Parameterized Algorithms

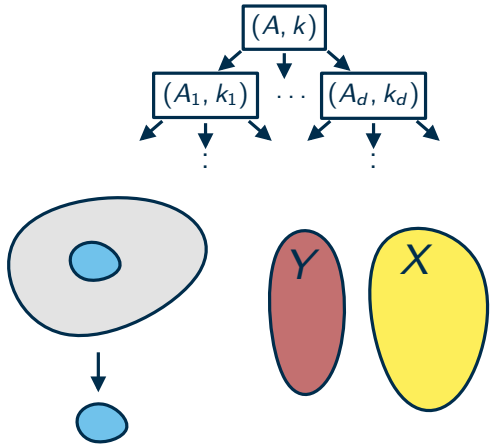
Lower Bounds: Reductions and the W-Hierarchy

Thomas Bläsius

Content

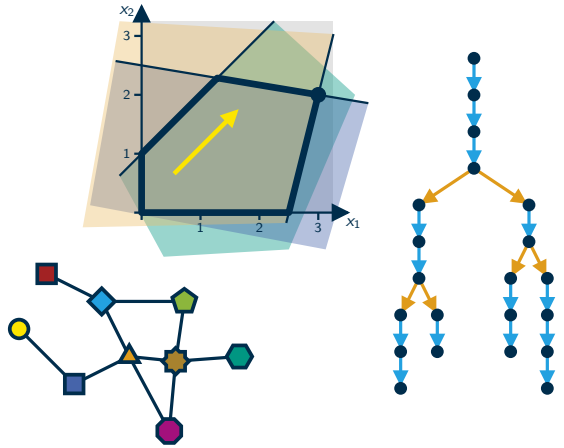
Basic toolbox

- bounded search trees
- kernelization
- iterative compression



Extended toolbox

- linear programs
- branch-and-reduce
- color coding

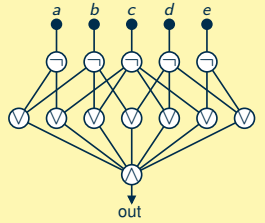


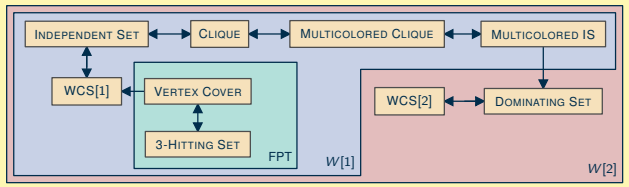
Tree width

- dynamic programming
- chordal and planar graphs
- Courcelle's theorem

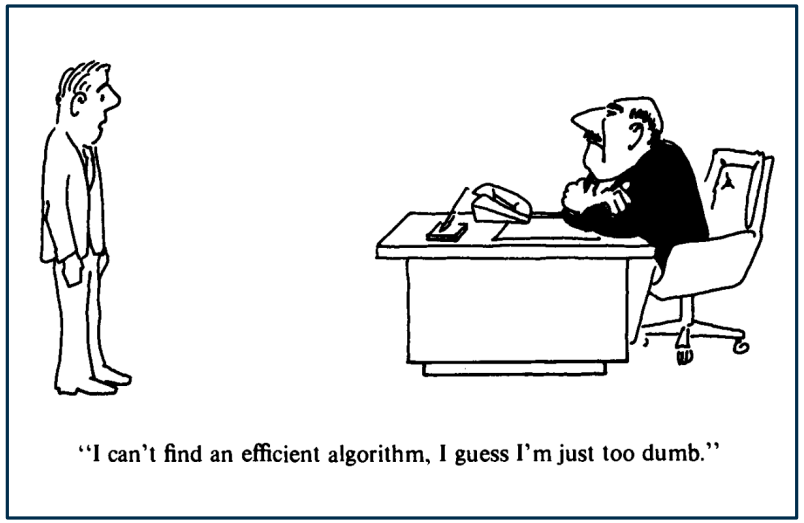
Lower bonds

- kernel lower bounds
- parameterized reductions
- circuits and the W-hierarchy
- ETH and SETH

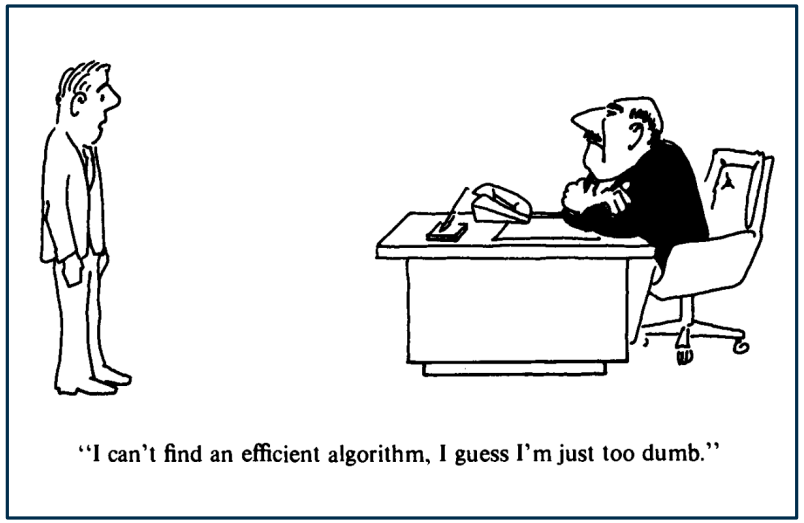




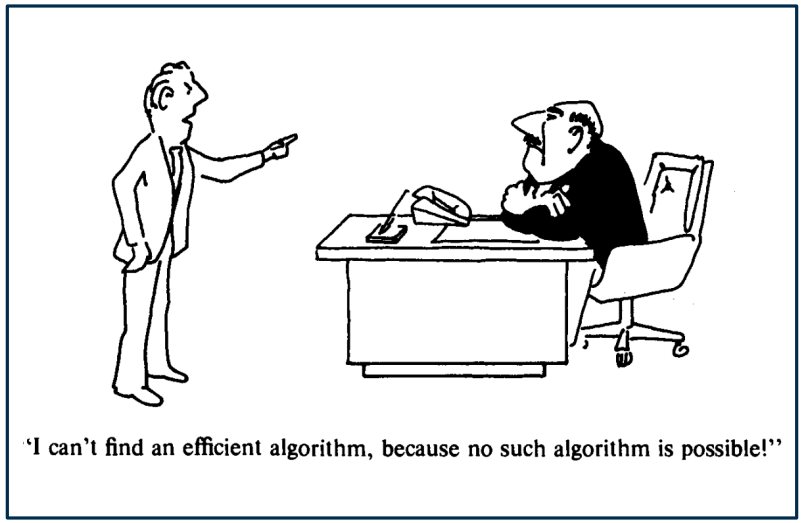
Why do we want lower bounds?



Why do we want lower bounds?

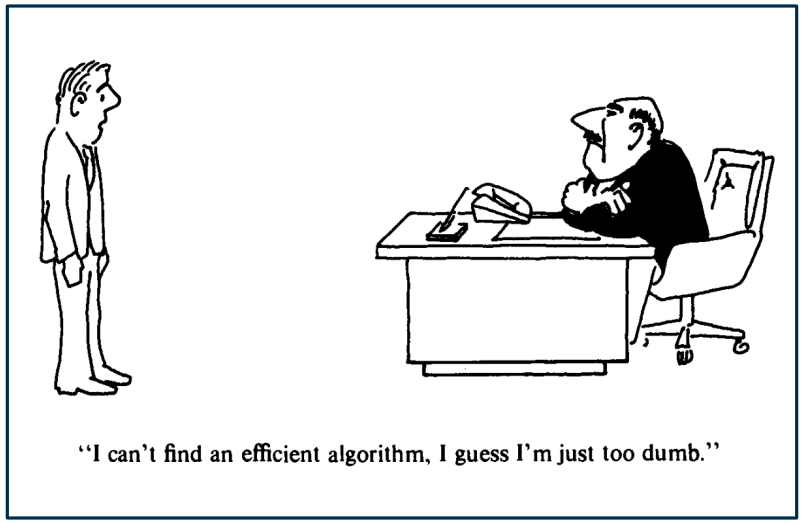


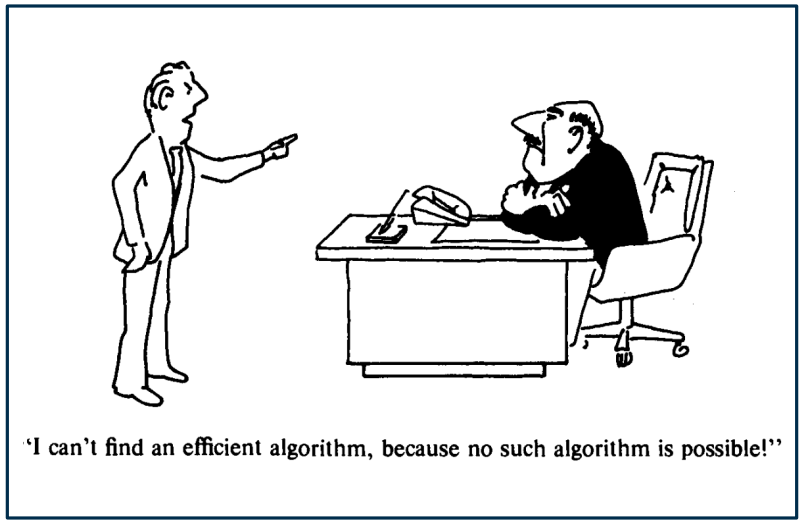
"I can't find an efficient algorithm, I guess I'm just too dumb."



"I can't find an efficient algorithm, because no such algorithm is possible!"

Why do we want lower bounds?

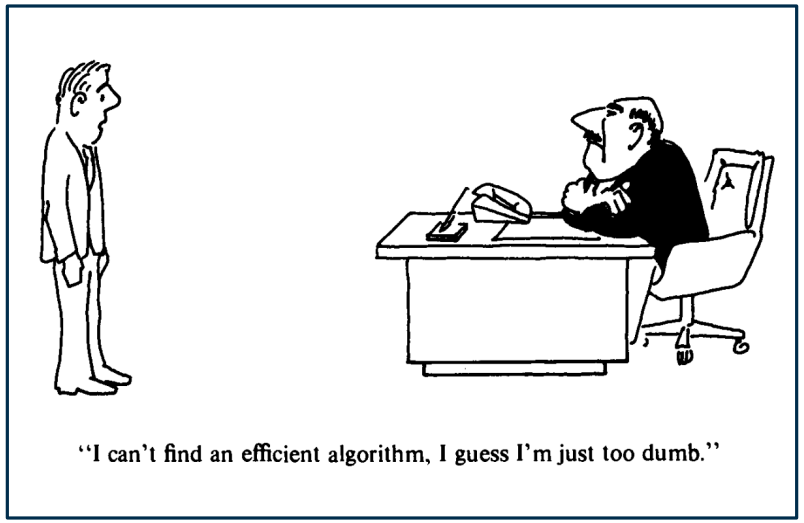


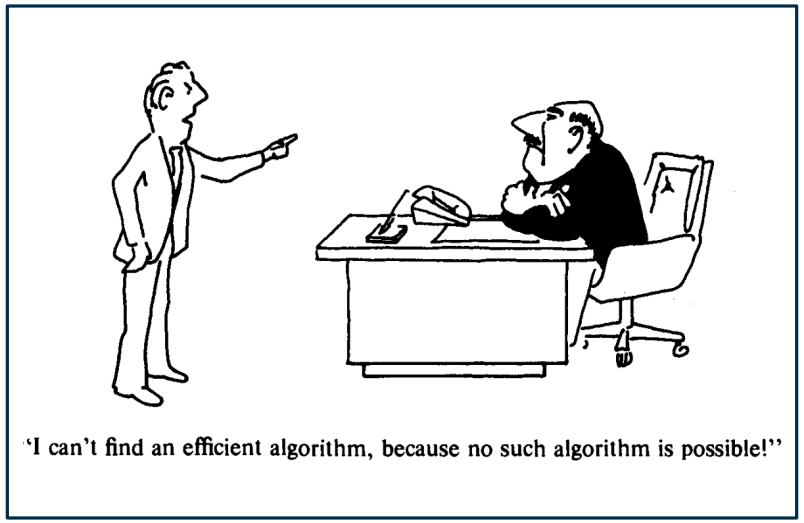


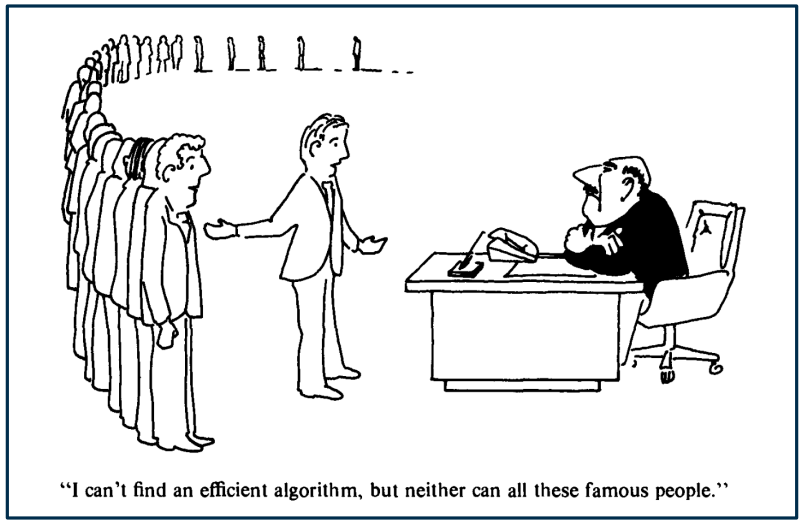
Problem

- non-existence of an algorithm is difficult to prove

Why do we want lower bounds?







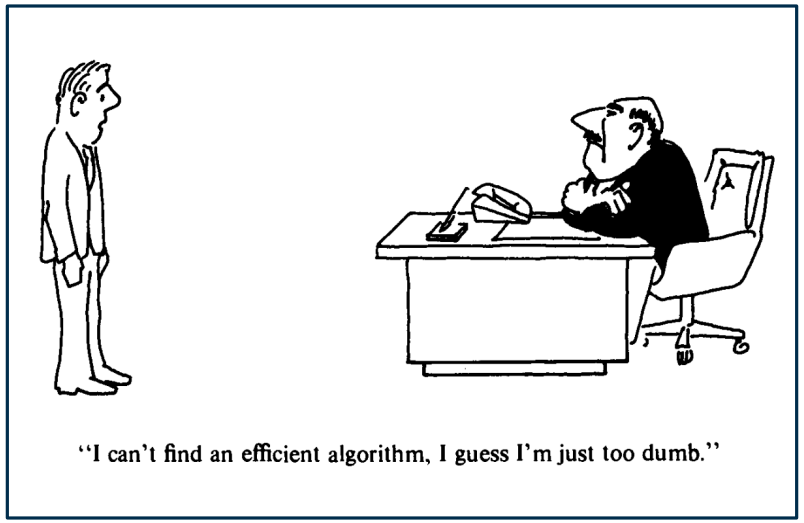
Problem

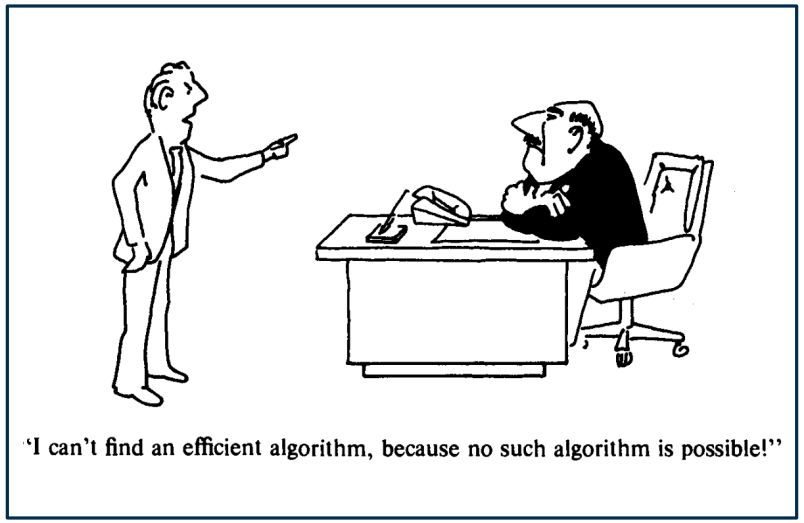
- non-existence of an algorithm is difficult to prove

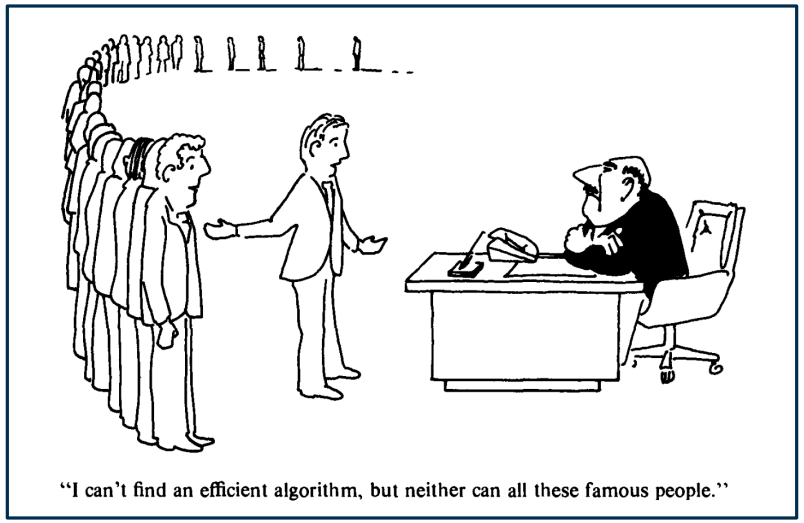
Solution

- show: my problem efficiently solvable $\Rightarrow$ a known hard problem efficiently solvable

Why do we want lower bounds?







