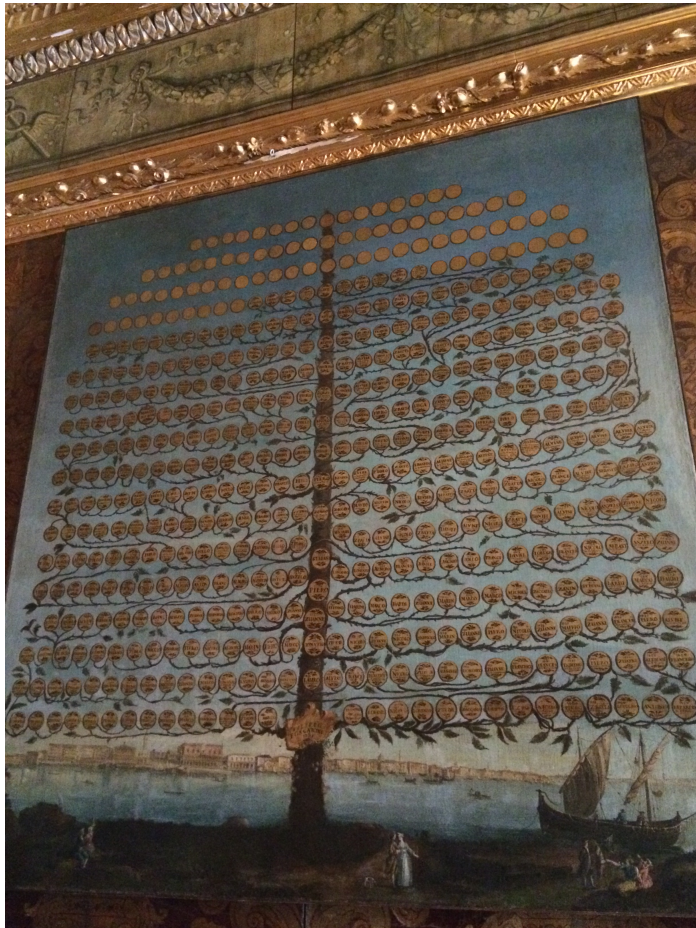
Tree Layout

Lecture Graph Drawing Algorithms · 192.053

Martin Nöllenburg

20.03.2018

ALGORITHMS AND
COMPLEXITY GROUP



Before we start...

- Registration for (mandatory) exercise groups starts today until Friday, March 23, 23:55.
- TUWEL page $\rightarrow$ Exercises has more details
- If no more groups with your favorite topic available, send us a message and we will open more.

Before we start...

- Registration for (mandatory) exercise groups starts today until Friday, March 23, 23:55.
- TUWEL page $\rightarrow$ Exercises has more details
- If no more groups with your favorite topic available, send us a message and we will open more.

Any further questions?

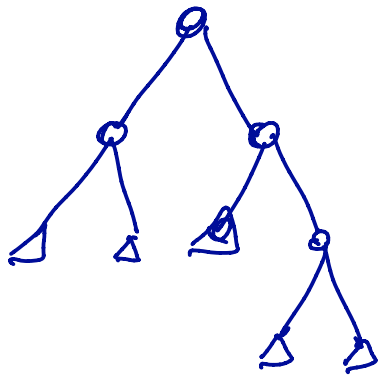
Motivation

What types of tree drawings do we know?

- what are typical drawing conventions?
- which aesthetics shall be optimized?

Let's collect some properties!

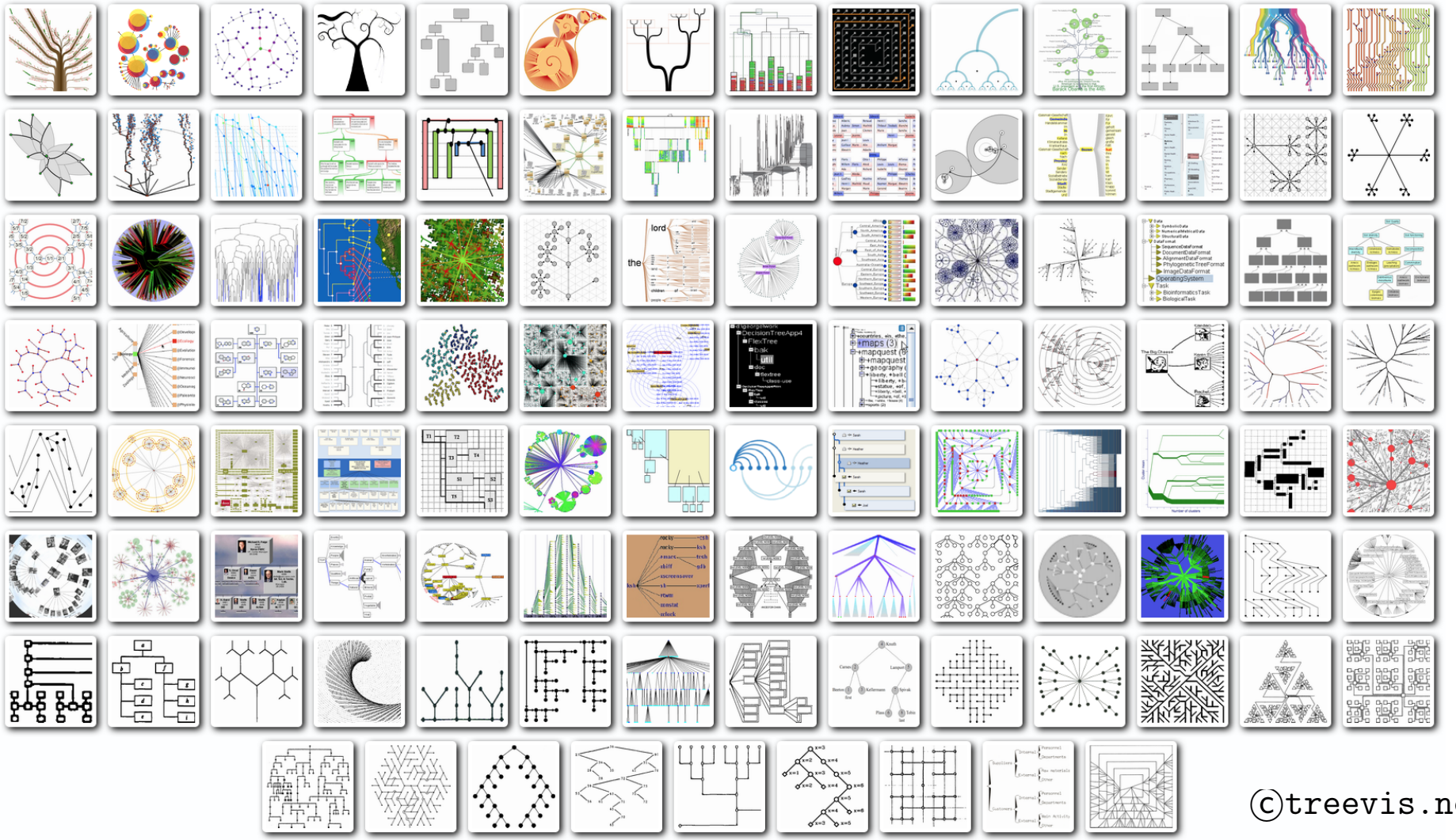
- symmetry btw. branches
- uniform edge length
- no overlaps / crossings
- uniform node distances
- upward / downward orientation
- different symbols for inner / leaves



Motivation

What types of tree drawings do we know?

- what are typical drawing conventions?
- which aesthetics shall be optimized?



©treevis.net

Overview

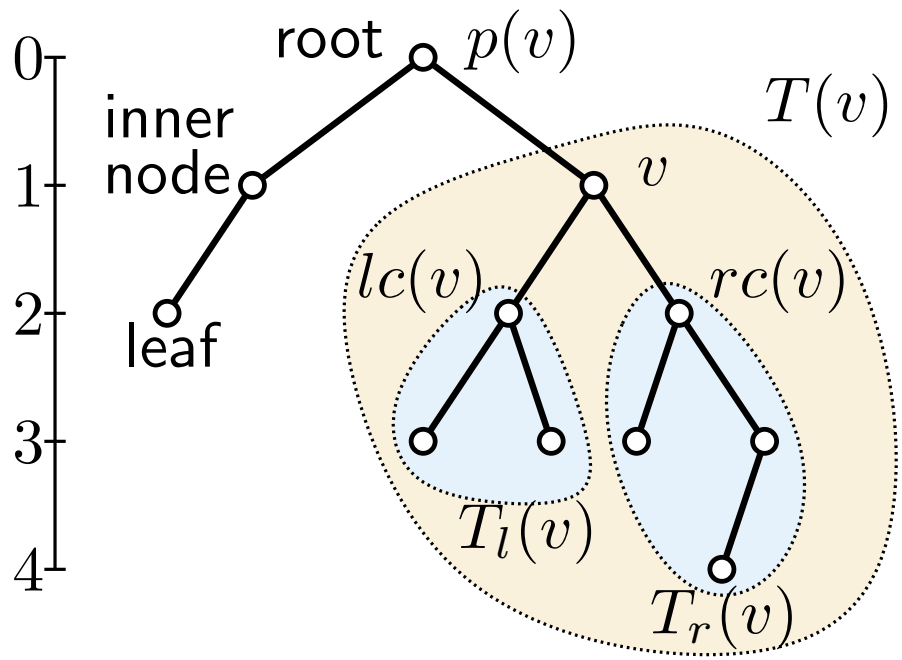
Layered drawings of rooted trees

Radial tree drawings

Compact orthogonal drawings of rooted trees

Layered Binary Tree Layout – Terminology

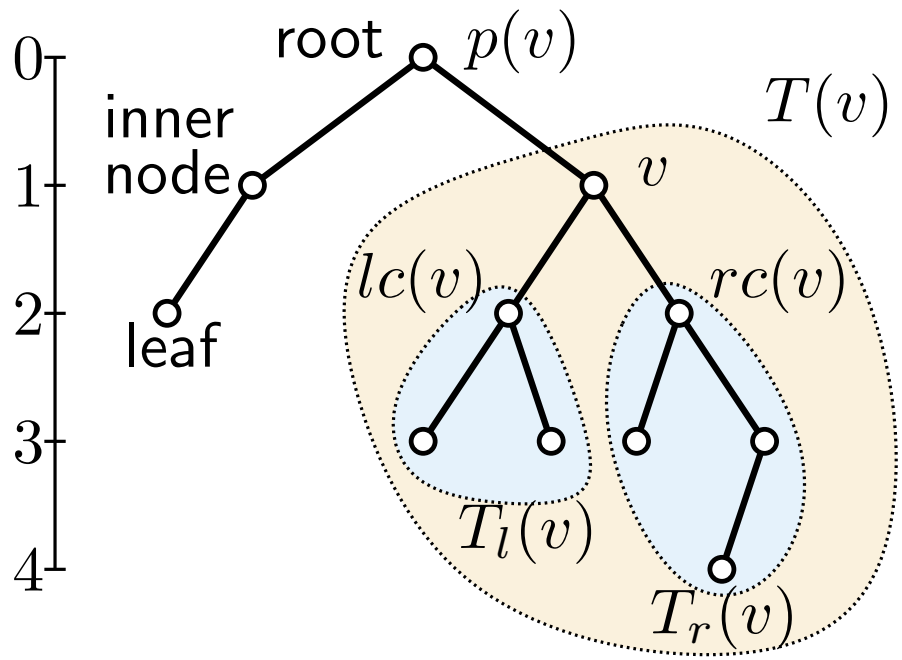
Let's start with textbook-style tree drawings in computer science:



- root r
- parent $p(v)$
- left child $lc(v)$, right child $rc(v)$
- subtree $T(v)$ rooted at v
- left and right subtrees $T_l(v), T_r(v)$
- leaf vs. inner node
- depth(v): distance to root
- height(T) $= \max_v$ depth(v)
- ordered/unordered: left/right given?

Layered Binary Tree Layout – Terminology

Let's start with textbook-style tree drawings in computer science:



- root r
- parent $p(v)$
- left child $lc(v)$, right child $rc(v)$
- subtree $T(v)$ rooted at v
- left and right subtrees $T_l(v), T_r(v)$
- leaf vs. inner node
- depth(v): distance to root
- height(T) $= \max_v$ depth(v)
- ordered/unordered: left/right given?

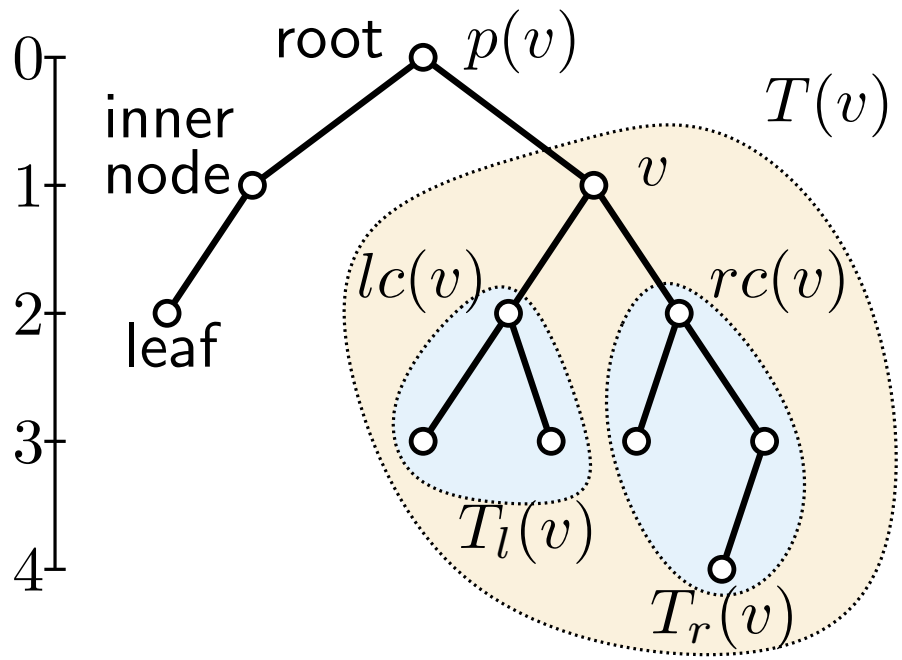
Input: rooted (ordered) binary tree T with n nodes

Output: layered tree drawing of T with

- y-coordinates given by depth
- parent centered above children
- straight-line edges
- no crossings
- isomorphic subtrees drawn the same
- integer grid positions (?)
- compact

Layered Binary Tree Layout – Terminology

Let's start with textbook-style tree drawings in computer science:



- root r
- parent $p(v)$
- left child $lc(v)$, right child $rc(v)$
- subtree $T(v)$ rooted at v
- left and right subtrees $T_l(v), T_r(v)$
- leaf vs. inner node
- depth(v): distance to root
- height$(T) = \max_v$ depth(v)
- ordered/unordered: left/right given?

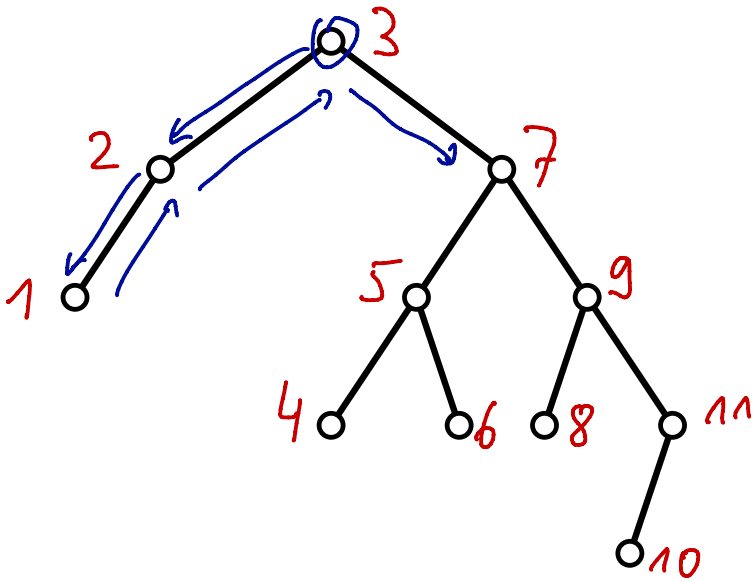
Input: rooted (ordered) binary tree T with n nodes

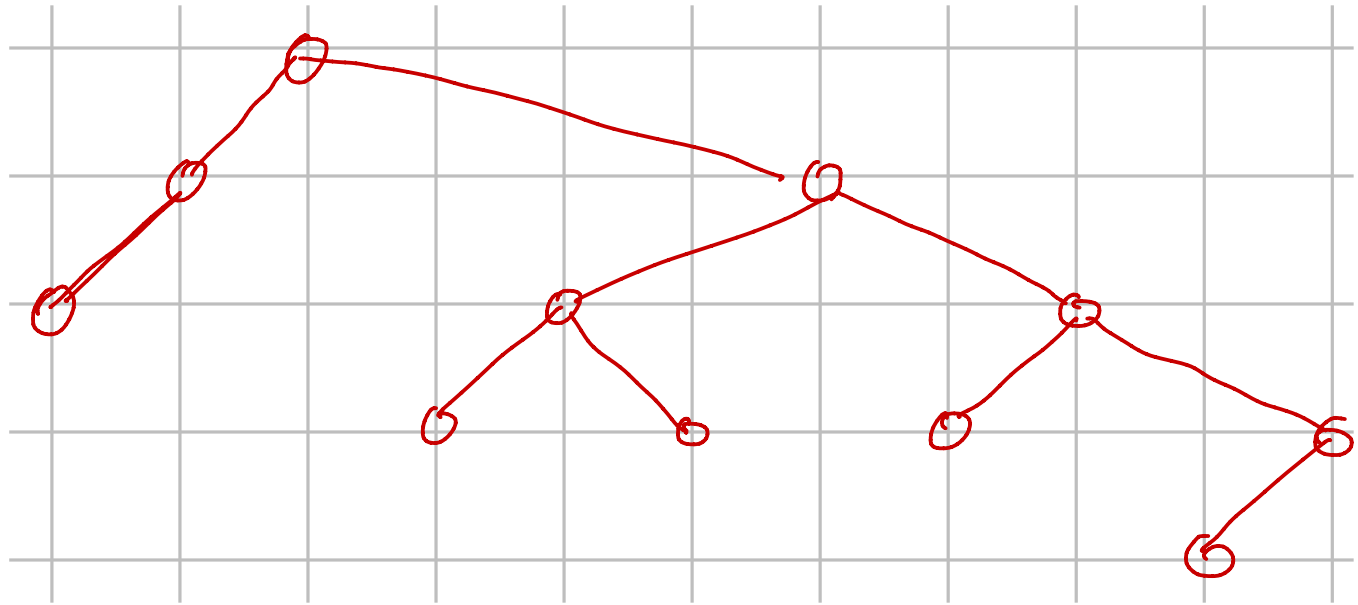
Output: layered tree drawing of T with

- y-coordinates given by depth
- parent centered above children
- straight-line edges
- no crossings
- isomorphic subtrees drawn the same
- integer grid positions (?)
- compact

Simple Algorithms: Tree Traversal

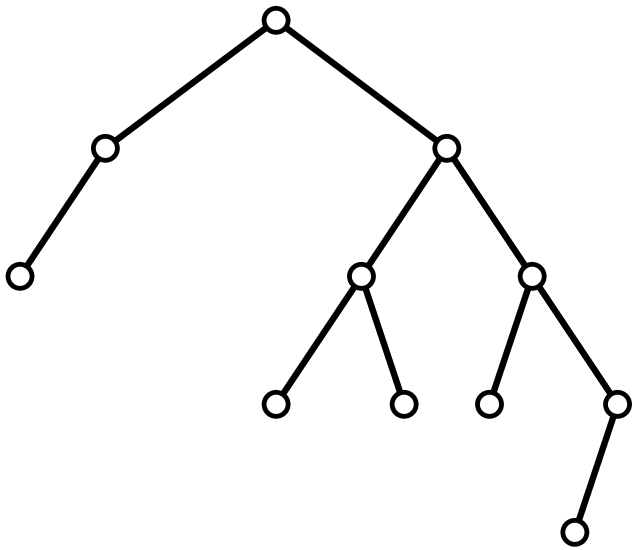
Recall: depth-first search and preorder, inorder, postorder traversal for x-coordinates





Simple Algorithms: Tree Traversal

Recall: depth-first search and preorder, inorder, postorder traversal for x-coordinates

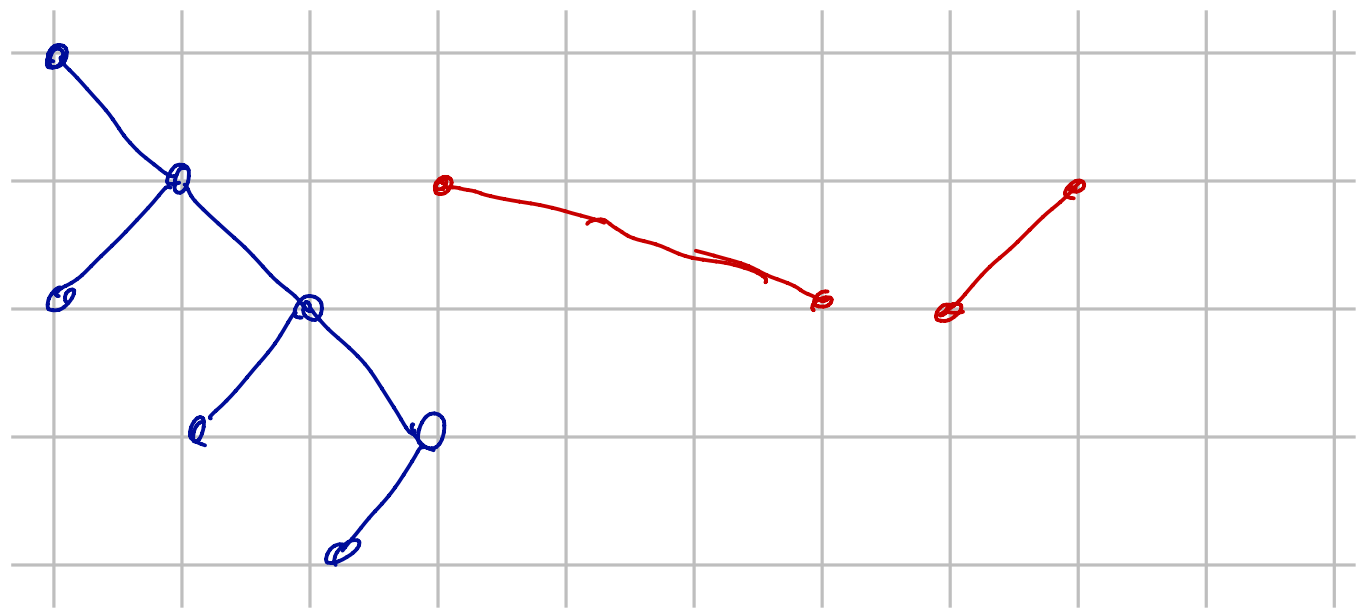


Properties

- running time: $O(n)$
- drawing area: $n \cdot \text{height}(T) = O(n^2)$
- conventions:
  - upward, straight line, no crossings, isomorphic subtrees

issues:

- area
- parent position



Improving Area and Parent Position

Recursive algorithm of Reingold/Tilford (1981):

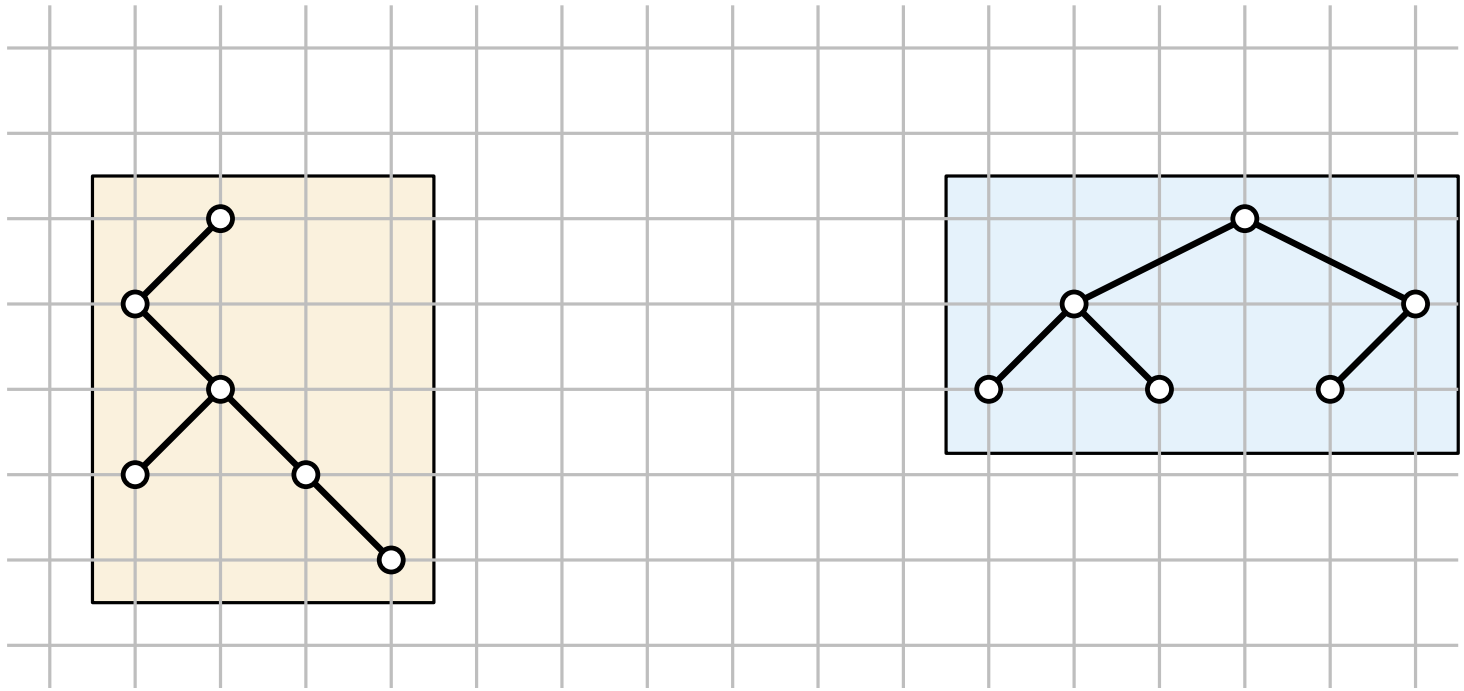
Base case: leaf v, place at layer $-\text{depth}(v)$ ⊡

Improving Area and Parent Position

Recursive algorithm of Reingold/Tilford (1981):

Base case: leaf v, place at layer $-\text{depth}(v)$ ◙

Divide: inner node v, recursively draw $T_l(v)$ and $T_r(v)$



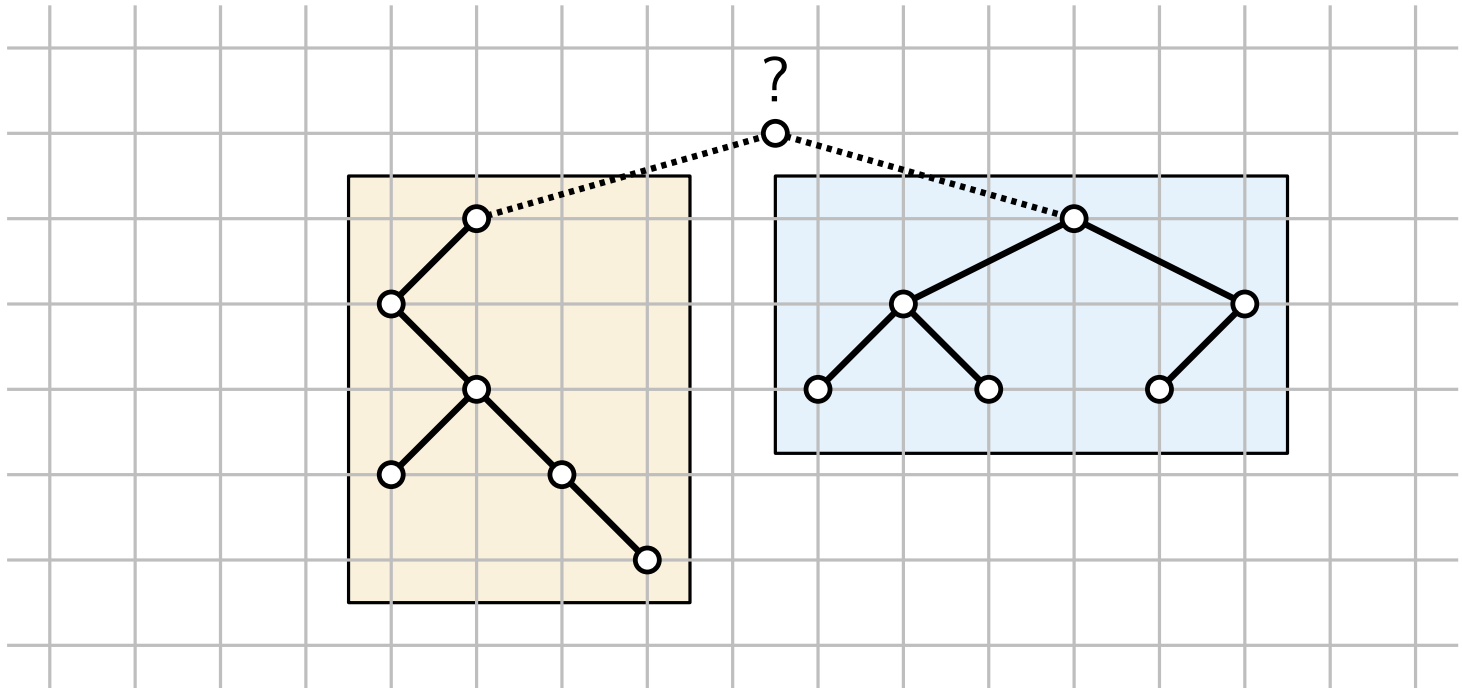
Improving Area and Parent Position

Recursive algorithm of Reingold/Tilford (1981):

