R-Trees

Albert-Jan Yzelman

October 22, 2007

Outline

R-trees

Background

- Internship at Alten Nederland, started the "R-tree Project"
- In close cooperation with Shell Exploration & Production

Background

- Internship at Alten Nederland, started the "R-tree Project"
- In close cooperation with Shell Exploration & Production
- Goals:

Background

- Internship at Alten Nederland, started the "R-tree Project"
- In close cooperation with Shell Exploration & Production
- Goals:
  1. Create a datastructure which will enhance Shell's oil reservoir simulation software

Background

- Internship at Alten Nederland, started the "R-tree Project"
- In close cooperation with Shell Exploration & Production
- Goals:
  1. Create a datastructure which will enhance Shell's oil reservoir simulation software
  2. Obtain a generic, customisable R-tree software library for using R-trees in many other areas

People involved

- Arno Swart (consultant from Alten, working at Shell; direct supervisor)
- Gerard Sleijpen (supervisor from Utrecht University)

People involved

- Arno Swart (consultant from Alten, working at Shell; direct supervisor)
- Gerard Sleijpen (supervisor from Utrecht University)
- Eric Haesen (manager from Alten Nederland)
- Hans Molenaar (manager from Shell E&P)

People involved

- Arno Swart (consultant from Alten, working at Shell; direct supervisor)
- Gerard Sleijpen (supervisor from Utrecht University)
- Eric Haesen (manager from Alten Nederland)
- Hans Molenaar (manager from Shell E&P)
- Many Alten consultants; Guy Calluy, Ron van Kesteren, Martijn Souman, Petr Sereda

Shell's oil reservoir simulation software; *Dynamo*

- Numerical modelling of oil reservoirs
- Simulates effects of, for example, well drilling at different locations

Outline

R-trees

Discretisation

- An oil reservoir discretisation (or *grid*) to work with is required for (FVM) simulation

Discretisation

- An oil reservoir discretisation (or *grid*) to work with is required for (FVM) simulation
- Grid extraction is done externally, by companies like *Petrel*; grid data thus is read in from externally supplied files

Discretisation

- An oil reservoir discretisation (or *grid*) to work with is required for (FVM) simulation
- Grid extraction is done externally, by companies like *Petrel*; grid data thus is read in from externally supplied files
- Such a grid is irregularly shaped, due to the many different ground layers and so called *fault lines* in the reservoir area

Example grid

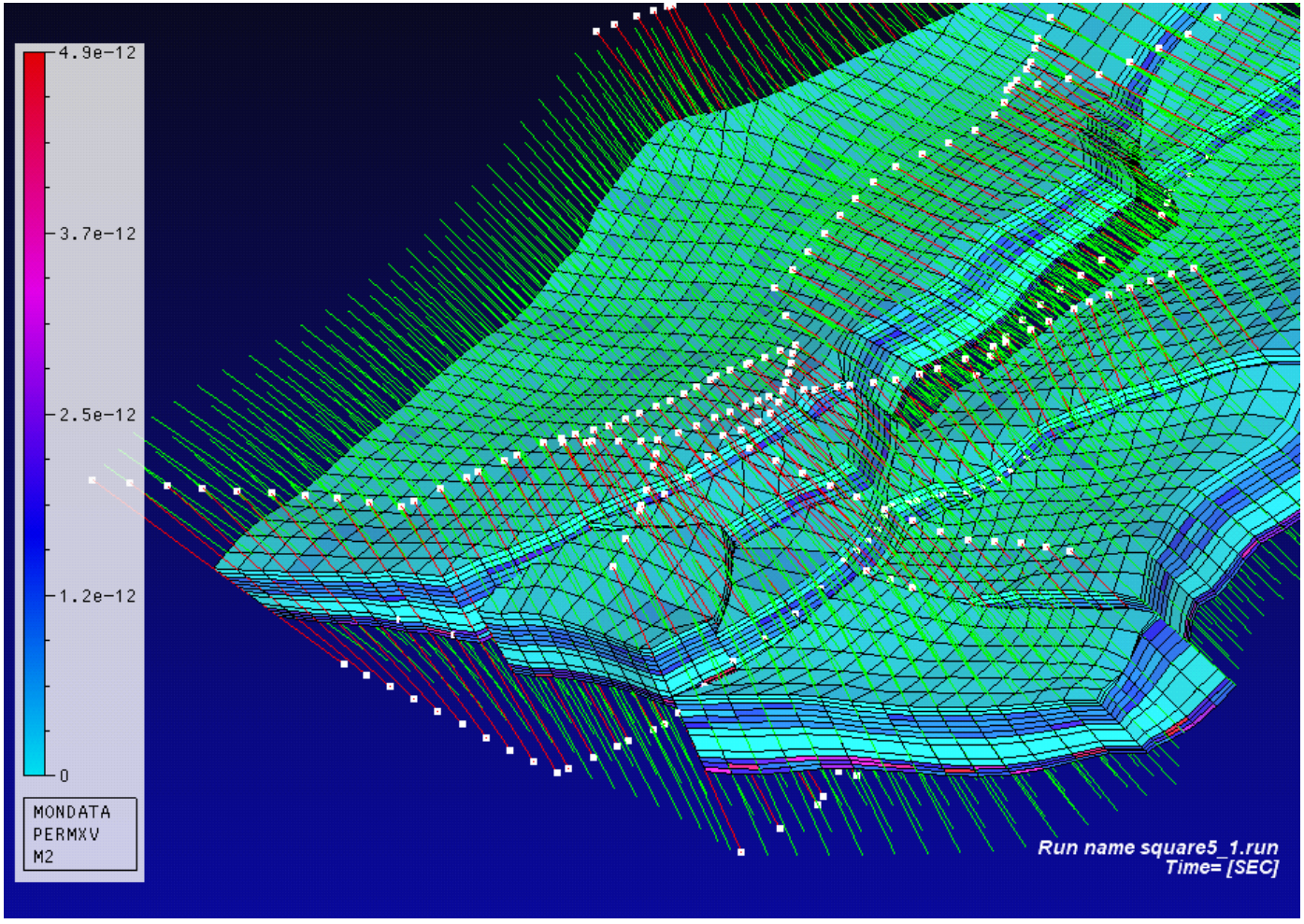


Figure: Example oil reservoir grid

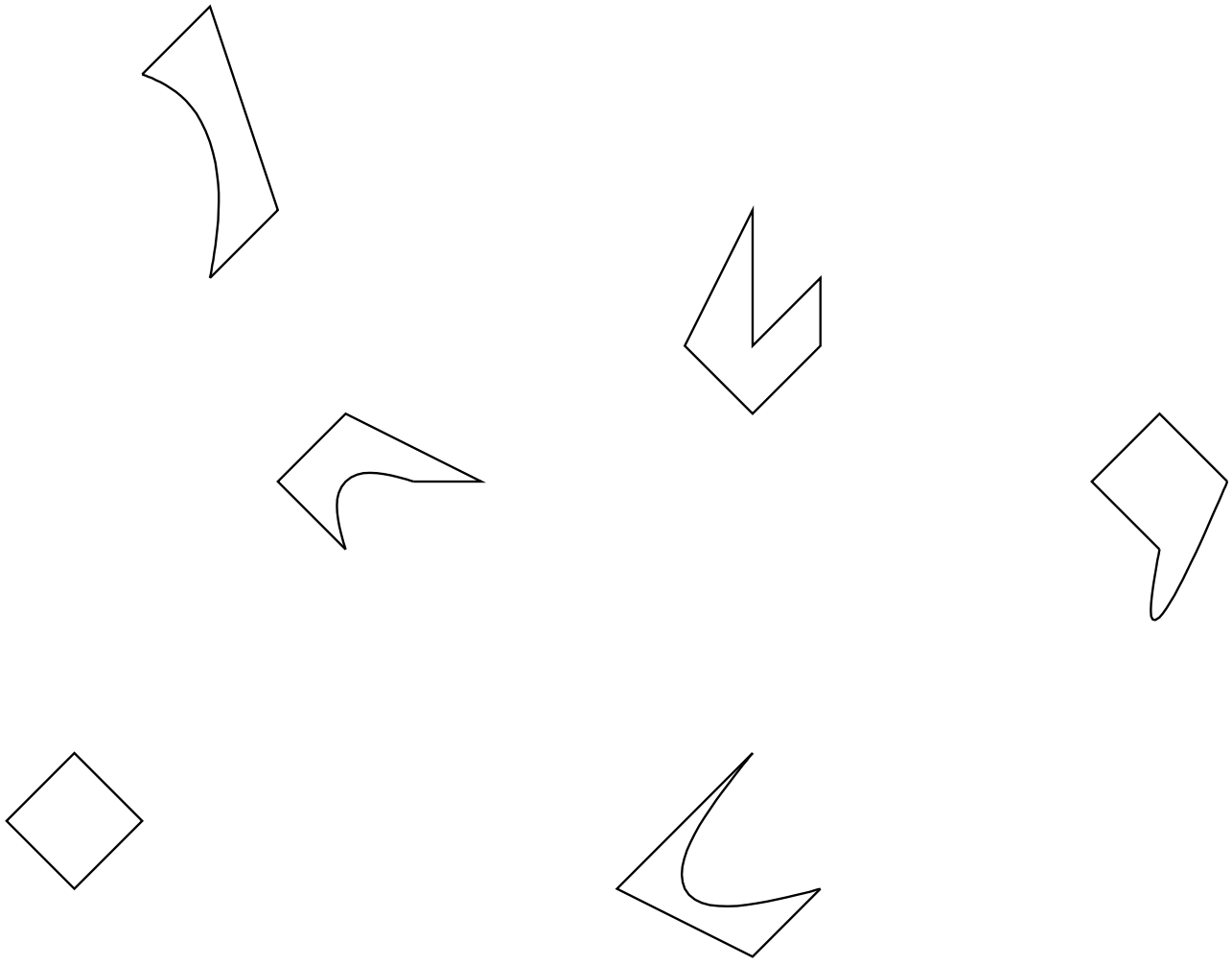
Grid operations

The simulation software and grid upscaling software included in Dynamo require efficient answers to at least the following types of queries:

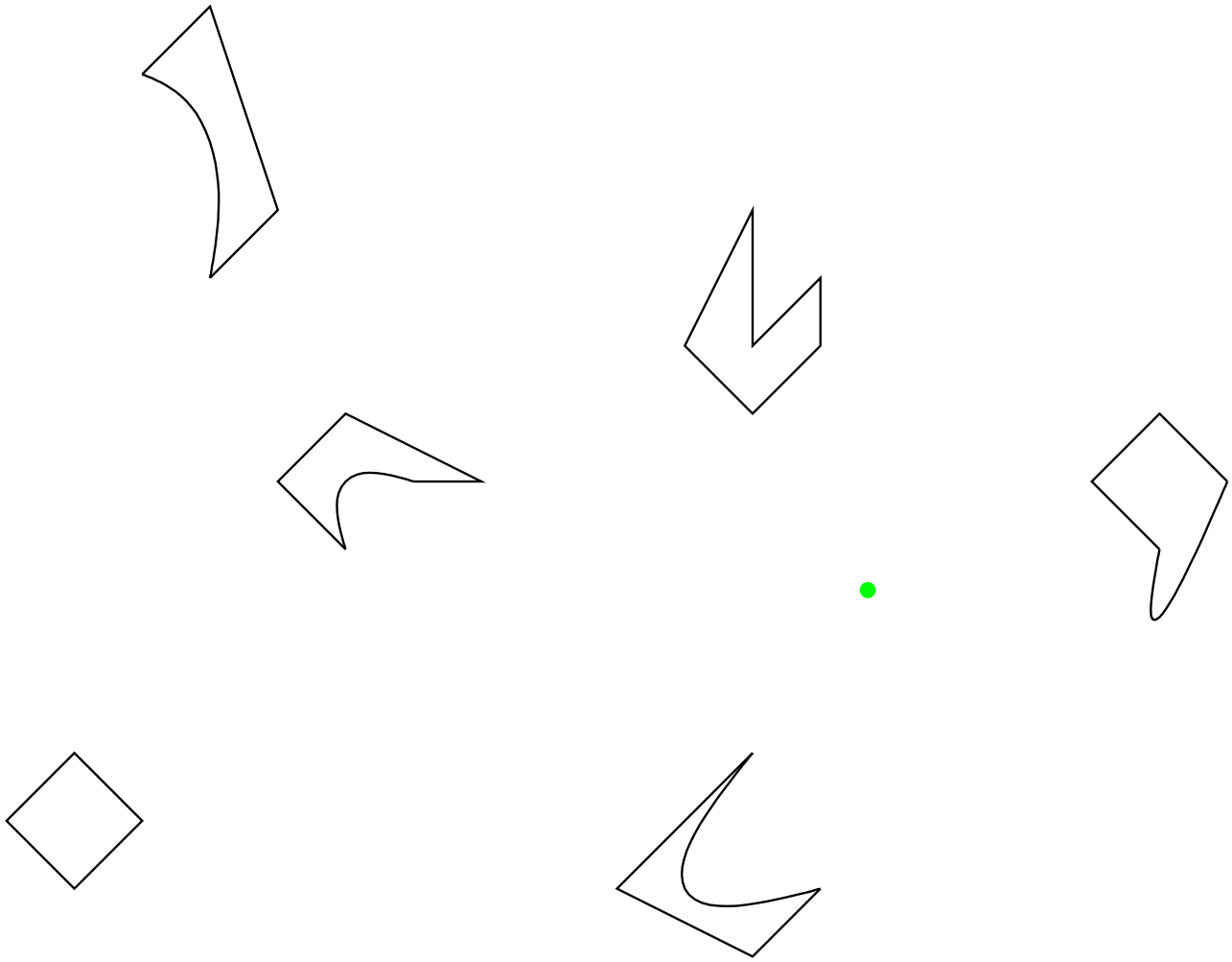
- Point
- Line
- Box

The irregular input grids and the geometric nature of querying call for a specialised data structure.