Problem

- non-existence of an algorithm is difficult to prove

Solution

- show: my problem efficiently solvable $\Rightarrow$ a known hard problem efficiently solvable
- tool: reduction between problems

Polynomial reductions

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$
- I is yes-instance $\Leftrightarrow$ I' is yes-instance

Polynomial reductions

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$
- I is yes-instance $\Leftrightarrow$ I' is yes-instance
- the map must be computable in polynomial time

Polynomial reductions

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$
- I is yes-instance $\Leftrightarrow$ I' is yes-instance
- the map must be computable in polynomial time

Implication

- polynomial algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial algorithm for $\mathcal{L}$

Polynomial reductions

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$
- I is yes-instance $\Leftrightarrow$ I' is yes-instance
- the map must be computable in polynomial time

Implication

- polynomial algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for “efficient” = “polynomial”)

Polynomial reductions

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$
- I is yes-instance $\Leftrightarrow$ I' is yes-instance
- the map must be computable in polynomial time

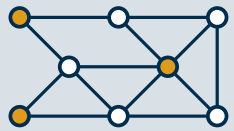
Implication

- polynomial algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for "efficient" = "polynomial")

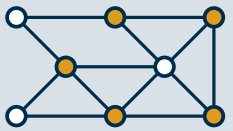
Example: Independent Set to Vertex Cover

Independent Set



$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$

Vertex Cover



$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \geq 1$

Polynomial reductions

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$
- I is yes-instance $\Leftrightarrow$ I' is yes-instance
- the map must be computable in polynomial time

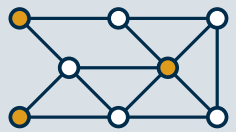
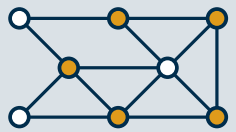
Implication

- polynomial algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for "efficient" = "polynomial")

Example: Independent Set to Vertex Cover

- reduction: G has IS of size k $\Leftrightarrow$ G has VC of size $n - k$

Independent Set	Vertex Cover
	
$V' \subseteq V$, such that $\forall e \in E: \lvert e \cap V' \rvert \leq 1$	$V' \subseteq V$, such that $\forall e \in E: \lvert e \cap V' \rvert \geq 1$

Polynomial reductions

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$
- I is yes-instance $\Leftrightarrow$ I' is yes-instance
- the map must be computable in polynomial time

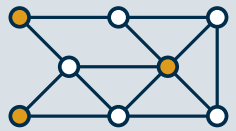
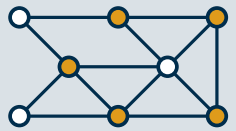
Implication

- polynomial algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for "efficient" = "polynomial")

Example: Independent Set to Vertex Cover

- reduction: G has IS of size k $\Leftrightarrow$ G has VC of size $n - k$
- thus: VC $\in$ P $\Rightarrow$ IS $\in$ P

Independent Set	Vertex Cover
	
$V' \subseteq V$, such that $\forall e \in E: \lvert e \cap V' \rvert \leq 1$	$V' \subseteq V$, such that $\forall e \in E: \lvert e \cap V' \rvert \geq 1$

Polynomial reductions

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$
- I is yes-instance $\Leftrightarrow$ I' is yes-instance
- the map must be computable in polynomial time

Implication

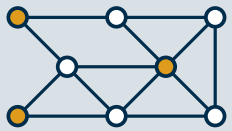
- polynomial algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for "efficient" = "polynomial")

Example: Independent Set to Vertex Cover

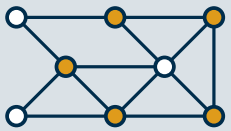
- reduction: G has IS of size k $\Leftrightarrow$ G has VC of size $n - k$
- thus: VC $\in$ P $\Rightarrow$ IS $\in$ P
- And "VC $\in$ FPT $\Rightarrow$ IS $\in$ FPT"?

Independent Set



$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$

Vertex Cover



$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \geq 1$

Polynomial reductions

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$
- I is yes-instance $\Leftrightarrow$ I' is yes-instance
- the map must be computable in polynomial time

Implication

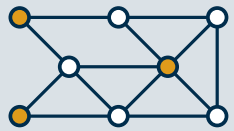
- polynomial algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for "efficient" = "polynomial")

Example: Independent Set to Vertex Cover

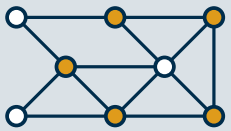
- reduction: G has IS of size k $\Leftrightarrow$ G has VC of size $n - k$
- thus: VC $\in$ P $\Rightarrow$ IS $\in$ P
- And "VC $\in$ FPT $\Rightarrow$ IS $\in$ FPT"? $\rightarrow$ parameter too large

Independent Set



$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$

Vertex Cover



$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \geq 1$

Polynomial reductions

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance I of $\mathcal{L}$ to an instance I' of $\mathcal{L}'$
- I is yes-instance $\Leftrightarrow$ I' is yes-instance
- the map must be computable in polynomial time

Implication

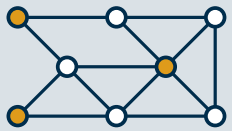
- polynomial algorithm for $\mathcal{L}'$ $\Rightarrow$ polynomial algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for "efficient" = "polynomial")

Example: Independent Set to Vertex Cover

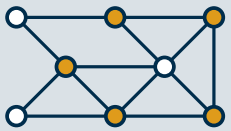
- reduction: G has IS of size k $\Leftrightarrow$ G has VC of size $n - k$
- thus: VC $\in$ P $\Rightarrow$ IS $\in$ P
- And "VC $\in$ FPT $\Rightarrow$ IS $\in$ FPT"? $\rightarrow$ parameter too large
- for "efficient" = "FPT", we need a different reduction

Independent Set



$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$

Vertex Cover



$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \geq 1$

Parameterized reduction

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to instance (I', k') of $\mathcal{L}'$ with $k' \leq g(k)$
- (I, k) yes-instance $\Leftrightarrow$ (I', k') yes-instance
- the map must be computable in FPT-time $f(k) \cdot |I|^{O(1)}$ (f and g computable)

Parameterized reduction

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to instance (I', k') of $\mathcal{L}'$ with $k' \leq g(k)$
- (I, k) yes-instance $\Leftrightarrow$ (I', k') yes-instance
- the map must be computable in FPT-time $f(k) \cdot |I|^{O(1)}$ (f and g computable)

Implication

- FPT algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$

Parameterized reduction

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to instance (I', k') of $\mathcal{L}'$ with $k' \leq g(k)$
- (I, k) yes-instance $\Leftrightarrow$ (I', k') yes-instance
- the map must be computable in FPT-time $f(k) \cdot |I|^{O(1)}$ (f and g computable)

Implication

- FPT algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for "efficient" = "FPT")

Parameterized reduction

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to instance (I', k') of $\mathcal{L}'$ with $k' \leq g(k)$
- (I, k) yes-instance $\Leftrightarrow$ (I', k') yes-instance
- the map must be computable in FPT-time $f(k) \cdot |I|^{O(1)}$ (f and g computable)

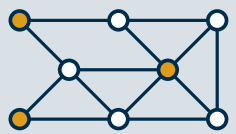
Implication

- FPT algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for "efficient" = "FPT")

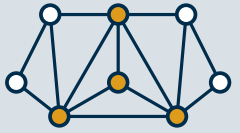
Example: Independent Set to Clique

Independent Set



$V' \subseteq V$, such that $\forall e \in E : |e \cap V'| \leq 1$

Clique



$V' \subseteq V$, such that $G[V']$ complete

Parameterized reduction

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to instance (I', k') of $\mathcal{L}'$ with $k' \leq g(k)$
- (I, k) yes-instance $\Leftrightarrow$ (I', k') yes-instance
- the map must be computable in FPT-time $f(k) \cdot |I|^{O(1)}$ (f and g computable)

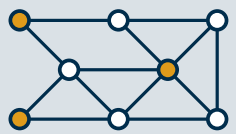
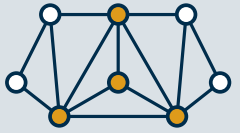
Implication

- FPT algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for "efficient" = "FPT")

Example: Independent Set to Clique

- G has IS of size k $\Leftrightarrow$ complement of G has k-clique

Independent Set	Clique
	
$V' \subseteq V$, such that $\forall e \in E : \lvert e \cap V' \rvert \leq 1$	$V' \subseteq V$, such that $G[V']$ complete

Parameterized reduction

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to instance (I', k') of $\mathcal{L}'$ with $k' \leq g(k)$
- (I, k) yes-instance $\Leftrightarrow$ (I', k') yes-instance
- the map must be computable in FPT-time $f(k) \cdot |I|^{O(1)}$ (f and g computable)

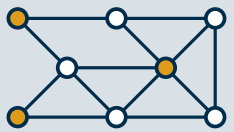
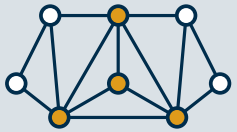
Implication

- FPT algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for "efficient" = "FPT")

Example: Independent Set to Clique

- G has IS of size k $\Leftrightarrow$ complement of G has k-clique
- thus: Clique $\in$ FPT $\Rightarrow$ IS $\in$ FPT

Independent Set	Clique
	
$V' \subseteq V$, such that $\forall e \in E : \lvert e \cap V' \rvert \leq 1$	$V' \subseteq V$, such that $G[V']$ complete

Parameterized reduction

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to instance (I', k') of $\mathcal{L}'$ with $k' \leq g(k)$
- (I, k) yes-instance $\Leftrightarrow$ (I', k') yes-instance
- the map must be computable in FPT-time $f(k) \cdot |I|^{O(1)}$ (f and g computable)

Implication

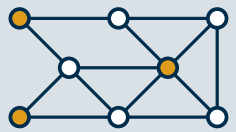
- FPT algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for "efficient" = "FPT")

Example: Independent Set to Clique

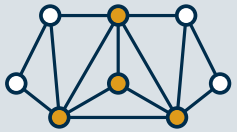
- G has IS of size k $\Leftrightarrow$ complement of G has k-clique
- thus: Clique $\in$ FPT $\Rightarrow$ IS $\in$ FPT
- inverse also holds: Clique $\in$ FPT $\Leftrightarrow$ IS $\in$ FPT

Independent Set



$V' \subseteq V$, such that $\forall e \in E : |e \cap V'| \leq 1$

Clique



$V' \subseteq V$, such that $G[V']$ complete

Parameterized reduction

Reduction from problem $\mathcal{L}$ to problem $\mathcal{L}'$

- map each instance (I, k) of $\mathcal{L}$ to instance (I', k') of $\mathcal{L}'$ with $k' \leq g(k)$
- (I, k) yes-instance $\Leftrightarrow$ (I', k') yes-instance
- the map must be computable in FPT-time $f(k) \cdot |I|^{O(1)}$ (f and g computable)

Implication

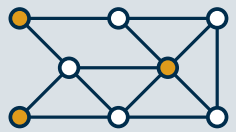
- FPT algorithm for $\mathcal{L}'$ $\Rightarrow$ FPT algorithm for $\mathcal{L}$
- my problem $\mathcal{L}'$ efficiently solvable $\Rightarrow$ known hard problem $\mathcal{L}$ efficiently solvable

(for "efficient" = "FPT")

Example: Independent Set to Clique

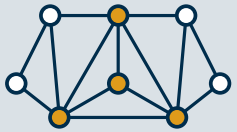
- G has IS of size k $\Leftrightarrow$ complement of G has k-clique
- thus: Clique $\in$ FPT $\Rightarrow$ IS $\in$ FPT
- inverse also holds: Clique $\in$ FPT $\Leftrightarrow$ IS $\in$ FPT
- conjecture: Clique, IS $\notin$ FPT

Independent Set



$V' \subseteq V$, such that $\forall e \in E : |e \cap V'| \leq 1$

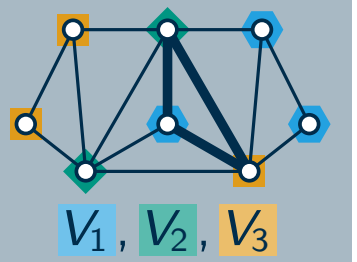
Clique



$V' \subseteq V$, such that $G[V']$ complete

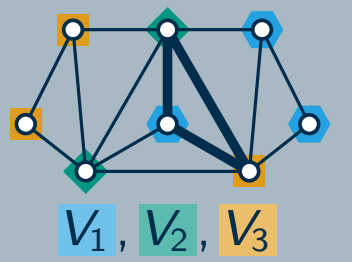
Clique $\rightarrow$ Multicolored Clique

Problem: Multicolored Clique
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?

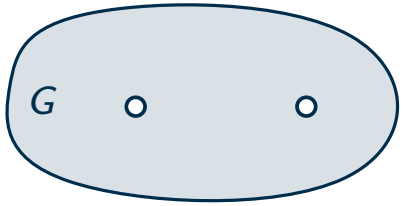


CLIQUE $\rightarrow$ MULTICOLORED CLIQUE

Problem: MULTICOLORED CLIQUE
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?

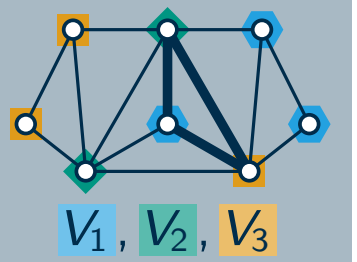


instance $(G, 3)$ of CLIQUE



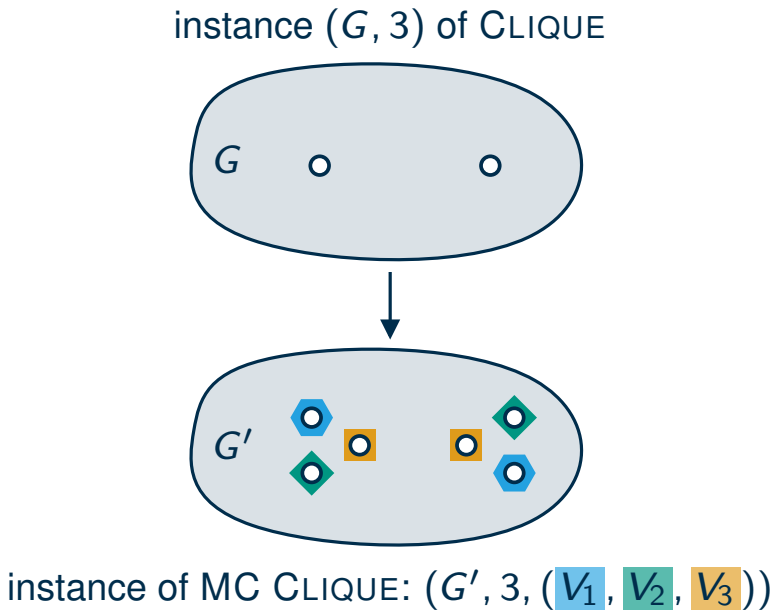
Clique → Multicolored Clique

Problem: Multicolored Clique
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?



