k-Sum Hardness Implies tw-SETH

Michael Lampis

Dauphine | PSL UNIVERSITÉ PARIS
LAMSADE UMR 7243

GRIF, June 5th 2026

Main result

Main result

Theorem

If there exists a $(2-\varepsilon)^{tw} n^{O(1)}$ *time algorithm for* 3-SAT
$\Rightarrow$ *there exists an* $n^{(1-\varepsilon)\frac{k}{2}}$ *algorithm for* k-SUM.

Main result

Theorem

If there exists a $(2-\varepsilon)^{tw} n^{O(1)}$ *time algorithm for* 3-SAT
$\Rightarrow$ *there exists an* $n^{(1-\varepsilon)\frac{k}{2}}$ *algorithm*[a] *for* k-SUM.

[a] Randomized, for some other ε and sufficiently large k

Main result

Theorem

If there exists a $(2-\varepsilon)^{tw}n^{O(1)}$ *time algorithm for* 3-SAT
$\Rightarrow$ *there exists an* $n^{(1-\varepsilon)\frac{k}{2}}$ *algorithm for* k-SUM.

Theorem

*(***More Informal:***) If current best algorithm for* k-SUM *is* **optimal**,
$\Rightarrow$ *standard DP algorithm for* 3-SAT$/tw$ *is also* **optimal**.

Main result

Theorem

If there exists a (

$\Rightarrow$ *there exists an*

Theorem

*(***More Informal:**

is **optimal**,

$\Rightarrow$ *standard DP a*

.

Image credit: nano banana

Main result

Theorem

If there exists a $(2-\varepsilon)^{tw}n^{O(1)}$ *time algorithm for* 3-SAT
$\Rightarrow$ *there exists an* $n^{(1-\varepsilon)\frac{k}{2}}$ *algorithm for* k-SUM.

Theorem

*(***More Informal:***) If current best algorithm for* k-SUM *is* **optimal**,
$\Rightarrow$ *standard DP algorithm for* 3-SAT$/tw$ *is also* **optimal**.

- Same statements hold for k-XOR.
- Result presented in SODA 2026.

Fair Warning

Fair Warning

Fair Warning

Talk will be heavy on:

- Controversial Opinions
- **Propaganda**

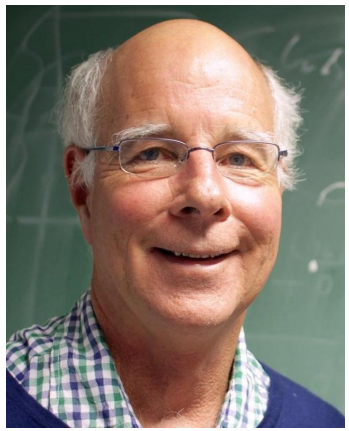
Objective:

- Build case for research direction
- (**Entertain**)

Background

Parameterized Complexity

- Do **not** measure complexity as function of input **size** n.
- Measure complexity as function of input **structure** k.
- k is the **parameter**.
- Motivating difference: $f(k)n^{O(1)}$ vs $n^{g(k)}$.



Engineer's Motivation:

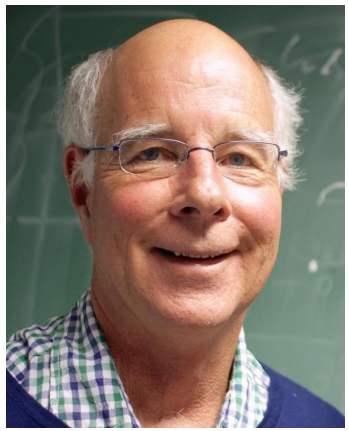
- If $f(k)$ is reasonable, this could actually be a useful algorithm!

Scientist's Motivation:

- **Why** are problems hard?

Parameterized Complexity

- Do **not** measure complexity as function of input **size** n.
- Measure complexity as function of input **structure** k.
- k is the **parameter**.
- Motivating difference: $f(k)n^{O(1)}$ vs $n^{g(k)}$.



Engineer's Motivation:

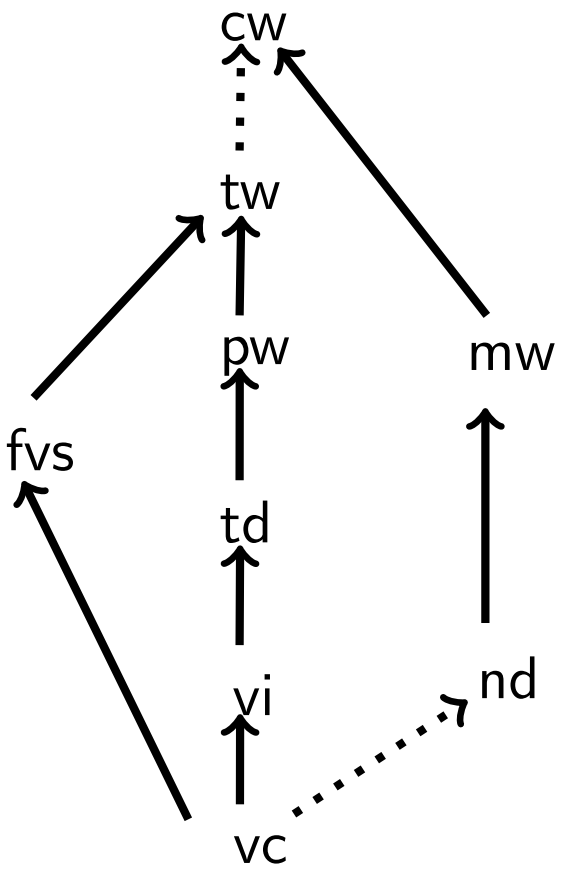
- If $f(k)$ is reasonable, this could actually be a useful algorithm!

Scientist's Motivation:

- **Why** are problems hard?

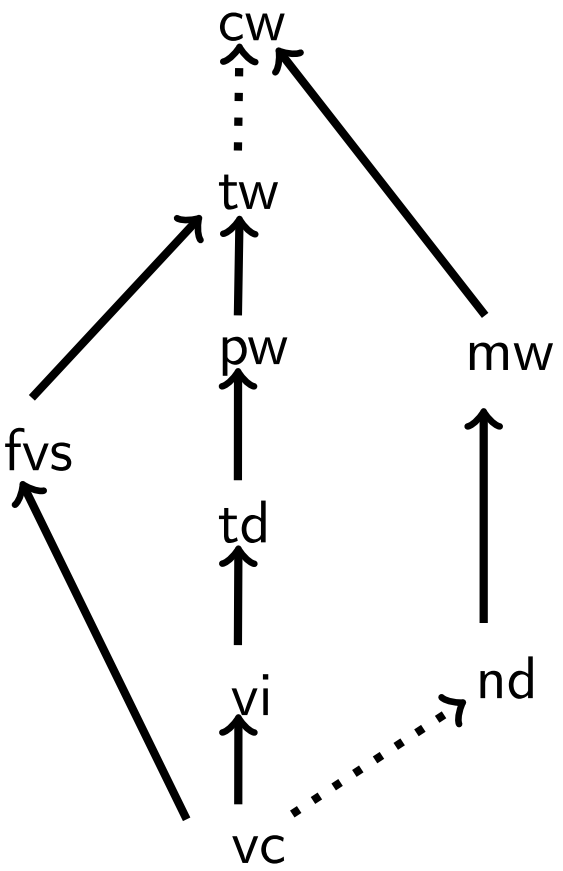
Bottom line: Complexity must take into account **input structure**.

Which parameters – which structure?



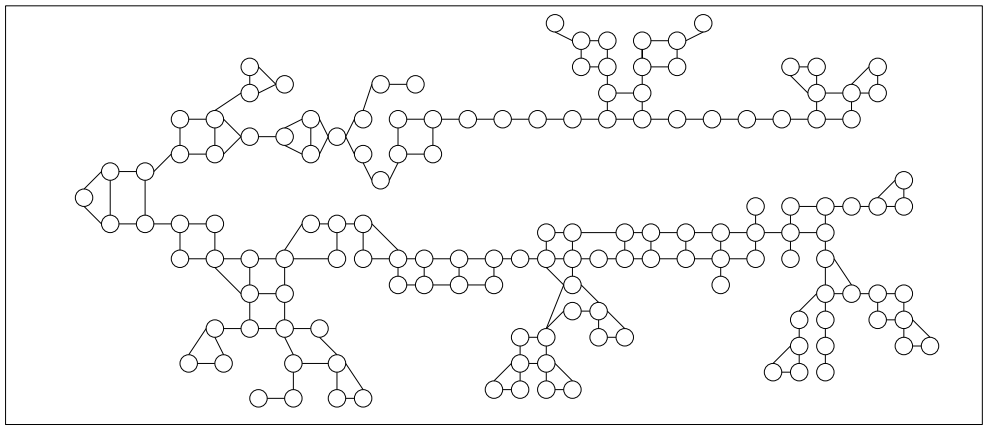
Which parameters – which structure?

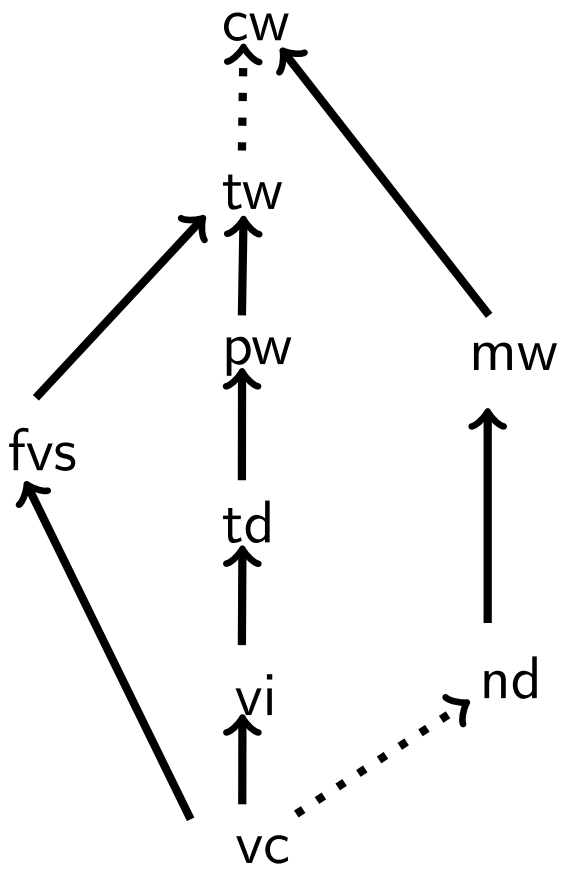
- Key parameter : **Treewidth**
  - $\mathrm{tw}(G)$ bounded $\rightarrow$ G is "tree-like"



Which parameters – which structure?

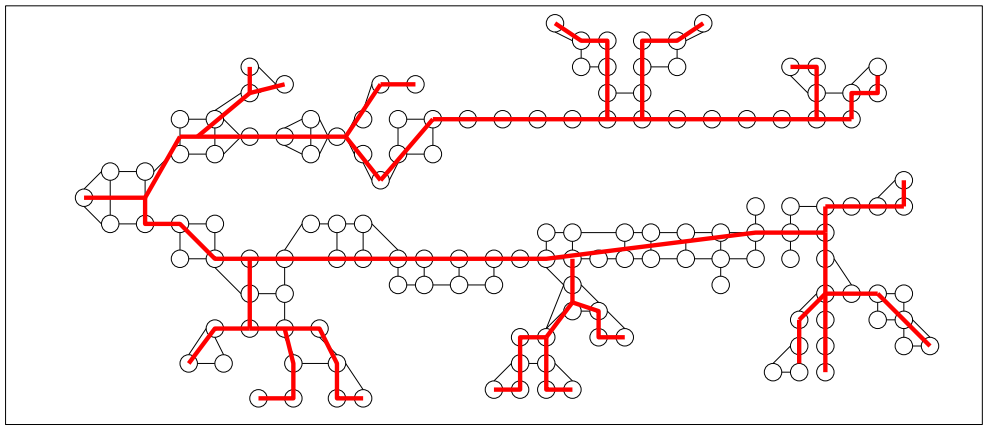
- Key parameter : **Treewidth**
  - $\mathrm{tw}(G)$ bounded $\rightarrow$ G is "tree-like"

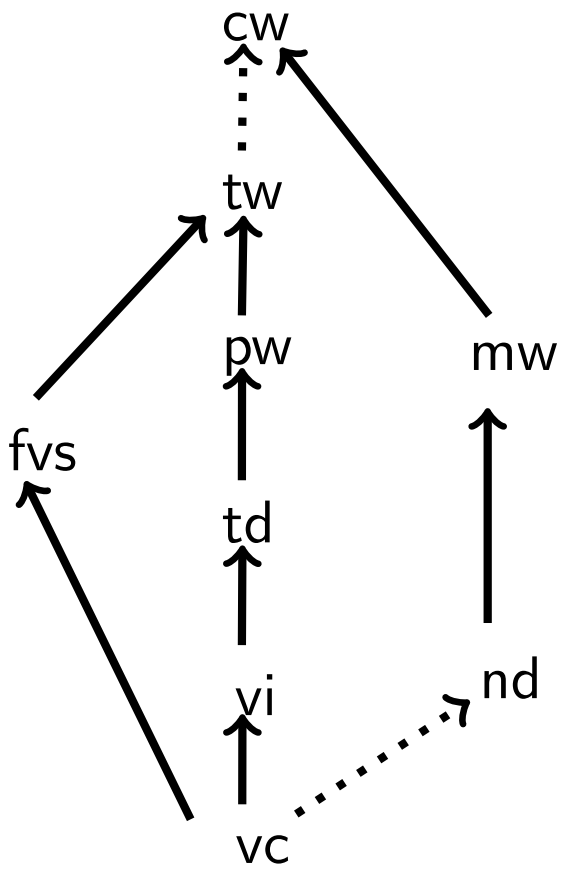




Which parameters – which structure?

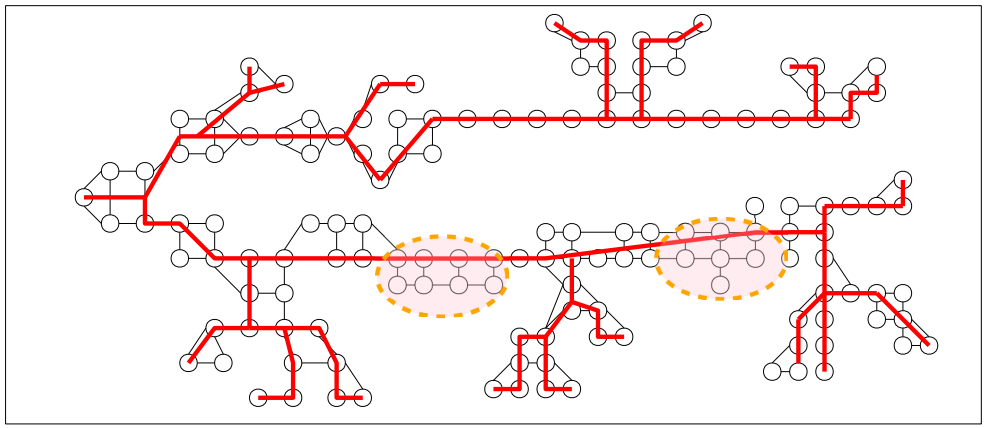
- Key parameter : **Treewidth**
  - $\mathrm{tw}(G)$ bounded $\rightarrow$ G is "tree-like"

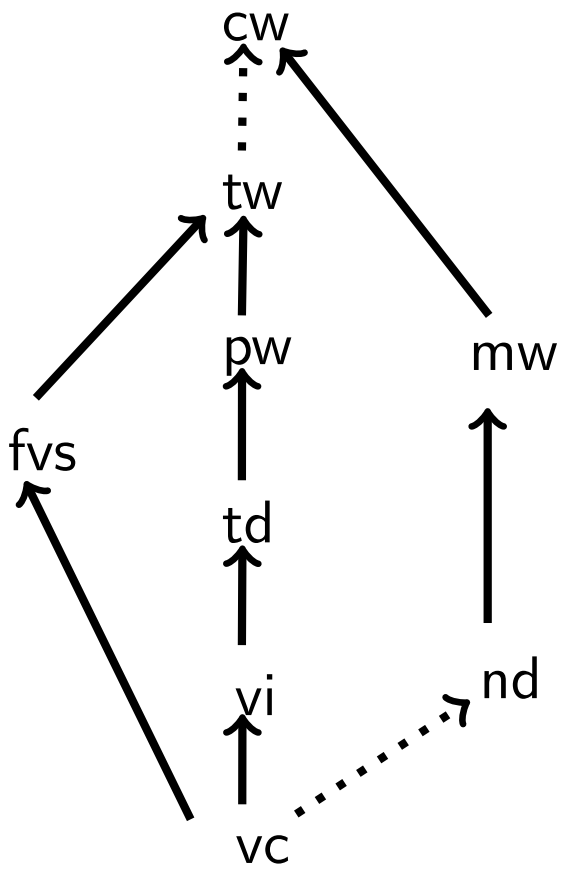




Which parameters – which structure?