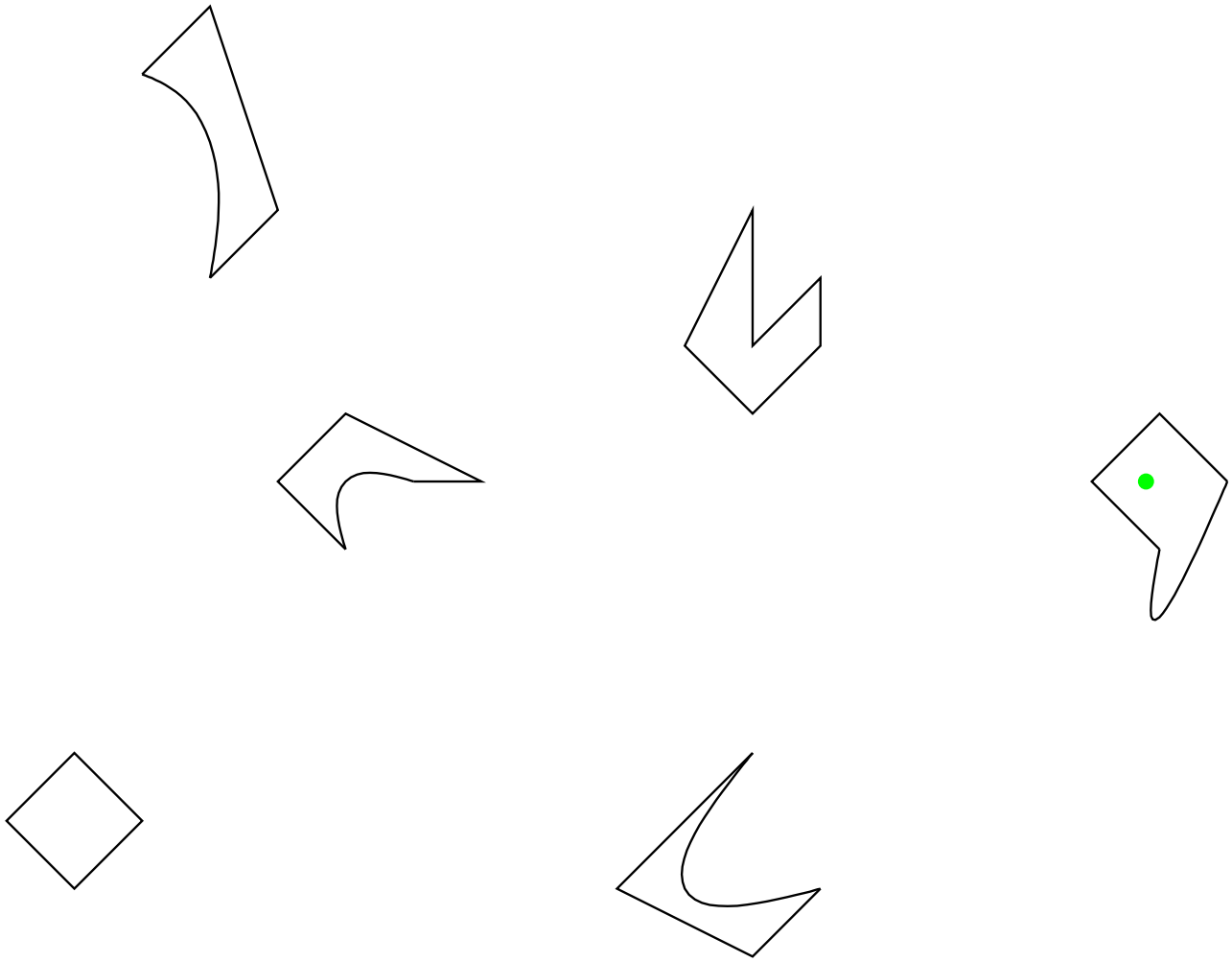
Example data



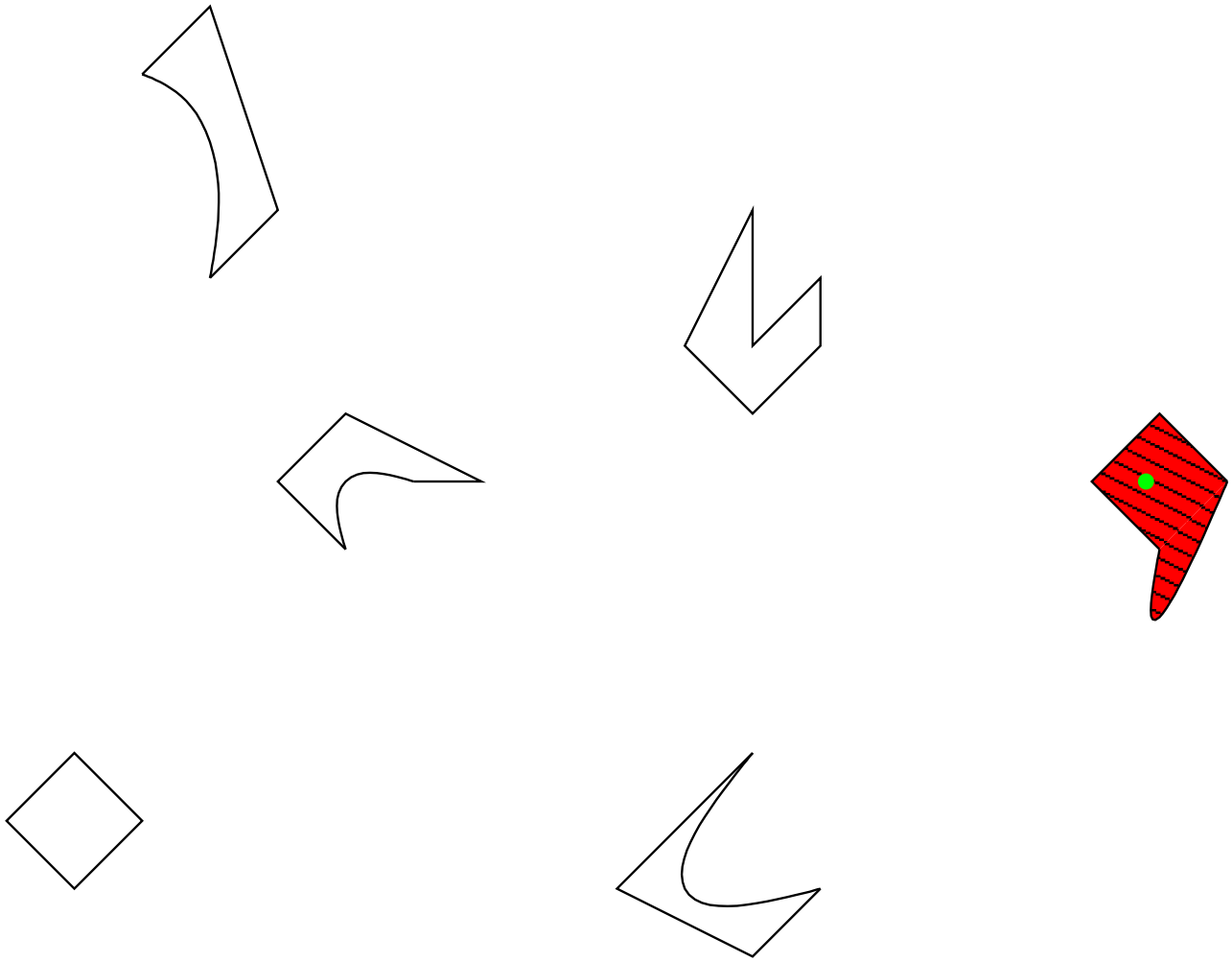
Point query



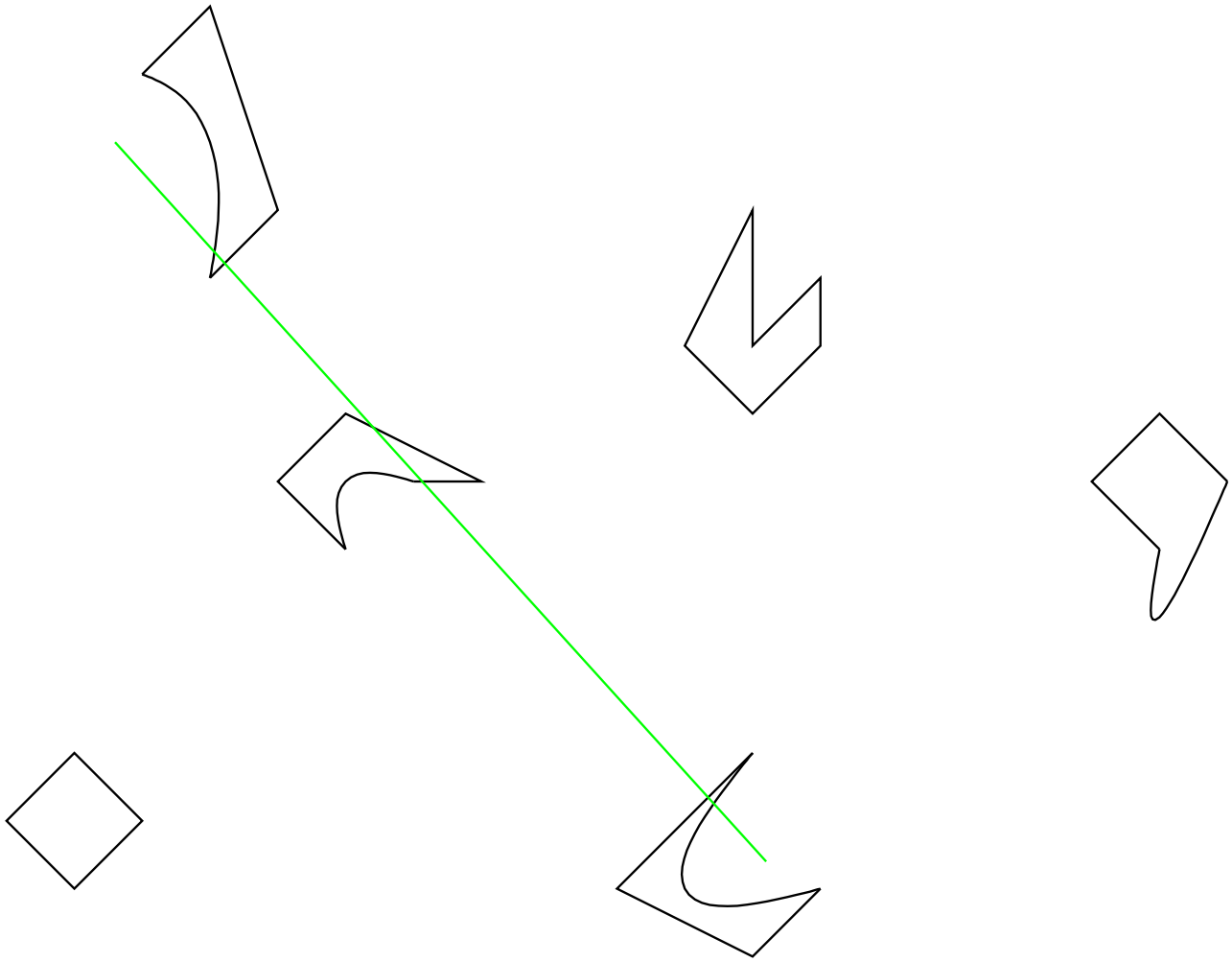
Point query



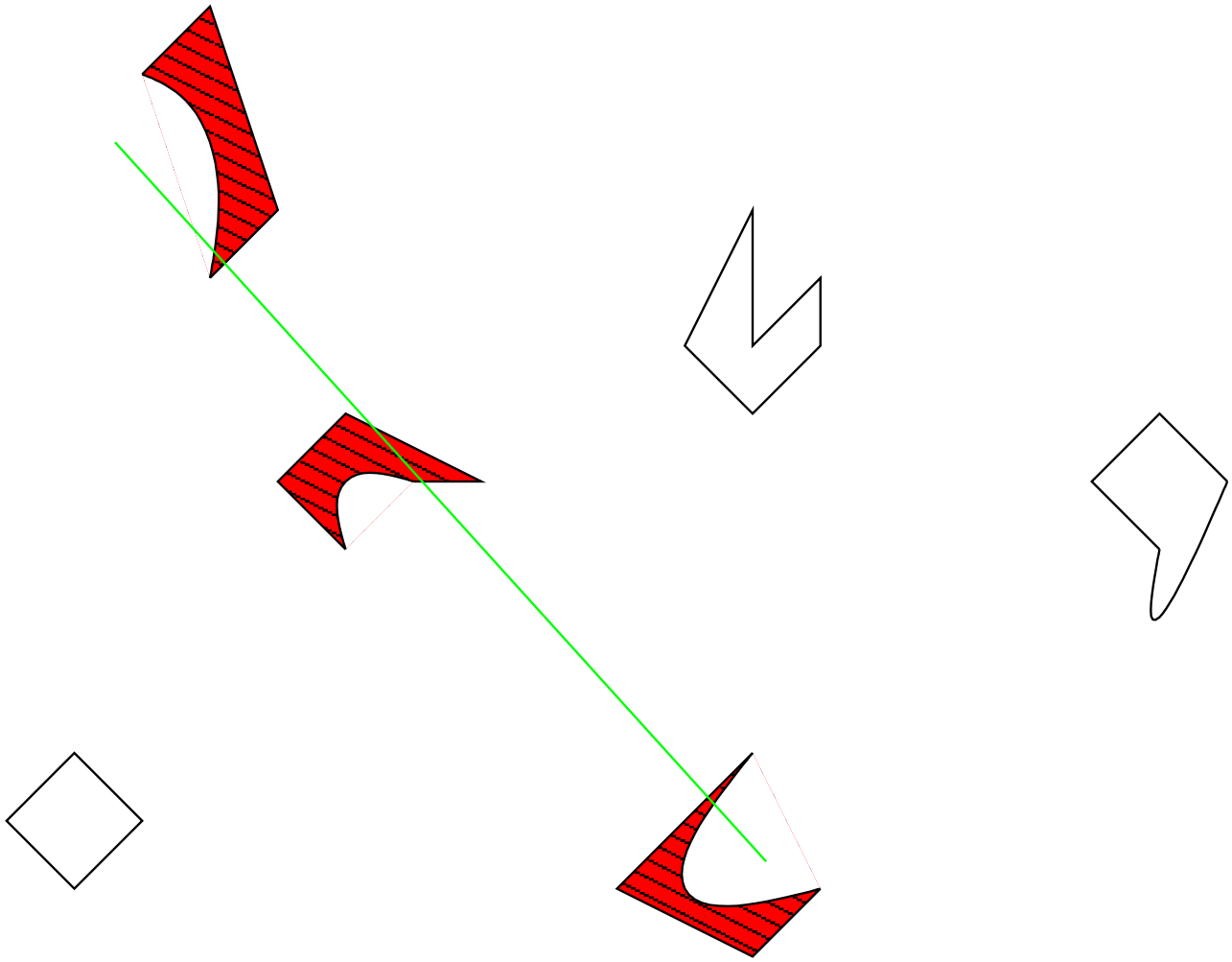
Point query



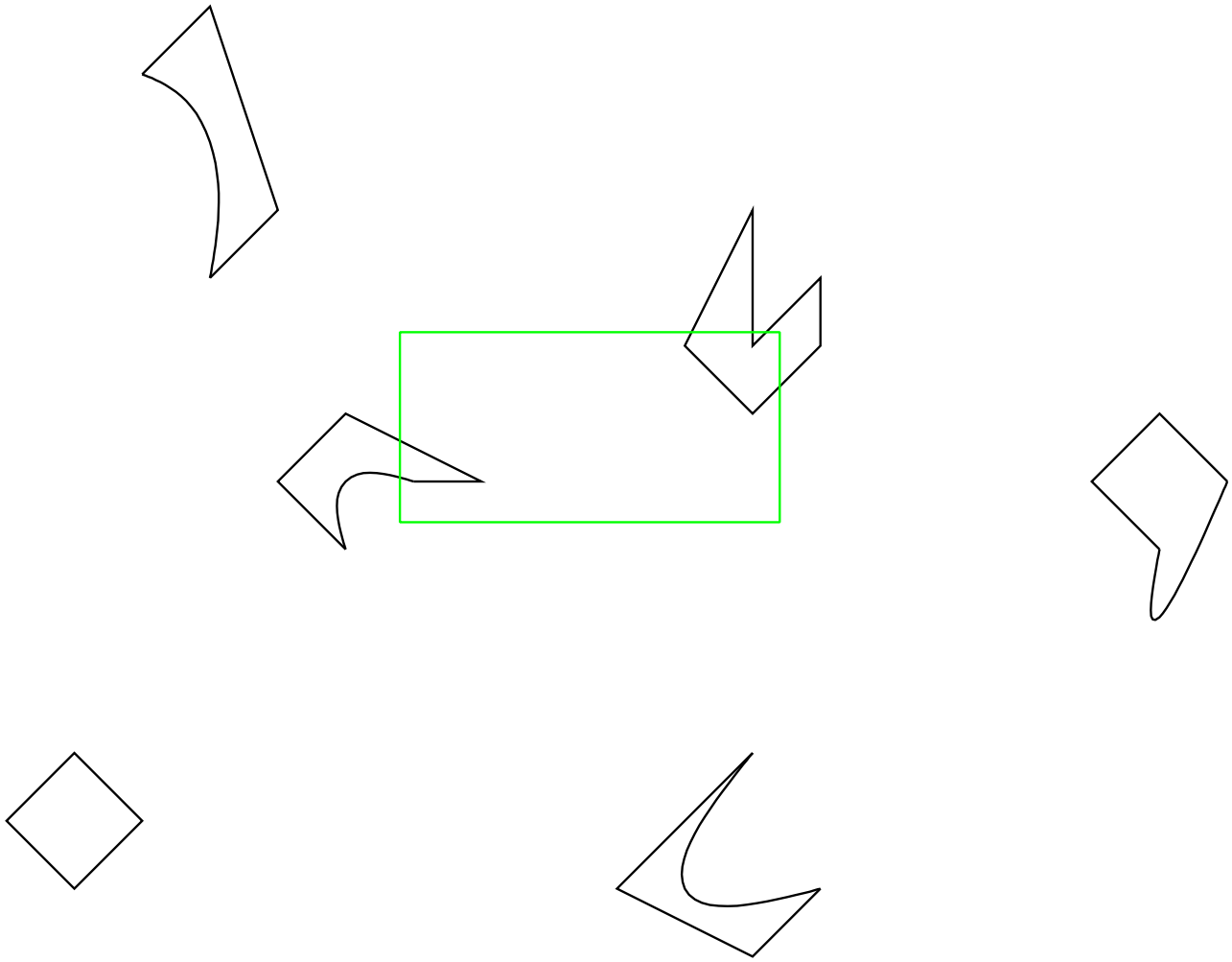
Line query



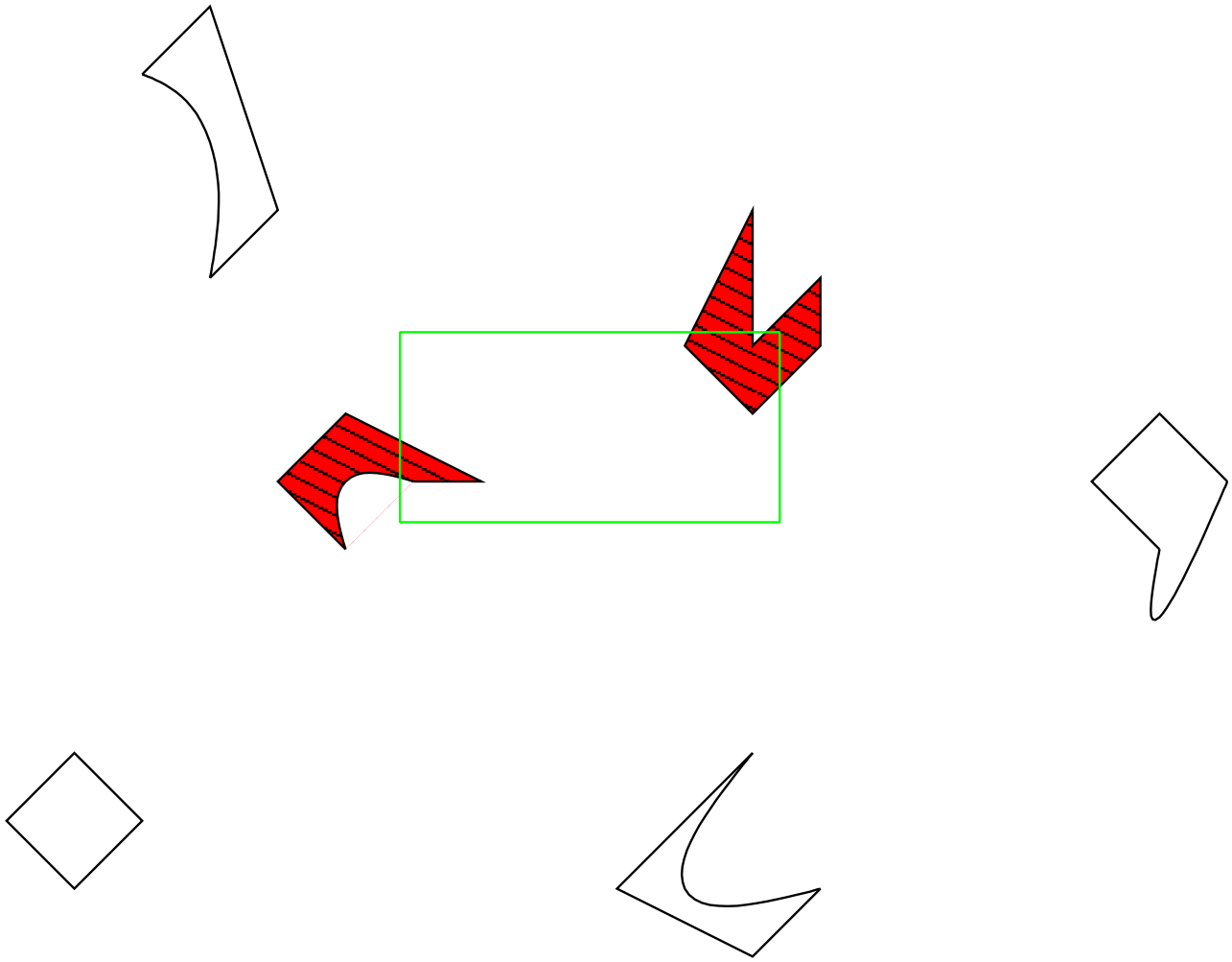
Line query



Box query



Box query



Data structure demands:

Data structure demands:

- Can store *d*-dimensional *hypercubes*

Data structure demands:

- Can store *d*-dimensional *hypercubes*
- Performs point, line, and box queries as fast as possible...

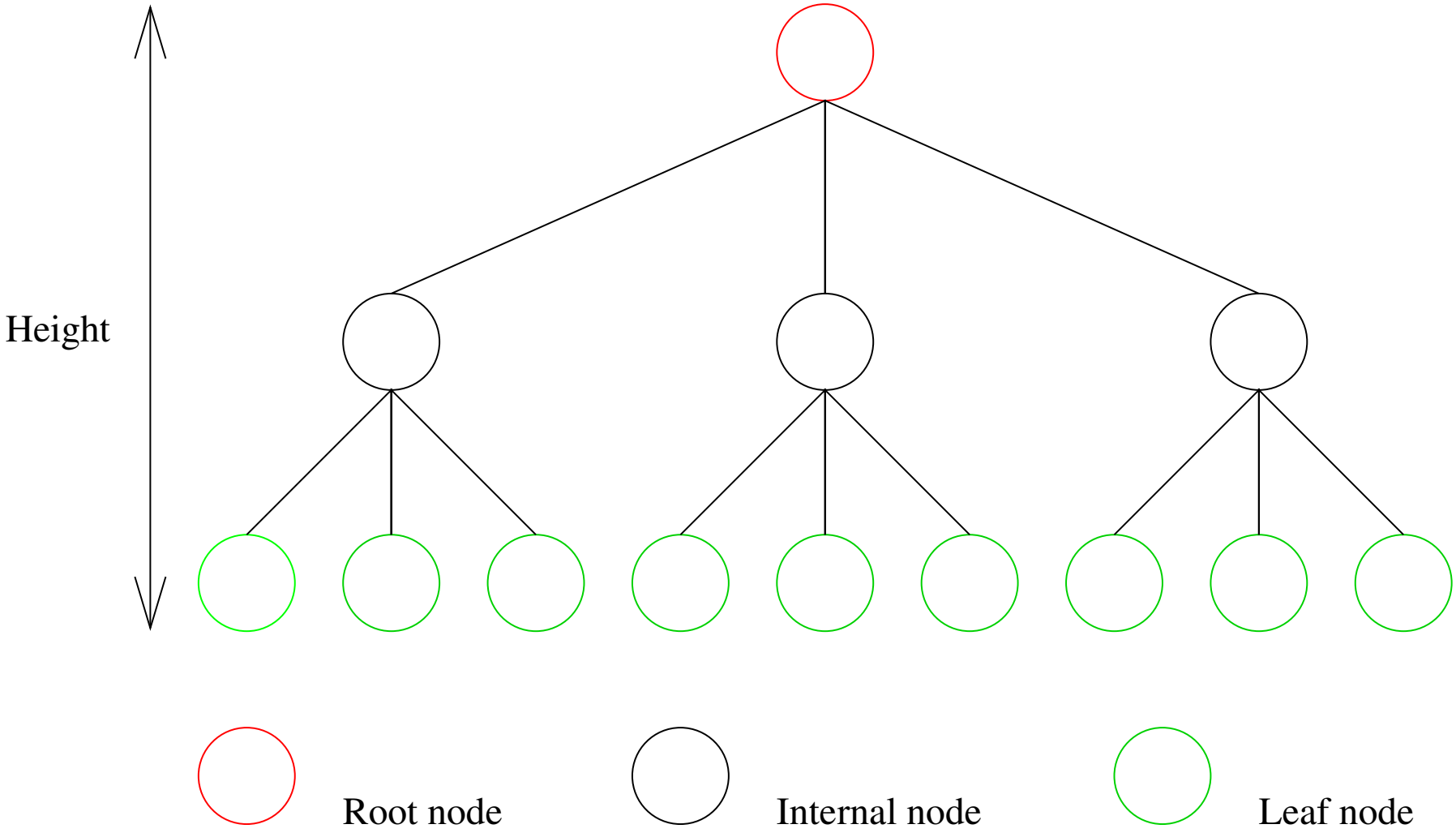
Data structure demands:

- Can store *d*-dimensional *hypercubes*
- Performs point, line, and box queries as fast as possible...
- ...but also keeps memory usage in check!

R-tree ingredients:

1. Tree-like data structure
2. Minimal Bounding Rectangles (MBRs)

(1) The tree data structure



(2) Minimum Bounding Rectangle (MBR)

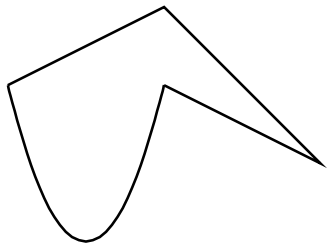


Figure: A two-dimensional object

(2) Minimum Bounding Rectangle (MBR)

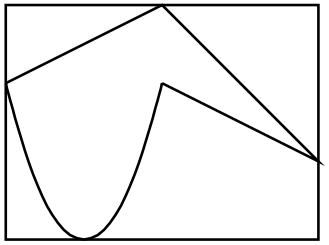


Figure: A two-dimensional object with its MBR

(2) Minimum Bounding Rectangle (MBR)

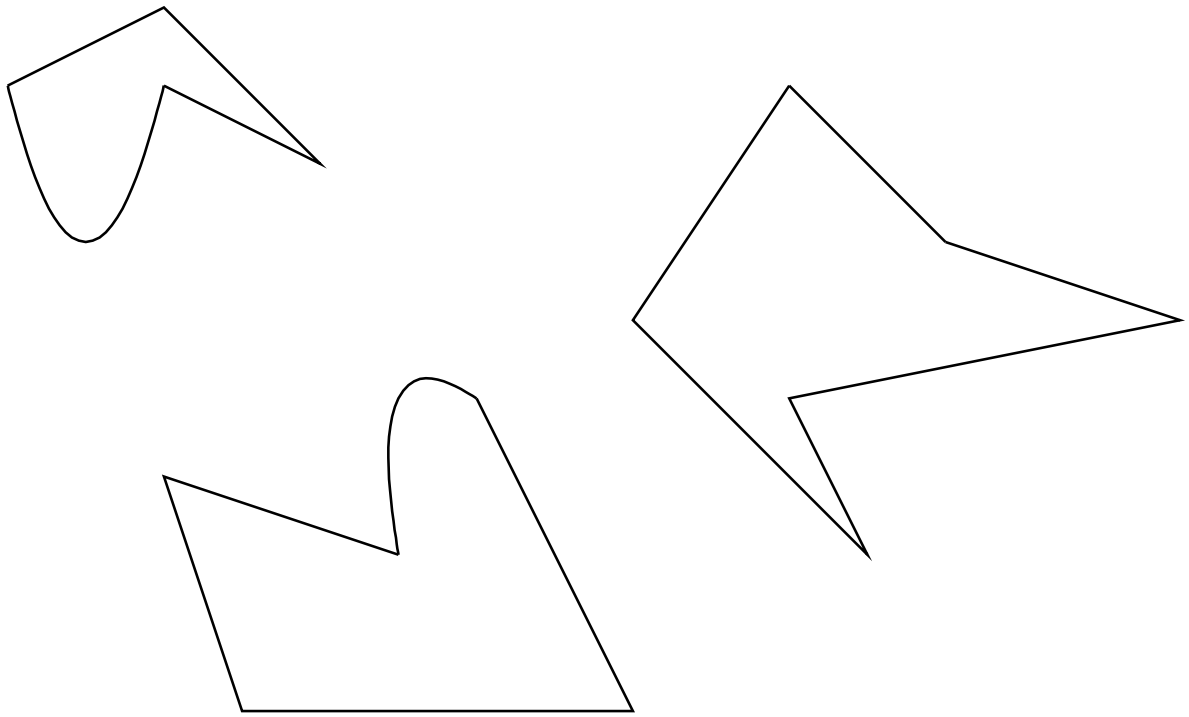


Figure: Multiple objects

(2) Minimum Bounding Rectangle (MBR)

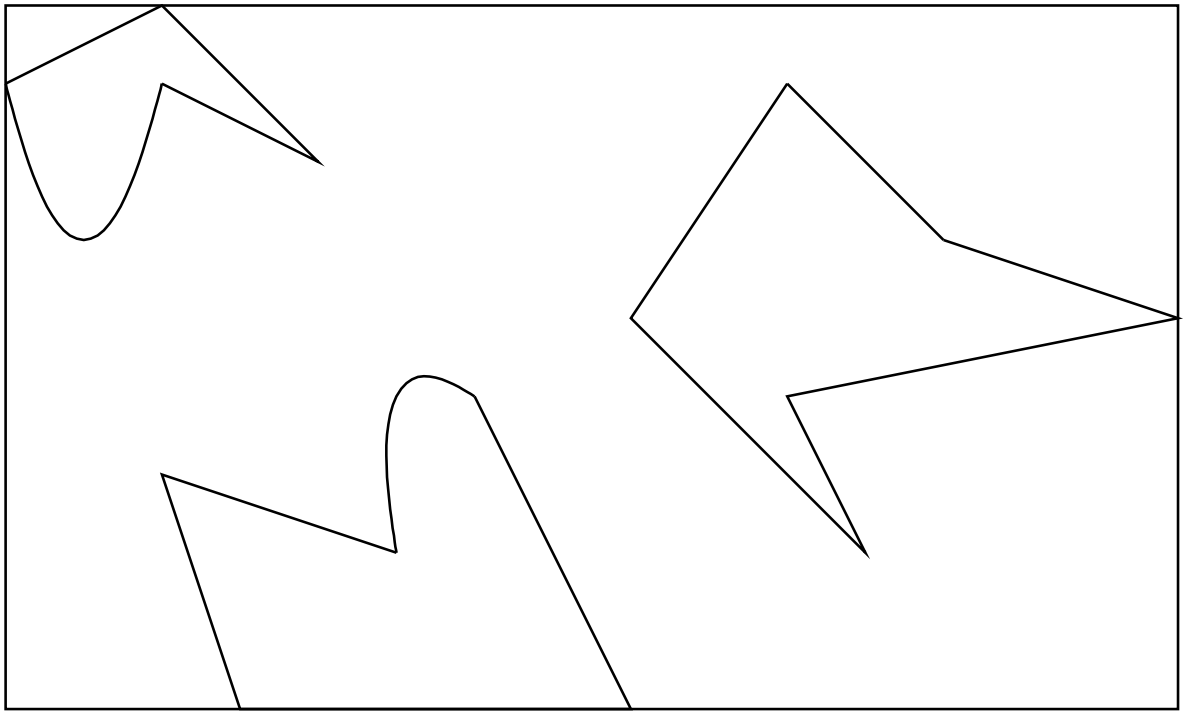


Figure: Multiple objects contained in their MBR

(2) Minimum Bounding Rectangle (MBR)

