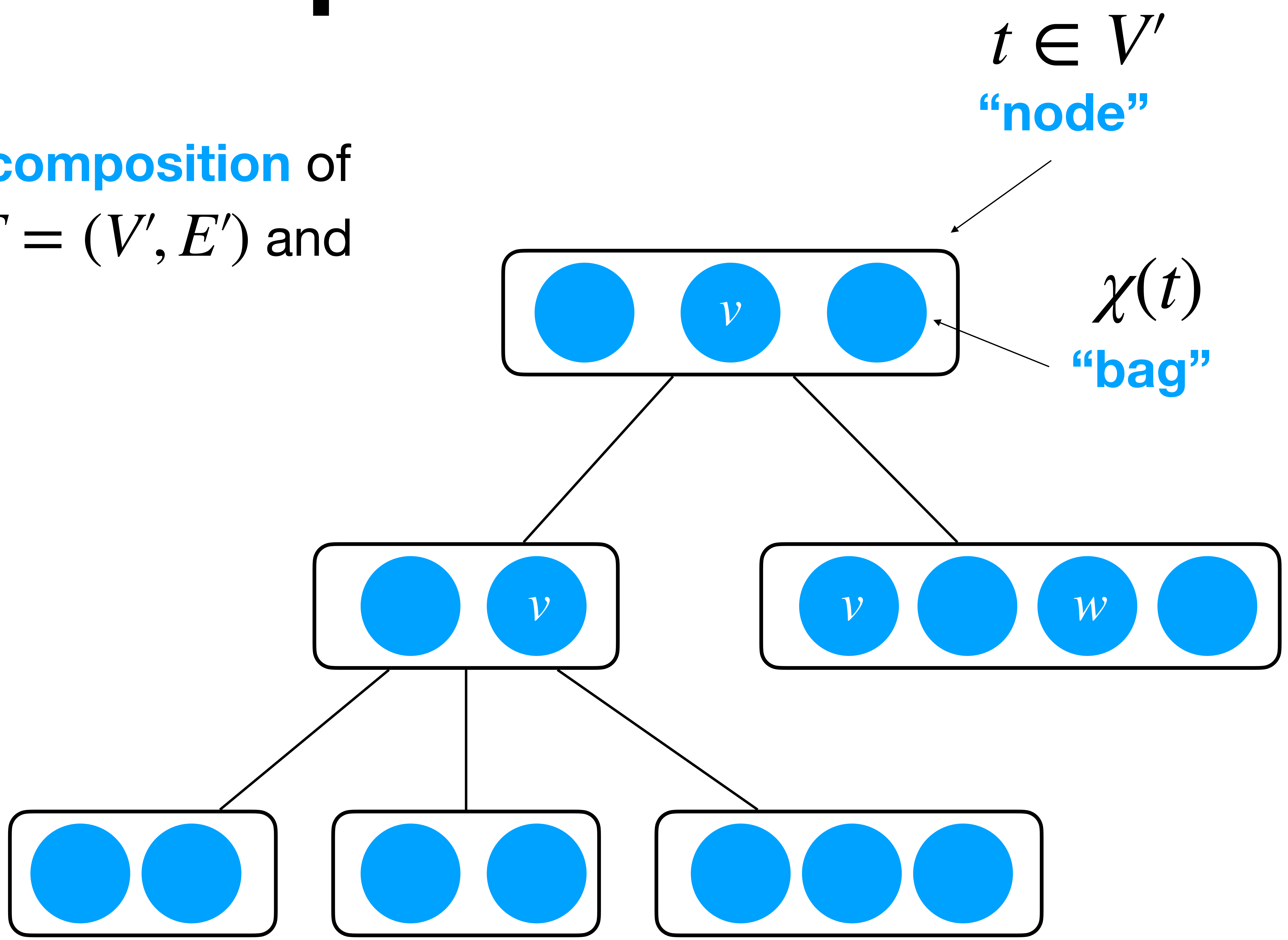


Tree Decomposition

Definition

Let $G = (V, E)$ be a graph. A **tree decomposition** of G is a pair (T, χ) consisting of a tree $T = (V', E')$ and a mapping $\chi : V' \to 2^V$ such that

1. $\bigcup_{t \in V'} \chi(t) = V$
2. For each $vw \in E$ there is a $t \in V'$ with $v, w \in \chi(t)$.
3. $T[\{t \in V' \mid v \in \chi(t)\}]$ is a connected subtree for each $v \in V$.



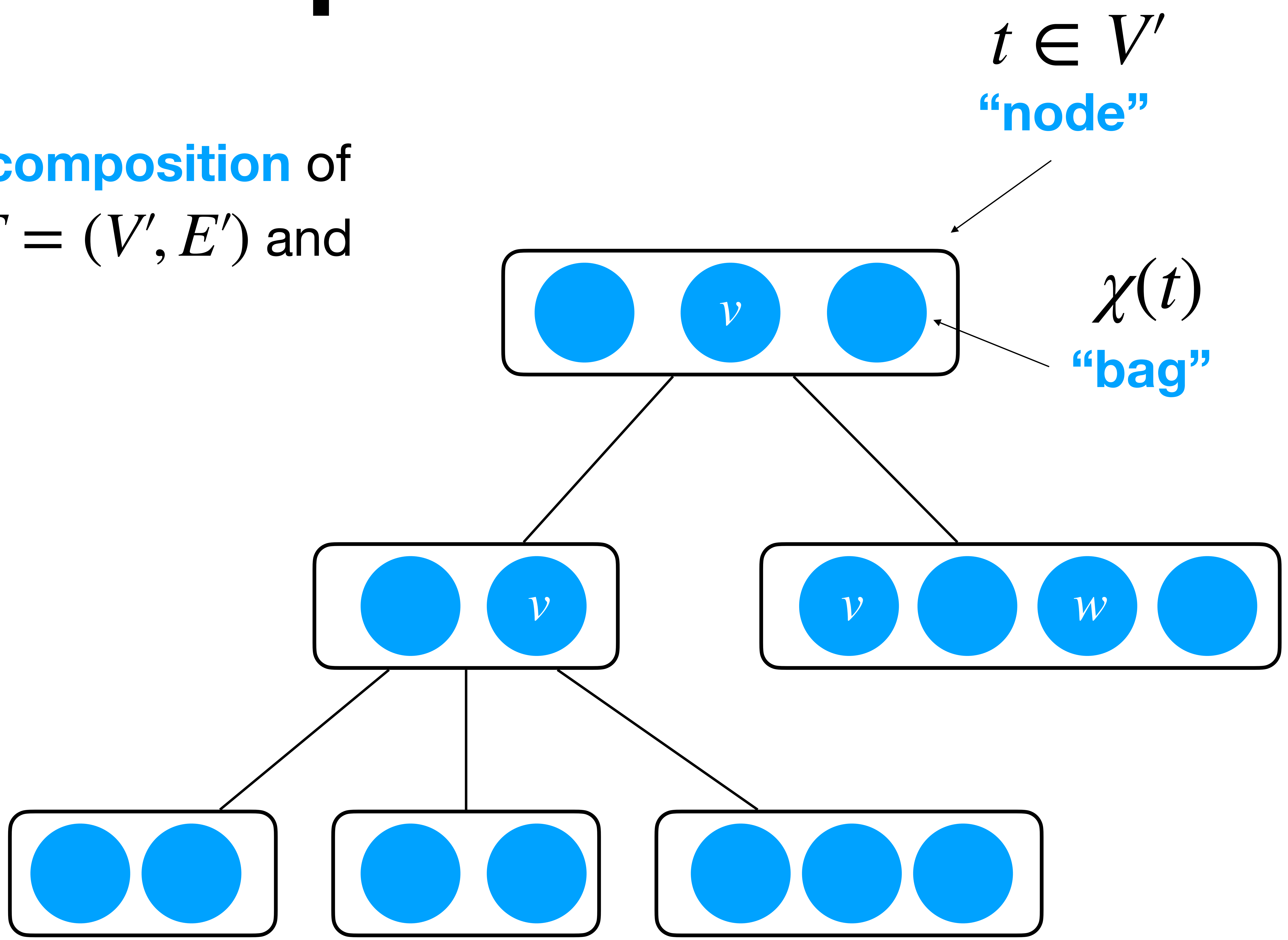
Tree Decomposition

Definition

Let $G = (V, E)$ be a graph. A **tree decomposition** of G is a pair (T, χ) consisting of a tree $T = (V', E')$ and a mapping $\chi : V' \to 2^V$ such that

1. $\bigcup_{t \in V'} \chi(t) = V$
2. For each $vw \in E$ there is a $t \in V'$ with $v, w \in \chi(t)$.
3. $T[\{t \in V' \mid v \in \chi(t)\}]$ is a connected subtree for each $v \in V$.

The **width** of (T, χ) is $\max_{t \in V'} |\chi(t)| - 1$.



Tree Decompositions and Treewidth

Tree Decompositions and Treewidth

Fact

A graph has a tree decomposition of width k if, and only if, it has an elimination ordering of width k.

Tree Decompositions and Treewidth

Fact

A graph has a tree decomposition of width k if, and only if, it has an elimination ordering of width k.

Tree Decompositions and Treewidth

Fact

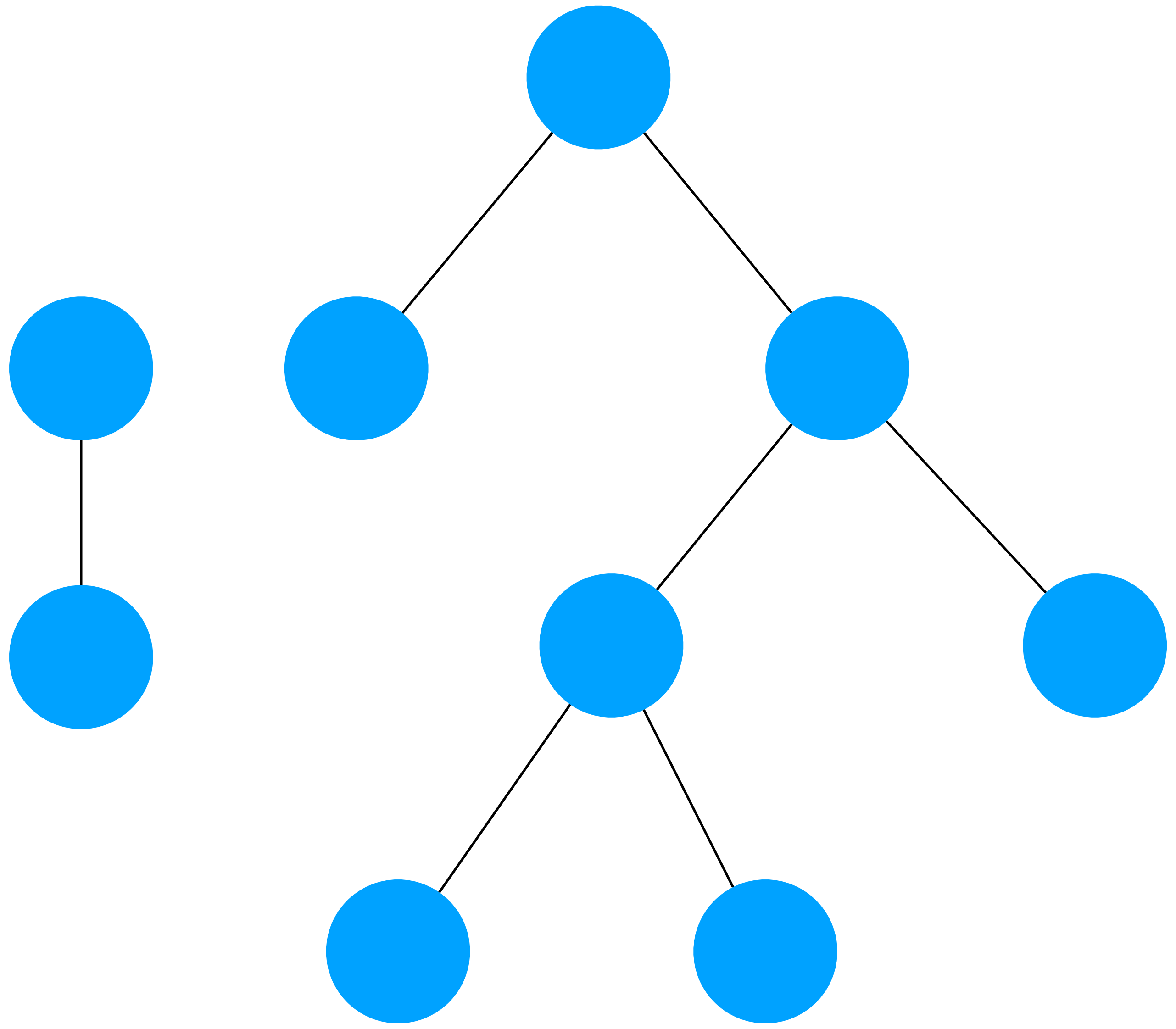
A graph has a tree decomposition of width k if, and only if, it has an elimination ordering of width k.

The treewidth of a graph G is the minimum width of a tree decomposition of G.

