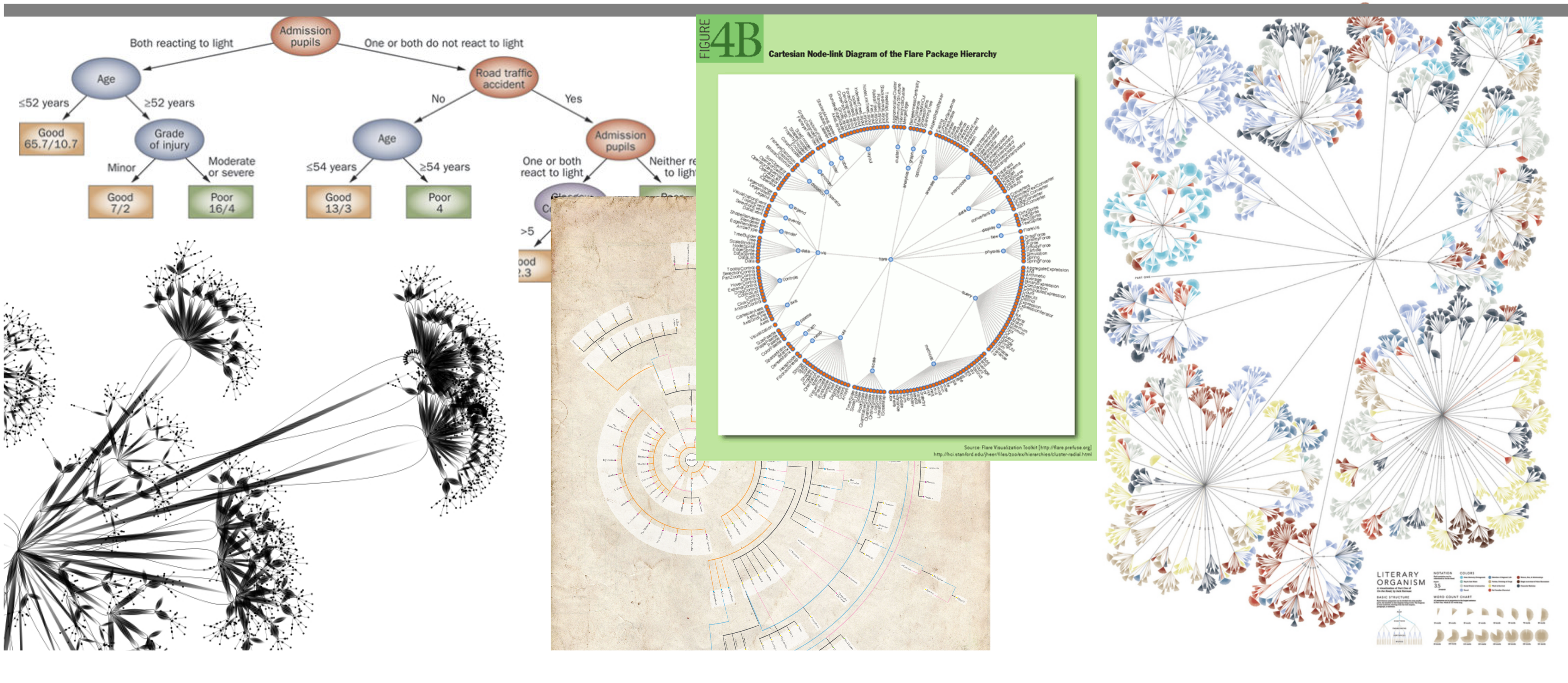
Algorithms for Visualization of Trees

Course : Data Visualization
Lecturer : Tamara Mchedlidze
Utrecht University, Dept. of Information and Computing Sciences

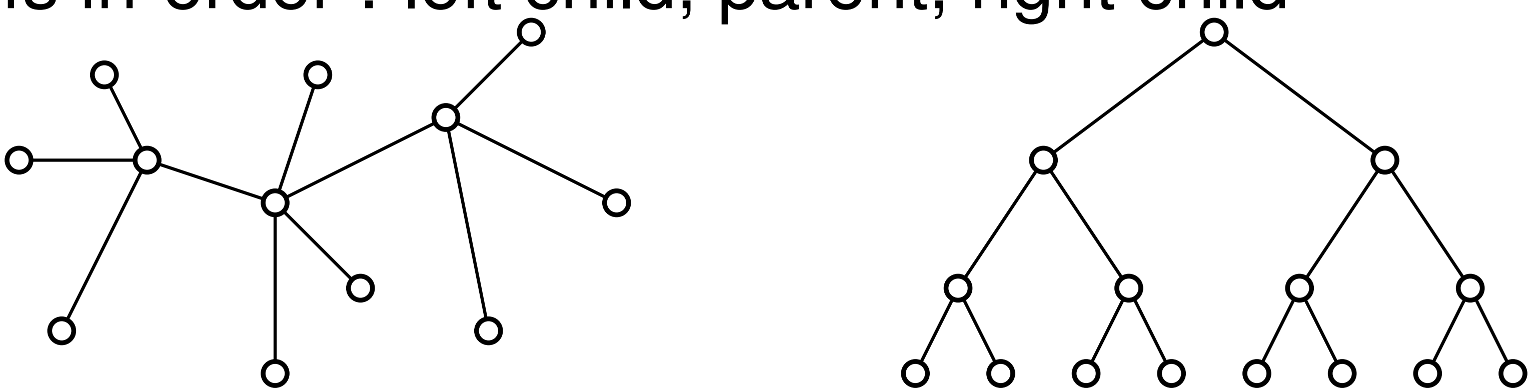


Lecture Overview

- Tree and its traversals
- Examples of trees and their visualizations
- Level-based layout
- Radial layout
- Bubble layout

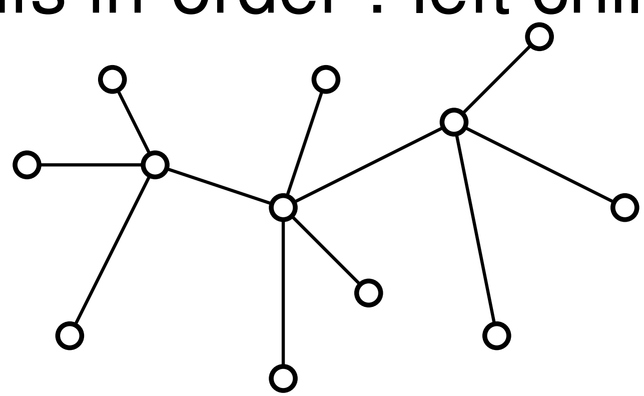
Tree and its traversals

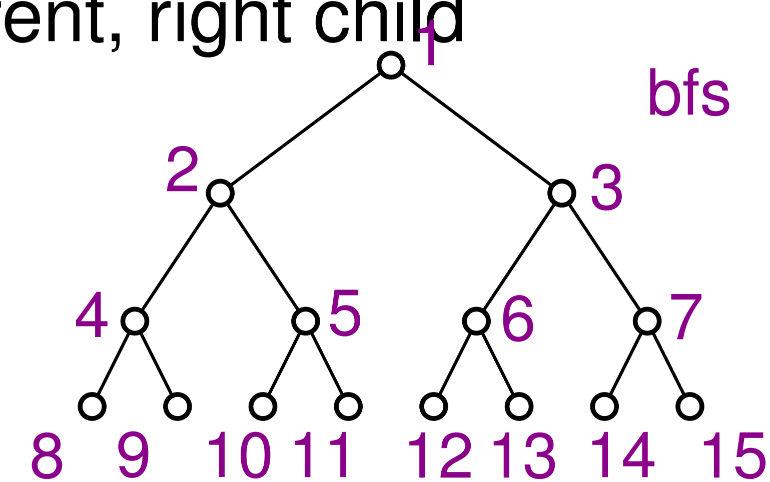
- Tree - a connected graph without cycles
- Rooted tree
- Binary tree
- Tree traversals: breadth-first search (bfs), depth-first search (dfs)
- bfs - visit vertices in layers
- dfs pre-order : first parent then subtrees
- dfs post-order : first subtrees then parent
- dfs in-order : left child, parent, right child



Tree and its traversals

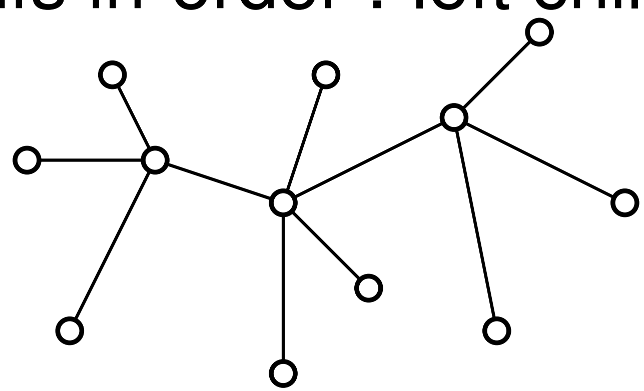
- Tree - a connected graph without cycles
- Rooted tree
- Binary tree
- Tree traversals: breadth-first search (bfs), depth-first search (dfs)
- bfs - visit vertices in layers
- dfs pre-order : first parent then subtrees
- dfs post-order : first subtrees then parent
- dfs in-order : left child, parent, right child

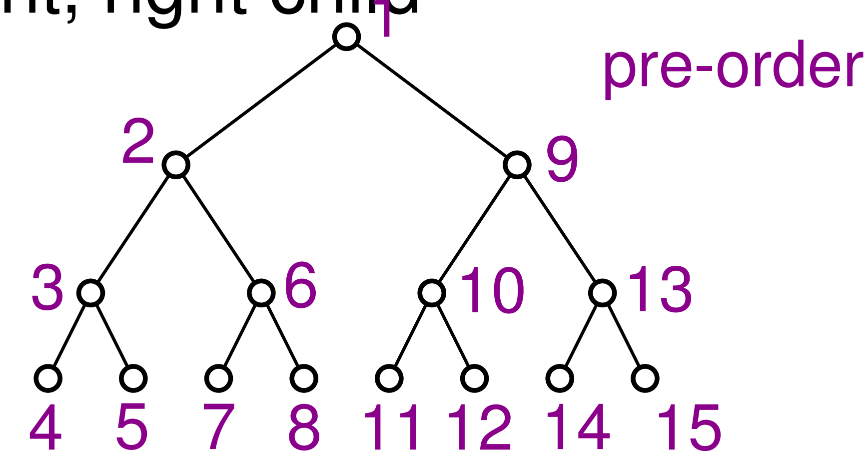




Tree and its traversals

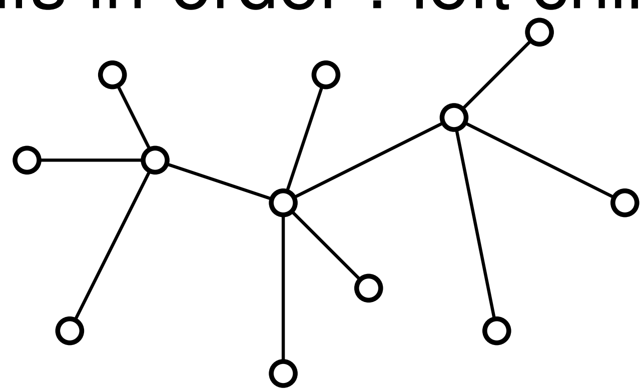
- Tree - a connected graph without cycles
- Rooted tree
- Binary tree
- Tree traversals: breadth-first search (bfs), depth-first search (dfs)
- bfs - visit vertices in layers
- dfs pre-order : first parent then subtrees
- dfs post-order : first subtrees then parent
- dfs in-order : left child, parent, right child

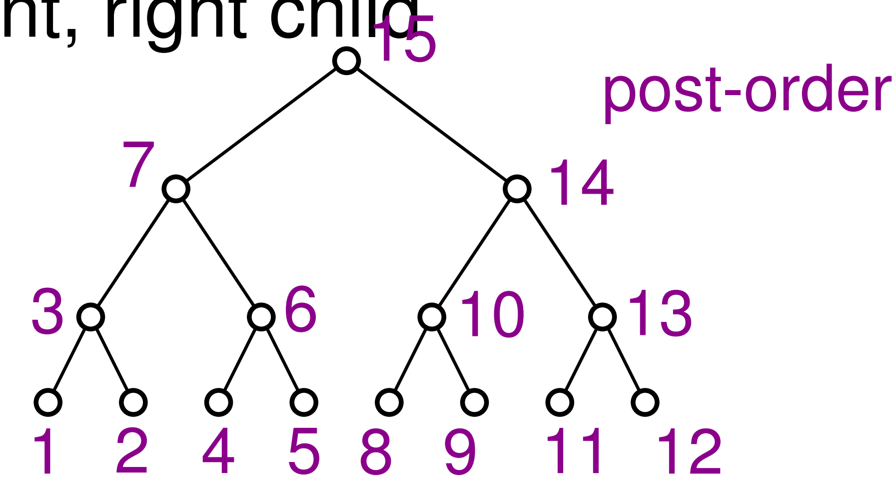




Tree and its traversals

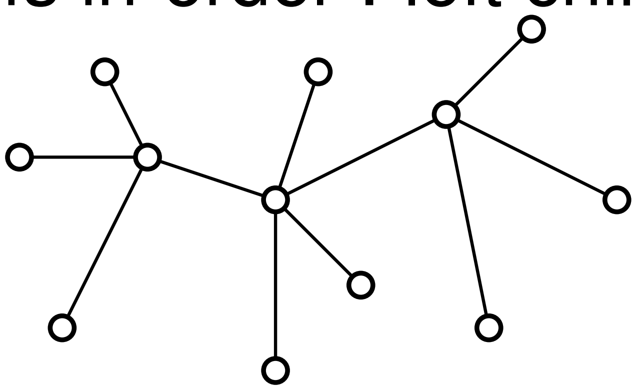
- Tree - a connected graph without cycles
- Rooted tree
- Binary tree
- Tree traversals: breadth-first search (bfs), depth-first search (dfs)
- bfs - visit vertices in layers
- dfs pre-order : first parent then subtrees
- dfs post-order : first subtrees then parent
- dfs in-order : left child, parent, right child

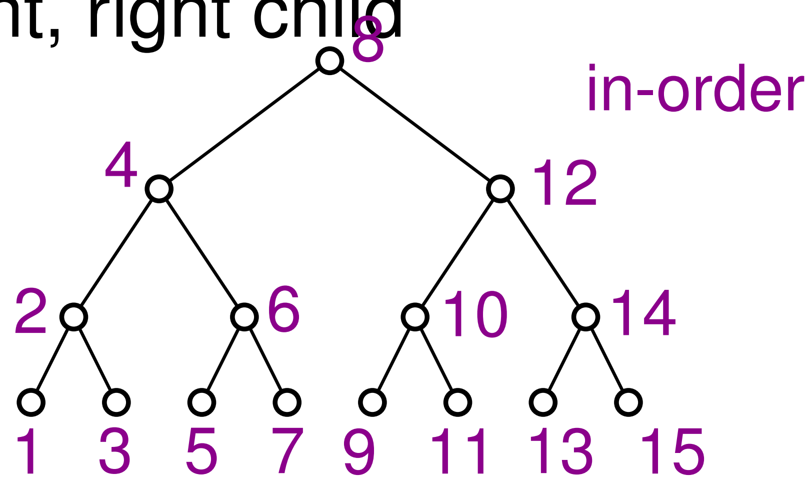




Tree and its traversals

- Tree - a connected graph without cycles
- Rooted tree
- Binary tree
- Tree traversals: breadth-first search (bfs), depth-first search (dfs)
- bfs - visit vertices in layers
- dfs pre-order : first parent then subtrees
- dfs post-order : first subtrees then parent
- dfs in-order : left child, parent, right child





Tree and its traversals

Task: Construct pre-order of the left tree and post-order of the right one

Go to Teams $->$ Lectures $->$ Whiteboard tree traversals

Hint

- dfs pre-order : first parent then subtrees
- dfs post-order : first subtrees then parent

Tree and its traversals

Task: What is the asymptotic time complexity of pre- and post-order traversals?

Hint

- Assume there are *n* vertices. How many times you visit a vertex?

Tree and its traversals

Task: What is the asymptotic time complexity of pre- and post-order traversals?

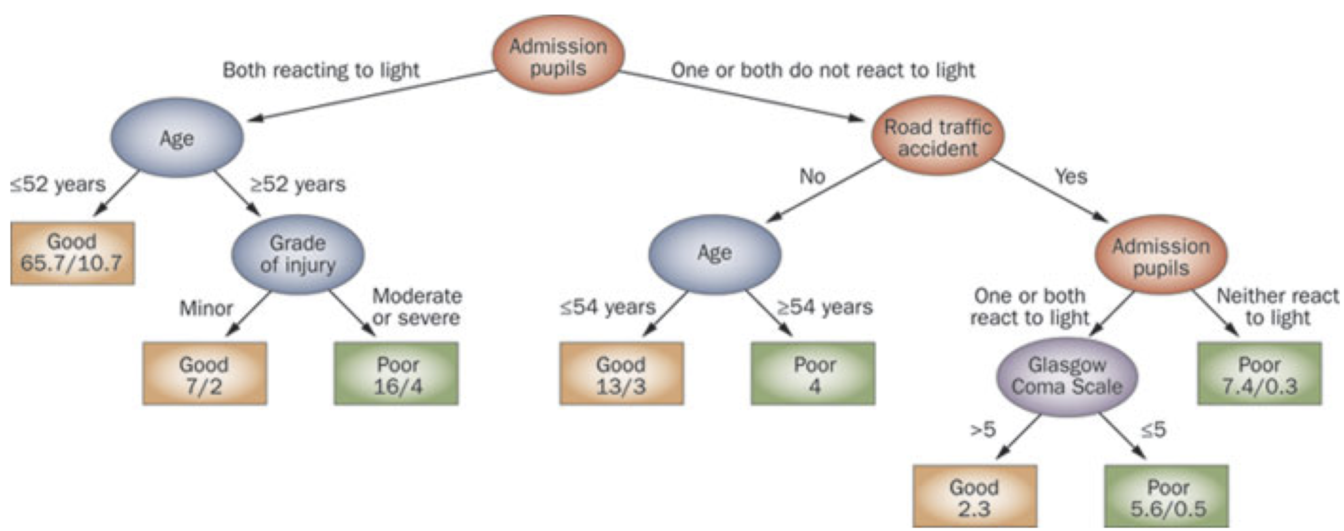
Answer: Generally all dfs and bfs traversals have time complexity $O(n)$.

Hint

- Assume there are n vertices. How many times you visit a vertex?

Level-based Layout

Task: What are the properties of this visualization? (recall ”drawing conventions”)

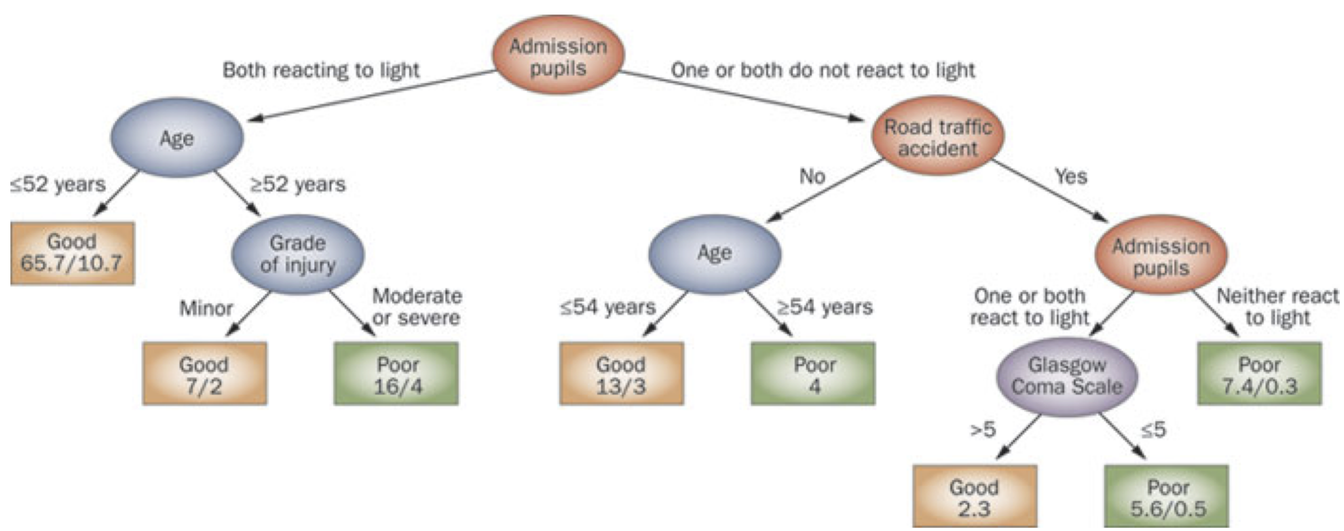


Level-based Layout

Task: What are the properties of this visualization? (recall ”drawing conventions”)

Drawing Conventions

- Vertices lie on parallel horizontal layers
- Parent is above the children
- Parent is centered with respect to the children
- Edges are straight lines
- Isomorphic subtrees have identical drawings



Level-based Layout

Algorithm Outline:

Input: A binary tree T

Output: A level-based drawing of T

Divide and Conquer algorithm

Base case: a single vertex

Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:

Level-based Layout

Algorithm Outline:

Input: A binary tree T

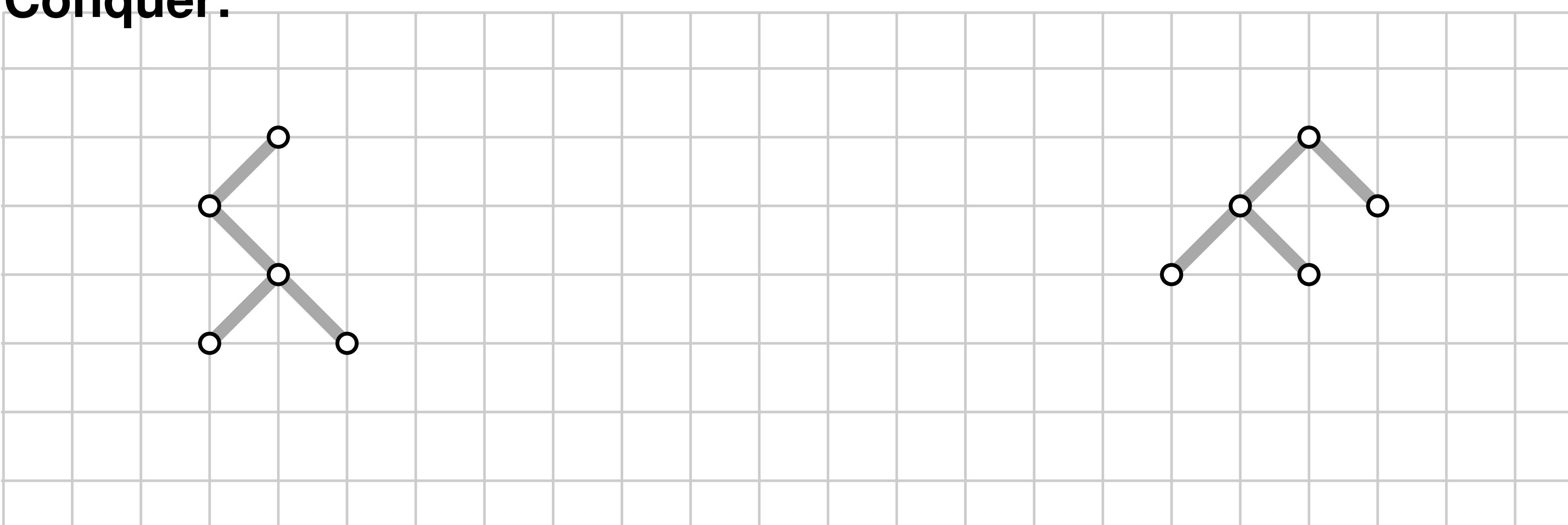
Output: A level-based drawing of T

Divide and Conquer algorithm

Base case: a single vertex

Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:



Level-based Layout

Algorithm Outline:

Input: A binary tree T

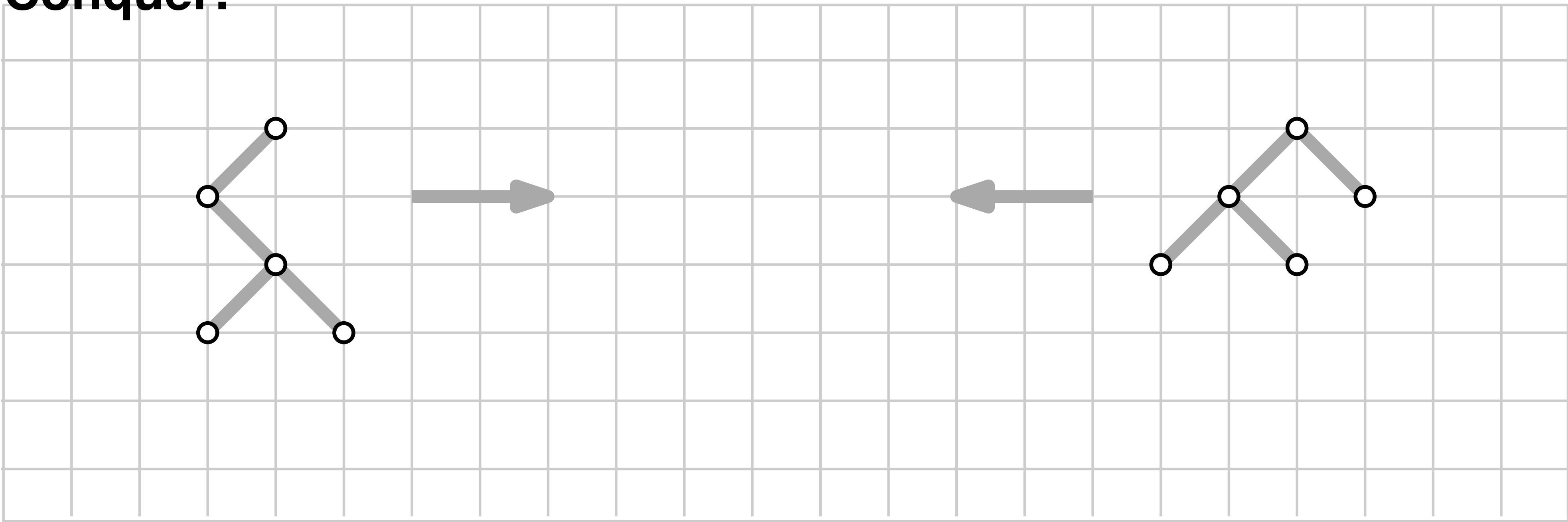
Output: A level-based drawing of T

Divide and Conquer algorithm

Base case: a single vertex

Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:



Level-based Layout

Algorithm Outline:

Input: A binary tree T

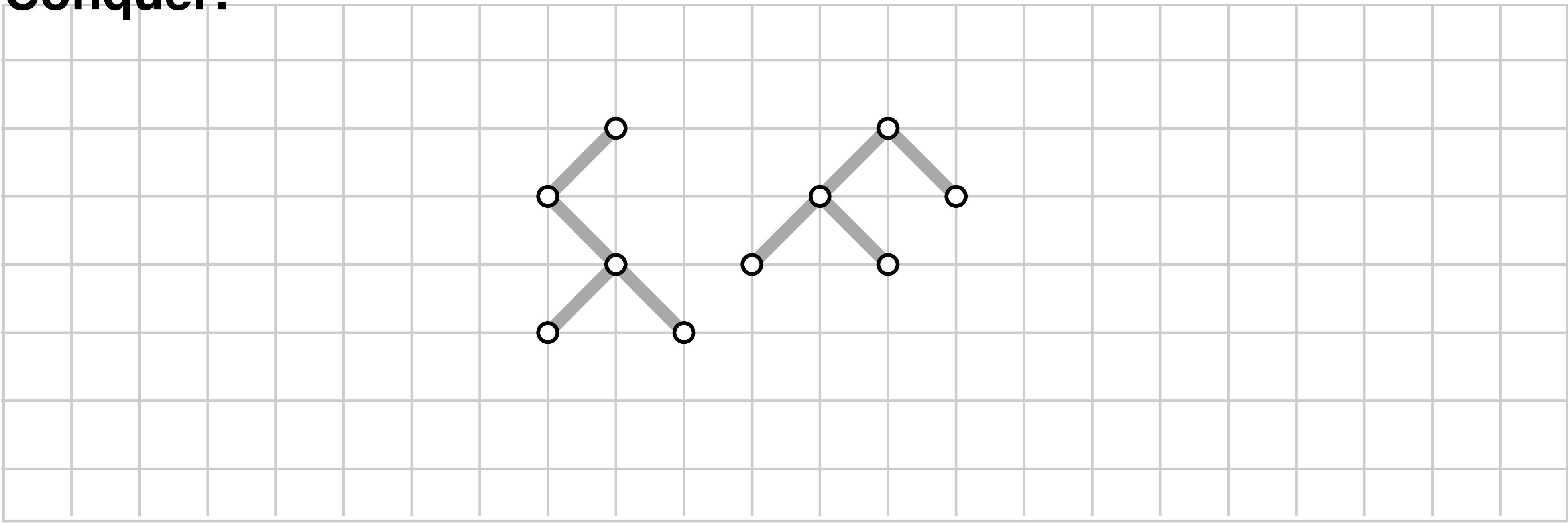
Output: A level-based drawing of T

Divide and Conquer algorithm

Base case: a single vertex

Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:



Level-based Layout

Algorithm Outline:

Input: A binary tree T

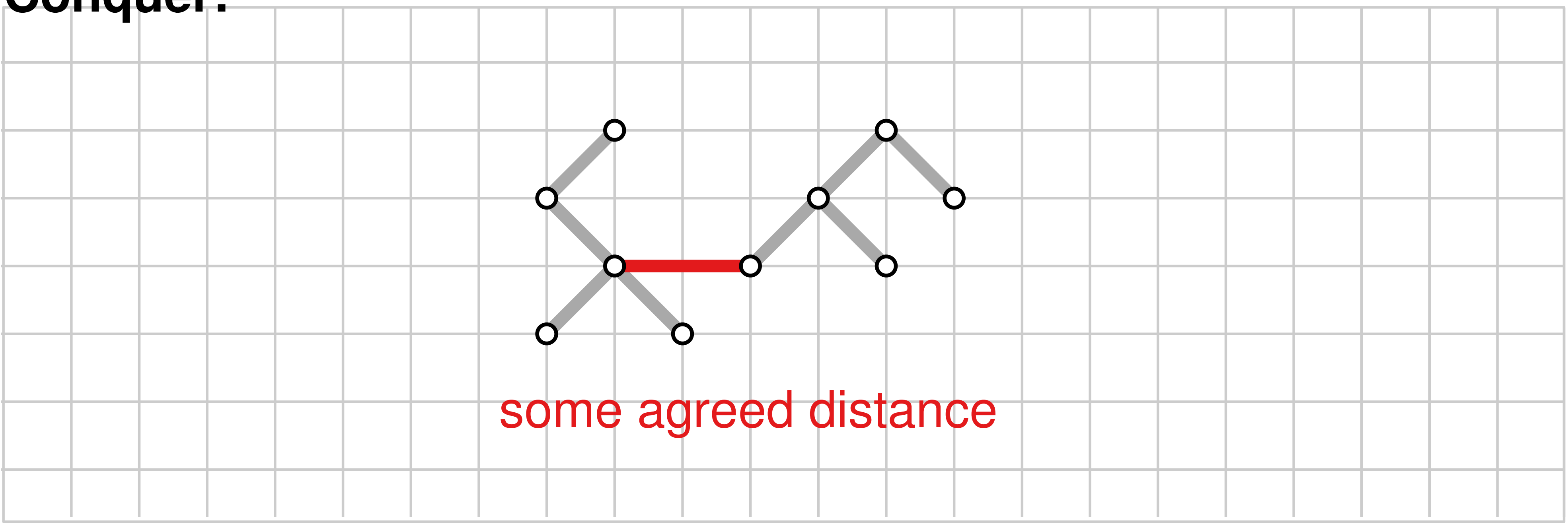
Output: A level-based drawing of T

Divide and Conquer algorithm

Base case: a single vertex

Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:



Level-based Layout

Algorithm Outline:

