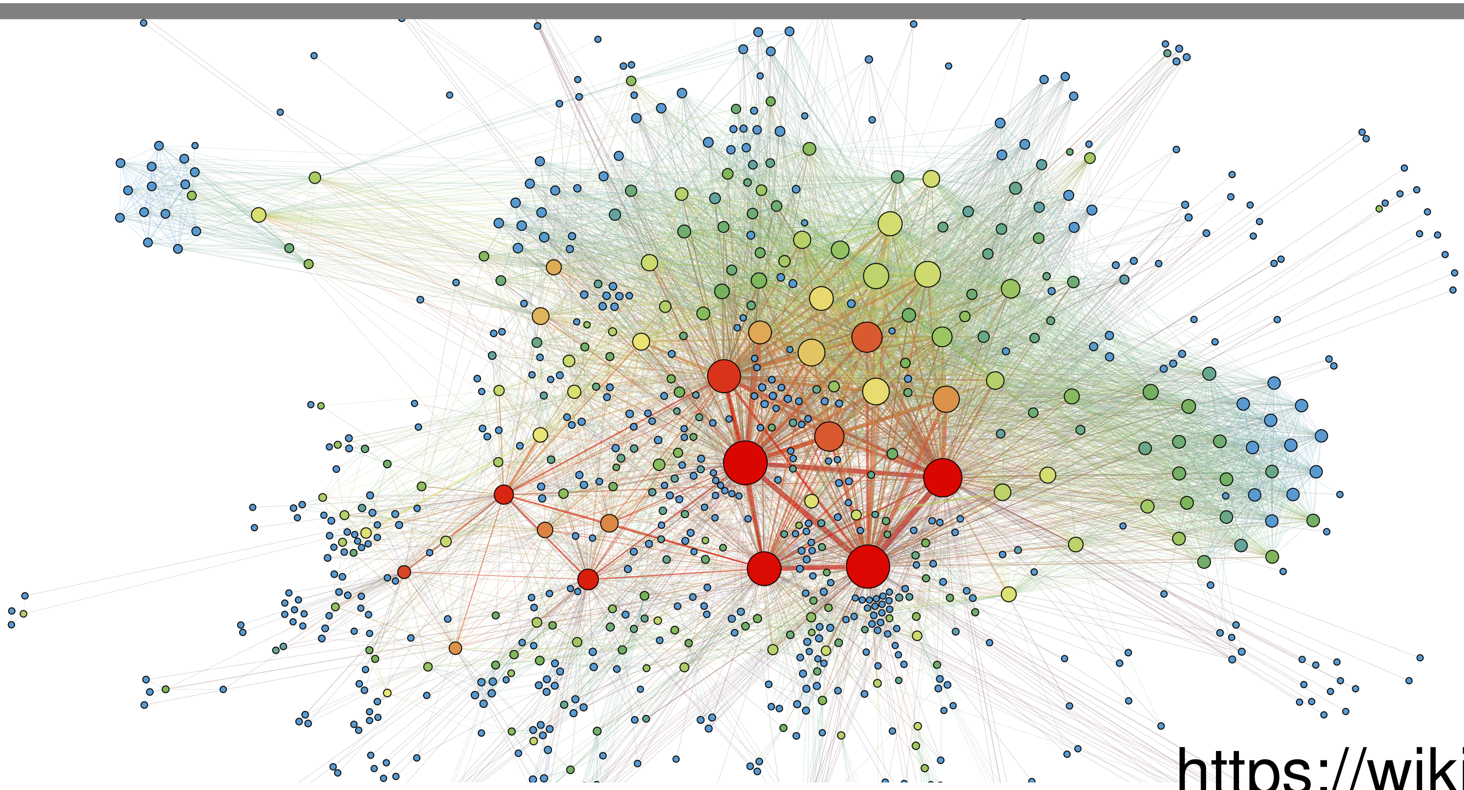
Algorithm for
Visualization of General Graphs

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



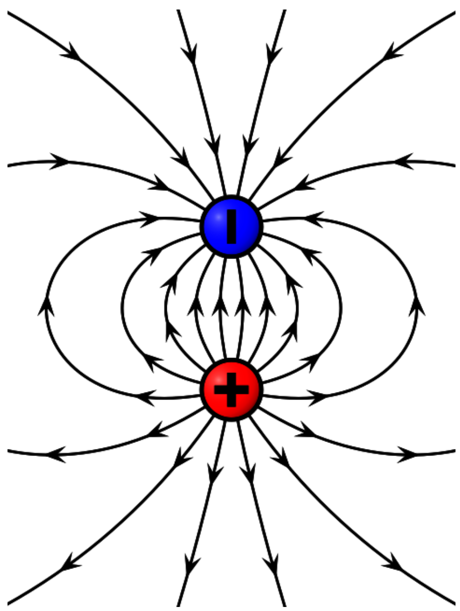
https://wikipedia.org

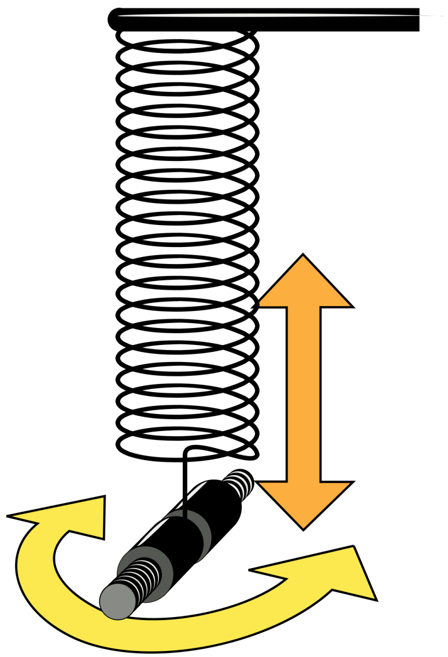
Lecture Overview

- Introduction and How to draw a general graph
- Eades algorithm
- Fruchterman-Reingold algorithm
- Improvements/Modifications
- Speed up (with quadtree)
- Other versions of force-directed algorithm

Introduction

- Method for visualization of general graphs inspired by physical analogies
- The methods are very popular: intuitiveness, easy to program, generality, fairly satisfactory results, easily adaptable for applications...

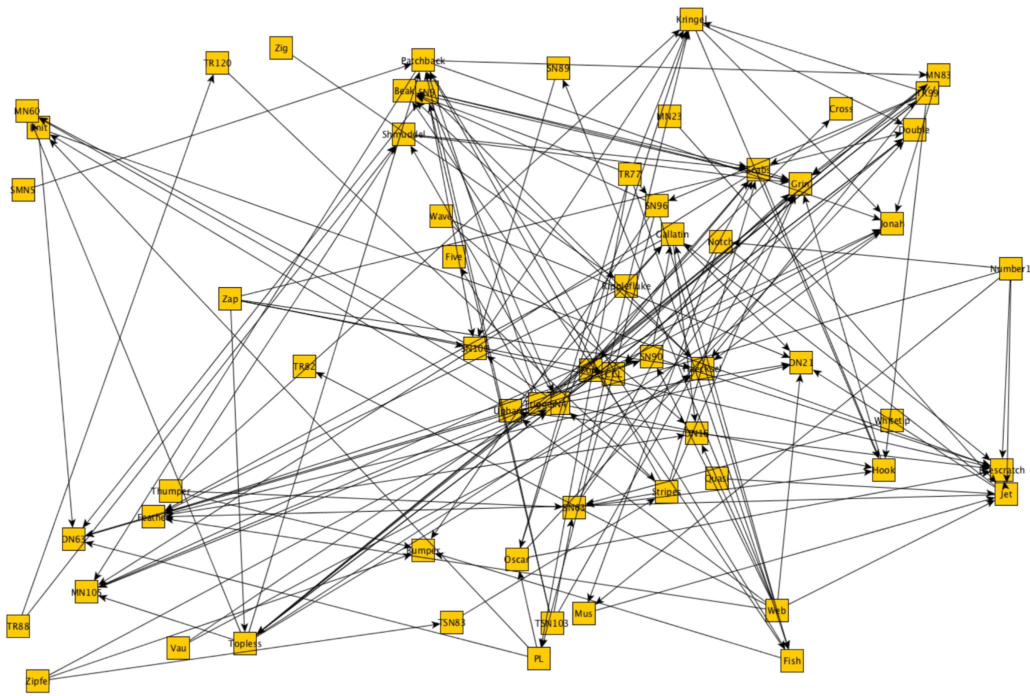


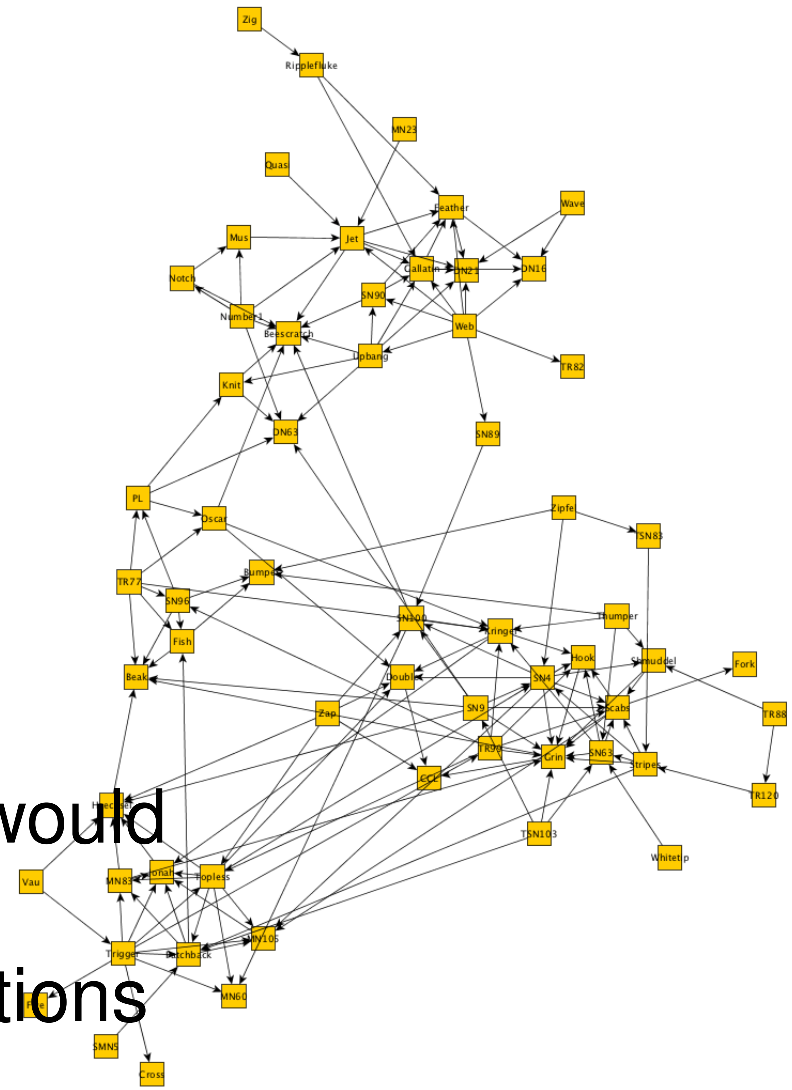



General Layout Problem

Given: Graph $G = (V, E)$

Find: Clear and readable drawing of G





Which quality metrics would you optimize?

Which drawing conventions would we apply?

General Layout Problem

Given: Graph $G = (V, E)$

Find: Clear and readable drawing of G

Criteria:

- adjacent nodes are close
- non-adjacent far apart
- edges short, straight-line, similar length
- densly connected parts (clusters) form communities
- as few crossings as possible
- nodes distributed evenly

General Layout Problem

Given: Graph $G = (V, E)$

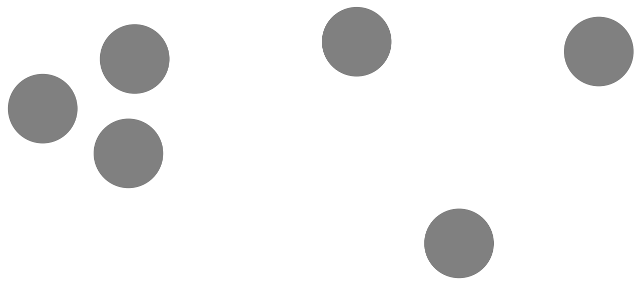
Find: Clear and readable drawing of G

Criteria:

- adjacent nodes are close
- non-adjacent far apart
- edges short, straight-line, similar length
- densly connected parts (clusters) form communities
- as few crossings as possible
- nodes distributed evenly

Gestalt Principle of human perception

Proximity



General Layout Problem

Given: Graph $G = (V, E)$

Find: Clear and readable drawing of G

Criteria:

- adjacent nodes are close
- non-adjacent far apart
- edges short, straight-line, similar length
- densly connected parts (clusters) form communities
- as few crossings as possible
- nodes distributed evenly

⚠ Optimization criteria partially contradict each other

General Layout Problem

Given: Graph $G = (V, E)$

Find: Clear and readable drawing of G

Criteria:

- adjacent nodes are close
- non-adjacent far apart
- edges short, straight-line, similar length
- densly connected parts (clusters) form communities
- as few crossings as possible
- nodes distributed evenly

⚠ Optimization criteria partially contradict each other

General Layout Problem

Given: Graph $G = (V, E)$

Find: Clear and readable drawing of G

Criteria:

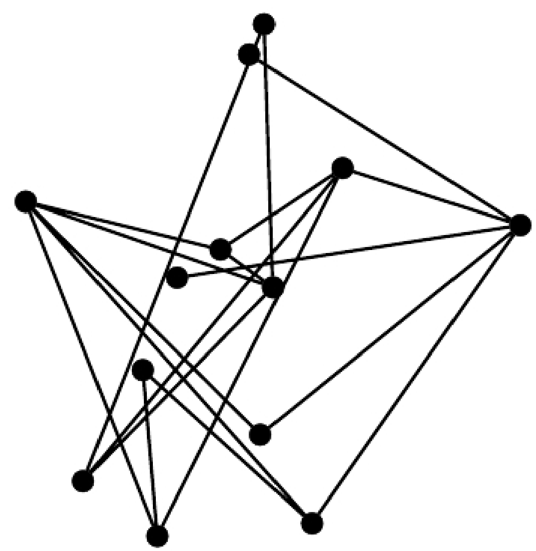
- adjacent nodes are close
- non-adjacent far apart
- edges short, straight-line, similar length
- densly connected parts (clusters) form communities
- as few crossings as possible
- nodes distributed evenly

! Optimization criteria partially contradict each other

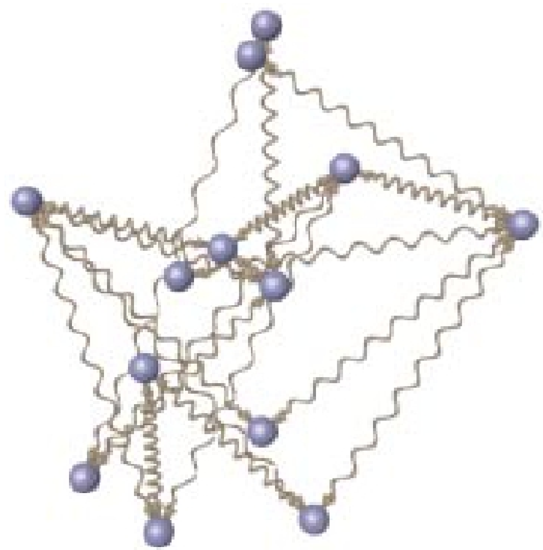
NP-hard for

- edge lengths $\{1, 2\}$ [Saxe, '80]
- planar drawing with unit edge length [Eades, Wormald, '90]

Physical Model

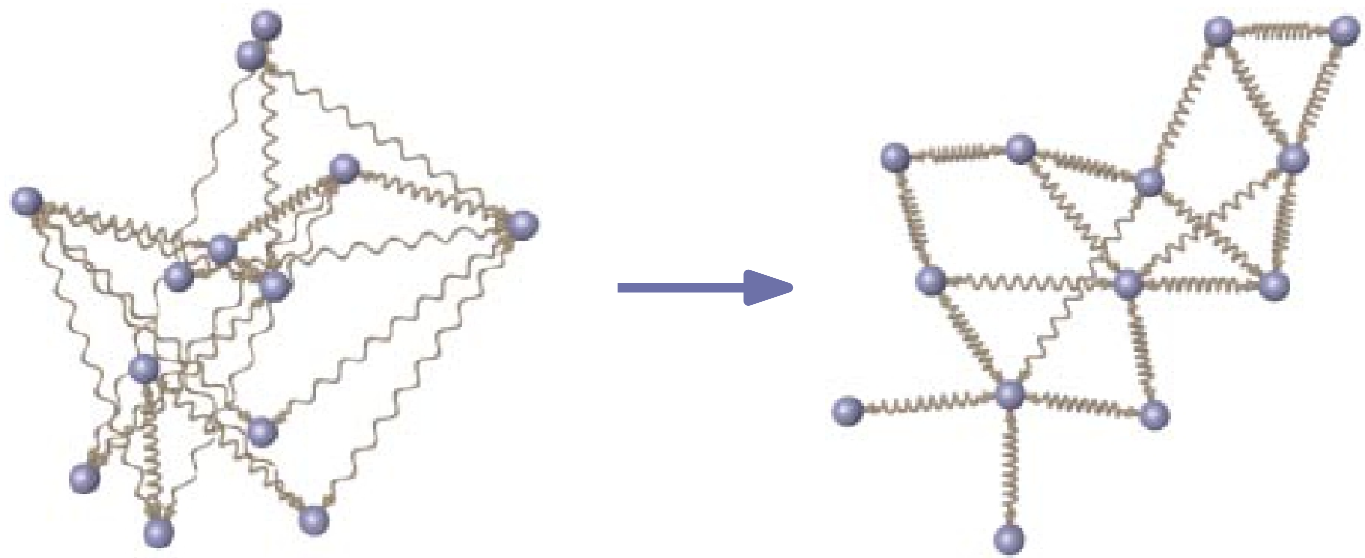


Physical Model



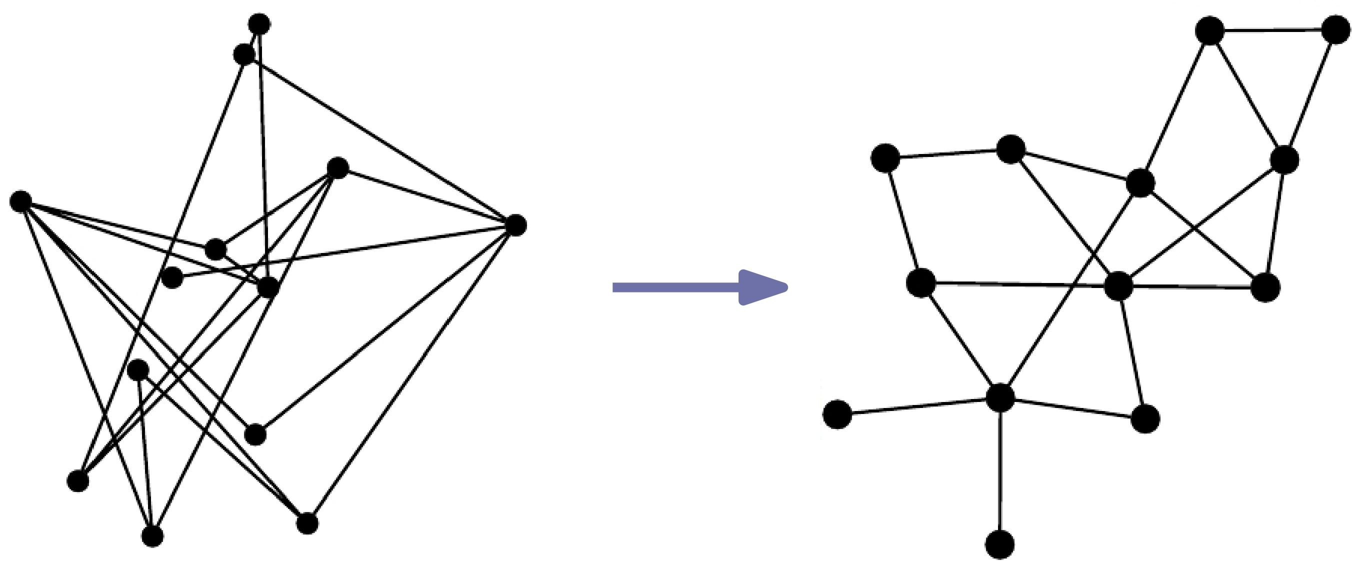
[Eades, '84] "To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system . . .

Physical Model



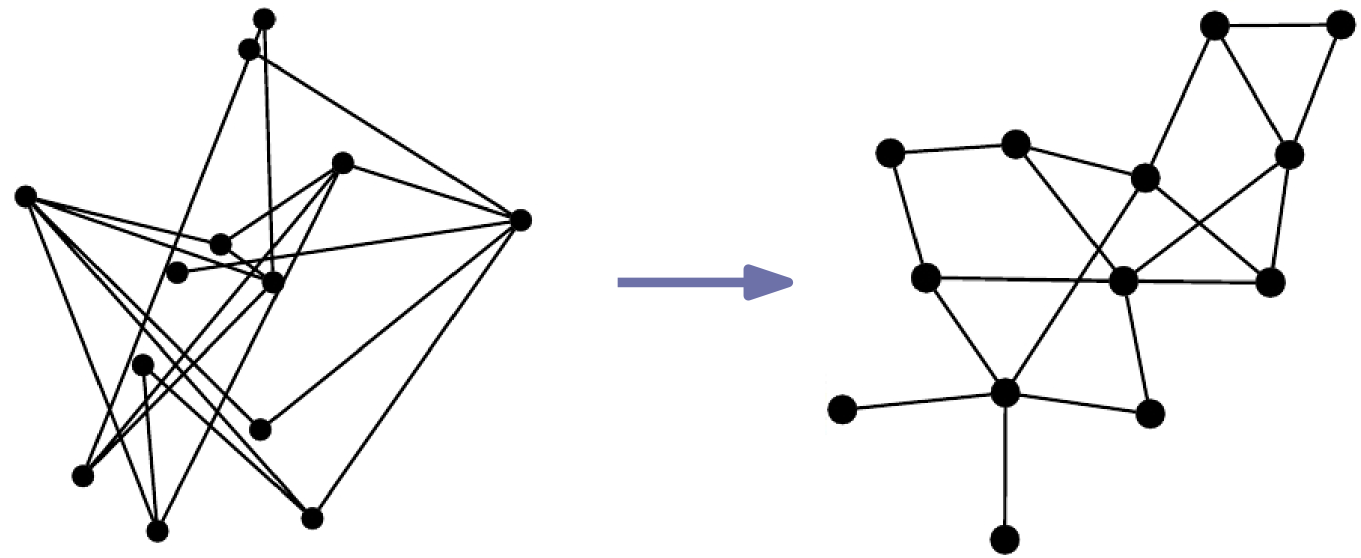
[Eades, ’84] “To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system . . . The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state.”

Physical Model



[Eades, '84] "To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system . . . The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state."

Physical Model



So-called **spring-embedder** algorithms that work according to this or similar principles are among the most frequently used graph-drawing methods in practice.

[Ea … gs
and … …
The …
spring forces on the rings move the system to a minimal energy state.”

Notation

$\ell = \ell(e)$	ideal spring length for edge e
$p_v = (x_v, y_v)$	position of node v
$\|\|p_u - p_v\|\|$	Euclidean distance between u and v
$\overrightarrow{p_u p_v}$	unit vector pointing from u to v

Spring-Embedder (Eades, 1984)

- repulsive force between two non-adjacent nodes u and v

$$f_{\text{rep}}(p_u, p_v) = \frac{c_{\text{rep}}}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$$

Spring-Embedder (Eades, 1984)

- repulsive force between two non-adjacent nodes u and v

$$f_{\text{rep}}(p_u, p_v) = \frac{c_{\text{rep}}}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$$

- attractive force between adjacent vertices u and v

$$f_{\text{spring}}(p_u, p_v) = c_{\text{spring}} \cdot \log \frac{||p_u - p_v||}{\ell} \cdot \overrightarrow{p_v p_u}$$

Spring-Embedder (Eades, 1984)

- repulsive force between two non-adjacent nodes u and v

$$f_{\text{rep}}(p_u, p_v) = \frac{c_{\text{rep}}}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$$

- attractive force between adjacent vertices u and v

$$f_{\text{spring}}(p_u, p_v) = c_{\text{spring}} \cdot \log \frac{||p_u - p_v||}{\ell} \cdot \overrightarrow{p_v p_u}$$

- The values $c_{spring} = 2$, $\ell = 1$, $c_{rep} = 1$ are appropriate for most graphs

Spring-Embedder (Eades, 1984)

- repulsive force between two non-adjacent nodes u and v

$$f_{\text{rep}}(p_u, p_v) = \frac{c_{\text{rep}}}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$$

- attractive force between adjacent vertices u and v

$$f_{\text{spring}}(p_u, p_v) = c_{\text{spring}} \cdot \log \frac{||p_u - p_v||}{\ell} \cdot \overrightarrow{p_v p_u}$$

- The values $c_{spring} = 2$, $\ell = 1$, $c_{rep} = 1$ are appropriate for most graphs
- resulting displacement vector for node v

$$F_v = \sum_{u:\{u,v\} \notin E} f_{\text{rep}}(p_u, p_v) + \sum_{u:\{u,v\} \in E} f_{\text{spring}}(p_u, p_v)$$

Spring-Embedder (Eades, 1984)

- repulsive force between two non-adjacent nodes u and v

Scalars!

$$f_{\text{rep}}(p_u, p_v) = \boxed{\frac{c_{\text{rep}}}{||p_v - p_u||^2}} \cdot \overrightarrow{p_u p_v}$$

- attractive force between adjacent vertices u and v

$$f_{\text{spring}}(p_u, p_v) = \boxed{c_{\text{spring}} \cdot \log \frac{||p_u - p_v||}{\ell}} \cdot \overrightarrow{p_v p_u}$$

- The values $c_{spring} = 2$, $\ell = 1$, $c_{rep} = 1$ are appropriate for most graphs

- resulting displacement vector for node v

$$F_v = \sum_{u:\{u,v\} \notin E} f_{\text{rep}}(p_u, p_v) + \sum_{u:\{u,v\} \in E} f_{\text{spring}}(p_u, p_v)$$

Spring-Embedder (Eades, 1984)

- repulsive force between two non-adjacent nodes *u* and *v*

Vectors!

$$f_{\text{rep}}(p_u, p_v) = \frac{c_{\text{rep}}}{||p_v - p_u||^2} \cdot \boxed{\overrightarrow{p_u p_v}}$$

- attractive force between adjacent vertices *u* and *v*

$$f_{\text{spring}}(p_u, p_v) = c_{\text{spring}} \cdot \log \frac{||p_u - p_v||}{\ell} \cdot \boxed{\overrightarrow{p_v p_u}}$$

- The values $c_{spring} = 2$, $\ell = 1$, $c_{rep} = 1$ are appropriate for most graphs

- resulting displacement vector for node *v*

$$F_v = \sum_{u:\{u,v\} \notin E} f_{\text{rep}}(p_u, p_v) + \sum_{u:\{u,v\} \in E} f_{\text{spring}}(p_u, p_v)$$

Intermission: Vectors vs. Scalars in Code

```
import numpy as np

point1 = np.array([1.0, 0.0])
point2 = np.array([0.0, 2.0])

force_intensity = C_REP / np.linalg.norm(point1 - point2)
force_direction = point2 - point1

force = force_direction * force_intensity
```

- `force_intensity` is a scalar
- `force_direction` is a vector!
- `force` is a vector!

