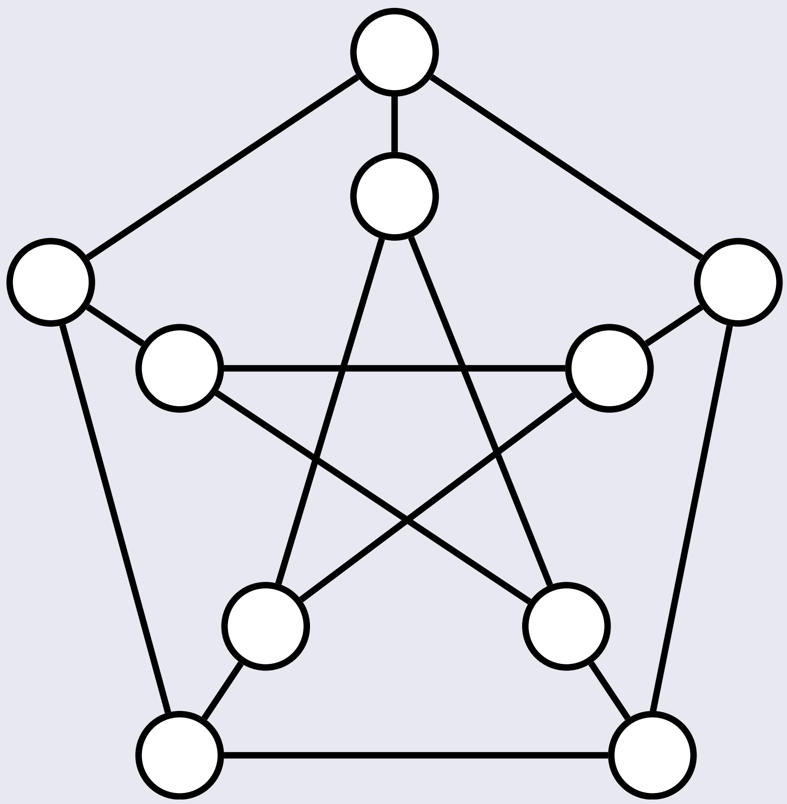
Finer Tight Bounds for Coloring on Clique-width

Michael Lampis
LAMSADE
Université Paris Dauphine

ICALP 2018

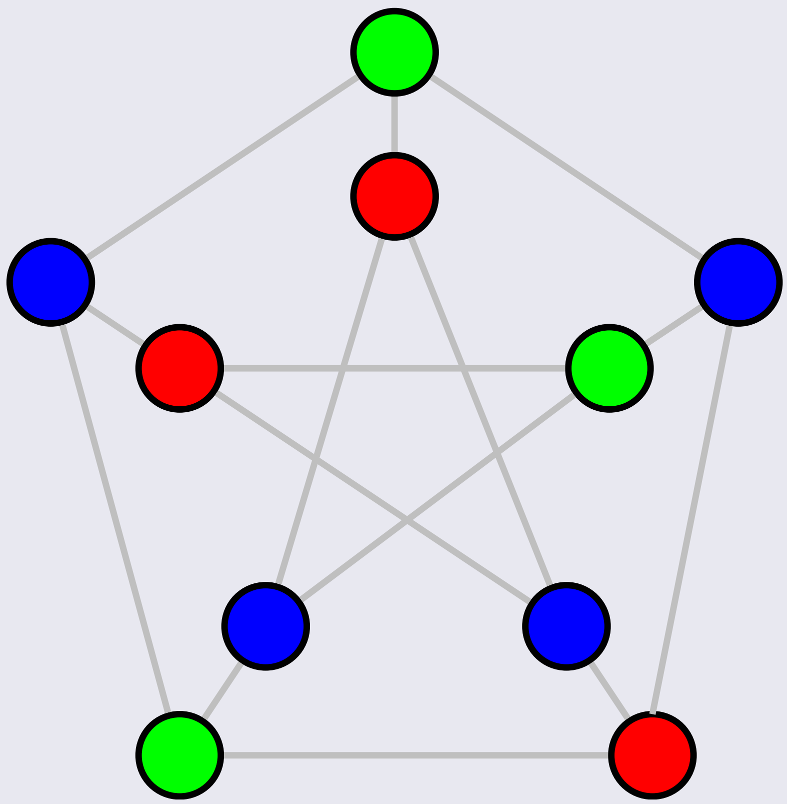
Coloring



Input:
Graph $G = (V, E)$
n vertices
k colors

Question:
Can we partition V into k independent sets?

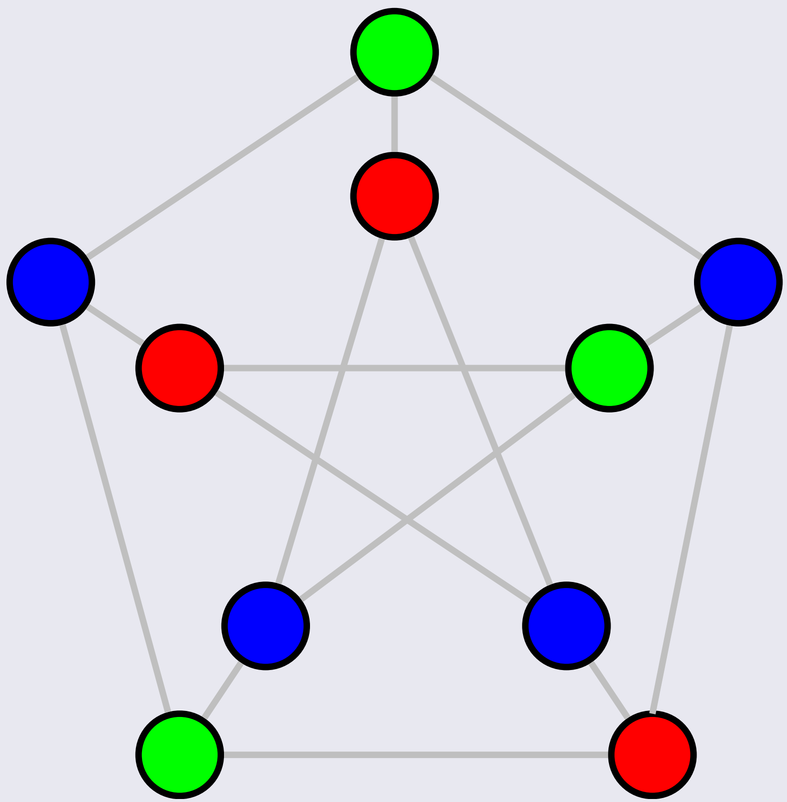
Coloring



Input:
Graph $G = (V, E)$
n vertices
k colors

Question:
Can we partition V into k independent sets?

Coloring



Input:
Graph $G = (V, E)$
n vertices
k colors

Question:
Can we partition V into k independent sets?

Note: For the rest of this talk, k denotes the number of **colors**.

Problem NP-hard for any $k \geq 3$:
We look at graphs with restricted structure.

Finer Tight Bounds?

- What is a “finer” tight bound?

Finer Tight Bounds?

- Tight bound: complexity-theoretic bound that “matches” running time of **existing** algorithm.
- Finer bounds:
  - Increased “granularity”.
  - More precise about secondary parameters.

Finer Tight Bounds?

- Tight bound: complexity-theoretic bound that "matches" running time of **existing** algorithm.
- Finer bounds:
  - Increased "granularity".
  - More precise about secondary parameters.

Coloring

- We know the "correct" complexity of Coloring for clique-width
  - $\ldots \approx k^{2^w}$ (more details in a bit)
- This bound is only tight for k sufficiently large.
- What is the **exact** complexity of 3-coloring, 4-coloring for clique-width?

Finer Tight Bounds?

- Tight bound: complexity-theoretic bound that “matches” running time of **existing** algorithm.
- Finer bounds:
  - Increased “granularity”.
  - More precise about secondary parameters.

Coloring

- We know the “correct” complexity of Coloring for clique-width
  - $\ldots \approx k^{2^w}$ (more details in a bit)
- This bound is only tight for k sufficiently large.
- What is the **exact** complexity of 3-coloring, 4-coloring for clique-width?

In this talk we show that, under the SETH, the **correct** complexity of k-Coloring for clique-width is

Finer Tight Bounds?

- Tight bound: complexity-theoretic bound that “matches” running time of **existing** algorithm.
- Finer bounds:
  - Increased “granularity”.
  - More precise about secondary parameters.

Coloring

- We know the “correct” complexity of Coloring for clique-width
  - $\ldots \approx k^{2^w}$ (m
- This bound is c
- What is the **exa** g for clique-width?

In this talk we **ect** complexity of k-Coloring fo

Finer Tight Bounds?

- Tight bound: complexity-theoretic bound that "matches" running time of **existing** algorithm.
- Finer bounds:
  - Increased "granularity".
  - More precise about secondary parameters.

Coloring

- We know the "correct" complexity of Coloring for clique-width
  - $\ldots \approx k^{2^w}$ (more details in a bit)
- This bound is only tight for k sufficiently large.
- What is the **exact** complexity of 3-coloring, 4-coloring for clique-width?

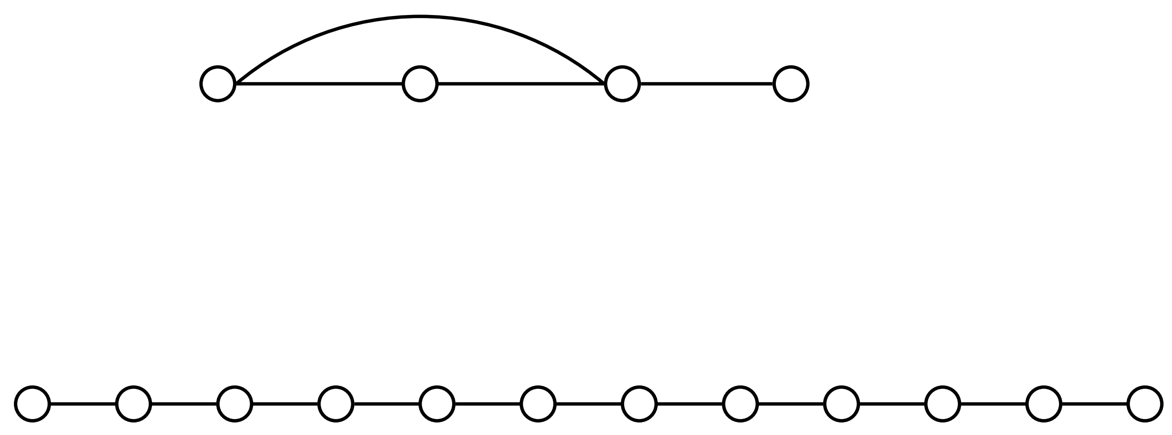
In this talk we show that, under the SETH, the **correct** complexity of k-Coloring for clique-width is c_k^w.

The story so far: Treewidth

Consider this (**very very special**) class of graphs of treewidth w:

- The graph consists of a long path

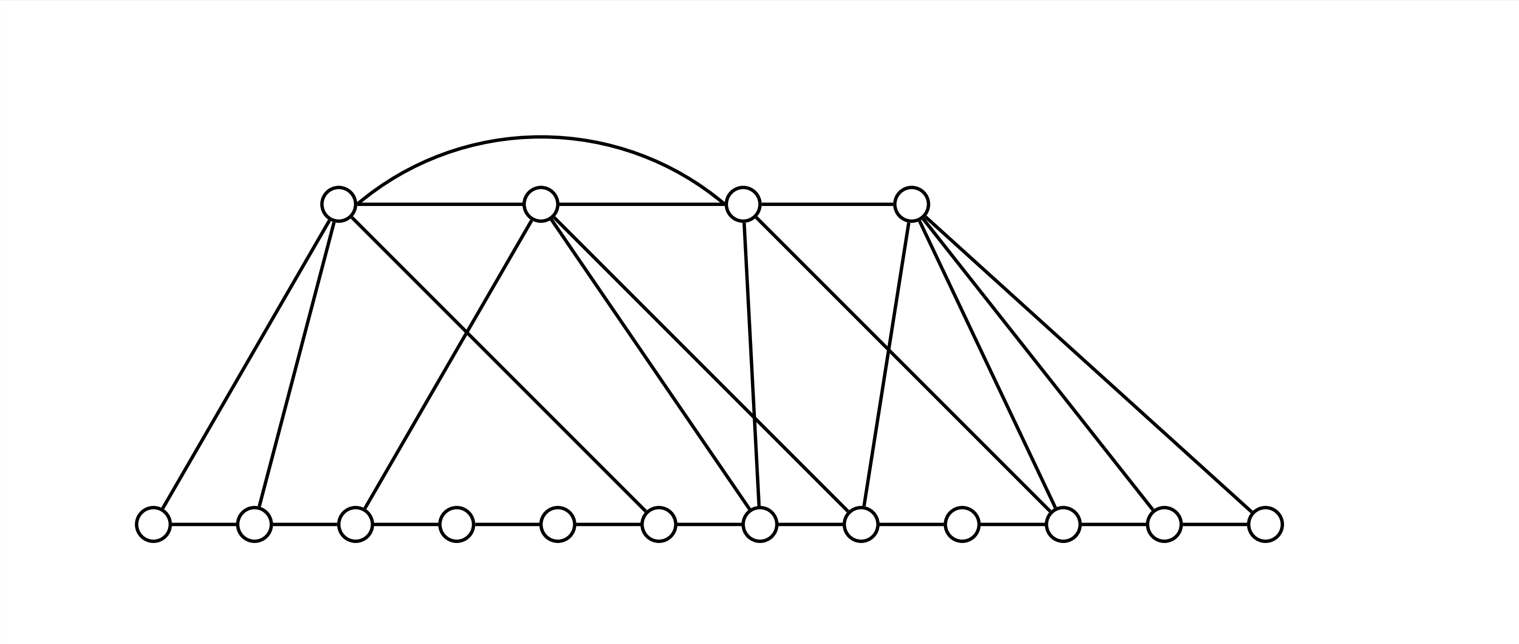
The story so far: Treewidth



Consider this (**very very special**) class of graphs of treewidth w:

- The graph consists of a long path
- w extra vertices, arbitrarily connected to each other

The story so far: Treewidth

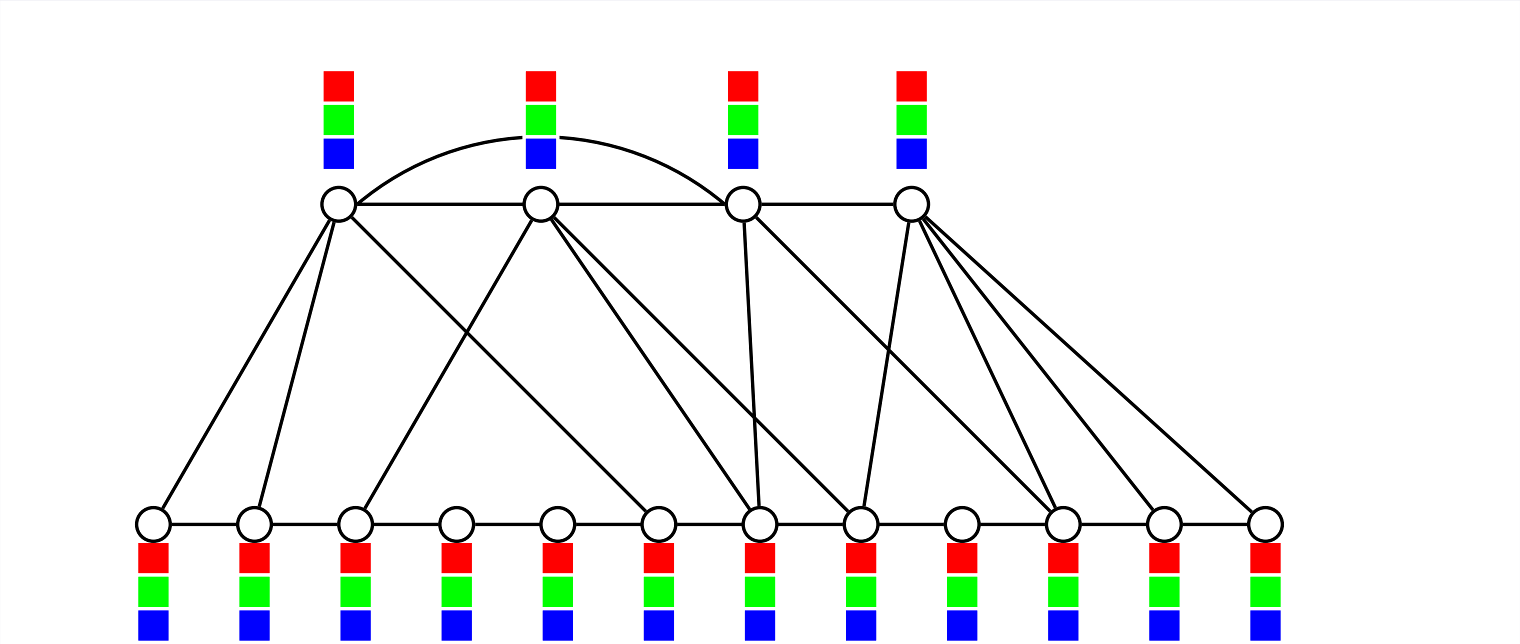


Consider this (**very very special**) class of graphs of treewidth w:

- The graph consists of a long path
- w extra vertices, arbitrarily connected to each other
- and arbitrary edges between these two parts

Interesting case: $w << n$.

