Tight (Double) Exponential Lower Bounds for Identification Problems

@ ISAAC, 2024

Dec 9, 2024

Prafullkumar Tale

Joint work with

D. Chakraborty, F. Foucaud, and D. Majumdar

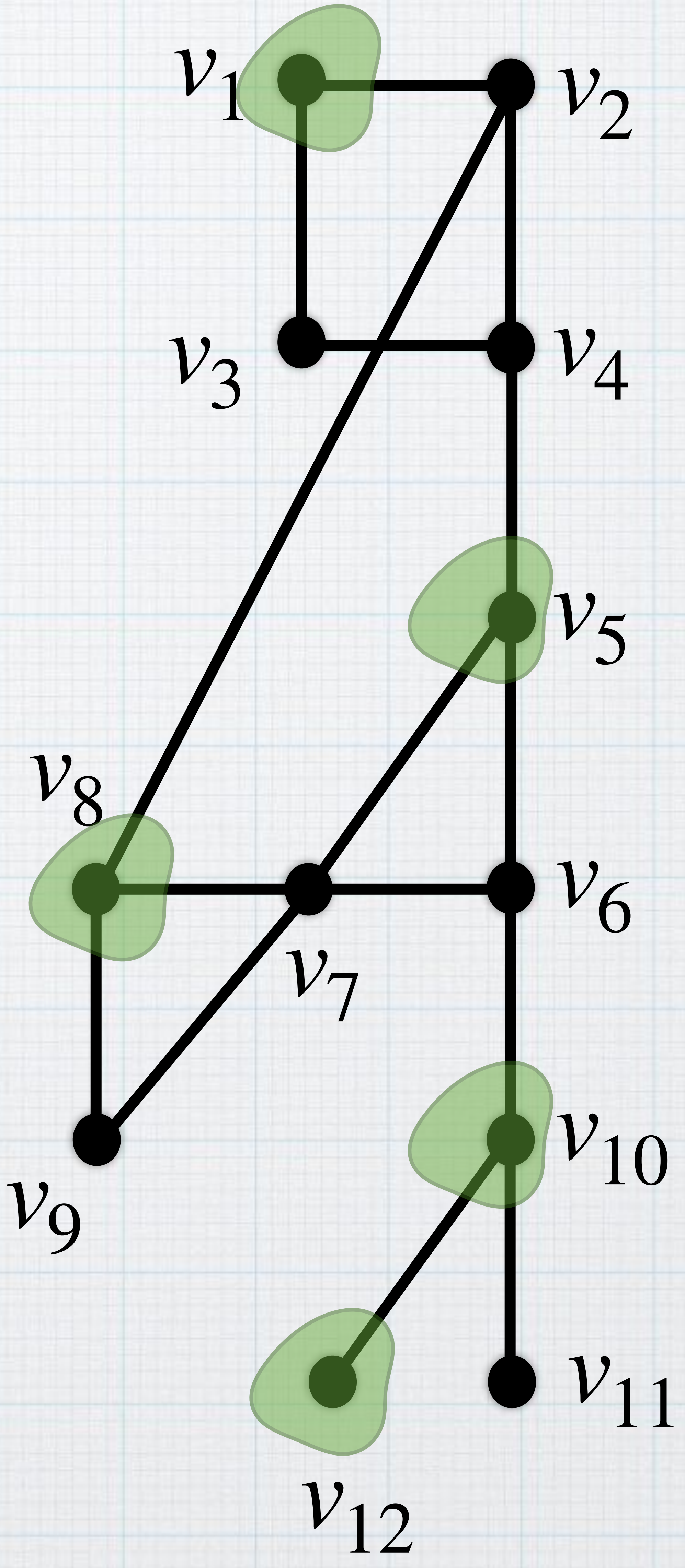
Locating Dominating Set

Input: Graph G, int k

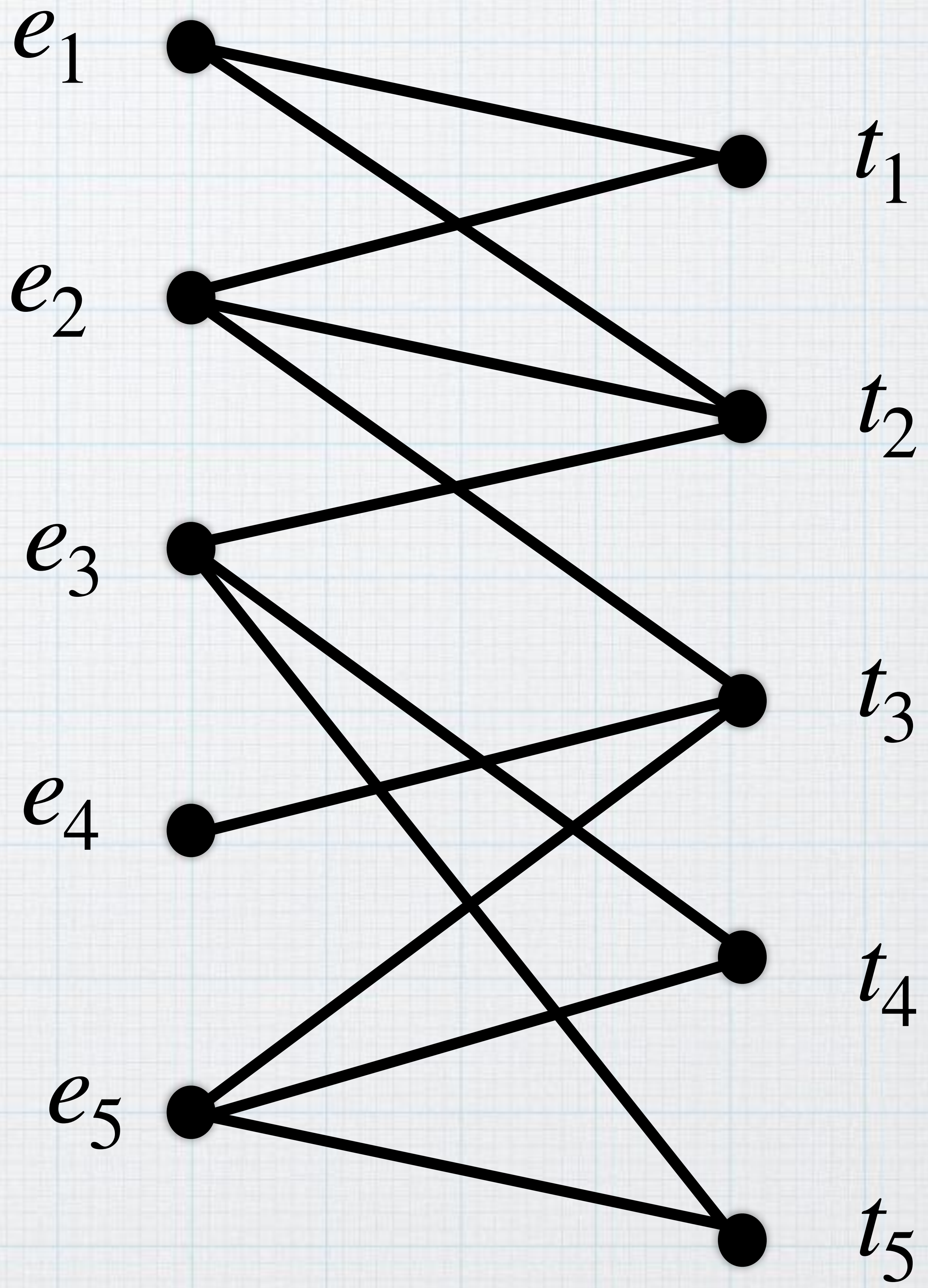
Output: $\exists$ a subset S of size k such that

(i) for any vertex $u \in V(G) \backslash S$, at least one of its neighbour is in S, and

(ii) for any two vertices $u \neq v \in V(G) \backslash S$, their neighbourhood in S are different, i.e., $N(u) \cap S \neq N(v) \cap S$.

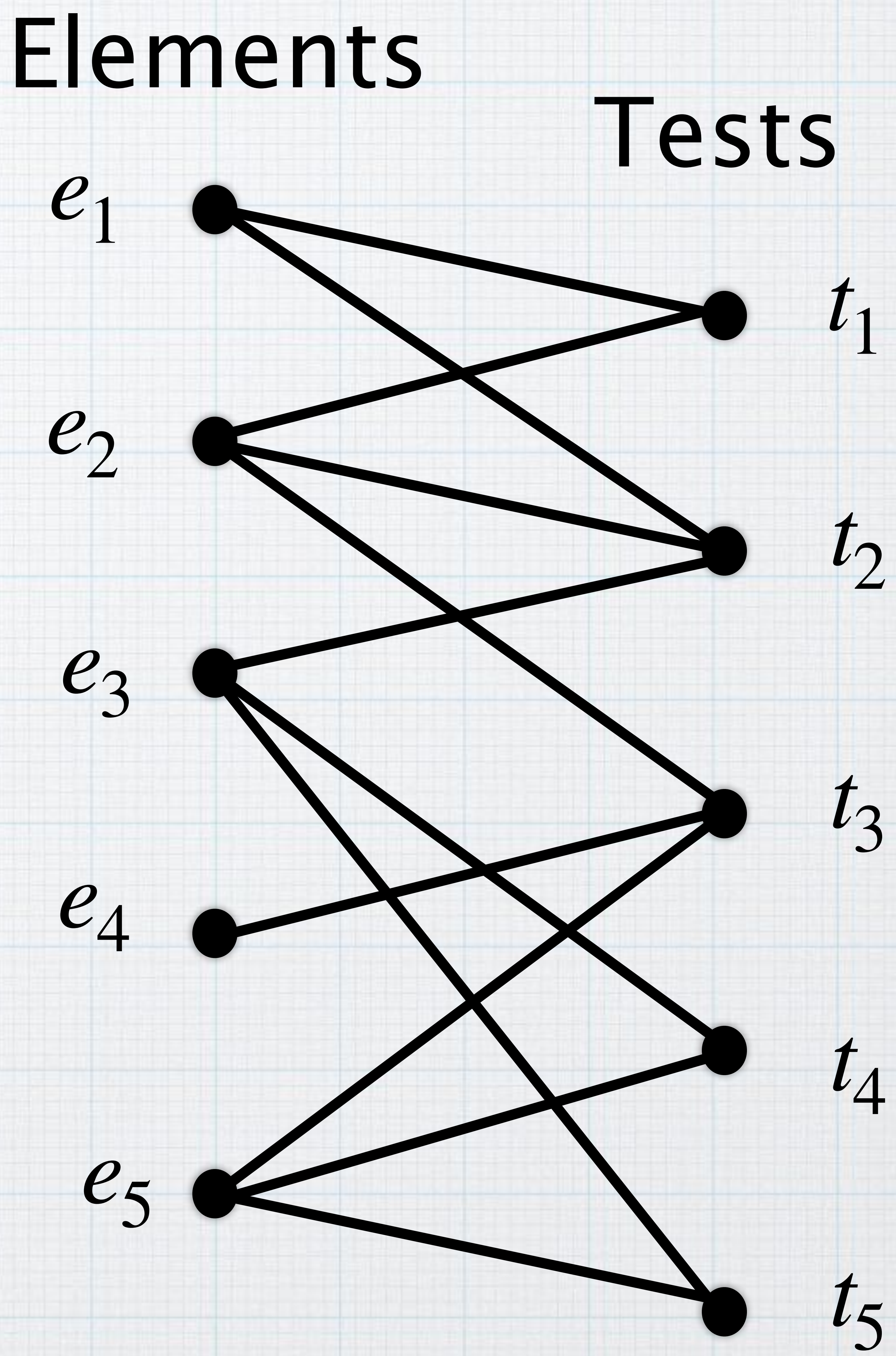


Test Cover



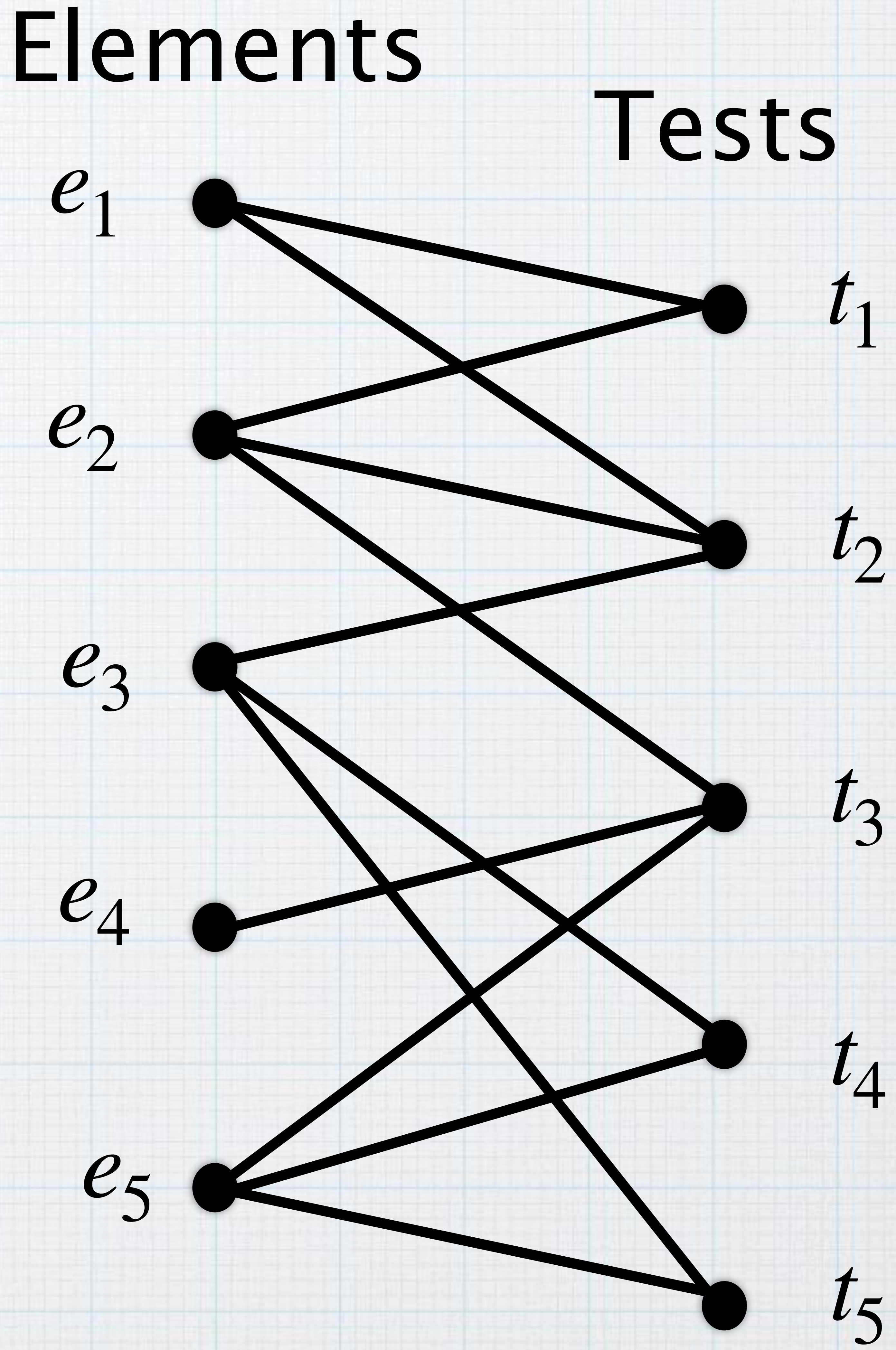
Test Cover

- Multiple elements, multiple tests



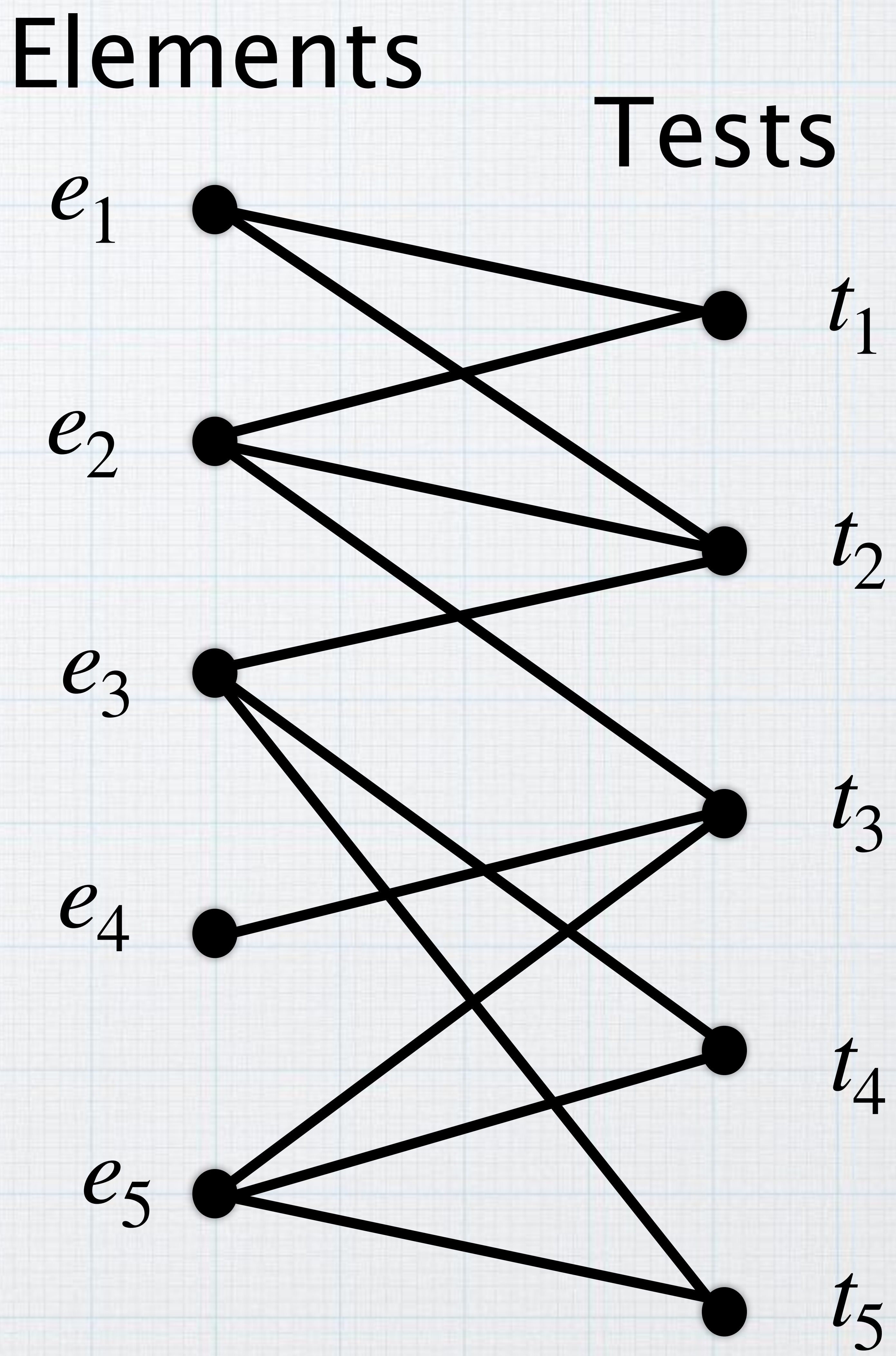
Test Cover

– Multiple elements, multiple tests
– Each test is positive for one or more elements



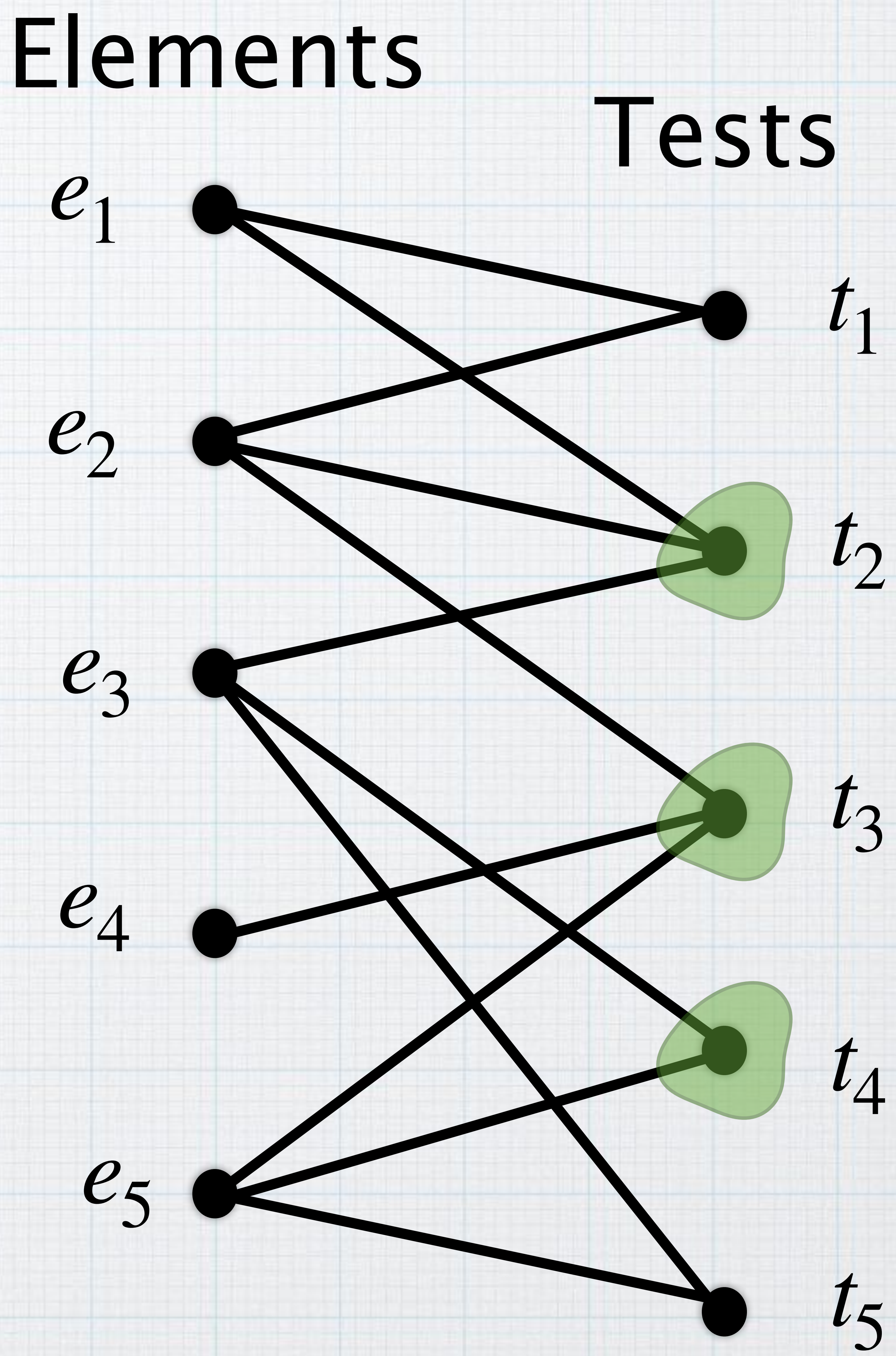
Test Cover

– Multiple elements, multiple tests
– Each test is positive for one or more elements
– Determine the smallest set of tests that need to be performed to uniquely identify the element



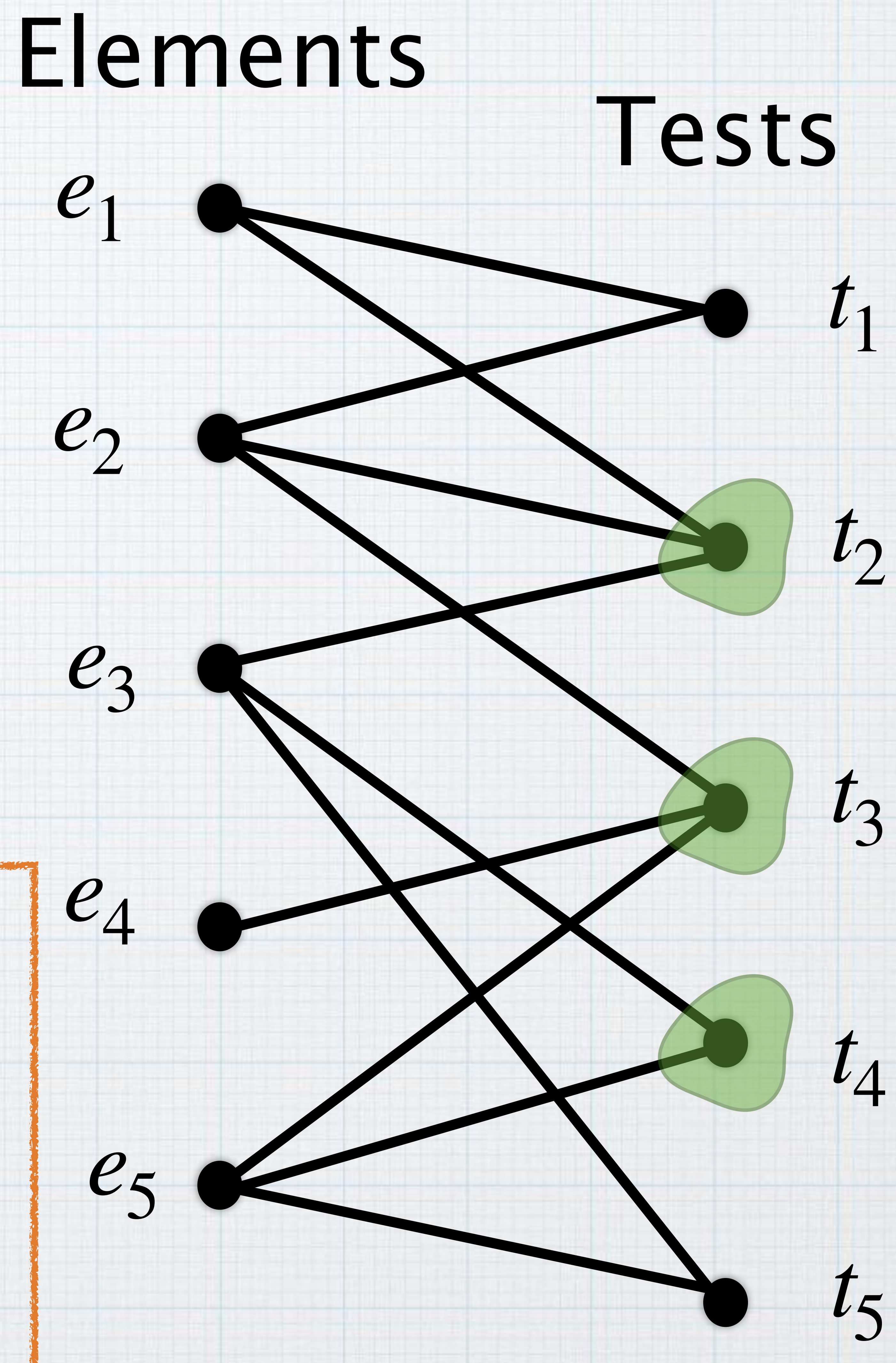
Test Cover

- Multiple elements, multiple tests
- Each test is positive for one or more elements
- Determine the smallest set of tests that need to be performed to uniquely identify the element



Test Cover

- Multiple elements, multiple tests
- Each test is positive for one or more elements
- Determine the smallest set of tests that need to be performed to uniquely identify the element



Test Cover

Input: Set of elements, collection of tests, int k

Output: Does there exist a collection of k tests s.t. for each pair of elements, there is a test that is positive for exactly one of them?