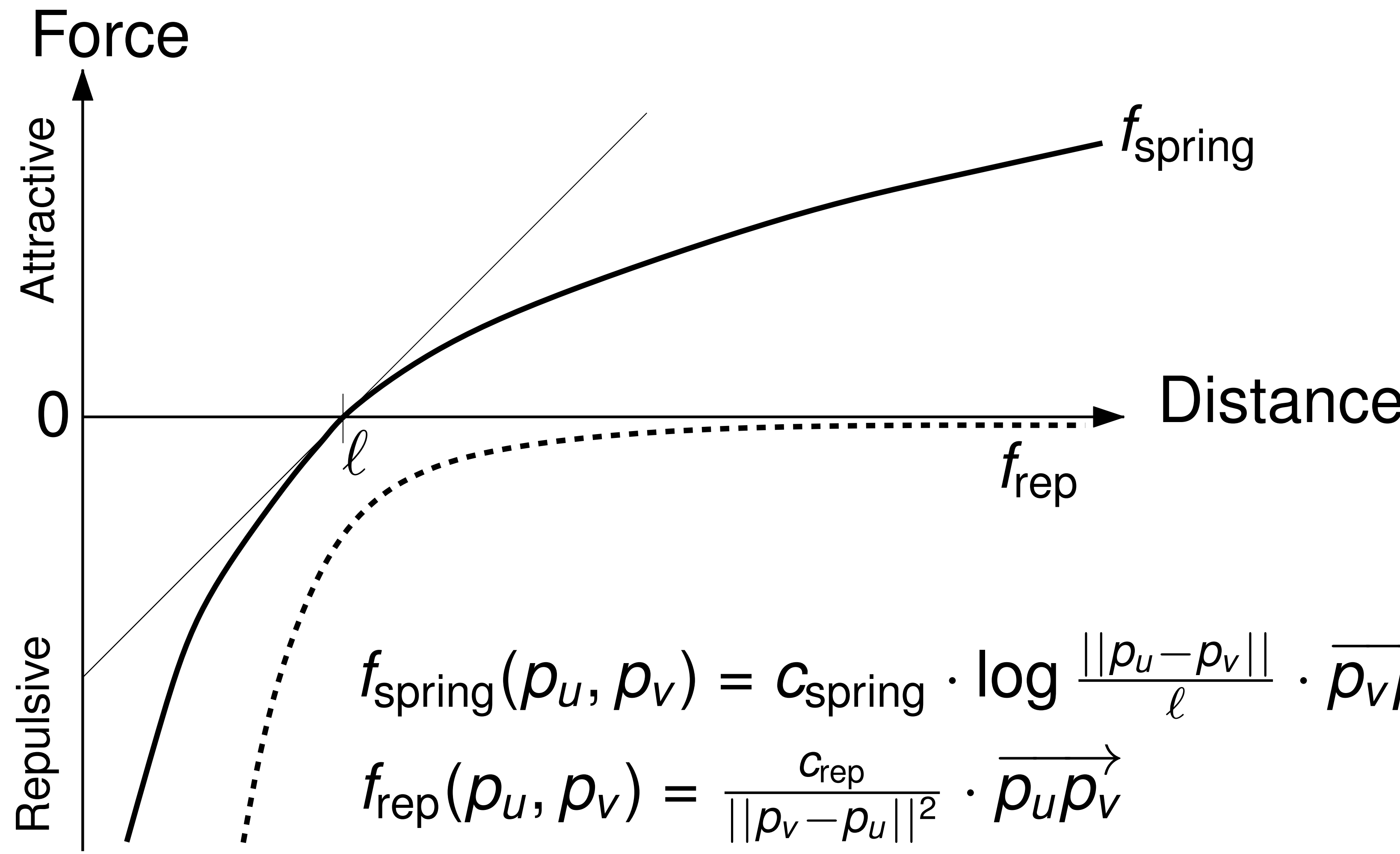
Intermission: Vectors vs. Scalars in Code

```
import numpy as np

point1 = np.array([1.0, 0.0])
point2 = np.array([0.0, 2.0])

force_intensity = C_REP / np.linalg.norm(point1 - point2)
force_direction = (point2 - point1) / np.linalg.norm(point1 - point2)

force = force_direction * force_intensity
```

- `force_intensity` is a scalar
- `force_direction` is a vector!
- `force` is a vector!

- Do **not** confuse the data types. This saves you from wrongly calculating/applying forces.

Diagram of Spring-Embedder Forces (Eades, 1984)



$$f_{\text{spring}}(p_u, p_v) = c_{\text{spring}} \cdot \log \frac{||p_u - p_v||}{\ell} \cdot \overrightarrow{p_v p_u}$$

$$f_{\text{rep}}(p_u, p_v) = \frac{c_{\text{rep}}}{||p_v - p_u||^2} \cdot \overrightarrow{p_u p_v}$$

$f_{\text{spring}}(p_u, p_v)$ contributes to attraction when $||p_u - p_v|| > \ell$ and to repulsion when $||p_u - p_v|| < \ell$

$f_{\text{rep}}(p_u, p_v)$ is approaching zero as the distance grows, faster for smaller c_{rep}

Algorithm Spring-Embedder (Eades, 1984)

Input: $G = (V, E)$ connected undirected graph with initial placement $p = (p_v)_{v \in V}$, number of interations $K \in \mathbb{N}$, threshold $\varepsilon > 0$, constant $\delta > 0$

Output: Layout p with ”low internal stress”'

$t \leftarrow 1$

while $t < K$ **and** $\max_{v \in V} \|F_v(t)\| > \varepsilon$ **do**

  foreach $v \in V$ **do**

    $F_v(t) \leftarrow \sum_{u:\{u,v\} \notin E} f_{\text{rep}}(p_u, p_v) + \sum_{u:\{u,v\} \in E} f_{\text{spring}}(p_u, p_v)$

  foreach $v \in V$ **do**

    $p_v \leftarrow p_v + \delta \cdot F_v(t)$

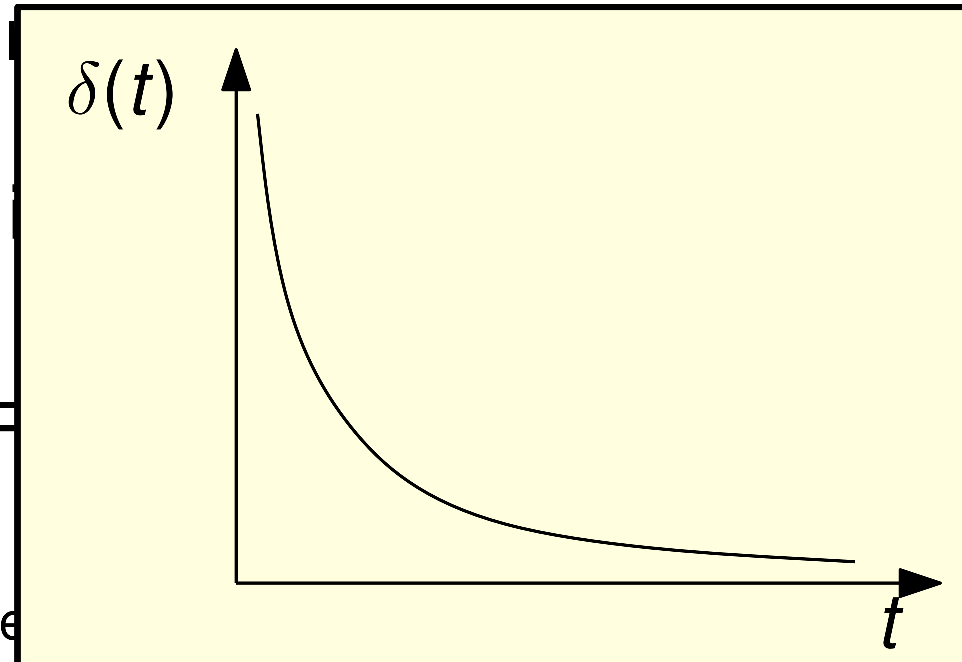
  $t \leftarrow t + 1$

Algorithm Spring-Embedder (Eades, 1984)

Input: $G = (V, E)$ connected undirected graph with initial placement $p = (p_v)_{v \in V}$, number of interations $K \in \mathbb{N}$, thr
$\delta > 0$

Output: Layout p with ”low i

$t \leftarrow 1$
while $t < K$ **and** $\max_{v \in V} \|F$
  foreach $v \in V$ **do**
    $F_v(t) \leftarrow \sum_{u:\{u,v\} \notin E} f_{\mathrm{re}}$
    $\sum_{u:\{u,v\} \in E} f_{\mathrm{spring}}(p_u, p_v)$
  foreach $v \in V$ **do**
    $p_v \leftarrow p_v + \delta(t) \cdot F_v(t)$
  $t \leftarrow t + 1$



Variant: Fruchterman & Reingold (1991)

Model:

- repulsive force between **all** node pairs *u* and *v*

$$f_{\text{rep}}(p_u, p_v) = \frac{\ell^2}{||p_v - p_u||} \cdot \overrightarrow{p_u p_v}$$

Variant: Fruchterman & Reingold (1991)

Model:

- repulsive force between **all** node pairs *u* and *v*

$$f_{\text{rep}}(p_u, p_v) = \frac{\ell^2}{||p_v - p_u||} \cdot \overrightarrow{p_u p_v}$$

- attractive force between two adjacent nodes *u* and *v*

$$f_{\text{attr}}(p_u, p_v) = \frac{||p_u - p_v||^2}{\ell} \cdot \overrightarrow{p_v p_u}$$

Variant: Fruchterman & Reingold (1991)

Model:

- repulsive force between **all** node pairs *u* and *v*

$$f_{\text{rep}}(p_u, p_v) = \frac{\ell^2}{||p_v - p_u||} \cdot \overrightarrow{p_u p_v}$$

- attractive force between two adjacent nodes *u* and *v*

$$f_{\text{attr}}(p_u, p_v) = \frac{||p_u - p_v||^2}{\ell} \cdot \overrightarrow{p_v p_u}$$

- $\ell = C\sqrt{\frac{\text{area}}{\text{number of vertices}}}$ - ideal edge length

Variant: Fruchterman & Reingold (1991)

Model:

- repulsive force between **all** node pairs *u* and *v*

$$f_{\text{rep}}(p_u, p_v) = \frac{\ell^2}{||p_v - p_u||} \cdot \overrightarrow{p_u p_v}$$

- attractive force between two adjacent nodes *u* and *v*

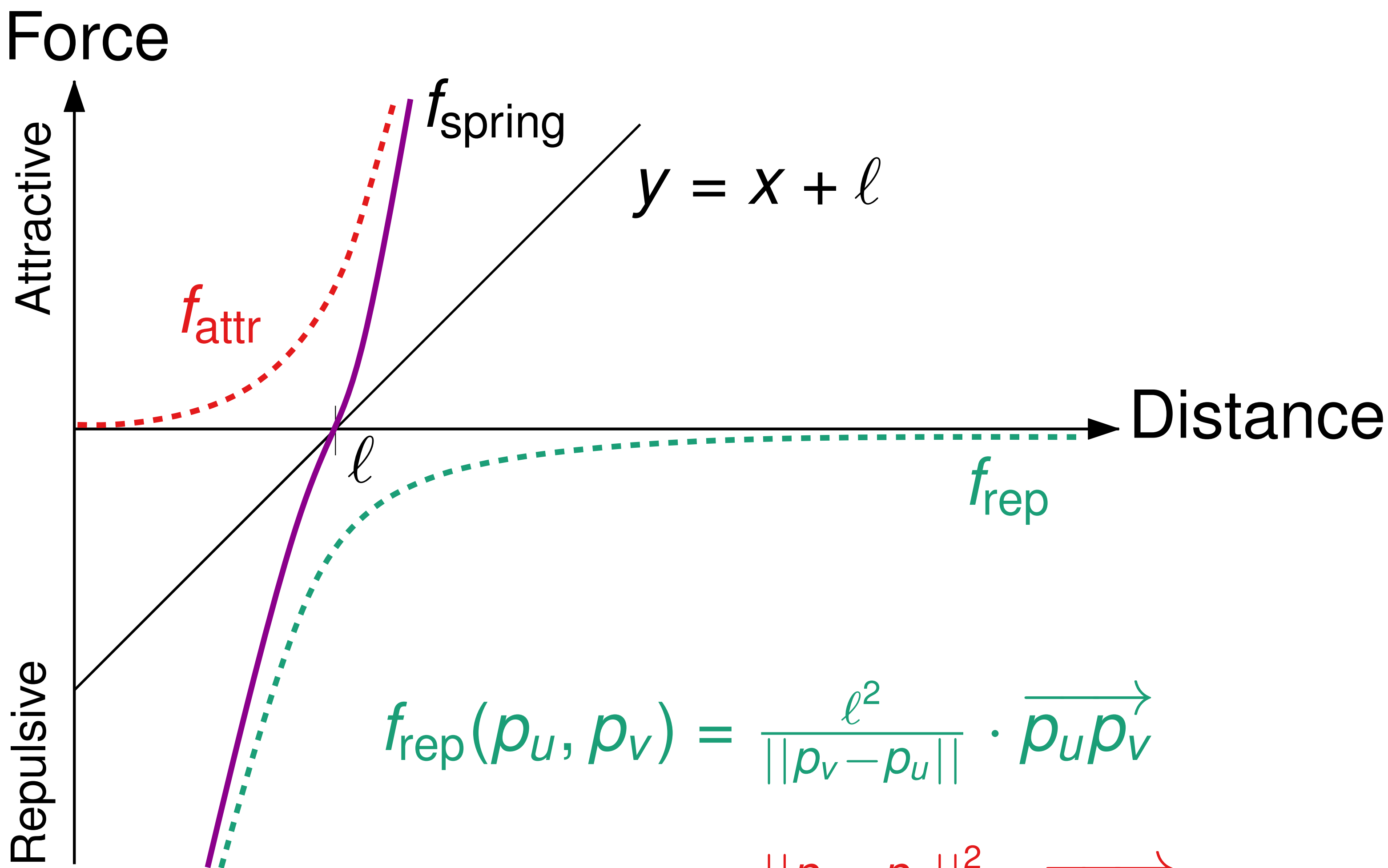
$$f_{\text{attr}}(p_u, p_v) = \frac{||p_u - p_v||^2}{\ell} \cdot \overrightarrow{p_v p_u}$$

- $\ell = C\sqrt{\frac{\text{area}}{\text{number of vertices}}}$ - ideal edge length

- resulting force between adjacent nodes *u* and *v*

$$f_{\text{spring}}(p_u, p_v) = f_{\text{rep}}(p_u, p_v) + f_{\text{attr}}(p_u, p_v)$$

Diagramm of Fruchtermann & Reingold Forces



$$f_{\text{rep}}(p_u, p_v) = \frac{\ell^2}{||p_v - p_u||} \cdot \overrightarrow{p_u p_v}$$

$$f_{\text{attr}}(p_u, p_v) = \frac{||p_u - p_v||^2}{\ell} \cdot \overrightarrow{p_v p_u}$$

$$f_{\text{spring}}(p_u, p_v) = f_{\text{rep}}(p_u, p_v) + f_{\text{attr}}(p_u, p_v)$$

Observe that $f_{\text{spring}}(p_u, p_v) = 0$ for $||p_v - p_u|| = \ell$

Discussion

Advantages

- very simple Algorithm
- good results for small and medium-sized graphs
- emphirically good representation of symmetry and structure

Discussion

Advantages

- very simple Algorithm
- good results for small and medium-sized graphs
- emphirically good representation of symmetry and structure

Disadvantages

- system is not stable at the end
- converging to local minima
- timewise f_{spring} in $\mathcal{O}(|E|)$ and f_{rep} in $\mathcal{O}(|V|^2)$

Discussion

Advantages

- very simple Algorithm
- good results for small and medium-sized graphs
- emphirically good representation of symmetry and structure

Disadvantages

- system is not stable at the end
- converging to local minima
- timewise f_{spring} in $\mathcal{O}(|E|)$ and f_{rep} in $\mathcal{O}(|V|^2)$

Influence

- Original paper by Peter Eades got 1775 citations (in 2019, no update possible)
- Variants of Fruchterman and Reingold algorithms are probably the most popular force-based methods (original paper cited 7457 times (doubled in the past 4 years)
- Basis for **MANY** further ideas

Other Possible Modifications

- **Inertia**

- **Gravitation**

- **Magnetic forces**

Other Possible Modifications

- **Inertia**

  define node mass as $\Phi(v) = 1 + \deg(v)/2$

  set $f_{\text{attr}}(p_u, p_v) \leftarrow f_{\text{attr}}(p_u, p_v) \cdot 1/\Phi(v)$

- **Gravitation**

- **Magnetic forces**

Other Possible Modifications

- **Inertia**

  define node mass as $\Phi(v) = 1 + \deg(v)/2$

  set $f_{\text{attr}}(p_u, p_v) \leftarrow f_{\text{attr}}(p_u, p_v) \cdot 1/\Phi(v)$

- **Gravitation**

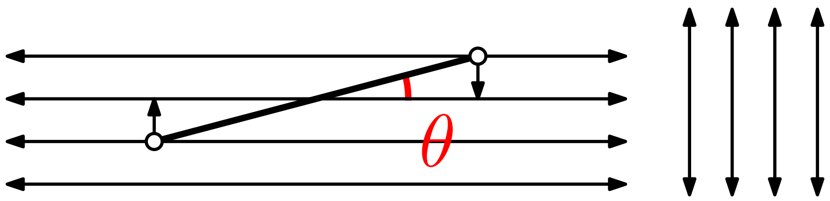
  define barycenter $p_{\text{bary}} = 1/|V| \cdot \sum_{v \in V} p_v$

  $f_{\text{grav}}(p_v) = c_{\text{grav}} \cdot \Phi(v) \cdot \overrightarrow{p_v p_{\text{bary}}}$

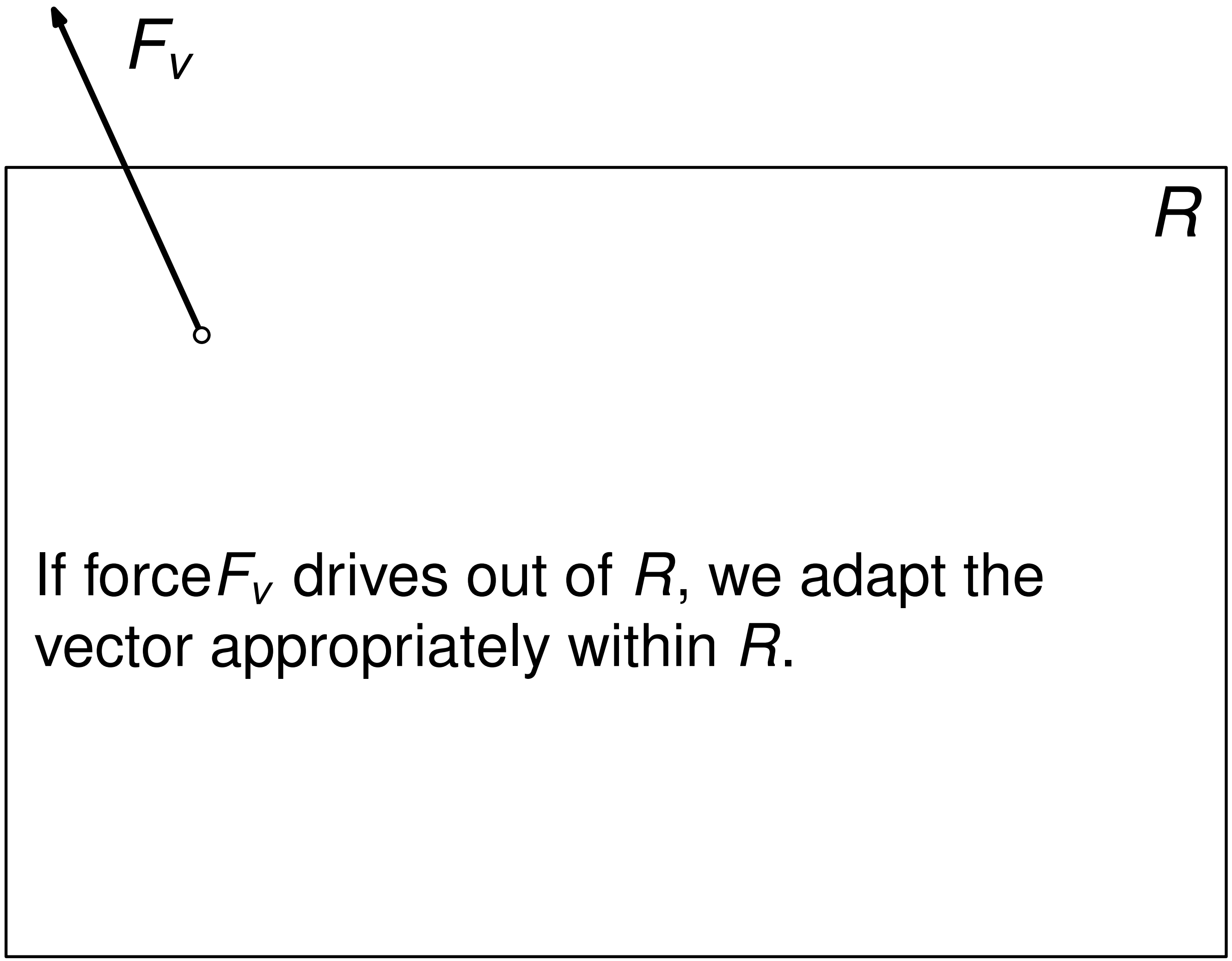
- **Magnetic forces**

Other Possible Modifications

- **Inertia**
  define node mass as $\Phi(v) = 1 + \deg(v)/2$
  set $f_{\text{attr}}(p_u, p_v) \leftarrow f_{\text{attr}}(p_u, p_v) \cdot 1/\Phi(v)$
- **Gravitation**
  define barycenter $p_{\text{bary}} = 1/|V| \cdot \sum_{v \in V} p_v$
  $f_{\text{grav}}(p_v) = c_{\text{grav}} \cdot \Phi(v) \cdot \overrightarrow{p_v p_{\text{bary}}}$
- **Magnetic forces**
  – define magnetic fields (e.g. vertical, horizontal)
  – angle θ between edge and the direction of the field
  – define force that reduces this angle