Base case: leaf v, place at layer $-\text{depth}(v)$ ◙

Divide: inner node v, recursively draw $T_l(v)$ and $T_r(v)$

Conquer: place subtree drawings next to each other with distance 2 or 3 and center v above $lc(v)$ and $rc(v)$



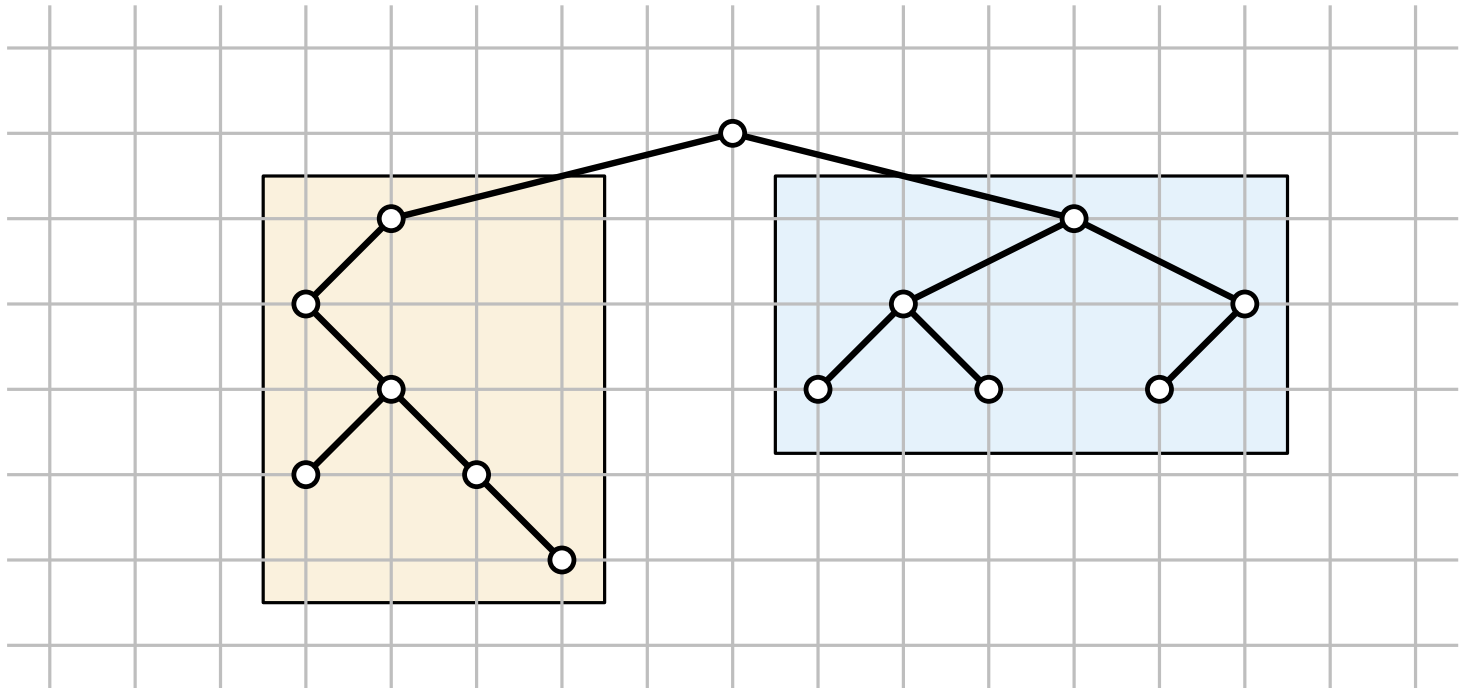
Improving Area and Parent Position

Recursive algorithm of Reingold/Tilford (1981):

Base case: leaf v, place at layer $-\text{depth}(v)$ ◙

Divide: inner node v, recursively draw $T_l(v)$ and $T_r(v)$

Conquer: place subtree drawings next to each other with distance 2 or 3 and center v above $lc(v)$ and $rc(v)$



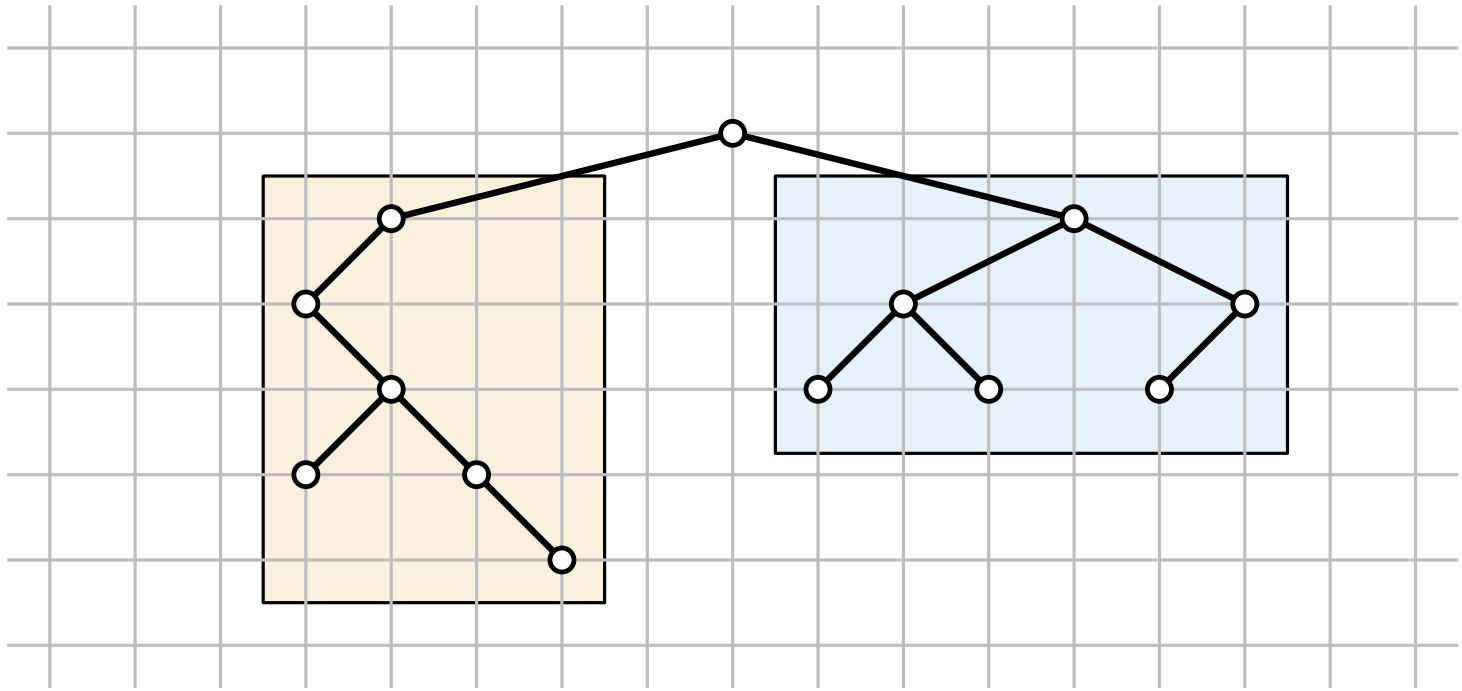
Improving Area and Parent Position

Recursive algorithm of Reingold/Tilford (1981):

Base case: leaf v, place at layer $-\text{depth}(v)$ ◙

Divide: inner node v, recursively draw $T_l(v)$ and $T_r(v)$

Conquer: place subtree drawings next to each other with distance 2 or 3 and center v above $lc(v)$ and $rc(v)$



parent centered ✓ smaller area?

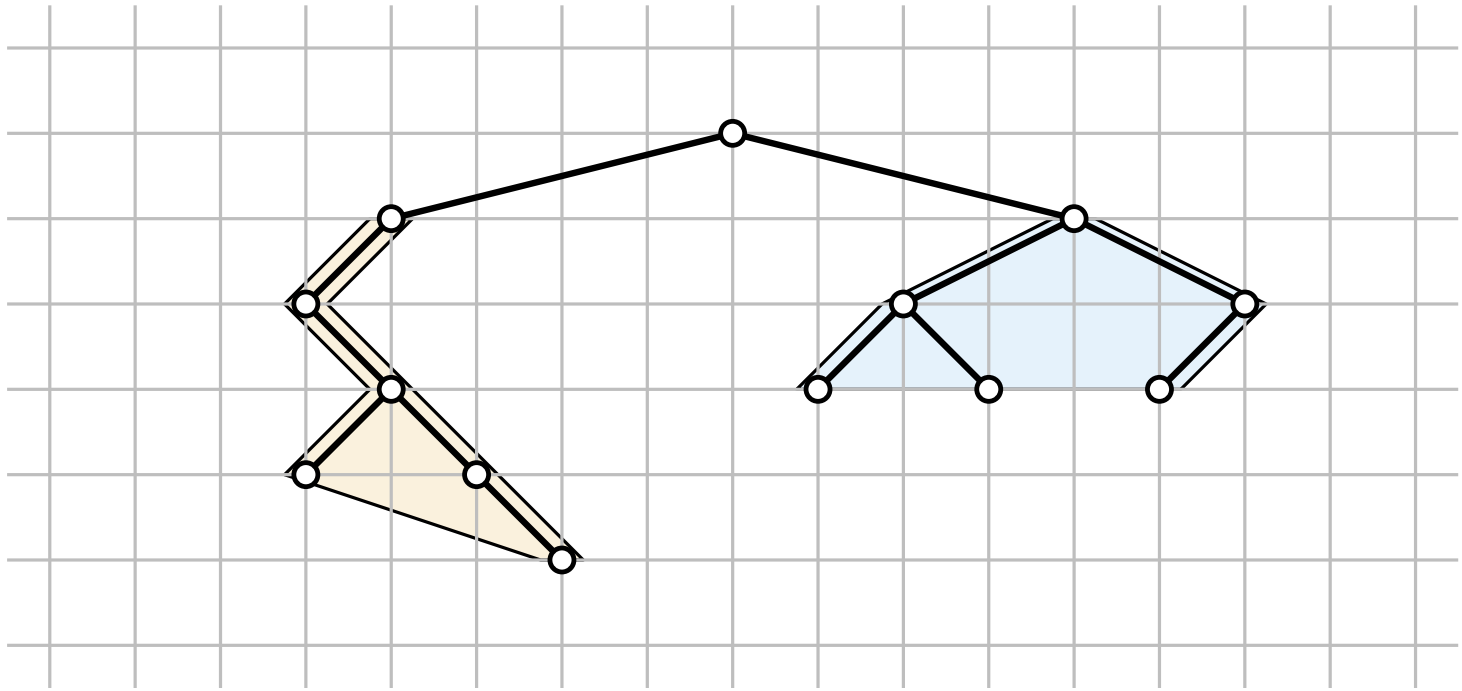
Improving Area and Parent Position

Recursive algorithm of Reingold/Tilford (1981):

Base case: leaf v, place at layer $-\text{depth}(v)$ ◙

Divide: inner node v, recursively draw $T_l(v)$ and $T_r(v)$

Conquer: place subtree drawings next to each other with distance 2 or 3 and center v above $lc(v)$ and $rc(v)$



parent centered ✓ smaller area?

Improvement: use tight **contour** for placement

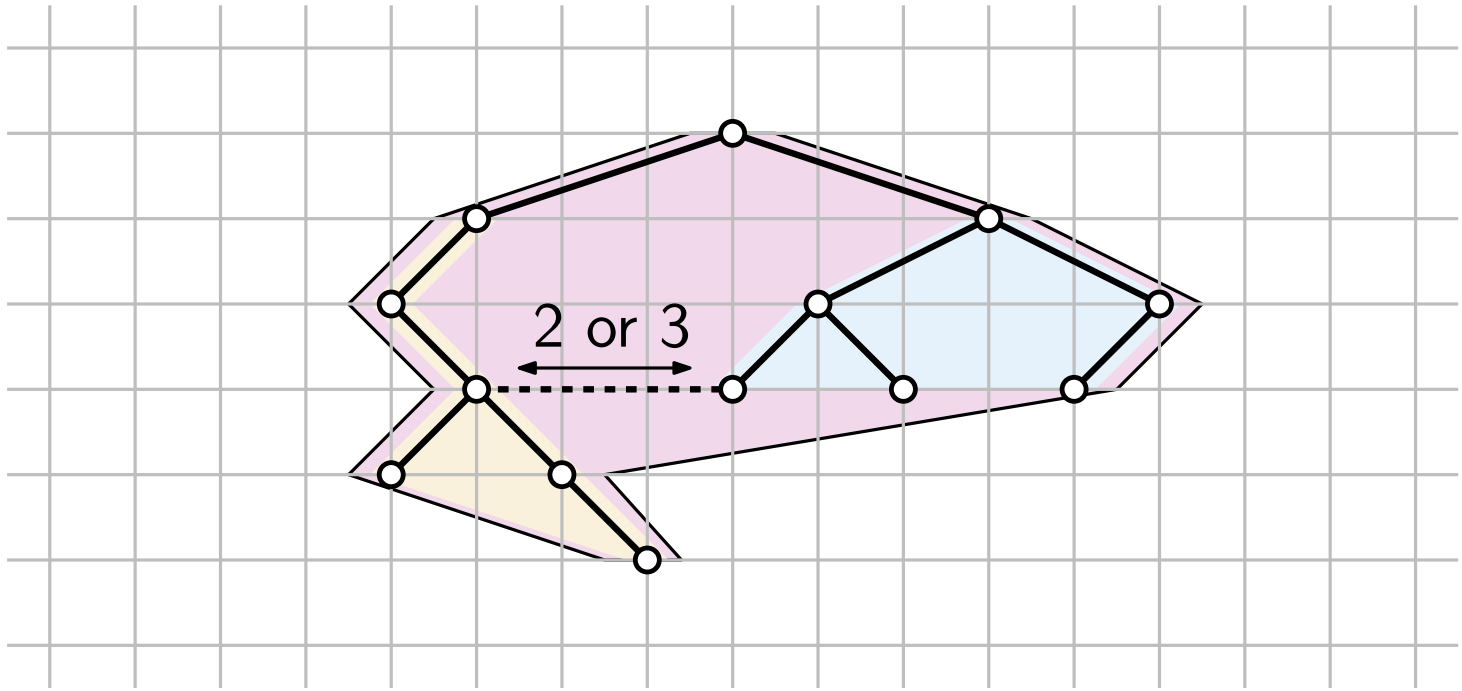
Improving Area and Parent Position

Recursive algorithm of Reingold/Tilford (1981):

Base case: leaf v, place at layer $-\text{depth}(v)$ ◙

Divide: inner node v, recursively draw $T_l(v)$ and $T_r(v)$

Conquer: place subtree drawings next to each other with distance 2 or 3 and center v above $lc(v)$ and $rc(v)$

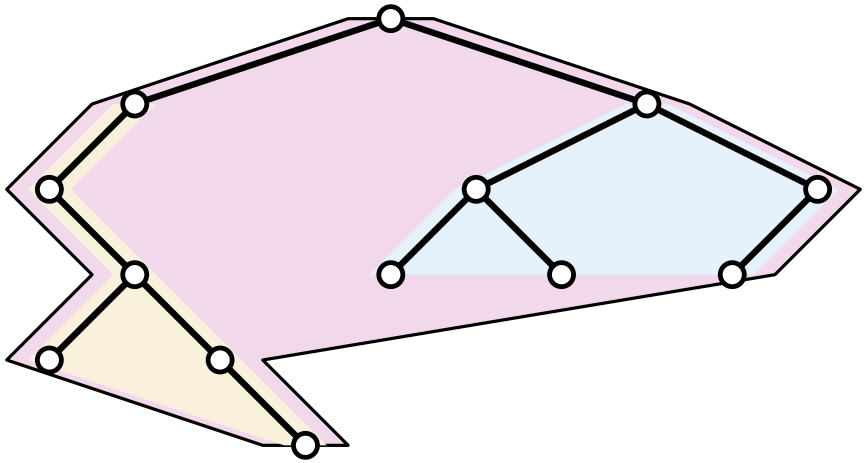


parent centered ✓ smaller area?

Improvement: use tight **contour** for more compact placement

Reingold/Tilford: 2-Phase Algorithm

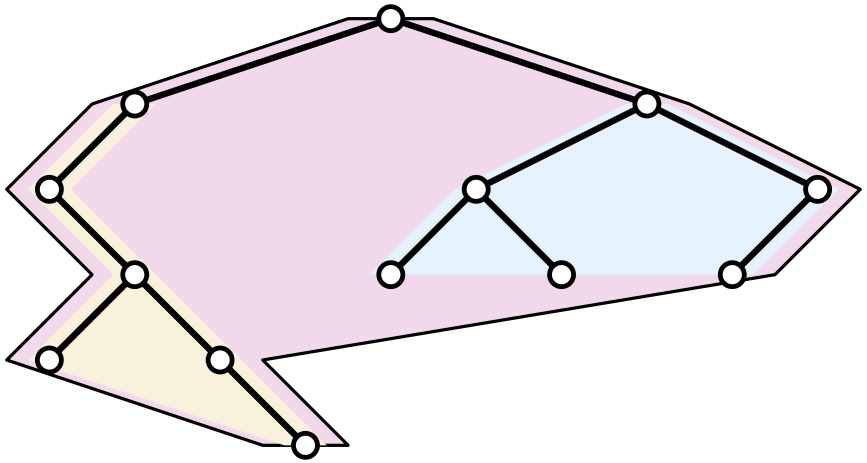
1. **postorder/bottom-up:**
   - compute contours and x-offsets to parent
2. **preorder/top-down:**
   - assign absolute x-coordinates



Contour data structure:
linked list of nodes on contour (left and right) with offsets to parent

Reingold/Tilford: 2-Phase Algorithm

1. **postorder/bottom-up:**
   - compute contours and x-offsets to parent
2. **preorder/top-down:**
   - assign absolute x-coordinates

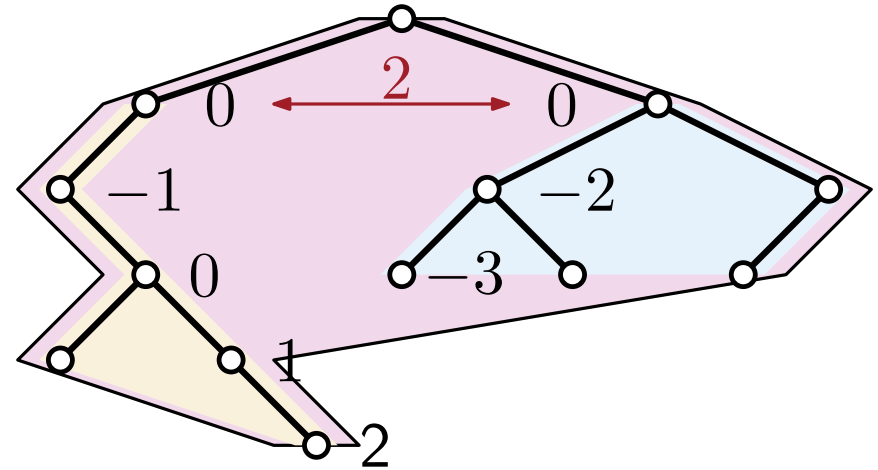


Contour data structure:
linked list of nodes on contour (left and right) with offsets to parent

Phase 1:
1. traverse right contour of $T_l(v)$ and left contour of $T_r(v)$
2. compute $d_v =$ accumulated minimum distance per level
3. define x-offset$(lc(v)) = -\lceil d_v/2 \rceil$, x-offset$(rc(v)) = \lceil d_v/2 \rceil$
4. define left contour of $T(v)$
   - $v +$ left contour of $T_l(v)$
   - if needed part of left contour of $T_r(v)$
5. define right contour of $T(v)$ analogously
   - $v +$ right contour of $T_r(v)$
   - if needed part of right contour of $T_l(v)$

Reingold/Tilford: 2-Phase Algorithm

1. **postorder/bottom-up:**
   - compute contours and x-offsets to parent
2. **preorder/top-down:**
   - assign absolute x-coordinates

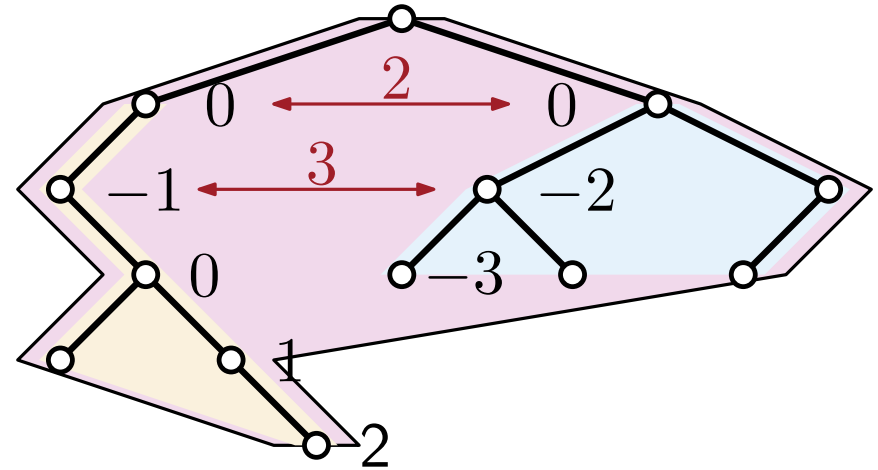


Contour data structure:
linked list of nodes on contour (left and right) with offsets to parent

Phase 1:
1. traverse right contour of $T_l(v)$ and left contour of $T_r(v)$
2. compute $d_v =$ accumulated minimum distance per level
3. define x-offset$(lc(v)) = -\lceil d_v/2 \rceil$, x-offset$(rc(v)) = \lceil d_v/2 \rceil$
4. define left contour of $T(v)$
   - $v +$ left contour of $T_l(v)$
   - if needed part of left contour of $T_r(v)$
5. define right contour of $T(v)$ analogously
   - $v +$ right contour of $T_r(v)$
   - if needed part of right contour of $T_l(v)$

Reingold/Tilford: 2-Phase Algorithm

1. **postorder/bottom-up:**
   - compute contours and x-offsets to parent
2. **preorder/top-down:**
   - assign absolute x-coordinates

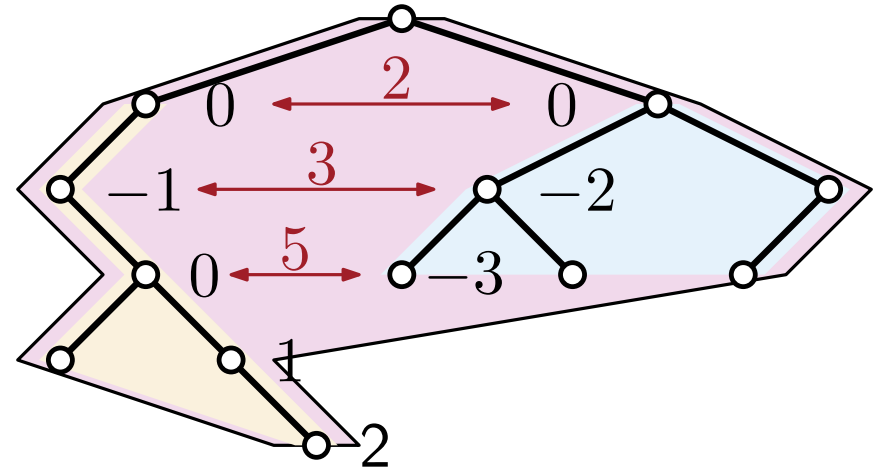


Contour data structure:
linked list of nodes on contour (left and right) with offsets to parent

Phase 1:
1. traverse right contour of $T_l(v)$ and left contour of $T_r(v)$
2. compute $d_v =$ accumulated minimum distance per level
3. define x-offset$(lc(v)) = -\lceil d_v/2 \rceil$, x-offset$(rc(v)) = \lceil d_v/2 \rceil$
4. define left contour of $T(v)$
   - $v +$ left contour of $T_l(v)$
   - if needed part of left contour of $T_r(v)$
5. define right contour of $T(v)$ analogously
   - $v +$ right contour of $T_r(v)$
   - if needed part of right contour of $T_l(v)$

Reingold/Tilford: 2-Phase Algorithm

1. **postorder/bottom-up:**
   - compute contours and x-offsets to parent
2. **preorder/top-down:**
   - assign absolute x-coordinates

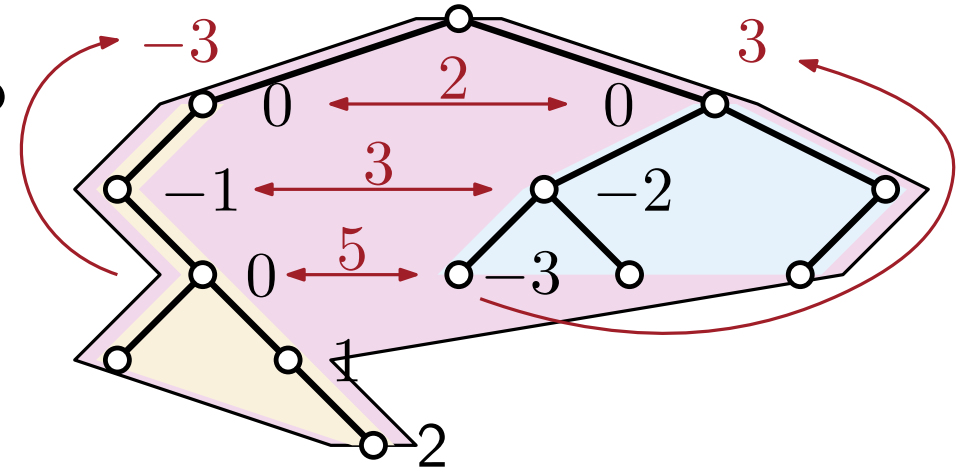


Contour data structure:
linked list of nodes on contour (left and right) with offsets to parent

Phase 1:
1. traverse right contour of $T_l(v)$ and left contour of $T_r(v)$
2. compute d_v = accumulated minimum distance per level
3. define x-offset$(lc(v)) = -\lceil d_v/2 \rceil$, x-offset$(rc(v)) = \lceil d_v/2 \rceil$
4. define left contour of $T(v)$
   - v + left contour of $T_l(v)$
   - if needed part of left contour of $T_r(v)$
5. define right contour of $T(v)$ analogously
   - v + right contour of $T_r(v)$
   - if needed part of right contour of $T_l(v)$

Reingold/Tilford: 2-Phase Algorithm

1. **postorder/bottom-up:**
   - compute contours and x-offsets to parent
2. **preorder/top-down:**
   - assign absolute x-coordinates

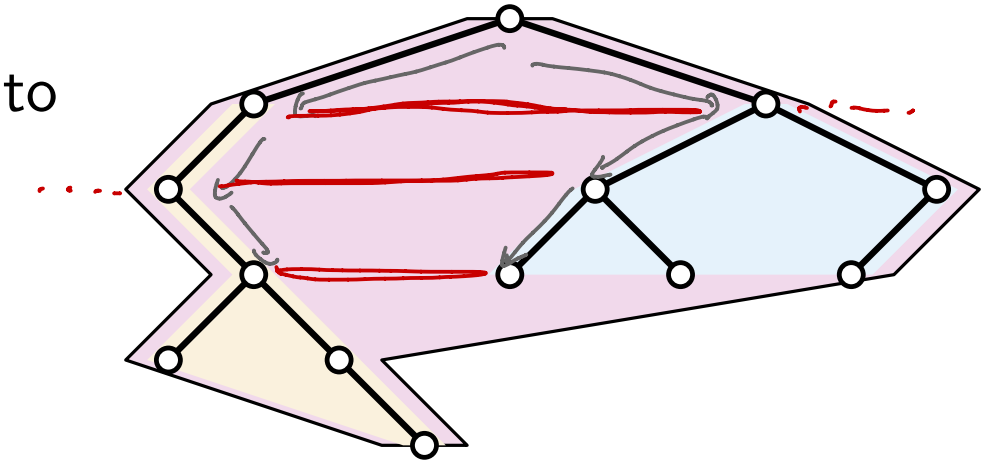


Contour data structure:
linked list of nodes on contour (left and right) with offsets to parent