The story so far: Treewidth



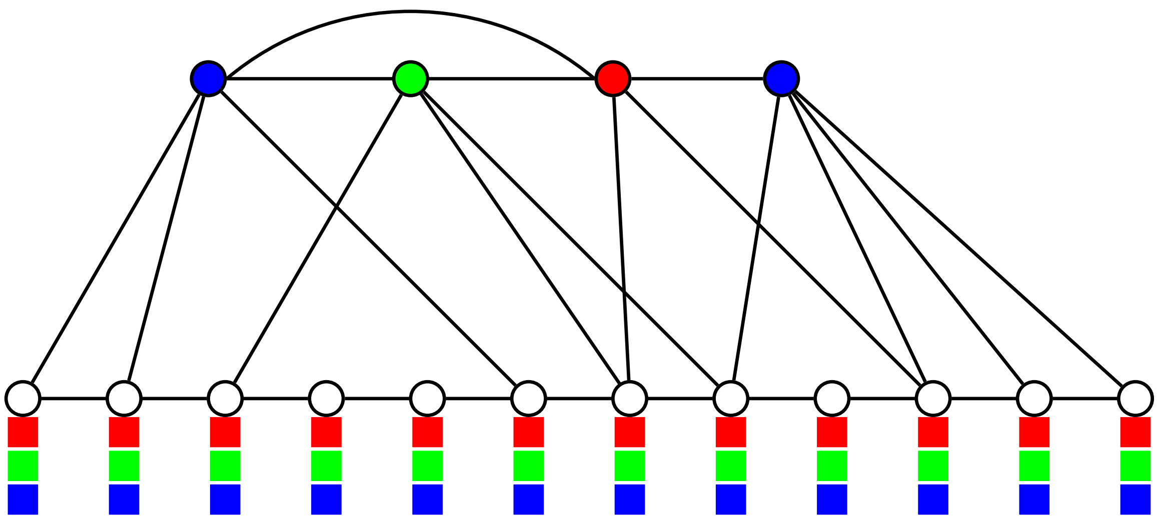
Consider this (**very very special**) class of graphs of treewidth w:

- The graph consists of a long path

3-Coloring algorithm on these graphs:

- Guess a valid coloring of the w non-path vertices
- Try to extend it to a coloring of the whole graph (easy!)

The story so far: Treewidth



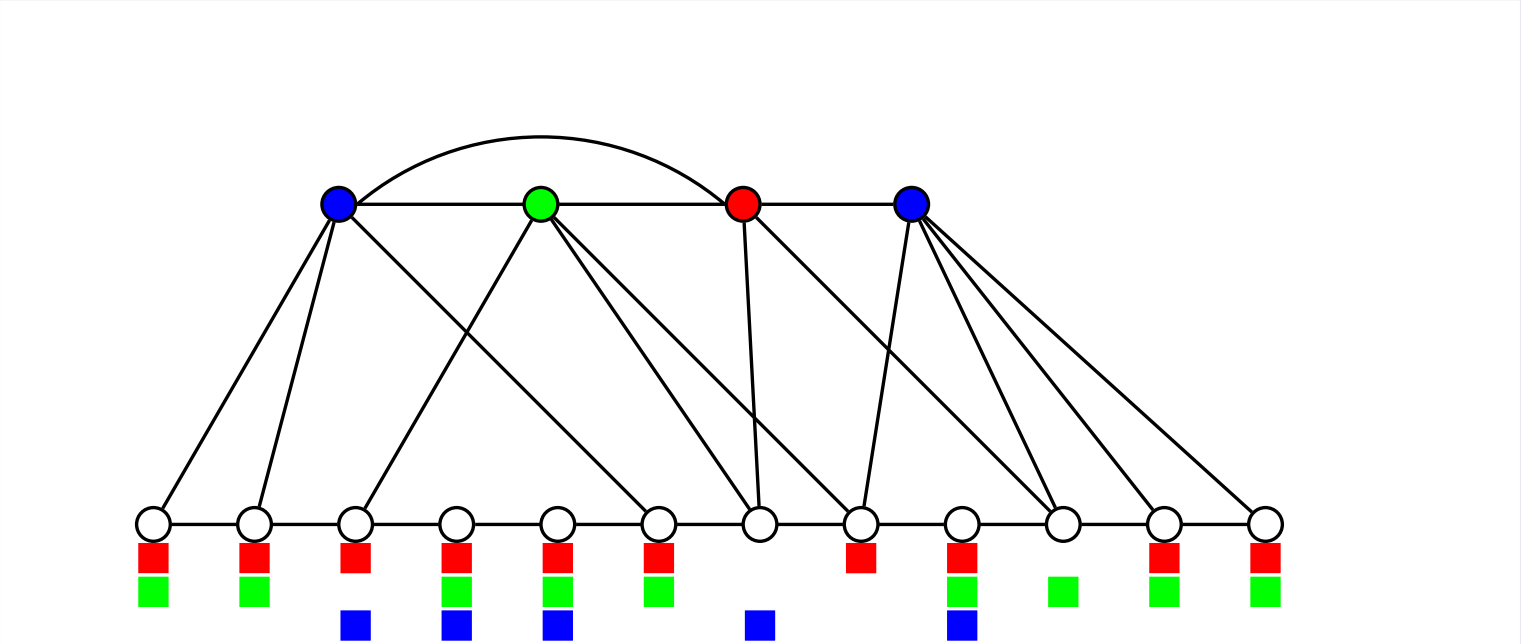
Consider this (**very very special**) class of graphs of treewidth w:

- The graph consists of a long path

3-Coloring algorithm on these graphs:

- Guess a valid coloring of the w non-path vertices
- Try to extend it to a coloring of the whole graph (easy!)

The story so far: Treewidth



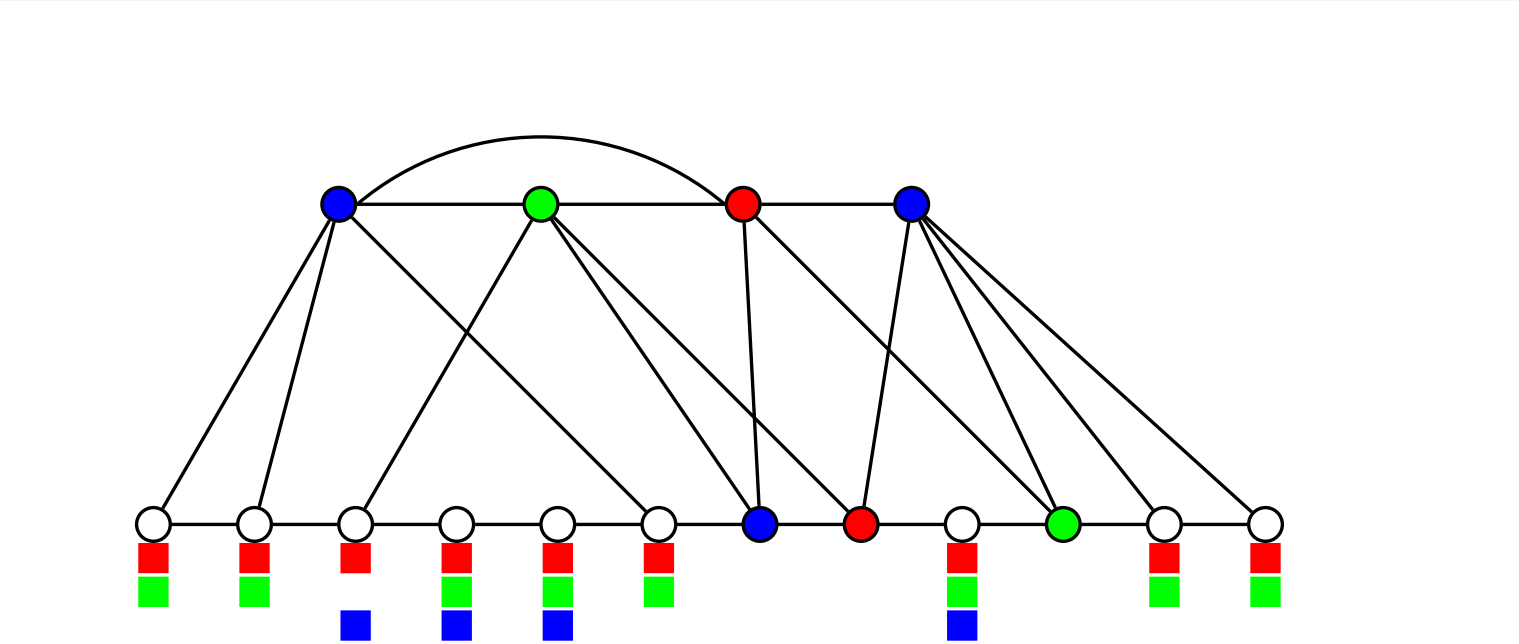
Consider this (**very very special**) class of graphs of treewidth w:

- The graph consists of a long path

3-Coloring algorithm on these graphs:

- Guess a valid coloring of the w non-path vertices
- Try to extend it to a coloring of the whole graph (easy!)

The story so far: Treewidth

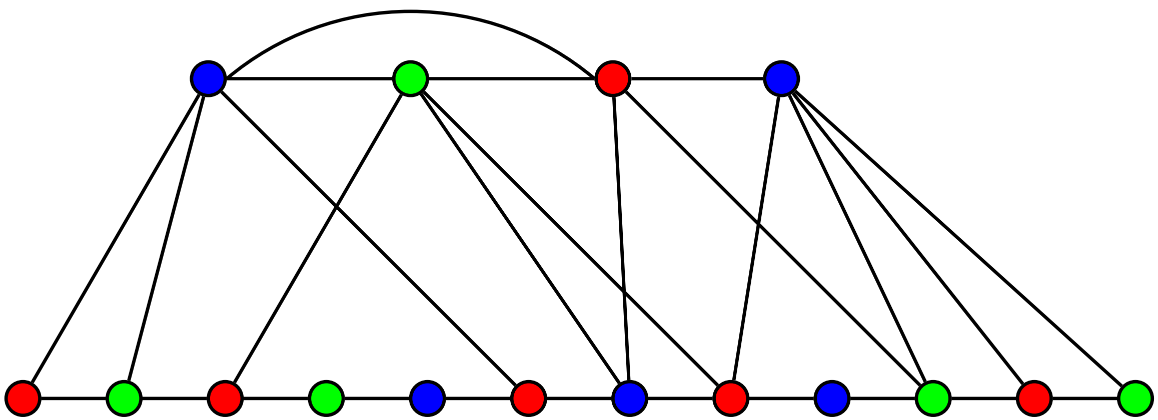


Consider this (**very very special**) class of graphs of treewidth w:

- The graph consists of a long path

3-Coloring algorithm on these graphs:

- Guess a valid coloring of the w non-path vertices
- Try to extend it to a coloring of the whole graph (easy!)



Consider this (**very very special**) class of graphs of treewidth w:

- The graph consists of a long path

3-Coloring algorithm on these graphs:

- Guess a valid coloring of the w non-path vertices
- Try to extend it to a coloring of the whole graph (easy!)
- Either found a valid coloring, or try another coloring for w vertices.

Running time: 3^w

The story so far: Treewidth

- Graphs of treewidth w are **much more general** than the graphs of the previous slide.
  - Algorithm generalizes easily (DP)
  - Running time: k^w.

The story so far: Treewidth

- Graphs of treewidth w are **much more general** than the graphs of the previous slide.
  - Algorithm generalizes easily (DP)
  - Running time: k^w.

Can we do better?

The story so far: Treewidth

- Graphs of treewidth w are **much more general** than the graphs of the previous slide.
  - Algorithm generalizes easily (DP)
  - Running time: k^w.

Can we do better?

The story so far: Treewidth

- Graphs of treewidth w are **much more general** than the graphs of the previous slide.
  - Algorithm generalizes easily (DP)
  - Running time: k^w.

Can we do better?

Previous Work:

- Lokshtanov, Marx, Saurabh, SODA'11
- Jaffke and Jansen, CIAC '17

Result:
(SETH) $\rightarrow$ cannot do $(k-\epsilon)^w$, **for any** k, ϵ, even for Paths+w!

Very **fine**, completely **tight** bound!

Note: SETH $\approx$ SAT has no 1.999^n algorithm.

- Graphs of treewidth w are **much more general** than the graphs of the previous slide.
  - Algorithm generalizes easily (DP)
  - Running time: k^w.

Can we do better?

Previous Work:

- Lokshtanov, Marx, Saurabh, SODA'11
- Jaffke and Jansen, CIAC

Result:
(SETH) $\rightarrow$ cannot do $(k - \epsilon$ en for Paths+w!

Very **fine**, completely **tight**

Note: SETH $\approx$ SAT has no

The story so far: Clique-width

- Clique-width is the second most widely studied graph width.
  - Intuition: Treewidth + Some dense graphs.
  - Definition in next slide.

Summary of what is known for k-Coloring on graphs of clique-width w:

- Algorithm in $k^{2^{O(w)}}$ (Kobler and Rotics DAM ’03)
- Algorithm in $4^{k \cdot w}$ (Kobler and Rotics DAM ’03)
- W-hard parameterized by w (Fomin, Golovach, Lokshtanov, and Saurabh SICOMP ’10)
- ETH LB of $n^{2^{o(w)}}$ (Golovach, Lokshtanov, Saurabh, Zehavi SODA’18)

The story so far: Clique-width

- Clique-width is the second most widely studied graph width.
  - Intuition: Treewidth + Some dense graphs.
  - Definition in next slide.

Summary of what is known for k-Coloring on graphs of clique-width w:

- Algorithm in $k^{2^{O(w)}}$ (Kobler and Rotics DAM '03)
- Algorithm in $4^{k \cdot w}$ (Kobler and Rotics DAM '03)
- W-hard parameterized by w (Fomin, Golovach, Lokshtanov, and Saurabh SICOMP '10)
- ETH LB of $n^{2^{o(w)}}$ (Golovach, Lokshtanov, Saurabh, Zehavi SODA'18)

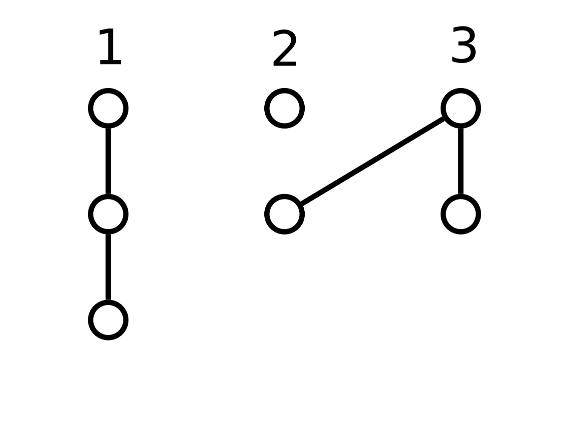
Remark: Last LB is tight (!), but requires k to be large (otherwise contradicts second algorithm)

Story not as clear as treewidth (yet)...

Clique-width: Definition and Intuition

Reminder of the inductive definition of clique-width:

- Each vertex is labelled with a label$\in \{1,\ldots,w\}$.
- Base operation:
  - Construct single-vertex graph.
- Inductive operations:
  - Join (add all edges between two labels)
  - Rename (one label to another)
  - Disjoint Union



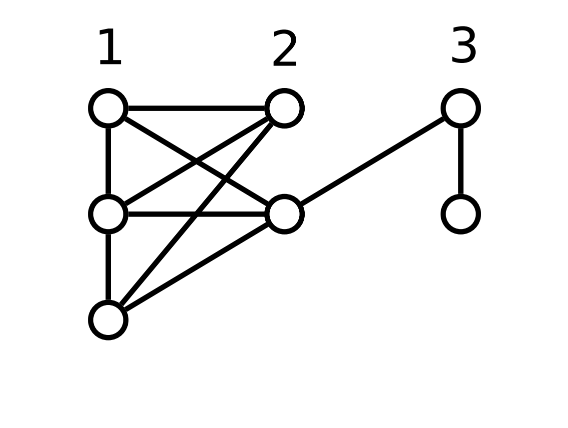
Intuition: Each label set is a **module** with respect to vertices that do not appear in the graph yet.

- Allows us to “forget” some information about what is happening inside a label set, do DP.

Clique-width: Definition and Intuition

Reminder of the inductive definition of clique-width:

- Each vertex is labelled with a label$\in \{1, \ldots, w\}$.
- Base operation:
  - Construct single-vertex graph.
- Inductive operations:
  - Join (add all edges between two labels)
  - Rename (one label to another)
  - Disjoint Union

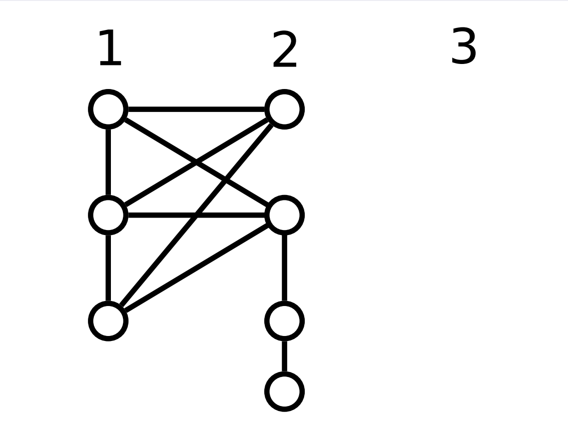


Intuition: Each label set is a **module** with respect to vertices that do not appear in the graph yet.

- Allows us to “forget” some information about what is happening inside a label set, do DP.