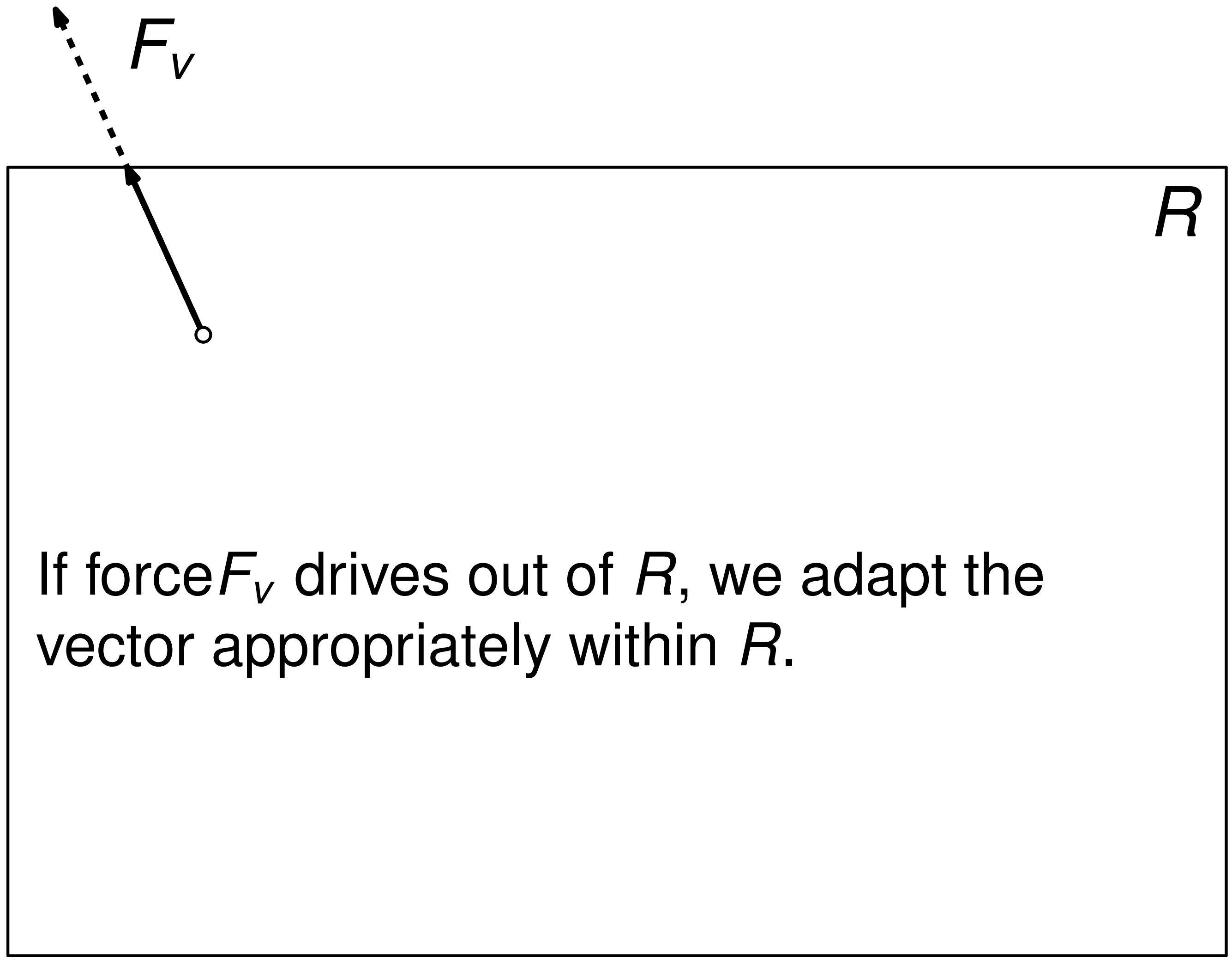


Bounded Drawing Area



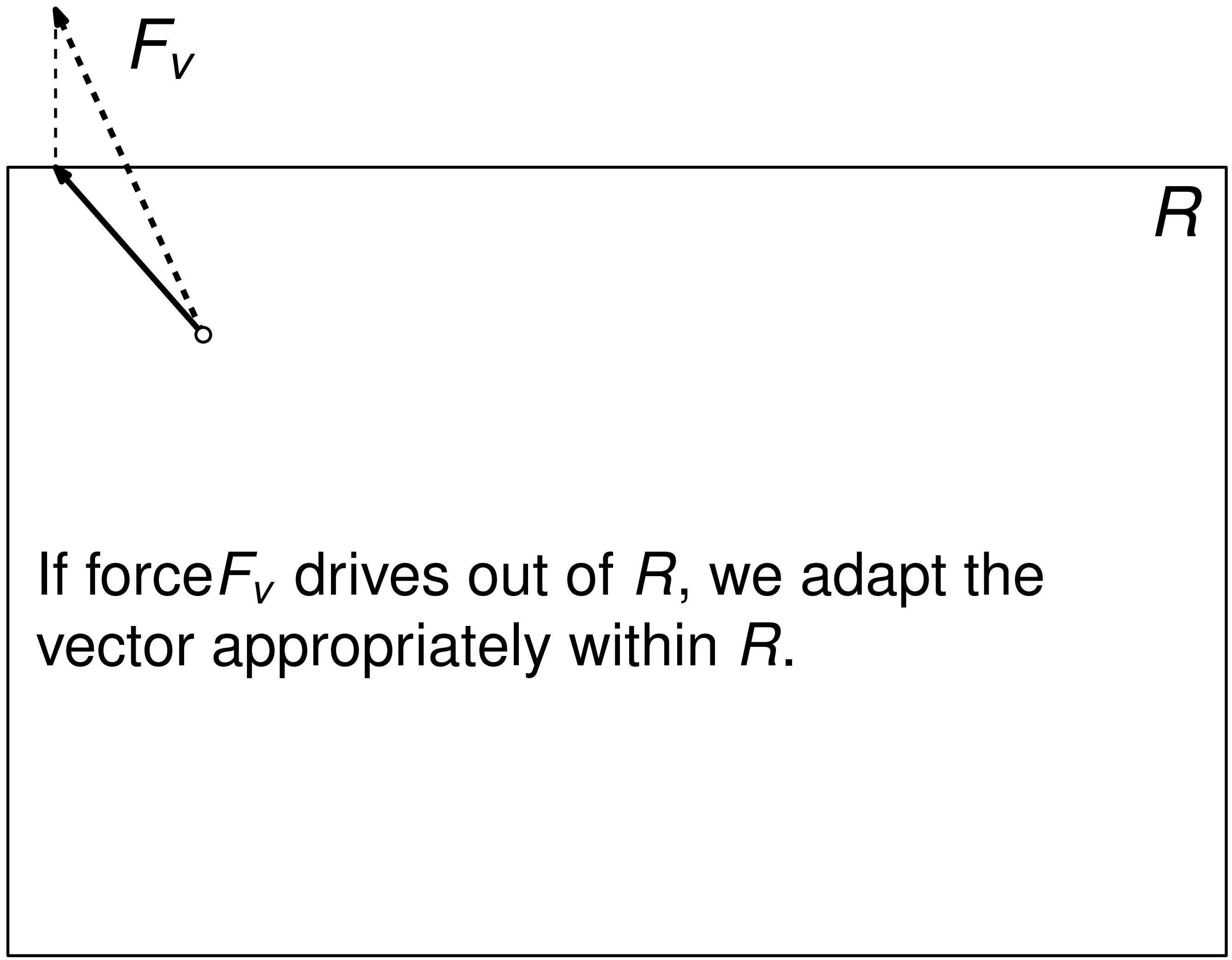
If force F_v drives out of R, we adapt the vector appropriately within R.

Bounded Drawing Area



If force F_v drives out of R, we adapt the vector appropriately within R.

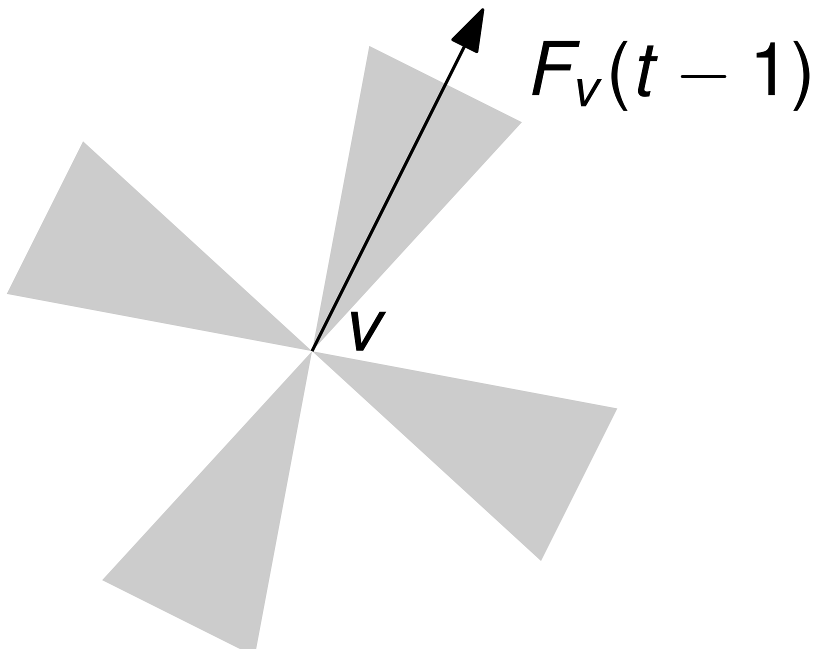
Bounded Drawing Area



If forceF_v drives out of R, we adapt the vector appropriately within R.

Adaptive Displacement (Frick, Ludwig, Mehldau 1995)

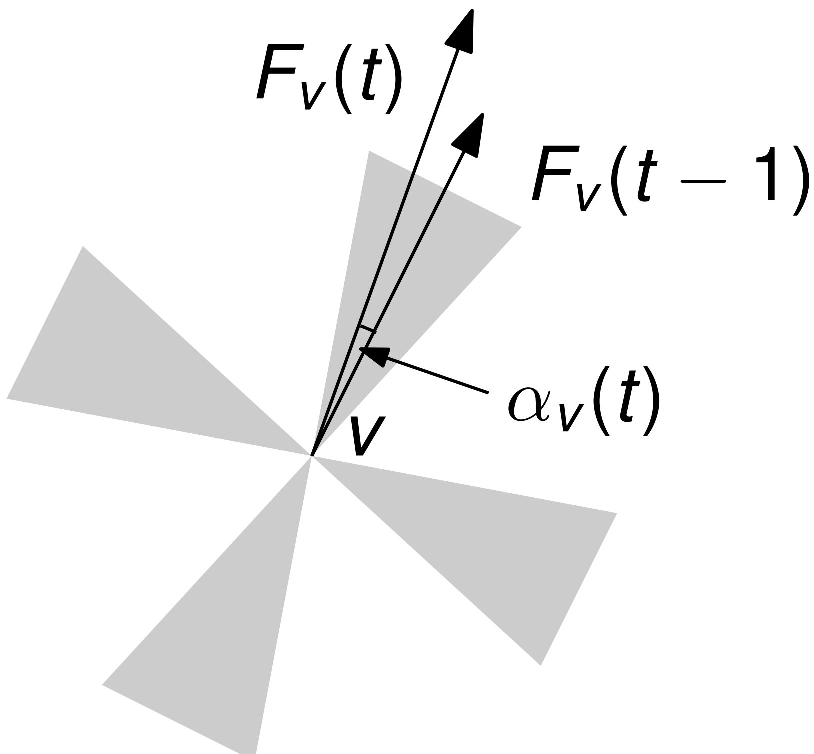
Method to prevent node oscillation and repeated rotations



- store previous displacement vector $F_v(t-1)$

Adaptive Displacement (Frick, Ludwig, Mehldau 1995)

Method to prevent node oscillation and repeated rotations



- store previous displacement vector $F_v(t-1)$

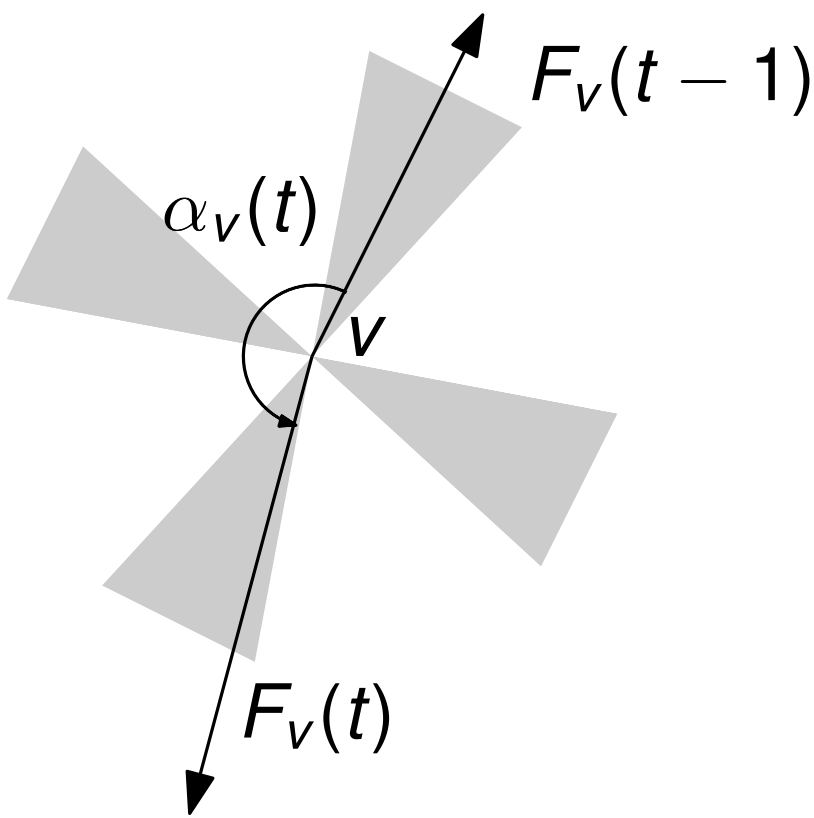
store local temperature for every node v

- $\cos(\alpha_v(t)) \approx 1$:
  similar direction
  $\rightarrow$ increase the temperature of v

Adaptive Displacement (Frick, Ludwig, Mehldau 1995)

Method to prevent node oscillation and repeated rotations

- store previous displacement vector $F_v(t-1)$



store local temperature for every node v

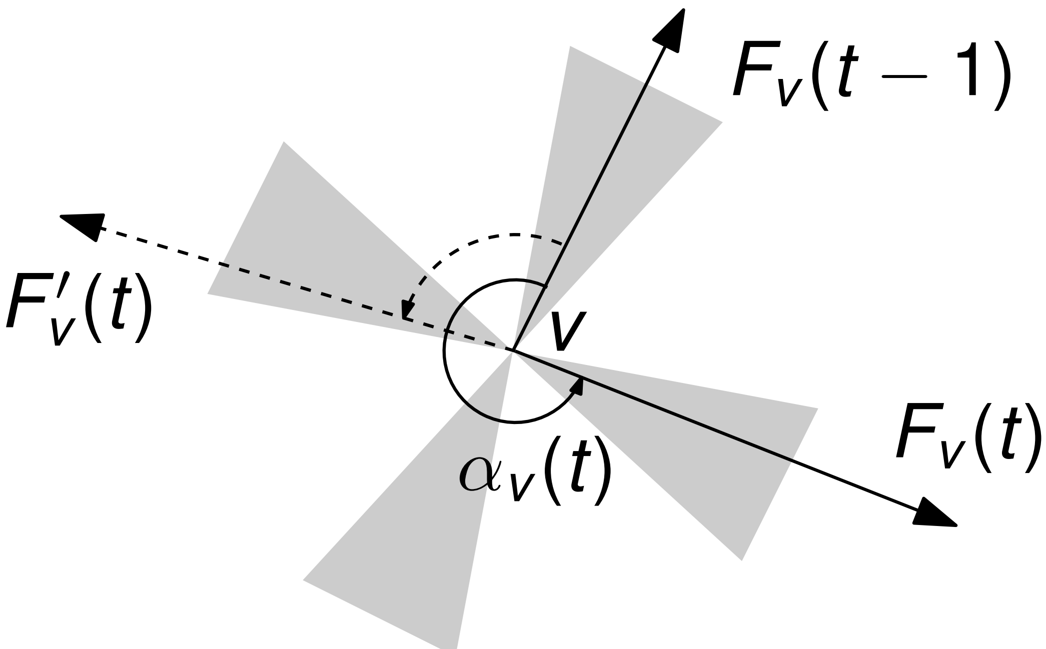
- $\cos(\alpha_v(t)) \approx 1$:
  similar direction
  $\rightarrow$ increase the temperature of v

- $\cos(\alpha_v(t)) \approx -1$:
  oscillation
  $\rightarrow$ reduce the temperature of v

Adaptive Displacement (Frick, Ludwig, Mehldau 1995)

Method to prevent node oscillation and repeated rotations

- store previous displacement vector $F_v(t-1)$



store local temperature for every node v

- $\cos(\alpha_v(t)) \approx 1$:
  similar direction
  $\rightarrow$ increase the temperature of v

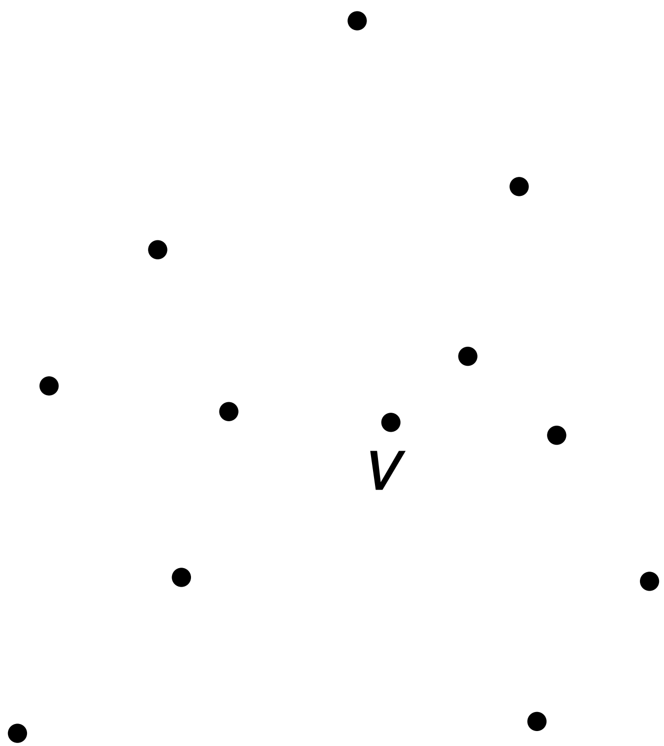
- $\cos(\alpha_v(t)) \approx -1$:
  oscillation
  $\rightarrow$ reduce the temperature of v

- $\cos(\alpha_v(t)) \approx 0$:
  Rotation
  $\rightarrow$ update rotation counter and decrease temperature if necessary

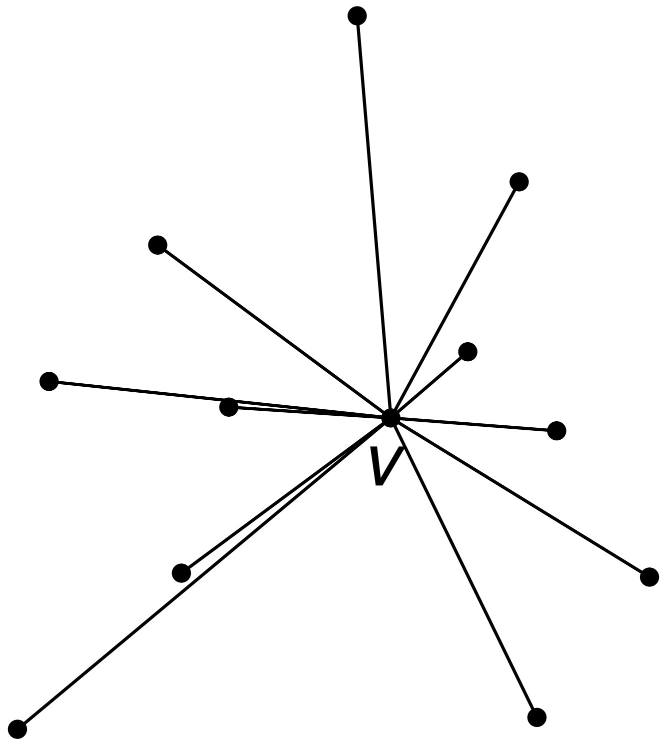
Lecture Overview

- Introduction and How to draw a general graph
- Eades algorithm
- Fruchterman-Reingold algorithm
- Improvements/Modifications
- **Speed-up (with quadtree)**
- **Other versions of force-directed algorithm**

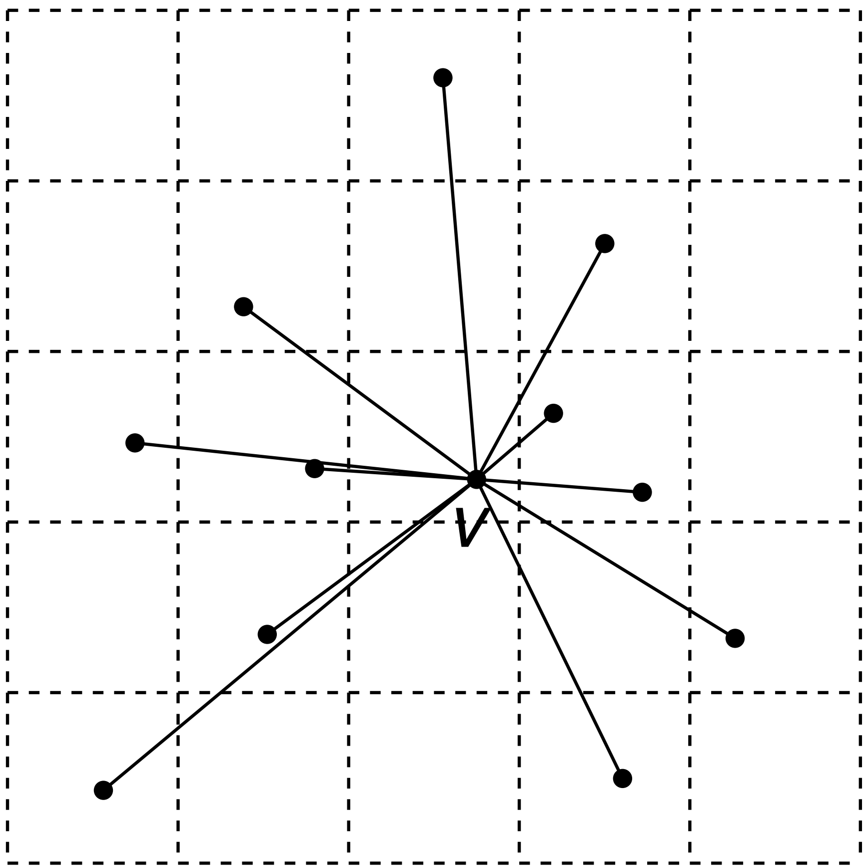
Grid Version (Fruchterman, Reingold, 1990)



Grid Version (Fruchterman, Reingold, 1990)

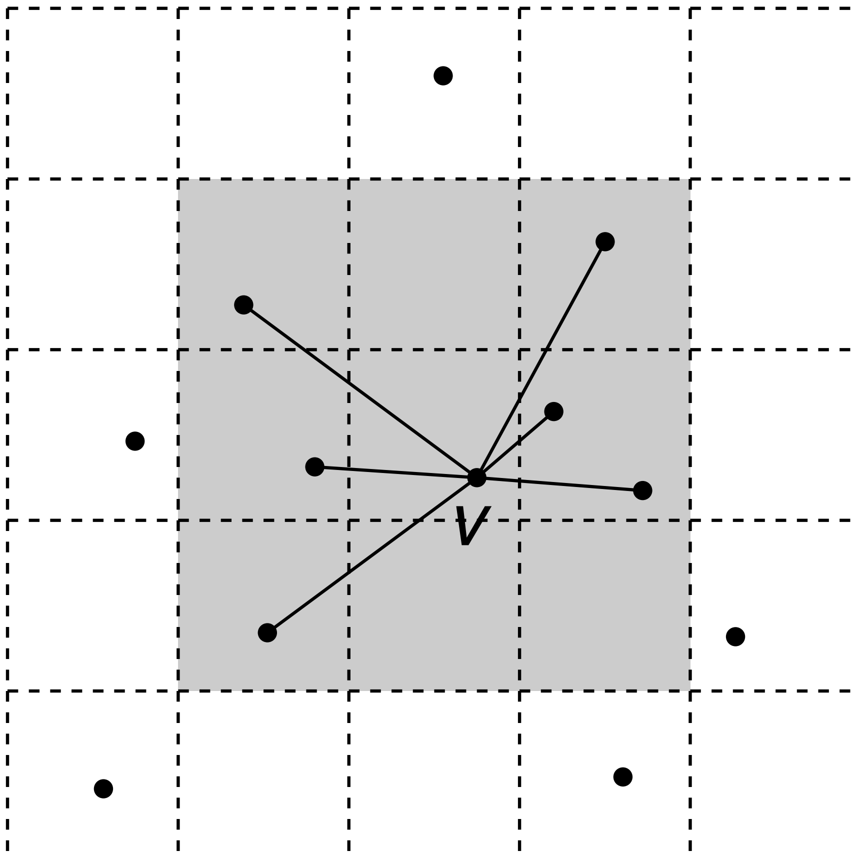


Grid Version (Fruchterman, Reingold, 1990)



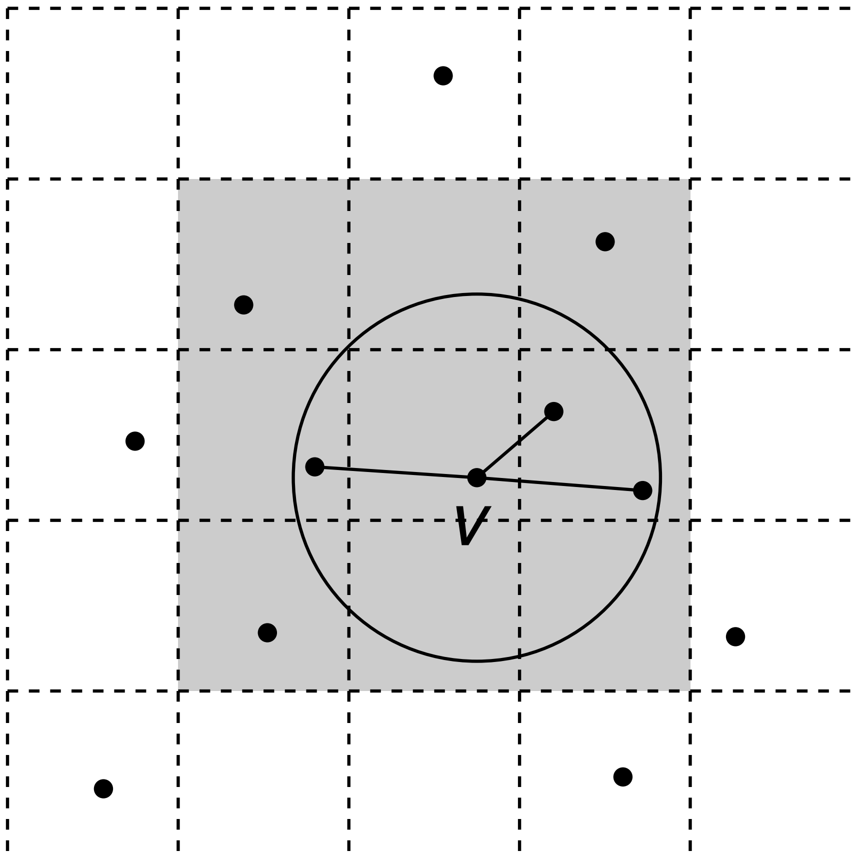
- subdivide plane by a grid

Grid Version (Fruchterman, Reingold, 1990)