Parameterized Complexity

Parameterized Complexity

- Identify a relevant secondary measure (i.e. parameter).

Parameterized Complexity

- Identify a relevant secondary measure (i.e. parameter).
- Parameter: solution size, property of input graph

Parameterized Complexity

- Identify a relevant secondary measure (i.e. parameter).
- Parameter: solution size, property of input graph
- Π is fixed-parameter tractable (FPT) parameterized by k if there is an algo that solves it in $f(k) \cdot poly(n)$ time.

Parameterized Complexity

- Identify a relevant secondary measure (i.e. parameter).
- Parameter: solution size, property of input graph
- Π is fixed-parameter tractable (FPT) parameterized by k if there is an algo that solves it in $f(k) \cdot poly(n)$ time.
- Π is $W[1]$-hard when parameterized by k if there is no algo that solves it in $f(k) \cdot poly(n)$ time.

- Parameter: solution size, property of input graph

- Parameter: solution size, property of input graph

Locating Dominating Set

- Parameter: solution size, property of input graph

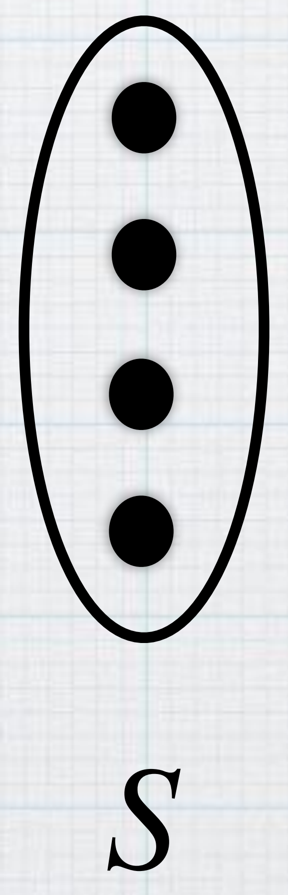
Locating Dominating Set

Obs: Any solution of size k can locate at most $2^k - 1$ vertices.

- Parameter: solution size, property of input graph

Locating Dominating Set

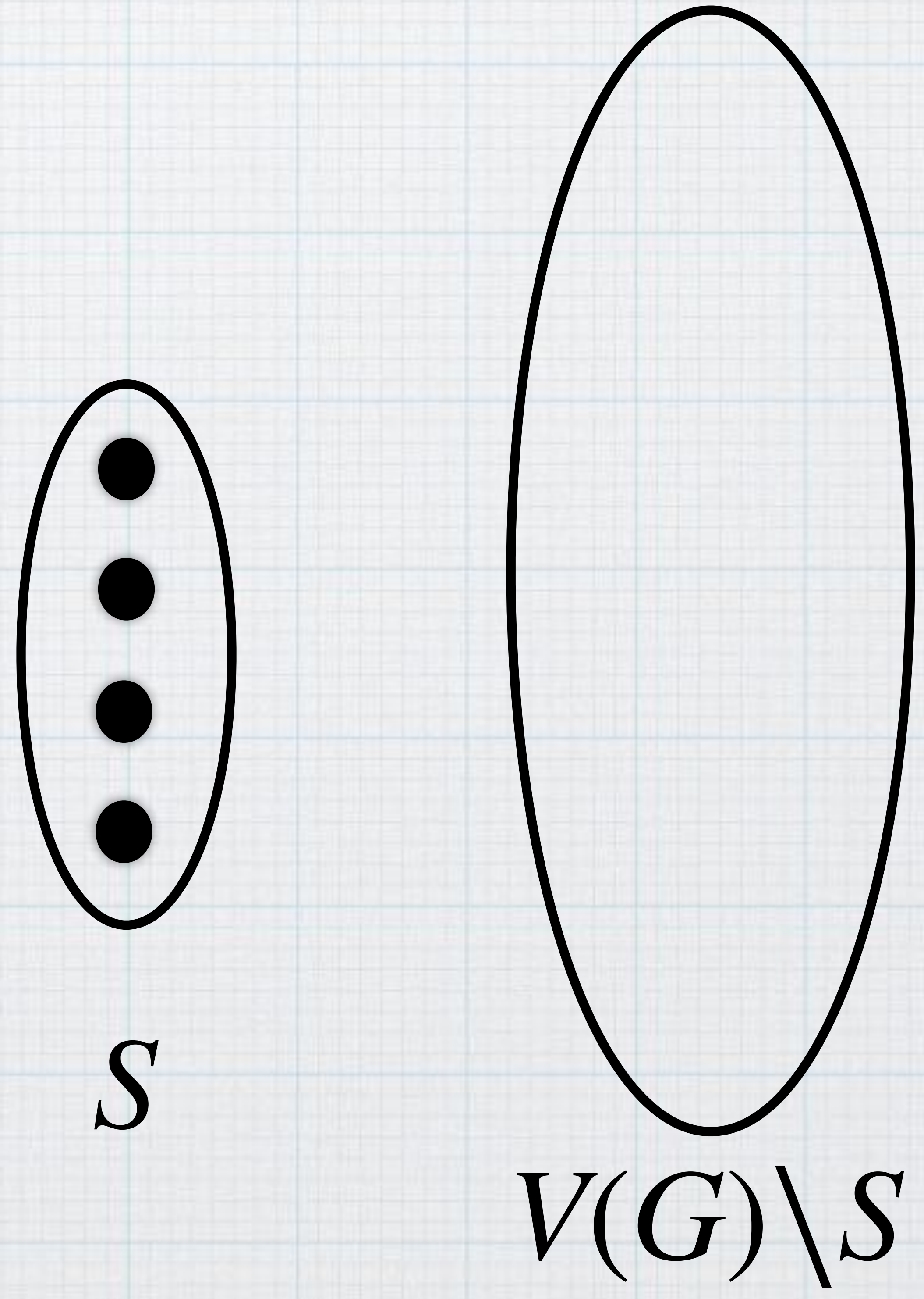
Obs: Any solution of size k can locate at most $2^k - 1$ vertices.



- Parameter: solution size, property of input graph

Locating Dominating Set

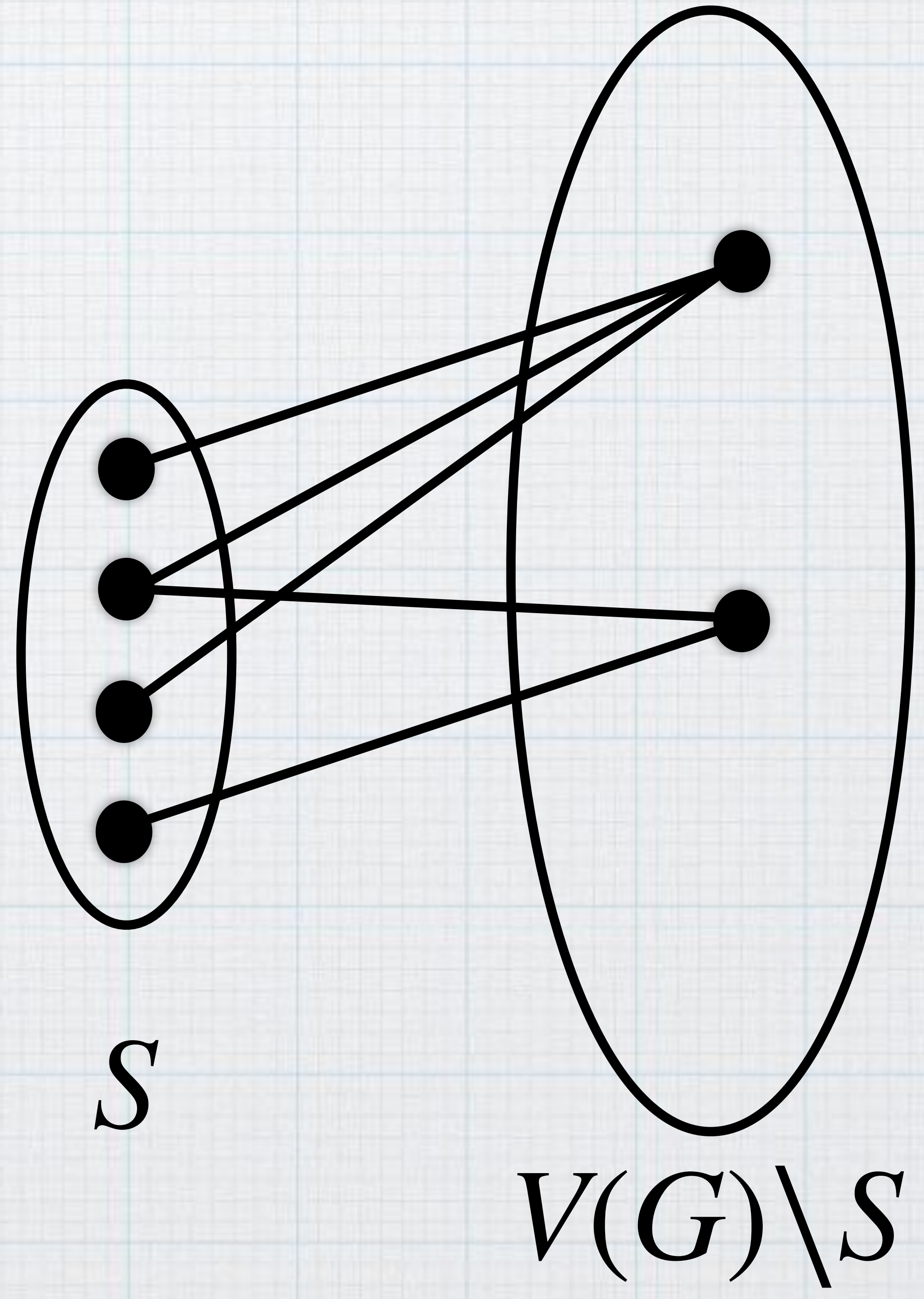
Obs: Any solution of size k can locate at most $2^k - 1$ vertices.



- Parameter: solution size, property of input graph

Locating Dominating Set

Obs: Any solution of size k can locate at most $2^k - 1$ vertices.

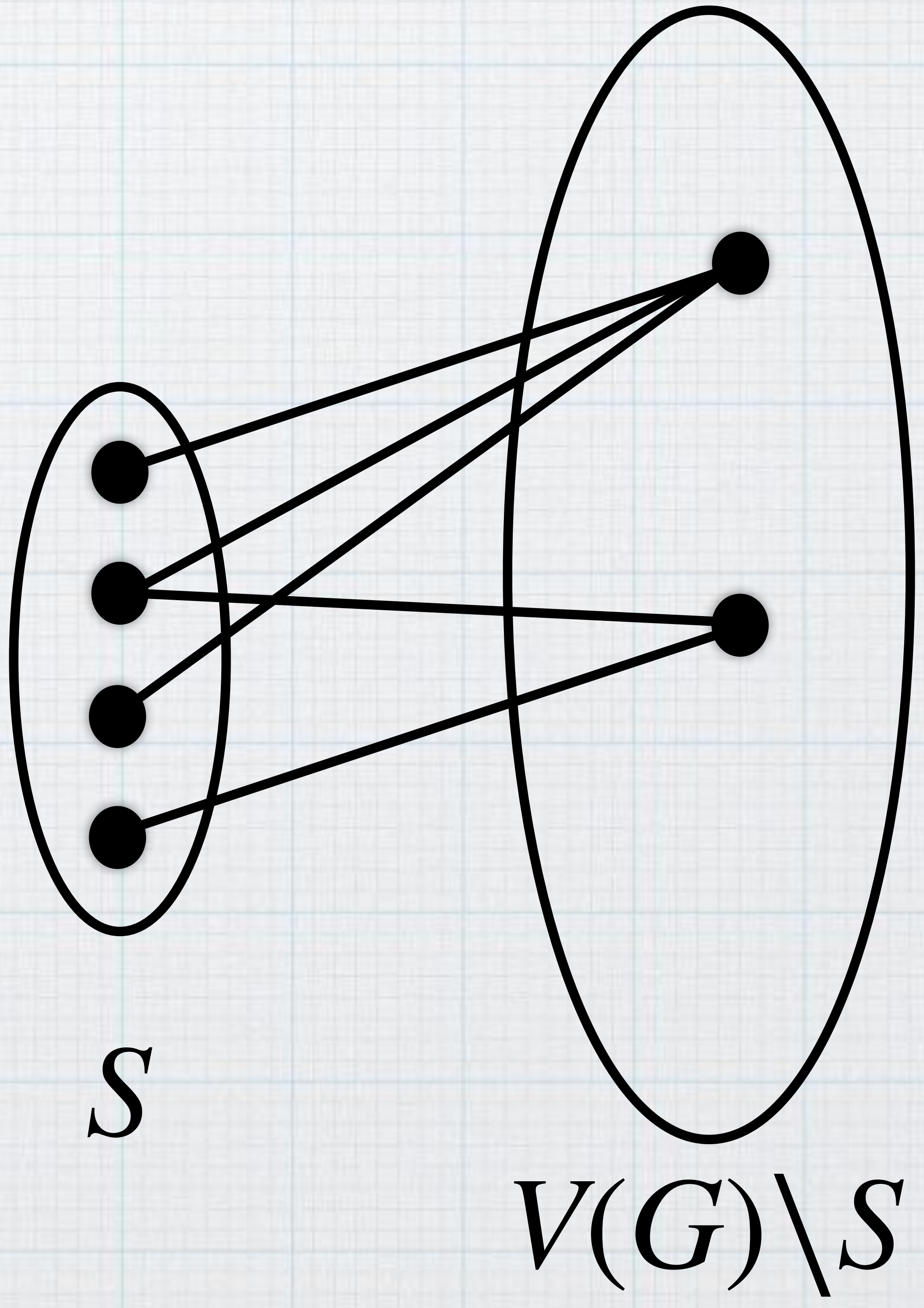


- Parameter: solution size, property of input graph

Locating Dominating Set

Obs: Any solution of size k can locate at most $2^k - 1$ vertices.

– If $|V(G)| > k + (2^k - 1)$ return No.
– Enumerate all possible subsets of size $\leq k$



- Parameter: solution size, property of input graph

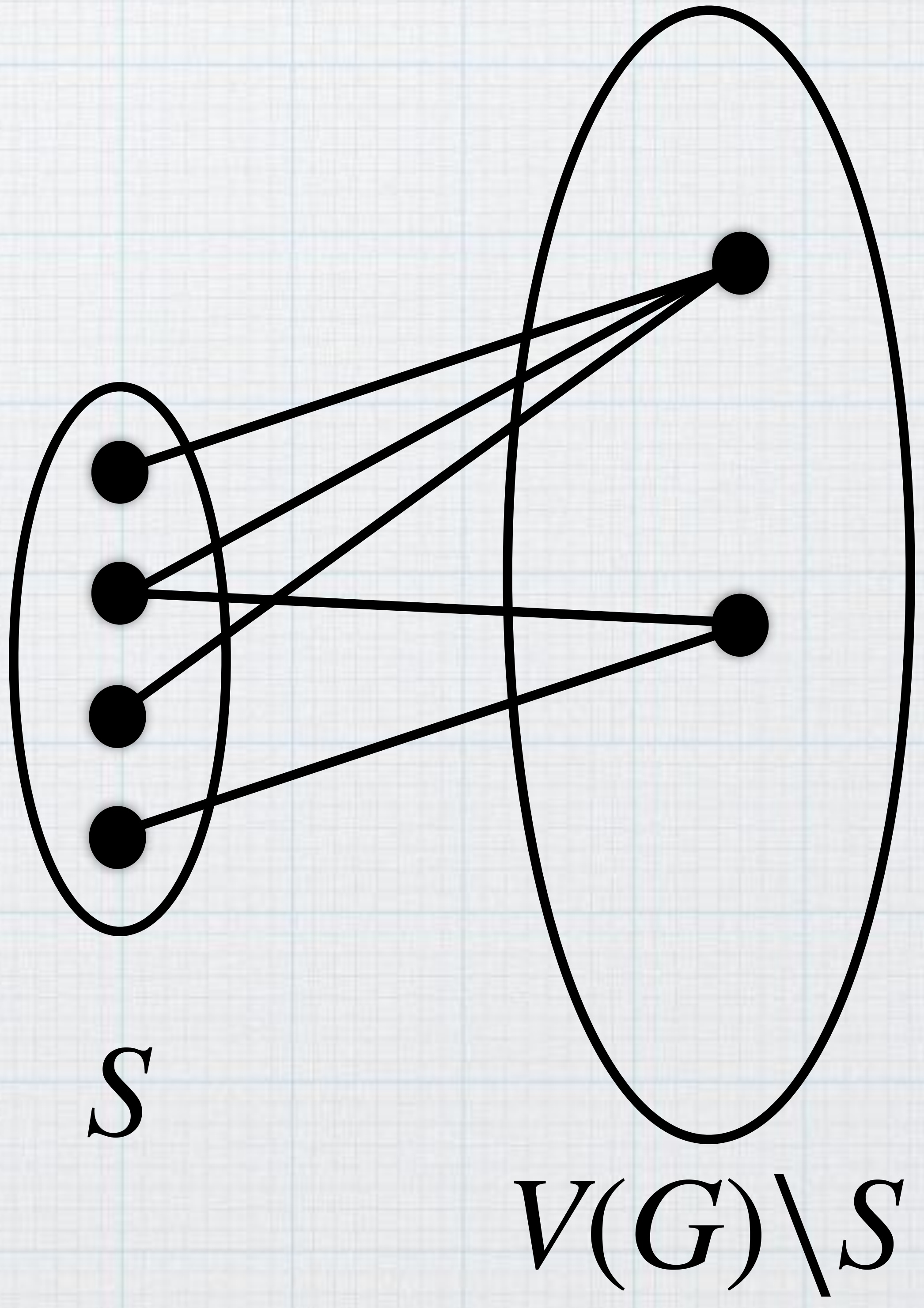
Locating Dominating Set

Obs: Any solution of size k can locate at most $2^k - 1$ vertices.

– If $|V(G)| > k + (2^k - 1)$ return No.

– Enumerate all possible subsets of size $\leq k$

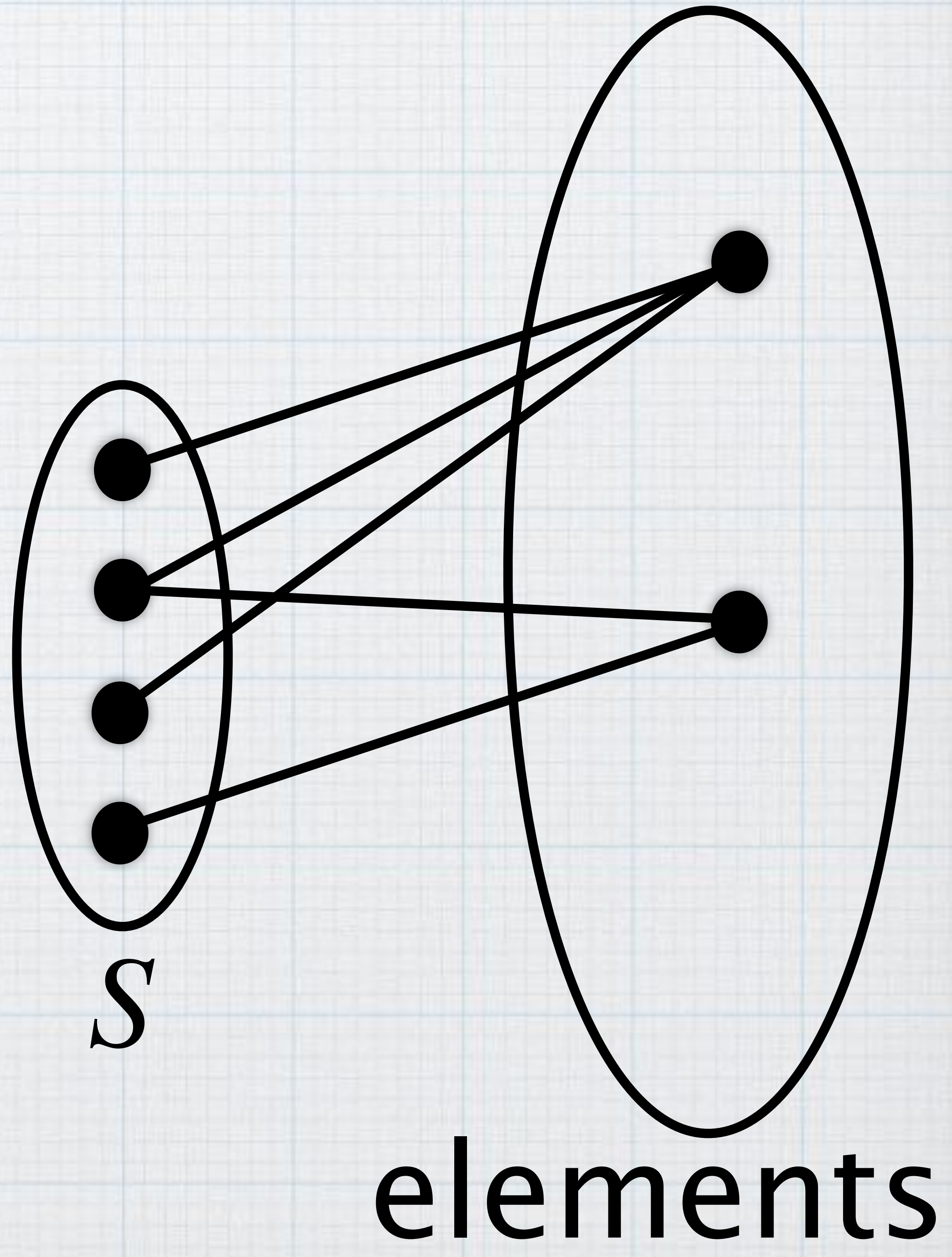
Claim: Locating Dominating Set admits algo running in time $\binom{2^k}{k} \cdot n^{\mathcal{O}(1)} = 2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$.



- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover

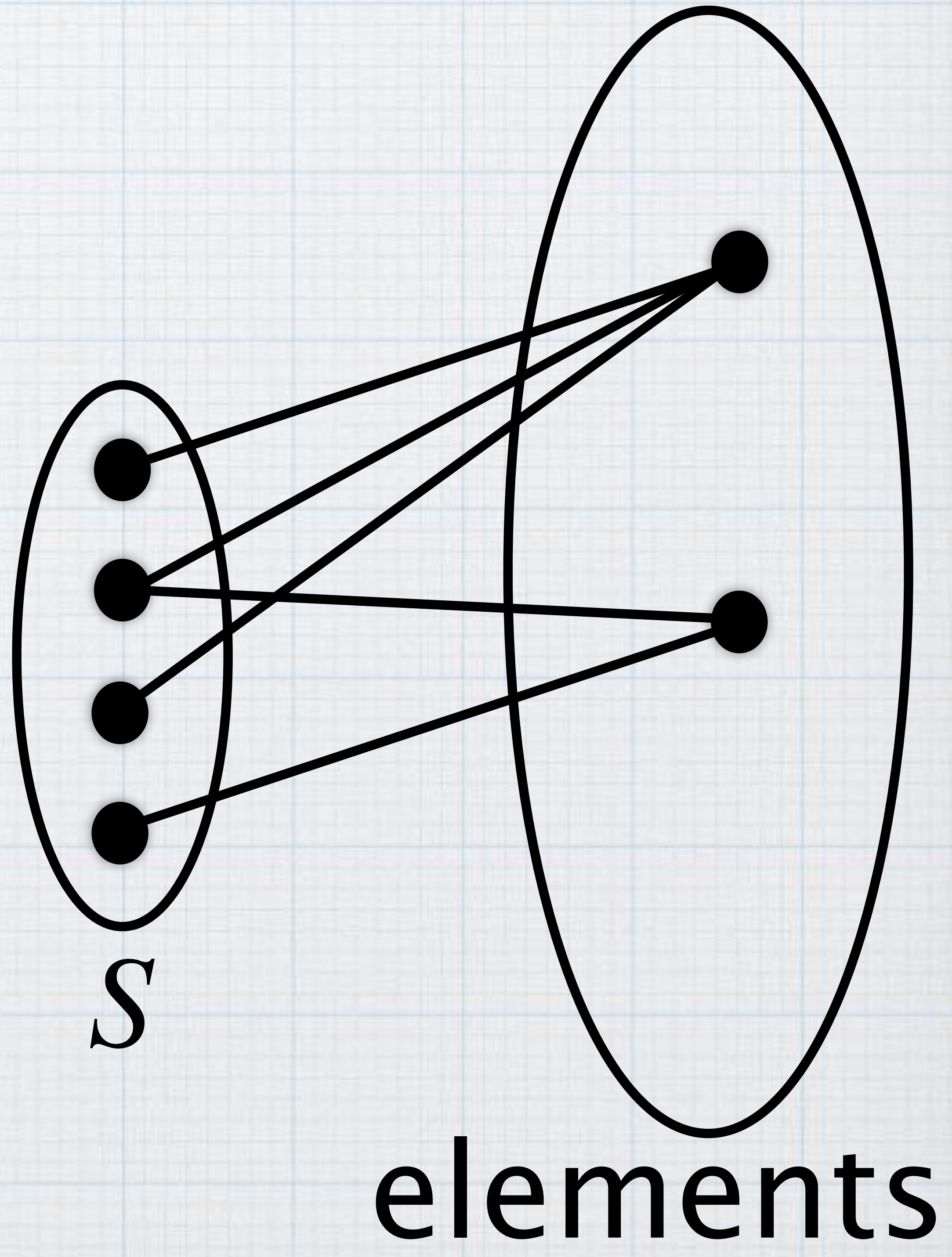


- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover

Obs: Any k tests can identify at most $2^k - 1$ elements.