Reminder of the inductive definition of clique-width:

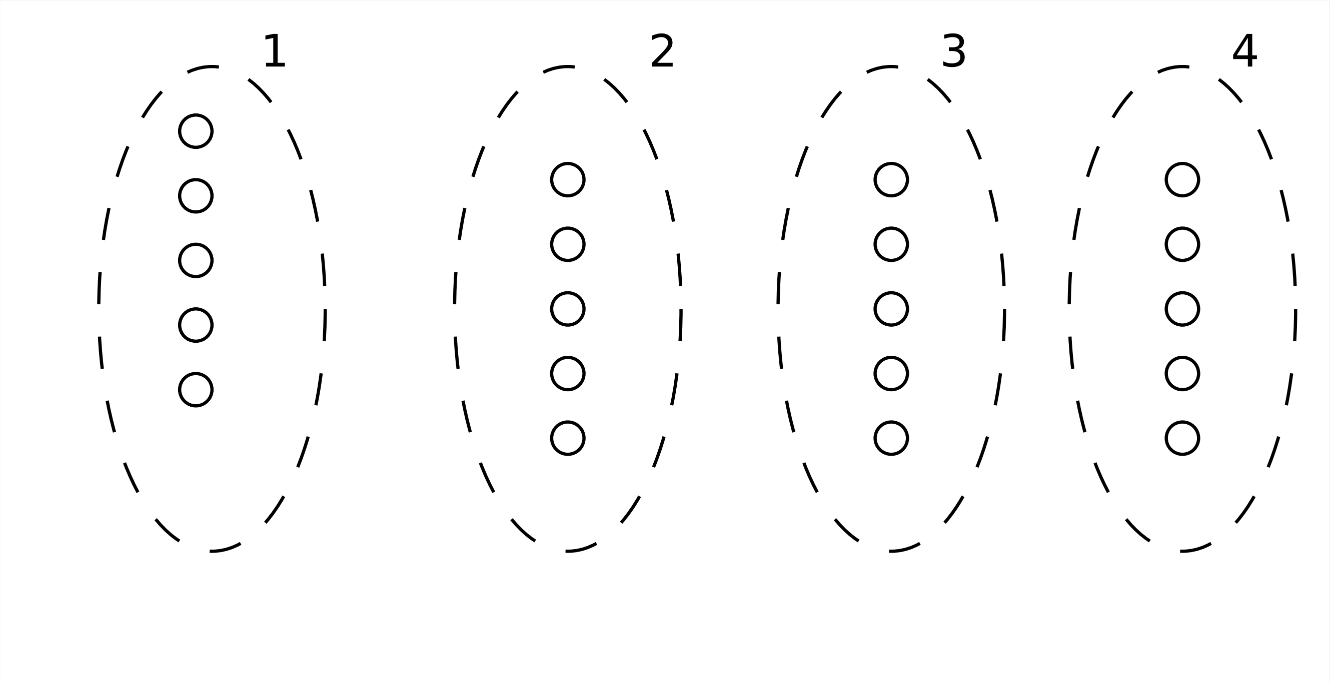
- Each vertex is labelled with a label $\in \{1, \ldots, w\}$.
- Base operation:
  - Construct single-vertex graph.
- Inductive operations:
  - Join (add all edges between two labels)
  - Rename (one label to another)
  - Disjoint Union



Intuition: Each label set is a **module** with respect to vertices that do not appear in the graph yet.

- Allows us to "forget" some information about what is happening inside a label set, do DP.

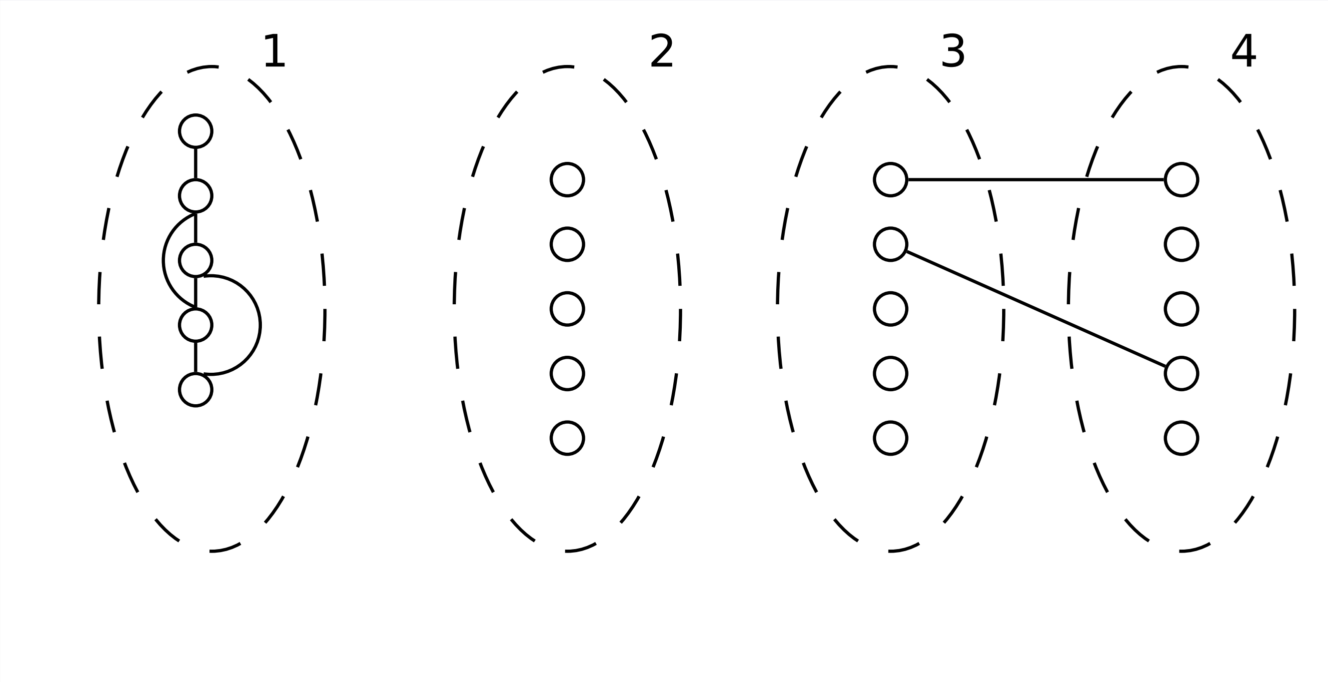
Clique-width: basic algorithm



We recall a basic DP algorithm:

- For every label we remember the **set** of colors used in this label set.

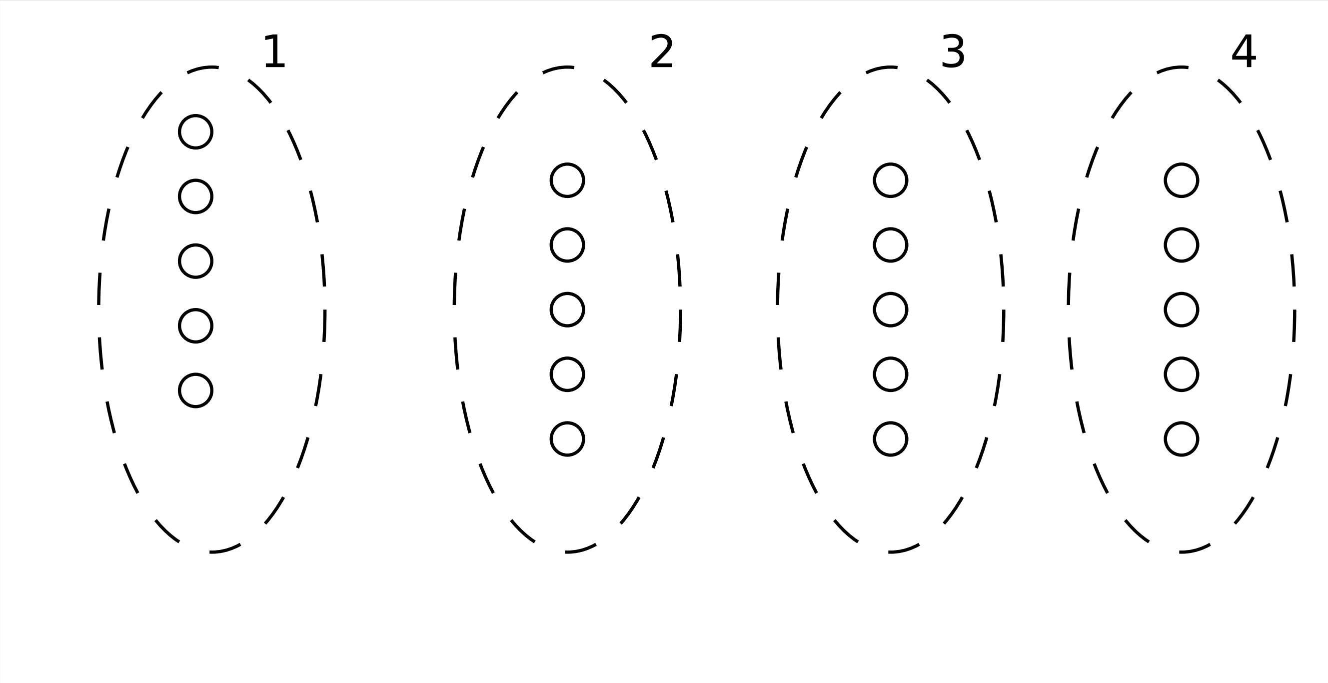
Clique-width: basic algorithm



We recall a basic DP algorithm:

- For every label we remember the **set** of colors used in this label set.

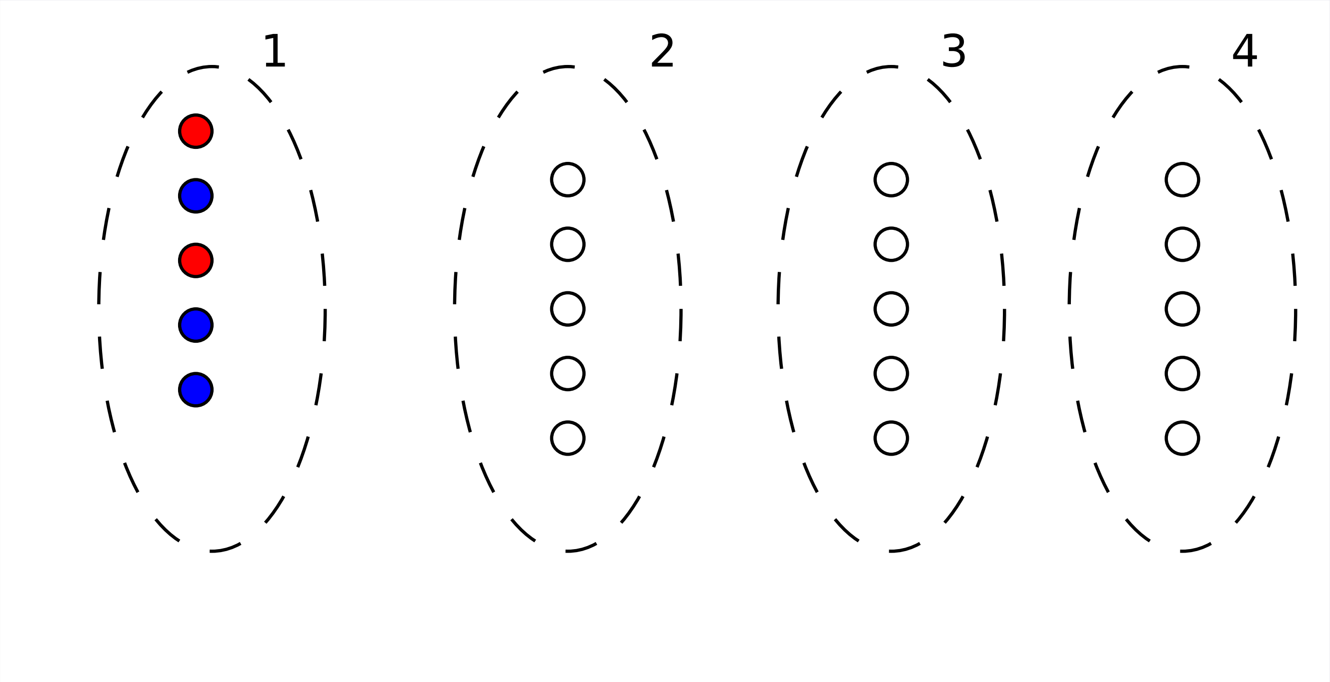
Clique-width: basic algorithm



We recall a basic DP algorithm:

- For every label we remember the **set** of colors used in this label set.

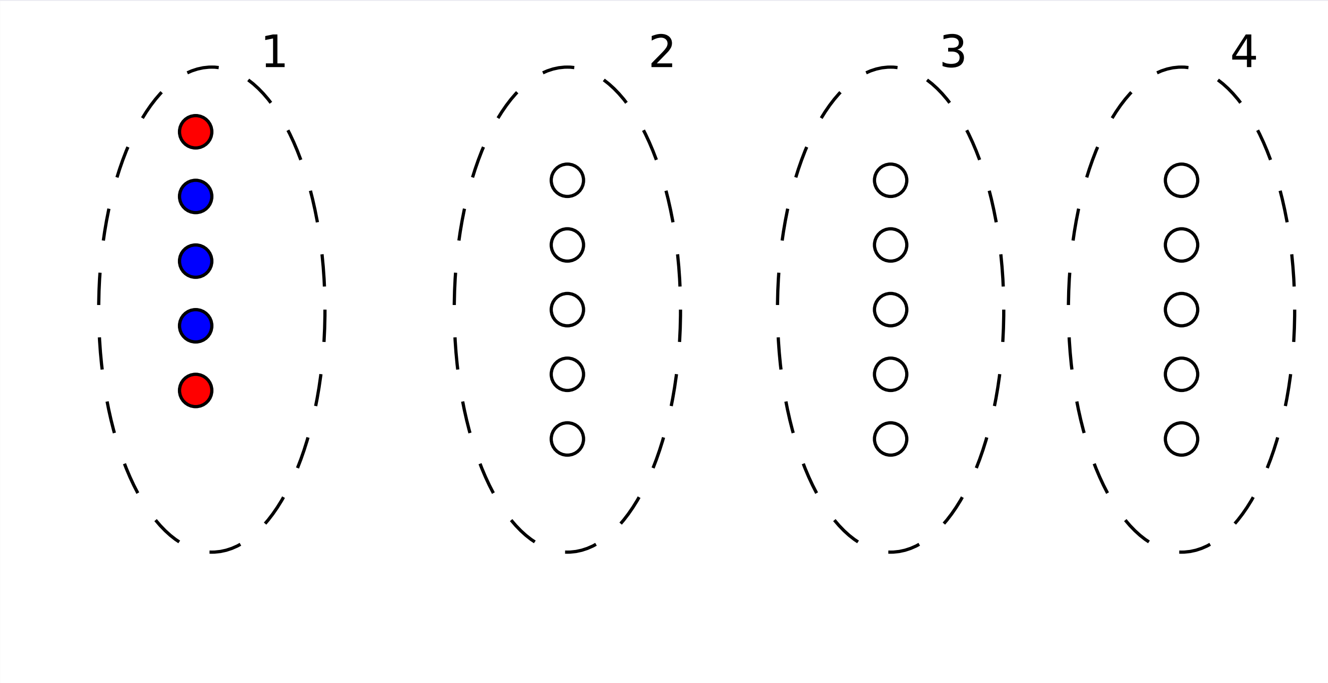
Clique-width: basic algorithm



We recall a basic DP algorithm:

- For every label we remember the **set** of colors used in this label set.

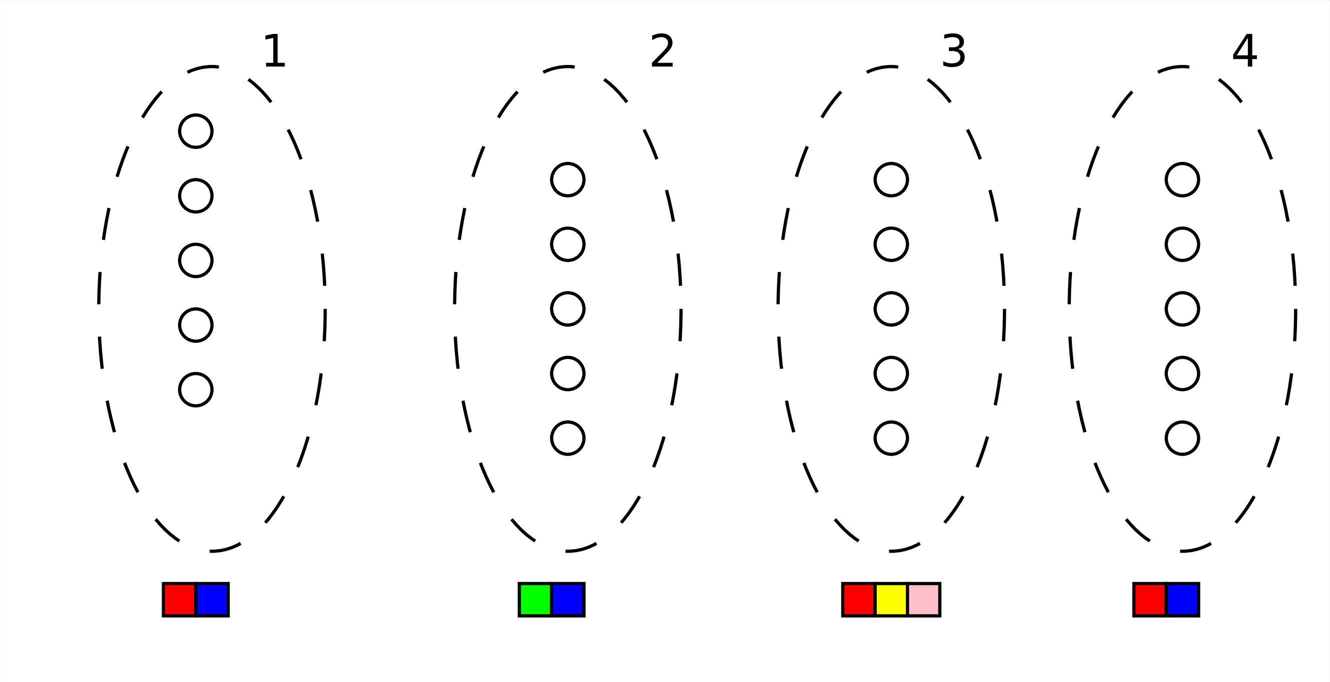
Clique-width: basic algorithm



We recall a basic DP algorithm:

- For every label we remember the **set** of colors used in this label set.
  - Observe: not important which/how many vertices received color red.
  - All future neighbors are common.