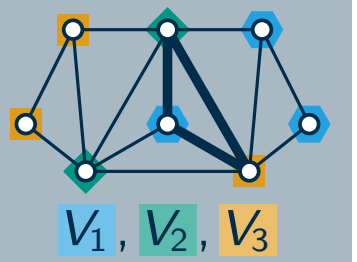
Reduction: Clique to Multicolored Clique

- replace $v \in V$ by $v^1, \ldots, v^k$ with $v^i \in V_i$



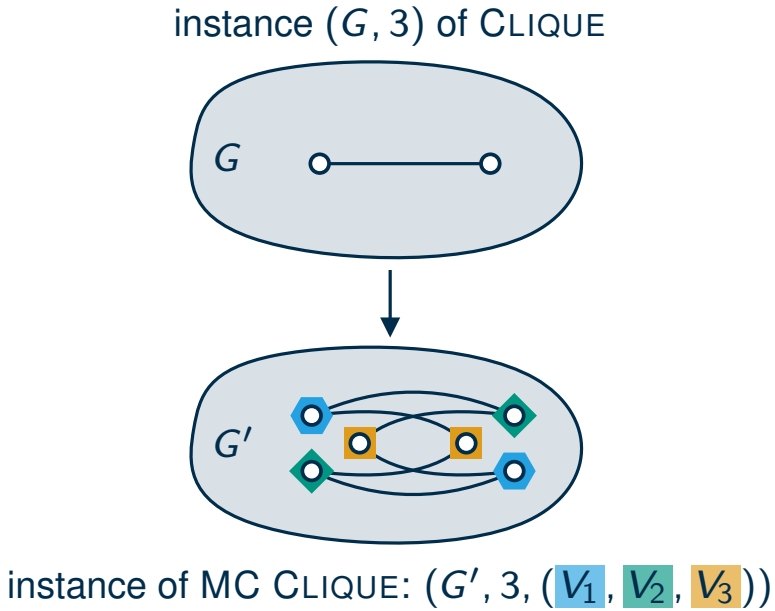
Clique → Multicolored Clique

Problem: Multicolored Clique
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?



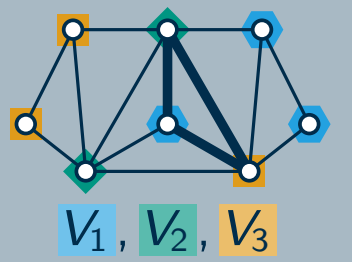
Reduction: Clique to Multicolored Clique

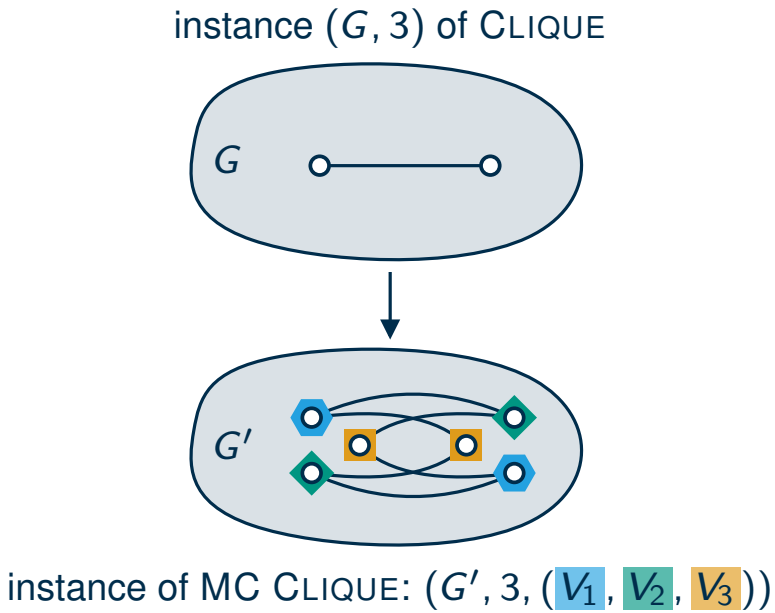
- replace $v \in V$ by $v^1, \ldots, v^k$ with $v^i \in V_i$
- for $uv \in E$ connect u^i with v^j for all $i \neq j$
- set $k' = k$



Clique → Multicolored Clique

Problem: Multicolored Clique
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?





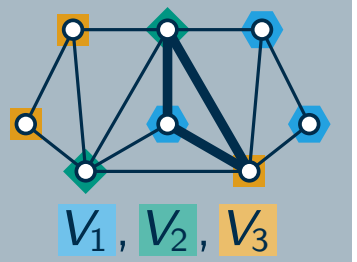
Reduction: Clique to Multicolored Clique

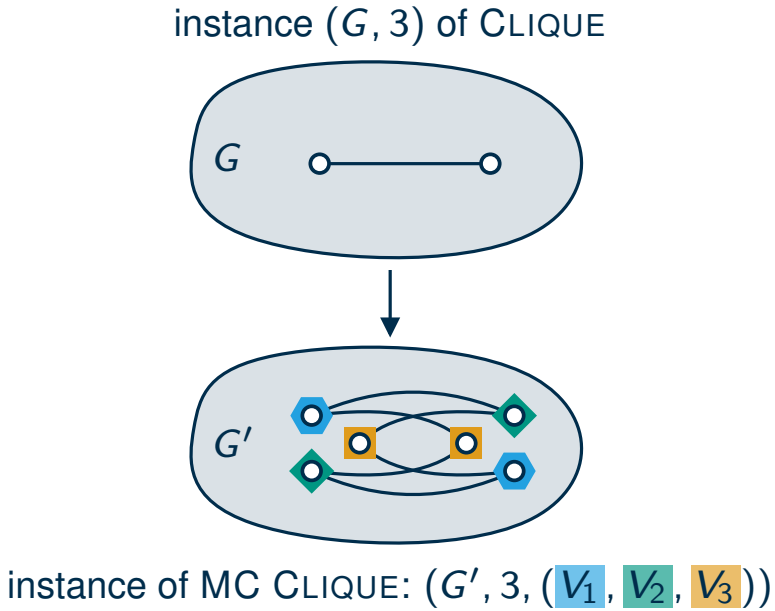
- replace $v \in V$ by $v^1, \ldots, v^k$ with $v^i \in V_i$
- for $uv \in E$ connect u^i with v^j for all $i \neq j$
- set $k' = k$

k-clique in G $\Rightarrow$ colorful k-clique in G'

Clique → Multicolored Clique

Problem: Multicolored Clique
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?





Reduction: Clique to Multicolored Clique

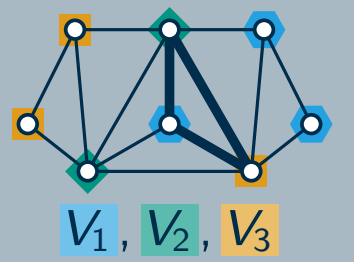
- replace $v \in V$ by $v^1, \ldots, v^k$ with $v^i \in V_i$
- for $uv \in E$ connect u^i with v^j for all $i \neq j$
- set $k' = k$

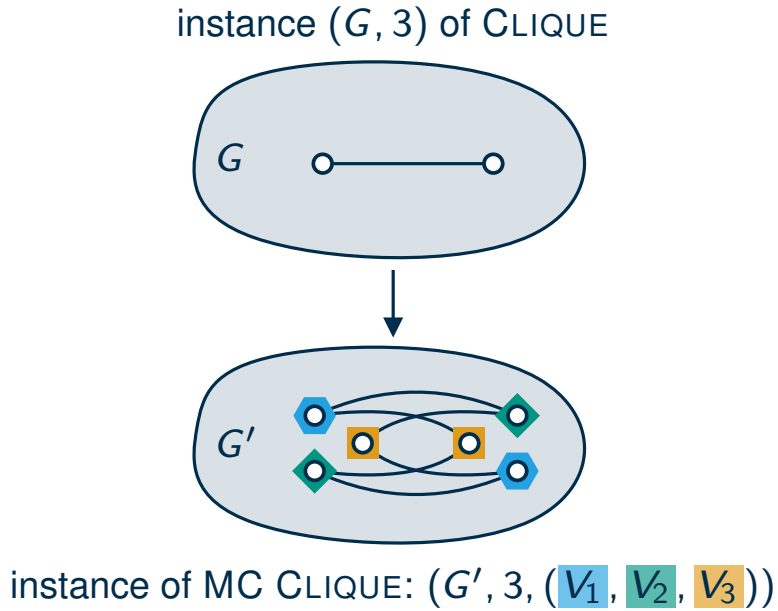
k-clique in G $\Rightarrow$ colorful k-clique in G'

- let $v_1, \ldots, v_k$ form a clique in G
- then $v_1^1, \ldots, v_k^k$ is a clique in G'
- all vertices have different colors

Clique → Multicolored Clique

Problem: Multicolored Clique
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?





Reduction: Clique to Multicolored Clique

- replace $v \in V$ by $v^1, \ldots, v^k$ with $v^i \in V_i$
- for $uv \in E$ connect u^i with v^j for all $i \neq j$
- set $k' = k$

k-clique in G ⇒ colorful k-clique in G'

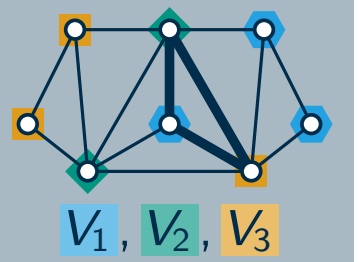
- let $v_1, \ldots, v_k$ form a clique in G
- then $v_1^1, \ldots, v_k^k$ is a clique in G'
- all vertices have different colors

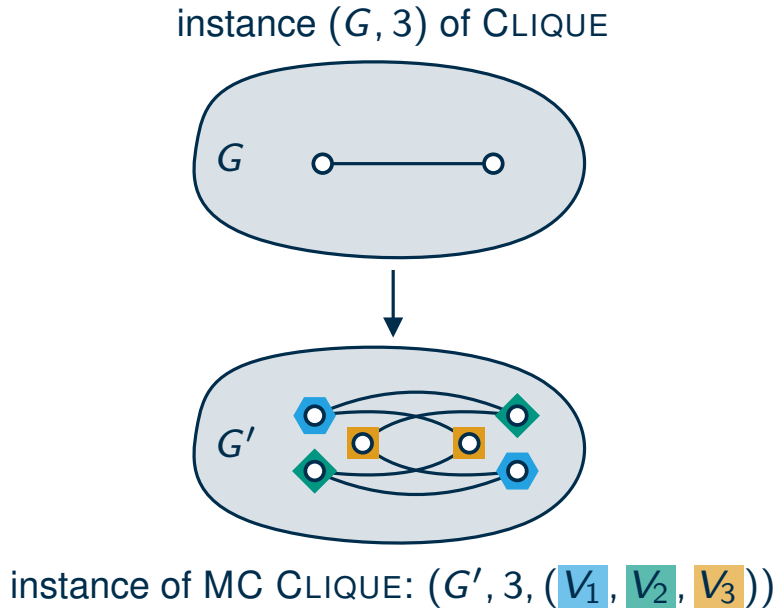
colorful k-clique in G' ⇒ k-clique in G

- colorful clique: $v^1_{\pi(1)}, \ldots, v^k_{\pi(k)}$ with $\pi : [k] \to [n]$

Clique → Multicolored Clique

Problem: Multicolored Clique
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?





Reduction: Clique to Multicolored Clique

- replace $v \in V$ by $v^1, \ldots, v^k$ with $v^i \in V_i$
- for $uv \in E$ connect u^i with v^j for all $i \neq j$
- set $k' = k$

k-clique in G ⇒ colorful k-clique in G'

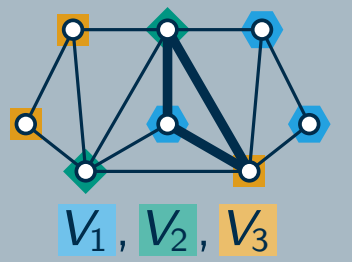
- let $v_1, \ldots, v_k$ form a clique in G
- then $v_1^1, \ldots, v_k^k$ is a clique in G'
- all vertices have different colors

colorful k-clique in G' ⇒ k-clique in G

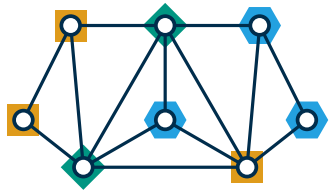
- colorful clique: $v^1_{\pi(1)}, \ldots, v^k_{\pi(k)}$ with $\pi : [k] \to [n]$
- $v_{\pi(1)}, \ldots, v_{\pi(k)}$ form a clique in G
- it has size k as π is injective

MULTICOLORED CLIQUE → CLIQUE

Problem: MULTICOLORED CLIQUE
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?

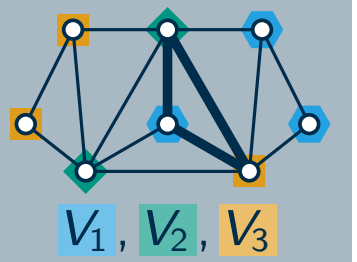


instance of MC CLIQUE: $(G, 3, (V_1, V_2, V_3))$



Multicolored Clique → Clique

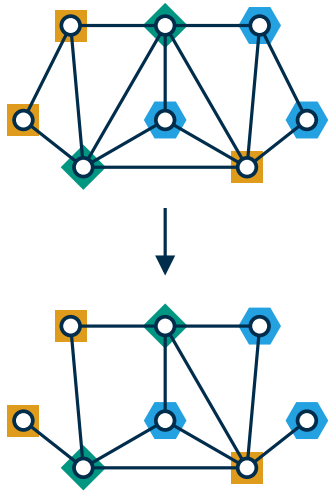
Problem: Multicolored Clique
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?



Reduction: Multicolored Clique to Clique

- delete edges between equally colored vertices
- set $k' = k$

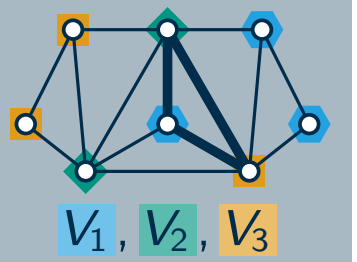
instance of MC Clique: $(G, 3, (V_1, V_2, V_3))$



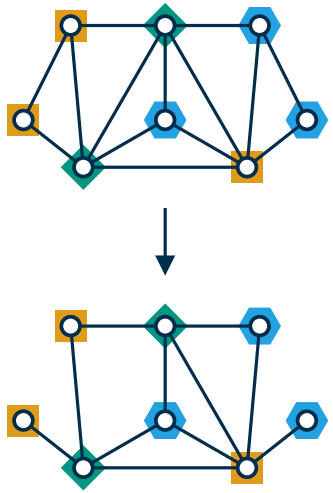
instance $(G', 3)$ of Clique

MULTICOLORED CLIQUE → CLIQUE

Problem: MULTICOLORED CLIQUE
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?



instance of MC CLIQUE: $(G, 3, (V_1, V_2, V_3))$



instance $(G', 3)$ of CLIQUE

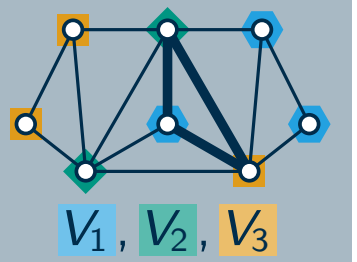
Reduction: MULTICOLORED CLIQUE to CLIQUE

- delete edges between equally colored vertices
- set $k' = k$

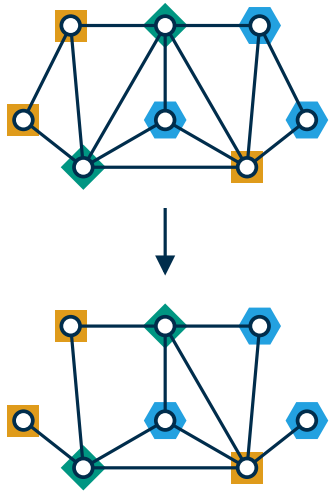
colorful k-clique in G $\Rightarrow$ k-clique in G'

MULTICOLORED CLIQUE → CLIQUE

Problem: MULTICOLORED CLIQUE
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?



instance of MC CLIQUE: $(G, 3, (V_1, V_2, V_3))$



instance $(G', 3)$ of CLIQUE

Reduction: MULTICOLORED CLIQUE to CLIQUE

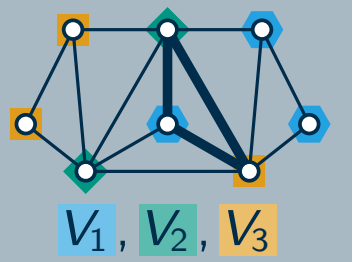
- delete edges between equally colored vertices
- set $k' = k$

colorful k-clique in G $\Rightarrow$ k-clique in G'

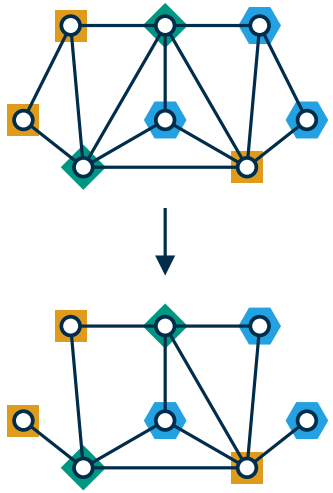
- no edges of the colorful clique are deleted
- it remains a clique of size k in G'