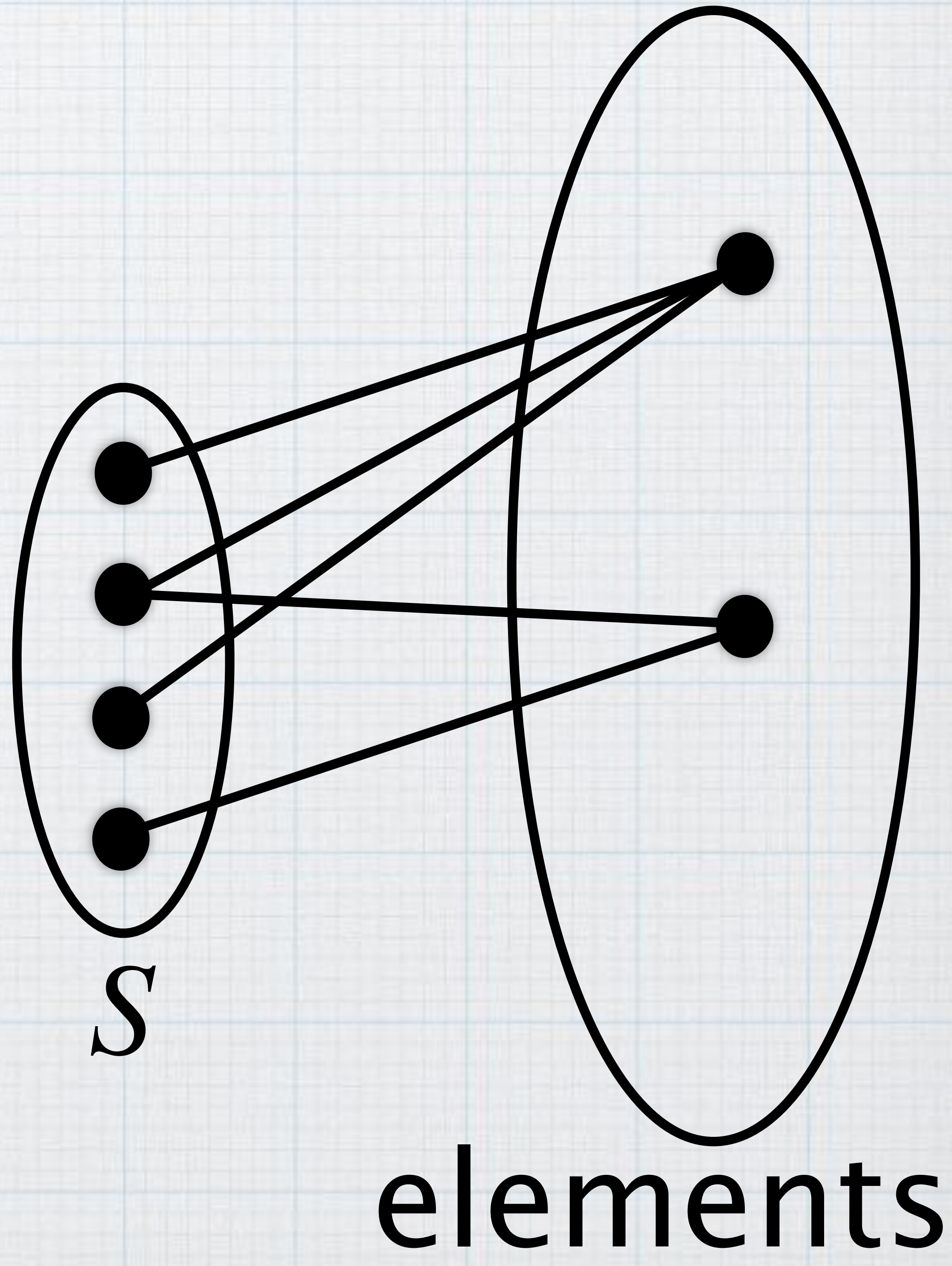
- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover

Obs: Any k tests can identify at most $2^k - 1$ elements.

Nr of elements is at most 2^k elements.



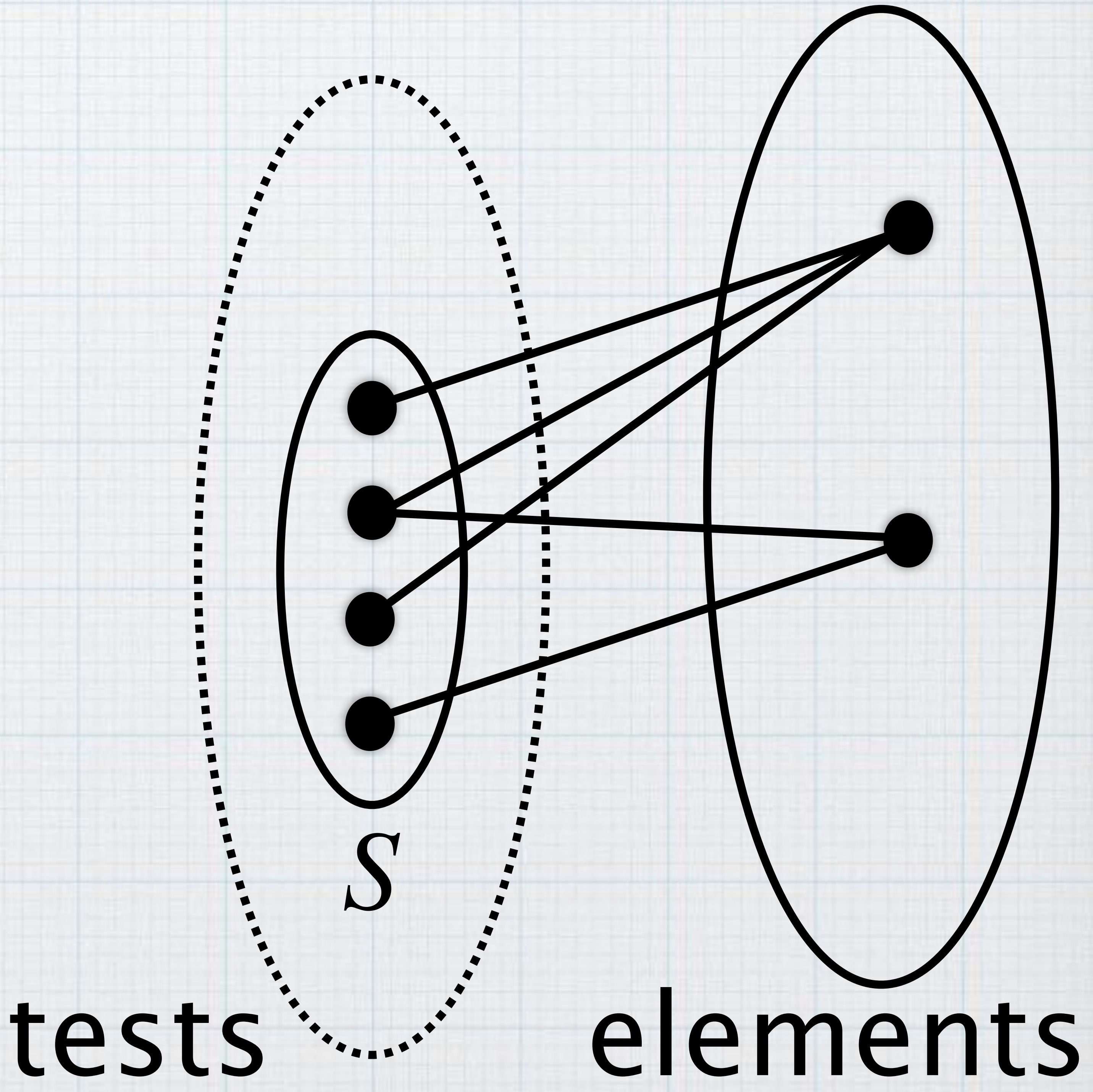
- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover

Obs: Any k tests can identify at most $2^k - 1$ elements.

Nr of elements is at most 2^k elements.



- Parameter: solution size, property of input graph

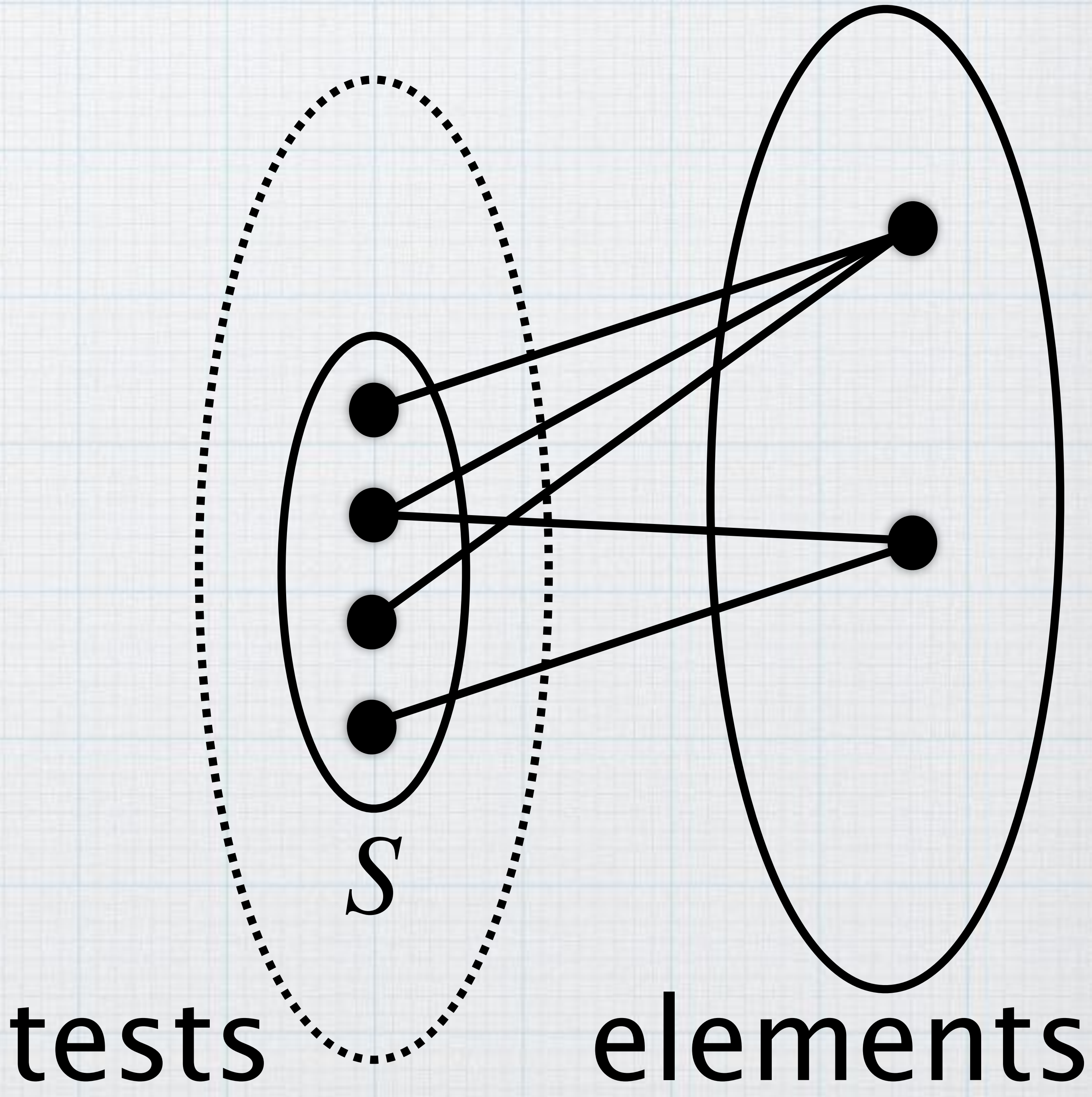
Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover

Obs: Any k tests can identify at most $2^k - 1$ elements.

Nr of elements is at most 2^k elements.

Nr of unique tests is at most 2^{2^k} elements.



- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

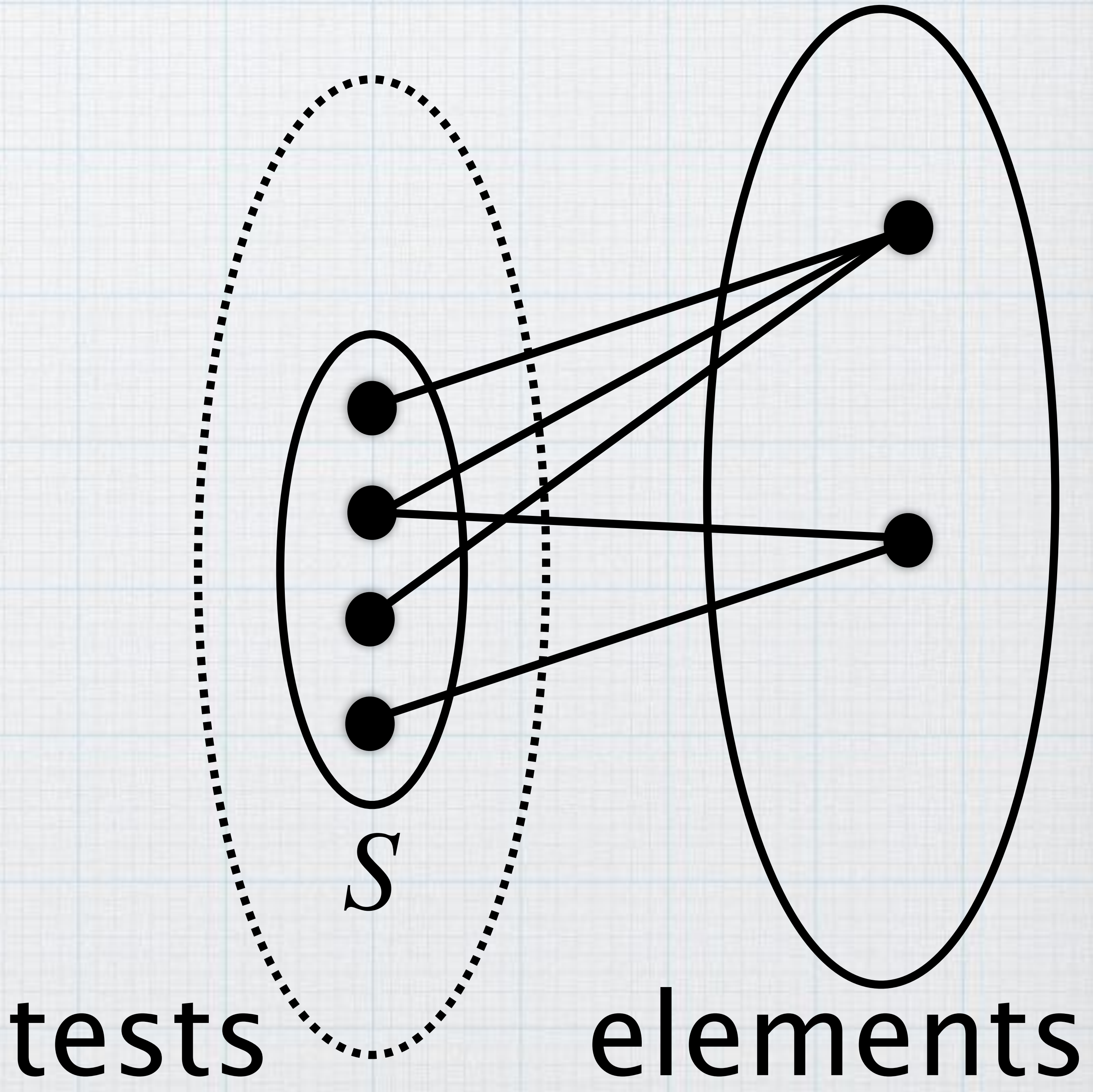
Test Cover

Obs: Any k tests can identify at most $2^k - 1$ elements.

Nr of elements is at most 2^k elements.

Nr of unique tests is at most 2^{2^k} elements.

Claim: Test Cover admits algo running in time $\binom{2^{2^k}}{k} \cdot n^{\mathcal{O}(1)} = 2^{2^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$.



- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover admits algo running in time $2^{2^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$

- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover admits algo running in time $2^{2^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$

- Parameter: solution size, property of input graph

- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover admits algo running in time $2^{2^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$

- Parameter: solution size, property of input graph

Treewidth (tw):
closeness to trees

- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover admits algo running in time $2^{2^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$

- Parameter: solution size, property of input graph

Treewidth (tw): closeness to trees

Locating Dominating Set admits algo running in time $2^{2^{\mathcal{O}(\text{tw})}} \cdot n^{\mathcal{O}(1)}$

- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover admits algo running in time $2^{2^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$

- Parameter: solution size, property of input graph

Treewidth (tw): closeness to trees

Locating Dominating Set admits algo running in time $2^{2^{\mathcal{O}(\text{tw})}} \cdot n^{\mathcal{O}(1)}$

Test Cover admits algo running in time $2^{2^{\mathcal{O}(\text{tw})}} \cdot n^{\mathcal{O}(1)}$.

- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover admits algo running in time $2^{2^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$

- Parameter: solution size, property of input graph

Treewidth (tw): closeness to trees

Locating Dominating Set admits algo running in time $2^{2^{\mathcal{O}(\text{tw})}} \cdot n^{\mathcal{O}(1)}$

Test Cover admits algo running in time $2^{2^{\mathcal{O}(\text{tw})}} \cdot n^{\mathcal{O}(1)}$.

What is the best possible multiplier $f(k)$ in the running time?

- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover admits algo running in time $2^{2^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$

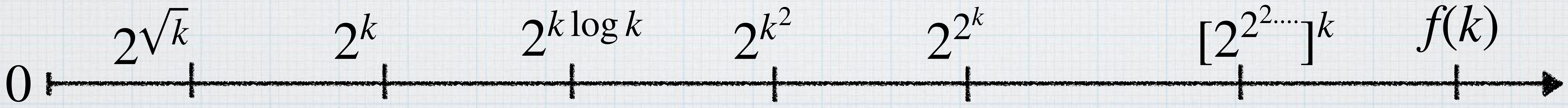
- Parameter: solution size, property of input graph

Treewidth (tw): closeness to trees

Locating Dominating Set admits algo running in time $2^{2^{\mathcal{O}(\text{tw})}} \cdot n^{\mathcal{O}(1)}$

Test Cover admits algo running in time $2^{2^{\mathcal{O}(\text{tw})}} \cdot n^{\mathcal{O}(1)}$.

What is the best possible multiplier $f(k)$ in the running time?



- Parameter: solution size, property of input graph

Locating Dominating Set admits algo running in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$

Test Cover admits algo running in time $2^{2^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$

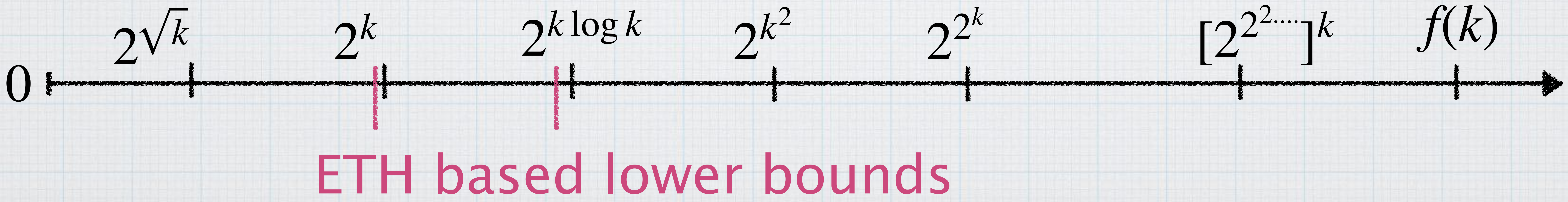
- Parameter: solution size, property of input graph

Treewidth (tw): closeness to trees

Locating Dominating Set admits algo running in time $2^{2^{\mathcal{O}(\text{tw})}} \cdot n^{\mathcal{O}(1)}$

Test Cover admits algo running in time $2^{2^{\mathcal{O}(\text{tw})}} \cdot n^{\mathcal{O}(1)}$.

What is the best possible multiplier $f(k)$ in the running time?



Our results

Our results

Thm. The Locating Dominating Set problem

— admits $2^{2^{\mathcal{O}(\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$–time algo, but

— does not admit $2^{2^{o(\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$ algo unless the ETH fails.

Our results

Thm. The Locating Dominating Set problem
— admits $2^{2^{\mathcal{O}(\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$–time algo, but
— does not admit $2^{2^{o(\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$ algo unless the ETH fails.

Thm. The Test Cover problem
— admits $2^{2^{\mathcal{O}(\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$–time algo, but
— does not admit $2^{2^{o(\mathsf{tw})}} \cdot n^{\mathcal{O}(1)}$ algo unless the ETH fails.

Our results

Our results

Thm. The Locating Dominating Set problem
— admits $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$–time algo, but
— does not admit $2^{o(k^2)} \cdot n^{\mathcal{O}(1)}$ algo unless the ETH fails.

Our results

Thm. The Locating Dominating Set problem
— admits $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$–time algo, but
— does not admit $2^{o(k^2)} \cdot n^{\mathcal{O}(1)}$ algo unless the ETH fails.

Thm. The Test Cover problem
— admits $2^{2^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$–time algo, but
— does not admit $2^{2^{o(k)}} \cdot n^{\mathcal{O}(1)}$ algo unless the ETH fails.

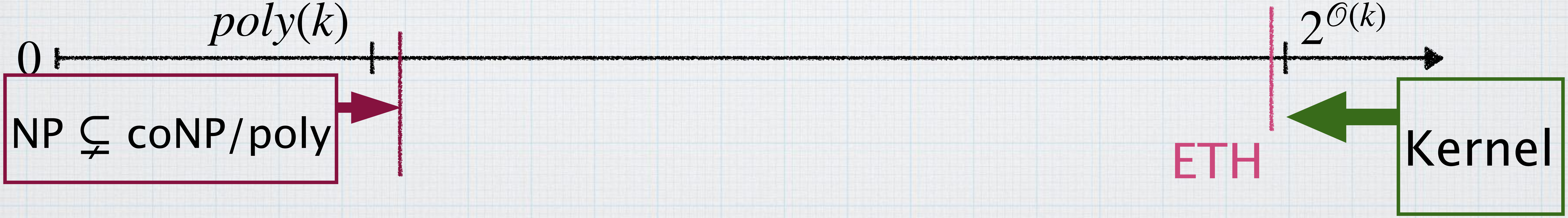
Our results

Our results

Thm. Locating Dominating Set and Test Cover
— admit kernel with $2^{\mathcal{O}(k)}$ and $2^{2^{\mathcal{O}(k)}}$ vertices, respectively, but
— do not admit a kernel with $2^{o(k)}$ and $2^{2^{o(k)}}$ vertices, respectively, unless the ETH fails.

Our results

Thm. Locating Dominating Set and Test Cover
— admit kernel with $2^{\mathcal{O}(k)}$ and $2^{2^{\mathcal{O}(k)}}$ vertices, respectively, but
— do not admit a kernel with $2^{o(k)}$ and $2^{2^{o(k)}}$ vertices, respectively, unless the ETH fails.

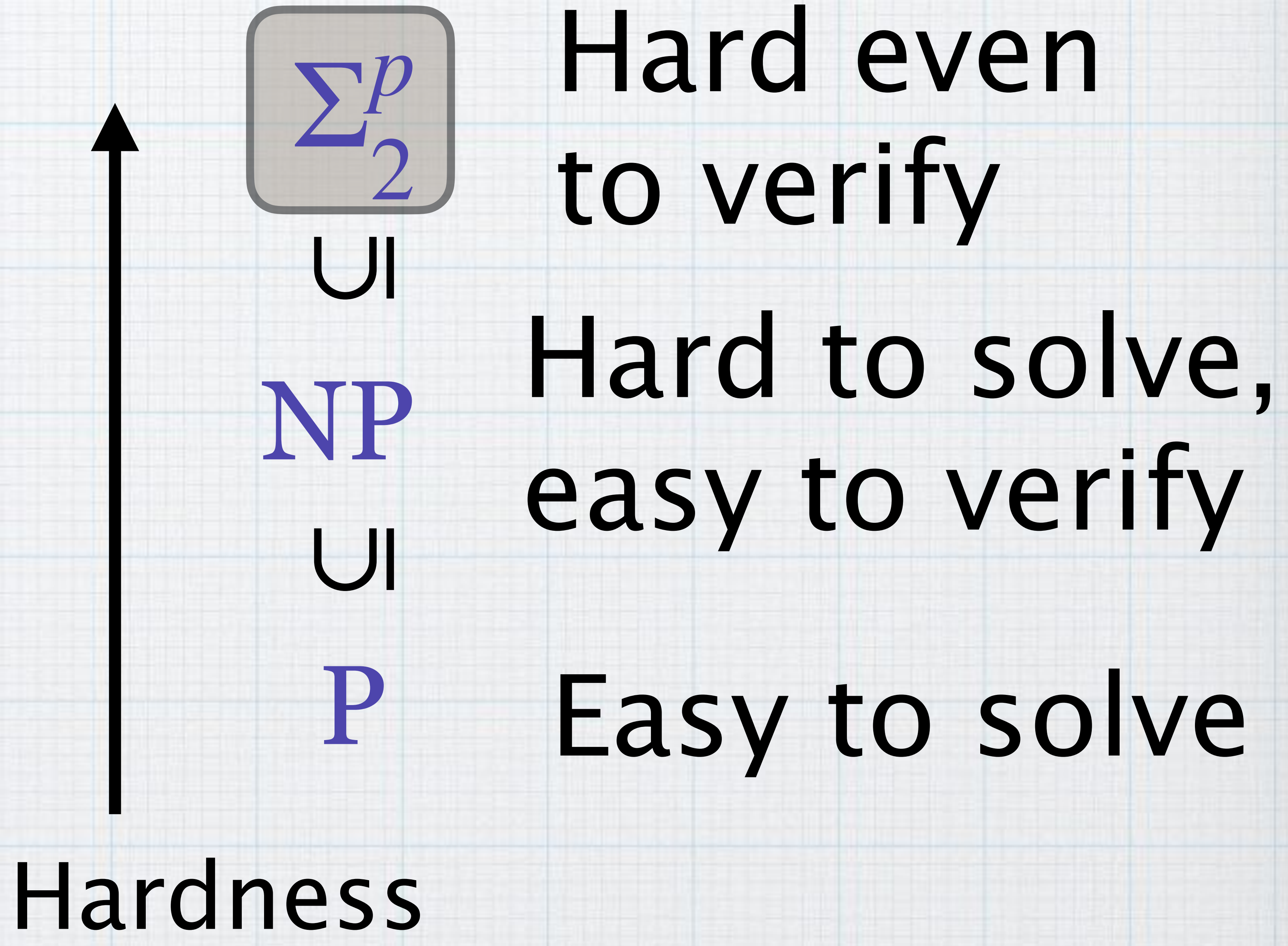


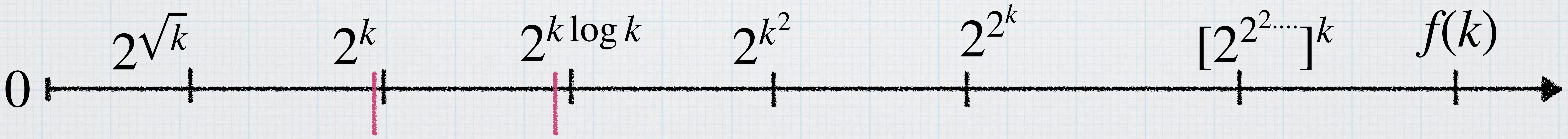
Thm [Marx, Mitsou (ICALP'16)]. The λ–Choosability problem

— admits $2^{2^{\mathcal{O}(\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$–time algo, but

— does not admit $2^{2^{o(\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$ algo unless the ETH fails.