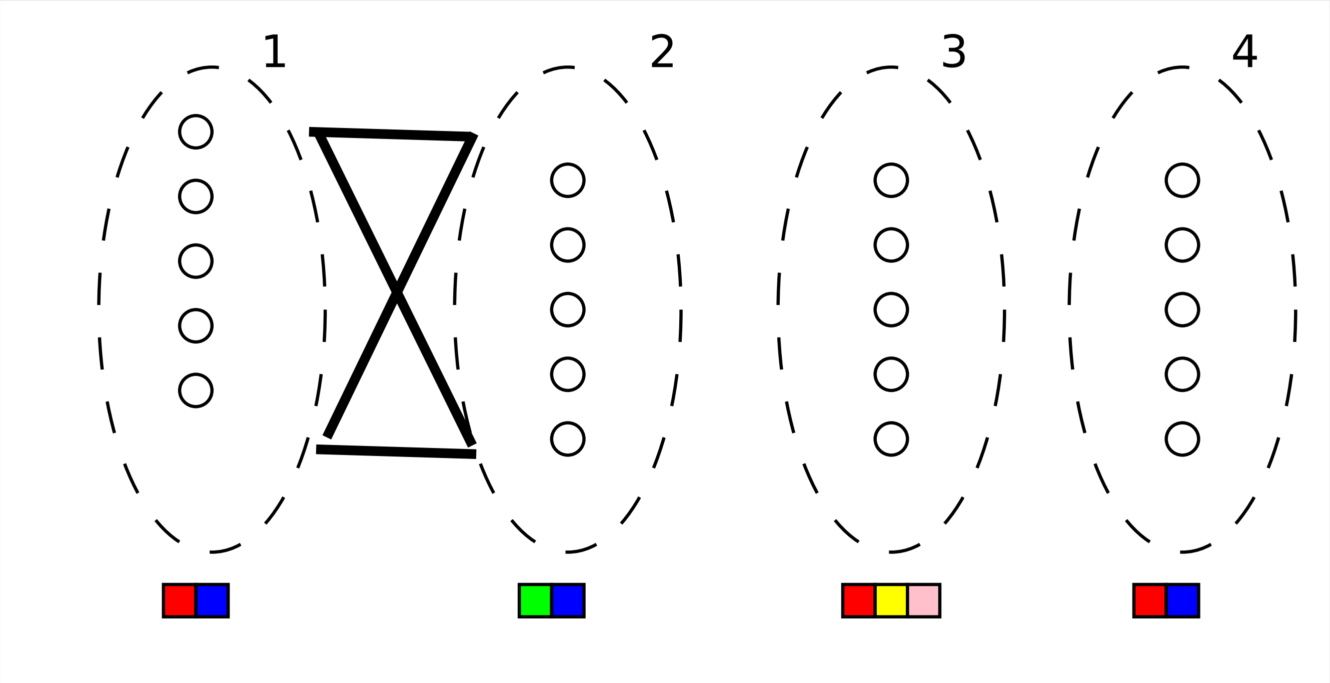
Clique-width: basic algorithm



We recall a basic DP algorithm:

- For every label we remember the **set** of colors used in this label set.

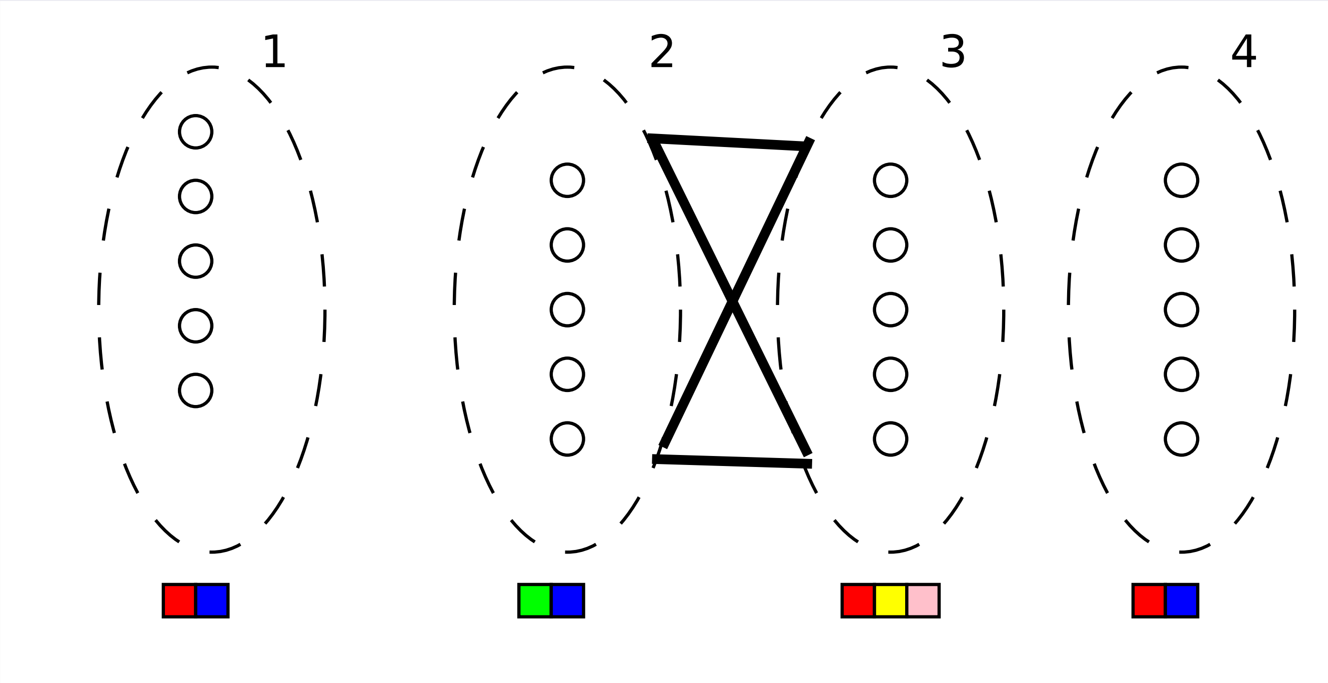
Clique-width: basic algorithm



We recall a basic DP algorithm:

- For every label we remember the **set** of colors used in this label set.
  - For Join operations we check if the sets are disjoint
  - Otherwise discard this partial solution

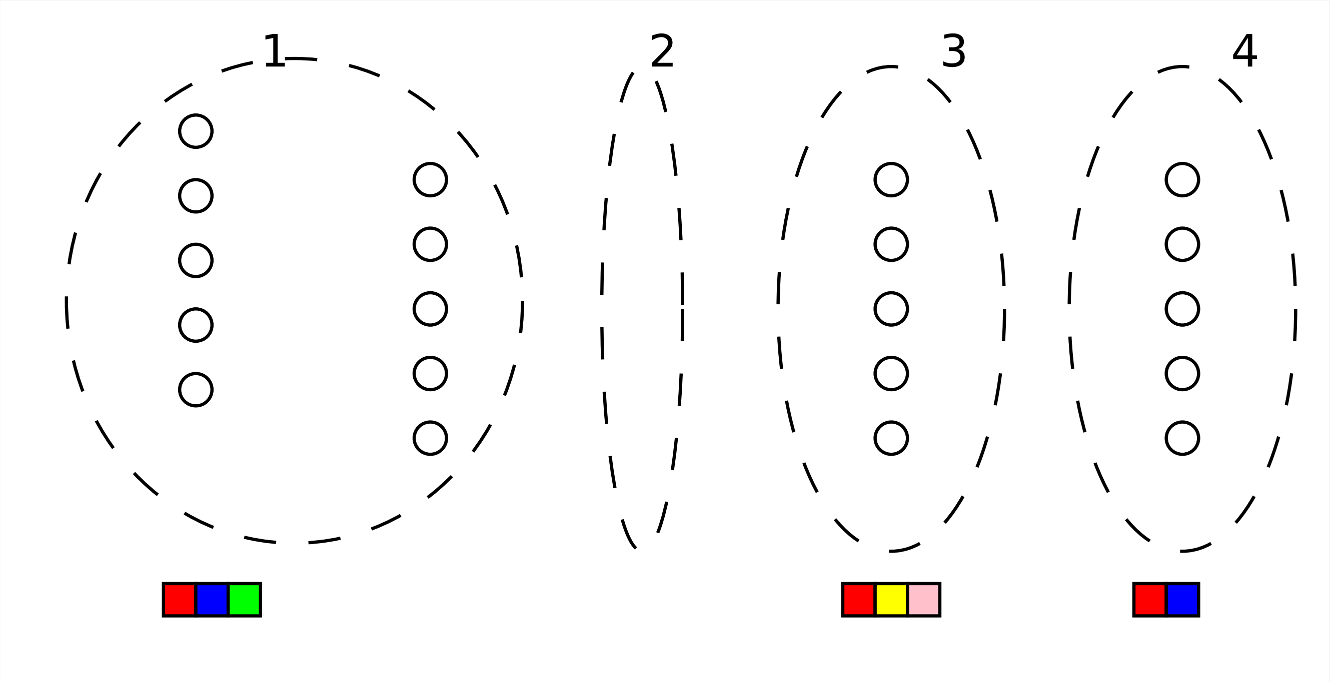
Clique-width: basic algorithm



We recall a basic DP algorithm:

- For every label we remember the **set** of colors used in this label set.
  - For Join operations we check if the sets are disjoint
  - Otherwise discard this partial solution

Clique-width: basic algorithm



We recall a basic DP algorithm:

- For every label we remember the **set** of colors used in this label set.
  - For Rename/Union operations we take unions of sets of colors.

Clique-width: basic algorithm

We recall a basic DP algorithm:

- For every label we remember the **set** of colors used in this label set.

- In the algorithm we sketched the DP has size:
  - 2^k for each label $\rightarrow 2^{k \cdot w}$ in total.
- The $4^{k \cdot w}$ running time claimed comes from a naive implementation of Union operations.
- With modern Fast Subset Convolution technology this can be improved to $2^{k \cdot w}$.

Clique-width: basic algorithm

We recall a basic DP algorithm:

- For every label we remember the **set** of colors used in this label set.
- In the algorithm we sketched the DP has size:
  - 2^k for each label $\rightarrow 2^{k \cdot w}$ in total.
- The $4^{k \cdot w}$ running time claimed comes from a naive implementation of Union operations.
- With modern Fast Subset Convolution technology this can be improved to $2^{k \cdot w}$.

Can we make the DP smaller than $2^{k \cdot w}$?

Clique-width: basic algorithm

We recall a basic DP algorithm:

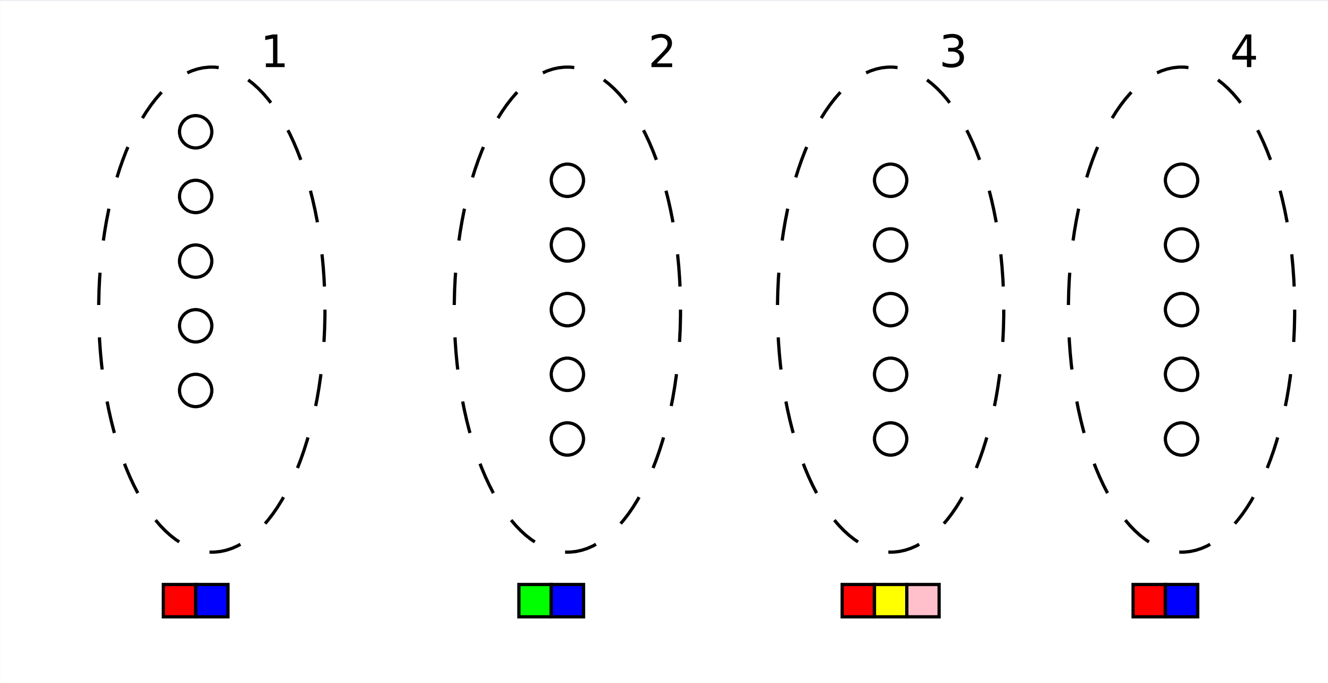
- For every label we remember the **set** of colors used in this label set.

- In the algorithm we sketched the DP has size:
  - 2^k for each label $\rightarrow 2^{k \cdot w}$ in total.
- The $4^{k \cdot w}$ running time claimed comes from a naive implementation of Union operations.
- With modern Fast Subset Convolution technology this can be improved to $2^{k \cdot w}$.

Can we make the DP smaller than $2^{k \cdot w}$?

(**Note:** The k^{2^w} algorithm is much more involved. . .)

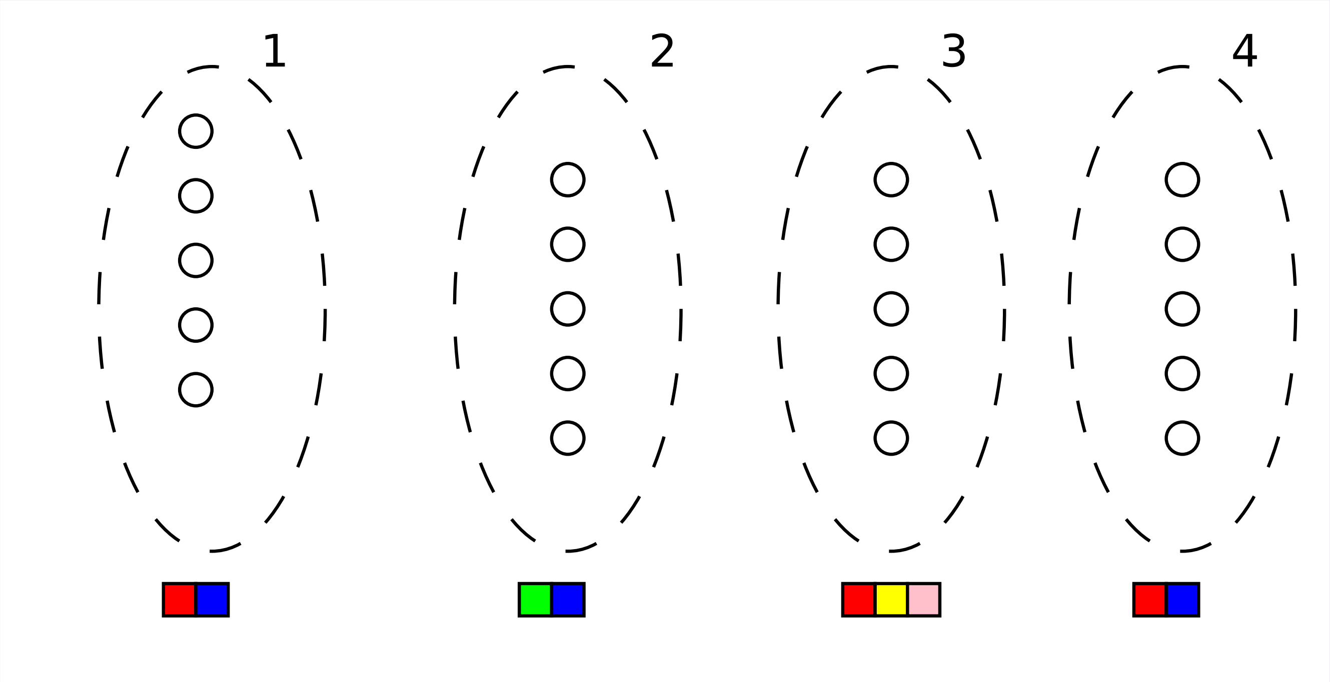
DP algorithm: a closer look



Basic Argument:

- For each label we store a set of colors.
- There are k colors $\rightarrow$ there are 2^k possible sets.