Input: A binary tree T

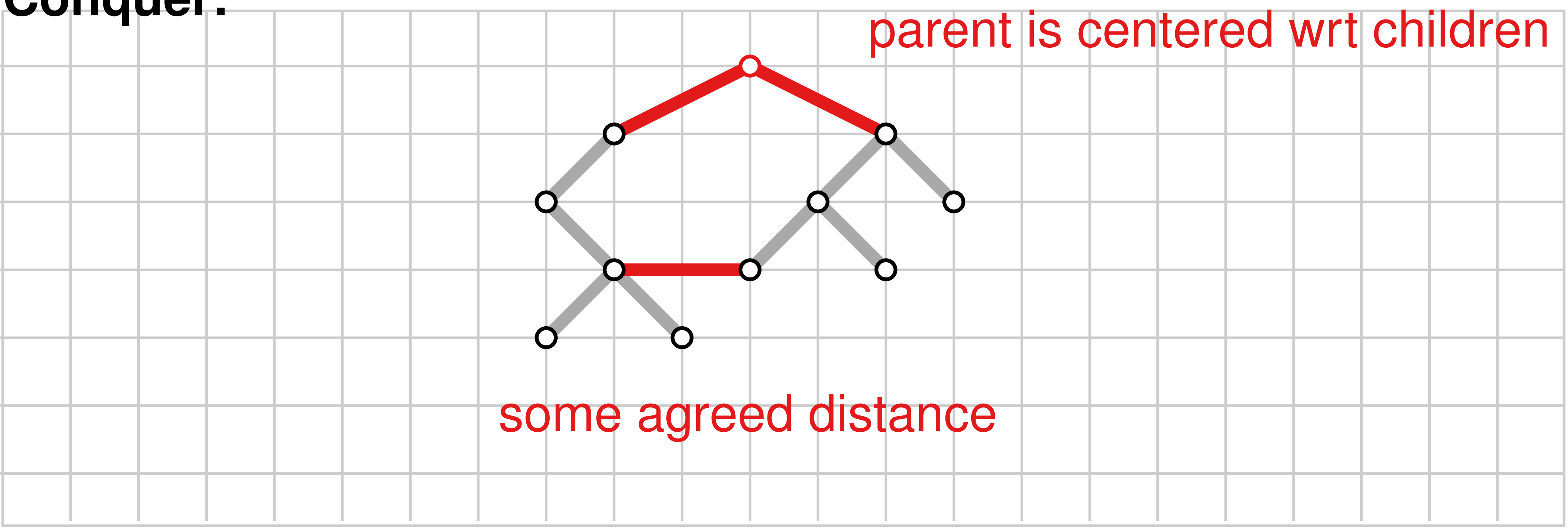
Output: A level-based drawing of T

Divide and Conquer algorithm

Base case: a single vertex

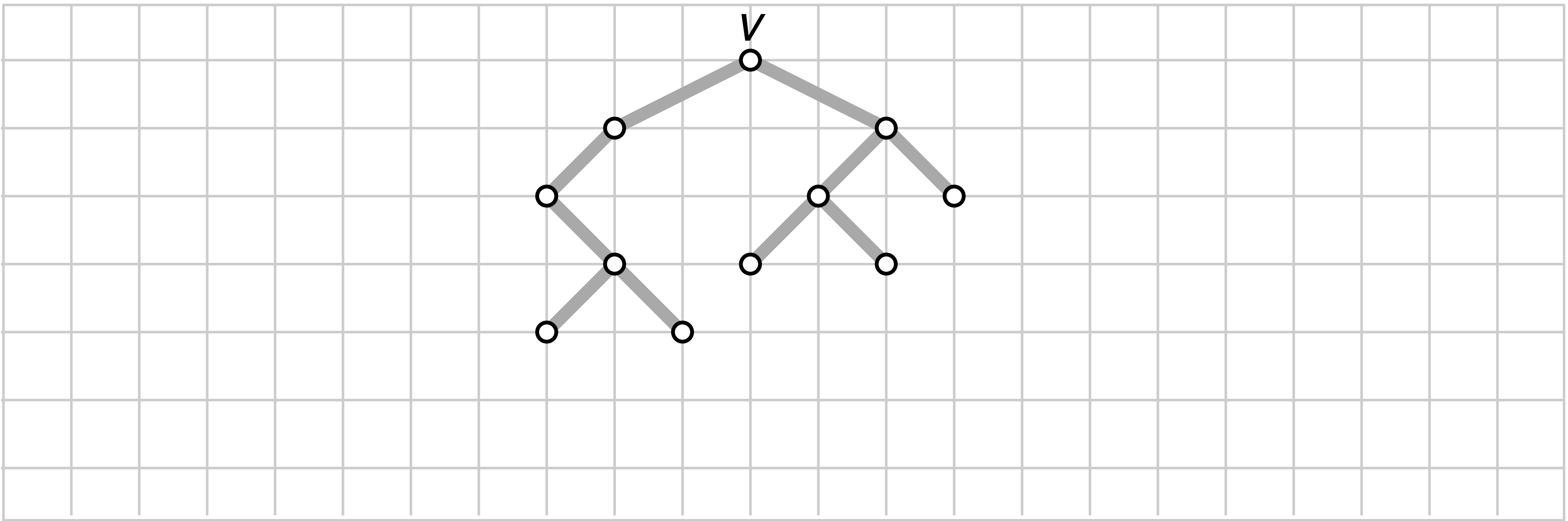
Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:



Level-based Layout

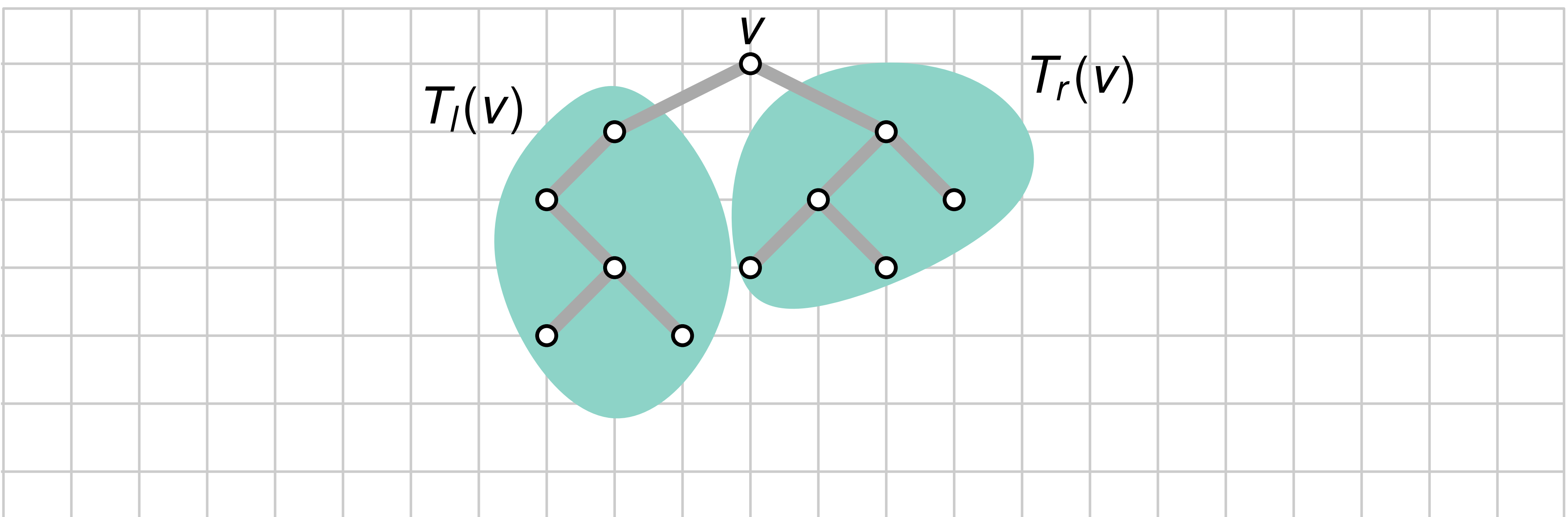
Implementation Details (postorder and preorder traversals)
Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child



Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

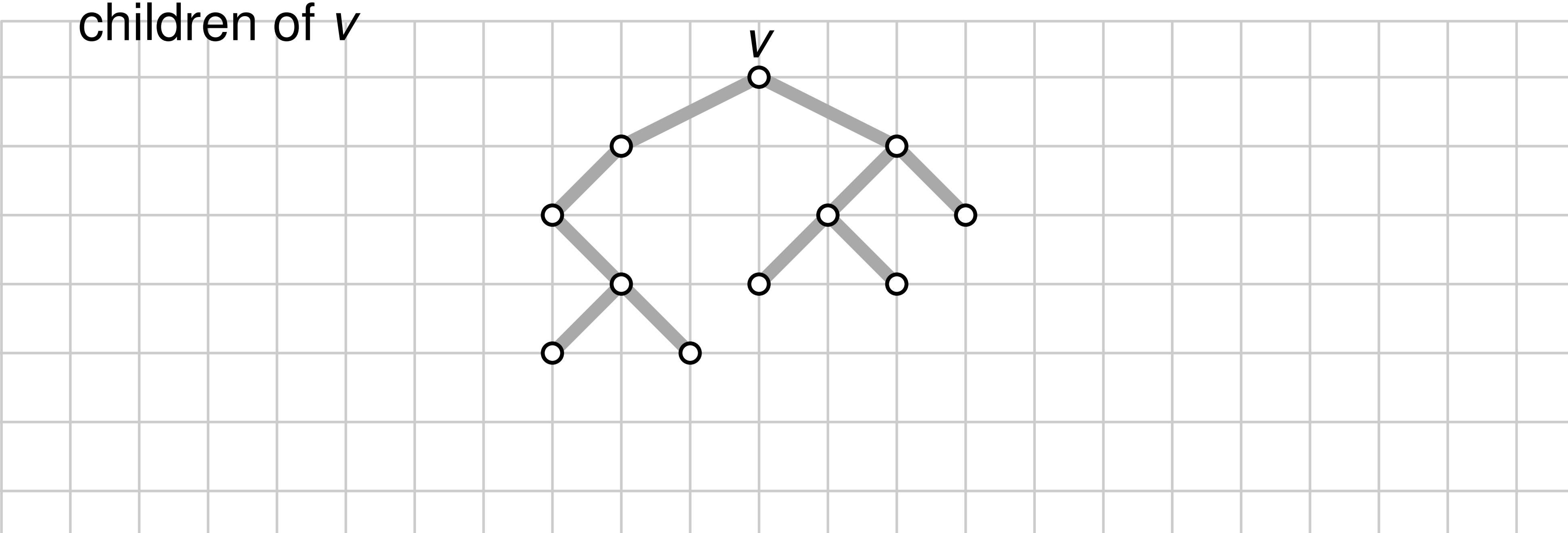


Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

- Assume at each vertex *u* (below *v*) we have stored the left and the right boundary of the subtree $T(u)$ and the horizontal displacements of the children
- "Summ up" the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of *v*

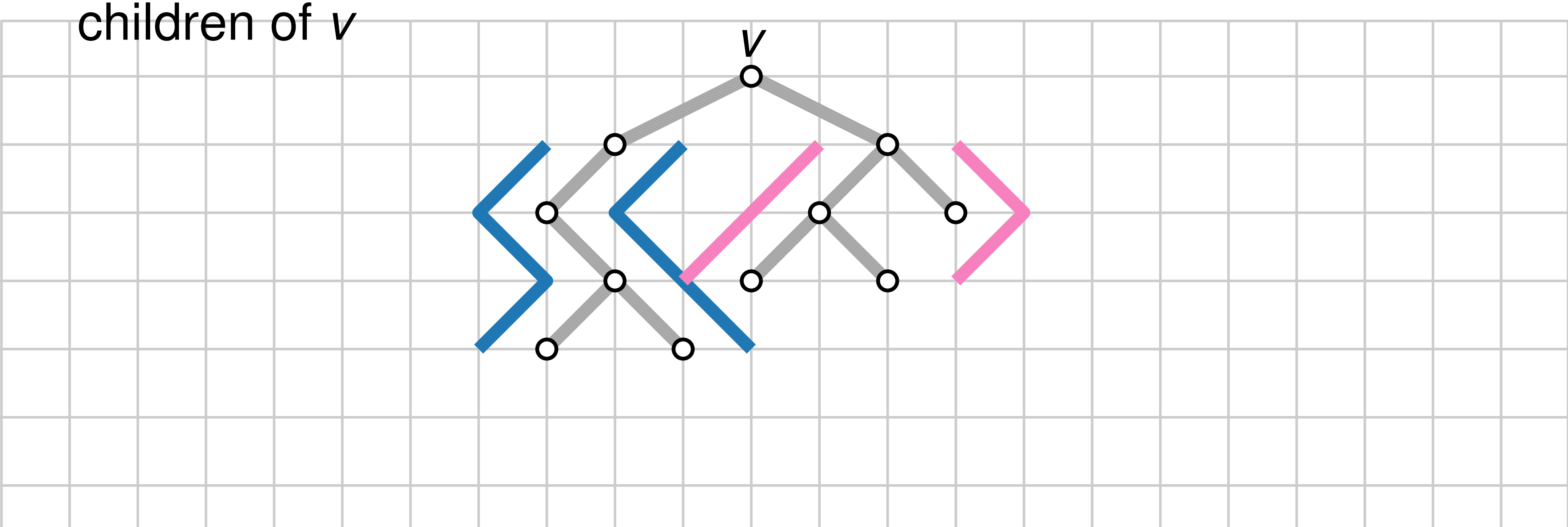


Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

- Assume at each vertex *u* (below *v*) we have stored the left and the right boundary of the subtree $T(u)$ and the horizontal displacements of the children
- "Summ up" the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of *v*

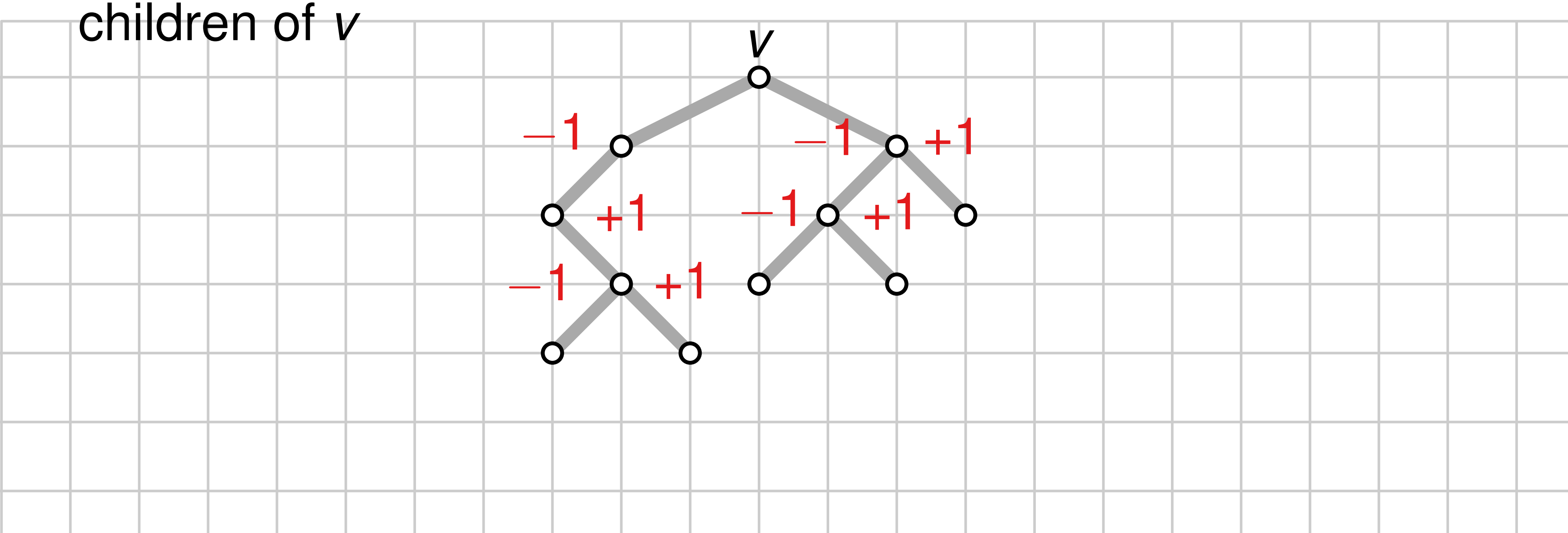


Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$ and the horizontal displacements of the children
- "Summ up" the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of v

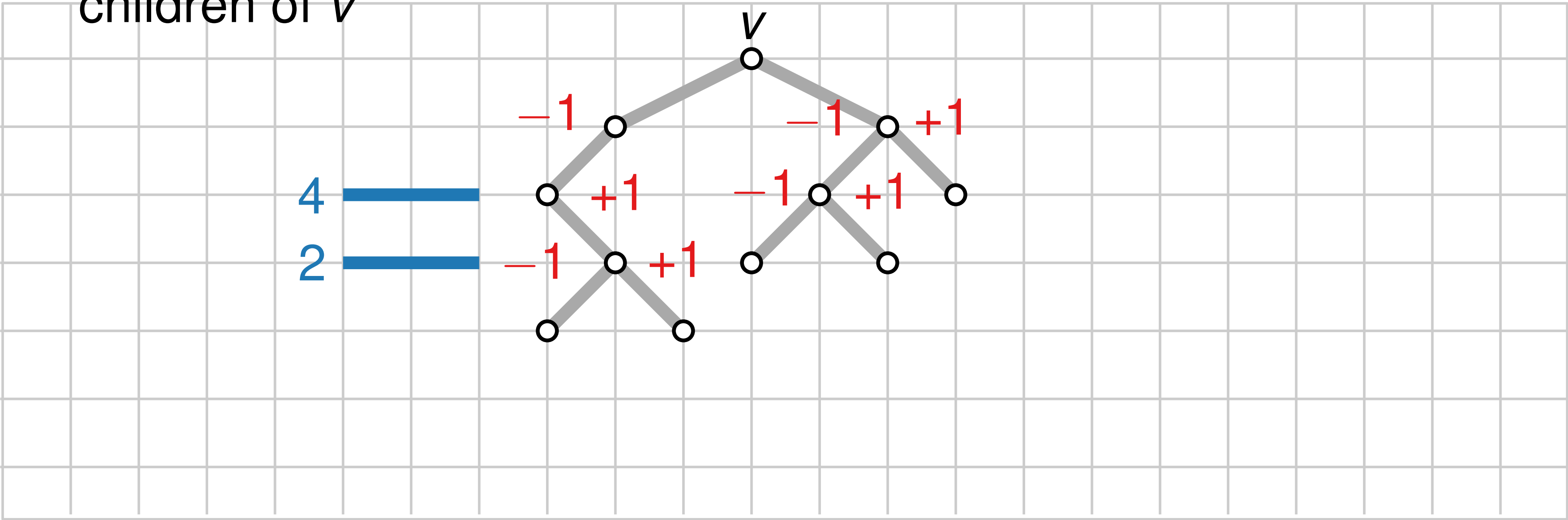


Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

- Assume at each vertex *u* (below *v*) we have stored the left and the right boundary of the subtree $T(u)$ and the horizontal displacements of the children
- "Summ up" the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of *v*

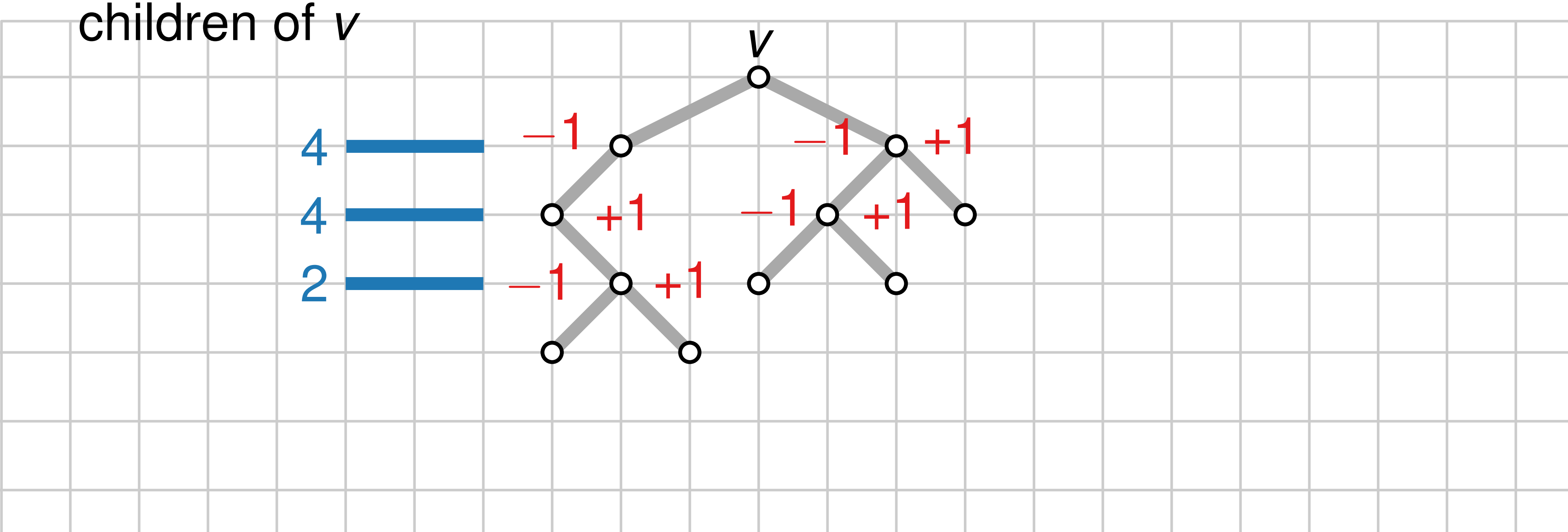


Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$ and the horizontal displacements of the children
- "Summ up" the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of v

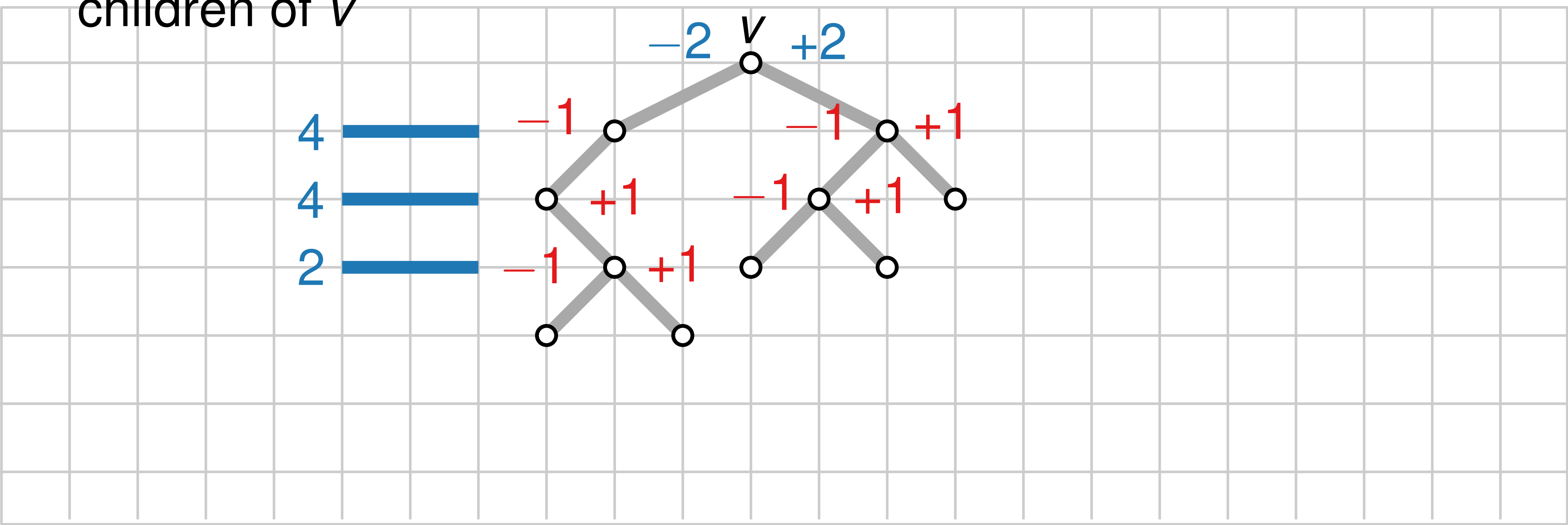


Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$ and the horizontal displacements of the children
- "Summ up" the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of v

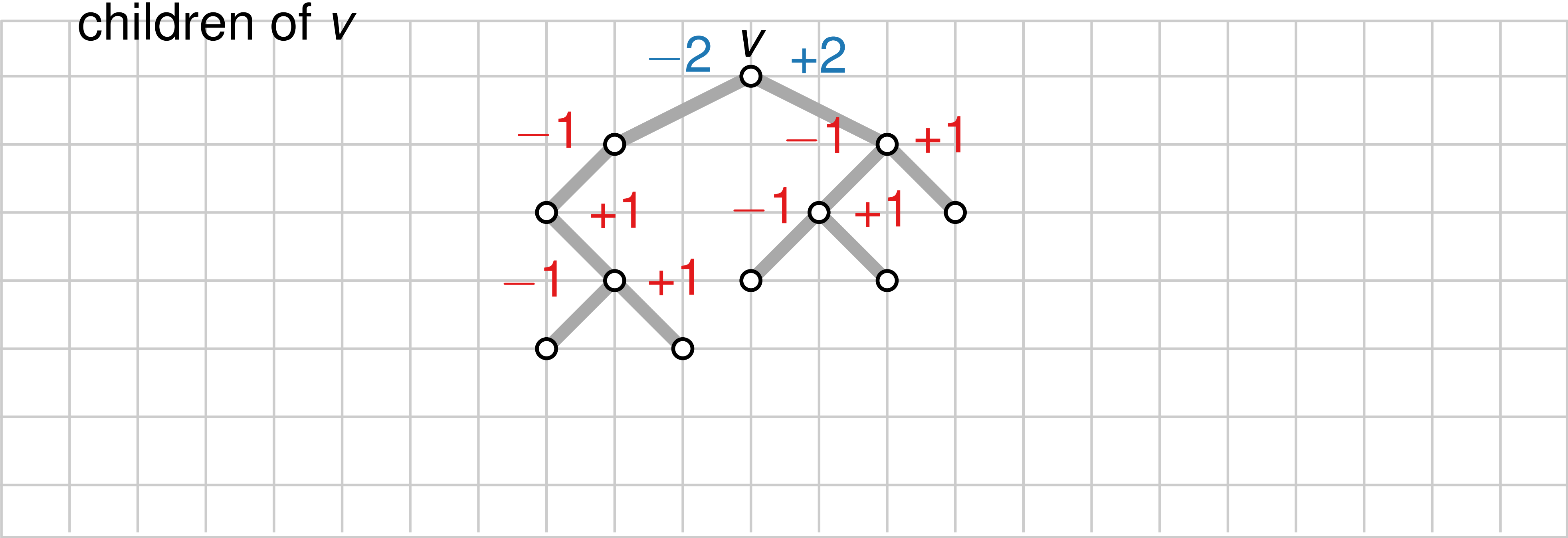


Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

- Assume at each vertex *u* (below *v*) we have stored the left and the right boundary of the subtree $T(u)$ and the horizontal displacements of the children
- "Summ up" the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of *v*

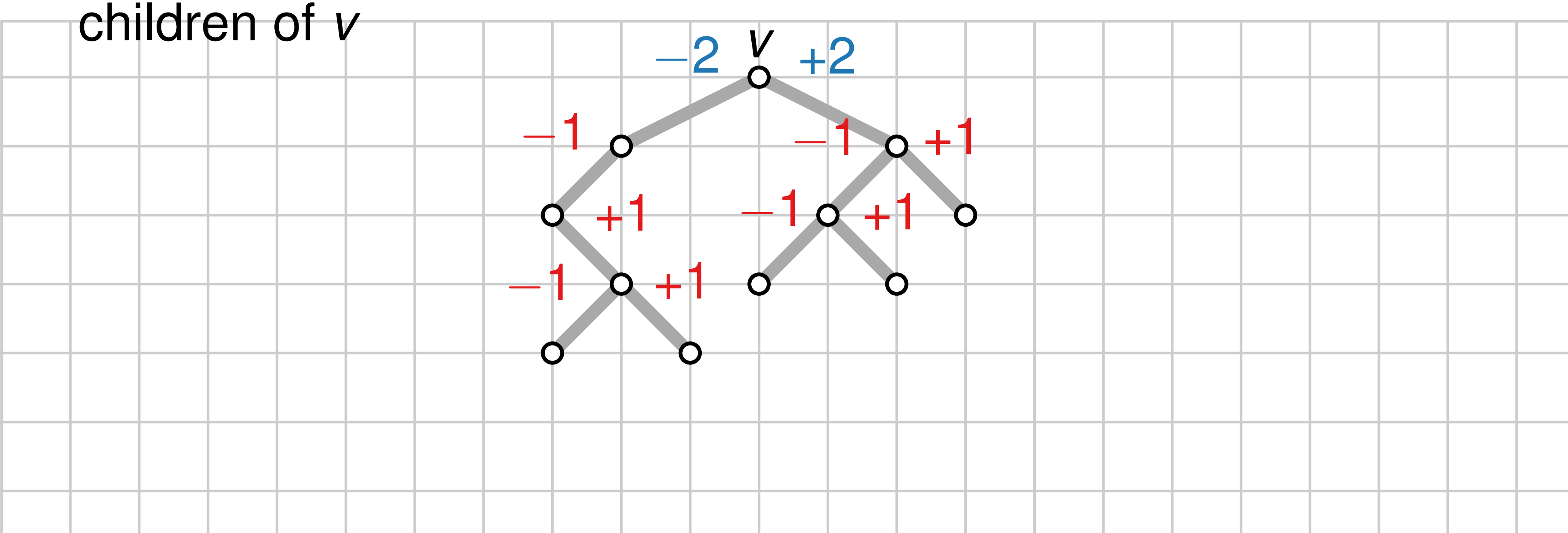


Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

- Assume at each vertex *u* (below *v*) we have stored the left and the right boundary of the subtree $T(u)$ and the horizontal displacements of the children
- “Summ up” the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of *v*



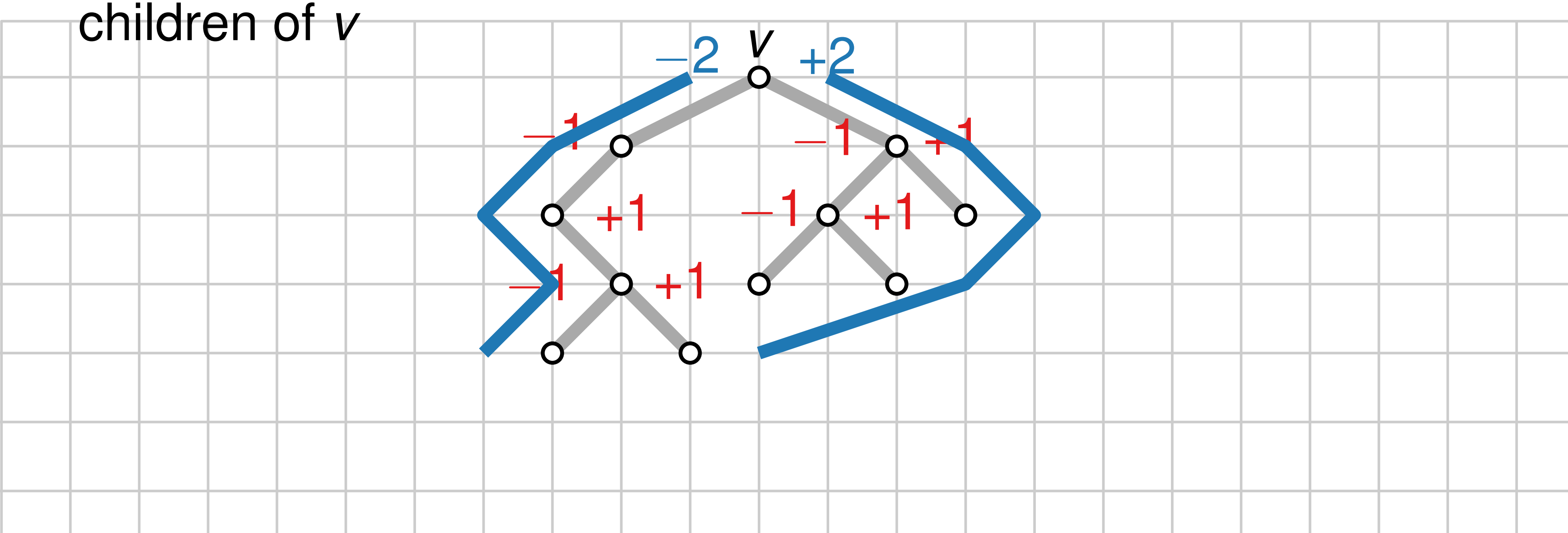
- Store at *v* the left and the right boundaries of T(v)

Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

- Assume at each vertex *u* (below *v*) we have stored the left and the right boundary of the subtree $T(u)$ and the horizontal displacements of the children
- "Summ up" the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of *v*



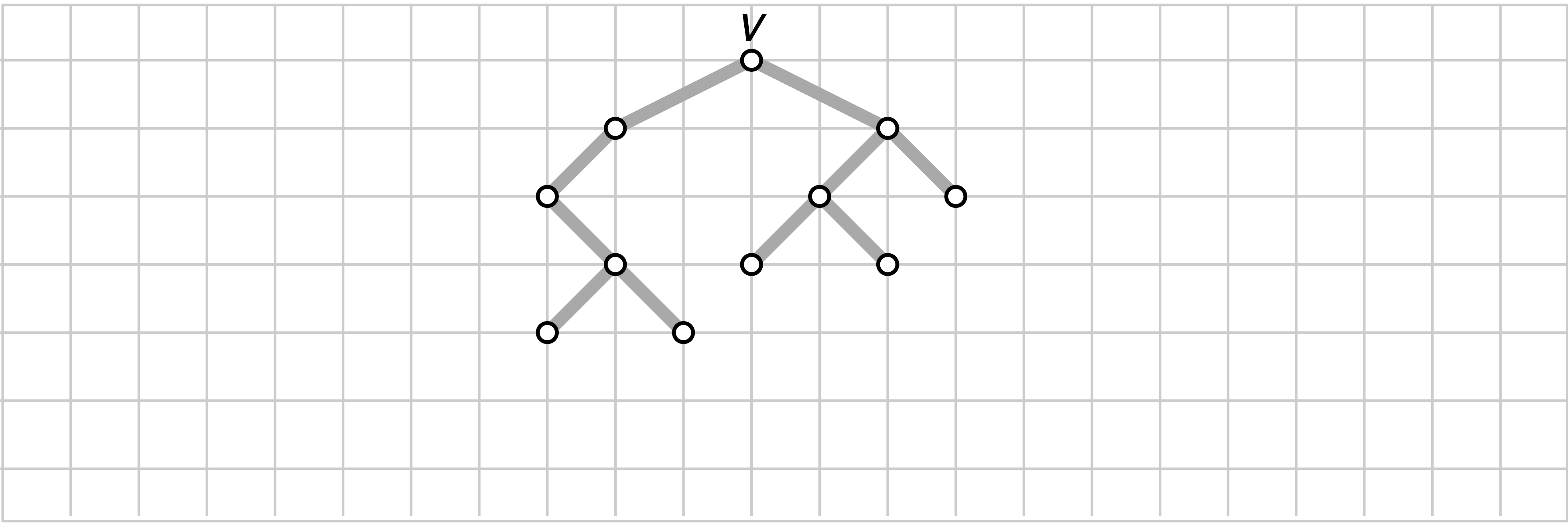
- Store at *v* the left and the right boundaries of T(v)

Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates

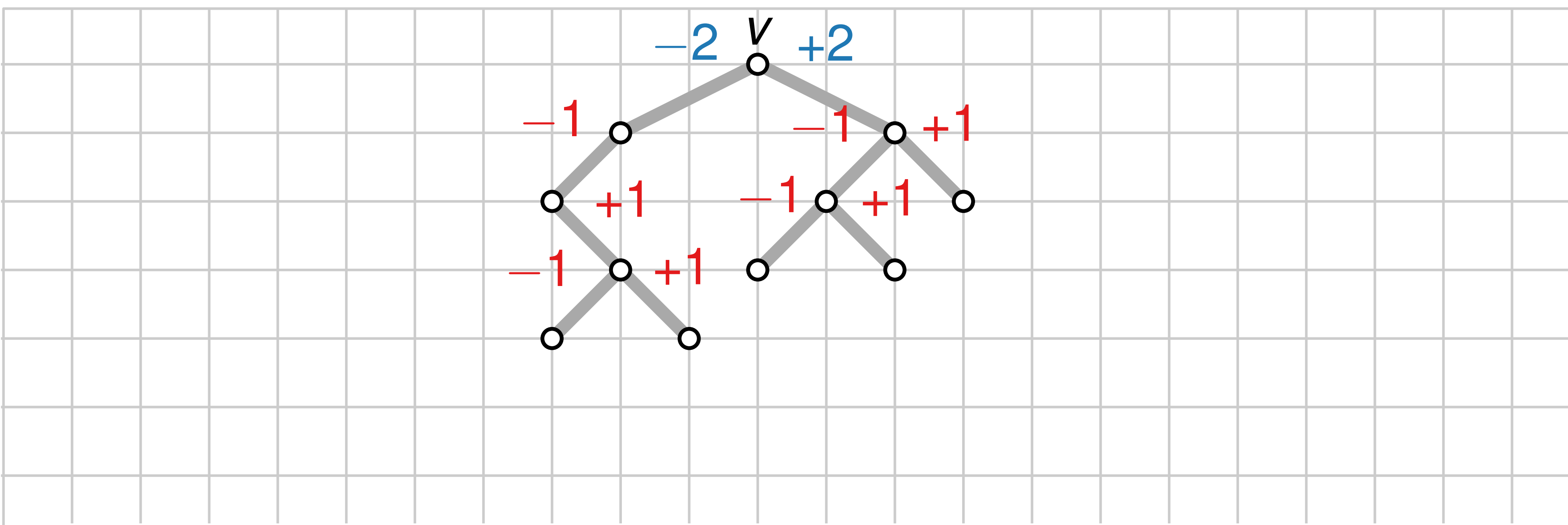


Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates

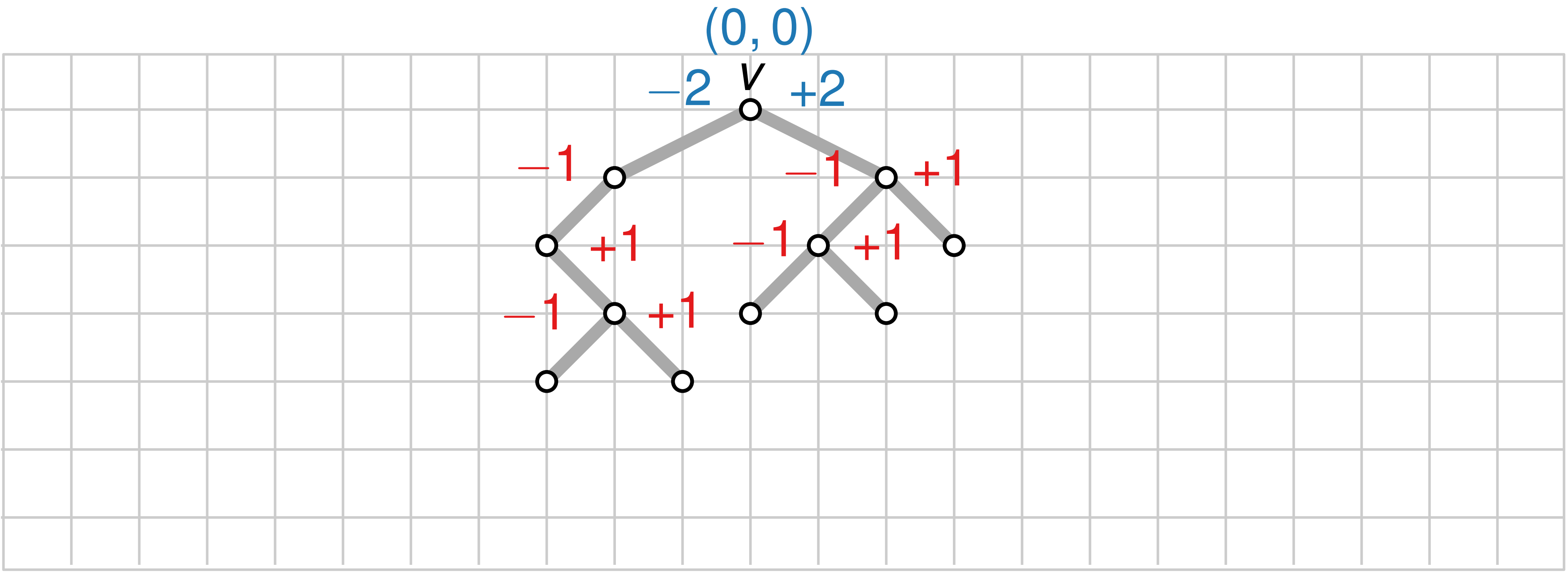


Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates

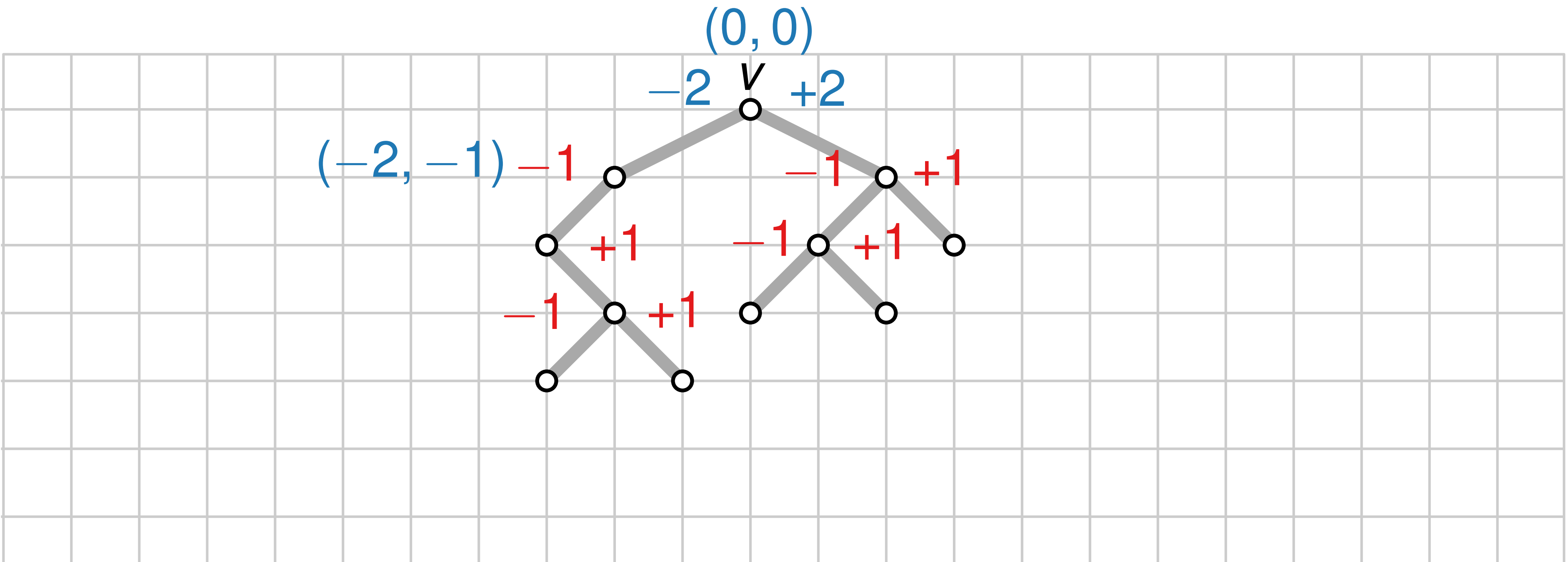


Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

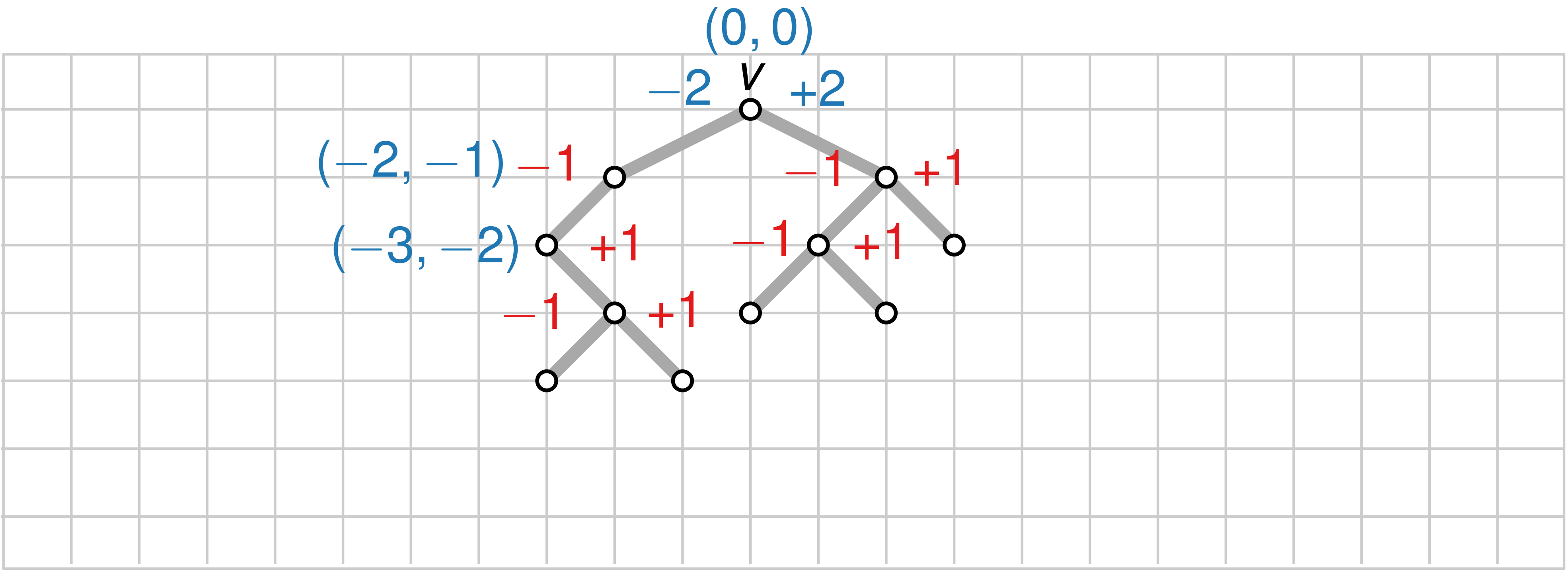
Preorder traversal: Compute x- and y- coordinates



Level-based Layout

Implementation Details (postorder and preorder traversals)
Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates

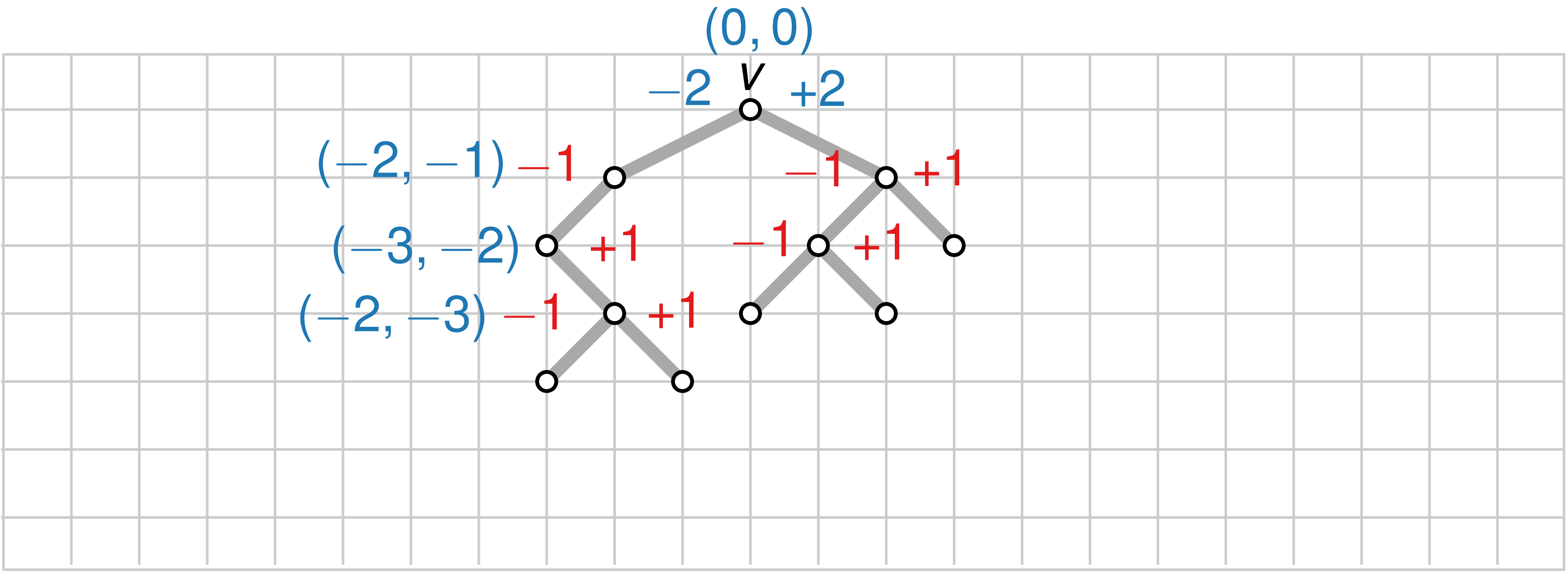


Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

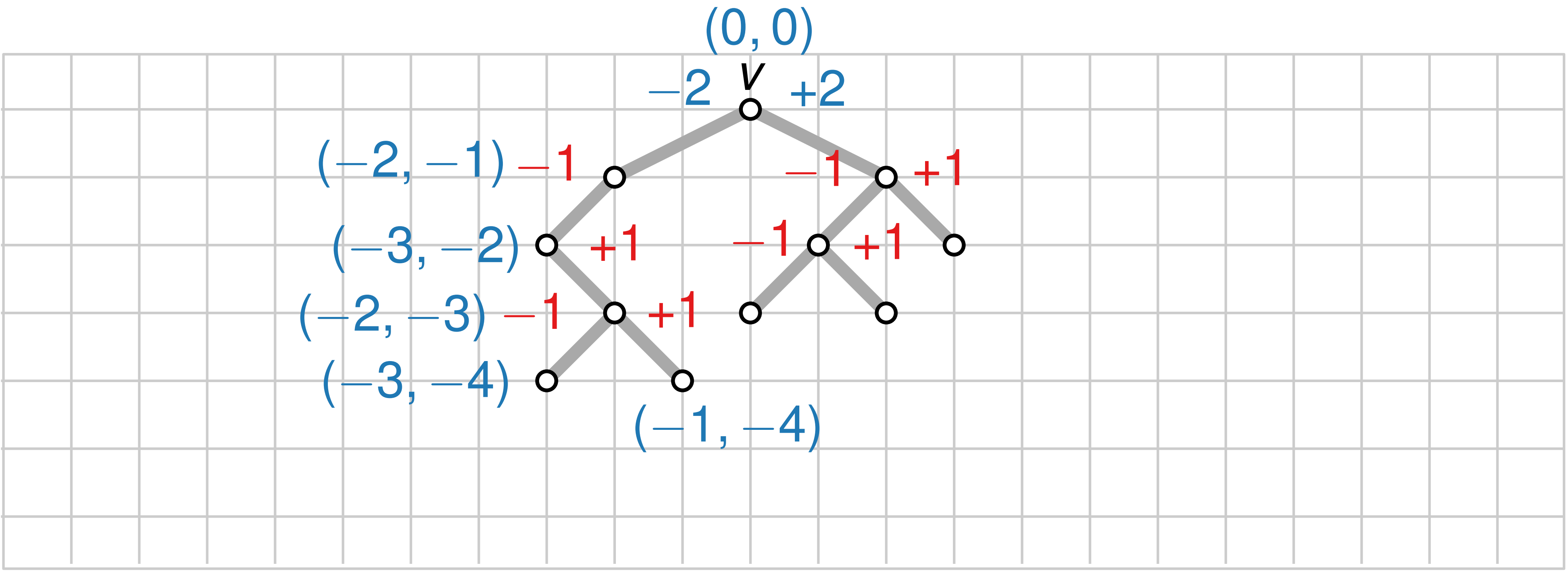
Preorder traversal: Compute x- and y- coordinates



Level-based Layout

Implementation Details (postorder and preorder traversals)
Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates



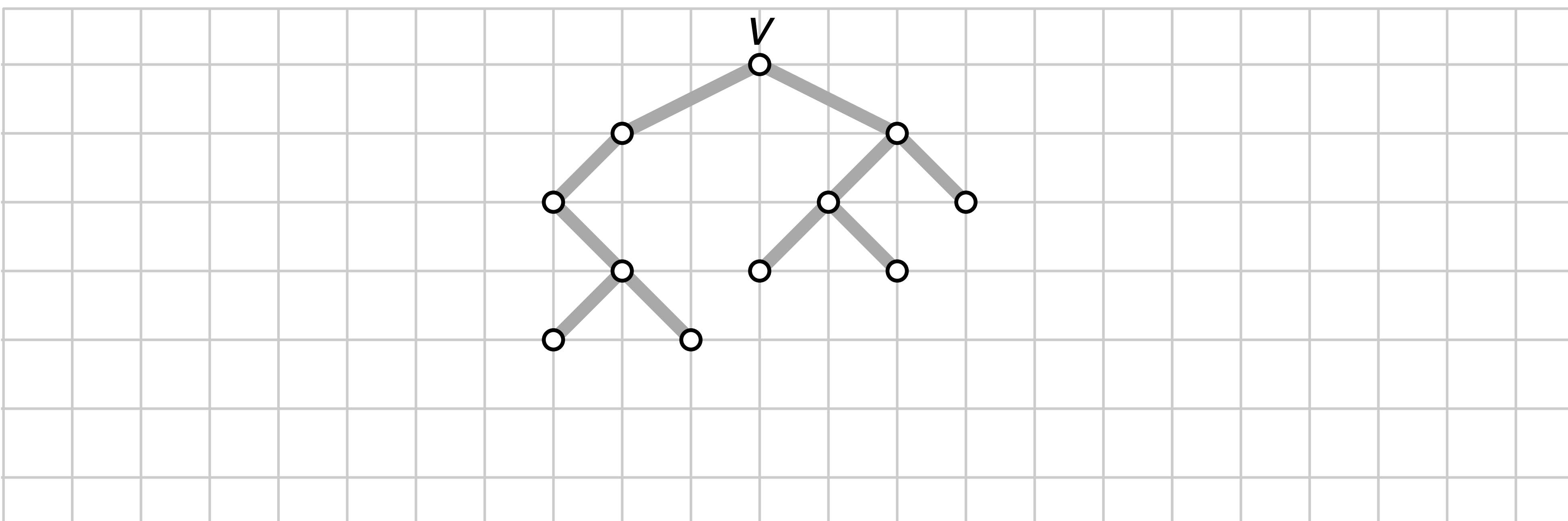
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates

Asymptotic time complexity:



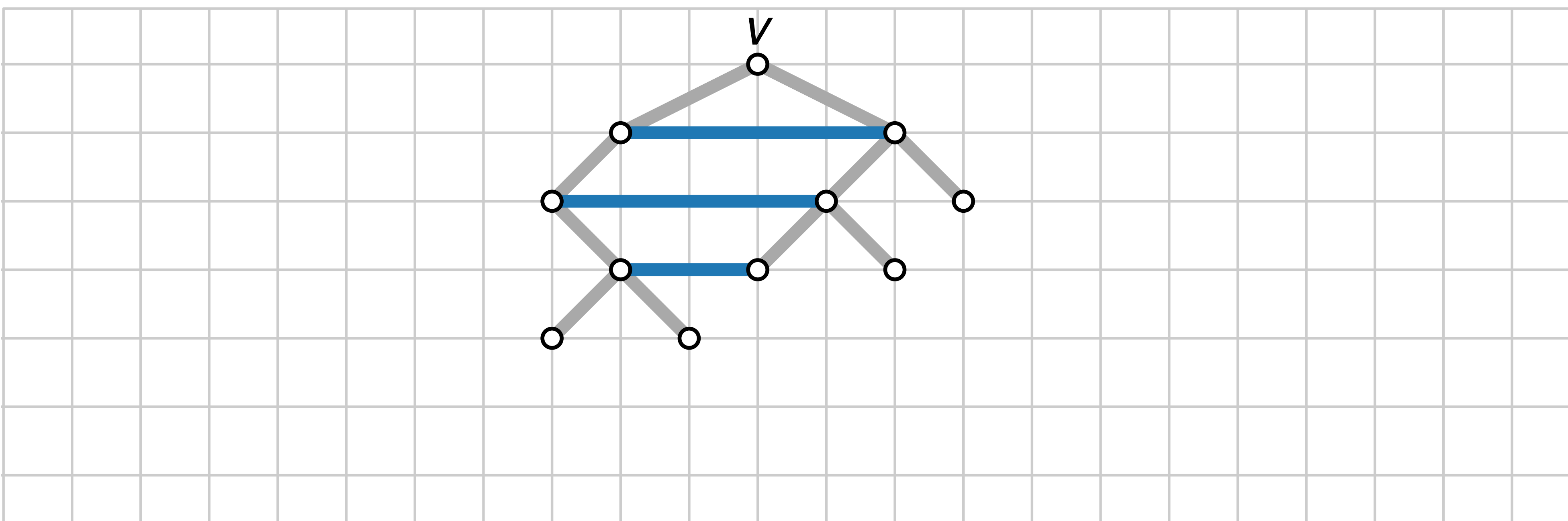
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates

Asymptotic time complexity:



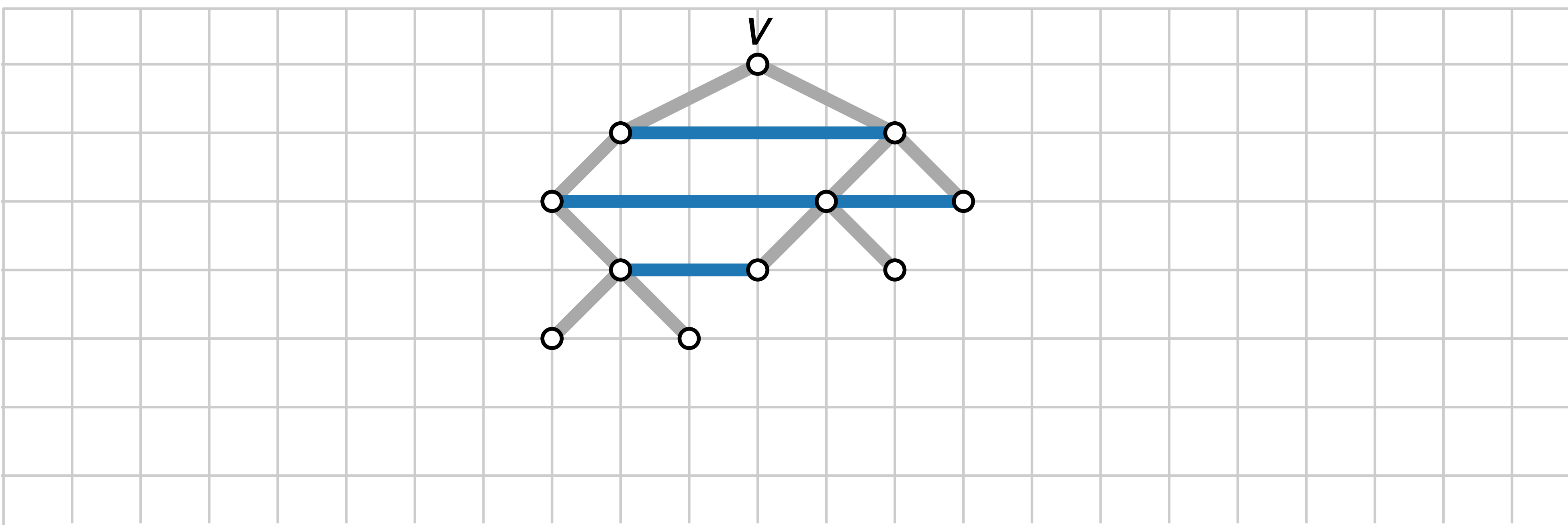
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates

Asymptotic time complexity:



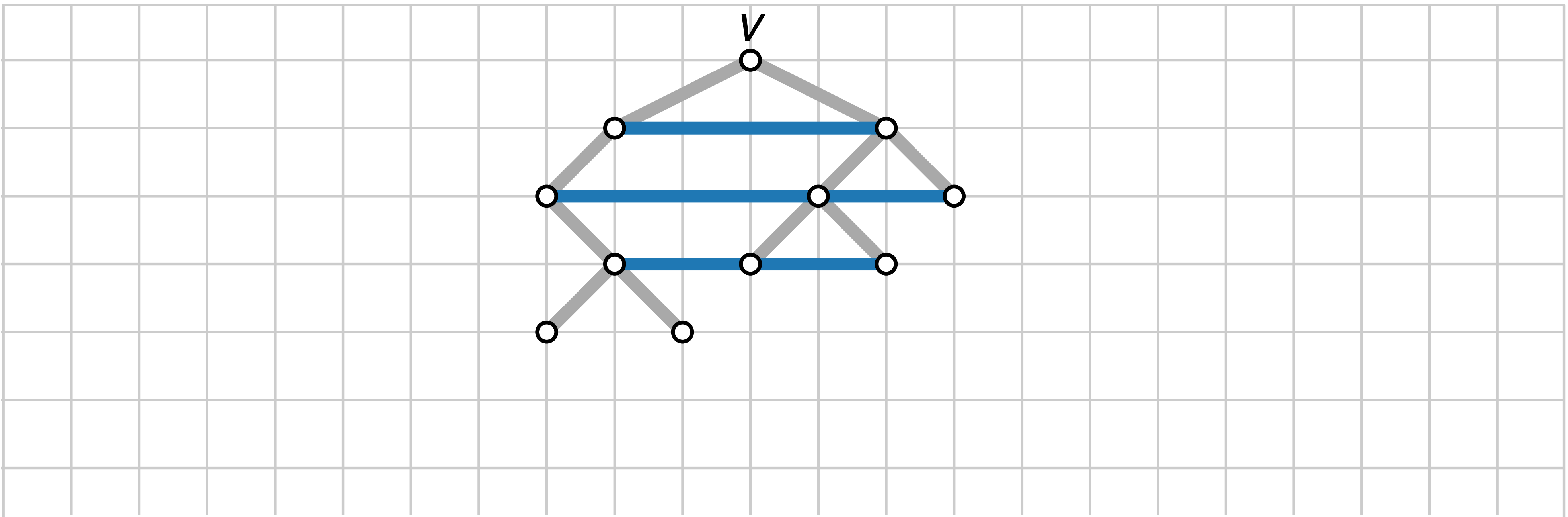
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates

Asymptotic time complexity:



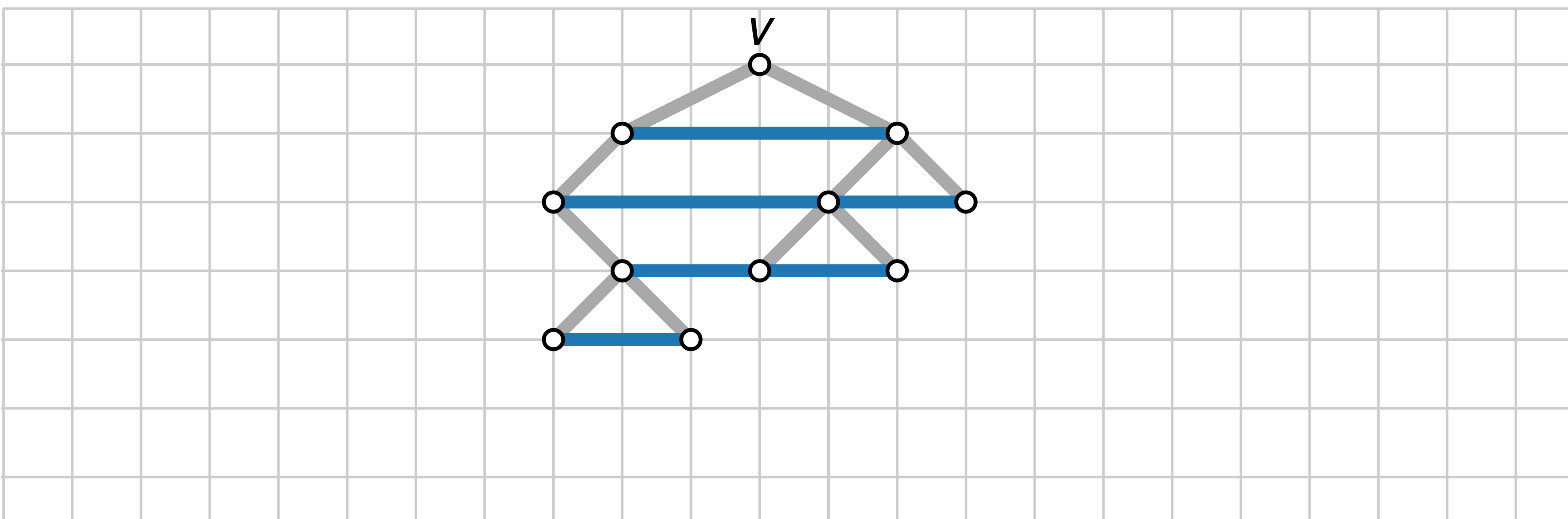
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates

Asymptotic time complexity:



Level-based Layout

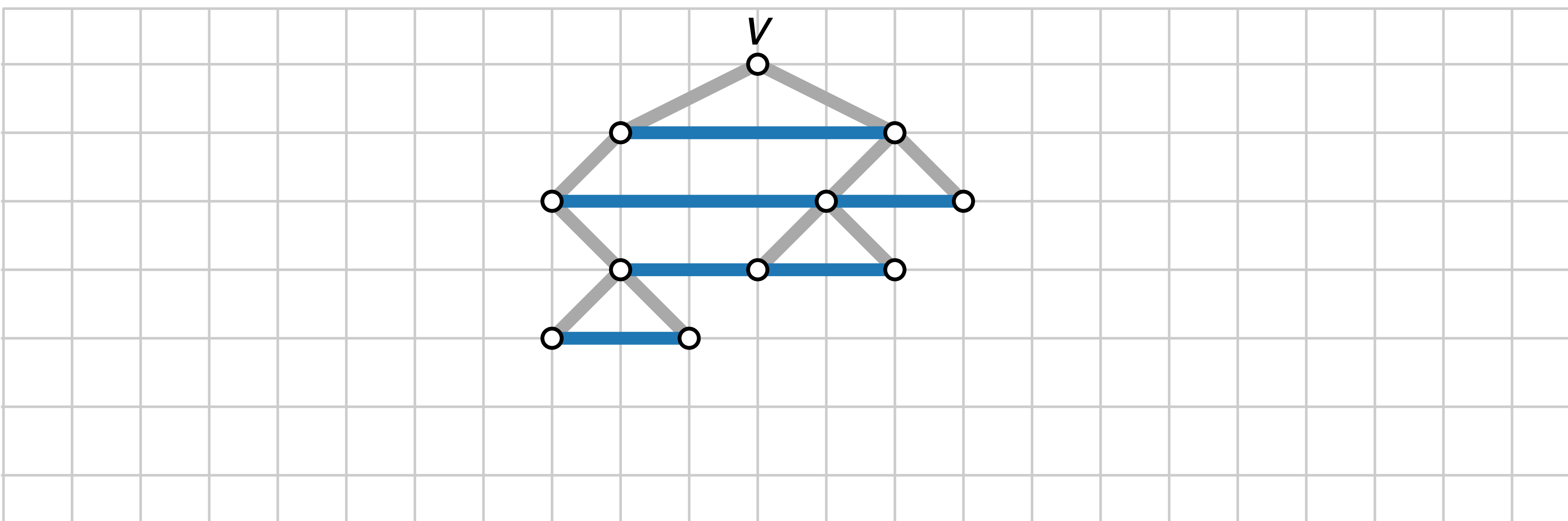
Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates

Asymptotic time complexity:

The overall procedure of summing up horizontal displacements is $O(n)$



Level-based Layout

Implementation Details (postorder and preorder traversals)

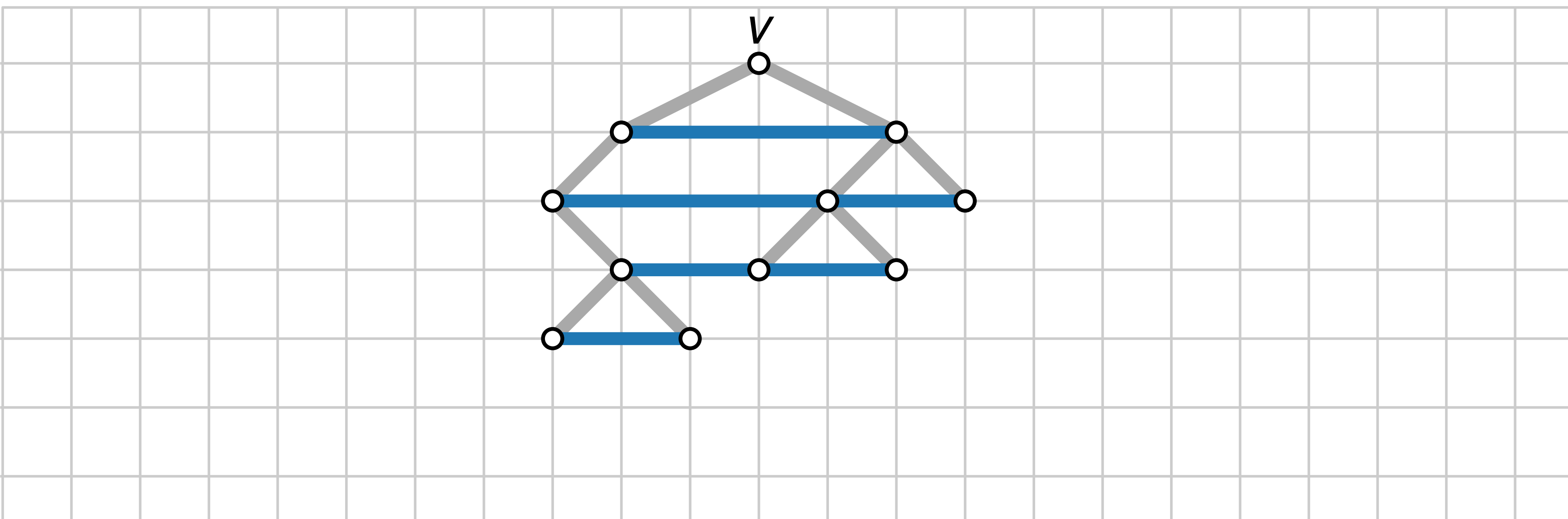
Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates

Asymptotic time complexity:

The overall procedure of summing up horizontal displacements is $O(n)$

Since both preorder and postorder are also $O(n)$, we need $O(n)$ in total



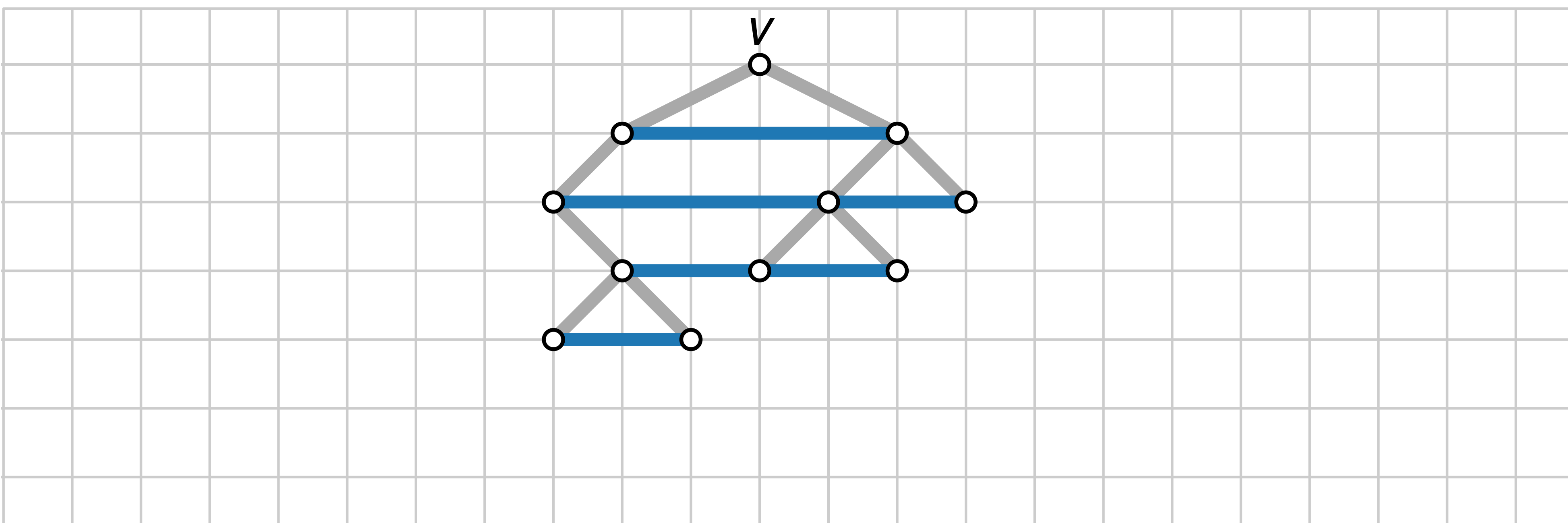
Level-based Layout

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacements of the left and the right child

Preorder traversal: Compute x- and y- coordinates

Asymptotic time complexity:



Level-based Layout

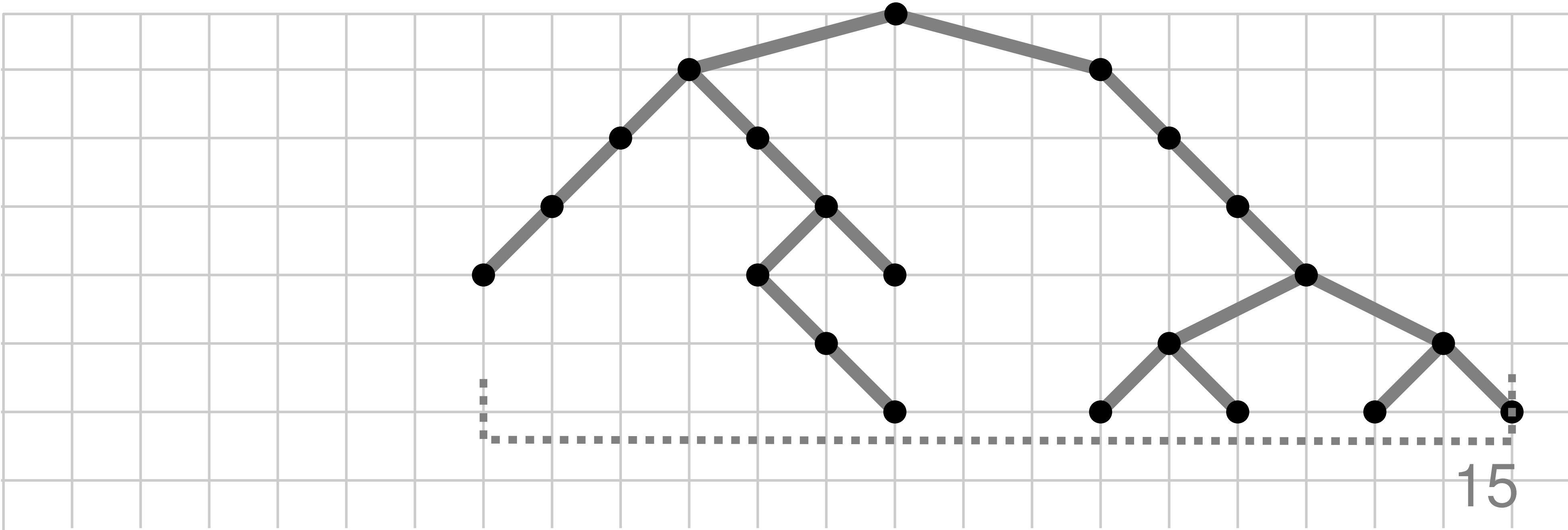
Theorem

Let T be a binary tree with n vertices. Algorithm of Reingold &Tilford constructs a drawing Γ of T in $O(n)$ time, such that:

- Γ is planar and straight-line
- $\forall v \in T$ y-coordinate of v is -depth(v)
- Vertical and horizontal distance is at least 1
- Area of Γ is $O(n^2)$
- Each vertex is centered with respect to its children
- Isomorphic trees have coincident drawings up to translation and reflection

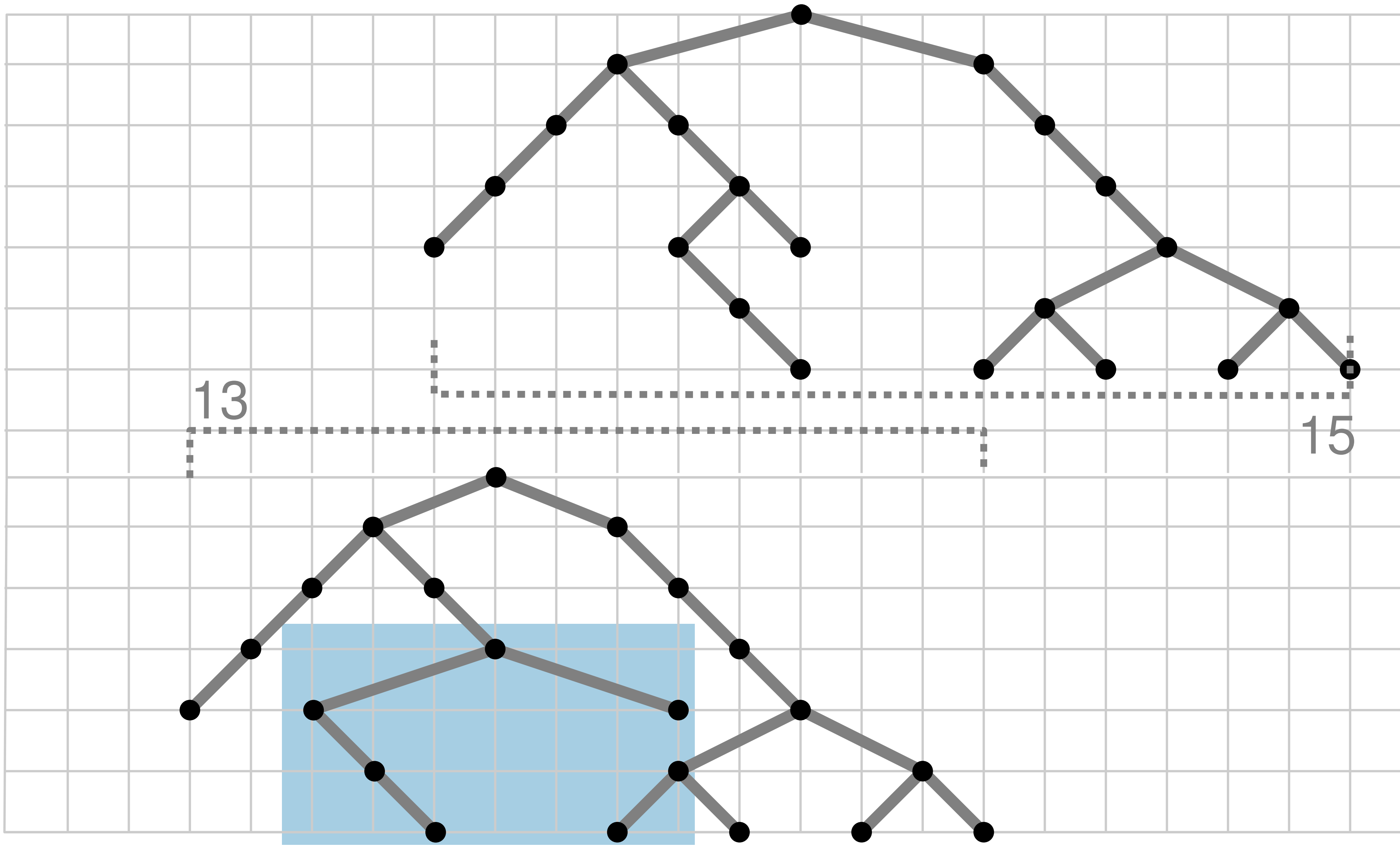
Level-based Layout

The presented algorithm tries to minimize width, does it achieve the minimum width?



Level-based Layout

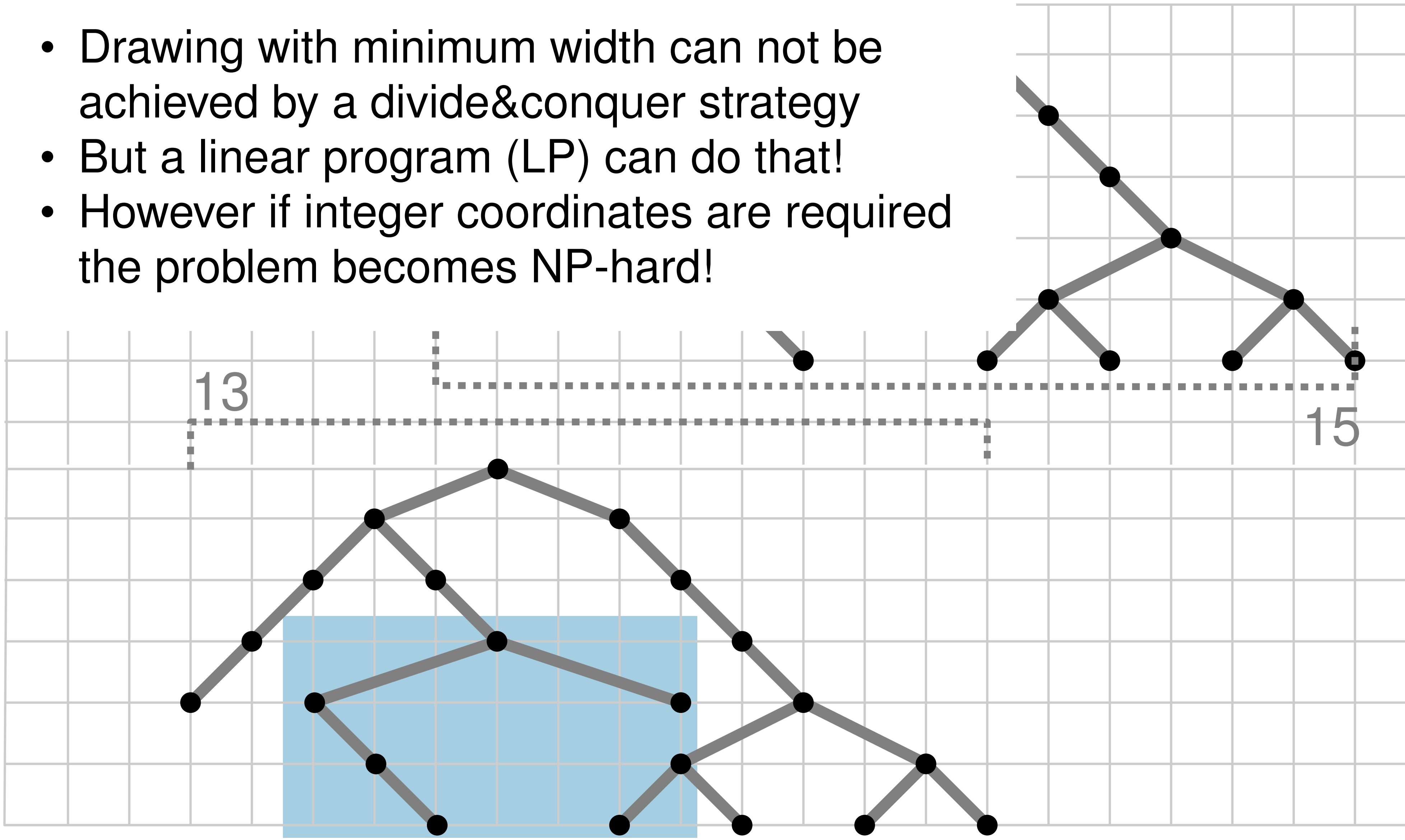
The presented algorithm tries to minimize width, does it achieve the minimum width?



Level-based Layout

The presented algorithm tries to minimize width, does it achieve the minimum width?

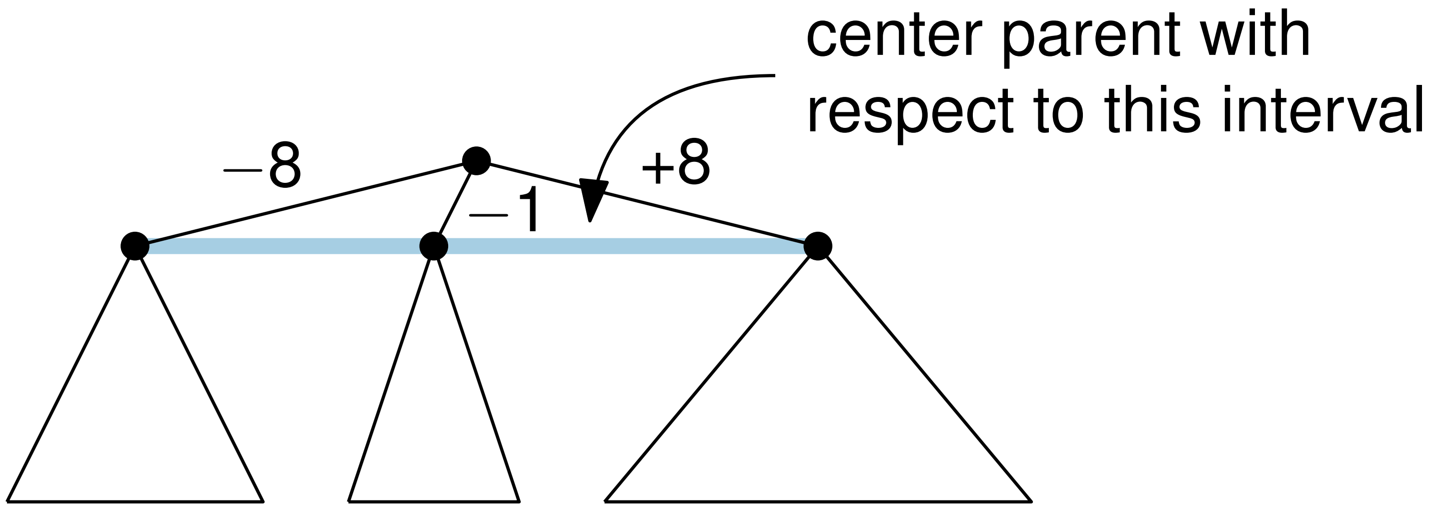
- Drawing with minimum width can not be achieved by a divide&conquer strategy
- But a linear program (LP) can do that!
- However if integer coordinates are required the problem becomes NP-hard!



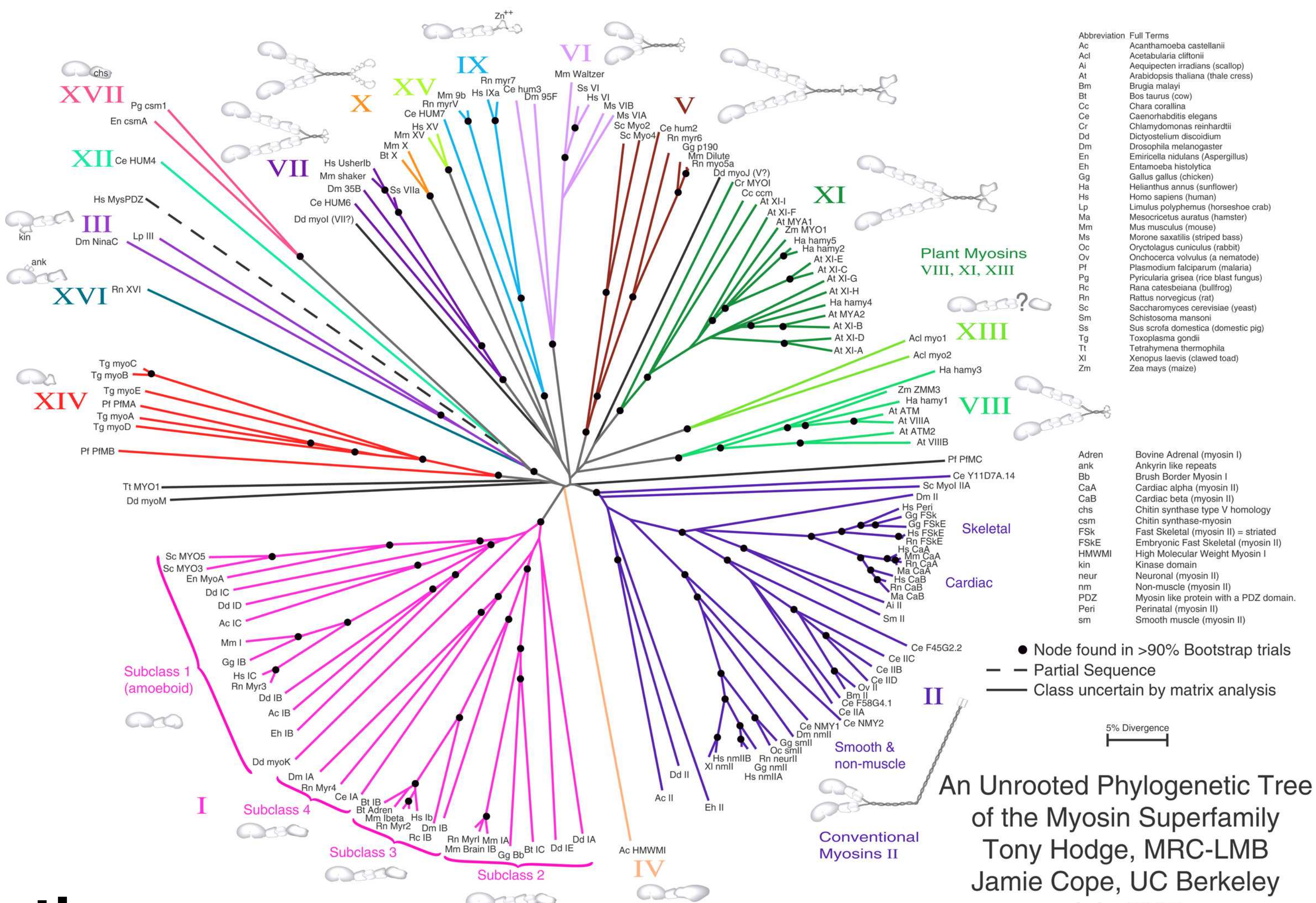
Level-based Layout

Note: We discussed an algorithm for binary trees. Your task is to generalize this to general trees!

Implementation Details (postorder and preorder traversals)
Postorder traversal: For each vertex v compute horizontal displacements of all the children



Radial Layout

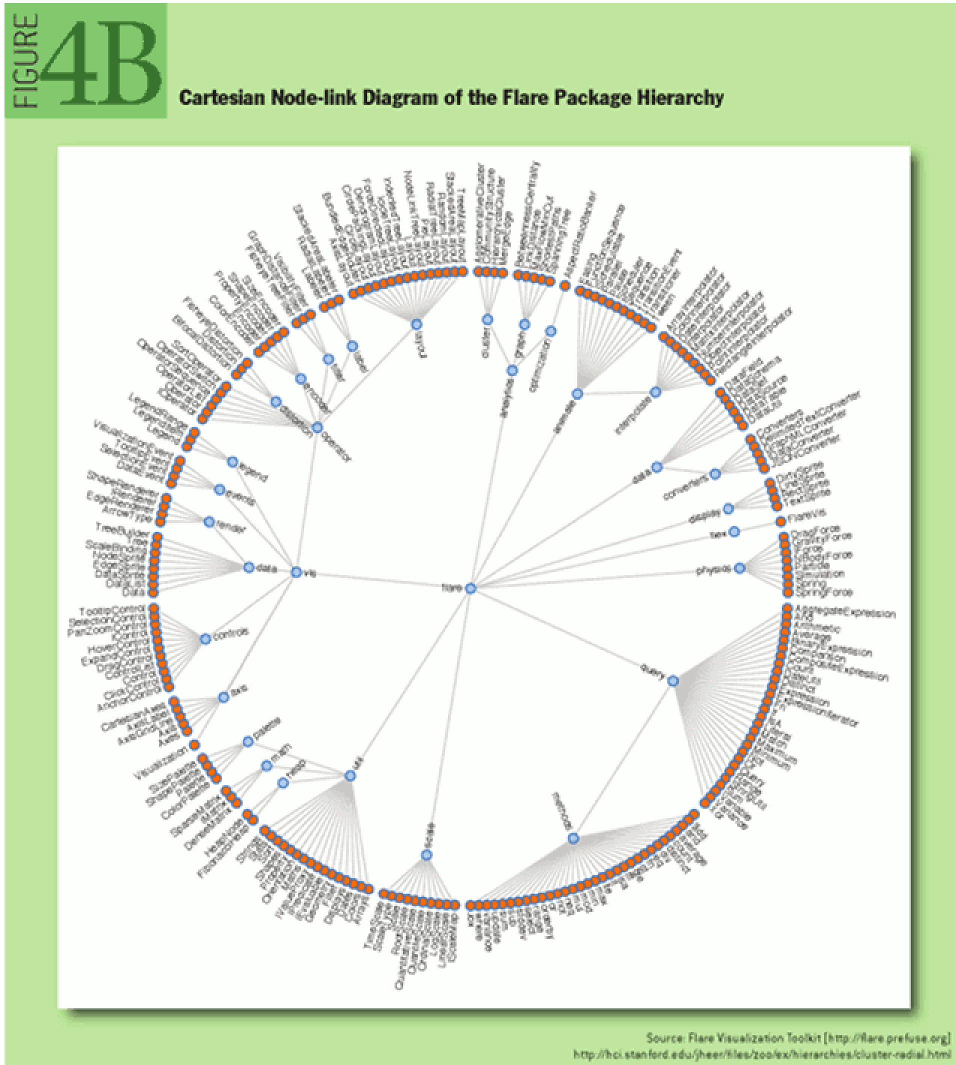


Application

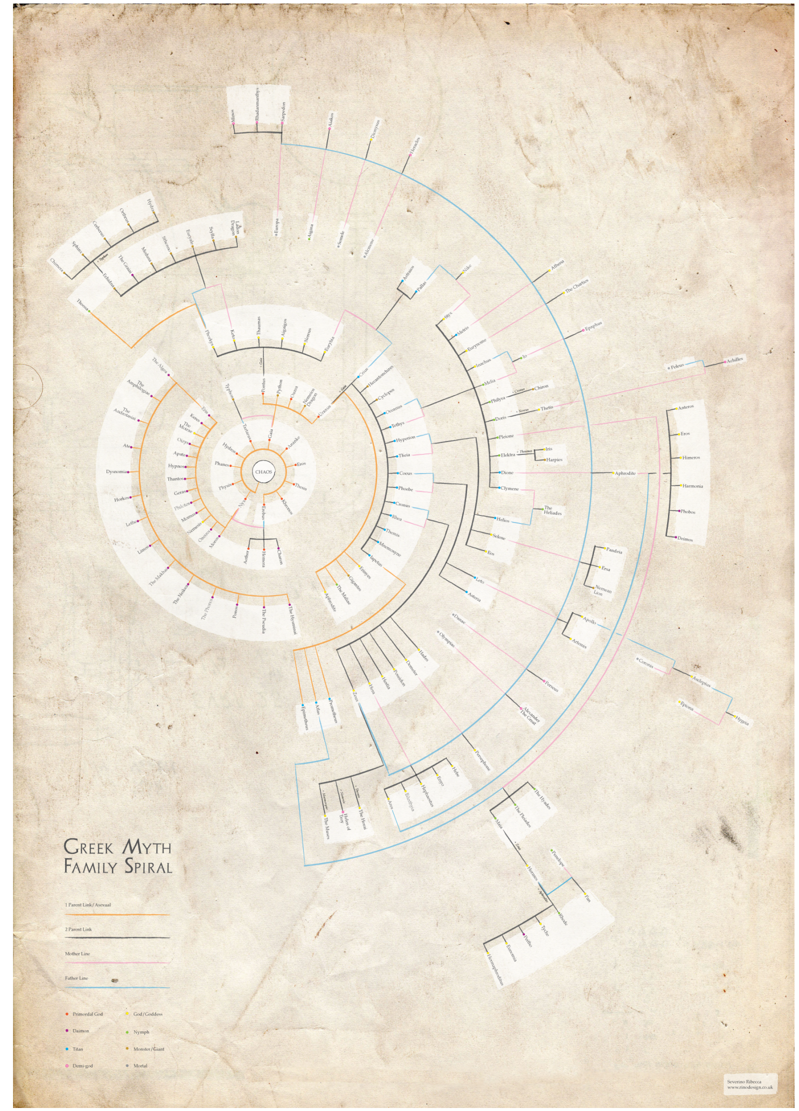
An unrooted phylogenetic tree for myosin, a superfamily of proteins.

”A myosin family tree“ *Journal of Cell Science*

Radial Layout

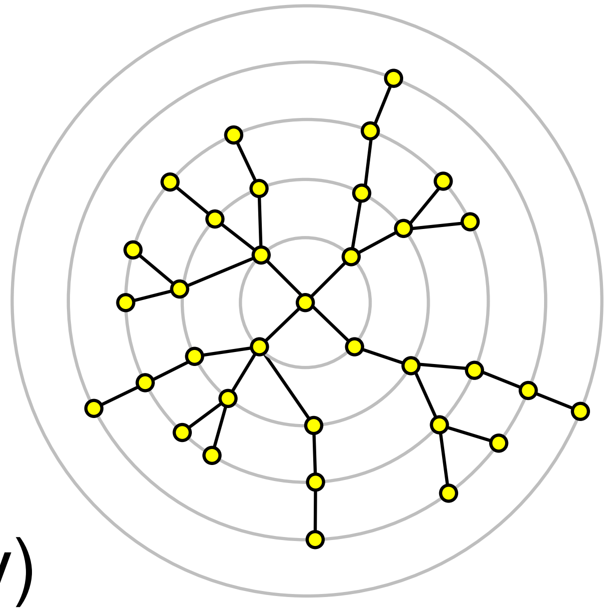


Flare Visualization Toolkit code structure by Heer, Bostock and Ogievetsky, 2010



Greek Myth Family by Ribecca, 2011

Radial Layout



Drawing Conventions:

- Vertices lie on circular layers according to their depth
- Drawing is planar

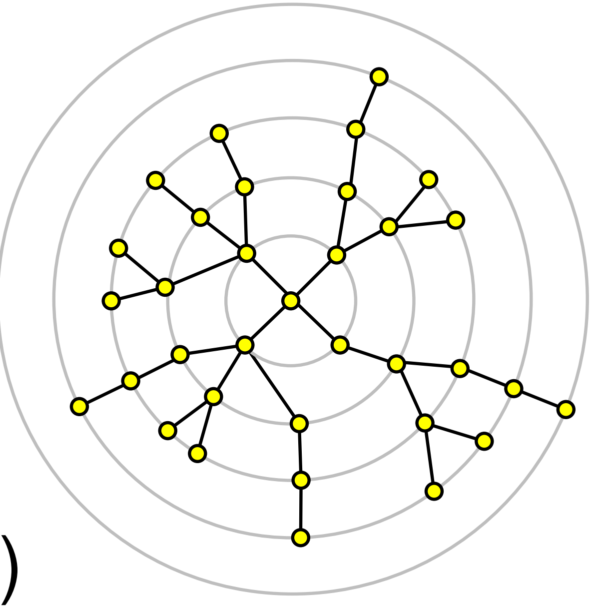
Quality Metrics:

- Distribution of the vertices (vaguely)

Radial Layout

Drawing Conventions:

- Vertices lie on circular layers according to their depth
- Drawing is planar



Quality Metrics:

- Distribution of the vertices (vaguely)

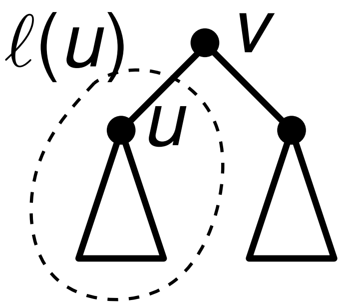
Take a minute to think about a possible algorithm to optimize the distribution of the vertices

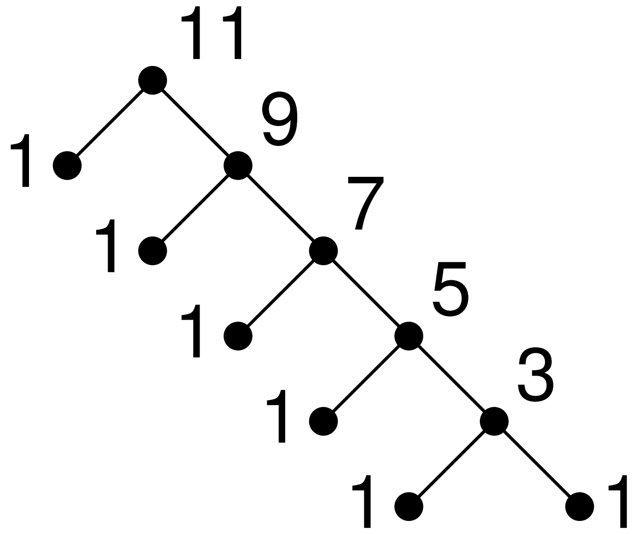
Radial Layout

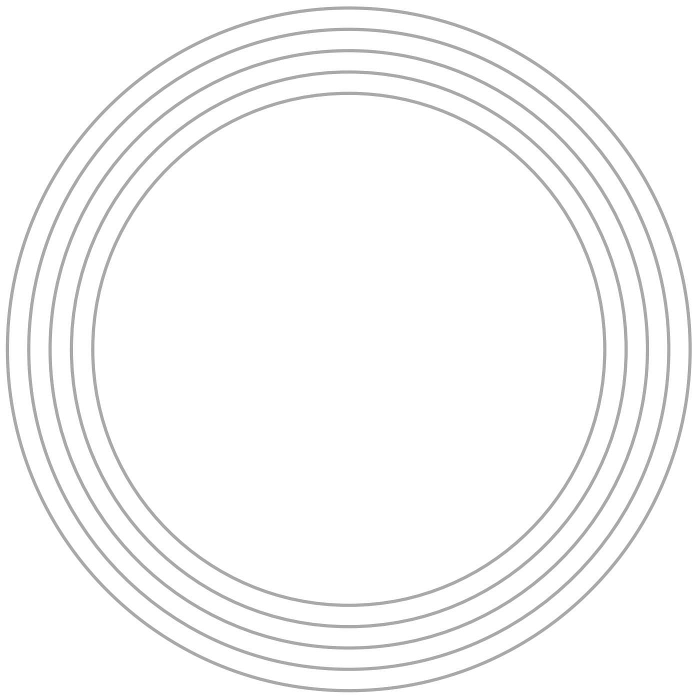
Example:

- Angle corresponding to the subtree rooted at *u*:

  $\tau_u = \frac{\ell(u)}{\ell(v)-1}$



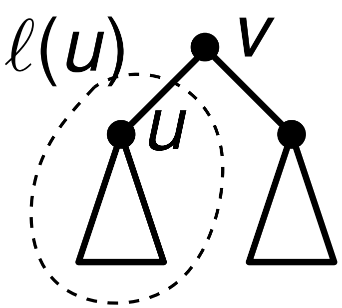


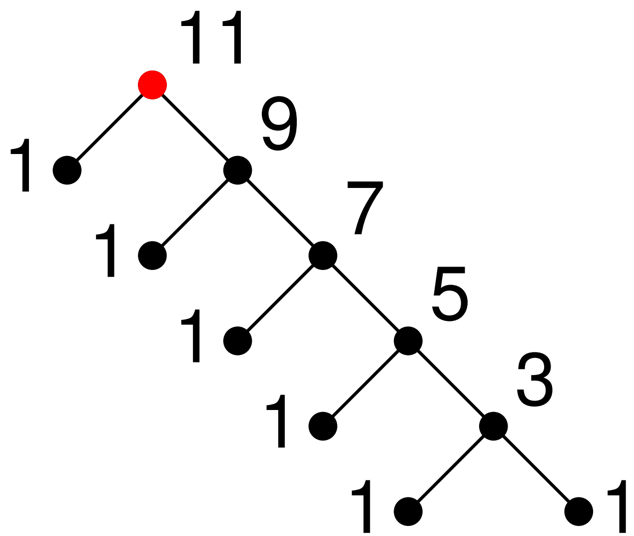


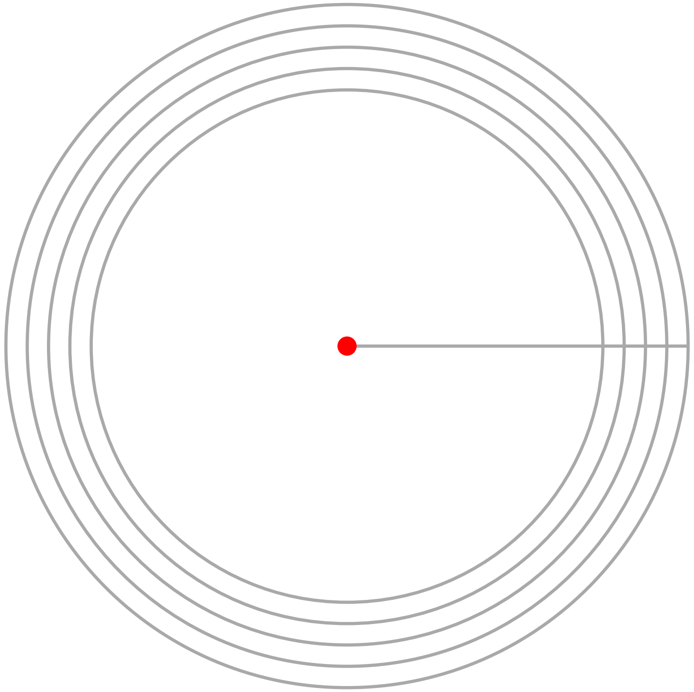
Radial Layout

Example:

- Angle corresponding to the subtree rooted at u:
  $\tau_u = \frac{\ell(u)}{\ell(v)-1}$





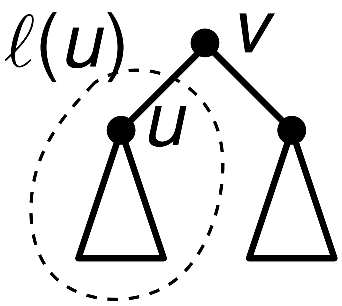


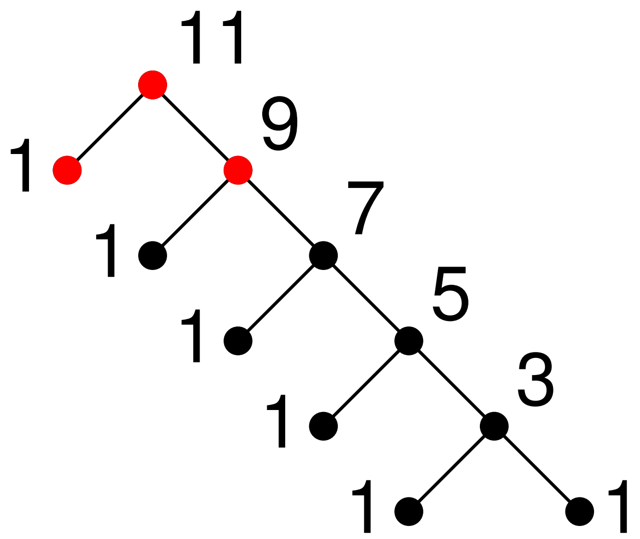
Radial Layout

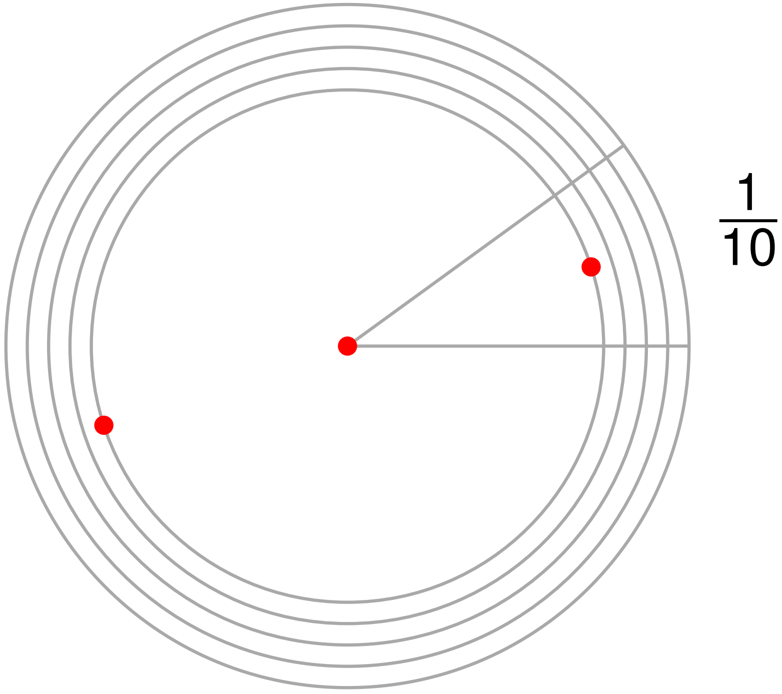
Example:

- Angle corresponding to the subtree rooted at *u*:

  $\tau_u = \frac{\ell(u)}{\ell(v)-1}$





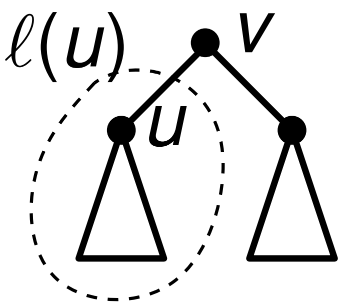


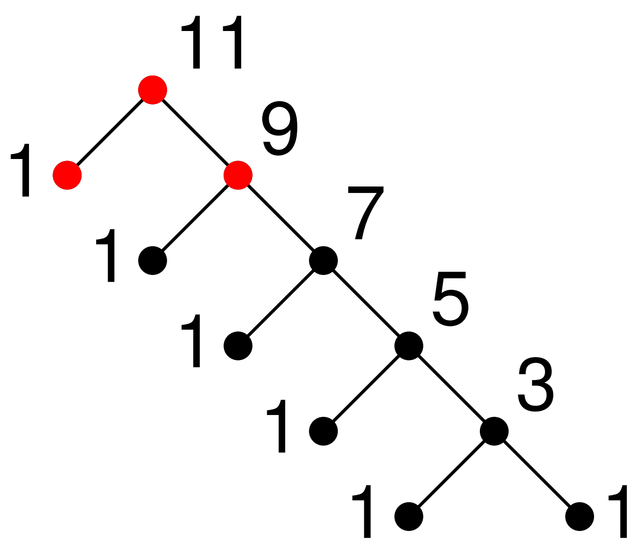
Radial Layout

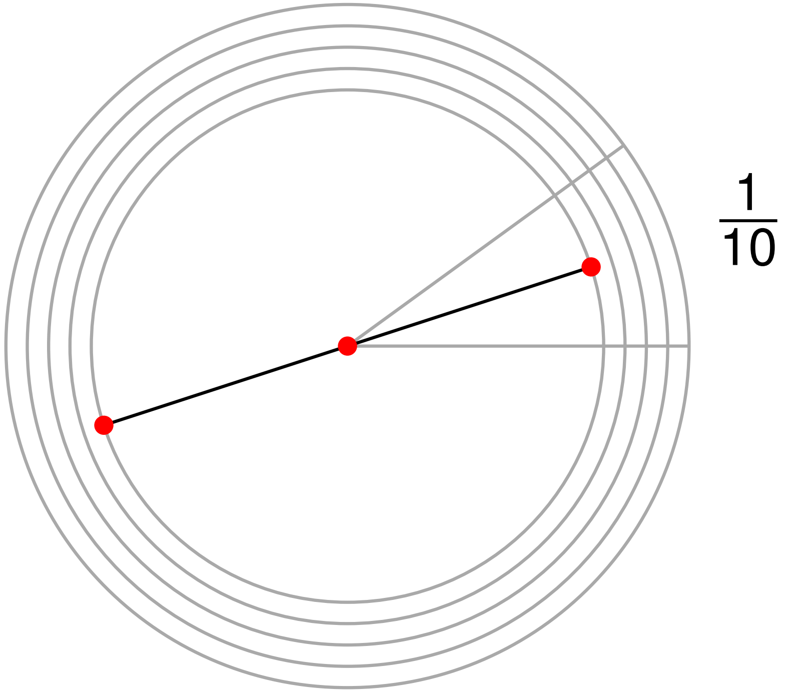
Example:

- Angle corresponding to the subtree rooted at u:

$$\tau_u = \frac{\ell(u)}{\ell(v)-1}$$





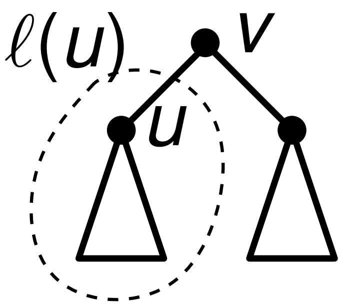


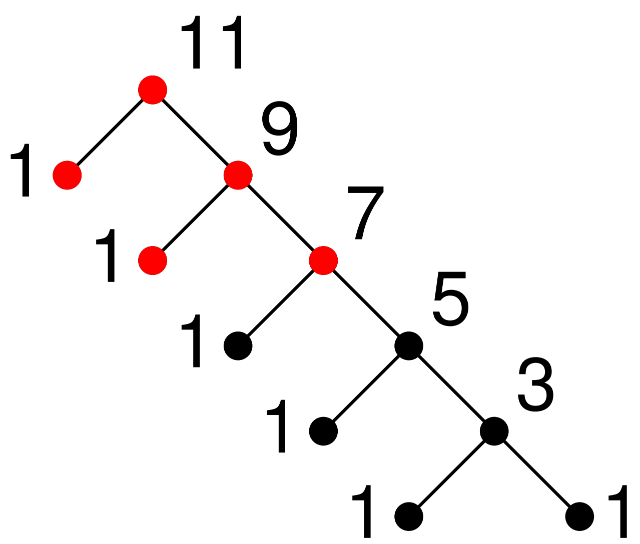
Radial Layout

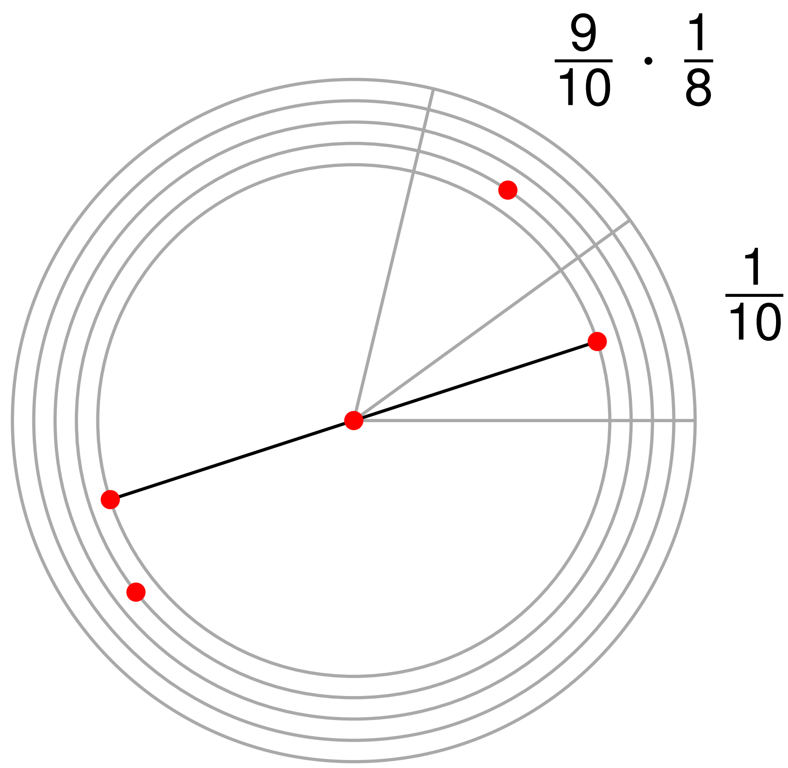
Example:

- Angle corresponding to the subtree rooted at u:

  $\tau_u = \frac{\ell(u)}{\ell(v)-1}$





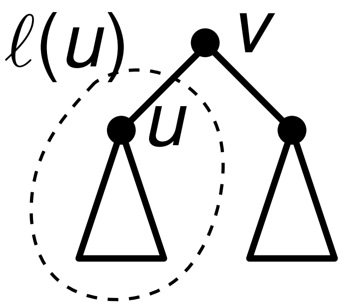


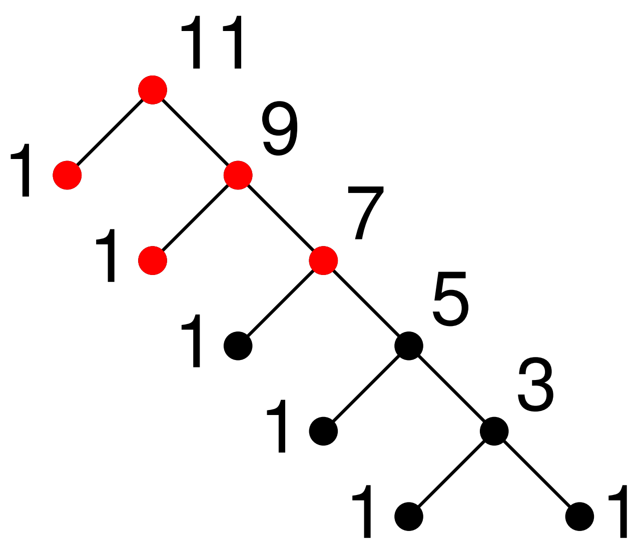
Radial Layout

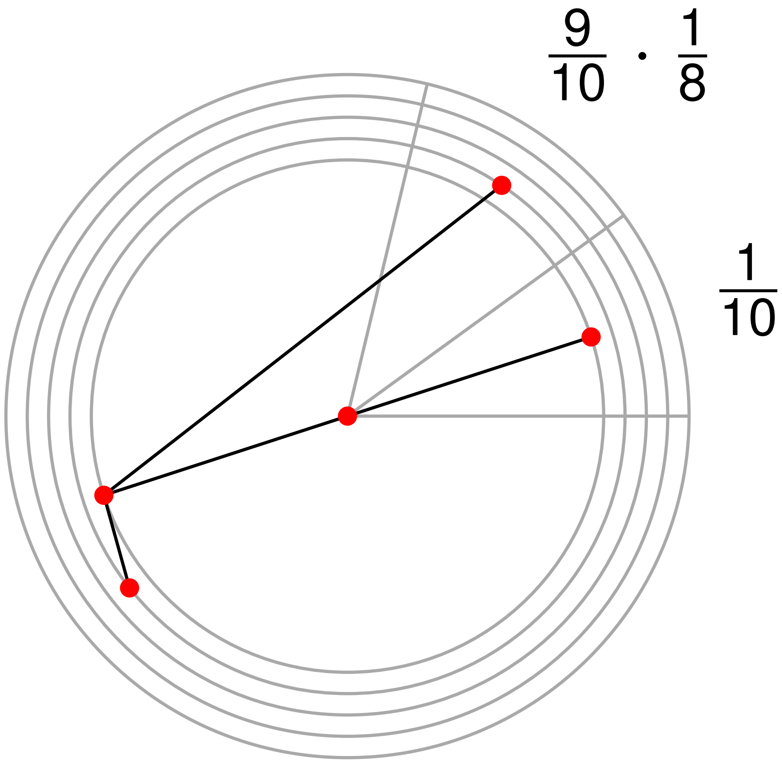
Example:

- Angle corresponding to the subtree rooted at u:

  $\tau_u = \frac{\ell(u)}{\ell(v)-1}$



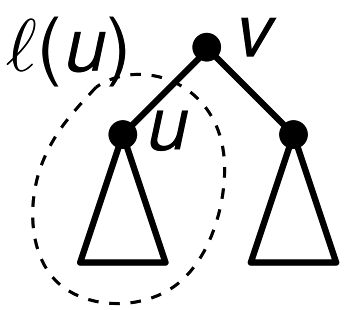


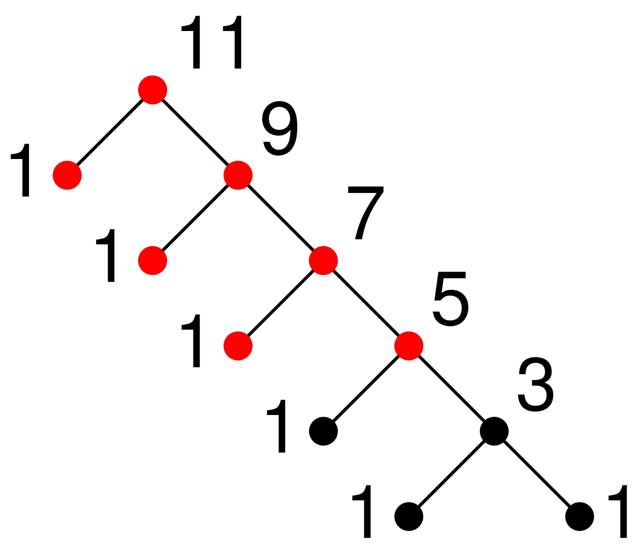


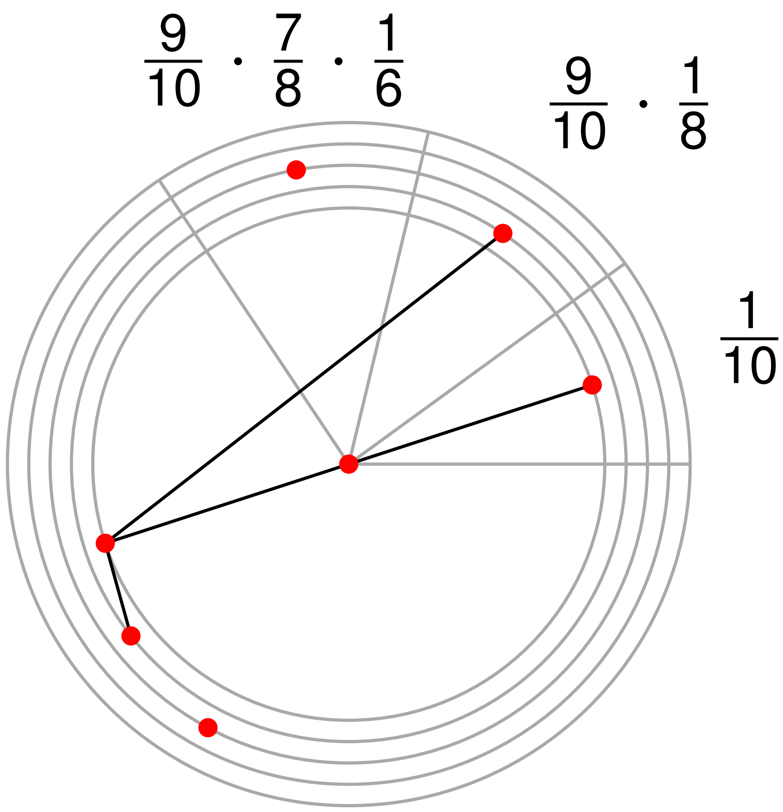
Radial Layout

Example:

- Angle corresponding to the subtree rooted at u:
  $\tau_u = \frac{\ell(u)}{\ell(v)-1}$





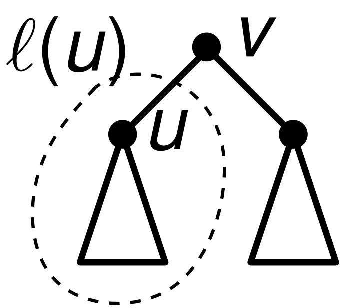


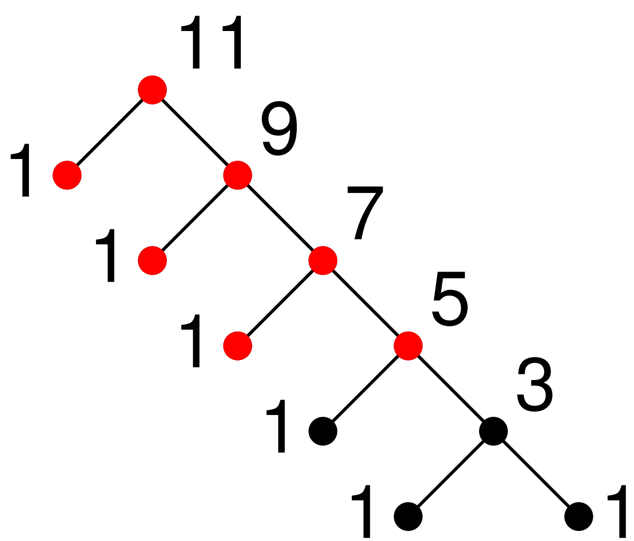
Radial Layout

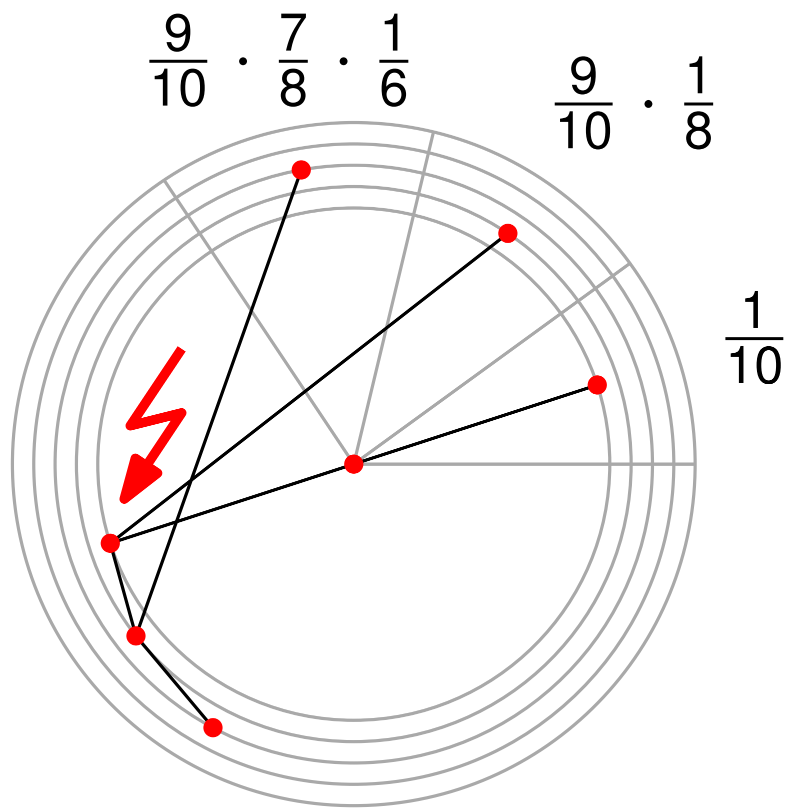
Example:

- Angle corresponding to the subtree rooted at u:

  $\tau_u = \frac{\ell(u)}{\ell(v)-1}$

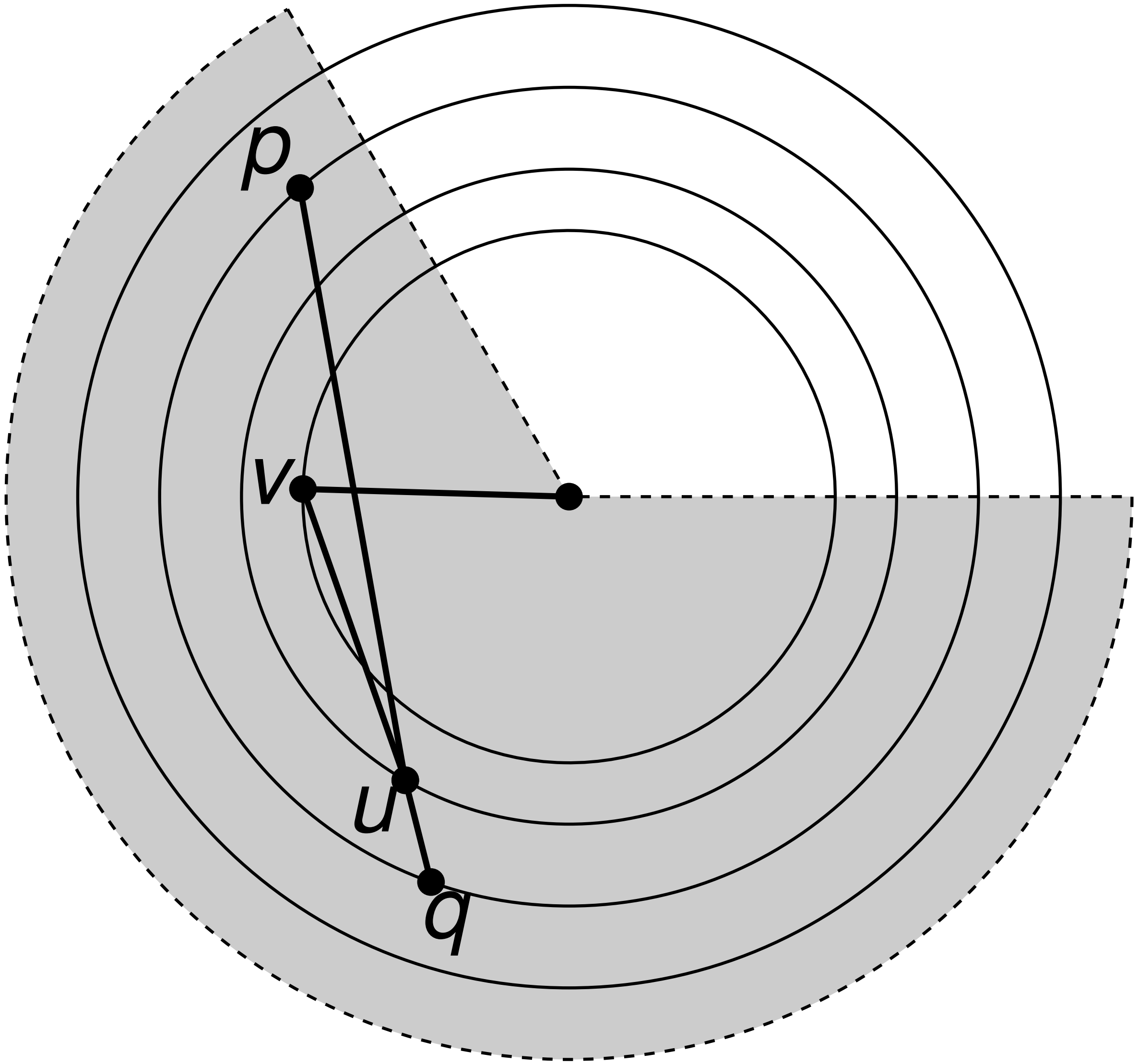






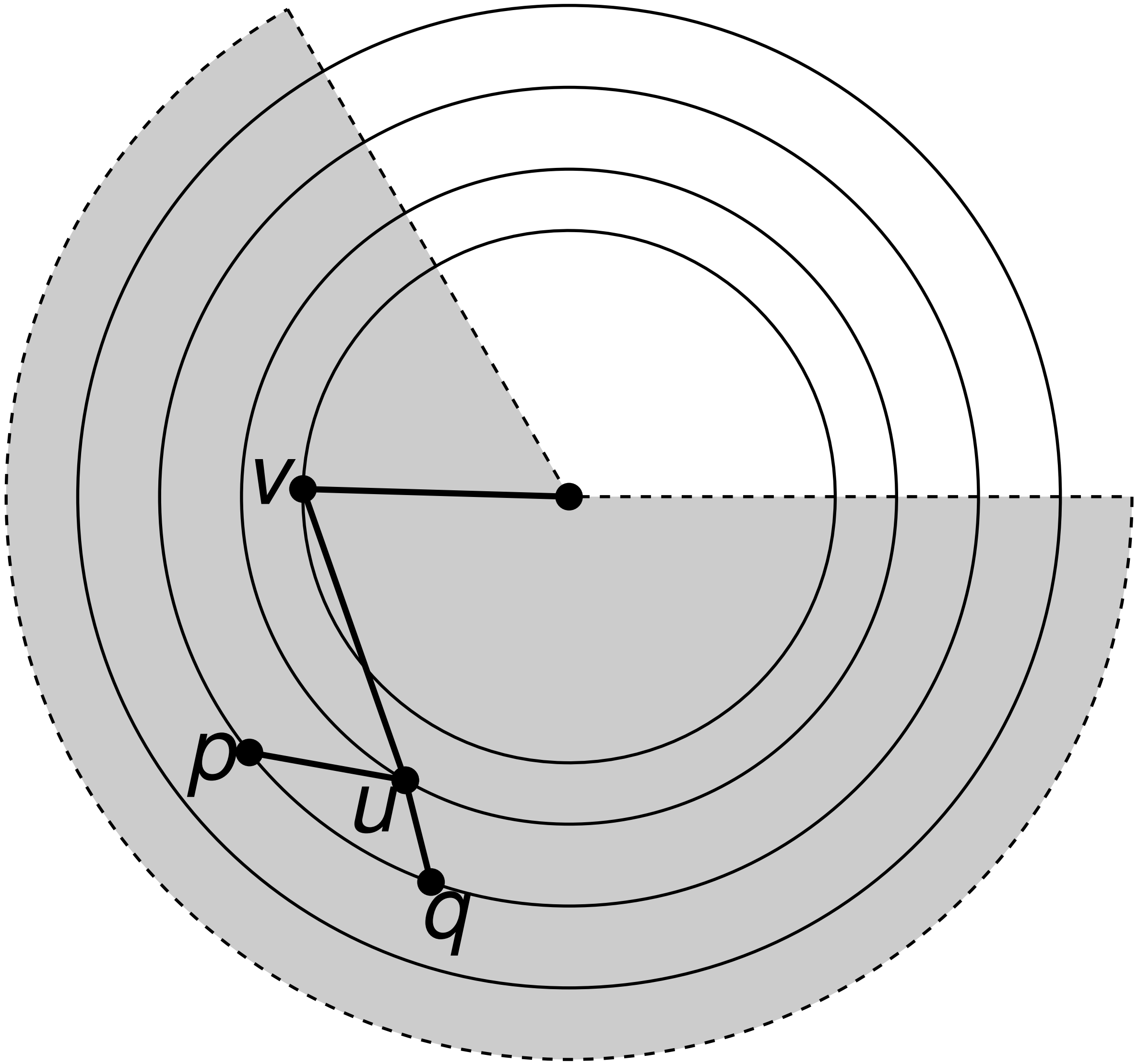
Radial Layout

How to avoid crossings:



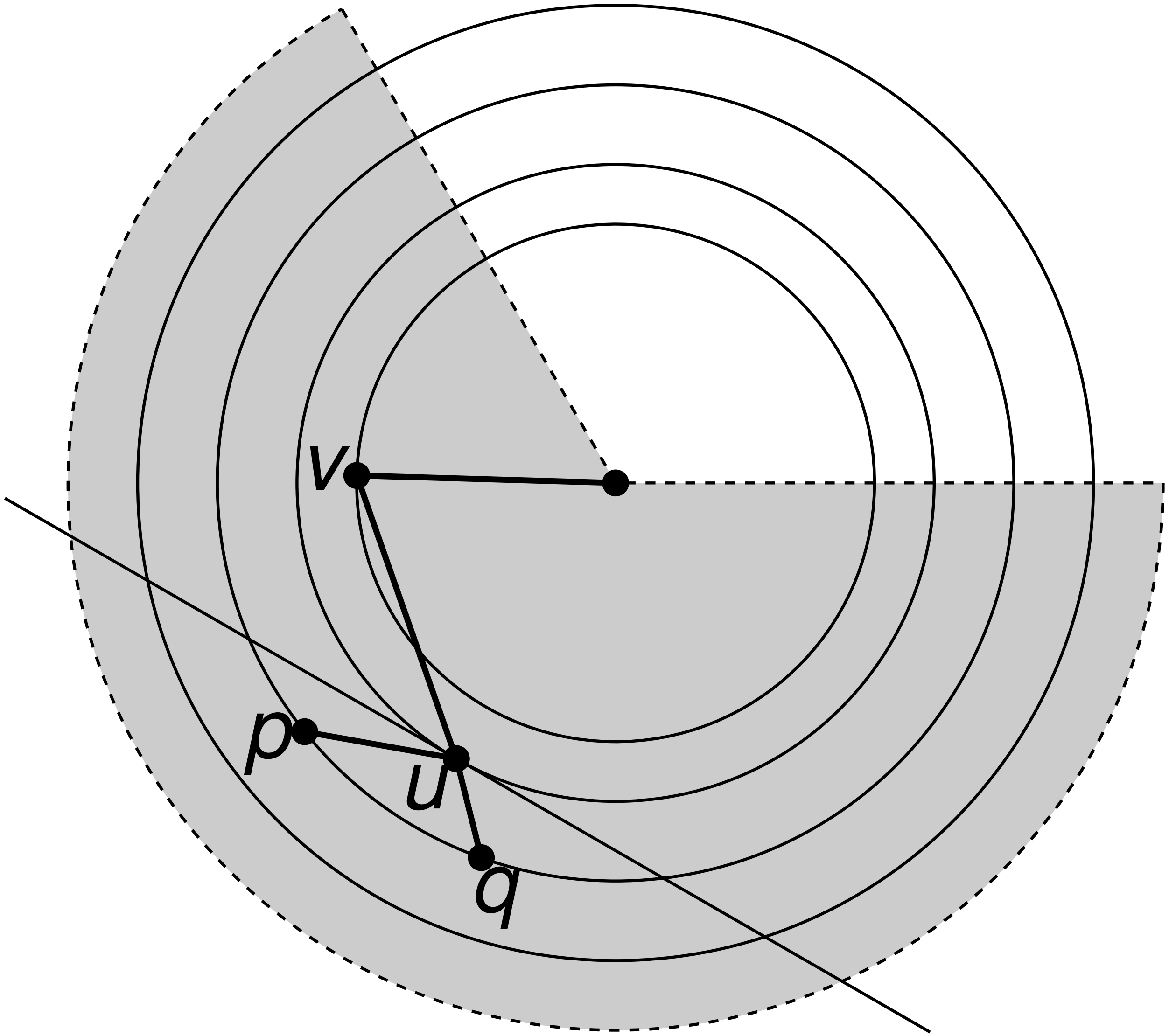
Radial Layout

How to avoid crossings:



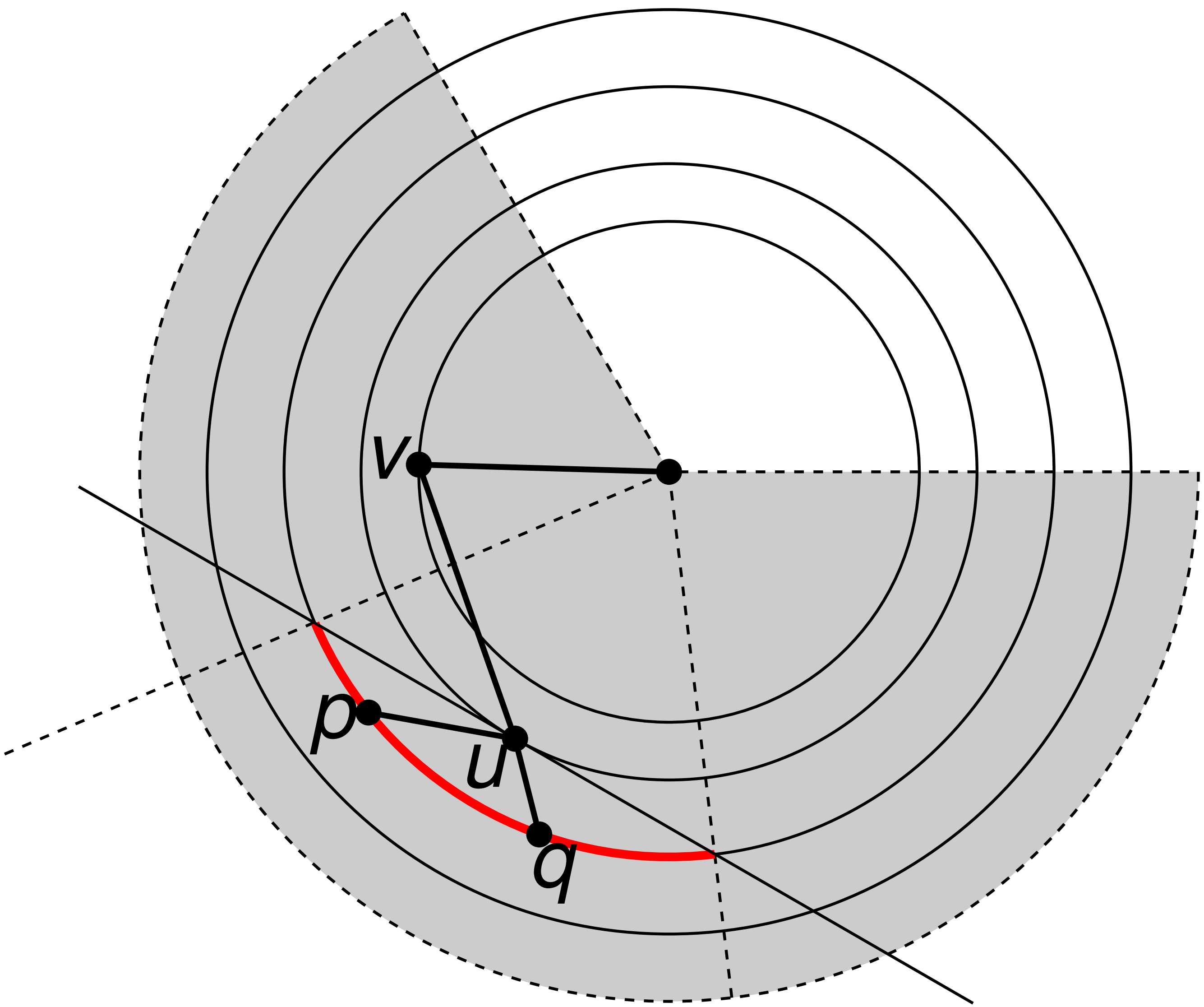
Radial Layout

How to avoid crossings:



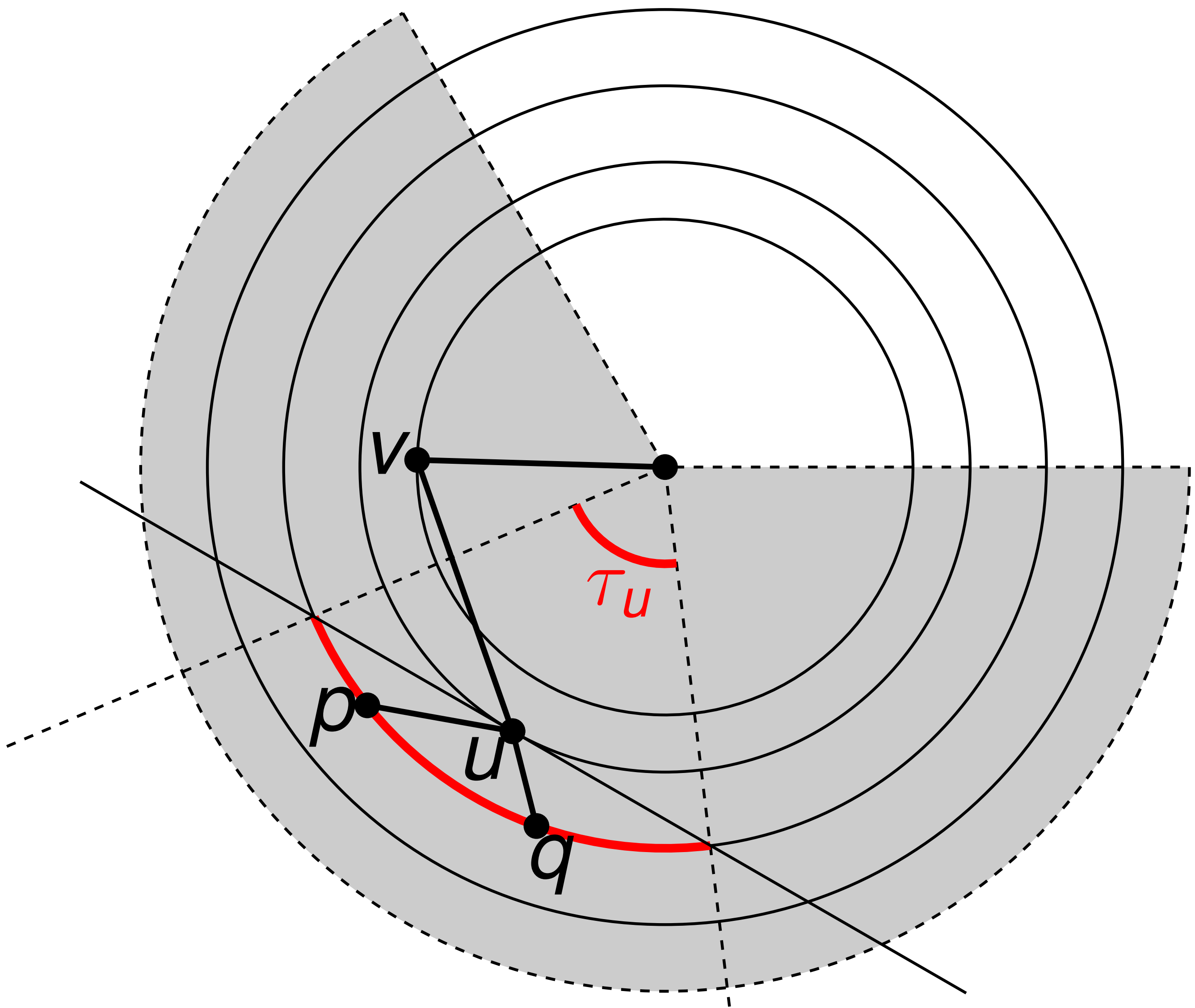
Radial Layout

How to avoid crossings:



Radial Layout

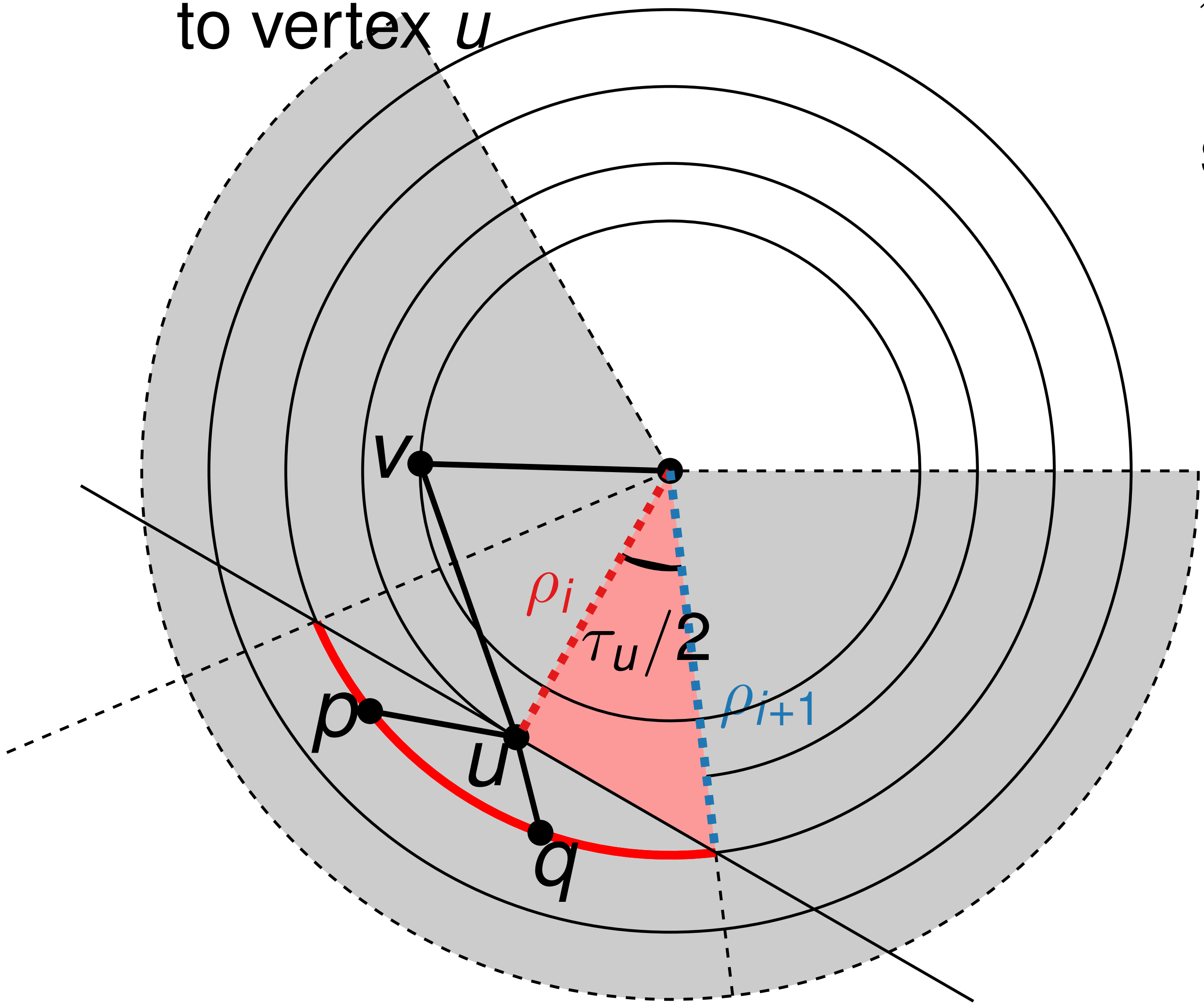
How to avoid crossings:



Radial Layout

How to avoid crossings:

- τ_u - angle of the wedge corresponding to vertex *u*



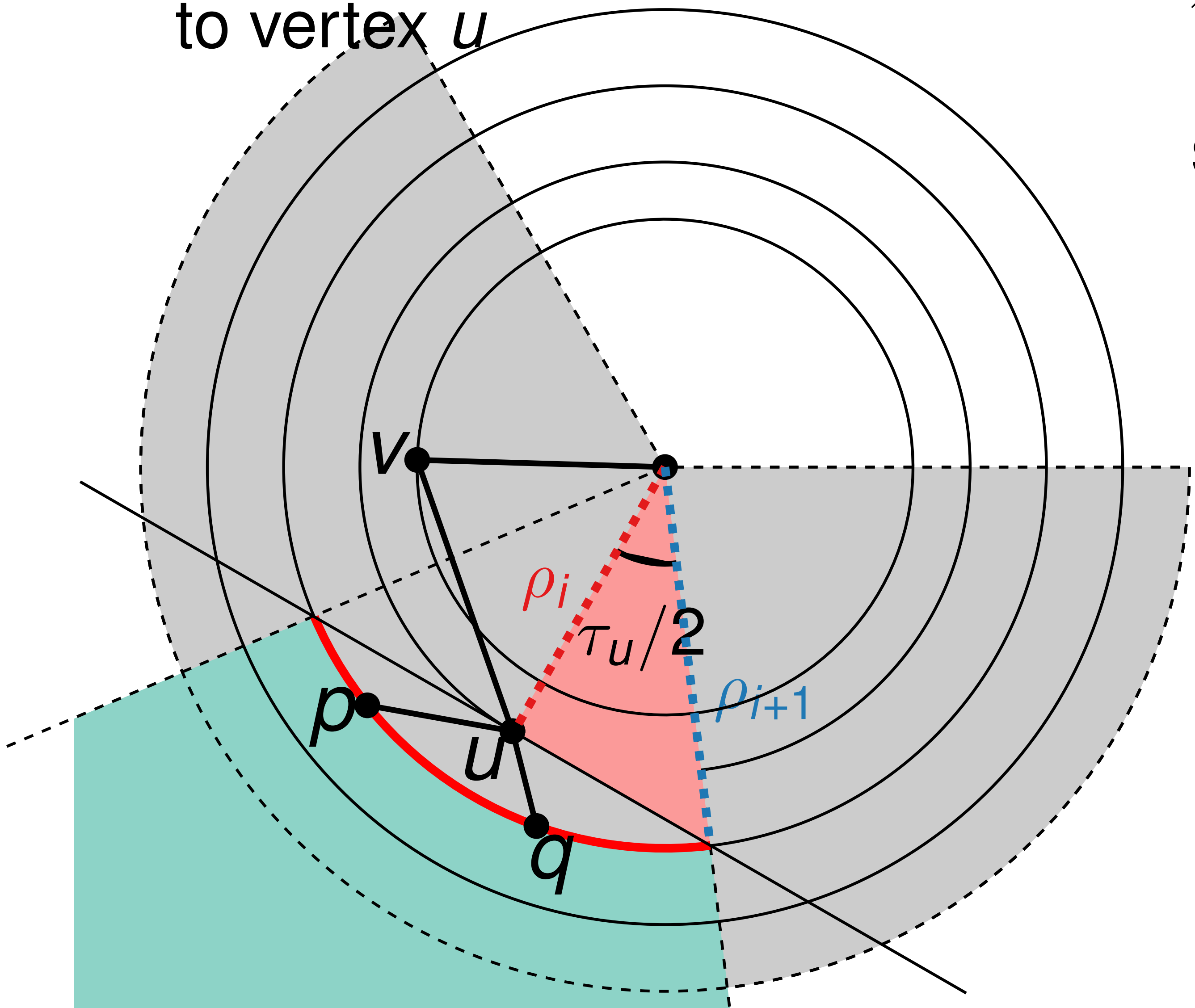
- ρ_i - raduis of layer *i*
  - $\ell(v)$-number of nodes in the subtree rooted at *v*
    - $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$

Radial Layout

How to avoid crossings:

- τ_u - angle of the wedge corresponding to vertex *u*

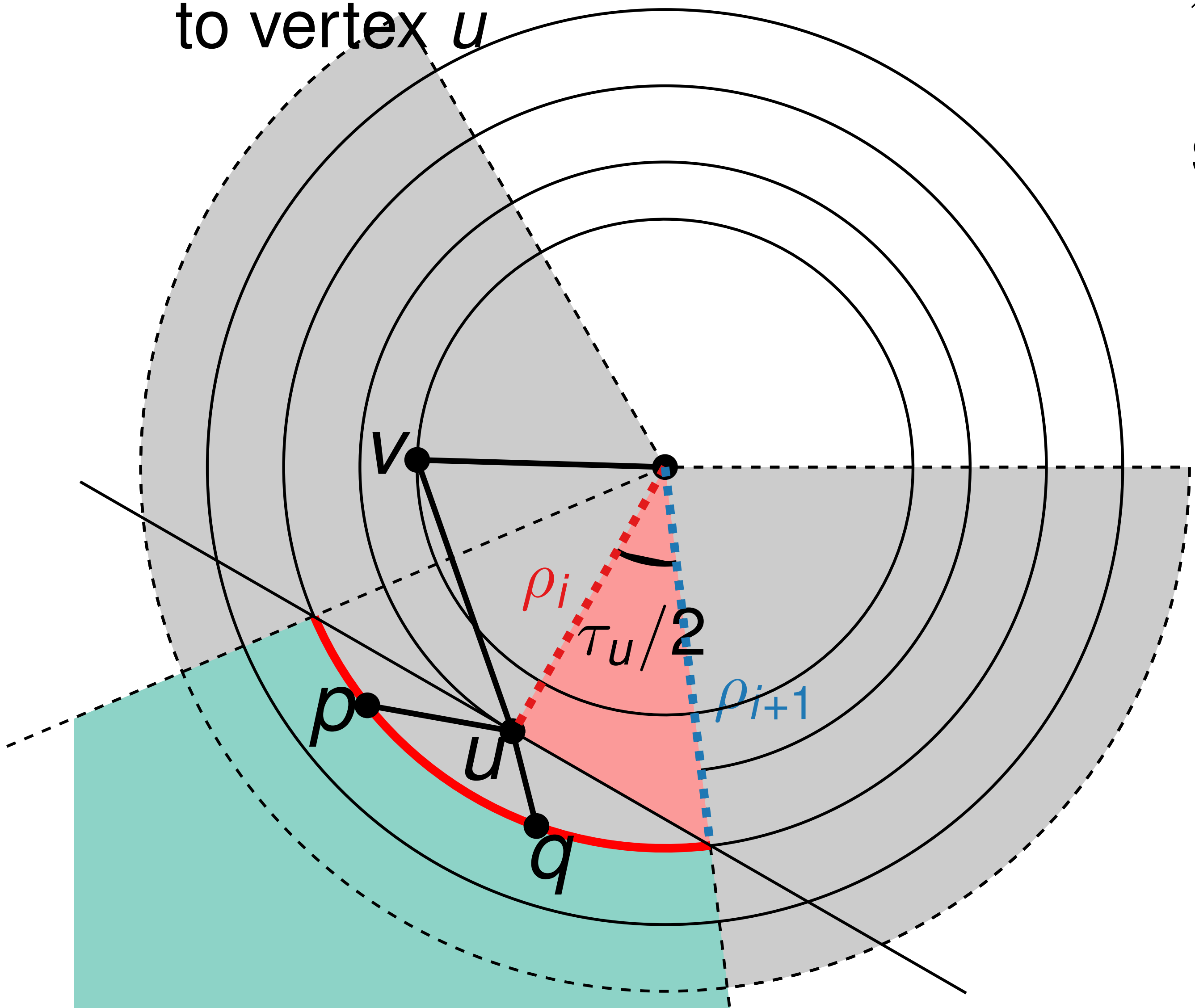
- ρ_i - raduis of layer *i*
- $\ell(v)$-number of nodes in the subtree rooted at *v*
  - $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$



Radial Layout

How to avoid crossings:

- τ_u - angle of the wedge corresponding to vertex u

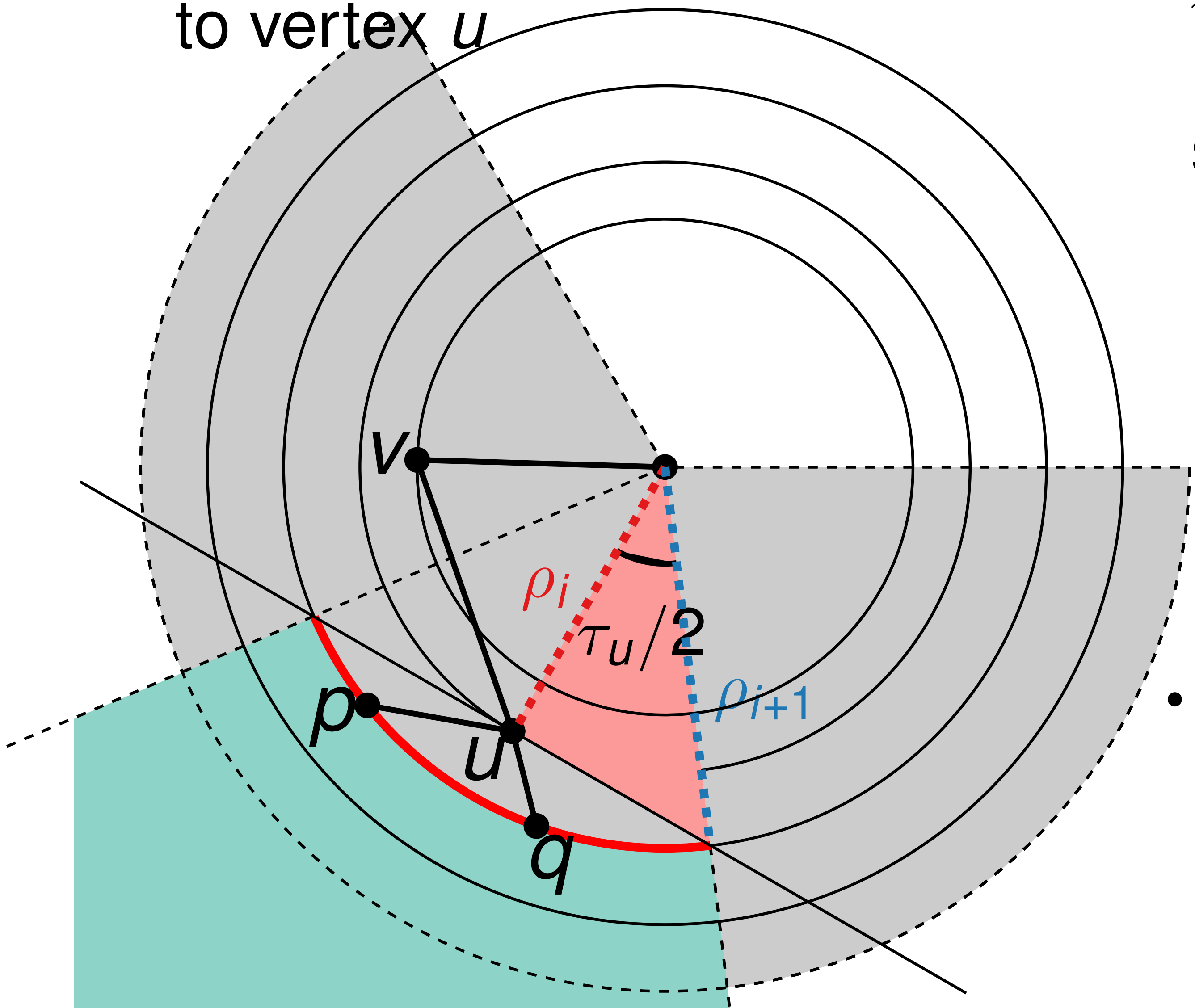


- ρ_i - raduis of layer i
  - $\ell(v)$-number of nodes in the subtree rooted at v
    - $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$
    - $\tau_u = \min\{\frac{\ell(u)}{\ell(v)-1}, 2\arccos\frac{\rho_i}{\rho_{i+1}}\}$ (correction)

Radial Layout

How to avoid crossings:

- τ_u - angle of the wedge corresponding to vertex u



- ρ_i - raduis of layer i
  - $\ell(v)$-number of nodes in the subtree rooted at v
    - $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$
    - $\tau_u = \min\{\frac{\ell(u)}{\ell(v)-1}, 2\arccos\frac{\rho_i}{\rho_{i+1}}\}$ (correction)
- Alternatively use number of leaves in the subtree to subdivide the angles

Radial Layout

Theorem

Let T be a rooted tree with n vertices. The radial algorithm constructs in $O(n)$ time a drawing Γ of T such that:

- Γ is planar
- Each vertex lies on the radial layer equal to its height
- The area of the drawing is at most $O(h^2 d_M^2)$, h-height, d_M-max number of children

Assuming that the radii of consecutive layers differ by the same number and the distance between the vertices on the layer is a constant

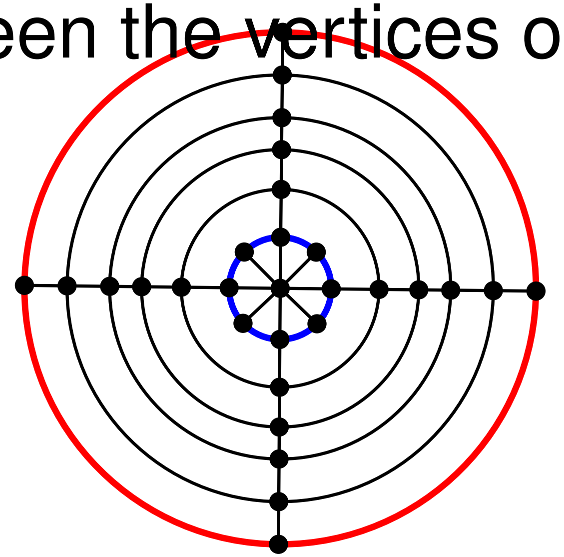
Radial Layout

Theorem

Let T be a rooted tree with n vertices. The radial algorithm constructs in $O(n)$ time a drawing Γ of T such that:

- Γ is planar
- Each vertex lies on the radial layer equal to its height
- The area of the drawing is at most $O(h^2 d_M^2)$, h-height, d_M-max number of children

Assuming that the radii of consecutive layers differ by the same number and the distance between the vertices on the layer is a constant



Radial Layout

Theorem

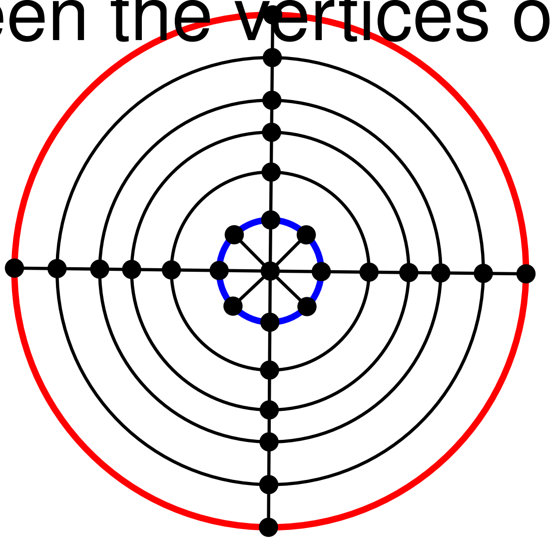
Let T be a rooted tree with n vertices. The radial algorithm constructs in $O(n)$ time a drawing Γ of T such that:

- Γ is planar
- Each vertex lies on the radial layer equal to its height
- The area of the drawing is at most $O(h^2 d_M^2)$, h-height, d_M-max number of children

Assuming that the radii of consecutive layers differ by the same number and the distance between the vertices on the layer is a constant

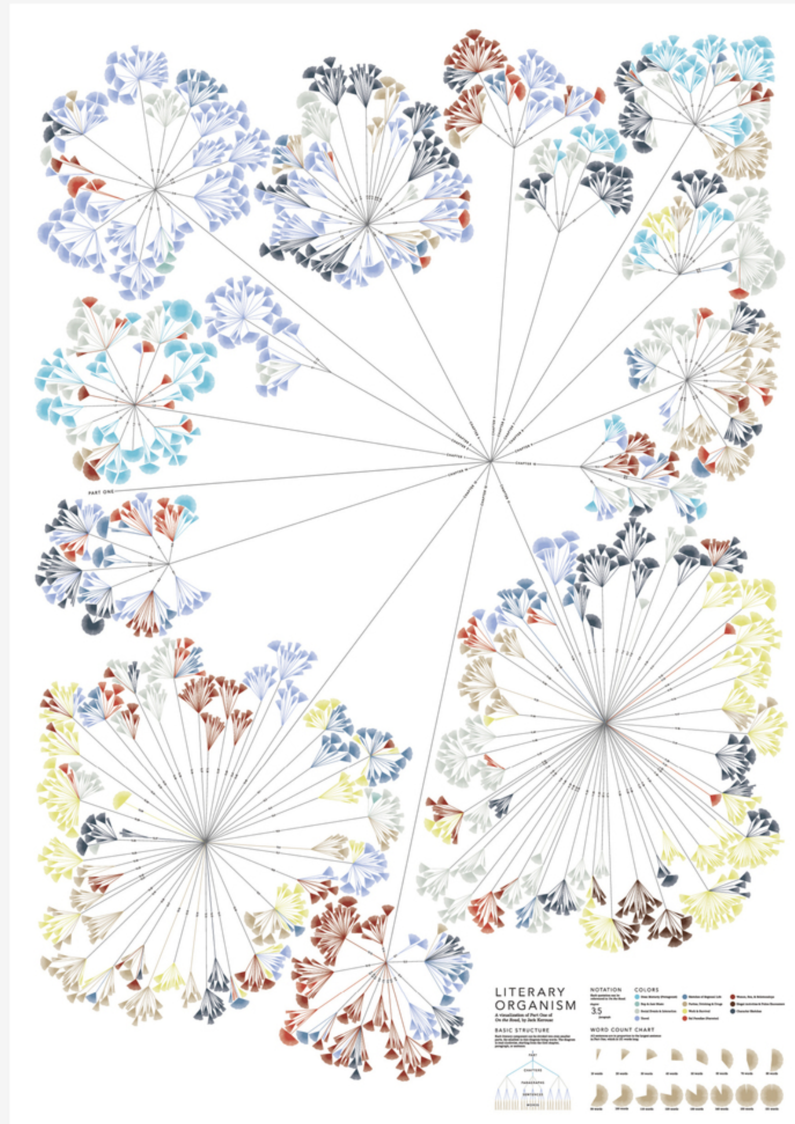
radius of the first layers is $O(d_M)$

radius of the last layer is $O(hd_M)$



Bubble Layout

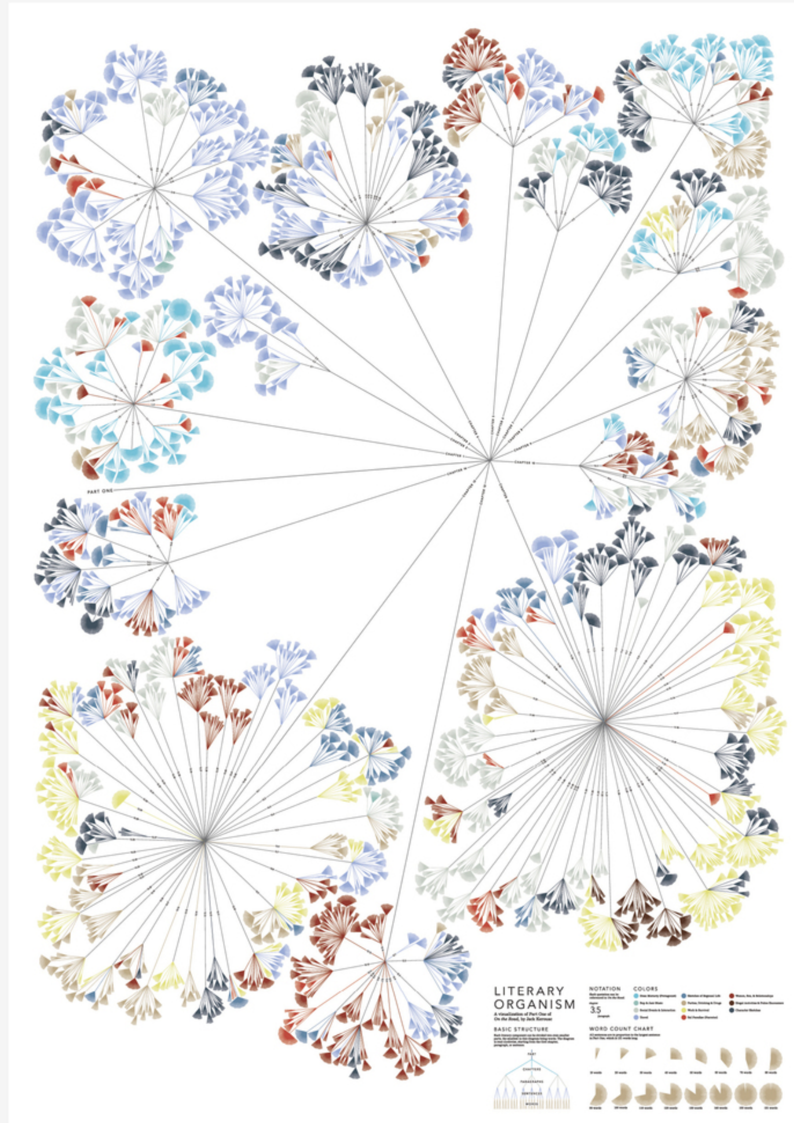
Stefanie Posavec: Writing Without Words:
the project explores methods of visually-representing text and visualises the differences in writing styles when comparing different authors.



Bubble Layout

Stefanie Posavec: Writing Without Words:
the project explores methods of visually-representing text and visualises the differences in writing styles when comparing different authors.