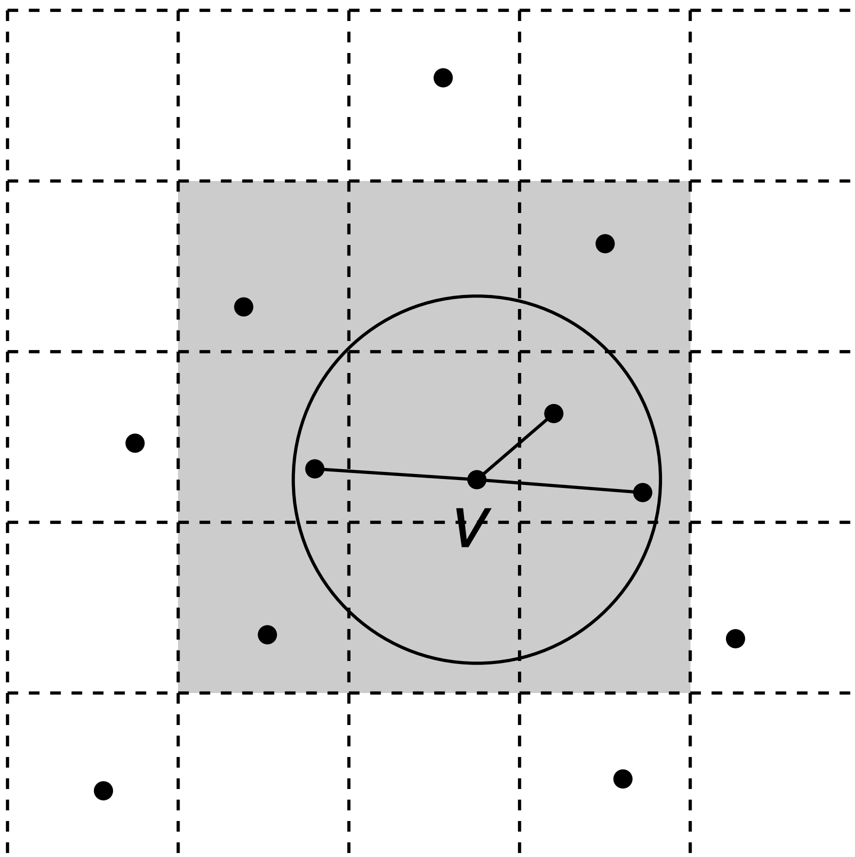
- subdivide plane by a grid
- compute repulsive forces only for the nodes in the neighbouring cells

Grid Version (Fruchterman, Reingold, 1990)



- subdivide plane by a grid
- compute repulsive forces only for the nodes in the neighbouring cells
- and only when the distance is at most $d_{\max}$

Grid Version (Fruchterman, Reingold, 1990)

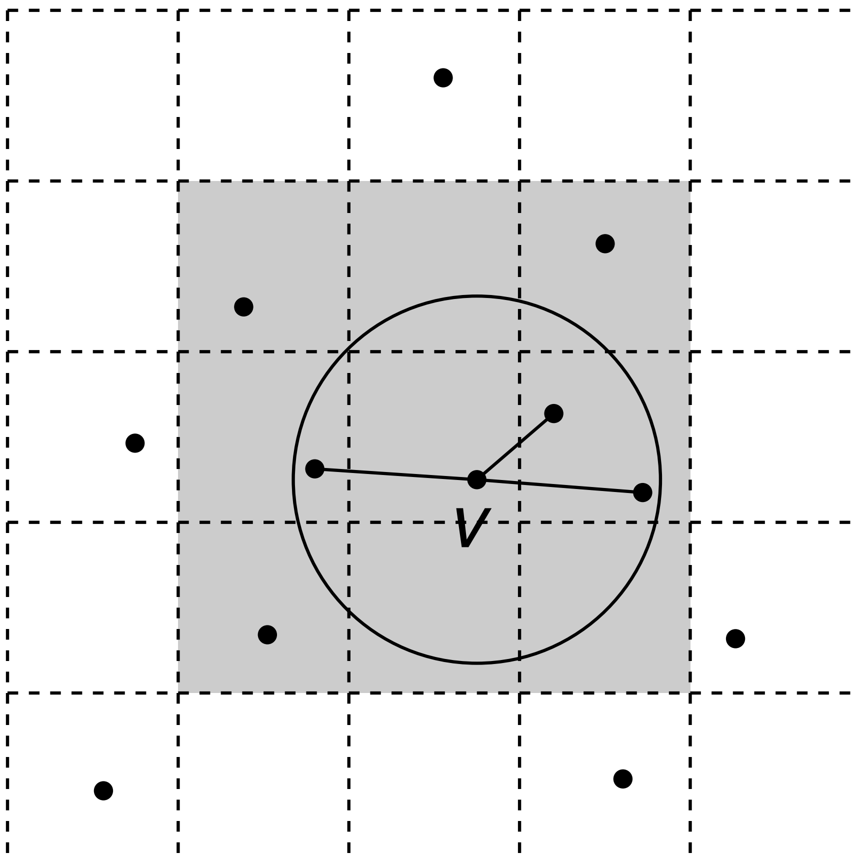


- subdivide plane by a grid
- compute repulsive forces only for the nodes in the neighbouring cells
- and only when the distance is at most $d_{\max}$

Discussion

- meaningful idea to improve runtime
- worst-case no advantage
- Quality loss

Grid Version (Fruchterman, Reingold, 1990)



- subdivide plane by a grid
- compute repulsive forces only for the nodes in the neighbouring cells
- and only when the distance is at most $d_{\max}$

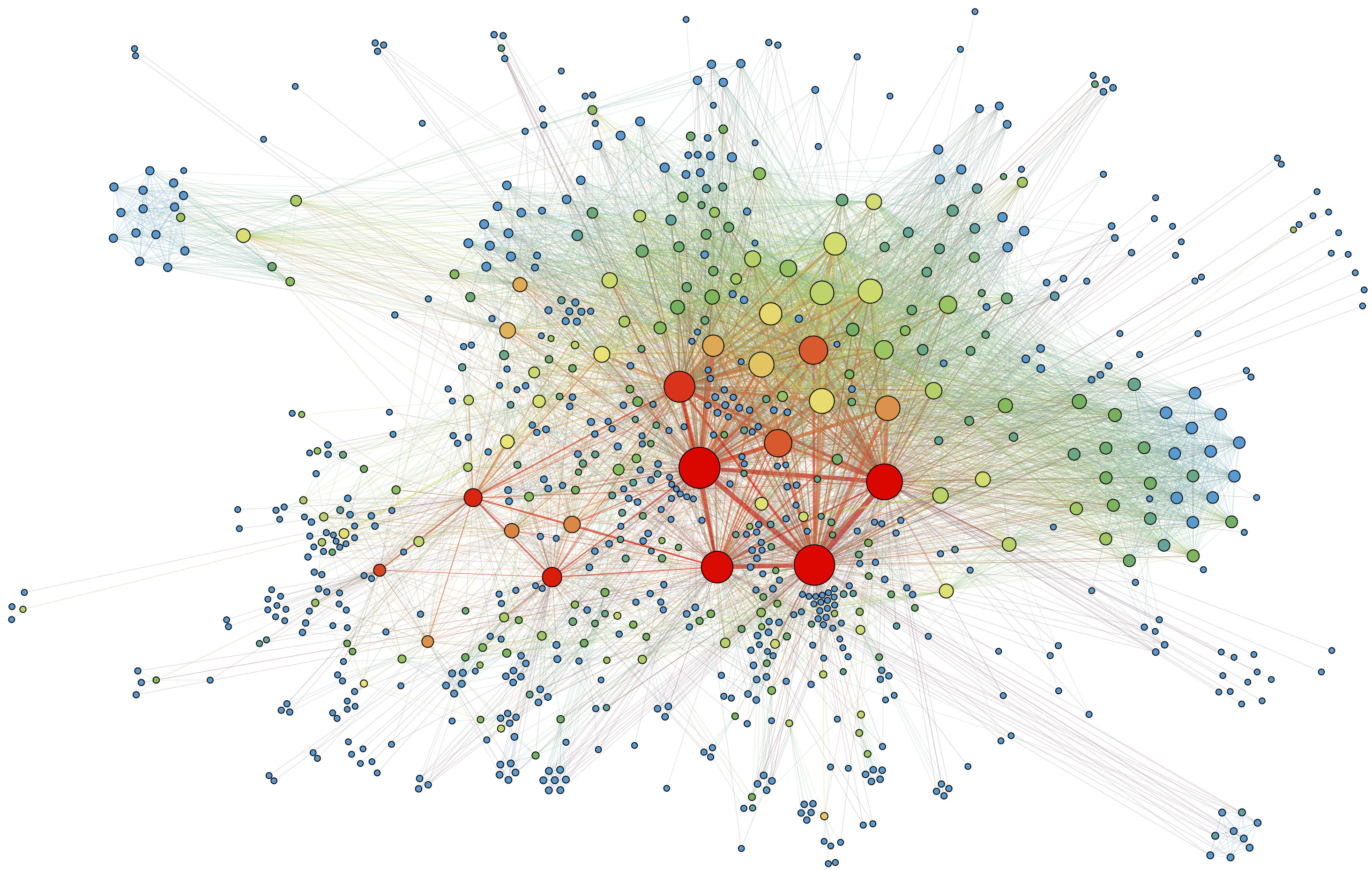
Discussion

- meaningful idea to improve runtime

- worst-case no advantage
- Quality loss

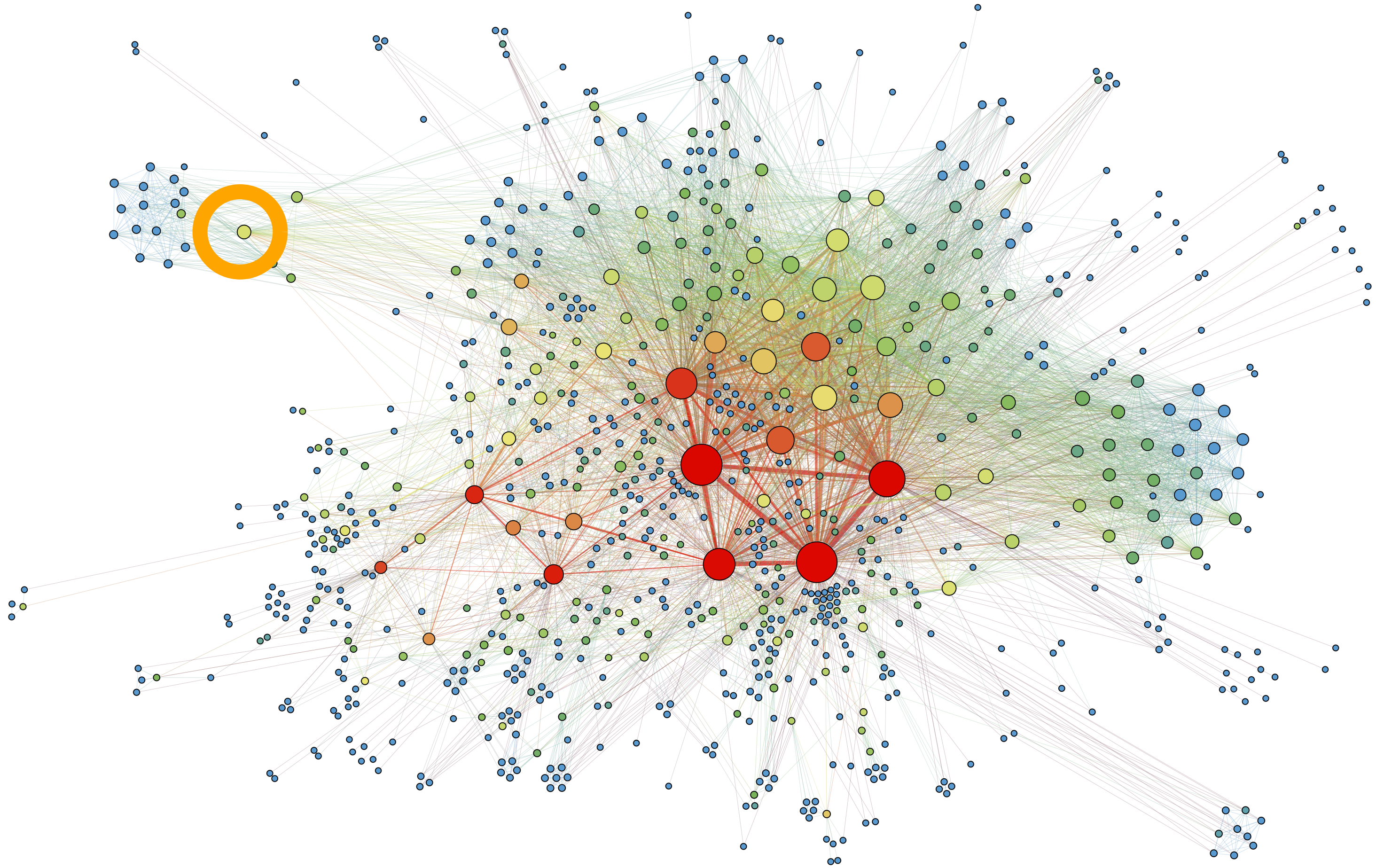
Speed-up with Quad-Tree

Main idea : when computing repulsive force for a vertex *v*, for groups of vertices that are far apart from *v* we do not need to account on individual force-influences



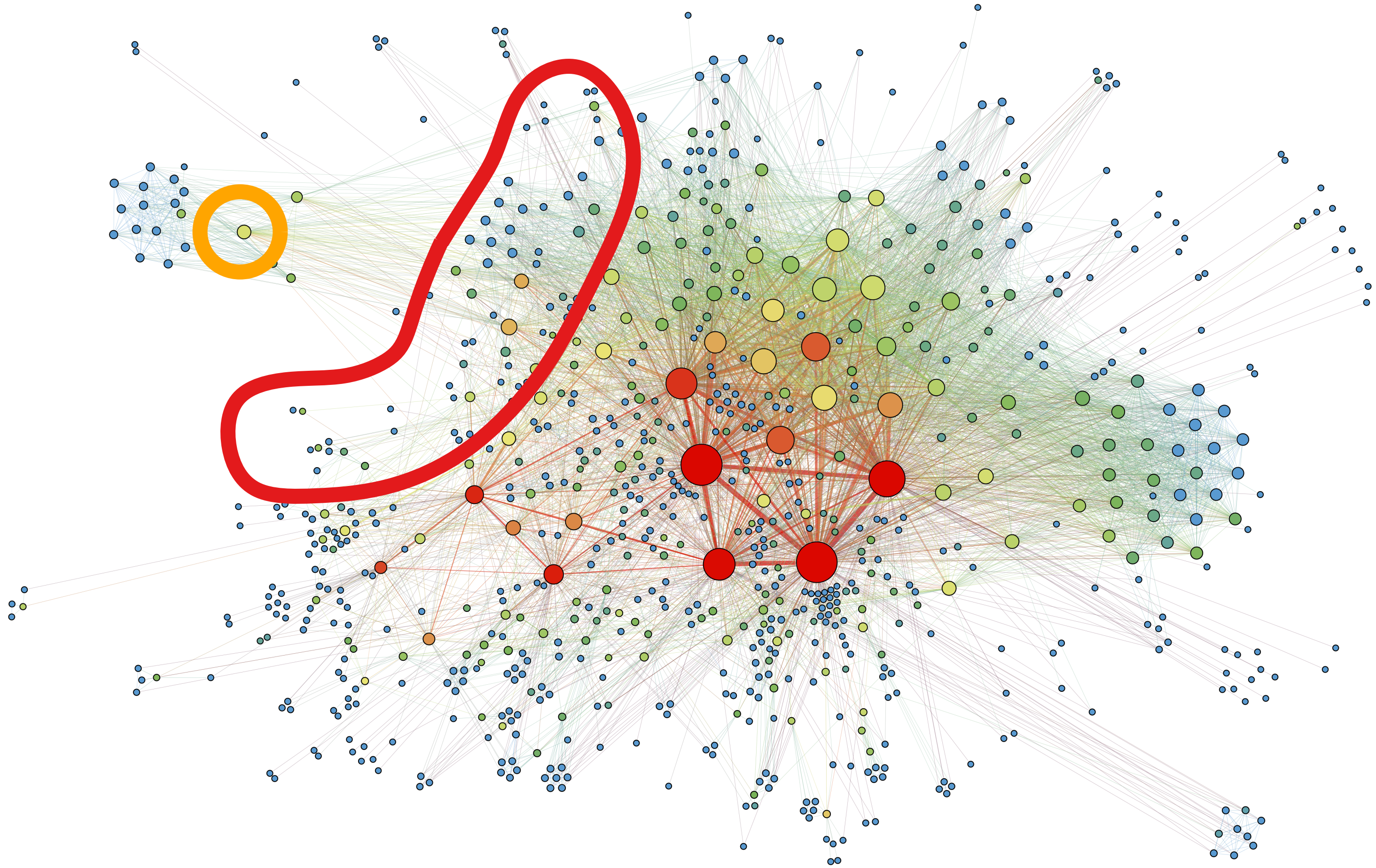
Speed-up with Quad-Tree

Main idea : when computing repulsive force for a vertex *v*, for groups of vertices that are far apart from *v* we do not need to account on individual force-influences



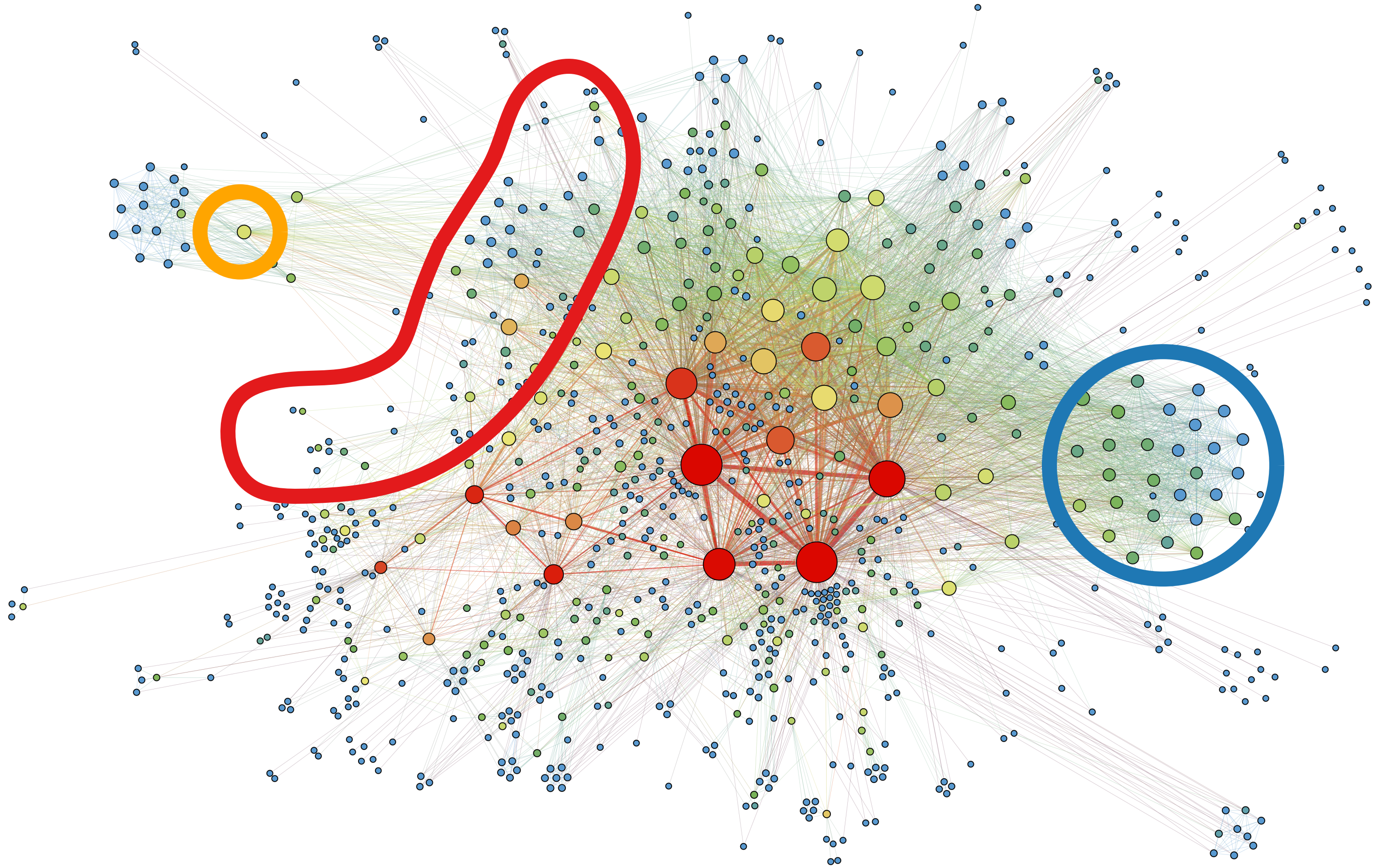
Speed-up with Quad-Tree

Main idea : when computing repulsive force for a vertex *v*, for groups of vertices that are far apart from *v* we do not need to account on individual force-influences



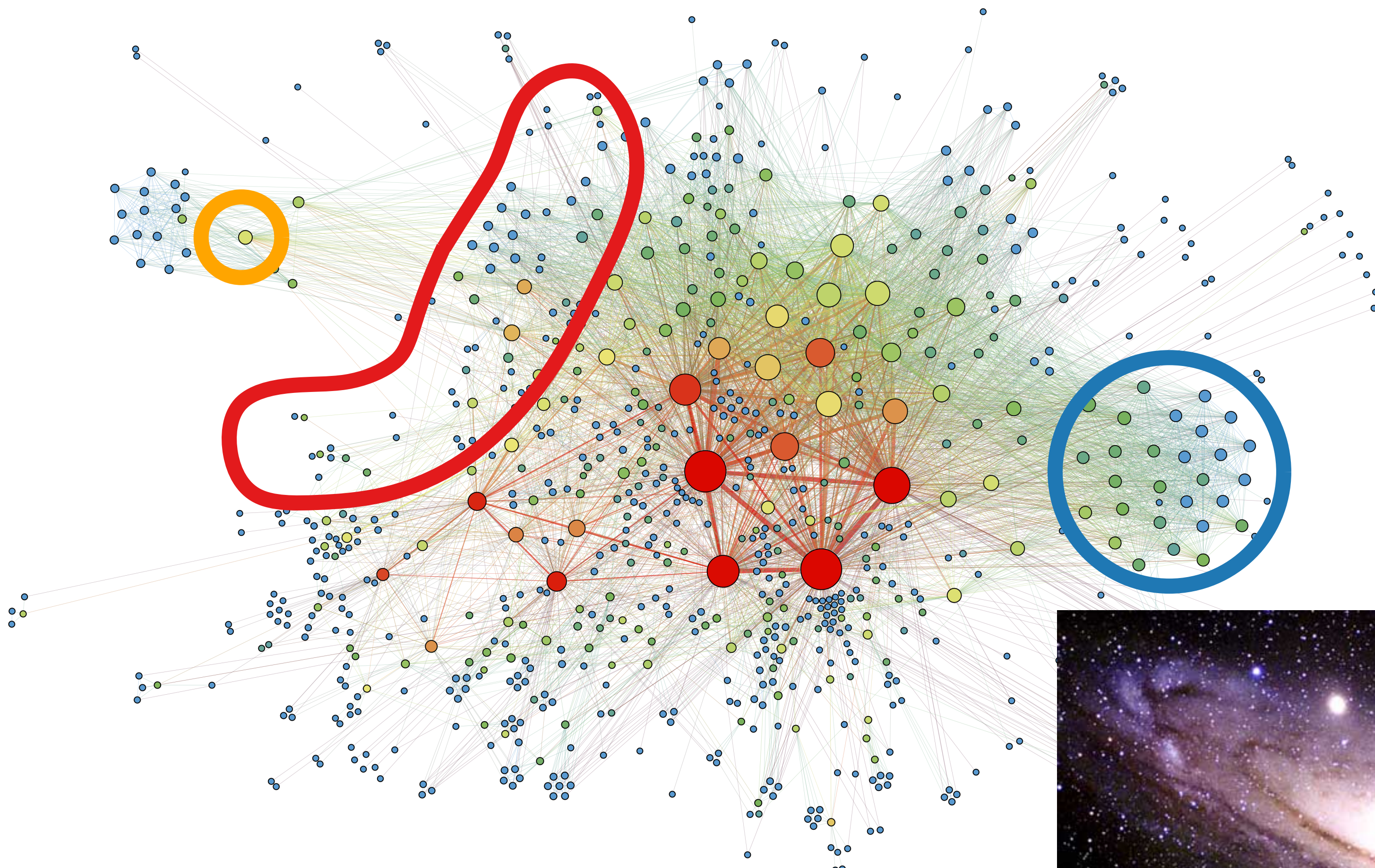
Speed-up with Quad-Tree

Main idea : when computing repulsive force for a vertex *v*, for groups of vertices that are far apart from *v* we do not need to account on individual force-influences