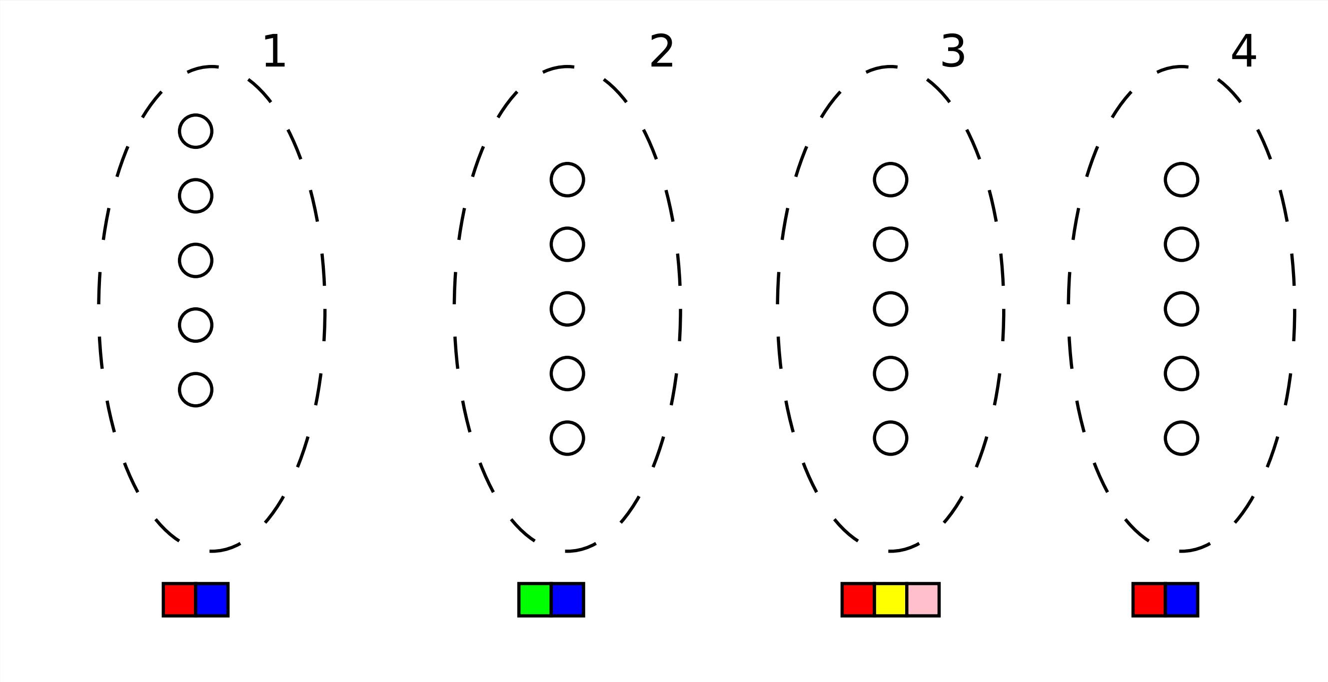
DP algorithm: a closer look



Basic Argument:

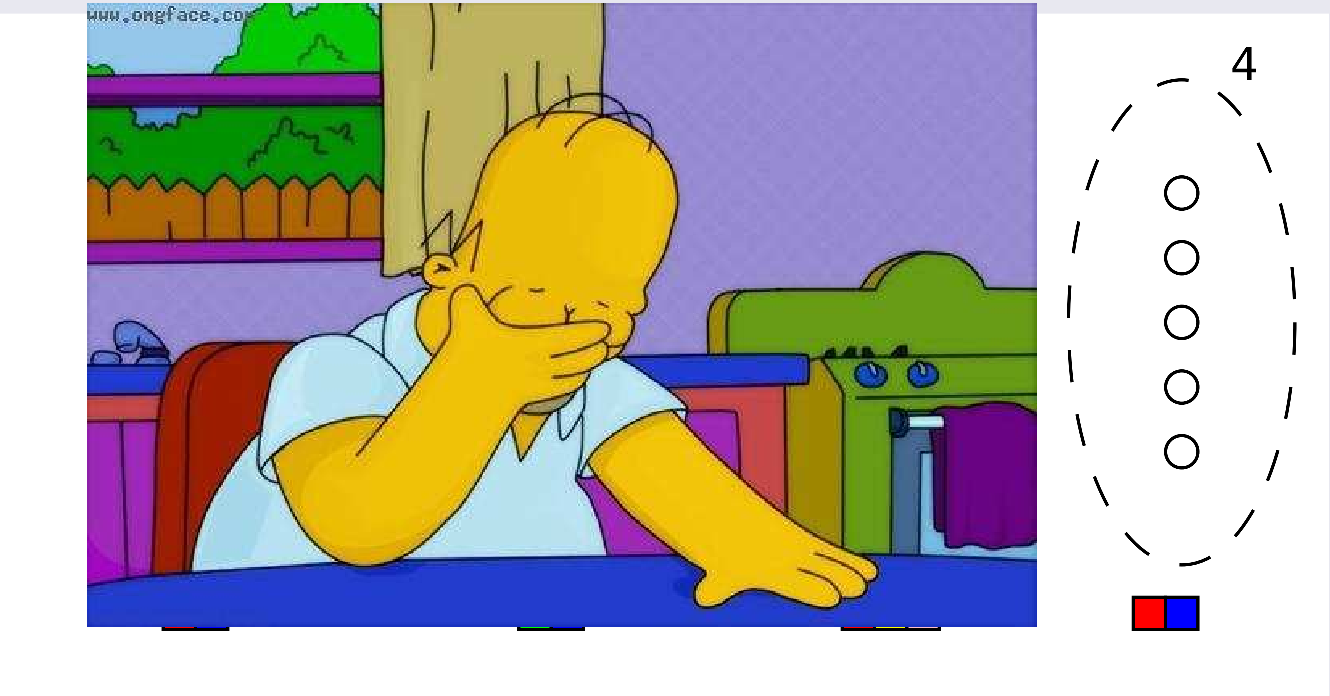
- For each label we store a set of colors.
- There are k colors $\rightarrow$ there are 2^k possible sets.
- **BUT!** How could a label set be colored with $\emptyset$?
    - Ignoring the empty set we improve the DP table to $(2^k - 1)^w$

DP algorithm: an even closer look



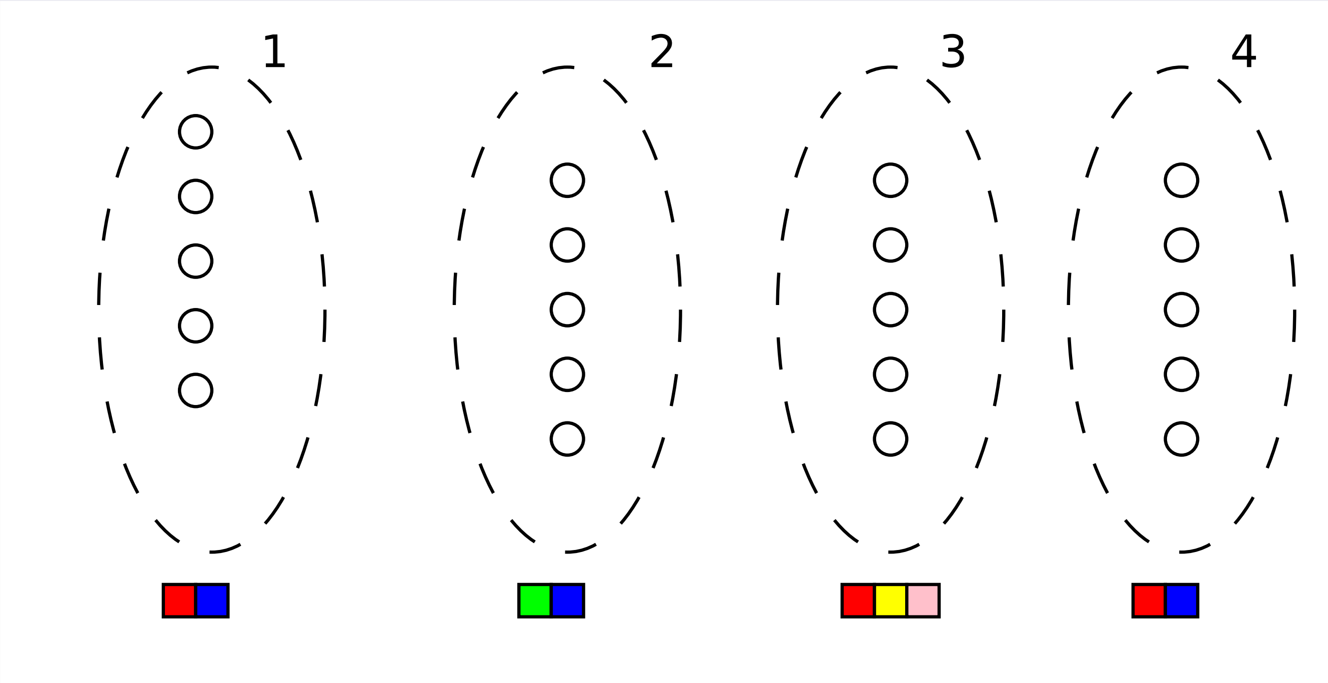
- Could a label set be using **ALL** k colors?

DP algorithm: an even closer look



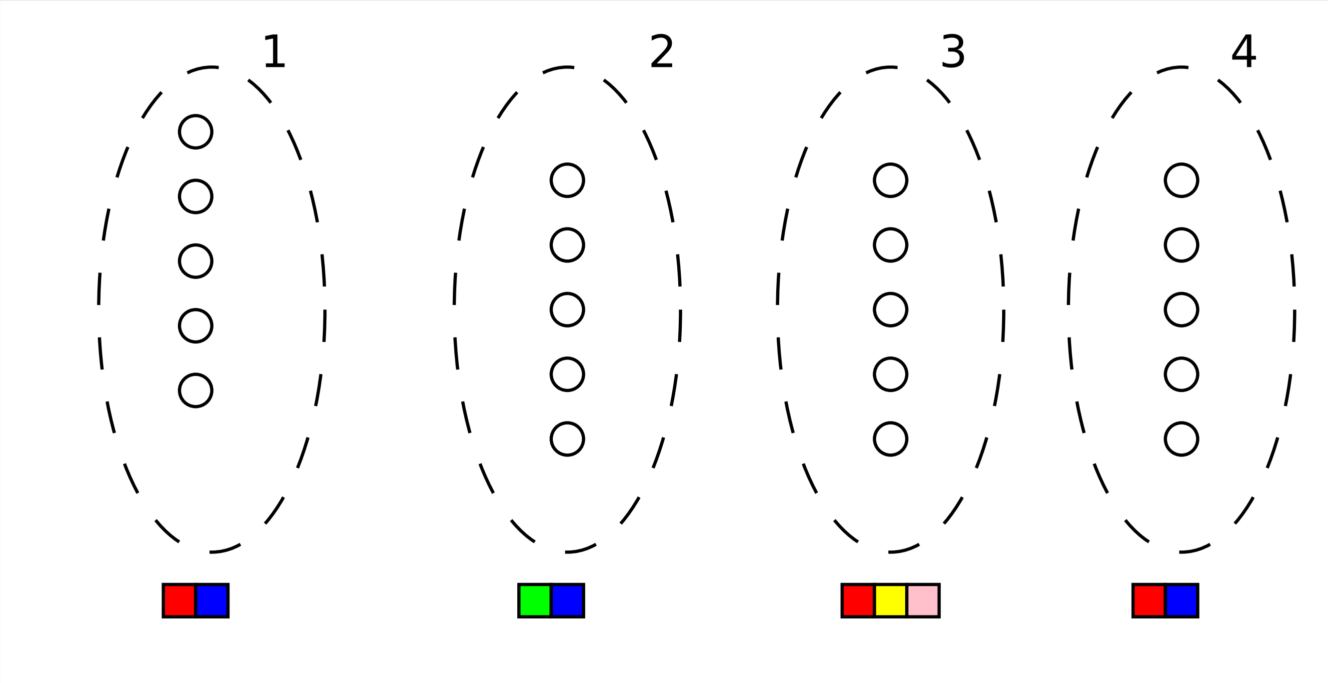
- Could a label set be using **ALL** k colors?
  Yes!

DP algorithm: an even closer look



- Could a label set be using **ALL** k colors?
- Yes, but, then we cannot apply join operations to this label.
  - Separate labels into **live** and **junk**.
  - For live labels $2^k - 2$ feasible sets.
  - For junk labels, who cares?? (no more edges!)

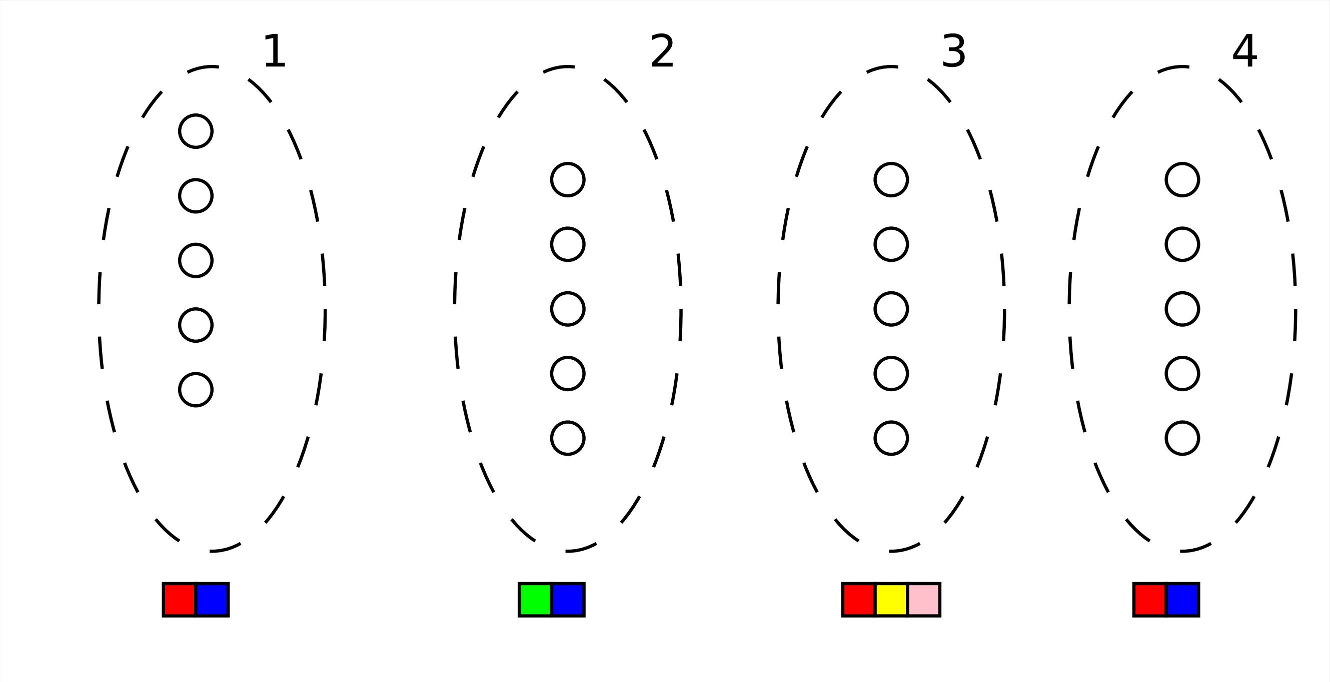
DP algorithm: an even closer look



- Could a label set be using **ALL** k colors?

Bottom line: DP size can be brought down to $(2^k - 2)^w$.

DP algorithm: an even closer look



- Could a label set be using **ALL** k colors?

Bottom line: DP size can be brought down to $(2^k - 2)^w$.

Main result: Under SETH, $(2^k - 2)^w$ is the correct complexity!

The Reduction

Outline

Result: Under SETH, $\forall k, \epsilon$ there is no $(2^k - 2 - \epsilon)^w$ Coloring algorithm.

- Starting Point: q-CSP-B not solvable in $(B - \epsilon)^n$
  - A convenient starting point!
- The main reduction
  - List Coloring
  - Weak Edges – Implications
  - The general structure

SETH more carefully

Goal: A reduction that works as follows

SAT LB	Coloring on clique-width LB
$\nexists (2-\epsilon)^n$	$\rightarrow \nexists\ (2-\epsilon)^w$
n variables	$w =$

SETH more carefully

Goal: A reduction that works as follows

SAT LB	Coloring on clique-width LB
$\nexists (2-\epsilon)^n$	$\rightarrow \nexists\ (2-\epsilon)^w$
n variables	$w = n$

SETH more carefully

Goal: A reduction that works as follows

SAT LB	Coloring on clique-width LB
$\nexists (2-\epsilon)^n$	$\rightarrow \nexists\ (4-\epsilon)^w$
n variables	$w = n/2$

SETH more carefully

Goal: A reduction that works as follows

SAT LB	Coloring on clique-width LB
$\not\exists (2-\epsilon)^n$	$\rightarrow \not\exists\ (8-\epsilon)^w$
n variables	$w = n/3$

SETH more carefully

Goal: A reduction that works as follows

SAT LB	Coloring on clique-width LB
$\nexists (2-\epsilon)^n$	$\rightarrow \nexists\ (6-\epsilon)^w$
n variables	$w = ??$

SETH more carefully

Goal: A reduction that works as follows

SAT LB	Coloring on clique-width LB
$\nexists (2-\epsilon)^n$	$\rightarrow \nexists\ (6-\epsilon)^w$
n variables	$w = n/\log 6$ Not an int!

- Reductions aiming for a LB of the form c^w, where c is a power of 2 are easy
  - Map $\log c$ SAT variables to each unit of width.
- If c is not a power of 2 things become messier:

Goal: A reduction that works as follows

SAT LB	Coloring on clique-width LB
$\nexists (2-\epsilon)^n$	$\rightarrow \nexists$
n variables	$w = n/\log 6$ Not an int!