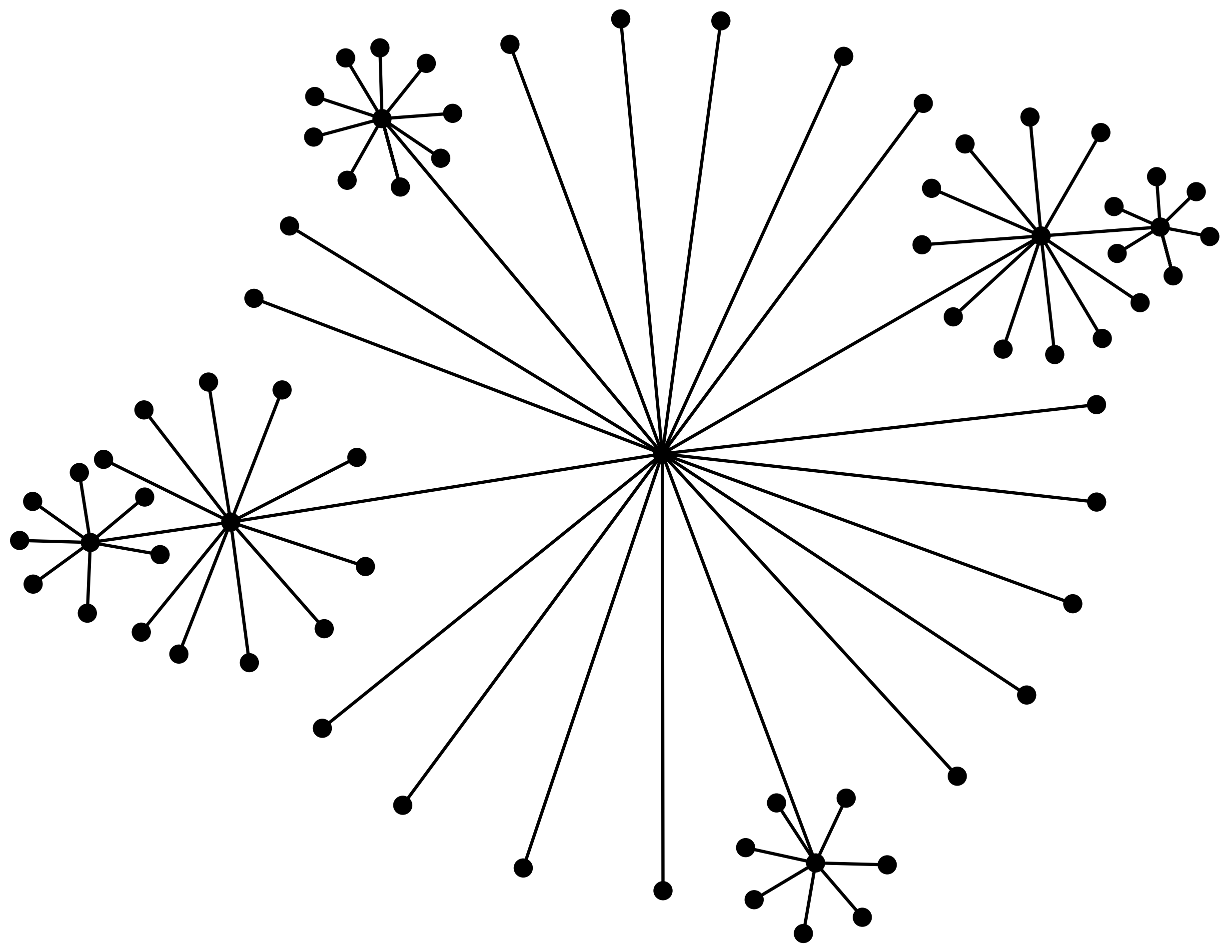
similar to Bubble layout



Bubble Layout

Drawing Conventions:

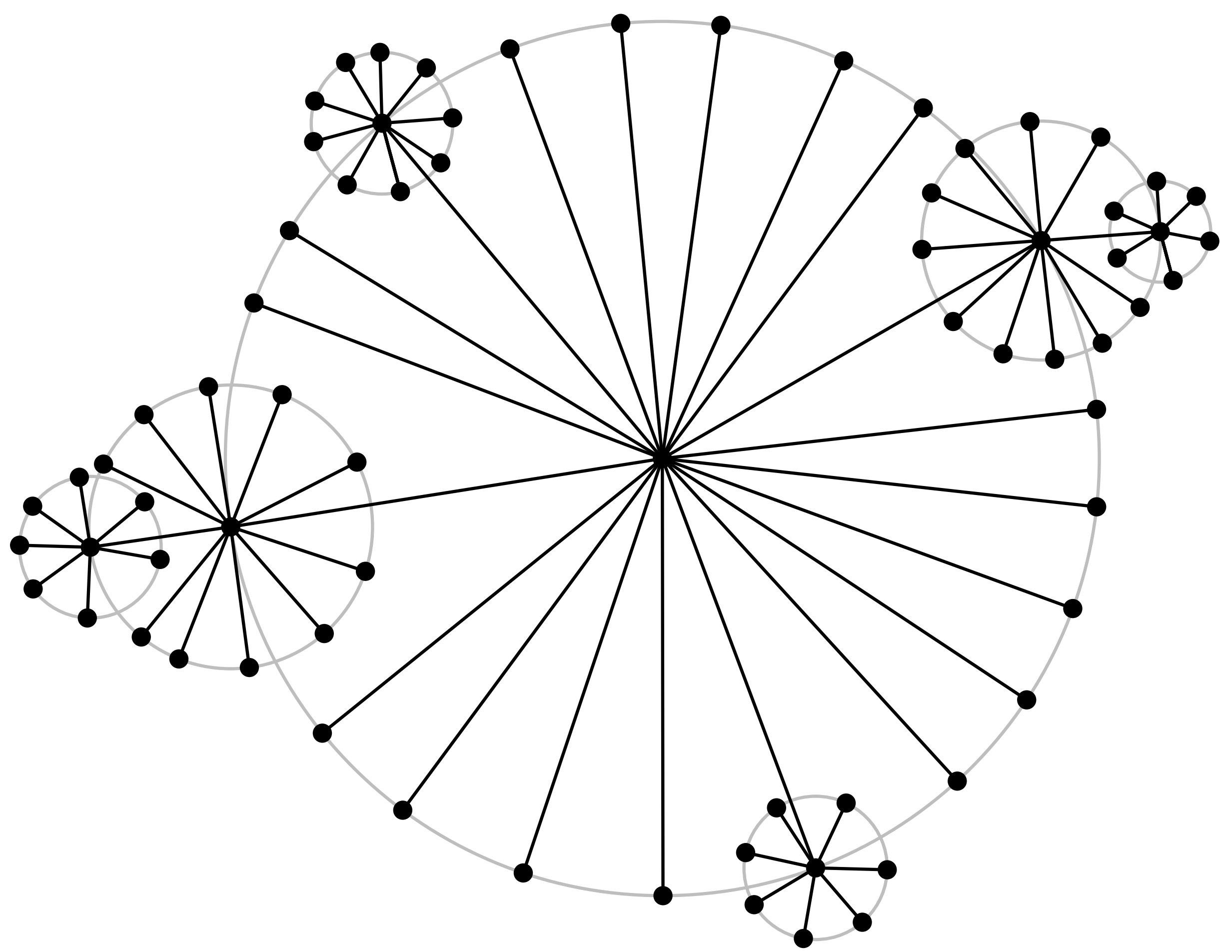
- All children of the same vertex lie on a circle
- Edges do not intersect



Bubble Layout

Drawing Conventions:

- All children of the same vertex lie on a circle
- Edges do not intersect



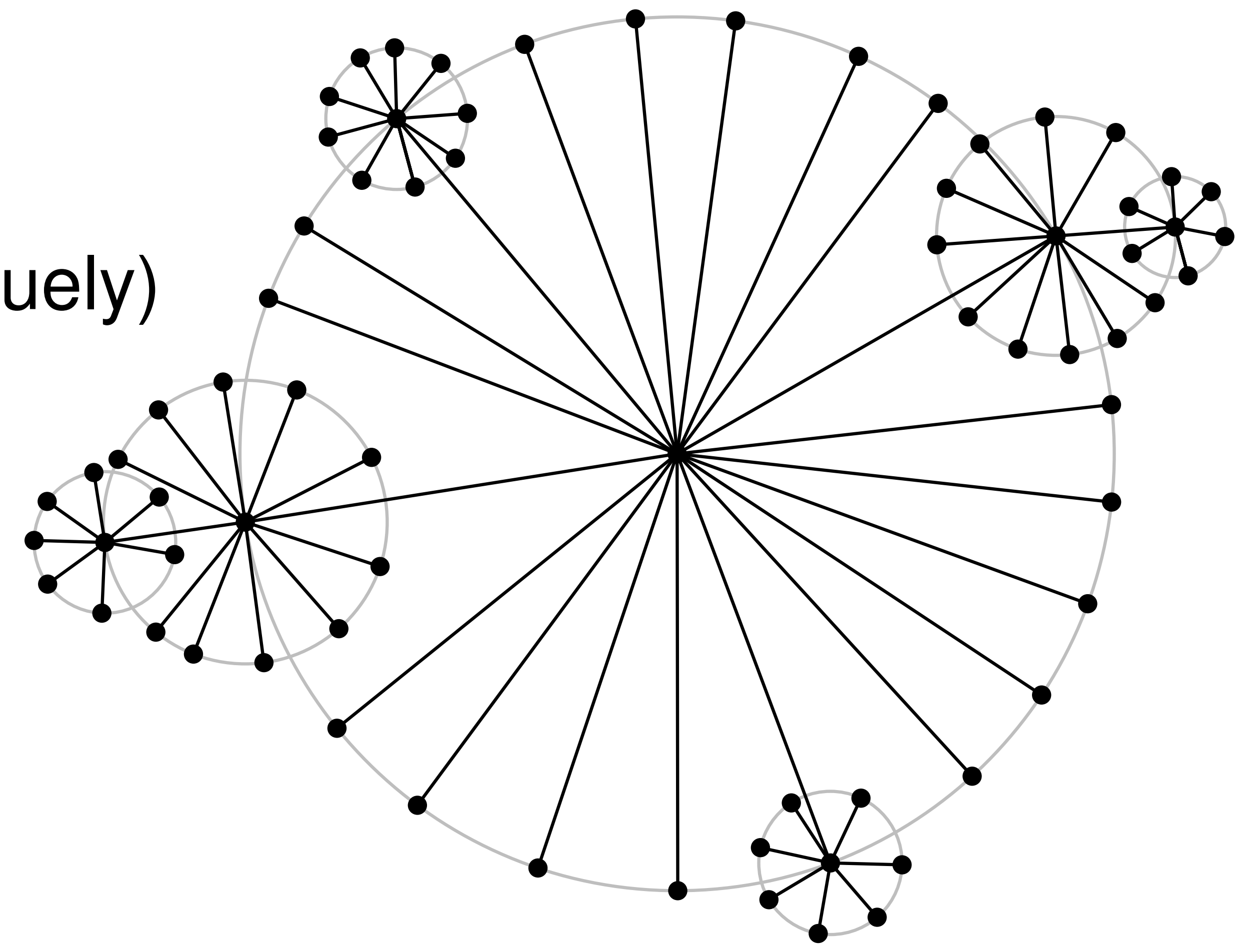
Bubble Layout

Drawing Conventions:

- All children of the same vertex lie on a circle
- Edges do not intersect

Quality Metrics:

- Distribution of the vertices (vaguely)



Bubble Layout

Similar to Reingold&Till ford algorithm (layered layout) - has two stages

First stage: Compute relative position of the children's circles relatively to each node

Second stage: coordinate assignment (taking care of no crossings)

Bubble Layout

Similar to Reingold&Till ford algorithm (layered layout) - has two stages

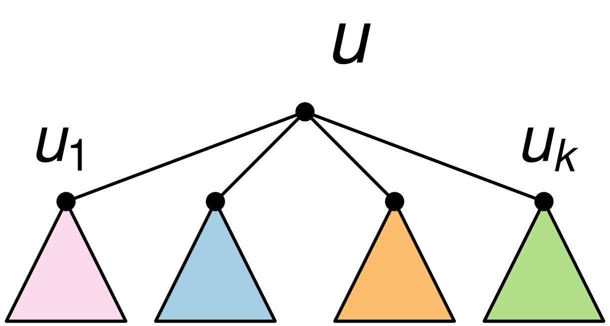
First stage: Compute relative position of the children's circles relatively to each node

Bubble Layout

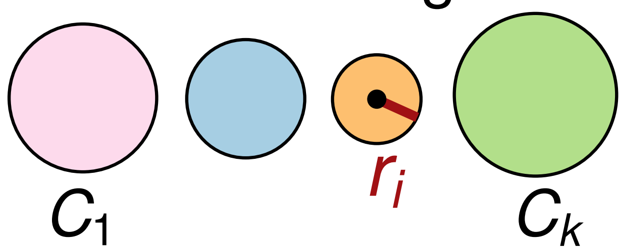
Similar to Reingold&Till ford algorithm (layered layout) - has two stages

First stage: Compute relative position of the children's circles relatively to each node

Postorder traversal: Compute relative coordinates w.r.t. parent



subtrees are already drawn
these are enclosing circles

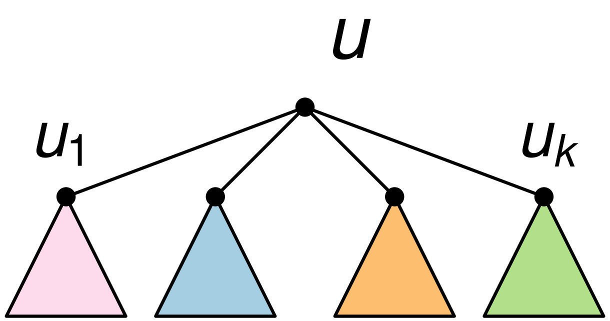


Bubble Layout

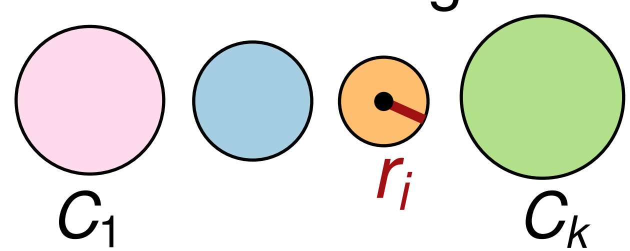
Similar to Reingold&Till ford algorithm (layered layout) - has two stages

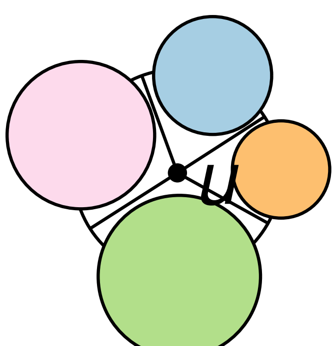
First stage: Compute relative position of the children's circles relatively to each node

Postorder traversal: Compute relative coordinates w.r.t. parent



subtrees are already drawn
these are enclosing circles



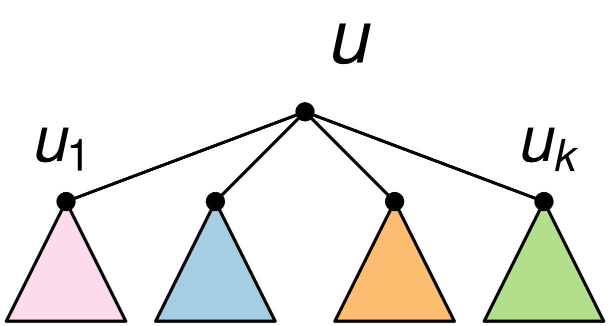


Bubble Layout

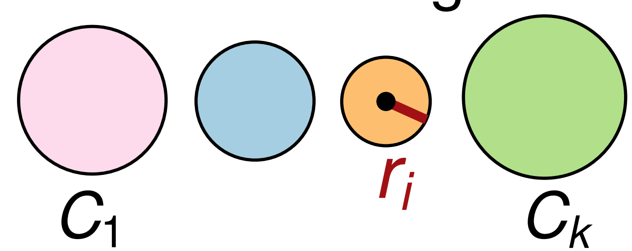
Similar to Reingold&Till ford algorithm (layered layout) - has two stages

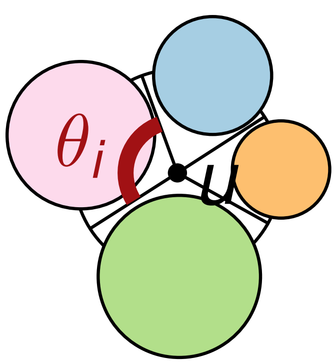
First stage: Compute relative position of the children's circles relatively to each node

Postorder traversal: Compute relative coordinates w.r.t. parent



subtrees are already drawn
these are enclosing circles





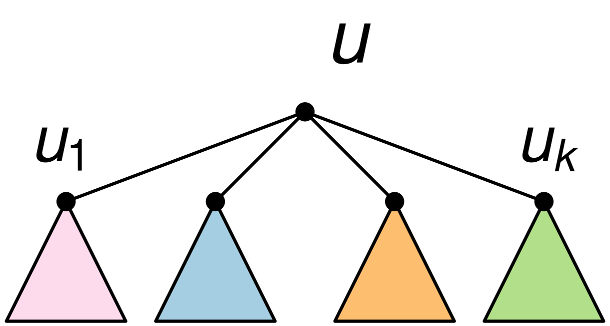
assign θ_i proportionally to r_i

Bubble Layout

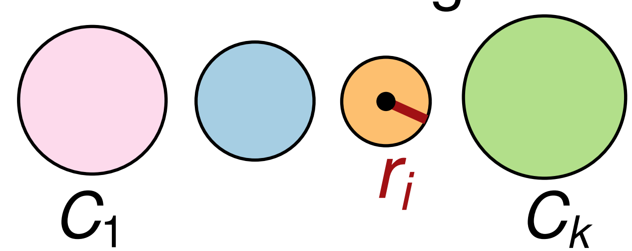
Similar to Reingold&Till ford algorithm (layered layout) - has two stages

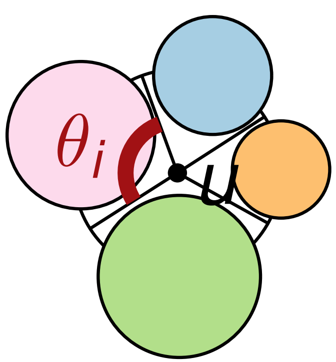
First stage: Compute relative position of the children's circles relatively to each node

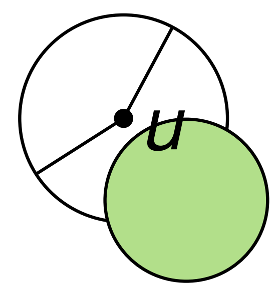
Postorder traversal: Compute relative coordinates w.r.t. parent



subtrees are already drawn
these are enclosing circles



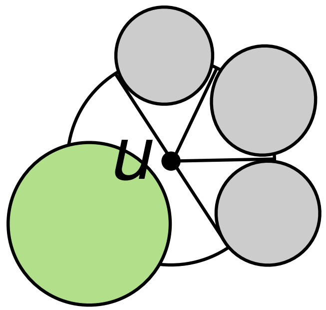




assign θ_i proportionally to r_i

may result in sector $> \pi$

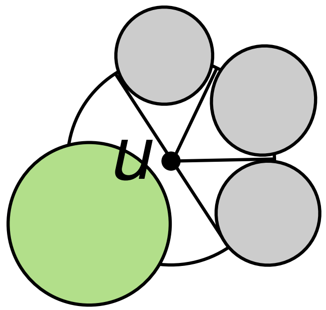
Bubble Layout



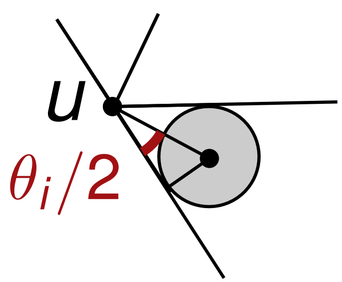
assign angle π to the biggest circle

distribute the rest angles proportionally to r_i

Bubble Layout



assign angle π to the biggest circle
distribute the rest angles proportionally to r_i

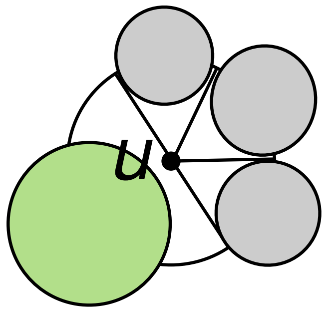


place circles tangent to their sectors

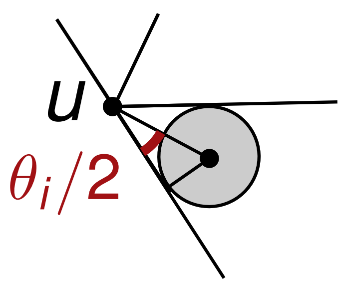
$\sin(\theta_i/2) = r_i/\delta_i$

where δ_i is a distance between u and the center of a circle

Bubble Layout



assign angle π to the biggest circle
distribute the rest angles proportionally to r_i



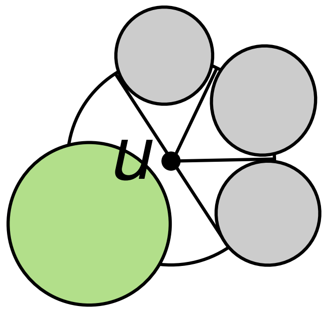
place circles tangent to their sectors

$\sin(\theta_i/2) = r_i/\delta_i$

where δ_i is a distance between u and the center of a circle

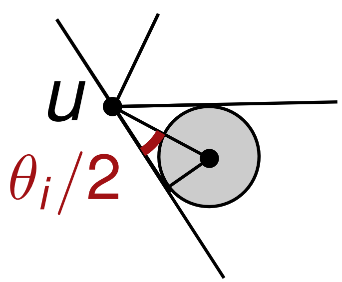
when sector is large the circle may overlap node *n*, so we correct as follows

Bubble Layout



assign angle π to the biggest circle

distribute the rest angles proportionally to r_i



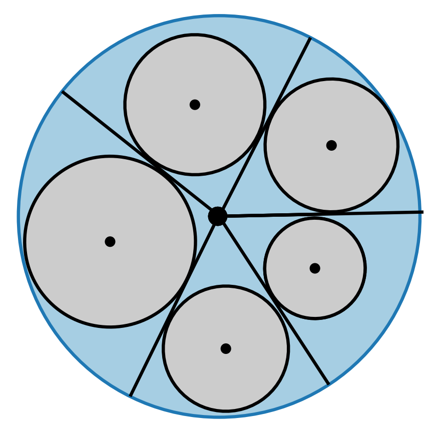
place circles tangent to their sectors

$\sin(\theta_i/2) = r_i/\delta_i$

where δ_i is a distance between u and the center of a circle

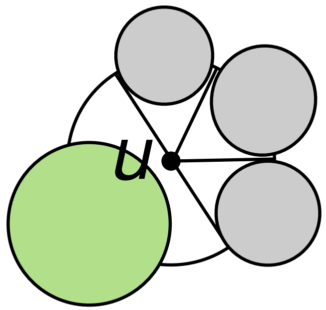
when sector is large the circle may overlap node *n*, so we correct as follows

$$\delta_i = \max\{size(u) + r_i, \frac{r_i}{\sin \theta_i/2}\}$$



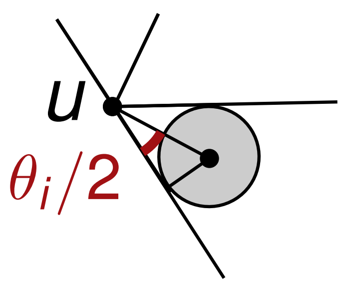
compute the smallest enclosing circle C_u of the circle arrangement

Bubble Layout



assign angle π to the biggest circle

distribute the rest angles proportionally to r_i



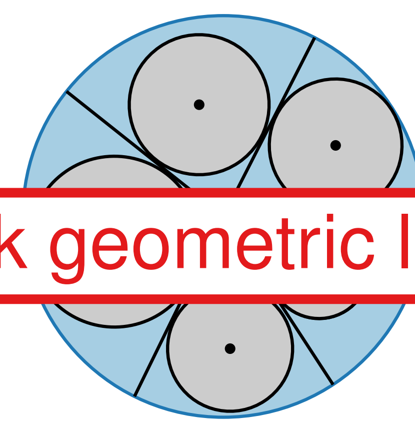
place circles tangent to their sectors

$\sin(\theta_i/2) = r_i/\delta_i$

where δ_i is a distance between u and the center of a circle

when sector is large the circle may overlap node *n*, so we correct as follows

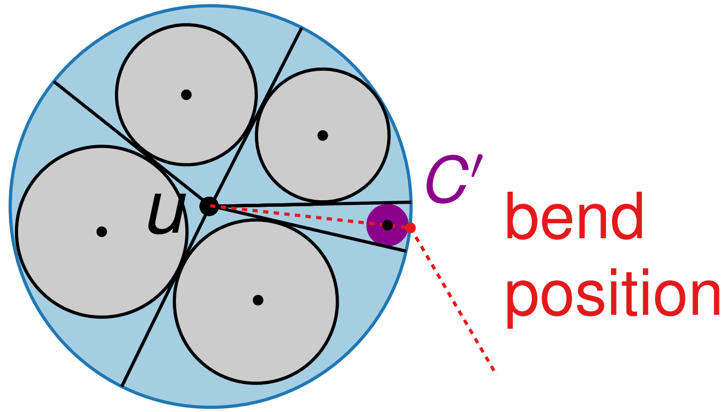
$$\delta_i = \max\{size(u) + r_i, \frac{r_i}{\sin \theta_i/2}\}$$



Check geometric libraries!

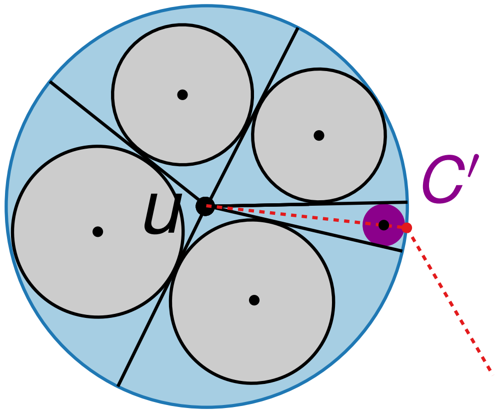
compute the smallest enclosing circle C_u of the circle arrangement

Bubble Layout



in order to connect node u to its ancestor, we use a polyline with one bend β_u. We add a small dummy circle C' and put the bend on the intersection of the C_u and line through u and center of C'

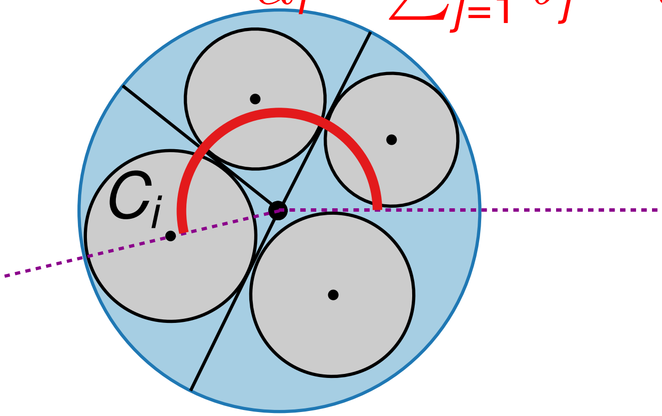
Bubble Layout



in order to connect node u to its ancestor, we use a polyline with one bend β_u. We add a small dummy circle C' and put the bend on the intersection of the C_u and line through u and center of C'

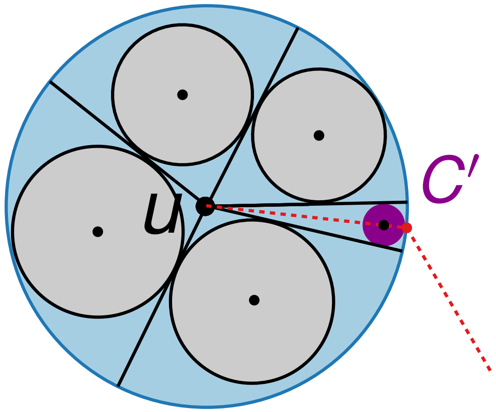
We compute the relative coordinates

$\alpha_i = \sum_{j=1}^{i} \theta_j - \theta_i/2$



$$\gamma_i = \begin{cases} x_i = \delta_i \cos \alpha_i \\ y_i = \delta_i \sin \alpha_i \end{cases}$$

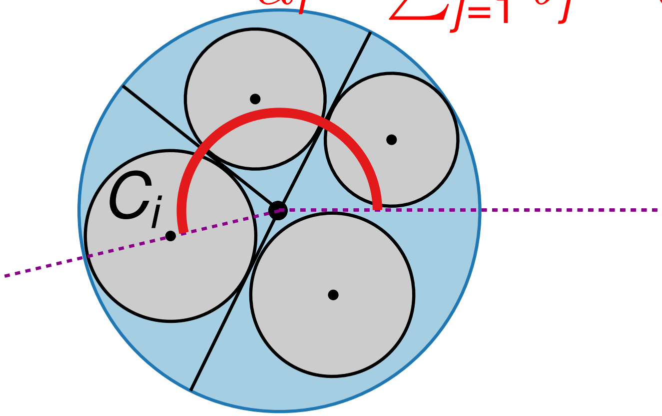
Bubble Layout



in order to connect node u to its ancestor, we use a polyline with one bend β_u. We add a small dummy circle C' and put the bend on the intersection of the C_u and line through u and center of C'

We compute the relative coordinates

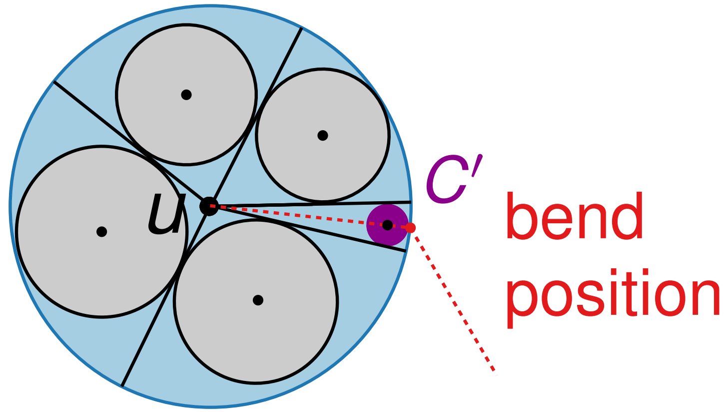
$\alpha_i = \sum_{j=1}^{i} \theta_j - \theta_i/2$



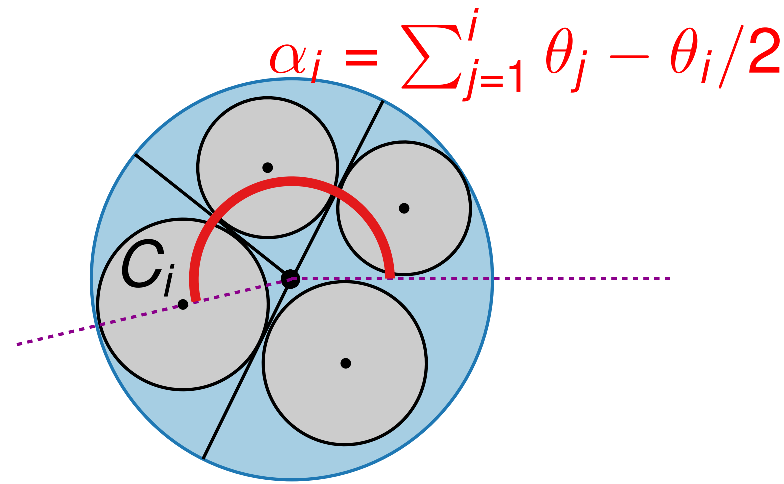
$$\gamma_i = \begin{cases} x_i = \delta_i \cos \alpha_i \\ y_i = \delta_i \sin \alpha_i \end{cases}$$

γ_i – position of the center of C_i with respect to the center of C_u

Bubble Layout



We compute the relative coordinates

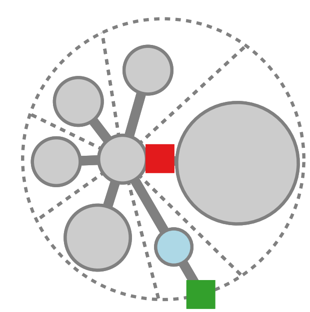


in order to connect node u to its ancestor, we use a polyline with one bend β_u. We add a small dummy circle C' and put the bend on the intersection of the C_u and line through u and center of C'

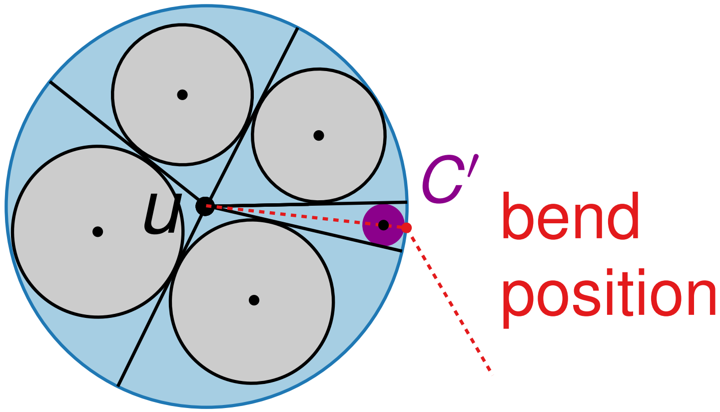
$$\gamma_i = \begin{cases} x_i = \delta_i \cos \alpha_i \\ y_i = \delta_i \sin \alpha_i \end{cases}$$

γ_i – position of the center of C_i with respect to the center of C_u

also compute ζ_u, β_u – position of u and β_u with respect to the center of C_u

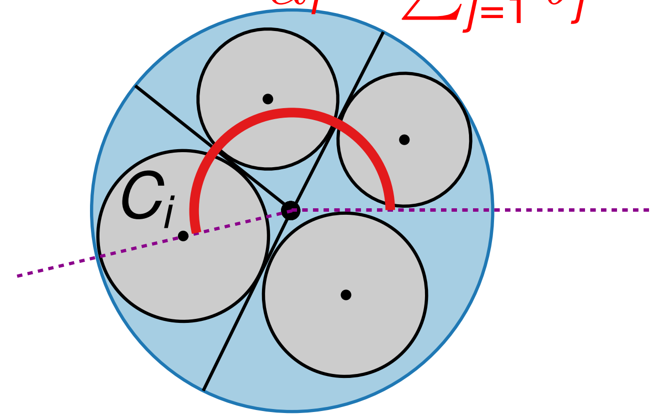


Bubble Layout



We compute the relative coordinates

$$\alpha_i = \sum_{j=1}^{i} \theta_j - \theta_i/2$$

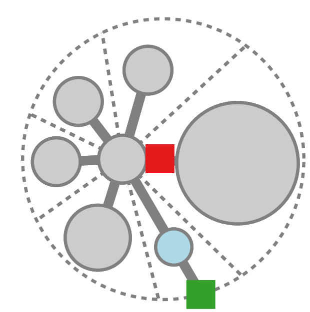


in order to connect node u to its ancestor, we use a polyline with one bend β_u. We add a small dummy circle C' and put the bend on the intersection of the C_u and line through u and center of C'

$$\gamma_i = \begin{cases} x_i = \delta_i \cos \alpha_i \\ y_i = \delta_i \sin \alpha_i \end{cases}$$

γ_i – position of the center of C_i with respect to the center of C_u

also compute ζ_u, β_u – position of u and β_u with respect to the center of C_u



Relative coordinates are vectors!