Phase 1:
1. traverse right contour of $T_l(v)$ and left contour of $T_r(v)$
2. compute d_v = accumulated minimum distance per level
3. define x-offset$(lc(v)) = -\lceil d_v/2 \rceil$, x-offset$(rc(v)) = \lceil d_v/2 \rceil$
4. define left contour of $T(v)$
   - v + left contour of $T_l(v)$
   - if needed part of left contour of $T_r(v)$
5. define right contour of $T(v)$ analogously
   - v + right contour of $T_r(v)$
   - if needed part of right contour of $T_l(v)$

Reingold/Tilford: 2-Phase Algorithm

1. **postorder/bottom-up:**
   - compute contours and x-offsets to parent
2. **preorder/top-down:**
   - assign absolute x-coordinates



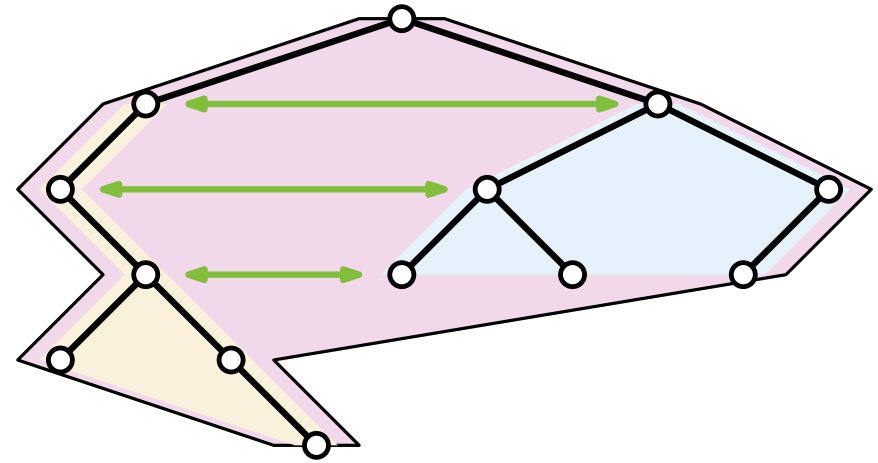
Contour data structure:
linked list of nodes on contour (left and right) with offsets to parent

Phase 1:
1. traverse right contour of $T_l(v)$ and left contour of $T_r(v)$
2. compute $d_v =$ accumulated minimum distance per level
3. define x-offset$(lc(v)) = -\lceil d_v/2 \rceil$, x-offset$(rc(v)) = \lceil d_v/2 \rceil$
4. define left contour of $T(v)$
   - $v +$ left contour of $T_l(v)$
   - if needed part of left contour of $T_r(v)$
5. define right contour of $T(v)$ analogously
   - $v +$ right contour of $T_r(v)$
   - if needed part of right contour of $T_l(v)$

Running time: postorder $+ \sum_v \min\{\text{height}(T_l(v), \text{height}(T_r(v)\}$

Reingold/Tilford: 2-Phase Algorithm

1. **postorder/bottom-up:**
   - compute contours and x-offsets to parent
2. **preorder/top-down:**
   - assign absolute x-coordinates



Contour data structure:
linked list of nodes on contour (left and right) with offsets to parent

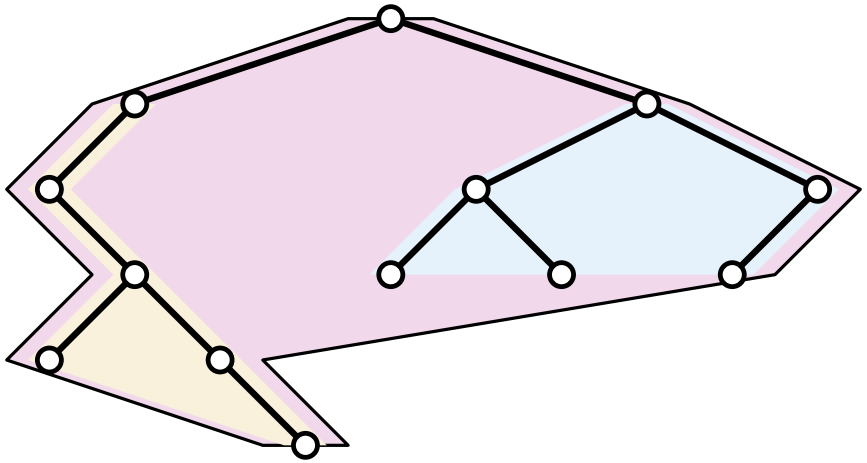
Phase 1:
1. traverse right contour of $T_l(v)$ and left contour of $T_r(v)$
2. compute $d_v =$ accumulated minimum distance per level
3. define x-offset$(lc(v)) = -\lceil d_v/2 \rceil$, x-offset$(rc(v)) = \lceil d_v/2 \rceil$
4. define left contour of $T(v)$
   - $v +$ left contour of $T_l(v)$
   - if needed part of left contour of $T_r(v)$
5. define right contour of $T(v)$ analogously
   - $v +$ right contour of $T_r(v)$
   - if needed part of right contour of $T_l(v)$

– each node compared at most twice (once left, once right)
– sums up to $2n$

Running time: postorder $+ \sum_v \min\{\text{height}(T_l(v), \text{height}(T_r(v)\}$ $= O(n)$

Reingold/Tilford: 2-Phase Algorithm

1. **postorder/bottom-up:**
   - compute contours and x-offsets to parent
2. **preorder/top-down:**
   - assign absolute x-coordinates



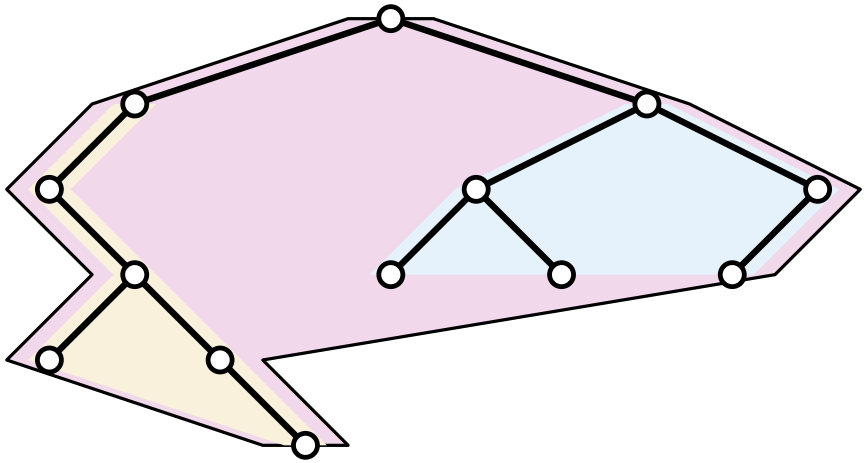
Contour data structure:
linked list of nodes on contour (left and right) with offsets to parent

Phase 2: set $x(\text{root}) = 0$
for all $v \in T$ in preorder
1. set $y(v) = -\text{depth}(v)$
2. set $x(lc(v)) = x(v) + \text{x-offset}(lc(v))$
3. set $x(rc(v)) = x(v) + \text{x-offset}(rc(v))$

Running time: preorder $= O(n)$

Reingold/Tilford: 2-Phase Algorithm

1. **postorder/bottom-up:**
   - compute contours and x-offsets to parent
2. **preorder/top-down:**
   - assign absolute x-coordinates



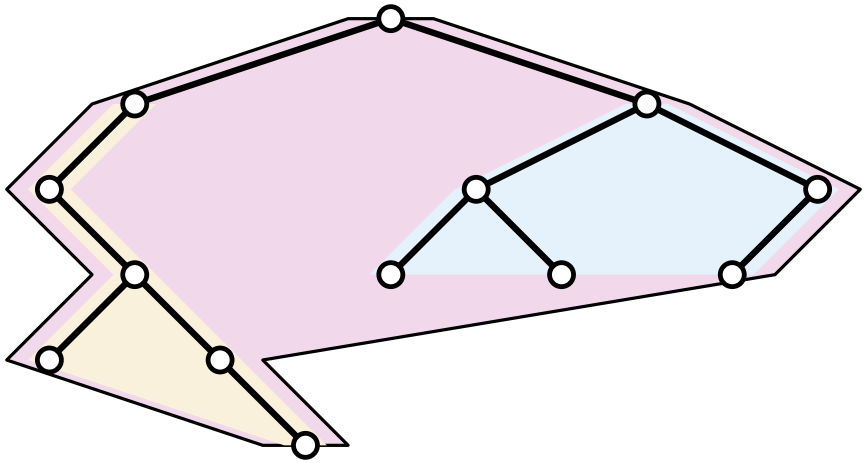
Contour data structure:
linked list of nodes on contour (left and right) with offsets to parent

Theorem: Given a rooted binary tree a layered, straight-line tree drawing with the following properties can be computed in $O(n)$ time:

- parents centered above children
- child order preserved
- isomorphic subtrees are drawn the same
- integer grid coordinates
- area is height$(T) \cdot O(n) = O(n^2)$

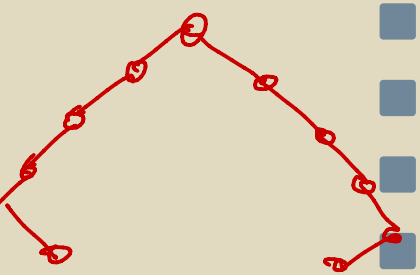
Reingold/Tilford: 2-Phase Algorithm

1. **postorder/bottom-up:**
   - compute contours and x-offsets to parent
2. **preorder/top-down:**
   - assign absolute x-coordinates

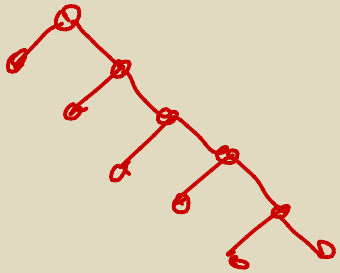


Contour data structure:
linked list of nodes on contour (left and right) with offsets to parent

Theorem: Given a rooted binary tree a layered, straight-line tree drawing with the following properties can be computed in $O(n)$ time:



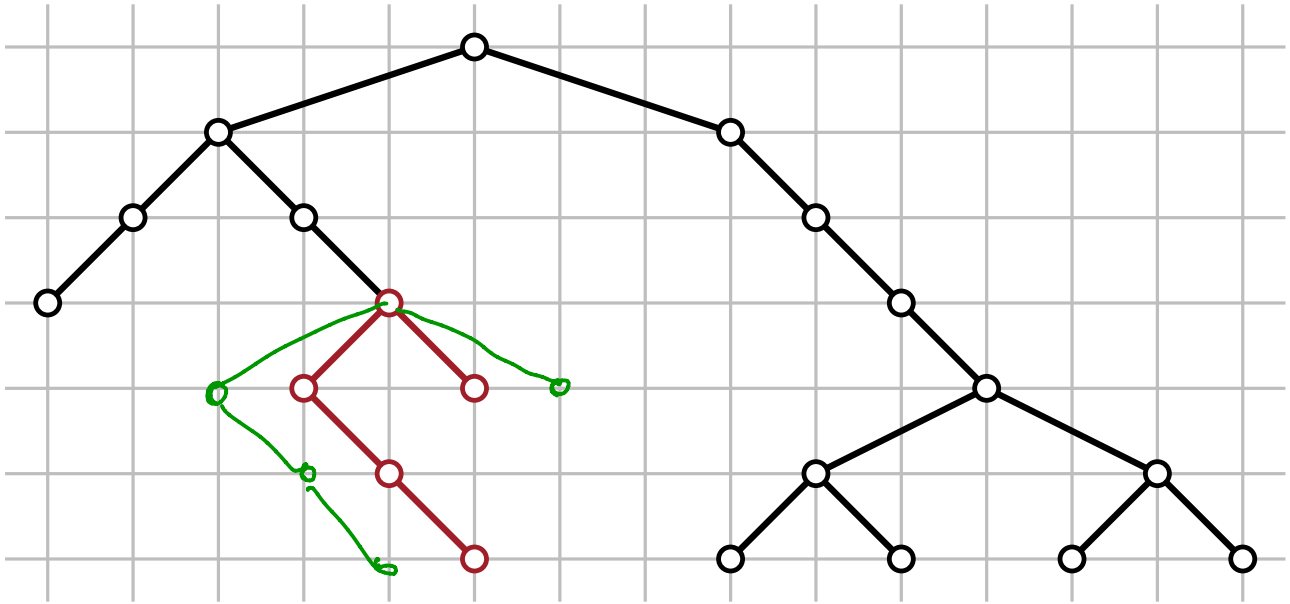
- parents centered above children
- child order preserved
- isomorphic subtrees are drawn the same
- integer grid coordinates
- area is height$(T) \cdot O(n) = O(n^2)$



What is a worst-case example for the area?

Reingold/Tilford: Problems and Extensions

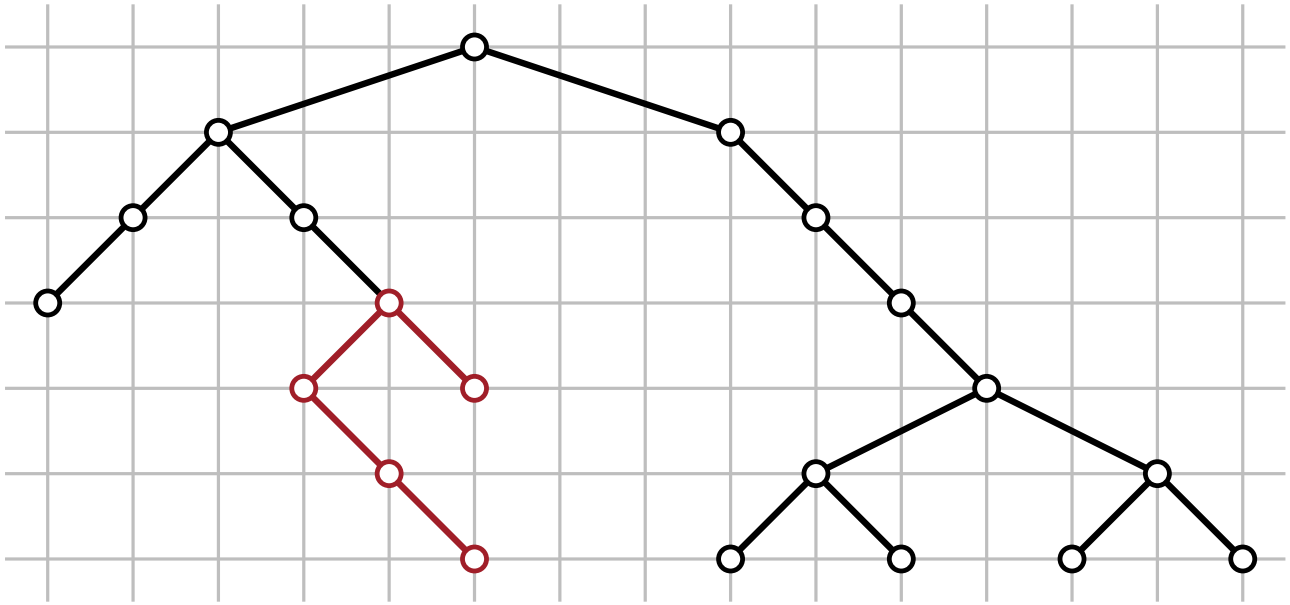
Width minimization:



width 14

Reingold/Tilford: Problems and Extensions

Width minimization:

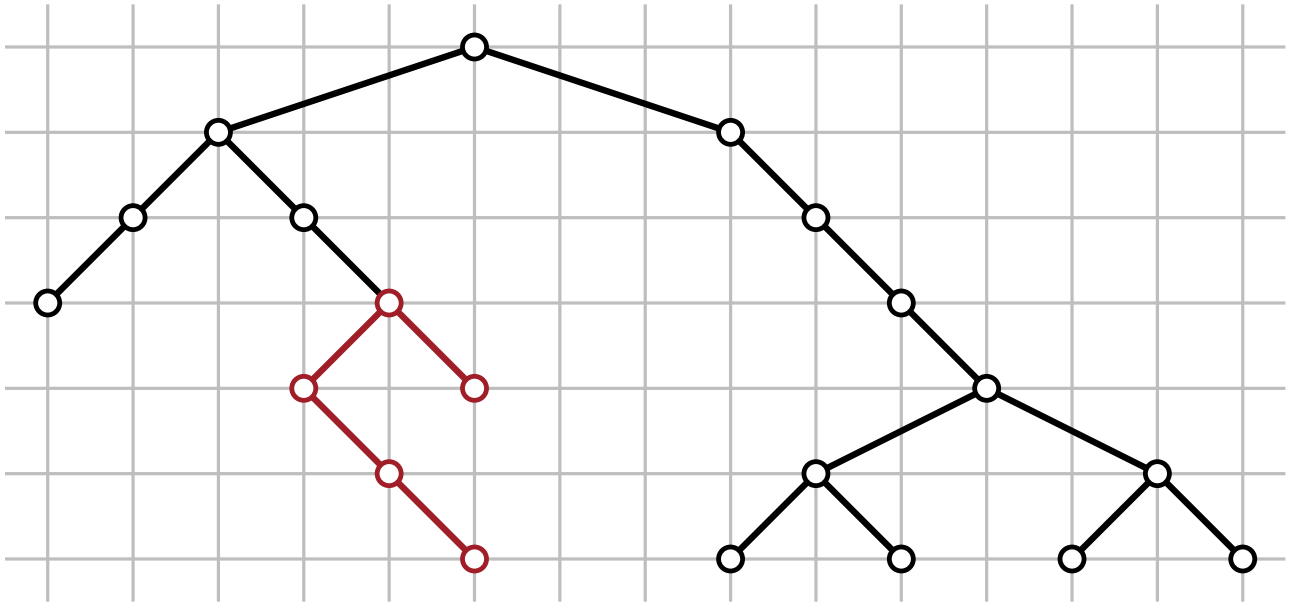


width 14

Could this be done better?

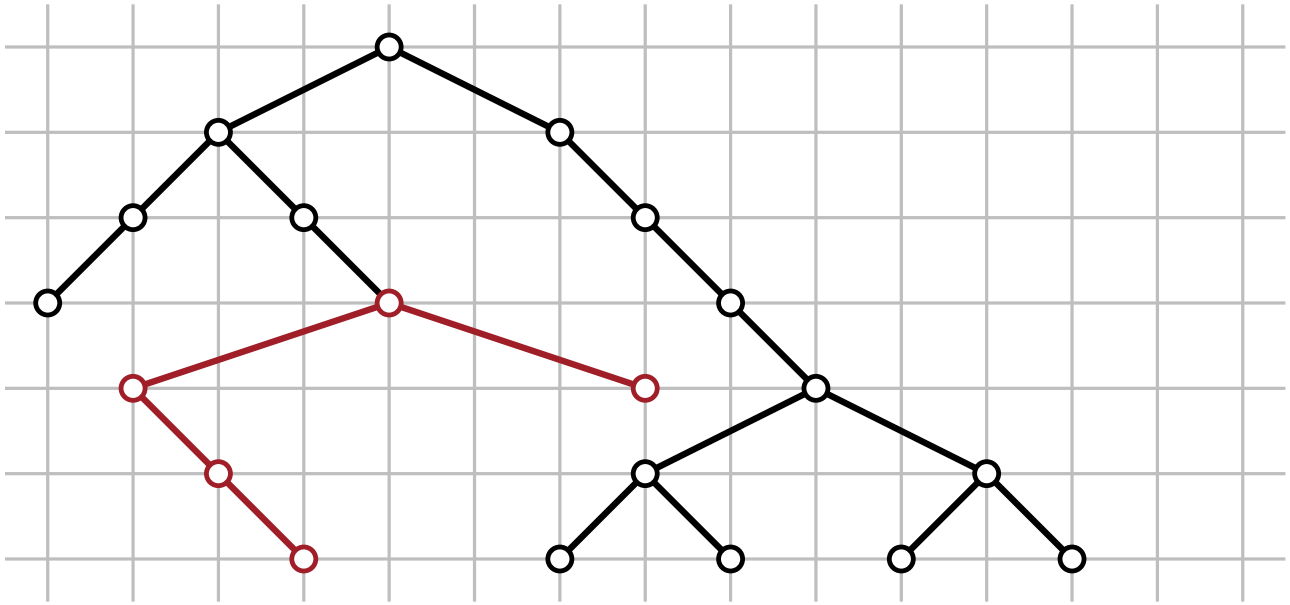
Reingold/Tilford: Problems and Extensions

Width minimization:



width 14

Could this be done better?



Yes, width 12

- dependencies among subtrees
- isomorphic trees drawn differently
- width minimization on grid is NP-hard!

[Supowit, Reingold '83]

Reingold/Tilford: Problems and Extensions

Width minimization: [Supowit, Reingold '83]

- recursive divide & conquer procedure unable to minimize width
- problem is NP-hard on grid

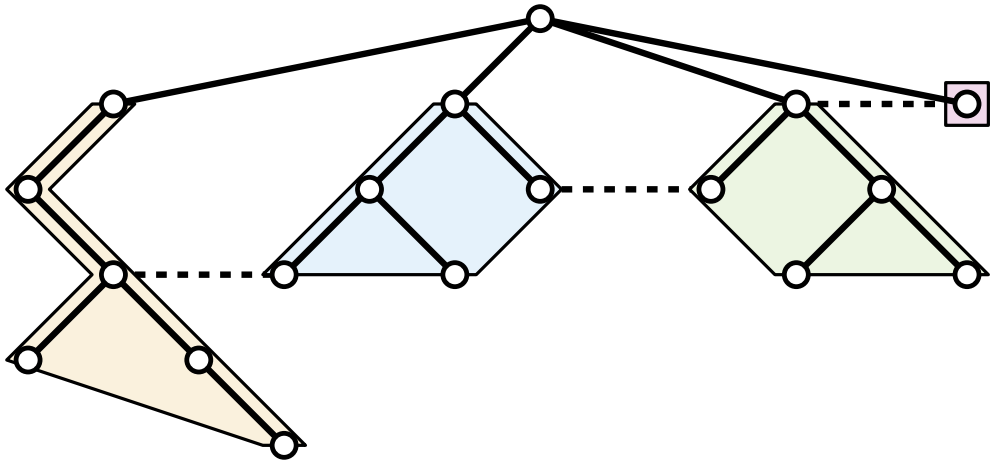
Reingold/Tilford: Problems and Extensions

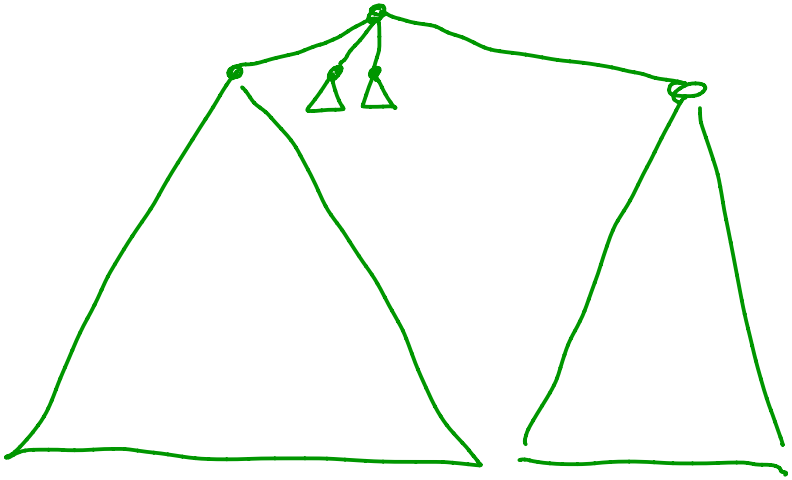
Width minimization: [Supowit, Reingold '83]

- recursive divide & conquer procedure unable to minimize width
- problem is NP-hard on grid

Non-binary Trees: [Walker '90], [Buchheim et al. '02]

- recursive algorithm easily extends
- additional post-processing to improve horizontal balancing
- remains $O(n)$ time