- Reductions aiming for a LB of the form c^w, where c is a power of 2 are easy
  - Map $\log c$ SAT variables to each unit of width.
- If c is not a power of 2 things become messier:

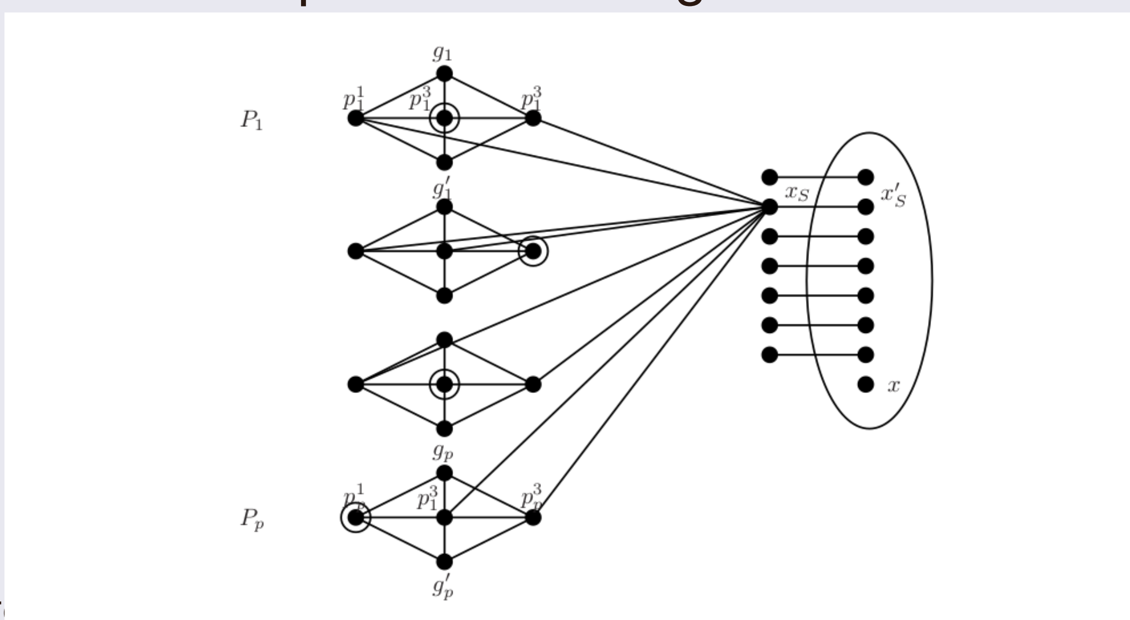


Figure 2: Reduction to DOMINATING SET: group gadget $\widehat{B}$. The set S is shown by the circled vertices.

SETH more carefully

Goal: A reduction that works as follows

SAT LB	Coloring on clique-width LB
$\nexists(2-\epsilon)^n$	$\rightarrow \nexists$
n variables	$w = n/\log 6$ Not an int!

- Reductions aiming for a LB of the form c^w, where c is a power of 2 are easy
  - Map $\log c$ SAT variables to each unit of width.
- If c is not a power of 2 things become messier:

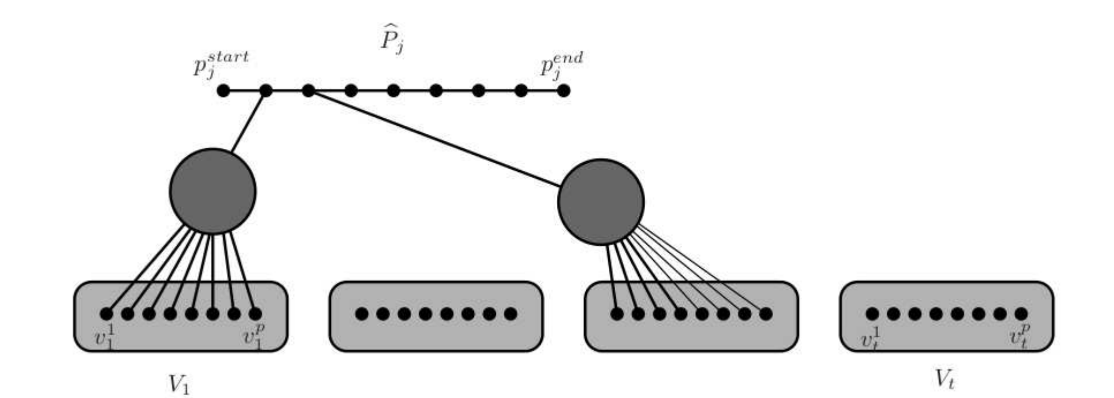


Figure 5: Reduction to q-COLORING. The t groups of vertices $V_1, \ldots, V_t$ represent the t groups of variables $F_1, \ldots, F_t$ (each of size $\lceil \log q^p \rceil$). Each vertex of the clause path $\widehat{P}_j$ is connected to one group V_i via a connector.

SETH more carefully

Goal: A reduction that works as follows

SAT LB	Coloring on clique-width LB
$\nexists (2-\epsilon)^n$	$\rightarrow \nexists$
n variables	$w = n/\log 6$ Not an int!

- Reductions aiming for a LB of the form c^w, where c is a power of 2 are easy
  - Map $\log c$ SAT variables to each unit of width.
- If c is not a power of 2 things become messier:
- Solution: Map $p \log c$ variables to p units of width, for p sufficiently large.
  - Usually done as sub-part of the reduction.
  - May complicate the problem unnecessarily. . .

SETH more carefully

- SETH informal: SAT cannot be solved in $(2-\epsilon)^n$.
- SETH more careful: for all $\epsilon > 0$ there exists q such that q-SAT cannot be solved in $(2-\epsilon)^n$.

SETH more carefully

- SETH informal: SAT cannot be solved in $(2-\epsilon)^n$.
- SETH more careful: for all $\epsilon > 0$ there exists q such that q-SAT cannot be solved in $(2-\epsilon)^n$.
- If we accept the more careful form of SETH we can obtain a convenient starting point for any lower bound

> If SETH is true, then for all $B \geq 2, \epsilon > 0$ there exists q such that q-CSP-B cannot be solved in $(B-\epsilon)^n$

SETH more carefully

- SETH informal: SAT cannot be solved in $(2-\epsilon)^n$.
- SETH more careful: for all $\epsilon > 0$ there exists q such that q-SAT cannot be solved in $(2-\epsilon)^n$.
- If we accept the more careful form of SETH we can obtain a convenient starting point for any lower bound

> If SETH is true, then for all $B \geq 2, \epsilon > 0$ there exists q such that q-CSP-B cannot be solved in $(B-\epsilon)^n$

- Translation: we get a problem that needs time 6^n, or 14^n, or 30^n, or ...
- **Ready to be used for all your reduction needs!**

Main Reduction – Step 1

Strategy: Reduce q-CSP-6 to 3-Coloring on clique-width.
• If $w = n + O(1)$, then we get $(6-\epsilon)^w = (2^k - 2 - \epsilon)^w$ lower bound, DONE!

- Step 1: Define an arbitrary mapping from the alphabet of the CSP $1, \ldots, 6$ to sets of colors.

1	R
2	G
3	B
4	RG
5	RB
6	GB

- **Intuition:** We define a label class for each variable. This label class uses exactly the colors given by the mapping of its satisfying value.

Main Reduction – Step 2

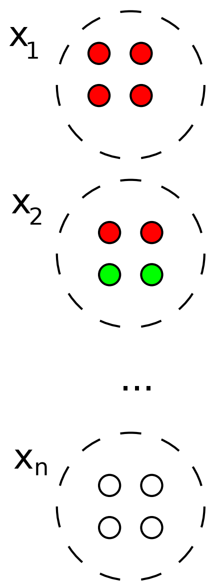
We assume the existence of the following gadgets:

- List Coloring: We can assign each vertex a list of feasible colors
- Implications: If source has a certain color, this forces a color on the sink

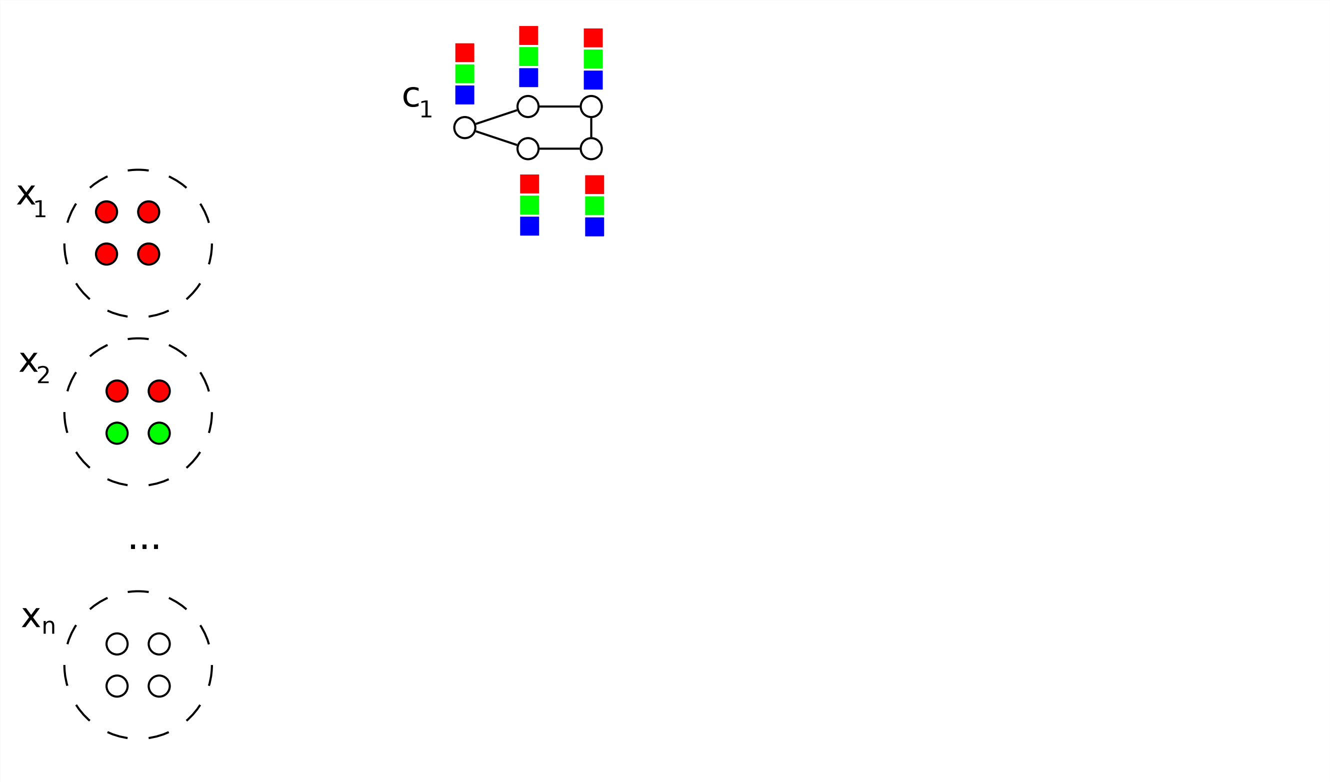
Main Reduction – Step 2

- We maintain n label sets (one for each variable).

Main Reduction – Step 2

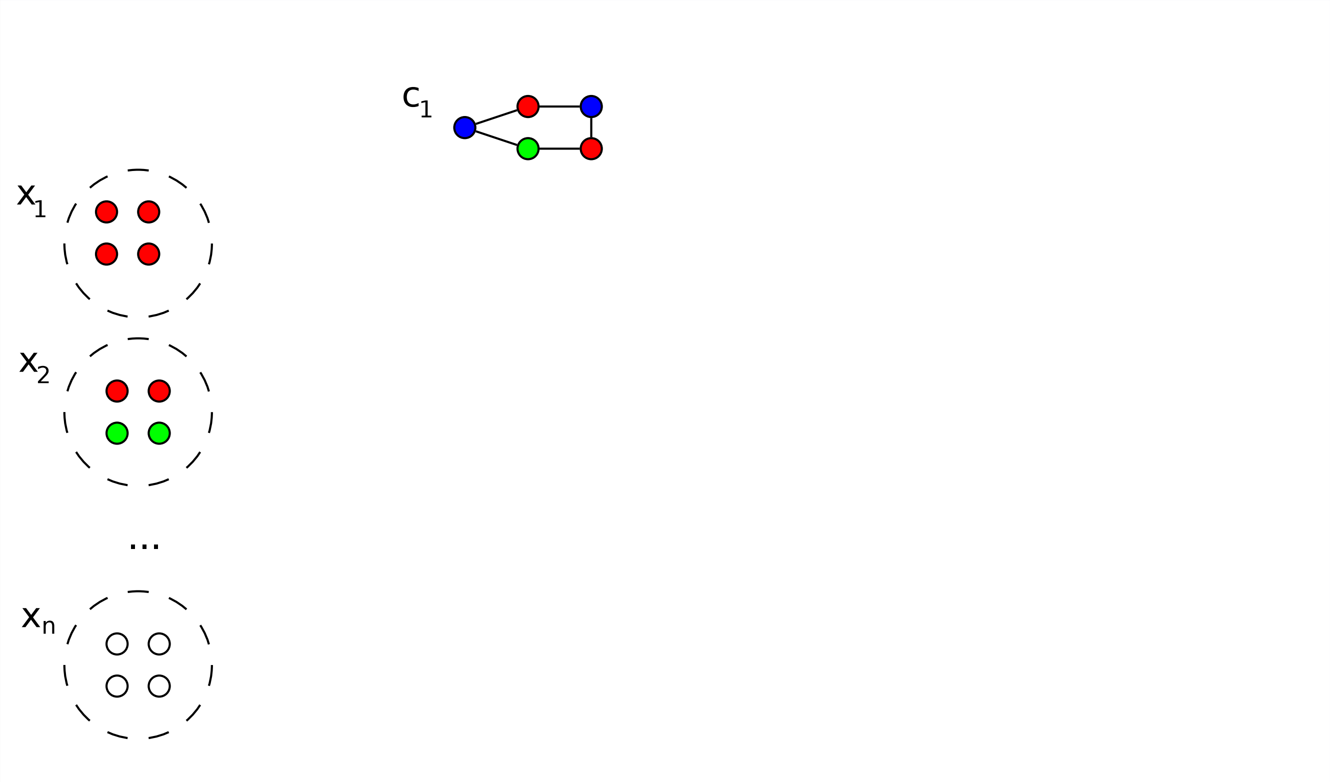


- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- Here: $x_1 = 1, x_2 = 4$



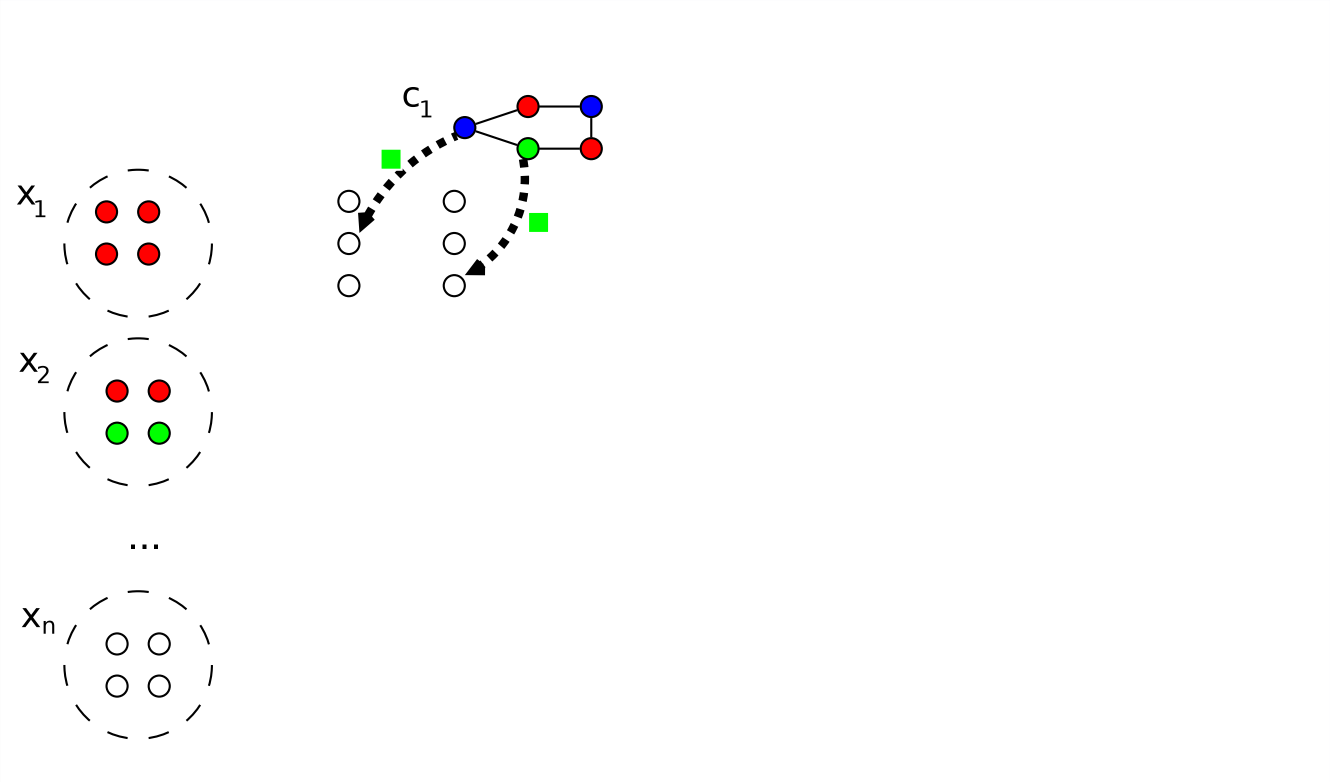
- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- For each constraint: odd cycle with 3 color list
- $\rightarrow$ Each vertex represents a satisfying assignment
- $\rightarrow$ Green vertex $\leftrightarrow$ selected assignment