Examples

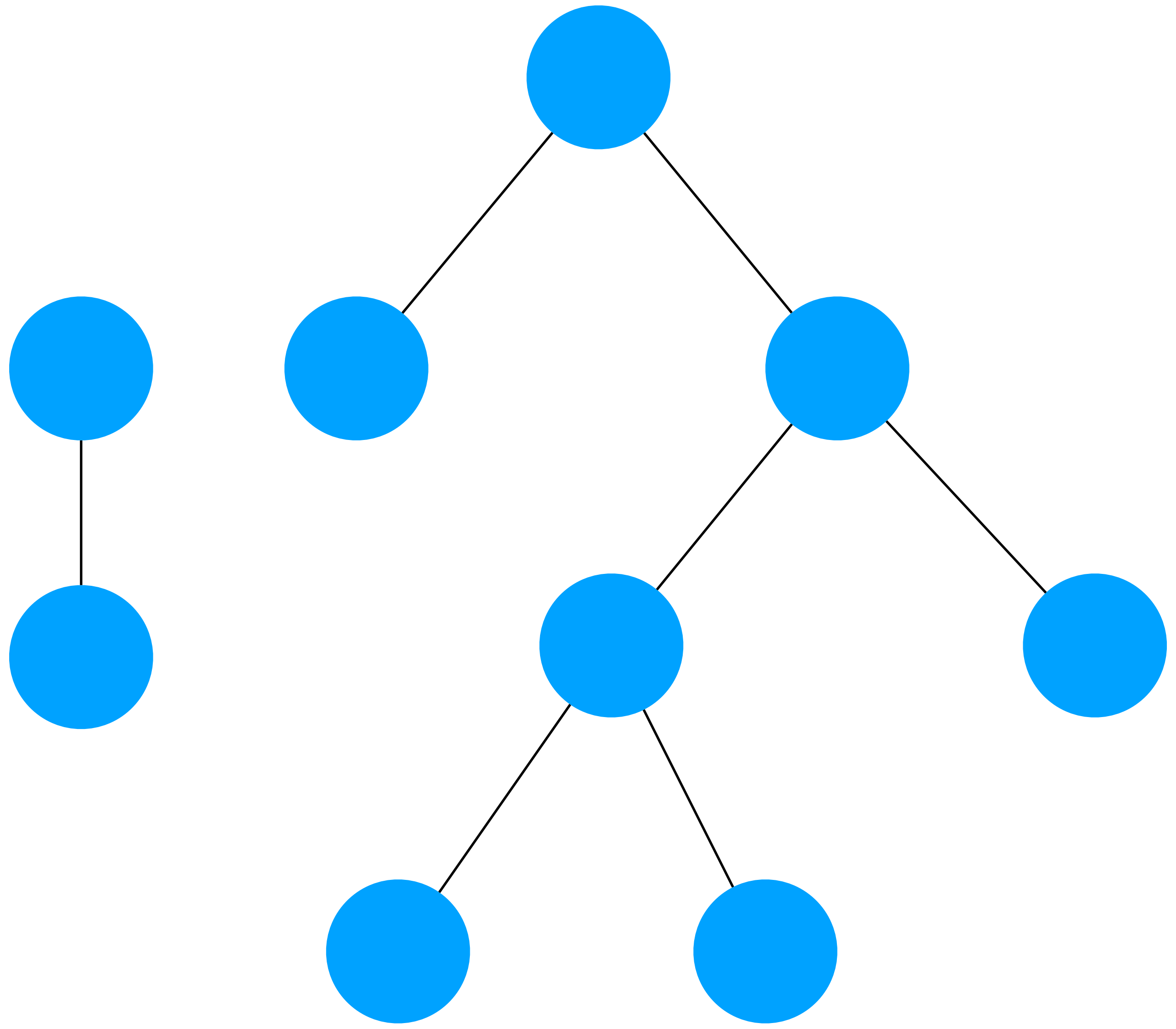
Examples

Forests



Examples

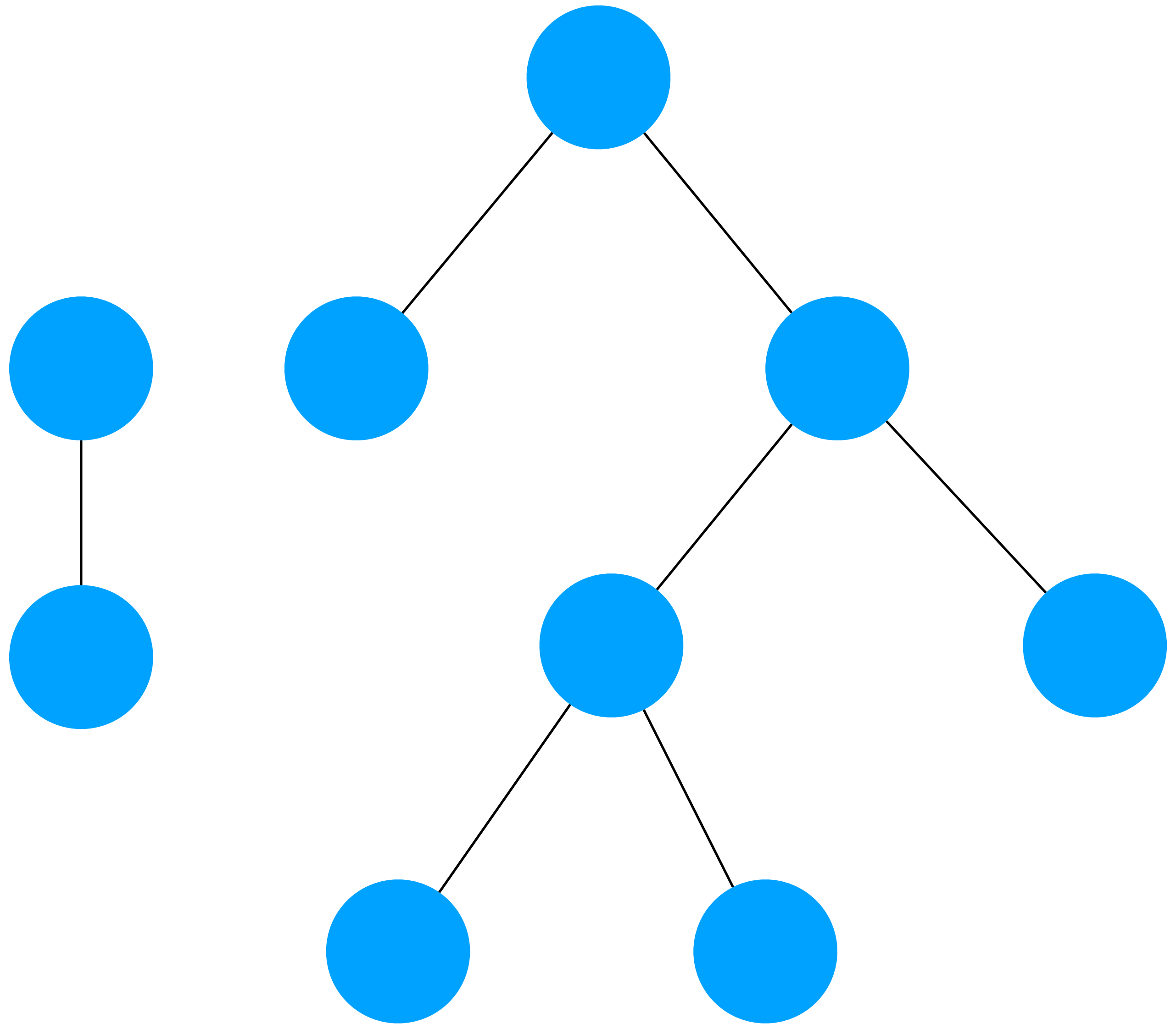
Forests



treewidth 1

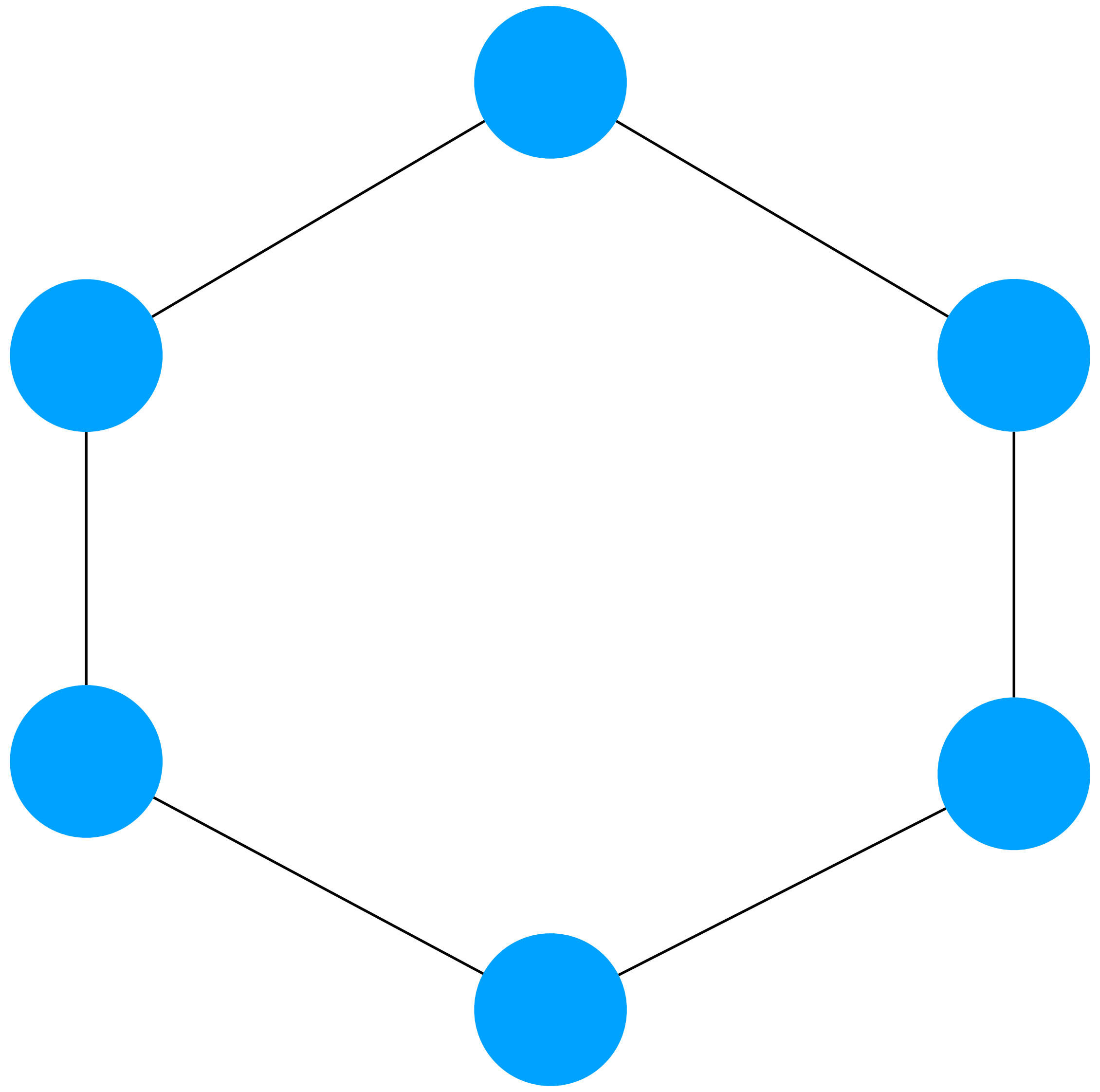
Examples

Forests



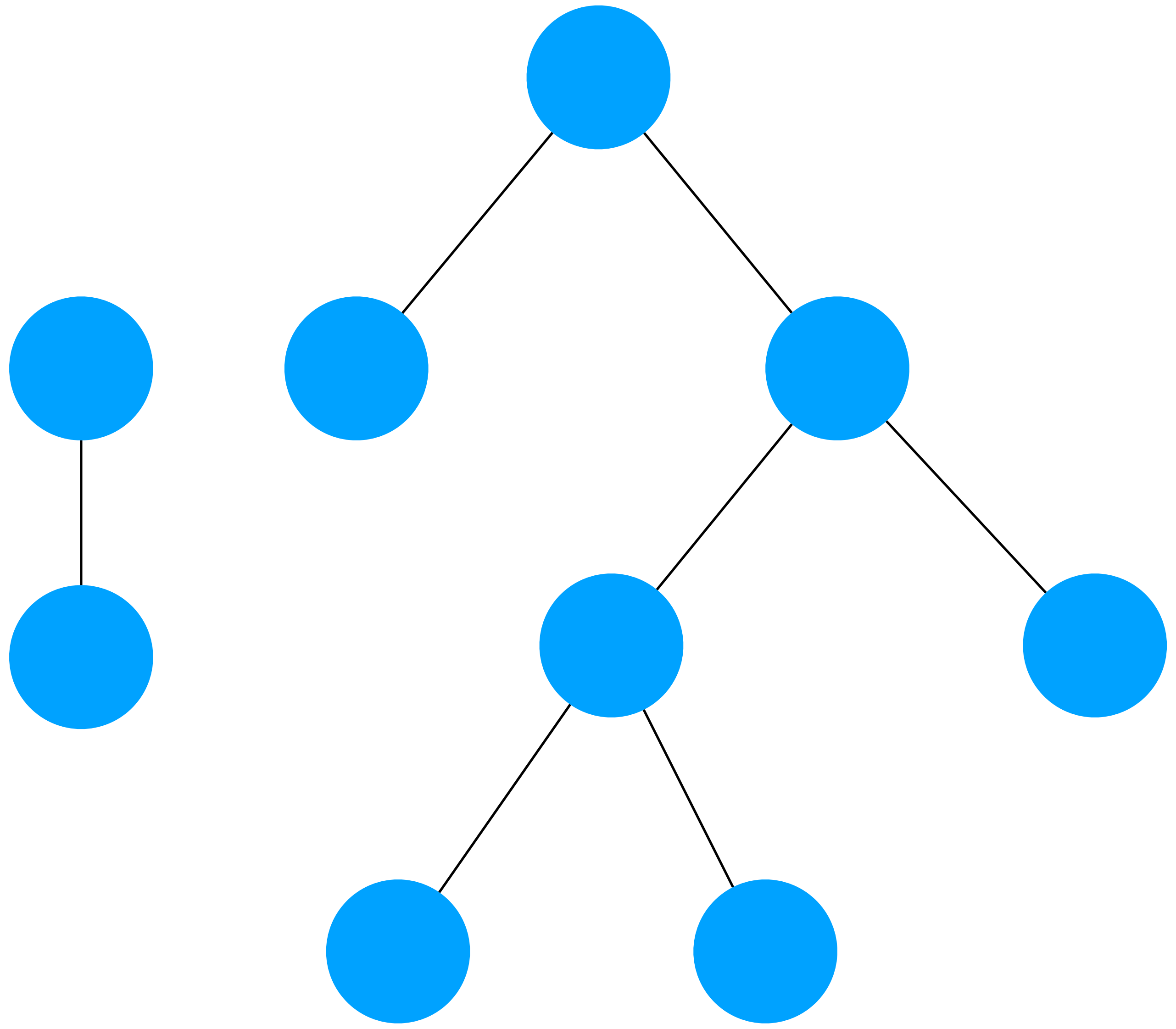
treewidth 1

Cycles



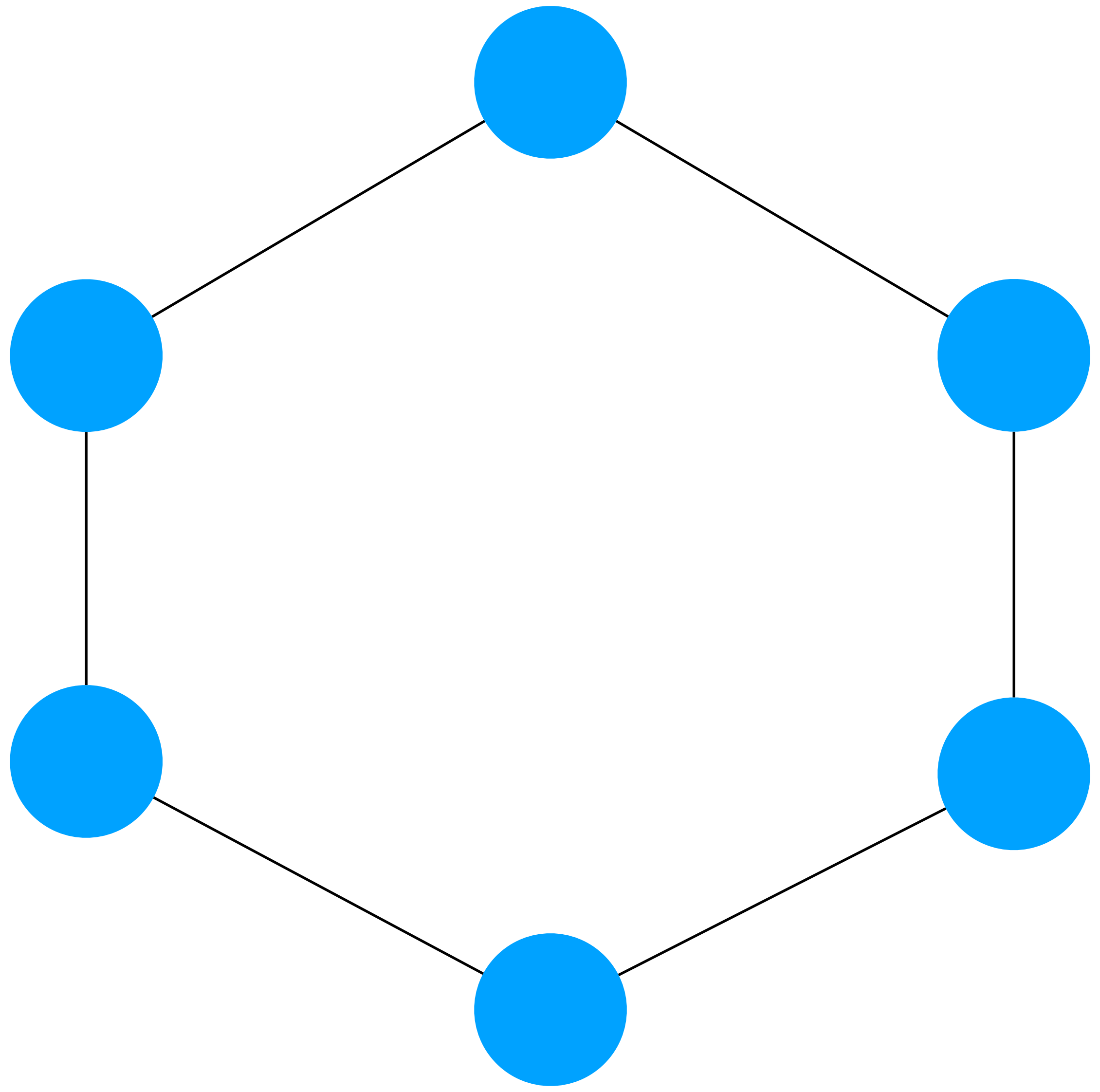
Examples

Forests



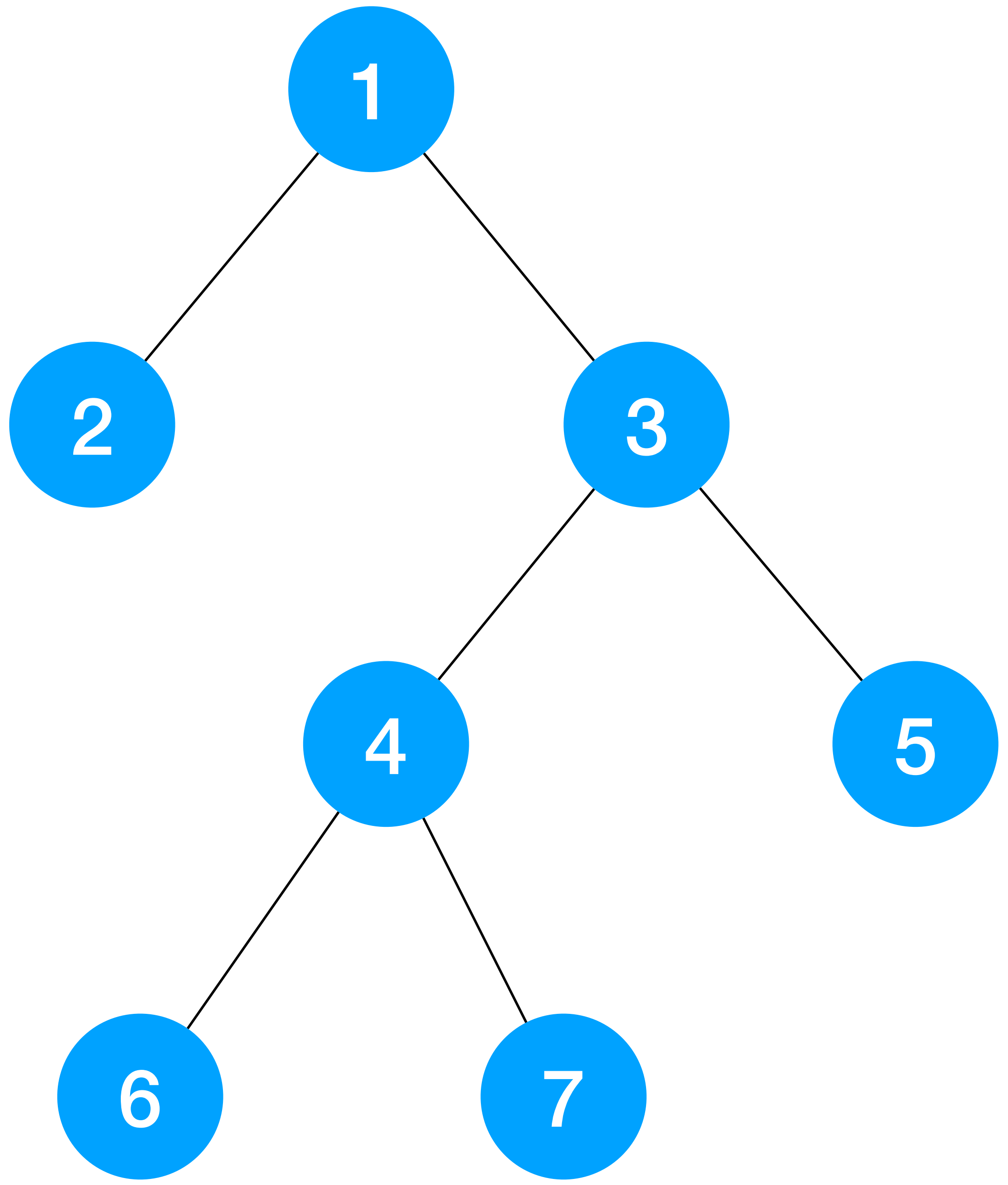
treewidth 1

Cycles

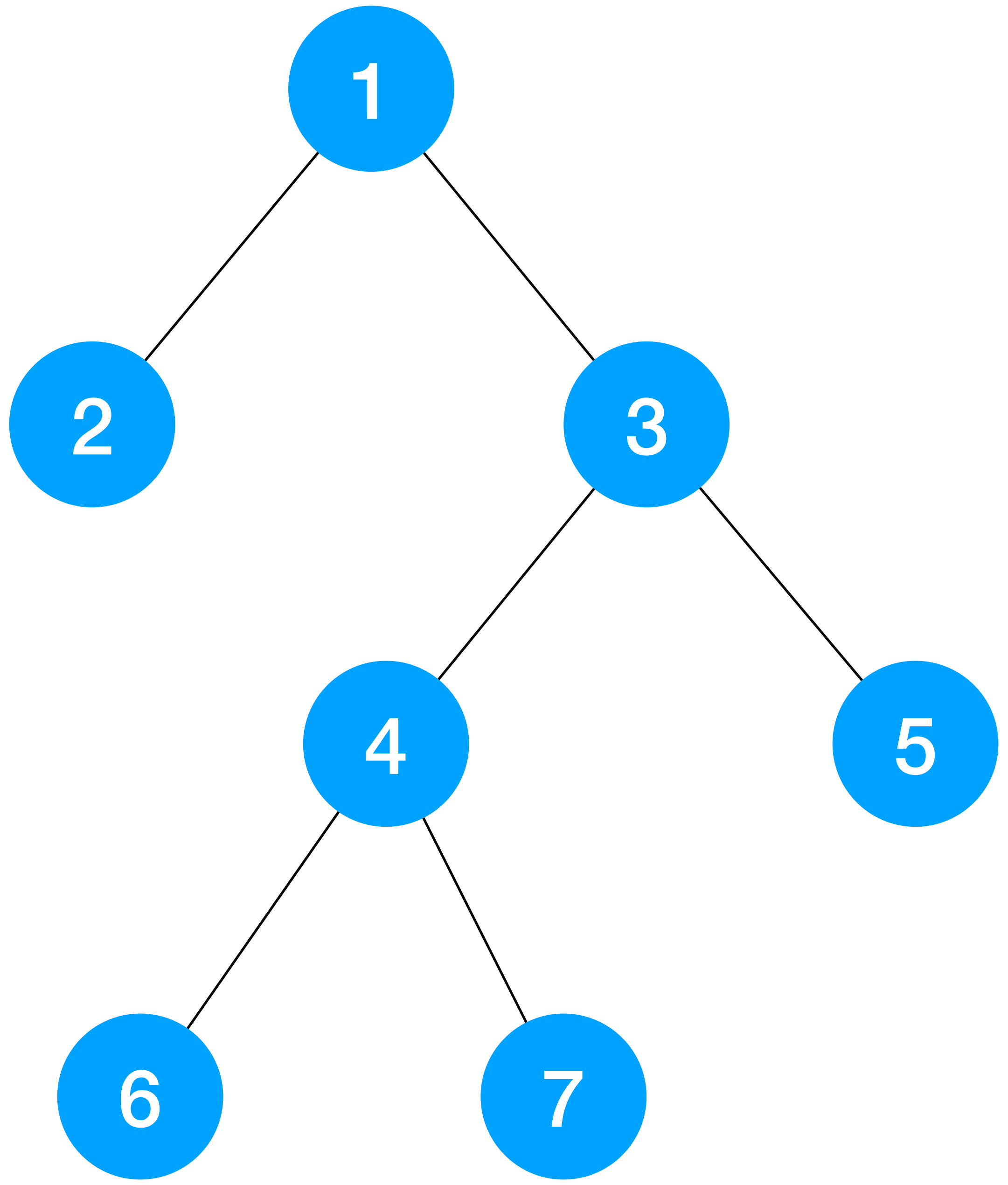


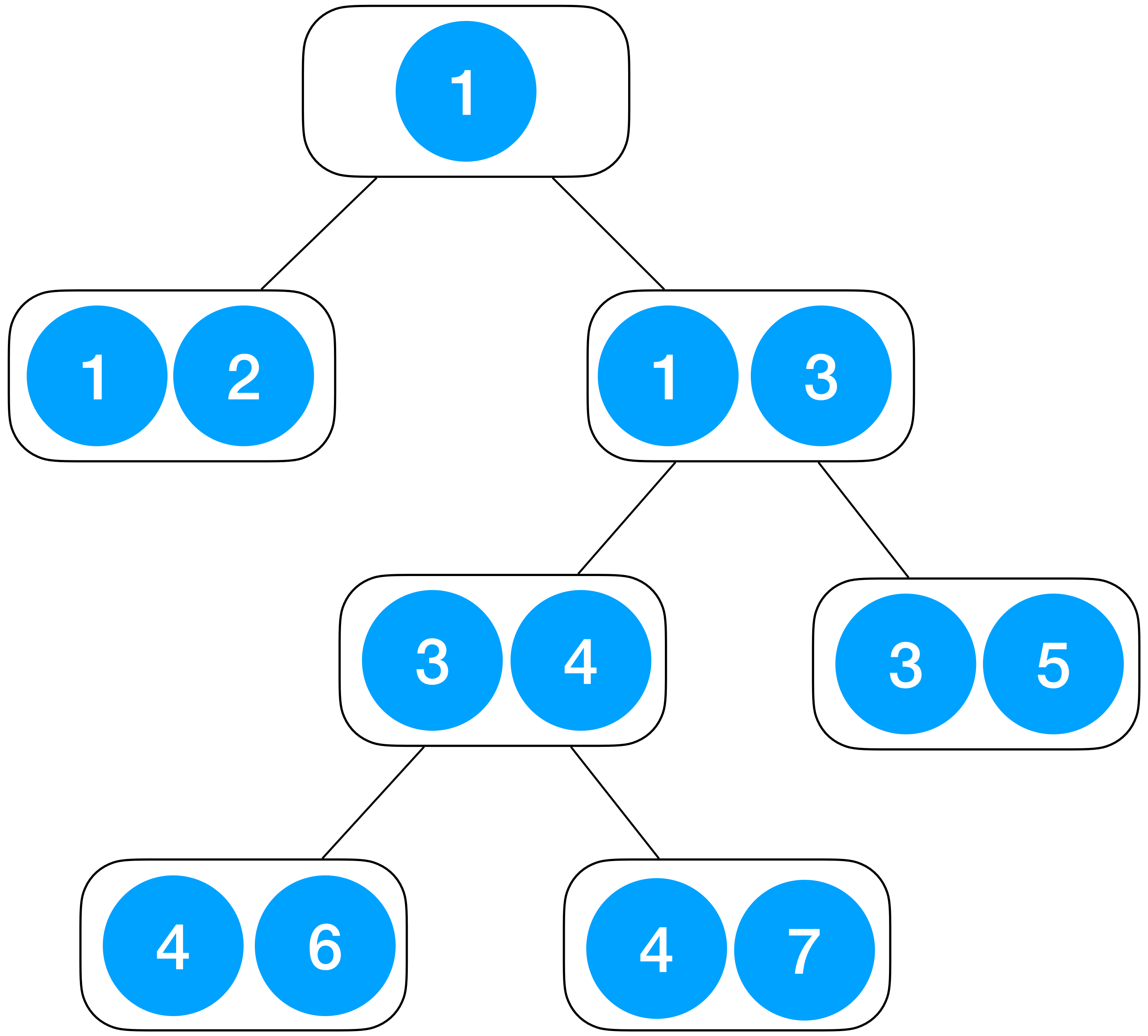
treewidth 2

Trees

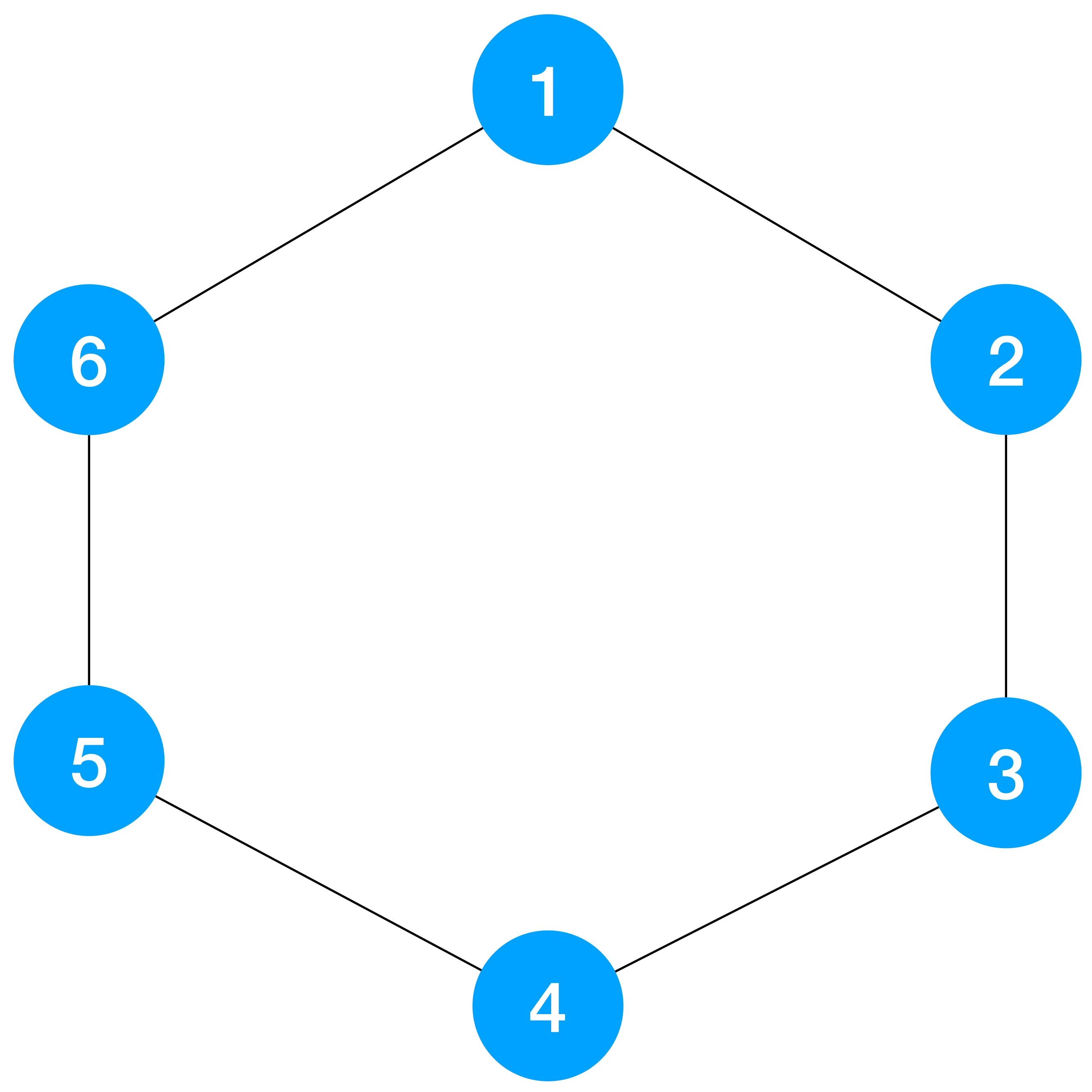


Trees

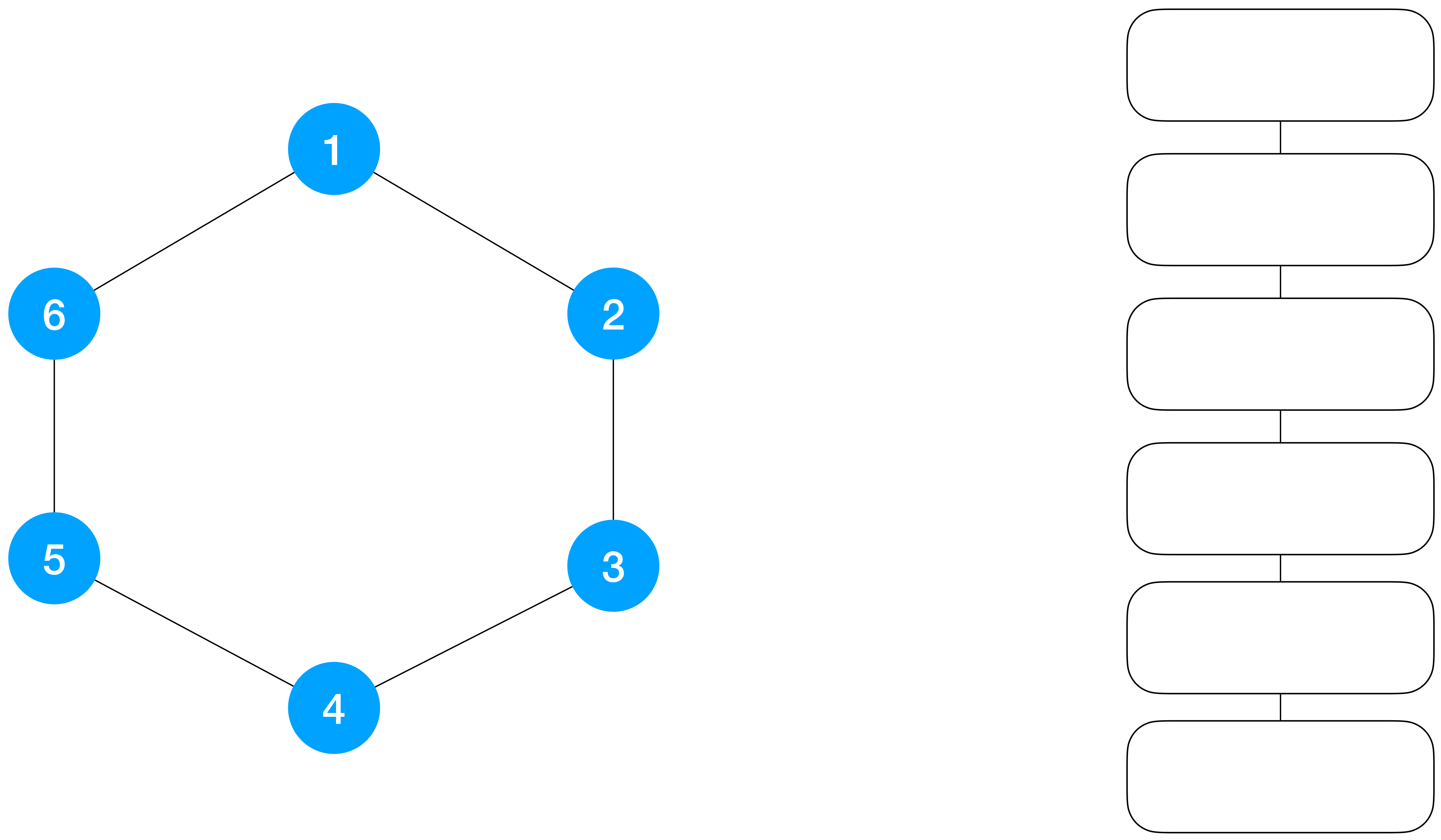




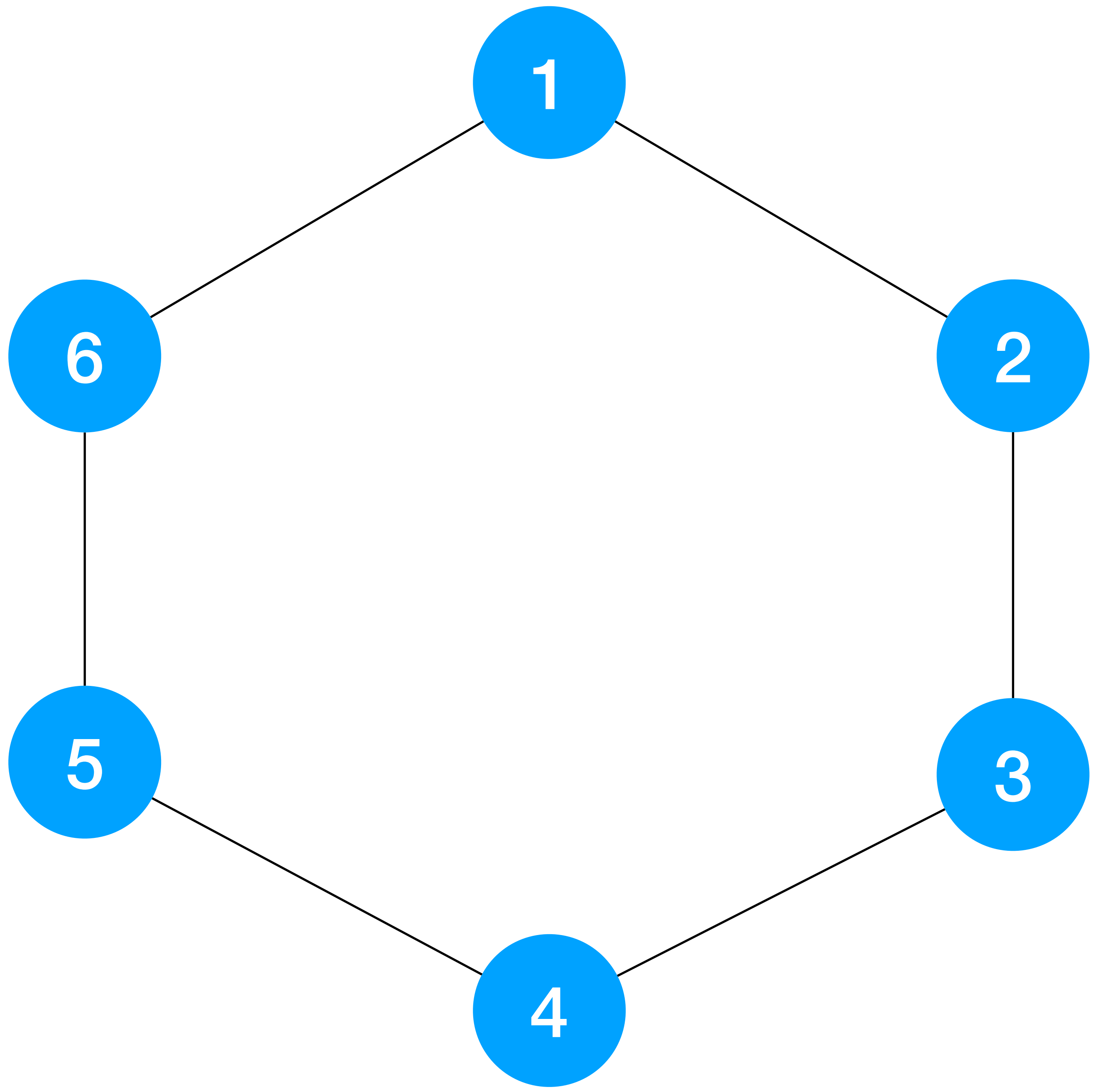
Cycles

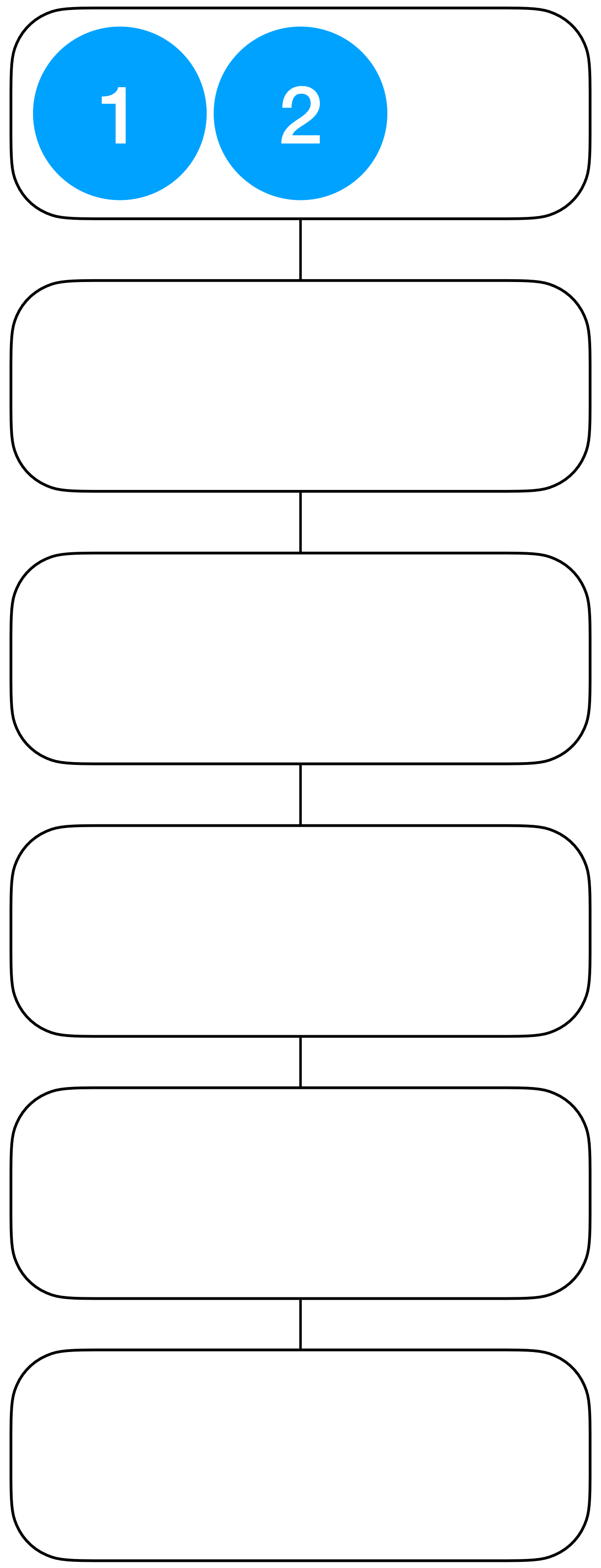


Cycles

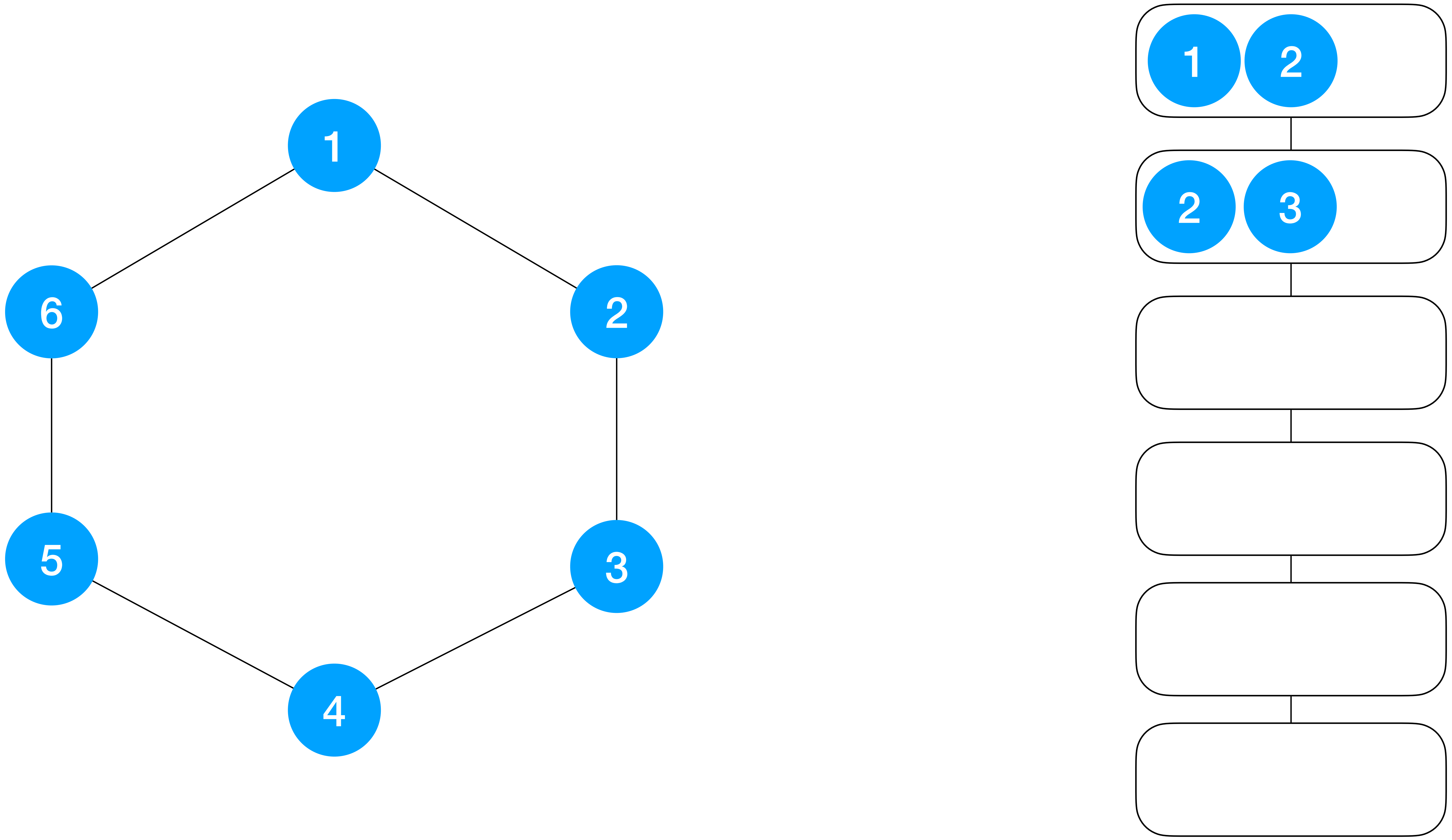


Cycles

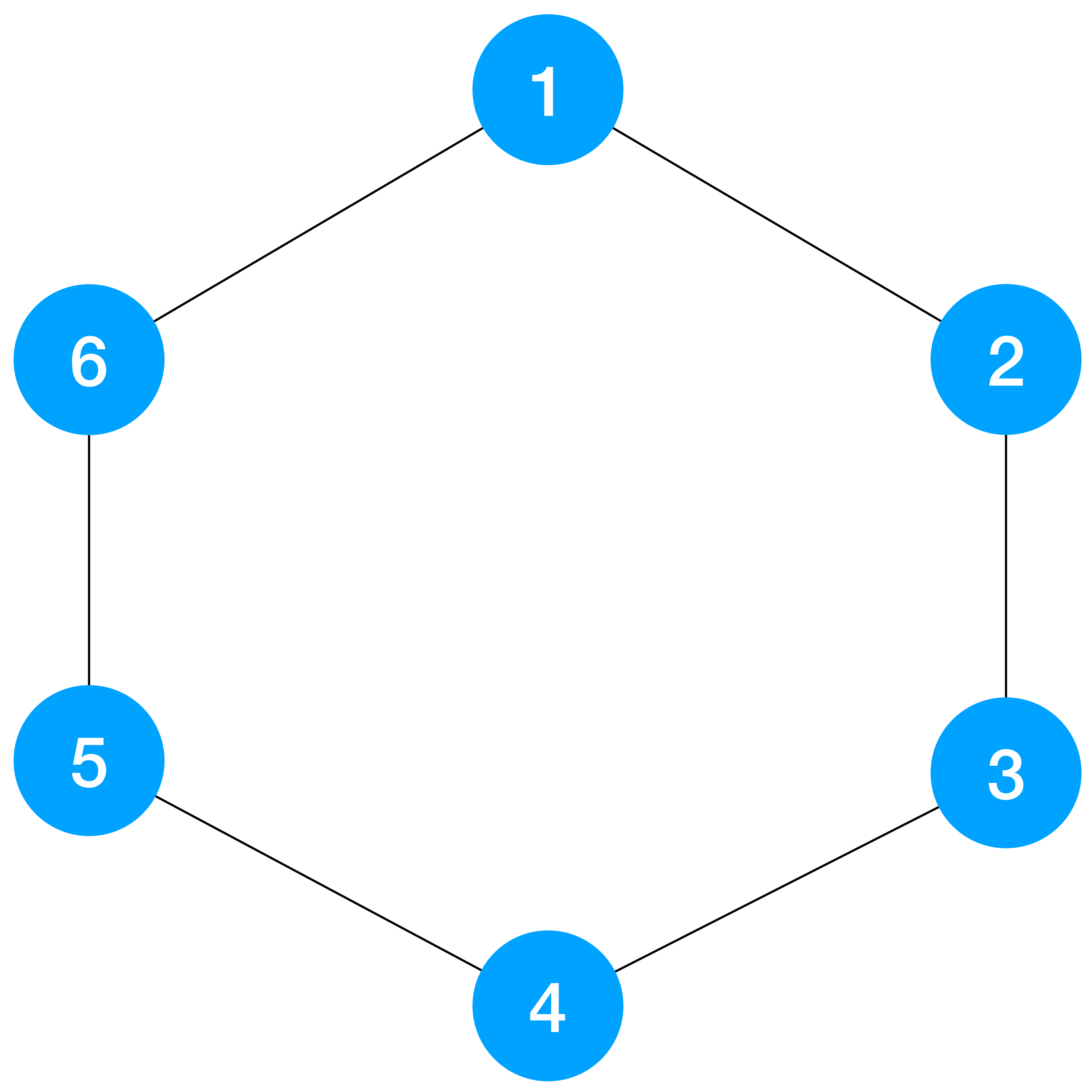


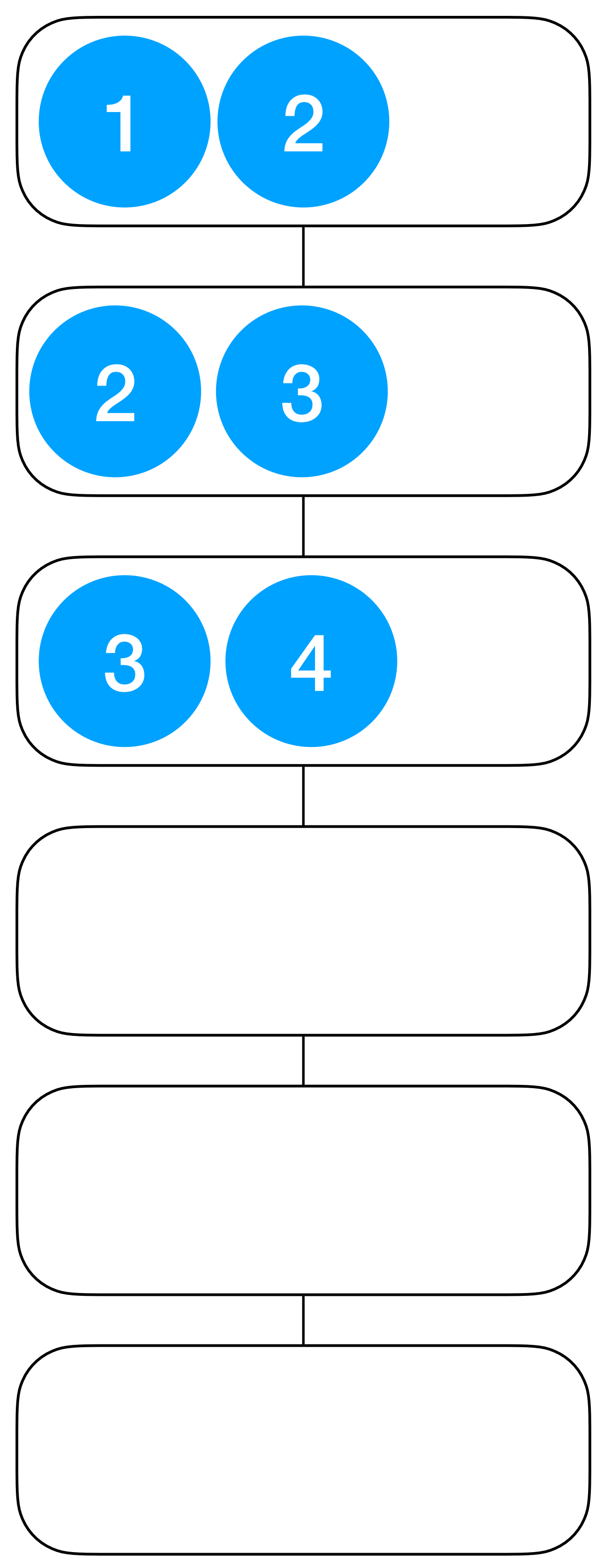


Cycles

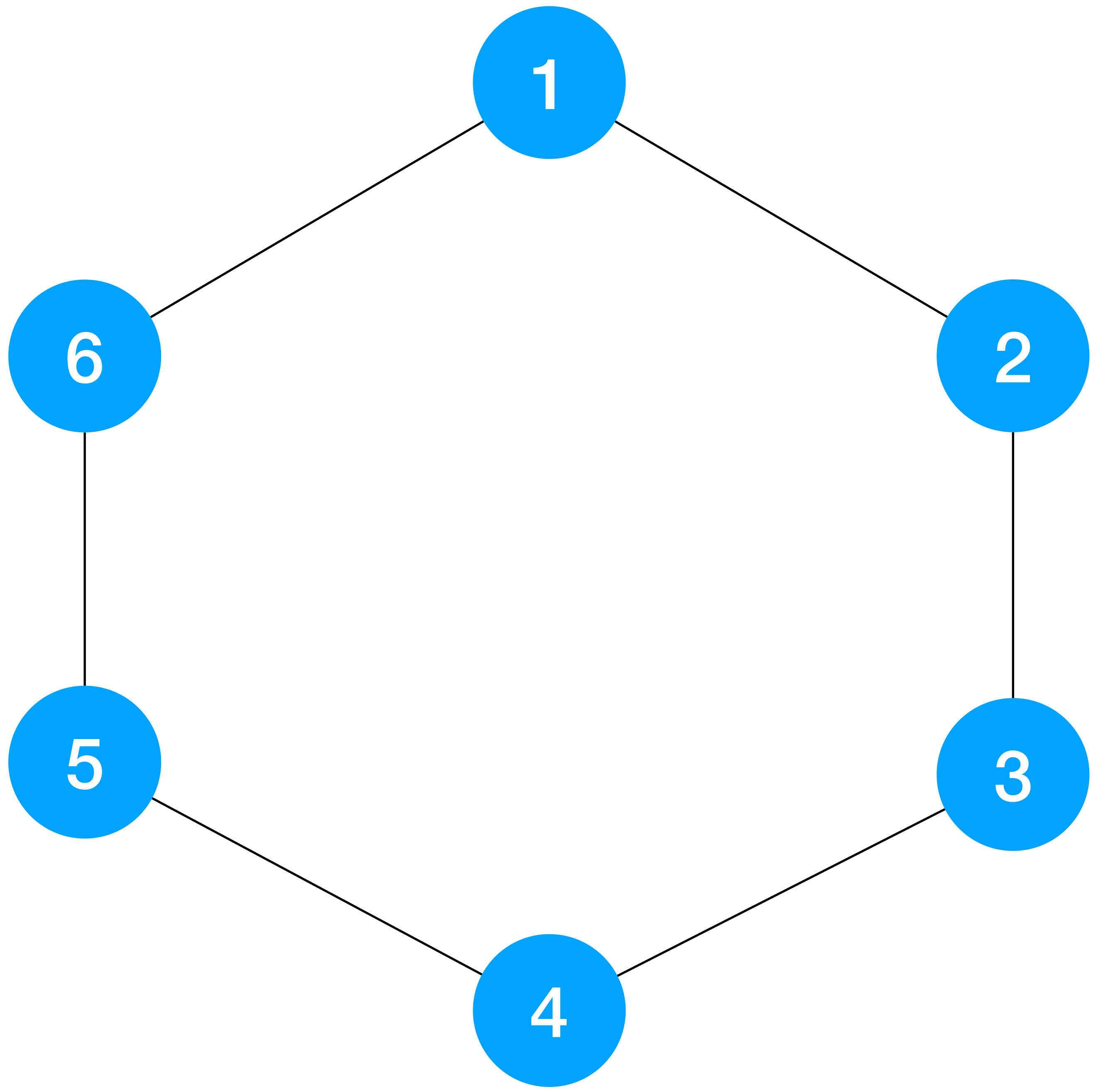


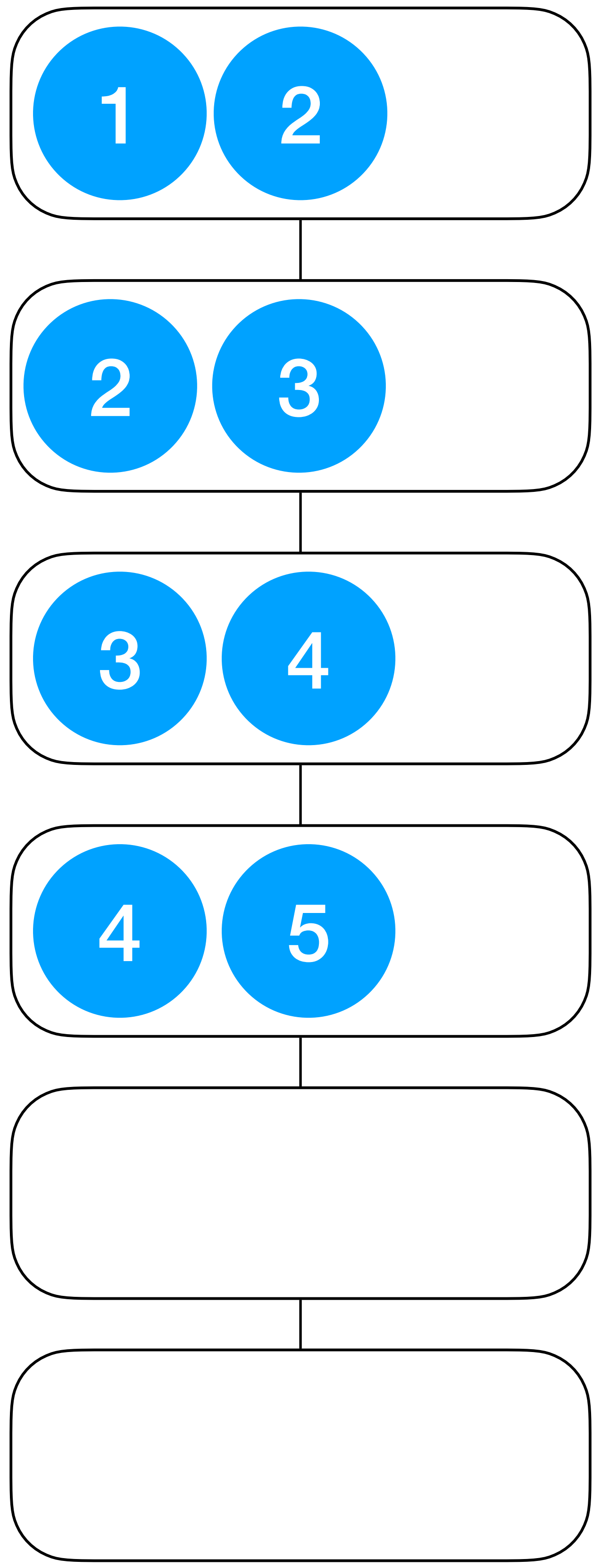
Cycles



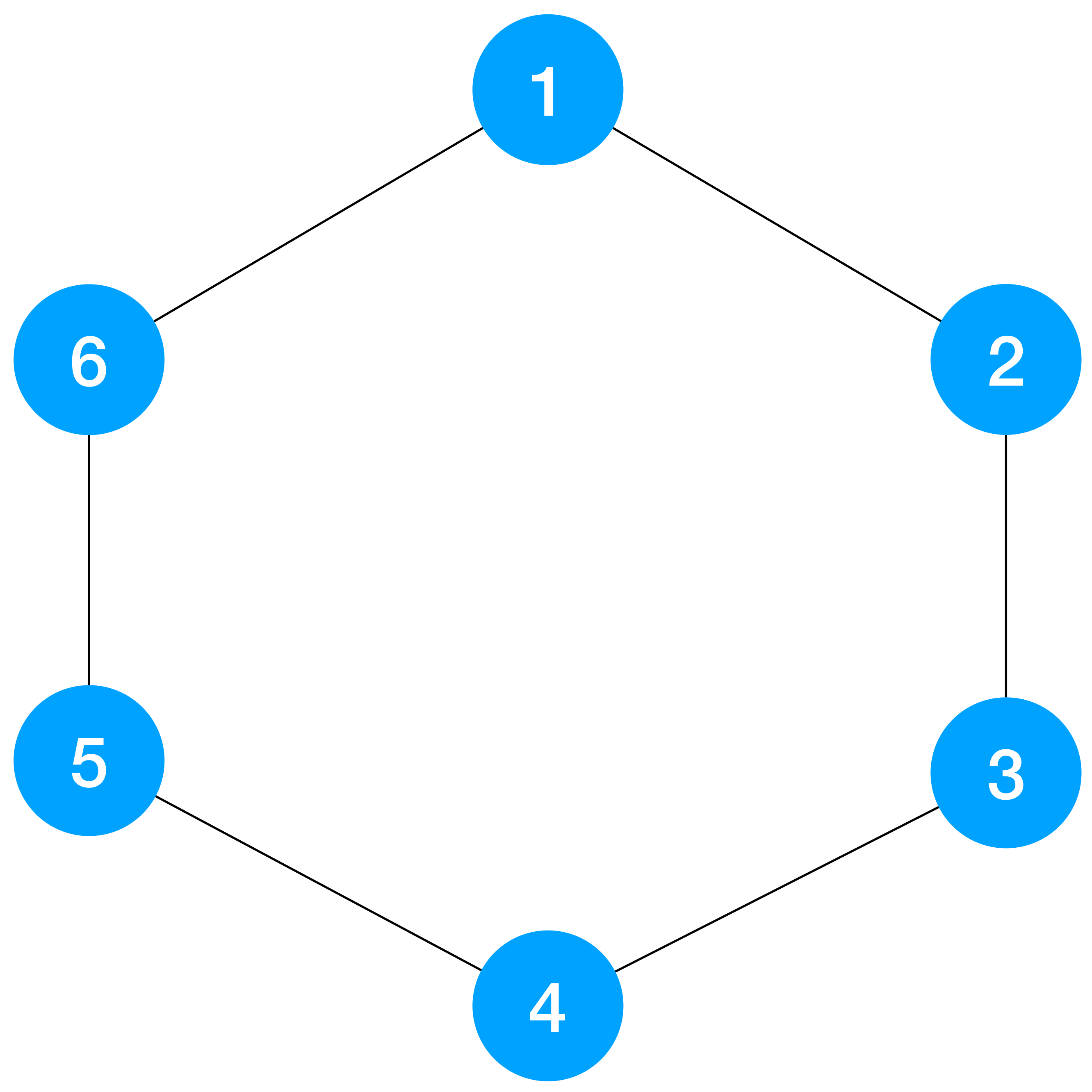


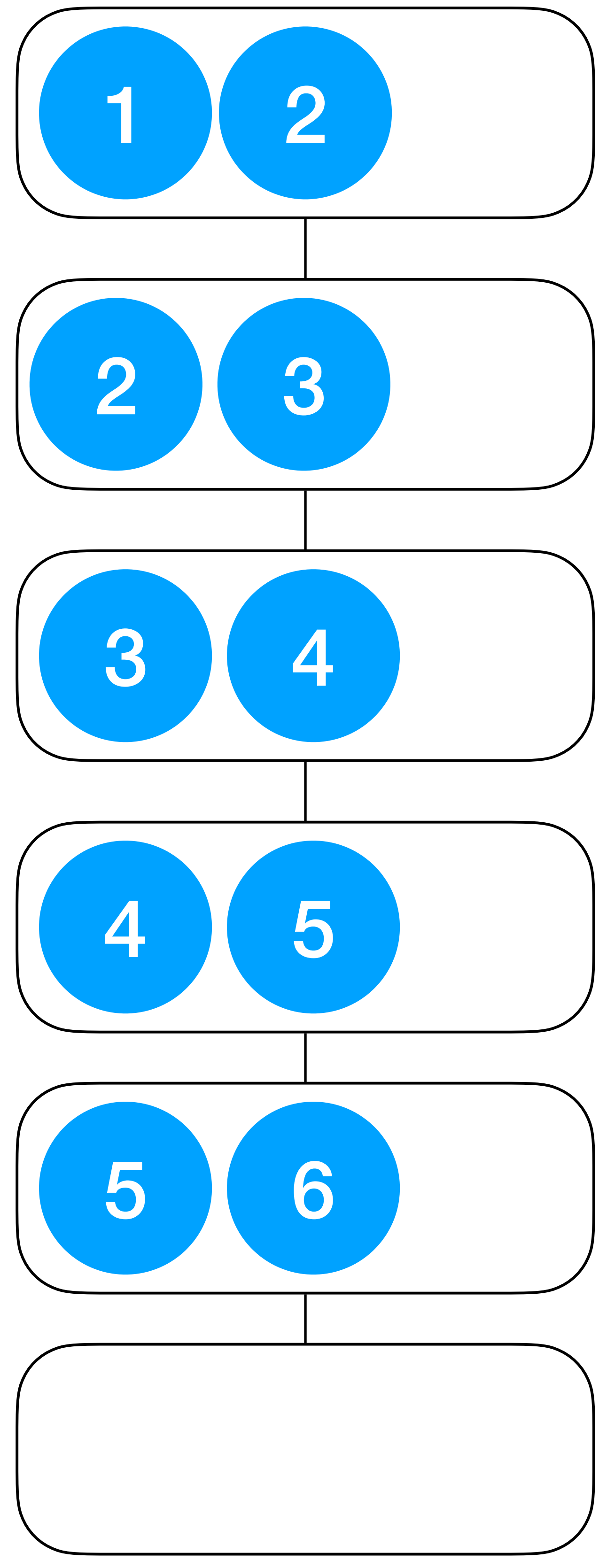
Cycles



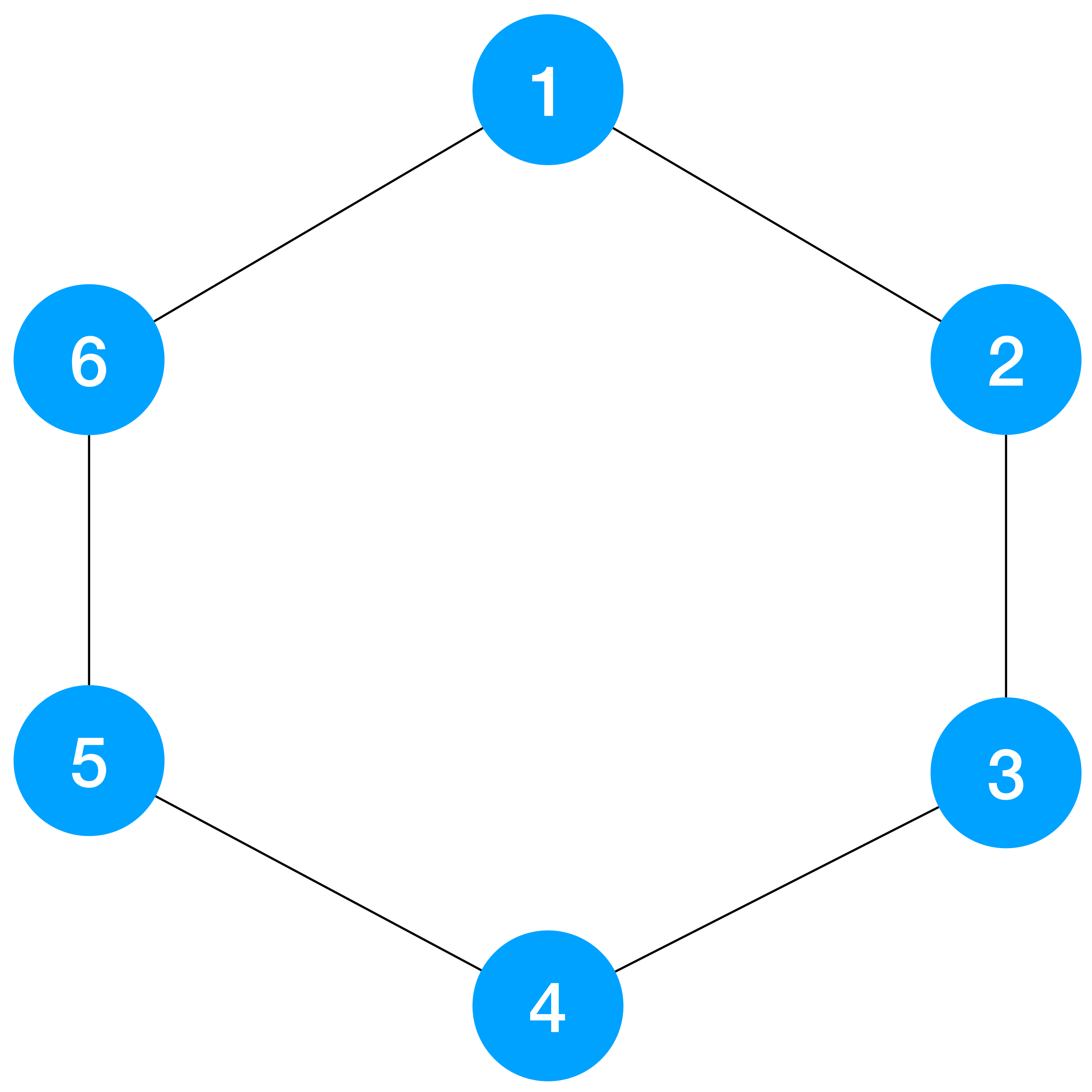


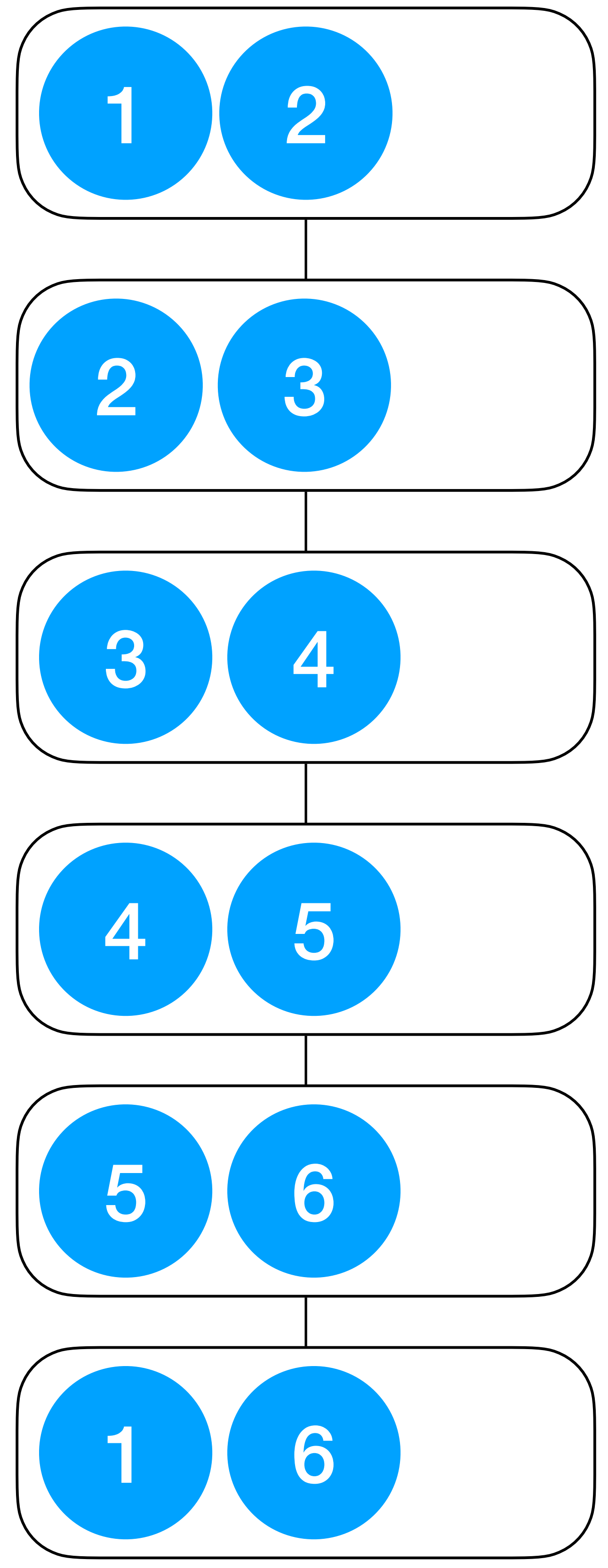
Cycles



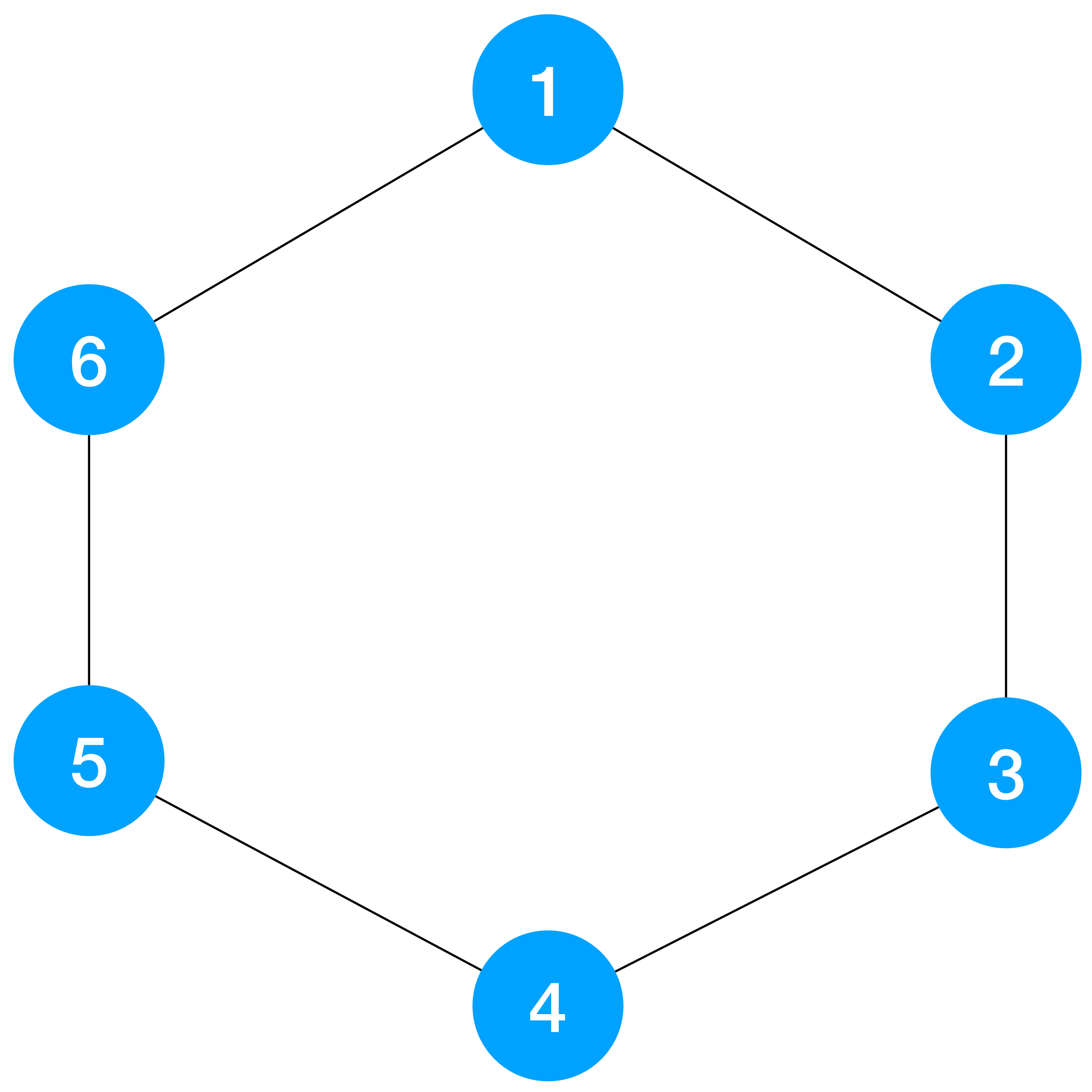


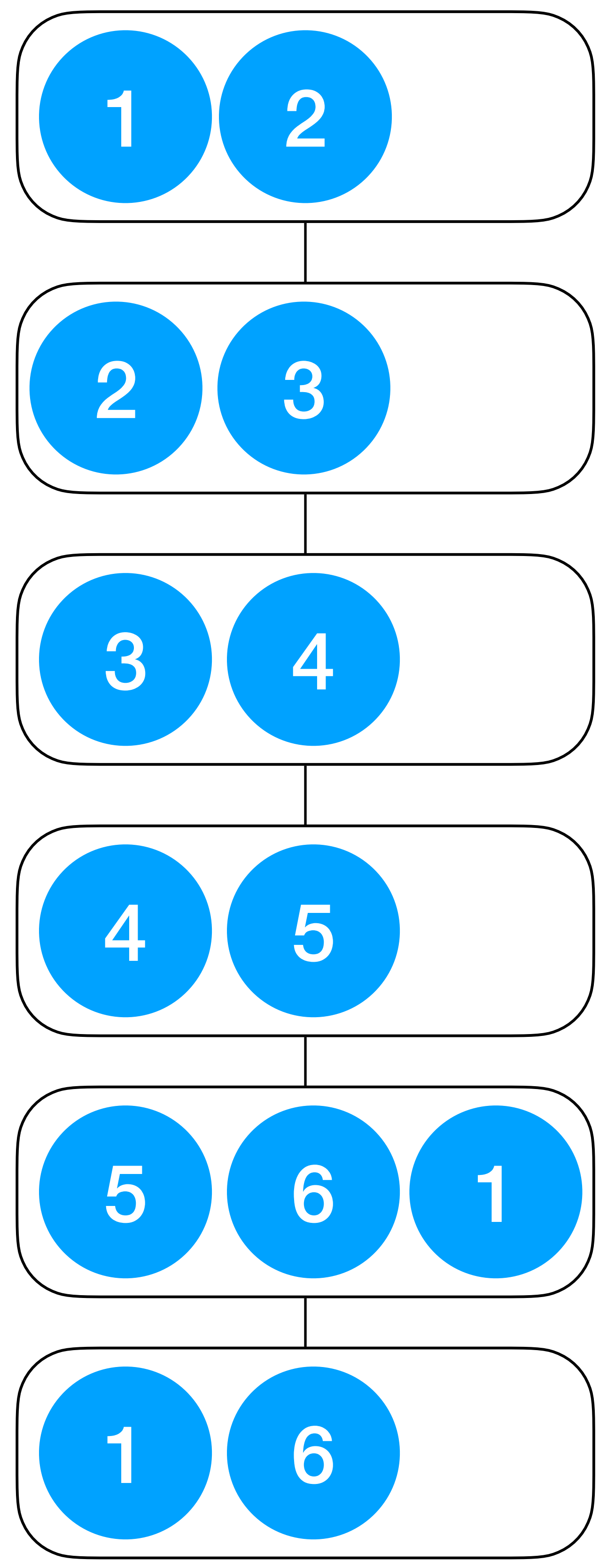
Cycles



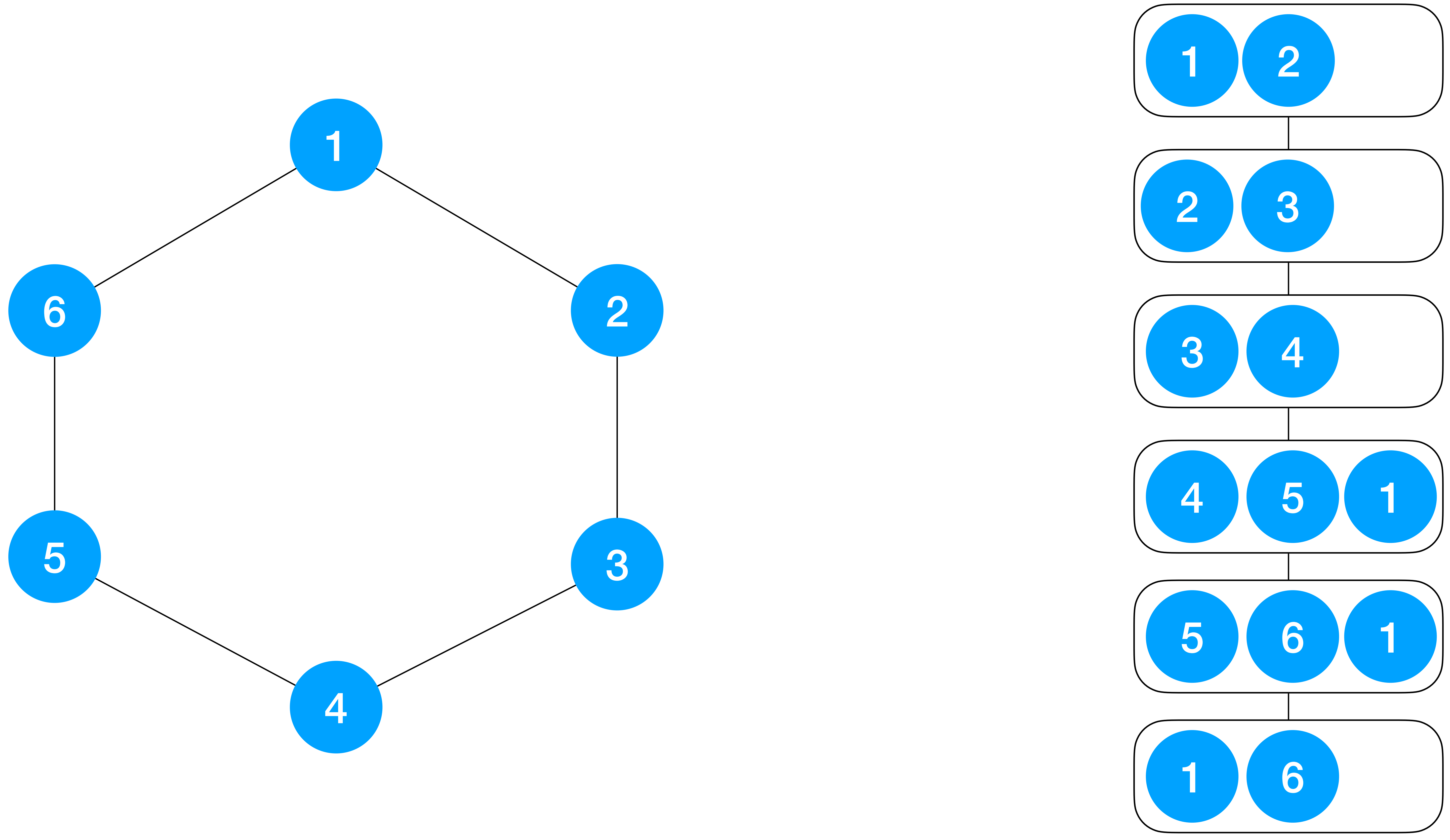


Cycles

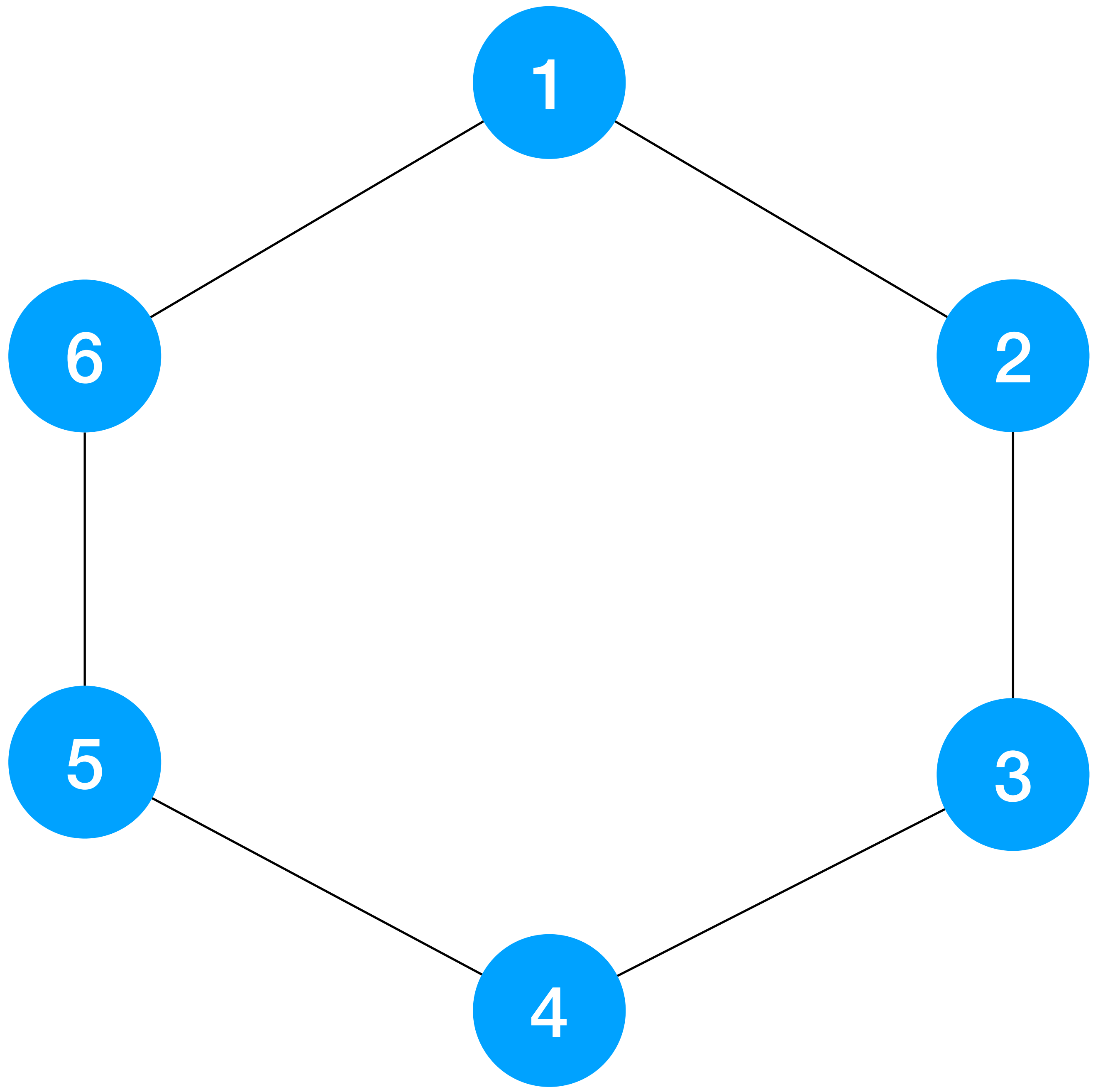


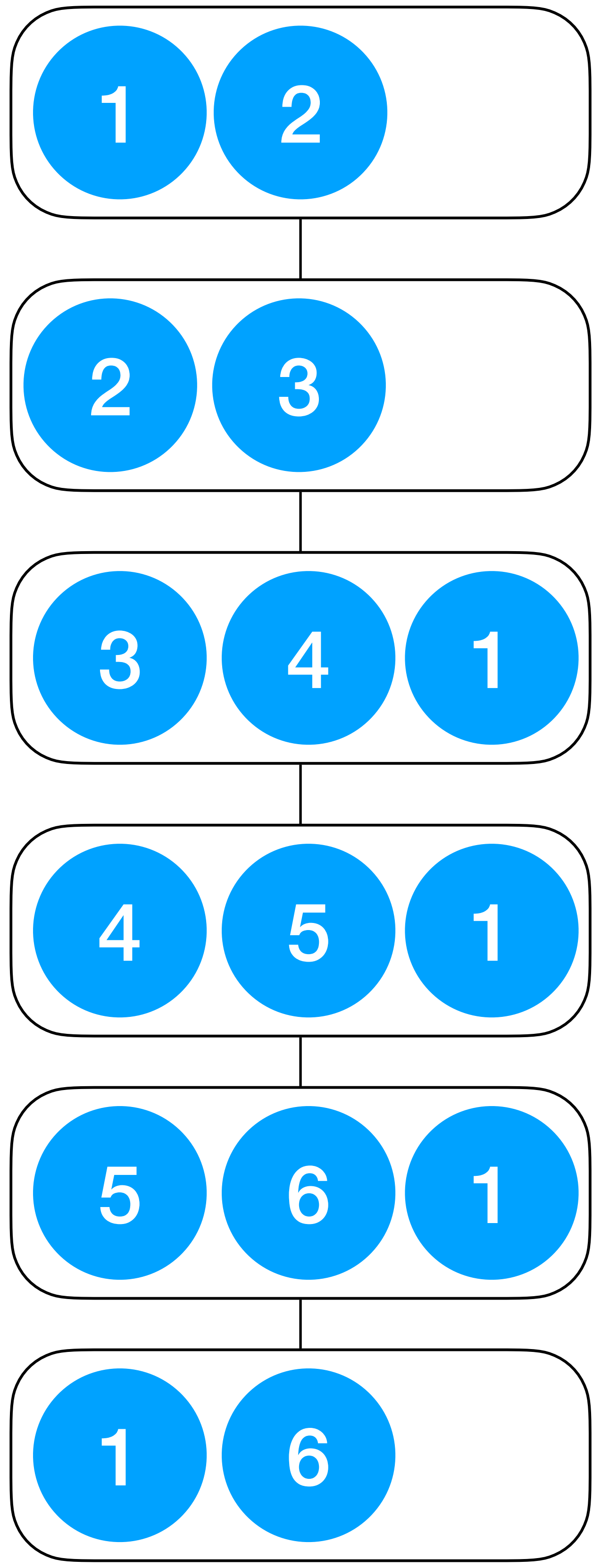


Cycles

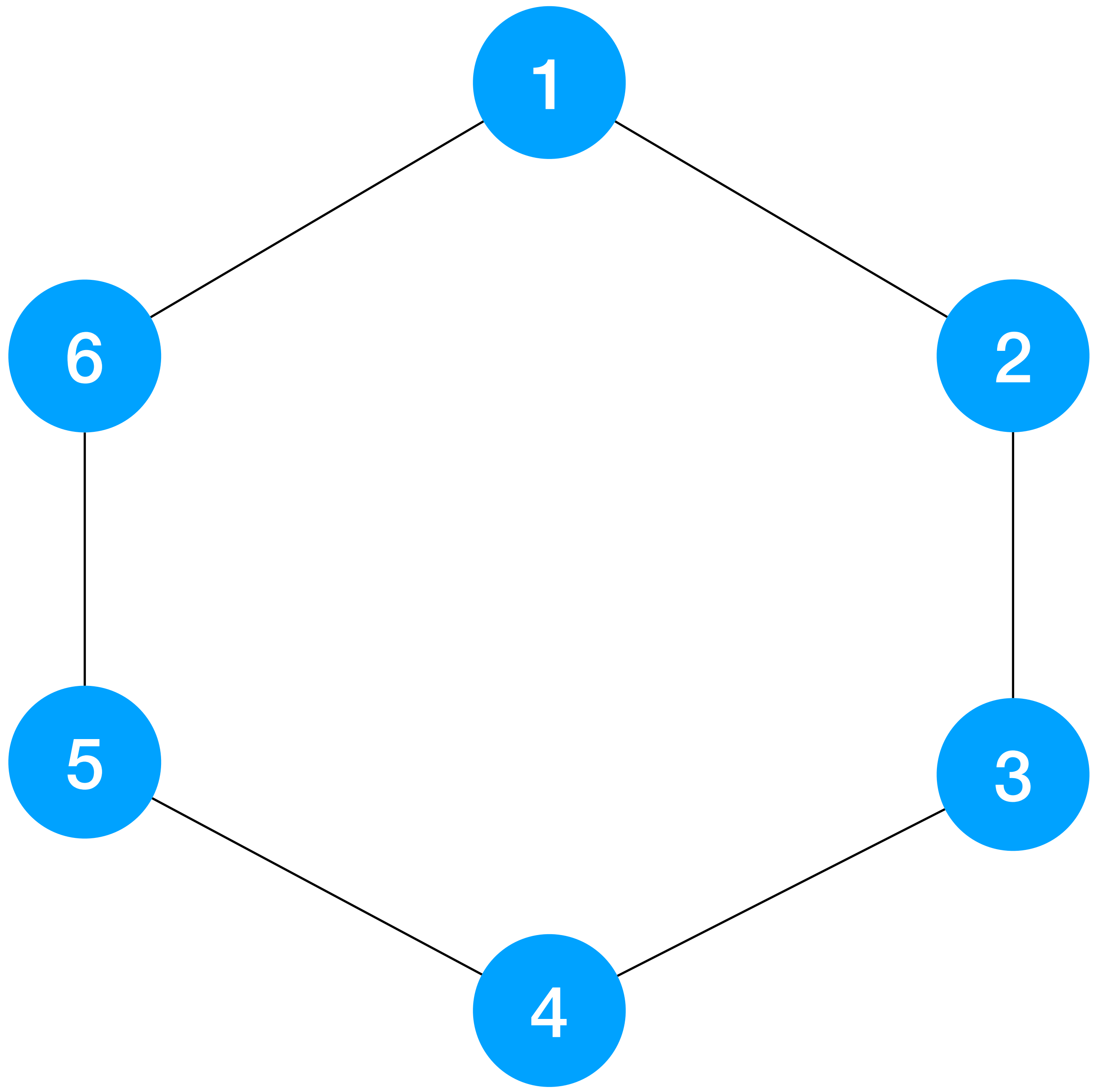


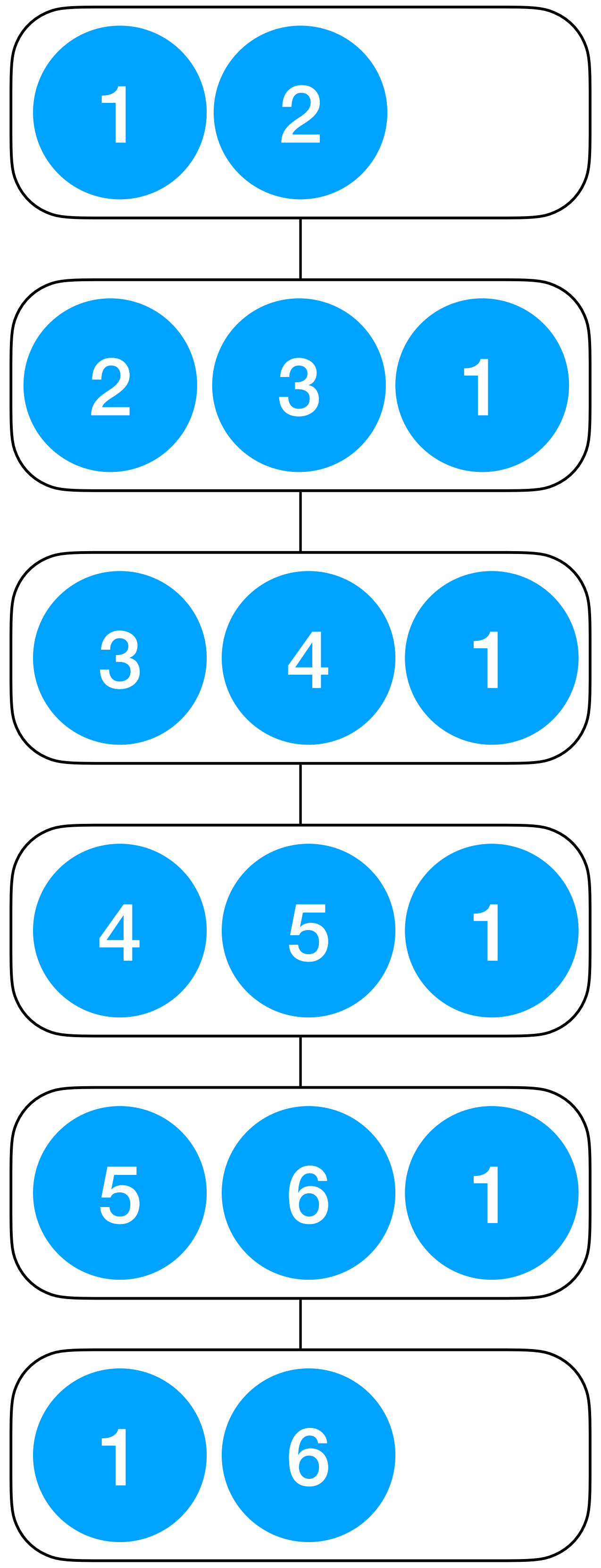
Cycles



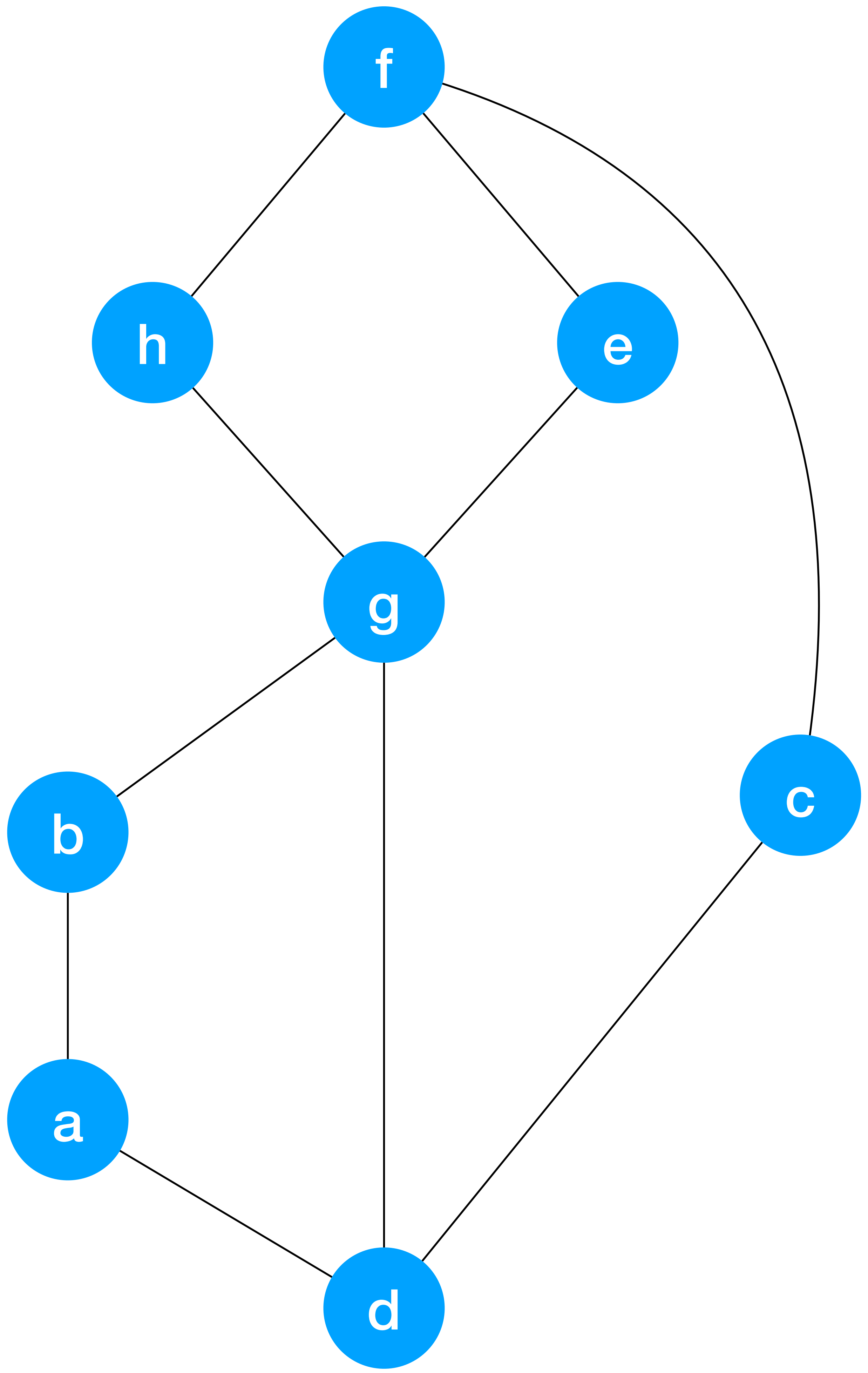


Cycles

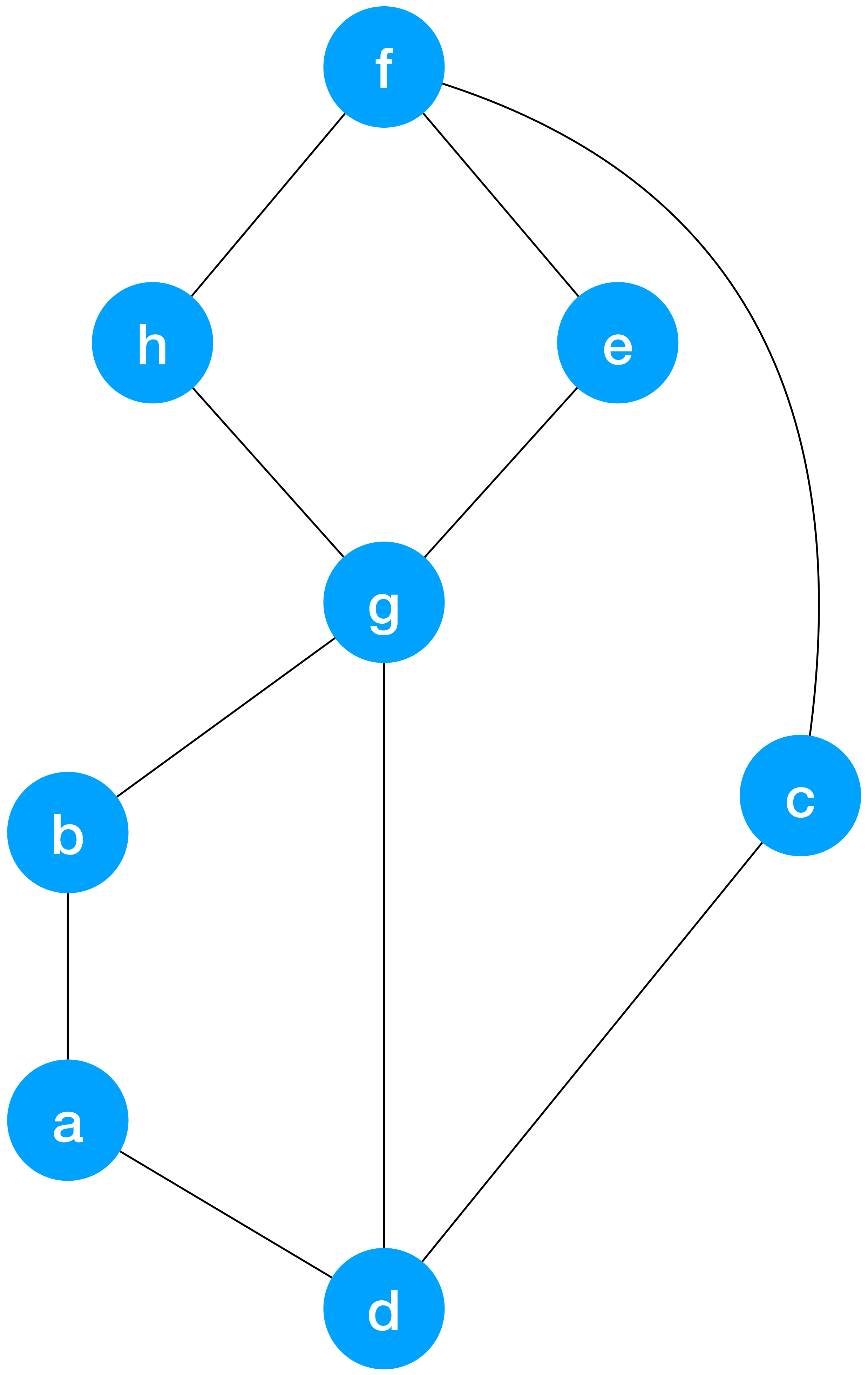


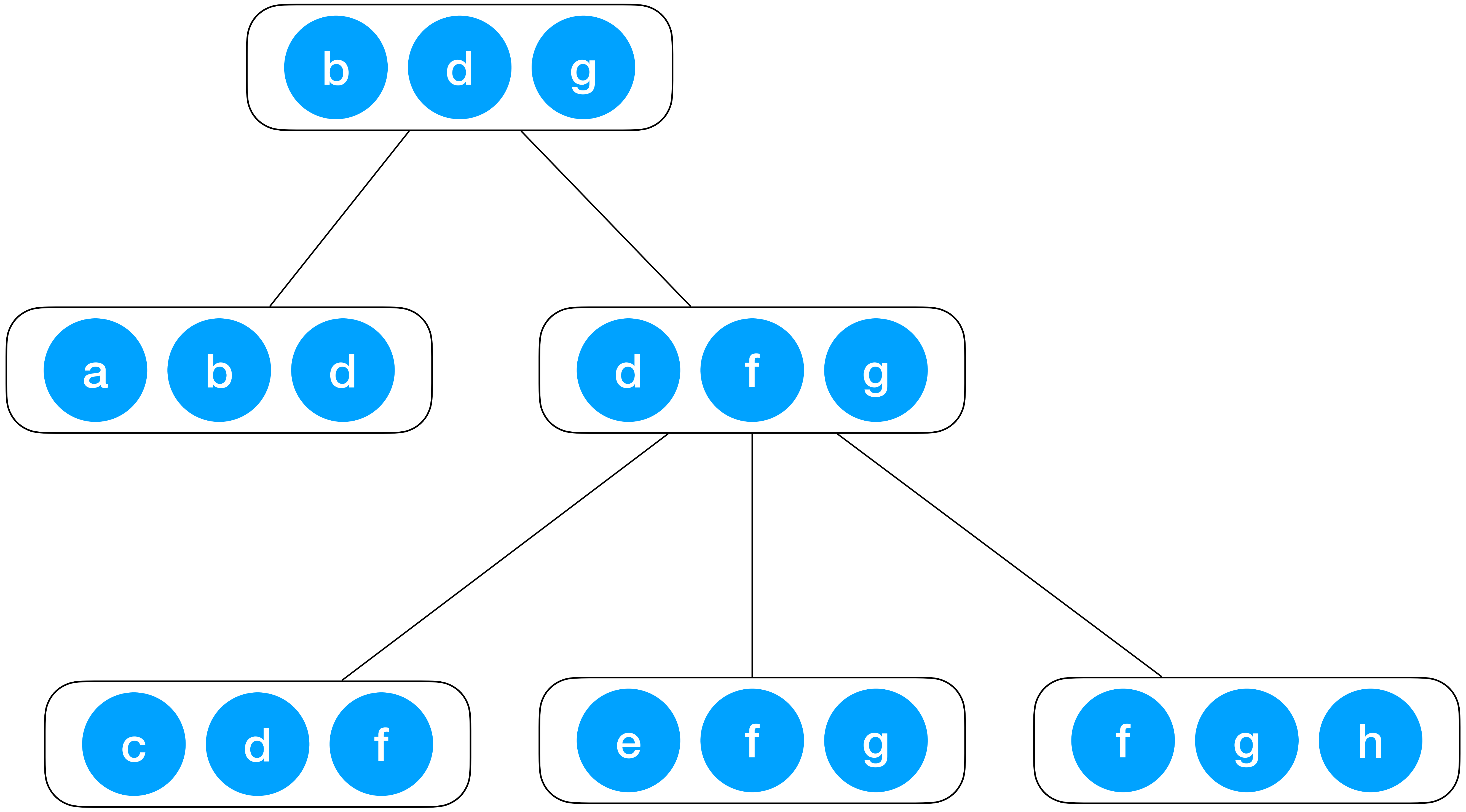


Another Example



Another Example





Properties of Treewidth

Subgraphs

Subgraphs

Observation

If H is a subgraph of G then $\mathbf{tw}(H) \leq \mathbf{tw}(G)$.

Subgraphs

Observation

If H is a subgraph of G then $\mathbf{tw}(H) \leq \mathbf{tw}(G)$.

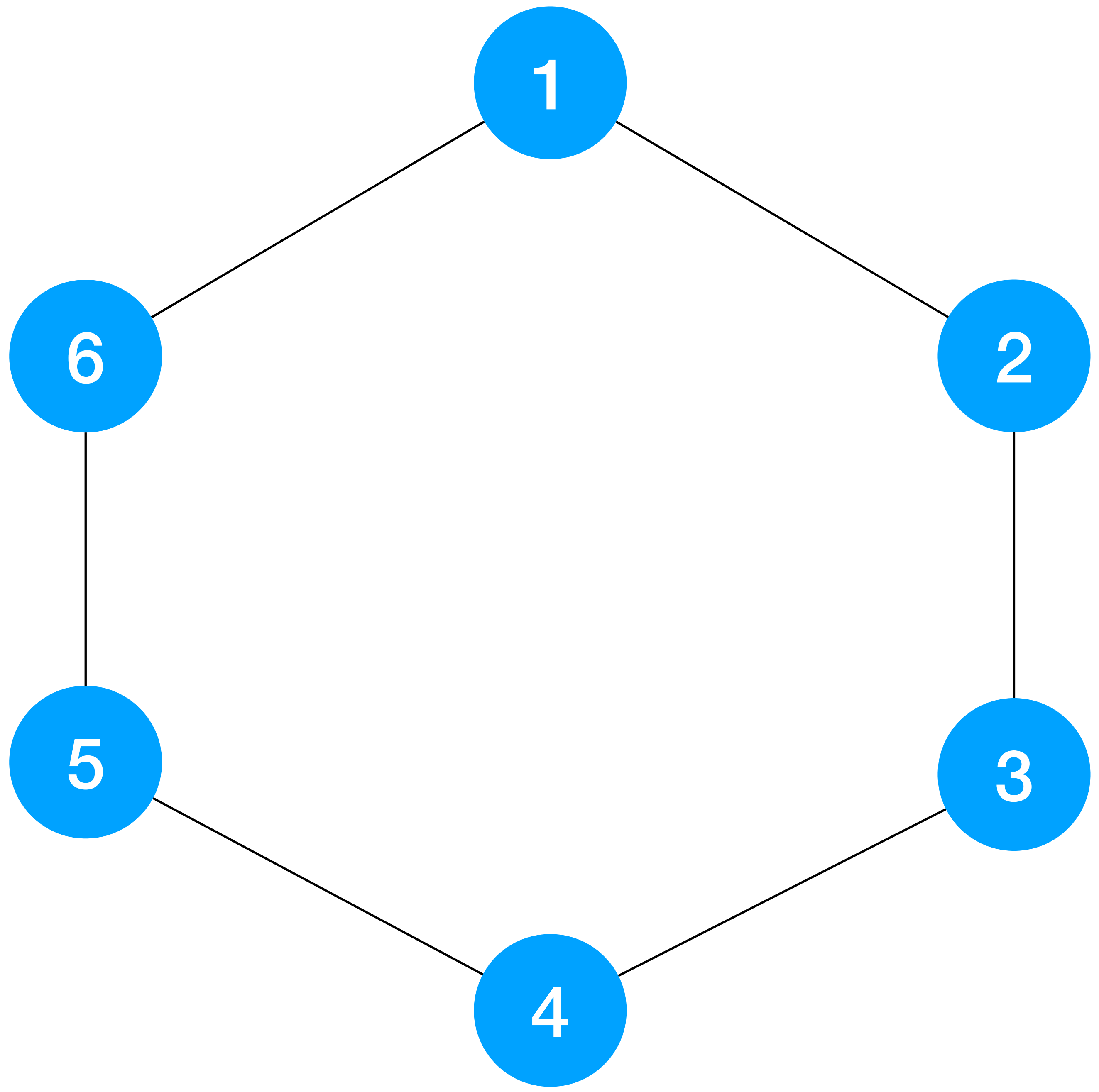
1. Each **vertex** appears in a bag.
2. Each **edge** vw is contained in a bag.
3. The set of nodes in whose bags v appears form a **connected subtree**.

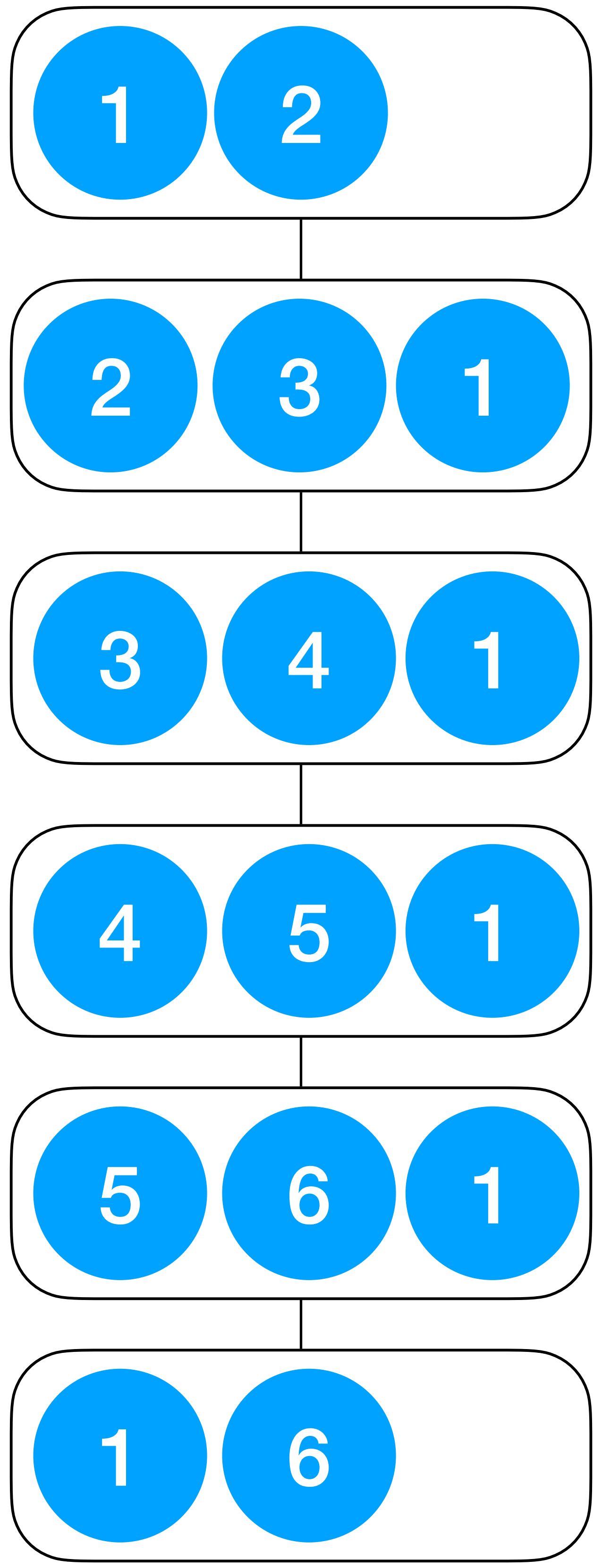
Subgraphs

Observation

If H is a subgraph of G then $\mathbf{tw}(H) \leq \mathbf{tw}(G)$.

1. Each **vertex** appears in a bag.
2. Each **edge** vw is contained in a bag.
3. The set of nodes in whose bags v appears form a **connected subtree**.



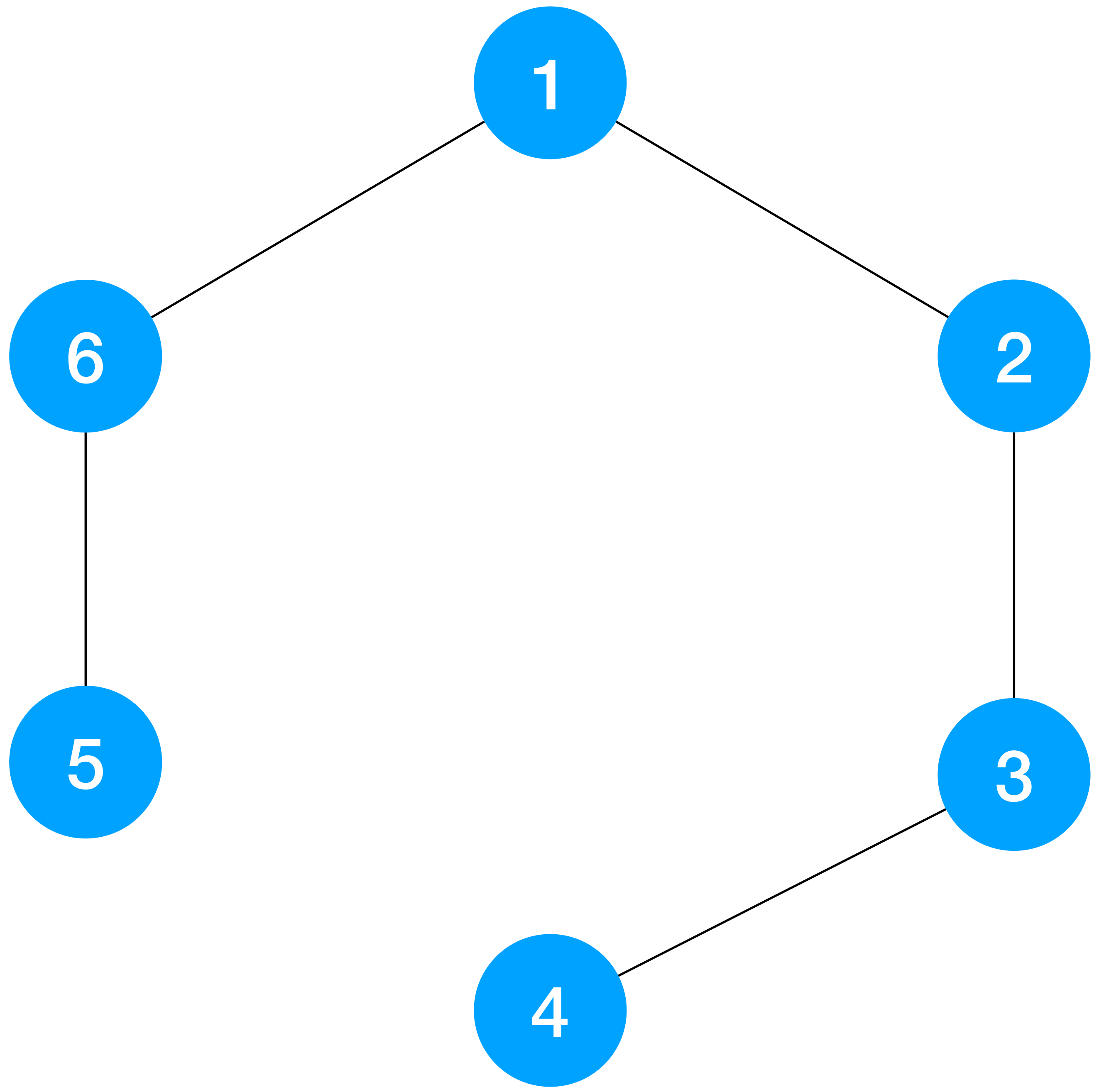


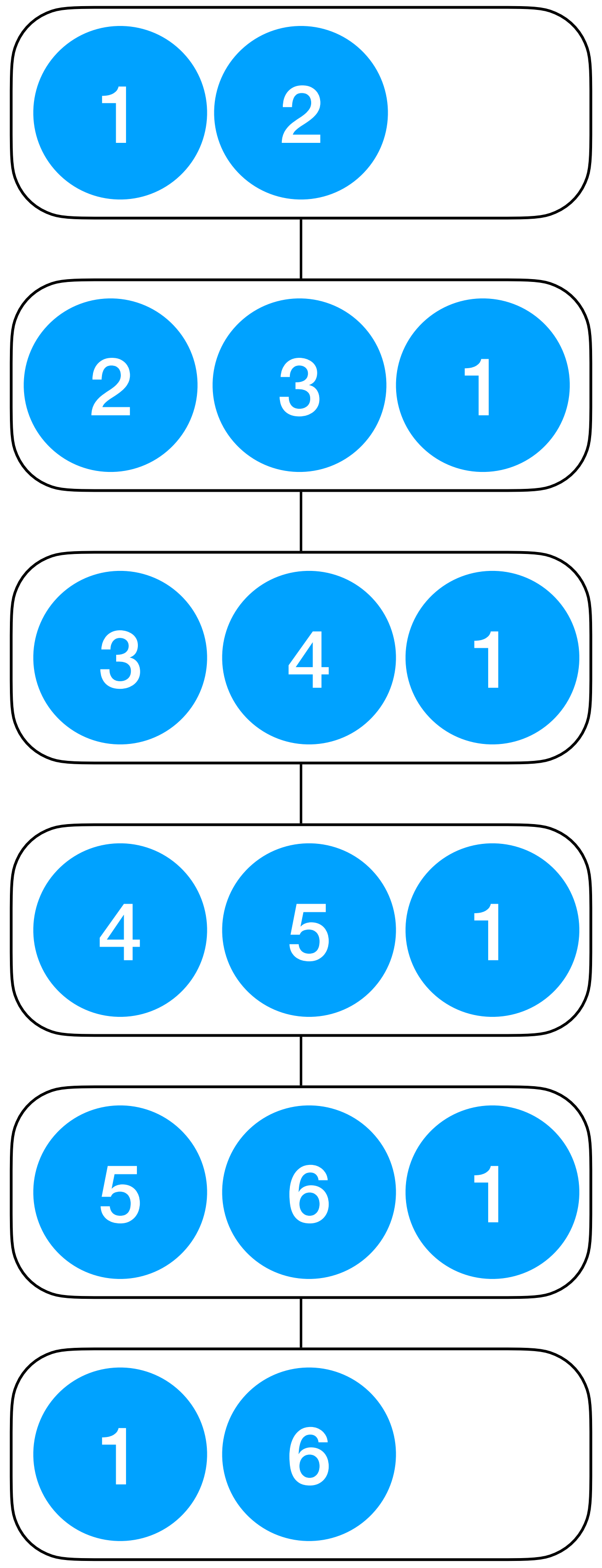
Subgraphs

Observation

If H is a subgraph of G then $\mathbf{tw}(H) \leq \mathbf{tw}(G)$.

1. Each **vertex** appears in a bag.
2. Each **edge** vw is contained in a bag.
3. The set of nodes in whose bags v appears form a **connected subtree**.



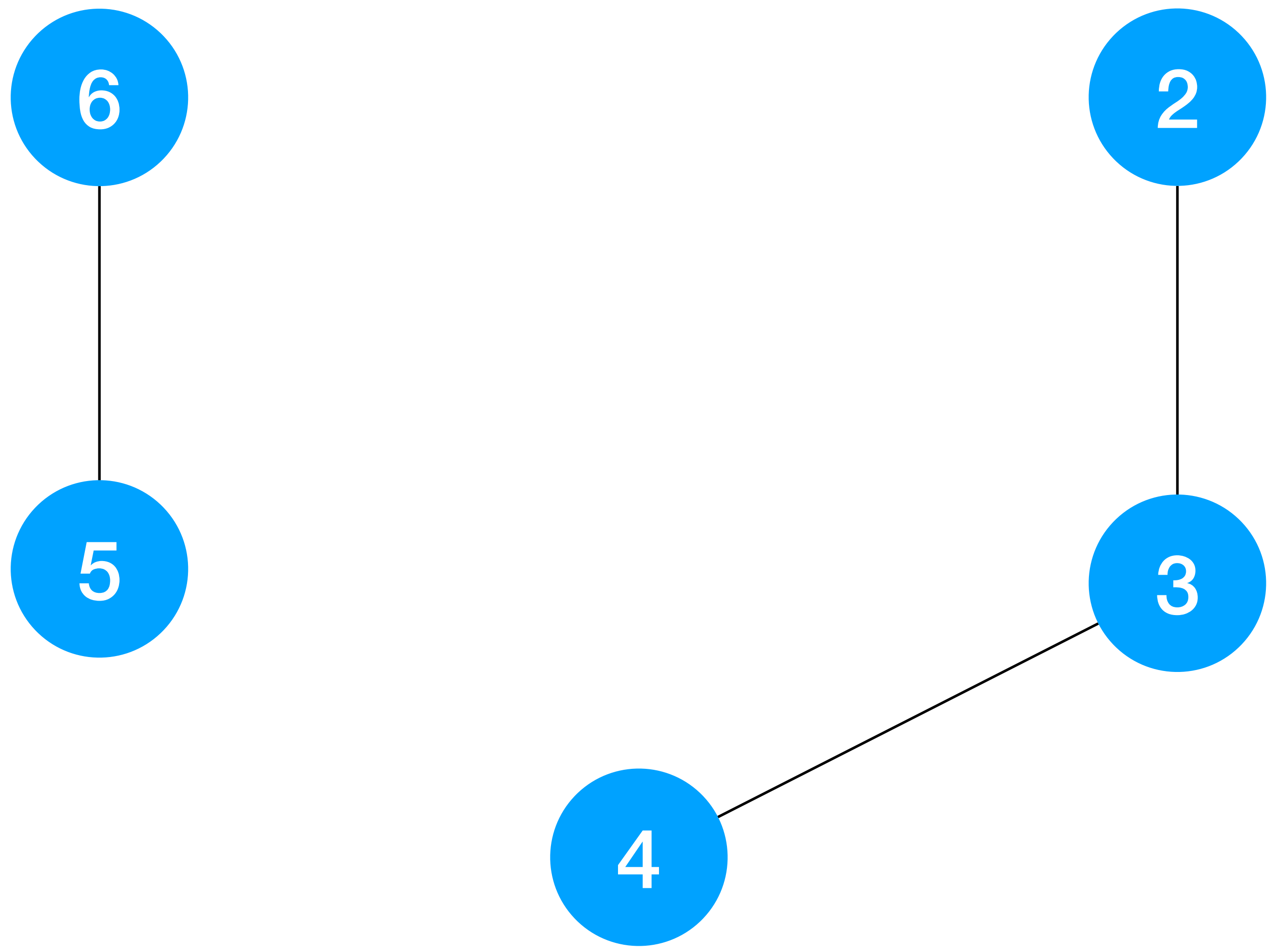


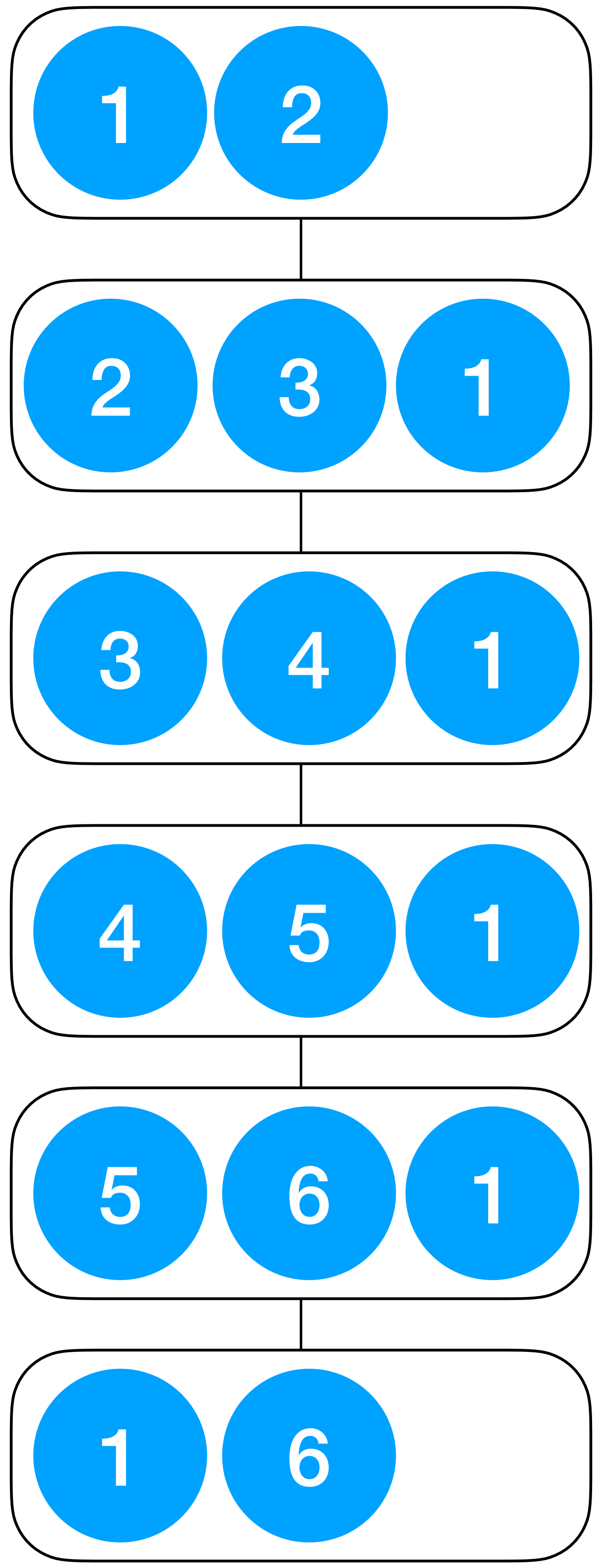
Subgraphs

Observation

If H is a subgraph of G then $\mathbf{tw}(H) \leq \mathbf{tw}(G)$.

1. Each **vertex** appears in a bag.
2. Each **edge** vw is contained in a bag.
3. The set of nodes in whose bags v appears form a **connected subtree**.



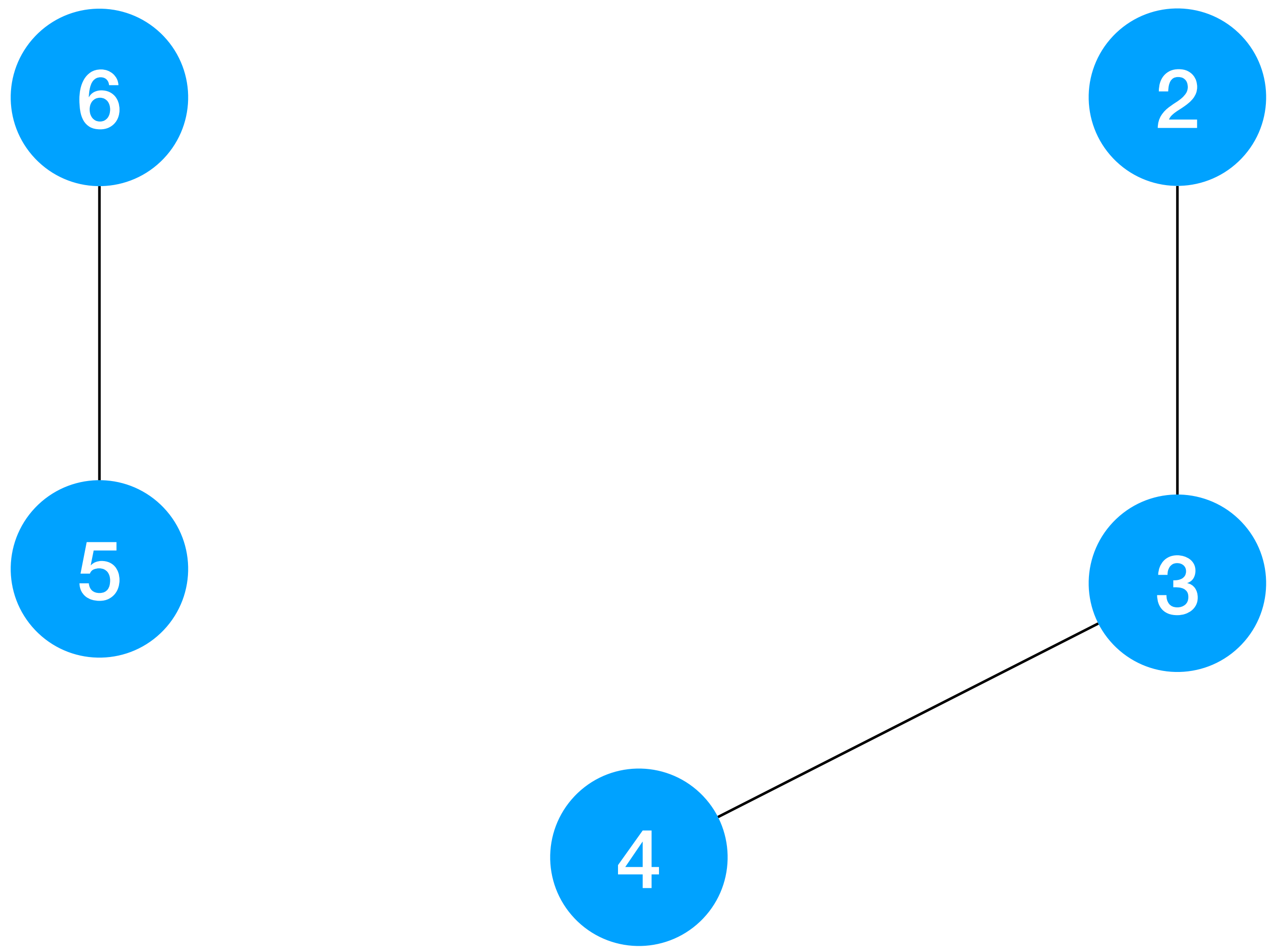


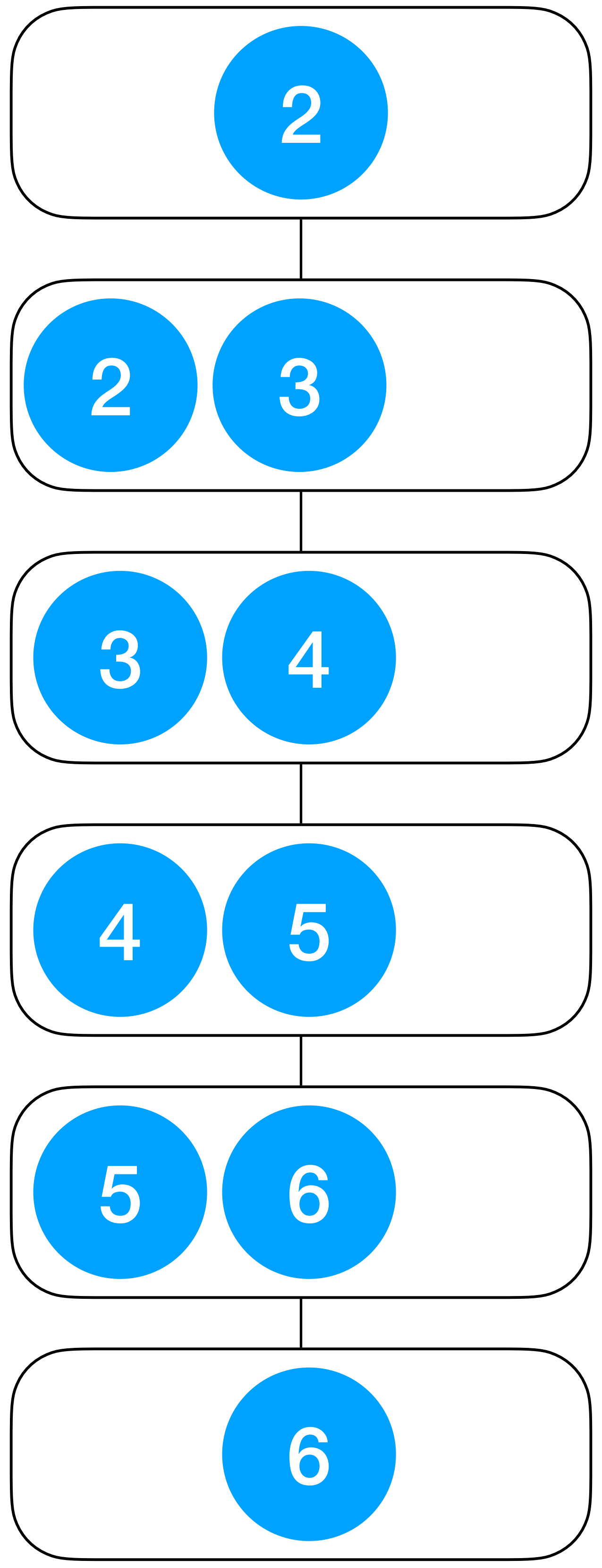
Subgraphs

Observation

If H is a subgraph of G then $\mathbf{tw}(H) \leq \mathbf{tw}(G)$.

1. Each **vertex** appears in a bag.
2. Each **edge** vw is contained in a bag.
3. The set of nodes in whose bags v appears form a **connected subtree**.





Disjoint Union

Observation

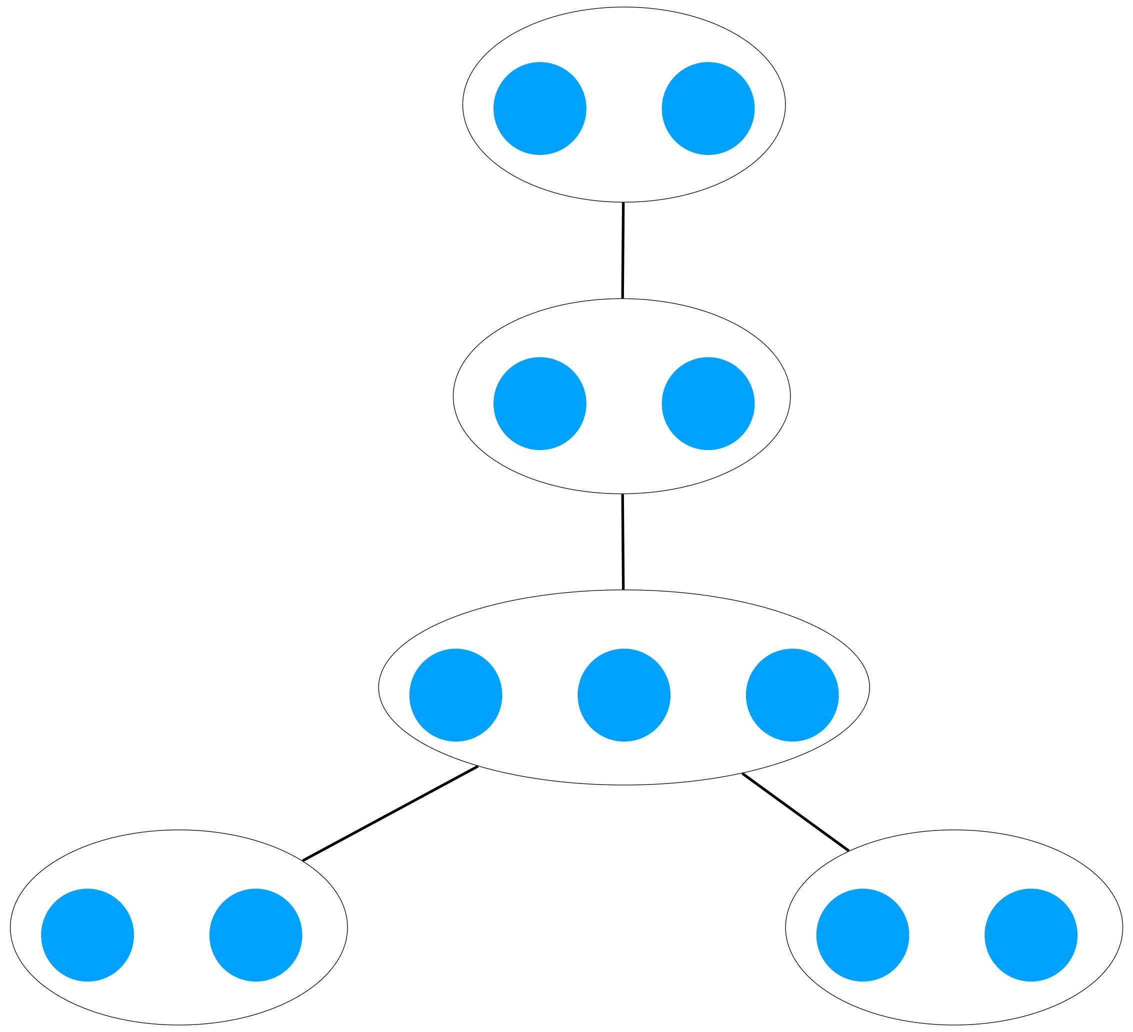
Let G be the disjoint union of graphs G_1 and G_2.
Then $\mathbf{tw}(G) = \max(\mathbf{tw}(G_1), \mathbf{tw}(G_2))$.

Disjoint Union

Observation

Let G be the disjoint union of graphs G_1 and G_2.
Then $\mathbf{tw}(G) = \max(\mathbf{tw}(G_1), \mathbf{tw}(G_2))$.

G_1

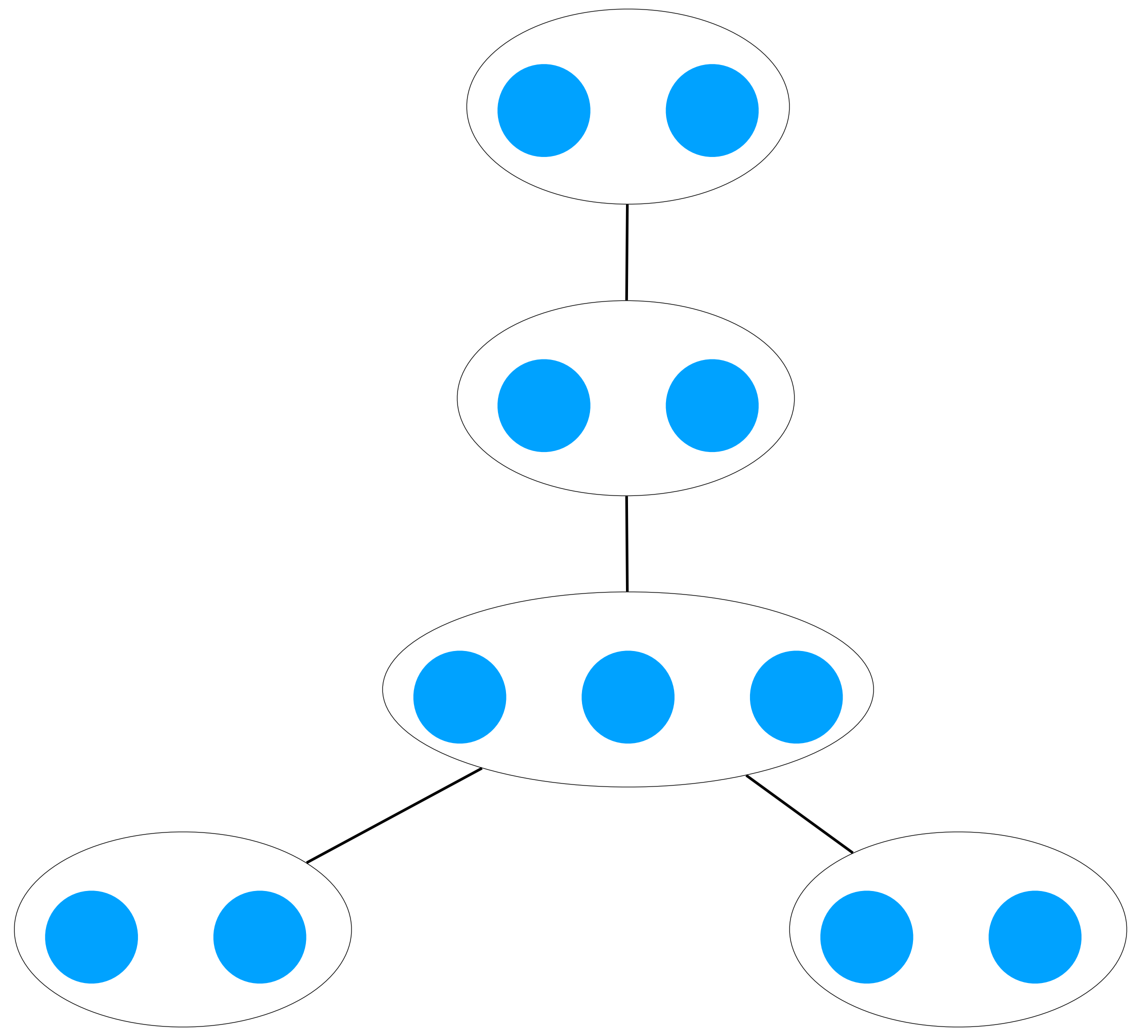


Disjoint Union

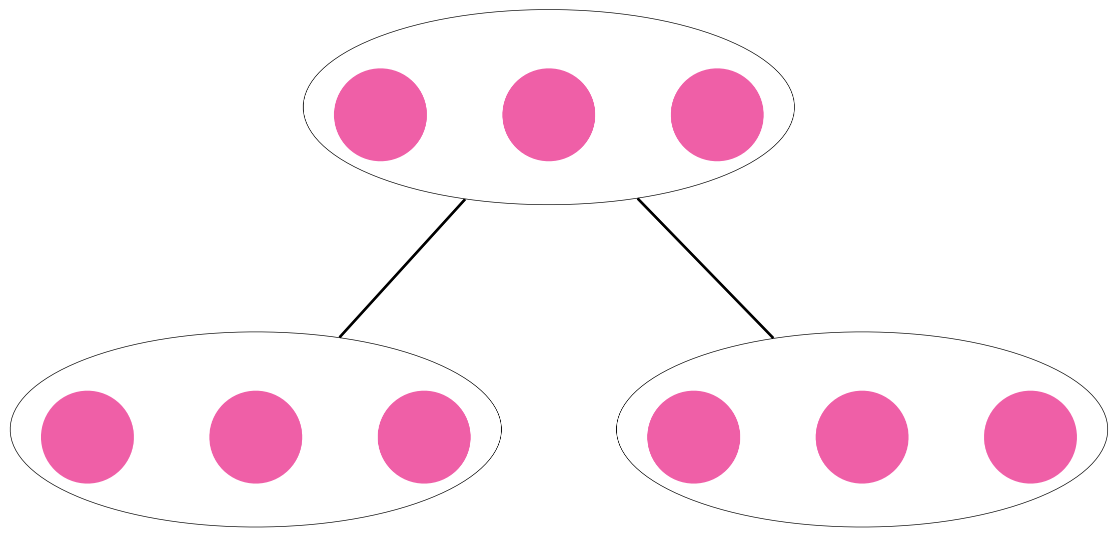
Observation

Let G be the disjoint union of graphs G_1 and G_2.
Then $\mathbf{tw}(G) = \max(\mathbf{tw}(G_1), \mathbf{tw}(G_2))$.