Treewidth (tw): closeness to trees





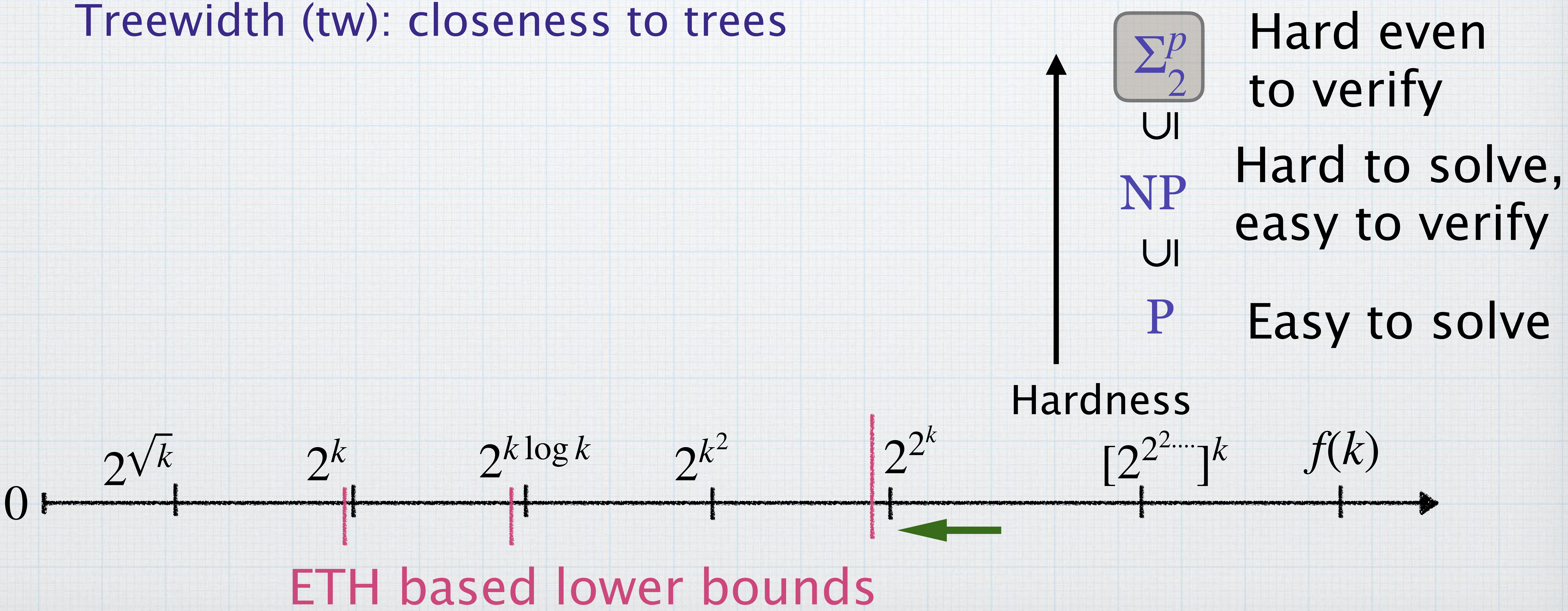
ETH based lower bounds

Thm [Marx, Mitsou (ICALP'16)]. The λ–Choosability problem

— admits $2^{2^{\mathcal{O}(\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$–time algo, but

— does not admit $2^{2^{o(\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$ algo unless the ETH fails.

Treewidth (tw): closeness to trees



Thm [Marx, Mitsou (ICALP'16)]. The λ–Choosability problem
— admits $2^{2^{\mathcal{O}(\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$–time algo, but
— does not admit $2^{2^{o(\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$ algo unless the ETH fails.

Treewidth (tw): closeness to trees

"… the problem definition is an underlying reason for being in the higher levels of the polynomial hierarchy and for requiring unusually large dependence on treewidth."

Σ_2^p — Hard even to verify

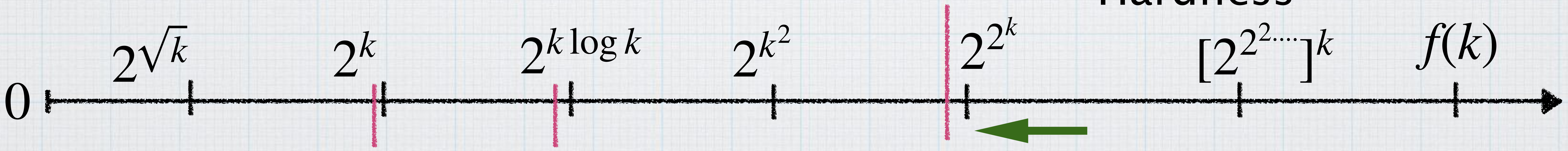
$\cup|$

NP — Hard to solve, easy to verify

$\cup|$

P — Easy to solve

Hardness



ETH based lower bounds

Thm [Marx, Mitsou (ICALP'16)]. The λ-Choosability problem
— admits $2^{2^{\mathcal{O}(\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$-time algo, but
— does not admit $2^{2^{o(\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$ algo unless the ETH fails.

Treewidth (tw): closeness to trees

"… the problem definition is an underlying reason for being in the higher levels of the polynomial hierarchy and for requiring unusually large dependence on treewidth."

Hardness	
Σ_2^p	Hard even to verify
$\cup\!\!\mid$	
NP	Hard to solve, easy to verify
$\cup\!\!\mid$	
P	Easy to solve

Our result (in ICALP'24): Not necessary to go higher up in the polynomial hierarchy to achieve double-exponential lower bounds.

Problems in NP can admit double-exponential lower bounds when parameterized by treewidth and vertex cover

by Foucaud, Galby, Khazaliya, Li, Mc Inerney, Sharma, Tale (2023)

Thm [FGKLIST (ICALP'24)]. The Metric Dimension problem on bounded diameter graphs

— admits $2^{2^{\mathcal{O}(\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$–time algo, but

— does not admit $2^{2^{o(\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$ algo

unless the ETH fails.

Σ_2^p Hard even to verify

$\cup\!\!|$

NP Hard to solve, easy to verify

$\cup\!\!|$

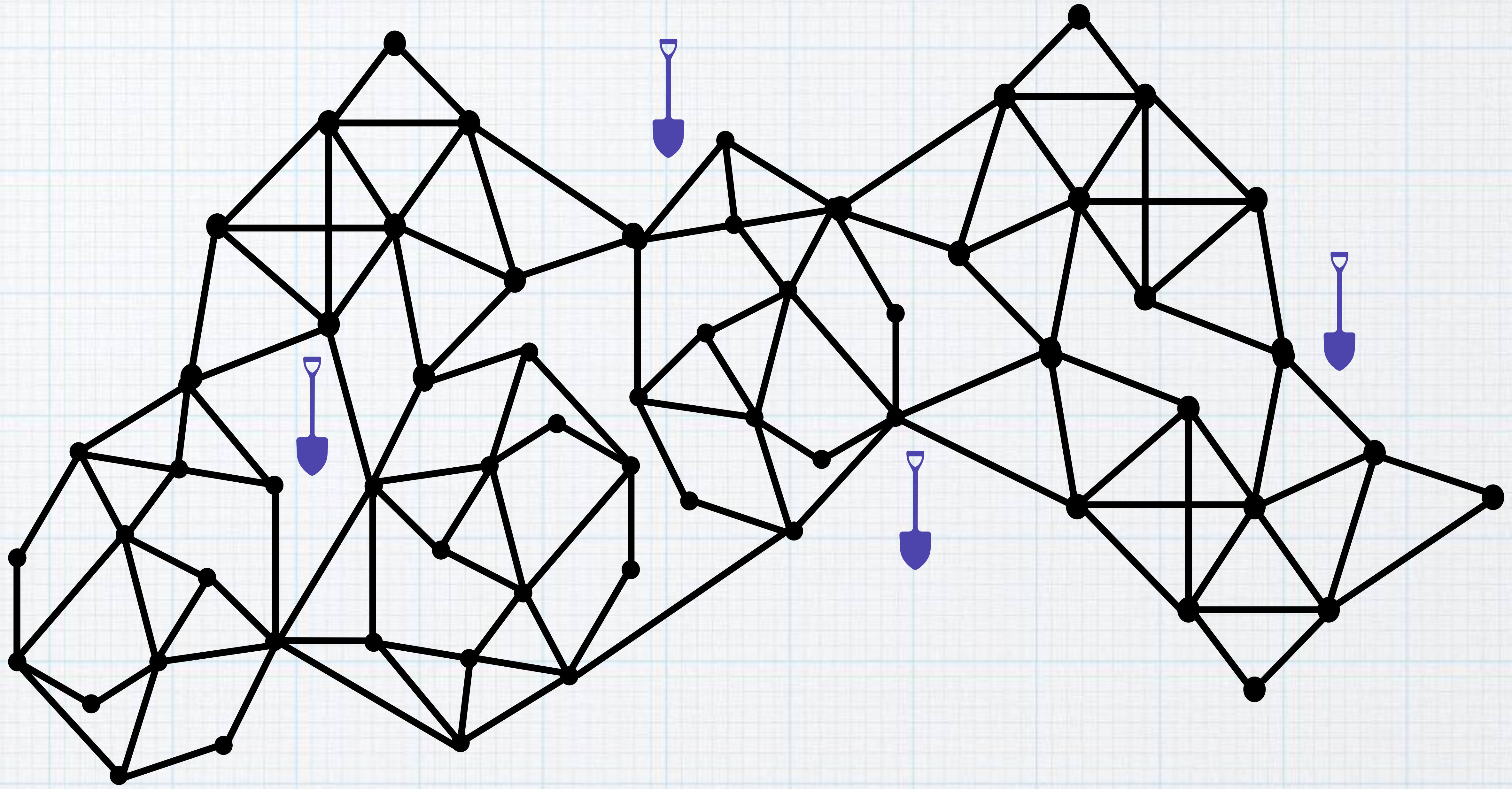
P Easy to solve

Hardness

– Metric Dimension (Treasure Hunt)

Digging at a vertex reveals its distance from the treasure.

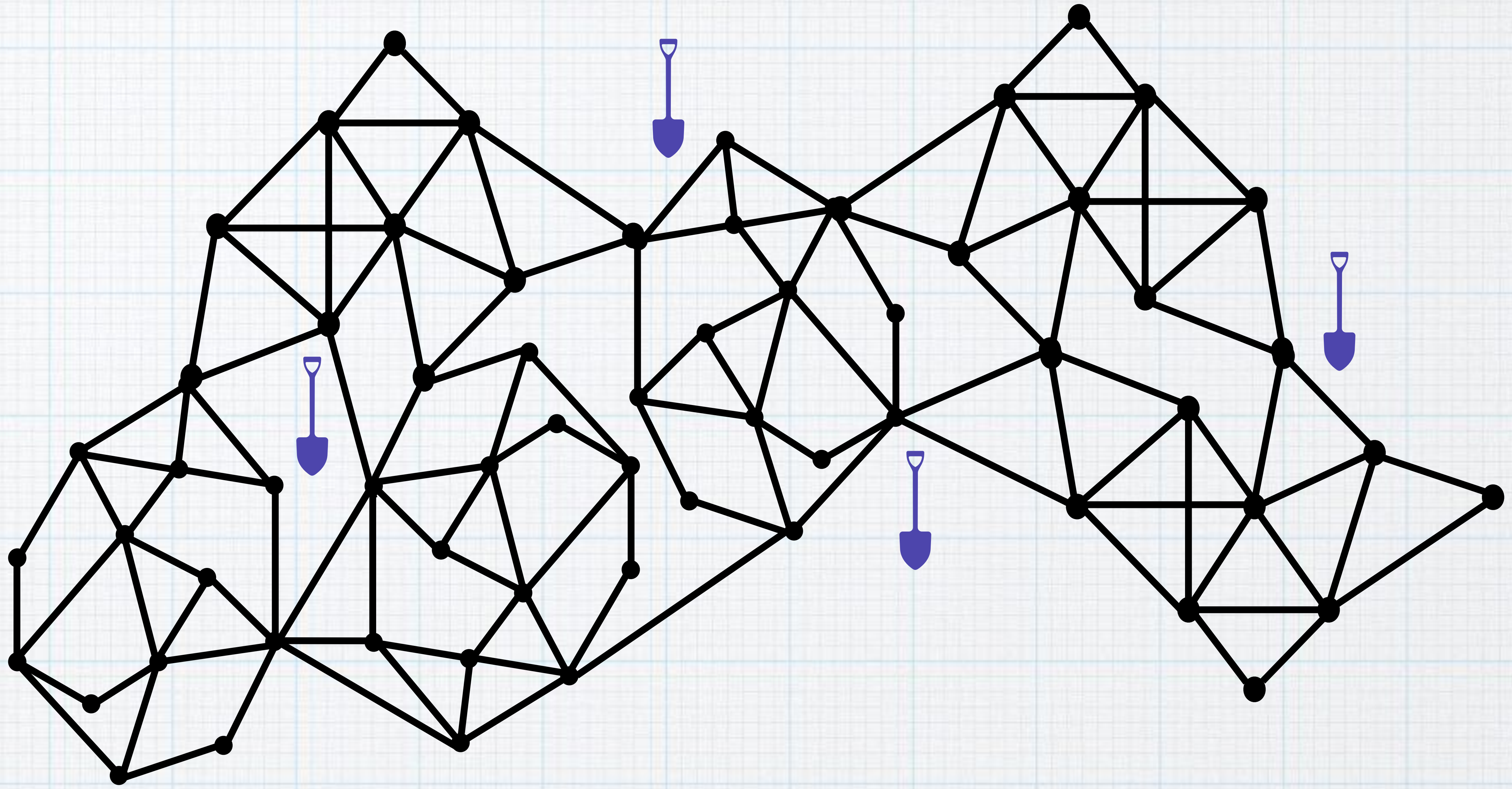
Can we dig at k vertices to locate the treasure?



– Metric Dimension (Treasure Hunt)

Digging at a vertex reveals its distance from the treasure.

Can we dig at k vertices to locate the treasure?

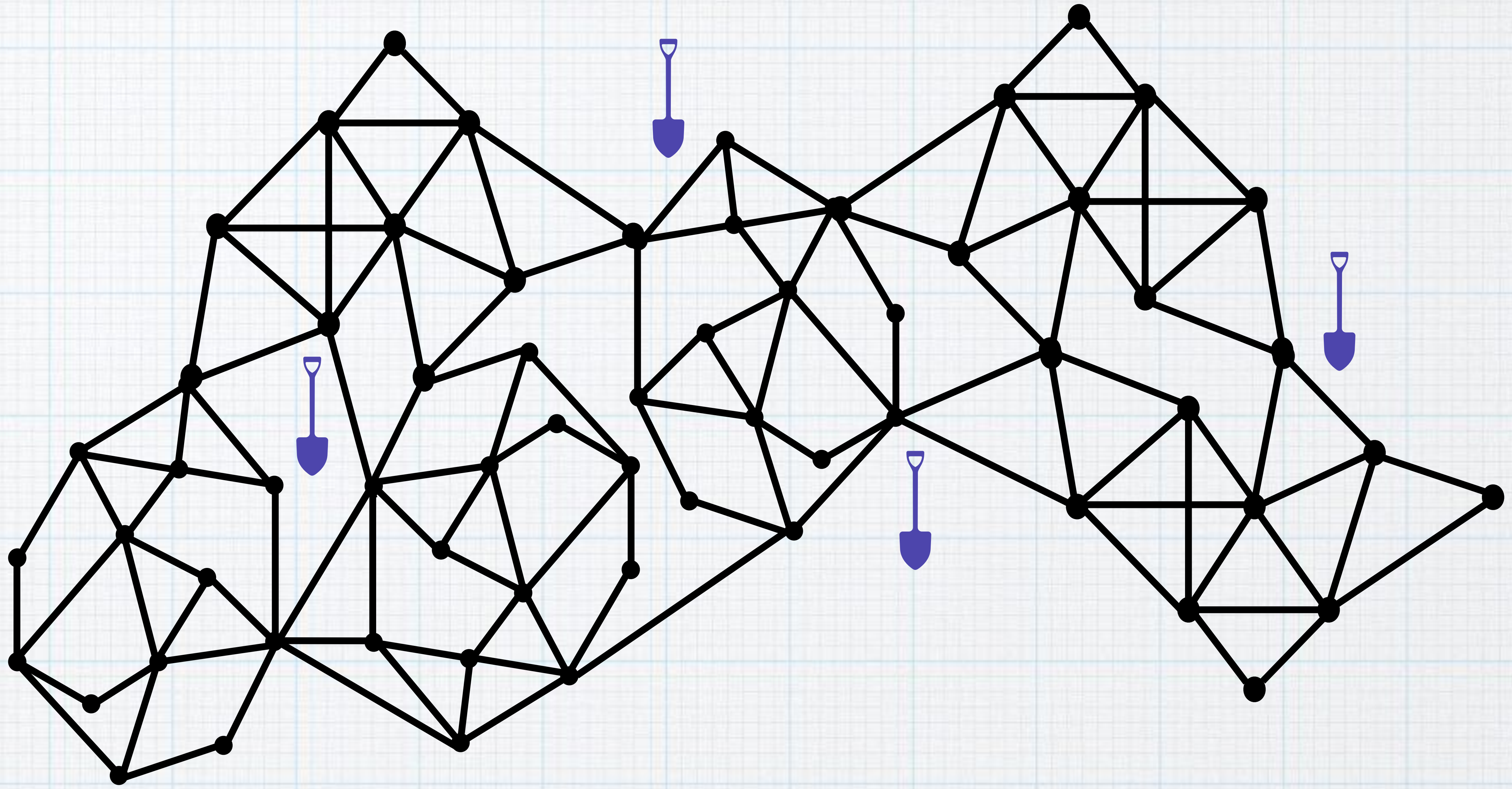


– Effect of a solution vertex is global in `metric graph problems’.

– Metric Dimension (Treasure Hunt)

Digging at a vertex reveals its distance from the treasure.

Can we dig at k vertices to locate the treasure?



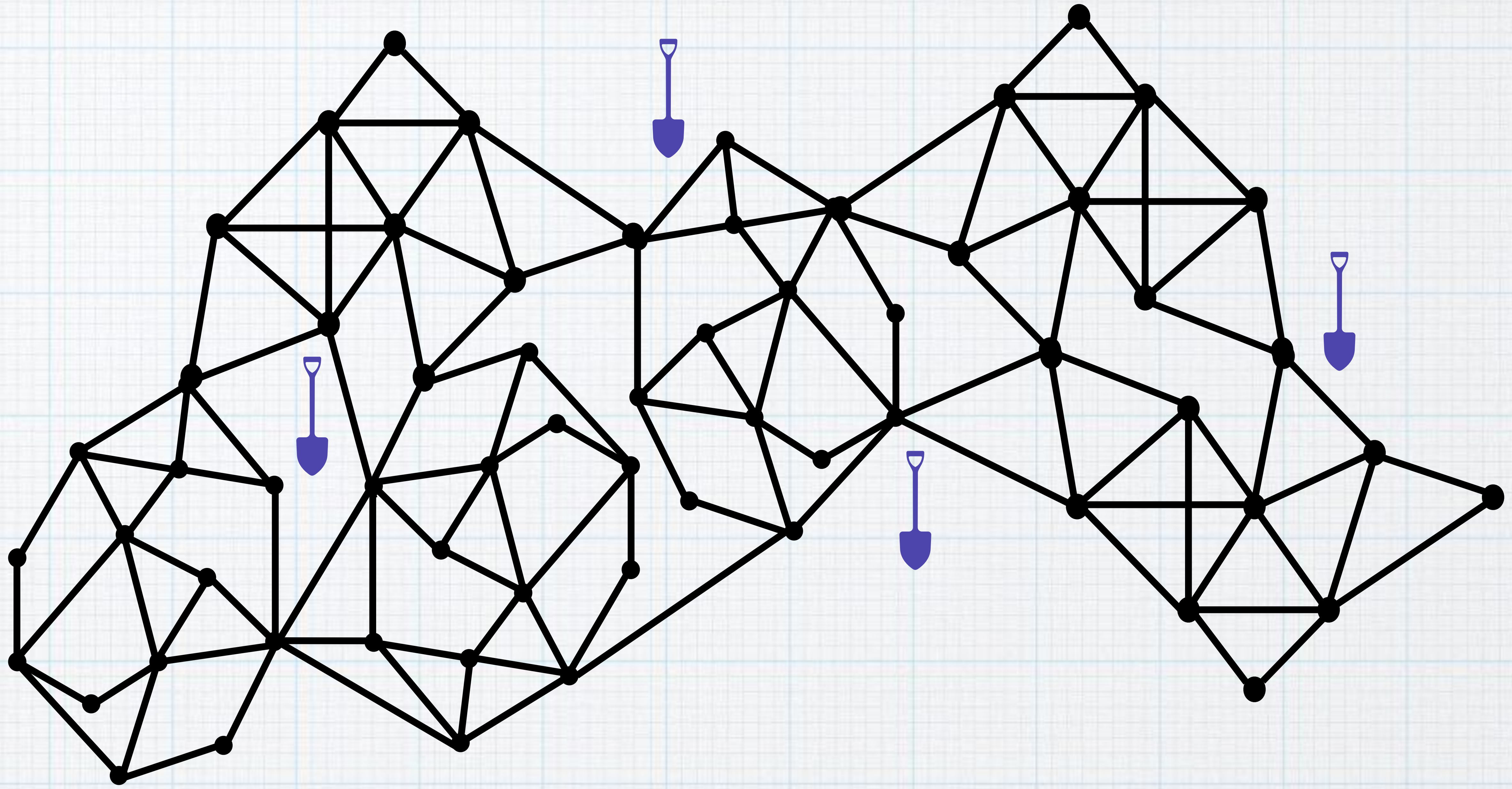
– Effect of a solution vertex is global in `metric graph problems'.

– Effect of a solution vertex is local in `identification problems'.

– Metric Dimension (Treasure Hunt)

Digging at a vertex reveals its distance from the treasure.

Can we dig at k vertices to locate the treasure?

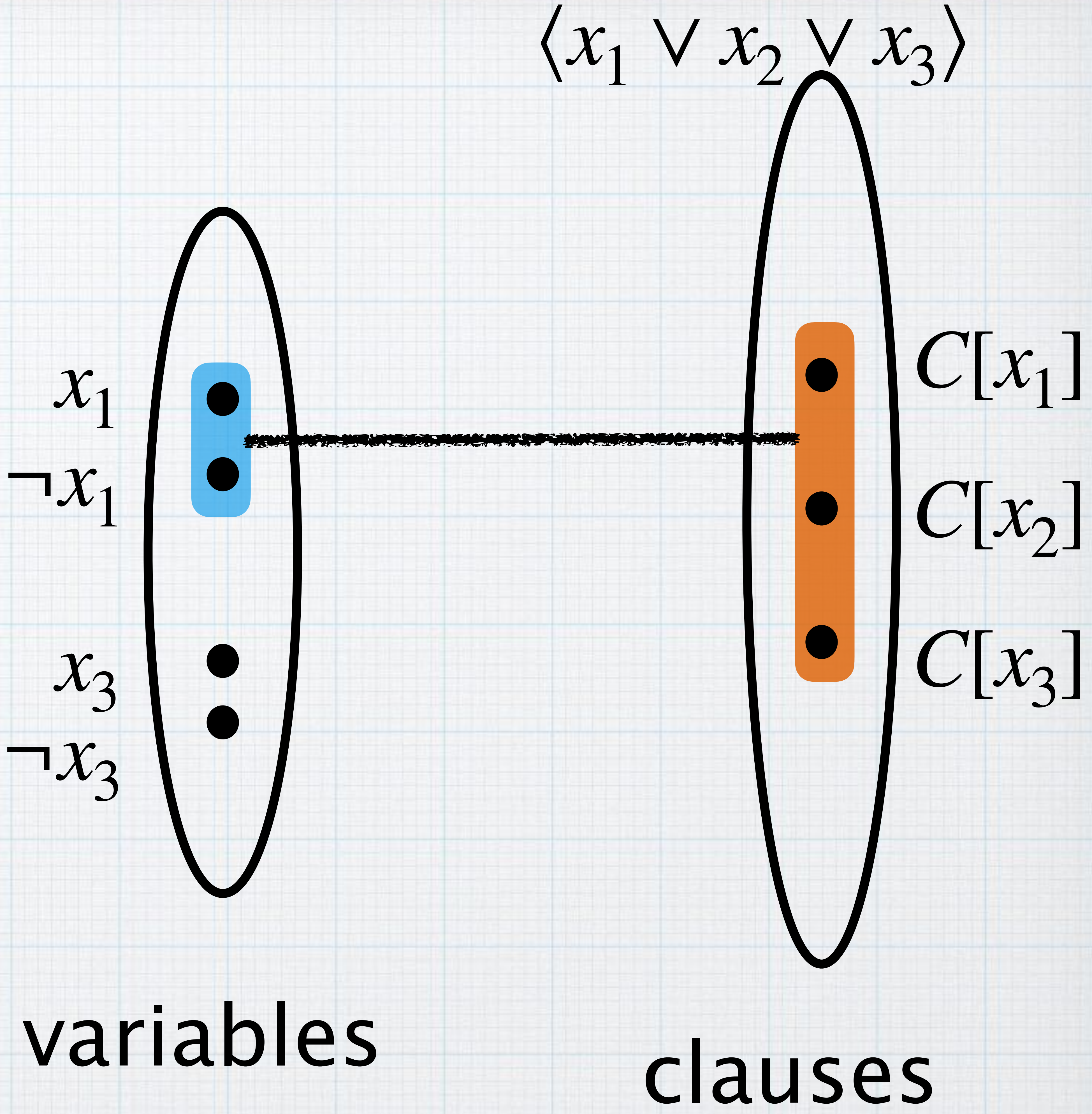


– Effect of a solution vertex is global in `metric graph problems'.