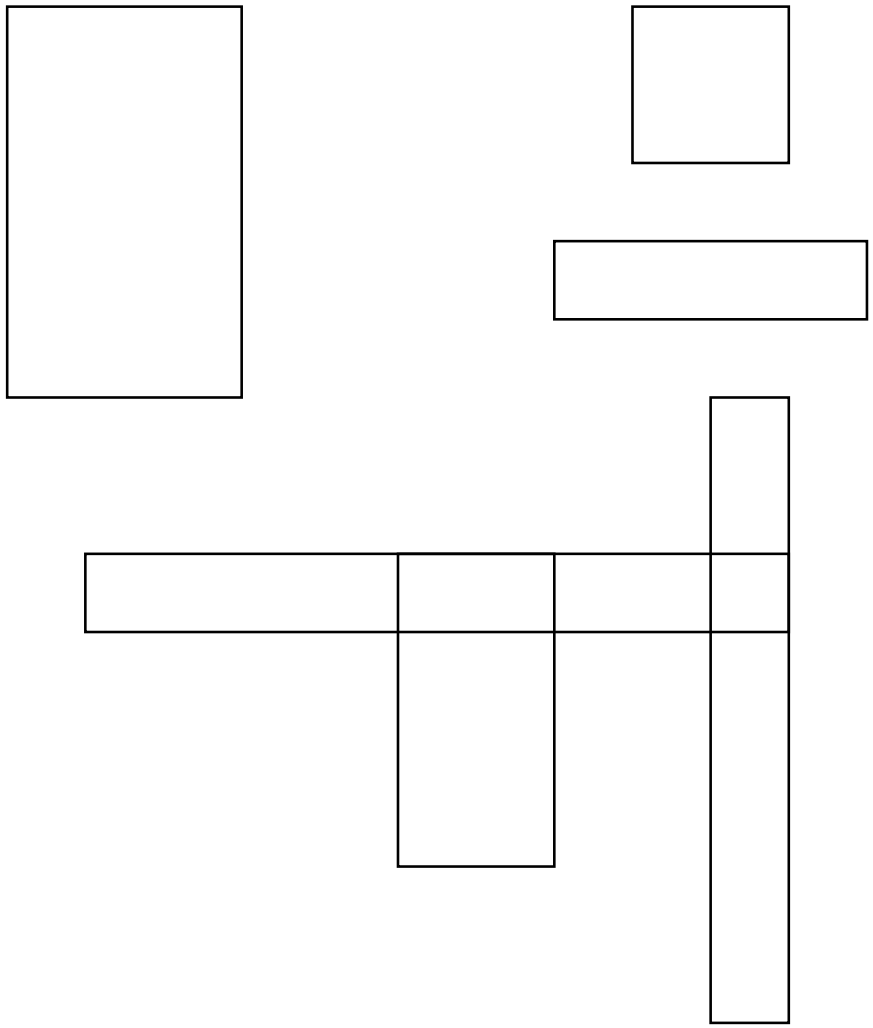


Figure: Multiple MBRs

(2) Minimum Bounding Rectangle (MBR)

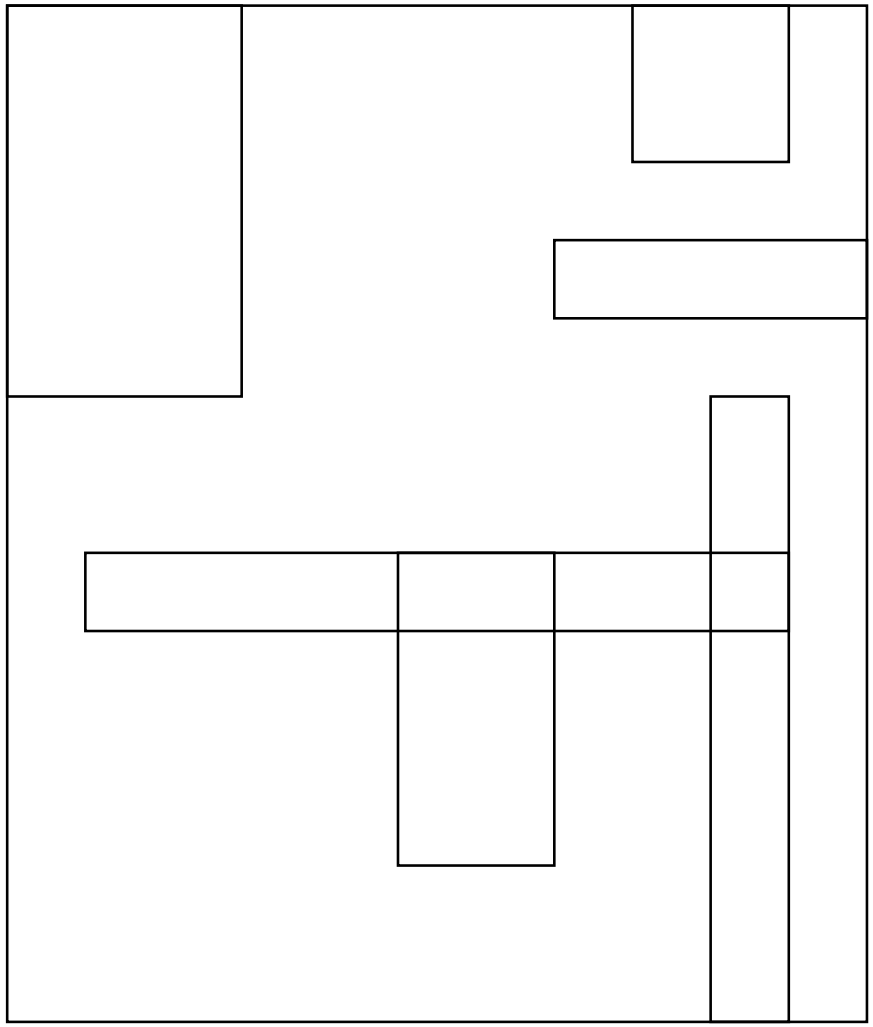


Figure: Multiple MBRs contained in another MBR

R-trees

We can let each node of a tree correspond to an MBR and obtain the so called R-trees.

R-trees

We can let each node of a tree correspond to an MBR and obtain the so called R-trees.

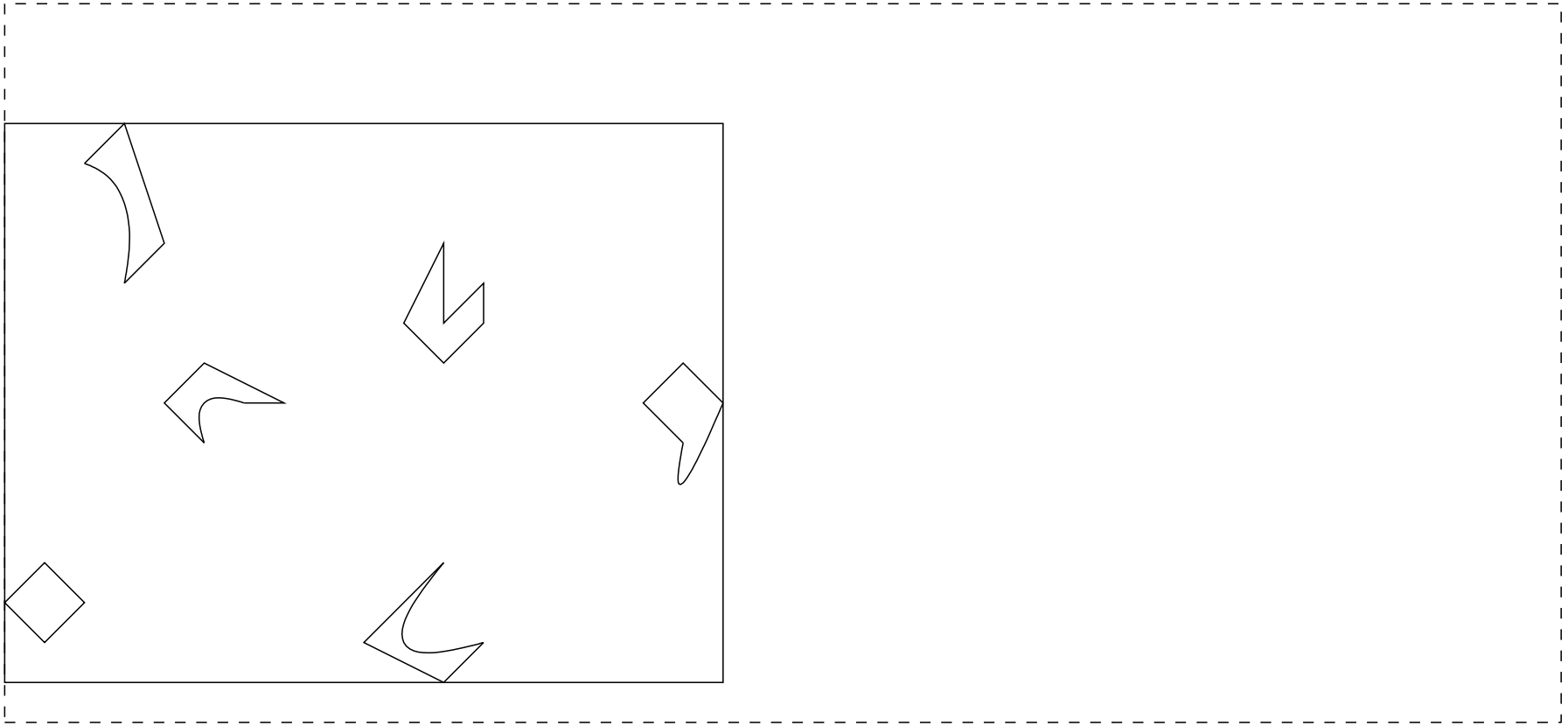
- Let each internal node of the R-tree correspond to the MBR of all MBRs stored in the children of that internal node

R-trees

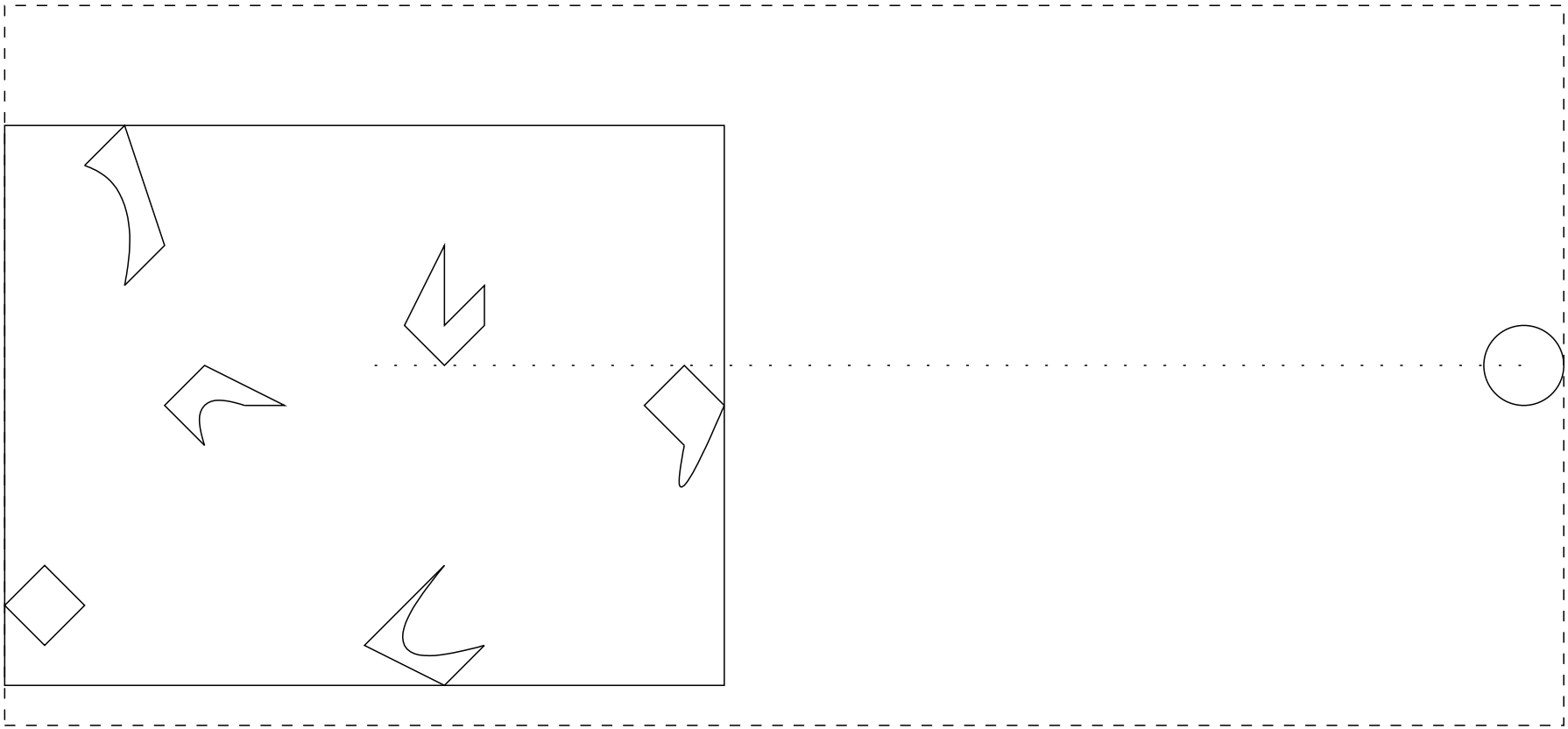
We can let each node of a tree correspond to an MBR and obtain the so called R-trees.

- Let each internal node of the R-tree correspond to the MBR of all MBRs stored in the children of that internal node
- Let each leaf node of the R-tree correspond to the MBR of a single object stored in the R-tree