G_1



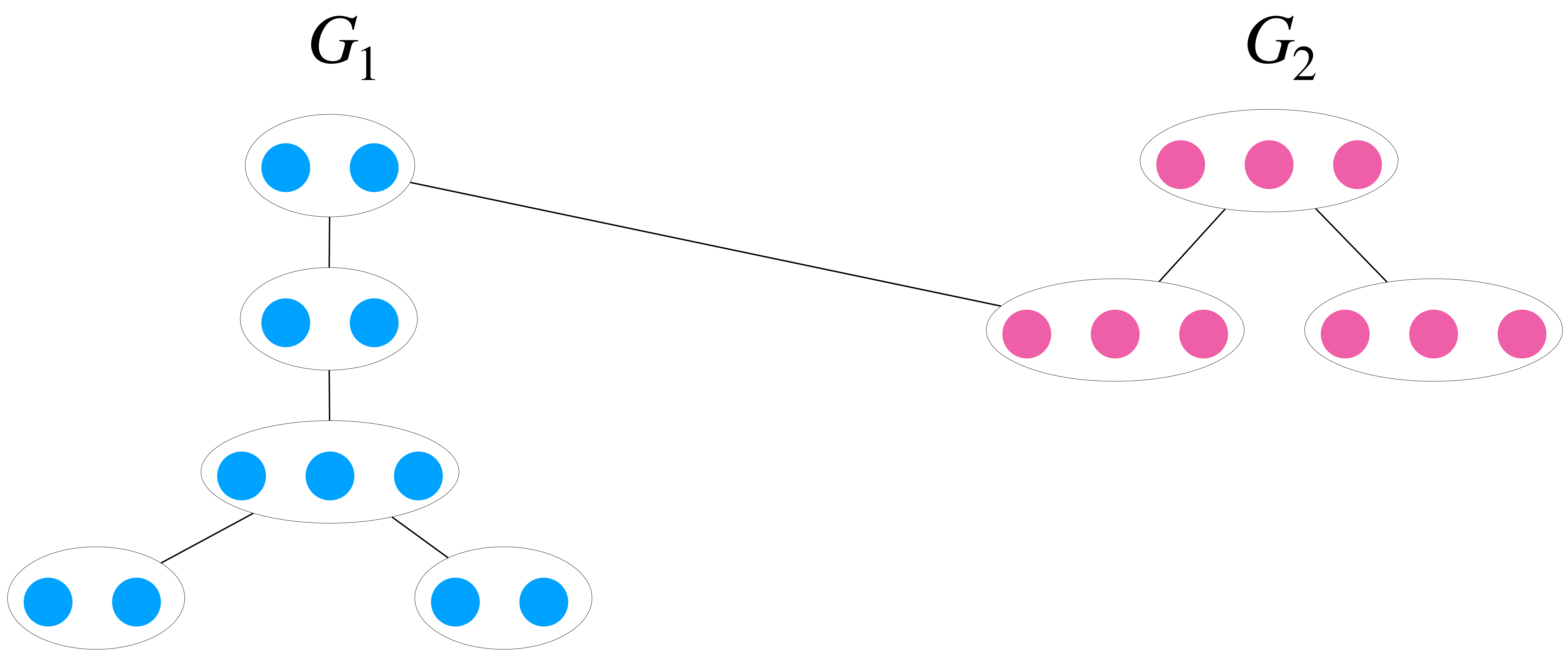
G_2



Disjoint Union

Observation

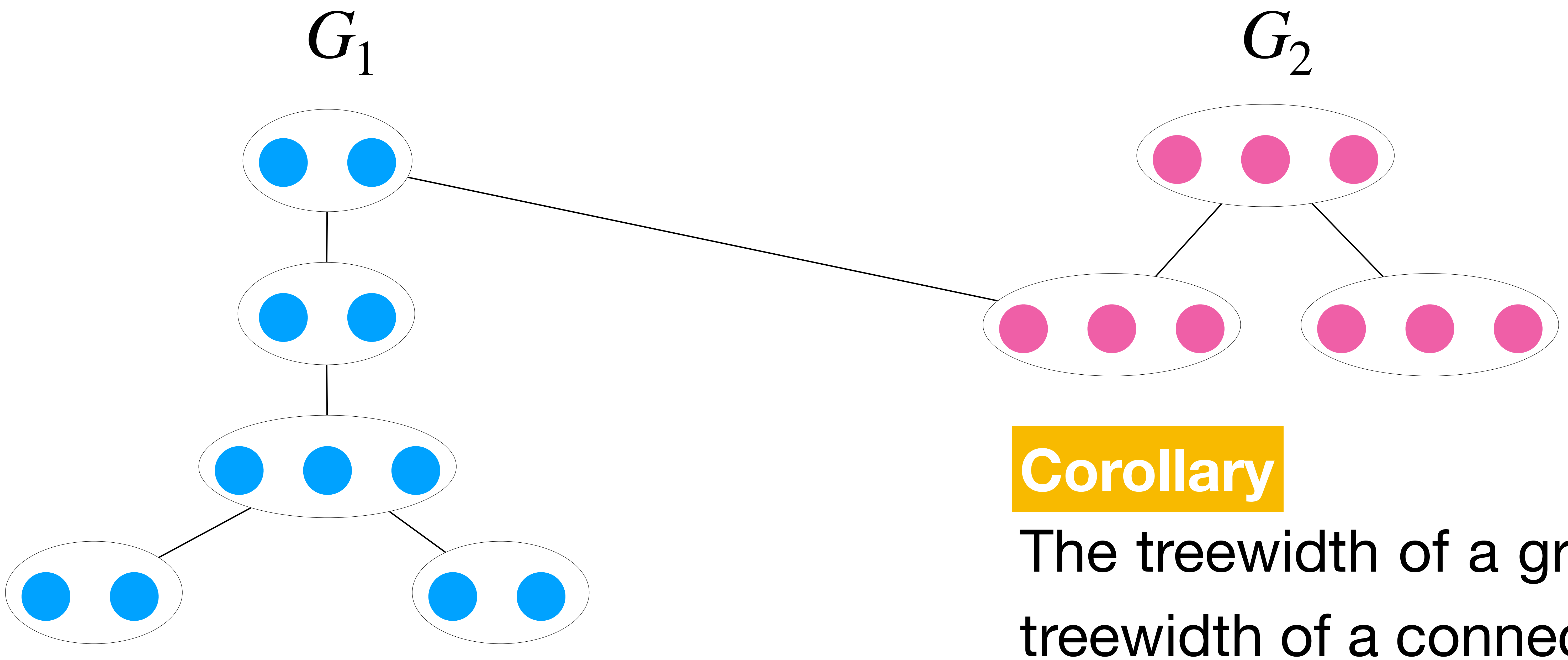
Let G be the disjoint union of graphs G_1 and G_2.
Then $\mathbf{tw}(G) = \max(\mathbf{tw}(G_1), \mathbf{tw}(G_2))$.



Disjoint Union

Observation

Let G be the disjoint union of graphs G_1 and G_2.
Then $\mathbf{tw}(G) = \max(\mathbf{tw}(G_1), \mathbf{tw}(G_2))$.



Corollary

The treewidth of a graph G is the maximum treewidth of a connected component of G.

Small Tree Decompositions

Definition

A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Small Tree Decompositions

Definition

A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Proposition

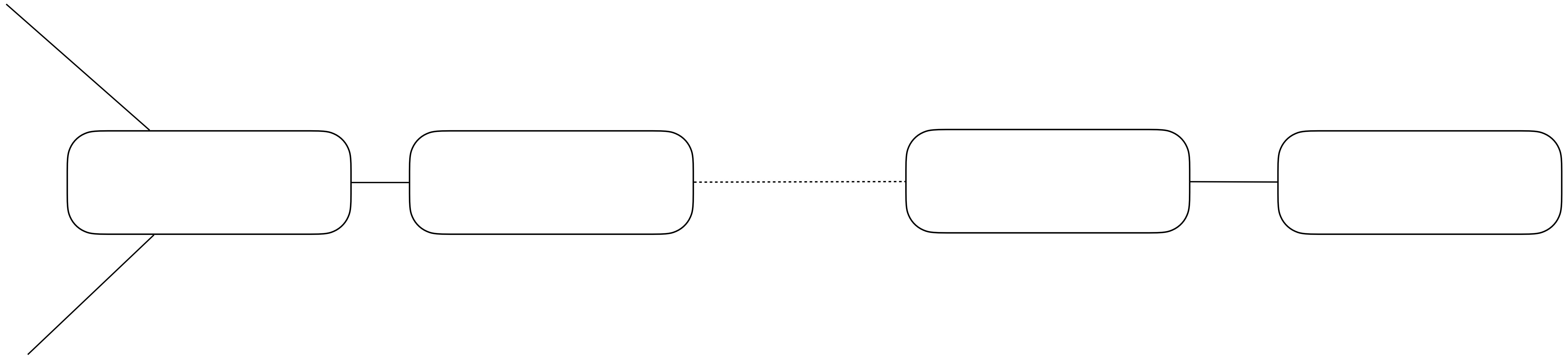
There is a polynomial-time algorithm that turns any tree decomposition into a small tree decomposition.

Small Tree Decompositions

Definition
A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Proposition

There is a polynomial-time algorithm that turns any tree decomposition into a small tree decomposition.



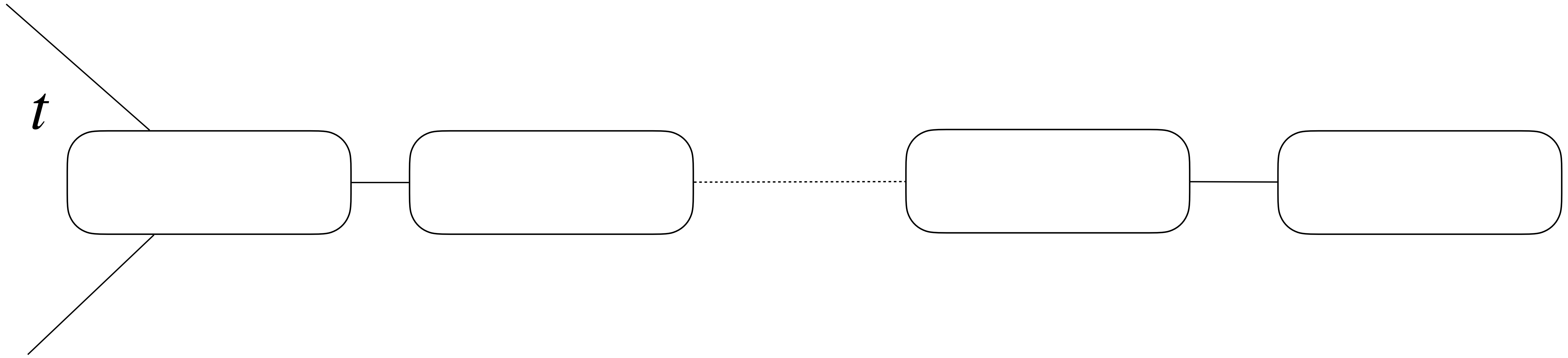
Small Tree Decompositions

Definition

A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Proposition

There is a polynomial-time algorithm that turns any tree decomposition into a small tree decomposition.



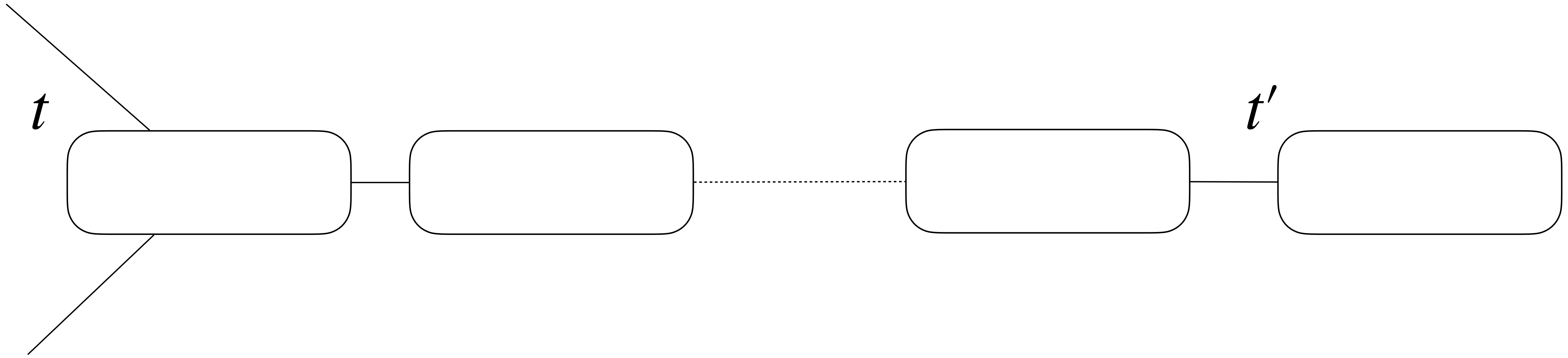
Small Tree Decompositions

Definition

A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Proposition

There is a polynomial-time algorithm that turns any tree decomposition into a small tree decomposition.



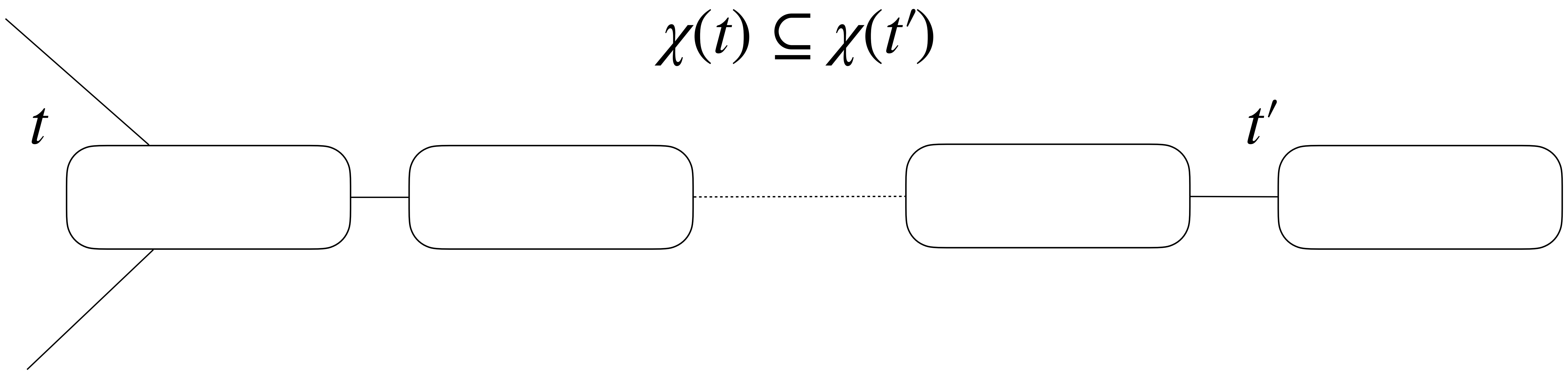
Small Tree Decompositions

Definition

A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Proposition

There is a polynomial-time algorithm that turns any tree decomposition into a small tree decomposition.



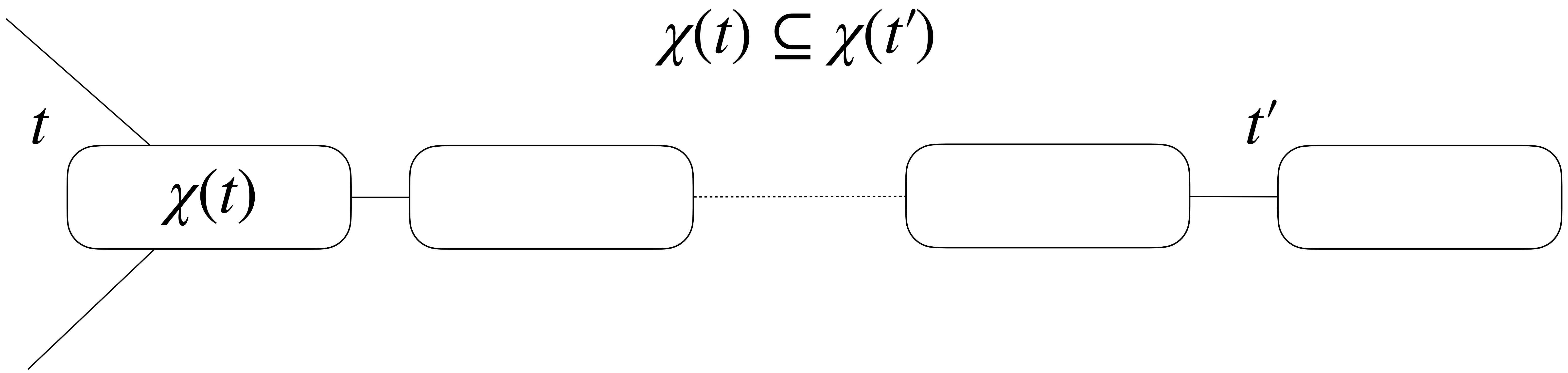
Small Tree Decompositions

Definition

A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Proposition

There is a polynomial-time algorithm that turns any tree decomposition into a small tree decomposition.



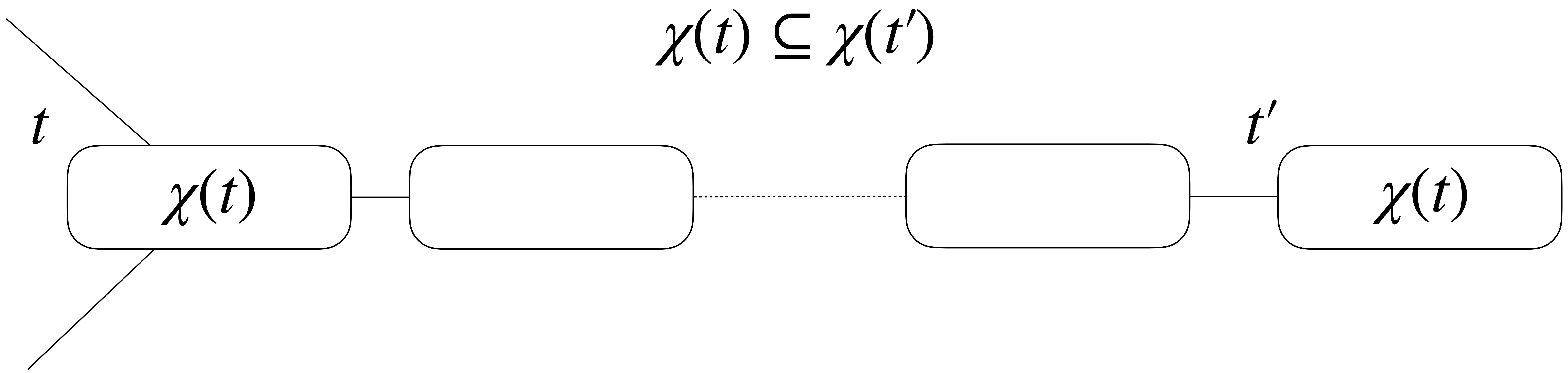
Small Tree Decompositions

Definition

A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Proposition

There is a polynomial-time algorithm that turns any tree decomposition into a small tree decomposition.



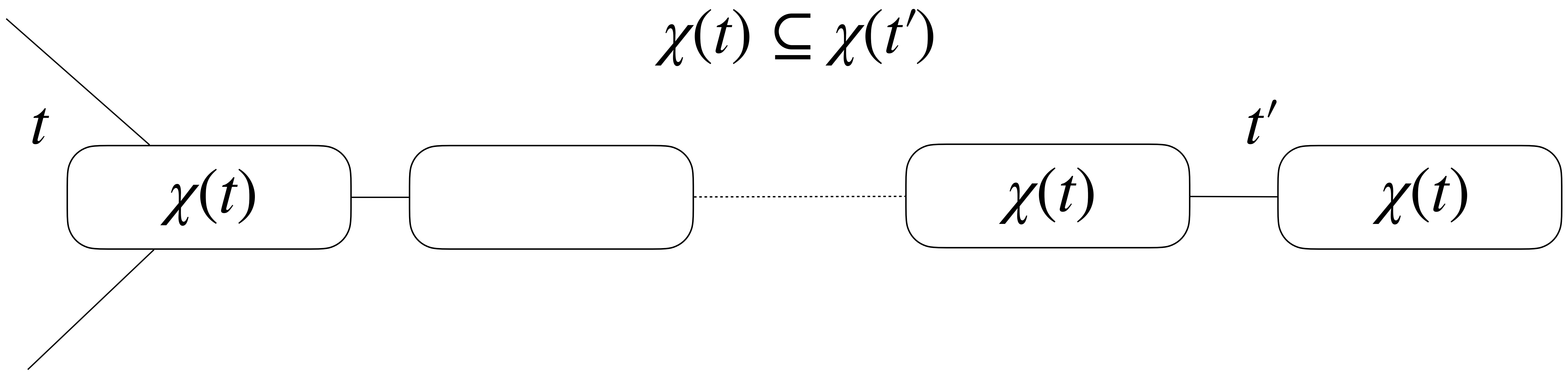
Small Tree Decompositions

Definition

A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Proposition

There is a polynomial-time algorithm that turns any tree decomposition into a small tree decomposition.



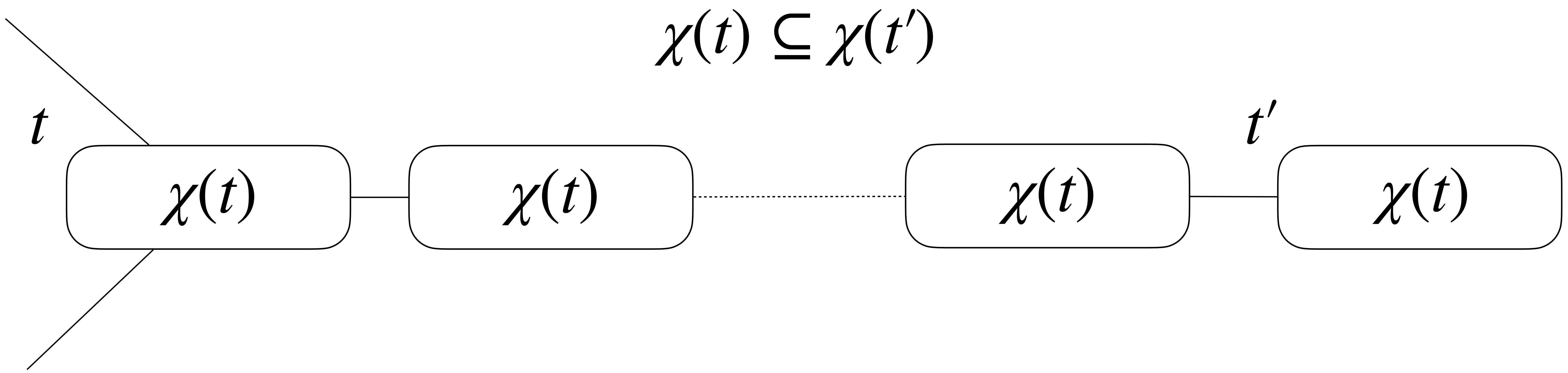
Small Tree Decompositions

Definition

A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Proposition

There is a polynomial-time algorithm that turns any tree decomposition into a small tree decomposition.



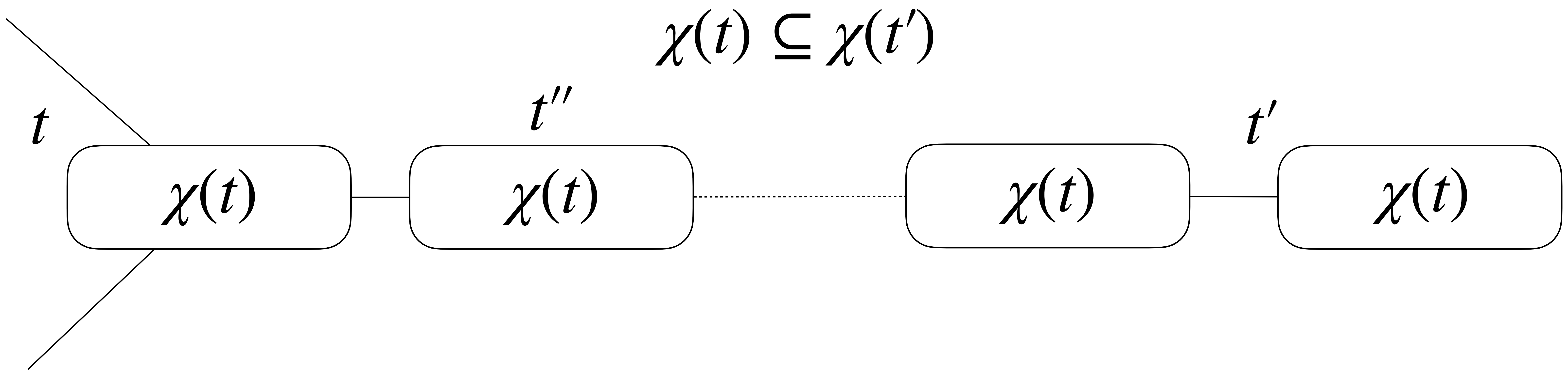
Small Tree Decompositions

Definition

A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Proposition

There is a polynomial-time algorithm that turns any tree decomposition into a small tree decomposition.



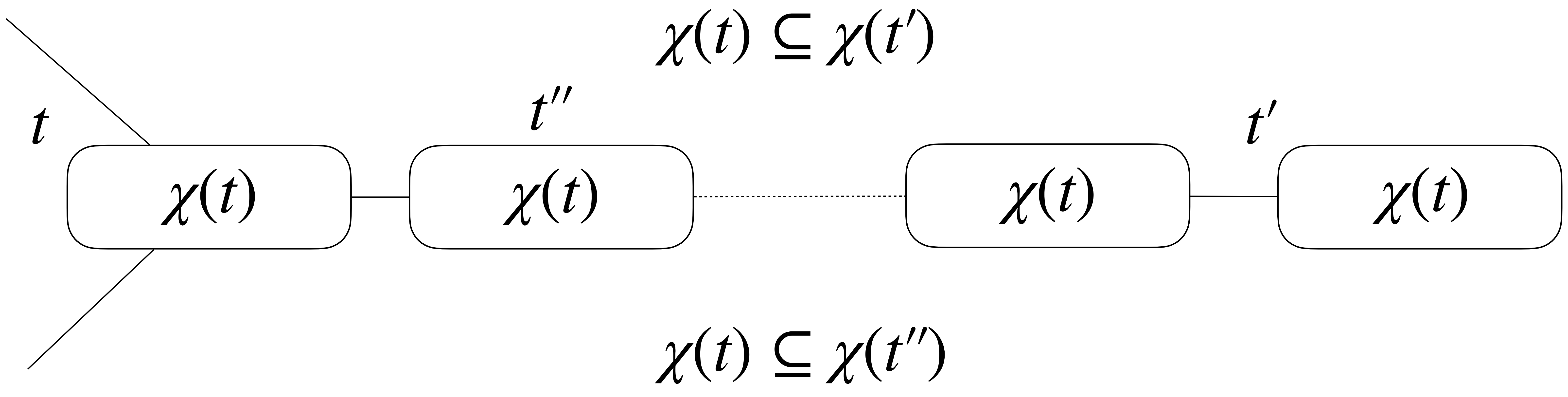
Small Tree Decompositions

Definition

A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Proposition

There is a polynomial-time algorithm that turns any tree decomposition into a small tree decomposition.



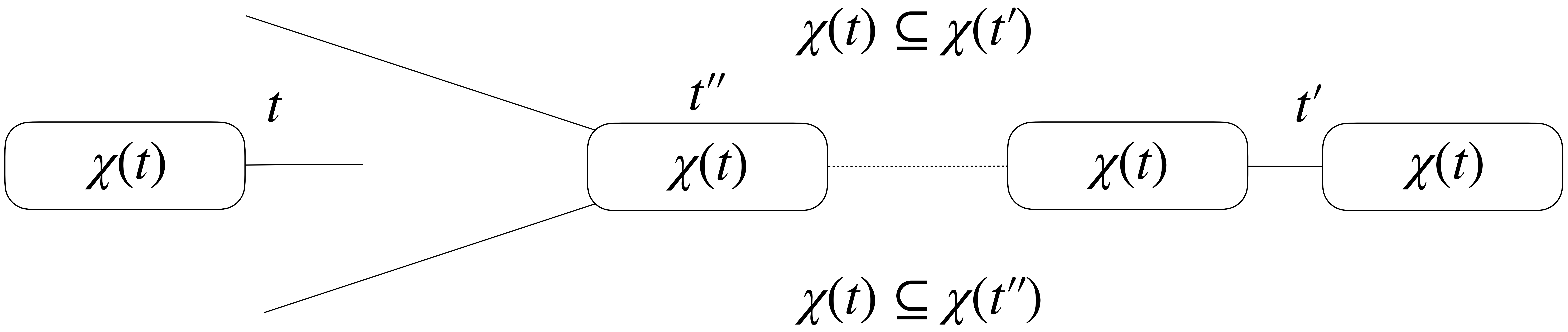
Small Tree Decompositions

Definition

A tree decomposition (T, χ) is **small** if there is no pair t, t' of distinct nodes such that $\chi(t) \subseteq \chi(t')$.

Proposition

There is a polynomial-time algorithm that turns any tree decomposition into a small tree decomposition.



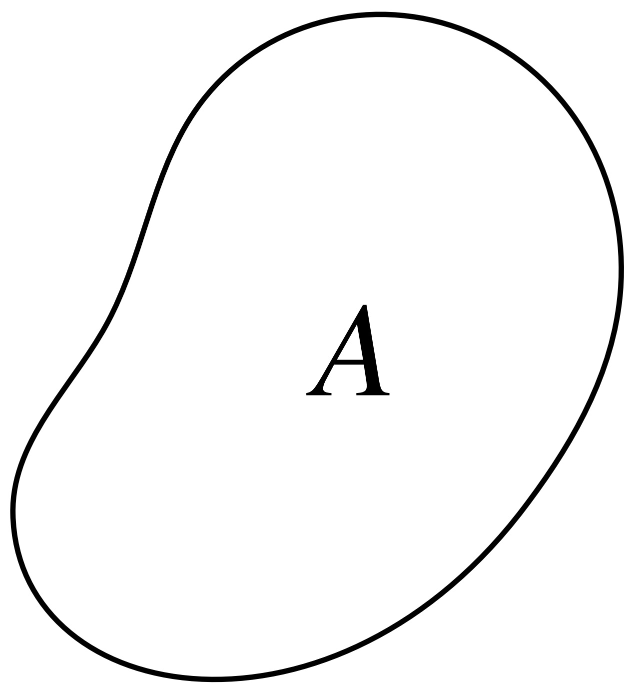
Separators

Separators

Let $G = (V, E)$ be a graph. The set $X \subseteq V$ **separates** $A \subseteq V$ and $B \subseteq V$ if every $A - B$ path contains a vertex from X.

Separators

Let $G = (V, E)$ be a graph. The set $X \subseteq V$ **separates** $A \subseteq V$ and $B \subseteq V$ if every $A - B$ path contains a vertex from X.

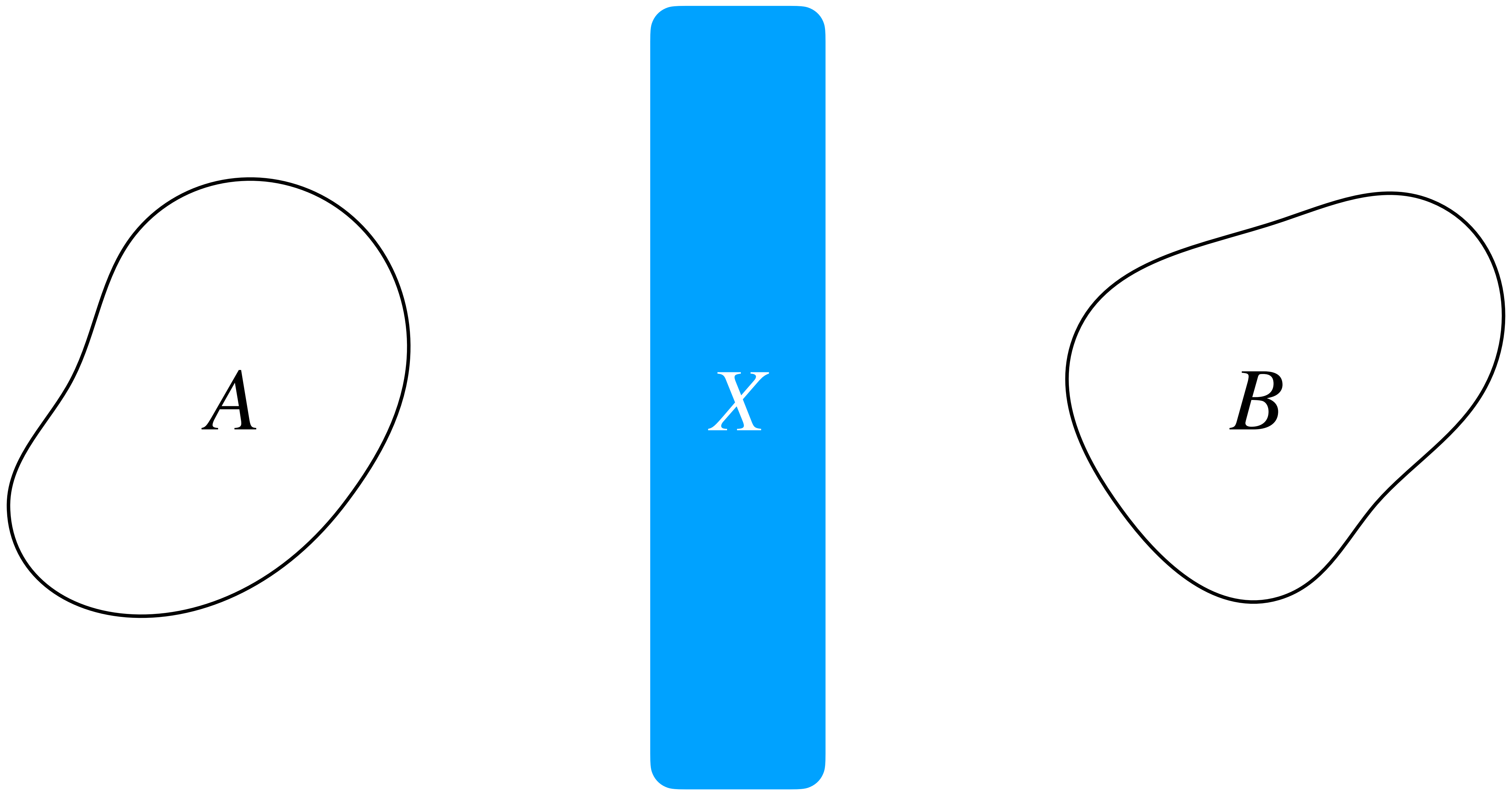


Separators

Let $G = (V, E)$ be a graph. The set $X \subseteq V$ **separates** $A \subseteq V$ and $B \subseteq V$ if every $A - B$ path contains a vertex from X.

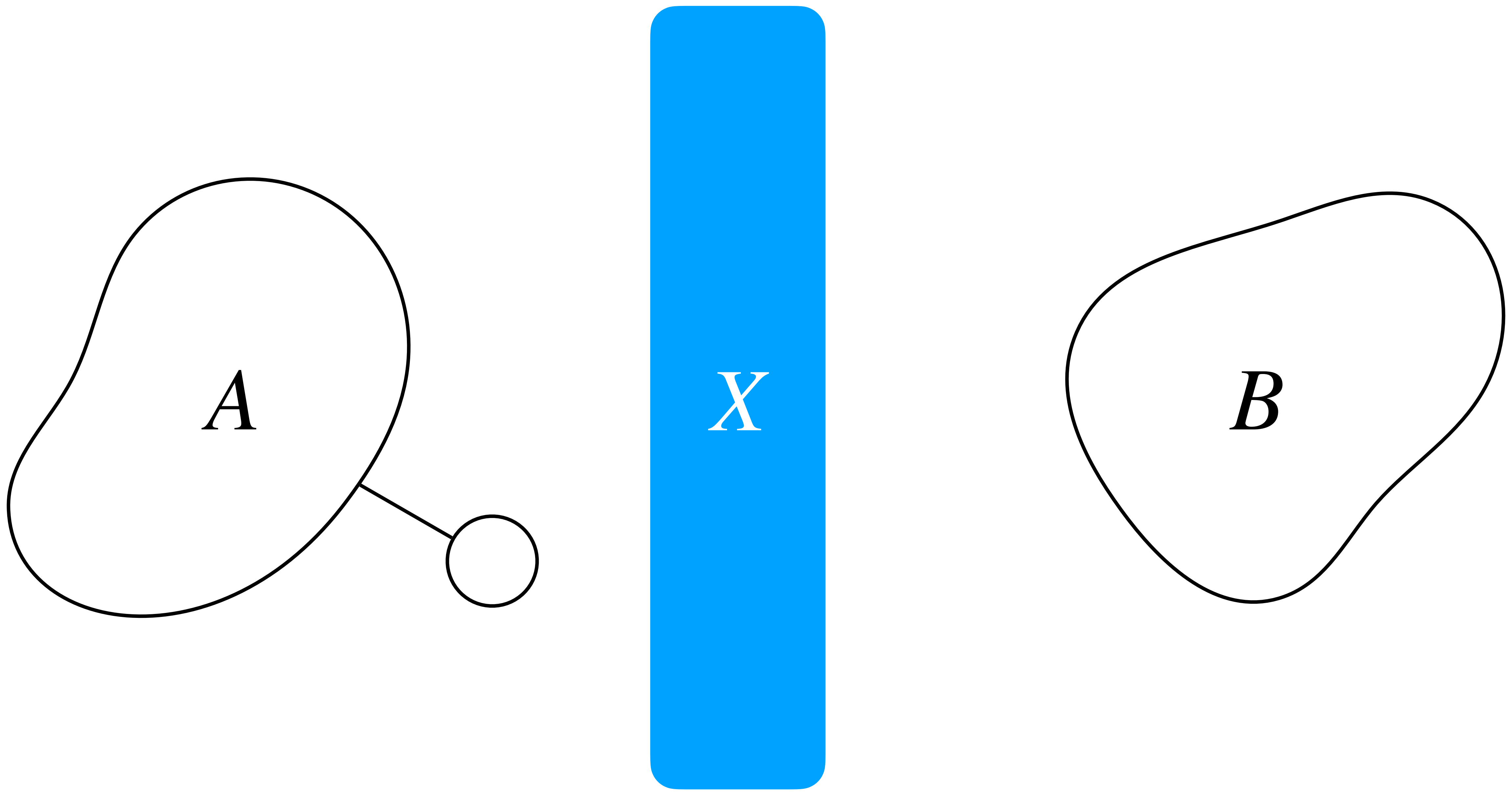
Separators

Let $G = (V, E)$ be a graph. The set $X \subseteq V$ **separates** $A \subseteq V$ and $B \subseteq V$ if every $A - B$ path contains a vertex from X.



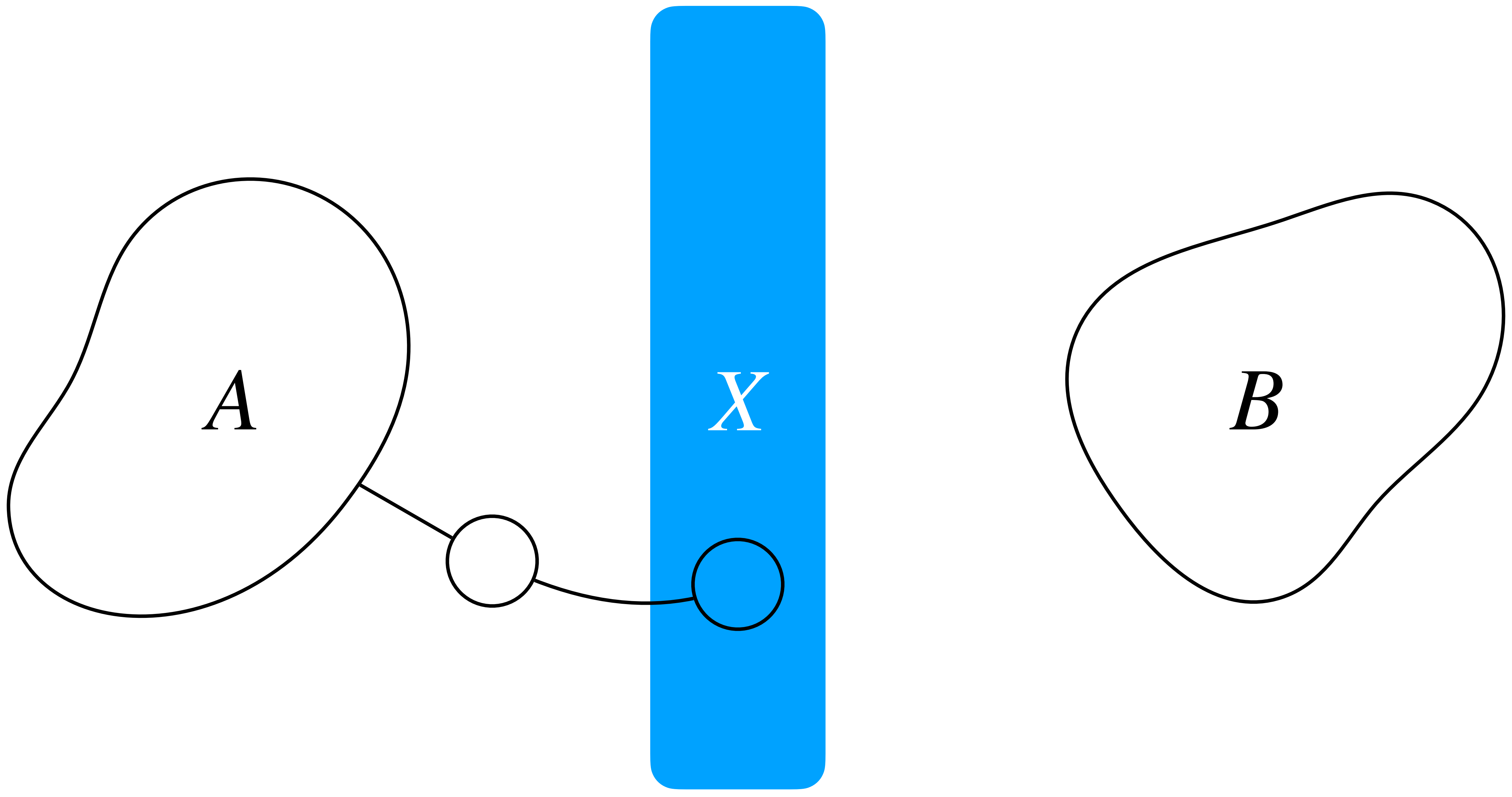
Separators

Let $G = (V, E)$ be a graph. The set $X \subseteq V$ **separates** $A \subseteq V$ and $B \subseteq V$ if every $A - B$ path contains a vertex from X.



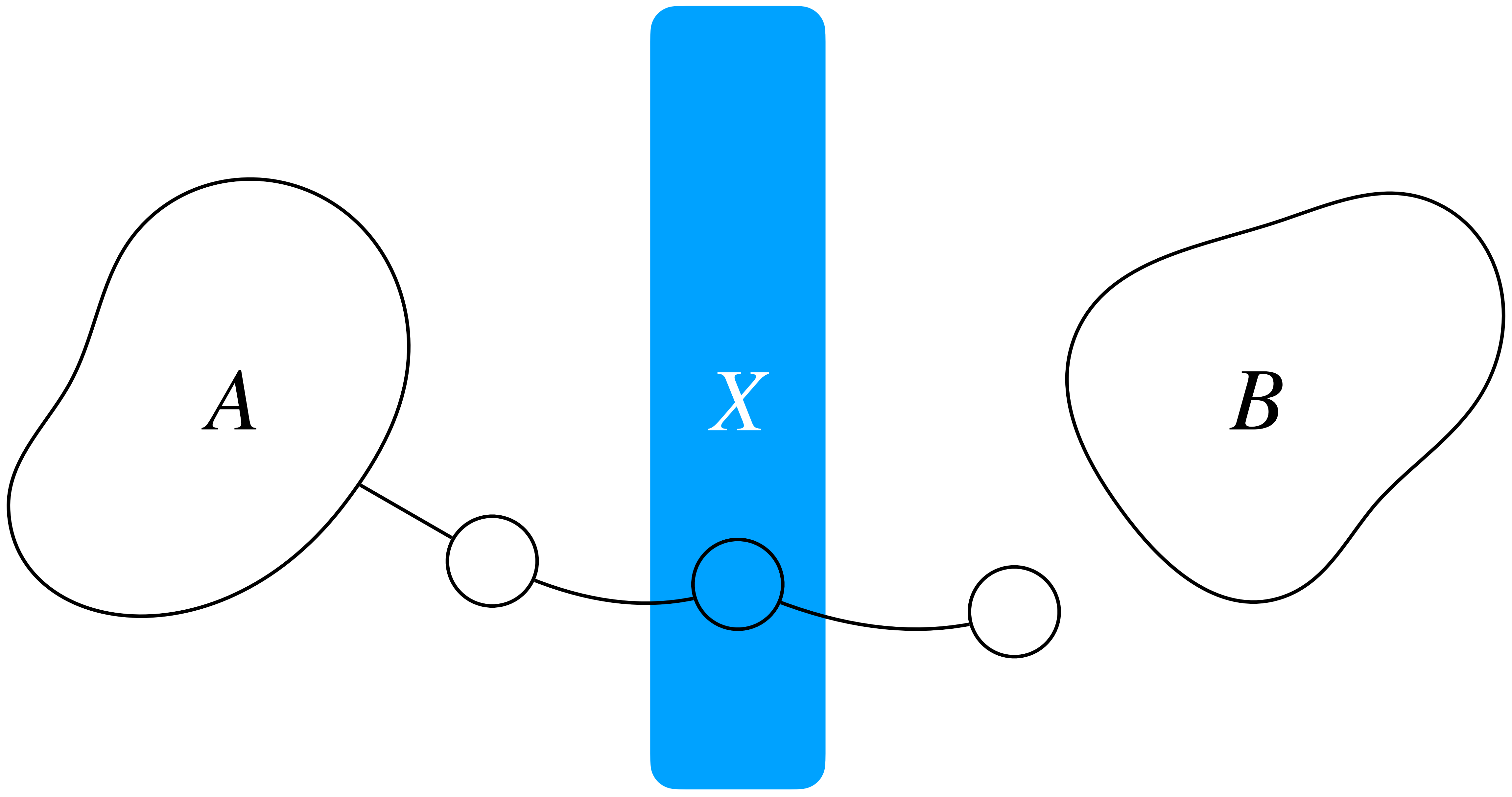
Separators

Let $G = (V, E)$ be a graph. The set $X \subseteq V$ **separates** $A \subseteq V$ and $B \subseteq V$ if every $A - B$ path contains a vertex from X.



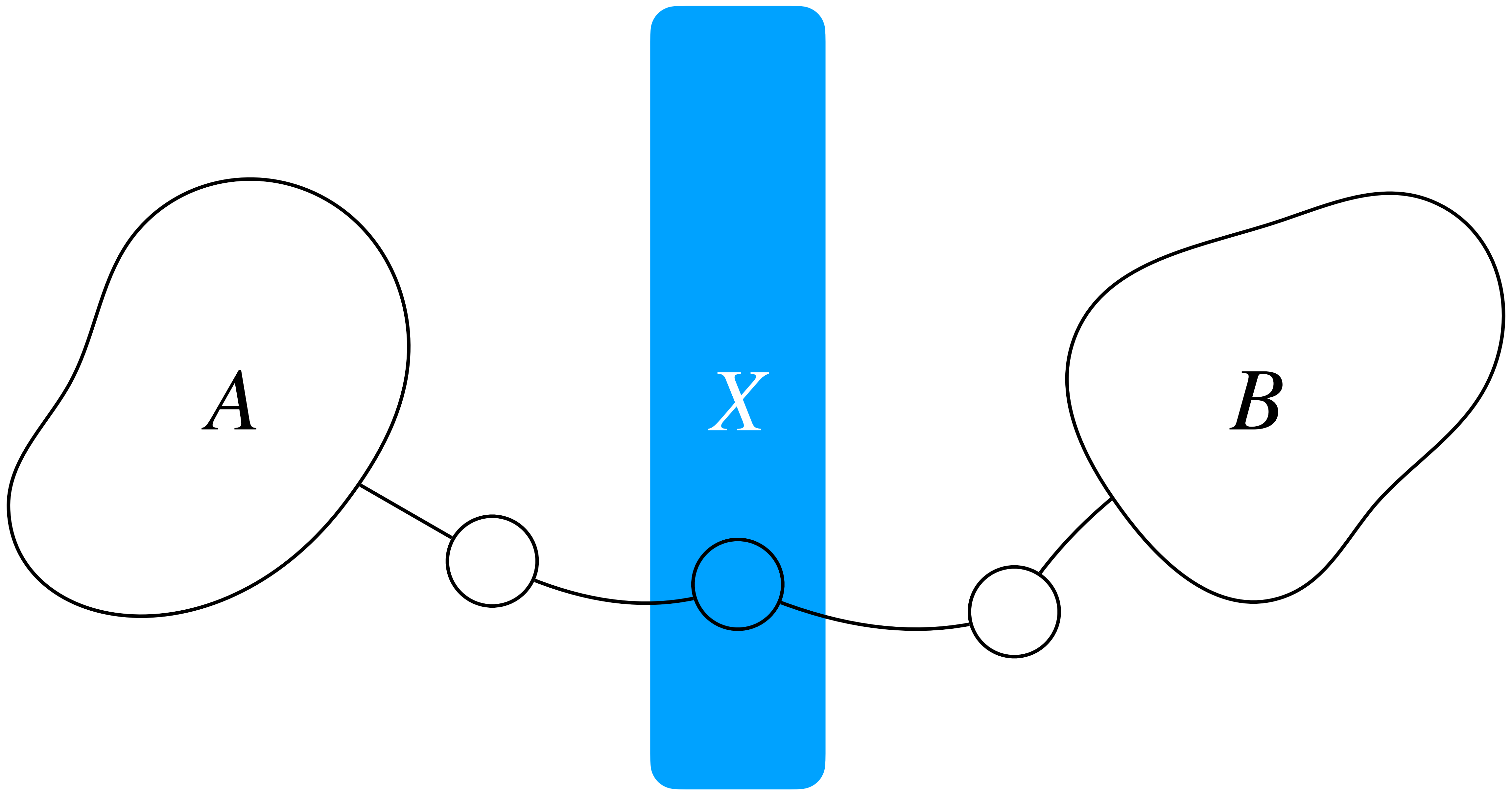
Separators

Let $G = (V, E)$ be a graph. The set $X \subseteq V$ **separates** $A \subseteq V$ and $B \subseteq V$ if every $A - B$ path contains a vertex from X.



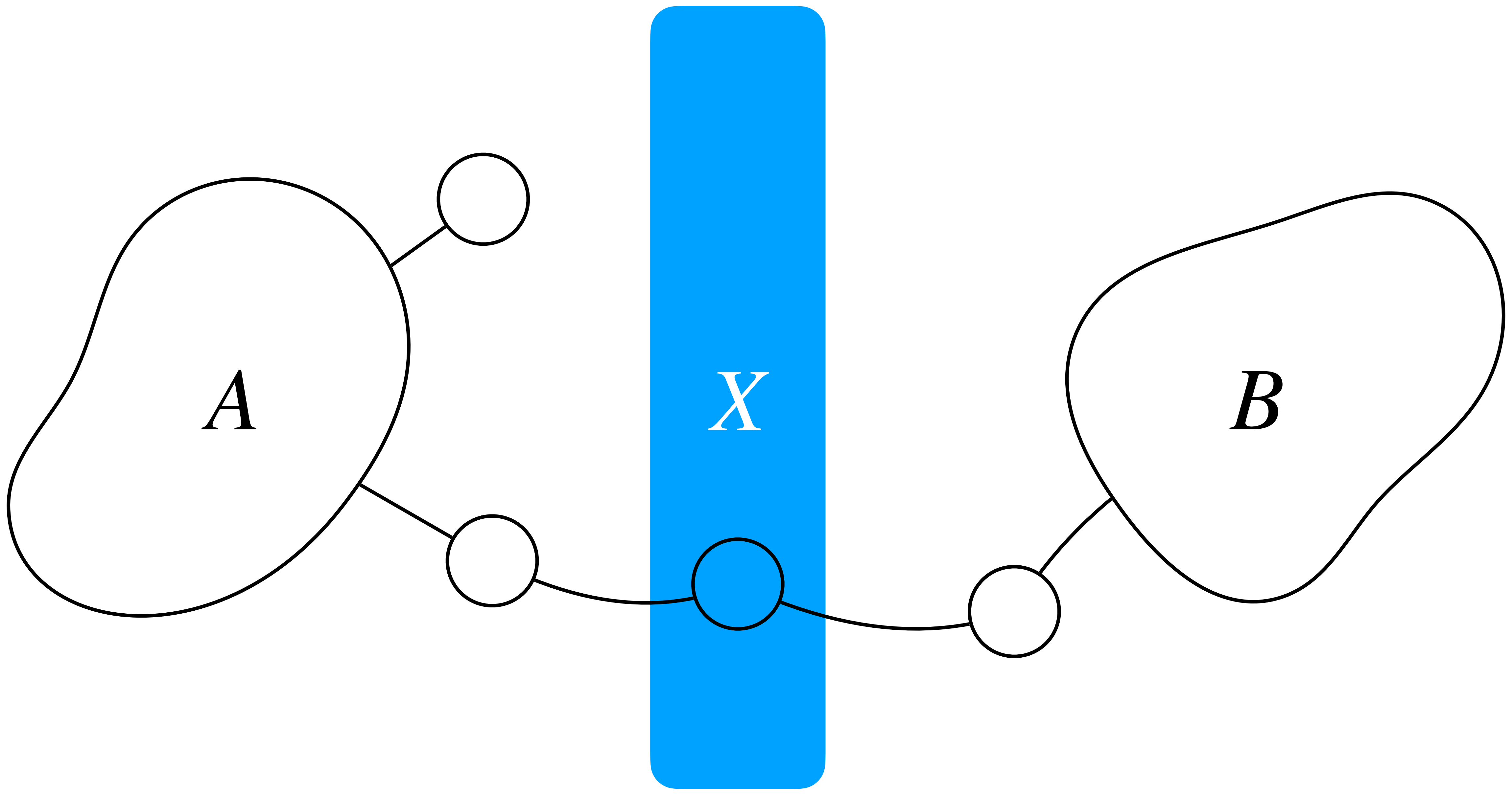
Separators

Let $G = (V, E)$ be a graph. The set $X \subseteq V$ **separates** $A \subseteq V$ and $B \subseteq V$ if every $A - B$ path contains a vertex from X.



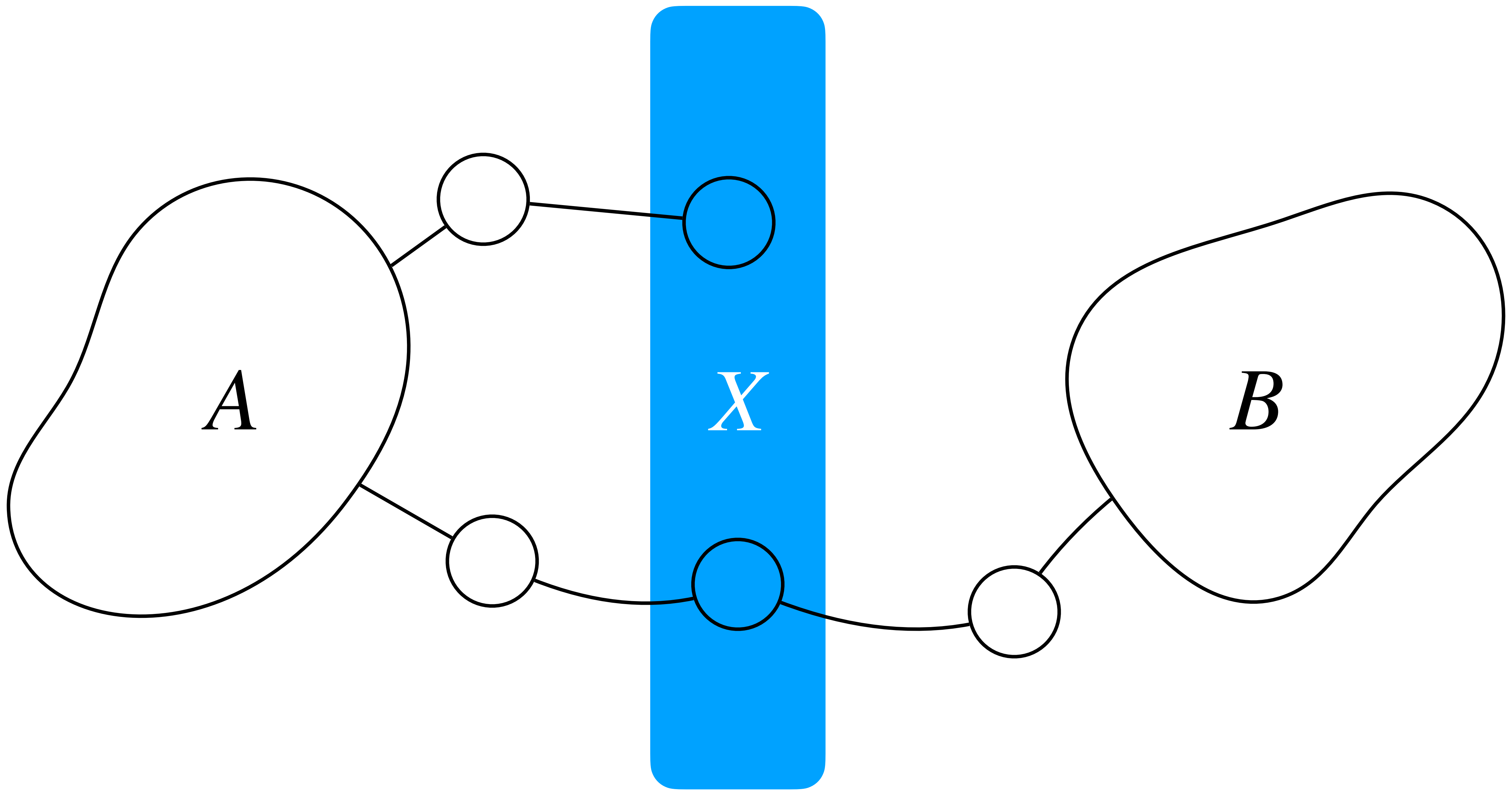
Separators

Let $G = (V, E)$ be a graph. The set $X \subseteq V$ **separates** $A \subseteq V$ and $B \subseteq V$ if every $A - B$ path contains a vertex from X.



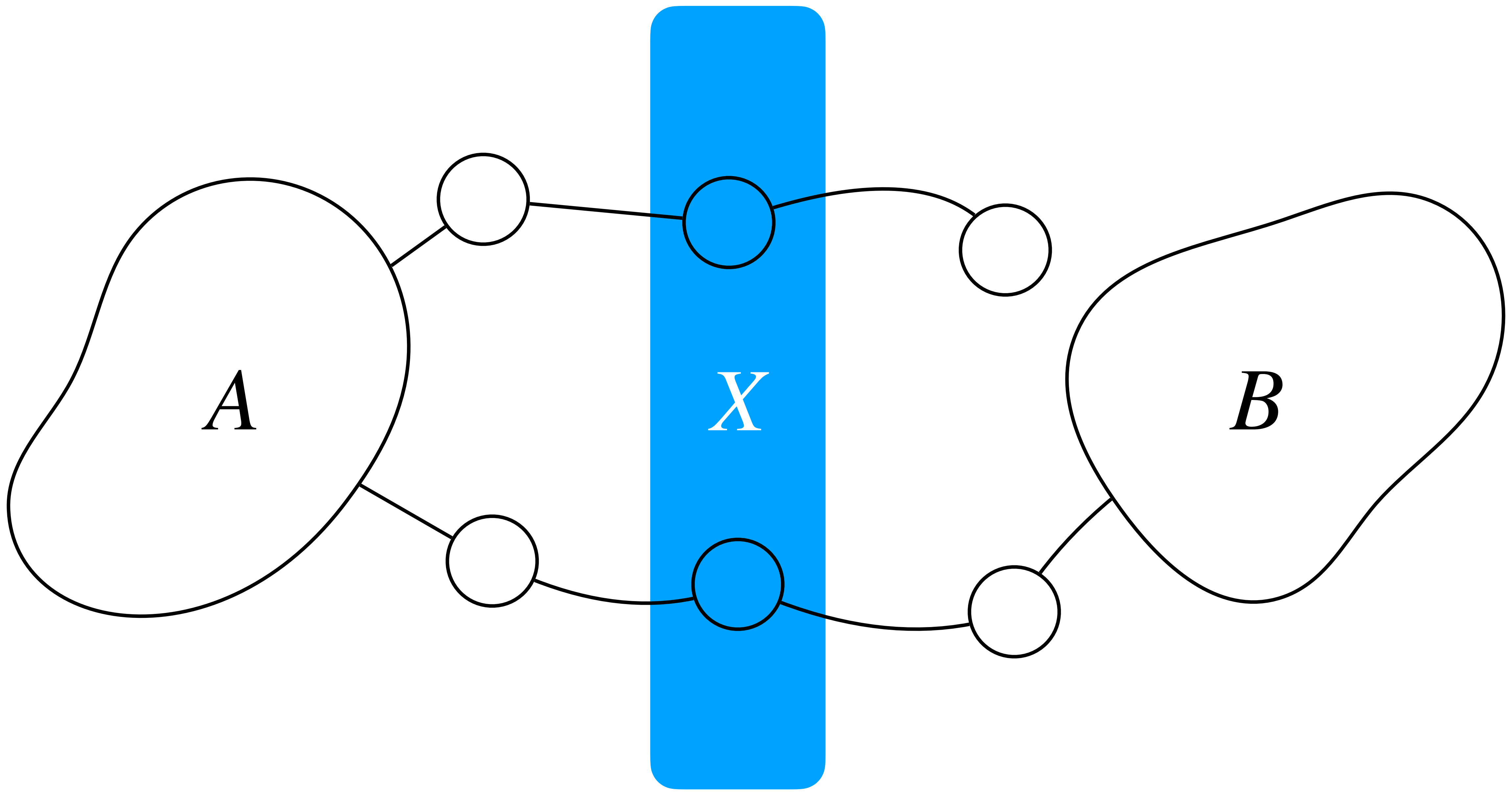
Separators

Let $G = (V, E)$ be a graph. The set $X \subseteq V$ **separates** $A \subseteq V$ and $B \subseteq V$ if every $A - B$ path contains a vertex from X.



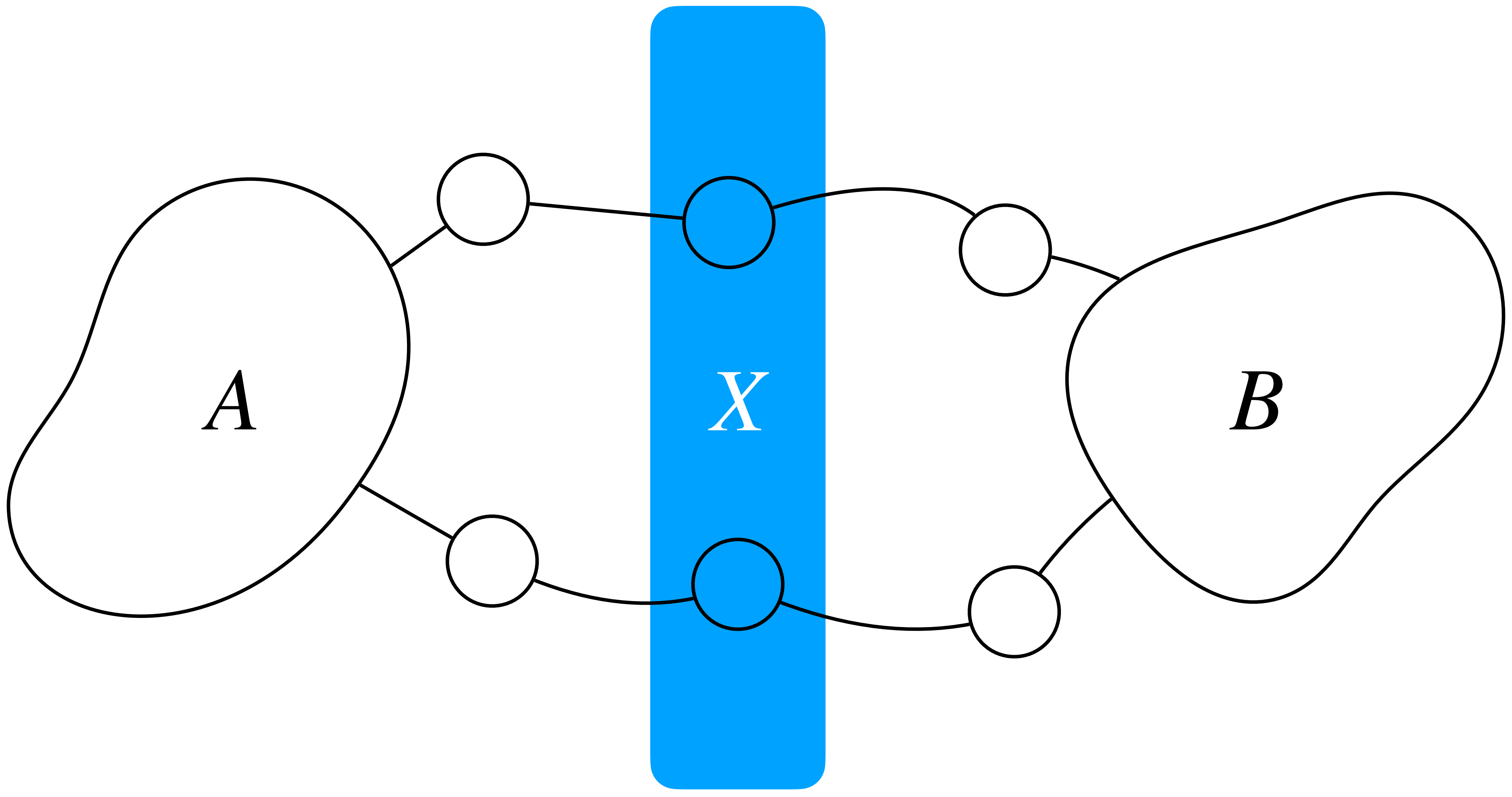
Separators

Let $G = (V, E)$ be a graph. The set $X \subseteq V$ **separates** $A \subseteq V$ and $B \subseteq V$ if every $A - B$ path contains a vertex from X.



Separators

Let $G = (V, E)$ be a graph. The set $X \subseteq V$ **separates** $A \subseteq V$ and $B \subseteq V$ if every $A - B$ path contains a vertex from X.



Bags are Separators

Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.

Bags are Separators

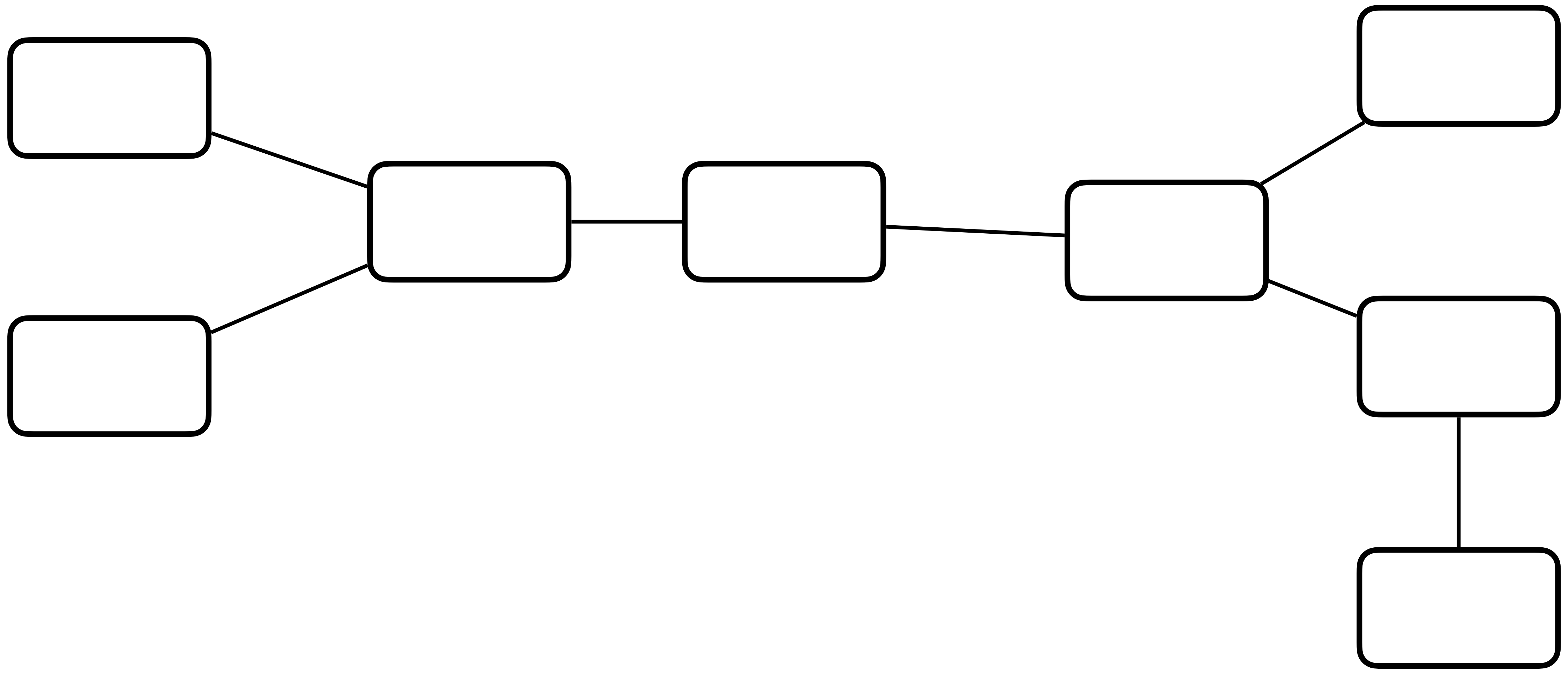
Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.

Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.