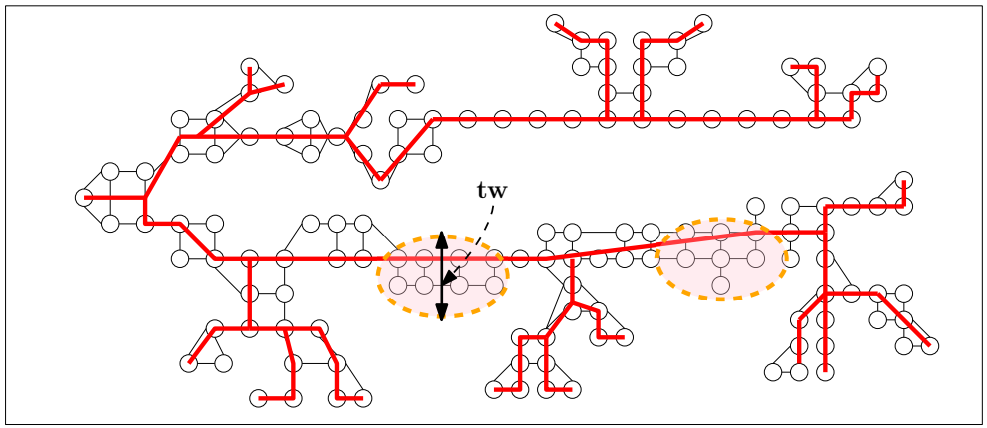
- Key parameter : **Treewidth**
  - $\mathrm{tw}(G)$ bounded $\rightarrow$ G is "tree-like"

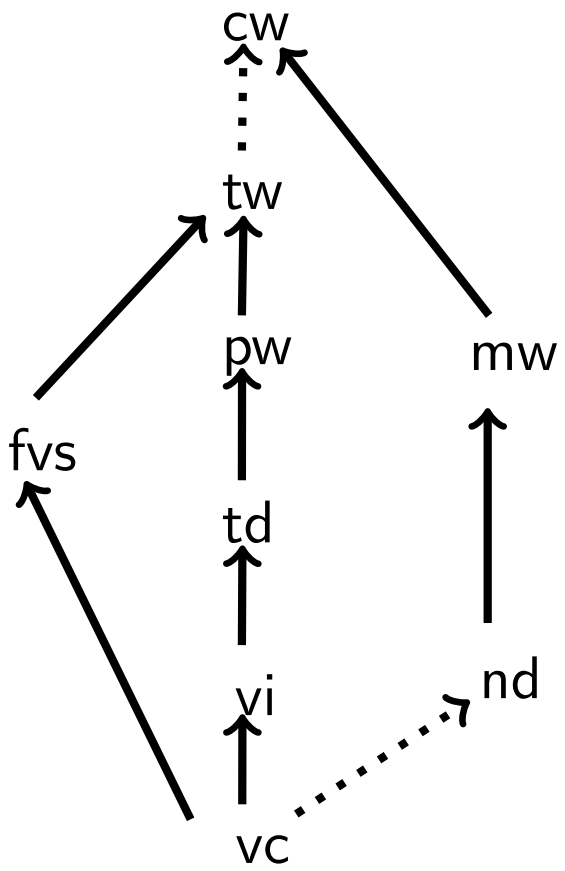




Which parameters – which structure?

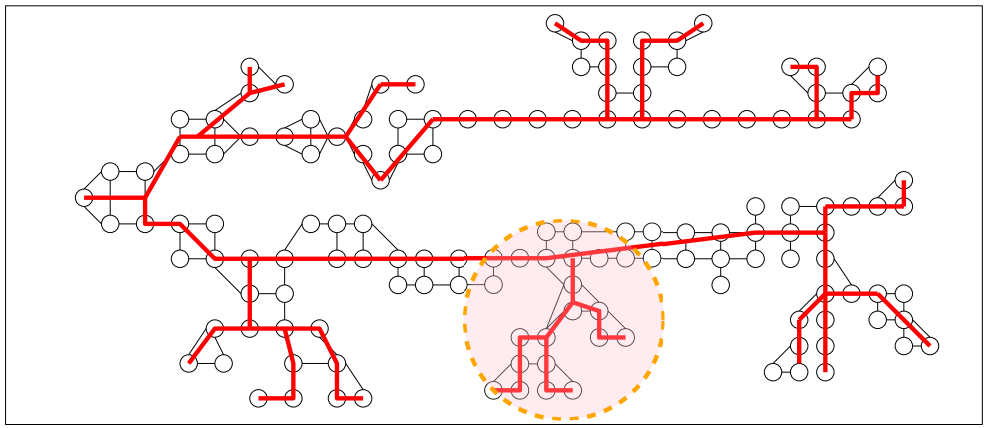
- Key parameter : **Treewidth**
  - $\mathrm{tw}(G)$ bounded $\rightarrow$ G is "tree-like"

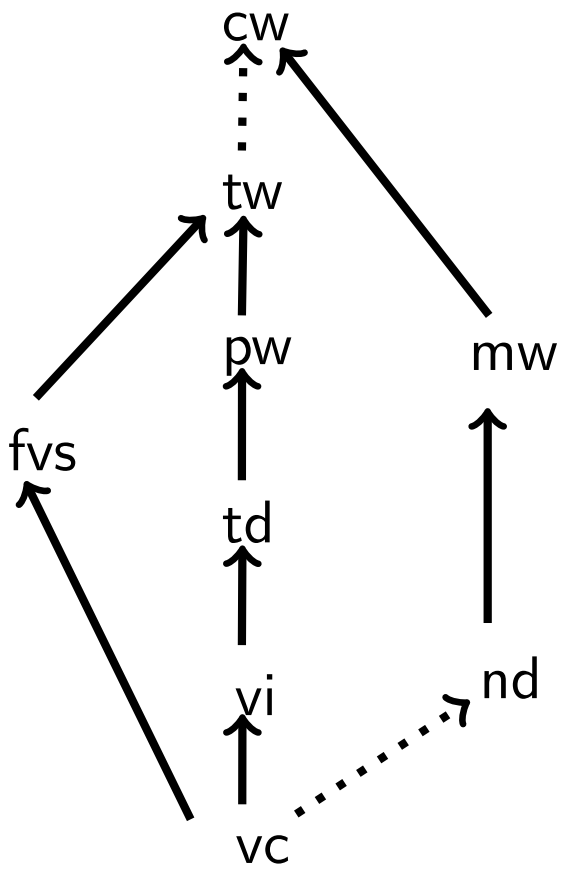




Which parameters – which structure?

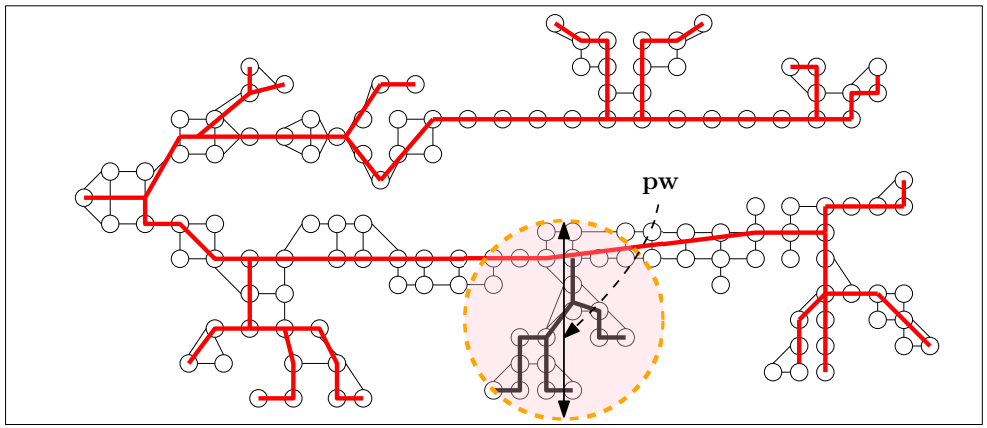
- Key parameter : **Treewidth**
  - $\mathrm{tw}(G)$ bounded $\rightarrow$ G is "tree-like"

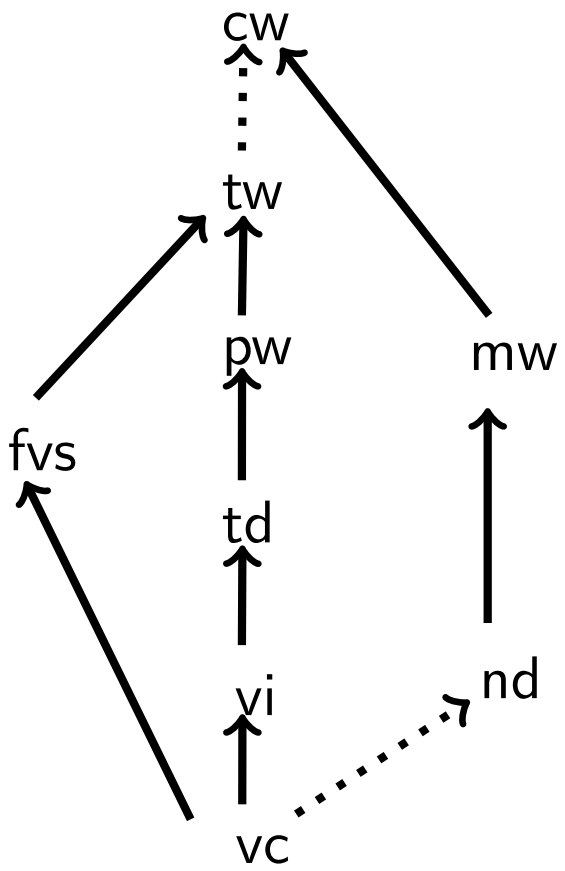




Which parameters – which structure?

- Key parameter : **Treewidth**
  - $\mathrm{tw}(G)$ bounded $\rightarrow$ G is "tree-like"





Concretely :

- Pathwidth should be **easier** than treewidth !
  - For which problems ?
  - Complexity impact ? $\rightarrow$ **Fine-Grained complexity**

Treewidth – Pathwidth

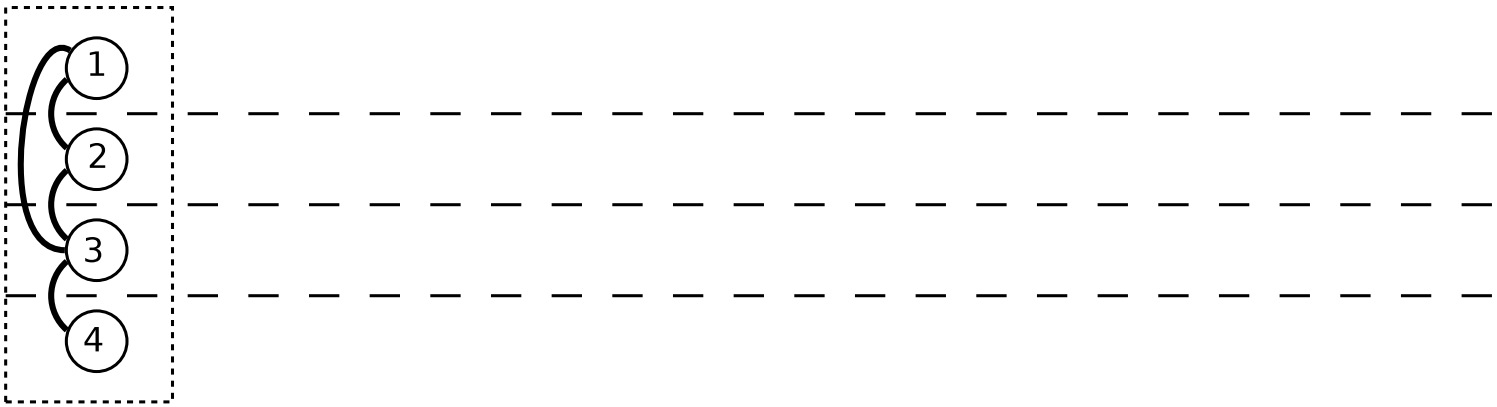
Gentle definition of pathwidth k:

- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.

Treewidth – Pathwidth

Gentle definition of pathwidth k:

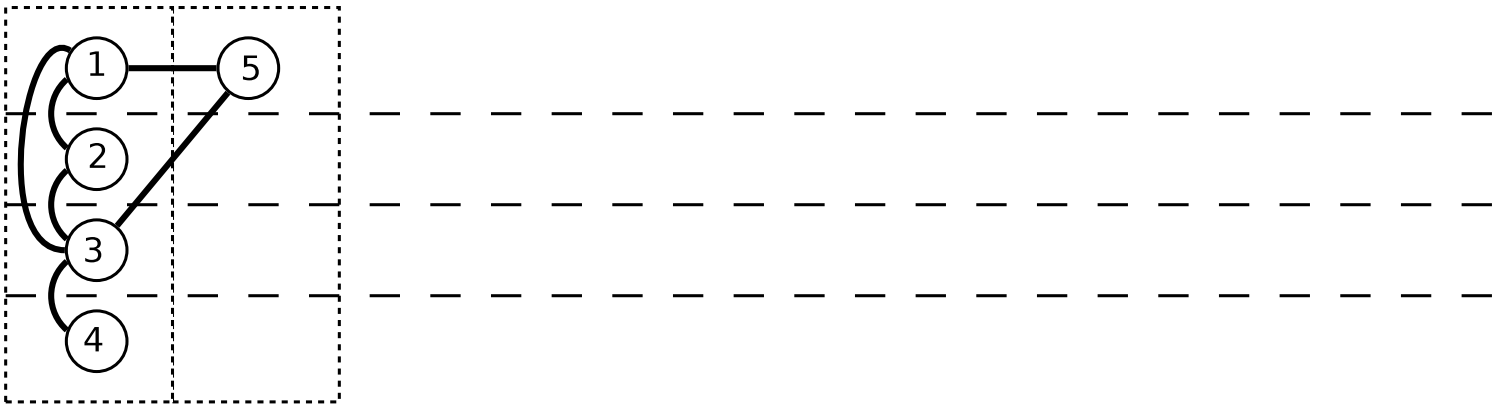
- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Treewidth – Pathwidth

Gentle definition of pathwidth k:

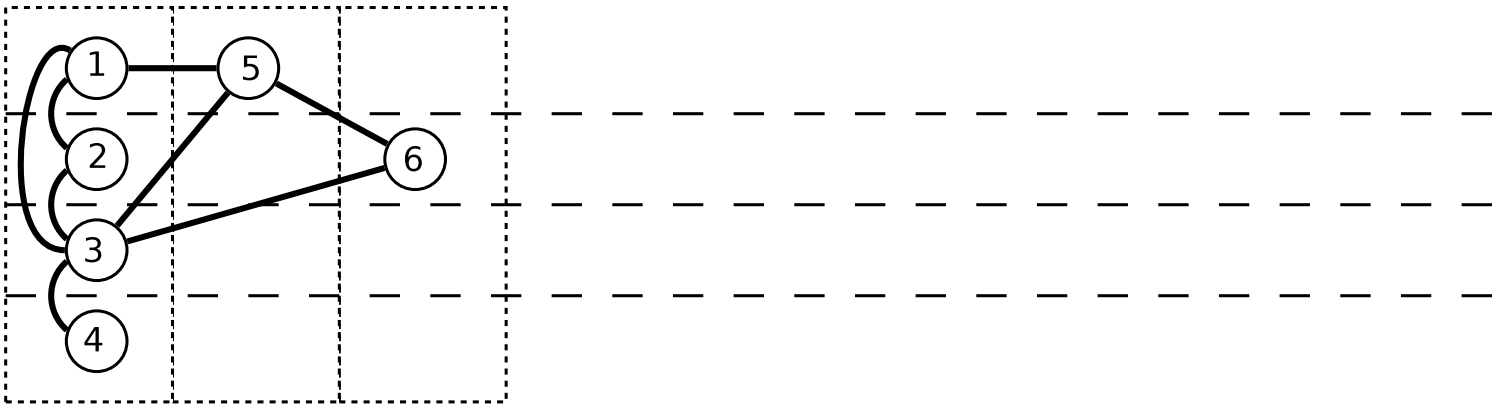
- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Treewidth – Pathwidth

Gentle definition of pathwidth k:

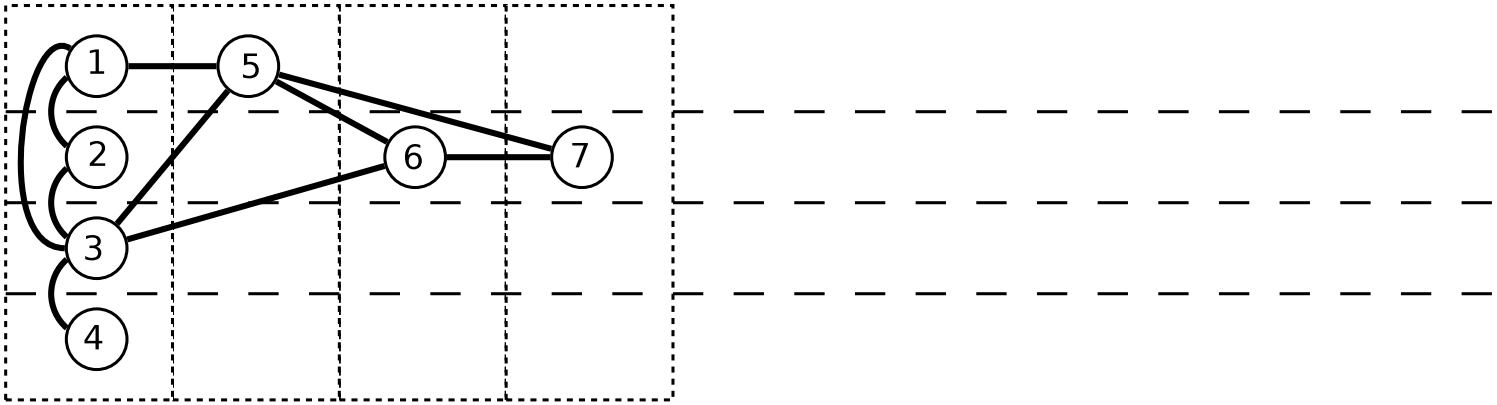
- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Treewidth – Pathwidth

Gentle definition of pathwidth k:

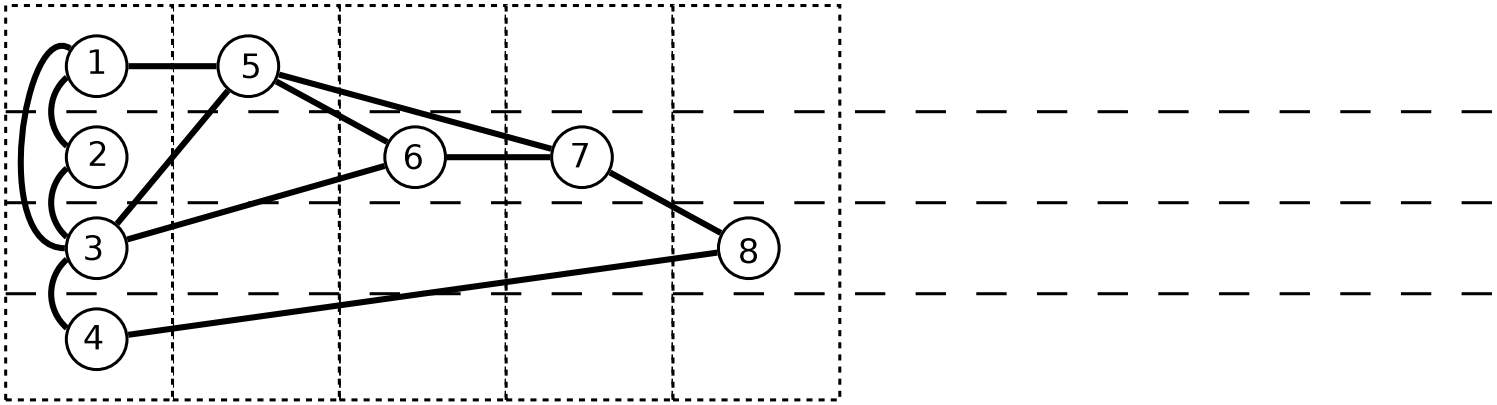
- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Treewidth – Pathwidth

Gentle definition of pathwidth k:

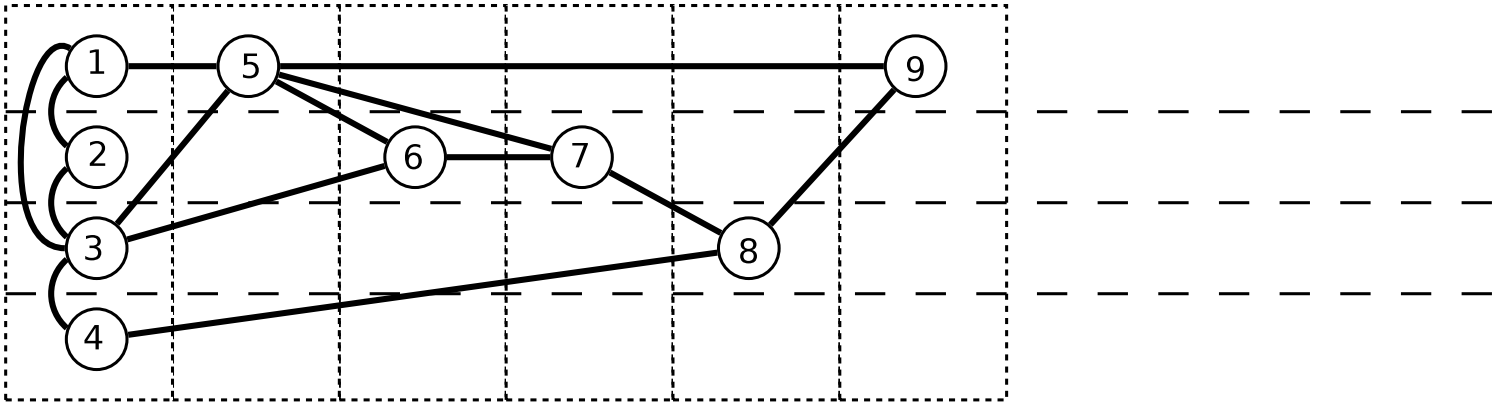
- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Treewidth – Pathwidth

Gentle definition of pathwidth k:

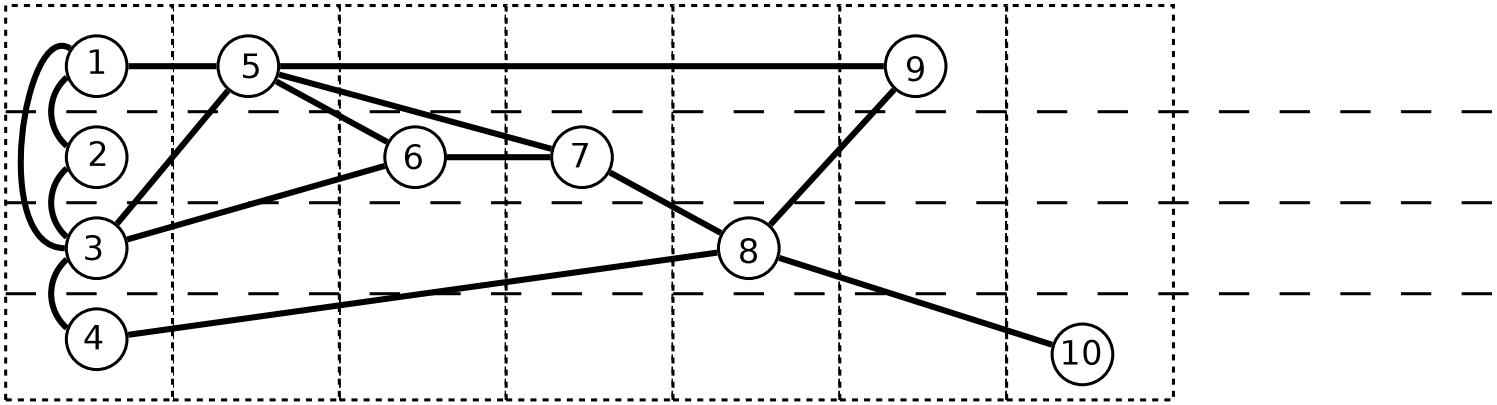
- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Treewidth – Pathwidth

Gentle definition of pathwidth k:

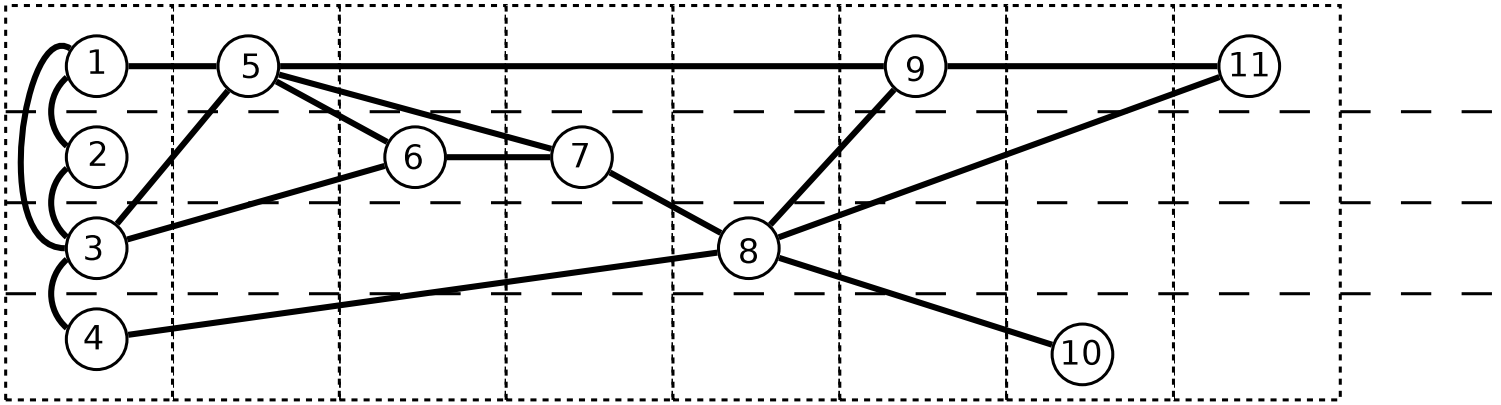
- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Treewidth – Pathwidth

Gentle definition of pathwidth k:

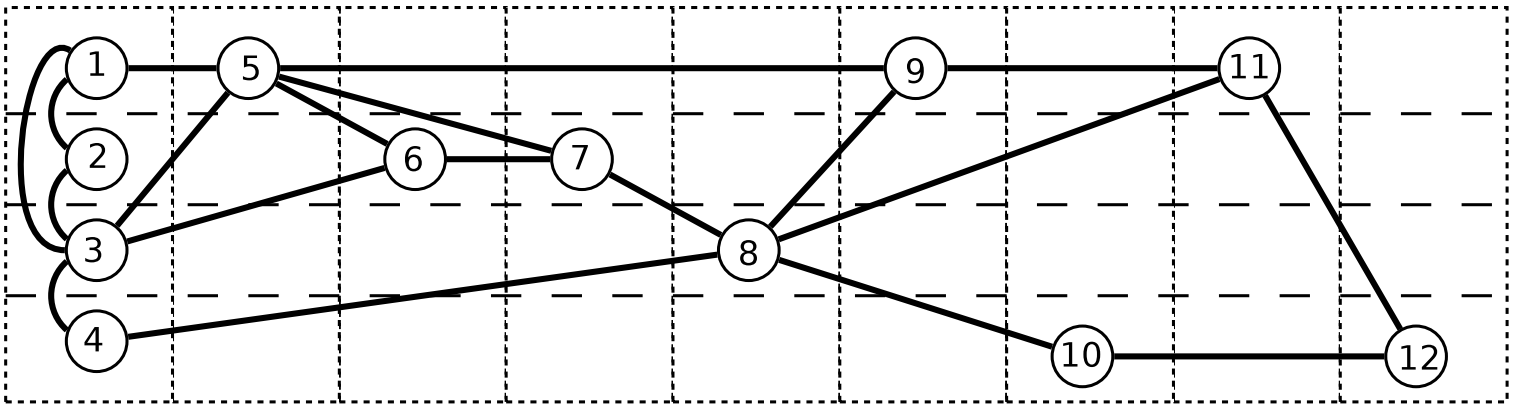
- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



Treewidth – Pathwidth

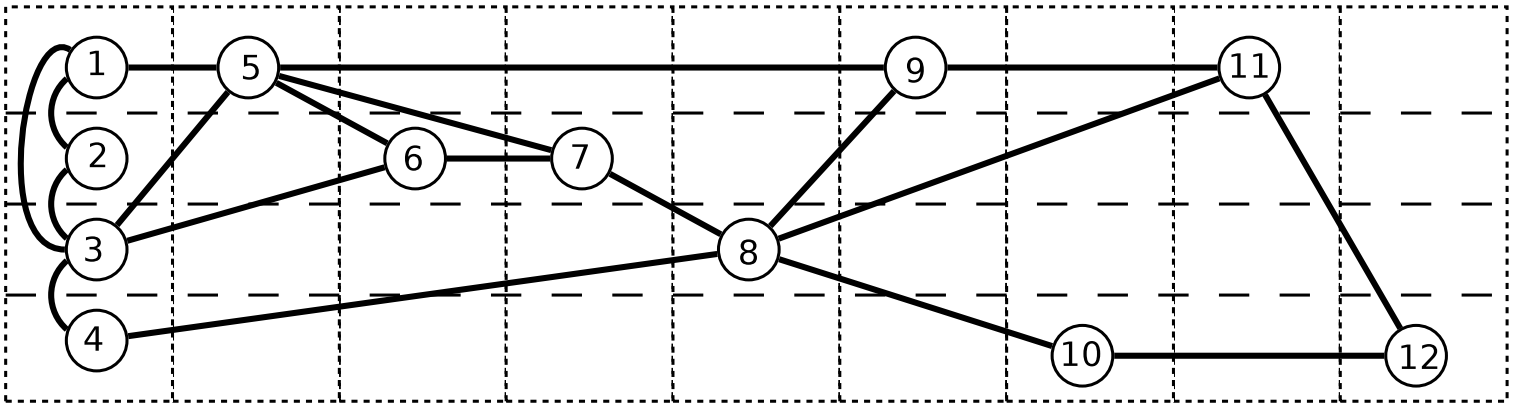
Gentle definition of pathwidth k:

- We have k stacks. Initially each contains a vertex. They are arbitrarily connected.
- At each step we add a vertex to the top of a stack. It can be connected to vertices currently on top of a stack.



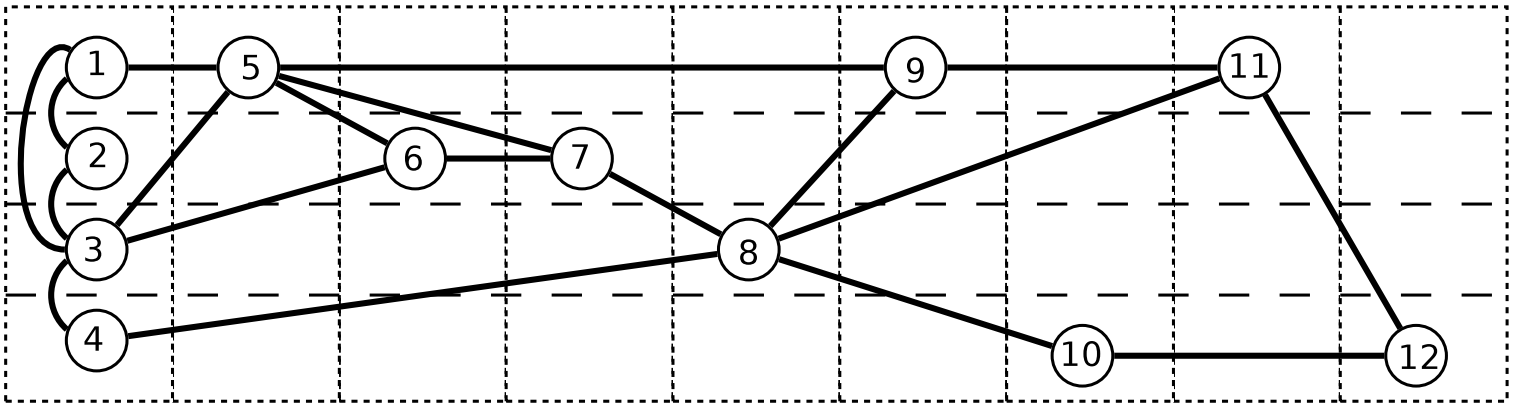
Treewidth – Pathwidth

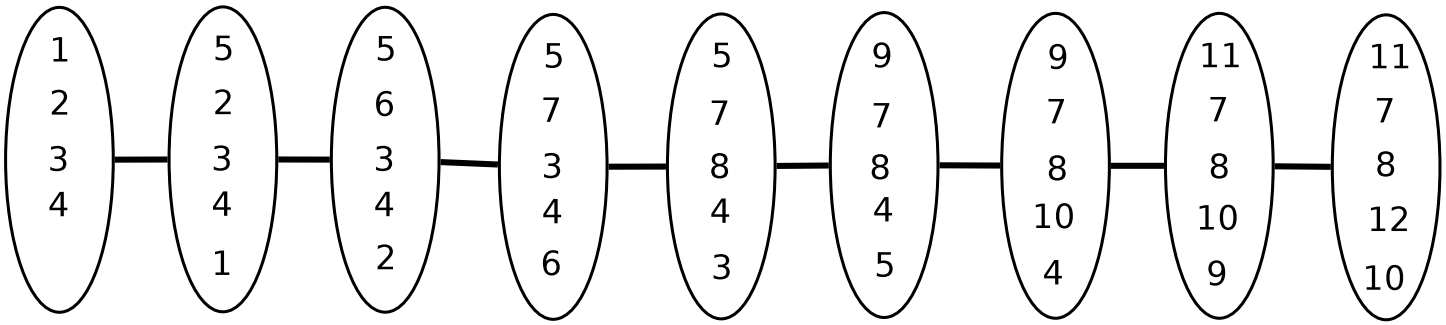
Note that this is equivalent to the standard definition of path decompositions.