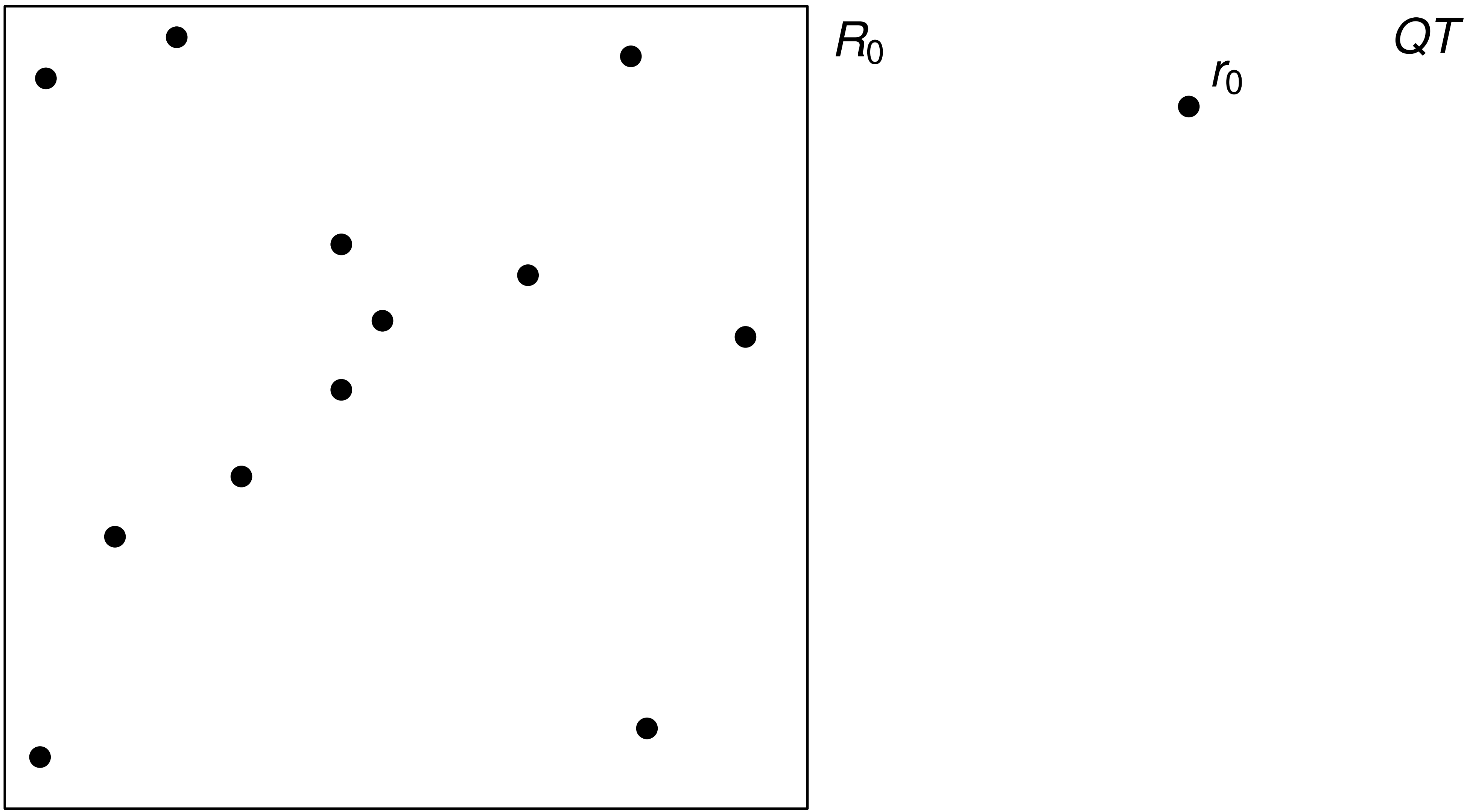


Speed-up with Quad-Tree

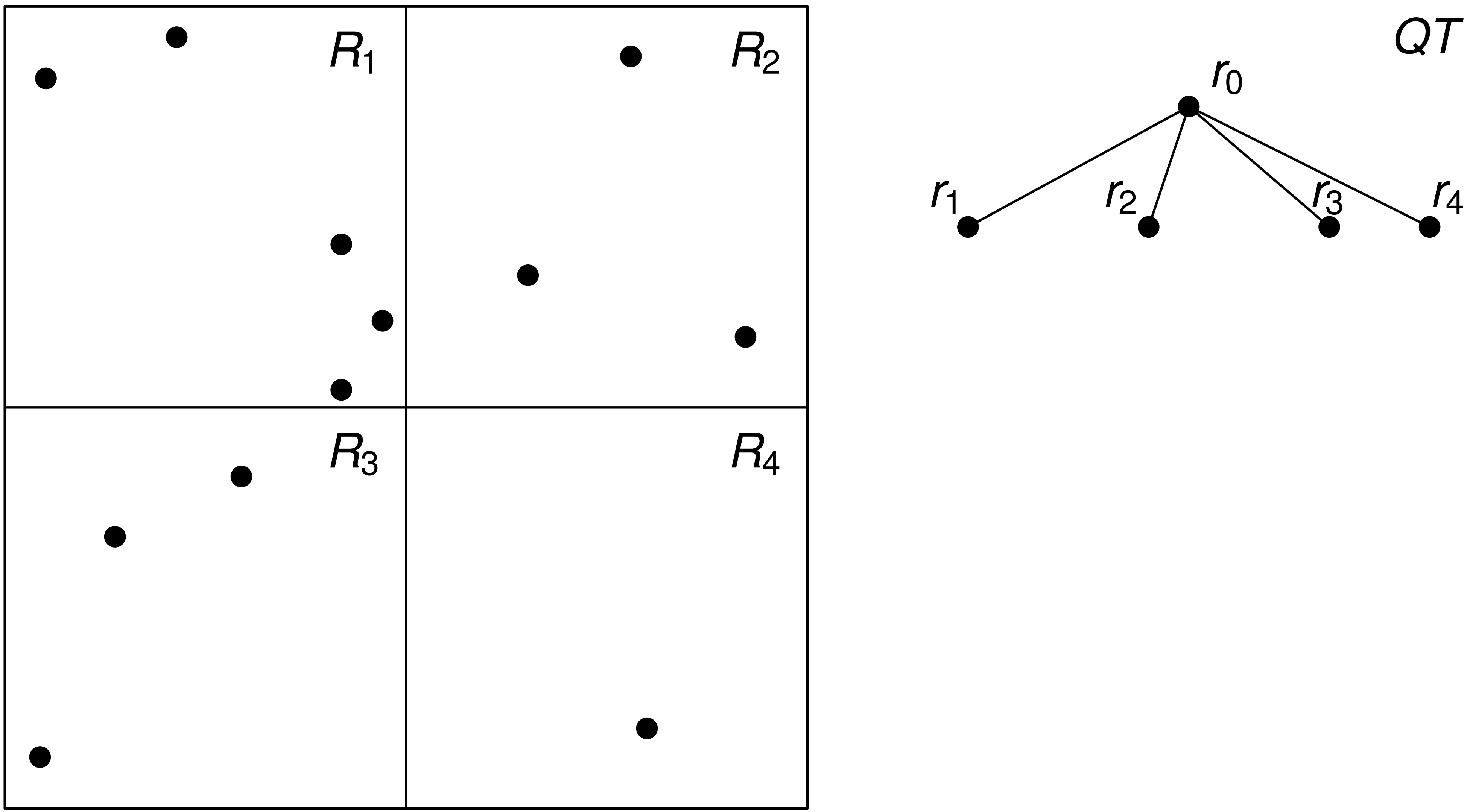
Main idea : when computing repulsive force for a vertex *v*, for groups of vertices that are far apart from *v* we do not need to account on individual force-influences