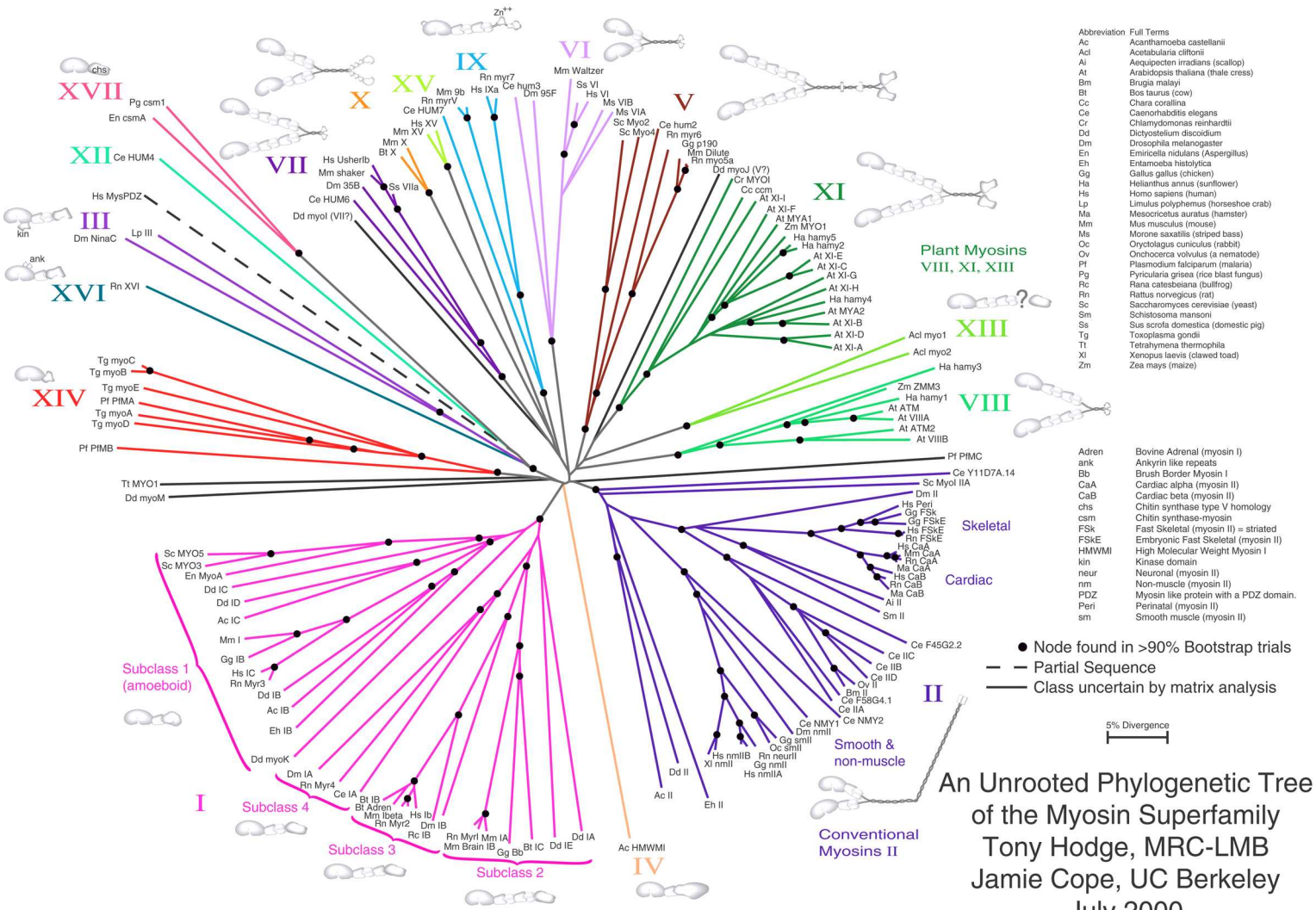
Overview

Layered drawings of rooted trees

Radial tree drawings

Compact orthogonal drawings of rooted trees

Radial Tree Drawings: Examples

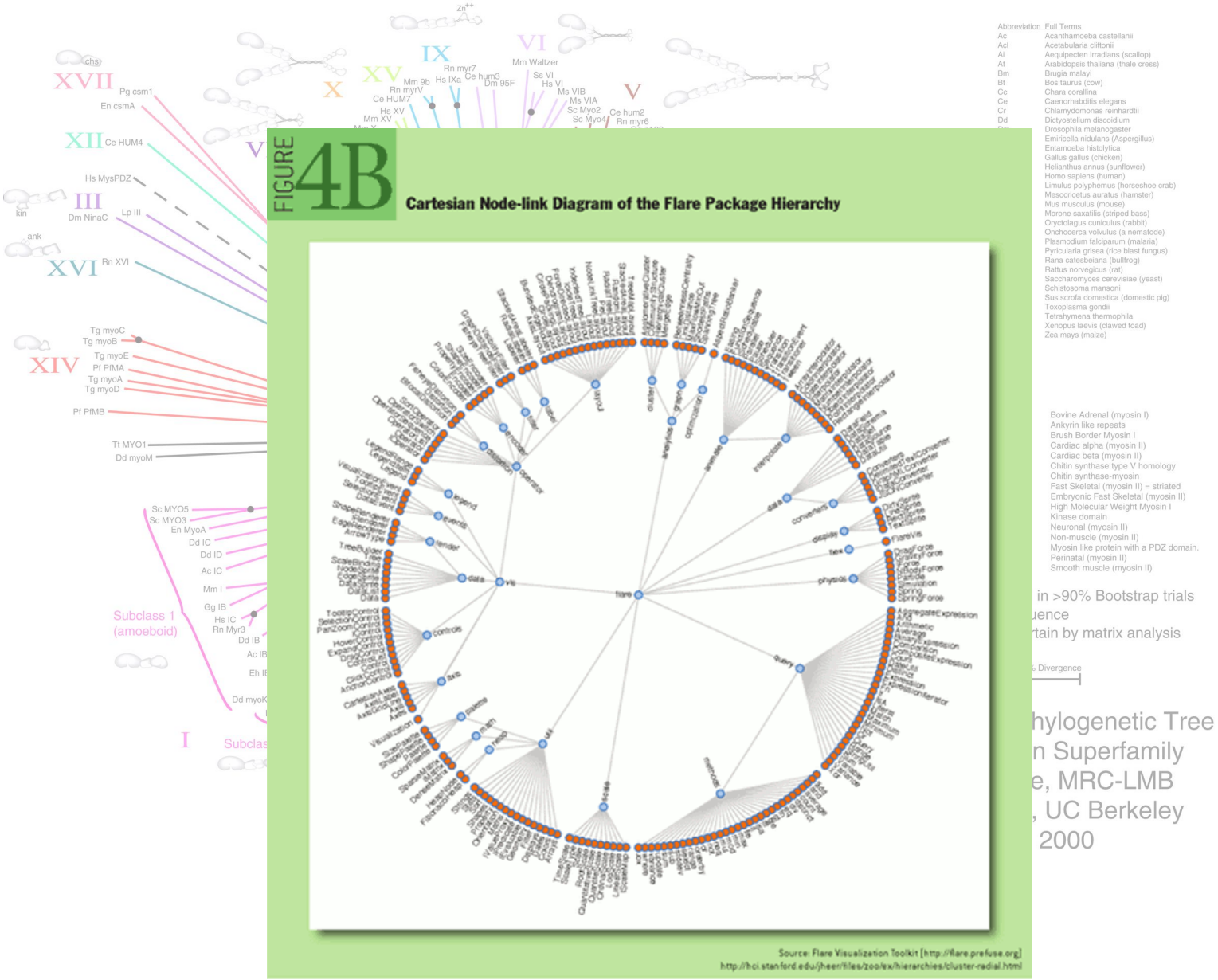


Abbreviation	Full Terms
Ac	Acanthamoeba castellanii
Acl	Acetabularia clifftonii
Ai	Aequipecten irradians (scallop)
At	Arabidopsis thaliana (thale cress)
Bm	Brugia malayi
Bt	Bos taurus (cow)
Cc	Chara corallina
Ce	Caenorhabditis elegans
Cr	Chlamydomonas reinhardtii
Dd	Dictyostelium discoidium
Dm	Drosophila melanogaster
En	Emericella nidulans (Aspergillus)
Eh	Entamoeba histolytica
Gg	Gallus gallus (chicken)
Ha	Helianthus annus (sunflower)
Hs	Homo sapiens (human)
Lp	Limulus polyphemus (horseshoe crab)
Ma	Mesocricetus auratus (hamster)
Mm	Mus musculus (mouse)
Ms	Morone saxatilis (striped bass)
Oc	Oryctolagus cuniculus (rabbit)
Ov	Onchocerca volvulus (a nematode)
Pf	Plasmodium falciparum (malaria)
Pg	Pyricularia grisea (rice blast fungus)
Rc	Rana catesbeiana (bullfrog)
Rn	Rattus norvegicus (rat)
Sc	Saccharomyces cerevisiae (yeast)
Sm	Schistosoma mansoni
Ss	Sus scrofa domestica (domestic pig)
Tg	Toxoplasma gondii
Tt	Tetrahymena thermophila
Xl	Xenopus laevis (clawed toad)
Zm	Zea mays (maize)

Adren	Bovine Adrenal (myosin I)
ank	Ankyrin like repeats
Bb	Brush Border Myosin I
CaA	Cardiac alpha (myosin II)
CaB	Cardiac beta (myosin II)
chs	Chitin synthase type V homology
csm	Chitin synthase-myosin
FSk	Fast Skeletal (myosin II) = striated
FSkE	Embryonic Fast Skeletal (myosin II)
HMWMI	High Molecular Weight Myosin I
kin	Kinase domain
neur	Neuronal (myosin II)
nm	Non-muscle (myosin II)
PDZ	Myosin like protein with a PDZ domain.
Peri	Perinatal (myosin II)
sm	Smooth muscle (myosin II)

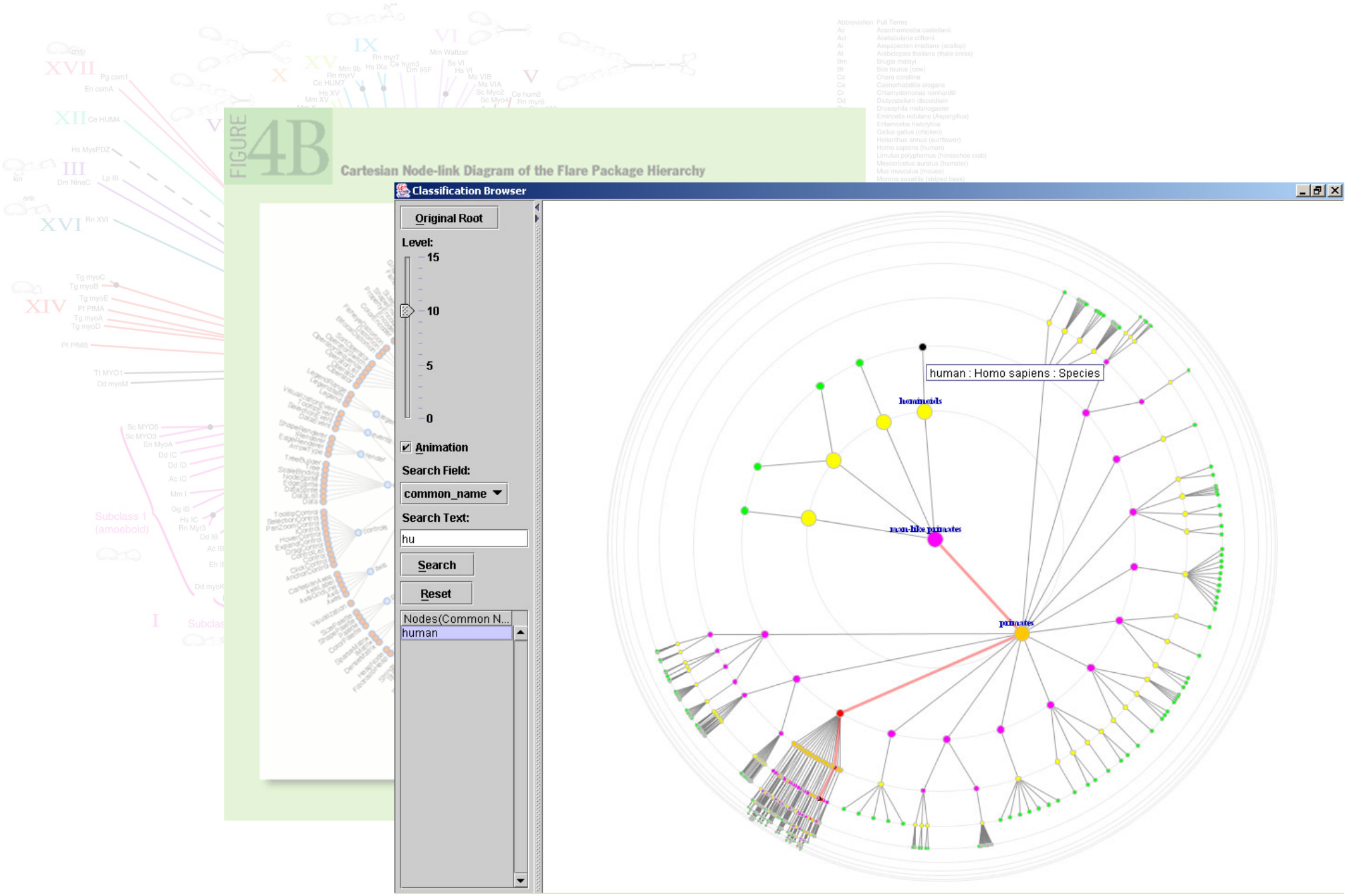
An Unrooted Phylogenetic Tree
of the Myosin Superfamily
Tony Hodge, MRC-LMB
Jamie Cope, UC Berkeley
July 2000

Radial Tree Drawings: Examples



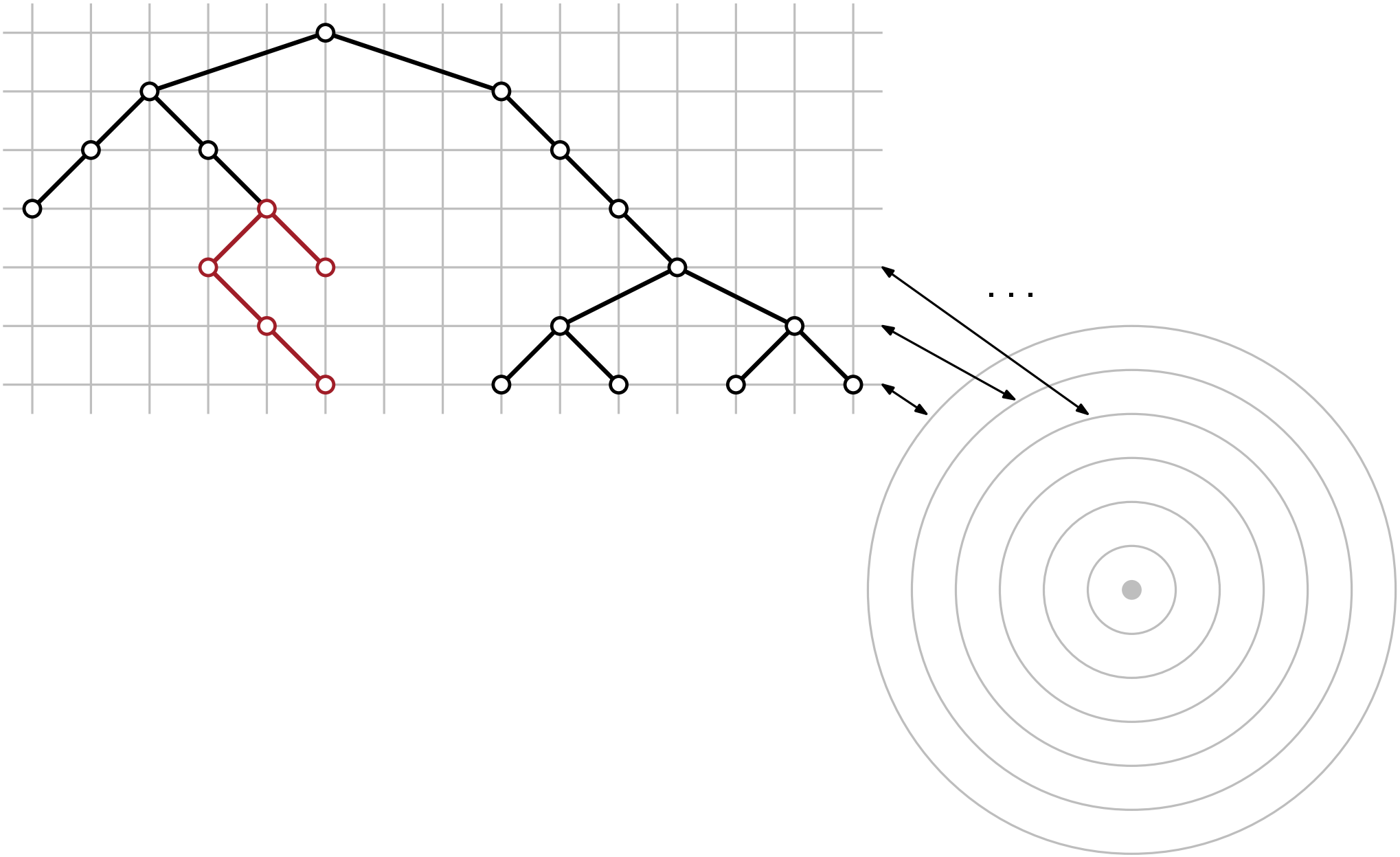
hylogenetic Tree
n Superfamily
e, MRC-LMB
, UC Berkeley
2000

Radial Tree Drawings: Examples



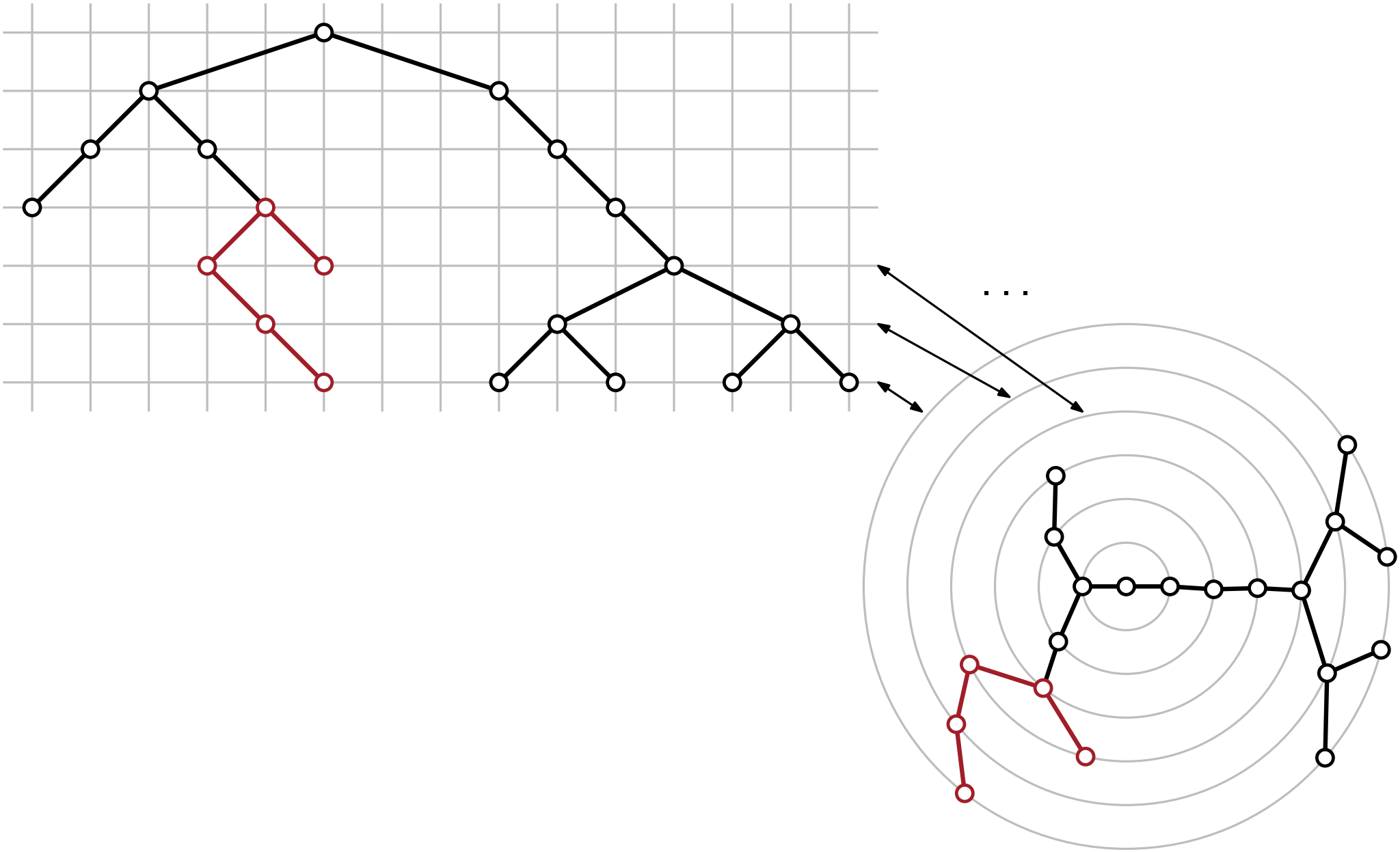
Radial Tree Drawings

Can we put layered drawings on rings?



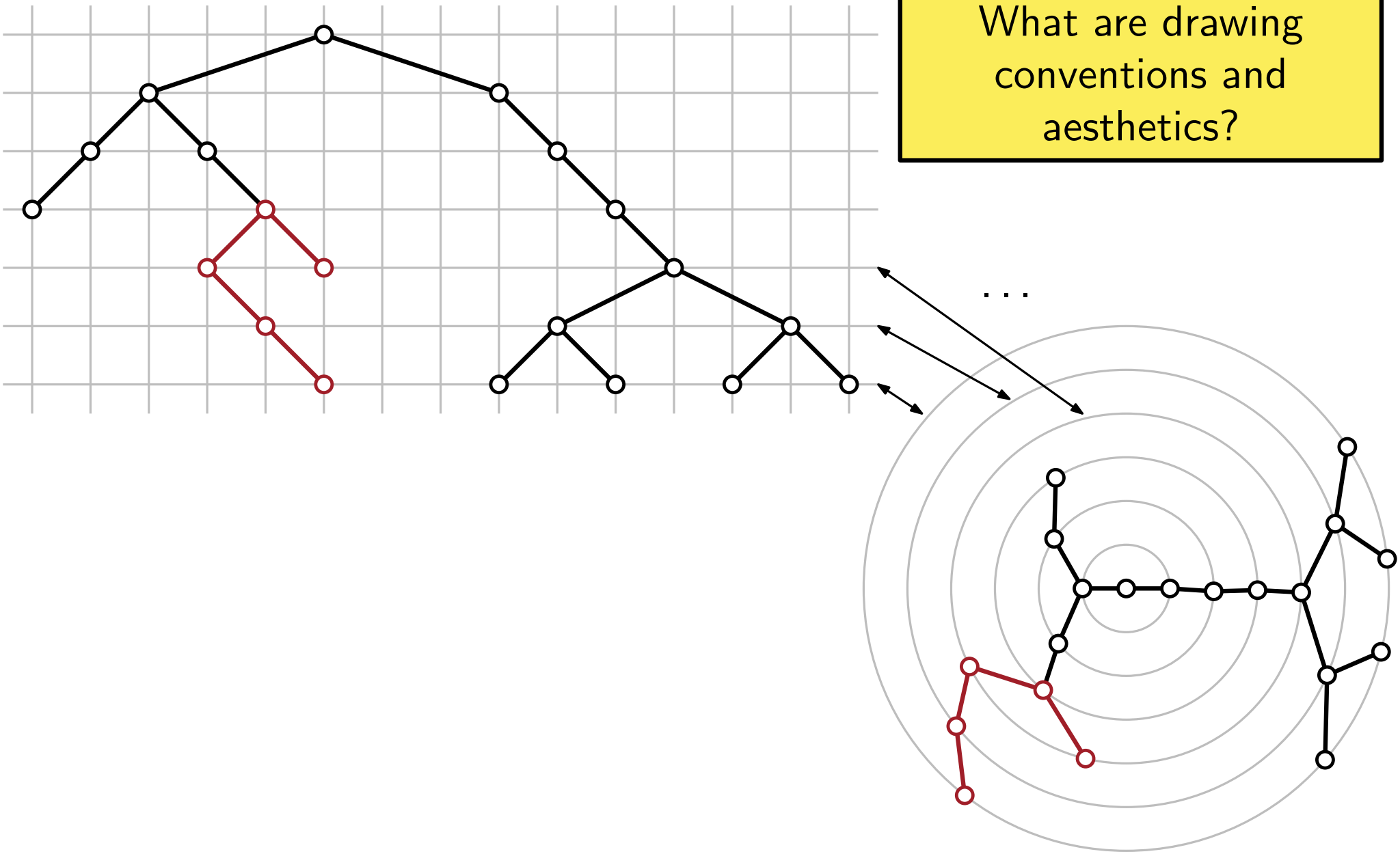
Radial Tree Drawings

Can we put layered drawings on rings?



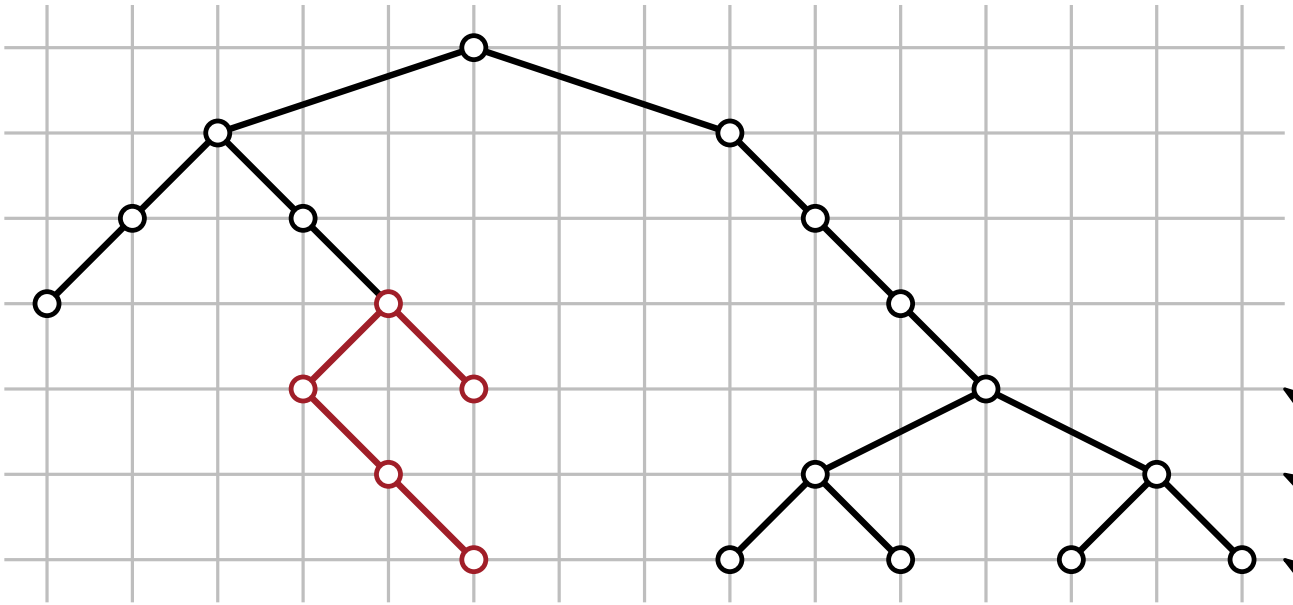
Radial Tree Drawings

Can we put layered drawings on rings?

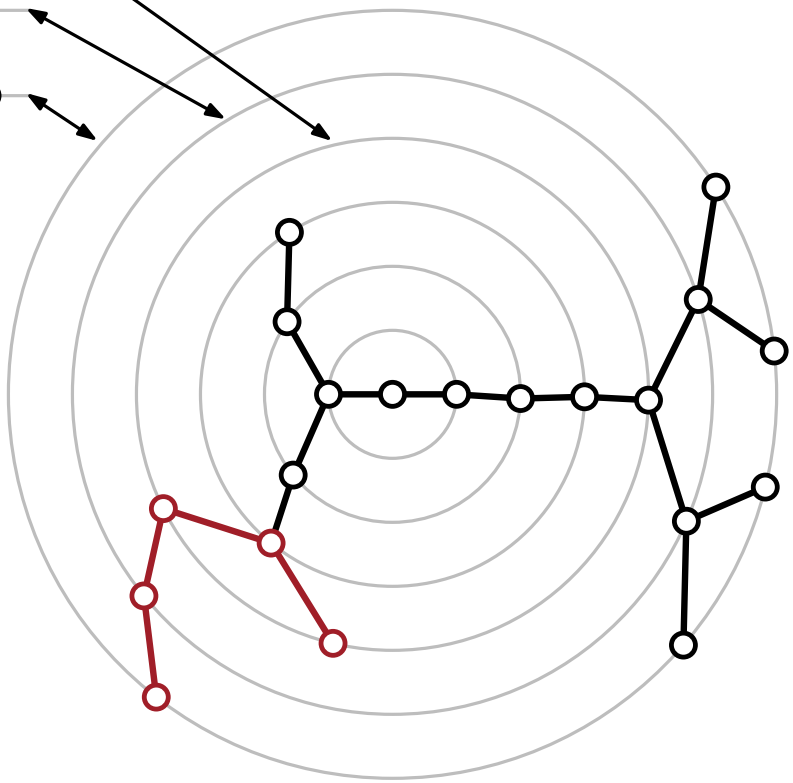


Radial Tree Drawings

Can we put layered drawings on rings?



What are drawing conventions and aesthetics?