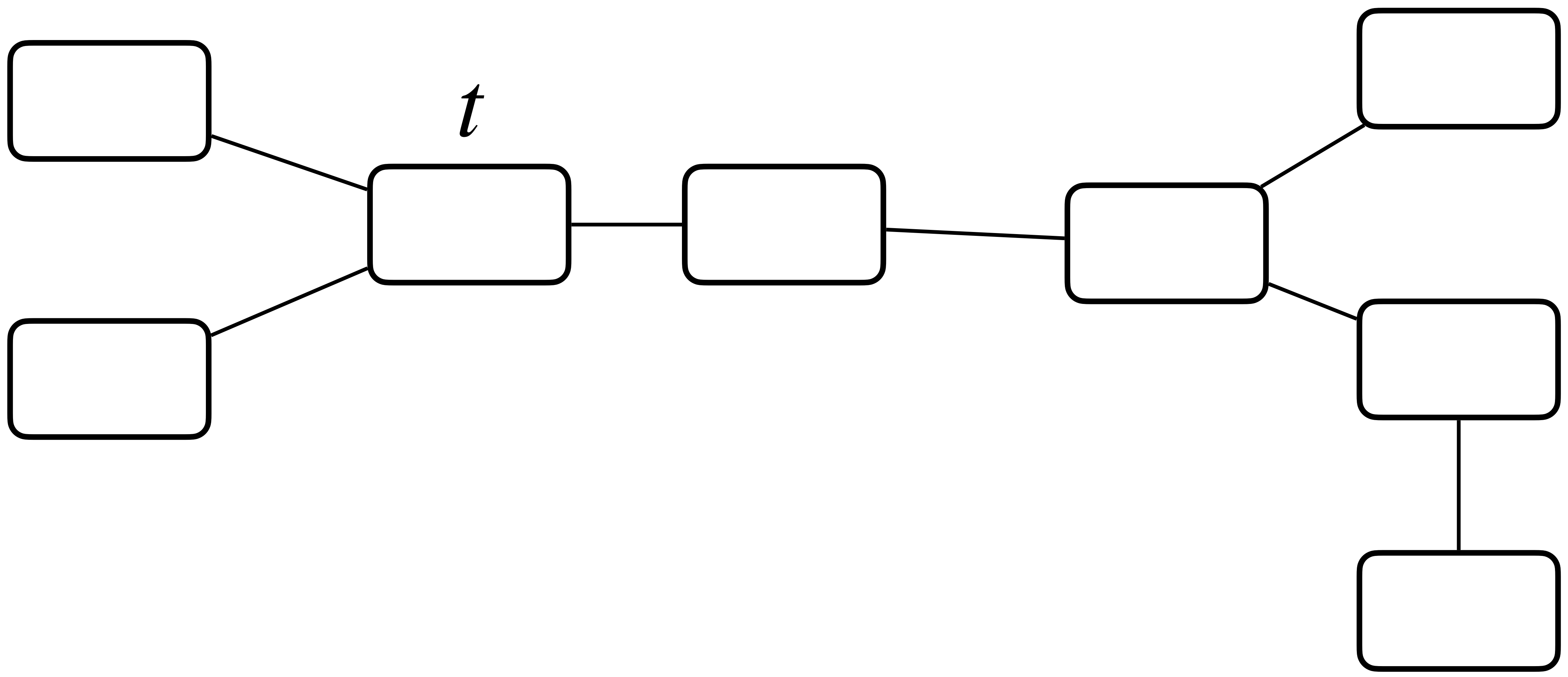
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



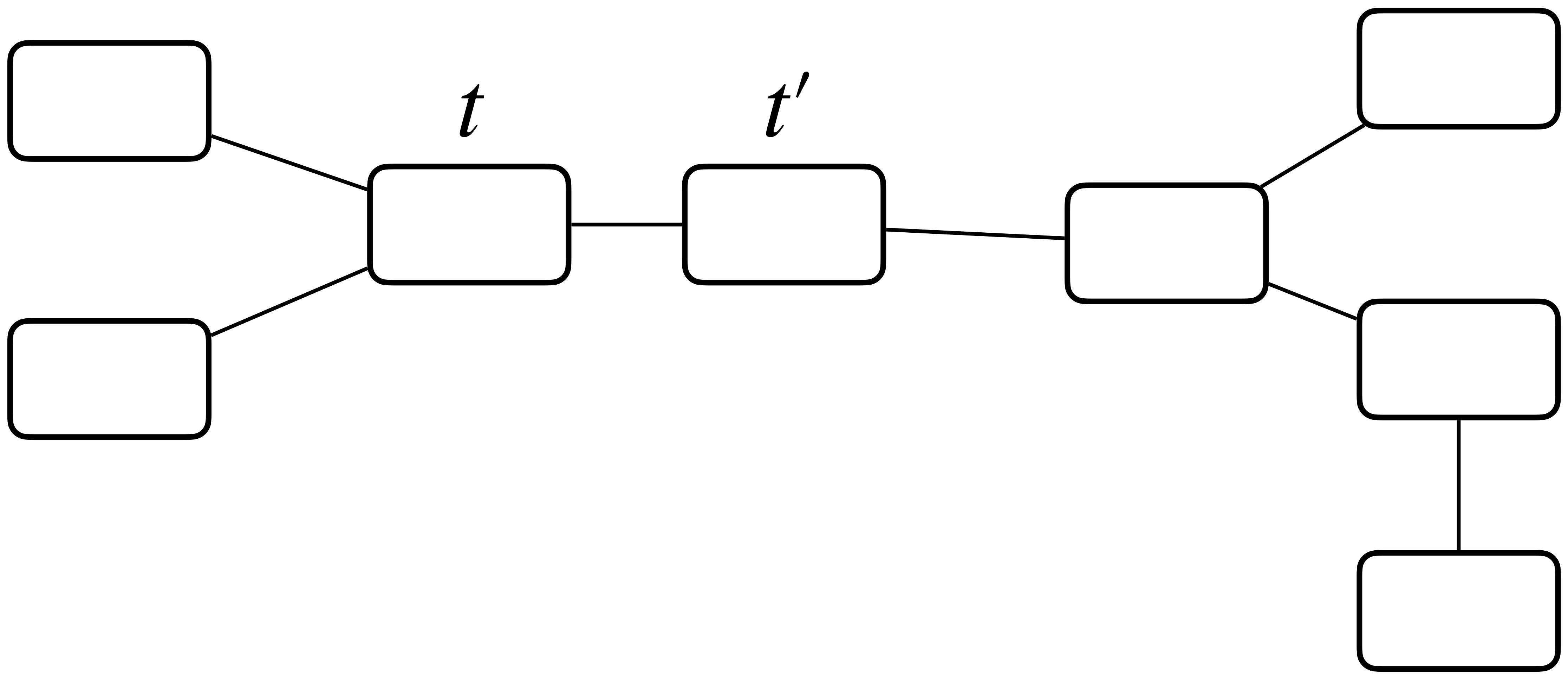
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



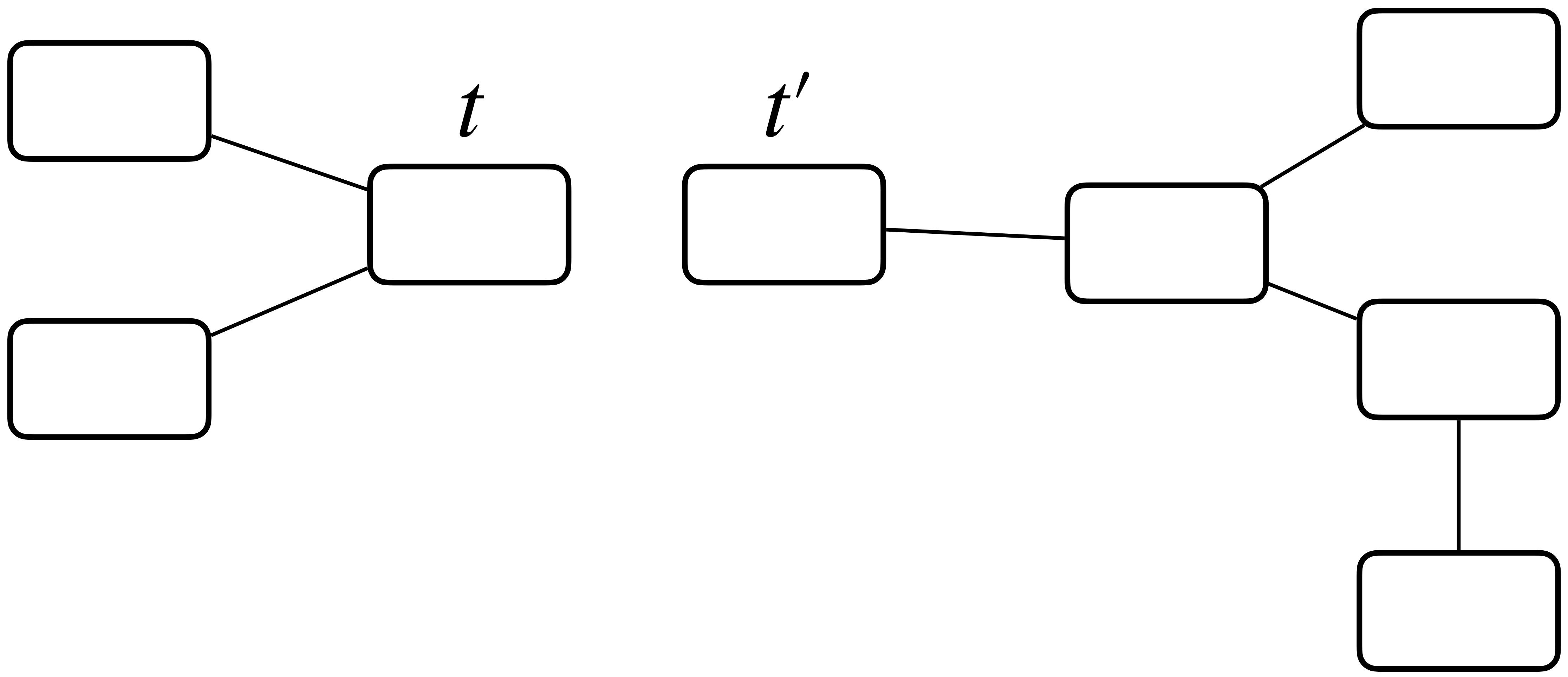
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



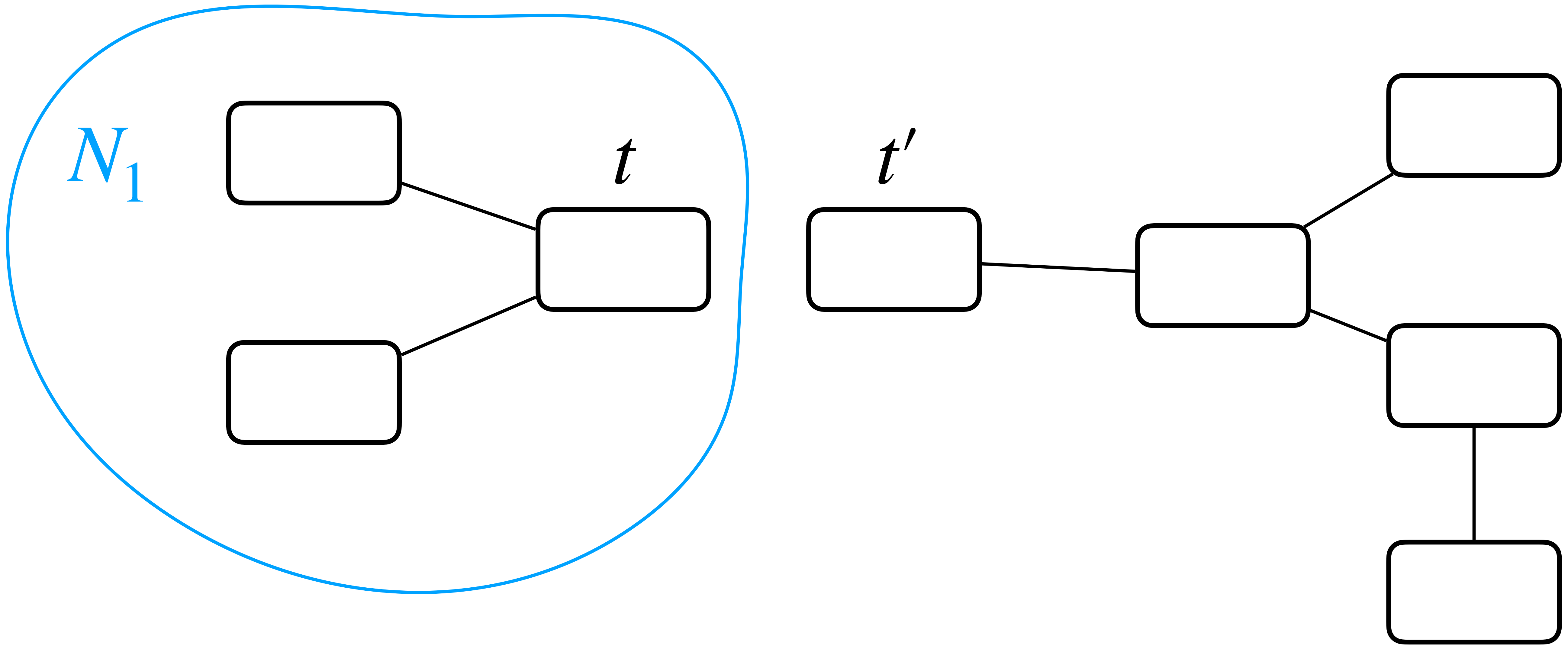
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



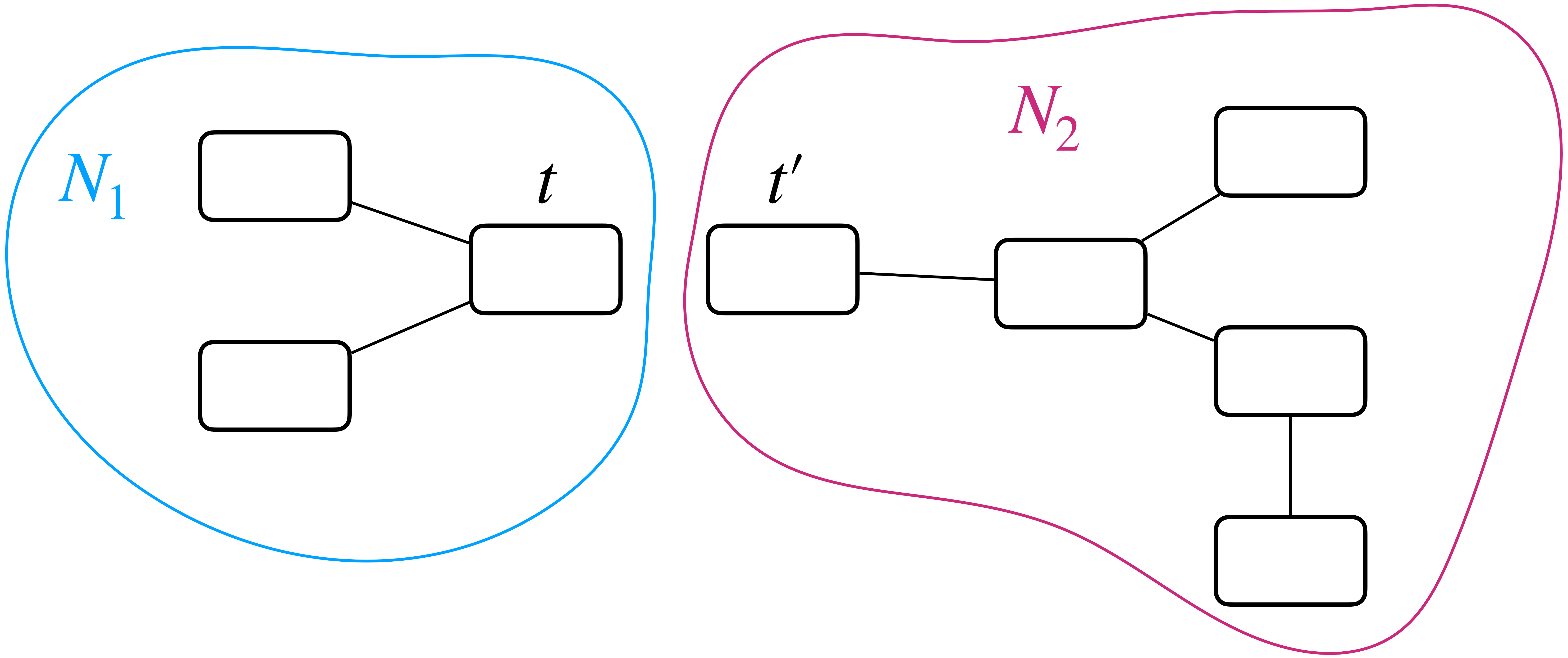
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



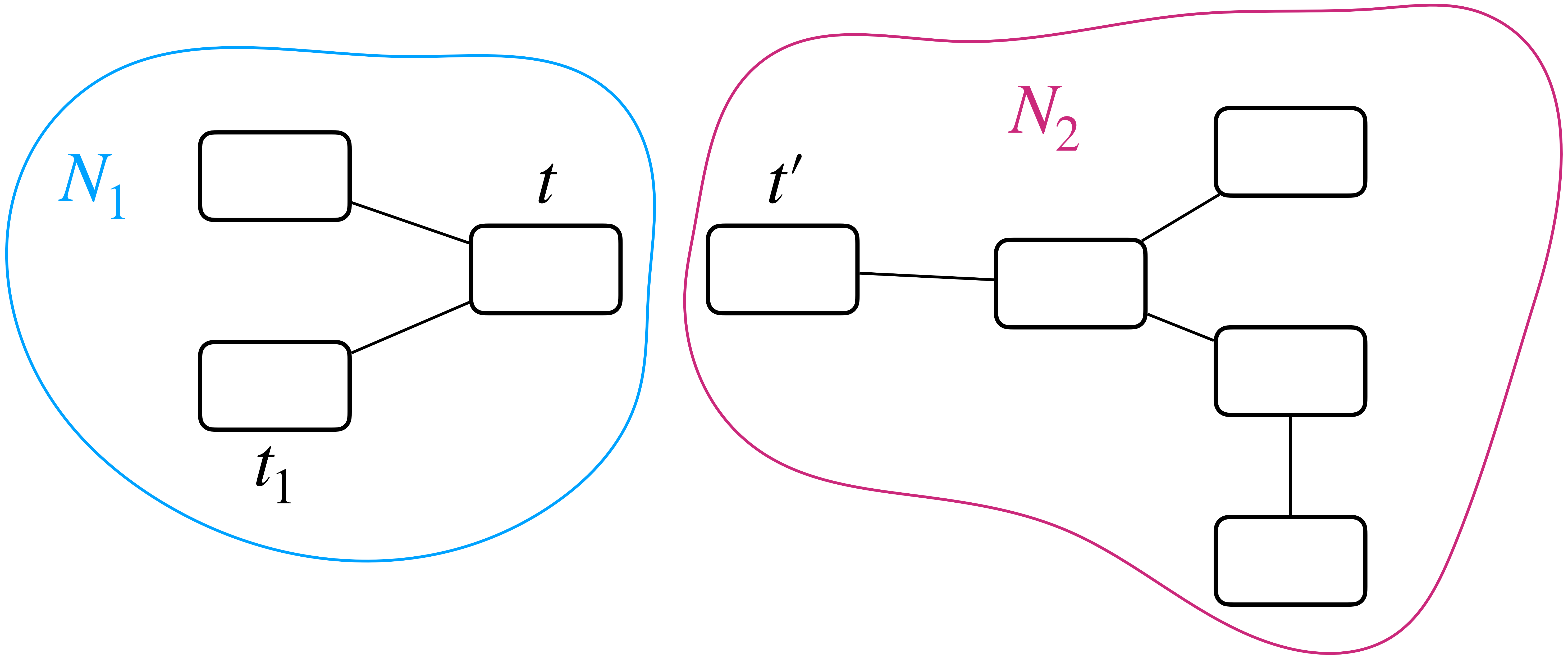
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



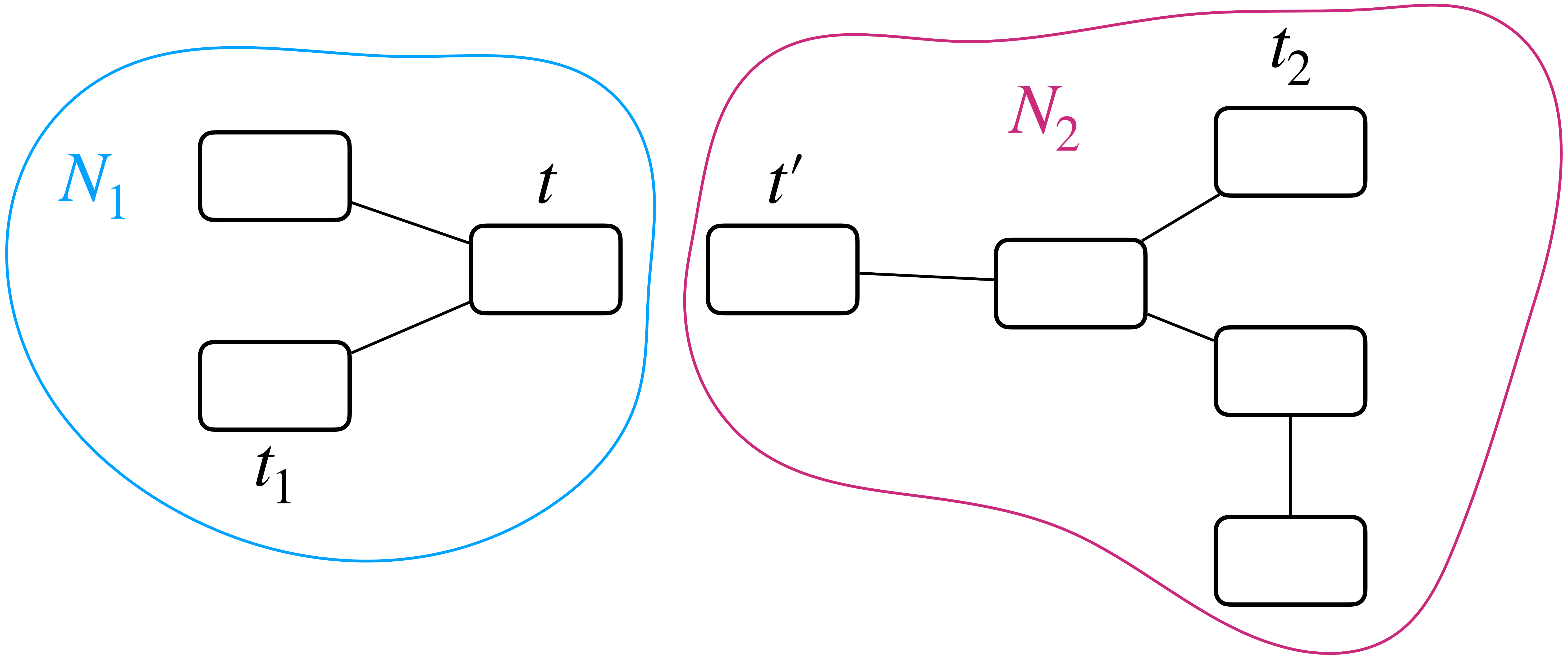
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



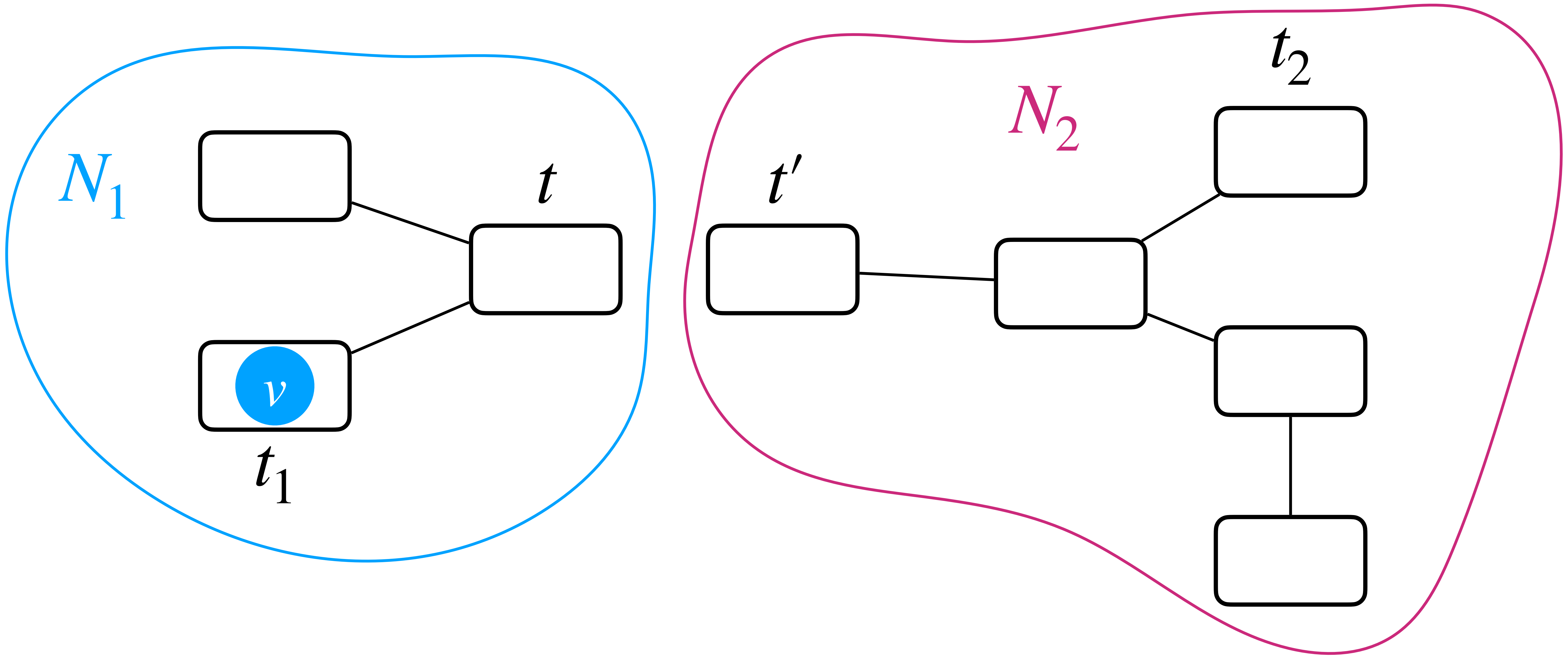
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



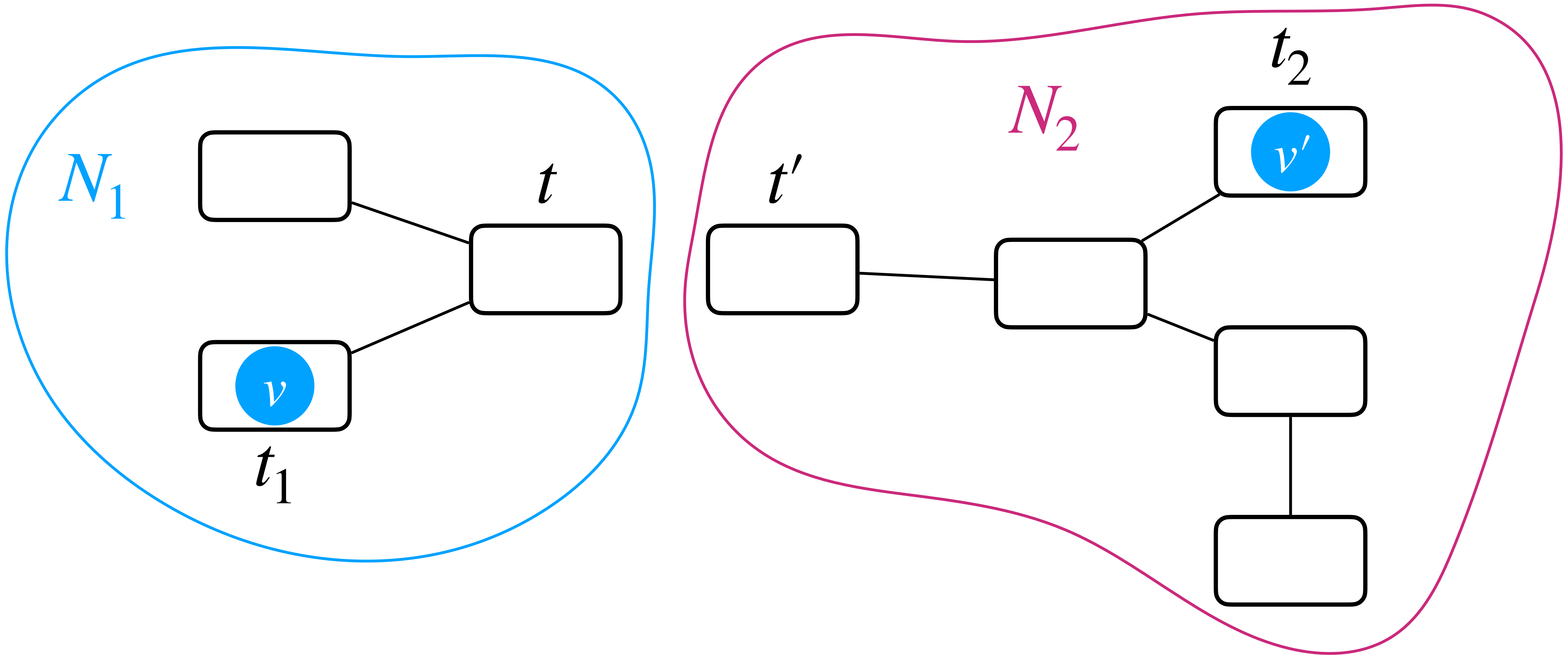
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



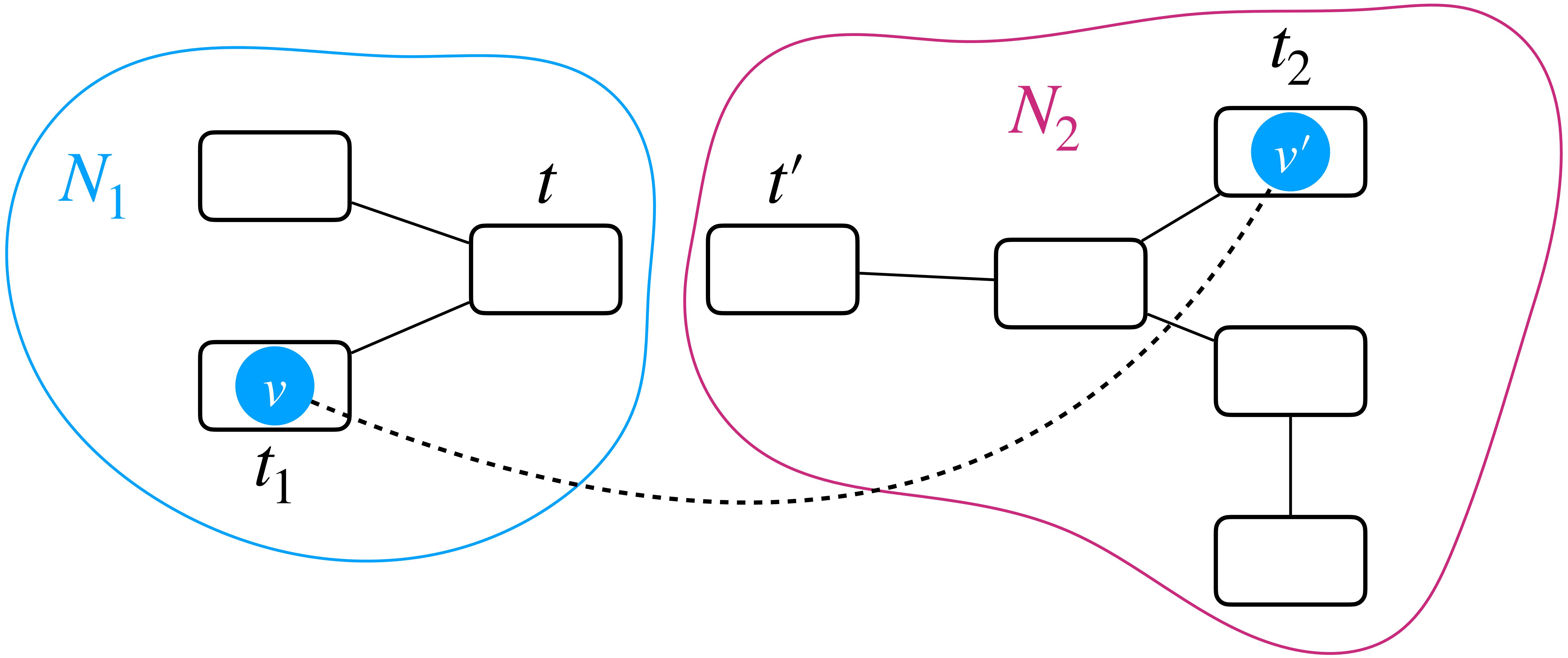
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



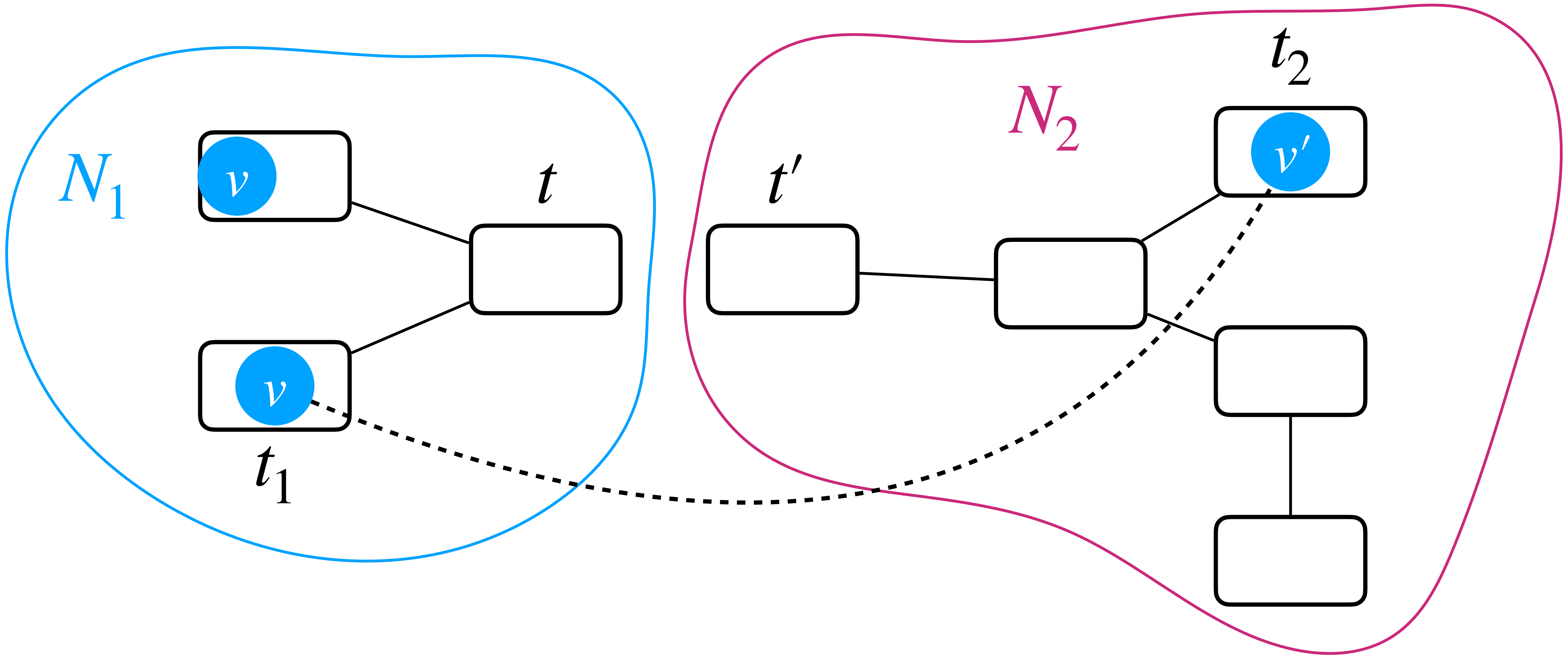
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



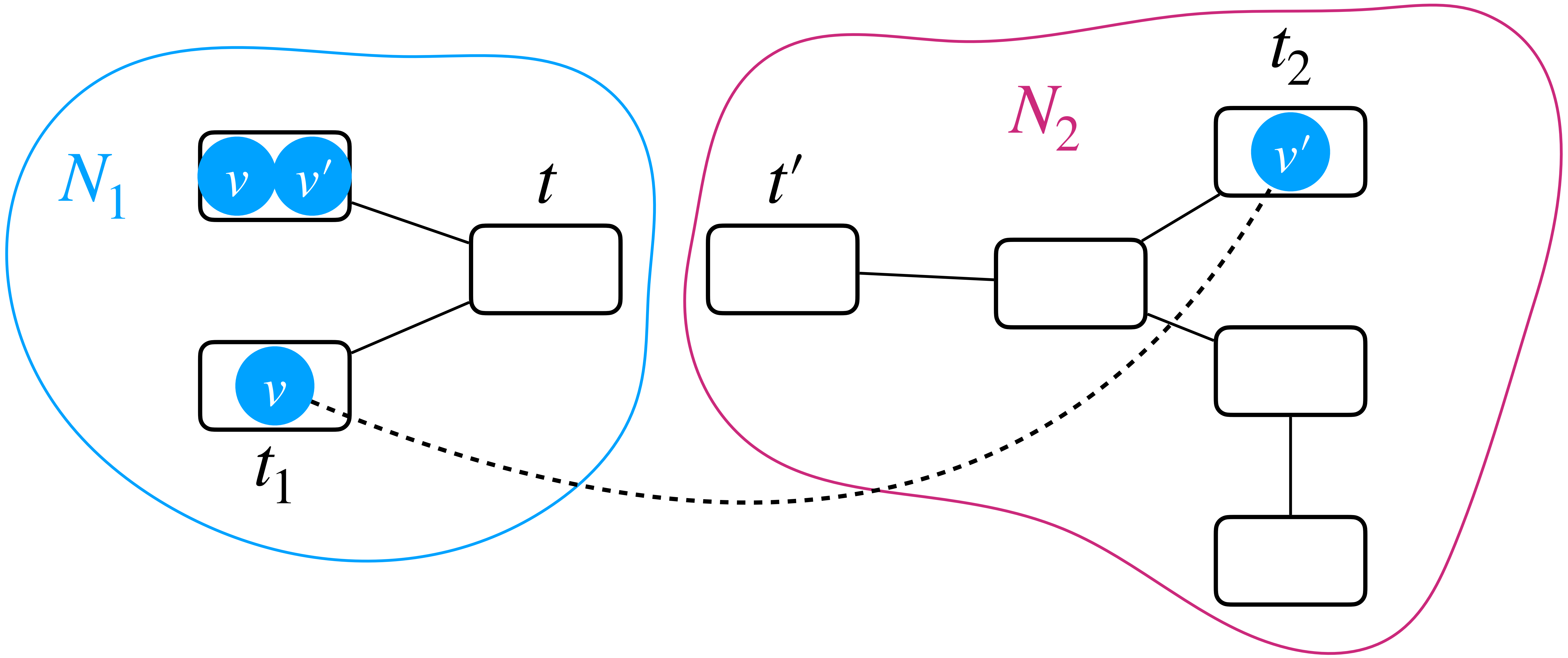
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



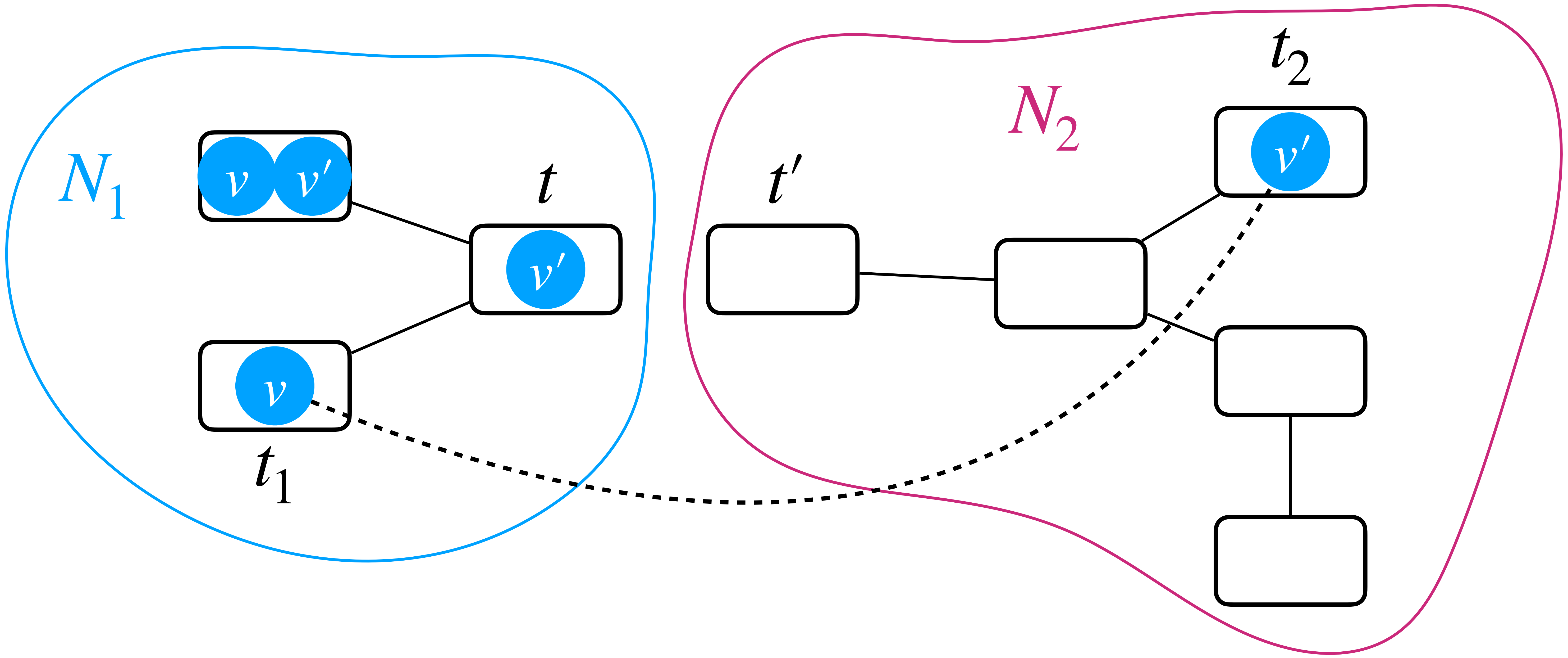
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



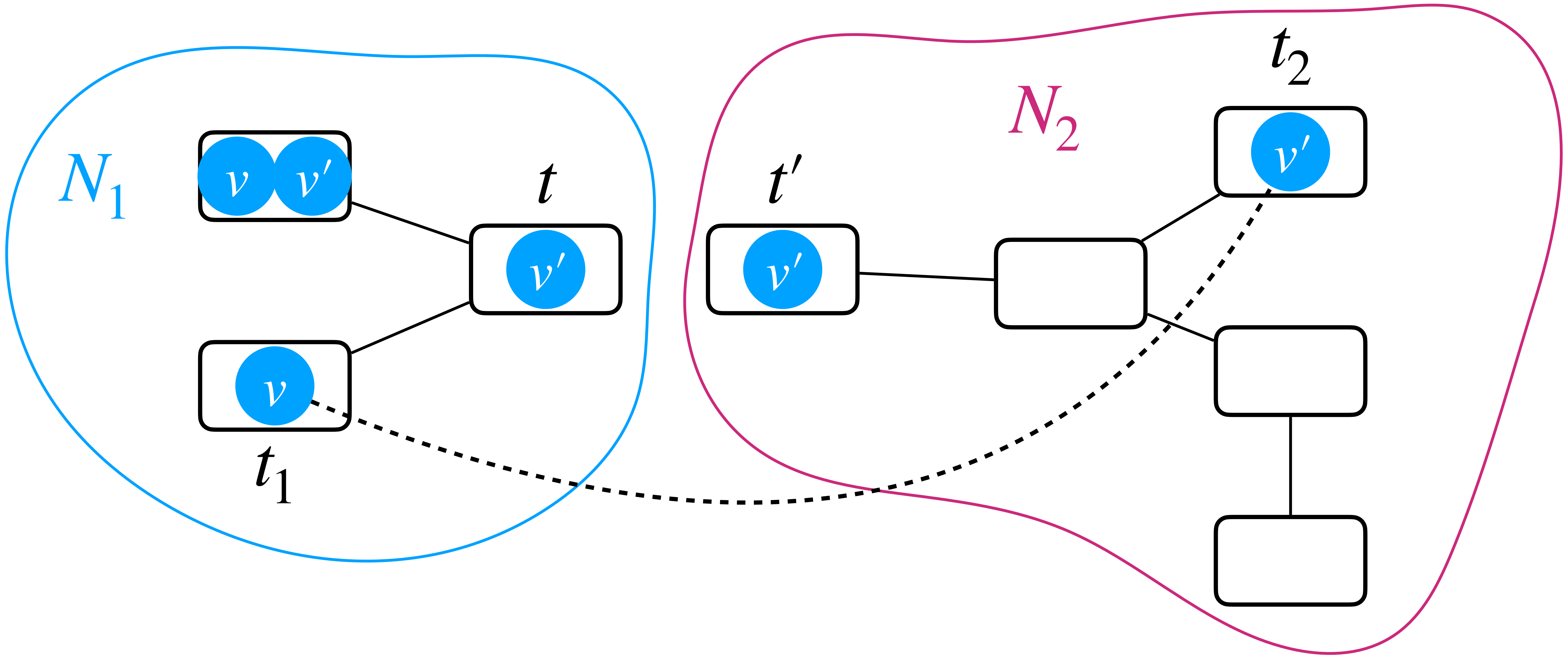
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



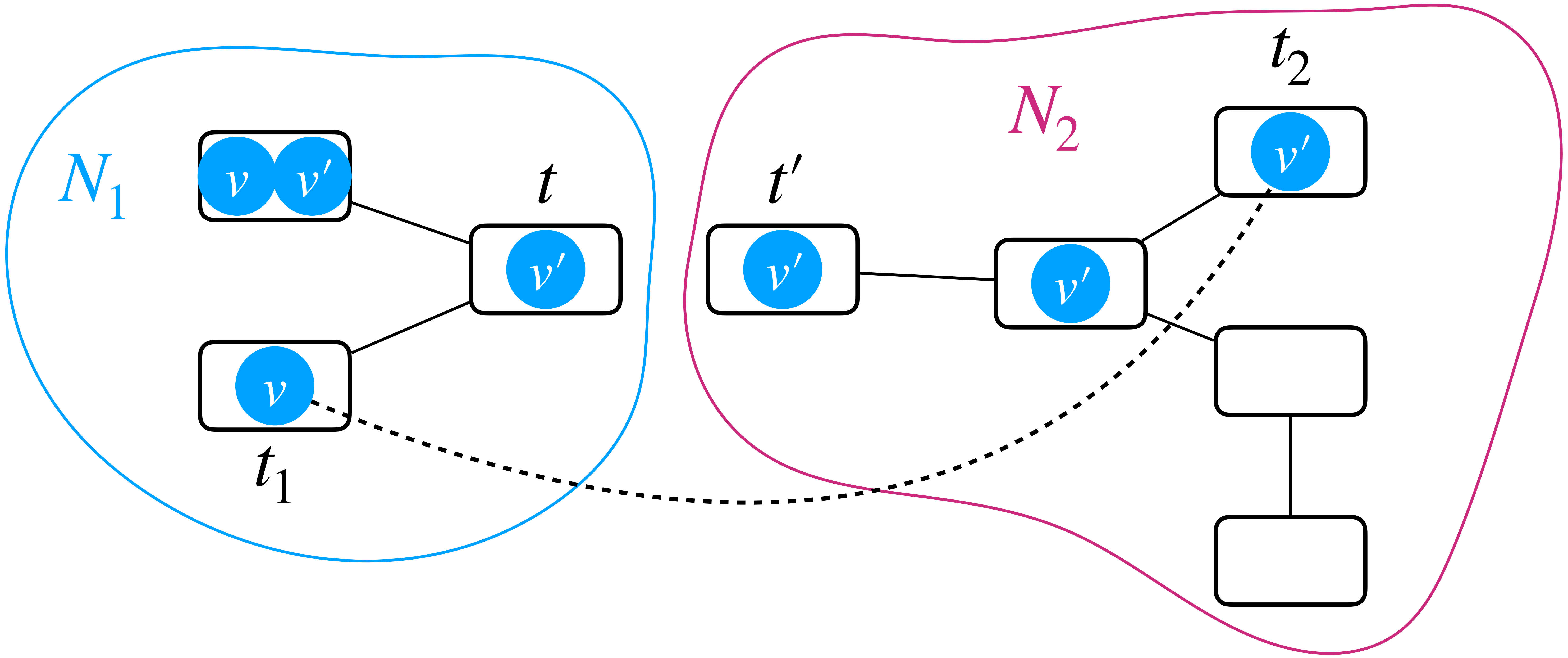
Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



Bags are Separators

Let (T, χ) be a tree decomposition of G, and t, t' distinct nodes of T.
Let N_1 and N_2 denote the components of $T - tt'$.
Then $\chi(t) \cap \chi(t')$ separates $V_1 = \bigcup_{t_1 \in N_1} \chi(t_1)$ and $V_2 = \bigcup_{t_2 \in N_1} \chi(t_2)$.



Graph Minors and Treewidth

Graph Minors

Definition

Let G be a graph. A graph H is called a **minor** of G if it can be obtained from G by successive applications of the following operations:

Graph Minors

Definition

Let G be a graph. A graph H is called a **minor** of G if it can be obtained from G by successive applications of the following operations:

1. Deleting an edge.

Graph Minors

Definition

Let G be a graph. A graph H is called a **minor** of G if it can be obtained from G by successive applications of the following operations:

1. Deleting an edge.
2. Deleting an isolated vertex.

Graph Minors

Definition

Let G be a graph. A graph H is called a **minor** of G if it can be obtained from G by successive applications of the following operations:

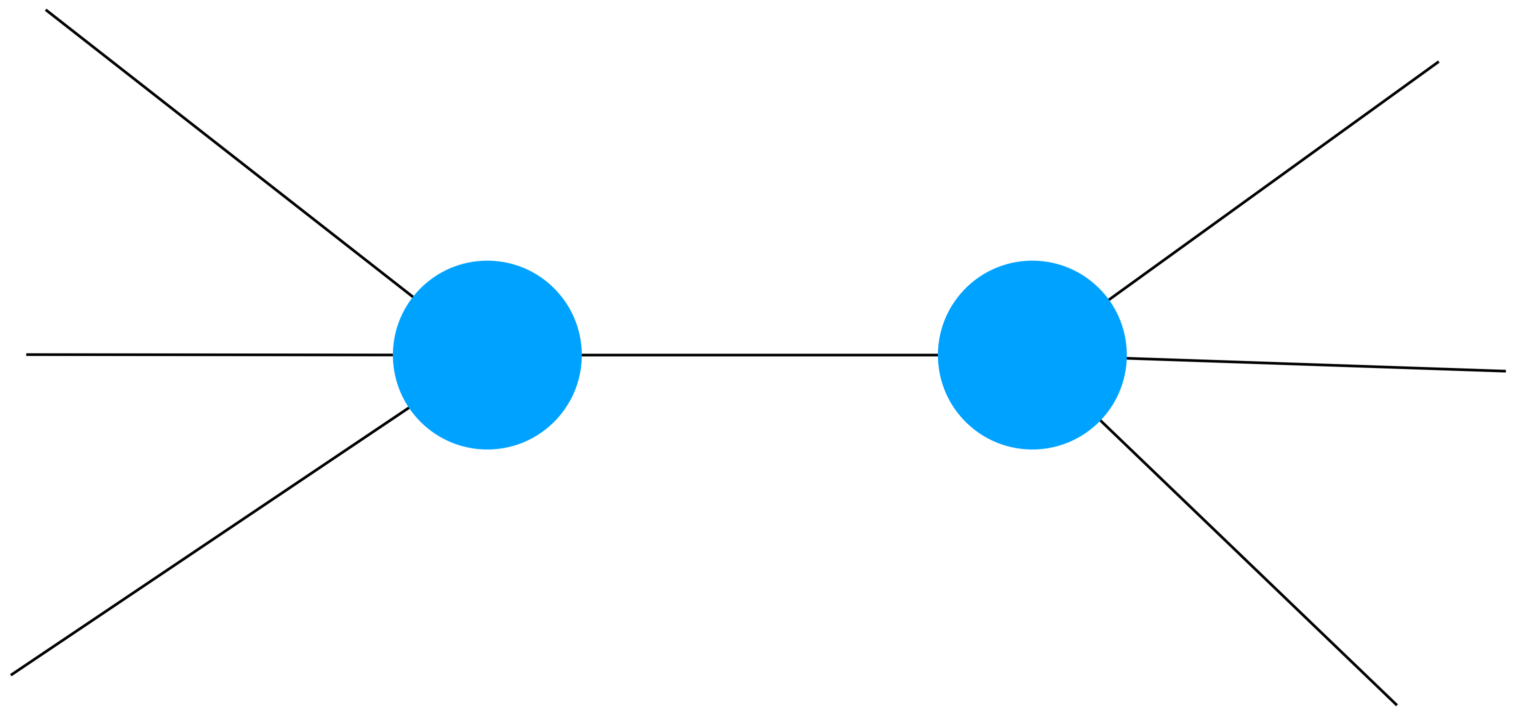
1. Deleting an edge.
2. Deleting an isolated vertex.
3. Contracting an edge.

Graph Minors

Definition

Let G be a graph. A graph H is called a **minor** of G if it can be obtained from G by successive applications of the following operations:

1. Deleting an edge.
2. Deleting an isolated vertex.
3. Contracting an edge.

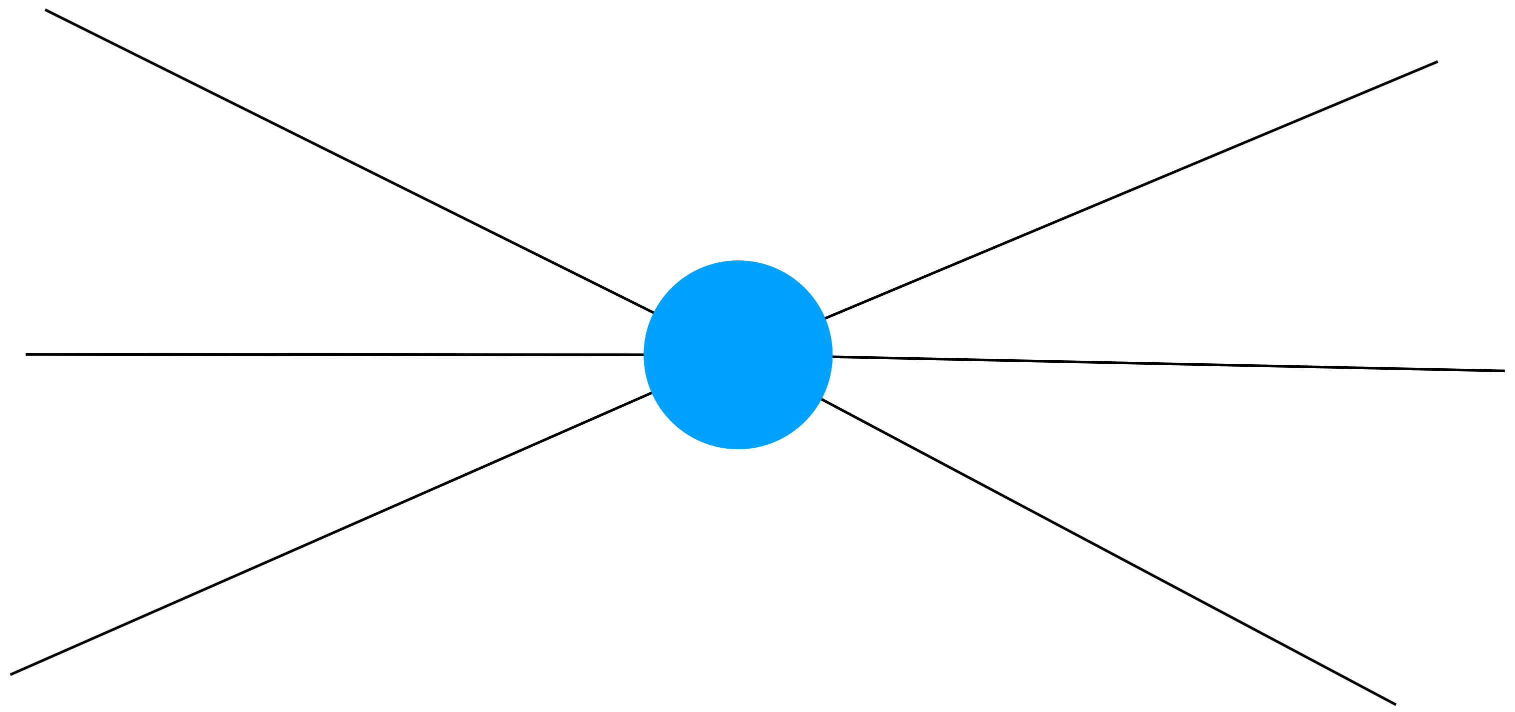


Graph Minors

Definition

Let G be a graph. A graph H is called a **minor** of G if it can be obtained from G by successive applications of the following operations:

1. Deleting an edge.
2. Deleting an isolated vertex.
3. Contracting an edge.



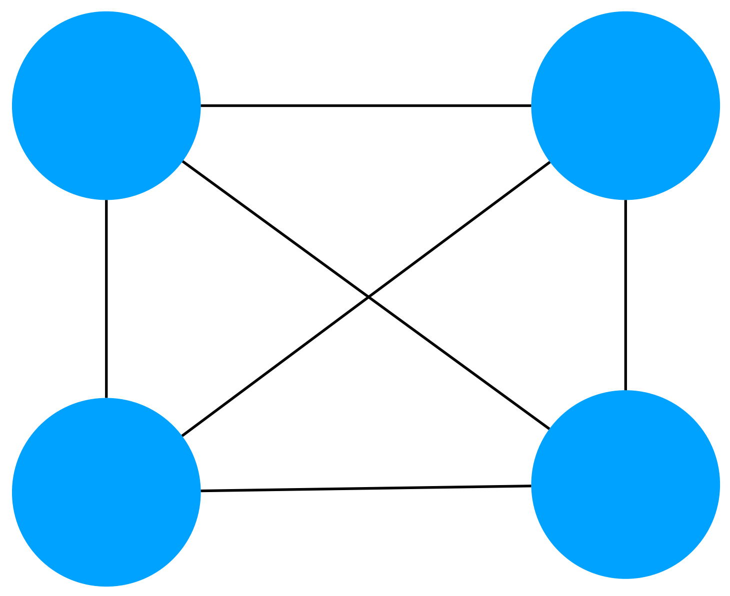
Forbidden Minors of Planar Graphs

Forbidden Minors of Planar Graphs

Definition
A graph is **planar** if it can be embedded in the plane without edge crossings.

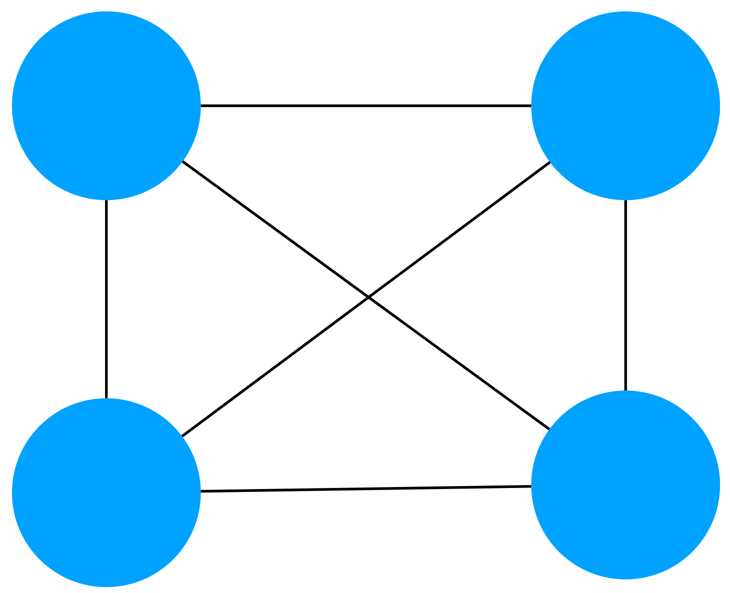
Forbidden Minors of Planar Graphs

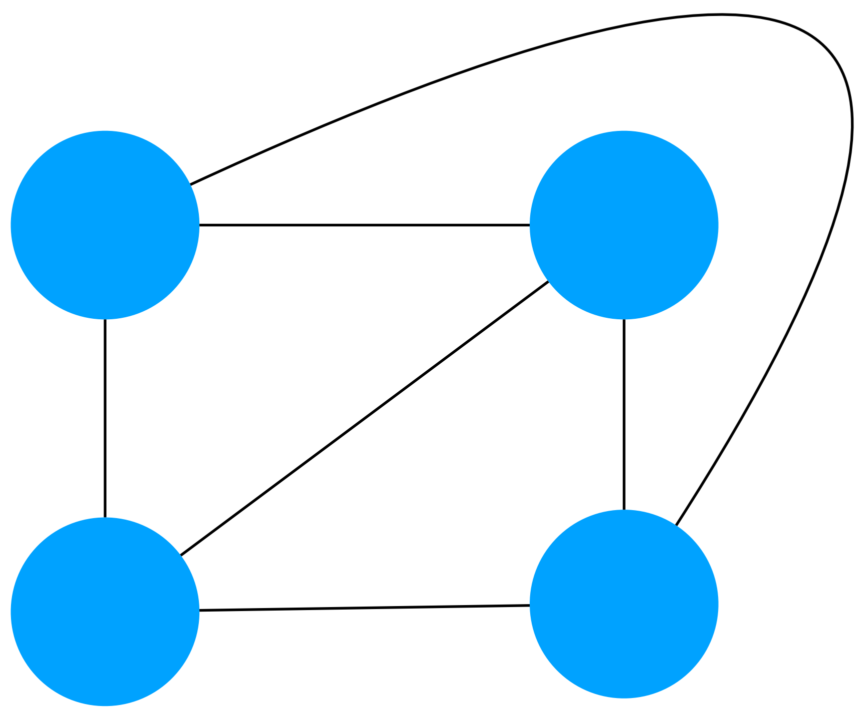
Definition
A graph is **planar** if it can be embedded in the plane without edge crossings.



Forbidden Minors of Planar Graphs

Definition
A graph is **planar** if it can be embedded in the plane without edge crossings.



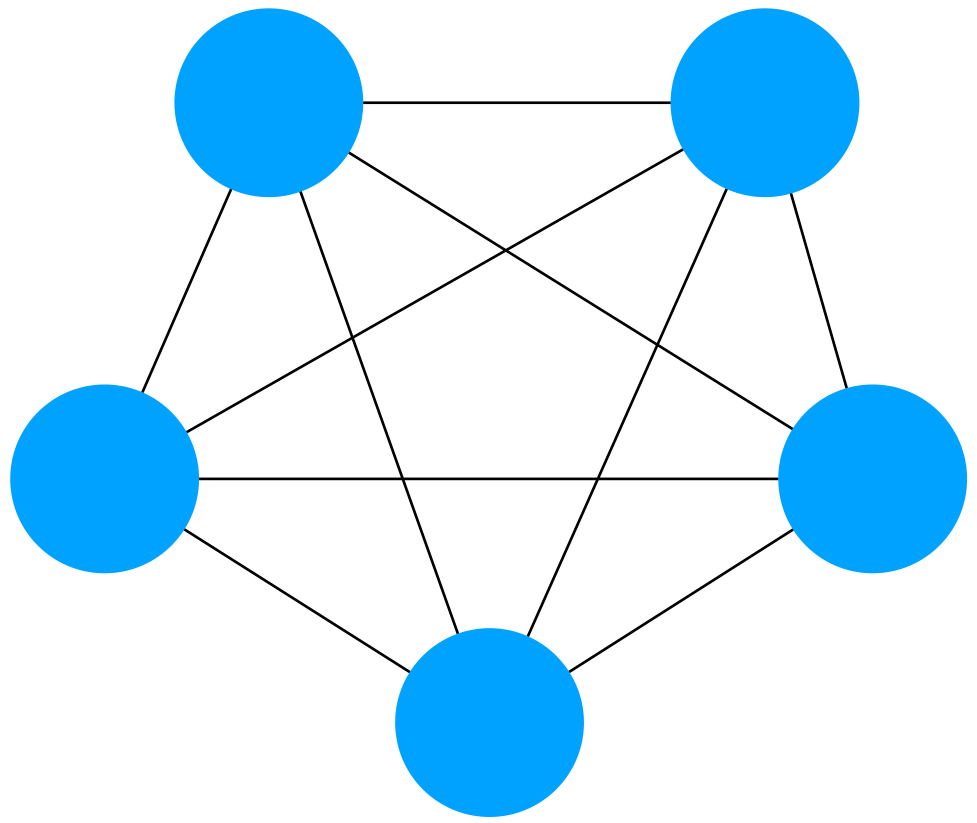


Forbidden Minors of Planar Graphs

Definition
A graph is **planar** if it can be embedded in the plane without edge crossings.

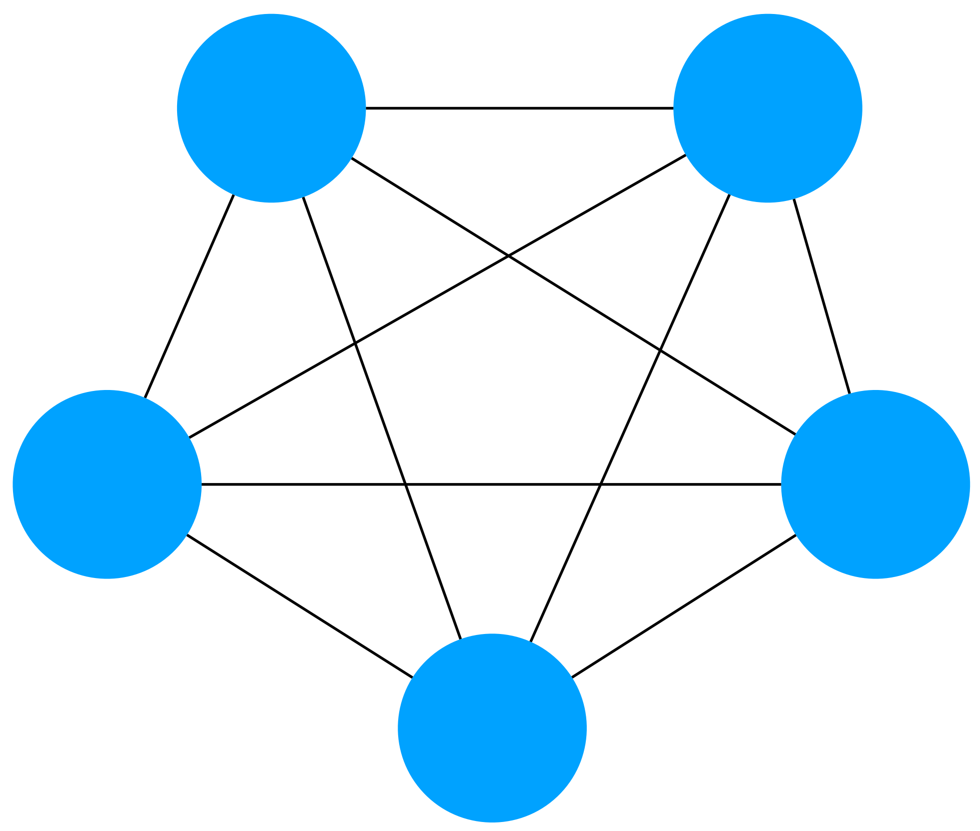
Forbidden Minors of Planar Graphs

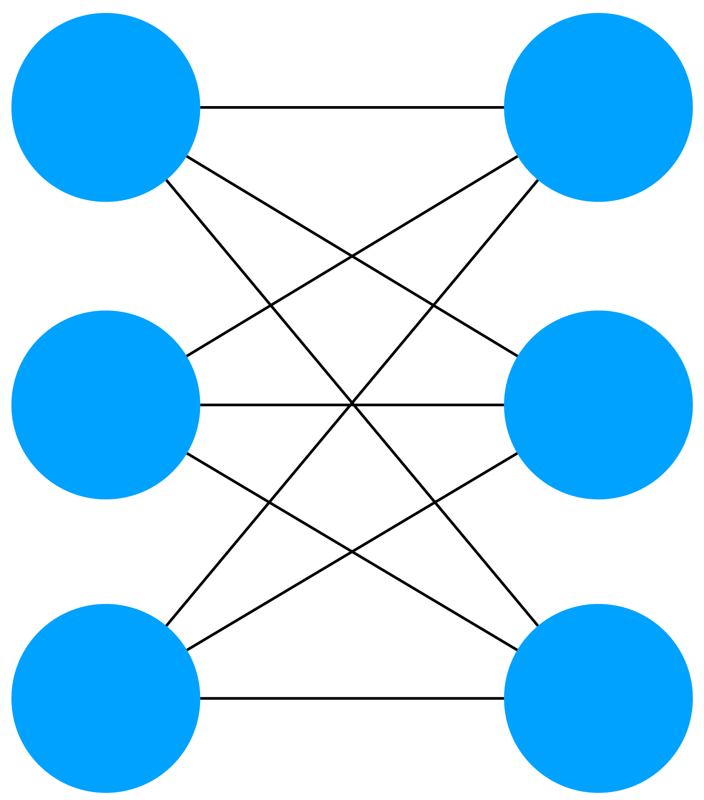
Definition
A graph is **planar** if it can be embedded in the plane without edge crossings.



Forbidden Minors of Planar Graphs

Definition
A graph is **planar** if it can be embedded in the plane without edge crossings.

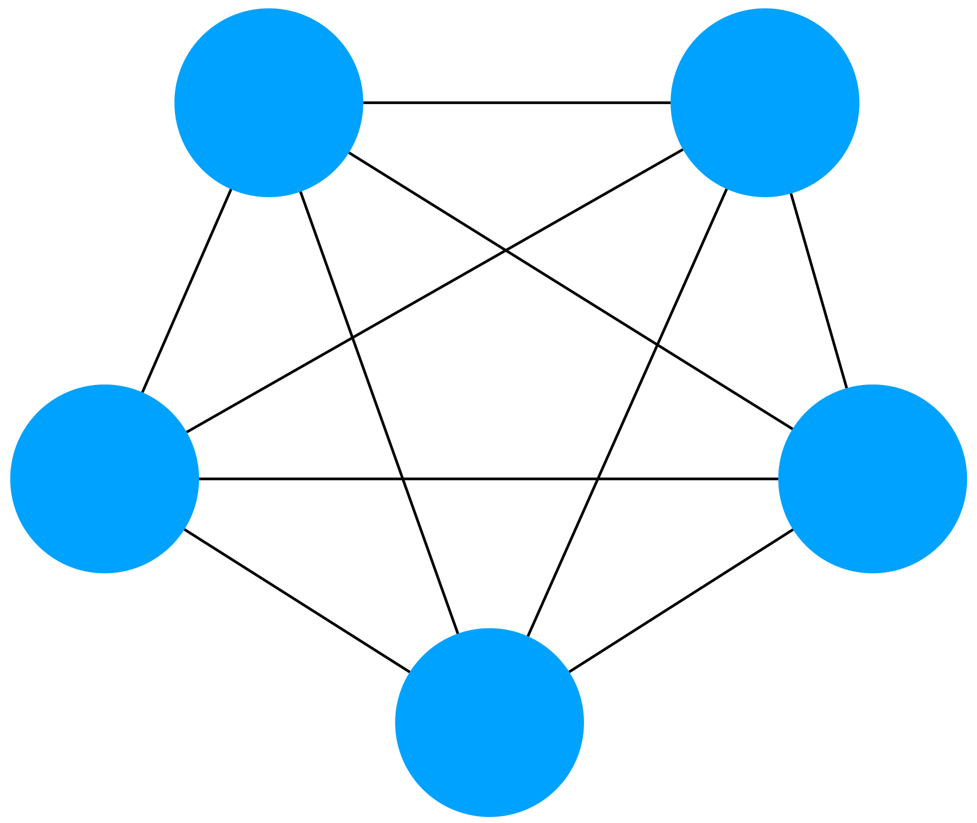


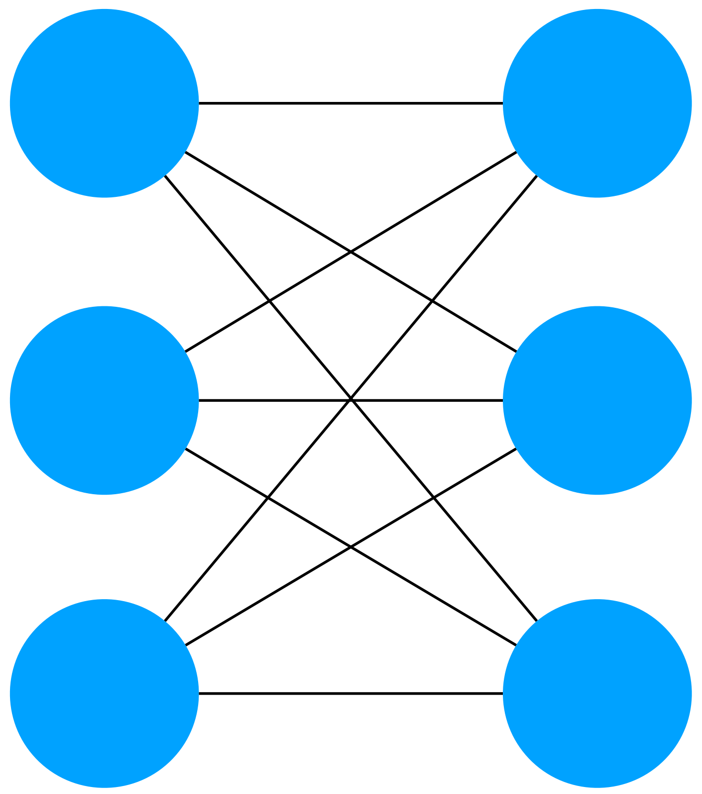


Forbidden Minors of Planar Graphs

Definition

A graph is **planar** if it can be embedded in the plane without edge crossings.





Theorem (Kuratowski's Theorem)

A graph is planar if, and only if, it does not contain K_5 or $K_{3,3}$ as a minor.

Treewidth and Minors

Treewidth and Minors

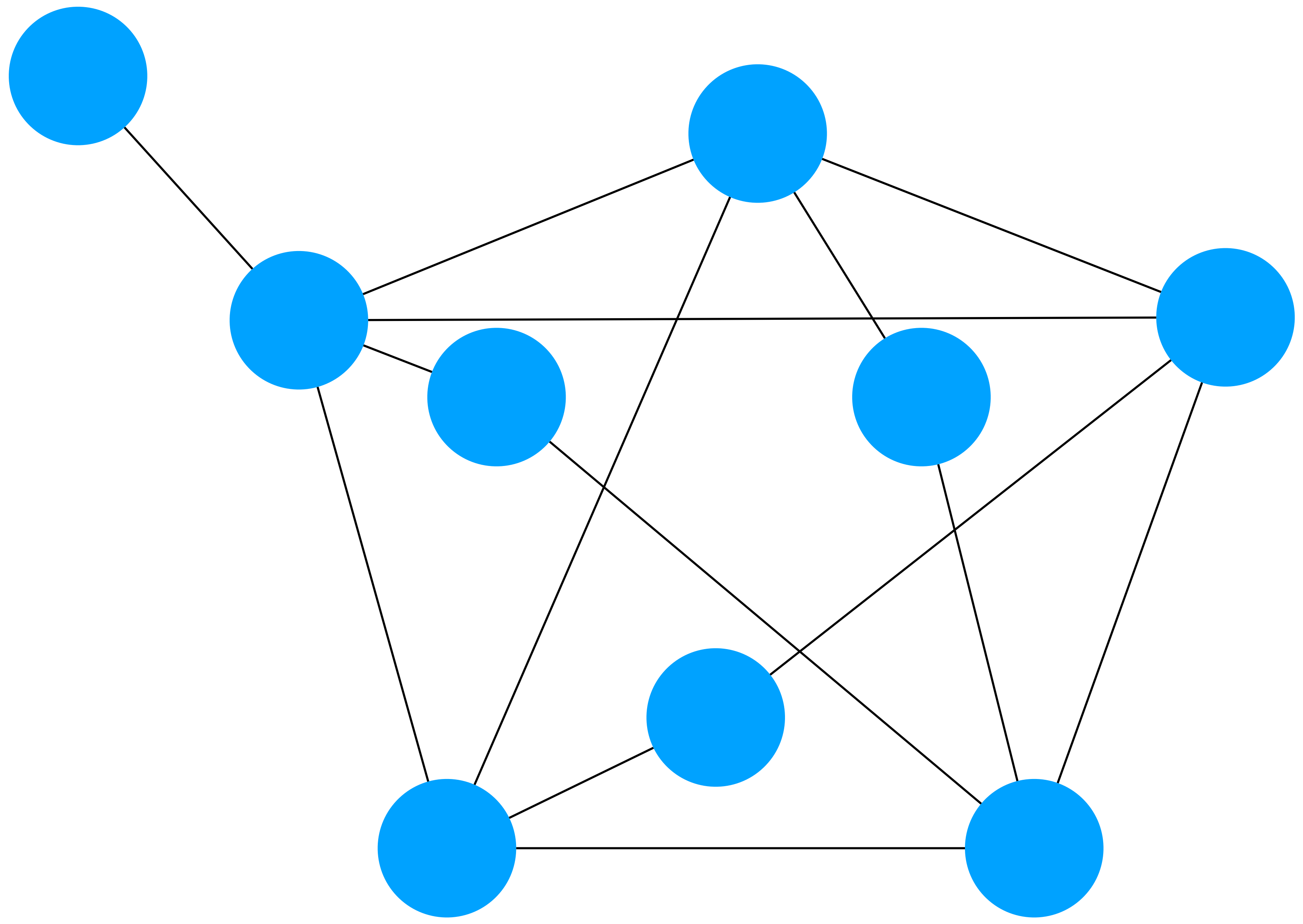
Observation

If H is a minor of G then $tw(H) \leq tw(G)$.

Treewidth and Minors

Observation

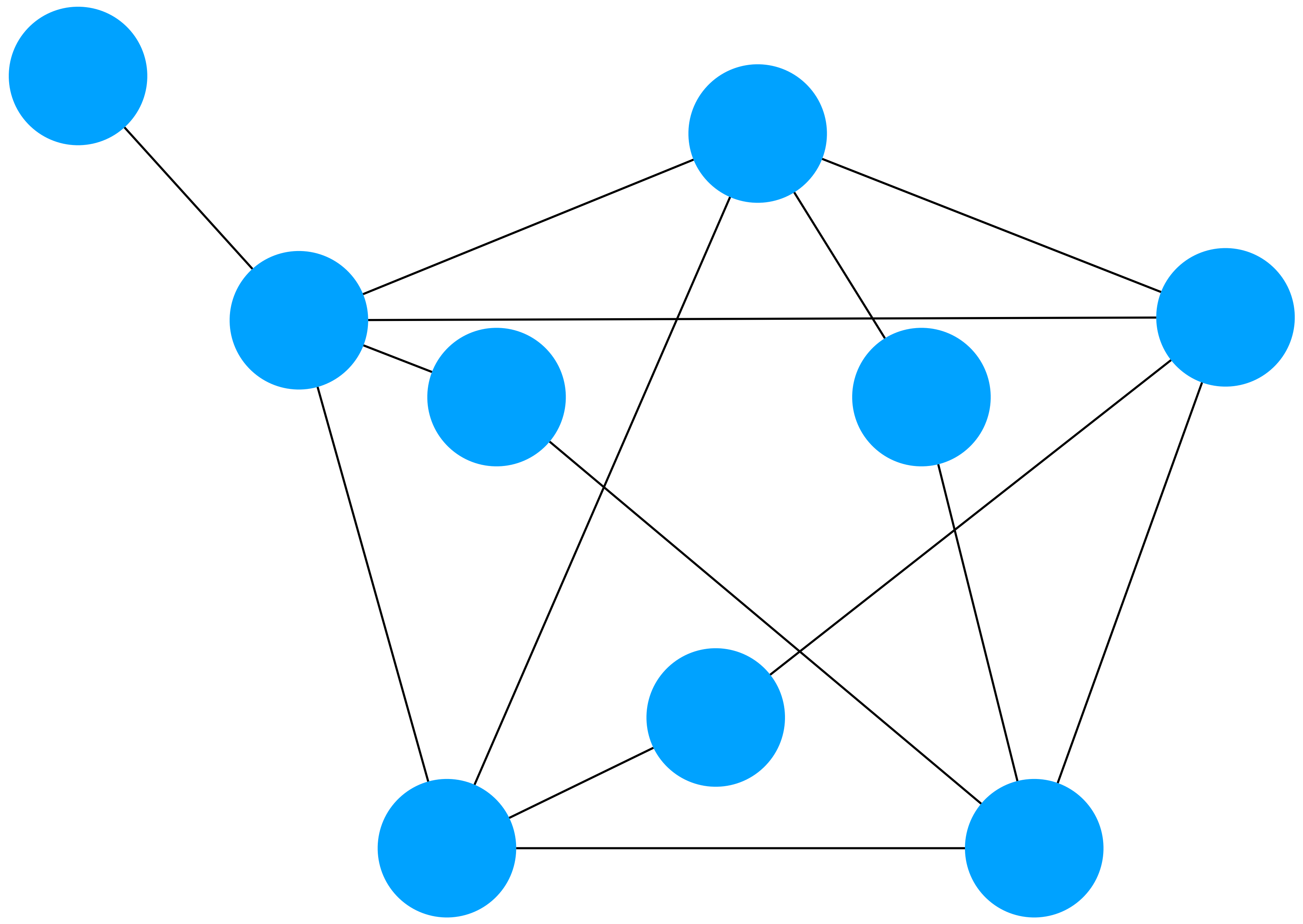
If H is a minor of G then $tw(H) \leq tw(G)$.

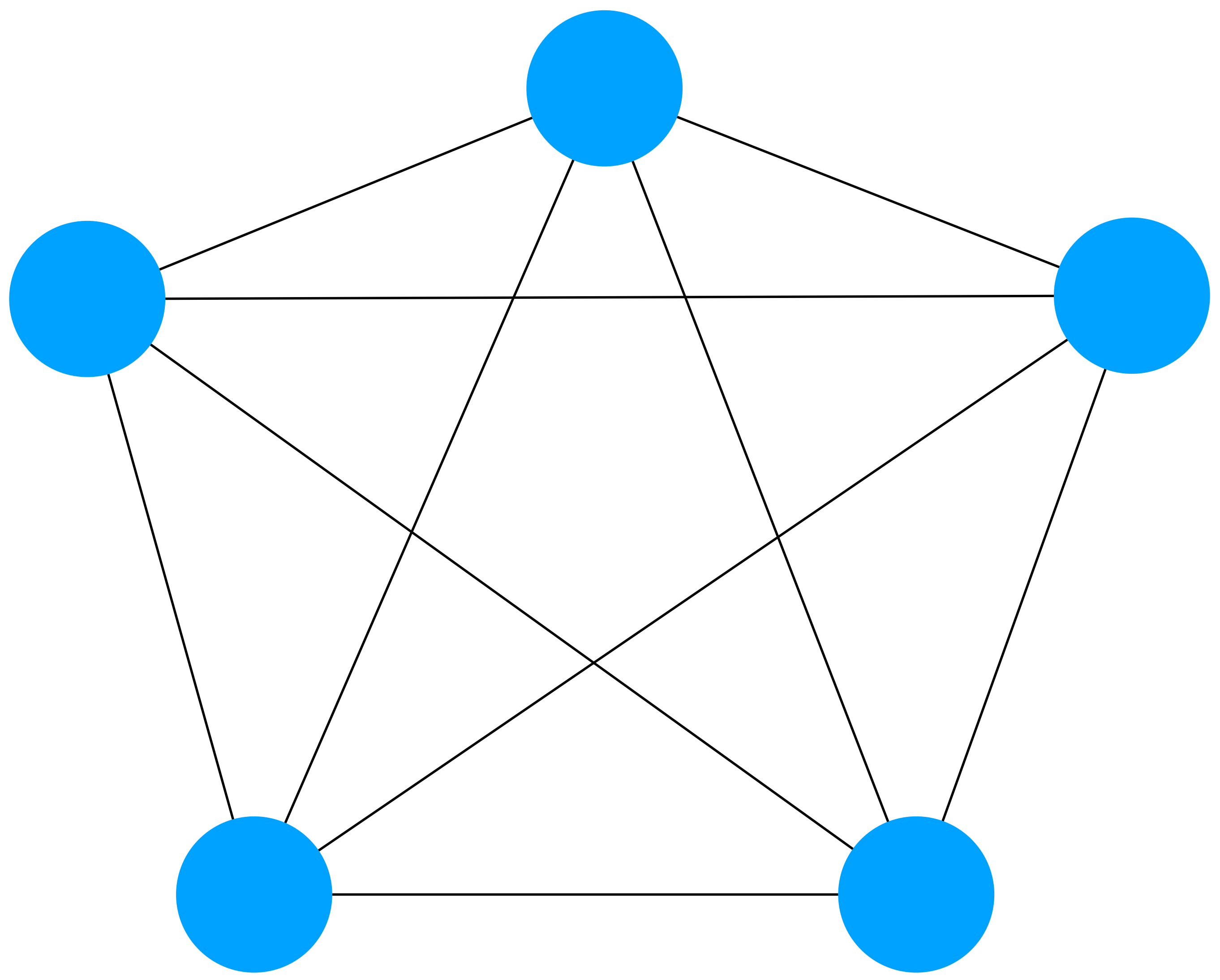


Treewidth and Minors

Observation

If H is a minor of G then $tw(H) \leq tw(G)$.