– Effect of a solution vertex is local in `identification problems'.

It is not necessary to use `metric graph problems' to use double exponential lower bound.
We can obtain similar results with `identification problems'.

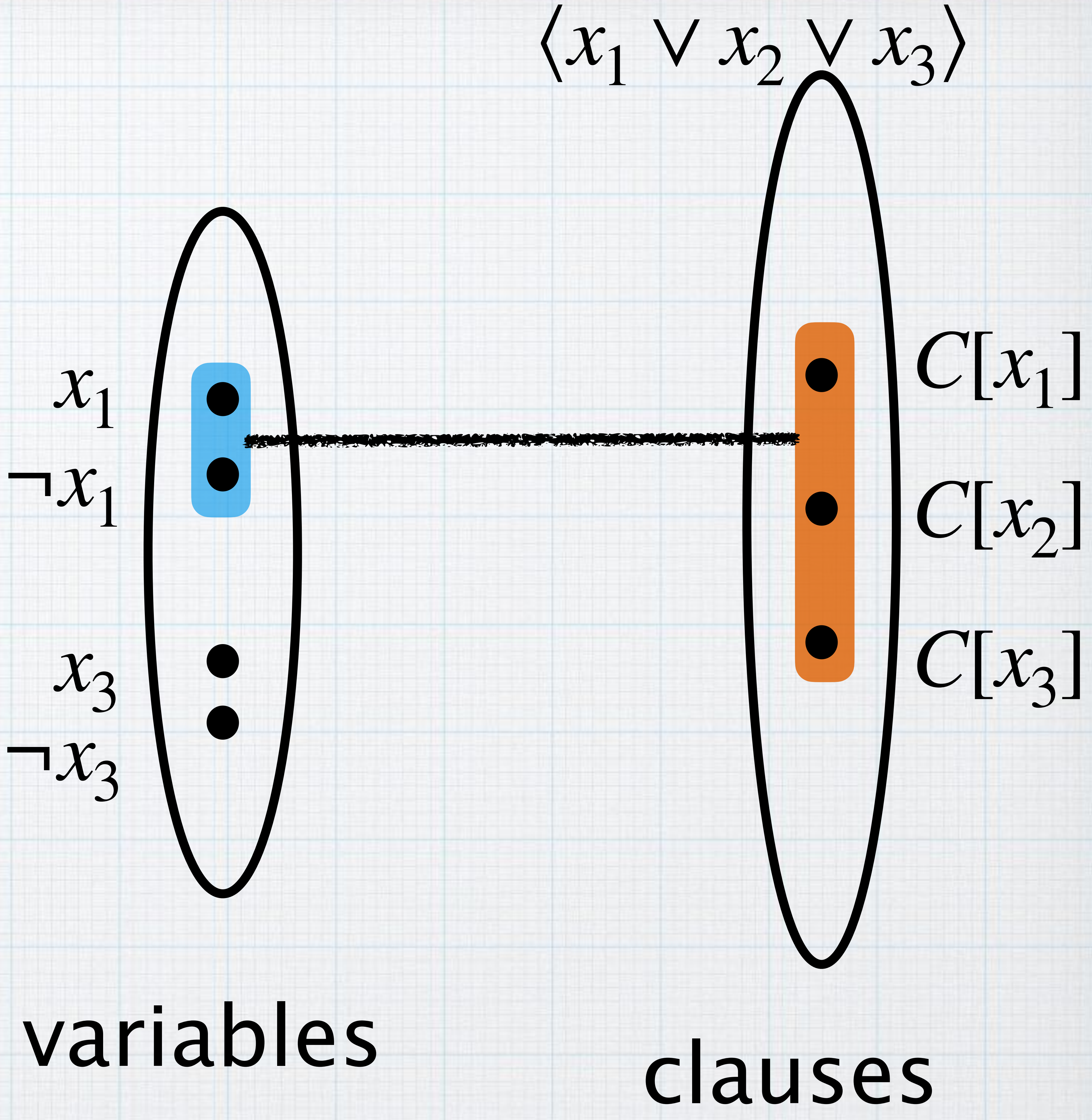
3-SAT to Loc-Dom-Set



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

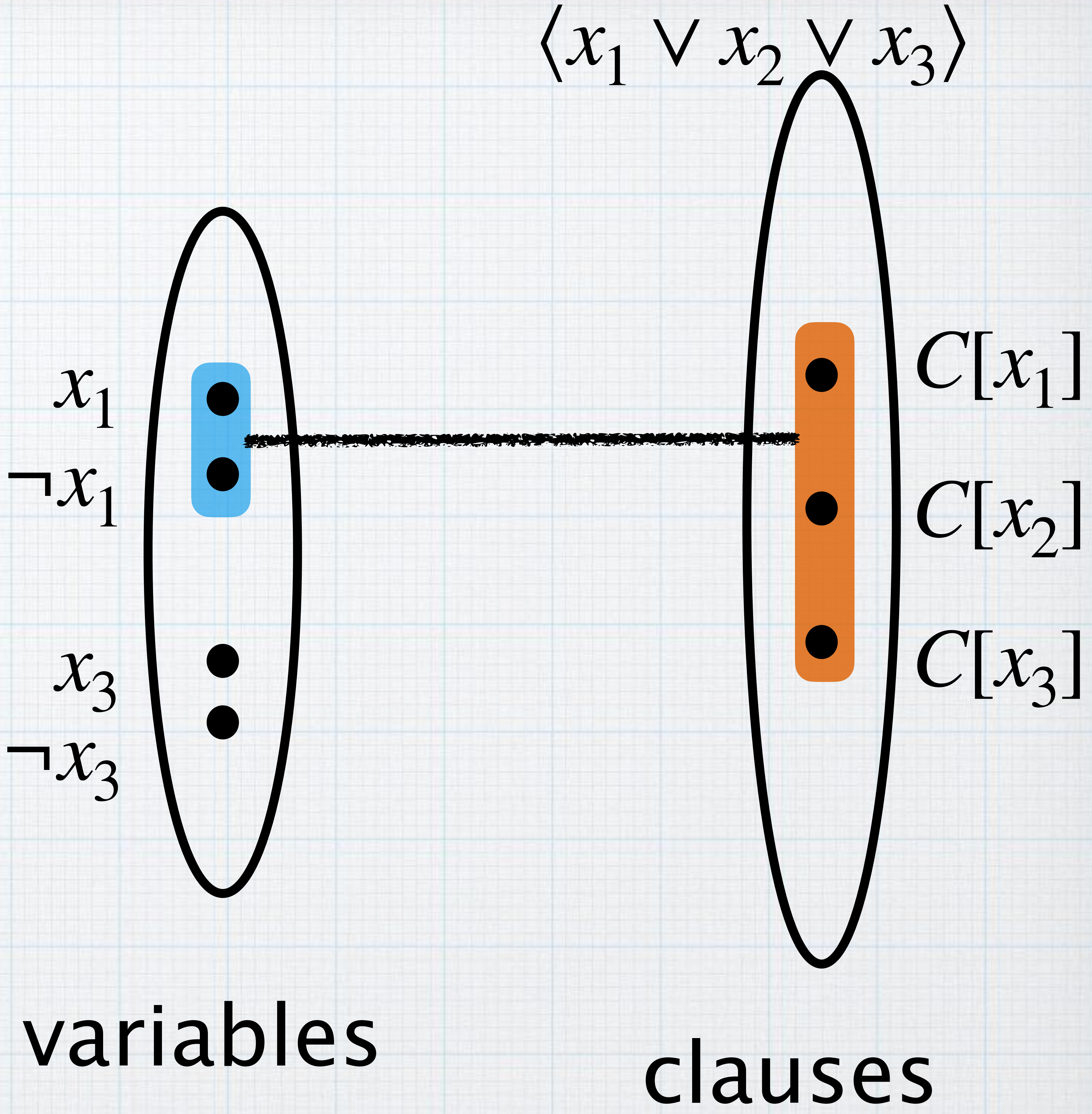
- n-variable formula → graph with tw $= \mathcal{O}(\log(n))$



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

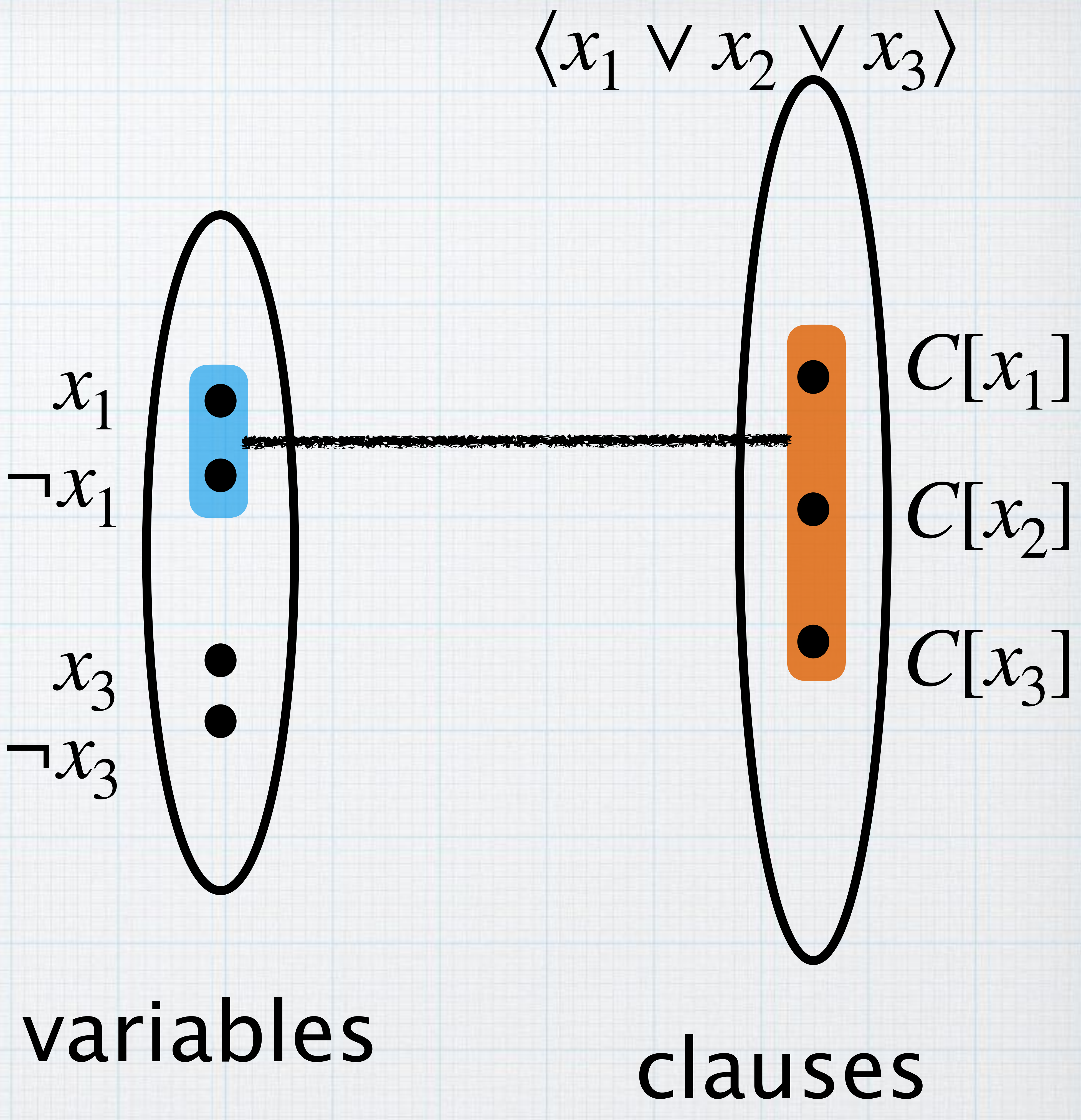
- n-variable formula → graph with tw $= \mathcal{O}(\log(n))$
- Add pair of vertices for each variable



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

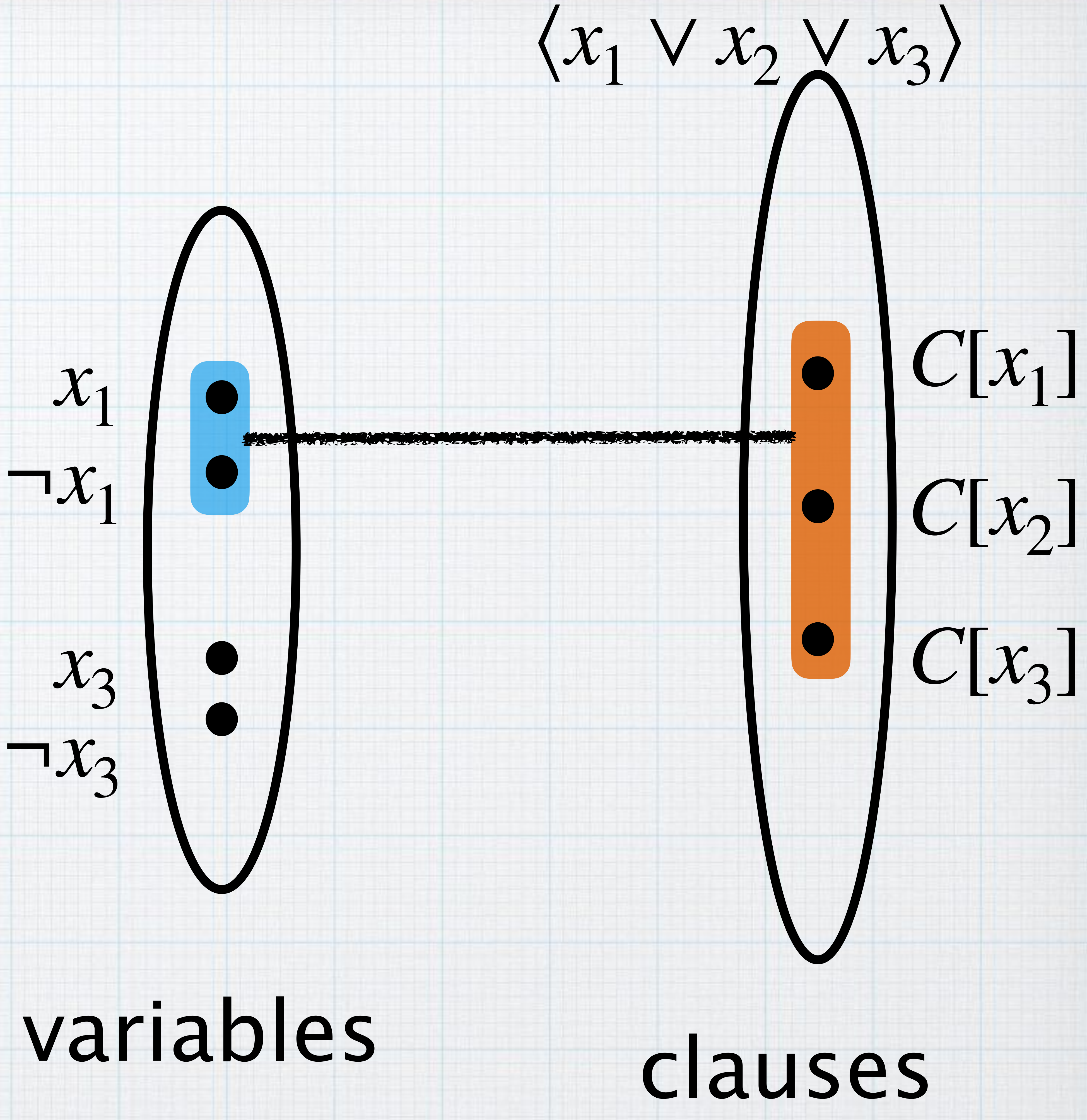
- n-variable formula → graph with tw $= \mathcal{O}(\log(n))$
- Add pair of vertices for each variable
- Add 3 vertices for each clause



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

- n-variable formula → graph with tw $= \mathcal{O}(\log(n))$
- Add pair of vertices for each variable
- Add 3 vertices for each clause
- For every variable, exactly one vertex is in solution.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3–SAT to Loc–Dom–Set

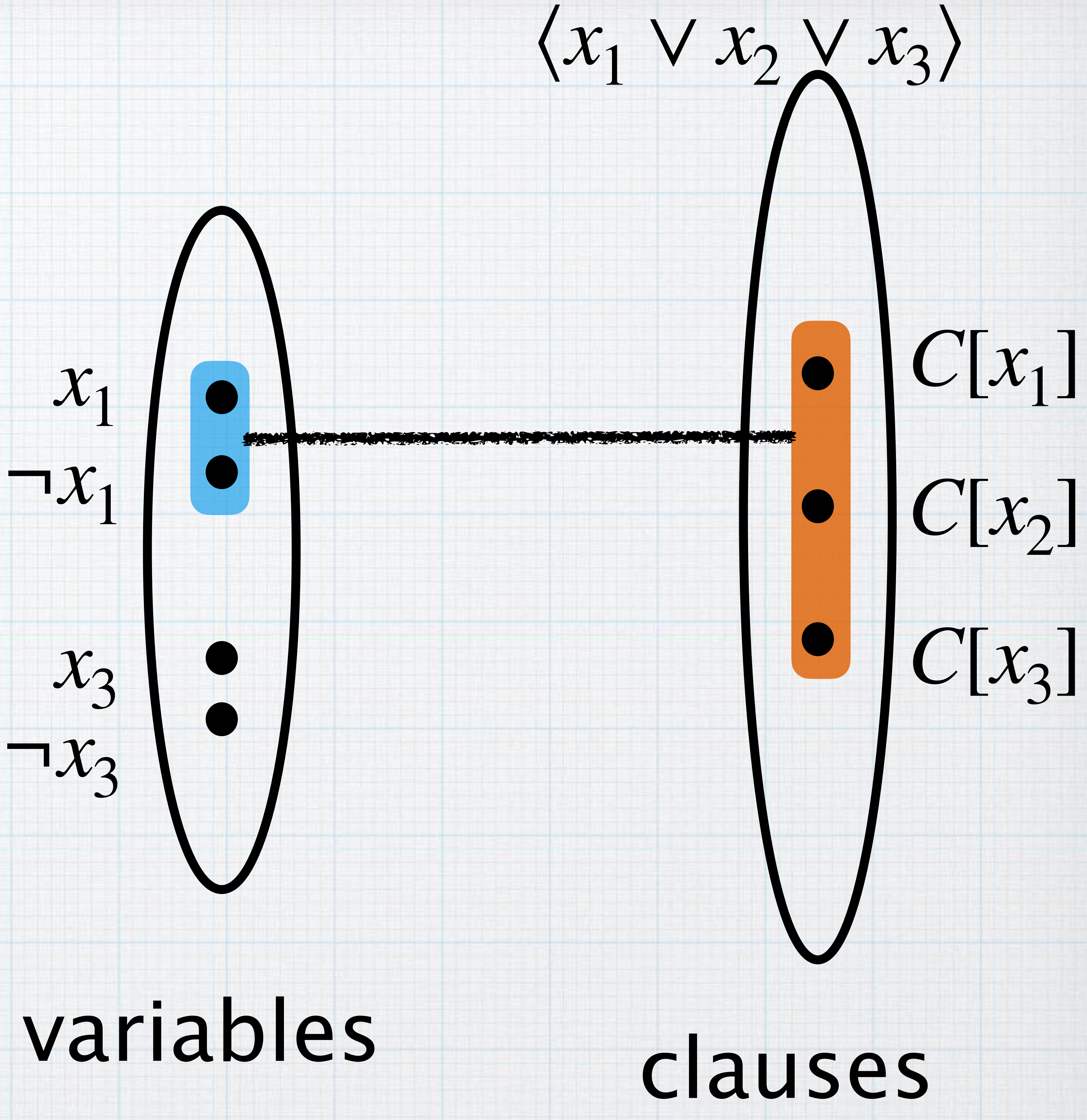
– n–variable formula → graph with $\text{tw} = \mathcal{O}(\log(n))$

– Add pair of vertices for each variable

– Add 3 vertices for each clause

– For every variable, exactly one vertex is in solution.

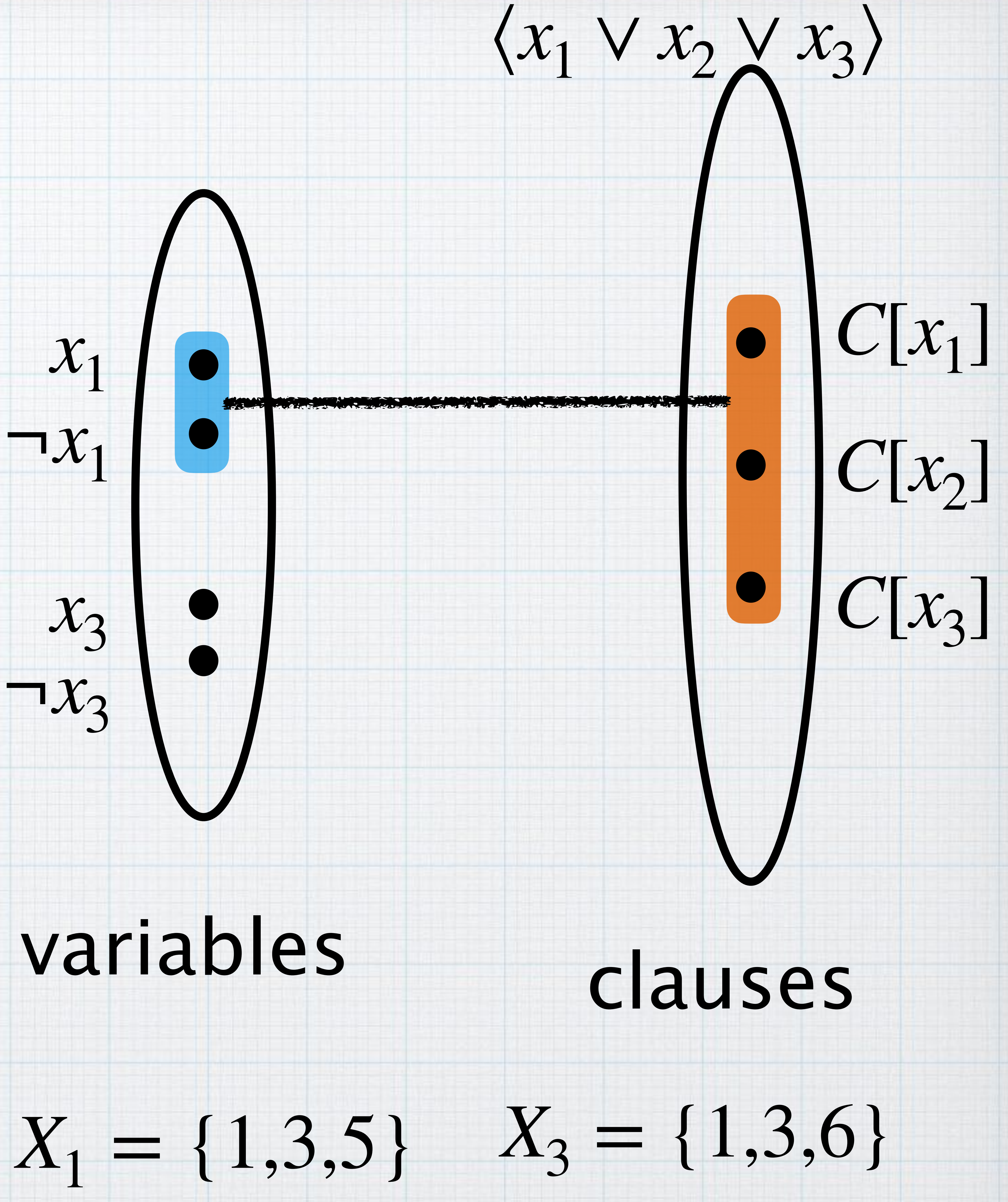
– For every clause, exactly two vertices are is in solution.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

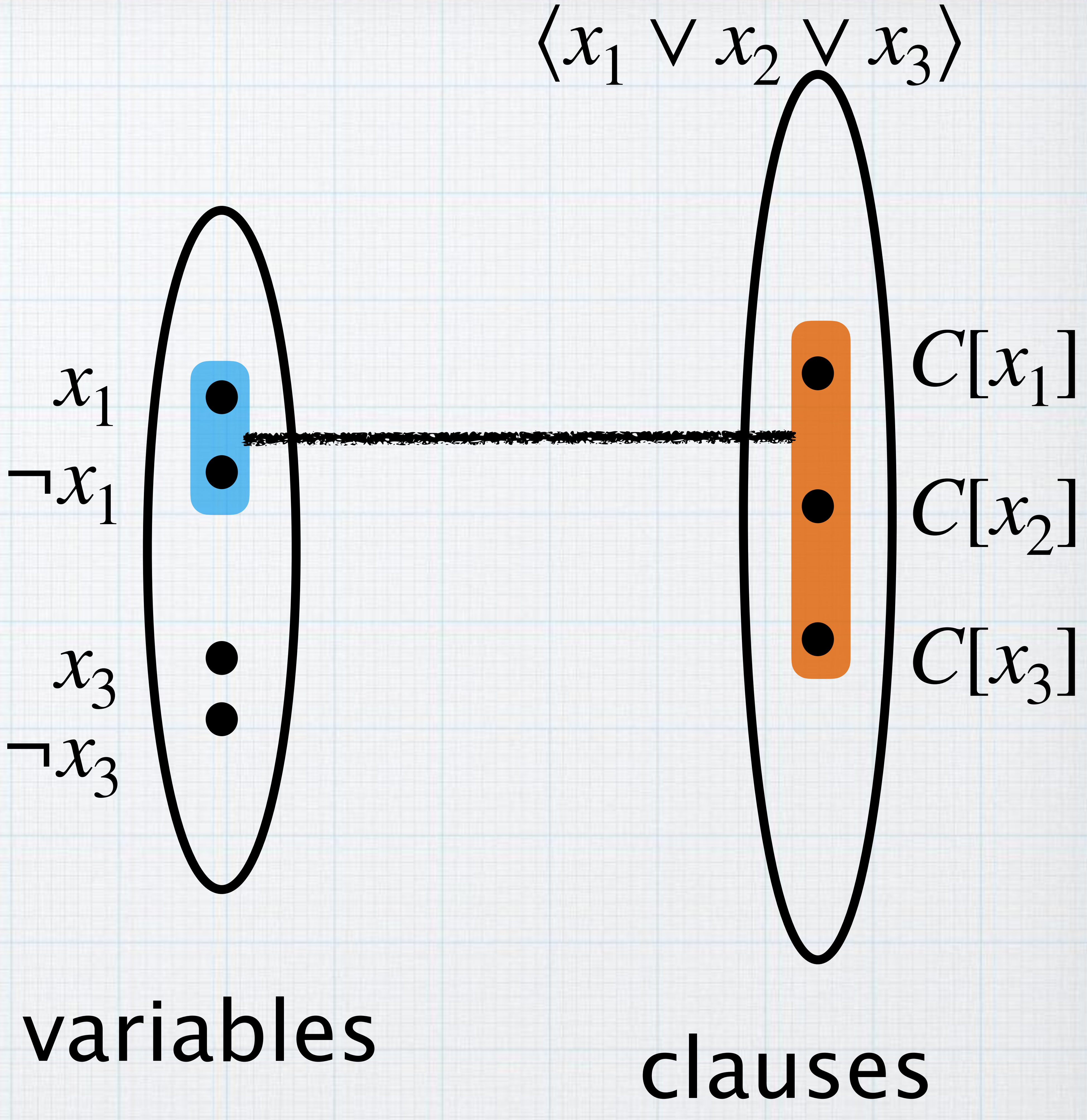
3-SAT to Loc-Dom-Set

- n-variable formula $\rightarrow$ graph with tw $= \mathcal{O}(\log(n))$
- Add pair of vertices for each variable
- Add 3 vertices for each clause
- For every variable, exactly one vertex is in solution.
- For every clause, exactly two vertices are is in solution.
- Selection in variable side should reflect selection on clause side and vice-versa.