Treewidth – Pathwidth

Note that this is equivalent to the standard definition of path decompositions.



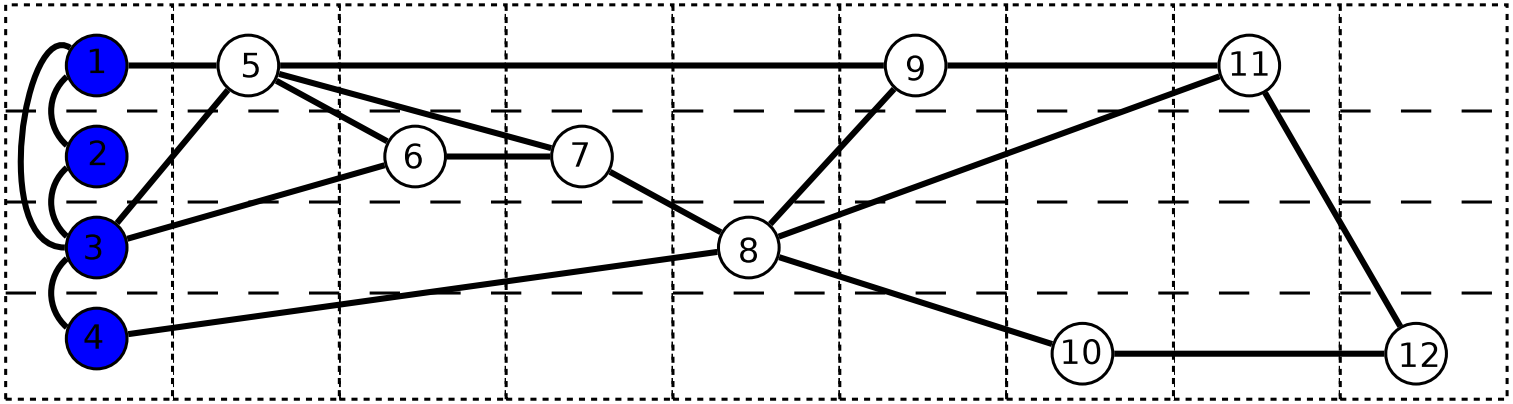


Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.

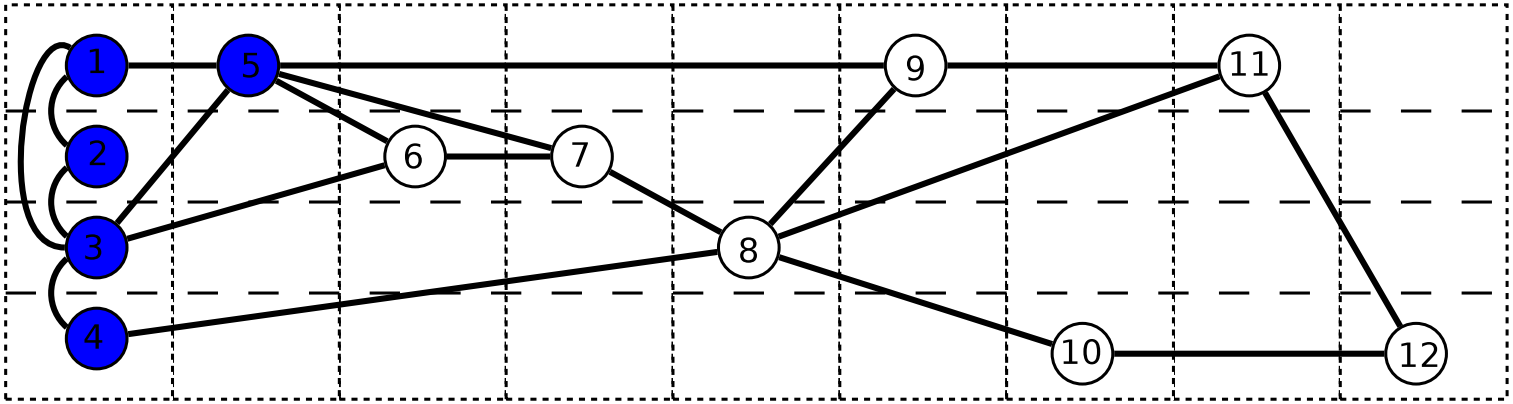
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



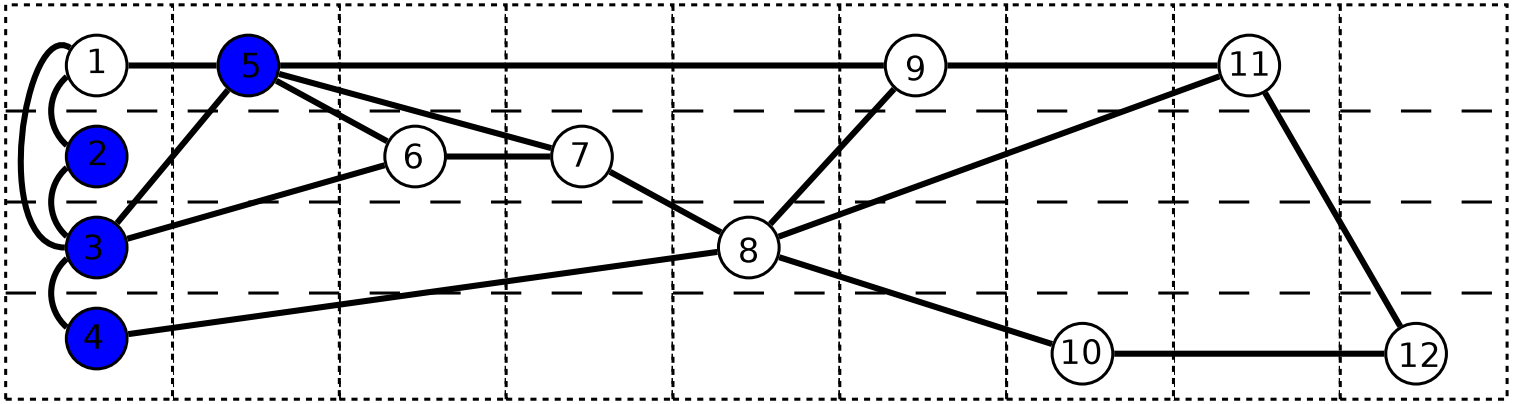
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



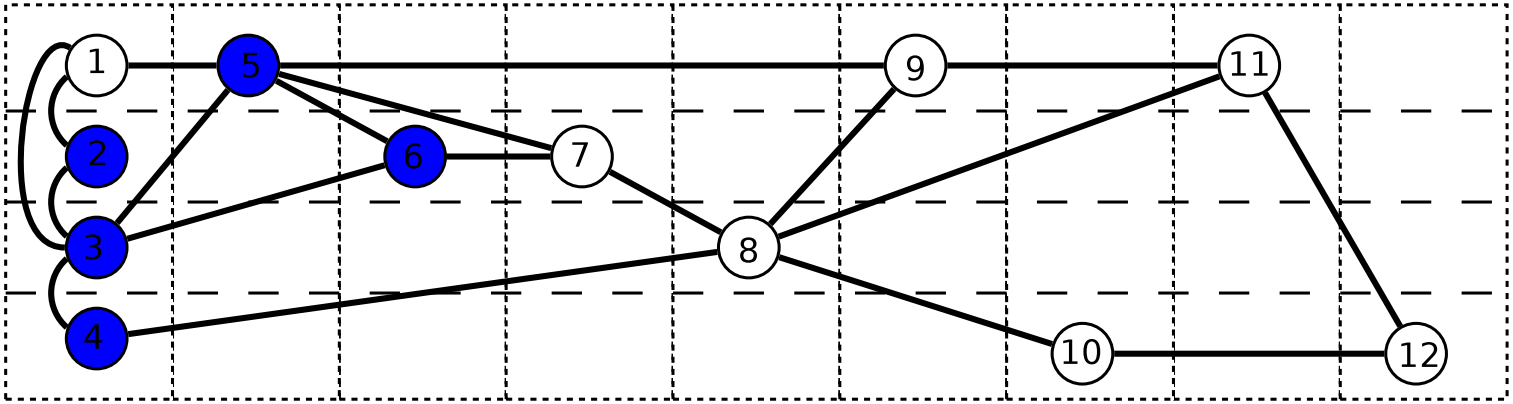
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



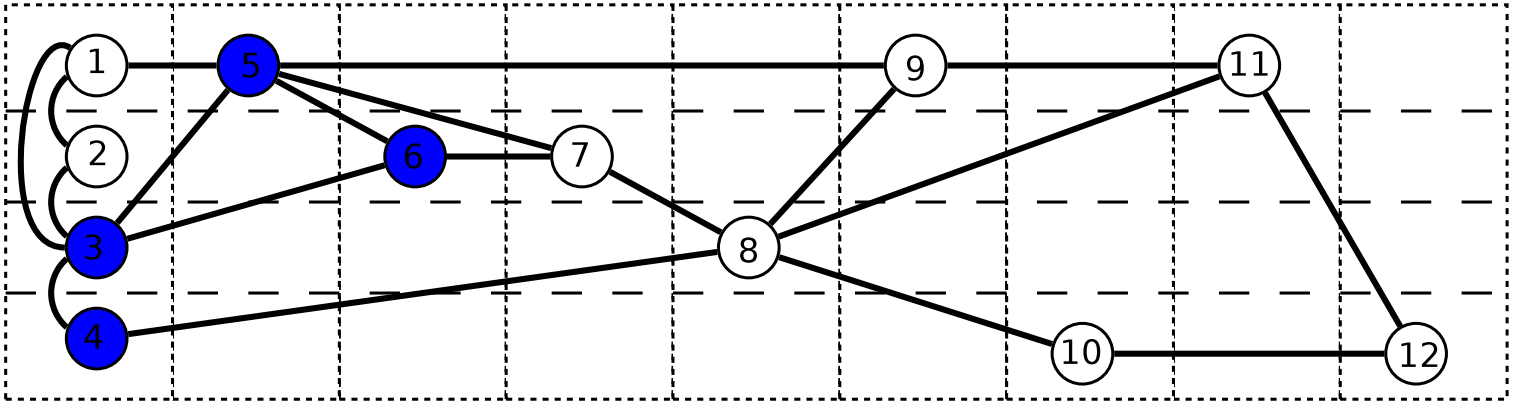
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



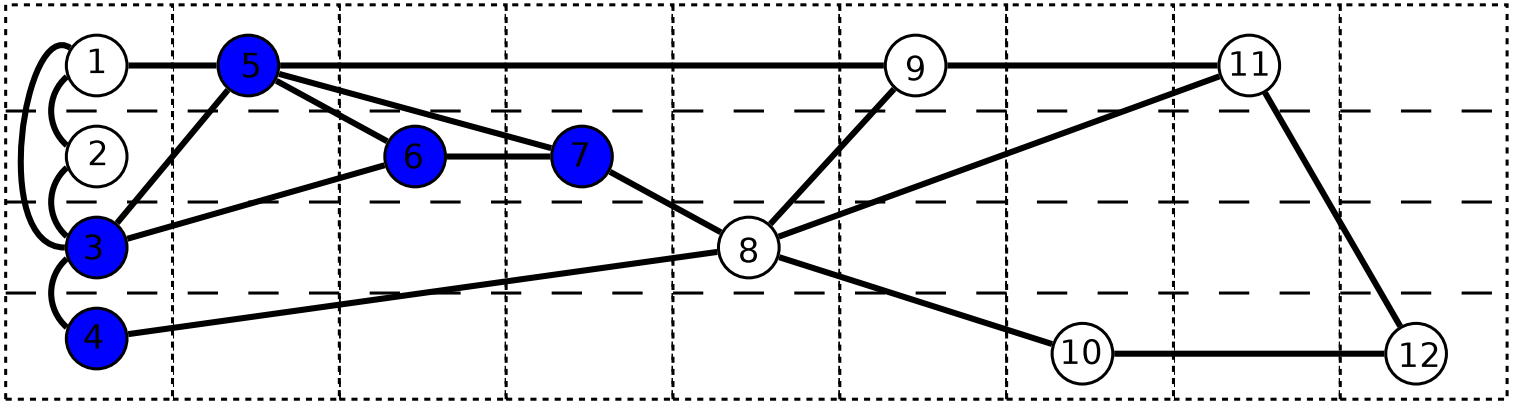
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



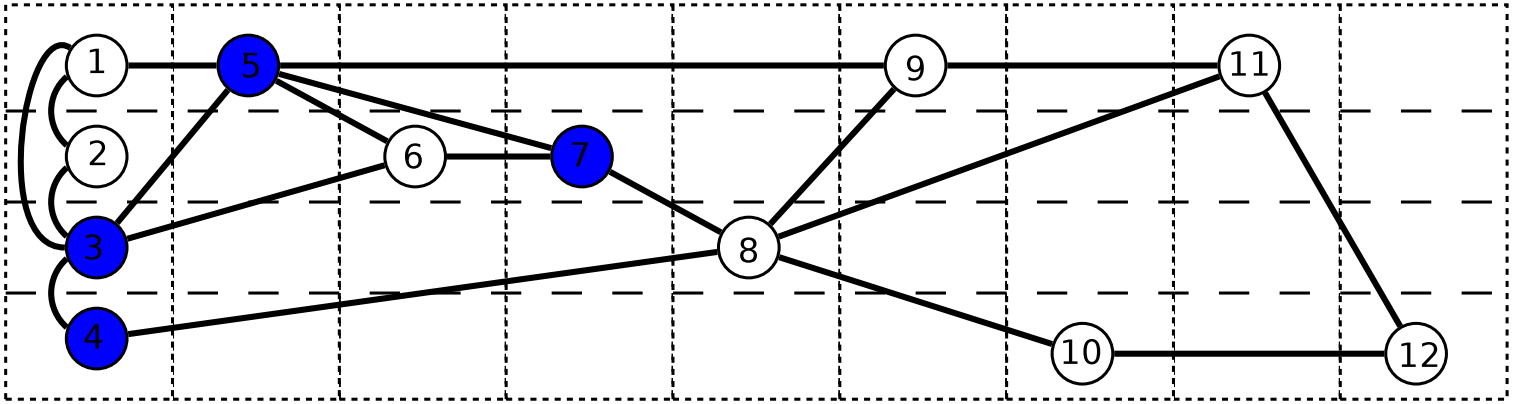
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



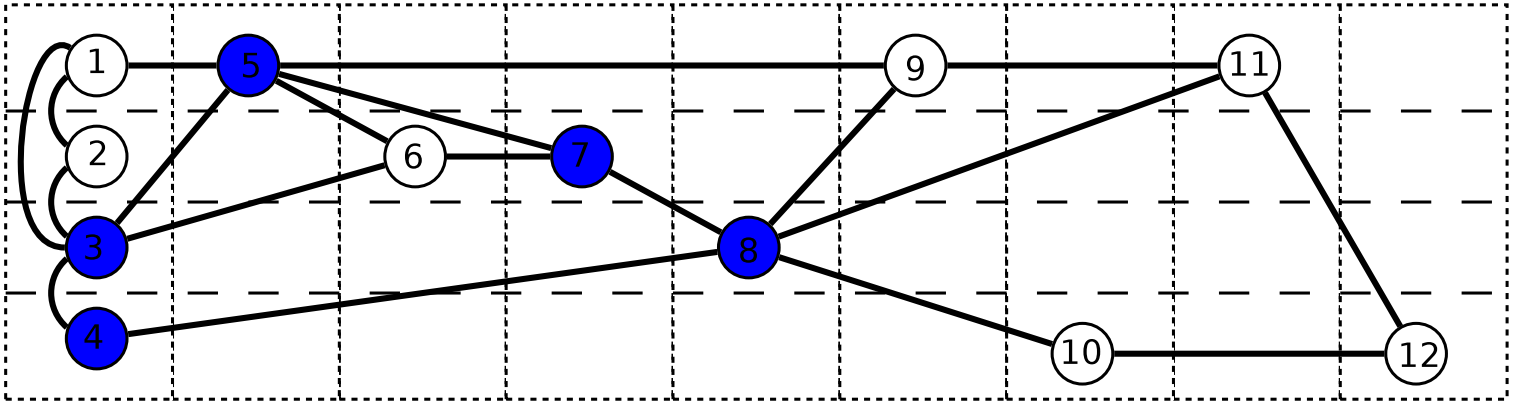
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



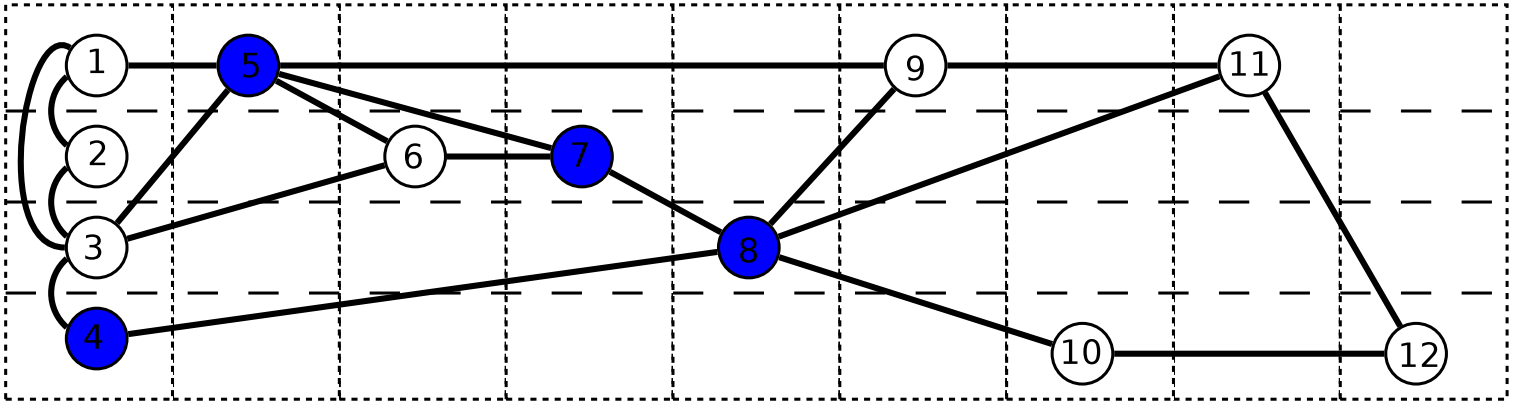
Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.



Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.

For 3-Coloring only need to remember information about boundary

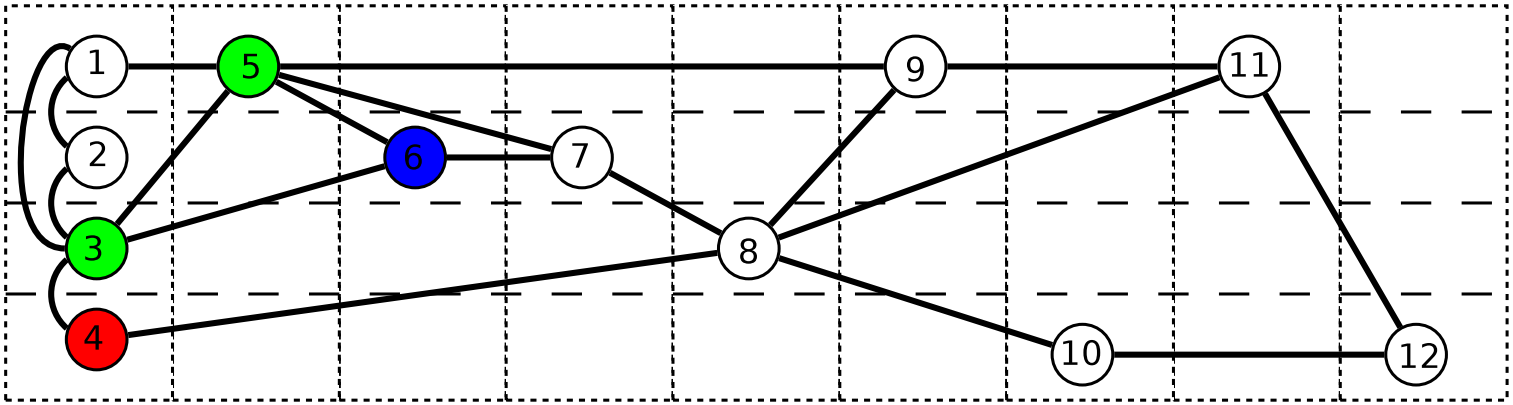
Which colorings of the boundary are properly extendible to the left?

Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.

For 3-Coloring only need to remember information about boundary

Which colorings of the boundary are properly extendible to the left?



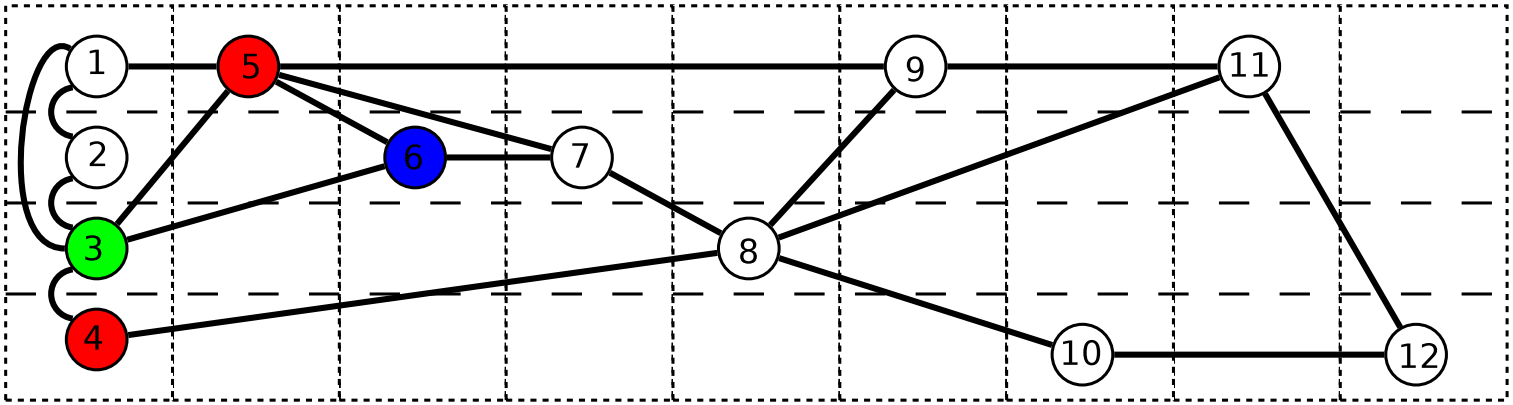
Separator: $\{3, 4, 5, 6\}$ includes tuple (3,4,5,6;No) because this coloring does not work

Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.

For 3-Coloring only need to remember information about boundary

Which colorings of the boundary are properly extendible to the left?



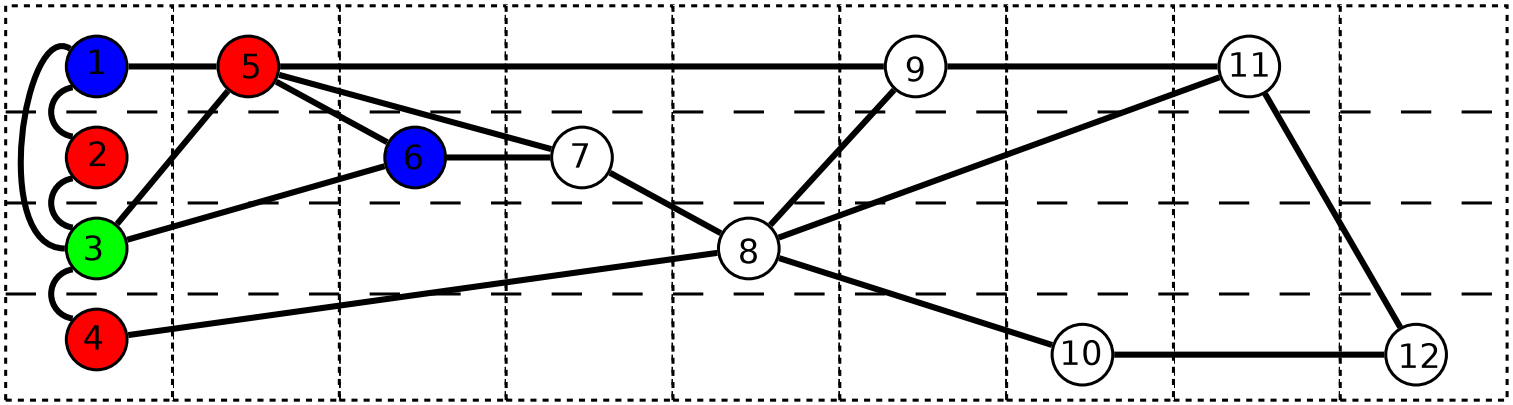
Separator: $\{3, 4, 5, 6\}$ includes tuple (3,4,5,6;Yes) because this coloring can be extended to the left

Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.

For 3-Coloring only need to remember information about boundary

Which colorings of the boundary are properly extendible to the left?



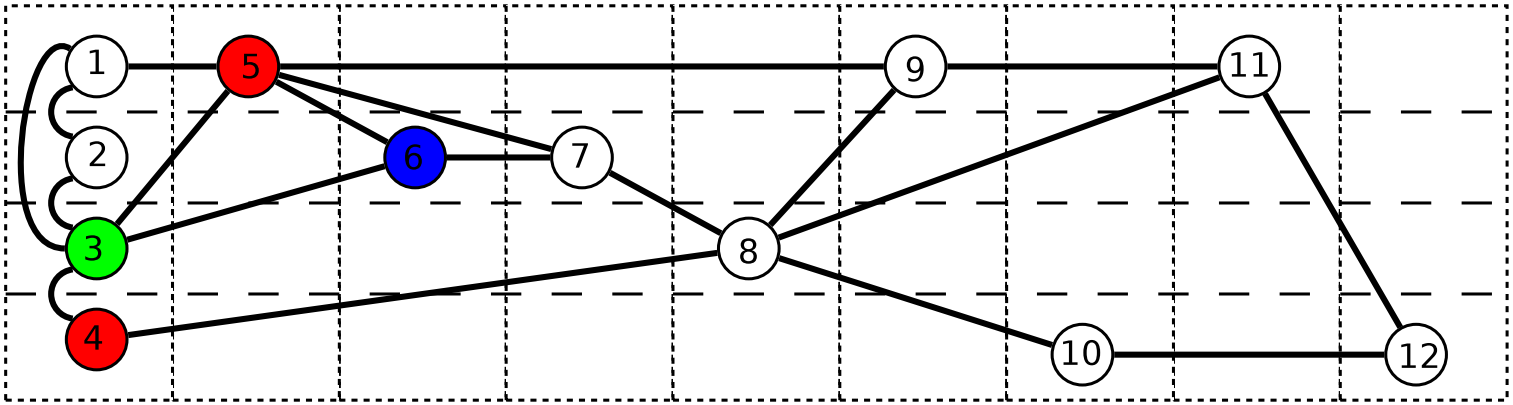
Separator: $\{3, 4, 5, 6\}$ includes tuple (3,4,5,6;Yes) because this coloring can be extended to the left

Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.

For 3-Coloring only need to remember information about boundary

Which colorings of the boundary are properly extendible to the left?



We now need to decide which are the good colorings for the separator $(3, 4, 5, 7)$.

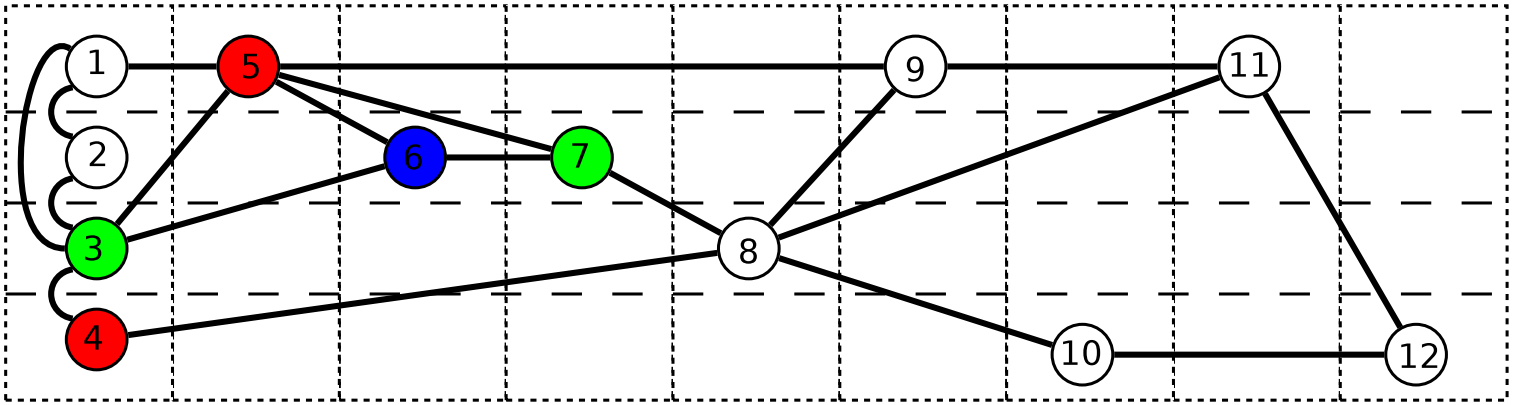
We consider each good coloring of $(3, 4, 5, 6)$.

Algorithmic view

The reason that tree/path decompositions are useful is that we have a moving boundary of small separators that "sweeps" the graph.

For 3-Coloring only need to remember information about boundary

Which colorings of the boundary are properly extendible to the left?



We now need to decide which are the good colorings for the separator $(3, 4, 5, 7)$.

We consider each good coloring of $(3, 4, 5, 6)$.

We see that $(3, 4, 5, 7)$ is a good coloring.

Important: we know the colors of all neighbors of 7.

Application: Treewidth Lower Bounds

Application: Treewidth Lower Bounds

Image credit: nano banana (including typo!)

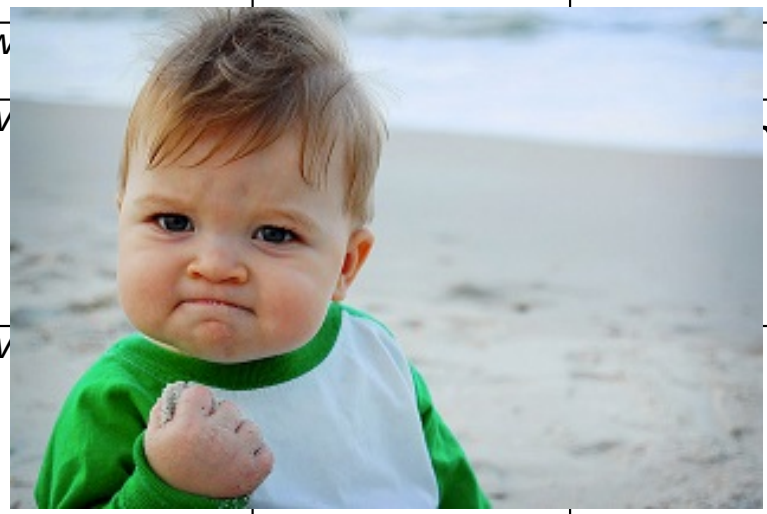
Application: Treewidth Lower Bounds

	Pathwidth	Treewidth	Cliquewidth	Ref
Ind Set	$= 2^w$	$= 2^w$	$= 2^w$	Lokshtanov,Marx,Saurabh TALG'18
Dom Set	$= 3^w$	$= 3^w$	$= 4^w$	LMS'18 and Katsikarelis, L., Paschos DAM'19
k-Col	$= k^w$	$= k^w$	$= (2^k - 2)^w$	LMS'18 and L. SIDMA'20
Max Cut	$= 2^w$	$= 2^w$	W[1]-h	LMS'18 and Fomin, Golovach, Lokshtanov, Saurabh SICOMP'10
Conn VC	$= 3^w$	$= 3^w$	$= 6^w$	Cygan et al. TALG'22, Hegerfeld, Kratsch ESA'23
Ham Cycle	$= (2+\sqrt{2})^w$	$\leq 4^w$	W[1]-h	FGLS'18, Cygan, Kratsch, Nederlof JACM'18

All lower bounds known under SETH.

Application: Treewidth Lower Bounds

	Pathwidth	Treewidth	Cliquewidth	Ref
Ind Set	$= 2^w$	$= 2^w$	$= 2^w$	Lokshtanov,Marx,Saurabh TALG'18
Dom Set	$= 3^w$	$= 3^w$	$= 4^w$	LMS'18 and Katsikarelis, L., Paschos DAM'19
k-Col	$= k^w$		$- 2)^w$	LMS'18 and L. SIDMA'20
Max Cut	$= 2^w$			LMS'18 and Fomin, Golovach, Lokshtanov, Saurabh SICOMP'10
Conn VC	$= 3^w$			Cygan et al. TALG'22, Hegerfeld, Kratsch ESA'23
Ham Cycle	$= (2+\sqrt{2})^w$	$\leq 4^w$	W[1]-h	FGLS'18, Cygan, Kratsch, Nederlof JACM'18



All lower bounds known under SETH.

Application: Treewidth Lower Bounds

	Pathwidth	Treewidth	Cliquewidth	Ref
Ind Set	$= 2^w$	$= 2^w$	$= 2^w$	Lokshtanov,Marx,Saurabh TALG'18
Dom Set	$= 3^w$	$= 3^w$	$= 4^w$	LMS'18 and Katsikarelis, L., Paschos DAM'19
k-Col	$= k^w$	$= k^w$	$= (2^k - 2)^w$	LMS'18 and L. SIDMA'20
Max Cut	$= 2^w$	$= 2^w$	W[1]-h	LMS'18 and Fomin, Golovach, Lokshtanov, Saurabh SICOMP'10
Conn VC	$= 3^w$	$= 3^w$	$= 6^w$	Cygan et al. TALG'22, Hegerfeld, Kratsch ESA'23
Ham Cycle	$= (2+\sqrt{2})^w$	$\leq 4^w$	W[1]-h	FGLS'18, Cygan, Kratsch, Nederlof JACM'18

All lower bounds known under SETH.