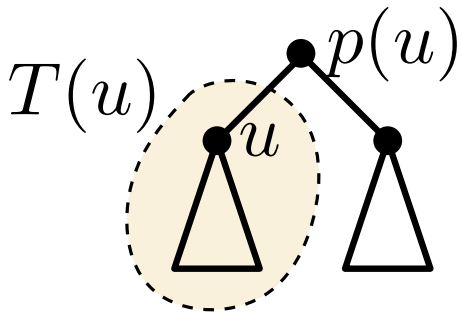
- layers are concentric circles
- straight-line edges
- no crossings
- balanced distribution of nodes
- often used for unrooted trees (pick central node as root)

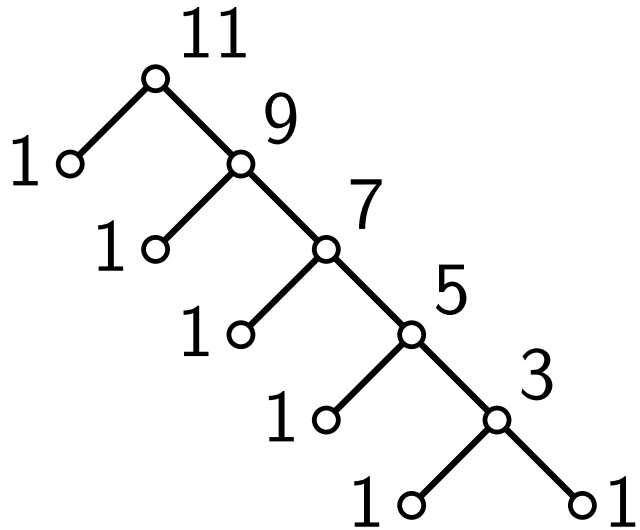
Radial Tree Drawings: Planarity

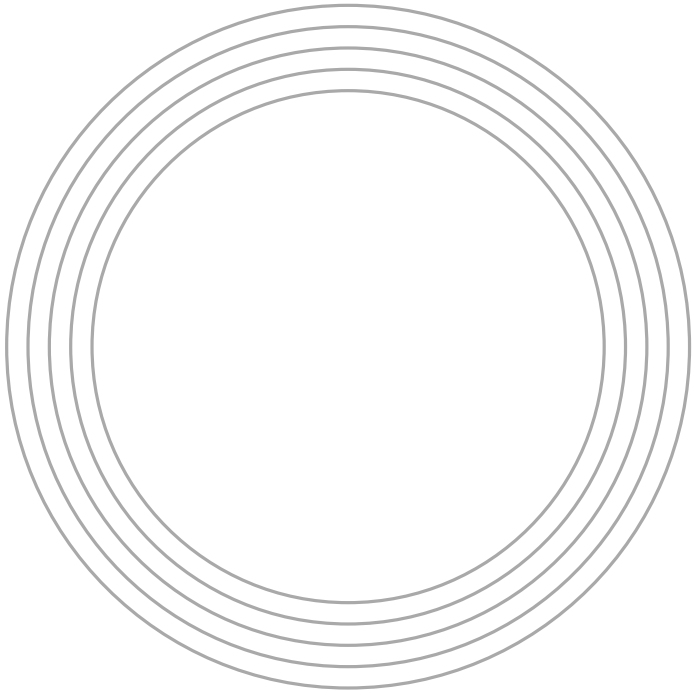
Distribute subtrees based on their node count (or leaf count)

Example: wedge angle τ_u for the subtree $T(u)$:

$$\tau_u = \frac{|T(u)|}{|T(p(u))|-1} \cdot \tau_{p(u)}; \text{ for root } r\text{: } \tau_r = 2\pi$$





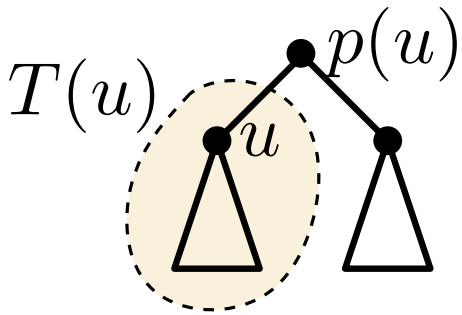


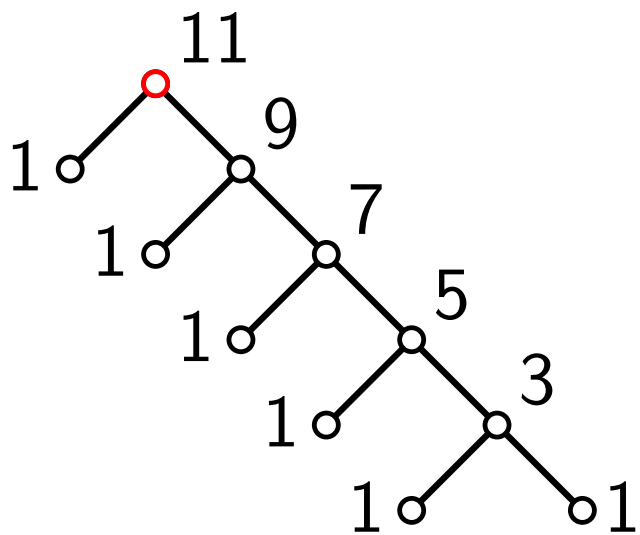
Radial Tree Drawings: Planarity

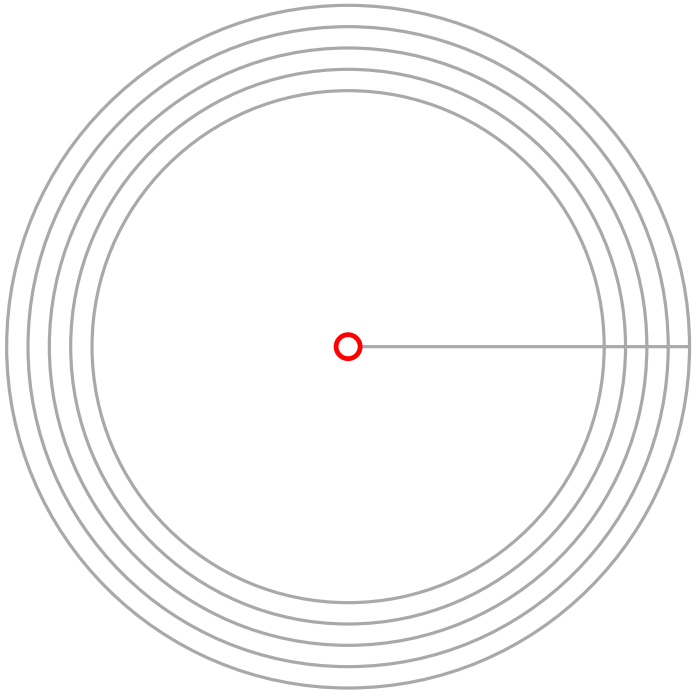
Distribute subtrees based on their node count (or leaf count)

Example: wedge angle τ_u for the subtree $T(u)$:

$$\tau_u = \frac{|T(u)|}{|T(p(u))|-1} \cdot \tau_{p(u)}; \text{ for root } r\text{: } \tau_r = 2\pi$$





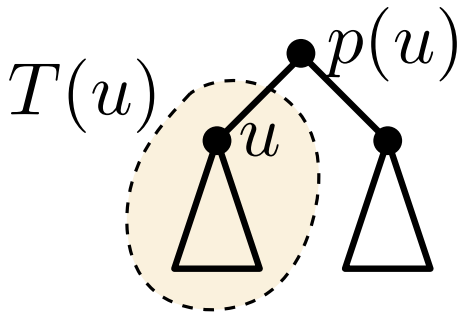


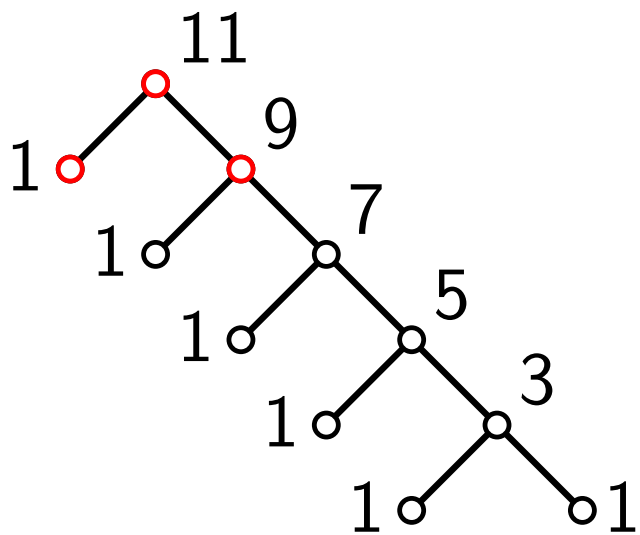
Radial Tree Drawings: Planarity

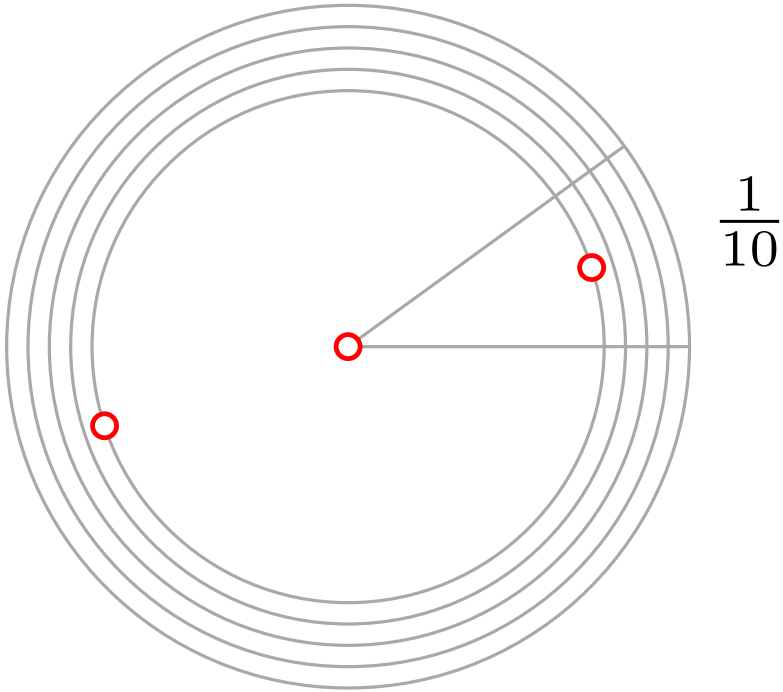
Distribute subtrees based on their node count (or leaf count)

Example: wedge angle τ_u for the subtree $T(u)$:

$$\tau_u = \frac{|T(u)|}{|T(p(u))|-1} \cdot \tau_{p(u)}; \text{ for root } r\text{: } \tau_r = 2\pi$$





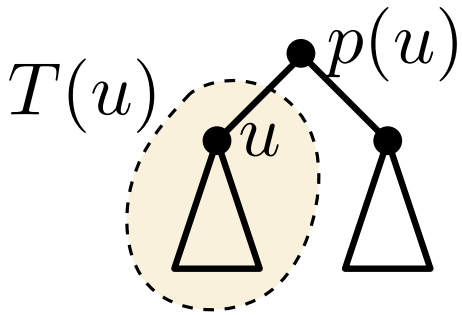


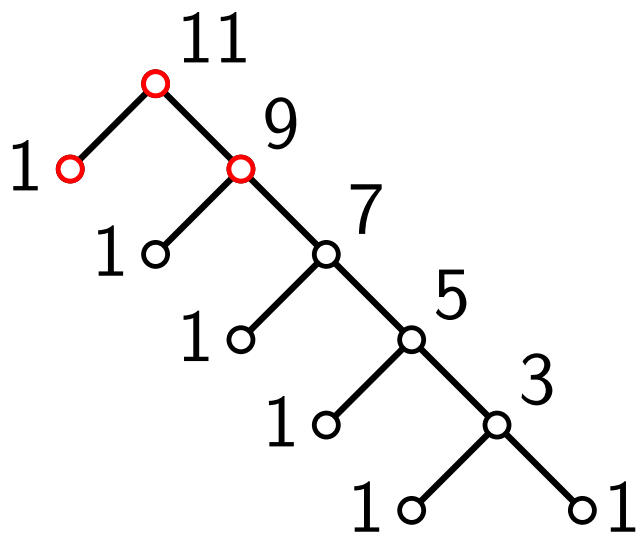
Radial Tree Drawings: Planarity

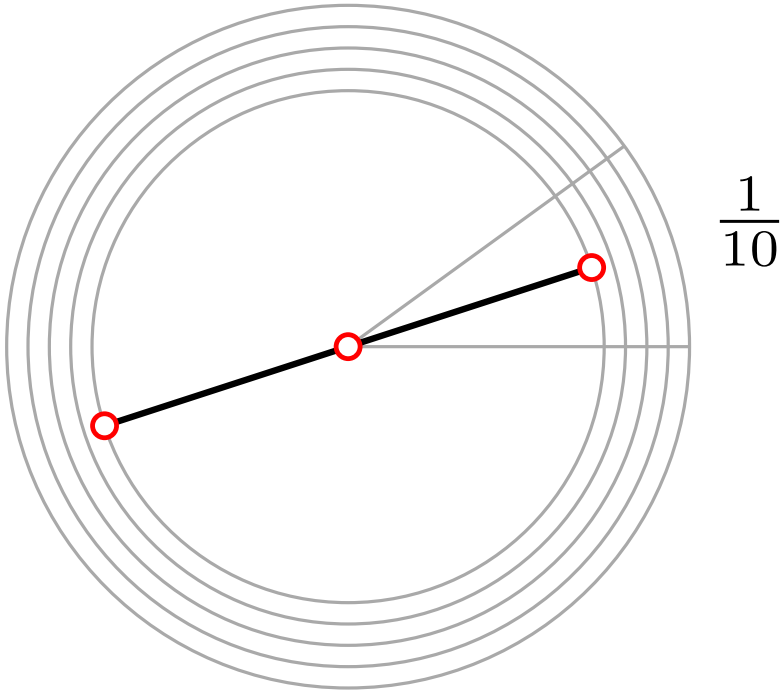
Distribute subtrees based on their node count (or leaf count)

Example: wedge angle τ_u for the subtree $T(u)$:

$$\tau_u = \frac{|T(u)|}{|T(p(u))|-1} \cdot \tau_{p(u)}; \text{ for root } r\text{: } \tau_r = 2\pi$$





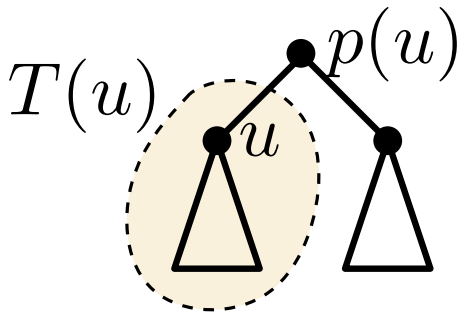


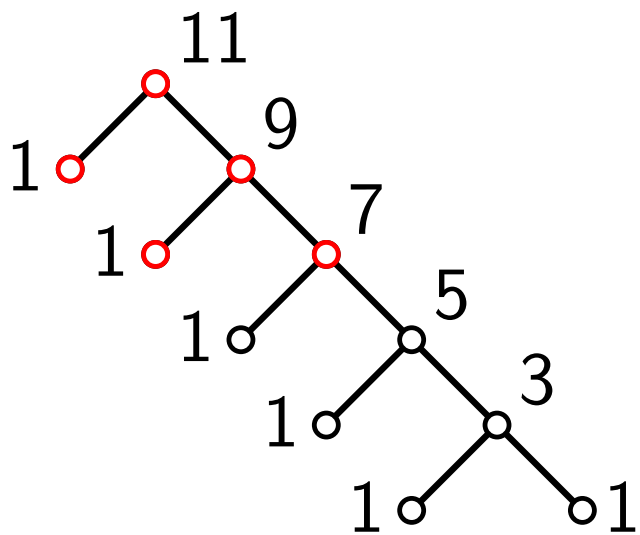
Radial Tree Drawings: Planarity

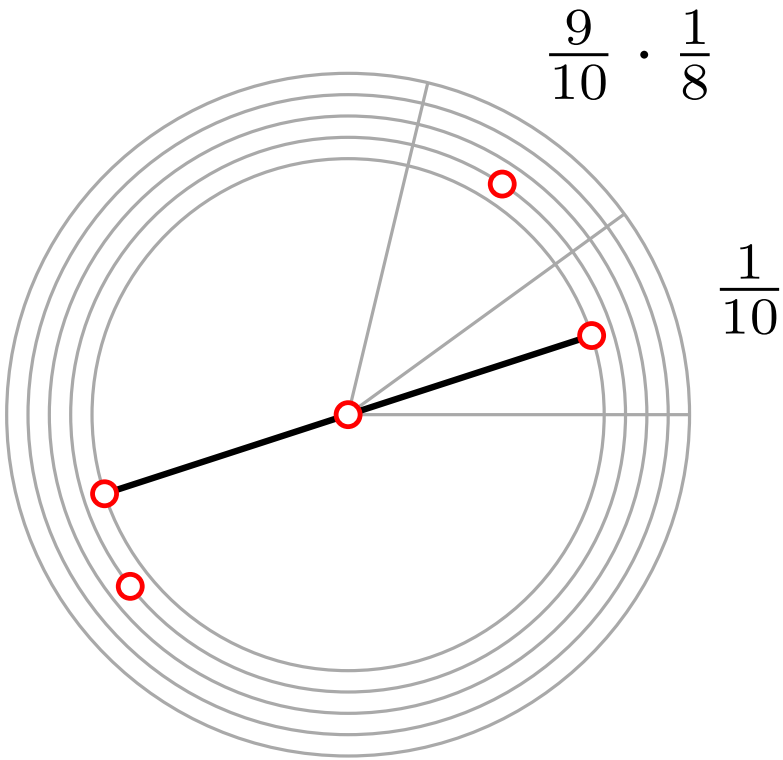
Distribute subtrees based on their node count (or leaf count)

Example: wedge angle τ_u for the subtree $T(u)$:

$$\tau_u = \frac{|T(u)|}{|T(p(u))|-1} \cdot \tau_{p(u)}; \text{ for root } r\text{: } \tau_r = 2\pi$$





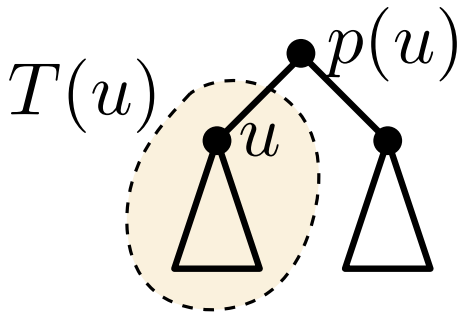


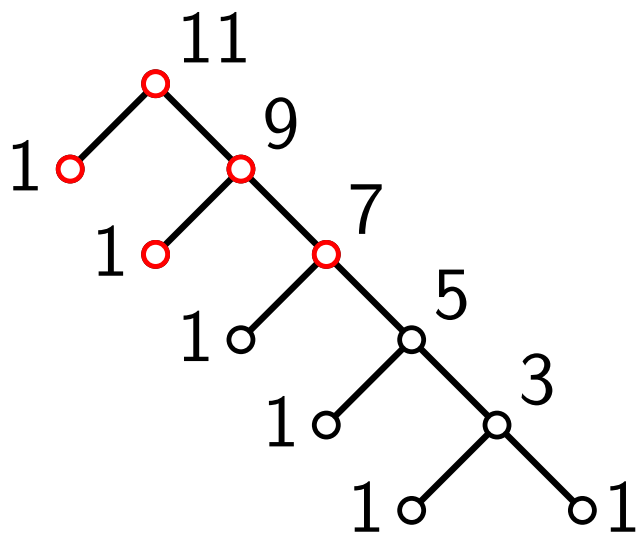
Radial Tree Drawings: Planarity

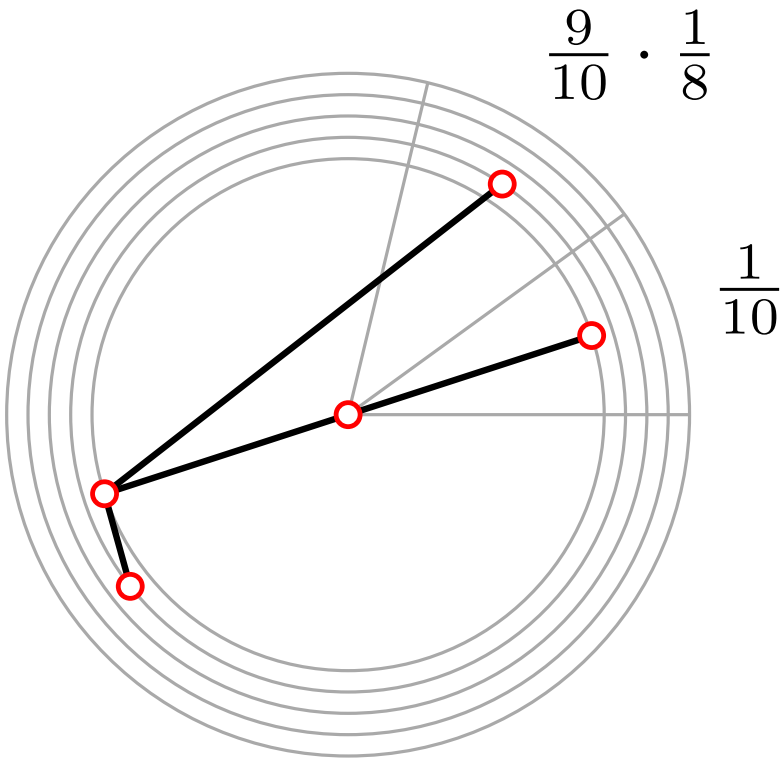
Distribute subtrees based on their node count (or leaf count)

Example: wedge angle τ_u for the subtree $T(u)$:

$$\tau_u = \frac{|T(u)|}{|T(p(u))|-1} \cdot \tau_{p(u)}; \text{ for root } r\text{: } \tau_r = 2\pi$$





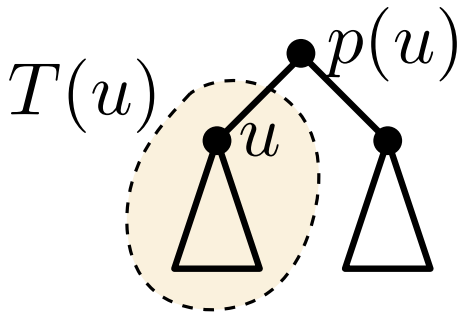


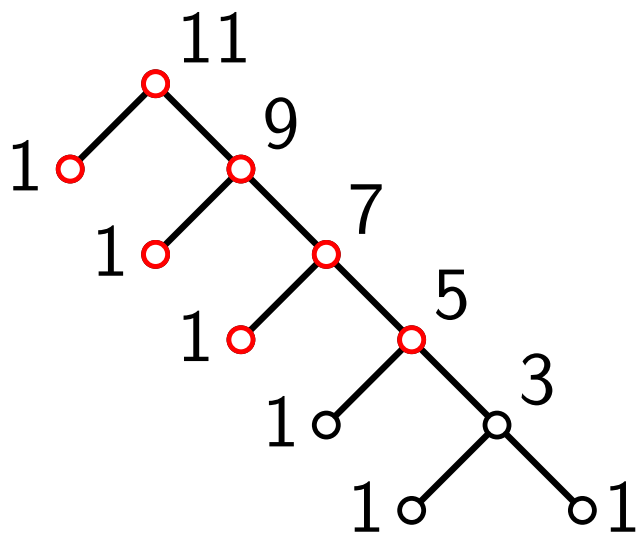
Radial Tree Drawings: Planarity

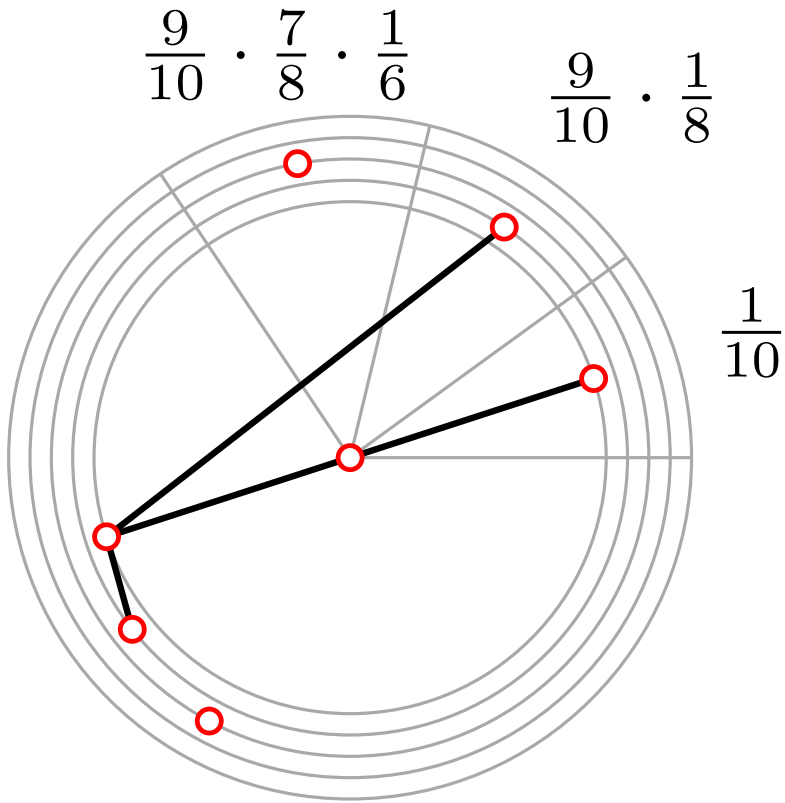
Distribute subtrees based on their node count (or leaf count)

Example: wedge angle τ_u for the subtree $T(u)$:

$$\tau_u = \frac{|T(u)|}{|T(p(u))|-1} \cdot \tau_{p(u)}; \text{ for root } r\text{: } \tau_r = 2\pi$$





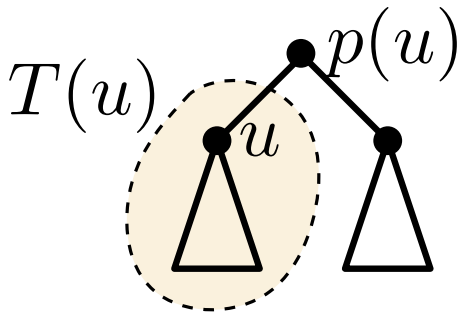


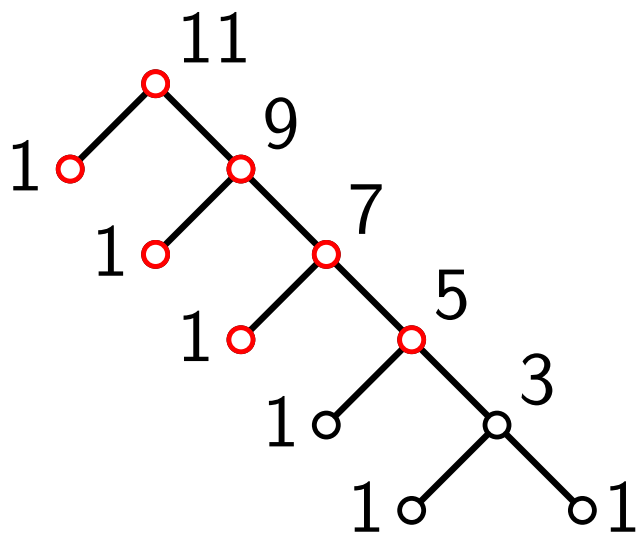
Radial Tree Drawings: Planarity

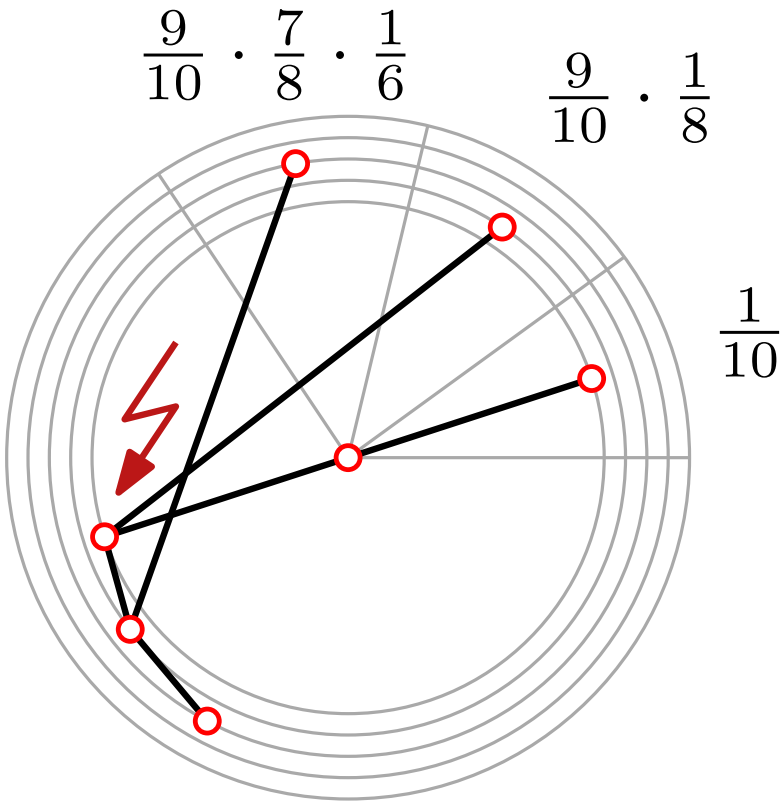
Distribute subtrees based on their node count (or leaf count)

Example: wedge angle τ_u for the subtree $T(u)$:

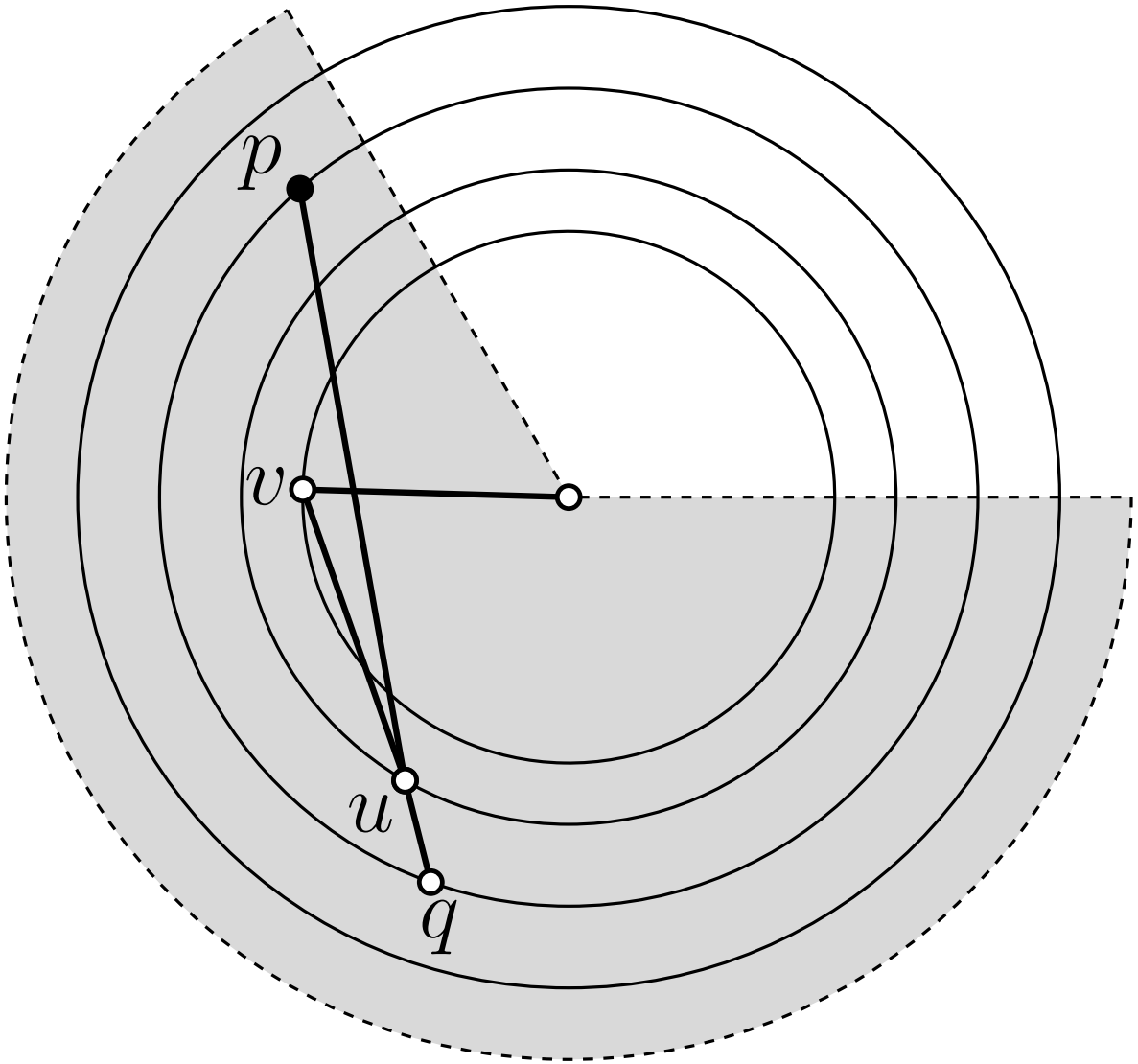
$$\tau_u = \frac{|T(u)|}{|T(p(u))|-1} \cdot \tau_{p(u)}; \text{ for root } r\text{: } \tau_r = 2\pi$$