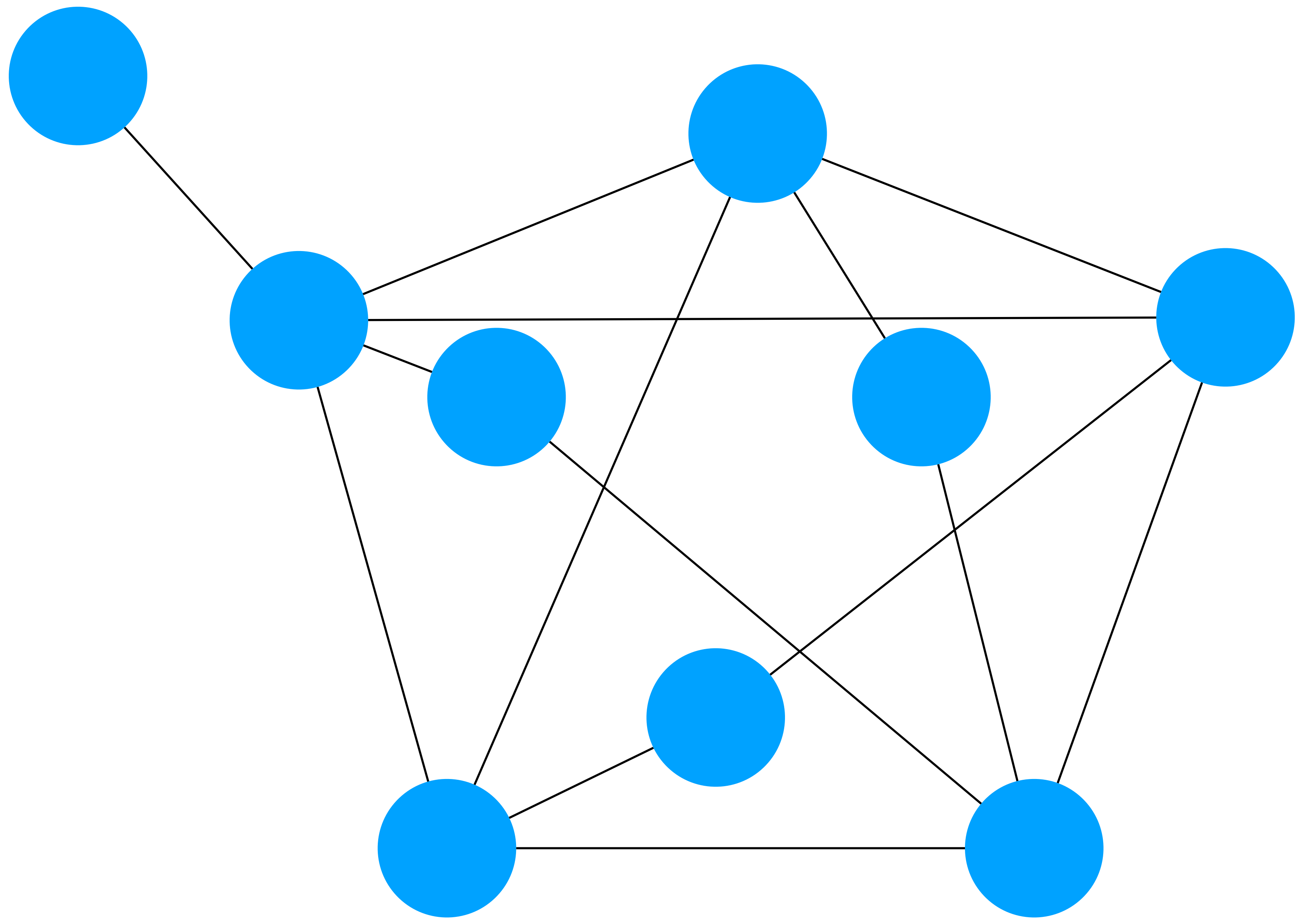


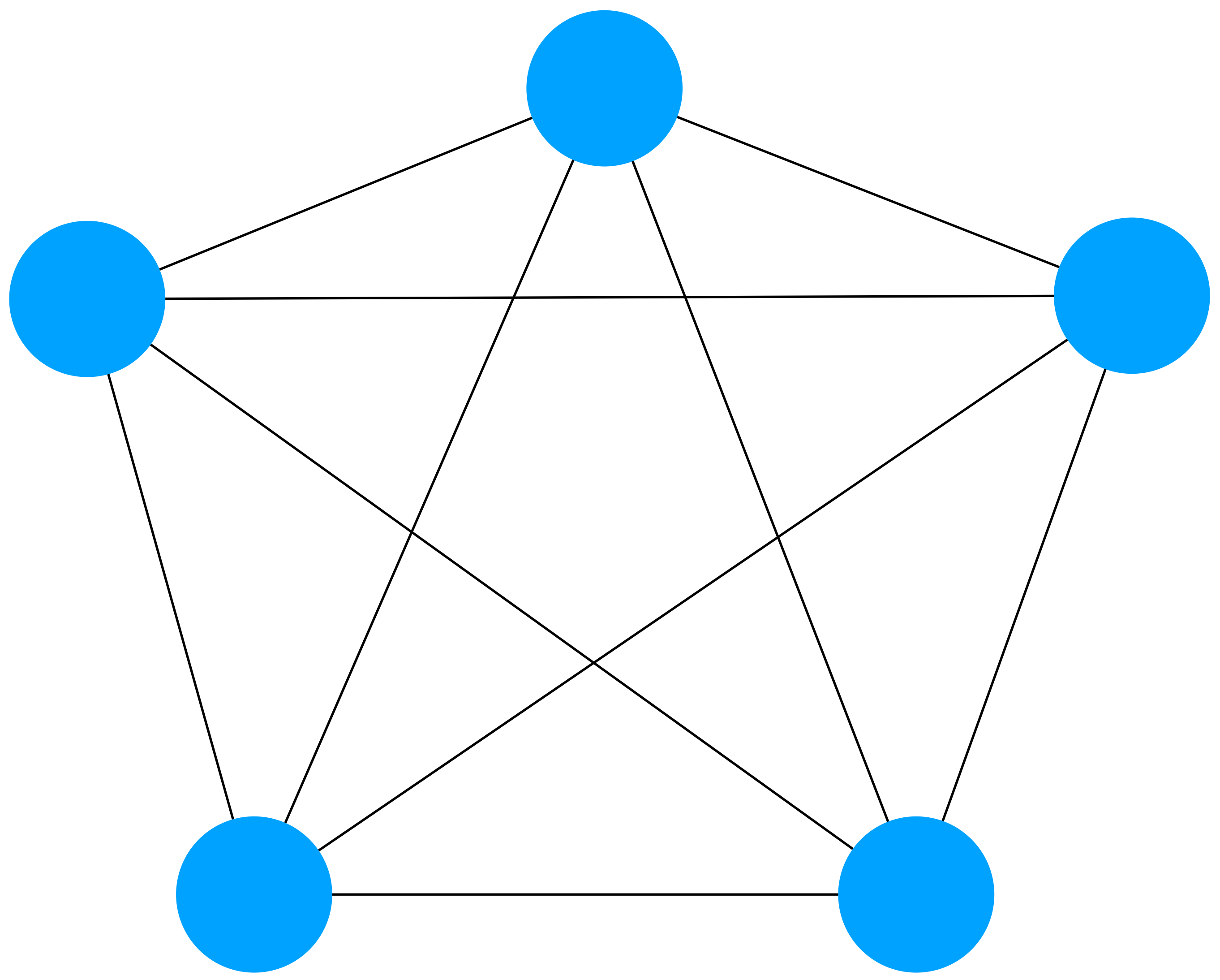


Treewidth and Minors

Observation

If H is a minor of G then $tw(H) \leq tw(G)$.



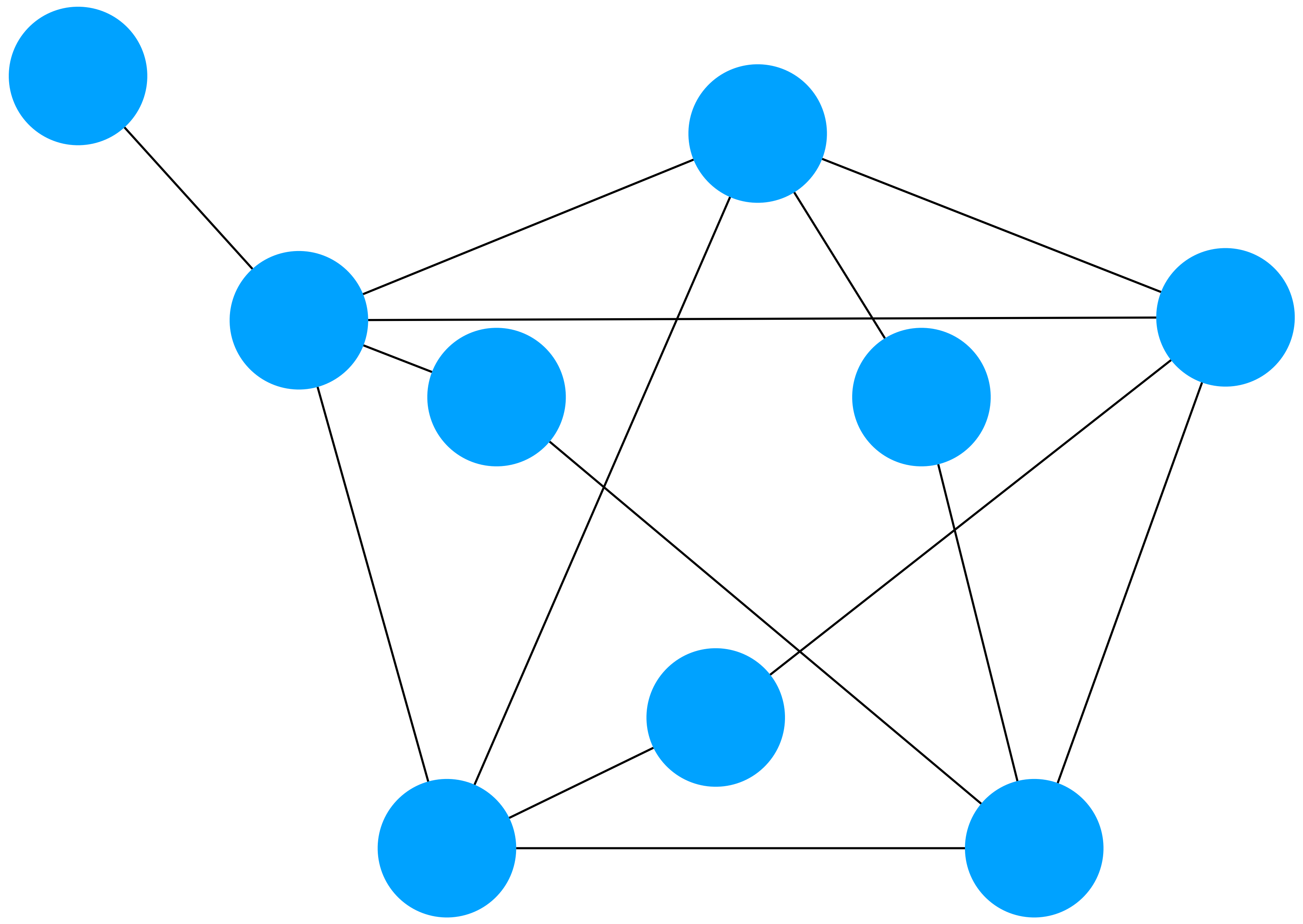


K_5 treewidth 4

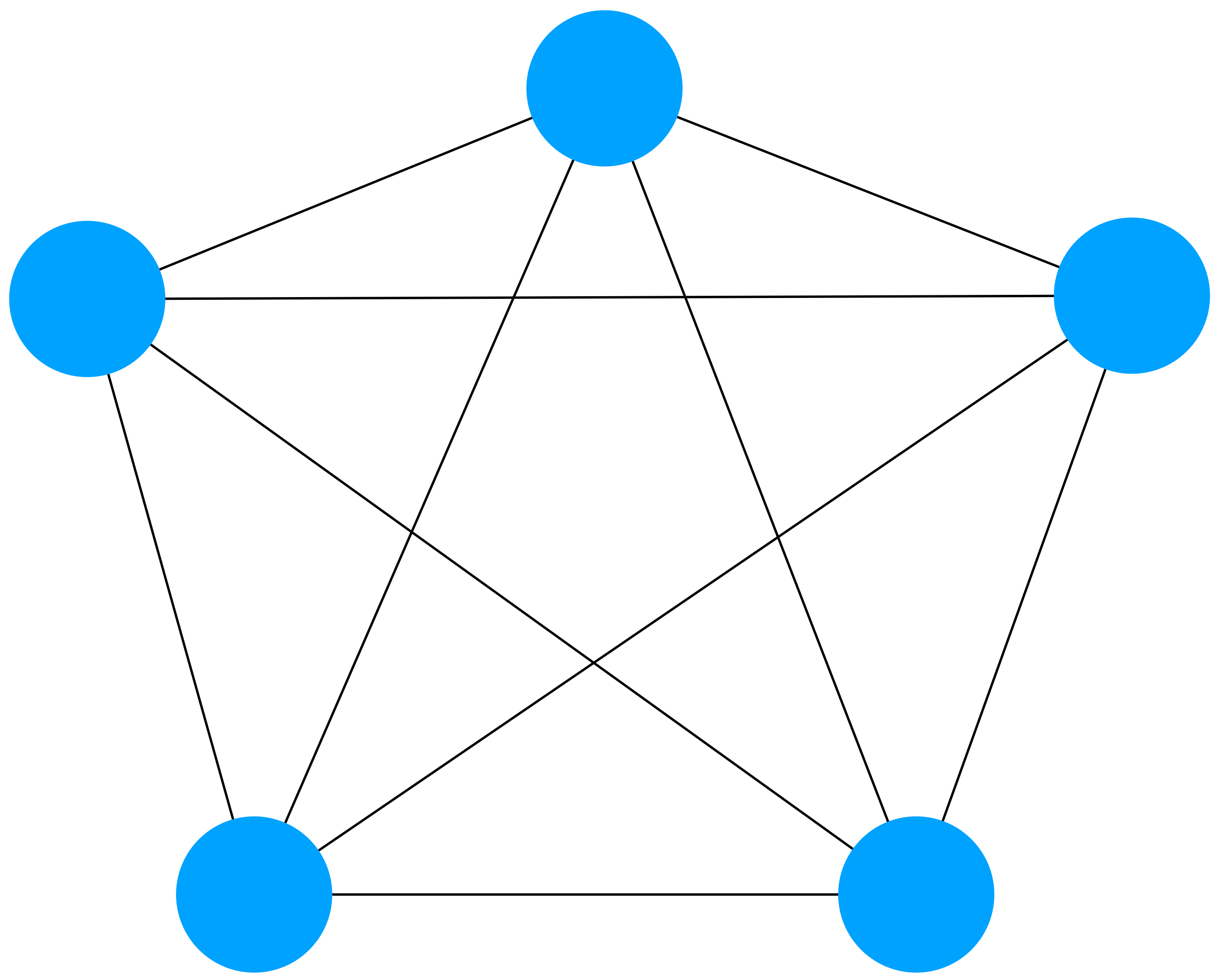
Treewidth and Minors

Observation

If H is a minor of G then $tw(H) \leq tw(G)$.

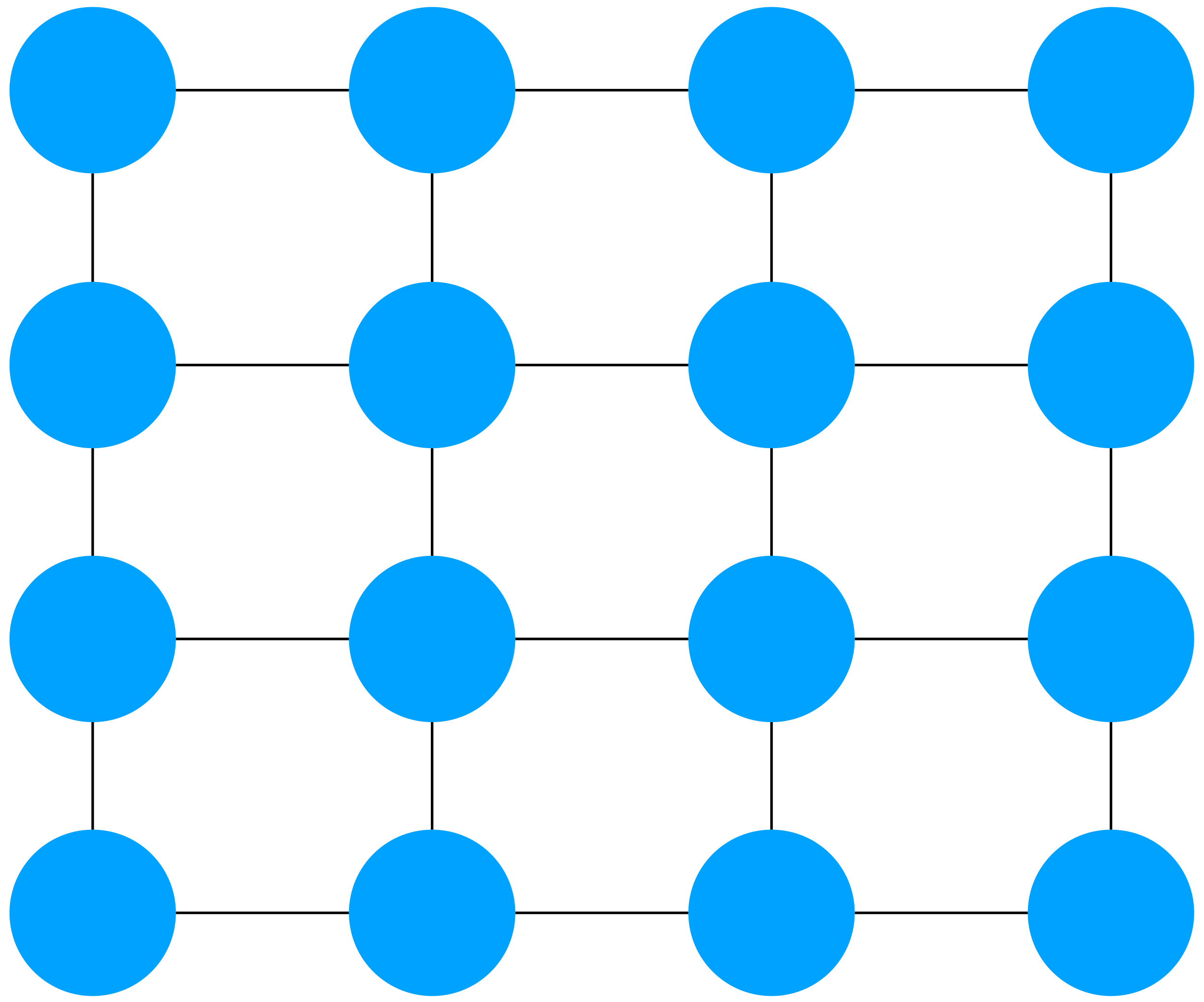


treewidth ≥ 4



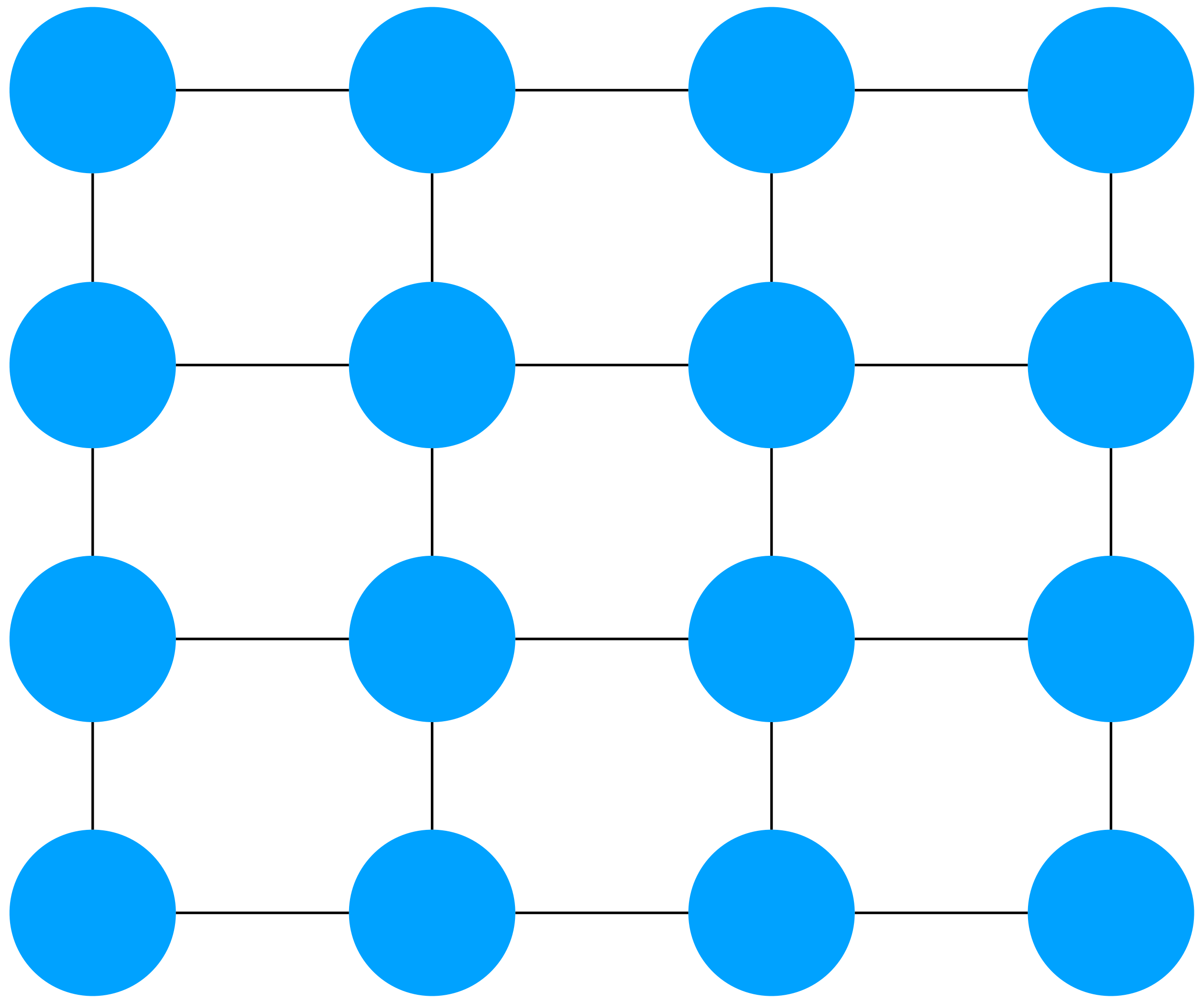
K_5 treewidth 4

Treewidth and Grid Minors



Q_4

Treewidth and Grid Minors

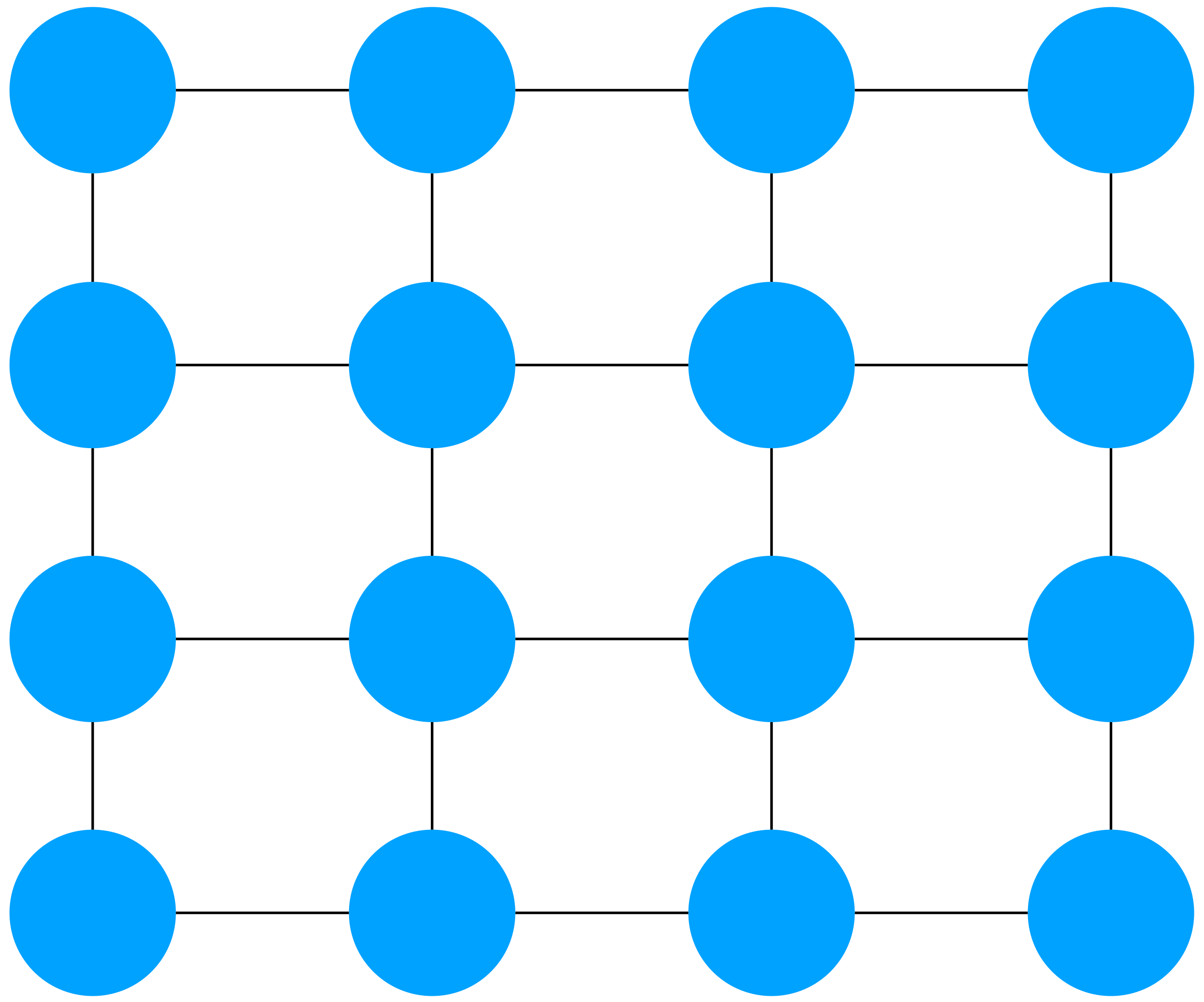


Q_4

Theorem (Robertson and Seymour)

A graph class $\mathscr{C}$ has bounded treewidth if, and only if, there is a k such that the grid Q_k is not a minor of any graph in $\mathscr{C}$.

Treewidth and Grid Minors



Q_4

Theorem (Robertson and Seymour)

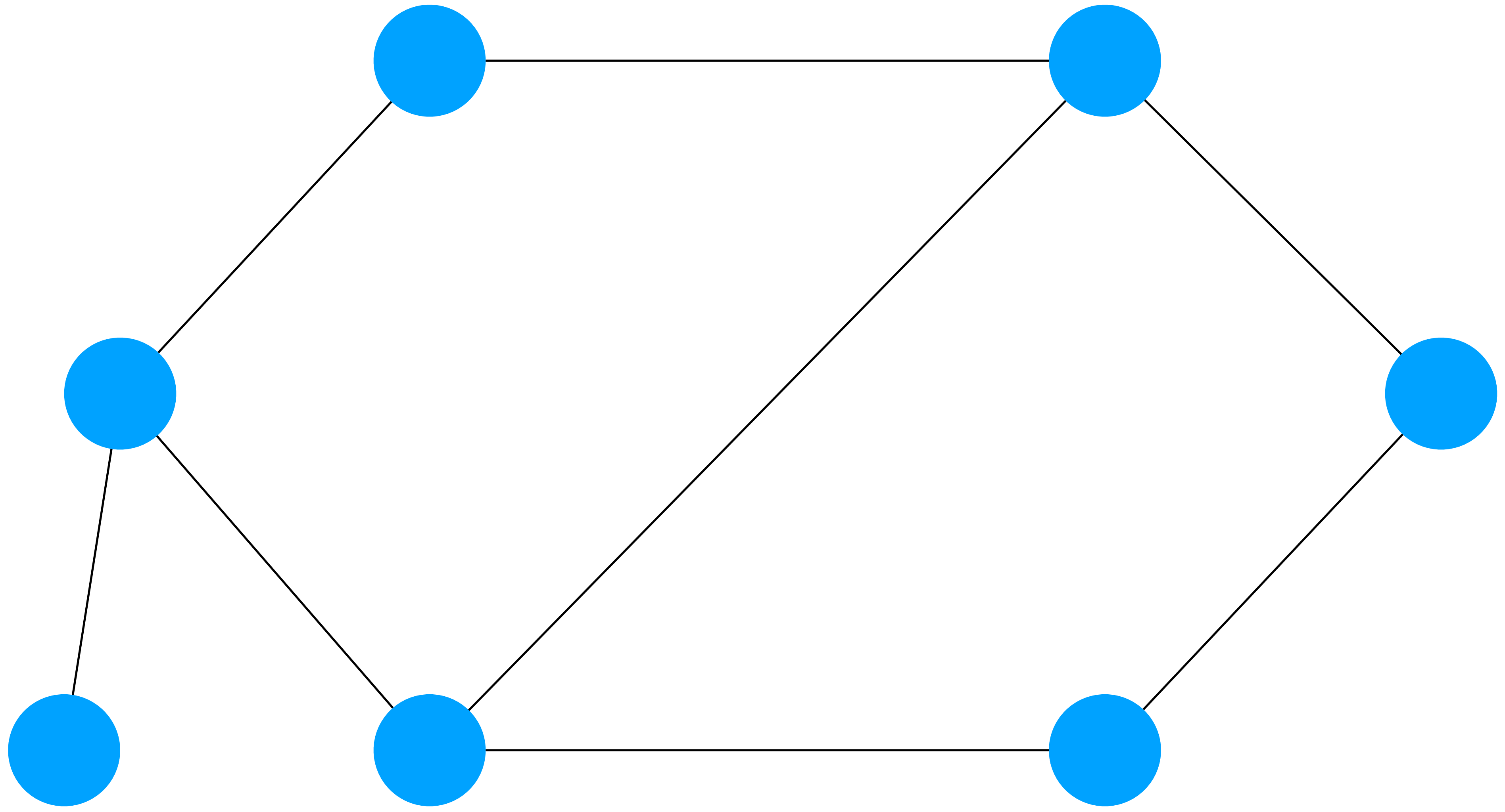
A graph class $\mathscr{C}$ has bounded treewidth if, and only if, there is a k such that the grid Q_k is not a minor of any graph in $\mathscr{C}$.

Theorem (Chekuri and Chuzhoy)

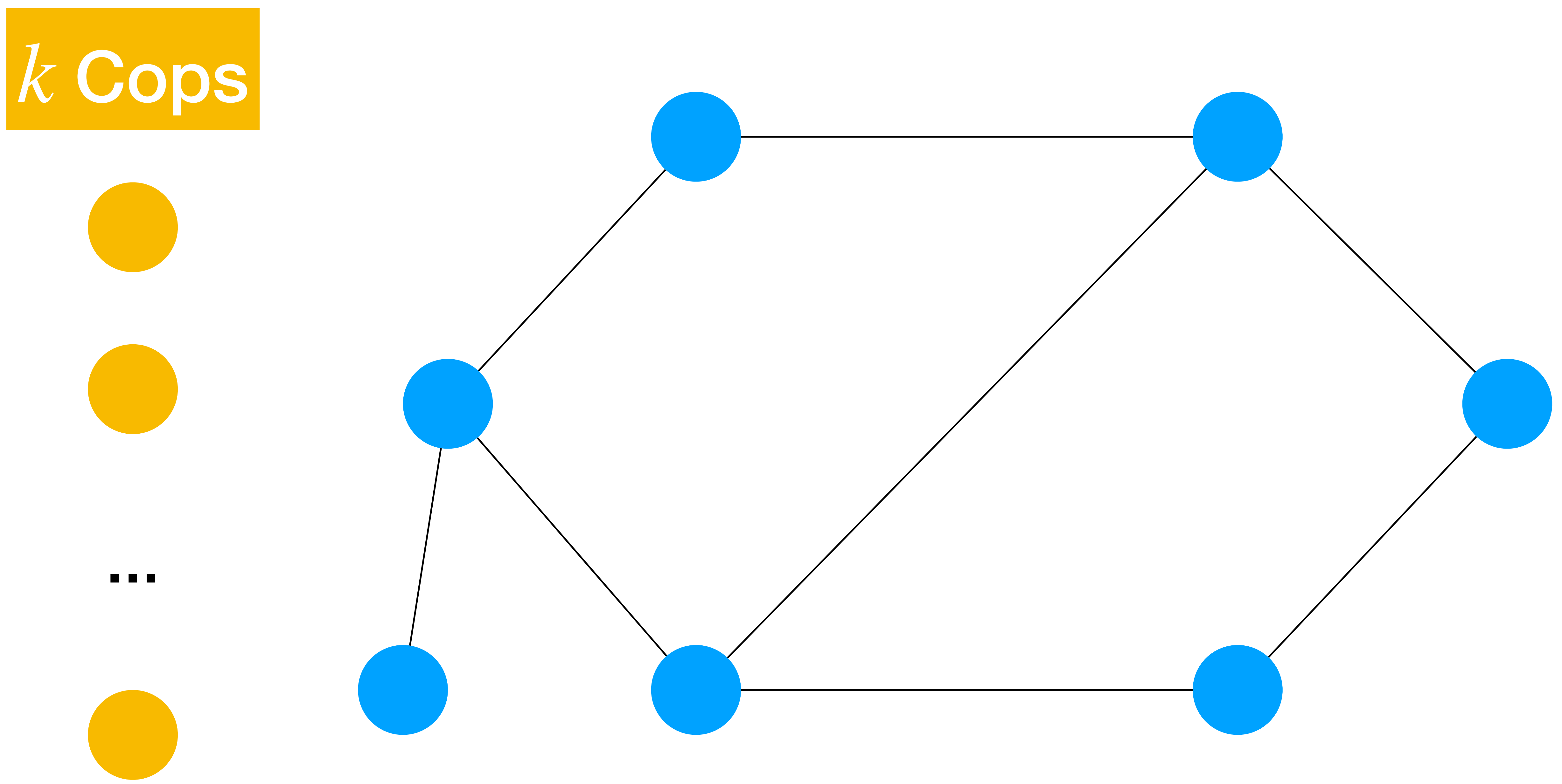
There is a polynomial p such that every graph of treewidth larger than $p(k)$ contains Q_k as a minor.

Cops and Robbers

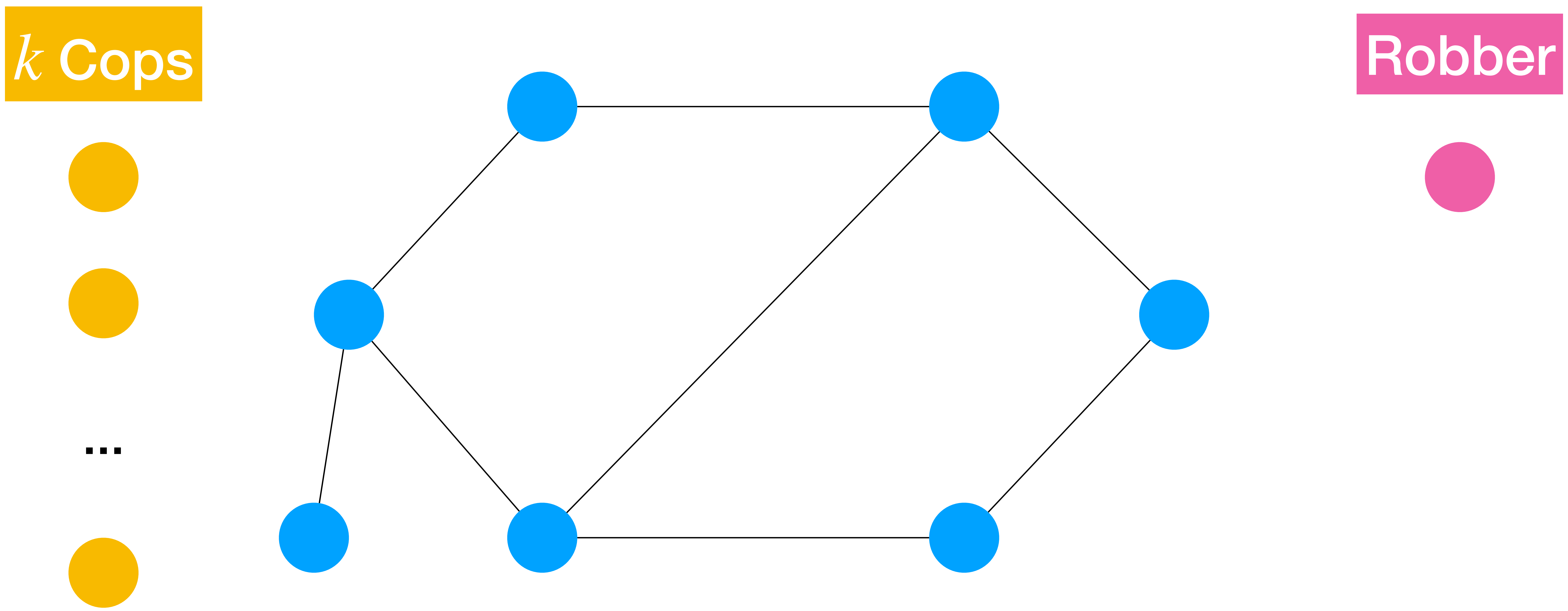
Cops and Robbers



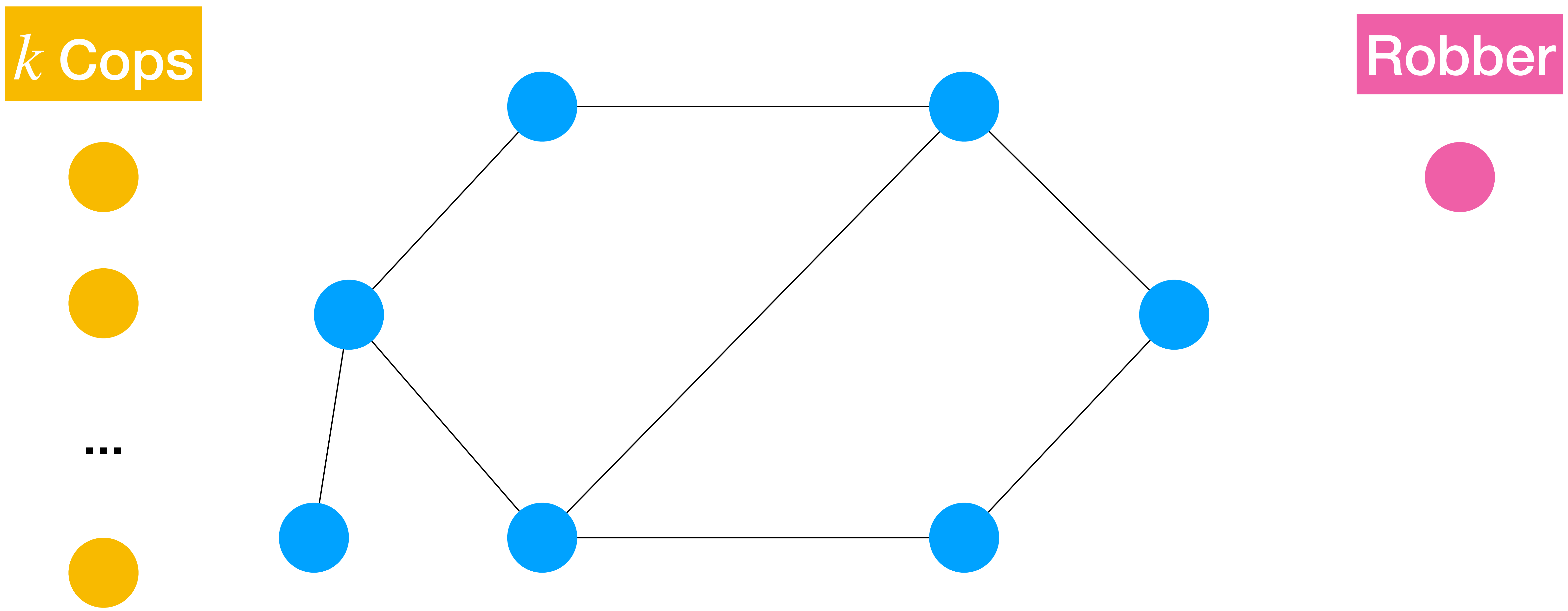
Cops and Robbers



Cops and Robbers

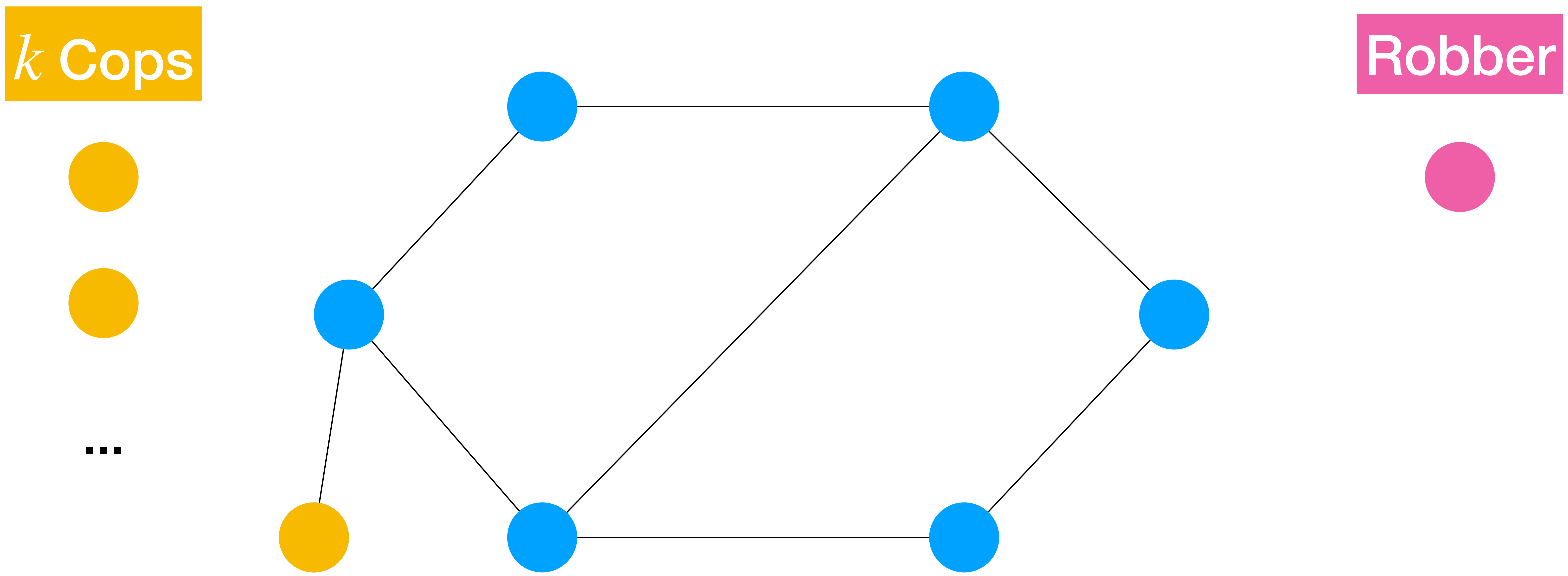


Cops and Robbers



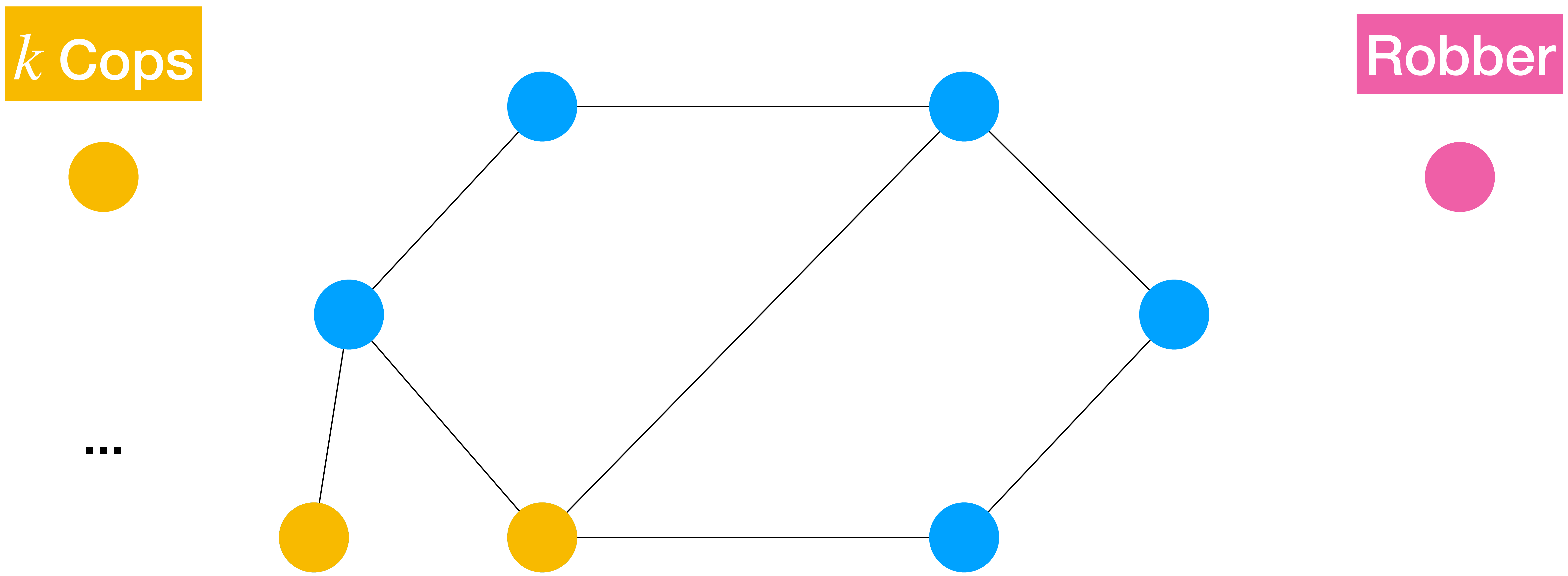
1. First, cops position themselves on the graph.

Cops and Robbers



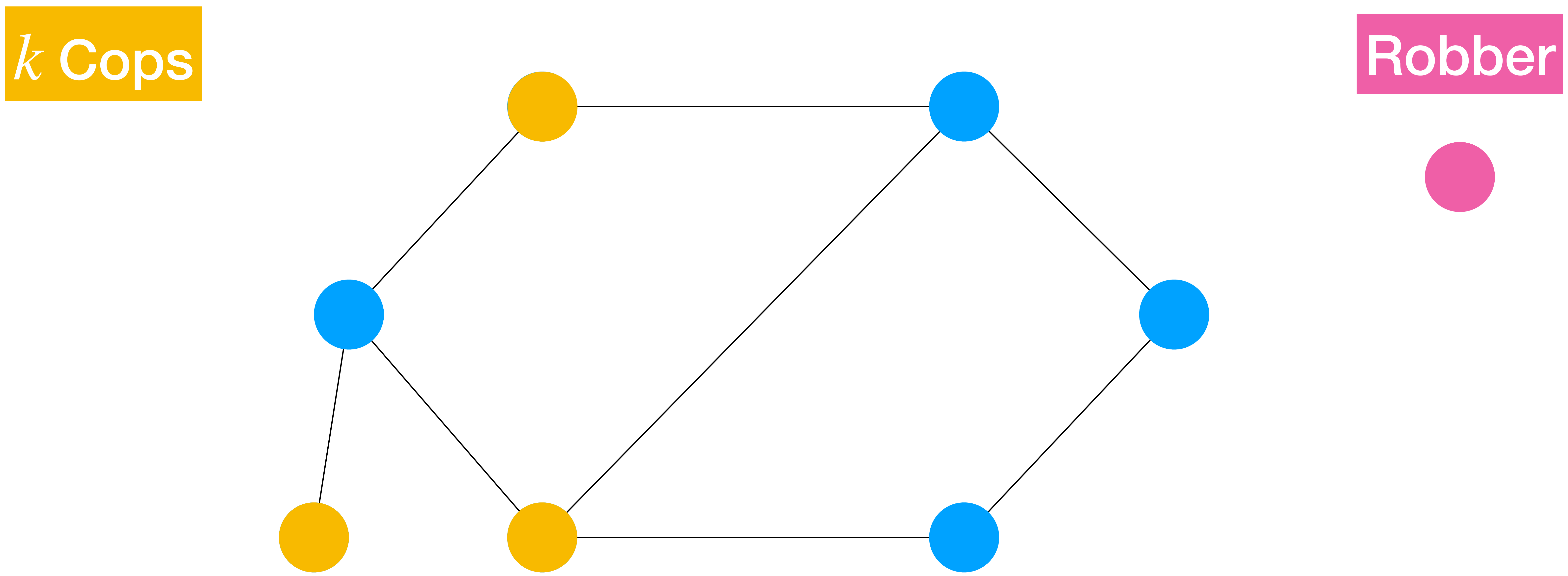
1. First, cops position themselves on the graph.

Cops and Robbers



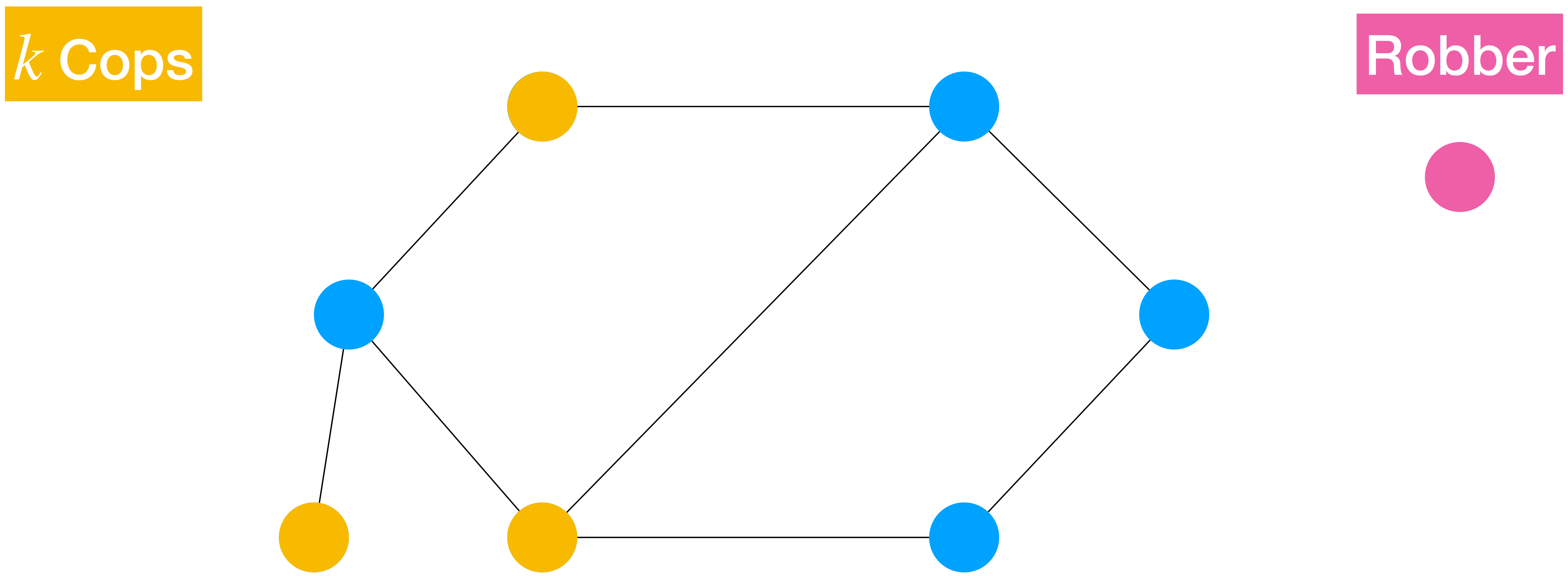
1. First, cops position themselves on the graph.

Cops and Robbers



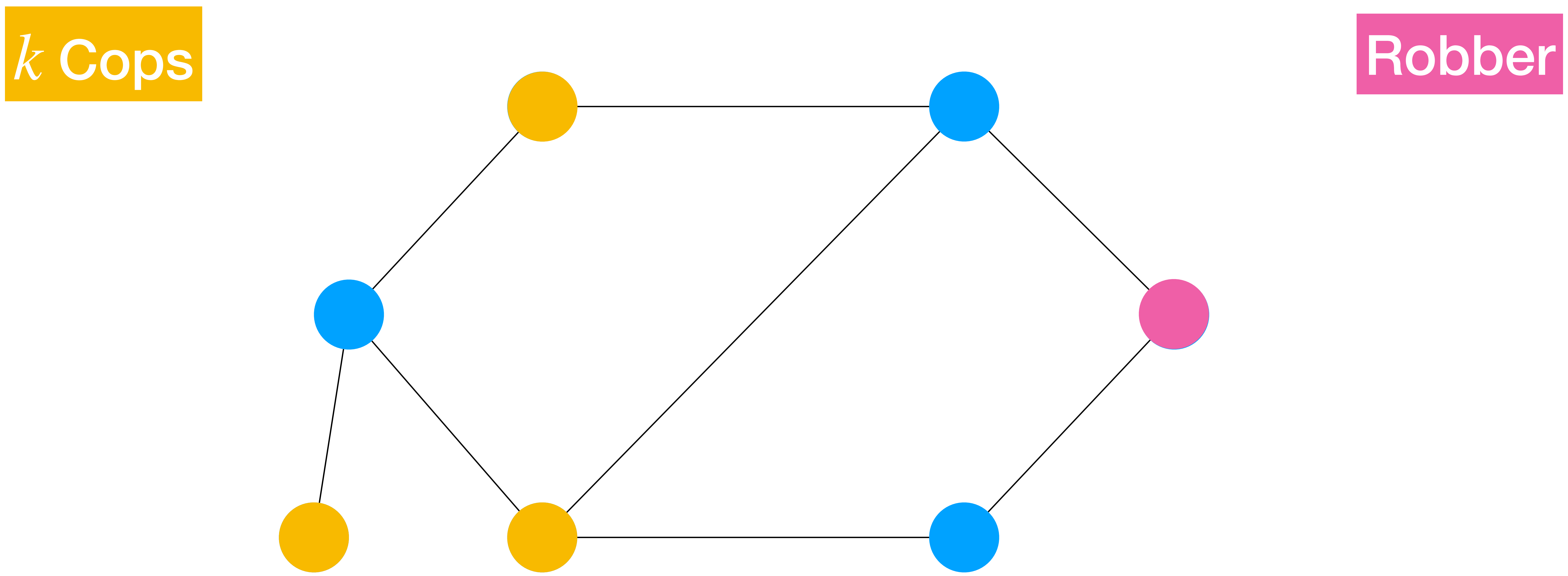
1. First, cops position themselves on the graph.

Cops and Robbers



1. First, cops position themselves on the graph.
2. Then the robber chooses a vertex.

Cops and Robbers

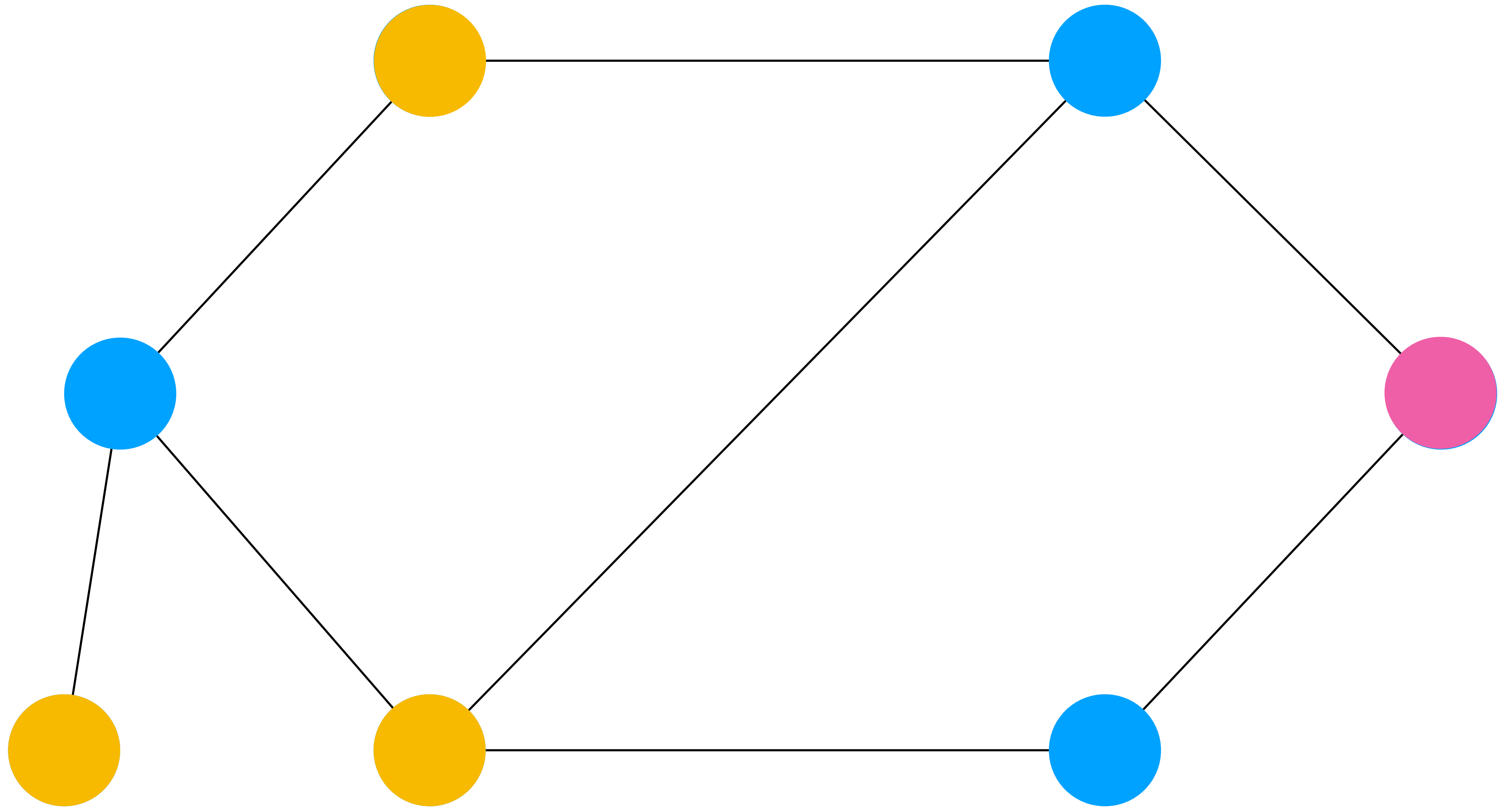


1. First, cops position themselves on the graph.
2. Then the robber chooses a vertex.

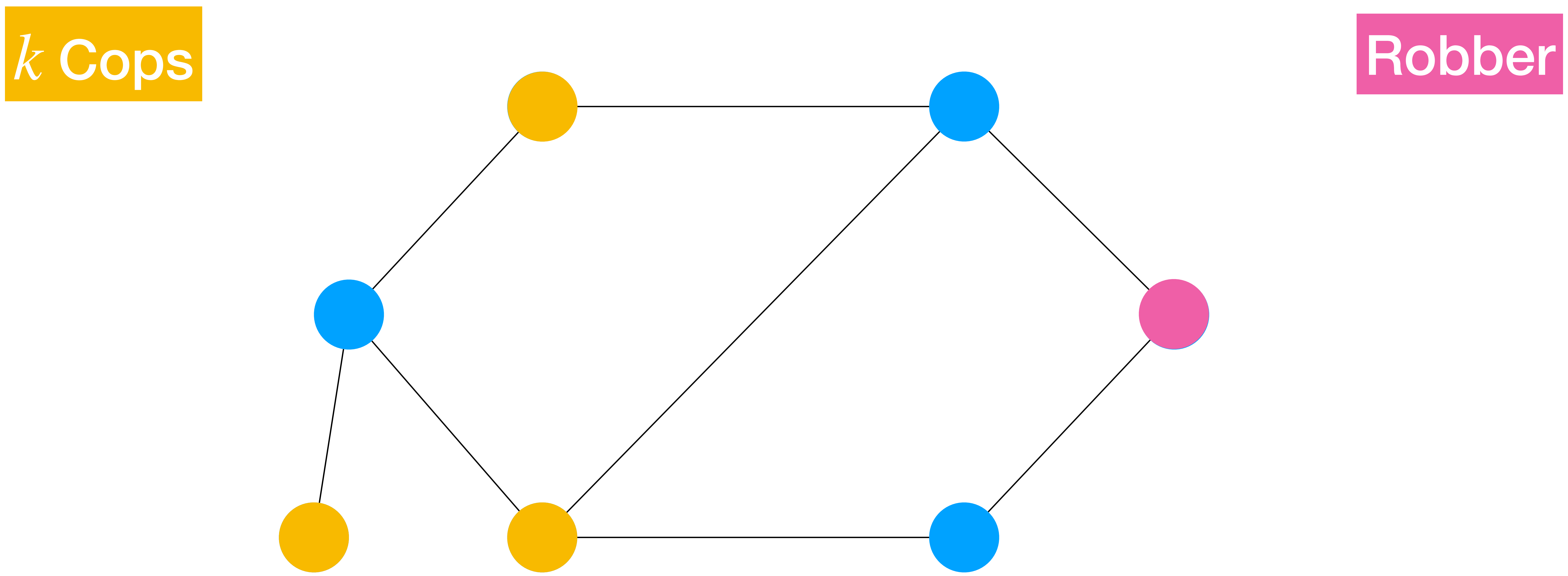
Cops and Robbers

k Cops

Robber

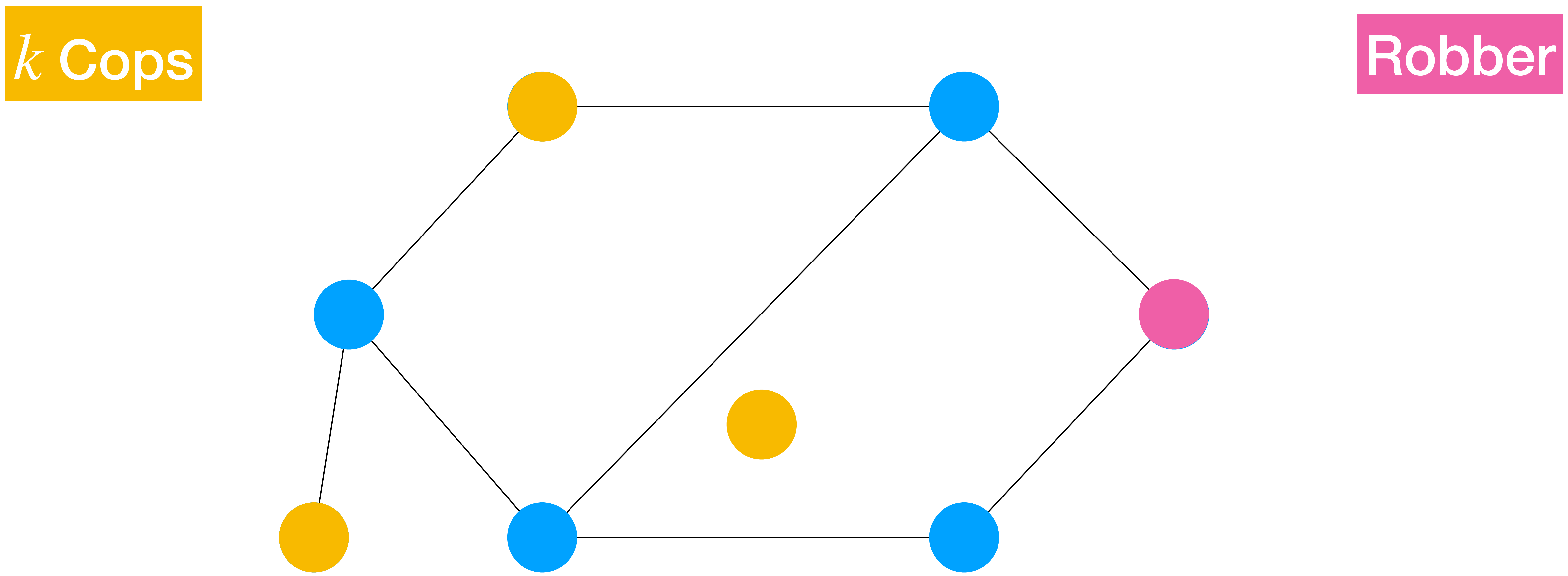


Cops and Robbers



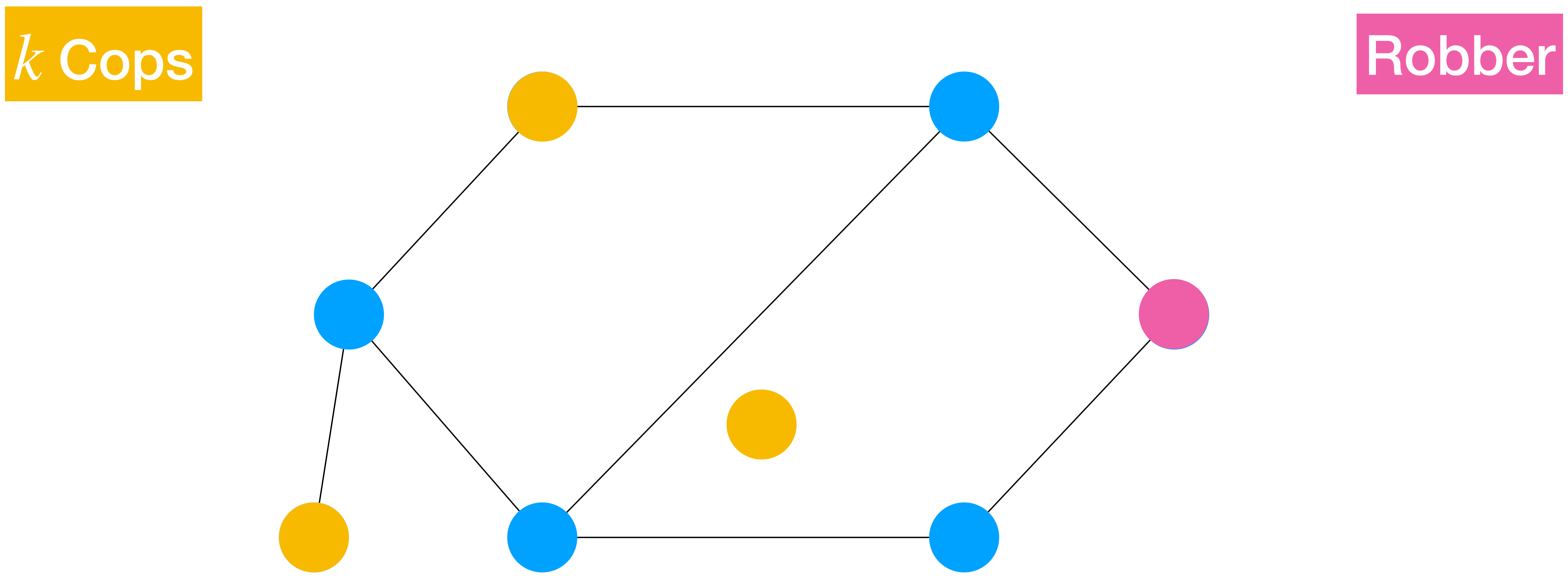
- Cops may move anywhere, but have to **"leave the graph"** to do so.

Cops and Robbers



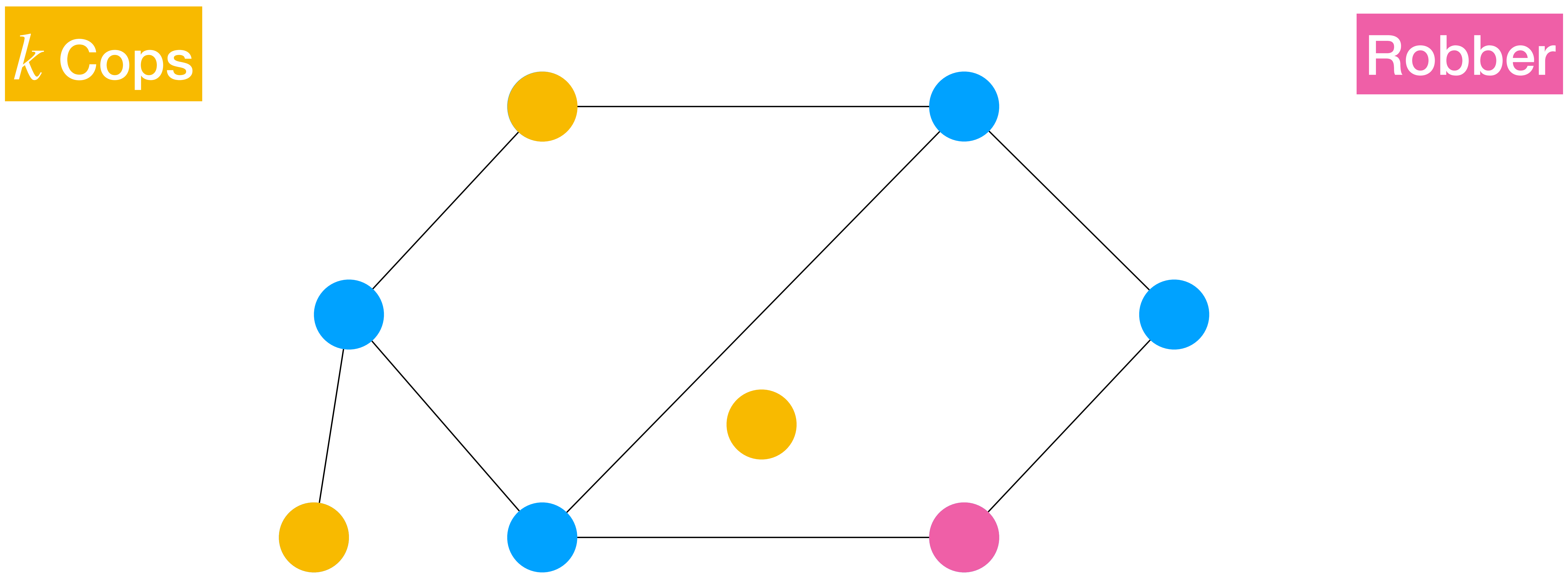
- Cops may move anywhere, but have to **“leave the graph”** to do so.

Cops and Robbers



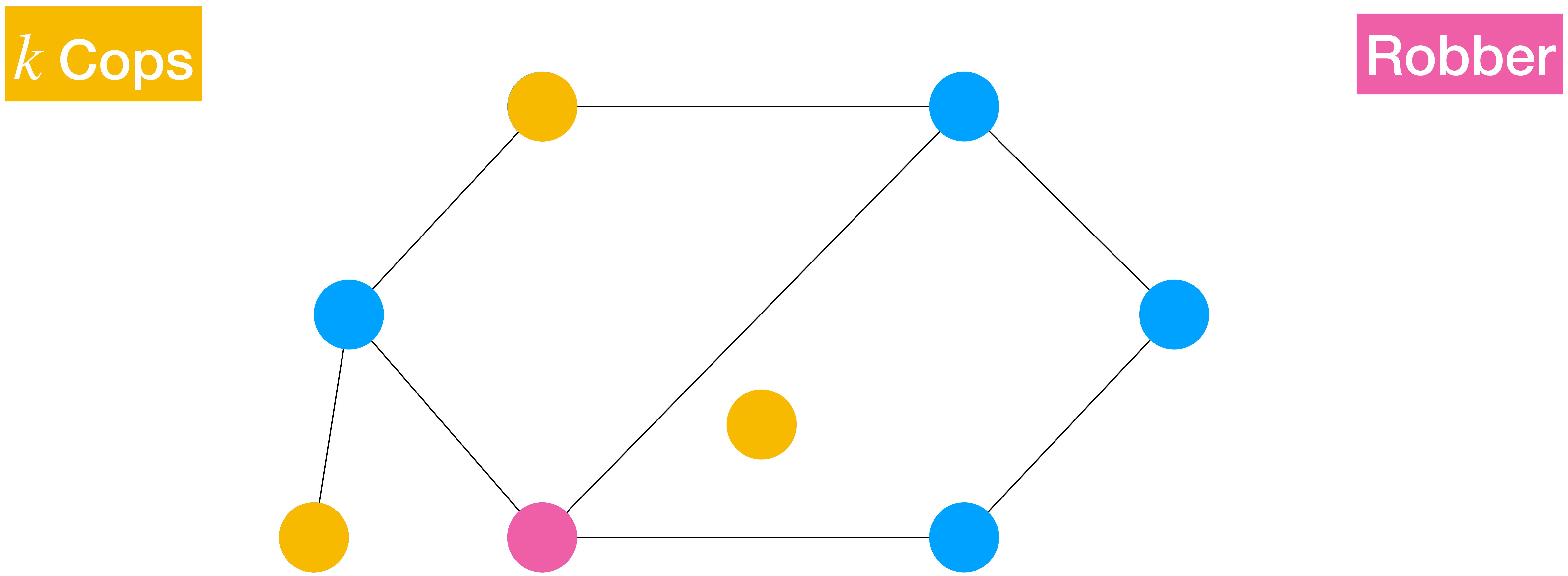
- Cops may move anywhere, but have to **"leave the graph"** to do so.
- Robber moves along the edges at **"infinite speed"**, must avoid cops.

Cops and Robbers



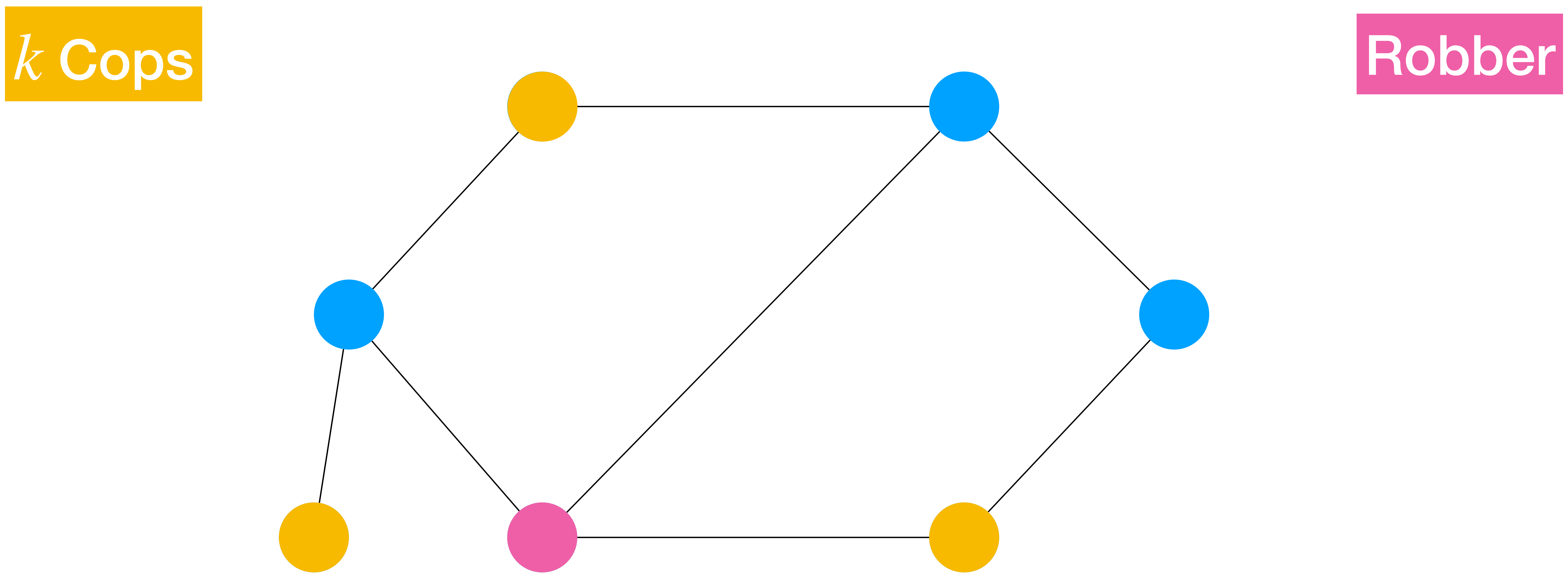
- Cops may move anywhere, but have to **“leave the graph”** to do so.
- Robber moves along the edges at **“infinite speed”**, must avoid cops.

Cops and Robbers



- Cops may move anywhere, but have to **"leave the graph"** to do so.
- Robber moves along the edges at **"infinite speed"**, must avoid cops.

Cops and Robbers

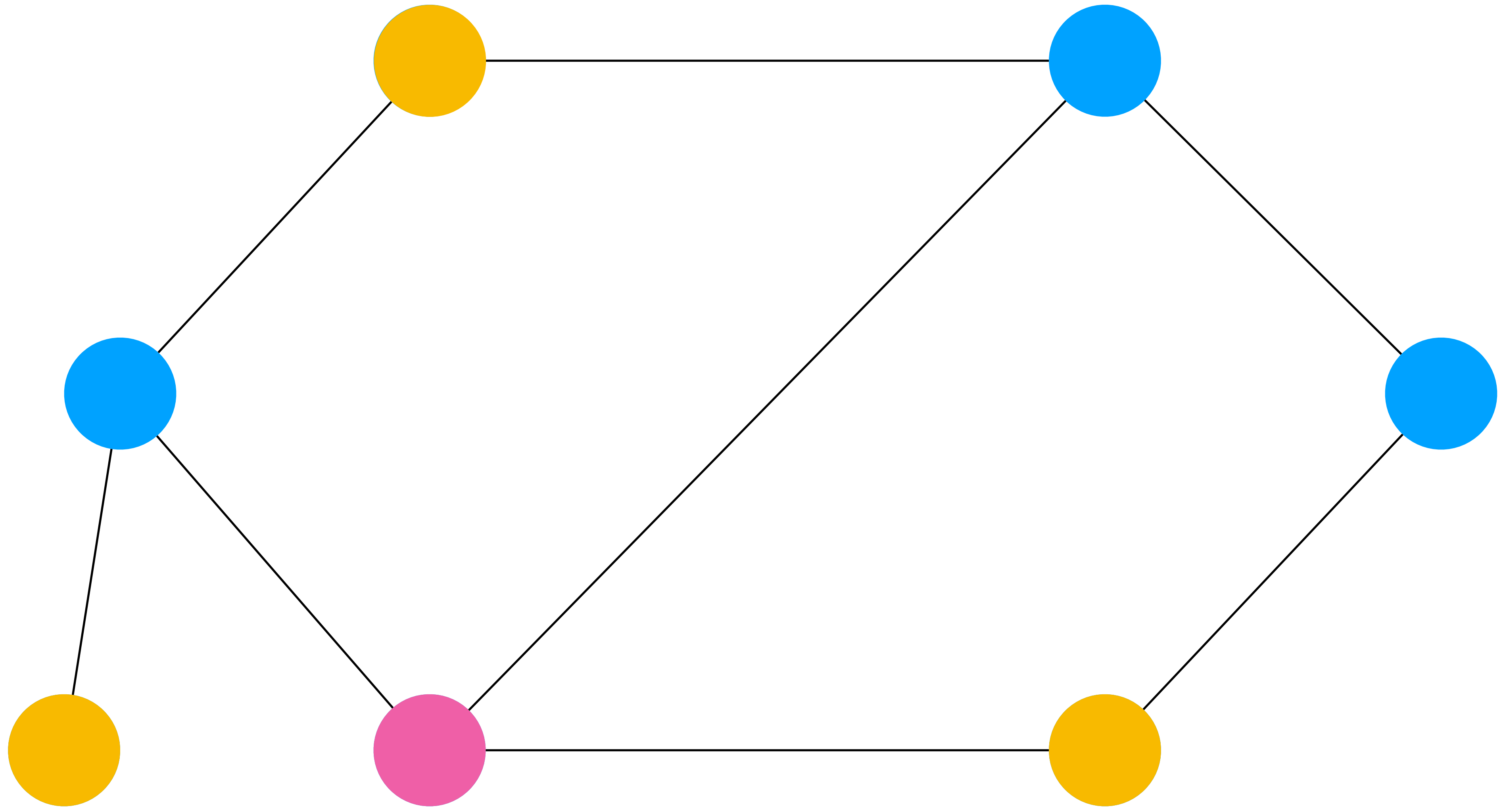


- Cops may move anywhere, but have to **“leave the graph”** to do so.
- Robber moves along the edges at **“infinite speed”**, must avoid cops.