Main Reduction – Step 2



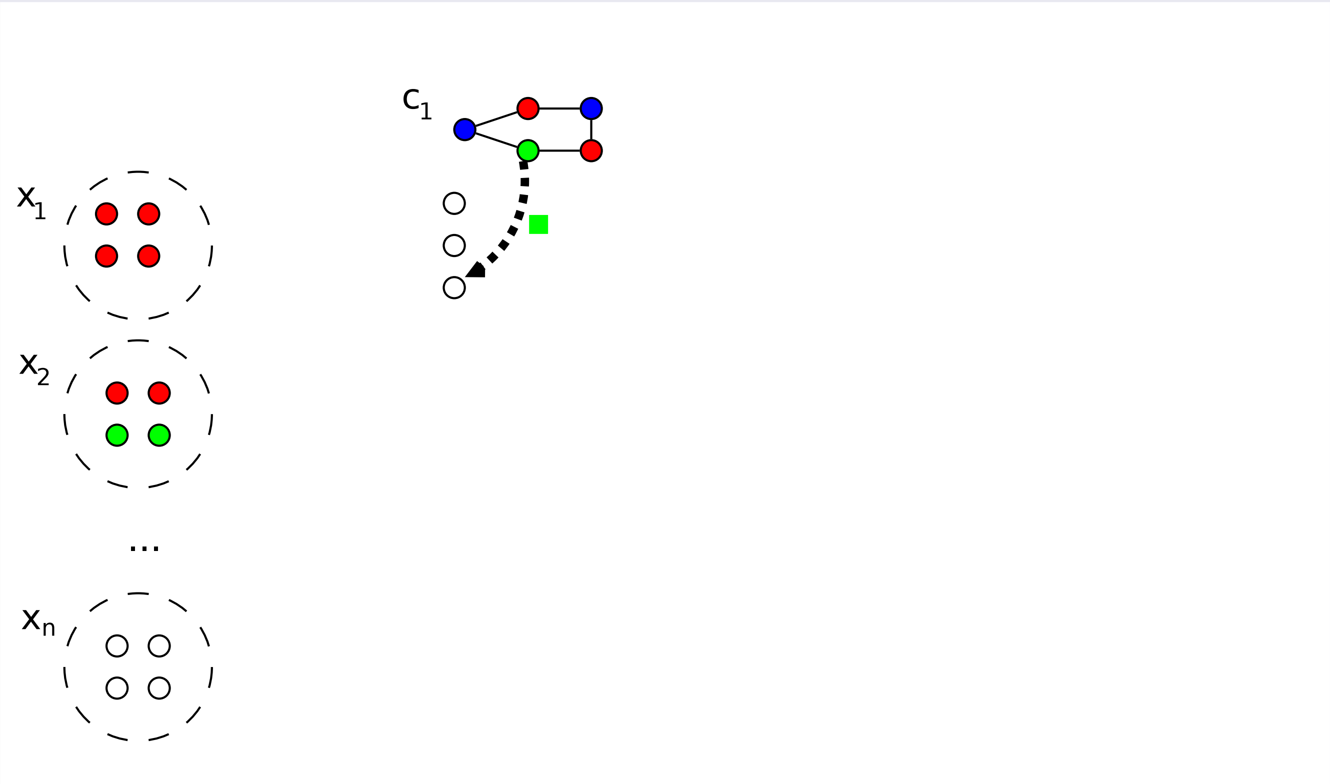
- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- For each constraint: odd cycle with 3 color list
- $\rightarrow$ Each vertex represents a satisfying assignment
- $\rightarrow$ Green vertex $\leftrightarrow$ selected assignment

Main Reduction – Step 2



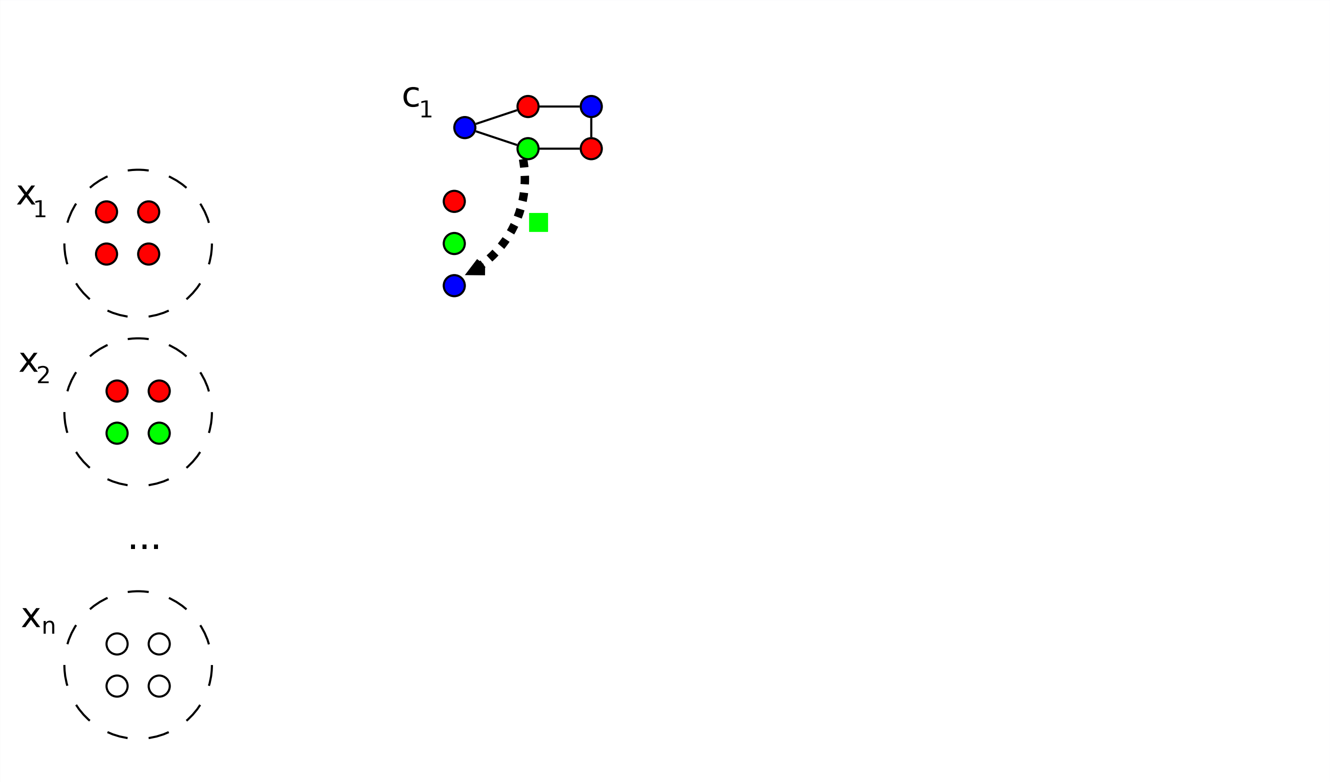
- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- $\rightarrow$ Green vertex $\leftrightarrow$ selected assignment
- Add Green-activated implications

Main Reduction – Step 2



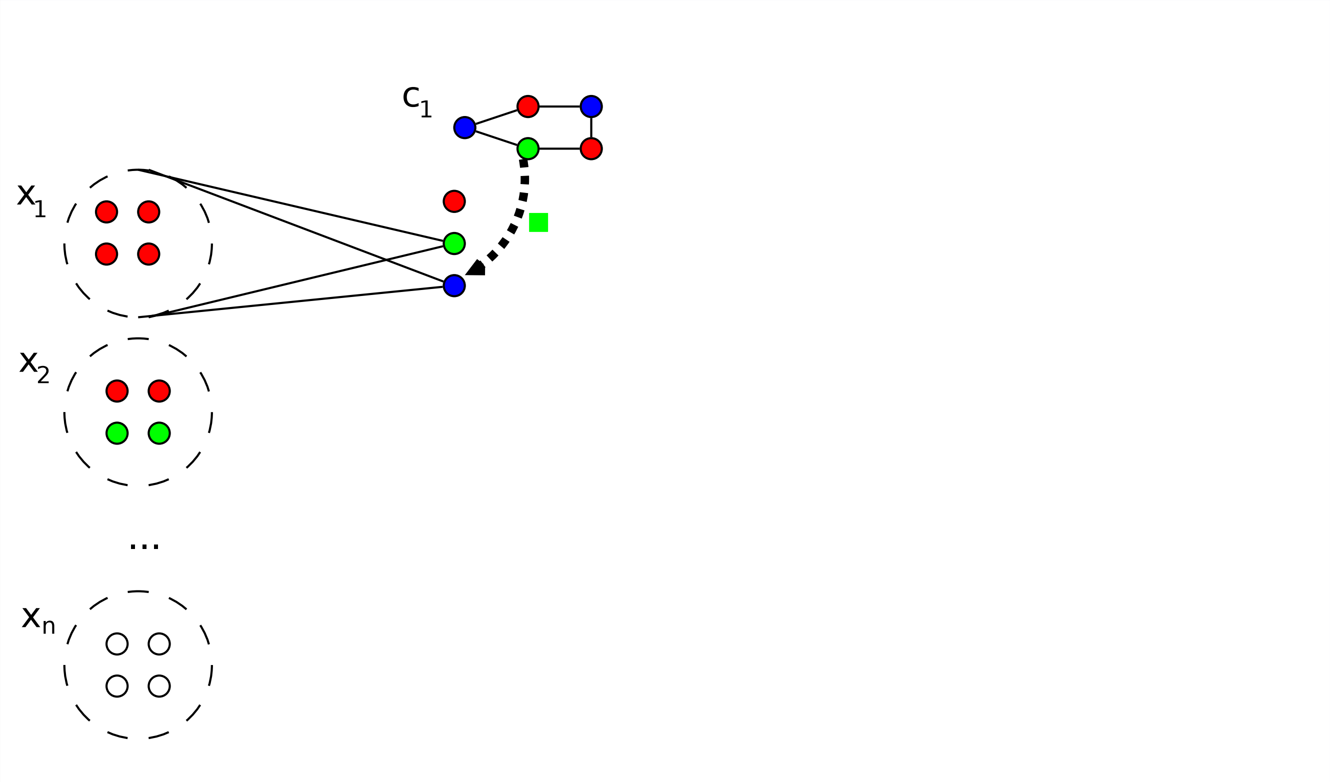
- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- $\rightarrow$ Green vertex $\leftrightarrow$ selected assignment
- Add Green-activated implications
- Non-selected assignment $\rightarrow$ implications irrelevant

Main Reduction – Step 2



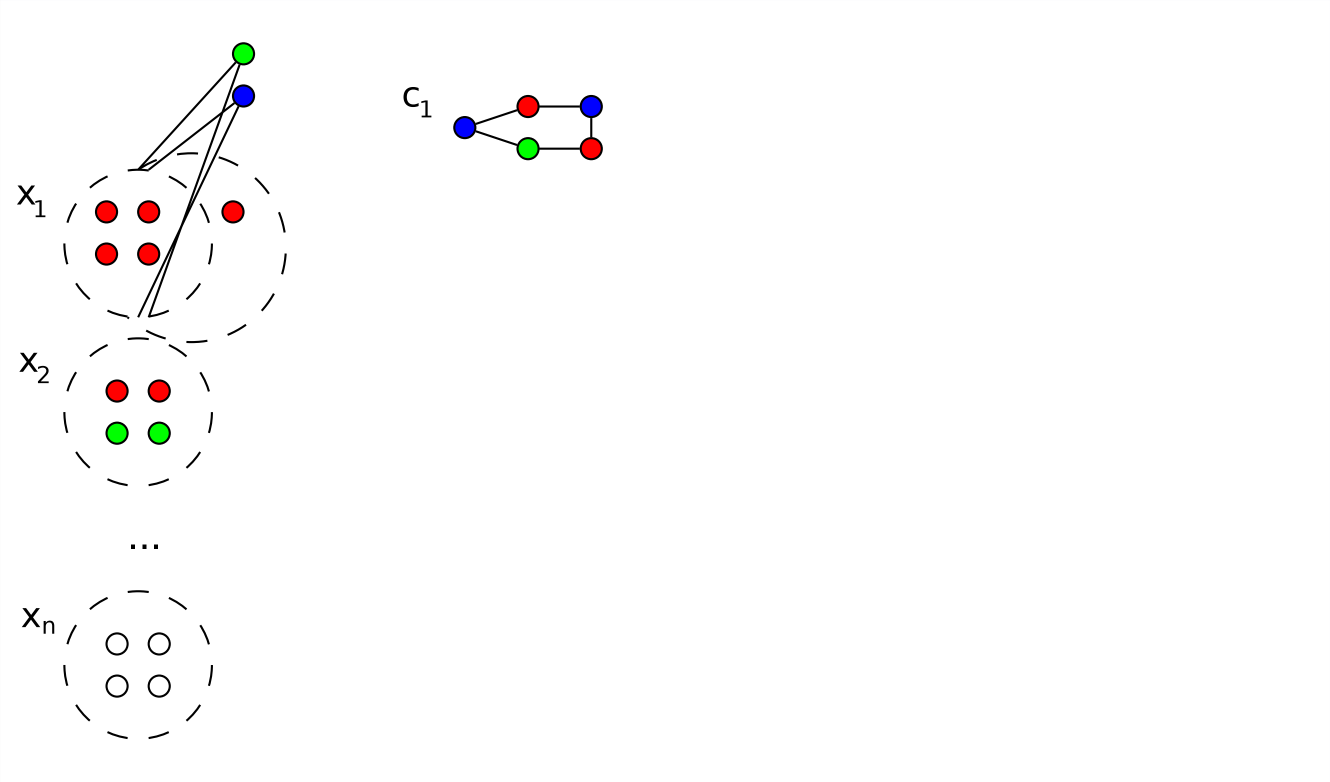
- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- $\rightarrow$ Green vertex $\leftrightarrow$ selected assignment
- Add Green-activated implications
- Selected assignment $\rightarrow$ Colors forced

Main Reduction – Step 2



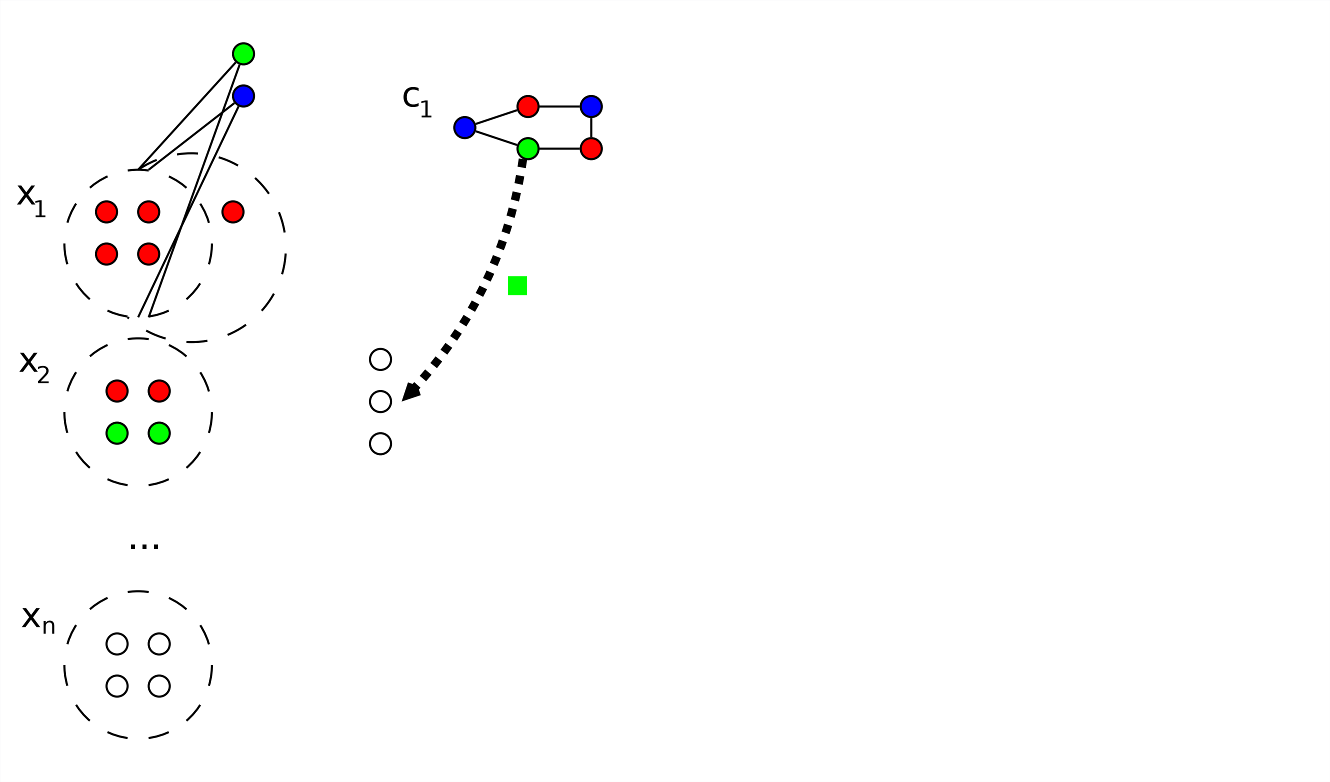
- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- $\rightarrow$ Green vertex $\leftrightarrow$ selected assignment
- Add edges from vertices not supposed to have a color in x_1 to x_1.