3-SAT to Loc-Dom-Set

- n-variable formula → graph with tw $= \mathcal{O}(\log(n))$
- Add pair of vertices for each variable
- Add 3 vertices for each clause
- For every variable, exactly one vertex is in solution.
- For every clause, exactly two vertices are is in solution.
- Selection in variable side should reflect selection on clause side and vice-versa.
- tw $= \mathcal{O}(n)$ (no better bound)



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

3-SAT to Loc-Dom-Set

Sperner Family: Collection $\mathscr{F}$ of subsets of $[2p]$ such that no set in $\mathscr{F}$ is contained in any other set in $\mathscr{F}$.

3-SAT to Loc-Dom-Set

Sperner Family: Collection $\mathscr{F}$ of subsets of $[2p]$ such that no set in $\mathscr{F}$ is contained in any other set in $\mathscr{F}$.

Ex: $\mathscr{F}$ — each set contains exactly p elements.

3-SAT to Loc-Dom-Set

Sperner Family: Collection $\mathscr{F}$ of subsets of $[2p]$ such that no set in $\mathscr{F}$ is contained in any other set in $\mathscr{F}$.

Ex: $\mathscr{F}$ — each set contains exactly p elements.

For $p = 3$, $X_1 = \{1,3,5\}, X_3 = \{1,3,6\} \in \mathscr{F}$

3-SAT to Loc-Dom-Set

Sperner Family: Collection $\mathscr{F}$ of subsets of $[2p]$ such that no set in $\mathscr{F}$ is contained in any other set in $\mathscr{F}$.

Ex: $\mathscr{F}$ — each set contains exactly p elements.

For $p = 3$, $X_1 = \{1,3,5\}, X_3 = \{1,3,6\} \in \mathscr{F}$

We get $\mathscr{F}$ which is of size 2^p

3-SAT to Loc-Dom-Set

Sperner Family: Collection $\mathscr{F}$ of subsets of $[2p]$ such that no set in $\mathscr{F}$ is contained in any other set in $\mathscr{F}$.

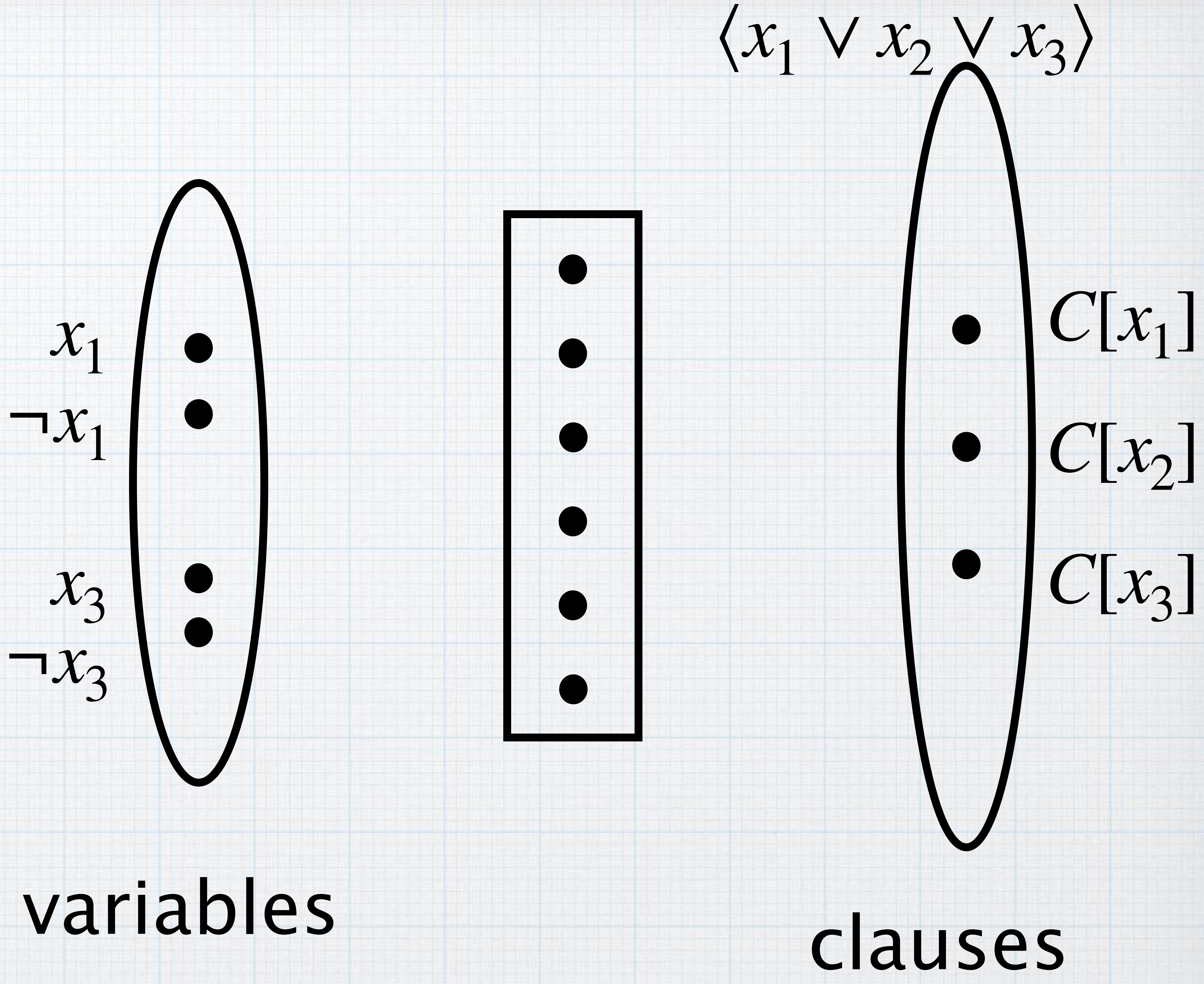
Ex: $\mathscr{F}$ — each set contains exactly p elements.

For $p = 3$, $X_1 = \{1,3,5\}, X_3 = \{1,3,6\} \in \mathscr{F}$

We get $\mathscr{F}$ which is of size 2^p

For $p = \log(n)$, $\mathscr{F}$ is of size n, i.e. unique set for each variable.

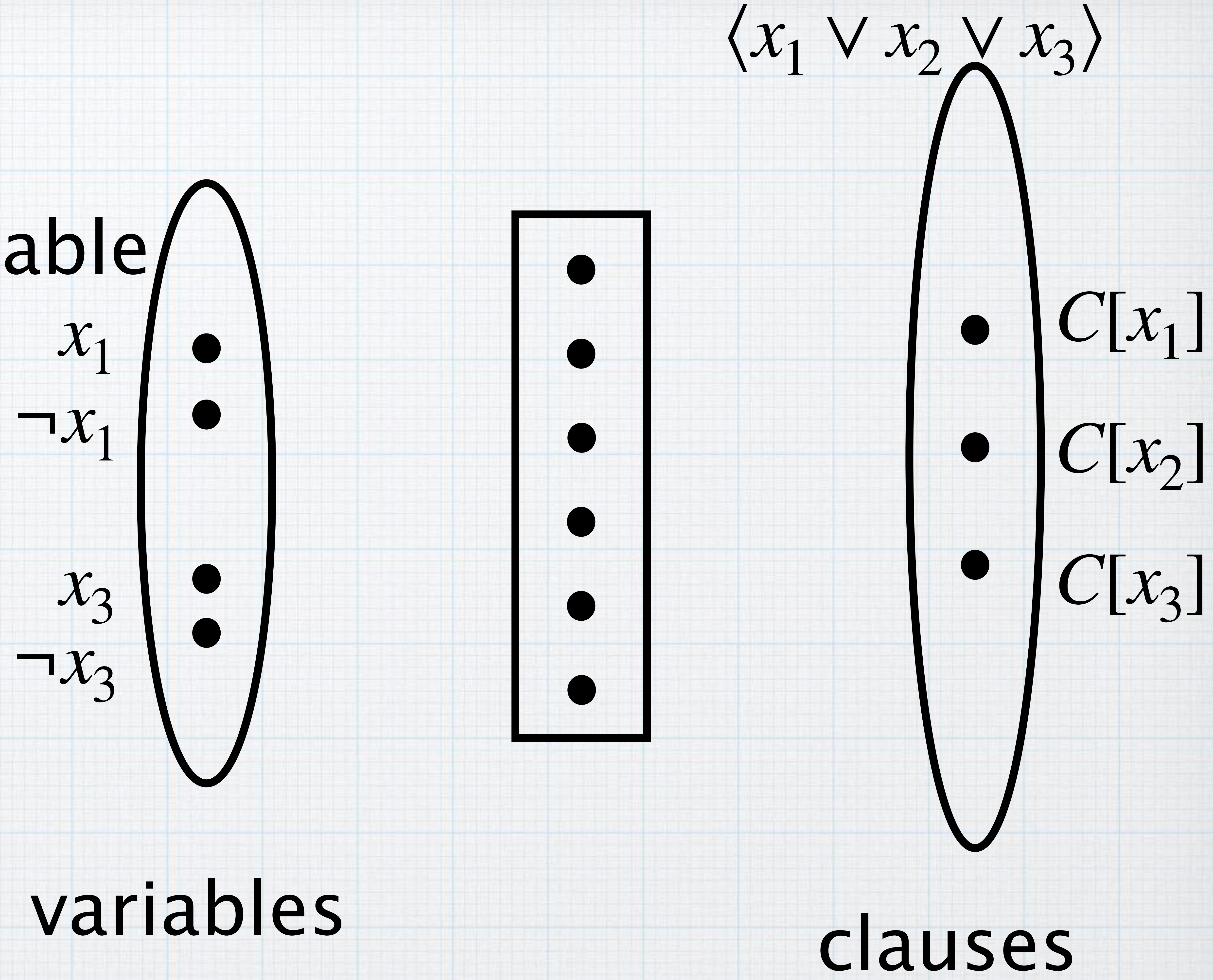
3-SAT to Loc-Dom-Set



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

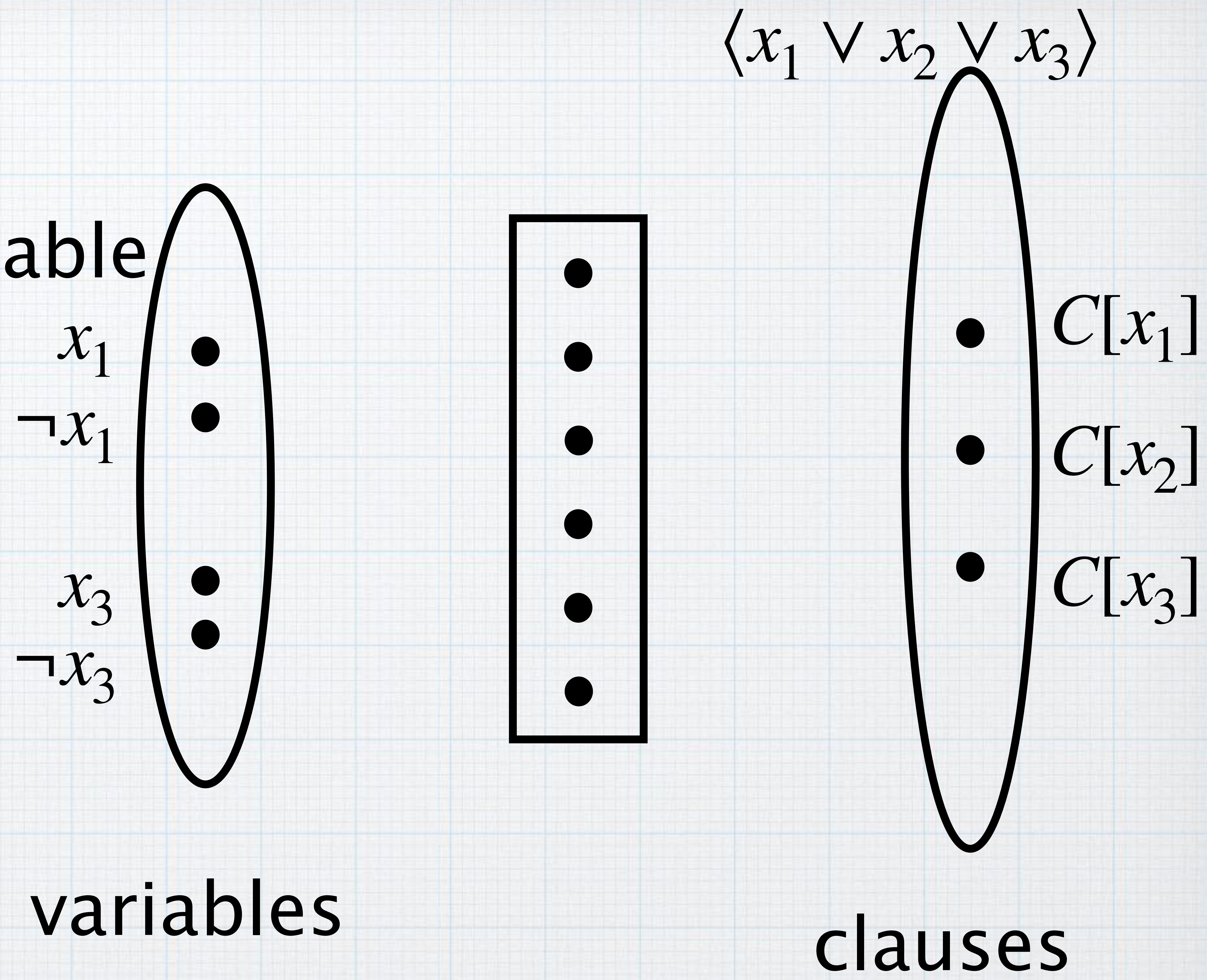
– Add pair of vertices for each variable



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

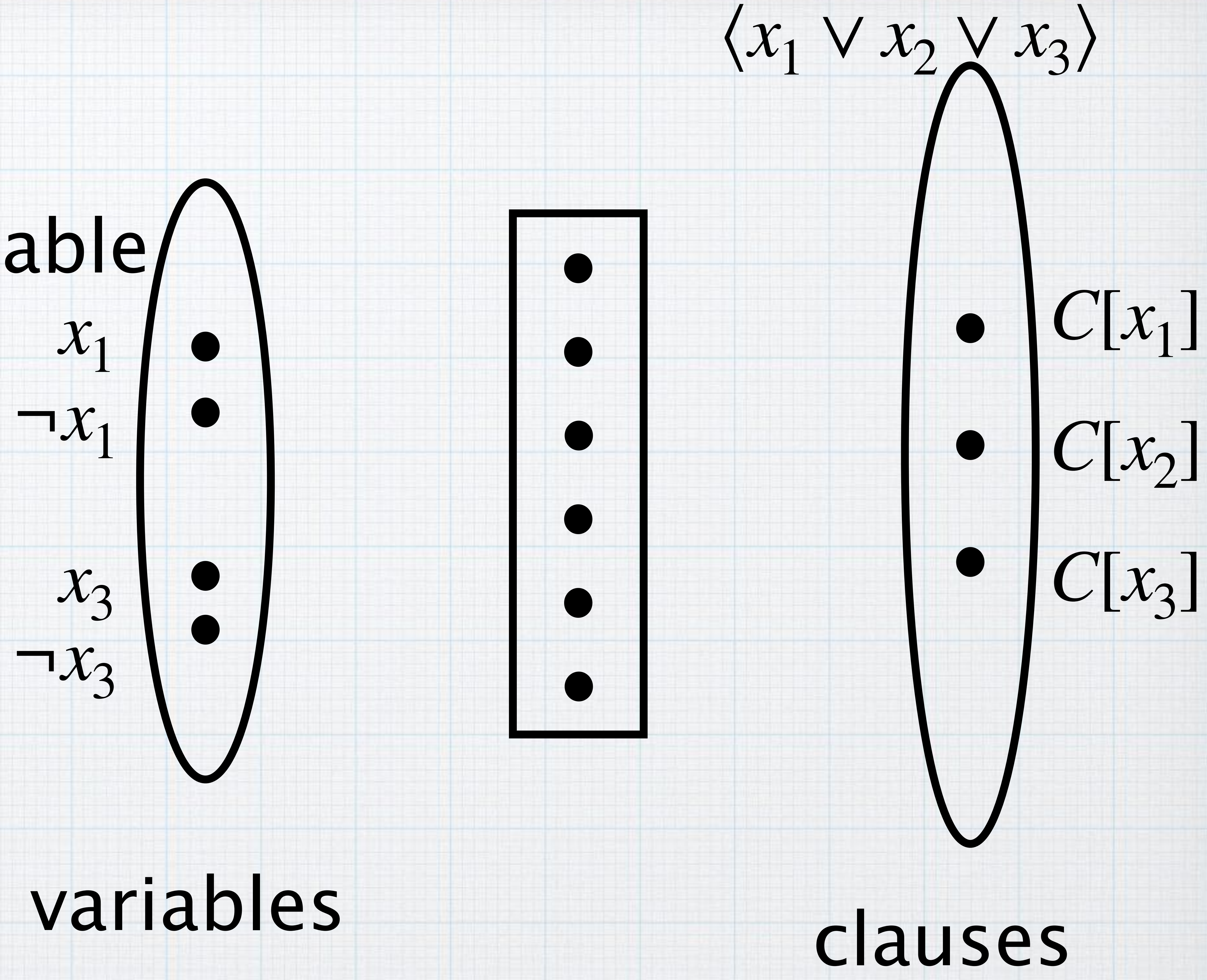
- Add pair of vertices for each variable
- Add 3 vertices for each clause



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

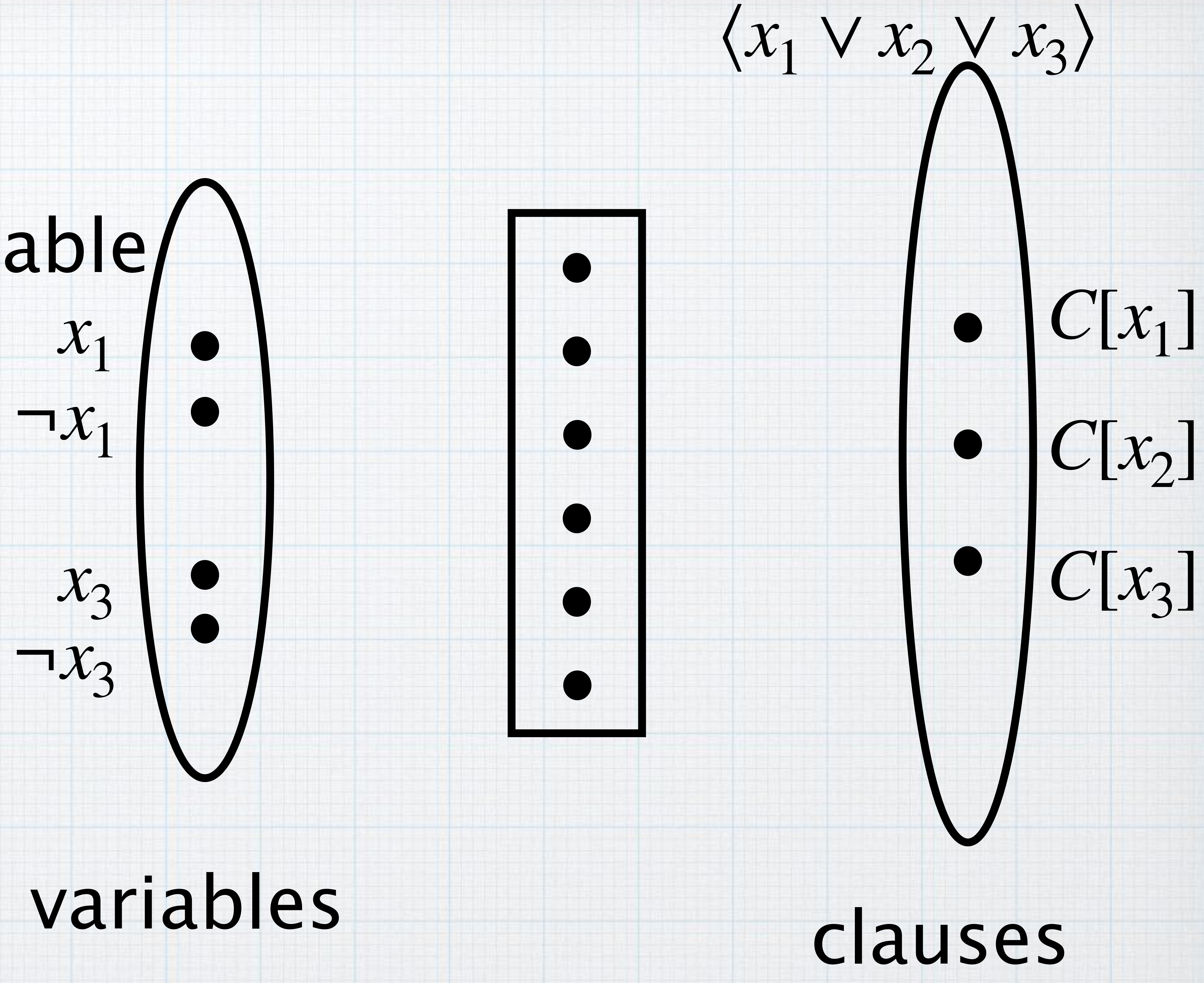
- Add pair of vertices for each variable
- Add 3 vertices for each clause
- Add $\mathcal{O}(\log(n))$ many vertices to facilitate the connections