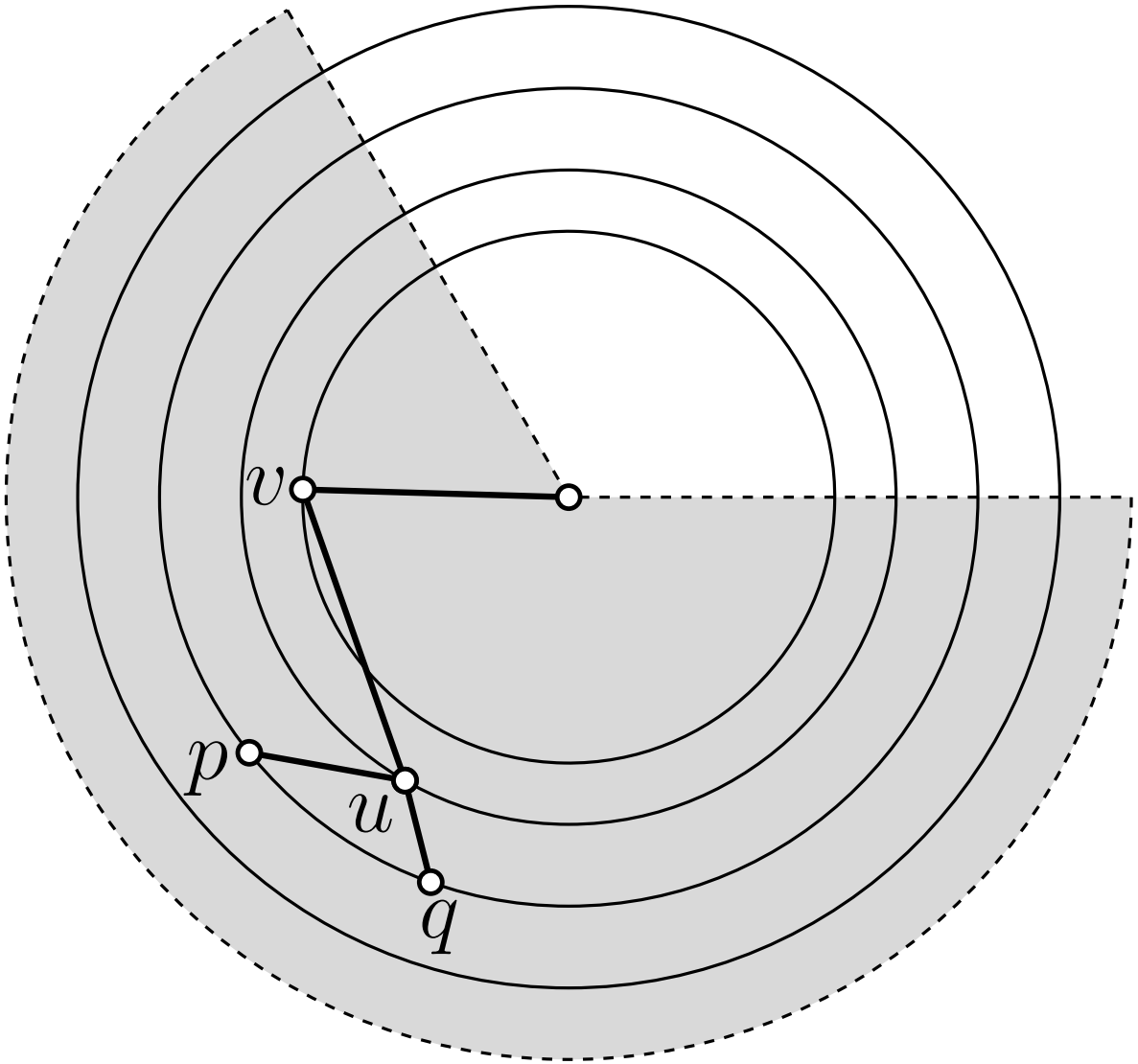




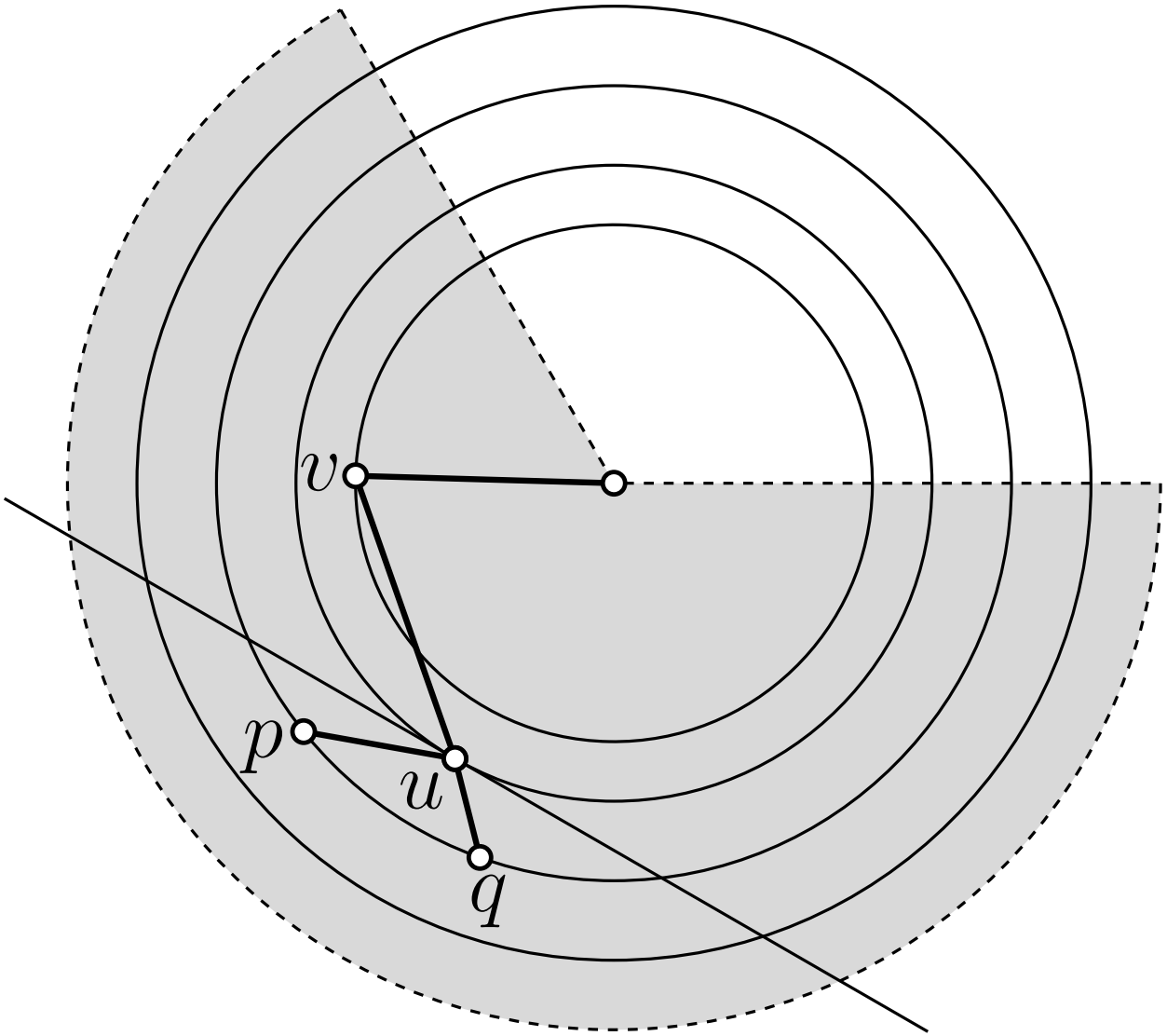
Radial Tree Drawings: Avoid Crossings



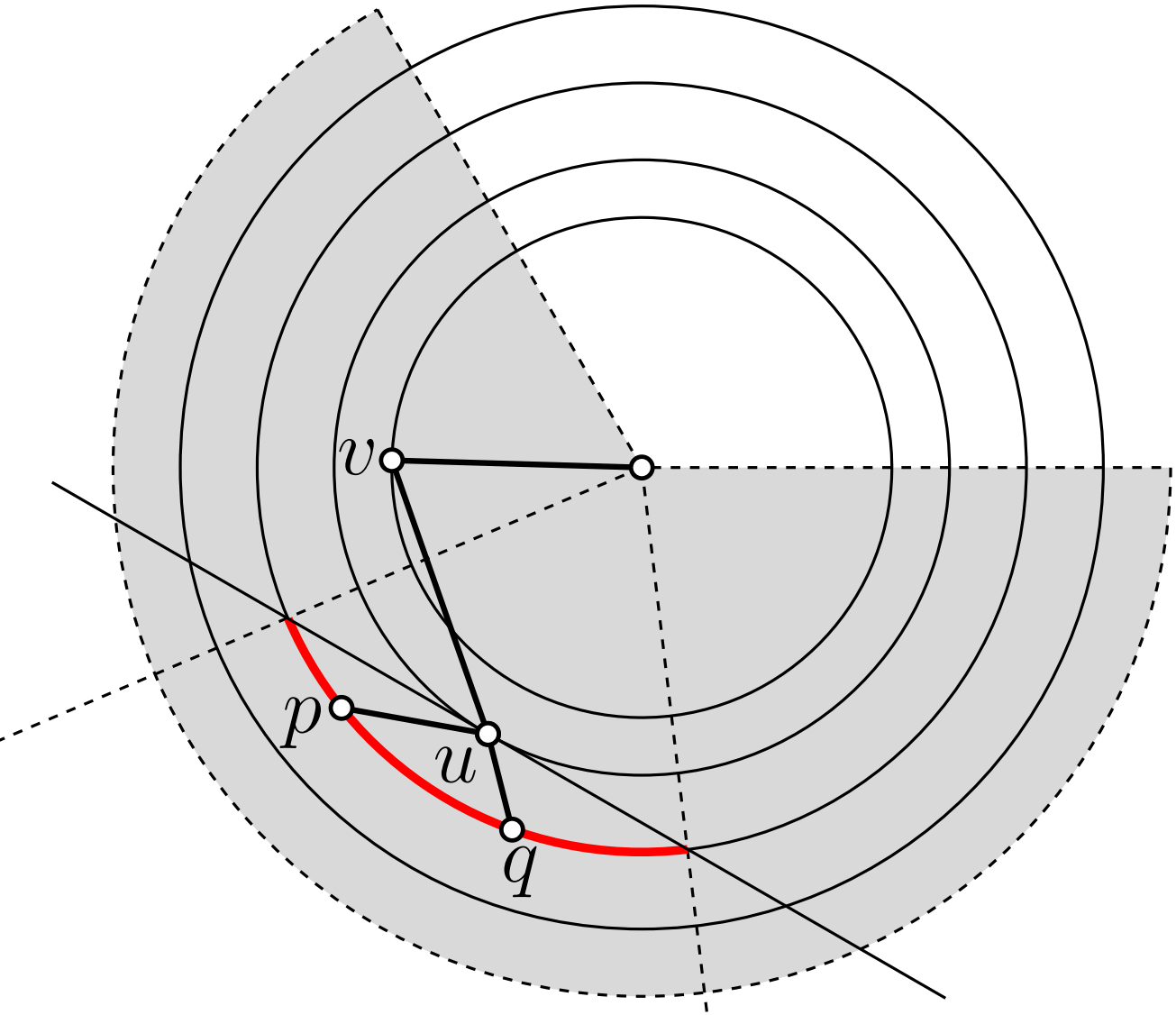
Radial Tree Drawings: Avoid Crossings



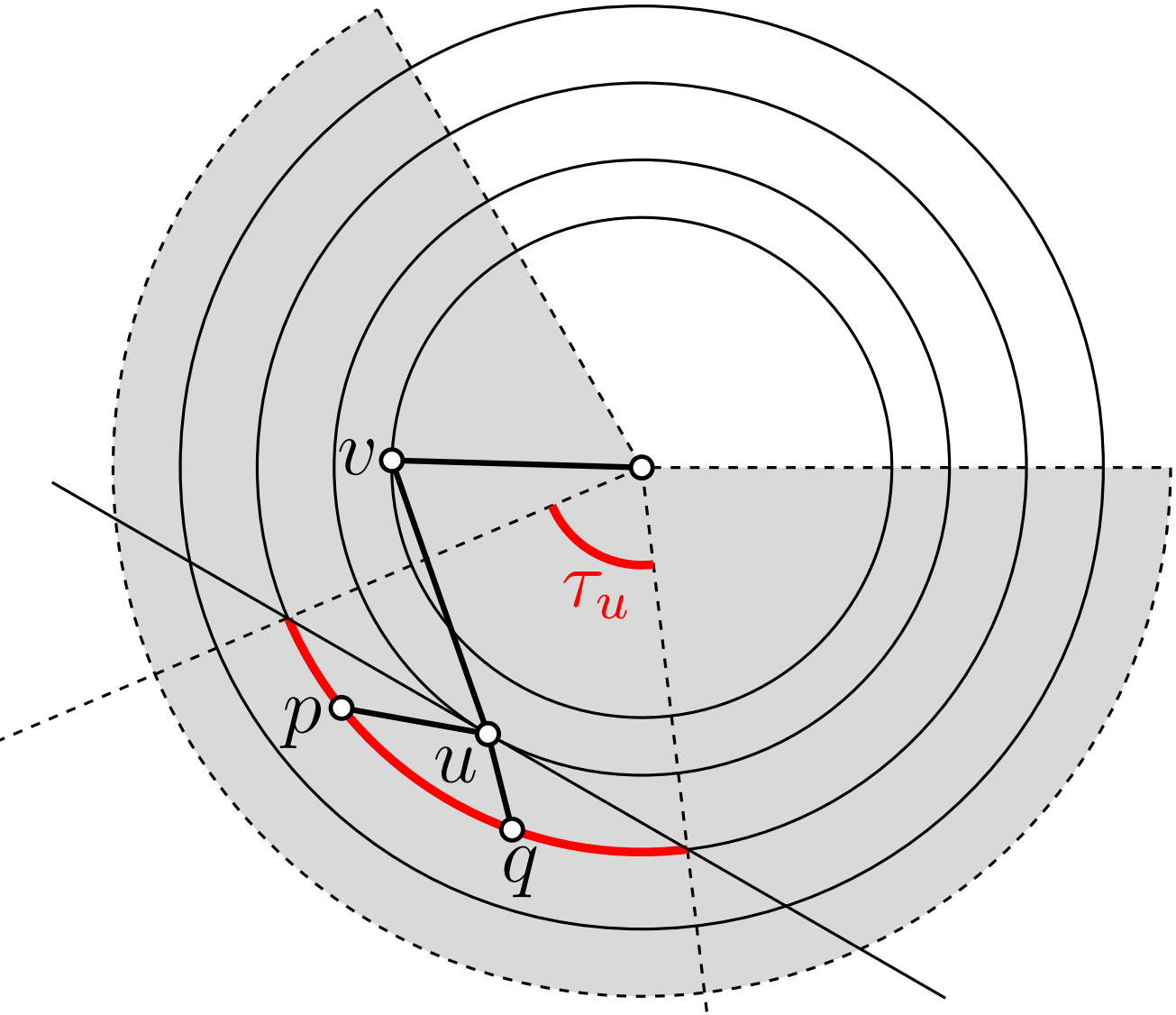
Radial Tree Drawings: Avoid Crossings



Radial Tree Drawings: Avoid Crossings

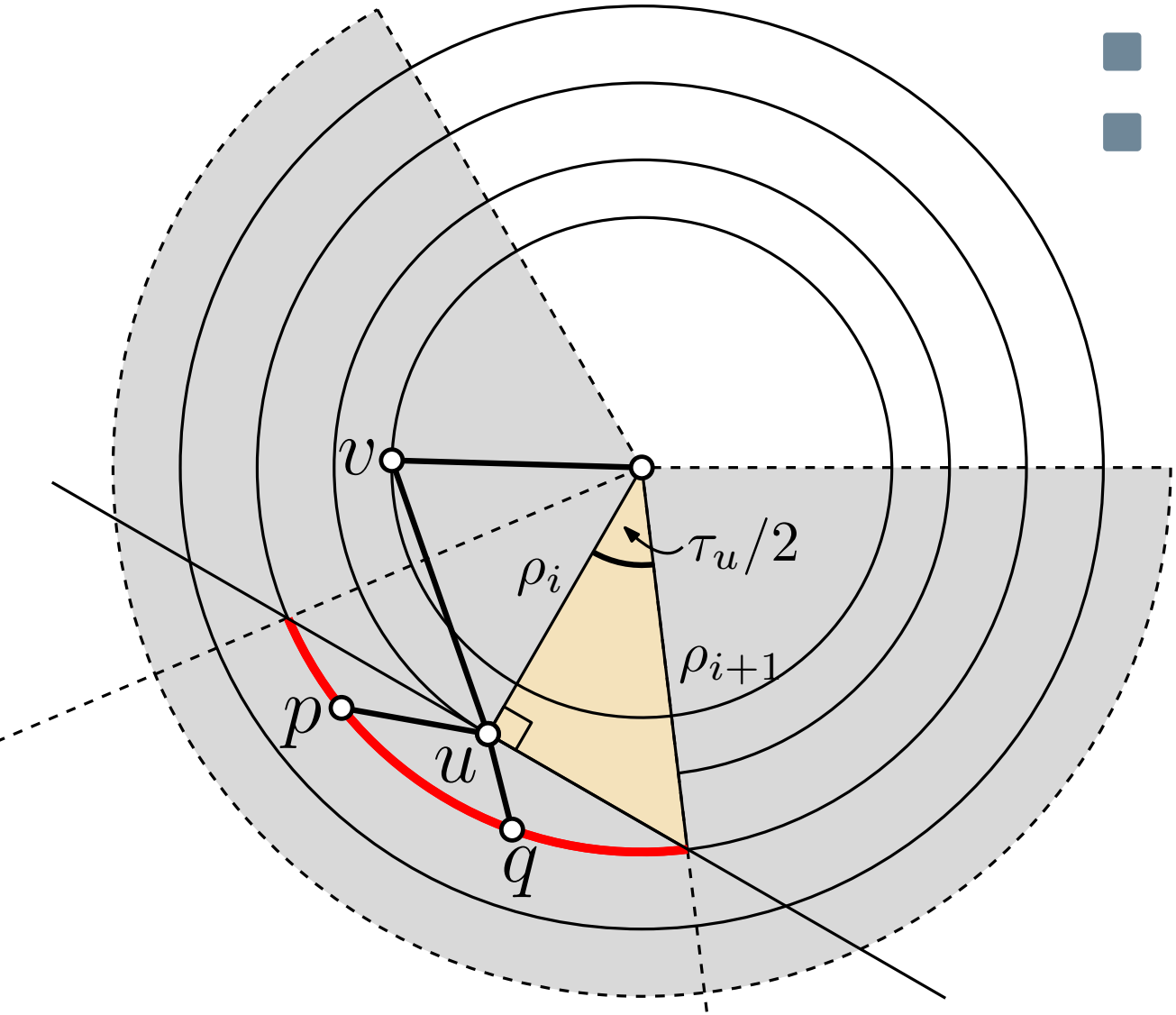


Radial Tree Drawings: Avoid Crossings



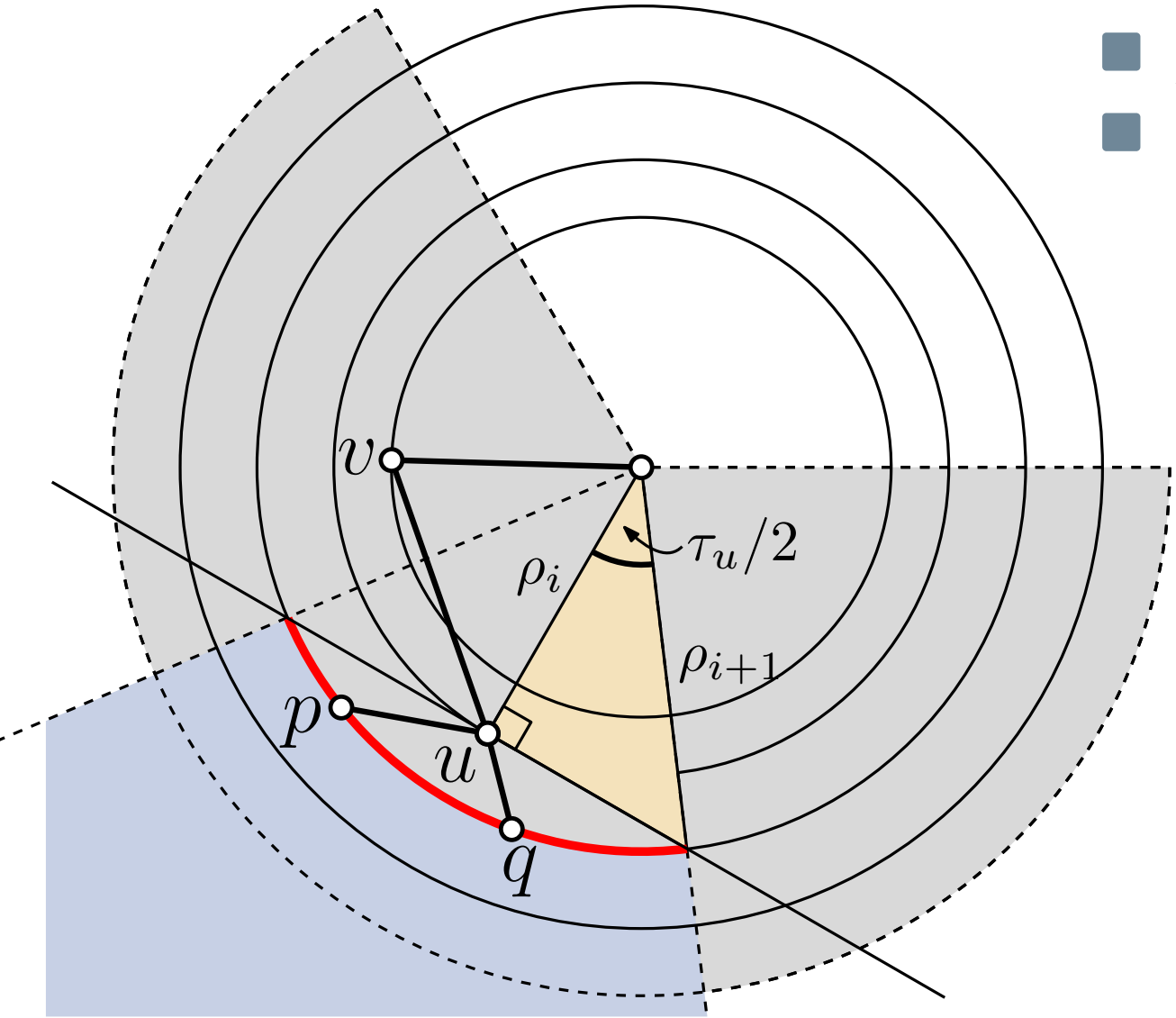
Radial Tree Drawings: Avoid Crossings

- τ_u: wedge angle for $T(u)$
- ρ_i: radius of layer i
- $\cos\frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$



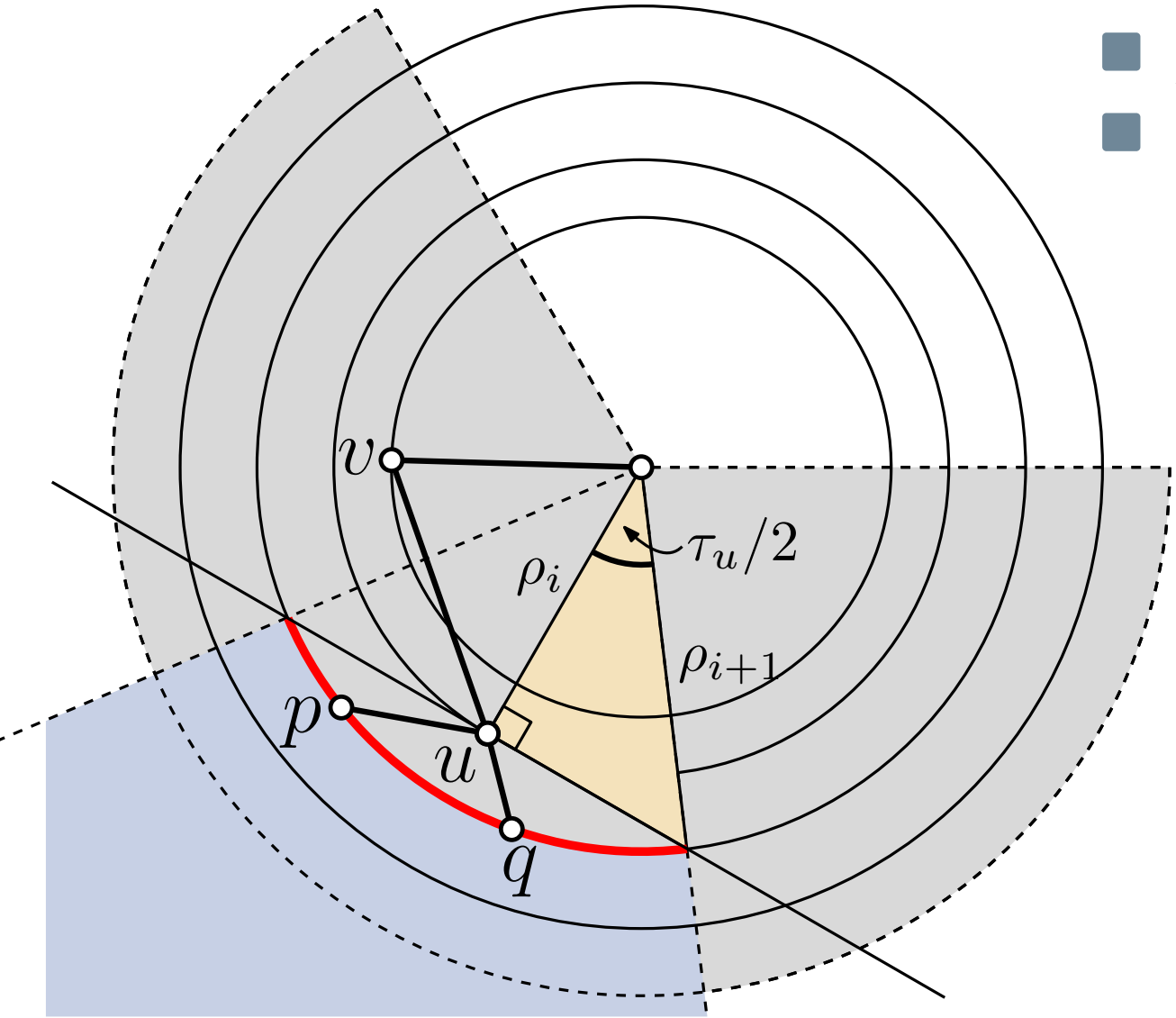
Radial Tree Drawings: Avoid Crossings

- τ_u: wedge angle for $T(u)$
- ρ_i: radius of layer i
- $\cos\frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$



Radial Tree Drawings: Avoid Crossings

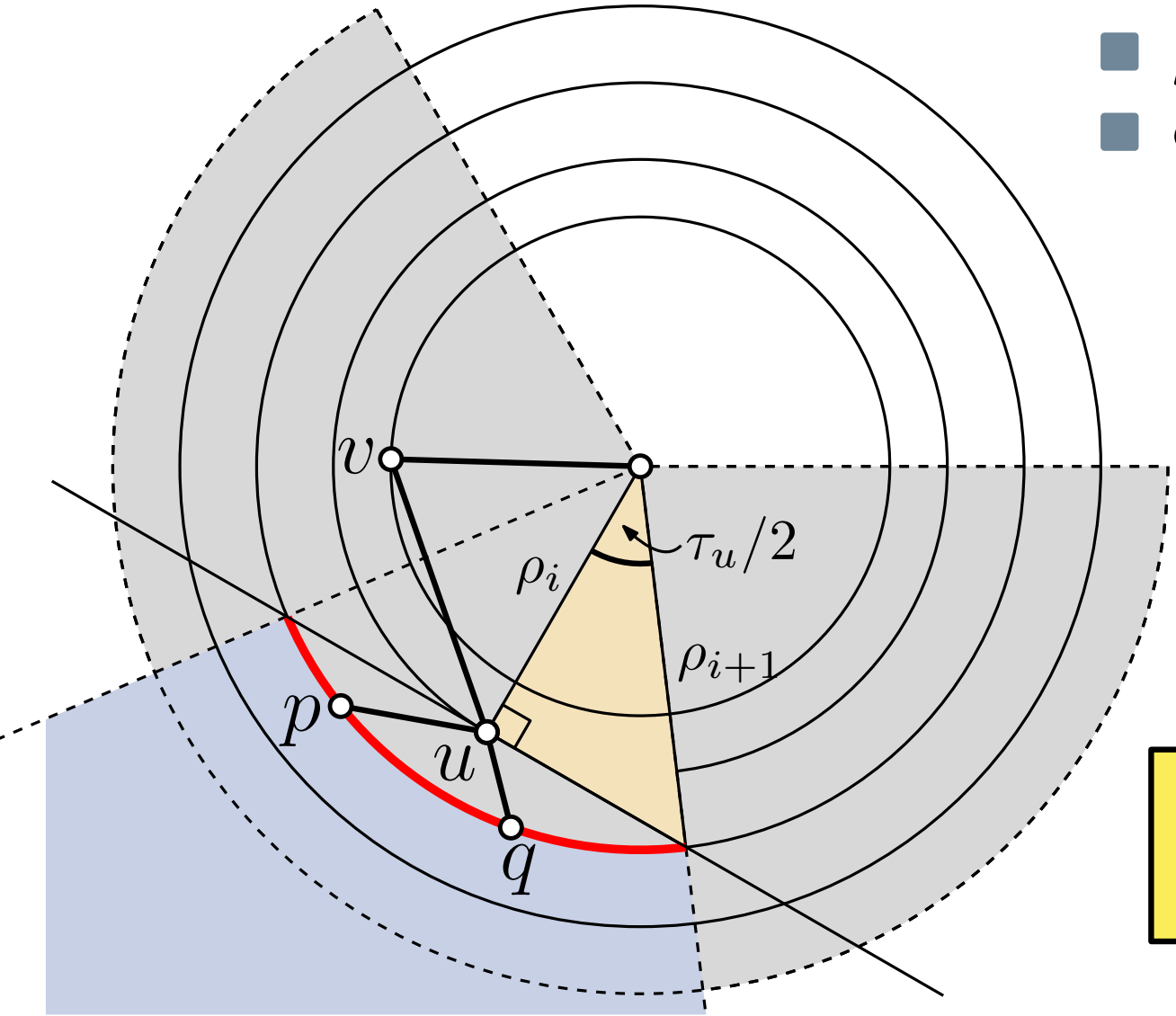
- τ_u: wedge angle for $T(u)$
- ρ_i: radius of layer i
- $\cos\frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$



- $\tau_u = \min\{\frac{|T(u)|}{|T(p(v))|-1} \cdot \tau_{p(u)}, 2\arccos\frac{\rho_i}{\rho_{i+1}}\}$ (correction)

Radial Tree Drawings: Avoid Crossings

- τ_u: wedge angle for $T(u)$
- ρ_i: radius of layer i
- $\cos\frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$



Why does this provide planarity?