MULTICOLORED CLIQUE → CLIQUE

Problem: MULTICOLORED CLIQUE
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?



instance of MC CLIQUE: $(G, 3, (V_1, V_2, V_3))$



instance $(G', 3)$ of CLIQUE

Reduction: MULTICOLORED CLIQUE to CLIQUE

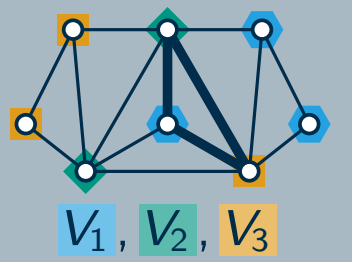
- delete edges between equally colored vertices
- set $k' = k$

colorful k-clique in G $\Rightarrow$ k-clique in G' **k-clique in G' $\Rightarrow$ colorful k-clique in G**

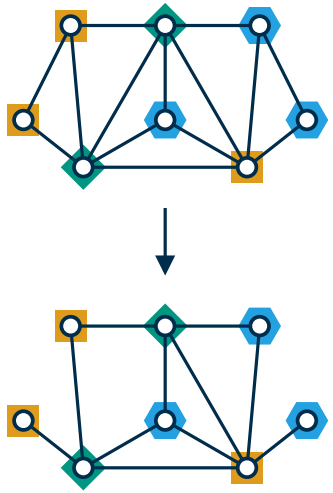
- no edges of the colorful clique are deleted
- it remains a clique of size k in G'

MULTICOLORED CLIQUE → CLIQUE

Problem: MULTICOLORED CLIQUE
Given a graph $G = (V, E)$, a parameter k and a partition $(V_1, \ldots, V_k)$ of the vertices. Is there a clique with exactly one vertex from each V_i?



instance of MC CLIQUE: $(G, 3, (V_1, V_2, V_3))$



instance $(G', 3)$ of CLIQUE

Reduction: MULTICOLORED CLIQUE to CLIQUE

- delete edges between equally colored vertices
- set $k' = k$

colorful k-clique in G $\Rightarrow$ k-clique in G'

- no edges of the colorful clique are deleted
- it remains a clique of size k in G'

k-clique in G' $\Rightarrow$ colorful k-clique in G

- no vertices with equal color are adjacent in G'
- each clique in G' is colorful in G

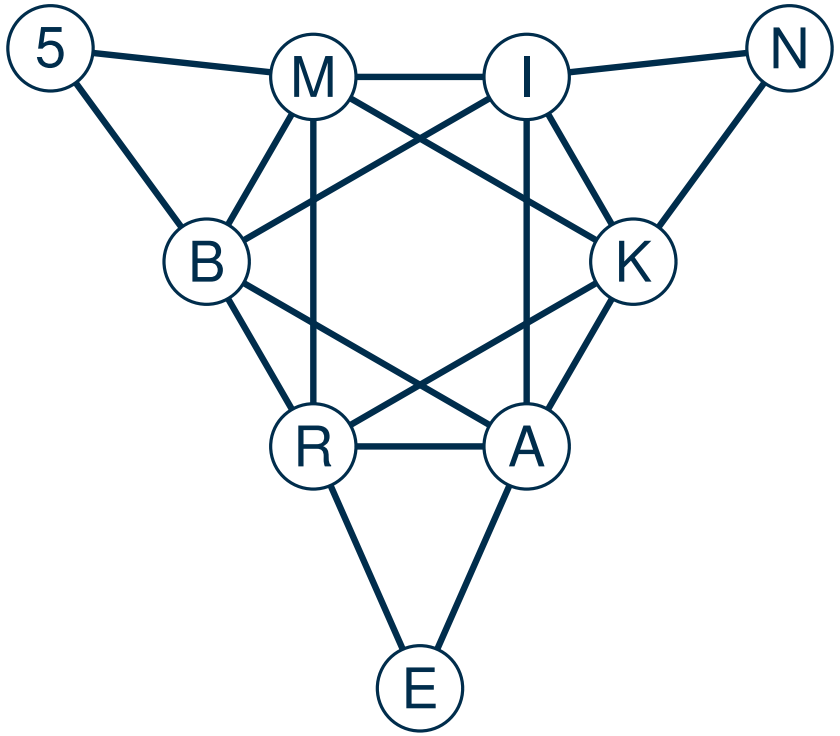
Reductions so far

Reductions so far

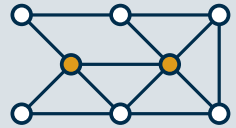
Dominating Set

Goal

- select as few vertices as possible
- for every vertex: the vertex or one of its neighbors is selected



Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

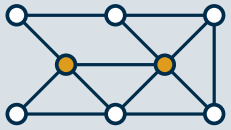
Dominating Set

Reduce Multicolored Independent Set to Dominating Set

MC Independent Set

$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

Dominating Set

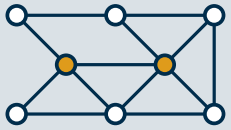
Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties in an instance of Dominating Set
  - exactly one vertex of each color must be chosen
  - forbid the choice of certain vertex pairs

MC Independent Set

$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

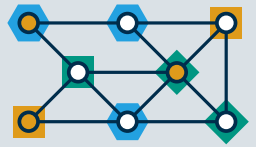
Dominating Set

Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties in an instance of Dominating Set
  - exactly one vertex of each color must be chosen
  - forbid the choice of certain vertex pairs

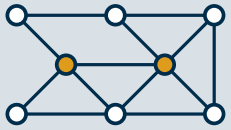
Exactly one vertex for each color

MC Independent Set

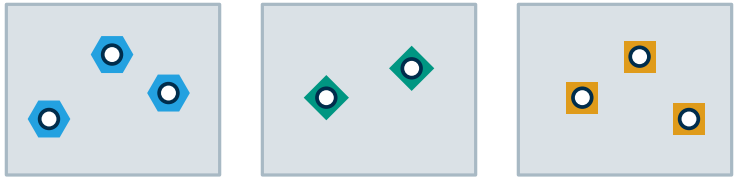


$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$



Dominating Set

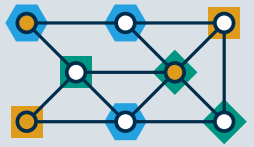
Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties in an instance of Dominating Set
  - exactly one vertex of each color must be chosen
  - forbid the choice of certain vertex pairs

Exactly one vertex for each color

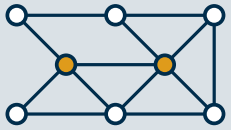
- vertex x_i with edges to all vertices in V_i $\rightarrow$ must choose one of $V_i \cup \{x_i\}$

MC Independent Set

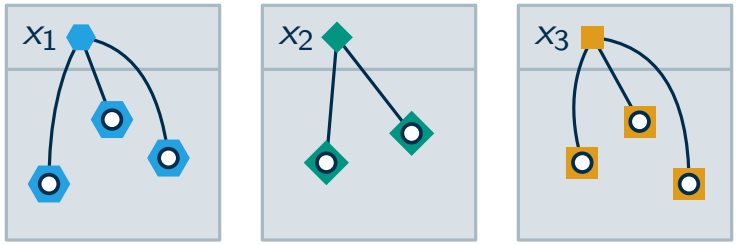


$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$



Dominating Set

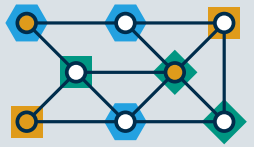
Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties in an instance of Dominating Set
  - exactly one vertex of each color must be chosen
  - forbid the choice of certain vertex pairs

Exactly one vertex for each color

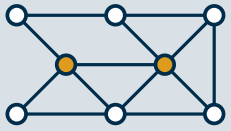
- vertex x_i with edges to all vertices in V_i → must choose one of $V_i \cup \{x_i\}$
- problem: one could choose x_i

MC Independent Set

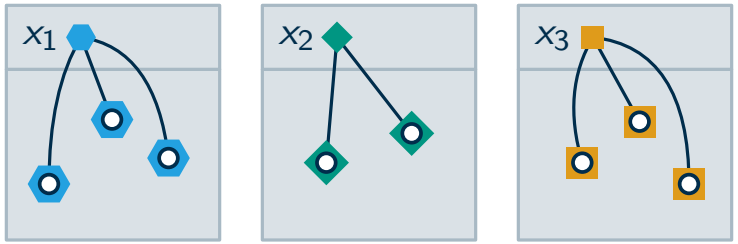


$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

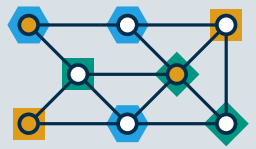


Dominating Set

Reduce Multicolored Independent Set to Dominating Set

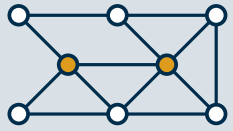
- enforce the following properties in an instance of Dominating Set
  - exactly one vertex of each color must be chosen
  - forbid the choice of certain vertex pairs

MC Independent Set



$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

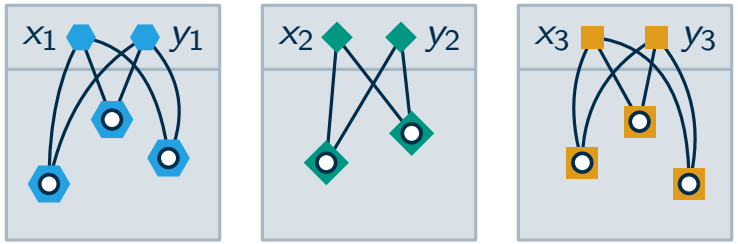
Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

Exactly one vertex for each color

- vertex x_i with edges to all vertices in V_i → must choose one of $V_i \cup \{x_i\}$
- problem: one could choose x_i
- solution: additional vertex y_i → choosing x_i and y_i is too expensive



Dominating Set

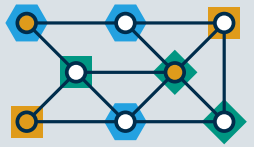
Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties in an instance of Dominating Set
  - exactly one vertex of each color must be chosen
  - forbid the choice of certain vertex pairs

Exactly one vertex for each color

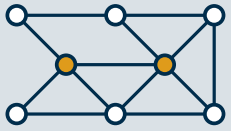
- vertex x_i with edges to all vertices in V_i → must choose one of $V_i \cup \{x_i\}$
- problem: one could choose x_i
- solution: additional vertex y_i → choosing x_i and y_i is too expensive
- make it sufficient to choose one of V_i:

MC Independent Set

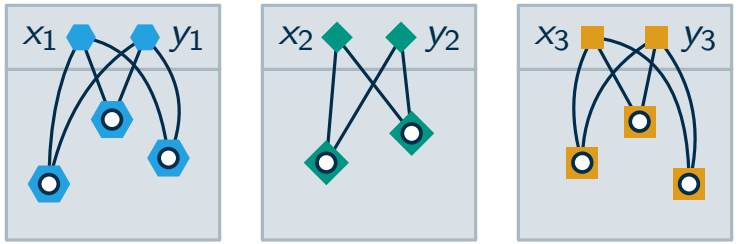


$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$



Dominating Set

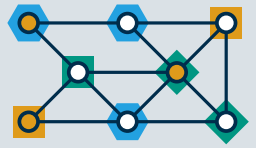
Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties in an instance of Dominating Set
  - exactly one vertex of each color must be chosen
  - forbid the choice of certain vertex pairs

Exactly one vertex for each color

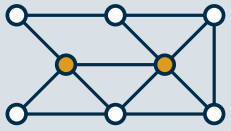
- vertex x_i with edges to all vertices in V_i → must choose one of $V_i \cup \{x_i\}$
- problem: one could choose x_i
- solution: additional vertex y_i → choosing x_i and y_i is too expensive
- make it sufficient to choose one of V_i: clique on V_i

MC Independent Set

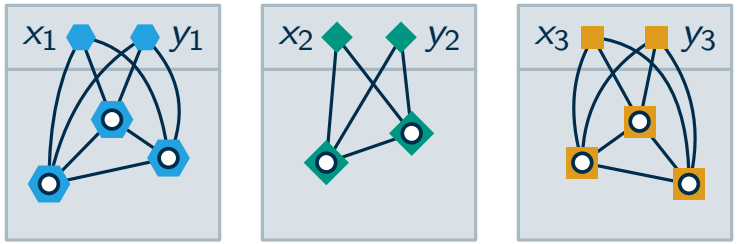


$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

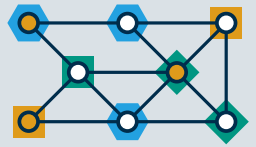


Dominating Set

Reduce Multicolored Independent Set to Dominating Set

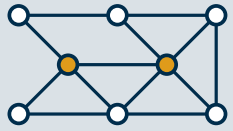
- enforce the following properties in an instance of Dominating Set
  - exactly one vertex of each color must be chosen ✓
  - forbid the choice of certain vertex pairs

MC Independent Set



$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

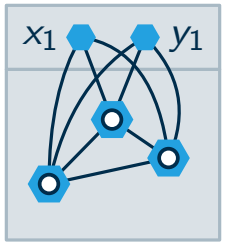
Dominating Set

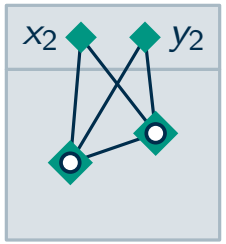


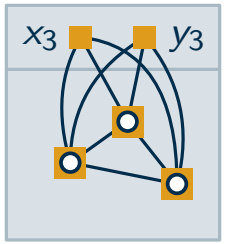
$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

Exactly one vertex for each color

- vertex x_i with edges to all vertices in V_i → must choose one of $V_i \cup \{x_i\}$
- problem: one could choose x_i
- solution: additional vertex y_i → choosing x_i and y_i is too expensive
- make it sufficient to choose one of V_i: clique on V_i







Dominating Set

Reduce Multicolored Independent Set to Dominating Set

- enforce the following properties in an instance of Dominating Set
  - exactly one vertex of each color must be chosen ✓
  - forbid the choice of certain vertex pairs

Exactly one vertex for each color

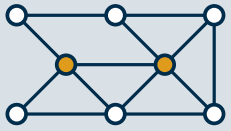
- vertex x_i with edges to all vertices in V_i → must choose one of $V_i \cup \{x_i\}$
- problem: one could choose x_i
- solution: additional vertex y_i → choosing x_i and y_i is too expensive
- make it sufficient to choose one of V_i: clique on V_i

Forbid to choose u and v at the same time

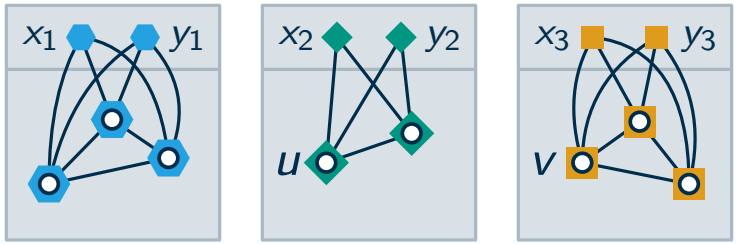
MC Independent Set

$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

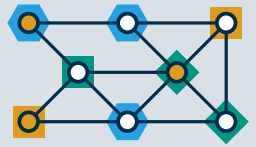


Dominating Set

Reduce Multicolored Independent Set to Dominating Set

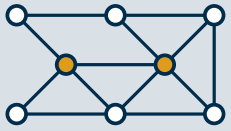
- enforce the following properties in an instance of Dominating Set
  - exactly one vertex of each color must be chosen ✓
  - forbid the choice of certain vertex pairs

MC Independent Set



$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

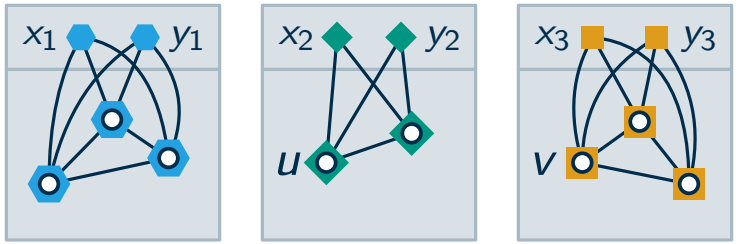
Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

Exactly one vertex for each color

- vertex x_i with edges to all vertices in V_i → must choose one of $V_i \cup \{x_i\}$
- problem: one could choose x_i
- solution: additional vertex y_i → choosing x_i and y_i is too expensive
- make it sufficient to choose one of V_i: clique on V_i



Forbid to choose u and v at the same time

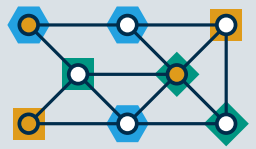
- idea: add vertex z that is covered unless u and v are chosen