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

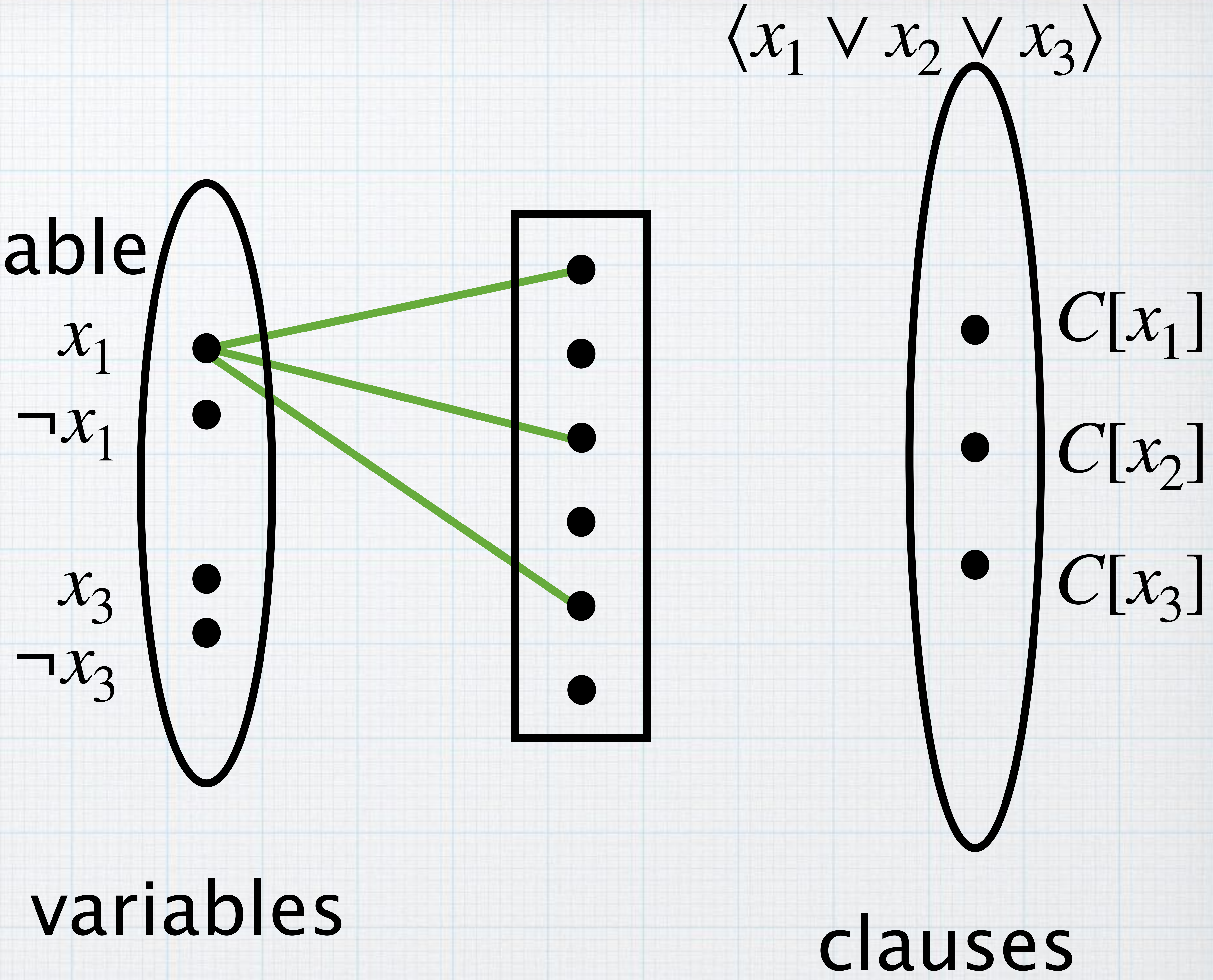
- Add pair of vertices for each variable
- Add 3 vertices for each clause
- Add $\mathcal{O}(\log(n))$ many vertices to facilitate the connections
- Variable vert. are connected to corresponding set element.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

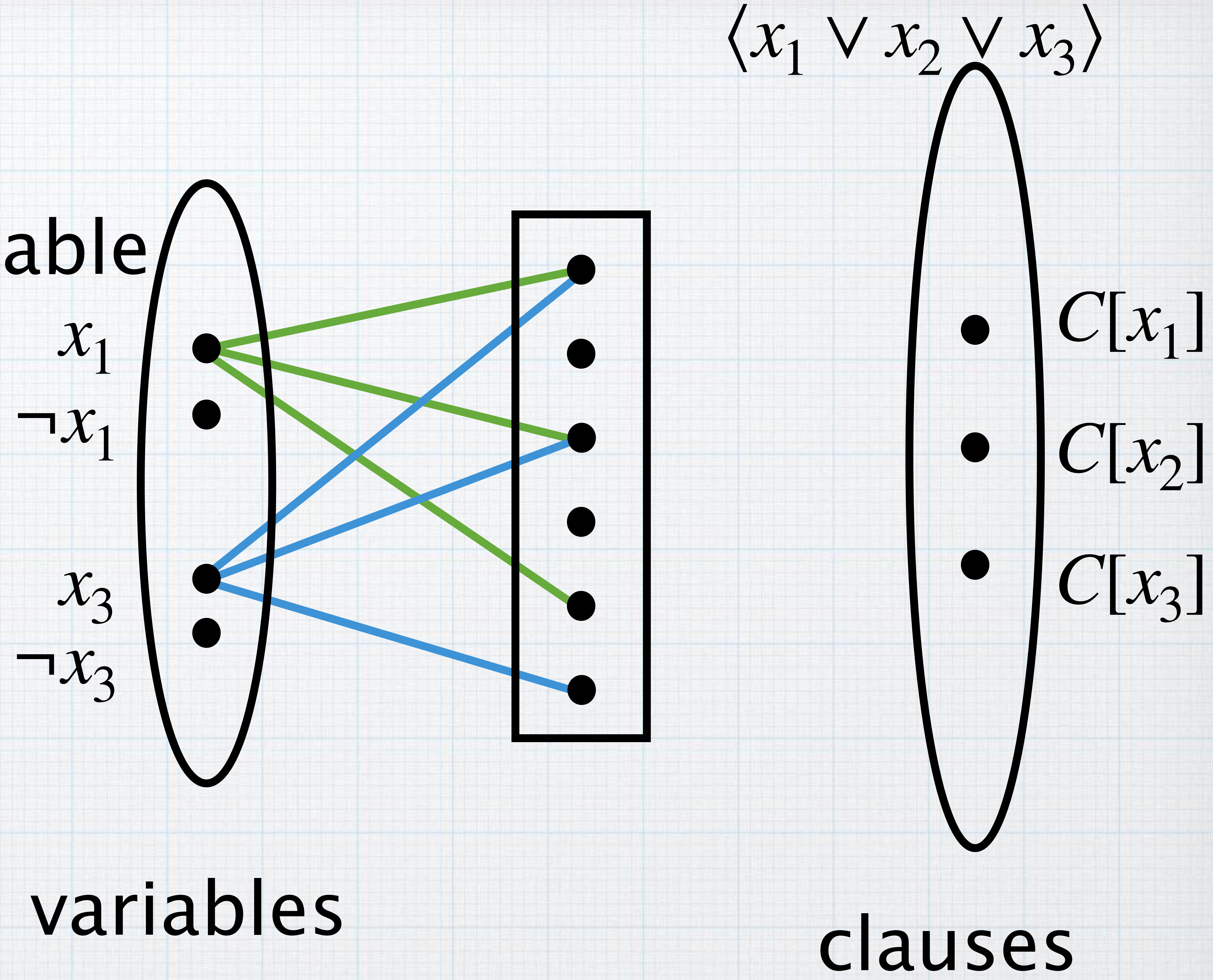
- Add pair of vertices for each variable
- Add 3 vertices for each clause
- Add $\mathcal{O}(\log(n))$ many vertices to facilitate the connections
- Variable vert. are connected to corresponding set element.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

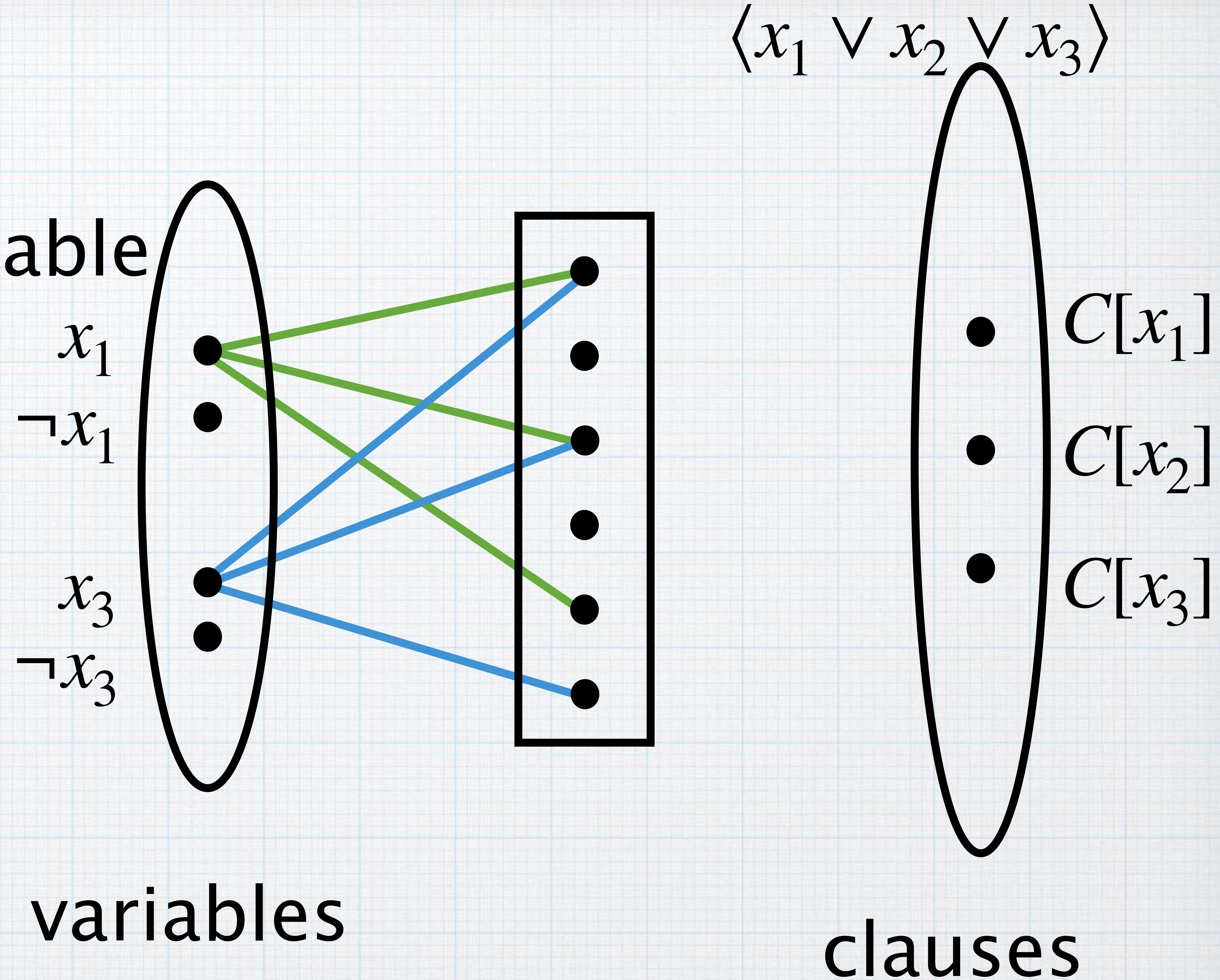
- Add pair of vertices for each variable
- Add 3 vertices for each clause
- Add $\mathcal{O}(\log(n))$ many vertices to facilitate the connections
- Variable vert. are connected to corresponding set element.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

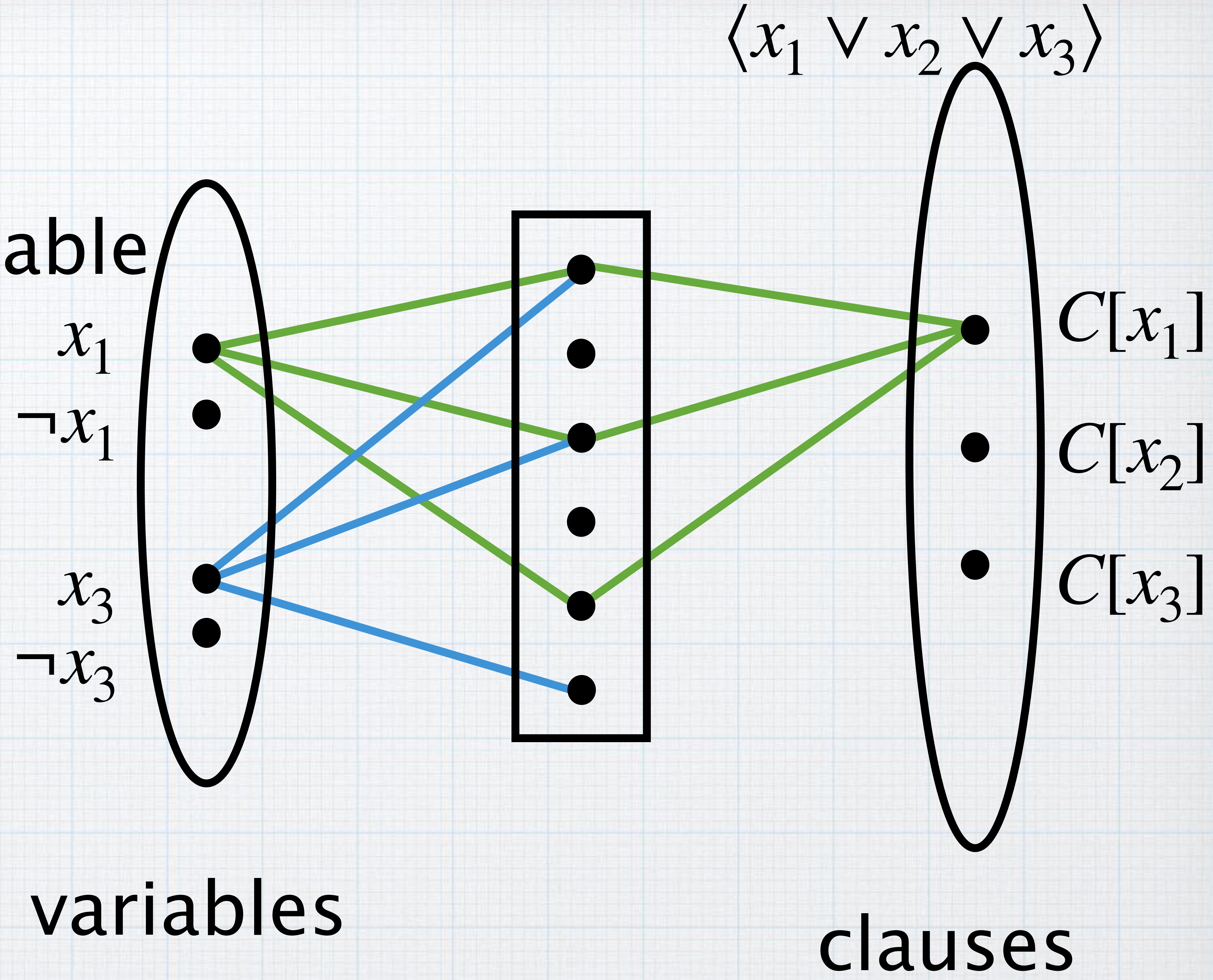
- Add pair of vertices for each variable
- Add 3 vertices for each clause
- Add $\mathcal{O}(\log(n))$ many vertices to facilitate the connections
- Variable vert. are connected to corresponding set element.
- Vertices in clause side are also connected to corresponding set elements.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

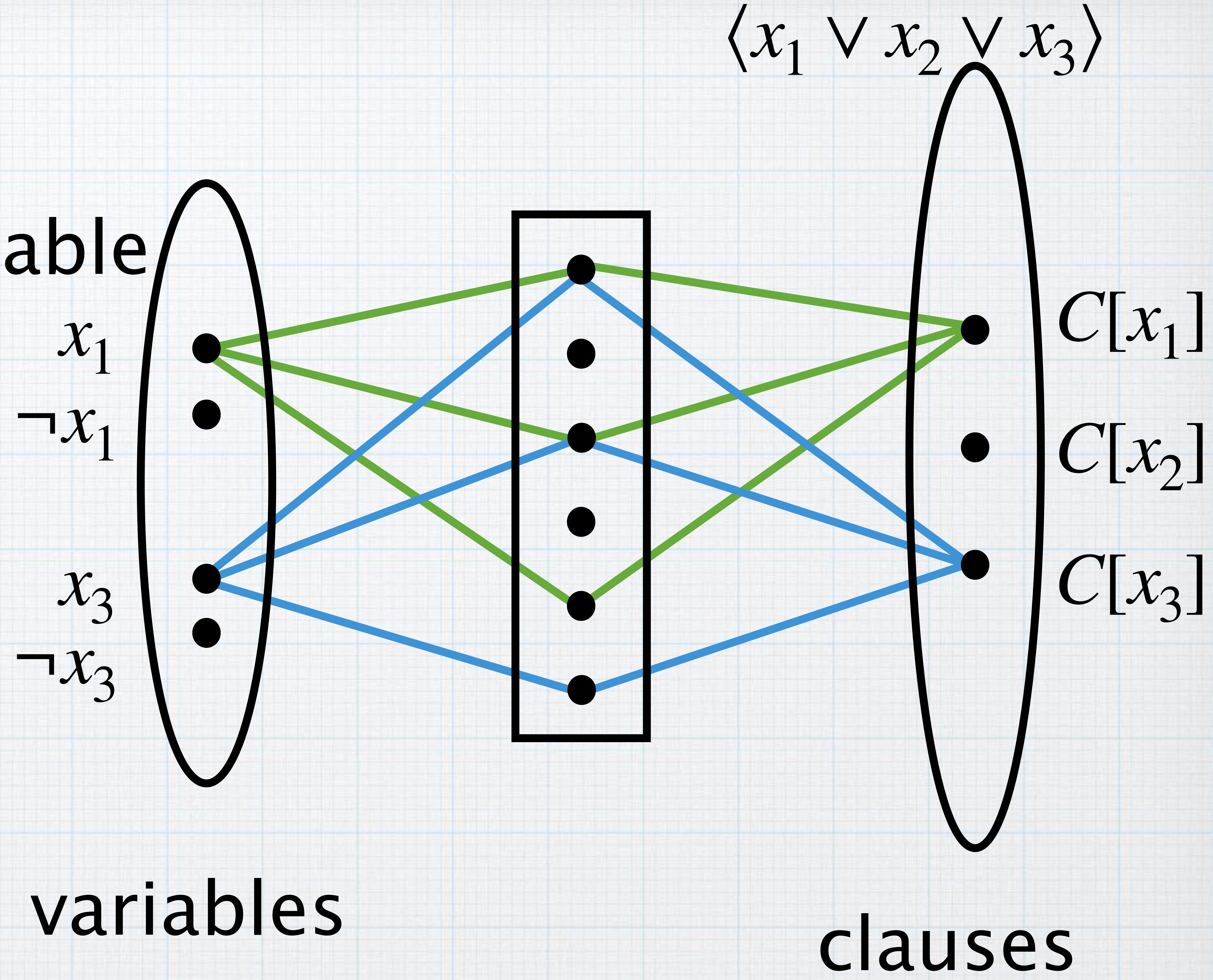
- Add pair of vertices for each variable
- Add 3 vertices for each clause
- Add $\mathcal{O}(\log(n))$ many vertices to facilitate the connections
- Variable vert. are connected to corresponding set element.
- Vertices in clause side are also connected to corresponding set elements.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

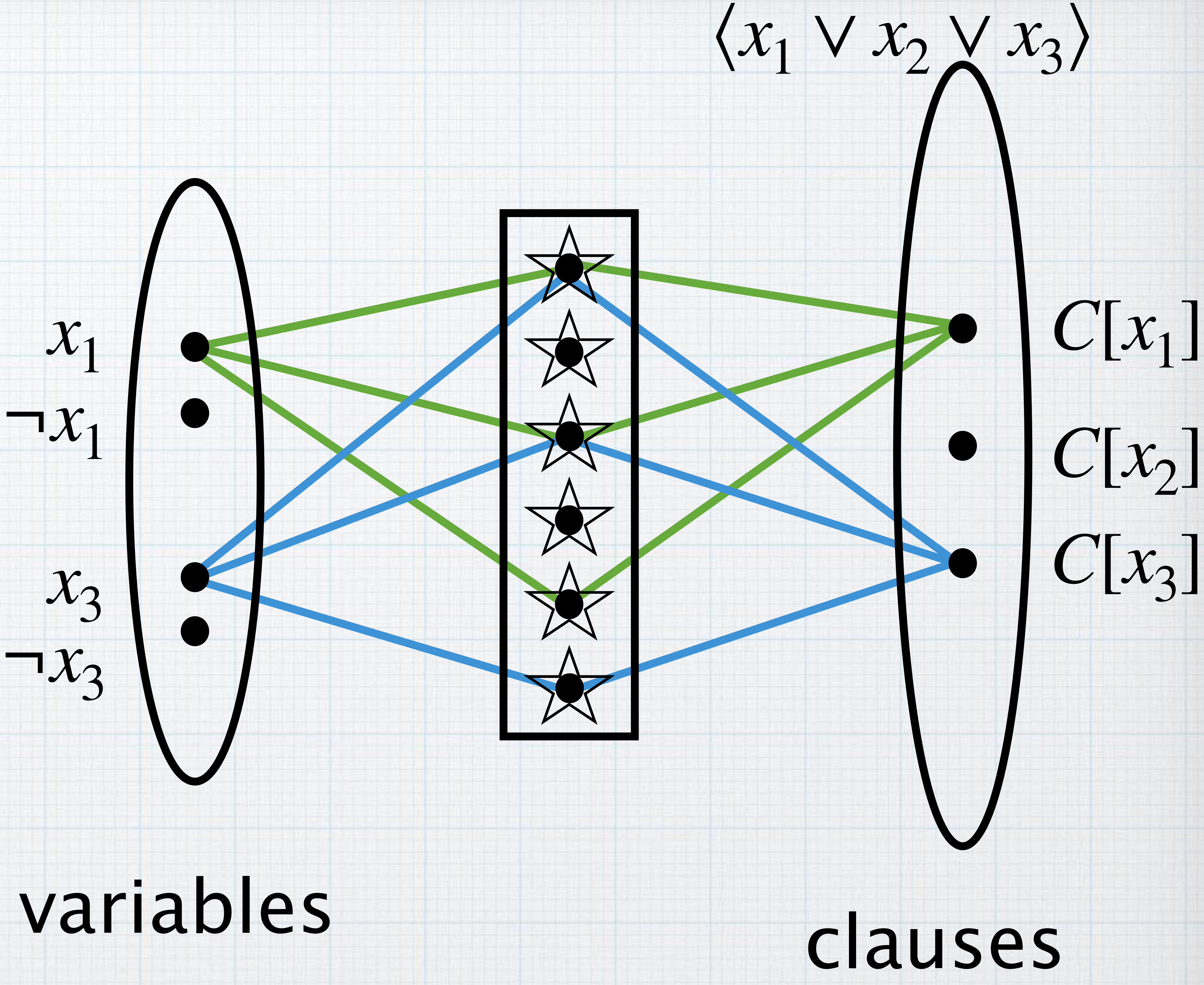
3-SAT to Loc-Dom-Set

- Add pair of vertices for each variable
- Add 3 vertices for each clause
- Add $\mathcal{O}(\log(n))$ many vertices to facilitate the connections
- Variable vert. are connected to corresponding set element.
- Vertices in clause side are also connected to corresponding set elements.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