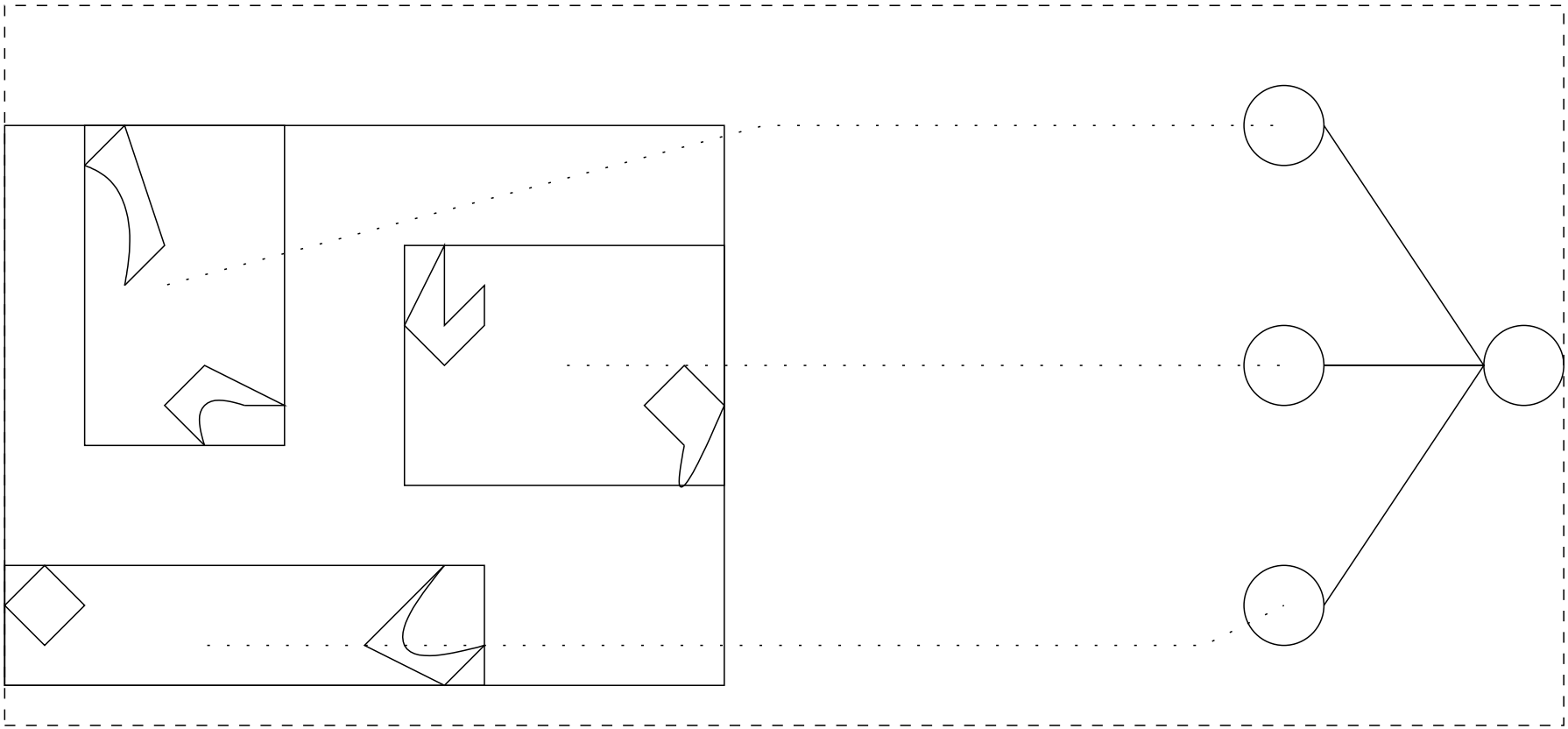
R-trees



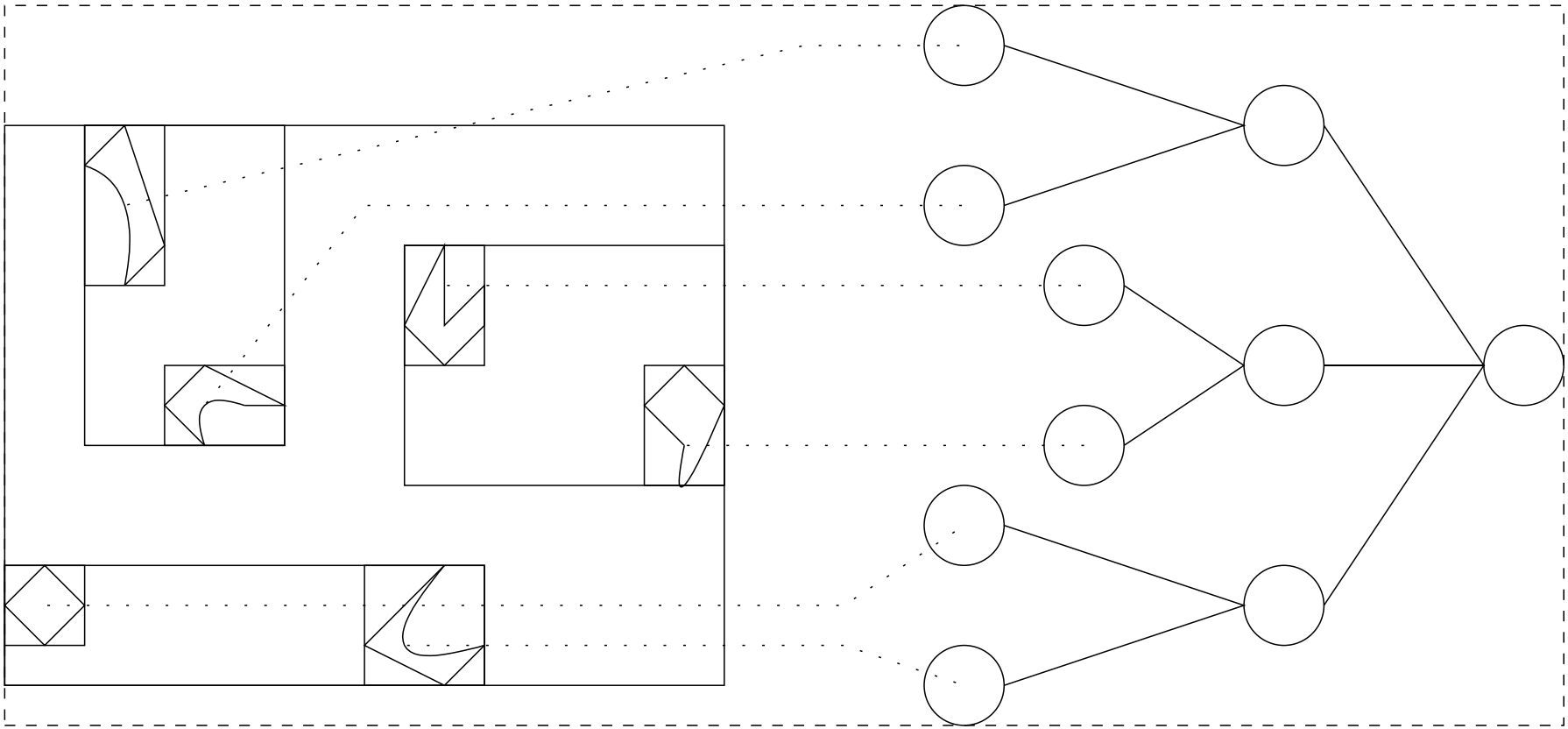
R-trees



R-trees



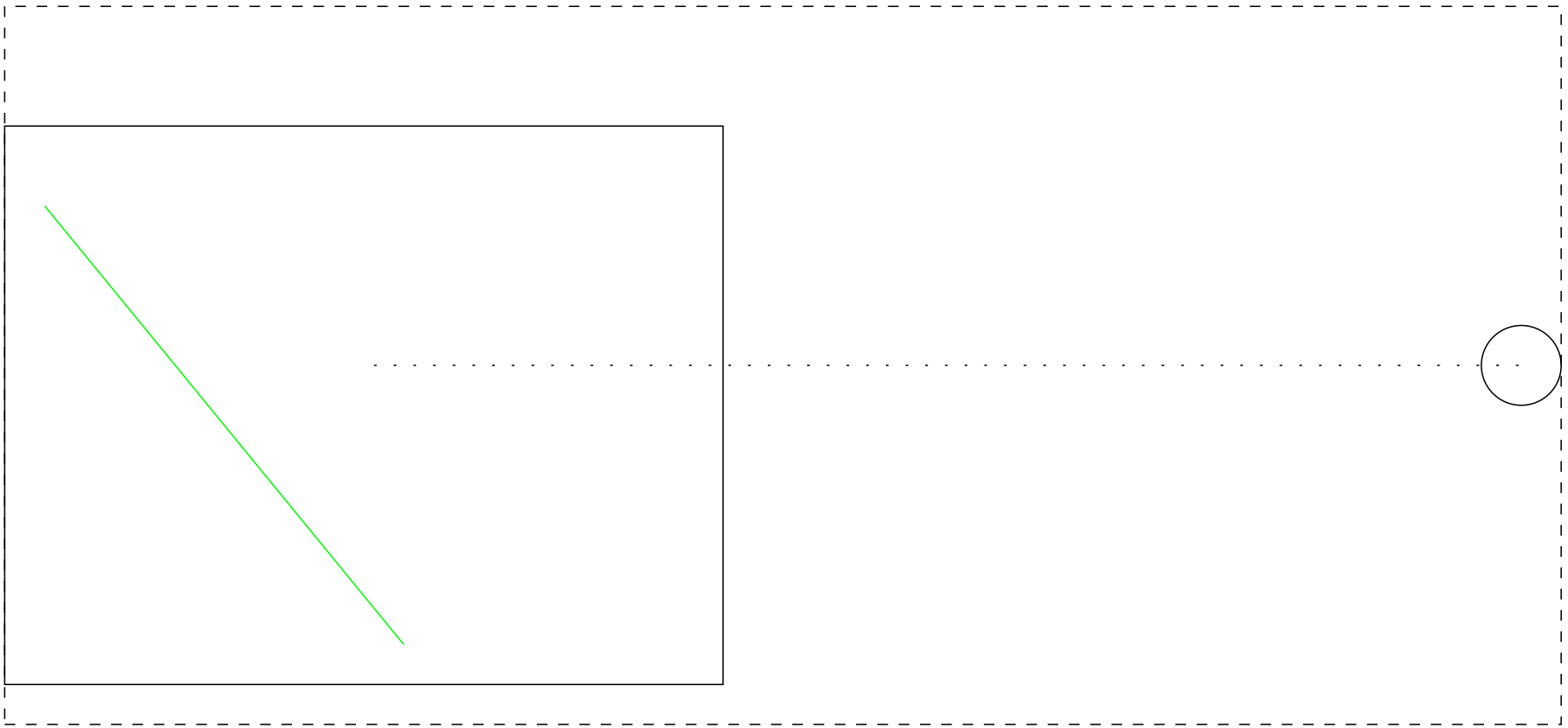
R-trees



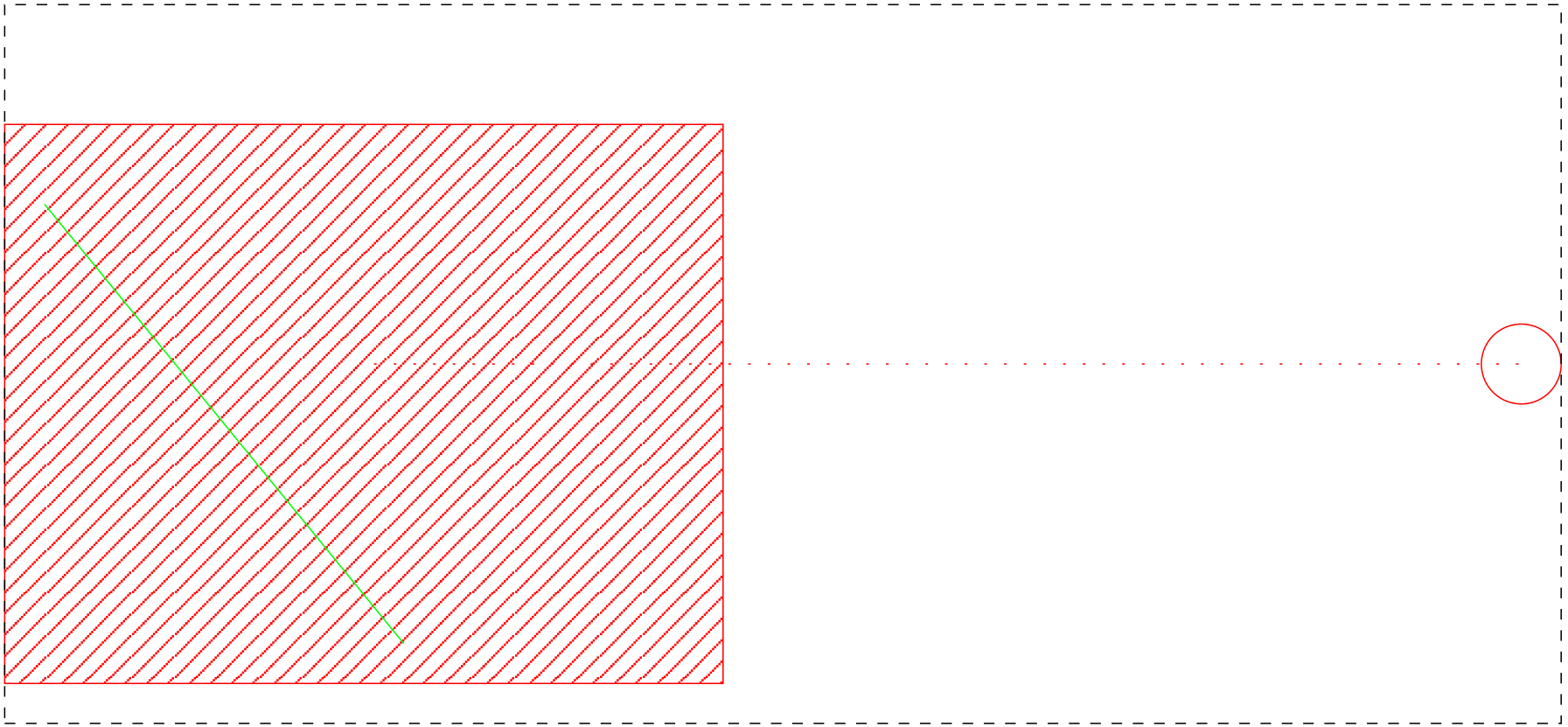
Further R-tree properties

- Each node contains between m and M children
- All leaves are on the same tree level

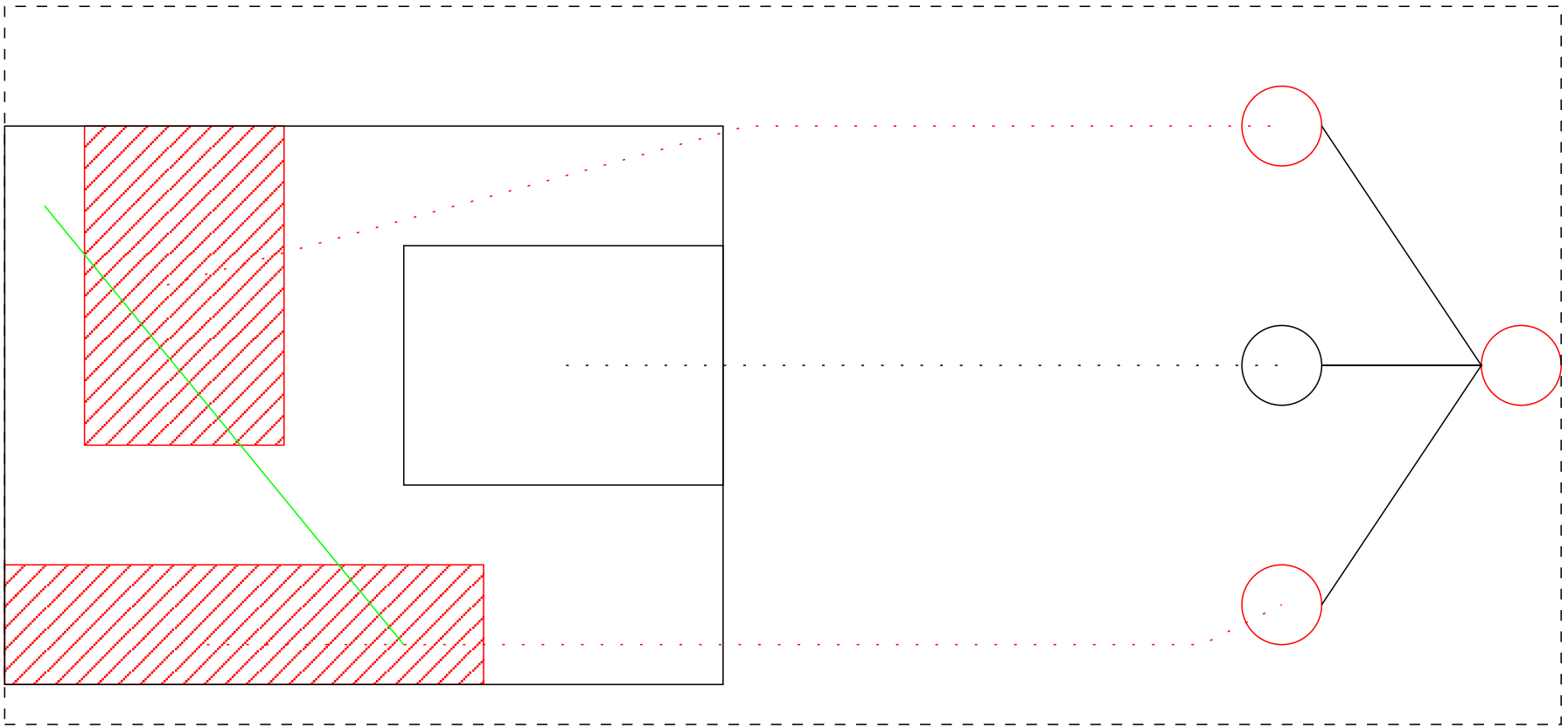
Demonstration of a line query



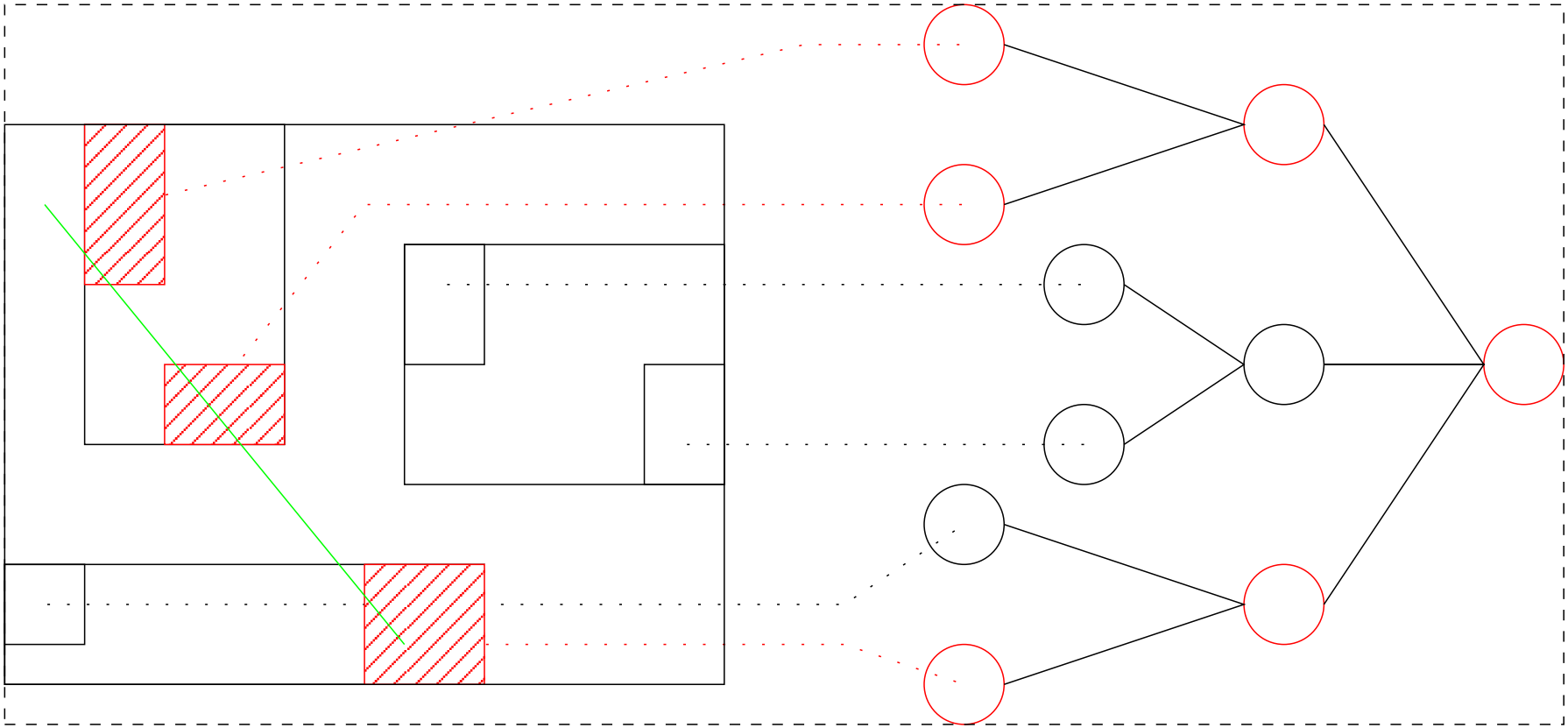
Demonstration of a line query



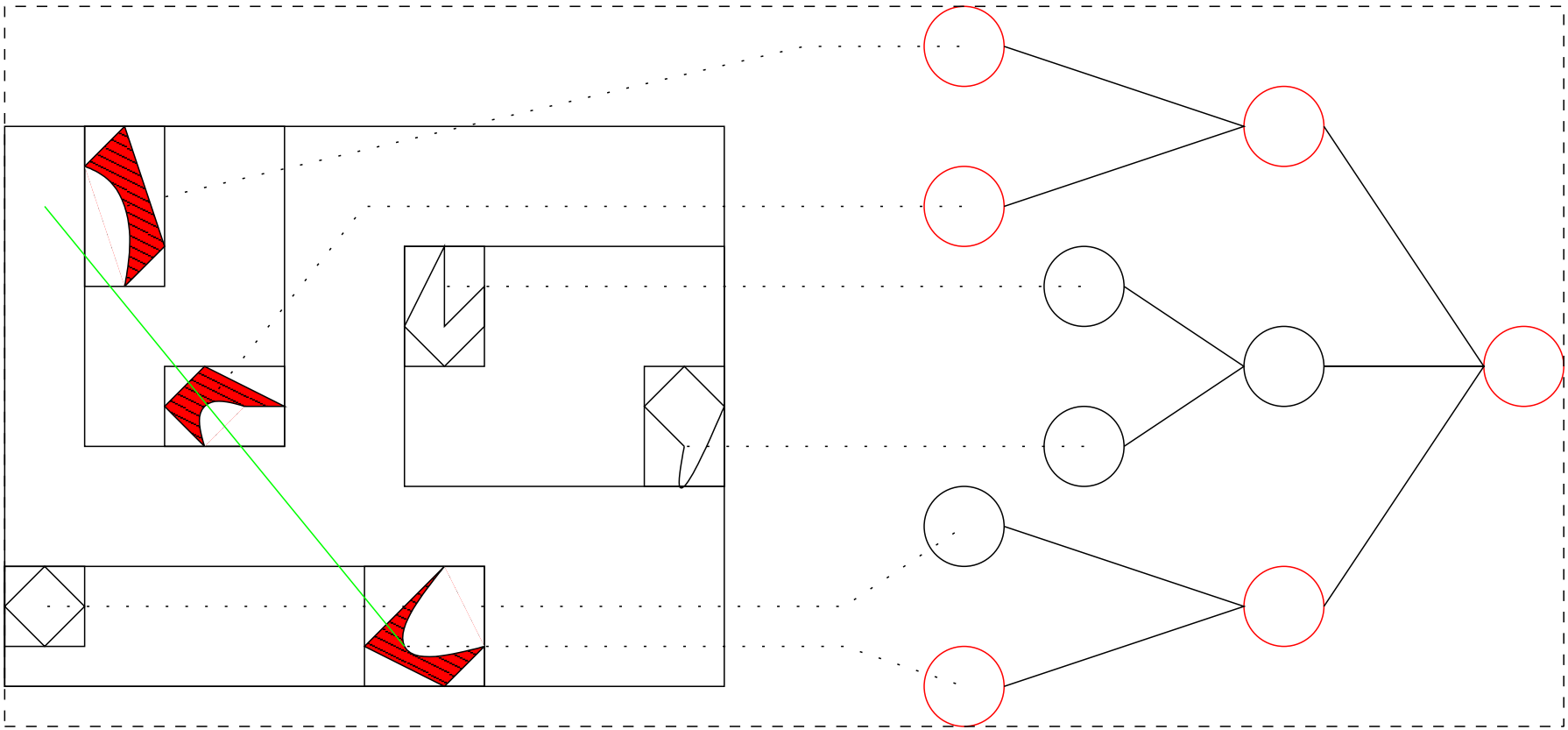
Demonstration of a line query



Demonstration of a line query



Demonstration of a line query



Asymptotic query time

$$\begin{aligned} t_{\text{query}} &\leq c \cdot \log_m n, \quad n \to \infty \\ &= O(\log n) \end{aligned}$$

Because:

- the tree is tallest when each internal node has precisely m children, and
- the tree is balanced.

Outline

R-trees

Actively researched R-tree variations:

- The original R-tree, introduced in 1984
- Top-down Greedy Split (TGS)
- Hilbert R-tree
- Hilbert TGS

Grouping criteria

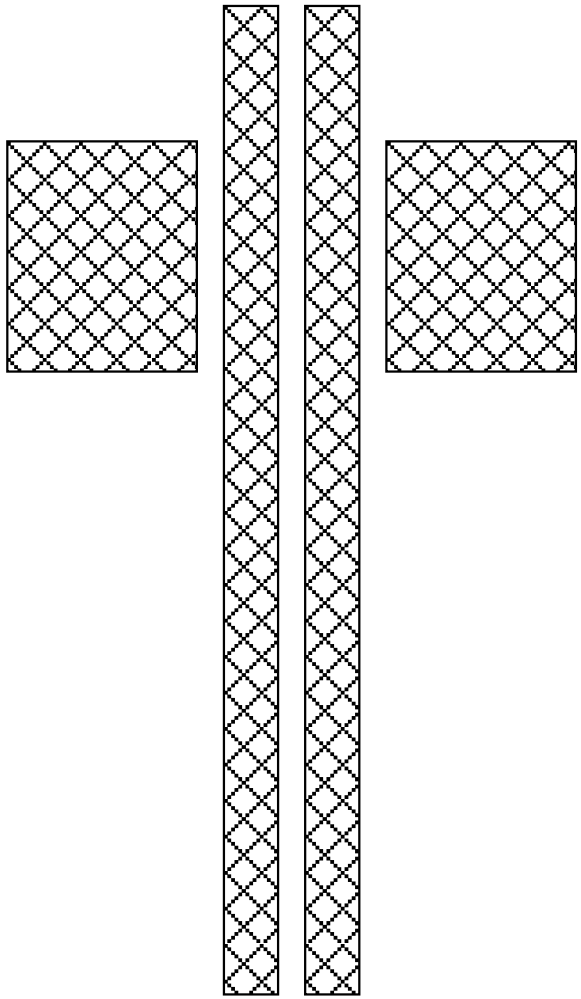


Figure: Four to-be grouped MBRs

Grouping criteria

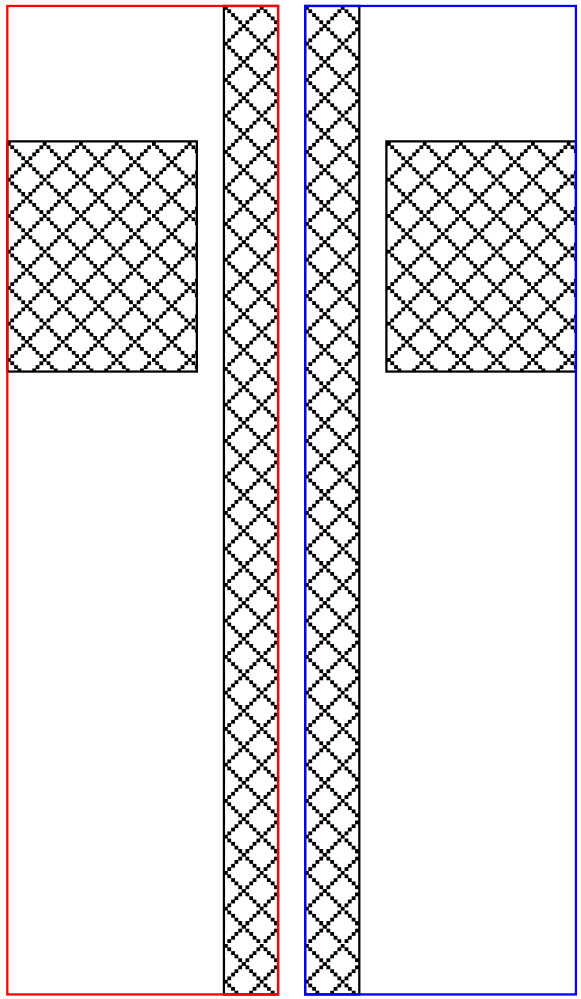


Figure: Minimum overlap criteria

Grouping criteria

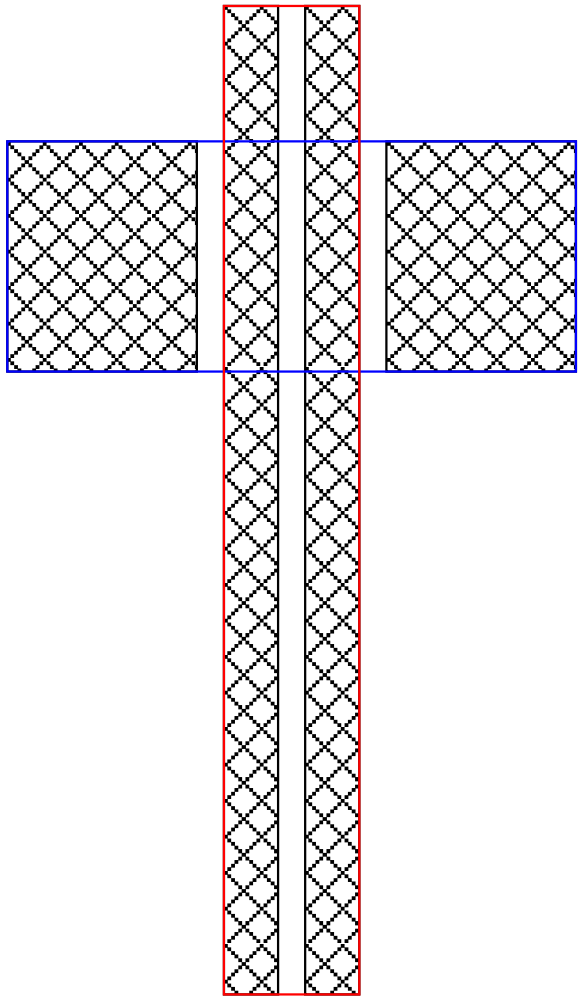
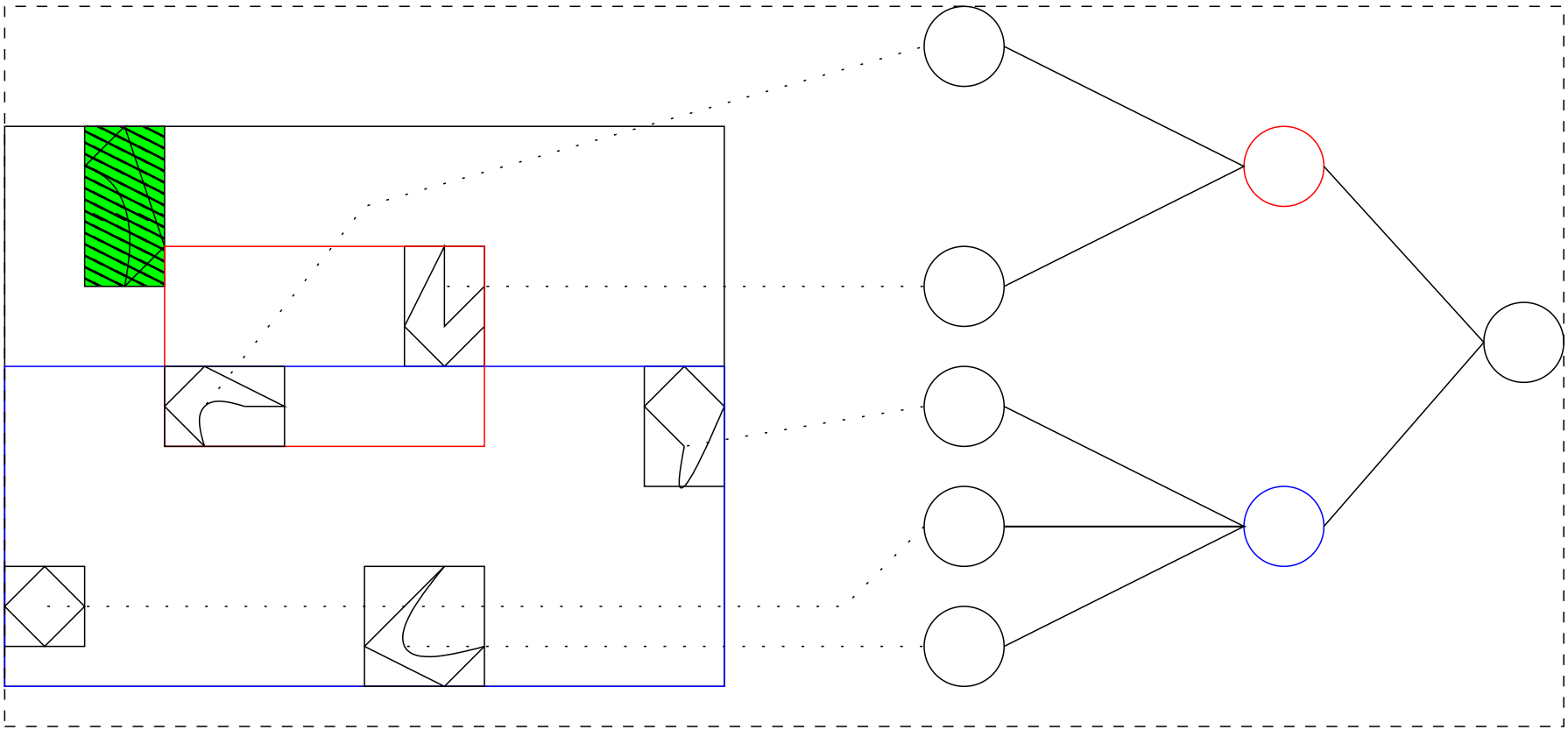