Main Reduction – Step 2



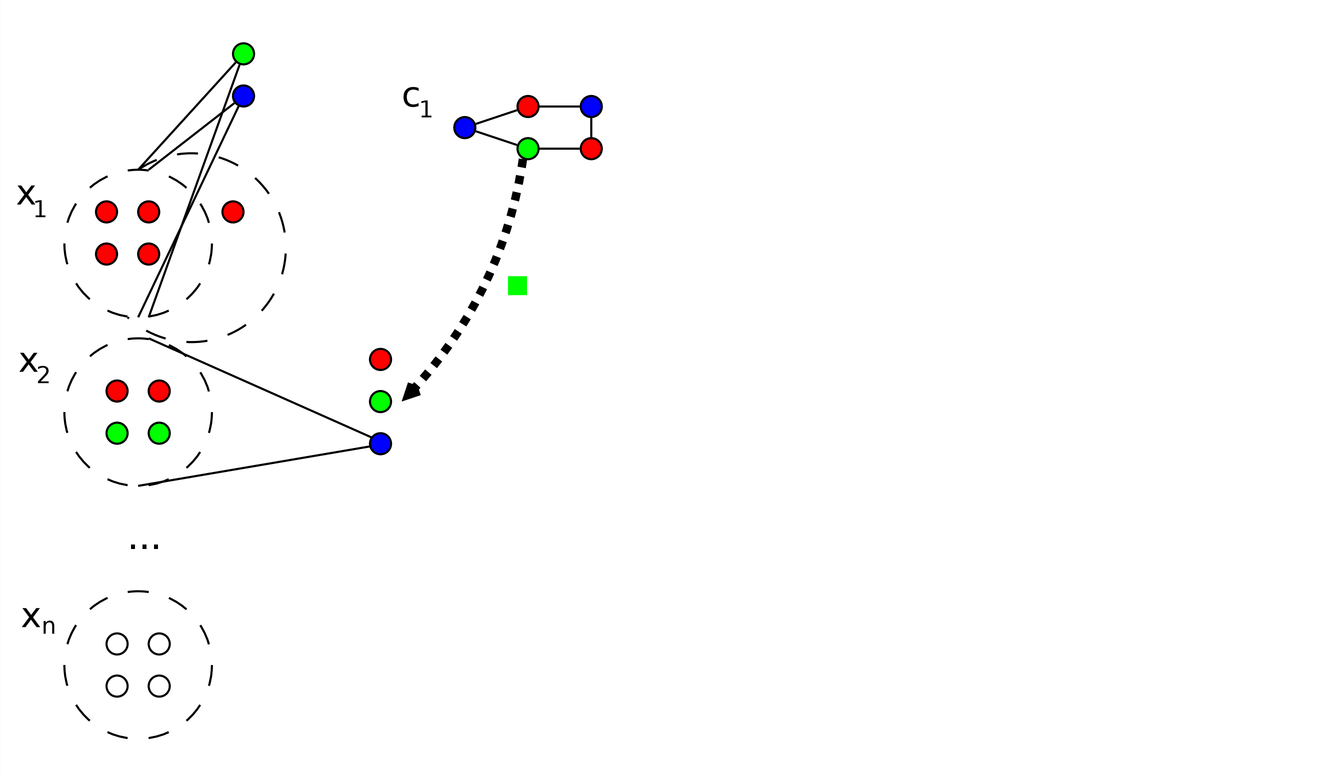
- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- $\rightarrow$ Green vertex $\leftrightarrow$ selected assignment
- Add edges from vertices not supposed to have a color in x_1 to x_1.
- Move these vertices to JUNK, others to x_1

Main Reduction – Step 2



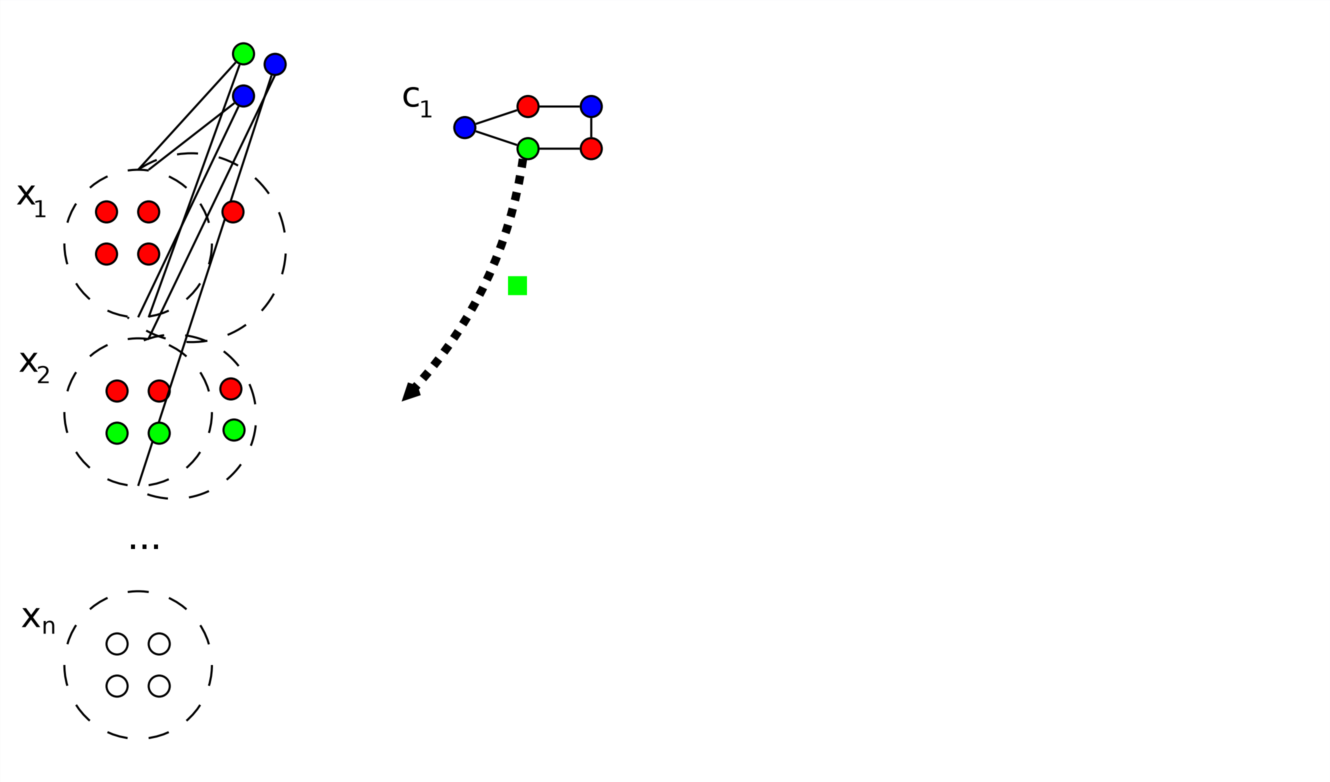
- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- $\rightarrow$ Green vertex $\leftrightarrow$ selected assignment
- Do the same for other variables of c_1

Main Reduction – Step 2



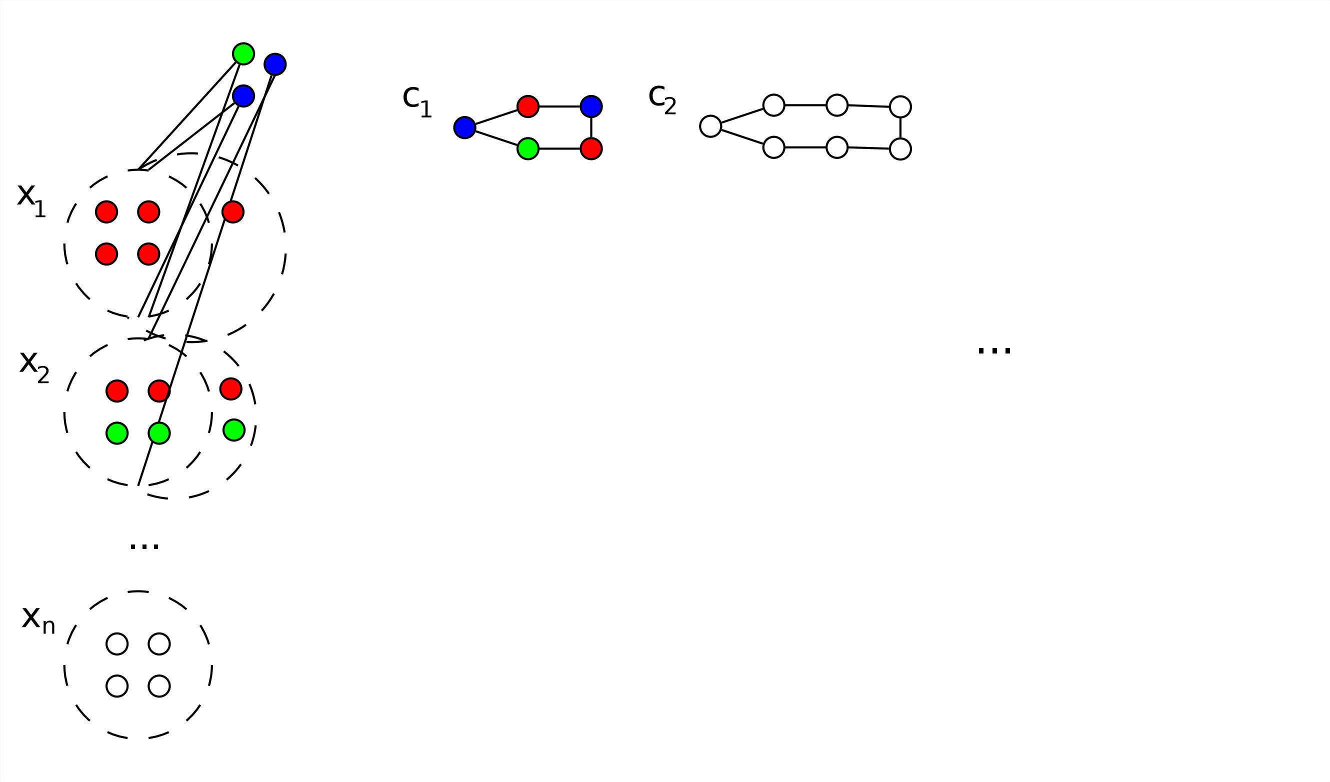
- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- $\rightarrow$ Green vertex $\leftrightarrow$ selected assignment
- Do the same for other variables of c_1

Main Reduction – Step 2



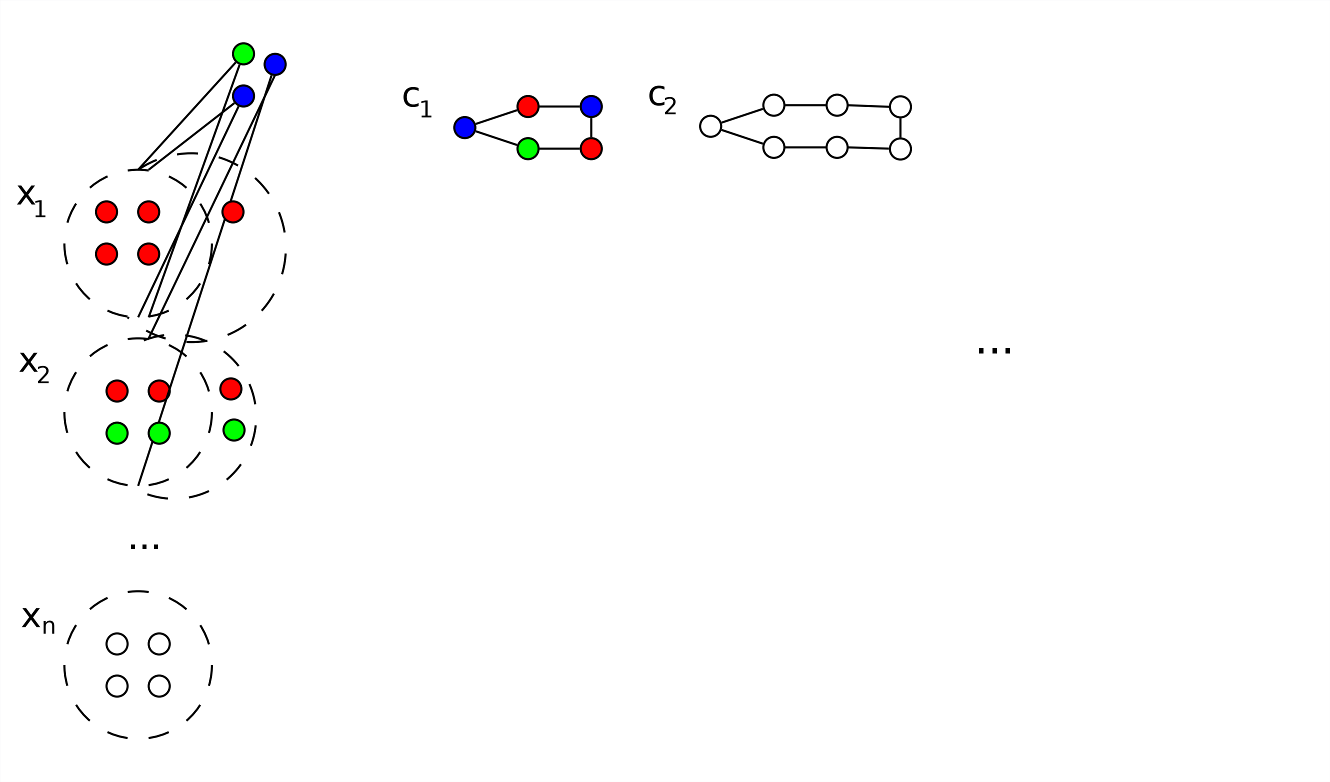
- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- $\rightarrow$ Green vertex $\leftrightarrow$ selected assignment
- Do the same for other variables of c_1

Main Reduction – Step 2



- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- $\rightarrow$ Green vertex $\leftrightarrow$ selected assignment
- Do the same for other constraints

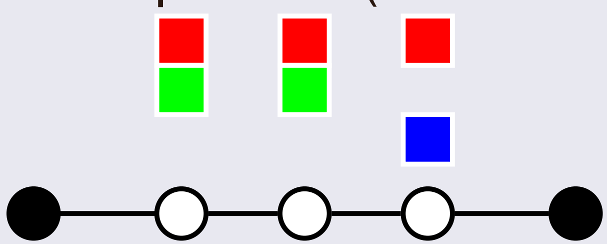
Main Reduction – Step 2



- We maintain n label sets (one for each variable).
- Invariant: Colors used $\leftrightarrow$ value
- $\rightarrow$ Green vertex $\leftrightarrow$ selected assignment
- Do the same for other constraints
- Repeating the sequence of constraints kn times ensures consistency!

Main Reduction – Gadgets

- List Coloring
  - Implemented by adding a complete k-partite graph to G, connecting each vertex with appropriate parts.
  - Tricky part: maintain clique-width.
- Weak Edges
  - Edges that only rule out one pair of colors (c_1, c_2).
  - Example: No (Red Blue)



- Implications
  - Implemented with weak edges.

Conclusions

Summary:

- Under SETH, $(2^k - 2)^w$ is the **correct** complexity of Coloring on clique-width, for any constant k.
- Similarly "fine tight" bounds for modular treewidth.

Open Problems:

- Why/how/when does complexity go from $2^{k \cdot w}$ to k^{2^w}???

Conclusions

Summary:

- Under SETH, $(2^k-2)^w$ is the **correct** complexity of Coloring on clique-width, for any constant k.
- Similarly "fine tight" bounds for modular treewidth.

Open Problems:

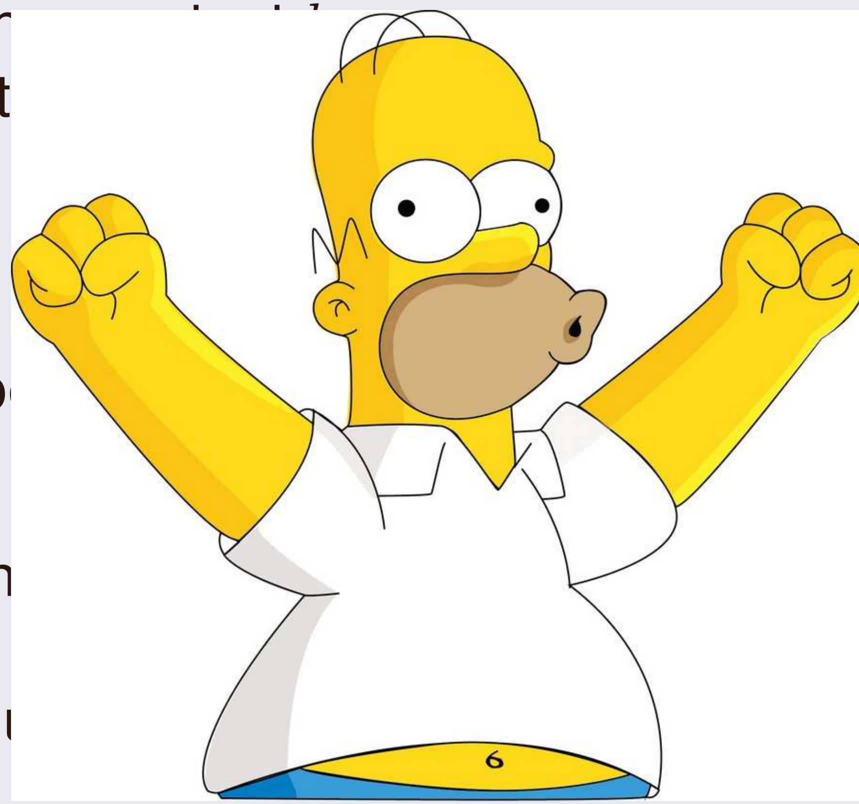
- Why/how/when does complexity go from $2^{k\cdot w}$ to k^{2^w}???
- Approximation?
  - Consistent with current knowledge: 2^{tw} 2-approximation for Coloring?
  - Can we distinguish 3 from 7-colorable graphs in 2^{tw}?

Summary:

- Under SETH, $(2^k - 2)^w$ is the **correct** complexity of Coloring on clique-width, for ar
- Similarly “fine tight

Open Problems:

- Why/how/when do ???
- Approximation?
  - Consistent with imation for Coloring?
  - Can we distingu 2^{tw}?



Thank you!