Dominating Set

Reduce Multicolored Independent Set to Dominating Set

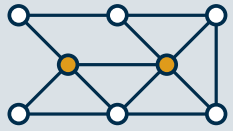
- enforce the following properties in an instance of Dominating Set
  - exactly one vertex of each color must be chosen ✓
  - forbid the choice of certain vertex pairs

MC Independent Set



$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

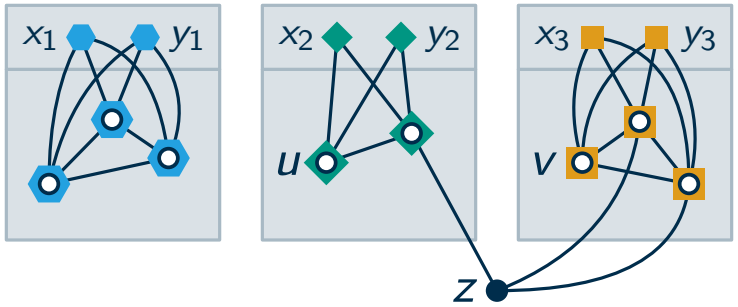
Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

Exactly one vertex for each color

- vertex x_i with edges to all vertices in V_i → must choose one of $V_i \cup \{x_i\}$
- problem: one could choose x_i
- solution: additional vertex y_i → choosing x_i and y_i is too expensive
- make it sufficient to choose one of V_i: clique on V_i



Forbid to choose u and v at the same time

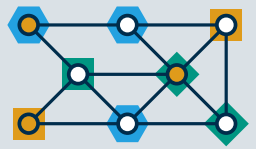
- idea: add vertex z that is covered unless u and v are chosen
- connect z with all vertices of u's and v's color, except u and v

Dominating Set

Reduce Multicolored Independent Set to Dominating Set

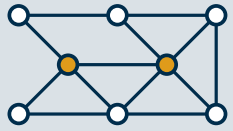
- enforce the following properties in an instance of Dominating Set
  - exactly one vertex of each color must be chosen ✓
  - forbid the choice of certain vertex pairs ✓

MC Independent Set



$V' \subseteq V$ colorful, such that
$\forall e \in E : |e \cap V'| \leq 1$

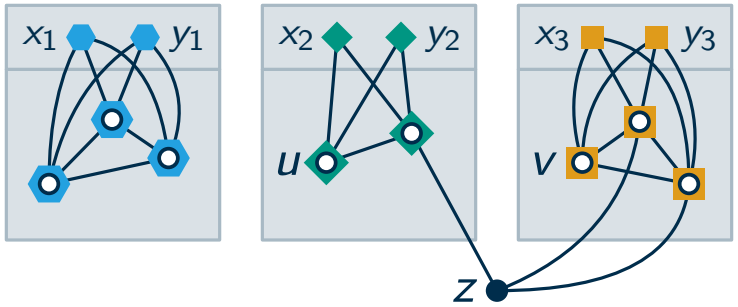
Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

Exactly one vertex for each color

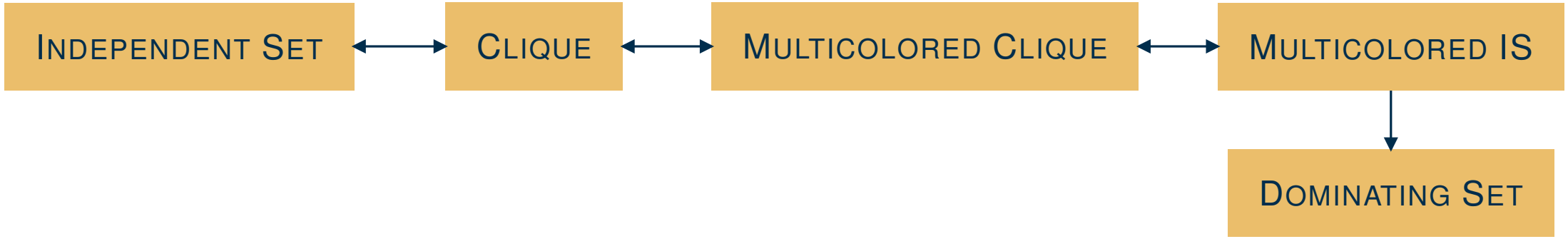
- vertex x_i with edges to all vertices in V_i → must choose one of $V_i \cup \{x_i\}$
- problem: one could choose x_i
- solution: additional vertex y_i → choosing x_i and y_i is too expensive
- make it sufficient to choose one of V_i: clique on V_i



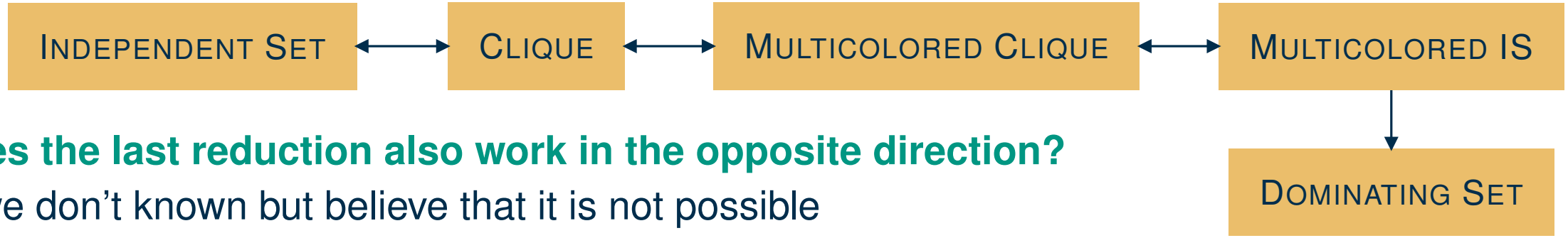
Forbid to choose u and v at the same time

- idea: add vertex z that is covered unless u and v are chosen
- connect z with all vertices of u's and v's color, except u and v

Reductions so far



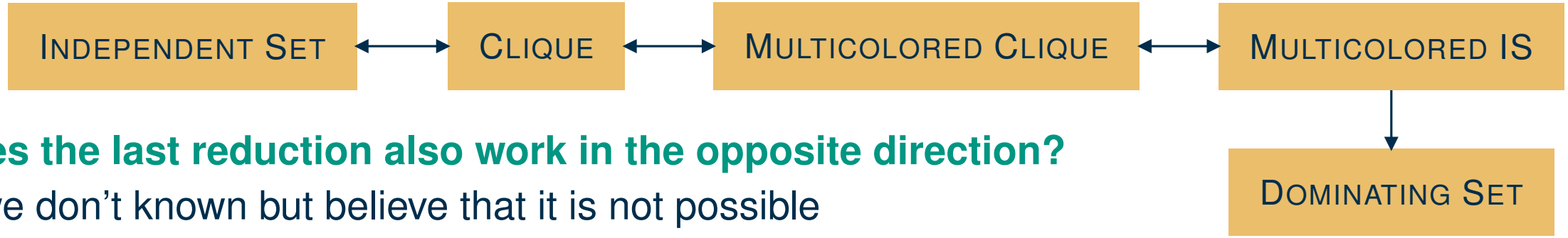
Reductions so far



Does the last reduction also work in the opposite direction?

- we don't known but believe that it is not possible

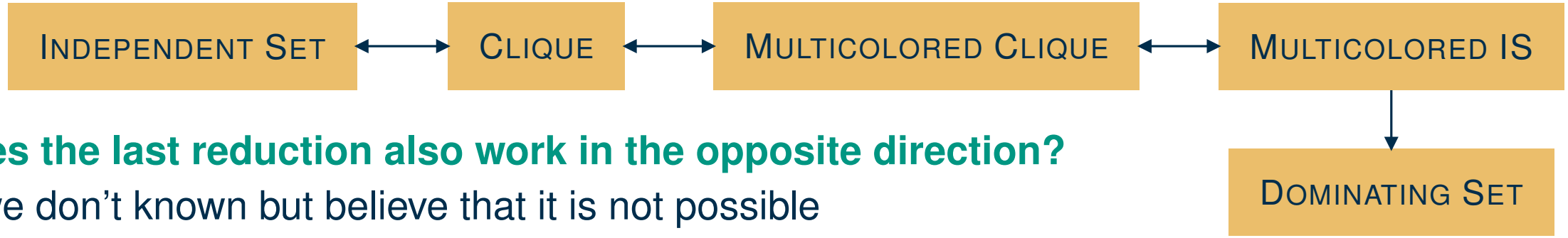
Reductions so far



Does the last reduction also work in the opposite direction?

- we don't known but believe that it is not possible
- DS seems strictly harder than CLIQUE or IS (DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way round)

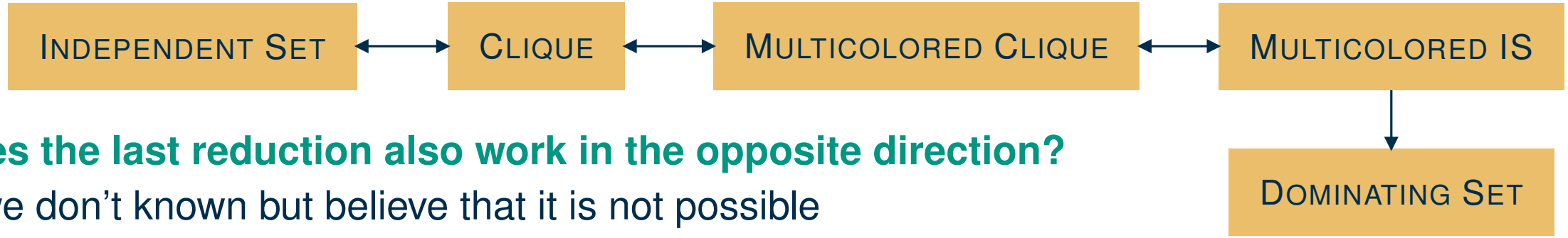
Reductions so far



Does the last reduction also work in the opposite direction?

- we don't known but believe that it is not possible
- DS seems strictly harder than CLIQUE or IS (DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way round)
- we need a more refined notion of hardness

Reductions so far

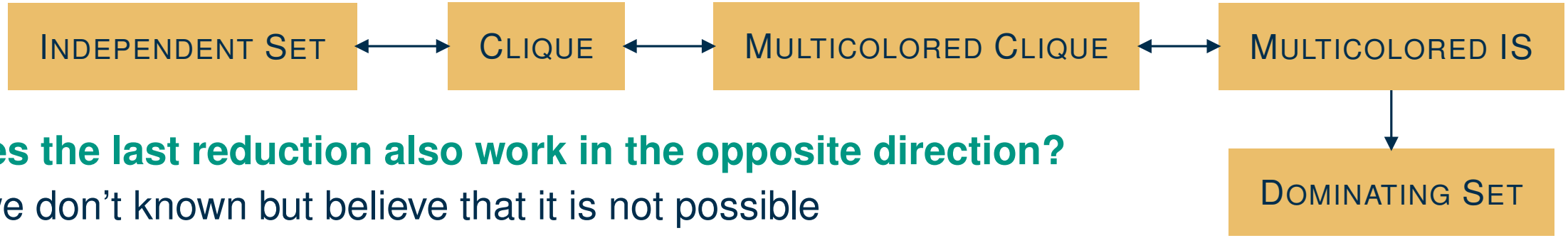


Does the last reduction also work in the opposite direction?

- we don't known but believe that it is not possible
- DS seems strictly harder than CLIQUE or IS (DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way round)
- we need a more refined notion of hardness

Comparison with complexity class P

Reductions so far



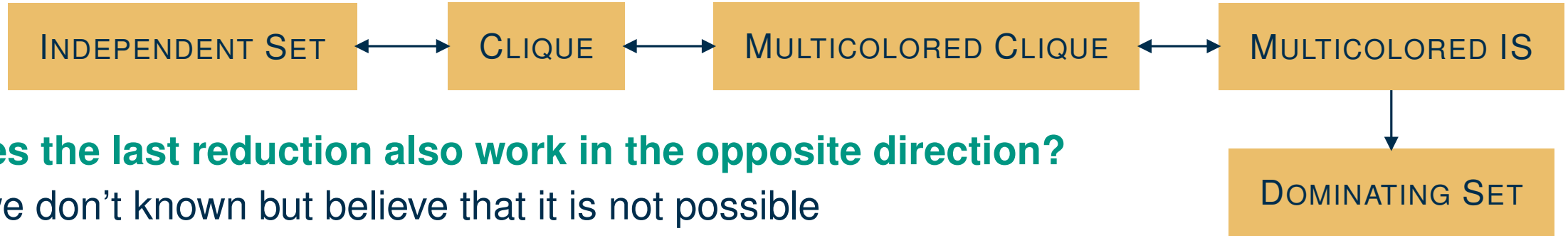
Does the last reduction also work in the opposite direction?

- we don't known but believe that it is not possible
- DS seems strictly harder than CLIQUE or IS (DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way round)
- we need a more refined notion of hardness

Comparison with complexity class P

- here usually one notion of hardness suffices: NP-hardness
- there are intermediate problems (assuming P $\neq$ NP), but we don't know any natural cases
  (candidates: prime factorization, graph isomorphism)

Reductions so far



Does the last reduction also work in the opposite direction?

- we don't known but believe that it is not possible
- DS seems strictly harder than CLIQUE or IS (DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way round)
- we need a more refined notion of hardness

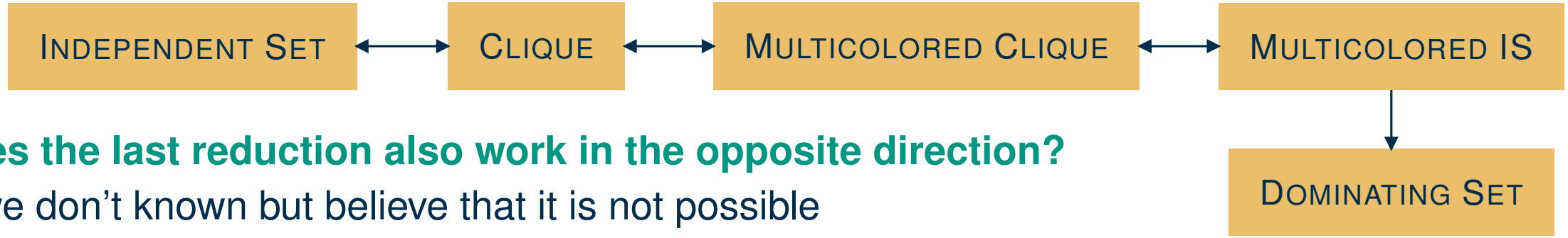
Comparison with complexity class P

- here usually one notion of hardness suffices: NP-hardness
- there are intermediate problems (assuming P $\neq$ NP), but we don't know any natural cases
  (candidates: prime factorization, graph isomorphism)

What now?

- define hierarchy of complexity classes

Reductions so far



Does the last reduction also work in the opposite direction?

- we don't known but believe that it is not possible
- DS seems strictly harder than CLIQUE or IS (DS $\in$ FPT $\Rightarrow$ IS $\in$ FPT, but not the other way round)
- we need a more refined notion of hardness

Comparison with complexity class P

- here usually one notion of hardness suffices: NP-hardness
- there are intermediate problems (assuming P $\neq$ NP), but we don't know any natural cases
  (candidates: prime factorization, graph isomorphism)

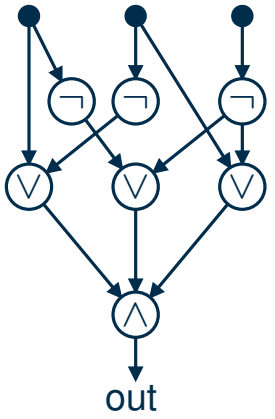
What now?