Motivation and Context

Motivation and Context

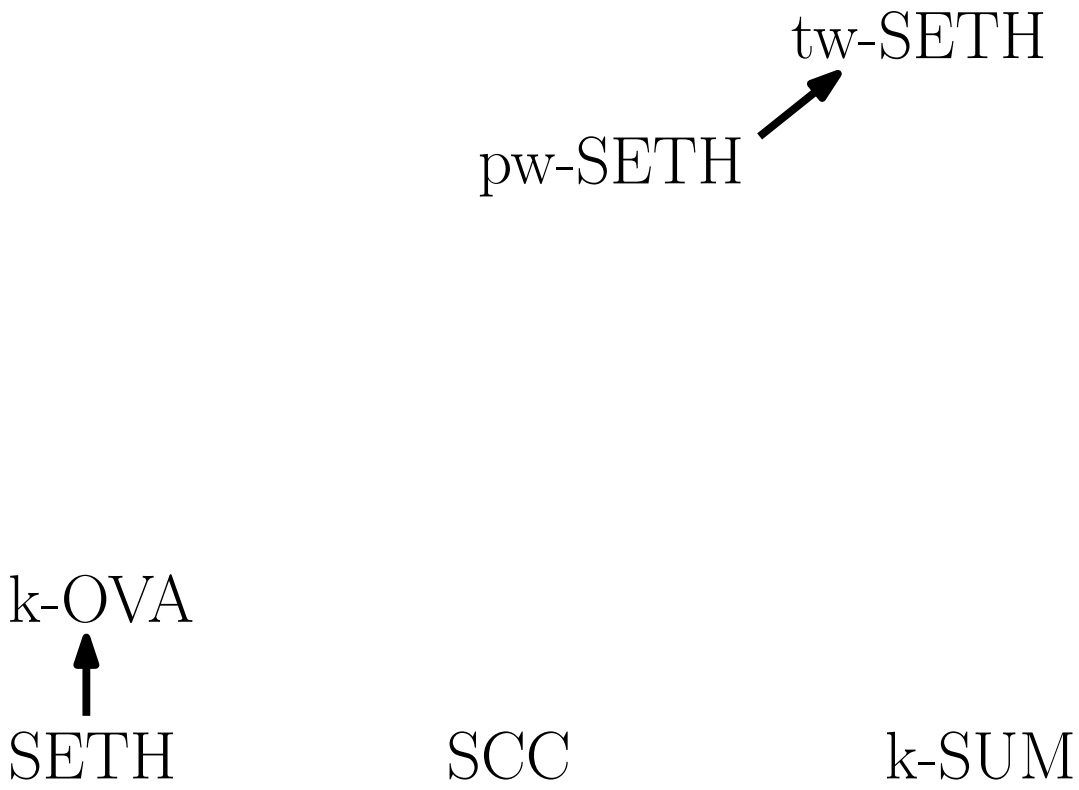
Image credit: nano banana (including typo!)

Motivation and Context

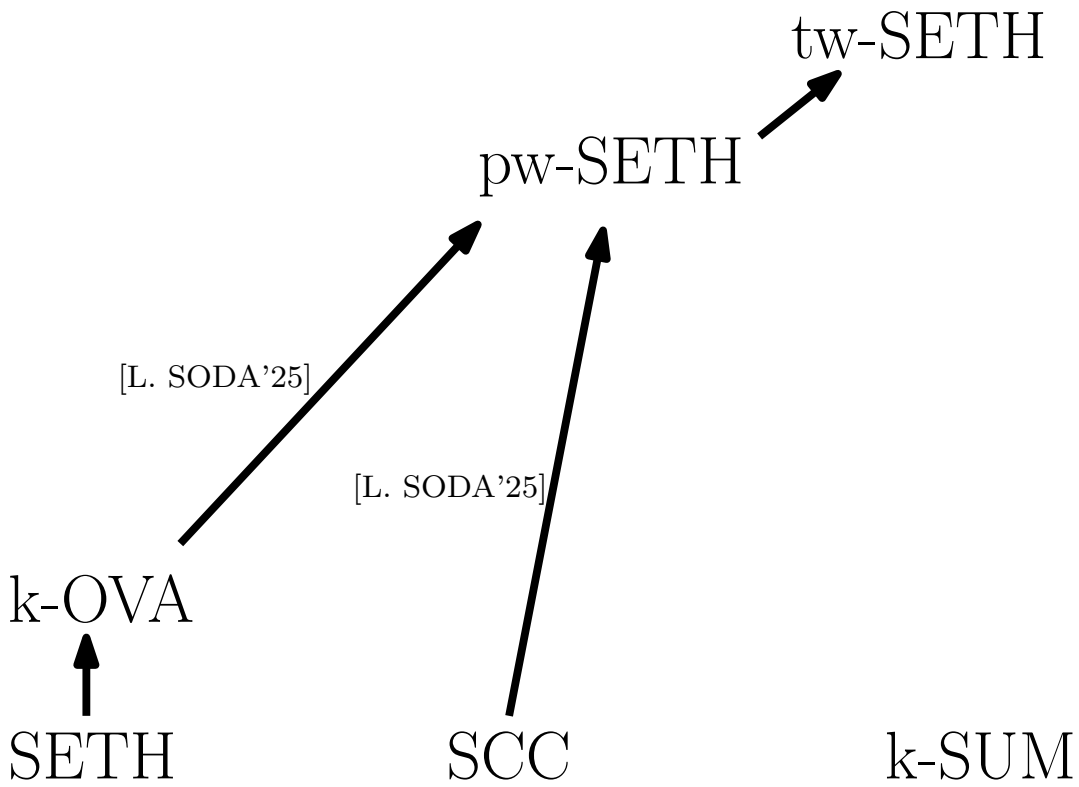
- Many different **flavors** of complexity hypotheses
  - SETH: k-SAT needs brute-force
  - Set Cover Conjecture (SCC): Set Cover needs brute-force DP
  - k-SUM/3-SUM: Meet in the middle is **optimal**
- Some more **credible** than others(?)
  - But typically few implications/relations known
- Some more **relevant** than others
  - SETH (and SCC) **main** hypotheses for **Parameterized Complexity**

Hypotheses

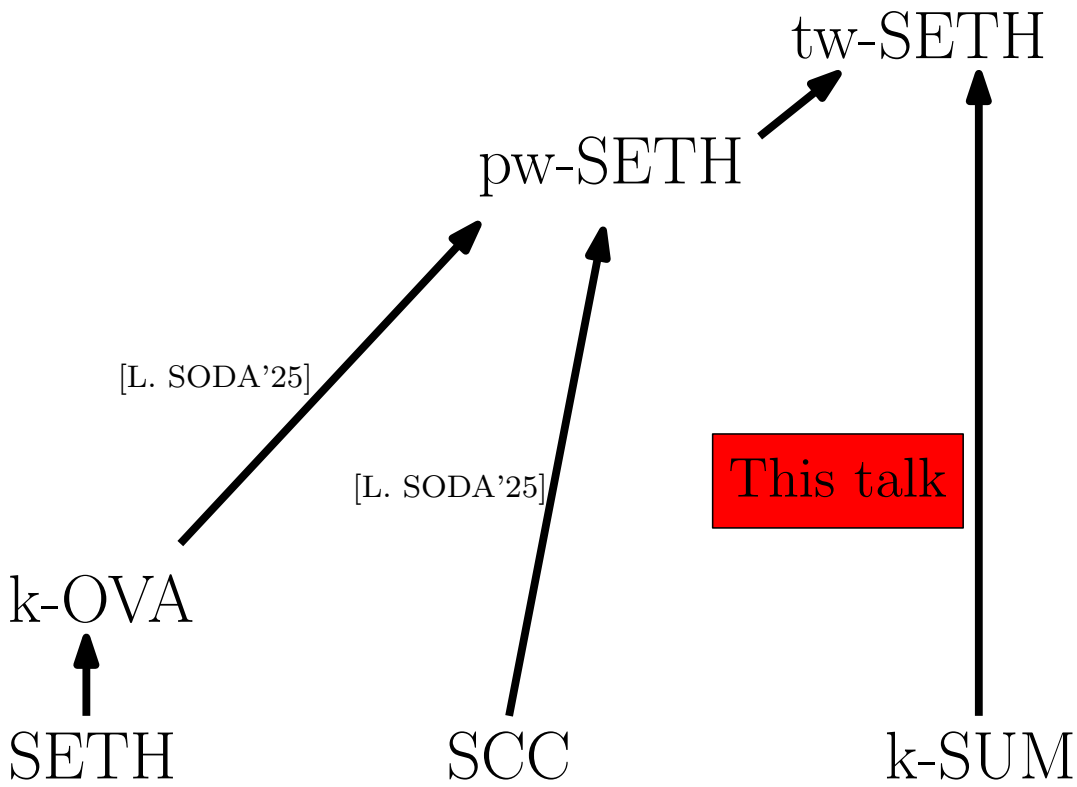
Hypotheses



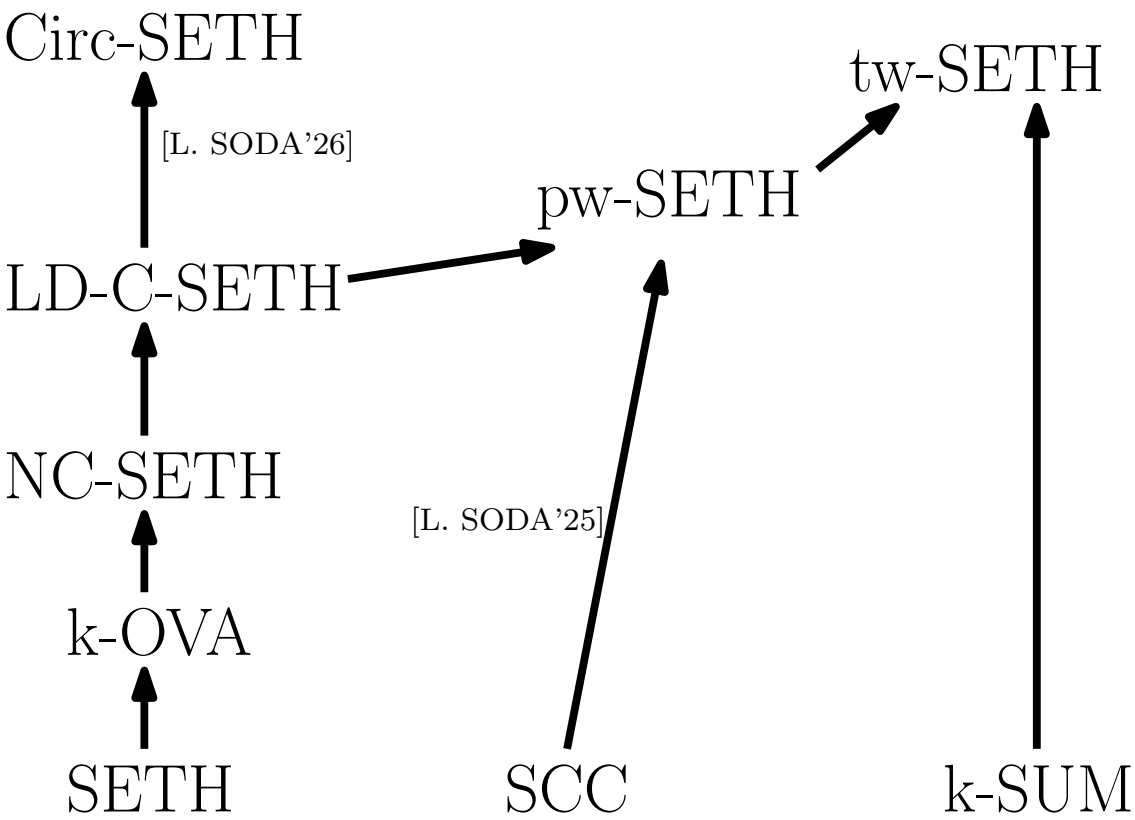
Hypotheses



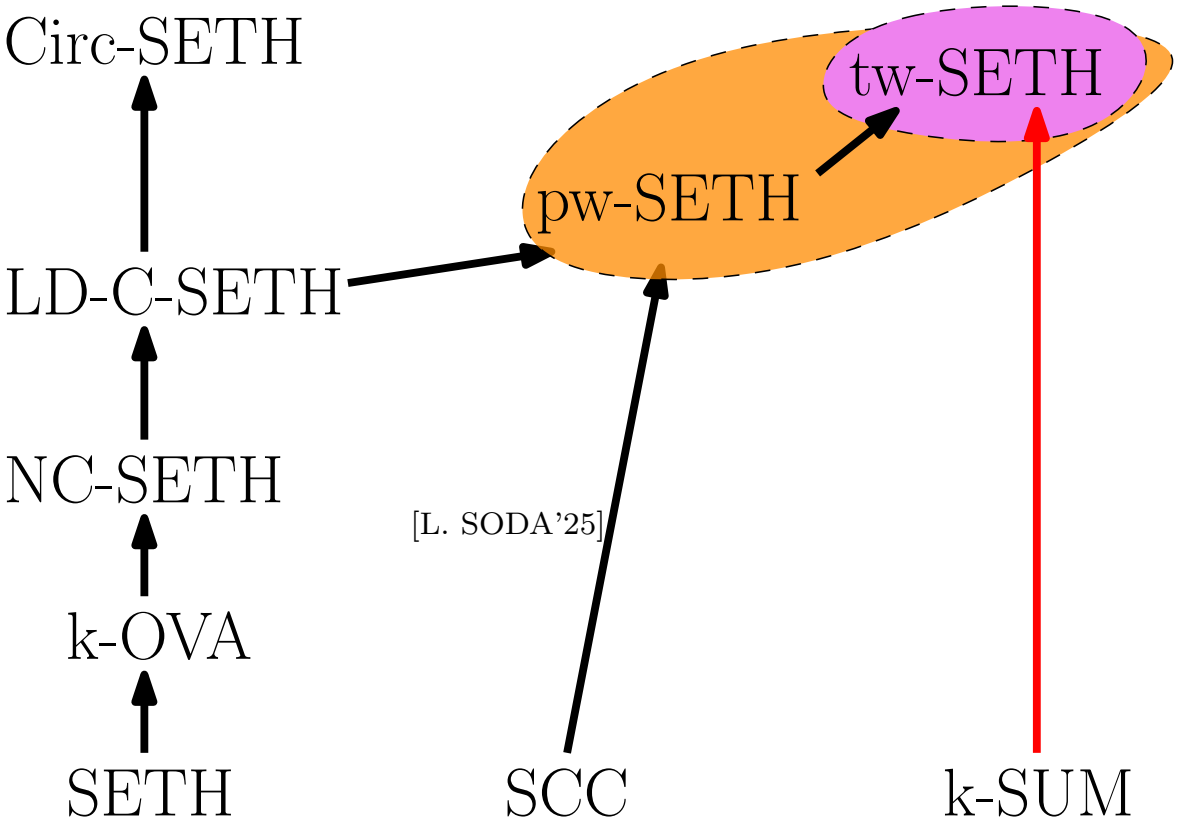
Hypotheses



Hypotheses


Circ-SETH
[L. SODA'26]
LD-C-SETH
NC-SETH
k-OVA
SETH
pw-SETH
[L. SODA'25]
SCC
tw-SETH
k-SUM

Hypotheses


Circ-SETH
LD-C-SETH
NC-SETH
k-OVA
SETH
pw-SETH
tw-SETH
[L. SODA'25]
SCC
k-SUM

Proof Sketch

Setup

Definition

In k-SUM, we are given k arrays $A_1, \ldots, A_k$ of n integers each and are asked if it is possible to select $i_1, \ldots, i_k \in [n]$ such that $\sum_j A_j[i_j] = 0$.

