CS 5003: Parameterized Algorithms

Lectures 34-35

Krithika Ramaswamy

IIT Palakkad

Reference Books: Parameterized Algorithms by Cygan et al. and Kernelization by Fomin et al.

Treewidth

* A tree decomposition of a graph G is a pair (T,B) where T is a tree and B: V(T) -> $2^{V(G)}$
  * For each vertex v in G, there is a node x in V(T) such that v is in B(x)
  * For each edge e={u, v} in G, there is a node x in V(T) such that u are v are in B(x)
  * For each vertex v in G, the set $\{x \in V(T) : v \in B(x)\}$ induces a connected graph
* Width of a tree decomposition T = $w(T) = \max \{|B(x)| : x \in V(T)\} - 1$
* Treewidth of G, $tw(G) = \min \{w(T) : T$ is a tree decomposition of G$\}$
* An optimal tree decomposition of G is a tree decomposition of G of width tw(G)

Treewidth

* A simple tree decomposition (T, B) is one where there is no pair of distinct nodes x and y in T such that $B(x) \subseteq B(y)$
* Any simple tree decomposition (T, B) of G satisfies |V(T)| <= |V(G)|
* For any G, there is an opt tree decomposition that is simple
* Given G, k, there exists an algorithm running in $2^{O(k^3)} n$ time that returns a tree decomp of G of width <=k (if one exists)

Properties of Treewidth

* If H is a subgraph of G, then tw(H) <= tw(G)
* If tw(G) <=k, G has a vertex of degree at most k
  * Look at a simple opt tree decomposition
  * Leaf t has a vertex v that is not in any other bag
  * v has neighbours only in B(t) and |B(t)| <= k+1
* If tw(G) <=k, G has at most nk - (k+1 choose 2) edges

thus O(nk) edges.

  * Induction on n (base: n=k+1)
  * Induction step: Let v be a vertex of deg <=k
  * tw(G-v) <= tw(G) and G-v has <= (n-1)k - (k+1 choose 2) edges
  * G has <= nk - (k+1 choose 2) edges

Properties of Treewidth

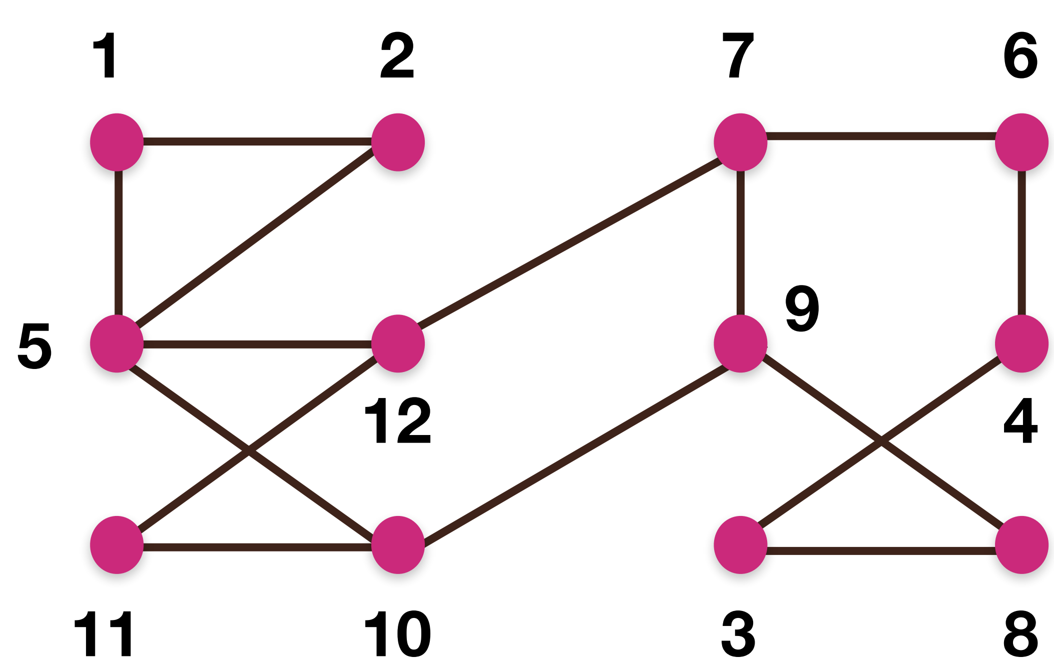
* If G is a tree, then tw(G) <= 1
    * Induction on n: base case: n=1, 2
    * Let v be a leaf in G. By induction hypothesis, tw(G-v) <= 1
    * Look at an optimal tree decomposition of G-v
    * To this, add a leaf node with bag {v,u} adjacent to a node containing u
* If G is a cycle, then tw(G)<=2
    * Let v be a vertex in G
    * G-v is a path and hence tw(G-v)<=1
    * Take an opt tree decomposition of G-v and add v to every bag
* If G is a cycle, then tw(G)>=2
    * Suppose tw(G)=1
    * Look at a simple opt tree decomposition
    * Leaf t has a vertex v that is not in any other bag
    * v has neighbours only in B(t) i.e., v has degree 1