- define hierarchy of complexity classes
- use prototypical complete problems for each level of the hierarchy

Weighted Circuit Satisfiability

Boolean circuit

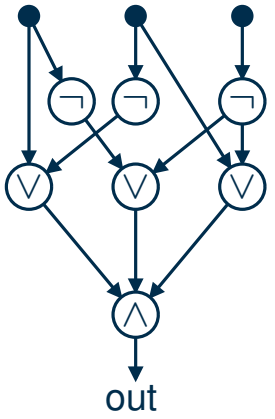
- directed acyclic graph (DAG) with the following types of nodes:



Weighted Circuit Satisfiability

Boolean circuit

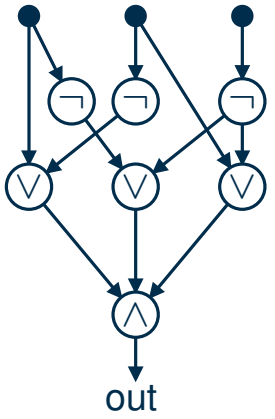
- directed acyclic graph (DAG) with the following types of nodes:
  - sources (in-degree 0) are **input nodes**



Weighted Circuit Satisfiability

Boolean circuit

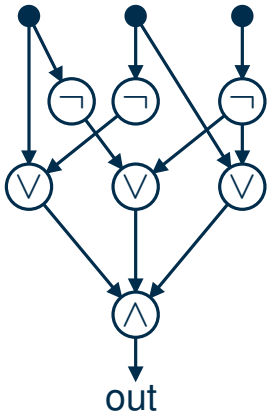
- directed acyclic graph (DAG) with the following types of nodes:
  - sources (in-degree 0) are **input nodes** •
  - **negation nodes** have in-degree 1 ¬



Weighted Circuit Satisfiability

Boolean circuit

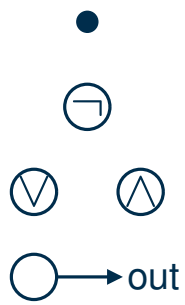
- directed acyclic graph (DAG) with the following types of nodes:
  - sources (in-degree 0) are **input nodes** •
  - **negation nodes** have in-degree 1 ¬
  - **and nodes** and **or nodes** have in-degree ≥ 2 ∨ ∧

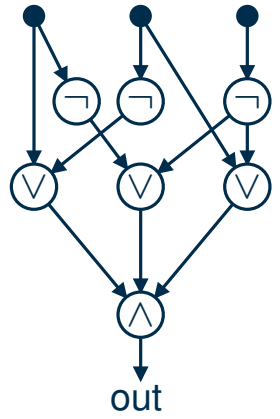


Weighted Circuit Satisfiability

Boolean circuit

- directed acyclic graph (DAG) with the following types of nodes:
  - sources (in-degree 0) are **input nodes**
  - **negation nodes** have in-degree 1
  - **and nodes** and **or nodes** have in-degree ≥ 2
  - one sink (out-degree 0) is the **output node**

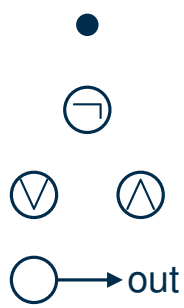




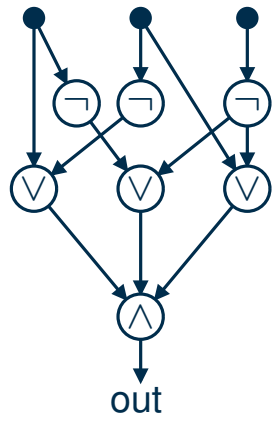
Weighted Circuit Satisfiability

Boolean circuit

- directed acyclic graph (DAG) with the following types of nodes:
  - sources (in-degree 0) are **input nodes**
  - **negation nodes** have in-degree 1
  - **and nodes** and **or nodes** have in-degree ≥ 2
  - one sink (out-degree 0) is the **output node**



- a assignment of the input nodes to 0 or 1 yields values for all other nodes (in the obvious way)
- an assignment is **satisfying** if the output node has value 1

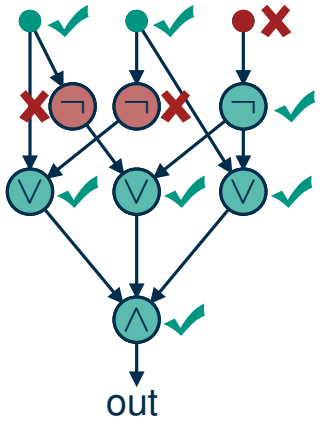


Weighted Circuit Satisfiability

Boolean circuit

- directed acyclic graph (DAG) with the following types of nodes:
  - sources (in-degree 0) are **input nodes**
  - **negation nodes** have in-degree 1
  - **and nodes** and **or nodes** have in-degree ≥ 2
  - one sink (out-degree 0) is the **output node**

- a assignment of the input nodes to 0 or 1 yields values for all other nodes (in the obvious way)
- an assignment is **satisfying** if the output node has value 1

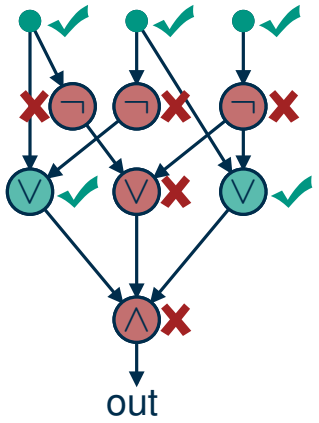


Weighted Circuit Satisfiability

Boolean circuit

- directed acyclic graph (DAG) with the following types of nodes:
  - sources (in-degree 0) are **input nodes**
  - **negation nodes** have in-degree 1
  - **and nodes** and **or nodes** have in-degree ≥ 2
  - one sink (out-degree 0) is the **output node**

- a assignment of the input nodes to 0 or 1 yields values for all other nodes (in the obvious way)
- an assignment is **satisfying** if the output node has value 1

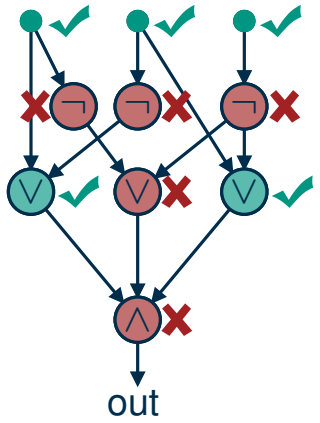


Weighted Circuit Satisfiability

Boolean circuit

- directed acyclic graph (DAG) with the following types of nodes:
  - sources (in-degree 0) are **input nodes**
  - **negation nodes** have in-degree 1
  - **and nodes** and **or nodes** have in-degree ≥ 2
  - one sink (out-degree 0) is the **output node**

- a assignment of the input nodes to 0 or 1 yields values for all other nodes (in the obvious way)
- an assignment is **satisfying** if the output node has value 1
- **weight** of an assignment $=$ number of 1s for input nodes

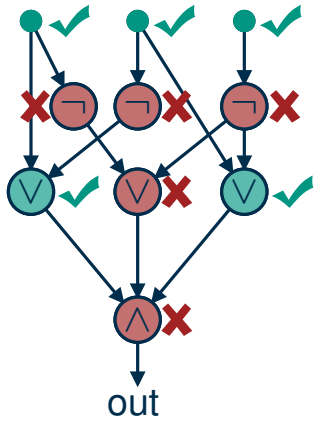


Weighted Circuit Satisfiability

Boolean circuit

- directed acyclic graph (DAG) with the following types of nodes:
  - sources (in-degree 0) are **input nodes**
  - **negation nodes** have in-degree 1
  - **and nodes** and **or nodes** have in-degree ≥ 2
  - one sink (out-degree 0) is the **output node**

- a assignment of the input nodes to 0 or 1 yields values for all other nodes (in the obvious way)
- an assignment is **satisfying** if the output node has value 1
- **weight** of an assignment $=$ number of 1s for input nodes



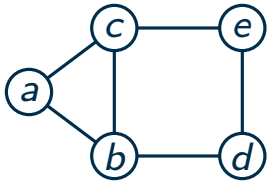
Problem: Weighted Circuit Satisfiability (WCS)
Given a Boolean circuit and a parameter k, is there a satisfying assignment of weight exactly k?