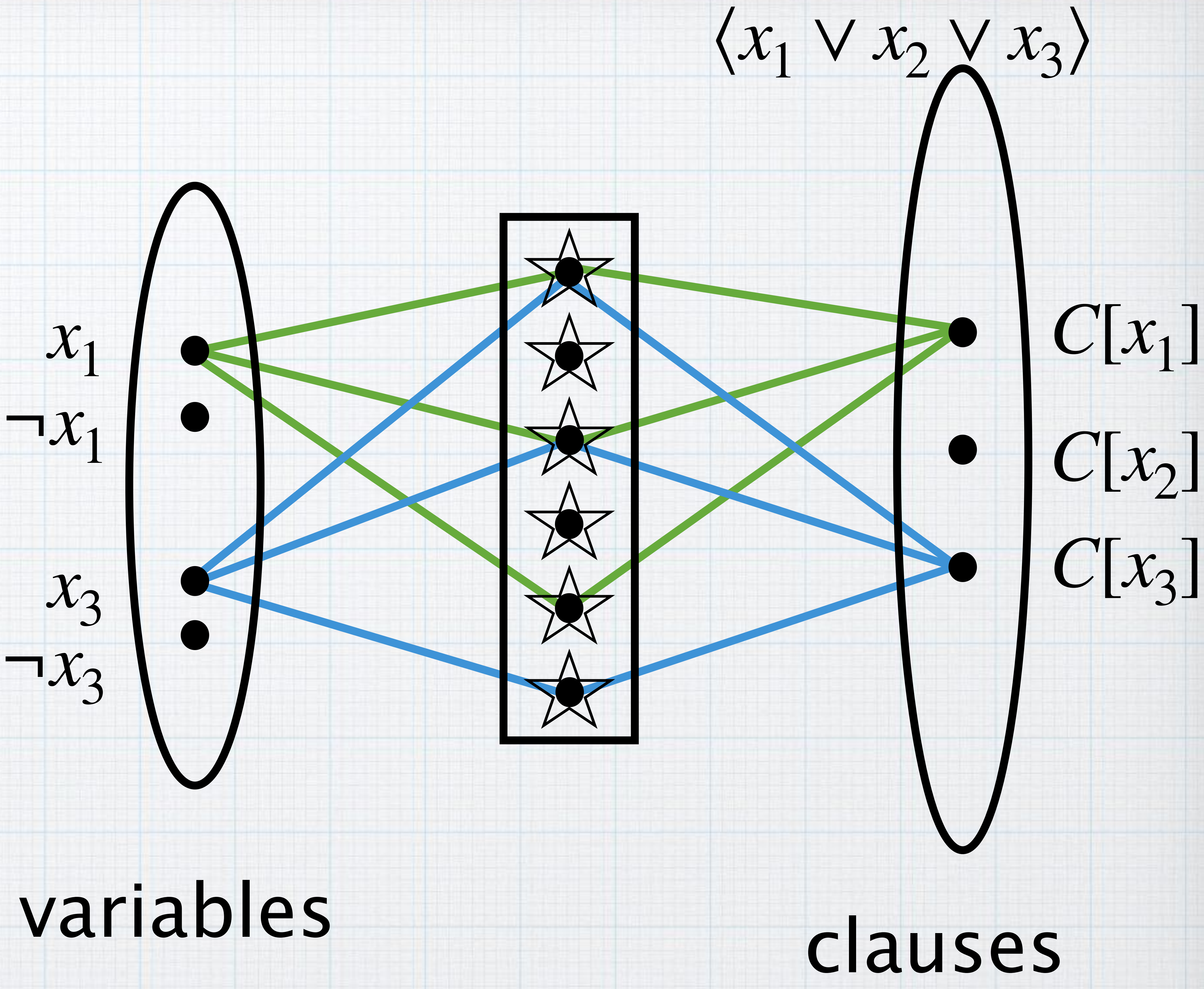
3-SAT to Loc-Dom-Set



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

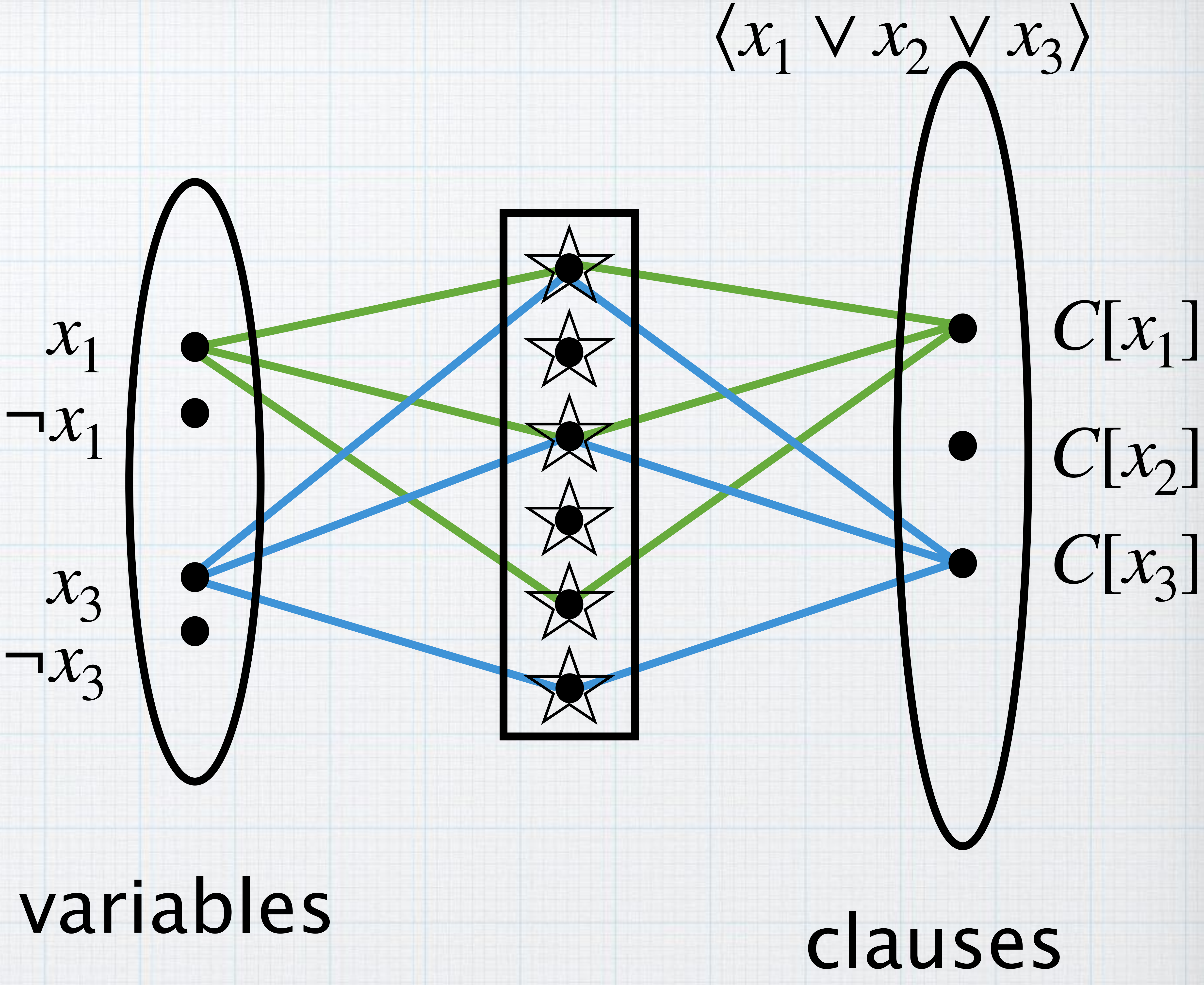
- ☆ vertex forced in solution



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

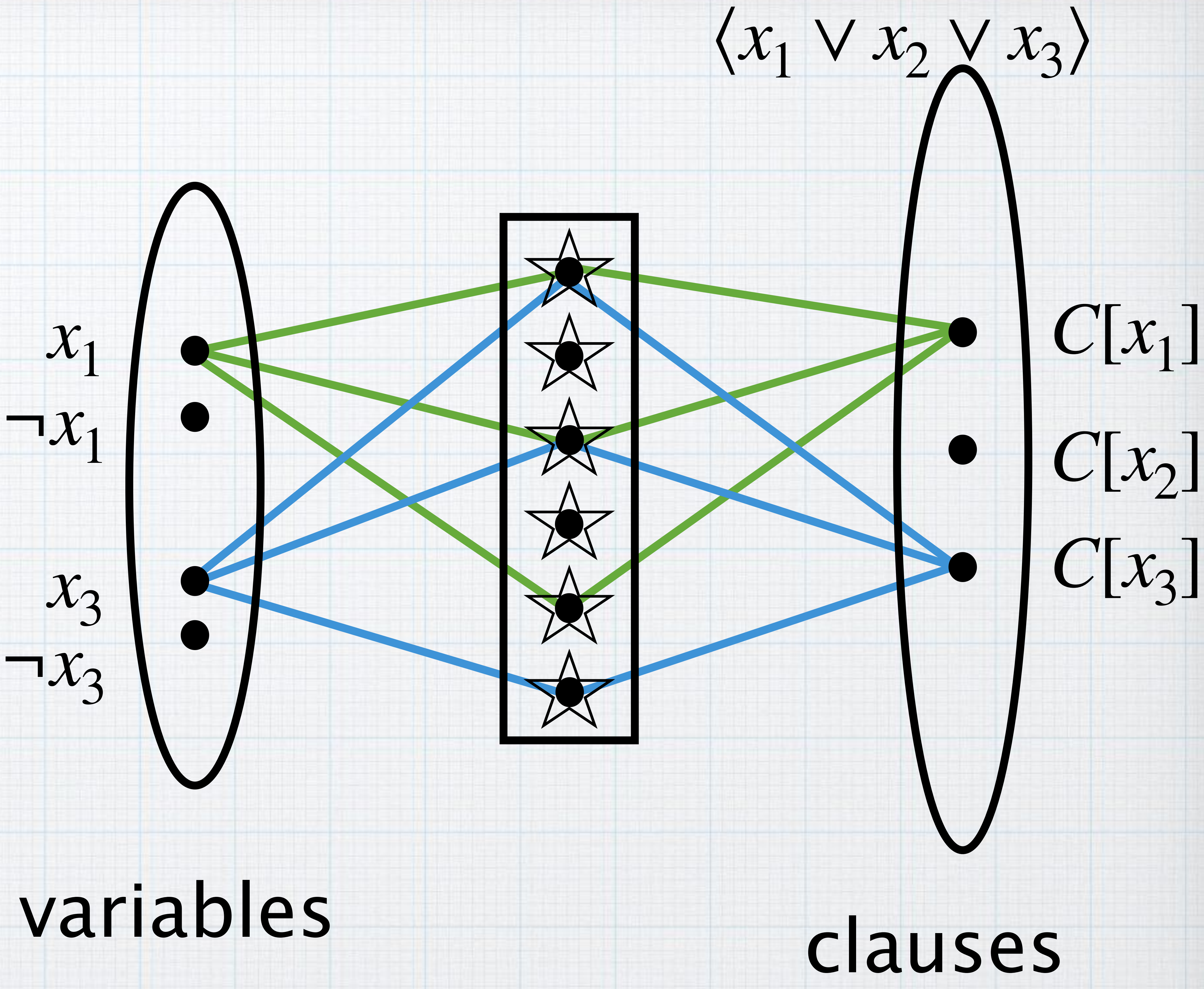
- ☆ vertex forced in solution
- For every variable, exactly one vertex is in solution.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

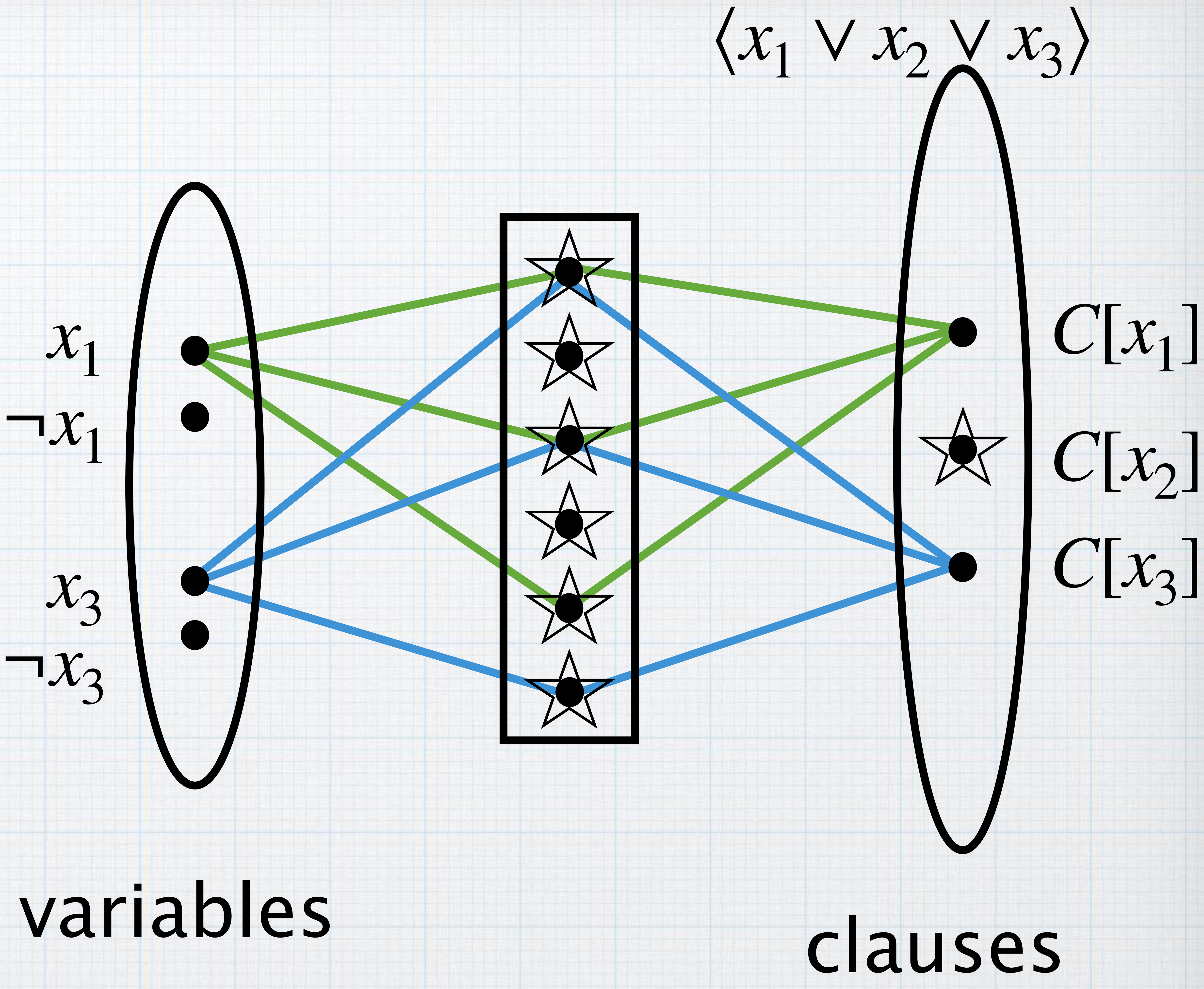
- ☆ vertex forced in solution
- For every variable, exactly one vertex is in solution.
- For every clause, exactly two vertices are is in solution.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

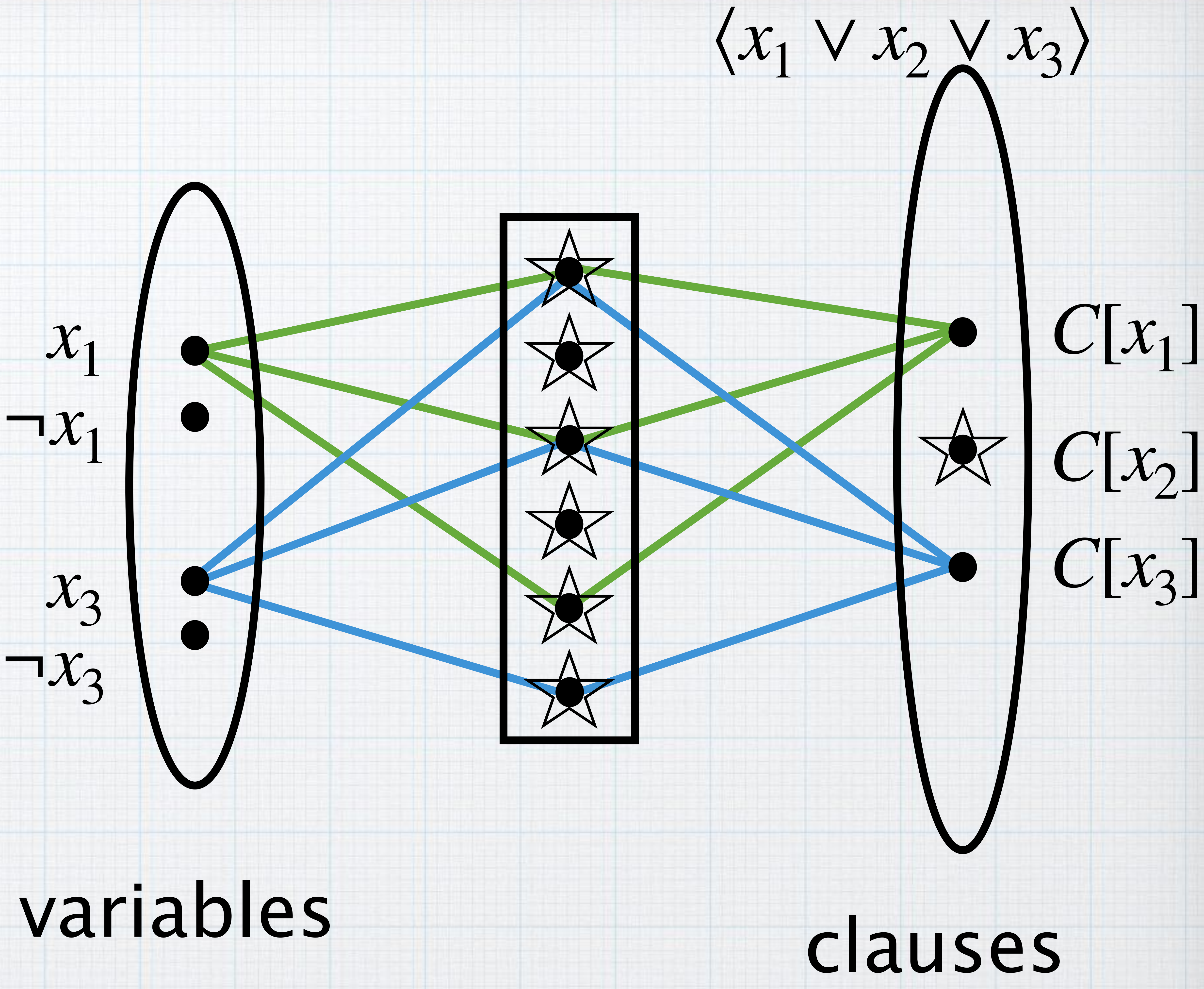
- ☆ vertex forced in solution
- For every variable, exactly one vertex is in solution.
- For every clause, exactly two vertices are is in solution.
- Suppose $C[x_2]$ is in solution (for the sake of analysis).



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

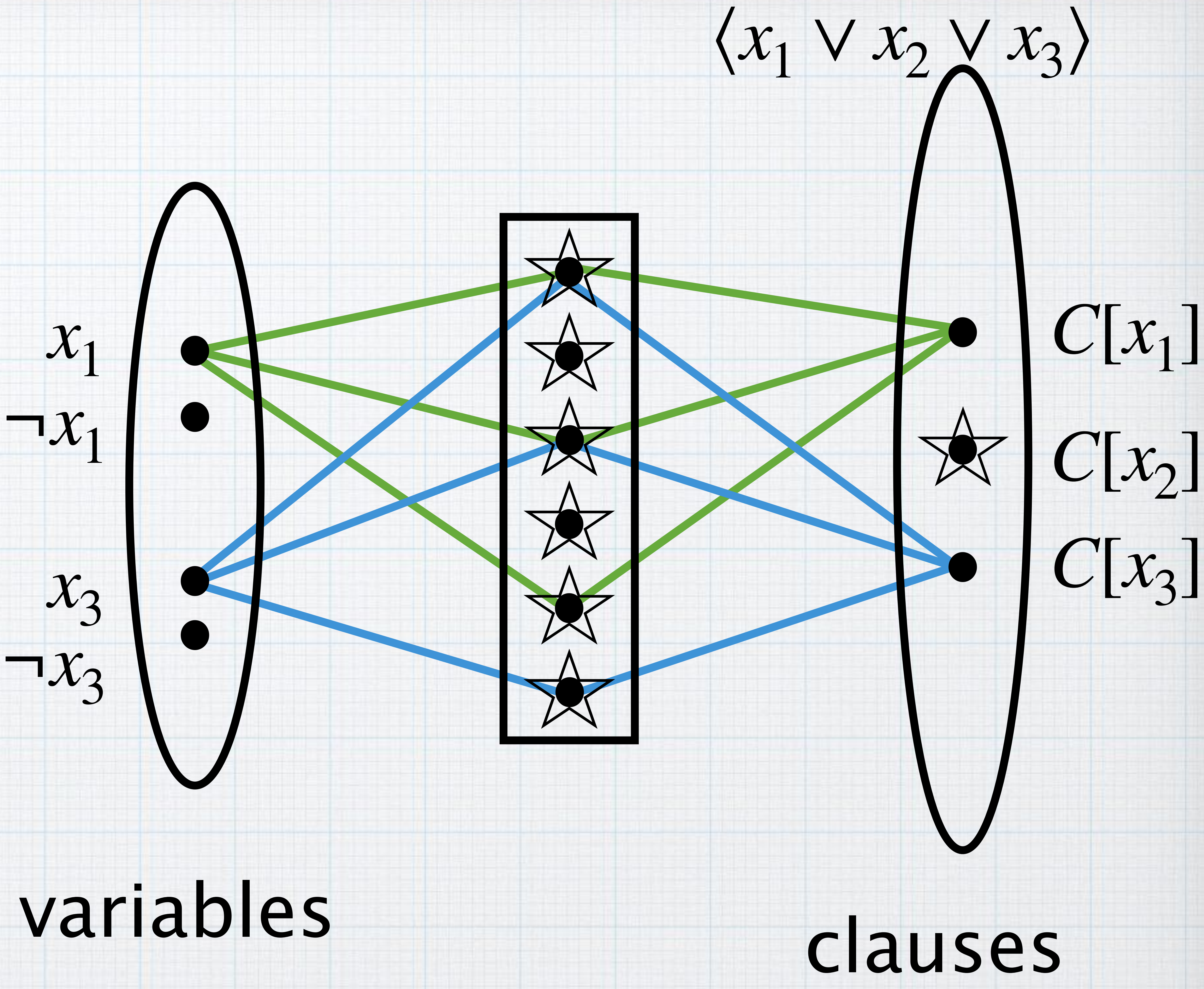
- ☆ vertex forced in solution
- For every variable, exactly one vertex is in solution.
- For every clause, exactly two vertices are is in solution.
- Suppose $C[x_2]$ is in solution (for the sake of analysis).
- Both x_1 and $C[x_1]$ can not be out of the solution.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

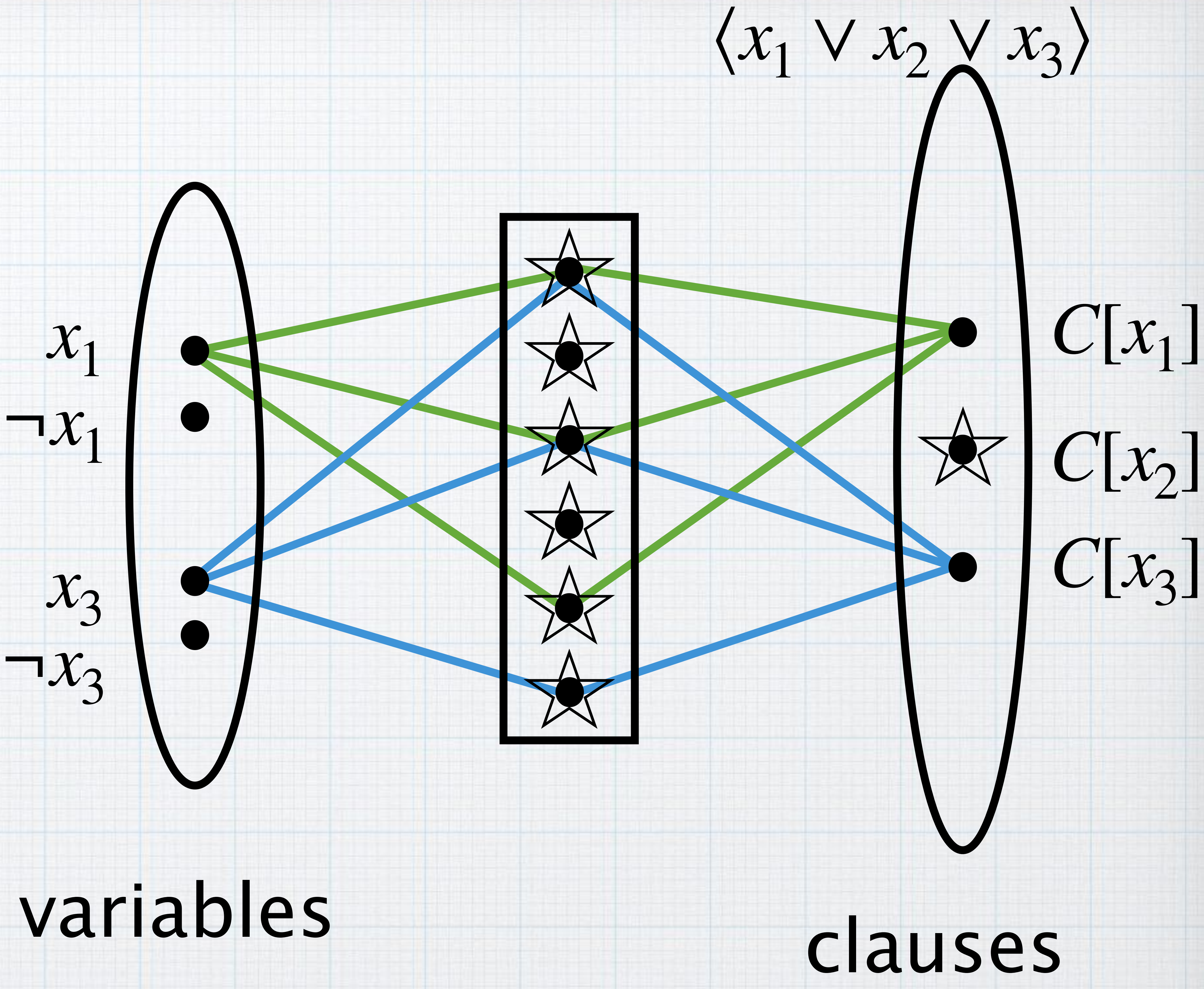
- ☆ vertex forced in solution
- For every variable, exactly one vertex is in solution.
- For every clause, exactly two vertices are is in solution.
- Suppose $C[x_2]$ is in solution (for the sake of analysis).
- Both x_1 and $C[x_1]$ can not be out of the solution.
- Both x_3 and $C[x_3]$ can not be out of the solution.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

3-SAT to Loc-Dom-Set

- ☆ vertex forced in solution
- For every variable, exactly one vertex is in solution.
- For every clause, exactly two vertices are is in solution.
- Suppose $C[x_2]$ is in solution (for the sake of analysis).
- Both x_1 and $C[x_1]$ can not be out of the solution.
- Both x_3 and $C[x_3]$ can not be out of the solution.



$X_1 = \{1,3,5\}$ $X_3 = \{1,3,6\}$

Useful to encode 3-SAT instance.

Future Directions

Future Directions

- Any other NP-Complete problems that admit double exponential lower bounds?

Future Directions

- Any other NP-Complete problems that admit double exponential lower bounds?

- Can our tight double-exponential lower bound for Locating-Dominating Set parameterized by treewidth be applied to the feedback vertex set number (a larger parameter)?

Thank you

λ-Choosability

λ–Choosability

- A proper λ–coloring of a graph G is a mapping $f: V(G) \mapsto [\lambda]$ st $f(u) \neq f(v)$ for any two adjacent vertices u, v.

λ–Choosability

– A proper λ–coloring of a graph G is a mapping $f : V(G) \mapsto [\lambda]$ st $f(u) \neq f(v)$ for any two adjacent vertices u, v.

– List coloring: Instead of having the same set $[\lambda]$ of colors, each vertex v has its own list $L(v)$ of available colors.

λ–Choosability

– A proper λ–coloring of a graph G is a mapping $f: V(G) \mapsto [\lambda]$ st $f(u) \neq f(v)$ for any two adjacent vertices u, v.

– List coloring: Instead of having the same set $[\lambda]$ of colors, each vertex v has its own list $L(v)$ of available colors.

– Graph G is λ–choosable if it has a coloring for **any** list assignment L that has size λ at each vertex.

λ–Choosability

– A proper λ–coloring of a graph G is a mapping $f : V(G) \mapsto [\lambda]$ st $f(u) \neq f(v)$ for any two adjacent vertices u, v.

– List coloring: Instead of having the same set $[\lambda]$ of colors, each vertex v has its own list $L(v)$ of available colors.

– Graph G is λ–choosable if it has a coloring for **any** list assignment L that has size λ at each vertex.

– λ–Choosability problem: Not in NP (unless …)

λ–Choosability

– A proper λ–coloring of a graph G is a mapping $f : V(G) \mapsto [\lambda]$ st $f(u) \neq f(v)$ for any two adjacent vertices u, v.

– List coloring: Instead of having the same set $[\lambda]$ of colors, each vertex v has its own list $L(v)$ of available colors.

– Graph G is λ–choosable if it has a coloring for **any** list assignment L that has size λ at each vertex.

– λ–Choosability problem: Not in NP (unless …)

Thm [Marx, Mitsou (ICALP'16)]. The λ–Choosability problem

— admits $2^{2^{\mathcal{O}(\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$–time algo, but

— does not admit $2^{2^{o(\mathrm{tw})}} \cdot n^{\mathcal{O}(1)}$ algo unless the ETH fails.