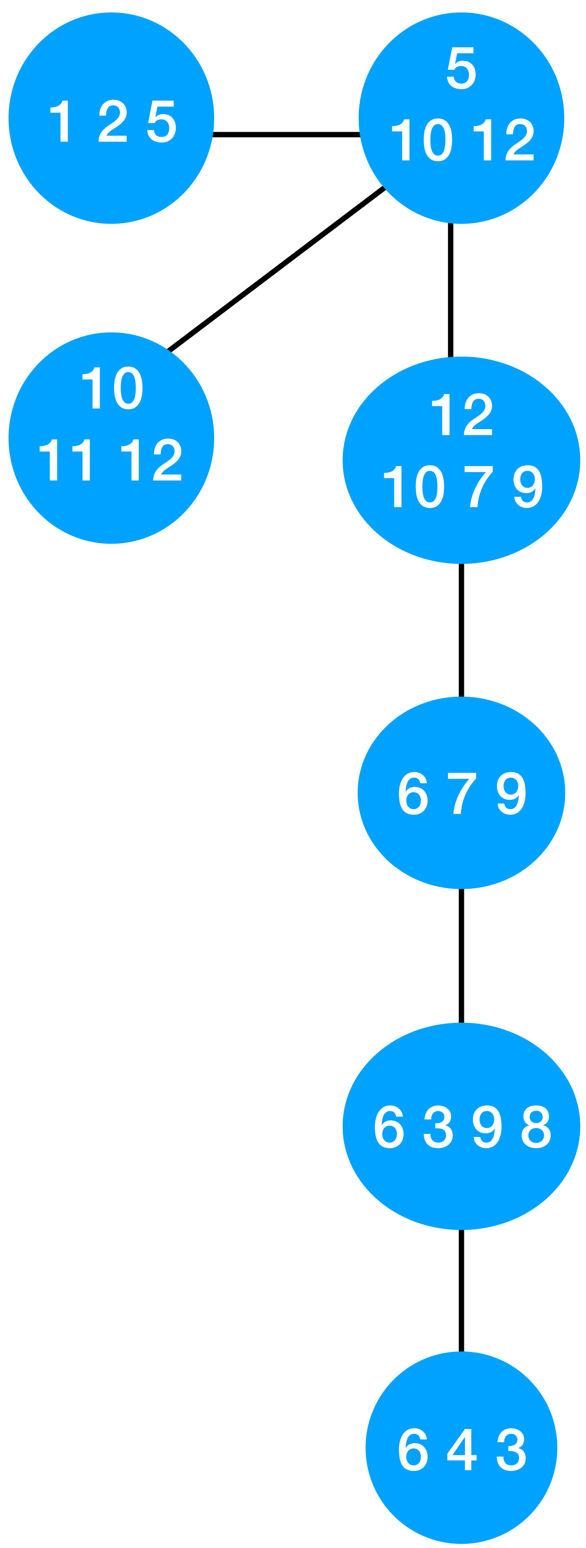
Properties of Treewidth

* A graph G on n vertices has tw <= 1 iff G is a forest
  * ($\Leftarrow$) If G is a forest, then tw(G) <= 1
  * ($\Rightarrow$) If G is a graph with tw(G) <=1, then
    * If tw=0, then G is a tree on single vertex
    * If tw=1 and G has a cycle C, then as tw(G(V(C),E(C)))=2 and tw(G(V(C),E(C))) <= tw(G), it follows that tw(G) >=2
* A graph G on n vertices has tw = n-1 iff G is a complete graph
  * Let G be a non-complete graph
    * Let u and v be non-adjacent
    * Then G has a tree decomposition consisting of 2 nodes with bags V(G)\{u} and V(G)\{v}
  * Suppose the complete graph on n vertices has tw <=n-2
    * There is a vertex v with degree at most n-2

Properties of Treewidth

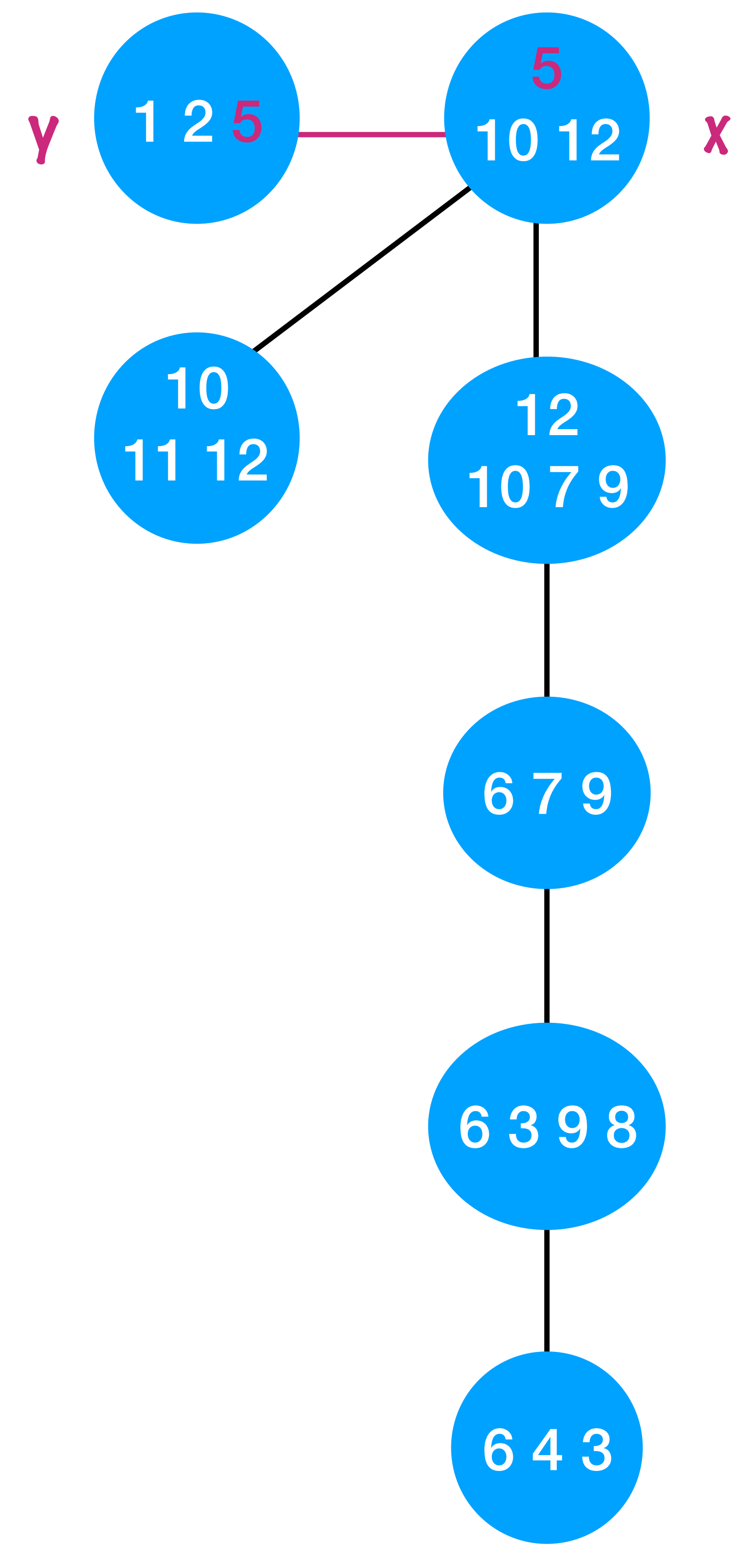
- Every bag of a tree decomposition T is a separator
  - For any two adjacent nodes x and y in T,
    - B(x) ∩ B(y) is a separator of G
    - B(x) ∩ B(y) separates $V(T \setminus T_x)$ and $V(T_x)$