- $\tau_u = \min\{\frac{|T(u)|}{|T(p(v))|-1} \cdot \tau_{p(u)}, 2\arccos\frac{\rho_i}{\rho_{i+1}}\}$ (correction)

Radial Tree Drawings: Algorithm

RadialTree(tree T, root $r \in T$, radii $\rho_1 < \cdots < \rho_{\mathsf{height}(T)}$)

begin
- $postorder(r)$
- $preorder(r, 0, 0, 2\pi)$
- **return** $(d_v, \alpha_v)_{v \in T}$
- `// polar coordinates`

postorder(vertex v)
- Compute all subtree sizes – recursively!

Radial Tree Drawings: Algorithm

RadialTree(tree T, root $r \in T$, radii $\rho_1 < \cdots < \rho_{\mathsf{height}(T)}$)

begin
- $postorder(r)$
- $preorder(r, 0, 0, 2\pi)$
- **return** $(d_v, \alpha_v)_{v \in T}$
- `// polar coordinates`

postorder(vertex v)
- $n_v \leftarrow 1$
- **foreach** child w of v **do**
  - $postorder(w)$
  - $n_v \leftarrow n_v + n_w$

Radial Tree Drawings: Algorithm

RadialTree(tree T, root $r \in T$, radii $\rho_1 < \cdots < \rho_{\mathsf{height}(T)}$)

begin
- $postorder(r)$
- $preorder(r, 0, 0, 2\pi)$
- **return** $(d_v, \alpha_v)_{v \in T}$
- `// polar coordinates`

postorder(vertex v)
- $n_v \leftarrow 1$
- **foreach** child w of v **do**
  - $postorder(w)$
  - $n_v \leftarrow n_v + n_w$

subtree size $|T(v)|$

Radial Tree Drawings: Algorithm

RadialTree(tree T, root $r \in T$, radii $\rho_1 < \cdots < \rho_{\mathsf{height}(T)}$)

begin
- $postorder(r)$
- $preorder(r, 0, 0, 2\pi)$
- **return** $(d_v, \alpha_v)_{v \in T}$
- `// polar coordinates`

postorder(vertex v)
- $n_v \leftarrow 1$
- **foreach** child w of v **do**
  - $postorder(w)$
  - $n_v \leftarrow n_v + n_w$

preorder(node v, $t, \alpha_{\min}, \alpha_{\max}$)
- $d_v \leftarrow \rho_t$
- $\alpha_v \leftarrow (\alpha_{\min} + \alpha_{\max})/2$
- **if** $t > 0$ **then**
  - $\alpha_{\min} \leftarrow \max\{\alpha_{\min}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}$
  - $\alpha_{\max} \leftarrow \min\{\alpha_{\max}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}$
- $left \leftarrow \alpha_{\min}$
- **foreach** child w of v **do**
  - $right \leftarrow left + \frac{n_w}{n_v - 1} \cdot (\alpha_{\max} - \alpha_{\min})$
  - $preorder(w, t+1, left, right)$
  - $left \leftarrow right$

Radial Tree Drawings: Algorithm

RadialTree(tree T, root $r \in T$, radii $\rho_1 < \cdots < \rho_{\mathsf{height}(T)}$)

begin
- $postorder(r)$
- $preorder(r, 0, 0, 2\pi)$
- **return** $(d_v, \alpha_v)_{v \in T}$
- `// polar coordinates`

postorder(vertex v)
- $n_v \leftarrow 1$
- **foreach** child w of v **do**
  - $postorder(w)$
  - $n_v \leftarrow n_v + n_w$

preorder(node v, $t, \alpha_{\min}, \alpha_{\max}$)
- $d_v \leftarrow \rho_t$
- $\alpha_v \leftarrow (\alpha_{\min} + \alpha_{\max})/2$ `// output`
- **if** $t > 0$ **then**
  - $\alpha_{\min} \leftarrow \max\{\alpha_{\min}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}$
  - $\alpha_{\max} \leftarrow \min\{\alpha_{\max}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}$
- $left \leftarrow \alpha_{\min}$
- **foreach** child w of v **do**
  - $right \leftarrow left + \frac{n_w}{n_v - 1} \cdot (\alpha_{\max} - \alpha_{\min})$
  - $preorder(w, t+1, left, right)$
  - $left \leftarrow right$

Radial Tree Drawings: Algorithm

RadialTree(tree T, root $r \in T$, radii $\rho_1 < \cdots < \rho_{\mathsf{height}(T)}$)

begin
- $postorder(r)$
- $preorder(r, 0, 0, 2\pi)$
- **return** $(d_v, \alpha_v)_{v \in T}$
- `// polar coordinates`

postorder(vertex v)
- $n_v \leftarrow 1$
- **foreach** child w of v **do**
  - $postorder(w)$
  - $n_v \leftarrow n_v + n_w$

preorder(node v, $t, \alpha_{\min}, \alpha_{\max}$)
- $d_v \leftarrow \rho_t$
- $\alpha_v \leftarrow (\alpha_{\min} + \alpha_{\max})/2$ `// output`
- **if** $t > 0$ **then**
  - $\alpha_{\min} \leftarrow \max\{\alpha_{\min}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}$
  - $\alpha_{\max} \leftarrow \min\{\alpha_{\max}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}$
- $left \leftarrow \alpha_{\min}$
- **foreach** child w of v **do**
  - $right \leftarrow left + \frac{n_w}{n_v - 1} \cdot (\alpha_{\max} - \alpha_{\min})$
  - $preorder(w, t+1, left, right)$
  - $left \leftarrow right$

Theorem: Algorithm RadialTree computes a planar radial drawing in $O(n)$ time.

Overview

Layered drawings of rooted trees

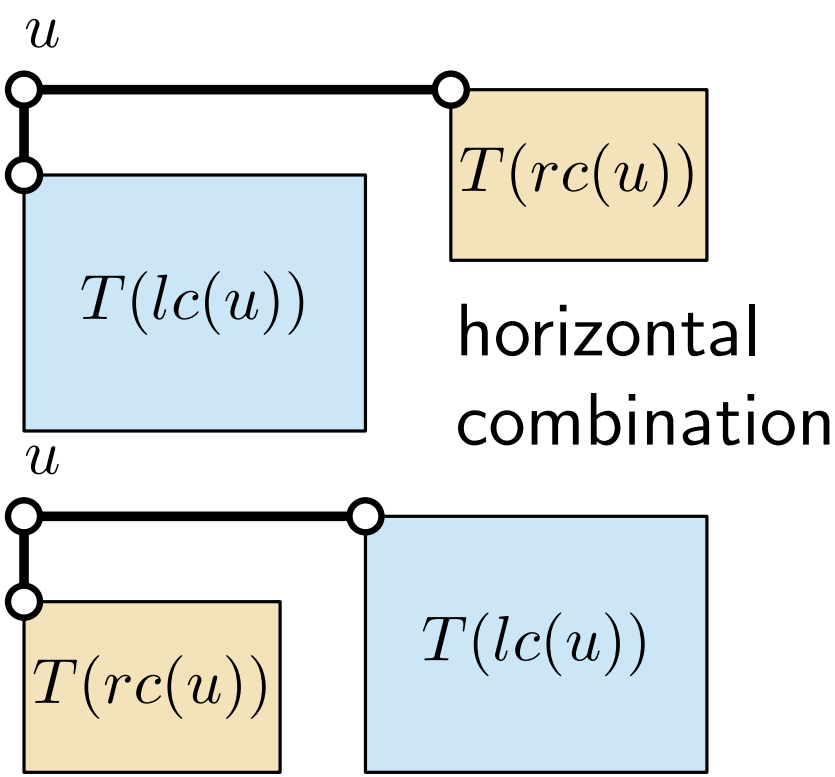
Radial tree drawings

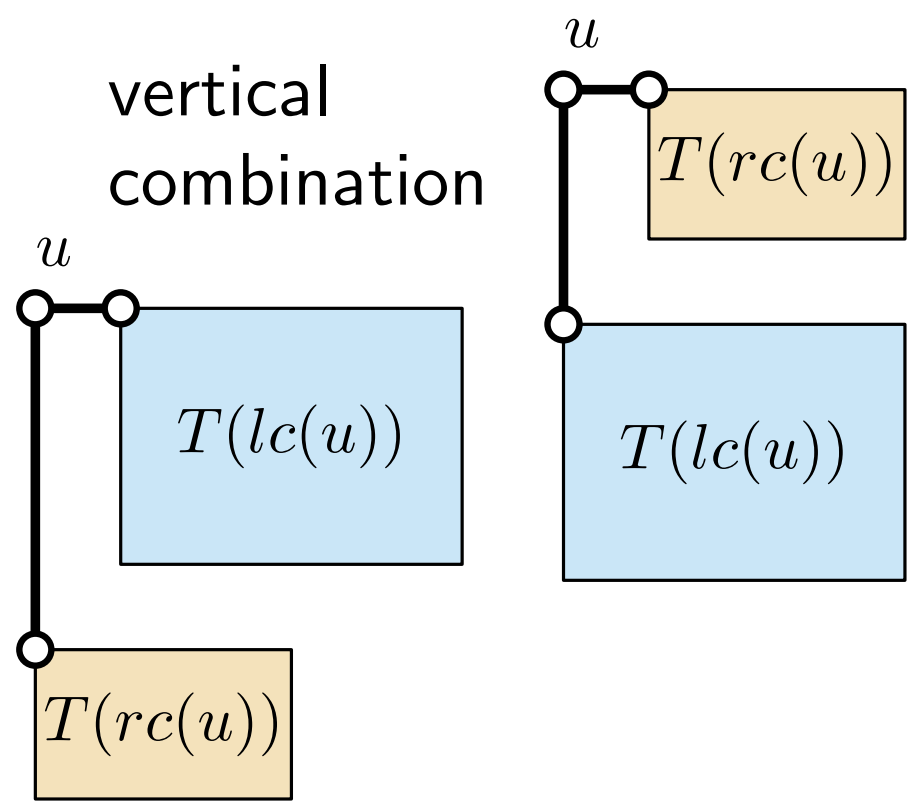
Compact orthogonal drawings of rooted trees

Compact Orthogonal Tree Drawings: HV Layout

Def: HV layout is a planar, straight-line grid drawing of a rooted binary tree with

- children either vertically aligned and below the parent or horizontally aligned and to the right
- recursively defined by placing bounding boxes of subtrees without intersections





Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)

Algorithm RightHeavyHVTreeDraw

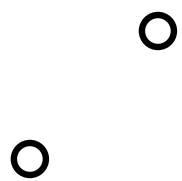
- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree

Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree

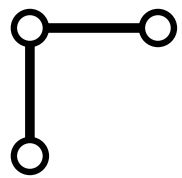
Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



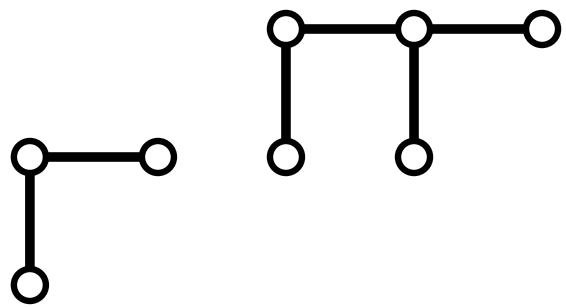
Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



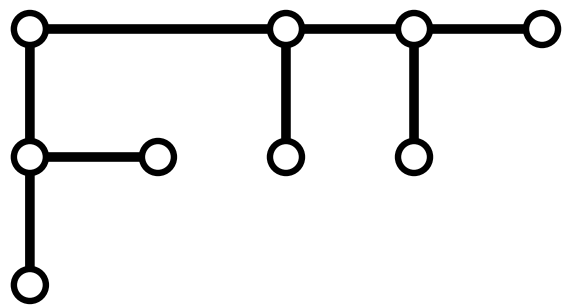
Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



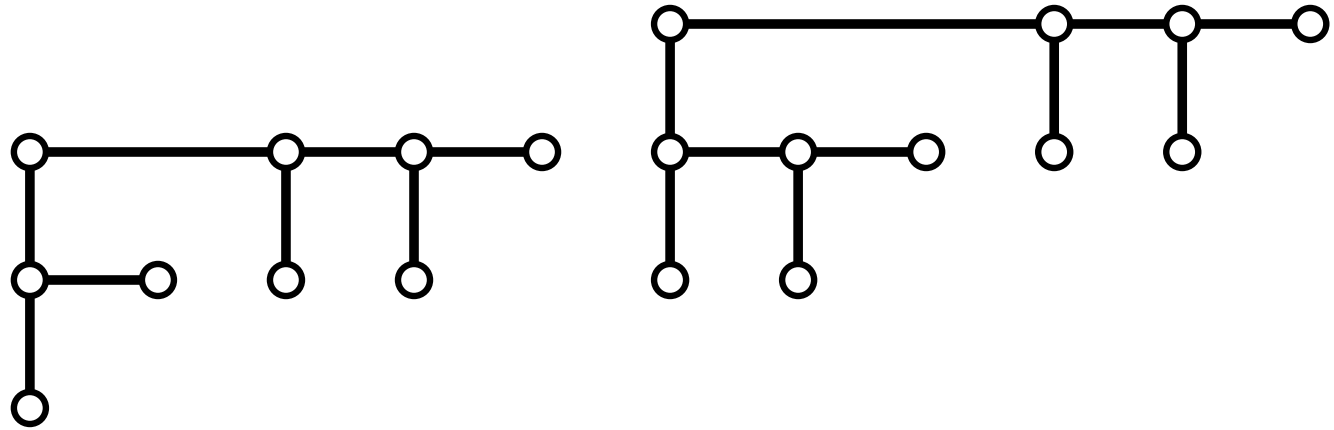
Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



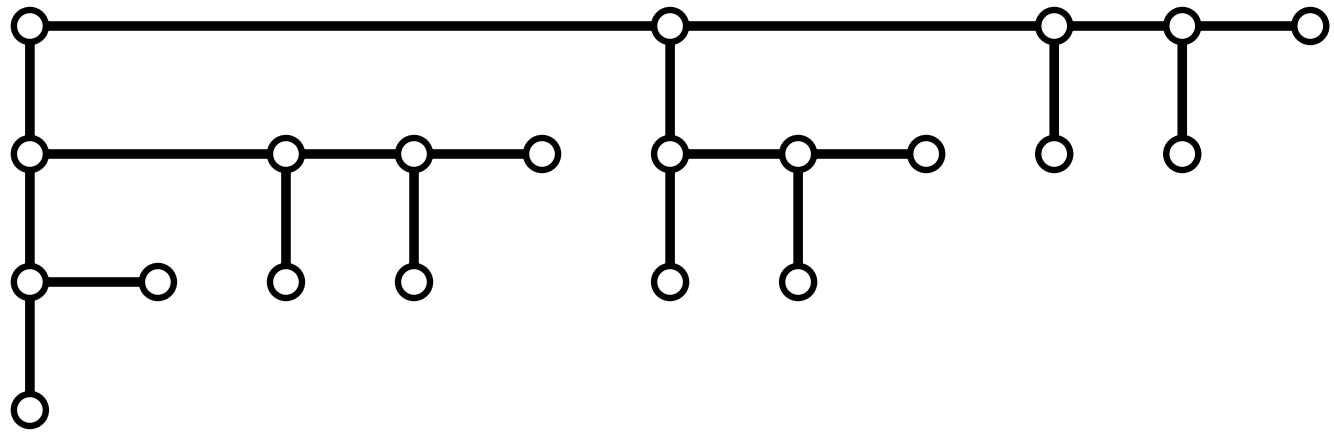
Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



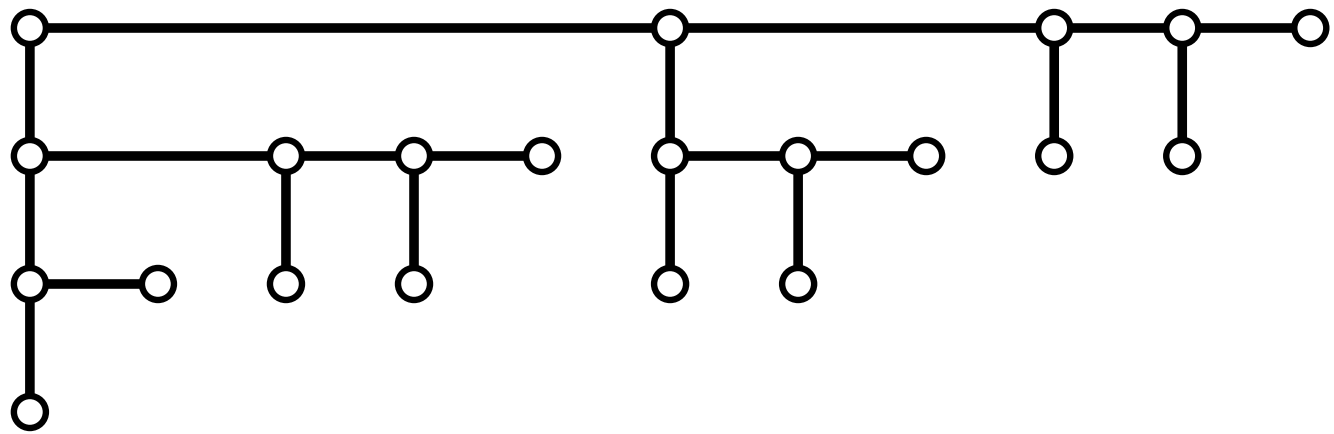
Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



Algorithm RightHeavyHVTreeDraw

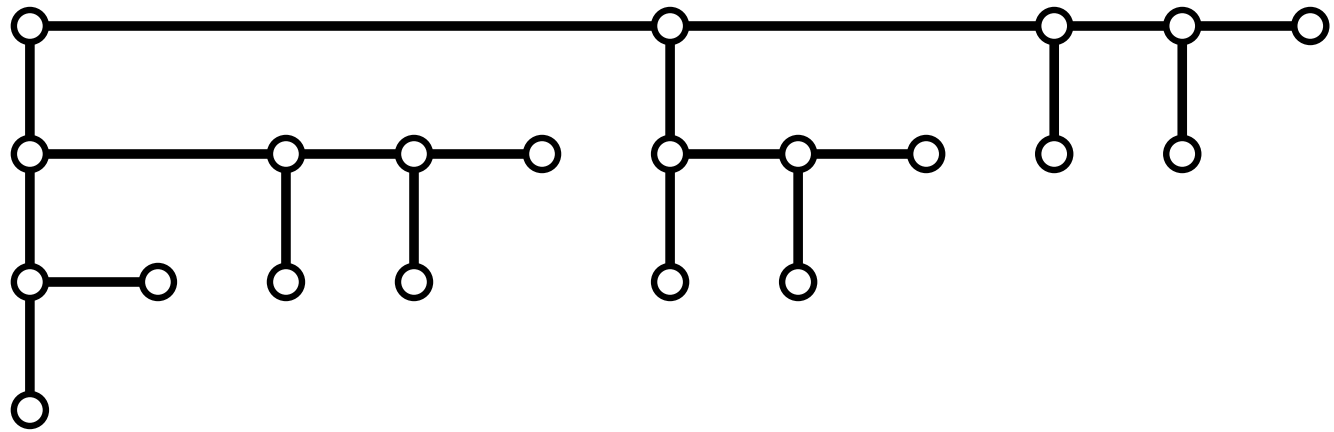
- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



Theorem: Drawing has width $\leq$

Algorithm RightHeavyHVTreeDraw

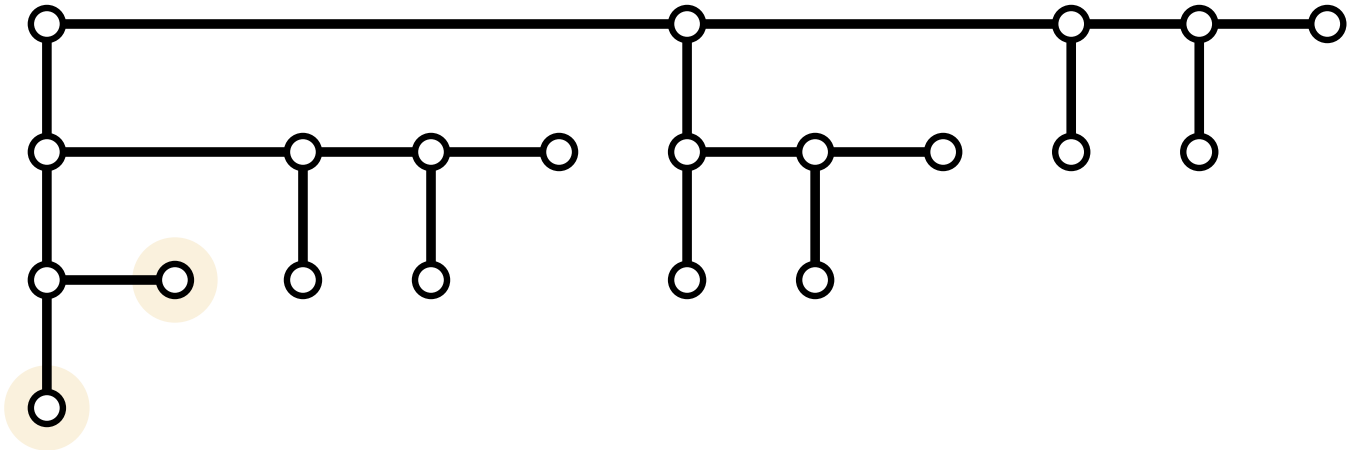
- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



Theorem: Drawing has width $\leq n$ and height $\leq$

Algorithm RightHeavyHVTreeDraw

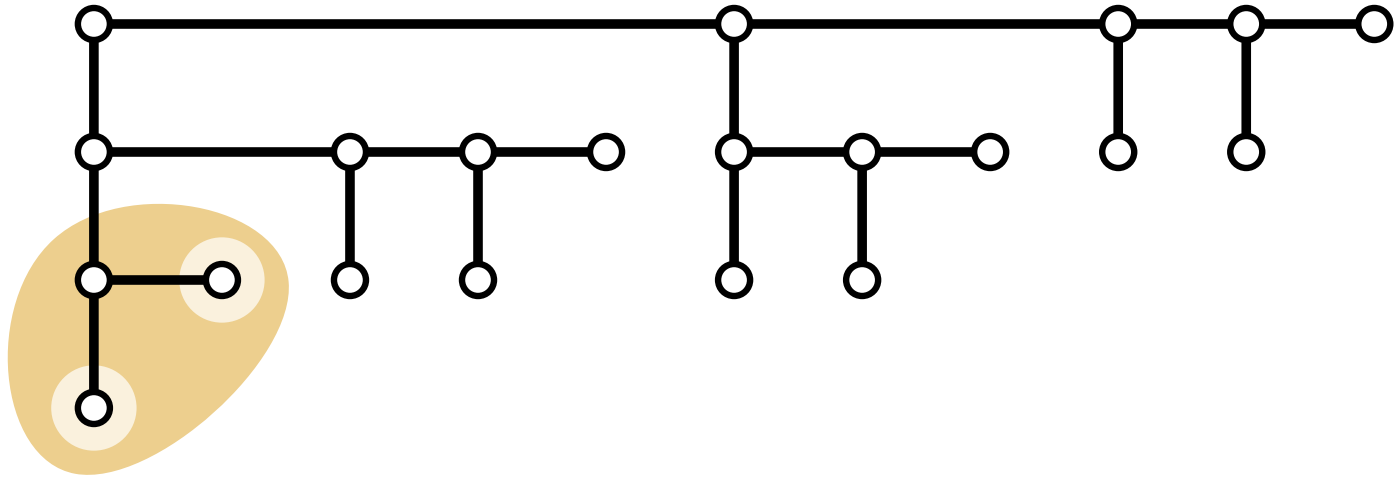
- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



Theorem: Drawing has width $\leq n$ and height $\leq$

Algorithm RightHeavyHVTreeDraw

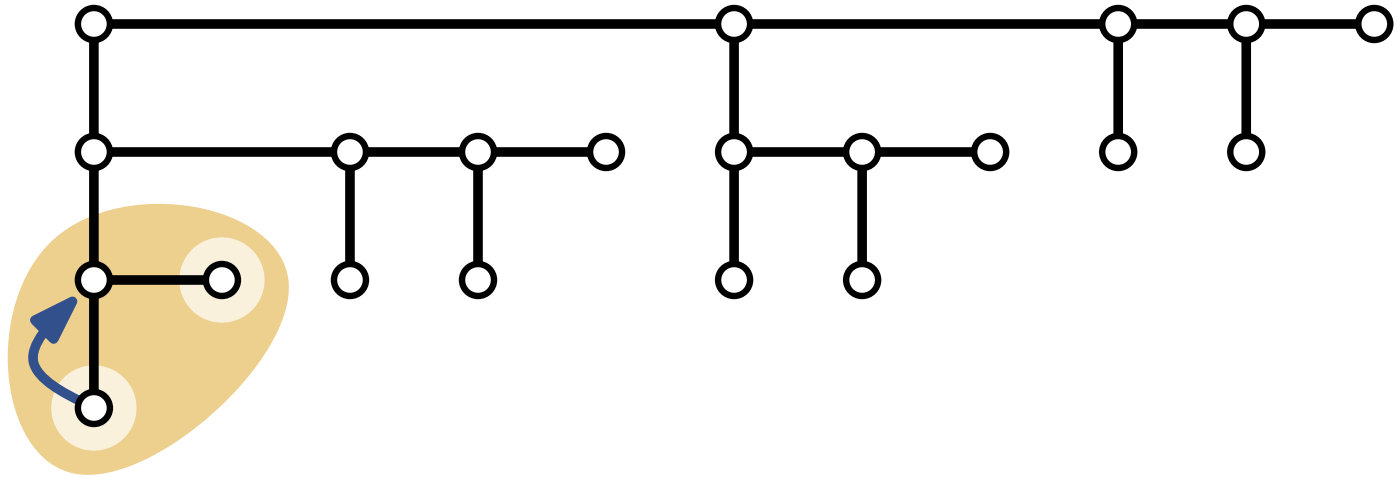
- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