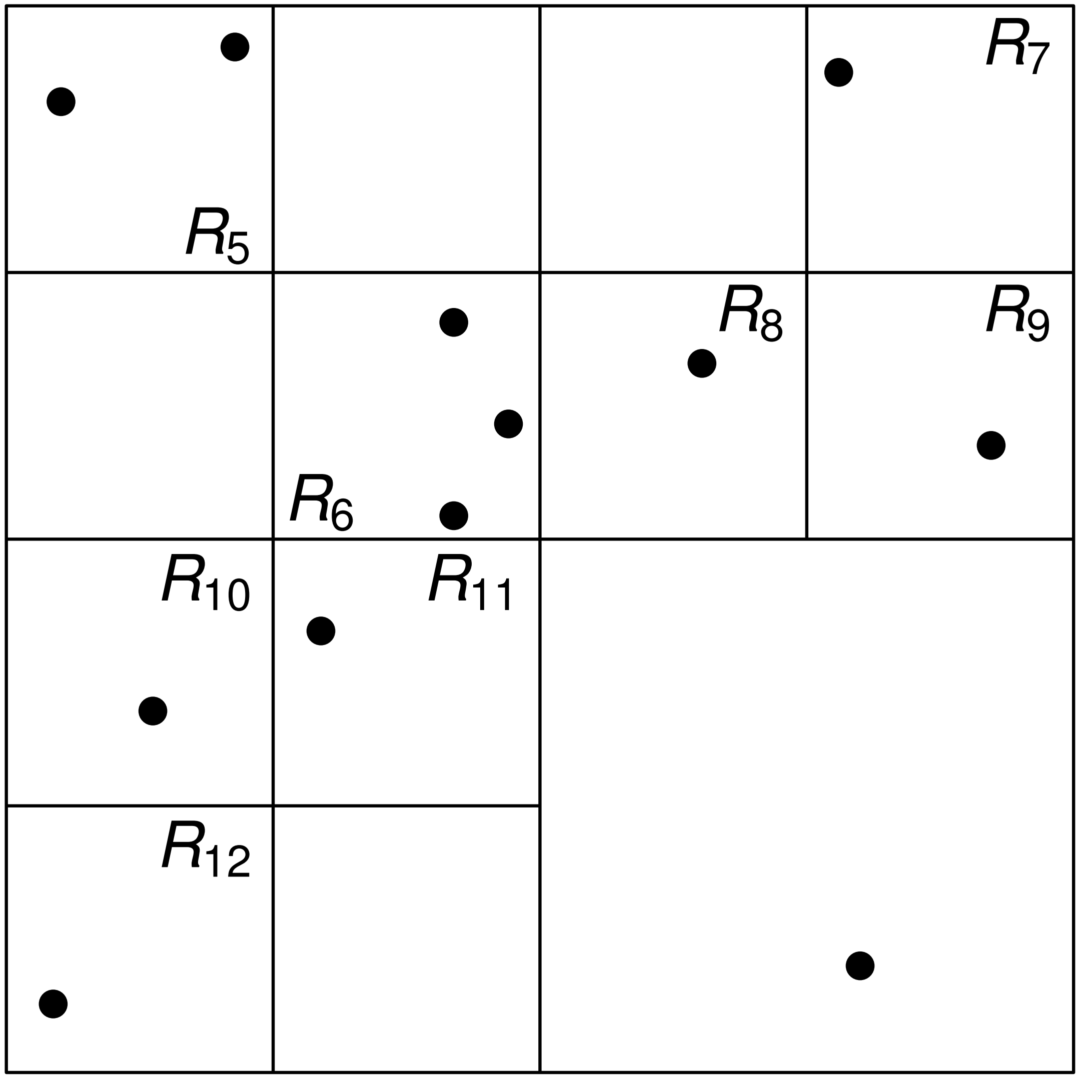
Quad-Tree

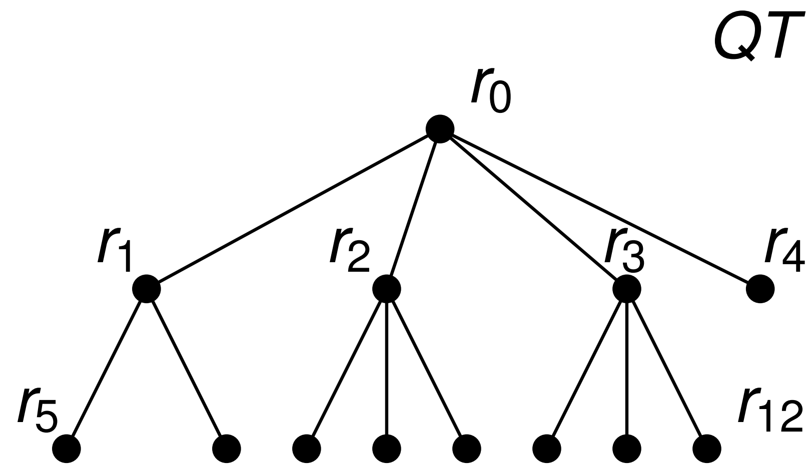


Quad-Tree

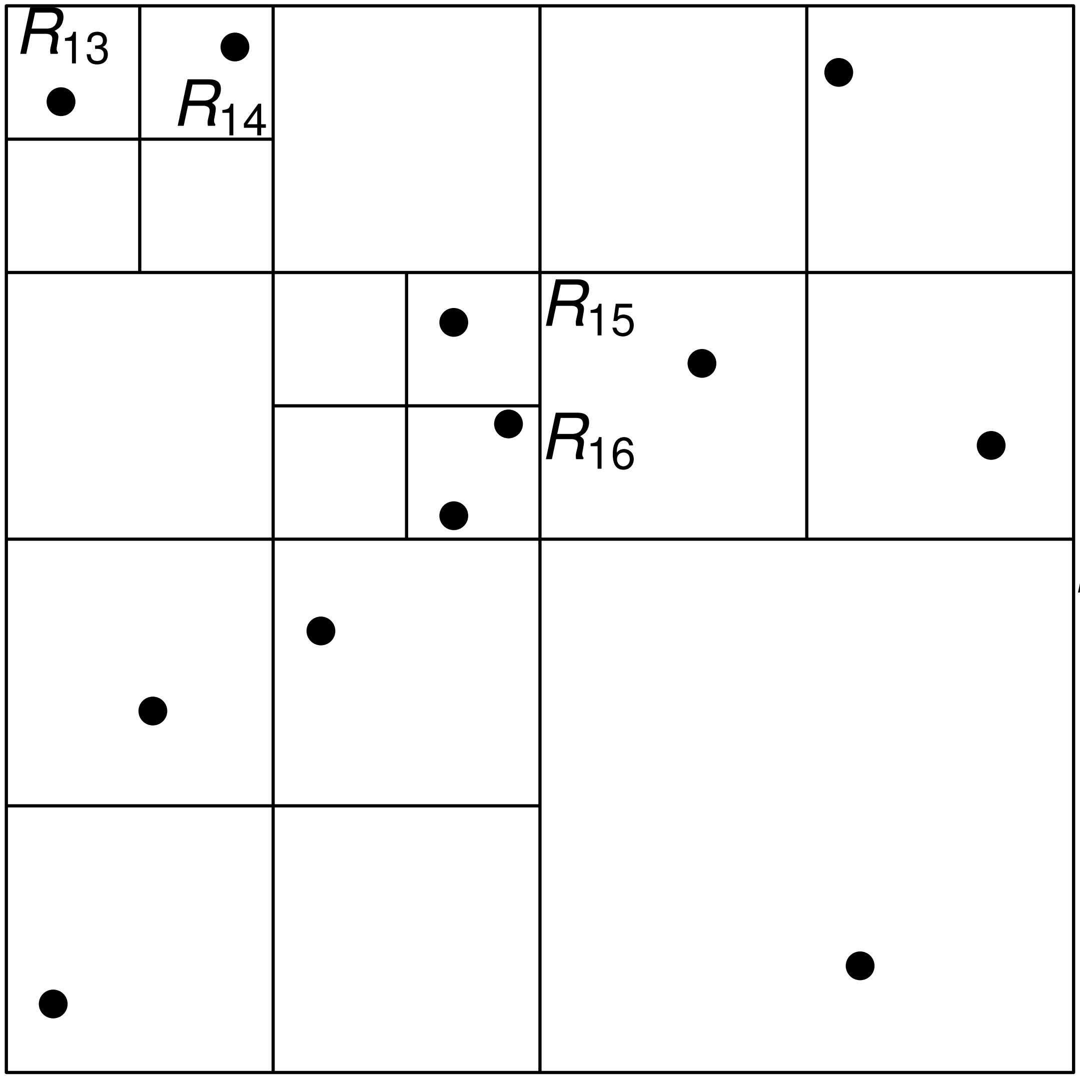


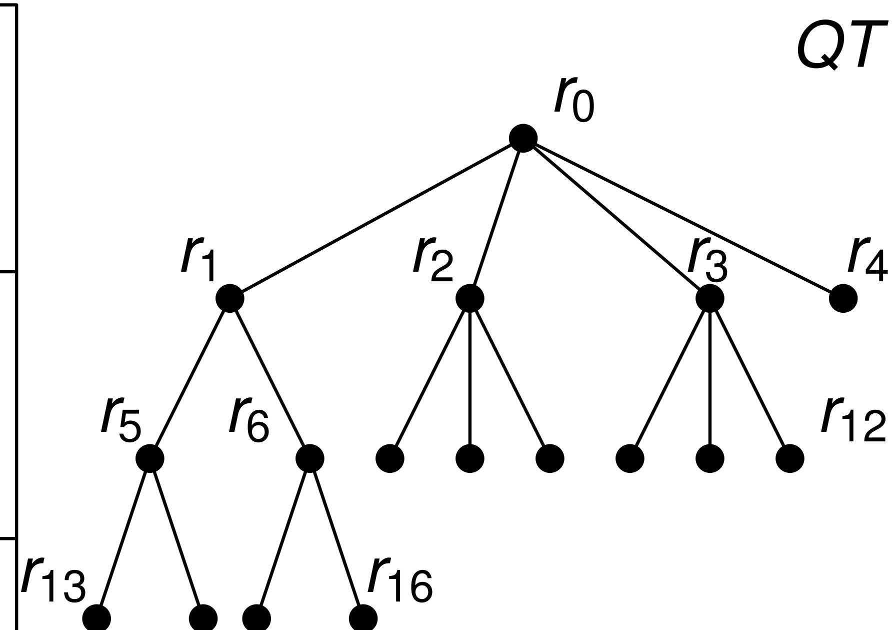
Quad-Tree



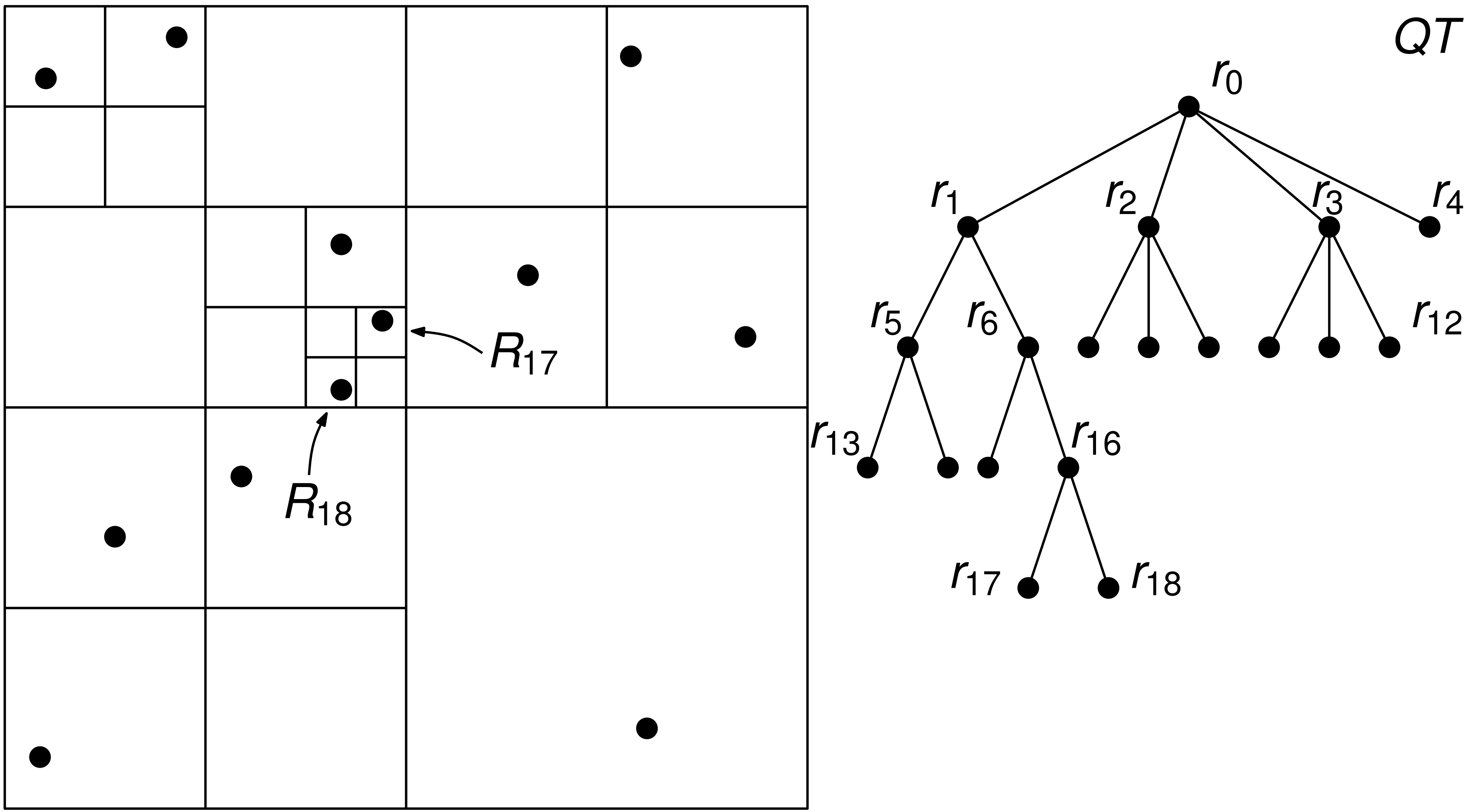


Quad-Tree



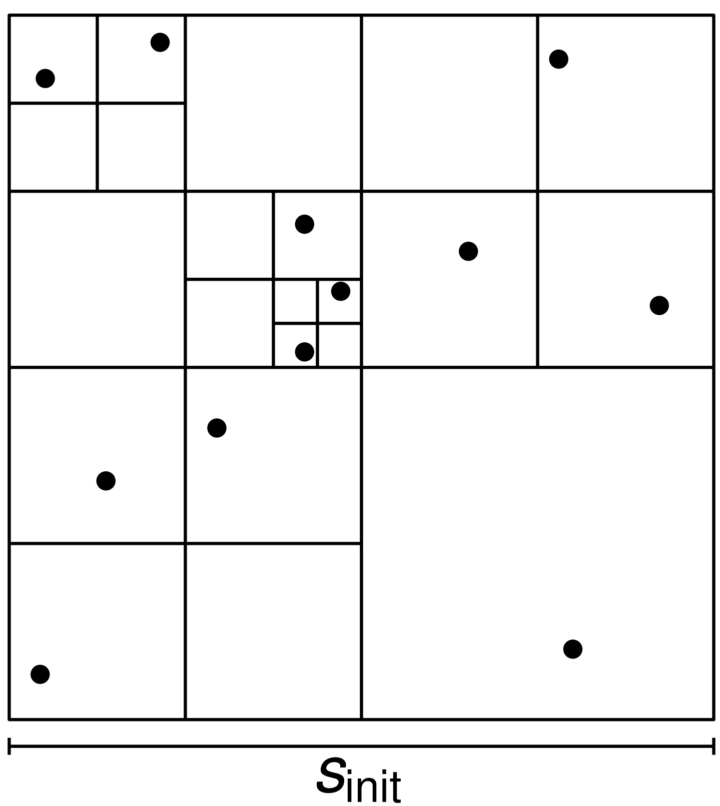


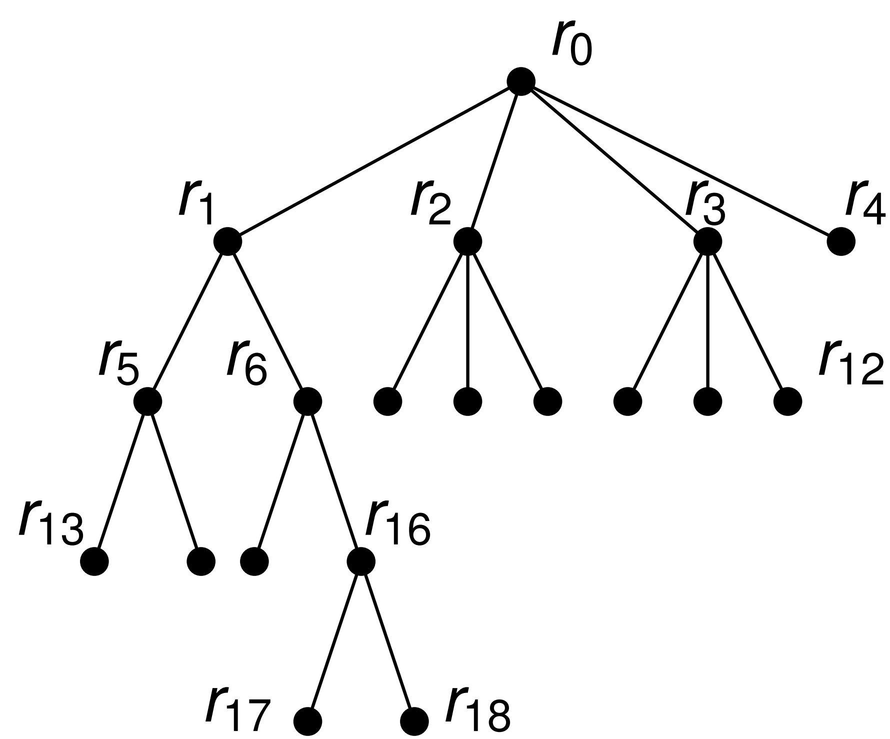
Quad-Tree



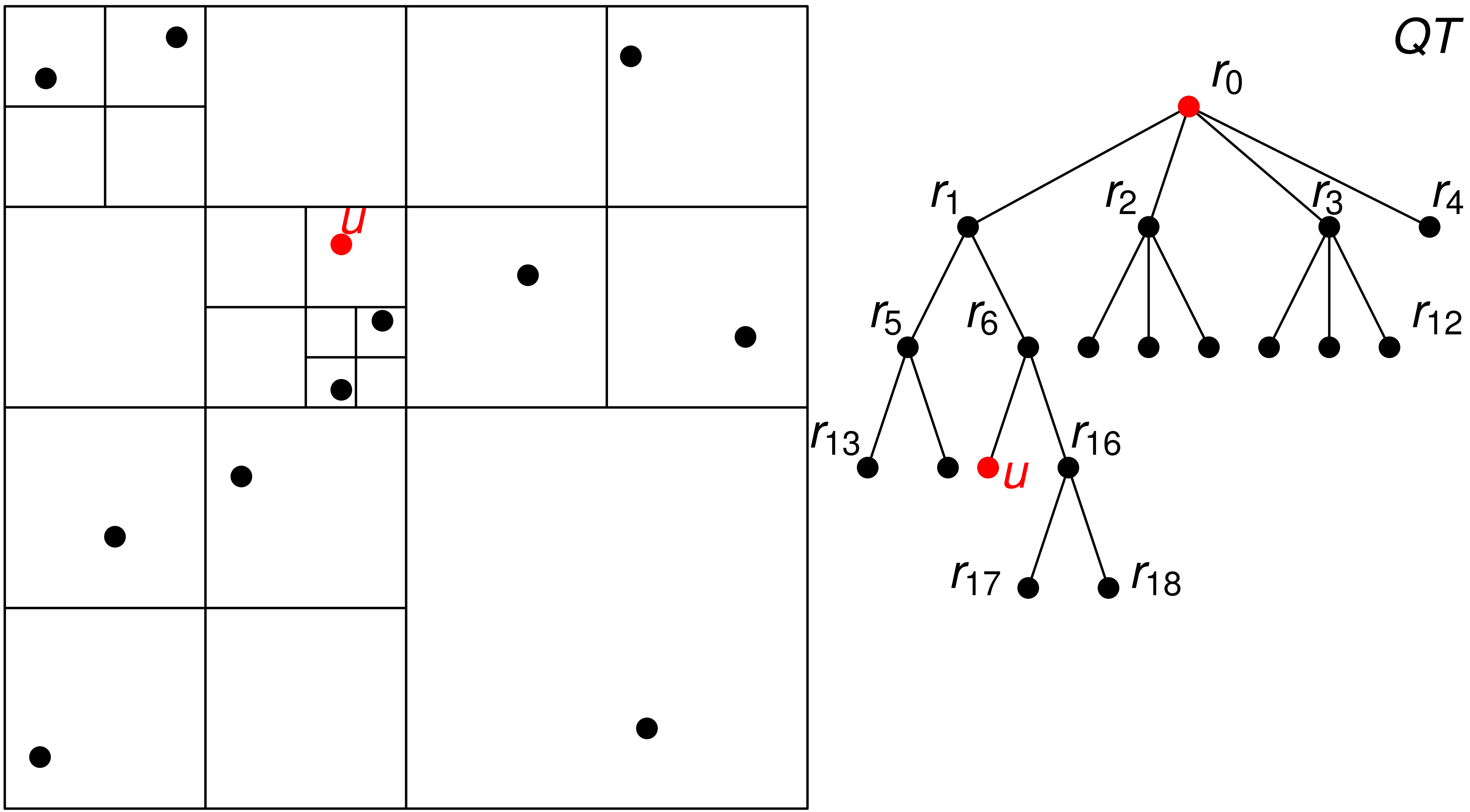
Properties of Quad-Tree

- height $h \le \log \frac{s_{\text{init}}}{d_{\min}} + \frac{3}{2}$, here d_{min}-smallest distance
- time and space $O(hn)$
- *compressed* quad-tree in $O(n \log n)$ time

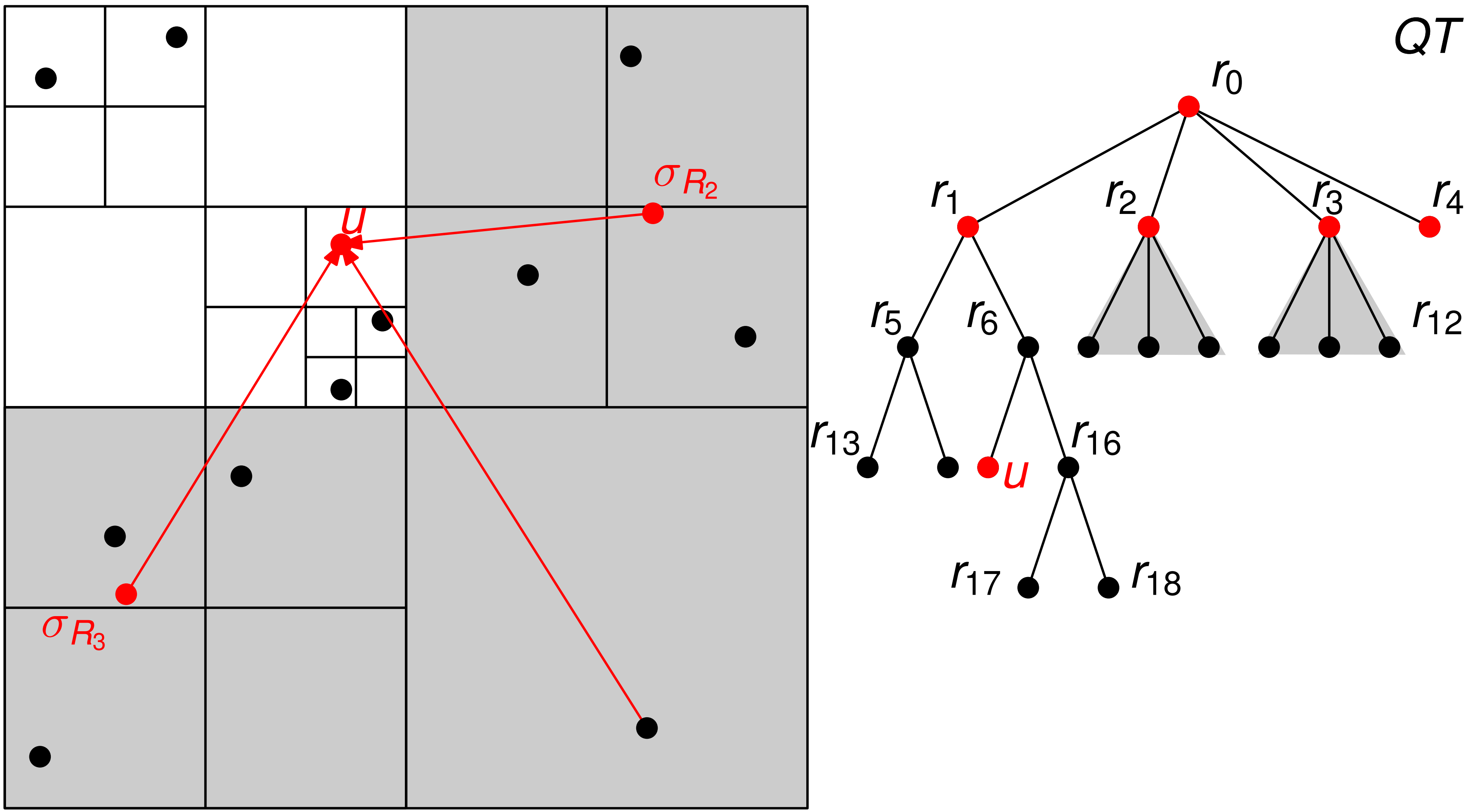




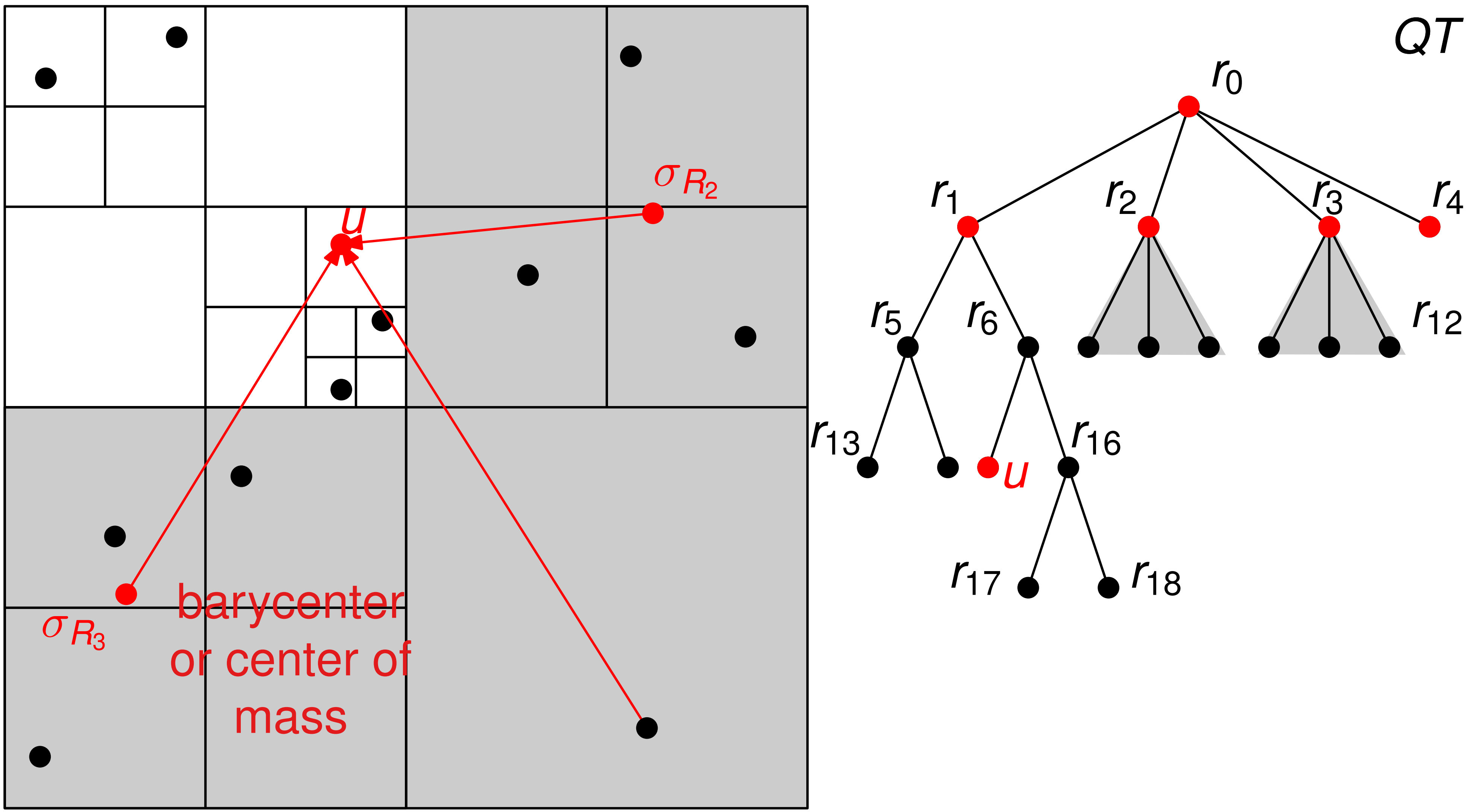
Forces with Quad-Trees (Barnes, Hut, 1986)



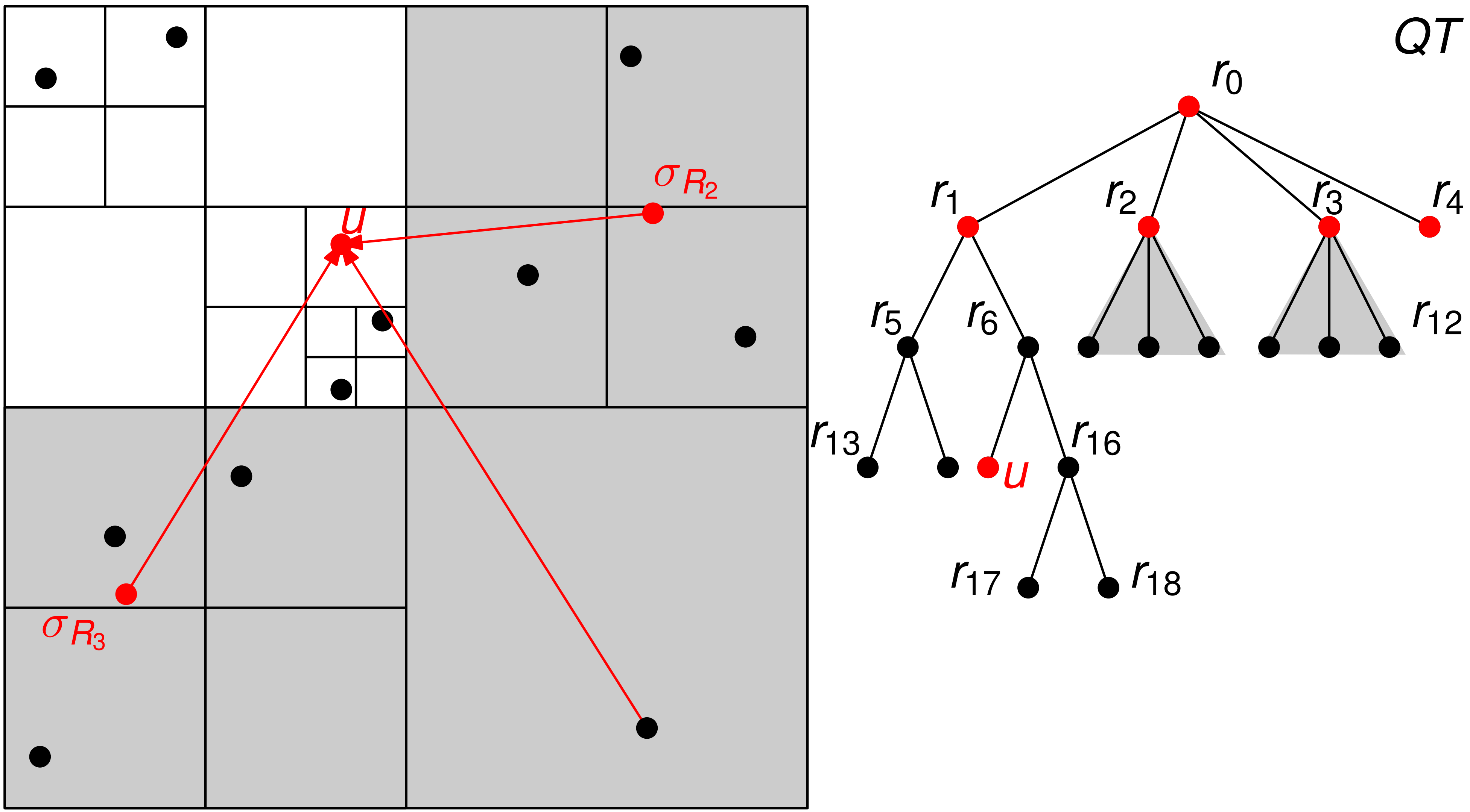
Forces with Quad-Trees (Barnes, Hut, 1986)



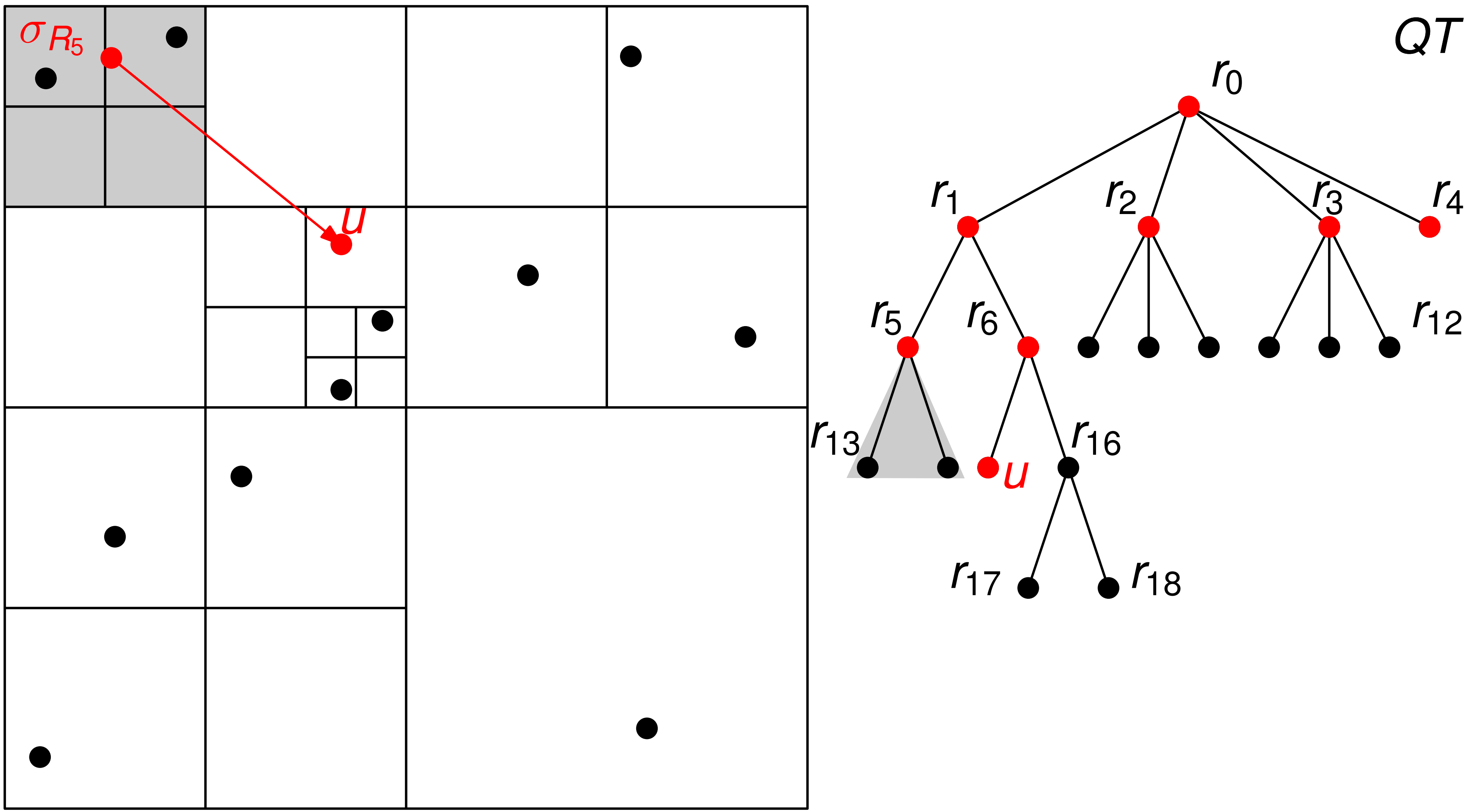
Forces with Quad-Trees (Barnes, Hut, 1986)



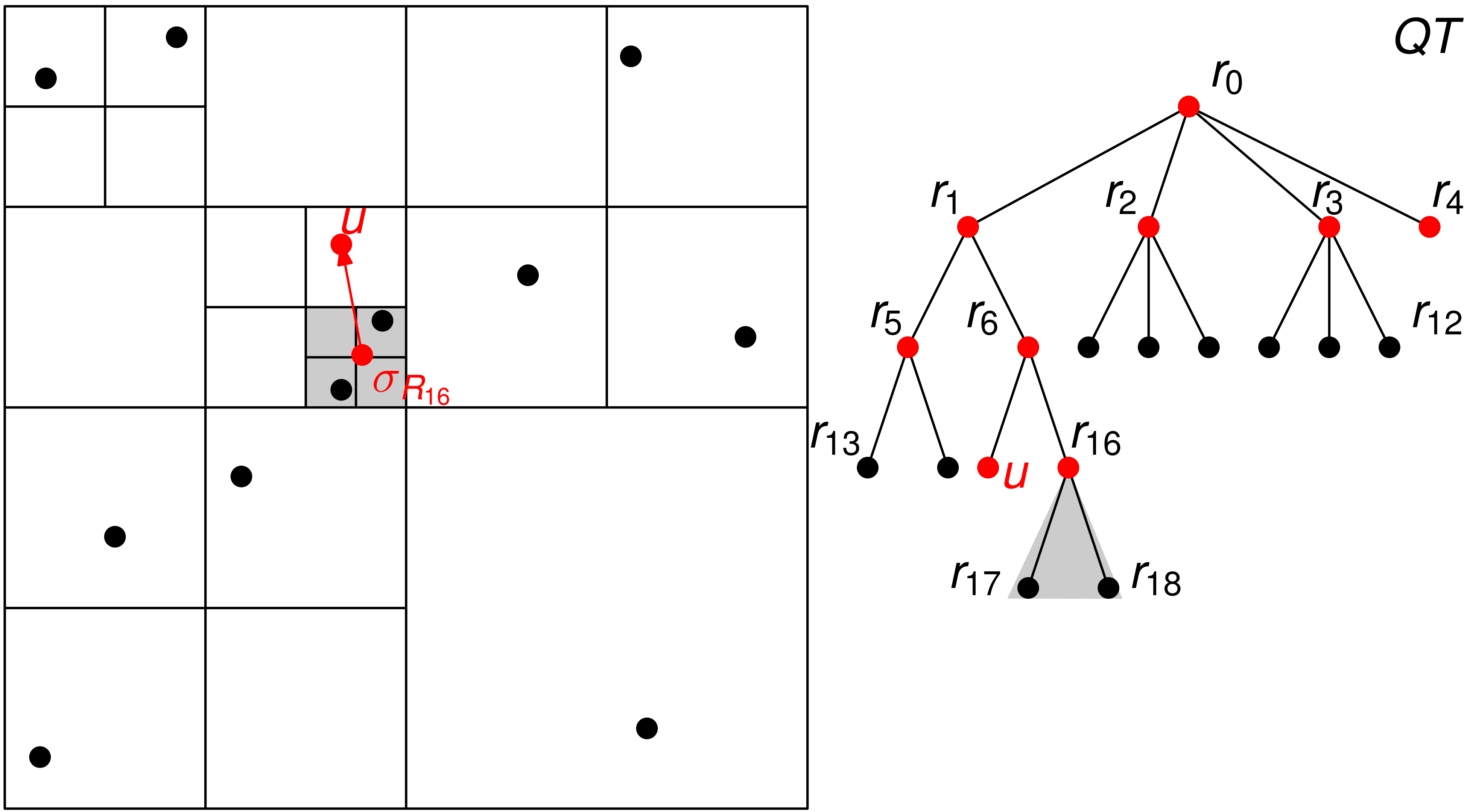
Forces with Quad-Trees (Barnes, Hut, 1986)