Cops and Robbers

k Cops

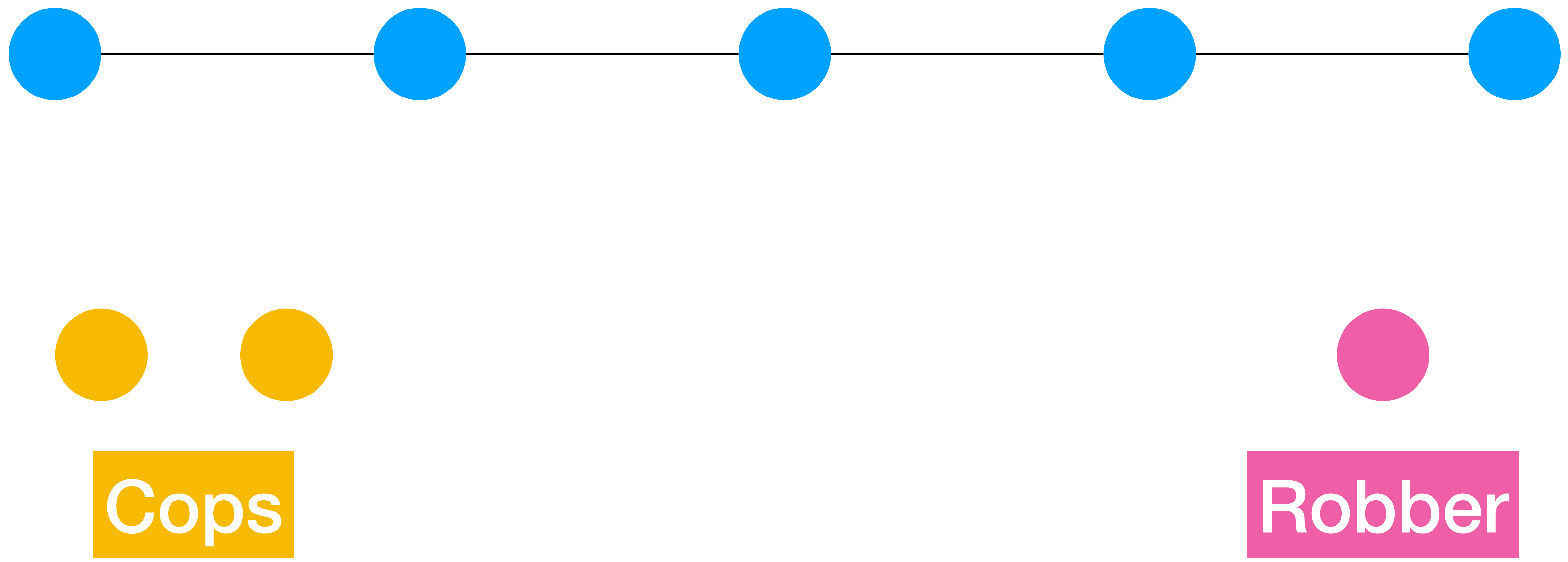
Robber



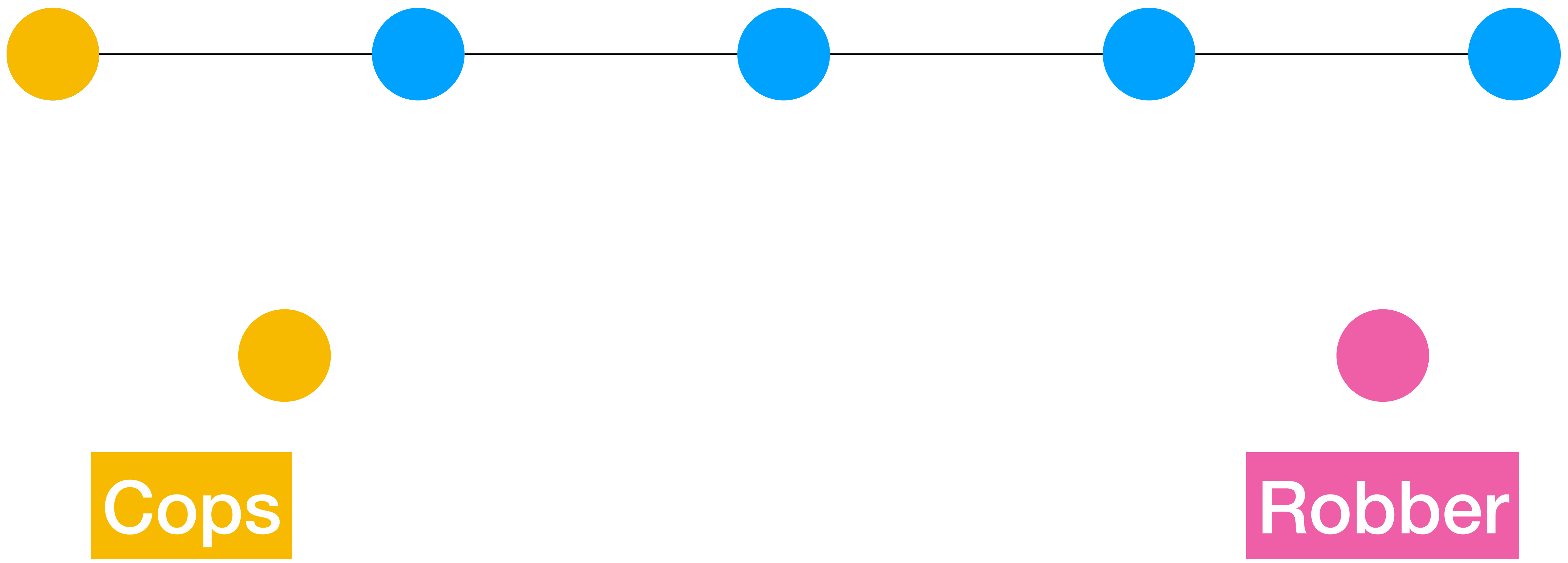
Cops win the game if the robber is caught.

- Cops may move anywhere, but have to **"leave the graph"** to do so.
- Robber moves along the edges at **"infinite speed"**, must avoid cops.

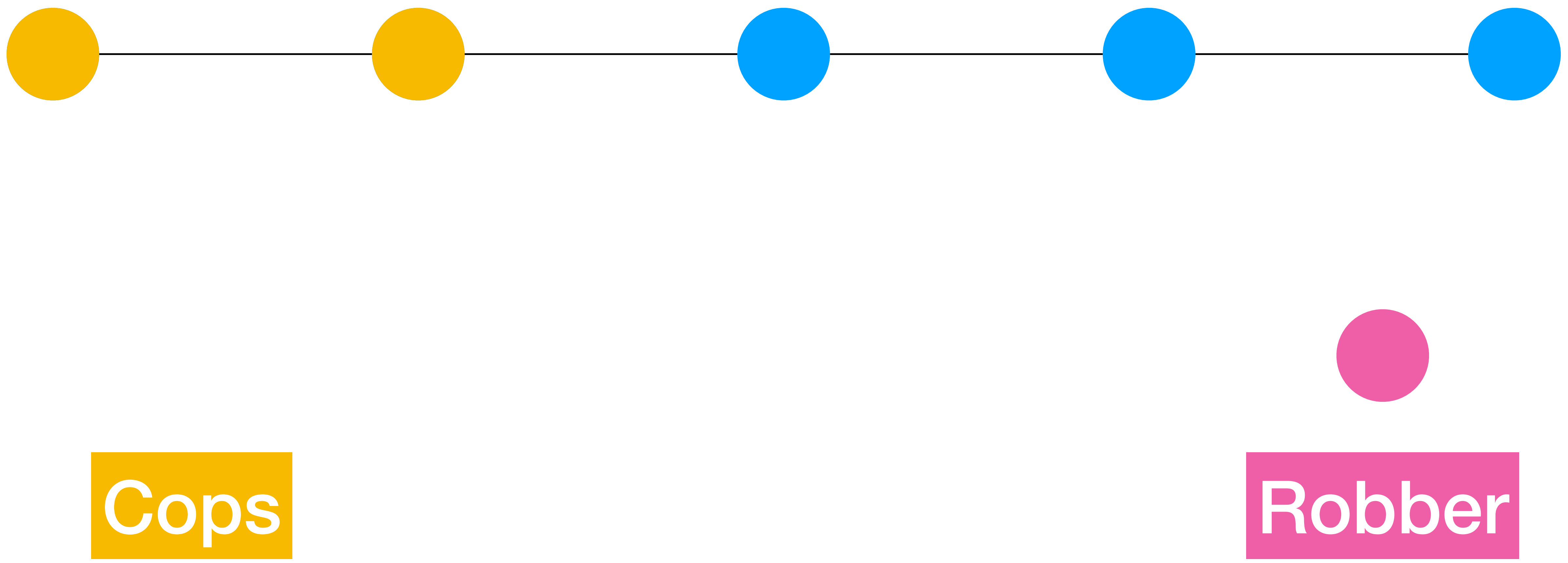
Cops and Robbers



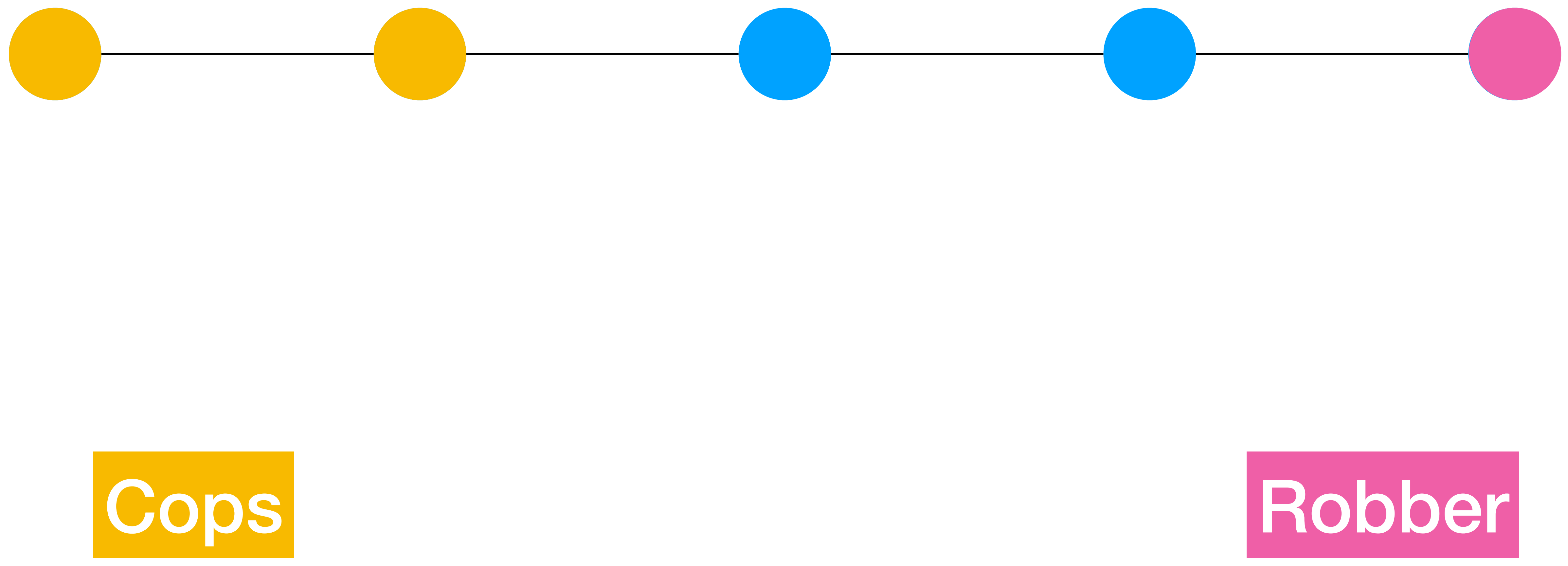
Cops and Robbers



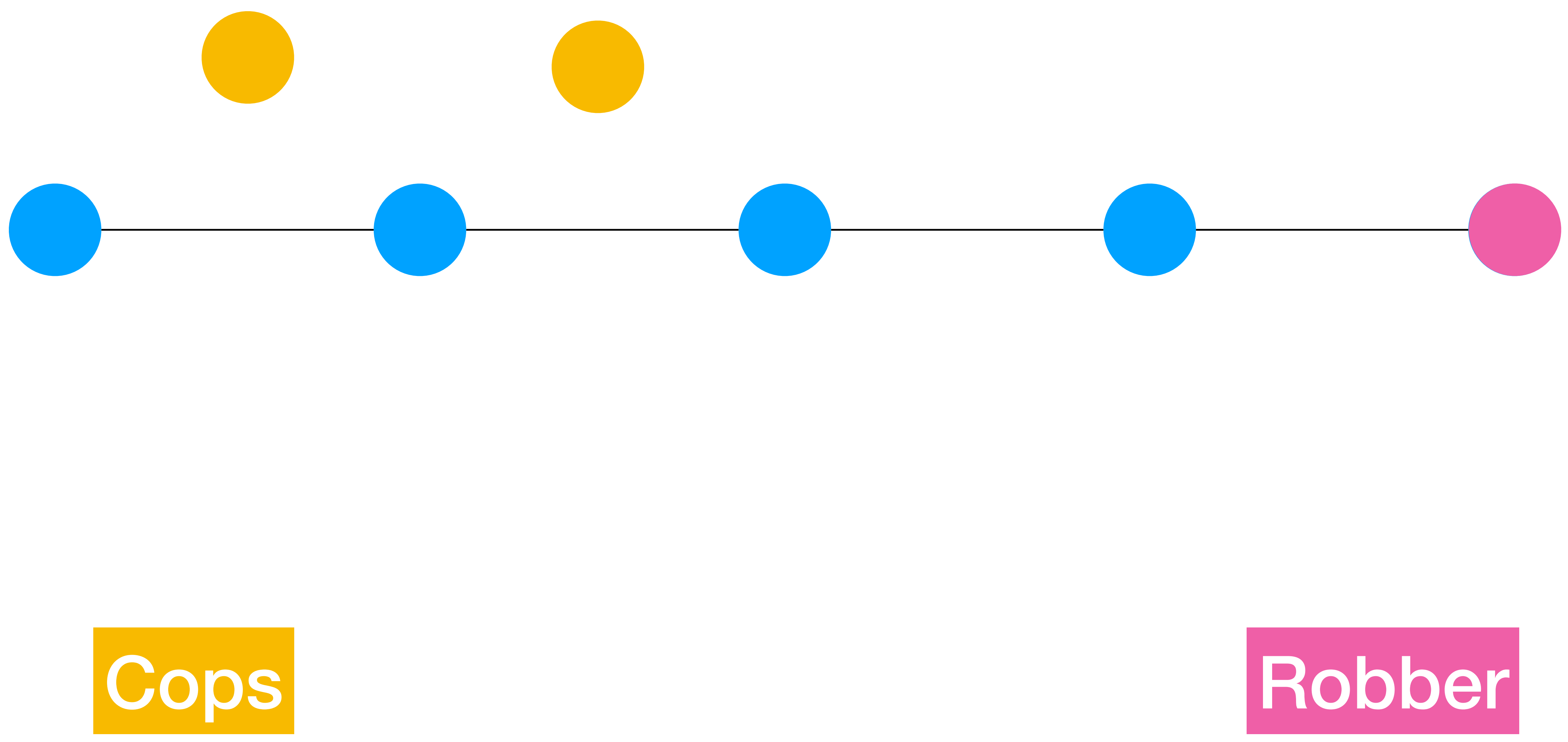
Cops and Robbers



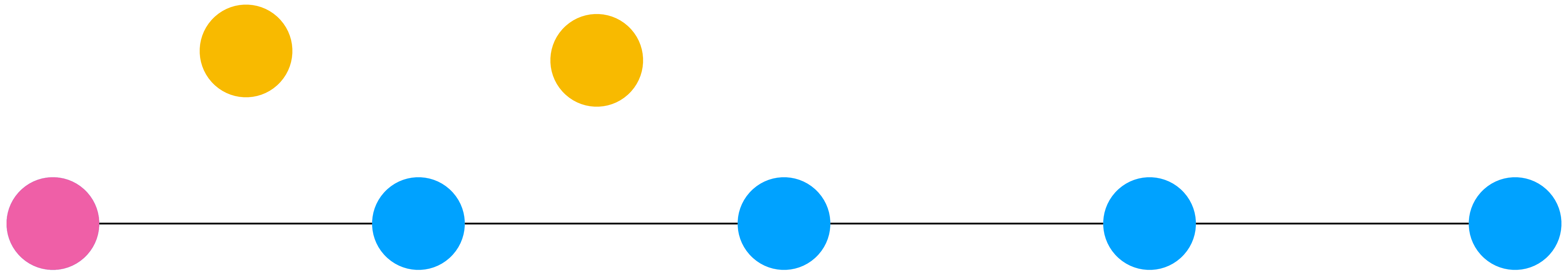
Cops and Robbers



Cops and Robbers



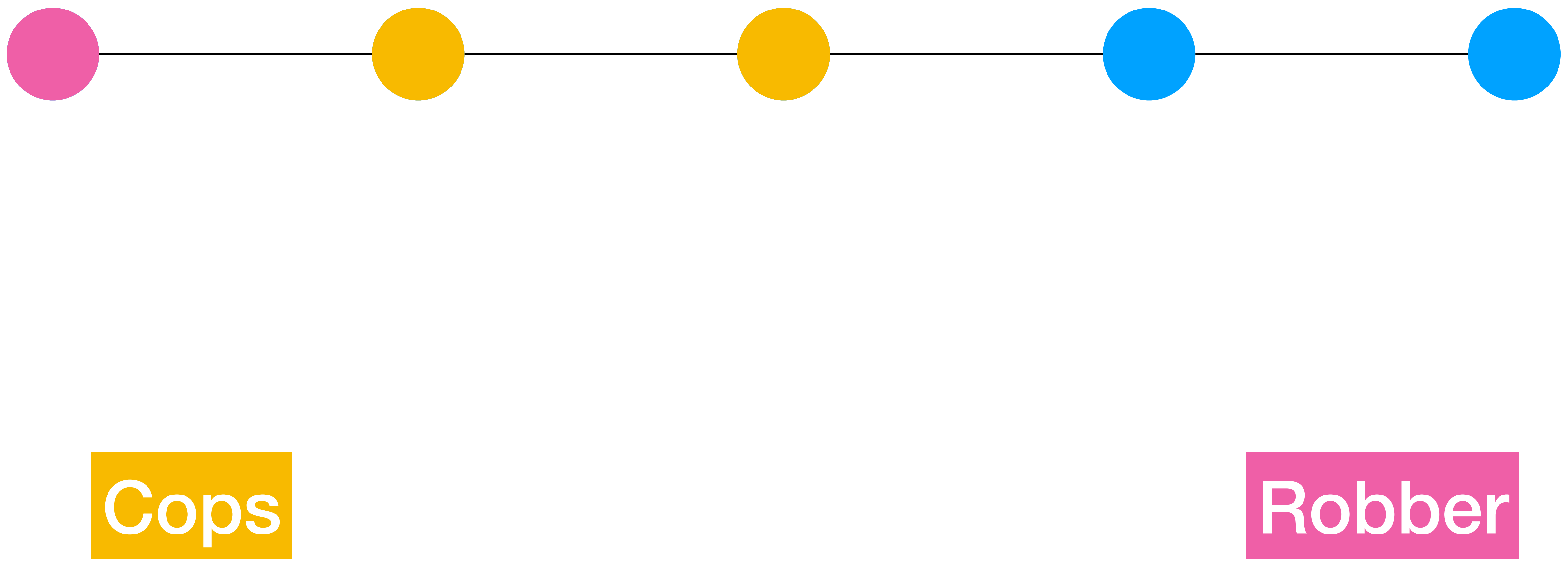
Cops and Robbers



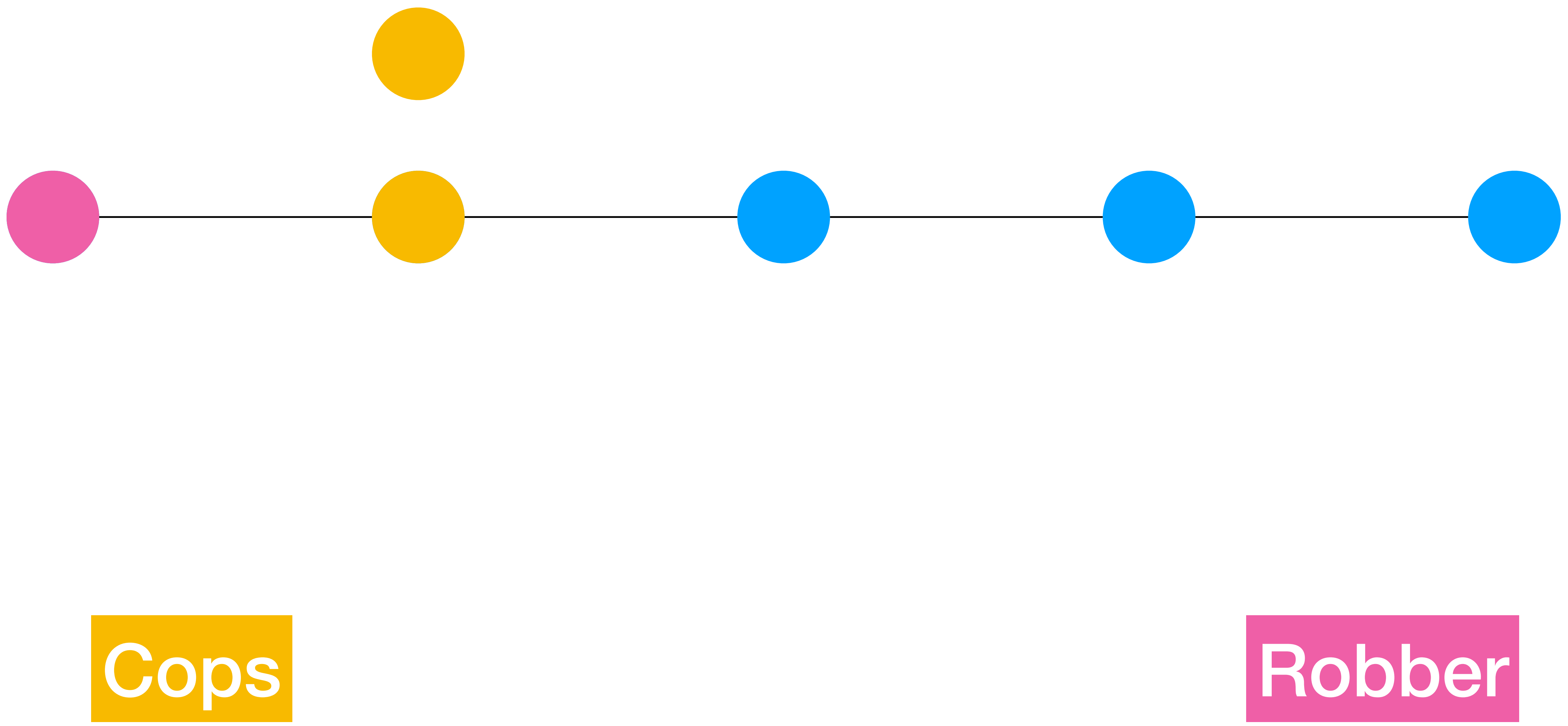
Cops

Robber

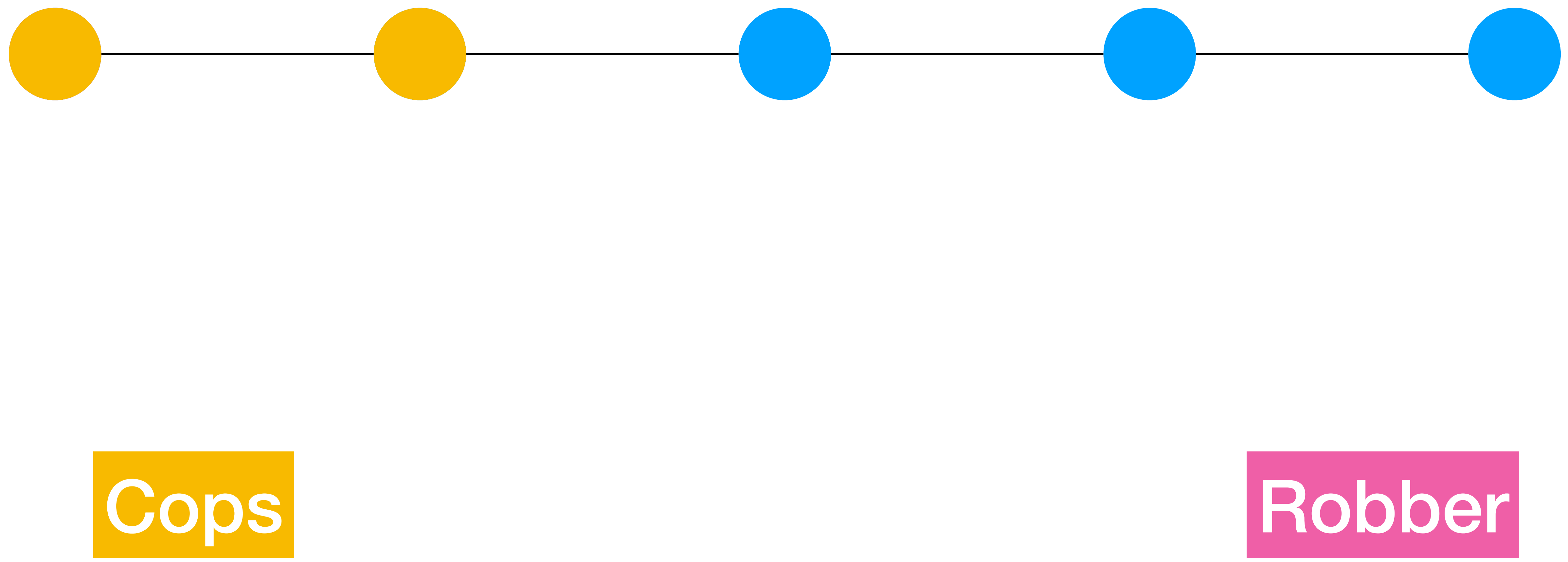
Cops and Robbers



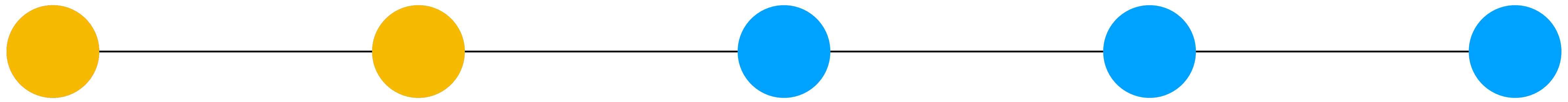
Cops and Robbers



Cops and Robbers



Cops and Robbers



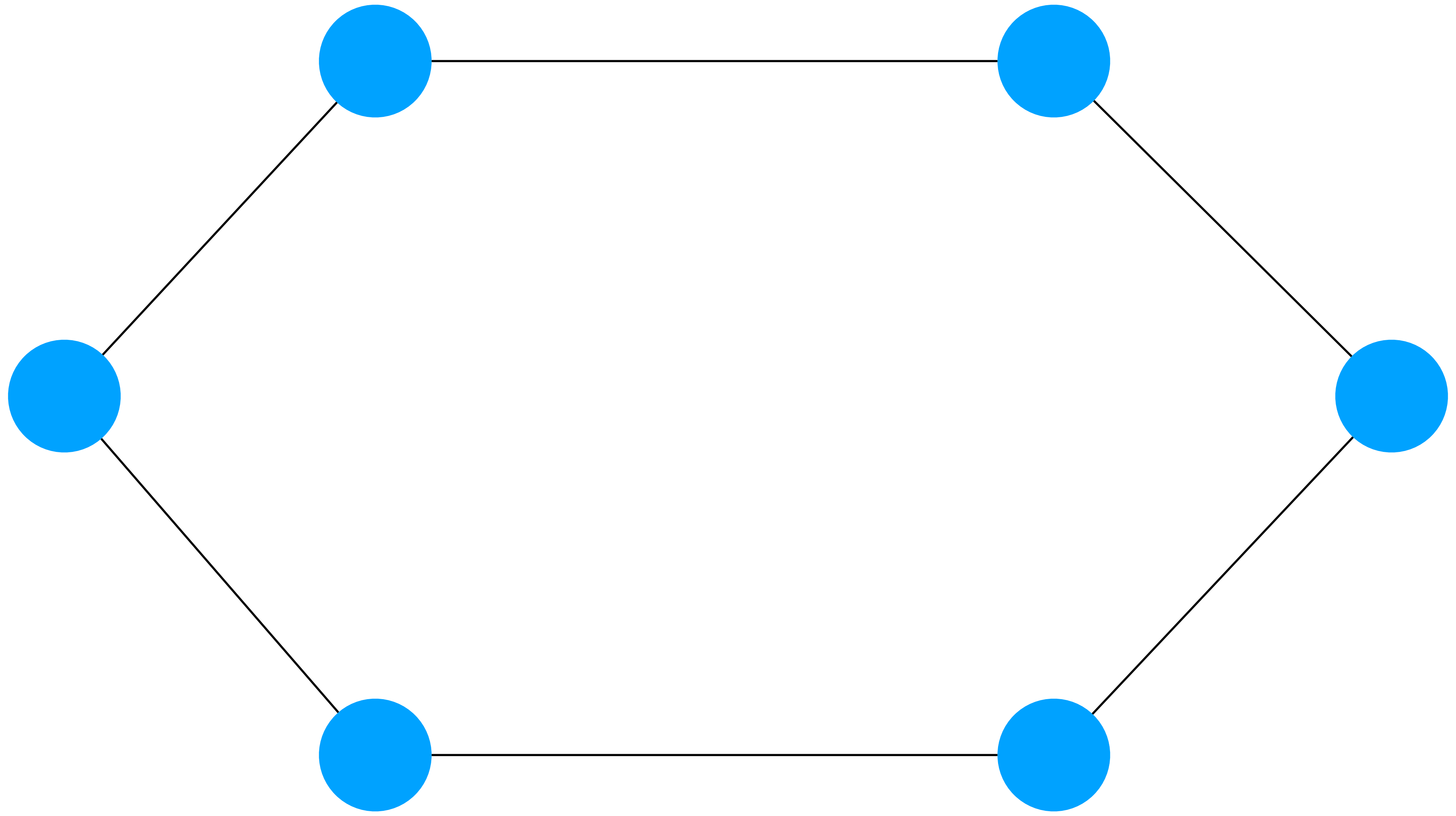
Robber caught.

Another Definition of Treewidth

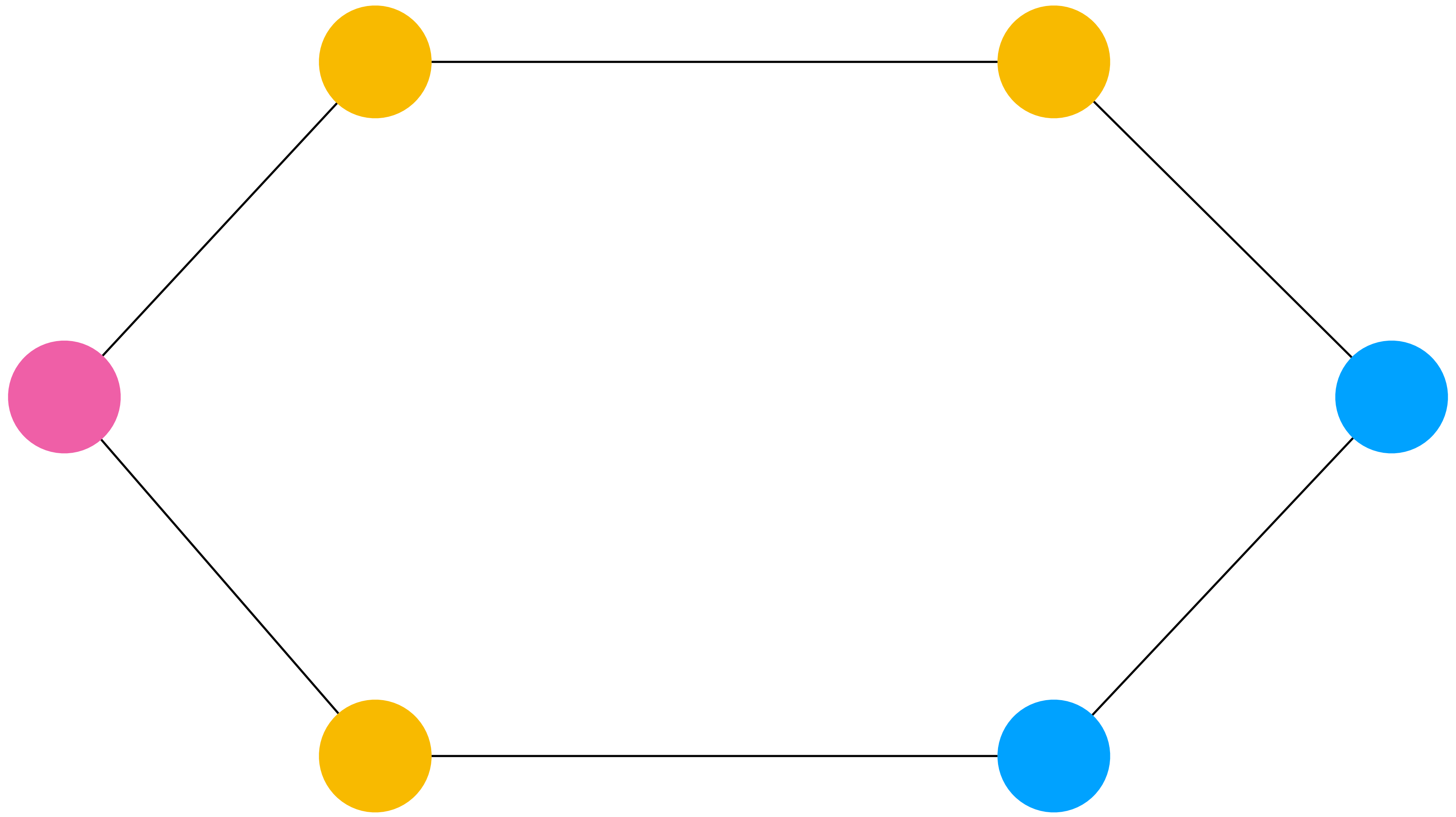
Fact

The **treewidth** of a graph G is the minimum k such that $k+1$ cops have a strategy to catch a robber in G.