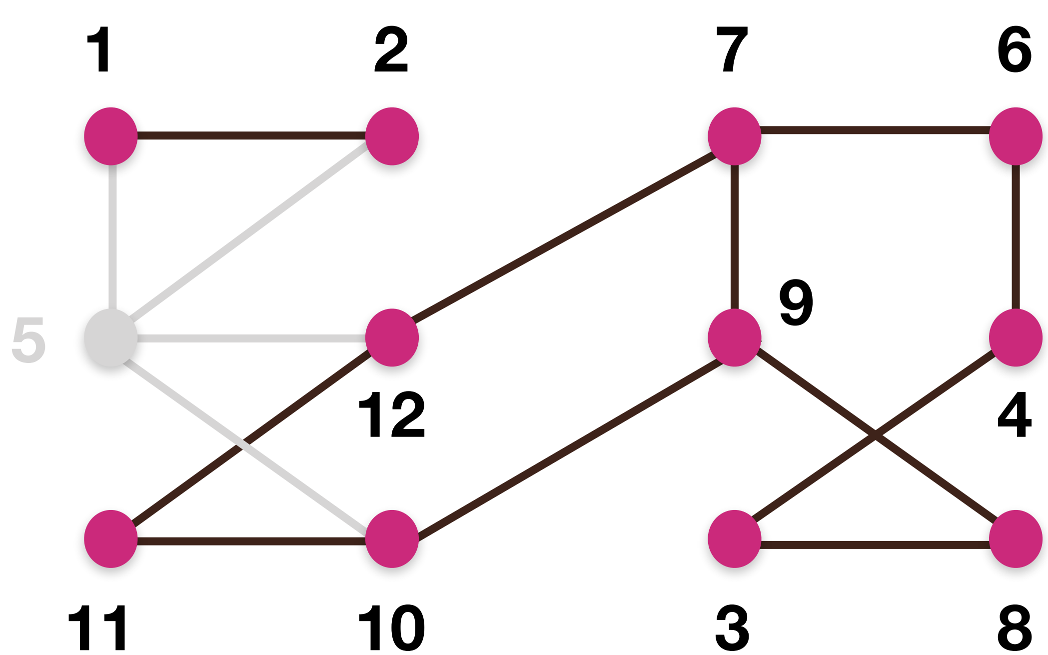




Properties of Treewidth

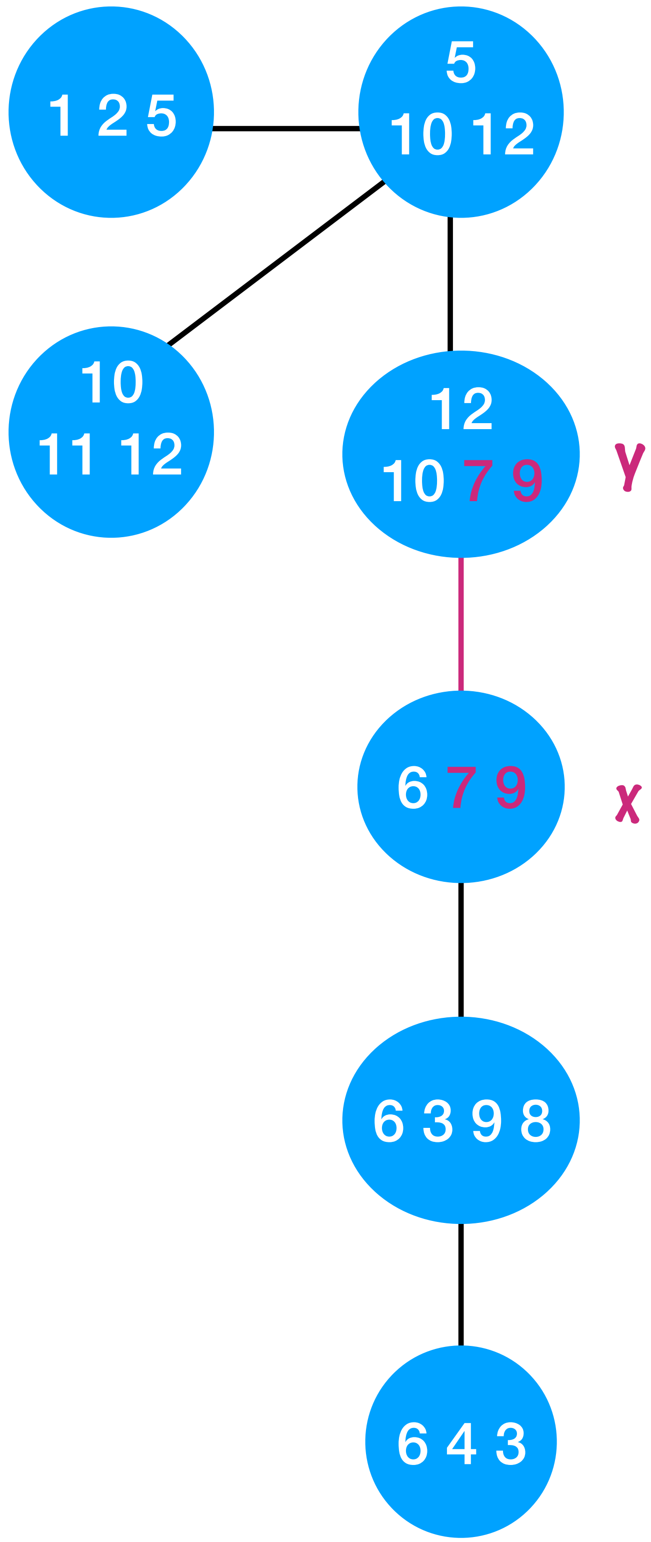
* Every bag of a tree decomposition T is a separator
    * For any two adjacent nodes x and y in T,
        * $B(x) \cap B(y)$ is a separator of G
        * $B(x) \cap B(y)$ separates $V(T \setminus T_x)$ and $V(T_x)$