Figure: Minimum total volume criteria

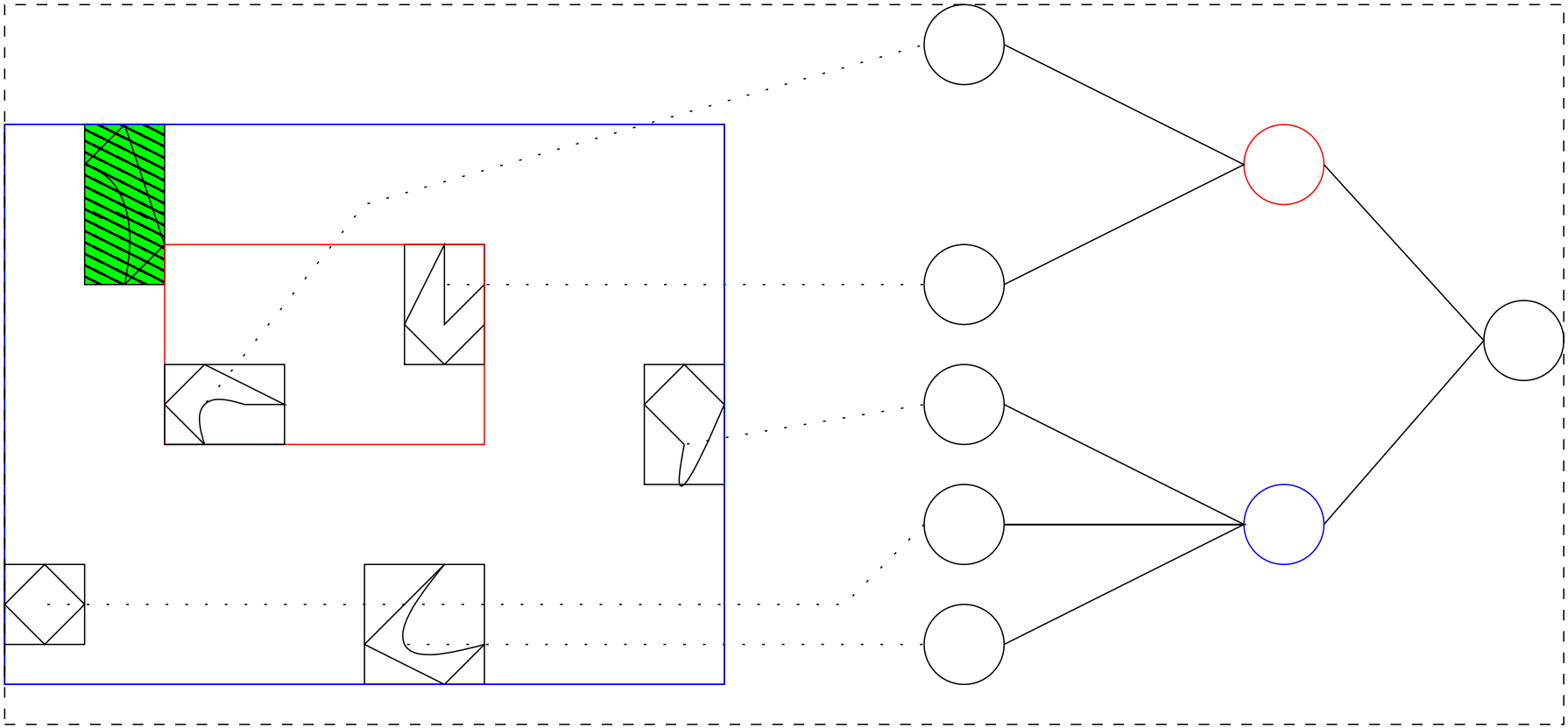
Original dynamic R-tree

We start out with an empty R-tree and insert new objects one-by-one.
For this we need an *insertion algorithm* which works on arbitrary R-trees.
Consider the following example with $m = 2$ and $M = 3$.

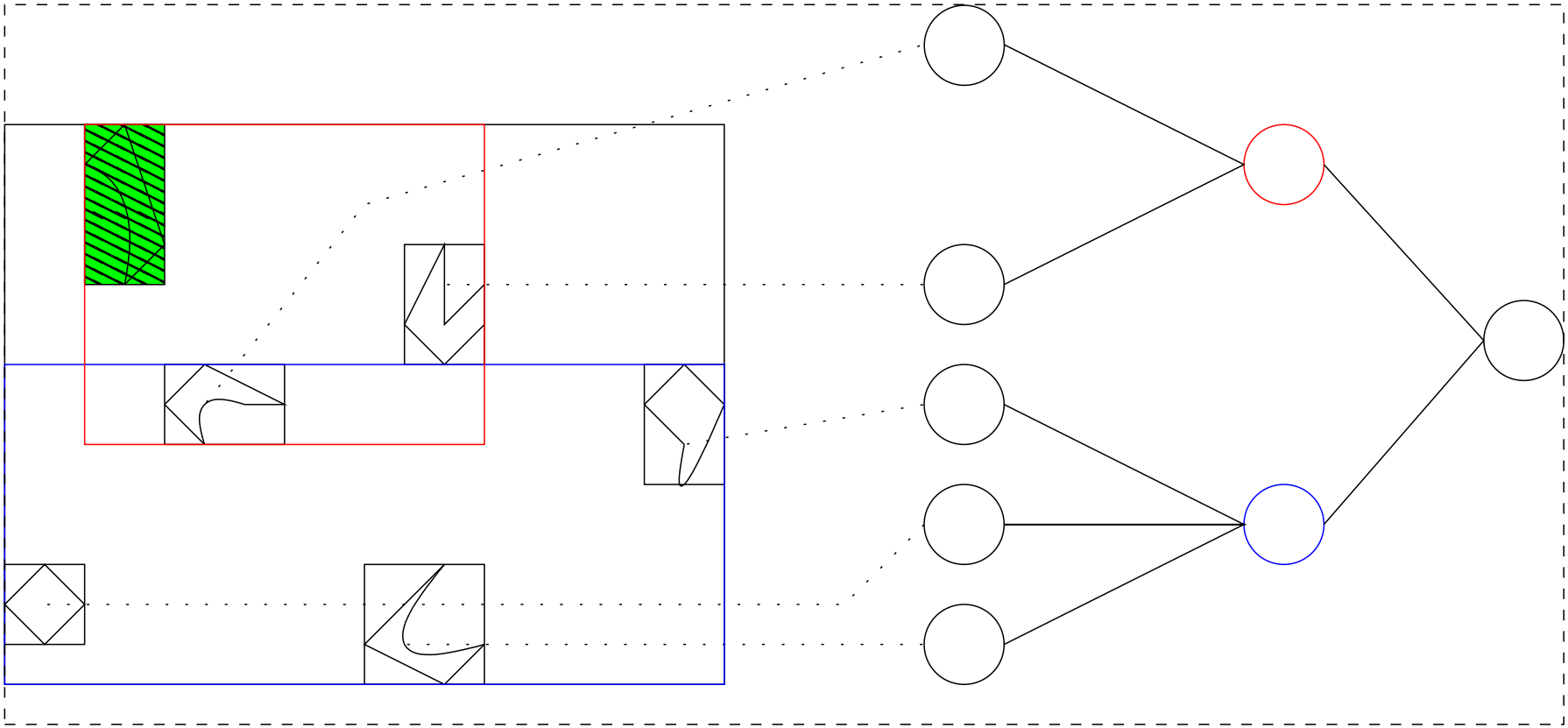
Original dynamic R-tree



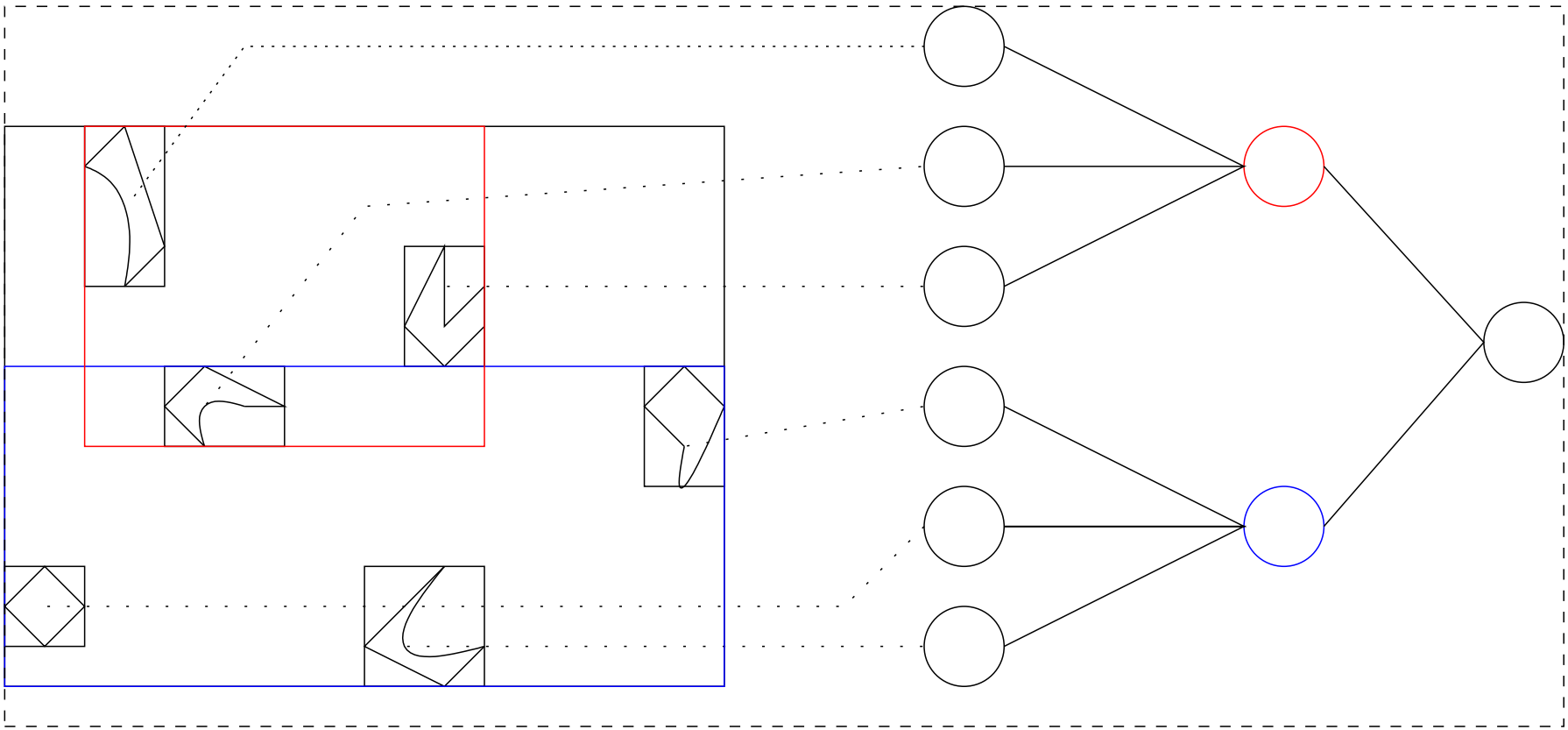
Original dynamic R-tree



Original dynamic R-tree



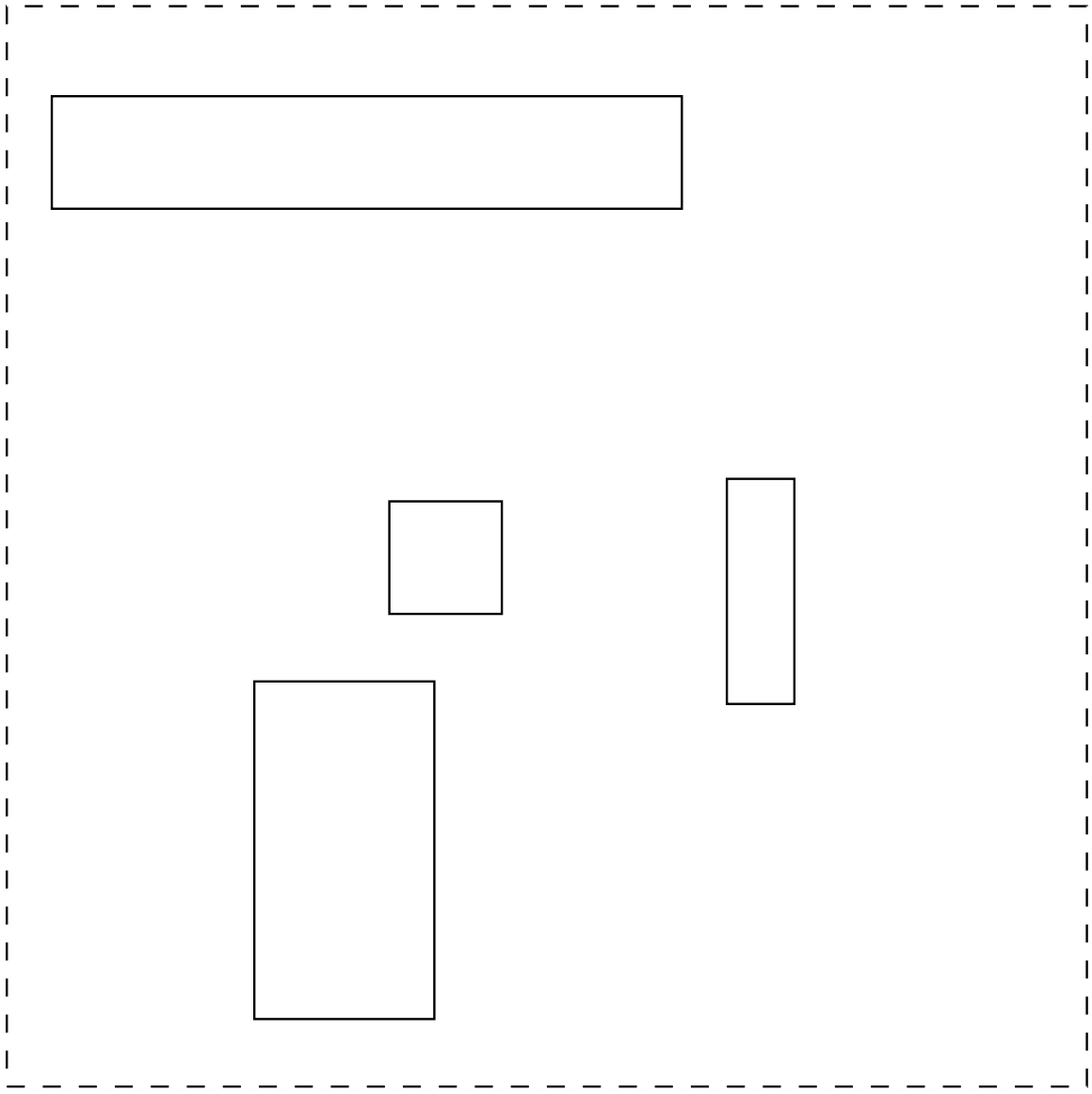
Original dynamic R-tree



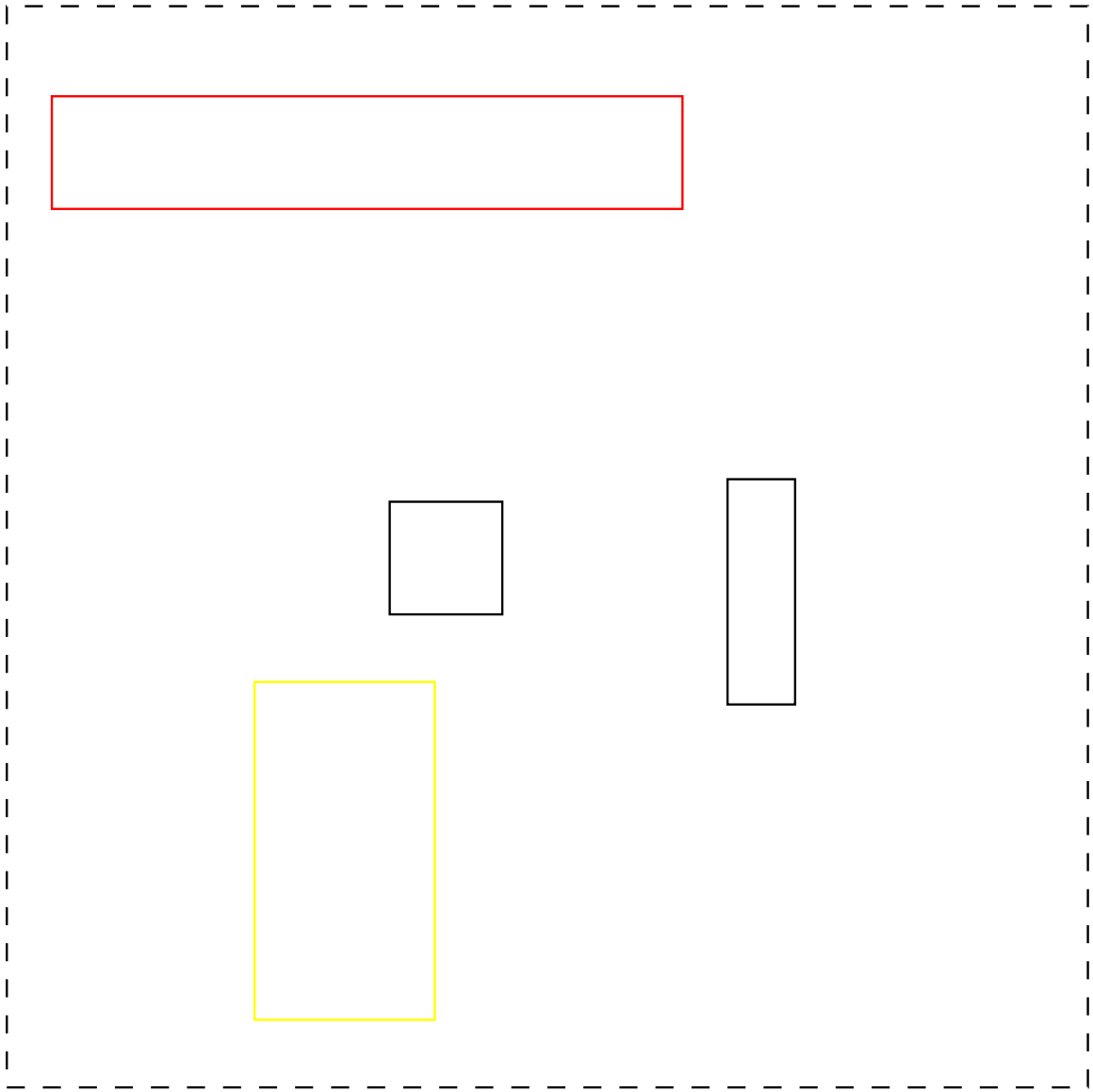
Original dynamic R-tree

Differences in overflow handling yield the *linear*, *quadratic* and the *polynomial* R-tree variants.
Consider the following example with $m = 2$ and $M = 3$.