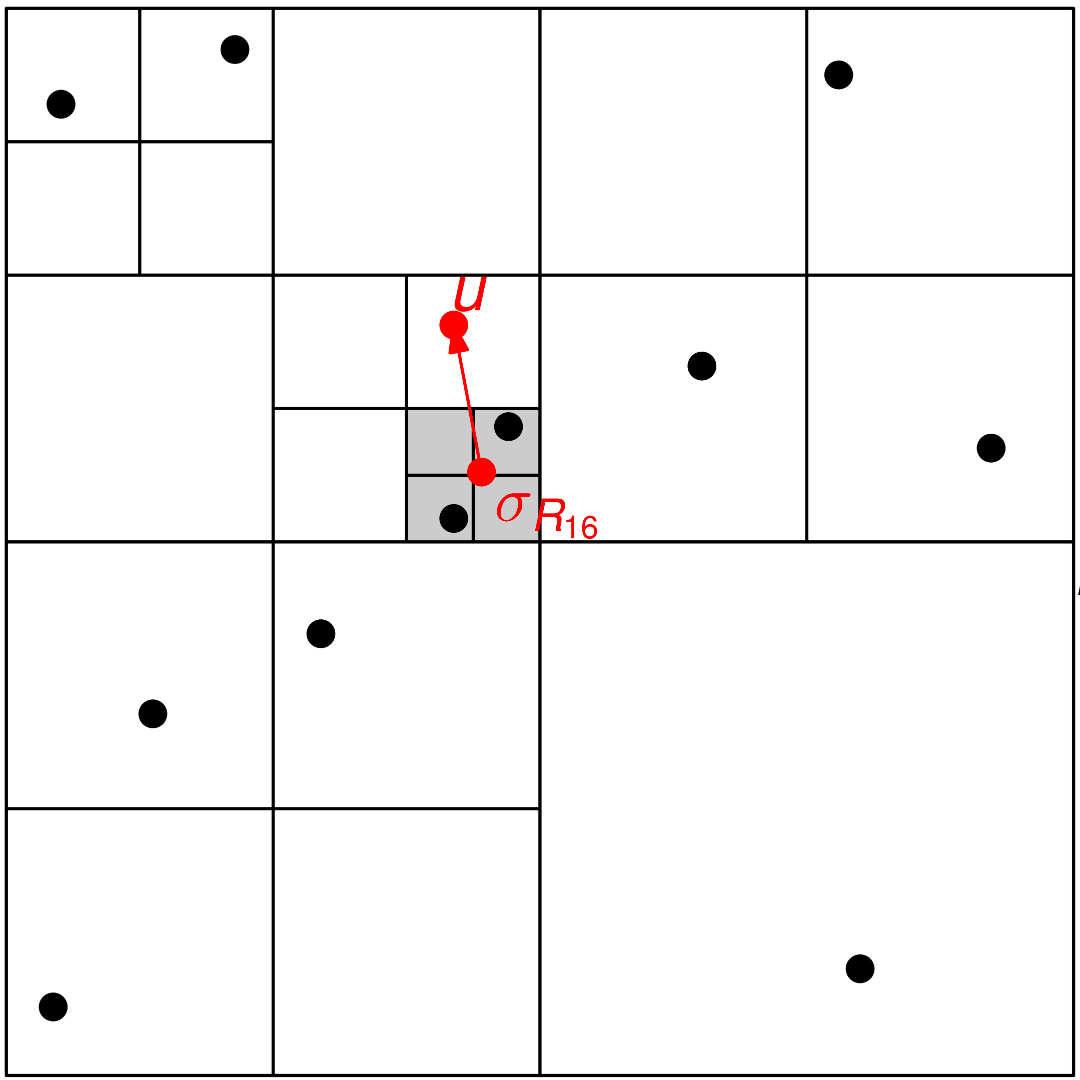
Forces with Quad-Trees (Barnes, Hut, 1986)

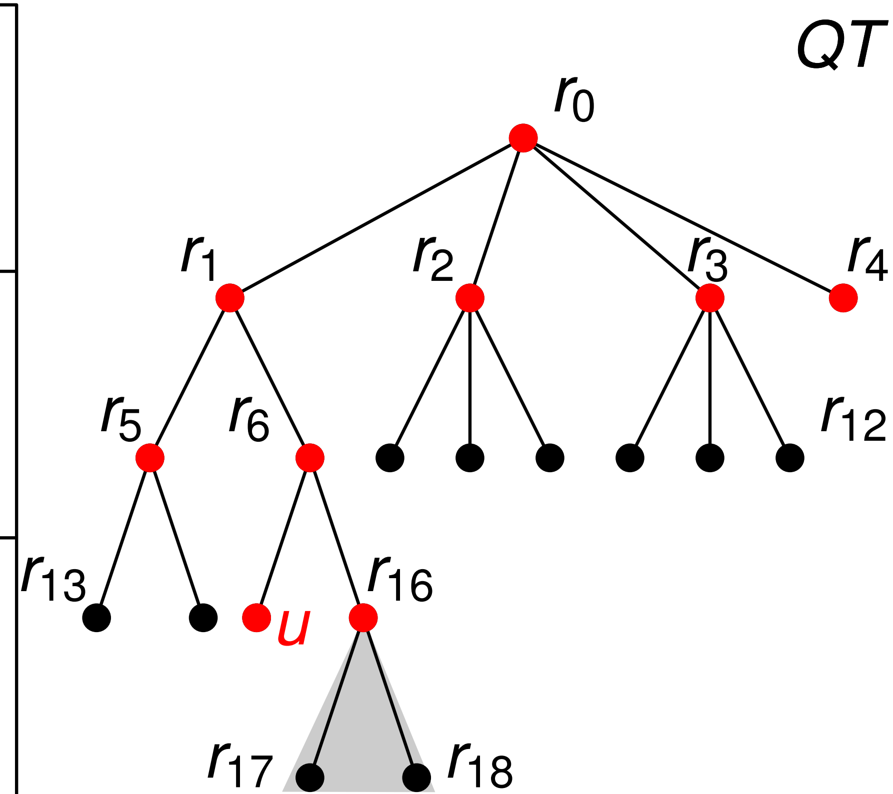


Forces with Quad-Trees (Barnes, Hut, 1986)



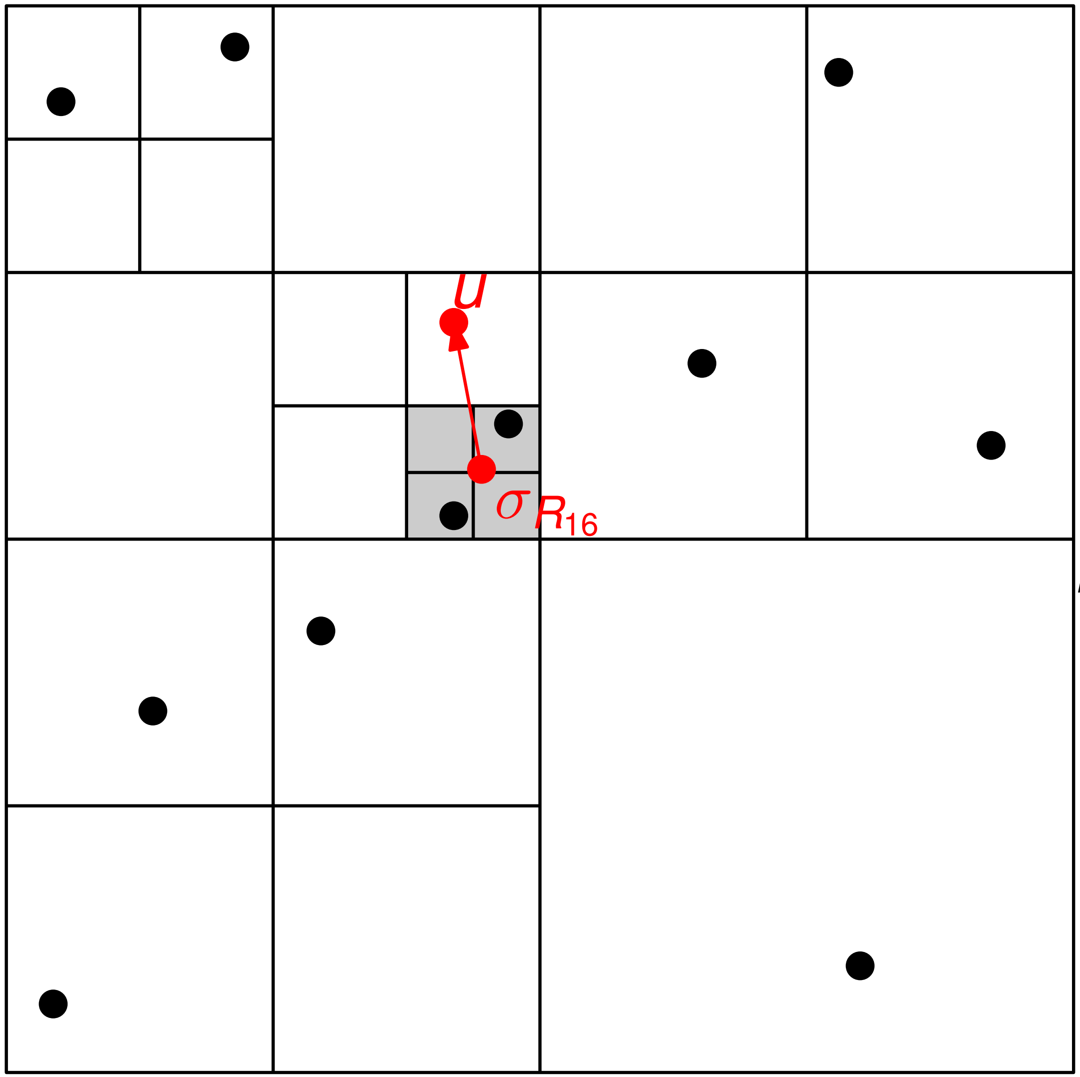
Forces with Quad-Trees (Barnes, Hut, 1986)

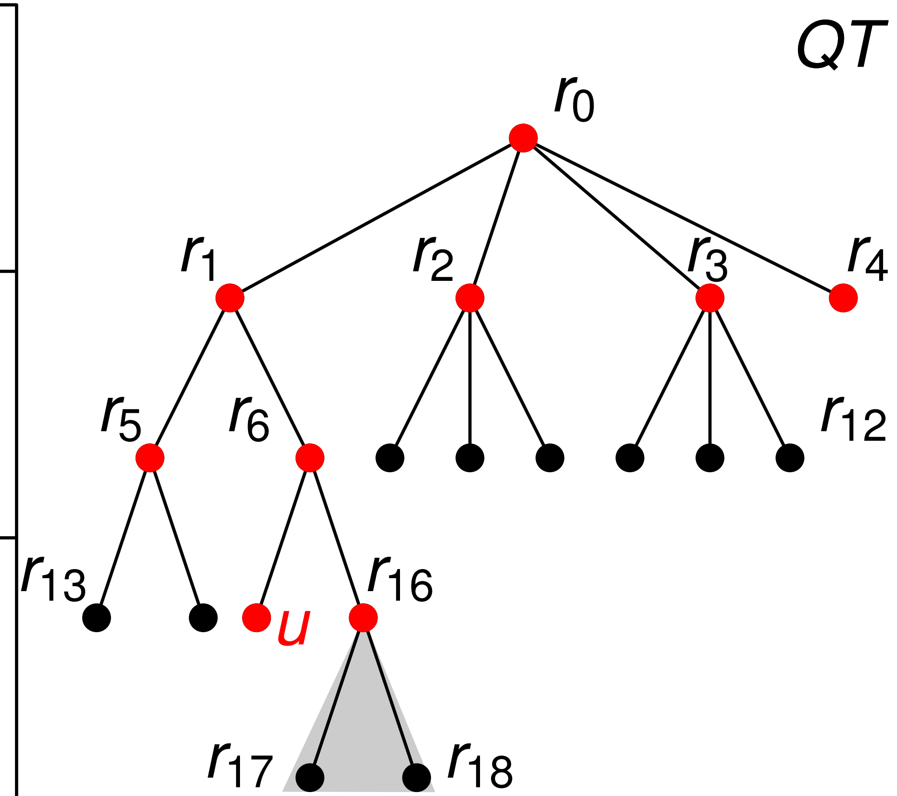




When to use center of mass and when real points?

Forces with Quad-Trees (Barnes, Hut, 1986)

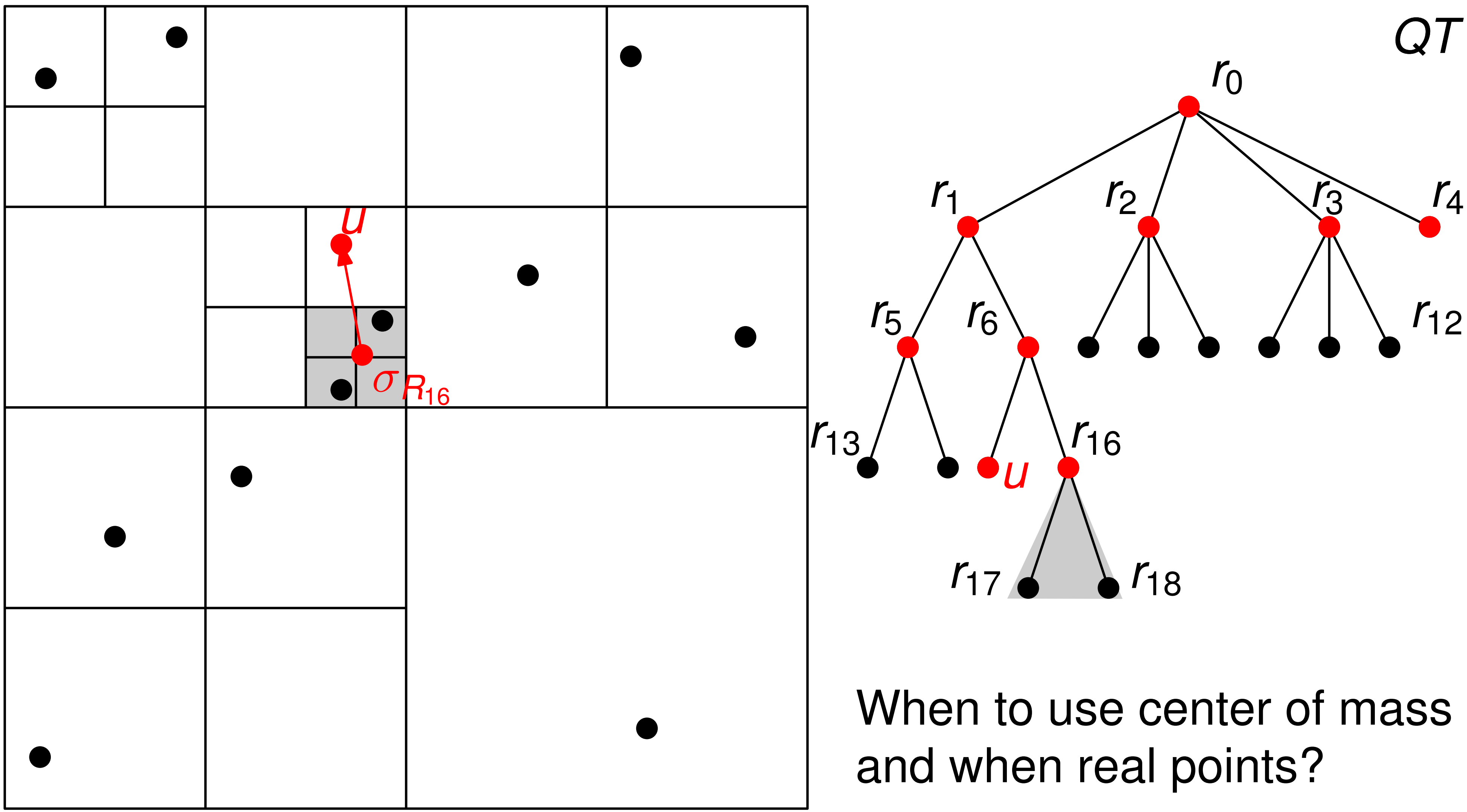




When to use center of mass and when real points?

Parameter θ: if $\frac{\text{width of the box}}{\text{distance to the center}} < \theta$ - use center of mass

Forces with Quad-Trees (Barnes, Hut, 1986)

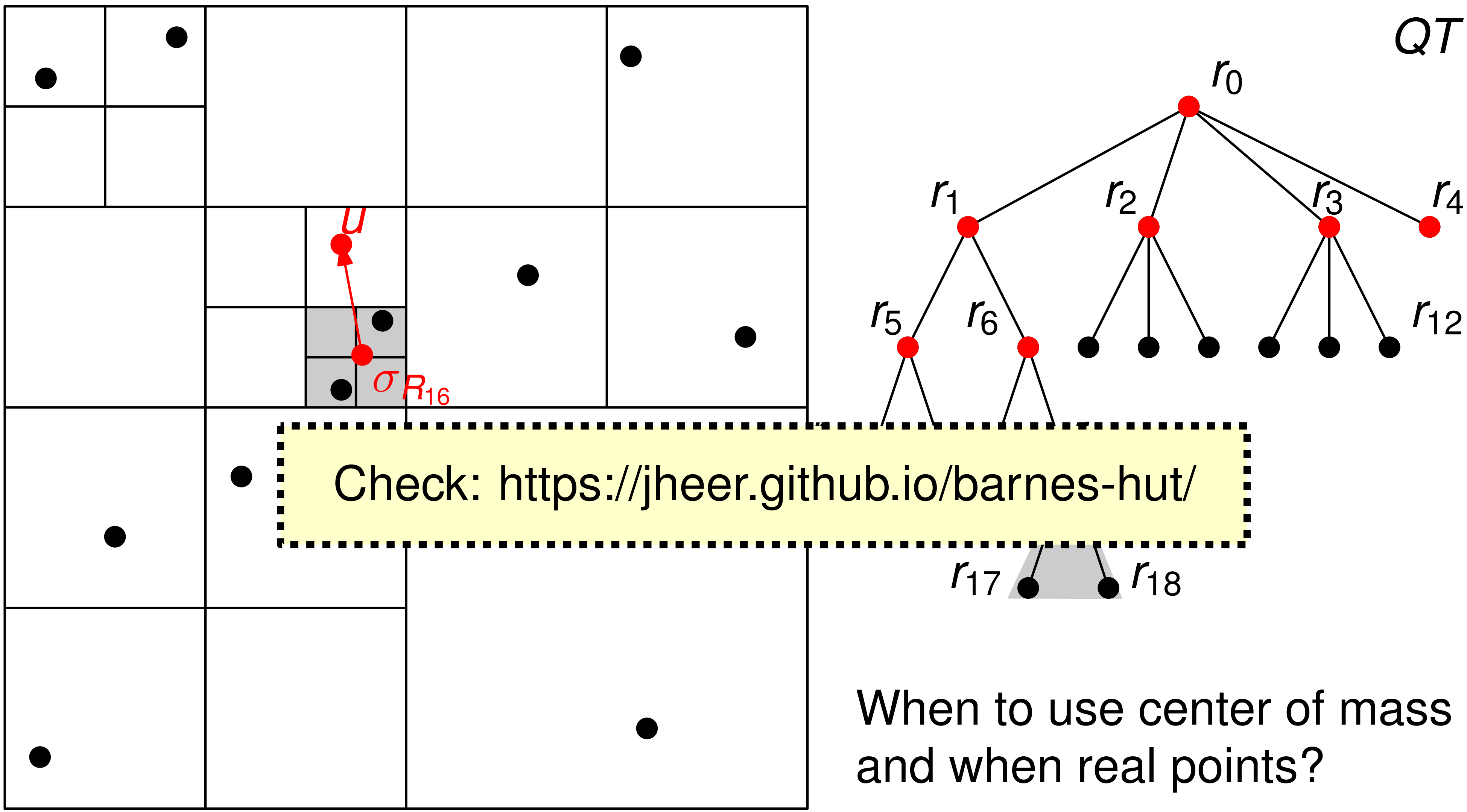


When to use center of mass and when real points?

Parameter θ: if $\frac{\text{width of the box}}{\text{distance to the center}} < \theta$ - use center of mass

Assuming homogeneous distribution, the calculations of forces can be done in $O(\log n)$ for a single vertex - $O(n \log n)$ overall vs $O(n^2)$

Forces with Quad-Trees (Barnes, Hut, 1986)



Check: https://jheer.github.io/barnes-hut/

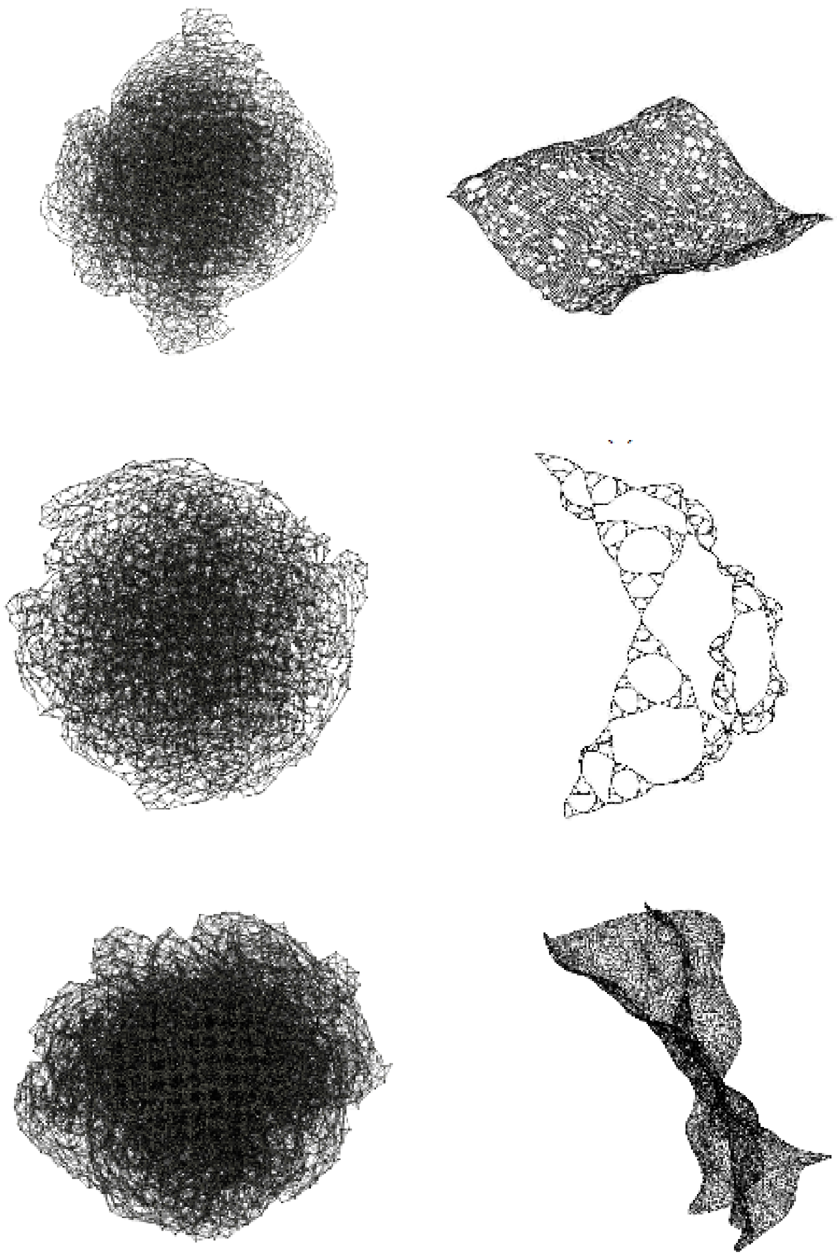
When to use center of mass and when real points?