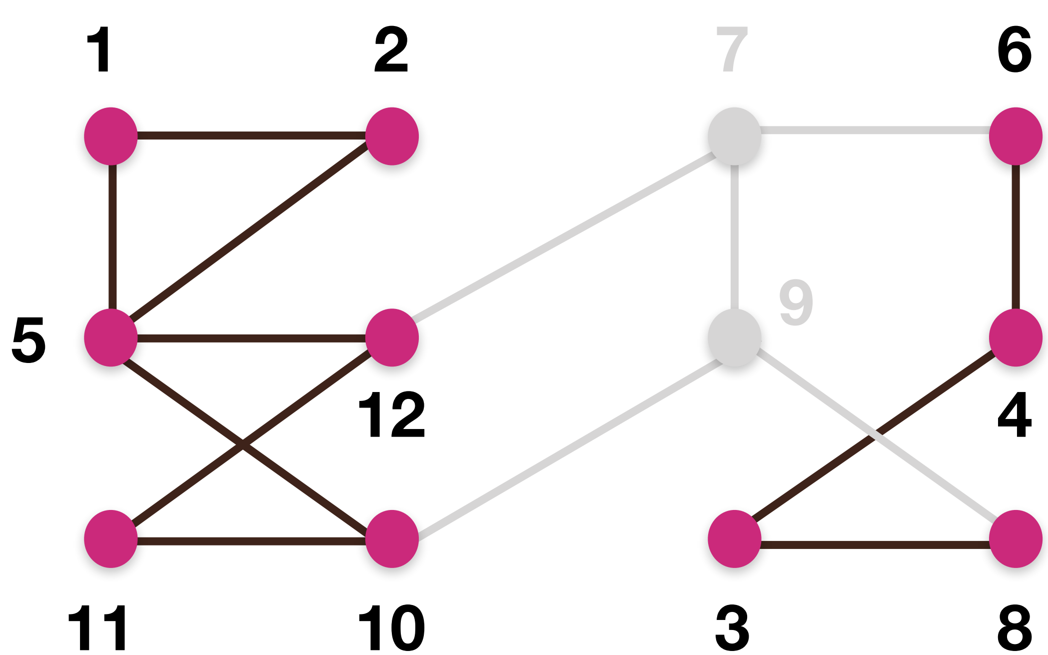




Properties of Treewidth

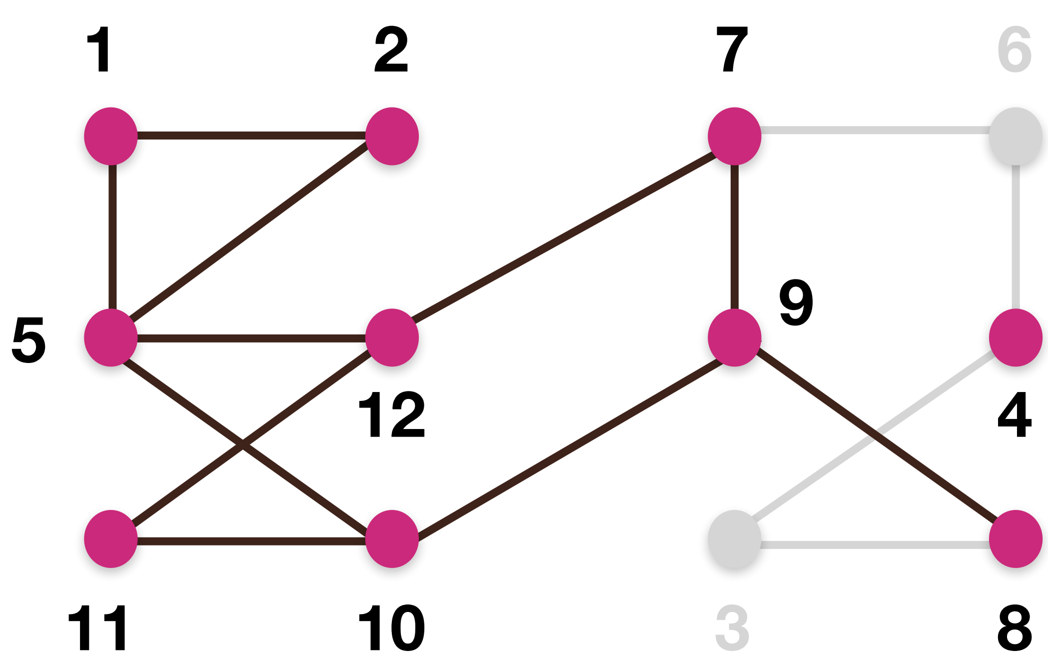
* Every bag of a tree decomposition T is a separator
  * For any two adjacent nodes x and y in T,
    * B(x) ∩ B(y) is a separator of G
    * B(x) ∩ B(y) separates $V(T \setminus T_x)$ and $V(T_x)$

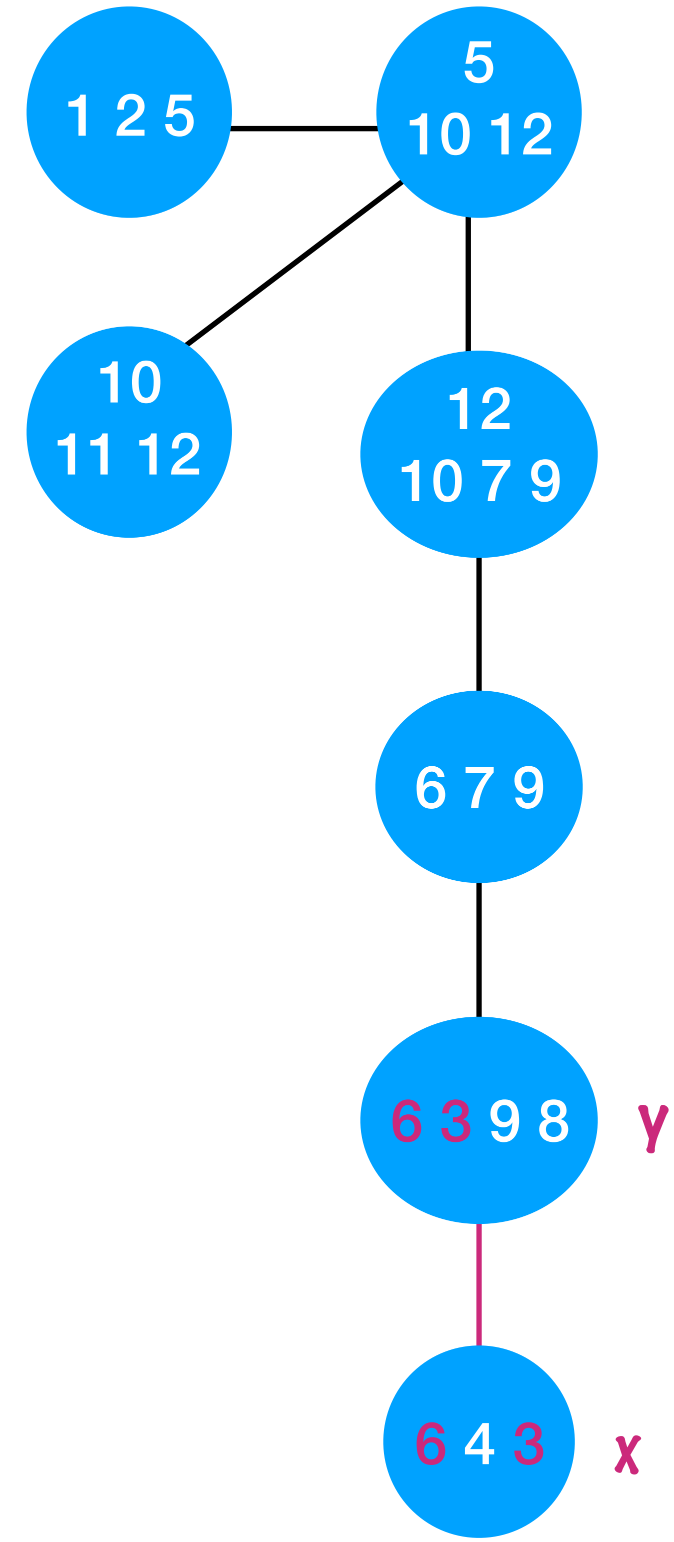




Properties of Treewidth

* Every bag of a tree decomposition T is a separator
    * For any two adjacent nodes x and y in T,
        * B(x) ∩ B(y) is a separator of G
        * B(x) ∩ B(y) separates $V(T \setminus T_x)$ and $V(T_x)$