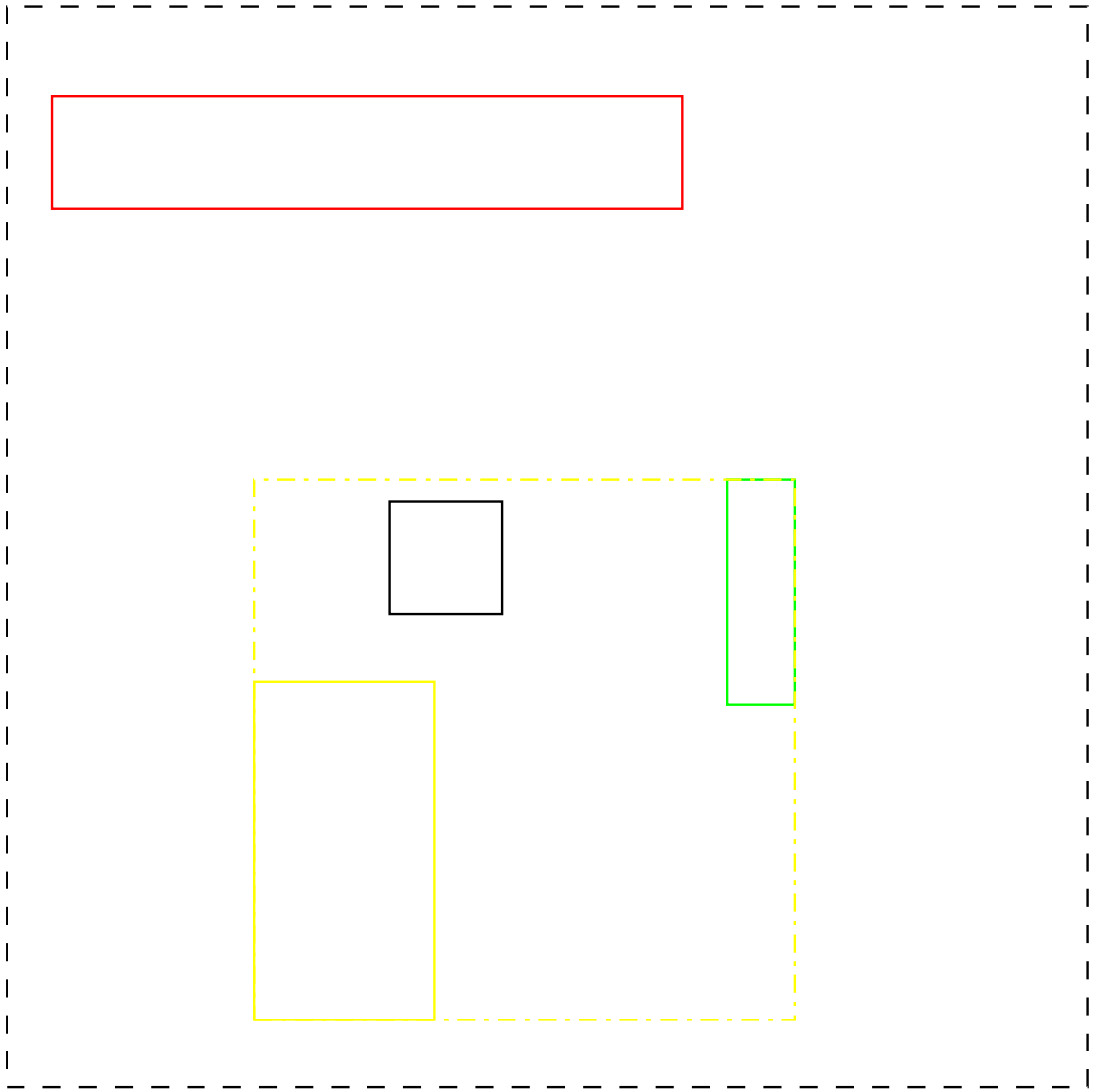
Overflow handling: linear splitting



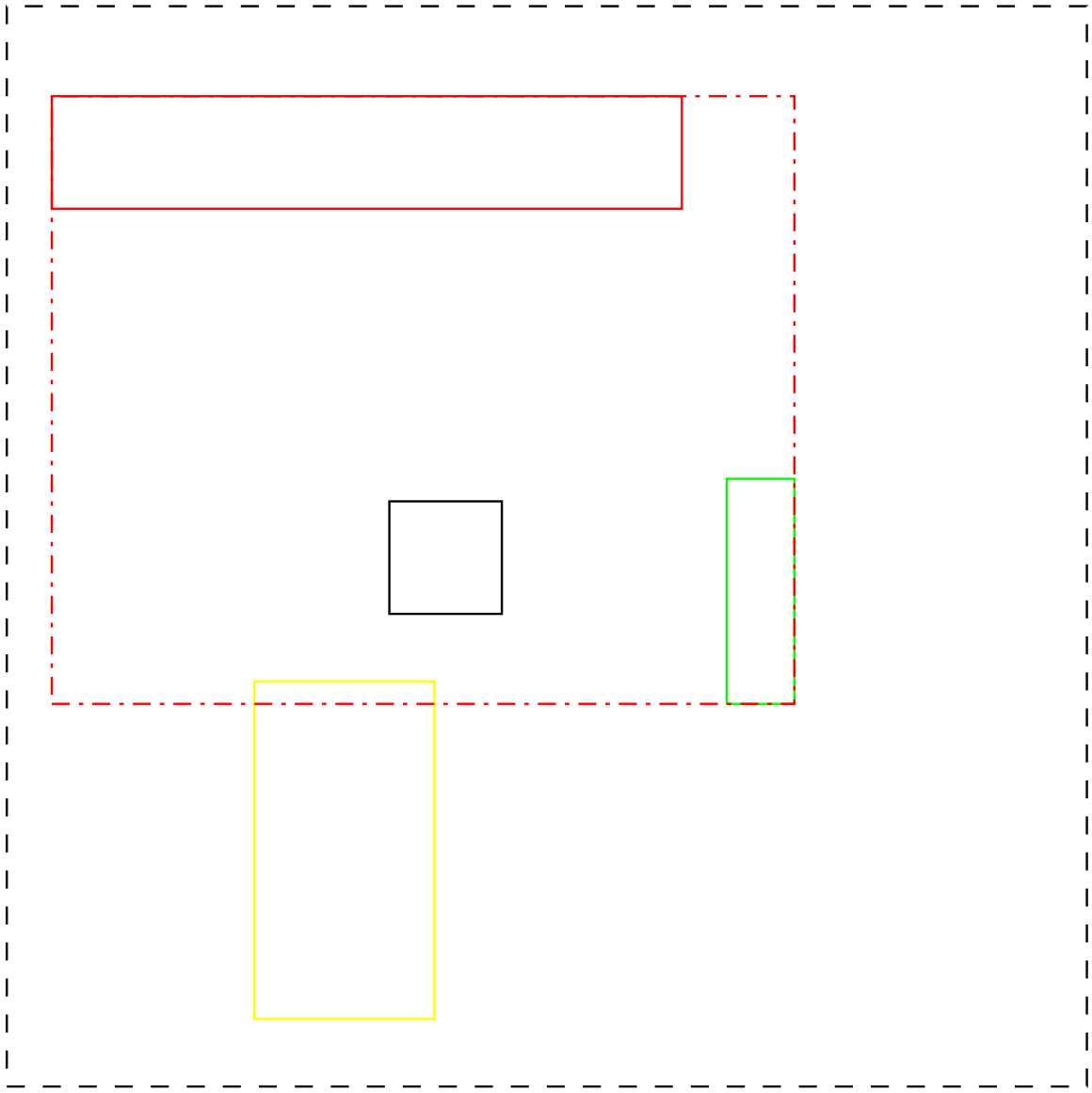
Overflow handling: linear splitting



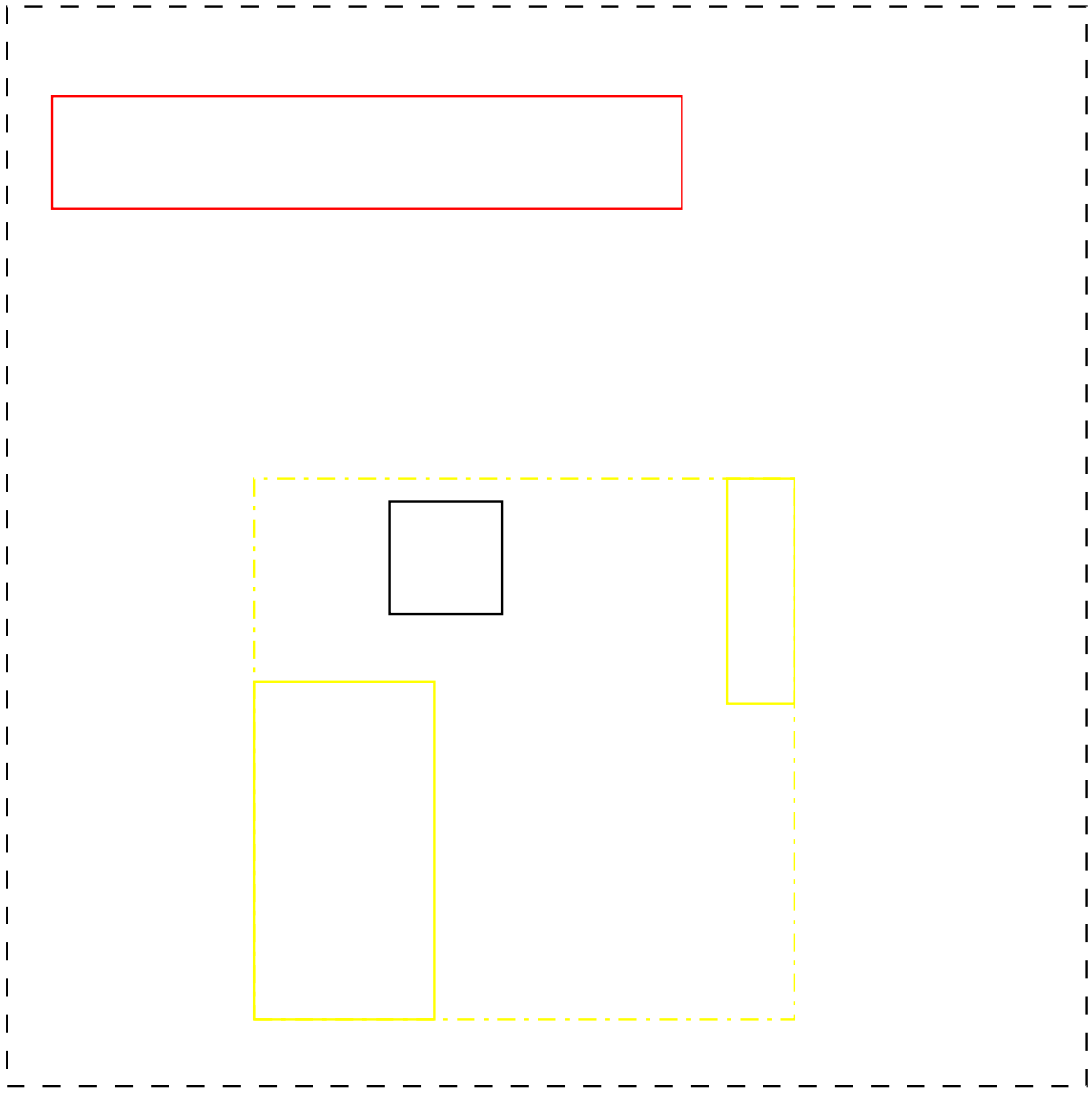
Overflow handling: linear splitting



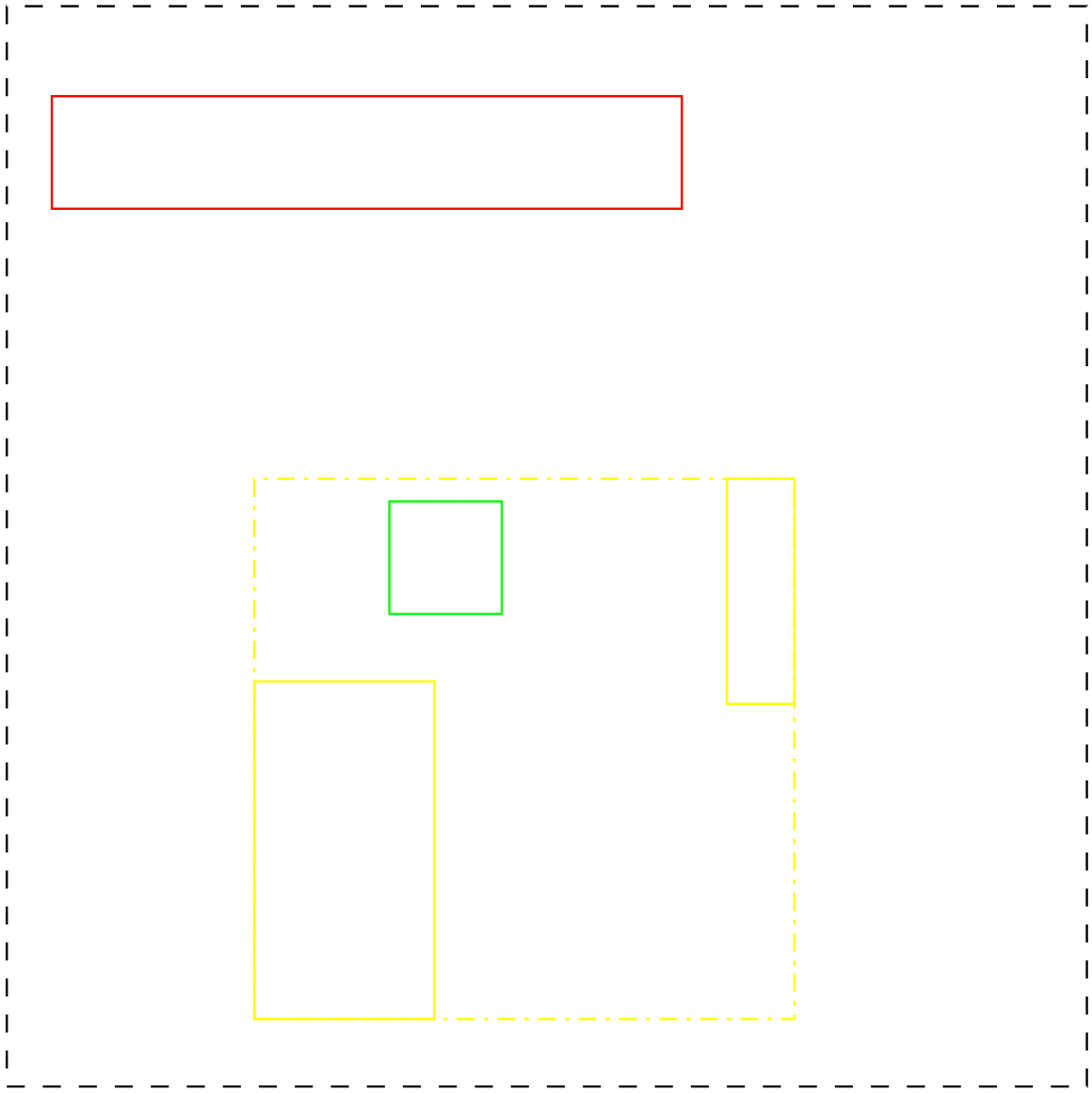
Overflow handling: linear splitting



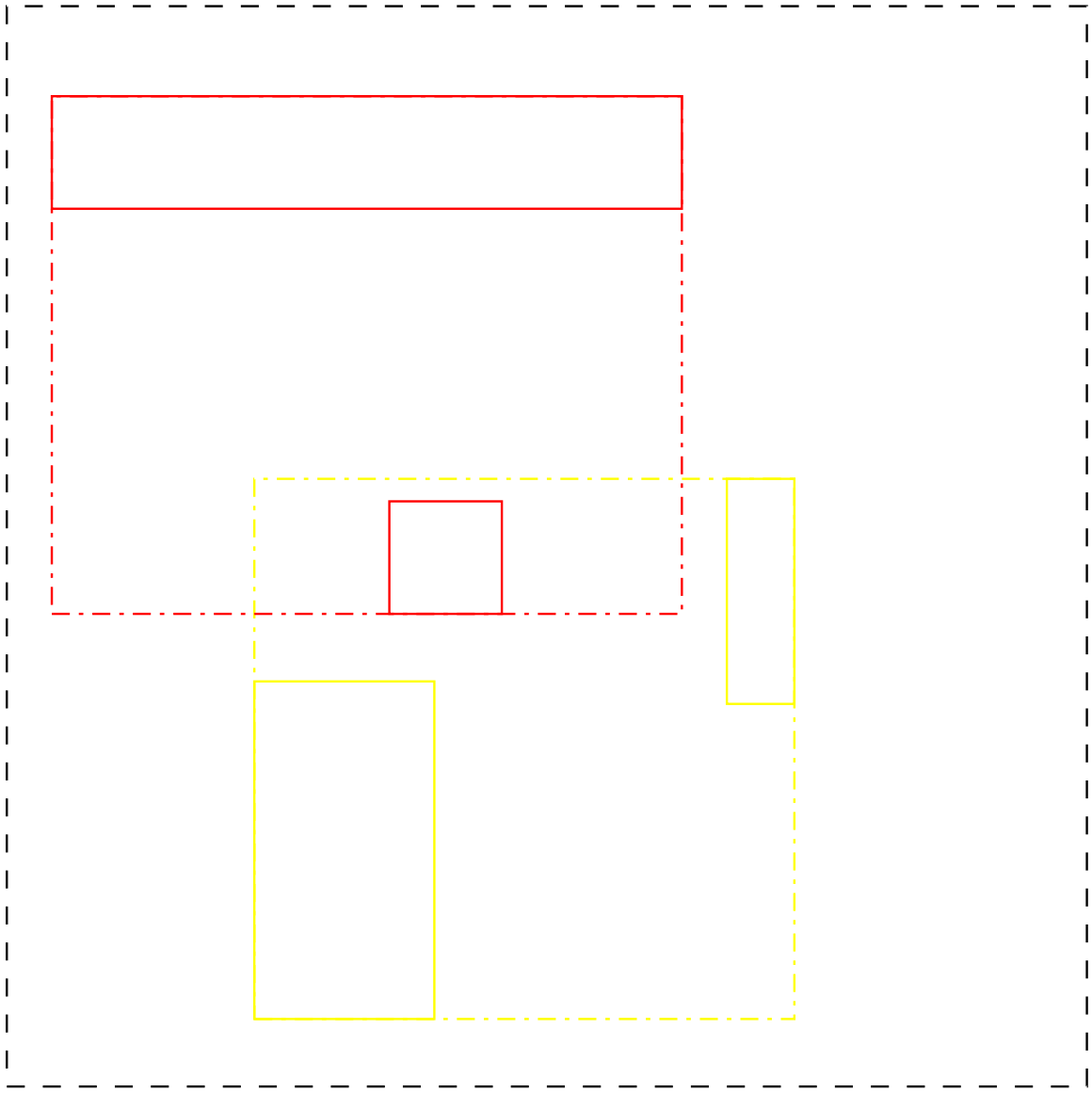
Overflow handling: linear splitting



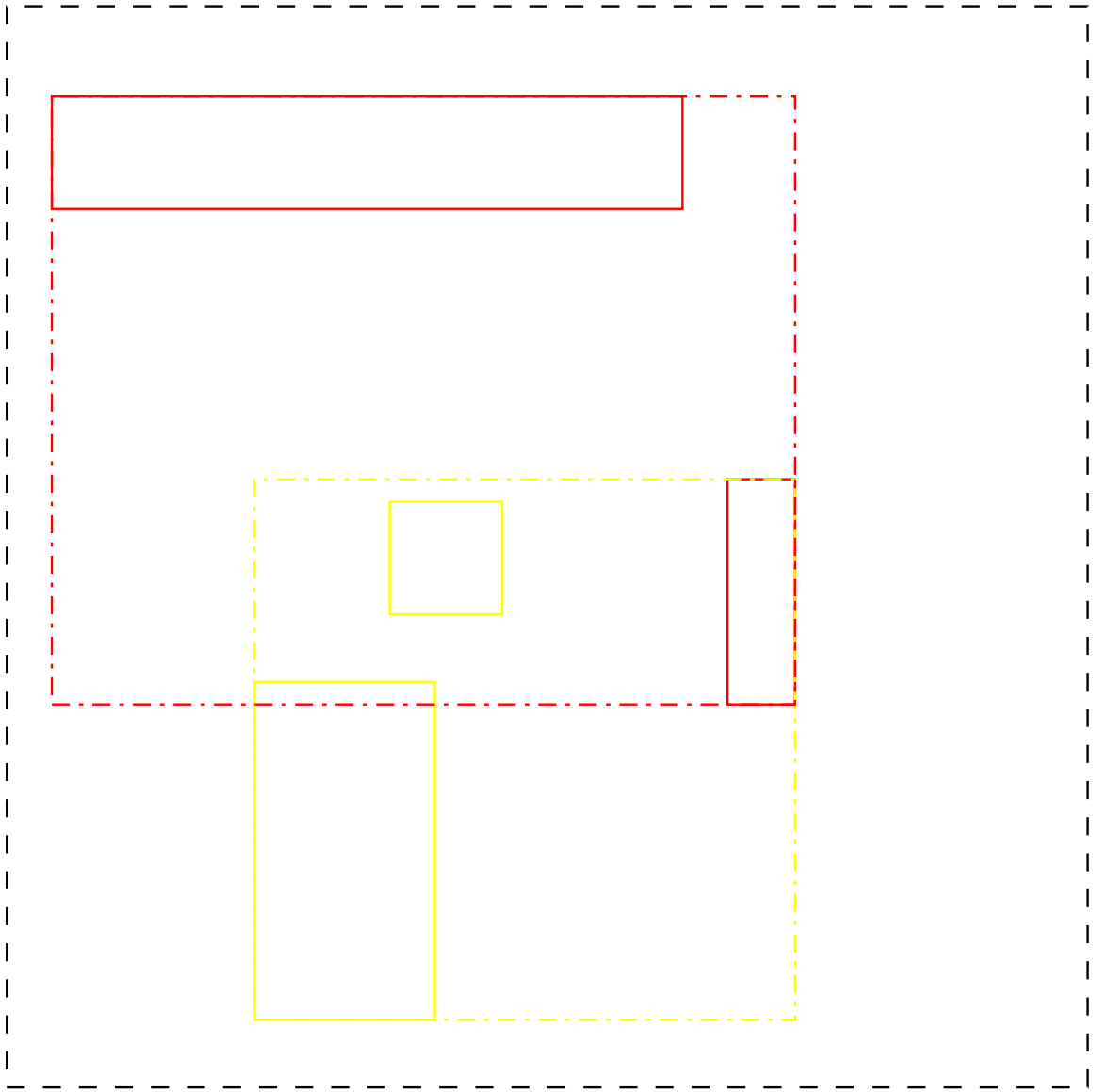
Overflow handling: linear splitting



Overflow handling: linear splitting



Overflow handling: linear splitting



Top-down Greedy Split (TGS)

Observation:

- Query efficiency is determined top-down by the shape of the bounding boxes

Why not build the R-tree top-down?