Setup

Definition

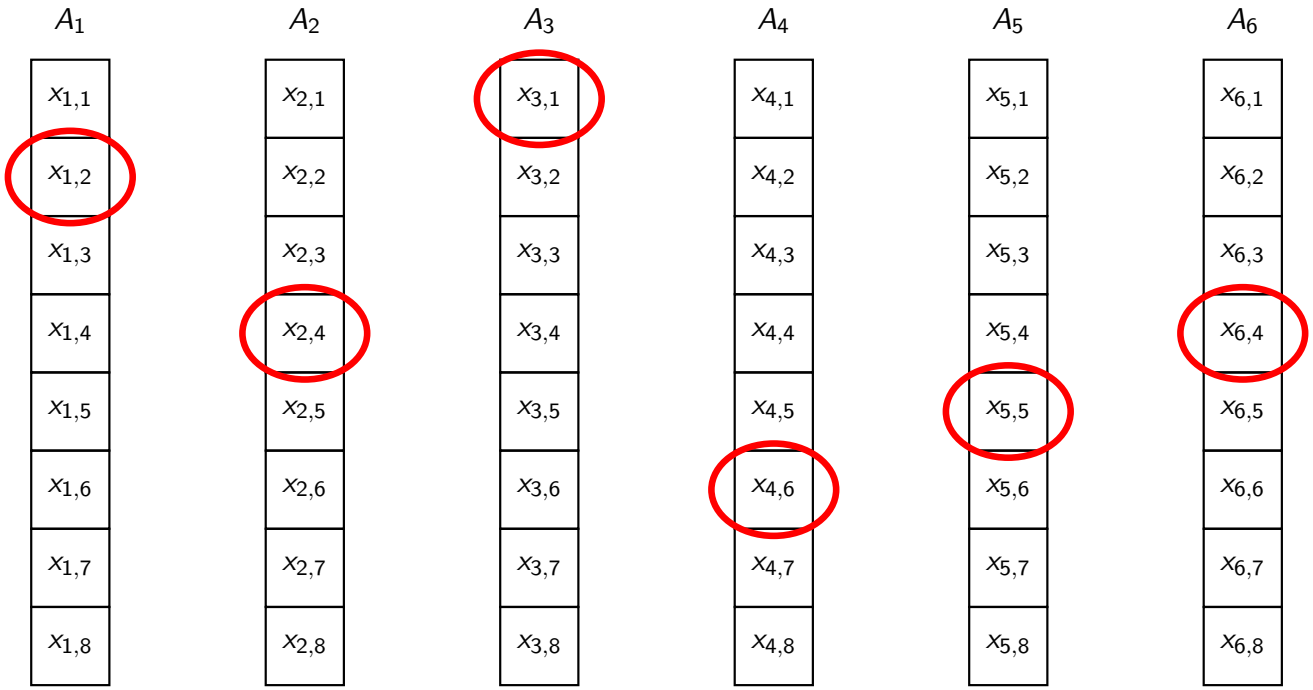
In k-SUM, we are given k arrays $A_1, \ldots, A_k$ of n integers each and are asked if it is possible to select $i_1, \ldots, i_k \in [n]$ such that $\sum_j A_j[i_j] = 0$.

A_1	A_2	A_3	A_4	A_5	A_6
$x_{1,1}$	$x_{2,1}$	$x_{3,1}$	$x_{4,1}$	$x_{5,1}$	$x_{6,1}$
$x_{1,2}$	$x_{2,2}$	$x_{3,2}$	$x_{4,2}$	$x_{5,2}$	$x_{6,2}$
$x_{1,3}$	$x_{2,3}$	$x_{3,3}$	$x_{4,3}$	$x_{5,3}$	$x_{6,3}$
$x_{1,4}$	$x_{2,4}$	$x_{3,4}$	$x_{4,4}$	$x_{5,4}$	$x_{6,4}$
$x_{1,5}$	$x_{2,5}$	$x_{3,5}$	$x_{4,5}$	$x_{5,5}$	$x_{6,5}$
$x_{1,6}$	$x_{2,6}$	$x_{3,6}$	$x_{4,6}$	$x_{5,6}$	$x_{6,6}$
$x_{1,7}$	$x_{2,7}$	$x_{3,7}$	$x_{4,7}$	$x_{5,7}$	$x_{6,7}$
$x_{1,8}$	$x_{2,8}$	$x_{3,8}$	$x_{4,8}$	$x_{5,8}$	$x_{6,8}$

Setup

Definition

In k-SUM, we are given k arrays $A_1, \ldots, A_k$ of n integers each and are asked if it is possible to select $i_1, \ldots, i_k \in [n]$ such that $\sum_j A_j[i_j] = 0$.



Sum$= 0$?

Setup

Definition

In k-SUM, we are given k arrays $A_1, \ldots, A_k$ of n integers each and are asked if it is possible to select $i_1, \ldots, i_k \in [n]$ such that $\sum_j A_j[i_j] = 0$.

High-level strategy:

- Reduce the given instance to a SAT formula of small treewidth
- Show that fast algorithm for such formulas $\Rightarrow$ fast algorithm for k-SUM.

Past success is not a guarantee

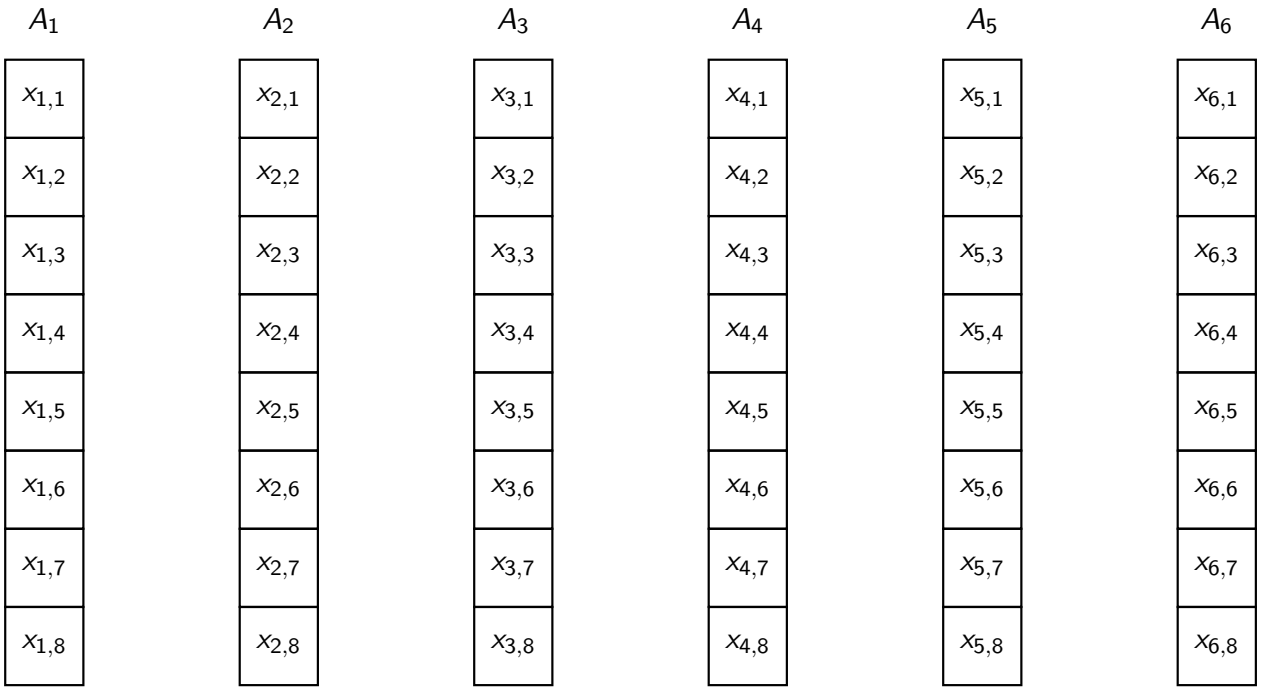
Definition

In k-OV, we are given k arrays $A_1, \ldots, A_k$ of n d-bit vectors each and are asked if it is possible to select $i_1, \ldots, i_k \in [n]$ such that $\prod_j A_j[i_j] = 0$.

Past success is not a guarantee

Definition

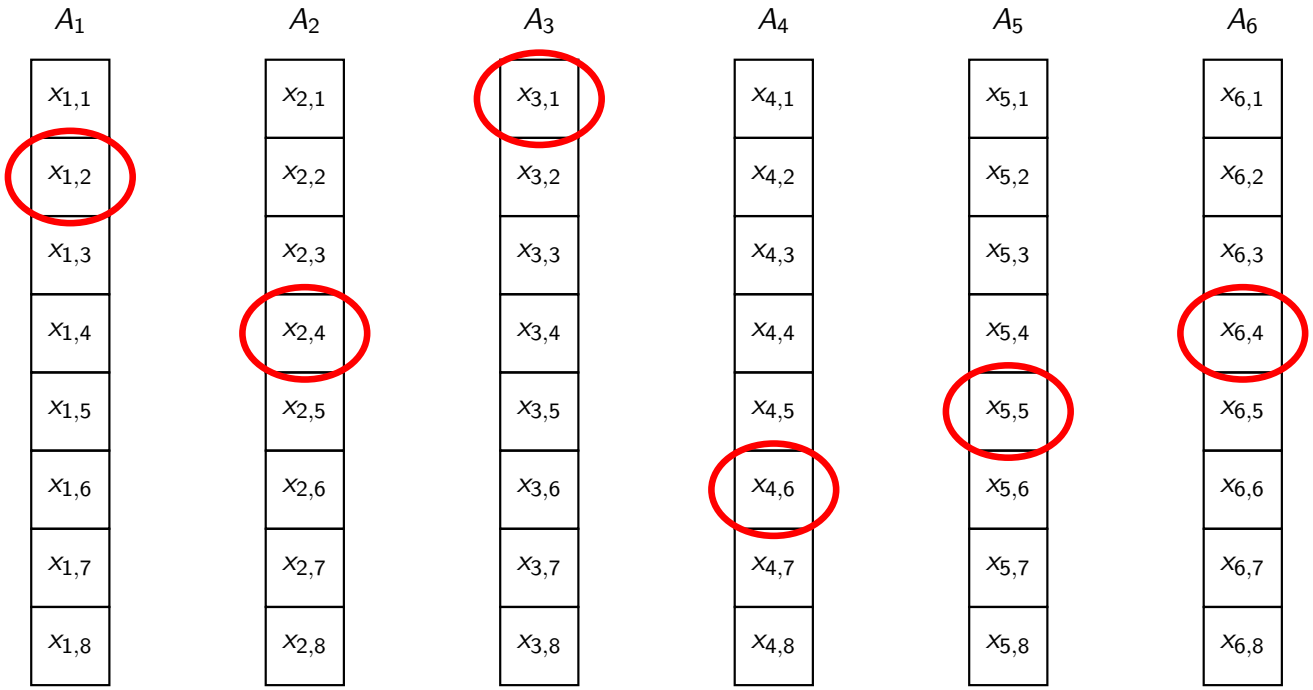
In k-OV, we are given k arrays $A_1, \ldots, A_k$ of n d-bit vectors each and are asked if it is possible to select $i_1, \ldots, i_k \in [n]$ such that $\prod_j A_j[i_j] = 0$.



Past success is not a guarantee

Definition

In k-OV, we are given k arrays $A_1, \ldots, A_k$ of n d-bit vectors each and are asked if it is possible to select $i_1, \ldots, i_k \in [n]$ such that $\prod_j A_j[i_j] = 0$.



Product$= 0$?

Past success is not a guarantee

Definition

In k-OV, we are given k arrays $A_1, \ldots, A_k$ of n d-bit vectors each and are asked if it is possible to select $i_1, \ldots, i_k \in [n]$ such that $\prod_j A_j[i_j] = 0$.

Looks similar?

Image credit: nano banana

Past success is not a guarantee

Definition

In k-OV, we are given k arrays $A_1, \ldots, A_k$ of n d-bit vectors each and are asked if it is possible to select $i_1, \ldots, i_k \in [n]$ such that $\prod_j A_j[i_j] = 0$.

- Construct a SAT formula with $k \log n$ **main** variables.
- Each group of $\log n$ variables encodes a different i_j.
- Add d groups of vars/clauses to check that each bit of the result is 0.

Past success is not a guarantee

Definition

In k-OV, we are given k arrays $A_1, \ldots, A_k$ of n d-bit vectors each and are asked if it is possible to select $i_1, \ldots, i_k \in [n]$ such that $\prod_j A_j[i_j] = 0$.

- Construct a SAT formula with $k \log n$ **main** variables.
- Each group of $\log n$ variables encodes a different i_j.
- Add d groups of vars/clauses to check that each bit of the result is 0.
- Pathwidth is (essentially) $k \log n$
- $(2-\varepsilon)^{\mathrm{pw}} n^{O(1)}$ algorithm for $\mathrm{SAT} \Rightarrow n^{(1-\varepsilon)k}$ algorithm for k-OV

[L. SODA'25]

Past success is not a guarantee

Definition

In k-OV, we are given k arrays $A_1, \ldots, A_k$ of n d-bit vectors each and are asked if it is possible to select $i_1, \ldots, i_k \in [n]$ such that $\prod_j A_j[i_j] = 0$.

- Construct a SAT formula with $k \log n$ **main** variables.
- Each group of $\log n$ variables encodes a different i_j.
- Add d groups of vars/clauses to check that each bit of the result is 0.
- Pathwidth is (essentially) $k \log n$
- $(2-\varepsilon)^{\mathrm{pw}} n^{O(1)}$ algorithm for $\mathrm{SAT} \Rightarrow n^{(1-\varepsilon)k}$ algorithm for k-OV

  [L. SODA'25]

- Great! Why don't we do the same for k-SUM?

Past success is not a guarantee

Definition

In k-OV, we ar… …tors each and are asked if it is po… …$_j A_j[i_j] = 0$.

- Construct …s.
- Each grou…
- Add d gro… of the result is 0.
- Pathwidth …
- $(2-\varepsilon)^{\mathrm{pw}} n$… …m for k-OV [L. SODA'25]
- Great! Wh…

Meet in the middle

Meet in the middle

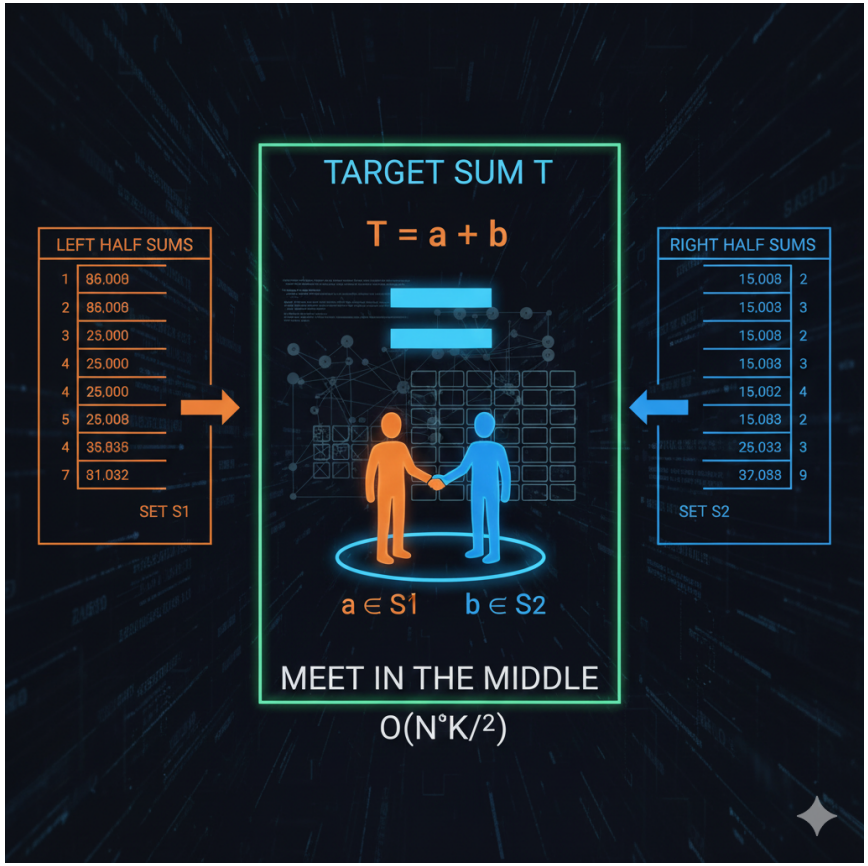
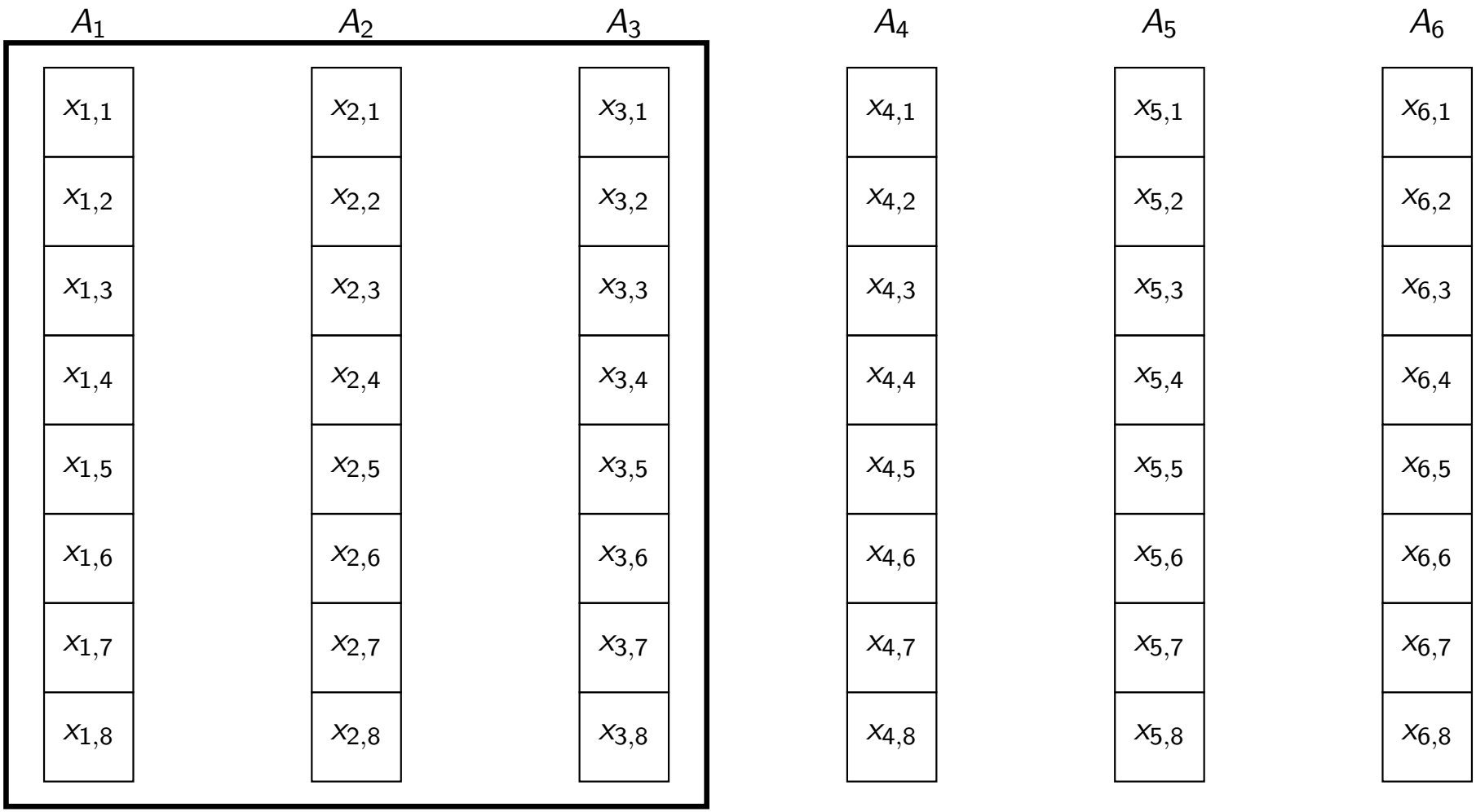