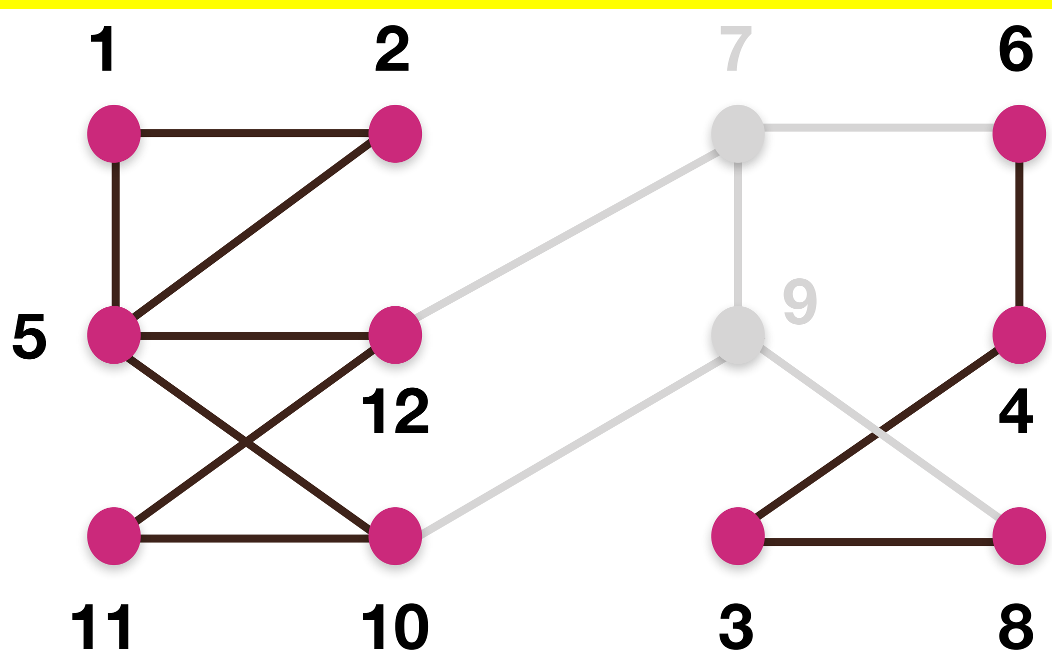


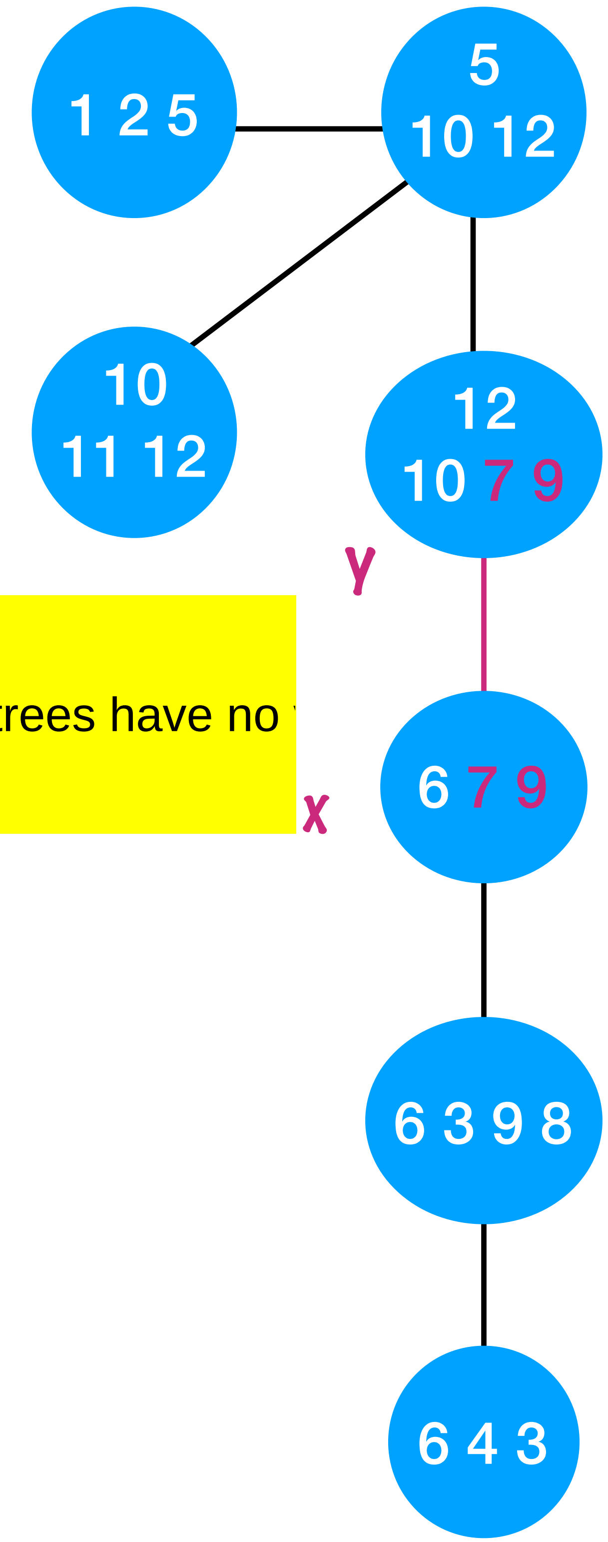
Properties of Treewidth

For any two adjacent nodes x, y in T, B(x) ∩ B(y) separates V(T\T$_x$) & V(T$_x$)

* Consider a path P between a in V(T\T$_x$) and b in V(T$_x$)
* P = (a, v_1, v_2, . . ., v_k, b)
    * P has a pair of vertices v_i in V(T\T$_x$) and v_{i+1} in V(T$_x$) (or)
    * P has a vertex v_i in V(T\T$_x$) ∩ V(T$_x$)
* Thus, there is i s.t v_i is in B(x) ∩ B(y)

Note that V(Tx) and the other thing doesn't include vertices of B(x) intersection B(y). So, these two trees have no



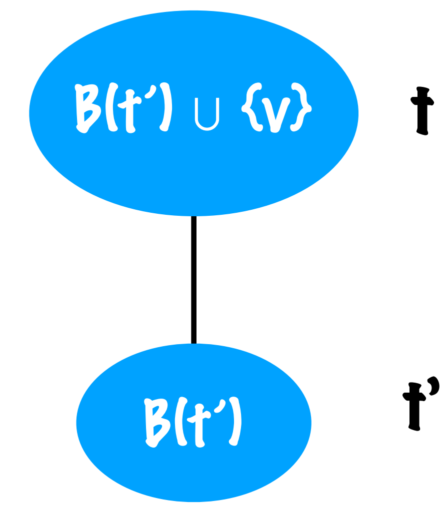


Nice Tree Decomposition

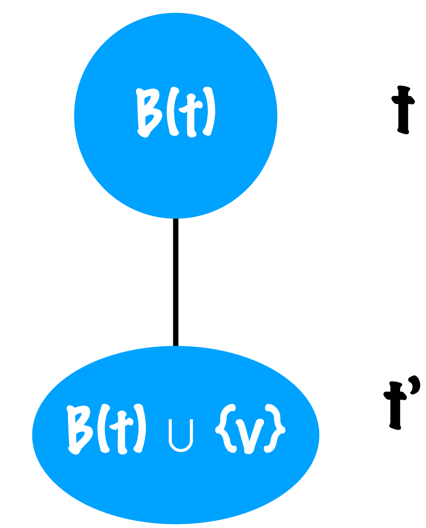
Definition: Nice Tree Decomposition (T, B)

* T is a rooted tree
* $B(root) = \emptyset$ and $B(x) = \emptyset$ for each leaf x in T
* Every non-leaf node of T is one of the following 3 types
  * Introduce node
  * Forget node
  * Join node

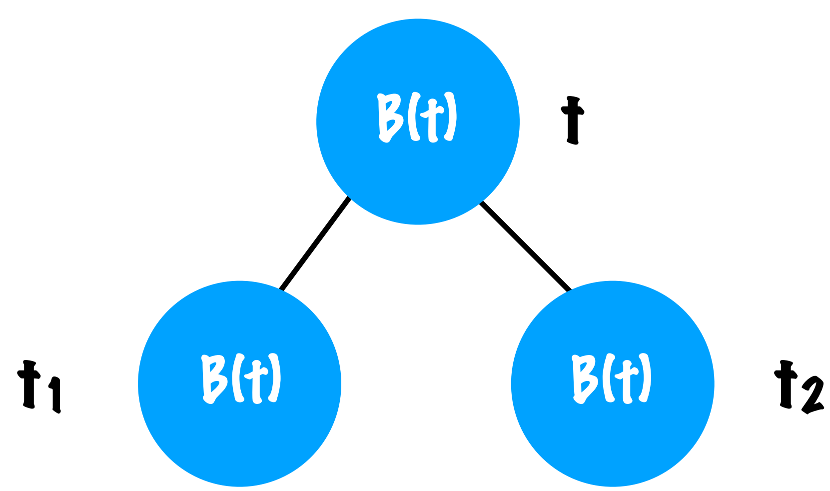
Nice Tree Decomposition



- **Introduce node:**
  - a node t with exactly one child t' such that $B(t) = B(t') \cup \{v\}$ for some vertex v not in $B(t')$