Top-down Greedy Split (TGS)

Uses an *collection* of orderings S.
Subdivide the input set into a maximum of M subsets each containing no more than $\tilde{n}$ elements:

- With respect to each ordering in S, subdivide the input set into groups of $\tilde{n}$ elements
- Find the best binary split with repect to all $s \in S$
- Recursively split both groups until all groups contain less than m elements
- So either we sort $|S|$ times to find this best binary split, or we duplicate the input set $|S|$ times

Top-down Greedy Split (TGS)

n=28, M=3, and so: h=4

Top-down Greedy Split (TGS)

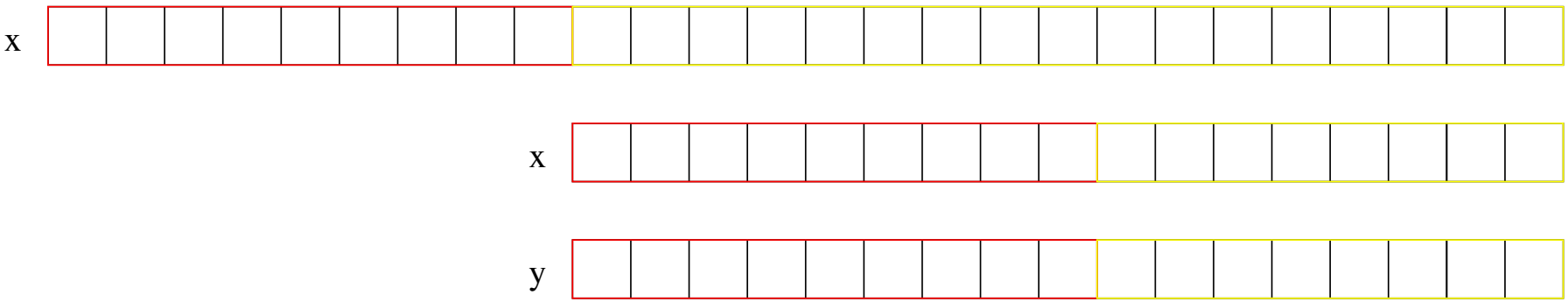
n=26, M=3, and so: h=3

at the first tree level, each subtree of height h−1=2 may contain 3^2=9 elements

Top-down Greedy Split (TGS)

n=26, M=3, and so: h=3

at the first tree level, each subtree of height h−1=2 may contain 3^2=9 elements



Top-down Greedy Split (TGS)

n=26, M=3, and so: h=3

at the first tree level, each subtree of height h−1=2 may contain 3^2=9 elements

At the second tree level, each subtree of height h−2=1 may contain 3 elements

Random TGS

For each of the K times divide some set in two, we sort $|S|$ times:

$$\begin{pmatrix} c_{11} & c_{21} & \cdots & c_{|S|1} \\ c_{12} & c_{22} & \cdots & c_{|S|2} \\ \vdots & & \ddots & \vdots \\ c_{1M} & c_{2M} & \cdots & c_{|S|M} \end{pmatrix}$$

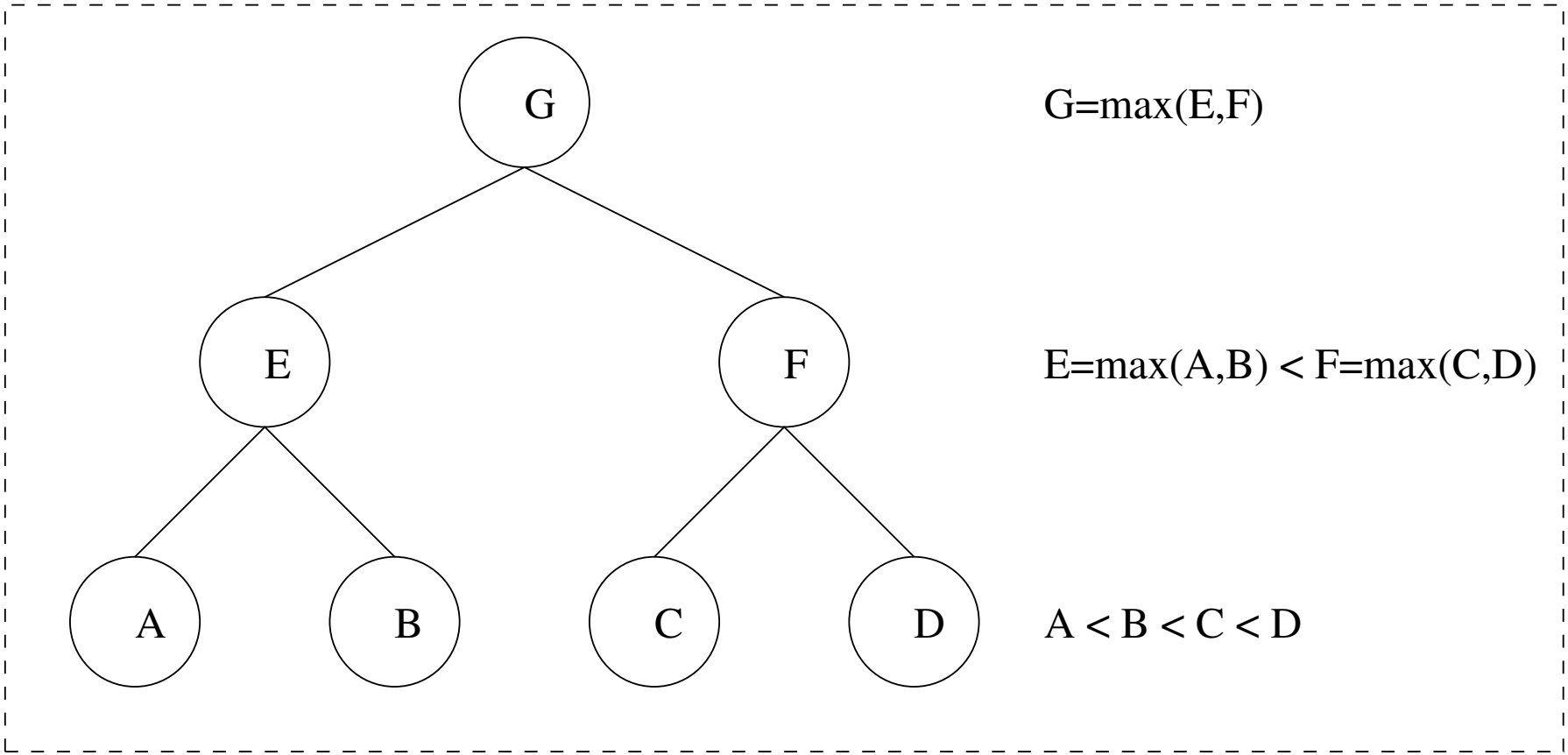
Ordering on MBRs

It would be useful to have an ordering on MBRs where:

$$X < Y < Z \tag{1}$$

Would imply that X is closer to Y than to Z.

Ordering on MBRs



Ordering on MBRs

Let us use the centre coordinate of MBRs for ordering.
This is trivial in one dimension;

$$x < y, \quad \text{with } x, y \in \mathbb{R}$$

is well-defined.

Ordering on MBRs

Let us use the centre coordinate of MBRs for ordering.
But for a higher number of dimensions $d \in \mathbb{N}, d > 1$:

$$\vec{x} < \vec{y}, \quad \text{with } \vec{x}, \vec{y} \in \mathbb{R}^d$$

is *not* well-defined.

Ordering on MBRs

To solve this, we find a mapping:

$$h : \mathbb{R}^d \rightarrow \mathbb{R}$$

by using the *Hilbert curve*.

Ordering on MBRs

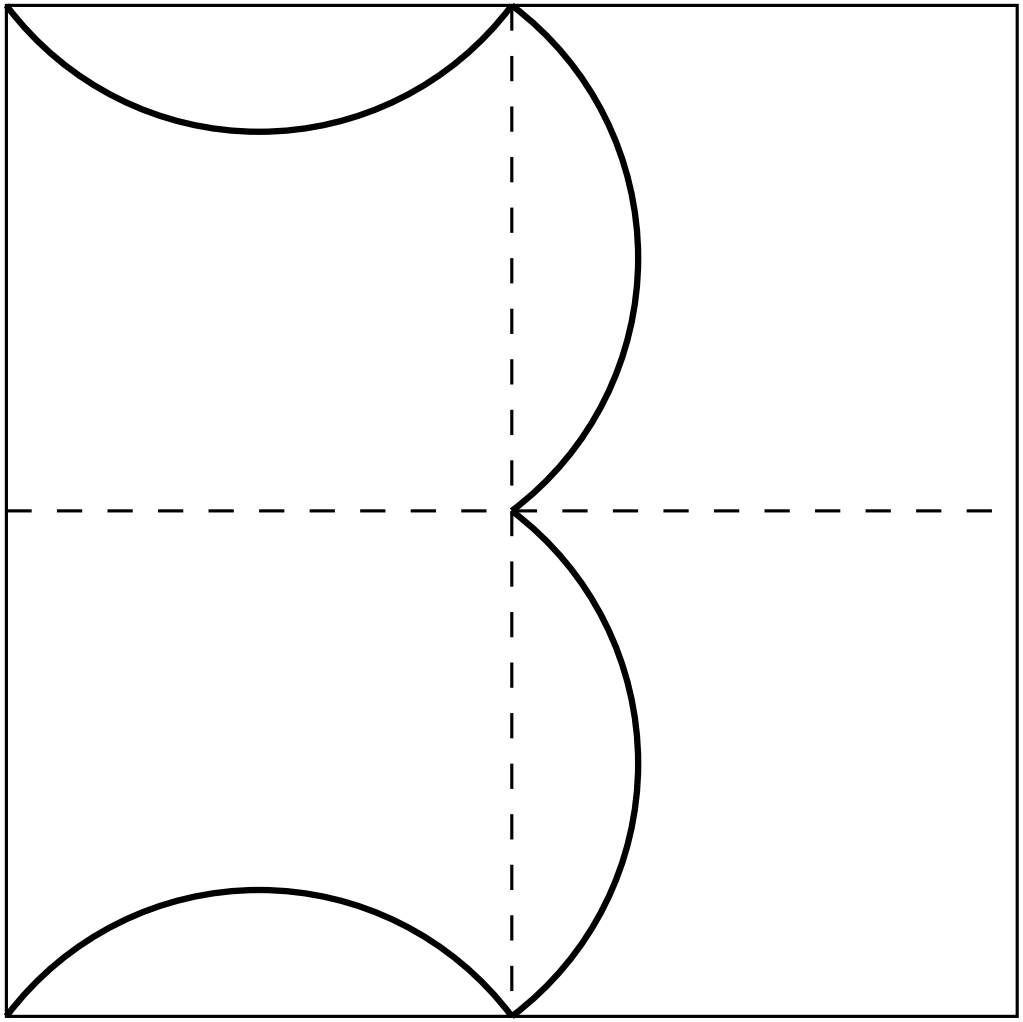


Figure: First-order Hilbert curve

Ordering on MBRs

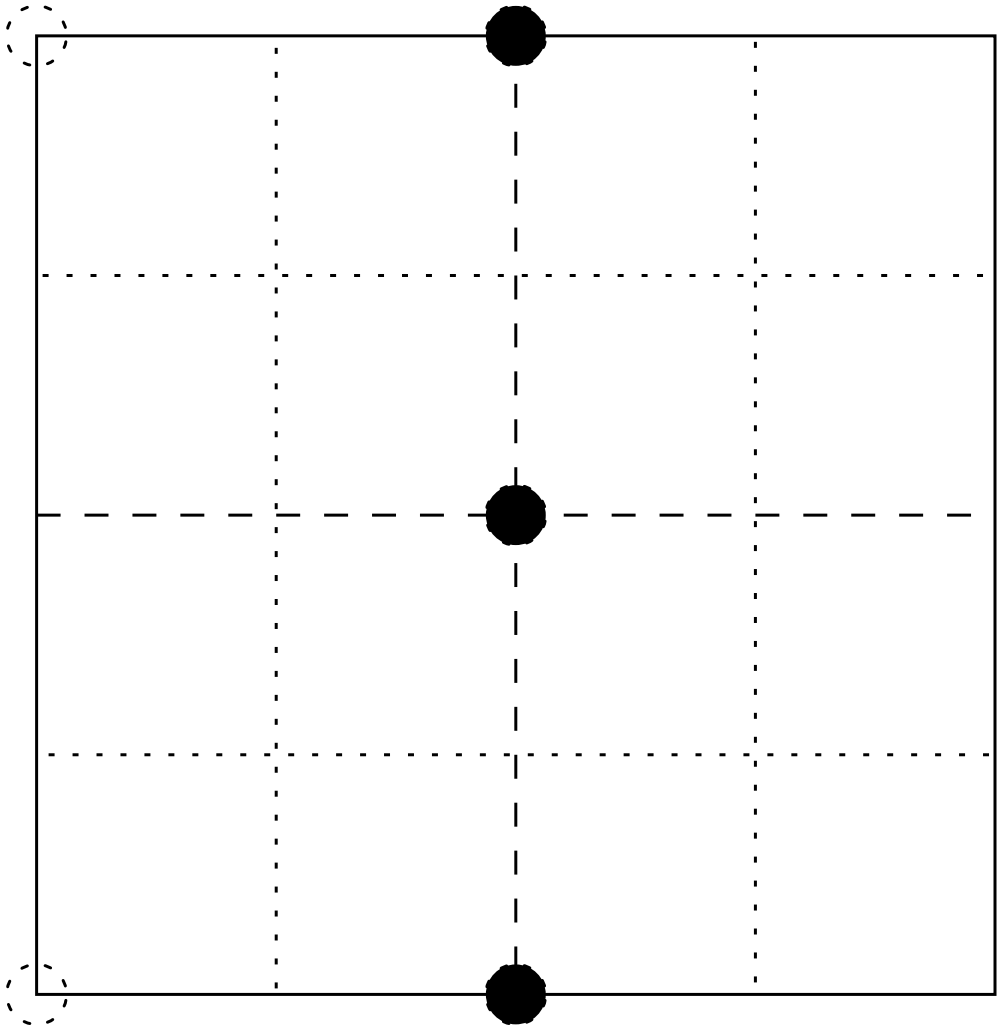


Figure: Recursion of the Hilbert curve

Ordering on MBRs

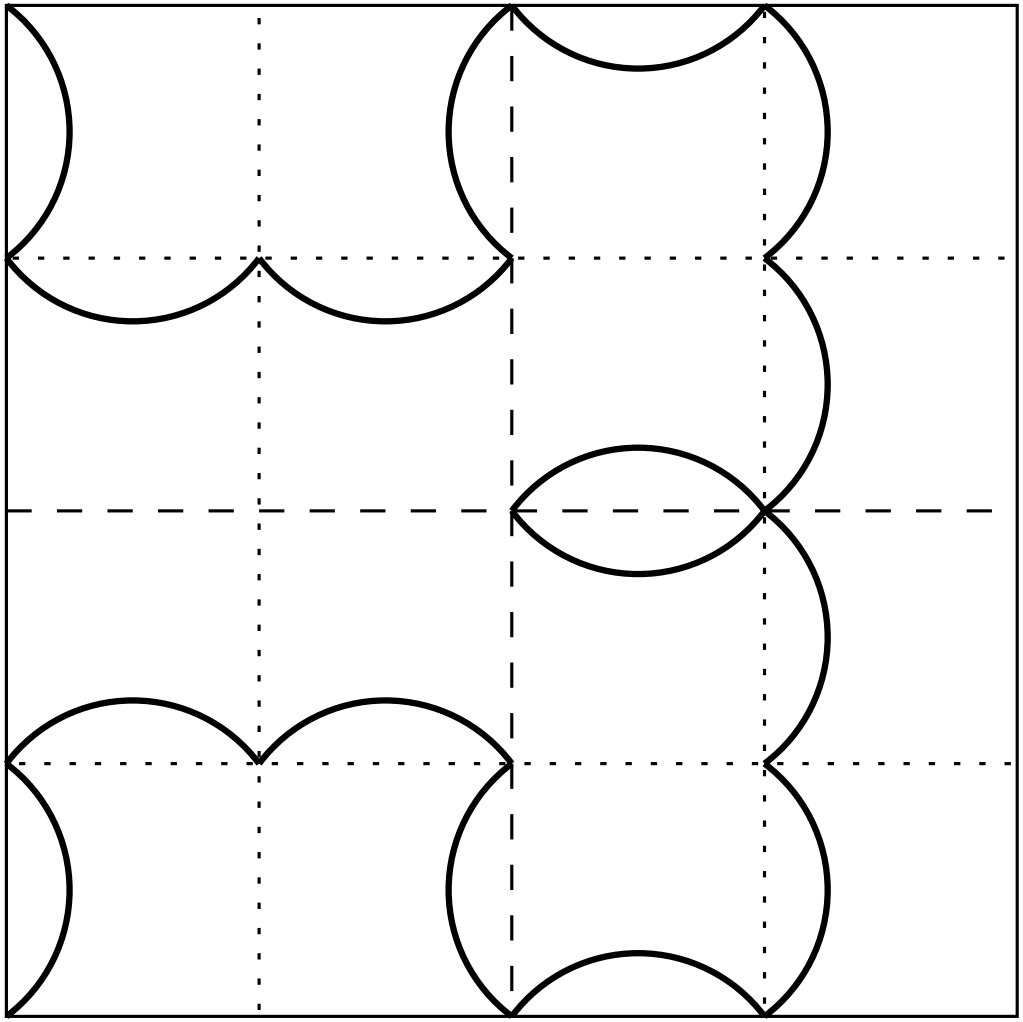


Figure: Second-order Hilbert curve

Ordering on MBRs

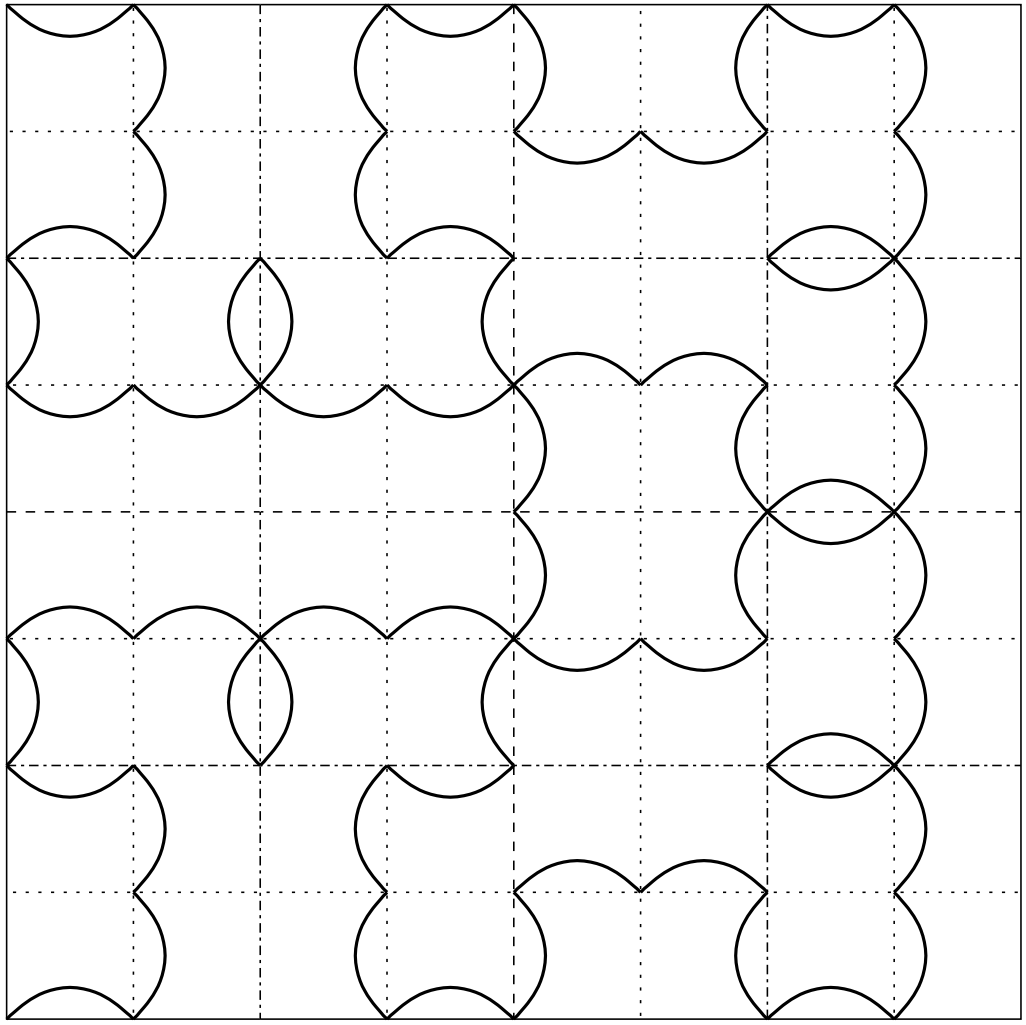


Figure: Third-order Hilbert curve

Ordering on MBRs

- Map points in $\mathbb{R}^d$ to a point $y \in \mathbb{R}^d$ on the nth order Hilbert curve
- Calculate the distance $d = \frac{i}{2^{dn}}$ of y on the nth order Hilbert curve
- Write $h_n(x) = d = \frac{i}{2^{dn}}$ for the nth order Hilbert coordinate transform of x

Ordering on MBRs

- Map points in $\mathbb{R}^d$ to a point $y \in \mathbb{R}^d$ on the nth order Hilbert curve
- Calculate the distance $d = \frac{i}{2^{dn}}$ of y on the nth order Hilbert curve
- Write $h_n(x) = d = \frac{i}{2^{dn}}$ for the nth order Hilbert coordinate transform of x

The true Hilbert coordinate transform h is then defined by:

$$h(x) = \lim_{n \to \infty} h_n(x), \quad x \in \mathbb{R}^d$$

Ordering on MBRs

In software calculation we use instead:

$$\tilde{h}(x) = \frac{c}{2^{dn}}$$

where c is the *cell number* containing x.