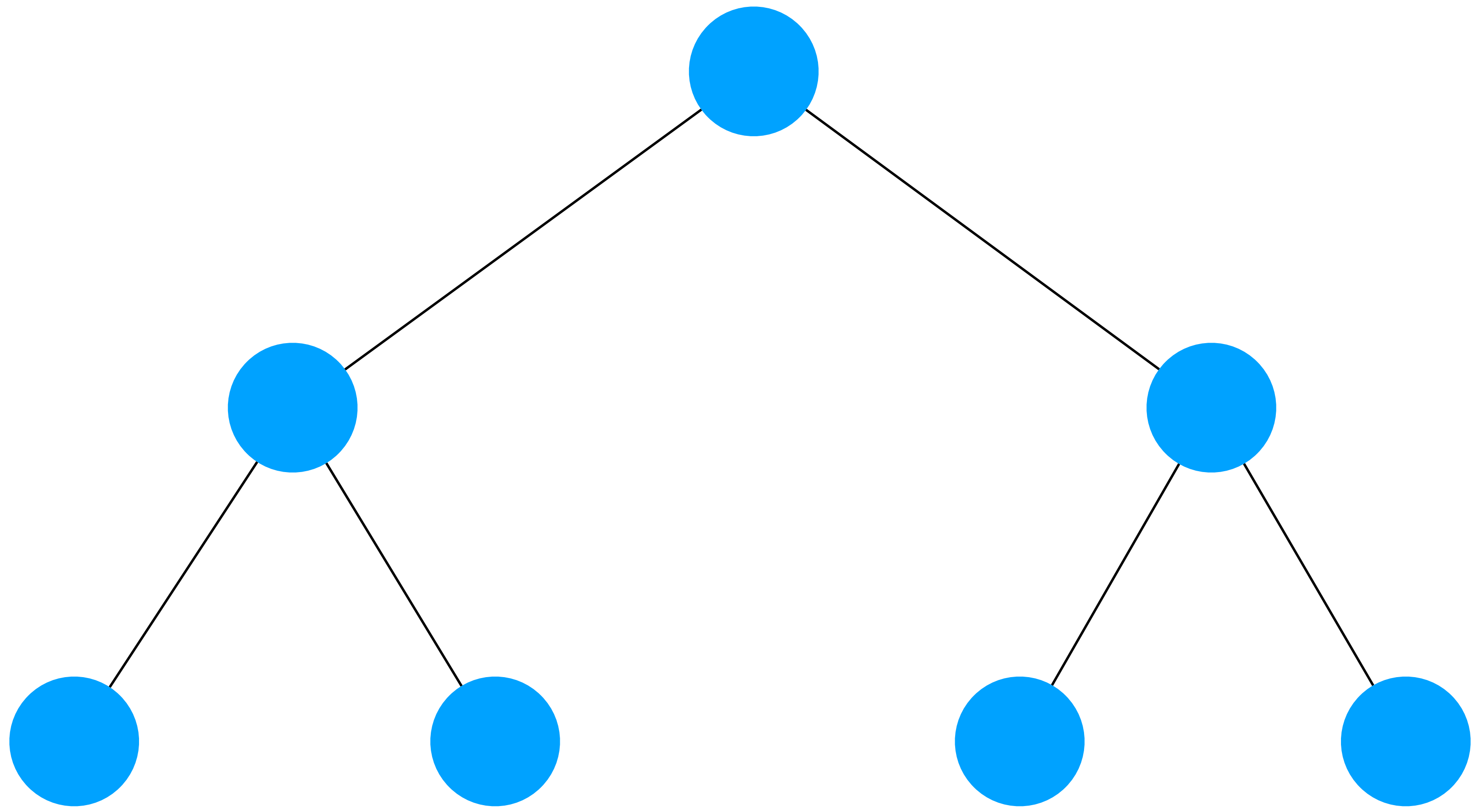
Example



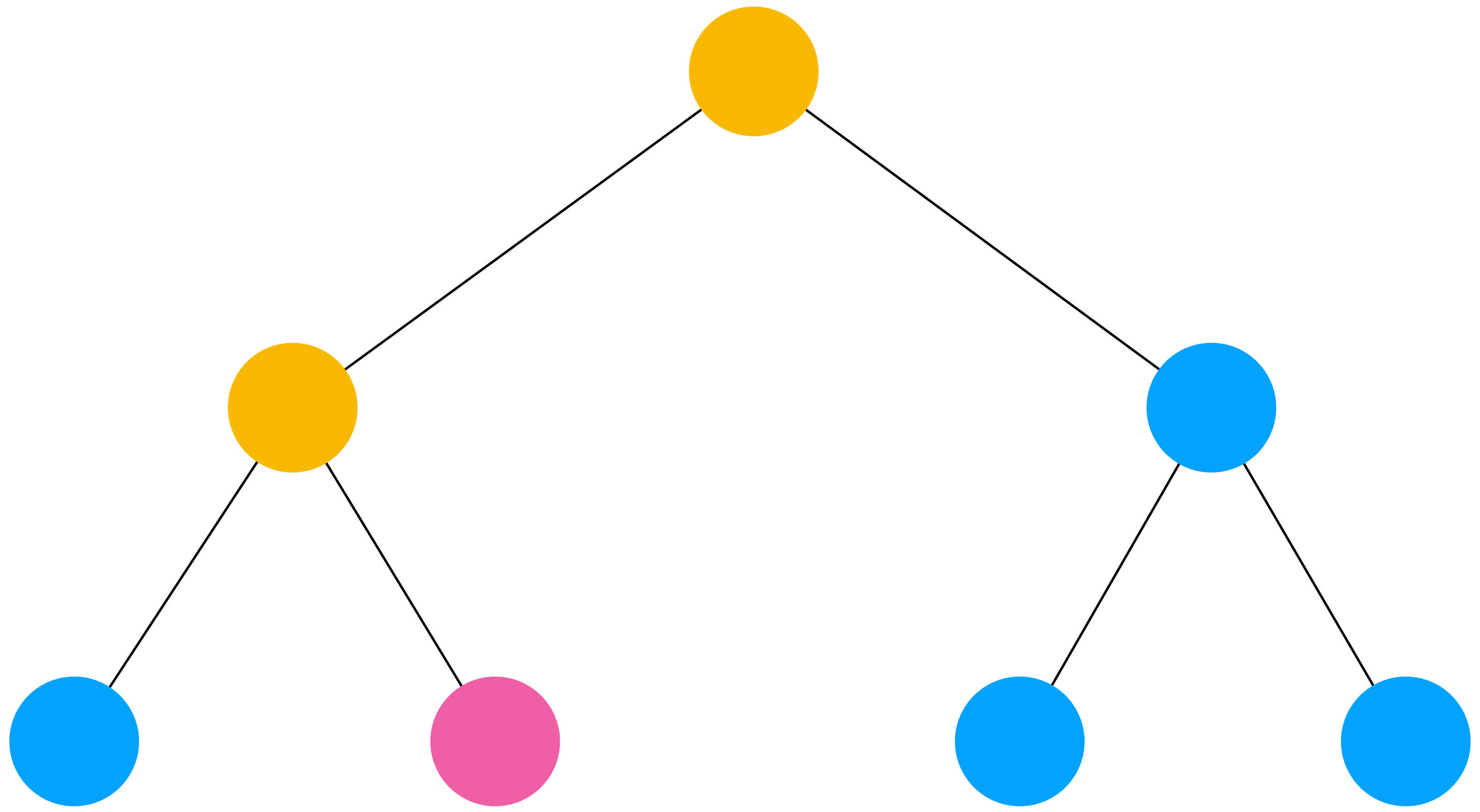
Example



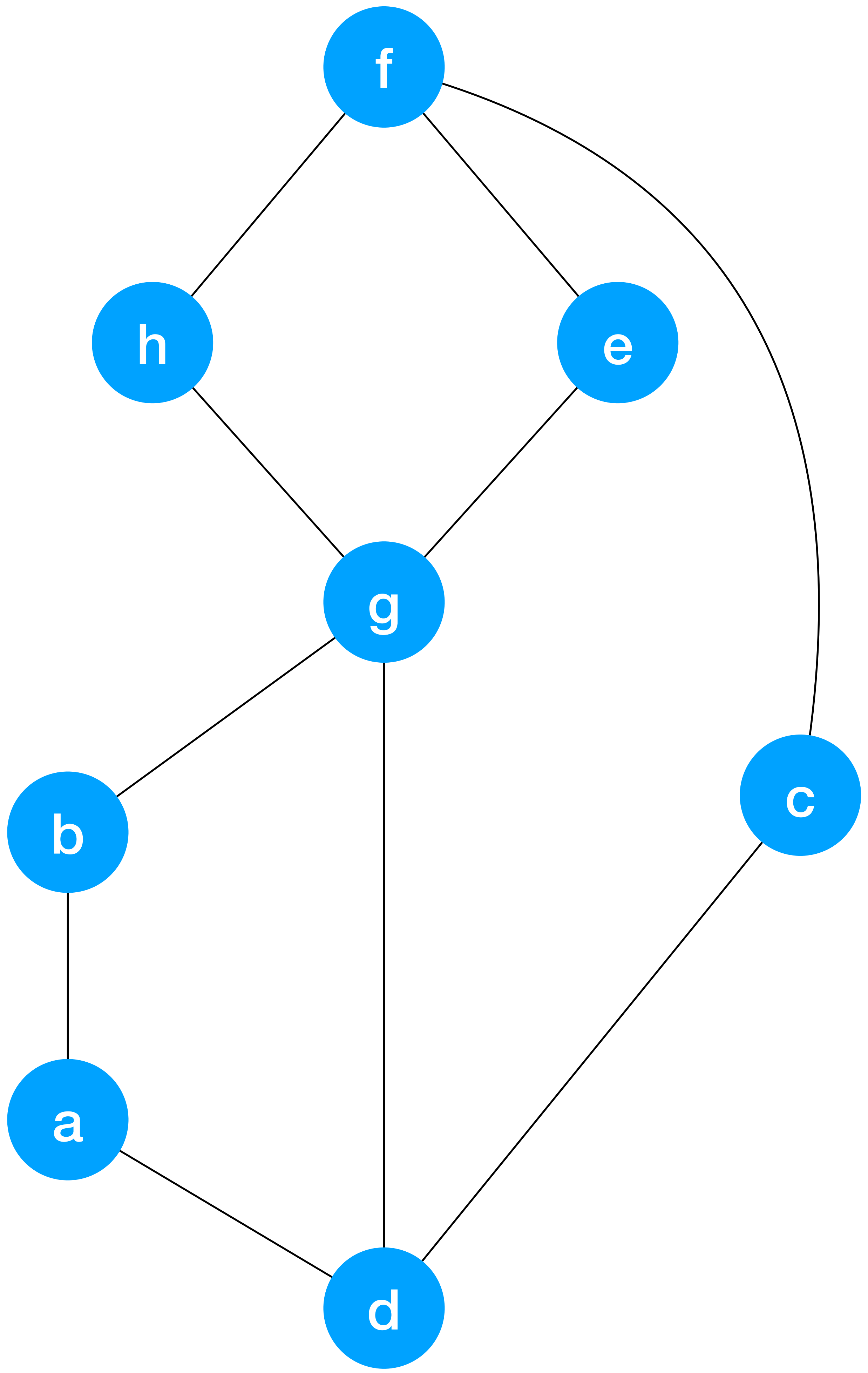
Example

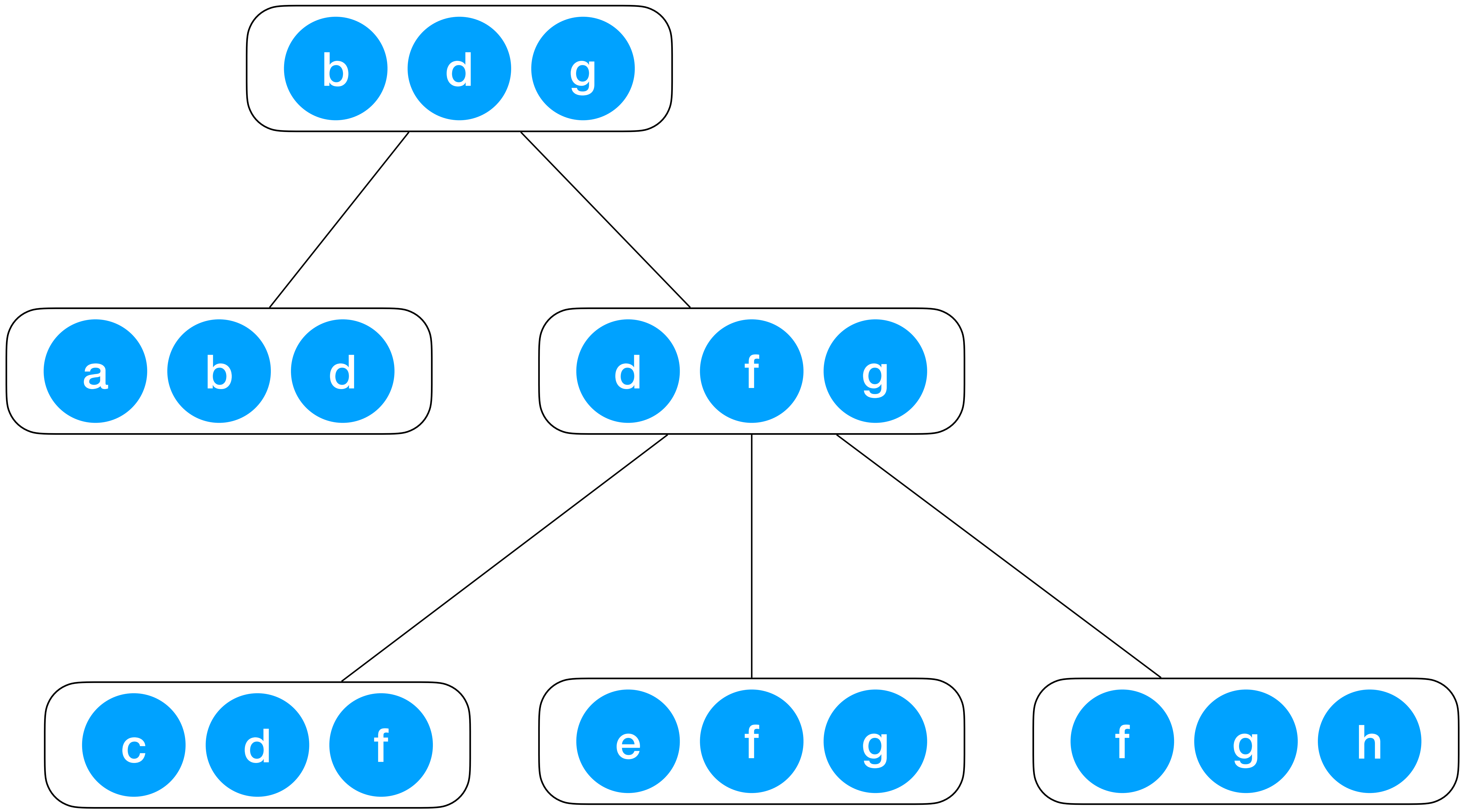


Example

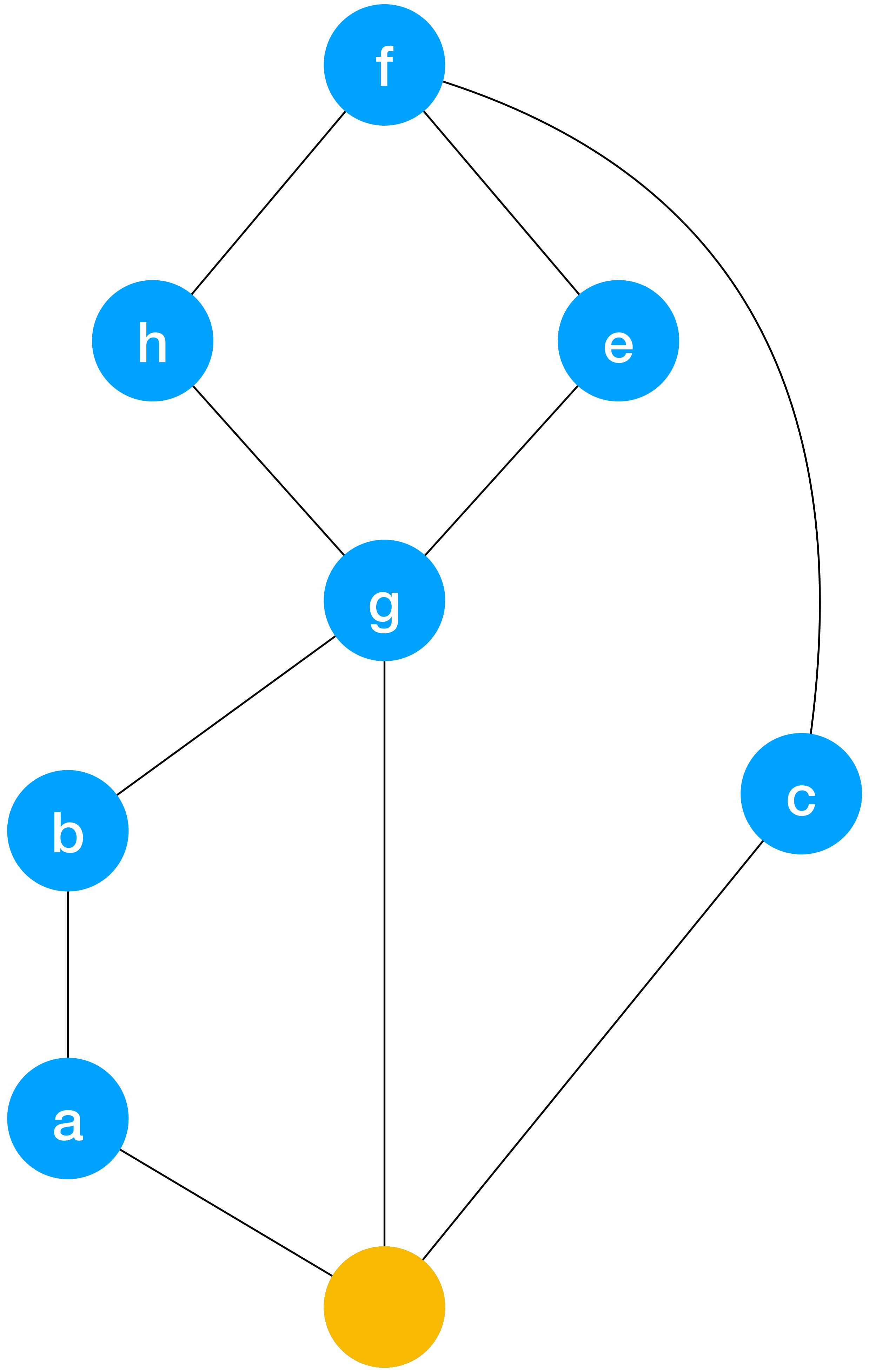


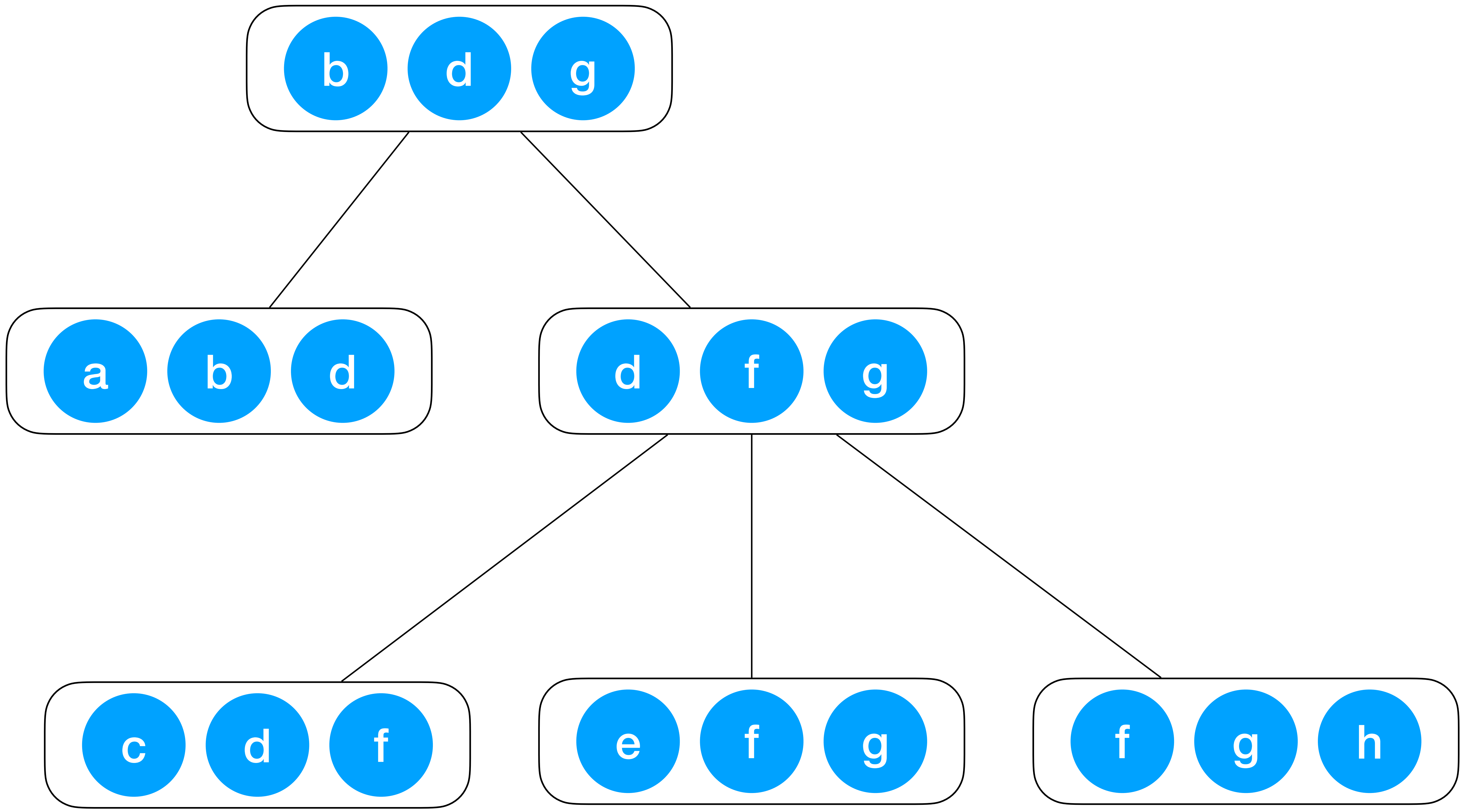
Example



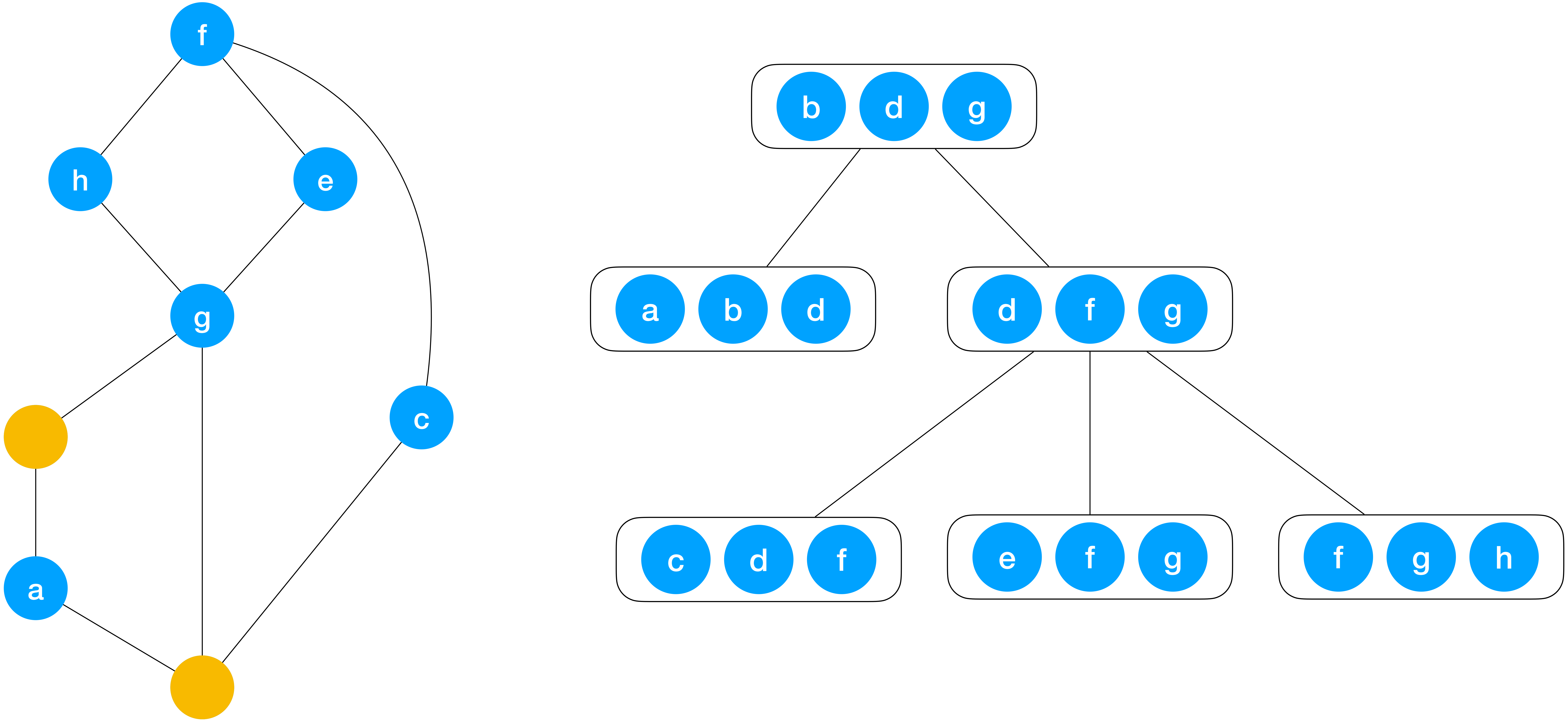


Example

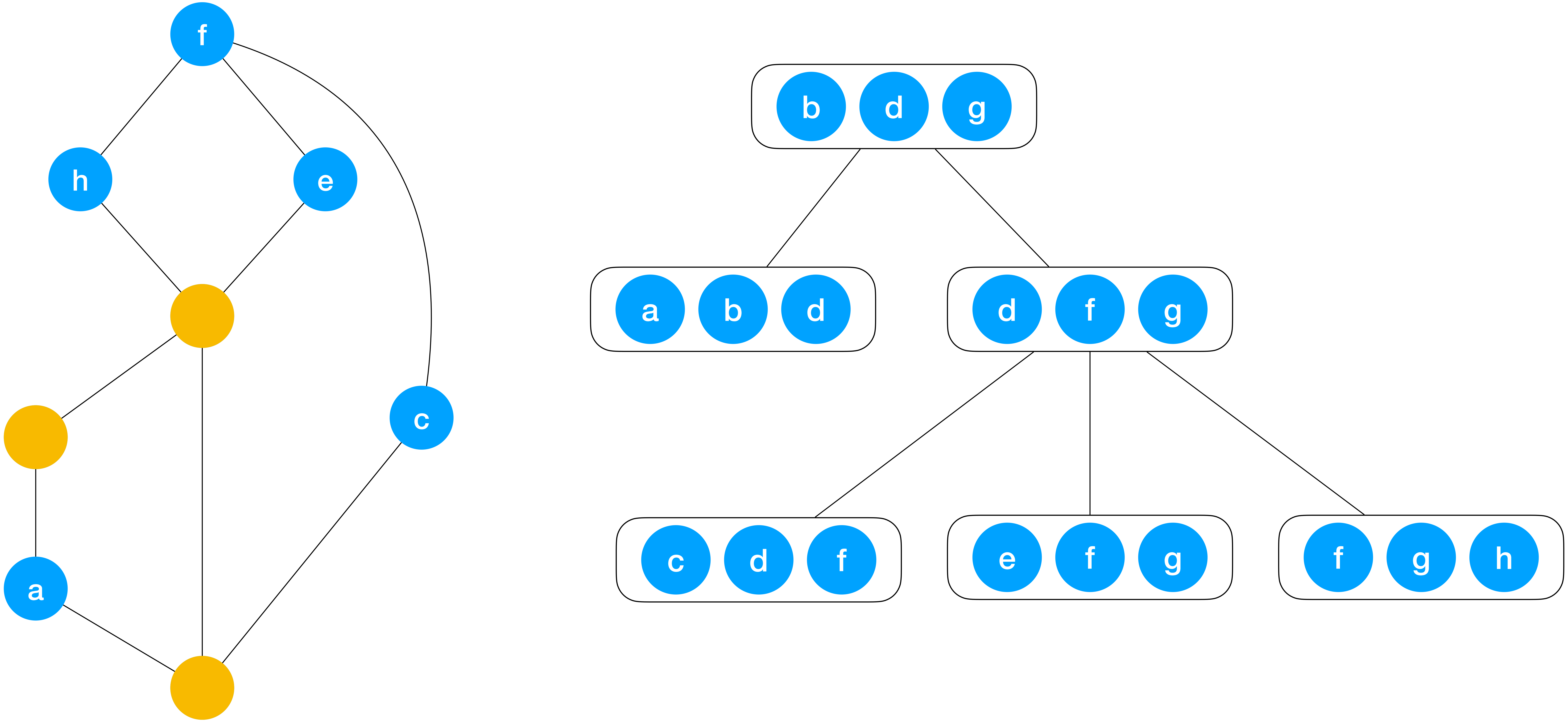




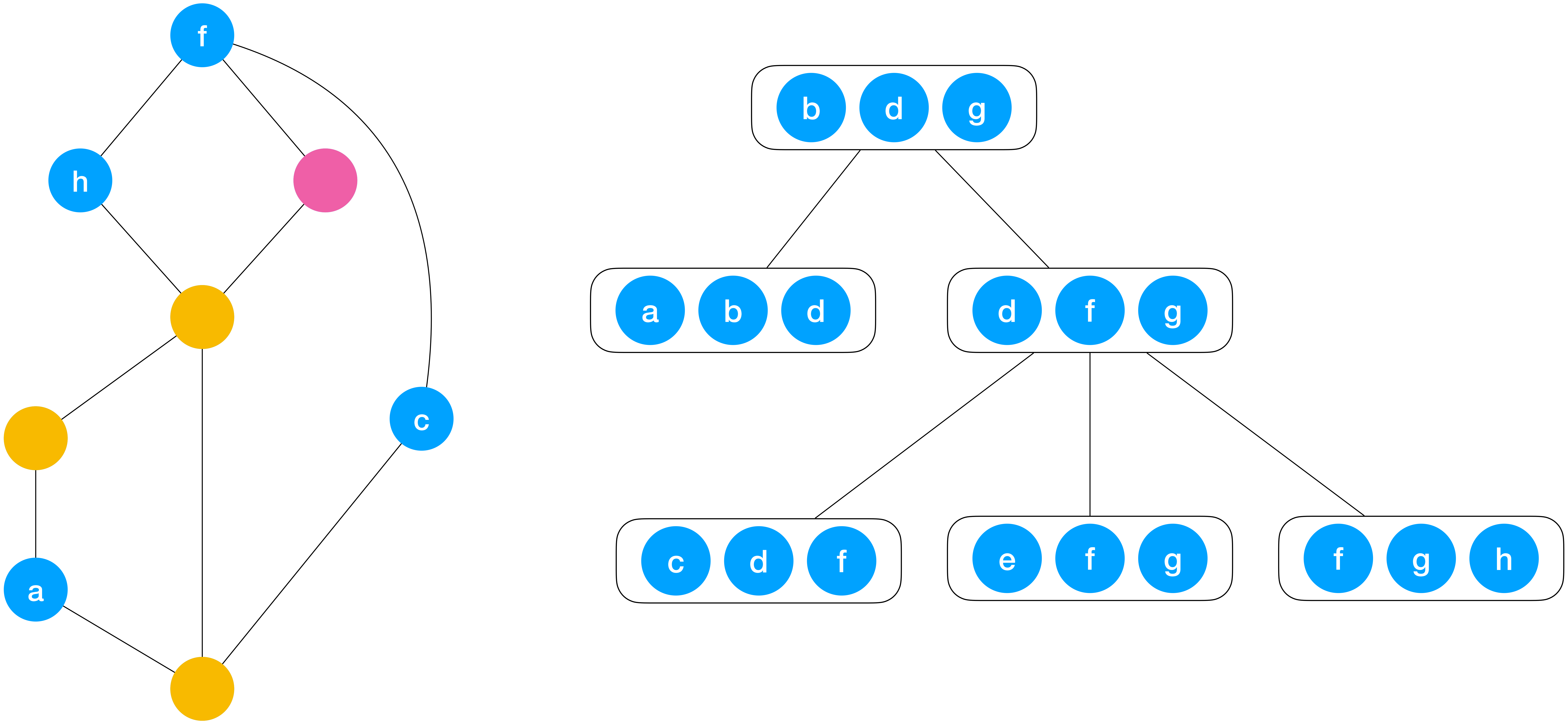
Example



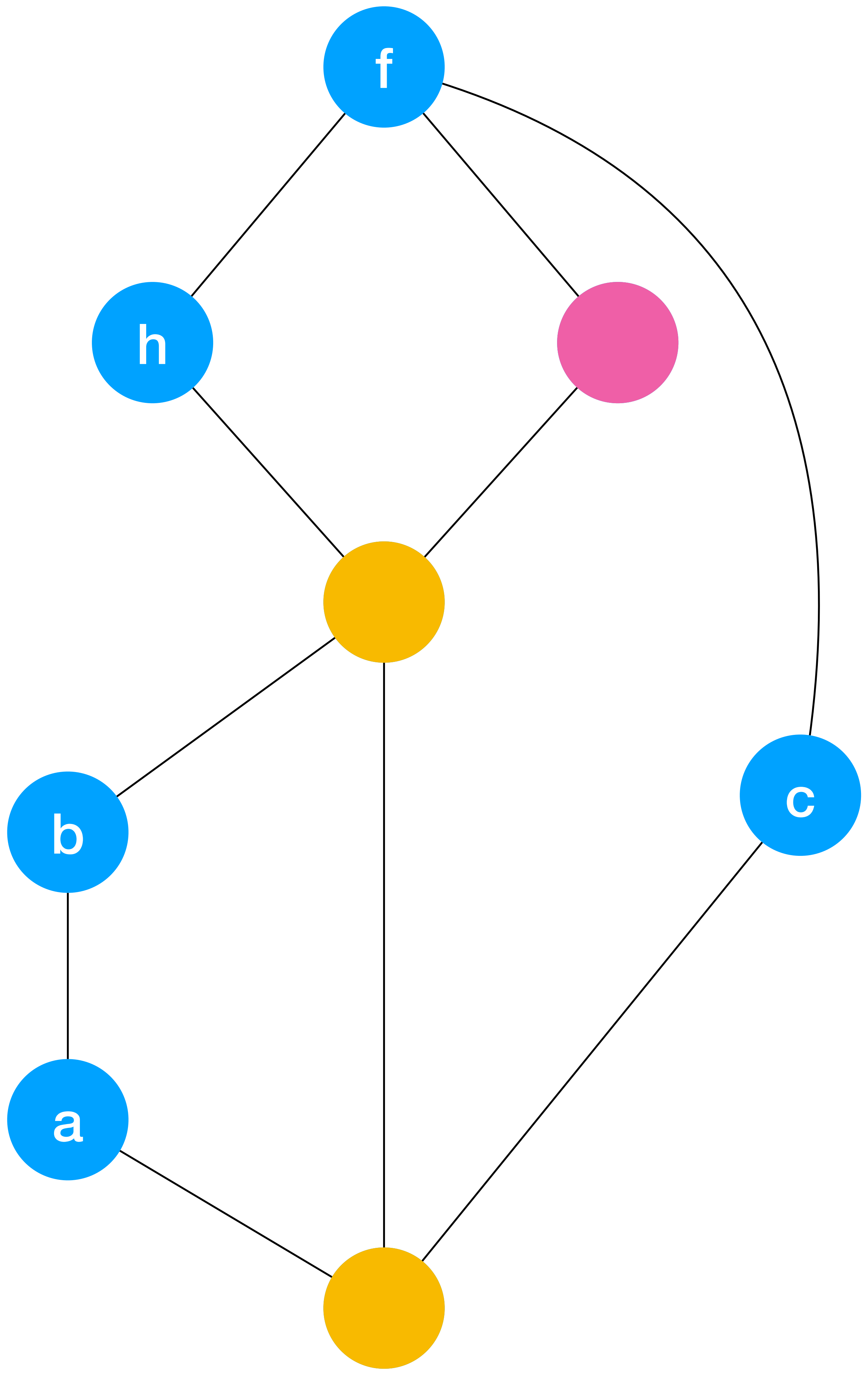
Example

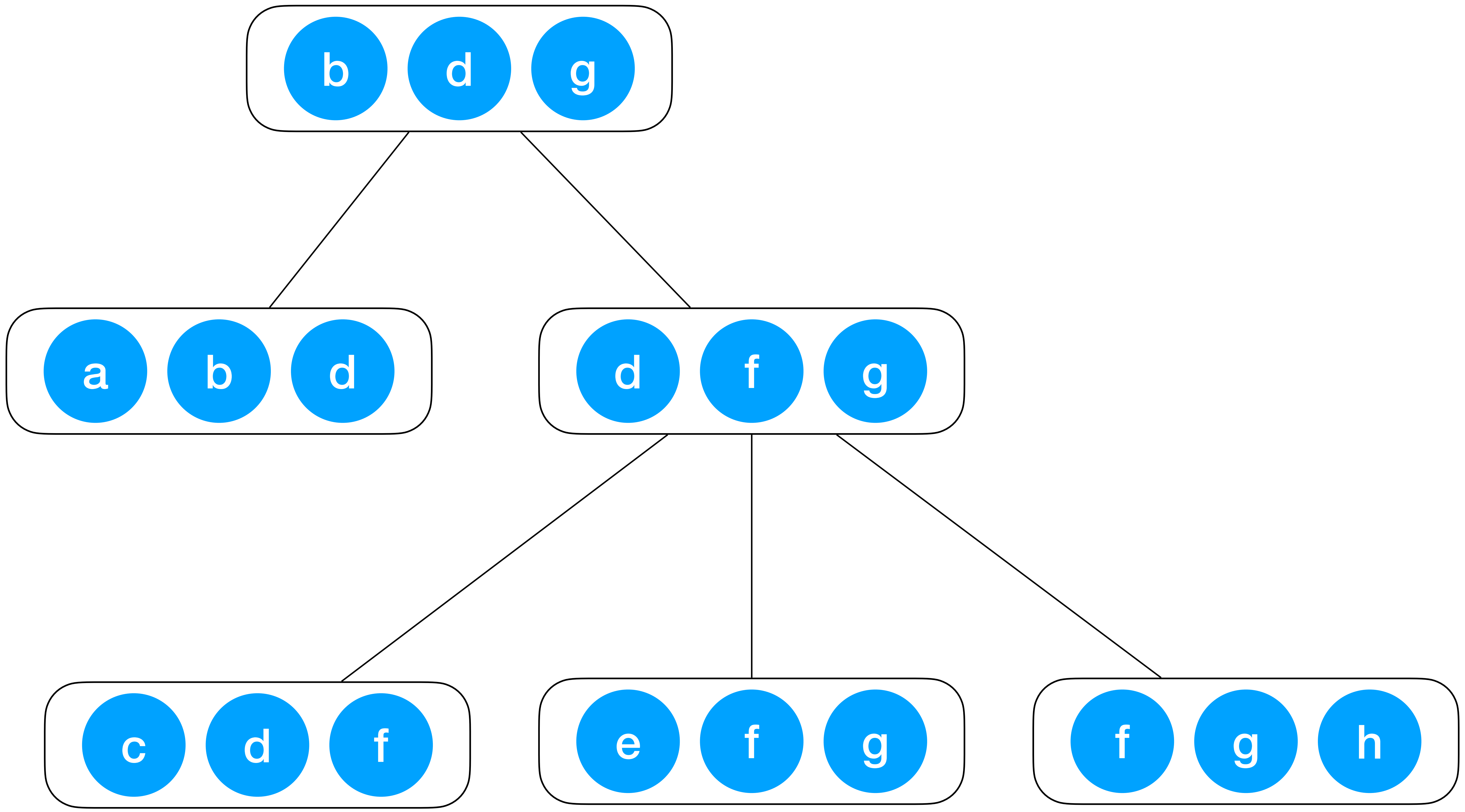


Example

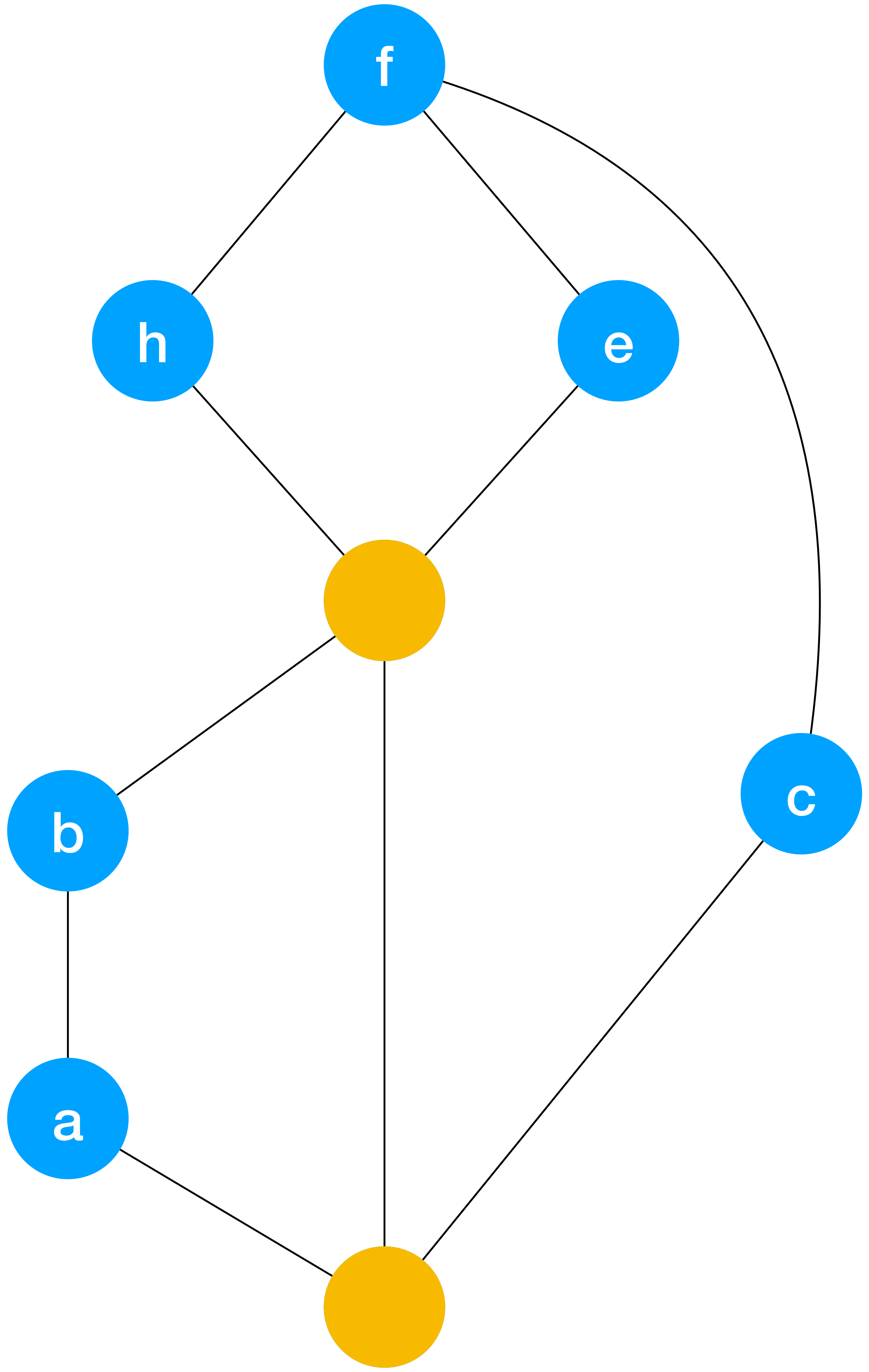


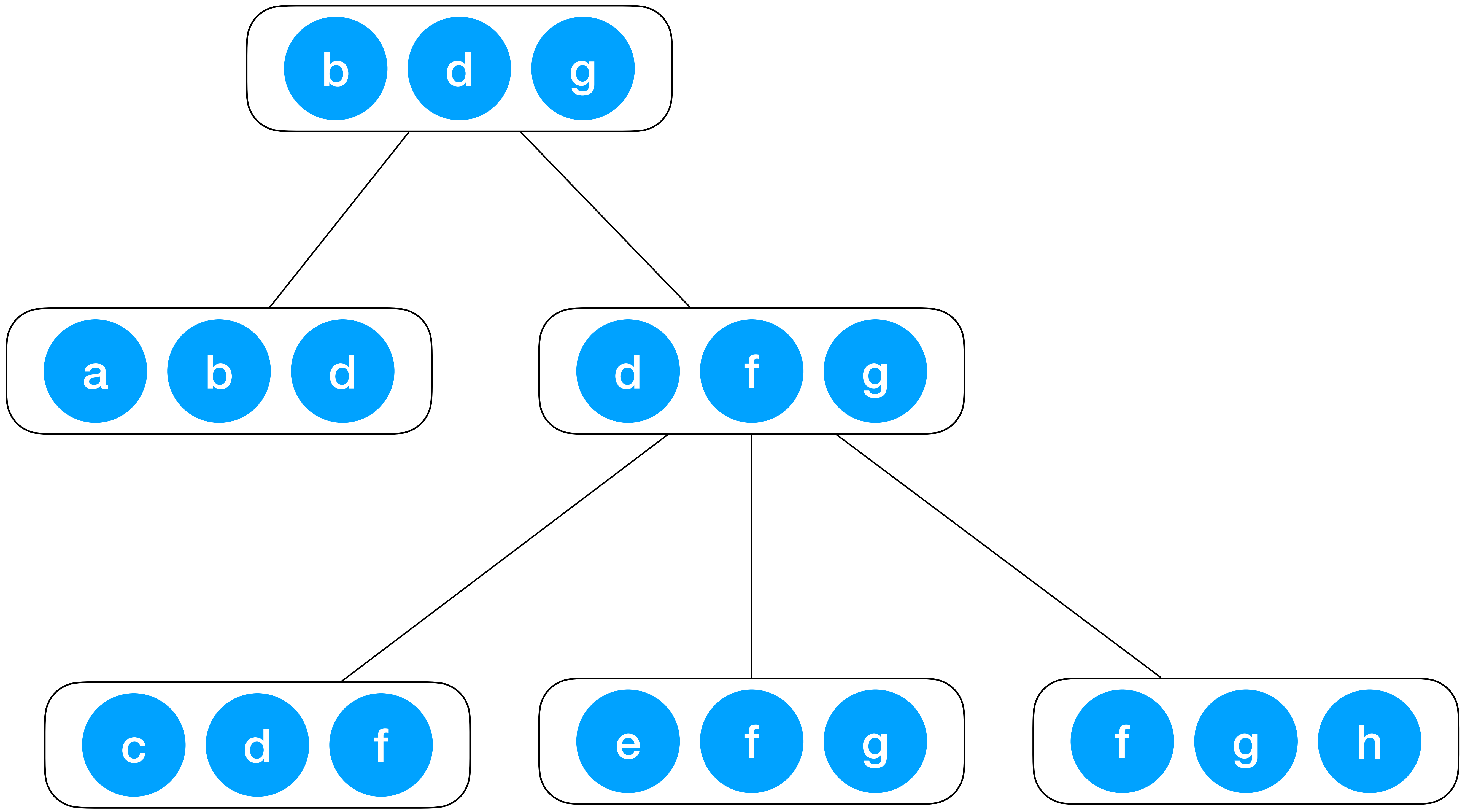
Example



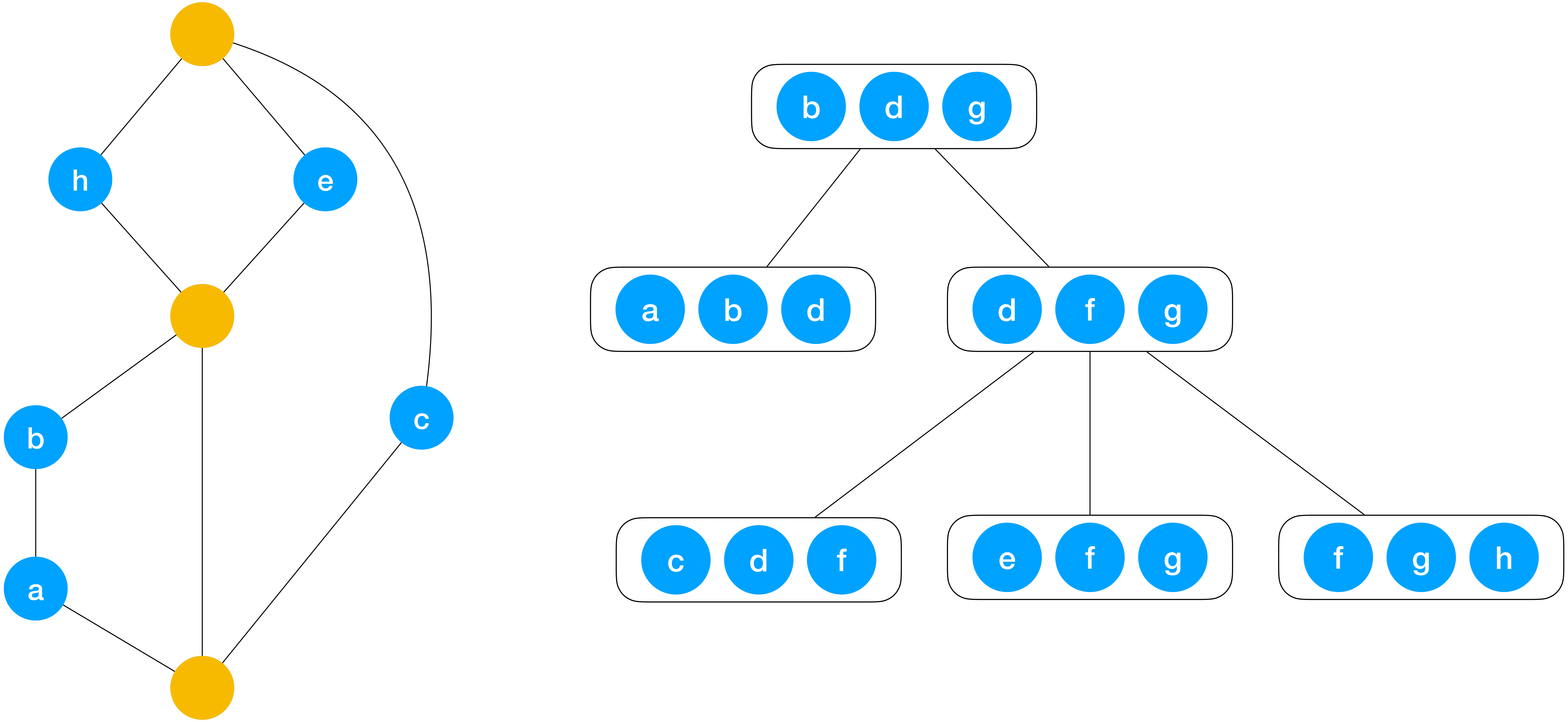


Example

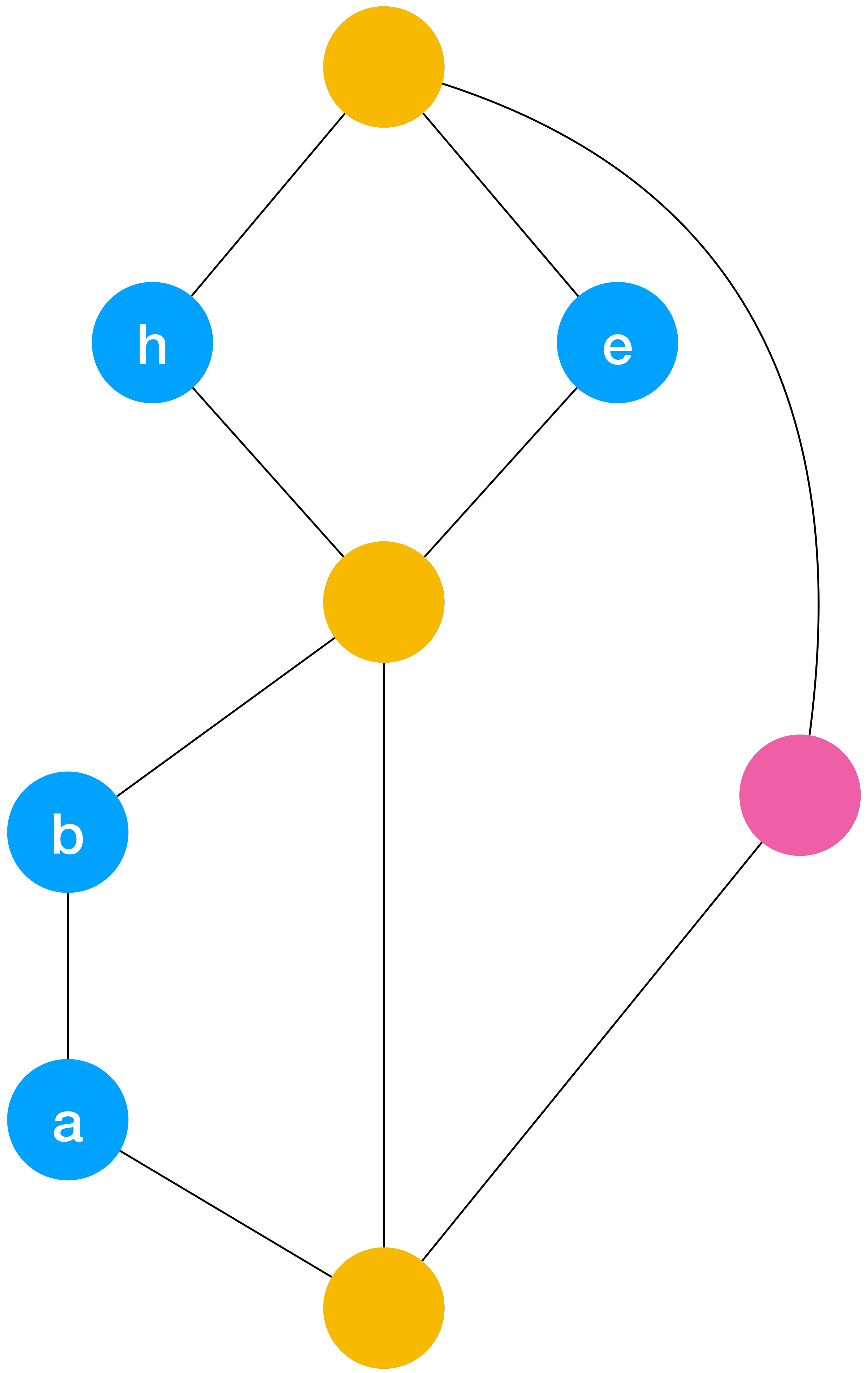


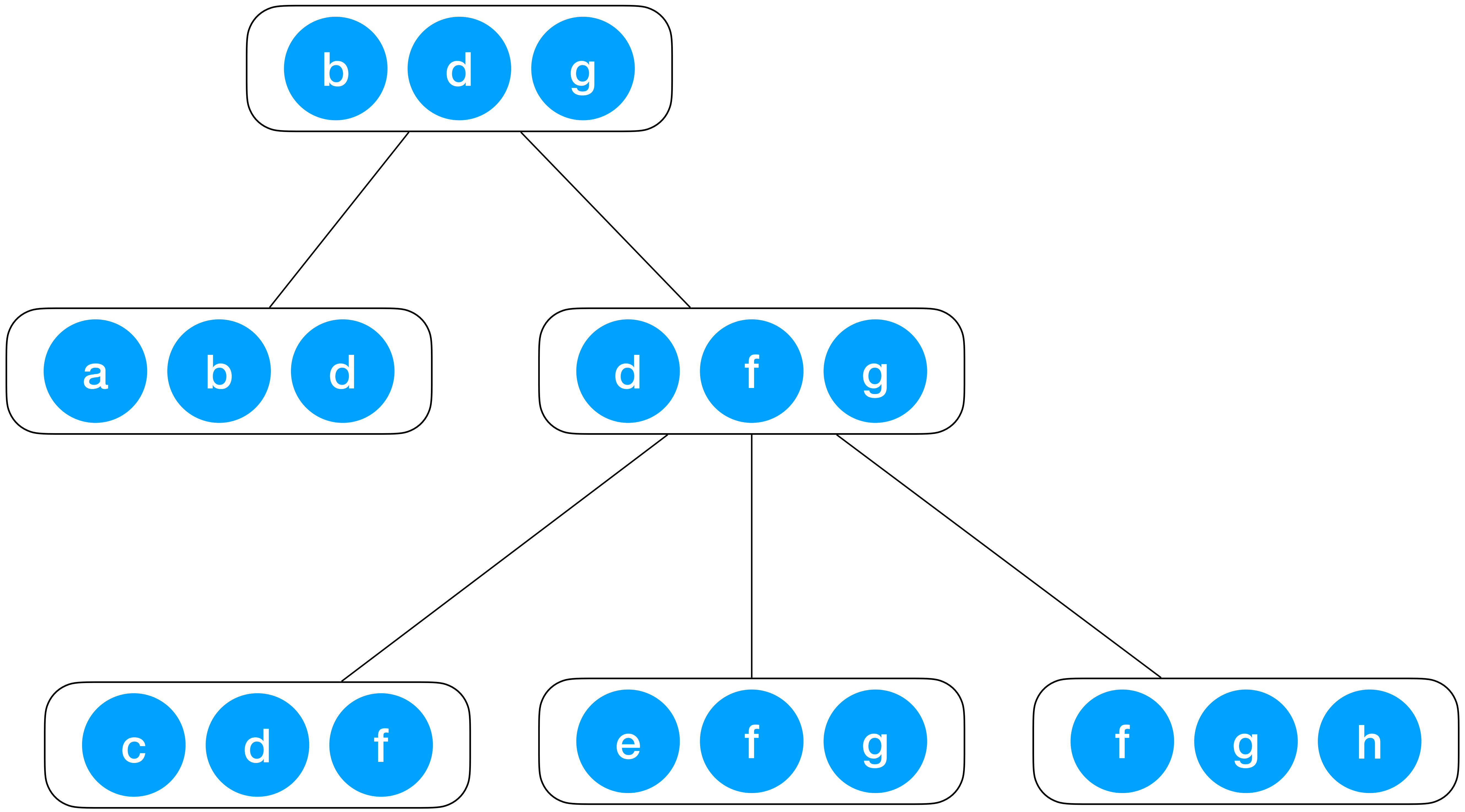


Example

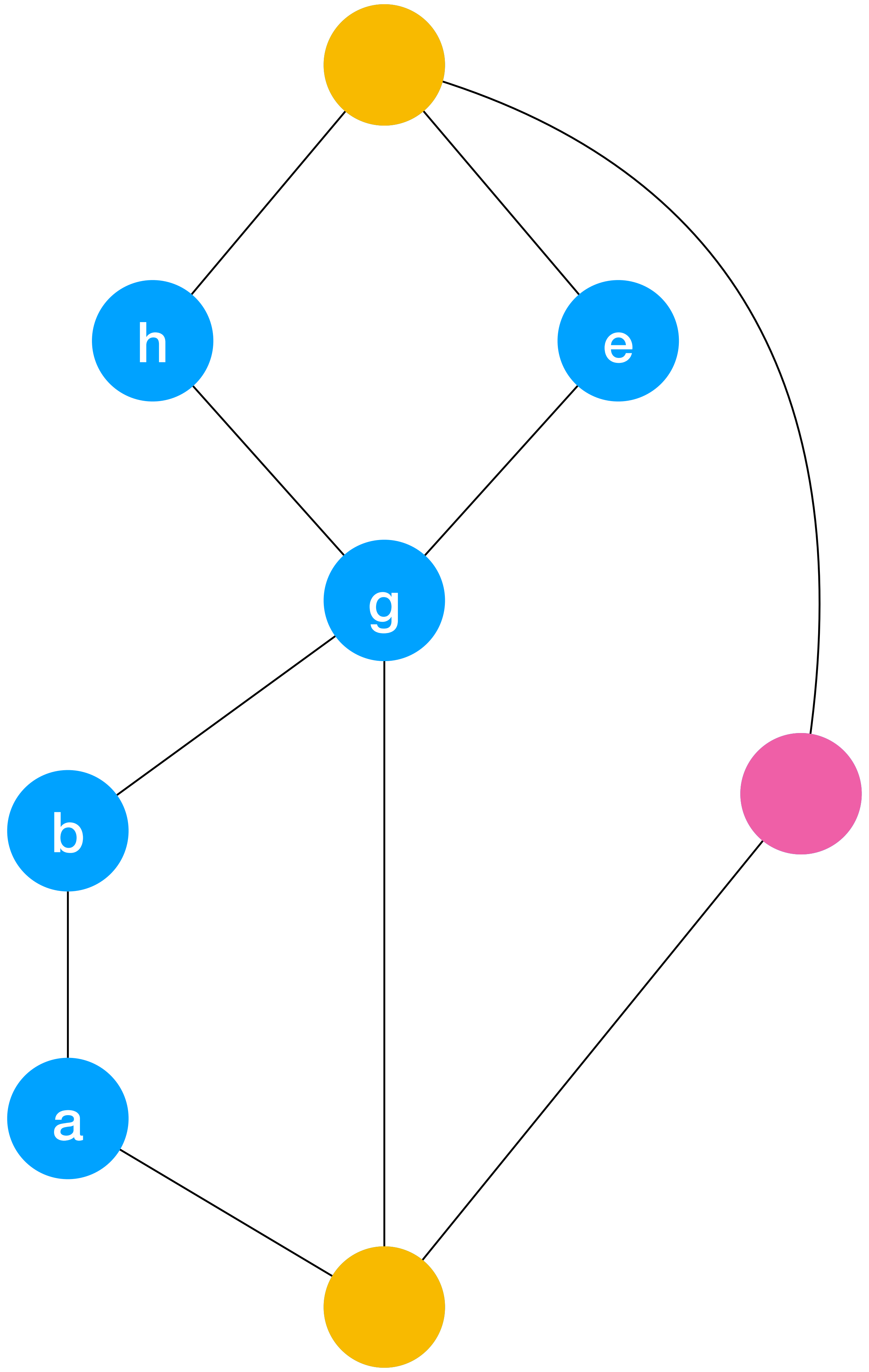


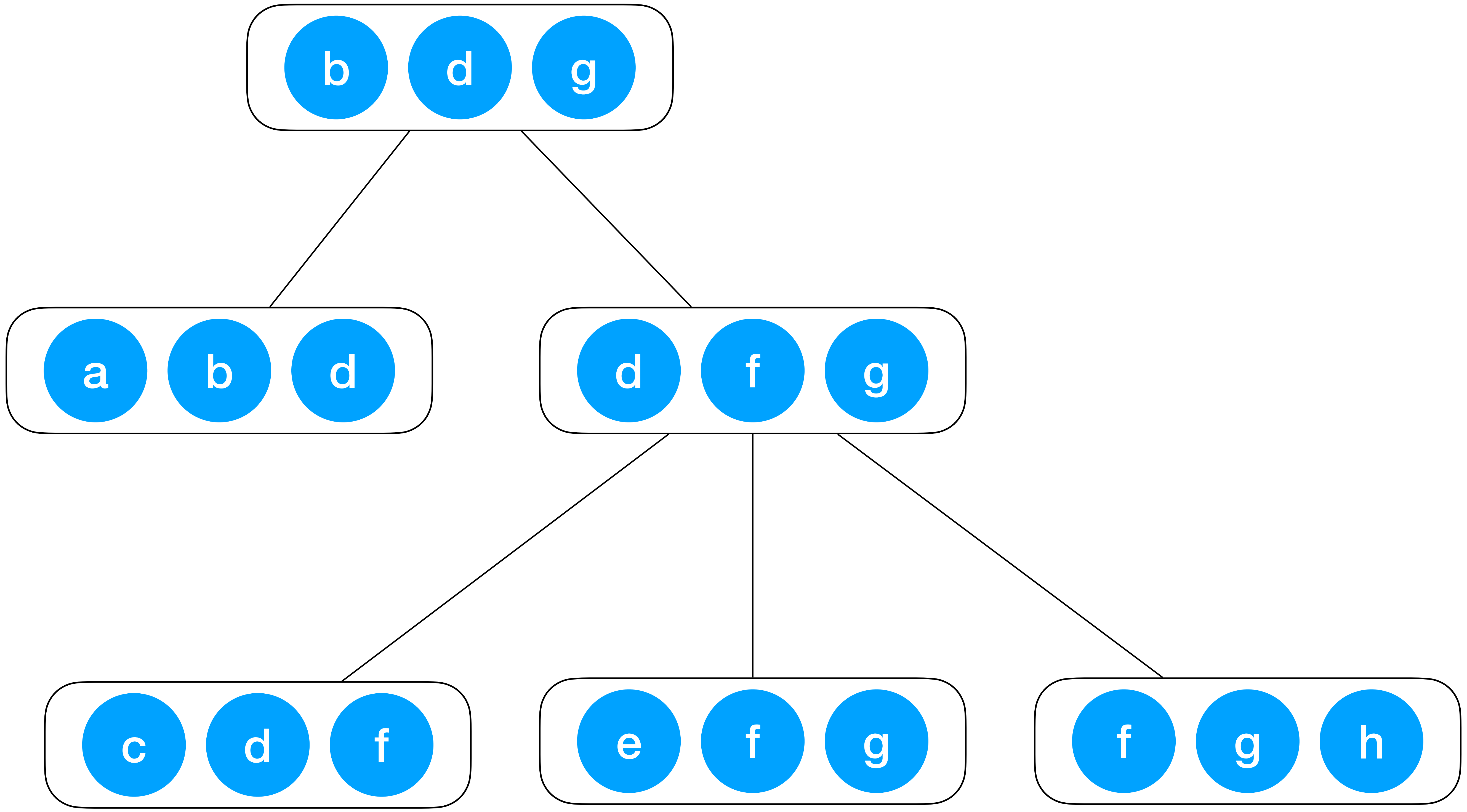
Example



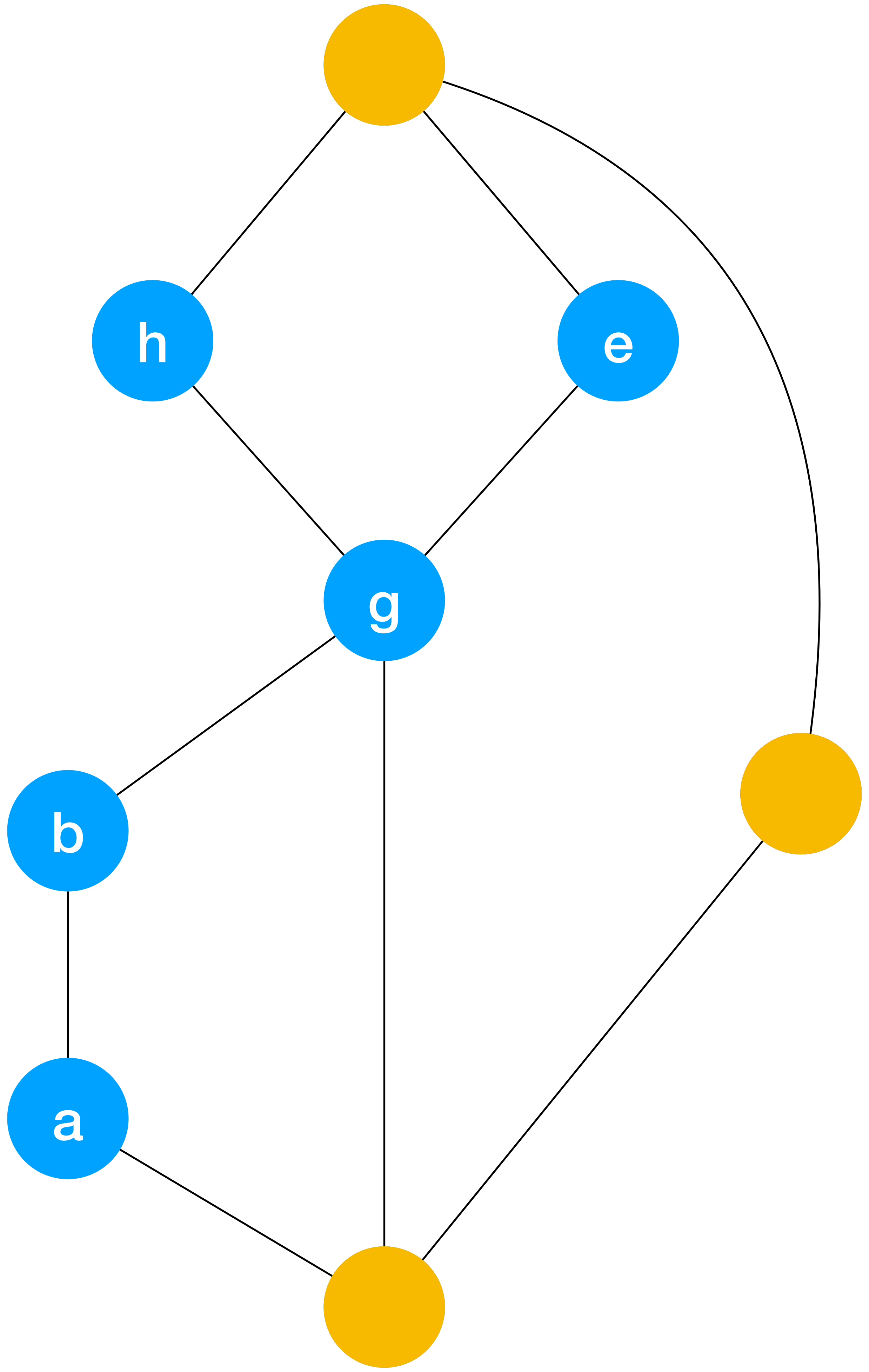


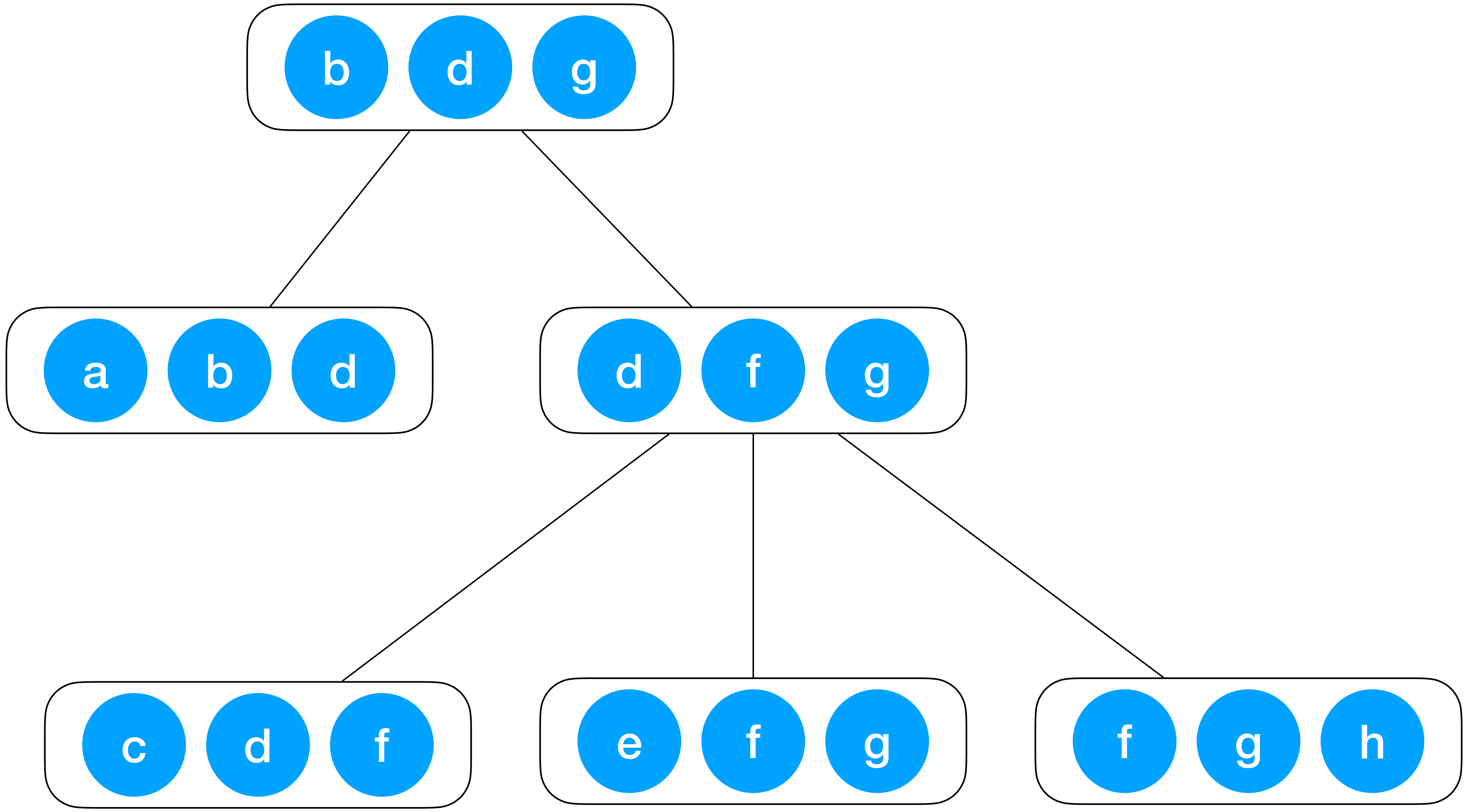
Example



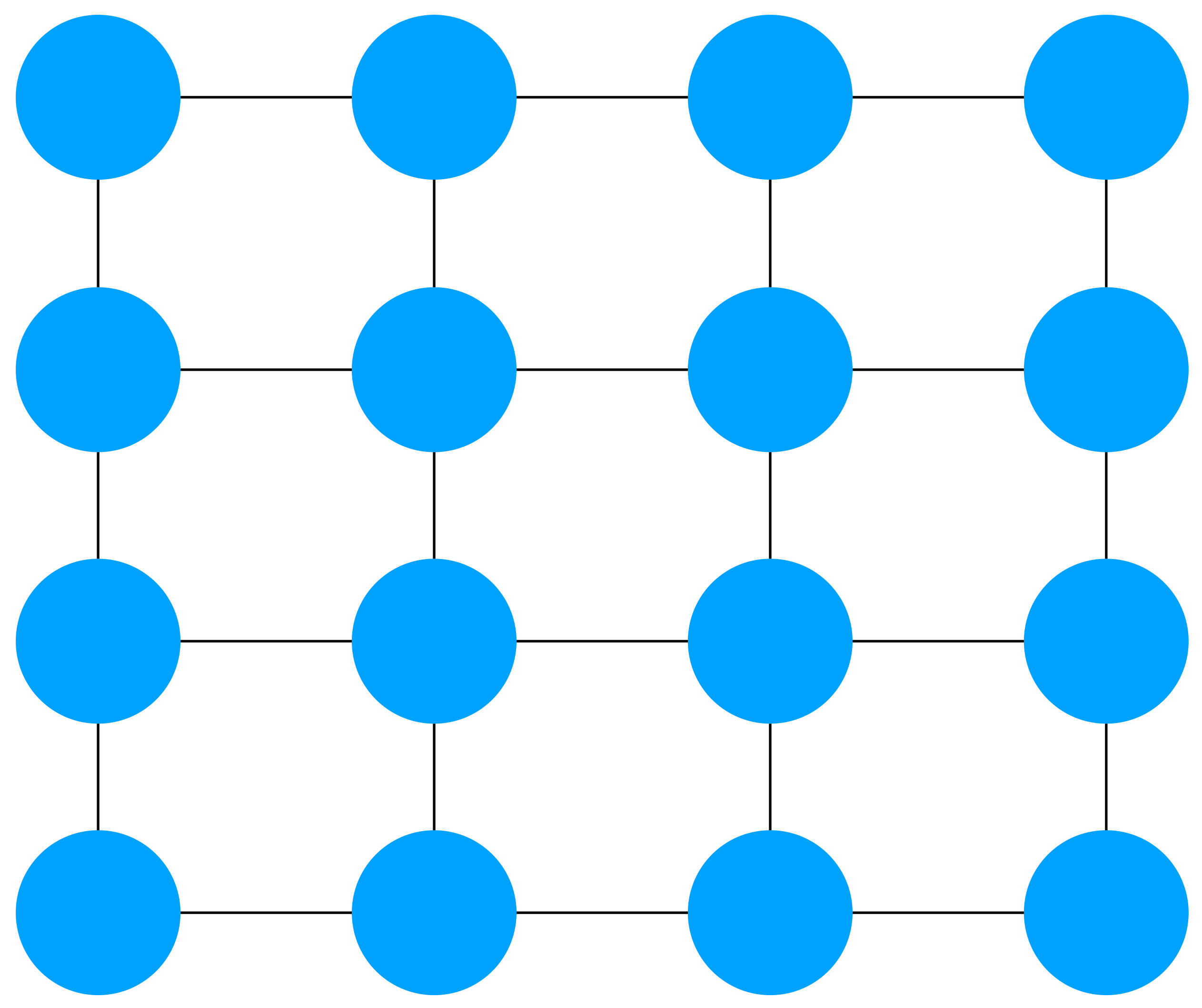


Example

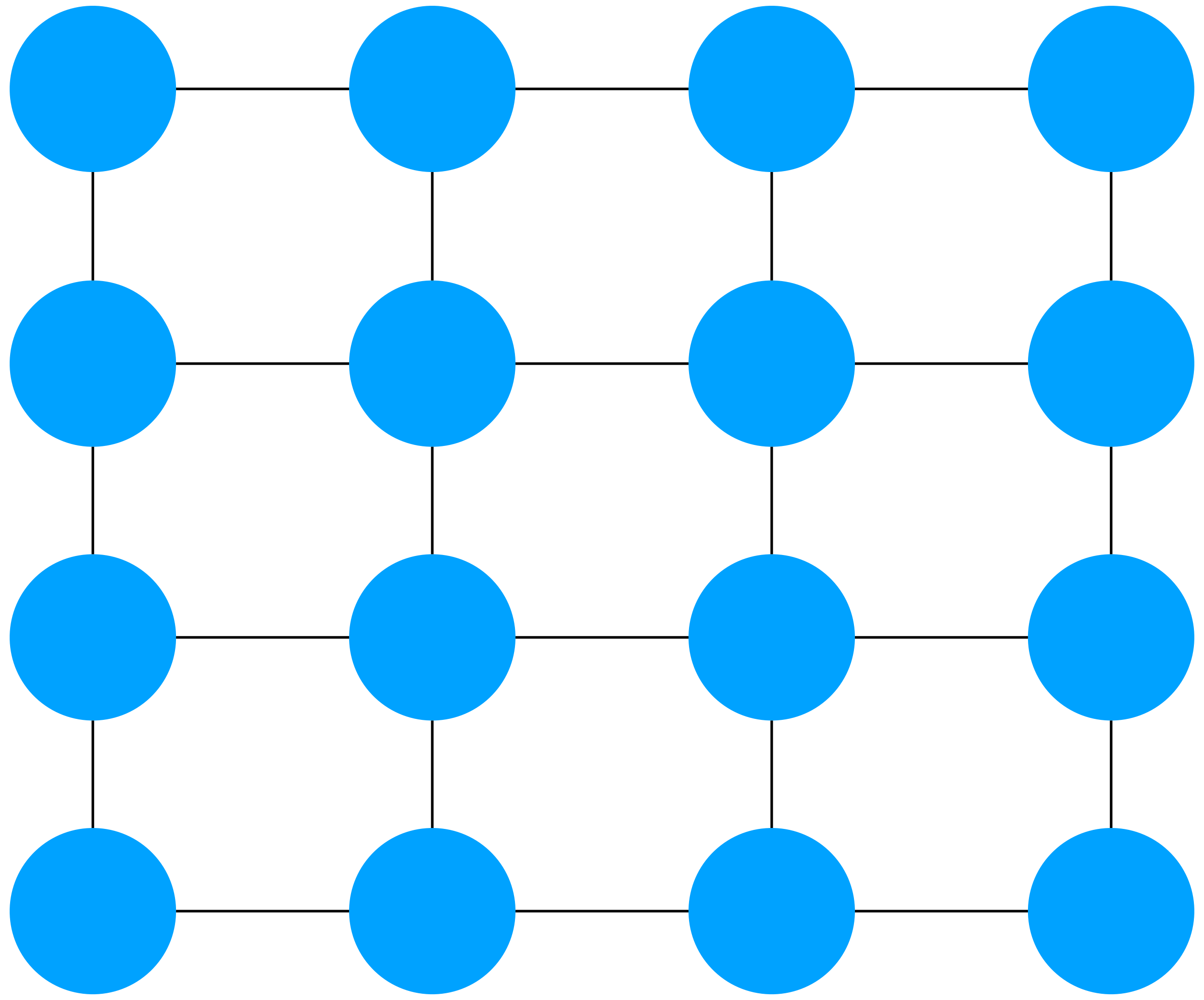




Grids have large Treewidth

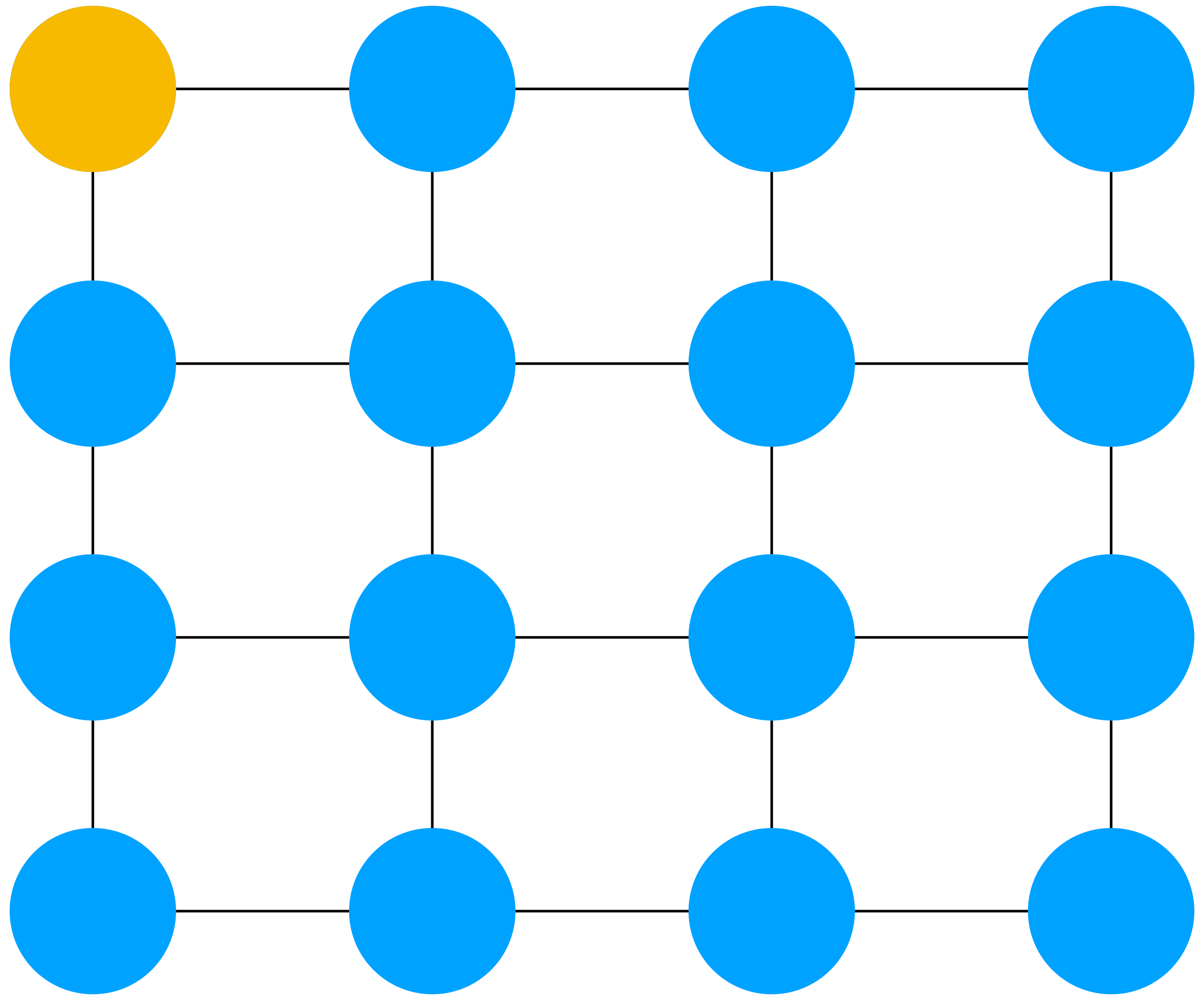


Grids have large Treewidth



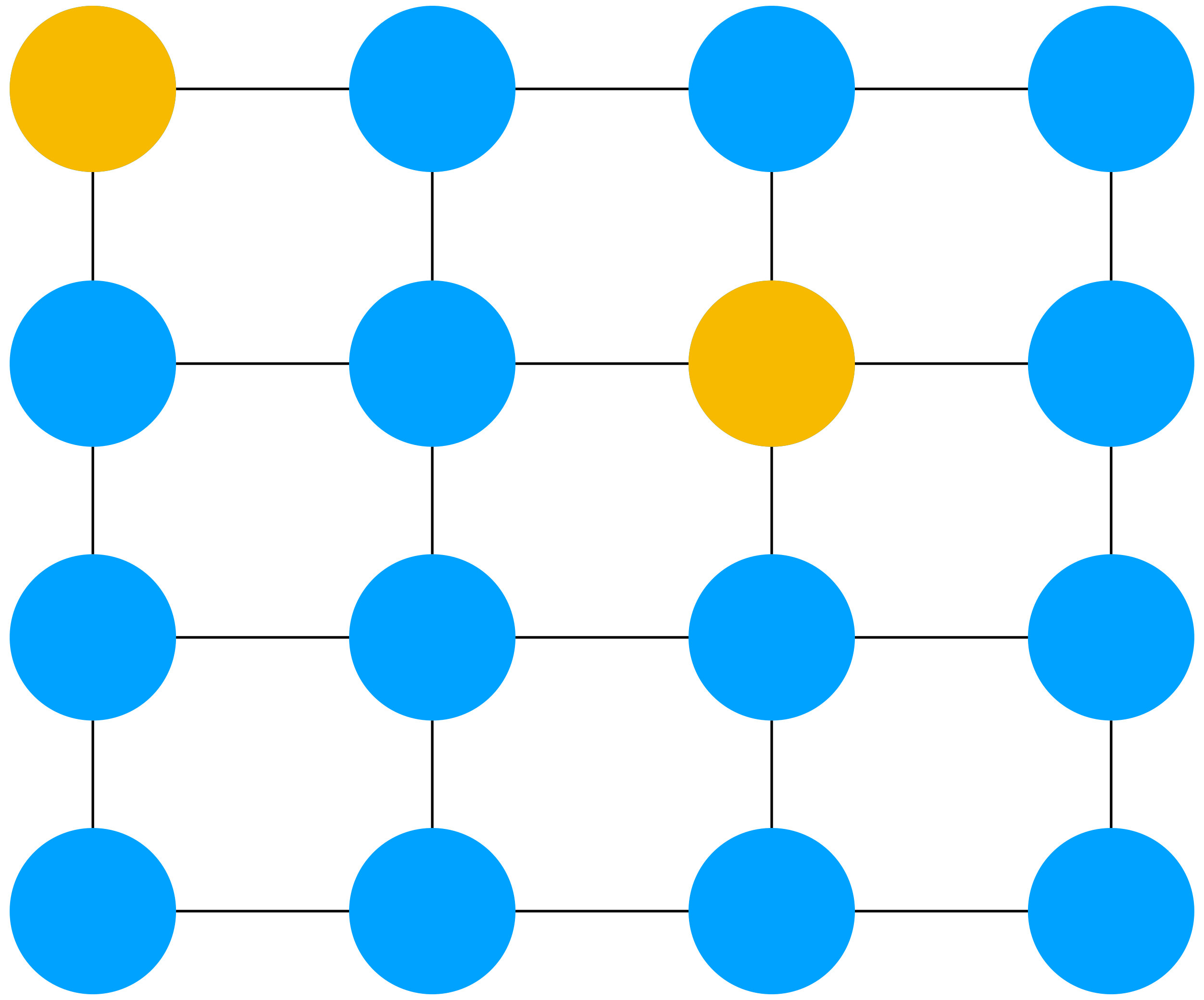
$n-1$ cops cannot win on Q_n

Grids have large Treewidth



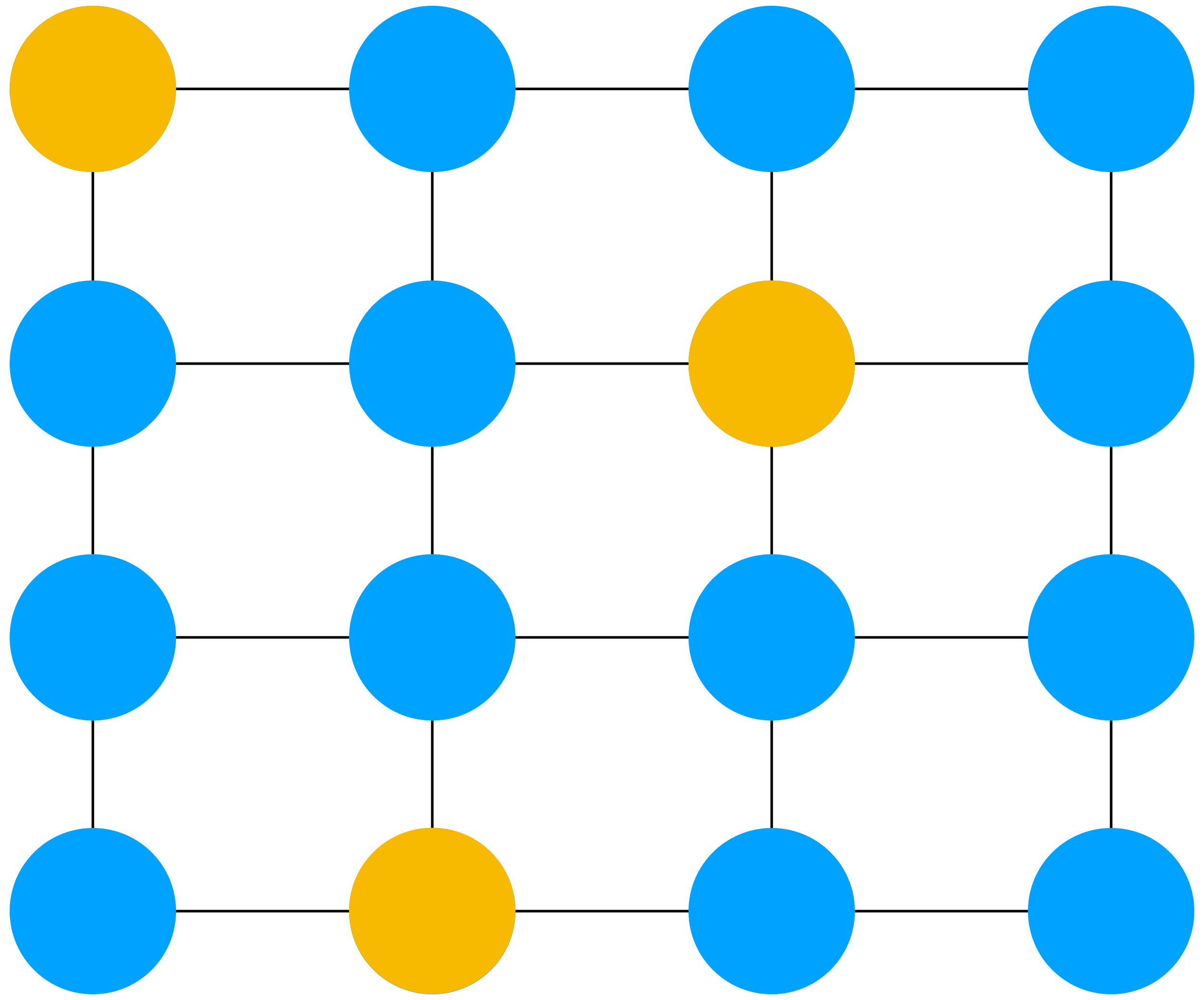
$n - 1$ cops cannot win on Q_n

Grids have large Treewidth



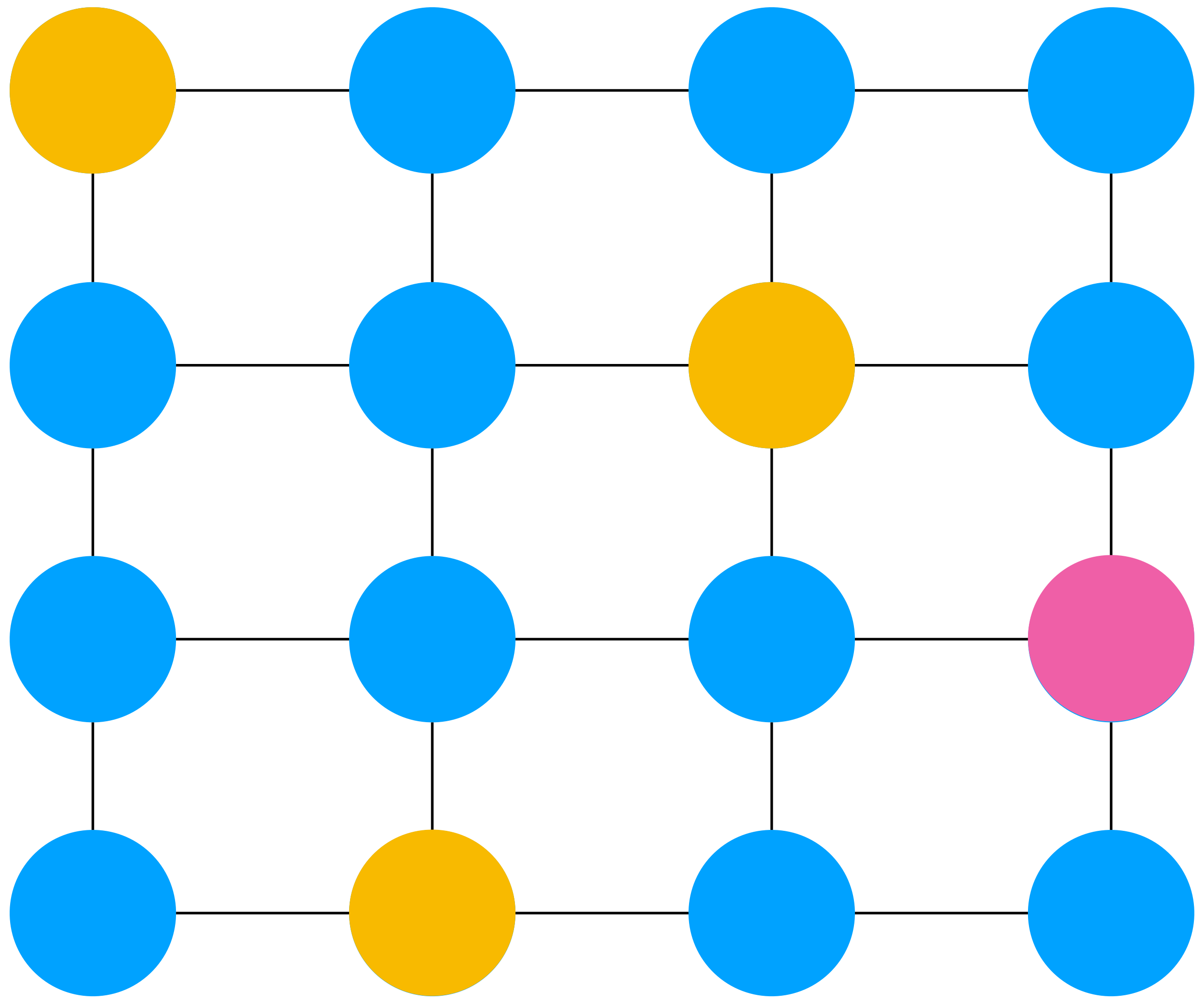
$n - 1$ cops cannot win on Q_n

Grids have large Treewidth



$n - 1$ cops cannot win on Q_n

Grids have large Treewidth



$n - 1$ cops cannot win on Q_n

In fact, the treewidth of Q_n is n.

Application: Variable Elimination for SAT

Propositional Satisfiability

Propositional Satisfiability

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

Propositional Satisfiability

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

SAT

Input: A CNF formula F.

Question: Does F have a satisfying assignment?

Propositional Satisfiability

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

$$x_1 \qquad x_2 \qquad x_3$$

SAT

Input: A CNF formula F.

Question: Does F have a satisfying assignment?

Propositional Satisfiability

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

x_1 x_2 x_3

SAT

Input: A CNF formula F.

Question: Does F have a satisfying assignment?

The Resolution Rule

$$\frac{C_1 \vee x \qquad\qquad \neg x \vee C_2}{C_1 \vee C_2}$$

The Resolution Rule

“pivot”

$$\frac{C_1 \lor x \qquad \neg x \lor C_2}{C_1 \lor C_2}$$

The Resolution Rule

“pivot”

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2}$$

“resolvent”

The Resolution Rule

"pivot"

$$\frac{C_1 \lor x \qquad \neg x \lor C_2}{C_1 \lor C_2}$$

"resolvent"

$C_1 \lor x$ $\quad \neg x \lor C_2$ $\quad$ both satisfied

The Resolution Rule

"pivot"

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2}$$

"resolvent"

$C_1 \vee x$ $\neg x \vee C_2$ both satisfied

x **false**

The Resolution Rule

“pivot”

$$\frac{C_1 \lor x \qquad \neg x \lor C_2}{C_1 \lor C_2}$$

“resolvent”

$C_1 \lor x$ $\qquad \neg x \lor C_2$ $\qquad$ both satisfied

x **false** ➡

The Resolution Rule

"pivot"

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2}$$

"resolvent"

$C_1 \vee x$ $\quad$ $\neg x \vee C_2$ $\quad$ both satisfied

x **false** ➡ C_1 satisfied

The Resolution Rule

“pivot”

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2}$$

“resolvent”

$C_1 \vee x$ $\neg x \vee C_2$ both satisfied

x **false** → C_1 satisfied →

The Resolution Rule

"pivot"

$$\frac{C_1 \lor x \qquad \neg x \lor C_2}{C_1 \lor C_2}$$

"resolvent"

$C_1 \lor x$ $\neg x \lor C_2$ both satisfied

x **false** → C_1 satisfied → $C_1 \lor C_2$ satisfied

The Resolution Rule

"pivot"

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2}$$

"resolvent"

$C_1 \vee x$ $\neg x \vee C_2$ both satisfied

x **false** ➡ C_1 satisfied ➡ $C_1 \vee C_2$ satisfied

x **true**

The Resolution Rule

"pivot"

$$\frac{C_1 \lor x \qquad \neg x \lor C_2}{C_1 \lor C_2}$$

"resolvent"

$C_1 \lor x$ $\quad$ $\neg x \lor C_2$ $\quad$ both satisfied

x **false** → C_1 satisfied → $C_1 \lor C_2$ satisfied

x **true** →

The Resolution Rule

"pivot"

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2}$$

"resolvent"

$C_1 \vee x$ $\quad$ $\neg x \vee C_2$ $\quad$ both satisfied

x **false** → C_1 satisfied → $C_1 \vee C_2$ satisfied

x **true** → C_2 satisfied

The Resolution Rule

"pivot"

$$\frac{C_1 \lor x \qquad \neg x \lor C_2}{C_1 \lor C_2}$$

"resolvent"

$C_1 \lor x$ $\quad$ $\neg x \lor C_2$ $\quad$ both satisfied

x **false** → C_1 satisfied → $C_1 \lor C_2$ satisfied

x **true** → C_2 satisfied →

The Resolution Rule

"pivot"

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2}$$

"resolvent"

$C_1 \vee x$ $\quad \neg x \vee C_2$ $\quad$ both satisfied

x **false** → C_1 satisfied → $C_1 \vee C_2$ satisfied

x **true** → C_2 satisfied → $C_1 \vee C_2$ satisfied

The Resolution Rule

"pivot"

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2}$$

"resolvent"

Theorem

Resolution is sound.

$C_1 \vee x$ $\quad$ $\neg x \vee C_2$ $\quad$ both satisfied

x **false** → C_1 satisfied → $C_1 \vee C_2$ satisfied

x **true** → C_2 satisfied → $C_1 \vee C_2$ satisfied

The Resolution Rule

"pivot"

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2}$$

"resolvent"

Theorem

Resolution is sound.

Adding resolvents does not make a formula unsatisfiable.

$C_1 \vee x$ $\quad$ $\neg x \vee C_2$ $\quad$ both satisfied

x **false** → C_1 satisfied → $C_1 \vee C_2$ satisfied

x **true** → C_2 satisfied → $C_1 \vee C_2$ satisfied

Examples

Examples

$(x_1 \vee x_2 \vee \neg x_3) \quad (\neg x_1 \vee x_2)$

Examples

$$(x_1 \vee x_2 \vee \neg x_3) \quad (\neg x_1 \vee x_2)$$

Examples

$$\frac{(x_1 \lor x_2 \lor \neg x_3) \quad (\neg x_1 \lor x_2)}{(x_2 \lor \neg x_3)}$$

Examples

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (\neg x_1 \vee x_2)}{(x_2 \vee \neg x_3)}$$

$$(x_1 \vee x_2 \vee \neg x_3) \quad (x_3 \vee \neg x_2)$$

Examples

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (\neg x_1 \vee x_2)}{(x_2 \vee \neg x_3)}$$

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (x_3 \vee \neg x_2)}{}$$

Examples

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (\neg x_1 \vee x_2)}{(x_2 \vee \neg x_3)}$$

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (x_3 \vee \neg x_2)}{(x_1 \vee x_2 \vee \neg x_2)}$$

Examples

$$\frac{(x_1 \lor x_2 \lor \neg x_3) \quad (\neg x_1 \lor x_2)}{(x_2 \lor \neg x_3)}$$

$$\frac{(x_1 \lor x_2 \lor \neg x_3) \quad (x_3 \lor \neg x_2)}{(x_1 \lor x_2 \lor \neg x_2)}$$

tautology (always satisfied)

Examples

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (\neg x_1 \vee x_2)}{(x_2 \vee \neg x_3)}$$

$$(x) \qquad (\neg x)$$

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (x_3 \vee \neg x_2)}{(x_1 \vee x_2 \vee \neg x_2)}$$

tautology (always satisfied)

Examples

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (\neg x_1 \vee x_2)}{(x_2 \vee \neg x_3)}$$

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (x_3 \vee \neg x_2)}{(x_1 \vee x_2 \vee \neg x_2)}$$

tautology (always satisfied)

$$\frac{(x) \quad (\neg x)}{}$$

Examples

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (\neg x_1 \vee x_2)}{(x_2 \vee \neg x_3)}$$

$$\frac{(x) \quad (\neg x)}{()}$$

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (x_3 \vee \neg x_2)}{(x_1 \vee x_2 \vee \neg x_2)}$$

tautology (always satisfied)

Examples

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (\neg x_1 \vee x_2)}{(x_2 \vee \neg x_3)}$$

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (x_3 \vee \neg x_2)}{(x_1 \vee x_2 \vee \neg x_2)}$$

tautology (always satisfied)

$$\frac{(x) \quad (\neg x)}{()}$$

"empty clause"

Examples

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (\neg x_1 \vee x_2)}{(x_2 \vee \neg x_3)}$$

$$\frac{(x_1 \vee x_2 \vee \neg x_3) \quad (x_3 \vee \neg x_2)}{(x_1 \vee x_2 \vee \neg x_2)}$$

tautology (always satisfied)

$$\frac{(x) \quad (\neg x)}{()}$$

"empty clause"

Observation

The empty clause cannot be satisfied.

Refutations

Observation

The empty clause cannot be satisfied.

Theorem

Resolution is sound.

Refutations

Observation

The empty clause cannot be satisfied.

Theorem

Resolution is sound.

$$(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3) \quad (\neg x_3) \quad (\neg x_2)$$

Refutations

Observation

The empty clause cannot be satisfied.

Theorem

Resolution is sound.

$$(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3) \quad (\neg x_3) \quad (\neg x_2)$$

$$(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3)$$

Refutations

Observation

The empty clause cannot be satisfied.

Theorem

Resolution is sound.

$$(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3) \quad (\neg x_3) \quad (\neg x_2)$$

$$\frac{(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3)}{(x_2 \vee x_3)}$$

Refutations

Observation

The empty clause cannot be satisfied.

Theorem

Resolution is sound.

$$(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3) \quad (\neg x_3) \quad (\neg x_2)$$

$$\frac{(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3)}{(x_2 \vee x_3)} \qquad (\neg x_3)$$

Refutations

Observation

The empty clause cannot be satisfied.

Theorem

Resolution is sound.

$$(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3) \quad (\neg x_3) \quad (\neg x_2)$$

$$\cfrac{\cfrac{(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3)}{(x_2 \vee x_3)} \quad (\neg x_3)}{(x_2)}$$

Refutations

Observation

The empty clause cannot be satisfied.

Theorem

Resolution is sound.

$$(x_1 \lor x_2) \quad (\neg x_1 \lor x_2 \lor x_3) \quad (\neg x_3) \quad (\neg x_2)$$

$$\cfrac{\cfrac{(x_1 \lor x_2) \qquad (\neg x_1 \lor x_2 \lor x_3)}{(x_2 \lor x_3)} \qquad (\neg x_3)}{(x_2)} \qquad (\neg x_2)$$

Refutations

Observation

The empty clause cannot be satisfied.

Theorem

Resolution is sound.

$$(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3) \quad (\neg x_3) \quad (\neg x_2)$$

$$\cfrac{\cfrac{\cfrac{(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3)}{(x_2 \vee x_3)} \quad (\neg x_3)}{(x_2)} \quad (\neg x_2)}{()}$$

Refutations

Observation

The empty clause cannot be satisfied.

Theorem

Resolution is sound.

$$(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3) \quad (\neg x_3) \quad (\neg x_2)$$

$$\cfrac{\cfrac{\cfrac{(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3)}{(x_2 \vee x_3)} \quad (\neg x_3)}{(x_2)} \quad (\neg x_2)}{()}$$

"refutation"

Refutations

Observation

The empty clause cannot be satisfied.

Theorem

Resolution is sound.

$$(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3) \quad (\neg x_3) \quad (\neg x_2)$$

$$\dfrac{\dfrac{\dfrac{(x_1 \vee x_2) \quad (\neg x_1 \vee x_2 \vee x_3)}{(x_2 \vee x_3)} \quad (\neg x_3)}{(x_2)} \quad (\neg x_2)}{()}$$

"refutation"

Corollary

If a formula has a refutation it is unsatisfiable.

Completeness

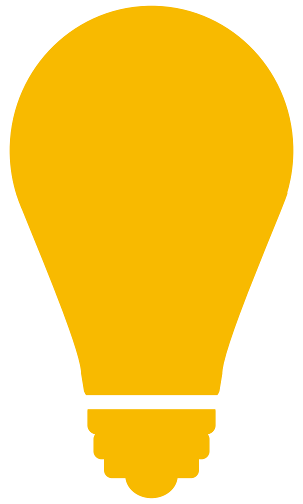
Theorem

Every unsatisfiable formula has a refutation.

Completeness

Theorem

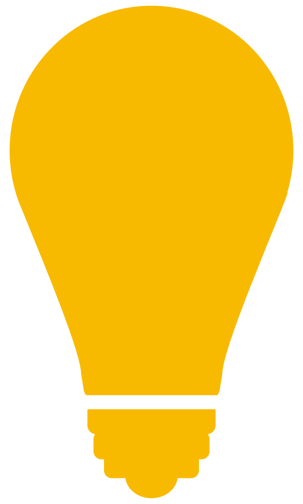
Every unsatisfiable formula has a refutation.



Completeness

Theorem

Every unsatisfiable formula has a refutation.



Algorithm for SAT: decide if there is a refutation.

Davis-Putnam Resolution

Davis-Putnam Resolution

Davis & Putnam 1960

Davis-Putnam Resolution

Davis & Putnam 1960

Input: A CNF formula F with m clauses

Davis-Putnam Resolution

Davis & Putnam 1960

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

Davis-Putnam Resolution

Davis & Putnam 1960

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

for x_i **in** σ **:**

Davis-Putnam Resolution

Davis & Putnam 1960

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

for x_i **in** σ **:**

add all possible resolvents on pivot x_i to F

Davis-Putnam Resolution

Davis & Putnam 1960

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

for x_i **in** σ **:**

add all possible resolvents on pivot x_i to F

remove clauses containing x_i from F

Davis-Putnam Resolution

Davis & Putnam 1960

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

for x_i **in** σ **:**

  add all possible resolvents on pivot x_i to F

  remove clauses containing x_i from F

  remove tautologies from F

Davis-Putnam Resolution

Davis & Putnam 1960

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

for x_i **in** σ **:**

add all possible resolvents on pivot x_i to F

remove clauses containing x_i from F

remove tautologies from F

return **false** $\Leftrightarrow$ F contains the empty clause

Davis-Putnam Resolution

Davis & Putnam 1960

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

for x_i **in** σ **:**

add all possible resolvents on pivot x_i to F

remove clauses containing x_i from F

remove tautologies from F

return false $\Leftrightarrow$ F contains the empty clause

Davis-Putnam Resolution

Davis & Putnam 1960

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

for x_i **in** σ **:**

- add all possible resolvents on pivot x_i to F
- remove clauses containing x_i from F
- remove tautologies from F

return false $\Leftrightarrow$ F contains the empty clause

"variable elimination"

Davis-Putnam Resolution

Davis & Putnam 1960

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

for x_i **in** σ **:**

add all possible resolvents on pivot x_i to F

remove clauses containing x_i from F

remove tautologies from F

return false $\Leftrightarrow$ F contains the empty clause

Worst case: m^2 resolvents in each iteration.

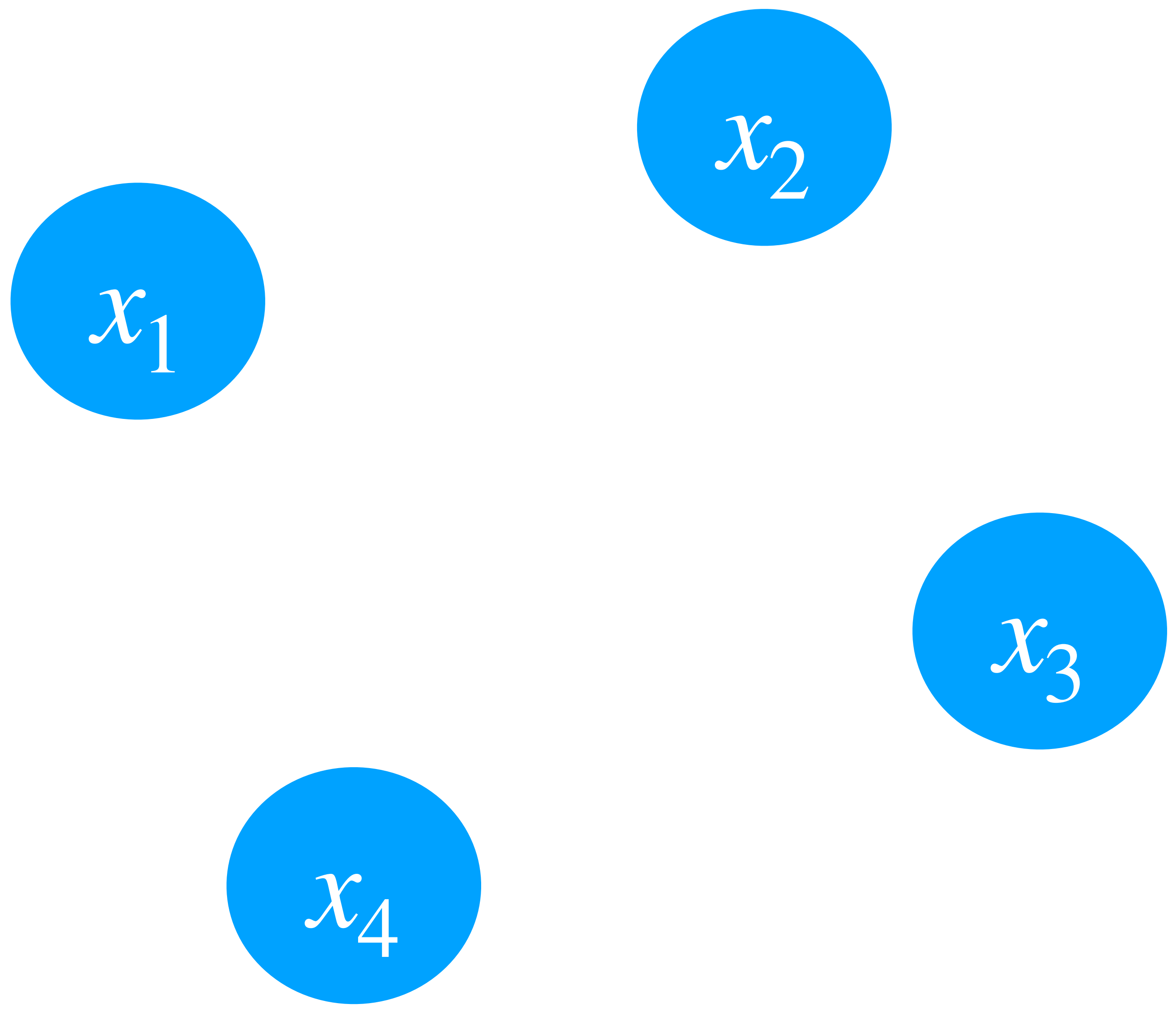
"variable elimination"

Primal Graph

$$(x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_2 \lor x_3) \land (x_3 \lor \neg x_2) \land (x_1 \lor x_2 \lor x_4)$$