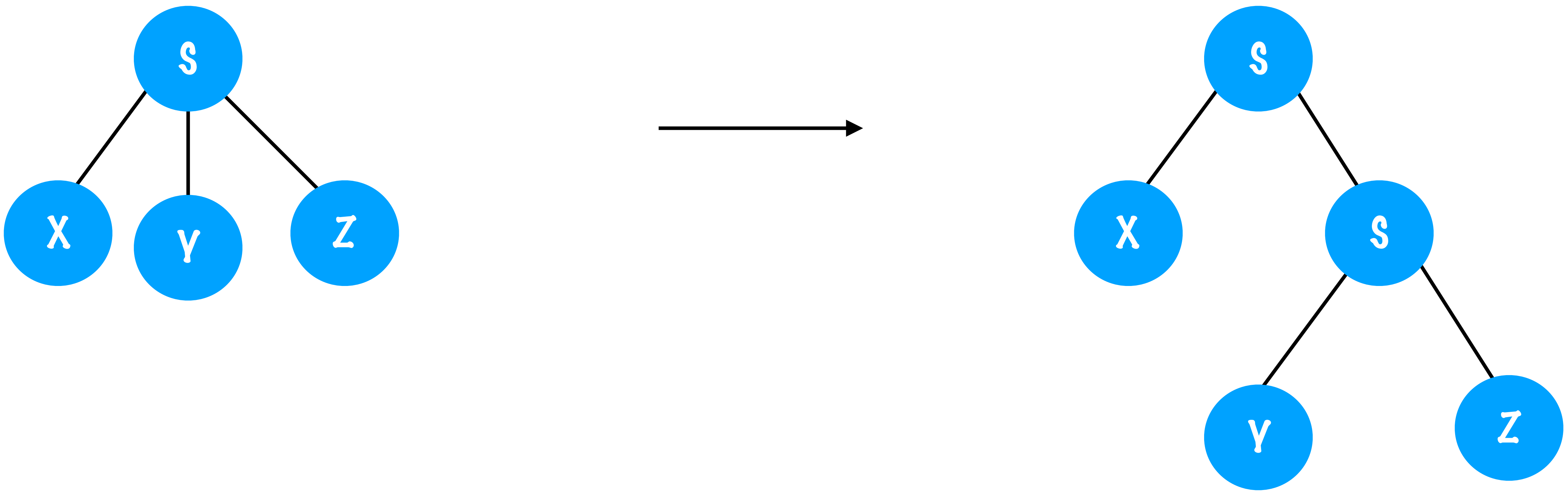


- **Forget node:**
  - a node t with exactly one child t' such that $B(t') = B(t) \cup \{v\}$ for some vertex v not in $B(t)$
- **Join node:**
  - a node t with 2 children t_1 and t_2 such that $B(t) = B(t_1) = B(t_2)$



Nice Tree Decomposition

Lemma: There is a poly-time algorithm that given a simple tree decomposition (T, B) of G, outputs a nice tree decomposition (T', B') s.t w(T') <= w(T) and |V(T')| is O(w(T)*|V(G)|).



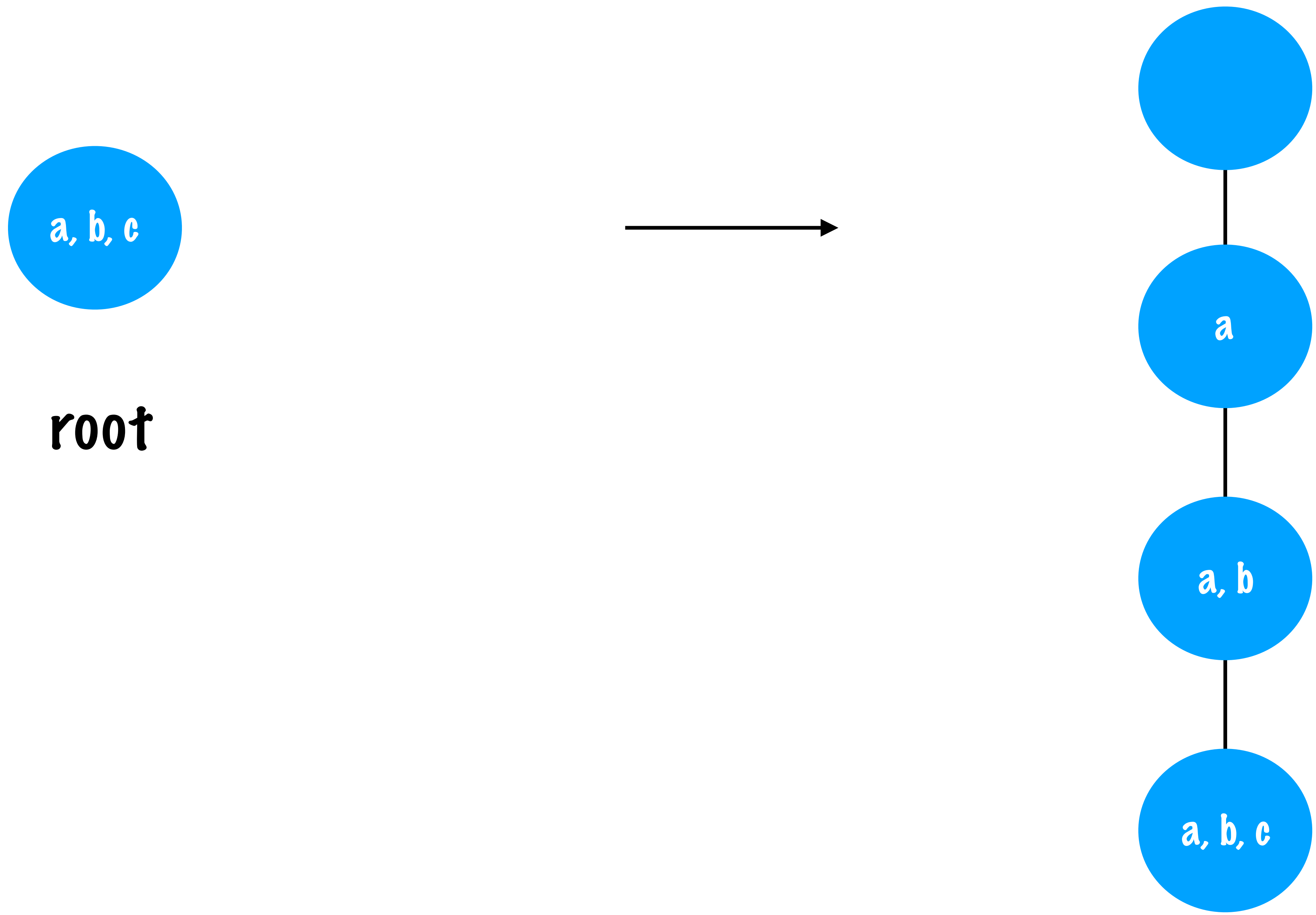
binary tree

Nice Tree Decomposition

Lemma: There is a poly-time algorithm that given a simple tree decomposition (T, B) of G, outputs a nice tree decomposition (T', B') s.t w(T') <= w(T) and |V(T')| is O(w(T)*|V(G)|).