Hilbert R-tree

Invariants:

- Each leaf node stores the Hilbert coordinate of the centre coordinate of the MBR of the object stored there
- Each internal node stores the maximum Hilbert coordinate value h_{max} found at its children

Hilbert R-tree

Invariants:

- Each leaf node stores the Hilbert coordinate of the centre coordinate of the MBR of the object stored there
- Each internal node stores the maximum Hilbert coordinate value h_{max} found at its children

Insertion:

- Get the Hilbert coordinate h of the MBR of the new object to-be inserted
- Find the deepest internal node v with the smallest h_{max} larger than h and insert the new object there
- Update the h_{max} value at the v and all its parent nodes
- Check if v overflows

Hilbert R-tree: overflow handling

When a single internal node v overflows:

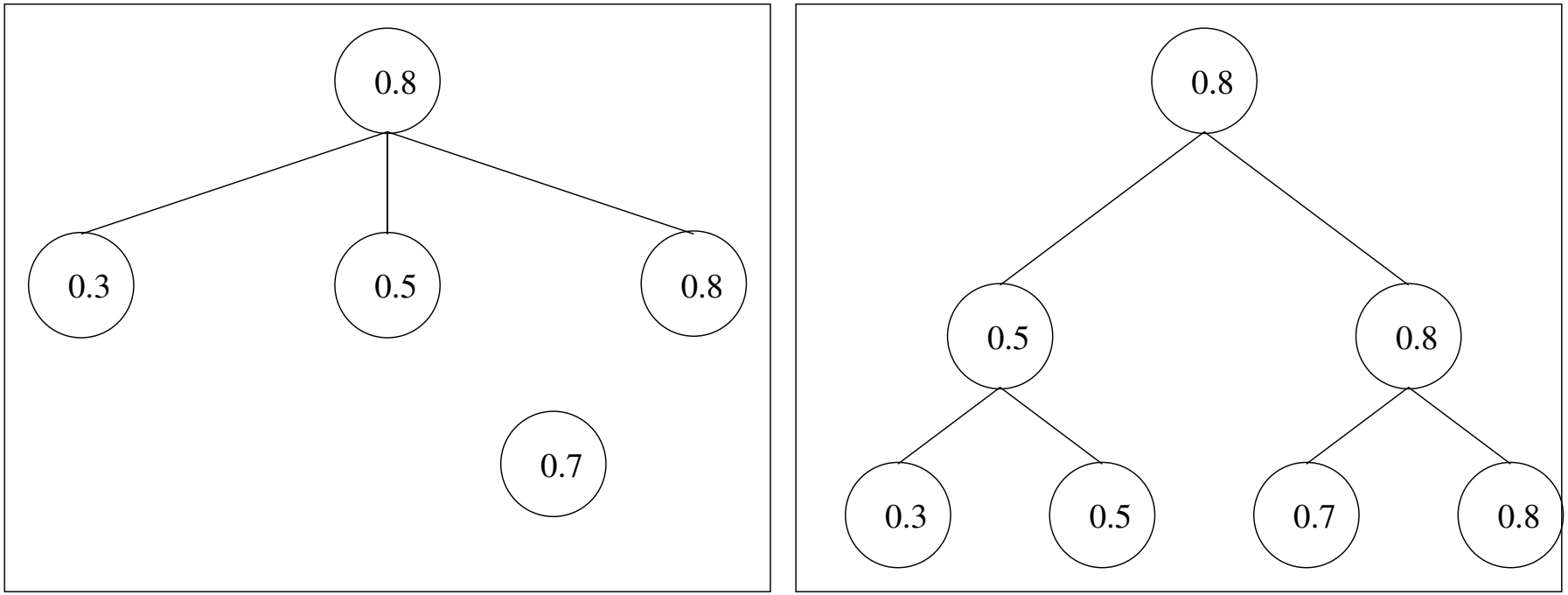


Figure: Overflow handling when there are no neighbour nodes

Hilbert R-tree: overflow handling

When a single internal node v overflows:

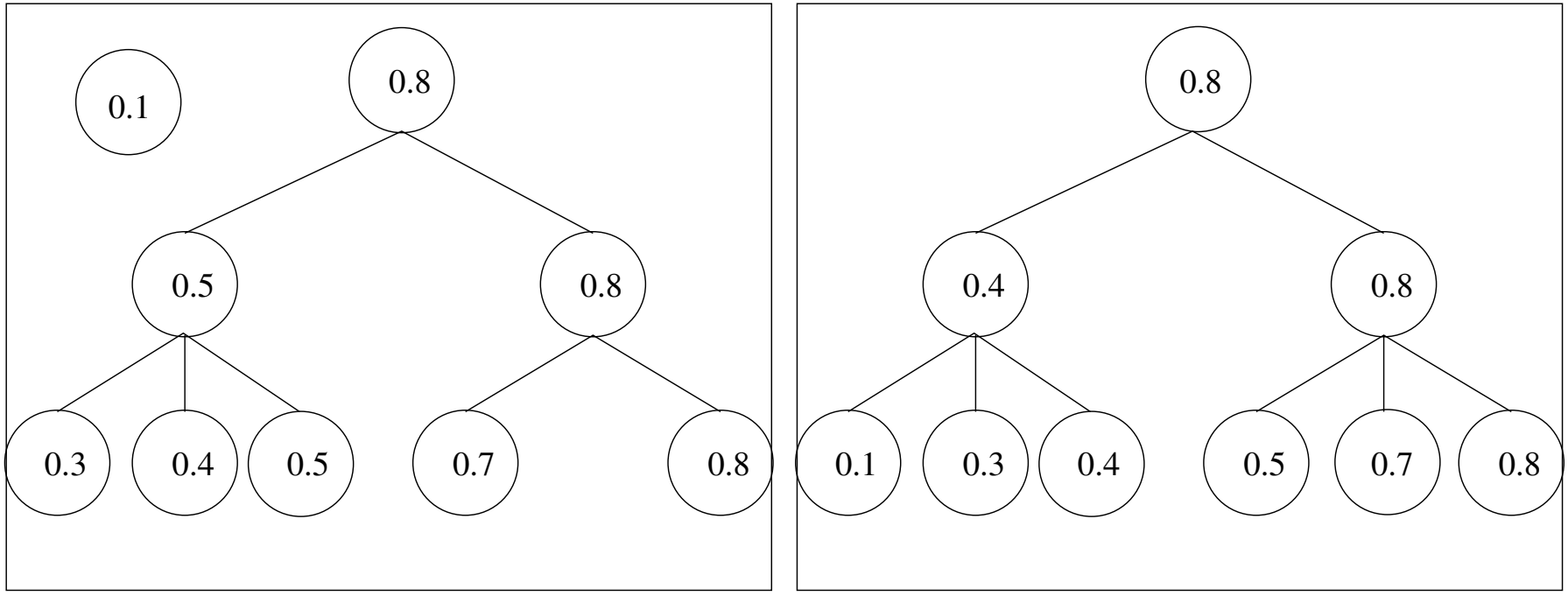


Figure: Overflow handling when there is a non-full neighbour

Hilbert Top-down Greedy Split

Like normal TGS, but with S containing only the Hilbert coordinate based ordering.

Outline

R-trees

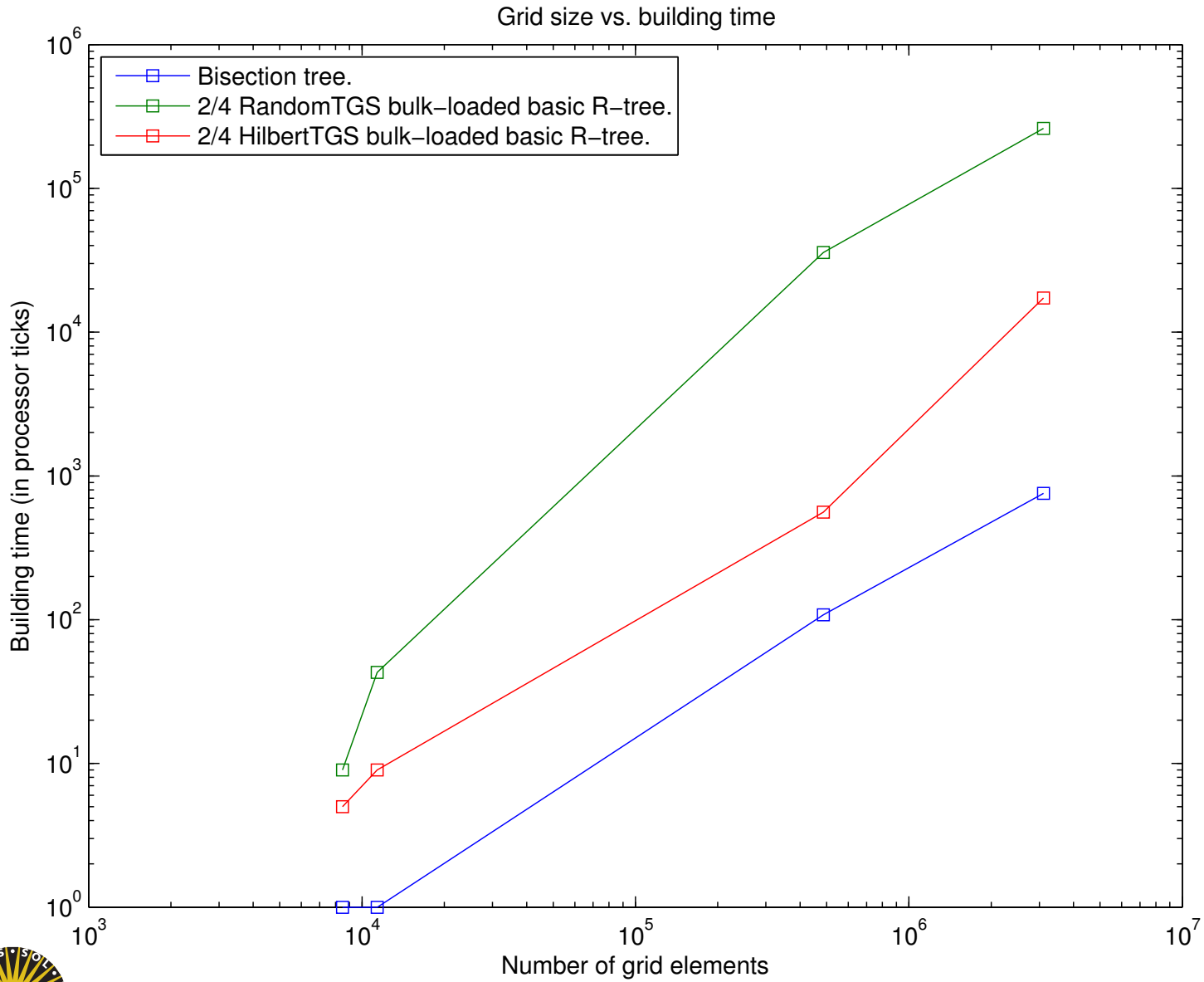
Experiments

Some variations have been implemented in C++. The resulting library recently went public as an open-source project:

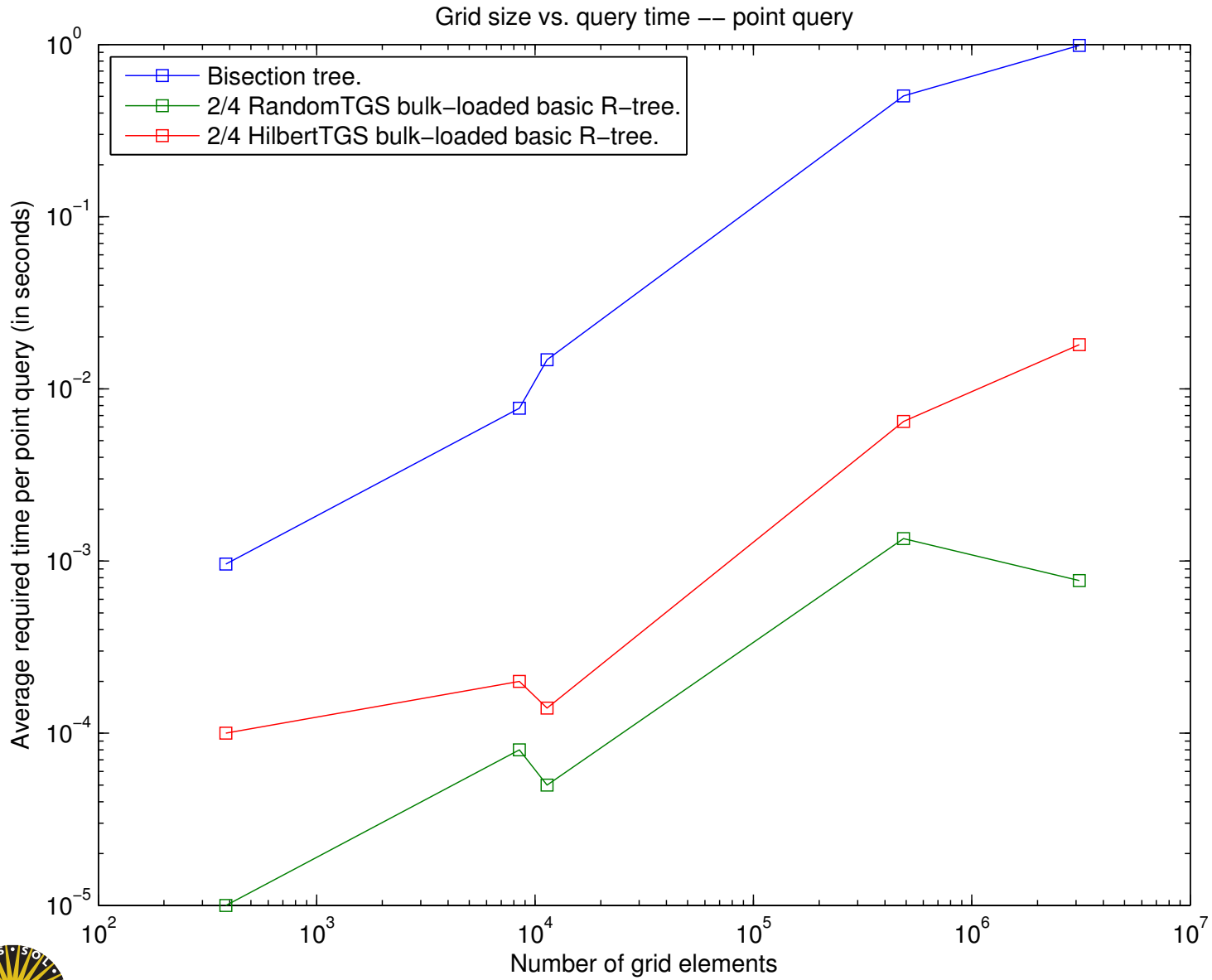
`http://www.sourceforge.net/projects/rtree-lib`

Applied to datasets supplied by Shell we obtained the following experimental results.

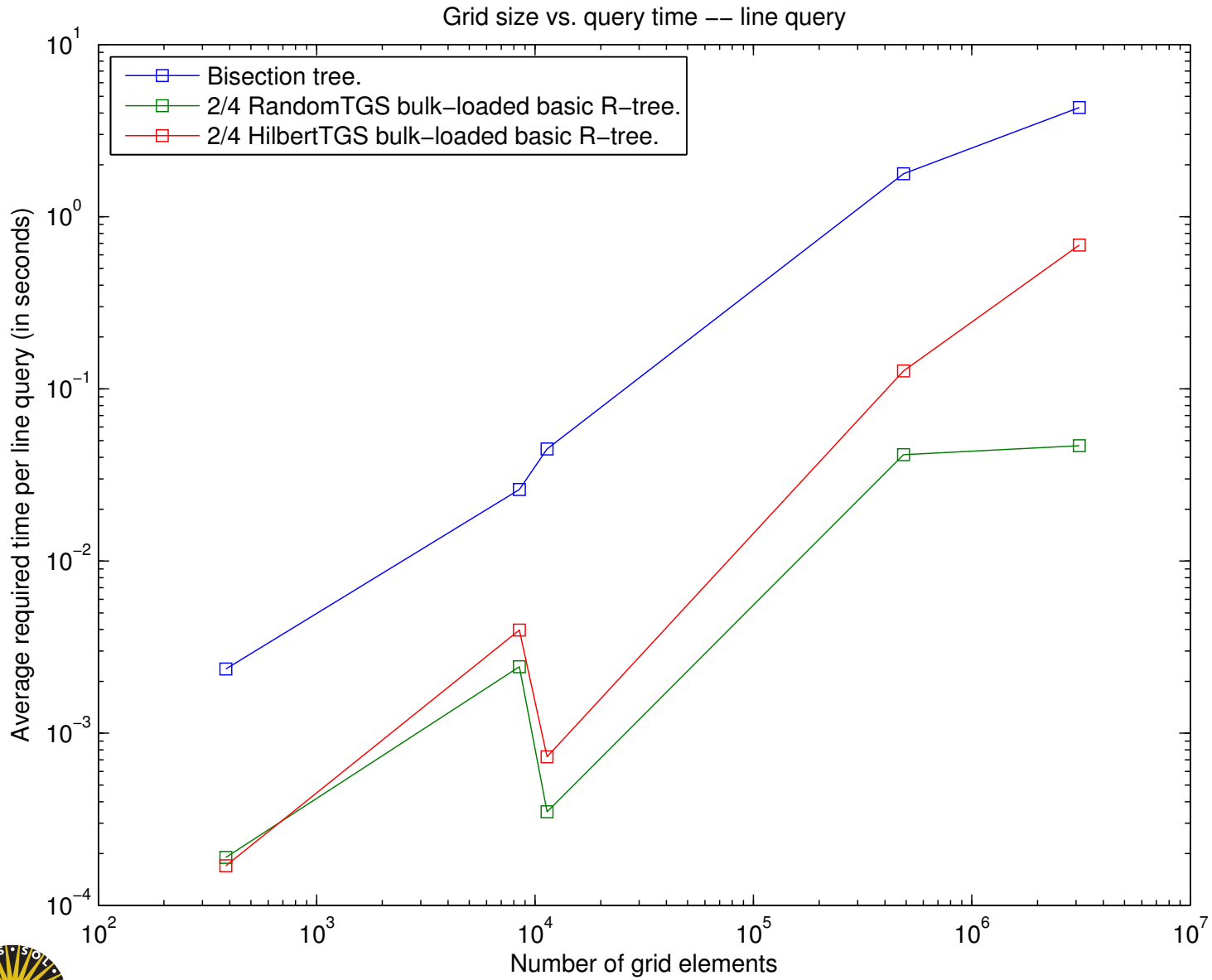
Experiments: construction time



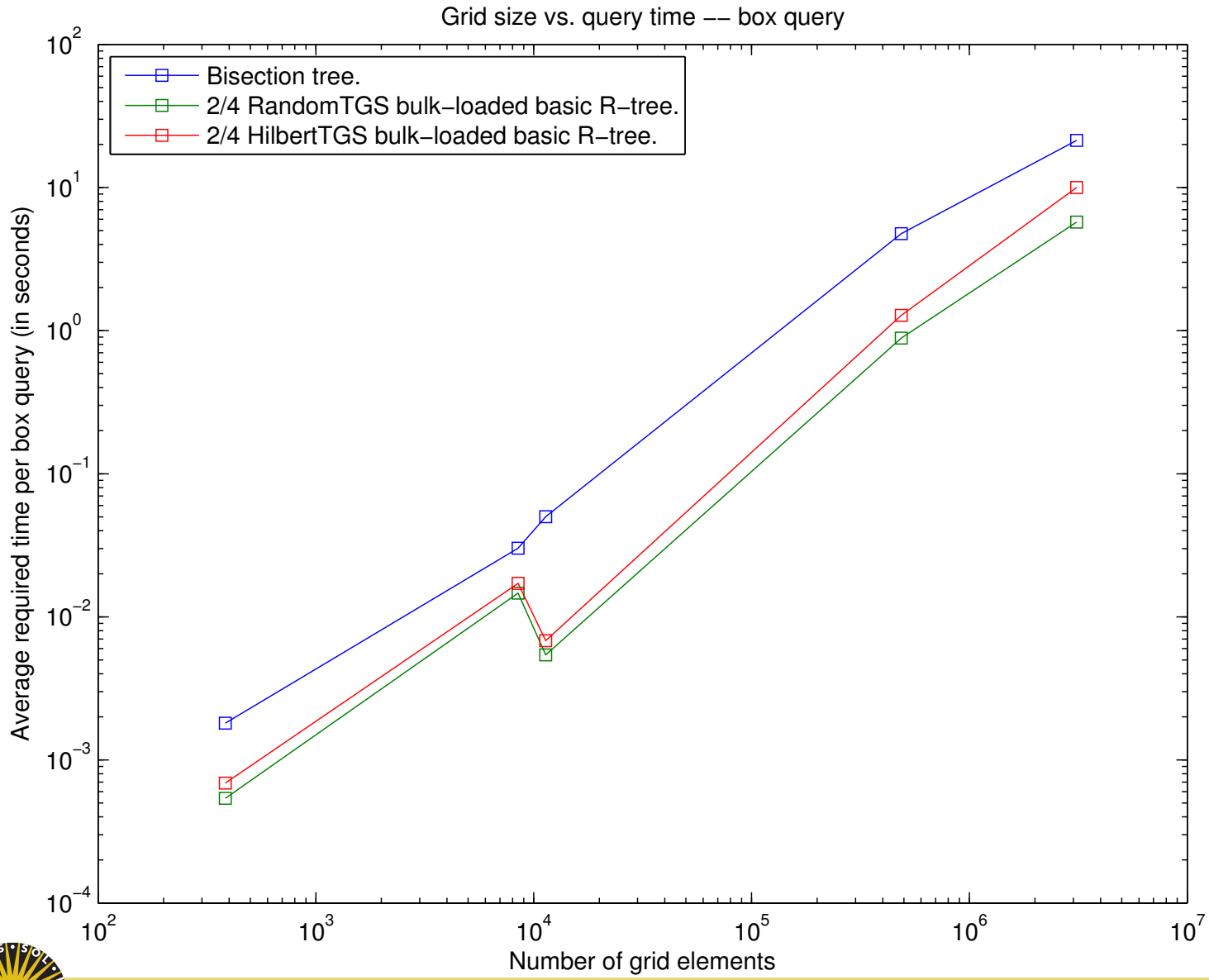
Experiments: point query time



Experiments: line query time



Experiments: box query time



Other query types:

- k-nn query (implemented)
- Hyperplane query (not implemented)

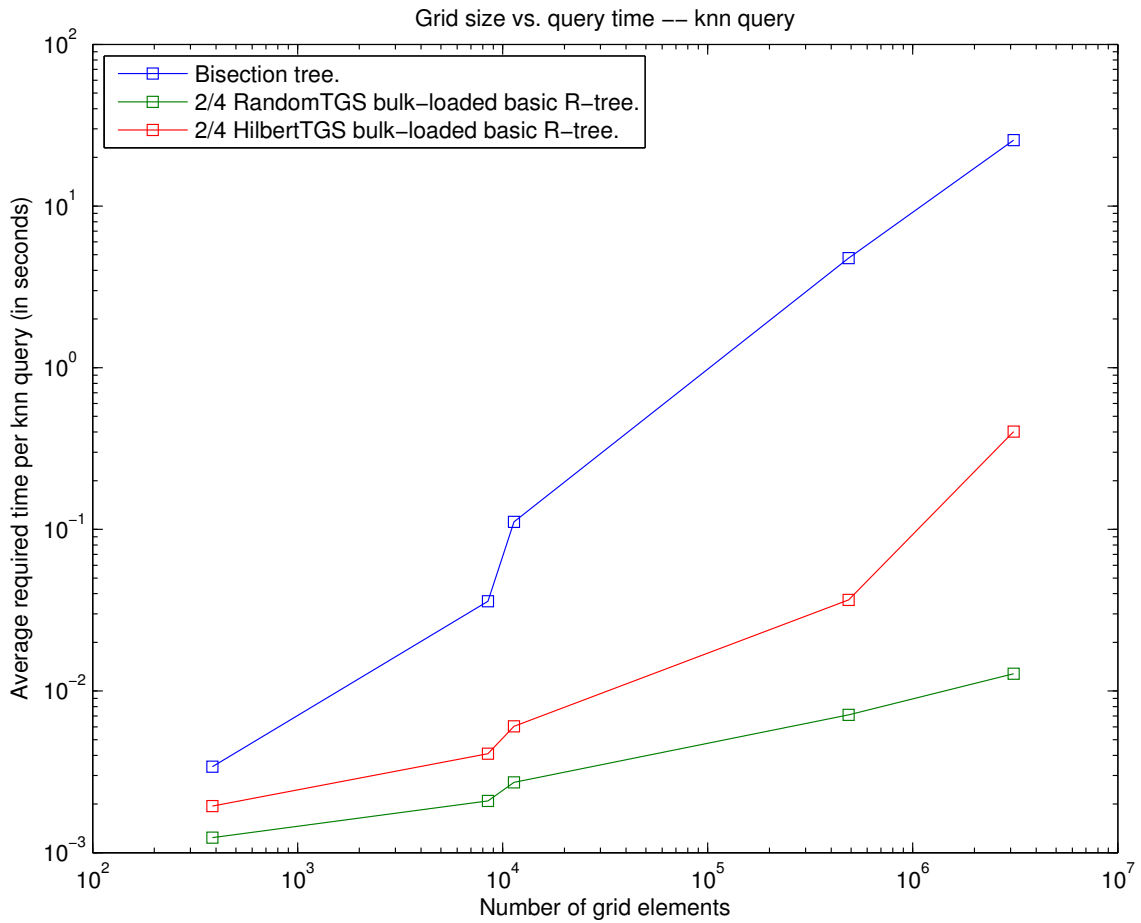


Figure: $k$nn query time

Possible other application areas:

- Databases
- Graphics/Gaming
- Chip Design
- Navigation Systems
- Image Processing
- ...?

Questions?