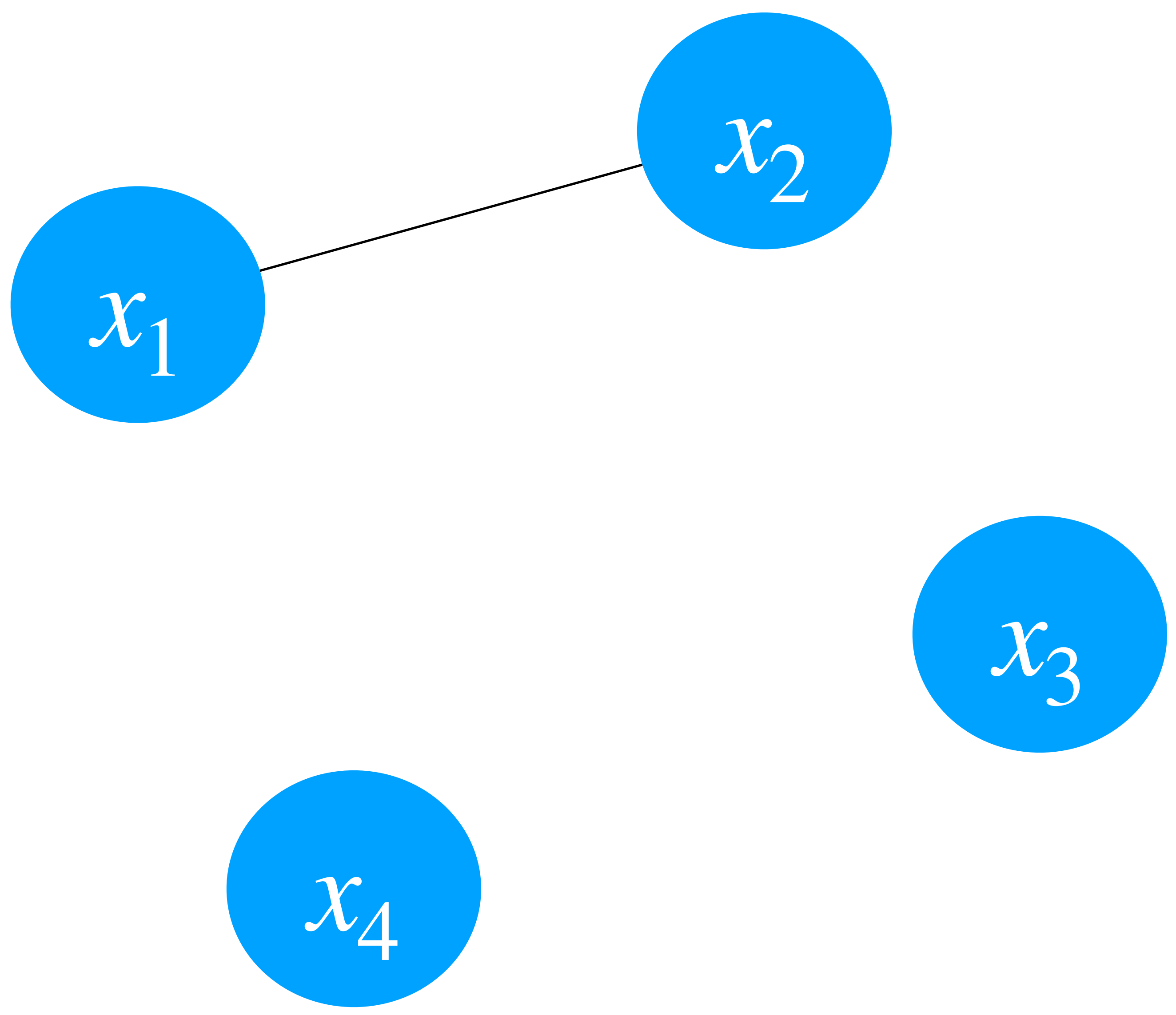
Primal Graph

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee x_4)$$



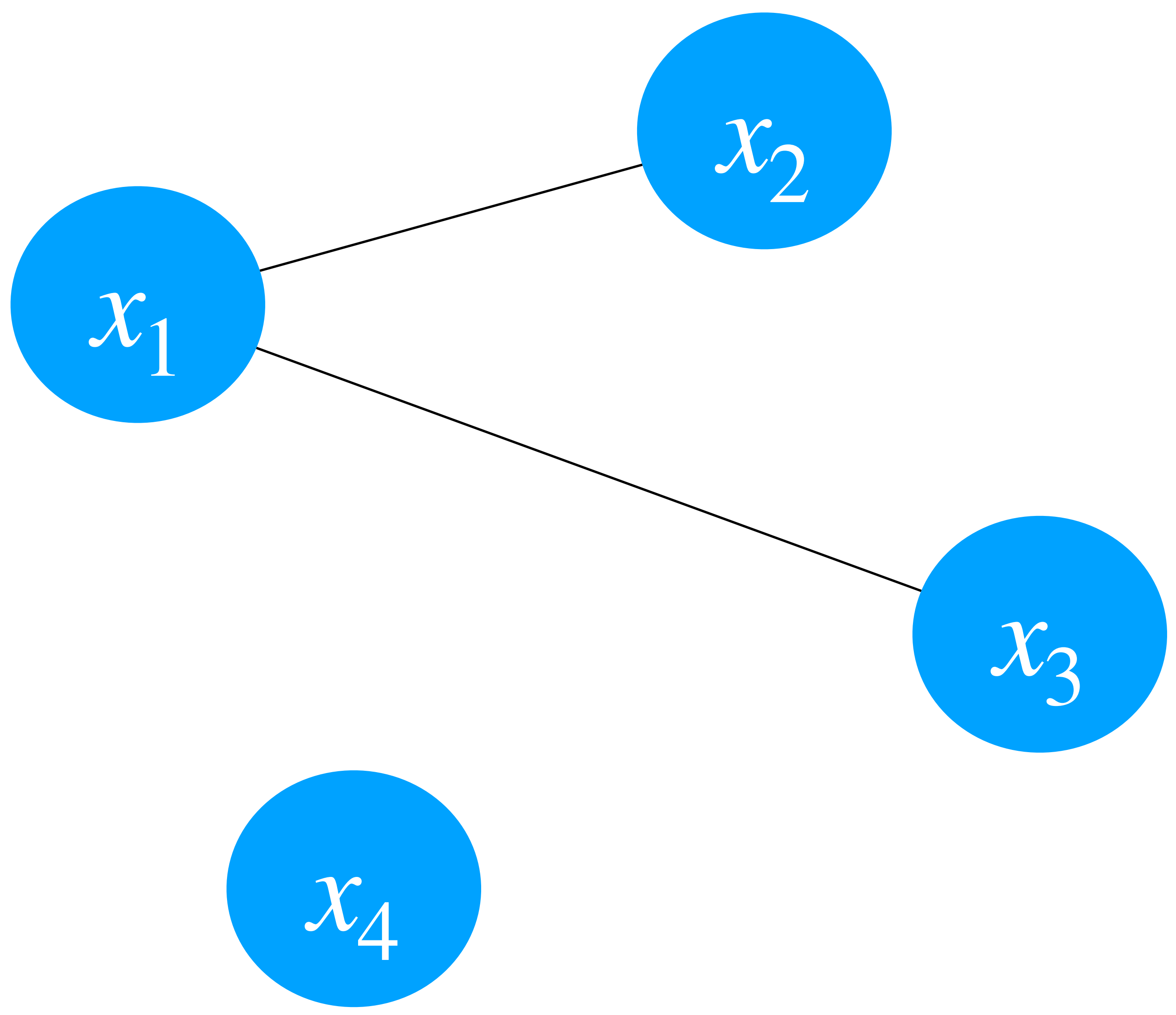
Primal Graph

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee x_4)$$



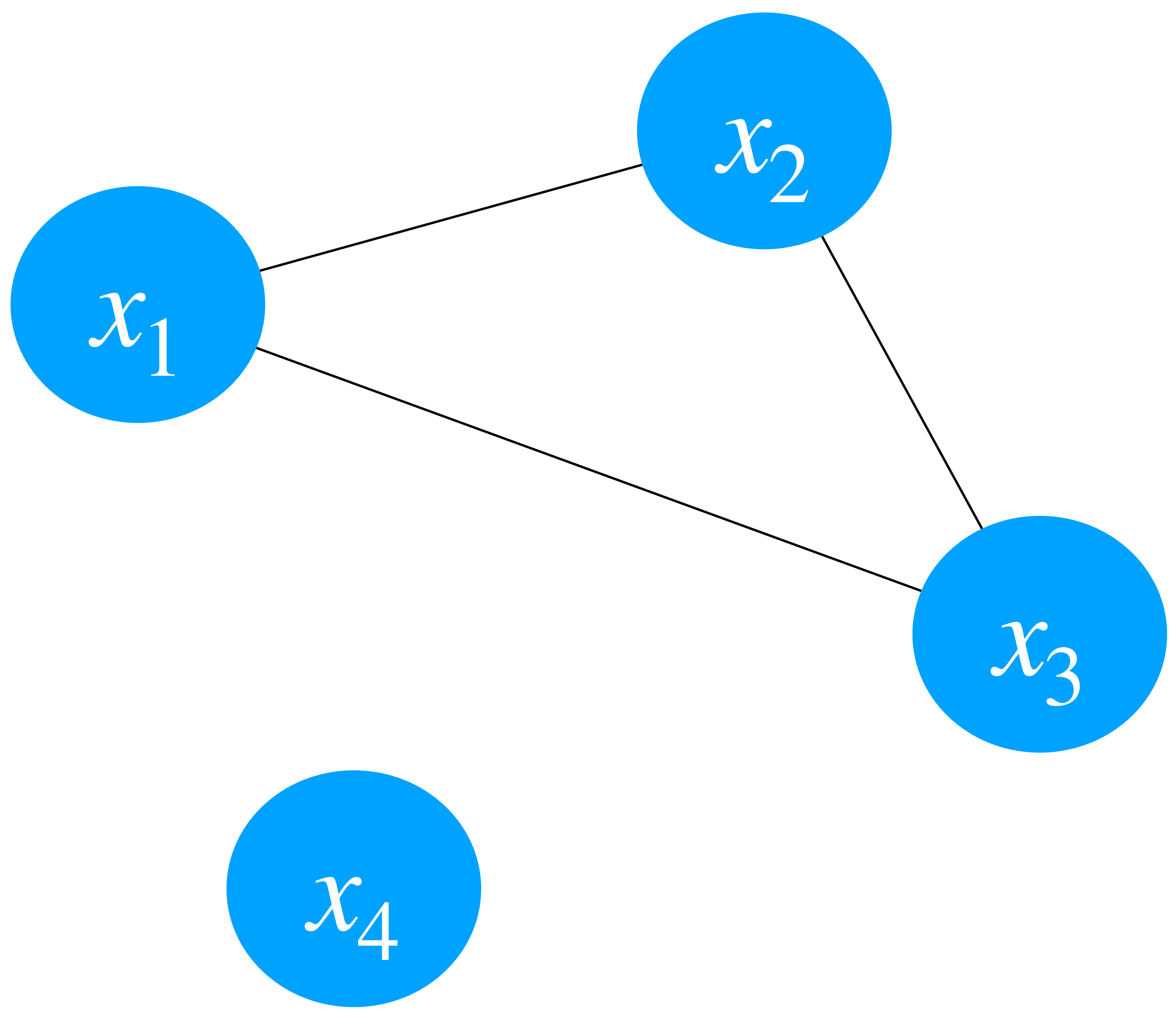
Primal Graph

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee x_4)$$



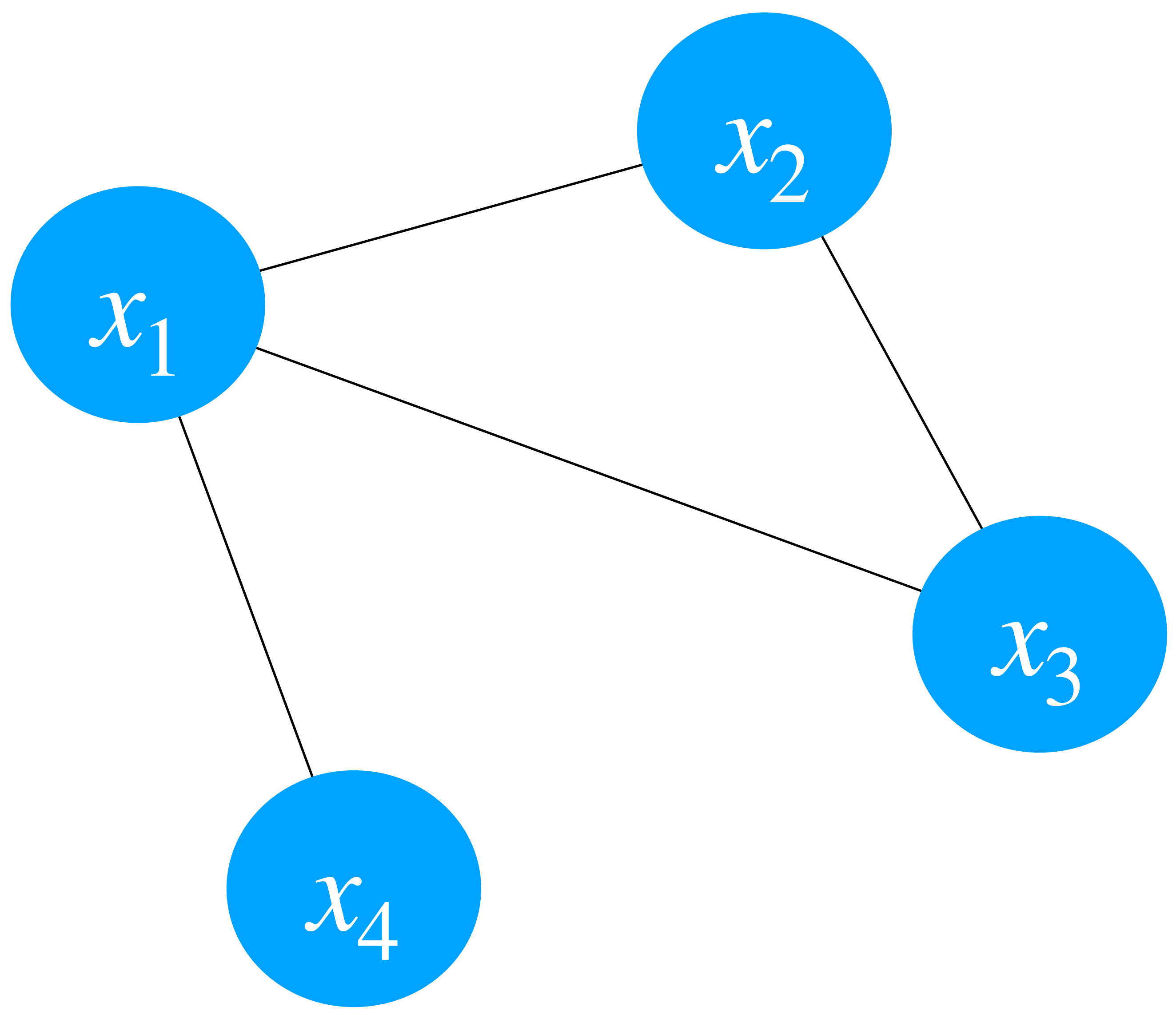
Primal Graph

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee x_4)$$



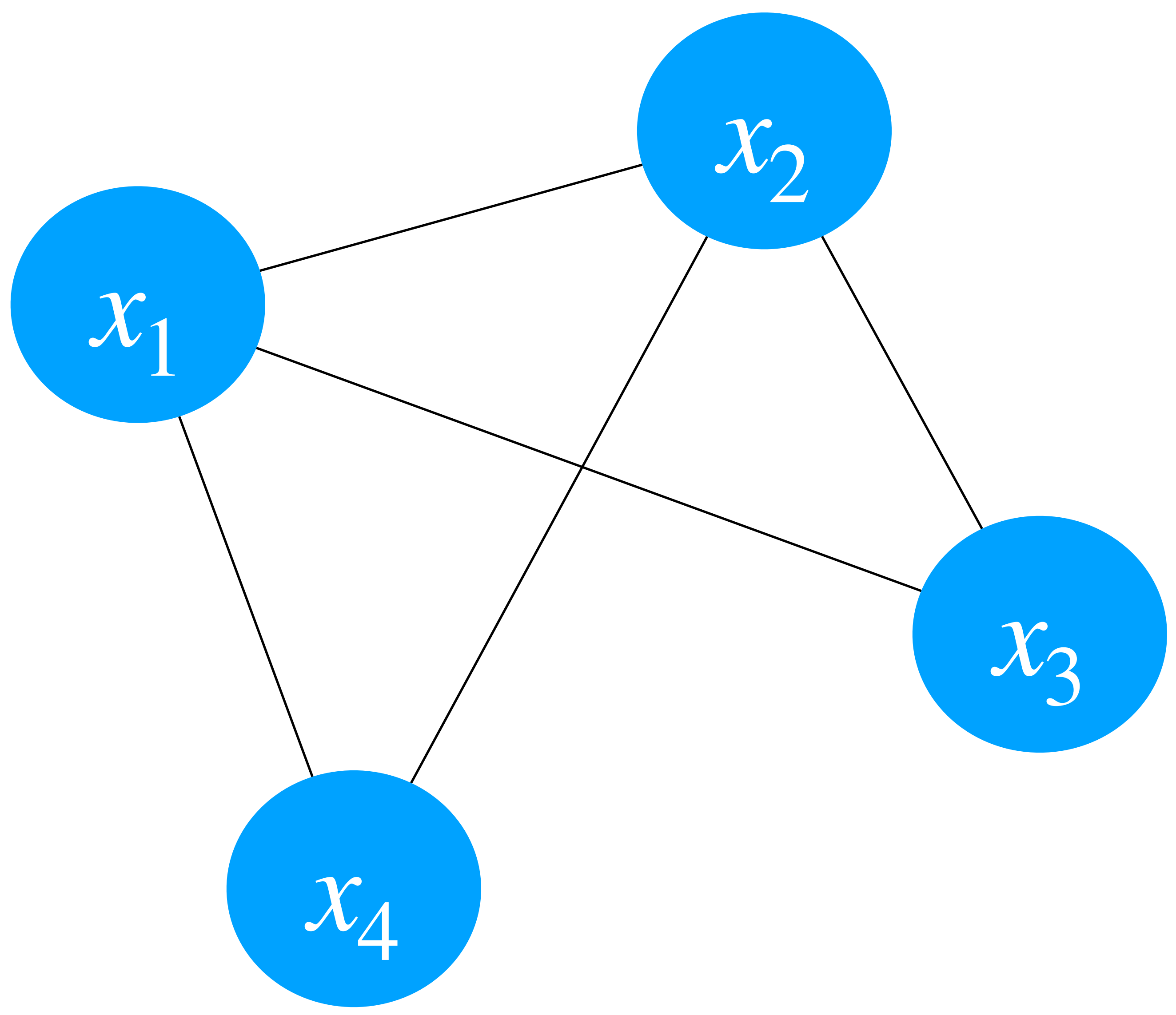
Primal Graph

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee x_4)$$



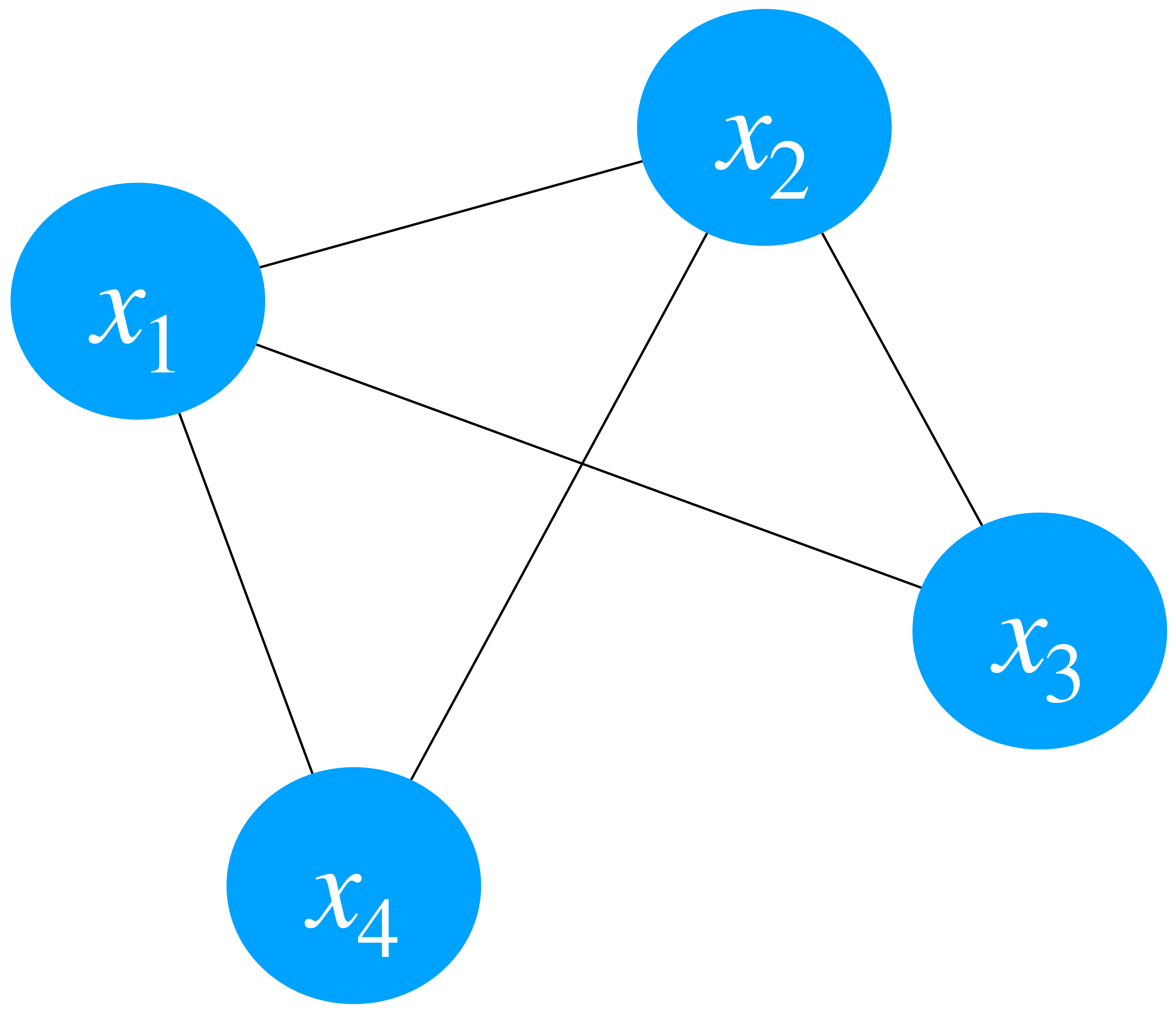
Primal Graph

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee x_4)$$



Primal Graph

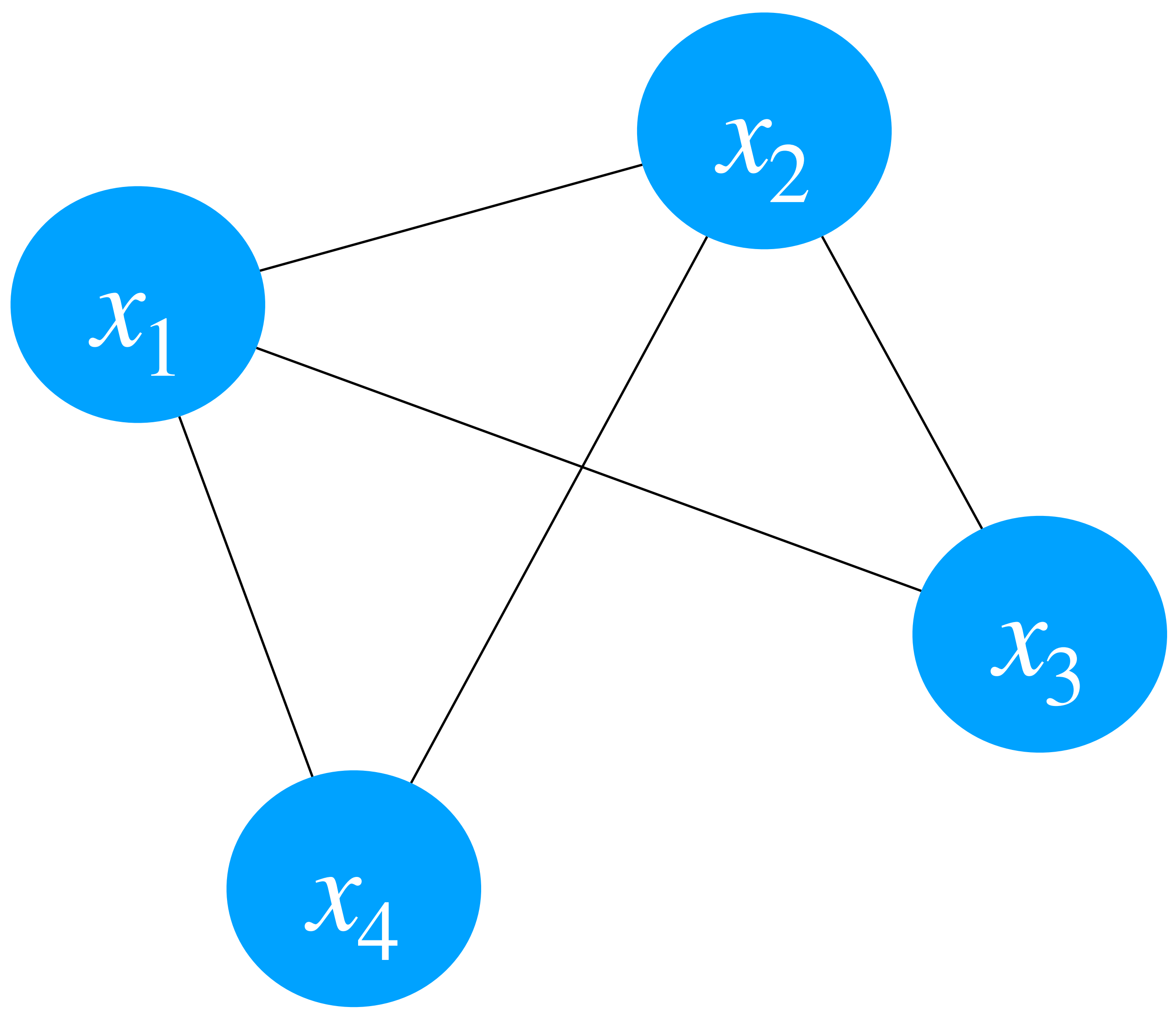
$$(x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_2 \lor x_3) \land (x_3 \lor \neg x_2) \land (x_1 \lor x_2 \lor x_4)$$



$$(x_3 \lor \neg x_2)$$

Primal Graph

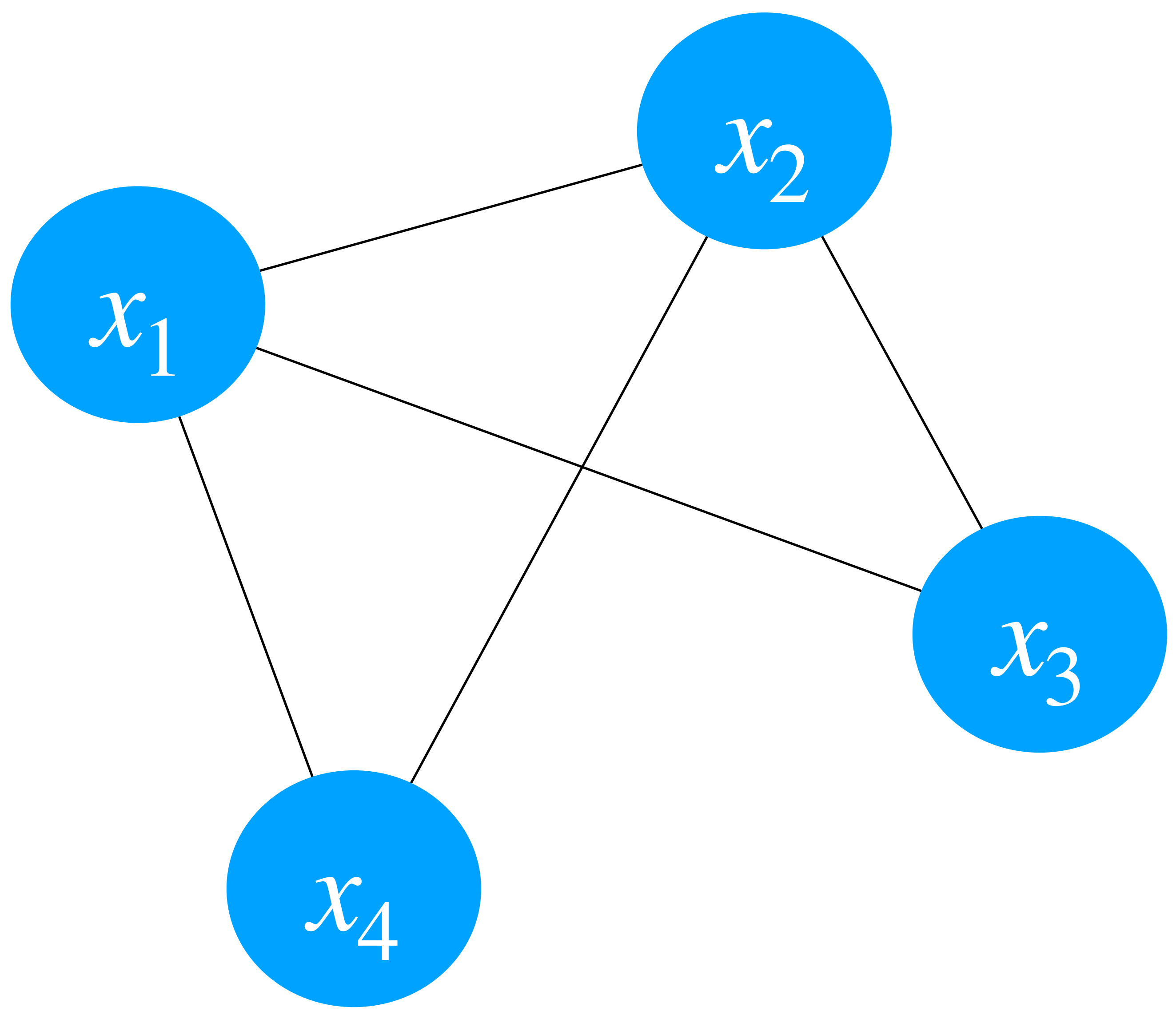
$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee x_4)$$



$(x_3 \vee \neg x_2)$ $\quad$ $(x_1 \vee x_2 \vee x_4)$

Primal Graph

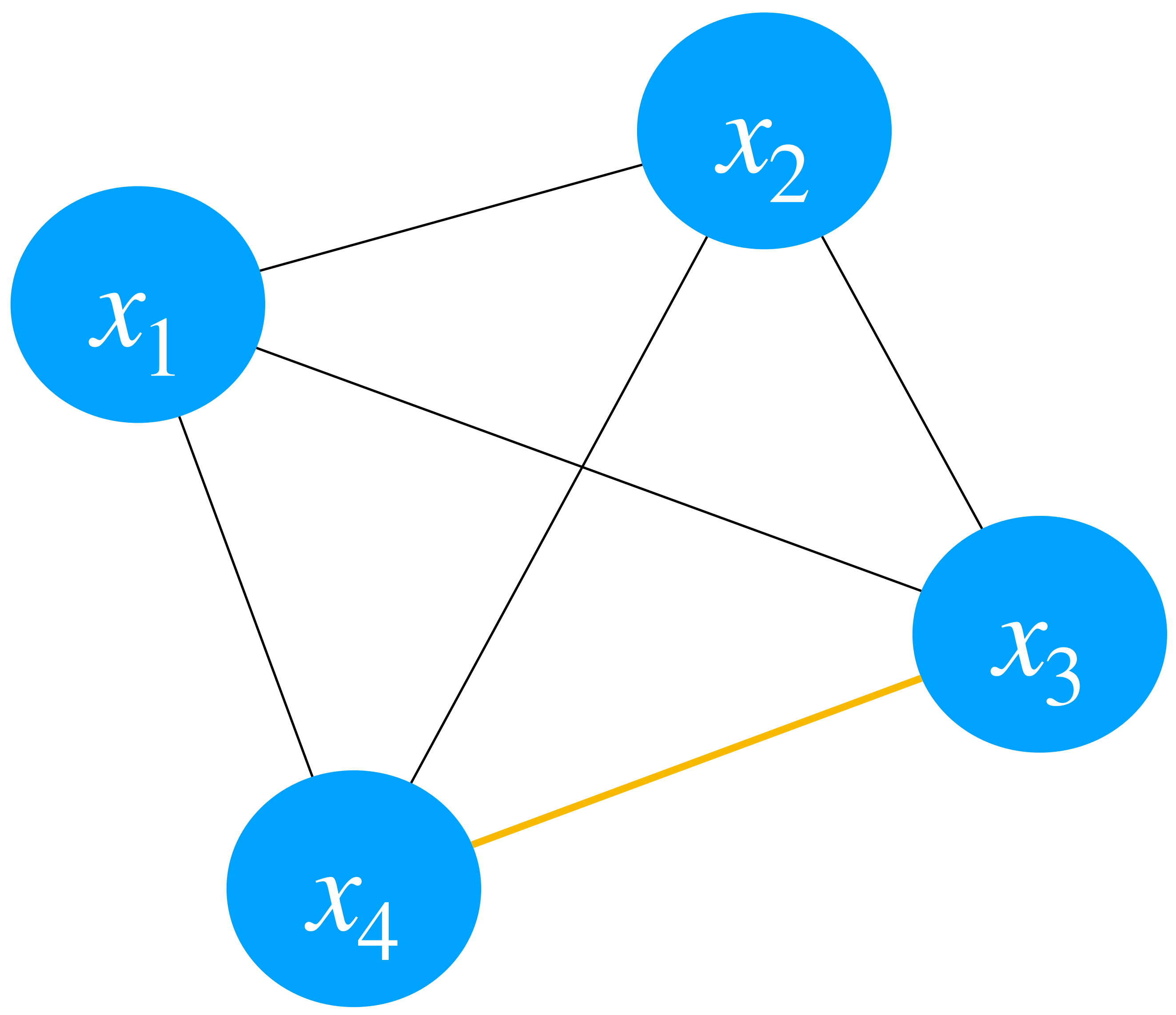
$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee x_4)$$



$$\frac{(x_3 \vee \neg x_2) \qquad (x_1 \vee x_2 \vee x_4)}{(x_1 \vee x_3 \vee x_4)}$$

Primal Graph

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee x_4)$$



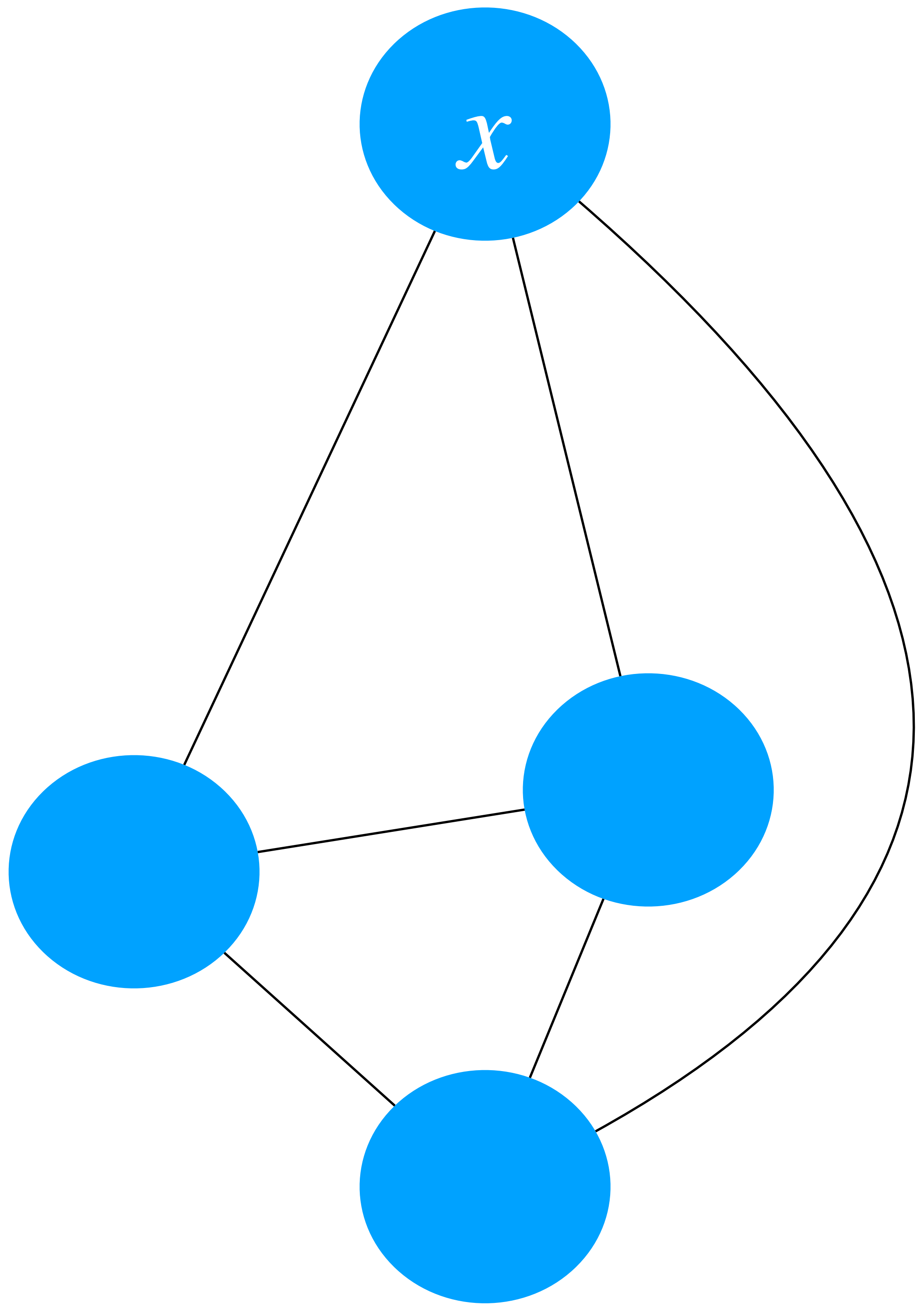
$$\frac{(x_3 \vee \neg x_2) \qquad (x_1 \vee x_2 \vee x_4)}{(x_1 \vee x_3 \vee x_4)}$$

Resolution in the Primal Graph

$$\frac{C_1 \lor x \qquad\qquad \neg x \lor C_2}{C_1 \lor C_2}$$

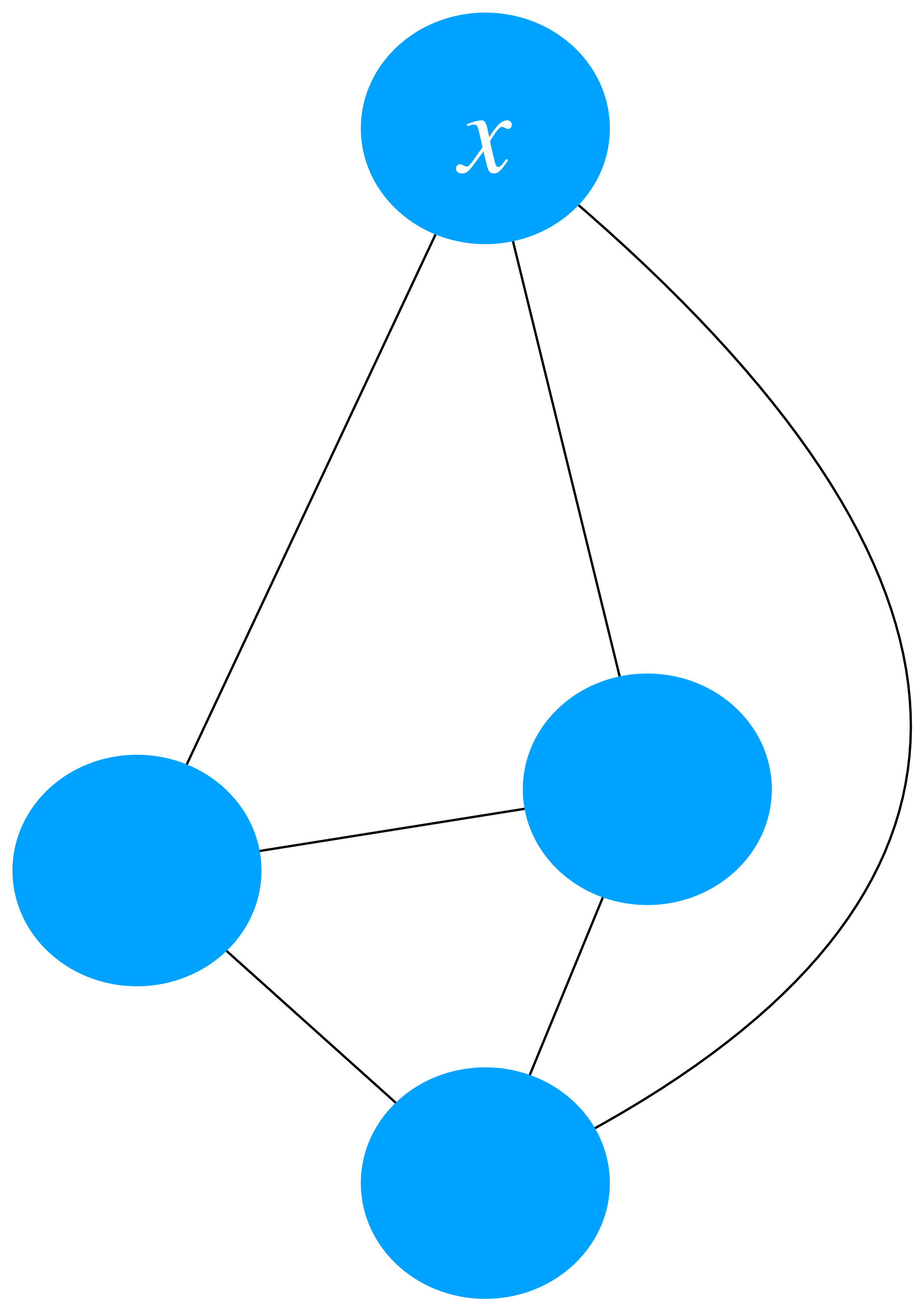
Resolution in the Primal Graph

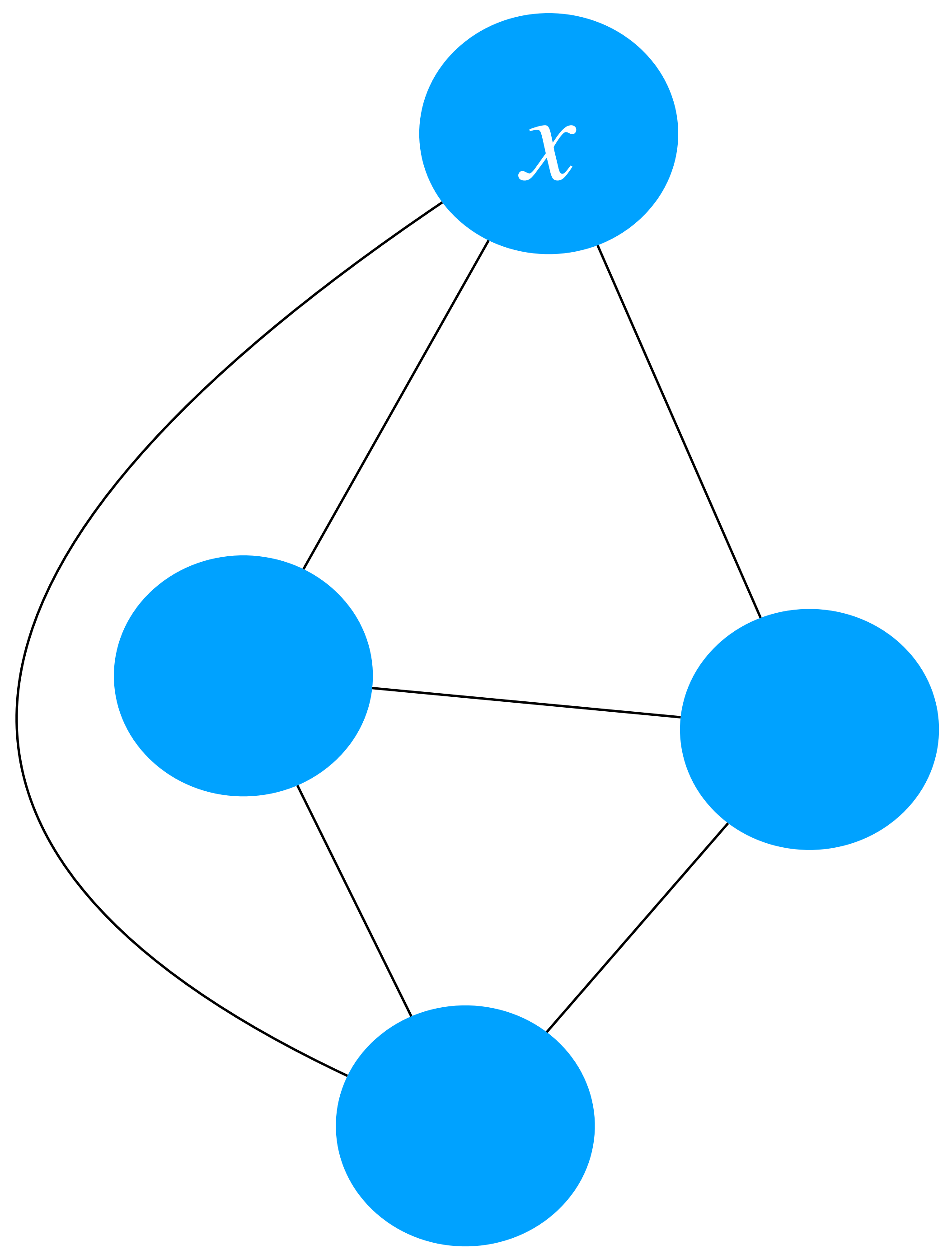
$$\frac{C_1 \vee x \qquad\qquad \neg x \vee C_2}{C_1 \vee C_2}$$



Resolution in the Primal Graph

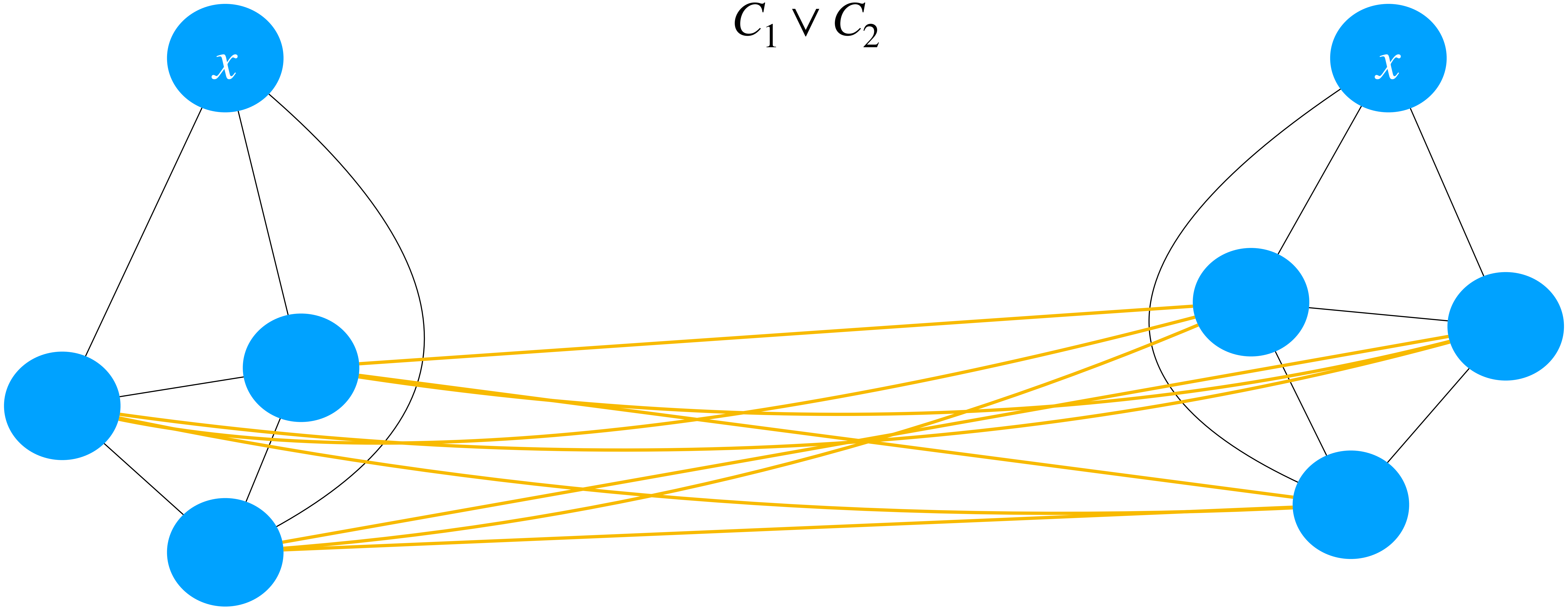
$$\frac{C_1 \lor x \qquad\qquad \neg x \lor C_2}{C_1 \lor C_2}$$





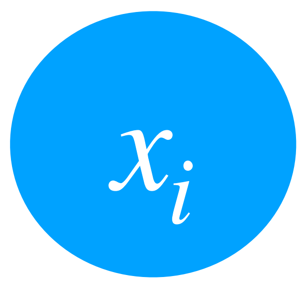
Resolution in the Primal Graph

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2}$$

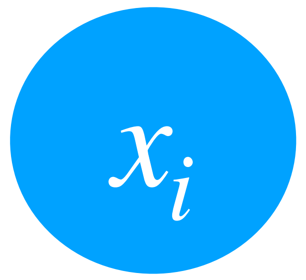


Variable Elimination in the Primal Graph

Variable Elimination in the Primal Graph

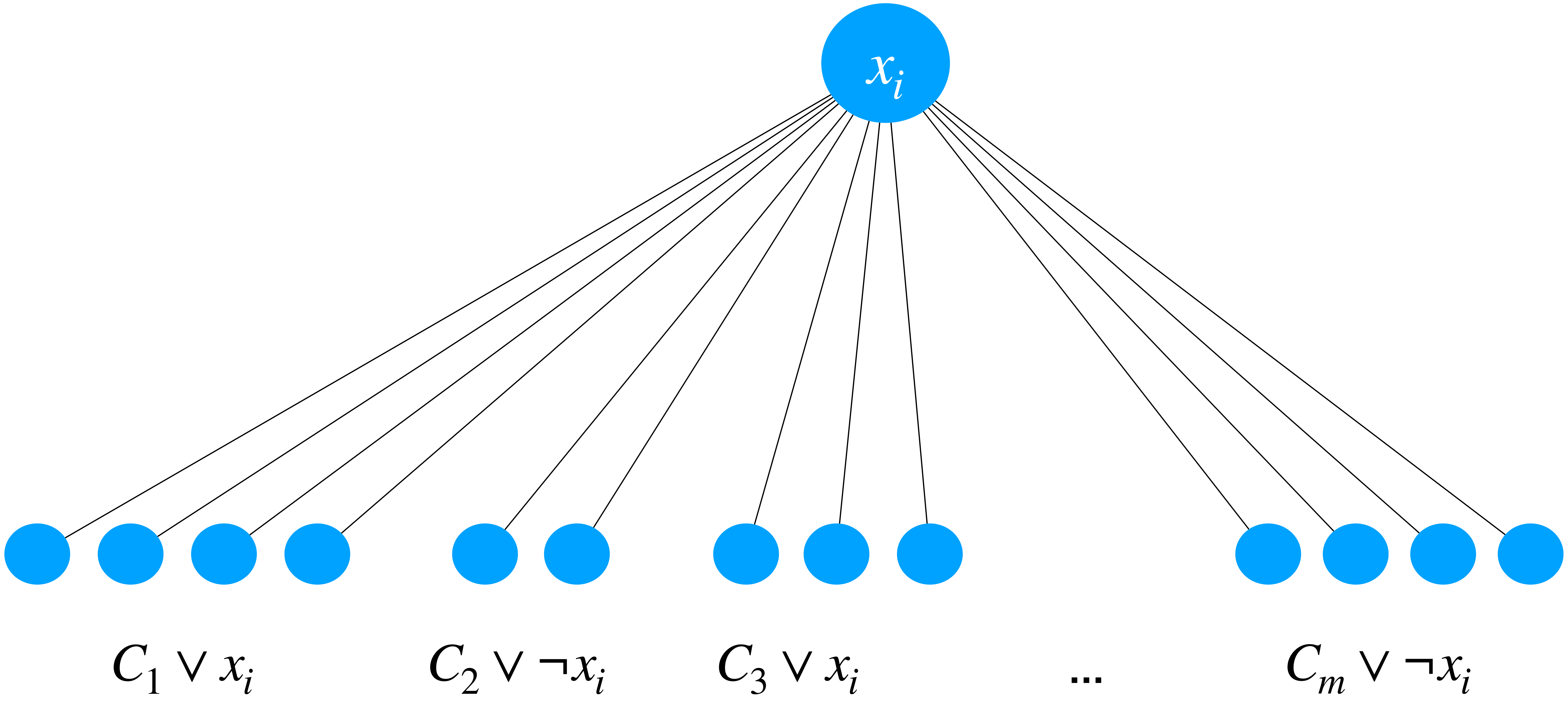


Variable Elimination in the Primal Graph



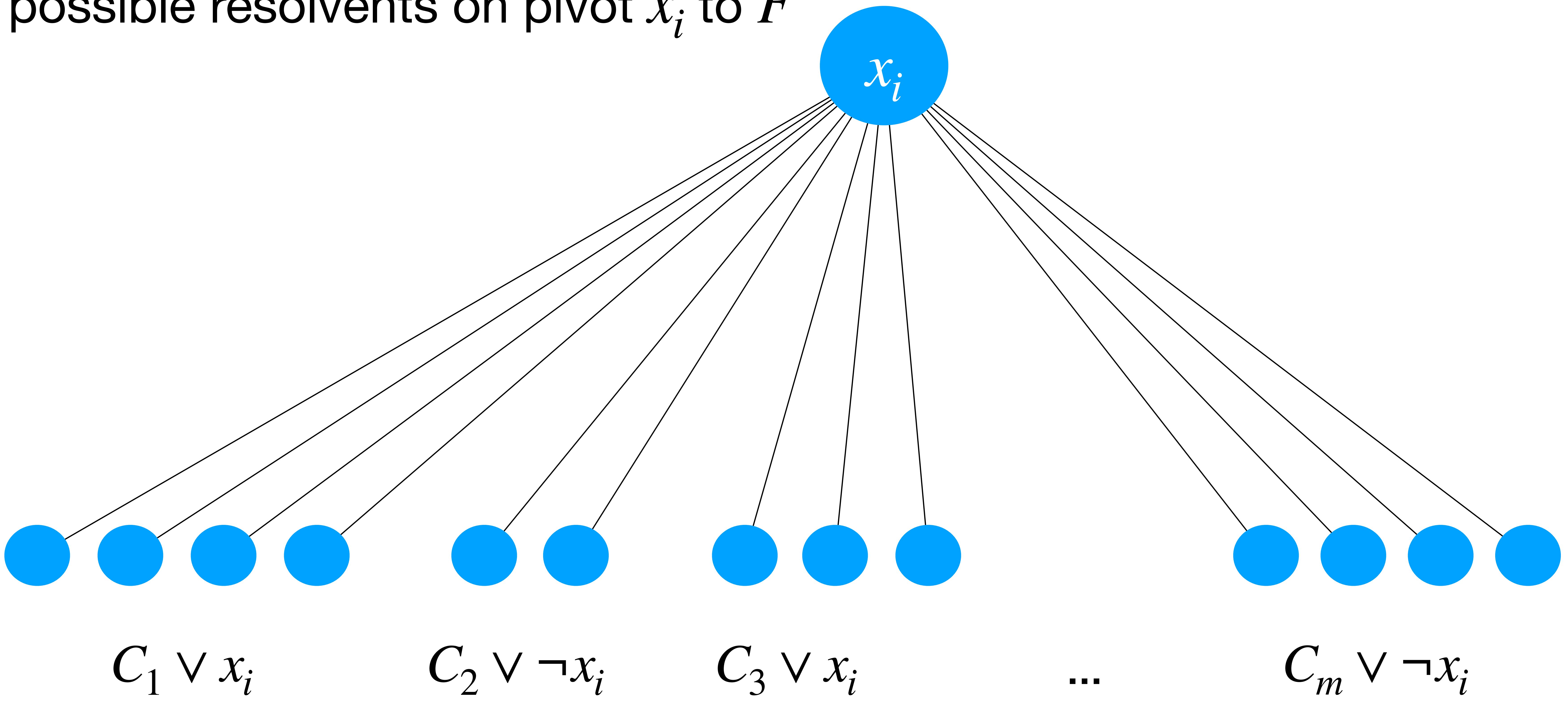
$C_1 \vee x_i$ $C_2 \vee \neg x_i$ $C_3 \vee x_i$ … $C_m \vee \neg x_i$

Variable Elimination in the Primal Graph



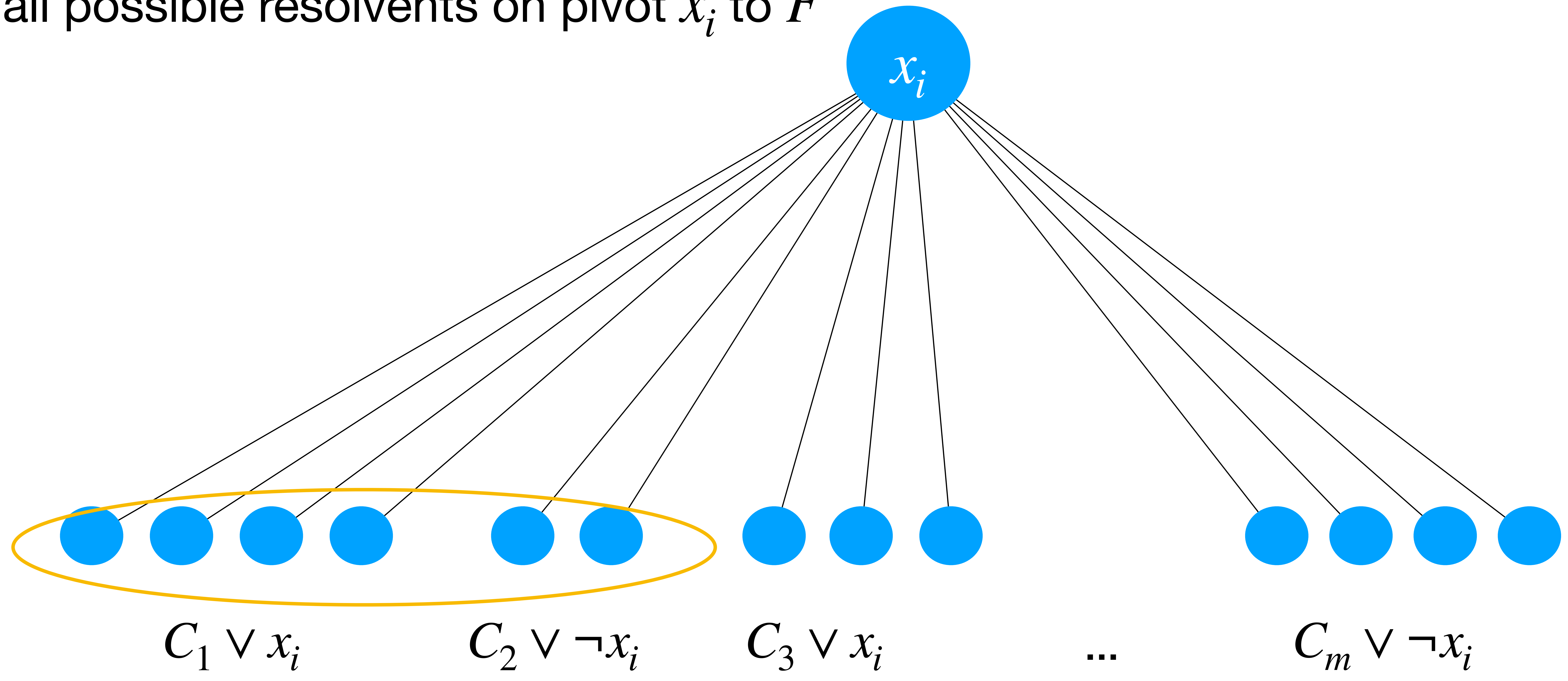
Variable Elimination in the Primal Graph

add all possible resolvents on pivot x_i to F



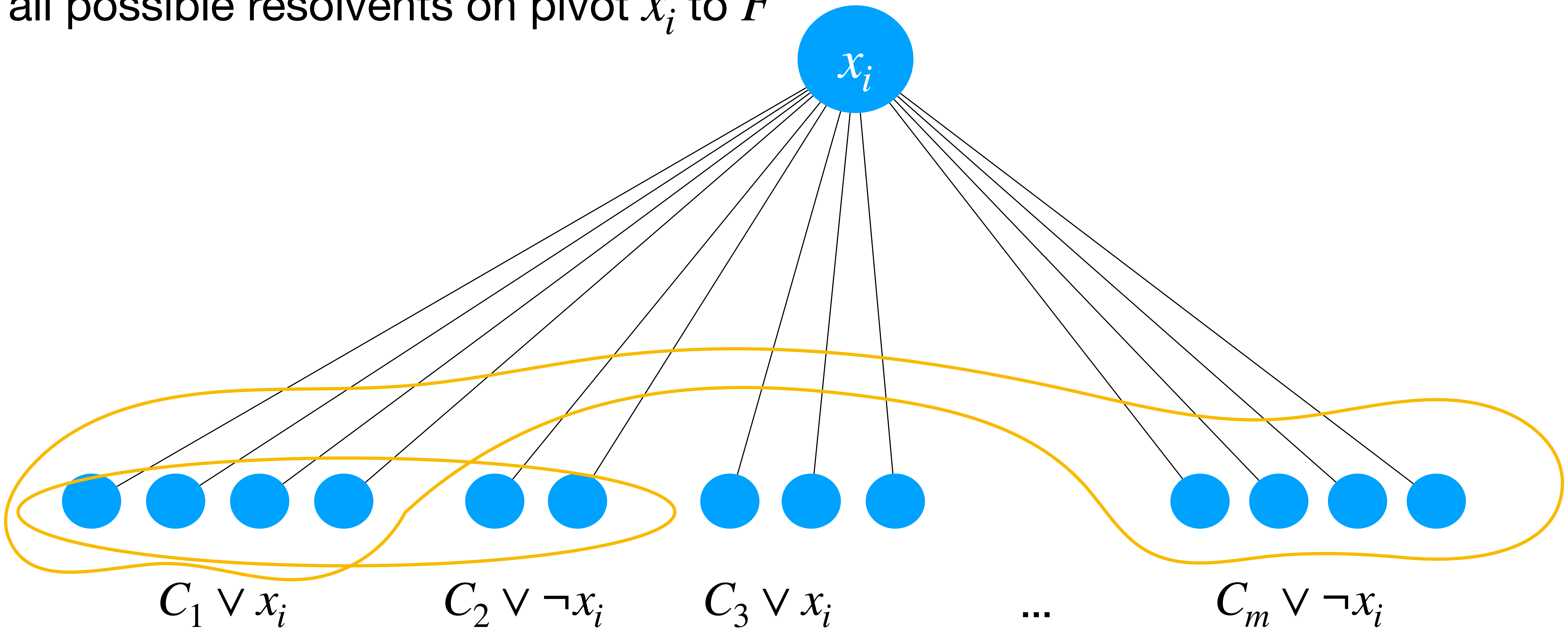
Variable Elimination in the Primal Graph

add all possible resolvents on pivot x_i to F



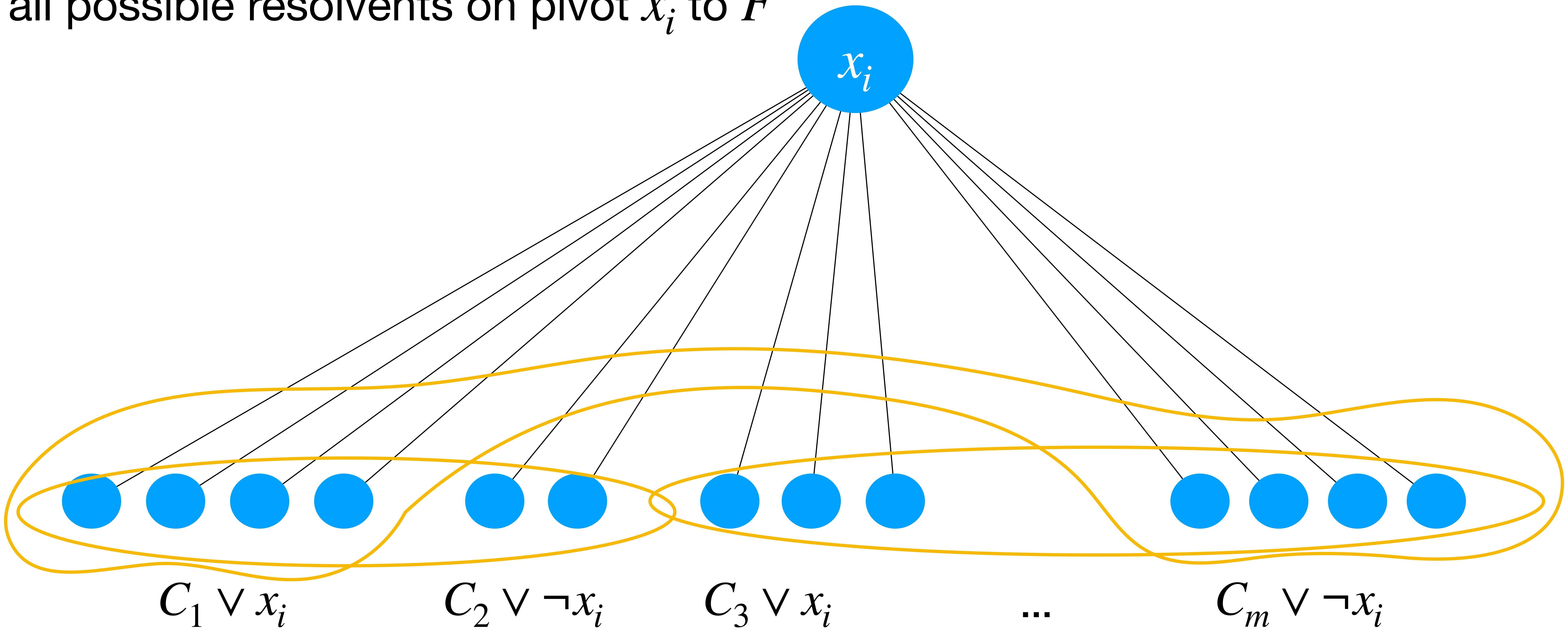
Variable Elimination in the Primal Graph

add all possible resolvents on pivot x_i to F



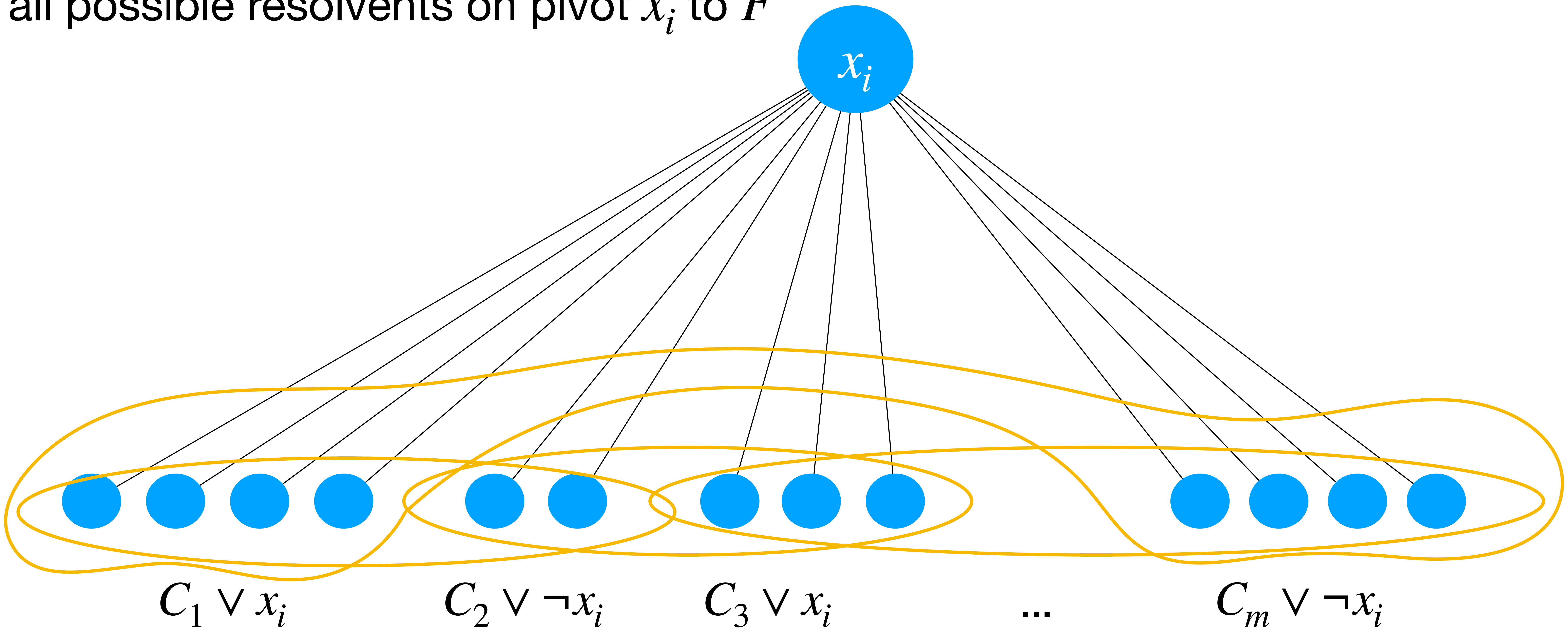
Variable Elimination in the Primal Graph

add all possible resolvents on pivot x_i to F



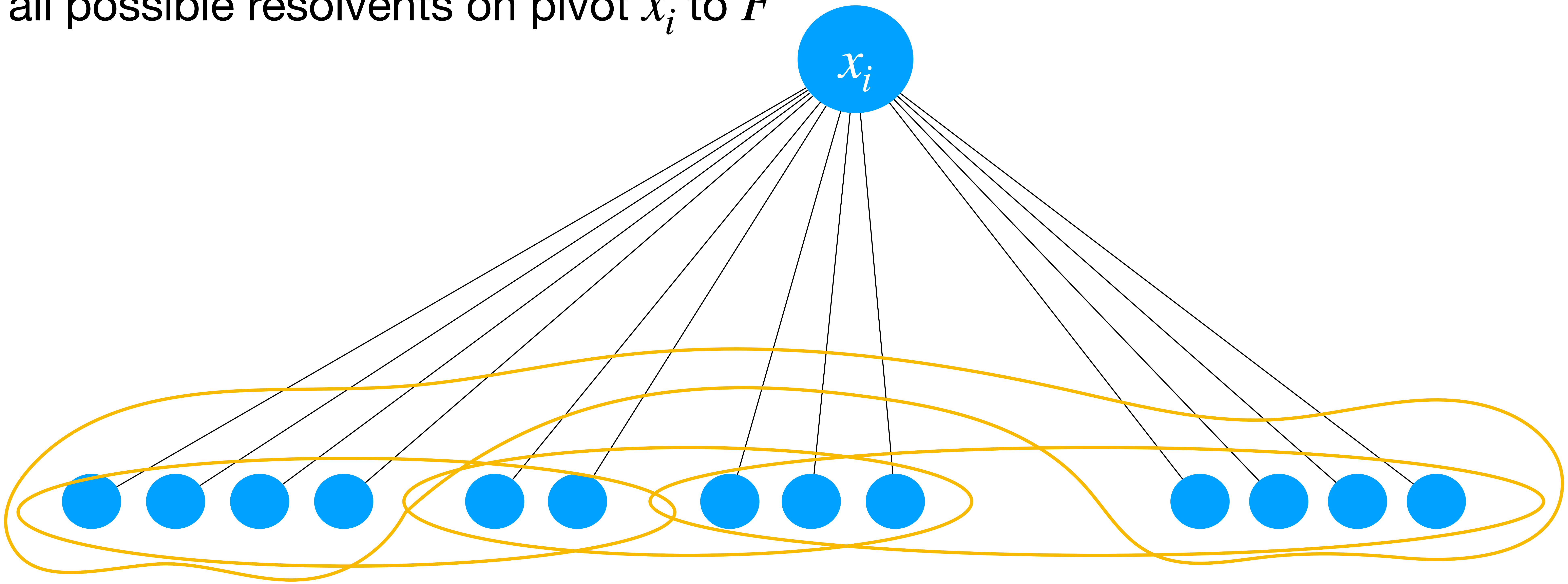
Variable Elimination in the Primal Graph

add all possible resolvents on pivot x_i to F



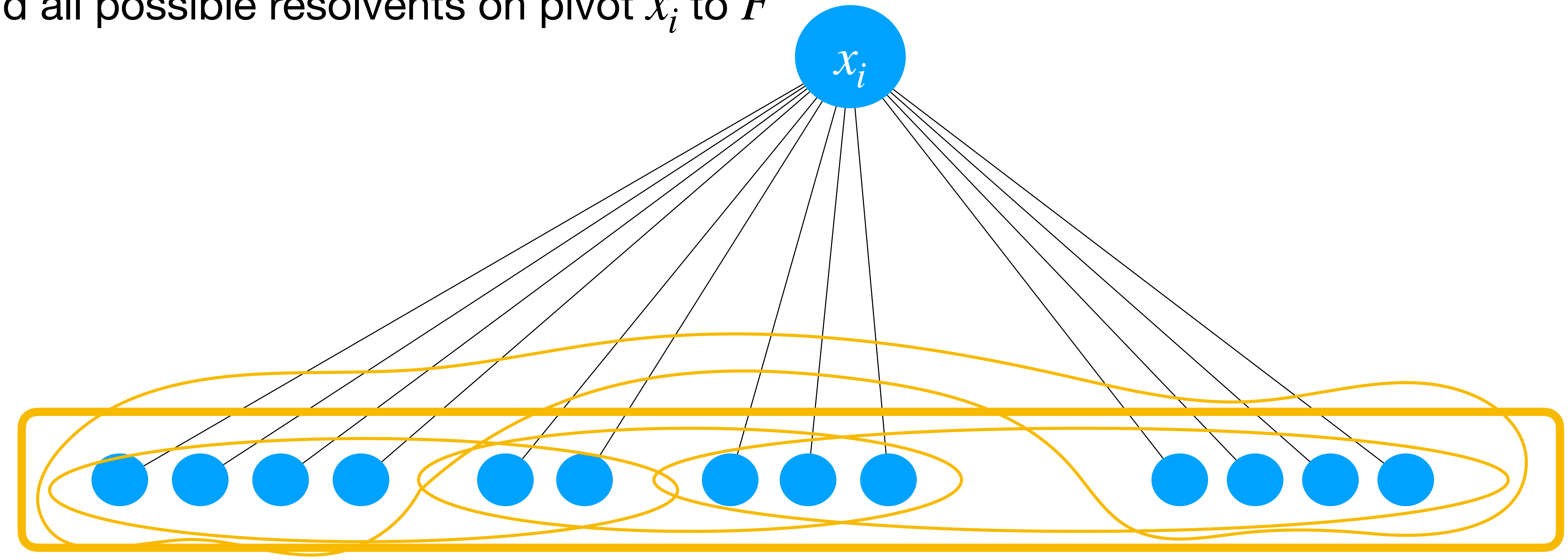
Variable Elimination in the Primal Graph

add all possible resolvents on pivot x_i to F



Variable Elimination in the Primal Graph

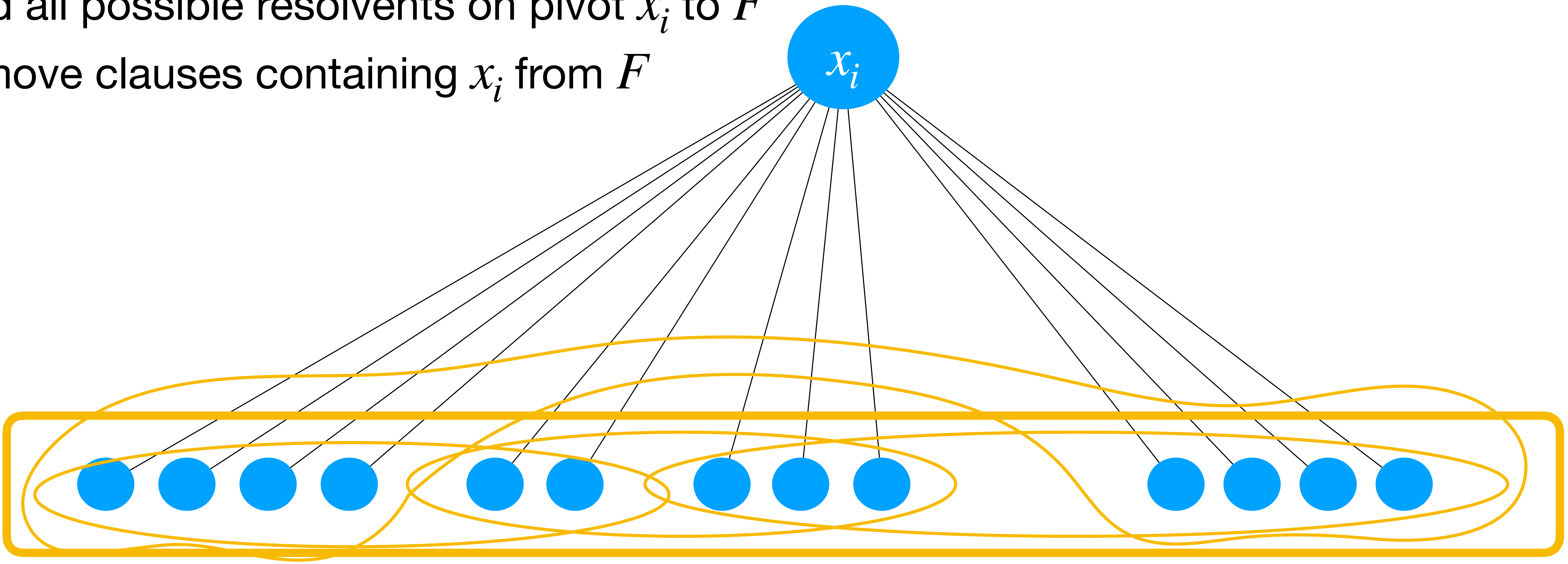
add all possible resolvents on pivot x_i to F



Variable Elimination in the Primal Graph

add all possible resolvents on pivot x_i to F

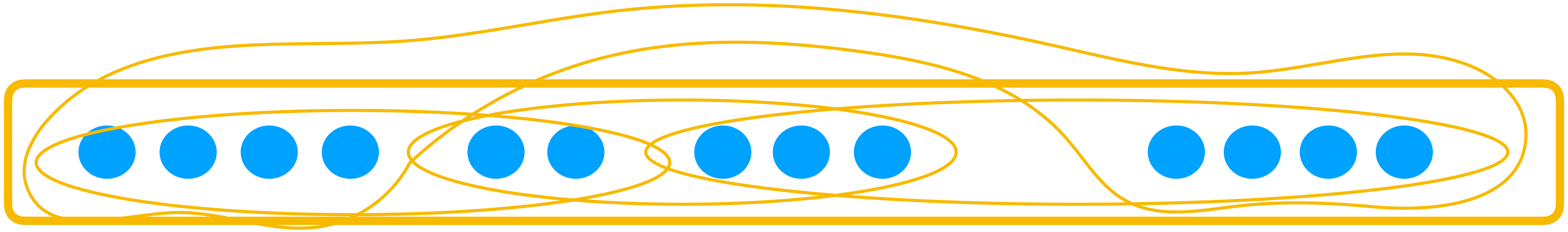
remove clauses containing x_i from F



Variable Elimination in the Primal Graph

add all possible resolvents on pivot x_i to F

remove clauses containing x_i from F



DP-Resolution and Elimination Orderings

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

for x_i **in** σ **:**

  add all possible resolvents on pivot x_i to F

  remove clauses containing x_i from F

  remove tautologies from F

return **false** $\Leftrightarrow$ F contains the empty clause

DP-Resolution and Elimination Orderings

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

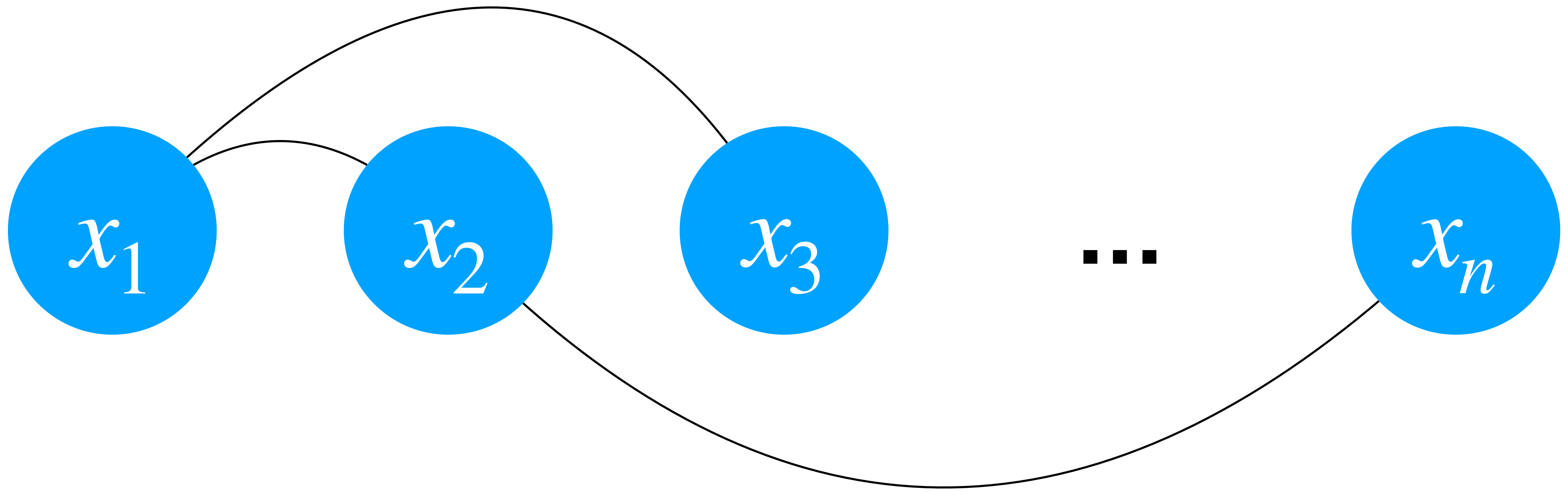
for x_i **in** σ **:**

    add all possible resolvents on pivot x_i to F

    remove clauses containing x_i from F

    remove tautologies from F

return false $\Leftrightarrow F$ contains the empty clause



DP-Resolution and Elimination Orderings

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

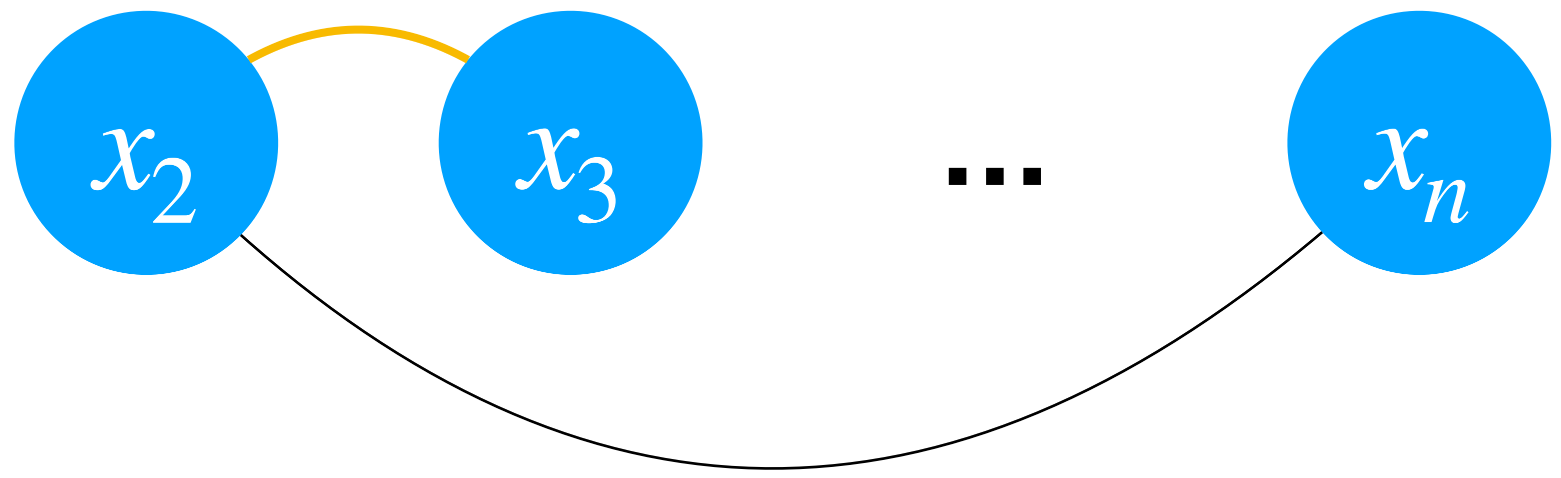
for x_i **in** σ **:**

add all possible resolvents on pivot x_i to F

remove clauses containing x_i from F

remove tautologies from F

return false $\Leftrightarrow$ F contains the empty clause



DP-Resolution and Elimination Orderings

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

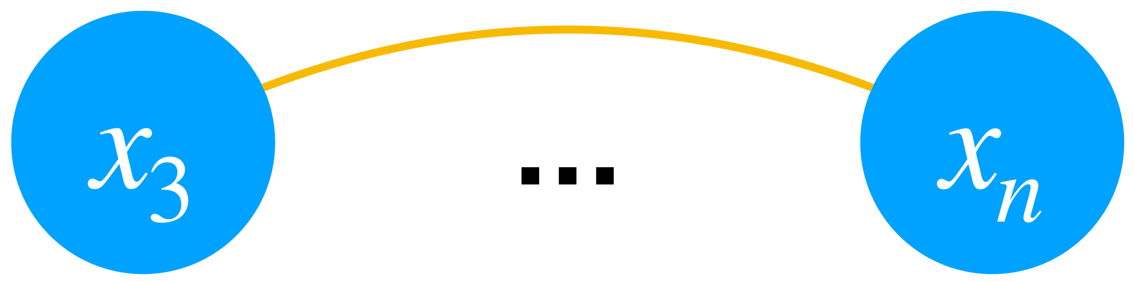
for x_i **in** σ **:**

add all possible resolvents on pivot x_i to F

remove clauses containing x_i from F

remove tautologies from F

return false $\Leftrightarrow$ F contains the empty clause



DP-Resolution and Elimination Orderings

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

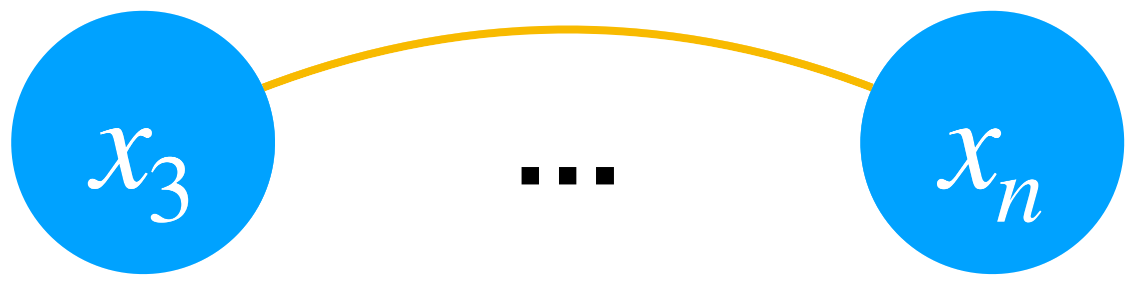
for x_i **in** σ **:**

add all possible resolvents on pivot x_i to F

remove clauses containing x_i from F

remove tautologies from F

return false $\Leftrightarrow$ F contains the empty clause



Observation

The size of any clause generated by Davis-Putnam Resolution is at most the width k of the elimination ordering in the primal graph.

DP-Resolution and Elimination Orderings

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

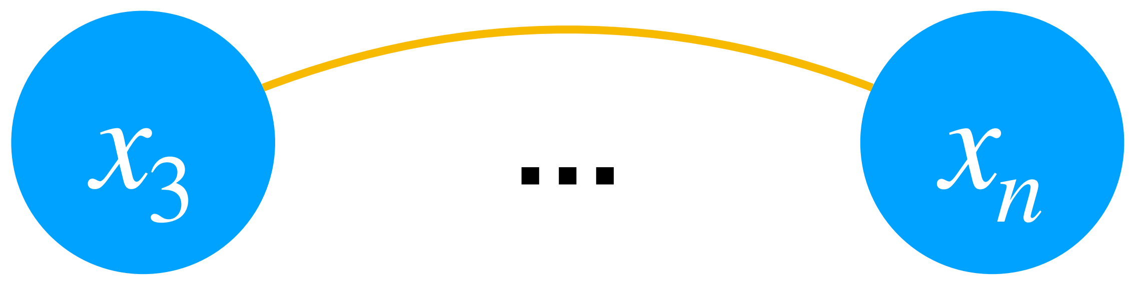
for x_i **in** σ **:**

  add all possible resolvents on pivot x_i to F

  remove clauses containing x_i from F

  remove tautologies from F

return false $\Leftrightarrow$ F contains the empty clause



Observation

The size of any clause generated by Davis-Putnam Resolution is at most the width k of the elimination ordering in the primal graph.

Worst case: 3^k resolvents in each iteration.

DP-Resolution and Elimination Orderings

Input: A CNF formula F with m clauses

Pick an ordering $\sigma := x_1, \ldots, x_n$ of variables

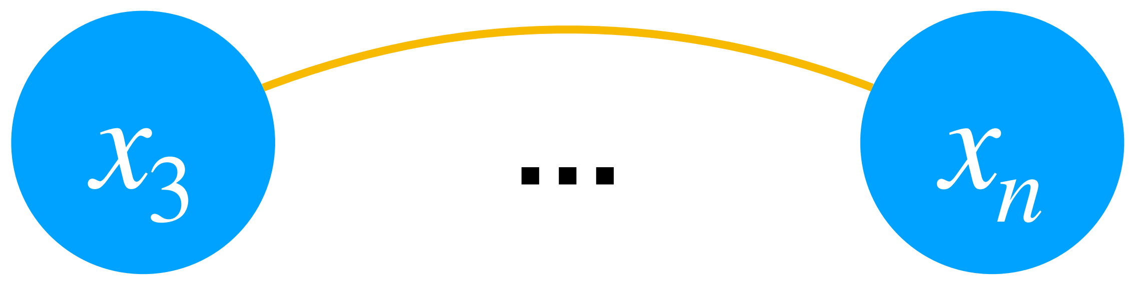
for x_i **in** σ **:**

add all possible resolvents on pivot x_i to F

remove clauses containing x_i from F

remove tautologies from F

return false $\Leftrightarrow$ F contains the empty clause



Observation

The size of any clause generated by Davis-Putnam Resolution is at most the width k of the elimination ordering in the primal graph.

Worst case: 3^k resolvents in each iteration.

Theorem

SAT is FPT parameterized by the treewidth of the primal graph.