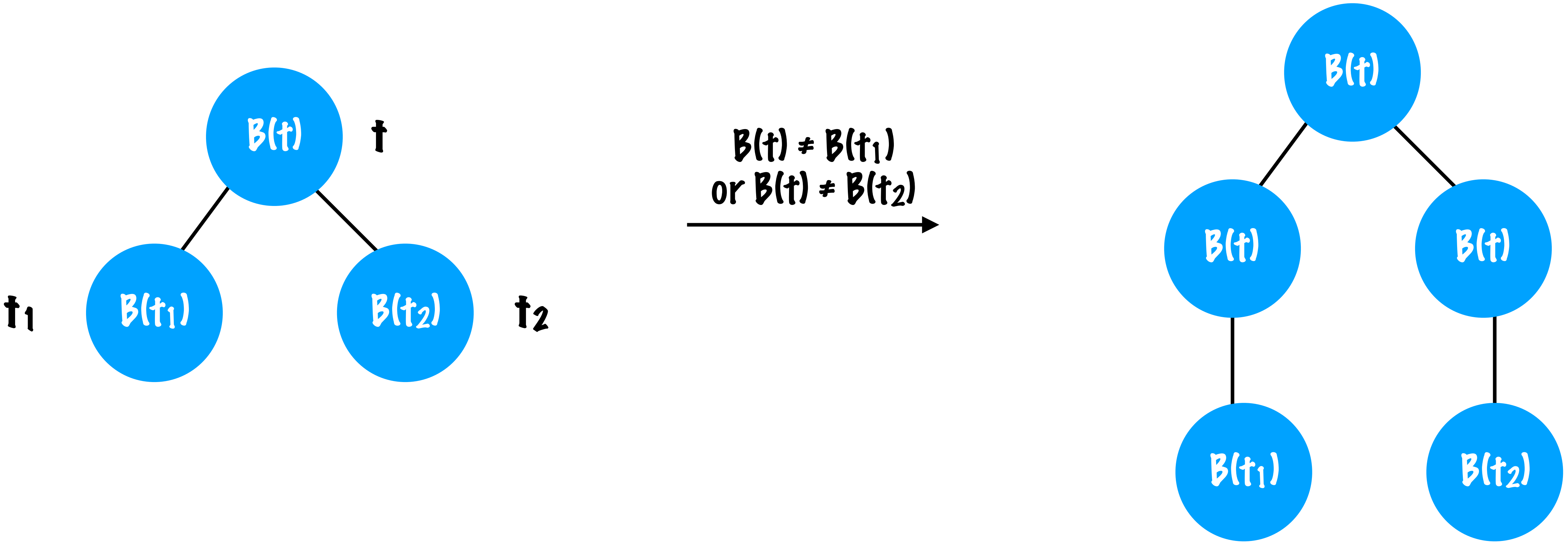
Nice Tree Decomposition

Lemma: There is a poly-time algorithm that given a simple tree decomposition (T, B) of G, outputs a nice tree decomposition (T', B') s.t w(T') <= w(T) and |V(T')| is O(w(T)*|V(G)|).



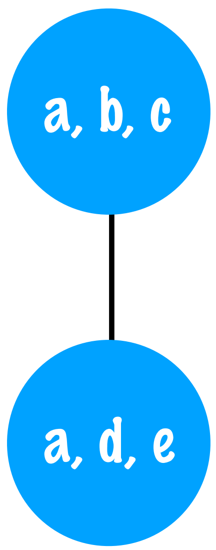
Nice Tree Decomposition

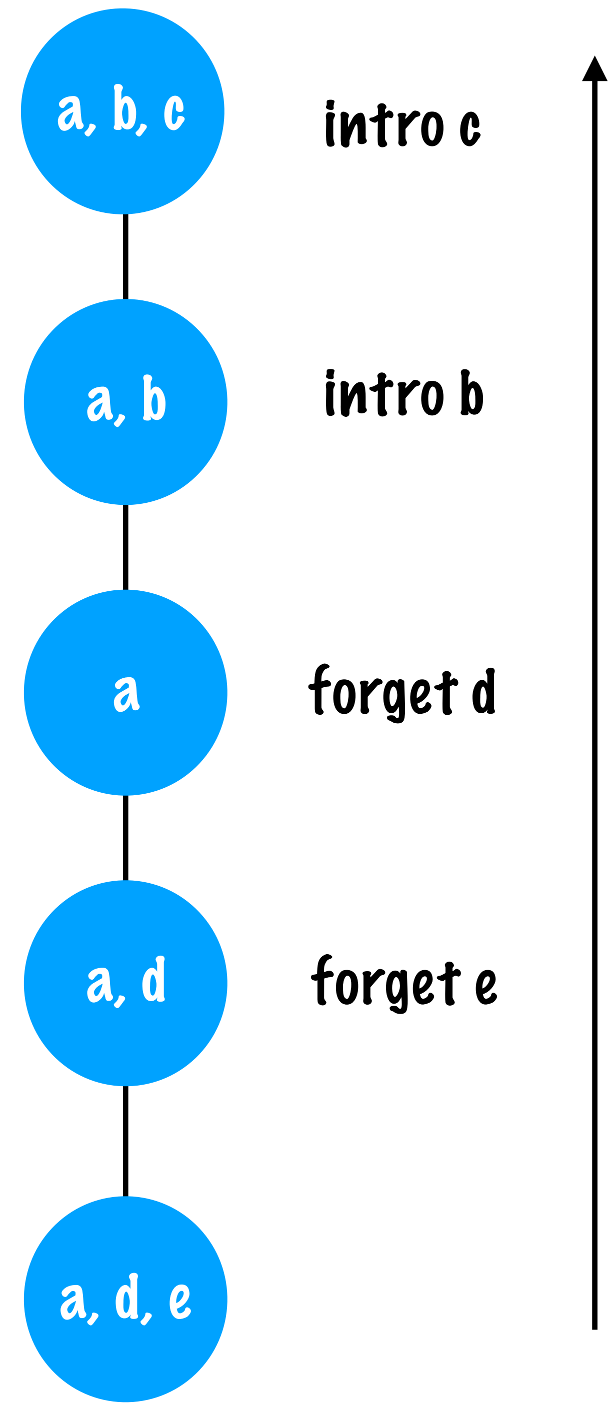
Lemma: There is a poly-time algorithm that given a simple tree decomposition (T, B) of G, outputs a nice tree decomposition (T', B') s.t w(T') <= w(T) and |V(T')| is O(w(T)*|V(G)|).