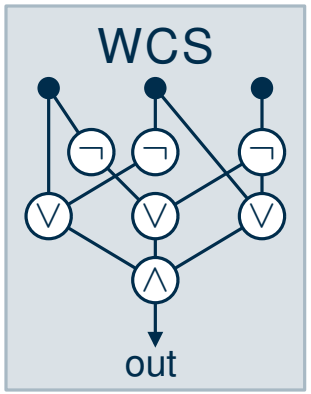
Reductions as far as the eye can see

Independent Set → WCS

Independent Set

$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$



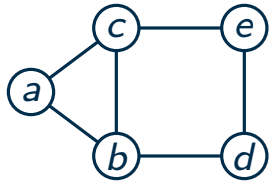


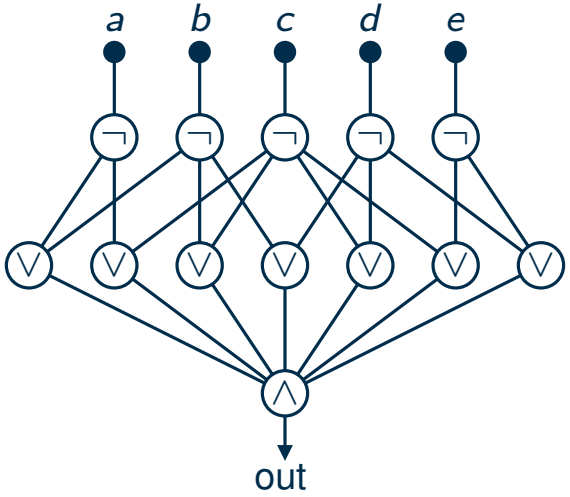
Reductions as far as the eye can see

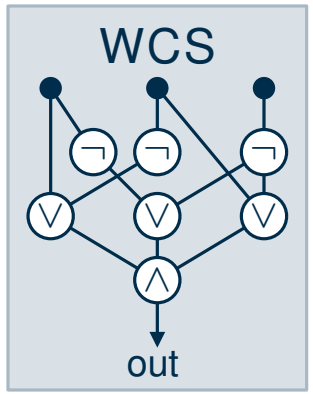
Independent Set → WCS

Independent Set

$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$





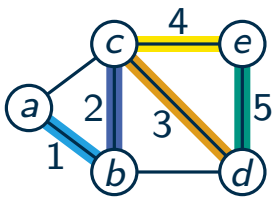


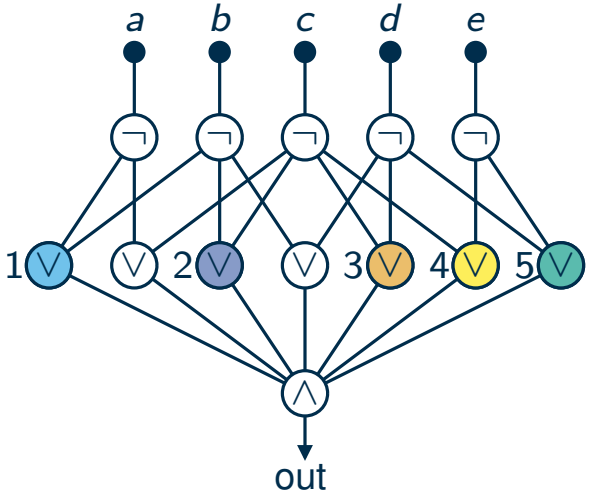
Reductions as far as the eye can see

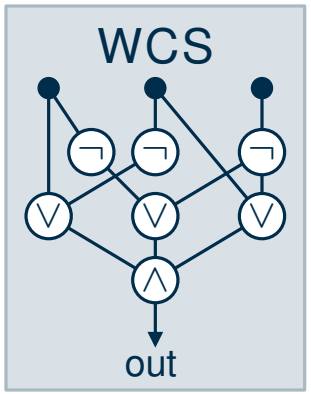
Independent Set → WCS

Independent Set

$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$



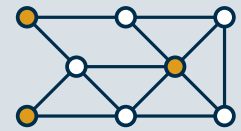




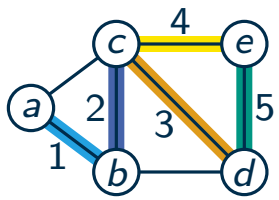
Reductions as far as the eye can see

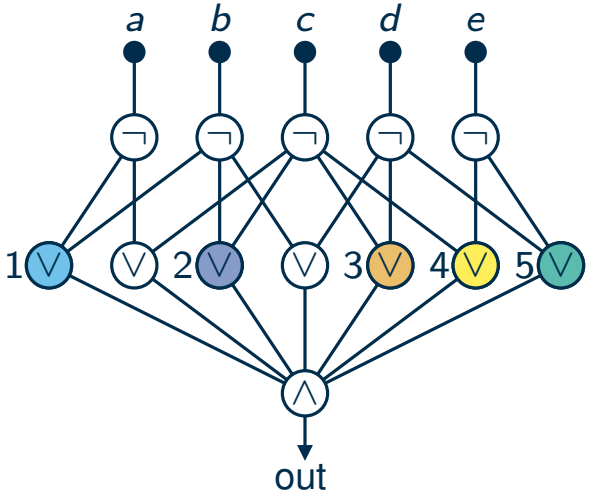
Independent Set → WCS

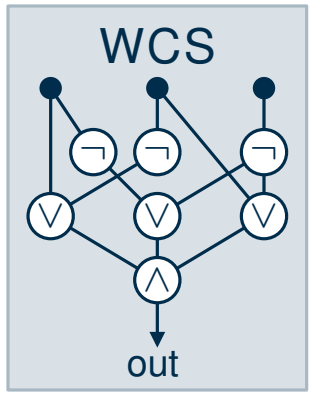
Independent Set



$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$

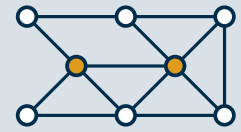




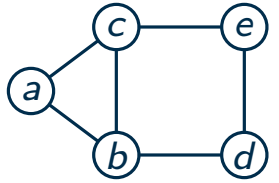


Dominating Set → WCS

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

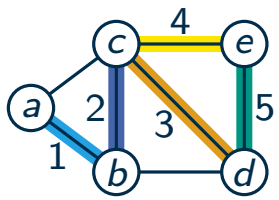


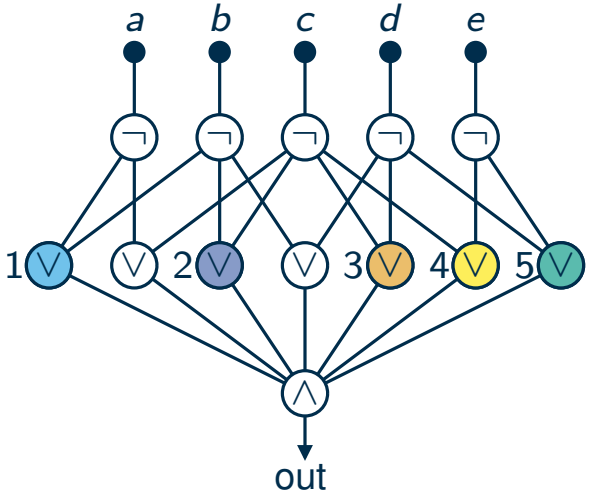
Reductions as far as the eye can see

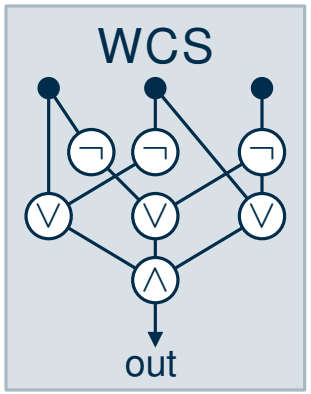
Independent Set → WCS

Independent Set

$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$

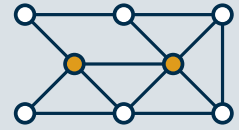




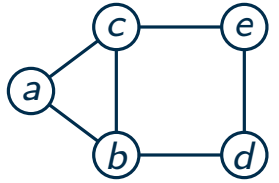


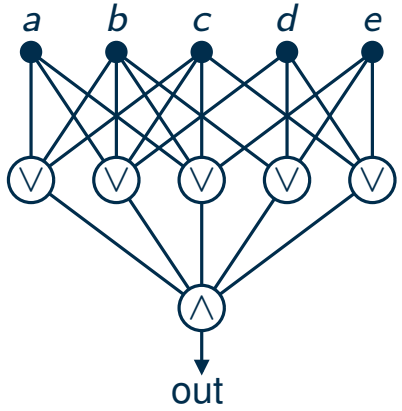
Dominating Set → WCS

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$



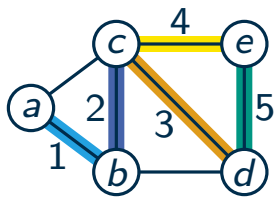


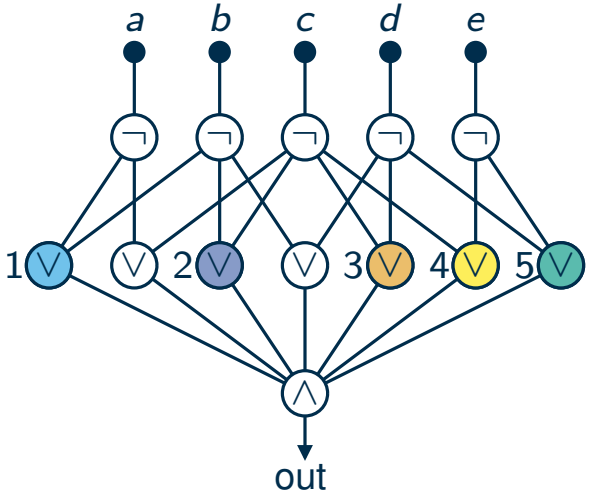
Reductions as far as the eye can see

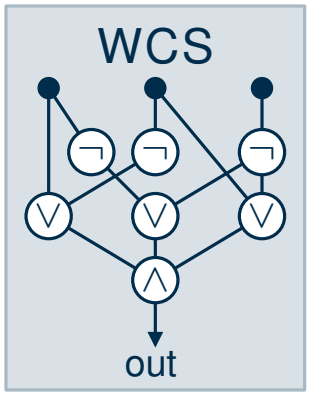
Independent Set → WCS

Independent Set

$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$

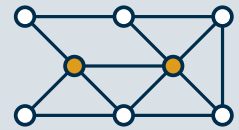




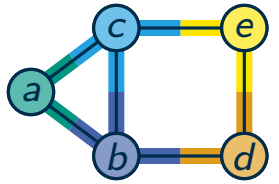


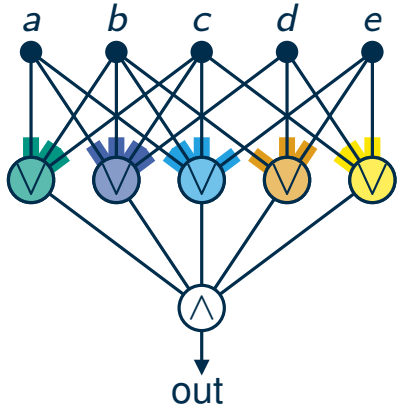
Dominating Set → WCS

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$

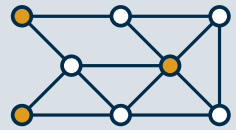




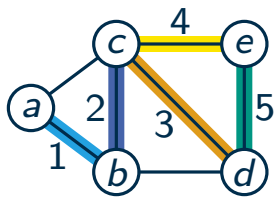
Reductions as far as the eye can see

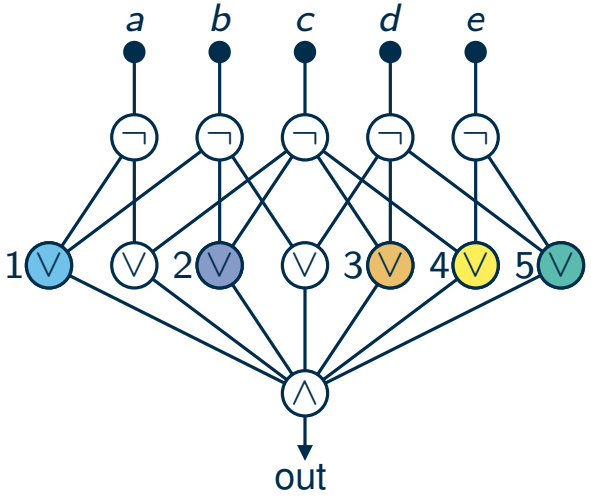
Independent Set → WCS

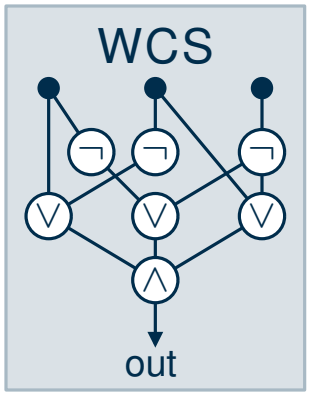
Independent Set



$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$

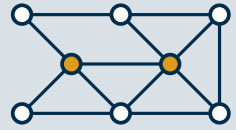




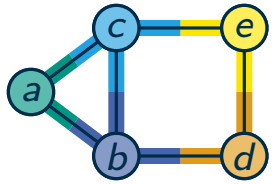


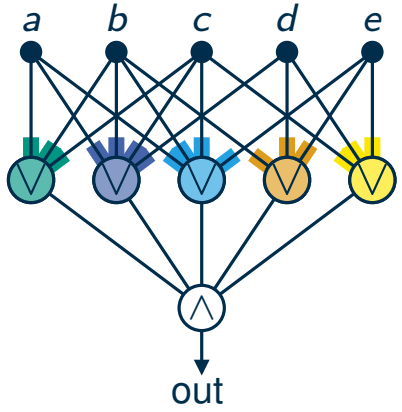
Dominating Set → WCS

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$





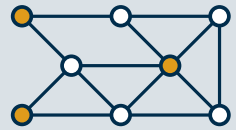
Observations

- both circuits: constant height

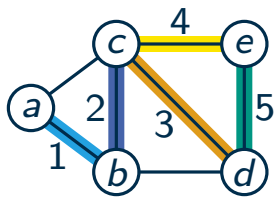
Reductions as far as the eye can see

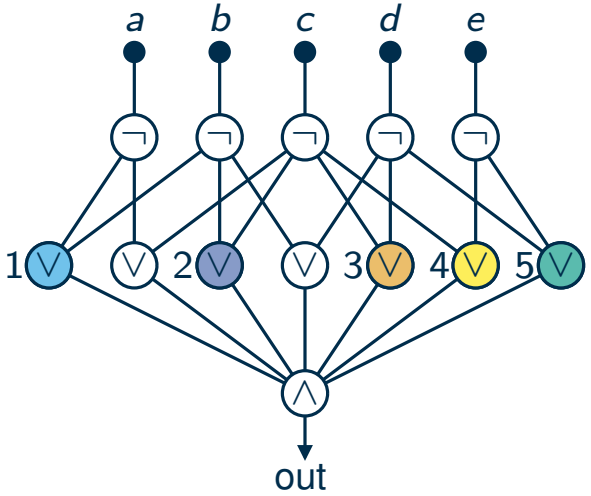
Independent Set → WCS

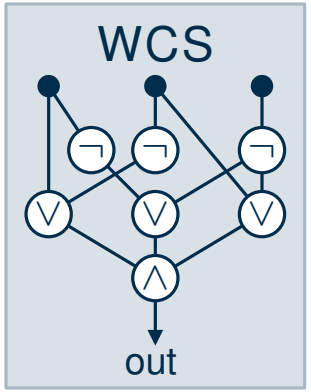
Independent Set



$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$

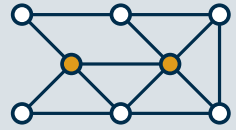




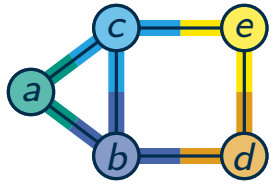


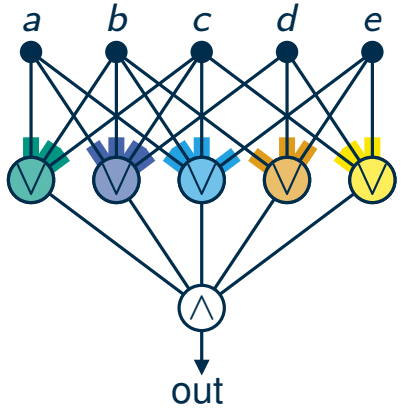
Dominating Set → WCS

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$





Observations

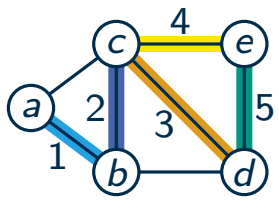
- both circuits: constant height
- the DS circuit has more nodes with in-degree > 2

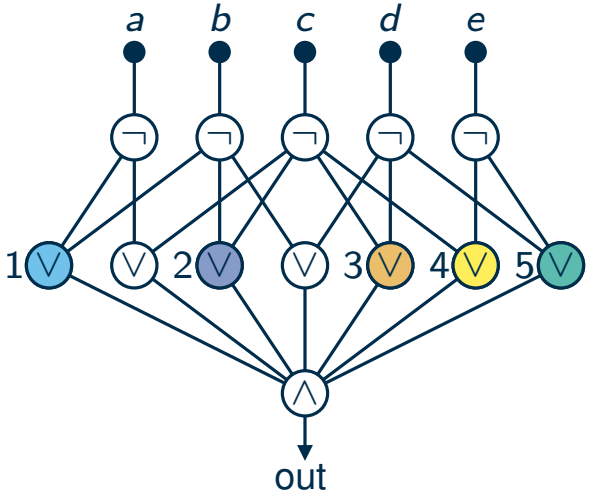
Reductions as far as the eye can see

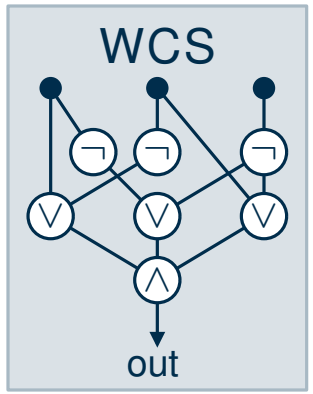
Independent Set → WCS

Independent Set

$V' \subseteq V$, such that
$\forall e \in E : |e \cap V'| \leq 1$

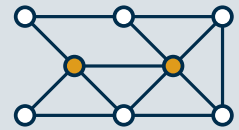




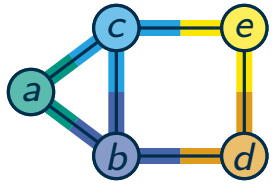


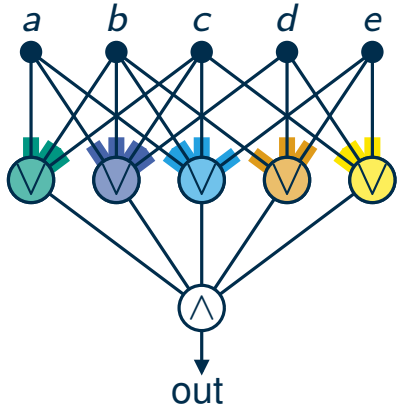
Dominating Set → WCS

Dominating Set



$V' \subseteq V$, such that
$\forall v \in V : |N[v] \cap V'| \geq 1$



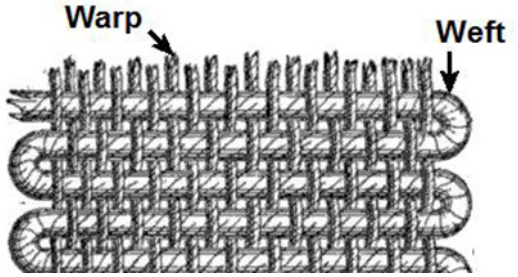


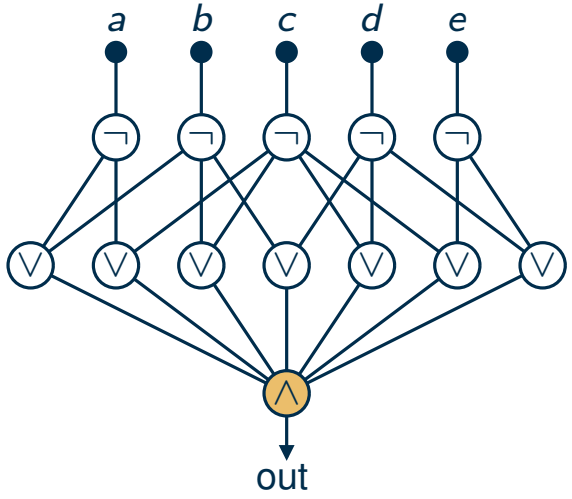
Observations

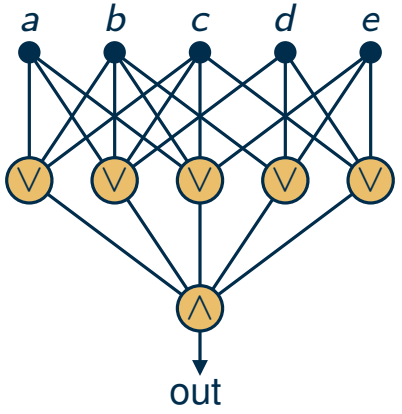
- both circuits: constant height
- the DS circuit has more nodes with in-degree > 2
- Does this make DS harder (with respect to FPT) than IS?

Weft

Definition
The largest number of nodes with in-degree > 2 on a directed path is called **weft** of the circuit.

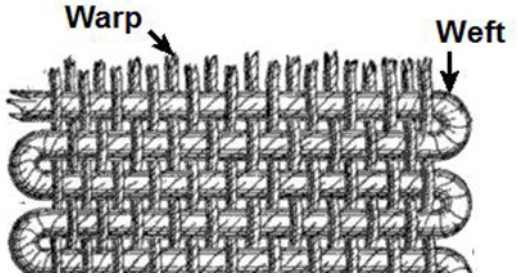


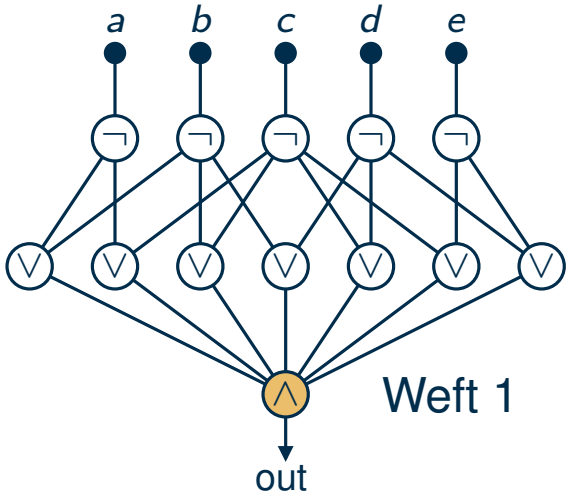


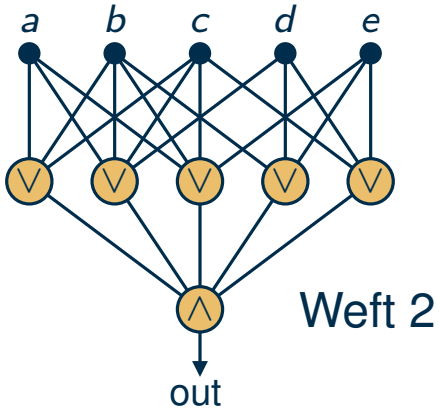


Weft

Definition
The largest number of nodes with in-degree > 2 on a directed path is called **weft** of the circuit.



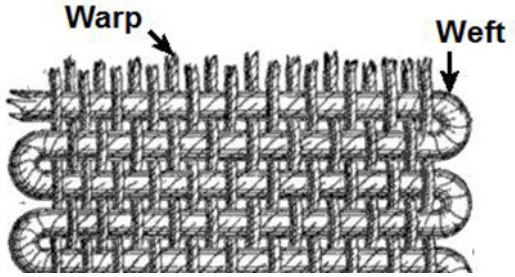


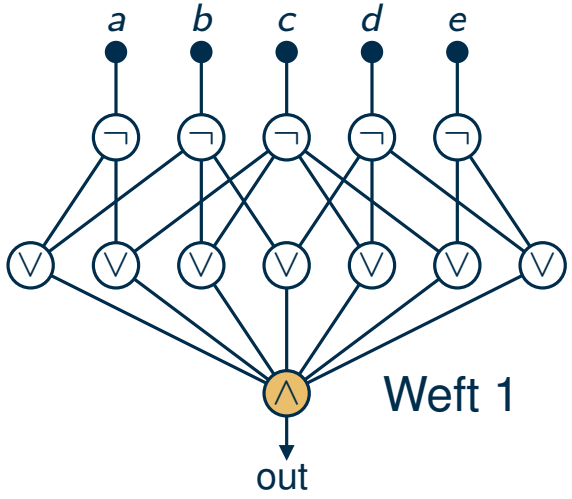


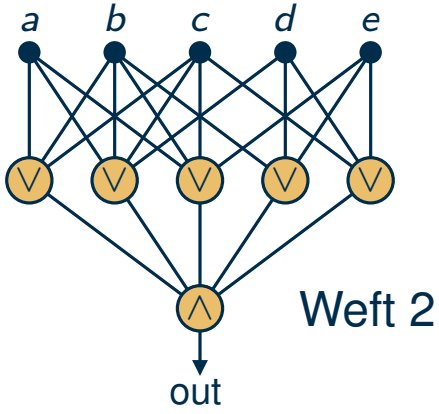
Weft

Definition
The largest number of nodes with in-degree > 2 on a directed path is called **weft** of the circuit.

Problem
WCS[t] restricts WCS to circuits of constant height and weft $\leq t$.