Bubble Layout

Second stage: coordinate assignment (taking care of no crossings)

Bubble Layout

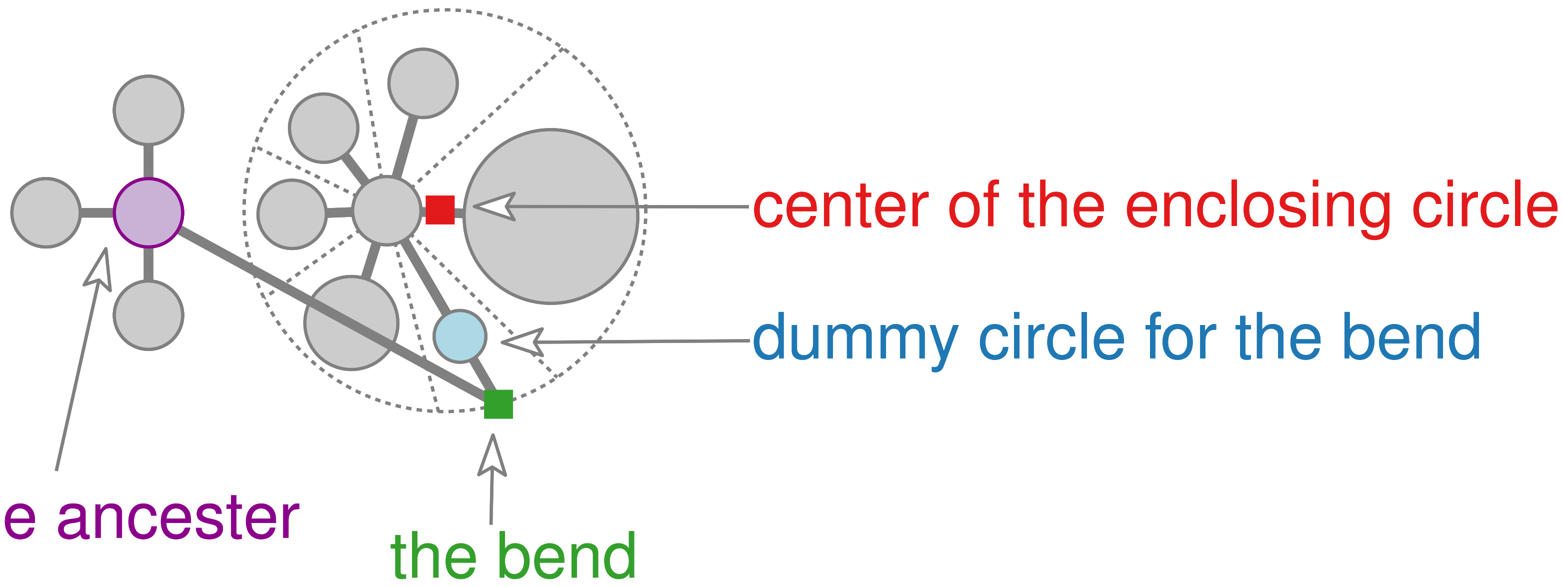
Second stage: coordinate assignment (taking care of no crossings)

Preorder traversal: Compute x- and y- coordinates. Before that, take care of the possible crossings by rotating each child circle as follows

Bubble Layout

Second stage: coordinate assignment (taking care of no crossings)

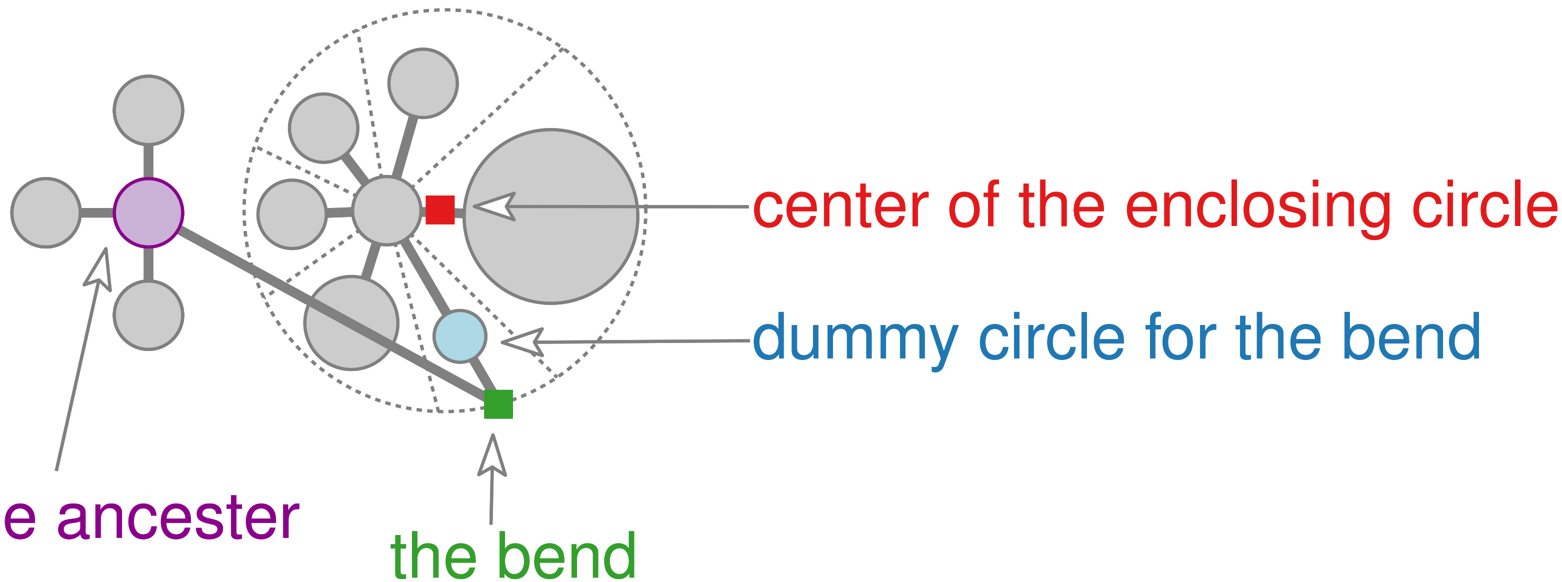
Preorder traversal: Compute x- and y- coordinates. Before that, take care of the possible crossings by rotating each child circle as follows

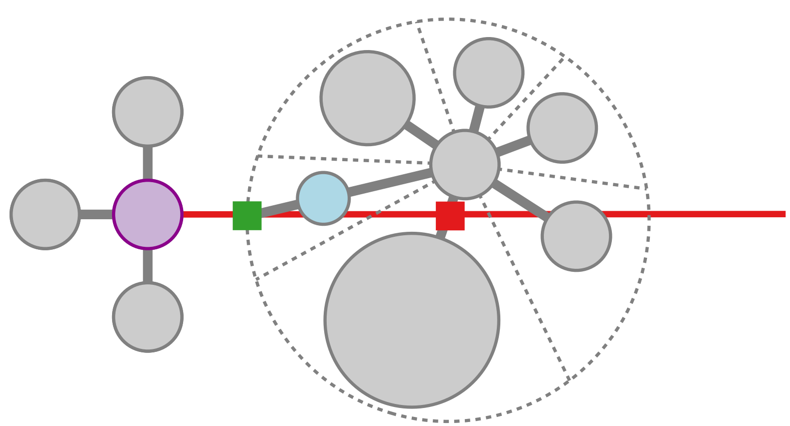


Bubble Layout

Second stage: coordinate assignment (taking care of no crossings)

Preorder traversal: Compute x- and y- coordinates. Before that, take care of the possible crossings by rotating each child circle as follows

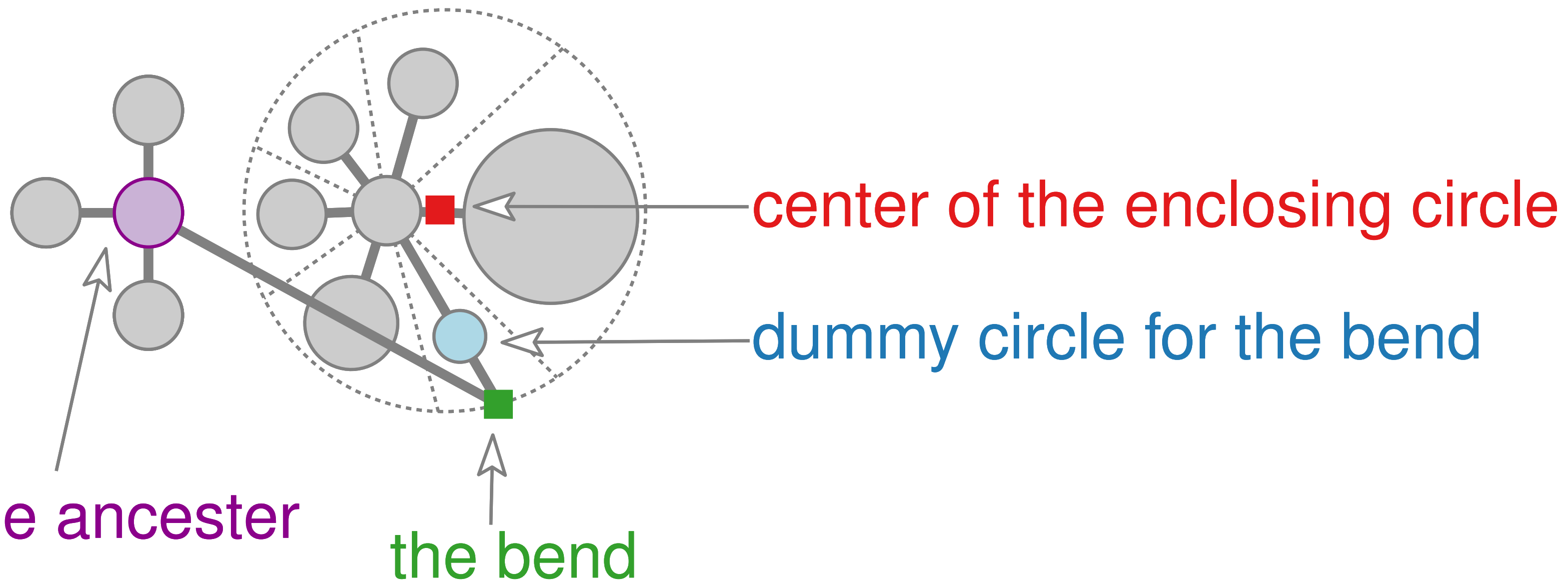


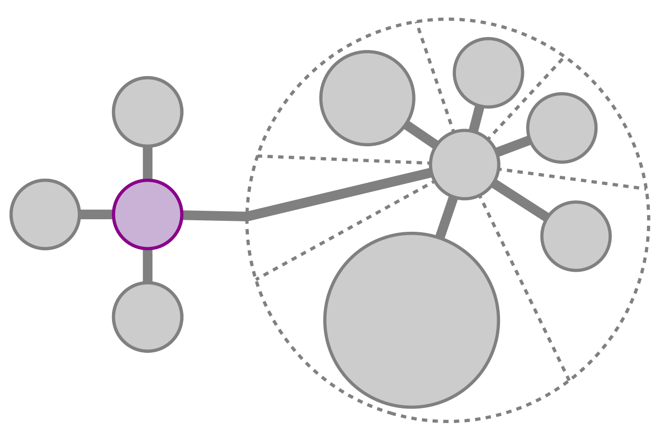


Bubble Layout

Second stage: coordinate assignment (taking care of no crossings)

Preorder traversal: Compute x- and y- coordinates. Before that, take care of the possible crossings by rotating each child circle as follows





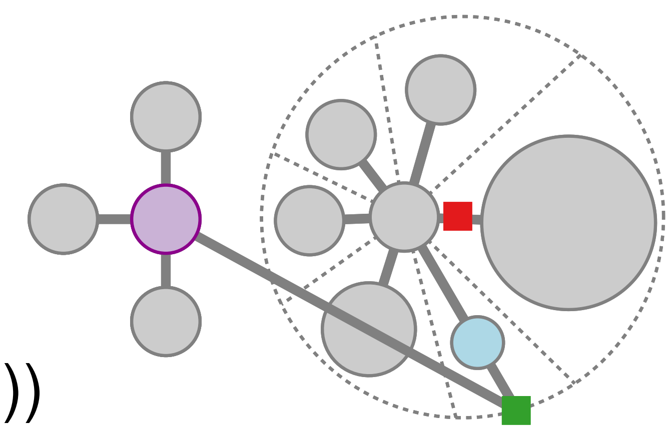
Bubble Layout

Algorithm : Coordinate Assignment

input : u – the node to draw (recall ζ_u, β_u)
C_u^{abs} – the absolute coordinate of the center of circle C_u

function *coordAssign*(u, C_u^{abs})

begin
Let *rot* be the rotation operation of the center of C_u, so that C_u, β_u, and *ancestor*(u) are aligned
Set P_u to $rot(\zeta_u) + C_u^{abs}$
Set P_u^{β} to $rot(\beta_u) + C_u^{abs}$
for all children u_i of u
begin
call *coordAssign*(u_i, $C_u^{abs} + rot(\zeta_u + \gamma_i)$)
end
end



β_u – position of the bend on the edge connecting u to its ancestor
ζ_u – position of u (both relative to the center of C_u)
P_u, P_u^{β} – final positions of u and β_u, respectively

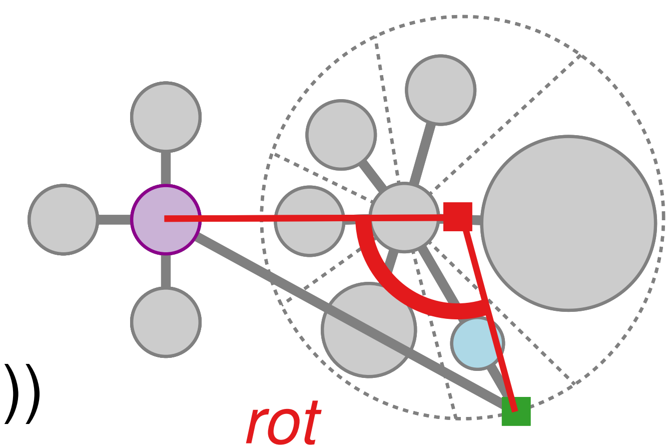
Bubble Layout

Algorithm : Coordinate Assignment

input : u – the node to draw (recall ζ_u, β_u)
C_u^{abs} – the absolute coordinate of the center of circle C_u

function *coordAssign*(u, C_u^{abs})

begin
Let *rot* be the rotation operation of the center of C_u, so that C_u, β_u, and *ancestor*(u) are aligned
Set P_u to $rot(\zeta_u) + C_u^{abs}$
Set P_u^{β} to $rot(\beta_u) + C_u^{abs}$
for all children u_i of u
begin
call *coordAssign*(u_i, $C_u^{abs} + rot(\zeta_u + \gamma_i)$)
end
end



β_u – position of the bend on the edge connecting u to its ancestor
ζ_u – position of u (both relative to the center of C_u)
P_u, P_u^{β} – final positions of u and β_u, respectively

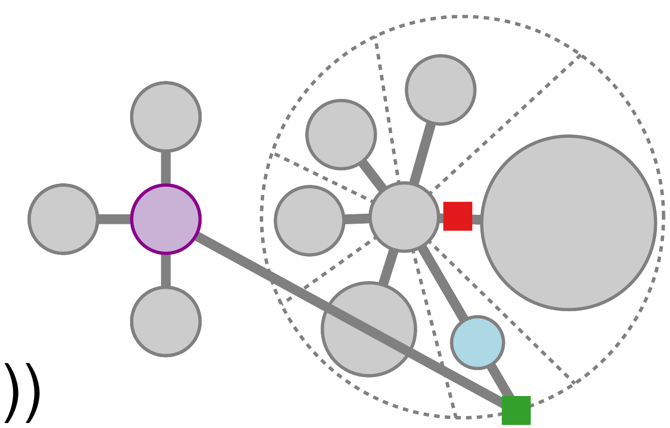
Bubble Layout

Algorithm : Coordinate Assignment

input : u – the node to draw (recall ζ_u, β_u)
C_u^{abs} – the absolute coordinate of the center of circle C_u

function *coordAssign*(u, C_u^{abs})

begin
Let *rot* be the rotation operation of the center of C_u, so that C_u, β_u, and *ancestor*(u) are aligned
Set P_u to $rot(\zeta_u) + C_u^{abs}$
Set P_u^{β} to $rot(\beta_u) + C_u^{abs}$
for all children u_i of u
begin
call *coordAssign*(u_i, $C_u^{abs} + rot(\zeta_u + \gamma_i)$)
end
end



β_u – position of the bend on the edge connecting u to its ancestor
ζ_u – position of u (both relative to the center of C_u)
P_u, P_u^{β} – final positions of u and β_u, respectively

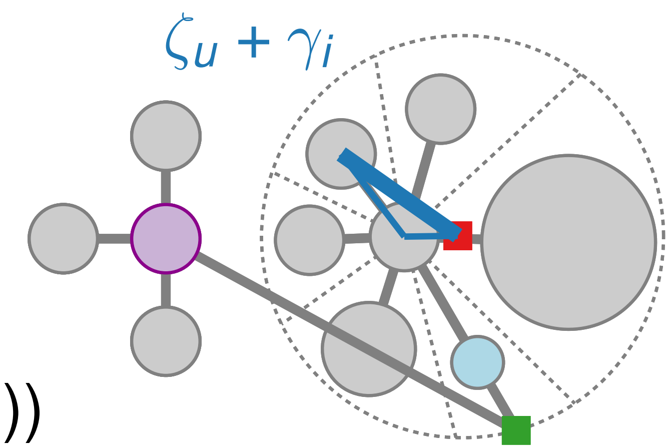
Bubble Layout

Algorithm : Coordinate Assignment

input : u – the node to draw (recall ζ_u, β_u)
C_u^{abs} – the absolute coordinate of the center of circle C_u

function *coordAssign*(u, C_u^{abs})

begin
Let *rot* be the rotation operation of the center of C_u, so that C_u, β_u, and *ancestor*(u) are aligned
Set P_u to $rot(\zeta_u) + C_u^{abs}$
Set P_u^{β} to $rot(\beta_u) + C_u^{abs}$
for all children u_i of u
begin
call *coordAssign*(u_i, $C_u^{abs} + rot(\zeta_u + \gamma_i)$)
end
end



β_u – position of the bend on the edge connecting u to its ancestor
ζ_u – position of u (both relative to the center of C_u)
P_u, P_u^{β} – final positions of u and β_u, respectively

Bubble Layout

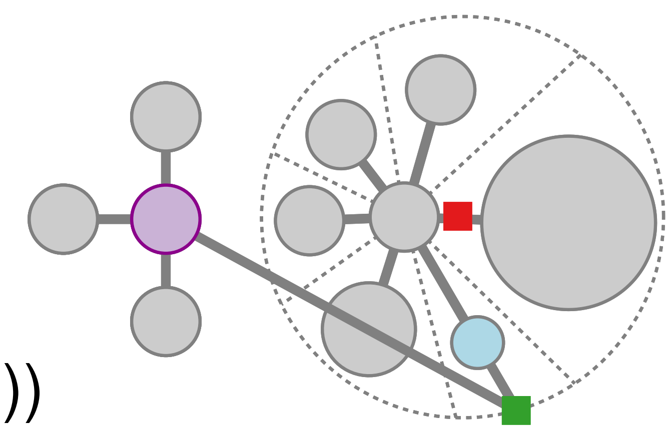
Algorithm : Coordinate Assignment

input : u – the node to draw (recall ζ_u, β_u)
C_u^{abs} – the absolute coordinate of the center of circle C_u

function $coordAssign(u, C_u^{abs})$

Vector operations!

begin
  Let rot be the rotation operation of the center of C_u, so that C_u, β_u, and $ancestor(u)$ are aligned
  Set P_u to $rot(\zeta_u) + C_u^{abs}$
  Set P_u^{β} to $rot(\beta_u) + C_u^{abs}$
  for all children u_i of u
  begin
    call $coordAssign(u_i, C_u^{abs} + rot(\zeta_u + \gamma_i))$
  end
end



β_u – position of the bend on the edge connecting u to its ancestor
ζ_u – position of u (both relative to the center of C_u)
P_u, P_u^{β} – final positions of u and β_u, respectively

Bubble Layout

Time complexity is $O(n \log n)$ if for the enclosing circle we use the algorithm by Welzl (“Smallest enclosing discs”, 1991).

Bubble Layout

Time complexity is $O(n \log n)$ if for the enclosing circle we use the algorithm by Welzl ("Smallest enclosing discs", 1991).

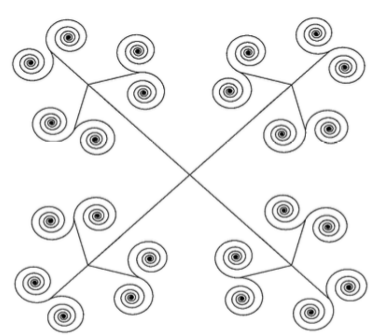
Task: Think and discuss in which situations the resulting drawings have many bends and in which no bends at all? (use virtual board for sketching your ideas)

Bubble Layout

Time complexity is $O(n \log n)$ if for the enclosing circle we use the algorithm by Welzl (“Smallest enclosing discs”, 1991).

Task: Think and discuss in which situations the resulting drawings have many bends and in which no bends at all? (use virtual board for sketching your ideas)

Bends create a spiral effect in case of very unballanced trees



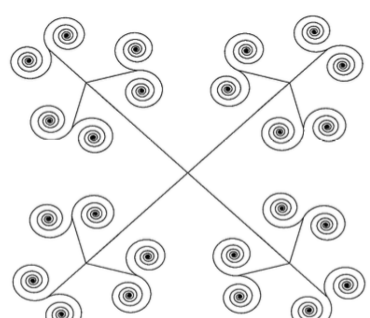
Bubble Layout

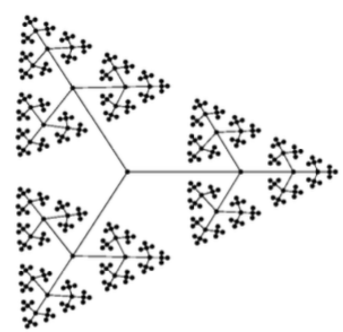
Time complexity is $O(n \log n)$ if for the enclosing circle we use the algorithm by Welzl ("Smallest enclosing discs", 1991).

Task: Think and discuss in which situations the resulting drawings have many bends and in which no bends at all? (use virtual board for sketching your ideas)

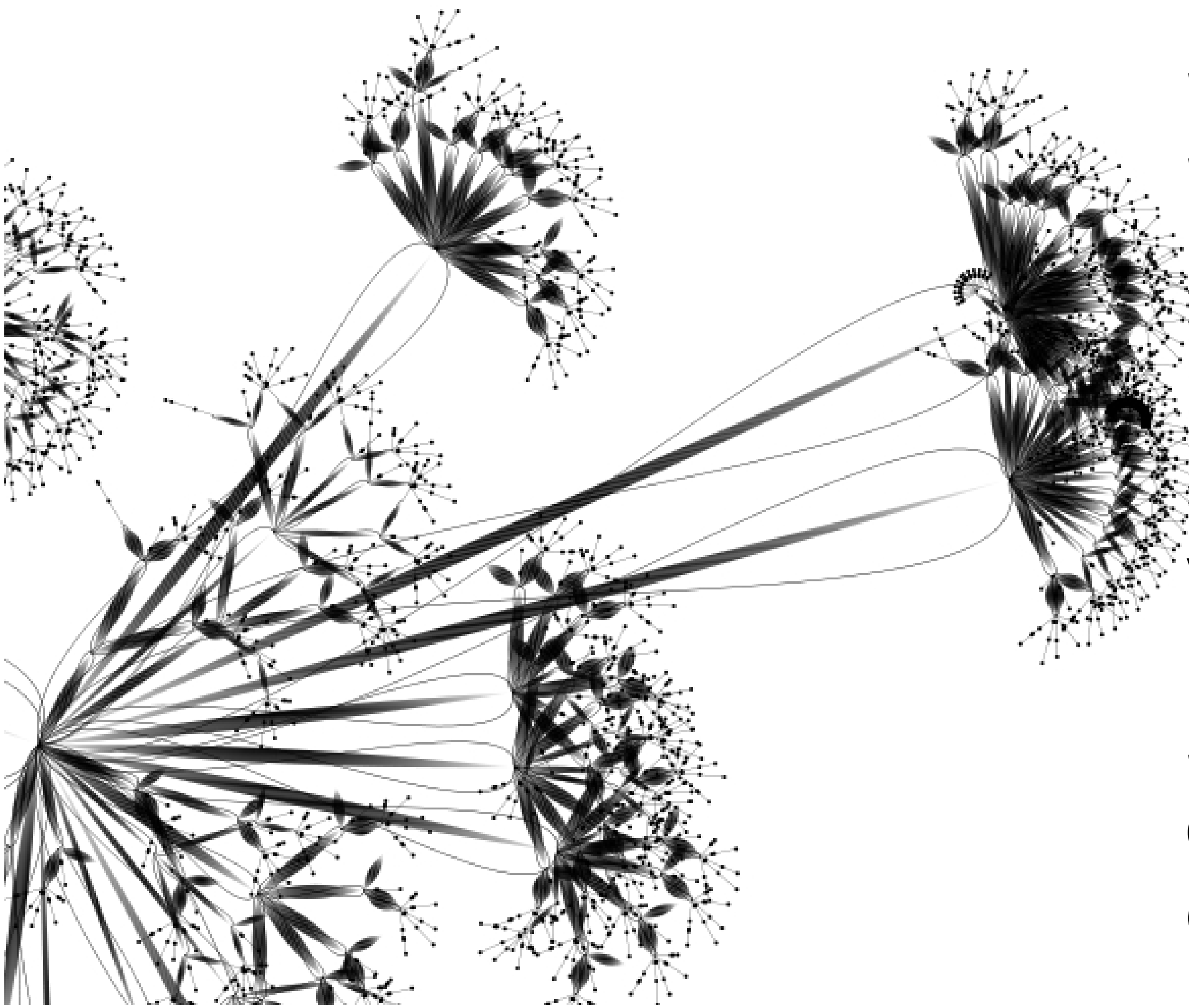
Bends create a spiral effect in case of very unballanced trees

On the other hand the number of bends is zero for a completely balanced tree





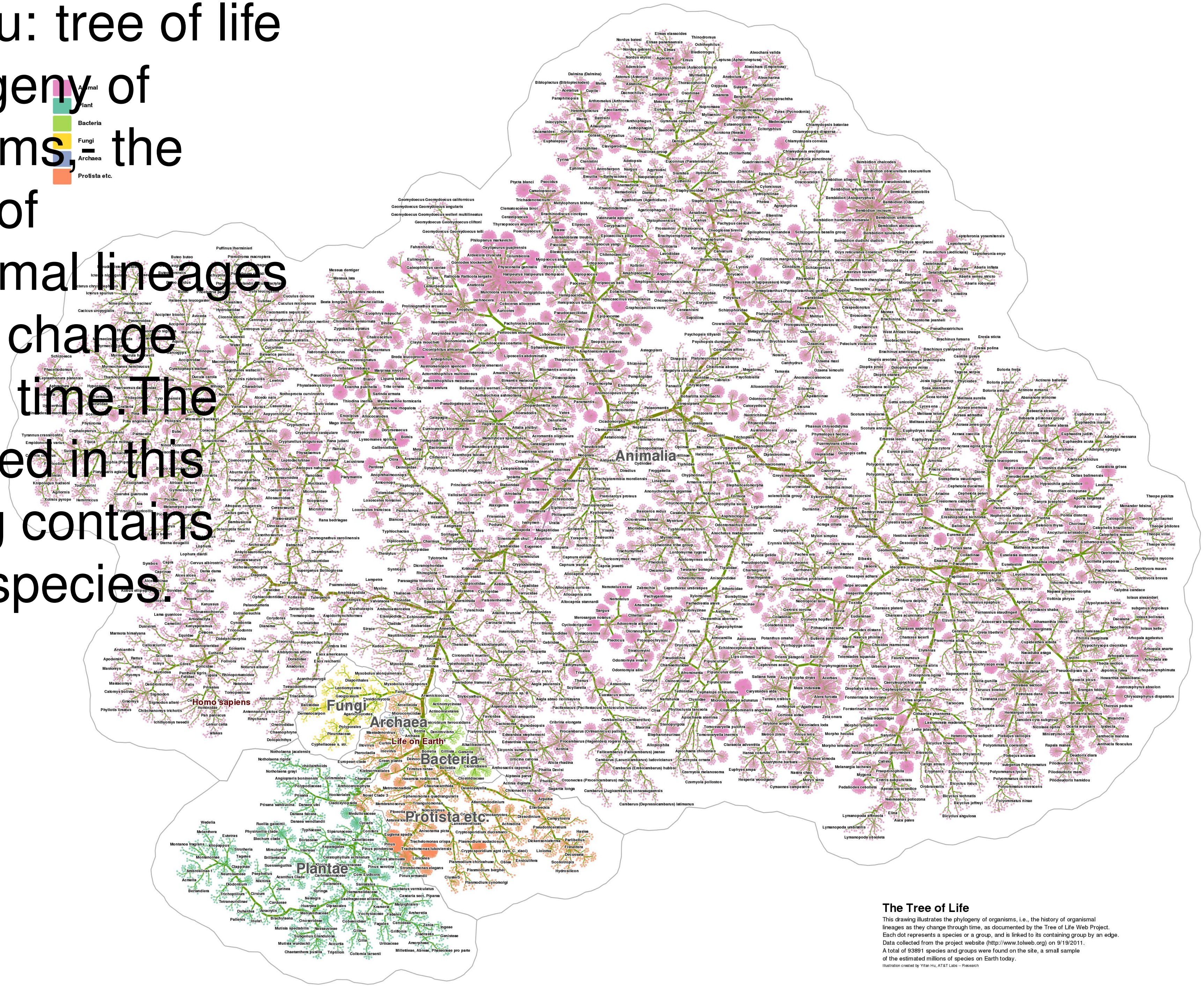
Inspired by Bubble Layout



Oli Laruelle ”The source code structure and work progress of software project. The intention was to represent the sheer volume of work put into open source software development usually created by a small number of people.”

Inspired by Bubble Layout

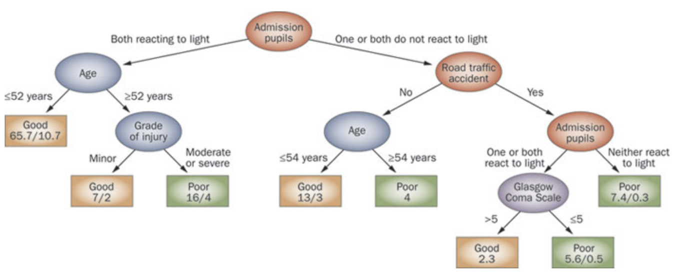
Yifan Hu: tree of life - phylogeny of organisms, - the history of organismal lineages as they change through time.The data used in this drawing contains 93891 species.

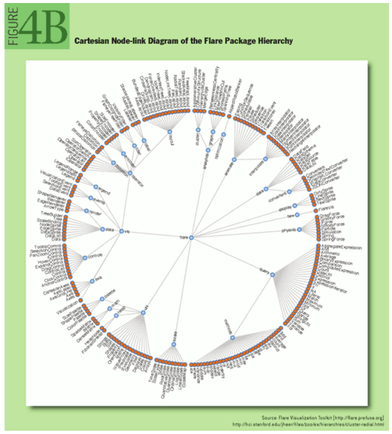


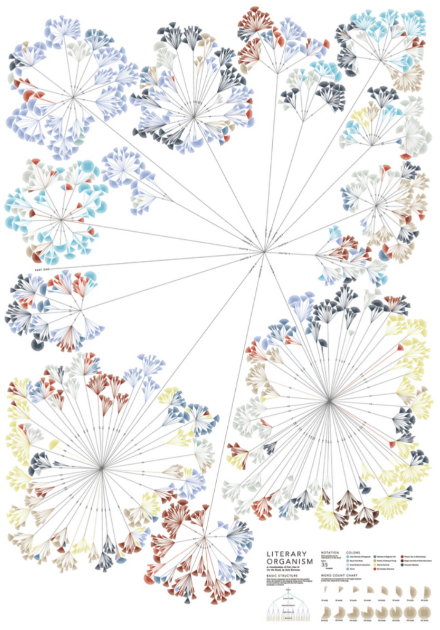
Summary and Reading

We looked at tree drawing algorithms:

Layered layout, radial layout and bubble layout



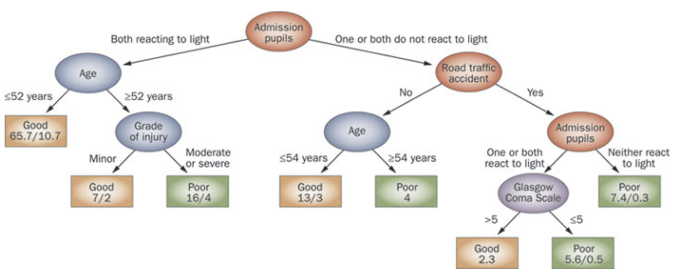


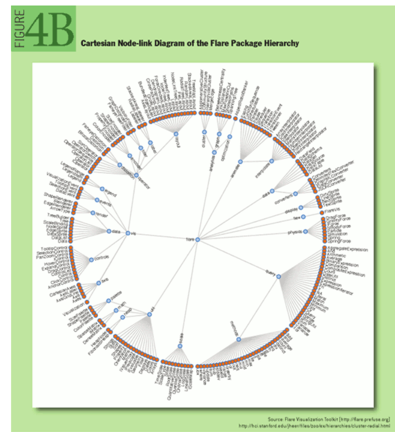


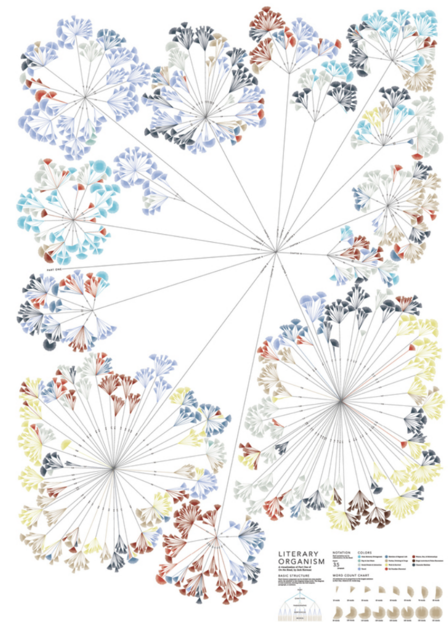
Summary and Reading

We looked at tree drawing algorithms:

Layered layout, radial layout and bubble layout







Additional Reading

Layered Layout: Book Di Battista et al: Chapter 3.1.2

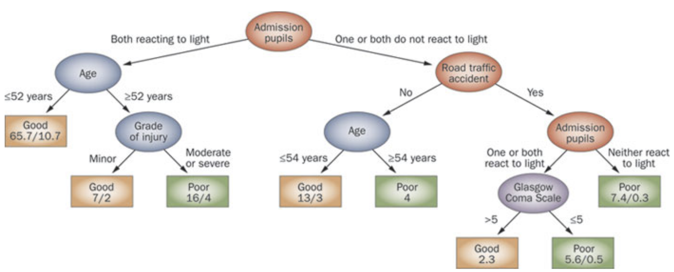
Radial Layout: Book Di Battista et al: Chapter 3.1.3

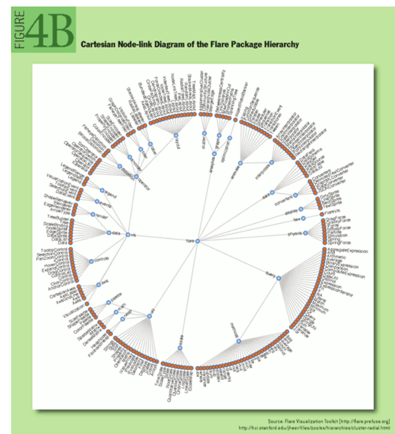
Bubble Layout: Paper “Bubble Tree Drawing Algorithm” Grivet et al.

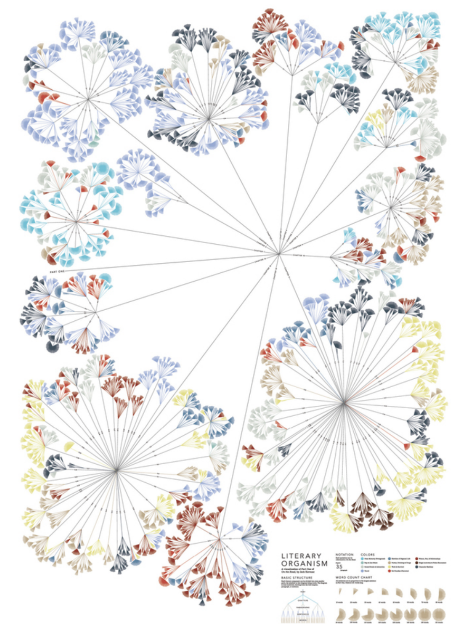
Summary and Reading

We looked at tree drawing algorithms:

Layered layout, radial layout and bubble layout







Next

Algorithm for visualization of general graphs