Theorem: Drawing has width $\leq n$ and height $\leq$

Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree

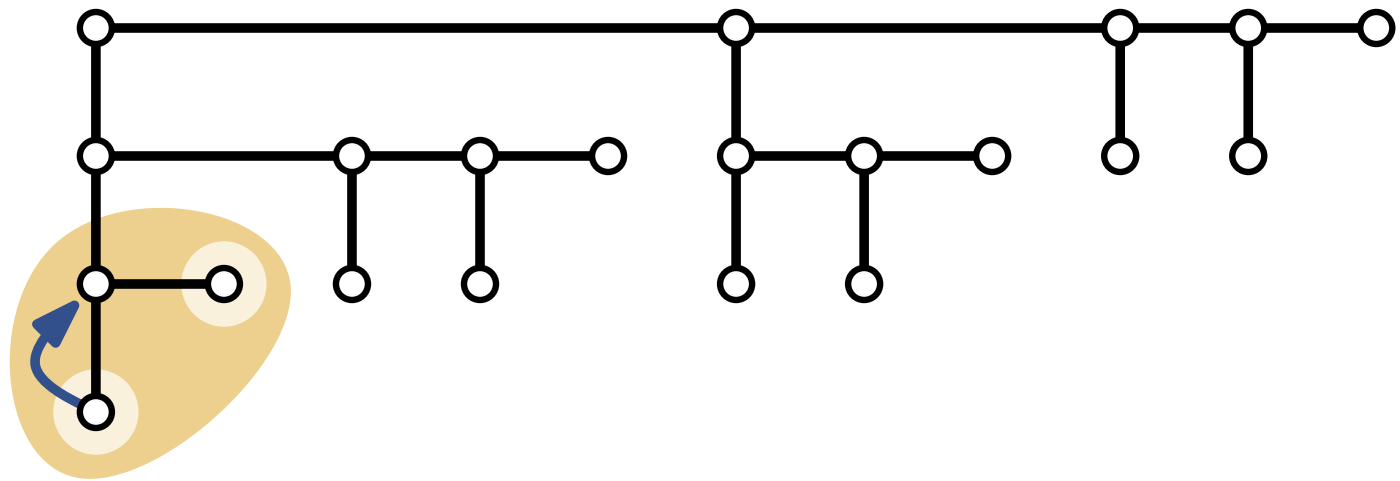


Theorem: Drawing has width $\leq n$ and height $\leq$

Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree

size at least doubles

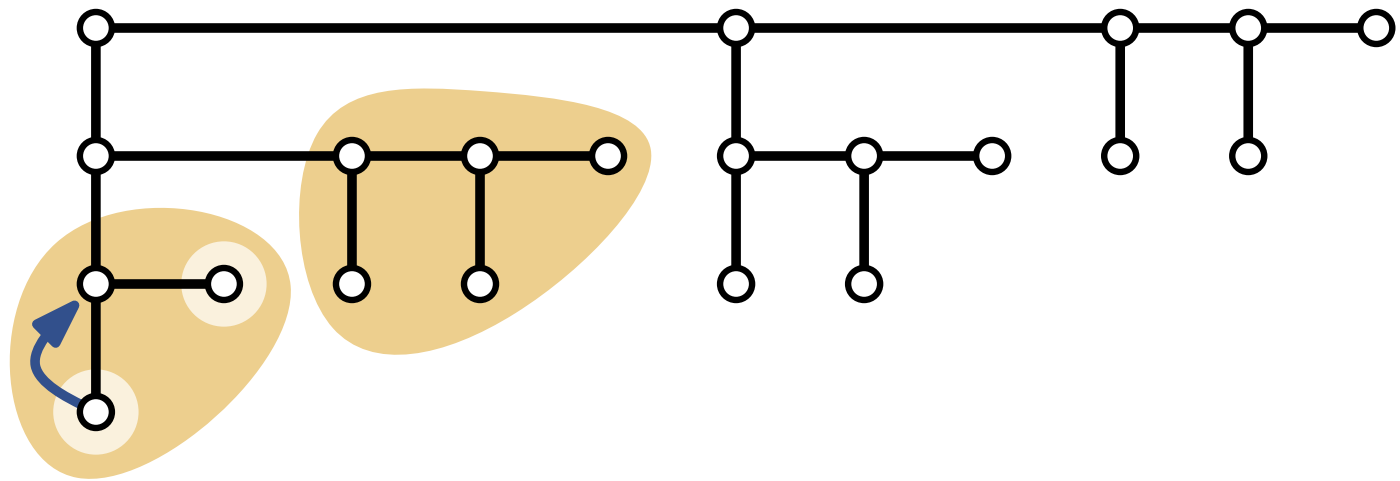


Theorem: Drawing has width $\leq n$ and height $\leq$

Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree

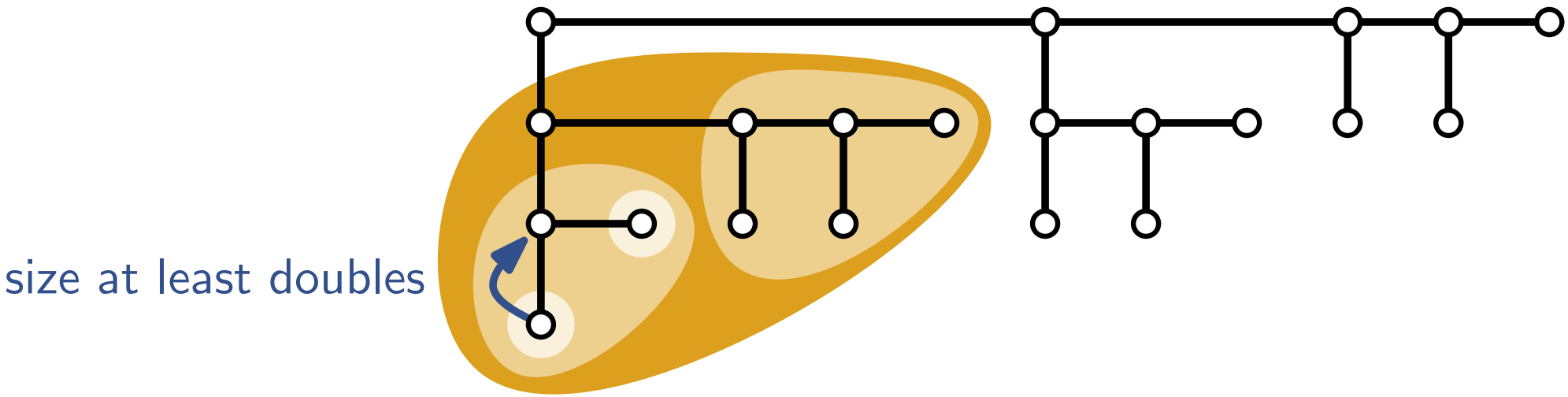
size at least doubles



Theorem: Drawing has width $\leq n$ and height $\leq$

Algorithm RightHeavyHVTreeDraw

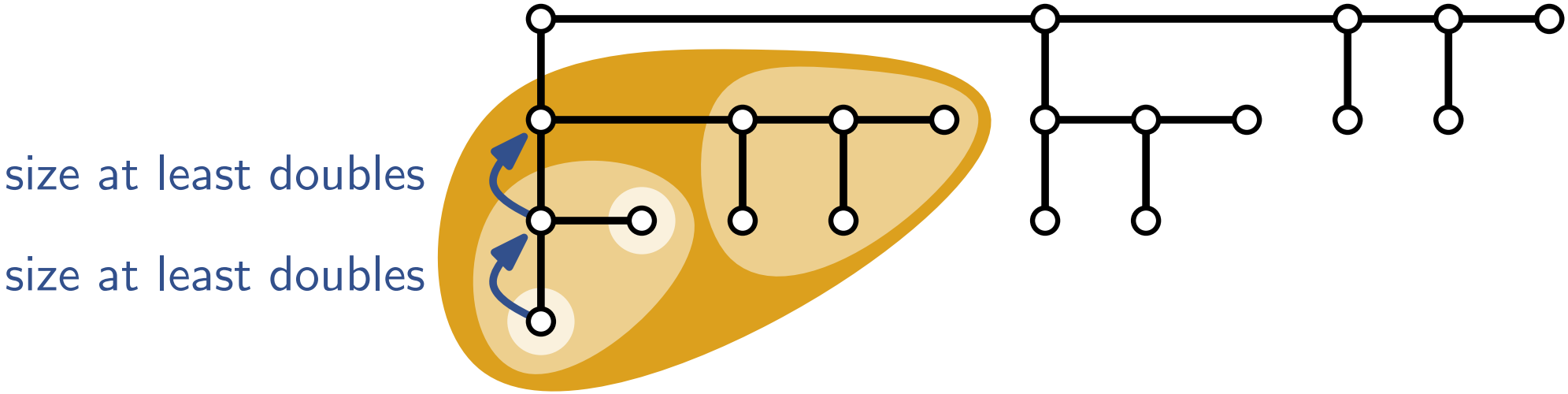
- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



Theorem: Drawing has width $\leq n$ and height $\leq$

Algorithm RightHeavyHVTreeDraw

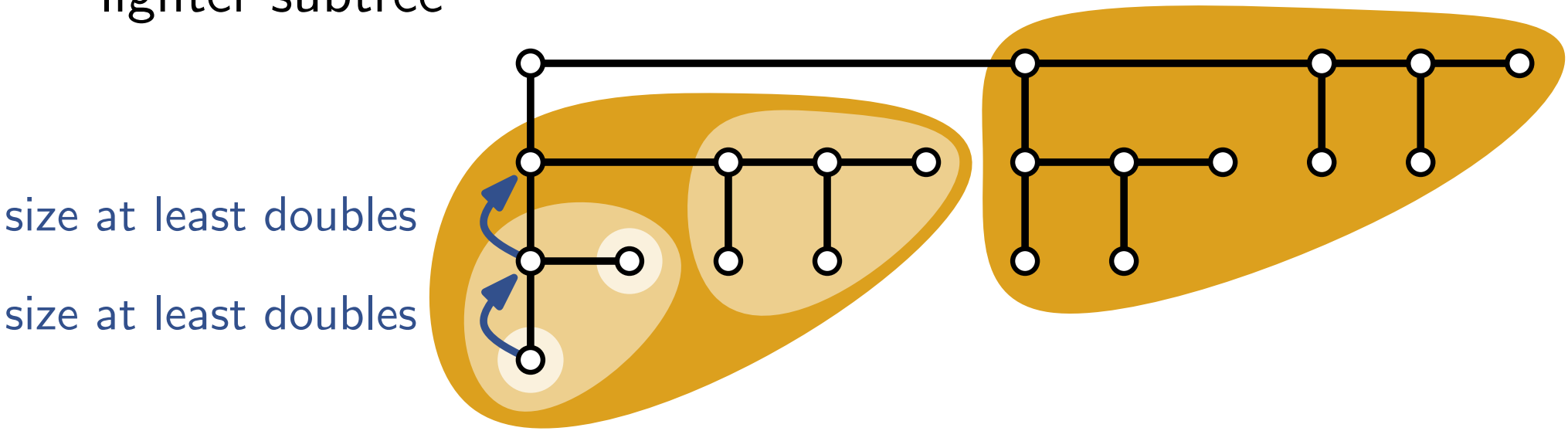
- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



Theorem: Drawing has width $\leq n$ and height $\leq$

Algorithm RightHeavyHVTreeDraw

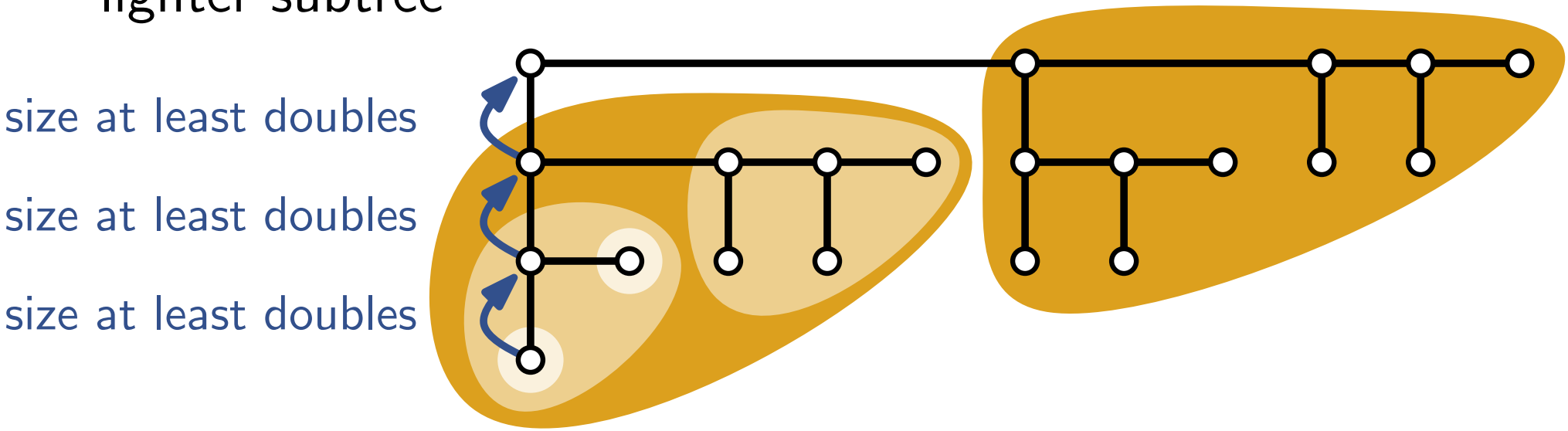
- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



Theorem: Drawing has width $\leq n$ and height $\leq$

Algorithm RightHeavyHVTreeDraw

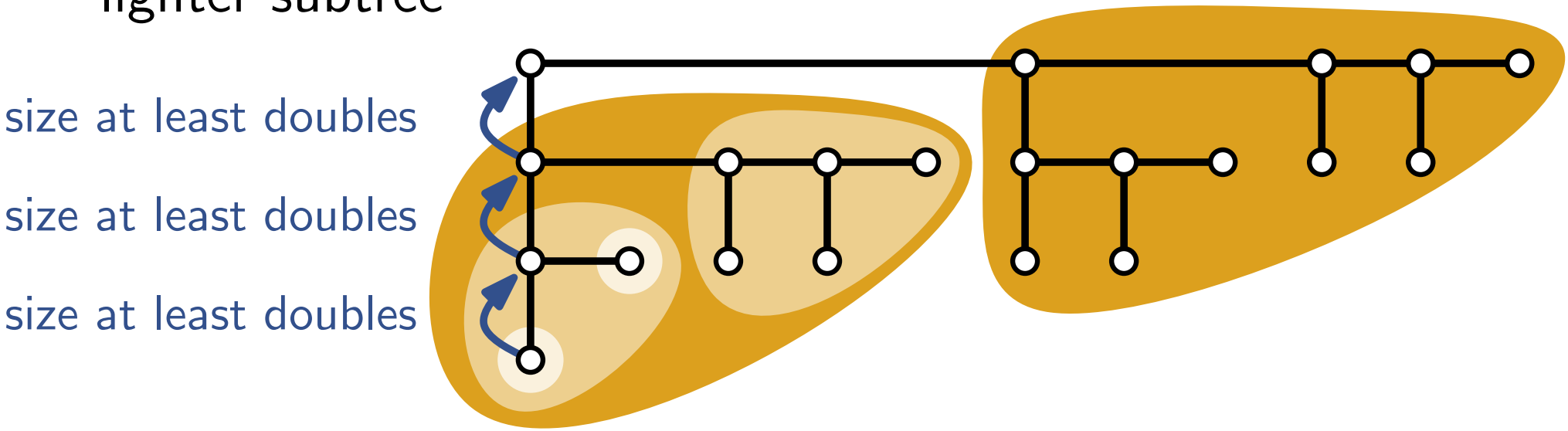
- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



Theorem: Drawing has width $\leq n$ and height $\leq$

Algorithm RightHeavyHVTreeDraw

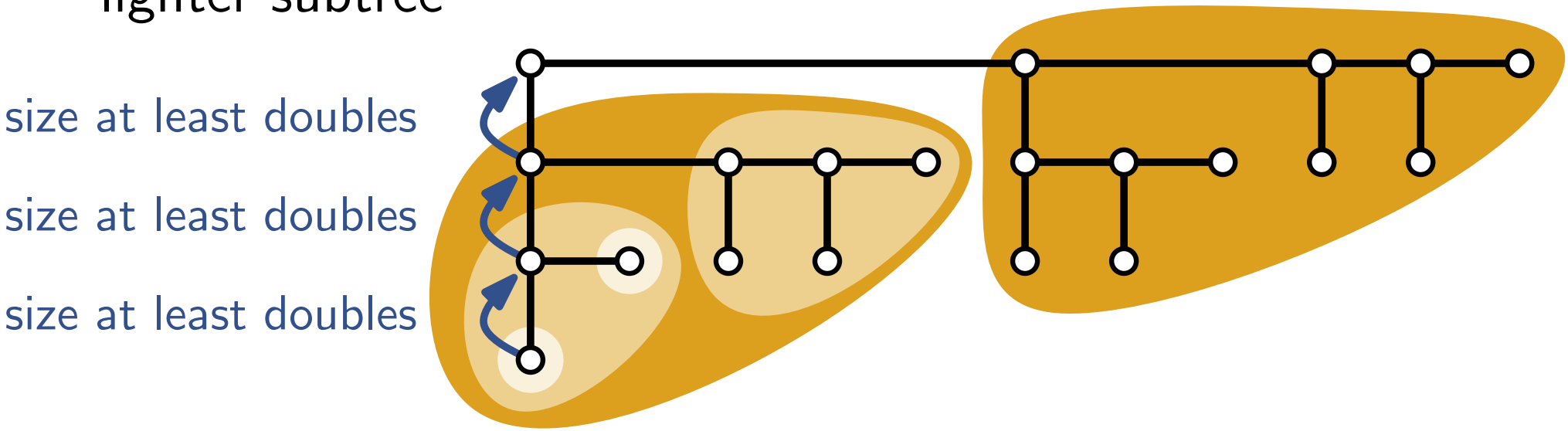
- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



Theorem: Drawing has width $\leq n$ and height $\leq \log_2 n$.

Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree

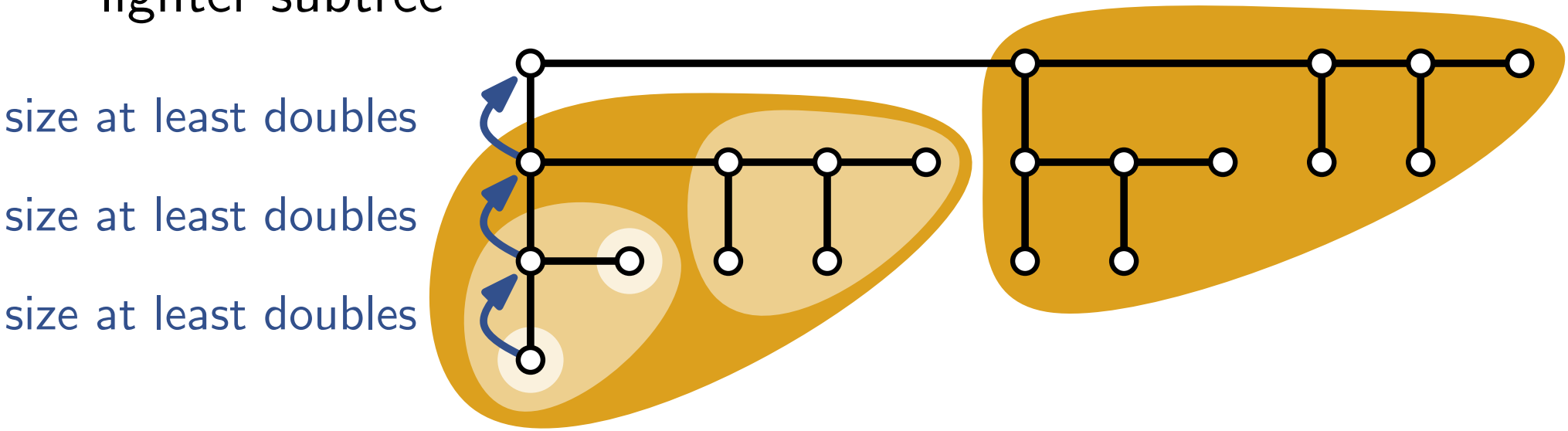


Theorem: Drawing has width $\leq n$ and height $\leq \log_2 n$.

In other words: A rooted binary tree can be drawn in $O(n)$ time on a grid of size $O(n \log n)$.

Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



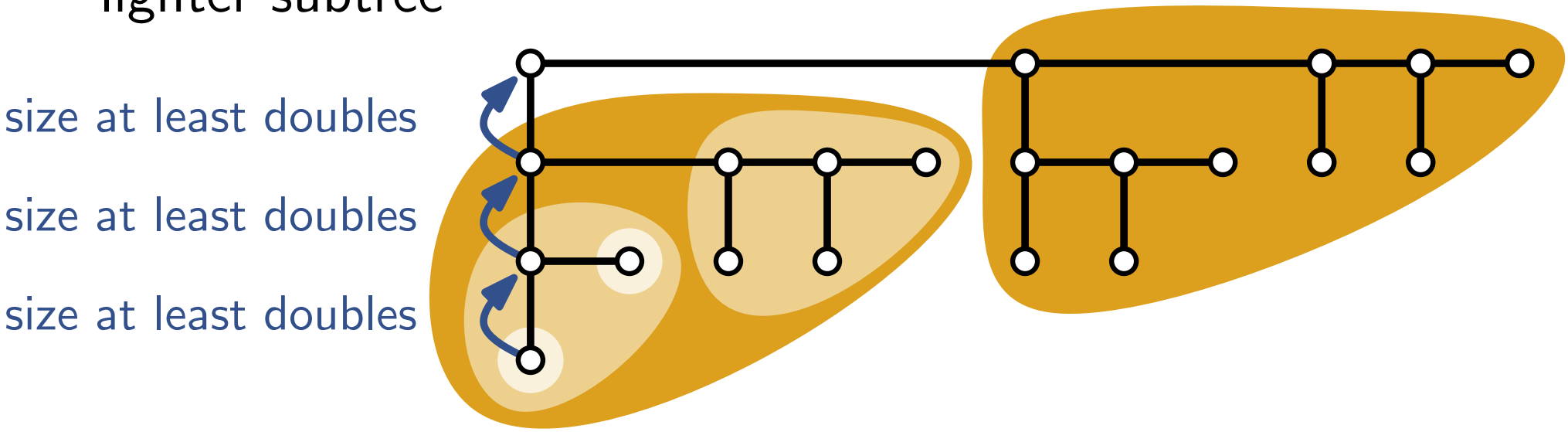
Theorem: Drawing has width $\leq n$ and height $\leq \log_2 n$.

In other words: A rooted binary tree can be drawn in $O(n)$ time on a grid of size $O(n \log n)$.

$\rightarrow$ using dynamic programming exact area minimization takes $O(n^2)$ time

Algorithm RightHeavyHVTreeDraw

- Construct drawings of left and right subtrees recursively (postorder)
- Place heavier subtree as horizontal combination right of lighter subtree



Theorem: Drawing has width $\leq n$ and height $\leq \log_2 n$.

In other words: A rooted binary tree can be drawn in $O(n)$ time on a grid of size $O(n \log n)$.

$\rightarrow$ using dynamic programming exact area minimization takes $O(n^2)$ time

But: we might have changed child order

Recall: Layout Problem

Graph visualization problem

given: graph $G = (V, E)$
find: drawing Γ of G that

- complies with the given drawing conventions
- optimizes the given aesthetics
- satisfies the partial/local constraints

$\rightarrow$ often lead to NP-hard optimization problems!
$\rightarrow$ often several competing criteria

Recall: Layout Problem

Graph visualization problem

given: graph $G = (V, E)$
find: drawing Γ of G that

- complies with the given drawing conventions
- optimizes the given aesthetics
- satisfies the partial/local constraints

$\rightarrow$ often lead to NP-hard optimization problems!
$\rightarrow$ often several competing criteria

today:

- (binary/ordered) trees
- planar, straight-line (grid) drawings
- optional: centered parent, layered, upward, ordered, isomorphic subtrees
- optimization: area
- NP-hard: width minimization
- competing: width vs. isomorphic subtrees, area vs. ordered

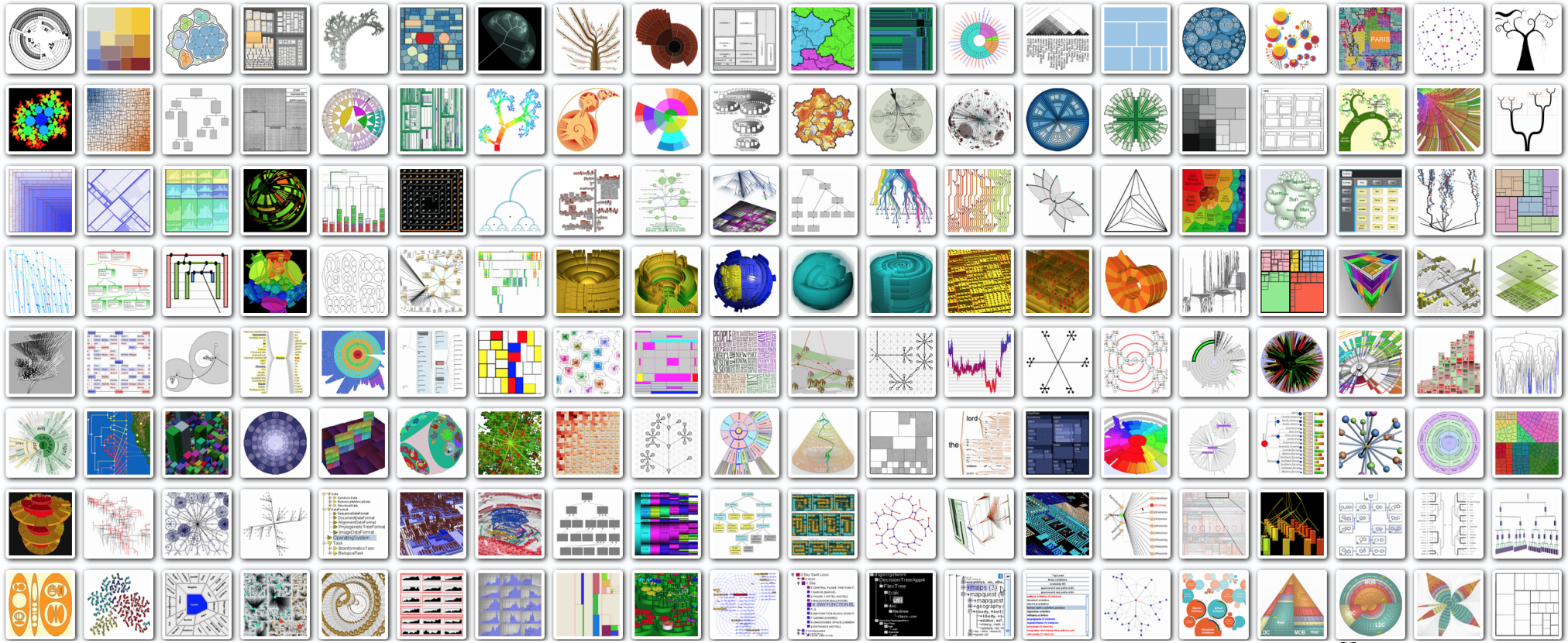
Summary and Outlook

- layered tree drawings
- radial tree drawings
- HV tree drawings

Summary and Outlook

- layered tree drawings
- radial tree drawings
- HV tree drawings

Trees are practically important graph class! Many layout algorithms exist and new ones are developed.



©treevis.net

FYI: Area Minimization for Trees

Area minimization and area bounds of tree drawings in many flavors is still an active (theoretical) research topic.

Some examples:

- unordered, upward, binary: $O(n \log \log n)$ (1996)
- ordered, upward, binary: $O(n \log n)$ (2003)
- ordered, non-upward, orthogonal, binary: $O(nc^{\sqrt{\log n}})$ (2017)
- unordered, non-upward, binary: $\Theta(n)$ (2003)
- unordered, upward, general: $O(n\sqrt{\log n}(\log \log n)^k)$ (2017)
- ordered, upward, general: $O(nc^{\sqrt{\log n}})$ (1999)
- ...

see [Chan, 2017] for more

$\rightarrow$ unknown, whether most bounds are tight