Nice Tree Decomposition

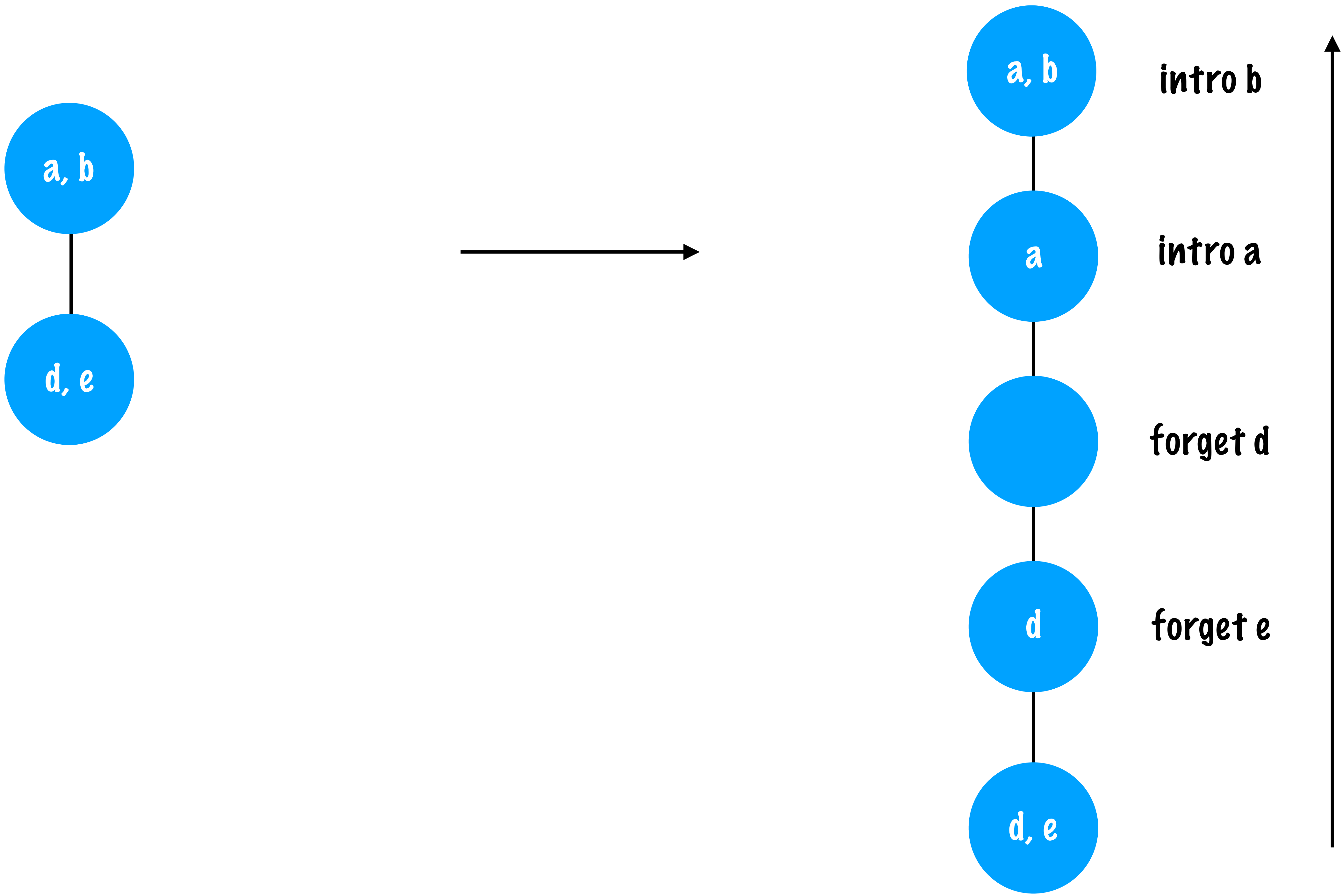
Lemma: There is a poly-time algorithm that given a simple tree decomposition (T, B) of G, outputs a nice tree decomposition (T', B') s.t w(T') <= w(T) and |V(T')| is O(w(T)*|V(G)|).





Nice Tree Decomposition

Lemma: There is a poly-time algorithm that given a simple tree decomposition (T, B) of G, outputs a nice tree decomposition (T', B') s.t w(T') <= w(T) and |V(T')| is O(w(T)*|V(G)|).



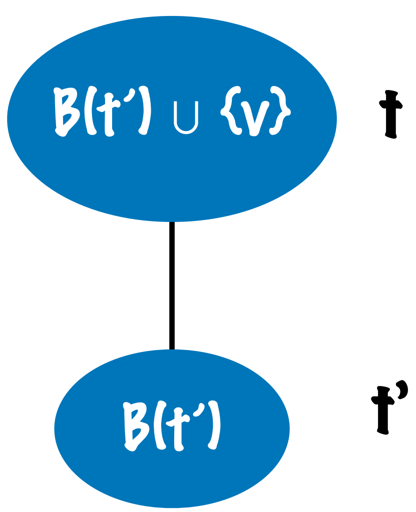
Nicer Tree Decomposition

Note: If tw(G) <=k, G has O(nk) edges

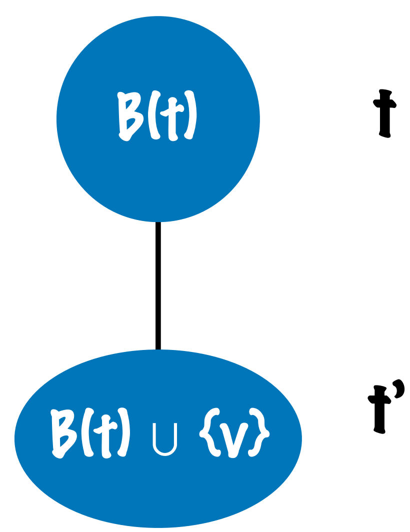
Lemma: There is a poly-time algorithm that given a nice tree decomposition (T, B) of G, outputs a nicer tree decomposition (T′, B′) s.t w(T′) = w(T) and |V(T′)| is O(|V(G)|*w(T)).

O(|V(G)|*w(T)) + O(|V(G)|*w(T)) = O(..)Also once we have introduced edge node, we can consider it for deciding closest to roo

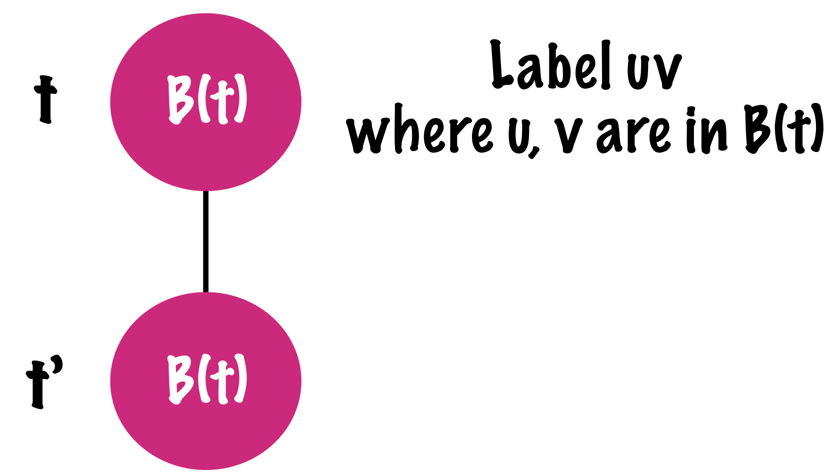
Introduce Node



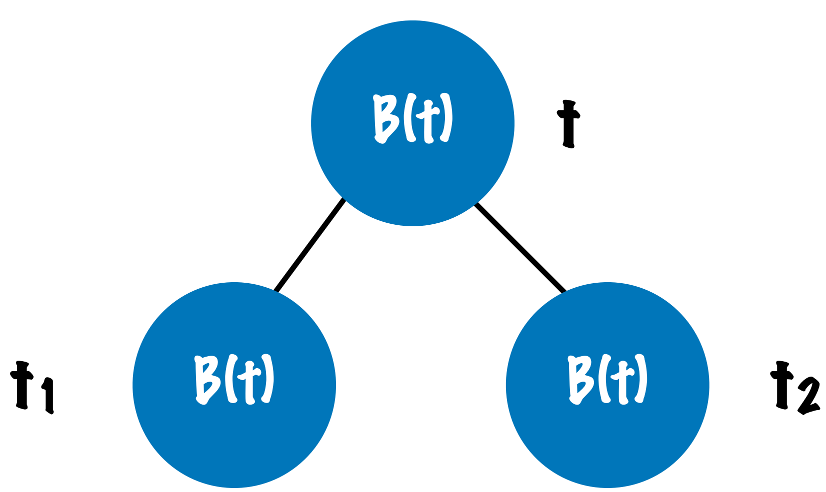
Forget Node



Introduce Edge Node



Join Node



- For every edge {u, v} in G, there is exactly one introduce edge node with label uv
  - Look at a node t′ closest to root containing u,v in its bag
  - Add an introduce edge node t as the parent of t′