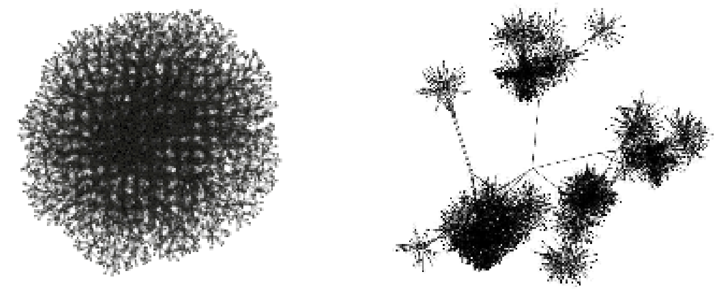
Parameter θ: if $\frac{\text{width of the box}}{\text{distance to the center}} < \theta$ - use center of mass

Assuming homogeneous distribution, the calculations of forces can be done in $O(\log n)$ for a single vertex - $O(n \log n)$ overall vs $O(n^2)$

Modifications

GRIP (Hachul, Jünger 2007)



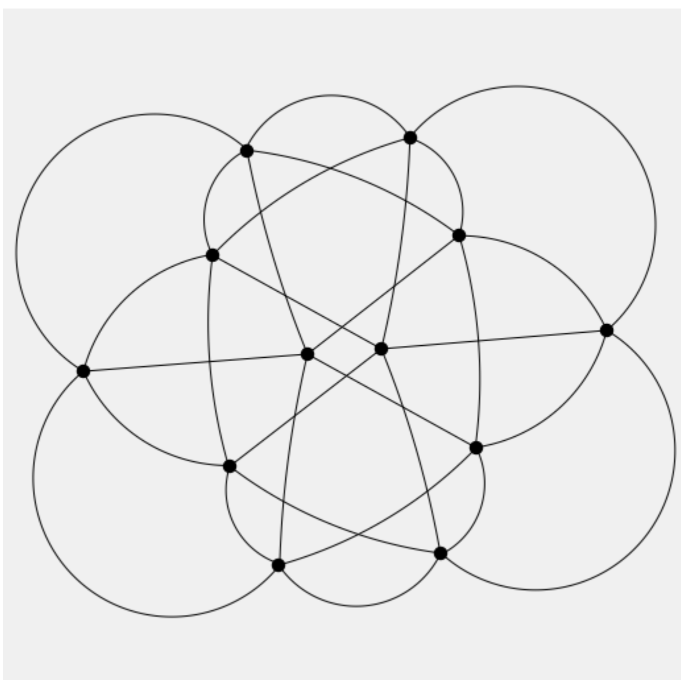


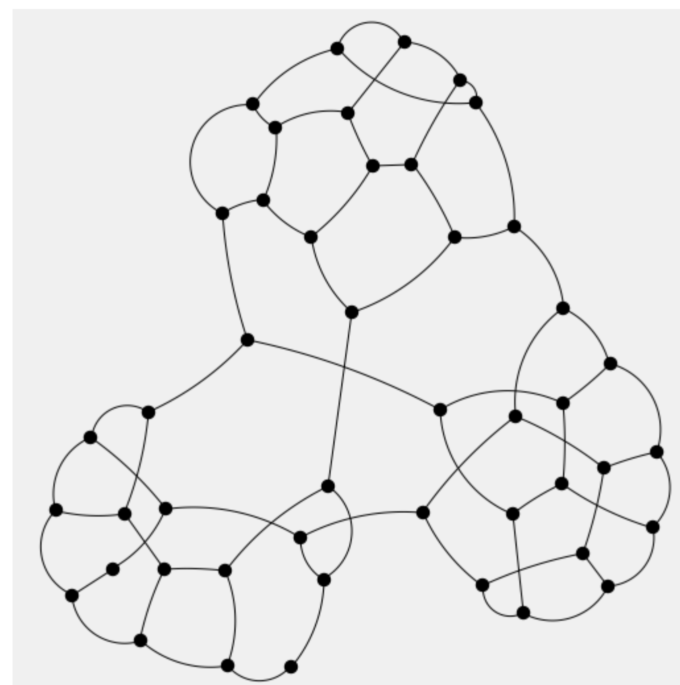
Left: Grid version of Fruchterman Reingold.
Right: GRIP
Idea: Construct layers of nodes, position layers one after the other,
force-directed shaking locally

Modifications

Lombardi-Spring-Embedder **(Chernobelskiy et al. 2012)**

- edges are circular arcs
- goal: optimal angular resolution $2\pi/\deg(v)$ at each node v
- additional rotational forces





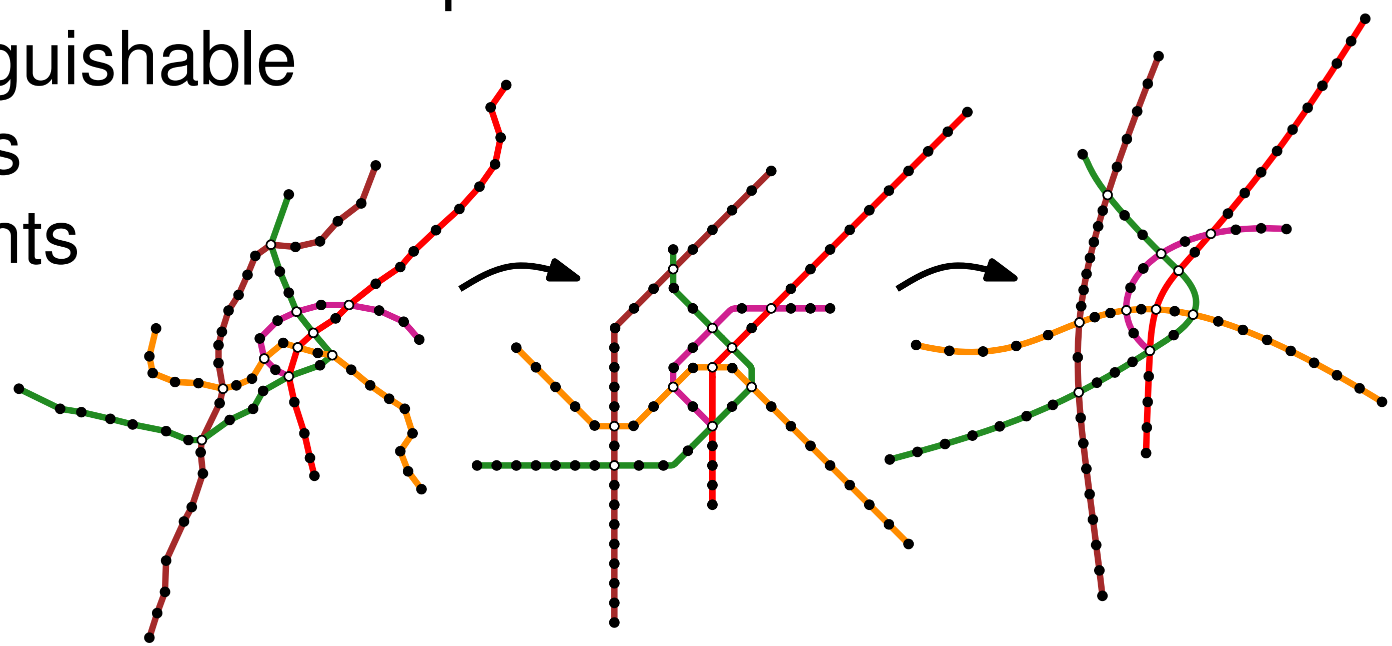
Modifications

Lombardi-Spring-Embedder (Chernobelskiy et al. 2012)

- edges are circular arcs
- goal: optimal angular resolution $2\pi / \deg(v)$ at each node v
- additional rotational forces

Metro Maps with Bézier curves (Fink et al. 2013)

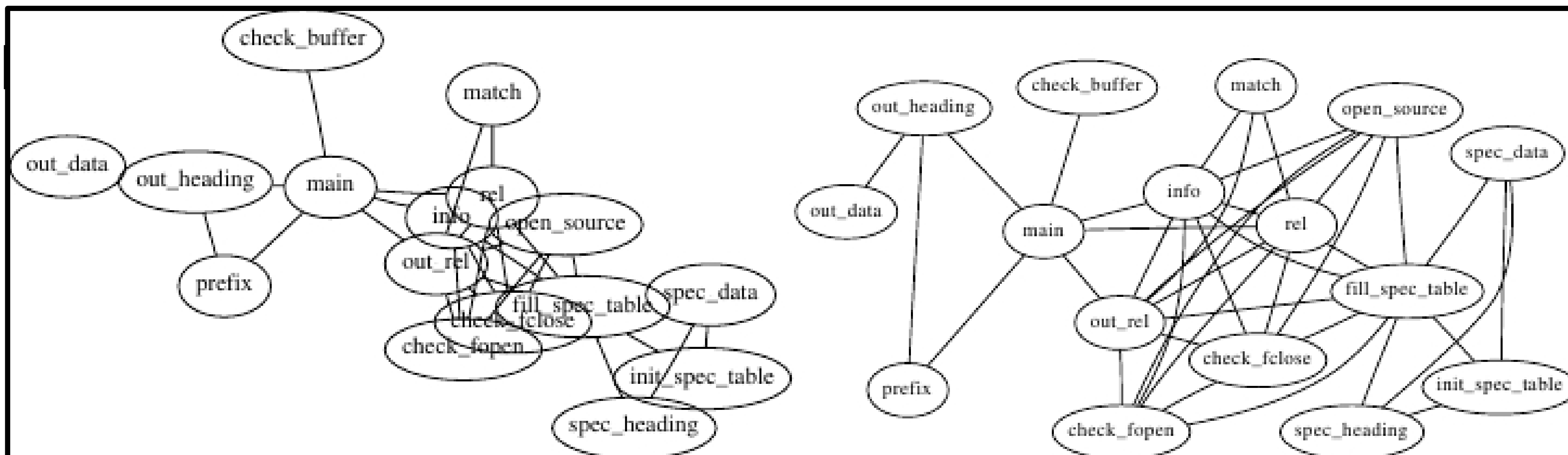
- model paths as Bézier curves
- forces on nodes and control points:
- lines are distinguishable
- few bend points
- few control points



Modifications

Lombardi-Spring-Embedder (Chernobelskiy et al. 2012)

- edges are circular arcs
- goal: optimal angular resolution $2\pi/\deg(v)$ at each node v
- additional rotational forces



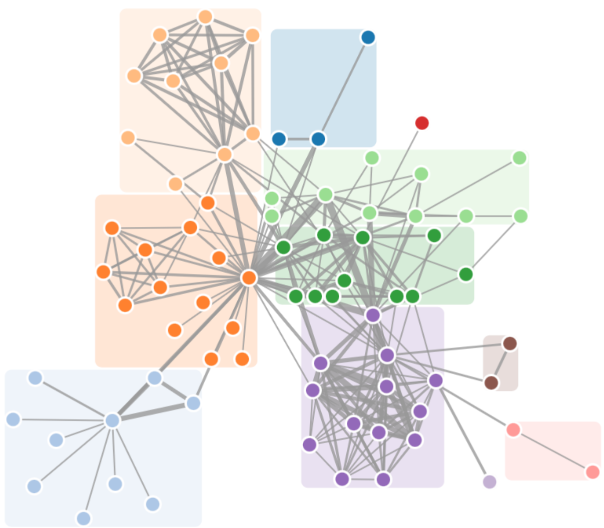
Realistic Node Sizes (Gansner, North 1998)

- node positions are adjusted to avoid overlaps

Modifications

Separation Constraints (Dwyer, Koren, Marriott, 2006)

- groups of vertices are constrained to lie in predetermined polygons/ other separation contraints

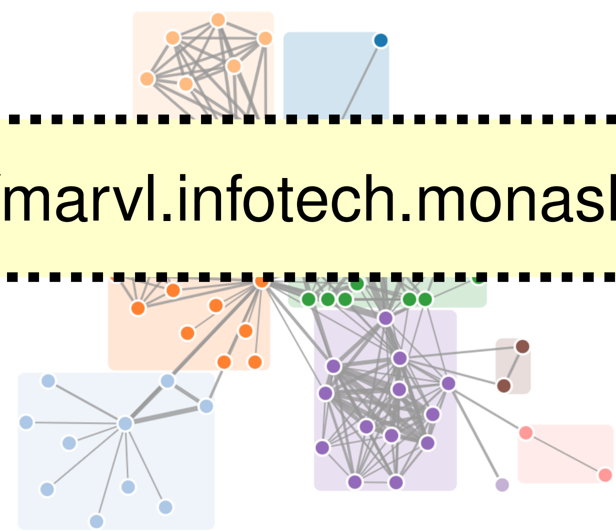


Modifications

Separation Constraints (Dwyer, Koren, Marriott, 2006)

- groups of vertices are constrained to lie in predetermined polygons/ other separation contraints

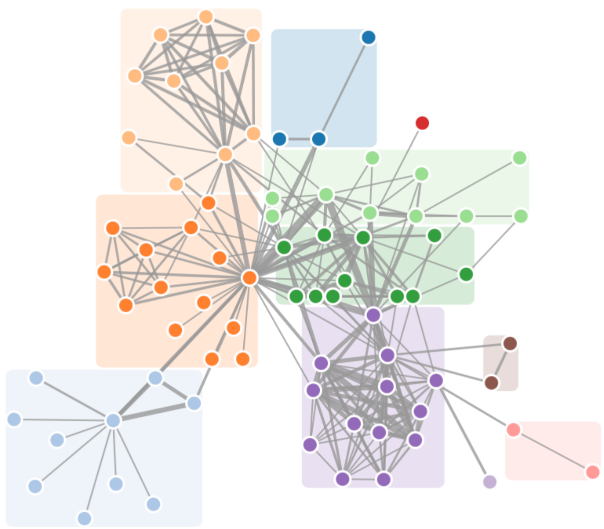
https://marvl.infotech.monash.edu/webcola/



Modifications

Separation Constraints (Dwyer, Koren, Marriott, 2006)

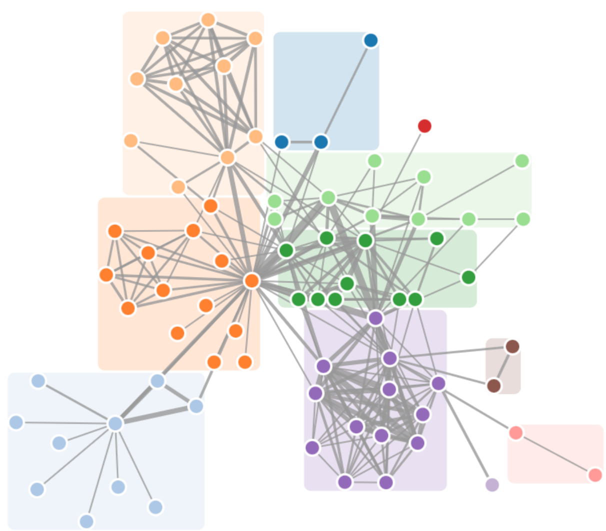
- groups of vertices are constrained to lie in predetermined polygons/ other separation contraints

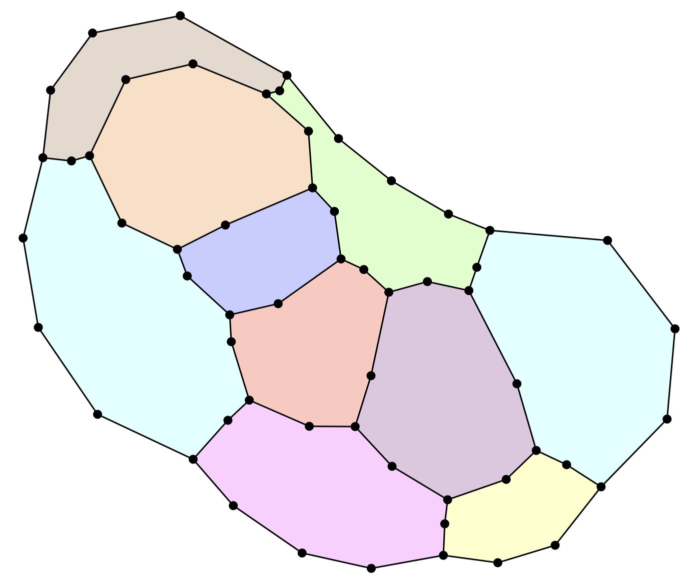


Modifications

Separation Constraints (Dwyer, Koren, Marriott, 2006)

- groups of vertices are constrained to lie in predetermined polygons/ other separation contraints





Dynamic maps (Mcheldidze, Schnorr, 2022)

- regions are proportional to given values
- regions have simple organic form
- regions come and go/change adjacency

Summary

Force-based Approaches are

- easily understandable and implementable
- no special requirements on the input graph
- depending on the graphs (small and sparce) amazingly good layouts (Symmetries, Clustering, ...)
- easily adaptable and configurable
- robust
- scalable

Summary

Force-based Approaches are

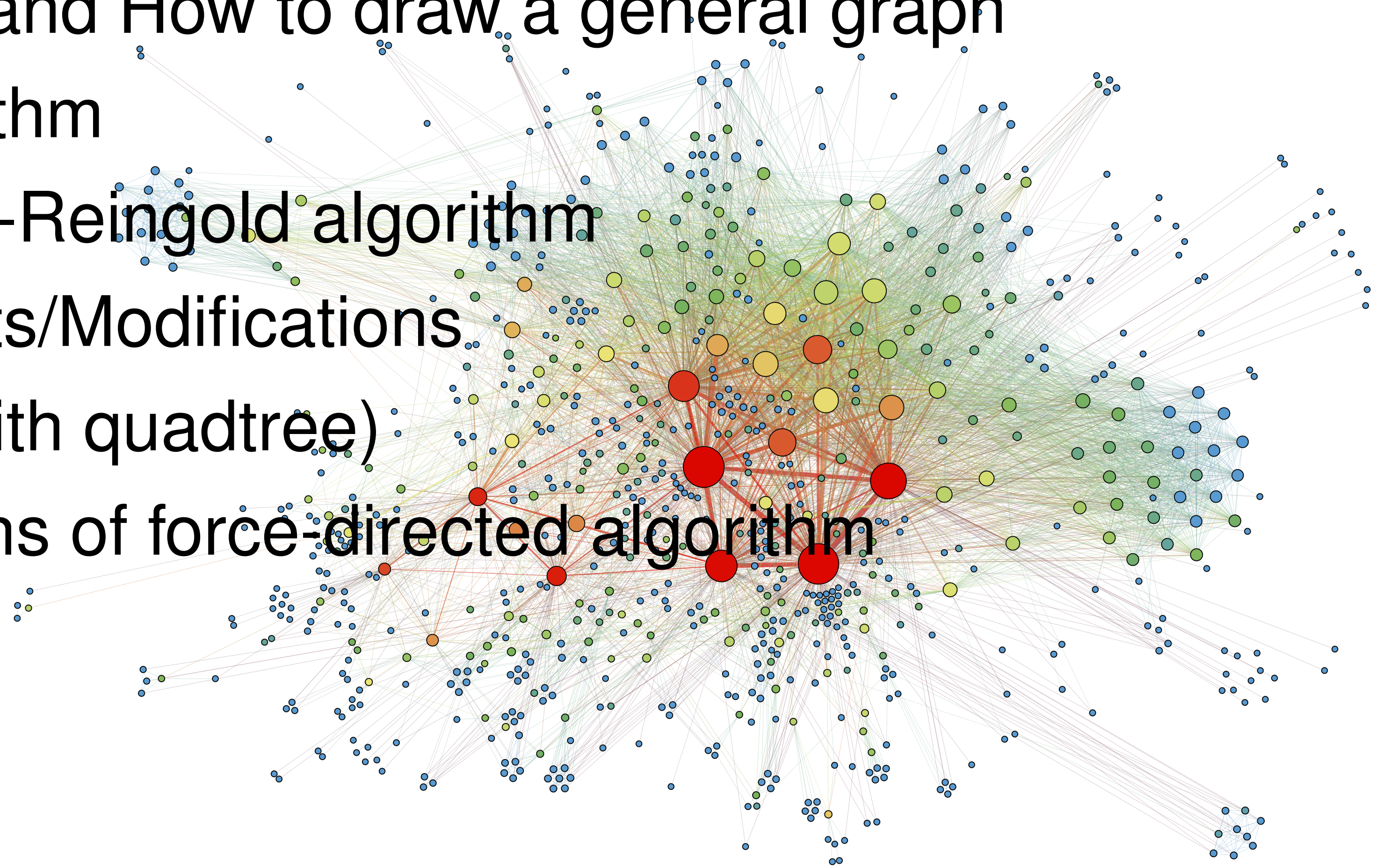
- easily understandable and implementable
- no special requirements on the input graph
- depending on the graphs (small and sparce) amazingly good layouts (Symmetries, Clustering, ...)
- easily adaptable and configurable
- robust
- scalable

But...

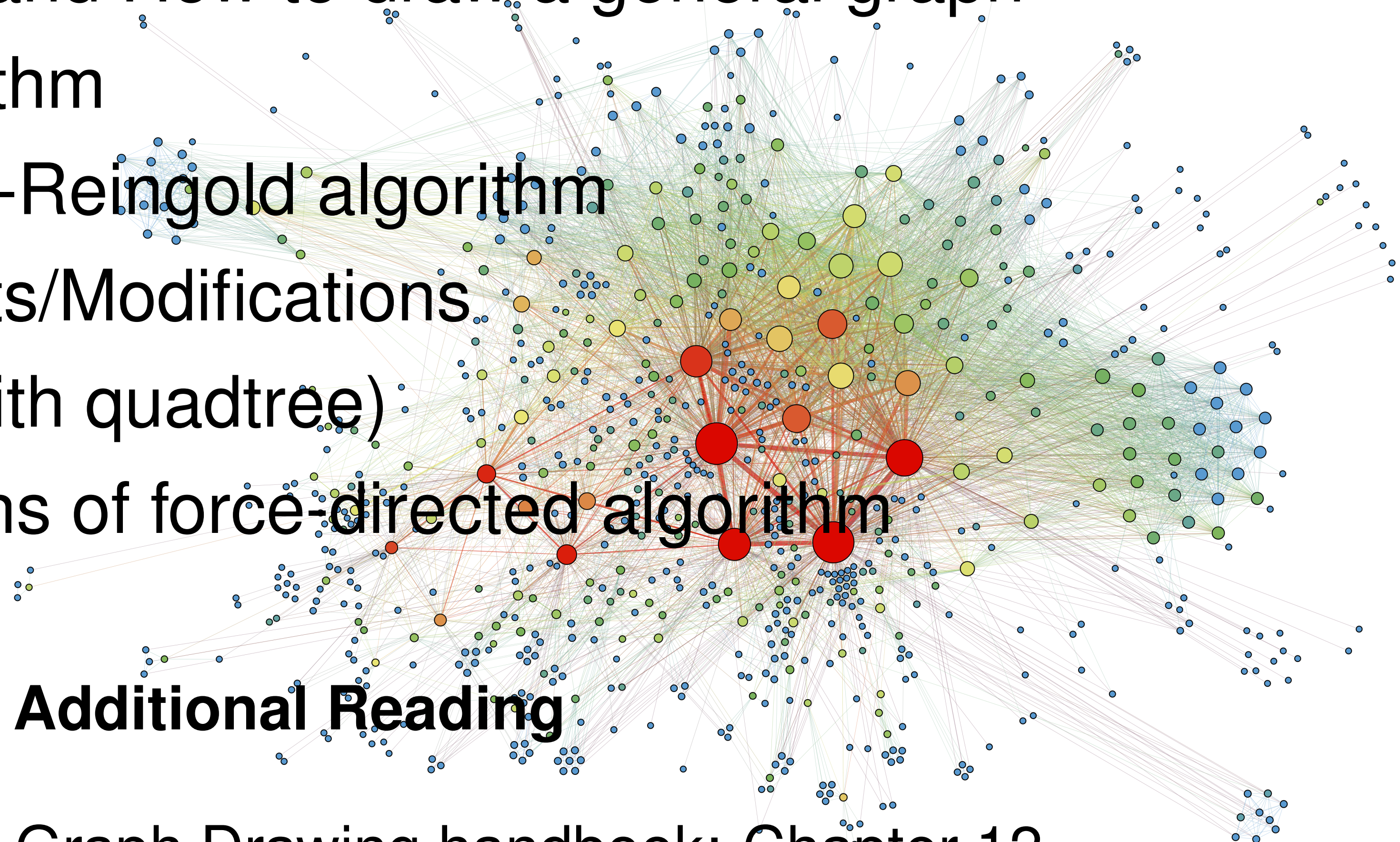
- usually no quality and running time guaranees
- bad choice of starting layout $\rightarrow$ slow convergence
- possibly slow for large graphs
- fine-turning need be done by experts

Summary and Reading

- Introduction and How to draw a general graph
- Eades algorithm
- Fruchterman-Reingold algorithm
- Improvements/Modifications
- Speed up (with quadtree)
- Other versions of force-directed algorithm



Summary and Reading



- Introduction and How to draw a general graph
- Eades algorithm
- Fruchterman-Reingold algorithm
- Improvements/Modifications
- Speed up (with quadtree)
- Other versions of force-directed algorithm

Additional Reading

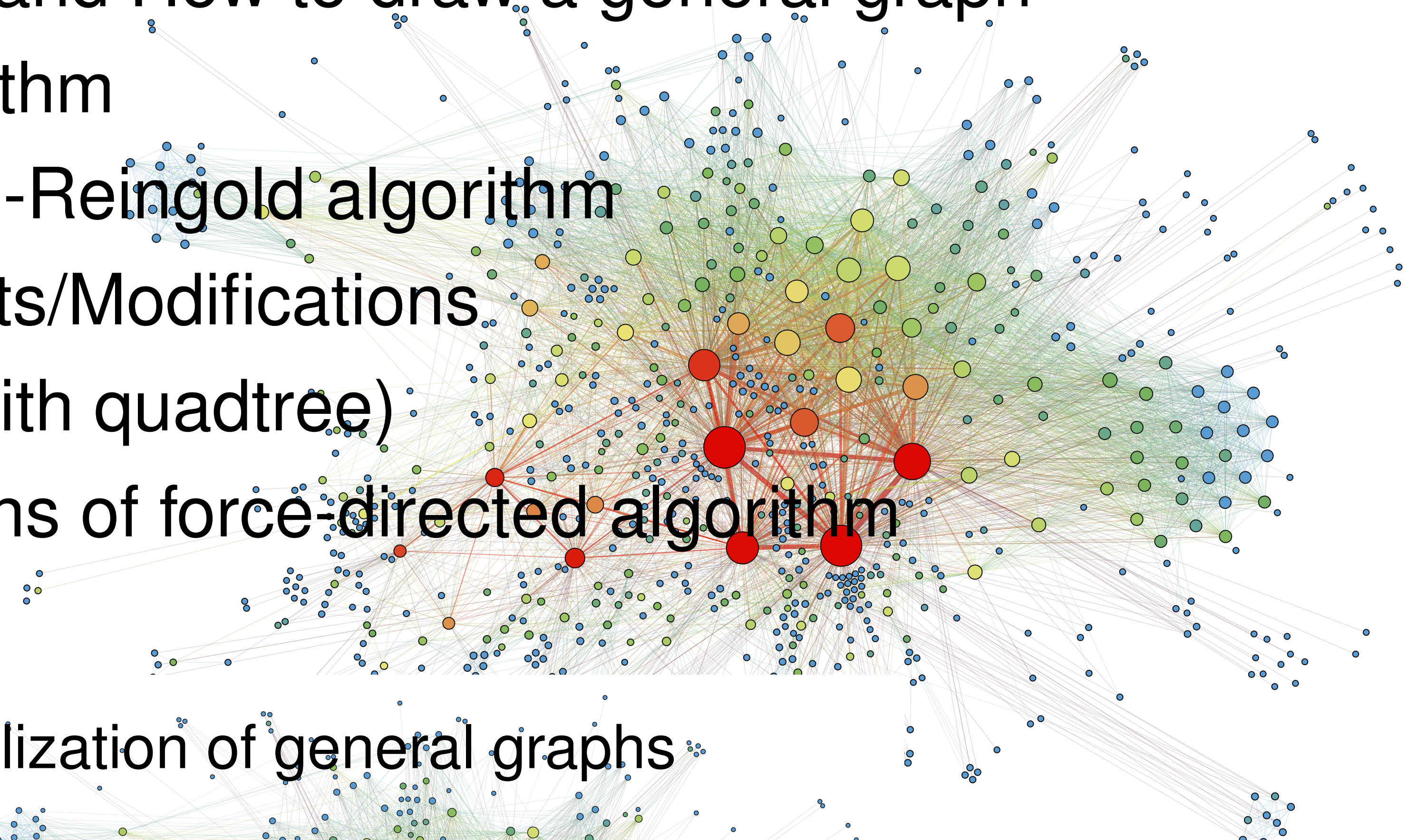
Graph Drawing handbook: Chapter 12

Paper "Graph Drawing by Force-directed placement" by Fruchterman and Reingold

Paper "ForceAtlas2..." by Jacomy et al. - perhaps best version of forces today

Summary and Reading

- Introduction and How to draw a general graph
- Eades algorithm
- Fruchterman-Reingold algorithm
- Improvements/Modifications
- Speed up (with quadtree)
- Other versions of force-directed algorithm



Next

Algorithm for visualization of general graphs