Image credit: nano banana (including **nonsensical** math)

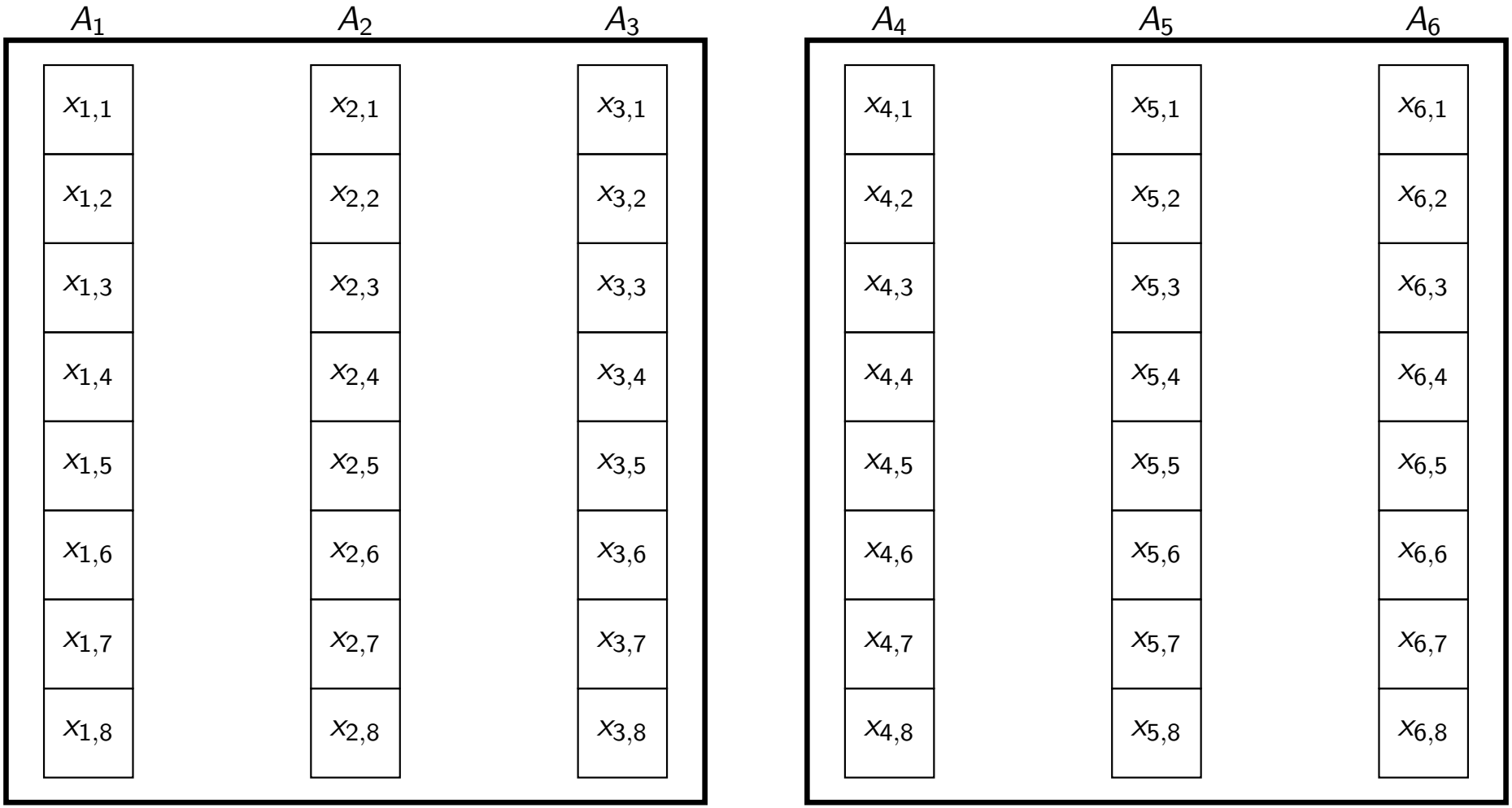
Meet in the middle

A_1	A_2	A_3	A_4	A_5	A_6
$x_{1,1}$	$x_{2,1}$	$x_{3,1}$	$x_{4,1}$	$x_{5,1}$	$x_{6,1}$
$x_{1,2}$	$x_{2,2}$	$x_{3,2}$	$x_{4,2}$	$x_{5,2}$	$x_{6,2}$
$x_{1,3}$	$x_{2,3}$	$x_{3,3}$	$x_{4,3}$	$x_{5,3}$	$x_{6,3}$
$x_{1,4}$	$x_{2,4}$	$x_{3,4}$	$x_{4,4}$	$x_{5,4}$	$x_{6,4}$
$x_{1,5}$	$x_{2,5}$	$x_{3,5}$	$x_{4,5}$	$x_{5,5}$	$x_{6,5}$
$x_{1,6}$	$x_{2,6}$	$x_{3,6}$	$x_{4,6}$	$x_{5,6}$	$x_{6,6}$
$x_{1,7}$	$x_{2,7}$	$x_{3,7}$	$x_{4,7}$	$x_{5,7}$	$x_{6,7}$
$x_{1,8}$	$x_{2,8}$	$x_{3,8}$	$x_{4,8}$	$x_{5,8}$	$x_{6,8}$

Meet in the middle



Meet in the middle



Meet in the middle

Algorithm:

- Construct two **large** arrays (size $n^{k/2}$):
  - L has all sums constructible from $A_1, \ldots, A_{k/2}$
  - R has all sums constructible from $A_{k/2+1}, \ldots, A_k$
- Check if exists $x \in L, y \in R$ such that $x + y = 0$
  - Equivalently: check if $L \cap R \neq \emptyset$
- Can be done in time $\widetilde{O}(n^{k/2})$.

Meet in the middle

Algorithm:

- Construct two **large** arrays (size $n^{k/2}$):
  - L has all sums constructible from $A_1, \ldots, A_{k/2}$
  - R has all sums constructible from $A_{k/2+1}, \ldots, A_k$
- Check if exists $x \in L, y \in R$ such that $x + y = 0$
  - Equivalently: check if $L \cap R \neq \emptyset$
- Can be done in time $\widetilde{O}(n^{k/2})$.

- **Consequence:** we need a reduction to SAT which produces a formula of width $\frac{k}{2} \log n$ (**not** $k \log n$)
- Running standard DP on this formula should be **as fast as** meet-in-the middle
  - Otherwise, a small improvement on the SAT algorithm will not beat meet-in-the-middle...

Machines and intuition

Machine Characterizations of pw-SETH and tw-SETH

- pw-SETH: Non-deterministic TM with space $\mathrm{pw} + O(\log n)$ [Iwata Yoshida ESA'15]
- tw-SETH: Alternating TM with space $\mathrm{pw} + O(\log n)$ and $O(\log n)$ alternations

Machines and intuition

Machine Characterizations of pw-SETH and tw-SETH

- pw-SETH: Non-deterministic TM with space $\mathrm{pw} + O(\log n)$ [Iwata Yoshida ESA'15]
- tw-SETH: Alternating TM with space $\mathrm{pw} + O(\log n)$ and $O(\log n)$ alternations

Is k-SUM in NL?

- Yes, but needs space $k \log n$...

Machines and intuition

nd tw-SETH

n space $\mathrm{pw} + O(\log n)$ [Iwata

$\mathrm{pw} + O(\log n)$ and $O(\log n)$

Low Space NDTM for k-SUM

How little memory can an NDTM use to solve k-SUM?

- Approach 1: guess solution indices, then verify
  - Need $k \log n$ bits to store solution...

Low Space NDTM for k-SUM

How little memory can an NDTM use to solve k-SUM?

- Approach 1: guess solution indices, then verify
  - Need $k \log n$ bits to store solution...
- Approach 2: keep track of current sum, for each element non-deterministically check whether to take it.
  - Space needed: $\log W + O(\log n)$, where W is max absolute value
  - Is this better??

Hash me if you can!

Strategy:

- Find a hash function $h : \mathbb{N} \to \{-W, \ldots, W\}$ such that
  - h is (almost-)linear: $h(x + y) \approx h(x) + h(y)$
  - h is "random-looking"
  - W is **small** (ideally $n^{k/2}$)

Hash me if you can!

Strategy:

- Find a hash function $h : \mathbb{N} \to \{-W, \ldots, W\}$ such that
  - h is (almost-)linear: $h(x + y) \approx h(x) + h(y)$
  - h is "random-looking"
  - W is **small** (ideally $n^{k/2}$)
- Almost-linearity guarantees no false negatives
- Good hash function $\Rightarrow$ no false positives
- NDTM needs $\log W + O(\log n) = \frac{k}{2} \log n + O(\log n)$ space
- **Therefore:** k-SUM$\Rightarrow$ pw-SETH

Hash me if you can!

such that

$O(\log n)$ space

Doesn't work!

How much can you hash?

What is the smallest viable value of W?

- Recall $L \cup R$ has size $O(n^{k/2})$
- We need to hash preserving the answer to $L \cap R \neq \emptyset$?
- There are $|L| \times |R|$ possible **bad** pairs

How much can you hash?

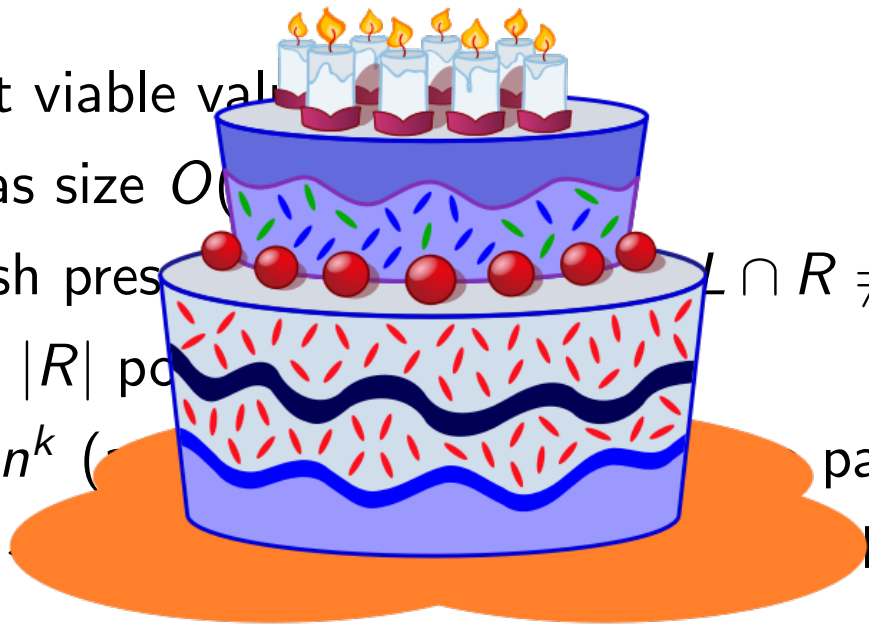
What is the smallest viable value of W?

- Recall $L \cup R$ has size $O(n^{k/2})$
- We need to hash preserving the answer to $L \cap R \neq \emptyset$?
- There are $|L| \times |R|$ possible **bad** pairs
- If we set $W > n^k$ (and h is "random") whp no pair collides
- However, if $W < n^k$, we will get false positives whp!!

How much can you hash?

What is the smallest viable val

- Recall $L \cup R$ has size $O($
- We need to hash pres $L \cap R \neq \emptyset$?
- There are $|L| \times |R|$ po
- If we set $W > n^k$ (pair collides
- However, if W hp!!



Birthday Paradox!

Perfect Hashing with Arthur and Merlin

Perfect Hashing with Arthur and Merlin

Image credit: ChatGPT

Perfect Hashing with Arthur and Merlin

Perfect Hashing:

- Can I hash N items into a range of size $O(N)$ without collisions?
  - In one shot, **No**, because of birthday paradox problem we saw
  - In two steps, **Yes**!
    - ⋆ (Fredman, Komlós, Szemerédi J.ACM'84)
  - Key intuition: run through a first hash function, each bucket will have few collisions, run through a second hash function to sort them out.

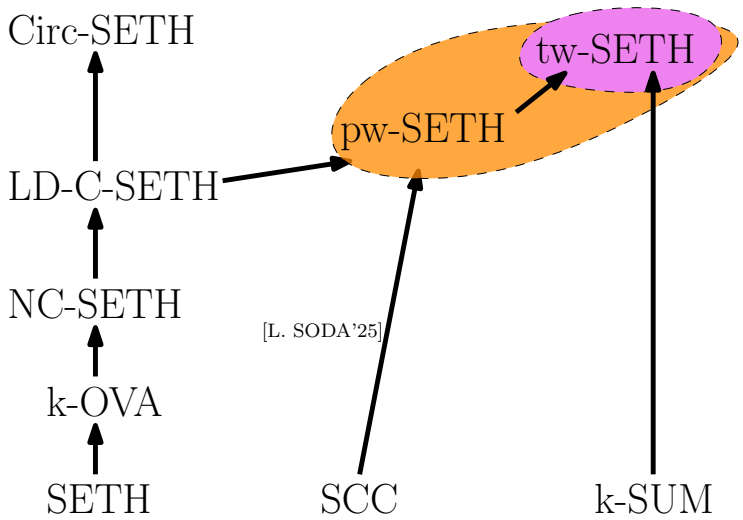
Perfect Hashing with Arthur and Merlin

Protocol:

- Apply an almost-linear hash $h : \mathbb{N} \to \{-W, \ldots, W\}$, with $W = n^{k/2}$
  - Will almost surely have **false positives**
- Challenge the prover to give $x \in L, y \in R$ with $h(x) = h(y)$
  - Prover **commits** to $h(x) = h(y)$, which we **store in memory**
- Pick a second hash function $h' : \mathbb{N} \to \{-w, \ldots, w\}$, where w is large enough to perfectly hash **a single bucket**
- Challenge prover to give $x \in L, y \in R$ with
  - $h(x) = h(y)$ as before
  - $h'(x) = h'(y)$
- **Key Intution**: Treewidth $\Leftrightarrow$ Alternating TM $\Leftrightarrow$ Interactive Proof

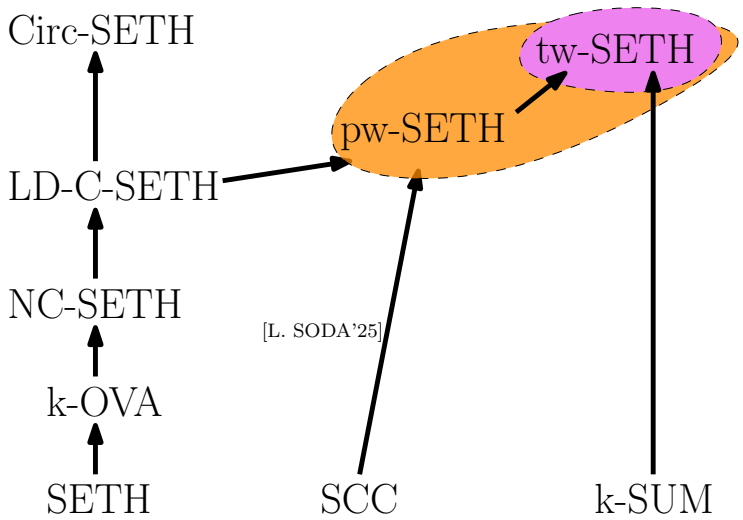
Open Questions

Open Questions



- k-SUM⇒pw-SETH?
- Other connections?

Open Questions



- k-SUM⇒pw-SETH?
- Other connections?

Thank you!