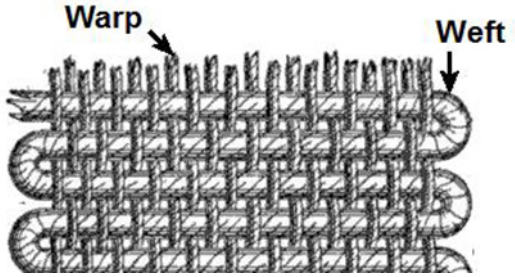




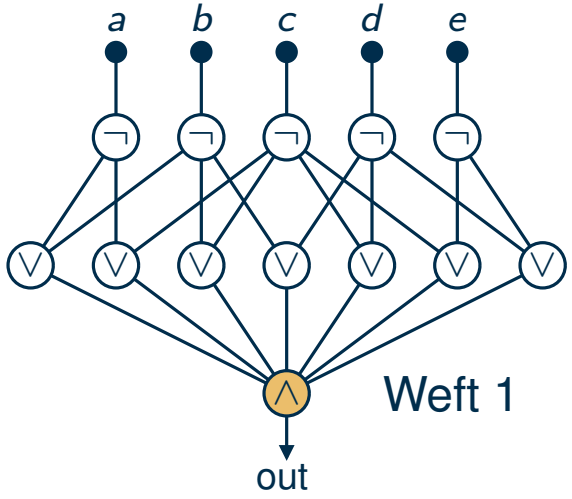
Weft

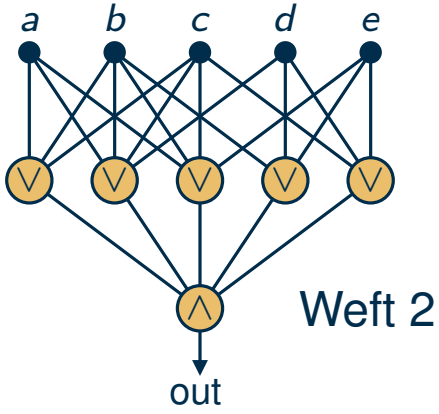
Definition
The largest number of nodes with in-degree > 2 on a directed path is called **weft** of the circuit.

Problem
WCS[t] restricts WCS to circuits of constant height and weft $\leq t$.



Definition
The class $\boldsymbol{W[t]}$ is the set of problems that have a parameterized reduction to WCS[t].

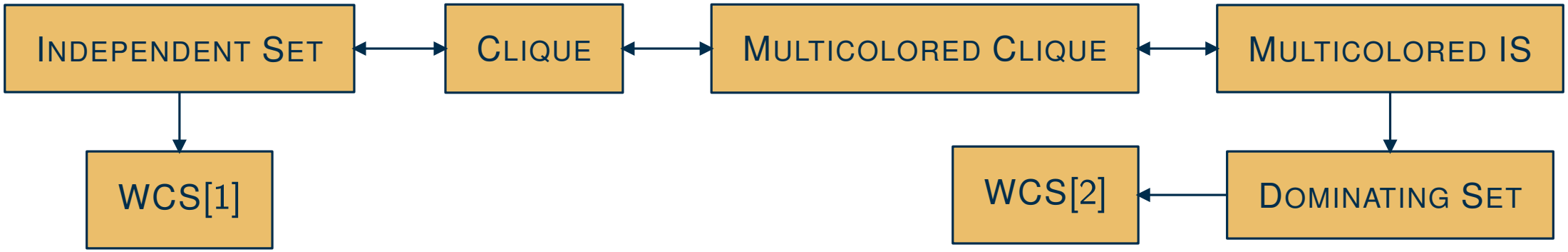




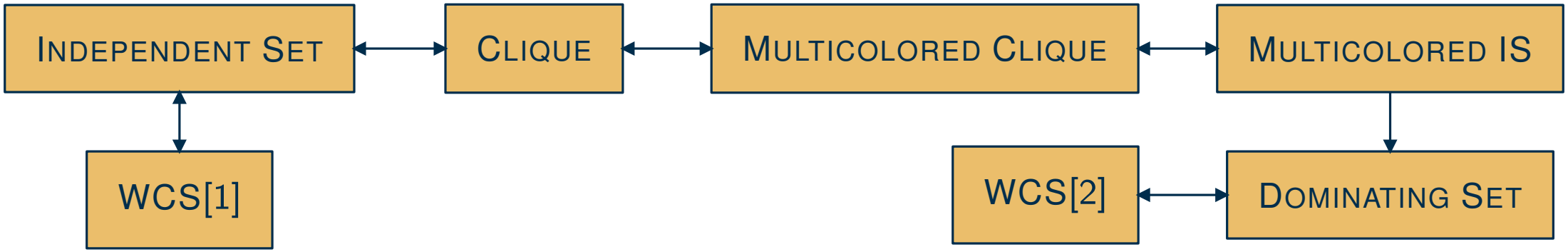
Just seen

- INDEPENDENT SET $\in W[1]$
- DOMINATING SET $\in W[2]$

Bisherige FPT-Reduktionen



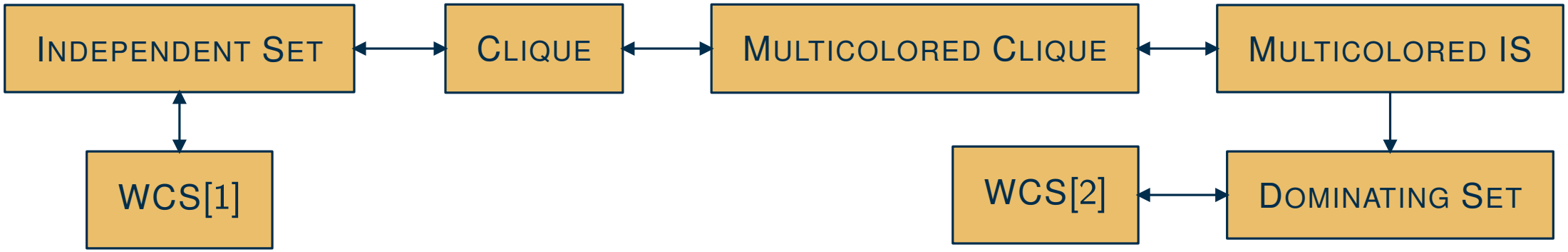
Bisherige FPT-Reduktionen



Additional reductions

- WCS[1] $\rightarrow$ IS and WCS[2] to DS

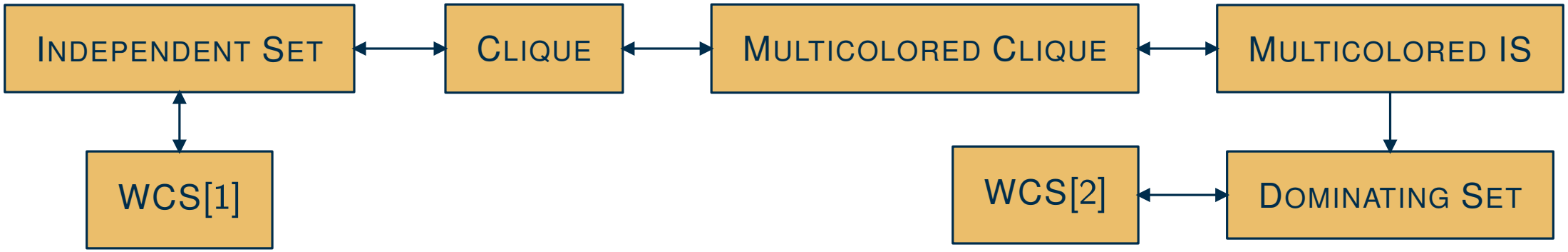
Bisherige FPT-Reduktionen



Additional reductions

- WCS[1] $\rightarrow$ IS and WCS[2] to DS
- every $W[1]$-problem reduces to IS $\rightarrow$ $W[1]$-complete

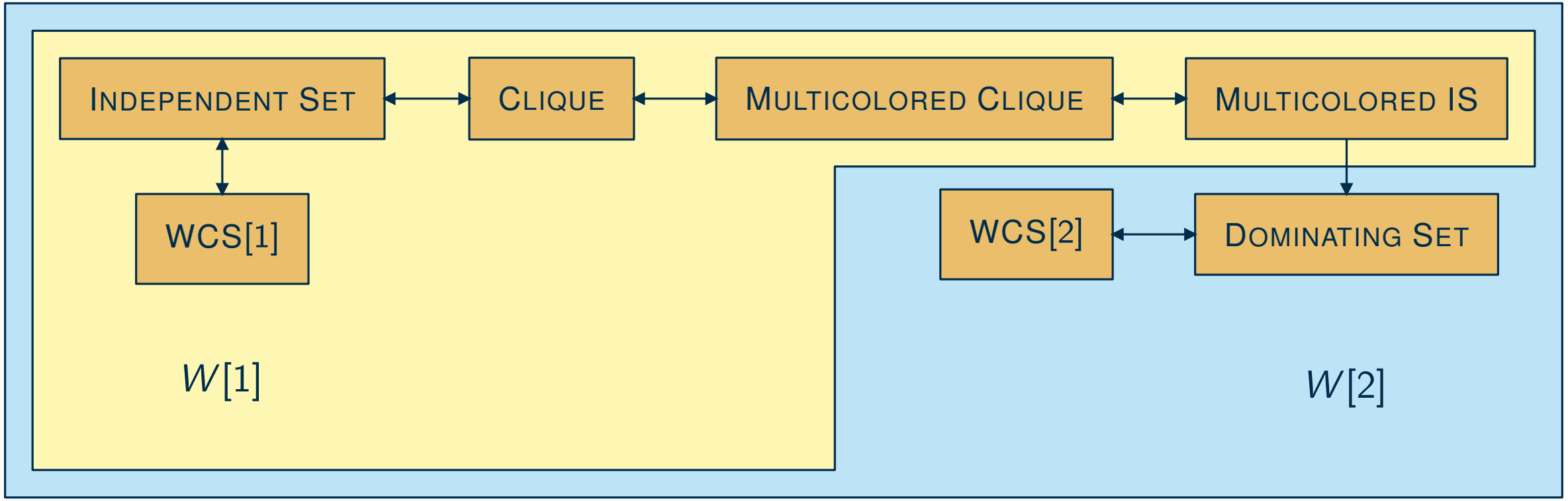
Bisherige FPT-Reduktionen



Additional reductions

- WCS[1] $\rightarrow$ IS and WCS[2] to DS
- every $W[1]$-problem reduces to IS $\rightarrow$ $W[1]$-complete
- analogously: DS is $W[2]$-complete

Bisherige FPT-Reduktionen



Additional reductions

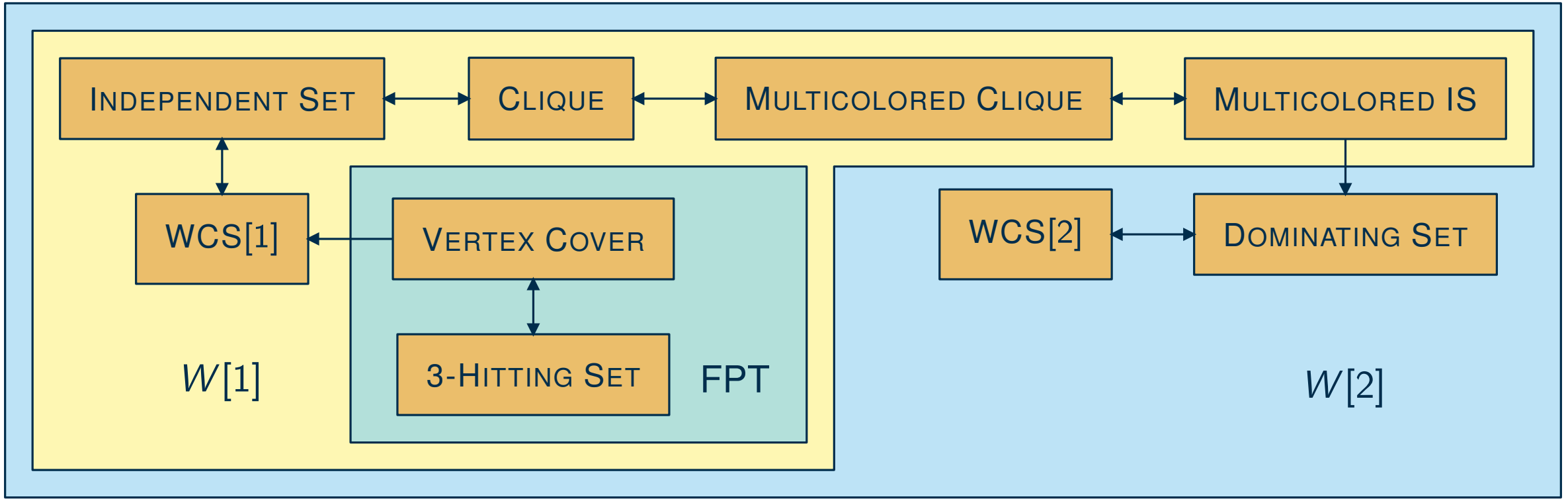
- WCS[1] $\rightarrow$ IS and WCS[2] to DS
- every $W[1]$-problem reduces to IS $\rightarrow$ $W[1]$-complete
- analogously: DS is $W[2]$-complete

Relations between classes

- $W[1] \subseteq W[2]$

Why?

Bisherige FPT-Reduktionen



Additional reductions

- WCS[1] $\rightarrow$ IS and WCS[2] to DS
- every $W[1]$-problem reduces to IS $\rightarrow$ $W[1]$-complete
- analogously: DS is $W[2]$-complete

Relations between classes

- $W[1] \subseteq W[2]$
- FPT $\subseteq W[1] \subseteq W[2]$

Why?

Why?

Wrap-Up

The W-hierarchy

- complexity classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via prototypical complete problem WCS$[t]$: $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ reducible to WCS$[t]$

(via parameterized reductions)

Wrap-Up

The W-hierarchy

- complexity classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via prototypical complete problem WCS$[t]$: $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ reducible to WCS$[t]$ (via parameterized reductions)
- we believe: each inclusion is strict

Wrap-Up

The W-hierarchy

- complexity classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via prototypical complete problem WCS$[t]$: $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ reducible to WCS$[t]$ (via parameterized reductions)
- we believe: each inclusion is strict

Is my $W[t]$-completeness proof useless, if $W[t] = \text{FPT}$?

Wrap-Up

The W-hierarchy

- complexity classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via prototypical complete problem WCS$[t]$: $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ reducible to WCS$[t]$ (via parameterized reductions)
- we believe: each inclusion is strict

Is my $W[t]$-completeness proof useless, if $W[t] = \text{FPT}$?

- still true: you don't find an FPT-algo due to a fundamental problem, not because you are stupid

Wrap-Up

The W-hierarchy

- complexity classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via prototypical complete problem WCS$[t]$: $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ reducible to WCS$[t]$ (via parameterized reductions)
- we believe: each inclusion is strict

Is my $W[t]$-completeness proof useless, if $W[t] = \text{FPT}$?

- still true: you don't find an FPT-algo due to a fundamental problem, not because you are stupid
- finding an FPT-algo for one complete problem $\rightarrow$ reductions yield additional FPT-algos

Wrap-Up

The W-hierarchy

- complexity classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via prototypical complete problem WCS$[t]$: $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ reducible to WCS$[t]$ (via parameterized reductions)
- we believe: each inclusion is strict

Is my $W[t]$-completeness proof useless, if $W[t] = \text{FPT}$?

- still true: you don't find an FPT-algo due to a fundamental problem, not because you are stupid
- finding an FPT-algo for one complete problem $\rightarrow$ reductions yield additional FPT-algos

How do I prove that my problem is hard?

- reduce from other hard problems

Wrap-Up

The W-hierarchy

- complexity classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via prototypical complete problem WCS$[t]$: $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ reducible to WCS$[t]$ (via parameterized reductions)
- we believe: each inclusion is strict

Is my $W[t]$-completeness proof useless, if $W[t] = \text{FPT}$?

- still true: you don't find an FPT-algo due to a fundamental problem, not because you are stupid
- finding an FPT-algo for one complete problem $\rightarrow$ reductions yield additional FPT-algos

How do I prove that my problem is hard?

- reduce from other hard problems
- reductions from Independent Set or Clique prove $W[1]$-hardness

Wrap-Up

The W-hierarchy

- complexity classes FPT $\subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots$
- $W[t]$ defined via prototypical complete problem WCS$[t]$: $\mathcal{L} \in W[t] \Leftrightarrow \mathcal{L}$ reducible to WCS$[t]$ (via parameterized reductions)
- we believe: each inclusion is strict

Is my $W[t]$-completeness proof useless, if $W[t] = \text{FPT}$?

- still true: you don't find an FPT-algo due to a fundamental problem, not because you are stupid
- finding an FPT-algo for one complete problem $\rightarrow$ reductions yield additional FPT-algos

How do I prove that my problem is hard?

- reduce from other hard problems
- reductions from Independent Set or Clique prove $W[1]$-hardness
- reductions from Dominating Set or Hitting Set prove $W[2]$-hardness